

Exhibit A

Part 2

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Dietzfelbinger discloses an information storage and retrieval system.</p> <p>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 <i>randomized</i> 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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990, at 1.</p> <p>“A <i>dictionary</i> over a universe $U = \{0, 1, \dots, N - 1\}$ 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.” <i>Id.</i> at 2.</p>
<p>[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,</p>	<p>[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</p>	<p>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.</p> <p>For example, Dietzfelbinger discloses that “[t]he dynamic dictionary problem</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").
	automatically expiring,	<p>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 <i>randomized</i> 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" <i>Id.</i> at 1.</p> <p>"A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ 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." <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that "[t]here are two major techniques for implementing dictionaries: trees and hashing." <i>Id.</i></p> <p>Dietzfelbinger discloses an extension of the Fredman, Kómlós, 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." <i>Id.</i></p> <p>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.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>Specifically, a function h chosen uniformly at random from H_s is used to partition the set S into s blocks.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus,</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>Dietzfelbinger 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.</p> <p>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.” <i>Id.</i> at 5.</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>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.</p> <p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p> <p>“A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ is a partial function S from U to some set I. The operations <i>Lookup</i>(x), <i>Insert</i>(x, i), and <i>Delete</i>(x) are available on a dictionary S; <i>Lookup</i>(x) returns $S(x)$, <i>Insert</i>(x, i) adds x to the domain of S and sets $S(x)$ to I, and <i>Delete</i>(x) removes x from the domain of S.” <i>Id.</i> at 2.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>this kind the best upper bound known on the expected <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$....” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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.</p>
<p>[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</p>	<p>[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,</p>	<p>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.</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
<p>accessed, and</p>	<p>and</p>	<p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p> <p>“A <i>dictionary</i> over a universe $U = \{0, 1, \dots, N - 1\}$ 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.” <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>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.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>common place to resolve collisions within hash tables. <i>Id.</i> Thus, Dietzfelbinger 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.</p> <p>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.” <i>Id.</i> at 5.</p> <p>A call to the procedure <i>Insert(x)</i> 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.” <i>Id.</i> at 6.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").</p>
<p>Plaintiff's Invalidation Contentions & Production of Documents</p> <p>US2008 1290297.1</p>	<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked "deleted" then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked "deleted" into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>; else (* <i>T_j</i> is too small *) <i>m_j</i> ← 2 · max{1, <i>m_j</i>}; <i>s_j</i> ← 2<i>m_j</i>(<i>m_j</i> - 1); if condition (**) is still satisfied then (* double capacity of <i>T_j</i> *) allocate new space, namely <i>s_j</i> cells, for new subtable <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
<p>[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.</p>	<p>[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.</p>	<p>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.</p> <p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p> <p>“A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ is a partial function S from U to some set I. The operations <i>Lookup</i>(x), <i>Insert</i>(x, i), and <i>Delete</i>(x) are available on a dictionary S; <i>Lookup</i>(x) returns $S(x)$, <i>Insert</i>(x, i) adds x to the domain of S and sets $S(x)$ to I, and <i>Delete</i>(x) removes x from the domain of S.” <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").
		<p>scheme, that FKS also uses hashing with chaining.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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.</p> <p>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.” <i>Id.</i> at 5.</p> <p>A call to the procedure <i>Insert(x)</i> 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.” <i>Id.</i> at 6.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>	
		<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked “deleted” then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked “deleted” into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").</p>
		<pre> else (* T_j is too small *) m_j ← 2 · max{1, m_j}; s_j ← 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 ← length of L_j; repeat h_j ← randomly chosen function in 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); </pre> <p align="right"><i>Id.</i> at Figure 1.</p>
<p>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</p>	<p>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</p>	<p>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.</p> <p>For example, a call to the procedure <i>Insert(x)</i> conducts the following steps, as displayed by the code in the figure below. The procedure hashes <i>x</i> to determine the proper position for <i>x</i>, <i>h_j(x)</i>. First, the procedure determines if <i>x</i> already exists in the sub table. If <i>x</i> does exist, then the only determination to</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i> , Revised Version January 7, 1990 ("Dietzfelbinger").
linked list of records.	linked list of records.	<p>make is whether x is marked for deletion. If x is marked for deletion, then the procedure removes this deletion tag. <i>Id.</i> at 6.</p> <p>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. <i>Id.</i></p> <p>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." <i>Id.</i></p> <p>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 <i>Insert(x)</i> only if the above two conditions are not met.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>	
		<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked “deleted” then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked “deleted” into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<pre> else (* T_j is too small *) m_j ← 2 · max{1, m_j}; s_j ← 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 ← length of L_j; repeat h_j ← randomly chosen function in 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); </pre> <p><i>Id.</i> at Figure 1.</p> <p>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</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>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.</p>
<p>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:</p>	<p>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</p>	<p>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.</p> <p>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</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>steps of:</p> <p>given: a <i>randomized</i> 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.</p> <p>“A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ is a partial function S from U to some set I. The operations <i>Lookup</i>(x), <i>Insert</i>(x, i), and <i>Delete</i>(x) are available on a dictionary S; <i>Lookup</i>(x) returns $S(x)$, <i>Insert</i> (x, i) adds x to the domain of S and sets $S(x)$ to I, and <i>Delete</i>(x) removes x from the domain of S.” <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>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 <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>resolve collisions within hash tables, and would recognize that it would improve similar systems and methods in the same way.</p> <p>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.” <i>Id.</i> at 5.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Dietzfelbinger discloses accessing a linked list of records. Dietzfelbinger also discloses accessing a linked list of records having same hash address.</p> <p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p> <p>“A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ is a partial function S from U to some set I. The operations <i>Lookup</i>(x), <i>Insert</i>(x, i), and <i>Delete</i>(x) are available on a dictionary S; <i>Lookup</i>(x) returns $S(x)$, <i>Insert</i>(x, i) adds x to the domain of S and sets $S(x)$ to I, and <i>Delete</i>(x) removes x from the domain of S.” <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> at 2-3. Since Dietzfelbinger does not</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Dietzfelbinger discloses identifying at least some of the automatically expired ones of the records.</p> <p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<p>“A dictionary over a universe $U = \{0, 1, \dots, N - 1\}$ 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.” <i>Id.</i> at 2.</p> <p>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.” <i>Id.</i> at 5.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Dietzfelbinger discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>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 <i>randomized</i> 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” <i>Id.</i> at 1.</p> <p>“A dictionary over a universe $U = \{0, 1, \dots, N - 1\}$ 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.” <i>Id.</i> at 2.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p> <p>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 <i>universal hashing</i> 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</p>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>this kind the best upper bound known on the expected <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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.</p> <p>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.” <i>Id.</i> at 5.</p> <p>A call to the procedure <i>Insert(x)</i> conducts the following steps, as displayed by the code in the figure below. The procedure hashes x to determine the proper</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>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.” <i>Id.</i> at 6.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked “deleted” then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked “deleted” into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").</p>
		<pre> else (* T_j is too small *) m_j ← 2 · max{1, m_j}; s_j ← 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 ← length of L_j; repeat h_j ← randomly chosen function in 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); </pre> <p align="right"><i>Id.</i> at Figure 1.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Dietzfelbinger discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>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 <i>randomized</i> 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" <i>Id.</i> at 1.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").
		<p>“A <i>dictionary</i> over a <i>universe</i> $U = \{0, 1, \dots, N - 1\}$ 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.” <i>Id.</i> at 2.</p> <p>Furthermore, Dietzfelbinger discloses that “[t]here are two major techniques for implementing dictionaries: trees and hashing.” <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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.” <i>Id.</i> at 3-4.</p> <p>While Dietzfelbinger does not disclose whether or not the FKS scheme uses hashing with chaining to handle the unavoidable key collisions problem,</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).
		<p>Dietzfelbinger discloses other schemes that do use hashing with chaining. “Carter and Wegman [CW79] proposed <i>universal hashing</i> 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 <i>worst case time</i> for an instruction (i.e., the length of the longest chain in the resulting hash table with chaining) is $O(\log n / \log \log n)$...” <i>Id.</i> 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.</p> <p>To the extent that Bedrock argues that Dietzfelbinger 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. <i>Id.</i> Thus, Dietzfelbinger 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.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>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.” <i>Id.</i> at 5.</p> <p>A call to the procedure <i>Insert(x)</i> 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.” <i>Id.</i> at 6.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
		<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked “deleted” then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked “deleted” into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 ("Dietzfelbinger").</p>
		<pre> else (* T_j is too small *) m_j ← 2 · max{1, m_j}; s_j ← 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 ← length of L_j; repeat h_j ← randomly chosen function in 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); </pre> <p align="right"><i>Id.</i> at Figure 1.</p>
<p>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.</p>	<p>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.</p>	<p>Dietzfelbinger discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, a call to the procedure <i>Insert(x)</i> conducts the following steps, as displayed by the code in the figure below. The procedure hashes <i>x</i> to determine the proper position for <i>x</i>, <i>h_j(x)</i>. First, the procedure determines if <i>x</i> already exists in the sub table. If <i>x</i> does exist, then the only determination to make is whether <i>x</i> is marked for deletion. If <i>x</i> is marked for deletion, then the procedure removes this deletion tag. <i>Id.</i> at 6.</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>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. <i>Id.</i></p> <p>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.” <i>Id.</i></p> <p>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 <i>Insert(x)</i> only if the above two conditions are not met.</p>

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, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i> , Revised Version January 7, 1990 ("Dietzfelbinger").
		<pre>procedure <i>Insert</i>(<i>x</i>); <i>count</i> ← <i>count</i> + 1; if <i>count</i> > <i>M</i> then <i>RehashAll</i>(<i>x</i>); else <i>j</i> ← <i>h</i>(<i>x</i>); if position <i>h_j</i>(<i>x</i>) of subtable <i>T_j</i> contains <i>x</i> then if <i>x</i> is marked "deleted" then remove this tag; else (* <i>x</i> is new for <i>W_j</i> *) <i>b_j</i> ← <i>b_j</i> + 1; if <i>b_j</i> ≤ <i>m_j</i> then (* size of <i>T_j</i> sufficient *) if position <i>h_j</i>(<i>x</i>) of <i>T_j</i> is empty then store <i>x</i> in position <i>h_j</i>(<i>x</i>) of <i>T_j</i>; else go through the subtable <i>T_j</i>, put all elements not marked "deleted" into a list <i>L_j</i>, and mark all positions of <i>T_j</i> empty; append <i>x</i> to list <i>L_j</i>; <i>b_j</i> ← length of <i>L_j</i>; repeat <i>h_j</i> ← randomly chosen function in \mathcal{H}_{s_j} until <i>h_j</i> is injective on the elements of list <i>L_j</i>; for all <i>y</i> on list <i>L_j</i> store <i>y</i> in position <i>h_j</i>(<i>y</i>) of <i>T_j</i>;</pre>

EXHIBIT C-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<pre> else (* T_j is too small *) m_j ← 2 · max{1, m_j}; s_j ← 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 ← length of L_j; repeat h_j ← randomly chosen function in 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); </pre> <p><i>Id.</i> at Figure 1.</p> <p>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</p>

EXHIBIT C-6

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Deide, Hans Rohnert, Robert E. Tarjan, <i>Dynamic Perfect Hashing: Upper and Lower Bounds</i>, Revised Version January 7, 1990 (“Dietzfelbinger”).</p>
	<p>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.</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Griffioen discloses an information storage and retrieval system.</p> <p>For example, the Remote Memory Model described in Griffioen “consists of multiple <i>client</i> machines, one or more <i>memory server machines</i>, 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.</p>
<p>[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,</p>	<p>[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,</p>	<p>Griffioen describes the use of a hash table with a double hashing algorithm to locate data. Griffioen at 101.</p> <p>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 (citing to several prior art</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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. <i>See also</i> Mark A. Weiss, <i>Data Structures and Algorithm Analysis</i>, p. 157, 1993 (“<i>Closed hashing</i>, also known as <i>open addressing</i>, is an alternative to resolving collisions with linked lists.”). Double hashing is another form of open addressing. <i>See</i> Knuth at 519-524.</p> <p>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.</p> <p>The records in the system Griffioen discloses includes records, at least some of which automatically expire.</p> <p>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</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120	James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, 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,	<p>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.</p> <p>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.</p> <p>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.</p>
[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	<p>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.</p> <p>For example, the remote memory server “reclaims memory while processing store and fetch requests. When a client issues a store or fetch request, the</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
<p>when the linked list is accessed, and</p>	<p>the linked list is accessed, and</p>	<p>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. <i>Id.</i></p> <p>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.</p>
<p>[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.</p>	<p>[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</p>	<p>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.</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>records in the accessed linked list of records.</p>	<p>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. <i>Id.</i> 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.</p> <p>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.</p>
<p>2. The information storage</p>	<p>6. The information storage</p>	<p>Griffioen combined with Dirks, Thatte, the ’663 patent and/or the</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p style="padding-left: 40px;">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.</p> <p>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).</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>occurrence of a certain event, e.g. by the time the list of free VSIDs is empty.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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.</p> <p>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.`</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>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. 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.</p> <p>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 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.</p> <p>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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>	
		<p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-7

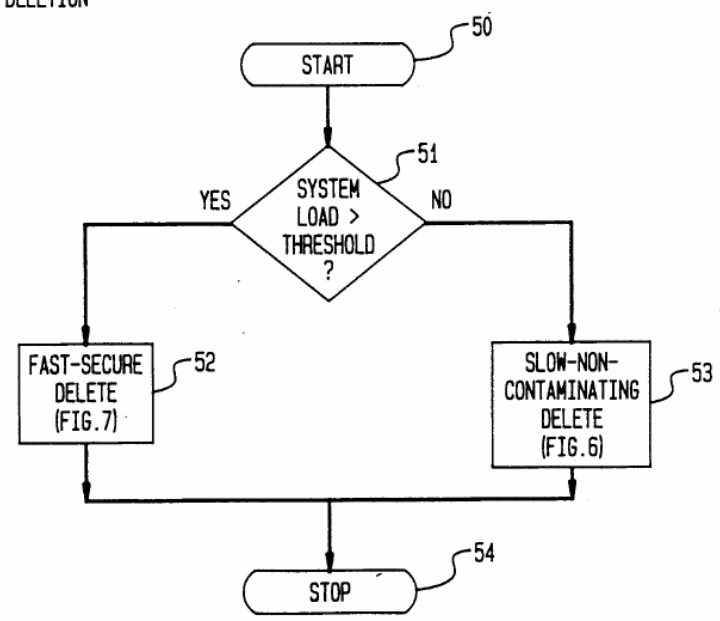
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>	
		<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 their established functions.</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Griffioen with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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.</p> <p>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.</p> <p>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 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</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>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.</p> <p>For example, the Remote Memory Model described in Griffioen “consists of multiple <i>client</i> machines, one or more <i>memory server machines</i>, 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.</p> <p>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</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>external chaining using linked lists could be used instead of a hash table with double hashing.</p> <p>The records in the system Griffioen discloses includes records, at least some of which automatically expire.</p> <p>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.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Griffioen discloses accessing the chain of records. Griffioen also discloses accessing a chain of records beginning at the same hash address.</p> <p>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</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
		list in search of the desired record.
[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,	<p>Griffioen discloses identifying at least some of the automatically expired ones of the records.</p> <p>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. <i>Id.</i></p>
[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	<p>Griffioen discloses removing at least some of the automatically expired records from the chain of records when the chain is accessed.</p> <p>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</p>

EXHIBIT C-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>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. <i>Id.</i></p> <p>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.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Griffioen discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>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.</p>
<p>4. The method according to claim 3 further including the step of dynamically determining maximum</p>	<p>8. The method according to claim 7 further including the step of dynamically determining maximum</p>	<p>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.</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
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.	<p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>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 that described 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 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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.</p> <p>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. Thatte,</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
		<p>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.</p> <p>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 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Griffioen with Thatte and recognized 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</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
		<p>Thatte.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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.</p>

EXHIBIT C-7

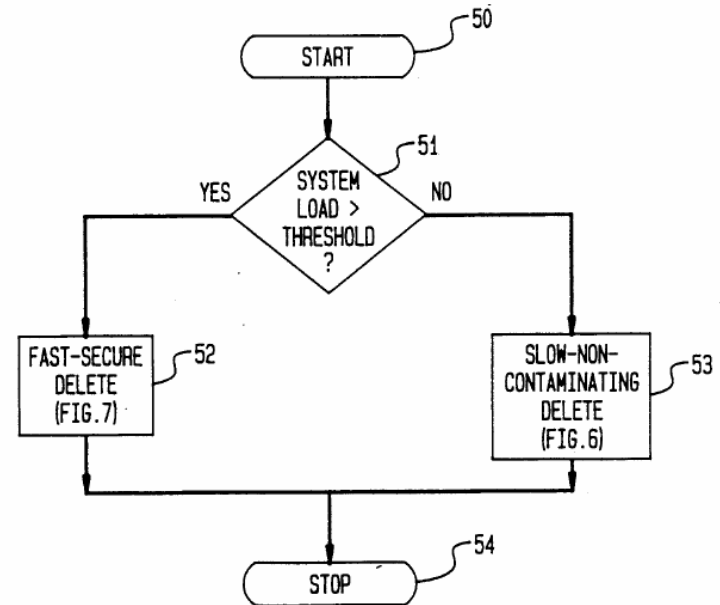
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p><i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
		<p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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.</p> <p>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.</p> <p>Alternatively, it would also be obvious to combine Griffioen with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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</p>

EXHIBIT C-7

Asserted Claims From U.S. Pat. No. 5,893,120		James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination
		<p>garbage collection is likely to be. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>
	<p>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.</p> <p>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 that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system.</p> <p>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</p>

EXHIBIT C-7

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>James Griffioen, <i>Remote Memory Backing Storage for Distributed Virtual Memory Operating Systems</i>, PhD (1992), available at http://protocols.netlab.uky.edu/~griff/papers/phd_thesis (“Griffioen”) alone and in combination</p>	
		<p>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.</p> <p>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.</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Comer discloses an information storage and retrieval system.</p> <p>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.</p>
<p>[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,</p>	<p>[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,</p>	<p>Comer describes the use of a hash table with a double hashing algorithm to locate data. P. 10.</p> <p>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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>“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. <i>See also</i> Mark A. Weiss, <i>Data Structures and Algorithm Analysis</i>, p. 157, 1993 (“<i>Closed hashing</i>, also known as <i>open addressing</i>, is an alternative to resolving collisions with linked lists.”). Double hashing is another form of open addressing. <i>See</i> Knuth at 519-524.</p> <p>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.</p> <p>The records in the system Comer discloses includes records, at least some of which automatically expire.</p> <p>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 <i>data</i> 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.</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>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.</p> <p>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.”</p> <p>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.</p>
<p>[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</p>	<p>[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</p>	<p>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.</p> <p>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.</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>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.</p>
<p>[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.</p>	<p>[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.</p>	<p>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.</p> <p>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.</p> <p>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.</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (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.	<p>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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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.</p> <p>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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>system from performing sometimes time-consuming sweeps.`</p> <p>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. 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.</p> <p>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 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.</p> <p>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 many records to delete based on, among other things, the system load.</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
	<p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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”).</p> <p>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</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>	
		<p>moving records in the chain as described above. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-8

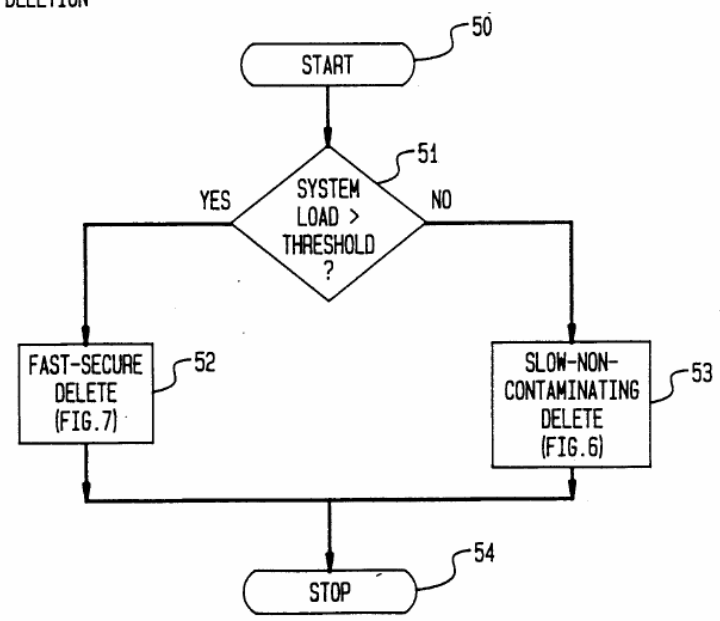
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 established functions.</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Comer with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>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.</p> <p>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 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</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>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.</p> <p>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; <i>see also</i> Comer at 1.</p> <p>Comer describes a method of using of a hash table with a double hashing algorithm to locate data on the remote memory server. P. 10. There are two</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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 (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, <i>Data Structures and Algorithm Analysis</i>, p. 157, 1993 (“<i>Closed hashing</i>, also known as <i>open addressing</i>, is an alternative to resolving collisions with linked lists.”). Double hashing is another form of open addressing. See Knuth at 519-524.</p> <p>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</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model (1990)</i>, available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
		<p>ordinary skill.</p> <p>The records in the system Comer discloses includes records, at least some of which automatically expire.</p> <p>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 <i>data</i> 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.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Comer discloses accessing the chain of records. Comer also discloses accessing a chain of records beginning at the same hash address.</p> <p>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. P. 10.</p> <p>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.</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Comer discloses identifying at least some of the automatically expired ones of the records.</p> <p>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.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Comer discloses the removing at least some of the automatically expired records from the chain of records when the chain of records is accessed.</p> <p>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.</p> <p>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.</p>

EXHIBIT C-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Comer discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Comer states that “[t]he server reclaims obsolete pages during later probes to the data hash table” P. 10.</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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 that described 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 would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>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. 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.</p> <p>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 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Comer with Thatte and recognized the benefits of doing so. For example, the removal</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model (1990)</i>, available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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”).</p> <p>In times of heavy use, when deletions must be done rapidly and</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>	
		<p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-8

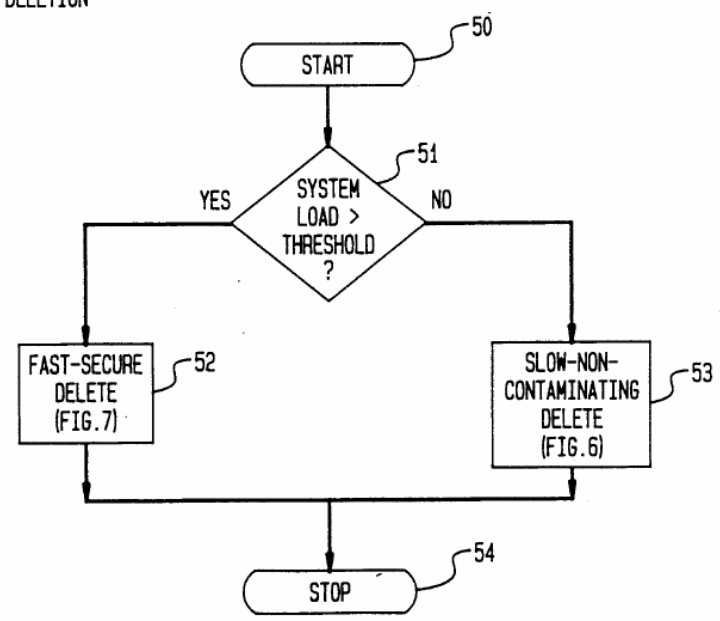
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 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 established functions.</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Comer with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

EXHIBIT C-8

Asserted Claims From U.S. Pat. No. 5,893,120		Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination
		<p>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.</p> <p>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 that the system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system.</p> <p>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 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</p>

EXHIBIT C-8

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Douglas Comer and James Griffioen, <i>A New Design for Distributed Systems: The Remote Memory Model</i> (1990), available at http://protocols.netlab.uky.edu/~griff/papers/usenix90.pdf (“Comer”) alone and in combination</p>
	<p>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.</p> <p>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.</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Sessions discloses an information storage and retrieval system.</p> <p>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. (<i>See id.</i>)</p>
<p>[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,</p>	<p>[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,</p>	<p>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.</p> <p>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. 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</p> <ul style="list-style-type: none"> • to a keyed table or to a keyed linked-list of those data-items whose keys all hash to that place in the table” <p>(<i>See id.</i>). 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</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>similar systems and methods in the same way.</p> <p>Sessions also discloses that a “function <code>ca_add()</code> 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).</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Sessions discloses a record search means utilizing a search key to access the linked list.</p> <p>For example, Sessions discloses that “[t]he next function, <code>ca_check()</code>, 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.</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<pre>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); } }</pre> <p>(Sessions at 30).</p> <p>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 compiler <i>should</i> include “garbage collection” in a program translated from a Pascal source that uses any pointer-types: when the program runs low on spare</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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.” (<i>See id.</i>) 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.</p>
<p>[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</p>	<p>[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</p>	<p>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.</p> <p>For example, Sessions discloses the standard function <code>free()</code>, which “receives a pointer to one of the blocks in the dynamic memory pool previously allocated by <code>malloc()</code>. The pool is first scanned to validate the pointer, and then the block is released for use. (Sessions at 78).</p>
<p>[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.</p>	<p>[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.</p>	<p>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.</p> <p>For example, Sessions discloses that the “functions <code>mmget()</code> and <code>mmfree()</code></p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>can solve this problem by dynamically collecting free memory into a single contiguous chunk. ... if a program uses <code>mmget()</code> to set a pointer <code>p1</code> to a block of memory, <code>p1</code> 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 <code>p1</code>.” (Sessions at 78-79).</p> <p>Also, Sessions discloses that “[t]he static functions <code>setfree()</code> and <code>consolidate()</code> perform the work of freeing a used memory block. ... This rudimentary garbage collection could be accomplished by the standard C library function <code>free()</code> as well. (Sessions at 83).</p> <p>Further, Sessions discloses an exercise where the student must “[m]odify <code>mmfree()</code> so that a full garbage collection is always performed with each call. What are the advantages and disadvantages of this change?” (Sessions at 86).</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>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).</p> <p>Sessions also discloses a routine <code>ca_check()</code> which “promote(s) any found items to the head of the linked list, indicating their changed status to ‘Most</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>Recently Referenced.” (Sessions at 91).</p> <p>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. (<i>See id.</i>)</p> <p>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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>combining Dirks’ deletion decision procedure with Sessions nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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.`</p> <p>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. 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.</p> <p>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.</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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.</p> <p>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 entirety. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>4,996,663 to Nemes at 2:24-34 (“The ’663 patent”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-9

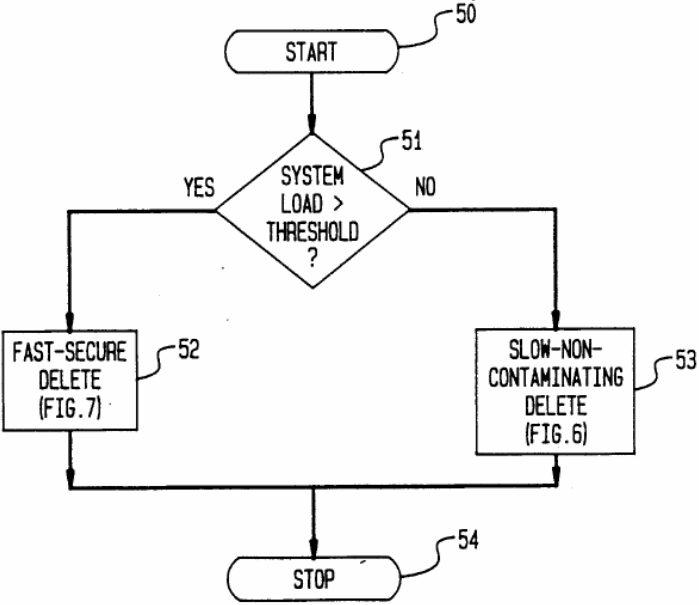
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 their established functions.</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Sessions with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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.</p> <p>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.</p> <p>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 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</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>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.</p> <p>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. 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</p> <ul style="list-style-type: none"> • to a keyed table or to a keyed linked-list of those data-items whose keys

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>all hash to that place in the table” (<i>See id.</i>). 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 similar systems and methods in the same way.</p> <p>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.</p> <p>Sessions also discloses that a “function <code>ca_add()</code> 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).</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Sessions discloses accessing a linked list of records.</p> <p>For example, Sessions discloses that “[t]he next function, <code>ca_check()</code>, 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.</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<pre> 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); } } </pre> <p>(Sessions at 30).</p> <p>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 compiler <i>should</i> include “garbage collection” in a program translated from a Pascal source that uses any pointer-types: when the program runs low on spare</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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.” (<i>See id.</i>) 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.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>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.</p> <p>For example, Sessions discloses the standard function <code>free()</code>, which “receives a pointer to one of the blocks in the dynamic memory pool previously allocated by <code>malloc()</code>. The pool is first scanned to validate the pointer, and then the block is released for use. (Sessions at 78).</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>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.</p> <p>Also, Sessions discloses that “[t]he static functions <code>setfree()</code> and <code>consolidate()</code> perform the work of freeing a used memory block. ... This rudimentary garbage collection could be accomplished by the standard C library function <code>free()</code> as well. (Sessions at 83).</p> <p>Further, Sessions discloses an exercise where the student must “[m]odify</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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).</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Sessions discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Sessions discloses deleting one of the records from the system in the consolidate() function following the step of removing.</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) ("Sessions") alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) ("Lester") and others</p>
		<pre> 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(); } } } </pre> <p>(Sessions at 83).</p>
4. The method according to	8. The method according	Sessions discloses dynamically determining maximum number of expired ones

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
<p>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.</p>	<p>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.</p>	<p>of the records to remove when the linked list is accessed.</p> <p>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).</p> <p>Sessions also discloses a routine <code>ca_check()</code> which “promote(s) any found items to the head of the linked list, indicating their changed status to ‘Most Recently Referenced.’” (Sessions at 91).</p> <p>Sessions further discloses an exercise where the student must “[m]odify <code>mmfree()</code> so that garbage is collected whenever the largest single chunk of available memory is less than $\frac{3}{4}$ 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. (<i>See id.</i>)</p> <p>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.</p> <p>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</p>

EXHIBIT C-9

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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.</p> <p>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.</p> <p>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. 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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>Thatte, which is hereby incorporated by reference in its entirety.</p> <p>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.</p> <p>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.</p> <p>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</p>

EXHIBIT C-9

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>entirety. As summarized in the '663 patent:</p> <p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-9

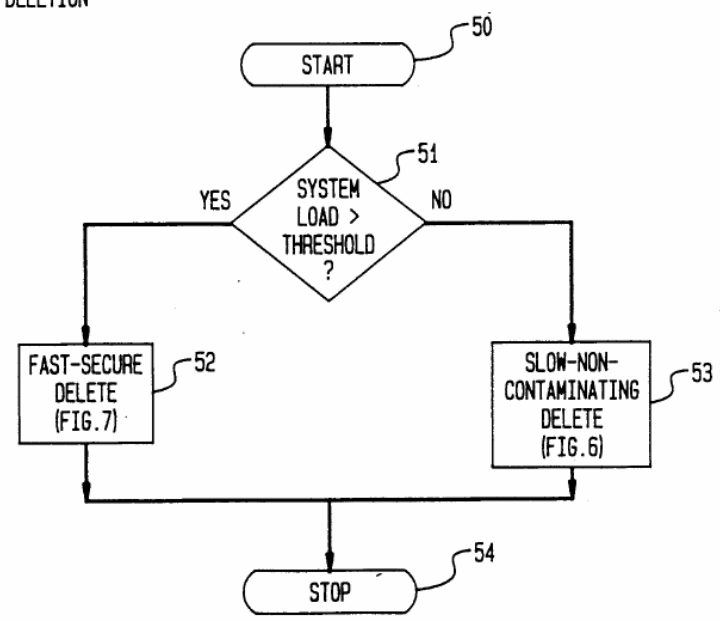
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre> graph TD Start([START 50]) --> Decision{SYSTEM LOAD > THRESHOLD? 51} Decision -- YES --> ProcessFast[FAST-SECURE DELETE (FIG. 7) 52] Decision -- NO --> ProcessSlow[SLOW-NON-CONTAMINATING DELETE (FIG. 6) 53] ProcessFast --> Stop([STOP 54]) ProcessSlow --> Stop </pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>functions.</p> <p>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.</p> <p>Alternatively, it would also be obvious to combine Sessions with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>Every time a user-input routine is invoked, a decision routine can decide</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

EXHIBIT C-9

Asserted Claims From U.S. Pat. No. 5,893,120		Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others
		<p>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.</p> <p>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.</p> <p>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 linked list of records to solve a number of potential problems. For example,</p>

EXHIBIT C-9

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Roger Sessions, <i>Reusable Data Structures for C</i> (Prentice-Hall, Inc. 1989) (“Sessions”) alone and in combination with Kit Lester, <i>A Practical Approach to Data Structures: Related Algorithms in Pascal with Applications</i> (Ellis Horwood Ltd. 1990) (“Lester”) and others</p>
		<p>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.</p> <p>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.</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">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</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Van Wyk 2 discloses an information storage and retrieval system.</p> <p>For example, Van Wyk 2 discloses external data structures, explaining that "<i>File systems</i> are a familiar example of an external data structure." (Van Wyk 2 at 142).</p>
<p>[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,</p>	<p>[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,</p>	<p>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.</p> <p>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. (<i>See id.</i>)</p> <p>Van Wyk 2 further discloses that "[i]n the <i>reference count</i> 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. (<i>See id.</i>)</p> <p>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).</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">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</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>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.</p> <p>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).</p>
<p>[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</p>	<p>[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</p>	<p>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.</p> <p>For example, Van Wyk 2 discloses that “[i]n the <i>reference count</i> 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).</p>
<p>[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.</p>	<p>[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.</p>	<p>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.</p> <p>For example, Van Wyk 2 discloses “[t]he <i>lazy</i> approach to garbage collection is to collect only in emergencies. Thus, when an allocation fails, we sweep</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">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</p>
		<p>through memory hoping to pick up and de-allocate enough garbage to permit the program to continue.” (Van Wyk 2 at 146).</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>For example, Van Wyk 2 discloses that “[o]ur programs have never used <code>free()</code>. ... [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).</p> <p>Van Wyk 2 further states that “[a]ccess to the heap is through a single pointer, <i>rover</i>, which always points to the last node allocated or de-allocated. ... As soon as <i>rover</i> points to a node with enough room, <i>malloc()</i> 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 <i>rover</i> 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. (<i>See id.</i>)</p> <p>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.</p> <p>Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system,</p>

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
		<p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">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</p>
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">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</p>
	<p>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.</p> <p>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.</p> <p>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. 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.</p>

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
		<p>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.</p> <p>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.</p> <p>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. As summarized in the '663 patent:</p>

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
		<p>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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-10

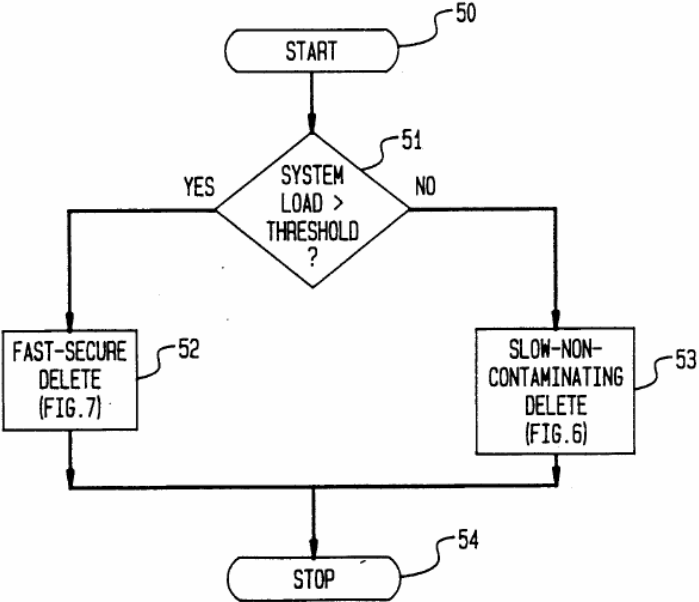
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Christopher J. Van Wyk, Data Structures and C Programs (Addison- Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">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</p>
	<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

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
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Van Wyk 2 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

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
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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</p>

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
		<p>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.</p> <p>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.</p> <p>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 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.</p> <p>One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">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</p>
		<p>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, <i>rover</i>, which always points to the last node allocated or de-allocated. ... As soon as <i>rover</i> points to a node with enough room, <i>malloc()</i> 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 <i>rover</i> as a pointer to the newly allocated space.” (Van Wyk 2 at 137).</p>
<p>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:</p>	<p>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:</p>	<p>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.</p> <p>For example, Van Wyk 2 discloses external data structures, explaining that “<i>File systems</i> are a familiar example of an external data structure.” (Van Wyk 2 at 142).</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">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</p>
		<p>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.</p> <p>Van Wyk 2 further discloses that “[i]n the <i>reference count</i> 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. (<i>See id.</i>)</p> <p>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).</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>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.</p> <p>For example, Van Wyk 2 states that “[a]ccess to the heap is through a single pointer, <i>rover</i>, which always points to the last node allocated or de-allocated. ... As soon as <i>rover</i> points to a node with enough room, <i>malloc()</i> 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 <i>rover</i> as a pointer to the newly allocated space.” (Van Wyk 2 at 137).</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>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.</p>

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
		For example, Van Wyk 2 discloses that “[i]n the <i>reference count</i> 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	<p>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.</p> <p>For example, Van Wyk 2 discloses “[t]he <i>lazy</i> 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).</p>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>Van Wyk 2 discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>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).</p>
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.	<p>Van Wyk 2 discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, Van Wyk 2 discloses that “[o]ur programs have never used <code>free()</code>. ... [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).</p> <p>Van Wyk 2 further states that “[a]ccess to the heap is through a single pointer,</p>

EXHIBIT C-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">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</p>
	<p><i>rover</i>, which always points to the last node allocated or de-allocated. ... As soon as <i>rover</i> points to a node with enough room, <i>malloc()</i> 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 <i>rover</i> 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.</p> <p>Van Wyk 2 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

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
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p>

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
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

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
		<p>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.</p> <p>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. 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.</p> <p>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.</p> <p>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 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</p>

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
		<p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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").</p> <p>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</p>

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
		<p>automatically remove the contaminating deleted records by moving records in the chain as described above. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-10

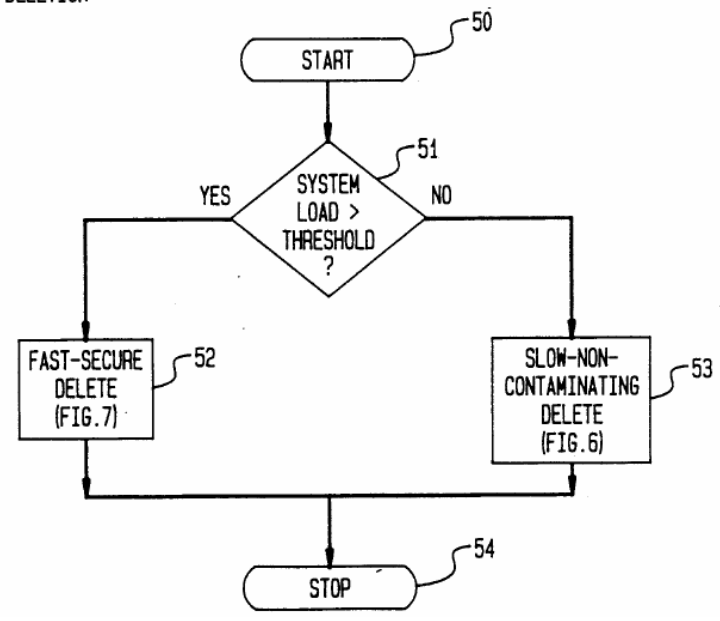
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Christopher J. Van Wyk, Data Structures and C Programs (Addison- Wesley Publ'g Co. & Bell Telephone Laboratories, Inc. 1988) ("Van Wyk 2") alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

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
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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 functions.</p>

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
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Van Wyk 2 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

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
		<p>continual run-time overhead. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 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.</p>

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
		<p>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.</p> <p>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.</p> <p>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 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.</p>

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
		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, <i>rover</i> , which always points to the last node allocated or de-allocated. ... As soon as <i>rover</i> points to a node with enough room, <i>malloc()</i> 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 <i>rover</i> as a pointer to the newly allocated space." (Van Wyk 2 at 137).

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (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. <i>See id.</i>
[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

EXHIBIT C-11

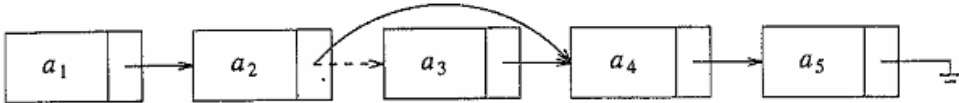
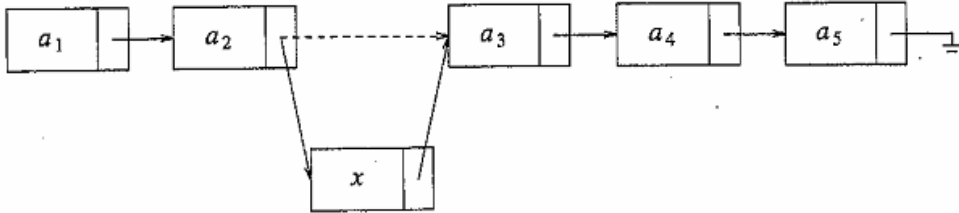
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
<p>access the linked list,</p>	<p>access a linked list of records having the same hash address,</p>	<p>access a linked list of records having the same hash address.</p> <p>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 <i>malloc</i> 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.</p> <div style="text-align: center;">  <p>Figure 3.3 Deletion from a linked list</p> </div> <div style="text-align: center;">  <p>Figure 3.4 Insertion into a linked list</p> </div> <p>Further, Weiss discloses, “[d]eciding what to do when two keys hash to the same value (this is known as a <i>collision</i>).” 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.</p>
<p>[1c] the record search means including a means for identifying and</p>	<p>[5c] the record search means including means for identifying and removing</p>	<p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
<p>removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and</p>	<p>at least some expired ones of the records from the linked list of records when the linked list is accessed, and</p>	<p>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.</p> <p>For example, Weiss discloses that “[w]hen things are no longer needed, you can issue a <i>free</i> command to inform the system that it may reclaim the space. A consequence of the <i>free</i>(<i>p</i>) command is that the address that <i>p</i> is pointing to is unchanged, but the data that resides at that address is now undefined.” Weiss at 50.</p> <p>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 <i>vivify</i> is to traverse the list <i>live</i>, determine whether each cell on it satisfies the conditions to become alive, and <i>vivify</i> 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 from 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 <i>AddNeighbors</i>.” 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.</p>
<p>[1d] means, utilizing the record search means, for accessing the linked list and, at the same time,</p>	<p>[5d] mea[n]s, utilizing the record search means, for inserting, retrieving, and deleting records from the</p>	<p>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</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
<p>removing at least some of the expired ones of the records in the linked list.</p>	<p>system and, at the same time, removing at least some expired ones of the records in the accessed linked list of records.</p>	<p>the same time, removing at least some expired ones of the records in the accessed linked list of records.</p> <p>For example, Weiss discloses method of retrieving records in a find routine as shown in figure 3.10.</p> <p>Figure 3.10 <i>Find routine</i></p> <hr/> <pre> /* 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; } </pre> <p>Weiss at 46.</p> <p>Weiss also discloses a method of inserting a record in an insert routine as shown in figure 3.12.</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>Figure 3.12 <i>Find_previous</i>—the <i>find</i> routine for use with <i>delete</i></p> <hr/> <pre> /* 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; } } </pre> <p><i>Handwritten annotations:</i></p> <ul style="list-style-type: none"> Handwritten text on the left: "if it", "is where", "ould", "4*/", "5*/", "6*/" A bracket groups the <code>tmp_cell = (position) malloc(sizeof(struct node));</code> and <code>if(tmp_cell == NULL)</code> lines. The lines <code>tmp_cell->next = p->next;</code> and <code>p->next = tmp_cell;</code> are circled in black. A question mark "?" is written next to the circled lines. <p>Weiss at 48.</p> <p>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 at 156. 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.”</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>Figure 3.15 Correct way to delete a list</p> <hr/> <pre> 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; } } </pre> <hr/> <p>Weiss at 50.</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>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.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>Figure 5.8 Initialization routine for open hash table</p> <pre>HASH_TABLE initialize_table(unsigned int table_size) { /* Allocate list pointers */ /*8*/ H->the_lists = (position *) /*9*/ malloc(sizeof (LIST) * H->table_size); /*10*/ if(H->the_lists == NULL) 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;</pre> <p><i>Handwritten annotations:</i> - A bracket on the left groups lines 8-10. - A circle around "table_size" in line 2. - A circle around "position" in line 8. - A circle around "H->table_size" in line 9. - A handwritten note "multiply" with an arrow pointing to the "*" in line 9. - A circle around "H->table_size" in line 9.</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
		<pre> 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; } } } </pre> <p><i>Gr.C. here</i></p> <p><i>would have</i></p> <hr/> <p>Figure 5.10 <i>Insert routine for open hash table</i></p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>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.</p> <p>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.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>Exhibit B-2, which is hereby incorporated by reference in its entirety.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>number of entries that are swept is identified as k, where:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>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.</p> <p>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.</p> <p>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. 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.</p> <p>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.</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>The resulting combination would include the capability to determine the maximum number for the record search means to remove.</p> <p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>4,996,663 to Nemes at 2:24-34 (“The ’663 patent”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-11

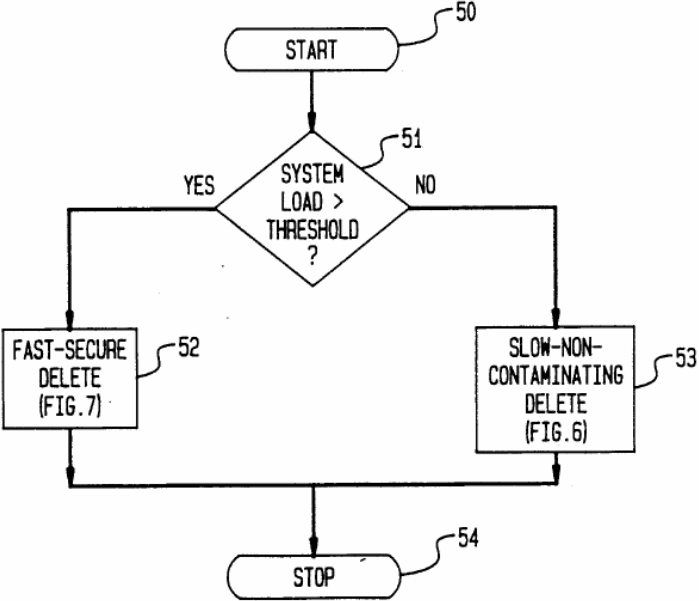
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33,</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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.</p> <p>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</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>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.</p> <p>Alternatively, it would also be obvious to combine Weiss with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenger. If the product of the allocation and the compute time is high, and if the stack is low, the scavenger favorability measure is high. If it is especially high, a multi-generation scavenger is in order. <i>Id.</i></p> <p>If these heuristics fail and a scavenger 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 scavenger pause within a larger pause. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 art elements according to their established functions.</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>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.</p> <p>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.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>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 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.</p> <p>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. <i>See id.</i></p> <p>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.</p>

EXHIBIT C-11

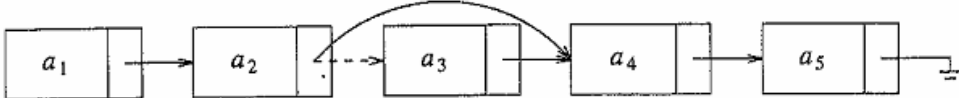
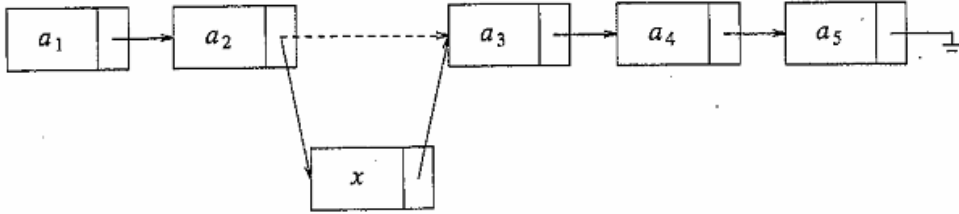
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	<p>Weiss discloses accessing a linked list of records. Weiss also discloses accessing a linked list of records having same hash address.</p> <p>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 <i>malloc</i> 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.</p>  <p>Figure 3.3 Deletion from a linked list</p>  <p>Figure 3.4 Insertion into a linked list</p> <p>Further, Weiss discloses, “[d]eciding what to do when two keys hash to the same value (this is known as a <i>collision</i>).” 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.</p>
[3b] identifying at least some of the automatically	[7b] identifying at least some of the automatically	Weiss discloses identifying at least some of the automatically expired ones of the records.

