Exhibit A
(Part 1 of 2)

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

Red Hat, Inc.,	
Plaintiff,	
v.	Case No. 6:09-CV-549-LED
Bedrock Computer Technologies LLC,	JURY TRIAL DEMANDED
Defendant.	

JOINT INVALIDITY CONTENTIONS AND PRODUCTION OF DOCUMENTS PURSUANT TO PATENT RULES 3-3 AND 3-4(b)

Pursuant to the Rules of Practice for Patent Cases for the Eastern District of Texas ("Patent Rules" or "P.R."), Plaintiff Red Hat, Inc.[1] and Cross-claim defendants NYSE Euronext, Rackspace Hosting, Inc., ThePlanet.com Internet Services, Inc., Whole Foods Market, Inc., 1&1 Internet, Inc., ConocoPhillips Co., ConAgra Foods, Inc., Facebook, Inc., Go Daddy Group, Inc., Nationwide Mutual Insurance Co., R.L. Polk & Co., SunGard Data Systems, Inc., The Gap Inc., and Virgin America Inc. (collectively, the "Defendants"), hereby disclose their Invalidity Contentions. The Defendants contend that each of the claims asserted by Bedrock Computer Technologies LLC ("Bedrock") is invalid under at least 35 U.S.C. §§ 102, 103, and/or 112.

[1] All of the references in Red Hat, Inc.'s May 14, 2010 Invalidity Contentions are included in this Joint Invalidity Contentions and Production of Documents. Given the July 6, 2010 Docket Control Order (dkt. 170) changing the deadline for serving invalidity contentions, Red Hat, Inc. does not need to seek leave to supplement its May 14, 2010 Invalidity Contentions.

I. GENERAL STATEMENTS AND OBJECTIONS

A. Asserted Claims

Bedrock has served the Defendants with Infringement Contentions alleging infringement of U.S. Patent No. 5,893,120 ("the '120 patent"). Specifically, Bedrock has alleged that the Defendants infringe claims 1 – 8 of the '120 patent (collectively, the "Asserted Claims").

B. Invalidity Contentions

Defendants reserve all rights to modify, amend, and/or supplement their Invalidity Contentions, including in accordance with P.R. 3-6 and 3-7 following the Court's claim construction ruling or upon any alteration/clarification by Bedrock of its asserted claim construction to the extent permitted by this Court.

C. Claim Construction

The Court has not yet construed the Asserted Claims. The Defendants reserve the right to identify other art or to supplement their disclosures or contentions because the Defendants' positions on the invalidity of particular claims will depend on how those claims are construed by the Court. The Defendants' Invalidity Contentions are based, at least in part, on their present understanding of the Asserted Claims and/or their present understanding of the claim constructions Bedrock appears to be asserting—based on Bedrock's Infringement Contentions—whether or not the Defendants agree with such claim constructions.

To the extent that these Invalidity Contentions reflect constructions of claim terms that may be consistent with or implicit in Bedrock's Infringement Contentions, no inference is intended or should be drawn that the Defendants agree with such claim constructions. The Defendants take no position on any matter of claim construction in these invalidity contentions. Any statement herein describing or tending to describe any claim element is provided solely for the purpose of understanding the relevant prior art. The Defendants expressly reserve the right to propose any claim construction they consider appropriate and/or to contest any claim construction they consider inappropriate.

In part because of the uncertainty of claim construction, the Defendants' Invalidity Contentions are sometimes made in the alternative and are not necessarily intended to be consistent with each other, and should be viewed accordingly. Further, by including in this disclosure prior art that would be anticipatory or render a claim obvious based on a particular scope or construction of the claims, including that apparently applied by Bedrock in its Infringement Contentions, the Defendants' Invalidity Contentions herein are not, and should in no way be seen as adoptions or admissions as to the accuracy of such scope or construction.

The Defendants reserve all rights to further supplement or modify the positions and information in these Invalidity Contentions, including without limitation, the prior art and grounds of invalidity set forth herein, after the Court has construed the asserted claims in accordance with the Patent Rules and/or the Court's Orders.

D. Ongoing Discovery and Disclosures

Discovery in this case is in its early stages and the Defendants' investigation, including the Defendants' search for prior art, is ongoing. The Defendants therefore reserve the right to further supplement or alter the positions taken and information disclosed in these Invalidity Contentions including, without limitation, the prior art and grounds of invalidity set forth herein, to take into account information or defenses that may come to light as a result of these continuing efforts. The Defendants hereby incorporate by reference the testimony of any fact witnesses that are deposed, that provide declarations, or that otherwise testify in this lawsuit. The Defendants also hereby incorporate by reference the reports and testimony of the Defendants' expert witnesses regarding invalidity of the patent.

The Defendants understand and are relying upon the fact that the date to which Bedrock may be entitled to as the earliest priority date of the ‘120 patent is its earliest filing date. The Defendants intend to diligently seek discovery to establish conception and reduction to practice dates, as appropriate, to demonstrate earlier invention by other parties under 35 U.S.C. § 102(g). The Defendants further intend to take discovery on the issues of improper inventorship and/or derivation under 35 U.S.C. § 102(f), public use and/or the on-sale bar under 35 U.S.C. § 102(b), and/or applicant’s failure to comply with 35 U.S.C. § 112. The Defendants therefore reserve all rights to further supplement or amend these invalidity contentions if and when further information becomes available.

E. Prior Art Identification and Citation

Pursuant to Patent Rule 3-3(a), the Defendants identify specific portions of prior art references that disclose the elements of the Asserted Claims. Although the Defendants have identified at least one citation per element for each reference, each and every disclosure of the same element in said reference is not necessarily identified. The Defendants identify only limited portions of the cited references as examples. It should be recognized that a person of ordinary skill in the art would generally read a prior art reference as a whole and in the context of other publications, literature, and general knowledge in the field. To understand and interpret any specific statement or disclosure in a prior art reference, a person of ordinary skill in the art would rely upon other information including other publications and general scientific or engineering knowledge. The Defendants therefore reserve the right to rely upon other unidentified portions of the prior art references and on other publications and expert testimony to provide context and to aid understanding and interpretation of the identified portions. The Defendants also reserve the right to rely upon other portions of the prior art references, other publications, and the testimony of experts to establish that the alleged inventions would have been obvious to a person of ordinary skill in the art, including on the basis of modifying or combining certain cited references. The Defendants also reserve the right to rely upon any admissions relating to prior art in the Asserted Patent or its respective prosecution history.

Where the Defendants identify a particular figure in a prior art reference, the identification should be understood to encompass the caption and description of the figure as well as any text relating to the figure in addition to the figure itself. Similarly, where an identified portion of text refers to a figure or other material, the identification should be understood to include the referenced figure or other material as well.

F. Reservation of Rights

The Defendants reserve all rights to further supplement or modify these Invalidity Contentions, including the prior art disclosed and stated grounds of invalidity, pursuant to the District's Patent Rules. In addition, the Defendants reserve the right to prove the invalidity of the asserted claims on bases other than those required to be disclosed in these disclosures and contentions pursuant to P.R. 3-3. For example, the Defendants further contend that each of the claims of the '120 patent is drawn to subject matter that is not patentable under 35 U.S.C. § 101. For example, the Asserted Claims are drawn to an "abstract idea" and are not patentable as explained by the Supreme Court in *Bilski v. Kappos*. 130 S.Ct. 3218, 3229-30 (2010).

II. <u>INVALIDITY CONTENTIONS PURSUANT TO P.R. 3-3</u>

A. Contentions Under P.R. 3-3(a)-(c)

Each of the Asserted Claims is anticipated and/or rendered obvious by prior art. Pursuant to P.R. 3-3(a) the Defendants identify the prior art that anticipates or renders an Asserted Claim obvious in Exhibits A - E which are hereby incorporated by reference as if fully set forth herein. On information and belief, each listed document or item became prior art at least as early as the dates given. Each of the foregoing prior art references identified in Exhibit A includes a chart in at least one of Exhibits B - D specifically identifying where each element of each asserted claim is found in the prior art pursuant to P.R. 3-3(c) including, for claims governed by 35 U.S.C. §112(6), the identity of structure(s), act(s), or materials(s) in each item of prior art that performs the claimed function.

To the extent any limitation of any of the Asserted Claims is construed to have a similar meaning, or to encompass similar feature(s) and/or function(s), with any other claim limitation of any of the Asserted Claims, as apparently contended by Bedrock in its Infringement Contentions, or later determined by the Court, and to the extent at least one claim chart in Exhibits B - D identifies any prior art reference, or a portion thereof, as disclosing or teaching such similarly construed claim limitation, such identified prior art reference, or the portion thereof, and the Defendants' contentions with respect to such claim limitation and such prior art reference as found in such claim chart, are incorporated by reference, and are part of, the Defendants' invalidity contentions with respect to each of the Asserted Claims that includes such similarly construed claim limitation.

To the extent that they are prior art, the Defendants reserve the right to rely upon foreign counterparts of the U.S. Patents identified in Defendants' Invalidity Contentions; U.S. counterparts of foreign patents and foreign patent applications identified in the Defendants' Invalidity Contentions; U.S. and foreign patents and patent applications corresponding to articles and publications identified in the Defendants' Invalidity Contentions; and any systems, products, or prior inventions that relate to any references identified in the Defendants' Invalidity Contentions.

The claim charts in Exhibits B - D provide example sections within the prior art references that teach or suggest each and every element of the asserted claims. Each reference or combination of references suggested by each chart indicates whether the prior art renders the claim obvious or anticipated pursuant to P.R. 3-3(b). In Exhibits B - D for each Asserted Claim, the Defendants set forth such obviousness combinations, and the motivation to combine such items.

The U.S. Supreme Court's decision in *KSR International Co. v. Teleflex Inc., et al.*, 550 U.S. 398, 415-16 (2007) *("KSR")* held that a claimed invention can be obvious even if there is no teaching, suggestion, or motivation for combining the prior art to produce that invention. In summary, *KSR* holds that patents that are based on new combinations of elements or components already known in a technical field may be found to be obvious. *See generally KSR*, 550 U.S. 398. Specifically, the Court in *KSR* rejected a rigid application of the "teaching, suggestion, or motivation [to combine]" test. *Id.* at 418. "In determining whether the subject matter of a patent claim is obvious, neither the particular motivation nor the avowed purpose of the patentee controls. What matters is the objective reach of the claim." *Id.* at 419. "Under the correct analysis, any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed." *Id.* at 420. In particular, in *KSR*, the Supreme Court emphasized the principle that "[t]he combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results." *Id.* at 416. A key inquiry is whether the "improvement is more than the predictable use of prior art elements according to their established functions." *Id.* at 417.

The rationale to combine or modify prior art references is significantly stronger when the references seek to solve the same problem, come from the same field, and correspond well. *In re Inland Steel Co.,* 265 F.3d 1354, 1362 (Fed. Cir. 2001). The Federal Circuit allowed two references to be combined as invalidating art under similar circumstances, namely "[the prior art] focus[es] on the same problem that the . . . patent addresses: enhancing [the flexibility of stents]. Moreover, both [prior art references] come from the same field . . . . Finally, the solutions to the identified problems found in the two references correspond well." *Id.* at 1364 (concerning patents and prior art relating to improving the magnetic and electrical properties of steel).

In view of the Supreme Court's *KSR* decision, the PTO issued a set of new Examination Guidelines. *See* Examination Guidelines for Determining Obviousness Under 35 U.S.C. §103 in view of the Supreme Court Decision in *KSR International Co. v. Teleflex, Inc.,* 72 Fed. Reg. 57526 (October 10, 2007). These Guidelines summarized the *KSR* decision, and identified various rationales for finding a claim obvious, including those based on other precedents. Those rationales include:

> (A) Combining prior art elements according to known methods to yield predictable results;
>
> (B) Simple substitution of one known element for another to obtain predictable results;
>
> (C) Use of known technique to improve similar devices (methods, or products) in the same way;
>
> (D) Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
>
> (E) "Obvious to try" – choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;
>
> (F) Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art;

> (G) Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

Id. at 57529. The Defendants contend that one or more of these rationales apply in considering the obviousness of the claims of the ’120 patent.

A person of ordinary skill at the time of the invention had reason to combine or modify one or more of the references listed and charted in Exhibits A - E in light of the knowledge of a person of ordinary skill in the art at the time of the invention and information in the prior art cited herein. For example, all of the references listed and/or charted in Exhibits A - E deal with the application of known techniques for creating and maintaining data structures. Applying these known techniques to a known system to yield predictable results, the creation and maintenance of data in the manner recited by the asserted claims, would have been obvious. Further, many of the references listed and/or charted in Exhibits A - E are excerpts of source code from one or more derivations of a UNIX operating system kernel. For example, the on-the-fly garbage collection techniques disclosed by the Morrison and Van Wyk references were incorporated into a Unix utility called IDEAL. Thus, since each of these references disclose a known technique to a known system to yield a predictable result, it would have been obvious to combine these references with each other as well as with IDEAL. Further, because IDEAL was a Unix utility, it would have been obvious to incorporate these techniques with other Unix and Linux software such as those listed and charted in Exhibit D. For at least the same reason, it would have been obvious to combine the techniques disclosed in the Unix and Linux software identified in Exhibit D with one another. Combining known data creation and storage techniques that have previously been applied within the kernel of derivations of the same operating system kernel to create and maintain data in the manner recited in the asserted claims would have been obvious to one of skill in the art.

It would have been obvious to combine any or all of Linux 1.3.52 route.c, BSD 4.2 if_ether.c, FreeBSD vfs_cache.c, FreeBSD arp.c, FreeBSD wavelan_cs.c, LISP, FreeBSD 2.0.5 kern_proc.c, Linux 1.2.13 – arp.c, Linux 1.3.51 route.c, Xinu Operating System for Sparc, gcache.c, Naval Research Laboratories IPv6 key.c and key.h, Slackware makepsres.c, Linux 2.0.1 route.c, and GCC 2.7.21 with one another, at least because each uses linked data structures to store expiring data. Further, each system is a data management utility designed to operate in a computer operating system. Additionally each system is associated with an open-source operating system and/or utility. Thus, a person of ordinary skill in the art would recognize that the combination of any of these systems is a predictable use of elements known in the art to solve a known problem,.

Furthermore, it would have been obvious to combine any or all of the above systems with the system described in the Morrison reference (see Exhibit C-2). On information and belief, the system described in the Morrison reference was implemented and distributed in a Unix utility known as IDEAL. Like the above systems, IDEAL used linked data structures to manage expiring data in a Unix environment. Thus, a person of ordinary skill in the art would recognize that the combination of Morrison with the systems listed above is a predictable use of elements known in the art to solve a known problem. Additional contentions regarding the combination and/or modification of and/or motivation to combine references for specific Asserted Claims are set forth in the claim charts of Exhibits B - D.

B. Contentions Under P.R. 3-3(d)

Pursuant to Patent Local Rule 3-3(d), the Defendants contend that certain claims of the Asserted Patent are invalid under 35 U.S.C. § 112 because: (1) the claims are indefinite; (2) the claims are not enabled; (3) the claims lack adequate written description; and/or (4) the specification fails to set forth the best mode contemplated by the inventor for practicing the invention. The Defendants' contentions that the following claims are invalid under 35 U.S.C. § 112 are made in the alternative, and do not constitute, and should not be interpreted as, admissions regarding the construction or scope of the claims of the '120 patent, or that any of the claims of the '120 patent are not anticipated or rendered obvious by any prior art.

The asserted claims identified below are invalid under 35 U.S.C § 112 paragraph 2, which requires that the specification "conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention."

Claims 1 and 5 require "a record search means utilizing a search key to access the linked list" and "a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed." Claim 1 further requires "a means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list." Claim 5 further requires "a means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records." These limitations can be construed to cover only the corresponding specific algorithmic structure disclosed in the specification under 35 U.S.C. 112, paragraph 6. *See WMS Gaming, Inc. v. Int'l Game Tech.*, 184 F.3d 1339, 1349 (Fed. Cir. 1999). Because no specific algorithm is disclosed in the specification for these limitations, Claims 1 and 5 (and the claims that depend from them) are invalid as indefinite under 35 U.S.C. §112, paragraph 2.

Claim 2 requires "means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records." Claim 6 requires "means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records." These limitations can be construed to cover only the corresponding specific algorithmic structure disclosed in the specification under 35 U.S.C. 112, paragraph 6. *See WMS Gaming, Inc. v. Int'l Game Tech.*, 184 F.3d 1339, 1349 (Fed. Cir. 1999). Because no specific algorithm is disclosed in the specification for "means for dynamically determining maximum number" Claims 2 and 6 are invalid as indefinite under 35 U.S.C. §112, paragraph 2.

Claim 7 and Claim 8 require "the system." This is indefinite, violating 35 U.S.C. §112, paragraph 2. Thus, Claims 7 and 8 are invalid as indefinite under 35 U.S.C. §112, paragraph 2.

The asserted claims identified below are invalid under 35 U.S.C. § 112 paragraph 1, which requires that the specification "contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention."

Claims 2, 4, 6, and 8 each require "dynamically determining maximum number." Neither the '120 patent nor its application describe "dynamically determining maximum number." Thus, Claims 2, 4, 6, and 8 are invalid for lack of written description and/or enablement under 35 U.S.C. §112, paragraph 1. Also, the means-plus-function claims noted above as invalid under 35 U.S.C. §112, paragraph 2 are invalid under 35 U.S.C. §112, paragraph 1 for the same reasons.

III. ADDITIONAL PRIOR ART

In addition to the prior art references charted, the Defendants list in Exhibit E, which is incorporated herein in its entirety, additional prior art references that are pertinent to the invalidity of the '120 patent. At this time, the Defendants are not providing claim charts for each of these additional references either because these references:

(1) have similar disclosure to the prior art references for which invalidity charts have been provided;

(2) were discovered recently and Defendants have not had a fair opportunity to analyze them;

(3) may be used to show state of the art; and/or

(4) may be used as supporting references in an obviousness combination depending on how the claims are ultimately construed by the Court.

The Defendants also incorporate, in full, all prior art references cited in the ’120 patent and all prior art references cited in the prosecution histories of the ’120 patent and any foreign counterparts.

The Defendants reserve the right to revise these Contentions to rely on any of these references to prove the invalidity of the asserted claims of the ’120 patent in a manner consistent with the Federal Rules of Civil Procedure, the Court’s Local Rules, and the Local Patent Rules.

IV. **ACCOMPANYING DOCUMENT PRODUCTION**

Pursuant to Patent Rule 3-4(a), the Defendants will produce, make available for inspection, or identify publicly available information sufficient to show the operation of any specifically identified aspects or elements of an Accused Instrumentality identified by Bedrock in its P. R. 3-1(c) chart to the extent such information is in the Defendants’ possession, custody or control. If such information comprises source code, the Defendants will produce publicly available source code or make non-public source code available for inspection after the entry of a suitable protective order in this action.

Pursuant to Patent Rule 3-4 (b), the Defendants are producing or making available for inspection copies of each item of prior art identified pursuant to Patent Rule 3-3(a) which does not appear in the file history of the Asserted Patent. To the extent that such item is not in English, an English translation is produced where available. The Defendants reserve the right to identify and produce additional documents pursuant to the Patent Rules and the orders of the Court.

Table of Exhibits	
Exhibit	**Description**
A.	List of prior art references that have been charted
B.	Invalidity charts for prior art patent references listed in Exhibit A
C.	Invalidity charts for prior art literature references listed in Exhibit A
D.	Invalidity charts for prior art systems listed in Exhibit A
E.	Additional prior art

Respectfully submitted, this the 18th day of October 2010.

/s/ E. Danielle T. Williams
Steven Gardner
E. Danielle T. Williams
John C. Alemanni
Alton L. Absher III
KILPATRICK STOCKTON LLP
1001 West 4th Street
Winston-Salem, NC 27101
Telephone: 336-607-7300
Facsimile: 336-607-7500

William H. Boice
Russell A. Korn
KILPATRICK STOCKTON LLP
Suite 2800
1100 Peachtree Street
Atlanta, GA 30309-4530
Telephone: 404-815-6500
Facsimile: 404-815-6555

J. Thad Heartfield
Texas Bar No. 09346800
thad@jth-law.com
M. Dru Montgomery
Texas Bar No. 24010800
dru@jth-law.com
THE HEARTFIELD LAW FIRM
2195 Dowlen Road
Beaumont, TX 77706
Telephone: 409-866-2800
Facsimile: 409-866-5789

Attorneys for Plaintiff RED HAT, INC., and Cross-Claim Defendants NYSE EURONEXT, RACKSPACE HOSTING, INC., THEPLANET.COM INTERNET SERVICES, INC., AND WHOLE FOODS MARKET, INC.

/s/ Michael E. Jones
Michael E. Jones
State Bar No. 10929400
Allen F. Gardner
State Bar No. 24043679
POTTER MINTON
A Professional Corporation
110 N. College Ave., Suite 500 (75702)
P.O. Box 359
Tyler, TX 75710
Telephone: 903-597-8311
Facsimile: 903-593-0846

H. Michael Hartmann
Robert T. Wittmann
LEYDIG, VOIT & MAYER, LTD.
Two Prudential Plaza
180 North Stetson, Suite 4900
Chicago, Illinois 60601-6731
Telephone: 312-616-5600
Facsimile: 312-616-5700

J. Christopher Erb
ERB LAW FIRM P.C.
5901 Ridge Ave., Suite 100
Philadelphia, PA 19128
Telephone: 215-508-4419
Facsimile: 215-508-4428

Attorneys for Cross-Claim Defendant 1& 1 INTERNET, INC..

/s/ Neil J. McNabnay
Thomas M. Melsheimer
Neil J. McNabnay
J. Nicholas Bunch
FISH & RICHARDSON P.C.
1717 Main Street - Suite 5000
Dallas, TX 75201
Tel: (214) 747-5070
Fax: (214-747-2091

Christopher Hadley
FISH & RICHARDSON P.C.
One Marina Park Drive
Boston, MA 02110
Tel: (617) 542-5070
Fax: (617) 542-8906

Attorneys for Crossclaim Defendant CONOCOPHILLIPS COMPANY

/s/ Heidi Keefe
Heidi Keefe (CA Bar No. 178960)
Mark Weinstein. (CA Bar No. 193043)
Adam Pivovar (CA Bar No. 246507) (*Pro Hac Vice*)
Lam K. Nguyen (CA Bar No. 265285)
COOLEY LLP
Five Palo Alto Square
3000 El Camino Real
Palo Alto, CA 94306-2155
Telephone: (650) 843-5000
Facsimile: (650) 857-0663
hkeefe@cooley.com
mweinstein@cooley.com
apivovar@cooley.com
lnguyen@cooley.com

Deron R. Dacus
State Bar No. 00790553
RAMEY & FLOCK, P.C.
100 E. Ferguson, Suite 500
Tyler, Texas 75702
Phone: (903) 597-3301
Fax: (903) 597-2413
derond@rameyflock.com

Attorneys for Crossclaim Defendant FACEBOOK, INC.

/s/ Jennifer H. Doan
Jennifer H. Doan
Texas Bar No. 08809050
HALTOM & DOAN
Crown Executive Center, Suite 100
6500 Summerhill Road
Texarkana, TX 75503
Telephone: (903) 255-1000
Facsimile: (903) 255-0800
Email: jdoan@haltomdoan.com

Andrew M. Grove
HONIGMAN MILLER SCHWARTZ AND COHN LLP
38500 Woodward Avenue, Suite 100
Bloomfield Hills, MI 48304
Telephone: (248) 566-8432
Facsimile: (248) 566-8433
Email: jgrove@honigman.com

/s/ Brian W. LaCorte
Brian W. LaCorte (pro hac vice)
GALLAGHER & KENNEDY, P.A.
2575 East Camelback Road
Phoenix, Arizona 85016-9225
Telephone: (602) 530-8020
Facsimile: (602) 530-8500
Email: bwl@gknet.com

Harry L. Gillam, Jr.
GILLAM & SMITH
303 South Washington Avenue
Marshall, Texas 75670
Telephone: (903) 934-8450
Facsimile: (903) 934-9257
Email: gil@gillamsmithlaw.com

Attorneys for Crossclaim Defendant THE GO DADDY GROUP, INC.

Emily J. Zelenock
HONIGMAN MILLER SCHWARTZ AND COHN LLP
38500 Woodward Avenue, Suite 100
Bloomfield Hills, MI 48304
Telephone: (248) 566-8508
Facsimile: (248) 566-8509
Email: ezelenock@honigman.com

Attorneys for Crossclaim Defendant R .L. POLK & CO.

/s/ Yar R. Chaikovsky
Yar R. Chaikovsky
California State Bar No. 175421
McDermott Will & Emery LLP
275 Middlefield Road
Suite 100
Menlo Park, California 94025-4004

Attorneys for Crossclaim Defendant THE GAP, INC.

/s/ Lynn H. Pasahow________
Lynn H. Pasahow, CA Bar No. 054283
(Admitted E.D. Texas)
Darren E. Donnelly, CA Bar No. 194335
(Admitted E.D. Texas)
Saina S. Shamilov, CA Bar No. 215636
(Admitted E.D. Texas)
Leslie Kramer, CA Bar No. 253313
(Admitted E.D. Texas)
FENWICK & WEST LLP
801 California Street
Mountain View, CA 94041
Telephone: (650) 988-8500
Facsimile: (650) 938-5200

Attorneys for Crossclaim Defendant VIRGIN AMERICA, INC

/s/Rick L. Rambo/__________
Rick L. Rambo
State Bar No. 00791479
MORGAN, LEWIS & BOCKIUS LLP
1000 Louisiana, Suite 4000
Houston, Texas 77002
Telephone No. (713) 890-5000
Telecopier No. (713) 890-5001
rrambo@morganlewis.com

Attorneys for Crossclaim Defendant NATIONWIDE MUTUAL INSURANCE COMPANY

/s/ James A. Glenn/
Paul E. Krieger
State Bar No. 11726470
James A. Glenn
State Bar No. 24032357
MORGAN, LEWIS & BOCKIUS LLP
1000 Louisiana, Suite 4000
Houston, Texas 77002
Telephone No. (713) 890-5000
Telecopier No. (713) 890-5001
pkrieger@morganlewis.com
jglenn@morganlewis.com

Attorneys for Crossclaim Defendant
SUNGARD DATA SYSTEMS INC.

/s/ Elizabeth L. DeRieux
S. Calvin Capshaw, III
State Bar No. 03783900
Elizabeth L. DeRieux
State Bar No. 05770585
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
Longview, Texas 75601
Telephone: (903) 236-9800
Facsimile: (903) 236-8787
E-mail: ccapshaw@capshawlaw.com
E-mail: ederieux@capshawlaw.com

Allen W. Hinderaker
ahinderaker@merchantgould.com
Christopher J. Sorenson (Lead Attorney)
csorenson@merchantgould.com
MERCHANT & GOULD, PC
3200 IDS Center
80 South Eighth Street
Minneapolis, MN 55402
Telephone: 612.332.5300
Facsimile: 612.332.9081

Attorneys for Crossclaim Defendant
CONAGRA FOODS INC

EXHIBIT A

IDENTIFICATION OF PRIOR ART

Pursuant to P.R. 3-3(a) and 3-3(b), Defendants identify prior art references and combinations of prior art on which Defendants intend to rely for their contentions that one or more asserted claims of U.S. Patent Number 5,893,120 ("the '120 patent") are invalid. The Defendants provide the following chart to help summarize the Defendants' Invalidity Contentions. To the extent that a claim chart in Exhibit B, C, and/or D identifies that a reference anticipates and/or presents obviousness combinations that are not represented in the following charts, the claim chart satisfies the Defendants' disclosure under P.R. 3-3(b).

I. ANTICIPATING PRIOR ART

At least the following prior art references anticipate one or more of the asserted claims of the '120 patent.

A. Prior Art Patents that Anticipate the '120 Patent under 35 U.S.C. §§ 102(a), (b), (e), and/or (g)

Defendants identify the following United States patents as prior art references that anticipate the Asserted Claims of the '120 patent.

Country of Origin, Patent No., Inventor, Date of Issue	Anticipates at Least Claims:	Exhibit
U.S. Patent No. 4,695,949, Thatte et al., September 22, 1987 ("Thatte").	1 - 8	B-1
U.S. Patent No. 6,119,214, Dirks, September 12, 2000 ("Dirks").	1 - 8	B-2
U.S. Patent No. 4,989,132, Mellender et al., January 29, 1991 ("Mellender").	1 - 8	B-3
U.S. Patent No. 5,043,885, Robinson, August 27, 1991 ("Robinson").	1, 3, 5, 7	B-4

EXHIBIT A

Country of Origin, Patent No., Inventor, Date of Issue	Anticipates at Least Claims:	Exhibit
U.S. Patent No. 5,778,430, Ish et al., July 7, 1998 ("Ish").	1 - 8	B-5
U.S. Patent No. 5,991,775, Beardsley et al., November 23, 1999 ("Beardsley").	1 - 8	B-6
U.S. Patent No. 5,765,174, Bishop, June 9, 1998 ("Bishop").	1 - 8	B-7
U.S. Patent No. 6,243,667, Kerr et al., June 5, 2001 ("Kerr").	1, 3, 5, 7	B-10
U.S. Patent No. 5,881,241, Corbin, March 9, 1999 ("Corbin").	1, 3	B-11
U.S. Patent No. 4,996,663, Nemes, February 26, 1991 ("the '663 patent").	1 - 8	B-13
U.S. Patent No. 5,577,237, November 19, 1996 ("the '237 patent")	1 - 8	B-14

B. Prior Art Publications that Anticipate the '120 Patent Under 35 U.S.C. §§ 102(a) and/or (b)

Defendants identify the following publications as prior art references that anticipate the Asserted Claims of the '120 patent.

Author, Title, Publisher, Publication Information, Date of Publication	Anticipates at Least Claims:	Exhibit
Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, Springer-Verlag New York, Algorithmica 1:17-29, 1986 ("Van Wyk").	1, 3, 5, 7	C-1
John A. Morrison, Larry A. Shepp, and Christopher J. Van Wyk, *A Queueing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6:1155-1164, December 1987 ("Morrison"); and on information and belief, the source code for IDEAL.	1, 3, 5, 7	C-2
Claire M. Matheiu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, Unite de Recherche	1, 3, 5, 7	C-3

EXHIBIT A

Author, Title, Publisher, Publication Information, Date of Publication	Anticipates at Least Claims:	Exhibit
Inria-Rocquencort Institut National de recherché en Informatique, June 1988 (“Matheiu”).		
Claire M. Kenyon-Matheiu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, Springer Berlin/Heidelberg, Automata, Languages and Programming, 473-487, Vol. 372, 1989 (“Kenyon-Matheiu”).	1, 3, 5, 7	C-4
David Aldous, Micha Hofri, and Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, Society for Industrial and Applied Mathematics, Vol. 21, No. 4:713-732, August 1992 (“Aldous”).	1, 3, 5, 7	C-5
Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).	1 - 8	C-6
Roger Sessions, *Reusable Data Structures for C* Prentice-Hall, Inc. 1989 (“Sessions”).	1 - 4	C-9
Christopher J. Van Wyk, *Data Structures and C Programs*, Addison-Wesley Publ’g Co. & Bell Telephone Laboratories, Inc. 1988 (“Van Wyk 2”).	1 - 8	C-10
Mark Allen Weiss, *Data Structures & Algorithm Analysis in C*, The Benjamin/Cummings Publ’g Co. 1993 (“Weiss”).	1 - 8	C-11
William B. Frakes & Ricardo Baeza-Yates, *Information Retrieval: Data Structures & Algorithms*, Prentice-Hall, Inc. 1992 (“Frakes”).	1, 3, 5, 7	C-12
Eric W. Brown, *Execution Performance Issues in Full Text Information Retrieval*, University of Massachusetts Amherst, October 1995 (“Brown”).	1, 3 , 5, 7	C-13
Costello, Adam, et al., *Redesigning the BSD - Callout and Time Facilities*, WUSC 95-23, November 2, 1995 (“Costello”).	1 - 8	C-14

EXHIBIT A

Author, Title, Publisher, Publication Information, Date of Publication	Anticipates at Least Claims:	Exhibit
J.M. Foster, *List Processing*, Macdonald & Co., 1967 (“Foster”).	1, 3, 5, 7	C-15
Sirinivasan Keshav, *On the Efficient Implementation of Fair Queueing*, Journal of Internetworking: Research and Experience, 1991 (“Keshav”).	1 – 8	C-16
George Varghese and Tony Lauck, *Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility*, ACM SIGOPS Operating Systems Review, Vol. 21, Issue 5, p. 25-38 (November 1987) (“Varghese and Lauck”).	1 - 8	C-17
Robert L. Kruse, *Data Structures and Program Design* Prentice-Hall, Inc. 1984 and 1987 (“Kruse”).	1, 3, 5, 7	C-18
Joseph T. Dixon and Kenneth Calvert, *Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)* (“Dixon and Calvert”)	1 - 8	C-19

C. Software that Anticipate the ’120 Patent Under 35 U.S.C. §§ 102(a), (b), and/or (g)

Defendants identify the following software as prior art references that anticipate the claims of the ’120 patents. Each piece of software was at least (1) known or used in this country and/or described in a printed publication in this or a foreign country, before the alleged invention of the claimed subject matter of the patent-in-suit, and/or (2) the invention was described in a printed publication in this or a foreign country and/or in public use and/or on sale in this country, more than one year before the filing date of the application for the patent-in-suit, and/or (3) was invented and not abandoned, suppressed, or concealed prior to the alleged invention of the patent-in-suit. Thus, each piece of software identified here qualifies at least as prior art both as a publication and as a prior art system or apparatus.

Software, Date Invented / Made / Used / Sold by	Anticipates at Least Claims:	Exhibit
Linux 1.3.52 - route.c, released on December 29, 1995 to the public.	1, 3, 5, 7	D-1
BSD 4.2 - if_ether.c, released to the public as part of the BSD 4.2 open source operating system in September 1983.	1 - 8	D-2
FreeBSD - vfs_cache.c, developed as part of the FreeBSD operating system, made public on Dec. 14, 1995 at http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/vfs_cache.c	1 - 8	D-3
FreeBSD - arp.c, 1994.	1, 3, 5, 7	D-4
FreeBSD - wavelan_cs.c, 1995.	1, 3, 5, 7	D-5
LISP, September 1981.	1 – 4	D-6
FreeBSD 2.0.5 – kern_proc.c, released on June 10, 1995 to the public.	1 – 8	D-7
Linux 1.2.13 - arp.c, released on August 2, 1995 to the public.	1 – 8	D-8
Linux 1.3.51 - route.c, released on December 27, 1995 to the public.	1 – 8	D-9
gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, *GCache: A Generalized Caching Mechanism*, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)	1 – 8	D-10
Naval Research Laboratories IPv6 – key.c and key.h, August 1995	1 – 8	D-11
Linux 2.0.1 - route.c, July 1996	1-8	D-12
GCC 2.7.2.1, August 1996	1-8	D-13

MK84 – net_filter() in net_io.c, 1993	1-8	D-14
MK84 – net_set_filter() in net_io.c, 1993	1-8	D-15
Plug and Play Linux – makepsres.c, 1995	1-8	D-16
Local Area Transport Protocol, prior to January 2, 1997	1-8	D-17

II. OBVIOUSNESS BASED ON COMBINATIONS OF PRIOR ART

The following list identifies combinations of prior art that Defendants presently intend to rely on for their contentions that one or more of the asserted claims of the ’120 patent are obvious.

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
Dirks	U.S. Patent No. 5,724,538, Morris et al., March 3, 1998 (“Morris”); Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	B-2
Robinson	U.S. Patent No. 4,530,054, Hamstra et al., July 16, 1985 (“Hamstra”); Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	B-4
Ish at al.	Hamstra; Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	B-5
Beardsley	Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	B-6
Bishop	U.S. Patent No. 5,991,775, Beardsley et al., November 23, 1999 (“Beardsley”); Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or	1 - 8	B-7

EXHIBIT A

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
	Opportunistic Garbage Collection		
U.S. Patent No. 5,918,249, Cox et al., June 29, 1999.	Beardsley; Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	B-8
U.S. Patent No. 6,424,992, Devarakonda et al., July 23, 2002.	Dirks; Thatte; the ’663 Patent; *The Art of Computer Programming*, Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 513, 518, 1973 (“Knuth”); Weiss; and/or Robert L. Kruse, *Data Structures and Program Design* (Prentice-Hall, Inc. 1984) (“Kruse”); Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	B-9
Kerr	Dirks; Thatte; the ’663 patent; and/or Linux 2.0.1; Opportunistic Garbage Collection	1 - 8	B-10
Corbin	Dirks; Thatte; the ’663 patent; and/or Linux 2.0.1; Opportunistic Garbage Collection	1 - 8	B-11
U.S. Patent No. 5,121,495 Nemes, June 9, 1992 (“the ‘495 Patent”).	Dirks; Thatte; the ’663 patent; and/or Linux 2.0.1; Opportunistic Garbage Collection	1 - 8	B-12
U.S. Patent No. 5,577,237, November 19, 1996 (“the ‘237 patent”)	Dirks; Thatte; the ’663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	B-14
Van Wyk	Kruse; Knuth; Dirks; Thatte; U.S. Patent No. 4,996,663, Nemes, February 26, 1991 (“the ’663 Patent”); and/or Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA ’89 Proceedings, October 1-6, 1989, and Paul R. Wilson, *Opportunistic Garbage*	1 - 8	C-1

EXHIBIT A

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
	Collection, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988 (collectively, "Opportunistic Garbage Collection"); and/or Linux 2.0.1.		
Morrison	Kruse; Knuth; Van Wyk; Dirks; Thatte; Mathieu; Kenyon-Mathieu; Aldous; the '663 Patent; Linux 2.0.1; and/or Opportunistic Garbage Collection.	1 - 8	C-2
Matheiu	Kruse; Knuth; Van Wyk; Dirks; Thatte; Morrison; Kenyon-Mathieu; Aldous; the '663 Patent; Linux 2.0.1; and/or Opportunistic Garbage Collection.	1 - 8	C-3
Kenyon-Matheiu	Kruse; Knuth; Van Wyk; Dirks; Thatte; Morrison; Mathieu; Aldous; the '663 Patent; Linux 2.0.1; and/or Opportunistic Garbage Collection.	1 - 8	C-4
Aldous	Kruse; Knuth; Van Wyk; Dirks; Thatte; Morrison; Mathieu; Kenyon-Mathieu; the '663 Patent; Linux 2.0.1; and/or Opportunistic Garbage Collection.	1 - 8	C-5
James Nelson Griffoen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, Purdue University Thesis, August 1991.	Dirks; Thatte; the '663 patent; Opportunistic Garbage Collection; Linux 2.0.1; and/or Weiss	1 - 8	C-7
Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model*, in Proceedings of	Dirks; Thatte; the '663 patent; Opportunistic Garbage Collection; Linux 2.0.1; and/or Weiss	1 - 8	C-8

EXHIBIT A

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
the USENIX Summer Conference, June 1990.			
Sessions	Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester"); Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 5 - 8	C-9
Van Wyk 2	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	C-10
Weiss	Kruse; Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	C-11
Frakes	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	C-12
Brown	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	C-13
Foster	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	C-15
Keshav	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	1 - 8	C-16
Kruse	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	C-18
Dixon and Calvert	Dirks; Thatte; the '663 patent; Linux 2.0.1; and/or Opportunistic Garbage Collection	2, 4, 6, 8	C-19
Linux 1.3.52 - route.c	Dirks, Thatte, the '663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1 - 8	D-1

EXHIBIT A

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
BSD 4.2 - if_ether.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Knuth; Kruse; and/or Weiss; Linux 2.0.1; GCache	1 - 8	D-2
FreeBSD - vfs_cache.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Knuth; and/or Kruse; Linux 2.0.1; GCache	1 - 8	D-3
FreeBSD - arp.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1 - 8	D-4
FreeBSD -wavelan_cs.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1 - 8	D-5
Lisp	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	5 – 8	D-6
FreeBSD – kern_proc.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	2, 4, 6, 8	D-7
Linux 1.2.13 – arp.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	2, 4, 6, 8	D-8
Linux 1.3.51 - route.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	2, 4, 6, 8	D-9
Xinu Operating System for Sparc – gcache.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; Naval Research Laboratories IPv6 – key.c and key.h	2, 4, 6, 8	D-10
Naval Research	Dirks, Thatte, the ’663 patent, and/or	1 - 8	D-11

EXHIBIT A

Prior Art Reference	Prior Art Reference	Renders Obvious at Least Claims:	Exhibit
Laboratories IPv6 – key.c and key.h	Opportunistic Garbage Collection; Kruse; Linux 2.0.1; GCache		
Linux 2.0.1 - route.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; GCache	1-8	D-12
GCC 2.7.21	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1-8	D-13
MK84 – net_filter() in net_io.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1-8	D-14
MK84 – net_set_filter() in net_io.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1; GCache	1-8	D-15
Plug and Play Linux – makepsres.c	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Kruse; Linux 2.0.1; GCache	1-8	D-16
Local Area Transport Protocol	Dirks, Thatte, the ’663 patent, and/or Opportunistic Garbage Collection; Linux 2.0.1	1-8	D-17

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Thatte discloses an information storage and retrieval system. For example, Thatte discloses: A method and apparatus for managing a block oriented memory of the type in which each memory block has an associated reference count representing the number of pointers to it from other memory blocks and itself. Efficient and cost-effective implementation of reference counting alleviates the need for frequent garbage collection, which is an expensive operation. The apparatus includes a hash table into which the virtual addresses of blocks of memory which equal zero are maintained. When the reference count of a block increases from zero, its virtual address is removed from the table. When the reference count of a block decreases to zero, its virtual address is inserted into the table. When the table is full, a reconciliation operation is performed to identify those addresses which are contained in a set of binding registers associated with the CPU, and any address not contained in the binding registers are evacuated into a garbage buffer for subsequent garbage collection operations. The apparatus can be implemented by a cache augmented by the hash table, providing a back-up store for the cache. Thatte et al., “Method for Efficient Support For Reference Counting,” U.S. Patent No. 4,695,949 (issued Sept, 22, 1987).at Abstract. In accordance with a broad aspect of the invention, an apparatus is provided for managing a block oriented memory of the type in which each memory block has an associated reference count representing the number of pointers to it from other memory blocks and itself. The apparatus includes means for implementing a data structure storing

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		virtual addresses, which can be implemented by a hash table or the like. Means are also provided for performing insert and delete operations on the data structure, means for inserting in the data structure the virtual address of each block of memory which has a reference count of zero, and means for deleting from the data structure the virtual address of each block of memory whose reference count changes from zero to one. In one aspect of the invention, the apparatus further includes means for performing a reconciliation operation on the data structure when it is full, the reconciliation means including means for obtaining a dump of pointers in binding registers, means for comparing the pointers of the pointer dump with the virtual addresses contained in the data structure, and means for deleting any virtual addresses contained in the data structure not in the pointer dump. *Id.* at 5:40-62. The method and apparatus for reference count management assistance disclosed in Thatte is referred to as a "reference count filter," which acts conceptually as a filter to control reference count management. *Id.* at 5:66-68. The reference count filter stores virtual addresses of blocks whose reference counts have dropped to zero, but which may have pointers to them at binding registers. *Id.* at 6:51-57. The method and apparatus also contains a memory management unit (MMU) 40 which provides memory management functions. *Id.* at 6:4-24. "[T]he MMU 40 is responsible for reference count management; that is, it increments and decrements reference counts of the referent blocks, when pointers to these blocks are created or destroyed in memory cells. The MMU 40 is also responsible for reclaiming inaccessible blocks." *Id.* at 6:25-31. "The MMU 40 maintains information about blocks which have zero reference counts, but which may have pointers to them originating at the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		binding registers. This information is conveniently recorded and maintained in a data structure, [such as a hash table as shown in Figure 7], in the form of virtual-addresses of such blocks." *Id.* at 6:36-41

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Thatte discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Thatte also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Thatte's reference count filter is preferably implemented in a "hash table that can efficiently support the insert, delete, and reconcile operations. Therefore, the basic search operation on the hash table must be quite fast." *Id.* at 8:39-43. "The hash table implementation of the reference count filter, in accordance with the invention, is illustrated in FIG. 7." *Id.* at 8:49-81. Figure 7 displays a virtual address 80 that is applied to a hash function 92. *Id.* at Figure 7, 8:39-62. The hashed output is then inserted into a hash bucket 83 which comprises a linked list 85, 86. *Id.*

EXHIBIT B-1

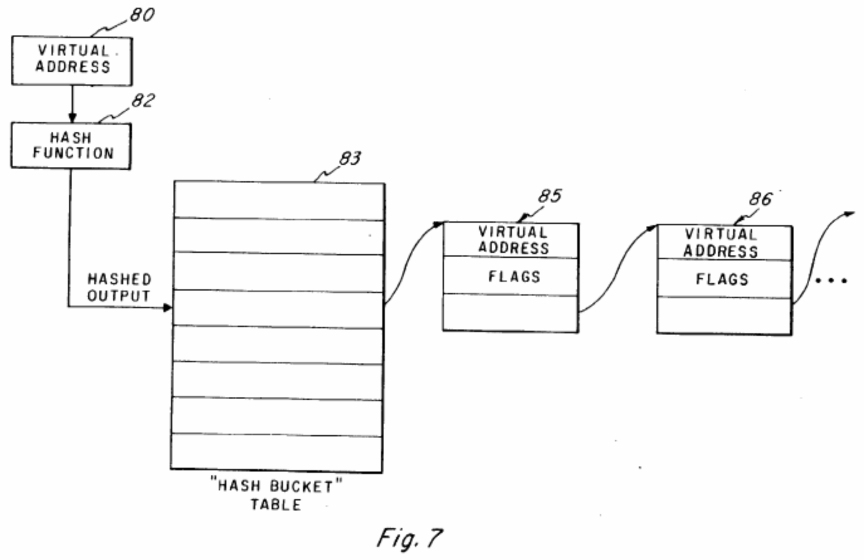
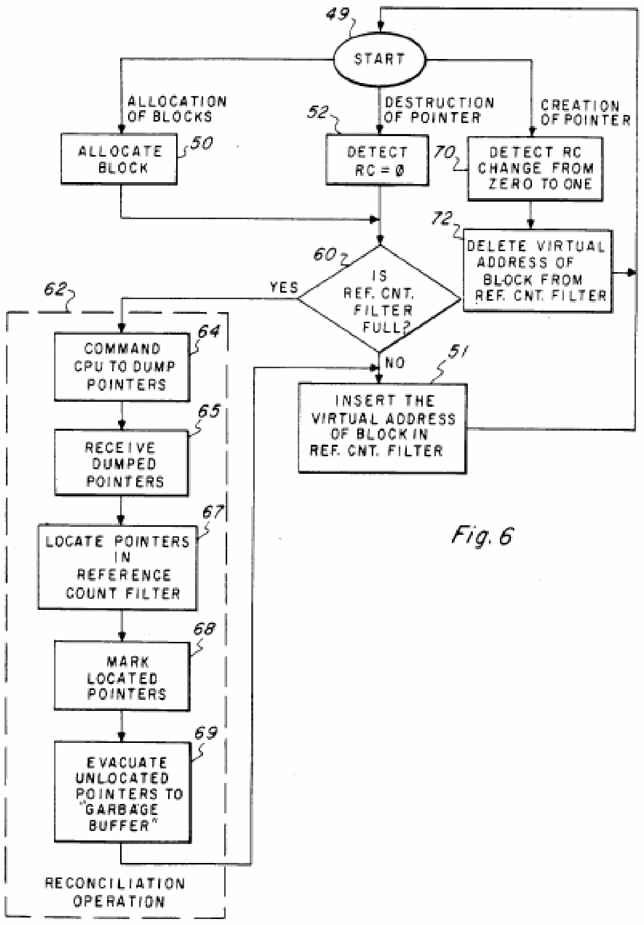
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		 80 VIRTUAL ADDRESS 82 HASH FUNCTION HASHED OUTPUT 83 "HASH BUCKET" TABLE 85 VIRTUAL ADDRESS FLAGS 86 VIRTUAL ADDRESS FLAGS Fig. 7 *Id.* at Figure 7. The records that are contained within the reference count filter automatically expire. *Id.* at 7:40-8:18. When the reference count filter is full a reconciliation operation 62 is performed. *Id.* The reconciliation operation removes all of the garbage blocks, i.e. virtual addresses stored in the hash table that are no longer referenced externally. *Id.* The MMU 40 makes the necessary room by performing a reconciliation operation, box 62. It sends a special command to the CPU 12 called "Dump-pointers," box 64. In response, the CPU 12 sends the contents of all binding registers that contain pointers (which are virtual addresses) to the MMU. The set of these pointers is called the "Dumped-out" set, which is received by the MMU, box 65. The pointers in the Dumped-out set indicate the block which have

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		references originating at binding registers. Of course, the size of the dumped-out set cannot exceed the number of binding registers. The reconciliation operation is guaranteed to create a room in the reference count filter, as the size of the reference count filter is greater than the number of binding registers. Therefore, there must be at least one virtual address in the reference count filter which is not in any binding register. *Id.* at 7:44-60. The MMU reconciles the state of the reference count filter with the set of Dumped-out pointers, by executing the following operations. (1) Each pointer from the dumped-out set is attempted to be located by the MMU within the virtual address contained in the reference count filter, box 67. If the pointer exists in the reference count filter, the MMU marks the pointer in the reference count filter, box 68. The pointers in the reference count filter, thus marked, indicate blocks that are still accessible and hence are not garbage. All unmarked pointers in the reference count filter, therefore, indicate garbage blocks. (2) The unmarked pointers are evacuated from the reference count filter and stored in another data structure, called the "garbage buffer" (not shown), box 69, which essentially holds pointers to garbage blocks. A background process not described herein in the MMU operates on the garbage buffer to reclaim the garbage blocks. As soon as the unmarked pointers are evacuated from the reference count filter to the garbage buffer, the reconciliation operation on the reference count filter is over, and the regular operation is resumed. As a result of the reconciliation operation, the state of the reference count filter has been reconciled with the state of binding registers, and pointers to all garbage blocks have been evacuated from the reference count filter. *Id.* at 7:61-8:18.
[1b] a record search means utilizing a search key to access the linked	[5b] a record search means utilizing a search key to access a linked list	Thatte discloses a record search means utilizing a search key to access the linked list. Thatte also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
list,	of records having the same hash address,	For example, Thatte states that "the preferred implementation of the reference count filter is a hash table that can efficiently support the insert, delete, and reconcile operations. Therefore, the basic search operation on the hash table must be quite fast." *Id.* at 8:39-43. As shown in Figure 7: [A] virtual address, shown in block 80, is applied to means for implementing a hash function, shown in block 82. Hash function implementing functions are well known in the art, and are not described in detail herein. The hashed output from the hash function is applied as an address to a "hash bucket" table 83, in which a corresponding entry is located. The located entry may be a pointer which may point to a linked list 85, 86, etc. of virtual addresses. If the virtual address searched for (i.e. the virtual address contained in block 80) is in the linked list 85, 86, etc., the "locate" operation is successful. *Id.* at 8:49-62.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Thatte discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Thatte also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Figure 6 displays a block diagram showing the steps of maintaining the reference count filter. *Id.* at 5:10-12. Each time a block is allocated or a pointer to a block is destroyed the reference count filter is

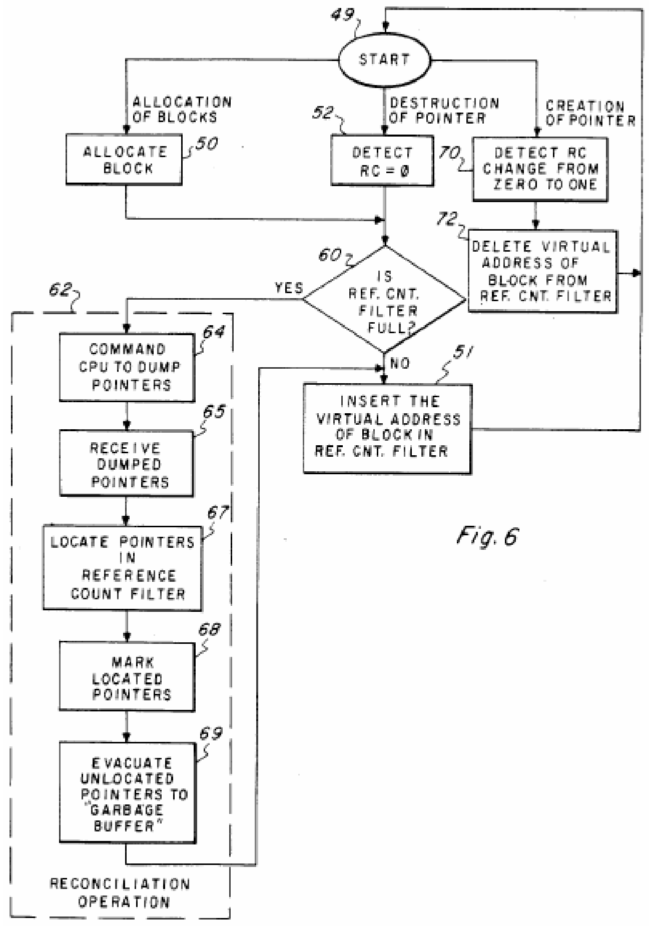
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		checked to see if the reference count filter is full. *Id.* at 7:1-26, Figure 6. If the reference count filter is full, then a reconciliation operation is performed where expired records are removed. *Id.*  49 START ALLOCATION OF BLOCKS DESTRUCTION OF POINTER CREATION OF POINTER 50 ALLOCATE BLOCK 52 DETECT RC = Ø 70 DETECT RC CHANGE FROM ZERO TO ONE 72 DELETE VIRTUAL ADDRESS OF BLOCK FROM REF. CNT. FILTER 60 IS REF. CNT. FILTER FULL? YES NO 51 INSERT THE VIRTUAL ADDRESS OF BLOCK IN REF. CNT. FILTER 62 64 COMMAND CPU TO DUMP POINTERS 65 RECEIVE DUMPED POINTERS 67 LOCATE POINTERS IN REFERENCE COUNT FILTER 68 MARK LOCATED POINTERS 69 EVACUATE UNLOCATED POINTERS TO "GARBAGE BUFFER" RECONCILIATION OPERATION Fig. 6 *Id.* at Figure 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		Beginning from a start position 49, three possible block affecting operations may be performed. The first is the allocation of a block [i.e. insert], the second is the destruction of a pointer to a block [i.e. delete], and the third is the creation of a pointer to a block. Accordingly, when a new block is allocated by the MMU 40 in response to an "Allocate" command from the CPU 12, box 50, the new block starts its life with a zero reference count. Therefore, in accordance with the invention, the virtual address of a newly allocated block is inserted into the reference count filter 25, box 51, assuming that there is a place in the reference count filter for insertion, i.e., that the reference count filter is not full. Similarly, when the reference count of a block drops to zero, box 52, the virtual address of the block is inserted in the reference count filter 25, again assuming that there is a place in the reference count filter for [insertion]. The insert operation on the reference count filter is implemented by the insert operation on the underlying hash table, as below described. *Id.* at 7:1-20. If the reference count filter is determined to be full, box 60, the MMU suspends the insertion operation and performs a reconciliation operation, box 62, on the reference count filter, as described below, to create a room in the reference count filter so that the suspended insertion operation can be completed. *Id.* at 7:21-26. When the reference count of a block goes up from zero to one, box 70, the virtual address of the block is deleted from the reference count filter, box 72. The deletion operation is necessary because a block with a non-zero reference count must not stay in the reference count filter. To accomplish the deletion, first the virtual address of the block with non-zero reference count (which is guaranteed to exist in the reference count filter) is searched for in the reference count filter, and then it is deleted.

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		The delete operation on the reference count filter is implemented by the delete operation on the underlying hash table implementing the reference count filter, again as below described. *Id.* at 7:27-39. As mentioned above, after an insert operation is suspended due to a full reference count filter, the MMU needs to make a room in the reference count filter so that the suspended insert operation can be resumed and completed. The MMU 40 makes the necessary room by performing a reconciliation operation, box 62. It sends a special command to the CPU 12 called "Dump-pointers," box 64. In response, the CPU 12 sends the contents of all binding registers that contain pointers (which are virtual addresses) to the MMU. The set of these pointers is called the "Dumped-out" set, which is received by the MMU, box 65. The pointers in the Dumped-out set indicate the block which have references originating at binding registers. Of course, the size of the dumped-out set cannot exceed the number of binding registers. The reconciliation operation is guaranteed to create a room in the reference count filter, as the size of the reference count filter is greater than the number of binding registers. Therefore, there must be at least one virtual address in the reference count filter which is not in any binding register. *Id.* at 7:40-60. The MMU reconciles the state of the reference count filter with the set of Dumped-out pointers, by executing the following operations. (1) Each pointer from the dumped-out set is attempted to be located by the MMU within the virtual address contained in the reference count filter, box 67. If the pointer exists in the reference count filter, the MMU marks the pointer in the reference count filter, box 68. The pointers in the reference count filter, thus marked, indicate blocks that are still accessible and hence are not garbage. All unmarked pointers in the reference count filter, therefore, indicate garbage blocks. (2) The

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		unmarked pointers are evacuated from the reference count filter and stored in another data structure, called the "garbage buffer" (not shown), box 69, which essentially holds pointers to garbage blocks. A background process not described herein in the MMU operates on the garbage buffer to reclaim the garbage blocks. As soon as the unmarked pointers are evacuated from the reference count filter to the garbage buffer, the reconciliation operation on the reference count filter is over, and the regular operation is resumed. As a result of the reconciliation operation, the state of the reference count filter has been reconciled with the state of binding registers, and pointers to all garbage blocks have been evacuated from the reference count filter. *Id.* at 7:61-8:18.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Thatte discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Thatte also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Figure 6 displays a block diagram showing the steps of maintaining the reference count filter. *Id.* at 5:10-12. Each time a block is allocated or a pointer to a block is destroyed the reference count filter is checked to see if the reference count filter is full. *Id.* at 7:1-26, Figure 6. If the reference count filter is full, then a reconciliation operation is performed where expired records are removed. *Id.*

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		 49 START ALLOCATION OF BLOCKS DESTRUCTION OF POINTER CREATION OF POINTER ALLOCATE BLOCK 50 52 DETECT RC = Ø 70 DETECT RC CHANGE FROM ZERO TO ONE 72 DELETE VIRTUAL ADDRESS OF BLOCK FROM REF. CNT. FILTER 60 IS REF. CNT. FILTER FULL? YES NO 62 64 COMMAND CPU TO DUMP POINTERS 51 INSERT THE VIRTUAL ADDRESS OF BLOCK IN REF. CNT. FILTER 65 RECEIVE DUMPED POINTERS 67 LOCATE POINTERS IN REFERENCE COUNT FILTER Fig. 6 68 MARK LOCATED POINTERS 69 EVACUATE UNLOCATED POINTERS TO "GARBAGE BUFFER" RECONCILIATION OPERATION *Id.* at Figure 6. Beginning from a start position 49, three possible block affecting operations may be performed. The first is the allocation of a block [i.e. insert], the second is the destruction of a pointer to a block [i.e. delete],

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		and the third is the creation of a pointer to a block. Accordingly, when a new block is allocated by the MMU 40 in response to an "Allocate" command from the CPU 12, box 50, the new block starts its life with a zero reference count. Therefore, in accordance with the invention, the virtual address of a newly allocated block is inserted into the reference count filter 25, box 51, assuming that there is a place in the reference count filter for insertion, i.e., that the reference count filter is not full. Similarly, when the reference count of a block drops to zero, box 52, the virtual address of the block is inserted in the reference count filter 25, again assuming that there is a place in the reference count filter for [insertion]. The insert operation on the reference count filter is implemented by the insert operation on the underlying hash table, as below described. *Id.* at 7:1-20. If the reference count filter is determined to be full, box 60, the MMU suspends the insertion operation and performs a reconciliation operation, box 62, on the reference count filter, as described below, to create a room in the reference count filter so that the suspended insertion operation can be completed. *Id.* at 7:21-26. When the reference count of a block goes up from zero to one, box 70, the virtual address of the block is deleted from the reference count filter, box 72. The deletion operation is necessary because a block with a non-zero reference count must not stay in the reference count filter. To accomplish the deletion, first the virtual address of the block with non-zero reference count (which is guaranteed to exist in the reference count filter) is searched for in the reference count filter, and then it is deleted. The delete operation on the reference count filter is implemented by the delete operation on the underlying hash table implementing the reference count filter, again as below described. *Id.* at 7:27-39.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		As mentioned above, after an insert operation is suspended due to a full reference count filter, the MMU needs to make a room in the reference count filter so that the suspended insert operation can be resumed and completed. The MMU 40 makes the necessary room by performing a reconciliation operation, box 62. It sends a special command to the CPU 12 called "Dump-pointers," box 64. In response, the CPU 12 sends the contents of all binding registers that contain pointers (which are virtual addresses) to the MMU. The set of these pointers is called the "Dumped-out" set, which is received by the MMU, box 65. The pointers in the Dumped-out set indicate the block which have references originating at binding registers. Of course, the size of the dumped-out set cannot exceed the number of binding registers. The reconciliation operation is guaranteed to create a room in the reference count filter, as the size of the reference count filter is greater than the number of binding registers. Therefore, there must be at least one virtual address in the reference count filter which is not in any binding register. *Id.* at 7:40-60. The MMU reconciles the state of the reference count filter with the set of Dumped-out pointers, by executing the following operations. (1) Each pointer from the dumped-out set is attempted to be located by the MMU within the virtual address contained in the reference count filter, box 67. If the pointer exists in the reference count filter, the MMU marks the pointer in the reference count filter, box 68. The pointers in the reference count filter, thus marked, indicate blocks that are still accessible and hence are not garbage. All unmarked pointers in the reference count filter, therefore, indicate garbage blocks. (2) The unmarked pointers are evacuated from the reference count filter and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		stored in another data structure, called the "garbage buffer" (not shown), box 69, which essentially holds pointers to garbage blocks. A background process not described herein in the MMU operates on the garbage buffer to reclaim the garbage blocks. As soon as the unmarked pointers are evacuated from the reference count filter to the garbage buffer, the reconciliation operation on the reference count filter is over, and the regular operation is resumed. As a result of the reconciliation operation, the state of the reference count filter has been reconciled with the state of binding registers, and pointers to all garbage blocks have been evacuated from the reference count filter. *Id.* at 7:61-8:18.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Thatte discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Thatte discloses a reconciliation operation that is only performed if the reference counter is full. *Id.* at 7:21-26, 7:40-8:18, Figure 7. This dynamic decision is clearly shown at 60 of Figure 6. *Id.* If the reference counter is not full, then Thatte dynamically determines that the maximum number of records to delete is zero. *Id.* If the reference counter is full, then Thatte dynamically determines that the maximum number of records to delete is all of the garbage in reference count filter. *Id.* Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Thatte to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		of ordinary skill in the art would have been motivated to combine the system disclosed in Thatte with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Thatte can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Thatte is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Alternatively, Thatte combined with Dirks discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		reference in its entirety. As summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*,

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)**
		where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Thatte and Dirks relate to memory management systems and deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other memory management systems such as Thatte. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)**
		removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks’ deletion decision procedure with Thatte’s memory management technique would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks’ dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Thatte’s method of memory management and would have seen the benefits of doing so. For example, Thatte notes that because “the CPU is stopped during the reconciliation operation” there could be “performance degradation” and “it is desirable to reduce the time for reconciliation as much as possible.” See Thatte, col. 8:19-22. One of ordinary skill in the art would have recognized that Dirks’ method of dynamically determining a maximum number of entries to examine could achieve this goal of limiting the performance degradation in Thatte’s method. To the extent that dynamically determining a maximum number of expired records is not disclosed by Thatte in combination with Dirks, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Thatte. For example, both Linux 2.0.1 and Thatte describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds

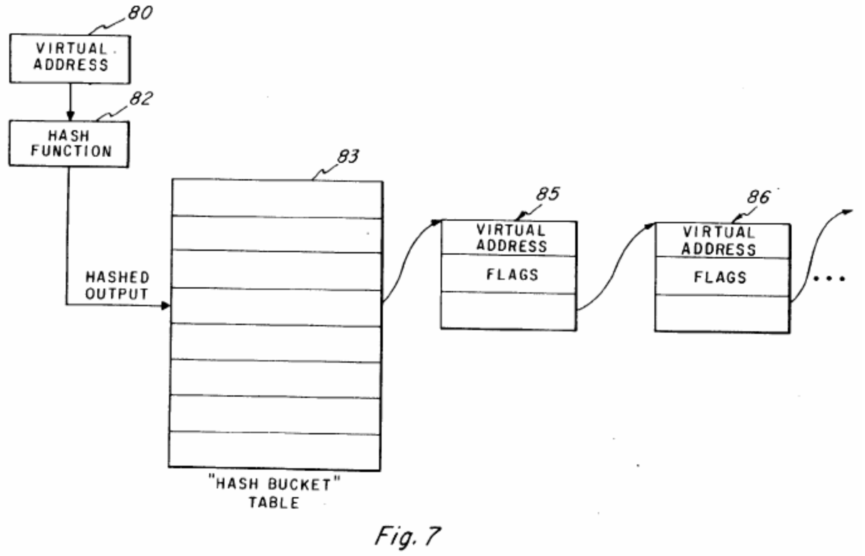
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Thatte discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Thatte also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Thatte discloses: A method and apparatus for managing a block oriented memory of the type in which each memory block has an associated reference count representing the number of pointers to it from other memory blocks and itself. Efficient and cost-effective implementation of reference counting alleviates the need for frequent garbage collection, which is an expensive operation. The apparatus includes a hash table into which the virtual addresses of blocks of memory which equal zero are maintained. When the reference count of a block increases from zero, its virtual address is removed from the table. When the reference count of a block decreases to zero, its virtual address is inserted into the table. When the table is full, a reconciliation operation is performed to identify those addresses which are contained in a set of binding registers associated with the CPU, and any address not contained in the binding registers are evacuated into a garbage buffer for subsequent garbage collection operations. The apparatus can be implemented by a cache augmented by the hash table, providing a

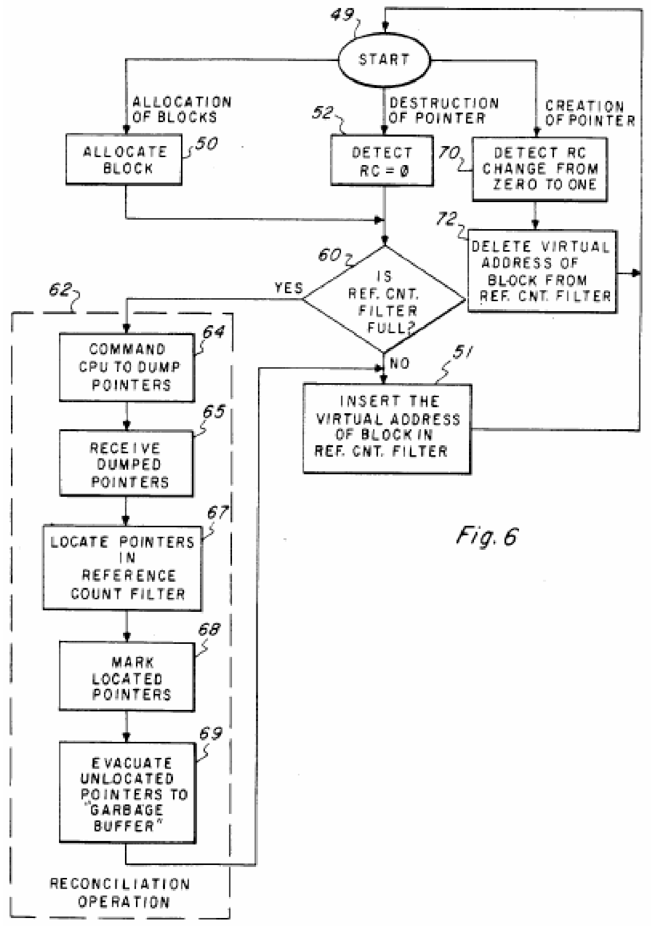
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		back-up store for the cache. Thatte at Abstract. In accordance with a broad aspect of the invention, an apparatus is provided for managing a block oriented memory of the type in which each memory block has an associated reference count representing the number of pointers to it from other memory blocks and itself. The apparatus includes means for implementing a data structure storing virtual addresses, which can be implemented by a hash table or the like. Means are also provided for performing insert and delete operations on the data structure, means for inserting in the data structure the virtual address of each block of memory which has a reference count of zero, and means for deleting from the data structure the virtual address of each block of memory whose reference count changes from zero to one. In one aspect of the invention, the apparatus further includes means for performing a reconciliation operation on the data structure when it is full, the reconciliation means including means for obtaining a dump of pointers in binding registers, means for comparing the pointers of the pointer dump with the virtual addresses contained in the data structure, and means for deleting any virtual addresses contained in the data structure not in the pointer dump. *Id.* at 5:40-62. The method and apparatus for reference count management assistance disclosed in Thatte is referred to as a "reference count filter," which acts conceptually as a filter to control reference count management. *Id.* at 5:66-68. The reference count filter stores virtual addresses of blocks whose reference counts have dropped to zero, but which may have pointers to them at binding registers. *Id.* at 6:51-57. The method and apparatus also contains a memory management unit (MMU)

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		40 which provides memory management functions. *Id.* at 6:4-24. "[T]he MMU 40 is responsible for reference count management; that is, it increments and decrements reference counts of the referent blocks, when pointers to these blocks are created or destroyed in memory cells. The MMU 40 is also responsible for reclaiming inaccessible blocks." *Id.* at 6:25-31. "The MMU 40 maintains information about blocks which have zero reference counts, but which may have pointers to them originating at the binding registers. This information is conveniently recorded and maintained in a data structure, [such as a hash table as shown in Figure 7], in the form of virtual-addresses of such blocks." *Id.* at 6:36-41
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Thatte discloses accessing a linked list of records. Thatte also discloses accessing a linked list of records having same hash address. For example, Thatte's reference count filter is preferably implemented in a "hash table that can efficiently support the insert, delete, and reconcile operations. Therefore, the basic search operation on the hash table must be quite fast." *Id.* at 8:39-43. "The hash table implementation of the reference count filter, in accordance with the invention, is illustrated in FIG. 7." *Id.* at 8:49-81. Figure 7 displays a virtual address 80 that is applied to a hash function 92. *Id.* at Figure 7, 8:39-62. The hashed output is then inserted into a hash bucket 83 which comprises a linked list 85, 86. *Id.*

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		 80 VIRTUAL ADDRESS 82 HASH FUNCTION 83 HASHED OUTPUT 85 VIRTUAL ADDRESS FLAGS 86 VIRTUAL ADDRESS FLAGS "HASH BUCKET" TABLE Fig. 7 *Id.* at Figure 6.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Thatte discloses identifying at least some of the automatically expired ones of the records. For example, The records that are contained within the reference count filter automatically expire. *Id.* at 7:40-8:18. When the reference count filter is full a reconciliation operation 62 is performed. *Id.* The reconciliation operation removes all of the garbage blocks, i.e. virtual addresses stored in the hash table that are no longer referenced externally. *Id.* The MMU 40 makes the necessary room by performing a reconciliation operation, box 62. It sends a special command to the CPU 12 called "Dump-pointers," box 64. In response, the CPU 12 sends the contents of all binding registers that contain pointers (which

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		are virtual addresses) to the MMU. The set of these pointers is called the "Dumped-out" set, which is received by the MMU, box 65. The pointers in the Dumped-out set indicate the block which have references originating at binding registers. Of course, the size of the dumped-out set cannot exceed the number of binding registers. The reconciliation operation is guaranteed to create a room in the reference count filter, as the size of the reference count filter is greater than the number of binding registers. Therefore, there must be at least one virtual address in the reference count filter which is not in any binding register. *Id.* at 7:44-60. The MMU reconciles the state of the reference count filter with the set of Dumped-out pointers, by executing the following operations. (1) Each pointer from the dumped-out set is attempted to be located by the MMU within the virtual address contained in the reference count filter, box 67. If the pointer exists in the reference count filter, the MMU marks the pointer in the reference count filter, box 68. The pointers in the reference count filter, thus marked, indicate blocks that are still accessible and hence are not garbage. All unmarked pointers in the reference count filter, therefore, indicate garbage blocks. (2) The unmarked pointers are evacuated from the reference count filter and stored in another data structure, called the "garbage buffer" (not shown), box 69, which essentially holds pointers to garbage blocks. A background process not described herein in the MMU operates on the garbage buffer to reclaim the garbage blocks. As soon as the unmarked pointers are evacuated from the reference count filter to the garbage buffer, the reconciliation operation on the reference count filter is over, and the regular operation is resumed. As a result of the reconciliation operation, the state of the reference count filter has been reconciled with the state of binding registers, and pointers to all garbage blocks have been evacuated from the reference count filter. *Id.* at 7:61-8:18.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Thatte discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Figure 6 displays a block diagram showing the steps of maintaining the reference count filter. *Id.* at 5:10-12. Each time a block is allocated or a pointer to a block is destroyed the reference count filter is checked to see if the reference count filter is full. *Id.* at 7:1-26, Figure 6. If the reference count filter is full, then a reconciliation operation is performed where expired records are removed. *Id.*

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		 49 START ALLOCATION OF BLOCKS DESTRUCTION OF POINTER CREATION OF POINTER 52 ALLOCATE BLOCK 50 DETECT RC = Ø 70 DETECT RC CHANGE FROM ZERO TO ONE 72 DELETE VIRTUAL ADDRESS OF BLOCK FROM REF. CNT. FILTER 60 62 YES IS REF. CNT. FILTER FULL? NO 64 COMMAND CPU TO DUMP POINTERS 51 INSERT THE VIRTUAL ADDRESS OF BLOCK IN REF. CNT. FILTER 65 RECEIVE DUMPED POINTERS 67 LOCATE POINTERS IN REFERENCE COUNT FILTER Fig. 6 68 MARK LOCATED POINTERS 69 EVACUATE UNLOCATED POINTERS TO "GARBAGE BUFFER" RECONCILIATION OPERATION *Id.* at Figure 6. Beginning from a start position 49, three possible block affecting operations may be performed. The first is the allocation of a block [i.e. insert], the second is the destruction of a pointer to a block [i.e. delete],

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		and the third is the creation of a pointer to a block. Accordingly, when a new block is allocated by the MMU 40 in response to an "Allocate" command from the CPU 12, box 50, the new block starts its life with a zero reference count. Therefore, in accordance with the invention, the virtual address of a newly allocated block is inserted into the reference count filter 25, box 51, assuming that there is a place in the reference count filter for insertion, i.e., that the reference count filter is not full. Similarly, when the reference count of a block drops to zero, box 52, the virtual address of the block is inserted in the reference count filter 25, again assuming that there is a place in the reference count filter for [insertion]. The insert operation on the reference count filter is implemented by the insert operation on the underlying hash table, as below described. *Id.* at 7:1-20. If the reference count filter is determined to be full, box 60, the MMU suspends the insertion operation and performs a reconciliation operation, box 62, on the reference count filter, as described below, to create a room in the reference count filter so that the suspended insertion operation can be completed. *Id.* at 7:21-26. When the reference count of a block goes up from zero to one, box 70, the virtual address of the block is deleted from the reference count filter, box 72. The deletion operation is necessary because a block with a non-zero reference count must not stay in the reference count filter. To accomplish the deletion, first the virtual address of the block with non-zero reference count (which is guaranteed to exist in the reference count filter) is searched for in the reference count filter, and then it is deleted. The delete operation on the reference count filter is implemented by the delete operation on the underlying hash table implementing the reference count filter, again as below described. *Id.* at 7:27-39.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		As mentioned above, after an insert operation is suspended due to a full reference count filter, the MMU needs to make a room in the reference count filter so that the suspended insert operation can be resumed and completed. The MMU 40 makes the necessary room by performing a reconciliation operation, box 62. It sends a special command to the CPU 12 called "Dump-pointers," box 64. In response, the CPU 12 sends the contents of all binding registers that contain pointers (which are virtual addresses) to the MMU. The set of these pointers is called the "Dumped-out" set, which is received by the MMU, box 65. The pointers in the Dumped-out set indicate the block which have references originating at binding registers. Of course, the size of the dumped-out set cannot exceed the number of binding registers. The reconciliation operation is guaranteed to create a room in the reference count filter, as the size of the reference count filter is greater than the number of binding registers. Therefore, there must be at least one virtual address in the reference count filter which is not in any binding register. *Id.* at 7:40-60. The MMU reconciles the state of the reference count filter with the set of Dumped-out pointers, by executing the following operations. (1) Each pointer from the dumped-out set is attempted to be located by the MMU within the virtual address contained in the reference count filter, box 67. If the pointer exists in the reference count filter, the MMU marks the pointer in the reference count filter, box 68. The pointers in the reference count filter, thus marked, indicate blocks that are still accessible and hence are not garbage. All unmarked pointers in the reference count filter, therefore, indicate garbage blocks. (2) The unmarked pointers are evacuated from the reference count filter and stored in another data structure, called the "garbage buffer" (not shown), box 69, which essentially holds pointers to garbage blocks. A

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		background process not described herein in the MMU operates on the garbage buffer to reclaim the garbage blocks. As soon as the unmarked pointers are evacuated from the reference count filter to the garbage buffer, the reconciliation operation on the reference count filter is over, and the regular operation is resumed. As a result of the reconciliation operation, the state of the reference count filter has been reconciled with the state of binding registers, and pointers to all garbage blocks have been evacuated from the reference count filter. *Id.* at 7:61-8:18.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Thatte discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, following the reconciliation operation 62 the next step is to insert the virtual address of the block in the reference count filter 51. *Id.* at 7:21-26, 7:40-8:18, Figure 7.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Thatte discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, Thatte discloses a reconciliation operation that is only performed if the reference counter is full. *Id.* This dynamic decision is clearly shown at 60 of Figure 6. *Id.* If the reference counter is not full, then Thatte dynamically determines that the maximum number of records to delete is zero. *Id.* If the reference counter is full, then Thatte dynamically determines that the maximum number of records to delete is all of the garbage in reference count filter. *Id.* Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Thatte to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Thatte with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Thatte can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Thatte is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Alternatively, Thatte combined with Dirks discloses the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed For example, Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)
		Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. As summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The

EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)**
		predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Thatte and Dirks relate to memory management systems and deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other memory management systems such as Thatte. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Thatte's memory management technique would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Thatte's method of memory management and would have seen the benefits of doing so. For example, Thatte notes that because "the CPU is stopped during the reconciliation operation" there could be "performance degradation" and "it is desirable to reduce the time for reconciliation as much as possible." See Thatte, col. 8:19-22. One of ordinary skill in the art would have recognized that Dirks' method of dynamically determining a maximum number of entries to examine could achieve this goal of limiting the performance degradation in Thatte's method. To the extent that dynamically determining a maximum number of expired records is not disclosed by Thatte in combination with Dirks, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Thatte. For example, both Linux 2.0.1 and Thatte describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,695,949 to Thatte et al. ("Thatte")**
		`RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)
		rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock’s proposed claim constructions, the records removed by the function rt_garbage_collect_1 are “expired” records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record’s last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record’s reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function

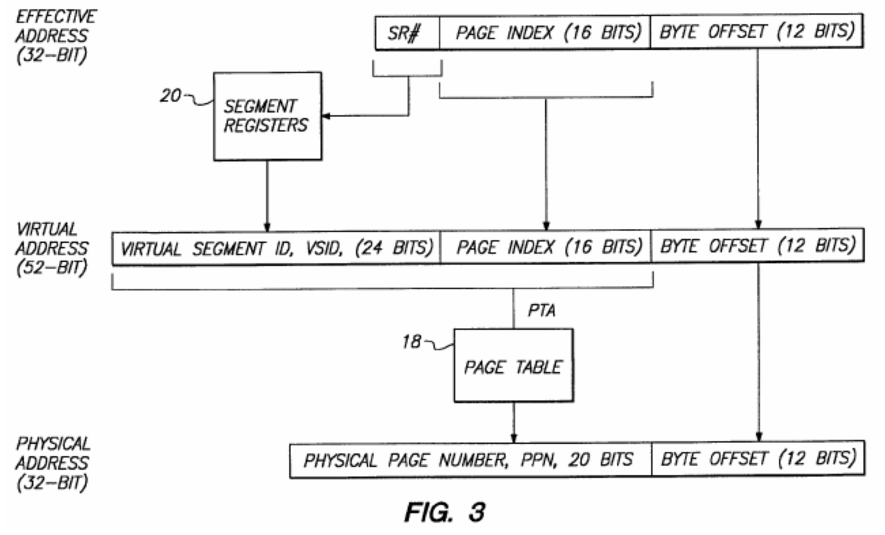
EXHIBIT B-1

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,695,949 to Thatte et al. (“Thatte”)
		`rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Dirks discloses an information storage and retrieval system. For example, Dirks discloses "the management of memory in a computer system, and more particularly to the allocation of address space in a virtual memory system for a computer." *Dirks*, "Method for Allocation of Address Space In A Virtual Memory System." U.S. Patent No. 6,119,214 (issued Sept. 12, 2000) at 1:5-8. In virtual memory technology, > the addresses that are assigned for use by individual programs are distinct from the actual physical addresses of the main memory. The addresses which are allocated to the programs are often referred to as "logical" or "virtual" or "effective" addresses, to distinguish them from the "physical" addresses of the main memory. Whenever a program requires access to memory, it makes a call to a logical address within its assigned address space. A memory manager associates this logical address with a physical address in the main memory, where the information called for by the program is actually stored. The identification of each physical address that corresponds to a logical address is commonly stored in a data structure known as a page table. This term is derived from the practice of dividing the memory into individually addressable blocks known as "pages." *Id.* at 1:33-50. The relationship between the virtual address, page table, and physical address is shown in Figure 3 reproduced below. *Id.* at Figure 3.

