Oracle America, Inc. v. Google Inc. Doc. 79 Att. 1

MORRISON & FOERSTER LLP
MICHAEL A. JACOBS (Bar No. 111664)
mjacobs@mofo.com
MARC DAVID PETERS (Bar No. 211725)
mdpeters@mofo.com
755 Page Mill Road
Palo Alto, CA 94304-1018
Telephone: (650) 813-5600 / Facsimile: (650) 494-0792

BOIES, SCHILLER & FLEXNER LLP
DAVID BOIES (Admitted *Pro Hac Vice*)
dboies@bsfllp.com
333 Main Street
Armonk, NY 10504
Telephone: (914) 749-8200 / Facsimile: (914) 749-8300
STEVEN C. HOLTZMAN (Bar No. 144177)
sholtzman@bsfllp.com
1999 Harrison St., Suite 900
Oakland, CA 94612
Telephone: (510) 874-1000 / Facsimile: (510) 874-1460

ORACLE CORPORATION
DORIAN DALEY (Bar No. 129049)
dorian.daley@oracle.com
DEBORAH K. MILLER (Bar No. 95527)
deborah.miller@oracle.com
MATTHEW M. SARBORARIA (Bar No. 211600)
matthew.sarboraria@oracle.com
500 Oracle Parkway
Redwood City, CA 94065
Telephone: (650) 506-5200 / Facsimile: (650) 506-7114

Attorneys for Plaintiff
ORACLE AMERICA, INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE, INC. Defendant.	Case No. 3:10-cv-03561-WHA **ORACLE AMERICA, INC.'S PATENT LOCAL RULE 3-1 DISCLOSURE OF ASSERTED CLAIMS AND PRELIMINARY INFRINGEMENT CONTENTIONS**

Dockets.Justia.com

Pursuant to Patent Local Rules 3-1 and 3-2, Plaintiff Oracle America, Inc. ("Oracle") hereby submits the following Disclosure of Asserted Claims and Infringement Contentions.

Fact discovery commenced today, and Oracle is serving initial discovery requests on Defendant Google Inc. ("Google") seeking information that may affect Oracle's infringement contentions. In addition, depositions that are directly relevant to Oracle's claims of infringement will be scheduled for after the date of this statement. Not all information about the various versions of the Accused Instrumentalities is publicly available. Further still, Oracle understands that Google may release future versions of the Accused Instrumentalities.[1]

As such, Oracle's investigation into the extent of infringement by Google is ongoing, and Oracle makes these disclosures based on present knowledge of Google's infringing activities. In light of the foregoing, Oracle reserves the right to supplement or amend these disclosures as further facts are revealed during the course of this litigation.

I. DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS.

A. Patent Local Rule 3-1(a) — Asserted Claims.

Oracle asserts that Defendant Google is liable under Title 35 U.S.C. § 271(a), (b), (c), and (f) for infringement of:

- Claims 11-41 of United States Patent No. RE38,104 ("the '104 reissue patent") (infringement claim chart attached as Exhibit A);
- Claims 1, 2, 3, 4, and 8 of United States Patent No. 6,910,205 ("the '205 patent") (infringement claim charts attached as Exhibits B-1 and Exhibit B-2);
- Claims 1, 5-7, 11-13, 15, and 16 of United States Patent No. 5,966,702 ("the '702 patent") (infringement claim chart attached as Exhibit C);
- Claims 1-24 of United States Patent No. 6,125,447 ("the '447 patent") (infringement claim chart attached as Exhibit D);

---

[1] *See, e.g.*, http://en.wikipedia.org/wiki/Android_(operating_system) (last visited Nov. 20, 2010) (Android versions "Honeycomb" and "Ice Cream" scheduled for 2011 launches).

- Claims 1-21 of United States Patent No. 6,192,476 ("the '476 patent") (infringement claim chart attached as Exhibit E);
- Claims 1-4 and 6-23 of United States Patent No. 6,061,520 ("the '520 patent") (infringement claim chart attached as Exhibit F); and
- Claims 1-8, 10-17, and 19-22 of United States Patent No. 7,426,720 ("the '720 patent") (infringement claim chart attached as Exhibit G).

B. Patent Local Rule 3-1(b) — Accused Instrumentalities.

Based on Oracle's investigation thus far, Oracle accuses the following Accused Instrumentalities of infringing each of asserted claims specified above: (i) "Android" or "the Android Platform";[2] (ii) Google devices running Android; and (iii) other mobile devices running Android. Representative examples of Google devices running Android include the Google Nexus One and the Google Nexus S.[3] Representative examples of other mobile devices running Android include HTC's EVO 4G, HTC's Droid Incredible, HTC's G2, Motorola's Droid, and Samsung's Captivate.

Google directly infringes the asserted claims enumerated above under 35 U.S.C. § 271(a) because Google, without authority, makes, uses, offers to sell, sells, or imports the Accused Instrumentalities within or into the United States. Further, Google induces the infringement of others under 35 U.S.C. § 271(b) to the extent it contracts, instructs, or otherwise induces others to make, use, offer to sell, sell, or import the Accused Instrumentalities within or into the United

---

[2] "Android" or "the Android Platform" means "Android" as referred to in Google's Answer (Docket No. 32) at Background ¶ 12 and in Google's Answer to Amended Complaint (Docket No. 51) at Background ¶ 12 and at Factual Background ¶¶ 11-17, and includes any versions thereof (whether released or unreleased) and related public or proprietary source code, executable code, and documentation.

[3] *See, e.g.*, JR Raphael, "The Nexus S and Google: Everything There Is To Know," featured in PCWorld (Nov. 11, 2010), available at http://www.pcworld.com/article/210460/the_nexus_s_and_google_everything_there_is_to_know.html (last visited Nov. 29, 2010) ("Today's buzz is all about the Samsung Nexus S -- a still-under-wraps smartphone believed to be the successor to Google's Nexus One. According to various leaks, the Nexus S will be a 'Google experience' device, meaning it'll run a stock version of Android without any of those baked-in manufacturer UIs. And, if the latest rumors prove to be true, the Samsung Nexus S will be rocking the as-of-yet-unannounced Android Gingerbread release.").

States. Google also contributes to the infringement of others under 35 U.S.C. § 271(c) to the extent it offers to sell, sells, or imports part or all of the Accused Instrumentalities within or into the United States. Further, Google supplies part or all of the Accused Instrumentalities in or from the United States to foreign contractors, including HTC, in violation of 35 U.S.C. § 271(f).

C. Patent Local Rule 3-1(c) — Claim Charts for the Accused Instrumentalities.

Attached as Exhibits A-G are claim charts that identify where each element of each asserted claim of the asserted patents is found within the Accused Instrumentalities, based on the information available to Oracle.

The infringement evidence cited in Exhibits A-G is exemplary and not exhaustive. The cited examples are taken from Android 2.2[4] and current versions of Google's Android websites. Oracle's infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Past releases include the Android SDK Preview, 0.9 beta, 1.0, 1.1, 1.5 ("Cupcake"), 1.6 ("Donut"), 2.0/2.1 ("Éclair"), and 2.2 ("Froyo").

Although Oracle's investigation is ongoing, the following summary indicates which versions of Android infringe the asserted claims of the specified patents:[5]

- the '104 reissue patent (infringement claim chart attached as Exhibit A): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 ("Cupcake"), 1.6 ("Donut"), 2.0/2.1 ("Éclair"), and 2.2 ("Froyo");
- the '205 patent (infringement claim chart attached as Exhibit B-1): infringed by all versions of Android subsequent to January 28, 2010, including at least Android 2.2 ("Froyo");

---

[4] Accessed through http://android.git.kernel.org/.

[5] It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the following list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

- the ’205 patent (infringement claim chart attached as Exhibit B-2): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”);
- the ’702 patent (infringement claim chart attached as Exhibit C): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”);
- the ’447 patent (infringement claim chart attached as Exhibit D): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”);
- the ’476 patent (infringement claim chart attached as Exhibit E): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”);
- the ’520 patent (infringement claim chart attached as Exhibit F): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”); and
- the ’720 patent (infringement claim chart attached as Exhibit G): infringed by all versions of Android subsequent to Oct. 21, 2008, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

D. Patent Local Rule 3-1(d) — Indirect Infringement.

In addition to the acts of direct infringement described above, Google actively contributes to and induces infringement by third parties of each of the asserted claims of the asserted patents. On information and belief, Google purposely and actively distributes the Accused Instrumentalities to manufacturers of products and application developers with the intention that they be used, copied and distributed to consumers. Google induces and contributes to the infringement of the asserted claims of each asserted patent, because Google encourages manufacturers, application developers, and service providers (including the members of the Open Handset Alliance), as well as end users, to copy, sell, distribute, re-distribute, and use products that embody or incorporate the Accused Instrumentalities. Google’s admissions in its Amended

Counterclaims prove its intent and encouragement of others. (*See, e.g.*, Google's Amended Counterclaims ¶¶ 6-7, 13.) As discussed below, Google has actual knowledge of Oracle's patents and its infringement is willful.

E. Patent Local Rule 3-1(e) — Nature of Infringement.

Oracle asserts that each element or limitation of each asserted claim of each asserted patent is literally present in the Accused Instrumentalities, except where explicitly indicated. To the extent that any element or limitation of the asserted claims is not found to have literal correspondence in the Accused Instrumentalities, Oracle alleges, on information and belief, that any such elements or limitations are present under the doctrine of equivalents in the Accused Instrumentalities.

F. Patent Local Rule 3.1(f) — Priority Dates.

The '104 reissue patent has a priority date of Dec. 22, 1992, being a continuation of 08/755,764 (filed Nov. 21, 1996) resulting in RE36,204 which is a Reissue of 07/994,655 (filed Dec. 22, 1992) which is U.S. Patent No. 5,367,685.

The '205 patent is a continuation of U.S. Pat. No. 6,513,156, having a priority date of Jun. 30, 1997, the filing date of U.S. patent application number 08/884,856.

G. Patent Local Rule 3.1(g) — Patentee's Asserted Practice of the Claimed Inventions.[6]

1. The '104 Reissue Patent

The following instrumentalities of Oracle practice the asserted claims of the '104 reissue patent:

- JDK 1.0 and subsequent versions;
- JRE 1.1.1 and subsequent versions;
- HotSpot 1.0 and subsequent versions;

[6] Oracle's investigation concerning the identification of instrumentalities that practice the asserted claims of the asserted patents is ongoing. There have been many different products relating to the Java Platform over the years, each having many versions or variants, and the lists presented below reflect Oracle's diligent efforts in identifying instrumentalities that practice the asserted claims of the asserted patents.

- Java SE for Embedded 1.4.2_11 and subsequent versions;
- CDC RI 1.0 and CDC-HI 1.0 and subsequent versions of each;
- CDC AMS 1.0, 1.0_1, 1.0_2, Personal Basis and Personal Profile versions;
- CLDC RI 1.0 and CLDC-HI 1.0 and subsequent versions;
- Foundation Profile 1.0 and subsequent versions;
- J2EE 1.2 (later called Java EE) and subsequent versions;
- WTK 1.0 / Java ME SDK 1.0, and subsequent versions of each;
- Java Real Time 1.0 and all subsequent versions;
- Personal Profile HI and RI 1.0 and subsequent versions;
- Personal Basis Profile-HI and RI 1.0 and subsequent versions;
- PersonalJava 1.0 and subsequent versions;
- EmbeddedJava 1.0 and subsequent versions;
- JavaOS 1.0 (all variants, including Java PC) and subsequent versions;
- Java Card connected platform 3.0 and subsequent versions;
- Oracle Java Wireless Client (formerly Sun Java Wireless Client) 1.0 and subsequent versions;
- MIDP 1.0 and subsequent versions; and
- Jrockit[7] from 2002 and subsequent versions.

2. The ’205 Patent

The following instrumentalities of Oracle practice the asserted claims of the ’205 patent:

- JDK 1.2 and subsequent versions;
- JRE 1.2 and subsequent versions;
- HotSpot 1.0 and subsequent versions;
- Java SE for Embedded 1.4.2 and subsequent versions;
- CDC RI 1.0.1 and CDC-HI 1.0 and subsequent versions of each;

[7] Oracle International Corporation, not Oracle America, owns Jrockit through Oracle Corporation’s acquisition of BEA Systems.

- CDC AMS 1.0, 1.0_1, 1.0_2, Personal Basis and Personal Profile versions;
- CLDC RI 1.1.1;
- CLDC-HI 1.0 and subsequent versions;
- Foundation Profile 1.0.2 and subsequent versions;
- J2EE 1.2 (later called Java EE) and subsequent versions;
- Java ME SDK 3.0 EA and subsequent versions;
- Java Real-Time System 1.0 and all subsequent versions;
- Personal Profile HI and RI 1.0 and subsequent versions;
- Personal Basis Profile HI and RI 1.0 and subsequent versions; and
- Jrockit from 2002 and subsequent versions.

3. The ’702 Patent

The following instrumentalities of Oracle practice the asserted claims of the ’702 patent:

- PersonalJava (“PJava”) 1.0 and subsequent versions;
- EmbeddedJava (“EJava”) 1.0 and subsequent versions;
- JavaOS 1.0 (and all variants, including Java PC) and subsequent versions;
- CDC RI 1.0 and CDC-HI 1.0, and all subsequent versions of each;
- CDC AMS 1.0, 1.0_1, 1.0_2, Personal Basis and Personal Profile versions;
- CLDC RI 1.1.1 and CLDC-HI 1.0.1, and all subsequent versions of each;
- Personal Profile HI and RI 1.0 and subsequent versions;
- Personal Basis Profile HI and RI 1.0 and subsequent versions;
- Foundation Profile 1.0 and subsequent versions; and
- Java Card platform 2.1 and subsequent versions.

4. The ’447 and ’476 Patents

The following instrumentalities of Oracle practice the asserted claims of the ’447 and ’446 patents:

- JDK 1.2 and subsequent versions;
- JRE 1.2 and subsequent versions;
- Java SE for Embedded 1.4.2_11 and subsequent versions;

- CDC RI 1.0 and CDC-HI 1.0, and all subsequent versions of each;
- CDC AMS 1.0, 1.0_1, 1.0_2, Personal Basis and Personal Profile versions;
- Foundation Profile 1.0.2 and subsequent versions;
- J2EE 1.2 (later called Java EE) and subsequent versions;
- Java ME SDK 3.0 EA and subsequent versions;
- Java Real-Time System 1.0 and all subsequent versions;
- Personal Profile HI and RI 1.0 and subsequent versions;
- Personal Basis Profile HI and RI 1.0 and subsequent versions;
- Java Card connected platform 3.0 and subsequent versions; and
- Jrockit from 2002 and subsequent versions.

Additionally, the following instrumentalities of Oracle practice the asserted claims of the ’447 patent:

- Oracle Java Wireless Client (formerly Sun Java Wireless Client) 1.1.3 and subsequent versions.

5. The ’520 Patent

The following instrumentalities of Oracle practice the asserted claims of the ’520 patent:

- CLDC RI 1.1.1;
- Java Card platform 2.1 and subsequent versions; and
- CLDC-HI 1.1.3 and subsequent versions.

6. The ’720 Patent

The following instrumentalities of Oracle practice the asserted claims of the ’720 patent:

- CDC AMS 1.0, 1.0_1, 1.0_2, Personal Basis and Personal Profile versions.

H. Patent Local Rule 3-1(h) — Willful Infringement.

Google has willfully infringed the patents-in-suit, which are directed to inventions incorporated in the Java Platform. Many factors reveal that Google acted recklessly, *i.e.*, despite a high likelihood that Google’s actions infringed a valid and enforceable patent, and that Google actually knew or should have known that its actions constituted an unjustifiably high risk of infringement of a valid and enforceable patent. These factors include:

- Google is a member of the Java Community Process (JCP) and has a seat on the Java SE/EE Executive Committee. *See* Java Community Process homepage, available at http://www.jcp.org/en/participation/committee (last visited Dec. 1, 2010). Through its lengthy participation in the JCP, Google is well aware of the need to obtain a license from Oracle in order to make use of Oracle's Java Platform technologies as Google does in Android. Google's admissions in its Amended Counterclaims prove this awareness. (*See, e.g.,* Google's Amended Counterclaims ¶¶ 6-7, 13.)
- At least three of the seven inventors named in the patents-in-suit, Robert Griesemer, Lars Bak, and Frank Yellin, have left Oracle and work at Google. Their knowledge is attributable to Google.
- Andy Rubin, Google's VP of Mobile Platforms, previously worked at Danger, Inc., which he founded. He understood the need to obtain a license from Oracle (then Sun) to use Java Platform technologies in Danger's Hiptop operating system, and Danger did obtain a commercial license. When Rubin left Danger and founded Android, Inc., he approached Sun about obtaining a commercial license to Java Platform technologies on behalf of Android, Inc. Those discussions ended without Android having obtained a commercial license. Rubin's knowledge is attributable to Google.
- Google has consistently resisted taking a license from Sun for Sun's patented Java Platform technologies.
- In copying Oracle's Java Platform technologies, Google deliberately disregarded a known risk that Oracle had protective patents covering Java Platform technologies.
- Google's Android source code and documentation directly references and copies Java Platform technology specifications, documentation, and source code. *See, e.g.,* mydroid\libcore\security\src\main\java\java\security\CodeSource.java; mydroid\libcore\support\src\test\java\org\apache\harmony\security\tests\support\cert\PoicyNodeImpl.java. Google admits that Android incorporates a subset of Apache Harmony, which it asserts is "an implementation of Sun's Java." (*See, e.g.,* Google's Amended Counterclaims ¶¶ 6-7, 13.)

- Google's website content directly references and demonstrates use of Java Platform technologies. *See, e.g.*, "What is Android?", available at http://developer.android.com/guide/basics/what-is-android.html (last visited Dec. 1, 2010) ("Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language."); Package Index, available at http://developer.android.com/reference/packages.html (last visited Dec. 1, 2010), and subsidiary webpages.
- Google's Android videos directly reference and demonstrate use of Java Platform technologies. *See, e.g.*, Google I/O 2008 Video entitled "Dalvik Virtual Machine Internals," presented by Dan Bornstein (Google), available at http://developer.android.com/videos/index.html#v=ptjedOZEXPM (last visited Dec. 1, 2010).

II. DOCUMENT PRODUCTION ACCOMPANYING DISCLOSURES.[8]

A. Patent Local Rule 3-2(a) — Documents Evidencing Pre-Application Disclosure.[9]

Copies of documents produced pursuant to Patent Local Rule 3-2(a) are at OAGOOGLE0000052860-53265, OAGOOGLE0000053266 -53749, OAGOOGLE0000053750-53759, OAGOOGLE0000059578, and OAGOOGLE0000059579-60385. Oracle also directs Google to three public websites: developer.sun.com, java.sun.com, and www.sun.com. Oracle's proprietary commercial releases will be made available for inspection subject to a Protective Order entered in this case or by agreement of the parties.

---

[8] Once the Parties agree to a protective order governing the production of source code in this litigation, Oracle will make available source code pursuant to Patent Local Rule 3-2 for inspection by Google in accordance with the anticipated protective order. Where different versions of specific Oracle source code do not vary with respect to the claimed inventions in suit (including variants and customized versions for specific customers), Oracle will produce the earliest general version practicing the claimed invention to avoid or minimize any duplicative productions.

[9] As Patent Local Rule 3-2(a) states, Oracle's production of a document as required by the rule shall not constitute an admission that such document evidences or is prior art under 35 U.S.C. § 102.

B. Patent Local Rule 3-2(b) — Documents Evidencing Conception and Reduction to Practice.

Copies of documents evidencing conception, reduction to practice, design and development of the claimed inventions are produced at OAGOOGLE0000000001-52022, OAGOOGLE0000053793-57166, and OAGOOGLE0000059571-59577. Oracle also directs Google to three public websites: developer.sun.com, java.sun.com, and www.sun.com. Oracle's proprietary commercial releases will be made available for inspection subject to a Protective Order entered in this case or by agreement of the parties.

C. Patent Local Rule 3-2(c) — File Histories for the Patents-in-Suit.

Copies of the patent file histories are produced at OAGOOGLE0000052023-52859 and OAGOOGLE0000057167-59570.

D. Patent Local Rule 3-2(d) — Ownership of the Patents-in-Suit.

Copies of documents evidencing ownership of the patent rights are produced at OAGOOGLE0000053760-53792 and OAGOOGLE0000056022- 56028.

E. Patent Local Rule 3-2(e) — Patentee's Asserted Practice of the Claimed Inventions.

Copies of documents sufficient to show the operation of any aspects or elements of instrumentalities Oracle relies upon as embodying the asserted claims can be found at the following three public websites: developer.sun.com, java.sun.com, and www.sun.com. Oracle's proprietary commercial releases will be made available for inspection subject to a Protective Order entered in this case or by agreement of the parties.

Dated: December 2, 2010

MICHAEL A. JACOBS
MARC DAVID PETERS
MORRISON & FOERSTER LLP

By: /s/ Marc David Peters

Attorneys for Plaintiff
ORACLE AMERICA, INC.

CERTIFICATE OF SERVICE

I declare that I am employed with the law firm of Morrison & Foerster LLP, whose address is 755 Page Mill Road, Palo Alto, California 94304-1018. I am not a party to the within cause, and I am over the age of eighteen years.

I further declare that on December 2, 2010, I served a copy of:

ORACLE AMERICA, INC.'S PATENT LOCAL RULES 3-1 DISCLOSURE OF ASSERTED CLAIMS AND PRELIMINARY INFRINGEMENT CONTENTIONS

BY ELECTRONIC SERVICE [Fed. Rule Civ. Proc. rule 5(b)] by electronically mailing a true and correct copy through Morrison & Foerster LLP's electronic mail system to the e-mail address(es) set forth below, or as stated on the attached service list per agreement in accordance with Federal Rules of Civil Procedure rule 5(b).

Robert F. Perry
Scott T. Weingaertner
Bruce W. Baber
KING & SPALDING LLP
1185 Avenue of the Americas
New York, NY 10036-4003

RPerry@kslaw.com
SWeingaertner@kslaw.com

Fax: 212.556.2222

Donald F. Zimmer, Jr.
Cheryl Z. Sabnis
KING & SPALDING LLP
101 Second Street, Suite 2300
San Francisco, CA 94105

fzimmer@kslaw.com
csabnis@kslaw.com

Fax: 415.318.1300

Joseph R. Wetzel
GREENBERG TRAURIG LLP
153 Townsend Street, 8th Floor
San Francisco, CA 94107

wetzelj@gtlaw.com

Fax: 415.707.2010

Timothy T. Scott
Geoffrey M. Ezgar
Leo Spooner III
KING & SPALDING, LLP
333 Twin Dolphin Drive, Suite 400
Redwood Shores, CA 94065

TScott@kslaw.com
GEzgar@kslaw.com
LSpooner@kslaw.com

Fax: 650.590.1900

Ian C. Ballon
Heather Meeker (*App for Admission to ND Cal to be filed*)
GREENBERG TRAURIG LLP
1900 University Avenue
East Palo Alto, CA 94303

ballon@gtlaw.com
meekerh@gtlaw.com

Fax: 650.328.8508

I declare under penalty of perjury that the foregoing is true and correct.

Executed at Palo Alto, California, this 2nd day of December, 2010.

Richard S. Ballinger	/s/ Richard S. Ballinger
(typed)	(signature)

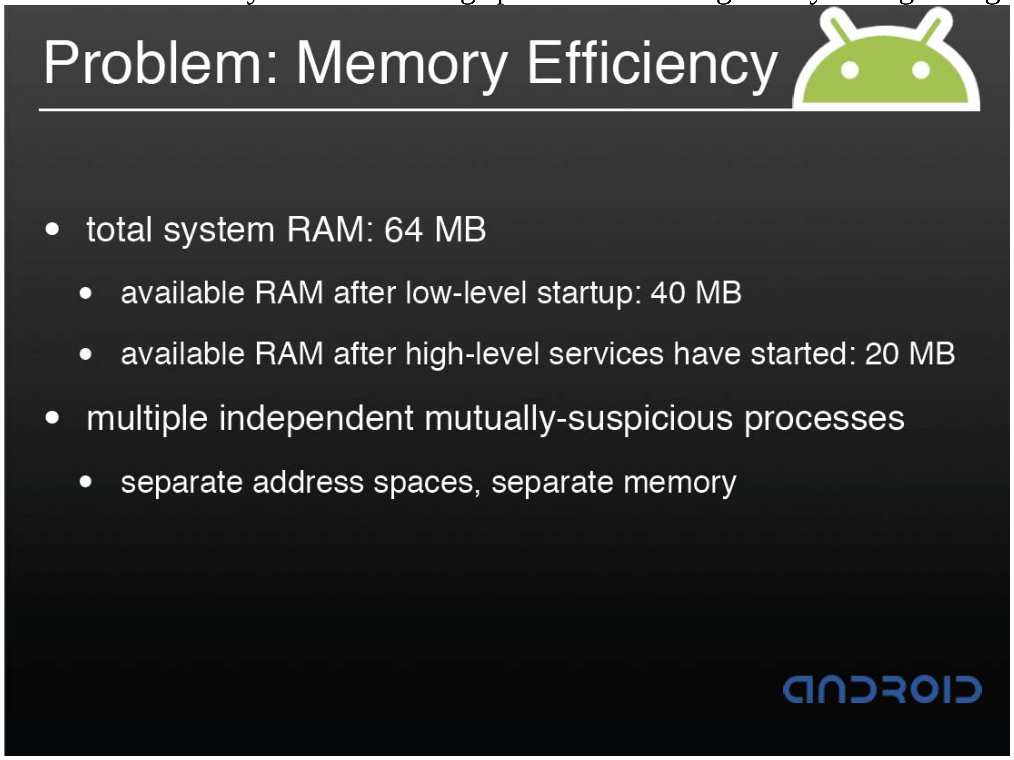
EXHIBIT A
Preliminary Infringement Contentions for the ’104 Reissue Patent

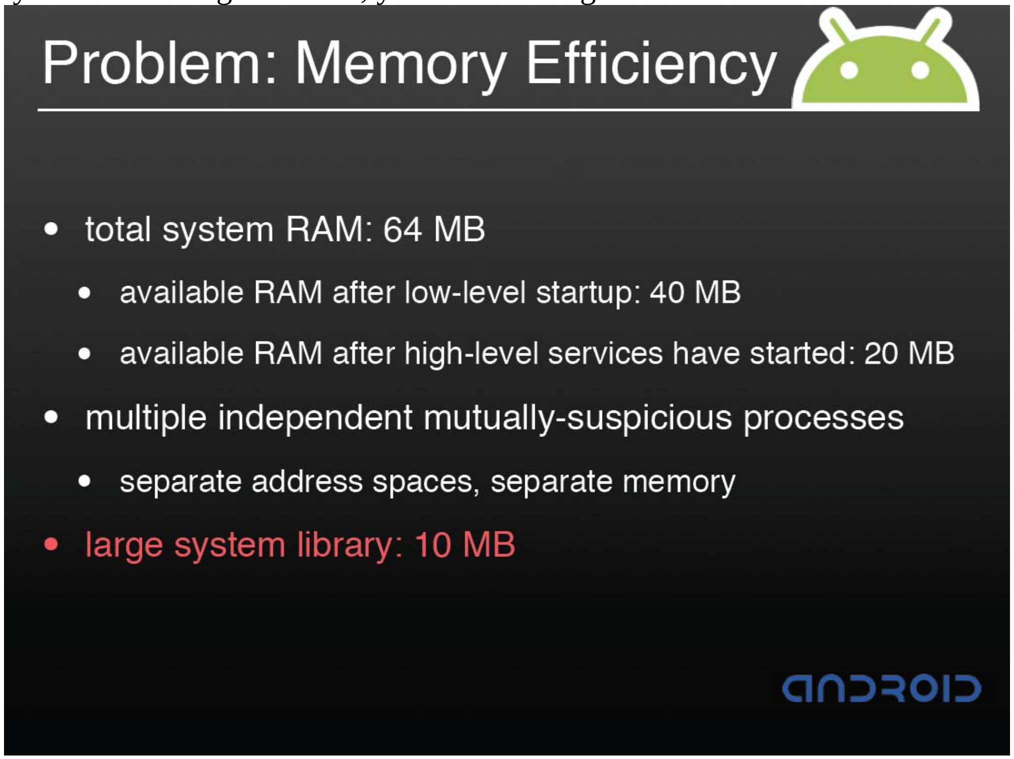
NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’104 reissue patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

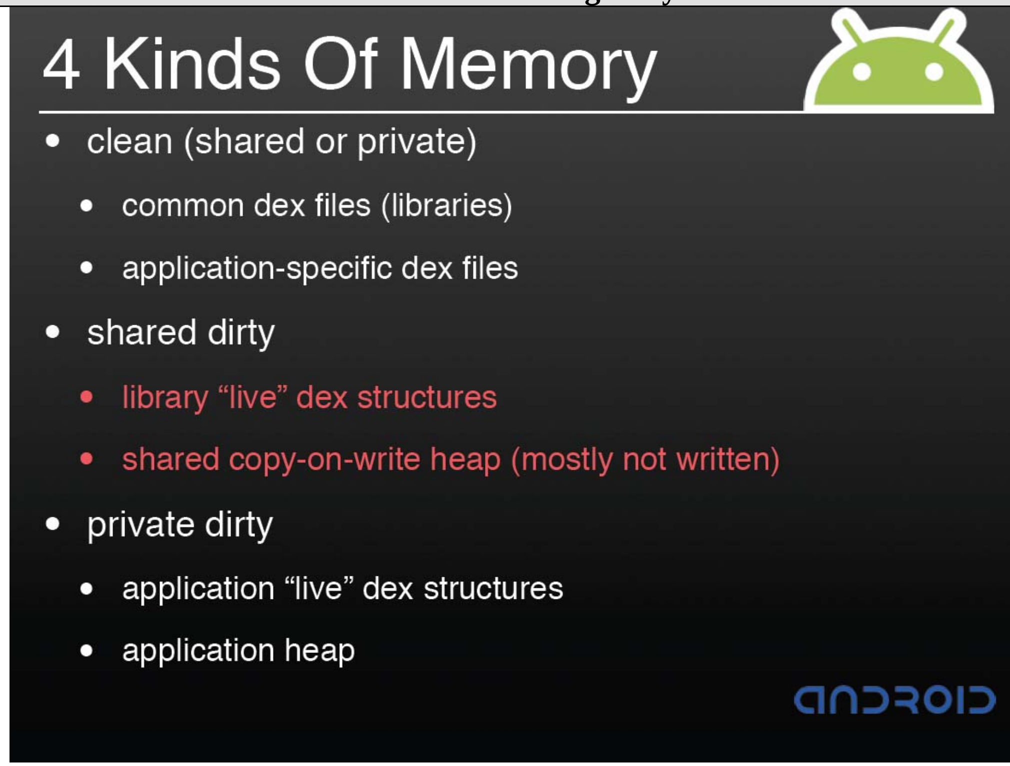
It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The ’104 Reissue Patent	Infringed By
[11-preamble] 11. An apparatus comprising:	The Accused Instrumentalities include devices that run Android. An Android-based device is an apparatus.
[11-a] a memory containing intermediate form object code constituted by a set of instructions, certain of said instructions containing one or more symbolic references;	An Android-based device has a memory containing intermediate form object code constituted by a set of instructions. *See, e.g.*, Google I/O 2008 Video, Google I/O 2008 Video, entitled “Dalvik Virtual Machine Internals,” presented by Dan Bornstein (Google Android Project), available at http://developer.android.com/videos/index.html#v=ptjedOZEXPM: • at 1:22 under “The Big Picture” (“Very briefly, Android is the new platform for mobile devices and it really is the complete stack, includes layers from the OS kernel at the bottom and drivers up through an application framework at the top and it even includes a few applications. You write your applications in the Java programming language and they get translated after compilation into a form that runs on the Dalvik virtual machine.”). • at 2:52 under “What is the Dalvik VM?” (“So the virtual machine, again, is designed based on the constraints of the platform and you can see a few of the key ones. We’re assuming, not a

The ’104 Reissue Patent	Infringed By
	particularly powerful CPU, not very much RAM especially by say today’s desktop standards. An easy way to think about it is as approximately equivalent to like a late 90s desktop machine with a little more modern operating system, but with one very important constraint.”). • at 4:06 under “Problem: Memory Efficiency” (“So, in particular this is, this is kind of how a low end Android device is gonna look in terms of, you know, system characteristics. So, you know, once everything is started up on the system we’re not really expecting there to be that much memory left for applications and, of course, so we try to make the most of that. But one wrinkle in the works is that our, the Android platform security relies on modern process separation. So each application is running in a separate process. There’s a separate address space. It has separate memory and apps are not allowed to interfere with each other at that level and so that means that unless you do something special that 20 megs really isn’t gonna go far at all.”). 

The ’104 Reissue Patent	Infringed By
	• at 5:05 under “Problem: Memory Efficiency” (“And in addition to this modern platform that, we try to make it, you know, have a rich, have a rich set of APIs for developers to use, we have a fairly large system library. And so again, if you don’t do anything special, well, with a 10 meg library, 20 megs left for apps, that really, really doesn’t leave much space at all. And I think I had a previous slide, we don’t have swap space. So I just wanna emphasize that, so there’s no, if you have 64 megs of RAM, you have 64 megs of RAM and that’s kind of the size of it. Okay.”).  • at 15:38 under “4 Kinds of Memory” (“So our goal, again, is to get as much, as much memory to be mapped clean as possible, but we at least have this out for where we really do have to allocate that we can reduce the cost in terms of the whole system performance.”).

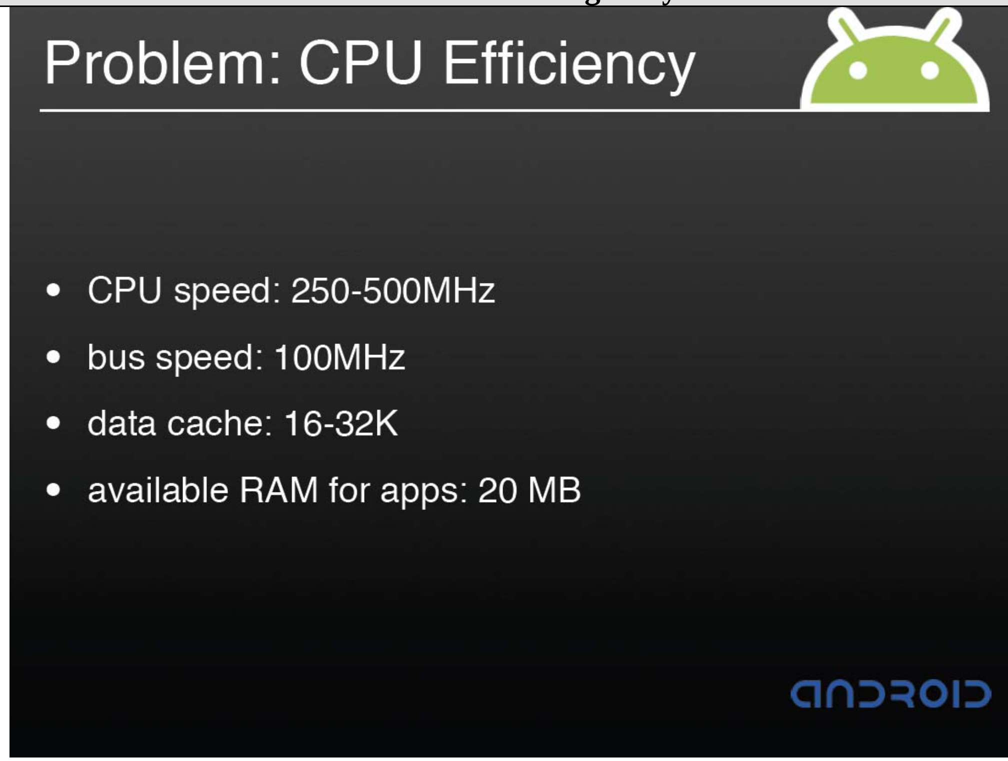
<table>
<tr><th>The ’104 Reissue Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



- at 19:07 under “Problem: CPU Efficiency” (“Again, as I said at the beginning, we’re running on a platform or expecting to run on a platform that looks like what you might have had on your desktop 10 years ago. And, you know, you can see that it’s a fairly slow bus, almost no data cache at all and I just wanna re-emphasize that there’s very little RAM for an app, for applications once you consider all of the things that your device is doing, say, as a phone. It has to answer phone calls, it has to be able to take and send SMSs. All of these things are essential services as far as the user is concerned.”).

</td></tr>
</table>

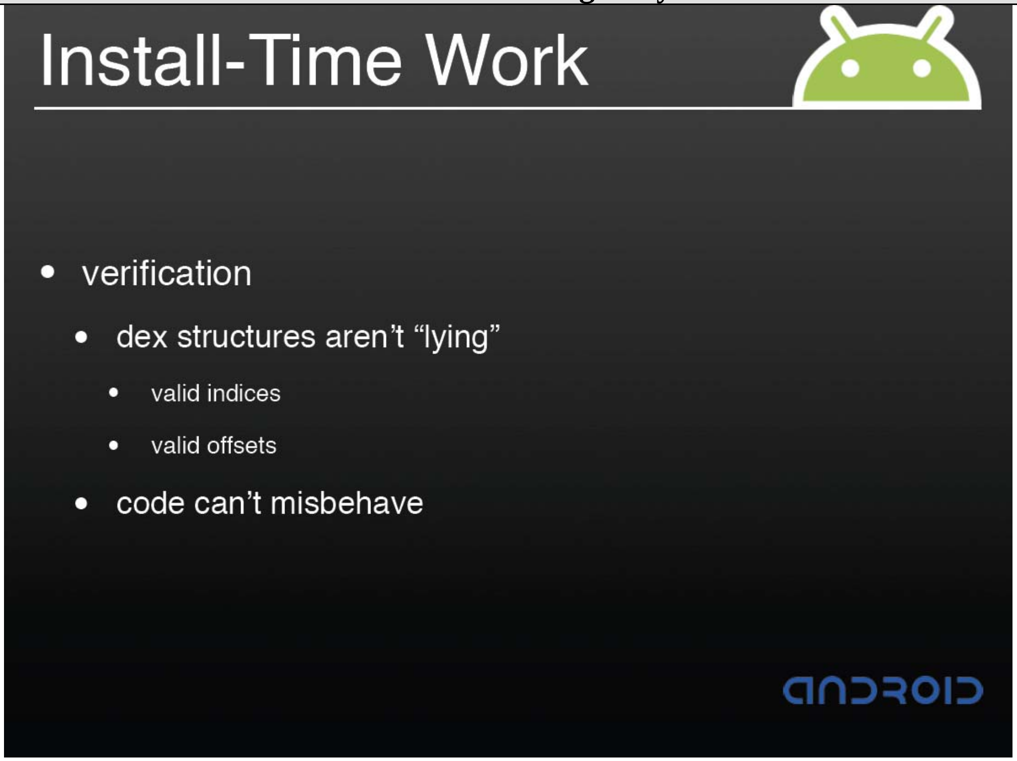
<table>
<tr><th>The ’104 Reissue Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



- at 21:54 under “Install-Time Work” (“So, what are we doing to actually be efficient on the platform? So, first of all, when an application gets installed and also when the system itself gets installed, the platform will, the system will do a lot of work up front to avoid doing work at runtime. So one of the major things we do is verification of dex files and what this means is that as a, as a type safe, reference safe runtime we want to ensure that the code that we’re running doesn’t violate the constraints of the system. It doesn’t violate type safety, it doesn’t, it doesn’t, it doesn’t violate reference safety. And for Android, this is really more about minimizing the app, the impact of bugs in an application as opposed to being a security consideration in and of itself.”).