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
expired ones of the records, and	expired ones of the records,	<p>For example, Weiss discloses that “[w]hen things are no longer needed, you can issue a <i>free</i> command to inform the system that it may reclaim the space. A consequence of the <i>free(p)</i> command is that the address that <i>p</i> is pointing to is unchanged, but the data that resides at that address is now undefined.” Weiss at 50.</p> <p>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 <i>vivify</i> is to traverse the list <i>live</i>, 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 from 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 <i>AddNeighbors</i>.” 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.</p>
[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	<p>Weiss discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, Weiss discloses that “[w]hen things are no longer needed, you can issue a <i>free</i> command to inform the system that it may reclaim the space. A consequence of the <i>free(p)</i> command is that the address that <i>p</i> is pointing to is unchanged, but the data that resides at that address is now undefined.”</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
		<p>Weiss at 50.</p> <p>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 <code>Vivify</code> is to traverse the list <code>live</code>, 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 from 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 <code>AddNeighbors</code>.” 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.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Weiss discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Weiss discloses method of retrieving records in a find routine as shown in figure 3.10.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>Figure 3.10 <u>Find routine</u></p> <pre>/* 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; }</pre> <p>Weiss at 46.</p> <p>Weiss also discloses a method of inserting a record in an insert routine as shown in figure 3.12.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>Figure 3.12 <i>Find_previous</i>—the <i>find</i> routine for use with <i>delete</i></p> <pre>/* 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; } }</pre> <p><i>Handwritten notes:</i> if it *1*/ *2*/ *3*/ is where ould *4*/ *5*/ *6*/ ? ?</p> <p>Weiss at 48.</p> <p>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 at 156. 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.”</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>Figure 3.15 Correct way to delete a list</p> <hr/> <pre> 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; } } </pre> <hr/> <p>Weiss at 50.</p>
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.	<p>Weiss discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>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.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>Figure 5.8 Initialization routine for open hash table</p> <pre>HASH_TABLE initialize_table(unsigned int table_size) { /* Allocate list pointers */ /*8*/ H->the_lists = (position *) /*9*/ malloc(sizeof (LIST) * H->table_size); /*10*/ if(H->the_lists == NULL) fatal_error("Out of space!!!"); /* Allocate list headers */ /*11*/ for(i=0; i<H->table_size; i++) /*12*/ { H->the_lists[i] = (LIST) malloc /*13*/ (sizeof (struct list_node)); /*14*/ if(H->the_lists[i] == NULL) fatal_error("Out of space!!!"); else /*15*/ H->the_lists[i]->next = NULL; }</pre> <p><i>Handwritten annotations:</i> - A bracket on the left side groups lines 8 through 15. - A circle around "table_size" in line 2. - A circle around "position" in line 8. - A circle around "H->table_size" in line 9. - A handwritten note "multiply" with an arrow pointing to the "*" operator in line 9.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<pre>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; } } }</pre> <p><i>Gr.C. here</i></p> <p>Figure 5.10 <i>Insert routine for open hash table</i></p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>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.</p> <p>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.</p> <p>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.</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>As summarized in Dirks,</p> <p>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.</p> <p>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:</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>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</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>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.</p> <p>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.</p> <p>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. 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.</p> <p>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</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>maximum number for the record search means to remove.</p> <p>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.</p> <p>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. As summarized in the '663 patent:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>4,996,663 to Nemes at 2:24-34 (“The ’663 patent”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-11

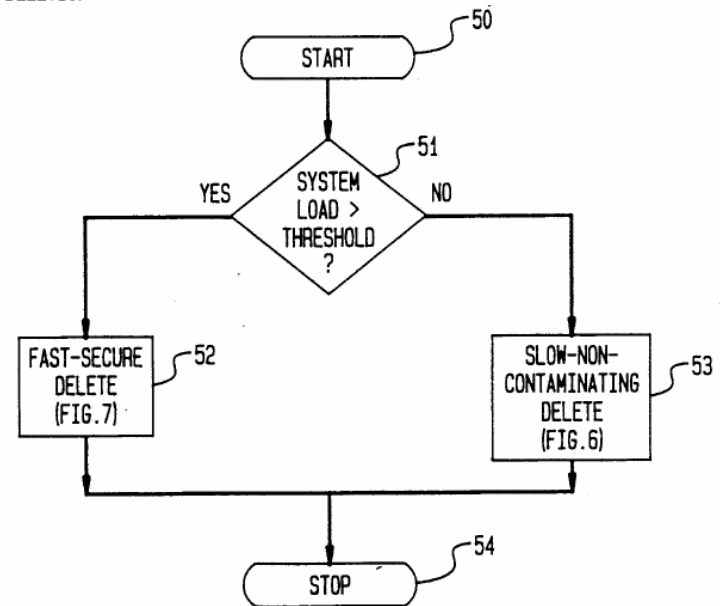
Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33,</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>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 understand 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.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120	Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
	<p>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.</p> <p>Alternatively, it would also be obvious to combine Weiss with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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 art elements according to their established functions.</p>

EXHIBIT C-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (The Benjamin/Cummings Publ'g Co. 1993) alone and in combination</p>
	<p>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.</p> <p>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.</p> <p>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</p>

EXHIBIT C-11

Asserted Claims From U.S. Pat. No. 5,893,120		Mark Allen Weiss, <i>Data Structures & Algorithm Analysis in C</i> (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.” ‘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.

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Frakes discloses an information storage and retrieval system.</p> <p>For example, Frakes discloses “hashing, an information storage and retrieval technique useful for implementing many ... other structures.” (Frakes at 293).</p>
<p>[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,</p>	<p>[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,</p>	<p>Frakes 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. Frakes 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.</p> <p>For example, Frakes discloses “hashing, an information storage and retrieval technique useful for implementing many ... other structures.” (Frakes at 293). Frakes discloses that <i>hashing</i> is “a ubiquitous information retrieval strategy for providing efficient access to information based on a key.” <i>Id.</i> Frakes further discloses “chained hashing. It is so named because each bucket stores a linked list—that is, a chain—of key-information pairs, rather than a single one.” (Frakes at 298).</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Frakes discloses a record search means utilizing a search key to access the linked list. Frakes also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, Frakes discloses that “[t]he goal (of hashing) is to avoid <i>collisions</i>. A collision occurs when two or more keys map to the same location. If no keys collide, then locating the information associated with a key is simply the process of determining the key’s location. Whenever a</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
		<p>collision occurs, some extra computation is necessary to further determine a unique location for a key.” (Frakes at 294).</p>
<p>[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</p>	<p>[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</p>	<p>Frakes 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. Frakes 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.</p> <p>For example, Frakes discloses that a “hash table with m buckets may therefore store more than m keys. However, performance will degrade as the number of keys increases. Computing the bucket in which a key resides is still fast—a matter of evaluating the hash function—but locating it within that bucket (or simply determining its presence, which is necessary in all operations) requires traversing the linked list.” (Frakes at 299).</p> <p>Additionally, Frakes discloses that “performance will degrade as the number of keys increases.” <i>Id.</i> Thus, Frakes suggests removing at least some of the automatically expired records from the linked list when the linked list is accessed. (<i>See id.</i>)</p>
<p>[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.</p>	<p>[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.</p>	<p>Frakes 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. Frakes 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.</p> <p>For example, Frakes discloses several operations that are usually provided by an implementation of hashing:</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
		<p>1. Initialization: indicate that the hash table contains no elements.</p> <p>2. Insertion: insert information, indexed by a key <i>k</i>, into a hash table. If the has table already contains <i>k</i>, then it cannot be inserted. (Some implementations do allow such insertion, to permit replacing existing information.)</p> <p>3. Retrieval: given a key <i>k</i>, retrieve the information associated with it.</p> <p>4. Deletion: remove the information associated with key <i>k</i> from a hash table, if any exists. New information indexed by <i>k</i> may subsequently be placed in the table.</p> <p>(Frakes at 297).</p> <p>Frakes further discloses that “[m]ost of the code from the routines Insert, Delete, Clear (for Initialize), and Member (for Retrieve) can be used directly.” (Frakes at 299).</p> <p>Additionally, Frakes discloses that “performance will degrade as the number of keys increases.” <i>Id.</i> Thus Frakes suggests removing at least some of the automatically expired records from the linked list when the linked list is accessed. (<i>See id.</i>)</p>
<p>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.</p>	<p>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.</p>	<p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Frakes 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 Frakes 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</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>number of potential problems. For example, the removal of expired records described in Frakes 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 Frakes 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.</p> <p>Frakes 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.</p> <p>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.</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>As summarized in Dirks,</p> <p>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.</p> <p>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:</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Frakes 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 Frakes. 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</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
	<p>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 Frakes nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Frakes and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Frakes 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.</p> <p>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 Frakes with Thatte would be nothing more than the</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Frakes with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Frakes 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 Frakes 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 Frakes with Thatte.</p> <p>Alternatively, it would also be obvious to combine Frakes 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-12

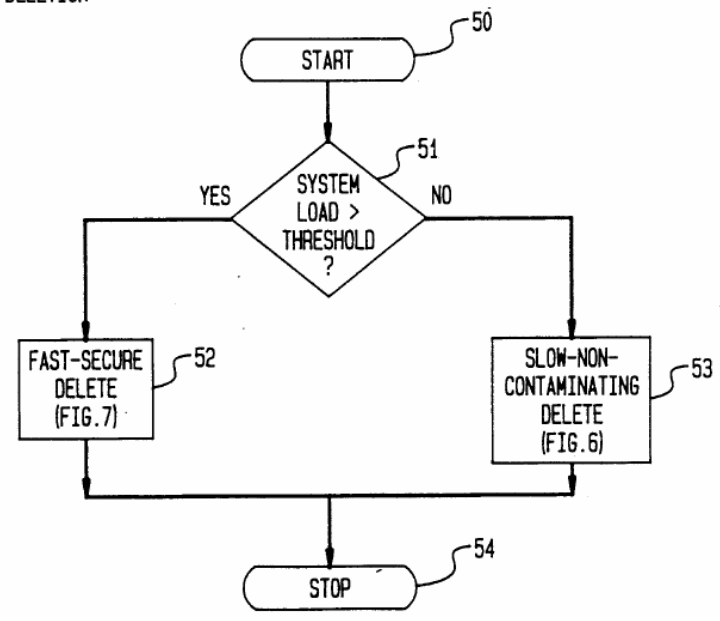
Asserted Claims From U.S. Pat. No. 5,893,120	William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Frakes and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Frakes. 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 Frakes would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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 Frakes 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.</p> <p>Alternatively, it would also be obvious to combine Frakes with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Frakes 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 Frakes. 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</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120	William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
	<p>combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with Frakes would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Frakes and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Frakes 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 Frakes 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 Frakes 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.</p> <p>One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Frakes 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. Frakes 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.</p> <p>For example, Frakes discloses “hashing, an information storage and retrieval technique useful for implementing many ... other structures.” (Frakes at 293).</p> <p>Frakes also discloses “hashing, an information storage and retrieval technique useful for implementing many ... other structures.” (Frakes at 293). Frakes discloses that <i>hashing</i> is “a ubiquitous information retrieval strategy for providing efficient access to information based on a key.” <i>Id.</i> Frakes further discloses “chained hashing. It is so named because each bucket stores a linked list—that is, a chain—of key-information pairs, rather than a single one.” (Frakes at 298).</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same</p>	<p>Frakes discloses accessing a linked list of records. Frakes also discloses accessing a linked list of records having same hash address.</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
	hash address,	For example, Frakes discloses that “[t]he goal (of hashing) is to avoid collisions. A collision occurs when two or more keys map to the same location. If no keys collide, then locating the information associated with a key is simply the process of determining the key’s location. Whenever a collision occurs, some extra computation is necessary to further determine a unique location for a key.” (Frakes at 294).
[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,	Frakes discloses identifying at least some of the automatically expired ones of the records. Frakes also discloses identifying at least some of the automatically expired ones of the records. For example, Frakes discloses that a “hash table with m buckets may therefore store more than m keys. However, performance will degrade as the number of keys increases. Computing the bucket in which a key resides is still fast—a matter of evaluating the hash function—but locating it within that bucket (or simply determining its presence, which is necessary in all operations) requires traversing the linked list.” (Frakes at 299).
[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	Frakes discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Frakes also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, Frakes discloses that “performance will degrade as the number of keys increases.” (Frakes at 299). Thus, Frakes suggests removing at least some of the automatically expired records from the linked list when the linked list is accessed. (<i>See id.</i>)
	[7d] inserting, retrieving or deleting one of the	Frakes discloses inserting, retrieving or deleting one of the records from the system following the step of removing.

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
	<p>records from the system following the step of removing.</p>	<p>For example, Frakes discloses several operations that “are usually provided by an implementation of hashing:</p> <ol style="list-style-type: none"> 1. Initialization: indicate that the hash table contains no elements. 2. Insertion: insert information, indexed by a key <i>k</i>, into a hash table. If the has table already contains <i>k</i>, then it cannot be inserted. (Some implementations do allow such insertion, to permit replacing existing information.) 3. Retrieval: given a key <i>k</i>, retrieve the information associated with it. 4. Deletion: remove the information associated with key <i>k</i> from a hash table, if any exists. New information indexed by <i>k</i> may subsequently be placed in the table.” <p>(Frakes at 297).</p> <p>Frakes further discloses that “[m]ost of the code from the routines Insert, Delete, Clear (for Initialize), and Member (for Retrieve) can be used directly.” (Frakes at 299).</p>
<p>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.</p>	<p>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.</p>	<p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Frakes 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 Frakes 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 Frakes can be burdensome on the system, adding to the system’s</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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 Frakes 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.</p> <p>Frakes 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>each time a VSID is assigned from the free list to a new application or</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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.</p> <p>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:</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Frakes 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 Frakes. 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</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120	William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
	<p>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 Frakes would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Frakes and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Frakes 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.</p> <p>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 Frakes with Thatte would be nothing more than the</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Frakes with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Frakescan 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 Frakes 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 Frakes with Thatte.</p> <p>Alternatively, it would also be obvious to combine Frakes 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-12

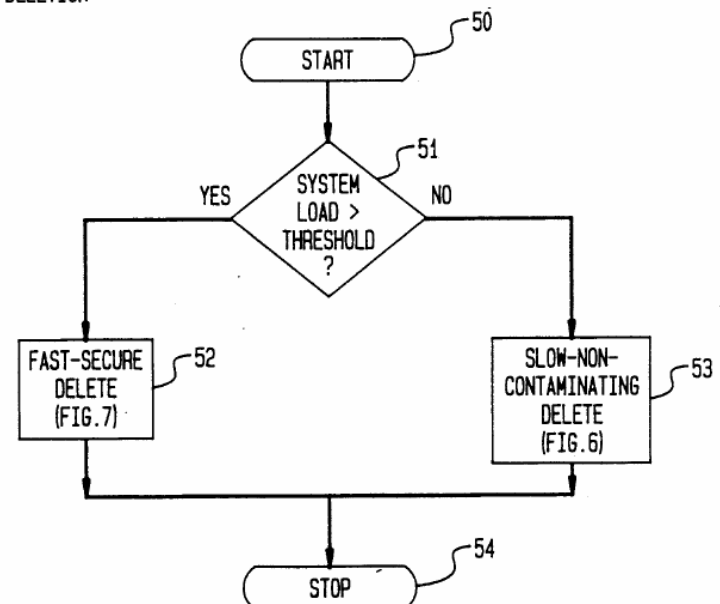
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Frakes and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Frakes. 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 Frakes would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
	<p>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 Frakes 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.</p> <p>Alternatively, it would also be obvious to combine Frakes with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Frakes 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 Frakes. 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</p>

EXHIBIT C-12

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) alone and in combination</p>
	<p>combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with Frakes would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Frakes 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Frakes 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 Frakes 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 Frakes 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.</p> <p>One of ordinary skill in the art would have known that dynamically</p>

EXHIBIT C-12

Asserted Claims From U.S. Pat. No. 5,893,120		William B. Frakes & Ricardo Baeza-Yates, <i>Information Retrieval: Data Structures & Algorithms</i> (Prentice-Hall, Inc. 1992) (“Frakes”) 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.

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Brown discloses an information storage and retrieval system.</p> <p>For example, Brown discloses an information storage and retrieval system made up of a hash table of linked lists. <i>See, e.g.</i>, Brown at 60-62, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p>
<p>[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,</p>	<p>[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,</p>	<p>Brown 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. Brown 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. <i>See, e.g.</i>, Brown at 33-34, 60-62, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p> <p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection.” <i>See</i> Brown at 33-34.</p>
<p>[1b] a record search means utilizing a search key to</p>	<p>[5b] a record search means utilizing a search key to</p>	<p>Brown discloses a record search means utilizing a search key to access the linked list. Brown also discloses a record search means utilizing a search key</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
<p>access the linked list,</p>	<p>access a linked list of records having the same hash address,</p>	<p>to access a linked list of records having the same hash address. <i>See, e.g.</i>, Brown at 60-62, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 . . . Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p>
<p>[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</p>	<p>[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</p>	<p>Brown 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. Brown 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. <i>See, e.g.</i>, Brown at 33, 60-62, 66-68, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 . . . Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p> <p>“The value array and the key heap grow towards each other, such that the maximum number of entries in a bucket is variable. The array and heap entries are paired-up from inside out, eliminating the need for string heap offsets in the value array entries and minimizing the amount of space required by the key/value pairs (compression techniques excluded). The tradeoff is a more complex bucket and search algorithm. To find a key/value pair in a bucket, we must scan the bucket’s key heap from left to right . . .” <i>See</i> Brown at 61.</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection.” <i>See</i> Brown at 33-34.</p>
<p>[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.</p>	<p>[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.</p>	<p>Brown 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. Brown 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. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p> <p>“The value array and the key heap grow towards each other, such that the maximum number of entries in a bucket is variable. The array and heap entries are paired-up from inside out, eliminating the need for string heap offsets in the value array entries and minimizing the amount of space required by the key/value pairs (compression techniques excluded). The tradeoff is a more complex bucket and search algorithm. To find a key/value pair in a bucket, we must scan the bucket’s key heap from left to right” <i>See</i> Brown at 61.</p> <p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition operations.” <i>See</i> Brown at 33-34.</p> <p>“In the third approach, all of the inverted lists in the inverted file are scanned and entries for the deleted document are removed from inverted lists as they are found The scan of the inverted file is driven at the object level and is supported by Mneme’s object scanning facility. This facility allows an object pool to iterate through its objects in order of object identifier.” <i>See</i> Brown at 67.</p>
<p>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.</p>	<p>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.</p>	<p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Brown 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 Brown 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 Brown 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</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>the removal to complete. Indeed, part of the motivation for the system disclosed in Brown 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.</p> <p>Brown 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Brown 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 Brown. 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 Brown nothing more than</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>the predictable use of prior art elements according to their established functions.</p> <p>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 Brown and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Brown 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.</p> <p>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 Brown 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Brown</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Brown 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 Brown 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 Brown with Thatte.</p> <p>Alternatively, it would also be obvious to combine Brown 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:</p> <p style="padding-left: 40px;">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”).</p> <p>In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-13

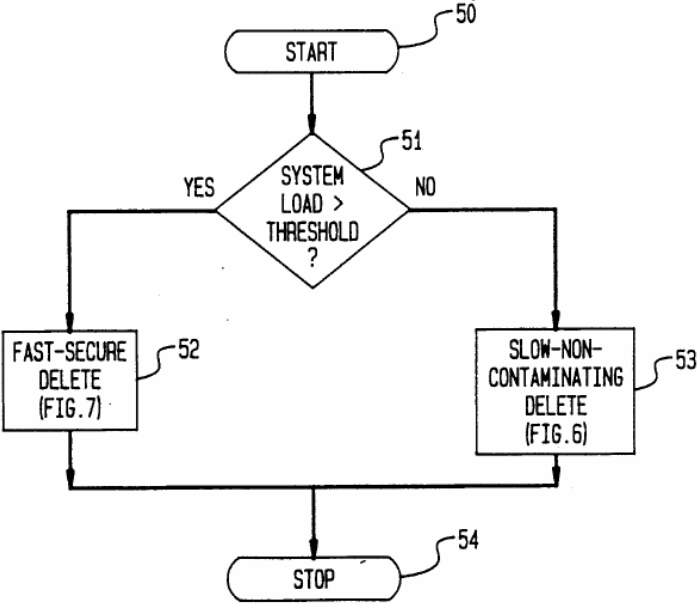
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>	
		<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33,</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Brown and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Brown. 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 Brown would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>based on a systems load as taught by the '663 patent and with Brown 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.</p> <p>Alternatively, it would also be obvious to combine Brown with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Brown 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 Brown. 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</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120	Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
	<p>procedure with Brown would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Brown and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Brown 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 Brown 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 Brown 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.</p> <p>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</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Brown 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. Brown 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. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 . . . Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p> <p>“The value array and the key heap grow towards each other, such that the maximum number of entries in a bucket is variable. The array and heap entries are paired-up from inside out, eliminating the need for string heap offsets in the value array entries and minimizing the amount of space required by the key/value pairs (compression techniques excluded). The tradeoff is a more complex bucket and search algorithm. To find a key/value pair in a bucket, we must scan the bucket’s key heap from left to right . . .” <i>See</i> Brown at 61.</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition operations.” <i>See</i> Brown at 33-34.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Brown discloses accessing a linked list of records. Brown also discloses accessing a linked list of records having same hash address. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“We have implemented a Mneme-based hash table for our inverted File Manager using the overall structure shown in Figure 3.5 Each slot points to a linked list of buckets, which contain the key/value pairs for the keys that hash to that slot.” <i>See</i> Brown at 60-62, Fig. 3.5.</p> <p>“The value array and the key heap grow towards each other, such that the maximum number of entries in a bucket is variable. The array and heap entries are paired-up from inside out, eliminating the need for string heap offsets in the value array entries and minimizing the amount of space required by the key/value pairs (compression techniques excluded). The tradeoff is a more complex bucket and search algorithm. To find a key/value pair in a bucket, we must scan the bucket’s key heap from left to right”</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition operations.” <i>See</i> Brown at 33-34.</p> <p>“In the third approach, all of the inverted lists in the inverted file are scanned and entries for the deleted document are removed from inverted lists as they are found The scan of the inverted file is driven at the object level and is supported by Mneme’s object scanning facility. This facility allows an object pool to iterate through its objects in order of object identifier.” <i>See</i> Brown at 67.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Brown discloses identifying at least some of the automatically expired ones of the records. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>operations.” <i>See</i> Brown at 33-34.</p> <p>“In the third approach, all of the inverted lists in the inverted file are scanned and entries for the deleted document are removed from inverted lists as they are found The scan of the inverted file is driven at the object level and is supported by Mneme’s object scanning facility. This facility allows an object pool to iterate through its objects in order of object identifier.” <i>See</i> Brown at 67.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Brown discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition operations.” <i>See</i> Brown at 33-34.</p> <p>“In the third approach, all of the inverted lists in the inverted file are scanned and entries for the deleted document are removed from inverted lists as they are found The scan of the inverted file is driven at the object level and is supported by Mneme’s object scanning facility. This facility allows an object pool to iterate through its objects in order of object identifier.” <i>See</i> Brown at</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
		<p>67.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Brown discloses inserting, retrieving or deleting one of the records from the system following the step of removing. <i>See, e.g.</i>, Brown at 33-34, 60-62, 66-68, Fig. 3.5.</p> <p>“The ability to modify an existing document collection is a natural requirement for any information retrieval system Additionally, old news articles will eventually expire and must be deleted from the current events document collection. Articles may expire either because their content is relevant only for a certain period of time, or because the size of the current events collection must be held below some threshold due to performance requirements or capacity limitations. Expired articles will either be discarded or archived in a larger secondary document collection, leading to further document addition operations.” <i>See</i> Brown at 33-34.</p> <p>“In the third approach, all of the inverted lists in the inverted file are scanned and entries for the deleted document are removed from inverted lists as they are found The scan of the inverted file is driven at the object level and is supported by Mneme’s object scanning facility. This facility allows an object pool to iterate through its objects in order of object identifier.” <i>See</i> Brown at 67.</p> <p>“Should all of the document entries be deleted from an inverted list, the list’s object can be freed and the corresponding term can be deleted from the term hash table The long inverted lists are processed next. The page-object pool that contains the link list object is scanned, giving us the first object of each long inverted list.” <i>See</i> Brown at 67-68.</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
<p>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.</p>	<p>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.</p>	<p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Brown 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 Brown 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 Brown 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 Brown 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.</p> <p>Brown combined with Dirks, Thatte, the ’663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>number of expired ones of the records to remove when the linked list is accessed.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120		Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>As both Brown 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 Brown. 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 Brown would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Brown and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Brown 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</p>

EXHIBIT C-13

Asserted Claims From U.S. Pat. No. 5,893,120	Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination
	<p>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.</p> <p>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 Brown 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Brown with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Brown 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 Brown 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 Brown with Thatte.</p> <p>Alternatively, it would also be obvious to combine Brown with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly shown in the</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>chart of the '663 patent, which is hereby incorporated by reference in its entirety. As summarized in the '663 patent:</p> <p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-13

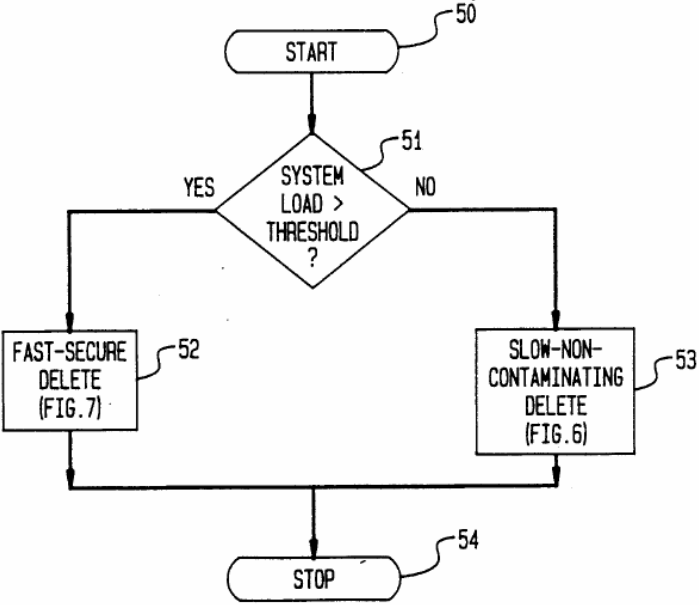
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33,</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Brown and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Brown. 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 Brown would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>based on a systems load as taught by the '663 patent and with Brown 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.</p> <p>Alternatively, it would also be obvious to combine Brown with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-13

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Brown 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 Brown. 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</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>procedure with Brown would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Brown 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Brown 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 Brown 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 Brown 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.</p> <p>One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the</p>

EXHIBIT C-13

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Eric W. Brown, <i>Execution Performance Issues in Full Text Information Retrieval</i>, University of Massachusetts Amherst (October 1995) (hereinafter “Brown”) alone and in combination</p>
	<p>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.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Costello discloses an information storage and retrieval system.</p> <p>For example, Costello describes a method and system for storing and retrieving callouts. See, e.g., Costello, page 3-4. See also, Costello Presentation, page 3, 15-18, and 26.</p> <p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” See, Costello, page 4.</p>
<p>[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,</p>	<p>[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,</p>	<p>Costello 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. Costello 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.</p> <p>For example, Costello describes storing outstanding callouts in a linked list. See, e.g., Costello, page 3, 7. The entries are removed when the callout is expired. <i>Id.</i> See also, Costello Presentation, page 3, 15-18, and 26.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>For example, Costello describes storing the callouts in both a circular array of linked lists and a hash table of linked lists. See, e.g., <i>Costello</i>, page 3. <i>Id.</i> at 4. See also, <i>Costello Presentation</i>, page 3, 15-18, and 26.</p> <p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, <i>Costello</i>, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” See, <i>Costello</i>, page 4.</p> <p>“At first, we used a closed-chaining hash table.” See, <i>Costello</i>, page 4.</p> <p>“We wanted both sorts of calls to have equal costs in the new implementation as well, so we switched to an open-chaining hash table, in which only one bucket needs to be searched in any case.” See, <i>Costello</i>, page 7.</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Costello discloses a record search means utilizing a search key to access the linked list. Costello also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, Costello describes using both a circular array and a hash table. See, e.g., <i>Costello</i>, page 3, 7. In either case, a search key is utilized to access the linked list of callouts. <i>Id.</i> at 4. See also, <i>Costello Presentation</i>, page 3, 15-18, and 26.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, <i>et al.</i>, <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). <i>See also</i>, Costello, Adam, <i>et al.</i>, <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” <i>See</i>, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” <i>See</i>, Costello, page 4.</p> <p>“At first, we used a closed-chaining hash table.” <i>See</i>, Costello, page 4.</p> <p>“We wanted both sorts of calls to have equal costs in the new implementation as well, so we switched to an open-chaining hash table, in which only one bucket needs to be searched in any case.” <i>See</i>, Costello, page 7.</p>
<p>[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</p>	<p>[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</p>	<p>Costello 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. Costello 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.</p> <p>For example, Costello describes identifying the expired callouts and removing them from the linked list when it is accessed. <i>See, e.g.</i>, Costello, page 3-4, 7. <i>See also</i>, Costello Presentation, page 3, 15-18, and 26.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, Costello, page 3.</p> <p>“If the first callout has expired, a software clock interrupt is generated. Its handler, <code>softclock()</code>, repeatedly checks the callout at the head of the list, and if it is expired (<code>c_time</code> member equal to zero), removes it and calls its function.” See, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” See, Costello, page 4.</p> <p>“At first, we used a closed-chaining hash table.” See, Costello, page 4.</p> <p>“We wanted both sorts of calls to have equal costs in the new implementation as well, so we switched to an open-chaining hash table, in which only one bucket needs to be searched in any case.” See, Costello, page 7.</p>
<p>[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.</p>	<p>[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</p>	<p>Costello 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. Costello 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.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
	<p>records in the accessed linked list of records.</p>	<p>For example, Costello describes using the record search means to access the linked list and retrieving and removing expired callouts from the linked list at the same time. See, e.g., Costello, page 3-4. See also, Costello Presentation, page 3, 15-18, and 26.</p> <p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, Costello, page 3.</p> <p>“If the first callout has expired, a software clock interrupt is generated. Its handler, <code>softclock()</code>, repeatedly checks the callout at the head of the list, and if it is expired (<code>c_time</code> member equal to zero), removes it and calls its function.” See, Costello, page 3.</p>
<p>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.</p>	<p>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.</p>	<p>Costello 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.</p> <p>For example, Costello describes limiting the number of records removed from the linked list during a particular sequence using a <code>max_softclock_steps</code> variable. See, e.g., Costello, page 8.</p> <p>“<code>softclock()</code> keeps track of the number of steps it has taken since it last enabled interrupts, and whenever the count reaches <code>MAX_SOFTCLOCK_STEPS</code>, it briefly enables them. Therefore, <code>softclock()</code> never disables interrupts for more than a constant amount of</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>time.” See, e.g., <i>Costello</i>, page 8.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Costello 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 Costello 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 Costello 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 Costello 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.</p>
<p>3. A method for storing and retrieving information</p>	<p>7. A method for storing and retrieving information</p>	<p>To the extent the preamble is a limitation, Costello discloses a method for storing and retrieving information records using a linked list to store and</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
<p>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:</p>	<p>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:</p>	<p>provide access to the records, at least some of the records automatically expiring. Costello 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.</p> <p>For example, Costello describes a method and system for storing and retrieving callouts which expire automatically based on timers. See, e.g., Costello, page 3-4. See also, Costello Presentation, page 3, 15-18, and 26.</p> <p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” See, Costello, page 4.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Costello discloses accessing a linked list of records. Costello also discloses accessing a linked list of records having same hash address.</p> <p>For example, Costello describes using both a circular array and a hash table. See, e.g., Costello, page 3-4, 7. In either case, a search key is utilized to access the linked list of callouts. <i>Id.</i> at 4. See also, Costello Presentation, page 3, 15-18, and 26.</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, <i>et al.</i>, <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). <i>See also</i>, Costello, Adam, <i>et al.</i>, <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” <i>See</i>, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” <i>See</i>, Costello, page 4.</p> <p>“At first, we used a closed-chaining hash table.” <i>See</i>, Costello, page 4.</p> <p>“We wanted both sorts of calls to have equal costs in the new implementation as well, so we switched to an open-chaining hash table, in which only one bucket needs to be searched in any case.” <i>See</i>, Costello, page 7.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Costello discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, the entries are removed when the callout is expired. <i>See, e.g.</i>, Costello, page 3-4, 7. <i>See also</i>, Costello Presentation, page 3, 15-18, and 26.</p> <p>“All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” <i>See</i>, Costello, page 3.</p> <p>“If the first callout has expired, a software clock interrupt is generated. Its handler, <code>softclock()</code>, repeatedly checks the callout at the head of the list,</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, <i>et al.</i>, <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). <i>See also</i>, Costello, Adam, <i>et al.</i>, <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>and if it is expired (<code>c_time</code> member equal to zero), removes it and calls its function.” <i>See</i>, Costello, page 3.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Costello discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, Costello describes identifying the expired callouts and removing them from the linked list when it is accessed. <i>See, e.g.</i>, Costello, page 3-4. <i>See also</i>, Costello Presentation, page 3, 15-18, and 26.</p> <p>“All callouts scheduled to expire at time <code>t</code> appear in the list <code>callwheel [t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” <i>See</i>, Costello, page 3.</p> <p>“If the first callout has expired, a software clock interrupt is generated. Its handler, <code>softclock()</code>, repeatedly checks the callout at the head of the list, and if it is expired (<code>c_time</code> member equal to zero), removes it and calls its function.” <i>See</i>, Costello, page 3.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Costello discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Costello describes using the record search means to access the linked list and insert, retrieving, or delete expired callouts from the linked list following the step of removing expired callouts <i>See, e.g.</i>, Costello, page 3-4, 7. <i>See also</i>, Costello Presentation, page 3, 15-18, and 26.</p> <p>“Instead of a single sorted list of callout structures, we use a circular array of unsorted lists. The array, called a <code>callwheel</code> (see Figure 2), contains <code>callwheelsize</code> entries. All callouts scheduled to expire at time <code>t</code> appear in</p>

EXHIBIT C-14

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Costello, Adam, et al., <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). See also, Costello, Adam, et al., <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
		<p>the list <code>callwheel[t% callwheelsize]</code>, and their <code>c_time</code> members are set to <code>t/callwheelsize</code>.” See, Costello, page 3.</p> <p>“If the first callout has expired, a software clock interrupt is generated. Its handler, <code>softclock()</code>, repeatedly checks the callout at the head of the list, and if it is expired (<code>c_time</code> member equal to zero), removes it and calls its function.” See, Costello, page 3.</p> <p>“We could try to find a way, given a function pointer and argument pointer, to produce a pointer to the matching callout in constant time. A hash table is the obvious mechanism.” See, Costello, page 4.</p> <p>“At first, we used a closed-chaining hash table.” See, Costello, page 4.</p> <p>“We wanted both sorts of calls to have equal costs in the new implementation as well, so we switched to an open-chaining hash table, in which only one bucket needs to be searched in any case.” See, Costello, page 7.</p>
<p>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.</p>	<p>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.</p>	<p>Costello discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, Costello describes limiting the number of records removed from the linked list during a particular sequence using a <code>max_softclock_steps</code> variable. See, e.g., Costello, page 8.</p> <p>“<code>softclock()</code> keeps track of the number of steps it has taken since it last enabled interrupts, and whenever the count reaches <code>MAX_SOFTCLOCK_STEPS</code>, it briefly enables them. Therefore, <code>softclock()</code> never disables interrupts for more than a constant amount of</p>

EXHIBIT C-14

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Costello, Adam, <i>et al.</i>, <i>Redesigning the BSD - Callout and Time Facilities</i>, WUSC 95-23, November 2, 1995 (“Costello”). <i>See also</i>, Costello, Adam, <i>et al.</i>, <i>Presentation - Redesigning the BSD Unix Callout and Time Facilities</i>, January 10, 1996 (“Costello Presentation”).</p>
	<p>time.” <i>See</i>, Costello, page 8.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Costello 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 Costello 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 Costello 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 Costello 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.</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Foster discloses an information storage and retrieval system. <i>See, e.g.</i>, Foster at 4-12, 24-26, 33-40.</p> <p>“A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See</i> Foster at 25.</p> <p>“When a name is read, the appropriate list is selected and searched. If 26 is a suitable value for <i>n</i> and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See</i> Foster at 25.</p>
<p>[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,</p>	<p>[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,</p>	<p>Foster 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. Foster 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. <i>See, e.g.</i>, Foster at 4-12, 33-40.</p> <p>“A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See</i> Foster at 25.</p> <p>“When a name is read, the appropriate list is selected and searched. If 26 is a suitable value for <i>n</i> and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See</i> Foster at 25.</p> <p>“But most list processing problems will require more store than can be made available in this straightforward manner, and something has to be done to</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>enable the re-use of stores of which the contents are no longer needed. . . . All list processing languages provide, either explicitly in the language or implicitly in the system, some method of reclaiming the store.” <i>See</i> Foster at 33.</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Foster discloses a record search means utilizing a search key to access the linked list. Foster also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. <i>See, e.g.,</i> Foster at 24-26, 33-38.</p> <p>“A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See</i> Foster at 25.</p> <p>“When a name is read, the appropriate list is selected and searched. If 26 is a suitable value for <i>n</i> and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See</i> Foster at 25.</p>
<p>[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</p>	<p>[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</p>	<p>Foster 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. Foster 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. <i>See, e.g.,</i> Foster at 35-38.</p> <p>“A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See</i> Foster at 25.</p> <p>“When a name is read, the appropriate list is selected and searched. If 26 is a</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>suitable value for n and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See</i> Foster at 25.</p> <p>“But most list processing problems will require more store than can be made available in this straightforward manner, and something has to be done to enable the re-use of stores of which the contents are no longer needed. . . . All list processing languages provide, either explicitly in the language or implicitly in the system, some method of reclaiming the store.” <i>See</i> Foster at 33.</p> <p>“A convention, which has been adopted in order to make the memory of the wanted cells easier to the user, is to say that a list is owned in one place only. If it appears in other places it is being borrowed. The list that is the owner is responsible for declaring that the cells are not wanted.” <i>See</i> Foster at 35.</p>
<p>[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.</p>	<p>[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.</p>	<p>Foster 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. Foster 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. <i>See, e.g.,</i> Foster at 33-40.</p> <p>“But most list processing problems will require more store than can be made available in this straightforward manner, and something has to be done to enable the re-use of stores of which the contents are no longer needed. . . . All list processing languages provide, either explicitly in the language or implicitly in the system, some method of reclaiming the store.” <i>See</i> Foster at 33.</p> <p>“A convention, which has been adopted in order to make the memory of the</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>wanted cells easier to the user, is to say that a list is owned in one place only. If it appears in other places it is being borrowed. The list that is the owner is responsible for declaring that the cells are not wanted.” <i>See Foster</i> at 35.</p> <p>“Hence the process of garbage collection has to proceed in two stages, the first of which goes through all of the lists dependent on the list heads and marks them as wanted. The second then scans the whole area allotted to lists and returns the unmarked cells to the free list, simultaneously unmarking the marked cells ready for next time.” <i>See Foster</i> at 35.</p>
<p>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.</p>	<p>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.</p>	<p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Foster 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 Foster 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 Foster 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 Foster 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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>“[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.</p> <p>Foster 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
	<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Foster 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 Foster. 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 Foster nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Foster and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Foster 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.</p> <p>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 Foster 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Foster with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Foster 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 Foster 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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>many records to delete can be a dynamic one." '120 at 7:10-15. Thus, the '120 patent provides motivations to combine Foster with Thatte.</p> <p>Alternatively, it would also be obvious to combine Foster 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:</p> <p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p>

EXHIBIT C-15

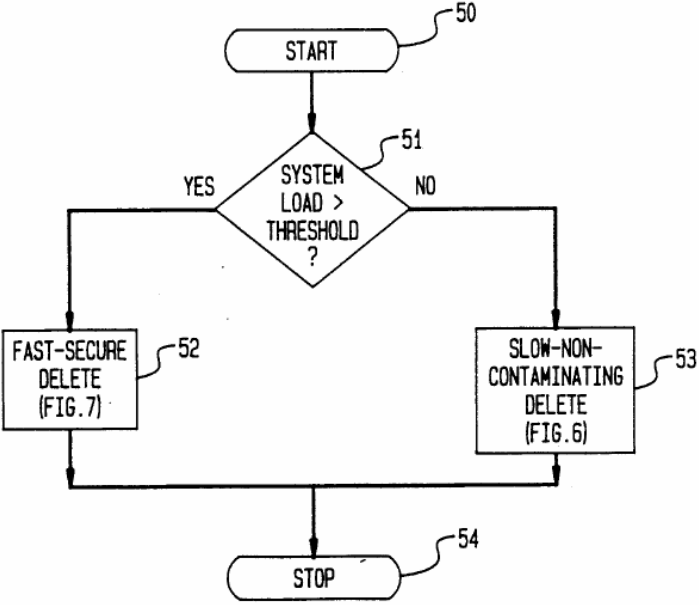
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
	<p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Foster and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Foster. 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 Foster would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>based on a systems load as taught by the '663 patent and with Foster 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.</p> <p>Alternatively, it would also be obvious to combine Foster with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>This decision routine should take several things into account: 1) the volume</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Foster 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 Foster. 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 Foster would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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 Foster and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Foster 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 Foster 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 Foster 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.</p> <p>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</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Foster 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. Foster 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. <i>See, e.g.</i>, Foster at 4-12, 24-26, and 33-40.</p> <p>“A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See</i> Foster at 25.</p> <p>“When a name is read, the appropriate list is selected and searched. If 26 is a suitable value for n and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See</i> Foster at 25.</p> <p>But most list processing problems will require more store than can be made available in this straightforward manner, and something has to be done to enable the re-use of stores of which the contents are no longer needed. . . . All list processing languages provide, either explicitly in the language or implicitly in the system, some method of reclaiming the store.” <i>See</i> Foster at 33.</p> <p>“A convention, which has been adopted in order to make the memory of the wanted cells easier to the user, is to say that a list is owned in one place only.</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120		J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
		If it appears in other places it is being borrowed. The list that is the owner is responsible for declaring that the cells are not wanted.” <i>See Foster</i> at 35.
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	Foster discloses accessing a linked list of records. Foster also discloses accessing a linked list of records having same hash address. <i>See, e.g., Foster</i> at 4-12, 24-26. “A very important use of the vector of lists is in one of the methods for doing ‘hash coding’. The problem is to find something which has been associated with an object, on being presented with the object itself.” <i>See Foster</i> at 25. “When a name is read, the appropriate list is selected and searched. If 26 is a suitable value for <i>n</i> and the unevenness of distribution of initial letters is not thought to matter, then these could serve as the index for the lists.” <i>See Foster</i> at 25.
[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,	Foster discloses identifying at least some of the automatically expired ones of the records. <i>See Foster</i> at 35-38. “But most list processing problems will require more store than can be made available in this straightforward manner, and something has to be done to enable the re-use of stores of which the contents are no longer needed. . . . All list processing languages provide, either explicitly in the language or implicitly in the system, some method of reclaiming the store.” <i>See Foster</i> at 33. “A convention, which has been adopted in order to make the memory of the wanted cells easier to the user, is to say that a list is owned in one place only. If it appears in other places it is being borrowed. The list that is the owner is

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>responsible for declaring that the cells are not wanted.” <i>See Foster</i> at 35.</p> <p>“Hence the process of garbage collection has to proceed in two stages, the first of which goes through all of the lists dependent on the list heads and marks them as wanted. The second then scans the whole area allotted to lists and returns the unmarked cells to the free list, simultaneously unmarking the marked cells ready for next time.” <i>See Foster</i> at 35.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Foster discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. <i>See Foster</i> at 35-38.</p> <p>“Hence the process of garbage collection has to proceed in two stages, the first of which goes through all of the lists dependent on the list heads and marks them as wanted. The second then scans the whole area allotted to lists and returns the unmarked cells to the free list, simultaneously unmarking the marked cells ready for next time.” <i>See Foster</i> at 35.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>Foster discloses inserting, retrieving or deleting one of the records from the system following the step of removing. <i>See Foster</i> at 4-12, 22-29, 33-40.</p> <p>“Hence the process of garbage collection has to proceed in two stages, the first of which goes through all of the lists dependent on the list heads and marks them as wanted. The second then scans the whole area allotted to lists and returns the unmarked cells to the free list, simultaneously unmarking the marked cells ready for next time.” <i>See Foster</i> at 35.</p> <p>It would be obvious to a person of skill in the art to perform known functions such as an insertion, deletion, or retrieval on the linked list after the step of</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120		J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
		removing is performed.
<p>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.</p>	<p>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.</p>	<p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Foster 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 Foster 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 Foster 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 Foster 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.</p> <p>Foster combined with Dirks, Thatte, the ’663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining maximum</p>

EXHIBIT C-15

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
	<p>number of expired ones of the records to remove when the linked list is accessed.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Foster 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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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 Foster. 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 Foster would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Foster and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Foster 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.</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
		<p>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 Foster 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Foster with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Foster 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 Foster 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 Foster with Thatte.</p> <p>Alternatively, it would also be obvious to combine Foster 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:</p> <p align="center">during normal times when the load on the storage system is not</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-15

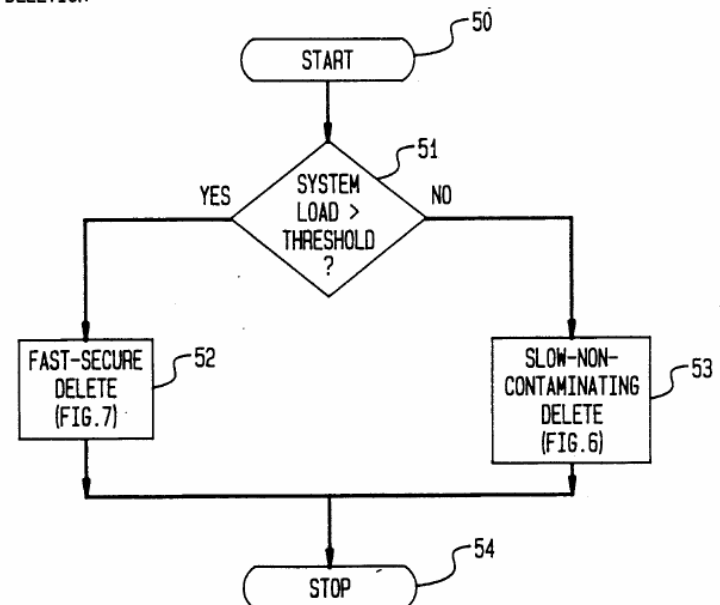
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter "Foster") alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-</p>

EXHIBIT C-15

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination</p>
	<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Foster and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Foster. 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 Foster would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Foster and would have seen the benefits of doing so. One such benefit, for example, is</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>that the system would avoid performing deletions when the system load exceeded a threshold.</p> <p>Alternatively, it would also be obvious to combine Foster with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Foster 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 Foster. 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 Foster would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) alone and in combination
	<p>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 Foster 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Foster 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 Foster 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 Foster 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.</p> <p>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</p>

EXHIBIT C-15

Asserted Claims From U.S. Pat. No. 5,893,120	J.M. FOSTER, LIST PROCESSING (Macdonald & Co. 1967) (hereinafter “Foster”) 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.” ‘120 at 7:10-15.