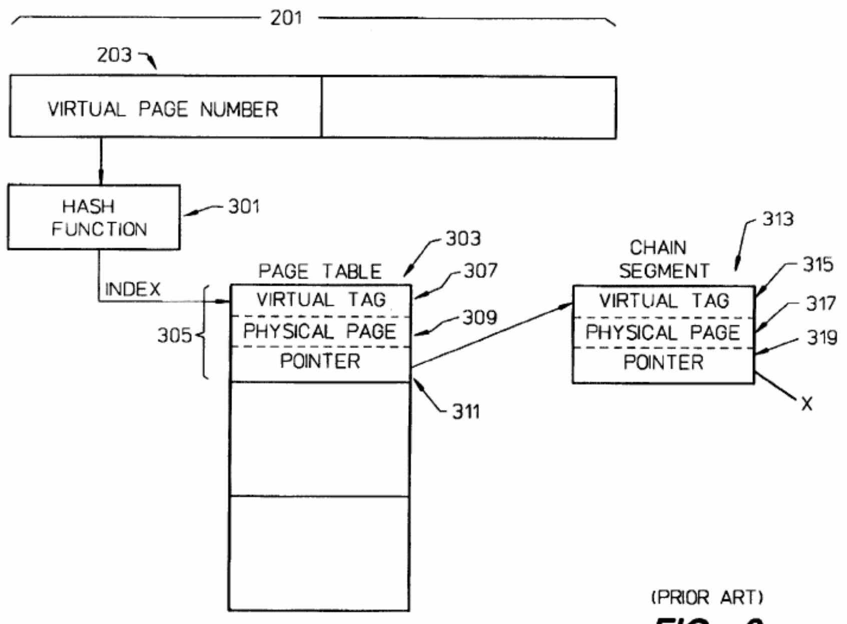
EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 EFFECTIVE ADDRESS (32-BIT) SR# PAGE INDEX (16 BITS) BYTE OFFSET (12 BITS) 20 SEGMENT REGISTERS VIRTUAL ADDRESS (52-BIT) VIRTUAL SEGMENT ID, VSID, (24 BITS) PAGE INDEX (16 BITS) BYTE OFFSET (12 BITS) PTA 18 PAGE TABLE PHYSICAL ADDRESS (32-BIT) PHYSICAL PAGE NUMBER, PPN, 20 BITS BYTE OFFSET (12 BITS) FIG. 3 *Id.*
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Dirks discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Dirks also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, the virtual segment identifiers (VSID) are stored in the page table by use of a hashing function. *Id.* at 5:10-31. "Through the use of the hashing function, the page table entries are efficiently distributed within the page

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		table." *Id.* at 5:25-27. For example, Dirks inherently discloses that page table entry groups, which are comprised of page table entries, are stored in a linked list. *See id.* at 9:32-46. > For example, in some page tables, a suitable number of entries are grouped together, and the addressing of entries is done by groups. In other words, the page table address PTA that results from the operation of the hashing function is the physical address of a page table entry group. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id at 9:36-46.* Because each page table entry stored in a page table entry group has the same hash address, the page table entries are being stored in a linked list; such an inherent characteristic necessarily flows from the teachings of the applied prior art. *See id.* To the extent that Bedrock argues that Dirks does not anticipate Claims 1 – 8 because the page table entry groups are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the page table entries of a page table group in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. The '120 patent

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		at 1:34-2:6. Thus, Dirks and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as shown in Figure 3 of Morris, it was well known in the prior art to have page tables entries distributed by a hash function, such as those in Dirks, and then store the page table entries in a hash table using linked lists or external chaining. U.S. Pat. No. 5,724,538 to Morris et al. ("Morris") at 3:54-4:24, Figure 3.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 201 203 VIRTUAL PAGE NUMBER HASH FUNCTION 301 INDEX PAGE TABLE 303 307 VIRTUAL TAG PHYSICAL PAGE POINTER 305 309 311 CHAIN SEGMENT 313 315 317 319 X (PRIOR ART) FIG. 3 *Id.* at Figure 3. As described by Morris: FIG. 3 illustrates the process of retrieving the physical page information given the virtual page number as would be required to update the TLB after a TLB miss. As described above, the virtual to physical mappings are maintained in a page table. For translating a given virtual address to a physical address, one

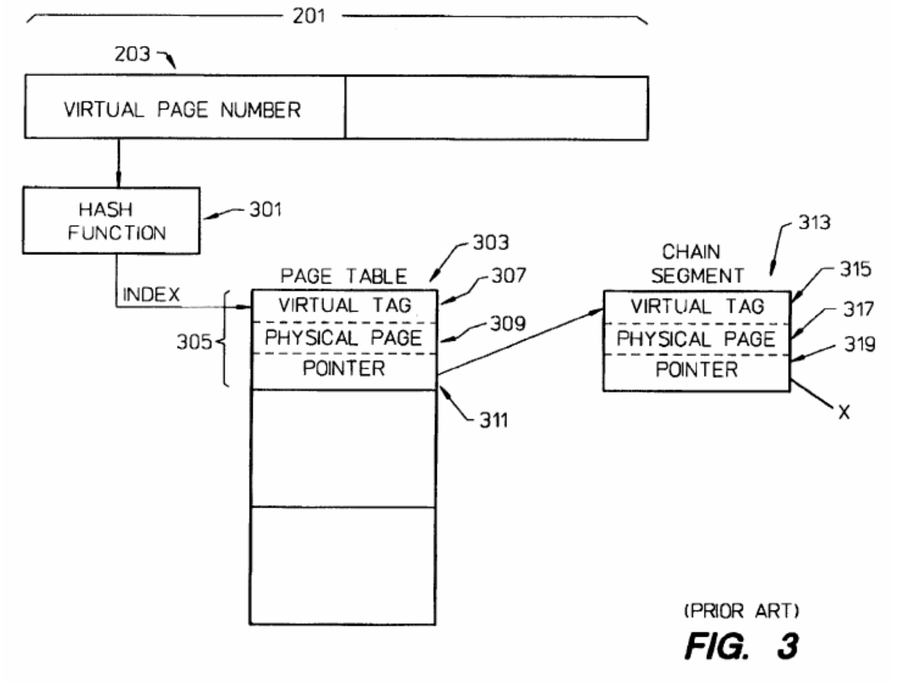
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		approach is to perform a many-to-one function (hash) on the virtual address to form an index into the page table. This gives a pointer to a linked list of entries. These entries are then searched for a match. To determine a match, the virtual page number is compared to an entry in the page table (virtual tag). If the two are equal, that page table entry provides the physical address translation. *Id.* at 3:54-65. In the example illustrated, a hash function 301 is performed on the virtual page number 203 to form an index. This index is an offset into the page table 303. As shown, the index is 0, that is, the index points to the first entry 305 in the page table. Each entry in the page table consists of multiple parts but typically contains at least a virtual tag 307, a physical page 309 and a pointer 311. If the virtual page number 203 equals the virtual tag 307, then physical page 309 gives the physical (real) memory page address desired. If the virtual tag does not match, then the pointer 311 points to a chain of entries in memory which contain virtual to physical translation information. The additional information contained in the chain is needed as more than one virtual page number can hash to the same page table entry. *Id.* at 3:66-4:12. As shown, pointer 311 points to a chain segment 313. This chain segment contains the same type of information as the page table. As before, the virtual page number 203 is compared to the next virtual tag 315 to see if there is a match. If a match occurs then the associated physical page 317 gives the address

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		of the physical memory page desired. If a match does not occur, then the pointer 319 is examined to locate the next chain segment, if any. If the pointer 319 does not point to another chain segment, as shown, then a page fault has occurred. A page fault software program is then used, as described in association with FIG. 1, to update the page table. *Id.* at 4:12-24. As both Dirks and Morris disclose systems and methods for allocating memory address controls using page table entries that are stored in hash tables, one of ordinary skill in the art would have understood how to combine the hashed page table with linked lists taught in Morris with Dirks. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks with Morris would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Dirks' page table being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and understood by one of ordinary skill in the art to the system disclosed in the admitted prior art, and would have seen the benefits of doing so. One possible benefit, for example, is hash table collision resolution. Dirks discloses VSIDs that automatically expire. Dirks at 6:24-30. At some point in time, the VSID becomes inactive. This can occur, for example, when a thread terminates. In the inactive state, pages within the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		virtual address range of the VSID are still mapped to the page table. However, the data contained in the associated pages of physical memory is no longer being accessed by the CPU. *Id.* Thus, the VSIDs are expired or obsolete. *Id.*
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Dirks discloses a record search means utilizing a search key to access the linked list. Dirks also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Dirks discloses hash table search means such as searching, creation, deletion, and sweeping. *Id.* at 5:31-47, 5:66-6:15, 5:39-44, 9:39-47. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id.* at 9:39-47. For example, Dirks inherently discloses that page table entry groups, which are comprised of page table entries, are stored in a linked list. *See id.* at 9:32-46. For example, in some page tables, a suitable number of entries are grouped together, and the addressing of entries is done by groups. In other words, the page table address PTA that results from the operation of the hashing function is the physical address of a page table entry group. Whenever a call is made to a particular virtual address, the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id.* Because each page table entry stored in a page table entry group has the same hash address, the page table entries are being stored in a linked list; such an inherent characteristic necessarily flows from the teachings of the applied prior art. *See id.* To the extent that Bedrock argues that Dirks does not anticipate Claims 1 – 8 because the page table entry groups are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the page table entries of a page table group in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. The '120 patent at 1:34-2:6. Thus, Dirks and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as shown in Figure 3 of Morris, it was well known in the prior art to have page tables entries distributed by a hash function, such as those in Dirks, and then store the page table entries in a hash table using linked lists or external chaining. Morris at 3:54-4:24, Figure 3.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 201 203 VIRTUAL PAGE NUMBER HASH FUNCTION 301 INDEX PAGE TABLE 303 307 VIRTUAL TAG PHYSICAL PAGE POINTER 309 305 311 CHAIN SEGMENT 313 315 317 319 X (PRIOR ART) FIG. 3 *Id.* at Figure 3. As described by Morris: > FIG. 3 illustrates the process of retrieving the physical page information given the virtual page number as would be required to update the TLB after a TLB miss. As described above, the virtual to physical mappings are maintained in a page table. For translating a given virtual address to a physical address, one

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		approach is to perform a many-to-one function (hash) on the virtual address to form an index into the page table. This gives a pointer to a linked list of entries. These entries are then searched for a match. To determine a match, the virtual page number is compared to an entry in the page table (virtual tag). If the two are equal, that page table entry provides the physical address translation. *Id.* at 3:54-65. In the example illustrated, a hash function 301 is performed on the virtual page number 203 to form an index. This index is an offset into the page table 303. As shown, the index is 0, that is, the index points to the first entry 305 in the page table. Each entry in the page table consists of multiple parts but typically contains at least a virtual tag 307, a physical page 309 and a pointer 311. If the virtual page number 203 equals the virtual tag 307, then physical page 309 gives the physical (real) memory page address desired. If the virtual tag does not match, then the pointer 311 points to a chain of entries in memory which contain virtual to physical translation information. The additional information contained in the chain is needed as more than one virtual page number can hash to the same page table entry. *Id.* at 3:66-4:12. As shown, pointer 311 points to a chain segment 313. This chain segment contains the same type of information as the page table. As before, the virtual page number 203 is compared to the next virtual tag 315 to see if there is a match. If a match occurs then the associated physical page 317 gives the address

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		of the physical memory page desired. If a match does not occur, then the pointer 319 is examined to locate the next chain segment, if any. If the pointer 319 does not point to another chain segment, as shown, then a page fault has occurred. A page fault software program is then used, as described in association with FIG. 1, to update the page table. *Id.* at 4:12-24. As both Dirks and Morris disclose systems and methods for allocating memory address controls using page table entries that are stored in hash tables, one of ordinary skill in the art would have understood how to combine the hashed page table with linked lists taught in Morris with Dirks. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks with Morris would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Dirks' page table being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and understood by one of ordinary skill in the art to the system disclosed in the admitted prior art, and would have seen the benefits of doing so. One possible benefit, for example, is hash table collision resolution.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the	Dirks discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Dirks also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
records from the linked list when the linked list is accessed, and	linked list of records when the linked list is accessed, and	For example, as discussed above at some point VSIDs become inactive. Dirks at 6:24-30. These expired or inactive VSIDs are not removed "in one colossal step, for example after all of the free VSIDs have been allocated. Rather, the sweeping is carried out in an incremental, ongoing manner to avoid significant interruptions in the running of programs." *Id.* at 6:39-44. "More specifically, each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table."*Id.* at 3:14-18. Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24. A flowchart which depicts the operation of the address allocation portion of a memory manager, in accordance with the present invention, is illustrated in FIG. 6. Referring thereto, operation begins when a request for address space is generated (Step 10). This can occur when a thread is created, for example. In response thereto, the operating system checks the free list to determine whether a free VSID is available (Step 12). If so, a free VSID is allocated to the thread that generated the request (Step 14). In the case of an application, two or more VSIDs might be assigned. If no free VSID is available at Step 12, a failure status is returned (Step 18). In practice, however, such a situation should not occur, since the process of the present invention ensures that free VSIDs are always available. *Id.* at 7:66-8:12. After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. "[If for example], *k*=10000/500=20. For a given value of the index n, therefore, the system sweeps table entries $PT_{k(n-1)}$ to $PT_{(n)-1}$. Thus, during the first sweep, table entries PT_0 to PT_{19} are examined in the example given above." *Id.* at 8:34-37. "If a recycling sweep is already in progress, i.e. the response is affirmative at Step 20, the index n is incremented (Step 32) and a sweep of the next *k* entries is carried out. Thus, where n=2, entries $PT_{[20]}$ to PT_{39} will be examined." *Id.* at 8:38-41. The process continues in this manner, with *k* entries in the page table

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		being examined each time a free VSID is allocated. Each entry is examined to determine whether it contains a mapping for a VSID on the recycle list. If it does, that entry is removed from the page table. *Id.* at 8:42-46. After each sweep, the system determines whether all entries in the page table have been checked (Step 36). This situation will occur when n=x. Once all of the entries have been examined, the VSIDs in the recycle list are transferred to the free list (Step 38), yielding x new VSIDs that are available to be allocated. In addition, the recycle sweep flag RFLG is reset, and control is then returned to the program. If all of the entries in the page table have not yet been checked, i.e. the response is negative at Step 36, control is directly returned to the application program, at Step 24. *Id.* at 8:47-56.

EXHIBIT B-2

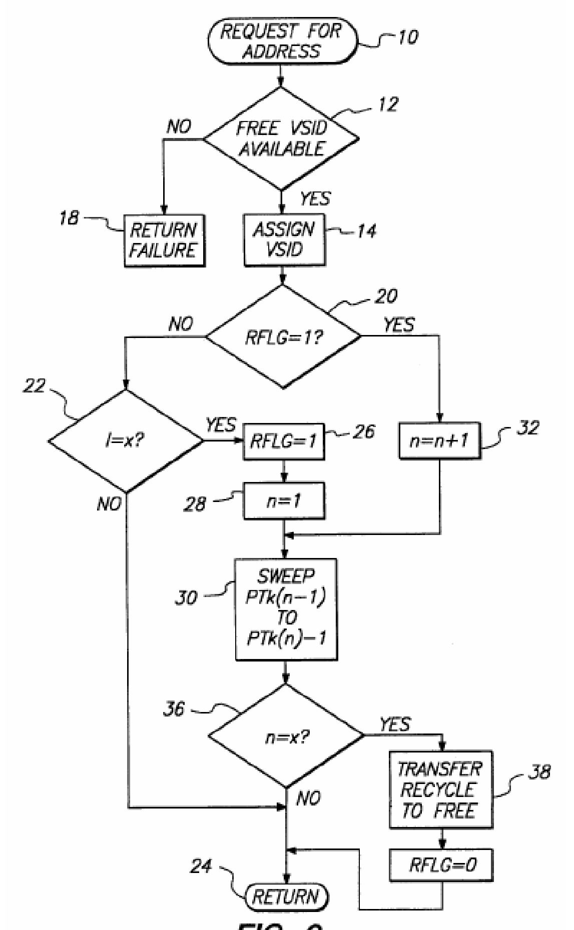
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 RFLG=1? 20 NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 SWEEP PTk(n−1) TO PTk(n)−1 30 36 n=x? YES TRANSFER RECYCLE TO FREE 38 NO RFLG=0 24 RETURN FIG. 6 *Id.* at Figure 6.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Dirks discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Dirks also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, as discussed above at some point VSIDs become inactive. *Id.* at 6:24-30. These expired or inactive VSIDs are not removed "in one colossal step, for example after all of the free VSIDs have been allocated. Rather, the sweeping is carried out in an incremental, ongoing manner to avoid significant interruptions in the running of programs." *Id.* at 6:39-44. "More specifically, each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table." *Id.* at 3:14-18. Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24. A flowchart which depicts the operation of the address allocation portion of a memory manager, in accordance with the present invention, is illustrated in FIG. 6. Referring thereto, operation begins when a request for address space is generated (Step 10). This can occur when a thread is created, for example. In response thereto, the operating system checks the free list to determine whether a free VSID is available (Step 12). If so, a free VSID is allocated to the thread that generated the request (Step 14). In the case of an application, two or more VSIDs might be assigned. If no free VSID is available at Step 12, a failure status is returned (Step 18). In

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		practice, however, such a situation should not occur, since the process of the present invention ensures that free VSIDs are always available. *Id.* at 7:66-8:12. After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. "[If for example], *k*=10000/500=20. For a given value of the index n, therefore, the system sweeps table entries $PT_{k(n-1)}$ to $PT_{(n)-1}$. Thus, during the first sweep, table entries PT_0 to PT_{19} are examined in the example given above." *Id.* at 8:34-37. "If a recycling sweep is already in progress, i.e. the response is affirmative at

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		Step 20, the index n is incremented (Step 32) and a sweep of the next *k* entries is carried out. Thus, where n=2, entries PT20 to PT_{39} will be examined." *Id.* at 8:38-41. The process continues in this manner, with *k* entries in the page table being examined each time a free VSID is allocated. Each entry is examined to determine whether it contains a mapping for a VSID on the recycle list. If it does, that entry is removed from the page table. *Id.* at 8:42-46. After each sweep, the system determines whether all entries in the page table have been checked (Step 36). This situation will occur when n=x. Once all of the entries have been examined, the VSIDs in the recycle list are transferred to the free list (Step 38), yielding x new VSIDs that are available to be allocated. In addition, the recycle sweep flag RFLG is reset, and control is then returned to the program. If all of the entries in the page table have not yet been checked, i.e. the response is negative at Step 36, control is directly returned to the application program, at Step 24. *Id.* at 8:47-56.

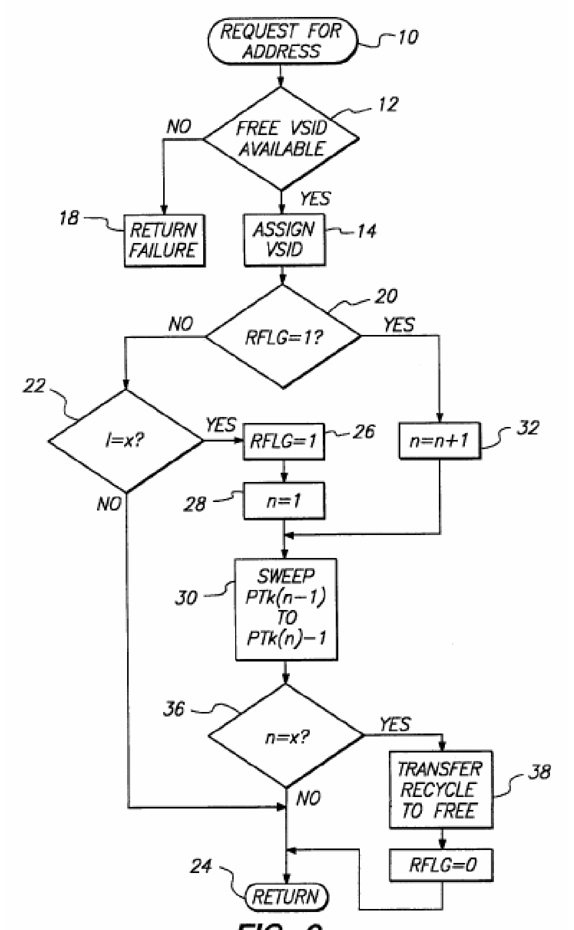
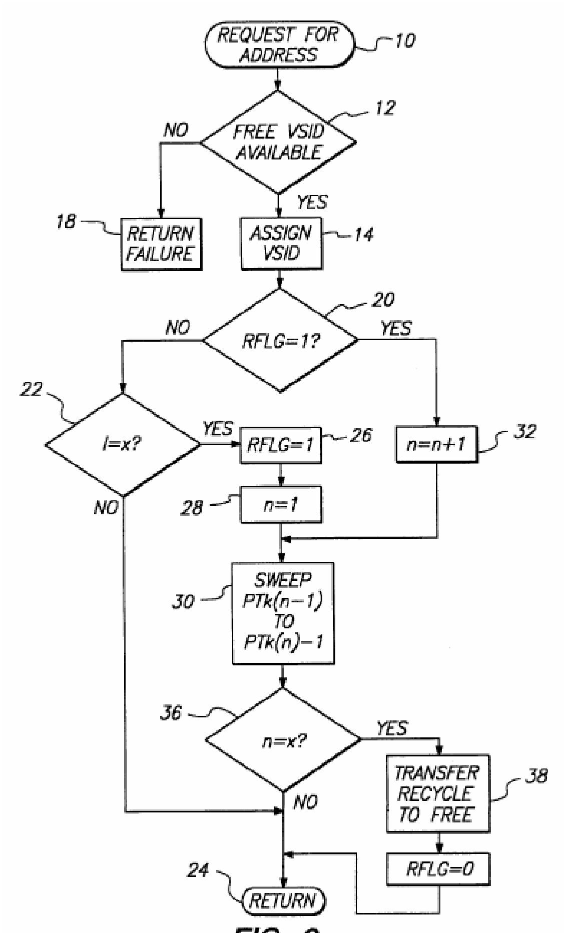
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 20 RFLG=1? NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 30 SWEEP PTk(n−1) TO PTk(n)−1 36 n=x? YES TRANSFER RECYCLE TO FREE 38 NO RFLG=0 24 RETURN FIG. 6 *Id.* at Figure 6.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Dirks discloses the information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, "each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table." *Id.* at 3:14-18. "Thus, rather than halting the operation of the computer for a considerable period of time to scan the entire page table when a logical address area is deleted, the memory manager of the present invention carries out a limited, time-bounded examination upon each address allocation." *Id.* at 3:25-29 Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24. In operation, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. *Id.* at 7:2-14.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 20 RFLG=1? NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 30 SWEEP PTk(n−1) TO PTk(n)−1 36 n=x? YES TRANSFER RECYCLE TO FREE 38 NO RFLG=0 24 RETURN FIG. 6 *Id.* at Figure 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		As shown in Figure 6, After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-37, 7:66-8:56. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dirks to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dirks with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dirks can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Dirks is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique

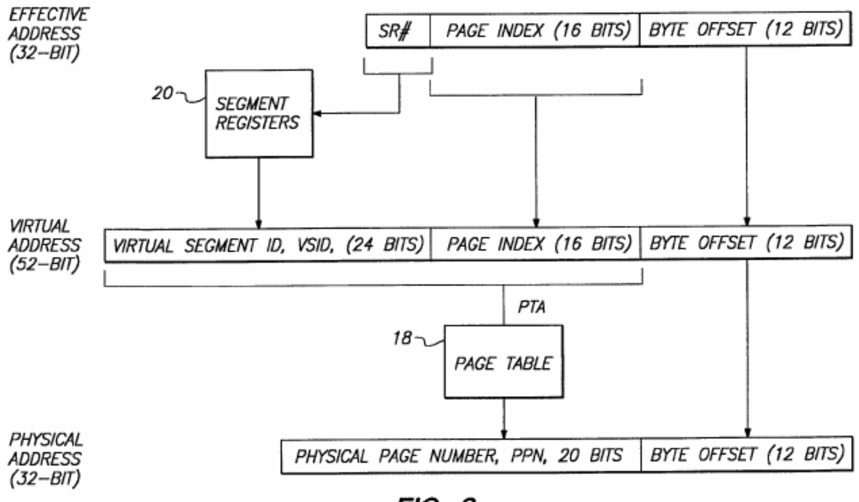
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Dirks, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Dirks. For example, both Linux 2.0.1 and Dirks describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1`

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last

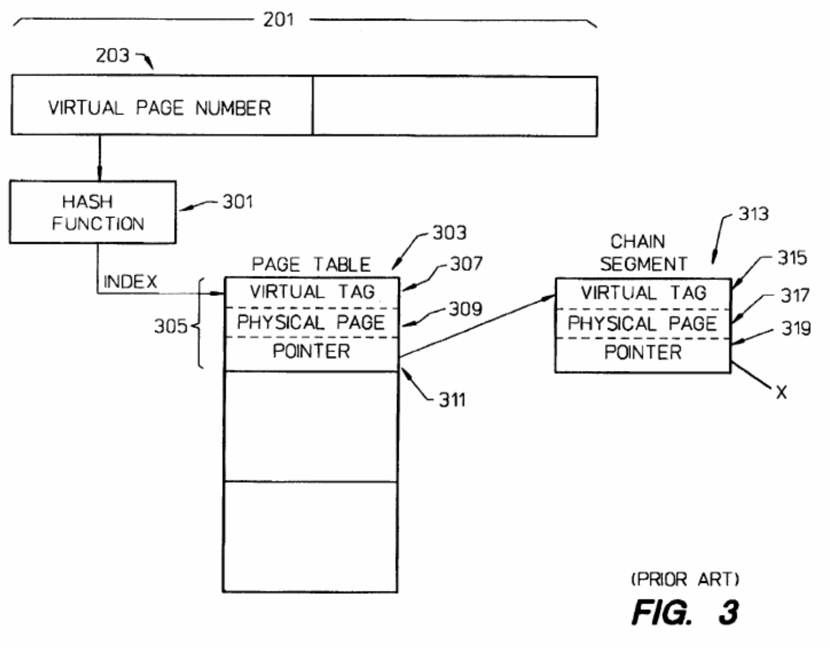
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records	To the extent the preamble is a limitation, Dirks discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Dirks also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Dirks discloses "the management of memory in a computer

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
	automatically expiring, the method comprising the steps of:	system, and more particularly to the allocation of address space in a virtual memory system for a computer." Dirks at 1:5-8. In virtual memory technology, the addresses that are assigned for use by individual programs are distinct from the actual physical addresses of the main memory. The addresses which are allocated to the programs are often referred to as "logical" or "virtual" or "effective" addresses, to distinguish them from the "physical" addresses of the main memory. Whenever a program requires access to memory, it makes a call to a logical address within its assigned address space. A memory manager associates this logical address with a physical address in the main memory, where the information called for by the program is actually stored. The identification of each physical address that corresponds to a logical address is commonly stored in a data structure known as a page table. This term is derived from the practice of dividing the memory into individually addressable blocks known as "pages". *Id.* at 1:33-50. The relationship between the virtual address, page table, and physical address is shown in Figure 3 reproduced below. *Id.* at Figure 3.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		 EFFECTIVE ADDRESS (32–BIT) SR# PAGE INDEX (16 BITS) BYTE OFFSET (12 BITS) 20 SEGMENT REGISTERS VIRTUAL ADDRESS (52–BIT) VIRTUAL SEGMENT ID, VSID, (24 BITS) PAGE INDEX (16 BITS) BYTE OFFSET (12 BITS) PTA 18 PAGE TABLE PHYSICAL ADDRESS (32–BIT) PHYSICAL PAGE NUMBER, PPN, 20 BITS BYTE OFFSET (12 BITS) FIG. 3 *Id.* The virtual segment identifiers (VSID) are stored in the page table by use of a hashing function. *Id.* at 5:10-31. "Through the use of the hashing function, the page table entries are efficiently distributed within the page table." *Id.* at 5:25-27. For example, Dirks inherently discloses that page table entry groups, which are comprised of page table entries, are stored in a linked list. *See id.* at 9:32-46. > For example, in some page tables, a suitable number of entries are grouped together, and the addressing of entries is done by groups. In other words, the page table address PTA that results from the operation

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		of the hashing function is the physical address of a page table entry group. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id. at 9:36-46.* Because each page table entry stored in a page table entry group has the same hash address, the page table entries are being stored in a linked list; such an inherent characteristic necessarily flows from the teachings of the applied prior art. *See id.* To the extent that Bedrock argues that Dirks does not anticipate Claims 1 – 8 because the page table entry groups are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the page table entries of a page table group in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. The '120 patent at 1:34-2:6. Thus, Dirks and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as shown in Figure 3 of Morris, it was well known in the prior art to have page tables entries distributed by a hash function, such as those in Dirks, and then store the page table entries in a hash table using linked lists or

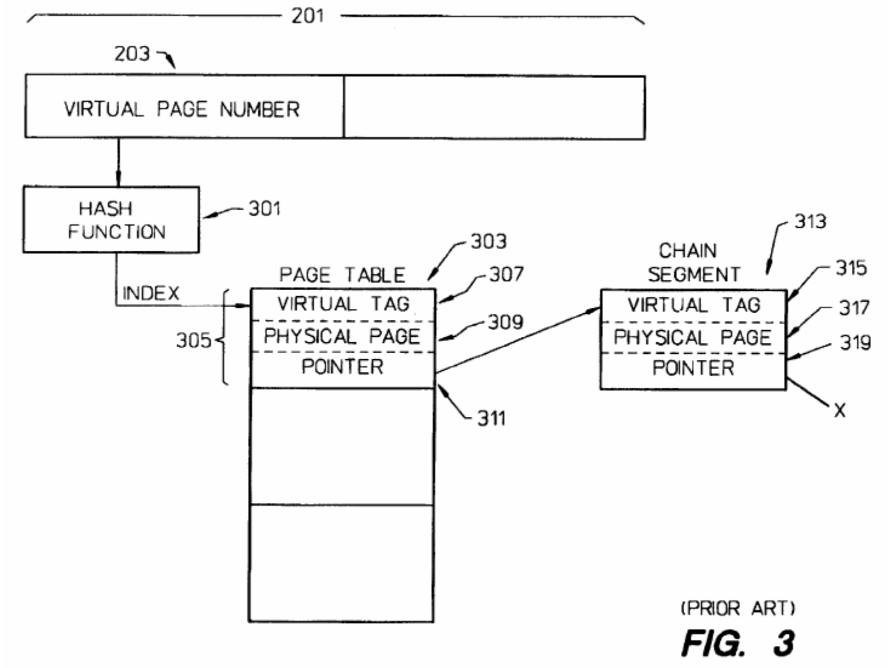
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		external chaining. Morris at 3:54-4:24, Figure 3.  201 203 VIRTUAL PAGE NUMBER HASH FUNCTION 301 INDEX PAGE TABLE 303 307 VIRTUAL TAG PHYSICAL PAGE POINTER 309 305 311 CHAIN SEGMENT 313 315 317 319 X (PRIOR ART) FIG. 3 *Id.* at Figure 3. As described by Morris: FIG. 3 illustrates the process of retrieving the physical page information given the virtual page number as would be required to update the TLB after a TLB miss. As described above, the virtual to physical mappings are maintained in a page table. For translating a given virtual address to a physical address, one

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		approach is to perform a many-to-one function (hash) on the virtual address to form an index into the page table. This gives a pointer to a linked list of entries. These entries are then searched for a match. To determine a match, the virtual page number is compared to an entry in the page table (virtual tag). If the two are equal, that page table entry provides the physical address translation. *Id.* at 3:54-65. In the example illustrated, a hash function 301 is performed on the virtual page number 203 to form an index. This index is an offset into the page table 303. As shown, the index is 0, that is, the index points to the first entry 305 in the page table. Each entry in the page table consists of multiple parts but typically contains at least a virtual tag 307, a physical page 309 and a pointer 311. If the virtual page number 203 equals the virtual tag 307, then physical page 309 gives the physical (real) memory page address desired. If the virtual tag does not match, then the pointer 311 points to a chain of entries in memory which contain virtual to physical translation information. The additional information contained in the chain is needed as more than one virtual page number can hash to the same page table entry. *Id.* at 3:66-4:12. As shown, pointer 311 points to a chain segment 313. This chain segment contains the same type of information as the page table. As before, the virtual page number 203 is compared to the next virtual tag 315 to see if there is a match. If a match occurs then the associated physical page 317 gives the address

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		of the physical memory page desired. If a match does not occur, then the pointer 319 is examined to locate the next chain segment, if any. If the pointer 319 does not point to another chain segment, as shown, then a page fault has occurred. A page fault software program is then used, as described in association with FIG. 1, to update the page table. *Id.* at 4:12-24. As both Dirks and Morris disclose systems and methods for allocating memory address controls using page table entries that are stored in hash tables, one of ordinary skill in the art would have understood how to combine the hashed page table with linked lists taught in Morris with Dirks. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks with Morris would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Dirks' page table being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and understood by one of ordinary skill in the art to the system disclosed in the admitted prior art, and would have seen the benefits of doing so. One possible benefit, for example, is hash table collision resolution. Dirks discloses VSIDs that automatically expire. Dirks at 6:24-30. At some point in time, the VSID becomes inactive. This can occur, for example, when a thread terminates. In the inactive state, pages within the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		virtual address range of the VSID are still mapped to the page table. However, the data contained in the associated pages of physical memory is no longer being accessed by the CPU. *Id.* Thus, the VSIDs are expired or obsolete. *Id.*
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Dirks discloses accessing a linked list of records. Dirks also discloses accessing a linked list of records having same hash address. For example, Dirks inherently discloses that page table entry groups, which are comprised of page table entries, are stored in a linked list.*See id.* at 9:32-46. For example, in some page tables, a suitable number of entries are grouped together, and the addressing of entries is done by groups. In other words, the page table address PTA that results from the operation of the hashing function is the physical address of a page table entry group. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id. at 9:36-46.* Because each page table entry stored in a page table entry group has the same hash address, the page table entries are being stored in a linked list; such an inherent characteristic necessarily flows from the teachings of the applied prior art. *See id.*

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		To the extent that Bedrock argues that Dirks does not anticipate Claims 1 – 8 because the page table entry groups are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the page table entries of a page table group in a linked list. The admitted prior art in the background of the ’120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. '120 patent at 1:34-2:6. Thus, Dirks and the admitted prior art of the ’120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as shown in Figure 3 of Morris, it was well known in the prior art to have page tables entries distributed by a hash function, such as those in Dirks, and then store the page table entries in a hash table using linked lists or external chaining. Morris at 3:54-4:24, Figure 3.

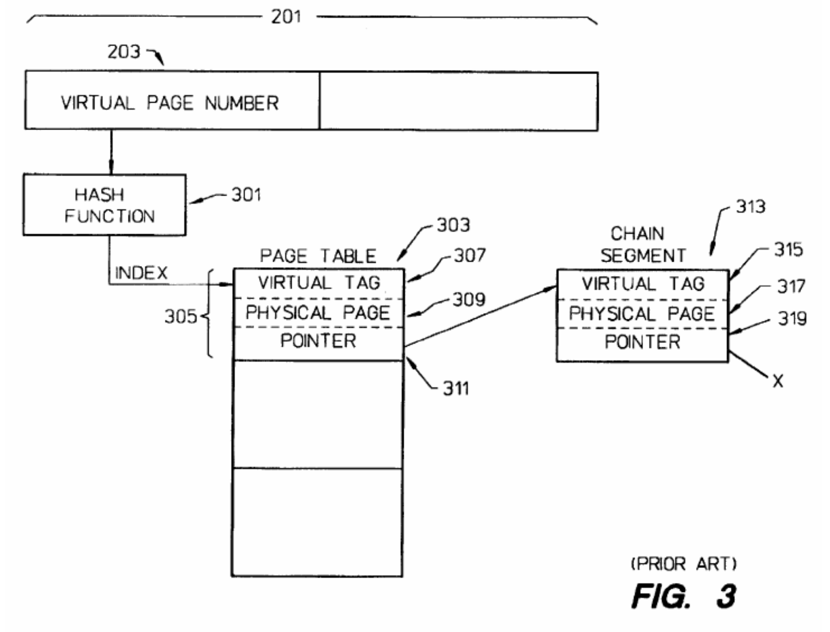
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 201 203 VIRTUAL PAGE NUMBER HASH FUNCTION 301 INDEX PAGE TABLE 303 307 VIRTUAL TAG PHYSICAL PAGE POINTER 309 305 311 CHAIN SEGMENT 313 315 317 319 VIRTUAL TAG PHYSICAL PAGE POINTER X (PRIOR ART) FIG. 3 *Id.* at Figure 3. As described by Morris: FIG. 3 illustrates the process of retrieving the physical page information given the virtual page number as would be required to update the TLB after a TLB miss. As described above, the virtual to physical mappings are maintained in a page table. For translating a given virtual address to a physical address, one

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		approach is to perform a many-to-one function (hash) on the virtual address to form an index into the page table. This gives a pointer to a linked list of entries. These entries are then searched for a match. To determine a match, the virtual page number is compared to an entry in the page table (virtual tag). If the two are equal, that page table entry provides the physical address translation. *Id.* at 3:54-65. In the example illustrated, a hash function 301 is performed on the virtual page number 203 to form an index. This index is an offset into the page table 303. As shown, the index is 0, that is, the index points to the first entry 305 in the page table. Each entry in the page table consists of multiple parts but typically contains at least a virtual tag 307, a physical page 309 and a pointer 311. If the virtual page number 203 equals the virtual tag 307, then physical page 309 gives the physical (real) memory page address desired. If the virtual tag does not match, then the pointer 311 points to a chain of entries in memory which contain virtual to physical translation information. The additional information contained in the chain is needed as more than one virtual page number can hash to the same page table entry. *Id.* at 3:66-4:12. As shown, pointer 311 points to a chain segment 313. This chain segment contains the same type of information as the page table. As before, the virtual page number 203 is compared to the next virtual tag 315 to see if there is a match. If a match occurs then the associated physical page 317 gives the address

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		of the physical memory page desired. If a match does not occur, then the pointer 319 is examined to locate the next chain segment, if any. If the pointer 319 does not point to another chain segment, as shown, then a page fault has occurred. A page fault software program is then used, as described in association with FIG. 1, to update the page table. *Id.* at 4:12-24. As both Dirks and Morris disclose systems and methods for allocating memory address controls using page table entries that are stored in hash tables, one of ordinary skill in the art would have understood how to combine the hashed page table with linked lists taught in Morris with Dirks. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks with Morris would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Dirks' page table being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and understood by one of ordinary skill in the art to the system disclosed in the admitted prior art, and would have seen the benefits of doing so. One possible benefit, for example, is hash table collision resolution.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Dirks discloses identifying at least some of the automatically expired ones of the records. For example, Dirks discloses hash table search means such as searching, creation, deletion, and sweeping. Dirks at 5:31-47, 5:66-6:15, 5:39-44, 9:39-

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		47. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id.* at 9:39-47. For example, Dirks inherently discloses that page table entry groups, which are comprised of page table entries, are stored in a linked list. *See id.* at 9:32-46. For example, in some page tables, a suitable number of entries are grouped together, and the addressing of entries is done by groups. In other words, the page table address PTA that results from the operation of the hashing function is the physical address of a page table entry group. Whenever a call is made to a particular virtual address, the individual page table entries within an addressed group are then checked, one by one, to determine whether they correspond to the virtual address that generated a page table search. Thus, it can be seen that it takes longer to locate an entry that resides in the last position in the group, relative to an entry in the first position in the group. *Id.* Because each page table entry stored in a page table entry group has the same hash address, the page table entries are being stored in a linked list; such an inherent characteristic necessarily flows from the teachings of the applied prior art. *See id.*

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		To the extent that Bedrock argues that Dirks does not anticipate Claims 1 – 8 because the page table entry groups are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the page table entries of a page table group in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. The '120 patent at 1:34-2:6. Thus, Dirks and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as shown in Figure 3 of Morris, it was well known in the prior art to have page tables entries distributed by a hash function, such as those in Dirks, and then store the page table entries in a hash table using linked lists or external chaining. Morris at 3:54-4:24, Figure 3.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 201 203 VIRTUAL PAGE NUMBER HASH FUNCTION 301 INDEX PAGE TABLE 303 307 VIRTUAL TAG PHYSICAL PAGE POINTER 305 309 311 CHAIN SEGMENT 313 315 317 319 X (PRIOR ART) FIG. 3 *Id.* at Figure 3. As described by Morris: FIG. 3 illustrates the process of retrieving the physical page information given the virtual page number as would be required to update the TLB after a TLB miss. As described above, the virtual to physical mappings are maintained in a page table. For translating a given virtual address to a physical address, one approach is to perform a many-to-one function (hash) on the virtual address to form an index into the page table. This gives a

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		pointer to a linked list of entries. These entries are then searched for a match. To determine a match, the virtual page number is compared to an entry in the page table (virtual tag). If the two are equal, that page table entry provides the physical address translation. *Id.* at 3:54-65. In the example illustrated, a hash function 301 is performed on the virtual page number 203 to form an index. This index is an offset into the page table 303. As shown, the index is 0, that is, the index points to the first entry 305 in the page table. Each entry in the page table consists of multiple parts but typically contains at least a virtual tag 307, a physical page 309 and a pointer 311. If the virtual page number 203 equals the virtual tag 307, then physical page 309 gives the physical (real) memory page address desired. If the virtual tag does not match, then the pointer 311 points to a chain of entries in memory which contain virtual to physical translation information. The additional information contained in the chain is needed as more than one virtual page number can hash to the same page table entry. *Id.* at 3:66-4:12. As shown, pointer 311 points to a chain segment 313. This chain segment contains the same type of information as the page table. As before, the virtual page number 203 is compared to the next virtual tag 315 to see if there is a match. If a match occurs then the associated physical page 317 gives the address of the physical memory page desired. If a match does not occur, then the pointer 319 is examined to locate the next chain

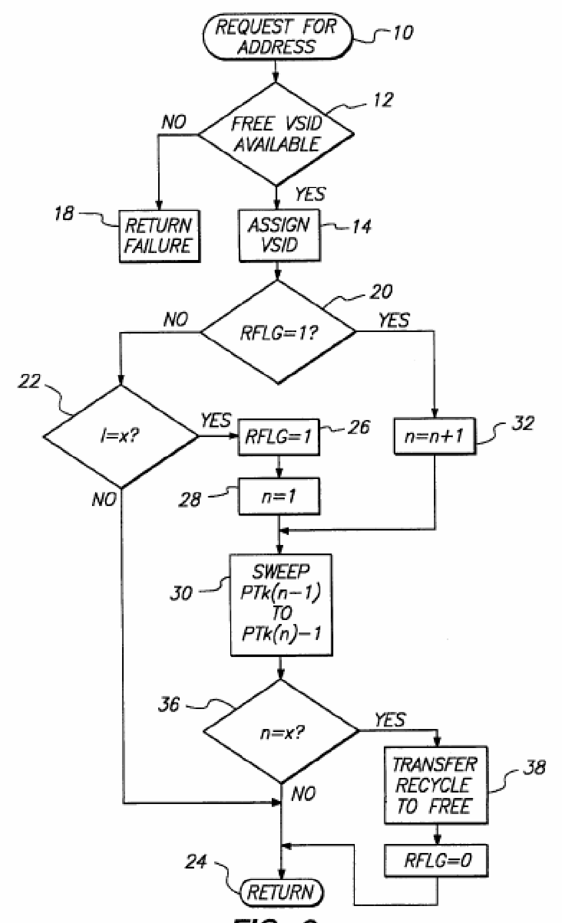
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		segment, if any. If the pointer 319 does not point to another chain segment, as shown, then a page fault has occurred. A page fault software program is then used, as described in association with FIG. 1, to update the page table. *Id.* at 4:12-24. As both Dirks and Morris disclose systems and methods for allocating memory address controls using page table entries that are stored in hash tables, one of ordinary skill in the art would have understood how to combine the hashed page table with linked lists taught in Morris with Dirks. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks with Morris would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Dirks' page table being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and understood by one of ordinary skill in the art to the system disclosed in the admitted prior art, and would have seen the benefits of doing so. One possible benefit, for example, is hash table collision resolution.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Dirks discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, as discussed above at some point VSIDs become inactive. Dirks at 6:24-30. These expired or inactive VSIDs are not removed "in one colossal step, for example after all of the free VSIDs have been allocated. Rather, the sweeping is carried out in an incremental, ongoing manner to avoid significant

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		interruptions in the running of programs." *Id.* at 6:39-44. "More specifically, each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table." *Id.* at 3:14-18. Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24. A flowchart which depicts the operation of the address allocation portion of a memory manager, in accordance with the present invention, is illustrated in FIG. 6. Referring thereto, operation begins when a request for address space is generated (Step 10). This can occur when a thread is created, for example. In response thereto, the operating system checks the free list to determine whether a free VSID is available (Step 12). If so, a free VSID is allocated to the thread that generated the request (Step 14). In the case of an application, two or more VSIDs might be assigned. If no free VSID is available at Step 12, a failure status is returned (Step 18). In practice, however, such a situation should not occur, since the process of the present invention ensures that free VSIDs are always available. *Id.* at 7:66-8:12. After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. "[If for example], *k*=10000/500=20. For a given value of the index n, therefore, the system sweeps table entries $PT_{k(n-1)}$ to $PT_{(n)-1}$. Thus, during the first sweep, table entries PT_0 to PT_{19} are examined in the example given above." *Id.* at 8:34-37. "If a recycling sweep is already in progress, i.e. the response is affirmative at Step 20, the index n is incremented (Step 32) and a sweep of the next *k* entries is carried out. Thus, where n=2, entries $PT_{[20]}$ to PT_{39} will be examined." *Id.* at 8:38-41. The process continues in this manner, with k entries in the page table being examined each time a free VSID is allocated. Each entry is examined to determine whether it contains a mapping for a VSID on the recycle list. If it does, that entry is removed from the page table. *Id.* at 8:42-46.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		After each sweep, the system determines whether all entries in the page table have been checked (Step 36). This situation will occur when n=x. Once all of the entries have been examined, the VSIDs in the recycle list are transferred to the free list (Step 38), yielding x new VSIDs that are available to be allocated. In addition, the recycle sweep flag RFLG is reset, and control is then returned to the program. If all of the entries in the page table have not yet been checked, i.e. the response is negative at Step 36, control is directly returned to the application program, at Step 24. *Id.* at 8:47-56.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 20 RFLG=1? NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 SWEEP PTk(n−1) TO PTk(n)−1 30 36 n=x? YES TRANSFER RECYCLE TO FREE 38 NO RFLG=0 24 RETURN FIG. 6 *Id.* at Figure 6.

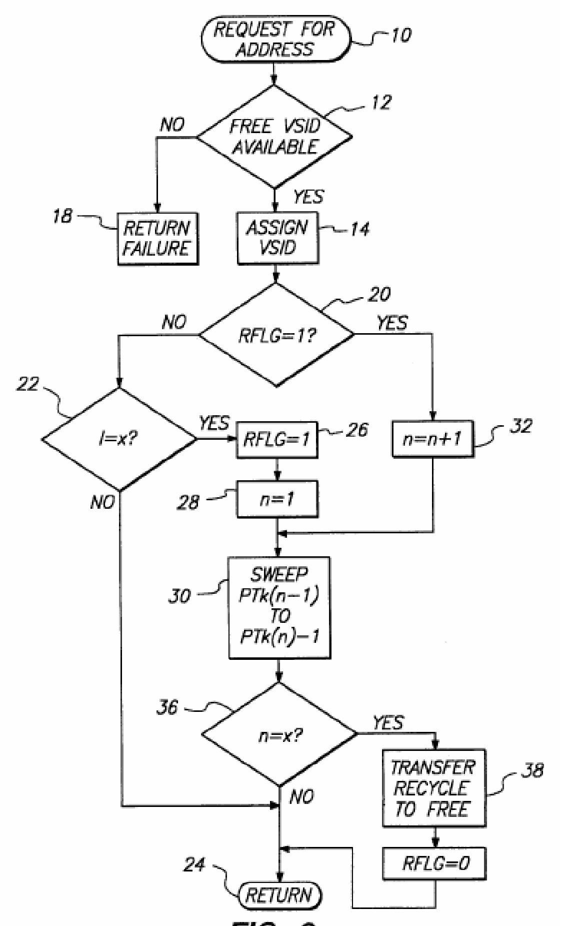
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Dirks discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, as discussed above at some point VSIDs become inactive. *Id.* at 6:24-30. These expired or inactive VSIDs are not removed "in one colossal step, for example after all of the free VSIDs have been allocated. Rather, the sweeping is carried out in an incremental, ongoing manner to avoid significant interruptions in the running of programs." *Id.* at 6:39-44 "More specifically, each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table." *Id.* at 3:14-18. Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24 A flowchart which depicts the operation of the address allocation portion of a memory manager, in accordance with the present invention, is illustrated in FIG. 6. Referring thereto, operation begins when a request for address space is generated (Step 10). This can occur when a thread is created, for example. In response thereto, the operating system checks the free list to determine whether a free VSID is available (Step 12). If so, a free VSID is allocated to the thread that generated the request (Step 14). In the case of an application, two or more VSIDs might be assigned. If no free VSID is available at Step 12, a failure status is returned (Step 18). In practice, however, such a situation should not occur, since the process of the present invention ensures that free VSIDs are always available. *Id.* at 7:66-8:12.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. "[If for example], *k*=10000/500=20. For a given value of the index n, therefore, the system sweeps table entries $PT_{k(n-1)}$ to $PT_{(n)-1}$. Thus, during the first sweep, table entries PT_0 to PT_{19} are examined in the example given above." *Id.* at 8:34-37. "If a recycling sweep is already in progress, i.e. the response is affirmative at Step 20, the index n is incremented (Step 32) and a sweep of the next *k* entries is carried out. Thus, where n=2, entries PT20 to PT_{39} will be examined." *Id.* at 8:38-41.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		The process continues in this manner, with k entries in the page table being examined each time a free VSID is allocated. Each entry is examined to determine whether it contains a mapping for a VSID on the recycle list. If it does, that entry is removed from the page table. *Id.* at 8:42-46. After each sweep, the system determines whether all entries in the page table have been checked (Step 36). This situation will occur when n=x. Once all of the entries have been examined, the VSIDs in the recycle list are transferred to the free list (Step 38), yielding x new VSIDs that are available to be allocated. In addition, the recycle sweep flag RFLG is reset, and control is then returned to the program. If all of the entries in the page table have not yet been checked, i.e. the response is negative at Step 36, control is directly returned to the application program, at Step 24. *Id.* at 8:47-56.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 20 RFLG=1? NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 SWEEP PTk(n−1) TO PTk(n)−1 30 36 n=x? YES NO TRANSFER RECYCLE TO FREE 38 RFLG=0 24 RETURN FIG. 6 *Id.* at Figure 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		As step 24 returns to the application program, a VSID will be inserted, searched for, or deleted following the step of removing. *Id*. at Figure 6, 8:55-56. For example, after step 24, a VSID will be assigned with the next request for address 10, which would have followed the step of removing. *Id*.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Dirks discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, "each time that a new range of addresses is allocated to a program, a limited number of entries in the page table are examined, to determine whether the addresses associated with those entries are no longer in use and the entries can be removed from the page table." *Id.* at 3:14-18. "Thus, rather than halting the operation of the computer for a considerable period of time to scan the entire page table when a logical address area is deleted, the memory manager of the present invention carries out a limited, time-bounded examination upon each address allocation." *Id.* at 3:25-29. Moreover, Dirks similarly teaches that a limited number of entries can be examined each time a thread is deleted. *Id.* at 10:23-24. In operation, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. *Id.* at 7:2-14.

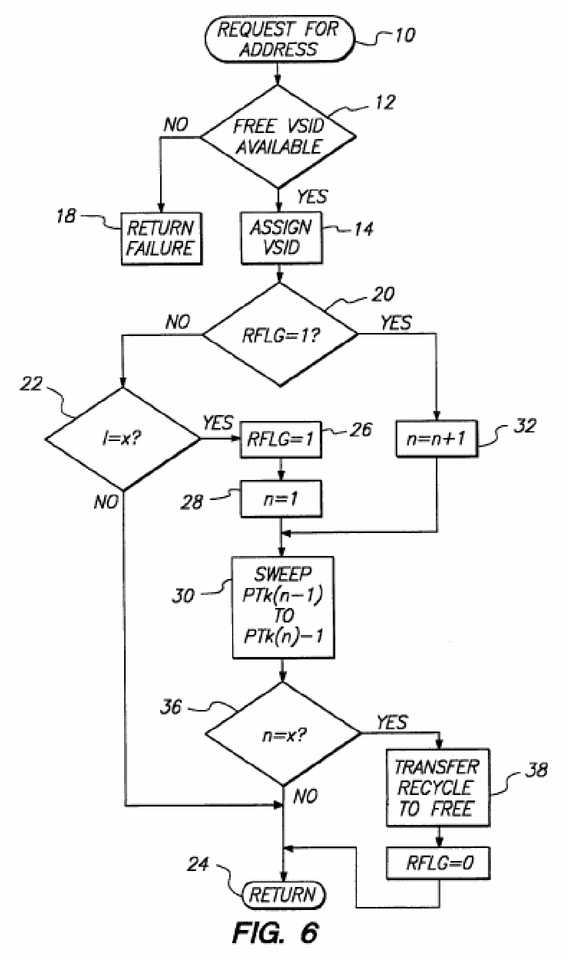
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		 REQUEST FOR ADDRESS 10 FREE VSID AVAILABLE 12 NO YES 18 RETURN FAILURE ASSIGN VSID 14 RFLG=1? 20 NO YES 22 I=x? YES RFLG=1 26 n=n+1 32 NO 28 n=1 30 SWEEP PTk(n-1) TO PTk(n)-1 36 n=x? YES TRANSFER RECYCLE TO FREE 38 NO RFLG=0 24 RETURN **FIG. 6** *Id.* at Figure 6.

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		As shown in Figure 6, After the new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as k, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dirks to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dirks with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dirks can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Dirks is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")**
		of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Dirks, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Dirks. For example, both Linux 2.0.1 and Dirks describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1`

EXHIBIT B-2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 6,119,214 to Dirks ("Dirks") alone and in combination with U.S. Patent No. 5,724,538 to Morris et al. ("Morris")
		the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-3

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Mellender discloses an information storage and retrieval system. For example, Mellender discloses that "[a] database residing in the mass memory stores objects and components of logic programs as objects in a common data structure format, applications data, and application stored as compiled interpreter code. The database is managed by an [sic] database manager that represents objects and components of the logic programming language in the common data structure format as objects and is responsive to calls for retrieving and storing objects in the database and for automatically deleting objects from the database when they have become obsolete." U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) at 2:22-33.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Mellender discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Mellender also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Mellender discloses that "[a] database residing in the mass memory stores objects and components of logic programs as objects in a common data structure format, applications data, and application stored as compiled interpreter code. The database is managed by an [sic] database manager that represents objects and components of the logic programming language in the common data structure format as objects and is responsive to calls for retrieving and storing objects in the database and for automatically deleting objects from the database when they have become obsolete." *Id.*

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		Furthermore, Mellender discloses that "[t]he database 40 consists of 2 UNIX files: db.key and db.prime. The key file provides associative access to the prime file: the access manager hashes into the key file (all of whose records are of fixed length), and finds the address (file offset) of the object in the prime file." *Id*. at 54:50-53. Furthermore, Mellender discloses that "[c]ollisions in the key file are handled by chaining the objects in the prime file together. If the object at the address indicated by the key file record does not have an id (oop, or string) that matches the target sought, the access manager 58 follows the 'overflow' chain in the records in the prime file, checking the target against the id until it is found. Fastest access to newest objects is provided by placing them first in the overflow chain." *Id*. at 55:20-27. "If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 56:14-19. Furthermore, Mellender states that "[g]arbage is defined as objects that are no longer reachable, and therefore can be safely discarded. Since there is no explicit delete command available to the programmer in Smalltalk language, removal of objects is entirely up to the system." *Id*. at 59:41-45.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of	Mellender discloses a record search means utilizing a search key to access the linked list. Mellender also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
	records having the same hash address,	For example, Mellender discloses that "[t]he database 40 consists of 2 UNIX files: db.key and db.prime. The key file provides associative access to the prime file: the access manager hashes into the key file (all of whose records are of fixed length), and finds the address (file offset) of the object in the prime file." *Id*. at 54:50-53 Furthermore, Mellender discloses that "[c]ollisions in the key file are handled by chaining the objects in the prime file together. If the object at the address indicated by the key file record does not have an id (oop, or string) that matches the target sought, the access manager 58 follows the 'overflow' chain in the records in the prime file, checking the target against the id until it is found. Fastest access to newest objects is provided by placing them first in the overflow chain." *Id*. at 55:20-27. "If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 56:14-19.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Mellender discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Mellender also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Mellender states that "[t]he forceit function will put a record in

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id.* at 56:22-32 Furthermore, Mellender discloses that "[m]any (non-reference counting) garbage collectors do little processing at reference creation time, but wait until the collector is called in order to clean out a region by moving objects to other regions. Our collector does most of its work when cross-region instance variable assignments are made, and when processing Smalltalk 'return' statements, which distributes the garbage collection processing evenly throughout the run. This means that the periods when the system is doing garbage collection (and is thus unavailable to the user) is spread evenly throughout the session and there are no long periods of time when the system is unavailable." *Id.* at 62:68-70 – 63:1-9.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed	Mellender discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Mellender also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Mellender discloses "[t]he fechit function retrieves an object

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
	linked list of records.	from the database 40, given a record type and key. The fetched record is placed in the buffers …along with the disk address of the retrieved record. This will be used if/when the record needs to be replaced in the database." *Id*. at 55:61-66. Furthermore, Mellender discloses that "[t]he storeit function is capable of adding a new object (or replacing same) in the database . . . If no key entry exists for the new record, it sets one up and adds the record to the end of the prime file. If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 55:67-70 – 56:1-19. Furthermore, Mellender states that "[t]he forceit function will put a record in the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id*. at 56:22-32. Furthermore, Mellender discloses that "[m]any (non-reference counting) garbage collectors do little processing at reference creation time, but wait until the collector is called in order to clean out a region by moving objects to other regions. Our collector does most of its work when cross-region instance

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		variable assignments are made, and when processing Smalltalk 'return' statements, which distributes the garbage collection processing evenly throughout the run. This means that the periods when the system is doing garbage collection (and is thus unavailable to the user) is spread evenly throughout the session and there are no long periods of time when the system is unavailable." *Id*. at 62:68-70 – 63:1-9.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Mellender discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Mellender states that "[t]he forceit function will put a record in the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id*. at 56:22-32. In summary, if no old copy of the new record exists or the new record is smaller than the one it is replacing, the forceit function calls the storeit function to add the new object and no deletion takes place. If an old copy of the new record does exist, the access manager deletes the old copy and then executes the storeit function to store the new record. Therefore, the determination of when to delete a record is dynamically determined by whether an old copy of the record exists or when the new record is smaller than the one

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		it is replacing. *Id*. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mellender to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mellender with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Mellender is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		records is not disclosed by Mellender, Mellender combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20).

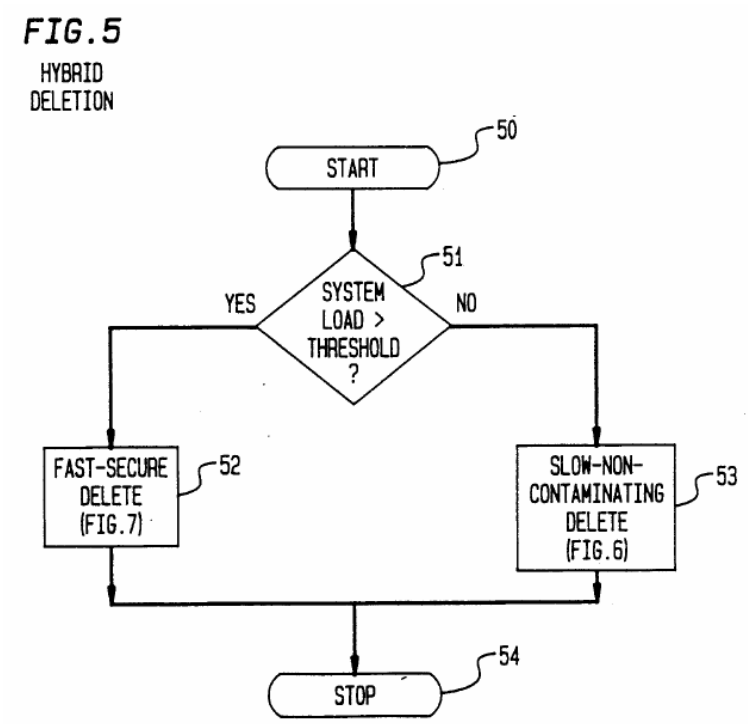
EXHIBIT B-3

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		Id. at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Mellender and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Mellender nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Mellender and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Mellender with the means

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Mellender and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Mellender with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Mellender with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Mellender with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Mellender with Thatte. Alternatively, it would also be obvious to combine Mellender with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). > In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. > This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Mellender and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15.