</td></tr>
</table>

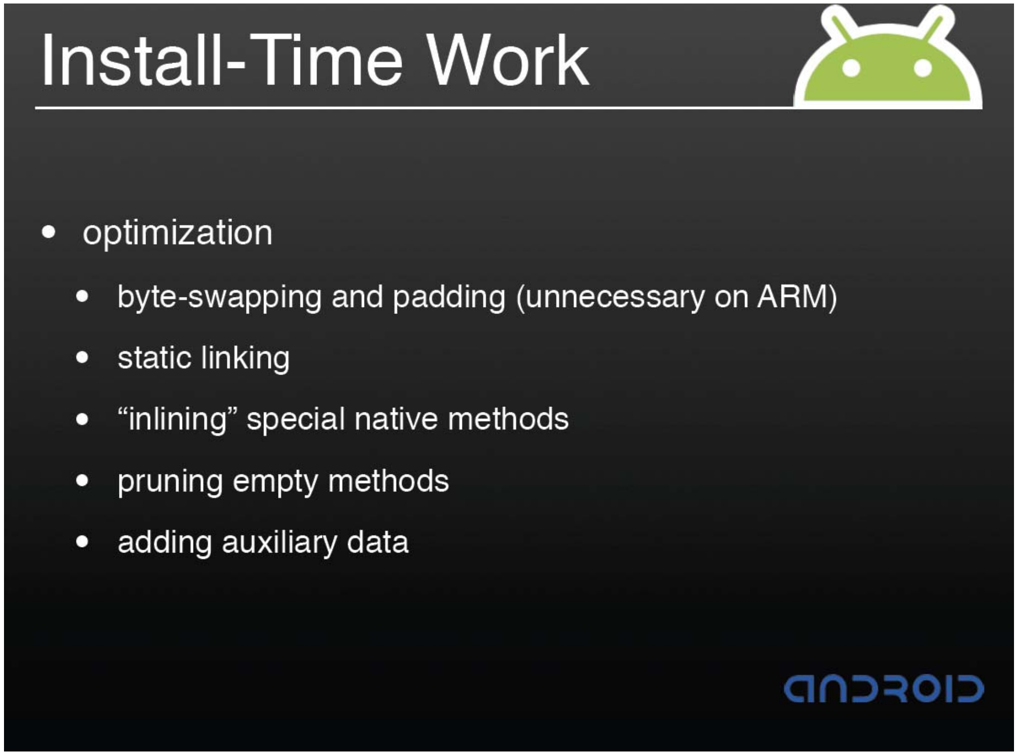
<table>
<tr><th>The ’104 Reissue Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



- at 23:35 under “Install-Time Work” (“We do optimization. And, so the first time that a dex file lands on a device, we do that verification work, we also, we also augment that file, if we have to we will do byte swapping and pad out structures and in addition, we have a bunch of other things that we do such that when it comes time to run, we can run that much faster. So as an example of static linking, before, when a dex files arrives on a device it will have symbolic references to methods and fields, but afterwards it might just be a simple, a simple integer vtable offset so that when, for invoking a method, instead of having to do say a string-based lookup, it can just simply index into a vtable. And just as another example, you are probably aware that the constructor for java.lang.object has nothing, does nothing inside it and the system can tell. So instead of, instead of actually doing that any time you’re constructing an object, we know to avoid just making that call and that actually does make a significant performance impact.”).

</td></tr>
</table>

The ’104 Reissue Patent	Infringed By



Android applications are packaged as .apk files containing intermediate form object code (.dex files). *See, e.g.*, Android Glossary Definition for “.apk file,” available at http://developer.android.com/guide/appendix/glossary.html:

> .apk file
> Android application package file. Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assets, and manifest file. The application package file can have any name but must use the .apk extension. For example: myExampleAppname.apk. For convenience, an application package file is often referred to as an ".apk".

The ’104 Reissue Patent	Infringed By
	Android Glossary Definition for “.dex file,” available at http://developer.android.com/guide/appendix/glossary.html: .dex file Compiled Android application code file. Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the Java programming language. Android Glossary Definition for “Dalvik,” available at http://developer.android.com/guide/appendix/glossary.html: Dalvik The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management). The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device. Android Basics, entitled “What is Android?,” available at http://developer.android.com/guide/basics/what-is-android.html: **What is Android?** Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

The ’104 Reissue Patent	Infringed By
	Features • Application framework enabling reuse and replacement of components • Dalvik virtual machine optimized for mobile devices • Integrated browser based on the open source WebKit engine • Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional) • SQLite for structured data storage • Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF) • GSM Telephony (hardware dependent) • Bluetooth, EDGE, 3G, and WiFi (hardware dependent) • Camera, GPS, compass, and accelerometer (hardware dependent) • Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE **Android Architecture** The following diagram shows the major components of the Android operating system. Each section is described in more detail below.

The ’104 Reissue Patent	Infringed By

Applications

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

…

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the

The ’104 Reissue Patent	Infringed By
	core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Another way that Android and Android-based devices meet the claim limitation is through the dexopt tool. *See, e.g.*, dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: **Dalvik Optimization and Verification With *dexopt*** The Dalvik virtual machine was designed specifically for the Android mobile platform. The target systems have little RAM, store data on slow internal flash memory, and generally have the performance characteristics of decade-old desktop systems. They also run Linux, which provides virtual memory, processes and threads, and UID-based security mechanisms. The features and limitations caused us to focus on certain goals: • Class data, notably bytecode, must be shared between multiple processes to minimize total system memory usage. • The overhead in launching a new app must be minimized to keep the device responsive. • Storing class data in individual files results in a lot of redundancy, especially with respect to strings. To conserve disk space we need to factor this out. • Parsing class data fields adds unnecessary overhead during class loading. Accessing data values (e.g. integers and strings) directly as C types is better.

The ’104 Reissue Patent	Infringed By
	• Bytecode verification is necessary, but slow, so we want to verify as much as possible outside app execution. • Bytecode optimization (quickened instructions, method pruning) is important for speed and battery life. • For security reasons, processes may not edit shared code. The typical VM implementation uncompresses individual classes from a compressed archive and stores them on the heap. This implies a separate copy of each class in every process, and slows application startup because the code must be uncompressed (or at least read off disk in many small pieces). On the other hand, having the bytecode on the local heap makes it easy to rewrite instructions on first use, facilitating a number of different optimizations. The goals led us to make some fundamental decisions: • Multiple classes are aggregated into a single "DEX" file. • DEX files are mapped read-only and shared between processes. • Byte ordering and word alignment are adjusted to suit the local system. • Bytecode verification is mandatory for all classes, but we want to "pre-verify" whatever we can. • Optimizations that require rewriting bytecode must be done ahead of time. • The consequences of these decisions are explained in the following sections. ….
[11-b] and a processor configured to execute said instructions containing one or more symbolic references by determining a numerical reference corresponding to said symbolic reference, storing said	Any device running Android has a processor configured to execute said instructions containing one or more symbolic references by determining a numerical reference corresponding to said symbolic reference, storing said numerical references, and obtaining data in accordance to said numerical references. *See, e.g.*, \dalvik\vm\oo\Resolve.h: /* * Resolve "constant pool" references into pointers to VM structs. */

The ’104 Reissue Patent	Infringed By
numerical references, and obtaining data in accordance to said numerical references.	``` #ifndef _DALVIK_OO_RESOLVE #define _DALVIK_OO_RESOLVE /* * "Direct" and "virtual" methods are stored independently. The type of call * used to invoke the method determines which list we search, and whether * we travel up into superclasses. * * (<clinit>, <init>, and methods declared "private" or "static" are stored * in the "direct" list. All others are stored in the "virtual" list.) */ typedef enum MethodType { METHOD_UNKNOWN = 0, METHOD_DIRECT, // <init>, private METHOD_STATIC, // static METHOD_VIRTUAL, // virtual, super METHOD_INTERFACE // interface } MethodType; /* * Resolve a class, given the referring class and a constant pool index * for the DexTypeId. * * Does not initialize the class. * * Throws an exception and returns NULL on failure. */ ClassObject* dvmResolveClass(const ClassObject* referrer, u4 classIdx, bool fromUnverifiedConstant); /* * Resolve a direct, static, or virtual method. ```

The ’104 Reissue Patent	Infringed By
	`` * * Can cause the method's class to be initialized if methodType is * METHOD_STATIC. * * Throws an exception and returns NULL on failure. */ Method* dvmResolveMethod(const ClassObject* referrer, u4 methodIdx, MethodType methodType); /* * Resolve an interface method. * * Throws an exception and returns NULL on failure. */ Method* dvmResolveInterfaceMethod(const ClassObject* referrer, u4 methodIdx); /* * Resolve an instance field. * * Throws an exception and returns NULL on failure. */ InstField* dvmResolveInstField(const ClassObject* referrer, u4 ifieldIdx); /* * Resolve a static field. * * Causes the field's class to be initialized. * * Throws an exception and returns NULL on failure. */ StaticField* dvmResolveStaticField(const ClassObject* referrer, u4 sfieldIdx); ```

The '104 Reissue Patent	Infringed By
	/* * Resolve a "const-string" reference. * * Throws an exception and returns NULL on failure. */ StringObject* dvmResolveString(const ClassObject* referrer, u4 stringIdx); /* * Return debug string constant for enum. */ const char* dvmMethodTypeStr(MethodType methodType); #endif /*_DALVIK_OO_RESOLVE*/ \dalvik\vm\oo\Resolve.c: /* * Resolve classes, methods, fields, and strings. * * According to the VM spec (v2 5.5), classes may be initialized by use * of the "new", "getstatic", "putstatic", or "invokestatic" instructions. * If we are resolving a static method or static field, we make the * initialization check here. * * (NOTE: the verifier has its own resolve functions, which can be invoked * if a class isn't pre-verified. Those functions must not update the * "resolved stuff" tables for static fields and methods, because they do * not perform initialization.) */ #include "Dalvik.h" #include <stdlib.h>

The ’104 Reissue Patent	Infringed By
	(see code below)

```
/*
 * Find the class corresponding to "classIdx", which maps to a class name
 * string.  It might be in the same DEX file as "referrer", in a different
 * DEX file, generated by a class loader, or generated by the VM (e.g.
 * array classes).
 *
 * Because the DexTypeId is associated with the referring class' DEX file,
 * we may have to resolve the same class more than once if it's referred
 * to from classes in multiple DEX files.  This is a necessary property for
 * DEX files associated with different class loaders.
 *
 * We cache a copy of the lookup in the DexFile's "resolved class" table,
 * so future references to "classIdx" are faster.
 *
 * Note that "referrer" may be in the process of being linked.
 *
 * Traditional VMs might do access checks here, but in Dalvik the class
 * "constant pool" is shared between all classes in the DEX file.  We rely
 * on the verifier to do the checks for us.
 *
 * Does not initialize the class.
 *
 * "fromUnverifiedConstant" should only be set if this call is the direct
 * result of executing a "const-class" or "instance-of" instruction, which
 * use class constants not resolved by the bytecode verifier.
 *
 * Returns NULL with an exception raised on failure.
 */
ClassObject* dvmResolveClass(const ClassObject* referrer, u4 classIdx,
    bool fromUnverifiedConstant)
```

The ’104 Reissue Patent	Infringed By
	`` { DvmDex* pDvmDex = referrer->pDvmDex; ClassObject* resClass; const char* className; /* * Check the table first -- this gets called from the other "resolve" * methods. */ resClass = dvmDexGetResolvedClass(pDvmDex, classIdx); if (resClass != NULL) return resClass; LOGVV("--- resolving class %u (referrer=%s cl=%p)\n", classIdx, referrer->descriptor, referrer->classLoader); /* * Class hasn't been loaded yet, or is in the process of being loaded * and initialized now. Try to get a copy. If we find one, put the * pointer in the DexTypeId. There isn't a race condition here -- * 32-bit writes are guaranteed atomic on all target platforms. Worst * case we have two threads storing the same value. * * If this is an array class, we'll generate it here. */ className = dexStringByTypeIdx(pDvmDex->pDexFile, classIdx); if (className[0] != '\0' && className[1] == '\0') { /* primitive type */ resClass = dvmFindPrimitiveClass(className[0]); } else { resClass = dvmFindClassNoInit(className, referrer->classLoader); } ```

<table>
<tr><th>The ’104 Reissue Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

```
if (resClass != NULL) {
    /*
     * If the referrer was pre-verified, the resolved class must come
     * from the same DEX or from a bootstrap class.  The pre-verifier
     * makes assumptions that could be invalidated by a wacky class
     * loader.  (See the notes at the top of oo/Class.c.)
     *
     * The verifier does *not* fail a class for using a const-class
     * or instance-of instruction referring to an unresolveable class,
     * because the result of the instruction is simply a Class object
     * or boolean -- there's no need to resolve the class object during
     * verification.  Instance field and virtual method accesses can
     * break dangerously if we get the wrong class, but const-class and
     * instance-of are only interesting at execution time.  So, if we
     * we got here as part of executing one of the "unverified class"
     * instructions, we skip the additional check.
     *
     * Ditto for class references from annotations and exception
     * handler lists.
     */
    if (!fromUnverifiedConstant &&
        IS_CLASS_FLAG_SET(referrer, CLASS_ISPREVERIFIED))
    {
        ClassObject* resClassCheck = resClass;
        if (dvmIsArrayClass(resClassCheck))
            resClassCheck = resClassCheck->elementClass;

        if (referrer->pDvmDex != resClassCheck->pDvmDex &&
            resClassCheck->classLoader != NULL)
        {
            LOGW("Class resolved by unexpected DEX:"
```

</td></tr>
</table>

The ’104 Reissue Patent	Infringed By
	`` " %s(%p):%p ref [%s] %s(%p):%p\n", referrer->descriptor, referrer->classLoader, referrer->pDvmDex, resClass->descriptor, resClassCheck->descriptor, resClassCheck->classLoader, resClassCheck->pDvmDex); LOGW("(%s had used a different %s during pre-verification)\n", referrer->descriptor, resClass->descriptor); dvmThrowException("Ljava/lang/IllegalAccessError;", "Class ref in pre-verified class resolved to unexpected " "implementation"); return NULL; } } LOGVV("##### +ResolveClass(%s): referrer=%s dex=%p ldr=%p ref=%d\n", resClass->descriptor, referrer->descriptor, referrer->pDvmDex, referrer->classLoader, classIdx); /* * Add what we found to the list so we can skip the class search * next time through. * * TODO: should we be doing this when fromUnverifiedConstant==true? * (see comments at top of oo/Class.c) */ dvmDexSetResolvedClass(pDvmDex, classIdx, resClass); } else { /* not found, exception should be raised */ LOGVV("Class not found: %s\n", dexStringByTypeIdx(pDvmDex->pDexFile, classIdx)); assert(dvmCheckException(dvmThreadSelf())); } ```

The ’104 Reissue Patent	Infringed By
	`` return resClass; } /* * Find the method corresponding to "methodRef". * * We use "referrer" to find the DexFile with the constant pool that * "methodRef" is an index into. We also use its class loader. The method * being resolved may very well be in a different DEX file. * * If this is a static method, we ensure that the method's class is * initialized. */ Method* dvmResolveMethod(const ClassObject* referrer, u4 methodIdx, MethodType methodType) { DvmDex* pDvmDex = referrer->pDvmDex; ClassObject* resClass; const DexMethodId* pMethodId; Method* resMethod; assert(methodType != METHOD_INTERFACE); LOGVV("--- resolving method %u (referrer=%s)\n", methodIdx, referrer->descriptor); pMethodId = dexGetMethodId(pDvmDex->pDexFile, methodIdx); resClass = dvmResolveClass(referrer, pMethodId->classIdx, false); if (resClass == NULL) { /* can't find the class that the method is a part of */ ```

The ’104 Reissue Patent	Infringed By
	`` assert(dvmCheckException(dvmThreadSelf())); return NULL; } if (dvmIsInterfaceClass(resClass)) { /* method is part of an interface */ dvmThrowExceptionWithClassMessage("Ljava/lang/IncompatibleClassChangeError;", resClass->descriptor); return NULL; } const char* name = dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx); DexProto proto; dexProtoSetFromMethodId(&proto, pDvmDex->pDexFile, pMethodId); /* * We need to chase up the class hierarchy to find methods defined * in super-classes. (We only want to check the current class * if we're looking for a constructor; since DIRECT calls are only * for constructors and private methods, we don't want to walk up.) */ if (methodType == METHOD_DIRECT) { resMethod = dvmFindDirectMethod(resClass, name, &proto); } else if (methodType == METHOD_STATIC) { resMethod = dvmFindDirectMethodHier(resClass, name, &proto); } else { resMethod = dvmFindVirtualMethodHier(resClass, name, &proto); } if (resMethod == NULL) { dvmThrowException("Ljava/lang/NoSuchMethodError;", name); return NULL; ```

The ’104 Reissue Patent	Infringed By
	(code below)

```
}

LOGVV("--- found method %d (%s.%s)\n",
    methodIdx, resClass->descriptor, resMethod->name);

/* see if this is a pure-abstract method */
if (dvmIsAbstractMethod(resMethod) && !dvmIsAbstractClass(resClass)) {
    dvmThrowException("Ljava/lang/AbstractMethodError;", name);
    return NULL;
}

/*
 * If we're the first to resolve this class, we need to initialize
 * it now.  Only necessary for METHOD_STATIC.
 */
if (methodType == METHOD_STATIC) {
    if (!dvmIsClassInitialized(resMethod->clazz) &&
        !dvmInitClass(resMethod->clazz))
    {
        assert(dvmCheckException(dvmThreadSelf()));
        return NULL;
    } else {
        assert(!dvmCheckException(dvmThreadSelf()));
    }
} else {
    /*
     * Edge case: if the <clinit> for a class creates an instance
     * of itself, we will call <init> on a class that is still being
     * initialized by us.
     */
    assert(dvmIsClassInitialized(resMethod->clazz) ||
          dvmIsClassInitializing(resMethod->clazz));
```

The ’104 Reissue Patent	Infringed By
	`` } /* * The class is initialized, the method has been found. Add a pointer * to our data structure so we don't have to jump through the hoops again. */ dvmDexSetResolvedMethod(pDvmDex, methodIdx, resMethod); return resMethod; } /* * Resolve an interface method reference. * * Returns NULL with an exception raised on failure. */ Method* dvmResolveInterfaceMethod(const ClassObject* referrer, u4 methodIdx) { DvmDex* pDvmDex = referrer->pDvmDex; ClassObject* resClass; const DexMethodId* pMethodId; Method* resMethod; int i; LOGVV("--- resolving interface method %d (referrer=%s)\n", methodIdx, referrer->descriptor); pMethodId = dexGetMethodId(pDvmDex->pDexFile, methodIdx); resClass = dvmResolveClass(referrer, pMethodId->classIdx, false); if (resClass == NULL) { /* can't find the class that the method is a part of */ assert(dvmCheckException(dvmThreadSelf())); ```

The ’104 Reissue Patent	Infringed By
	`` return NULL; } if (!dvmIsInterfaceClass(resClass)) { /* whoops */ dvmThrowExceptionWithClassMessage("Ljava/lang/IncompatibleClassChangeError;", resClass->descriptor); return NULL; } /* * This is the first time the method has been resolved. Set it in our * resolved-method structure. It always resolves to the same thing, * so looking it up and storing it doesn't create a race condition. * * If we scan into the interface's superclass -- which is always * java/lang/Object -- we will catch things like: * interface I ... * I myobj = (something that implements I) * myobj.hashCode() * However, the Method->methodIndex will be an offset into clazz->vtable, * rather than an offset into clazz->iftable. The invoke-interface * code can test to see if the method returned is abstract or concrete, * and use methodIndex accordingly. I'm not doing this yet because * (a) we waste time in an unusual case, and (b) we're probably going * to fix it in the DEX optimizer. * * We do need to scan the superinterfaces, in case we're invoking a * superinterface method on an interface reference. The class in the * DexTypeId is for the static type of the object, not the class in * which the method is first defined. We have the full, flattened * list in "iftable". ```

The ’104 Reissue Patent	Infringed By
	`` */ const char* methodName = dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx); DexProto proto; dexProtoSetFromMethodId(&proto, pDvmDex->pDexFile, pMethodId); LOGVV("+++ looking for '%s' '%s' in resClass='%s'\n", methodName, methodSig, resClass->descriptor); resMethod = dvmFindVirtualMethod(resClass, methodName, &proto); if (resMethod == NULL) { LOGVV("+++ did not resolve immediately\n"); for (i = 0; i < resClass->iftableCount; i++) { resMethod = dvmFindVirtualMethod(resClass->iftable[i].clazz, methodName, &proto); if (resMethod != NULL) break; } if (resMethod == NULL) { dvmThrowException("Ljava/lang/NoSuchMethodError;", methodName); return NULL; } } else { LOGVV("+++ resolved immediately: %s (%s %d)\n", resMethod->name, resMethod->clazz->descriptor, (u4) resMethod->methodIndex); } LOGVV("--- found interface method %d (%s.%s)\n", methodIdx, resClass->descriptor, resMethod->name); /* we're expecting this to be abstract */ ```

The ’104 Reissue Patent	Infringed By
	assert(dvmIsAbstractMethod(resMethod)); /* interface methods are always public; no need to check access */ /* * The interface class *may* be initialized. According to VM spec * v2 2.17.4, the interfaces a class refers to "need not" be initialized * when the class is initialized. * * It isn't necessary for an interface class to be initialized before * we resolve methods on that interface. * * We choose not to do the initialization now. */ //assert(dvmIsClassInitialized(resMethod->clazz)); /* * The class is initialized, the method has been found. Add a pointer * to our data structure so we don't have to jump through the hoops again. */ dvmDexSetResolvedMethod(pDvmDex, methodIdx, resMethod); return resMethod; } /* * Resolve an instance field reference. * * Returns NULL and throws an exception on error (no such field, illegal * access). */ InstField* dvmResolveInstField(const ClassObject* referrer, u4 ifieldIdx)

The ’104 Reissue Patent	Infringed By
	`` { DvmDex* pDvmDex = referrer->pDvmDex; ClassObject* resClass; const DexFieldId* pFieldId; InstField* resField; LOGVV("--- resolving field %u (referrer=%s cl=%p)\n", ifieldIdx, referrer->descriptor, referrer->classLoader); pFieldId = dexGetFieldId(pDvmDex->pDexFile, ifieldIdx); /* * Find the field's class. */ resClass = dvmResolveClass(referrer, pFieldId->classIdx, false); if (resClass == NULL) { assert(dvmCheckException(dvmThreadSelf())); return NULL; } resField = dvmFindInstanceFieldHier(resClass, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx), dexStringByTypeIdx(pDvmDex->pDexFile, pFieldId->typeIdx)); if (resField == NULL) { dvmThrowException("Ljava/lang/NoSuchFieldError;", dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx)); return NULL; } /* * Class must be initialized by now (unless verifier is buggy). We * could still be in the process of initializing it if the field ```

The ’104 Reissue Patent	Infringed By
	<pre> * access is from a static initializer. */ assert(dvmIsClassInitialized(resField->field.clazz) || dvmIsClassInitializing(resField->field.clazz)); /* * The class is initialized, the method has been found. Add a pointer * to our data structure so we don't have to jump through the hoops again. */ dvmDexSetResolvedField(pDvmDex, ifieldIdx, (Field*)resField); LOGVV(" field %u is %s.%s\n", ifieldIdx, resField->field.clazz->descriptor, resField->field.name); return resField; } /* * Resolve a static field reference. The DexFile format doesn't distinguish * between static and instance field references, so the "resolved" pointer * in the Dex struct will have the wrong type. We trivially cast it here. * * Causes the field's class to be initialized. */ StaticField* dvmResolveStaticField(const ClassObject* referrer, u4 sfieldIdx) { DvmDex* pDvmDex = referrer->pDvmDex; ClassObject* resClass; const DexFieldId* pFieldId; StaticField* resField; pFieldId = dexGetFieldId(pDvmDex->pDexFile, sfieldIdx);</pre>

The ’104 Reissue Patent	Infringed By
	`` /* * Find the field's class. */ resClass = dvmResolveClass(referrer, pFieldId->classIdx, false); if (resClass == NULL) { assert(dvmCheckException(dvmThreadSelf())); return NULL; } resField = dvmFindStaticFieldHier(resClass, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx), dexStringByTypeIdx(pDvmDex->pDexFile, pFieldId->typeIdx)); if (resField == NULL) { dvmThrowException("Ljava/lang/NoSuchFieldError;", dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx)); return NULL; } /* * If we're the first to resolve the field in which this class resides, * we need to do it now. Note that, if the field was inherited from * a superclass, it is not necessarily the same as "resClass". */ if (!dvmIsClassInitialized(resField->field.clazz) && !dvmInitClass(resField->field.clazz)) { assert(dvmCheckException(dvmThreadSelf())); return NULL; } /* * The class is initialized, the method has been found. Add a pointer ```

The ’104 Reissue Patent	Infringed By
	`` * to our data structure so we don't have to jump through the hoops again. */ dvmDexSetResolvedField(pDvmDex, sfieldIdx, (Field*) resField); return resField; } /* * Resolve a string reference. * * Finding the string is easy. We need to return a reference to a * java/lang/String object, not a bunch of characters, which means the * first time we get here we need to create an interned string. */ StringObject* dvmResolveString(const ClassObject* referrer, u4 stringIdx) { DvmDex* pDvmDex = referrer->pDvmDex; StringObject* strObj; StringObject* internStrObj; const char* utf8; u4 utf16Size; LOGVV("+++ resolving string, referrer is %s\n", referrer->descriptor); /* * Create a UTF-16 version so we can trivially compare it to what's * already interned. */ utf8 = dexStringAndSizeById(pDvmDex->pDexFile, stringIdx, &utf16Size); strObj = dvmCreateStringFromCstrAndLength(utf8, utf16Size, ALLOC_DEFAULT); ```

The ’104 Reissue Patent	Infringed By
	if (strObj == NULL) { /* ran out of space in GC heap? */ assert(dvmCheckException(dvmThreadSelf())); goto bail; } /* * Add it to the intern list. The return value is the one in the * intern list, which (due to race conditions) may or may not be * the one we just created. The intern list is synchronized, so * there will be only one "live" version. * * By requesting an immortal interned string, we guarantee that * the returned object will never be collected by the GC. * * A NULL return here indicates some sort of hashing failure. */ internStrObj = dvmLookupImmortalInternedString(strObj); dvmReleaseTrackedAlloc((Object*) strObj, NULL); strObj = internStrObj; if (strObj == NULL) { assert(dvmCheckException(dvmThreadSelf())); goto bail; } /* save a reference so we can go straight to the object next time */ dvmDexSetResolvedString(pDvmDex, stringIdx, strObj); bail: return strObj; }

The ’104 Reissue Patent	Infringed By
	/* * For debugging: return a string representing the methodType. */ const char* dvmMethodTypeStr(MethodType methodType) { switch (methodType) { case METHOD_DIRECT: return "direct"; case METHOD_STATIC: return "static"; case METHOD_VIRTUAL: return "virtual"; case METHOD_INTERFACE: return "interface"; case METHOD_UNKNOWN: return "UNKNOWN"; } assert(false); return "BOGUS"; } Another way that Android and Android-based devices meet the claim limitation is through the dexopt tool. *See, e.g.*, dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: **dexopt** We want to verify and optimize all of the classes in the DEX file. The easiest and safest way to do this is to load all of the classes into the VM and run through them. Anything that fails to load is simply not verified or optimized. Unfortunately, this can cause allocation of some resources that are difficult to release (e.g. loading of native shared libraries), so we don't want to do it in the same virtual machine that we're running applications in. The solution is to invoke a program called dexopt, which is really just a back door into the VM. It performs an abbreviated VM initialization, loads zero or more DEX files from the bootstrap class path, and then sets about verifying and optimizing whatever it can from the target DEX. On

The ’104 Reissue Patent	Infringed By
	completion, the process exits, freeing all resources. It is possible for multiple VMs to want the same DEX file at the same time. File locking is used to ensure that dexopt is only run once. …. *See also, e.g.*, dalvik\docs\ embedded-vm-control.html#verifier (“The system tries to pre-verify all classes in a DEX file to reduce class load overhead, and performs a series of optimizations to improve runtime performance. Both of these are done by the dexopt command, either in the build system or by the installer. On a development device, dexopt may be run the first time a DEX file is used and whenever it or one of its dependencies is updated ("just-in-time" optimization and verification).”). Dexopt loads the intermediate code class files, and when it encounters a symbolic reference (e.g., virtual method calls, field gets/puts), it determines the numerical reference corresponding to the symbolic reference and stores the numerical reference so that the processor can obtain data in accordance to the numerical references. *See, e.g.*, dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: **Dalvik Optimization and Verification With *dexopt*** The Dalvik virtual machine was designed specifically for the Android mobile platform. The target systems have little RAM, store data on slow internal flash memory, and generally have the performance characteristics of decade-old desktop systems. They also run Linux, which provides virtual memory, processes and threads, and UID-based security mechanisms. The features and limitations caused us to focus on certain goals: • Class data, notably bytecode, must be shared between multiple processes to minimize total system memory usage. • The overhead in launching a new app must be minimized to keep the device responsive. • Storing class data in individual files results in a lot of redundancy, especially with respect

The ’104 Reissue Patent	Infringed By
	to strings. To conserve disk space we need to factor this out. • Parsing class data fields adds unnecessary overhead during class loading. Accessing data values (e.g. integers and strings) directly as C types is better. • Bytecode verification is necessary, but slow, so we want to verify as much as possible outside app execution. • Bytecode optimization (quickened instructions, method pruning) is important for speed and battery life. • For security reasons, processes may not edit shared code. The typical VM implementation uncompresses individual classes from a compressed archive and stores them on the heap. This implies a separate copy of each class in every process, and slows application startup because the code must be uncompressed (or at least read off disk in many small pieces). On the other hand, having the bytecode on the local heap makes it easy to rewrite instructions on first use, facilitating a number of different optimizations. The goals led us to make some fundamental decisions: • Multiple classes are aggregated into a single "DEX" file. • DEX files are mapped read-only and shared between processes. • Byte ordering and word alignment are adjusted to suit the local system. • Bytecode verification is mandatory for all classes, but we want to "pre-verify" whatever we can. • Optimizations that require rewriting bytecode must be done ahead of time. • The consequences of these decisions are explained in the following sections. …. **dexopt** We want to verify and optimize all of the classes in the DEX file. The easiest and safest way to do this is to load all of the classes into the VM and run through them. Anything that fails to load is simply not verified or optimized. Unfortunately, this can cause allocation of some resources that are difficult to release (e.g. loading of native shared libraries), so we don't want to do it in the same virtual machine that we're running applications in.

The ’104 Reissue Patent	Infringed By
	The solution is to invoke a program called dexopt, which is really just a back door into the VM. It performs an abbreviated VM initialization, loads zero or more DEX files from the bootstrap class path, and then sets about verifying and optimizing whatever it can from the target DEX. On completion, the process exits, freeing all resources. It is possible for multiple VMs to want the same DEX file at the same time. File locking is used to ensure that dexopt is only run once. …. **Optimization** Virtual machine interpreters typically perform certain optimizations the first time a piece of code is used. Constant pool references are replaced with pointers to internal data structures, operations that always succeed or always work a certain way are replaced with simpler forms. Some of these require information only available at runtime, others can be inferred statically when certain assumptions are made. The Dalvik optimizer does the following: • For virtual method calls, replace the method index with a vtable index. • For instance field get/put, replace the field index with a byte offset. Also, merge the boolean / byte / char / short variants into a single 32-bit form (less code in the interpreter means more room in the CPU I-cache). • Replace a handful of high-volume calls, like String.length(), with "inline" replacements. This skips the usual method call overhead, directly switching from the interpreter to a native implementation. • Prune empty methods. The simplest example is Object.<init>, which does nothing, but must be called whenever any object is allocated. The instruction is replaced with a new version that acts as a no-op unless a debugger is attached. • Append pre-computed data. For example, the VM wants to have a hash table for lookups on class name. Instead of computing this when the DEX file is loaded, we can compute it now, saving heap space and computation time in every VM where the DEX is loaded.

The ’104 Reissue Patent	Infringed By
	All of the instruction modifications involve replacing the opcode with one not defined by the Dalvik specification. This allows us to freely mix optimized and unoptimized instructions. The set of optimized instructions, and their exact representation, is tied closely to the VM version. Most of the optimizations are obvious "wins". The use of raw indices and offsets not only allows us to execute more quickly, we can also skip the initial symbolic resolution. Pre-computation eats up disk space, and so must be done in moderation. There are a couple of potential sources of trouble with these optimizations. First, vtable indices and byte offsets are subject to change if the VM is updated. Second, if a superclass is in a different DEX, and that other DEX is updated, we need to ensure that our optimized indices and offsets are updated as well. A similar but more subtle problem emerges when user-defined class loaders are employed: the class we actually call may not be the one we expected to call. These problems are addressed with dependency lists and some limitations on what can be optimized. *See, e.g.*, dalvik\vm\analysis\ReduceConstants.c: /* Overview When a class, method, field, or string constant is referred to from Dalvik bytecode, the reference takes the form of an integer index value. This value indexes into an array of type_id_item, method_id_item, field_id_item, or string_id_item in the DEX file. The first three themselves contain (directly or indirectly) indexes to strings that the resolver uses to convert the instruction stream index into a pointer to the appropriate object or struct. For example, an invoke-virtual instruction needs to specify which method is to be invoked. The method constant indexes into the method_id_item

The ’104 Reissue Patent	Infringed By
	array, each entry of which has indexes that specify the defining class (type_id_item), method name (string_id_item), and method prototype (proto_id_item). The type_id_item just holds an index to a string_id_item, which holds the file offset to the string with the class name. The VM finds the class by name, then searches through the class' table of virtual methods to find one with a matching name and prototype. This process is fairly expensive, so after the first time it completes successfully, the VM records that the method index resolved to a specific Method struct. On subsequent execution, the VM just pulls the Method ptr out of the resolved-methods array. A similar approach is used with the indexes for classes, fields, and string constants. The problem with this approach is that we need to have a "resolved" entry for every possible class, method, field, and string constant in every DEX file, even if some of those aren't used from code. The DEX string constant table has entries for method prototypes and class names that are never used by the code, and "public static final" fields often turn into immediate constants. The resolution table entries are only 4 bytes each, but there are roughly 200,000 of them in the bootstrap classes alone. DEX optimization removes many index references by replacing virtual method indexes with vtable offsets and instance field indexes with byte offsets. In the earlier example, the method would be resolved at "dexopt" time, and the instruction rewritten as invoke-virtual-quick with the vtable offset. (There are comparatively few classes compared to other constant pool entries, and a much higher percentage (typically 60-70%) are used. The biggest gains come from the string pool.) Using the resolved-entity tables provides a substantial performance improvement, but results in applications allocating 1MB+ of tables that

The ’104 Reissue Patent	Infringed By
	are 70% unused. The used and unused entries are freely intermixed, preventing effective sharing with the zygote process, and resulting in large numbers of private/dirty pages on the native heap as the tables populate on first use. The trick is to reduce the memory usage without decreasing performance. Using smaller resolved-entity tables can actually give us a speed boost, because we'll have a smaller "live" set of pages and make more effective use of the data cache. The approach we're going to use is to determine the set of indexes that could potentially be resolved, generate a mapping from the minimal set to the full set, and append the mapping to the DEX file. This is done at "dexopt" time, because we need to keep the changes in shared/read-only pages or we'll lose the benefits of doing the work. There are two ways to create and use the new mapping: (1) Write the entire full->minimal mapping to the ".odex" file. On every instruction that uses an index, use the mapping to determine the "compressed" constant value, and then use that to index into the resolved-entity tables on the heap. The instruction stream is unchanged, and the resolver can easily tell if a given index is cacheable. (2) Write the inverse miminal->full mapping to the ".odex" file, and rewrite the constants in the instruction stream. The interpreter is unchanged, and the resolver code uses the mapping to find the original data in the DEX. Approach #1 is easier and safer to implement, but it requires a table lookup every time we execute an instruction that includes a constant

The ’104 Reissue Patent	Infringed By
	pool reference. This causes an unacceptable performance hit, chiefly because we're hitting semi-random memory pages and hosing the data cache. This is mitigated somewhat by DEX optimizations that replace the constant with a vtable index or field byte offset. Approach #1 also requires a larger map table, increasing the size of the DEX on disk. One nice property of approach #1 is that most of the DEX file is unmodified, so use of the mapping is a runtime decision. Approach #2 is preferred for performance reasons. The class/method/field/string resolver code has to handle indices from three sources: interpreted instructions, annotations, and exception "catch" lists. Sometimes these occur indirectly, e.g. we need to resolve the declaring class associated with fields and methods when the latter two are themselves resolved. Parsing and rewriting instructions is fairly straightforward, but annotations use a complex format with variable-width index values. We can safely rewrite index values in annotations if we guarantee that the new value is smaller than the original. This implies a two-pass approach: the first determines the set of indexes actually used, the second does the rewrite. Doing the rewrite in a single pass would be much harder. Instances of the "original" indices will still be found in the file; if we try to be all-inclusive we will include some stuff that doesn't need to be there (e.g. we don't generally need to cache the class name string index result, since once we have the class resolved we don't need to look it up by name through the resolver again). There is some potential for performance improvement by caching more than we strictly need, but we can afford to give up a little performance during class loading if it allows us to regain some memory.