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation Keshav discloses an information storage and retrieval system.</p> <p>For example,</p> <p>“The algorithm is implemented at the server that schedules packets on the output trunk of a router or switch in a store-and-forward network.” Srinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i> (hereinafter “Keshav”) at 2.</p>
<p>[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,</p>	<p>[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,</p>	<p>Keshav 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. Keshav 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.</p> <p>For example,</p> <p>“Buffering Alternatives We considered four buffering schemes: an ordered linked list (LINK), a binary tree (TREE), a double heap (HEAP), and a combination of per-conversation queuing and heaps (PERC). We expect that the reader is familiar with details of the list, tree and heap data structures. They are also described in standard texts such as References [10, 11].</p> <p>Ordered List Tag values usually increase with time, since bid numbers are strictly monotonic within each conversation. This suggests that packets should be</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>buffered in a ordered linked list, inserting incoming packets by linearly scanning from the largest tag value.” Keshav at 8.</p> <p>“If all the buffers are full, the server drops the packet with the largest bid number (unlike the algorithm in Reference [1], this buffer allocation policy accounts for differences in packet lengths). The abstract data structure required for packet buffering is a bounded heap. A bounded heap is named by its root, and contains a set of packets that are tagged by their bid number.” Keshav at 7.</p> <p>“The conversation ID is used to access a data structure for storing state. Since IDs could span large address spaces, the standard solution is to hash the ID onto a index, and the technology for this is well known [9]. Recently, a simple and efficient hashing scheme that ignores hash collections has been proposed [5]. In this approach, some conversations could share the same state, leading to unfair service, since these conversations are served first-come-first-served. However, this is attenuated by occasionally perturbing the hash function.” Keshav at 5.</p> <p>“Assume for the moment that data from each source destination pair (a conversation) can be distinguished, and is stored in a logically distinct per-conversation queue.” Keshav at 2.</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Keshav discloses a record search means utilizing a search key to access the linked list. Keshav also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example,</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>“Assume for the moment that data from each source destination pair (a conversation) can be distinguished, and is stored in a logically distinct per-conversation queue.” Keshav at 2.</p> <p>The paper discloses in detail how to generate this key in a unique and coherent manner. <i>See</i> page 4:</p> <p>“The choice of the conversation ID depends on the entity to whom fair service is granted (see the discussion in Reference [1]), and the naming space of the network. For example, if the unit is a transport connection in the IP Internet, one such unique identifier is the tuple (source address, destination address, source port number, destination port number, protocol type).” Keshav at 4.</p> <p>“The conversation ID is used to access a data structure for storing state. Since IDs could span large address spaces, the standard solution is to hash the ID onto a index, and the technology for this is well known [9]. Recently, a simple and efficient hashing scheme that ignores hash collections has been proposed [5]. In this approach, some conversations could share the same state, leading to unfair service, since these conversations are served first-come-first-served. However, this is attenuated by occasionally perturbing the hash function.” Keshav at 5.</p>
<p>[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</p>	<p>[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</p>	<p>Keshav 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. Keshav 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.</p> <p>For example,</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>“Assume for the moment that data from each source destination pair (a conversation) can be distinguished, and is stored in a logically distinct per-conversation queue.” Keshav at 2.</p> <p>“The choice of the conversation ID depends on the entity to whom fair service is granted (see the discussion in Reference [1]), and the naming space of the network. For example, if the unit is a transport connection in the IP Internet, one such unique identifier is the tuple (source address, destination address, source port number, destination port number, protocol type).” Keshav at 4.</p> <p>“The conversation ID is used to access a data structure for storing state.” Keshav at 5.</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free apace in the buffer to accommodate a maximum sized packet.” Keshav at 7.</p>
<p>[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.</p>	<p>[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.</p>	<p>Keshav 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. Keshav 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.</p> <p>For example,</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free [s]pace in the buffer to accommodate a maximum sized packet.” Keshav at 7.</p>
<p>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.</p>	<p>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.</p>	<p>Keshav 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.</p> <p>For example,</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to hand this case if the item is already in the heap. To allow this, we always keep enough free [s]pace in the buffer to accommodate a maximum sized packet, get_min () returns a pointer to the item with the smallest tag value and deletes it.” Keshav at 7.</p> <p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Keshav 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 Keshav with the fundamental concept of dynamically determining the maximum number of</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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 Keshav 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 Keshav 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.</p> <p>Keshav 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.</p> <p>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.</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>As summarized in Dirks,</p> <p>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.</p> <p>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:</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Keshav 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 Keshav. 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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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 Keshav nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Keshav and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Keshav 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.</p> <p>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 Keshav 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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>maximum number for the record search means to remove.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Keshav with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Keshav 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 Keshav 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 Keshav with Thatte.</p> <p>Alternatively, it would also be obvious to combine Keshav 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>4,996,663 to Nemes at 2:24-34 (“The ’663 patent”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-16

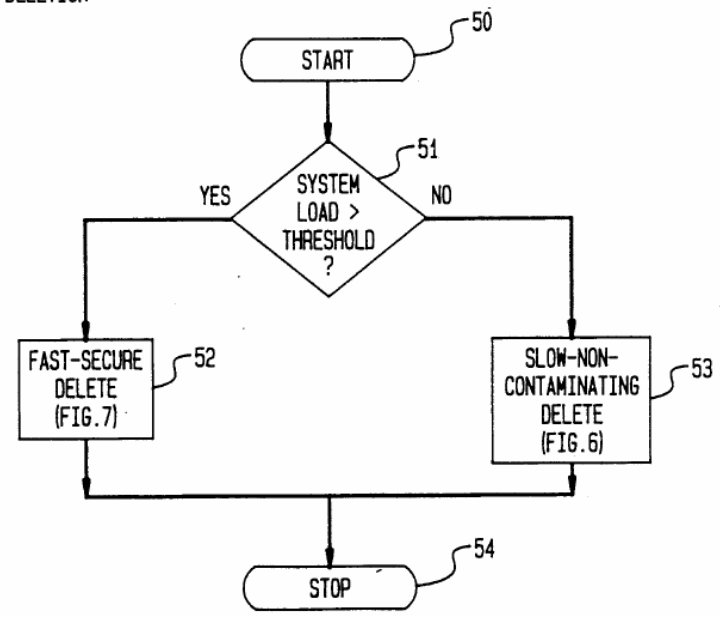
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33,</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Keshav and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Keshav. 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 Keshav would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Keshav and</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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.</p> <p>Alternatively, it would also be obvious to combine Keshav with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Keshav 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 Keshav. 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 Keshav would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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 Keshav and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Keshav 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 Keshav 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 Keshav 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.</p> <p>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</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Keshav 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. Keshav 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.</p> <p>For example,</p> <p>“The algorithm is implemented at the server that schedules packets on the output trunk of a router or switch in a store-and-forward network.” Keshav at 2.</p> <p>“Buffering Alternatives We considered four buffering schemes: an ordered linked list (LINK), a binary tree (TREE), a double heap (HEAP), and a combination of per-conversation queuing and heaps (PERC). We expect that the reader is familiar with details of the list, tree and heap data structures. They are also described in standard texts such as References [10, 11].</p> <p>Ordered List Tag values usually increase with time, since bid numbers are strictly monotonic within each conversation. This suggests that packets should be buffered in a ordered linked list, inserting incoming packets by linearly scanning from the largest tag value.” Keshav at 8.</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>“If all the buffers are full, the server drops the packet with the largest bid number (unlike the algorithm in Reference [1], this buffer allocation policy accounts for differences in packet lengths). The abstract data structure required for packet buffering is a bounded heap. A bounded heap is named by its root, and contains a set of packets that are tagged by their bid number.” Keshav at 7.</p> <p>“The conversation ID is used to access a data structure for storing state. Since IDs could span large address spaces, the standard solution is to hash the ID onto a index, and the technology for this is well known [9]. Recently, a simple and efficient hashing scheme that ignores hash collections has been proposed [5]. In this approach, some conversations could share the same state, leading to unfair service, since these conversations are served first-come-first-served. However, this is attenuated by occasionally perturbing the hash function.” Keshav at 5.</p> <p>“Assume for the moment that data from each source destination pair (a conversation) can be distinguished, and is stored in a logically distinct per-conversation queue.” Keshav at 2.</p> <p>“The choice of the conversation ID depends on the entity to whom fair service is granted (see the discussion in Reference [1]), and the naming space of the network. For example, if the unit is a transport connection in the IP Internet, one such unique identifier is the tuple (source address, destination address, source port number, destination port number, protocol type).” Keshav at 4.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Keshav discloses accessing a linked list of records. Keshav also discloses accessing a linked list of records having same hash address.</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>For example,</p> <p>“Assume for the moment that data from each source destination pair (a conversation) can be distinguished, and is stored in a logically distinct per-conversation queue.” Keshav at 2.</p> <p>“The choice of the conversation ID depends on the entity to whom fair service is granted (see the discussion in Reference [1]), and the naming space of the network. For example, if the unit is a transport connection in the IP Internet, one such unique identifier is the tuple (source address, destination address, source port number, destination port number, protocol type).” Keshav at 4.</p> <p>The conversation ID is used as hash key to get to the conversation data records (<i>i.e.</i> having the same hash address). <i>See</i> page 5, line 9:</p> <p>“The conversation ID is used to access a data structure for storing state. Since IDs could span large address spaces, the standard solution is to hash the ID onto a index, and the technology for this is well known [9]. Recently, a simple and efficient hashing scheme that ignores hash collections has been proposed [5]. In this approach, some conversations could share the same state, leading to unfair service, since these conversations are served first-come-first-served. However, this is attenuated by occasionally perturbing the hash function.” Keshav at 5.</p> <p>“Buffering Alternatives We considered four buffering schemes: an ordered linked list (LINK), a binary tree (TREE), a double heap (HEAP), and a combination of per-conversation queuing and heaps (PERC). We expect that the reader is familiar with details of the list, tree and heap data structures. They are also described in standard</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>texts such as References [10, 11].</p> <p>Ordered List Tag values usually increase with time, since bid numbers are strictly monotonic within each conversation. This suggests that packets should be buffered in a ordered linked list, inserting incoming packets by linearly scanning from the largest tag value.” Keshav at 8.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Keshav discloses identifying at least some of the automatically expired ones of the records. Keshav also discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example,</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free [s]pace in the buffer to accommodate a maximum sized packet.” Keshav at 7.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Keshav discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. Keshav also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example,</p> <p>“The abstract data structure required for packet buffering is a bounded heap. A bounded heap is named by its root, and contains a set of packets that are tagged by their bid number. It is associated with two operations, insert (root, item,</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>conversation_ID) and get_min(root), and a parameter, MAX, which is the maximum size of the heap.</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free space in the buffer to accommodate a maximum sized packet.” Keshav at 7.</p>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>Keshav discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to handle this case if the item is already in the heap. To allow this, we always keep enough free space in the buffer to accommodate a maximum sized packet get_min () returns a pointer to the item with the smallest tag value and deletes.” Keshav at 7.</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
<p>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.</p>	<p>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.</p>	<p>Keshav discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example,</p> <p>“insert () first places an item in the bounded heap. While the heap size exceeds MAX, it repeatedly discards the item with the largest tag value. We insert an item before removing the largest item since the inserted packet itself may be deleted, and it is easier to hand this case if the item is already in the heap. To allow this, we always keep enough free [s]pace in the buffer to accommodate a maximum sized packet, get_min () returns a pointer to the item with the smallest tag value and deletes it.” Keshav at 7.</p> <p>It would have been obvious to one of ordinary skill in the art to modify the system disclosed in Keshav 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 Keshav 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 Keshav 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 Keshav is avoiding these problems. One of ordinary skill in the art would have known that dynamically determining the maximum number to</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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.</p> <p>Keshav 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>Any other suitable approach can be employed to determine the number of</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
	<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Keshav 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 Keshav. 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 Keshav would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Keshav and would</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Keshav 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.</p> <p>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 Keshav 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Keshav with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Keshav 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 Keshav 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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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 Keshav with Thatte.</p> <p>Alternatively, it would also be obvious to combine Keshav 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-</p>

EXHIBIT C-16

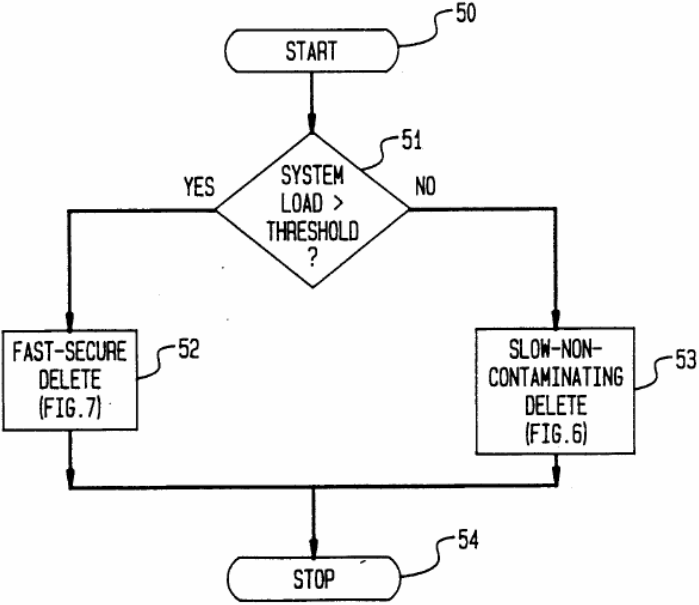
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p>Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
		<p>secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p>

EXHIBIT C-16

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination</p>
	<p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Keshav and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Keshav. 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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>patent’s deletion decision procedure with Keshav would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Keshav 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.</p> <p>Alternatively, it would also be obvious to combine Keshav with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>Every time a user-input routine is invoked, a decision routine can decide</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Keshav 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 Keshav. 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</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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 Keshav would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Keshav 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Keshav 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 Keshav 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 Keshav 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.</p>

EXHIBIT C-16

Asserted Claims From U.S. Pat. No. 5,893,120		Sirinivasan Keshav, <i>On the Efficient Implementation of Fair Queueing</i>, Journal of Internetworking: Research and Experience, 1991 (“Keshav”) alone and in combination
		<p>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.</p>

EXHIBIT C-16

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Varghese and Lauck discloses an information storage and retrieval system. <i>See, e.g.</i>, George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”) at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p>
<p>[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,</p>	<p>[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,</p>	<p>Varghese and Lauck 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.</p> <p>Varghese and Lauck 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. <i>See, e.g.</i>, Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p>

EXHIBIT C-17

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>	
		<p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“PER_TICK_BOOKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” <i>See</i> Varghese and Lauck at p. 31.</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Varghese and Lauck discloses a record search means utilizing a search key to access the linked list. Varghese and Lauck also discloses a record search means utilizing a search key to access a linked list of records having the same hash address. See, e.g., Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” See Varghese and Lauck at p. 30.</p> <p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” See Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” See Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” See Varghese and Lauck at p. 30.</p>
<p>[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</p>	<p>[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</p>	<p>Varghese and Lauck 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. Varghese and Lauck 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. See, e.g., Varghese and Lauck at p.</p>

EXHIBIT C-17

Asserted Claims From U.S. Pat. No. 5,893,120		George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)
when the linked list is accessed, and	the linked list is accessed, and	<p>25-27, 29-31, and Figs. 8-9.</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“PER_TICK_BOOKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is</p>

EXHIBIT C-17

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>	
		<p>always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” <i>See</i> Varghese and Lauck at p. 31.</p>
<p>[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.</p>	<p>[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.</p>	<p>Varghese and Lauck 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. Varghese and Lauck 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. <i>See, e.g.</i>, Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
		<p>“PER_TICK_BOOKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” <i>See</i> Varghese and Lauck at p. 31.</p>
<p>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.</p>	<p>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.</p>	<p>Varghese and Lauck 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. <i>See, e.g.</i>, Varghese and Lauck at p. 28.</p> <p>“The simulation proceeds by processing the earliest event, which in turn may schedule further events. The simulation continues until the event list is empty or some condition (e.g. clock > MAX-SIMULATION-TIME} holds.” <i>See</i>, Varghese and Lauck at p. 28.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Varghese and Lauck 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</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
		<p>disclosed in Varghese and Lauck 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 Varghese and Lauck 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 Varghese and Lauck 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.</p>
<p>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:</p>	<p>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</p>	<p>To the extent the preamble is a limitation, Varghese and Lauck 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. Varghese and Lauck 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. <i>See, e.g.</i>, Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
	<p>automatically expiring, the method comprising the steps of:</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“PER_TICK_BOOKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>hash function for Scheme 6 is insignificant.” <i>See</i> Varghese and Lauck at p. 31.</p> <p>Varghese and Lauck discloses accessing a linked list of records. Varghese and Lauck also discloses accessing a linked list of records having same hash address. <i>See, e.g.</i>, Varghese and Lauck at p. 29-31, and Figs. 8-9.</p> <p>“6.1.1 Scheme 5: Hash Table with Sorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“6.1.2 Scheme 6: Hash Table with Unsorted Lists in each Bucket” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“For example, if the table size is a power of 2, an arbitrary size timer can easily be divided by the table size; the remainder (low order bits) is added to the current time pointer to yield the index within the array. The result of the division (high order bits) is stored in a list pointed to by the index.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“The previous scheme has an obvious analogy to inserting an element in an array using the element value as an index. If there is insufficient memory we can hash the element value to yield an index.” <i>See</i> Varghese and Lauck at p. 30.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Varghese and Lauck discloses identifying at least some of the automatically expired ones of the records. <i>See, e.g.</i>, Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“PER_TICK_BOOKKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
		<p>work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” See Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” See Varghese and Lauck at p. 31.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Varghese and Lauck discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. See, e.g., Varghese and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“PER_TICK_BOOKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” See Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” See Varghese and Lauck at p. 31.</p>
	<p>[7d] inserting, retrieving or deleting one of the</p>	<p>Varghese and Lauck discloses inserting, retrieving or deleting one of the records from the system following the step of removing. See, e.g., Varghese</p>

EXHIBIT C-17

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)</p>
	<p>records from the system following the step of removing.</p>	<p>and Lauck at p. 25-27, 29-31, and Figs. 8-9.</p> <p>“PER_TICK_BOOKKEEPING increments the current time pointer. If the value stored in the array element being pointed to is zero, there is no more work. Otherwise, as in Scheme 2, the top of the list is decremented. If it expires, EXPIRY_PROCESSING is called and the top list element is deleted.” <i>See</i> Varghese and Lauck at p. 30.</p> <p>“Thus the hash distribution in Scheme 6 only controls the “burstiness” (variance) of the latency of PER_TICK_BOOKEEPING, and not the average latency. Since the worst-case latency of PER_TICK_BOOKEEPING is always $O(n)$ (all timers expire at the same time), we believe that the choice of hash function for Scheme 6 is insignificant.” <i>See</i> Varghese and Lauck at p. 31.</p> <p>It would be obvious to one of skill in the art to perform a known function such as inserting, retrieving, or deleting, following the step of calling EXPIRY_PROCESSING to remove an element from the list.</p>
<p>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.</p>	<p>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.</p>	<p>Varghese and Lauck discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. <i>See, e.g.</i>, Varghese and Lauck at p. 28.</p> <p>“The simulation proceeds by processing the earliest event, which in turn may schedule further events. The simulation continues until the event list is empty or some condition (e.g. clock > MAX-SIMULATION-TIME) holds.” <i>See</i>, Varghese and Lauck at p. 28.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to</p>

EXHIBIT C-17

Asserted Claims From U.S. Pat. No. 5,893,120	George Varghese and Tony Lauck, <i>Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility</i>, ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 21, Issue 5, p. 25-38 (November 1987) (hereinafter “Varghese and Lauck”)
	<p>modify the system disclosed in Varghese and Lauck 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 Varghese and Lauck 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 Varghese and Lauck 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 Varghese and Lauck 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.</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, <i>Kruse</i> discloses an information storage and retrieval system.</p> <p>For example, <i>Kruse</i> discloses “[w]hen writing a program, we have had to decide on the maximum amount of memory that would be needed for our arrays and set this aside in the declarations.” <i>Kruse</i> at 105.</p> <p><i>Kruse</i> also discloses “First, an array must be declared that will hold the hash table. ... To insert a record into the hash table, the hash function for the key is first calculated. ... To retrieve the record with a given key is entirely similar.” <i>Kruse</i> at 200.</p>
<p>[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,</p>	<p>[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,</p>	<p><i>Kruse</i> 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. <i>Kruse</i> 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.</p> <p>For example, <i>Kruse</i> discloses that “[t]he idea we use is that of a pointer. A <i>pointer</i>, also called a <i>link</i> or a <i>reference</i>, is defined to be a variable that gives the location of some other variable, typically of a record containing data that we wish to use. If we use pointers to locate all the records in which we are interested, then we need not be concerned about where the records themselves are actually stored, since by using a pointer, we can let the computer system itself locate the record when required.” <i>Kruse</i> at 105. <i>Kruse</i> also discloses that the “idea of a linked list is, for every record in the list, to put a pointer into the record giving the location of the next record in the list.” <i>Kruse</i> at 106.</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>Additionally, <i>Kruse</i> states that “when an item is no longer needed, its space can be returned to the system, which can then assign it to another user.” <i>Kruse</i> at 107. “If our program is one that continually sets up new nodes and disposes of others, then we shall often find it necessary to set up our own procedures to keep track of nodes that are no longer needed, and to reuse the space when new nodes are later required.” <i>Kruse</i> at 117.</p> <p>Moreover, <i>Kruse</i> discloses “[w]e have already decided to represent our sparse array of cells as a hash table, but we have not yet decided between open addressing and chaining. ... Do we need to make deletions, and, if so, when? We could keep track of all cells until the memory is full, and then delete those that are not needed. But this would require rehashing the full array, which would be slow and painful. With chaining we can easily dispose of cells as soon as they are not needed, and thereby reduce the number of cells in the hash table as much as possible.” <i>Kruse</i> at 216. <i>Kruse</i> also discloses, “[i]f we use chaining, then we can add a cell to a list either by inserting the cell itself or a pointer to it, rather than by inserting its coordinates as before. In this way we can locate the cell directly with no need for any search.” ... “For reasons both of flexibility and time saving, therefore, let us decide to use dynamic memory allocation, a chained hash table, and linked lists.” <i>Kruse</i> at 217.</p> <p>Finally, <i>Kruse</i> discloses that “[i]n using a hash table, let the nature of the data and the required operations help you decide between chaining and open addressing. Chaining is generally preferable if deletions are required, if the records are relatively large, or if overflow might be a problem.” <i>Kruse</i> at 223.</p>
<p>[1b] a record search means utilizing a search key to</p>	<p>[5b] a record search means utilizing a search key to</p>	<p><i>Kruse</i> discloses a record search means utilizing a search key to access the linked list. <i>Kruse</i> also discloses a record search means utilizing a search key to</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
access the linked list,	access a linked list of records having the same hash address,	<p>access a linked list of records having the same hash address.</p> <p>For example, <i>Kruse</i> discloses “[t]he idea of a hash table (such as the one shown in Figure 6.10) is to allow many of the different possible keys that might occur to be mapped to the same location in an array under the action of the index function.” <i>Kruse</i> at 199.</p> <p><i>Kruse</i> also discloses “[t]he task of the procedure is first to look in the hash table for the cell with the given coordinates. If the search is successful, then the procedure returns a pointer to the cell; otherwise, it must create a new cell, assign it the given coordinates, initialize its other fields to the default values, and put it in the hash table as well as return a pointer to it.” <i>Kruse</i> at 220-21.</p>
[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	<p><i>Kruse</i> 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. <i>Kruse</i> 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.</p> <p>For example, <i>Kruse</i> discloses “[t]he task of the procedure <code>vivify</code> is to traverse the list <code>live</code>, 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 <code>AddNeighbors</code>.” <i>Kruse</i> at 219.</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p><i>Kruse</i> also discloses an exercise where the reader “rewrite(s) the procedure <code>Vivify</code> to use two pointers in traversing the list <code>live</code>, and dispose of redundant nodes when they are encountered. Also make the accompanying simplifications in the procedures <code>AddNeighbors</code> and <code>SubtractNeighbors</code>.” <i>Kruse</i> at 222.</p> <p>By way of further example, one of ordinary skill in the art would have combined linked lists as taught by this reference 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.</p>
<p>[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.</p>	<p>[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.</p>	<p><i>Kruse</i> 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. <i>Kruse</i> 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.</p> <p>For example, <i>Kruse</i> discloses “[t]he task of the procedure <code>vivify</code> is to traverse the list <code>live</code>, 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>AddNeighbors.” <i>Kruse</i> at 219.</p> <p><i>Kruse</i> also discloses an exercise where the reader “rewrite(s) the procedure Vivify to use two pointers in traversing the list live, and dispose of redundant nodes when they are encountered. Also make the accompanying simplifications in the procedures AddNeighbors and SubtractNeighbors.” <i>Kruse</i> at 222.</p> <p>Finally, <i>Kruse</i> discloses inserting and retrieving records from the system:</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>- A chained hash table in Pascal takes declarations like</p> <pre>declarations type pointer = 1 node; list = record head: pointer end; hashtable = array [0 .. hashmax] of list;</pre> <p>The record type called node consists of an item, called info, and an additional field, called next, that points to the next node on a linked list. The code needed to initialize the hash table is</p> <pre>initialization for i := 0 to hashmax do H[i].head := nil;</pre> <p>We can even use previously written procedures to access the hash table. The hash function itself is no different from that used with open addressing; for data retrieval we can simply use the procedure SequentialSearch (linked version) from Section 5.2, as follows:</p> <pre>retrieval procedure Retrieve(var H: hashtable; target: keytype; var found: Boolean; var location: pointer); //finds the node with key target in the hash table H, and returns with location //pointing to that node, provided that found becomes true begin SequentialSearch(H[Hash(target)], target, found, location) end;</pre> <p>Our procedure for inserting a new entry will assume that the key does not appear already; otherwise, only the most recent insertion with a given key will be retrieved:</p> <pre>insertion procedure Insert(var H: hashtable; p: pointer); //inserts node p into the chained hash table H, assuming no other node with //key p.info.key is in the table var i: integer; //used for index in hash table begin i := Hash(p.info.key); //Find the index of the linked list for p p.next := H[i].head; //insert p at the head of the list H[i].head := p; //Set the head of the list to the new item end;</pre> <p><i>Kruse at 208.</i></p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>By way of further example, one of ordinary skill in the art would have combined linked lists as taught by this reference 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.</p>
<p>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.</p>	<p>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.</p>	<p><i>Kruse</i> 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <i>Kruse</i> 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 <i>Kruse</i> 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 <i>Kruse</i> 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 <i>Kruse</i> 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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.</p> <p>Kruse 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both <i>Kruse</i> 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 <i>Kruse</i>. 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 <i>Kruse</i> nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120	Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
	<p>combined Dirks’ dynamic determination of the suitable number of entries to examine during each step of the sweeping process with <i>Kruse</i> and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <i>Kruse</i> with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by <i>Thatte</i>. <i>Thatte</i>, 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 <i>Thatte</i> is clearly shown in the chart of <i>Thatte</i>, which is hereby incorporated by reference in its entirety.</p> <p>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 <i>Kruse</i> with <i>Thatte</i> 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine <i>Kruse</i> with <i>Thatte</i> and recognize the benefits of doing so. For example, the removal of expired records described in <i>Kruse</i> 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 <i>Kruse</i> with the</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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 Kruse with Thatte.</p> <p>Alternatively, it would also be obvious to combine Kruse 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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</p>

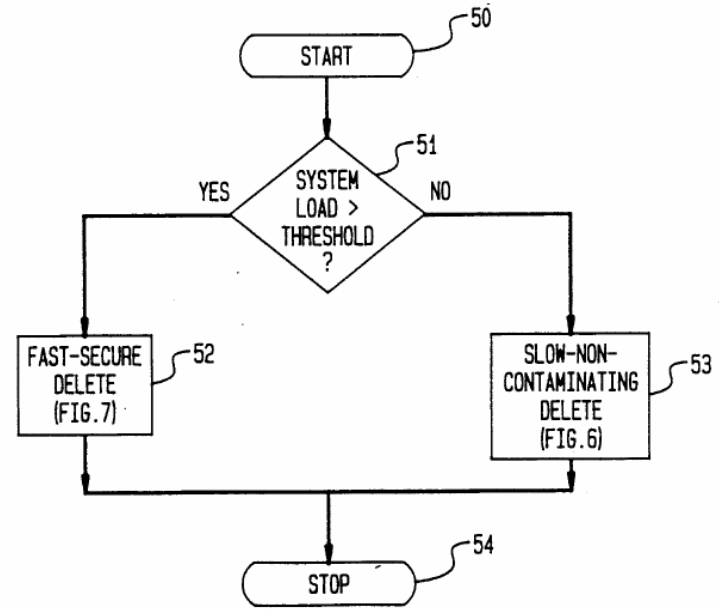
EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>moving records in the chain as described above. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-18

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination</p>
--	---

FIG. 5
HYBRID
DELETION



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 C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Kruse and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Kruse. 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 Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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 Kruse 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.</p> <p>Alternatively, it would also be obvious to combine Kruse with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination</p>
		<p>This decision routine should take several things into account: 1) the volume of data allocated since the last scavenger, 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 scavenger. If the product of the allocation and the compute time is high, and if the stack is low, the scavenger favorability measure is high. If it is especially high, a multi-generation scavenger is in order. <i>Id.</i></p> <p>If these heuristics fail and a scavenger 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 scavenger pause within a larger pause. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Kruse 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 Kruse. 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Kruse and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Kruse 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 Kruse 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 Kruse 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.</p> <p>One of ordinary skill in the art would have known that dynamically determining the maximum number to remove would limit the burden on the</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>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.</p> <p>Thus, the ’120 patent provides motivations to combine <i>Kruse</i> with Thatte, Dirks, the ‘663 patent, and/or the Opportunistic Garbage Collection Articles in addition to motivations within the text of <i>Kruse</i>, such as “[s]imilarly, when an item is no longer needed, its space can be returned to the system, which can then assign it to another user. In this way a program can start small and grow only as necessary, so that when it is small, it can run more efficiently, and when necessary, it can grow to the limits of the computer system.” <i>Kruse</i> at 107. <i>Kruse</i> further provides motivations within the text by posing the question “[d]o we need to make deletions, and, if so, when? We could keep track of all cells until the memory is full, and then delete those that are not needed.” <i>Kruse</i> at 216.</p>
<p>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:</p>	<p>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</p>	<p>To the extent the preamble is a limitation, <i>Kruse</i> 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. <i>Kruse</i> 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.</p> <p>For example, <i>Kruse</i> discloses “[w]hen writing a program, we have had to</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
	<p>automatically expiring, the method comprising the steps of:</p>	<p>decide on the maximum amount of memory that would be needed for our arrays and set this aside in the declarations.” <i>Kruse</i> at 105.</p> <p><i>Kruse</i> also discloses “First, an array must be declared that will hold the hash table. ... To insert a record into the hash table, the hash function for the key is first calculated. ... To retrieve the record with a given key is entirely similar.” <i>Kruse</i> at 200.</p> <p>Furthermore, <i>Kruse</i> discloses that “[t]he idea we use is that of a pointer. A <i>pointer</i>, also called a <i>link</i> or a <i>reference</i>, is defined to be a variable that gives the location of some other variable, typically of a record containing data that we wish to use. If we use pointers to locate all the records in which we are interested, then we need not be concerned about where the records themselves are actually stored, since by using a pointer, we can let the computer system itself locate the record when required.” <i>Kruse</i> at 105. <i>Kruse</i> also discloses that the “idea of a linked list is, for every record in the list, to put a pointer into the record giving the location of the next record in the list.” <i>Kruse</i> at 106.</p> <p>Additionally, <i>Kruse</i> states that “when an item is no longer needed, its space can be returned to the system, which can then assign it to another user.” <i>Kruse</i> at 107. “If our program is one that continually sets up new nodes and disposes of others, then we shall often find it necessary to set up our own procedures to keep track of nodes that are no longer needed, and to reuse the space when new nodes are later required.” <i>Kruse</i> at 117.</p> <p>Moreover, <i>Kruse</i> discloses “[w]e have already decided to represent our sparse array of cells as a hash table, but we have not yet decided between open addressing and chaining. ... Do we need to make deletions, and, if so, when?”</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>We could keep track of all cells until the memory is full, and then delete those that are not needed. But this would require rehashing the full array, which would be slow and painful. With chaining we can easily dispose of cells as soon as they are not needed, and thereby reduce the number of cells in the hash table as much as possible.” <i>Kruse</i> at 216. <i>Kruse</i> also discloses, “[i]f we use chaining, then we can add a cell to a list either by inserting the cell itself or a pointer to it, rather than by inserting its coordinates as before. In this way we can locate the cell directly with no need for any search.” ... “For reasons both of flexibility and time saving, therefore, let us decide to use dynamic memory allocation, a chained hash table, and linked lists.” <i>Kruse</i> at 217.</p> <p>Finally, <i>Kruse</i> discloses that “[i]n using a hash table, let the nature of the data and the required operations help you decide between chaining and open addressing. Chaining is generally preferable if deletions are required, if the records are relatively large, or if overflow might be a problem.” <i>Kruse</i> at 223.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p><i>Kruse</i> discloses accessing a linked list of records. <i>Kruse</i> also discloses accessing a linked list of records having same hash address.</p> <p>For example, <i>Kruse</i> discloses “[t]he idea of a hash table (such as the one shown in Figure 6.10) is to allow many of the different possible keys that might occur to be mapped to the same location in an array under the action of the index function.” <i>Kruse</i> at 199.</p> <p><i>Kruse</i> also discloses “[t]he task of the procedure is first to look in the hash table for the cell with the given coordinates. If the search is successful, then the procedure returns a pointer to the cell; otherwise, it must create a new cell, assign it the given coordinates, initialize its other fields to the default values, and put it in the hash table as well as return a pointer to it.” <i>Kruse</i> at 220-21.</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) 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,	<p><i>Kruse</i> discloses identifying at least some of the automatically expired ones of the records. <i>Kruse</i> also discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, <i>Kruse</i> discloses “[t]he task of the procedure <code>vivify</code> is to traverse the list <code>live</code>, 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 <code>AddNeighbors</code>.” <i>Kruse</i> at 219.</p> <p><i>Kruse</i> also discloses an exercise where the reader “rewrite(s) the procedure <code>Vivify</code> to use two pointers in traversing the list <code>live</code>, and dispose of redundant nodes when they are encountered. Also make the accompanying simplifications in the procedures <code>AddNeighbors</code> and <code>SubtractNeighbors</code>.” <i>Kruse</i> at 222.</p>
[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	<p><i>Kruse</i> discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. <i>Kruse</i> also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, <i>Kruse</i> discloses “[t]he task of the procedure <code>vivify</code> is to traverse the list <code>live</code>, 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</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>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 <code>AddNeighbors</code>.” <i>Kruse</i> at 219.</p> <p><i>Kruse</i> also discloses an exercise where the reader “rewrite(s) the procedure <code>Vivify</code> to use two pointers in traversing the list <code>live</code>, and dispose of redundant nodes when they are encountered. Also make the accompanying simplifications in the procedures <code>AddNeighbors</code> and <code>SubtractNeighbors</code>.” <i>Kruse</i> at 222.</p> <p>By way of further example, one of ordinary skill in the art would have combined linked lists as taught by this reference 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.</p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p><i>Kruse</i> discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, <i>Kruse</i> discloses “[t]he task of the procedure <code>vivify</code> is to traverse the list <code>live</code>, 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>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 from 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 <code>AddNeighbors</code>.” <i>Kruse</i> at 219.</p> <p><i>Kruse</i> also discloses an exercise where the reader “rewrite(s) the procedure <code>Vivify</code> to use two pointers in traversing the list <code>live</code>, and dispose of redundant nodes when they are encountered. Also make the accompanying simplifications in the procedures <code>AddNeighbors</code> and <code>SubtractNeighbors</code>.” <i>Kruse</i> at 222.</p> <p>Finally, <i>Kruse</i> discloses inserting and retrieving records from the system:</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>By way of further example, one of ordinary skill in the art would have combined linked lists as taught by this reference 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.</p>
<p>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.</p>	<p>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.</p>	<p><i>Kruse</i> discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <i>Kruse</i> 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 <i>Kruse</i> 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 <i>Kruse</i> 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 <i>Kruse</i> 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>‘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.</p> <p>Kruse 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination</p>
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Kruse 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 Kruse. 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 Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Kruse and would have</p>

EXHIBIT C-18

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination</p>
		<p>seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <i>Kruse</i> with the means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records disclosed by <i>Thatte</i>. <i>Thatte</i>, 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 <i>Thatte</i> is clearly shown in the chart of <i>Thatte</i>, which is hereby incorporated by reference in its entirety.</p> <p>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 <i>Kruse</i> with <i>Thatte</i> 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine <i>Kruse</i> with <i>Thatte</i> and recognized the benefits of doing so. For example, the removal of expired records described in <i>Kruse</i> 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 <i>Kruse</i> with the teachings of <i>Thatte</i> would solve this problem by dynamically determining how many records to delete based on, among other things, the system load.</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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 Kruse with Thatte.</p> <p>Alternatively, it would also be obvious to combine Kruse 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-18

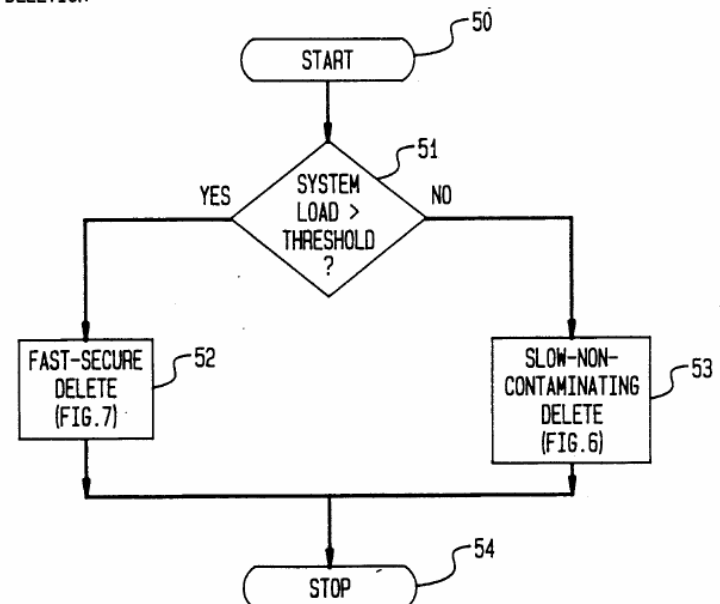
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Kruse and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Kruse. 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 Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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 Kruse 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.</p> <p>Alternatively, it would also be obvious to combine Kruse with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. Kruse, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“Kruse”) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Kruse 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 Kruse. 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</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with <i>Kruse</i> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <i>Kruse</i> 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <i>Kruse</i> 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 <i>Kruse</i> 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 <i>Kruse</i> 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.</p> <p>One of ordinary skill in the art would have known that dynamically</p>