EXHIBIT B-3

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Mellender would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Mellender and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Mellender with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Mellender and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		implementations such as Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Mellender would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Mellender and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mellender to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mellender with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Mellender in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Mellender. For example, both Linux 2.0.1 and Mellender describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		When the function rt_cache_add removes an expired record, the function rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold RT_CACHE_SIZE_MAX. If the number of records in the hash table exceeds the predetermined threshold RT_CACHE_SIZE_MAX, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		`rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		`rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Mellender discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Mellender also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Mellender discloses that "[a] database residing in the mass memory stores objects and components of logic programs as objects in a common data structure format, applications data, and application stored as compiled interpreter code. The database is managed by an [sic] database manager that represents objects and components of the logic programming language in the common data structure format as objects and is responsive to calls for retrieving and storing objects in the database and for automatically deleting objects from the database when they have become obsolete." Mellender at 2:22-33. Furthermore, Mellender discloses that "[t]he database 40 consists of 2 UNIX files: db.key and db.prime. The key file provides associative access to the prime file: the access manager hashes into the key file (all of whose records are of fixed length), and finds the address (file offset) of the object in the prime file." *Id*. at 54:50-53.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		Furthermore, Mellender discloses that "[c]ollisions in the key file are handled by chaining the objects in the prime file together. If the object at the address indicated by the key file record does not have an id (oop, or string) that matches the target sought, the access manager 58 follows the 'overflow' chain in the records in the prime file, checking the target against the id until it is found. Fastest access to newest objects is provided by placing them first in the overflow chain." *Id*. at 55:20-27. "If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 56:14-19. Furthermore, Mellender states that "[g]arbage is defined as objects that are no longer reachable, and therefore can be safely discarded. Since there is no explicit delete command available to the programmer in Smalltalk language, removal of objects is entirely up to the system." *Id*. at 59:41-45.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Mellender discloses accessing a linked list of records. Mellender also discloses accessing a linked list of records having same hash address. For example, Mellender discloses that "[t]he database 40 consists of 2 UNIX files: db.key and db.prime. The key file provides associative access to the prime file: the access manager hashes into the key file (all of whose records are of fixed length), and finds the address (file offset) of the object in the prime file." *Id*. at 54:50-53. Furthermore, Mellender discloses that "[c]ollisions in the key file are handled

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		by chaining the objects in the prime file together. If the object at the address indicated by the key file record does not have an id (oop, or string) that matches the target sought, the access manager 58 follows the 'overflow' chain in the records in the prime file, checking the target against the id until it is found. Fastest access to newest objects is provided by placing them first in the overflow chain." *Id*. at 55:20-27. "If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 56:14-19.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Mellender discloses identifying at least some of the automatically expired ones of the records. For example, Mellender discloses that "[a] database residing in the mass memory stores objects and components of logic programs as objects in a common data structure format, applications data, and application stored as compiled interpreter code. The database is managed by an [sic] database manager that represents objects and components of the logic programming language in the common data structure format as objects and is responsive to calls for retrieving and storing objects in the database and for automatically deleting objects from the database when they have become obsolete." *Id*. at 2:22-33. Furthermore, Mellender states that "[g]arbage is defined as objects that are no longer reachable, and therefore can be safely discarded. Since there is no explicit delete command available to the programmer in Smalltalk language,

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		removal of objects is entirely up to the system." *Id*. at 59:41-45.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Mellender discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Mellender states that "[t]he forceit function will put a record in the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id*. at 56:22-32. Furthermore, Mellender discloses that "[m]any (non-reference counting) garbage collectors do little processing at reference creation time, but wait until the collector is called in order to clean out a region by moving objects to other regions. Our collector does most of its work when cross-region instance variable assignments are made, and when processing Smalltalk 'return' statements, which distributes the garbage collection processing evenly throughout the run. This means that the periods when the system is doing garbage collection (and is thus unavailable to the user) is spread evenly throughout the session and there are no long periods of time when the system is unavailable." *Id*. at 62:68-70 – 63:1-9.
	[7d] inserting, retrieving or deleting one of the	Mellender discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
	records from the system following the step of removing.	For example, Mellender discloses "[t]he fechit function retrieves an object from the database 40, given a record type and key. The fetched record is placed in the buffers [] along with the disk address of the retrieved record. This will be used if/when the record needs to be replaced in the database." *Id*. at 55:61-66. Furthermore, Mellender discloses that "[t]he storeit function is capable of adding a new object (or replacing same) in the database . . . If no key entry exists for the new record, it sets one up and adds the record to the end of the prime file. If the key entry does exist a collision results. The access manager 58 fetches the record pointed to by the key, updates the new record's overflow pointer to point to the record currently pointed to by the key record, and then updates the key record to point to the new record being added." *Id*. at 55:67-70 – 56:1-19. Furthermore, Mellender states that "[t]he forceit function will put a record in the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id*. at 56:22-32. Furthermore, Mellender discloses that "[m]any (non-reference counting) garbage collectors do little processing at reference creation time, but wait until

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		the collector is called in order to clean out a region by moving objects to other regions. Our collector does most of its work when cross-region instance variable assignments are made, and when processing Smalltalk 'return' statements, which distributes the garbage collection processing evenly throughout the run. This means that the periods when the system is doing garbage collection (and is thus unavailable to the user) is spread evenly throughout the session and there are no long periods of time when the system is unavailable." *Id*. at 62:68-70 – 63:1-9.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Mellender discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, Mellender states that "[t]he forceit function will put a record in the database, but (unlike storeit) checks to see if it is already there. If so, it logically deletes the old copy and adds the new one. This function is called when an object is newly created with an oop that already exists (e.g. a Class), and when an object is lengthened. It uses the storeit function if the new object is not already in the database, or is smaller than the one it is replacing. Else, the access manager gets the old object and logically deletes it (by placing a special mark in the rec_type), and then executes the storeit logic." *Id*. at 56:22-32 In summary, if no old copy of the new record exists or the new record is smaller than the one it is replacing, the forceit function calls the storeit function to add the new object and no deletion takes place. If an old copy of the new record does exist, the access manager deletes the old copy and then executes the storeit function to store the new record. Therefore, the determination of when to delete a record can be dynamically determined by

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		whether an old copy of the record exists or when the new record is smaller than the one it is replacing. *Id*. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mellender to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mellender with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Mellender is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired

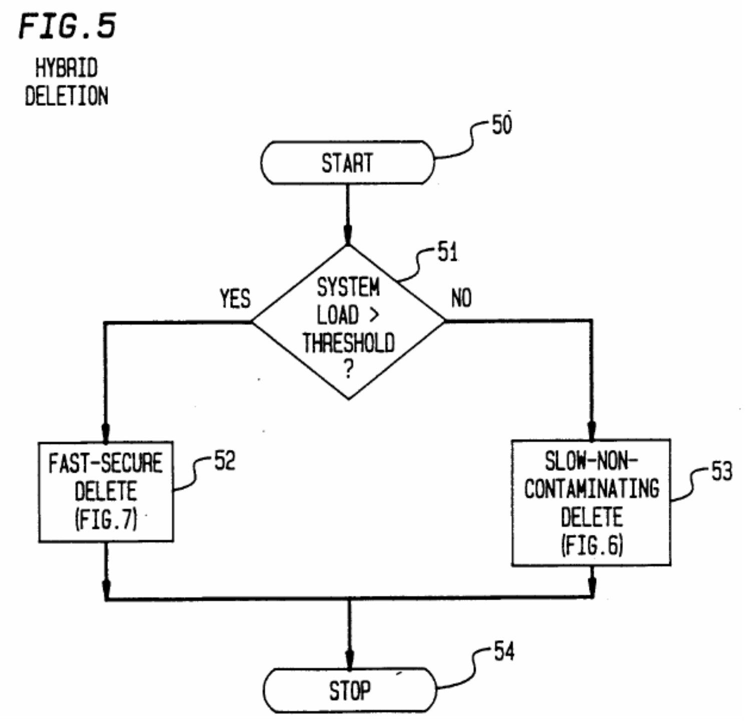
Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		records is not disclosed by Mellender, Mellender combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20).

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		Id. at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Mellender and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Mellender nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Mellender and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Mellender with the means

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Mellender and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Mellender with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Mellender with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Mellender with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Mellender with Thatte. Alternatively, it would also be obvious to combine Mellender with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.  FIG.5 HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Mellender and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Mellender would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Mellender and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Mellender with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the

Asserted Claims From U.S. Pat. No. 5,893,120		**Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")**
		garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Mellender and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		implementations such as Mellender. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Mellender would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Mellender and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mellender to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mellender with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Mellender can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Mellender in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Mellender. For example, both Linux 2.0.1 and Mellender describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		When the function rt_cache_add removes an expired record, the function rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold RT_CACHE_SIZE_MAX. If the number of records in the hash table exceeds the predetermined threshold RT_CACHE_SIZE_MAX, the function

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		rt_cache_add invokes a function rt_garbage_collect. *See* Linux 2.0.1, route.c at lines 1341-1342. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function

Asserted Claims From U.S. Pat. No. 5,893,120		Mellender et al., "Object-Oriented, Logic, and Database Programming Tool with Garbage Collection," U.S. Patent No. 4,989,132 (issued Jan. 29, 1991) ("Mellender")
		`rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-4

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Robinson discloses an information storage and retrieval system. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." Robinson, "Data Cache Using Dynamic Frequency Based Replacement and Boundary Criteria," U.S. Patent No. 5,043,885 (issued Aug. 27, 1991) at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records	Robinson discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Robinson also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Robinson discloses "[a] cache directory keeps track of which

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
	automatically expiring,	blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41. "Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Robinson discloses a record search means utilizing a search key to access the linked list. Robinson also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41.

EXHIBIT B-4

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		"Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for." *Id*. at 6:14-27.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Robinson discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Robinson also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. Robinson discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Robinson also discloses a hashing means to provide access to records stored in

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41. "Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		"Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Robinson discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Robinson also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. Robinson discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Robinson also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss."

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		Id. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41. "Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Robinson combined with Hamstra discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, as summarized in Hamstra: If the data is not resident in the cache memory then it is staged by segments from a disk, placed in the cache memory, and sent to the host. However, this *may* require that *some of the segments* in the cache memory be replaced by the segments from the disks. *Hamstra et al.*, "Processor-Addressable Timestamp for Indicating Oldest Written-to-Cache Entry Not Copied Back to Bulk Memory," U.S. Patent No. 4,530,054 (issued Jul. 16, 1985) at 1:36-41 (emphasis added). [W]hen a new segment or segments has to be brought from a device 104 to the cache memory 106 *and* there are no empty segments in the cache memory 106 then *some of the segments* resident in the cache memory 106 must be deleted or removed therefrom in order to make room for the new segments.*Id.* at 6:29-34 (emphasis added). [W]hen the SCU 100 has no other work to do it may search the SDT [– the segment descriptor (SDT) contains an entry for each segment of data resident in the cache memory, similar to a hash table (*Id.* at 5:33-35) –] to locate segments which have been written to the cache and trickle these segments to the devices

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		104 thus making space available in cache 106 for additional segments. The trickling of written-to segments takes place on the same basis as segment replacement, that is, the least recently used segments are trickled first . . . . *Id*. at 6:53-60. In Hamstra, empty segments are created by either first initialization of the cache before segments in cache memory are filled or by the trickle down functionality of the system. According to Hamstra, *if* there are no empty segments in the cache memory, *then some of the segments* in the cache memory are deleted by the system. However, if there *are* empty segments in the cache, then there is no need to delete any segments and the requested segments are transfered from a disk into those empty segments. Therefore, the system described in Hamstra dynamically determines whether to delete a segment based on the availability of empty segments in the cache memory. As both Robinson and Hamstra relate to a system of cache maintenance using a least recently used method for replacement of cache segments organized in linked lists, one of ordinary skill in the art would understand how to use the Hamstra patent's dynamic decision on whether to perform a segment deletion based on the availability of empty segments in a cache memory when implementing a cache maintenance system such as Robinson. Moreover, one of ordinary skill in the art would recongize that it would improve similar systems and methods in the same way. The fundamental goal of a cache memory, as described by both Robinson and Hamstra, is to provide users fast access to frequently used data. *Robinson*, U.S. Patent No. 5,043,885 at 1:9-36; *Hamstra et al*, U.S. Patent No. 4,530,054 at 1:22-31. This is achieved by insuring that data residing in the cache is the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		most recently and frequently used data. A person skilled in the art would appreciate that the technique of replacing least recently used data segments from cache memory upon an insertion of a new data segment can be expanded to include logic for trickling down least recently used segments to make space available in the cache for additional segments when the system has no other work to do. One of ordinary skill in the art would recognize that the result of combining the system disclosed in Robinson with the additional system of trickling down least recently used segments and dynamically determining whether a deletion is necessary upon a call as disclosed in Hamstra would further promote the goals of a cache memory. For example, such benefits would include creating faster access to data for the user – the additional step of a deletion would only take place if there was no empty space in the cache memory, and creating increased efficiency of system performance by utilizing unused time, when there is no work to be done, to trickle down least recently used segments from cache memory back to disk. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Robinson to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Robinson with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example,

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Robinson is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Robinson and Hamstra, Robinson combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

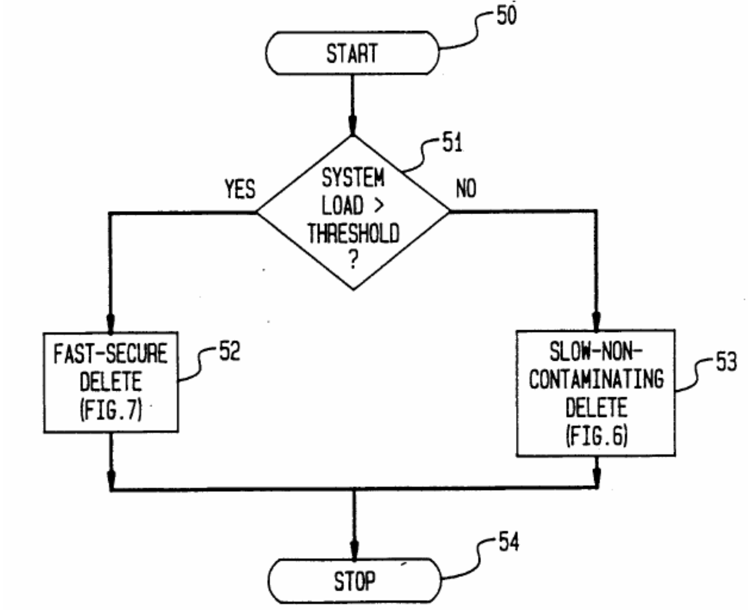
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Robinson and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Robinson nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Robinson and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Robinson with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Robinson and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		combining Robinson with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Robinson with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Robinson with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Robinson with Thatte. Alternatively, it would also be obvious to combine Robinson with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-4

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Robinson and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the ’663 patent’s deletion decision procedure with Robinson would be nothing more than the predictable use of prior art elements according to their established functions.

EXHIBIT B-4

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Robinson and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Robinson with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Robinson and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Robinson would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Robinson and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Robinson to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Robinson with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

EXHIBIT B-4

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Robinson in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Robinson. For example, both Linux 2.0.1 and Robinson describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Robinson discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Robinson also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. Robinson discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Robinson also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Robinson*, U.S. Patent No. 5,043,885 at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41 "Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation."*Id*. at 4:53-62.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Robinson discloses accessing a linked list of records. Robinson also discloses accessing a linked list of records having same hash address. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41 "Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for." *Id*. at 6:14-27.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Robinson discloses identifying at least some of the automatically expired ones of the records. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Robinson discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34. "The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values." *Id*. at 5:35-41. "Since the preferred embodiment makes use of a hash function to locate [cache

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field." *Id*. at 5:65-69 – 6:1-2. "A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS." *Id*. at 6:14-34. "[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Robinson discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Robinson discloses "[a] cache directory keeps track of which blocks are in the cache, the number of times each block in the cache has been referenced after aging at least a predetermined amount (reference count), and the age of each block since the last reference to that block, for use in determining which of the cache blocks is replaced when there is a cache miss." *Id*. at Abstract. "Each block in the cache has a corresponding cache directory entry in the cache directory and is found by means of the cache directory. The cache directory consists of an array of cache directory entries, individual pointers and pointer tables used for locating blocks, updating the directory, and making replacement decisions. The location in the cache of a block found in the cache directory is known from the offset (i.e., the position) of the corresponding cache directory entry in the array of cache directory entries." *Id*. at 5:25-34.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson (“Robinson”) alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. (“Hamstra”)**
		“The preferred embodiment makes use of several known data structures and techniques, including a hash table for locating blocks in the cache, and doubly-linked lists for (1) the overall LRU chain, (2) individual LRU chains for each count value below a threshold, and (3) the chains of cache directory entries having identical hash values.” *Id*. at 5:35-41. “Since the preferred embodiment makes use of a hash function to locate [cache directory entries], more than one origin may correspond with the same hash value. Fields 34, 36 contain the pointers PHASH and NHASH which establish the doubly-linked list for the hash value which corresponds with the ID in field.” *Id*. at 5:65-69 – 6:1-2. “A [cache directory entry] for a block is found by first hashing the block ID using a hash function 44 to produce a hash value or offset, which corresponds with a particular element 46 in hash table 48. The hash table is an array of points to CDEs. The hash table element points either to the head or the tail of a doubly liked list of CDE having the same hash value. Starting with the CDE referred to in the hash table, the list of CDEs having the same hash value is searched sequentially (using either the pointer PHASH or NHASH in the CDEs depending upon whether the hash table pointed to the head or the tail) comparing the ID field in each such entry with the ID of the block being searched for. If an identical ID is found, the block is in the cache, and this case is referred to as a HIT. Otherwise either the hash table pointer is null or the ID was not found in the list. In such case the block is not in the cache and must be brought into the cache, which in general means that an existing block must be replaced with this new block. This case is referred to as a MISS.” *Id*. at 6:14-34.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		"[T]he cache directory essentially works in LRU fashion, with a cache directory entry being put in the MRU position each time a block is referenced. According to the invention, however, the block associated with the cache directory entry in the LRU position 16 will not necessarily be the one that it replaced when there is a miss. Additionally, according to the invention, there is a preselected boundary (age boundary) 18. Each time a block ages past this boundary, this fact is updated for that block in the cache directory, so that for each block in the cache it is known which side of the age boundary it is on." *Id*. at 4:34-45. "Various versions of the invention result from the way the reference counts are used to select blocks to replace. One simple version is as follows: when there is a miss, select the least recently used block in the non-local section whose count is below a preselected threshold. If there is no such block below the preselected count threshold, then select the least recently used block. Blocks whose counts are below the threshold can be tracked with a separate LRU chain, leading to an efficient implementation." *Id*. at 4:53-62.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Robinson combined with Hamstra discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, as summarized in Hamstra: If the data is not resident in the cache memory then it is staged by segments from a disk, placed in the cache memory, and sent to the host. However, this *may* require that *some of the segments* in the cache memory be replaced by the segments

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		from the disks. Hamstra et al, "Processor-Addressable Timestamp for Indicating Oldest Written-to-Cache Entry Not Copied Back to Bulk Memory," U.S. Patent No. 4,530,054 (issued Jul. 16, 1985) at 1:36-41 (emphasis added). [W]hen a new segment or segments has to be brought from a device 104 to the cache memory 106 *and* there are no empty segments in the cache memory 106 then *some of the segments* resident in the cache memory 106 must be deleted or removed therefrom in order to make room for the new segments. *Id*. at 6:29-34 (emphasis added). [W]hen the SCU 100 has no other work to do it may search the SDT [– the segment descriptor (SDT) contains an entry for each segment of data resident in the cache memory, similar to a hash table (*Id*. at 5:33-35) –] to locate segments which have been written to the cache and trickle these segments to the devices 104 thus making space available in cache 106 for additional segments. The trickling of written-to segments takes place on the same basis as segment replacement, that is, the least recently used segments are trickled first . . . . *Id*. at 6:53-60. In Hamstra, empty segments are created by either first initialization of the cache before segments in cache memory are filled or by the trickle down functionality of the system. According to Hamstra, *if* there are no empty segments in the cache memory, *then* some of the segments in the cache memory are deleted by the system. However, if there *are* empty segments in the cache, then there is no need to delete any segments and the requested

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		segments are transfered from a disk into those empty segments. Therefore, the system described in Hamstra dynamically determines whether to delete a segment based on the availability of empty segments in the cache memory. As both Robinson and Hamstra relate to a system of cache maintenance using a least recently used method for replacement of cache segments organized in linked lists, one of ordinary skill in the art would understand how to use the Hamstra patent's dynamic decision on whether to perform a segment deletion based on the availability of empty segments in a cache memory when implementing a cache maintenance system such as Robinson. Moreover, one of ordinary skill in the art would recongize that it would improve similar systems and methods in the same way. The fundamental goal of a cache memory, as described by both Robinson and Hamstra, is to provide users fast access to frequently used data. *Robinson*, U.S. Patent No. 5,043,885 at 1:9-36; *Hamstra et al*, U.S. Patent No. 4,530,054 at 1:22-31. This is achieved by insuring that data residing in the cache is the most recently and frequently used data. A person skilled in the art would appreciate that the technique of replacing least recently used data segments from cache memory upon an insertion of a new data segment can be expanded to include logic for trickling down least recently used segments to make space available in the cache for additional segments when the system has no other work to do. One of ordinary skill in the art would recognize that the result of combining the system disclosed in Robinson with the additional system of trickling down least recently used segments and dynamically determining whether a deletion is necessary upon a call as disclosed in Hamstra would further promote the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		goals of a cache memory. For example, such benefits would include creating faster access to data for the user – the additional step of a deletion would only take place if there was no empty space in the cache memory, and creating increased efficiency of system performance by utilizing unused time, when there is no work to be done, to trickle down least recently used segments from cache memory back to disk. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Robinson to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Robinson with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Robinson is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Robinson and Hamstra, Robinson combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list

EXHIBIT B-4

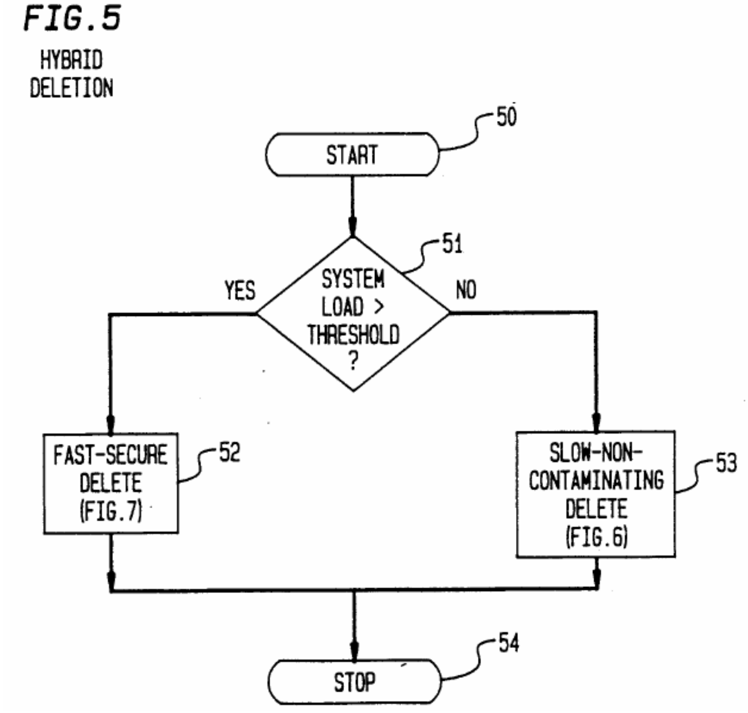
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson (“Robinson”) alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. (“Hamstra”)**
		without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Robinson and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Robinson nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Robinson and would

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Robinson with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Robinson and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Robinson with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Robinson with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Robinson with the teachings of Thatte would solve this problem by

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Robinson with Thatte. Alternatively, it would also be obvious to combine Robinson with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Robinson and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Robinson would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Robinson and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Robinson with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Robinson and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Robinson. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Robinson would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Robinson and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Robinson to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Robinson with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Robinson can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Robinson in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Robinson. For example, both Linux 2.0.1 and Robinson describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,043,885 to Robinson ("Robinson") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-4

EXHIBIT B-5

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Ish discloses an information storage and retrieval system. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Ish et al.*, "Method and Apparatus for Computer Disk Cache Management" U.S. Patent No. 5,778,430 (issued Jul. 7, 1998) at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with	Ish discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Ish also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
	same hash address, at least some of the records automatically expiring,	For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Ish discloses a record search means utilizing a search key to access the linked list. Ish also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id*. at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Ish discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Ish also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id*. at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. (“Ish”) alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. (“Hamstra”)**
		prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out “flushed” to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it.” *Id*. at 9:2-26.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Ish discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Ish also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Ish discloses an invention that “is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id.* at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id.* at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.

EXHIBIT B-5

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Ish alone or in combination with Hamstra discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. It is inherent to the Ish disclosure that upon a call to READ or WRITE, if there is empty space available in the, created by either (1) the initialization of the cache when segments within the cache have not yet been filled or (2) continuous flushing of the cache to remove the best candidates for replacement, blocks are transferred from a direct device to the cache without a deletion. If, however, empty space is not available, then the necessary space is made available within the cache by performing a delete. In order to promote the goals of a cache memory system, an efficient system would not make a deletion unless it was necessary, because the system would like to keep the cache filled to make retrieval for the user as fast as possible. Therefore, Ish inherently incorporates a dynamic decision making process in determining whether to delete a block within the cache upon a call to READ or WRITE. To the extent it is not inherent, it would be obvious to combine Ish with Hamstra. For example, as summarized in Hamstra: If the data is not resident in the cache memory then it is staged by segments from a disk, placed in the cache memory, and sent to the host. However, this *may* require that *some of the segments* in the cache memory be replaced by the segments from the disks. *Hamstra et al.*, "Processor-Addressable

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		Timestamp for Indicating Oldest Written-to-Cache Entry Not Copied Back to Bulk Memory," U.S. Patent No.4,530,054 (issued Jul. 16, 1985) at 1:36-41 (emphasis added). [W]hen a new segment or segments has to be brought from a device 104 to the cache memory 106 *and* there are no empty segments in the cache memory 106 then *some of the segments* resident in the cache memory 106 must be deleted or removed therefrom in order to make room for the new segments. *Id*. at 6:29-34 (emphasis added). [W]hen the SCU 100 has no other work to do it may search the SDT [– the segment descriptor (SDT) contains an entry for each segment of data resident in the cache memory, similar to a hash table (*Id*. at 5:33-35) –] to locate segments which have been written to the cache and trickle these segments to the devices 104 thus making space available in cache 106 for additional segments. The trickling of written-to segments takes place on the same basis as segment replacement, that is, the least recently used segments are trickled first . . . . *Id*. at 6:53-60. In Hamstra, empty segments are created by either first initialization of the cache before segments in cache memory are filled or by the trickle down functionality of the system. According to Hamstra, *if* there are no empty segments in the cache memory, *then* some of the segments in the cache memory are deleted by the system. However, if there *are* empty segments in the cache, then there is no need to delete any segments and the requested segments are transfered from a disk into those empty segments. Therefore, the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		system described in Hamstra dynamically determines whether to delete a segment based on the availability of empty segments in the cache memory. As both Ish and Hamstra relate (1) to a system of cache maintenance using a least recently used method for replacement of cache segments organized in linked lists and (2) a system to automatically delete the best candidates for replacement prior to a call from a user, one of ordinary skill in the art would understand how to use the Hamstra patent's dynamic decision on whether to perform a segment deletion based on the availability of empty segments in a cache memory when implementing a cache maintenance system such as Ish. Moreover, one of ordinary skill in the art would recongize that it would improve similar systems and methods in the same way. The fundamental goal of a cache memory, as described by both Ish and Hamstra, is to provide users fast access to frequently used data. *Ish et al.*, U.S. Patent No. 5,778,430 at 1:6-30; *Hamstra et al*, U.S. Patent No. 4,530,054 at 1:22-31. This is achieved by insuring that data residing in the cache is the most recently and frequently used data. A person skilled in the art would appreciate that the technique of replacing least recently used data segments from cache memory upon an insertion of a new data segment can be expanded to include dynamic decision making functionality to delete segments only when no empty segments are available in the cache. One of ordinary skill in the art would recognize that the result of combining the system disclosed in Ish with the logic of dynamically determining whether a deletion is necessary upon a user call as disclosed in Hamstra would further promote the goals of a cache memory. For example, one such benefit would include creating faster access to data for the user – the additional step of a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		deletion would only take place if there was no empty space in the cache memory. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Ish to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Ish with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Ish is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		To the extent that dynamically determining a maximum number of expired records is not disclosed by Ish and Hamstra, Ish combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG

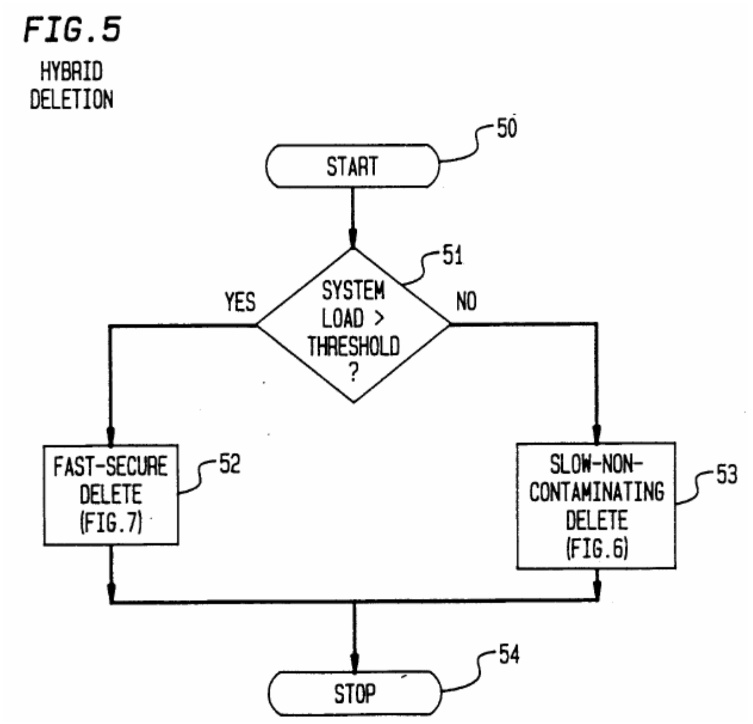
EXHIBIT B-5

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. (“Ish”) alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. (“Hamstra”)
		to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Ish and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Ish nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Ish and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Ish with the means for

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Ish and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Ish with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Ish with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Ish with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Ish with Thatte. Alternatively, it would also be obvious to combine Ish with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5.  FIG.5 HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Ish and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		patent's deletion decision procedure with Ish would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Ish and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Ish with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Ish and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Ish would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Ish and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Ish to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Ish with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Ish in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Ish. For example, both Linux 2.0.1 and Ish describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		`rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		`rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Ish discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Ish also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Ish et al.*, U.S. Patent No. 5,778,430 at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id.* at 3:30-43.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e.,

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Ish discloses accessing a linked list of records. Ish also discloses accessing a linked list of records having same hash address. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Ish discloses identifying at least some of the automatically expired ones of the records. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE,

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.
[3c] removing at least some of the automatically expired records from the linked list when the linked	[7c] removing at least some of the automatically expired records from the linked list when the linked	Ish discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
list is accessed.	list is accessed, and	storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id*. at 3:30-43 Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.

EXHIBIT B-5

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Ish discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Ish discloses an invention that "is implemented in a storage subsystem having, preferably, an array of selectively accessible direct access storage devices (disks), a processor, program memory, cache memory, and non-volatile memory and is responsive to commands received from at least one external source." *Id.* at 3:24-29. "In response to commands received from the external source, i.e., WRITE, READ, the storage system transfer data, preferably organized as blocks, to/from the direct access devices to/from the external source, as indicated. In order to speed access to the data, the blocks are held in an intermediary cache—when possible. Blocks which are the subject of a READ request and present in the cache are transferred directly from the cache to the external source. Conversely, Blocks which are the subject of a READ request and are not present in the cache, are first transferred from the direct access devices to the cache. Finally, blocks which are the subject of a WRITE request are stored in the cache, and subsequently flushed to the direct access devices at a convenient time." *Id.* at 3:30-43. Furthermore, Ish discloses that "[t]he method employs a hashing function which takes as its input a block number and outputs a hash index into a hash table of pointers. Each pointer in the hash table points to a doubly-linked list of headers, with each header having a bit map wherein the bits contained in the map identify whether a particular block of data is contained within the cache. Upon entry into the hash table, the linked headers are sequentially searched. If

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		no header is found that contains the particular block, a cache "miss" occurs, at which point in time an available space is made within the cache to hold the block and the block is subsequently retrieved from the direct access device and stored within the cache." *Id*. at 3:47-59. Furthermore, Ish discloses that "[e]ach header contains information is particularly relevant to the implementation of the replacement policy. As show in Fig. 4b, the Frequency member 406 indicates how many times the particular cache line has been accessed since it was placed in the cache. The Timestamp member 407 indicates the time of the last access of the cache line. The position within the head of the header point to a cache line is determined by its Frequency member 406 and its Timestamp member 407. In accordance with the present invention, the replacement head is modified to keep the best candidate for replacement at the root of the heap." *Id*. at 7:16-26. Moreover, Ish discloses that "structures have been incorporated within the present invention which further enhance its performance and advances over the prior art . . . . the cache header 403 contains a bitmap, DirtyMap 409, which like bitmap, ValidMap 408, has a bit in the bitmap for each block of data contained within the cache line associated by the header. The bits contained in DirtyMap 409 identify those blocks of data which have been modified, i.e., new data has been written into them, and have not yet been written out "flushed" to the direct access storage devices. Such dirty block could pose a performance problem for a cache system because if such a block were part of a cache line identified as a most likely candidate for replacement, the entire cache line would have be written to the direct access device BEFORE the cache line was replaced by new blocks, thereby degrading performance of the overall cache system. The presence of the DirtyMap 409, however, permits a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		flushing daemon process to continuously operate, flushing direct blocks to the direct access storage devices BEFORE the cache line in the block requires replacement. The daemon sequentially checks the headers identified by the array of points (implicit heap) until a first dirty cache line is found and flushed. In effect, this operation finds the first, direct candidate for replacement and then flushes it." *Id*. at 9:2-26.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Ish alone or in combination with Hamstra discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It is inherent to the Ish disclosure that upon a call to READ or WRITE, if there is empty space available in the, created by either (1) the initialization of the cache when segments within the cache have not yet been filled or (2) continuous flushing of the cache to remove the best candidates for replacement, blocks are transferred from a direct device to the cache without a deletion. If, however, empty space is not available, then the necessary space is made available within the cache by performing a delete. In order to promote the goals of a cache memory system, an efficient system would not make a deletion unless it was necessary, because the system would like to keep the cache filled to make retrieval for the user as fast as possible. Therefore, Ish inherently incorporates a dynamic decision making process in determining whether to delete a block within the cache upon a call to READ or WRITE. To the extent it is not inherent, it would be obvious to combine Ish with Hamstra. For example, as summarized in Hamstra:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		If the data is not resident in the cache memory then it is staged by segments from a disk, placed in the cache memory, and sent to the host. However, this *may* require that *some of the segments* in the cache memory be replaced by the segments from the disks. *Hamstra et al.*, "Processor-Addressable Timestamp for Indicating Oldest Written-to-Cache Entry Not Copied Back to Bulk Memory," U.S. Patent No. 4,530,054 (issued Jul. 16, 1985) at 1:36-41 (emphasis added). [W]hen a new segment or segments has to be brought from a device 104 to the cache memory 106 *and* there are no empty segments in the cache memory 106 then *some of the segments* resident in the cache memory 106 must be deleted or removed therefrom in order to make room for the new segments. *Id*. at 6:29-34 (emphasis added). [W]hen the SCU 100 has no other work to do it may search the SDT [– the segment descriptor (SDT) contains an entry for each segment of data resident in the cache memory, similar to a hash table (*Id*. at 5:33-35) –] to locate segments which have been written to the cache and trickle these segments to the devices 104 thus making space available in cache 106 for additional segments. The trickling of written-to segments takes place on the same basis as segment replacement, that is, the least recently used segments are trickled first . . . . *Id*. at 6:53-60. In Hamstra, empty segments are created by either first initialization of the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		cache before segments in cache memory are filled or by the trickle down functionality of the system. According to Hamstra, *if* there are no empty segments in the cache memory, *then some of the segments* in the cache memory are deleted by the system. However, if there *are* empty segments in the cache, then there is no need to delete any segments and the requested segments are transfered from a disk into those empty segments. Therefore, the system described in Hamstra dynamically determines whether to delete a segment based on the availability of empty segments in the cache memory. As both Ish and Hamstra relate (1) to a system of cache maintenance using a least recently used method for replacement of cache segments organized in linked lists and (2) a system to automatically delete the best candidates for replacement prior to a call from a user, one of ordinary skill in the art would understand how to use the Hamstra patent's dynamic decision on whether to perform a segment deletion based on the availability of empty segments in a cache memory when implementing a cache maintenance system such as Ish. Moreover, one of ordinary skill in the art would recongize that it would improve similar systems and methods in the same way. The fundamental goal of a cache memory, as described by both Ish and Hamstra, is to provide users fast access to frequently used data. *Ish et al.*, U.S. Patent No. 5,778,430 at 1:6-30; *Hamstra et al*, U.S. Patent No. 4,530,054 at 1:22-31. This is achieved by insuring that data residing in the cache is the most recently and frequently used data. A person skilled in the art would appreciate that the technique of replacing least recently used data segments from cache memory upon an insertion of a new data segment can be expanded to include dynamic decision making functionality to delete segments only

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		when no empty segments are available in the cache. One of ordinary skill in the art would recognize that the result of combining the system disclosed in Ish with the logic of dynamically determining whether a deletion is necessary upon a user call as disclosed in Hamstra would further promote the goals of a cache memory. For example, one such benefit would include creating faster access to data for the user – the additional step of a deletion would only take place if there was no empty space in the cache memory. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Ish to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Ish with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Ish is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Ish and Hamstra, Ish combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be

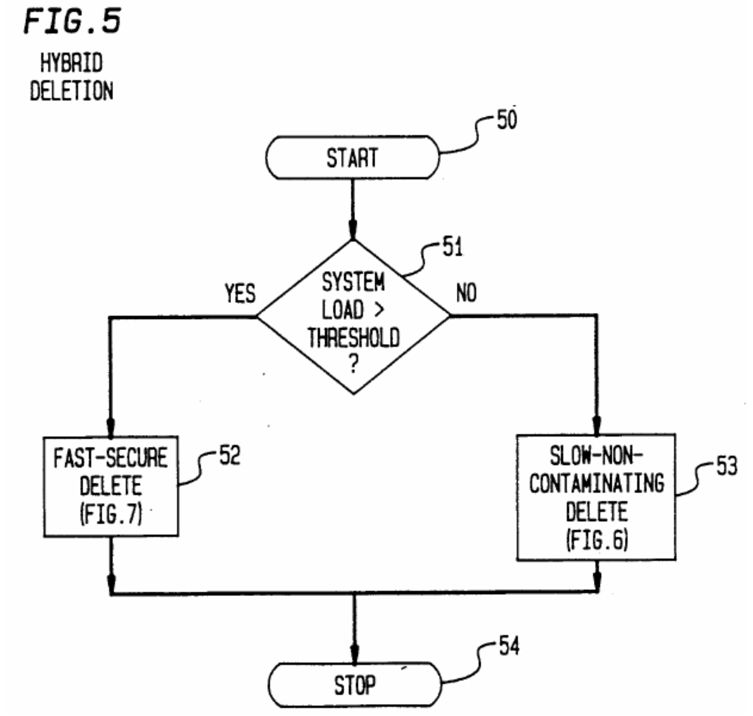
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. (“Ish”) alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. (“Hamstra”)**
		transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Ish and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Ish nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Ish and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Ish with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, as both Ish and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Ish with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Ish with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		skill in the art would recognize that combining Ish with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Ish with Thatte. Alternatively, it would also be obvious to combine Ish with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 (“The ’663 patent”). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as “deleted” and left in place. Later non-contaminating probes in the vicinity of such deleted record locations

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Ish and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Ish would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Ish and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Ish with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Ish and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Ish. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Ish would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Ish and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Ish to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Ish with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Ish can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

EXHIBIT B-5

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Ish in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Ish. For example, both Linux 2.0.1 and Ish describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX.

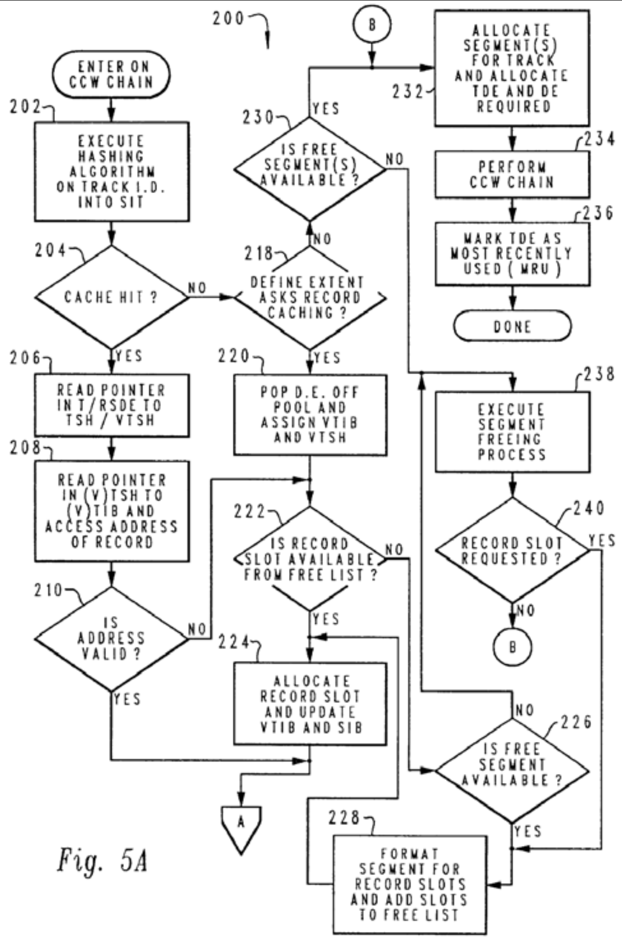
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,778,430 to Ish et al. ("Ish") alone and in combination with U.S. Patent No. 4,530,054 to Hamstra et al. ("Hamstra")**
		Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Beardsley discloses an information storage and retrieval system. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Beardsley et al.*, "Method and System for Dynamic Cache Allocation Between Record and Track Entries" U.S. Patent No. 5,991,775 (issued Nov. 23, 1999) at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Beardsley discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Beardsley also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id.* at 5:65 – 6:8. Interaction of the various data structures is best understood with reference to Figure 5A: *Id.* at Figure 5A.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES 218 DEFINE EXTENT ASKS RECORD CACHING ? 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB A 230 IS FREE SEGMENT(S) AVAILABLE ? B 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? 226 IS FREE SEGMENT AVAILABLE ? 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST Fig. 5A "To determine presence of a record or track in cache, step 202 is executed to

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

EXHIBIT B-6

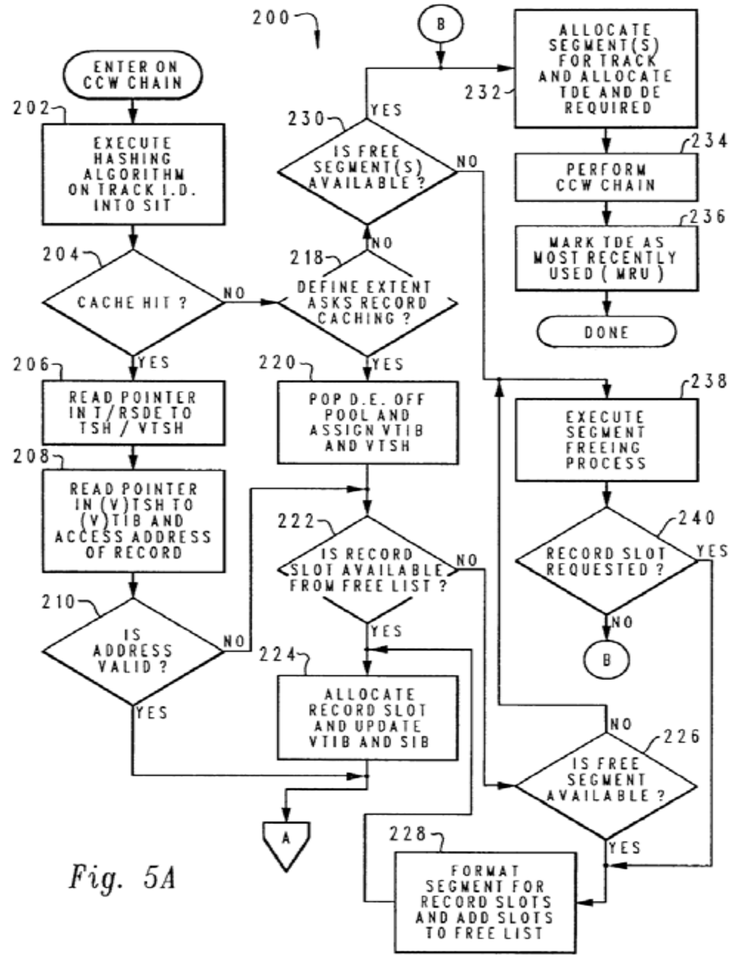
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Beardsley discloses a record search means utilizing a search key to access the linked list. Beardsley also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		record directory entries 86." *Id.* at 5:65 – 6:8.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Beardsley discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Beardsley also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id*. at 5:65 – 6:8. Interaction of the various data structures is best understood with reference to Figure 5A: *Id*. at Figure 5A.

EXHIBIT B-6

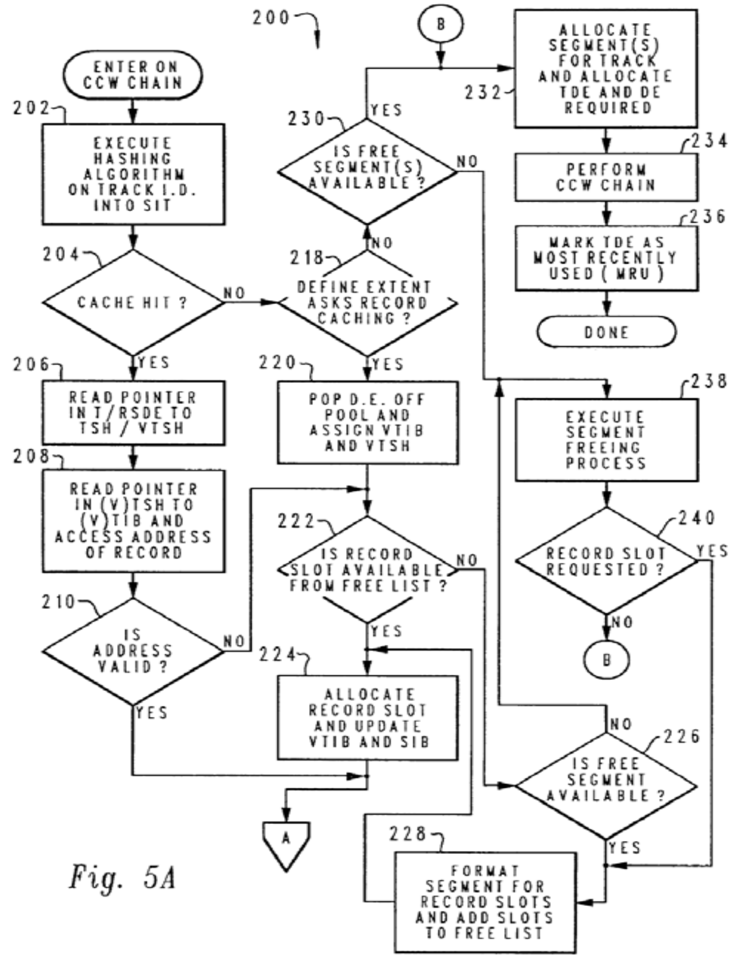
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO 218 DEFINE EXTENT ASKS RECORD CACHING ? NO YES 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? NO YES 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? YES NO B 226 IS FREE SEGMENT AVAILABLE ? NO YES *Fig. 5A* "To determine presence of a record or track in cache, step 202 is executed to

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Beardsley discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Beardsley also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id*. at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id*. at 5:65 – 6:8. Interaction of the various data structures is best understood with reference to Figure 5A: *Id*. at Figure 5A.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 YES IS FREE SEGMENT(S) AVAILABLE ? NO 218 NO DEFINE EXTENT ASKS RECORD CACHING ? 220 YES POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? NO YES 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? YES NO B NO 226 IS FREE SEGMENT AVAILABLE ? YES *Fig. 5A* "To determine presence of a record or track in cache, step 202 is executed to

US2008 1661512.4

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Beardsley discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. The dynamic decision-making process of the system is best understood with reference to Figure 5A: *Id*. at Figure 5A.

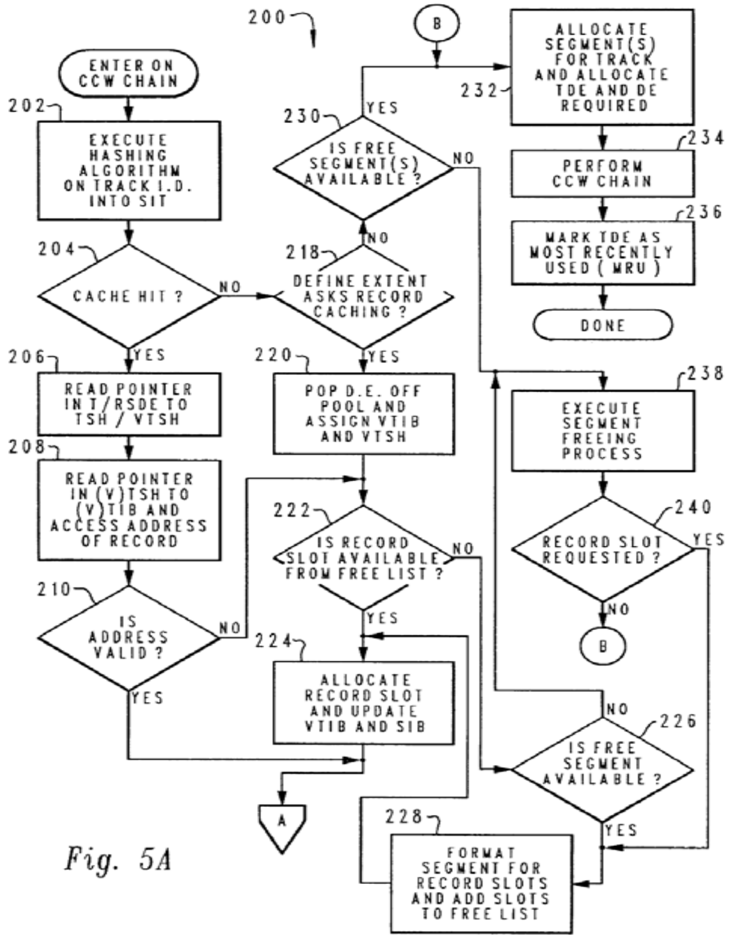
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO 218 DEFINE EXTENT ASKS RECORD CACHING ? 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? 226 IS FREE SEGMENT AVAILABLE ? *Fig. 5A*

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		For example, When the system confronts a cache "miss," the system follows "the NO branch from step 204, [and] step 218 is executed." *Id*. at 8:44-47. "The NO branch [from step 218] is followed when track caching has been requested in the define extent. Step 230 is then executed to determine if free segments are available for creation of a track slot. If YES, step 232 is executed to allocate segments from the track slot and to allocate a track slot directory entry and supplementary track directories entries as required. . . . If, however, at step 230, free segments were not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process as described below in relation to FIG. 6. Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:1-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33. In summary, if free segments are available, the system does not delete or free any allocated segments. If, however, free segments are not available, then the system executes the segment freeing process, which deallocates segments and places them in the free segment list for use in the call. Therefore, the determination of when to delete a segment is dynamically determined by whether a free segment exists. Additionally, it would have been obvious to combine Beardsley with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles to disclose an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. As summarized in Dirks, > each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. (“Beardsley”)**
		removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Beardsley and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Beardsley nothing more than the predictable use of prior art elements according to their established

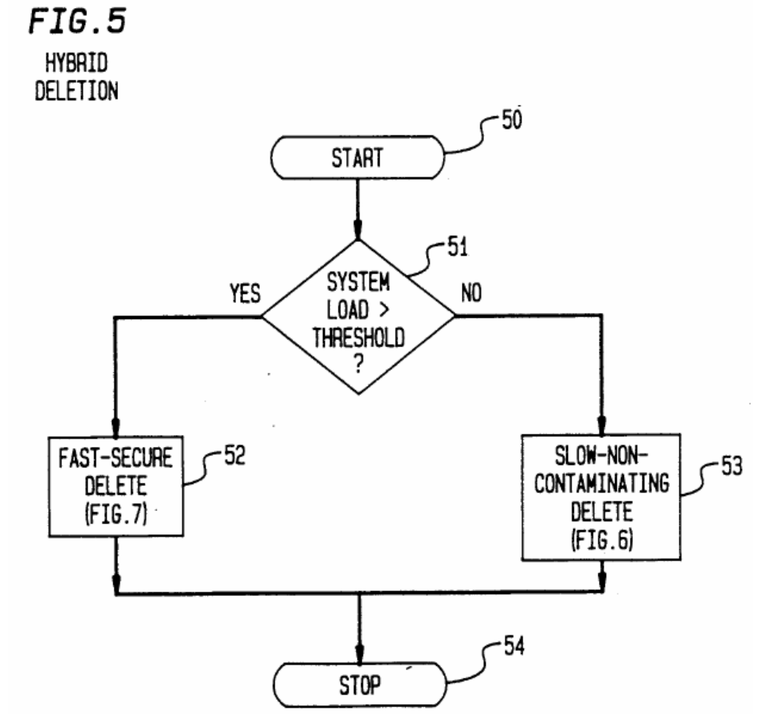
EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Beardsley and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Beardsley with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. Thatte, discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Beardsley with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Beardsley with Thatte and recognize the benefits of doing so. For example, the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		removal of expired records described in Beardsley can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Beardsley with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Beardsley with Thatte. Alternatively, it would also be obvious to combine Beardsleywith the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. As summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		FIG.5  SYSTEM LOAD > THRESHOLD ? FAST-SECURE DELETE (FIG.7) SLOW-NON-CONTAMINATING DELETE (FIG.6) *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Beardsley and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Beardsley with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Beardsley and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Beardsley and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Beardsley to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Beardsley with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Beardsleycan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Beardsley in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Beardsley. For example, both Linux 2.0.1 and Beardsley describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in

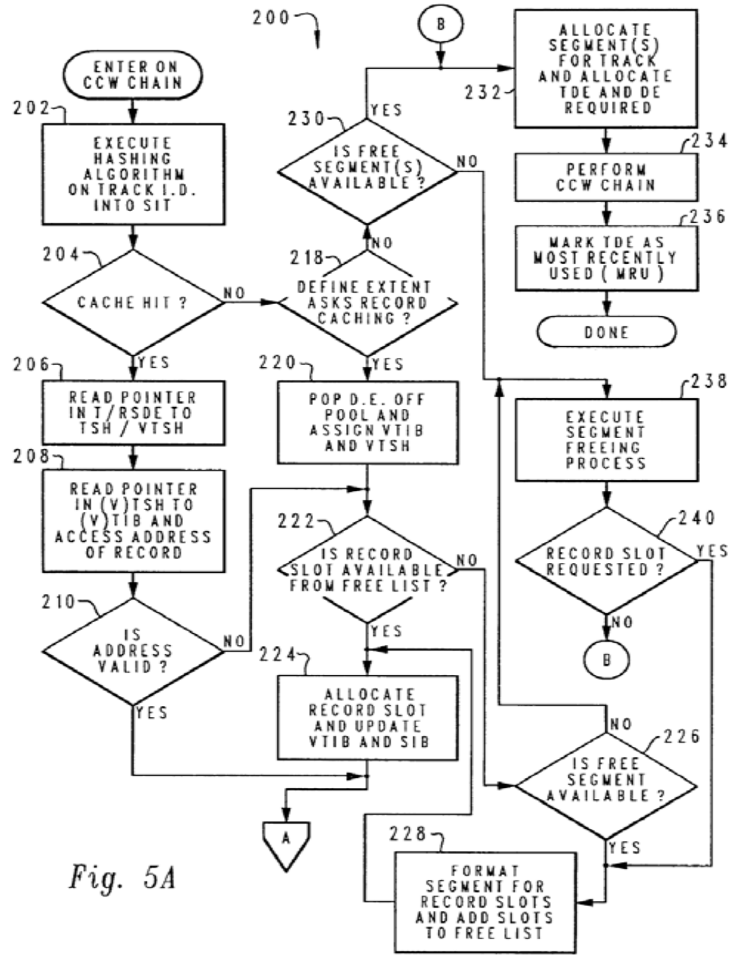
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list	7. A method for storing and retrieving information records using a hashing	To the extent the preamble is a limitation, Beardsley discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	expiring. Beardsley also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." Beardsley at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id.* at 5:65 – 6:8.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		Interaction of the various data structures is best understood with reference to Figure 5A: *Id*. at Figure 5A.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO 218 DEFINE EXTENT ASKS RECORD CACHING ? NO YES 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? NO YES 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? YES NO B 226 IS FREE SEGMENT AVAILABLE ? NO YES 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST *Fig. 5A* "To determine presence of a record or track in cache, step 202 is executed to

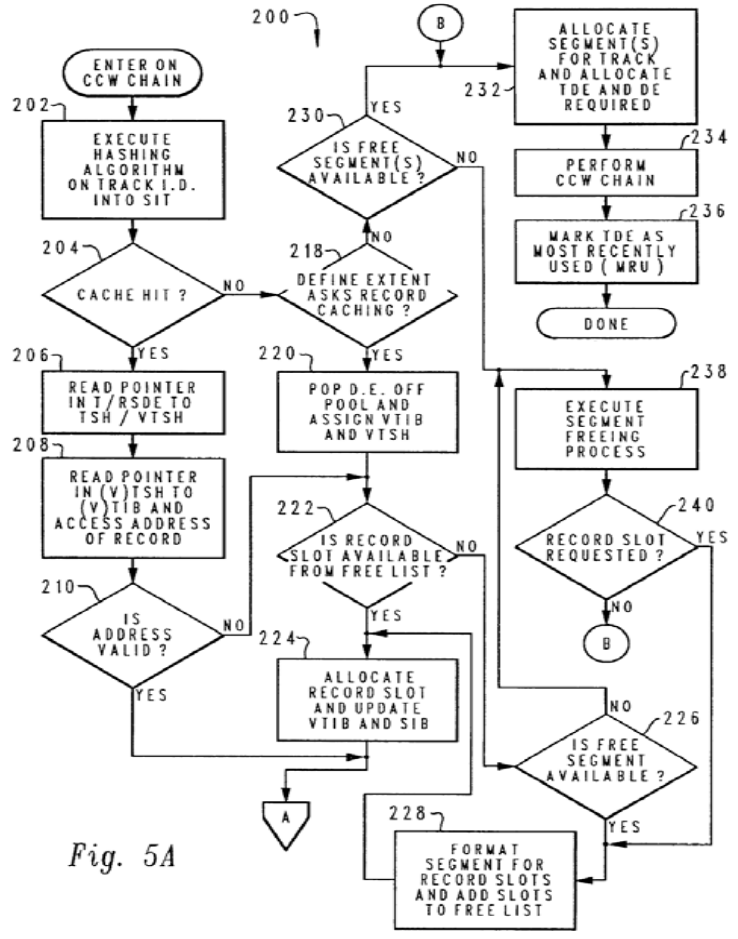
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Beardsley discloses accessing a linked list of records. Beardsley also discloses accessing a linked list of records having same hash address. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id*. at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id.* at 5:65 – 6:8.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Beardsley discloses identifying at least some of the automatically expired ones of the records. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id.* at 5:65 – 6:8. Interaction of the various data structures is best understood with reference to

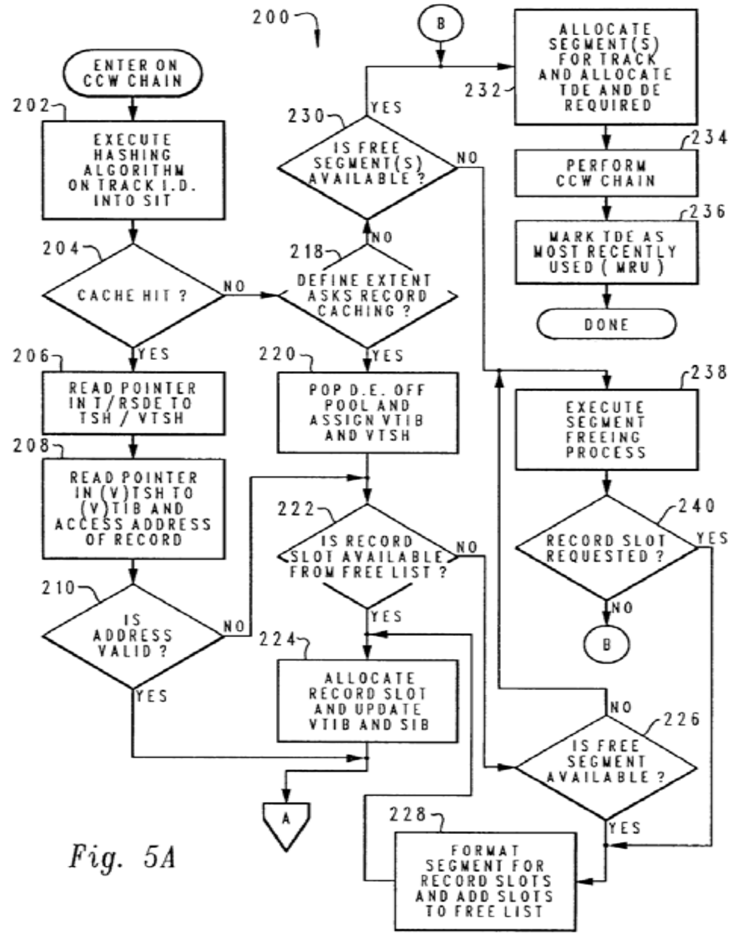
EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Figure 5A: *Id.* at Figure 5A.  200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A Fig. 5A B 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO 218 DEFINE EXTENT ASKS RECORD CACHING ? NO YES 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? NO YES 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? YES NO B NO 226 IS FREE SEGMENT AVAILABLE ? YES

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		"To determine presence of a record or track in cache, step 202 is executed to process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		will be understood by those skilled in the art that the process at block 238 is laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Beardsley discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id.* at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and

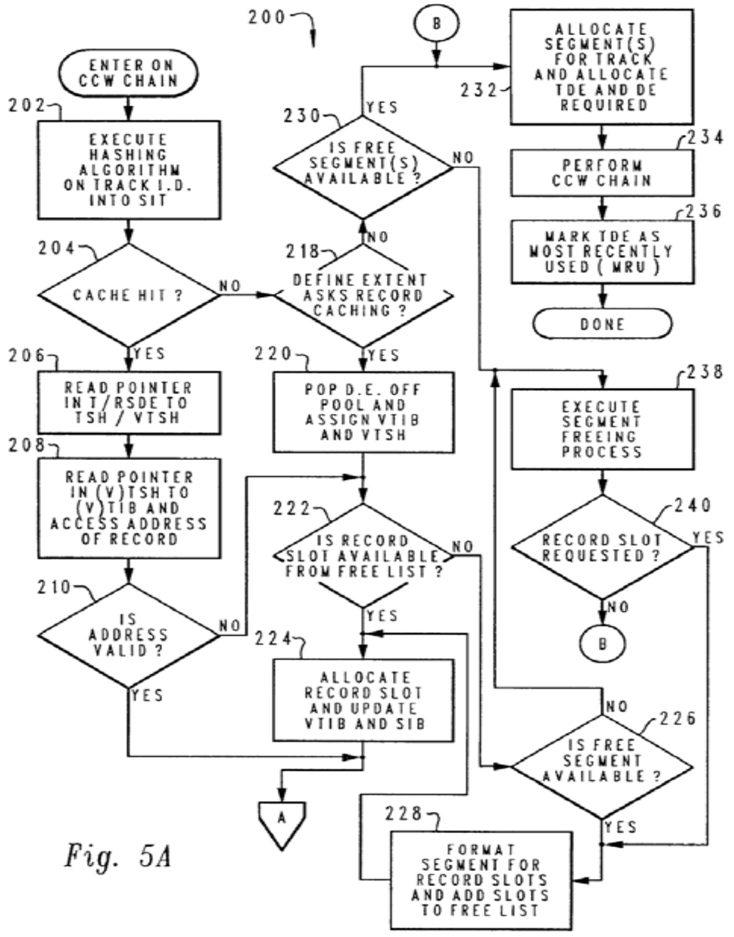
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		record directory entries 86." *Id.* at 5:65 – 6:8. Interaction of the various data structures is best understood with reference to Figure 5A: *Id.* at Figure 5A.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO 218 DEFINE EXTENT ASKS RECORD CACHING ? 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? 226 IS FREE SEGMENT AVAILABLE ? Fig. 5A "To determine presence of a record or track in cache, step 202 is executed to

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Beardsley discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Beardsley discloses that "[i]t is one object of the invention to provide an improved data processing system having two or more levels of data storage in a data storage system. It is still another object to provide a method of caching data from a lower level of storage on a higher level of storage both as individual records and in whole tracks." *Id.* at 3:13-18. "Storage controller 12 is internally divided into sections corresponding independent power supplies. Two sections are storage clusters 36 and 38 respectively. A third section includes a memory cache 58. . . . Cache 58 provides storage for frequently accessed data and for the buffering functions in order to provide similar response times for cache writes and cache reads." *Id*. at 5:1-9. Furthermore, Beardsley discloses that a "[s]catter index table 80 is used to quickly determine whether data exists in cache or not. A hashing algorithm is processed by a microcomputer to convert a device number and physical address to a scatter index table entry. Hashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts. Within the scatter index table are a plurality of indices to track directory entries 82 and record directory entries 86." *Id*. at 5:65 – 6:8.

EXHIBIT B-6

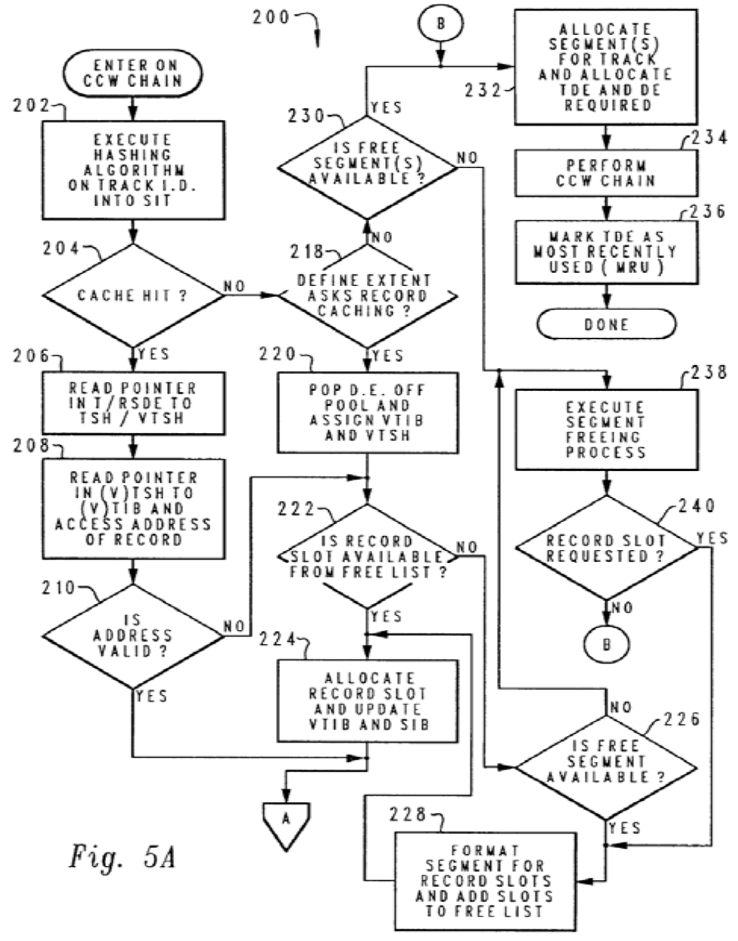
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Interaction of the various data structures is best understood with reference to Figure 5A: *Id*. at Figure 5A.  200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A 218 DEFINE EXTENT ASKS RECORD CACHING ? NO YES 230 IS FREE SEGMENT(S) AVAILABLE ? YES NO B 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? NO YES 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? YES NO B 226 IS FREE SEGMENT AVAILABLE ? NO YES *Fig. 5A*

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		"To determine presence of a record or track in cache, step 202 is executed to process a hashing algorithm on the direct access storage device and physical address for a track. The hashing algorithm will return an offset into the scatter index table 80 which in turn provides an index to a track directory entry or record directory entry, if present. Return of such an index is taken by step 204 to be a cache hit." *Id*. at 8:3-10. In the case of a cache miss, the steps of the process responsive to non-occurrence of a cache hit follow the NO branch from step 204. Step 218 is then executed." *Id*. at 8:44-47. Furthermore, Beardsley discloses that if "at step 230 [] free segments [are] not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process . . . . Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:11-39.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		"[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required." *Id*. at 10:28-33.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Beardsley discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. The dynamic decision-making process of the system is best understood with reference to Figure 5A: *Id*. at Figure 5A.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 200 ENTER ON CCW CHAIN 202 EXECUTE HASHING ALGORITHM ON TRACK I.D. INTO SIT 204 CACHE HIT ? NO YES 206 READ POINTER IN T/RSDE TO TSH / VTSH 208 READ POINTER IN (V)TSH TO (V)TIB AND ACCESS ADDRESS OF RECORD 210 IS ADDRESS VALID ? NO YES A B 230 YES IS FREE SEGMENT(S) AVAILABLE ? NO 218 DEFINE EXTENT ASKS RECORD CACHING ? 220 POP D.E. OFF POOL AND ASSIGN VTIB AND VTSH 222 IS RECORD SLOT AVAILABLE FROM FREE LIST ? 224 ALLOCATE RECORD SLOT AND UPDATE VTIB AND SIB 228 FORMAT SEGMENT FOR RECORD SLOTS AND ADD SLOTS TO FREE LIST 232 ALLOCATE SEGMENT(S) FOR TRACK AND ALLOCATE TDE AND DE REQUIRED 234 PERFORM CCW CHAIN 236 MARK TDE AS MOST RECENTLY USED (MRU) DONE 238 EXECUTE SEGMENT FREEING PROCESS 240 RECORD SLOT REQUESTED ? 226 IS FREE SEGMENT AVAILABLE ? Fig. 5A For example, when the system confronts a cache "miss," the system follows

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		"the NO branch from step 204, [and] step 218 is executed . . . . the NO branch [from step 218] is followed when track caching has been requested in the define extent. Step 230 is then executed to determine if free segments are available for creation of a track slot. If YES, step 232 is executed to allocate segments from the track slot and to allocate a track slot directory entry and supplementary track directories entries as required. . . . If, however, at step 230, free segments were not available for caching the track, the NO branch is followed to step 238. . . . Step 238 represents execution of a segment freeing process as described below in relation to FIG. 6. Upon completion of step 238 segments will be available for allocation to record caching or track caching. . . . Execution of the segment freeing process represented by block 238 should rarely be required to make record slots available. A record slot freeing process is executed upon completion of each access to a record. Returning to step 216 and following the YES branch to step 242, the record slot freeing process is initiated. At step 242 the record slot just accessed is stamped as most recently used in the appropriate segment information block on the local most recently use/least recently used list of segment information block. The record slot time stamp is also updated. Next, the segment information block is accessed to determine the least recently used record in the segment information block. At step 246, the time stamp of the least recently used record slot is compared with the global time stamp 94. . . . [i]f the least recently used record slot time stamp is older than the global time stamp 94 the record slot is freed." *Id*. at 9:1-39. "[A]t step 314 the segment is deallocated and placed in the free segment list. It will be understood by those skilled in the art that the process at block 238 is laid out for to illustrate freeing of a single segment, but execution can be repeated to free several segments if required."*Id*. at 10:28-33.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		In summary, if free segments are available, the system does not delete or free any allocated segments. If, however, free segments are not available, then the system executes the segment freeing process, which deallocates segments and places them in the free segment list for use in the call. Therefore, the determination of when to delete a segment is dynamically determined by whether a free segment exists. Additionally, it would have been obvious to combine Beardsley with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles to disclose dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. As summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of

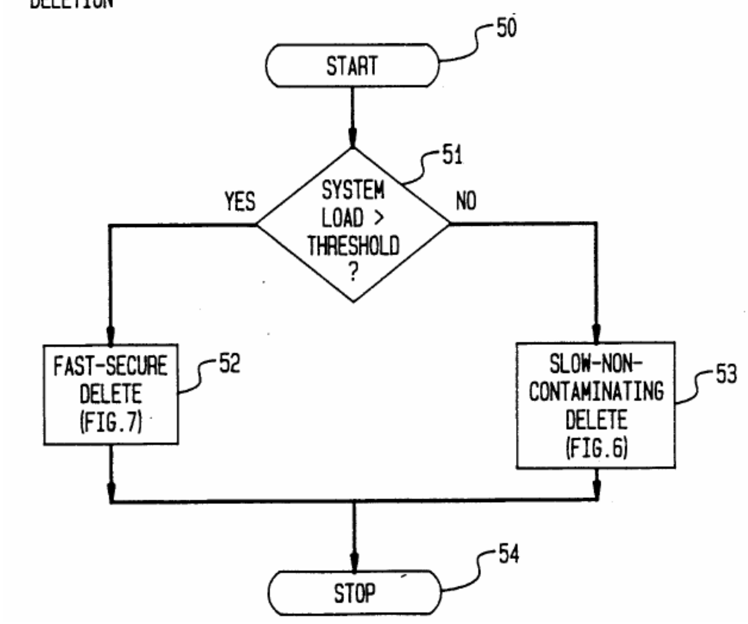
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Beardsley and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		examine during each step of the sweeping process with Beardsley and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Beardsley with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. Thatte, discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Beardsley with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Beardsley with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Beardsleycan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Beardsley with the teachings of Thatte would solve this problem by

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Beardsley with Thatte. Alternatively, it would also be obvious to combine Beardsley with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. As summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Beardsley and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Beardsley with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Beardsley and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Beardsley to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Beardsley with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Beardsley can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Beardsley in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Beardsley. For example, both Linux 2.0.1 and Beardsley describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`.