The ’104 Reissue Patent	Infringed By
	For safety and debugging, it's useful to distinguish the "compressed" constants in some way, e.g. setting the high bit when we rewrite them. In practice we don't have any free bits: indexes are usually 16-bit values, and we have more than 32,767 string constants in at least one of our core DEX files. Also, this does not work with constants embedded in annotations, because of the variable-width encoding. We should be safe if we can establish a clear distinction between sources of "original" and "compressed" indices. If the values get crossed up we can end up with elusive bugs. The easiest approach is to declare that only indices pulled from certain locations (the instruction stream and/or annotations) are compressed. This prevents us from adding indices in arbitrary locations to the compressed set, but should allow a reasonably robust implementation. … */ dalvik\vm\analysis\DexOptimize.h: /* * Abbreviated resolution functions, for use by optimization and verification * code. */ ClassObject* dvmOptResolveClass(ClassObject* referrer, u4 classIdx, VerifyError* pFailure); Method* dvmOptResolveMethod(ClassObject* referrer, u4 methodIdx, MethodType methodType, VerifyError* pFailure); Method* dvmOptResolveInterfaceMethod(ClassObject* referrer, u4 methodIdx); InstField* dvmOptResolveInstField(ClassObject* referrer, u4 ifieldIdx, VerifyError* pFailure); StaticField* dvmOptResolveStaticField(ClassObject* referrer, u4 sfieldIdx,

The ’104 Reissue Patent	Infringed By
	(continued below)

```
    VerifyError* pFailure);

dalvik\vm\analysis\DexOptimize.c:
/*
 * ===========================================================================
 *      Optimizations
 * ===========================================================================
 */

/*
 * Perform in-place rewrites on a memory-mapped DEX file.
 *
 * This happens in a short-lived child process, so we can go nutty with
 * loading classes and allocating memory.
 */
static bool rewriteDex(u1* addr, int len, bool doVerify, bool doOpt,
    u4* pHeaderFlags, DexClassLookup** ppClassLookup)
{
    u8 prepWhen, loadWhen, verifyWhen, optWhen;
    DvmDex* pDvmDex = NULL;
    bool result = false;

    *pHeaderFlags = 0;

    LOGV("+++ swapping bytes\n");
    if (dexFixByteOrdering(addr, len) != 0)
        goto bail;
#if __BYTE_ORDER != __LITTLE_ENDIAN
```

The ’104 Reissue Patent	Infringed By
	`` *pHeaderFlags |= DEX_OPT_FLAG_BIG; #endif /* * Now that the DEX file can be read directly, create a DexFile for it. */ if (dvmDexFileOpenPartial(addr, len, &pDvmDex) != 0) { LOGE("Unable to create DexFile\n"); goto bail; } /* * Create the class lookup table. */ //startWhen = dvmGetRelativeTimeUsec(); *ppClassLookup = dexCreateClassLookup(pDvmDex->pDexFile); if (*ppClassLookup == NULL) goto bail; /* * Bail out early if they don't want The Works. The current implementation * doesn't fork a new process if this flag isn't set, so we really don't * want to continue on with the crazy class loading. */ if (!doVerify && !doOpt) { result = true; goto bail; } /* this is needed for the next part */ pDvmDex->pDexFile->pClassLookup = *ppClassLookup; ```

The ’104 Reissue Patent	Infringed By
	<pre> prepWhen = dvmGetRelativeTimeUsec(); /* * Load all classes found in this DEX file. If they fail to load for * some reason, they won't get verified (which is as it should be). */ if (!loadAllClasses(pDvmDex)) goto bail; loadWhen = dvmGetRelativeTimeUsec(); /* * Verify all classes in the DEX file. Export the "is verified" flag * to the DEX file we're creating. */ if (doVerify) { dvmVerifyAllClasses(pDvmDex->pDexFile); *pHeaderFlags |= DEX_FLAG_VERIFIED; } verifyWhen = dvmGetRelativeTimeUsec(); /* * Optimize the classes we successfully loaded. If the opt mode is * OPTIMIZE_MODE_VERIFIED, each class must have been successfully * verified or we'll skip it. */ #ifndef PROFILE_FIELD_ACCESS if (doOpt) { optimizeLoadedClasses(pDvmDex->pDexFile); *pHeaderFlags |= DEX_OPT_FLAG_FIELDS	DEX_OPT_FLAG_INVOCATIONS; } #endif optWhen = dvmGetRelativeTimeUsec();</pre>

The ’104 Reissue Patent	Infringed By
	`` LOGD("DexOpt: load %dms, verify %dms, opt %dms\n", (int) (loadWhen - prepWhen) / 1000, (int) (verifyWhen - loadWhen) / 1000, (int) (optWhen - verifyWhen) / 1000); result = true; bail: /* free up storage */ dvmDexFileFree(pDvmDex); return result; } … /* * Alternate version of dvmResolveClass for use with verification and * optimization. Performs access checks on every resolve, and refuses * to acknowledge the existence of classes defined in more than one DEX * file. * * Exceptions caused by failures are cleared before returning. * * On failure, returns NULL, and sets *pFailure if pFailure is not NULL. */ ClassObject* dvmOptResolveClass(ClassObject* referrer, u4 classIdx, VerifyError* pFailure) { DvmDex* pDvmDex = referrer->pDvmDex; ```

The ’104 Reissue Patent	Infringed By
	ClassObject* resClass; /* * Check the table first. If not there, do the lookup by name. */ resClass = dvmDexGetResolvedClass(pDvmDex, classIdx); if (resClass == NULL) { const char* className = dexStringByTypeIdx(pDvmDex->pDexFile, classIdx); if (className[0] != '\0' && className[1] == '\0') { /* primitive type */ resClass = dvmFindPrimitiveClass(className[0]); } else { resClass = dvmFindClassNoInit(className, referrer->classLoader); } if (resClass == NULL) { /* not found, exception should be raised */ LOGV("DexOpt: class %d (%s) not found\n", classIdx, dexStringByTypeIdx(pDvmDex->pDexFile, classIdx)); if (pFailure != NULL) { /* dig through the wrappers to find the original failure */ Object* excep = dvmGetException(dvmThreadSelf()); while (true) { Object* cause = dvmGetExceptionCause(excep); if (cause == NULL) break; excep = cause; } if (strcmp(excep->clazz->descriptor, "Ljava/lang/IncompatibleClassChangeError;") == 0) { *pFailure = VERIFY_ERROR_CLASS_CHANGE;

The ’104 Reissue Patent	Infringed By
	`` } else { *pFailure = VERIFY_ERROR_NO_CLASS; } } dvmClearOptException(dvmThreadSelf()); return NULL; } /* * Add it to the resolved table so we're faster on the next lookup. */ dvmDexSetResolvedClass(pDvmDex, classIdx, resClass); } /* multiple definitions? */ if (IS_CLASS_FLAG_SET(resClass, CLASS_MULTIPLE_DEFS)) { LOGI("DexOpt: not resolving ambiguous class '%s'\n", resClass->descriptor); if (pFailure != NULL) *pFailure = VERIFY_ERROR_NO_CLASS; return NULL; } /* access allowed? */ tweakLoader(referrer, resClass); bool allowed = dvmCheckClassAccess(referrer, resClass); untweakLoader(referrer, resClass); if (!allowed) { LOGW("DexOpt: resolve class illegal access: %s -> %s\n", referrer->descriptor, resClass->descriptor); if (pFailure != NULL) *pFailure = VERIFY_ERROR_ACCESS_CLASS; ```

The ’104 Reissue Patent	Infringed By
	`return NULL;` `}` `return resClass;` `}` `/*` ` * Alternate version of dvmResolveInstField().` ` *` ` * On failure, returns NULL, and sets *pFailure if pFailure is not NULL.` ` */` `InstField* dvmOptResolveInstField(ClassObject* referrer, u4 ifieldIdx,` `    VerifyError* pFailure)` `{` `    DvmDex* pDvmDex = referrer->pDvmDex;` `    InstField* resField;` `    resField = (InstField*) dvmDexGetResolvedField(pDvmDex, ifieldIdx);` `    if (resField == NULL) {` `        const DexFieldId* pFieldId;` `        ClassObject* resClass;` `        pFieldId = dexGetFieldId(pDvmDex->pDexFile, ifieldIdx);` `        /*` `         * Find the field's class.` `         */` `        resClass = dvmOptResolveClass(referrer, pFieldId->classIdx, pFailure);` `        if (resClass == NULL) {` `            //dvmClearOptException(dvmThreadSelf());` `            assert(!dvmCheckException(dvmThreadSelf()));` `            if (pFailure != NULL) { assert(!VERIFY_OK(*pFailure)); }`

The ’104 Reissue Patent	Infringed By
	`` return NULL; } resField = (InstField*)dvmFindFieldHier(resClass, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx), dexStringByTypeIdx(pDvmDex->pDexFile, pFieldId->typeIdx)); if (resField == NULL) { LOGD("DexOpt: couldn't find field %s.%s\n", resClass->descriptor, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx)); if (pFailure != NULL) *pFailure = VERIFY_ERROR_NO_FIELD; return NULL; } if (dvmIsStaticField(&resField->field)) { LOGD("DexOpt: wanted instance, got static for field %s.%s\n", resClass->descriptor, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx)); if (pFailure != NULL) *pFailure = VERIFY_ERROR_CLASS_CHANGE; return NULL; } /* * Add it to the resolved table so we're faster on the next lookup. */ dvmDexSetResolvedField(pDvmDex, ifieldIdx, (Field*) resField); } /* access allowed? */ tweakLoader(referrer, resField->field.clazz); bool allowed = dvmCheckFieldAccess(referrer, (Field*)resField); ```

The ’104 Reissue Patent	Infringed By
	<pre> untweakLoader(referrer, resField->field.clazz); if (!allowed) { LOGI("DexOpt: access denied from %s to field %s.%s\n", referrer->descriptor, resField->field.clazz->descriptor, resField->field.name); if (pFailure != NULL) *pFailure = VERIFY_ERROR_ACCESS_FIELD; return NULL; } return resField; } /* * Alternate version of dvmResolveStaticField(). * * Does not force initialization of the resolved field's class. * * On failure, returns NULL, and sets *pFailure if pFailure is not NULL. */ StaticField* dvmOptResolveStaticField(ClassObject* referrer, u4 sfieldIdx, VerifyError* pFailure) { DvmDex* pDvmDex = referrer->pDvmDex; StaticField* resField; resField = (StaticField*)dvmDexGetResolvedField(pDvmDex, sfieldIdx); if (resField == NULL) { const DexFieldId* pFieldId; ClassObject* resClass; pFieldId = dexGetFieldId(pDvmDex->pDexFile, sfieldIdx);</pre>

The ’104 Reissue Patent	Infringed By
	<pre>/* * Find the field's class. */ resClass = dvmOptResolveClass(referrer, pFieldId->classIdx, pFailure); if (resClass == NULL) { //dvmClearOptException(dvmThreadSelf()); assert(!dvmCheckException(dvmThreadSelf())); if (pFailure != NULL) { assert(!VERIFY_OK(*pFailure)); } return NULL; } resField = (StaticField*)dvmFindFieldHier(resClass, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx), dexStringByTypeIdx(pDvmDex->pDexFile, pFieldId->typeIdx)); if (resField == NULL) { LOGD("DexOpt: couldn't find static field\n"); if (pFailure != NULL) *pFailure = VERIFY_ERROR_NO_FIELD; return NULL; } if (!dvmIsStaticField(&resField->field)) { LOGD("DexOpt: wanted static, got instance for field %s.%s\n", resClass->descriptor, dexStringById(pDvmDex->pDexFile, pFieldId->nameIdx)); if (pFailure != NULL) *pFailure = VERIFY_ERROR_CLASS_CHANGE; return NULL; } /* * Add it to the resolved table so we're faster on the next lookup.</pre>

The ’104 Reissue Patent	Infringed By
	`` * * We can only do this if we're in "dexopt", because the presence * of a valid value in the resolution table implies that the class * containing the static field has been initialized. */ if (gDvm.optimizing) dvmDexSetResolvedField(pDvmDex, sfieldIdx, (Field*) resField); } /* access allowed? */ tweakLoader(referrer, resField->field.clazz); bool allowed = dvmCheckFieldAccess(referrer, (Field*)resField); untweakLoader(referrer, resField->field.clazz); if (!allowed) { LOGI("DexOpt: access denied from %s to field %s.%s\n", referrer->descriptor, resField->field.clazz->descriptor, resField->field.name); if (pFailure != NULL) *pFailure = VERIFY_ERROR_ACCESS_FIELD; return NULL; } return resField; } /* * Rewrite an iget/iput instruction. These all have the form: * op vA, vB, field@CCCC * * Where vA holds the value, vB holds the object reference, and CCCC is * the field reference constant pool offset. We want to replace CCCC * with the byte offset from the start of the object. ```

The ’104 Reissue Patent	Infringed By
	`` * * "clazz" is the referring class. We need this because we verify * access rights here. */ static void rewriteInstField(Method* method, u2* insns, OpCode newOpc) { ClassObject* clazz = method->clazz; u2 fieldIdx = insns[1]; InstField* field; int byteOffset; field = dvmOptResolveInstField(clazz, fieldIdx, NULL); if (field == NULL) { LOGI("DexOpt: unable to optimize field ref 0x%04x at 0x%02x in %s.%s\n", fieldIdx, (int) (insns - method->insns), clazz->descriptor, method->name); return; } if (field->byteOffset >= 65536) { LOGI("DexOpt: field offset exceeds 64K (%d)\n", field->byteOffset); return; } insns[0] = (insns[0] & 0xff00) | (u2) newOpc; insns[1] = (u2) field->byteOffset; LOGVV("DexOpt: rewrote access to %s.%s --> %d\n", field->field.clazz->descriptor, field->field.name, field->byteOffset); } /* ```

The ’104 Reissue Patent	Infringed By
	* Alternate version of dvmResolveMethod(). * * Doesn't throw exceptions, and checks access on every lookup. * * On failure, returns NULL, and sets *pFailure if pFailure is not NULL. */ Method* dvmOptResolveMethod(ClassObject* referrer, u4 methodIdx, MethodType methodType, VerifyError* pFailure) { DvmDex* pDvmDex = referrer->pDvmDex; Method* resMethod; assert(methodType == METHOD_DIRECT || methodType == METHOD_VIRTUAL || methodType == METHOD_STATIC); LOGVV("--- resolving method %u (referrer=%s)\n", methodIdx, referrer->descriptor); resMethod = dvmDexGetResolvedMethod(pDvmDex, methodIdx); if (resMethod == NULL) { const DexMethodId* pMethodId; ClassObject* resClass; pMethodId = dexGetMethodId(pDvmDex->pDexFile, methodIdx); resClass = dvmOptResolveClass(referrer, pMethodId->classIdx, pFailure); if (resClass == NULL) { /* * Can't find the class that the method is a part of, or don't * have permission to access the class. */

The ’104 Reissue Patent	Infringed By
	`` LOGV("DexOpt: can't find called method's class (?.%s)\n", dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx)); if (pFailure != NULL) { assert(!VERIFY_OK(*pFailure)); } return NULL; } if (dvmIsInterfaceClass(resClass)) { /* method is part of an interface; this is wrong method for that */ LOGW("DexOpt: method is in an interface\n"); if (pFailure != NULL) *pFailure = VERIFY_ERROR_GENERIC; return NULL; } /* * We need to chase up the class hierarchy to find methods defined * in super-classes. (We only want to check the current class * if we're looking for a constructor.) */ DexProto proto; dexProtoSetFromMethodId(&proto, pDvmDex->pDexFile, pMethodId); if (methodType == METHOD_DIRECT) { resMethod = dvmFindDirectMethod(resClass, dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx), &proto); } else { /* METHOD_STATIC or METHOD_VIRTUAL */ resMethod = dvmFindMethodHier(resClass, dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx), &proto); } if (resMethod == NULL) { LOGV("DexOpt: couldn't find method '%s'\n", ```

The ’104 Reissue Patent	Infringed By
	<pre> dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx)); if (pFailure != NULL) *pFailure = VERIFY_ERROR_NO_METHOD; return NULL; } if (methodType == METHOD_STATIC) { if (!dvmIsStaticMethod(resMethod)) { LOGD("DexOpt: wanted static, got instance for method %s.%s\n", resClass->descriptor, resMethod->name); if (pFailure != NULL) *pFailure = VERIFY_ERROR_CLASS_CHANGE; return NULL; } } else if (methodType == METHOD_VIRTUAL) { if (dvmIsStaticMethod(resMethod)) { LOGD("DexOpt: wanted instance, got static for method %s.%s\n", resClass->descriptor, resMethod->name); if (pFailure != NULL) *pFailure = VERIFY_ERROR_CLASS_CHANGE; return NULL; } } /* see if this is a pure-abstract method */ if (dvmIsAbstractMethod(resMethod) && !dvmIsAbstractClass(resClass)) { LOGW("DexOpt: pure-abstract method '%s' in %s\n", dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx), resClass->descriptor); if (pFailure != NULL) *pFailure = VERIFY_ERROR_GENERIC; return NULL; }</pre>

The ’104 Reissue Patent	Infringed By
	`` /* * Add it to the resolved table so we're faster on the next lookup. * * We can only do this for static methods if we're not in "dexopt", * because the presence of a valid value in the resolution table * implies that the class containing the static field has been * initialized. */ if (methodType != METHOD_STATIC || gDvm.optimizing) dvmDexSetResolvedMethod(pDvmDex, methodIdx, resMethod); } LOGVV("--- found method %d (%s.%s)\n", methodIdx, resMethod->clazz->descriptor, resMethod->name); /* access allowed? */ tweakLoader(referrer, resMethod->clazz); bool allowed = dvmCheckMethodAccess(referrer, resMethod); untweakLoader(referrer, resMethod->clazz); if (!allowed) { IF_LOGI() { char* desc = dexProtoCopyMethodDescriptor(&resMethod->prototype); LOGI("DexOpt: illegal method access (call %s.%s %s from %s)\n", resMethod->clazz->descriptor, resMethod->name, desc, referrer->descriptor); free(desc); } if (pFailure != NULL) *pFailure = VERIFY_ERROR_ACCESS_METHOD; return NULL; } ```

The ’104 Reissue Patent	Infringed By
	`` return resMethod; } /* * Rewrite invoke-virtual, invoke-virtual/range, invoke-super, and * invoke-super/range. These all have the form: * op vAA, meth@BBBB, reg stuff @CCCC * * We want to replace the method constant pool index BBBB with the * vtable index. */ static bool rewriteVirtualInvoke(Method* method, u2* insns, OpCode newOpc) { ClassObject* clazz = method->clazz; Method* baseMethod; u2 methodIdx = insns[1]; baseMethod = dvmOptResolveMethod(clazz, methodIdx, METHOD_VIRTUAL, NULL); if (baseMethod == NULL) { LOGD("DexOpt: unable to optimize virt call 0x%04x at 0x%02x in %s.%s\n", methodIdx, (int) (insns - method->insns), clazz->descriptor, method->name); return false; } assert((insns[0] & 0xff) == OP_INVOKE_VIRTUAL || (insns[0] & 0xff) == OP_INVOKE_VIRTUAL_RANGE || (insns[0] & 0xff) == OP_INVOKE_SUPER || (insns[0] & 0xff) == OP_INVOKE_SUPER_RANGE); ```

The ’104 Reissue Patent	Infringed By
	`` /* * Note: Method->methodIndex is a u2 and is range checked during the * initial load. */ insns[0] = (insns[0] & 0xff00) | (u2) newOpc; insns[1] = baseMethod->methodIndex; //LOGI("DexOpt: rewrote call to %s.%s --> %s.%s\n", // method->clazz->descriptor, method->name, // baseMethod->clazz->descriptor, baseMethod->name); return true; } … /* * Resolve an interface method reference. * * No method access check here -- interface methods are always public. * * Returns NULL if the method was not found. Does not throw an exception. */ Method* dvmOptResolveInterfaceMethod(ClassObject* referrer, u4 methodIdx) { DvmDex* pDvmDex = referrer->pDvmDex; Method* resMethod; int i; LOGVV("--- resolving interface method %d (referrer=%s)\n", methodIdx, referrer->descriptor); resMethod = dvmDexGetResolvedMethod(pDvmDex, methodIdx); ```

The ’104 Reissue Patent	Infringed By
	`` if (resMethod == NULL) { const DexMethodId* pMethodId; ClassObject* resClass; pMethodId = dexGetMethodId(pDvmDex->pDexFile, methodIdx); resClass = dvmOptResolveClass(referrer, pMethodId->classIdx, NULL); if (resClass == NULL) { /* can't find the class that the method is a part of */ dvmClearOptException(dvmThreadSelf()); return NULL; } if (!dvmIsInterfaceClass(resClass)) { /* whoops */ LOGI("Interface method not part of interface class\n"); return NULL; } const char* methodName = dexStringById(pDvmDex->pDexFile, pMethodId->nameIdx); DexProto proto; dexProtoSetFromMethodId(&proto, pDvmDex->pDexFile, pMethodId); LOGVV("+++ looking for '%s' '%s' in resClass='%s'\n", methodName, methodSig, resClass->descriptor); resMethod = dvmFindVirtualMethod(resClass, methodName, &proto); if (resMethod == NULL) { /* scan superinterfaces and superclass interfaces */ LOGVV("+++ did not resolve immediately\n"); for (i = 0; i < resClass->iftableCount; i++) { resMethod = dvmFindVirtualMethod(resClass->iftable[i].clazz, methodName, &proto); ```

The ’104 Reissue Patent	Infringed By
	`` if (resMethod != NULL) break; } if (resMethod == NULL) { LOGVV("+++ unable to resolve method %s\n", methodName); return NULL; } } else { LOGVV("+++ resolved immediately: %s (%s %d)\n", resMethod->name, resMethod->clazz->descriptor, (u4) resMethod->methodIndex); } /* we're expecting this to be abstract */ if (!dvmIsAbstractMethod(resMethod)) { char* desc = dexProtoCopyMethodDescriptor(&resMethod->prototype); LOGW("Found non-abstract interface method %s.%s %s\n", resMethod->clazz->descriptor, resMethod->name, desc); free(desc); return NULL; } /* * Add it to the resolved table so we're faster on the next lookup. */ dvmDexSetResolvedMethod(pDvmDex, methodIdx, resMethod); } LOGVV("--- found interface method %d (%s.%s)\n", methodIdx, resMethod->clazz->descriptor, resMethod->name); /* interface methods are always public; no need to check access */ ```

The '104 Reissue Patent	Infringed By
	`` return resMethod; } … ``` *See also*, *e.g.*, dalvik\vm\analysis\DexOptimize.c: ``` /* * Optimize instructions in a method. * * Returns "true" if all went well, "false" if we bailed out early when * something failed. */ static bool optimizeMethod(Method* method, const InlineSub* inlineSubs) { u4 insnsSize; u2* insns; u2 inst; if (dvmIsNativeMethod(method) || dvmIsAbstractMethod(method)) return true; insns = (u2*) method->insns; assert(insns != NULL); insnsSize = dvmGetMethodInsnsSize(method); while (insnsSize > 0) { int width; inst = *insns & 0xff; switch (inst) { ```

The ’104 Reissue Patent	Infringed By
	case OP_IGET: case OP_IGET_BOOLEAN: case OP_IGET_BYTE: case OP_IGET_CHAR: case OP_IGET_SHORT: rewriteInstField(method, insns, OP_IGET_s); break; case OP_IGET_WIDE: rewriteInstField(method, insns, OP_IGET_WIDE_QUICK); break; case OP_IGET_OBJECT: rewriteInstField(method, insns, OP_IGET_OBJECT_QUICK); break; case OP_IPUT: case OP_IPUT_BOOLEAN: case OP_IPUT_BYTE: case OP_IPUT_CHAR: case OP_IPUT_SHORT: rewriteInstField(method, insns, OP_IPUT_QUICK); break; case OP_IPUT_WIDE: rewriteInstField(method, insns, OP_IPUT_WIDE_QUICK); break; case OP_IPUT_OBJECT: rewriteInstField(method, insns, OP_IPUT_OBJECT_QUICK); break; case OP_INVOKE_VIRTUAL: if (!rewriteExecuteInline(method, insns, METHOD_VIRTUAL,inlineSubs)) { if (!rewriteVirtualInvoke(method, insns, OP_INVOKE_VIRTUAL_QUICK)) return false;

The ’104 Reissue Patent	Infringed By
	<pre> } break; case OP_INVOKE_VIRTUAL_RANGE: if (!rewriteExecuteInlineRange(method, insns, METHOD_VIRTUAL, inlineSubs)) { if (!rewriteVirtualInvoke(method, insns, OP_INVOKE_VIRTUAL_QUICK_RANGE)) { return false; } } break; case OP_INVOKE_SUPER: if (!rewriteVirtualInvoke(method, insns, OP_INVOKE_SUPER_QUICK)) return false; break; case OP_INVOKE_SUPER_RANGE: if (!rewriteVirtualInvoke(method, insns, OP_INVOKE_SUPER_QUICK_RANGE)) return false; break; case OP_INVOKE_DIRECT: if (!rewriteExecuteInline(method, insns, METHOD_DIRECT, inlineSubs)) { if (!rewriteEmptyDirectInvoke(method, insns)) return false; } break; case OP_INVOKE_DIRECT_RANGE: rewriteExecuteInlineRange(method, insns, METHOD_DIRECT, inlineSubs); break;</pre>

The ’104 Reissue Patent	Infringed By
	`` case OP_INVOKE_STATIC: rewriteExecuteInline(method, insns, METHOD_STATIC, inlineSubs); break; case OP_INVOKE_STATIC_RANGE: rewriteExecuteInlineRange(method, insns, METHOD_STATIC, inlineSubs); break; default: // ignore this instruction ; } if (*insns == kPackedSwitchSignature) { width = 4 + insns[1] * 2; } else if (*insns == kSparseSwitchSignature) { width = 2 + insns[1] * 4; } else if (*insns == kArrayDataSignature) { u2 elemWidth = insns[1]; u4 len = insns[2] | (((u4)insns[3]) << 16); width = 4 + (elemWidth * len + 1) / 2; } else { width = dexGetInstrWidth(gDvm.instrWidth, inst); } assert(width > 0); insns += width; insnsSize -= width; } assert(insnsSize == 0); return true; ```

The ’104 Reissue Patent	Infringed By
	}

The ’104 Reissue Patent	Infringed By
12. A computer-readable medium containing instructions for controlling a data processing system to perform a method for interpreting intermediate form object code comprised of instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of:	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices. They encompass a computer readable medium containing instructions for controlling a data processing system to perform a method for interpreting intermediate form object code comprised of instructions, certain of said instructions containing one or more symbolic references, to perform the steps described in the claim. *See* Claim 11, *supra*.
interpreting said instructions in accordance with a program execution control;	*See* Claim 11, *supra*. The Android platform has a Dalvik virtual machine that interprets intermediate form object code. Dexopt is part of the bytecode interpretation process because it’s a pre-pass made over the bytecodes to facilitate optimized bytecode execution. *See, e.g.*, dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: …. **dexopt** We want to verify and optimize all of the classes in the DEX file. The easiest and safest way to do this is to load all of the classes into the VM and run through them. Anything that fails to load is simply not verified or optimized. Unfortunately, this can cause allocation of some resources that are difficult to release (e.g. loading of native shared libraries), so we don't want to do it in the same virtual machine that we're running applications in. The solution is to invoke a program called dexopt, which is really just a back door into the VM. It performs an abbreviated VM initialization, loads zero or more DEX files

The ’104 Reissue Patent	Infringed By
	from the bootstrap class path, and then sets about verifying and optimizing whatever it can from the target DEX. On completion, the process exits, freeing all resources. It is possible for multiple VMs to want the same DEX file at the same time. File locking is used to ensure that dexopt is only run once. …. **Optimization** Virtual machine interpreters typically perform certain optimizations the first time a piece of code is used. Constant pool references are replaced with pointers to internal data structures, operations that always succeed or always work a certain way are replaced with simpler forms. Some of these require information only available at runtime, others can be inferred statically when certain assumptions are made. The Dalvik optimizer does the following: • For virtual method calls, replace the method index with a vtable index. • For instance field get/put, replace the field index with a byte offset. Also, merge the boolean / byte / char / short variants into a single 32-bit form (less code in the interpreter means more room in the CPU I-cache). ... *See also, e.g.*, dalvik\docs\ embedded-vm-control.html#verifier (“The system tries to pre-verify all classes in a DEX file to reduce class load overhead, and performs a series of optimizations to improve runtime performance. Both of these are done by the dexopt command, either in the build system or by the installer. On a development device, dexopt may be run the first time a DEX file is used and whenever it or one of its dependencies is updated ("just-in-time" optimization and verification).”). *See, e.g.*, Google I/O 2008 Video, Google I/O 2008 Video, entitled “Dalvik Virtual Machine Internals,” presented by Dan Bornstein (Google Android Project), available at http://developer.android.com/videos/index.html#v=ptjedOZEXPM:

The ’104 Reissue Patent	Infringed By
	• at 25:00 under “Register Machine” (“A dex byte code is defined in terms of an infinite register machine with no intra frames stack. So there’s a normal machine stack in terms of one method calling another, but within a method it’s all just registers. And we chose this because it lets us have a very efficient interpreter because each instruction that we interpret is semantically more dense and I’ll show an example of that in a couple slides.”). • at 25:42 under “Register Machine” (“This is just a general expectation what you can find when you’re converting a set of .class files into a dex file. We have, we have fewer instructions and we have fewer code units and I’ll talk about that in a second. But we do have, we do have more bytes and the distinction here is that a code unit in a .class file is a single byte and a code unit in dex is 2 bytes and in the interpreter itself we can issue reads to read those pairs of bytes at a time and so that helps mitigate the impact of having more bytes typically.”). • at 34:30 under “Interpreters 101” (“So it’s possible to write effectively the same technique as that last one in assembly. So here you can see I’ve elided a lot of the details, but the dispatch I’ve written, I have fully written out. You can see there’s still two reads of memory to do that dispatch. The first read is reading from your interpreted program counter and the second read is reading from your opcode table to find the address of that next opcode. So, as I’ve implied before at least, any memory activity that you have is going to be a significant performance impact. So if you can get rid of a read, you’re doing pretty good. And that’s exactly what we do.”). • at 35:15 under “Interpreters 101” (“So, instead of having that second memory read, what we do is have the base address of the entire interpreter sitting in a register and we guarantee that each opcode takes up the exact same number of bytes, so we will pad it. If an opcode happens to be short, there are a few of those, we won’t, we won’t use the entire space allocated for that opcode. And similarly, if an opcode happens to be one of the more heavier weight ones, such as a method invocation or field access, stuff like that, we will branch off to a helper function. But what that means, so there’s really two benefits here. One is, again, we’re avoiding that dispatch and another one is that we get to align the implementation of each opcode on a cache line boundary. So if there’s an, if there is an opcode which doesn’t happen to be used a lot, it won’t be taking up any space in the cache and that helps, that helps with the, with, you know,

The ’104 Reissue Patent	Infringed By
	leaving space for any other data accesses that your code might want to do.”). *See also, e.g.*, Dalvik Virtual Machine, “Porting Dalvik,” available at http://source.android.com/porting/dalvik.html: **Dalvik** The Dalvik virtual machine is intended to run on a variety of platforms. The baseline system is expected to be a variant of UNIX (Linux, BSD, Mac OS X) running the GNU C compiler. Little-endian CPUs have been exercised the most heavily, but big-endian systems are explicitly supported. There are two general categories of work: porting to a Linux system with a previously unseen CPU architecture, and porting to a different operating system. This document covers the former. … **Interpreter** The Dalvik runtime includes two interpreters, labeled "portable" and "fast". The portable interpreter is largely contained within a single C function, and should compile on any system that supports gcc. (If you don't have gcc, you may need to disable the "threaded" execution model, which relies on gcc's "goto table" implementation; look for the THREADED_INTERP define.) The fast interpreter uses hand-coded assembly fragments. If none are available for the current architecture, the build system will create an interpreter out of C "stubs". The resulting "all stubs" interpreter is quite a bit slower than the portable interpreter, making "fast" something of a misnomer. The fast interpreter is enabled by default. On platforms without native support, you may want to switch to the portable interpreter. This can be controlled with the dalvik.vm.execution-mode system property. For example, if you:

The ’104 Reissue Patent	Infringed By
	adb shell "echo dalvik.vm.execution-mode = int:portable >> /data/local.prop" and reboot, the Android app framework will start the VM with the portable interpreter enabled.
and resolving a symbolic reference in an instruction being interpreted, said step of resolving said symbolic reference including the substeps of:	*See* Claim 11-b, *supra*.
determining a numerical reference corresponding to said symbolic reference,	*See* Claim 11-b, *supra*.
and storing said numerical reference in a memory.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
13. A computer-implemented method for executing instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of:	Android includes methods for performing the steps described in the claim. *See* Claim 11, *supra*.
resolving a symbolic reference in an instruction, said step of resolving said symbolic reference including the substeps of:	*See* Claim 11-b, *supra*.
determining a numerical reference corresponding to said symbolic reference, and	*See* Claim 11-b, *supra*.
storing said numerical reference in a memory.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
14. The method of claim [13], wherein said substep of storing said numerical reference comprises the substep of replacing said symbolic reference with said numerical reference.	*See* Claim 13, *supra*.

The ’104 Reissue Patent	Infringed By
15. The method of claim [13], wherein said step of resolving said symbolic reference further comprises the substep of executing said instruction containing said symbolic reference using the stored numerical reference.	*See* Claim 13, *supra*. Also, *see*, *e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized?

The '104 Reissue Patent	Infringed By
	bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.

The '104 Reissue Patent	Infringed By
16. The method of claim [13], wherein said step of resolving said symbolic reference further comprises the substep of advancing program execution control after said substep of executing said instruction containing said symbolic reference.	*See* Claim 13, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class.

The ’104 Reissue Patent	Infringed By
	`` */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction ``` *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By

The ’104 Reissue Patent	Infringed By
17. In a computer system comprising a program, a method for executing said program comprising the steps of:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are computer systems. *See* Claim 11, *supra*.
receiving intermediate form object code for said program with symbolic data references in certain instructions of said intermediate form object code; and	*See* Claim 11-a, *supra*.
converting the instructions of the intermediate form object code having symbolic data references, said converting step comprising the substeps of:	*See* Claim 11-b, *supra*.
resolving said symbolic references to corresponding numerical references,	*See* Claim 11-b, *supra*.
storing said numerical references, and	*See* Claim 11-b, *supra*.
obtaining data in accordance to said numerical references.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
18. A computer-implemented method for executing program operations, each operation being comprised of a set of instructions,	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
certain of said instructions containing one or more symbolic references, said method comprising the steps of:	
receiving a set of instructions reflecting an operation; and	*See* Claim 11-a, *supra*.
performing the operation corresponding to the received set of instructions, wherein at least one of said symbolic references is resolved by determining a numerical reference corresponding to said symbolic reference, storing said numerical reference, and obtaining data in accordance to said stored numerical reference.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
19. A memory for use in executing a program by a processor, the memory comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and computers running the Android SDK encompass a memory for use in executing a program by a processor. *See* Claim 11, *supra*.
intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program,	*See* Claim 11-a, *supra*.
the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis, and	Lexical analysis and bytecode compilation is handled by javac. The bytecode is then further processed into .dex format by Android’s dx tool. *See* Claim 11-a, *supra*.
wherein the symbolic field	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
references are resolved by determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	

The ’104 Reissue Patent	Infringed By
20. A computer-implemented method for executing a compiled program containing instructions in an intermediate form code, at least one of the instructions containing a symbolic reference, said method comprising the steps of:	*See* Claim 11, *supra*.
resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and	*See* Claim 11-b, *supra*.
performing all operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
21. A memory encoded with a compiled program, the memory comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and computers running the Android SDK encompass a memory encoded with a compiled program. *See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program,	*See* Claim 11-a, *supra*.
the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis,	Lexical analysis and bytecode compilation is handled by javac. The bytecode is then further processed into .dex format by Android’s dx tool. *See* Claim 11-a, *supra*.
such that when the program is executed by a processor each symbolic field reference is resolved by determining a numerical reference corresponding to the symbolic field reference and data is obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
22. An apparatus comprising:	See Claim 11, *supra*.
a memory containing a compiled program in intermediate form object code constituted by a set of instructions, at least one of the instructions containing a symbolic reference; and	See Claim 11-a, *supra*.
a processor configured to execute the instruction by determining a	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
numerical reference corresponding to the symbolic reference, and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	

The ’104 Reissue Patent	Infringed By
23. A computer-readable medium containing instructions for controlling a data processing system to perform a method for interpreting a compiled program in intermediate form object code comprised of instructions, at least one of the instructions containing a symbolic reference, said method comprising the steps of:	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices. They encompass a computer readable medium containing instructions for controlling a data processing system to perform a method for interpreting a compiled program in intermediate form object code comprised of instructions, at least one of the instructions containing a symbolic reference, to perform the steps described in the claim. *See* Claim 11, *supra*.
resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and	*See* Claim 11-b, *supra*.
performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
24. A computer-implemented method for executing a program comprised of bytecodes, the method comprising:	*See* Claim 11, *supra*.
determining immediately prior to execution whether a bytecode of the program contains a symbolic data reference;	*See* Claim 11-b, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue ….

The ’104 Reissue Patent	Infringed By
	/* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.
when it is determined that the bytecode of the program contains a symbolic data reference, invoking a dynamic field reference routine to resolve the symbolic data reference; and	*See* Claim 11-b, *supra*.
executing thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
25. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a program	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
comprised of bytecodes and instructions for causing the processor to (i) determine immediately prior to execution of the program whether a bytecode of the program contains a symbolic data reference, (ii) when it is determined that the bytecode of the program contains a symbolic data reference, invoke a dynamic field reference routine to resolve the symbolic data reference, and (iii) execute thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: `/* ------------------------------ */` `.balign 64` `.L_OP_NEW_INSTANCE: /* 0x22 */` `/* File: armv5te/OP_NEW_INSTANCE.S */` `/*` ` * Create a new instance of a class.` ` */` `/* new-instance vAA, class@BBBB */` `ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex` `FETCH(r1, 1) @ r1<- BBBB` `ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses` `ldr r0, [r3, r1, lsl #2] @ r0<- resolved class` `EXPORT_PC() @ req'd for init, resolve, alloc` `cmp r0, #0 @ already resolved?` `beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now` `.LOP_NEW_INSTANCE_resolved: @ r0=class` `ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum` `cmp r1, #CLASS_INITIALIZED @ has class been initialized?` `bne .LOP_NEW_INSTANCE_needinit @ no, init class now` `.LOP_NEW_INSTANCE_initialized: @ r0=class` `mov r1, #ALLOC_DONT_TRACK @ flags for alloc call` `bl dvmAllocObject @ r0<- new object` `b .LOP_NEW_INSTANCE_finish @ continue` …. `/* continuation for OP_NEW_INSTANCE */` `.balign 32 @ minimize cache lines` `.LOP_NEW_INSTANCE_finish: @ r0=new object` `mov r3, rINST, lsr #8 @ r3<- AA`

The ’104 Reissue Patent	Infringed By
	`cmp r0, #0 @ failed?` `beq common_exceptionThrown @ yes, handle the exception` `FETCH_ADVANCE_INST(2) @ advance rPC, load rINST` `GET_INST_OPCODE(ip) @ extract opcode from rINST` `SET_VREG(r0, r3) @ vAA<- r0` `GOTO_OPCODE(ip) @ jump to next instruction` *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By
26. A computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, to perform the steps described in the claim. *See* Claim 11, *supra*.
determining immediately prior to execution whether a bytecode of the program contains a symbolic data reference;	*See* Claim 11-b, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: `/* ------------------------------ */` `.balign 64` `.L_OP_NEW_INSTANCE: /* 0x22 */` `/* File: armv5te/OP_NEW_INSTANCE.S */` `/*` ` * Create a new instance of a class.` ` */` `/* new-instance vAA, class@BBBB */` `ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex` `FETCH(r1, 1) @ r1<- BBBB` `ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses` `ldr r0, [r3, r1, lsl #2] @ r0<- resolved class` `EXPORT_PC() @ req'd for init, resolve, alloc`

The ’104 Reissue Patent	Infringed By
	cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.
when it is determined that the bytecode of the program contains a symbolic data reference, invoking a dynamic field reference routine to resolve the symbolic data reference; and	*See* Claim 11-b, *supra*.
executing thereafter the bytecode	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	

The ’104 Reissue Patent	Infringed By
27. A computer-implemented method comprising:	Android includes computer-implemented methods for performing the steps described in the claim. *See* Claim 11, *supra*.
receiving a program with a set of original instructions written in an intermediate form code;	*See* Claim 11-a, *supra*.
generating a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions; and	*See* Claim 11-b, *supra*.
executing the program using the set of new instructions.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
28. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a control program for causing the processor to (i) receive a program with a set of original instructions written in an intermediate form code, (ii) generate a set of new instructions for the	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions, and (iii) executing the program using the set of new instructions.	