EXHIBIT C-18

Asserted Claims From U.S. Pat. No. 5,893,120		Robert L. <i>Kruse</i>, Data Structures and Program Design, Prentice-Hall, Inc. 1984 and 1987 (“<i>Kruse</i>”) alone and in combination
		<p>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.</p> <p>Thus, the ‘120 patent provides motivations to combine <i>Kruse</i> with Thatte, Dirks, the ‘663 patent, and/or the Opportunistic Garbage Collection Articles in addition to motivations within the text of <i>Kruse</i>, such as “[s]imilarly, when an item is no longer needed, its space can be returned to the system, which can then assign it to another user. In this way a program can start small and grow only as necessary, so that when it is small, it can run more efficiently, and when necessary, it can grow to the limits of the computer system.” <i>Kruse</i> at 107. <i>Kruse</i> further provides motivations within the text by posing the question “[d]o we need to make deletions, and, if so, when? We could keep track of all cells until the memory is full, and then delete those that are not needed.” <i>Kruse</i> at 216.</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, Dixon and Calvert disclose an information storage and retrieval system.</p> <p>For example, Dixon and Calvert disclose a system for demultiplexing using a hash table of linked lists:</p> <p>“McKenney and Dove first introduced a demultiplexing algorithm that combines software caching and multiple hash chains. The algorithm maintains a linear list of PCBs for each of several hash chains. Each hash chain has an associated cache that points to the last PCB found on that chain.” <i>See</i> Calvert and Dixon at 6.</p>
<p>[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,</p>	<p>[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,</p>	<p>Dixon and Calvert disclose 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.</p> <p>Dixon and Calvert also disclose 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.</p> <p>For example, Dixon and Calvert disclose a system for demultiplexing using a hash table of linked lists:</p> <p>“McKenney and Dove first introduced a demultiplexing algorithm that combines software caching and multiple hash chains. The algorithm maintains a linear list of PCBs for each of several hash chains. Each hash chain has an</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>associated cache that points to the last PCB found on that chain.” See Calvert and Dixon at 6.</p> <p>“The next logical step in our investigation was to characterize the performance gains obtained by dividing the conventional single TCP PCB list into multiple shorter lists (<i>hash chains</i>) and use a single cache per hash chain to avoid lookups.” See Calvert and Dixon at 13.</p> <div data-bbox="984 639 1877 1157" data-label="Diagram"> </div> <p>Figure 4.3 Diagram of an <i>N</i> hash chain algorithm with one cache per chain.</p> <p>In addition, Dixon and Calvert disclose some of the records automatically</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>expiring:</p> <p><i>See, e.g.,</i> Dixon and Calvert at 8: “In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.”</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Dixon and Calvert disclose a record search means utilizing a search key to access the linked list. Dixon and Calvert also disclose a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, Dixon and Calvert disclose using a hash key comprising connection information is used in conjunction with a hash function to access the appropriate hash chain:</p> <p>“When the connection is first created, a hash function uses some part of the connection’s information (e. g., IP address) to generate a hash value. The PCB is then added to the hash chain that corresponds to the generated hash value. Subsequently, the hash function will route any incoming packets destined for that PCB to the appropriate hash chain. Note that the same hash key (i. e., same connection information) must be present in the arriving packet in order to assure proper routing.” <i>See</i> Dixon and Calvert at 6.</p>
<p>[1c] the record search means including a means for identifying and</p>	<p>[5c] the record search means including means for identifying and removing</p>	<p>Dixon and Calvert directly or inherently disclose 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. Dixon</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
<p>removing at least some of the expired ones of the records from the linked list when the linked list is accessed, and</p>	<p>at least some expired ones of the records from the linked list of records when the linked list is accessed, and</p>	<p>and Calvert also directly or inherently disclose 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.</p> <p>For example, Dixon and Calvert disclose a time a limit for a record remaining in the list of PCBs:</p> <p>“In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.” <i>See</i> Dixon and Calvert at 7.</p> <p>The existence of a limit on the time a record remains in the list requires removal at some point. Removal inherently requires the step of identification. Furthermore, because removal of a record from a linked list requires updating the links of other entries in the list, it inherently includes accessing the linked list of records.</p>
<p>[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</p>	<p>[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</p>	<p>Dixon and Calvert directly or inherently disclose 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. Dixon and Calvert also directly or inherently disclose 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</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
<p>records in the linked list.</p>	<p>some expired ones of the records in the accessed linked list of records.</p>	<p>list of records.</p> <p>For example, Dixon and Calvert disclose a system for demultiplexing using a hash table of linked lists:</p> <p>“McKenney and Dove first introduced a demultiplexing algorithm that combines software caching and multiple hash chains. The algorithm maintains a linear list of PCBs for each of several hash chains. Each hash chain has an associated cache that points to the last PCB found on that chain. When the connection is first created, a hash function uses some part of the connection’s information (e. g., IP address) to generate a hash value. The PCB is then added to the hash chain that corresponds to the generated hash value. Subsequently, the hash function will route any incoming packets destined for that PCB to the appropriate hash chain. Note that the same hash key (i. e., same connection information) must be present in the arriving packet in order to assure proper routing. The packet is assigned to its PCB via a BSD 4.3-Reno type search of the list.” <i>See</i> Calvert and Dixon at 6.</p> <p>As described in the citation above, Calvert and Dixon disclose a search means—the combination of using a hash key and hash function to select a hash bucket and a “BSD 4.3-Reno type search” of the linked list chained to the hash bucket. As further disclosed in the citation above, insertion and retrieval when a new packet incoming packets arrive utilize the search means.</p> <p>In addition, Dixon and Calvert disclose some of the records automatically expiring:</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p><i>See, e.g.</i>, Dixon and Calvert at 8: “In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.”</p> <p>The existence of a limit on the time a record remains in the list requires removal at some point. Furthermore, because removal of a record from a linked list requires updating the links of other entries in the list, it inherently includes accessing the linked list of records. Where the linked list is chained to a hash table, accessing the item to remove inherently requires use of the search means discussed above. Consequently, where a system maintains a list of PCBs using a hash table with external chaining and where said system inserts, retrieves and deletes using a record search means, then such a system is a means for inserting, retrieving, and deleting records, that utilizes a record search means and at the same time removes an expired entry from the system.</p>
<p>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</p>	<p>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</p>	<p>Dixon and Calvert 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.</p> <p>Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system,</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120		Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
linked list of records.	linked list of records.	<p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Dixon and Calvert and Dirks relate to deletion of aged records, one of ordinary skill in the art would have understood how to use Dirks’ dynamic</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>decision making process of determining the maximum number of records to sweep/remove in other hash tables implementations such as Dixon and Calvert . 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 Dixon and Calvert nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Dixon and Calvert and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Dixon and Calvert 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</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>the chart of Thatte, which is hereby incorporated by reference in its entirety.</p> <p>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 Dixon and Calvert 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Dixon and Calvert with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Dixon and Calvert 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 Dixon and Calvert 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 Dixon and Calvert with Thatte.</p> <p>Alternatively, it would also be obvious to combine Dixon and Calvert with the '663 patent. Disclosure of these claim elements in the '663 patent is clearly</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>shown in the chart of the '663 patent, which is hereby incorporated by reference in its entirety. As summarized in the '663 patent:</p> <p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT C-19

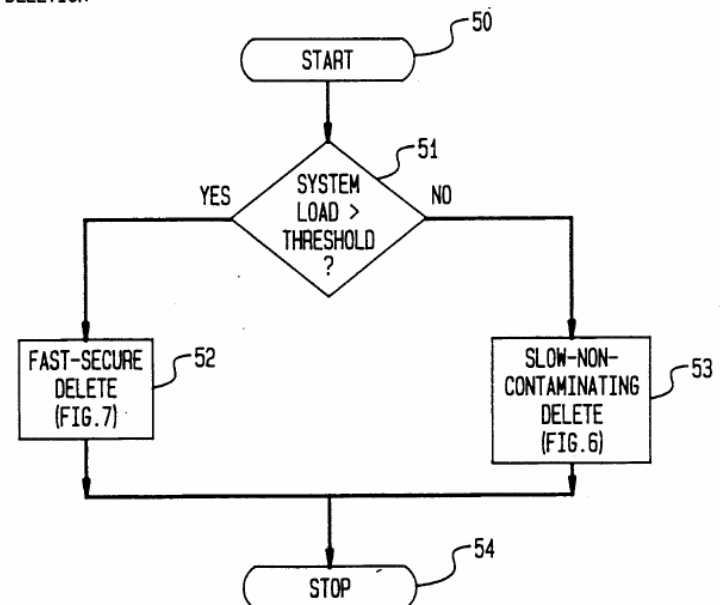
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers</i> (1996) (hereinafter “Dixon and Calvert”)</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Dixon and Calvert and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Dixon and Calvert. 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 Dixon and Calvert would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120		Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
		<p>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 Dixon and Calvert 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.</p> <p>Alternatively, it would also be obvious to combine Dixon and Calvert with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Dixon and Calvert 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 Dixon and Calvert. 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</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>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 Dixon and Calvert would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Dixon and Calvert and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dixon and Calvert 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 Dixon and Calvert 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 Dixon and Calvert can be burdensome on the system, adding to the system’s load and slowing down the</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>system’s processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Dixon and Calvert disclose 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. Dixon and Calvert also disclose 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.</p> <p>For example, Dixon and Calvert disclose a system for demultiplexing using a hash table of linked lists:</p> <p>“McKenney and Dove first introduced a demultiplexing algorithm that combines software caching and multiple hash chains. The algorithm maintains a linear list of PCBs for each of several hash chains. Each hash chain has an</p>

EXHIBIT C-19

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>associated cache that points to the last PCB found on that chain.” See Calvert and Dixon at 6.</p> <p>“The next logical step in our investigation was to characterize the performance gains obtained by dividing the conventional single TCP PCB list into multiple shorter lists (<i>hash chains</i>) and use a single cache per hash chain to avoid lookups.” See Calvert and Dixon at 13.</p> <div data-bbox="984 639 1877 1159" style="border: 1px solid black; padding: 10px;"> </div> <p>Figure 4.3 Diagram of an N hash chain algorithm with one cache per chain.</p> <p>In addition, Dixon and Calvert disclose some of the records automatically</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>expiring:</p> <p><i>See, e.g.</i>, Dixon and Calvert at 8: “In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.”</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>Dixon and Calvert disclose accessing a linked list of records. Dixon and Calvert also disclose accessing a linked list of records having same hash address.</p> <p>For example, Dixon and Calvert disclose using a hash key comprising connection information in conjunction with a hash function to access the appropriate hash chain:</p> <p>“When the connection is first created, a hash function uses some part of the connection’s information (e. g., IP address) to generate a hash value. The PCB is then added to the hash chain that corresponds to the generated hash value. Subsequently, the hash function will route any incoming packets destined for that PCB to the appropriate hash chain. Note that the same hash key (i. e., same connection information) must be present in the arriving packet in order to assure proper routing.” <i>See</i> Dixon and Calvert at 6.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>Dixon and Calvert directly or inherently disclose identifying at least some of the automatically expired ones of the records.</p> <p>For example, Dixon and Calvert disclose a time a limit for a record remaining</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>in the list of PCBs:</p> <p>“In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.” <i>See</i> Dixon and Calvert at 7.</p> <p>The existence of a limit on the time a record remains in the list requires removal at some point. Removal inherently requires the step of identifying records to be removed.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Dixon and Calvert directly or inherently disclose removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, Dixon and Calvert disclose a time a limit for a record remaining in the list of PCBs:</p> <p>“In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.” <i>See</i> Dixon and Calvert at 7.</p> <p>The existence of a limit on the time a record remains in the list requires removal at some point. Removal inherently requires the step of identification.</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120		Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
		Furthermore, because removal of a record from a linked list requires updating the links of other entries in the list, it inherently includes accessing the linked list of records.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>Dixon and Calvert directly or inherently disclose inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Dixon and Calvert disclose a system for demultiplexing using a hash table of linked lists:</p> <p>“McKenney and Dove first introduced a demultiplexing algorithm that combines software caching and multiple hash chains. The algorithm maintains a linear list of PCBs for each of several hash chains. Each hash chain has an associated cache that points to the last PCB found on that chain. When the connection is first created, a hash function uses some part of the connection’s information (e. g., IP address) to generate a hash value. The PCB is then added to the hash chain that corresponds to the generated hash value. Subsequently, the hash function will route any incoming packets destined for that PCB to the appropriate hash chain. Note that the same hash key (i. e., same connection information) must be present in the arriving packet in order to assure proper routing. The packet is assigned to its PCB via a BSD 4.3-Reno type search of the list.” <i>See</i> Calvert and Dixon at 6.</p> <p>As disclosed in the citation above, insertion and retrieval when a new packet incoming packets arrive utilize the search means.</p> <p>In addition, Dixon and Calvert disclose some of the records automatically</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>expiring:</p> <p><i>See, e.g.</i>, Dixon and Calvert at 8: “In addition, the TCP implementation of all four servers had a <i>maximum segment lifetime</i> value of 60 seconds. We used this same value in our simulations. This value is important because a TCP connection remains in the PCB list for twice this length of time after it is closed.”</p> <p>The existence of a limit on the time a record remains in the list requires removal. Furthermore, as mentioned above, Calvert and Dixon disclose using a chained hash table in the context of demultiplexing. Demultiplexing is “the process of decomposing” a packet stream, such as a TCP/IP packet stream, to provide delivery to destination processes. <i>See</i> Calvert and Dixon at 2. A server employing a system for demultiplexing, such as the system disclosed by Calvert and Dixon, typically would receive a significant number of packets. For example, in an experiment Calvert and Dixon observed millions of incoming packets on four servers in under two hours. <i>See</i> Calvert and Dixon at 3-4, Table 3.1. As such, even with the use of a caching mechanism to avoid having to perform a lookup into a PCB list for each incoming packet, it is inherent that the disclosed system, after removing an expired entry from the PCB list, will insert a new entry or retrieve an entry in response to subsequent incoming packets.</p>
<p>4. The method according to claim 3 further including</p>	<p>8. The method according to claim 7 further including</p>	<p>Dixon and Calvert combined with Dirks, Thatte, the '663 patent, and/or the Opportunistic Garbage Collection Articles discloses dynamically determining</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
<p>the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p>	<p>the step of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p>	<p>maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Dixon and Calvert and Dirks relate to deletion of aged records, 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 Dixon and Calvert. 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 Dixon and Calvert would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Dixon and Calvert and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Dixon and Calvert with</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>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.</p> <p>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 Dixon and Calvert 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Dixon and Calvert with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Dixon and Calvert 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 Dixon and Calvert 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</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120		Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
		<p>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 Dixon and Calvert with Thatte.</p> <p>Alternatively, it would also be obvious to combine Dixon and Calvert 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-</p>

EXHIBIT C-19

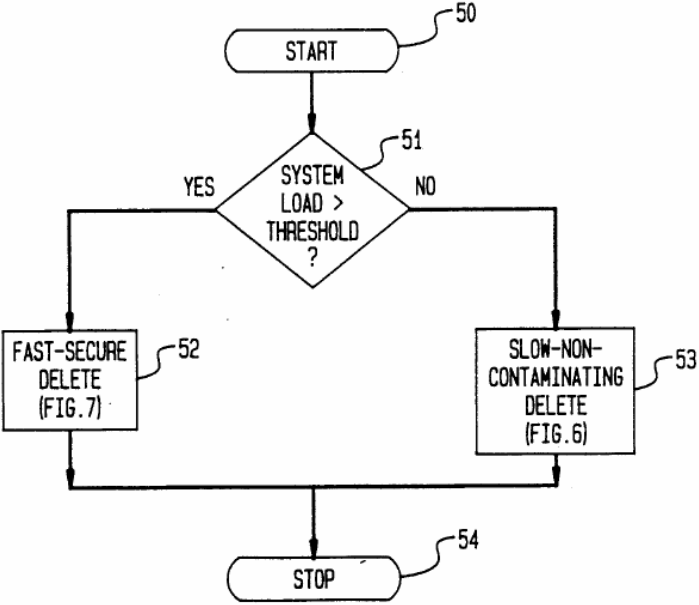
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p>The flowchart, labeled FIG. 5 HYBRID DELETION, illustrates a decision-based process. It begins with a 'START' terminal (50) leading to a decision diamond (51) asking 'SYSTEM LOAD > THRESHOLD?'. If the answer is 'YES', the process flows to a rectangular box (52) labeled 'FAST-SECURE DELETE (FIG. 7)'. If the answer is 'NO', it flows to another rectangular box (53) labeled 'SLOW-NON-CONTAMINATING DELETE (FIG. 6)'. Both paths converge and lead to a final 'STOP' terminal (54).</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120		Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
		<p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Dixon and Calvert and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Dixon and Calvert. 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</p>

EXHIBIT C-19

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>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 Dixon and Calvert would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Dixon and Calvert 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.</p> <p>Alternatively, it would also be obvious to combine Dixon and Calvert with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>When a significant pause has been detected, a decision procedure is invoked to decide whether to garbage collect, and how many generations to</p>

EXHIBIT C-19

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)</p>
	<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Dixon and Calvert and the Opportunistic Garbage Collection Articles</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>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 Dixon and Calvert. 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 Dixon and Calvert would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Dixon and Calvert 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Dixon and Calvert 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</p>

EXHIBIT C-19

Asserted Claims From U.S. Pat. No. 5,893,120	Joseph T. Dixon and Kenneth Calvert, <i>Increasing Demultiplexing Efficiency in TCP/IP Network Servers (1996)</i> (hereinafter “Dixon and Calvert”)
	<p>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 Dixon and Calvert 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 Dixon and Calvert 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.</p> <p>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.</p>

EXHIBIT D-1

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, LINUX 1.3.52 discloses an information storage and retrieval system.</p> <p>For example, LINUX 1.3.52 includes the <code>ip_rt_hash_table</code> global variable, which is an information storage and retrieval system.</p>
<p>[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,</p>	<p>[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,</p>	<p>LINUX 1.3.52 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. LINUX 1.3.52 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.</p> <p>For example, LINUX 1.3.52 includes the <code>ip_rt_hash_table</code> global variable, which is composed of an array of pointers to <code>struct rtable</code>s. See line 144. Each <code>struct rtable</code> contains an <code>rt_next</code> field, which is a pointer to another <code>struct rtable</code>. See <code>/include/net/route.h</code>, line 124. Accordingly, <code>struct rtable</code> defines (among other things) a linked list. As suggested by its name, the <code>ip_rt_hash_table</code> global variable uses a hashing means to provide access to its stored linked lists. The access is described below; the hash address itself is computed at lines 1109 and 1467, which call the function <code>ip_rt_hash_code</code>.</p> <p>The records in the system LINUX 1.3.52 discloses includes records, at least some of which automatically expire.</p> <p><code>struct rtable</code> also includes the <code>rt_lastuse</code> field, which is used to determine whether the record has automatically expired. See</p>

EXHIBIT D-1

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination</p>
		<p>/include/net/route.h, line 131 and analysis below.</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>LINUX 1.3.52 discloses a record search means utilizing a search key to access the linked list. LINUX 1.3.52 also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, as detailed in part [1a/5a], the <code>ip_rt_hash_table</code> global variable contains an array of linked lists of type <code>struct rtable</code>.</p> <p>As suggested by its name, the <code>ip_rt_hash_table</code> global variable is accessed using a search key. Specifically, the function <code>rt_cache_add</code> uses the search key <code>hash</code> to access the linked list at the “hash” index of the <code>ip_rt_hash_table</code> array. See lines 1415 and 1426.</p>
<p>[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</p>	<p>[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</p>	<p>LINUX 1.3.52 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.</p> <p>For example, as detailed in step [1b/5b], the function <code>rt_cache_add</code> accesses a linked list within the <code>ip_rt_hash_table</code> global variable at line 1415, and appends a new record to the front of the linked list at line 1426. As detailed in the comment at line 1432, <code>rt_cache_add</code> then iterates through the same linked list to remove aged off or “automatically expired” entries. Specifically, line 1439 determines whether the record has expired, line 1442 removes the expired record from the linked list, and line 1448 deletes the expired record</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>from memory.</p> <p>Thus, the linked list at <code>ip_rt_hash_table[hash]</code> is accessed at lines 1415 through 1427, when the <code>rt_cache_add</code> method adds the new record to the front of the linked list, and from lines 1435 through 1453, when the <code>rt_cache_add</code> method iterates through the linked list looking for duplicate and expired entries.</p>
<p>[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.</p>	<p>[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.</p>	<p>LINUX 1.3.52 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. LINUX 1.3.52 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.</p> <p>For example, <code>rt_cache_add</code> discloses a means for inserting a record into the linked list stored at <code>ip_rt_hash_table[hash]</code> and, at the same time, removing at least some of the records in that accessed linked list.</p> <p><code>rt_cache_add</code> also discloses a means for retrieving records from the linked list. See, e.g., lines 1415, 1435.</p> <p><code>rt_cache_add</code> also discloses a means for deleting records from the linked list. See, e.g., lines 1442, 1448. Further, the <code>while</code> loop of lines 1435 to 1453 is checking for duplicate entries (deleting entries) at the same time that it is checking for automatically expired entries. See lines 1439 and 1440.</p>

EXHIBIT D-1

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination</p>
<p>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.</p>	<p>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.</p>	<p>LINUX 1.3.52 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p style="padding-left: 40px;">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.</p> <p>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).</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both LINUX 1.3.52 and Dirks relate to deletion of aged records, 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 LINUX 1.3.52. 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 LINUX 1.3.52 nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 LINUX 1.3.52 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in LINUX 1.3.52 with the means for dynamically determining maximum number for the record search</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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.</p> <p>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 LINUX 1.3.52 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine LINUX 1.3.52 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in LINUX 1.3.52 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 LINUX 1.3.52 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 LINUX 1.3.52</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>with Thatte.</p> <p>Alternatively, it would also be obvious to combine LINUX 1.3.52 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:</p> <p style="padding-left: 40px;">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”).</p> <p style="padding-left: 40px;">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. <i>Id.</i> at 2:35-41.</p> <p style="padding-left: 40px;">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.</p>

EXHIBIT D-1

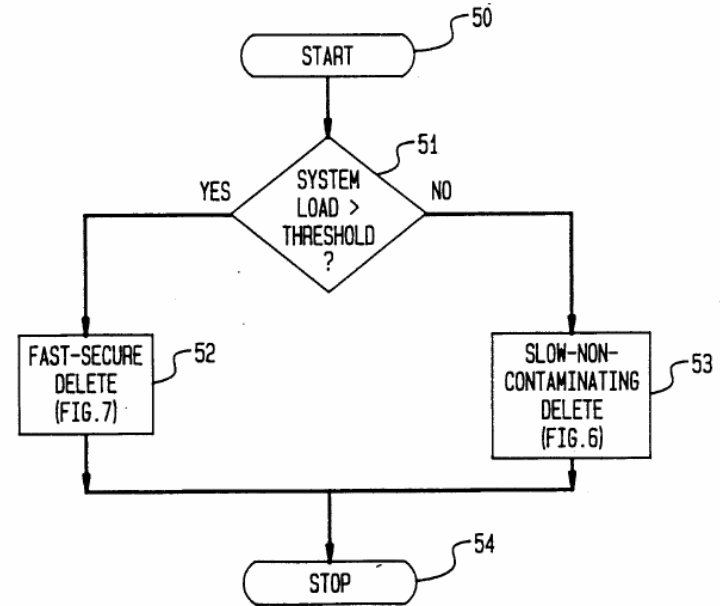
Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p><i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>During the hybrid deletion procedure decision block 51 checks the system load</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both LINUX 1.3.52 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 LINUX 1.3.52. 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</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120	LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
	<p>combining the '663 patent's deletion decision procedure with LINUX 1.3.52 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 LINUX 1.3.52 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.</p> <p>Alternatively, it would also be obvious to combine LINUX 1.3.52 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both LINUX 1.3.52 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 LINUX 1.3.52. 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</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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 LINUX 1.3.52 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 LINUX 1.3.52 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in LINUX 1.3.52 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 LINUX 1.3.52 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 LINUX 1.3.52 can be burdensome on the system, adding to the system’s load and slowing down the system’s</p>

EXHIBIT D-1

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination</p>
		<p>processing. Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, LINUX 1.3.52 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. LINUX 1.3.52 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.</p> <p>For example, LINUX 1.3.52 includes the <code>ip_rt_hash_table</code> global variable, which is composed of an array of pointers to <code>struct rtables</code>. See line 144. Each <code>struct rtable</code> contains an <code>rt_next</code> field, which is a pointer to another <code>struct rtable</code>. See <code>/include/net/route.h</code>, line 124. Accordingly, <code>struct rtable</code> defines (among other things) a linked list.</p>

EXHIBIT D-1

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination</p>
		<p>struct rtable also includes the rt_lastuse field, which is used to determine whether the record has automatically expired. See /include/net/route.h, line 131 and analysis below.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>LINUX 1.3.52 discloses accessing a linked list of records. Linux 1.3.52 also discloses accessing a linked list of records having same hash address.</p> <p>For example, the function rt_cache_add accesses the linked list at the “hash” index of the ip_rt_hash_table array. See lines 1415 and 1426. In addition, the linked list at ip_rt_hash_table[hash] is accessed from lines 1435 through 1453, when the rt_cache_add method iterates through the linked list.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>LINUX 1.3.52 discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, the function rt_cache_add accesses a linked list within the ip_rt_hash_table global variable at line 1415, and appends a new record to the front of the linked list at line 1426. As detailed in the comment at line 1432, rt_cache_add then iterates through the same linked list to remove aged off or “automatically expired” entries. Specifically, the loop beginning at line 1435 iterates through the records in the previously-accessed linked list, and line 1439 identifies whether a particular record has expired.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked</p>	<p>LINUX 1.3.52 discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, the loop beginning at line 1435 iterates through the records in the</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
list is accessed.	list is accessed, and	previously-accessed linked list, and line 1439 identifies whether a particular record has expired. Line 1442 removes the expired record from the linked list, and line 1448 deletes the expired record from memory.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	LINUX 1.3.52 discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, the loop beginning at line 1435 iterates through the records in the previously-accessed linked list, and line 1439 identifies whether a particular record has expired. Line 1442 removes the expired record from the linked list, and line 1448 deletes the expired record from memory. Further, the while loop of lines 1435 to 1453 is checking for duplicate entries (deleting entries) at the same time that it is checking for automatically expired entries. See lines 1439 and 1440. A duplicate entry may be deleted following the removal of at least some of the automatically expired records from the linked list.
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.	LINUX 1.3.52 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. As summarized in Dirks,

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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.</p> <p>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:</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both LINUX 1.3.52 and Dirks relate to deletion of aged records, 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 LINUX 1.3.52. 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</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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 LINUX 1.3.52 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 LINUX 1.3.52 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in LINUX 1.3.52 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.</p> <p>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 LINUX 1.3.52 with Thatte would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>The resulting combination would include the capability to determine the maximum number for the record search means to remove.</p> <p>Further, one of ordinary skill in the art would be motivated to combine LINUX 1.3.52 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in LINUX 1.3.52 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 LINUX 1.3.52 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 LINUX 1.3.52 with Thatte.</p> <p>Alternatively, it would also be obvious to combine LINUX 1.3.52 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT D-1

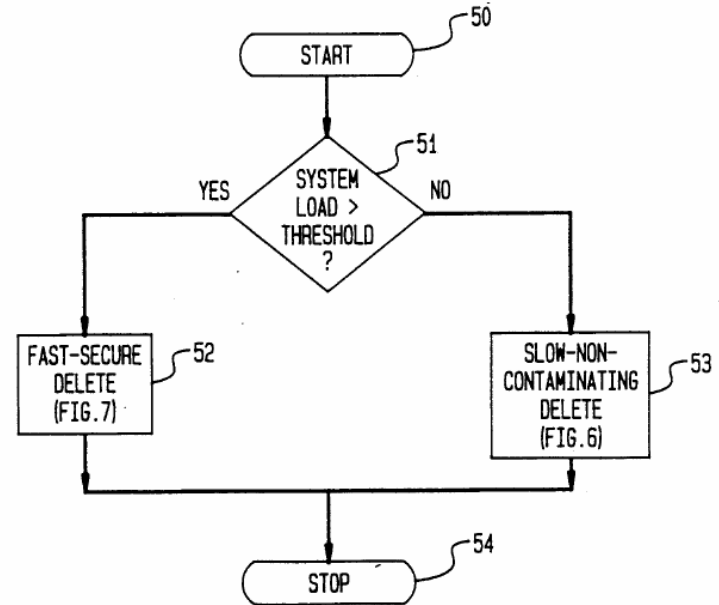
Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>to be deleted. This leaves no deleted record locations in the storage space to slow down future searches. U.S. Patent 4,996,663 to Nemes at 2:24-34 (“The ’663 patent”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-1

**Asserted Claims From
U.S. Pat. No. 5,893,120**

**LINUX 1.3.52 net\ipv4\route.c, available at
<http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz> (“LINUX
1.3.52”) alone and in combination**

FIG. 5
HYBRID
DELETION



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 D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both LINUX 1.3.52 and the ’663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 LINUX 1.3.52. 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 LINUX 1.3.52 would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>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 LINUX 1.3.52 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.</p> <p>Alternatively, it would also be obvious to combine LINUX 1.3.52 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>continual run-time overhead. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both LINUX 1.3.52 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 LINUX 1.3.52. 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.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) alone and in combination
		<p>Additionally, one of ordinary skill in the art would recognize that the result of combining the Opportunistic Garbage Collection Articles’ deletion decision procedure with LINUX 1.3.52 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 LINUX 1.3.52 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in LINUX 1.3.52 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 LINUX 1.3.52 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 LINUX 1.3.52 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.</p>

EXHIBIT D-1

Asserted Claims From U.S. Pat. No. 5,893,120		LINUX 1.3.52 net\ipv4\route.c, available at http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.52.tar.gz (“LINUX 1.3.52”) 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.

EXHIBIT D-2

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, if_ether.c discloses an information storage and retrieval system.</p> <p>For example, the implementation of the Address Resolution Protocol in if_ether.c in BSD 4.2 includes an information storage and retrieval system that stores and retrieves records used by the protocol.</p>
<p>[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,</p>	<p>[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,</p>	<p>if_ether.c discloses a hash table (arptab) which resolves collisions through arrays. See, e.g., lines 42-49. It would have been obvious to one of ordinary skill in the art that arptab could resolve collisions with linked lists rather than arrays, as both linked lists and arrays are fundamental data structures used to store multiple data items. See below.</p> <p>if_ether.c 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.</p> <p>For example, the structure if_ether.c describes the use of a hash table, arptab, with external chaining to resolve collisions. See, e.g., lines 42-49. Though the external chaining involves the use of an array rather than a linked list, it would have been obvious to a person skilled in the art that a linked list could be used instead of an array. The use of linked lists for external chaining in hash tables was well known in the art. Indeed, according to Knuth, “the most obvious way to solve this problem [of collisions] is to maintain <i>M</i> linked lists, one for each possible hash code.” See “The Art of Computer Programming”, Sorting and Searching, D.E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 513, 1973. See also Mark A. Weiss, Data Structures and Algorithm Analysis, p. 152-157, 1993 (using linked lists to resolve collisions in external chaining but noting that any</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code> , available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (" <code>if_ether.c</code> ") alone and in combination
		<p>scheme besides linked lists could be used). One of ordinary skill in the art would have been motivated to try using "the most obvious" solution to external chaining, linked lists, instead of the array taught in <code>if_ether.c</code>.</p> <p>The records in the system <code>if_ether.c</code> discloses includes records, at least some of which automatically expire.</p> <p>For example, the <code>arptab</code> table in <code>if_ether.c</code> includes within each entry an <code>at_timer</code> variable which keeps track of the minutes since the last reference. That <code>at_time</code> variable is used to expire the entry when the time since the last reference exceeds a given amount. See function <code>arptimer()</code>, lines 126-130.</p>
[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,	<p><code>if_ether.c</code> discloses a record search means utilizing a search key to access an array. Similarly, <code>if_ether.c</code> discloses a record search means utilizing a search key to access an array of records having the same hash address.</p> <p>For example, the <code>arptab</code> structure in <code>if_ether.c</code> can be accessed with a search key. That search key provides access to a series of entries within the <code>arptab</code> structure that have the same hash address. See, e.g., function <code>arptnew()</code>, lines 376-384. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could be used instead of an array to resolve the collisions in <code>arptab</code>, in which case the access in this element would occur on a linked list.</p>
[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	<p><code>if_ether.c</code> 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.</p>

EXHIBIT D-2

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination</p>
<p>the expired ones of the records from the linked list when the linked list is accessed, and</p>	<p>of the records from the linked list of records when the linked list is accessed, and</p>	<p>For example, the function <code>arptimer()</code> accesses the <code>arptab</code> structure and removes all entries that have expired when that access occurs. The function <code>arptnew()</code> also accesses the <code>arptab</code> structure in order to add an entry, and if the structure is full, <code>arptnew()</code> removes the oldest entry in the table and inserts the new entry in its place. Though this removal of expired records occurs in an array, as discussed above in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could be used to resolve collisions in the hash table, in which case the removal taught in this element would occur in the linked list.</p>
<p>[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.</p>	<p>[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.</p>	<p><code>if_ether.c</code> discloses means, utilizing the record search means, for accessing an array and, at the same time, removing at least some of the expired ones of the records in the array. <code>if_ether.c</code> 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 array of records. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could be used instead of an array.</p> <p>For example, <code>arptfree()</code> utilizes the record search means to insert a new entry in the <code>arptab</code> structure and, at the same time, remove one of the expired records from the structure when the structure is full. It would have been obvious to one skilled in the art that retrieval or deletion could have been done as well as the insert, since these are all basic functions that can be performed on a hash table. <i>See, e.g.</i>, “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.</p> <p>Though these actions occur in an array, as discussed above in [1a/5a], it would</p>

EXHIBIT D-2

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination</p>
		<p>have been obvious to one of ordinary skill in the art that a linked list could be used to resolve collisions in the hash table, in which case the actions taught in this element would occur in the linked list.</p>
<p>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.</p>	<p>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.</p>	<p><code>if_ether.c</code> discloses dynamically determining maximum number of expired ones of the records to remove when the array is accessed.</p> <p>For example, the function <code>arptfree()</code> only removes an expired element when the <code>arptab</code> structure is full. In this way, the function dynamically determines the maximum number of elements to remove by computing whether to remove some or none of the expired elements.</p> <p>Though these removals of expired records occur in an array, as discussed above in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could be used to resolve collisions in the hash table, in which case the removals taught in this claim would occur in the linked list.</p> <p>To the extent <code>if_ether.c</code> does not disclose this element, it would have been obvious to one of ordinary skill in the art.</p> <p><code>if_ether.c</code> 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.</p> <p>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.</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>As summarized in Dirks,</p> <p>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.</p> <p>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:</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both <code>if_ether.c</code> and Dirks relate to deletion of aged records, 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 <code>if_ether.c</code>. 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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
	<p>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 <code>if_ether.c</code> nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>if_ether.c</code> and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <code>if_ether.c</code> 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.</p> <p>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 <code>if_ether.c</code> 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination
		<p>if_ether.c with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in if_ether.c 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 if_ether.c 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 if_ether.c with Thatte.</p> <p>Alternatively, it would also be obvious to combine if_ether.c 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:</p> <p style="padding-left: 40px;">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”).</p> <p>In times of heavy use, when deletions must be done rapidly and no time is available for decontamination, the record is simply</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

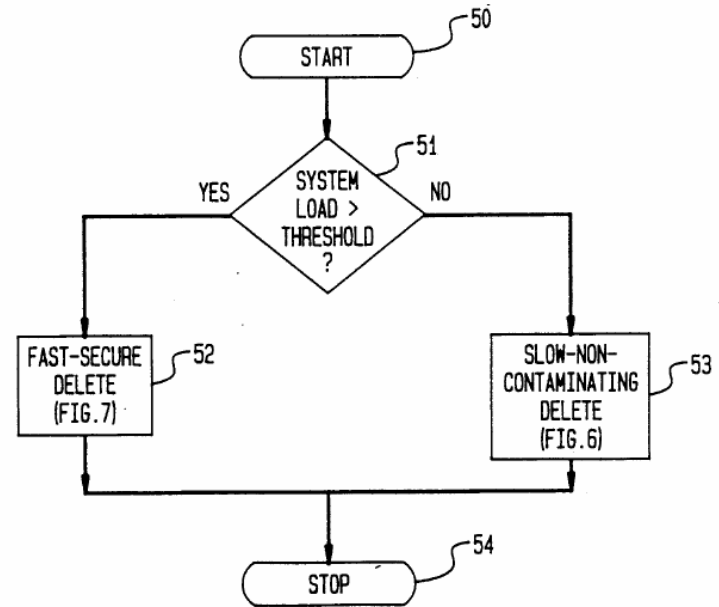
EXHIBIT D-2

Asserted Claims From
U.S. Pat. No. 5,893,120

BSD 4.2 netinet/if_ether.c, available at
<http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz> ("if_ether.c") alone and in combination

FIG. 5

HYBRID
DELETION



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.

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both <code>if_ether.c</code> and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 <code>if_ether.c</code>. 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 <code>if_ether.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>if_ether.c</code> 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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
	<p>exceeded a threshold.</p> <p>Alternatively, it would also be obvious to combine <code>if_ether.c</code> with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>scavenge favorability measure is high. If it is especially high, a multi-generation scavenge is in order. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both <code>if_ether.c</code> 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 <code>if_ether.c</code>. 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 <code>if_ether.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>if_ether.c</code> and</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination
		<p>would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <code>if_ether.c</code> 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 <code>if_ether.c</code> 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 <code>if_ether.c</code> 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.</p> <p>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.</p>
3. A method for storing	7. A method for storing	To the extent the preamble is a limitation, <code>if_ether.c</code> discloses a method

EXHIBIT D-2

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination</p>
<p>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:</p>	<p>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:</p>	<p>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. if_ether.c 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.</p> <p>For example, the arptab structure defined in if_ether.c is a hash table that uses external chaining with arrays to resolve collisions between entries with the same hash address. As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could have been utilized instead of an array to resolve collisions.</p> <p>The records in the system if_ether.c discloses includes records, at least some of which automatically expire.</p> <p>For example, the arptab table in if_ether.c includes within each entry an at_timer variable which keeps track of the minutes since the last reference. That at_time variable is used to expire the entry when the time since the last reference exceeds a given amount. See function arptimer(), lines 126-130.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>if_ether.c discloses accessing an array of records. Similarly, if_ether.c discloses accessing an array of records having the same hash address. As discussed in [1b/5b], it would have been obvious to one or ordinary skill in the art that the access could occur in a linked list rather than an array.</p> <p>For example, both the arptimer() and arptnew() functions in</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code> , available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (" <code>if_ether.c</code> ") alone and in combination
		<code>if_ether.c</code> access the array that stores records having the same hash address. See lines 123-32, 376-84.
[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,	<code>if_ether.c</code> discloses identifying at least some of the automatically expired ones of the records. For example, <code>arptimer()</code> identifies whether any of the entries in <code>arptab</code> have expired on lines 126-29. The function <code>arptnew()</code> identifies automatically expired records on lines 380-83.
[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	<code>if_ether.c</code> discloses removing at least some of the automatically expired records from the array when the array is accessed. As discussed in [1c/5c], it would have been obvious to one of ordinary skill in the art that the same step could be performed on a linked list where the records were stored in a linked list. For example, the function <code>arptimer()</code> accesses the <code>arptab</code> structure and removes all entries that have expired when that access occurs. The function <code>arptnew()</code> also accesses the <code>arptab</code> structure in order to add an entry, and if the structure is full, <code>arptnew()</code> identifies an expired entry in the table and removes the entry by calling the function <code>arptfree()</code> . Lines 385-86.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<code>if_ether.c</code> discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, function <code>arptnew()</code> inserts a new entry into the <code>arptab</code> structure after the expired element is removed. Lines 388-89.
4. The method according to claim 3 further including the step of dynamically	8. The method according to claim 7 further including the step of dynamically	<code>if_ether.c</code> discloses dynamically determining maximum number of expired ones of the records to remove when the array is accessed.

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
determining maximum number of expired ones of the records to remove when the linked list is accessed.	determining maximum number of expired ones of the records to remove when the linked list is accessed.	<p>For example, the function <code>arptfree()</code> only removes an expired element when the <code>arptab</code> structure is full. In this way, the function dynamically determines the maximum number of elements to remove by computing whether to remove some or none of the expired elements. Though these removals of expired records occur in an array, as discussed above in [1a/5a], it would have been obvious to one of ordinary skill in the art that a linked list could be used to resolve collisions in the hash table, in which case the removals taught in this claim would occur in the linked list.</p> <p>To the extent <code>if_ether.c</code> does not disclose this element, it would have been obvious to one of ordinary skill in the art.</p> <p><code>if_ether.c</code> 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both <code>if_ether.c</code> and Dirks relate to deletion of aged records, 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 <code>if_ether.c</code>. 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 <code>if_ether.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>if_ether.c</code> and would have seen the benefits of doing so. One possible benefit, for example, is</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
	<p>saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <code>if_ether.c</code> 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.</p> <p>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 <code>if_ether.c</code> 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine <code>if_ether.c</code> with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in <code>if_ether.c</code> 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 <code>if_ether.c</code> 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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
	<p>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 <code>if_ether.c</code> with Thatte.</p> <p>Alternatively, it would also be obvious to combine <code>if_ether.c</code> 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:</p> <p style="padding-left: 40px;">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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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.</p>

EXHIBIT D-2

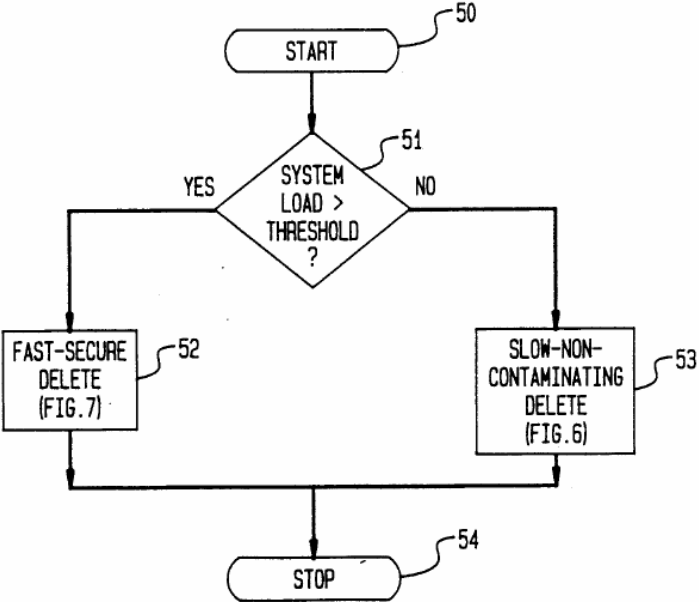
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) alone and in combination</p>
	<p><i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
	<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both <code>if_ether.c</code> and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 <code>if_ether.c</code>. 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 <code>if_ether.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>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 <code>if_ether.c</code> 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.</p> <p>Alternatively, it would also be obvious to combine <code>if_ether.c</code> with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>This decision routine should take several things into account: 1) the volume</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both <code>if_ether.c</code> 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 <code>if_ether.c</code>. 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 <code>if_ether.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120		BSD 4.2 <code>netinet/if_ether.c</code>, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz ("<code>if_ether.c</code>") alone and in combination
		<p>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 <code>if_ether.c</code> 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <code>if_ether.c</code> 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 <code>if_ether.c</code> 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 <code>if_ether.c</code> 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.</p> <p>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</p>

EXHIBIT D-2

Asserted Claims From U.S. Pat. No. 5,893,120	BSD 4.2 netinet/if_ether.c, available at http://ftp.math.utah.edu/pub/mirrors/minnie.tuhs.org/4BSD/Distributions/4.2BSD/srcsys.tar.gz (“if_ether.c”) 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.” ‘120 at 7:10-15.

EXHIBIT D-3

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain (“<code>vfs_cache.c</code>”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, <code>vfs_cache.c</code> discloses an information storage and retrieval system.</p> <p>For example, the implementation of the name cache in <code>vfs_cache.c</code> in FreeBSD includes an information storage and retrieval system that stores and retrieves names found by directory scans.</p>
<p>[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,</p>	<p>[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,</p>	<p><code>vfs_cache.c</code> 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. <code>vfs_cache.c</code> 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.</p> <p>For example, <code>vfs_cache.c</code> describes the use of a hash table, <code>nchashtbl</code>, with external chaining using linked lists to resolve collisions. <i>See</i> lines 75-84.</p> <p>The records in the system <code>vfs_cache.c</code> discloses includes records, at least some of which automatically expire. <i>See, e.g.</i>, lines 51-69, 214-226.</p> <p>For example, <code>vfs_cache.c</code> maintains a list of least recently used entries in the hash table in the structure <code>nclruhead</code>. Line 76. An entry automatically expires when (1) it is the least recently used entry, (2) the function <code>cache_enter()</code> tries to insert another entry into <code>nchashtbl</code> and (3) <code>nchashtbl</code> is already full. <i>See</i> lines 51-69, 214-226.</p>
<p>[1b] a record search means utilizing a search key to</p>	<p>[5b] a record search means utilizing a search key to</p>	<p><code>vfs_cache.c</code> discloses a record search means utilizing a search key to access the linked list. <code>vfs_cache.c</code> also discloses a record search means</p>

EXHIBIT D-3

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
<p>access the linked list,</p>	<p>access a linked list of records having the same hash address,</p>	<p>utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, the function <code>cache_enter()</code> utilizes a search key to access a linked list of records having the same hash address. <i>See</i> lines 193-246.</p>
<p>[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</p>	<p>[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</p>	<p><code>vfs_cache.c</code> 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.</p> <p>For example, the function <code>cache_enter()</code> accesses the <code>nchashtbl</code> structure and identifies an expired entry, which it removes from the linked list of records when it adds another entry to the hash table. <i>See</i> lines 214-245.</p>
<p>[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.</p>	<p>[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.</p>	<p><code>vfs_cache.c</code> 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. <code>vfs_cache.c</code> also discloses means, utilizing the record search means, for inserting 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.</p> <p>For example, the function <code>cache_enter()</code> accesses the <code>nchashtbl</code> structure and identifies an expired entry, which it removes from the linked list of records when it adds another entry to the hash table. <i>See</i> lines 193-245.</p> <p>To the extent <code>cache_enter()</code> does not include means for retrieving and deleting records utilizing the record search means, it would have been obvious to one skilled in the art that retrieval or deletion could have been done as well as the insert, since these are all basic functions that can be performed on a hash</p>

EXHIBIT D-3

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
		<p>table or a linked list in similar ways. <i>See, e.g.</i>, "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.</p>
<p>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.</p>	<p>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.</p>	<p><code>vfs_cache.c</code> discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, the function <code>cache_enter()</code> only removes an expired element when the <code>arptab</code> structure is full. In this way, the function dynamically determines the maximum number of elements to remove by computing whether to remove some or none of the expired elements. <i>See</i> lines 193-245.</p> <p>To the extent <code>vfs_cache.c</code> does not disclose this element, it would have been obvious to one of ordinary skill in the art.</p> <p><code>vfs_cache.c</code> 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.</p> <p>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.</p>

EXHIBIT D-3

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
	<p>As summarized in Dirks,</p> <p>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.</p> <p>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:</p>

EXHIBIT D-3

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
	<p>$k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$<p style="text-align: right;"><i>Id.</i> at 8:12-30.</p><p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p><p>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.</p><p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p><p>As both <code>vfs_cache.c</code> and Dirks relate to deletion of aged records, 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 <code>vfs_cache.c</code>. 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,</p></p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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 <code>vfs_cache.c</code> nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>vfs_cache.c</code> and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <code>vfs_cache.c</code> 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.</p> <p>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 <code>vfs_cache.c</code> 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.</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>Further, one of ordinary skill in the art would be motivated to combine <code>vfs_cache.c</code> with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> with Thatte.</p> <p>Alternatively, it would also be obvious to combine <code>vfs_cache.c</code> 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT D-3

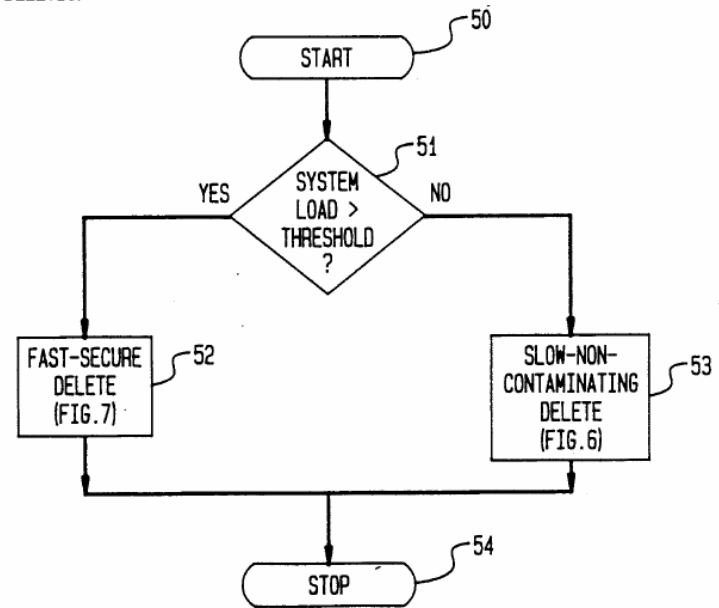
Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>4,996,663 to Nemes at 2:24-34 ("The '663 patent").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-3

Asserted Claims From
U.S. Pat. No. 5,893,120

FreeBSD `sys/kernel/vfs_cache.c` v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("`vfs_cache.c`") alone and in combination

FIG. 5
HYBRID
DELETION



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-

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both <code>vfs_cache.c</code> and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 <code>vfs_cache.c</code>. 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 <code>vfs_cache.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>vfs_cache.c</code></p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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.</p> <p>Alternatively, it would also be obvious to combine <code>vfs_cache.c</code> with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code>. 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 <code>vfs_cache.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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 <code>vfs_cache.c</code> and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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.</p> <p>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</p>

EXHIBIT D-3

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, <code>vfs_cache.c</code> 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. <code>vfs_cache.c</code> 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.</p> <p>For example, <code>vfs_cache.c</code> describes the use of a hash table, <code>nchashtbl</code>, with external chaining using linked lists to resolve collisions. <i>See</i> lines 75-84.</p> <p>The records in the system <code>vfs_cache.c</code> discloses includes records, at least some of which automatically expire. <i>See, e.g.</i>, lines 51-69, 214-226.</p> <p>For example, <code>vfs_cache.c</code> maintains a list of least recently used entries in the hash table in the structure <code>nclrhead</code>. Line 76. An entry automatically expires when (1) it is the least recently used entry, (2) the function <code>cache_enter()</code> tries to insert another entry into <code>nchashtbl</code> and (3) <code>nchashtbl</code> is already full. <i>See</i>, lines 51-69, 214-226.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p><code>vfs_cache.c</code> discloses accessing the linked list of records. <code>vfs_cache.c</code> also discloses accessing a linked list of records having same hash address</p> <p>For example, the <code>cache_enter()</code> function in <code>vfs_cache.c</code> accesses the linked list that stores records having the same hash address. <i>See</i> lines 193-245.</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain (“ <code>vfs_cache.c</code> ”) 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,	<code>vfs_cache.c</code> discloses identifying at least some of the automatically expired ones of the records. For example, the function <code>cache_enter()</code> accesses the <code>nchashtbl</code> structure and identifies an expired entry. <i>See</i> lines 193-245.
[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	<code>vfs_cache.c</code> discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. For example, the function <code>cache_enter()</code> removes the previously identified expired record from the linked list of records when it adds another entry to the hash table. <i>See</i> lines 193-245.
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<code>vfs_cache.c</code> discloses inserting, retrieving or deleting one of the records from the system following the step of removing. For example, function <code>cache_enter()</code> inserts a new entry into the <code>nchashtbl</code> structure after the expired element is removed. <i>See</i> lines 236-45.
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.	<code>vfs_cache.c</code> discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed. For example, the function <code>cache_enter()</code> only removes an expired element when the <code>arptab</code> structure is full. In this way, the function dynamically determines the maximum number of elements to remove. <i>See</i> lines 193-245. To the extent <code>vfs_cache.c</code> does not disclose this element, it would have been obvious to one of ordinary skill in the art.