EXHIBIT B-6

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Bishop discloses an information storage and retrieval system. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, "System and Method for Distributed Object Resource Management" U.S. Patent No. 5,765,174 (issued Jun. 9, 1998) at 2:9-18.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Bishop discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Bishop also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		corresponding ones of the loaded programs." *Id*. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220." *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, "Methods and Apparatus for Information Storage and Retrieval Using a Hashing Technique with external chaining and On-The-Fly Removal of Expired Data," U.S. Patent No. 5,893,120 (issued Apr. 6, 1999) at 1:34-2:6 ("the '120 patent"). Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, "Method and System for Dynamic Cache Allocation Between Record

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		and Track Entries," U.S. Patent No. 5,991,775 (issued Nov. 23, 1999) at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Bishop discloses a record search means utilizing a search key to access the linked list. Bishop also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Bishop discloses that in an "aspect of the invention, a portion of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220. *Id.* at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44 Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120at 1:34-2:6. Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
[1c] the record search means including a means for identifying and	[5c] the record search means including means for identifying and removing	Bishop discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Bishop also discloses the record search

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	at least some expired ones of the records from the linked list of records when the linked list is accessed, and	means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id.* at 5:29-44 "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		recently used after the linker images referenced by the primary linker cache 220. *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Bishop and the admitted prior art of the '120 patent show that

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		resolution.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Bishop discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Bishop also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with

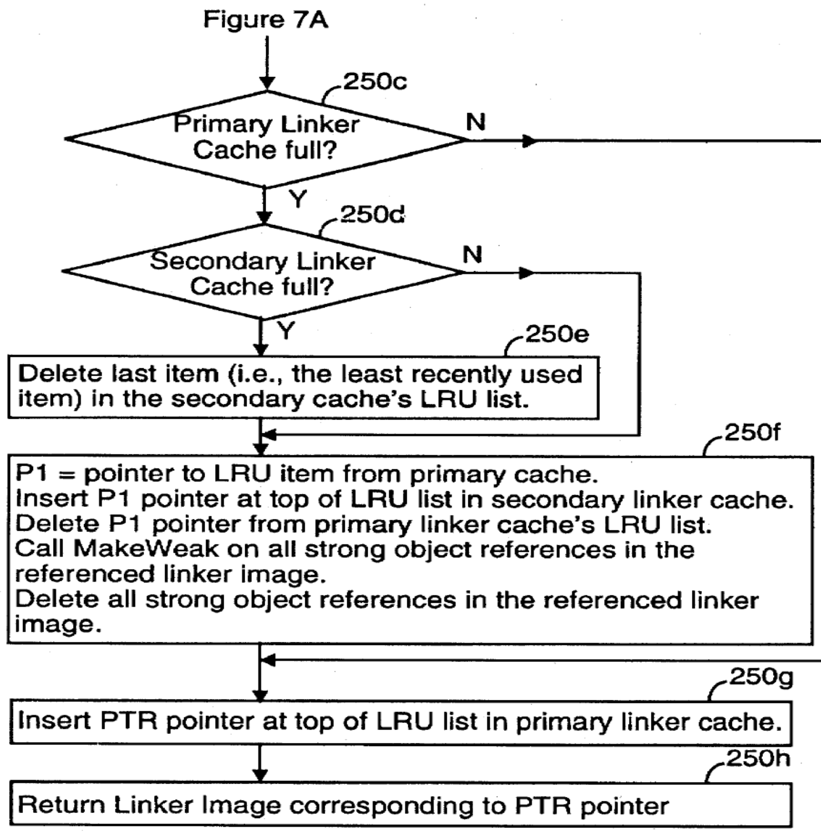
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220. *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly load program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Bishop discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Bishop discloses that "the 20 or so most recently used linker images are referenced by the LRU list 226 stored in the primary linker cache 220, while the next 200 or so most recently used linker images are referenced by the LRU list 230 stored in the secondary linker cache 232. When a new linker image is generated by the modified linker procedure 234, that causes all items in the primary linker cache's LRU list 226 to be moved down one position in the list. *If the primary linker cache 226 was full* before the new linker image was generated, then the object referenced to least recently used of the linker images in the primary linker cache will be moved into the secondary linker cache, and all the strong object references in the moved linker image are replaced by weak object references. When a linker image is moved into the secondary cache, the objects referenced by the linker image are not automatically deleted, because other entities in the computer system may also be referencing those same objects. For instance, if a user process is still executing the program corresponding to the moved linker image, all the objects, referenced by the moved linker image will have strong object references held by that user process." *Bishop*, U.S. Patent No. 5,765,174 at

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		8:34-51 (emphasis added). The dynamic decision-making process of the system is best understood with reference to Figure 7B: *Id*. at Figure 7B.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 Figure 7A 250c Primary Linker Cache full? N Y 250d Secondary Linker Cache full? N Y 250e Delete last item (i.e., the least recently used item) in the secondary cache's LRU list. 250f P1 = pointer to LRU item from primary cache. Insert P1 pointer at top of LRU list in secondary linker cache. Delete P1 pointer from primary linker cache's LRU list. Call MakeWeak on all strong object references in the referenced linker image. Delete all strong object references in the referenced linker image. 250g Insert PTR pointer at top of LRU list in primary linker cache. 250h Return Linker Image corresponding to PTR pointer **FIGURE 7B** "Steps 250*c* through 250*f* of the modified linker procedure service to create room in the primary linker cache's LRU list 226, *if such room is needed*, for a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		reference to the linker image for the user specified program. In particular, if the primary linker cache is full (step 250*c*), and the secondary linker cache is full (step 250*d*), the last item in the secondary linker cache's LRU list is deleted (step 250*e*). Then the last item in the primary linker cache's LRU list 226 is inserted at the top of the secondary linker cache's LRU list 230 and deleted from the primary linker cache's LRU list 226 (step 250*f*)." *Id*. at 8:60 – 9:3 (emphasis added). In summary, if there is space available in the primary linker cache or in the secondary linker cache, then the system does not delete any items from the cache. If, however, free space is not available in both the primary linker cache and the secondary linker cache, then the system deletes the last item (i.e., the least recently used item) in the secondary cache's LRU list to make room for the new item. Therefore, the determination of when to delete an item is dynamically determined by whether a free space exists within either the primary linker cache or the secondary linker cache. Additionally it would have been obvious to one of skill in the art to combine Bishop with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles to disclose an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their

EXHIBIT B-7

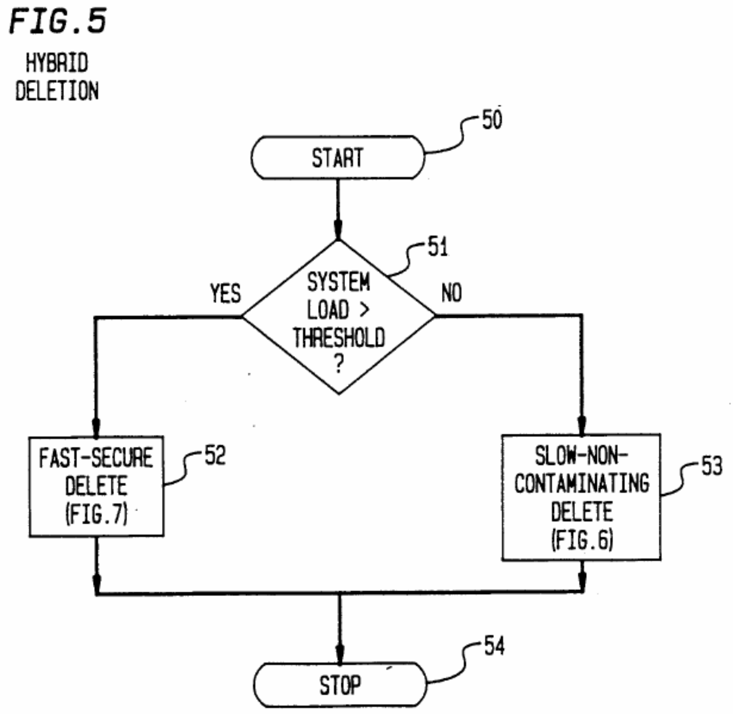
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Bishop and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Bishop nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Bishop and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Bishop with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. Thatte, discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Bishop with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Bishop can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Bishop with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Bishop with Thatte. Alternatively, it would also be obvious to combine Bishop with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Bishop and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Bishop would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Bishop and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Bishop with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles recite in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Bishop and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Bishop would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Bishop and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Bishop to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Bishop with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Bishopcan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Bishop in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Bishop. For example, both Linux 2.0.1 and Bishop describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function rt_garbage_collect_1 can remove from a linked list is different than the maximum number of records that the function rt_cache_add can remove from a linked list.

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Bishop discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Bishop also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id.* at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220. *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6 Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Bishop discloses accessing a linked list of records. Bishop also discloses accessing a linked list of records having same hash address. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220." *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id.* at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Bishop discloses identifying at least some of the automatically expired ones of the records. Bishop also discloses identifying at least some of the automatically expired ones of the records. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id.* at 5:29-44

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		"[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220." *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Bishop discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Bishop also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker image in the primary linker cache has strong object references to objects

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220." *Id*. at 7:61-67 – 8:1-3. "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding linker image 160 when all user program domains also relinquish their

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley* U.S. Patent No. 5,991,775 at 6:2-8. – it was well known in the prior art to have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Bishop discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Bishop discloses that in an "aspect of the invention, a portion of the computer's memory is set aside for a primary linker cache and a secondary linker image cache. Linker images, generated while loading programs for execution are stored in the primary and secondary linker caches. Each linker

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		image in the primary linker cache has strong object references to objects included in corresponding ones of the loaded programs, and each linker image in the secondary linker cache has weak object references to objects included in corresponding ones of the loaded programs." *Bishop*, U.S. Patent No. 5,765,174 at 2:9-18. Furthermore, Bishop discloses that "[t]he linker cache 170 has a limited capacity, typically sufficient to hold references to about fifty or so linker images 160. The linker 152 maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order (i.e., each time a linker image is used, it is moved to the head of the list). The hash table 171 is a conventional hash table used to quickly locate items in the LRU list 172." *Id*. at 5:29-44. "[I]n the preferred embodiment the linker cache 170 of FIG. 2 is replaced with a smaller primary linker cache 220 and an expanded secondary linker cache 222. The primary linker cache 220 includes a hash table 224 and an LRU list 226 that stores strong object references to the least recently used linker images 160, while the secondary linker cache 222 includes a hash table 228 and an LRU list 230 that stores strong object references to the linker images 160 least recently used after the linker images referenced by the primary linker cache 220." *Id*. at 7:61-67 – 8:1-3 "When the linker cache is full, and another linker image needs to be generated in response to a load program command, the reference in the LRU list 172 to the least recently used linker image is deleted and the resulting space is used to store a reference to a linker image for the newly loaded program. Deletion of an object reference in the LRU list 172 enables deletion of the corresponding

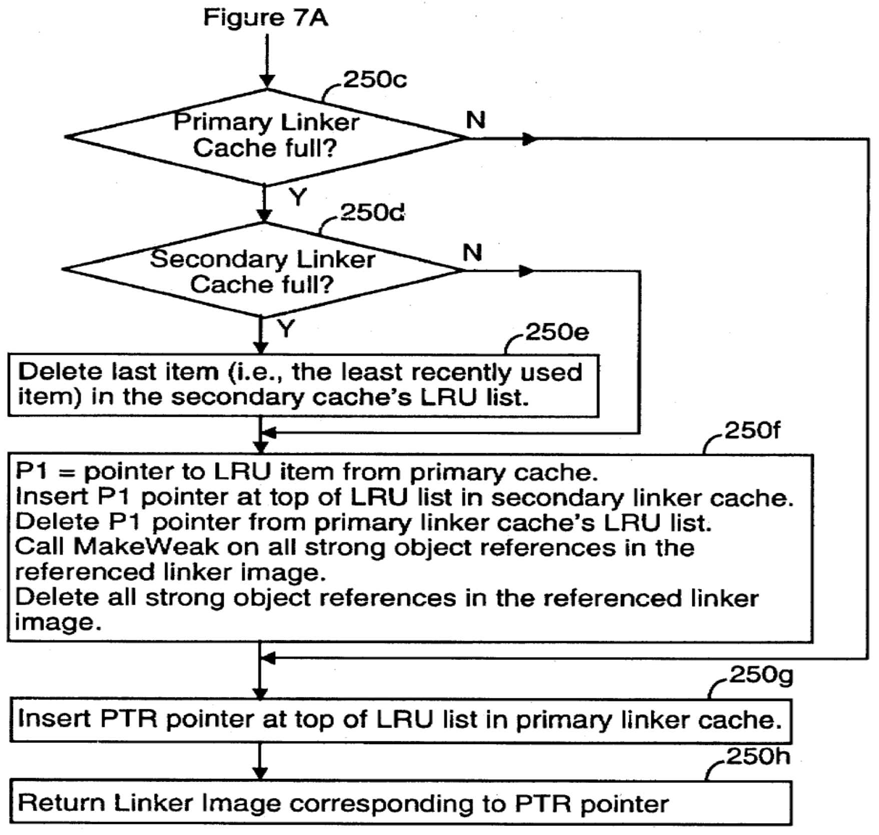
EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		linker image 160 when all user program domains also relinquish their references to the linker image" *Id*. at 5:35-44. Furthermore, Bishop inherently discloses that cached linker images are stored in a linked list, "the linker maintains a hash table 171 and a list 172 of the cached linker images in "least recently used" order . . . . The hash table 171 is a *conventional hash table* used to quickly locate items in the LRU list." *Id*. at 5:31-35 (emphasis added). Because conventional hash tables need a system by which to deal with the key collision problem and the most frequently used method in dealing with such collisions is with the use of a linked list, such inherent characteristic necessarily flows from the teachings of the applied prior art. To the extent that Bedrock argues that Bishop does not anticipate Claims 1 – 8 because linker images are not inherently stored in a linked list, it would have been obvious to one of ordinary skill in the art to store the linker images in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6 . Thus, Bishop and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Moreover, as disclosed in Beardsley – "[h]ashing functions are well known techniques used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley*

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		et al., "Method and System for Dynamic Cache Allocation Between Record and Track Entries," U.S. Patent No. 5,991,775 (issued Nov. 23, 1999) at 6:2-8. – it was well known in the prior art to have objects distributed by a hash function, such as those in Bishop, and then store the objects in a hash table using linked lists or external chaining. As both Bishop and Beardsley disclose systems and methods for allocating data structures in segments within a cache that are stored in hash tables with a least recently used methodology for replacement of old structures, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Bishop. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Bishop's linker cache being implemented with a hashing function using linked lists/external chaining. By way of further example, one of ordinary skill in the art would have combined linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. One such benefit, for example, is hash table collision resolution.
4. The method according to claim 3 further including the step of dynamically determining maximum	8. The method according to claim 7 further including the step of dynamically determining maximum	Bishop discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, Bishop discloses that "the 20 or so most recently used linker

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
number of expired ones of the records to remove when the linked list is accessed.	number of expired ones of the records to remove when the linked list is accessed.	images are referenced by the LRU list 226 stored in the primary linker cache 220, while the next 200 or so most recently used linker images are referenced by the LRU list 230 stored in the secondary linker cache 232. When a new linker image is generated by the modified linker procedure 234, that causes all items in the primary linker cache's LRU list 226 to be moved down one position in the list. *If the primary linker cache 226 was full* before the new linker image was generated, then the object referenced to least recently used of the linker images in the primary linker cache will be moved into the secondary linker cache, and all the strong object references in the moved linker image are replaced by weak object references. When a linker image is moved into the secondary cache, the objects referenced by the linker image are not automatically deleted, because other entities in the computer system may also be referencing those same objects. For instance, if a user process is still executing the program corresponding to the moved linker image, all the objects, referenced by the moved linker image will have strong object references held by that user process." *Bishop*, U.S. Patent No. 5,765,174 at 8:34-51.. The dynamic decision-making process of the system is best understood with reference to Figure 7B: *Id*. at Figure 7B.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 Figure 7A 250c Primary Linker Cache full? N Y 250d Secondary Linker Cache full? N Y 250e Delete last item (i.e., the least recently used item) in the secondary cache's LRU list. 250f P1 = pointer to LRU item from primary cache. Insert P1 pointer at top of LRU list in secondary linker cache. Delete P1 pointer from primary linker cache's LRU list. Call MakeWeak on all strong object references in the referenced linker image. Delete all strong object references in the referenced linker image. 250g Insert PTR pointer at top of LRU list in primary linker cache. 250h Return Linker Image corresponding to PTR pointer **FIGURE 7B** "Steps 250*c* through 250*f* of the modified linker procedure service to create

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		room in the primary linker cache's LRU list 226, *if such room is needed*, for a reference to the linker image for the user specified program. In particular, if the primary linker cache is full (step 250*c*), and the secondary linker cache is full (step 250*d*), the last item in the secondary linker cache's LRU list is deleted (step 250*e*). Then the last item in the primary linker cache's LRU list 226 is inserted at the top of the secondary linker cache's LRU list 230 and deleted from the primary linker cache's LRU list 226 (step 250*f*)." *Id.* at 8:60 – 9:3 (emphasis added). In summary, if there is space available in the primary linker cache or in the secondary linker cache, then the system does not delete any items from the cache. If, however, free space is not available in both the primary linker cache and the secondary linker cache, then the system deletes the last item (i.e., the least recently used item) in the secondary cache's LRU list to make room for the new item. Therefore, the determination of when to delete an item is dynamically determined by whether a free space exists within either the primary linker cache or the secondary linker cache. Additionally, it would have been obvious to one of ordinary skill in the art to combine Bishop with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles to disclose dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their

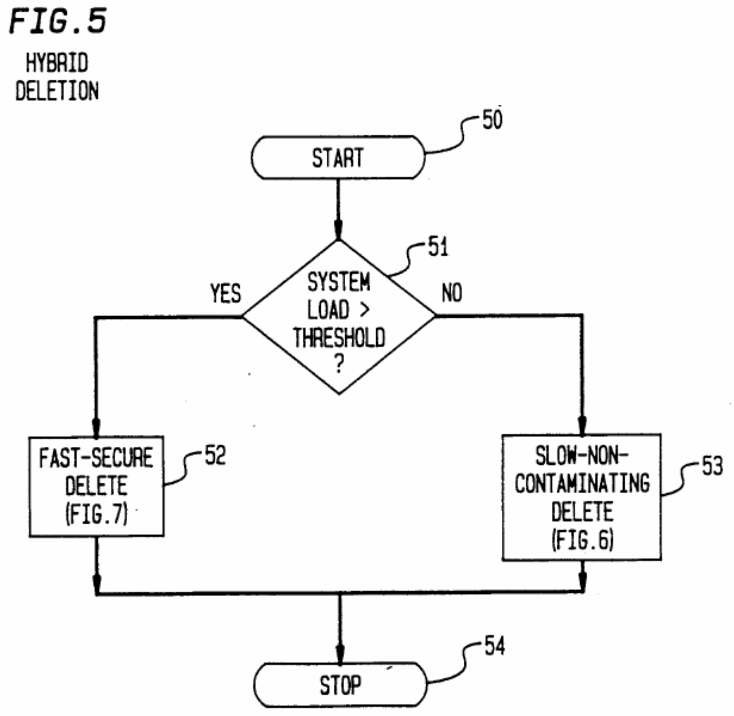
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Bishop and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Bishop would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Bishop and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Bishop with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte, discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		way. Additionally, one of ordinary skill in the art would recognize that the result of combining Bishop with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Bishop with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Bishopcan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Bishop with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Bishop with Thatte. Alternatively, it would also be obvious to combine Bishop with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Bishop and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Bishop would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Bishop and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Bishop with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Bishop and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Bishop. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Bishop would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Bishop and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Bishop to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Bishop with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Bishop can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

EXHIBIT B-7

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Bishop in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Bishop. For example, both Linux 2.0.1 and Bishop describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold RT_CACHE_SIZE_MAX. If the number of records in the hash table exceeds the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_cache_add invokes a function rt_garbage_collect. *See* Linux 2.0.1, route.c at lines 1341-1342. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect invokes a function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,765,174 to Bishop ("Bishop") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-7

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Cox discloses an information storage and retrieval system. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al*, "Promoting Local Memory Accessing and Data Migration in Non-Uniform Memory Access System Architectures," U.S. Patent No. 5,918,249 (issued: Jun. 29, 1999) at Abstract.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Cox in combination with Beardsley discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Cox also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Id*.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, "Method and System for Dynamic Cache Allocation Between Record and Track Entries," U.S. Patent No. 5,991,775 (issued Nov. 23, 1999) at 6:2-8 9 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, "Methods and Apparatus for Information Storage and Retrieval Using a Hashing Technique with external chaining and On-The-Fly Removal of Expired Data," U.S. Patent No. 5,893,120 (issued Apr. 6, 1999) at 1:34-2:6 ("the '120 patent"). Thus, Cox

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of	Cox in combination with Beardsley discloses a record search means utilizing a search key to access the linked list. Cox also discloses a record search means utilizing a search key to access a linked list of records having the same hash

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
	records having the same hash address,	address. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system.

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. (“Cox”) alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. (“Beardsley”)**
		hash table collision resolution and the promotion of efficient local memory accessing.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Cox in combination with Beardsley discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Cox also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Cox discloses “[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes.” *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that “*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts” *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing. Furthermore, as summarized in Cox: In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Cox in combination with Beardsley discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Cox also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley, *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8. one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing. Furthermore, as summarized in Cox: > In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

EXHIBIT B-8

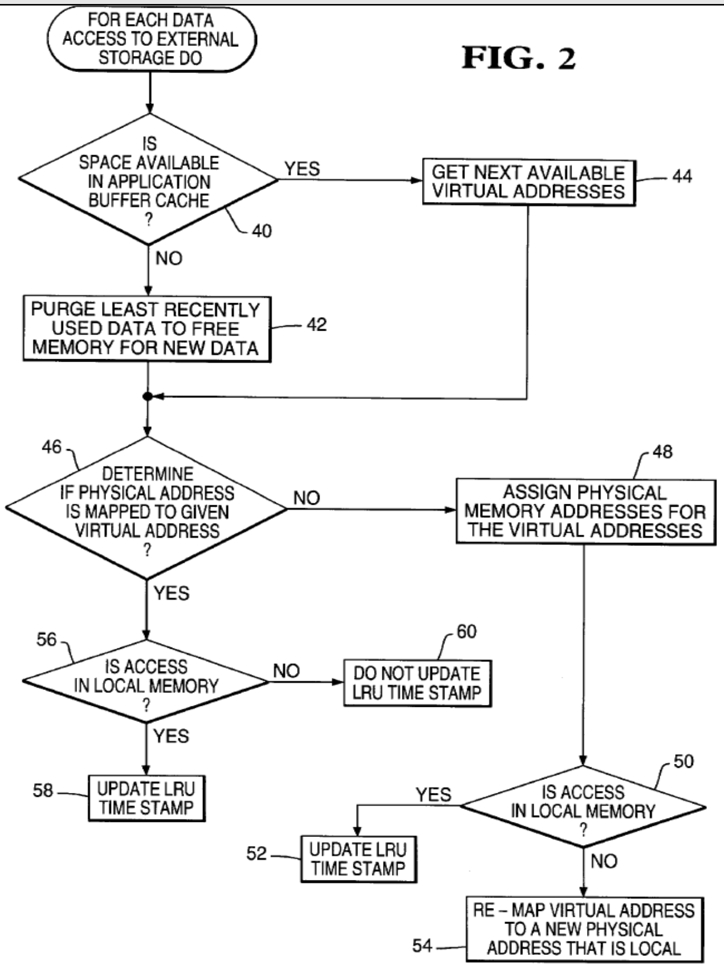
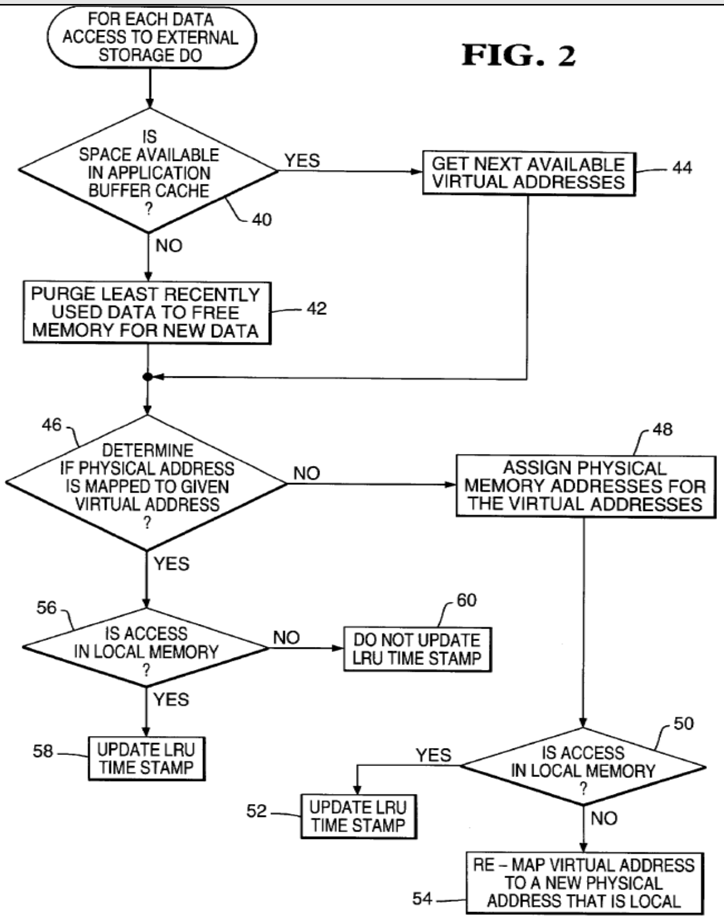
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? YES GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 40 NO PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO 48 ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES YES 56 IS ACCESS IN LOCAL MEMORY ? NO 60 DO NOT UPDATE LRU TIME STAMP YES 58 UPDATE LRU TIME STAMP 50 YES IS ACCESS IN LOCAL MEMORY ? 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 FIG. 2

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Cox discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, as summarized in Cox: In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		If the physical address is local to the node (step 50), the application determines if the local memory has sufficient capacity available to store the data, similar to step 40. If there is not enough memory space available in the buffer cache LRU data is purged, similar to step 42. Once the LRU data is purged or if space is available to store the data, the application process on node A retrieves, time stamps and stores the data in memory (step 52). *Id*. at 3:55-63. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? YES GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 40 NO PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO 48 ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES YES 56 IS ACCESS IN LOCAL MEMORY ? NO 60 DO NOT UPDATE LRU TIME STAMP YES 58 UPDATE LRU TIME STAMP 50 YES IS ACCESS IN LOCAL MEMORY ? 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 FIG. 2 In summary, if there is space available in the buffer cache, then the system does not purge any data from the cache. If, however, free space is not

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		available in the buffer cache, then the system purges the last item (i.e., the least recently used item) from the buffer cache based on the least recently mechanism used by the system to make room for the new data. Therefore, the determination of when to delete data is dynamically determined by whether free space exists within the buffer cache. Additionally, it would have been obvious to one of ordinary skill in the art to combine Cox and Beardsley with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles to disclose an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. (“Cox”) alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. (“Beardsley”)**
		will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks:

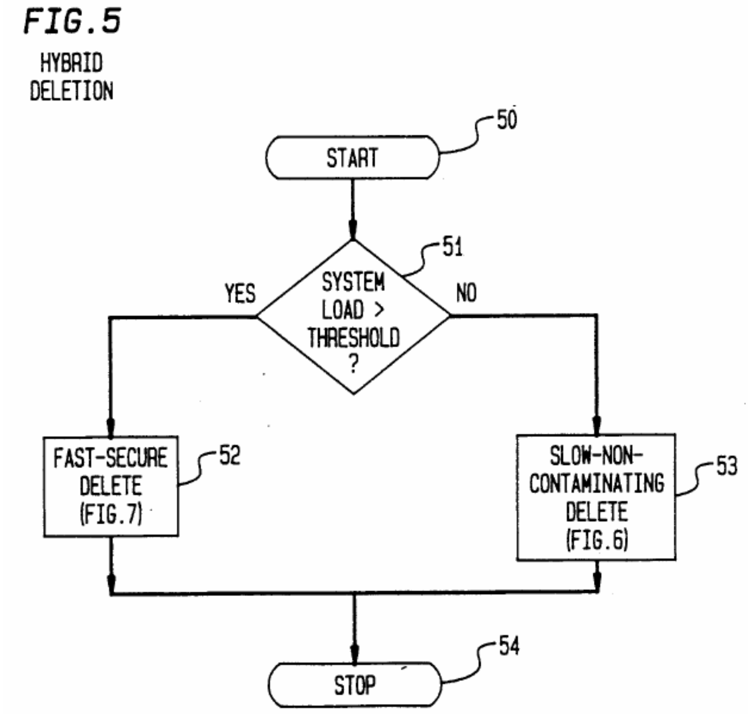
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Cox and Beardsley and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Cox and Beardsley nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Cox and Beardsley and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Cox and Beardsley with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox and Beardsley with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Cox and Beardsley with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Cox and Beardsley can be burdensome on the system, adding to the system's load and slowing down

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the system's processing. One of ordinary skill in the art would recognize that combining Cox and Beardsley with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Cox and Beardsley with Thatte. Alternatively, it would also be obvious to combine Cox and Beardsley with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Cox and Beardsley and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would have understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Cox and Beardsley would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Cox and Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Cox and Beardsley with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Cox and Beardsley and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Cox and Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Cox and Beardsley and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Cox and Beardsley to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Cox and Beardsley with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Cox and Beardsleycan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Cox in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Cox. For example, both Linux 2.0.1 and Cox describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		`rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-8

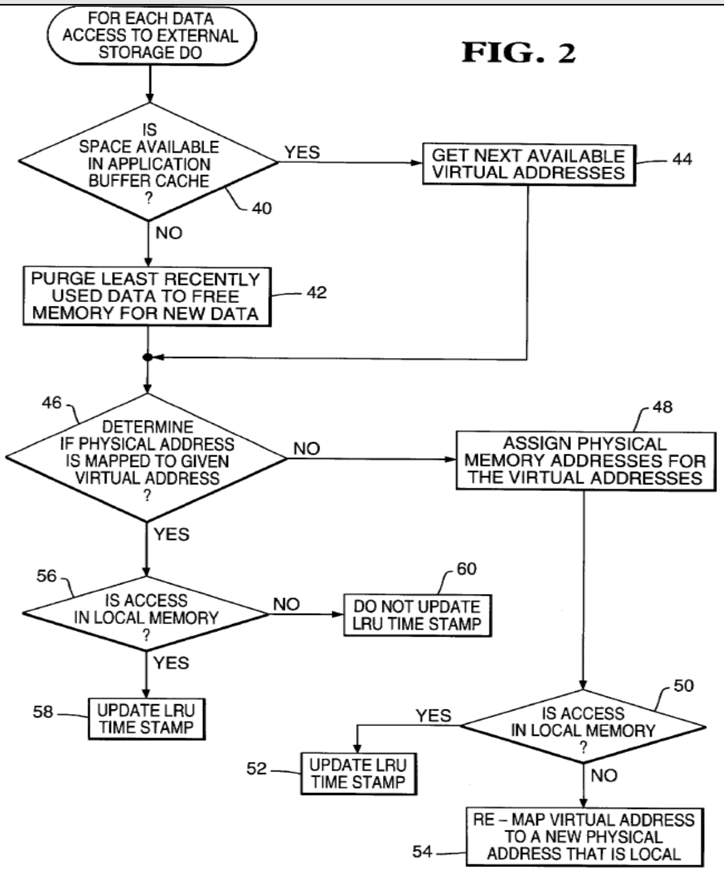
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Cox discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Cox also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing. Furthermore, as summarized in Cox: In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? YES GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 40 NO PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO 48 ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES YES 56 IS ACCESS IN LOCAL MEMORY ? NO 60 DO NOT UPDATE LRU TIME STAMP YES 58 UPDATE LRU TIME STAMP 50 YES IS ACCESS IN LOCAL MEMORY ? 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 FIG. 2
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Cox discloses accessing a linked list of records. Cox also discloses accessing a linked list of records having same hash address.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Cox discloses identifying at least some of the automatically expired ones of the records. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Cox discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Id*. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley*

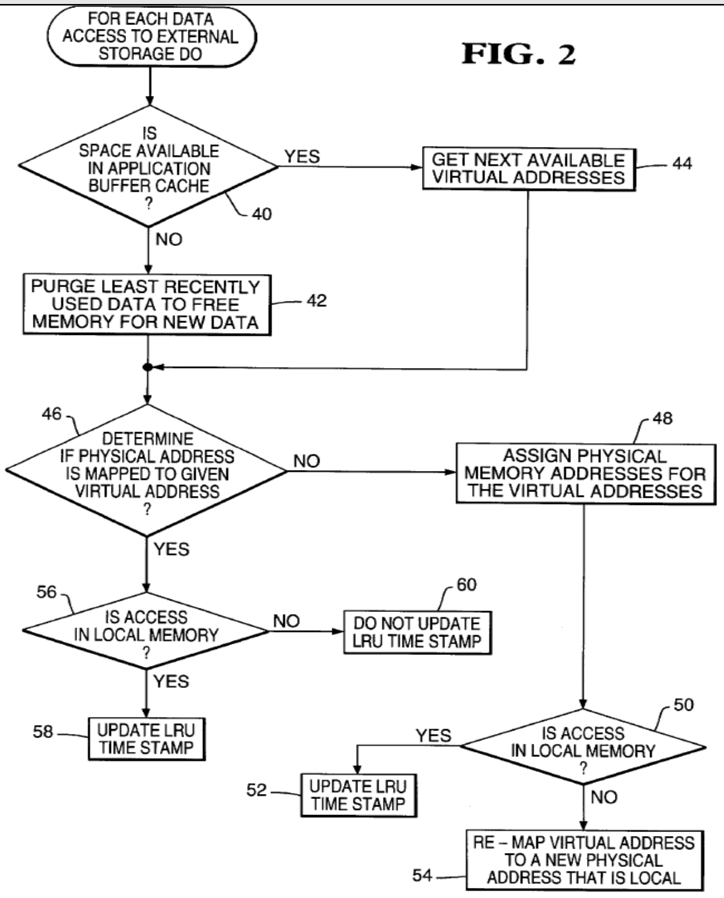
EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		et al., U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system. Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory accessing. Furthermore, as summarized in Cox: In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

EXHIBIT B-8

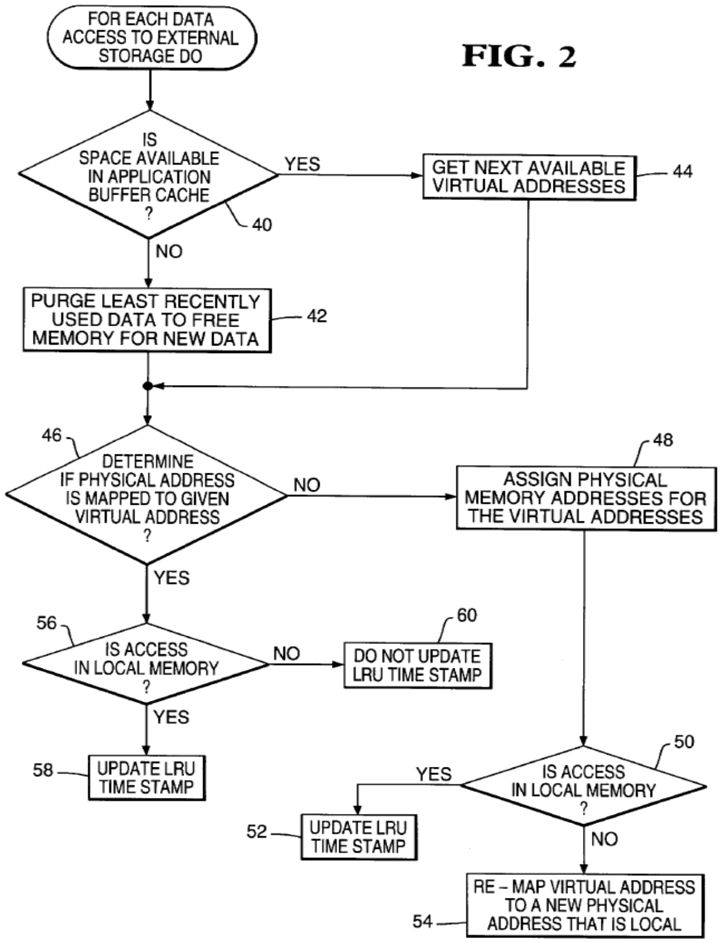
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? YES NO GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 40 PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO 48 ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES YES 56 IS ACCESS IN LOCAL MEMORY ? NO 60 DO NOT UPDATE LRU TIME STAMP YES 58 UPDATE LRU TIME STAMP 50 YES IS ACCESS IN LOCAL MEMORY ? 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 **FIG. 2**
	[7d] inserting, retrieving or deleting one of the	Cox discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
	records from the system following the step of removing.	For example, Cox discloses "[a] non-uniform memory accessing (NUMA) based multi-processor system that promotes local memory accessing and data migration to local processor nodes. The system includes mechanisms to re-map virtual and physical addresses to promote local memory accessing and implements a least recently used memory allocation mechanism to age non-local memory accesses out of memory to be re-read into local memory, which promotes data migration to local memory on processor nodes." *Cox et al.*, U.S. Patent No. 5,918.249 at Abstract. For example, Beardsley discloses that "*[h]ashing functions are well known techniques* used to randomly allocate locations in one address space to addresses of a second address space. Those skilled in the art will realize that the usual techniques of hashing chains are used to resolve conflicts" *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8 (emphasis added) – it was well known in the prior art to have objects distributed by a hash function, as demonstrated in Beardsley, and then store the objects in a hash table using linked lists or external chaining. For example, Cox discloses that physical memory addresses are re-assigned to corresponding local memory addresses of the processor node. *Cox et al.*, U.S. Patent No. 5,918,249 at 2:29-38. Since hashing involves allocating locations in one address space to addresses of a second address space, as disclosed in Beardsley *Beardsley et al.*, U.S. Patent No. 5,991,775 at 6:2-8., one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox to implement the re-mapping Cox requires in its system.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		Furthermore, with respect to the use of linked list to address the key collision problem, it would have been obvious to one of ordinary skill in the art to store data in a linked list. The admitted prior art in the background of the '120 patent discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Nemes*, U.S. Patent No. 5,893,120 at 1:34-2:6. Thus, Cox and the admitted prior art of the '120 patent show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. As both Cox and Beardsley disclose systems and methods for allocating data structures in segments within a cache for fast retrieval of data with a least recently used methodology for replacement of old data, one of ordinary skill in the art would understand how to combine the hashed page table with linked lists taught in Beardsley with Cox. Moreover, one of ordinary skill in art would recognize that it would improve similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox with Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. The result would simply be Cox's buffer cache memory being implemented with a hashing function using linked lists/external chaining as to way to re-map physical addresses to local memory addresses. By way of further example, one of ordinary skill in the art would have combined hashing with linked lists as taught by these references and one of ordinary skill in the art to the system disclosed in the admitted prior and would have seen the benefits of doing so. Two such benefits, for example, include hash table collision resolution and the promotion of efficient local memory

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		accessing. Furthermore, as summarized in Cox: > In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

EXHIBIT B-8

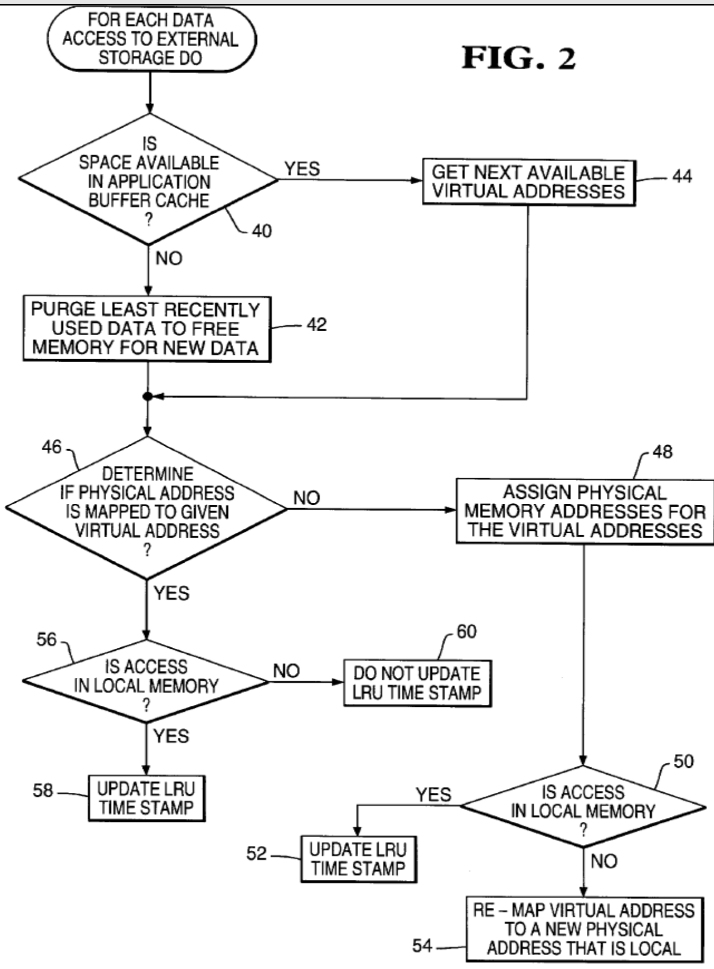
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? 40 YES GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 NO PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES 48 YES 56 IS ACCESS IN LOCAL MEMORY ? NO DO NOT UPDATE LRU TIME STAMP 60 YES 58 UPDATE LRU TIME STAMP 50 IS ACCESS IN LOCAL MEMORY ? YES 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 FIG. 2

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Cox discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, as summarized in Cox: In database management and other applications that allocate a cache in main memory to temporarily store data from external subsystems, the buffer cache memory 14 typically has a limited capacity. In operation, when a process running on a processor node 18 retrieves data into the application cache for the first time and the cache connected to the processor does not have sufficient capacity available to store the new data (step 40), then previously stored data has to be removed from the application cache in order to free-up memory space (step 42). The mechanism discussed in the present application determines what data will be purged (or overwritten) from the local memory based on a least recently used (LRU) memory management mechanism. The LRU memory management mechanism stores the most recently retrieved (or used) data in the local memory. Typically, when data is stored in the local memory the local processor time stamps the data. Thus, the data stored in the memory with the oldest time stamp will typically be overwritten. When space is available in the buffer cache (step 40) the process assigns the next available virtual address, to the data (step 44). *Cox et al.*, U.S. Patent No. 5,918,249 at 3:21-42. If the physical address is local to the node (step 50), the

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		application determines if the local memory has sufficient capacity available to store the data, similar to step 40. If there is not enough memory space available in the buffer cache LRU data is purged, similar to step 42. Once the LRU data is purged or if space is available to store the data, the application process on node A retrieves, time stamps and stores the data in memory (step 52). *Id*. at 3:55-63. The process of the system is best understood with reference to Fig. 2: *Id*. at Figure 2.

EXHIBIT B-8

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		 FOR EACH DATA ACCESS TO EXTERNAL STORAGE DO IS SPACE AVAILABLE IN APPLICATION BUFFER CACHE ? YES GET NEXT AVAILABLE VIRTUAL ADDRESSES 44 40 NO PURGE LEAST RECENTLY USED DATA TO FREE MEMORY FOR NEW DATA 42 46 DETERMINE IF PHYSICAL ADDRESS IS MAPPED TO GIVEN VIRTUAL ADDRESS ? NO 48 ASSIGN PHYSICAL MEMORY ADDRESSES FOR THE VIRTUAL ADDRESSES YES 56 IS ACCESS IN LOCAL MEMORY ? NO 60 DO NOT UPDATE LRU TIME STAMP YES 58 UPDATE LRU TIME STAMP 50 YES IS ACCESS IN LOCAL MEMORY ? 52 UPDATE LRU TIME STAMP NO RE – MAP VIRTUAL ADDRESS TO A NEW PHYSICAL ADDRESS THAT IS LOCAL 54 FIG. 2

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		In summary, if there is space available in the buffer cache, then the system does not purge any data from the cache. If, however, free space is not available in the buffer cache, then the system purges the last item (i.e., the least recently used item) from the buffer cache based on the least recently mechanism used by the system to make room for the new data. Therefore, the determination of when to delete data is dynamically determined by whether free space exists within the buffer cache. Additionally, it would have been obvious to one of ordinary skill in the art to combine the system taught in Cox and Beardsley with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles to disclose dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be

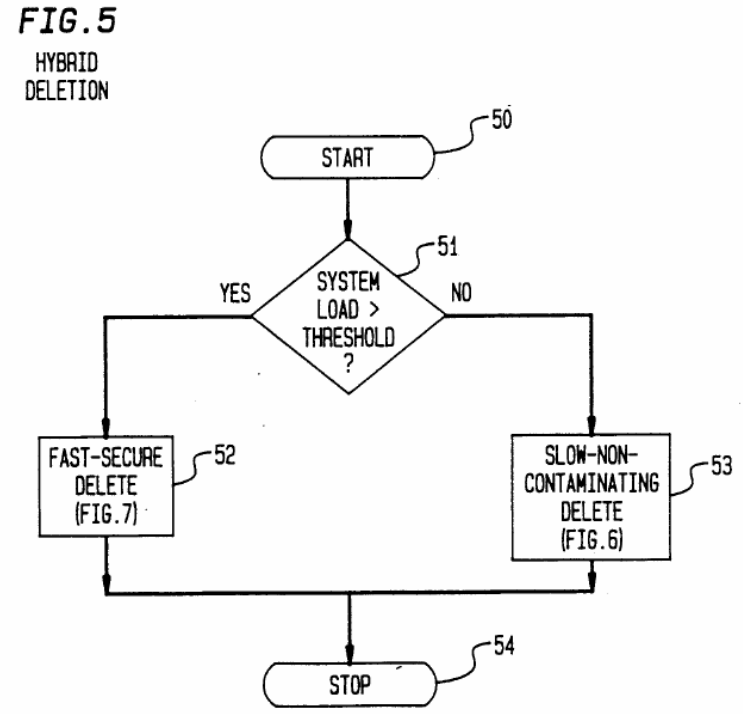
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Cox and Beardsley and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Cox and Beardsley would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Cox and Beardsley and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Cox and Beardsley with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Cox and Beardsley with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Cox and Beardsley with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Cox and Beardsleycan be

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Cox and Beardsley with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Cox and Beardsley with Thatte. Alternatively, it would also be obvious to combine Cox and Beardsley with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Cox and Beardsley and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the ’663 patent’s deletion decision procedure with Cox and Beardsley would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Cox and Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Cox and Beardsley with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Cox and Beardsley and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Cox and Beardsley. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Cox and Beardsley would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Cox and Beardsley and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Cox and Beardsley to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Cox and Beardsley with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Cox and Beardsley can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Cox in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Cox. For example, both Linux 2.0.1 and Cox describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")**
		number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		line 1293. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,918,249 to Cox et al. ("Cox") alone and in combination with U.S. Patent No. 5,991,775 to Beardsley et al. ("Beardsley")
		process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-8

EXHIBIT B-9

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Devarakonda discloses an information storage and retrieval system. For example, Devarakonda describes a "method for providing an encapsulated cluster with affinity-based routing of client requests to nodes in the cluster," a part of which includes extending "the TCP router to maintain an affinity table of recent client TCP connections, after the TCP connections have been closed (by a FIN command)." U.S. Pat. No. '992 col. 4:8-9, 4:58-61. The affinity table stores and retrieves information about where to route packets from particular clients. *Id.*
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Devarakonda discloses a hash table to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. For example, Devarakonda describes an affinity table that "contains information about recent connections. . . . Each row in this table is known as an affinity record 340. The affinity table 300 is searched for an affinity record with the same address as the client address in the newly arrived packet." Devarakonda explains that "[a]lternative implementations [for the affinity table] include, but are not limited to: arrays; balanced trees; and hash tables." Given this suggestion, it would have been obvious to one of ordinary skill in the art that the affinity table could be created using a linked list or a hash table with external chaining. There are a limited number of methods to solve the problem of collisions in a hash table. One of those methods, external chaining, was commonly known by those skilled in the art, as Nemes admits in the '120 patent. See U.S. Pat. No. '120 col. 1:53-59 ("Some form of collision resolution must therefore be provided. For example, the simple strategy called

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		'linear probing,' . . . is often used. Another method for resolving collisions is called 'external chaining.'"). This method of collision resolution is described in the prior art cited by the '120 patent. See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 513, 518, 1973. Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Thus, one of ordinary skill in the art would have been motivated by Knuth and Weiss to use external chaining to solve the problem in collision resolution in the hash table taught by Devarakonda. The records in the system Devarakonda discloses include records, at least some of which automatically expire. For example, the affinity records stored in the affinity table expire when they pass a certain age. "In decision block 1070, the affinity record is tested to determine if it is too old. For example, each affinity record could include a timestamp, which is then compared to the current time. If the difference in those times exceeds a given threshold (also called the affinity period), for example 100 seconds, then execution proceeds to function block 1080, otherwise the affinity record is not too old, so execution proceeds to function block 1120." U.S. Pat. No. '992 col. 7:7-8, 7:25-32.
[1b] a record search means utilizing a search key to	[5b] a record search means utilizing a search key to	Devarakonda discloses a record search means utilizing a search key to access the affinity table.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
access the linked list,	access a linked list of records having the same hash address,	For example, "[i]n function block 1030, a search is made in a table called the affinity table 300 shown in FIG. 3. . . . The affinity table 300 is searched for an affinity record with the same address as the client address in the newly arrived packet." U.S. Pat. No. '922 col. 7:7-19. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Devarakonda discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the affinity table when the affinity table is accessed. For example, during the search of the affinity table, a check is made to determine whether the affinity record found is "too old." If so, "the affinity record for which the affinity period has elapsed is removed from the affinity table 300. Then, in function block 1100, which follows both decision block 1050 and function block 1080, a new affinity record is created." U.S. Pat. No. '992 col. 7:38-42. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time,	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the	Devarakonda discloses means, utilizing the record search means, for accessing the affinity table and, at the same time, removing at least some of the expired ones of the records in the affinity table. Devarakonda also discloses means, utilizing the record search means, for inserting and retrieving records from the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
removing at least some of the expired ones of the records in the linked list.	system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	system and, at the same time, removing at least some expired ones of the records in the accessed affinity table. For example, during the search of the affinity table, a check is made to determine whether the affinity record found is "too old." If so, "the affinity record for which the affinity period has elapsed is removed from the affinity table 300." U.S. Pat. No. '992 col. 7:38-40. After this removal, a new affinity record is created and the information in the record is retrieved to create a connection record. "[I]n function block 1100, which follows both decision block 1050 and function block 1080, a new affinity record is created." U.S. Pat. No. '992 col. 7:40-42. "In function block 1200, a connection record is created. Such records contain sufficient information to identify this connection and a field indicating which server was assigned for this connection." U.S. Pat. No. '992 col. 7:66-8:2. It would have been obvious to one skilled in the art that a deletion could have been done at the same time as the removal of records, since insertion, retrieval, and deletion are all basic functions that can be performed on a hash table. *See, e.g.*, "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
2. The information storage	6. The information storage	It would have been obvious to one of ordinary skill in the art to modify the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda (“Devarakonda”) alone and in combination**
and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	system disclosed in Devarakonda to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. Devarakonda combined with Dirks, Thatte, the ’663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14.

EXHIBIT B-9

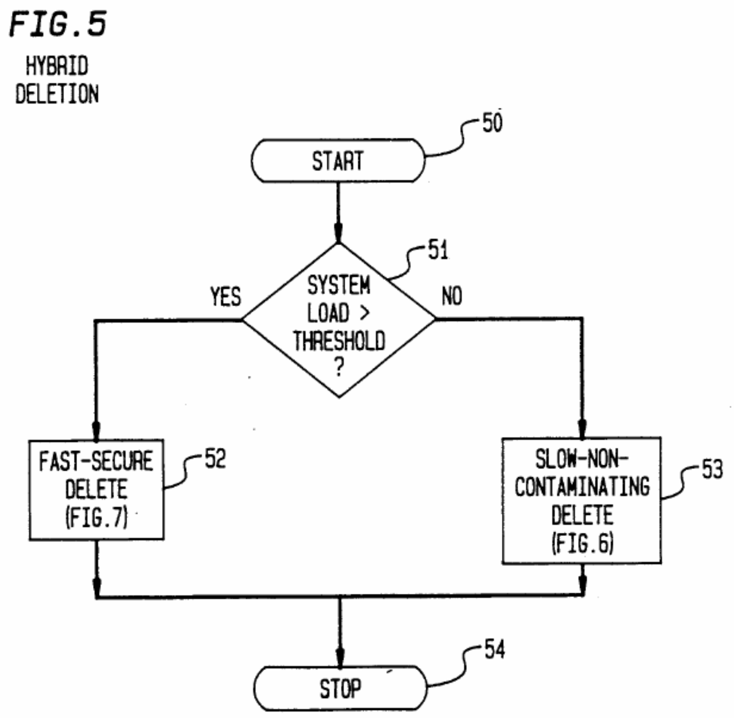
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Devarakonda and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Devarakonda nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Devarakonda and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.`

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Devarakonda with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Devarakonda with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Devarakonda with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Devarakonda can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Devarakonda with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Devarakonda with Thatte. Alternatively, it would also be obvious to combine Devarakonda with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-

EXHIBIT B-9

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5. FIG. 5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		Id. at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Devarakonda and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Devarakonda would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Devarakonda and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Devarakonda with the Opportunistic Garbage Collection Articles. For example, the Opportunistic Garbage Collections Articles disclose in part: The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Devarakonda and the Opportunistic Garbage Collection Articles relate

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Devarakonda would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Devarakonda and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Devarakonda to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		ordinary skill in the art would have been motivated to combine the system disclosed in Devarakonda with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Devarakondacan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Devarakonda in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Devarakonda. For example, both Linux 2.0.1 and Devarakonda describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Devarakonda discloses a method for storing and retrieving information records using an affinity table to store and provide access to the records, at least some of the records automatically expiring. It would have been obvious to one of ordinary skill in the art that a linked list or hashing with external chaining could be used to implement the affinity table. For example, Devarakonda describes a "method for providing an encapsulated cluster with affinity-based routing of client requests to nodes in the cluster," a part of which includes extending "the TCP router to maintain an affinity table of recent client TCP connections, after the TCP connections have been closed (by a FIN command)." U.S. Pat. No. '992 col. 4:8-9, 4:58-61. The affinity table stores and retrieves information about where to route packets from particular clients. The affinity table "contains information about recent connections. . . . Each row in this table is known as an affinity record 340. The affinity table 300 is searched for an affinity record with the same address as the client address in the newly arrived packet." Devarakonda states that "[a]lternative implementations [for the affinity table] include, but are not limited to: arrays; balanced trees; and hash tables." As discussed in [1a/5a], one of ordinary skill

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		in the art would have known that a linked list or a hash table with external chaining using linked lists could be used for the affinity table. The records in the system Devarakonda discloses include records, at least some of which automatically expire. For example, the affinity records stored in the affinity table expire when they pass a certain age. "In decision block 1070, the affinity record is tested to determine if it is too old. For example, each affinity record could include a timestamp, which is then compared to the current time. If the difference in those times exceeds a given threshold (also called the affinity period), for example 100 seconds, then execution proceeds to function block 1080, otherwise the affinity record is not too old, so execution proceeds to function block 1120." U.S. Pat. No. '992 col. 7:7-8, 7:25-32.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Devarakonda discloses accessing the affinity table of records. For example, "[i]n function block 1030, a search is made in a table called the affinity table 300 shown in FIG. 3. . . . The affinity table 300 is searched for an affinity record with the same address as the client address in the newly arrived packet." As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
[3b] identifying at least some of the automatically expired ones of the records,	[7b] identifying at least some of the automatically expired ones of the records,	Devarakonda discloses identifying at least some of the automatically expired ones of the records.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
and		For example, during the search of the affinity table, a check is made to determine whether the affinity record found is "too old." U.S. Pat. No. '992 col. 7:25-32. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Devarakonda discloses removing at least some of the automatically expired records from the affinity table when the affinity table is accessed. For example, during the search of the affinity table, a check is made to determine whether the affinity record found is "too old." If so, "the affinity record for which the affinity period has elapsed is removed from the affinity table 300. Then, in function block 1100, which follows both decision block 1050 and function block 1080, a new affinity record is created." U.S. Pat. No. '992 col. 7:38-42. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Devarakonda discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, during the search of the affinity table, a check is made to determine whether the affinity record found is "too old." If so, "the affinity record for which the affinity period has elapsed is removed from the affinity

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		table 300." U.S. Pat. No. '992 col. 7:38-40. After this removal, a new affinity record is created and the information in the record is retrieved to create a connection record. "[I]n function block 1100, which follows both decision block 1050 and function block 1080, a new affinity record is created." U.S. Pat. No. '992 col. 7:40-42. "In function block 1200, a connection record is created. Such records contain sufficient information to identify this connection and a field indicating which server was assigned for this connection." U.S. Pat. No. '992 col. 7:66-8:2. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list or a hash table using external chaining with linked lists could be used for the affinity table.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	It would have been obvious to one of ordinary skill in the art to modify the systems and methods disclosed in Devarakonda to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. Devarakonda combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks,

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

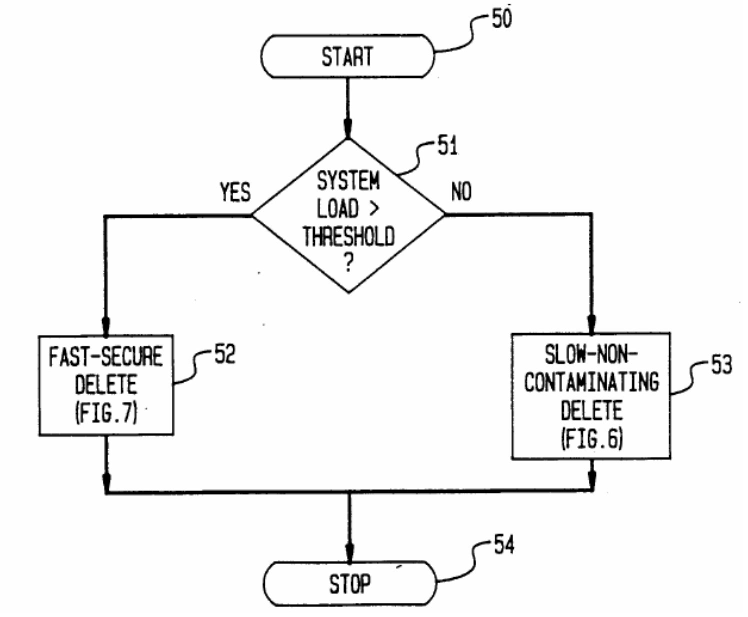
EXHIBIT B-9

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Devarakonda and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Devarakonda would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Devarakonda and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Devarakonda with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Devarakonda with Thatte would be nothing more than the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Devarakonda with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Devarakondacan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Devarakonda with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Devarakonda with Thatte. Alternatively, it would also be obvious to combine Devarakonda with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Devarakonda and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Devarakonda would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda (“Devarakonda”) alone and in combination**
		By way of further example, one of ordinary skill in the art would have combined the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load as taught by the ’663 patent and with Devarakonda and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Devarakonda with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Devarakonda and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Devarakonda. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Devarakonda would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Devarakonda and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Devarakonda to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Devarakonda with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Devarakonda can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination**
		time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Devarakonda in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Devarakonda. For example, both Linux 2.0.1 and Devarakonda describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda ("Devarakonda") alone and in combination
		process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-9

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,424,992 Devarakonda (“Devarakonda”) alone and in combination

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Kerr discloses an information storage and retrieval system. For example, Kerr describes a "method and system for switching in networks responsive to message flow patterns." Kerr at col. 1:48-49. Part of that system includes a flow cache "in which routing information to be used for packets 150 in each particular message flow 160 is recorded and from which such routing information is retrieved for use." Kerr at col. 3:42-45.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Kerr discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Kerr also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Kerr describes a flow cache that "comprises a memory which associates flow keys 210 with information about message flows 160 identified by those flow keys 310. The flow cache 300 includes a set of buckets 301. Each bucket 301 includes a linked list of entries 302. Each entry 302 includes information about a particular message flow 160 … and a pointer to information about the treatment of packets 150 to the destination device 130 for that message flow 160." Kerr at col. 6:32-41. The records in the system Kerr discloses includes records, at least some of which automatically expire. For example, Kerr explains that "[a]t step 241, the routing device 140

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		examines each entry in the flow cache and compares a current time with a last time a packet 150 was routed using that particular entry. If the difference exceeds a first selected timeout, the message flow 160 represented by that entry is considered to have expired due to nonuse and thus to no longer be valid." Kerr at col. 5:52-57. Kerr further explains that "[i]n a preferred embodiment, the routing device 140 also examines the entry in the flow cache and compares a current time with a first time a packet 150 was routed using that particular entry. If the difference exceeds a second selected timeout, the message flow 160 represented by that entry is considered to have expired due to age and thus to no longer be valid. The second selected timeout is preferably about one minute." Kerr at col. 5:58-65. Kerr also states that "[i]n a preferred embodiment, the routing device 140 also examines the entry in the flow cache and determines if the "next hop" information has changed. If so, the message flow 160 is expired due to changed conditions. Other changed conditions which might cause a message flow 160 to be expired include changes in access control lists or other changes which might affect the proper treatment of packets 150 in the message flow 160. The routing device 140 also expires entries in the flow cache on a least-recently-used basis if the flow cache becomes too full." Kerr at col. 6:10-19.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Kerr discloses a record search means utilizing a search key to access the linked list. Kerr also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, the flow cache "associates flow keys 310 with information about

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		message flows 160 identified by those flow keys 310." Kerr at col. 6:32-34. Those flow keys are used to access records stored in the linked lists in the flow cache hash table. Kerr at col. 6:32-35. *See also* Kerr at 3:57-4:12.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Kerr discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. For example, the routing device described in Kerr identifies expired entries for message flows and "[i]f the message flow is no longer valid, the routing device continues with the step 242. At step 242, the routing device 140 collects historical information about the message flow 160 from the entry in the flow cache, and deletes the entry." Kerr at col. 6:22-27. *See also* Kerr at 3:57:4:12, 6:11-19.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Kerr discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Kerr also discloses means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, the routing device described in Kerr will build a new entry in the flow cache if one does not already exist for the packet's message flow. See Kerr at col. 4:12-13. The device then identifies expired entries for message flows and "[i]f the message flow is no longer valid, the routing device continues with the step 242. At step 242, the routing device 140 collects historical information about the message flow 160 from the entry in the flow

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		cache, and deletes the entry." Kerr at col. 6:22-27. The routing device in Kerr also performs a retrieval and deletion. For example, "[a]t a step 225, the routing device 140 retrieves routing information from the entry in the flow cache for the identified message flow 160." Kerr at col. 4:52-55; *see also* Kerr at col. 6:25-27. To the extent Kerr does not include means for inserting, retrieving, and deleting records utilizing the record search means, it would have been obvious to one skilled in the art that insertion, retrieval or deletion could have been done, since these are all basic functions that can be performed on a hash table or a linked list in similar ways when the hash table or linked list is accessed. *See*, *e.g.*, "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Kerr combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety.

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr (“Kerr”) alone and in combination**
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Kerr and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Kerr nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Kerr and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Kerr with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Kerr with Thatte would be nothing more than the predictable use of prior art elements according to their established functions.

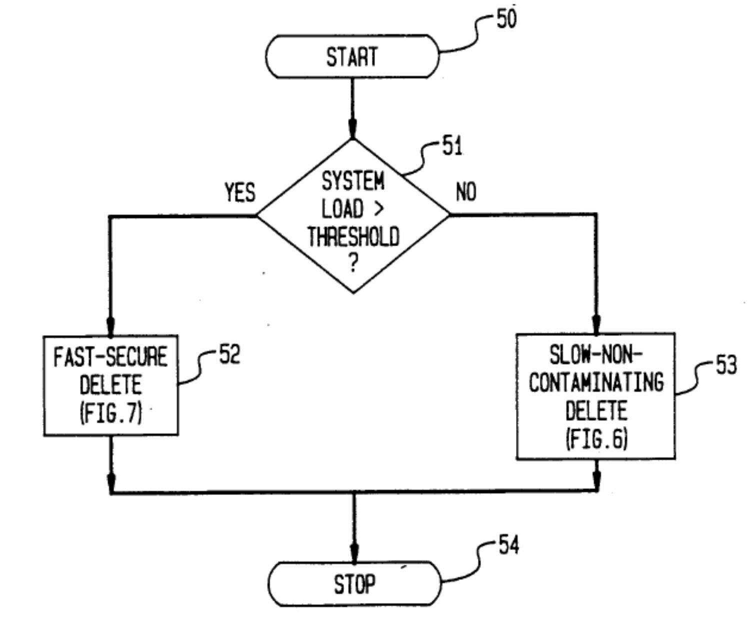
EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Kerr with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Kerr can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Kerr with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Kerr with Thatte. Alternatively, it would also be obvious to combine Kerrwith the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Kerr and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Kerr would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		based on a systems load as taught by the '663 patent and with Kerr and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Kerr with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Kerr and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Kerr would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Kerr and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Kerr to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Kerr with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Kerrcan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Further, Kerr states that "[o]ne problem which has arisen in the art is that processing demands on routing and switching devices continue to grow with increased network demand," thus "[i]t continues to be advantageous to provide techniques for processing packets more quickly." Kerr at 1:22-38. Given this focus on efficiency in Kerr, one of ordinary skill in the art would have been motivated to try the teachings of Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles in combination with Kerr to optimize the performance of the system and method disclosed in Kerr. To the extent that dynamically determining a maximum number of expired records is not disclosed by Kerr in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Kerr. For example, both Linux 2.0.1 and Kerr describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero.