The ’104 Reissue Patent	Infringed By
29. A computer program product containing instructions for causing a computer to perform a method, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform the steps described in the claim. *See* Claim 11, *supra*.
receiving a program with a set of original instructions written in an intermediate form code;	*See* Claim 11-a, *supra*.
generating a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions; and	*See* Claim 11, *supra*.
executing the program using the set of new instructions.	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
30. A computer-implemented method comprising:	Android includes computer-implemented methods for performing the steps described in the claim. *See* Claim 11, *supra*.
receiving a program that comprises [a set of instructions] written in an	*See* Claim 11-a, *supra*.

The ’104 Reissue Patent	Infringed By
intermediate form code;	
replacing each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference; and	*See* Claim 11, *supra*.
executing the program by performing an operation in accordance with each instruction or new instruction, depending upon whether an instruction has been replaced with a new instruction in accordance with the replacing step.	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
31. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a control program for causing the processor to (i) receive a program that comprises [a set of instructions] written in an intermediate form code, (ii) replace each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
invocation of a dynamic field reference [routine] to resolve the symbolic data reference, and (iii) execute the program by performing an operation in accordance with each instruction or new instruction, depending upon whether an instruction has been replaced with a new instruction in accordance with the replacing step.	

The ’104 Reissue Patent	Infringed By
32. A computer program product containing control instructions for causing a computer to perform a method, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform the steps described in the claim. *See* Claim 11, *supra*.
receiving a program that comprises [a set of instructions] written in an intermediate form code;	*See* Claim 11-a, *supra*.
replacing each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference; and	*See* Claim 11, *supra*.
executing the program by performing an operation in accordance with each instruction or new instruction, depending upon	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
whether an instruction has been replaced with a new instruction in accordance with the replacing step.	

The ’104 Reissue Patent	Infringed By
33. A computer-implemented method, comprising:	Android includes computer-implemented methods for performing the steps described in the claim. *See* Claim 11, *supra*.
receiving a program with [a set of instructions] written in an intermediate form code;	*See* Claim 11-a, *supra*.
analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and	*See* Claim 11-b, *supra*.
executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
34. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a control program for causing the processor to (i) receive a program with [a set of instructions] written in an intermediate form code, (ii) analyze each instruction of the program to determine whether the instruction contains a symbolic reference to a data object, and (iii) execute the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	*See* Claim 11, *supra*.

The ’104 Reissue Patent	Infringed By
35. A computer program product containing control instructions for causing a computer to perform a method, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform the steps described in the claim. *See* Claim 11, *supra*.
receiving a program with [a set of instructions] written in an intermediate form code;	*See* Claim 11-a, *supra*.

The ’104 Reissue Patent	Infringed By
analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and	*See* Claim 11-b, *supra*.
executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	*See* Claims 11-b and 16, *supra*.

The ’104 Reissue Patent	Infringed By
36. A computer-implemented method for executing a program comprised of bytecodes, the method comprising:	Android includes computer-implemented methods for performing the steps described in the claim. *See* Claims 11 and 16, *supra*.
determining whether a bytecode of the program contains a symbolic reference;	*See* Claims 11-b and 16, *supra*.
when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field reference routine to resolve the symbolic reference; and	*See* Claims 11-b and 16, *supra*.
performing an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from	*See* Claims 11-b and 16, *supra*.

The ’104 Reissue Patent	Infringed By
the invocation of the dynamic field reference routine.	

The ’104 Reissue Patent	Infringed By
37. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a program comprised of bytecodes and instructions for causing the processor to (i) determine whether a bytecode of the program contains a symbolic reference, (ii) when it is determined that the bytecode contains a symbolic reference, invoke a dynamic field reference routine to resolve the symbolic reference, and (iii) perform an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	*See* Claim 11-b, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class

The ’104 Reissue Patent	Infringed By
	mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also, e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By
38. A computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform the steps described in the claim. *See* Claim 11, *supra*.
determining whether a bytecode of the program contains a symbolic reference;	*See* Claim 11-b, *supra*.
when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
reference routine to resolve the symbolic reference; and	
performing an operation identified by the bytecode [thereafter] using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	*See* Claim 11-b, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */

The ’104 Reissue Patent	Infringed By
	.balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By
39. A computer-implemented method comprising:	Android includes computer-implemented methods for performing the steps described in the claim. *See* Claim 11, *supra*.
receiving a program formed of instructions written in an intermediate form code compiled from source code;	Lexical analysis and bytecode compilation is handled by javac. The bytecode is then further processed into .dex format by Android’s dx tool. *See* Claim 11-a, *supra*.
analyzing each instruction to determine whether it contains a symbolic field reference; and	*See* Claim 11-b, *supra*.
executing the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the	*See* Claim 11-b, *supra*. Also, *see*, *e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class.

The ’104 Reissue Patent	Infringed By
symbolic field reference.	``` */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction ``` *See also, e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By
40. A data processing system, comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are data processing systems. *See* Claim 11, *supra*.
a processor; and	*See* Claim 11, *supra*.
a memory comprising a control program for causing the processor to (i) receive a program formed of instructions written in an intermediate form code compiled from source code, (ii) analyze each instruction to determine whether it contains a symbolic field reference, and (iii) execute the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	*See* Claim 11, *supra*. Lexical analysis and bytecode compilation is handled by javac. The bytecode is then further processed into .dex format by Android’s dx tool. *See* Claim 11-a, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class

The ’104 Reissue Patent	Infringed By
	mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also, e.g.*, source files in dalvik\vm\mterp\out\.

The ’104 Reissue Patent	Infringed By
41. A computer program product containing control instructions for causing a computer to perform a method, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. An Android-based device and the Android SDK encompass a computer program product containing instructions for causing a computer to perform the steps described in the claim. *See* Claim 11, *supra*.
receiving a program formed of instructions written in an intermediate form code compiled from source code;	Lexical analysis and bytecode compilation is handled by javac. The bytecode is then further processed into .dex format by Android’s dx tool. *See* Claim 11-a, *supra*.
analyzing each instruction to determine whether it contains a symbolic field reference; and	*See* Claim 11-b, *supra*.

The ’104 Reissue Patent	Infringed By
executing the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is used thereafter for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	*See* Claim 11-b, *supra*. Also, *see, e.g.*, dalvik\vm\mterp\out\InterpAsm-armv5te.S: /* ------------------------------ */ .balign 64 .L_OP_NEW_INSTANCE: /* 0x22 */ /* File: armv5te/OP_NEW_INSTANCE.S */ /* * Create a new instance of a class. */ /* new-instance vAA, class@BBBB */ ldr r3, [rGLUE, #offGlue_methodClassDex] @ r3<- pDvmDex FETCH(r1, 1) @ r1<- BBBB ldr r3, [r3, #offDvmDex_pResClasses] @ r3<- pDvmDex->pResClasses ldr r0, [r3, r1, lsl #2] @ r0<- resolved class EXPORT_PC() @ req'd for init, resolve, alloc cmp r0, #0 @ already resolved? beq .LOP_NEW_INSTANCE_resolve @ no, resolve it now .LOP_NEW_INSTANCE_resolved: @ r0=class ldrb r1, [r0, #offClassObject_status] @ r1<- ClassStatus enum cmp r1, #CLASS_INITIALIZED @ has class been initialized? bne .LOP_NEW_INSTANCE_needinit @ no, init class now .LOP_NEW_INSTANCE_initialized: @ r0=class mov r1, #ALLOC_DONT_TRACK @ flags for alloc call bl dvmAllocObject @ r0<- new object b .LOP_NEW_INSTANCE_finish @ continue …. /* continuation for OP_NEW_INSTANCE */ .balign 32 @ minimize cache lines .LOP_NEW_INSTANCE_finish: @ r0=new object

The ’104 Reissue Patent	Infringed By
	mov r3, rINST, lsr #8 @ r3<- AA cmp r0, #0 @ failed? beq common_exceptionThrown @ yes, handle the exception FETCH_ADVANCE_INST(2) @ advance rPC, load rINST GET_INST_OPCODE(ip) @ extract opcode from rINST SET_VREG(r0, r3) @ vAA<- r0 GOTO_OPCODE(ip) @ jump to next instruction *See also*, *e.g.*, source files in dalvik\vm\mterp\out\.

<u>EXHIBIT B-1</u>
Preliminary Infringement Contentions for the ’205 Patent

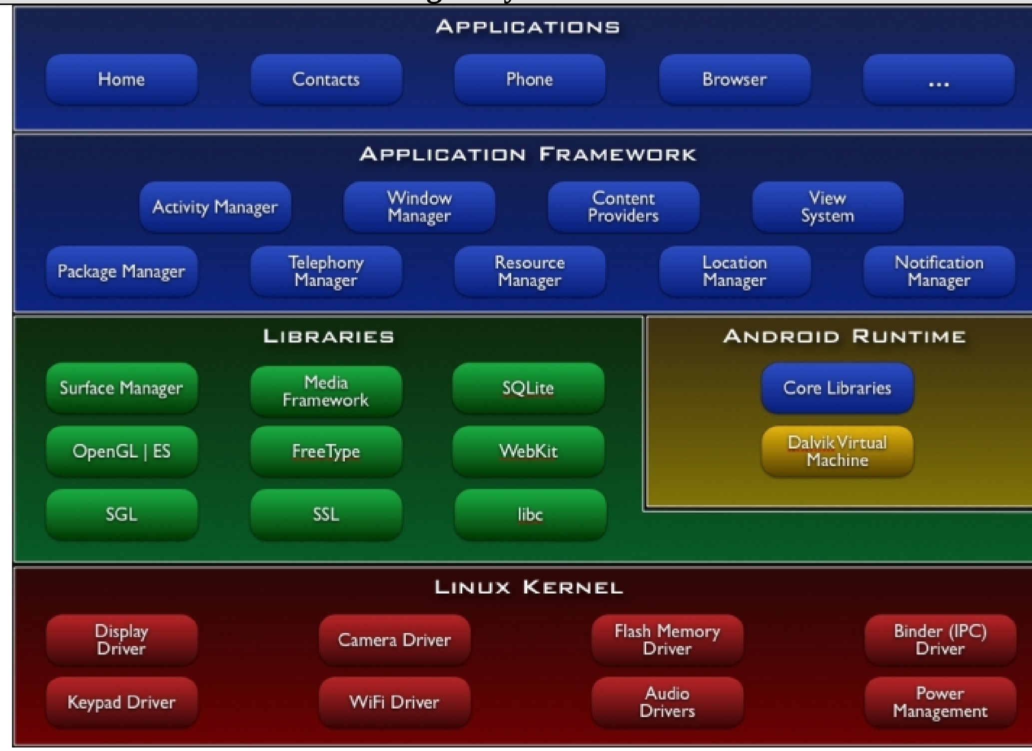
NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’205 patent is infringed by all versions of Android subsequent to January 28, 2010, including at least Android 2.2 (“Froyo”).

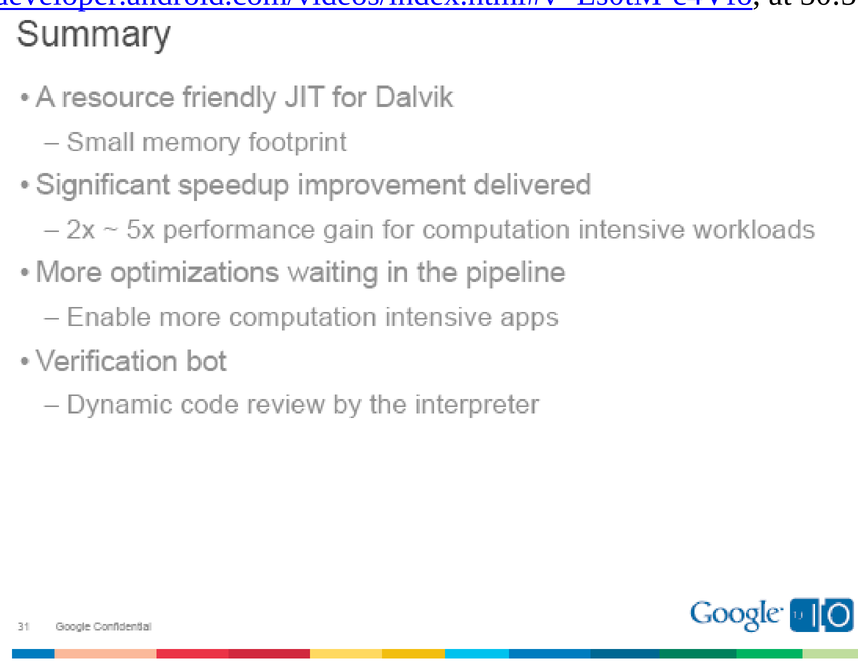
The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

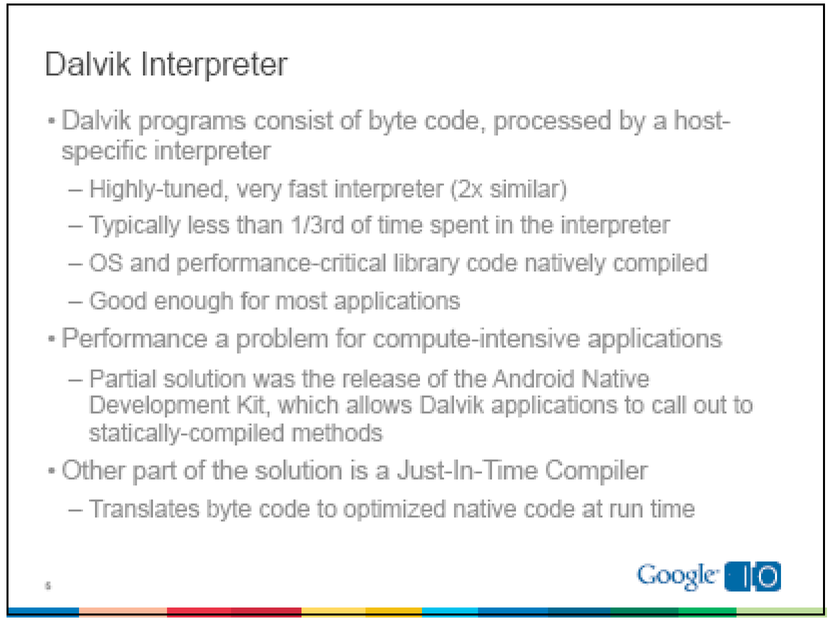
The ’205 Patent	Infringed By
[1-pre]1. In a computer system, a method for increasing the execution speed of virtual machine instructions at runtime, the method comprising:	Android uses the Dalvik virtual machine to execute virtual machine bytecode instructions at runtime. The Dalvik virtual machine performs certain optimizations to increase the execution speed of virtual machine instructions at runtime. *See, e.g.*, Android Glossary Definition for “Dalvik,” available at http://developer.android.com/guide/appendix/glossary.html: Dalvik The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management). The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a

The ’205 Patent	Infringed By
	small mobile device. Android Basics, entitled “What is Android?,” available at http://developer.android.com/guide/basics/what-is-android.html. **What is Android?** Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. **Features** • Application framework enabling reuse and replacement of components • Dalvik virtual machine optimized for mobile devices • … **Android Architecture** The following diagram shows the major components of the Android operating system. Each section is described in more detail below.

The ’205 Patent	Infringed By
	 **Applications** Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. … **Android Runtime** Android includes a set of core libraries that provides most of the functionality available

The ’205 Patent	Infringed By
	in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool. … Google I/O 2008 Video, entitled “A JIT Compiler for Android's Dalvik VM,” presented by Ben Cheng and Bill Buzbee (Google’s Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo, at 50:53:  Okay. So to wrap up today's talk, over the past year, our team has delivered a resource-

The '205 Patent	Infringed By
	friendly JIT for the Dalvik version machine from ground up. We pay special attention to the memory overhead, so that it can fit the budget on embedded systems. And through a set of CPU intensive workloads, we demonstrated that it can provide 2x to 5x speedup over the Eclair release. And we already have new optimizations waiting in the pipeline, and we believe that JIT will enable a new class of applications for the Android platform.
[1-a]receiving a first virtual machine instruction;	Android's Dalvik virtual machine receives a first virtual machine instruction. *See, e.g.*, Google I/O 2010 Video, entitled "A JIT Compiler for Android's Dalvik VM," presented by A JIT Compiler for Android's Dalvik VM, presented by Ben Cheng and Bill Buzbee (Google's Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo: • at 3:35: Now at the center of every Dalvik virtual machine implementation is an instruction at a time interpreter. Now, interpreters, the way they work, they will go out and fetch one Dalvik instruction at a time – we call them Dalvik bytecodes – pull the instruction apart, see what it is, that's called the decode phase, and then go ahead and interpret it or execute it. And that execution is done by using actually the host instructions on the host processor. But what you have in effect there with interpretation is an extra stage of execution. So you have to pull up the bytecode, figure out what it is, then use host instructions to carry it out, and the CPU will pick up the host instructions and execute them. And that extra level of evaluation is what gives interpreters a bit of a bad name for being slow. I mean, there's some great reasons why you would want to use an interpreter, but the down side is that they are often a bit slower than natively compiled code. Now, we didn't think that -- We think that sometimes this interpretation gets more of a bad rap than it really should in an Android system, and there are several reasons for that. First of all, our interpreter is really, really fast. It's a very well done interpreter. We had one of our partners benchmark our Dalvik interpreter against a traditional Java interpreter, and they told us that the Dalvik interpreter was roughly twice as fast, which we were happy to hear. But the other and perhaps more important reason is that not everything -- when your application is running on an Android system, it's not really in

The ’205 Patent	Infringed By
	interpretation the whole time. The system itself has already been natively compiled and optimized. A lot of the libraries, the really key libraries, graphics and other things that you need to run fast, are done in native code already, so it's really just sometimes a smallish amount of code that's actually being interpreted when your program runs.  *See, e.g.*, dalvik\vm\interp\Jit.c: /* * Adds to the current trace request one instruction at a time, just * before that instruction is interpreted. This is the primary trace * selection function. NOTE: return instruction are handled a little * differently. In general, instructions are "proposed" to be added * to the current trace prior to interpretation. If the interpreter * then successfully completes the instruction, is will be considered * part of the request. This allows us to examine machine state prior * to interpretation, and also abort the trace request if the instruction

The ’205 Patent	Infringed By
	`` * throws or does something unexpected. However, return instructions * will cause an immediate end to the translation request - which will * be passed to the compiler before the return completes. This is done * in response to special handling of returns by the interpreter (and * because returns cannot throw in a way that causes problems for the * translated code. */ int dvmCheckJit(const u2* pc, Thread* self, InterpState* interpState) { … /* Prepare to handle last PC and stage the current PC */ const u2 *lastPC = interpState->lastPC; interpState->lastPC = pc; switch (interpState->jitState) { char* nopStr; int target; int offset; DecodedInstruction decInsn; case kJitTSelect: /* First instruction - just remember the PC and exit */ if (lastPC == NULL) break; /* Grow the trace around the last PC if jitState is kJitTSelect */ dexDecodeInstruction(gDvm.instrFormat, lastPC, &decInsn); /* * Treat {PACKED,SPARSE}_SWITCH as trace-ending instructions due * to the amount of space it takes to generate the chaining * cells. */ if (interpState->totalTraceLen != 0 && ```

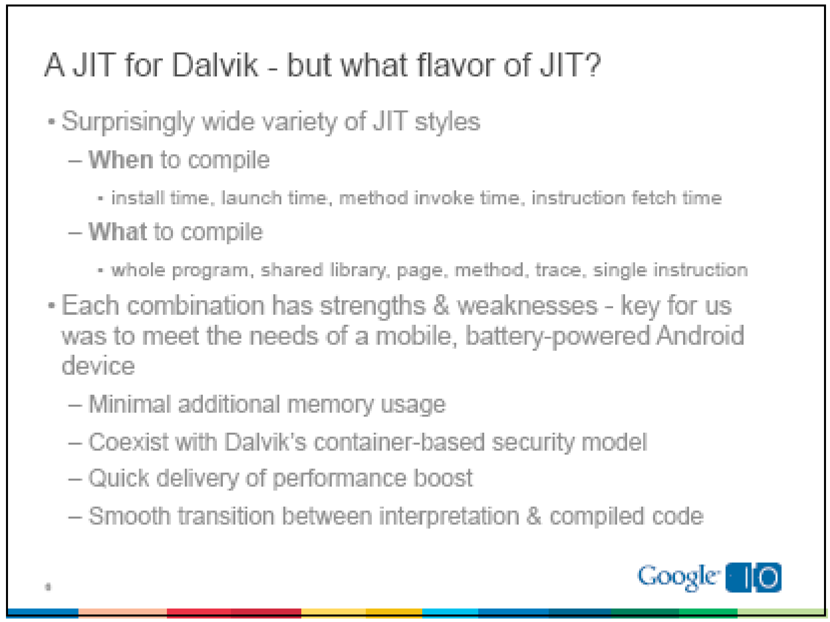
The ’205 Patent	Infringed By
	`` (decInsn.opCode == OP_PACKED_SWITCH || decInsn.opCode == OP_SPARSE_SWITCH)) { interpState->jitState = kJitTSelectEnd; break; } … interpState->trace[interpState->currTraceRun].frag.runEnd = true; desc->method = interpState->method; memcpy((char*)&(desc->trace[0]), (char*)&(interpState->trace[0]), sizeof(JitTraceRun) * (interpState->currTraceRun+1)); #if defined(SHOW_TRACE) LOGD("TraceGen: trace done, adding to queue"); #endif if (dvmCompilerWorkEnqueue(interpState->currTraceHead,kWorkOrderTrace,desc)) { /* Work order successfully enqueued */ if (gDvmJit.blockingMode) { dvmCompilerDrainQueue(); } } else { /* * Make sure the descriptor for the abandoned work order is * freed. */ free(desc); } /* * Reset "trace in progress" flag whether or not we * successfully entered a work order. */ ```

The ’205 Patent	Infringed By
	`JitEntry *jitEntry =` `   lookupAndAdd(interpState->currTraceHead, false);` `if (jitEntry) {` `   setTraceConstruction(jitEntry, false);` `}` `interpState->jitState = kJitDone;` `switchInterp = true;` `}` `break;` `…`
[1-b]generating, at runtime, a new virtual machine instruction that represents or references one or more native instructions that can be executed instead of said first virtual machine instruction; and	Android’s Dalvik virtual machine has a just-in-time (“JIT”) compiler, which serves to generate at runtime a new virtual machine instruction that represents or references one or more native instructions that can be executed instead of said first virtual machine instruction. *See, e.g.*, Google I/O 2010 Video, entitled “A JIT Compiler for Android's Dalvik VM,” presented by *A JIT Compiler for Android’s Dalvik VM*, presented by Ben Cheng and Bill Buzbee (Google’s Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo: • at 6:09: So if you translate a Dalvik bytecode into say underlying ARM instructions, you get a significant code expansion. So keeping it Dalvik is actually a good idea. But this good enough for most applications doesn't mean it's perfect. For some applications, you really, really do feel the pain of interpretation. Applications in which you do a lot of computation. And that gets painful, because you experience the slowdown of the interpreter, which is often on the order of five to ten times. Now, we have had two strategies in play -- or two strategies planned to deal with this slowdown that computation intensive programs would face. The first of those, the result we came out last year with the release of the Android NDK or native development kit. This was a software development kit that makes it easier to isolate the compute intensive portions of your program, rewrite those in a natively compiled language and then call them from your Android application. The other part of the solution is what we have announced today, and that's the just-in-

The ’205 Patent	Infringed By

time compiler. A just-in-time compiler still has an interpreter involved. The interpreter will interpret your program until it identifies what's the most compute intensive part of it. What's the really hot chunks of your program. And then it will pull those out, compile them and optimize them into native code so the next time you invoke that section of code, you are not doing interpretation anymore, you are just doing direct execution of the native code.

- at 7:44:

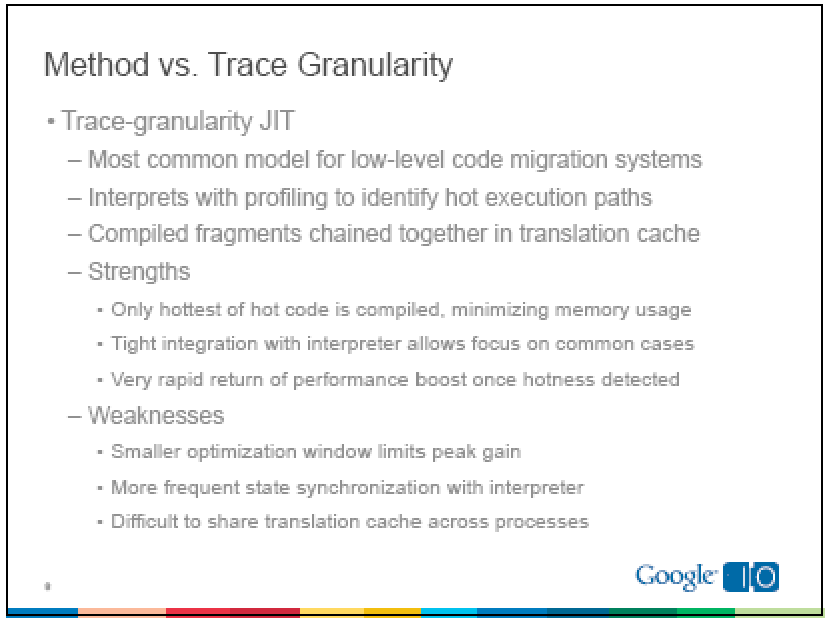


Now, putting a JIT on Android has been something we have talked about for a long time internally, but the big question was what kind of a JIT can we fit into an Android system?

As it turns out, the JIT design space is pretty large. I mean, with the popularity of Java, most people now are familiar with JITs. And generally, a particular style of JIT, but actually, there's quite a broad design field. The way we like to think of it is you can break it down into -- break the JIT design down into two axes. Along one axis would be what does it mean for just-in-time. When do you do the compilation? You could do

The ’205 Patent	Infringed By
	it when you first install the application or maybe you do it the first time a method is called or when you page in some code off of disk. The other axis is what is the unit of compilation? Do you compile the whole program, a whole shared library, just a physical page of code? Or maybe you do a method or a string of instructions or even a single instruction’s worth of code at once. Looking back over the last 20 years or so with just-in-time compilers and dynamic translation systems, you could actually pretty much fill in every square of that matrix about the chunk of thing were you compiling and when it was you were doing the compilation. And it's not really the case that any one combination is best. Each combination had a set of characteristics that were good in some situations, and bad in another. And what our trick -- the trick for us was to find the combination that worked really well in a portable memory constrained device. So our key requirements going into this process, we needed to find a JIT system that could deliver performance using a very small amount of memory. From all the talks today you will hear about how important memory is on these small devices, and I think you know this. So we couldn't have a JIT that wasted a lot of memory or took too much. Then next, it had to co-exist with this processor container security model. And this -- I won't get into this too much, but a lot of JIT styles would have information being bled between processes, and we needed to avoid that. And finally, the last two ones are something that we considered really important for this type of device. We wanted the performance that were going to be delivered by the just-in-time compilation to come to the user quickly. We didn't want somebody to have an application that they had to run to warm it up for, you know, a minute, an hour, a day. We wanted them to get the boost from compilation as soon as possible. And finally, we're kind of sensitive about this is an interactive device, so we are sensitive to jerky execution and pausing. We wanted the transition between interpretation and compiled code to be really smooth. • at 13:14:

<table>
<tr><th>The ’205 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



So the other style, the trace-based JIT, this is a style that a lot of people hadn't heard much about, but it actually is really popular.

It is the style of JIT that is typically used when you are doing code migrations or virtualizations of one architecture on another. This was very popular back in the 90's especially. What it does is very similar to the method-based JIT, is you start off interpreting, but you interpret until you find out what the hot chunks of code are. But in this case, the chunks are not contained methods. They are actually just a run of instructions that will start someplace, you will execute for a while, maybe you will follow a branch or two, perhaps you will identify a loop, and it will pull out just those instructions, straighten them out into a straight line trace of code, and then optimize that straight line trace of code.

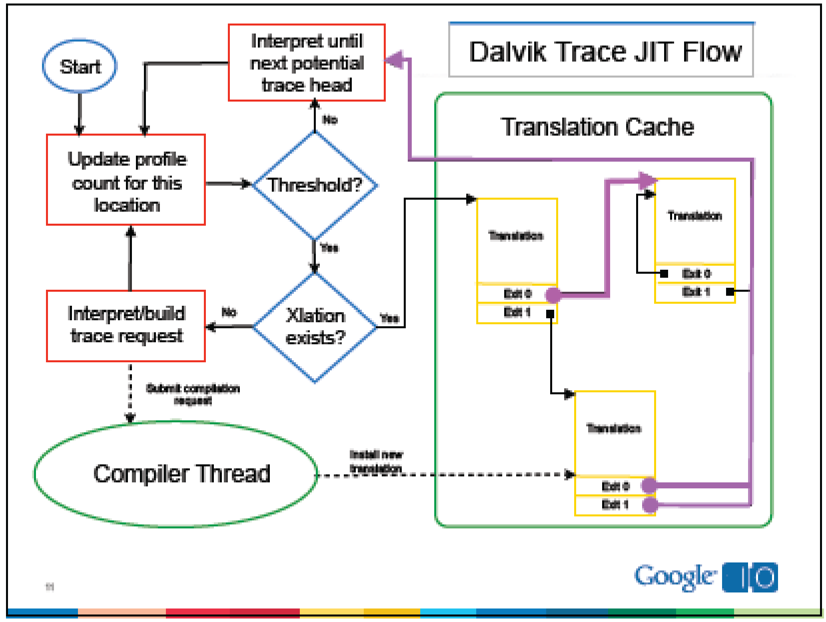
It will take these translated chunks and store them in a translation cache, and chain them all together so that execution will kind of bounce from one trace to another.

And this type of trace formation, you don't actually even need to respect method call boundaries. You can even have a trace that goes through a method call.

</td></tr>
</table>

The ’205 Patent	Infringed By
	The great strength of this one is that you are only optimizing the hottest of the hot code. It really has to be the code that's running before you bother to put the resources in to compiling it. Another benefit that's a little bit more subtle, but quite useful, is that this type of JIT system is typically more tightly integrated with the interpreter. So you are bouncing between the interpreter and the translated code a little bit more frequently, but what it also means is that you are never very far away from the interpreter. So in the translated code, you can arrange it, if you choose, to have it that the translated code doesn't have to deal with any exceptional cases. If it detects that some assumption that's made has gone wrong or it sees there's a null pointer here I have to deal with, it doesn't actually have to deal with it. It can just roll back the state, return to the interpreter and let the interpreter handle all the messy nastiness associated with handling these unusual cases. And this actually turns out to be pretty powerful in allowing a trace compiler to be simpler and focus really on what's going to return the performance rather than all the details about handling every possible corner case. But, of course, still being correct. Finally, you get a really rapid return on investment. The performance comes back quickly. You are compiling little small chunks. You don't have to wait very long before you decide some chunk is hot and you can stitch it right into an application and get the boost from that compilation right away. Now, as with everything, there are down sides. • at 19:39:

The ’205 Patent	Infringed By

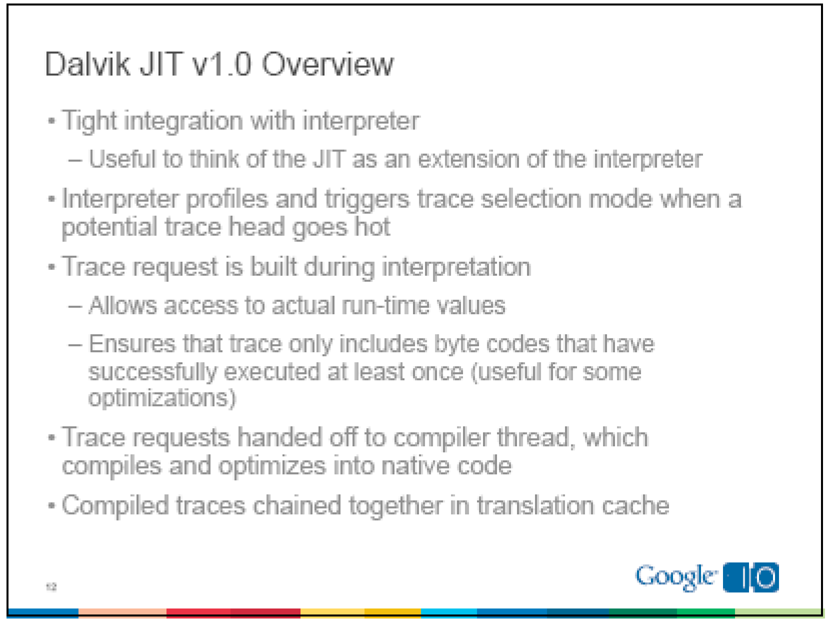


Okay, so time for a flowchart.
Let's just walk through very quickly kind of a simplified version of how the trace JIT works. So when I mentioned that a trace is a string of instructions, kind of a straight-line string of instructions.
And that has to have a start. We call the potential start of any trace to be the trace head. That's, you know, the point in the code that the trace can begin from. Let's say we're in the interpreter and let's say we're at one of these trace head points. Now, that's generally the target of a backward branch, the entry point to a method, the target of an indirect branch.
There are several possibilities. The interpreter will say, hey, I'm at a potential trace head, and so it will increment a profile counter associated with that potential head. Then it will ask itself, have I been here enough that this thing matters?
In the beginning, the answer will be no. So the interpreter will go back to interpreting as fast as it can, until it comes to the next potential trace head.
It will update its profile counter, ask the question again, have I reached my threshold?

The ’205 Patent	Infringed By
	Is this something that's interesting? And eventually, the answer will come back, yes, this is interesting. I've been here enough. This is something I want to take a look at. The next question is asked, do I already have a translation for this address? And if I do, then will this send execution directly to that translation so it can start translating natively? But, again, let's say we're in the beginning of the world. The answer would be no. So we don't have a translation for that address, so we want to build one. So we go into -- we go back to the interpreter. So we're going to continue interpreting, but we're going to continue interpreting in a special mode. We call this trace-building mode. We essentially single-step the interpreter, and every time we successfully interpret an instruction, we'll add that instruction to the list of instructions for that trace that we want to have translated. Now, how long we keep doing that and when we stop, that's one of the tuning parameters we have in the JIT. Basically, you know, how many branches do you follow before your trace is terminated? At some point, we'll decide, okay, this is long enough. We want to terminate this trace. And then we'll send the request off to the compiler thread. Meanwhile, the interpreter goes back to interpreting so you can continue making forward progress. Now, at some point, the compiler thread will get around to compiling that trace into a sequence of native instructions, and it will install it into the translation cache. Now, one of the -- something to keep in mind, when we first put a translation in the translation cache, it's going to have some -- all traces are going to have branches that exit the trace. When we first put it in the translation cache, those branches that exit the trace are going to be hard-wired to send us back to the interpreter. So if the very first time we jump in, we do the translation, then we get back to the interpreter and go look for new hot traces to compile. But on the way out, once we have completed that trace and we're going back to the interpreter, the question will be asked, is there a translation for where I'm going? And if the answer is yes, then we'll replace that exit branch with a direct trace-to-trace branch. The upshot of this is that, in practice, very quickly, we spend very little time in the interpreter. We'll collect all of our hot traces, chain them all together, and so we're really just bouncing from trace to trace to trace to trace in the translation cache, with an

<table>
<tr><th>The ’205 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>
occasional bounce out to the interpreter to find some other hot trace before we come back in to the translation cache.
It works remarkably well. Okay.

at 23:16:



at 48:49:
Since we manage the code cache by not only storing the generated code but also the original trace information, we can easily replay the compilation request with verbose mode turned on so that you can see the Dalvik code and the corresponding native code. And that's exactly the same mechanism used by the profiler to report the content of the hot translations.
at Questions and Answers:
Ben Cheng: So the next one is how well does the JIT compiler use the native processor?
Does it produce generic ARMv 5 code or is it smart enough to optimize on ARMv9 with neon extensions.