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p><code>vfs_cache.c</code> 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-3

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
		<p>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 <code>RFLG</code> 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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both <code>vfs_cache.c</code> and Dirks relate to deletion of aged records, 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 <code>vfs_cache.c</code>. 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 <code>vfs_cache.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>vfs_cache.c</code> and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in <code>vfs_cache.c</code> 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</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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.</p> <p>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 <code>vfs_cache.c</code> 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine <code>vfs_cache.c</code> with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> with Thatte.</p> <p>Alternatively, it would also be obvious to combine <code>vfs_cache.c</code> with the</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>'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:</p> <p>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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-3

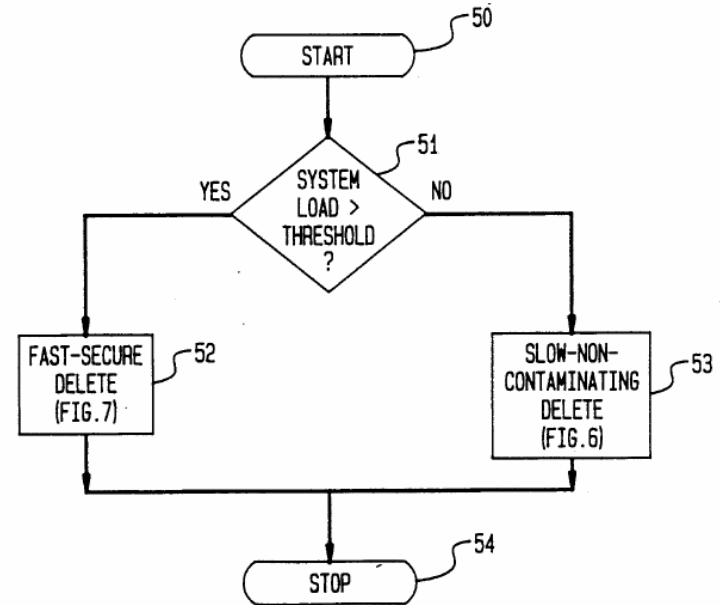
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both <code>vfs_cache.c</code> and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 <code>vfs_cache.c</code>. 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 <code>vfs_cache.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 <code>vfs_cache.c</code></p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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.</p> <p>Alternatively, it would also be obvious to combine <code>vfs_cache.c</code> with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code>. 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 <code>vfs_cache.c</code> would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-3

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination
		<p>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 <code>vfs_cache.c</code> 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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 <code>vfs_cache.c</code> 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.</p> <p>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</p>

EXHIBIT D-3

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>FreeBSD <code>sys/kernel/vfs_cache.c</code> v. 1.17, available at http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/sys/kern/vfs_cache.c?rev=1.18;content-type=text%2Fplain ("<code>vfs_cache.c</code>") alone and in combination</p>
	<p>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.</p>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, arp.c discloses an information storage and retrieval system.</p> <p>For example, in arp.c, discloses a hash table of linked lists of automatically expiring data. <i>See</i>, arp.c from FreeBSD (1994) (hereinafter “arp.c”) at Lines 360-448.</p> <p>In arp.c, the “entry” data structure is a linked list <code>*pentry = entry->next; /* delete from linked list */</code> <i>See</i> arp.c at line 416.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[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,</p>	<p>[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,</p>	<p>arp.c 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. arp.c 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.</p> <p>In arp.c, the “entry” data structure is a linked list <code>*pentry = entry->next; /* delete from linked list */</code> <i>See</i> arp.c at line 416.</p> <p>arp.c includes a function that uses a hash to determine which linked to traverse:</p> <pre>hash = HASH(entry >ip); pentry = &arp_tables[hash]; while (_pentry != NULL) {</pre>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<pre> if (_pentry == entry) { *entry = entry >next; /* delete from linked list */ del_timer(&entry >timer); restore_flags(flags); arp_release_entry(entry); return; 420 } pentry = &(_pentry) >next; } </pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p>If the entry is not resolved within a specific amount of time, the entry (which is a linked list element) is automatically freed (expired).</p> <pre> /* * This function is called, if an entry is not resolved in ARP_RES_TIME. * Either resend a request, or give it up and free the entry. */ </pre> <p><i>See</i>, arp.c at lines 361-362.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>arp.c discloses a record search means utilizing a search key to access the linked list. arp.c also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>arp.c includes a function that uses a hash to determine which linked to traverse:</p>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<pre> hash = HASH(entry >ip); pentry = &arp_tables[hash]; while (_pentry != NULL){ if (_pentry == entry){ *entry = entry >next; /* delete from linked list */ del_timer(&entry >timer); restore_flags(flags); arp_release_entry(entry); return; 420 } pentry = &(_pentry) >next; } </pre> <p align="center"><i>See</i>, arp.c at Lines 195-210.</p> <p align="center"><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[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</p>	<p>[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</p>	<p>arp.c 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. arp.c 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</p> <p>For example, arp.c deletes an entry that meets specified criteria:</p> <pre> while (*pentry != NULL) { if (*pentry == entry) { *pentry = entry->next; /* delete from linked list */ </pre>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<pre> del_timer(&entry->timer); restore_flags(flags); arp_release_entry(entry); arp_cache_stamp++; return; } pentry = &(*pentry)->next; } </pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[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.</p>	<p>[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.</p>	<p>arp.c 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. arp.c 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.</p> <p>For example, arp.c includes a function that uses a hash to determine which linked to traverse:</p> <pre> hash = HASH(entry >ip); pentry = &arp_tables[hash]; while (_pentry != NULL){ if (_pentry == entry){ *entry = entry >next; /* delete from linked list */ del_timer(&entry >timer); restore_flags(flags); arp_release_entry(entry); return; 420 } </pre>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<pre> pentry = &(_pentry) ->next; } </pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p>arp.c also includes a function that deletes an entry that meets specified criteria:</p> <pre> while (*pentry != NULL) { if (*pentry == entry) { *pentry = entry->next; /* delete from linked list */ del_timer(&entry->timer); restore_flags(flags); arp_release_entry(entry); arp_cache_stamp++; return; } pentry = &(*pentry) ->next; } </pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p>Any code that calls this function would meet this limitation because it is “utilizing the record search mean . . .” i.e., this function.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>2. The information storage</p>	<p>6. The information storage</p>	<p>arp.c combined with Dirks, Thatte, the '663 patent and/or the Opportunistic</p>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
<p>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.</p>	<p>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.</p>	<p>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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), See also, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>As both arp.c and Dirks relate to deletion of aged records, 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 arp.c. 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 arp.c nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 arp.c and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in arp.c 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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>Thatte, which is hereby incorporated by reference in its entirety.</p> <p>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 arp.c 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine arp.c with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in arp.c 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 arp.c 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 arp.c with Thatte.</p> <p>Alternatively, it would also be obvious to combine arp.c 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:</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-4

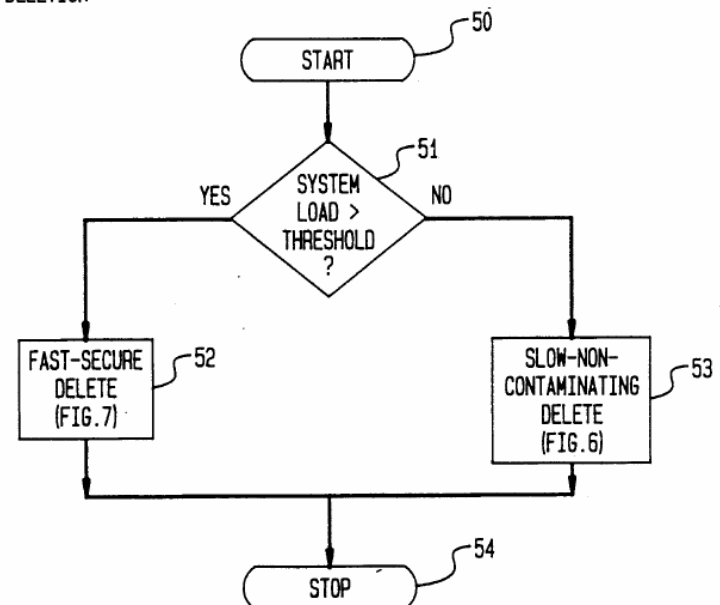
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>arp.c from FreeBSD (1994) (hereinafter “arp.c”), See also, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), See also, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both arp.c and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 arp.c. 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 arp.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 arp.c and would have seen the benefits of doing so. One such benefit, for example, is that the</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>system would avoid performing deletions when the system load exceeded a threshold.</p> <p>Alternatively, it would also be obvious to combine arp.c with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both arp.c 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 arp.c. 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 arp.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>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 arp.c and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in arp.c 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 arp.c 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 arp.c 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.</p> <p>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</p>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<p>many records to delete can be a dynamic one.” ‘120 at 7:10-15.</p> <p><i>See e.g.</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, arp.c 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. arp.c 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.</p> <p>In arp.c, the “entry” data structure is a linked list <code>*pentry = entry->next; /* delete from linked list */</code> <i>See arp.c at line 416.</i></p> <p>arp.c includes a function that uses a hash to determine which linked to traverse:</p> <pre> hash = HASH(entry >ip); pentry = &arp_tables[hash]; while (_pentry != NULL){ if (_pentry == entry){ *entry = entry >next; /* delete from linked list */ del_timer(&entry >timer); restore_flags(flags); arp_release_entry(entry); return; 420 } pentry = &(_pentry) >next; } </pre>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<p>If the entry is not resolved within a specific amount of time, the entry (which is a linked list element) is automatically freed (expired).</p> <pre> /* * This function is called, if an entry is not resolved in ARP_RES_TIME. * Either resend a request, or give it up and free the entry. */ </pre> <p><i>See</i>, arp.c at lines 361-362.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>arp.c discloses accessing a linked list of records. arp.c also discloses accessing a linked list of records having same hash address.</p> <p>In arp.c, the “entry” data structure is a linked list <code>*pentry = entry->next;</code> <i>/* delete from linked list */</i></p> <p><i>See</i> arp.c at line 416.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>arp.c discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, arp.c includes a function that uses a hash to determine which linked to traverse:</p> <pre> hash = HASH(entry >ip); pentry = &arp_tables[hash]; while (_pentry != NULL){ </pre>

EXHIBIT D-4

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
		<pre> if (_pentry == entry) { *entry = entry >next; /* delete from linked list */ del_timer(&entry >timer); restore_flags(flags); arp_release_entry(entry); return; 420 } pentry = &(_pentry) >next; } </pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>arp.c discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, arp. deletes an entry that meets specified criteria:</p> <pre> while (*pentry != NULL) { if (*pentry == entry) { *pentry = entry->next; /* delete from linked list */ del_timer(&entry->timer); restore_flags(flags); arp_release_entry(entry); arp_cache_stamp++; return; } } </pre>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
		<pre> } pentry = &(*pentry)->next; }</pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>arp.c discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, arp.c deletes an entry that meets specified criteria after arp.c traverses the linked list searching for the record of that criteria:</p> <pre>while (*pentry != NULL) { if (*pentry == entry) { *pentry = entry->next; /* delete from linked list */ del_timer(&entry->timer); restore_flags(flags); arp_release_entry(entry); arp_cache_stamp++; return; } pentry = &(*pentry)->next; }</pre> <p><i>See</i>, arp.c at Lines 195-210.</p> <p><i>See also</i>, arp.c at Lines 360-448. <i>See also</i>, arp.c from Linux 1.1.20 (1994).</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
<p>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.</p>	<p>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.</p>	<p>arp.c 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p style="padding-left: 40px;">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.</p> <p>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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), See also, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>As both arp.c and Dirks relate to deletion of aged records, 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 arp.c. 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 arp.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 arp.c and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in arp.c 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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>Thatte, which is hereby incorporated by reference in its entirety.</p> <p>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 arp.c 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine arp.c with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in arp.c 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 arp.c 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 arp.c with Thatte.</p> <p>Alternatively, it would also be obvious to combine arp.c 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:</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-4

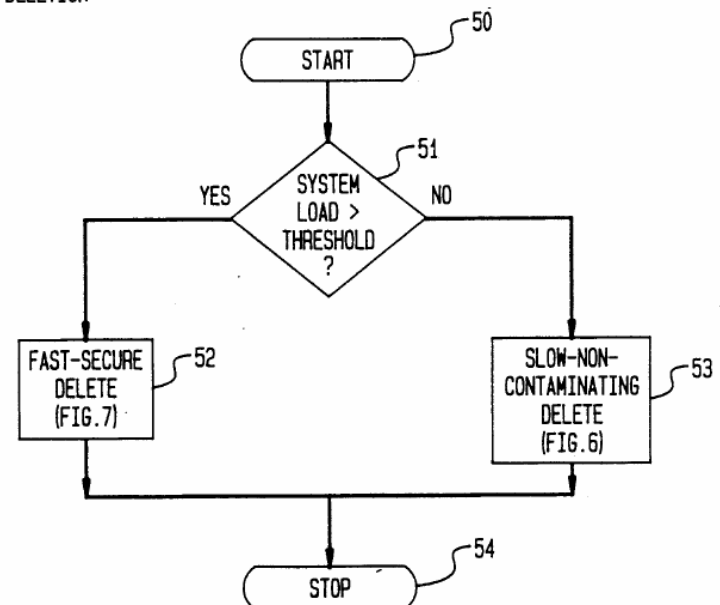
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>arp.c from FreeBSD (1994) (hereinafter “arp.c”), See also, arp.c from Linux 1.1.20 (1994) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both arp.c and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 arp.c. 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 arp.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 arp.c and would have seen the benefits of doing so. One such benefit, for example, is that the</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>system would avoid performing deletions when the system load exceeded a threshold.</p> <p>Alternatively, it would also be obvious to combine arp.c with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120		arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both arp.c 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 arp.c. 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 arp.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) alone and in combination
	<p>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 arp.c 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in arp.c 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 arp.c 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 arp.c 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.</p> <p>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</p>

EXHIBIT D-4

Asserted Claims From U.S. Pat. No. 5,893,120	arp.c from FreeBSD (1994) (hereinafter “arp.c”), <i>See also</i>, arp.c from Linux 1.1.20 (1994) 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.” ‘120 at 7:10-15. <i>See e.g.</i> , arp.c at Lines 360-448. <i>See also</i> , arp.c from Linux 1.1.20 (1994).

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, wavelan_cs.c discloses an information storage and retrieval system.</p> <p>For example, an information storage and retrieval system disclosed by wavelan_cs.c is a linked list:</p> <pre> /* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; </pre> <p><i>See, wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) at Lines 4630-4635. See also, wavelan_cs.c from Linux 2.4.26.</i></p> <p><i>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</i></p>
<p>[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,</p>	<p>[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,</p>	<p>wavelan_cs.c 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. wavelan_cs.c 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.</p> <p>For example, wavelan_cs.c includes a function that deletes an instance of a driver from the linked list if the device is released. Thus, releasing the device causes the device to automatically expire.</p> <pre> /* * This deletes a driver "instance". The device is </pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre> de-registered with * Card Services. If it has been released, all local data structures * are freed. Otherwise, the structures will be freed when the device * is released. */ See wavelan_cs.c at Lines 4595-4600. The data structure is a linked list: /* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; See wavelan_cs.c at Lines 4630-4635. See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26. </pre>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>wavelan_cs.c discloses a record search means utilizing a search key to access the linked list. wavelan_cs.c also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list.</p> <pre> /* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; </pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre> while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL) { #ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } prev->next = link->next </pre> <p>See wavelan_cs.c at Lines 4632-4644.</p> <p>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</p>
<p>[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</p>	<p>[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</p>	<p>wavelan_cs.c 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. wavelan_cs.c 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.</p> <p>For example, wavelan_cs.c includes the functionality to remove expired records from the linked list:</p> <pre> /* Remove the interface data from the linked list */ if(dev_list == link) </pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre align="center">dev_list = link->next;</pre> <p><i>See wavelan_cs.c at Lines 4632-4644.</i></p> <p><i>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</i></p>
<p>[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.</p>	<p>[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.</p>	<p>wavelan_cs.c 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. wavelan_cs.c 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.</p> <p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list. Further, wavelan_cs.c will remove records from the linked list as it traverses the linked list. Any code that calls this function would “utilize the record search means.”</p> <pre align="center">/* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL)</pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre> { #ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } </pre> <p align="center">prev->next = link->next See wavelan_cs.c at Lines 4632-4644.</p> <p align="center">See, e.g., wavelan_cs.c at Lines 4596-4678.</p>
<p>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.</p>	<p>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.</p>	<p>wavelan_cs.c 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both wavelan_cs.c and Dirks relate to deletion of aged records, 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 wavelan_cs.c. 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 wavelan_cs.c nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>examine during each step of the sweeping process with wavelan_cs.c and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in wavelan_cs.c 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.</p> <p>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 wavelan_cs.c 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine wavelan_cs.c with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in wavelan_cs.c 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 wavelan_cs.c with the teachings of Thatte would solve this problem by dynamically determining how many records to delete based on, among other</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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 wavelan_cs.c with Thatte.</p> <p>Alternatively, it would also be obvious to combine wavelan_cs.c 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>This hybrid hashing technique has the decided advantage of</p>

EXHIBIT D-5

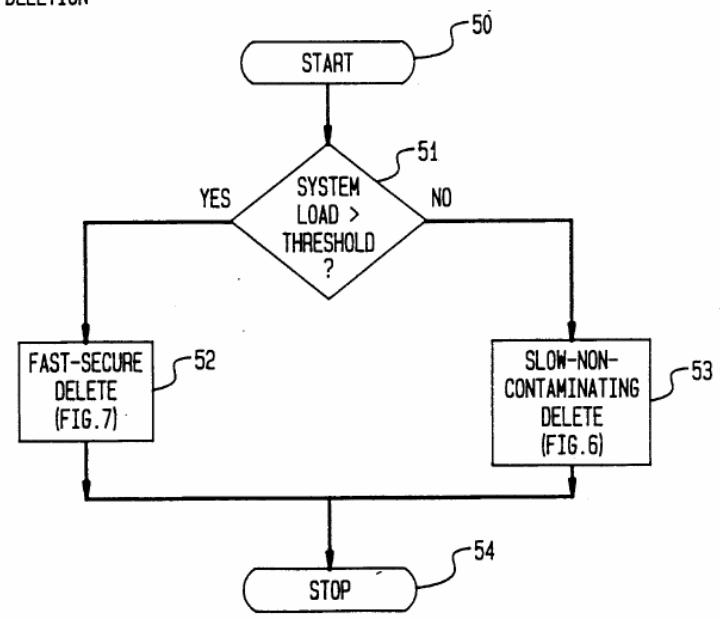
Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating deletion is used when the load on the system is at lower levels. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both wavelan_cs.c and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 wavelan_cs.c. 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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>combining the '663 patent's deletion decision procedure with wavelan_cs.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 wavelan_cs.c 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.</p> <p>Alternatively, it would also be obvious to combine wavelan_cs.c with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>Every time a user-input routine is invoked, a decision routine can decide</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both wavelan_cs.c 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 wavelan_cs.c. 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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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 wavelan_cs.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 wavelan_cs.c and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in wavelan_cs.c 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 wavelan_cs.c 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 wavelan_cs.c 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.</p>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, wavelan_cs.c 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. wavelan_cs.c 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.</p> <p>For example, wavelan_cs.c includes a function that deletes an instance of a driver from the linked list if the device is released. Thus, releasing the device causes the record to automatically expire.</p> <pre> /* * This deletes a driver "instance". The device is de-registered with * Card Services. If it has been released, all local data structures * are freed. Otherwise, the structures will be freed when the device * is released. </pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre> */ See wavelan_cs.c at Lines 4595-4600. The data structure is a linked list: /* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; See wavelan_cs.c at Lines 4630-4635. See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26. </pre>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>wavelan_cs.c discloses accessing a linked list of records. wavelan_cs.c also discloses accessing a linked list of records having same hash address.</p> <p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list.</p> <pre> /* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL) { </pre>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<pre>#ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } prev->next = link->next</pre> <p>See wavelan_cs.c at Lines 4632-4644.</p> <p>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</p>
[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,	<p>wavelan_cs.c discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list.</p> <pre>/* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL)</pre>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<pre> { #ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } </pre> <p>prev->next = link->next See wavelan_cs.c at Lines 4632-4644.</p> <p>For example, wavelan_cs.c includes a function that deletes an instance of a driver from the linked list if the device is released. Thus, releasing the device causes the record to automatically expire.</p> <pre> /* * This deletes a driver "instance". The device is de-registered with * Card Services. If it has been released, all local data structures * are freed. Otherwise, the structures will be freed when the device * is released. */ </pre> <p>See wavelan_cs.c at Lines 4595-4600.</p> <p>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</p>
<p>[3c] removing at least some of the automatically</p>	<p>[7c] removing at least some of the automatically</p>	<p>wavelan_cs.c discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
expired records from the linked list when the linked list is accessed.	expired records from the linked list when the linked list is accessed, and	<p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list. Further, wavelan_cs.c will remove records from the linked list as it traverses the linked list.</p> <pre data-bbox="951 521 1969 1203">/* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL) { #ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } } prev->next = link->next</pre> <p>See wavelan_cs.c at Lines 4632-4644.</p> <p>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26.</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>wavelan_cs.c discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, wavelan_cs.c includes functionality to use a pointer to traverse a linked list. Further, wavelan_cs.c will remove records from the linked list as it traverses the linked list. The deletion will occur after wavelan_cs.c traverses through the element.</p> <pre>/* Remove the interface data from the linked list */ if(dev_list == link) dev_list = link->next; else { dev_link_t * prev = dev_list; while((prev != (dev_link_t *) NULL) && (prev->next != link)) prev = prev->next; if(prev == (dev_link_t *) NULL) { #ifdef DEBUG_CONFIG_ERRORS printk(KERN_WARNING "wavelan_detach : Attempting to remove a nonexistent device.\n"); #endif return; } prev->next = link->next </pre> <p>See wavelan_cs.c at Lines 4632-4644.</p>

EXHIBIT D-5

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
		<p><i>See also, wavelan_cs.c at Lines 4596-4678. See also, wavelan_cs.c from Linux 2.4.26</i></p>
<p>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.</p>	<p>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.</p>	<p>wavelan_cs.c 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>After [a] new VSID has been allocated, the system checks a flag RFLG</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both wavelan_cs.c and Dirks relate to deletion of aged records, 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 wavelan_cs.c. 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 wavelan_cs.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 wavelan_cs.c and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in wavelan_cs.c 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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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.</p> <p>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 wavelan_cs.c 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine wavelan_cs.c with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in wavelan_cs.c 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 wavelan_cs.c 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 wavelan_cs.c with Thatte.</p> <p>Alternatively, it would also be obvious to combine wavelan_cs.c with the '663</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>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:</p> <p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-5

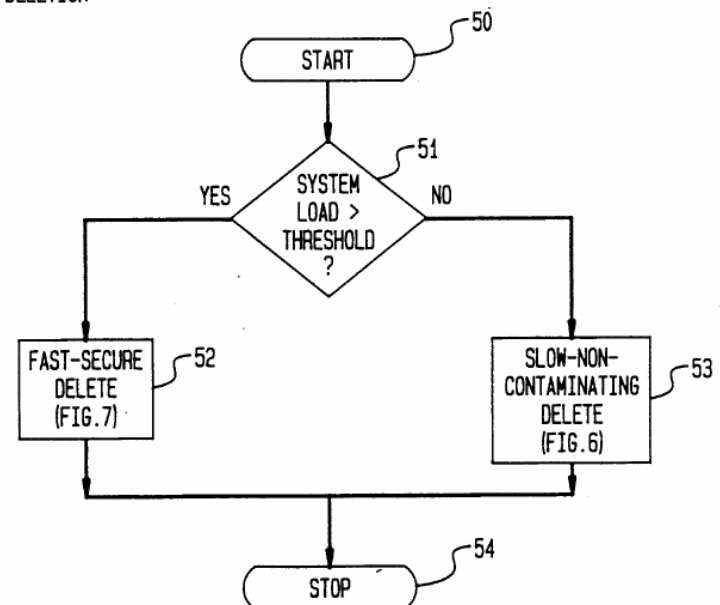
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both wavelan_cs.c and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 wavelan_cs.c. 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 wavelan_cs.c would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 wavelan_cs.c</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
	<p>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.</p> <p>Alternatively, it would also be obvious to combine wavelan_cs.c with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>user has had an opportunity to interact, and 3) the height of the stack relative to its average height at reads since the last scavenger. If the product of the allocation and the compute time is high, and if the stack is low, the scavenger favorability measure is high. If it is especially high, a multi-generation scavenger is in order. <i>Id.</i></p> <p>If these heuristics fail and a scavenger 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 scavenger pause within a larger pause. <i>Design of the Opportunistic Garbage Collector at 32.</i></p> <p>As both wavelan_cs.c 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 wavelan_cs.c. 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 wavelan_cs.c would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120		wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) alone and in combination
		<p>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 wavelan_cs.c 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in wavelan_cs.c 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 wavelan_cs.c 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 wavelan_cs.c 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.</p> <p>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</p>

EXHIBIT D-5

Asserted Claims From U.S. Pat. No. 5,893,120	wavelan_cs.c from FreeBSD (1995) (hereinafter “wavelan_cs.c”) 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.” ‘120 at 7:10-15.

EXHIBIT D-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LISP alone and in combination</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, LISP discloses an information storage and retrieval system.</p> <p>For example,</p> <p>“This paper should be useful to readers interested in data structures and their applications in compiler construction, language design, and database management.” Jacques Cohen, <i>Garbage Collection of Linked Data Structures</i>, Computing Surveys, 341, 342 (hereinafter “Cohen”).</p> <p>“This model consists of a memory, <i>i.e.</i> a one-dimensional array of words, each of which is large enough to hold (at least) the representation of a nonnegative integer which is an index into that array.” Henry G. Baker, <i>List Processing in Real Time on a Serial Computer</i>, Communications of the ACM 21, 280, second page, (April 1978) (hereinafter “Baker”).</p> <p>“There are two fundamental kinds of data in LISP: list cells and atoms . . . CAR(x) and CDR(x) return the car and cdr components of the list cell x, respectively.” Baker at 2.</p> <p>“If the method is used for the management of a large database residing on secondary storage.” Baker at 6.</p> <p>“We conceive of a huge database having millions of records, which may contain pointers to other records, being managed by our algorithm.” Baker at 12.</p>
<p>[1a] a linked list to store and provide access to records stored in a memory of the system, at least some</p>	<p>[5a] a hashing means to provide access to records stored in a memory of the system and using an</p>	<p>LISP 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. LISP 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</p>

EXHIBIT D-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LISP alone and in combination</p>
<p>of the records automatically expiring,</p>	<p>external chaining technique to store the records with same hash address, at least some of the records automatically expiring,</p>	<p>records with same hash address, at least some of the records automatically expiring.</p> <p>For example,</p> <p>“Of the hundreds of thousands of computer languages which have been invented, there is one particular family of languages whose common ancestor was the original LISP, developed by McCarthy and others in the late 1950's. [LISP History] These languages are generally characterized by a simple, fully parenthesized ("Cambridge Polish") syntax; the ability to manipulate general, linked-list data structures; a standard representation for programs of the language in terms of these structures; and an interactive programming system based on an interpreter for the standard representation. Examples of such languages are LISP 1.5 [LISP 1.5M], MacLISP [Moon], InterLISP [Teitelman], CONNIVER UIcDermott and Sussman], QM [Rullfson], PLASNA [Smith and Hewitt] [Hewitt and Smith], and SCHUIE [SCHEME] [Revised Report]. We will call this family the LISP-like languages.” Guy Lewis Steele, Jr., <i>The Art of the Interpreter or The Modularity Complex</i> (Parts Zero, One, and Two), Massachusetts Institute of Technology AI Memo No. 453 at 2 (May 1978). (Hereinafter “Steele”).</p> <p>“A concise and unified view of the numerous existing algorithms for performing garbage collection of linked data structures is presented.” Cohen Abstract.</p> <p>“The primary list processing language in use today is LISP.” Baker at 2.</p> <p>“We conceive of a huge database having millions of records, which may contain pointers to other records, being managed by our algorithm.” Baker at 12.</p> <p>”A cell becomes unused, or garbage, when it can no longer be accessed</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>through any pointer fields of any reachable cell.” Cohen at 342.</p> <p>“the property list can be found by looking in a hash table, using the address of the list cell as the key.” Baker at 10.</p> <p>“Our copying scheme gives each semispace its own hash table, and when a cell is copied over into to space, its property list pointer is entered in the "to" table under the cell's new address. Then when the copied cell is encountered by the "scan" pointer, its property list pointer is updated along with its normal components.” Baker at 10.</p> <p>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.” The applicant has conceded that both forms of collision resolution are known to those of ordinary skill in the art. <i>See, e.g.</i>, ‘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). Double hashing is another form of open addressing.</p> <p>It would have been obvious to one skilled in the art to apply the teachings in LISP to a hash table which resolves collisions using external chaining with linked lists. The method of LISP is a method for processing and garbage collecting on linked structures generally, which includes externally chained</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		records. Externally chained records would still need a method of memory management, so it would be obvious to use LISP as a method of memory management and list processing on externally chained records with the same hash address.
[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,	<p>LISP discloses a record search means utilizing a search key to access the linked list. LISP also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example,</p> <p>“This model consists of a memory, <i>i.e.</i> a one-dimensional array of words, each of which is large enough to hold (at least) the representation of a nonnegative integer which is an index into that array.” Baker at 2.</p> <p>“the property list can be found by looking in a hash table, using the address of the list cell as the key.” Baker at 10.</p> <p>As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use LISP for list processing and memory management on externally chained structures. As such, the search means utilizing the search key would be accessing a linked list of records beginning at the same hash address.</p>
[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	<p>LISP 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. LISP 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.</p> <p>For example,</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>“For example, in LISP, the function <i>cons</i> also calls the garbage collector.” Cohen at 342.</p> <p>“Baker’s modification is such that each time a cell is requested (<i>i.e.</i>, a <i>cons</i> is requested), a fixed number of cells, <i>k</i>, are moved from one semispace to the other.” Cohen at 355.</p> <p>“The amount of storage and time used by a real-time list processing system can be compared with that used by a classical list processing system using garbage collection on tasks not requiring bounded response times.” Baker at 11.</p> <p>As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use LISP for list processing and memory management on externally chained structures. 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 linked list is accessed.</p>
<p>[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.</p>	<p>[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.</p>	<p>LISP 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. LISP 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.</p> <p>For example,</p> <p>“The moving of <i>k</i> cells during a <i>cons</i> corresponds to the tracing of that many cells in classical garbage collection. By distributing some of the garbage collection tasks during list processing, Baker’s method provides a guarantee</p>

EXHIBIT D-6

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">LISP alone and in combination</p>
		<p>that actual garbage collection cannot last more than a fixed (tolerable) amount of time.” Cohen at 355.</p> <p>“A real-time list processing system is presented which continuously reclaims garbage.” Baker Abstract.</p> <p>“In order to convert MFYCA into a real-time algorithm, we force the mark ratio m to be constant by changing CONS so that it does k iterations of garbage collection before performing each allocation.” Baker at 4.</p> <p>“There is another problem caused by interleaving garbage collection with normal list processing.” Baker at 4.</p> <p>“garbage collection in our real-time system is almost identical to that in the MFYCA system, except that it is done incrementally during calls to CONS. In other words, the user program pays for the cost of cell’s reclamation at the time the cell is created by tracing some other cell.” Baker at 11.</p> <p>“We have exhibited a method for doing list-processing on a serial computer in a real-time environment . . . Our real time scheme is strikingly similar to the incremental garbage collector proposed independently by Barbacci.” Baker at 13.</p> <p>As discussed in [1a/5a], it would have been obvious to one of ordinary skill in the art to use LISP for list processing and memory management on externally chained structures. 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 linked list is accessed.</p>
<p>2. The information storage</p>	<p>6. The information storage</p>	<p>LISP discloses an information storage and retrieval system further including</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP 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.	<p>means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.</p> <p>For example,</p> <p>“Baker’s modification is such that each time a cell is requested (<i>i.e.</i>, a <i>cons</i> is requested), a fixed number of cells, <i>k</i>, are moved from one semispace to the other.” Cohen at 355.</p> <p>“With a little more effort, <i>k</i> can even be made variable in our method, thus allowing the program to dynamically optimize its space-time tradeoff.” Baker at 6.</p> <p>Lisp 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Lisp and Dirks relate to deletion of aged records, 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 Lisp. 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 Lisp nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>examine during each step of the sweeping process with Lisp and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Lisp 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.</p> <p>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 Lisp 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Lisp with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Lisp 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 Lisp 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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>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 Lisp with Thatte.</p> <p>Alternatively, it would also be obvious to combine Lisp 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:</p> <p style="padding-left: 40px;">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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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.</p>

EXHIBIT D-6

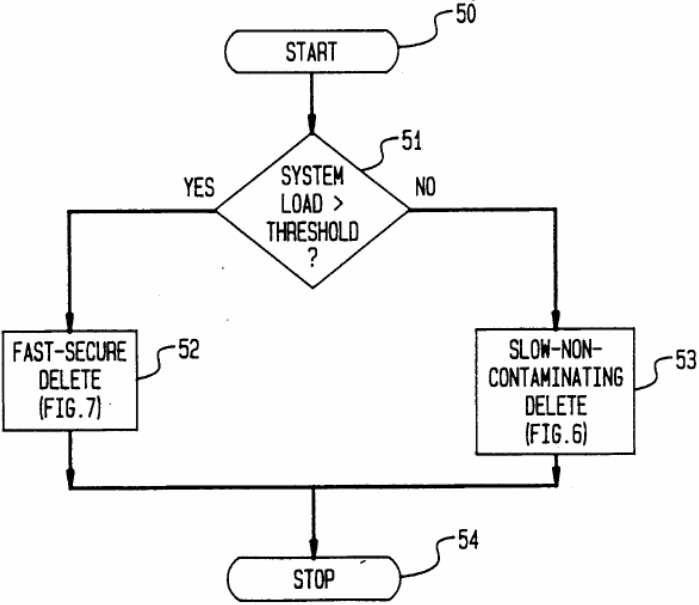
Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p><i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Lisp and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Lisp. 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 Lisp would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>based on a systems load as taught by the '663 patent and with Lisp 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.</p> <p>Alternatively, it would also be obvious to combine Lisp with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector at 32.</i></p> <p>As both Lisp 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 Lisp. 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 Lisp would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>By way of further example, one of ordinary skill in the art would have</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>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 Lisp and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Lisp 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 Lisp 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 Lisp 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.</p> <p>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.</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>Thus, the '120 patent provides motivations to combine LISP (<i>e.g.</i> the system disclosed in Baker or Cohen) with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles, in addition to motivations within the text of Baker or Cohen. Baker at 1 (“Third, processing had to be halted periodically to reclaim storage by a long process know as garbage collection, which laboriously traced every accessible cell so that those inaccessible cells could be recycled. . . . This paper presents a solution to the third problem . . . and removes the roadblock to their more general use.”); Cohen at 342 (“A most vexing aspect of garbage collection is that program execution comes to a halt while the collector attempts to reclaim storage space.”).</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, LISP 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. LISP 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.</p> <p>For example,</p> <p>“This paper should be useful to readers interested in data structures and their applications in compiler construction, language design, and database management.” Cohen at 342.</p> <p>“This model consists of a memory, <i>i.e.</i> a one-dimensional array of words, each of which is large enough to hold (at least) the representation of a nonnegative integer which is an index into that array.” Baker at 2.</p> <p>“There are two fundamental kinds of data in LISP: list cells and atoms</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>CAR(x) and CDR(x) return the car and cdr components of the list cell x, respectively.” Baker at 2.</p> <p>“If the method is used for the management of a large database residing on secondary storage.” Baker at 6.</p> <p>“We conceive of a huge database having millions of records, which may contain pointers to other records, being managed by our algorithm.” Baker at 12.</p> <p>“the property list can be found by looking in a hash table, using the address of the list cell as the key.” Baker at 10.</p> <p>“Our copying scheme gives each semispace its own hash table, and when a cell is copied over into to space, its property list pointer is entered in the "to" table under the cell's new address. Then when the copied cell is encountered by the "scan" pointer, its property list pointer is updated along with its normal components.” Baker at 10.</p> <p>“Of the hundreds of thousands of computer languages which have been invented, there is one particular family of languages whose common ancestor was the original LISP, developed by McCarthy and others in the late 1950's. [LISP History] These languages are generally characterized by a simple, fully parenthesized ("Cambridge Polish") syntax; the ability to manipulate general, linked-list data structures; a standard representation for programs of the language in terms of these structures; and an interactive programming system based on an interpreter for the standard representation. Examples of such languages are LISP 1.5 [LISP 1.5M], MacLISP [Moon], InterLISP [Teiteiman], CONNIVER McDermott and Sussman], QM [Rulfson], PLASNA [Smith and Hewitt] [Hewitt and Smith], and SCHUIE [SCHEME] [Revised</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>Report]. We will call this family the LISP-like languages.” Steele at 2.</p> <p>“A cell becomes unused, or garbage, when it can no longer be accessed through any pointer fields of any reachable cell.” Cohen at 342.</p> <p>It would have been obvious to one skilled in the art to apply the teachings in LISP to a hash table which resolves collisions using external chaining with linked lists. The method of LISP is a method for processing and garbage collecting on linked structures generally, which includes externally chained records. Externally chained records would still need a method of memory management, so it would be obvious to use LISP as a method of memory management and list processing on externally chained records with the same hash address.</p>
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	<p>LISP discloses accessing a linked list of records. LISP also discloses accessing a linked list of records having same hash address.</p> <p>For example,</p> <p>“For example, in LISP, the function <i>cons</i> also calls the garbage collector.” Cohen at 342.</p> <p>“The amount of storage and time used by a real-time list processing system can be compared with that used by a classical list processing system using garbage collection on tasks not requiring bounded response times.” Baker at 11.</p> <p>“the property list can be found by looking in a hash table, using the address of the list cell as the key.” Baker at 10.</p> <p>“This model consists of a memory, <i>i.e.</i> a one-dimensional array of words, each of which is large enough to hold (at least) the representation of a nonnegative</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>integer which is an index into that array.” Baker at 2.</p> <p>As discussed in [3/7], it would have been obvious to one of ordinary skill in the art to use LISP for list processing and memory management on externally chained structures having the same hash address.</p>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>LISP discloses identifying at least some of the automatically expired ones of the records. LISP also discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example,</p> <p>“For example, in LISP, the function <i>cons</i> also calls the garbage collector.” Cohen at 342.</p> <p>“Baker’s modification is such that each time a cell is requested (<i>i.e.</i>, a <i>cons</i> is requested), a fixed number of cells, <i>k</i>, are moved from one semispace to the other.” Cohen at 355.</p> <p>“The amount of storage and time used by a real-time list processing system can be compared with that used by a classical list processing system using garbage collection on tasks not requiring bounded response times.” Baker at 11.</p> <p>“A cell becomes unused, or garbage, when it can no longer be accessed through any pointer fields of any reachable cell.” Cohen at 342.</p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked</p>	<p>LISP discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed. LISP also discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
list is accessed.	list is accessed, and	<p>For example,</p> <p>“For example, in LISP, the function <i>cons</i> also calls the garbage collector.” Cohen at 342.</p> <p>“In order to convert MFYCA into a real-time algorithm, we force the mark ratio <i>m</i> to be constant by changing <i>CONS</i> so that it does <i>k</i> iterations of garbage collection before performing each allocation.” Baker at 4.</p> <p>“There is another problem caused by interleaving garbage collection with normal list processing.” Baker at 4.</p> <p>“garbage collection in our real-time system is almost identical to that in the MFYCA system, except that it is done incrementally during calls to <i>CONS</i>. In other words, the user program pays for the cost of cell’s reclamation at the time the cell is created by tracing some other cell.” Baker at 11.</p> <p>“We have exhibited a method for doing list-processing on a serial computer in a real-time environment Our real time scheme is strikingly similar to the incremental garbage collector proposed independently by Barbacci.” Baker at 13.</p>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>LISP discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example,</p> <p>“For example, in LISP, the function <i>cons</i> also calls the garbage collector.” Cohen at 342.</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>“In order to convert MFYCA into a real-time algorithm, we force the mark ratio m to be constant by changing CONS so that it does k iterations of garbage collection before performing each allocation.” Baker at 4.</p> <p>“There is another problem caused by interleaving garbage collection with normal list processing.” Baker at 4.</p> <p>“garbage collection in our real-time system is almost identical to that in the MFYCA system, except that it is done incrementally during calls to CONS. In other words, the user program pays for the cost of cell’s reclamation at the time the cell is created by tracing some other cell.” Baker at 11.</p> <p>“We have exhibited a method for doing list-processing on a serial computer in a real-time environment Our real time scheme is strikingly similar to the incremental garbage collector proposed independently by Barbacci.” Baker at 13.</p>
<p>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.</p>	<p>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.</p>	<p>LISP discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example,</p> <p>“Baker’s modification is such that each time a cell is requested (<i>i.e.</i>, a <i>cons</i> is requested), a fixed number of cells, k, are moved from one semispace to the other.” Cohen at 355.</p> <p>“With a little more effort, k can even be made variable in our method, thus allowing the program to dynamically optimize its space-time tradeoff.” Baker at 6.</p> <p>Lisp combined with Dirks, Thatte, the ’663 patent, and/or the Opportunistic</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>Garbage Collection Articles discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Lisp and Dirks relate to deletion of aged records, one of ordinary skill in the art would have understood how to use Dirks' dynamic decision making</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>process of determining the maximum number of records to sweep/remove in other hash tables implementations such as that described Lisp. 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 Lisp would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Lisp and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Lisp 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.</p> <p>Moreover, one of ordinary skill in the art would recognize that these</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>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 Lisp 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Lisp with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Lispcan 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 Lisp 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 Lisp with Thatte.</p> <p>Alternatively, it would also be obvious to combine Lisp 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-6

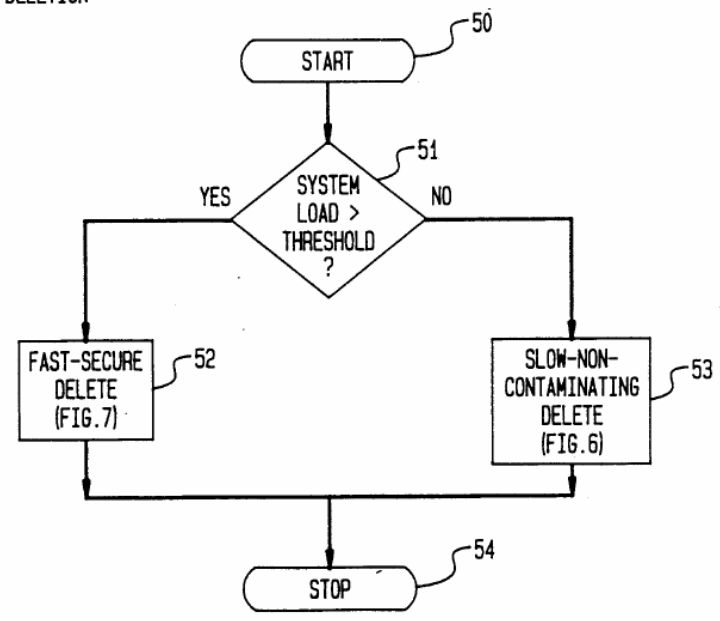
Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Lisp and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Lisp. 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 Lisp would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Lisp 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.</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>Alternatively, it would also be obvious to combine Lisp with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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-</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		<p>generation scavenge is in order. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Lisp 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 Lisp. 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 Lisp would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Lisp and would have seen the benefits of doing so. One such benefit, for example, is that the</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120	LISP alone and in combination
	<p>system would only perform deletions when the system was not already too overloaded, thus preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Lisp 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 Lisp 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 Lisp 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.</p> <p>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.</p> <p>Thus, the '120 patent provides motivations to combine LISP (<i>e.g.</i> Baker or</p>

EXHIBIT D-6

Asserted Claims From U.S. Pat. No. 5,893,120		LISP alone and in combination
		Cohen) with Thatte, Dirks, the '663 patent, and/or the Opportunistic Garbage Collection Articles, in addition to motivations within the text of Baker or Cohen. Baker at 1 (“Third, processing had to be halted periodically to reclaim storage by a long process know as garbage collection, which laboriously traced every accessible cell so that those inaccessible cells could be recycled. . . . This paper presents a solution to the third problem . . . and removes the roadblock to their more general use.”); Cohen at 342 (“A most vexing aspect of garbage collection is that program execution comes to a halt while the collector attempts to reclaim storage space.”).