EXHIBIT B-10

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Kerr discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Kerr also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Kerr describes a "method and system for switching in networks responsive to message flow patterns." Kerr at col. 1:48-49. Part of that system includes a flow cache, which is used to store and retrieve information about message flows. "At a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160." U.S. Pat. No. '667 col. 4:1-2. The flow cache "comprises a memory which associates flow keys 210 with information about message flows 160 identified by those flow keys 310. The flow cache 300 includes a set of buckets 301. Each bucket 301 includes a linked list of entries 302. Each entry 302 includes information about a particular message flow 160 … and a pointer to information about the treatment of packets 150 to the destination device 130 for that message flow 160." Kerr at col. 6:32-41. The records in the system Kerr discloses includes records, at least some of which automatically expire.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		For example, Kerr explains that "[a]t step 241, the routing device 140 examines each entry in the flow cache and compares a current time with a last time a packet 150 was routed using that particular entry. If the difference exceeds a first selected timeout, the message flow 160 represented by that entry is considered to have expired due to nonuse and thus to no longer be valid." Kerr at col. 5:52-57. Kerr further explains that "[i]n a preferred embodiment, the routing device 140 also examines the entry in the flow cache and compares a current time with a first time a packet 150 was routed using that particular entry. If the difference exceeds a second selected timeout, the message flow 160 represented by that entry is considered to have expired due to age and thus to no longer be valid. The second selected timeout is preferably about one minute." Kerr at col. 5:58-65. Kerr also states that "[i]n a preferred embodiment, the routing device 140 also examines the entry in the flow cache and determines if the "next hop" information has changed. If so, the message flow 160 is expired due to changed conditions. Other changed conditions which might cause a message flow 160 to be expired include changes in access control lists or other changes which might affect the proper treatment of packets 150 in the message flow 160. The routing device 140 also expires entries in the flow cache on a least-recently-used basis if the flow cache becomes too full. Kerr at col. 6:10-19.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Kerr discloses accessing the linked list of records. Kerr also discloses accessing a linked list of records having same hash address

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		For example, "[a]t a step 223, the routing device 140 performs a lookup in a flow cache for the identified message flow 160." Kerr at col. 4:1-2. "At a step 225, the routing device 140 retrieves routing information from the entry in the flow cache for the identified message flow 160." Kerr at col. 4:53-55. "At a step 241, the routing device 140 examines each entry in the flow cache and compares a current time with a last time a packet 150 was routed using that particular entry." Kerr at col. 5:52-54. *See also* Kerr at 6:32-41.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Kerr discloses identifying at least some of the automatically expired ones of the records. For example, the routing device described in Kerr identifies expired entries for message flows and "[i]f the message flow is no longer valid, the routing device continues with the step 242. At step 242, the routing device 140 collects historical information about the message flow 160 from the entry in the flow cache, and deletes the entry." Kerr at col. 6:22-27. *See also* Kerr at 5:52-57.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Kerr discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, the routing device described in Kerr identifies expired entries for message flows and "[i]f the message flow is no longer valid, the routing device continues with the step 242. At step 242, the routing device 140 collects historical information about the message flow 160 from the entry in the flow cache, and deletes the entry." Kerr at col. 6:22-27. *See also* Kerr at 6:11-19.
	[7d] inserting, retrieving or deleting one of the	Kerr discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr (“Kerr”) alone and in combination**
	records from the system following the step of removing.	For example, *see* Kerr at 5:66-6:10. “Expiring message flows 160 due to age artificially requires that a new message flow 160 must be created for the next packet 150 in the same communication session represented by the old message flow 160 which was expired. *Id.* at 5:66-6:2.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Kerr combined with Dirks, Thatte, the ’663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list

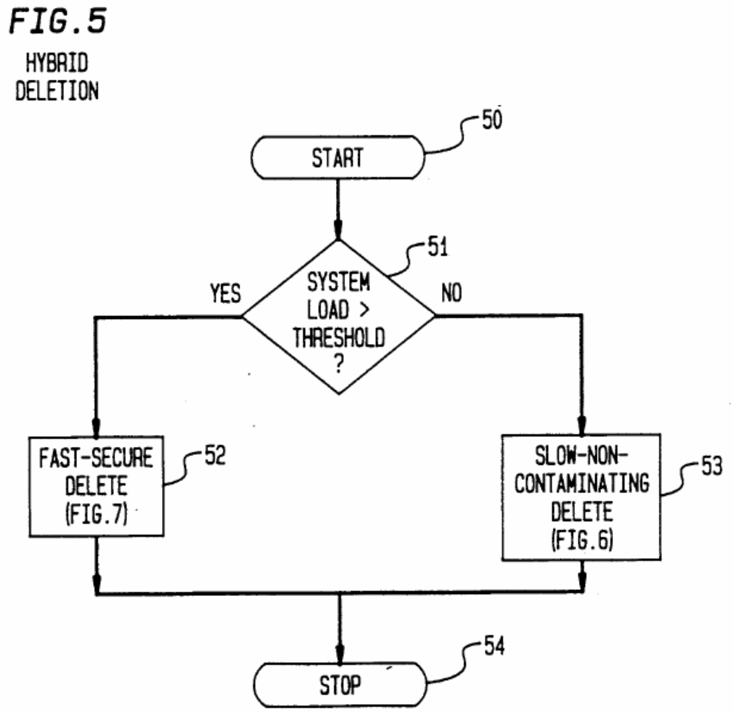
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Kerr and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Kerr would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Kerr and would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Kerr with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Kerr with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Kerr with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Kerrcan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Kerr with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Kerr with Thatte. Alternatively, it would also be obvious to combine Kerr with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Kerr and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Kerr would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination**
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Kerr and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Kerr with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Kerr and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Kerr. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Kerr would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Kerr and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Kerr to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Kerr with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Kerr can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Further, Kerr states that "[o]ne problem which has arisen in the art is that processing demands on routing and switching devices continue to grow with increased network demand," thus "[i]t continues to be advantageous to provide techniques for processing packets more quickly." Kerr at 1:22-38. Given this focus on efficiency in Kerr, one of ordinary skill in the art would have been motivated to try the teachings of Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles in combination with Kerr to optimize the performance of the system and method disclosed in Kerr. To the extent that dynamically determining a maximum number of expired records is not disclosed by Kerr in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Kerr. For example, both Linux 2.0.1 and Kerr describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		When invoked, the function rt_cache_add automatically increments an integer variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1359. When the function rt_cache_add removes an expired record, the function rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 6,243,667 Kerr ("Kerr") alone and in combination
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Corbin discloses an information storage and retrieval system. For example, Corbin describes a "method and an apparatus for a new communication framework," a part of which includes a "data route table contain[ing] a list of predetermined pattern of bits which represent a set of pre-determined or registered routes." U.S. Pat. No. '241 col. 2:21-2, 2:55-57.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Corbin inherently discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. It would have been obvious to one of ordinary skill in the art to use a hash table with external chaining with the teachings disclosed by Corbin. For example, though Corbin describes a data route table implemented as a Patricia Tree, it also notes that "[t]he present invention's architecture of data routing is not dependent on Patricia Trees and hence other search algorithms may be used instead." U.S. Pat. No. '241 col. 10:54-56. A linked list is simply a special case of a "tree" data structure wherein each node contains a single "branch," thus Corbin's disclosure of the use of Patricia Trees inherently discloses the use of linked lists as well. To the extent such a disclosure is not inherent in Corbin, it would have been obvious to one of ordinary skill in the art that a linked list or a hash table with external chaining could be used in a search algorithm instead of a Patricia Tree given Corbin's suggestion that other search algorithms may be used instead. Hashing with external chaining using linked lists was a well known search algorithm familiar to those of ordinary skill in the art with its own particular advantages. *See, e.g.*, "The Art of Computer Programming", Sorting and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549. The records in the system Corbin discloses include records, at least some of which automatically expire. For example, "[a] lazy deletion algorithm may be used in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree are expired.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Corbin discloses a record search means utilizing a search key to access the Patricia Tree. For example, "FIGS. 6b-6f illustrate an exemplary Patricia Tree search implementation of a data route table search as utilized by the present invention. Data route entries in the Patricia Tree, also referred to herein as data route keys, are variable with sequences of bytes with some keys as long as 80 bytes." U.S. Pat. No. '241 col. 9:43-447. As discussed in [1a/5a], Corbin inherently discloses use of a linked list. Also, as discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a hash table using external chaining with linked lists for the search algorithm could be used instead of the Patricia Tree.
[1c] the record search means including a means for identifying and removing at least some of	[5c] the record search means including means for identifying and removing at least some expired ones	Corbin discloses the use of a lazy deletion algorithm on the Patricia Tree. It would have been obvious to one of ordinary skill in the art that a lazy deletion algorithm could be used on a linked list as well, which would result in the removal of expired elements when the linked list is accessed. As noted above,

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
the expired ones of the records from the linked list when the linked list is accessed, and	of the records from the linked list of records when the linked list is accessed, and	a linked list is simply a special case of a "tree" data structure wherein each node contains a single "branch." For example, Corbin explains that "[a] lazy deletion algorithm may be used [on the data route table] in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree are expired. When the "number of deleted keys become significant, the tree is rebuilt." U.S. Pat. No. '241 col. 10:47-49. The rebuild of the tree identifies and removes the expired records from the Patricia Tree. As discussed in [1a/5a], Corbin inherently discloses use of a linked list. Also, as discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a hash table using external chaining with linked lists could be used for the search algorithm instead of the Patricia Tree. One of ordinary skill in the art would also know that lazy deletion could be used in such structures as well. *See* Exhibit C-2, which is incorporated by reference. When performing lazy deletion on a linked list, the expired elements would be removed on subsequent accesses to the linked list.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Corbin discloses means, utilizing the record search means, for accessing the Patricia Tree and, at the same time, removing at least some of the expired ones of the records in the Patricia Tree. For example, Corbin explains that "[a] lazy deletion algorithm may be used [on the data route table] in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree would be expired. When the "number of deleted keys become significant, the tree is rebuilt." U.S. Pat. No. '241 col. 10:47-49. The rebuild of the tree

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		identifies and removes the expired records from the Patricia Tree. As discussed in [1a/5a], Corbin inherently discloses use of a linked list. Also, as discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a hash table using external chaining with linked lists could be used for the search algorithm instead of the Patricia Tree. One of ordinary skill in the art would also know that lazy deletion could be used in such structures as well. *See* Exhibit C-2, which is incorporated by reference. When performing lazy deletion on a linked list, the expired elements would be removed on subsequent accesses to the linked list. Corbin also discloses means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed Patricia Tree of records. Corbin explains that "if a flag is set in the tree head indicating that a rebuild is taking place, then the insertion and deletion operations should go to sleep waiting for the rebuild to be complete." Moreover, as discussed above, Corbin describes the use of a lazy deletion algorithm, and one of ordinary skill in the art would know that implementing a lazy deletion algorithm in a linked list would involve the removal of expired records from the list when the list is accessed for insertions, retrievals, and deletions. *See* Exhibit C-2, which is incorporated by reference.
2. The information storage and retrieval system according to claim 1	6. The information storage and retrieval system according to claim 5	Corbin combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		As both Corbin and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Corbin nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Corbin and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Corbin with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Corbin with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Corbin with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Corbin can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Corbin with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Corbin with Thatte. Alternatively, it would also be obvious to combine Corbin with the '663

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-11

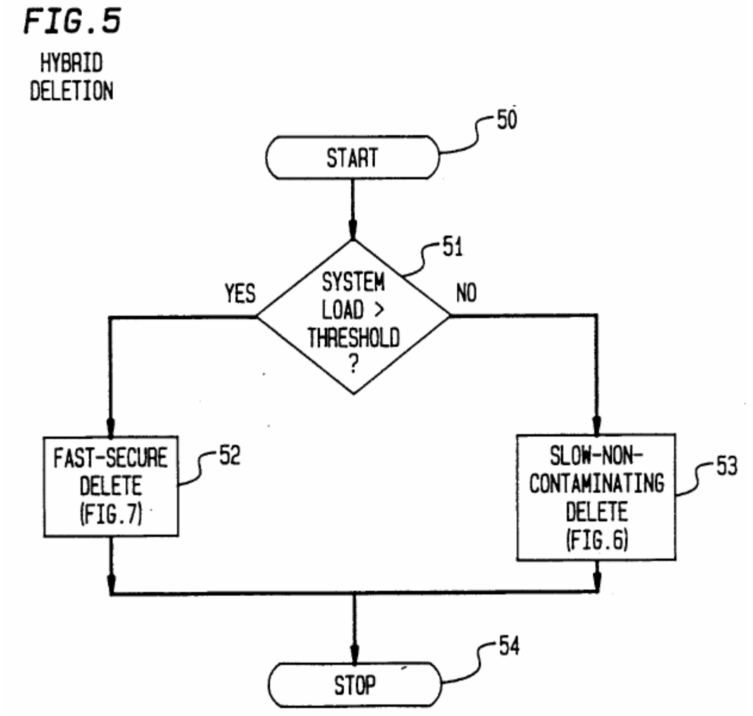
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Corbin and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Corbin would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Corbin and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Corbin with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Corbin and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Corbin would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Corbin and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Corbin to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Corbin with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Corbin can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Further, Corbin describes the importance of performance in its packet routing system, and hence one of ordinary skill in the art would be looking for ways to optimize this performance. *See, e.g.,* Corbin col. 1:49-2:18, 10:42-56. To the extent that dynamically determining a maximum number of expired records is not disclosed by Corbin in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Corbin. For example, both Linux 2.0.1 and Corbin describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		`rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		2.0.1, route.c at lines 1341-1342. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function rt_garbage_collect_1 can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Corbin inherently discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Moreover, it would have been obvious to one of ordinary skill in the art that hash tables with external chaining using linked lists could be used instead of the Patricia Tree disclosed in Corbin. For example, Corbin describes a "method and an apparatus for a new communication framework," a part of which includes a "data route table contain[ing] a list of predetermined pattern of bits which represent a set of pre-determined or registered routes." U.S. Pat. No. '241 col. 2:21-2, 2:55-57. Though Corbin describes the data route table implemented as a Patricia Tree, it also notes that "[t]he present invention's architecture of data routing is not dependent on Patricia Trees and hence other search algorithms may be used instead." U.S. Pat. No. '241 col. 10:54-56. A linked list is simply a special case of a "tree" data structure wherein each node contains a single "branch," thus Corbin's disclosure of the use of Patricia Trees inherently discloses the use of linked lists as well. To the extent such a disclosure is not inherent in Corbin, it would have been obvious to one of ordinary skill in the art that a linked list or a hash table with external chaining could be used in a search algorithm instead of a Patricia Tree given Corbin's suggestion that other search algorithms may be used instead. Hashing with external chaining using linked lists was a well known search algorithm familiar to those of ordinary skill in the art with its own particular

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		advantages. *See, e.g.*, "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549. The records in the system Corbin discloses include records, at least some of which automatically expire. For example, "[a] lazy deletion algorithm may be used in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree are expired. *See id.* Corbin further explains that "[i]n a preferred embodiment, the routing device 140 also examines the entry in the flow cache and compares a current time with a first time a packet 150 was routed using that particular entry. If the difference exceeds a second selected timeout, the message flow 160 represented by that entry is considered to have expired due to age and thus to no longer be valid. The second selected timeout is preferably about one minute." U.S. Pat. No. '667 Col. 5:58-65.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Corbin inherently discloses accessing the linked list of records. Additonally, it would have been obvious to one of ordinary skill in the art that a linked list could be accessed instead of the Patricia Tree disclosed by Corbin. See above. For example, "FIGS. 6b-6f illustrate an exemplary Patricia Tree search implementation of a data route table search as utilized by the present invention. Data route entries in the Patricia Tree, also referred to herein as data route keys, are variable with sequences of bytes with some keys as long as 80 bytes." U.S. Pat. No. '241 col. 9:43-447.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		As discussed in [3/7], the disclosure of the Patricia Tree inherently discloses the use of linked lists. Moreover, as discussed in [3/7], it would have been obvious to one of ordinary skill in the art that a hash table using external chaining with linked lists for the search algorithm instead of the Patricia Tree. *See id.*
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Corbin discloses identifying at least some of the automatically expired ones of the records. For example, Corbin explains that "[a] lazy deletion algorithm may be used [on the data route table] in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree are expired. When the "number of deleted keys become significant, the tree is rebuilt." U.S. Pat. No. '241 col. 10:47-49. The rebuild of the tree identifies and removes the expired records from the Patricia Tree. *See id.*
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Corbin inherently discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Moreover, removing automatically expired records from a linked list would have been obvious to one of ordinary skill in the art given the teachings in Corbin. For example, Corbin explains that "[a] lazy deletion algorithm may be used [on the data route table] in which a key is marked as deleted but left in the tree." U.S. Pat. No. '241 col. 10:46-47. Those items marked as deleted but left in the tree are expired. When the "number of deleted keys become significant, the tree is rebuilt." U.S. Pat. No. '241 col. 10:47-49. The rebuild of the tree identifies and removes the expired records from the Patricia Tree. *See id.*

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		As discussed in [3/7], the disclosure of the Patricia Tree inherently discloses the use of linked lists. Moreover, as discussed in [3/7], it would have been obvious to one of ordinary skill in the art that a hash table using external chaining with linked lists for the search algorithm instead of the Patricia Tree. *See id.* One of ordinary skill in the art would also know that lazy deletion could be used in such structures as well. *See* Exhibit C-2 which is incorporated by reference. When performing lazy deletion on a linked list, the expired elements would be removed on subsequent accesses to the linked list.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Corbin discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Corbin explains that "if a flag is set in the tree head indicating that a rebuild is taking place, then the insertion and deletion operations should go to sleep waiting for the rebuild to be complete." Because the insert and deletion processes sleep while the rebuild is occurring, the insert and delete operations necessarily take place following the removal of the expired records.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Corbin combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Corbin and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and

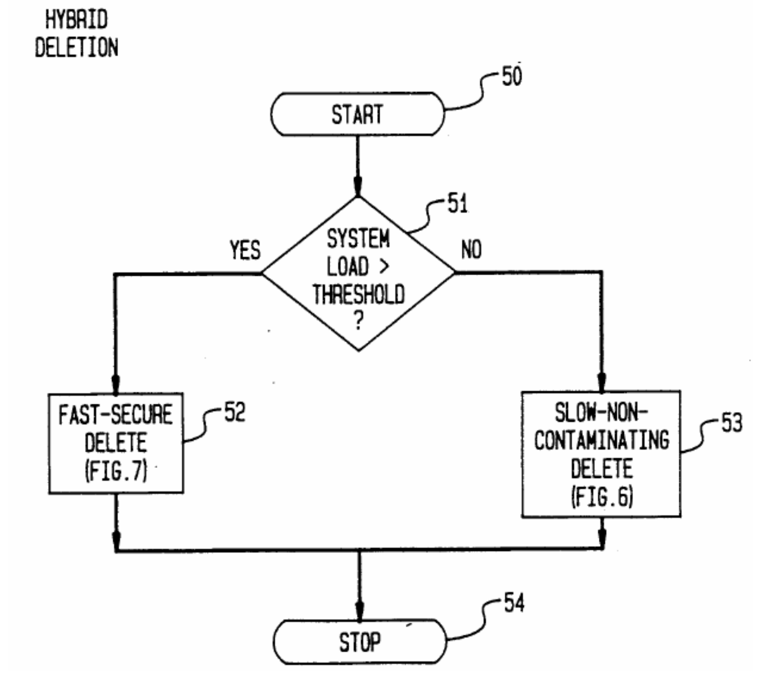
EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Corbin would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Corbin and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Corbin with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		way. Additionally, one of ordinary skill in the art would recognize that the result of combining Corbin with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Corbin with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Corbincan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Corbin with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Corbin with Thatte. Alternatively, it would also be obvious to combine Corbin with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Corbin and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Corbin would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Corbin and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Corbin with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination**
		responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Corbin and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Corbin. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Corbin would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Corbin and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Corbin to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Corbin with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Corbin can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Further, Corbin describes the importance of performance in its packet routing system, and hence one of ordinary skill in the art would be looking for ways to optimize this performance. *See, e.g.,* Corbin col. 1:49-2:18, 10:42-56. To the extent that dynamically determining a maximum number of expired records is not disclosed by Corbin in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Corbin. For example, both Linux 2.0.1 and Corbin describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold RT_CACHE_SIZE_MAX. If the number of records in the hash table exceeds the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_cache_add invokes a function rt_garbage_collect. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function rt_garbage_collect_1 can remove from a linked list is different than

EXHIBIT B-11

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,881,241 Corbin ("Corbin") alone and in combination
		the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, the '495 Patent discloses an information storage and retrieval system. For example, the '495 Patent claims "[a]n information storage and retrieval system using hashing techniques to provide rapid access to the records of said system . . . ." U.S. Pat. No. '495 col.11:68-12:2. Moreover, the '495 Patent states that "[t]his invention relates to information storage and retrieval systems and, more particularly, to the use of hashing techniques in such systems." U.S. Pat. No. '495 col. 1:10-12; see also, Figs. 1-7, Appendix, and Columns 5-8. In addition, to the extent the preamble is a limitation, the '499 Patent discloses an information storage and retrieval system. For example, the Abstract of the '499 patent discloses "An apparatus for performing storage and retrieval in an information storage system is disclosed which uses the hashing technique." '499 patent at Abstract. This storage and retrieval system is further described in columns 4-5, which recite in part, "The present invention is concerned with information storage and retrieval systems. Such a system would form one of the application software packages 23, 24, ... ,25 of FIG. 2. The various processes (26,27,28) which implement the information storage and retrieval system are herein disclosed as flow charts in FIGS. 3, 4 and 5, and shown as pseudocode in the APPENDIX to this specification." Further detail is given in column 7, which recites in part "Referring then to FIG. 3, there is shown a flowchart of a retrieve algorithm for retrieving records from a data storage and retrieval system in accordance with the present invention and involving dual collision resolution schemes dynamically selected depending on load factor." *See also*, Figs. 1-5, Appendix, and Col 8-9.

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	The '495 Patent discloses a chain of records to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. The '495 Patent also discloses a hashing means to provide access to records stored in a memory of the system and using a linear probing technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '495 Patent claims "[a]n information storage and retrieval system using hashing techniques to provide rapid access to the records of said system and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring." U.S. Pat. No. '495 col.11:68-12:4. This claim is identical to [5a] except for the substitution of a linear probing technique instead of an external chaining technique. *See also* '495 Patent col. 4:33-64; *id.* at 8:65-9:9 (code defining hash table), Abstract, Fig. 3, Col. 2:30-48, 5:36-57, 6:53-8:16, claim 1, claim 5. It would have been obvious to one of ordinary skill in the art that an external chaining technique could be used instead of a linear probing technique to store records with the same hash address. Linear probing and external chaining are two methods to solve the problem of collisions in a hash table. Both were commonly known by those skilled in the art, as Nemes admits in the '120 patent. See U.S. Pat. No. '120 col. 1:53-59 ("Some form of collision resolution must therefore be provided. For example, the simple strategy called 'linear probing,' . . . is often used. Another method for resolving collisions is called 'external chaining.'"). Both of these methods of collision resolution are described in the prior art cited by both the '495 and '120 patents. See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		Wesley Series in Computer Science and Information Processing, pp. 513, 518, 1973. Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Further, U.S. Patent No. 5,289,499 ("'499 patent), discloses the use of external chaining techniques for performing storage and retrieval in an information system. *See '499* patent abstract ("In order to provide efficient and graceful operation under varying load conditions, the system shifts between collision avoidance by linear probing with open addressing when the load is below a threshold, and collision avoidance by external chaining when the load is above a threshold"); '499 patent, Figs. 3-5 (use of external chaining – linked list to perform information storage and retrieval); '499 patent, col. 2:60-65, 3:5-9, col. 5:63-65; 8:53-9:20, claims 1-3 (external chaining a required element). Thus, one of ordinary skill in the art would have been motivated by Knuth , Weiss, or the '499 patent to apply the teachings of the '495 patent to hash tables with external chaining using linked lists. Where an external chaining collision resolution strategy is used instead of linear probing, the records would be stored in linked lists rather than in chains of records. Thus, it would also have been obvious to one of ordinary skill in the art that the records could be stored in a linked list. The records in the system the '495 Patent discloses include records, at least some of which automatically expire. The '495 Patent discloses that "data records become obsolete merely by the passage of time or by the occurrence of some event" and that "[i]f such expired, lapsed or obsolete records are not

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		removed from the storage table, they will, in time, seriously degrade or contaminate the performance of the retrieval system." U.S. Pat. No. '495 col. 4:23-28. *See also* '495 Patent at Abstract, Fig. 3, Col. 2:30-48, 5:22-57, 6:53-8:16, claim 1, claim 5.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	The '495 Patent discloses a record search means utilizing a search key to access a chain of records. The '495 Patent also discloses a record search means utilizing a search key to access a chain of records having the same hash address. For example, Claim 1 of the '495 patent is almost identical to [5b], claiming "a record search means utilizing a search key to access a chain of records having the same hash address." U.S. Pat. No. '495 col. 12:6-7. Again, the only difference between the two patents is the reference to a chain of records instead of a linked list. *See also* '495 Pat. col. 5:45-56; *id.* at 10:2511:16 (search table function code); *id.* at Figs. 3-4. It would have been obvious to one of ordinary skill in the art that a linked list could be used to store records with the same hash value rather than a chain of records. Linear probing and external chaining are two methods to solve the problem of collisions in a hash table. Both were commonly known by those skilled in the art, as Nemes admits in the '120 patent. See U.S. Pat. No. '120 col. 1:53-59 ("Some form of collision resolution must therefore be provided. For example, the simple strategy called 'linear probing,' . . . is often used. Another method for resolving collisions is called 'external chaining.'"). Both of these methods of collision resolution are described in the prior art cited by both the '495 and '120 patents. See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		Science and Information Processing, pp. 513, 518, 1973. Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Further, U.S. Patent No. 5,289,499 ("'499 patent), discloses the use of external chaining techniques for performing storage and retrieval in an information system. *See '499* patent abstract ("In order to provide efficient and graceful operation under varying load conditions, the system shifts between collision avoidance by linear probing with open addressing when the load is below a threshold, and collision avoidance by external chaining when the load is above a threshold"); '499 patent, Figs. 3-5 (use of external chaining – linked list to perform information storage and retrieval); '499 patent, col. 2:60-65, 3:5-9, col. 5:63-65; 8:53-9:20, claims 1-3 (external chaining a required element). Thus, one of ordinary skill in the art would have been motivated by Knuth, Weiss, or the '499 patent to apply the teachings of the '495 patent to hash tables with external chaining using linked lists. *See also*, the '499 patent at Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1. Where an external chaining collision resolution strategy is used instead of linear probing, the records would be stored in linked lists rather than in chains of records. Thus, it would also have been obvious to one of ordinary skill in the art that the records could be stored in a linked list.
[1c] the record search means including a means	[5c] the record search means including means for	The '495 Patent discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	the chain of records when the chain of records is accessed. For example, Claim 1 of the '495 patent is almost identical to [5c], claiming "said record search means including means for identifying and removing all expired ones of said records from said chain of records each time said chain is accessed." U.S. Pat. No. '495 col. 12:8-11. Again, the only differences between the two patents are (1) the reference to a chain of records instead of a linked list and (2) the deletion of all of the records in the '495 patent instead of some of the records. *See also* the '495 Patent col. 5:36-57, *id.* at 10:25-11:16 (search table function code); *id.* at Figs. 3-4, Abstract, Col. 2:30-48, 6:53-8:16, claim 1, claim 5. Though this describes accessing a chain of records rather than a linked list, as discussed above in sections [1a/5a] and [1b/5b], it would have been obvious to one of ordinary skill in the art that a linked list could be used to store the records instead of a chain of records. For example, the use of a linked list to store records is discussed in detail in the '499 patent. For example, the '499 patent discloses in part, "Another technique for resolving collisions is called external chaining. In this technique, each hash table position is able to store all records hashing to that location. More particularly, a linked list is used to store the actual records outside of the hash table. The hash table entry, then, is no more than a pointer to the head of the linked list. The linked list is itself searched sequentially when retrieving or storing a record. Deletion is accomplished by adjusting pointers to eliminate the deleted record from the linked list." *See* the '499 patent at column 2. *See, also,* the '499 patent at Abstract, Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1 Moreover, the deletion of all of the expired records includes the deletion of at

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		least some of the expired records and thus satisfies this element of the '120 patent.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	The '495 Patent discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The '495 Patent also discloses means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Claim 1 of the '495 patent is almost identical to [5d], claiming "means, utilizing said record search means, for inserting retrieving and deleting records from said system and, at the same time, removing all expired ones of said records in the accessed chains of records." '495 Patent col. 12:12-16. Again, the only differences between the two patents are (1) the reference to a chain of records instead of a linked list and (2) the deletion of all of the records in the '495 patent instead of some of the records. *See also* '495 Patent col. 5:65-8:16; *id.* at 9:10-10:20 (insert, retrieve, and delete function code); *id.* at Figs. 5-7. Though this describes accessing a chain of records rather than a linked list, as discussed above in sections [1a/5a] and [1b/5b], it would have been obvious to one of ordinary skill in the art that a linked list could be used to store the records instead of a chain of records. For example, the use of a linked list to store records is discussed in detail in the '499 patent. For example, the '499 patent discloses in part, "Another technique for resolving collisions is called external chaining. In this technique, each hash table position is able to store

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		all records hashing to that location. More particularly, a linked list is used to store the actual records outside of the hash table. The hash table entry, then, is no more than a pointer to the head of the linked list. The linked list is itself searched sequentially when retrieving or storing a record. Deletion is accomplished by adjusting pointers to eliminate the deleted record from the linked list." *See* the '499 patent at column 2. *See, also*, the '499 patent at Abstract, Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1. Moreover, the deletion of all of the expired records includes the deletion of at least some of the expired records and thus satisfies this element of the '120 patent.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	The '495 Patent combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

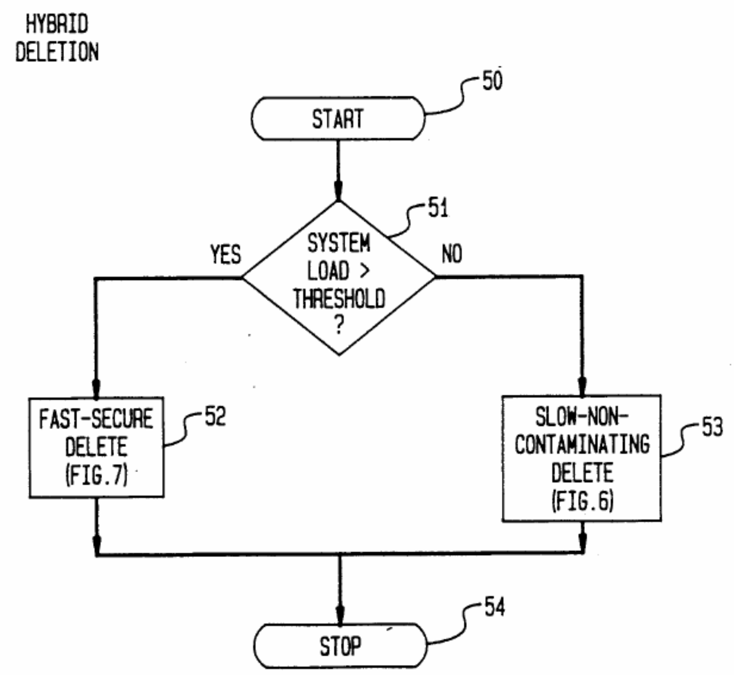
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		$$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both the '495 Patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with the '495 Patent nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with the '495 Patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in the '495 Patent with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining the '495 Patent with Thatte would be nothing more than

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine the '495 Patent with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in the '495 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining the '495 Patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine the '495 Patent with Thatte. Alternatively, it would also be obvious to combine the '495 Patent with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both the '495 Patent and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with the '495 Patent would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with the '495 Patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine the '495 Patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: > When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. > Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both the '495 Patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with the '495 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with the '495 Patent and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '495 Patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in the '495 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in the '495 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. The '499 patent discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, the '499 patent describes performing a dynamic determination of the number of records hashed to the same cell in the hash table. Then determining if this number exceeds a certain threshold. If the threshold is exceeded, then the system described by the '499 patent reorganizes and removes the records at this location. See the '499 Patent at Col. 6, which recites in part "In accordance with the present invention, the major advantages of both techniques, open addressing and external chaining, are achieved in the same system. More particularly, these two techniques are combined in one system, and the actual storage strategy is selected dynamically, depending on the then current local load factor. Initially, all records are stored in the hash table using the open addressing with linear probing technique. When the local load factor exceeds a preselected threshold, the system shifts dynamically to the external chaining technique. That is, while inserting or deleting a record,

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		the local load factor, as reflected in the number of records hashed to this same hash table cell, is examined. If this number exceeds a threshold, the records hashed to this cell address are reorganized, removed from the hash table itself and organized into an external chain in another part of the store. While such reorganization involves considerable overhead, the payoff comes in subsequent searches where the external chaining greatly reduces the search time. It is assumed, of course, that the frequency of retrievals greatly exceeds the frequency of insertions and deletions, an assumption which holds true for most data storage and retrieval systems. When a deletion from a linked list causes the chain length to fall below a threshold, not necessarily the same threshold that triggered chain formation, the chain is destroyed and the entries reabsorbed into the hash table." The '499 Patent at Col. 6. *See also*, the '499 patent at Col. 6:38-65, 8:14-39, 8:62-9:2, and claim 3. To the extent that dynamically determining a maximum number of expired records is not disclosed by the '495 Patent in combination with Dirks, Thatte, the '663 Patent, the '499 Patent, or the Opportunistic Garbage Collection References,, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with the '495 Patent. For example, both Linux 2.0.1 and the '495 Patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. For example, when invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX.

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing	7. A method for storing	To the extent the preamble is a limitation, the '495 Patent discloses a method

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	for storing and retrieving information records using a chain of records to store and provide access to the records, at least some of the records automatically expiring. The '495 Patent also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using a linear probing technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '495 Patent claims "[a] method for storing and retrieving information records using hashing techniques to provide rapid access to said records and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring." U.S. Pat. No. '495 col.12:37-41. *See also* '495 Patent col. 4:33-64; *id.* at 8:65-9:9 (code defining hash table). As discussed in [1a/5a] and [1b/5b], it would have been obvious to one of ordinary skill in the art that an external chaining technique could be used instead of a linear probing technique to store records with the same hash address. It would have been obvious to one of ordinary skill in the art that a linked list could be used to store records with the same hash value rather than a chain of records. Linear probing and external chaining are two methods to solve the problem of collisions in a hash table. Both were commonly known by those skilled in the art, as Nemes admits in the '120 patent. See U.S. Pat. No. '120 col. 1:53-59 ("Some form of collision resolution must therefore be provided. For example, the simple strategy called 'linear probing,' . . . is often used. Another method for resolving collisions is called 'external chaining.'"). Both of these methods of collision resolution are described in the prior art cited by both the '495 and '120 patents. See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		Science and Information Processing, pp. 513, 518, 1973. Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Further, U.S. Patent No. 5,289,499 ("'499 patent), discloses the use of external chaining techniques for performing storage and retrieval in an information system. *See '499* patent abstract ("In order to provide efficient and graceful operation under varying load conditions, the system shifts between collision avoidance by linear probing with open addressing when the load is below a threshold, and collision avoidance by external chaining when the load is above a threshold"); '499 patent, Figs. 3-5 (use of external chaining – linked list to perform information storage and retrieval); '499 patent, col. 2:60-65, 3:5-9, col. 5:63-65; 8:53-9:20, claims 1-3 (external chaining a required element). Thus, one of ordinary skill in the art would have been motivated by Knuth, Weiss, or the '499 patent to apply the teachings of the '495 patent to hash tables with external chaining using linked lists. *See also*, the '499 patent at Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1. The records in the system the '495 Patent discloses includes records, at least some of which automatically expire. The '495 Patent discloses that "data records become obsolete merely by the passage of time or by the occurrence of some event" and that "[i]f such expired, lapsed or obsolete records are not removed from the storage table, they will, in time, seriously degrade or contaminate the performance of the retrieval system." U.S. Pat. No. '495 col. 4:23-28. *See also* '495 Patent col. 5:22-29.

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	The '495 Patent discloses accessing a chain of records. The '495 Patent also discloses accessing a chain of records having same hash address. For example, Claim 5 of the '495 patent is almost identical to [7a], claiming "accessing a chain or [sic] records having the same hash address." U.S. Pat. No. '495 col. 12:43-44. The only difference between the two patents is the reference to a chain of records instead of a linked list. *See also* '495 Pat. col. 5:45-56, *id.* at 10:2511:16 (search table function code); *id.* at Figs. 3-4. Though this describes accessing a chain of records rather than a linked list, as discussed above in sections [1a/5a] and [1b/5b], it would have been obvious to one of ordinary skill in the art that a linked list could be used to store the records instead of a chain of records. It would have been obvious to one of ordinary skill in the art that a linked list could be used to store records with the same hash value rather than a chain of records. Linear probing and external chaining are two methods to solve the problem of collisions in a hash table. Both were commonly known by those skilled in the art, as Nemes admits in the '120 patent. See U.S. Pat. No. '120 col. 1:53-59 ("Some form of collision resolution must therefore be provided. For example, the simple strategy called 'linear probing,' . . . is often used. Another method for resolving collisions is called 'external chaining.'"). Both of these methods of collision resolution are described in the prior art cited by both the '495 and '120 patents. See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 513, 518, 1973. Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		known as *open addressing*, is an alternative to resolving collisions with linked lists."). Further, U.S. Patent No. 5,289,499 ("'499 patent), discloses the use of external chaining techniques for performing storage and retrieval in an information system. *See* '*499* patent abstract ("In order to provide efficient and graceful operation under varying load conditions, the system shifts between collision avoidance by linear probing with open addressing when the load is below a threshold, and collision avoidance by external chaining when the load is above a threshold"); '499 patent, Figs. 3-5 (use of external chaining – linked list to perform information storage and retrieval); '499 patent, col. 2:60-65, 3:5-9, col. 5:63-65; 8:53-9:20, claims 1-3 (external chaining a required element). Thus, one of ordinary skill in the art would have been motivated by Knuth, Weiss, or the '499 patent to apply the teachings of the '495 patent to hash tables with external chaining using linked lists. *See also,* the '499 patent at Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	The '495 Patent discloses identifying at least some of the automatically expired ones of the records. For example, Claim 5 of the '495 patent is substantively identical to [7b], claiming "identifying the automatically expired ones of said records." U.S. Pat. No. '495 col. 12:45-46. *See also* '495 Pat. col. 5:36-56, *id.* at 6:53-8:16, 10:25-11:16 (search table function code); *id.* at Figs. 3-4, Abstract.
[3c] removing at least some of the automatically expired records from the linked list when the linked	[7c] removing at least some of the automatically expired records from the linked list when the linked	The '495 Patent discloses removing at least some of the automatically expired records from the chain of records when the chain of records is accessed. For example, Claim 5 of the '495 patent is almost identical to [7c], claiming

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
list is accessed.	list is accessed, and	"removing all automatically expired records from said chain of records each time said chain is accessed." U.S. Pat. No. '495 col. 12:47-48. The only differences between the two patents are (1) the reference to a chain of records instead of a linked list and (2) the deletion of all of the records in the '495 patent instead of some of the records. . *See also* '495 Pat. col. 5:36-57, 6:53-8:16, *id.* at 10:25-11:16 (search table function code); *id.* at Figs. 3-4. Though this describes accessing a chain of records rather than a linked list, as discussed above in sections [1a/5a] and [1b/5b], it would have been obvious to one of ordinary skill in the art that a linked list could be used to store the records instead of a chain of records. For example, the use of a linked list to store records is discussed in detail in the '499 patent. For example, the '499 patent discloses in part, "Another technique for resolving collisions is called external chaining. In this technique, each hash table position is able to store all records hashing to that location. More particularly, a linked list is used to store the actual records outside of the hash table. The hash table entry, then, is no more than a pointer to the head of the linked list. The linked list is itself searched sequentially when retrieving or storing a record. Deletion is accomplished by adjusting pointers to eliminate the deleted record from the linked list." *See* the '499 patent at column 2. *See, also*, the '499 patent at Abstract, Fig. 3-5, Col. 2:9-48, 2:50-3:7, 5:63-6:28, 7:10-10:5, claim 1 Moreover, the deletion of all of the expired records includes the deletion of at least some of the expired records and thus satisfies this element of the '120 patent.
	[7d] inserting, retrieving or deleting one of the	The '495 patent discloses inserting, retrieving, or deleting one of the records from the system following the step of removing.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
	records from the system following the step of removing.	For example, Claim 5 of the '495 patent is substantively identical to [7d], claiming "inserting, retrieving or deleting one of said records from said system following said step of removing." U.S. Pat. No. '495 col. 12:50-51. *See also* '495 Patent col. 5:65-8:16; *id.* at 9:10-10:20 (insert, retrieve, and delete function code); *id.* at Figs. 5-7. Further, the '499 patent discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, the Abstract of the '499 Patent discloses in part, "Insertion, deletion and retrieval operations are arranged to switch dynamically between the two collision avoidance stratagems as the local loading factor on the system, as measured by the number of records hashed to the same address, crosses preselected thresholds." Further, the '499 discloses "In FIG. 5 there is shown a flowchart of a record deletion process. Starting at start box 60, box 61 is entered where the search key is hashed to provide a hash table cell location. In box 62, the cell count field at that cell location is decremented by one. Decision box 63 is then entered to determine whether or not the contents of that cell is a list pointer. If it is not, box 68 is entered to use any known table deletion algorithm to remove the record from the hash table. As previously noted, the record can merely be marked "deleted" and left in place or can by physically deleted by some algorithm such as Knuth's algorithm. The process terminates in terminal box 66. If it is determined in decision box 63 that the contents of the cell is a list pointer, box 64 is entered where the record to be deleted is removed from the linked list. This is easily accomplished by adjusting the pointer in the chain just before the record to be deleted to point the the record following the record to deleted. The storage space of the thus "deleted" record can then be returned

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		to free storage space for future assignment to another record." See the '499 Patent, columns 8-9. *See also*, the '499 patent at Abstract, Fig. 3-5, Col. 1:29-2:8, 2:9-48, 2:50-3:7, 5:7-6:28, 7:10-10:5, claim 1.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	the '495 Patent combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14.

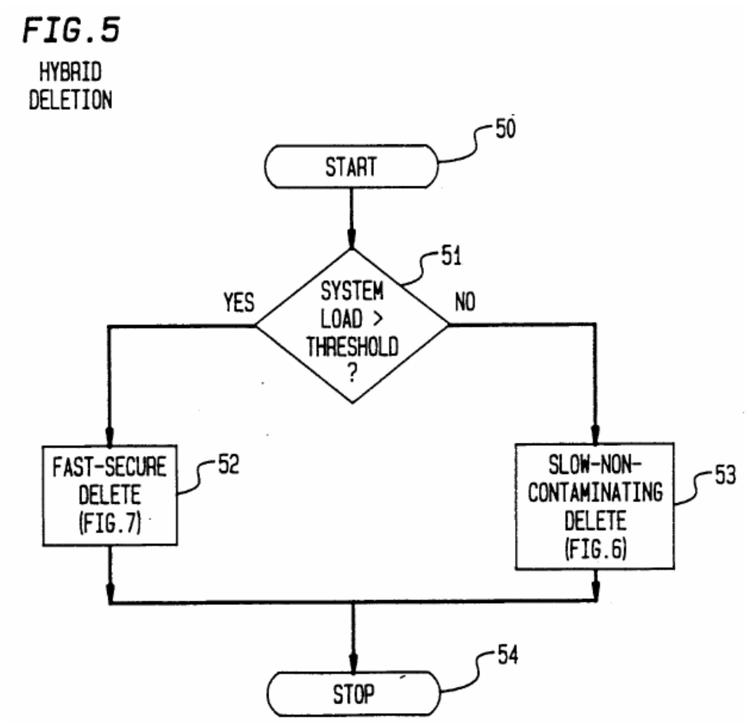
EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both the '495 Patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with the '495 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with the '495 Patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in the '495 Patent with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining the '495 Patent with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine the '495 Patent with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in the '495 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining the '495 Patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine the '495 Patent with Thatte. Alternatively, it would also be obvious to combine the '495 Patent with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		Id. at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both the '495 Patent and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with the '495 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with the '495 Patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine the '495 Patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both the '495 Patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as the '495 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with the '495 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with the '495 Patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '495 Patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		ordinary skill in the art would have been motivated to combine the system disclosed in the '495 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in the '495 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. The '499 patent discloses an information storage and retrieval system further including means for dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, the '499 patent describes performing a dynamic determination of the number of records hashed to the same cell in the hash table. Then determining if this number exceeds a certain threshold. If the threshold is exceeded, then the system described by the '499 patent reorganizes and removes the records at this location. See the '499 Patent at Col. 6, which

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		recites in part "In accordance with the present invention, the major advantages of both techniques, open addressing and external chaining, are achieved in the same system. More particularly, these two techniques are combined in one system, and the actual storage strategy is selected dynamically, depending on the then current local load factor. Initially, all records are stored in the hash table using the open addressing with linear probing technique. When the local load factor exceeds a preselected threshold, the system shifts dynamically to the external chaining technique. That is, while inserting or deleting a record, the local load factor, as reflected in the number of records hashed to this same hash table cell, is examined. If this number exceeds a threshold, the records hashed to this cell address are reorganized, removed from the hash table itself and organized into an external chain in another part of the store. While such reorganization involves considerable overhead, the payoff comes in subsequent searches where the external chaining greatly reduces the search time. It is assumed, of course, that the frequency of retrievals greatly exceeds the frequency of insertions and deletions, an assumption which holds true for most data storage and retrieval systems. When a deletion from a linked list causes the chain length to fall below a threshold, not necessarily the same threshold that triggered chain formation, the chain is destroyed and the entries reabsorbed into the hash table." The '499 Patent at Col. 6. *See also*, the '499 patent at Col. 6:38-65, 8:14-39, 8:62-9:2, and claim 3. To the extent that dynamically determining a maximum number of expired records is not disclosed by the '495 Patent in combination with Dirks, Thatte, the '663 Patent, the '499 Patent, or the Opportunistic Garbage Collection References,, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination**
		with the '495 Patent. For example, both Linux 2.0.1 and the '495 Patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. For example, when invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. Line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. Line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. See lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. See lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. See lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. Line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. Line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. See lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. See lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. See line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. See lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. Line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. Line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. See line 1135. In this way, the function

EXHIBIT B-12

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		`rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. Line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. Line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Pat. No. 5,121,495 to Nemes ("the '495 Patent") alone and in combination
		a linked list.

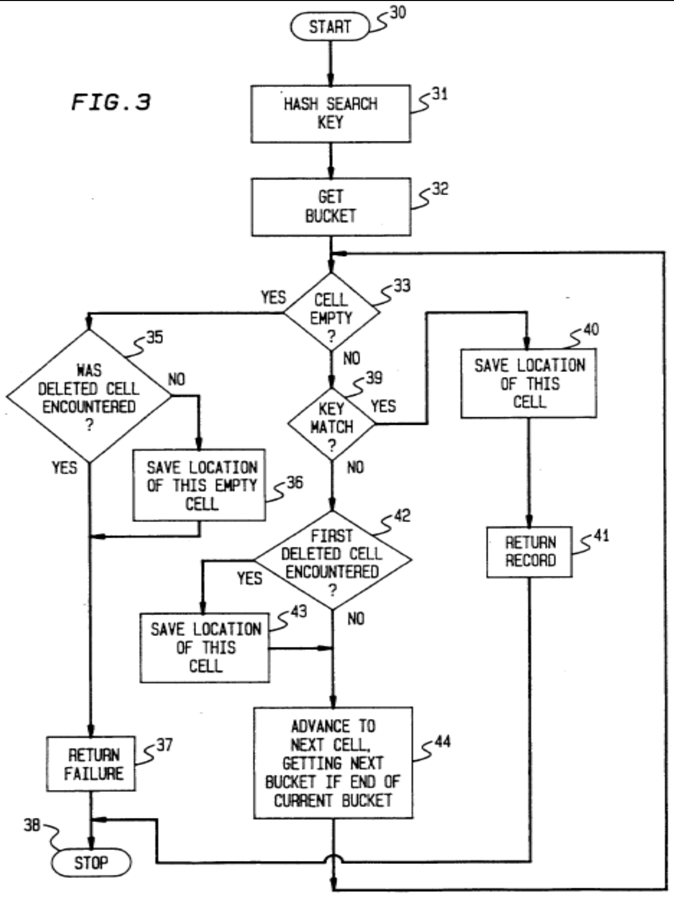
EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, the '663 Patent discloses an information storage and retrieval system. For example, The '663 Patent discloses "[a] method and apparatus for performing storage and retrieval in an information storage system [] which uses the hashing technique." U.S. Patent No. 4,996,663 to Nemes at Abstract.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	The '663 Patent discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. The '663 Patent also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '663 Patent discloses "[a] method and apparatus for performing storage and retrieval in an information storage system [] which uses the hashing technique." *Id*. Furthermore, the '663 Patent discloses that: [a] hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions are usually not one-to-one in that they map many distinct keys to the same location.

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes (“the ’663 Patent”)
		Id. at 4:41-50. When such a hash table data base is stored on a slower external storage [], the hash table is best organized as a consecutively numbered circular sequence of larger, multi-cell, fixed-sized storage units, each of which is termed a bucket. A bucket is the largest physically efficient input/output unit of the storage mechanism, such as a disk track in a disk storage unit. Each bucket consists of a sequence of consecutively numbered cells. The hashing function operates on the search key to translate or map the search key into a bucket number or address. Then the entire bucket is retrieved in a single access or probe, and the entire bucked may be processed at RAM speed. *Id*. at 4:55-68. The hashing technique The ’663 Patent discloses is better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		FIG.3  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 NO YES SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 RETURN RECORD 41 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP *Id*. at Figure 3. "Starting in box 30 of the retrieve procedure, the search key of the record to be

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		retrieved is hashed in box 31 to provide the address of a bucket." *Id*. at 5: 23-25. "It can be seen that boxes 33, 39 and 44 operate to search for a cell with matching key by linear probing with open addressing." *Id*. at 5:63-65. The '663 Patent further qualifies the necessity of linear probing by disclosing that: > [a] disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing 'collisions' in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called 'rehashing') must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the desired storage location will resolve the collision. This latter technique is called linear probing. *Id*. at 1:40-48. Furthermore, The '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	The '663 Patent discloses a record search means utilizing a search key to access the linked list. The '663 Patent also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		For example, the '663 Patent discloses "[a] method and apparatus for performing storage and retrieval in an information storage system [] which uses the hashing technique." *Id.* at Abstract. Furthermore, the '663 Patent discloses that: [a] hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions are usually not one-to-one in that they map many distinct keys to the same location. *Id.* at 4:41-50. When such a hash table data base is stored on a slower external storage [], the hash table is best organized as a consecutively numbered circular sequence of larger, multi-cell, fixed-sized storage units, each of which is termed a bucket. A bucket is the largest physically efficient input/output unit of the storage mechanism, such as a disk track in a disk storage unit. Each bucket consists of a sequence of consecutively numbered cells. The hashing function operates on the search key to translate or map the search key into a bucket number or address. Then the entire bucket is retrieved in a single access or probe, and the entire bucked may be processed at RAM speed. *Id.* at 4:55-68.

EXHIBIT B-13

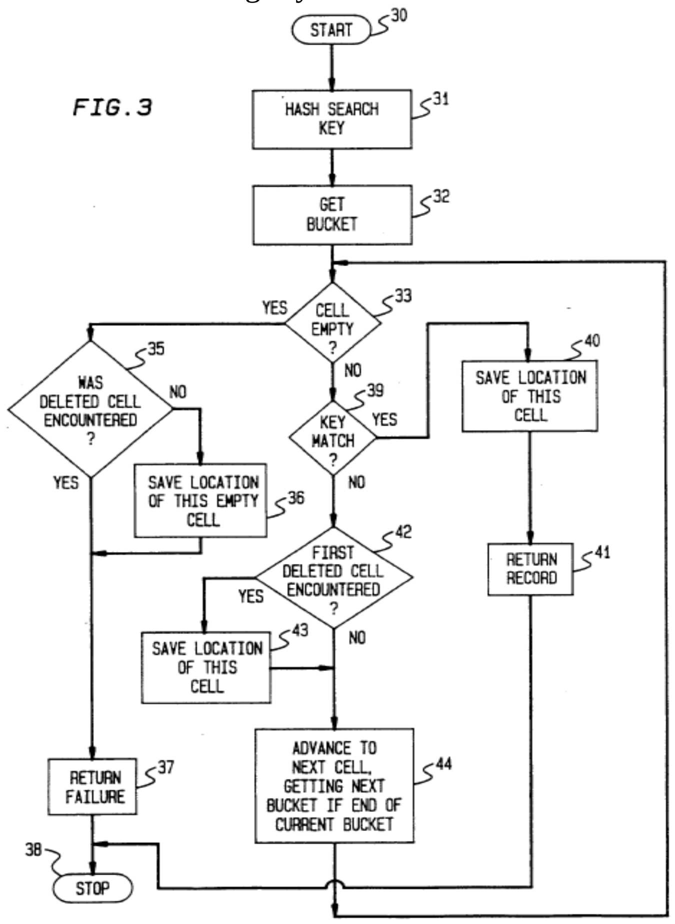
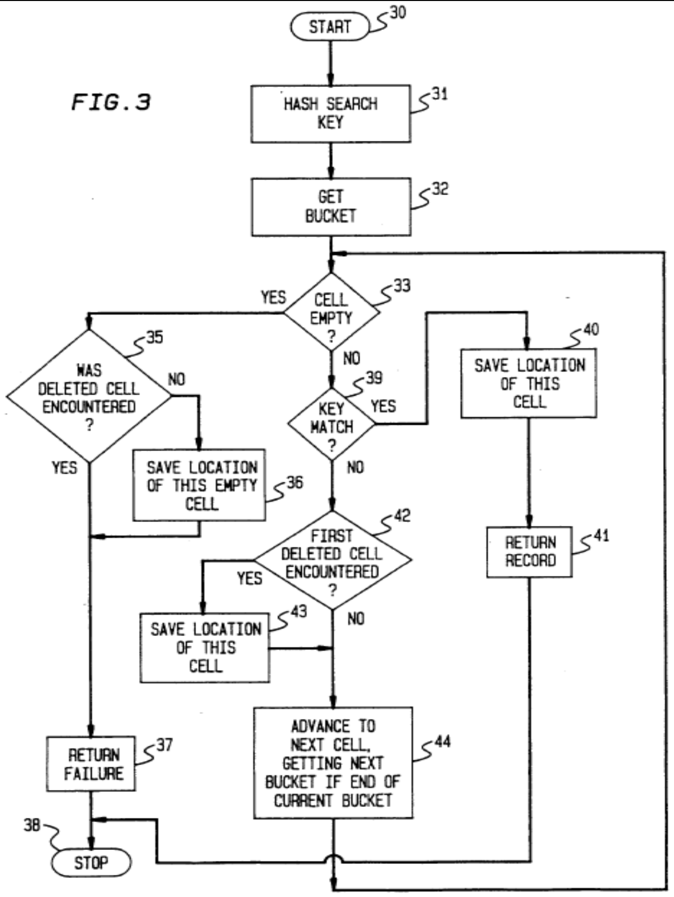
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		The hashing technique the '663 Patent discloses is better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 YES NO SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP

FIG. 3

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		Id. at Figure 3. "Starting in box 30 of the retrieve procedure, the search key of the record to be retrieved is hashed in box 31 to provide the address of a bucket." *Id*. at 5: 23-25. "It can be seen that boxes 33, 39 and 44 operate to search for a cell with matching key by linear probing with open addressing." *Id*. at 5:63-65. The '663 Patent further qualifies the necessity of linear probing by disclosing that [a] disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing 'collisions' in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called 'rehashing') must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the desired storage location will resolve the collision. This latter technique is called linear probing. *Id*. at 1:40-48.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	The '663 Patent discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. The '663 Patent also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, the '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination,

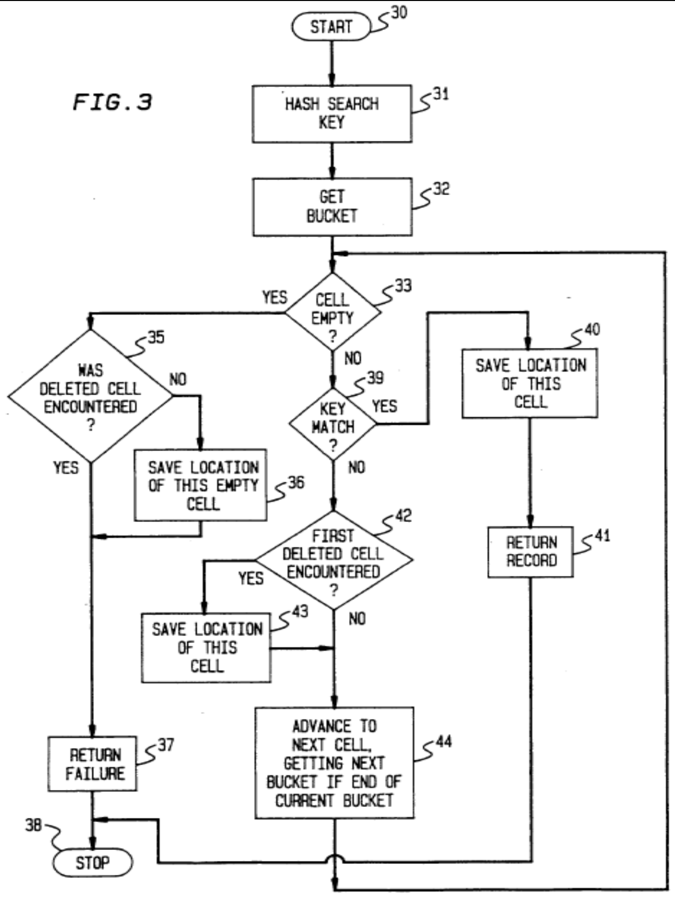
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41. The process disclosed by the '663 Patent is again better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		FIG.3  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 YES NO SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP *Id*. at Figure 3. Starting in box 30 of the retrieve procedure, the search key of the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		record to be retrieved is hashed in box 31 to provide the address of a bucket. In box 32, that bucket is retrieved in its entirety and stored in internal computer memory. Decision box 33 examines the first cell of that bucket to determine if the cell is empty or not. If the cell tested in decision box 42 is empty, the decision box 35 is entered to determine if a deleted cell was encountered before the empty cell was encountered. If no deleted cell was encountered, box 36 is entered to save the location of the empty cell, since this is the first empty cell in the bucket and hence can be used to store a new record. If any deleted cells were encountered prior to the empty cell, the location of the first deleted cell would have already been saved []. It should be recalled that a deleted cell can also be used to store a new record. *Id*. at 21-40. [I]n order to insert or store a record, starting at start box 45, it is assumed that the retrieve procedure of FIG 3 has already been invoked to see if a record with this key has already been stored in the data base. If not, the retrieve procedure of FIG. 3 returns a failure indication along with the location of the first empty or first deleted cell encountered in its search for the record. In the insert procedure [], it is therefore only necessary to store the new record in the location provided by the retrieve procedure in box 46, return the accessed bucket to the data base in box 47 and terminate the insert procedure in box 48. *Id*. at 6:8-20, Figures 3 and 4. Upon an insertion, if a deleted cell is returned in the process described above,

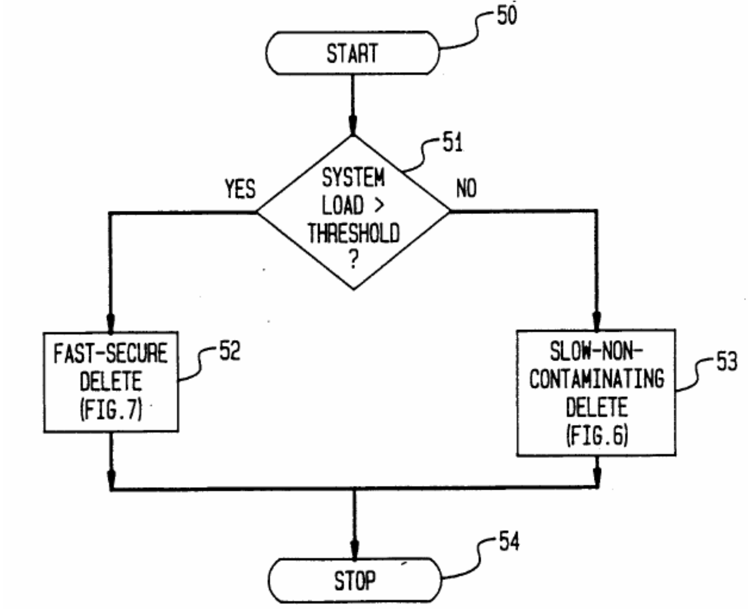
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		the new record replaces the contents in the deleted cell.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	The '663 Patent discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The '663 Patent also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, the '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41. The process disclosed by the '663 Patent is again better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		FIG.3  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 NO YES SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP *Id.* at Figure 3. Starting in box 30 of the retrieve procedure, the search key of the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		record to be retrieved is hashed in box 31 to provide the address of a bucket. In box 32, that bucket is retrieved in its entirety and stored in internal computer memory. Decision box 33 examines the first cell of that bucket to determine if the cell is empty or not. If the cell tested in decision box 42 is empty, the decision box 35 is entered to determine if a deleted cell was encountered before the empty cell was encountered. If no deleted cell was encountered, box 36 is entered to save the location of the empty cell, since this is the first empty cell in the bucket and hence can be used to store a new record. If any deleted cells were encountered prior to the empty cell, the location of the first deleted cell would have already been saved []. It should be recalled that a deleted cell can also be used to store a new record. *Id*. at 21-40. [I]n order to insert or store a record, starting at start box 45, it is assumed that the retrieve procedure of FIG 3 has already been invoked to see if a record with this key has already been stored in the data base. If not, the retrieve procedure of FIG. 3 returns a failure indication along with the location of the first empty or first deleted cell encountered in its search for the record. In the insert procedure [], it is therefore only necessary to store the new record in the location provided by the retrieve procedure in box 46, return the accessed bucket to the data base in box 47 and terminate the insert procedure in box 48. *Id*. at 6:8-20. Figures 3 and 4. Upon an insertion, if a deleted cell is returned in the process described above,

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		the new record replaces the contents in the deleted cell.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	The '663 Patent discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, the '663 Patent discloses that: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. *Id*. at 2:24-34. In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the '663 Patent discloses a hybrid deletion procedure that dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '663 Patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in the '663 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in the '663 Patent can be burdensome on the system, adding to the system's load and slowing down the system's

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in the '663 Patent is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent it is not disclosed in the '663 Patent, Dirks, Thatte, the Opportunistic Garbage Collection References, and/or Linux 2.0.1 disclose a means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes (“the ’663 Patent”)**
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both The ‘663 Patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as The ‘663 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with The '663 Patent nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with The '663 Patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in The '663 Patent with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining The '663 Patent with Thatte would be nothing more than

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine The '663 Patent with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in The '663 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining The '663 Patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine The '663 Patent with Thatte. Alternatively, it would also be obvious to combine The '663 Patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both The '663 Patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as The '663 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with The '663 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with The '663 Patent and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in The '663 Patent to dynamically determine the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in The '663 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in The '663 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. The '499 patent discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, the '499 patent describes performing a dynamic determination of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		the number of records hashed to the same cell in the hash table. Then determining if this number exceeds a certain threshold. If the threshold is exceeded, then the system described by the '499 patent reorganizes and removes the records at this location. See the '499 Patent at Col. 6, which recites in part "In accordance with the present invention, the major advantages of both techniques, open addressing and external chaining, are achieved in the same system. More particularly, these two techniques are combined in one system, and the actual storage strategy is selected dynamically, depending on the then current local load factor. Initially, all records are stored in the hash table using the open addressing with linear probing technique. When the local load factor exceeds a preselected threshold, the system shifts dynamically to the external chaining technique. That is, while inserting or deleting a record, the local load factor, as reflected in the number of records hashed to this same hash table cell, is examined. If this number exceeds a threshold, the records hashed to this cell address are reorganized, removed from the hash table itself and organized into an external chain in another part of the store. While such reorganization involves considerable overhead, the payoff comes in subsequent searches where the external chaining greatly reduces the search time. It is assumed, of course, that the frequency of retrievals greatly exceeds the frequency of insertions and deletions, an assumption which holds true for most data storage and retrieval systems. When a deletion from a linked list causes the chain length to fall below a threshold, not necessarily the same threshold that triggered chain formation, the chain is destroyed and the entries reabsorbed into the hash table." The '499 Patent at Col. 6. *See also*, the '499 patent at Col. 6:38-65, 8:14-39, 8:62-9:2, and claim 3. To the extent that dynamically determining a maximum number of expired records is not disclosed by The '663 Patent in combination with Dirks, Thatte,

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with The '663 Patent. For example, both Linux 2.0.1 and The '663 Patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. For example, when invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		`rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, the '663 Patent discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. the '663 Patent also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '663 Patent discloses "[a] method and apparatus for performing storage and retrieval in an information storage system [] which uses the hashing technique." U.S. Patent No. 4,996,663 to Nemes at Abstract. Furthermore, the '663 Patent discloses that [a] hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions are usually not one-to-one in that they map many distinct keys to the same location. *Id*. at 4:41-50. When such a hash table data base is stored on a slower external storage [], the hash table is best organized as a consecutively numbered circular sequence of larger, multi-cell, fixed-sized storage units, each of which is termed a bucket. A bucket is the largest physically efficient input/output unit of the storage mechanism, such as a disk track in a disk storage unit. Each bucket consists of a sequence of consecutively numbered cells. The hashing function operates on the search key to translate or map the search key into a bucket number or address. Then the entire bucket is retrieved in a single access or probe, and the entire bucked may be processed at RAM speed. *Id*. at 4:55-68. The hashing technique the '663 Patent discloses is better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

EXHIBIT B-13

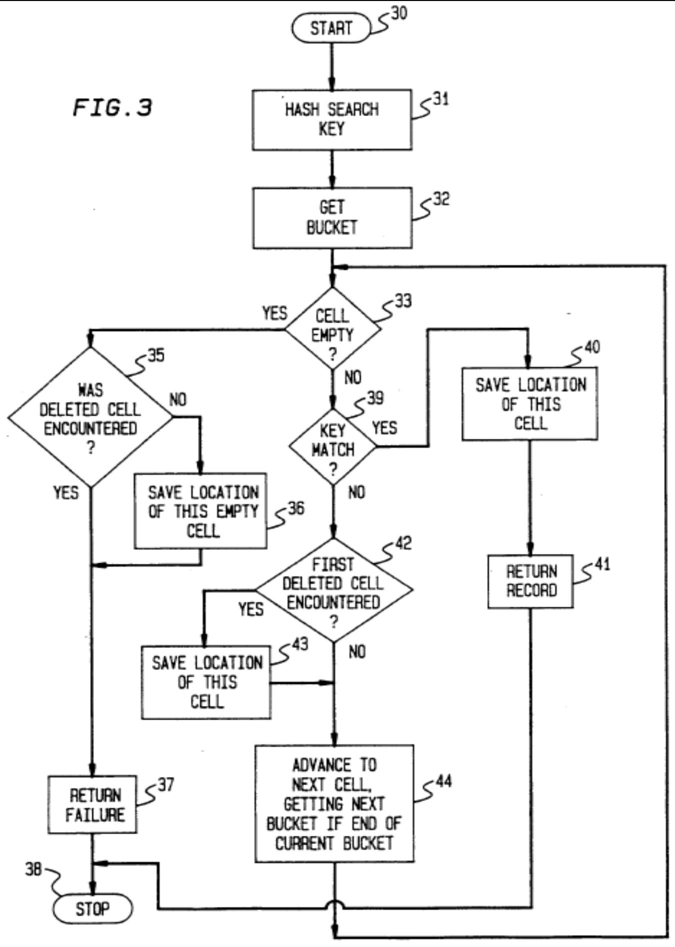
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		FIG. 3  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 NO YES SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP *Id*. at Figure 3.

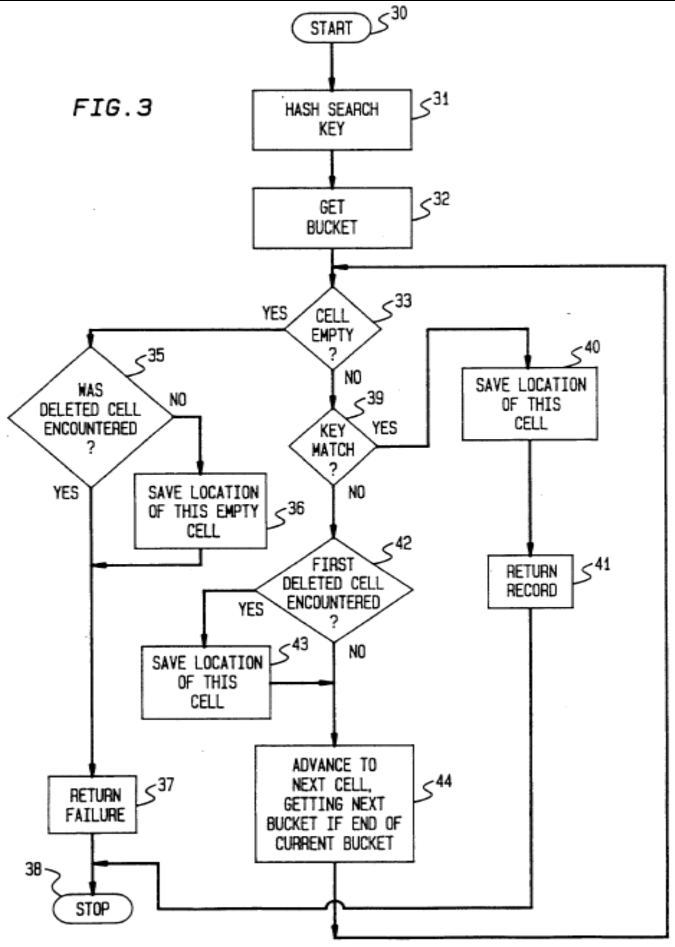
EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		"Starting in box 30 of the retrieve procedure, the search key of the record to be retrieved is hashed in box 31 to provide the address of a bucket." *Id*. at 5: 23-25. "It can be seen that boxes 33, 39 and 44 operate to search for a cell with matching key by linear probing with open addressing." *Id*. at 5:63-65. The '663 Patent further qualifies the necessity of linear probing by disclosing that > [a] disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing 'collisions' in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called 'rehashing') must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the desired storage location will resolve the collision. This latter technique is called linear probing. *Id*. at 1:40-48. Furthermore, the '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	The '663 Patent discloses accessing a linked list of records. The '663 Patent also discloses accessing a linked list of records having same hash address. For example, the '663 Patent discloses "[a] method and apparatus for

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		performing storage and retrieval in an information storage system [] which uses the hashing technique." U.S. Patent No. 4,996,663 to Nemes at Abstract. Furthermore, the '663 Patent discloses that: [a] hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions are usually not one-to-one in that they map many distinct keys to the same location. *Id*. at 4:41-50. When such a hash table data base is stored on a slower external storage [], the hash table is best organized as a consecutively numbered circular sequence of larger, multi-cell, fixed-sized storage units, each of which is termed a bucket. A bucket is the largest physically efficient input/output unit of the storage mechanism, such as a disk track in a disk storage unit. Each bucket consists of a sequence of consecutively numbered cells. The hashing function operates on the search key to translate or map the search key into a bucket number or address. Then the entire bucket is retrieved in a single access or probe, and the entire bucked may be processed at RAM speed. *Id*. at 4:55-68. The hashing technique the '663 Patent discloses is better understood with

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes (“the ’663 Patent”)**
		reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		 START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 NO YES SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP FIG. 3 *Id*. at Figure 3.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		"Starting in box 30 of the retrieve procedure, the search key of the record to be retrieved is hashed in box 31 to provide the address of a bucket." *Id*. at 5: 23-25. "It can be seen that boxes 33, 39 and 44 operate to search for a cell with matching key by linear probing with open addressing." *Id*. at 5:63-65. The '663 Patent further qualifies the necessity of linear probing by disclosing that: > [a] disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing 'collisions' in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called 'rehashing') must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the desired storage location will resolve the collision. This latter technique is called linear probing. *Id*. at 1:40-48.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	The '663 Patent discloses identifying at least some of the automatically expired ones of the records. For example, the '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41.
[3c] removing at least	[7c] removing at least	The '663 Patent discloses removing at least some of the automatically expired

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
some of the automatically expired records from the linked list when the linked list is accessed.	some of the automatically expired records from the linked list when the linked list is accessed, and	records from the linked list when the linked list is accessed. For example, the '663 Patent discloses that "[i]n times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as 'deleted' and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain. . . ." *Id*. at 2:35-41. The process disclosed by the '663 Patent is again better understood with reference to Figure 3, which is a flow chart of a procedure for the retrieval of a record from the storage system:

EXHIBIT B-13

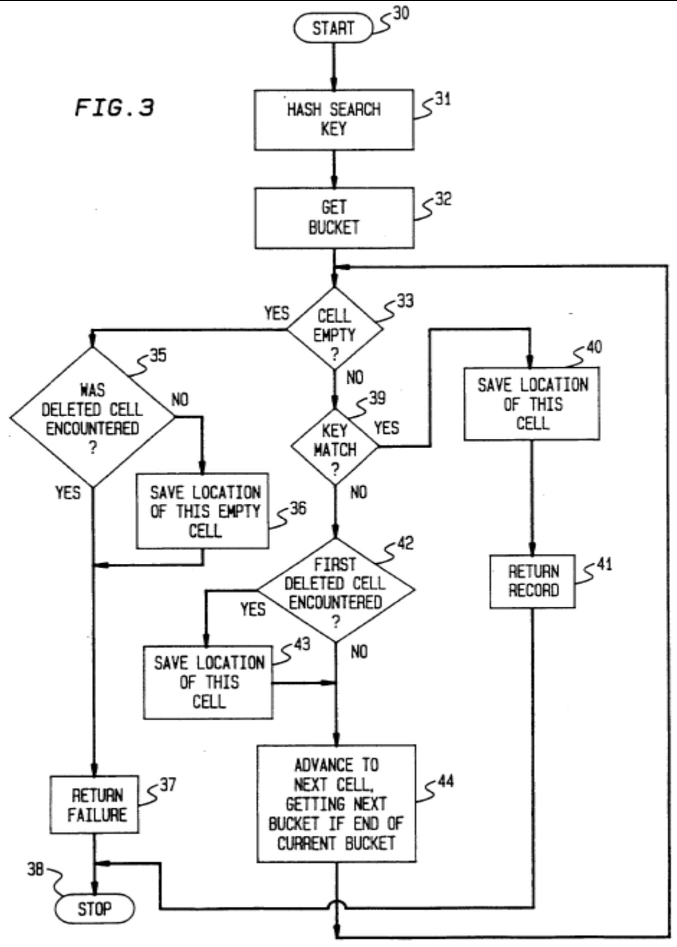
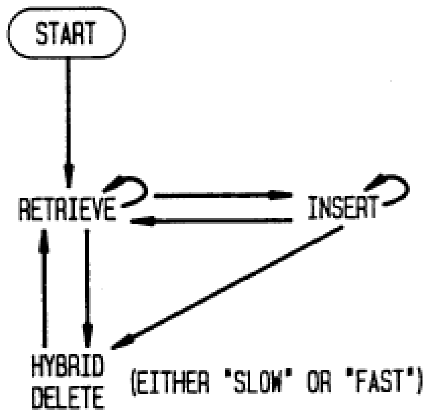
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes (“the ’663 Patent”)**
		FIG.3  START 30 HASH SEARCH KEY 31 GET BUCKET 32 CELL EMPTY ? 33 YES NO WAS DELETED CELL ENCOUNTERED ? 35 NO YES SAVE LOCATION OF THIS EMPTY CELL 36 KEY MATCH ? 39 YES NO SAVE LOCATION OF THIS CELL 40 RETURN RECORD 41 FIRST DELETED CELL ENCOUNTERED ? 42 YES NO SAVE LOCATION OF THIS CELL 43 ADVANCE TO NEXT CELL, GETTING NEXT BUCKET IF END OF CURRENT BUCKET 44 RETURN FAILURE 37 38 STOP *Id*. at Figure 3.