</td></tr>
</table>

The ’205 Patent	Infringed By
	Bill Buzbee: I think we answered that one. It's configurable. Android source code confirms the same. *See, e.g.*, dalvik\vm\native\dalvik_system_VMRuntime.c: /* * public native void startJitCompilation() * * Callback function from the framework to indicate that an app has gone * through the startup phase and it is time to enable the JIT compiler. */ static void Dalvik_dalvik_system_VMRuntime_startJitCompilation(const u4* args, JValue* pResult) { #if defined(WITH_JIT) if (gDvm.executionMode == kExecutionModeJit && gDvmJit.disableJit == false) { … pthread_cond_signal(&gDvmJit.compilerQueueActivity); … } #endif RETURN_VOID(); } dalvik\vm\compiler\Compiler.c: bool dvmCompilerSetupCodeCache(void) { … /* Allocate the code cache */

The ’205 Patent	Infringed By
	`` fd = ashmem_create_region("dalvik-jit-code-cache", gDvmJit.codeCacheSize); … gDvmJit.codeCache = mmap(NULL, gDvmJit.codeCacheSize, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_PRIVATE , fd, 0); … /* Copy the template code into the beginning of the code cache */ int templateSize = (intptr_t) dmvCompilerTemplateEnd - (intptr_t) dvmCompilerTemplateStart; memcpy((void *) gDvmJit.codeCache, (void *) dvmCompilerTemplateStart, templateSize); … } … static void *compilerThreadStart(void *arg) { … /* * If we're not running stand-alone, wait a little before * recieving translation requests on the assumption that process start * up code isn't worth compiling. We'll resume when the framework * signals us that the first screen draw has happened, or the timer * below expires (to catch daemons). * * There is a theoretical race between the callback to * VMRuntime.startJitCompiation and when the compiler thread reaches this * point. In case the callback happens earlier, in order not to permanently * hold the system_server (which is not using the timed wait) in * interpreter-only mode we bypass the delay here. ```

The ’205 Patent	Infringed By
	`` */ if (gDvmJit.runningInAndroidFramework && !gDvmJit.alreadyEnabledViaFramework) { /* * If the current VM instance is the system server (detected by having * 0 in gDvm.systemServerPid), we will use the indefinite wait on the * conditional variable to determine whether to start the JIT or not. * If the system server detects that the whole system is booted in * safe mode, the conditional variable will never be signaled and the * system server will remain in the interpreter-only mode. All * subsequent apps will be started with the --enable-safemode flag * explicitly appended. */ if (gDvm.systemServerPid == 0) { dvmLockMutex(&gDvmJit.compilerLock); pthread_cond_wait(&gDvmJit.compilerQueueActivity, &gDvmJit.compilerLock); dvmUnlockMutex(&gDvmJit.compilerLock); LOGD("JIT started for system_server"); } else { dvmLockMutex(&gDvmJit.compilerLock); /* * TUNING: experiment with the delay & perhaps make it * target-specific */ dvmRelativeCondWait(&gDvmJit.compilerQueueActivity, &gDvmJit.compilerLock, 3000, 0); dvmUnlockMutex(&gDvmJit.compilerLock); } … } ```

The ’205 Patent	Infringed By
	`` compilerThreadStartup(); … /* * Since the compiler thread will not touch any objects on the heap once * being created, we just fake its state as VMWAIT so that it can be a * bit late when there is suspend request pending. */ while (!gDvmJit.haltCompilerThread) { if (workQueueLength() == 0) { int cc; cc = pthread_cond_signal(&gDvmJit.compilerQueueEmpty); assert(cc == 0); pthread_cond_wait(&gDvmJit.compilerQueueActivity, &gDvmJit.compilerLock); continue; } else { do { CompilerWorkOrder work = workDequeue(); … /* * Check whether there is a suspend request on me. This * is necessary to allow a clean shutdown. * * However, in the blocking stress testing mode, let the * compiler thread continue doing compilations to unblock * other requesting threads. This may occasionally cause * shutdown from proceeding cleanly in the standalone invocation * of the vm but this should be acceptable. */ … if (gDvmJit.haltCompilerThread) { LOGD("Compiler shutdown in progress - discarding request"); ```

The ’205 Patent	Infringed By
	`` } else if (!gDvmJit.codeCacheFull) { … if (!aborted) { compileOK = dvmCompilerDoWork(&work); } if (aborted || !compileOK) { dvmCompilerArenaReset(); work.result.codeAddress = gDvmJit.interpretTemplate; } else if (!work.result.discardResult) { dvmJitSetCodeAddr(work.pc, work.result.codeAddress, work.result.instructionSet); } } … } bool dvmCompilerStartup(void) { … pthread_cond_init(&gDvmJit.compilerQueueActivity, NULL); pthread_cond_init(&gDvmJit.compilerQueueEmpty, NULL); /* Reset the work queue */ gDvmJit.compilerWorkEnqueueIndex = gDvmJit.compilerWorkDequeueIndex = 0; gDvmJit.compilerQueueLength = 0; dvmUnlockMutex(&gDvmJit.compilerLock); /* * Defer rest of initialization until we're sure JIT'ng makes sense. Launch * the compiler thread, which will do the real initialization if and ```

The ’205 Patent	Infringed By
	`` * when it is signalled to do so. */ return dvmCreateInternalThread(&gDvmJit.compilerHandle, "Compiler", compilerThreadStart, NULL); } dalvik\vm\interp\Jit.h: /* * Entries in the JIT's address lookup hash table. * Fields which may be updated by multiple threads packed into a * single 32-bit word to allow use of atomic update. */ typedef struct JitEntryInfo { unsigned int traceConstruction:1; /* build underway? */ unsigned int isMethodEntry:1; unsigned int inlineCandidate:1; unsigned int profileEnabled:1; JitInstructionSetType instructionSet:4; unsigned int unused:8; u2 chain; /* Index of next in chain */ } JitEntryInfo; typedef union JitEntryInfoUnion { JitEntryInfo info; volatile int infoWord; } JitEntryInfoUnion; typedef struct JitEntry { JitEntryInfoUnion u; const u2* dPC; /* Dalvik code address */ void* codeAddress; /* Code address of native translation */ ```

The ’205 Patent	Infringed By
	`` } JitEntry; ``` dalvik\vm\interp\Jit.c: ``` /* * Find an entry in the JitTable, creating if necessary. * Returns null if table is full. */ static JitEntry *lookupAndAdd(const u2* dPC, bool callerLocked) { … u4 idx = dvmJitHash(dPC); /* Walk the bucket chain to find an exact match for our PC */ while ((gDvmJit.pJitEntryTable[idx].u.info.chain != chainEndMarker) && (gDvmJit.pJitEntryTable[idx].dPC != dPC)) { idx = gDvmJit.pJitEntryTable[idx].u.info.chain; } if (gDvmJit.pJitEntryTable[idx].dPC != dPC) { /* * No match. Aquire jitTableLock and find the last * slot in the chain. Possibly continue the chain walk in case * some other thread allocated the slot we were looking * at previuosly (perhaps even the dPC we're trying to enter). */ … do { oldValue = gDvmJit.pJitEntryTable[prev].u; newValue = oldValue; newValue.info.chain = idx; ```

The ’205 Patent	Infringed By
	} while (!ATOMIC_CMP_SWAP(&gDvmJit.pJitEntryTable[prev].u.infoWord, oldValue.infoWord, newValue.infoWord)); } } if (gDvmJit.pJitEntryTable[idx].dPC == NULL) { /* * Initialize codeAddress and allocate the slot. Must * happen in this order (since dPC is set, the entry is live. */ gDvmJit.pJitEntryTable[idx].dPC = dPC; gDvmJit.jitTableEntriesUsed++; } … return (idx == chainEndMarker) ? NULL : &gDvmJit.pJitEntryTable[idx]; } dalvik\vm\interp\Jit.c: /* * Adds to the current trace request one instruction at a time, just * before that instruction is interpreted. This is the primary trace * selection function. NOTE: return instruction are handled a little * differently. In general, instructions are "proposed" to be added * to the current trace prior to interpretation. If the interpreter * then successfully completes the instruction, is will be considered * part of the request. This allows us to examine machine state prior * to interpretation, and also abort the trace request if the instruction * throws or does something unexpected. However, return instructions * will cause an immediate end to the translation request - which will * be passed to the compiler before the return completes. This is done * in response to special handling of returns by the interpreter (and * because returns cannot throw in a way that causes problems for the

The ’205 Patent	Infringed By
	`` * translated code. */ int dvmCheckJit(const u2* pc, Thread* self, InterpState* interpState) { … /* Prepare to handle last PC and stage the current PC */ const u2 *lastPC = interpState->lastPC; interpState->lastPC = pc; switch (interpState->jitState) { char* nopStr; int target; int offset; DecodedInstruction decInsn; case kJitTSelect: /* First instruction - just remember the PC and exit */ … JitEntry *dvmFindJitEntry(const u2* pc) { int idx = dvmJitHash(pc); /* Expect a high hit rate on 1st shot */ if (gDvmJit.pJitEntryTable[idx].dPC == pc) return &gDvmJit.pJitEntryTable[idx]; else { int chainEndMarker = gDvmJit.jitTableSize; while (gDvmJit.pJitEntryTable[idx].u.info.chain != chainEndMarker) { idx = gDvmJit.pJitEntryTable[idx].u.info.chain; if (gDvmJit.pJitEntryTable[idx].dPC == pc) return &gDvmJit.pJitEntryTable[idx]; ```

The ’205 Patent	Infringed By
	`` } } return NULL; } ``` dalvik\vm\interp\Jit.c: ``` /* * If a translated code address exists for the davik byte code * pointer return it. This routine needs to be fast. */ void* dvmJitGetCodeAddr(const u2* dPC) { int idx = dvmJitHash(dPC); const u2* npc = gDvmJit.pJitEntryTable[idx].dPC; … if (npc == dPC) { … return hideTranslation ? NULL : gDvmJit.pJitEntryTable[idx].codeAddress; } else { int chainEndMarker = gDvmJit.jitTableSize; while (gDvmJit.pJitEntryTable[idx].u.info.chain != chainEndMarker) { idx = gDvmJit.pJitEntryTable[idx].u.info.chain; if (gDvmJit.pJitEntryTable[idx].dPC == dPC) { … return hideTranslation ? NULL : gDvmJit.pJitEntryTable[idx].codeAddress; } } } } ```

The ’205 Patent	Infringed By
	<pre>… return NULL; } /* * Register the translated code pointer into the JitTable. * NOTE: Once a codeAddress field transitions from initial state to * JIT'd code, it must not be altered without first halting all * threads. This routine should only be called by the compiler * thread. */ void dvmJitSetCodeAddr(const u2* dPC, void *nPC, JitInstructionSetType set) { JitEntryInfoUnion oldValue; JitEntryInfoUnion newValue; JitEntry *jitEntry = lookupAndAdd(dPC, false); assert(jitEntry); /* Note: order of update is important */ do { oldValue = jitEntry->u; newValue = oldValue; newValue.info.instructionSet = set; } while (!ATOMIC_CMP_SWAP(&jitEntry->u.infoWord, oldValue.infoWord, newValue.infoWord)); jitEntry->codeAddress = nPC; }</pre> dalvik\vm\compiler\codegen\arm\CodegenDriver.c: <pre>/* Accept the work and start compiling */ bool dvmCompilerDoWork(CompilerWorkOrder *work) { bool res;</pre>

The ’205 Patent	Infringed By
	<pre>… switch (work->kind) { case kWorkOrderMethod: res = dvmCompileMethod(work->info, &work->result); break; case kWorkOrderTrace: /* Start compilation with maximally allowed trace length */ res = dvmCompileTrace(work->info, JIT_MAX_TRACE_LEN, &work->result, work->bailPtr); break; … } … } return res; }</pre> *See generally*, *e.g.*, dalvik\vm\compiler\codegen\arm\CodegenDriver.c. *E.g.:* <pre>static bool genArithOpInt(CompilationUnit *cUnit, MIR *mir, RegLocation rlDest, RegLocation rlSrc1, RegLocation rlSrc2) { OpKind op = kOpBkpt; … switch (mir->dalvikInsn.opCode) { case OP_NEG_INT: op = kOpNeg; unary = true; break; case OP_NOT_INT:</pre>

The ’205 Patent	Infringed By
	`` op = kOpMvn; unary = true; break; case OP_ADD_INT: … } if (!callOut) { rlSrc1 = loadValue(cUnit, rlSrc1, kCoreReg); if (unary) { rlResult = dvmCompilerEvalLoc(cUnit, rlDest, kCoreReg, true); opRegReg(cUnit, op, rlResult.lowReg, rlSrc1.lowReg); } else { rlSrc2 = loadValue(cUnit, rlSrc2, kCoreReg); if (shiftOp) { int tReg = dvmCompilerAllocTemp(cUnit); opRegRegImm(cUnit, kOpAnd, tReg, rlSrc2.lowReg, 31); rlResult = dvmCompilerEvalLoc(cUnit, rlDest, kCoreReg, true); opRegRegReg(cUnit, op, rlResult.lowReg, rlSrc1.lowReg, tReg); dvmCompilerFreeTemp(cUnit, tReg); } else { rlResult = dvmCompilerEvalLoc(cUnit, rlDest, kCoreReg, true); opRegRegReg(cUnit, op, rlResult.lowReg, rlSrc1.lowReg, rlSrc2.lowReg); } } storeValue(cUnit, rlDest, rlResult); } else { RegLocation rlResult; dvmCompilerFlushAllRegs(cUnit); /* Send everything to home location */ loadValueDirectFixed(cUnit, rlSrc2, r1); ```

The ’205 Patent	Infringed By
	LOAD_FUNC_ADDR(cUnit, r2, (int) callTgt); loadValueDirectFixed(cUnit, rlSrc1, r0); if (checkZero) { genNullCheck(cUnit, rlSrc2.sRegLow, r1, mir->offset, NULL); } opReg(cUnit, kOpBlx, r2); dvmCompilerClobberCallRegs(cUnit); if (retReg == r0) rlResult = dvmCompilerGetReturn(cUnit); else rlResult = dvmCompilerGetReturnAlt(cUnit); storeValue(cUnit, rlDest, rlResult); } return false; } *See also, e.g.*, dalvik\vm\compiler\codegen\arm\Thumb\Factory.c: static ArmLIR *opRegRegImm(CompilationUnit *cUnit, OpKind op, int rDest, int rSrc1, int value) { ArmLIR *res; bool neg = (value < 0); int absValue = (neg) ? -value : value; ArmOpCode opCode = kThumbBkpt; bool shortForm = (absValue & 0x7) == absValue; switch(op) { case kOpAdd: if (rDest == rSrc1) return opRegImm(cUnit, op, rDest, value); if ((rSrc1 == 13) && (value <= 1020)) { /* sp */ assert((value & 0x3) == 0); shortForm = true;

The ’205 Patent	Infringed By
	`` opCode = kThumbAddSpRel; value >>= 2; } else if ((rSrc1 == 15) && (value <= 1020)) { /* pc */ assert((value & 0x3) == 0); shortForm = true; opCode = kThumbAddPcRel; value >>= 2; } else if (shortForm) { opCode = (neg) ? kThumbSubRRI3 : kThumbAddRRI3; } else if ((absValue > 0) && (absValue <= (255 + 7))) { /* Two shots - 1st handle the 7 */ opCode = (neg) ? kThumbSubRRI3 : kThumbAddRRI3; res = newLIR3(cUnit, opCode, rDest, rSrc1, 7); opCode = (neg) ? kThumbSubRI8 : kThumbAddRI8; newLIR2(cUnit, opCode, rDest, absValue - 7); return res; } else opCode = kThumbAddRRR; break; case kOpSub: if (rDest == rSrc1) return opRegImm(cUnit, op, rDest, value); if (shortForm) { opCode = (neg) ? kThumbAddRRI3 : kThumbSubRRI3; } else if ((absValue > 0) && (absValue <= (255 + 7))) { /* Two shots - 1st handle the 7 */ opCode = (neg) ? kThumbAddRRI3 : kThumbSubRRI3; res = newLIR3(cUnit, opCode, rDest, rSrc1, 7); opCode = (neg) ? kThumbAddRI8 : kThumbSubRI8; newLIR2(cUnit, opCode, rDest, absValue - 7); return res; ```

The ’205 Patent	Infringed By
	`` } else opCode = kThumbSubRRR; break; case kOpLsl: shortForm = (!neg && value <= 31); opCode = kThumbLslRRI5; break; case kOpLsr: shortForm = (!neg && value <= 31); opCode = kThumbLsrRRI5; break; case kOpAsr: shortForm = (!neg && value <= 31); opCode = kThumbAsrRRI5; break; case kOpMul: case kOpAnd: case kOpOr: case kOpXor: if (rDest == rSrc1) { int rScratch = dvmCompilerAllocTemp(cUnit); res = loadConstant(cUnit, rScratch, value); opRegReg(cUnit, op, rDest, rScratch); } else { res = loadConstant(cUnit, rDest, value); opRegReg(cUnit, op, rDest, rSrc1); } return res; default: LOGE("Jit: bad case in opRegRegImm"); dvmCompilerAbort(cUnit); break; ```

The ’205 Patent	Infringed By
	`` } if (shortForm) res = newLIR3(cUnit, opCode, rDest, rSrc1, absValue); else { if (rDest != rSrc1) { res = loadConstant(cUnit, rDest, value); newLIR3(cUnit, opCode, rDest, rSrc1, rDest); } else { int rScratch = dvmCompilerAllocTemp(cUnit); res = loadConstant(cUnit, rScratch, value); newLIR3(cUnit, opCode, rDest, rSrc1, rScratch); } } return res; } ``` *See* Claim 1-c, *infra*. To the extent Android does not literally infringe this claim element, Android contains equivalent elements corresponding to each and every requirement of this claim limitation. When the Android JIT compiles a trace, Android adds the corresponding bytecode instruction counter of the bytecode (the first virtual machine instruction) and a pointer to the compiled trace to the jitEntry table. When interpreting the instruction located at the bytecode instruction counter, Android does a lookup of the bytecode instruction counter in the jitEntry table. If Android finds an entry, Android will execute a branch to the compiled trace instead of executing the bytecode instruction. The differences, if any, between a “new virtual machine instruction” and an entry in the jitEntry table are insubstantial. An entry in the jitEntry table (1) performs the same or substantially the same function (direct that native code be executed in place of bytecode) and (2) works in substantially the same way (store a pointer to native code at a location indexed by the bytecode instruction counter) (3) to achieve the same or substantially the same result (faster execution) as this element of the claim.

The '205 Patent	Infringed By
[1-c] executing said new virtual machine instruction instead of said first virtual machine instruction.	JIT-enabled Android devices execute new virtual machine instructions representing or referencing native machine instructions instead of the original bytecode. *See, e.g.* source code files in dalvik\vm\mterp\out\InterpAsm-armv4t.S. *E.g.*, dalvik\vm\mterp\out\InterpAsm-armv4t.S, also available at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/mterp/out/InterpAsm-armv4t.S#l9788: 163 /* 164 * Put the instruction's opcode field into the specified register. 165 */ 166 #define GET_INST_OPCODE(_reg) and _reg, rINST, #255 .... 173 /* 174 * Begin executing the opcode in _reg. Because this only jumps within the 175 * interpreter, we don't have to worry about pre-ARMv5 THUMB interwork. 176 */ 177 #define GOTO_OPCODE(_reg) add pc, rIBASE, _reg, lsl #6 178 #define GOTO_OPCODE_IFEQ(_reg) addeq pc, rIBASE, _reg, lsl #6 179 #define GOTO_OPCODE_IFNE(_reg) addne pc, rIBASE, _reg, lsl #6 .... 9788 common_updateProfile: 9789 eor r3,rPC,rPC,lsr #12 @ cheap, but fast hash function 9790 lsl r3,r3,#(32 - JIT_PROF_SIZE_LOG_2) @ shift out excess bits 9791 ldrb r1,[r0,r3,lsr #(32 - JIT_PROF_SIZE_LOG_2)] @ get counter 9792 GET_INST_OPCODE(ip) 9793 subs r1,r1,#1 @ decrement counter 9794 strb r1,[r0,r3,lsr #(32 - JIT_PROF_SIZE_LOG_2)] @ and store it 9795 GOTO_OPCODE_IFNE(ip) @ if not threshold, fallthrough otherwise */ At 9791, Android gets the counter for the current PC; at 9793 Android subtracts 1 from it; and

The ’205 Patent	Infringed By
	at 9794 Android stores back the decremented value. Then at 9795, Android uses the actual ARM program counter to branch to the “handler” for the opcode in ip. If the counter in r1 *is* zero, Android falls through to: 9797 /* 9798 * Here, we switch to the debug interpreter to request 9799 * trace selection. First, though, check to see if there 9800 * is already a native translation in place (and, if so, 9801 * jump to it now). 9802 */ .... 9807 mov r0,rPC 9808 bl dvmJitGetCodeAddr @ r0<- dvmJitGetCodeAddr(rPC) .... 9812 cmp r0,#0 .... 9814 bxne r0 @ jump to the translation Android uses dmvJitGetCodeAddr to see if there is a native translation and if so, jump to it. The pJitProfTable is allocated and initialized at: dalvik\vm\compiler\Compiler.c, available at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/compiler/Compiler.c#l371: 371 pJitProfTable = (unsigned char *)malloc(JIT_PROF_SIZE); .... 377 memset(pJitProfTable, gDvmJit.threshold, JIT_PROF_SIZE); gDvmJit.threshold is initialized differently based on the platform, *e.g.*: vm/compiler/codegen/arm/armv5te-vfp/ArchVariant.c

The ’205 Patent	Infringed By
	54 gDvmJit.threshold = 200; vm/compiler/codegen/arm/armv5te/ArchVariant.c 54 gDvmJit.threshold = 200; vm/compiler/codegen/arm/armv7-a-neon/ArchVariant.c 49 gDvmJit.threshold = 40; vm/compiler/codegen/arm/armv7-a/ArchVariant.c 49 gDvmJit.threshold = 40; dalvik\vm\interp\Interp.c: /* * Main interpreter loop entry point. Select "standard" or "debug" * interpreter and switch between them as required. * * This begins executing code at the start of "method". On exit, "pResult" * holds the return value of the method (or, if "method" returns NULL, it * holds an undefined value). * * The interpreted stack frame, which holds the method arguments, has * already been set up. */ void dvmInterpret(Thread* self, const Method* method, JValue* pResult) { InterpState interpState; bool change; #if defined(WITH_JIT) /* Target-specific save/restore */ extern void dvmJitCalleeSave(double *saveArea); extern void dvmJitCalleeRestore(double *saveArea);

The ’205 Patent	Infringed By
	`` /* Interpreter entry points from compiled code */ extern void dvmJitToInterpNormal(); extern void dvmJitToInterpNoChain(); extern void dvmJitToInterpPunt(); extern void dvmJitToInterpSingleStep(); extern void dvmJitToInterpTraceSelectNoChain(); extern void dvmJitToInterpTraceSelect(); extern void dvmJitToPatchPredictedChain(); … … #if defined(WITH_JIT) dvmJitCalleeSave(interpState.calleeSave); /* Initialize the state to kJitNot */ interpState.jitState = kJitNot; /* Setup the Jit-to-interpreter entry points */ interpState.jitToInterpEntries = jitToInterpEntries; /* * Initialize the threshold filter [don't bother to zero out the * actual table. We're looking for matches, and an occasional * false positive is acceptible. */ interpState.lastThreshFilter = 0; #endif /* * Initialize working state. * * No need to initialize "retval". */ interpState.method = method; ```

The ’205 Patent	Infringed By
	interpState.fp = (u4*) self->curFrame; interpState.pc = method->insns; interpState.entryPoint = kInterpEntryInstr; …. if (dvmDebuggerOrProfilerActive()) interpState.nextMode = INTERP_DBG; else interpState.nextMode = INTERP_STD; assert(!dvmIsNativeMethod(method)); … typedef bool (*Interpreter)(Thread*, InterpState*); Interpreter stdInterp; if (gDvm.executionMode == kExecutionModeInterpFast) stdInterp = dvmMterpStd; #if defined(WITH_JIT) else if (gDvm.executionMode == kExecutionModeJit) /* If profiling overhead can be kept low enough, we can use a profiling * mterp fast for both Jit and "fast" modes. If overhead is too high, * create a specialized profiling interpreter. */ stdInterp = dvmMterpStd; #endif else stdInterp = dvmInterpretStd; change = true; while (change) { switch (interpState.nextMode) {

The ’205 Patent	Infringed By
	<pre> case INTERP_STD: LOGVV("threadid=%d: interp STD\n", self->threadId); change = (*stdInterp)(self, &interpState); break; … } } *pResult = interpState.retval; #if defined(WITH_JIT) dvmJitCalleeRestore(interpState.calleeSave); #endif }</pre> dalvik\vm\mterp\Mterp.c: <pre> /* * "Standard" mterp entry point. This sets up a "glue" structure and then * calls into the assembly interpreter implementation. * * (There is presently no "debug" entry point.) */ bool dvmMterpStd(Thread* self, InterpState* glue) { int changeInterp; /* configure mterp items */ glue->self = self; glue->methodClassDex = glue->method->clazz->pDvmDex; glue->interpStackEnd = self->interpStackEnd; glue->pSelfSuspendCount = &self->suspendCount;</pre>

The ’205 Patent	Infringed By
	<pre>#if defined(WITH_JIT) glue->pJitProfTable = gDvmJit.pProfTable; glue->ppJitProfTable = &gDvmJit.pProfTable; glue->jitThreshold = gDvmJit.threshold; #endif … changeInterp = dvmMterpStdRun(glue); #if defined(WITH_JIT) if (glue->jitState != kJitSingleStep) { glue->self->inJitCodeCache = NULL; } #endif … }</pre> dalvik\vm\mterp\cstubs\entry.c: <pre>/* * C mterp entry point. This just calls the various C fallbacks, making * this a slow but portable interpeter. * * This is only used for the "allstubs" variant. */ bool dvmMterpStdRun(MterpGlue* glue) { jmp_buf jmpBuf; int changeInterp; glue->bailPtr = &jmpBuf;</pre>

The ’205 Patent	Infringed By
	<pre> /* * We want to return "changeInterp" as a boolean, but we can't return * zero through longjmp, so we return (boolean+1). */ changeInterp = setjmp(jmpBuf) -1; if (changeInterp >= 0) { Thread* threadSelf = dvmThreadSelf(); LOGVV("mterp threadid=%d returning %d\n", threadSelf->threadId, changeInterp); return changeInterp; } … /* run until somebody longjmp()s out */ while (true) { typedef void (*Handler)(MterpGlue* glue); u2 inst = /*glue->*/pc[0]; Handler handler = (Handler) gDvmMterpHandlers[inst & 0xff]; … (*handler)(glue); } }</pre> dalvik\vm\oo\Object.h: <pre>INLINE bool dvmIsNativeMethod(const Method* method) { return (method->accessFlags & ACC_NATIVE) != 0; }</pre> dalvik\vm\mterp\armv5te\footer.S:

The ’205 Patent	Infringed By
	/* * Here, we switch to the debug interpreter to request * trace selection. First, though, check to see if there * is already a native translation in place (and, if so, * jump to it now). */ GET_JIT_THRESHOLD(r1) ldr r10, [rGLUE, #offGlue_self] @ callee saved r10 <- glue->self strb r1,[r0,r3,lsr #(32 - JIT_PROF_SIZE_LOG_2)] @ reset counter EXPORT_PC() mov r0,rPC bl dvmJitGetCodeAddr @ r0<- dvmJitGetCodeAddr(rPC) str r0, [r10, #offThread_inJitCodeCache] @ set the inJitCodeCache flag mov r1, rPC @ arg1 of translation may need this mov lr, #0 @ in case target is HANDLER_INTERPRET cmp r0,#0 … GET_INST_OPCODE(ip) GOTO_OPCODE(ip) /* no return */ #endif *See also, e.g.,* references to “r0<- dvmJitGetCodeAddr(rPC)” in: • dalvik\vm/mterp/out/InterpAsm-armv4t.S; • dalvik\vm/mterp/out/InterpAsm-armv5te-vfp.S; • dalvik\vm/mterp/out/InterpAsm-armv5te.S; • dalvik\vm/mterp/out/InterpAsm-armv7-a-neon.S; • dalvik\vm/mterp/out/InterpAsm-armv7-a.S. *See* Claim 1-b, *supra*. To the extent Android does not literally infringe this claim element, Android contains

The ’205 Patent	Infringed By
	equivalent elements corresponding to each and every requirement of this claim limitation. When the Android JIT compiles a trace, Android adds the corresponding bytecode instruction counter of the bytecode (the first virtual machine instruction) and a pointer to the compiled trace to the jitEntry table. When interpreting the instruction located at the bytecode instruction counter, Android does a lookup of the bytecode instruction counter in the jitEntry table. If Android finds an entry, Android will execute a branch to the compiled trace instead of executing the bytecode instruction. The differences, if any, between a “new virtual machine instruction” and an entry in the jitEntry table are insubstantial. An entry in the jitEntry table (1) performs the same or substantially the same function (direct that native code be executed in place of bytecode) and (2) works in substantially the same way (store a pointer to native code at a location indexed by the bytecode instruction counter) (3) to achieve the same or substantially the same result (faster execution) as this element of the claim.

The ’205 Patent	Infringed By
2. The method of claim 1, further comprising overwriting a selected virtual machine instruction with a new virtual machine instruction, the new virtual machine instruction specifying execution of the at least one native machine instruction.	*See* Claim 1, *supra*. *See* Claim 1-b and 1-c, *supra*. To the extent Android does not literally infringe this claim element, Android contains equivalent elements corresponding to each and every requirement of this claim limitation. When the Android JIT compiles a trace, Android adds the corresponding bytecode instruction counter of the bytecode (the first virtual machine instruction) and a pointer to the compiled trace to the jitEntry table. When interpreting the instruction located at the bytecode instruction counter, Android does a lookup of the bytecode instruction counter in the jitEntry table. If Android finds an entry, Android will execute a branch to the compiled trace instead of executing the bytecode instruction. The differences, if any, between a “new virtual machine instruction” and an entry in the jitEntry table are insubstantial. An entry in the jitEntry table (1) performs the same or substantially the same function (direct that native code be executed in place of bytecode) and (2) works in substantially the same way (store a pointer to native code at a location indexed by the bytecode instruction counter) (3) to achieve the same or substantially the same result (faster execution) as this element of the claim.

The ’205 Patent	Infringed By
3. The method of claim 2, wherein the [new virtual machine] instruction includes a pointer to the at least one native machine instruction.	*See* Claim 2, *supra*. *See* Claim 1-b, *supra*. Each JitEntry contains a pointer to the native translation.

The ’205 Patent	Infringed By
4. The method of claim 2, further comprising storing the selected virtual machine instruction before it is overwritten.	*See* Claim 2, *supra*. *See, e.g.*, Google I/O 2010 Video, entitled “A JIT Compiler for Android's Dalvik VM,” presented by *A JIT Compiler for Android’s Dalvik VM*, presented by Ben Cheng and Bill Buzbee (Google’s Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo: • at 48:49: Since we manage the code cache by not only storing the generated code but also the original trace information, we can easily replay the compilation request with verbose mode turned on so that you can see the Dalvik code and the corresponding native code. And that's exactly the same mechanism used by the profiler to report the content of the hot translations.

The ’205 Patent	Infringed By
8. In a computer system, a method for increasing the execution speed of virtual machine instructions, the method comprising:	The Accused Instrumentalities include devices that run Android and the Android SDK. Devices running Android and the Android SDK are computer systems. *See* Claim 1, *supra*.
inputting virtual machine instructions for a function;	*See* Claim 1-a, *supra*.
compiling a portion of the function	*See* Claim 1-b, *supra*.

The ’205 Patent	Infringed By
into at least one native machine instruction so that the function includes both virtual and native machine instruction;	
representing said at least one native machine instruction with a new virtual machine instruction that is executed after the compiling of [the function].	*See* Claim 1-b and 1-c, *supra*.

EXHIBIT B-2
Preliminary Infringement Contentions for the ’205 Patent

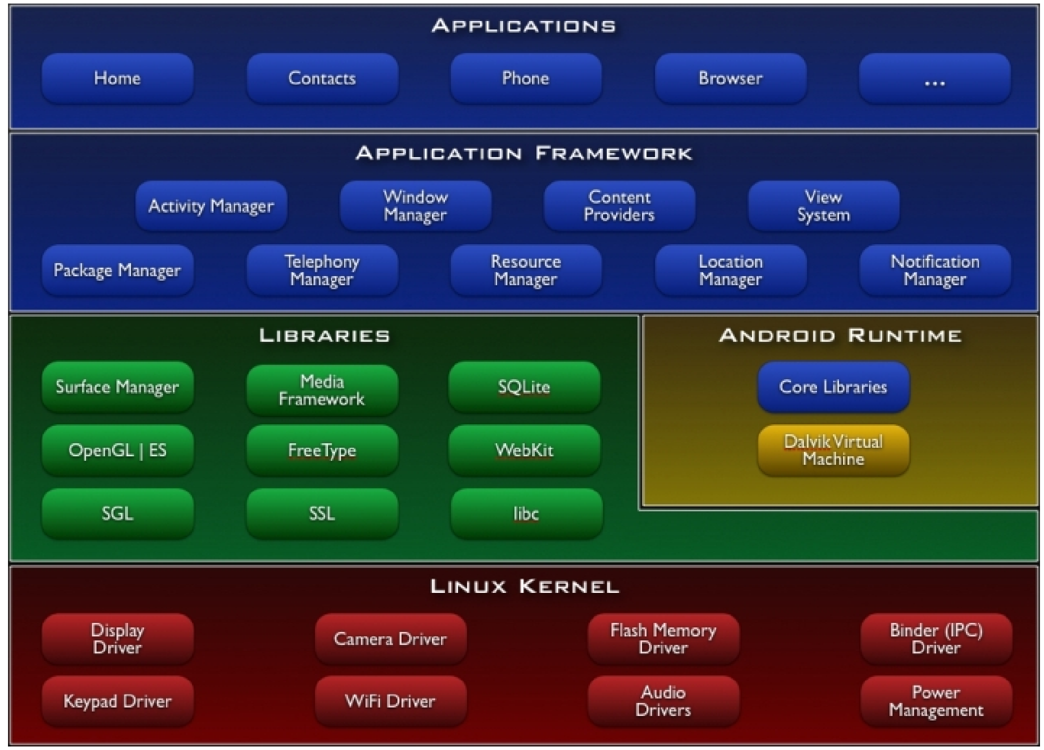
NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’205 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The ’205 Patent	Infringed By
1. In a computer system, a method for increasing the execution speed of virtual machine instructions at runtime, the method comprising:	Android uses the Dalvik virtual machine to execute virtual machine bytecode instructions at runtime. The Dalvik virtual machine performs and runs code resulting from certain optimizations to increase the execution speed of virtual machine instructions at runtime. *See, e.g.*, Android Glossary Definition for “Dalvik,” available at http://developer.android.com/guide/appendix/glossary.html: Dalvik The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management). The Dalvik core class library is

The ’205 Patent	Infringed By
	intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device. Android Basics, entitled “What is Android?,” available at http://developer.android.com/guide/basics/what-is-android.html. **What is Android?** Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. **Features** • Application framework enabling reuse and replacement of components • Dalvik virtual machine optimized for mobile devices • Integrated browser based on the open source WebKit engine • Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional) • SQLite for structured data storage • Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF) • GSM Telephony (hardware dependent) • Bluetooth, EDGE, 3G, and WiFi (hardware dependent) • Camera, GPS, compass, and accelerometer (hardware dependent) • Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE **Android Architecture** The following diagram shows the major components of the Android operating system. Each section is described in more detail below.

The ’205 Patent	Infringed By
	 **Applications** Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. … **Android Runtime**

The ’205 Patent	Infringed By
	Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Android uses the dexopt tool, which increases the execution speed of virtual machine instructions at runtime: *See, e.g.,* dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: **Dalvik Optimization and Verification With *dexopt*** The Dalvik virtual machine was designed specifically for the Android mobile platform. The target systems have little RAM, store data on slow internal flash memory, and generally have the performance characteristics of decade-old desktop systems. They also run Linux, which provides virtual memory, processes and threads, and UID-based security mechanisms. The features and limitations caused us to focus on certain goals: • Class data, notably bytecode, must be shared between multiple processes to minimize total system memory usage. • The overhead in launching a new app must be minimized to keep the device responsive. • Storing class data in individual files results in a lot of redundancy, especially with respect to strings. To conserve disk space we need to factor this out.

The ’205 Patent	Infringed By
	• Parsing class data fields adds unnecessary overhead during class loading. Accessing data values (e.g. integers and strings) directly as C types is better. • Bytecode verification is necessary, but slow, so we want to verify as much as possible outside app execution. • Bytecode optimization (quickened instructions, method pruning) is important for speed and battery life. • For security reasons, processes may not edit shared code. The typical VM implementation uncompresses individual classes from a compressed archive and stores them on the heap. This implies a separate copy of each class in every process, and slows application startup because the code must be uncompressed (or at least read off disk in many small pieces). On the other hand, having the bytecode on the local heap makes it easy to rewrite instructions on first use, facilitating a number of different optimizations. The goals led us to make some fundamental decisions: • Multiple classes are aggregated into a single "DEX" file. • DEX files are mapped read-only and shared between processes. • Byte ordering and word alignment are adjusted to suit the local system. • Bytecode verification is mandatory for all classes, but we want to "pre-verify" whatever we can. • Optimizations that require rewriting bytecode must be done ahead of time. • The consequences of these decisions are explained in the following sections. …. **dexopt** We want to verify and optimize all of the classes in the DEX file. The easiest and safest way to do this is to load all of the classes into the VM and run through them. Anything that fails to load is simply not verified or optimized. Unfortunately, this can cause allocation of some resources that are difficult to release (e.g. loading of native shared libraries), so we don't want to do it in the same virtual machine that we're running applications in.

The ’205 Patent	Infringed By
	The solution is to invoke a program called dexopt, which is really just a back door into the VM. It performs an abbreviated VM initialization, loads zero or more DEX files from the bootstrap class path, and then sets about verifying and optimizing whatever it can from the target DEX. On completion, the process exits, freeing all resources. It is possible for multiple VMs to want the same DEX file at the same time. File locking is used to ensure that dexopt is only run once. …. **Optimization** Virtual machine interpreters typically perform certain optimizations the first time a piece of code is used. Constant pool references are replaced with pointers to internal data structures, operations that always succeed or always work a certain way are replaced with simpler forms. Some of these require information only available at runtime, others can be inferred statically when certain assumptions are made. The Dalvik optimizer does the following: • For virtual method calls, replace the method index with a vtable index. • For instance field get/put, replace the field index with a byte offset. Also, merge the boolean / byte / char / short variants into a single 32-bit form (less code in the interpreter means more room in the CPU I-cache). • Replace a handful of high-volume calls, like String.length(), with "inline" replacements. This skips the usual method call overhead, directly switching from the interpreter to a native implementation. • Prune empty methods. The simplest example is Object.<init>, which does nothing, but must be called whenever any object is allocated. The instruction is replaced with a new version that acts as a no-op unless a debugger is attached. • Append pre-computed data. For example, the VM wants to have a hash table for lookups on class name. Instead of computing this when the DEX file is loaded, we can compute it now, saving heap space and computation time in every VM where the DEX is loaded.