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	<p>To the extent the preamble is a limitation, FreeBSD 2.0.5 discloses an information storage and retrieval system.</p> <p>For example, in kern_proc.c and proc.h, FreeBSD 2.0.5 discloses a hash table of linked lists of automatically expiring data. <u>See, e.g.</u>, struct pgrp defined at lines 61-70 of proc.h and struct proc defined at lines 72-172 of proc.h. Excerpts are below:</p> <pre>61 /* 62 * One structure allocated per process group. 63 */ 64 struct pgrp { 65 struct pgrp *pg_hforw; /* Forward link in hash bucket. */ 66 struct proc *pg_mem; /* Pointer to pgrp members. */ 67 struct session *pg_session; /* Pointer to session. */ 68 pid_t pg_id; /* Pgrp id. */ 69 int pg_jobc; /* # procs qualifying pgrp for job control */ 70 }; 72 /* 73 * Description of a process. 74 * 75 * This structure contains the information needed to manage a thread of 76 * control, known in UN*X as a process; it has references to substructures 77 * containing descriptions of things that the process uses, but may share 78 * with related processes. The process structure and the substructures 79 * are always addressable except for those marked "(PROC ONLY)" below, 80 * which might be addressable only on a processor on which the process</pre>

¹ Publicly available as of June 10, 1995; available at <ftp://ftp-archive.freebsd.org/pub/FreeBSD-Archive/old-releases/i386/2.0.5-RELEASE/src/>.

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<pre> 81 * is running. 82 */ 83 struct proc { 84 struct proc *p_forw; /* Doubly-linked run/sleep queue. */ 85 struct proc *p_back; 86 struct proc *p_next; /* Linked list of active procs */ 87 struct proc **p_prev; /* and zombies. */ </pre>
<p>[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,</p>	<p>[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,</p>	<p>FreeBSD 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” and “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 pidhash[] and pgrp[] structures defined in param.c meet the “hashing means” limitation.</p> <pre> 206 struct proc *pidhash[PIDHSZ]; 207 struct pgrp *pgrp[PIDHSZ]; </pre> <p>Also, FreeBSD defines the pgrp structure as including a forward link in the hash bucket as well as a pointer to a linked list of proc structures. This is an example of how FreeBSD meets the “linked list” and “external chaining” limitations, as shown in the excerpts from proc.h below.</p> <pre> 64 struct pgrp { 65 struct pgrp *pg_hforw; /* Forward link in hash bucket. */ 66 struct proc *pg_mem; /* Pointer to pgrp members. */ 67 struct session *pg_session; /* Pointer to session. */ 68 pid_t pg_id; /* Pgrp id. */ 69 int pg_jobc; /* # procs qualifying pgrp for job control */ 70 }; </pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<pre>83 struct proc { 84 struct proc *p_forw; /* Doubly-linked run/sleep queue. */ 85 struct proc *p_back; 86 struct proc *p_next; /* Linked list of active procs */ 87 struct proc **p_prev; /* and zombies. */</pre> <p>Examples of how FreeBSD uses the hashing technique with external chaining can be found in the enterpgrp() function in kern_proc.c, such as the following:</p> <pre>230 pgrp->pg_hforw = pgrphash[n = PIDHASH(pgid)]; 231 pgrphash[n] = pgrp;</pre> <p>The function that calls enterpgrp() passes in a proc structure, as shown in lines 175-79.</p> <pre>175 int 176 enterpgrp(p, pgid, mksex) 177 register struct proc *p; 178 pid_t pgid; 179 int mksex;</pre> <p>Code within the enterpgrp() structure unlinks the proc from its old process group, as shown below in lines 248-53 of kern_proc.c. Also, enterpgrp() calls pgdelete() if the process group is empty, as shown in lines 261-62. Depending on claim construction, these are two examples of automatic expiration.</p> <pre>245 /* 246 * unlink p from old process group 247 */</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<pre> 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break; 252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); </pre>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>FreeBSD discloses “a record search means utilizing a search key to access the linked list” and “a record search means utilizing a search key to access a linked list of records having the same hash address.”</p> <p>The following code from the enterpgrp() function in kern_proc.c is an example of accessing a linked list of records having the same hash address and using a search key to access a linked list. The [n] index is an example of a search key. The enterpgrp() function is an example of a “record search means” as claimed.</p> <pre> 230 pgrp->pg_hforw = pgrphash[n = PIDHASH(pgid)]; 231 pgrphash[n] = pgrp; </pre>
<p>[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</p>	<p>[5c] the record search means including means for identifying and removing at least some expired ones of the records from the linked list of</p>	<p>FreeBSD 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” and “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,” as claimed.</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
list when the linked list is accessed, and	records when the linked list is accessed, and	<p>For example, code within the enterpgrp() structure unlinks the proc from its old process group, as shown below in lines 248-53 of kern_proc.c. Also, enterpgrp() calls pgdelete() if the process group is empty, as shown in lines 261-62. These are two examples of automatic expiration.</p> <pre> 245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break; 252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); </pre>
[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	FreeBSD 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 “ and “meals [<i>sic</i> “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,” as claimed. An example of a “means utilizing the record search means” can be found in kern_prot.c. For example, the setsid() function calls enterpgrp(), as shown below at line 196.

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>records.</p> <pre>186 int 187 setsid(p, uap, retval) 188 register struct proc *p; 189 struct args *uap; 190 int *retval; 191 { 192 193 if (p->p_pgid == p->p_pid pgfind(p->p_pid)) { 194 return (EPERM); 195 } else { 196 (void)enterpgrp(p, p->p_pid, 1); 197 *retval = p->p_pid; 198 return (0); 199 } 200 }</pre> <p>An example of “retrieving” can be found in enterpgrp(), in the for loop found at lines 248-53. Depending on claim construction, an example of “removing” and “deleting” can be found in the call to pgdelete() at line 262, and the operation of pgdelete() at lines 300-22. An example of “inserting” can be found at lines 266-68. Each of these steps is performed within enterpgrp() and “at the same time,” as recited in the claims.</p> <pre>245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break;</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); 263 /* 264 * link into new one 265 */ 266 p->p_pgrp = pgrp; 267 p->p_pgrpnxt = pgrp->pg_mem; 268 pgrp->pg_mem = p; 269 return (0); 297 /* 298 * delete a process group 299 */ 300 void 301 pgdelete(pgrp) 302 register struct pgrp *pgrp; 303 { 304 register struct pgrp **pgp = &pgrphash[PIDHASH(pgrp->pg_id)]; 305 306 if (pgrp->pg_session->s_ttyp != NULL && 307 pgrp->pg_session->s_ttyp->t_pgrp == pgrp) 308 pgrp->pg_session->s_ttyp->t_pgrp = NULL; 309 for (; *pgp; pgp = &(*pgp)->pg_hforw) { 310 if (*pgp == pgrp) {</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>311 *pgp = pgrp->pg_hforw; 312 break; 313 } 314 } 315 #ifdef DIAGNOSTIC 316 if (pgp == NULL) 317 panic("pgdelete: can't find pgrp on hash chain"); 318 #endif 319 if (--pgrp->pg_session->s_count == 0) 320 FREE(pgrp->pg_session, M_SESSION); 321 FREE(pgrp, M_PGRP); 322 }</pre> <p>FreeBSD 2.0.5 defines FREE() as used in lines 320-21 of kern_proc.c in malloc.h, as shown below. Depending on whether KMEMSTATS or DIAGNOSTIC is defined, FREE() is either set to free() in line 288 or defined as in lines 304-20.</p> <pre>283 /* 284 * Macro versions for the usual cases of malloc/free 285 */ 286 #if defined(KMEMSTATS) defined(DIAGNOSTIC) 287 #define MALLOC(space, cast, size, type, flags) \ 288 (space) = (cast)malloc((u_long)(size), type, flags) 289 #define FREE(addr, type) free((caddr_t)(addr), type) 290 291 #else /* do not collect statistics */ 292 #define MALLOC(space, cast, size, type, flags) { \ 293 register struct kmembuckets *kbp = &bucket[BUCKETINDX(size)]; \ 294 long s = splimp(); \ 295 if (kbp->kb_next == NULL) { \</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>296 (space) = (cast)malloc((u_long)(size), type, flags); \ 297 } else { \ 298 (space) = (cast)kbp->kb_next; \ 299 kbp->kb_next = *(caddr_t*)(space); \ 300 } \ 301 splx(s); \ 302 } 303 304 #define FREE(addr, type) { \ 305 register struct kmembuckets *kbp; \ 306 register struct kmemusage *kup = btokup(addr); \ 307 long s = splimp(); \ 308 if (1 << kup->ku_indx > MAXALLOCSAVE) { \ 309 free((caddr_t)(addr), type); \ 310 } else { \ 311 kbp = &bucket[kup->ku_indx]; \ 312 if (kbp->kb_next == NULL) \ 313 kbp->kb_next = (caddr_t)(addr); \ 314 else \ 315 *(caddr_t*)(kbp->kb_last) = (caddr_t)(addr); \ 316 *(caddr_t*)(addr) = NULL; \ 317 kbp->kb_last = (caddr_t)(addr); \ 318 } \ 319 splx(s); \ 320 } 321 #endif /* do not collect statistics */ The free() function, as defined in kern_malloc.c, is as follows. 248 void</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<pre>249 free(addr, type) 250 void *addr; 251 int type; 252 { 253 register struct kmembuckets *kbp; 254 register struct kmemusage *kup; 255 register struct freelist *freep; 256 long size; 257 int s; 258 #ifdef DIAGNOSTIC 259 caddr_t cp; 260 long *end, *lp, alloc, copysize; 261 #endif 262 #ifdef KMEMSTATS 263 register struct kmemstats *ksp = &kmemstats[type]; 264 #endif 265 266 #ifdef DIAGNOSTIC 267 if ((char *)addr < kmembase (char *)addr >= kmemlimit) { 268 panic("free: address 0x%x out of range", addr); 269 } 270 if ((u_long)type > M_LAST) { 271 panic("free: type %d out of range", type); 272 } 273 #endif 274 kup = btokup(addr); 275 size = 1 << kup->ku_indx; 276 kbp = &bucket[kup->ku_indx]; 277 s = splhigh(); 278 #ifdef DIAGNOSTIC</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>279 /* 280 * Check for returns of data that do not point to the 281 * beginning of the allocation. 282 */ 283 if (size > NBPG * CLSIZE) 284 alloc = addrmask[BUCKETINDX(NBPG * CLSIZE)]; 285 else 286 alloc = addrmask[kup->ku_indx]; 287 if (((u_long)addr & alloc) != 0) 288 panic("free: unaligned addr 0x%x, size %d, type %s, mask %d", 289 addr, size, memname[type], alloc); 290 #endif /* DIAGNOSTIC */ 291 if (size > MAXALLOCSAVE) { 292 kmem_free(kmem_map, (vm_offset_t)addr, ctob(kup- 293 >ku_pagecnt)); 294 #ifdef KMEMSTATS 295 size = kup->ku_pagecnt << PGSHIFT; 296 ksp->ks_memuse -= size; 297 kup->ku_indx = 0; 298 kup->ku_pagecnt = 0; 299 if (ksp->ks_memuse + size >= ksp->ks_limit && 300 ksp->ks_memuse < ksp->ks_limit) 301 wakeup((caddr_t)ksp); 302 ksp->ks_inuse--; 303 kbp->kb_total -= 1; 304 #endif 305 splx(s); 306 return; 307 }</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
<p>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.</p>	<p>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.</p>	<p>FreeBSD includes code that meets the “dynamically determining maximum number for the record search means to remove in the accessed linked list of records” claim limitation.</p> <p>For example, in lines 248-53 of kern_proc.c this first piece of code, the <i>if</i> and <i>for</i> statements dynamically determine whether the maximum number to remove is 0 or 1. If the <i>if</i> statement evaluates TRUE, then the maximum number to remove 1. If the <i>if</i> statement is FALSE and the <i>for</i> loop is not reached the last record, then the maximum number to remove is 1. If the <i>for</i> loop has reached the last record, and the <i>if</i> is FALSE, then it’s 0.</p> <pre> 245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break; 252 } 253 } </pre> <p>Another example of the “dynamically determining” limitation can be found at lines 261-62 of kern_proc.c. The <i>if</i> statement dynamically determines the maximum number to remove. If the <i>if</i> statement evaluates TRUE, then the maximum number to remove is 1; if the <i>if</i> statement evaluates FALSE, then the maximum number to remove is 0.</p> <pre> 258 /* 259 * delete old if empty </pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp);</pre> <p>Further, FreeBSD 2.0.5 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>As both FreeBSD 2.0.5 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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.`</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in FreeBSD 2.0.5 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.</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>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 FreeBSD 2.0.5 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine FreeBSD 2.0.5 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in FreeBSD 2.0.5 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 FreeBSD 2.0.5 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 FreeBSD 2.0.5 with Thatte.</p> <p>Alternatively, it would also be obvious to combine FreeBSD 2.0.5 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:</p> <p>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").</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-7

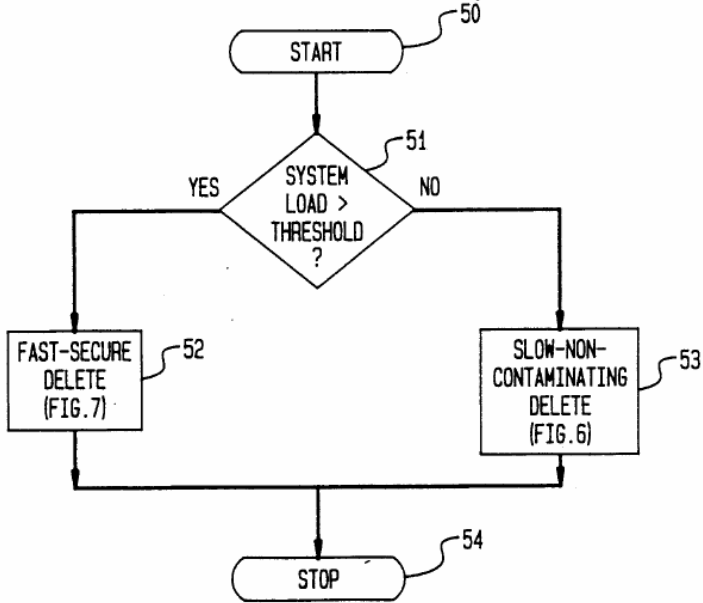
Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5 ¹
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both FreeBSD 2.0.5 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 and would have seen the benefits of doing so. One such benefit, for example, is that the system would avoid</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>performing deletions when the system load exceeded a threshold.</p> <p>Alternatively, it would also be obvious to combine FreeBSD 2.0.5 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>If these heuristics fail and a scavenge is forced instead by the filling of a generation's</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both FreeBSD 2.0.5 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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in FreeBSD 2.0.5 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</p>

EXHIBIT D-7

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">FreeBSD 2.0.5¹</p>
		<p>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 FreeBSD 2.0.5 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 FreeBSD 2.0.5 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.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, FreeBSD 2.0.5 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" and 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," as claimed.</p> <p>For example, in kern_proc.c and proc.h, FreeBSD 2.0.5 discloses a hash table of linked lists of automatically expiring data. <u>See, e.g.</u>, struct pgrp defined at lines 61-70 of proc.h and struct proc defined at lines 72-172 of proc.h. Excerpts are below:</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<pre>61 /* 62 * One structure allocated per process group. 63 */ 64 struct pgrp { 65 struct pgrp *pg_hforw; /* Forward link in hash bucket. */ 66 struct proc *pg_mem; /* Pointer to pgrp members. */ 67 struct session *pg_session; /* Pointer to session. */ 68 pid_t pg_id; /* Pgrp id. */ 69 int pg_jobc; /* # procs qualifying pgrp for job control */ 70 }; 72 /* 73 * Description of a process. 74 * 75 * This structure contains the information needed to manage a thread of 76 * control, known in UN*X as a process; it has references to substructures 77 * containing descriptions of things that the process uses, but may share 78 * with related processes. The process structure and the substructures 79 * are always addressible except for those marked "(PROC ONLY)" below, 80 * which might be addressible only on a processor on which the process 81 * is running. 82 */ 83 struct proc { 84 struct proc *p_forw; /* Doubly-linked run/sleep queue. */ 85 struct proc *p_back; 86 struct proc *p_next; /* Linked list of active procs */ 87 struct proc **p_prev; /* and zombies. */ </pre> <p>FreeBSD discloses a hashing means in connection with a linked list using an external</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>chaining technique to store records with the same hash address. For example, the pidhash[] and pgrp[] structures defined in param.c meet the “hashing means” limitation.</p> <pre>206 struct proc *pidhash[PIDHSZ]; 207 struct pgrp *pgrp[PIDHSZ];</pre> <p>As shown in lines 61-70 and 72-172 of proc.h (portions of which are excerpted above), FreeBSD defines the pgrp structure as including a forward link in the hash bucket as well as a pointer to a linked list of proc structures. This is an example of how FreeBSD meets the “linked list” and “external chaining” limitations.</p> <p>Examples of how FreeBSD uses the hashing technique with external chaining can be found in the enterpgrp() function in kern_proc.c, such as the following:</p> <pre>230 pgrp->pg_hforw = pgrp[pgrp[n = PIDHASH(pgid)]]; 231 pgrp[pgrp[n]] = pgrp;</pre> <p>The function that calls enterpgrp() passes in a proc structure, as shown in lines 175-79.</p> <pre>175 int 176 enterpgrp(p, pgid, mkscs) 177 register struct proc *p; 178 pid_t pgid; 179 int mkscs;</pre> <p>Code within the enterpgrp() structure unlinks the proc from its old process group, as shown below in lines 248-53 of kern_proc.c. Also, enterpgrp() calls pgdelete() if the process group is empty, as shown in lines 261-62. Depending on claim construction,</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<p>these are two examples of automatic expiration.</p> <pre> 245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break; 252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); </pre>
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	<p>FreeBSD discloses “accessing the linked list of records” and “accessing a linked list of records having same hash address,” as claimed. For example, the following code from the enterpgrp() function in kern_proc.c is an example of accessing a linked list of records having the same hash address. The [n] index is an example of a search key. The enterpgrp() function is an example of a “record search means” as claimed.</p> <pre> 230 pgrp->pg_hforw = pgrphash[n = PIDHASH(pgid)]; 231 pgrphash[n] = pgrp; </pre>
[3b] identifying at least some of the automatically expired	[7b] identifying at least some of the automatically expired	<p>FreeBSD includes the step of “identifying at least some of the automatically expired ones of the records,” as claimed. For example, code from enterpgrp() in kern_proc.c</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
ones of the records, and	ones of the records,	<p>discloses these limitations. One such example is the <i>if</i> statement at line 249 which identifies an automatically-expired record. Another example is the <i>if</i> statement at line 261 which identifies an empty record, which is an automatically-expired record.</p> <pre> 245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnext) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnext; 251 break; 252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); </pre>
[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	<p>FreeBSD includes the step of “removing at least some of the automatically expired records from the linked list when the linked list is accessed,” as claimed. For example, code from <code>enterpgrp()</code> in <code>kern_proc.c</code> discloses these limitations. One such example is the call to <code>pgdelete()</code> at line 262. The operation of <code>pgdelete()</code> is discussed in more detail herein at the discussion of elements 1d and 5d, herein.</p> <p>Depending on claim construction, the code at line 250 also meets the “removing” limitation.</p> <pre> 245 /* </pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<pre> 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnext) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnext; 251 break; 252 } 253 } 258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0) 262 pgdelete(p->p_pgrp); </pre>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>FreeBSD includes the step of “inserting, retrieving or deleting one of the records from the system following the step of removing,” as claimed. For example, the code at lines 266-68 of kern_proc.c inserts records into the system, immediately following the call to pgdelete() at line 262, which is an example of FreeBSD code that meets the “deleting” limitation.</p> <pre> 263 /* 264 * link into new one 265 */ 266 p->p_pgrp = pgrp; 267 p->p_pgrpnext = pgrp->pg_mem; 268 pgrp->pg_mem = p; 269 return (0); </pre>
4. The method according to	8. The method according to	FreeBSD includes code that meets the “dynamically determining maximum number

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
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.	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.	<p>for the record search means to remove in the accessed linked list of records” claim limitation.</p> <p>For example, in lines 248-53 of kern_proc.c this first piece of code, the <i>if</i> and <i>for</i> statements dynamically determine whether the maximum number to remove is 0 or 1. If the <i>if</i> statement evaluates TRUE, then the maximum number to remove 1. If the <i>if</i> statement is FALSE and the <i>for</i> loop is not reached the last record, then the maximum number to remove is 1. If the <i>for</i> loop has reached the last record, and the <i>if</i> is FALSE, then it’s 0.</p> <pre>245 /* 246 * unlink p from old process group 247 */ 248 for (pp = &p->p_pgrp->pg_mem; *pp; pp = &(*pp)->p_pgrpnxt) { 249 if (*pp == p) { 250 *pp = p->p_pgrpnxt; 251 break; 252 } 253 }</pre> <p>Another example of the “dynamically determining” limitation can be found at lines 261-62 of kern_proc.c. The <i>if</i> statement dynamically determines the maximum number to remove. If the <i>if</i> statement evaluates TRUE, then the maximum number to remove is 1; if the <i>if</i> statement evaluates FALSE, then the maximum number to remove is 0.</p> <pre>258 /* 259 * delete old if empty 260 */ 261 if (p->p_pgrp->pg_mem == 0)</pre>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>262 <code>pgdelete(p->p_pgrp);</code></p> <p>Further, FreeBSD 2.0.5 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both FreeBSD 2.0.5 and Dirks relate to deletion of aged records upon the</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in FreeBSD 2.0.5 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.</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>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 FreeBSD 2.0.5 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine FreeBSD 2.0.5 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in FreeBSD 2.0.5 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 FreeBSD 2.0.5 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 FreeBSD 2.0.5 with Thatte.</p> <p>Alternatively, it would also be obvious to combine FreeBSD 2.0.5 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:</p> <p style="padding-left: 40px;">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</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5 ¹
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-7

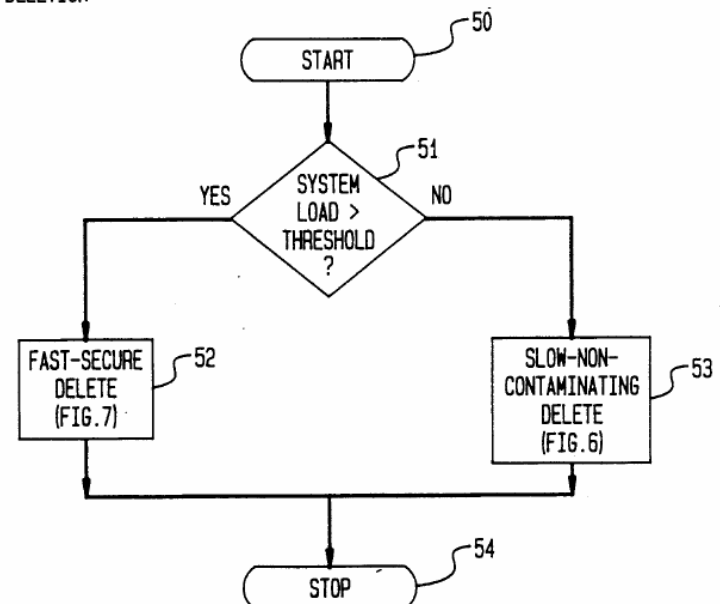
Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both FreeBSD 2.0.5 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 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.</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120	FreeBSD 2.0.5¹
	<p>Alternatively, it would also be obvious to combine FreeBSD 2.0.5 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector at 32.</i></p> <p>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. <i>Opportunistic Garbage Collection at 100.</i></p> <p>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. <i>Id.</i></p> <p>If these heuristics fail and a scavenge is forced instead by the filling of a</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both FreeBSD 2.0.5 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 FreeBSD 2.0.5. 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 FreeBSD 2.0.5 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 FreeBSD 2.0.5 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify</p>

EXHIBIT D-7

Asserted Claims From U.S. Pat. No. 5,893,120		FreeBSD 2.0.5¹
		<p>the system disclosed in FreeBSD 2.0.5 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 FreeBSD 2.0.5 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 FreeBSD 2.0.5 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.</p> <p>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.</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	<p>To the extent the preamble is a limitation, Linux 1.2.13 discloses an “information storage and retrieval system,” as claimed.</p> <p>For example, in arp.c, discloses a hash table of linked lists of automatically expiring data. <u>See, e.g.</u>, struct arp_table defined at lines 79-98.</p>
[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,	<p>Linux 1.2.13 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” and “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.”</p> <p>For example, the arp_table structure is a linked list, as shown in the code below.</p> <pre> 72/* 73 * This structure defines the ARP mapping cache. As long as we make changes 74 * in this structure, we keep interrupts of. But normally we can copy the 75 * hardware address and the device pointer in a local variable and then make 76 * any "long calls" to send a packet out. 77 */ 78 79 struct arp_table 80 { 81 struct arp_table *next; /* Linked entry list */ 82 unsigned long last_used; /* For expiry */ 83 unsigned int flags; /* Control status */ 84 unsigned long ip; /* ip address of entry */ 85 unsigned long mask; /* netmask - used for generalised proxy arps (tridge) */ </pre>

¹ Publicly available as of August 2, 1995; available at <http://www.kernel.org/pub/linux/kernel/v1.2/linux-1.2.13.tar.gz>.

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<pre>86 unsigned char ha[MAX_ADDR_LEN];/* Hardware address */ 87 unsigned char hlen; /* Length of hardware address */ 88 unsigned short htype; /* Type of hardware in use */ 89 struct device *dev; /* Device the entry is tied to */ 90 91 /* 92 * The following entries are only used for unresolved hw addresses. 93 */ 94 95 struct timer_list timer; /* expire timer */ 96 int retries; /* remaining retries */ 97 struct sk_buff_head skb; /* list of queued packets */</pre> <p>The arp_table structure is also used in the context of hashing and external chaining. An example of this is shown in the following code from arp.c.</p> <pre>156/* 157 * The size of the hash table. Must be a power of two. 158 * Maybe we should remove hashing in the future for arp and concentrate 159 * on Patrick Schaaf's Host-Cache-Lookup... 160 */ 161 162 163 #define ARP_TABLE_SIZE 16 164 165 /* The ugly +1 here is to cater for proxy entries. They are put in their 166 own list for efficiency of lookup. If you don't want to find a proxy 167 entry then don't look in the last entry, otherwise do 168 */ 169</pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 170 #define FULL_ARP_TABLE_SIZE (ARP_TABLE_SIZE+1) 171 172 struct arp_table *arp_tables[FULL_ARP_TABLE_SIZE] = 173 { 174 NULL, 175 }; </pre> <p>Also, functions such as arp_expire_request() deals with automatically-expiring records in the linked list.</p> <pre> 367 /* 368 * This function is called, if an entry is not resolved in ARP_RES_TIME. 369 * Either resend a request, or give it up and free the entry. 370 */ 371 372 static void arp_expire_request (unsigned long arg) </pre>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>Linux 1.2.13 discloses “a record search means utilizing a search key to access the linked list” and “a record search means utilizing a search key to access a linked list of records having the same hash address.” For example, the following code from arp_expire_request() in arp.c meets the “record search means” limitation. An example of using a search key to access a linked list of records having the same hash address is the hash value set at line 416 and used as at line 424. As discussed herein, the arp_tables [] structure is a hash table that uses linked lists to perform external chaining.</p> <pre> 409 /* 410 * Arp request timed out. Delete entry and all waiting packets. 411 * If we give each entry a pointer to itself, we don't have to </pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 412 * loop through everything again. Maybe hash is good enough, but 413 * I will look at it later. 414 */ 415 416 hash = HASH(entry->ip); 417 418 /* proxy entries shouldn't really time out so this is really 419 only here for completeness 420 */ 421 if (entry->flags & ATF_PUBL) 422 pentry = &arp_tables[PROXY_HASH]; 423 else 424 pentry = &arp_tables[hash]; </pre>
<p>[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</p>	<p>[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</p>	<p>Linux 1.2.13 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” and “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,” as claimed.</p> <p>For example, in arp_expire_request() in arp.c, the while loop beginning at line 425 accesses the linked list as claimed. The if statement at line 427 identifies an expired record. Depending on claim construction, the “removing” limitation is met at, for example, line 429 and/or 432.</p> <pre> 425 while (*pentry != NULL) 426 { 427 if (*pentry == entry) 428 { 429 *pentry = entry->next; /* delete from linked list */ </pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 430 del_timer(&entry->timer); 431 restore_flags(flags); 432 arp_release_entry(entry); 433 return; 434 } 435 pentry = &(*pentry)->next; 436 }</pre>
<p>[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.</p>	<p>[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.</p>	<p>Linux 1.2.13 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 “ and “meals [<i>sic</i> “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,” as claimed.</p> <p>The “means, utilizing the record search means” limitation is met, for example, by a function that calls arp_expire_request(). As shown in the comments below, other code calls arp_expire_request().</p> <pre> 367/ * 368 * This function is called, if an entry is not resolved in ARP_RES_TIME. 369 * Either resend a request, or give it up and free the entry. 370 */ 371 372 static void arp_expire_request (unsigned long arg)</pre> <p>For example, depending on claim construction, lines 429 and 432 in arp_expire_request() meet the “deleting” and “removing” limitations. An example of the “retrieving” step is line 435. Also, line 435 provides an example of “inserting.” These operations take place within a single while loop and “at the same time,” as claimed.</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 425 while (*pentry != NULL) 426 { 427 if (*pentry == entry) 428 { 429 *pentry = entry->next; /* delete from linked list */ 430 del_timer(&entry->timer); 431 restore_flags(flags); 432 arp_release_entry(entry); 433 return; 434 } 435 pentry = &(*pentry)->next; 436 } </pre>
<p>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.</p>	<p>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.</p>	<p>Linux 1.2.13 includes code that meets the “dynamically determining maximum number for the record search means to remove in the accessed linked list of records” claim limitation.</p> <p>For example, lines each time the if statement at line 427 in arp_expire_request() is executed, it dynamically determines the maximum number of records to remove—it is either 1 or 0. If the if statement evaluates TRUE, then it’s 1; if FALSE, then it’s 0.</p> <pre> 425 while (*pentry != NULL) 426 { 427 if (*pentry == entry) 428 { 429 *pentry = entry->next; /* delete from linked list */ 430 del_timer(&entry->timer); 431 restore_flags(flags); 432 arp_release_entry(entry); </pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<pre>433 return; 434 } 435 pentry = &(*pentry)->next; 436 }</pre> <p>Further, Linux 1.2.13 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>After [a] new VSID has been allocated, the system checks a flag RFLG to</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Linux 1.2.13 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 Linux 1.2.13. 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 Linux 1.2.13 nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.2.13 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Linux 1.2.13 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</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>its entirety.</p> <p>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 Linux 1.2.13 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Linux 1.2.13 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Linux 1.2.13 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 Linux 1.2.13 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 Linux 1.2.13 with Thatte.</p> <p>Alternatively, it would also be obvious to combine Linux 1.2.13 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:</p> <p style="padding-left: 40px;">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-</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-8

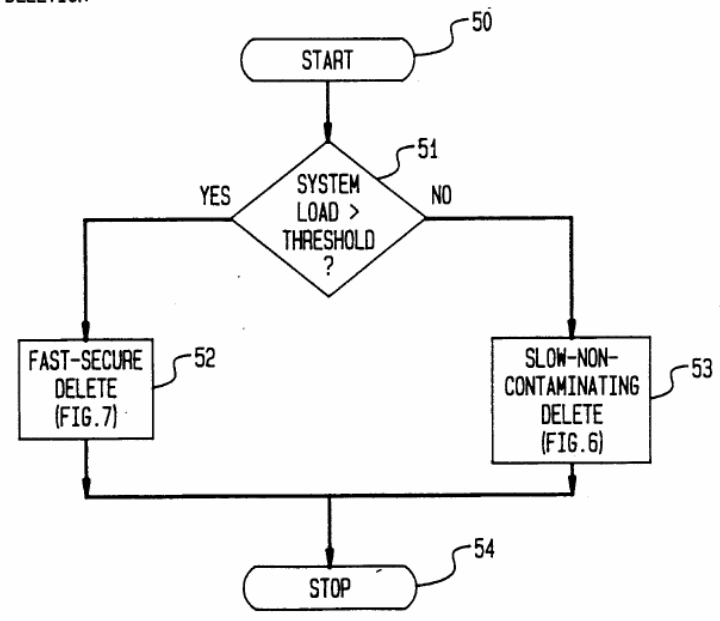
Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Linux 1.2.13 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Linux 1.2.13. 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 Linux 1.2.13 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.2.13 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.</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>Alternatively, it would also be obvious to combine Linux 1.2.13 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector at 32.</i></p> <p>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. <i>Opportunistic Garbage Collection at 100.</i></p> <p>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. <i>Id.</i></p> <p>If these heuristics fail and a scavenge is forced instead by the filling of a</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Linux 1.2.13 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 Linux 1.2.13. 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 Linux 1.2.13 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.2.13 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Linux 1.2.13 to dynamically determine the maximum number of expired records to remove in the accessed linked list of records. It is a</p>

EXHIBIT D-8

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Linux 1.2.13¹</p>
		<p>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 Linux 1.2.13 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 Linux 1.2.13 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.</p> <p>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.</p>
<p>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:</p>	<p>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</p>	<p>To the extent the preamble is a limitation, Linux 1.2.13 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" and 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," as claimed.</p> <p>For example, the <code>arp_table</code> structure defined in <code>arp.c</code> is an example of a linked list</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>expiring, the method comprising the steps of:</p> <pre>72 /* 73 * This structure defines the ARP mapping cache. As long as we make changes 74 * in this structure, we keep interrupts of. But normally we can copy the 75 * hardware address and the device pointer in a local variable and then make 76 * any "long calls" to send a packet out. 77 */ 78 79 struct arp_table 80 { 81 struct arp_table *next; /* Linked entry list */ 82 unsigned long last_used; /* For expiry */ 83 unsigned int flags; /* Control status */ 84 unsigned long ip; /* ip address of entry */ 85 unsigned long mask; /* netmask - used for generalised proxy arps (tridge) */ 86 unsigned char ha[MAX_ADDR_LEN]; /* Hardware address */ 87 unsigned char hlen; /* Length of hardware address */ 88 unsigned short htype; /* Type of hardware in use */ 89 struct device *dev; /* Device the entry is tied to */ 90 91 /* 92 * The following entries are only used for unresolved hw addresses. 93 */ 94 95 struct timer_list timer; /* expire timer */ 96 int retries; /* remaining retries */ 97 struct sk_buff_head skb; /* list of queued packets */ 98 };</pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>The arp_table structure is also used in the context of hashing and external chaining. An example of this is shown in the following code from arp.c.</p> <pre>156 /* 157 * The size of the hash table. Must be a power of two. 158 * Maybe we should remove hashing in the future for arp and concentrate 159 * on Patrick Schaaf's Host-Cache-Lookup... 160 */ 161 162 163 #define ARP_TABLE_SIZE 16 164 165 /* The ugly +1 here is to cater for proxy entries. They are put in their 166 own list for efficiency of lookup. If you don't want to find a proxy 167 entry then don't look in the last entry, otherwise do 168 */ 169 170 #define FULL_ARP_TABLE_SIZE (ARP_TABLE_SIZE+1) 171 172 struct arp_table *arp_tables[FULL_ARP_TABLE_SIZE] = 173 { 174 NULL, 175 };</pre> <p>Also, functions such as arp_check_expire() deal with automatically-expiring records in the linked list. For example, the comments at lines 187-91 discuss records that automatically expire.</p> <pre>186/* 187 * Check if there are too old entries and remove them. If the ATF_PERM</pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<pre>188 * flag is set, they are always left in the arp cache (permanent entry). 189 * Note: Only fully resolved entries, which don't have any packets in 190 * the queue, can be deleted, since ARP_TIMEOUT is much greater than 191 * ARP_MAX_TRIES*ARP_RES_TIME. 192 */ 193 194 static void arp_check_expire(unsigned long dummy) 195 { 196 int i; 197 unsigned long now = jiffies; 198 unsigned long flags; 199 save_flags(flags); 200 cli(); 201 202 for (i = 0; i < FULL_ARP_TABLE_SIZE; i++) 203 { 204 struct arp_table *entry; 205 struct arp_table **pentry = &arp_tables[i]; 206 207 while ((entry = *pentry) != NULL) 208 { 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } 216 else 217 pentry = &entry->next; /* go to next entry */</pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 218 } 219 } 220 restore_flags(flags); 221 222 /* 223 * Set the timer again. 224 */ 225 226 del_timer(&arp_timer); 227 arp_timer.expires = ARP_CHECK_INTERVAL; 228 add_timer(&arp_timer); 229 } </pre>
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	<p>Linux 1.2.13 discloses “accessing the linked list of records” and “accessing a linked list of records having same hash address,” as claimed. For example, as discussed herein, the arp_tables[] structure is a hash table, and each linked list to which it points contains records having the same hash address. For example, the for loop at line 202 iterates through each hash value, and the while loop at line 207 iterates through the linked list associated with each has value. Thus, the while loop accesses the linked list of records having the same hash address, as claimed.</p> <pre> 202 for (i = 0; i < FULL_ARP_TABLE_SIZE; i++) 203 { 204 struct arp_table *entry; 205 struct arp_table **pentry = &arp_tables[i]; 206 207 while ((entry = *pentry) != NULL) 208 { 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) </pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } 216 else 217 pentry = &entry->next; /* go to next entry */ 218 } 219 }</pre>
[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,	<p>Linux 1.2.13 includes the step of “identifying at least some of the automatically expired ones of the records,” as claimed. For example, the if statement at line 209-10 identifies expired records by comparing the last_used element to ARP_TIMEOUT. If last_used is greater than ARP_TIMEOUT, then that entry has expired.</p> <pre> 207 while ((entry = *pentry) != NULL) 208 { 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } 216 else 217 pentry = &entry->next; /* go to next entry */ 218 }</pre>
[3c] removing at least some of the automatically expired	[7c] removing at least some of the automatically expired	<p>Linux 1.2.13 includes the step of “removing at least some of the automatically expired records from the linked list when the linked list is accessed,” as claimed. For</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
records from the linked list when the linked list is accessed.	records from the linked list when the linked list is accessed, and	<p>example, line 212 moves the pointer so that the entry is no longer in the linked list. Also, line 214 calls the <code>kfree_s()</code> function (found in <code>kmalloc.c</code>), which removes the expired element by marking the memory that it occupied as free. Depending on claim construction, at least one of these actions is an example of “removing,” as claimed.</p> <pre> 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } </pre>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>Linux 1.2.13 includes the step of “inserting, retrieving or deleting one of the records from the system following the step of removing,” as claimed. For example, the function <code>arp_check_expire()</code> from <code>arp.c</code> is an example of code from Linux 1.2.13 that meets this element. Note that the code at line 212 moves the pointer so that the element is no longer in the linked list, then at line 214, <code>kfree_s()</code> is called which frees the memory associated with the element.</p> <p>After <code>kfree_s()</code> is called, control passes back to the <i>while</i> loop at line 207 and the next record is retrieved. If that record is NULL, control passes back to the <i>for</i> loop at line 202 and, unless the end of the hash table has been reached, the linked list associated with the next hash entry is retrieved.</p> <p>Thus, this is an example of inserting, retrieving, or deleting one of the records from the system following the step of removing.</p> <pre> 202 for (i = 0; i < FULL_ARP_TABLE_SIZE; i++) 203 { </pre>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13 ¹
		<pre> 204 struct arp_table *entry; 205 struct arp_table **pentry = &arp_tables[i]; 206 207 while ((entry = *pentry) != NULL) 208 { 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } 216 else 217 pentry = &entry->next; /* go to next entry */ 218 } 219 } </pre>
<p>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.</p>	<p>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.</p>	<p>Linux 1.2.13 includes code that meets the “dynamically determining maximum number for the record search means to remove in the accessed linked list of records” claim limitation. For example, the following code from <code>arp_check_expire()</code> in <code>arp.c</code> is an example of dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>This code meets this limitation in at least two ways. First, when the list is first accessed by the <i>while</i> loop beginning at line 207, the maximum number of records to remove is equal to the number of records in the list. But each time the <i>while</i> loop iterates and the <i>if</i> statement evaluates FALSE, that number decreases by one. Hence, it is dynamic.</p> <p>Second, each time the <i>if</i> statement at line 209-10 is called, the maximum number of</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>records to remove is either 1 or 0, depending on whether the <i>if</i> statement evaluates to TRUE or FALSE. If the if statement evaluates TRUE, then the maximum number to remove is 1; if the if statement evaluates FALSE, the maximum number to remove is 0.</p> <pre>207 while ((entry = *pentry) != NULL) 208 { 209 if ((now - entry->last_used) > ARP_TIMEOUT 210 && !(entry->flags & ATF_PERM)) 211 { 212 *pentry = entry->next; /* remove from list */ 213 del_timer(&entry->timer); /* Paranoia */ 214 kfree_s(entry, sizeof(struct arp_table)); 215 } 216 } else 217 pentry = &entry->next; /* go to next entry */ 218 }</pre> <p>Further, Linux 1.2.13 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.</p> <p>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.</p> <p>As summarized in Dirks,</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Linux 1.2.13 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 Linux 1.2.13. 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 Linux 1.2.13 would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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 Linux 1.2.13 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Linux 1.2.13 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.</p> <p>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 Linux 1.2.13 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Linux 1.2.13 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Linux 1.2.13 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 Linux 1.2.13 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</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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 Linux 1.2.13 with Thatte.</p> <p>Alternatively, it would also be obvious to combine Linux 1.2.13 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:</p> <p style="padding-left: 40px;">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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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</p>

EXHIBIT D-8

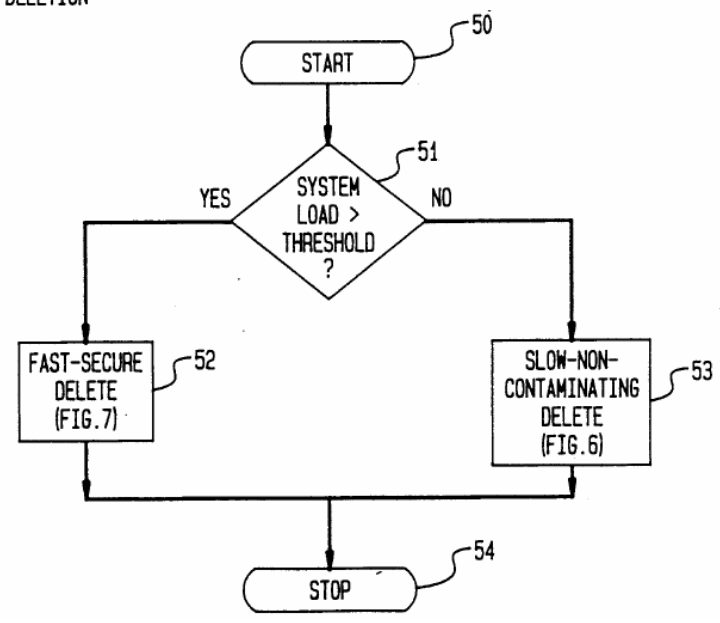
Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>used when the load on the system is at lower levels. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>than the threshold, then a fast-secure delete 52 is used. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Linux 1.2.13 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Linux 1.2.13. 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 Linux 1.2.13 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.2.13¹
	<p>load as taught by the '663 patent and with Linux 1.2.13 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.</p> <p>Alternatively, it would also be obvious to combine Linux 1.2.13 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Linux 1.2.13 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 Linux 1.2.13. 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 Linux 1.2.13 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.2.13 and would have seen the benefits of doing so. One such benefit, for example, is that the system would only perform</p>

EXHIBIT D-8

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.2.13¹
		<p>deletions when the system was not already too overloaded, thus preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Linux 1.2.13 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 Linux 1.2.13 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 Linux 1.2.13 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.</p> <p>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.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
1. An information storage and retrieval system, the system comprising:	5. An information storage and retrieval system, the system comprising:	<p>To the extent the preamble is limiting, Linux 1.3.51 discloses an “information storage and retrieval system,” as claimed.</p> <p>For example, route.c in Linux 1.3.51 includes fib_node and fib_zone structures that are used to provide hashing with external chaining using one or more linked lists. These structures are defined at lines 77-85, 104-112, and 114-117.</p> <pre>73 /* 74 * Forwarding Information Base definitions. 75 */ 76 77 struct fib_node 78 { 79 struct fib_node *fib_next; 80 __u32 fib_dst; 81 unsigned long fib_use; 82 struct fib_info *fib_info; 83 short fib_metric; 84 unsigned char fib_tos; 85 }; 86 87 /* 88 * This structure contains data shared by many of routes. 89 */ 90 91 struct fib_info 92 { 93 struct fib_info *fib_next;</pre>