EXHIBIT B-13

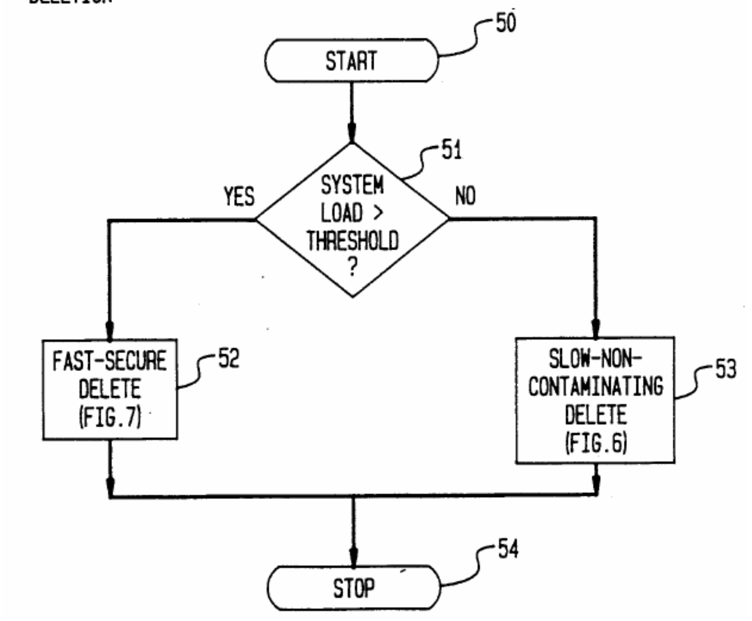
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		Starting in box 30 of the retrieve procedure, the search key of the record to be retrieved is hashed in box 31 to provide the address of a bucket. In box 32, that bucket is retrieved in its entirety and stored in internal computer memory. Decision box 33 examines the first cell of that bucket to determine if the cell is empty or not. If the cell tested in decision box 42 is empty, the decision box 35 is entered to determine if a deleted cell was encountered before the empty cell was encountered. If no deleted cell was encountered, box 36 is entered to save the location of the empty cell, since this is the first empty cell in the bucket and hence can be used to store a new record. If any deleted cells were encountered prior to the empty cell, the location of the first deleted cell would have already been saved []. It should be recalled that a deleted cell can also be used to store a new record. *Id*. at 21-40. [I]n order to insert or store a record, starting at start box 45, it is assumed that the retrieve procedure of FIG 3 has already been invoked to see if a record with this key has already been stored in the data base. If not, the retrieve procedure of FIG. 3 returns a failure indication along with the location of the first empty or first deleted cell encountered in its search for the record. In the insert procedure [], it is therefore only necessary to store the new record in the location provided by the retrieve procedure in box 46, return the accessed bucket to the data base in box 47 and terminate the insert procedure in box 48. *Id*. at 6:8-20. Figures 3 and 4.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		Upon an insertion, if a deleted cell is returned in the process described above, the new record replaces the contents in the deleted cell.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	The '663 Patent discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, the process disclosed in the '663 Patent is better understood with reference to Figure 8, which shows a state diagram illustrating the sequence in which the various procedures – inserting, deleting, and retrieving – can be called. *Id*. at 8:35-37, Figures 3-5.  START RETRIEVE INSERT HYBRID DELETE (EITHER "SLOW" OR "FAST") *Id*. at Figure 8. The '663 Patent discloses that: [s]tarting at box 85, the retrieve procedure of FIG. 3 must be

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		called first, before any other procedure. Therefore, the insert procedure of FIG. 4, or the hybrid delete procedure of FIG. 5 can be called, or the retrieve procedure of FIG. 3 reinvoked. Either the retrieve procedure of FIG. 3 or the hybrid delete procedure of FIG. 5 can be called after the insert procedure, but only the retrieve procedure can be called following the hybrid delete procedure of FIG. 5. The circular arrows at the retrieve and insert procedures indicate that these procedures can be called repetitively without calling any other procedure. *Id*. at 8:37-47.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	The '663 Patent discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, the '663 Patent discloses that: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. *Id*. at 2:24-34. In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		FIG.5 HYBRID DELETION  50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the '663 Patent discloses a hybrid deletion procedure that dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '663 Patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in the '663 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in the '663 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in the '663 Patent is avoiding these problems. One of ordinary skill in the art would have known that

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent it is not disclosed in the '663 Patent, Dirks, Thatte, the Opportunistic Garbage Collection References, and/or Linux 2.0.1 disclose dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both The '663 Patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as The '663 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with The '663 Patent nothing more than the predictable use of prior art elements according to their established functions.

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with The '663 Patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in The '663 Patent with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining The '663 Patent with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine The '663 Patent with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in The '663 Patent can be

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining The '663 Patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine The '663 Patent with Thatte. Alternatively, it would also be obvious to combine The '663 Patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: > When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both The '663 Patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		table implementations such as The '663 Patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with The '663 Patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with The '663 Patent and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in The '663 Patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in The '663 Patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in The '663 Patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. The '499 patent discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, the '499 patent describes performing a dynamic determination of the number of records hashed to the same cell in the hash table. Then determining if this number exceeds a certain threshold. If the threshold is exceeded, then the system described by the '499 patent reorganizes and removes the records at this location. See the '499 Patent at Col. 6, which recites in part "In accordance with the present invention, the major advantages of both techniques, open addressing and external chaining, are achieved in the same system. More particularly, these two techniques are combined in one

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		system, and the actual storage strategy is selected dynamically, depending on the then current local load factor. Initially, all records are stored in the hash table using the open addressing with linear probing technique. When the local load factor exceeds a preselected threshold, the system shifts dynamically to the external chaining technique. That is, while inserting or deleting a record, the local load factor, as reflected in the number of records hashed to this same hash table cell, is examined. If this number exceeds a threshold, the records hashed to this cell address are reorganized, removed from the hash table itself and organized into an external chain in another part of the store. While such reorganization involves considerable overhead, the payoff comes in subsequent searches where the external chaining greatly reduces the search time. It is assumed, of course, that the frequency of retrievals greatly exceeds the frequency of insertions and deletions, an assumption which holds true for most data storage and retrieval systems. When a deletion from a linked list causes the chain length to fall below a threshold, not necessarily the same threshold that triggered chain formation, the chain is destroyed and the entries reabsorbed into the hash table." The '499 Patent at Col. 6. *See also*, the '499 patent at Col. 6:38-65, 8:14-39, 8:62-9:2, and claim 3. To the extent that dynamically determining a maximum number of expired records is not disclosed by The '663 Patent in combination with Dirks, Thatte, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with The '663 Patent. For example, both Linux 2.0.1 and The '663 Patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		For example, when invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")**
		predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function

EXHIBIT B-13

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 4,996,663 to Nemes ("the '663 Patent")
		`rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, The '237 patent discloses an information storage and retrieval system. For example, the '237 patent discloses a circular array having linked lists chained to array entries for storing time storage entries. *See* the '237 patent at Col. 3:19-4:25 and FIG. 2.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	The '237 patent discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. The '237 patent also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '237 patent discloses using linked lists to store and provide access to records: "If there are several structures sharing the same time slot (TS), that is, the same cell of the circular array, such as time storage entries 60, 65 in FIG. 2, then the time storage entries are joined to one another in a linked list, as in linked list 50, or linked list 55 linking the plurality of TSE's 90, 91, 92 sharing the same TS of cell 4 in FIG. 4." *See* the '237 patent, Col. 3:58-3:64, FIG. 2 and FIG. 4. In addition, the '237 patent discloses a hashing means, for example: "the time slot (TS) associated with the expiration time is computed at step 240, which corresponds to a particular numbered cell of the circular array, as

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		measured from the reference cell." *See* the '237 patent at Col. 7:38-7:41. Furthermore, the '237 patent discloses external chaining, for example: "If there are several structures sharing the same time slot (TS), that is, the same cell of the circular array, such as time storage entries 60, 65 in FIG. 2, then the time storage entries are joined to one another in a linked list, as in linked list 50, or linked list 55 linking the plurality of TSE's 90, 91, 92 sharing the same TS of cell 4 in FIG. 4." *See* the '237 patent, Col. 3:58-3:64, FIG. 2 and FIG. 4. Finally, the '237 patent discloses automatically expiring, for example: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39. "Deletion occurs every time a data packet is successfully sent by a local (sending) node and an acknowledgement is received from a remote (receiving) node. The timer for that data then is no longer needed and is deleted. By contrast, popping occurs only every so often when it is desired to see which timers have expired between the reference time (cell 1) and some other time, say an elapsed time 10 units from the reference time (indicated as cell 11 in FIG. 4, marked by reference number 100)." *See* the '237 patent at Col. 5:54-5:65.

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	The '237 patent discloses a record search means utilizing a search key to access the linked list. The '237 patent also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, the '237 patent discloses a computing a slot of the circular array based on the expiration time of the timer to be stored. As discussed above linked lists of records are attached to slots in the circular array: "the time slot (TS) associated with the expiration time is computed at step 240, which corresponds to a particular numbered cell of the circular array, as measured from the reference cell." *See* the '237 patent at Col. 7:38-7:41.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	The '237 patent discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. The '237 patent also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, the '237 patent discloses a means for identifying and removing expired records. Timers expire either when a packet has been received or when the timer times out: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39. "After traversing and processing an expired time storage entry (popping), the time storage entry is removed from the linked list of TSEs, and after all such TSEs are removed from a given non-empty cell, the array entry associated with the non-empty cell is removed from the doubly linked list, and any memory associated any data structure is reallocated." *See* the '237 patent at col. 6:36-6:41. The citation above also discloses removing expired timers when accessing the linked list.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	The '237 patent discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. The '237 patent also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, the '237 patent discloses means for inserting, retrieving and deleting records: "FIG. 6 shows a flowchart depicting how a calling function (or more generally, the protocol program) may call and interact with three functions or subroutines of the present invention, which would be extern functions in the C language: the Add function (which adds a time storage entry), the Delete function (which deletes a time storage entry), and the Pop function (which executes the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		handler(s) associated with a time storage entry and removes the TSE)." *See* the '237 patent at Col. 7:3-7:10. In addition, the '237 patent discloses using the record search means and at the same time removing at least some of the expired ones from the linked list of records: "The Delete function, an external function, receives passed parameters from the calling function that identifies which time storage entry (TSE) is to be deleted, as indicated by step 300. Such passed parameters include, for example, the data packet ID. The passed parameters are used to identify the particular TSE referenced by the circular array that needs to be deleted, as indicated by step 310. The time slot (TS) that the TSE is found in is then computed by the processor, in step 320. The circular array cell associated with the TS is found by processor 15, and in step 330 the cell is checked to see if it contains more than one TSE. If so, the TSE is deleted from the linked list associated with the cell." *See* the '237 patent at Col. 7:62-8:7. As discussed above, entries expire when a packet has been successfully delivered and are then deleted using the Delete function: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39.

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	The '237 patent combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20).

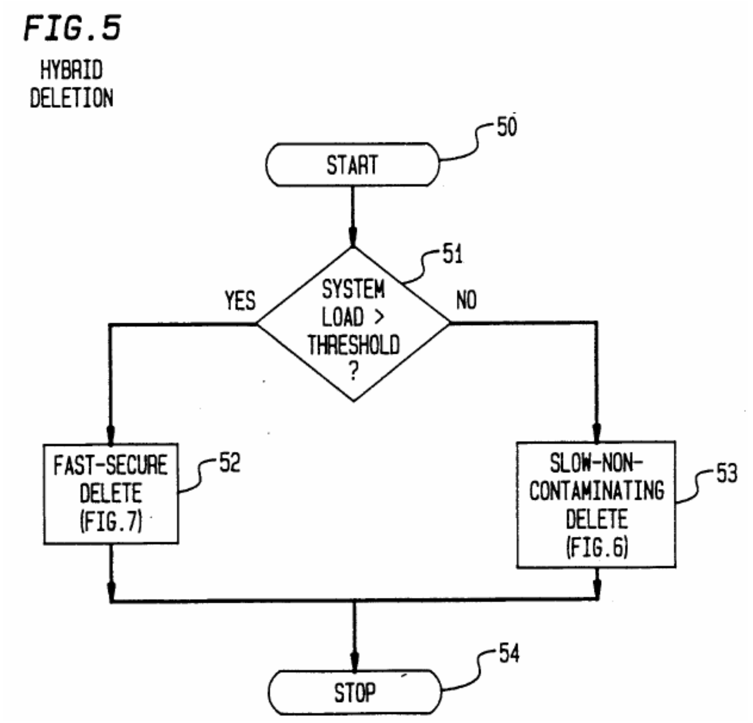
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		Id. at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both the '237 patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with the '237 patent nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with the '237 patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in the '237 patent with the

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining the '237 patent with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine the '237 patent with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in the '237 patent can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining the '237 patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		7:10-15. Thus, the '120 patent provides motivations to combine the '237 patent with Thatte. Alternatively, it would also be obvious to combine the '237 patentwith the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5.  FIG.5 HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both the '237 patent and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		combining the '663 patent's deletion decision procedure with the '237 patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with the '237 patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine the '237 patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection References disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both the '237 patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash

EXHIBIT B-14

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		table implementations such as the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with the '237 patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with the '237 patent and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '237 patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in the '237 patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter “the ’237 patent”)
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in the ‘237 patent can be burdensome on the system, adding to the system’s load and slowing down the system’s processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the ‘120 patent that “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” ‘120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by the ‘237 patent in combination with Dirks, Thatte, the ‘663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with the ‘237 patent. For example, both Linux 2.0.1 and the ‘237 patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		rt_cache_add invokes a function rt_garbage_collect. *See* Linux 2.0.1, route.c at lines 1341-1342. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function

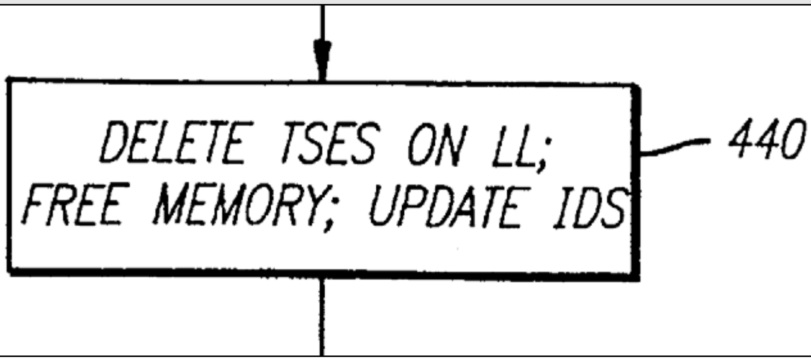
Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		`rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, The '237 patent discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. The '237 patent also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, the '237 patent discloses using linked lists to store and provide access to records: "If there are several structures sharing the same time slot (TS), that is, the same cell of the circular array, such as time storage entries 60, 65 in FIG. 2, then the time storage entries are joined to one another in a linked list, as in linked list 50, or linked list 55 linking the plurality of TSE's 90, 91, 92 sharing the same TS of cell 4 in FIG. 4." *See* the '237 patent, Col. 3:58-3:64, FIG. 2 and FIG. 4. In addition, the '237 patent discloses a hashing technique, for example: "the time slot (TS) associated with the expiration time is computed at step 240, which corresponds to a particular numbered cell of the circular array, as measured from the reference cell." *See* the '237 patent at Col. 7:38-7:41. Furthermore, the '237 patent discloses external chaining, for example:

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		"If there are several structures sharing the same time slot (TS), that is, the same cell of the circular array, such as time storage entries 60, 65 in FIG. 2, then the time storage entries are joined to one another in a linked list, as in linked list 50, or linked list 55 linking the plurality of TSE's 90, 91, 92 sharing the same TS of cell 4 in FIG. 4." *See* the '237 patent, Col. 3:58-3:64, FIG. 2 and FIG. 4. Finally, the '237 patent discloses automatically expiring, for example: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39. "Deletion occurs every time a data packet is successfully sent by a local (sending) node and an acknowledgement is received from a remote (receiving) node. The timer for that data then is no longer needed and is deleted. By contrast, popping occurs only every so often when it is desired to see which timers have expired between the reference time (cell 1) and some other time, say an elapsed time 10 units from the reference time (indicated as cell 11 in FIG. 4, marked by reference number 100)." *See* the '237 patent at Col. 5:54-5:65.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	The '237 patent discloses accessing a linked list of records. The '237 patent also discloses accessing a linked list of records having same hash address.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		For example, the '237 patent discloses a computing a slot of the circular array based on the expiration time of the timer to be stored. As discussed above linked lists of records are attached to slots in the circular array: "the time slot (TS) associated with the expiration time is computed at step 240, which corresponds to a particular numbered cell of the circular array, as measured from the reference cell." *See* the '237 patent at Col. 7:38-7:41.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	The '237 patent discloses identifying at least some of the automatically expired ones of the records. For example, the '237 patent discloses a means for identifying and removing expired records. Timers expire either when a packet has been received or when the timer times out: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39. "After traversing and processing an expired time storage entry (popping), the time storage entry is removed from the linked list of TSEs, and after all such TSEs are removed from a given non-empty cell, the array entry associated with the non-empty cell is removed from the doubly linked list, and any memory associated any data structure is reallocated." *See* the '237 patent at col. 6:36-6:41.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		The citation above also discloses removing expired timers when accessing the linked list.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	The '237 patent discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, the '237 patent discloses a means for identifying and removing expired records. Timers expire either when a packet has been received or when the timer times out: "If a timer is to be deleted (canceled), such as if the data packet has been successfully received by a remote node and an acknowledgement signal associated with this packet is received by a local node, the TSE associated with the packet (say TSE 60) is looked up along with the cell in the circular array it is found in (here cell 6) and the TSE is deleted from the linked list it is found in (such as linked list 50)." *See* the '237 patent at Col. 5:33-5:39. "After traversing and processing an expired time storage entry (popping), the time storage entry is removed from the linked list of TSEs, and after all such TSEs are removed from a given non-empty cell, the array entry associated with the non-empty cell is removed from the doubly linked list, and any memory associated any data structure is reallocated." *See* the '237 patent at col. 6:36-6:41. The citation above also discloses removing expired timers when accessing the linked list
	[7d] inserting, retrieving	The '237 patent discloses inserting, retrieving or deleting one of the records

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
	or deleting one of the records from the system following the step of removing.	from the system following the step of removing. For example, the '237 patent discloses removing timers from the linked list followed by deleting records from the system: "Turning attention now to FIG. 9, there is shown the Pop function of the present invention, which receives from a calling function in step 400 the new time slot (TS.sub.new), which marks the elapsed time for the Pop function as well as identifies the new reference time, such as represented by cell marker 100 in FIGS. 4 and 5 described above. In step 410 the processor uses the information in TS.sub.new to compute a new reference cell (such as cell 11 in FIG. 4) which is greater in time from the old reference cell by an amount equal to the number of cells between the new minus the old reference cells times the unit of time represented by each cell (the granularity). The processor, starting at the old reference cell, TS.sub.0, (such as cell 1 in FIG. 4) traverses the non-empty cells in the doubly linked list (DLL) between the original reference cell, TS.sub.0 and the new reference cell, TS.sub.new as indicated by step 420. As indicated by step 430, the processor finds any TSE's in the non-empty cell and executes the handlers associated with the TSE. Thereafter, in step 440, the TSE is freed and any memory deallocated, and for each non-empty cell traversed all pointers associated with the non-empty cell are deleted, and any ID map is updated." *See* the '237 patent at Col. 8:19-8:39, FIG. 9.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	The '237 patent combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have

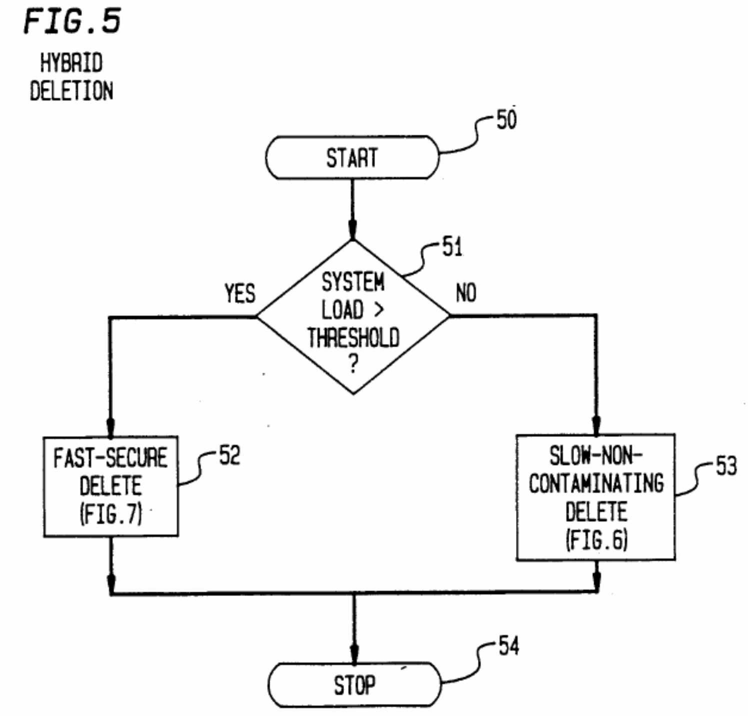
Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both the '237 patent and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with the '237 patent would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with the '237 patent and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in the '237 patent with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining the '237 patent with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine the '237 patent with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in the '237 patentcan be burdensome

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining the '237 patent with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine the '237 patent with Thatte. Alternatively, it would also be obvious to combine the '237 patent with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both the '237 patent and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with the '237 patent would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with the '237 patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine the '237 patent with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection References disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both the '237 patent and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as the '237 patent. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with the '237 patent would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with the '237 patent and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in the '237 patent to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in the '237 patent with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in the '237 patent can be burdensome on the system, adding to the system's load and slowing down the system's

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by the '237 patent in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with the '237 patent. For example, both Linux 2.0.1 and the '237 patent describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect`

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")**
		invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked

Asserted Claims From U.S. Pat. No. 5,893,120		U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter "the '237 patent")
		lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can

Asserted Claims From U.S. Pat. No. 5,893,120		**U.S. Patent No. 5,577,237 (filed Jan. 23, 1995) (hereinafter “the ’237 patent”)**
		remove from a linked list.

EXHIBIT C-1

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Van Wyk discloses an information storage and retrieval system. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 17 (November, 1986). Moreover, Van Wyk discloses "[h]ashing with lazy deletion is a simple algorithm for dynamic dictionaries whose occupants expire spontaneously. It is also compatible with the need to accommodate explicit user requests to delete elements. Even if the expiration times of active occupants change, one can use hashing with lazy deletion: if expiration times only increase, they can be changed by moving items further down hash chains; if they may also decrease, they may need to move back in their hash chain or even be deleted immediately. (An alternative would be to forget about keeping the hash chains in expiration-time order; this would make updating expiration times simpler, and would not change the asymptotic time complexity.)" *Id*. at 28-29.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records	Van Wyk discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Van Wyk also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Van Wyk discloses "a version of the dynamic dictionary problem

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
	automatically expiring,	in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17. Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*. Moreover, Van Wyk discloses that "balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree [] by a separate-chaining hash table." *Id*. at 18.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Van Wyk discloses a record search means utilizing a search key to access the linked list. Van Wyk also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17. Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*. Moreover, Van Wyk discloses that "balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree . . . by a separate-chaining hash table." *Id*. at 18.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Van Wyk discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Van Wyk also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17. Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*.

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) (“Van Wyk”) alone and in combination**
		Moreover, Van Wyk discloses that “balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree . . . by a separate-chaining hash table.” *Id*. at 18. Moreover, Van Wyk discloses “a strategy of lazy deletion. That is, we keep the chains in each hash bucket sorted by expiration time; when inserting an element, we delete any elements in its hash bucket whose expiration times precede its starting time.” *Id*. “[U]nlike other applications of hashing in which one seeks to minimize the number of collisions not caused by successful searches, in hashing with lazy deletion we hope that enough collisions happen so the table is kept mostly free of expired items.” *Id*.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Van Wyk discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Van Wyk also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Van Wyk discloses “a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired.” *Id*. at 17. Moreover, Van Wyk discloses that “[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id* Moreover, Van Wyk discloses that "balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree . . . by a separate-chaining hash table." *Id*. at 18. Moreover, Van Wyk discloses "a strategy of lazy deletion. That is, we keep the chains in each hash bucket sorted by expiration time; when inserting an element, we delete any elements in its hash bucket whose expiration times precede its starting time." *Id*. "[U]nlike other applications of hashing in which one seeks to minimize the number of collisions not caused by successful searches, in hashing with lazy deletion we hope that enough collisions happen so the table is kept mostly free of expired items." *Id*. To the extent Bedrock argues that Van Wyk does not anticipate this element, based on general knowledge and/or in combination with Knuth and/or Kruse, one of ordinary skill in the art would recognize that the Van Wyk applies to insertions, retrievals, and/or deletions of automatically expired records because these are all basic functions that can be performed in the same manner on a hash table or a liked list. See, e.g., "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer

EXHIBIT C-1

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148."
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Van Wyk combined with Morrison, Mathieu, Kenyon-Mathieu or Aldous discloses the information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Morrison, Mathieu, Kenyon-Mathieu, and Aldous each teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Morrison at 1156-1161, Mathieu at 12-22, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Van Wyk to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, Kenyon-Mathieu, and Aldous with Van Wyk because Van Wyk raised the calculation of such bounds as an open question for further research. See Van Wyk at 29. Moreover, Morrison, Mathieu, and Kenyon-Mathieu all refer to and build off of the teachings of Van Wyk. See Kenyon-Mathieu at 473, 475, Mathieu at 1-4, Morrison at 1155,1158, Aldous at 3. Van Wyk combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, > each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. > After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) (“Van Wyk”) alone and in combination**
		entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56.

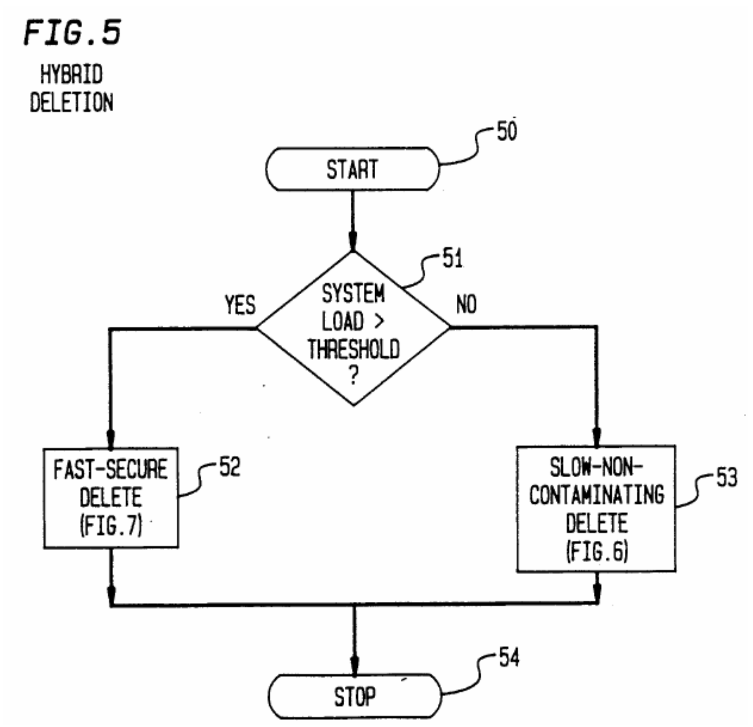
EXHIBIT C-1

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		As both Van Wyk and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Van Wyk with the means for dynamically determining maximum number for the record search means to

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		remove in the accessed linked list of records disclosed by Thatte. For example, Thatte, discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Van Wyk with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Van Wyk with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Van Wyk can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Van Wyk with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		with Thatte. Alternatively, it would also be obvious to combine Van Wyk with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

EXHIBIT C-1

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Van Wyk and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		patent's deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Van Wyk with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Van Wyk and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Van Wyk to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		ordinary skill in the art would have been motivated to combine the system disclosed in Van Wyk with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Van Wyk can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Van Wyk is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records	To the extent the preamble is a limitation, Van Wyk discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Van Wyk also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Van Wyk discloses "a version of the dynamic dictionary problem

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
	automatically expiring, the method comprising the steps of:	in which stored items have expiration times and can be removed from the dictionary once they have expired." Van Wyk at 17. Moreover, Van Wyk discloses "[h]ashing with lazy deletion is a simple algorithm for dynamic dictionaries whose occupants expire spontaneously. It is also compatible with the need to accommodate explicit user requests to delete elements. Even if the expiration times of active occupants change, one can use hashing with lazy deletion: if expiration times only increase, they can be changed by moving items further down hash chains; if they may also decrease, they may need to move back in their hash chain or even be deleted immediately. (An alternative would be to forget about keeping the hash chains in expiration-time order; this would make updating expiration times simpler, and would not change the asymptotic time complexity.)" *Id*. at 29.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Van Wyk discloses accessing a linked list of records. Van Wyk also discloses accessing a linked list of records having same hash address. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17. Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*.

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		Moreover, Van Wyk discloses that "balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree . . . by a separate-chaining hash table." *Id*. at 18.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Van Wyk discloses identifying at least some of the automatically expired ones of the records. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17. Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Van Wyk discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the dictionary once they have expired." *Id*. at 17.

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		Moreover, Van Wyk discloses that "[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item's starting time can be deleted from the dictionary: they no longer represent valid search results." *Id*. Moreover, Van Wyk discloses that "balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree …by a separate-chaining hash table." *Id*. at 18. Moreover, Van Wyk discloses "a strategy of lazy deletion. That is, we keep the chains in each hash bucket sorted by expiration time; when inserting an element, we delete any elements in its hash bucket whose expiration times precede its starting time." *Id*. "[U]nlike other applications of hashing in which one seeks to minimize the number of collisions not caused by successful searches, in hashing with lazy deletion we hope that enough collisions happen so the table is kept mostly free of expired items." *Id*.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Van Wyk discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Van Wyk discloses "a version of the dynamic dictionary problem in which stored items have expiration times and can be removed from the

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) (“Van Wyk”) alone and in combination**
		dictionary once they have expired.” *Id*. at 17. Moreover, Van Wyk discloses that “[a] sequence of items arrives to be stored in a dynamic dictionary. Besides the key used when searching for items, each item includes two times, a *starting* time and an *expiration* time. Items arrive in order of their starting times. Each time an item arrives, any items in the dictionary whose expiration times precede the incoming item’s starting time can be deleted from the dictionary: they no longer represent valid search results.” *Id*. Moreover, Van Wyk discloses that “balanced trees only offer logarithmic time complexity; besides, they are complicated to implement and require several pointers per item. So we might decide to replace the search tree . . . by a separate-chaining hash table.” *Id*. at 18. Moreover, Van Wyk discloses “a strategy of lazy deletion. That is, we keep the chains in each hash bucket sorted by expiration time; when inserting an element, we delete any elements in its hash bucket whose expiration times precede its starting time.” *Id*. “[U]nlike other applications of hashing in which one seeks to minimize the number of collisions not caused by successful searches, in hashing with lazy deletion we hope that enough collisions happen so the table is kept mostly free of expired items.” *Id*. Moreover, Van Wyk and Vitter explain that “[t]he insertion of element x_i proceeds as follows:

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		1) Compute *h* = n(k_i) 2) Remove from the hash chain in bucket *h* any items x_j with $t_j < s_j$ [i.e. remove all expired elements] 3) Add x_i so the chain in bucket *h* remains sorted by termination time." *Id*. at 19. This method clearly contemplates the insertion occurring after the removal of expired elements.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Van Wyk combined with Morrison, Mathieu, Kenyon-Mathieu or Aldous discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Morrison, Mathieu, Kenyon-Mathieu, and Aldous each teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Morrison at 1156-1161, Mathieu at 12-22, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Van Wyk to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, Kenyon-Mathieu, and Aldous with Van Wyk because Van Wyk raised the calculation of such bounds as an open question for further research. See Van Wyk at 29. Moreover, Morrison, Mathieu, and Kenyon-Mathieu all refer to and build off of the teachings of Van Wyk. See Kenyon-Mathieu at

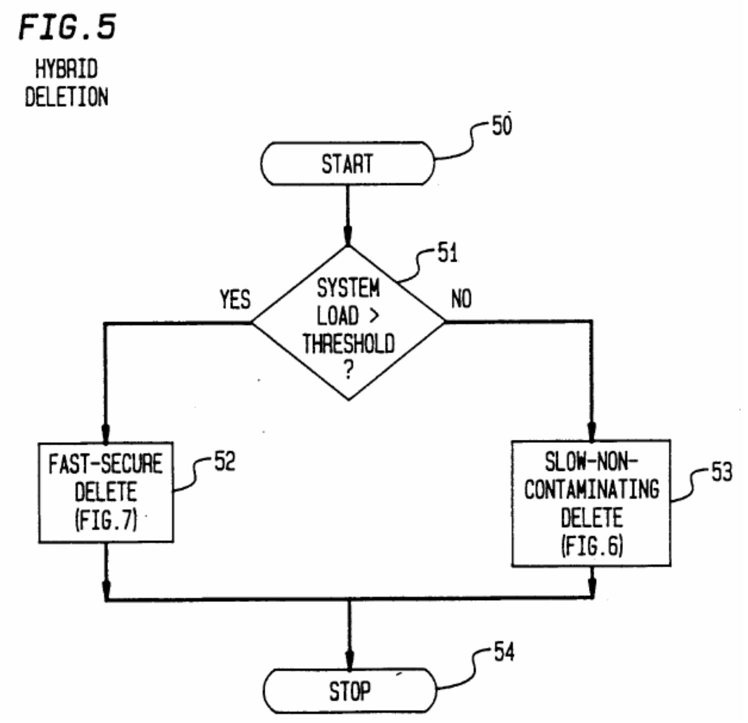
Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		473, 475, Mathieu at 1-4, Morrison at 1155,1158, Aldous at 3. Van Wyk combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Van Wyk and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps.

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Van Wyk with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Van Wyk with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Van Wyk with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Van Wyk can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Van Wyk with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) (“Van Wyk”) alone and in combination**
		records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk with Thatte. Alternatively, it would also be obvious to combine Van Wyk with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 (“The ’663 patent”). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as “deleted” and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		Id. at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Van Wyk and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Van Wyk with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector,* OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Van Wyk and the Opportunistic Garbage Collection Articles relate to

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination
		deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Van Wyk. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Van Wyk's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Van Wyk's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Van Wyk to dynamically determine the maximum number of expired records to remove in the accessed linked list of

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 ALGORITHMICA 17 (November, 1986) ("Van Wyk") alone and in combination**
		records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Van Wyk with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Van Wyk can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Van Wyk is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15.

EXHIBIT C-2

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Morrison discloses an information storage and retrieval system. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." Morrison, et al., *A Queuing Analysis of Hashing with Lazy Deletion, 16:6 SIAM J. Comput.*, 1155 (December 6, 1987). On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses an information storage and retrieval system. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Morrison discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Morrison also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		table." *Id.* Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"),

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. On information and belief, the source code for IDEAL also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Morrison discloses a record search means utilizing a search key to access the linked list. Morrison also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." *Id*

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses a record search means utilizing a search key to access the linked list. On information and belief, the source code for IDEAL also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Morrison discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Morrison also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." *Id.*

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155.Van Wyk discloses a method of hashing with lazy deletion using a

EXHIBIT C-2

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. On information and belief, the source code for IDEAL also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Morrison discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Morrison also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." *Id*

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. To the extent Bedrock argues that Morrison does not anticipate this element, based on general knowledge and/or in combination with Knuth and/or Kruse, one of ordinary skill in the art would recognize that the Morrison applies to insertions, retrievals, and/or deletions of automatically expired records because these are all basic functions that can be performed in the same manner on a hash table or a liked list. See, e.g., "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148." On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. On information and belief, the source code for IDEAL also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
2. The information storage	6. The information storage	Morrison discloses the information storage and retrieval system further

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Morrison teaches a method for calculating a probabilistic bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Morrison at 1156-1161. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Morrison does not disclose this element, Morrison combined with Mathieu, Kenyon-Mathieu or Aldous discloses this element. Mathieu, Kenyon-Mathieu, and Aldous also teach methods for calculating a probabilistic bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Mathieu at 12-22, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Morrison to remove. One of ordinary skill would have been motivated to combine the teachings of Mathieu, Kenyon-Mathieu, and Aldous with Morrison because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Morrison combined with Dirks, Thatte, the '663 patent and/or the

EXHIBIT C-2

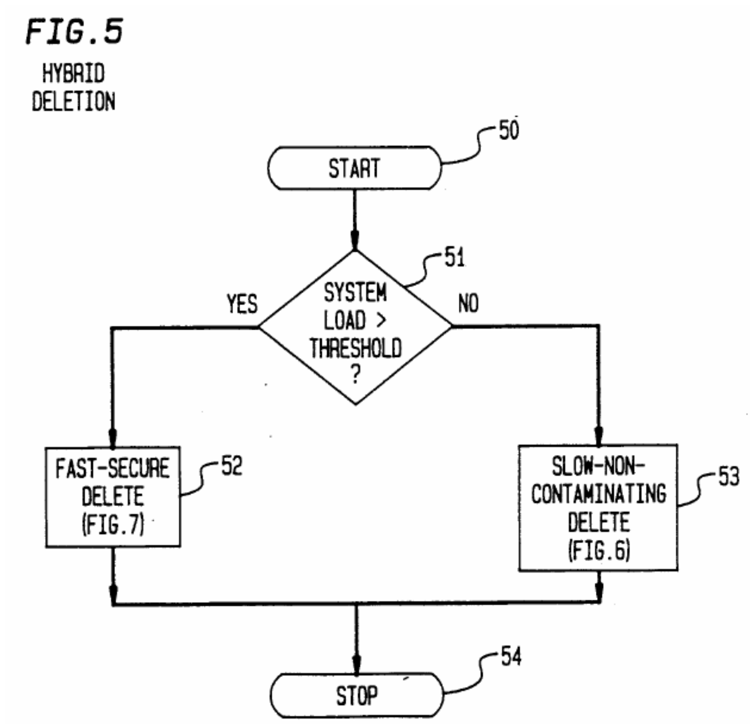
Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, > each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. > After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Id. at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Morrison and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Morrison. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Morrison nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Morrison and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Morrison with the means for dynamically determining maximum number for the record search means to

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Morrison and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Morrison with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Morrison with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Morrison can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Morrison with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		7:10-15. Thus, the '120 patent provides motivations to combine Morrison with Thatte. Alternatively, it would also be obvious to combine Morrison with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 (“The ’663 patent”). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as “deleted” and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Morrison and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Morrison. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		combining the '663 patent's deletion decision procedure with Morrison would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Morrison and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Morrison with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage*

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Collector at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Morrison and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Morrison. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Morrison would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Morrison and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Morrison to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Morrison with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example,

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		the removal of expired records described in Morrison can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Morrison in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Morrison. For example, both Linux 2.0.1 and Morrison describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		`rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		`rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		the maximum number of records that the function `rt_cache_add` can remove from a linked list. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, a person of ordinary skill in the art would have been motivated to combine IDEAL with the techniques taught by Linux 2.0.1, Dirks, Thatte, the '663 patent, the Opportunistic Garbage Collection Articles, Kenyon-Mathieu, and/or Aldous to disclose means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Morrison discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Morrison also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." Morrison at 1155.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) (“Morrison”) alone and in combination**
		On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the ’120 Patent. On information and belief, the source code for IDEAL discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. On information and belief, the source code for IDEAL also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Morrison discloses accessing a linked list of records. Morrison also discloses accessing a linked list of records having same hash address. For example, Morrison discloses “a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table.” *Id.* Moreover, Morrison discloses “a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses accessing a linked list of records. On information and belief, the source code for IDEAL also discloses accessing a linked list of records having same hash address. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Morrison discloses identifying at least some of the automatically expired ones of the records. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." *Id.* Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion,* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses identifying at least some of the automatically expired ones of the records. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) (“Morrison”) alone and in combination**
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Morrison discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Morrison discloses “a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table.” *Id.* Moreover, Morrison discloses “a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item’s expiration time.” *Id.* “Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is ‘lazy’ because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation” *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion,* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Morrison discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Morrison discloses "a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table." *Id.* Moreover, Morrison discloses "a sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time." *Id.* "Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is 'lazy' because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation" *Id.* Moreover, it is inherent to the Morrison disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Morrison with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*,

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Morrison describes and incorporates Van Wyk by reference. Morrison at 1155. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. Moreover, Van Wyk and Vitter explain that "[t]he insertion of element x_i proceeds as follows: 1) Compute h = n(k_i) 2) Remove from the hash chain in bucket h any items x_j with $t_j < s_j$ [i.e. remove all expired elements] 3) Add x_i so the chain in bucket h remains sorted by termination time." Van Wyk at 19. This method clearly contemplates the insertion occurring after the removal of expired elements. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, the source code for IDEAL discloses inserting, retrieving or deleting one of the records from the system following the step of removing. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Morrison discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Morrison teaches a method for calculating a probabilistic bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Morrison at 1156-1161. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Morrison does not disclose this element, Morrison combined with Mathieu, Kenyon-Mathieu or Aldous discloses this element. Mathieu, Kenyon-Mathieu, and Aldous also teach methods for calculating a probabilistic bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Mathieu at 12-22, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Morrison to remove. One of ordinary skill would have been motivated to combine the teachings of Mathieu, Kenyon-Mathieu, and Aldous with Morrison because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Morrison combined with Dirks, Thatte, the '663 patent and/or the

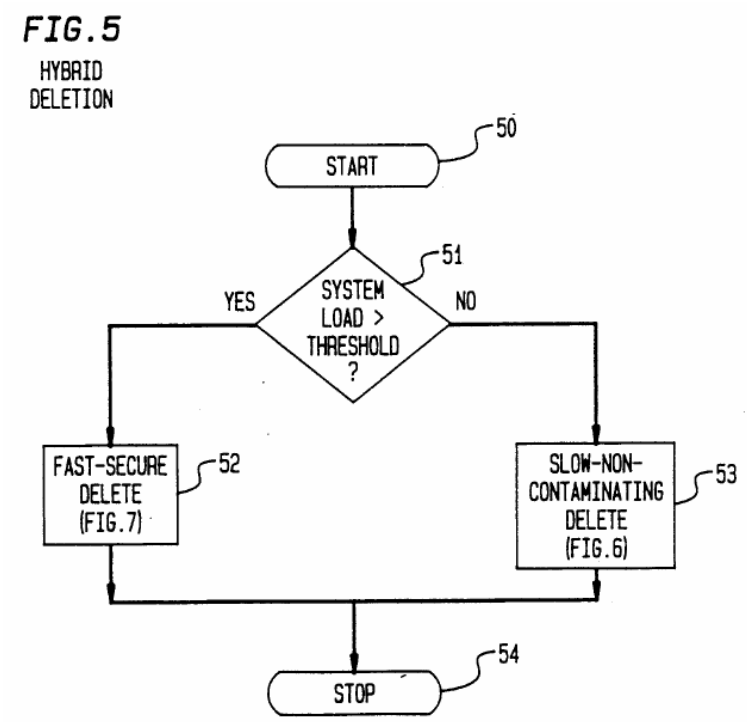
Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Morrison and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Morrison. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Morrison nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Morrison and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Morrison with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example,

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Morrison and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Morrison with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Morrison with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Morrison can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Morrison with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Morrison with

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Thatte. Alternatively, it would also be obvious to combine Morrison with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in the chart of the ‘663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) (“Morrison”) alone and in combination**
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5 HYBRID DELETION  50 START 51 YES SYSTEM LOAD > THRESHOLD ? NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Morrison and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Morrison. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination
		combining the '663 patent's deletion decision procedure with Morrison would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Morrison and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Morrison with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage*

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		Collector at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id*. If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Morrison and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Morrison. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Morrison would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Morrison and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Morrison to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Morrison with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example,

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		the removal of expired records described in Morrison can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Morrison in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Morrison. For example, both Linux 2.0.1 and Morrison describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		`rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function rt_garbage_collect_1 can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		**John A. Morrison et al., *A Queuing Analysis of Hashing with Lazy Deletion*, Society for Industrial and Applied Mathematics, Vol. 16, No. 6, 1155-1164 (December 1987) ("Morrison") alone and in combination**
		the maximum number of records that the function `rt_cache_add` can remove from a linked list. On information and belief, the techniques disclosed in Morrison were implemented and distributed in a Unix utility called IDEAL, prior to the filing date of the '120 Patent. On information and belief, a person of ordinary skill in the art would have been motivated to combine IDEAL with the techniques taught by Linux 2.0.1, Dirks, Thatte, the '663 patent, the Opportunistic Garbage Collection Articles, Kenyon-Mathieu, and/or Aldous to disclose dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Defendants reserve the right to supplement these contentions once a complete version of the source code for IDEAL is produced.

EXHIBIT C-3

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Mathieu discloses an information storage and retrieval system. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 Rapports de Recherche 1, 2 (June 1988).
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Mathieu discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Mathieu also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." *Id*. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		"In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1b] a record search means	[5b] a record search means	Mathieu discloses a record search means utilizing a search key to access the

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
utilizing a search key to access the linked list,	utilizing a search key to access a linked list of records having the same hash address,	linked list. Mathieu also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"),

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Mathieu discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Mathieu also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. "In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time,	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the	Mathieu discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Mathieu also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
removing at least some of the expired ones of the records in the linked list.	system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. "In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated

EXHIBIT C-3

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion,* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. To the extent Bedrock argues that Mathieu does not anticipate this element, based on general knowledge and/or in combination with Knuth and/or Kruse, one of ordinary skill in the art would recognize that the Mathieu applies to insertions, retrievals, and/or deletions of automatically expired records because these are all basic functions that can be performed in the same manner on a hash table or a liked list. See, e.g., "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148."
2. The information storage and retrieval system according to claim 1 further including means for	6. The information storage and retrieval system according to claim 5 further including means for	Mathieu discloses the information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Mathieu teaches a method for calculating a bound on the space complexity of the

EXHIBIT C-3

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Mathieu at 12-22. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Mathieu does not disclose this element, Mathieu combined with Morrison, Kenyon-Mathieu or Aldous discloses this element. Morrison, Kenyon-Mathieu, and Aldous also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Mathieu to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Kenyon-Mathieu, and Aldous with Mathieu because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Mathieu combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of

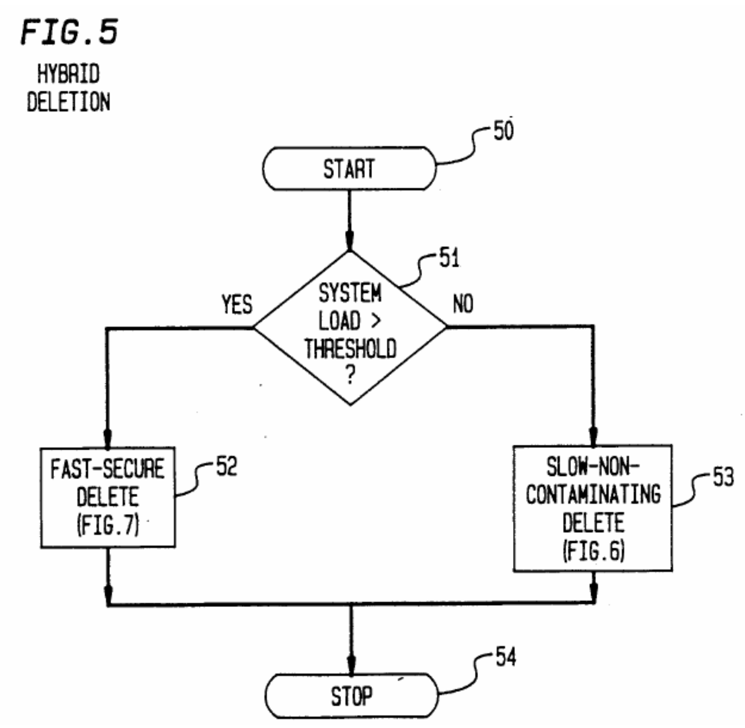
Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) (“Mathieu”) alone and in combination**
		x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		As both Mathieu and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Mathieu with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example,

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Mathieu with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Mathieu with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Mathieu with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Mathieu with Thatte.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		Alternatively, it would also be obvious to combine Mathieu with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example as summarized in the ’663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		This hybrid deletion is shown in Figure 5. FIG.5 HYBRID DELETION  50 START 51 YES SYSTEM LOAD > THRESHOLD ? NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64,

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) (“Mathieu”) alone and in combination**
		Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Mathieu and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the ’663 patent’s deletion decision procedure with Mathieu’s technique of “lazy deletion” would be nothing more than the predictable use of prior art elements

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Mathieu with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Mathieu and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mathieu to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mathieu with the fundamental concept of dynamically determining

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Mathieu is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Mathieu in combination with Morrison, Kenyon-Mathieu, Aldous, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Mathieu. For example, both Linux 2.0.1 and Mathieu describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		integer variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1359. When the function rt_cache_add removes an expired record, the function rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold RT_CACHE_SIZE_MAX. If the number of records in the hash table exceeds

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero.

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Mathieu discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Mathieu also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. To the extent the preamble is a limitation, Mathieu discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Mathieu discloses accessing a linked list of records. Mathieu also discloses accessing a linked list of records having same hash address. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time t the data structure stores the items that are 'living' at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The ith item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions,

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Mathieu discloses identifying at least some of the automatically expired ones of the records. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." Mathieu, *supra* at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		*i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. "In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Mathieu discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Mathieu discloses "a non-Markovian process (data structure) for processing plane-sweep information in computational geometry, called 'hashing with lazy deletion' (HwLD)." *Id*. at 1. Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. "In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
	[7d] inserting, retrieving or deleting one of the records from the system following	Mathieu discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
	the step of removing.	Moreover, Mathieu discloses that "[p]lane-sweep algorithms process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval [s_i, t_i] in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure must be able to support the dynamic operation of searching the living items based on key value." *Id*. at 2-3. "In HwLD, items are stored in a hash table of *H* buckets, based on the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing some dead items." *Id*. at 3. Moreover, it is inherent to the Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing*

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		with Lazy Deletion, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Mathieu describes and incorporates Van Wyk by reference. Mathieu at 2. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. Moreover, Van Wyk and Vitter explain that "[t]he insertion of element x_i proceeds as follows: 1) Compute $h = n(k_i)$ 2) Remove from the hash chain in bucket h any items x_j with $t_j < s_j$ [i.e. remove all expired elements] 3) Add x_i so the chain in bucket h remains sorted by termination time." Van Wyk at 19. This method clearly contemplates the insertion occurring after the removal of expired elements.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Mathieu discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Mathieu teaches a method for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Mathieu at 12-22. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists.

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		To the extent Mathieu does not disclose this element, Mathieu combined with Morrison, Kenyon-Mathieu or Aldous discloses this element. Morrison, Kenyon-Mathieu, and Aldous also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Kenyon-Mathieu at 475-486, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Mathieu to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Kenyon-Mathieu, and Aldous with Mathieu because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Mathieu combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety.

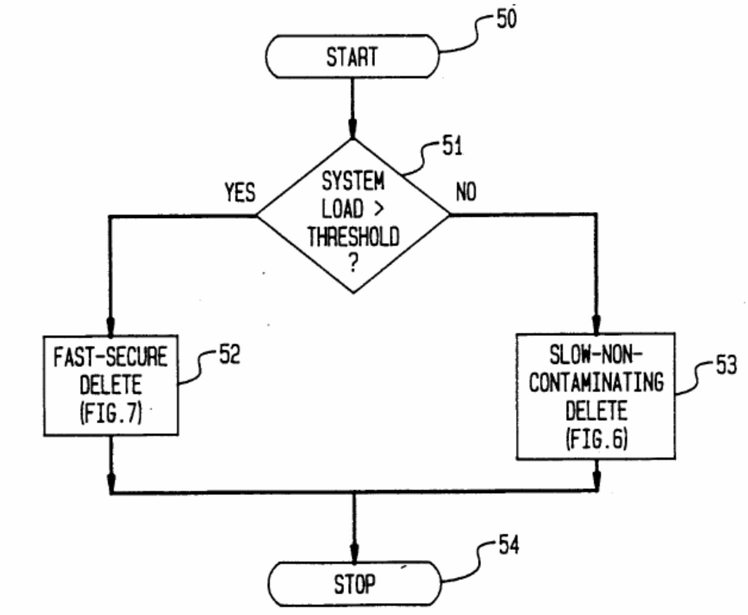
Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter,** ***Maximum Queue Size and Hashing with Lazy Deletion*****, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) (“Mathieu”) alone and in combination**
		$$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: “Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.” *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Mathieu and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Mathieu with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		way. Additionally, one of ordinary skill in the art would recognize that the result of combining Mathieu with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Mathieu with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Mathieu with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Mathieu with Thatte. Alternatively, it would also be obvious to combine Mathieu with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) (“Mathieu”) alone and in combination**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Mathieu and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the ’663 patent’s deletion decision procedure with Mathieu’s technique of “lazy deletion” would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) (“Mathieu”) alone and in combination**
		By way of further example, one of ordinary skill in the art would have combined the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load as taught by the ’663 patent and with Mathieu’s system and method of “lazy deletion” and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Mathieu with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Mathieu and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Mathieu to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Mathieu with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		of expired records described in Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Mathieu is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Mathieu in combination with Morrison, Kenyon-Mathieu, Aldous, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Mathieu. For example, both Linux 2.0.1 and Mathieu describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		`rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination**
		2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		`rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Mathieu and Jeffrey Scott Vitter, *Maximum Queue Size and Hashing with Lazy Deletion*, 851 RAPPORTS DE RECHERCHE 1 (June 1988) ("Mathieu") alone and in combination
		the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-4

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Kenyon-Mathieu discloses an information storage and retrieval system. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989).
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Kenyon-Mathieu discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Kenyon-Mathieu also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." *Id*. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval [s_i, t_i] in the unit interval, containing a unique key k_i of supplementary information. The *i*th

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. "In HwLD, items are stored in a hash table of *H* buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Kenyon-Mathieu discloses a record search means utilizing a search key to access the linked list. Kenyon-Mathieu also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time t the data structure stores the items that are 'living' at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The ith item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Kenyon-Mathieu discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Kenyon-Mathieu also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval [s_i, t_i] in the unit

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. "In HwLD, items are stored in a hash table of *H* buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Kenyon-Mathieu discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Kenyon-Mathieu also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time t the data structure stores the items that are 'living' at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The ith item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id.* at 474. "In HwLD, items are stored in a hash table of H buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number H of buckets is

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. To the extent Bedrock argues that Kenyon-Mathieu does not anticipate this element, based on general knowledge and/or in combination with Knuth and/or Kruse, one of ordinary skill in the art would recognize that the Kenyon-Mathieu applies to insertions, retrievals, and/or deletions of automatically

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		expired records because these are all basic functions that can be performed in the same manner on a hash table or a liked list. See, e.g., "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148."
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Kenyon-Mathieu discloses the information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Kenyon-Mathieu teaches a method for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Kenyon-Mathieu at 475-86. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Kenyon-Mathieu does not disclose this element, Kenyon-Mathieu combined with Morrison, Mathieu or Aldous discloses this element. Morrison, Mathieu, and Aldous also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Mathieu at 12-22, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		the record search means disclosed in Kenyon-Mathieu to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, and Aldous with Kenyon-Mathieu because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Kenyon-Mathieu combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of

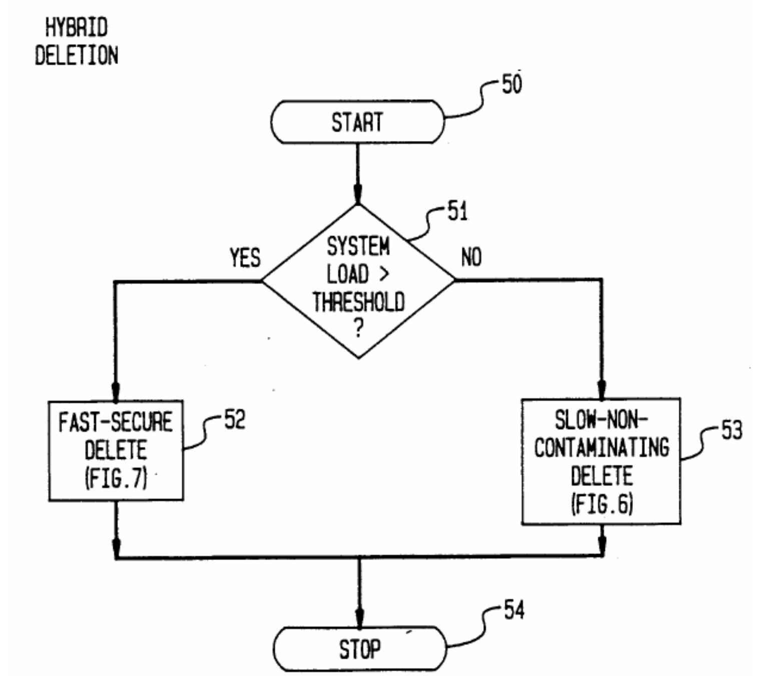
Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Kenyon-Mathieu and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Kenyon-Mathieu's

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Kenyon-Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Kenyon-Mathieu with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Kenyon-Mathieu with Thatte would be nothing more than the predictable use of prior art elements according to their established

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Kenyon-Mathieu with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Kenyon-Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Kenyon-Mathieu with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Kenyon-Mathieu with Thatte. Alternatively, it would also be obvious to combine Kenyon-Mathieu with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT C-4

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Kenyon-Mathieu and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Kenyon-Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Kenyon-Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Kenyon-Mathieu with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage*

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		Collector at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Kenyon-Mathieu and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash

EXHIBIT C-4

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		table implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Kenyon-Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Kenyon-Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Kenyon-Mathieu to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Kenyon-Mathieu with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Kenyon-Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Kenyon-Mathieu is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Kenyon-Mathieu in combination with Morrison, Mathieu, Aldous, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to

EXHIBIT C-4

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		combine Linux 2.0.1 with Kenyon-Mathieu. For example, both Linux 2.0.1 and Kenyon-Mathieu describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Kenyon-Mathieu discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Kenyon-Mathieu also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion,* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Kenyon-Mathieu discloses accessing a linked list of records. Kenyon-Mathieu also discloses accessing a linked list of records having same hash address. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval [s_i, t_i] in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3b] identifying at least some of the automatically	[7b] identifying at least some of the automatically	Kenyon-Mathieu discloses identifying at least some of the automatically expired ones of the records.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
expired ones of the records, and	expired ones of the records,	For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time t the data structure stores the items that are 'living' at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The ith item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. "In HwLD, items are stored in a hash table of H buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number H of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Kenyon-Mathieu discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." *Id*. at 473.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time *t* the data structure stores the items that are 'living' at time *t*. Let us think of the *i*th item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The *i*th item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time *t* when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id*. at 474. "In HwLD, items are stored in a hash table of *H* buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number *H* of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy*

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		Deletion, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Kenyon-Mathieu discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Kenyon-Mathieu discloses "a non-Markovian process called *hashing with lazy deletion* (HwLD), which corresponds to an efficient way of processing sweepline information in computational geometry." Kenyon-Mathieu, *supra* at 473. Moreover, Kenyon-Mathieu discloses "[d]ata structures process a sequence of items over time; at time t the data structure stores the items that are 'living' at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information. The ith item is 'born' at time s_i, 'dies' at time t_i, and is 'living' at time t when $t \in [s_i, t_i]$. The data structure also handles dynamic queries over time." *Id.* at 474. "In HwLD, items are stored in a hash table of H buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the 'lazy deletion' strategy deletes a dead item only when a later insertion accesses the same bucket. The number H of buckets is

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing 'vigilant-deletion,' at a cost of storing dead items." *Id*. Moreover, it is inherent to the Kenyon-Mathieu disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Kenyon-Mathieu with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986) ("Van Wyk"), because Kenyon-Mathieu describes and incorporates Van Wyk by reference. Kenyon-Mathieu at 473. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. Moreover, Van Wyk and Vitter explain that "[t]he insertion of element x_i proceeds as follows: 1) Compute h = n(k_i)

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		2) Remove from the hash chain in bucket *h* any items x_j with $t_j < s_j$ [i.e. remove all expired elements] 3) Add x_i so the chain in bucket *h* remains sorted by termination time." Van Wyk at 19. This method clearly contemplates the insertion occurring after the removal of expired elements.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Kenyon-Mathieu discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Kenyon-Mathieu teaches a method for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Kenyon-Mathieu at 475-86. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Kenyon-Mathieu does not disclose this element, Kenyon-Mathieu combined with Morrison, Mathieu or Aldous discloses this element. Morrison, Mathieu, and Aldous also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Mathieu at 12-22, Aldous 17-21. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		the record search means disclosed in Kenyon-Mathieu to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, and Aldous with Kenyon-Mathieu because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Kenyon-Mathieu combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be

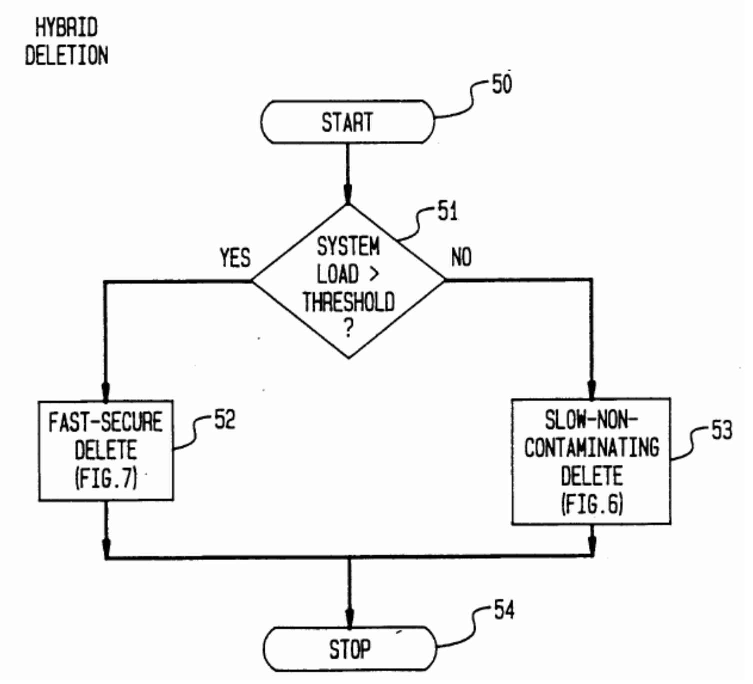
Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Kenyon-Mathieu and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Kenyon-Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Kenyon-Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Kenyon-Mathieu with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Kenyon-Mathieu with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Kenyon-Mathieu with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Kenyon-Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Kenyon-Mathieu with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Kenyon-Mathieu with Thatte. Alternatively, it would also be obvious to combine Kenyon-Mathieu with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Kenyon-Mathieu and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Kenyon-Mathieu's technique of "lazy deletion" would be nothing more than the predictable use of prior art

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Kenyon-Mathieu's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Kenyon-Mathieu with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage*

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		Collector at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Kenyon-Mathieu and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		table implementations such as Kenyon-Mathieu. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with Kenyon-Mathieu’s technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles’ dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Kenyon-Mathieu’s system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Kenyon-Mathieu to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Kenyon-Mathieu with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Kenyon-Mathieu can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Kenyon-Mathieu is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Kenyon-Mathieu in combination with Morrison, Mathieu, Aldous, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		combine Linux 2.0.1 with Kenyon-Mathieu. For example, both Linux 2.0.1 and Kenyon-Mathieu describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a

Asserted Claims From U.S. Pat. No. 5,893,120		Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination
		dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero.

Asserted Claims From U.S. Pat. No. 5,893,120		**Claire M. Kenyon-Mathieu and Jeffrey Scott Vitter, *General Methods for the Analysis of the Maximum Size of Dynamic Data Structures*, in Proc. 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, 473 (1989) ("Kenyon-Mathieu") alone and in combination**
		In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Aldous discloses an information storage and retrieval system. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." David Aldous et al., *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) at 713.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Aldous discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Aldous also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." *Id*. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, (Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986)), by Van Wyk and Vitter because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1b] a record search means	[5b] a record search means	Aldous discloses a record search means utilizing a search key to access the

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
utilizing a search key to access the linked list,	utilizing a search key to access a linked list of records having the same hash address,	linked list. Aldous also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*,

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Aldous discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Aldous also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713 Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[1d] means, utilizing the	[5d] mea[n]s, utilizing the	Aldous discloses means, utilizing the record search means, for accessing the

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Aldous also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715. Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713. Moreover, it is inherent to the Aldous disclosure that the hash table described

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion,* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. To the extent Bedrock argues that Aldous does not anticipate this element, based on general knowledge and/or in combination with Knuth and/or Kruse, one of ordinary skill in the art would recognize that the Aldous applies to insertions, retrievals, and/or deletions of automatically expired records because these are all basic functions that can be performed in the same manner on a hash table or a liked list. See, e.g., "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549; "Data Structures and Program Design", R.L. Kruse, Prentice-Hall, Inc. 1984, pp. 104-148."

EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Aldous discloses the information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Aldous teaches a method for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Aldous at 17-21. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Aldous does not disclose this element, Aldous combined with Morrison, Mathieu or Kenyon-Mathieu discloses this element. Morrison, Mathieu, and Kenyon-Mathieu also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Mathieu at 12-22, Kenyon-Mathieu 475-86. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Aldous to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, and Kenyon-Mathieu with Aldous because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Aldous combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20).

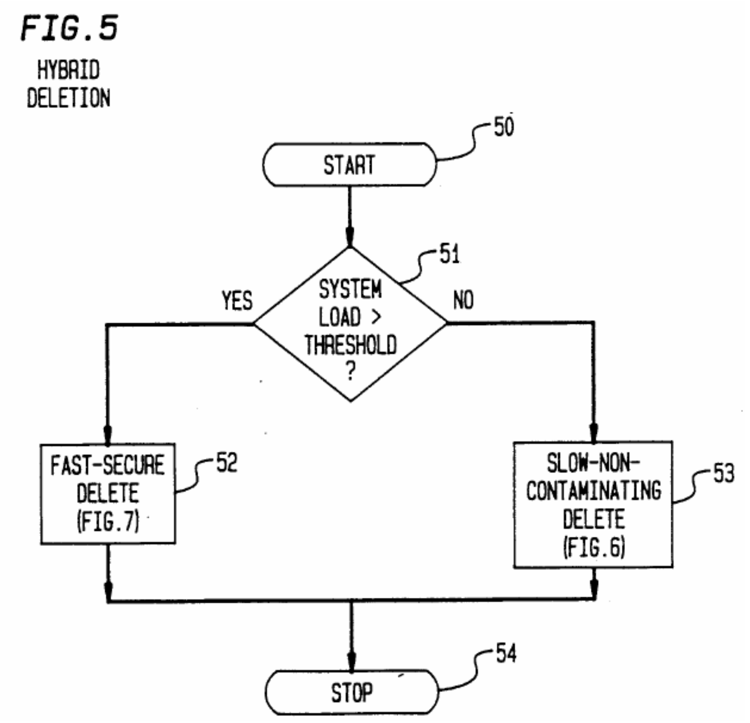
EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		Id. at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Aldous and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Aldous. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		understand how to, combine the system disclosed in Aldous with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Aldous with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Aldous with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Aldous can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Aldous with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Aldous with Thatte. Alternatively, it would also be obvious to combine Aldous with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the ’663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The ’663 patent"). > In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. > This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5.  FIG.5 HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Aldous and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Aldous. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		patent's deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Aldous with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally,* Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage*

EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		Collector at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Aldous and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Aldous. Moreover, one of ordinary skill in the art

EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Aldous to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		Aldous with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Aldous can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Aldous is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Aldous in combination with Morrison, Mathieu, Kenyon-Mathieu, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Aldous. For example, both Linux 2.0.1 and Aldous describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		`RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function rt_garbage_collect_1 can remove records whose reference counts are zero and records whose reference counts are greater than zero.

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Aldous discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Aldous also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Aldous discloses accessing a linked list of records. Aldous also discloses accessing a linked list of records having same hash address. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions,

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Aldous discloses identifying at least some of the automatically expired ones of the records. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items."Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H*

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket."[1] Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Aldous discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." *Id*. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*.