The ’205 Patent	Infringed By
	All of the instruction modifications involve replacing the opcode with one not defined by the Dalvik specification. This allows us to freely mix optimized and unoptimized instructions. The set of optimized instructions, and their exact representation, is tied closely to the VM version. Most of the optimizations are obvious "wins". The use of raw indices and offsets not only allows us to execute more quickly, we can also skip the initial symbolic resolution. Pre-computation eats up disk space, and so must be done in moderation. There are a couple of potential sources of trouble with these optimizations. First, vtable indices and byte offsets are subject to change if the VM is updated. Second, if a superclass is in a different DEX, and that other DEX is updated, we need to ensure that our optimized indices and offsets are updated as well. A similar but more subtle problem emerges when user-defined class loaders are employed: the class we actually call may not be the one we expected to call. These problems are addressed with dependency lists and some limitations on what can be optimized. *See also, e.g.*, dalvik\docs\ embedded-vm-control.html#verifier (“The system tries to pre-verify all classes in a DEX file to reduce class load overhead, and performs a series of optimizations to improve runtime performance. Both of these are done by the dexopt command, either in the build system or by the installer. On a development device, dexopt may be run the first time a DEX file is used and whenever it or one of its dependencies is updated ("just-in-time" optimization and verification).”).
receiving a first virtual machine instruction;	Android’s dexopt tool receives a first virtual machine instruction. *See, e.g.*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l1486 `/*` `* Run through all classes that were successfully loaded from this DEX` `* file and optimize their code sections.` `*/` `static void optimizeLoadedClasses(DexFile* pDexFile)` `{`

The ’205 Patent	Infringed By

```
u4 count = pDexFile->pHeader->classDefsSize;
u4 idx;
InlineSub* inlineSubs = NULL;

LOGV("DexOpt: +++ optimizing up to %d classes\n", count);
assert(gDvm.dexOptMode != OPTIMIZE_MODE_NONE);

inlineSubs = createInlineSubsTable();

for (idx = 0; idx < count; idx++) {
const DexClassDef* pClassDef;
const char* classDescriptor;
ClassObject* clazz;

pClassDef = dexGetClassDef(pDexFile, idx);
classDescriptor = dexStringByTypeIdx(pDexFile, pClassDef->classIdx);

/* all classes are loaded into the bootstrap class loader */
clazz = dvmLookupClass(classDescriptor, NULL, false);
if (clazz != NULL) {
if ((pClassDef->accessFlags & CLASS_ISPREVERIFIED) == 0 &&
gDvm.dexOptMode == OPTIMIZE_MODE_VERIFIED)
{
LOGV("DexOpt: not optimizing '%s': not verified\n",
classDescriptor);
} else if (clazz->pDvmDex->pDexFile != pDexFile) {
/* shouldn't be here -- verifier should have caught */
LOGD("DexOpt: not optimizing '%s': multiple definitions\n",
classDescriptor);
} else {
optimizeClass(clazz, inlineSubs);

/* set the flag whether or not we actually did anything */
((DexClassDef*)pClassDef)->accessFlags |=
CLASS_ISOPTIMIZED;
}
} else {
LOGV("DexOpt: not optimizing unavailable class '%s'\n",
classDescriptor);
}
}

free(inlineSubs);
}
```

The ’205 Patent	Infringed By

At 1501..1522..1532, Android calls `optimizeClass()` for each class in DEX file, passing in a table of inline substitutions, inlineSubs.

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l1537

```
1537 /*
1538 * Optimize the specified class.
1539 */
1540 static void optimizeClass(ClassObject* clazz, const InlineSub*
inlineSubs)
1541 {
1542 int i;
1543
1544 for (i = 0; i < clazz->directMethodCount; i++) {
1545 if (!optimizeMethod(&clazz->directMethods[i], inlineSubs))
1546 goto fail;
1547 }
1548 for (i = 0; i < clazz->virtualMethodCount; i++) {
1549 if (!optimizeMethod(&clazz->virtualMethods[i], inlineSubs))
1550 goto fail;
1551 }
1552
1553 return;
1554
1555 fail:
1556 LOGV("DexOpt: ceasing optimization attempts on %s\n", clazz-
>descriptor);
1557 }
```

At 1544-1545..1547 and 1548-1549..1551 Android calls `optimizeMethod()` on each direct or virtual method in the incoming DEX class.

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l1559

```
1559 /*
1560 * Optimize instructions in a method.
1561 *
1562 * Returns "true" if all went well, "false" if we bailed out early when
1563 * something failed.
1564 */
1565 static bool optimizeMethod(Method* method, const InlineSub* inlineSubs)
1566 {
```

<table>
<tr><th>The ’205 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

```
1567 u4 insnsSize;
1568 u2* insns;
1569 u2 inst;
1570
1571 if (dvmIsNativeMethod(method) || dvmIsAbstractMethod(method))
1572 return true;
1573
1574 insns = (u2*) method->insns;
1575 assert(insns != NULL);
1576 insnsSize = dvmGetMethodInsnsSize(method);
1577
1578 while (insnsSize > 0) {
1579 int width;
1580
1581 inst = *insns & 0xff;
1582
1583 switch (inst) {
...
1645 case OP_INVOKE_DIRECT_RANGE:
1646 rewriteExecuteInlineRange(method, insns, METHOD_DIRECT, inlineSubs);
1647 break;
...
1652 case OP_INVOKE_STATIC_RANGE:
1653 rewriteExecuteInlineRange(method, insns, METHOD_STATIC, inlineSubs);
1654 break;
...
1656 default:
1657 // ignore this instruction
1658 ;
1659 }
...
1674 insns += width;
1675 insnsSize -= width;
1676 }
1677
1678 assert(insnsSize == 0);
1679 return true;
1680 }
```

1578..1645-1647..1652-1654..1676 Android calls `rewriteExecuteInlineRange()` for each OP_INVOKE_DIRECT_RANGE or OP_INVOKE_STATIC_RANGE virtual machine instruction.</td></tr>
<tr><td>generating, at runtime, a new</td><td>Android generates at runtime a new virtual machine instruction that represents or references one</td></tr>
</table>

The ’205 Patent	Infringed By
virtual machine instruction that represents or references one or more native instructions that can be executed instead of said first virtual machine instruction; and	or more native instructions that can be executed instead of said first virtual machine instruction. As described above, Android includes a utility called dexopt: *See, e.g.*, dalvik\docs\dexopt.html; *see also*, http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=docs/dexopt.html: **dexopt** We want to verify and optimize all of the classes in the DEX file. The easiest and safest way to do this is to load all of the classes into the VM and run through them. Anything that fails to load is simply not verified or optimized. Unfortunately, this can cause allocation of some resources that are difficult to release (e.g. loading of native shared libraries), so we don't want to do it in the same virtual machine that we're running applications in. The solution is to invoke a program called dexopt, which is really just a back door into the VM. It performs an abbreviated VM initialization, loads zero or more DEX files from the bootstrap class path, and then sets about verifying and optimizing whatever it can from the target DEX. On completion, the process exits, freeing all resources. It is possible for multiple VMs to want the same DEX file at the same time. File locking is used to ensure that dexopt is only run once. …. *See also, e.g.*, dalvik\docs\ embedded-vm-control.html#verifier (“The system tries to pre-verify all classes in a DEX file to reduce class load overhead, and performs a series of optimizations to improve runtime performance. Both of these are done by the dexopt command, either in the build system or by the installer. On a development device, dexopt may be run the first time a DEX file is used and whenever it or one of its dependencies is updated ("just-in-time" optimization and verification).”). *See, e.g.*,

The ’205 Patent	Infringed By
	http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l2345

```
2345 /*
2346 * See if the method being called can be rewritten as an inline
operation.
2347 * Works for invoke-virtual/range, invoke-direct/range, and invoke-
static/range.
2348 *
2349 * Returns "true" if we replace it.
2350 */
2351 static bool rewriteExecuteInlineRange(Method* method, u2* insns,
2352 MethodType methodType, const InlineSub* inlineSubs)
2353 {
2354 ClassObject* clazz = method->clazz;
2355 Method* calledMethod;
2356 u2 methodIdx = insns[1];
2357
2358 calledMethod = dvmOptResolveMethod(clazz, methodIdx, methodType, NULL);
2359 if (calledMethod == NULL) {
2360 LOGV("+++ DexOpt inline/range: can't find %d\n", methodIdx);
2361 return false;
2362 }
2363
2364 while (inlineSubs->method != NULL) {
2365 if (inlineSubs->method == calledMethod) {
2366 assert((insns[0] & 0xff) == OP_INVOKE_DIRECT_RANGE ||
2367 (insns[0] & 0xff) == OP_INVOKE_STATIC_RANGE ||
2368 (insns[0] & 0xff) == OP_INVOKE_VIRTUAL_RANGE);
2369 insns[0] = (insns[0] & 0xff00) | (u2) OP_EXECUTE_INLINE_RANGE;
2370 insns[1] = (u2) inlineSubs->inlineIdx;
2371
2372 //LOGI("DexOpt: execute-inline/range %s.%s --> %s.%s\n",
2373 // method->clazz->descriptor, method->name,
2374 // calledMethod->clazz->descriptor, calledMethod->name);
2375 return true;
2376 }
2377
2378 inlineSubs++;
2379 }
2380
2381 return false;
2382 }
```

at 2369-70 Android generates a new instruction, with OP_EXECUTE_INLINE_RANGE as the

The ’205 Patent	Infringed By
	new opcode.

at 2370 Android stores inlineSubs->inlineIdx (the index of the native code in the inlineSubs table) as the instruction data to reference the native code.

Android passes in `const InlineSub* inlineSubs`, which is constructed at line 1499 by calling `createInlineSubsTable()`, which is:

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l1411

```
1411 /*
1412 * Create a table of inline substitutions.
1413 *
1414 * TODO: this is currently just a linear array. We will want to put this
1415 * into a hash table as the list size increases.
1416 */
1417 static InlineSub* createInlineSubsTable(void)
1418 {
1419 const InlineOperation* ops = dvmGetInlineOpsTable();
1420 const int count = dvmGetInlineOpsTableLength();
1421 InlineSub* table;
1422 Method* method;
1423 ClassObject* clazz;
1424 int i, tableIndex;
1425
1426 /*
1427 * Allocate for optimism: one slot per entry, plus an end-of-list marker.
1428 */
1429 table = malloc(sizeof(InlineSub) * (count+1));
1430
1431 tableIndex = 0;
1432 for (i = 0; i < count; i++) {
1433 clazz = dvmFindClassNoInit(ops[i].classDescriptor, NULL);
1434 if (clazz == NULL) {
1435 LOGV("DexOpt: can't inline for class '%s': not found\n",
1436 ops[i].classDescriptor);
1437 dvmClearOptException(dvmThreadSelf());
1438 } else {
1439 /*
1440 * Method could be virtual or direct. Try both. Don't use
1441 * the "hier" versions.
1442 */
```

The ’205 Patent	Infringed By

```
method = dvmFindDirectMethodByDescriptor(clazz, ops[i].methodName,
ops[i].methodSignature);
if (method == NULL)
method = dvmFindVirtualMethodByDescriptor(clazz, ops[i].methodName,
ops[i].methodSignature);
if (method == NULL) {
LOGW("DexOpt: can't inline %s.%s %s: method not found\n",
ops[i].classDescriptor, ops[i].methodName,
ops[i].methodSignature);
} else {
if (!dvmIsFinalClass(clazz) && !dvmIsFinalMethod(method)) {
LOGW("DexOpt: WARNING: inline op on non-final class/method "
"%s.%s\n",
clazz->descriptor, method->name);
/* fail? */
}
if (dvmIsSynchronizedMethod(method) ||
dvmIsDeclaredSynchronizedMethod(method))
{
LOGW("DexOpt: WARNING: inline op on synchronized method "
"%s.%s\n",
clazz->descriptor, method->name);
/* fail? */
}

table[tableIndex].method = method;
table[tableIndex].inlineIdx = i;
tableIndex++;

LOGV("DexOpt: will inline %d: %s.%s %s\n", i,
ops[i].classDescriptor, ops[i].methodName,
ops[i].methodSignature);
}
}
}

/* mark end of table */
table[tableIndex].method = NULL;
LOGV("DexOpt: inline table has %d entries\n", tableIndex);

return table;
}
```

a map from Method*’s to indexes into the table returned by `dvmGetInlineOpsTable()`,

The ’205 Patent	Infringed By
	which is:

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/InlineNative.c#l706

```
/*
* Get a pointer to the inlineops table.
*/
const InlineOperation* dvmGetInlineOpsTable(void)
{
return gDvmInlineOpsTable;
}
```

where `gDvmInlineOpsTable` is:

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/InlineNative.c#l628

```
/*
* Table of methods.
*
* The DEX optimizer uses the class/method/signature string fields to decide
* which calls it can trample. The interpreter just uses the function
* pointer field.
*
* IMPORTANT: you must update DALVIK_VM_BUILD in DalvikVersion.h if you make
* changes to this table.
*
* NOTE: If present, the JIT will also need to know about changes
* to this table. Update the NativeInlineOps enum in InlineNative.h and
* the dispatch code in compiler/codegen/<target>/Codegen.c.
*/
const InlineOperation gDvmInlineOpsTable[] = {
{ org_apache_harmony_dalvik_NativeTestTarget_emptyInlineMethod,
"Lorg/apache/harmony/dalvik/NativeTestTarget;",
"emptyInlineMethod", "()V" },

{ javaLangString_charAt,
"Ljava/lang/String;", "charAt", "(I)C" },
{ javaLangString_compareTo,
"Ljava/lang/String;", "compareTo", "(Ljava/lang/String;)I" },
{ javaLangString_equals,
"Ljava/lang/String;", "equals", "(Ljava/lang/Object;)Z" },
{ javaLangString_indexOf_I,
```

The ’205 Patent	Infringed By

```
654 "Ljava/lang/String;", "indexOf", "(I)I" },
655 { javaLangString_indexOf_II,
656 "Ljava/lang/String;", "indexOf", "(II)I" },
657 { javaLangString_length,
658 "Ljava/lang/String;", "length", "()I" },
659
660 { javaLangMath_abs_int,
661 "Ljava/lang/Math;", "abs", "(I)I" },
662 { javaLangMath_abs_long,
663 "Ljava/lang/Math;", "abs", "(J)J" },
664 { javaLangMath_abs_float,
665 "Ljava/lang/Math;", "abs", "(F)F" },
666 { javaLangMath_abs_double,
667 "Ljava/lang/Math;", "abs", "(D)D" },
668 { javaLangMath_min_int,
669 "Ljava/lang/Math;", "min", "(II)I" },
670 { javaLangMath_max_int,
671 "Ljava/lang/Math;", "max", "(II)I" },
672 { javaLangMath_sqrt,
673 "Ljava/lang/Math;", "sqrt", "(D)D" },
674 { javaLangMath_cos,
675 "Ljava/lang/Math;", "cos", "(D)D" },
676 { javaLangMath_sin,
677 "Ljava/lang/Math;", "sin", "(D)D" },
678 };
```

where the elements are instances of:

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/InlineNative.h#l26

```
26 /*
27 * Basic 4-argument inline operation handler.
28 */
29 typedef bool (*InlineOp4Func)(u4 arg0, u4 arg1, u4 arg2, u4 arg3,
30 JValue* pResult);
...
44 typedef struct InlineOperation {
45 InlineOp4Func func; /* MUST be first entry */
46 const char* classDescriptor;
47 const char* methodName;
48 const char* methodSignature;
49 } InlineOperation;
```

The pointers to the functions are references to native instructions, for example,

<table>
<tr><th>The ’205 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/InlineNative.c#l374

```
/*
* public int length()
*/
static bool javaLangString_length(u4 arg0, u4 arg1, u4 arg2, u4 arg3,
JValue* pResult)
{
//LOGI("String.length this=0x%08x pResult=%p\n", arg0, pResult);

/* null reference check on "this" */
if (!dvmValidateObject((Object*) arg0))
return false;

pResult->i = dvmGetFieldInt((Object*) arg0, STRING_FIELDOFF_COUNT);
return true;
}
```

which is compiled into native instructions. The alternative would be to interpret many virtual machine instructions to do the same thing.

</td></tr>
<tr><td>executing said new virtual machine instruction instead of said first virtual machine instruction.</td><td>

Android executes the new virtual machine instruction instead of said first virtual machine instruction.

See, e.g., http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/mterp/out/InterpAsm-armv4t.S#l7686

```
/* ------------------------------ */
.balign 64
.L_OP_EXECUTE_INLINE_RANGE: /* 0xef */
/* File: armv5te/OP_EXECUTE_INLINE_RANGE.S */
/*
* Execute a "native inline" instruction, using "/range" semantics.
* Same idea as execute-inline, but we get the args differently.
*
* We need to call an InlineOp4Func:
* bool (func)(u4 arg0, u4 arg1, u4 arg2, u4 arg3, JValue* pResult)
*
* The first four args are in r0-r3, pointer to return value storage
* is on the stack. The function's return value is a flag that tells
```

</td></tr>
</table>

The '205 Patent	Infringed By

```
7699 * us if an exception was thrown.
7700 */
7701 /* [opt] execute-inline/range {vCCCC..v(CCCC+AA-1)}, inline@BBBB */
7702 FETCH(r10, 1) @ r10<- BBBB
7703 add r1, rGLUE, #offGlue_retval @ r1<- &glue->retval
7704 EXPORT_PC() @ can throw
7705 sub sp, sp, #8 @ make room for arg, +64 bit align
7706 mov r0, rINST, lsr #8 @ r0<- AA
7707 str r1, [sp] @ push &glue->retval
7708 bl .LOP_EXECUTE_INLINE_RANGE_continue @ make call; will return after
7709 add sp, sp, #8 @ pop stack
7710 cmp r0, #0 @ test boolean result of inline
7711 beq common_exceptionThrown @ returned false, handle exception
7712 FETCH_ADVANCE_INST(3) @ advance rPC, load rINST
7713 GET_INST_OPCODE(ip) @ extract opcode from rINST
7714 GOTO_OPCODE(ip) @ jump to next instruction
```

This is the computed-goto threaded-interpreter code for the OP_EXECUTE_INLINE_RANGE virtual machine instruction (0xef), which has replaced the OP_INVOKE_DIRECT_RANGE or OP_INVOKE_STATIC_RANGE as the virtual machine instruction.

7708 calls `.LOP_EXECUTE_INLINE_RANGE_continue` to perform the actual transfer to the native instructions.

http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/mterp/out/InterpAsm-armv4t.S#l9502

```
9502 /* continuation for OP_EXECUTE_INLINE_RANGE */
9503
9504 /*
9505 * Extract args, call function.
9506 * r0 = #of args (0-4)
9507 * r10 = call index
9508 * lr = return addr, above [DO NOT bl out of here w/o preserving LR]
9509 */
9510 .LOP_EXECUTE_INLINE_RANGE_continue:
9511 rsb r0, r0, #4 @ r0<- 4-r0
9512 FETCH(r9, 2) @ r9<- CCCC
9513 add pc, pc, r0, lsl #3 @ computed goto, 2 instrs each
9514 bl common_abort @ (skipped due to ARM prefetch)
9515 4: add ip, r9, #3 @ base+3
9516 GET_VREG(r3, ip) @ r3<- vBase[3]
```

The ’205 Patent	Infringed By
	`9517 3: add ip, r9, #2 @ base+2` `9518 GET_VREG(r2, ip) @ r2<- vBase[2]` `9519 2: add ip, r9, #1 @ base+1` `9520 GET_VREG(r1, ip) @ r1<- vBase[1]` `9521 1: add ip, r9, #0 @ (nop)` `9522 GET_VREG(r0, ip) @ r0<- vBase[0]` `9523 0:` `9524 ldr r9, .LOP_EXECUTE_INLINE_RANGE_table @ table of InlineOperation` `9525 LDR_PC "[r9, r10, lsl #4]" @ sizeof=16, "func" is first entry` `9526 @ (not reached)` which at 9524-2525 uses the reference to the table of native instructions to fetch a new native program counter, and transfers to those native instructions.

The ’205 Patent	Infringed By
2. The method of claim 1, further comprising overwriting a selected virtual machine instruction with a new virtual machine instruction, the new virtual machine instruction specifying execution of the at least one native machine instruction.	*See* Claim 1, *supra*. The overwriting of the selected virtual machine instruction with the new virtual machine instruction is in rewriteExecuteInlineRange, cited above, but repeated here for clarity: http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/analysis/DexOptimize.c#l2345 `2345 /*` `2346 * See if the method being called can be rewritten as an inline operation.` `2347 * Works for invoke-virtual/range, invoke-direct/range, and invoke-static/range.` `2348 *` `2349 * Returns "true" if we replace it.` `2350 */` `2351 static bool rewriteExecuteInlineRange(Method* method, u2* insns,` `2352 MethodType methodType, const InlineSub* inlineSubs)` `...` `2369 insns[0] = (insns[0] & 0xff00) | (u2) OP_EXECUTE_INLINE_RANGE;` `2370 insns[1] = (u2) inlineSubs->inlineIdx;`

The ’205 Patent	Infringed By
3. The method of claim 2, wherein the [new virtual machine] instruction includes a pointer to the at least one native machine instruction.	*See* Claim 2, *supra*. The OP_EXECUTE_INLINE_RANGE bytecode takes as an argument the index of the method in the table of inline subroutines. The first field of each element in that table is a pointer to the native code for that subroutine. `2370 insns[1] = (u2) inlineSubs->inlineIdx;`

EXHIBIT C
Preliminary Infringement Contentions for the '702 Patent

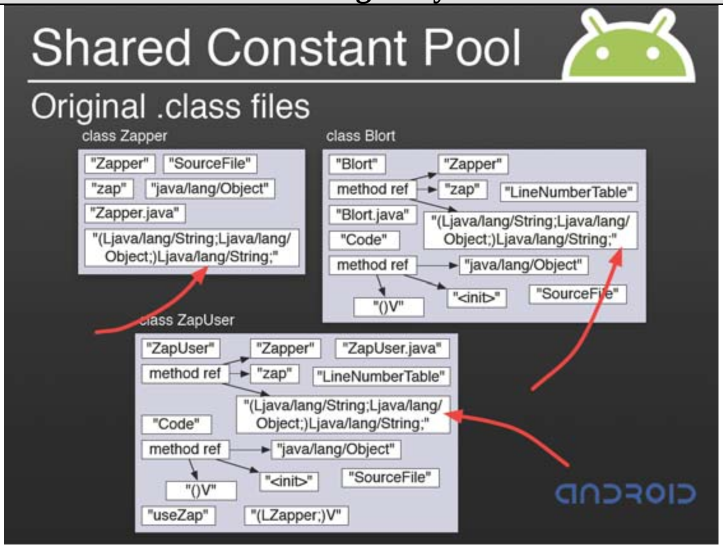
NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google's Android websites. Oracle's infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle's investigation is ongoing, the '702 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 ("Cupcake"), 1.6 ("Donut"), 2.0/2.1 ("Éclair"), and 2.2 ("Froyo").

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, "dalvik\vm\native\InternalNative.c" maps to "[platform/dalvik.git] / vm / native / InternalNative.c" (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

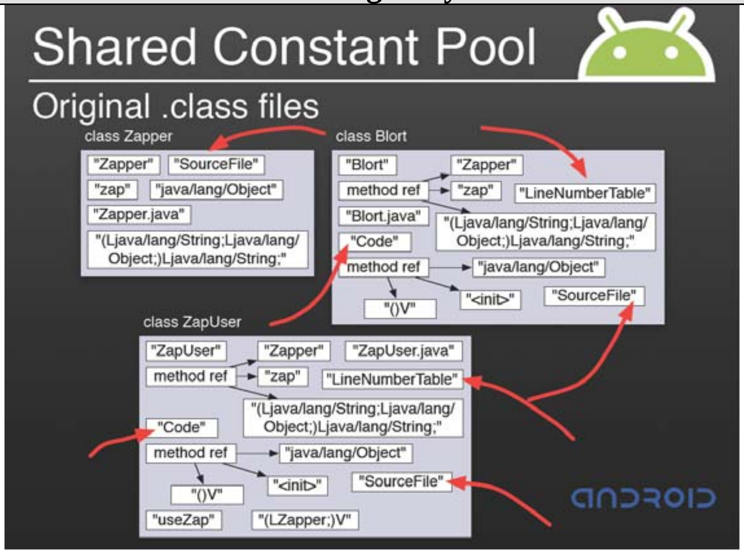
It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The '702 Patent	Infringed By
1. A method of pre-processing class files comprising:	The Android dx tool involves a method of pre-processing .class files into a Dalvik executable format (.dex) file. **"dx** The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories to Dalvik executable format (.dex) files, so that they can run in the Android environment." Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html The method of pre-processing class files into a .dex file that can be interpreted by the Dalvik Virtual Machine (Dalvik VM) is explained in the Dalvik VM video presentation and related presentation from Google I/O 2008, dated 5/29/2008.

The ’702 Patent	Infringed By
	See Google I/O 2008 Video entitled “*Google I/O 2008 - Dalvik Virtual Machine Internals*,” presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM (“Dalvik Video”), at time 5:45–10:45. *See also* Google I/O 2008 Presentation Slides, entitled, “*Dalvik Virtual Machine Internals, Google I/O 2008*,” presented by Dan Bornstein (“Dalvik Presentation”) at slides 11-22, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0. In the Android source code, *see generally*: “Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code • com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util” dalvik\dx\src\com\android\dx\dex\cf\package.html.
determining plurality of duplicated elements in a plurality of class files;	The Android dx tool determines a plurality of duplicated elements in a plurality of class files, as explained in the Dalvik Video at time 7:50-8:45 and Dalvik Presentation, slides 18-19. The Dalvik Presentation shows the determination of a plurality of duplicated elements (e.g., class signatures and string names) in a plurality of class files:

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 18) (Shows identification of common class signatures in the class files)

<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



(Dalvik Presentation, slide 19)
(Shows identification of common string names in the class files)

In the Android source code, *see also generally*:

> “Interfaces and implementation of things related to the constant pool.
> PACKAGES USED:
> * com.android.dx.rop.type
> * com.android.dx.util”

dalvik/dx/src/com/android/dx/rop/cst/package.html.

See also DexFile.java:

```

    /**
     * Gets the {@link IndexedItem} corresponding to the given constant,
     * if it is a constant that has such a correspondence, or return
     * {@code null} if it isn't such a constant. This will throw
```

</td></tr>
</table>

The ’702 Patent	Infringed By

```
     * an exception if the given constant <i>should</i> have been found
     * but wasn't.
     *
     * @param cst {@code non-null;} the constant to look up
     * @return {@code null-
ok;} its corresponding item, if it has a corresponding
     * item, or {@code null} if it's not that sort of constant
     */
    /*package*/ IndexedItem findItemOrNull(Constant cst) {
        IndexedItem item;

        if (cst instanceof CstString) {
            return stringIds.get(cst);
        } else if (cst instanceof CstType) {
            return typeIds.get(cst);
        } else if (cst instanceof CstBaseMethodRef) {
            return methodIds.get(cst);
        } else if (cst instanceof CstFieldRef) {
            return fieldIds.get(cst);
        } else {
            return null;
        }
    }

    /**
     * Returns the contents of this instance as a {@code .dex} file,
     * in a {@link ByteArrayAnnotatedOutput} instance.
     *
     * @param annotate whether or not to keep annotations
     * @param verbose if annotating, whether to be verbose
     * @return {@code non-null;} a {@code .dex} file for this instance
     */
    private ByteArrayAnnotatedOutput toDex0(boolean annotate,
            boolean verbose) {
        /*
         * The following is ordered so that the prepare() calls which
         * add items happen before the calls to the sections that get
         * added to.
         */

        classDefs.prepare();
        classData.prepare();
        wordData.prepare();
        byteData.prepare();
```

The ’702 Patent	Infringed By

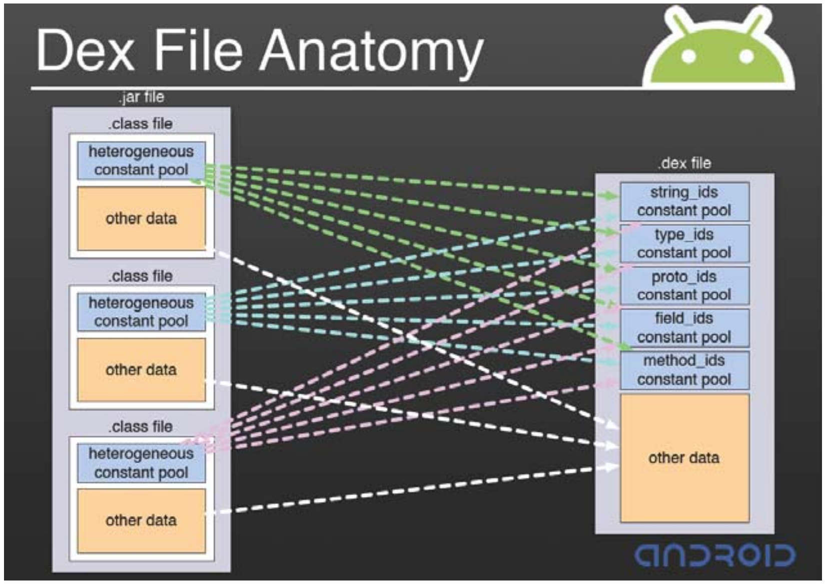
```
        methodIds.prepare();
        fieldIds.prepare();
        protoIds.prepare();
        typeLists.prepare();
        typeIds.prepare();
        stringIds.prepare();
        stringData.prepare();
        header.prepare();

        // Place the sections within the file.

        int count = sections.length;
        int offset = 0;

        for (int i = 0; i < count; i++) {
            Section one = sections[i];
            int placedAt = one.setFileOffset(offset);
            if (placedAt < offset) {
                throw new RuntimeException("bogus placement for section " + i);
            }

            try {
                if (one == map) {
                    /*
                     * Inform the map of all the sections, and add it
                     * to the file. This can only be done after all
                     * the other items have been sorted and placed.
                     */
                    MapItem.addMap(sections, map);
                    map.prepare();
                }

                if (one instanceof MixedItemSection) {
                    /*
                     * Place the items of a MixedItemSection that just
                     * got placed.
                     */
                    ((MixedItemSection) one).placeItems();
                }

                offset = placedAt + one.writeSize();
            } catch (RuntimeException ex) {
                throw ExceptionWithContext.withContext(ex,
```

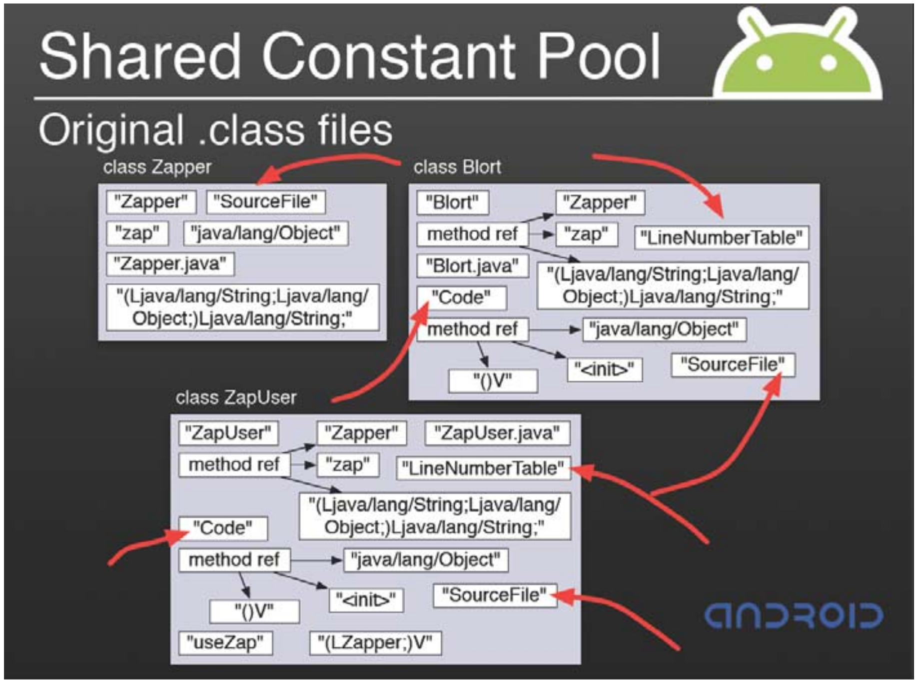
The ’702 Patent	Infringed By

```
                        "...while writing section " + i);
            }
        }

        // Write out all the sections.

        fileSize = offset;
        byte[] barr = new byte[fileSize];
        ByteArrayAnnotatedOutput out = new ByteArrayAnnotatedOutput(barr);

        if (annotate) {
            out.enableAnnotations(dumpWidth, verbose);
        }

        for (int i = 0; i < count; i++) {
            try {
                Section one = sections[i];
                int zeroCount = one.getFileOffset() - out.getCursor();
                if (zeroCount < 0) {
                    throw new ExceptionWithContext("excess write of " +
                            (-zeroCount));
                }
                out.writeZeroes(one.getFileOffset() - out.getCursor());
                one.writeTo(out);
            } catch (RuntimeException ex) {
                ExceptionWithContext ec;
                if (ex instanceof ExceptionWithContext) {
                    ec = (ExceptionWithContext) ex;
                } else {
                    ec = new ExceptionWithContext(ex);
                }
                ec.addContext("...while writing section " + i);
                throw ec;
            }
        }

        if (out.getCursor() != fileSize) {
            throw new RuntimeException("foreshortened write");
        }

        // Perform final bookkeeping.

        calcSignature(barr);
        calcChecksum(barr);
```

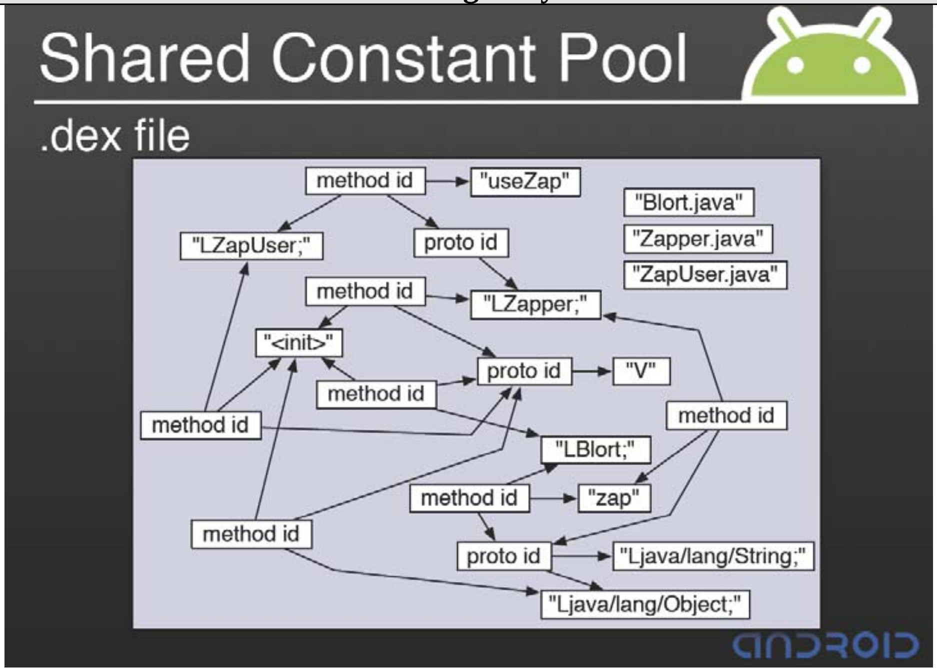
The ’702 Patent	Infringed By
	`` if (annotate) { wordData.writeIndexAnnotation(out, ItemType.TYPE_CODE_ITEM, "\nmethod code index:\n\n"); getStatistics().writeAnnotation(out); out.finishAnnotating(); } return out; } /** * Generates and returns statistics for all the items in the file. * * @return {@code non-null;} the statistics */ public Statistics getStatistics() { Statistics stats = new Statistics(); for (Section s : sections) { stats.addAll(s); } return stats; } /** * Calculates the signature for the {@code .dex} file in the * given array, and modify the array to contain it. * * @param bytes {@code non-null;} the bytes of the file */ private static void calcSignature(byte[] bytes) { MessageDigest md; try { md = MessageDigest.getInstance("SHA-1"); } catch (NoSuchAlgorithmException ex) { throw new RuntimeException(ex); } md.update(bytes, 32, bytes.length - 32); try { ```

The ’702 Patent	Infringed By
	<pre> int amt = md.digest(bytes, 12, 20); if (amt != 20) { throw new RuntimeException("unexpected digest write: " + amt + " bytes"); } } catch (DigestException ex) { throw new RuntimeException(ex); } } /** * Calculates the checksum for the {@code .dex} file in the * given array, and modify the array to contain it. * * @param bytes {@code non-null;} the bytes of the file */ private static void calcChecksum(byte[] bytes) { Adler32 a32 = new Adler32(); a32.update(bytes, 12, bytes.length - 12); int sum = (int) a32.getValue(); bytes[8] = (byte) sum; bytes[9] = (byte) (sum >> 8); bytes[10] = (byte) (sum >> 16); bytes[11] = (byte) (sum >> 24); } }</pre> dalvik/dx/src/com/android/dx/dex/file/DexFile.java. See also *See also*: dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
forming a shared table comprising	The Android dx tool forms a shared table of the duplicated elements from the plurality of

The ’702 Patent	Infringed By
said plurality of duplicated elements;	class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 15-20. The Dalvik Presentation shows the elements of the class files combining into a shared constant pool (shared tables) in the .dex file.  (Dalvik Presentation, slide 15) In the illustration above, each of “string_ids,” “type_ids” and “method_ids” are examples of the shared tables (or, equivalently, a collective shared table). In addition, the discussion of the “Shared Constant Pool” in the Dalvik Video explains that the duplicated elements in the class files are consolidated into the shared constant pool (shared table) of the .dex file. *See* Dalvik Presentation, slides 15-21. For example, slide 19 of the Dalvik Presentation shows the separate class files having

The ’702 Patent	Infringed By
	duplicated elements.  (Dalvik Presentation, slide 19) Next, slide 20 of the Dalvik Presentation shows a representation of the class files after being processed into a single .dex file, with the duplicate elements removed; the elements are then stored in a shared constant pool (shared table):

The ’702 Patent	Infringed By



(Dalvik Presentation, slide 20)

In the Android source code, *see also generally*:

> “Interfaces and implementation of things related to the constant pool.
> PACKAGES USED:
> * com.android.dx.rop.type
> * com.android.dx.util”

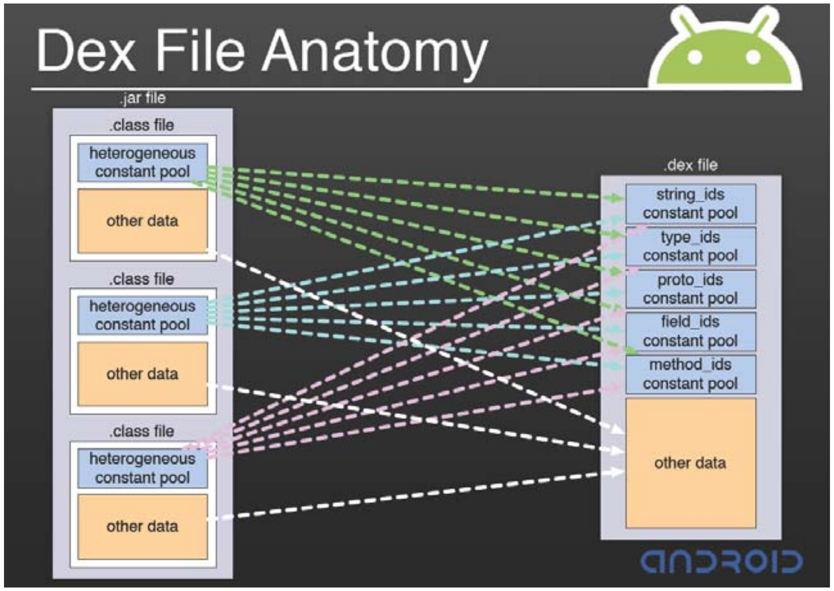
dalvik/dx/src/com/android/dx/rop/cst/package.html.