¹ Publicly available as of December 27, 1995; available at <http://www.kernel.org/pub/linux/kernel/v1.3/linux-1.3.51.tar.gz>.
Plaintiff's Invalidation Contentions & Production of Documents

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 94 struct fib_info *fib_prev; 95 __u32 fib_gateway; 96 struct device *fib_dev; 97 int fib_refcnt; 98 unsigned long fib_window; 99 unsigned short fib_flags; 100 unsigned short fib_mtu; 101 unsigned short fib_irtt; 102 }; 103 104 struct fib_zone 105 { 106 struct fib_zone *fz_next; 107 struct fib_node **fz_hash_table; 108 struct fib_node *fz_list; 109 int fz_nent; 110 int fz_logmask; 111 __u32 fz_mask; 112 }; 113 114 static struct fib_zone *fib_zones[33]; 115 static struct fib_zone *fib_zone_list; 116 static struct fib_node *fib_loopback = NULL; 117 static struct fib_info *fib_info_list; </pre>
<p>[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</p>	<p>[5a] a hashing means to provide access to records stored in a memory of the system and using an external chaining technique to store</p>	<p>Linux 1.3.51 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” and “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 fib_node</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
expiring,	the records with same hash address, at least some of the records automatically expiring,	<p>structure defined at lines 77-85 of route.c includes a pointer to the next fib_node structure in a linked list (see line 79), which is used in the context of hashing with external chaining.</p> <p>The fib_zone structure can contain a pointer to a hash table (see line 107), which uses external chaining.</p> <p>An example of how these structures operate can be seen in the fib_add_1() function in route.c, which creates a hash table. The fz variable (e.g., at line 624) represents a fib_zone structure, which, as described above, includes a hash table, which is a pointer to a pointer to a fib_node element. The fib_node structure is a linked list, as shown by the fact that each element contains a pointer to the next element in the list (i.e., fib_next).</p> <pre>620 /* 621 * If zone overgrows RTZ_HASHING_LIMIT, create hash table. 622 */ 623 624 if (fz->fz_nent >= RTZ_HASHING_LIMIT && !fz->fz_hash_table && logmask<32) 625 { 626 struct fib_node ** ht; 627 #if RT_CACHE_DEBUG 628 printk("fib_add_1: hashing for zone %d started\n", logmask); 629 #endif 630 ht = kmalloc(RTZ_HASH_DIVISOR*sizeof(struct rtable*), GFP_KERNEL); 631 632 if (ht) 633 { 634 memset(ht, 0, RTZ_HASH_DIVISOR*sizeof(struct fib_node*));</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>635 cli(); 636 f1 = fz->fz_list; 637 while (f1) 638 { 639 struct fib_node * next; 640 unsigned hash = fz_hash_code(f1->fib_dst, logmask); 641 next = f1->fib_next; 642 f1->fib_next = ht[hash]; 643 ht[hash] = f1; 644 f1 = next; 645 } 646 fz->fz_list = NULL; 647 fz->fz_hash_table = ht; 648 sti(); 649 } 650 } 651 652 if (fz->fz_hash_table) 653 fp = &fz->fz_hash_table[fz_hash_code(dst, logmask)]; 654 else 655 fp = &fz->fz_list; 656 657 /* 658 * Scan list to find the first route with the same destination 659 */ 660 while ((f1 = *fp) != NULL) 661 { 662 if (f1->fib_dst == dst) 663 break; 664 fp = &f1->fib_next;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 665 } 666 An example of code that meets the “automatically expiring” limitation can be found at lines 670-82. For example, a route with the same destination and less than or equal metric value has automatically expired, and, according to the comment at lines 675- 77, is purged. 667 /* 668 * Find route with the same destination and less (or equal) metric. 669 */ 670 while ((f1 = *fp) != NULL && f1->fib_dst == dst) 671 { 672 if (f1->fib_metric >= metric) 673 break; 674 /* 675 * Record route with the same destination and gateway, 676 * but less metric. We'll delete it 677 * after instantiation of new route. 678 */ 679 if (f1->fib_info->fib_gateway == gw) 680 dup_fp = fp; 681 fp = &f1->fib_next; 682 } </pre>
[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,	<p>Linux 1.3.51 discloses “a record search means utilizing a search key to access the linked list” and “a record search means utilizing a search key to access a linked list of records having the same hash address.”</p> <p>For example, the fib_add_1() function is an example of a record search means, as</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<p>claimed. An example of how this uses a “search key” can be found at lines 652-53. This code uses a hash value to find the address of the first element of a linked list associated with a particular hash value.</p> <pre>652 if (fz->fz_hash_table) 653 fp = &fz->fz_hash_table[fz_hash_code(dst, logmask)];</pre>
<p>[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</p>	<p>[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</p>	<p>Linux 1.3.51 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” and “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,” as claimed.</p> <p>For example, the code at lines 670-82 of route.c identifies a record in the linked list corresponding to a route with the same destination and less or equal metric. Thus, it accesses the linked list and identifies automatically expiring records.</p> <pre>667 /* 668 * Find route with the same destination and less (or equal) metric. 669 */ 670 while ((f1 = *fp) != NULL && f1->fib_dst == dst) 671 { 672 if (f1->fib_metric >= metric) 673 break; 674 /* 675 * Record route with the same destination and gateway, 676 * but less metric. We'll delete it 677 * after instantiation of new route. 678 */ 679 if (f1->fib_info->fib_gateway == gw)</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>680 dup_fp = fp; 681 fp = &f1->fib_next; 682 }</pre> <p>Code within Linux 1.3.51 also performs the “removing” step when the list is accessed. An example of this can be found at lines 707-732 of route.c. For example, the <i>if</i> statement at line 718 identifies expired records, and if an expired record is found then line 721 moves the pointer so that record is no longer in the list. Depending on claim construction, this is the “removing” step. Also, the call to fib_free_node() at line 727 frees the memory used by the record. Depending on claim construction, this is the “removing” step. Both steps “identifying and removing” are performed within the <i>while</i> loop that starts at line 716 which accesses the list.</p> <pre>707 /* 708 * Delete route with the same destination and gateway. 709 * Note that we should have at most one such route. 710 */ 711 if (dup_fp) 712 fp = dup_fp; 713 else 714 fp = &f->fib_next; 715 716 while ((f1 = *fp) != NULL && f1->fib_dst == dst) 717 { 718 if (f1->fib_info->fib_gateway == gw) 719 { 720 cli(); 721 *fp = f1->fib_next; 722 if (fib_loopback == f1) 723 fib_loopback = NULL;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre>724 sti(); 725 ip_netlink_msg(RTMSG_DELROUTE, dst, gw, mask, flags, 726 metric, f1->fib_info->fib_dev->name); 727 fib_free_node(f1); 728 fz->fz_nent--; 729 break; 730 } 731 fp = &f1->fib_next; 732 }</pre> <p>The fib_free_node() function is found at lines 185-203 in route.c. As shown below, fib_free_node() moves pointers (lines 193-98) and calls kfree_s() to mark the memory as available (line 200).</p> <pre>181 /* 182 * Free FIB node. 183 */ 184 185 static void fib_free_node(struct fib_node * f) 186 { 187 struct fib_info * fi = f->fib_info; 188 if (!--fi->fib_refcnt) 189 { 190 #if RT_CACHE_DEBUG >= 2 191 printk("fib_free_node: fi %08x/%s is free\n", fi- 192 >fib_gateway, fi->fib_dev->name); 193 #endif 194 if (fi->fib_next) 195 fi->fib_next->fib_prev = fi->fib_prev; 196 if (fi->fib_prev)</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre>196 fi->fib_prev->fib_next = fi->fib_next; 197 if (fi == fib_info_list) 198 fib_info_list = fi->fib_next; 199 } 200 kfree_s(f, sizeof(struct fib_node)); 201 }</pre> <p>The malloc.h file in Linux 1.3.51 defines kfree_s() as follows:</p> <pre>9 #define kfree_s(a,b) kfree(a)</pre> <p>The kmalloc.c file in Linux 1.3.51 defines kfree() as follows:</p> <pre>276 void kfree(void *ptr) 277 { 278 int size; 279 unsigned long flags; 280 int order; 281 register struct block_header *p; 282 struct page_descriptor *page, **pg; 283 284 if (!ptr) 285 return; 286 p = ((struct block_header *) ptr) - 1; 287 page = PAGE_DESC(p); 288 order = page->order; 289 pg = &sizes[order].firstfree; 290 if (p->bh_flags == MF_DMA) { 291 p->bh_flags = MF_USED; 292 pg = &sizes[order].dmafree;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<pre>293 } 294 295 if ((order < 0) 296 (order >= sizeof(sizes) / sizeof(sizes[0])) 297 (((long) (page->next)) & ~PAGE_MASK) 298 (p->bh_flags != MF_USED)) { 299 printk("kfree of non-kmallocated memory: %p, next= %p, order=%d\n", 300 p, page->next, page->order); 301 return; 302 } 303 size = p->bh_length; 304 p->bh_flags = MF_FREE; /* As of now this block is officially free */ 305 save_flags(flags); 306 cli(); 307 p->bh_next = page->firstfree; 308 page->firstfree = p; 309 page->nfree++; 310 311 if (page->nfree == 1) { 312 /* Page went from full to one free block: put it on the freelist. */ 313 page->next = *pg; 314 *pg = page; 315 } 316 /* If page is completely free, free it */ 317 if (page->nfree == NBLOCKS(order)) { 318 for (;;) { 319 struct page_descriptor *tmp = *pg; 320 if (!tmp) { 321 printk("Oops. page %p doesn't show on freelist.\n",</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> page); 322 break; 323 } 324 if (tmp == page) { 325 *pg = page->next; 326 break; 327 } 328 pg = &tmp->next; 329 } 330 sizes[order].npages--; 331 free_pages((long) page, sizes[order].gfporder); 332 } 333 sizes[order].nfrees++; 334 sizes[order].nbytesmallored -= size; 335 restore_flags(flags); 336 } </pre>
<p>[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.</p>	<p>[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.</p>	<p>Linux 1.3.51 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 “ and “meals [<i>sic</i> “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,” as claimed.</p> <p>For example, functions such as <code>rt_add()</code> call <code>fib_add_1()</code>—e.g., at line 1310 of <code>route.c</code> below. Such functions are examples of “means utilizing the record search means,” as claimed.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>1303 static void rt_add(short flags, __u32 dst, __u32 mask, 1304 __u32 gw, struct device *dev, unsigned short mss, 1305 unsigned long window, unsigned short irtt, short metric) 1306 { 1307 while (ip_rt_lock) 1308 sleep_on(&rt_wait); 1309 ip_rt_fast_lock(); 1310 fib_add_1(flags, dst, mask, gw, dev, mss, window, irtt, metric); 1311 ip_rt_unlock(); 1312 wake_up(&rt_wait); 1313 }</pre> <p>The fib_add_1() function is an example of code that meets the “inserting” limitations. For example, see the code below from route.c.</p> <pre>694 /* 695 * Insert new entry to the list. 696 */ 697 698 cli(); 699 f->fib_next = f1; 700 *fp = f; 701 if (!fib_loopback && (fi->fib_dev->flags & IFF_LOOPBACK)) 702 fib_loopback = f; 703 sti(); 704 fz->fz_nent++; 705 ip_netlink_msg(RTMSG_NEWROUTE, dst, gw, mask, flags, metric, fi- ->fib_dev->name);</pre> <p>The fib_add_1() function also is an example of code that meets the “accessing,”</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>“retrieving” and “deleting” limitations at the same time as “removing.” The <i>while</i> loop beginning at line 716 is an example of code that meets the “retrieving” and “accessing” claim limitations. In order to iterate through the linked list, the <i>while</i> loop must retrieve and access each element of the list.</p> <p>Examples of code meeting the “removing” and “deleting” limitations can be found at lines 721 and 727. Line 721 moves the pointer to the next element. Depending on claim construction, this is the “removing” step. Alternatively, depending on claim construction, this is the “deleting” step, and the call to <code>fib_free_node()</code> is the “removing” step. The <i>if</i> statement at line 718 is an example of identifying expired records.</p> <p>Both of these steps (identifying and removing) take place when the list is accessed. For example, the <i>while</i> loops above iterate through the elements of the list in order to test each element to determine whether it should be removed. In order to identify the elements, the list must be accessed</p> <pre>707 /* 708 * Delete route with the same destination and gateway. 709 * Note that we should have at most one such route. 710 */ 711 if (dup_fp) 712 fp = dup_fp; 713 else 714 fp = &f->fib_next; 715 716 while ((f1 = *fp) != NULL && f1->fib_dst == dst) 717 { 718 if (f1->fib_info->fib_gateway == gw)</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 719 { 720 cli(); 721 *fp = f1->fib_next; 722 if (fib_loopback == f1) 723 fib_loopback = NULL; 724 sti(); 725 ip_netlink_msg(RTMSG_DELRROUTE, dst, gw, mask, flags, 726 metric, f1->fib_info->fib_dev->name); 727 fib_free_node(f1); 728 fz->fz_nent--; 729 break; 730 } 731 fp = &f1->fib_next; 732 } </pre>
<p>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.</p>	<p>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.</p>	<p>Linux 1.3.51 includes code that meets the “dynamically determining maximum number for the record search means to remove in the accessed linked list of records” claim limitation.</p> <p>For example, code in the fib_add_1() function in route.c determines the maximum number of records to remove. As the comment at lines 708-09 states, there should be no more than one route removed. Thus, the maximum number to remove is either 0 or 1. The <i>if</i> statement at line 718 dynamically determines whether that number is 0 or 1.</p> <pre> 707 /* 708 * Delete route with the same destination and gateway. 709 * Note that we should have at most one such route. 710 */ 711 if (dup_fp) 712 fp = dup_fp; 713 else </pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<pre>714 fp = &f->fib_next; 715 716 while ((f1 = *fp) != NULL && f1->fib_dst == dst) 717 { 718 if (f1->fib_info->fib_gateway == gw) 719 { 720 cli(); 721 *fp = f1->fib_next; 722 if (fib_loopback == f1) 723 fib_loopback = NULL; 724 sti(); 725 ip_netlink_msg(RTMSG_DELROUTE, dst, gw, mask, flags, 726 metric, f1->fib_info->fib_dev->name); 727 fib_free_node(f1); 728 fz->fz_nent--; 729 break; 730 } 731 fp = &f1->fib_next; 732 }</pre> <p>Further, Linux 1.3.51 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.</p> <p>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.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>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.</p> <p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Linux 1.3.51 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 Linux 1.3.51. 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 Linux 1.3.51 nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>each step of the sweeping process with Linux 1.3.51 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.`</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Linux 1.3.51 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.</p> <p>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 Linux 1.3.51 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Linux 1.3.51 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in Linux 1.3.51 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 Linux 1.3.51 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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>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 Linux 1.3.51 with Thatte.</p> <p>Alternatively, it would also be obvious to combine Linux 1.3.51 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:</p> <p>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").</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-9

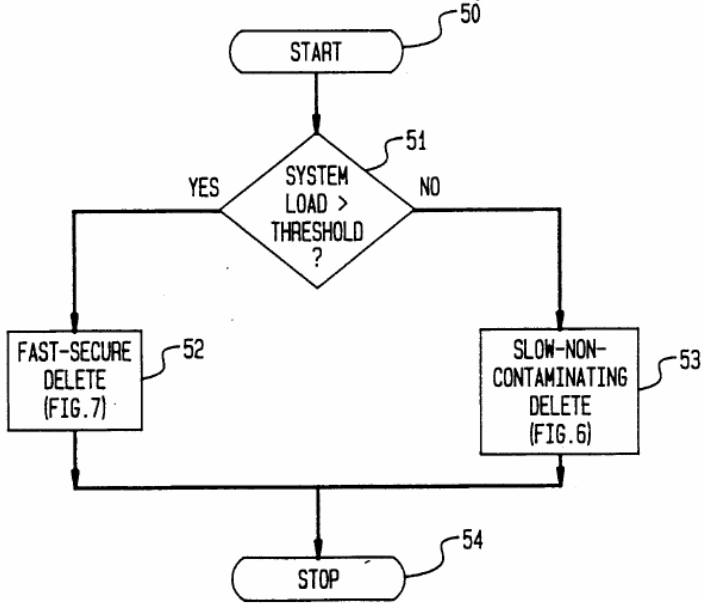
Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Linux 1.3.51 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Linux 1.3.51. 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 Linux 1.3.51 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.3.51 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.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>Alternatively, it would also be obvious to combine Linux 1.3.51 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Linux 1.3.51 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 Linux 1.3.51. 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 Linux 1.3.51 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.3.51 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Linux 1.3.51 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</p>

EXHIBIT D-9

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Linux 1.3.51¹</p>
		<p>information available to the system. One of ordinary skill in the art would have been motivated to combine the system disclosed in Linux 1.3.51 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 Linux 1.3.51 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.</p> <p>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.</p>
<p>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:</p>	<p>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:</p>	<p>To the extent the preamble is a limitation, Linux 1.3.51 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” and 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,” as claimed.</p> <p>For example, route.c in Linux 1.3.51 includes fib_node and fib_zone structures that are used to provide hashing with external chaining using one or more linked lists. These structures are defined at lines 77-85, 104-112, and 114-117.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre>73 /* 74 * Forwarding Information Base definitions. 75 */ 76 77 struct fib_node 78 { 79 struct fib_node *fib_next; 80 __u32 fib_dst; 81 unsigned long fib_use; 82 struct fib_info *fib_info; 83 short fib_metric; 84 unsigned char fib_tos; 85 }; 86 87 /* 88 * This structure contains data shared by many of routes. 89 */ 90 91 struct fib_info 92 { 93 struct fib_info *fib_next; 94 struct fib_info *fib_prev; 95 __u32 fib_gateway; 96 struct device *fib_dev; 97 int fib_refcnt; 98 unsigned long fib_window; 99 unsigned short fib_flags; 100 unsigned short fib_mtu; 101 unsigned short fib_irtt;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>102 }; 103 104 struct fib_zone 105 { 106 struct fib_zone *fz_next; 107 struct fib_node **fz_hash_table; 108 struct fib_node *fz_list; 109 int fz_nent; 110 int fz_logmask; 111 __u32 fz_mask; 112 }; 113 114 static struct fib_zone *fib_zones[33]; 115 static struct fib_zone *fib_zone_list; 116 static struct fib_node *fib_loopback = NULL; 117 static struct fib_info *fib_info_list;</pre> <p>An example of code disclosing a hashing technique as claimed can be found in fib_del_1() in route.c, such as at lines 415 and 429. As shown in the discussion of the fib_node and fib_zone structures, this hashing technique uses external chaining, wherein a linked list is associated with elements having the same hash value.</p> <pre>409 if (!mask) 410 { 411 for (fz=fib_zone_list; fz; fz = fz->fz_next) 412 { 413 int tmp; 414 if (fz->fz_hash_table) 415 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- >fz_logmask)];</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>416 else 417 fp = &fz->fz_list; 418 419 tmp = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 420 fz->fz_nent -= tmp; 421 found += tmp; 422 } 423 } 424 else 425 { 426 if ((fz = fib_zones[rt_logmask(mask)]) != NULL) 427 { 428 if (fz->fz_hash_table) 429 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- 430 >fz_logmask)]; 431 else 432 fp = &fz->fz_list; 433 434 found = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 435 fz->fz_nent -= found; 436 } </pre> <p>Linux 1.3.51 also includes examples of automatically expiring records. For example, as shown in the comments at line 1286, <code>rt_del()</code> is only called by user processes. The condition that triggers the user process to call <code>rt_del</code> is an external condition. Note that in line 1297, <code>rt_del()</code> calls <code>fib_del_1()</code> to delete the expired records passed to it by the user program that called <code>rt_del()</code>.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 1286 * rt_{del add flush} called only from USER process. Waiting is OK. 1287 */ 1288 1289 static int rt_del(__u32 dst, __u32 mask, 1290 struct device * dev, __u32 gtw, short rt_flags, short metric) 1291 { 1292 int retval; 1293 1294 while (ip_rt_lock) 1295 sleep_on(&rt_wait); 1296 ip_rt_fast_lock(); 1297 retval = fib_del_1(dst, mask, dev, gtw, rt_flags, metric); 1298 ip_rt_unlock(); 1299 wake_up(&rt_wait); 1300 return retval; 1301 } </pre>
[3a] accessing the linked list of records,	[7a] accessing a linked list of records having same hash address,	<p>Linux 1.3.51 discloses “accessing the linked list of records” and “accessing a linked list of records having same hash address,” as claimed. For example, line 415 and 429 each identify the location of a linked list of records pointed to by the hash table by calling the fz_hash_code() function.</p> <pre> 409 if (!mask) 410 { 411 for (fz=fib_zone_list; fz; fz = fz->fz_next) 412 { 413 int tmp; </pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 414 if (fz->fz_hash_table) 415 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- >fz_logmask)]; 416 else 417 fp = &fz->fz_list; 418 419 tmp = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 420 fz->fz_nent -= tmp; 421 found += tmp; 422 } 423 } 424 else 425 { 426 if ((fz = fib_zones[rt_logmask(mask)]) != NULL) 427 { 428 if (fz->fz_hash_table) 429 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- >fz_logmask)]; 430 else 431 fp = &fz->fz_list; 432 433 found = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 434 fz->fz_nent -= found; 435 } 436 } </pre>
[3b] identifying at least some of the automatically expired	[7b] identifying at least some of the automatically expired	Linux 1.3.51 includes the step of “identifying at least some of the automatically expired ones of the records,” as claimed. For example, fib_del_1() function calls

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
ones of the records, and	ones of the records,	<pre>fib_del_list() at lines 419 and 433 to perform the identifying, removing, retrieving, and deleting functions. 409 if (!mask) 410 { 411 for (fz=fib_zone_list; fz; fz = fz->fz_next) 412 { 413 int tmp; 414 if (fz->fz_hash_table) 415 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- >fz_logmask)]; 416 else 417 fp = &fz->fz_list; 418 419 tmp = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 420 fz->fz_nent -= tmp; 421 found += tmp; 422 } 423 } 424 else 425 { 426 if ((fz = fib_zones[rt_logmask(mask)]) != NULL) 427 { 428 if (fz->fz_hash_table) 429 fp = &fz->fz_hash_table[fz_hash_code(dst, fz- >fz_logmask)]; 430 else 431 fp = &fz->fz_list;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>432 433 found = fib_del_list(fp, dst, dev, gtw, flags, metric, mask); 434 fz->fz_nent -= found; 435 } 436 }</pre> <p>The fib_del_list() function is also found in route.c in Linux 1.3.51. The <i>while</i> loop beginning at line 373 in fib_del_list() iterates through the linked list and identifies the elements. The elements are “automatically expired” because, for example, the user program that called rt_del() determined that the elements needed to be removed. Then, rt_del() called fib_del_1(), which in turn called fib_del_list() to remove the automatically expired records.</p> <pre>367 static int fib_del_list(struct fib_node **fp, __u32 dst, 368 struct device * dev, __u32 gtw, short flags, short metric, __u32 mask) 369 { 370 struct fib_node *f; 371 int found=0; 372 373 while((f = *fp) != NULL) 374 { 375 struct fib_info * fi = f->fib_info; 376 377 /* 378 * Make sure the destination and netmask match. 379 * metric, gateway and device are also checked 380 * if they were specified. 381 */</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 382 if (f->fib_dst != dst 383 (gtw && fi->fib_gateway != gtw) 384 (metric >= 0 && f->fib_metric != metric) 385 (dev && fi->fib_dev != dev)) 386 { 387 fp = &f->fib_next; 388 continue; 389 } 390 cli(); 391 *fp = f->fib_next; 392 if (fib_loopback == f) 393 fib_loopback = NULL; 394 sti(); 395 ip_netlink_msg(RTMSG_DELRROUTE, dst, gtw, mask, flags, metric, 396 fi->fib_dev->name); 397 fib_free_node(f); 398 found++; 399 } 400 } </pre>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>Linux 1.3.51 includes the step of “removing at least some of the automatically expired records from the linked list when the linked list is accessed,” as claimed. For example, the <i>while</i> loop beginning at line 373 access the linked list. The <i>if</i> statement at lines 382-89 causes the loop to move to the next element if there is no match. If there is a match, lines 391 and/or 396 meet the “removing” limitation, depending on claim construction. The “removing” takes place when the linked list is accessed. For example, the commands are executed within a while loop in the <code>fib_del_list()</code> function.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<pre>367 static int fib_del_list(struct fib_node **fp, __u32 dst, 368 struct device * dev, __u32 gtw, short flags, short metric, __u32 mask) 369 { 370 struct fib_node *f; 371 int found=0; 372 373 while((f = *fp) != NULL) 374 { 375 struct fib_info * fi = f->fib_info; 376 377 /* 378 * Make sure the destination and netmask match. 379 * metric, gateway and device are also checked 380 * if they were specified. 381 */ 382 if (f->fib_dst != dst 383 (gtw && fi->fib_gateway != gtw) 384 (metric >= 0 && f->fib_metric != metric) 385 (dev && fi->fib_dev != dev)) 386 { 387 fp = &f->fib_next; 388 continue; 389 } 390 cli(); 391 *fp = f->fib_next; 392 if (fib_loopback == f) 393 fib_loopback = NULL;</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre>394 sti(); 395 ip_netlink_msg(RTMSG_DELROUTE, dst, gtw, mask, flags, metric, fi->fib_dev->name); 396 fib_free_node(f); 397 found++; 398 } 399 return found; 400 }</pre>
	[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.	<p>Linux 1.3.51 includes the step of “inserting, retrieving or deleting one of the records from the system following the step of removing,” as claimed.</p> <p>For example, depending on claim construction, the code at line 391 meets the “removing” limitation, and the code at 396 meets the “deleting” limitation. And because line 396 follows line 391, the deleting takes place “following the step of removing” as claimed.</p> <p>As another example, depending on claim construction, line 396 constitutes “removing,” then when control passes back to the <i>while</i> loop at line 373, that code performs the “retrieving” step. Because this takes place after line 396 is executed, the “retrieving” step takes place “following the step of removing,” as claimed.</p> <pre>367 static int fib_del_list(struct fib_node **fp, __u32 dst, 368 struct device * dev, __u32 gtw, short flags, short metric, __u32 mask) 369 { 370 struct fib_node *f; 371 int found=0; 372 373 while((f = *fp) != NULL)</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<pre>374 { 375 struct fib_info * fi = f->fib_info; 376 377 /* 378 * Make sure the destination and netmask match. 379 * metric, gateway and device are also checked 380 * if they were specified. 381 */ 382 if (f->fib_dst != dst 383 (gtw && fi->fib_gateway != gtw) 384 (metric >= 0 && f->fib_metric != metric) 385 (dev && fi->fib_dev != dev)) 386 { 387 fp = &f->fib_next; 388 continue; 389 } 390 cli(); 391 *fp = f->fib_next; 392 if (fib_loopback == f) 393 fib_loopback = NULL; 394 sti(); 395 ip_netlink_msg(RTMSG_DELRROUTE, dst, gtw, mask, flags, metric, 396 fi->fib_dev->name); 397 fib_free_node(f); 398 found++; 399 } 400 return found; 401 }</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>The fib_free_node() function called by fib_del_list() is found at lines 185-203 in route.c. As shown below, fib_free_node() moves pointers (lines 193-98) and calls kfree_s() to mark the memory as available (line 200). For example, depending on claim construction, code in fib_free_node() meets the “removing” limitation and the code in kfree() meets the “deleting” limitation.</p> <pre>181 /* 182 * Free FIB node. 183 */ 184 185 static void fib_free_node(struct fib_node * f) 186 { 187 struct fib_info * fi = f->fib_info; 188 if (!--fi->fib_refcnt) 189 { 190 #if RT_CACHE_DEBUG >= 2 191 printk("fib_free_node: fi %08x/%s is free\n", fi- 192 >fib_gateway, fi->fib_dev->name); 193 #endif 194 if (fi->fib_next) 195 fi->fib_next->fib_prev = fi->fib_prev; 196 if (fi->fib_prev) 197 fi->fib_prev->fib_next = fi->fib_next; 198 if (fi == fib_info_list) 199 fib_info_list = fi->fib_next; 200 } 201 kfree_s(f, sizeof(struct fib_node)); 202 }</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>The malloc.h file in Linux 1.3.51 defines kfree_s() as follows:</p> <pre>9 #define kfree_s(a,b) kfree(a)</pre> <p>The kmalloc.c file in Linux 1.3.51 defines kfree() as follows:</p> <pre>276 void kfree(void *ptr) 277 { 278 int size; 279 unsigned long flags; 280 int order; 281 register struct block_header *p; 282 struct page_descriptor *page, **pg; 283 284 if (!ptr) 285 return; 286 p = ((struct block_header *) ptr) - 1; 287 page = PAGE_DESC(p); 288 order = page->order; 289 pg = &sizes[order].firstfree; 290 if (p->bh_flags == MF_DMA) { 291 p->bh_flags = MF_USED; 292 pg = &sizes[order].dmafree; 293 } 294 295 if ((order < 0) 296 (order >= sizeof(sizes) / sizeof(sizes[0])) 297 (((long) (page->next)) & ~PAGE_MASK) 298 (p->bh_flags != MF_USED)) { 299 printk("kfree of non-kmalloced memory: %p, next= %p,</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<pre>order=%d\n", 300 p, page->next, page->order); 301 return; 302 } 303 size = p->bh_length; 304 p->bh_flags = MF_FREE; /* As of now this block is officially free */ 305 save_flags(flags); 306 cli(); 307 p->bh_next = page->firstfree; 308 page->firstfree = p; 309 page->nfree++; 310 311 if (page->nfree == 1) { 312 /* Page went from full to one free block: put it on the freelist. */ 313 page->next = *pg; 314 *pg = page; 315 } 316 /* If page is completely free, free it */ 317 if (page->nfree == NBLOCKS(order)) { 318 for (;;) { 319 struct page_descriptor *tmp = *pg; 320 if (!tmp) { 321 printk("Oops. page %p doesn't show on freelist.\n", page); 322 break; 323 } 324 if (tmp == page) { 325 *pg = page->next; 326 break; 327 } </pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre> 328 pg = &tmp->next; 329 } 330 sizes[order].npages--; 331 free_pages((long) page, sizes[order].gfporder); 332 } 333 sizes[order].nfrees++; 334 sizes[order].nbytesmallocated -= size; 335 restore_flags(flags); 336 } </pre>
<p>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.</p>	<p>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.</p>	<p>Linux 1.3.51 includes code that meets the “dynamically determining maximum number for the record search means to remove in the accessed linked list of records” claim limitation.</p> <p>For example, the <i>while</i> loop beginning at line 373 of route.c iterates through the linked list passed in as <i>**fp</i> at line 367. Thus, when the <i>fib_del_list()</i> function is called, the maximum number of expired records to remove is the length of the <i>**fp</i> linked list. This number is dynamic because, if the <i>if</i> statement beginning at line 382 evaluates TRUE for an element, the maximum number to delete decreases by one.</p> <pre> 367 static int fib_del_list(struct fib_node **fp, __u32 dst, 368 struct device * dev, __u32 gtw, short flags, short metric, __u32 mask) 369 { 370 struct fib_node *f; 371 int found=0; 372 373 while((f = *fp) != NULL) 374 { </pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51 ¹
		<pre>375 struct fib_info * fi = f->fib_info; 376 377 /* 378 * Make sure the destination and netmask match. 379 * metric, gateway and device are also checked 380 * if they were specified. 381 */ 382 if (f->fib_dst != dst 383 (gtw && fi->fib_gateway != gtw) 384 (metric >= 0 && f->fib_metric != metric) 385 (dev && fi->fib_dev != dev)) 386 { 387 fp = &f->fib_next; 388 continue; 389 } 390 cli(); 391 *fp = f->fib_next; 392 if (fib_loopback == f) 393 fib_loopback = NULL; 394 sti(); 395 ip_netlink_msg(RTMSG_DELROUTE, dst, gtw, mask, flags, metric, 396 fi->fib_dev->name); 397 fib_free_node(f); 398 found++; 399 } 400 return found; 401 }</pre>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>Further, Linux 1.3.51 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both Linux 1.3.51 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 Linux 1.3.51. 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 Linux 1.3.51 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.3.51 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in Linux 1.3.51 with the means for dynamically determining maximum number for the record search</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>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.</p> <p>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 Linux 1.3.51 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine Linux 1.3.51 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in Linux 1.3.51 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 Linux 1.3.51 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 Linux 1.3.51</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>with Thatte.</p> <p>Alternatively, it would also be obvious to combine Linux 1.3.51 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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.</p>

EXHIBIT D-9

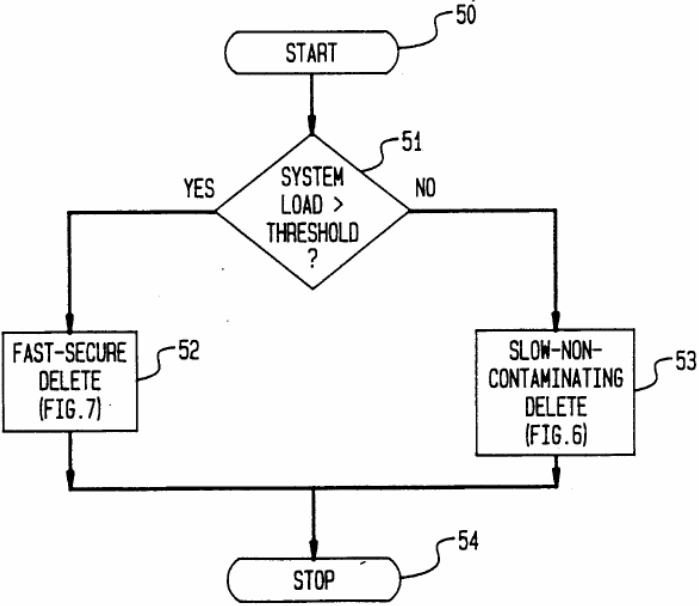
Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p><i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>During the hybrid deletion procedure decision block 51 checks the system load</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both Linux 1.3.51 and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 Linux 1.3.51. 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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120	Linux 1.3.51¹
	<p>patent's deletion decision procedure with Linux 1.3.51 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.3.51 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.</p> <p>Alternatively, it would also be obvious to combine Linux 1.3.51 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both Linux 1.3.51 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 Linux 1.3.51. 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</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>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 Linux 1.3.51 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 Linux 1.3.51 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in Linux 1.3.51 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 Linux 1.3.51 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 Linux 1.3.51 can be burdensome</p>

EXHIBIT D-9

Asserted Claims From U.S. Pat. No. 5,893,120		Linux 1.3.51¹
		<p>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.</p> <p>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.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, GCache discloses an information storage and retrieval system.</p> <p>For example, Comer discloses an information storage and retrieval system using hash tables of linked lists. <i>See, e.g.</i>, Comer at 2-11, Fig. 1.</p> <p>“This section describes our implementation of a generalized caching system.” <i>See</i> Comer at 2.</p> <p><i>See also</i>, gcache.c which implements the generalized caching mechanism as described in Comer.</p>
<p>[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,</p>	<p>[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,</p>	<p>GCache 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. GCache 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.</p> <p>For example, Comer discloses using linked lists to store records, the linked lists chained to a hash table using an external chaining technique:</p> <p>“GCache implements each cache as a separate hash table of buckets. Buckets are implemented as doubly linked lists of cache entry headers for easy insertion and deletion.” <i>See</i> Comer at 3.</p> <p>“GCache keeps all <i>cacheentry</i> structures for an active cache either on the free</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p><u>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”)</u></p>
	<p>list or, when they are in use, on a doubly linked list attached to a hash table slot. GCache computes a hash value as a function of the sum of bytes in the key buffer. GCache then computes the hash table slot as the hash value modulo the size of the hash table” <i>See</i> Comer at 5.</p> <p>“GCache implements the hash table as an array of structures representing buckets, each containing the head of a (possibly empty) doubly linked cache entry chain.” <i>See</i> Comer at 6.</p>

EXHIBIT D-10

**Asserted Claims From
U.S. Pat. No. 5,893,120**

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”)

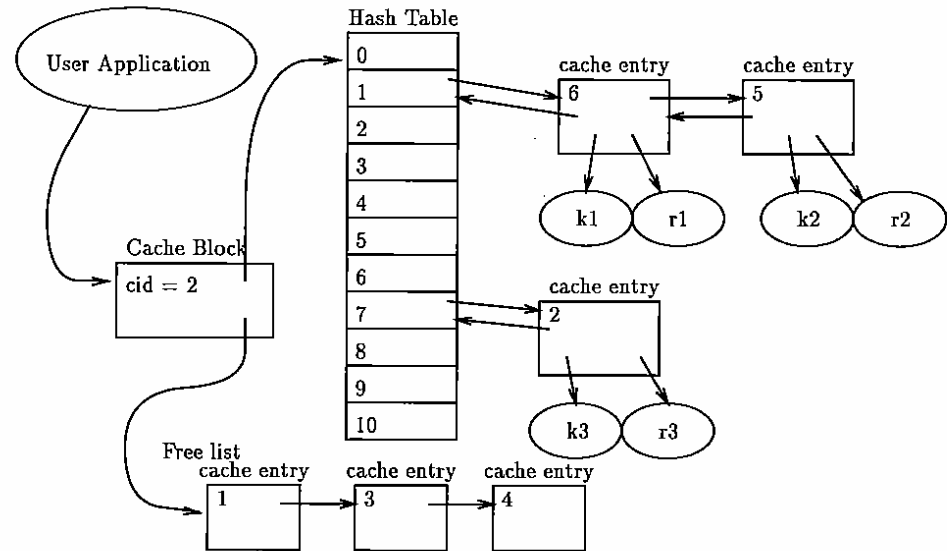


Figure 1: GCache Data Structures

See also, gcache.c at lines 53-64, defining cacheentry as a linked list, as shown in the code below:

```

53 struct cacheentry {
54 ce_status ce_status; /* INUSE or FREE */
55 char *ce_keyptr; /* pointer to the key */
56 tcelen ce_keylen; /* length of the key */
57 char *ce_resptr; /* pointer to the result */
58 tcelen ce_reslen; /* length of the result */

```

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<pre>59 thval ce hash; /* value that was hashed in */ 60 ttstamp ce_tsinsert; /* timestamp - time inserted */ 61 ttstamp ce_tsaccess; /* timestamp - last access */ 62 tceix ce_prev; /* next entry on list */ 63 tceix ce_next; /* prev entry on list */ 64 };</pre> <p>Comer discloses storing records in a linked list, for example:</p> <p>“Cainsert() inserts a new mapping, key => res, into the cache.” See Comer at 4.</p> <p>See also, gcache.c at lines 241-304, defining cainsert().</p> <p>Comer discloses providing access to records stored in a linked list, for example:</p> <p>“Calookup() searches for a cached entry matching the key passed as an argument.” See Comer at 4.</p> <p>See also, gcache.c at lines 307-347 and 637-678, defining calookup() and cagetindex().</p> <p>Comer discloses at least some of the records automatically expiring, for example:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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")</u></p>
		<p>contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned." See Comer at 10.</p> <p>See also, gcache.c at lines 617-634, defining caisold() which determines if the record has expired:</p> <pre> 617 /* 618 * ===== 619 * caisold - return TRUE if the given entry is "too old" 620 * ===== 621 */ 622 LOCAL int caisold(pcb,pce) 623 struct cacheblk *pcb; 624 struct cacheentry *pce; 625 { 626 unsigned now; 627 628 if (pcb->cb_maxlife == 0) 629 return(FALSE); 630 631 gettime(&now); 632 633 return ((now - pce->ce_tsaccess) > pcb->cb_maxlife); 634 } </pre>
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of</p>	<p>GCache discloses a record search means utilizing a search key to access the linked list. GCache also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
	<p>records having the same hash address,</p>	<p>For example, Comer discloses utilizing a search key to access a linked list of records having the same hash address. In the quoted text below, the use of hash value to determine a hash table with an attached linked list of records having the same hash address is an example of “utilizing a search key to access a linked list.”</p> <p>“GCache implements each cache as a separate hash table of buckets. Buckets are implemented as doubly linked lists of cache entry headers for easy insertion and deletion.” <i>See</i> Comer at 3.</p> <p>“GCache keeps all <i>cacheentry</i> structures for an active cache either on the free list or, when they are in use, on a doubly linked list attached to a hash table slot. GCache computes a hash value as a function of the sum of bytes in the key buffer. GCache then computes the hash table slot as the hash value modulo the size of the hash table” <i>See</i> Comer at 5.</p> <p>“GCache implements the hash table as an array of structures representing buckets, each containing the head of a (possibly empty) doubly linked cache entry chain.” <i>See</i> Comer at 6.</p> <p><i>See also</i>, <i>gcache.c</i> at lines 637-677, defining <i>cagetindex()</i>, which contains code constituting a “record search means” that uses a hash value to access and traverse a linked list of records having the same hash address:</p> <pre> 637 /* 638 * ===== </pre>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
		<pre>===== 639 * cagetindex - return the index of a matching entry, or SYSERR 640 * N.B. assumes MUTEX is already held 641 * =====</pre> <pre>===== 642 */ 643 LOCAL tceix cagetindex(pcb,pkey,keylen,hash) 644 struct cacheblk *pcb; 645 char *pkey; 646 tcelen keylen; 647 thval hash; 648 { 649 struct cacheentry *pce; 650 tceix ix; 651 tceix nextix; 652 653 ++pcb->cb_lookups; 654 655 ix = pcb->cb_hash [HASHTOIX (hash,pcb)] .he_ix; 656 657 while (ix != NULL_IX) { 658 pce = &pcb->cb_cache[ix]; 659 nextix = pce->ce_next; 660 661 if ((pce->ce_hash == hash) && 662 (pce->ce_keylen == keylen) && 663 (blkequ (pkey,pce->ce_keyptr,keylen))) { 664 /* this is a match */ 665 ++pcb->cb_hits; 666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return (NULL_IX); 670 } else { 671 return (ix);</pre>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<pre> 672 } 673 } 674 ix = nextix; 675 } 676 677 return(NULL_IX); 678 } </pre>
[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	<p>GCache 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. GCache 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.</p> <p>For example, Comer discloses that “<i>Calookup()</i> searches for a cached entry matching the key passed as an argument.” <i>See</i> Comer at 4.</p> <p>At line 333 of gcache.c, calookup() calls the function cagetindex():</p> <pre> 333 if ((ix = cagetindex(pcb,pkey,keylen,hash)) != NULL_IX) { </pre> <p>In addition Comer discloses a means for identifying and removing expired records from the linked list of records:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See</i> Comer at 10.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<p>At lines 655-670 of gcache.c, cagetindex() executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, cagetindex() utilizes caisold() to identify if a matching record is expired and removes the expired record from the linked list:</p> <pre> 666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else { </pre>
<p>[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.</p>	<p>[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.</p>	<p>GCache 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. GCache 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.</p> <p>For example, Comer discloses means for inserting, retrieving, and deleting records:</p> <p>“<i>Cainsert()</i> inserts a new mapping, key => res, into the cache.” <i>See</i> Comer at 4.</p> <p><i>See also</i>, gcache.c at lines 246-304, defining cainsert().</p> <p>“<i>Calookup()</i> searches for a cached entry matching the key passed as an argument.” <i>See</i> Comer at 4.</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<p><i>See also</i>, gcache.c at lines 312-347 and 643-678, defining calookup() and cagetindex().</p> <p>“<i>Caremove()</i> removes the cached entry whose key is given, if one exists.” <i>See</i> Comer at 4.</p> <p><i>See also</i>, gcache.c at lines 355-376, defining caremove().</p> <p>Each of the means for inserting, retrieving, and deleting, utilizes a record search means, the function cagetindex(), which removes an expired record from the list as described below. The individual calls of cagetindex() are listed here:</p> <p>In cainsert(): 275 if ((ixnew = cagetindex(pcb,pkey,keylen,hash)) != NULL_IX) {</p> <p>In calookup(): 333 if ((ix = cagetindex(pcb,pkey,keylen,hash)) != NULL_IX) {</p> <p>In caremove(): 370 if ((ix = cagetindex(pcb,pkey,keylen,hash)) != NULL_IX) {</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See</i> Comer at 10.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<p>At lines 655-670 of gcache.c, cagetindex() executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, cagetindex() utilizes caisold() to identify if a matching record is expired and removes the expired record from the linked list using caunlink():</p> <pre> 666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else { </pre>
<p>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.</p>	<p>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.</p>	<p>GCache 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.</p> <p>For example, Comer discloses dynamically determining whether to delete one record or zero records from the accessed linked list of records:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See Comer at 10.</i></p> <p>At lines 655-670 of gcache.c, cagetindex() executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p><u>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”)</u></p>
	<p>therein. In the subset of that code listed below, <code>cagetindex()</code> utilizes <code>caisold()</code> to identify if a matching record is expired and removes the expired record from the linked list using <code>caunlink()</code> and returns. If the matching record is not expired, the code returns the index of the matched record:</p> <pre>666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else { 671 return(ix); 672 }</pre> <p>In a second example, Comer discloses dynamically determining whether to delete one record or zero records from an accessed list of records:</p> <p>“Insertion of a new entry into a full cache forces the deletion of the entry that was looked up the least recently.” <i>See</i> Comer at 3.</p> <p>At line 275 of <code>gcache.c</code>, <code>cainset()</code> calls <code>cagetindex()</code> to access a linked list of records and see if a matching entry already exists. If a matching entry does not exist, <code>cainset()</code> calls <code>cagetfree()</code> at line 281 to get a free entry. In <code>cagetfree()</code>, the following code dynamically determines whether to delete one record or zero records:</p> <pre>719 /* if the free list is empty, delete the oldest entry */ 720 if (pcb->cb_freelist == NULL_IX) { 721 cdeleteold(pcb); 722 ++pcb->cb_fulls; 723 }</pre>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>In addition, GCache 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.</p> <p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both GCache 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 GCache. 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 GCache nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 GCache and would</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center"><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
	<p>have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in GCache 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.</p> <p>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 GCache 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine GCache with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in GCache 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 GCache with the teachings of Thatte would solve this problem by dynamically determining how</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>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 GCache with Thatte.</p> <p>Alternatively, it would also be obvious to combine GCache 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:</p> <p style="padding-left: 40px;">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”).</p> <p>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</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
		<p>moving records in the chain as described above. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-10