[1] *Id*. at 715.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		"An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715. Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion* Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Aldous discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Aldous discloses "a dynamic data structure management technique called *hashing with lazy deletion* (HwLD) . . . A table managed under HwLD is built by a sequence of insertions and deletions of items." Aldous, *supra* at 713. "An item arrives at a hashing table and needs to be stored for some period (the item's *lifetime*) . . . We always assume that the assignment of items to the *H* buckets of the hashing table is uniform: That is, each item has probability 1/*H* to select each bucket, independent for different items and independent of the arrival and lifetimes." *Id*. "An arriving item selects one out of the *H* buckets at random (with uniform probability) and joins the items assigned to this bucket." *Id*. at 715. Moreover, Aldous discloses that "[t]he principle HwLD is very simple: An item in a bucket is not deleted as soon as possible (i.e., when its lifetime expires). Instead, the item is removed at the first arrival to the item's bucket after the item's expiration time. The point is that algorithms that delete items as soon as possible may have unacceptably high overhead, even though they require less storage space for the items themselves." *Id*. at 713. Moreover, it is inherent to the Aldous disclosure that the hash table described herein is a separate-chaining hash table to resolve unavoidable key collisions, where each slot of the bucket array is a pointer to a linked list that contains the

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		key-value pairs that hash to the same location. Other data structures, such as balanced trees, only offer logarithmic time complexity and can be complicated to implement and require several pointers per item. With a suitable number of buckets, the separate-chaining hash table solution offers constant expected search times and makes deletion easy. To the extent is it not inherent, it would be obvious to combine Aldous with the prior art disclosed in, *The Complexity of Hashing with Lazy Deletion*, Christopher J. Van Wyk and Jeffrey Scott Vitter, *The Complexity of Hashing with Lazy Deletion*, 1 Algorithmica 17, 18 (November, 1986), because Aldous describes and incorporates Van Wyk by reference. Aldous at 713. Van Wyk discloses a method of hashing with lazy deletion using a separate-chaining hash table as a more efficient solution to collisions than the available alternatives, such as balanced trees. *See* Exhibit C-1 which is herein incorporated by reference. Moreover, Van Wyk and Vitter explain that "[t]he insertion of element x_i proceeds as follows: 1) Compute h = n(k_i) 2) Remove from the hash chain in bucket h any items x_j with $t_j < s_j$ [i.e. remove all expired elements] 3) Add x_i so the chain in bucket h remains sorted by termination time." Van Wyk at 19. This method clearly contemplates the insertion occurring after the removal of expired elements.
4. The method according to claim 3 further including	8. The method according to claim 7 further including	Aldous discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Aldous teaches a

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	method for calculating a bound on the space complexity of the hashing with lazy deletion algorithm, i.e., the number of expired elements that would remain in the linked lists forming the external chains of the hash table. See, e.g., Aldous at 17-21. That bound calculation is a means for dynamically determining the maximum number to remove, since the maximum number of elements to remove should never exceed the number of expired elements that would remain in the linked lists. To the extent Aldous does not disclose this element, Aldous combined with Morrison, Mathieu or Kenyon-Mathieu discloses this element. Morrison, Mathieu, and Kenyon-Mathieu also teach methods for calculating a bound on the space complexity of the hashing with lazy deletion algorithm. See, e.g., Morrison at 1156-61, Mathieu at 12-22, Kenyon-Mathieu 475-86. It would have been obvious to one of ordinary skill in the art that these methods of calculating bounds on the number of expired elements stored in the linked list could also be used to dynamically determine a maximum number for the record search means disclosed in Aldous to remove. One of ordinary skill would have been motivated to combine the teachings of Morrison, Mathieu, and Kenyon-Mathieu with Aldous because they all sought to analyze the complexity of the hashing with lazy deletion algorithm, and each of the papers refers to and builds upon the work of those papers preceding. See, e.g., Morrison at 1155, 1158, Mathieu at 1-4, Kenyon-Mathieu at 473-475, Aldous at 2-3. Aldous combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is

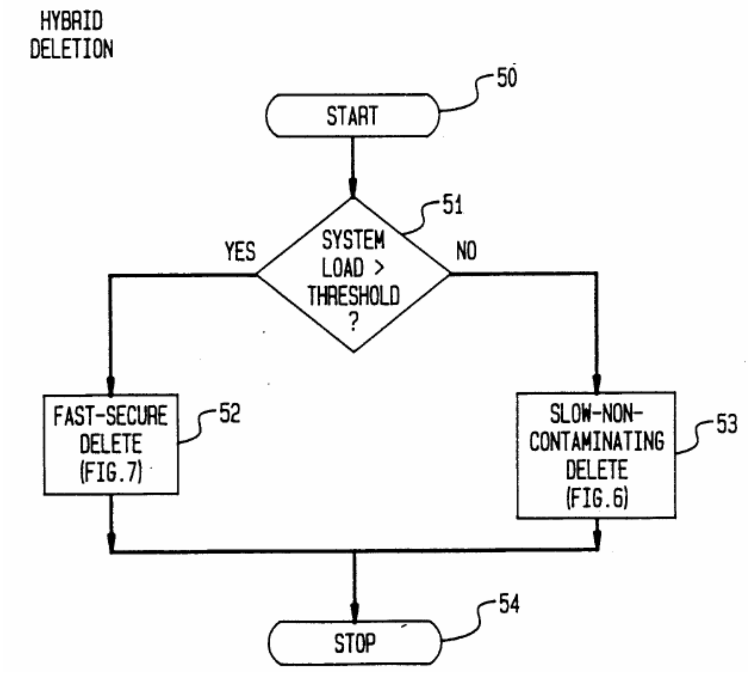
Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Aldous and Dirks relate to deletion of aged records upon the allocation

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Aldous. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One possible benefit, for example, is that the system and method of lazy deletion could perform a specified number of deletions on any given sweep, thus saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Aldous with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in Exhibit B-1, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Aldous with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Aldous with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Aldous can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Aldous with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Aldous with Thatte. Alternatively, it would also be obvious to combine Aldous with the '663

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		patent. Disclosure of these claim elements in the '663 patent is clearly shown in Exhibit B-13, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). > In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. > This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Aldous and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Aldous. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Aldous with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski,** ***Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*****, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Aldous and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Aldous. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Aldous's technique of "lazy deletion" would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Aldous's system and method of "lazy deletion" and would have seen the benefits of doing so. One such benefit, for example, is that the system and method of lazy deletion would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Aldous to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Aldous with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Aldous can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Aldous is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Aldous in combination with Morrison, Mathieu, Kenyon-Mathieu, Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Aldous. For example, both Linux 2.0.1 and Aldous describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux

Asserted Claims From U.S. Pat. No. 5,893,120		**David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination**
		2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect`

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect invokes a function rt_garbage_collect_1. *See* Linux 2.0.1, route.c at line 1293. The function rt_garbage_collect_1 loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function rt_garbage_collect_1 looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function rt_garbage_collect_1 determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function rt_garbage_collect_1 removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable expire and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value RT_CACHE_TIMEOUT. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can

EXHIBIT C-5

Asserted Claims From U.S. Pat. No. 5,893,120		David Aldous, Micha Hofri, & Wojciech Szpankowski, *Maximum Size of a Dynamic Data Structure: Hashing with Lazy Deletion Revisited*, 21 SIAM J. COMPUT. 713 (August 1992) ("Aldous") alone and in combination
		remove from a linked list.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Dietzfelbinger discloses an information storage and retrieval system. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990, at 1. "A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records	Dietzfelbinger discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Dietzfelbinger also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
	automatically expiring,	is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Kómlos, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] described a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log\log n)$…." *Id*. at 2-3. Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus,

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Furthermore, Dietzfelbinger discloses that “[d]eletions are performed by attaching a tag “deleted” to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag “deleted” from T_j.” *Id*. at 5.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Dietzfelbinger discloses a record search means utilizing a search key to access the linked list. Dietzfelbinger also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Dietzfelbinger discloses that “[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . .” *Id*. at 1. “A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*.” *Id*. at 2.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in n) and constant query time for all N and n, where n is the size of S." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into s sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function h chosen uniformly at random from H_s is used to partition the set S into s blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for n keys being stored in the dictionary in a scheme of

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is *O*(log *n* / log log *n*)…." *Id.* at 2-3. Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id.* Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed,	Dietzfelbinger discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Dietzfelbinger also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
accessed, and	and	For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log\log n)$..." *Id.* at 2-3. Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5. A call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. If the position in which *x* is to be placed is empty, then *x* is stored in the position. However, if the position is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. at 6.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked "deleted" then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked "deleted" into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		else (* T_j is too small *) $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$; **if** condition (**) is still satisfied **then** (* double capacity of T_j *) allocate new space, namely s_j cells, for new subtable T_j; go through old subtable T_j, put all elements not marked “deleted” into a list L_j, and mark all positions empty; append x to list L_j; $b_j \leftarrow$ length of L_j; **repeat** $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$ **until** h_j is injective on the elements of list L_j; **for** all y on list L_j store y in position $h_j(y)$ of T_j; **else** (* level-1-function h “bad” *) $RehashAll(x)$; *Id*. at Figure 1.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Dietzfelbinger discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Dietzfelbinger also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Dietzfelbinger discloses that “[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is

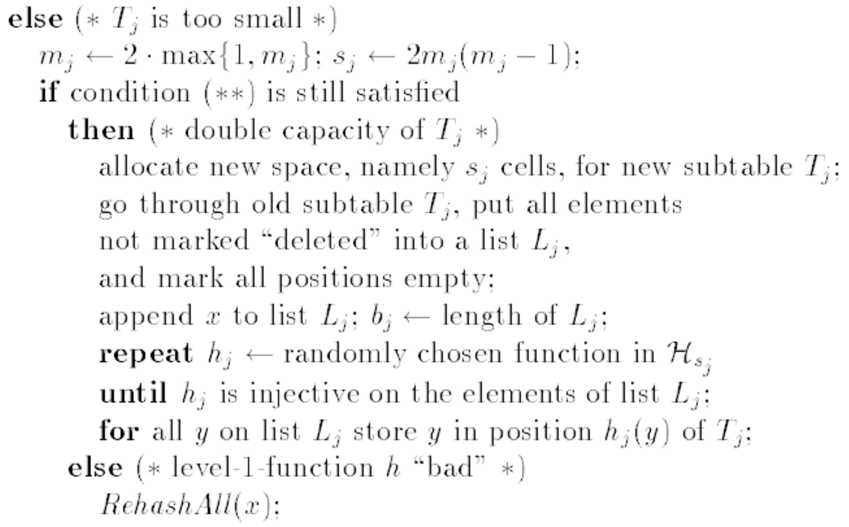
Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4.

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$..." *Id*. at 2-3.Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5. A call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. If the position in which *x* is to be placed is empty, then *x* is stored in the position. However, if the position is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. at 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked “deleted” then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked “deleted” into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		 **else** (* T_j is too small *) $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$; **if** condition (**) is still satisfied **then** (* double capacity of T_j *) allocate new space, namely s_j cells, for new subtable T_j; go through old subtable T_j, put all elements not marked "deleted" into a list L_j, and mark all positions empty; append x to list L_j; $b_j \leftarrow$ length of L_j; **repeat** $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$ **until** h_j is injective on the elements of list L_j; **for** all y on list L_j store y in position $h_j(y)$ of T_j; **else** (* level-1-function h "bad" *) $RehashAll(x)$; *Id*. at Figure 1.
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Dietzfelbinger discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, a call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. First, the procedure determines if *x* already exists in the sub table. If *x* does exist, then the only determination to make is whether *x* is marked for deletion. If *x* is marked for deletion, then the procedure removes this deletion tag. *Id*. at 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Second, if the sub table does not contain *x* ***and*** the position in which *x* is to be placed is found empty, then *x* is stored in the position. *Id*. Third, if the sub table does not contain *x* ***and*** the position in which *x* is to be placed is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. Therefore, the procedure dynamically determines whether or not to delete the records marked "deleted" if either of the following two conditions occur: (1) *x* already exists in the sub table or (2) the position in which *x* is to be placed is empty. Deletion takes place during a call to procedure *Insert(x)* only if the above two conditions are not met.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked “deleted” then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked “deleted” into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		else (∗ T_j is too small ∗) $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$; **if** condition (∗∗) is still satisfied **then** (∗ double capacity of T_j ∗) allocate new space, namely s_j cells, for new subtable T_j; go through old subtable T_j, put all elements not marked “deleted” into a list L_j, and mark all positions empty; append x to list L_j; $b_j \leftarrow$ length of L_j; **repeat** $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$ **until** h_j is injective on the elements of list L_j; **for** all y on list L_j store y in position $h_j(y)$ of T_j; **else** (∗ level-1-function h “bad” ∗) $RehashAll(x)$; *Id*. at Figure 1. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dietzfelbinger to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dietzfelbinger with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Dietzfelbinger is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Dietzfelbinger combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Dietzfelbinger and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		implementations such as Dietzfelbinger. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Dietzfelbinger nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Dietzfelbinger and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Dietzfelbinger with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Dietzfelbinger and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Dietzfelbinger with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Dietzfelbinger with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Dietzfelbinger with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Dietzfelbinger with Thatte. Alternatively, it would also be obvious to combine Dietzfelbinger with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). > In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. > This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

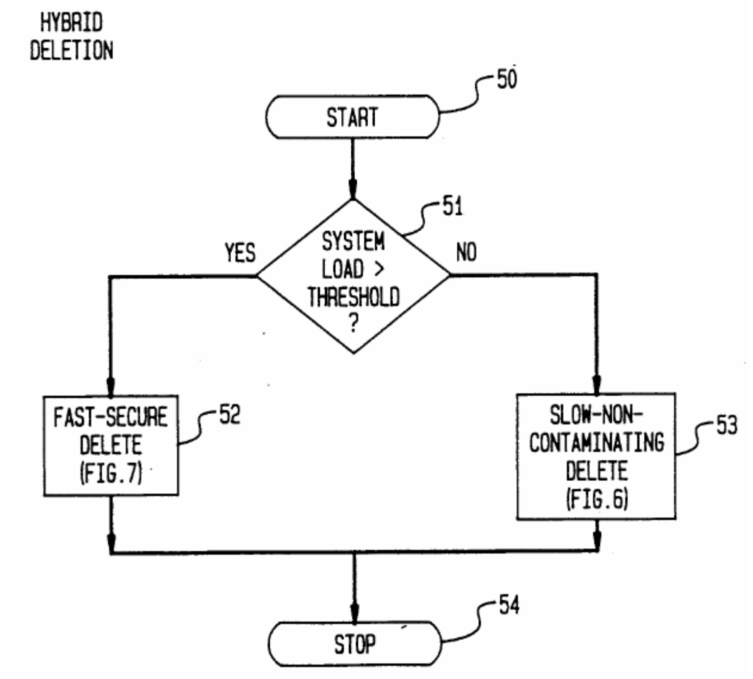
Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Dietzfelbinger and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Dietzfelbinger. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Dietzfelbinger would be nothing more than the predictable use of prior art elements according

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		to their established functions. By way of further example, one of ordinary skill in the art would have combined the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load as taught by the ’663 patent and with Dietzfelbinger and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Dietzfelbinger with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Dietzfelbinger and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Dietzfelbinger. Moreover, one of ordinary skill

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Dietzfelbinger would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Dietzfelbinger and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dietzfelbinger to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dietzfelbinger with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Dietzfelbinger in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Dietzfelbinger. For example, both Linux 2.0.1 and Dietzfelbinger describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).
		`RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record’s last use time plus the record’s expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record’s last use time plus the record’s expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record’s expiration factor is based on a variable `expire` and the record’s reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Dietzfelbinger discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Dietzfelbinger also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." Deitzfelbinger at 1. "A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*."

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Id. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is *O*(log *n* / log log *n*)..." *Id*. at 2-3.Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Dietzfelbinger discloses accessing a linked list of records. Dietzfelbinger also discloses accessing a linked list of records having same hash address. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe* *U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$..." *Id.* at 2-3. Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Dietzfelbinger discloses identifying at least some of the automatically expired ones of the records. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Dietzfelbinger discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1. "A *dictionary* over a *universe U* = {0, 1, . . . , *N* - 1} is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós,

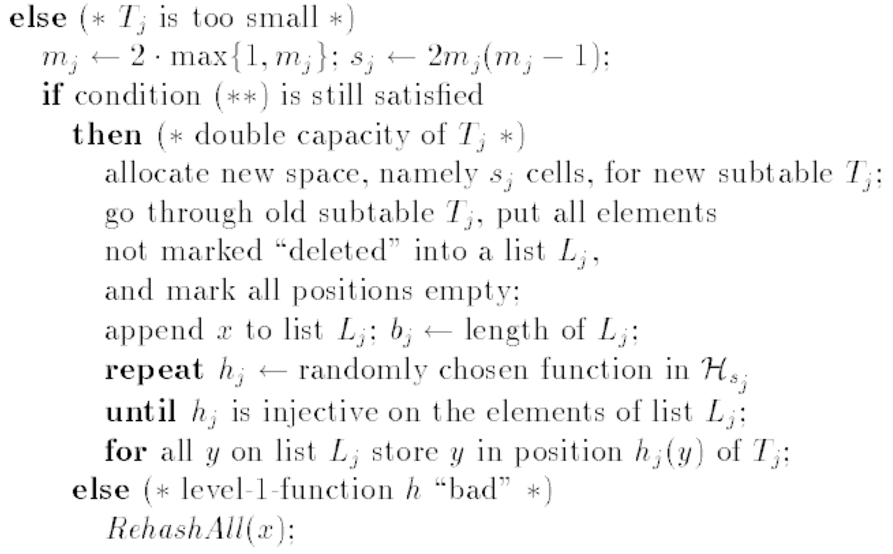
Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem, Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is *O*(log *n* / log log *n*)...." *Id*. at 2-3. Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way. Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5. A call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. If the position in which *x* is to be placed is empty, then *x* is stored in the position. However, if the position is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. at 6.

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		see pseudocode below

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked "deleted" then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked "deleted" into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

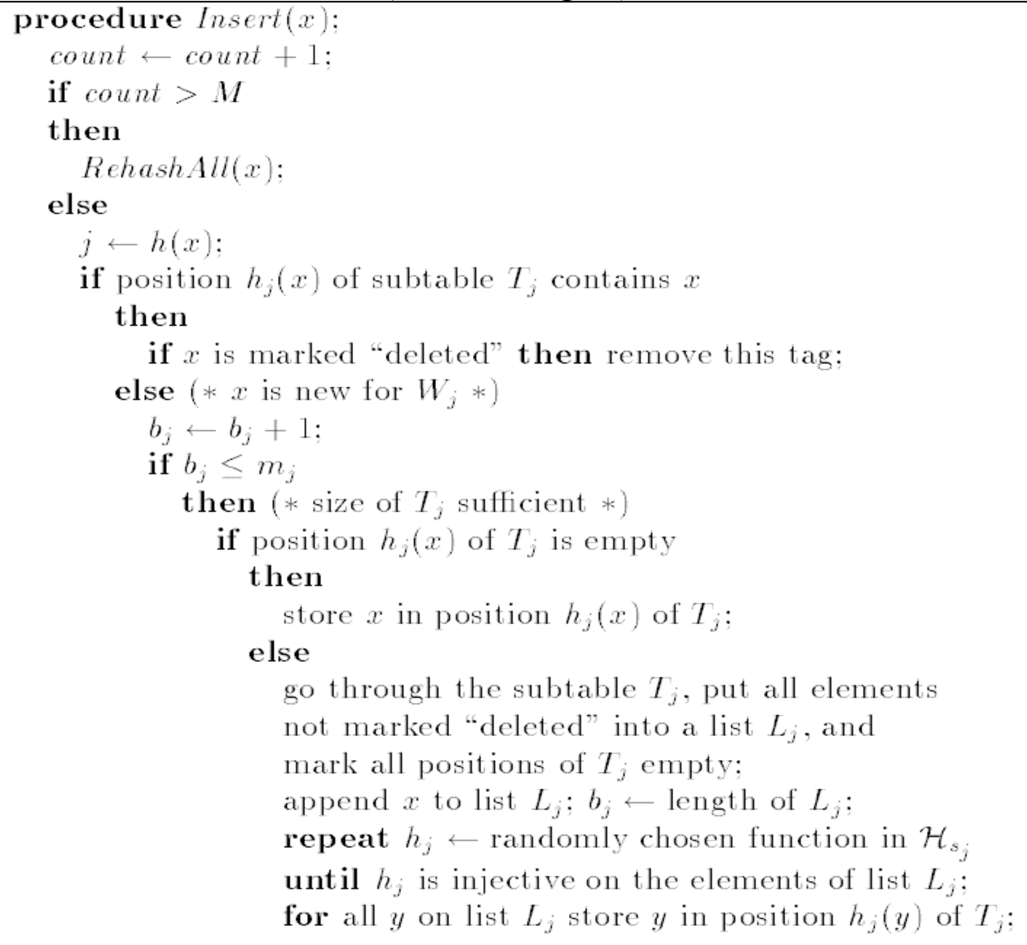
EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		 **else** (* T_j is too small *) $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$; **if** condition (**) is still satisfied **then** (* double capacity of T_j *) allocate new space, namely s_j cells, for new subtable T_j; go through old subtable T_j, put all elements not marked "deleted" into a list L_j, and mark all positions empty; append x to list L_j; $b_j \leftarrow$ length of L_j; **repeat** $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$ **until** h_j is injective on the elements of list L_j; **for** all y on list L_j store y in position $h_j(y)$ of T_j; **else** (* level-1-function h "bad" *) $RehashAll(x)$; *Id*. at Figure 1.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Dietzfelbinger discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Dietzfelbinger discloses that "[t]he dynamic dictionary problem is considered: provide an algorithm for storing a dynamic set, allowing the operations insert, delete, and lookup. A dynamic perfect hashing strategy is given: a *randomized* algorithm for the dynamic dictionary problem that takes *O*(1) worst-case time for lookups and *O*(1) amortized expected time for insertions and deletions . . . ." *Id*. at 1.

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		"A *dictionary* over a *universe* $U = \{0, 1, . . . , N - 1\}$ is a partial function S from *U* to some set *I*. The operations *Lookup(x)*, *Insert(x, i)*, and *Delete(x)* are available on a dictionary *S*; *Lookup(x)* returns *S(x)*, *Insert (x, i)* adds *x* to the domain of *S* and sets *S(x)* to *I*, and *Delete(x)* removes *x* from the domain of *S*." *Id*. at 2. Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." *Id*. Dietzfelbinger discloses an extension of the Fredman, Komlós, and Szemerédi scheme to the dynamic situation, wherein membership queries are processed in constant worst-case time, insertions and deletions are processed in constant expected amortized time, and the storage used at any time is proportional to the number of elements currently stored in the dictionary. "[F]redman, Komlós, and Szemerédi [FKS84] describe a hashing technique that achieves linear storage (in *n*) and constant query time for all *N* and *n*, where *n* is the size of *S*." *Id*. This FSK Scheme that Dietzfelbinger discloses "has two levels. At the top level, a hash function partitions the elements being stored into *s* sets. The second level consists of a perfect hash function for each of these sets. Specifically, a function *h* chosen uniformly at random from H_s is used to partition the set *S* into *s* blocks." *Id.* at 3-4. While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem,

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Dietzfelbinger discloses other schemes that do use hashing with chaining. "Carter and Wegman [CW79] proposed *universal hashing* as a way of avoiding assumption on the distribution of input values. This approach works particularly well in combination with the idea of "continuous rehashing" introduced by Brassard and Kannan [BK88]. In this way an algorithm is obtained that needs linear space and expected constant time for each single instruction. However, for *n* keys being stored in the dictionary in a scheme of this kind the best upper bound known on the expected *worst case time* for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is *O*(log *n* / log log *n*)..." *Id*. at 2-3.Since Dietzfelbinger does not specify which collision handling approach FKS uses (the focus of this paper is not collision handling but a computation of the upper and lower bounds for the time complexity of a class of deterministic algorithms for the dictionary problem), it is inherent from the disclosure of other schemes, such as the CW scheme, that FKS also uses hashing with chaining. To the extent that Bedrock argues that Dietzefelbinger does not anticipate Claims 1 – 8 because it is not inherent that the FKS scheme incorporates linked lists to handling key collisions, it would have been obvious to one of ordinary skill in the art to store table entries in a linked list. The admitted prior art in the Introduction discloses that linked lists/external chaining were already common place to resolve collisions within hash tables. *Id*. Thus, Dietzefelbinger and the admitted prior art in the Introduction show that one of ordinary skill in the art understood how to use linked lists/external chaining to resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Furthermore, Dietzfelbinger discloses that "[d]eletions are performed by attaching a tag "deleted" to the table entry to be erased; only when a new level-1 hash function *h* or a new hash function h_j for the sub table T_j is chosen, do we drop the elements with a tag "deleted" from T_j." *Id*. at 5. A call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. If the position in which *x* is to be placed is empty, then *x* is stored in the position. However, if the position is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. at 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked "deleted" then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked "deleted" into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		else (* T_j is too small *) $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$; **if** condition (**) is still satisfied **then** (* double capacity of T_j *) allocate new space, namely s_j cells, for new subtable T_j; go through old subtable T_j, put all elements not marked “deleted” into a list L_j, and mark all positions empty; append x to list L_j; $b_j \leftarrow$ length of L_j; **repeat** $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$ **until** h_j is injective on the elements of list L_j; **for** all y on list L_j store y in position $h_j(y)$ of T_j; **else** (* level-1-function h “bad” *) $RehashAll(x)$; *Id*. at Figure 1.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Dietzfelbinger discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, a call to the procedure *Insert(x)* conducts the following steps, as displayed by the code in the figure below. The procedure hashes *x* to determine the proper position for *x*, $h_j(x)$. First, the procedure determines if *x* already exists in the sub table. If *x* does exist, then the only determination to make is whether *x* is marked for deletion. If *x* is marked for deletion, then the procedure removes this deletion tag. *Id*. at 6.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Second, if the sub table does not contain *x* ***and*** the position in which *x* is to be placed is found empty, then *x* is stored in the position. *Id*. Third, if the sub table does not contain *x* ***and*** the position in which *x* is to be placed is not empty, then the procedure goes through and places all the elements in the sub table that are not labeled "deleted" into a list L_j, and marks all positions in the sub table as empty. Then, *x* is appended to the list L_j. By marking all positions in the sub table as empty, the procedure effectively deletes all those that are left in the sub table -- those that were labeled "deleted." *Id*. Therefore, the procedure dynamically determines whether or not to delete the records marked "deleted" if either of the following two conditions occur: (1) *x* already exists in the sub table or (2) the position in which *x* is to be placed is empty. Deletion takes place during a call to procedure *Insert(x)* only if the above two conditions are not met.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").

```
procedure $Insert(x)$;
  $count \leftarrow count + 1$;
  if $count > M$
  then
    $RehashAll(x)$;
  else
    $j \leftarrow h(x)$;
    if position $h_j(x)$ of subtable $T_j$ contains $x$
      then
        if $x$ is marked "deleted" then remove this tag;
      else (* $x$ is new for $W_j$ *)
        $b_j \leftarrow b_j + 1$;
        if $b_j \leq m_j$
          then (* size of $T_j$ sufficient *)
            if position $h_j(x)$ of $T_j$ is empty
              then
                store $x$ in position $h_j(x)$ of $T_j$;
              else
                go through the subtable $T_j$, put all elements
                not marked "deleted" into a list $L_j$, and
                mark all positions of $T_j$ empty;
                append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
                repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
                until $h_j$ is injective on the elements of list $L_j$;
                for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
```

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**

```
else (* $T_j$ is too small *)
  $m_j \leftarrow 2 \cdot \max\{1, m_j\}$; $s_j \leftarrow 2m_j(m_j - 1)$;
  if condition (**) is still satisfied
    then (* double capacity of $T_j$ *)
      allocate new space, namely $s_j$ cells, for new subtable $T_j$;
      go through old subtable $T_j$, put all elements
      not marked "deleted" into a list $L_j$,
      and mark all positions empty;
      append $x$ to list $L_j$; $b_j \leftarrow$ length of $L_j$;
      repeat $h_j \leftarrow$ randomly chosen function in $\mathcal{H}_{s_j}$
      until $h_j$ is injective on the elements of list $L_j$;
      for all $y$ on list $L_j$ store $y$ in position $h_j(y)$ of $T_j$;
    else (* level-1-function $h$ "bad" *)
      $RehashAll(x)$;
```

Id. at Figure 1.

Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dietzfelbinger to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dietzfelbinger with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Dietzfelbinger is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Dietzfelbinger combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Dietzfelbinger and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables

EXHIBIT C-6

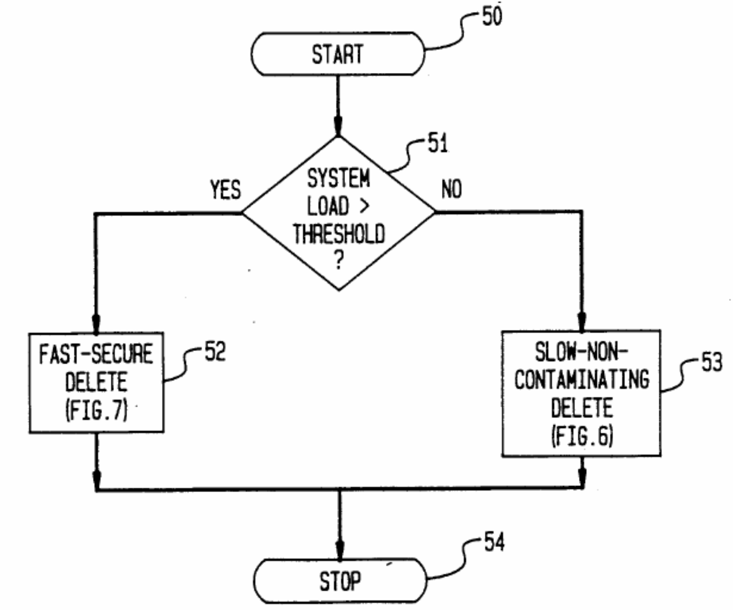
Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).
		implementations such as Dietzfelbinger. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks’ deletion decision procedure with Dietzfelbinger nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks’ dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Dietzfelbinger and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Dietzfelbinger with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 (“Dietzfelbinger”).**
		Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Dietzfelbinger and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Dietzfelbinger with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Dietzfelbinger with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system’s load and slowing down the system’s processing. One of ordinary skill in the art would recognize that combining Dietzfelbinger with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Dietzfelbinger with Thatte. Alternatively, it would also be obvious to combine Dietzfelbinger with the ’663 patent. Disclosure of these claim elements in the ’663 patent is clearly

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT C-6

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Dietzfelbinger and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Dietzfelbinger. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Dietzfelbinger would be nothing more than the predictable use of prior art elements according

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Dietzfelbinger and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Dietzfelbinger with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Dietzfelbinger and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Dietzfelbinger. Moreover, one of ordinary skill

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Dietzfelbinger would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Dietzfelbinger and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dietzfelbinger to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Dietzfelbinger with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Dietzfelbinger can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Dietzfelbinger in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Dietzfelbinger. For example, both Linux 2.0.1 and Dietzfelbinger describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		integer variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1359. When the function rt_cache_add removes an expired record, the function rt_cache_add decrements the variable rt_cache_size. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable rt_cache_size indicates the number of records in the hash table (i.e., ip_rt_hash_table). Because the function rt_cache_add automatically increments and decrements the variable rt_cache_size, the variable rt_cache_size is determined dynamically. Furthermore, LINUX 2.0.1 includes the function rt_garbage_collect_1. The function rt_garbage_collect_1 loops through each of the linked lists in the ip_rt_hash_table global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function rt_garbage_collect_1 accesses the linked list. When the function rt_garbage_collect_1 identifies a record that is expired, the function rt_garbage_collect_1 decrements the variable rt_cache_size and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable rt_cache_size can represent a dynamically determined maximum number of expired ones of the records to remove when function rt_garbage_collect_1 accesses the linked list. Furthermore, the function rt_cache_add determines whether the number of records in the hash table exceeds a predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		`RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

Asserted Claims From U.S. Pat. No. 5,893,120		Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can

Asserted Claims From U.S. Pat. No. 5,893,120		**Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Revised Version January 7, 1990 ("Dietzfelbinger").**
		remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Griffioen discloses an information storage and retrieval system. For example, the Remote Memory Model described in Griffioen "consists of multiple *client* machines, one or more *memory server machines*, various other servers (e.g., time servers, name servers, or file servers), and a communication channel interconnecting all the machines." Griffioen at 21. The model "centers around the use of remote memory servers for backing storage." See Griffioen at 91.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Griffioen describes the use of a hash table with a double hashing algorithm to locate data. Griffioen at 101. For example, there are two well-known approaches to solving the problem of collisions within a hash table, which occur whenever two entries "hash" or are assigned to the same "bucket" within the hash table. The computer programmer may store the records external to the hash table—that is, using memory separate from the memory allocated to the hash table—or he may store the records internal to the hash table—that is, using memory that is allocated to other buckets within the hash table. Using external memory is termed "external chaining," while using internal memory is termed "open addressing." See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549, 1973. The applicant has conceded that both forms of collision resolution are known to those of ordinary skill in the art. See, e.g., '120 patent at 1:53-57 (describing linear probing—a type of open addressing—as being "often used" for "collision resolution"); 1:58-2:6

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		(citing to several prior art resources that describe external chaining as using linked lists). Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." Knuth at 513. *See also* Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Double hashing is another form of open addressing. *See* Knuth at 519-524. It would have been obvious to one skilled in the art to apply the teachings in Griffioen to a hash table which resolves collisions using external chaining with linked lists. As detailed above, one of ordinary skill has a limited number of ways of resolving hash collisions: he may either store the entries within the hash table or outside the hash table, and both were well known to those of ordinary skill. The records in the system Griffioen discloses includes records, at least some of which automatically expire. For example, Griffioen describes a memory reclamation process through which the remote memory server maintains a timestamp for each process in the system in a separate process hash table. "When the memory server receives a page store request for a process, the server saves the timestamp of the requesting process in the virtual-page hash table entry. Consequently, all pages belonging to a VS have the VS's timestamp. When the memory server receives a terminate request, it updates the current timestamp in the VS table, thereby invalidating all the pages in the VS." Griffioen at 106. "Updating a

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		timestamp invalidates all the data in a VS or LMS in a single operation." Griffioen at 108.
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Griffioen discloses a record search means utilizing a search key to access the chain of records. Griffioen also discloses a record search means utilizing a search key to access a chain of records beginning at the same hash address. For example, Griffioen explains that "[c]lient machines uniquely label each page with an ordered triple containing the LMS ID, VS ID, and page number. Given a paging request, the server can quickly verify whether a particular hash table entry contains the requested page." Griffioen at 100. "The memory server efficiently locates a hash table entry by applying a double hashing algorithm to the ordered triple that uniquely identifies the desired page." Griffioen at 101. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. As such, the search means utilizing the search key would be accessing a linked list of records beginning at the same hash address.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when	Griffioen discloses the record search means including means for identifying and removing at least some expired ones of the records from the chain of records when the chain is accessed. For example, the remote memory server "reclaims memory while processing store and fetch requests. When a client issues a store or fetch request, the

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
when the linked list is accessed, and	the linked list is accessed, and	server applies the double hashing algorithm to locate an entry in the virtual-page hash table. The double hashing algorithm requires the server to check for a collision on each probe to the table. That is, the server must compare the requested page's information against the information found in the hash table entry to see if the entry contains the desired page. If a collision occurs, the server checks the timestamp found in the hash table entry against the owner's timestamp in the VS and LMS hash tables. If the timestamps differ, the server reclaims the page. If the timestamps match, the hash table entry is still valid, and the double hashing algorithm proceeds as normal. This modification to the double hashing algorithm allows the server to reclaim invalid pages during its normal processing." Griffioen at 108. The remote memory server also performs garbage collection in the background. *Id.* As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the	Griffioen discloses means, utilizing the record search means, for accessing the chain of records and, at the same time, removing at least some of the expired ones of the records in the chain. Griffioen also discloses means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed chain of records.

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
	records in the accessed linked list of records.	For example, the remote memory server "reclaims memory while processing store and fetch requests. When a client issues a store or fetch request, the server applies the double hashing algorithm to locate an entry in the virtual-page hash table. The double hashing algorithm requires the server to check for a collision on each probe to the table. That is, the server must compare the requested page's information against the information found in the hash table entry to see if the entry contains the desired page. If a collision occurs, the server checks the timestamp found in the hash table entry against the owner's timestamp in the VS and LMS hash tables. If the timestamps differ, the server reclaims the page. If the timestamps match, the hash table entry is still valid, and the double hashing algorithm proceeds as normal. This modification to the double hashing algorithm allows the server to reclaim invalid pages during its normal processing." Griffioen at 108. The remote memory server also performs garbage collection in the background. *Id.* To the extent Griffioen does not disclose removal of expired records during a deletion of records, it would have been obvious to one of ordinary skill in the art that a deletion could occur at such a time, since insertion, retrievals, and deletions are basic operations that can be performed on all hash tables. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.
2. The information storage	6. The information storage	Griffioen combined with Dirks, Thatte, the '663 patent and/or the

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20).

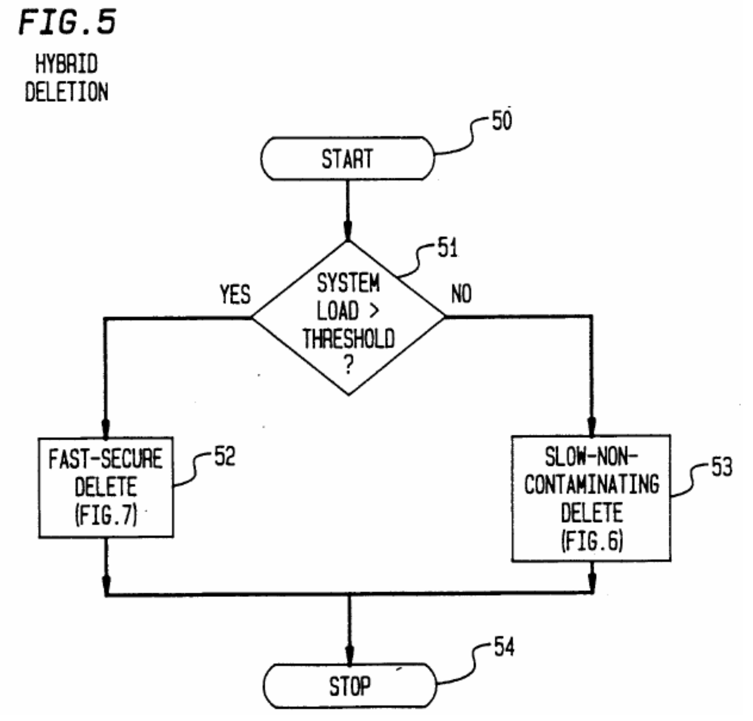
Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Griffioen and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Griffioen. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Griffioen nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Griffioen and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.`

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Griffioen with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Griffioen and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Griffioen with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Griffioen with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Griffioen can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Griffioen with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Griffioen with Thatte. Alternatively, it would also be obvious to combine Griffioen with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Griffioen and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Griffioen. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Griffioen would be nothing more than the predictable use of prior art elements according to

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Griffioen and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Griffioen with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Griffioen and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Griffioen. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Griffioen would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Griffioen and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Griffioen to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Griffioen with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Griffioen can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Griffioen in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Griffioen. For example, both Linux 2.0.1 and Griffioen describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		`RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Griffioen also discloses a method for storing and retrieving information records using a hashing technique, at least some of the records automatically expiring. For example, the Remote Memory Model described in Griffioen "consists of multiple *client* machines, one or more *memory server machines*, various other servers (e.g., time servers, name servers, or file servers), and a communication channel interconnecting all the machines." Griffioen at 21. The model "centers around the use of remote memory servers for backing storage." See Griffioen at 91. Griffioen describes a method of using a hash table with a double hashing algorithm to locate data. Griffioen at 101. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a hash table with external chaining using linked lists could be used instead of a hash table with double hashing. The records in the system Griffioen discloses includes records, at least some of which automatically expire.

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		For example, Griffioen describes a memory reclamation process through which the remote memory server maintains a timestamp for each process in the system in a separate process hash table. "When the memory server receives a page store request for a process, the server saves the timestamp of the requesting process in the virtual-page hash table entry. Consequently, all pages belonging to a VS have the VS's timestamp. When the memory server receives a terminate request, it updates the current timestamp in the VS table, thereby invalidating all the pages in the VS." Griffioen at 106. "Updating a timestamp invalidates all the data in a VS or LMS in a single operation." Griffioen at 108.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Griffioen discloses accessing the chain of records. Griffioen also discloses accessing a chain of records beginning at the same hash address. For example, Griffioen describes a method of using a hash table with a double hashing algorithm to locate data. Griffioen at 101. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Griffioen discloses identifying at least some of the automatically expired ones of the records. For example, the remote memory server "reclaims memory while processing store and fetch requests. When a client issues a store or fetch request, the server applies the double hashing algorithm to locate an entry in the virtual-page hash table. The double hashing algorithm requires the server to check for a collision on each probe to the table. That is, the server must compare the requested page's information against the information found in the hash table entry to see if the entry contains the desired page. If a collision occurs, the server checks the timestamp found in the hash table entry against the owner's timestamp in the VS and LMS hash tables. If the timestamps differ, the server reclaims the page. If the timestamps match, the hash table entry is still valid, and the double hashing algorithm proceeds as normal. This modification to the double hashing algorithm allows the server to reclaim invalid pages during its normal processing." Griffioen at 108. The remote memory server also performs garbage collection in the background. *Id.*
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Griffioen discloses removing at least some of the automatically expired records from the chain of records when the chain is accessed. For example, the remote memory server "reclaims memory while processing store and fetch requests. When a client issues a store or fetch request, the server applies the double hashing algorithm to locate an entry in the virtual-page hash table. The double hashing algorithm requires the server to check for a collision on each probe to the table. That is, the server must compare the requested page's information against the information found in the hash table

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		entry to see if the entry contains the desired page. If a collision occurs, the server checks the timestamp found in the hash table entry against the owner's timestamp in the VS and LMS hash tables. If the timestamps differ, the server reclaims the page. If the timestamps match, the hash table entry is still valid, and the double hashing algorithm proceeds as normal. This modification to the double hashing algorithm allows the server to reclaim invalid pages during its normal processing." Griffioen at 108. The remote memory server also performs garbage collection in the background. *Id.* As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Griffioen discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Griffioen states that the memory server "reclaims memory while processing store and fetch requests." Griffioen at 108. These store and fetch requests constitute insertions and retrievals from the hash table.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove	Griffioen combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
when the linked list is accessed.	when the linked list is accessed.	particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this

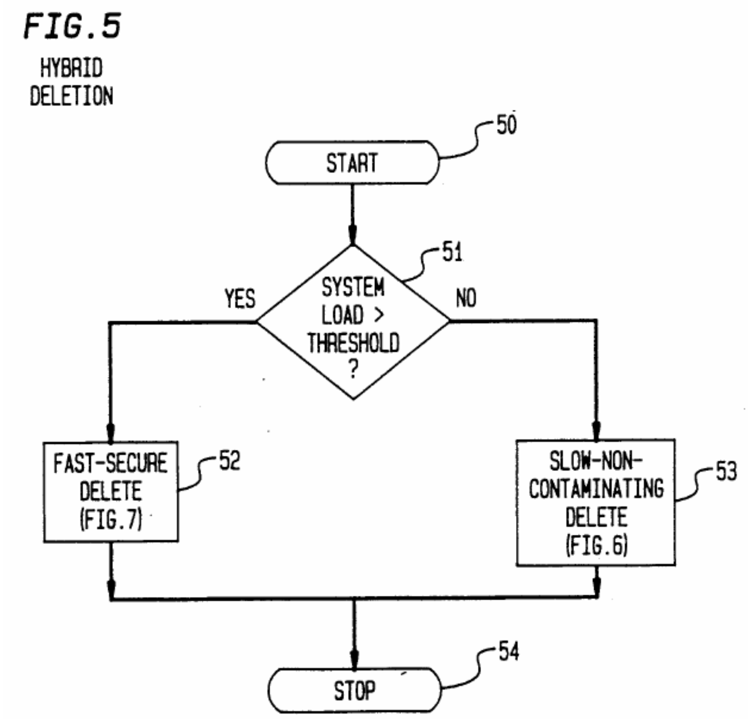
EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Griffioen and Dirks relate to deletion of aged records upon the

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Griffioen. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Griffioen nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Griffioen and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Griffioen with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Griffioen and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Griffioen with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Griffioen with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Griffioen can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Griffioen with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Griffioen with Thatte.

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		Alternatively, it would also be obvious to combine Griffioen with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels.

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		Id. at 2:42-46. This hybrid deletion is shown in Figure 5. FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Griffioen and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Griffioen. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Griffioen would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Griffioen and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Griffioen with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Griffioen and the Opportunistic Garbage Collection Articles relate to

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Griffioen. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Griffioen would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Griffioen and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Griffioen to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Griffioen with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Griffioen can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Griffioen in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Griffioen. For example, both Linux 2.0.1 and Griffioen describe systems and methods for performing data storage and retrieval using known programming techniques to

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list.

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination
		Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		`rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function

Asserted Claims From U.S. Pat. No. 5,893,120		**James Griffioen, *Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems*, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis ("Griffioen") alone and in combination**
		`rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

<u>EXHIBIT C-8</u>

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Comer discloses an information storage and retrieval system. For example, the Remote Memory Model described in Comer "consists of several client machines, various server machines, one or more dedicated machines called remote memory servers, and a communication channel interconnecting all the machines." See p. 2. "[C]lient machines use the remote memory server for backing storage." See p. 3. The "remote memory server transfers data to and from heterogeneous clients in an architecture-independent manner." *See* Comer at 9; *see also* Comer at 1.
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Comer describes the use of a hash table with a double hashing algorithm to locate data. See, e.g.,. 10. There are two well-known approaches to solving the problem of collisions within a hash table, which occur whenever two entries "hash" or are assigned to the same "bucket" within the hash table. The computer programmer may store the records external to the hash table—that is, using memory separate from the memory allocated to the hash table—or he may store the records internal to the hash table—that is, using memory that is allocated to other buckets within the hash table. Using external memory is termed "external chaining," while using internal memory is termed "open addressing." See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549, 1973. The applicant has conceded that both forms of collision resolution are known to those of ordinary skill in the art. See, e.g., '120 patent at 1:53-57 (describing linear probing—a type of open addressing—as being

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		"often used" for "collision resolution"); 1:58-2:6 (citing to several prior art resources that describe external chaining as using linked lists). Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." Knuth at 513. *See also* Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Double hashing is another form of open addressing. *See* Knuth at 519-524. It would have been obvious to one skilled in the art to apply the teachings in Comer to a hash table which resolves collisions using external chaining with linked lists. As detailed above, one of ordinary skill has a limited number of ways to resolve hash collisions: he may either store the entries within the hash table or outside the hash table, and both were well known to those of ordinary skill. The records in the system Comer discloses includes records, at least some of which automatically expire. For example, Comer describes a memory reclamation process through which the remote memory server maintains a timestamp for each process in the system in a separate process hash table. "When the server receives a page store request for a process, the server saves the process's timestamp with the page in the *data* hash table. Each time the server receives a terminate request, the server updates the timestamp in the process hash table, thereby invalidating all pages associated with the terminated process." P. 10.

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Comer discloses a record search means utilizing a search key to access the chain of records. Comer also discloses a record search means utilizing a search key to access a chain of records beginning at the same hash address. For example, Comer explains that "[c]lient machines uniquely identify a page with an ordered triple consisting of a unique machine identifier, a process identifier, and a page identifier. The server applies a double hashing algorithm to the triple to locate the hash table entry that contains pointers to the data." As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. As such, the search means utilizing the search key would be accessing a linked list of records beginning at the same hash address.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Comer discloses the record search means including means for identifying and removing at least some expired ones of the records from the chain of records when the chain is accessed. For example, the remote memory server "reclaims obsolete pages during later probes to the data hash table and with a garbage collection process that executes in the background." P. 10-11. "Each time a probe to the data hash table results in a collision, the server checks the timestamp on the page against the timestamp of the owner. If the timestamps differ, the server reclaims the page. Together, the garbage collection process and the lazy reclamation algorithm amortize the cost of reclaiming memory over time." P. 11.

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Comer discloses means, utilizing the record search means, for accessing the chain of records and, at the same time, removing at least some of the expired ones of the records in the chain. Comer also discloses means, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed chain of records. For example, the remote memory server "reclaims obsolete pages during later probes to the data hash table." P. 10. "Each time a probe to the data hash table results in a collision, the server checks the timestamp on the page against the timestamp of the owner. If the timestamps differ, the server reclaims the page." P. 10. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Comer combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14.

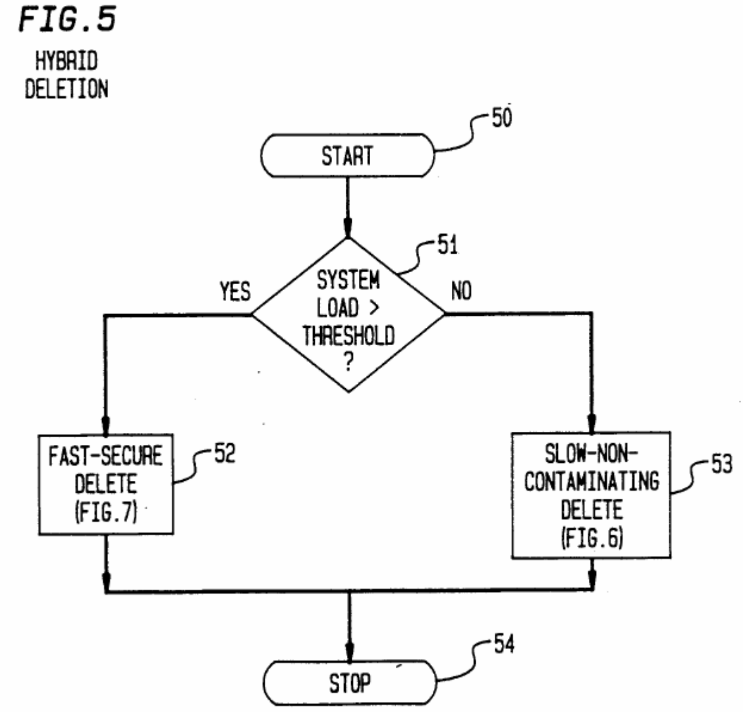
Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination**
		After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT$_i$ on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Comer and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Comer. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Comer nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Comer and would have seen the benefits of doing so. One possible benefit, for example, is saving the

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Comer with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Comer and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Comer with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Comer with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Comer can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Comer with the teachings of Thatte would solve this problem by dynamically determining how

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Comer with Thatte. Alternatively, it would also be obvious to combine Comer with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		 FIG.5 HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Comer and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Comer. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Comer would be nothing more than the predictable use of prior art elements according to their

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Comer and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Comer with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Comer and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Comer. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Comer would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Comer and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Comer to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Comer with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		records to solve a number of potential problems. For example, the removal of expired records described in Comer can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Comer in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Comer. For example, both Linux 2.0.1 and Comer describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		`RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Comer also discloses method for storing and retrieving information records using a hashing technique, at least some of the records automatically expiring. For example, the Remote Memory Model described in Comer "consists of several client machines, various server machines, one or more dedicated machines called remote memory servers, and a communication channel interconnecting all the machines." See Comer at 2. "[C]lient machines use the remote memory server for backing storage." See Comer at 3. The "remote memory server transfers data to and from heterogeneous clients in an architecture-independent manner." See Comer at 9; *see also* Comer at 1. Comer describes a method of using of a hash table with a double hashing algorithm to locate data on the remote memory server. See, e.g., p. 10. There are two well-known approaches to solving the problem of collisions within a hash table, which occur whenever two entries "hash" or are assigned to the same "bucket" within the hash table. The computer programmer may store the records external to the hash table—that is, using memory separate from the memory allocated to the hash table—or he may store the records internal to the hash table—that is, using memory that is allocated to other buckets within the

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		hash table. Using external memory is termed "external chaining," while using internal memory is termed "open addressing." See "The Art of Computer Programming", Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506-549, 1973. The applicant has conceded that both forms of collision resolution are known to those of ordinary skill in the art. See, e.g., '120 patent at 1:53-57 (describing linear probing—a type of open addressing—as being "often used" for "collision resolution"); 1:58-2:6 (citing to several prior art resources that describe external chaining as using linked lists). Indeed, Knuth recognizes that "[p]erhaps the most obvious way to solve this problem [of collision resolution] is to maintain M linked Lists, one for each possible hash code [i.e. external chaining]." Knuth at 513. *See also* Mark A. Weiss, Data Structures and Algorithm Analysis, p. 157, 1993 ("*Closed hashing*, also known as *open addressing*, is an alternative to resolving collisions with linked lists."). Double hashing is another form of open addressing. *See* Knuth at 519-524. It would have been obvious to one skilled in the art to apply the teachings in Comer to a hash table which resolves collisions using external chaining with linked lists. As detailed above, one of ordinary skill has a limited number of ways of resolving hash collisions: he may either store the entries within the hash table or outside the hash table, and both were well known to those of ordinary skill. The records in the system Comer discloses includes records, at least some of which automatically expire. For example, Comer describes a memory reclamation process through which

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		the remote memory server maintains a timestamp for each process in the system in a separate process hash table. "When the server receives a page store request for a process, the server saves the process's timestamp with the page in the *data* hash table. Each time the server receives a terminate request, the server updates the timestamp in the process hash table, thereby invalidating all pages associated with the terminated process." P. 10.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Comer discloses accessing the chain of records. Comer also discloses accessing a chain of records beginning at the same hash address. For example, Comer describes a method of using of a hash table with a double hashing algorithm to locate data on the remote memory server. See, e.g., p. 10. As discussed in [3/7], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Comer discloses identifying at least some of the automatically expired ones of the records. For example, the remote memory server "reclaims obsolete pages during later probes to the data hash table." P. 10. "Each time a probe to the data hash table results in a collision, the server checks the timestamp on the page against the timestamp of the owner. If the timestamps differ, the server reclaims the page." P. 10.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Comer discloses the removing at least some of the automatically expired records from the chain of records when the chain of records is accessed. For example, the remote memory server "reclaims obsolete pages during later probes to the data hash table and with a garbage collection process that executes in the background." P. 10-11. "Each time a probe to the data hash table results in a collision, the server checks the timestamp on the page against the timestamp of the owner. If the timestamps differ, the server reclaims the page. Together, the garbage collection process and the lazy reclamation algorithm amortize the cost of reclaiming memory over time." P. 11. As discussed in [3/7], it would have been obvious to one of ordinary skill in the art to use a hash table with external chaining instead of a hash table with open addressing/double hashing. In such a system, the probe that resulted from a collision would occur on the linked list used to resolve the collision. As such, the expired records from the linked list would be removed when the algorithm processes the linked list in search of the desired record.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Comer discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Comer states that "[t]he server reclaims obsolete pages during later probes to the data hash table . . . ." P. 10.
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Comer combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries

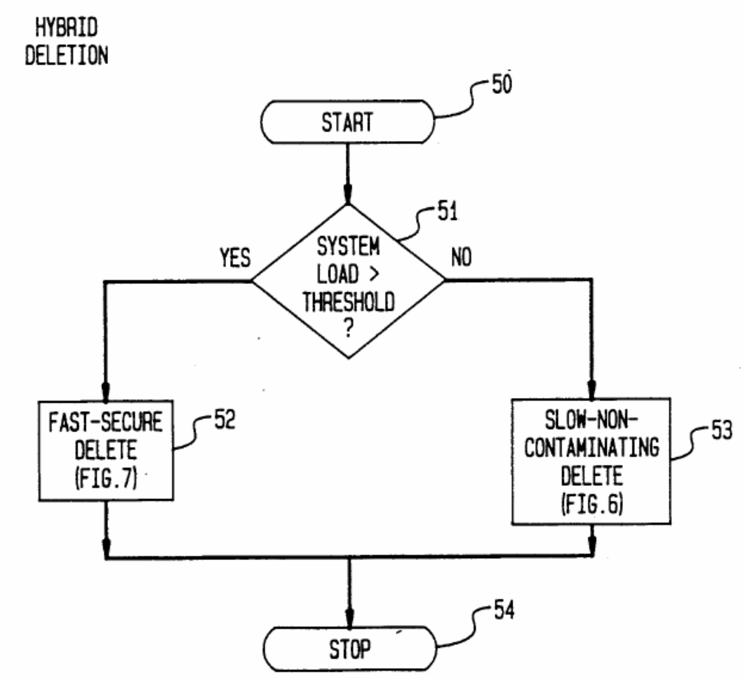
Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks:

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Comer and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Comer. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Comer nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Comer and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Comer with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, Ass both Comer and Thatte teach a system of data storage and retrieval, one of ordinary skill in the art would recognize that the result of combining Comer with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove as taught by Thatte. Further, one of ordinary skill in the art would be motivated to combine Comer

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Comer can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Comer with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Comer with Thatte. Alternatively, it would also be obvious to combine Comer with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent").

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		FIG.5  SYSTEM LOAD > THRESHOLD ? FAST-SECURE DELETE (FIG.7) SLOW-NON-CONTAMINATING DELETE (FIG.6) *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Comer and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Comer. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Comer would be nothing more than the predictable use of prior art elements according to their

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Comer and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Comer with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Comer and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Comer. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Comer would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Comer and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Comer to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Comer with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		records to solve a number of potential problems. For example, the removal of expired records described in Comer can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Comer in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Comer. For example, both Linux 2.0.1 and Comer describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359.

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		**Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination**
		`RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, *A New Design for Distributed Systems: The Remote Memory Model* (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf ("Comer") alone and in combination
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-9

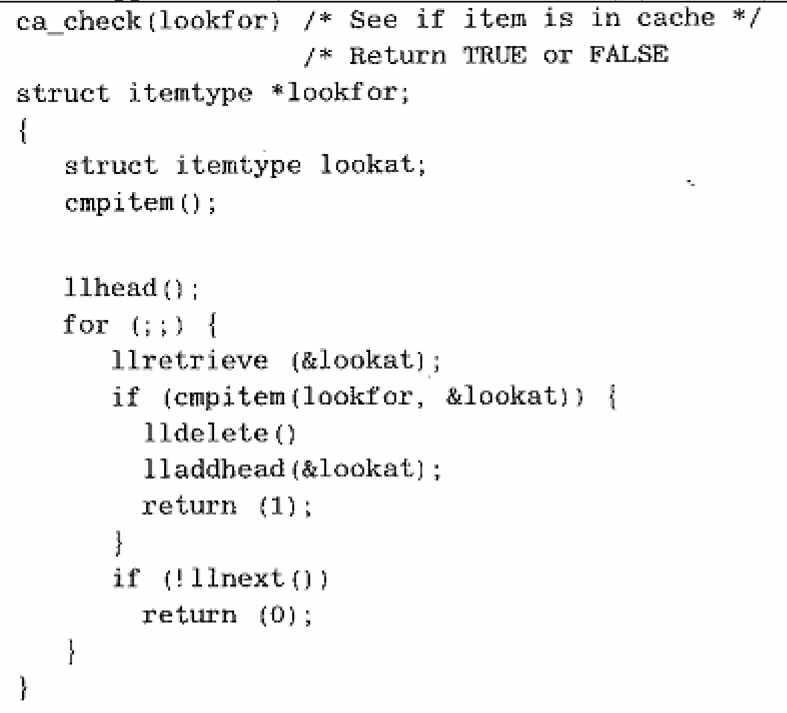
Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Sessions discloses an information storage and retrieval system. For example, Sessions discloses that "[o]ur newly completed linked list package can now form the basis for a cache." (Sessions at 29). Thus, Sessions inherently discloses an information storage and retrieval system. (*See id.*)
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Sessions discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. To the extent that Bedrock argues that Sessions does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Sessions with Lester. For example, Lester discloses hash tables where "the hash table entries could be pointer-type values, or array indexes, or some encoding of array indexes." (Lester at 153). Lester further discloses an external chaining technique to store the records with same hash address by stating "[e]ach hash-table element that doesn't yet correspond to any data would contain **nil**, otherwise it would point either • to a keyed table or to a keyed linked-list of those data-items whose keys all hash to that place in the table" (*See id.*). Lester also states that "I usually use a keyed linked-list: it is simple to program, and if any of the lists get long it means that I should have made the hash-array bigger, to get more lists, and therefore shorter lists." (Lester at 154). Thus, Sessions and Lester show that one of ordinary skill in the art understood how to utilize a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, and would recognize that it would improve

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		similar systems and methods in the same way. Sessions also discloses that a “function `ca_add()` assumes that an item is not in the cache,” and that “this function has three variable states. … 2. The cache is full and an item must be discarded. As already described, the tail item is always discarded.” (Sessions at 29).
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Sessions discloses a record search means utilizing a search key to access the linked list. For example, Sessions discloses that “[t]he next function, `ca_check`(), searches the cache for a particular item, returning a value of true or false depending on the search results. If the item is found, it becomes the most recently reference item and is promoted to the head of the list.

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		(continued below)



```
ca_check(lookfor) /* See if item is in cache */
                  /* Return TRUE or FALSE
struct itemtype *lookfor;
{
   struct itemtype lookat;
   cmpitem();

   llhead();
   for (;;) {
      llretrieve (&lookat);
      if (cmpitem(lookfor, &lookat)) {
        lldelete()
        lladdhead(&lookat);
        return (1);
      }
      if (!llnext())
        return (0);
   }
}
```

(Sessions at 30).

To the extent that Bedrock argues that Sessions does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Sessions with Lester. For example, Lester discloses that "[t]he complier *should* include "garbage collection" in a program translated from a Pascal source that uses any pointer-types: when the program runs low on spare

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		memory the garbage collection should search for any nodes that no longer have pointers to them, and deallocate them, freeing up some spare." (Lester at 69). Therefore, Lester inherently discloses "accessing a linked list of records having the same hash address." (*See id.*) Thus, Sessions and Lester show that one of ordinary skill in the art understood a record search means utilizing a search key to access a linked list of records having the same hash address, and would recognize that it would improve similar systems and methods in the same way.
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Sessions discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Sessions also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Sessions discloses the standard function `free`(), which "receives a pointer to one of the blocks in the dynamic memory pool previously allocated by `malloc`(). The pool is first scanned to validate the pointer, and then the block is released for use. (Sessions at 78).
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Sessions discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Sessions also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Sessions discloses that the "functions `mmget`() and `mmfree`()

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		can solve this problem by dynamically collecting free memory into a single contiguous chunk. … if a program uses `mmget`() to set a pointer `p1` to a block of memory, `p1` will always point to the same values. Any dynamic memory collection that results in relocating the memory block also results in an automatic update of `p1`." (Sessions at 78-79). Also, Sessions discloses that "[t]he static functions `setfree`() and consolidate() perform the work of freeing a used memory block. … This rudimentary garbage collection could be accomplished by the standard C library function `free`() as well. (Sessions at 83). Further, Sessions discloses an exercise where the student must "[m]odify `mmfree`() so that a full garbage collection is always performed with each call. What are the advantages and disadvantages of this change?" (Sessions at 86).
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Sessions discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Sessions discloses that "[h]owever many blocks we decide to save, we would eventually read one block too many. Then we would have to reuse the memory space occupied by one of the blocks. Which block should be thrown away? The best block to discard is the block we would least likely want again." (Sessions at 89). Sessions also discloses a routine `ca_check()` which "promote(s) any found items to the head of the linked list, indicating their changed status to 'Most

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		Recently Referenced.’” (Sessions at 91). Sessions further discloses an exercise where the student must “[m]odify mmfree() so that garbage is collected whenever the largest single chunk of available memory is less than ¾ of the total available pool.” (Sessions at 86). Thus, Sessions inherently discloses a means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. (*See id.*) Sessions combined with Dirks, Thatte, the ’663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30.

EXHIBIT C-9

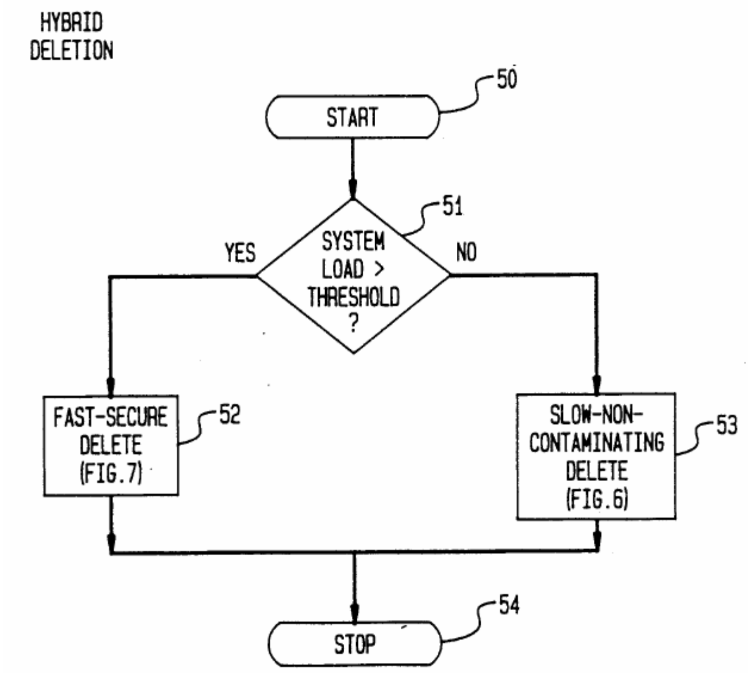
Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Sessions and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Sessions. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		combining Dirks' deletion decision procedure with Sessions nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Sessions and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Sessions with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Sessions with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove.

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		Further, one of ordinary skill in the art would be motivated to combine Sessions with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Sessions can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Sessions with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Sessions with Thatte. Alternatively, it would also be obvious to combine Sessionswith the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: > during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		4,996,663 to Nemes at 2:24-34 (“The ’663 patent”). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as “deleted” and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Sessions and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Sessions. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Sessions would be nothing more than the predictable use of prior art elements according to

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Sessions and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Sessions with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation’s space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Sessions and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles’ dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Sessions. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Sessions would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Sessions and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Sessions to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Sessions with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		linked list of records to solve a number of potential problems. For example, the removal of expired records described in Sessionscan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Sessions in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Sessions. For example, both Linux 2.0.1 and Sessions describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		`RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

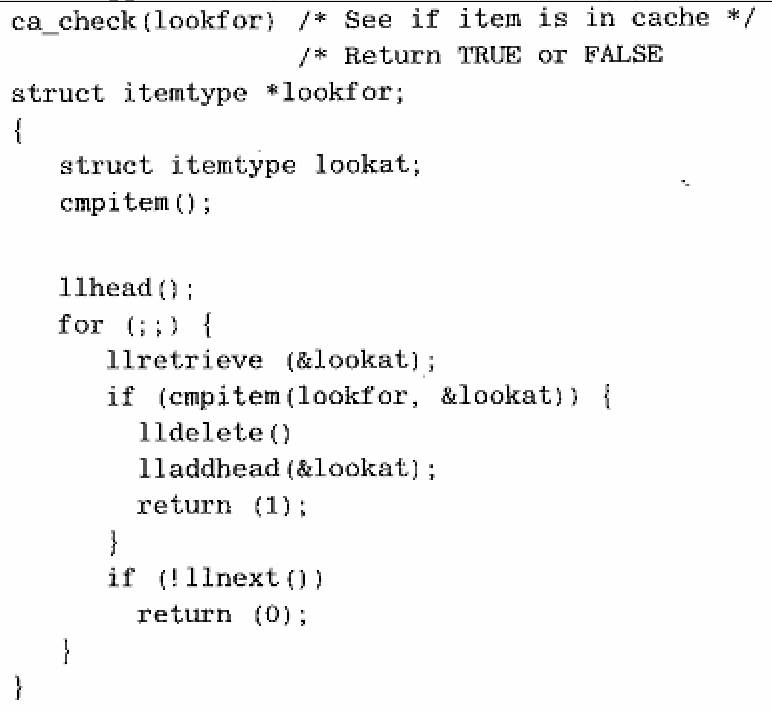
Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Sessions discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. To the extent that Bedrock argues that Sessions does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Sessions with Lester. For example, Lester discloses hash tables where "the hash table entries could be pointer-type values, or array indexes, or some encoding of array indexes." (Lester at 153). Lester further discloses an external chaining technique to store the records with same hash address by stating "[e]ach hash-table element that doesn't yet correspond to any data would contain **nil**, otherwise it would point either • to a keyed table or to a keyed linked-list of those data-items whose keys all hash to that place in the table" (*See id.*). Lester also states that "I usually use a keyed linked-list: it is simple to program, and if any of the lists get long it means that I should have made the hash-array bigger, to get more lists, and therefore shorter lists." (Lester at

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		154). Thus, Sessions and Lester show that one of ordinary skill in the art understood how to utilize a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, and would recognize that it would improve similar systems and methods in the same way. For example, Sessions discloses that "[o]ur newly completed linked list package can now form the basis for a cache." (Sessions at 29). Thus, Sessions inherently discloses a method for storing and retrieving information records using a linked list to store and provide access to the records. Sessions also discloses that a "function `ca_add()` assumes that an item is not in the cache," and that "this function has three variable states. … 2. The cache is full and an item must be discarded. As already described, the tail item is always discarded." (Sessions at 29).
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Sessions discloses accessing a linked list of records. For example, Sessions discloses that "[t]he next function, `ca_check`(), searches the cache for a particular item, returning a value of true or false depending on the search results. If the item is found, it becomes the most recently reference item and is promoted to the head of the list.

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		

```
ca_check(lookfor) /* See if item is in cache */
                  /* Return TRUE or FALSE
struct itemtype *lookfor;
{
   struct itemtype lookat;
   cmpitem();

   llhead();
   for (;;) {
      llretrieve (&lookat);
      if (cmpitem(lookfor, &lookat)) {
        lldelete()
        lladdhead(&lookat);
        return (1);
      }
      if (!llnext())
        return (0);
   }
}
```

(Sessions at 30).

To the extent that Bedrock argues that Sessions does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Sessions with Lester. For example, Lester discloses that “[t]he complier *should* include “garbage collection” in a program translated from a Pascal source that uses any pointer-types: when the program runs low on spare

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		memory the garbage collection should search for any nodes that no longer have pointers to them, and deallocate them, freeing up some spare." (Lester at 69). Thus, Lester inherently discloses "accessing a linked list of records having same hash address." (*See id.*) Together, Sessions and Lester show that one of ordinary skill in the art understood how to access a linked list of records having same hash address, and would recognize that it would improve similar systems and methods in the same way.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Sessions discloses identifying at least some of the automatically expired ones of the records. Sessions also discloses identifying at least some of the automatically expired ones of the records. For example, Sessions discloses the standard function `free`(), which "receives a pointer to one of the blocks in the dynamic memory pool previously allocated by `malloc`(). The pool is first scanned to validate the pointer, and then the block is released for use. (Sessions at 78).
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Sessions discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Sessions also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Also, Sessions discloses that "[t]he static functions `setfree`() and consolidate() perform the work of freeing a used memory block. … This rudimentary garbage collection could be accomplished by the standard C library function `free`() as well. (Sessions at 83). Further, Sessions discloses an exercise where the student must "[m]odify

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		`mmfree`() so that a full garbage collection is always performed with each call. What are the advantages and disadvantages of this change?" (Sessions at 86).
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Sessions discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Sessions discloses deleting one of the records from the system in the `consolidate`() function following the step of removing.

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others

		`static consolidate(first, second)`
		`struct memory_type *first, *second;`
		`{`
		`   first->nbytes += second->nbytes;      /* Consolidate sizes.  */`
		`   first->start =                         /* See which is first. */`
		`      (first->start < second->start) ?`
		`      (first->start) : (second->start);`
		`}`
		`static setfree()`
		`{`
		`struct memory_type *free, *other;`
		
		`/* Prepare current block for freedom.`
		`   ---------------------------------- */`
		`   free = llexamine();          /* See which block is current.   */`
		`   free->addr_owner = 0;        /* Note that block is unowned... */`
		`   free->type = FREE;           /* ... and available for use.    */`
		
		`/* If previous block is also free, consolidate.`
		`   -------------------------------------------- */`
		`   if (!llishead()) {               /* Don't go beyond head.    */`
		`      llprevious();                 /* Check the previous block.*/`
		`      other = llexamine();`
		`      if (other->type == FREE) {    /* If free, consolidate.    */`
		`         consolidate(free, other);`
		`         lldelete();`
		(Sessions at 83).