See also:

dalvik/dx/src/com/android/dx/dex/file/DexFile.java
dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java

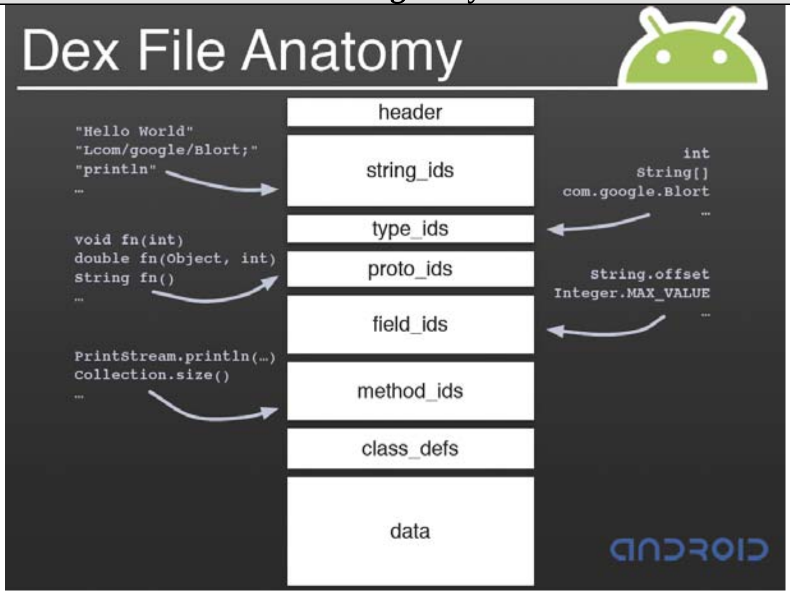
<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java
dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java</td></tr>
<tr><td>removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and</td><td>The Android dx tool removes the duplicated elements from the plurality of class files and obtains a plurality of reduced class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 15-20.

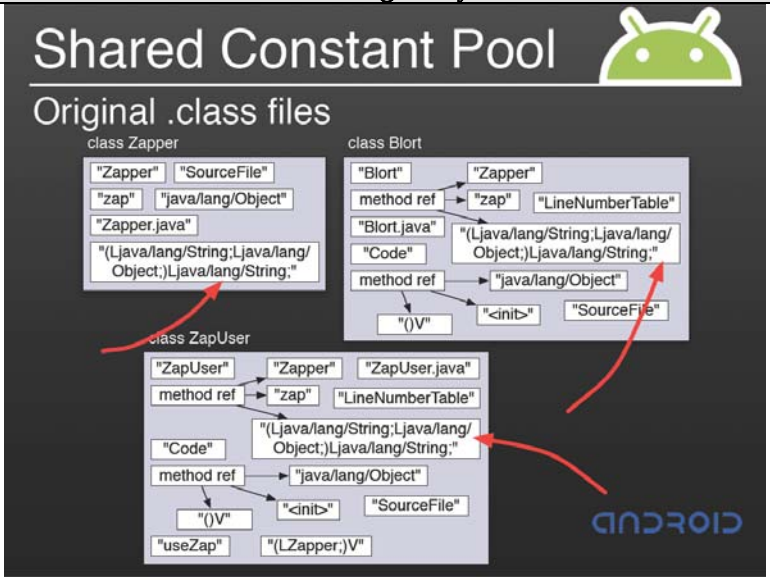
The Dalvik Presentation shows the class files combining into a shared constant pool (shared table) in the .dex file.

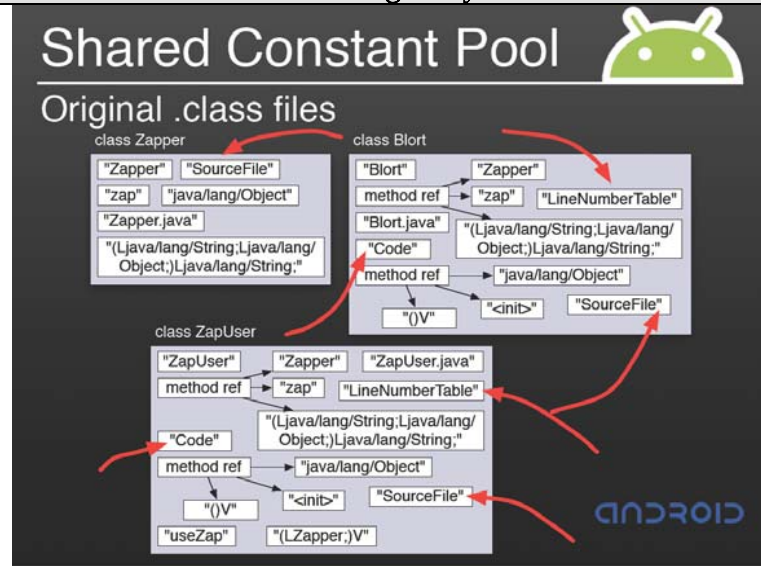


(Dalvik Presentation, slide 15)

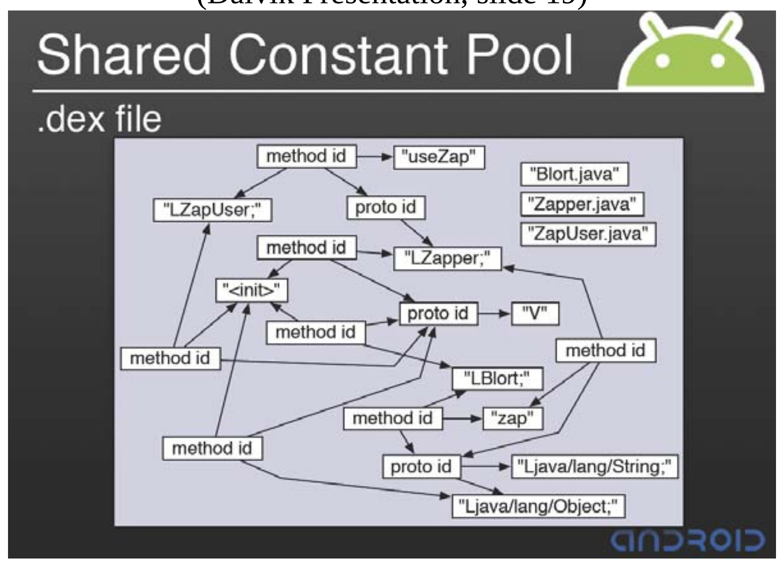
The original class files are combined into a single .dex file, which includes a plurality of reduced class files (i.e., with duplicates removed). This is also illustrated in slide 11 of the Dalvik presentation, which shows the anatomy of a .dex file:</td></tr>
</table>

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 11) Next, slides 18-20 of the Dalvik Presentation show the removal of the duplicated elements of the plurality of class files such that the resulting .dex file contains only one copy of each element in its shared constant pool (shared table).

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 18)

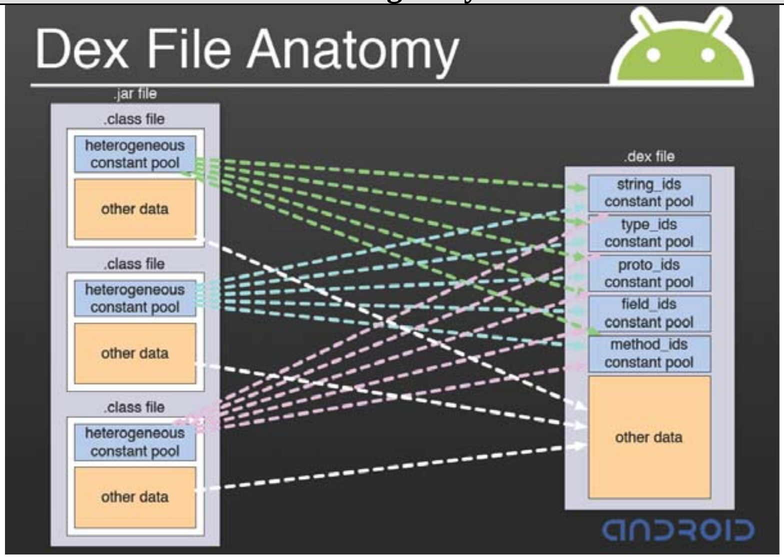
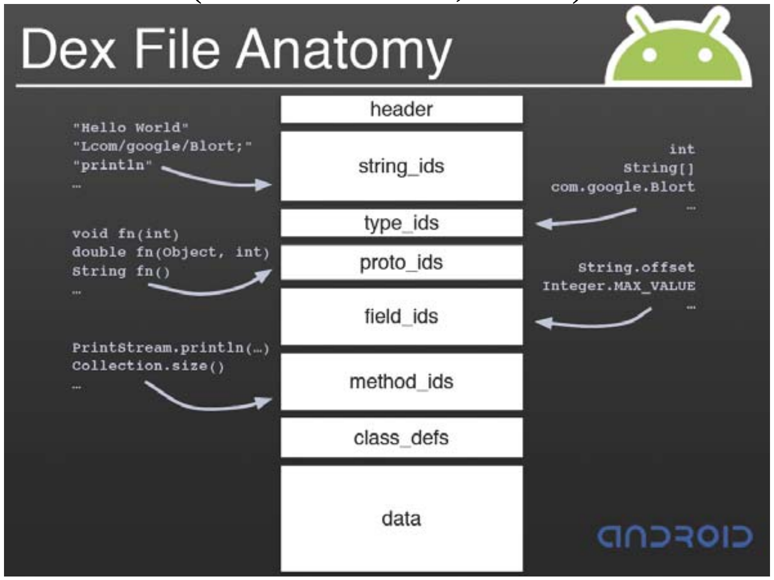
<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>


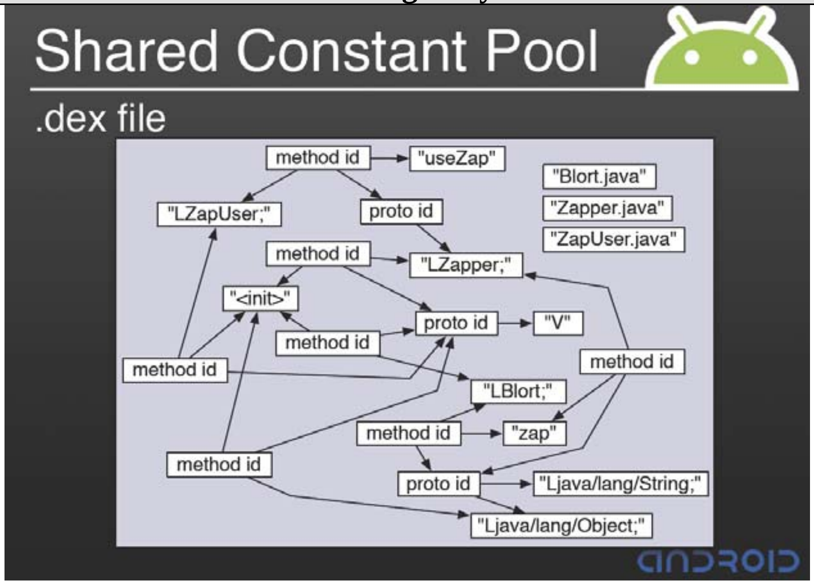
(Dalvik Presentation, slide 19)



(Dalvik Presentation, slide 20)
</td></tr>
</table>

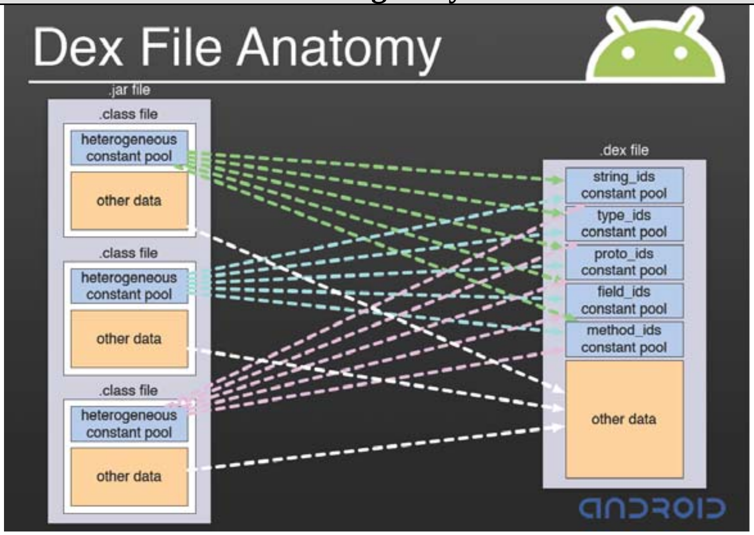
The ’702 Patent	Infringed By
	In the Android source code, *see also generally*: “Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util” dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java, dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
forming a multi-class file comprising said plurality of reduced class files and said shared table.	As explained above, the Android dx tool forms a multi-class file—the .dex file—comprising the reduced class files and a shared constant pool (shared table) such that duplicate elements have been removed. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 11 and 15-20. The Dalvik Presentation shows the original class files being combined into a .dex file (multi-class file) comprising the plurality of reduced class files and the shared constant pool (shared table):

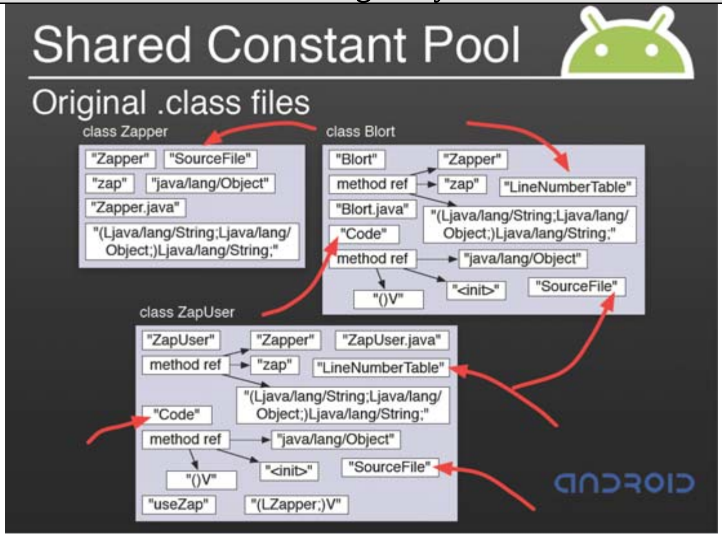
The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 15)  (Dalvik Presentation, slide 11)

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 20) In the Android source code, *see generally*: “Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code • com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util” dalvik\dx\src\com\android\dx\dex\cf\package.html.

The ’702 Patent	Infringed By
	See also: /** * Representation of an entire {@code .dex} (Dalvik EXecutable) * file, which itself consists of a set of Dalvik classes. */ public final class DexFile { /** {@code non-null;} word data section */ private final MixedItemSection wordData; dalvik\dx\src\com\android\dx\dex\file\DexFile.java. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java, dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

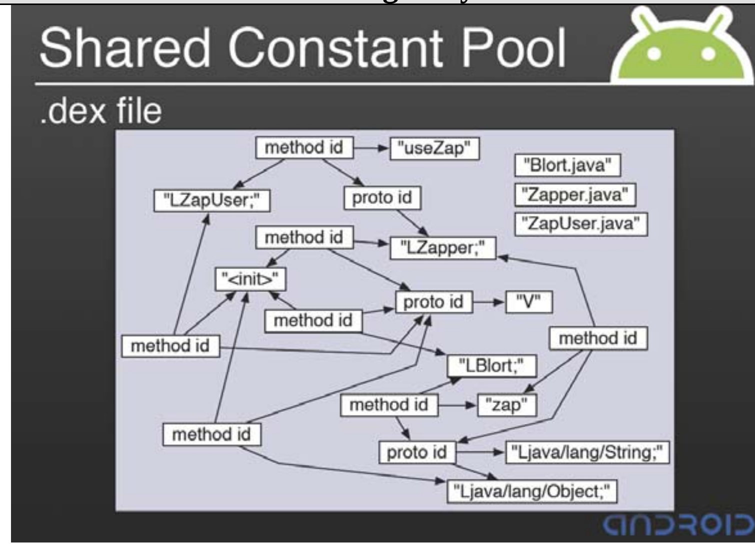
The ’702 Patent	Infringed By
5. The method of claim 1, wherein said step of determining a plurality of duplicated elements comprises:	*See* Claim 1, *supra*.
determining one or more constants shared between two or more class files.	The Android dx tool determines constants shared between two or more class files. This process is explained in the Dalvik Video at time 7:20-9:25 and Dalvik Presentation, slides 11-20. The Dalvik Presentation shows the elements of the class files identified for combining into a shared constant pool (shared tables) in the .dex file.

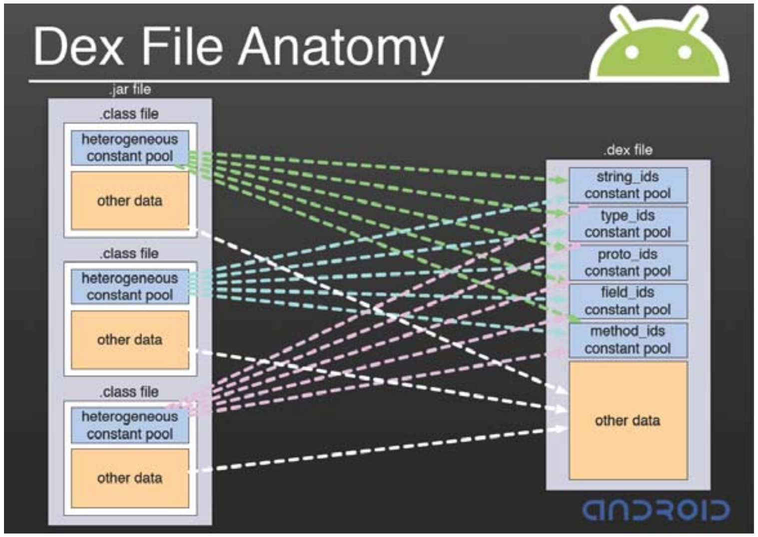
The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 15) In the illustration above, each of “string_ids,” “type_ids” and “method_ids” are examples of the shared tables (or, equivalently, a collective shared table). In addition, the discussion of the “Shared Constant Pool” in the Dalvik Video explains that the duplicated elements in the class files are consolidated into the shared constant pool (shared table) of the .dex file. *See* Dalvik Presentation, slides 15-21. For example, slide 19 of the Dalvik Presentation shows the separate class files having duplicated elements.

<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>


(Dalvik Presentation, slide 19)

Next, slide 20 of the Dalvik Presentation shows a representation of the class files after being processed into a single .dex file, with the duplicate elements removed; the elements are then stored in a shared constant pool (shared table):
</td></tr>
</table>

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 20) In the Android source code, *see also generally*: “Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util” dalvik/dx/src/com/android/dx/rop/cst/package.html from Android 2.2 (Nov. 2, 2010). *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java, dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

The ’702 Patent	Infringed By
6. The method of claim 5, wherein said step of forming a shared table comprises:	*See* Claim 1, *supra*.
forming a shared constant table comprising said one or more constants shared between said two or more class files.	The Android dx tool forms a shared constant table comprising the constants shared between the two or more class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slide 15. The Dalvik Presentation at 7:20-9:25 shows the elements of the class files combining into a shared constant pool (shared tables) in the .dex file.  (Dalvik Presentation, slide 15) In the illustration above, each of “string_ids,” “type_ids” and “method_ids” are examples of the shared tables (or, equivalently, a collective shared table). In addition, the discussion of the “Shared Constant Pool” in the Dalvik Video at 7:20-9:25

pa-1435312

The ’702 Patent	Infringed By
	explains that the duplicated elements in the class files are consolidated into the shared constant pool (shared table) of the .dex file. *See* Dalvik Presentation, slides 15-21. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java, dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

The ’702 Patent	Infringed By
7. A computer program product comprising:	Android is a computer program product.
a computer usable medium having computer readable program code embodied therein for pre-processing class files, said computer program product comprising:	Android is computer readable program, including computer readable program code for pre-processing class files. Further, Android is stored on a computer usable medium, e.g., RAM of a device or computer running Android. *See* corresponding element of claim 1, *supra*.
computer readable program code configured to cause a computer to determine a plurality of duplicated elements in a plurality of class files;	The Android dx tool determines a plurality of duplicated elements in a plurality of class files, as explained in the Dalvik Video and Dalvik Presentation: *See* corresponding element of claim 1, *supra*.
computer readable program code configured to cause a computer to form a shared table comprising said plurality of duplicated elements;	The Android dx tool forms a shared table of the duplicated elements from the plurality of class files. This process is explained in the Dalvik Video and Dalvik Presentation. *See* corresponding element of claim 1, *supra*.
computer readable program code configured to cause a computer to remove said duplicated elements from said plurality of class files to obtain a plurality of reduced class	The Android dx tool removes the duplicated elements from the plurality of class files and obtains a plurality of reduced class files. This process is explained in the Dalvik Video and Dalvik Presentation. *See* corresponding element of claim 1, *supra*.

The ’702 Patent	Infringed By
files; and	
computer readable program code configured to cause a computer to form a multi-class file comprising said plurality of reduced class files and said shared table.	As explained above, the Android dx tool forms a multi-class file—the .dex file—comprising the reduced class files and a shared constant pool (shared table) such that duplicate elements have been removed. This process is explained in the Dalvik Video and Dalvik Presentation. *See* corresponding element of claim 1, *supra*.

The ’702 Patent	Infringed By
11. The computer program product of claim 7, wherein said computer readable program code configured to cause a computer to determine said plurality of duplicated elements comprises:	*See* corresponding element of claim 5, *supra*.
computer readable program code configured to cause a computer to determine one or more constants shared between two or more class files.	*See* corresponding element of claim 5, *supra*.

The ’702 Patent	Infringed By
12. The computer program product of claim 11, wherein said computer readable program code configured to cause a computer to form said shared table comprises:	*See* corresponding element of claim 6, *supra*.

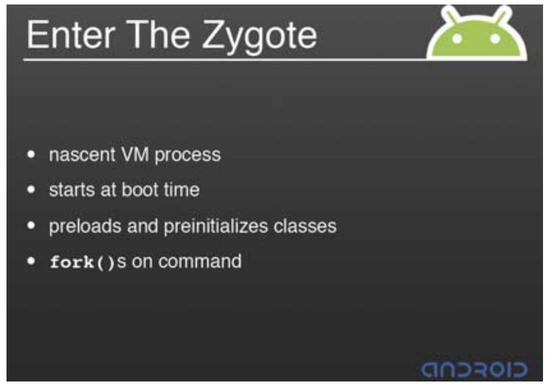
The ’702 Patent	Infringed By
computer readable program code configured to cause a computer to form a shared constant table comprising said one or more constants shared between said two or more class files.	*See* corresponding element of claim 6, *supra*.

The ’702 Patent	Infringed By
13. An apparatus comprising:	Any device or computer which can run the Android dx tool.
a processor;	A processor or CPU of the device or computer running Android.
a memory coupled to said processor;	A storage memory, e.g., RAM, of the device or computer running Android.
a plurality of class files stored in said memory;	See above disclosures of the plurality of class files that are processed by the dx tool into a .dex file. The class files would necessarily be stored in the memory, e.g., RAM, of the computer while they are being processed by the dx tool. *See* corresponding element of claim 1, *supra*.
a process executing on said processor, said process configured to form a multi-class file comprising:	See corresponding disclosures in the claims above, detailing the process by which the dx tool forms a multi-class .dex file. *See* corresponding element of claim 1, *supra*.
a plurality of reduced class files obtained from said plurality of class files by removing one or more elements that are duplicated between two or more of said plurality of class files; and	See corresponding disclosures in the claims above, detailing the removal of duplicate elements among the plurality of class files when the dx tool forms a multi-class .dex file. *See* corresponding element of claim 1, *supra*.
a shared table comprising said duplicated elements.	See corresponding disclosures in the claims above, detailing the shared constant pool of the duplicated elements within the .dex file. *See* corresponding element of claim 1, *supra*.

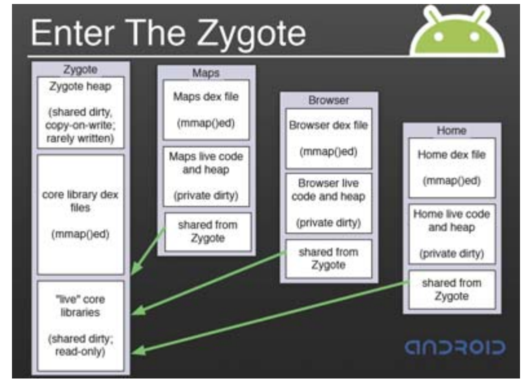
The ’702 Patent	Infringed By
15. The apparatus of claim 13, wherein said duplicated elements comprise elements of constant pools of respective class files, said shared table comprising a shared constant pool.	*See* corresponding element of claims 1, 5, and 6, *supra*.

<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td>16. The apparatus of claim 13, further comprising:</td><td>See Claim 13, supra.</td></tr>
<tr><td>a virtual machine having a class loader and a runtime data area, said class loader configured to obtain and load said multi-class file into said runtime data area.</td><td>

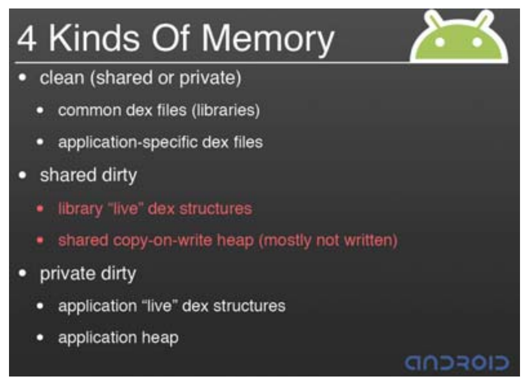
Android includes the Dalvik Virtual Machine, which includes a class loader (the Zygote) and runtime data area, where the class loader obtains and loads the multi-class file (the .dex file) into the runtime data area.



(Dalvik Presentation, Slide 25)



(Dalvik Presentation, Slide 26)



(Dalvik Presentation, Slide 27)

See also corresponding element of claim 1, supra.

</td></tr>
</table>

pa-1435312

EXHIBIT D
Preliminary Infringement Contentions for the ’447 Patent

NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’447 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The ’447 Patent	Infringed By
[1-pre] 1. A method for providing security, the method comprising the steps of:	Android includes methods for providing security. *See generally, e.g.*: • dalvik\vm\native\InternalNative.c • dalvik\vm\native\java_security_AccessController.c • dalvik\vm\native\java_lang_VMClassLoader.c • source code files in libcore\security\src\main\java\java\security • source code files in libcore\security-kernel\src\main\java\java\security • libcore\security\src\main\java\org\apache\harmony\security *See also, e.g.*: • Android APIs for “java.security,” available at http://developer.android.com/reference/java/security/package-summary.html • Android Framework Topics for “Security and Permissions,” available at http://developer.android.com/guide/topics/security/security.html

The ’447 Patent	Infringed By
	• Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/permission-element.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/application-element.html • Android Framework Topics for “The AndroidManifest.xml File,” available at http://developer.android.com/guide/topics/manifest/manifest-intro.html *See also, e.g.*: • libcore\security\src\test
[1-a] establishing one or more protection domains, wherein a protection domain is associated with zero or more permissions;	Android’s security framework establishes one or more protection domains, wherein a protection domain is associated with zero or more permissions. *See, e.g.:* libcore\security\src\main\java\java\security\ProtectionDomain.java: /** * {@code ProtectionDomain} represents all permissions that are granted to a * specific code source. The {@link ClassLoader} associates each class with the * corresponding {@code ProtectionDomain}, depending on the location and the * certificates (encapsulates in {@link CodeSource}) it loads the code from. * <p> * A class belongs to exactly one protection domain and the protection domain * can not be changed during the lifetime of the class. */ public class ProtectionDomain { // CodeSource for this ProtectionDomain private CodeSource codeSource; // Static permissions for this ProtectionDomain private PermissionCollection permissions;

The ’447 Patent	Infringed By
	// ClassLoader private ClassLoader classLoader; // Set of principals associated with this ProtectionDomain private Principal[] principals; // false if this ProtectionDomain was constructed with static // permissions, true otherwise. private boolean dynamicPerms; /** * Constructs a new instance of {@code ProtectionDomain} with the specified * code source and the specified static permissions. * <p> * If {@code permissions} is not {@code null}, the {@code permissions} * collection is made immutable by calling * {@link PermissionCollection#setReadOnly()} and it is considered as * granted statically to this {@code ProtectionDomain}. * <p> * The policy will not be consulted by access checks against this {@code * ProtectionDomain}. * <p> * If {@code permissions} is {@code null}, the method {@link * ProtectionDomain#implies(Permission)} always returns {@code false}. * * @param cs * the code source associated with this domain, maybe {@code * null}. * @param permissions * the {@code PermissionCollection} containing all permissions to * be statically granted to this {@code ProtectionDomain}, maybe

The ’447 Patent	Infringed By
	`` * {@code null}. */ public ProtectionDomain(CodeSource cs, PermissionCollection permissions) { this.codeSource = cs; if (permissions != null) { permissions.setReadOnly(); } this.permissions = permissions; //this.classLoader = null; //this.principals = null; //dynamicPerms = false; } /** * Constructs a new instance of {@code ProtectionDomain} with the specified * code source, the permissions, the class loader and the principals. * <p> * If {@code permissions} is {@code null}, and access checks are performed * against this protection domain, the permissions defined by the policy are * consulted. If {@code permissions} is not {@code null}, the {@code * permissions} collection is made immutable by calling * {@link PermissionCollection#setReadOnly()}. If access checks are * performed, the policy and the provided permission collection are checked. * <p> * External modifications of the provided {@code principals} array has no * impact on this {@code ProtectionDomain}. * * @param cs * the code source associated with this domain, maybe {@code * null}. * @param permissions * the permissions associated with this domain, maybe {@code ```

The ’447 Patent	Infringed By
	`` * null}. * @param cl * the class loader associated with this domain, maybe {@code * null}. * @param principals * the principals associated with this domain, maybe {@code * null}. */ public ProtectionDomain(CodeSource cs, PermissionCollection permissions, ClassLoader cl, Principal[] principals) { this.codeSource = cs; if (permissions != null) { permissions.setReadOnly(); } this.permissions = permissions; this.classLoader = cl; if (principals != null) { this.principals = new Principal[principals.length]; System.arraycopy(principals, 0, this.principals, 0, this.principals.length); } dynamicPerms = true; } … /** * Returns the static permissions that are granted to this {@code * ProtectionDomain}. * * @return the static permissions that are granted to this {@code * ProtectionDomain}, maybe {@code null}. */ public final PermissionCollection getPermissions() { ```

The ’447 Patent	Infringed By
	return permissions; } *See also, e.g.*: • Android APIs for “java.security,” available at http://developer.android.com/reference/java/security/package-summary.html • Android Framework Topics for “Security and Permissions,” available at http://developer.android.com/guide/topics/security/security.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/permission-element.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/application-element.html • Android Framework Topics for “The AndroidManifest.xml File,” available at http://developer.android.com/guide/topics/manifest/manifest-intro.html
[1-b] establishing an association between said one or more protection domains and one or more classes of one or more objects; and	Android’s security framework establishes an association between said one or more protection domains and one or more classes of one or more objects. *See* Claim 1-a, *supra*. *See also, e.g.:* dalvik\vm\native\java_lang_VMClassLoader.c: /* * java.lang.VMClassLoader */ … /* * static Class defineClass(ClassLoader cl, String name, * byte[] data, int offset, int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object.

The ’447 Patent	Infringed By
	`` */ static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; StringObject* nameObj = (StringObject*) args[1]; const u1* data = (const u1*) args[2]; int offset = args[3]; int len = args[4]; Object* pd = (Object*) args[5]; char* name = NULL; name = dvmCreateCstrFromString(nameObj); LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n", loader, name, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); free(name); RETURN_VOID(); } /* * static Class defineClass(ClassLoader cl, byte[] data, int offset, * int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. Deprecated version of * previous method, lacks name parameter. */ static void Dalvik_java_lang_VMClassLoader_defineClass2(const u4* args, JValue* pResult) ```

The ’447 Patent	Infringed By
	{ Object* loader = (Object*) args[0]; const u1* data = (const u1*) args[1]; int offset = args[2]; int len = args[3]; Object* pd = (Object*) args[4]; LOGE("ERROR: defineClass(%p, %p, %d, %d, %p)\n", loader, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); RETURN_VOID(); }
[1-c] determining whether an action requested by a particular object is permitted based on said association between said one or more protection domains and said one or more classes.	Android’s security framework determines whether an action requested by a particular object is permitted based on said association between said one or more protection domains and said one or more classes. *See* Claim 1-a and 1-b, *supra*. *See also, e.g.*: libcore\security\src\main\java\java\security\ProtectionDomain.java: /** * Indicates whether the specified permission is implied by this {@code * ProtectionDomain}. * <p> * If this {@code ProtectionDomain} was constructed with * {@link #ProtectionDomain(CodeSource, PermissionCollection)}, the * specified permission is only checked against the permission collection * provided in the constructor. If {@code null} was provided, {@code false}

The ’447 Patent	Infringed By
	<pre> * is returned. * <p> * If this {@code ProtectionDomain} was constructed with * {@link #ProtectionDomain(CodeSource, PermissionCollection, ClassLoader, Principal[])} * , the specified permission is checked against the policy and the * permission collection provided in the constructor. * * @param permission * the permission to check against the domain. * @return {@code true} if the specified {@code permission} is implied by * this {@code ProtectionDomain}, {@code false} otherwise. */ public boolean implies(Permission permission) { // First, test with the Policy, as the default Policy.implies() // checks for both dynamic and static collections of the // ProtectionDomain passed... if (dynamicPerms && Policy.getAccessiblePolicy().implies(this, permission)) { return true; } // ... and we get here if // either the permissions are static // or Policy.implies() did not check for static permissions // or the permission is not implied return permissions == null ? false : permissions.implies(permission); }</pre> Android APIs for “ProtectionDomain,” available at http://developer.android.com/reference/java/security/ProtectionDomain.html:

The ’447 Patent	Infringed By
	public **ProtectionDomain** (CodeSource cs, PermissionCollection permissions) Since: API Level 1 Constructs a new instance of `ProtectionDomain` with the specified code source and the specified static permissions. If `permissions` is not `null`, the `permissions` collection is made immutable by calling `setReadOnly()` and it is considered as granted statically to this `ProtectionDomain`. The policy will not be consulted by access checks against this `ProtectionDomain`. If `permissions` is `null`, the method `implies(Permission)` always returns `false`. **Parameters** *cs* the code source associated with this domain, maybe `null`. *permissions* the `PermissionCollection` containing all permissions to be statically granted to this `ProtectionDomain`, maybe `null`. public **ProtectionDomain** (CodeSource cs, PermissionCollection permissions, ClassLoader cl, Principal[] principals) Since: API Level 1 Constructs a new instance of `ProtectionDomain` with the specified code source, the permissions, the class loader and the principals. If `permissions` is `null`, and access checks are performed against this protection domain, the permissions defined by the policy are consulted. If `permissions` is not `null`, the `permissions` collection is made immutable by calling `setReadOnly()`. If access checks

The ’447 Patent	Infringed By
	are performed, the policy and the provided permission collection are checked. External modifications of the provided `principals` array has no impact on this `ProtectionDomain`. **Parameters** *cs* the code source associated with this domain, maybe `null`. *permissions* the permissions associated with this domain, maybe `null`. *cl* the class loader associated with this domain, maybe `null`. *principals* the principals associated with this domain, maybe `null`. libcore\security\src\main\java\java\security\Policy.java: /** * Indicates whether the specified {@code Permission} is implied by the * {@code PermissionCollection} of the specified {@code ProtectionDomain}. * * @param domain * the {@code ProtectionDomain} for which the permission should * be granted. * @param permission * the {@code Permission} for which authorization is to be * verified. * @return {@code true} if the {@code Permission} is implied by the {@code * ProtectionDomain}, {@code false} otherwise. */ public boolean implies(ProtectionDomain domain, Permission permission) { if (domain != null) {

<table>
<tr><th>The ’447 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

```
            PermissionCollection total = getPermissions(domain);
            PermissionCollection inherent = domain.getPermissions();
            if (total == null) {
                total = inherent;
            } else if (inherent != null) {
                for (Enumeration<Permission> en = inherent.elements(); en.hasMoreElements();) {
                    total.add(en.nextElement());
                }
            }
            if (total != null && total.implies(permission)) {
                return true;
            }
        }
        return false;
    }
```

libcore\luni\src\main\java\java\lang\ SecurityManager.java:

```
/**
 * <strong>Warning:</strong> security managers do <strong>not</strong> provide a
 * secure environment for executing untrusted code. Untrusted code cannot be
 * safely isolated within the Dalvik VM.
 *
 * <p>Provides security verification facilities for applications. {@code
 * SecurityManager} contains a set of {@code checkXXX} methods which determine
 * if it is safe to perform a specific operation such as establishing network
 * connections, modifying files, and many more. In general, these methods simply
 * return if they allow the application to perform the operation; if an
 * operation is not allowed, then they throw a {@link SecurityException}. The
 * only exception is {@link #checkTopLevelWindow(Object)}, which returns a
 * boolean to indicate permission.
 */
public class SecurityManager {
```

</td></tr>
</table>

The ’447 Patent	Infringed By
	… /** * Checks whether the calling thread is allowed to access the resource being * guarded by the specified permission object. * * @param permission * the permission to check. * @throws SecurityException * if the requested {@code permission} is denied according to * the current security policy. */ public void checkPermission(Permission permission) { try { inCheck = true; AccessController.checkPermission(permission); } finally { inCheck = false; } } /** * Checks whether the specified security context is allowed to access the * resource being guarded by the specified permission object. * * @param permission * the permission to check. * @param context * the security context for which to check permission. * @throws SecurityException * if {@code context} is not an instance of {@code * AccessControlContext} or if the requested {@code permission} * is denied for {@code context} according to the current

The ’447 Patent	Infringed By
	`` * security policy. */ public void checkPermission(Permission permission, Object context) { try { inCheck = true; // Must be an AccessControlContext. If we don't check // this, then applications could pass in an arbitrary // object which circumvents the security check. if (context instanceof AccessControlContext) { ((AccessControlContext) context).checkPermission(permission); } else { throw new SecurityException(); } } finally { inCheck = false; } } } ``` libcore\security-kernel\src\main\java\java\security\AccessController.java: ``` /** * Checks the specified permission against the vm's current security policy. * The check is performed in the context of the current thread. This method * returns silently if the permission is granted, otherwise an {@code * AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * the current execution context has been granted the specified permission. * If privileged operations are on the execution context, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> ```

The ’447 Patent	Infringed By
	* This method delegates the permission check to * {@link AccessControlContext#checkPermission(Permission)} on the current * callers' context obtained by {@link #getContext()}. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessControlContext#checkPermission(Permission) * * @since Android 1.0 */ public static void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("permission can not be null"); } getContext().checkPermission(perm); } libcore\security-kernel\src\main\java\java\security\AccessController.java: /** * Checks the specified permission against the vm's current security policy. * The check is performed in the context of the current thread. This method * returns silently if the permission is granted, otherwise an {@code * AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * the current execution context has been granted the specified permission.