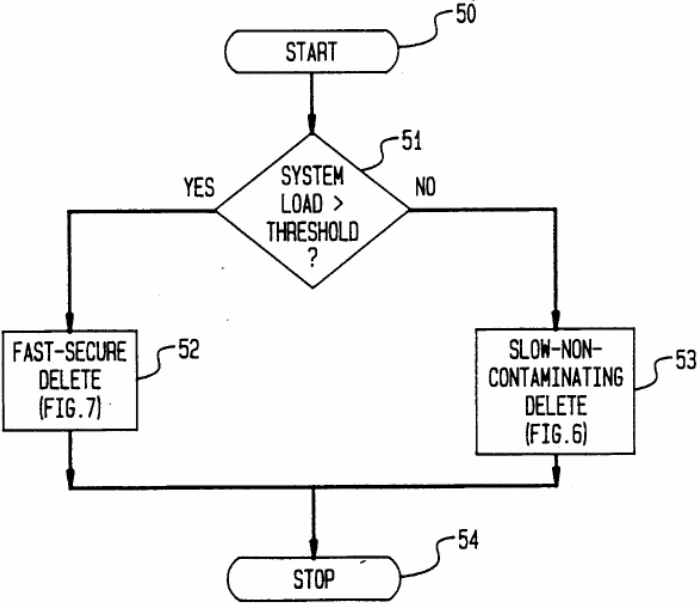
<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center"><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, GCache: A Generalized Caching Mechanism, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
	<p align="center">FIG. 5 HYBRID DELETION</p>  <pre> graph TD 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?} 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)] 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)] 52 --> 54([STOP]) 53 --> 54 </pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both GCache and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 GCache. 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 GCache would be nothing more than the predictable use of prior art elements according to their established functions.</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
		<p>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 GCache 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.</p> <p>Alternatively, it would also be obvious to combine GCache with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both GCache 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 GCache. 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</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<p>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 GCache would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 GCache and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in GCache 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 GCache 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 GCache can be burdensome on the system, adding to the system’s load and slowing down the system’s processing.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<p>Moreover, the removal could also force an interruption in real-time processing as the processing waits for the removal to complete.</p> <p>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.</p>
<p>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:</p>	<p>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</p>	<p>To the extent the preamble is a limitation, GCache 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. GCache 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.</p> <p>For example, Comer discloses using linked lists to store records, the linked lists chained to a hash table using an external chaining technique:</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u>
	method comprising the steps of:	<p>“GCache implements each cache as a separate hash table of buckets. Buckets are implemented as doubly linked lists of cache entry headers for easy insertion and deletion.” <i>See</i> Comer at 3.</p> <p>“GCache keeps all <i>cacheentry</i> structures for an active cache either on the free list or, when they are in use, on a doubly linked list attached to a hash table slot. GCache computes a hash value as a function of the sum of bytes in the key buffer. GCache then computes the hash table slot as the hash value modulo the size of the hash table” <i>See</i> Comer at 5.</p> <p>“GCache implements the hash table as an array of structures representing buckets, each containing the head of a (possibly empty) doubly linked cache entry chain.” <i>See</i> Comer at 6.</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center"><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
	<div data-bbox="987 370 1921 917" data-label="Diagram"> <p>The diagram illustrates the GCache data structures. A central Hash Table is shown as a vertical array of 11 slots, indexed 0 to 10. A User Application is shown interacting with the Hash Table and a Cache Block. The Cache Block contains 'cid = 2'. A Free list consists of three cache entries with indices 1, 3, and 4. Three cache entries are shown: one at index 6 containing key '6' and result '5', one at index 5 containing key '5' and result '2', and one at index 7 containing key '2' and result '3'. Arrows indicate pointers from the Hash Table to the cache entries and from the cache entries to their respective keys and results.</p> </div> <p align="center">Figure 1: GCache Data Structures</p> <p><i>See also, gcache.c at lines 53-64, defining cacheentry as a linked list, as shown in the code below:</i></p> <pre> 53 struct cacheentry { 54 ce_status ce_status; /* INUSE or FREE */ 55 char *ce_keyptr; /* pointer to the key */ 56 tcelen ce_keylen; /* length of the key */ 57 char *ce_resptr; /* pointer to the result */ 58 tcelen ce_reslen; /* length of the result */ </pre>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<pre>59 thval ce hash; /* value that was hashed in */ 60 ttstamp ce_tsinsert; /* timestamp - time inserted */ 61 ttstamp ce_tsaccess; /* timestamp - last access */ 62 tceix ce_prev; /* next entry on list */ 63 tceix ce_next; /* prev entry on list */ 64 };</pre> <p>Comer discloses storing records in a linked list, for example:</p> <p>“Cainsert() inserts a new mapping, key => res, into the cache.” See Comer at 4.</p> <p>See also, gcache.c at lines 241-304, defining cainsert().</p> <p>Comer discloses providing access to records stored in a linked list, for example:</p> <p>“Calookup() searches for a cached entry matching the key passed as an argument.” See Comer at 4.</p> <p>See also, gcache.c at lines 307-347 and 637-678, defining calookup() and cagetindex().</p> <p>Comer discloses at least some of the records automatically expiring, for example:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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")</u></p>
		<p>contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned." See Comer at 10.</p> <p>See also, gcache.c at lines 617-634, defining caisold() which determines if the record has expired:</p> <pre> 617 /* 618 * ===== ===== 619 * caisold - return TRUE if the given entry is "too old" 620 * ===== ===== 621 */ 622 LOCAL int caisold(pcb,pce) 623 struct cacheblk *pcb; 624 struct cacheentry *pce; 625 { 626 unsigned now; 627 628 if (pcb->cb_maxlife == 0) 629 return(FALSE); 630 631 gettime(&now); 632 633 return ((now - pce->ce_tsaccess) > pcb->cb_maxlife); 634 } </pre>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>GCache discloses accessing a linked list of records. GCache also discloses accessing a linked list of records having same hash address.</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p><u>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”)</u></p>
	<p>For example, Comer discloses accessing a linked list of records having the same hash address:</p> <p>“GCache implements each cache as a separate hash table of buckets. Buckets are implemented as doubly linked lists of cache entry headers for easy insertion and deletion.” <i>See</i> Comer at 3.</p> <p>“GCache keeps all <i>cacheentry</i> structures for an active cache either on the free list or, when they are in use, on a doubly linked list attached to a hash table slot. GCache computes a hash value as a function of the sum of bytes in the key buffer. GCache then computes the hash table slot as the hash value modulo the size of the hash table” <i>See</i> Comer at 5.</p> <p>“GCache implements the hash table as an array of structures representing buckets, each containing the head of a (possibly empty) doubly linked cache entry chain.” <i>See</i> Comer at 6.</p> <p><i>See also</i>, gcache.c at lines 637-677, defining <code>cagetindex()</code>, which contains code constituting a “record search means” that uses a hash value to access and traverse a linked list of records having the same hash address:</p> <pre>637 /* 638 * ===== ===== 639 * cagetindex - return the index of a matching entry, or SYSERR 640 * N.B. assumes MUTEX is already held 641 *</pre>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<pre>===== 642 */ 643 LOCAL tceix cagetindex(pcb,pkey,keylen,hash) 644 struct cacheblk *pcb; 645 char *pkey; 646 tcelen keylen; 647 thval hash; 648 { 649 struct cacheentry *pce; 650 tceix ix; 651 tceix nextix; 652 653 ++pcb->cb_lookups; 654 655 ix = pcb->cb_hash[HASHTOIX(hash,pcb)].he_ix; 656 657 while (ix != NULL_IX) { 658 pce = &pcb->cb_cache[ix]; 659 nextix = pce->ce_next; 660 661 if ((pce->ce_hash == hash) && 662 (pce->ce_keylen == keylen) && 663 (blkequ(pkey,pce->ce_keyptr,keylen))) { 664 /* this is a match */ 665 ++pcb->cb_hits; 666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else { 671 return(ix); 672 } 673 } 674 ix = nextix; 675 } 676</pre>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<pre>677 return(NULL_IX); 678 }</pre>
<p>[3b] identifying at least some of the automatically expired ones of the records, and</p>	<p>[7b] identifying at least some of the automatically expired ones of the records,</p>	<p>GCache discloses identifying at least some of the automatically expired ones of the records.</p> <p>For example, Comer discloses a means for identifying and removing expired records from the linked list of records:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” See Comer at 10.</p> <p>At lines 655-670 of gcache.c, cagetindex() executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, cagetindex() utilizes caisold() to identify if a matching record is expired and removes the expired record from the linked list:</p> <pre>666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else {</pre>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>GCache discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example:</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See</i> Comer at 10.</p> <p>At lines 655-670 of gcache.c, <i>cagetindex()</i> executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, <i>cagetindex()</i> utilizes <i>caisold()</i> to identify if a matching record is expired and removes the expired record from the linked list:</p> <pre> 666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else { </pre>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>GCache discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, Comer discloses means for inserting records:</p> <p>“<i>Cainsert()</i> inserts a new mapping, key => res, into the cache.” <i>See</i> Comer at 4.</p> <p>At line 275 of gcache.c, <i>cainsert()</i> utilizes a record search means, the function <i>cagetindex()</i>, which removes an expired record from the list as described below.</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See</i> Comer at 10.</p> <p>At lines 655-670 of gcache.c, cagetindex() executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, cagetindex() utilizes caisold() to identify if a matching record is expired and if so, removes the expired record from the linked list using caunlink():</p> <pre>666 if (caisold(pcb,pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb,ix); 669 return(NULL_IX); 670 } else {</pre> <p>After the call to cagetindex() returns, through which the expired entry was removed, cainser() proceeds to insert a new entry at the head of the list and populates the fields of the structure, as shown in the code below:</p> <pre>275 if ((ixnew = cagetindex(pcb,pkey,keylen,hash)) != NULL_IX) { 276 /* use the old one */ 277 caclear(pcb,ixnew); 278 pce = &pcb->cb_cache[ixnew]; 279 } else { 280 /* get a free cacheentry */ 281 ixnew = cagetfree(pcb); 282 pce = &pcb->cb_cache[ixnew]; 283</pre>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, <i>GCache: A Generalized Caching Mechanism</i>, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
		<pre>284 /* ... and put it at the head of the list */ 285 pce->ce_prev = 0; 286 pce->ce_next = phe->he_ix; 287 pcb->cb_cache[phe->he_ix].ce_prev = ixnew; 288 phe->he_ix = ixnew; 289 } 290 291 pce->ce_status = CE_INUSE; 292 pce->ce_hash = hash; 293 pce->ce_keyptr = cagetmem(keylen); 294 pce->ce_keylen = keylen; 295 blkcopy(pce->ce_keyptr, pkey, keylen); 296 pce->ce_resptr = cagetmem(reslen); 297 pce->ce_reslen = reslen; 298 blkcopy(pce->ce_resptr, pres, reslen); 299 gettime(&pce->ce_tsinsert); 300 pce->ce_tsaccess = pce->ce_tsinsert;</pre> <p>In a second example, caunlink(), which is called by other functions including caremove() and cagetindex(), removes a record from the linked list by modifying the values of ce_next and ce_prev in the records to which it was linked and then deletes the data stored in a record and frees memory by calling cclear(), as shown in the code below:</p> <pre>750 pce = &pcb->cb_cache[ix]; 751 hash = pce->ce_hash; 752 phe = &pcb->cb_hash[HASHTOIX(hash, pcb)]; 753 754 if (pce->ce_prev == NULL_IX) 755 phe->he_ix = pce->ce_next; 756 else 757 pcb->cb_cache[pce->ce_prev].ce_next = pce->ce_next; 758 759 pcb->cb_cache[pce->ce_next].ce_prev = pce->ce_prev; 760</pre>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center"><u>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”)</u></p>
		<pre>761 caclear(pcb, ix);</pre>
<p>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.</p>	<p>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.</p>	<p>GCache discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, Comer discloses dynamically determining whether to delete one record or zero records from the accessed linked list of records:</p> <p>“In a simpler and cleaner design chosen for GCache, each cached entry contains a timestamp encoding the insertion time. If a lookup matches an entry with an expired timestamp, that entry is removed rather than being returned.” <i>See Comer at 10.</i></p> <p>At lines 655-670 of gcache.c, <code>cagetindex()</code> executes a <i>while</i> loop to traverse a linked list attached to a bucket of the hash table, accessing records stored therein. In the subset of that code listed below, <code>cagetindex()</code> utilizes <code>caisold()</code> to identify if a matching record is expired and removes the expired record from the linked list using <code>caunlink()</code> and returns. If the matching record is not expired, the code returns the index of the matched record:</p> <pre>666 if (caisold(pcb, pce)) { 667 ++pcb->cb_tos; 668 caunlink(pcb, ix); 669 return(NULL_IX); 670 } else { 671 return(ix); 672 }</pre>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>In a second example, Comer discloses dynamically determining whether to delete one record or zero records from an accessed linked list of records:</p> <p>“Insertion of a new entry into a full cache forces the deletion of the entry that was looked up the least recently.” <i>See</i> Comer at 3.</p> <p>At line 275 of gcache.c, cainset() calls cagetindex() to access a linked list of records and see if a matching entry already exists. If a matching entry does not exist, cainset() calls cagetfree() at line 281 to get a free entry. In cagetfree(), the following code dynamically determines whether to delete one record or zero records:</p> <pre>719 /* if the free list is empty, delete the oldest entry */ 720 if (pcb->cb_freelist == NULL_IX) { 721 cdeleteold(pcb); 722 ++pcb->cb_fulls; 723 }</pre> <p>In addition, GCache 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.</p> <p>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.</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-10

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center"><u>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”)</u></p>
	<p>associated VSID is on the recycle list (Step 30). The predetermined number of entries that are swept is identified as k, where:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p align="right"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both GCache 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 GCache. Moreover, one of ordinary</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>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 GCache would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 GCache and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in GCache 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.</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>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 GCache 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine GCache with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in GCache 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 GCache 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 GCache with Thatte.</p> <p>Alternatively, it would also be obvious to combine GCache 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:</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>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”).</p> <p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-10

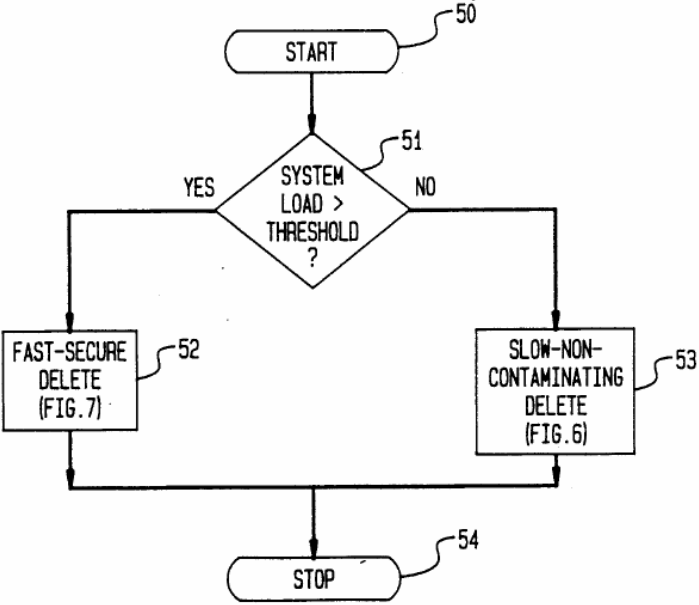
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p><u>gcache.c from Xinu Operating System for Sparc (1991) (hereinafter “gcache.c”) and Douglas Comer and Shawn Ostermann, GCache: A Generalized Caching Mechanism, Purdue University (Revised March 1992) (hereinafter “Comer”) (collectively hereinafter “GCache”)</u></p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64, Figure 5. On the other hand, if the system load is less than the threshold, then a</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both GCache and the '663 patent relate to deletion of records from hash tables using external chaining, one of ordinary skill in the art would understand 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 GCache. 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 GCache would be nothing more than the predictable use of prior art elements according to their established</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>functions.</p> <p>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 GCache 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.</p> <p>Alternatively, it would also be obvious to combine GCache with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>Every time a user-input routine is invoked, a decision routine can decide</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both GCache 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 GCache. 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</p>

EXHIBIT D-10

<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p><u>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”)</u></p>
		<p>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 GCache would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 GCache 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in GCache 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 GCache 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</p>

EXHIBIT D-10

Asserted Claims From U.S. Pat. No. 5,893,120		<u>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”)</u>
		<p>of expired records described in GCache 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.</p> <p>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.</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist-domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
<p>1. An information storage and retrieval system, the system comprising:</p>	<p>5. An information storage and retrieval system, the system comprising:</p>	<p>To the extent the preamble is a limitation, NRL IPv6 discloses an information storage and retrieval system.</p> <p>For example, NRL IPv6 discloses a linked list of automatically expiring data. <i>See, e.g.</i>, struct_keyacquirelist defined in key.h at lines 188-194.</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>
<p>[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,</p>	<p>[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,</p>	<p>NRL IPv6 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.</p> <p>NRL IPv6 in combination with Robert L. Kruse, <i>Data Structures & Program Design</i> (Prentice Hall 1987) (hereinafter “Kruse”) 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.</p> <p>One of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in NRL IPv6 with the 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 disclosed by Kruse. <i>See, e.g.</i>, Kruse at 206-208. For example, since NRL IPv6, as discussed below, utilizes a linked list for storing records and Kruse discloses attaching or chaining linked lists to a hash table for storing records, one of ordinary skill in the art would be motivated to combine the linked list of NRL IPv6 with the hash table using</p>

¹ Available as of August 1995.

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>external chaining of linked lists disclosed by Kruse. The disclosure of these claim elements in Kruse is clearly shown in the chart of Kruse, which is hereby incorporated by reference in its entirety.</p> <p>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 NRL IPv6 with Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre>188 struct key_acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 };</pre> <p>In addition, key.c discloses traversing key_acquirelist and accessing entries stored therein. <i>See, e.g.</i>, key.c at lines 1411, 1430-1459.</p> <p>Also, key.c and key.h disclose automatically expiring records. For example, the key_acquirelist structure defined in key.h contains the following code:</p> <pre>192 u_long expiretime; /* expiration time for acquire message */</pre>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>For example, key.c checks to see if a record has expired using the above-described field in key_acquirelist, as shown in the code below:</p> <pre>1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 }</pre> <p>Further, Kruse discloses hash tables with external chaining. <i>See, e.g.</i>, Kruse at 206-208. One of ordinary skill in the art would be motivated to, and would understand how to, combine the systems and methods of NRL IPv6 with the systems and methods of using hash tables with external chaining disclosed by Kruse.</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”) ¹
<p>[1b] a record search means utilizing a search key to access the linked list,</p>	<p>[5b] a record search means utilizing a search key to access a linked list of records having the same hash address,</p>	<p>The combination of NRL IPv6 and Kruse discloses a record search means utilizing a search key to access the linked list. The combination NRL IPv6 and Kruse also discloses a record search means utilizing a search key to access a linked list of records having the same hash address.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre> 188 struct key_acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 }; </pre> <p>In addition, key.c discloses traversing key_acquirelist—accessing records stored therein--to search for matching record., as shown in the code below:</p> <pre> 1411 struct key_acquirelist *ap, *prevap; . . . 1430 prevap = key_acquirelist; 1431 for(ap = key_acquirelist->next; ap; ap = ap->next) { 1432 if (addrpart_equal(dst, (struct sockaddr *)&(ap->target)) && 1433 (etype == ap->type)) { 1434 DPRINTF(IDL_MAJOR_EVENT, ("acquire message previously sent!\n")); </pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<pre>1435 if (ap->expiretime < time.tv_sec) { 1436 DPRINTF(IDL_MAJOR_EVENT, ("acquire message has expired!\n")); 1437 ap->count = 0; 1438 break; 1439 }</pre> <p>Further, Kruse discloses hash tables and hash tables with external chaining. <i>See, e.g.</i>, Kruse at 198-208. One of ordinary skill in the art would be motivated to, and would understand how to, combine the systems and methods of NRL IPv6 with the systems and methods of using hash tables with external chaining disclosed by Kruse. In such a combination, one of ordinary skill in the art would recognize that a hash key is used to access a list of records having the same hash address (a linked list chained to a hash bucket).</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>
<p>[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</p>	<p>[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</p>	<p>NRL IPv6 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. NRL IPv6 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.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre>188 struct key_acquirelist { 189 u int8 type; /* secassoc type to acquire */</pre>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<pre>190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 };</pre> <p>In addition, key.c, within the function key_acquire(), discloses traversing key_acquirelist and accessing entries stored therein. <i>See, e.g.</i>, key.c at lines 1411, 1430-1459.</p> <p>Also, key.c and key.h disclose automatically expiring records. For example, the key_acquirelist structure defined in key.h contains the following code:</p> <pre>192 u_long expiretime; /* expiration time for acquire message */</pre> <p>Furthermore, key.c discloses 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, as shown in the code below:</p> <pre>1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */</pre>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”) ¹
		<pre> 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 } </pre> <p><i>See also, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</i></p>
[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.	<p>NRL IPv6 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. NRL IPv6 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.</p> <p>The “means, utilizing the record search means” limitation is met, for example, by a function that calls key_acquire(). At line 1835 of key.c, for example, key_acquire() is called by the function getassocbysocket().</p> <p>In addition, the function key_acquire() in key.c contains a <i>for</i> loop beginning at line 1431. Within the <i>for</i> loop, the code starting at the <i>elseif</i> at line 1445 and ending at line 1457, modifies a pointer in an element of the linked list such that it removes the expired item from the linked list and then calls KFree(). After KFree() is called, control returns to the <i>for</i> loop which, unless it has reached the end of the linked list, will access the next record in the list. If the record has not been previously sent and has expired, it will be removed and deleted in the code starting at the <i>elseif</i> at line 1445 and ending at line 1457. Finally, after the</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<p>loop described above completes, key_acquire() contains the following code to insert an entry into key_acquirelist:</p> <pre> 1542 /* 1543 * Update the acquirelist 1544 */ 1545 if (success) { 1546 if (!ap) { 1547 DPRINTF(IDL_MAJOR_EVENT, ("Adding new entry in acquirelist\n")); 1548 K_Malloc(ap, struct key_acquirelist *, sizeof(struct key_acquirelist)); 1549 if (ap == 0) 1550 return(success ? 0 : -1); 1551 bzero((char *)ap, sizeof(struct key_acquirelist)); 1552 bcopy((char *)dst, (char *)&(ap->target), dst->sa_len); 1553 ap->type = etype; 1554 ap->next = key_acquirelist->next; 1555 key_acquirelist->next = ap; 1556 } 1557 DPRINTF(IDL_EVENT, ("Updating acquire counter and expiration time\n")); 1558 ap->count++; 1559 ap->expiretime = time.tv_sec + maxacquiretime; 1560 } </pre> <p><i>See also, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</i></p>
<p>2. The information storage and retrieval system according to claim 1</p>	<p>6. The information storage and retrieval system according to claim 5</p>	<p>NRL IPv6 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.</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
<p>further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.</p>	<p>further including means for dynamically determining maximum number for the record search means to remove in the accessed linked list of records.</p>	<p>For example, in key.c each time the else if statement at line 1445 is executed, it dynamically determines the maximum number of records to remove—one or zero—based on whether the record has expired.</p> <pre> 1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 } </pre> <p>Further, NRL IPv6 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.</p> <p>Dirks discloses the management of memory in a computer system and more particularly to the allocation of address space in a virtual memory system,</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>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.</p> <p>As summarized in Dirks,</p> <p>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.</p> <p>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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value k. <i>Id.</i> at 7:15-46, 7:66-8:56.</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>As both NRL IPv6 and Dirks relate to deletion of aged records, 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 NRL IPv6. 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 NRL IPv6 nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 NRL IPv6 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in NRL IPv6 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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>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.</p> <p>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 NRL IPv6 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine NRL IPv6 with Thatte and recognize the benefits of doing so. For example, the removal of expired records described in NRL IPv6 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 NRL IPv6 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 NRL IPv6</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<p>with Thatte.</p> <p>Alternatively, it would also be obvious to combine NRL IPv6 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:</p> <p style="padding-left: 40px;">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”).</p> <p style="padding-left: 40px;">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. <i>Id.</i> at 2:35-41.</p> <p style="padding-left: 40px;">This hybrid hashing technique has the decided advantage of automatically eliminating contamination caused by the fast-secure deletion procedure when the slower, non-contaminating</p>

EXHIBIT D-11

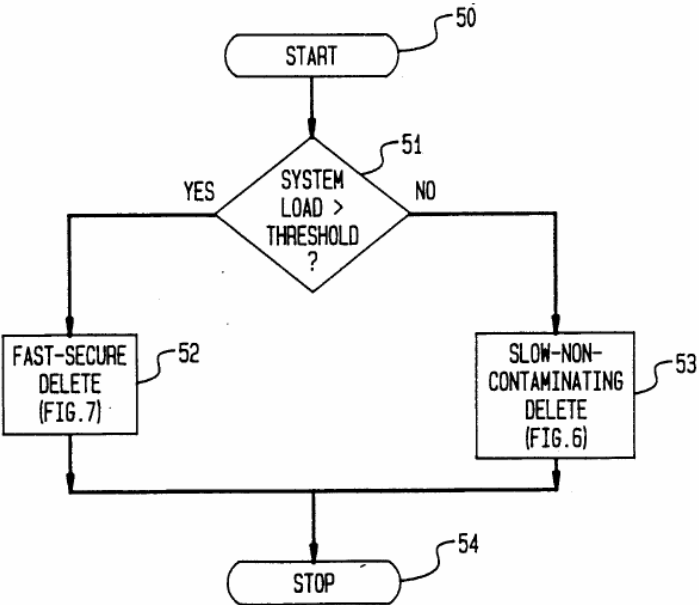
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<p>deletion is used when the load on the system is at lower levels. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p> <p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p>The flowchart, labeled FIG. 5 HYBRID DELETION, illustrates a decision-based process. It begins with a 'START' terminal (50) leading to a decision diamond (51) asking 'SYSTEM LOAD > THRESHOLD?'. If the answer is 'YES', the process flows to a 'FAST-SECURE DELETE (FIG. 7)' block (52). If the answer is 'NO', it flows to a 'SLOW-NON-CONTAMINATING DELETE (FIG. 6)' block (53). Both paths converge to a 'STOP' terminal (54).</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist-domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> 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. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the ’663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both NRL IPv6 and the ’663 patent relate to deletion of records from an information storage system, one of ordinary skill in the art would understand how to use the ’663 patent’s dynamic decision on whether to perform a deletion based on a systems load in other information storage system implementations such as that described by NRL IPv6. 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</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<p>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 NRL IPv6 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 NRL IPv6 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.</p> <p>Alternatively, it would also be obvious to combine NRL IPv6 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA ’89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>When a significant pause has been detected, a decision procedure is</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>As both NRL IPv6 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 NRL IPv6. 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 NRL IPv6 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 NRL IPv6 and would have seen the benefits of doing so. One such benefit, for example, is preventing slowdown of the system.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in NRL IPv6 to dynamically determine the</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<p>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 NRL IPv6 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 NRL IPv6 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.</p> <p>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.</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>
<p>3. A method for storing and retrieving information</p>	<p>7. A method for storing and retrieving information</p>	<p>To the extent the preamble is a limitation, NRL IPv6 discloses a method for storing and retrieving information records using a linked list to store and</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
<p>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:</p>	<p>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:</p>	<p>provide access to the records, at least some of the records automatically expiring. The combination of NRL IPv6 and Kruse 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.</p> <p>One of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in NRL IPv6 with hashing technique to provide access to the records and using an external chaining technique to store the records with same hash address as disclosed by Kruse. <i>See, e.g.</i>, Kruse at 206-208. For example, since NRL IPv6, as discussed below, utilizes a linked list for storing records and Kruse discloses attaching or chaining linked lists to a hash table for storing records, one of ordinary skill in the art would be motivated to combine the linked list of NRL IPv6 with the hash table using external chaining of linked lists disclosed by Kruse. The disclosure of these claim elements in Kruse is clearly shown in the chart of Kruse, which is hereby incorporated by reference in its entirety.</p> <p>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 NRL IPv6 with Kruse would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<pre>188 struct key acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 };</pre> <p>In addition, key.c discloses traversing key_acquirelist and accessing entries stored therein. <i>See, e.g.</i>, key.c at lines 1411, 1430-1459.</p> <p>Also, key.c and key.h disclose automatically expiring records. For example, the key_acquirelist structure defined in key.h contains the following code:</p> <pre>192 u_long expiretime; /* expiration time for acquire message */</pre> <p>For example, key.c checks to see if a record has expired using the above-described field in key_acquirelist, as shown in the code below:</p> <pre>1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL MAJOR EVENT, ("found an expired</pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<pre>entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 }</pre> <p>Further, Kruse discloses hash tables with external chaining. <i>See, e.g.</i>, Kruse at 206-208. One of ordinary skill in the art would be motivated to, and would understand how to, combine the systems and methods of NRL IPv6 with the systems and methods of using hash tables with external chaining disclosed by Kruse.</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>
<p>[3a] accessing the linked list of records,</p>	<p>[7a] accessing a linked list of records having same hash address,</p>	<p>NRL IPv6 discloses accessing a linked list of records. The combination of NRL IPv6 and Kruse discloses accessing a linked list of records having same hash address.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre>188 struct key_acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 };</pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<p>In addition, key.c discloses traversing key_acquirelist—accessing records stored therein--to search for matching record, as shown in the code below:</p> <pre> 1411 struct key_acquirelist *ap, *prevap; . . . 1430 prevap = key_acquirelist; 1431 for(ap = key_acquirelist->next; ap; ap = ap->next) { 1432 if (addrpart_equal(dst, (struct sockaddr *)&(ap->target)) && 1433 (etype == ap->type)) { 1434 DPRINTF(IDL_MAJOR_EVENT, ("acquire message previously sent!\n")); 1435 if (ap->expiretime < time.tv_sec) { 1436 DPRINTF(IDL_MAJOR_EVENT, ("acquire message has expired!\n")); 1437 ap->count = 0; 1438 break; 1439 } </pre> <p>Further, Kruse discloses hash tables and hash tables with external chaining. <i>See, e.g.</i>, Kruse at 198-208. One of ordinary skill in the art would be motivated to, and would understand how to, combine the systems and methods of NRL IPv6 with the systems and methods of using hash tables with external chaining disclosed by Kruse. In such a combination, one of ordinary skill in the art would recognize that a hash key is used to access a list of records having the same hash address (a linked list chained to a hash bucket).</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>
<p>[3b] identifying at least</p>	<p>[7b] identifying at least</p>	<p>NRL IPv6 discloses identifying at least some of the automatically expired ones</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
some of the automatically expired ones of the records, and	some of the automatically expired ones of the records,	<p>of the records.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre>188 struct key_acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 };</pre> <p>In addition, key.c, within the function key_acquire(), discloses traversing key_acquirelist and accessing entries stored therein. <i>See, e.g.</i>, key.c at lines 1411, 1430-1459.</p> <p>Also, key.c and key.h disclose automatically expiring records. For example, the key_acquirelist structure defined in key.h contains the following code:</p> <pre>192 u_long expiretime; /* expiration time for acquire message */</pre> <p>Furthermore, key.c discloses 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, as shown in the code below:</p> <pre>1445 } else if (ap->expiretime < time.tv sec) {</pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<pre> 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 } </pre> <p><i>See also, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</i></p>
<p>[3c] removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p>	<p>[7c] removing at least some of the automatically expired records from the linked list when the linked list is accessed, and</p>	<p>NRL IPv6 discloses removing at least some of the automatically expired records from the linked list when the linked list is accessed.</p> <p>For example, as defined in key.h and shown in the code below, the key_acquirelist structure is a linked list:</p> <pre> 188 struct key_acquirelist { 189 u_int8 type; /* secassoc type to acquire */ 190 struct sockaddr_in6 target; /* destination address of secassoc */ 191 u_int32 count; /* number of acquire messages sent */ 192 u_long expiretime; /* expiration time for acquire message */ 193 struct key_acquirelist *next; 194 }; </pre>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>In addition, key.c, within the function key_acquire(), discloses traversing key_acquirelist and accessing entries stored therein. <i>See, e.g.,</i> key.c at lines 1411, 1430-1459.</p> <p>Also, key.c and key.h disclose automatically expiring records. For example, the key_acquirelist structure defined in key.h contains the following code:</p> <pre>192 u_long expiretime; /* expiration time for acquire message */</pre> <p>Furthermore, key.c discloses 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, as shown in the code below:</p> <pre>1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); 1456 ap = prevap; 1457 }</pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>		<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
		<p><i>See also, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</i></p>
	<p>[7d] inserting, retrieving or deleting one of the records from the system following the step of removing.</p>	<p>NRL IPv6 discloses inserting, retrieving or deleting one of the records from the system following the step of removing.</p> <p>For example, the function <code>key_acquire()</code> in <code>key.c</code> contains a <i>for</i> loop beginning at line 1431. Within the <i>for</i> loop, the code starting at the <i>elseif</i> at line 1445 and ending at line 1457, modifies a pointer in an element of the linked list such that it removes the expired item from the linked list and then calls <code>KFree()</code>. After <code>KFree()</code> is called, control returns to the <i>for</i> loop which, unless it has reached the end of the linked list, will retrieve the next record in the list. If the record has not been previously sent and has expired, it will be removed and deleted in the code starting at the <i>elseif</i> at line 1445 and ending at line 1457. Finally, after the loop described above completes, <code>key_acquire()</code> contains the following code to insert an entry into <code>key_acquirelist</code>:</p> <pre> 1542 /* 1543 * Update the acquirelist 1544 */ 1545 if (success) { 1546 if (!ap) { 1547 DPRINTF(IDL_MAJOR_EVENT, ("Adding new entry in acquirelist\n")); 1548 K_Malloc(ap, struct key_acquirelist *, sizeof(struct key_acquirelist)); 1549 if (ap == 0) 1550 return(success ? 0 : -1); 1551 bzero((char *)ap, sizeof(struct key_acquirelist)); 1552 bcopy((char *)dst, (char *)&(ap->target), dst->sa_len); 1553 ap->type = etype; </pre>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”) ¹
		<pre> 1554 ap->next = key_acquirelist->next; 1555 key_acquirelist->next = ap; 1556 } 1557 DPRINTF(IDL_EVENT, ("Updating acquire counter and expiration time\n")); 1558 ap->count++; 1559 ap->expiretime = time.tv_sec + maxacquiretime; 1560 } </pre> <p><i>See also, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</i></p>
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.	<p>NRL IPv6 discloses dynamically determining maximum number of expired ones of the records to remove when the linked list is accessed.</p> <p>For example, in key.c each time the else if statement at line 1445 is executed, it dynamically determines the maximum number of records to remove—one or zero—based on whether the record has expired.</p> <pre> 1445 } else if (ap->expiretime < time.tv_sec) { 1446 /* 1447 * Since we're already looking at the list, we may as 1448 * well delete expired entries as we scan through the list. 1449 * This should really be done by a function like key_reaper() 1450 * but until we code key_reaper(), this is a quick and dirty 1451 * hack. 1452 */ 1453 DPRINTF(IDL_MAJOR_EVENT, ("found an expired entry...deleting it!\n")); 1454 prevap->next = ap->next; 1455 KFree(ap); </pre>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<pre>1456 ap = prevap; 1457 }</pre> <p>In addition, NRL IPv6 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.</p> <p>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 incorporated¹ by reference in its entirety.</p> <p>As summarized in Dirks,</p> <p>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.</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>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:</p> $k = \frac{\text{total number of page table entries}}{\text{maximum number of active threads}}$ <p style="text-align: right;"><i>Id.</i> at 8:12-30.</p> <p>Dirks discloses that any approach can be employed to determine the number of entries to be examined during each step of the sweeping process. <i>Id.</i> at 7:37-40. As stated in Dirks:</p> <p>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</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Naval Research Laboratories ipv6-dist-domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<p>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.</p> <p><i>Id.</i> at 7:38-46. Thus, Dirks dynamically determines the maximum number of records to sweep/remove by calculating a value <i>k</i>. <i>Id.</i> at 7:15-46, 7:66-8:56.</p> <p>As both NRL IPv6 and Dirks relate to deletion of aged records, 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 NRL IPv6. 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 NRL IPv6 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>examine during each step of the sweeping process with NRL IPv6 and would have seen the benefits of doing so. One possible benefit, for example, is saving the system from performing sometimes time-consuming sweeps.</p> <p>Alternatively, one of ordinary skill in the art would be motivated to, and would understand how to, combine the system disclosed in NRL IPv6 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.</p> <p>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 NRL IPv6 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.</p> <p>Further, one of ordinary skill in the art would be motivated to combine NRL IPv6 with Thatte and recognized the benefits of doing so. For example, the removal of expired records described in NRL IPv6 can be burdensome on the system, adding to the system’s load and slowing down the system’s</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>processing. One of ordinary skill in the art would recognize that combining NRL IPv6 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 NRL IPv6 with Thatte.</p> <p>Alternatively, it would also be obvious to combine NRL IPv6 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:</p> <p style="padding-left: 40px;">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”).</p> <p>In times of heavy use, when deletions must be done rapidly and</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>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. <i>Id.</i> at 2:35-41.</p> <p>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. <i>Id.</i> at 2:42-46.</p> <p>This hybrid deletion is shown in Figure 5.</p>

EXHIBIT D-11

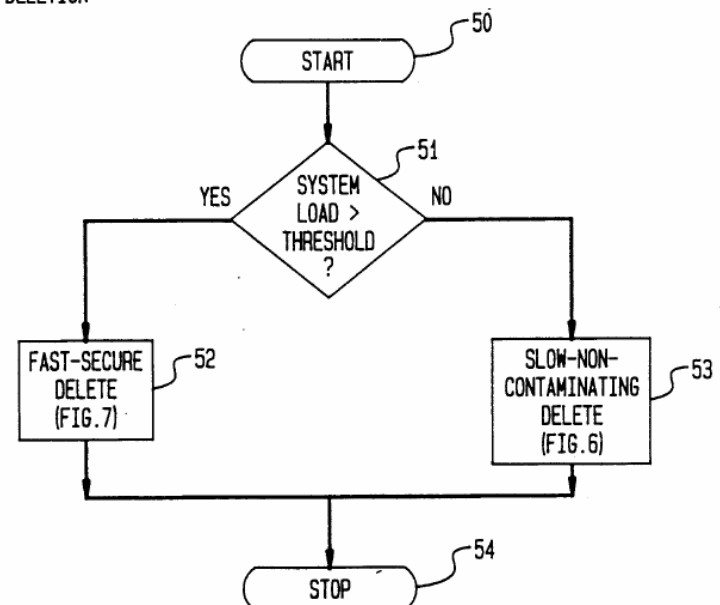
<p>Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p>Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter "NRL IPv6")¹</p>
	<p>FIG. 5 HYBRID DELETION</p>  <pre>graph TD; 50([START]) --> 51{SYSTEM LOAD > THRESHOLD?}; 51 -- YES --> 52[FAST-SECURE DELETE (FIG. 7)]; 51 -- NO --> 53[SLOW-NON-CONTAMINATING DELETE (FIG. 6)]; 52 --> 54([STOP]); 53 --> 54;</pre> <p><i>Id.</i> at Figure 5.</p> <p>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. <i>Id.</i> at 6:40-64,</p>

EXHIBIT D-11

<p align="center">Asserted Claims From U.S. Pat. No. 5,893,120</p>	<p align="center">Naval Research Laboratories ipv6-dist-domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹</p>
	<p>Figure 5. On the other hand, if the system load is less than the threshold, then a slow-non-contaminating delete 53 is used. <i>Id.</i> The fast-secure delete 52 does not actually delete records, rather it marks records as deleted. <i>Id.</i> at 8:1-33, Figure 7. These records are then actually deleted by a subsequent slow-non-contaminating delete 53. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>Thus, the hybrid deletion procedure in the '663 patent dynamically determines a maximum number of records to remove. <i>See id.</i> 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. <i>Id.</i> 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. <i>Id.</i> at 6:65-7:68, Figures 6, 6A, 6B.</p> <p>As both NRL IPv6 and the '663 patent relate to deletion of records from an information storage system, one of ordinary skill in the art would understand how to use the '663 patent's dynamic decision on whether to perform a deletion based on a systems load in other information storage system implementations such as NRL IPv6. 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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p>combining the '663 patent's deletion decision procedure with NRL IPv6 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 NRL IPv6 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.</p> <p>Alternatively, it would also be obvious to combine NRL IPv6 with the Opportunistic Garbage Collection Articles.</p> <p>The Opportunistic Garbage Collection Articles disclose a generational based garbage collection which dynamically determines how much garbage to collect. <i>See generally</i>, Paul R. Wilson and Thomas G. Moher, <i>Design of the Opportunistic Garbage Collector</i>, OOPSLA '89 Proceedings, October 1-6, 1989; Paul R. Wilson, <i>Opportunistic Garbage Collection</i>, ACM SIGPLAN Notices, Vol. 23, No. 12, December 1988.</p> <p>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. <i>Design of the Opportunistic Garbage</i></p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120		Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
		<p><i>Collector</i> at 32.</p> <p>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. <i>Opportunistic Garbage Collection</i> at 100.</p> <p>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. <i>Id.</i></p> <p>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. <i>Design of the Opportunistic Garbage Collector</i> at 32.</p> <p>As both NRL IPv6 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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>on whether to perform a deletion based on a system load in other hash table implementations such as NRL IPv6. 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 NRL IPv6 would be nothing more than the predictable use of prior art elements according to their established functions.</p> <p>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 NRL IPv6 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.</p> <p>Additionally, it would have been obvious to one of ordinary skill in the art to modify the system disclosed in NRL IPv6 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</p>

EXHIBIT D-11

Asserted Claims From U.S. Pat. No. 5,893,120	Naval Research Laboratories ipv6-dist- domestic\sys.common\netinet6\key.c and key.h (hereinafter “NRL IPv6”)¹
	<p>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 NRL IPv6 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 NRL IPv6 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.</p> <p>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.</p> <p><i>See also</i>, key.c at lines 1396-1563, 1768-1845; key.h at lines 188-194.</p>

EXHIBIT E

ADDITIONAL PRIOR ART

I. ADDITIONAL PRIOR ART PUBLICATIONS

Author, Title, Publisher, (Publication Information, Date of Publication).
Moshe Augenstein and Aaron Tennebaum, Data Structures and p/I Programming 536-542, 550-555, 585-604 (Prentice-Hall 1979) (QA76.9.D35 A93)
Robert J. Baron and Linda G. Shapiro, Data Structures and their Implementation 239-253, 303-314 (Von Nostrand Reinhold 1980) (QA76.9.D35 B37)
A.T. Berztiss, Data Structures Theory and Practice 316-319, 329-339 (Academic Press 1971) (QA76.6.B745)
A.T. Berztiss, Data Structures Theory and Practice 416-420, 431-460 (Academic Press 2d Edition 1975) (QA 76.6.B475 1975)
David Clark, Van Jacobson, John Romkey, and Howard Salwen, <i>An Analysis of TCP Processing Overhead</i> , IEEE COMMUNICATIONS MAGAZINE, June 1989, at p. 23-29
Luc Devroye, Lecture Notes on Bucket Algorithms 10-16 (Birkhauser Boston 1986) (QA76.9.D35 D48)
I. Ganapathy and R.F. Hobson, <i>GPMS, A general Purpose Memory Management System --- User's Memory --- That is</i> , Proceedings of the Eighth International Conference on APL, 155-165 (1976)
C.C. Gotlieb and L.R. Gotlieb, Data Types and Structures 341-346 (Prentice-Hall 1978) (QA76.9.D35 G67)
Patrick A.V. Hal, Computational Structures an Introduction to Non-numerical Computing 119-134 (Macdonald & Co. 1975) (QA76.9.D35 H34)
Ellis Horowitz and Sartaj Sahni, Fundamentals of Data Structures 456-471 (Computer Science Press 1983) (QA76.9.D35 H67 1983)
A Klinger, K.S. Fu, T.L. Kunii, Data Structures, Computer Graphics, and Pattern recognition 109-114 (Academic Press 1977) (QA 76.9.D35 D37)
Robert L. Kruse, Programming with Data Structures, Pascal Version 499-523 (Prentice-Hall 1989) (QA76.6.K774 1989)
James Richard Low, Automatic Coding: Choice of Data Structures 24,25,32(Birkhauser Verlag

EXHIBIT E

Author, Title, Publisher, (Publication Information, Date of Publication).
Basel 1976) (QA76.9.D35 L68)
Udi Manber, Introduction to Algorithms a Creative Approach 78-83 (Addison-Wesley 1989) (QA 76.9+.D35 M36 1989)
William G. McArthur and J. Winston Crawley, Structuring Data with PASCAL 604-608 (Prentice Hall 1992) (QA76.9.D35 M39 1992)
Edward M. Reingold and Wilfred J. Hansen, Data Structures in Pascal 268-277, 376-414 (Little, Brown 1986) (QA76.9.D35 R443 1986)
Edward M. Reingold and Wilfred J. Hansen, Data Structures 246-253, 332-364 (Little, Brown 1983) (QA76.9.D35 R44 1983)
M.J.R. Shave, Data Structures 94-116 (McGraw Hill 1975) (QA 76.9.D35 S47)
Jean-Paul Tremblay and Paul G. Sorenson, An Introduction to data Structures with Applications 518-524, 563-568, 611-623 (McGraw-Hill 2d Edition 1984) (QA76.9.D35 T73 1984)
Steven Wartik, Boolean Operations p.282-292 and Steven Wartik, Edward Fox, Lenwood Heath, and Qi-fan Chen, Hashing Algorithms p.293-318; both published in Information Retrieval Data Structures & Algorithms, edited by William B. Frakes and Ricardo Baeza-Yates (Prentice Hall 1992) (QA76.9.D35 I543 1992)