4. The method according to	8. The method according	Sessions discloses dynamically determining maximum number of expired ones

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	of the records to remove when the linked list is accessed. For example, Sessions discloses that “[h]owever many blocks we decide to save, we would eventually read one block too many. Then we would have to reuse the memory space occupied by one of the blocks. Which block should be thrown away? The best block to discard is the block we would least likely want again.” (Sessions at 89). Sessions also discloses a routine `ca_check()` which “promote(s) any found items to the head of the linked list, indicating their changed status to ‘Most Recently Referenced.’” (Sessions at 91). Sessions further discloses an exercise where the student must “[m]odify mmfree() so that garbage is collected whenever the largest single chunk of available memory is less than ¾ of the total available pool.” (Sessions at 86). Thus, Sessions inherently discloses a step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. (*See id.*) Sessions combined with Dirks, Thatte, the ’663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The

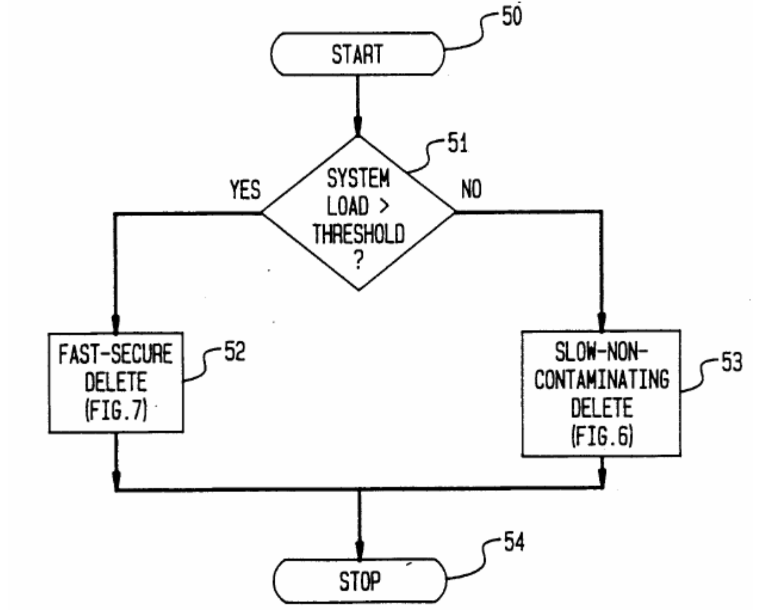
Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Sessions and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) (“Lester”) and others**
		the maximum number of records to sweep/remove in other hash tables implementations such as that described Sessions. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one.” The ’120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks’ deletion decision procedure with Sessions would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks’ dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Sessions and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Sessions with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Sessions with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Sessions with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Sessions can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Sessions with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Sessions with Thatte. Alternatively, it would also be obvious to combine Sessions with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		FIG.5 HYBRID DELETION  START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Sessions and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Sessions. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Sessions would be nothing more than the predictable use of prior art elements according to their established

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Sessions and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Sessions with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Sessions and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Sessions. Moreover, one of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Sessions would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Sessions and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Sessions to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Sessions with the fundamental concept of dynamically

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Sessions can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. To the extent that dynamically determining a maximum number of expired records is not disclosed by Sessions in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Sessions. For example, both Linux 2.0.1 and Sessions describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold

Asserted Claims From U.S. Pat. No. 5,893,120		**Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others**
		RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1,

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, *Reusable Data Structures for C* (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, *A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications* (Ellis Horwood Ltd. 1990) ("Lester") and others
		route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Van Wyk 2 discloses an information storage and retrieval system. For example, Van Wyk 2 discloses external data structures, explaining that "*File systems* are a familiar example of an external data structure." (Van Wyk 2 at 142).
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Van Wyk 2 discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Van Wyk 2 also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Van Wyk 2 discloses that "[i]f the header nodes can be linked together, then no arbitrary limit on file size need be imposed." (Van Wyk 2 at 142). Therefore, Van Wyk 2 inherently discloses a linked list to store and provide access to records stored in a memory of the system. (*See id.*) Van Wyk 2 further discloses that "[i]n the *reference count* scheme for garbage collection, each node contains a value that tells how many pointers point to it. When a node's reference count becomes zero, the node is garbage and can be collected. (Van Wyk 2 at 145). Thus, Van Wyk 2 inherently discloses that at least some of the records are automatically expiring. (*See id.*) Van Wyk 2 further discloses that "[h]ashing with linked lists is an excellent solution to many searching problems. At the price of some space for pointers, we obtain a table of potentially unlimited size that readily supports insertions and deletions." (Van Wyk 2 at 186).

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination**
[1b] a record search means utilizing a search key to access the linked list,	[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,	Van Wyk 2 discloses a record search means utilizing a search key to access the linked list. Van Wyk 2 also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. For example, Van Wyk 2 discloses that "a successful search will examine only slots that contain items with the same hash value as that of the sought item; we need only examine other items when we seek space in which to store a new item." (Van Wyk 2 at 187).
[1c] the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed, and	Van Wyk 2 discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Van Wyk 2 also discloses the record search means including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Van Wyk 2 discloses that "[i]n the *reference count* scheme for garbage collection, each node contains a value that tells how many pointers point to it. When a node's reference count becomes zero, the node is garbage and can be collected. (Van Wyk 2 at 145).
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list.	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	Van Wyk 2 discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Van Wyk 2 also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records. For example, Van Wyk 2 discloses "[t]he *lazy* approach to garbage collection is to collect only in emergencies. Thus, when an allocation fails, we sweep

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination**
		through memory hoping to pick up and de-allocate enough garbage to permit the program to continue." (Van Wyk 2 at 146).
2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Van Wyk 2 discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Van Wyk 2 discloses that "[o]ur programs have never used `free`(). … [T]hey can leave dynamically allocated nodes unaccessibly lost in space. If memory space were scarce, however, we could revise them to free space explicitly when appropriate." (Van Wyk 2 at 136). Van Wyk 2 further states that "[a]ccess to the heap is through a single pointer, *rover*, which always points to the last node allocated or de-allocated. … As soon as *rover* points to a node with enough room, *malloc*() chops off a piece of the appropriate size and marks it occupied, adds the remainder of the node to the heap as a free node, and finally returns *rover* as a pointer to the newly allocated space." (Van Wyk 2 at 137). Thus, Van Wyk 2 inherently discloses means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. (*See id.*) Van Wyk 2 combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system,

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination**
		which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		system then sweeps a predetermined number of page table entries PT$_i$ on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Van Wyk 2 and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables

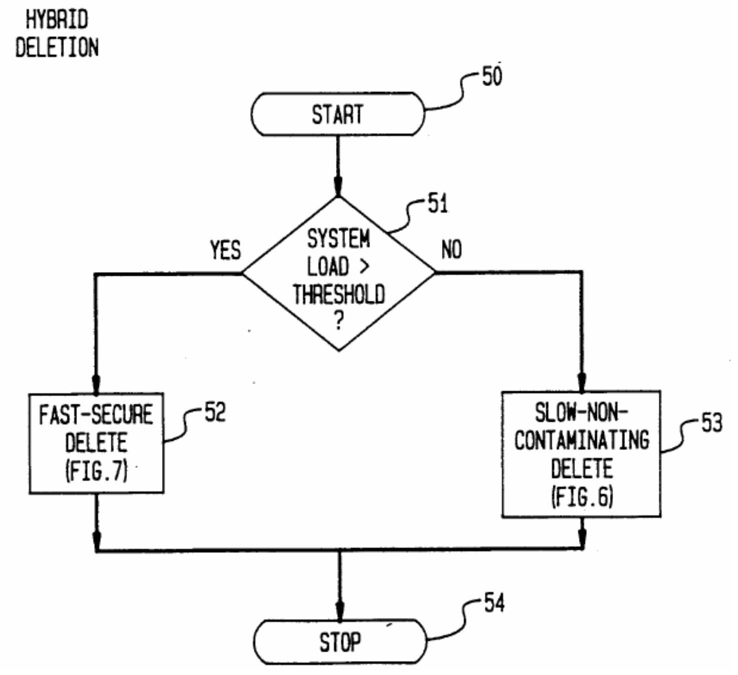
Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		implementations such as Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Van Wyk 2 nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Van Wyk 2 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Van Wyk 2 with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety.

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Van Wyk 2 with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Van Wyk 2 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Van Wyk 2 can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Van Wyk 2 with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk 2 with Thatte. Alternatively, it would also be obvious to combine Van Wyk 2with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent:

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		FIG.5  SYSTEM LOAD > THRESHOLD ? FAST-SECURE DELETE (FIG.7) SLOW-NON-CONTAMINATING DELETE (FIG.6) *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Van Wyk 2 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Van Wyk 2 would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination**
		By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Van Wyk 2 and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Van Wyk 2 with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Van Wyk 2 and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Van Wyk 2 would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Van Wyk 2 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Van Wyk 2 to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Van Wyk 2 with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Van Wyk 2can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk 2 with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles, in addition to motivations within the text of Van Wyk 2, "[a]ccess to the heap is through a single pointer, *rover*, which always points to the last node allocated or de-allocated. … As soon as *rover* points to a node with enough room, *malloc*() chops off a piece of the appropriate size and marks it occupied, adds the remainder of the node to the heap as a free node, and finally returns *rover* as a pointer to the newly allocated space." (Van Wyk 2 at 137). To the extent that dynamically determining a maximum number of expired records is not disclosed by Van Wyk 2 in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Van Wyk 2. For example, both Linux 2.0.1 and Van Wyk 2 describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ’g Co. & Bell Telephone Laboratories, Inc. 1988) (“Van Wyk 2”) alone and in combination
		When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring, the method comprising the steps of:	7. A method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	To the extent the preamble is a limitation, Van Wyk 2 discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Van Wyk 2 also discloses a method for storing and retrieving information records using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Van Wyk 2 discloses external data structures, explaining that "*File systems* are a familiar example of an external data structure." (Van Wyk 2 at 142). Van Wyk 2 also discloses that "[i]f the header nodes can be linked together, then no arbitrary limit on file size need be imposed." (Van Wyk 2 at 142). Therefore, Van Wyk 2 inherently discloses a linked list to store and provide access to records. Van Wyk 2 further discloses that "[i]n the *reference count* scheme for garbage collection, each node contains a value that tells how many pointers point to it. When a node's reference count becomes zero, the node is garbage and can be collected. (Van Wyk 2 at 145). Thus, Van Wyk 2 inherently discloses that at

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		least some of the records are automatically expiring. (*See id.*) Van Wyk 2 further discloses that "[h]ashing with linked lists is an excellent solution to many searching problems. At the price of some space for pointers, we obtain a table of potentially unlimited size that readily supports insertions and deletions." (Van Wyk 2 at 186).
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Van Wyk 2 discloses accessing a linked list of records. Van Wyk 2 also discloses accessing a linked list of records having same hash address. For example, Van Wyk 2 states that "[a]ccess to the heap is through a single pointer, *rover*, which always points to the last node allocated or de-allocated. … As soon as *rover* points to a node with enough room, *malloc*() chops off a piece of the appropriate size and marks it occupied, adds the remainder of the node to the heap as a free node, and finally returns *rover* as a pointer to the newly allocated space." (Van Wyk 2 at 137).
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Van Wyk 2 discloses identifying at least some of the automatically expired ones of the records. Van Wyk 2 also discloses identifying at least some of the automatically expired ones of the records. For example, Van Wyk 2 discloses that "[i]n the *reference count* scheme for garbage collection, each node contains a value that tells how many pointers point to it. When a node's reference count becomes zero, the node is garbage and can be collected. (Van Wyk 2 at 145).
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Van Wyk 2 discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Van Wyk 2 also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		For example, Van Wyk 2 discloses "[t]he *lazy* approach to garbage collection is to collect only in emergencies. Thus, when an allocation fails, we sweep through memory hoping to pick up and de-allocate enough garbage to permit the program to continue." (Van Wyk 2 at 146).
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Van Wyk 2 discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, Van Wyk 2 discloses that "we can delete an item from the dictionary using the straightforward algorithm to remove a node from a linked list." (Van Wyk 2 at 186).
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Van Wyk 2 discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, Van Wyk 2 discloses that "[o]ur programs have never used `free`(). … [T]hey can leave dynamically allocated nodes unaccessibly lost in space. If memory space were scarce, however, we could revise them to free space explicitly when appropriate." (Van Wyk 2 at 136). Van Wyk 2 further states that "[a]ccess to the heap is through a single pointer, *rover*, which always points to the last node allocated or de-allocated. … As soon as *rover* points to a node with enough room, *malloc*() chops off a piece of the appropriate size and marks it occupied, adds the remainder of the node to the heap as a free node, and finally returns *rover* as a pointer to the newly allocated space." (Van Wyk 2 at 137). Thus, Van Wyk 2 inherently discloses means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Van Wyk 2 combined with Dirks, Thatte, the '663 patent, and/or the

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done

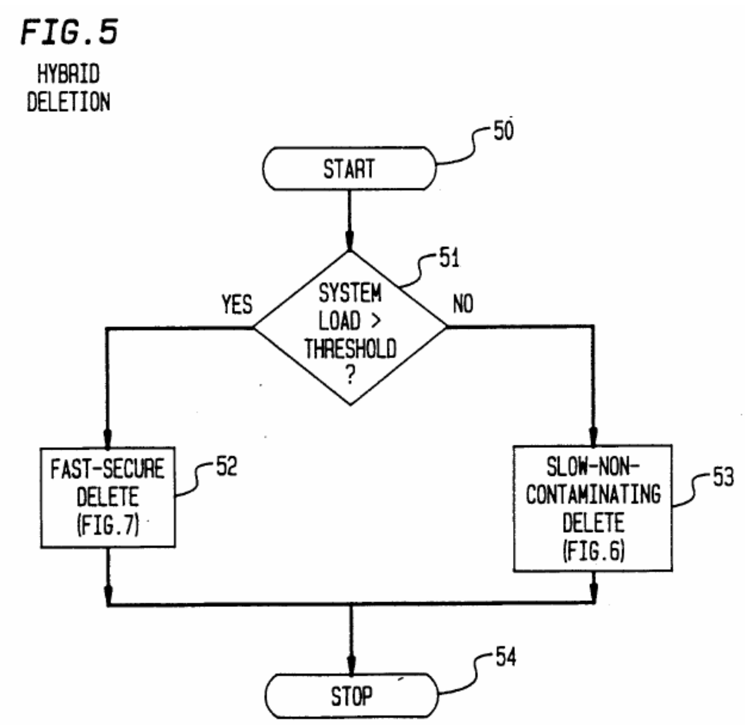
EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ’g Co. & Bell Telephone Laboratories, Inc. 1988) (“Van Wyk 2”) alone and in combination
		by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Van Wyk 2 and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Van Wyk 2 would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Van Wyk 2 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Van Wyk 2 with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ’g Co. & Bell Telephone Laboratories, Inc. 1988) (“Van Wyk 2”) alone and in combination
		example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Van Wyk 2 with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Van Wyk 2 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Van Wyk 2can be burdensome on the system, adding to the system’s load and slowing down the system’s processing. One of ordinary skill in the art would recognize that combining Van Wyk 2 with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk 2 with Thatte.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		Alternatively, it would also be obvious to combine Van Wyk 2 with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		This hybrid deletion is shown in Figure 5. FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64,

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Van Wyk 2 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Van Wyk 2 would be nothing more than the predictable use of prior art elements according to their established

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Van Wyk 2 and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Van Wyk 2 with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32.

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100. This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Van Wyk 2 and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Van Wyk 2. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Van Wyk 2 would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Van Wyk 2 and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Van Wyk 2 to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Van Wyk 2 with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Van Wyk 2 can be burdensome on

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Van Wyk 2 with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles in addition to motivations within the text of Van Wyk 2, "[a]ccess to the heap is through a single pointer, *rover*, which always points to the last node allocated or de-allocated. … As soon as *rover* points to a node with enough room, *malloc*() chops off a piece of the appropriate size and marks it occupied, adds the remainder of the node to the heap as a free node, and finally returns *rover* as a pointer to the newly allocated space." (Van Wyk 2 at 137). To the extent that dynamically determining a maximum number of expired records is not disclosed by Van Wyk 2 in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Van Wyk

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		2. For example, both Linux 2.0.1 and Van Wyk 2 describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function

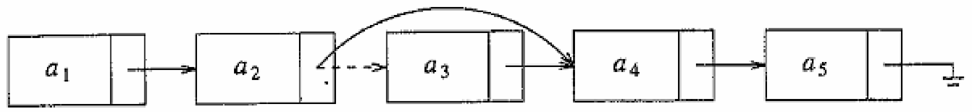
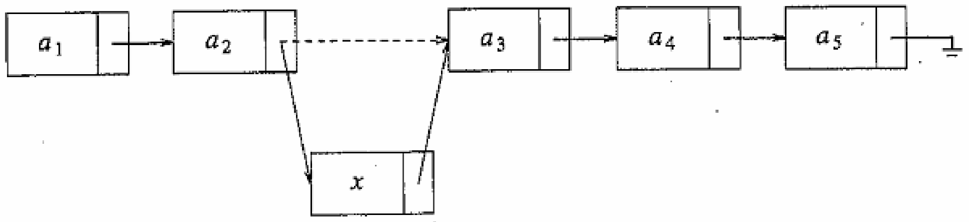
Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		rt_garbage_collect_1 determines again whether the number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold RT_CACHE_SIZE_MAX, the function rt_garbage_collect_1 halves the variable expire and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function rt_garbage_collect_1 can remove additional records from the linked lists in the hash table. The function rt_garbage_collect_1 repeats this process until the total number of records in the hash table is less than the predetermined threshold RT_CACHE_SIZE_MAX. Under Bedrock's proposed claim constructions, the records removed by the function rt_garbage_collect_1 are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function rt_cache_add only removes a record from a linked list when the record's last use time plus the fixed timeout value RT_CACHE_TIMEOUT is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function rt_cache_add can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function rt_garbage_collect_1 can remove from a given linked list is not

EXHIBIT C-10

Asserted Claims From U.S. Pat. No. 5,893,120		Christopher J. Van Wyk, Data Structures and C Programs (Addison-Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination
		limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	To the extent the preamble is a limitation, Weiss discloses an information storage and retrieval system. For example, Weiss discloses that a "linked list consists of a series of structures, which are not necessarily adjacent in memory. Each structure contains the element and a pointer to a structure containing its successor." Weiss at 43. Thus, Weiss inherently discloses an information storage and retrieval system. *See id.*
[1a] a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring,	[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring,	Weiss discloses a linked list to store and provide access to records stored in a memory of the system, at least some of the records automatically expiring. Weiss also discloses a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Weiss discloses that a "linked list consists of a series of structures, which are not necessarily adjacent in memory. Each structure contains the element and a pointer to a structure containing its successor." Weiss at 43. Weiss further discloses that "hashing is a technique used for performing insertions, deletions and finds in constant average time." Weiss at 149. Weiss discloses a method of collision resolution called Open Hashing (Separate Chaining) which "keeps a list of all elements that hash to the same value." Weiss at 152. "The hash table structure contains the actual size and an array of linked lists, which are dynamically allocated when the table is initialized. The HASH_TABLE type is just a pointer to this structure." Weiss at 153-54.
[1b] a record search means utilizing a search key to	[5b] a record search means utilizing a search key to	Weiss discloses a record search means utilizing a search key to access the linked list. Weiss also discloses a record search means utilizing a search key to

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
access the linked list,	access a linked list of records having the same hash address,	access a linked list of records having the same hash address. For example, Weiss discloses accessing by way of an insert command, "the insert command requires obtaining a new cell from the system by using an *malloc* call (more on this later) and then executing two pointer maneuvers. The general idea is shown in Figure 3.4. The dashed line represents the old pointer." Weiss at 44.  **Figure 3.3** Deletion from a linked list  **Figure 3.4** Insertion into a linked list Further, Weiss discloses, "[d]eciding what to do when two keys hash to the same value (this is known as a *collision*)." Weiss at 150. Weiss also discloses that "when inserting an element, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it." Weiss at 152.
[1c] the record search means including a means for identifying and	[5c] the record search means including means for identifying and removing	Weiss discloses the record search means including a means for identifying and removing at least some of the expired ones of the records from the linked list when the linked list is accessed. Weiss also discloses the record search means

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and	at least some expired ones of the records from the linked list of records when the linked list is accessed, and	including means for identifying and removing at least some expired ones of the records from the linked list of records when the linked list is accessed. For example, Weiss discloses that "[w]hen things are no longer needed, you can issue a *free* command to inform the system that it may reclaim the space. A consequence of the *free*(p) command is that the address that *p* is pointing to is unchanged, but the data that resides at that address is now undefined." Weiss at 50. To the extent that Bedrock argues that Weiss does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Weiss with Kruse. Kruse discloses "[t]he task of the procedure `Vivify` is to traverse the list `live`, determine whether each cell on it satisfies the conditions to become alive, and vivify it if so, else delete it from the list. The usual way to facilitate deletion from a linked list is to keep two pointers in lock step, one position apart, while traversing the list." … "Let us take advantage of the indirect linkage of our lists, and when we wish to delete an entry form the list, let us leave the node in place, but set its entry field to **nil**. In this way, the node will be flagged as empty when it is again encountered in the procedure `AddNeighbors`." Kruse at 219. Thus, Weiss and Kruse show that one of ordinary skill in the art understood how to identify and remove at least some expired ones of the records from the linked list of records when the linked list is accessed, and would recognize that it would improve similar systems and methods in the same way.
[1d] means, utilizing the record search means, for accessing the linked list and, at the same time,	[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the	Weiss discloses means, utilizing the record search means, for accessing the linked list and, at the same time, removing at least some of the expired ones of the records in the linked list. Weiss also discloses utilizing the record search means, for inserting, retrieving, and deleting records from the system and, at

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**

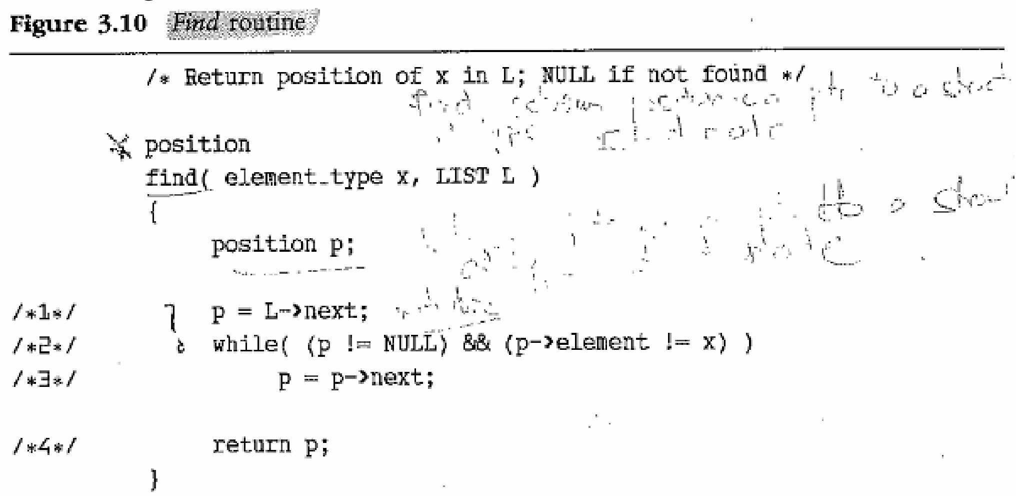
removing at least some of the expired ones of the records in the linked list.	system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.	the same time, removing at least some expired ones of the records in the accessed linked list of records.
		
		For example, Weiss discloses method of retrieving records in a find routine as shown in figure 3.10.
		
		Figure 3.10 *Find* routine
		`/* Return position of x in L; NULL if not found */`
		`position`
		`find( element_type x, LIST L )`
		`{`
		`    position p;`
		`/*1*/    p = L->next;`
		`/*2*/    while( (p != NULL) && (p->element != x) )`
		`/*3*/        p = p->next;`
		`/*4*/    return p;`
		`}`
		Weiss at 46.
		
		Weiss also discloses a method of inserting a record in an insert routine as shown in figure 3.12.

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination

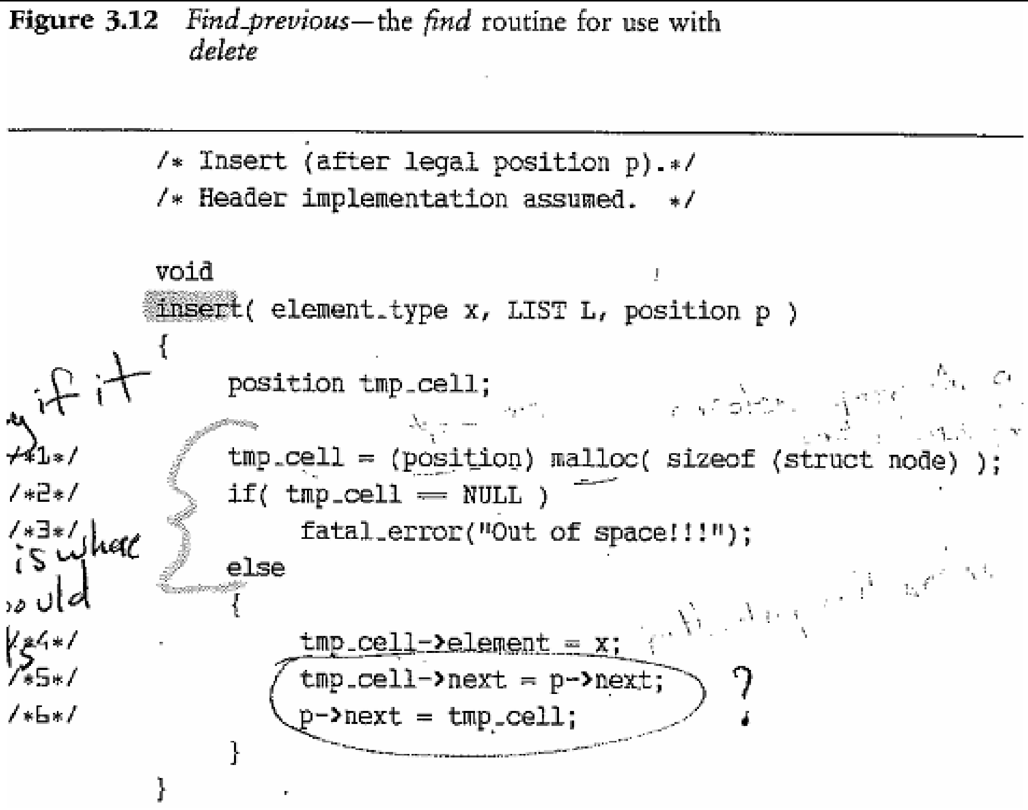
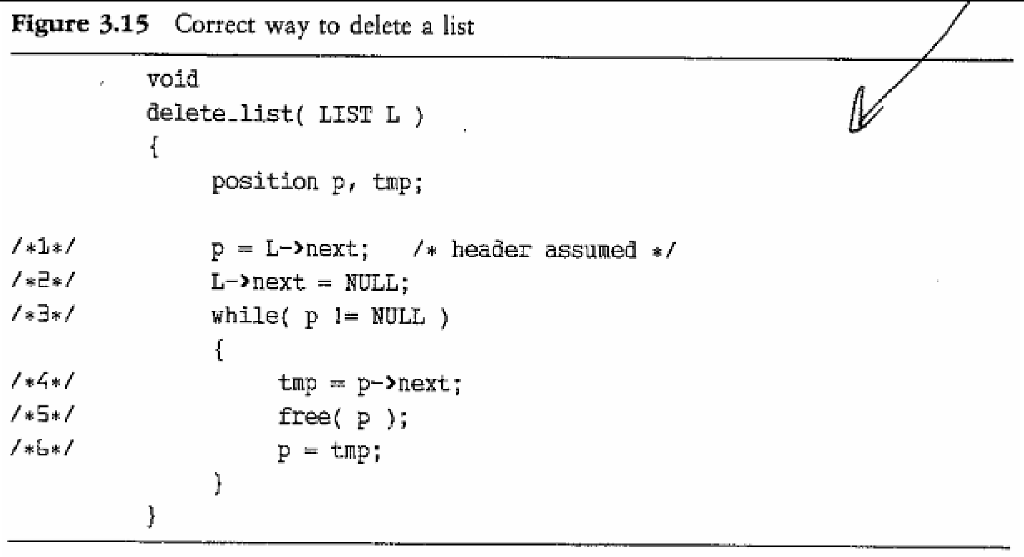
		 **Figure 3.12** *Find_previous*—the *find* routine for use with *delete*
		
		`/* Insert (after legal position p).*/`
		`/* Header implementation assumed.  */`
		
		`void`
		`insert( element_type x, LIST L, position p )`
		`{`
		`    position tmp_cell;`
		
		`    tmp_cell = (position) malloc( sizeof (struct node) );`
		`    if( tmp_cell == NULL )`
		`        fatal_error("Out of space!!!");`
		`    else`
		`    {`
		`        tmp_cell->element = x;`
		`        tmp_cell->next = p->next;`
		`        p->next = tmp_cell;`
		`    }`
		`}`
		
		if it
		is where
		uld
		?
		
		Weiss at 48.
		
		Weiss further discloses that a "deletion routine is a straightforward implementation of deletion in a linked list, so we will not bother with it here." Weiss at156. Weiss also discloses that "[a]fter a deletion in a linked list, it is usually a good idea to free the cell, especially if there are lots of insertions and deletions intermingled and memory might become a problem."

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination

		 **Figure 3.15** Correct way to delete a list
		`void`
		`delete_list( LIST L )`
		`{`
		`    position p, tmp;`
		`/*1*/    p = L->next;   /* header assumed */`
		`/*2*/    L->next = NULL;`
		`/*3*/    while( p != NULL )`
		`    {`
		`/*4*/        tmp = p->next;`
		`/*5*/        free( p );`
		`/*6*/        p = tmp;`
		`    }`
		`}`
		Weiss at 50.

2. The information storage and retrieval system according to claim 1 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	6. The information storage and retrieval system according to claim 5 further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.	Weiss discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. For example, Weiss discloses in Figure 5.8, an initialization routine for open hash table, and in Figure 5.1, an insert routine for open hash table, that can dynamically determine when there is a fatal error due to a lack of space. This determination inherently discloses dynamically determining maximum number of expired ones of the records to remove. Weiss at 154-56.

EXHIBIT C-11

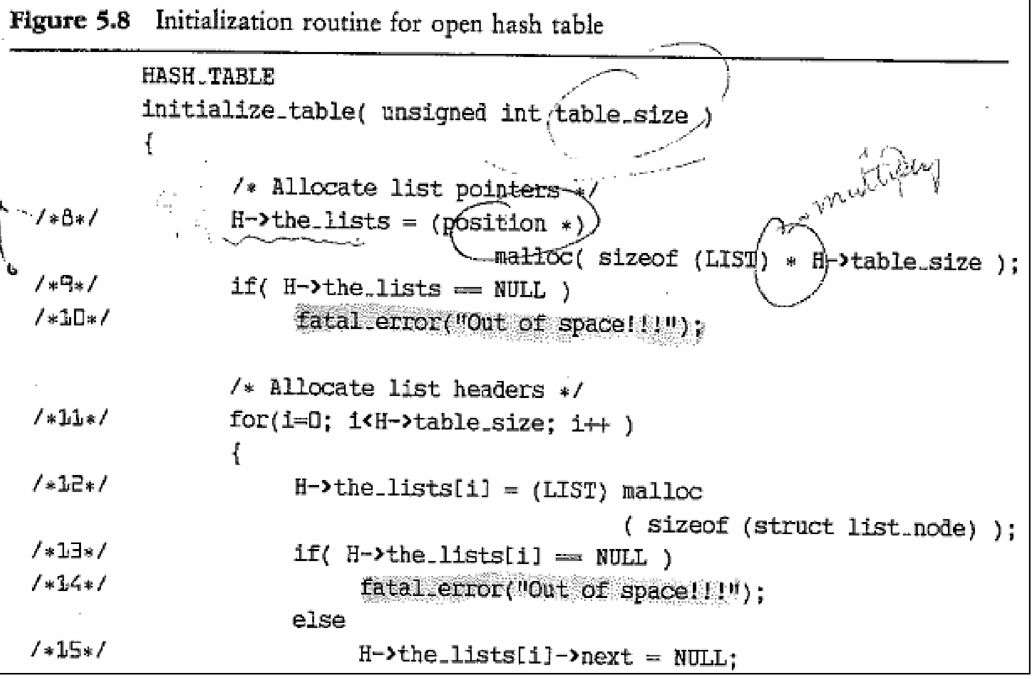
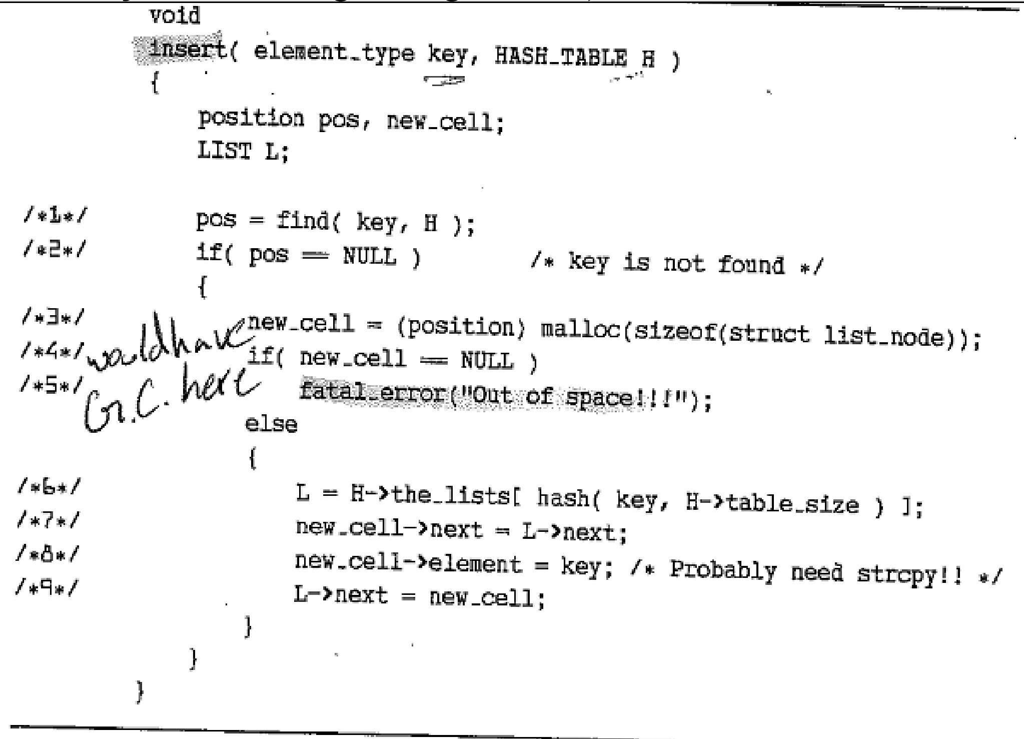
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		

Figure 5.8 Initialization routine for open hash table

```
        HASH_TABLE
        initialize_table( unsigned int table_size )
        {
                /* Allocate list pointers */
/*8*/           H->the_lists = (position *)
                                malloc( sizeof (LIST) * H->table_size );
/*9*/           if( H->the_lists == NULL )
/*10*/              fatal_error("Out of space!!!");

                /* Allocate list headers */
/*11*/          for(i=0; i<H->table_size; i++ )
                {
/*12*/              H->the_lists[i] = (LIST) malloc
                                        ( sizeof (struct list_node) );
/*13*/              if( H->the_lists[i] == NULL )
/*14*/                  fatal_error("Out of space!!!");
                    else
/*15*/                  H->the_lists[i]->next = NULL;
```

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		 Figure 5.10 *Insert* routine for open hash table It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Weiss to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art

```
        void
        insert( element_type key, HASH_TABLE H )
        {
            position pos, new_cell;
            LIST L;

/*1*/       pos = find( key, H );
/*2*/       if( pos == NULL )         /* key is not found */
            {
/*3*/           new_cell = (position) malloc(sizeof(struct list_node));
/*4*/           if( new_cell == NULL )
/*5*/               fatal_error("Out of space!!!");
                else
                {
/*6*/               L = H->the_lists[ hash( key, H->table_size ) ];
/*7*/               new_cell->next = L->next;
/*8*/               new_cell->element = key; /* Probably need strcpy!! */
/*9*/               L->next = new_cell;
                }
            }
        }
```

would have
G.C. here

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		would have been motivated to combine the system disclosed in Weiss with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Weiss can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Weiss is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Weiss combined with Dirks, Thatte, the '663 patent and/or the Opportunistic Garbage Collection Articles discloses an information storage and retrieval system further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in

EXHIBIT C-11

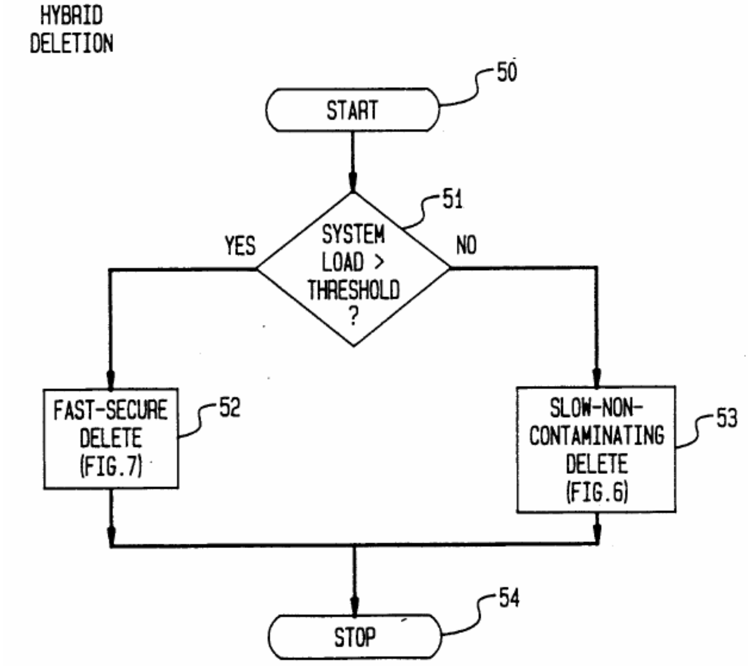
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		Exhibit B-2, which is hereby incorporated by reference in its entirety. For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ’g Co. 1993) alone and in combination**
		number of entries that are swept is identified as *k*, where: $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Weiss and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks’ dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the ’120 patent states “[a] person skilled in the art will appreciate that the technique of removing all expired records while searching

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Weiss nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Weiss and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.` Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Weiss with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Weiss with Thatte would be nothing more than the predictable use of prior art elements according to their established functions.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		The resulting combination would include the capability to determine the maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Weiss with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Weiss can be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Weiss with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Weiss with Thatte. Alternatively, it would also be obvious to combine Weisswith the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		FIG.5  HYBRID DELETION 50 START 51 SYSTEM LOAD > THRESHOLD ? YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 54 STOP *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33,

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Weiss and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as that described in Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Weiss would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		based on a systems load as taught by the '663 patent and with Weiss and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Weiss with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Weiss and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		procedure with Weiss would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Weiss and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Weiss to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Weiss with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Weisscan be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Weiss with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles, in addition to motivations within the text of Weiss, such as the initialization routine of Figure 5.8, and the insert routine of Figure 5.1 that can dynamically determine when there is a fatal error due to a lack of space. Weiss at 154-56. To the extent that dynamically determining a maximum number of expired records is not disclosed by Weiss in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Weiss. For example, both Linux 2.0.1 and Weiss describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the

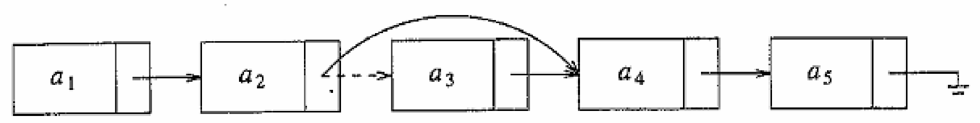
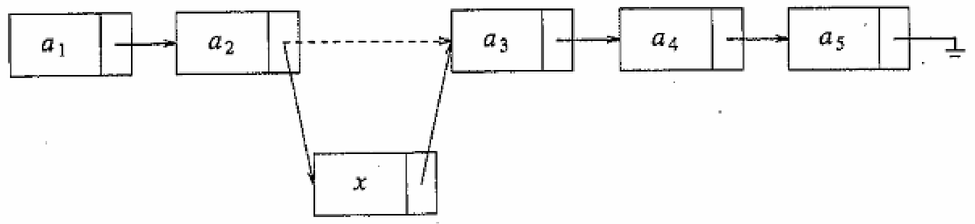
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293.

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.
3. A method for storing and retrieving information records using a linked list to store and provide access	7. A method for storing and retrieving information records using a hashing technique to provide access	To the extent the preamble is a limitation, Weiss discloses a method for storing and retrieving information records using a linked list to store and provide access to the records, at least some of the records automatically expiring. Weiss also discloses a method for storing and retrieving information records

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
to the records, at least some of the records automatically expiring, the method comprising the steps of:	to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring, the method comprising the steps of:	using a hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address, at least some of the records automatically expiring. For example, Weiss discloses that a "linked list consists of a series of structures, which are not necessarily adjacent in memory. Each structure contains the element and a pointer to a structure containing its successor." Weiss at 43. Thus, Weiss inherently discloses a method for storing and retrieving information records. *See id.* Weiss further discloses that "hashing is a technique used for performing insertions, deletions and finds in constant average time." Weiss at 149. Weiss discloses a method of collision resolution called Open Hashing (Separate Chaining) which "keeps a list of all elements that hash to the same value." Weiss at 152. "The hash table structure contains the actual size and an array of linked lists, which are dynamically allocated when the table is initialized. The HASH_TABLE type is just a pointer to this structure." Weiss at 153-54.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Weiss discloses accessing a linked list of records. Weiss also discloses accessing a linked list of records having same hash address. For example, Weiss discloses accessing by way of an insert command, "the insert command requires obtaining a new cell from the system by using an *malloc* call (more on this later) and then executing two pointer maneuvers. The general idea is shown in Figure 3.4. The dashed line represents the old pointer." Weiss at 44.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		 Figure 3.3 Deletion from a linked list  Figure 3.4 Insertion into a linked list Further, Weiss discloses, "[d]eciding what to do when two keys hash to the same value (this is known as a *collision*)." Weiss at 150. Weiss also discloses that "when inserting an element, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it." Weiss at 152.
[3b] identifying at least some of the automatically expired ones of the records, and	[7b] identifying at least some of the automatically expired ones of the records,	Weiss discloses identifying at least some of the automatically expired ones of the records. For example, Weiss discloses that "[w]hen things are no longer needed, you can issue a *free* command to inform the system that it may reclaim the space. A consequence of the *free*(p) command is that the address that *p* is pointing to is unchanged, but the data that resides at that address is now undefined." Weiss at 50. To the extent that Bedrock argues that Weiss does not anticipate this claim

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		element, it would have been obvious to one of ordinary skill in the art to combine Weiss with Kruse. Kruse discloses "[t]he task of the procedure `Vivify` is to traverse the list `live`, determine whether each cell on it satisfies the conditions to become alive, and vivify it if so, else delete it from the list. The usual way to facilitate deletion from a linked list is to keep two pointers in lock step, one position apart, while traversing the list." … "Let us take advantage of the indirect linkage of our lists, and when we wish to delete an entry form the list, let us leave the node in place, but set its entry field to **nil**. In this way, the node will be flagged as empty when it is again encountered in the procedure `AddNeighbors`." Kruse at 219. Thus, Weiss and Kruse show that one of ordinary skill in the art understood how to identify at least some of the automatically expired ones of the records, and would recognize that it would improve similar systems and methods in the same way.
[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.	[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and	Weiss discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Weiss discloses that "[w]hen things are no longer needed, you can issue a *free* command to inform the system that it may reclaim the space. A consequence of the *free*(p) command is that the address that *p* is pointing to is unchanged, but the data that resides at that address is now undefined." Weiss at 50. To the extent that Bedrock argues that Weiss does not anticipate this claim element, it would have been obvious to one of ordinary skill in the art to combine Weiss with Kruse. Kruse discloses "[t]he task of the procedure `Vivify` is to traverse the list `live`, determine whether each cell on it satisfies the conditions to become alive, and vivify it if so, else delete it from the list. The usual way to facilitate deletion from a linked list is to keep two pointers in

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		lock step, one position apart, while traversing the list." … "Let us take advantage of the indirect linkage of our lists, and when we wish to delete an entry form the list, let us leave the node in place, but set its entry field to **nil**. In this way, the node will be flagged as empty when it is again encountered in the procedure `AddNeighbors`." Kruse at 219. Thus, Weiss and Kruse show that one of ordinary skill in the art understood how to remove at least some of the automatically expired records from the linked list when the linked list is accessed, and would recognize that it would improve similar systems and methods in the same way.

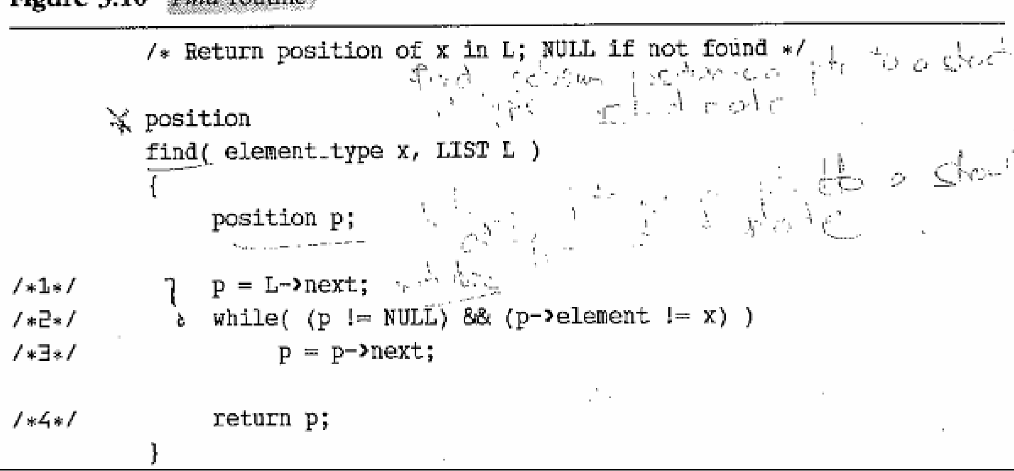
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	Weiss discloses inserting, retrieving or deleting one of the records from the system following the step of removing.
		
		For example, Weiss discloses method of retrieving records in a find routine as shown in figure 3.10.
		Figure 3.10 *Find* routine
		
		`/* Return position of x in L; NULL if not found */`
		`position`
		`find( element_type x, LIST L )`
		`{`
		`    position p;`
		`/*1*/    p = L->next;`
		`/*2*/    while( (p != NULL) && (p->element != x) )`
		`/*3*/        p = p->next;`
		`/*4*/    return p;`
		`}`

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination

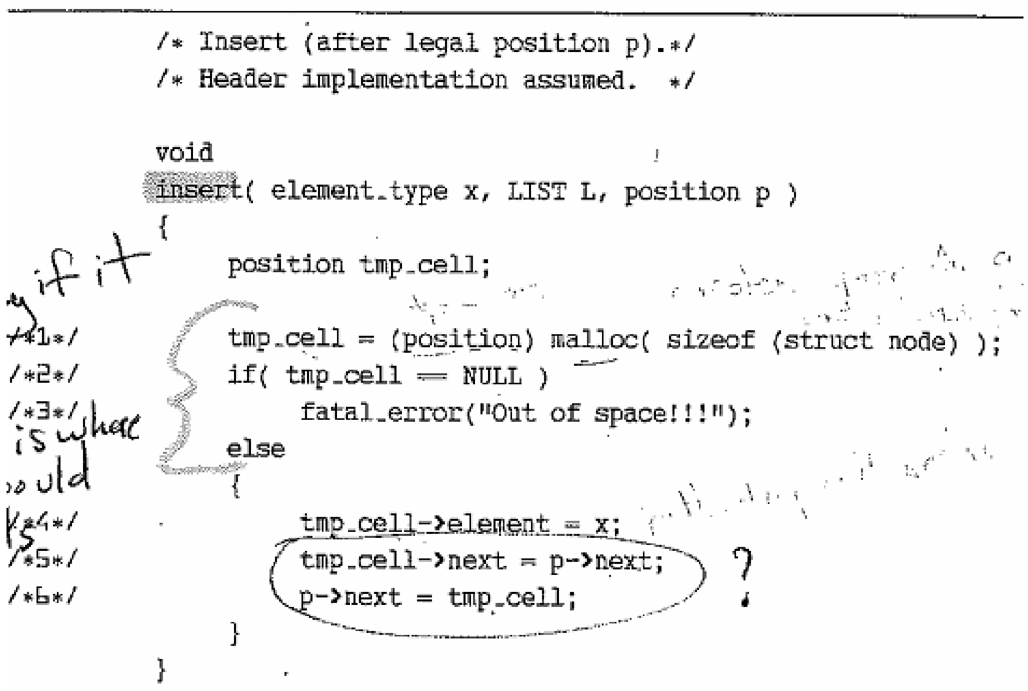
		Weiss at 46.
		
		Weiss also discloses a method of inserting a record in an insert routine as shown in figure 3.12.
		Figure 3.12 *Find_previous*—the *find* routine for use with *delete*
		
		`/* Insert (after legal position p).*/`
		`/* Header implementation assumed.  */`
		
		`void`
		`insert( element_type x, LIST L, position p )`
		`{`
		`    position tmp_cell;`
		
		`/*1*/    tmp_cell = (position) malloc( sizeof (struct node) );`
		`/*2*/    if( tmp_cell == NULL )`
		`/*3*/        fatal_error("Out of space!!!");`
		`         else`
		`         {`
		`/*4*/        tmp_cell->element = x;`
		`/*5*/        tmp_cell->next = p->next;`
		`/*6*/        p->next = tmp_cell;`
		`         }`
		`}`
		Weiss at 48.
		
		Weiss further discloses that a "deletion routine is a straightforward

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		implementation of deletion in a linked list, so we will not bother with it here." Weiss at156. Weiss also discloses that "[a]fter a deletion in a linked list, it is usually a good idea to free the cell, especially if there are lots of insertions and deletions intermingled and memory might become a problem."

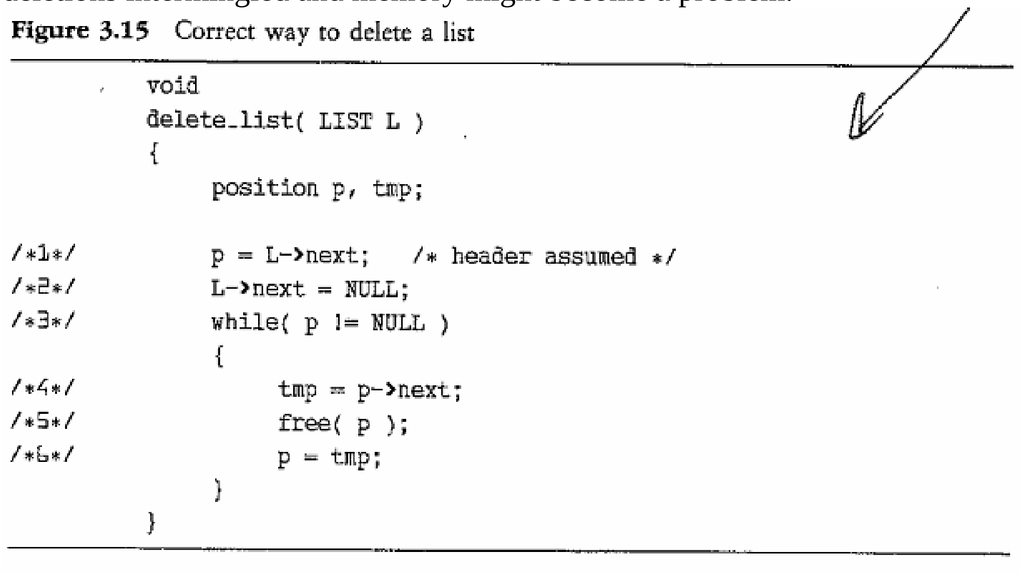


Figure 3.15 Correct way to delete a list

```
        void
        delete_list( LIST L )
        {
            position p, tmp;

            p = L->next;   /* header assumed */
            L->next = NULL;
            while( p != NULL )
            {
                tmp = p->next;
                free( p );
                p = tmp;
            }
        }
```

Weiss at 50.

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
4. The method according to claim 3 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	8. The method according to claim 7 further including the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.	Weiss discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, Weiss discloses in Figure 5.8, an initialization routine for open hash table, and in Figure 5.1, an insert routine for open hash table, that can dynamically determine when there is a fatal error due to a lack of space. This determination inherently discloses dynamically determining maximum number of expired ones of the records to remove. Weiss at 154-56.

EXHIBIT C-11

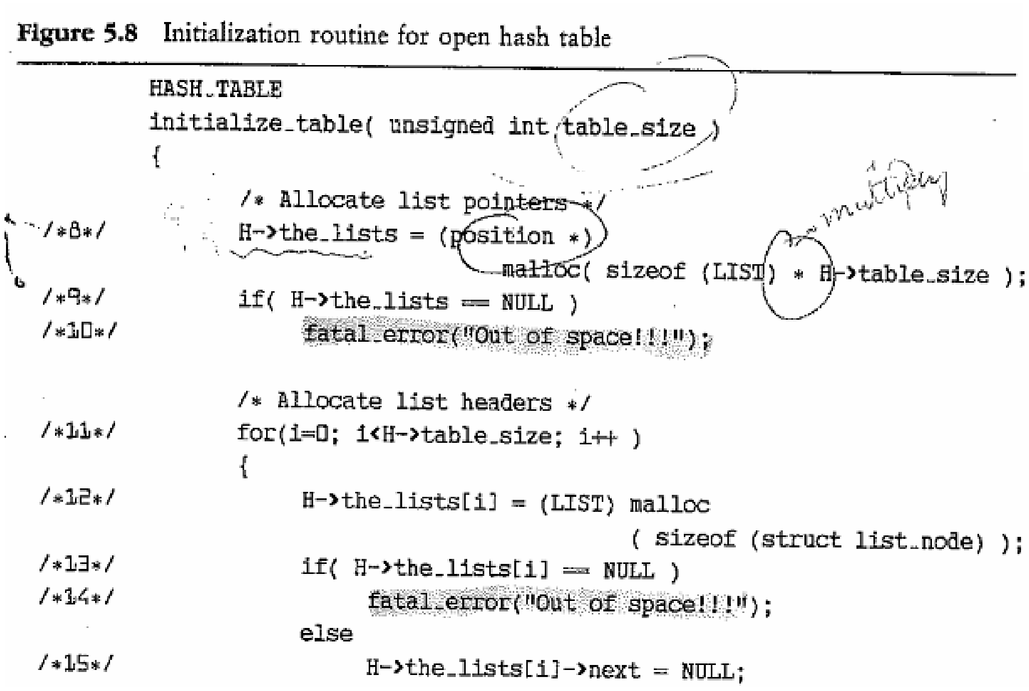
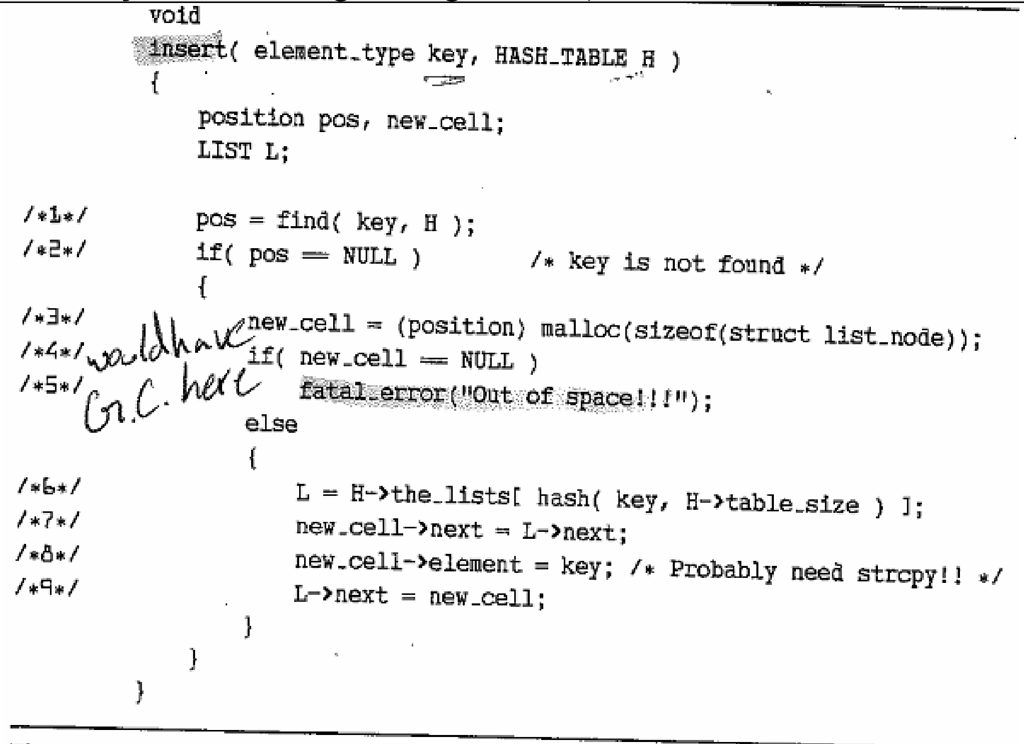
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		

Figure 5.8 Initialization routine for open hash table

```
        HASH_TABLE
        initialize_table( unsigned int table_size )
        {
              /* Allocate list pointers */
/*8*/         H->the_lists = (position *)
                          malloc( sizeof (LIST) * H->table_size );
/*9*/         if( H->the_lists == NULL )
/*10*/              fatal_error("Out of space!!!");

              /* Allocate list headers */
/*11*/        for(i=0; i<H->table_size; i++ )
              {
/*12*/              H->the_lists[i] = (LIST) malloc
                                  ( sizeof (struct list_node) );
/*13*/              if( H->the_lists[i] == NULL )
/*14*/                    fatal_error("Out of space!!!");
                    else
/*15*/                    H->the_lists[i]->next = NULL;
```

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		

```
        void
        insert( element_type key, HASH_TABLE H )
        {
            position pos, new_cell;
            LIST L;

/*1*/       pos = find( key, H );
/*2*/       if( pos == NULL )       /* key is not found */
            {
/*3*/           new_cell = (position) malloc(sizeof(struct list_node));
/*4*/           if( new_cell == NULL )
/*5*/               fatal_error("Out of space!!!");
                else
                {
/*6*/               L = H->the_lists[ hash( key, H->table_size ) ];
/*7*/               new_cell->next = L->next;
/*8*/               new_cell->element = key; /* Probably need strcpy!! */
/*9*/               L->next = new_cell;
                }
            }
        }
```

would have G.C. here

Figure 5.10 *Insert* routine for open hash table

It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Weiss to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		would have been motivated to combine the system disclosed in Weiss with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Weiss can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. Indeed, part of the motivation for the system disclosed in Weiss is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Weiss combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system, which dynamically determines how many records to sweep/remove upon each allocation. Disclosure of these claim elements in Dirks is clearly shown in the chart of Dirks, which is hereby incorporated by reference in its entirety.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		For example, as summarized in Dirks, each time a VSID is assigned from the free list to a new application or thread, a fixed number of entries in the page table are scanned to determine whether they have become inactive, by checking them against the VSIDs on the recycle list. Each entry which is identified as being inactive is removed from the page table. After all of the entries in the page table have been examined in this manner, the VSIDs in the recycle list can be transferred to the free list, since all of their associated page table entries will have been removed. This approach thereby guarantees that a predetermined number of VSIDs are always available in the free list without requiring a time-consuming scan of the complete page table at once. U.S. Patent No. 6,119,214 to Dirks at 7:2-14. After [a] new VSID has been allocated, the system checks a flag RFLG to determine whether a recycle sweep is currently in progress (Step 20). If there is no sweep in progress, i.e. RFLG is not equal to one, a determination is made whether a sweep should be initiated. This is done by checking whether the inactive list is full, i.e. whether it contains x entries (Step 22). If the number of entries I on the inactive list is less than x, no further action is taken, and processing control returns to the operating system (Step 24). If, however, the inactive list is full at this time, the flag RFLG is set (Step 26), the VSIDs on the inactive list are transferred to the recycle list, and an index n is reset to 1 (Step 28). The system then sweeps a predetermined number of page table entries PT_i on the page table, to detect whether any of them are inactive, i.e. their associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as *k*, where:

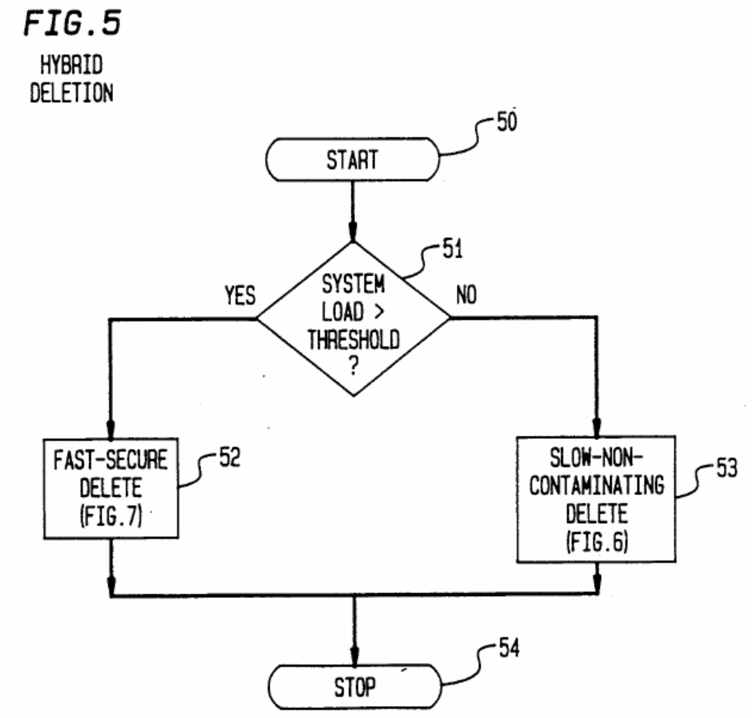
Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		$$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$$ *Id.* at 8:12-30. Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. *Id.* at 7:37-40. As stated in Dirks: Any other suitable approach can be employed to determine the number of entries to be examined during each step of the sweeping process. In this regard, it is not necessary that the number of examined entries be fixed for each step. Rather, it might vary from one step to the next. The only criterion is that the number of entries examined on each step be such that all entries in the page table are examined in a determinable amount of time or by the occurrence of a certain event, e.g. by the time the list of free VSIDs is empty. *Id.* at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value *k*. *Id.* at 7:15-46, 7:66-8:56. As both Weiss and Dirks relate to deletion of aged records upon the allocation of a new incoming record, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining Dirks' deletion decision procedure with Weiss would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined Dirks' dynamic determination of the suitable number of entries to examine during each step of the sweeping process with Weiss and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps. Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Weiss with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by Thatte. For example, Thatte discloses a system and method using hash tables and/or linked lists and further discloses means for dynamically determining the maximum number for the record search means to remove in the accessed linked list of records. The disclosure of these claim elements in Thatte is clearly shown in the chart of Thatte, which is hereby incorporated by reference in its entirety. Moreover, one of ordinary skill in the art would recognize that these combinations would improve the similar systems and methods in the same way. Additionally, one of ordinary skill in the art would recognize that the result of combining Weiss with Thatte would be nothing more than the predictable use of prior art elements according to their established functions. The resulting combination would include the capability to determine the

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		maximum number for the record search means to remove. Further, one of ordinary skill in the art would be motivated to combine Weiss with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Weisscan be burdensome on the system, adding to the system's load and slowing down the system's processing. One of ordinary skill in the art would recognize that combining Weiss with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other things, the system load. Moreover, the '120 patent discloses that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Weiss with Thatte. Alternatively, it would also be obvious to combine Weiss with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. For example, as summarized in the '663 patent: during normal times when the load on the storage system is not excessive, a non-contaminating but slow deletion of records is used. This slow, non-contaminating deletion involves closing the collision-resolution chain of locations by moving a record from a later position in the chain into the position of the record to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		4,996,663 to Nemes at 2:24-34 ("The '663 patent"). In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply marked as "deleted" and left in place. Later non-contaminating probes in the vicinity of such deleted record locations automatically remove the contaminating deleted records by moving records in the chain as described above. *Id.* at 2:35-41. This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. *Id.* at 2:42-46. This hybrid deletion is shown in Figure 5.

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		FIG.5  HYBRID DELETION START 50 SYSTEM LOAD > THRESHOLD ? 51 YES NO FAST-SECURE DELETE (FIG.7) 52 SLOW-NON-CONTAMINATING DELETE (FIG.6) 53 STOP 54 *Id.* at Figure 5. During the hybrid deletion procedure decision block 51 checks the system load to determine if the system load is greater than a threshold. If the system load is greater than the threshold, then a fast-secure delete 52 is used. *Id.* at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. *Id.* The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. *Id.* at 8:1-33,

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. *See id.* at 6:40-64, Figure 5. If the fast-secure delete 52 is used, then maximum number of records is zero because records are not deleted they are only marked. *Id.* at 8:1-33, Figure 7. If the slow-non-contaminating delete 53 is used, then the maximum number of records to remove is all of the contaminated records in the bucket. *Id.* at 6:65-7:68, Figures 6, 6A, 6B. As both Weiss and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understood how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other hash table implementations such as Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the '663 patent's deletion decision procedure with Weiss would be nothing more than the predictable use of prior art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the '663 patent's dynamic decision on whether to perform a deletion based on a systems load as taught by the '663 patent and with Weiss and would

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		have seen the benefits of doing so. One such benefit, for example, is that the system would avoid performing deletions when the system load exceeded a threshold. Alternatively, it would also be obvious to combine Weiss with the Opportunistic Garbage Collection Articles. The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. *See generally*, Paul R. Wilson and Thomas G. Moher, *Design of the Opportunistic Garbage Collector*, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, *Opportunistic Garbage Collection*, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988. For example, the Opportunistic Garbage Collection Articles disclose in part: When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to scavenge. The fuller a generation is, the more likely it is to be scavenged; also, the longer the pause that has been detected, the larger the scope of the garbage collection is likely to be. *Design of the Opportunistic Garbage Collector* at 32. Every time a user-input routine is invoked, a decision routine can decide whether to garbage collect. As long as the decision routine takes no more than a few milliseconds to execute, it should not interfere with responsiveness. Since it is only invoked at these times, it does not incur a continual run-time overhead. *Opportunistic Garbage Collection* at 100.

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		This decision routine should take several things into account: 1) the volume of data allocated since the last scavenge, 2) how long it has been since the user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenge. If the product of the allocation and the compute time is high, and if the stack is low, the scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. *Id.* If these heuristics fail and a scavenge is forced instead by the filling of a generation's space, it is likely to happen during a significant compute-bound pause--the one that has just allocated the data that forced the collection. When the opportunistic mechanism fails to find the end of a pause, it may still succeed by default, embedding a scavenge pause within a larger pause. *Design of the Opportunistic Garbage Collector* at 32. As both Weiss and the Opportunistic Garbage Collection Articles relate to deletion of aged records, one of ordinary skill in the art would have understood how to use the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion based on a system load in other hash table implementations such as Weiss. Moreover, one of ordinary skill in the art would recognize that it would improve similar systems and methods in the same way. As the '120 patent states "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." The '120 patent at 7:10-15. Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles' deletion decision procedure with Weiss would be nothing more than the predictable use of prior

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		art elements according to their established functions. By way of further example, one of ordinary skill in the art would have combined the Opportunistic Garbage Collection Articles' dynamic decision on whether to perform a deletion and how many generations to scavenge as taught by the Opportunistic Garbage Collection Articles and with Weiss and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system. Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Weiss to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a fundamental concept in computer science and the relevant art that any variable or parameter affecting any aspect of a system can be dynamically determined based on information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Weiss with the fundamental concept of dynamically determining the maximum number of expired records to remove in an accessed linked list of records to solve a number of potential problems. For example, the removal of expired records described in Weiss can be burdensome on the system, adding to the system's load and slowing down the system's processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete. One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the system and bound the length of any real-time interruption to prevent delays in processing. Indeed, Nemes concedes that such dynamic determination was

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		obvious when he states in the '120 patent that "[a] person skilled in the art will appreciate that the technique of removing all expired records while searching the linked list can be expanded to include techniques whereby not necessarily all expired records are removed, and that the decision regarding if and how many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Weiss with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles in addition to motivations within the text of Weiss, such as the initialization routine of Figure 5.8, and the insert routine of Figure 5.1 that can dynamically determine when there is a fatal error due to a lack of space. Weiss at 154-56. To the extent that dynamically determining a maximum number of expired records is not disclosed by Weiss in combination with Dirks, Thatte, the '663 Patent, or the Opportunistic Garbage Collection References, it is disclosed by Linux 2.0.1, which describes dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. It would have been obvious to combine Linux 2.0.1 with Weiss. For example, both Linux 2.0.1 and Weiss describe systems and methods for performing data storage and retrieval using known programming techniques to yield a predictable result. When invoked, the function `rt_cache_add` automatically increments an integer variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1359. When the function `rt_cache_add` removes an expired record, the function `rt_cache_add` decrements the variable `rt_cache_size`. *See* Linux 2.0.1, route.c at line 1373. Thus, the variable `rt_cache_size` indicates the number of records in the hash table (i.e., `ip_rt_hash_table`). Because the function `rt_cache_add` automatically increments and decrements the

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		variable `rt_cache_size`, the variable `rt_cache_size` is determined dynamically. Furthermore, LINUX 2.0.1 includes the function `rt_garbage_collect_1`. The function `rt_garbage_collect_1` loops through each of the linked lists in the `ip_rt_hash_table` global variable. *See* Linux 2.0.1, route.c at lines 1122-1138. In this way, the function `rt_garbage_collect_1` accesses the linked list. When the function `rt_garbage_collect_1` identifies a record that is expired, the function `rt_garbage_collect_1` decrements the variable `rt_cache_size` and frees the record. *See* Linux 2.0.1, route.c at lines 1128-1135. Because all records in the linked list can be expired and all records in the hash table can be in the linked list, the variable `rt_cache_size` can represent a dynamically determined maximum number of expired ones of the records to remove when function `rt_garbage_collect_1` accesses the linked list. Furthermore, the function `rt_cache_add` determines whether the number of records in the hash table exceeds a predetermined threshold `RT_CACHE_SIZE_MAX`. If the number of records in the hash table exceeds the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_cache_add` invokes a function `rt_garbage_collect`. *See* Linux 2.0.1, route.c at lines 1341-1342. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293. The function `rt_garbage_collect` invokes a function `rt_garbage_collect_1`. *See* Linux 2.0.1, route.c at line 1293.

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		The function `rt_garbage_collect_1` loops through each linked list in the hash table. *See* Linux 2.0.1, route.c at lines 1116-1132. For each linked list in the hash table, the function `rt_garbage_collect_1` looks at each record in the linked list. *See* Linux 2.0.1, route.c at lines 1120-1131. For each record in a linked list, the function `rt_garbage_collect_1` determines whether the record's last use time plus the record's expiration factor is later than the current time. *See* Linux 2.0.1, route.c at line 1122. If the record's last use time plus the record's expiration factor is less than the current time, the function `rt_garbage_collect_1` removes the record from the linked list. *See* Linux 2.0.1, route.c at lines 1124-1130. The record's expiration factor is based on a variable `expire` and the record's reference count. *See* Linux 2.0.1, route.c at line 1122. The variable expire is initially one half of the fixed timeout value `RT_CACHE_TIMEOUT`. *See* Linux 2.0.1, route.c at line 1110. After looping through all of the linked lists in this manner, the function `rt_garbage_collect_1` determines again whether the number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. *See* Linux 2.0.1, route.c at line 1133. If the number of items in the hash table is still greater than the predetermined threshold `RT_CACHE_SIZE_MAX`, the function `rt_garbage_collect_1` halves the variable `expire` and loops through each of the linked lists in the hash table. *See* Linux 2.0.1, route.c at line 1135. In this way, the function `rt_garbage_collect_1` can remove additional records from the linked lists in the hash table. The function `rt_garbage_collect_1` repeats this process until the total number of records in the hash table is less than the predetermined threshold `RT_CACHE_SIZE_MAX`. Under Bedrock's proposed claim constructions, the records removed by the

Asserted Claims From U.S. Pat. No. 5,893,120		**Mark Allen Weiss, *Data Structures & Algorithm Analysis in C* (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination**
		function `rt_garbage_collect_1` are "expired" records. That is, the records removed by the function rt_garbage_collect_1 are data items which after a limited time or after the occurrence of some event become obsolete, such that their presence in the storage system is no longer needed or desired. The function `rt_cache_add` only removes a record from a linked list when the record's last use time plus the fixed timeout value `RT_CACHE_TIMEOUT` is less than the current time and the record's reference count is zero. *See* Linux 2.0.1, route.c at line 1369. Thus, the maximum number of records that the function `rt_cache_add` can remove from a given linked list is limited to those records whose reference counts are zero. In contrast, the maximum number of records that the function `rt_garbage_collect_1` can remove from a given linked list is not limited to those records whose reference counts are zero. *See* Linux 2.0.1, route.c at line 1122. Rather, the function `rt_garbage_collect_1` can remove records whose reference counts are zero and records whose reference counts are greater than zero. Consequently, the maximum number of records that the function `rt_garbage_collect_1` can remove from a linked list is different than the maximum number of records that the function `rt_cache_add` can remove from a linked list.