The ’447 Patent	Infringed By
	* If privileged operations are on the execution context, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * This method delegates the permission check to * {@link AccessControlContext#checkPermission(Permission)} on the current * callers' context obtained by {@link #getContext()}. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessControlContext#checkPermission(Permission) * * @since Android 1.0 */ public static void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("permission can not be null"); } getContext().checkPermission(perm); } libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: ProtectionDomain[] context; … /** * Checks the specified permission against the vm's current security policy.

The ’447 Patent	Infringed By
	` * The check is based on this {@code AccessControlContext} as opposed to the` ` * {@link AccessController#checkPermission(Permission)} method which` ` * performs access checks based on the context of the current thread. This` ` * method returns silently if the permission is granted, otherwise an` ` * {@code AccessControlException} is thrown.` ` * <p>` ` * A permission is considered granted if every {@link ProtectionDomain} in` ` * this context has been granted the specified permission.` ` * <p>` ` * If privileged operations are on the call stack, only the {@code` ` * ProtectionDomain}s from the last privileged operation are taken into` ` * account.` ` * <p>` ` * If inherited methods are on the call stack, the protection domains of the` ` * declaring classes are checked, not the protection domains of the classes` ` * on which the method is invoked.` ` *` ` * @param perm` ` *            the permission to check against the policy` ` * @throws AccessControlException` ` *             if the specified permission is not granted` ` * @throws NullPointerException` ` *             if the specified permission is {@code null}` ` * @see AccessController#checkPermission(Permission)` ` * @since Android 1.0` ` */` `public void checkPermission(Permission perm) throws AccessControlException {` `    if (perm == null) {` `        throw new NullPointerException("Permission cannot be null");` `    }` `    for (int i = 0; i < context.length; i++) {` `        if (!context[i].implies(perm)) {`

The ’447 Patent	Infringed By
	throw new AccessControlException("Permission check failed " + perm, perm); } } if (inherited != null) { inherited.checkPermission(perm); } }

The ’447 Patent	Infringed By
2. The method of claim 1, wherein:	*See* Claim 1, *supra*.
at least one protection domain of said one or more protection domains is associated with a code identifier;	*See* Claim 1-a and 1-b, *supra*. *E.g.:* dalvik\vm\native\java_lang_VMClassLoader.c: /* * java.lang.VMClassLoader */ … /* * static Class defineClass(ClassLoader cl, String name, * byte[] data, int offset, int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. */ static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0];

The ’447 Patent	Infringed By
	`` StringObject* nameObj = (StringObject*) args[1]; const u1* data = (const u1*) args[2]; int offset = args[3]; int len = args[4]; Object* pd = (Object*) args[5]; char* name = NULL; name = dvmCreateCstrFromString(nameObj); LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n", loader, name, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); free(name); RETURN_VOID(); } /* * static Class defineClass(ClassLoader cl, byte[] data, int offset, * int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. Deprecated version of * previous method, lacks name parameter. */ static void Dalvik_java_lang_VMClassLoader_defineClass2(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; const u1* data = (const u1*) args[1]; int offset = args[2]; int len = args[3]; ```

The ’447 Patent	Infringed By
	`` Object* pd = (Object*) args[4]; LOGE("ERROR: defineClass(%p, %p, %d, %d, %p)\n", loader, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); RETURN_VOID(); } ``` *See also, e.g.*: libcore\security\src\main\java\java\security\ CodeSource.java: ``` /** * {@code CodeSource} encapsulates the location from where code is loaded and * the certificates that were used to verify that code. This information is used * by {@code SecureClassLoader} to define protection domains for loaded classes. * * @see SecureClassLoader * @see ProtectionDomain */ public class CodeSource implements Serializable { private static final long serialVersionUID = 4977541819976013951L; // Location of this CodeSource object private URL location; // Array of certificates assigned to this CodeSource object private transient java.security.cert.Certificate[] certs; // Array of CodeSigners ```

The ’447 Patent	Infringed By
	`` private transient CodeSigner[] signers; // SocketPermission() in implies() method takes to many time. // Need to cache it for better performance. private transient SocketPermission sp; // Cached factory used to build CertPath-s in <code>getCodeSigners()</code>. private transient CertificateFactory factory; /** * Constructs a new instance of {@code CodeSource} with the specified * {@code URL} and the {@code Certificate}s. * * @param location * the {@code URL} representing the location from where code is * loaded, maybe {@code null}. * @param certs * the {@code Certificate} used to verify the code, loaded from * the specified {@code location}, maybe {@code null}. */ public CodeSource(URL location, Certificate[] certs) { this.location = location; if (certs != null) { this.certs = new Certificate[certs.length]; System.arraycopy(certs, 0, this.certs, 0, certs.length); } } /** * Constructs a new instance of {@code CodeSource} with the specified * {@code URL} and the {@code CodeSigner}s. * ```

The ’447 Patent	Infringed By
	`* @param location` `*          the {@code URL} representing the location from where code is` `*          loaded, maybe {@code null}.` `* @param signers` `*          the {@code CodeSigner}s of the code, loaded from the specified` `*          {@code location}. Maybe {@code null}.` `*/` `public CodeSource(URL location, CodeSigner[] signers) {` `   this.location = location;` `   if (signers != null) {` `      this.signers = new CodeSigner[signers.length];` `      System.arraycopy(signers, 0, this.signers, 0, signers.length);` `   }` `}` `…`
at least one class of said one or more classes is associated with said code identifier; and	*See* Claim 1-b, *supra*, and above.
the step of establishing an association between said one or more protection domains and said one or more classes of one or more objects further includes the step of associating said one or more protection domains and said one or more classes based on said code identifier.	*See* Claim 1, *supra*, and above.

The ’447 Patent	Infringed By
3. The method of claim 2, wherein	*See* Claim 2, *supra*.

The ’447 Patent	Infringed By
said code identifier indicates a source of code used to define each class of said one or more classes.	

The ’447 Patent	Infringed By
4. The method of claim 2, wherein said code identifier indicates a key associated with each class of said one or more classes.	*See* Claim 2, *supra*. The certificate mentioned in Claim 2, *supra*, includes a key. *See, e.g.*: libcore\security\src\main\java\java\security\ CodeSource.java: /** * {@code CodeSource} encapsulates the location from where code is loaded and * the certificates that were used to verify that code. This information is used * by {@code SecureClassLoader} to define protection domains for loaded classes. * * @see SecureClassLoader * @see ProtectionDomain */ … // Array of certificates assigned to this CodeSource object private transient java.security.cert.Certificate[] certs; … /** * Constructs a new instance of {@code CodeSource} with the specified * {@code URL} and the {@code Certificate}s. * * @param location * the {@code URL} representing the location from where code is * loaded, maybe {@code null}.

The ’447 Patent	Infringed By
	` * @param certs` ` *           the {@code Certificate} used to verify the code, loaded from` ` *           the specified {@code location}, maybe {@code null}.` ` */` `public CodeSource(URL location, Certificate[] certs) {` `    this.location = location;` `    if (certs != null) {` `        this.certs = new Certificate[certs.length];` `        System.arraycopy(certs, 0, this.certs, 0, certs.length);` `    }` `}` `…` `/**` ` * Returns the certificates of this {@code CodeSource}. If the` ` * {@link #CodeSource(URL, CodeSigner[])} constructor was used to create` ` * this instance, the certificates are obtained from the supplied signers.` ` * <p>` ` * External modifications of the returned {@code Certificate[]} has no` ` * impact on this {@code CodeSource}.` ` *` ` * @return the certificates of this {@code CodeSource} or {@code null} if` ` *         there is none.` ` */` `public final Certificate[] getCertificates() {` `    getCertificatesNoClone();` `    if (certs == null) {` `        return null;` `    }` `    Certificate[] tmp = new Certificate[certs.length];` `    System.arraycopy(certs, 0, tmp, 0, certs.length);` `    return tmp;` `}`

The ’447 Patent	Infringed By
	… libcore\security\src\main\java\java\security\Certificate.java: /** * {@code Certificate} represents an identity certificate, such as X.509 or PGP. * Note: A {@code Certificate} instances does not make any statement about the * validity of itself. It's in the responsibility of the application to verify * the validity of its certificates. * * @deprecated Replaced by behavior in {@link java.security.cert} * @see java.security.cert.Certificate */ X.509 is an internet standard certificate format. *See*, *e.g.*, RFC2459, available at www.ietf.org/rfc/rfc2459.txt (discussing keys and certificates). Information about PGP certificates is available at, *e.g.*, www.pgpi.org; http://en.wikipedia.org/wiki/Pretty_Good_Privacy (and references cited therein). *See also*, *e.g.*: libcore\security\src\main\java\java\security\ Key.java: /** * {@code Key} is the common interface for all keys. * * @see PublicKey * @see PrivateKey */ public interface Key extends Serializable { … *See also*, *e.g.*, Android APIs for “java.security.cert,” available at http://developer.android.com/reference/java/security/cert/package-summary.html.

The ’447 Patent	Infringed By
	See also, e.g.: • Android Framework Topics for “Security and Permissions,” available at http://developer.android.com/guide/topics/security/security.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/permission-element.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/application-element.html • Android Framework Topics for “The AndroidManifest.xml File,” available at http://developer.android.com/guide/topics/manifest/manifest-intro.html

The ’447 Patent	Infringed By
5. The method of claim 2, wherein said code identifier indicates a source of code used to define each class of said one or more classes and indicates a key associated with each class of said one or more classes.	*See* Claims 2 and 4, *supra*.

The ’447 Patent	Infringed By
6. The method of claim 2, wherein the step of associating said one or more protection domains and said one or more classes based on said code identifier further includes associating said one or more protection domains and said one or more classes based on data	*See* Claim 2, *supra*. *See also, e.g.*: libcore\security\src\main\java\java\security\ CodeSource.java: /** * {@code CodeSource} encapsulates the location from where code is loaded and * the certificates that were used to verify that code. This information is used

persistently stored, wherein said data associates code identifiers with a set of one or more permissions.	* by {@code SecureClassLoader} to define protection domains for loaded classes. * * @see SecureClassLoader * @see ProtectionDomain */ public class CodeSource implements Serializable { libcore\security\src\main\java\java\security\Permission.java: /** * {@code Permission} is the common base class of all permissions that * participate in the access control security framework around * {@link AccessController} and {@link AccessControlContext}. A permission * constitutes of a name and associated actions. */ public abstract class Permission implements Guard, Serializable { *See also, e.g.*: libcore\security\src\main\java\java\security\ Key.java: /** * {@code Key} is the common interface for all keys. * * @see PublicKey * @see PrivateKey */ public interface Key extends Serializable { … *E.g.*, "Serializable" is generally understood as: In computer science, in the context of data storage and transmission, serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link

	to be "resurrected" later in the same or another computer environment.[1] When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. For many complex objects, such as those that make extensive use of references, this process is not straightforward. http://en.wikipedia.org/wiki/Serialization (footnote omitted). Android APIs for "java.io.Serializable," available at http://developer.android.com/reference/java/io/Serializable.html: Class Overview An empty marker interface for classes that want to support serialization and deserialization based on the ObjectOutputStream and ObjectInputStream classes. Implementing this interface is enough to make most classes serializable. If a class needs more fine-grained control over the serialization process (for example to implement compatibility with older versions of the class), it can achieve this by providing the following two methods (signatures must match exactly): private void writeObject(java.io.ObjectOutputStream out) throws IOException private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException *See also, e.g.*: • Android Framework Topics for "Security and Permissions," available at http://developer.android.com/guide/topics/security/security.html • Android Framework Topics for "Security and Permissions" under "The AndroidManifest.xml File," http://developer.android.com/guide/topics/manifest/permission-element.html • Android Framework Topics for "Security and Permissions" under "The AndroidManifest.xml File," http://developer.android.com/guide/topics/manifest/application-element.html • Android Framework Topics for "The AndroidManifest.xml File," available at http://developer.android.com/guide/topics/manifest/manifest-intro.html

The ’447 Patent	Infringed By
7. A method for providing security, the method comprising the steps of:	*See* Claim 1-pre, *supra*.
establishing one or more protection domains, wherein a protection domain is associated with zero or more permissions;	*See* Claim 1-a, *supra*.
establishing an association between said one or more protection domains and one or more sources of code; and	*See* Claim 1-a and 1-b, *supra*.
in response to executing code making a request to perform an action, determining whether said request is permitted based on a source of said code making said request and said association between said one or more protection domains and said one or more sources of code.	*See* Claim 1-c, *supra*.

The ’447 Patent	Infringed By
8. The method of claim 7, wherein the step of establishing an association between said one or more protection domains and said one or more sources of code further includes establishing an association between said one or more protection domains and said one or more sources of code and one or more	*See* Claims 2, 4, and 7, *supra*.

The ’447 Patent	Infringed By
keys associated with said one or more sources of code.	

The ’447 Patent	Infringed By
9. The method of claim 8, wherein the step of establishing an association between said one or more protection domains and said one or more sources of code and said one or more keys associated with said one or more sources of code further includes establishing said association between said one or more protection domains and said one or more sources of code and said one or more keys associated with said one or more sources of code based on data persistently stored, wherein said data associates particular sources of code and particular keys with a set of one or more permissions.	*See* Claims 6 and 8, *supra*.

The ’447 Patent	Infringed By
10. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices. These encompass a computer readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps described in the claim. *See* Claim 1-pre, *supra*.

The ’447 Patent	Infringed By
of:	
establishing one or more protection domains, wherein a protection domain is associated with zero or more permissions;	*See* Claim 1-a, *supra*.
establishing an association between said one or more protection domains and one or more classes of one or more objects; and	*See* Claim 1-b, *supra*.
determining whether an action requested by a particular object is permitted based on said association between said one or more protection domains and said one or more classes.	*See* Claim 1-c, *supra*.

The ’447 Patent	Infringed By
11. The computer readable medium of claim 10, wherein:	*See* Claim 10, *supra*.
at least one protection domain of said one or more protection domains is associated with a code identifier;	*See* Claims 1-a and 2, *supra*.
at least one class of said one or more classes is associated with said code identifier; and	*See* Claims 1-b and 2, *supra*.
the step of establishing an association between said one or more protection domains and said one or more classes of one or more objects further includes the step of associating said one or more protection domains and said one or	*See* Claim 1-c and 2, *supra*.

The ’447 Patent	Infringed By
more classes based on said code identifier.	

The ’447 Patent	Infringed By
12. The computer readable medium of claim 11, wherein said code identifier indicates a source of code used to define each class of said one or more classes.	*See* Claim 11, *supra*.

The ’447 Patent	Infringed By
13. The computer readable medium of claim 11, wherein said code identifier indicates a key associated with each class of said one or more classes.	*See* Claims 2, 4, and 11, *supra*.

The ’447 Patent	Infringed By
14. The computer readable medium of claim 11, wherein said code identifier indicates a source of code used to define each class of said one or more classes and indicates a key associated with each class of said one or more classes.	*See* Claims 2, 4, and 11, *supra*.

The ’447 Patent	Infringed By
15. The computer readable medium of claim 14, wherein the step of associating said one or more protection domains and said one or	*See* Claims 6 and 14, *supra*.

The ’447 Patent	Infringed By
more classes based on said code identifier further includes associating said one or more protection domains and said one or more classes based on data persistently stored, wherein said data associates code identifiers with a set of one or more permissions.	

The ’447 Patent	Infringed By
16. A computer-readable medium carrying one or more sequences of one or more instructions, wherein the execution of the one or more sequences of the one or more instructions causes the one or more processors to perform the steps of:	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices. These encompass a computer readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps described in the claim. *See* Claim 1-pre, *supra*.
establishing one or more protection domains, wherein a protection domain is associated with zero or more permissions;	*See* Claim 1 and 1-a, *supra*.
establishing an association between said one or more protection domains and one or more sources of code; and	*See* Claim 1, 1-a, and 1-b, *supra*.
in response to executing code making a request to perform an action, determining whether said request is permitted based on a source of said code making said request and said association between said one or more protection domains	*See* Claim 1 and 1-c, *supra*.

The ’447 Patent	Infringed By
and said one or more sources of code.	

The ’447 Patent	Infringed By
17. The computer readable medium of claim 16, wherein the step of establishing an association between said one or more protection domains and said one or more sources of code further includes establishing an association between said one or more protection domains and said one or more sources of code and one or more keys associated with said one or more sources of code.	*See* Claim 16, *supra*.

The ’447 Patent	Infringed By
18. The computer readable medium of claim 17, wherein the step of establishing an association between said one or more protection domains and said one or more sources of code and said one or more keys associated with said one or more sources of code further includes establishing said association between said one or more protection domains and said one or more sources of code and said one or more keys associated with said one or more sources of code based on	*See* Claim 17, *supra*.

The ’447 Patent	Infringed By
data persistently stored, wherein said data associates particular sources of code and particular keys with a set of one or more permissions.	

The ’447 Patent	Infringed By
19. A computer system comprising:	The Accused Instrumentalities include devices that run Android or the Android SDK. Devices running Android or the Android SDK are computer systems. *See* Claim 1, *supra*.
a processor;	Devices running Android and computers running the Android SDK have processors.
a memory coupled to said processor;	Devices running Android and computers running the Android SDK have a memory coupled to said processor.
one or more protection domains stored as objects in said memory, wherein each protection domain is associated with zero or more permissions;	*See* Claim 1 and 1-a, *supra*.
a domain mapping object stored in said memory, said domain mapping object establishing an association between said one or more protection domains and one or more classes of one or more objects; and	*See* Claim 1, 1-a, and 1-b, *supra*.
said processor being configured to determine whether an action requested by a particular object is permitted based on said association between said one or more protection domains and said one or more classes.	*See* Claim 1 and 1-c, *supra*.

The ’447 Patent	Infringed By
20. The computer system of claim 19, wherein:	*See* Claim 19, *supra*.
at least one protection domain of said one or more protection domains is associated with a code identifier;	*See* Claim 2, *supra*.
at least one class of said one or more classes is associated with said code identifier; and	*See* Claim 2, *supra*.
said computer system further comprises said processor configured to establish an association between said one or more protection domains and said one or more classes of one or more objects by associating said one or more protection domains and said one or more classes based on said code identifier.	*See* Claim 2, *supra*.

The ’447 Patent	Infringed By
21. The computer system of claim 20, wherein said code identifier indicates a source of code used to define each class of said one or more classes.	*See* Claims 2 and 20, *supra*.

The ’447 Patent	Infringed By
22. The computer system of claim 20, wherein said code identifier indicates a key associated with each class of said one or more classes.	*See* Claims 2, 4, and 20, *supra*.

The ’447 Patent	Infringed By
23. The computer system of claim 20, wherein said code identifier indicates a source of code used to define each class of said one or more classes and indicates a key associated with each class of said one or more classes.	*See* Claims 2, 4, and 20, *supra*.

The ’447 Patent	Infringed By
24. The computer system of claim 20, further comprising said processor configured to associate said one or more protection domains and said one or more classes based on said code identifier by associating said one or more protection domains and said one or more classes based on data persistently stored in said computer system, wherein said data associates code identifiers with a set of one or more permissions.	*See* Claims 2, 6, and 20, *supra*.

EXHIBIT E
Preliminary Infringement Contentions for the ’476 Patent

NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’476 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The ’476 Patent	Infringed By
1. A method for providing security, the method comprising the steps of:	Android includes methods for providing security. *See, e.g.:* • Android APIs for “java.security,” available at http://developer.android.com/reference/java/security/package-summary.html • Android Framework Topics for “Security and Permissions,” available at http://developer.android.com/guide/topics/security/security.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/permission-element.html • Android Framework Topics for “Security and Permissions” under “The AndroidManifest.xml File,” http://developer.android.com/guide/topics/manifest/application-element.html • Android Framework Topics for “The AndroidManifest.xml File,” available at

pa-1432801

The ’476 Patent	Infringed By
	http://developer.android.com/guide/topics/manifest/manifest-intro.html *See generally*, Policy.java, PolicyEntry.java, SecurityManager.java, AccessController.java, AccessControlContext.java, Permission.java, ProtectionDomain.java, Key.java, and CodeSource.java, as well as: http://developer.android.com/reference/java/security/Policy.html http://developer.android.com/reference/java/security/ProtectionDomain.html http://developer.android.com/reference/java/security/CodeSource.html http://developer.android.com/guide/developing/tools/othertools.html Android Developer Tools available at http://developer.android.com. In the Android source code, *see generally* PolicyEntry.java (dalvik\libcore\security\src\main\java\org\apache\harmony\security) and ProtectionDomain.java (dalvik\libcore\security\src\main\java\java\security). The class PolicyEntry associates data of executable code (i.e., CodeSource, including methods and routines), principals, and permissions. *See* PolicyEntry.java: /** * This class represents an elementary block of a security policy. It associates * a CodeSource of an executable code, Principals allowed to execute the code, * and a set of granted Permissions. * * @see org.apache.harmony.security.fortress.DefaultPolicy */ public class PolicyEntry { // Store CodeSource private final CodeSource cs; // Array of principals private final Principal[] principals;

The ’476 Patent	Infringed By
	// Permissions collection private final Collection<Permission> permissions; *See* dalvik\libcore\security\src\main\java\org\apache\harmony\security. *See generally, e.g.*: • dalvik\vm\native\InternalNative.c • dalvik\vm\native\java_security_AccessController.c • dalvik\vm\native\java_lang_VMClassLoader.c • source code files in libcore\security\src\main\java\java\security • source code files in libcore\security-kernel\src\main\java\java\security • libcore\security\src\main\java\org\apache\harmony\security
[1-a] detecting when a request for an action is made by a principal; and	Android detects when a request for an action is made by a principal. For example, the Policy class of Android is designed to implement a system security policy to detect whether principals have proper permissions to execute their requested actions. *See, e.g.*, Android Developer Tools available at http://developer.android.com/reference/java/security/Policy.html: **Class Overview** `Policy` is the common super type of classes which represent a system security policy. The `Policy` specifies which permissions apply to which code sources. Android Developer Tools available at http://developer.android.com/reference/java/security/AccessController.html: **Class Overview** `AccessController` provides static methods to perform access control checks and

The ’476 Patent	Infringed By
	privileged operations. Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html: **public AccessControlContext ([AccessControlContext]() acc, [DomainCombiner]() combiner)** *** If a `SecurityManager` is installed, code calling this constructor needs the `SecurityPermission` `createAccessControlContext` to be granted, otherwise a `SecurityException` will be thrown. *See also* PolicyEntry.java: /** * Constructor with initialization parameters. Passed collections are not * referenced directly, but copied. */ public PolicyEntry(CodeSource cs, Collection<? extends Principal> prs, Collection<? extends Permission> permissions) { this.cs = cs; this.principals = (prs == null || prs.isEmpty()) ? null : (Principal[]) prs.toArray(new Principal[prs.size()]); this.permissions = (permissions == null || permissions.isEmpty()) ? null : Collections.unmodifiableCollection(permissions); } /** * Checks if passed CodeSource matches this PolicyEntry. Null CodeSource of * PolicyEntry implies any CodeSource; non-null CodeSource forwards to its * imply() method.

The ’476 Patent	Infringed By
	`` */ public boolean impliesCodeSource(CodeSource codeSource) { return (cs == null) ? true : cs.implies(codeSource); } /** * Checks if specified Principals match this PolicyEntry. Null or empty set * of Principals of PolicyEntry implies any Principals; otherwise specified * array must contain all Principals of this PolicyEntry. */ public boolean impliesPrincipals(Principal[] prs) { return PolicyUtils.matchSubset(principals, prs); } /** * Returns unmodifiable collection of permissions defined by this * PolicyEntry, may be <code>null</code>. */ public Collection<Permission> getPermissions() { return permissions; } ``` *See* dalvik\libcore\security\src\main\java\org\apache\harmony\security. *See generally, e.g.*: • dalvik\vm\native\InternalNative.c • dalvik\vm\native\java_security_AccessController.c • dalvik\vm\native\java_lang_VMClassLoader.c • source code files in libcore\security\src\main\java\java\security • source code files in libcore\security-kernel\src\main\java\java\security • libcore\security\src\main\java\org\apache\harmony\security

The ’476 Patent	Infringed By
[1-b] in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal,	In response to detecting the request, Android determines whether the action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with the principal. *See, e.g.*, PolicyEntry.java: `/**` ` * Checks if passed CodeSource matches this PolicyEntry. Null CodeSource of` ` * PolicyEntry implies any CodeSource; non-null CodeSource forwards to its` ` * imply() method.` ` */` `public boolean impliesCodeSource(CodeSource codeSource) {` `    return (cs == null) ? true : cs.implies(codeSource);` `}` `/**` ` * Checks if specified Principals match this PolicyEntry. Null or empty set` ` * of Principals of PolicyEntry implies any Principals; otherwise specified` ` * array must contain all Principals of this PolicyEntry.` ` */` `public boolean impliesPrincipals(Principal[] prs) {` `    return PolicyUtils.matchSubset(principals, prs);` `}` `/**` ` * Returns unmodifiable collection of permissions defined by this` ` * PolicyEntry, may be <code>null</code>.` ` */` `public Collection<Permission> getPermissions() {` `    return permissions;` `    }`

The ’476 Patent	Infringed By
	See dalvik\libcore\security\src\main\java\org\apache\harmony\security. Regarding the “calling hierarchy associated with said principal,” *see, e.g.*: http://developer.android.com/reference/java/security/AccessController.html static AccessControlContext getContext() Returns the `AccessControlContext` for the current `Thread` including the inherited access control context of the thread that spawned the current thread (recursively). Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html *See also* , *e.g.*, java.security.AccessController: /* * java.security.AccessController */ #include "Dalvik.h" #include "native/InternalNativePriv.h" /* * private static ProtectionDomain[] getStackDomains() * * Return an array of ProtectionDomain objects from the classes of the * methods on the stack. Ignore reflection frames. Stop at the first * privileged frame we see. */ static void Dalvik_java_security_AccessController_getStackDomains(const u4* args, JValue* pResult) { UNUSED_PARAMETER(args);

The ’476 Patent	Infringed By
	const Method** methods = NULL; int length; /* * Get an array with the stack trace in it. */ if (!dvmCreateStackTraceArray(dvmThreadSelf()->curFrame, &methods, &length)) { LOGE("Failed to create stack trace array\n"); dvmThrowException("Ljava/lang/InternalError;", NULL); RETURN_VOID(); } //int i; //LOGI("dvmCreateStackTraceArray results:\n"); //for (i = 0; i < length; i++) // LOGI(" %2d: %s.%s\n", i, methods[i]->clazz->name, methods[i]->name); /* * Generate a list of ProtectionDomain objects from the frames that * we're interested in. Skip the first two methods (this method, and * the one that called us), and ignore reflection frames. Stop on the * frame *after* the first privileged frame we see as we walk up. * * We create a new array, probably over-allocated, and fill in the * stuff we want. We could also just run the list twice, but the * costs of the per-frame tests could be more expensive than the * second alloc. (We could also allocate it on the stack using C99 * array creation, but it's not guaranteed to fit.) * * The array we return doesn't include null ProtectionDomain objects,

The '476 Patent	Infringed By
	`` * so we skip those here. */ Object** subSet = (Object**) malloc((length-2) * sizeof(Object*)); if (subSet == NULL) { LOGE("Failed to allocate subSet (length=%d)\n", length); free(methods); dvmThrowException("Ljava/lang/InternalError;", NULL); RETURN_VOID(); } int idx, subIdx = 0; for (idx = 2; idx < length; idx++) { const Method* meth = methods[idx]; Object* pd; if (dvmIsReflectionMethod(meth)) continue; if (dvmIsPrivilegedMethod(meth)) { /* find nearest non-reflection frame; note we skip priv frame */ //LOGI("GSD priv frame at %s.%s\n", meth->clazz->name, meth->name); while (++idx < length && dvmIsReflectionMethod(methods[idx])) ; length = idx; // stomp length to end loop meth = methods[idx]; } /* get the pd object from the method's class */ assert(gDvm.offJavaLangClass_pd != 0); pd = dvmGetFieldObject((Object*) meth->clazz, gDvm.offJavaLangClass_pd); //LOGI("FOUND '%s' pd=%p\n", meth->clazz->name, pd); if (pd != NULL) ```

The ’476 Patent	Infringed By
	`` subSet[subIdx++] = pd; } //LOGI("subSet:\n"); //for (i = 0; i < subIdx; i++) // LOGI(" %2d: %s\n", i, subSet[i]->clazz->name); /* * Create an array object to contain "subSet". */ ClassObject* pdArrayClass = NULL; ArrayObject* domains = NULL; pdArrayClass = dvmFindArrayClass("[Ljava/security/ProtectionDomain;", NULL); if (pdArrayClass == NULL) { LOGW("Unable to find ProtectionDomain class for array\n"); goto bail; } domains = dvmAllocArray(pdArrayClass, subIdx, kObjectArrayRefWidth, ALLOC_DEFAULT); if (domains == NULL) { LOGW("Unable to allocate pd array (%d elems)\n", subIdx); goto bail; } /* copy the ProtectionDomain objects out */ Object** objects = (Object**) domains->contents; for (idx = 0; idx < subIdx; idx++) *objects++ = subSet[idx]; bail: free(subSet); free(methods); ```

The ’476 Patent	Infringed By
	`` dvmReleaseTrackedAlloc((Object*) domains, NULL); RETURN_PTR(domains); } const DalvikNativeMethod dvm_java_security_AccessController[] = { { "getStackDomains", "()[Ljava/security/ProtectionDomain;", Dalvik_java_security_AccessController_getStackDomains }, { NULL, NULL, NULL }, }; ``` dalvik\vm\native\java_security_AccessController.c. *See also, e.g.:* dalvik\vm\native\java_lang_VMClassLoader.c: ``` /* * java.lang.VMClassLoader */ … /* * static Class defineClass(ClassLoader cl, String name, * byte[] data, int offset, int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. */ static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; StringObject* nameObj = (StringObject*) args[1]; const u1* data = (const u1*) args[2]; int offset = args[3]; ```

The ’476 Patent	Infringed By
	<pre> int len = args[4]; Object* pd = (Object*) args[5]; char* name = NULL; name = dvmCreateCstrFromString(nameObj); LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n", loader, name, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); free(name); RETURN_VOID(); } /* * static Class defineClass(ClassLoader cl, byte[] data, int offset, * int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. Deprecated version of * previous method, lacks name parameter. */ static void Dalvik_java_lang_VMClassLoader_defineClass2(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; const u1* data = (const u1*) args[1]; int offset = args[2]; int len = args[3]; Object* pd = (Object*) args[4]; LOGE("ERROR: defineClass(%p, %p, %d, %d, %p)\n",</pre>

The ’476 Patent	Infringed By
	loader, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); RETURN_VOID(); } libcore\luni\src\main\java\java\lang\SecurityManager.java: /** * Warning: security managers do not provide a * secure environment for executing untrusted code. Untrusted code cannot be * safely isolated within the Dalvik VM. * * <p>Provides security verification facilities for applications. {@code * SecurityManager} contains a set of {@code checkXXX} methods which determine * if it is safe to perform a specific operation such as establishing network * connections, modifying files, and many more. In general, these methods simply * return if they allow the application to perform the operation; if an * operation is not allowed, then they throw a {@link SecurityException}. The * only exception is {@link #checkTopLevelWindow(Object)}, which returns a * boolean to indicate permission. */ public class SecurityManager { … /** * Checks whether the calling thread is allowed to access the resource being * guarded by the specified permission object. * * @param permission * the permission to check. * @throws SecurityException * if the requested {@code permission} is denied according to

The ’476 Patent	Infringed By
	`` * the current security policy. */ public void checkPermission(Permission permission) { try { inCheck = true; AccessController.checkPermission(permission); } finally { inCheck = false; } } /** * Checks whether the specified security context is allowed to access the * resource being guarded by the specified permission object. * * @param permission * the permission to check. * @param context * the security context for which to check permission. * @throws SecurityException * if {@code context} is not an instance of {@code * AccessControlContext} or if the requested {@code permission} * is denied for {@code context} according to the current * security policy. */ public void checkPermission(Permission permission, Object context) { try { inCheck = true; // Must be an AccessControlContext. If we don't check // this, then applications could pass in an arbitrary // object which circumvents the security check. if (context instanceof AccessControlContext) { ```

The ’476 Patent	Infringed By
	((AccessControlContext) context).checkPermission(permission); } else { throw new SecurityException(); } } finally { inCheck = false; } } } libcore\security-kernel\src\main\java\java\security\AccessController.java: /** * Checks the specified permission against the vm's current security policy. * The check is performed in the context of the current thread. This method * returns silently if the permission is granted, otherwise an {@code * AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * the current execution context has been granted the specified permission. * If privileged operations are on the execution context, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * This method delegates the permission check to * {@link AccessControlContext#checkPermission(Permission)} on the current * callers' context obtained by {@link #getContext()}. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException

The ’476 Patent	Infringed By
	`` * if the specified permission is {@code null} * @see AccessControlContext#checkPermission(Permission) * * @since Android 1.0 */ public static void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("permission can not be null"); } getContext().checkPermission(perm); } ``` libcore\security-kernel\src\main\java\java\security\AccessController.java: ``` /** * Checks the specified permission against the vm's current security policy. * The check is performed in the context of the current thread. This method * returns silently if the permission is granted, otherwise an {@code * AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * the current execution context has been granted the specified permission. * If privileged operations are on the execution context, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * This method delegates the permission check to * {@link AccessControlContext#checkPermission(Permission)} on the current * callers' context obtained by {@link #getContext()}. * * @param perm ```

The ’476 Patent	Infringed By
	` * the permission to check against the policy` ` * @throws AccessControlException` ` * if the specified permission is not granted` ` * @throws NullPointerException` ` * if the specified permission is {@code null}` ` * @see AccessControlContext#checkPermission(Permission)` ` *` ` * @since Android 1.0` ` */` `public static void checkPermission(Permission perm)` `        throws AccessControlException {` `    if (perm == null) {` `        throw new NullPointerException("permission can not be null");` `    }` `    getContext().checkPermission(perm);` `}` libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: `ProtectionDomain[] context;` `…` `/**` ` * Checks the specified permission against the vm's current security policy.` ` * The check is based on this {@code AccessControlContext} as opposed to the` ` * {@link AccessController#checkPermission(Permission)} method which` ` * performs access checks based on the context of the current thread. This` ` * method returns silently if the permission is granted, otherwise an` ` * {@code AccessControlException} is thrown.` ` * <p>` ` * A permission is considered granted if every {@link ProtectionDomain} in` ` * this context has been granted the specified permission.` ` * <p>`

The ’476 Patent	Infringed By
	<pre>* If privileged operations are on the call stack, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * If inherited methods are on the call stack, the protection domains of the * declaring classes are checked, not the protection domains of the classes * on which the method is invoked. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessController#checkPermission(Permission) * @since Android 1.0 */ public void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("Permission cannot be null"); } for (int i = 0; i < context.length; i++) { if (!context[i].implies(perm)) { throw new AccessControlException("Permission check failed " + perm, perm); } } if (inherited != null) { inherited.checkPermission(perm); } }</pre>

The ’476 Patent	Infringed By
	See also, e.g: libcore-disabled\sound\src\main\java\org\apache\harmony\sound\utils\ProviderService.java; dalvik\tests\025-access-controller\expected.txt; dalvik\tests\025-access-controller\src\Main.java; dalvik\tests\025-access-controller\src\Privvy.java; libcore\security\src\test\java\tests\security.
[1-c] wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.	The permissions are associated with the routines based on an association between protection domains and permissions. *See* Claim 1-b, *supra*. *See* libcore\security\src\main\java\java\security\ProtectionDomain.java: * {@code ProtectionDomain} represents all permissions that are granted to a * specific code source. The {@link ClassLoader} associates each class with the * corresponding {@code ProtectionDomain}, depending on the location and the * certificates (encapsulates in {@link CodeSource}) it loads the code from. * <p> * A class belongs to exactly one protection domain and the protection domain * can not be changed during the lifetime of the class. */ public class ProtectionDomain { // CodeSource for this ProtectionDomain private CodeSource codeSource; // Static permissions for this ProtectionDomain private PermissionCollection permissions;

The ’476 Patent	Infringed By
	// ClassLoader private ClassLoader classLoader; // Set of principals associated with this ProtectionDomain private Principal[] principals; // false if this ProtectionDomain was constructed with static // permissions, true otherwise. private boolean dynamicPerms;

The ’476 Patent	Infringed By
2. The method of claim 1, wherein:	*See* Claim 1, *supra*.
the step of detecting when a request for an action is made includes detecting when a request for an action is made by a thread; and	Android includes java.security, which includes the AccessController class that “perform[s] access control checks and privileged operations.” *See* Android Developer Tools available at http://developer.android.com/reference/java/security/AccessController.html: public static void checkPermission (Permission perm) Checks the specified permission against the vm's current security policy. The check is performed in the context of the current thread. This method returns silently if the permission is granted, otherwise an AccessControlException is thrown. A permission is considered granted if every ProtectionDomain in the current execution context has been granted the specified permission. If privileged operations are on the execution context, only the ProtectionDomains from the last privileged operation are taken into account. This method delegates the permission check to checkPermission(Permission) on the current callers' context obtained by getContext().

The '476 Patent	Infringed By
	Parameters perm the permission to check against the policy Throws AccessControlException if the specified permission is not granted NullPointerException if the specified permission is null See Also checkPermission(Permission) *See* libcore\luni\src\main\java\java\lang\ SecurityManager.java: /** * Warning: security managers do not provide a * secure environment for executing untrusted code. Untrusted code cannot be * safely isolated within the Dalvik VM. * * <p>Provides security verification facilities for applications. {@code * SecurityManager} contains a set of {@code checkXXX} methods which determine * if it is safe to perform a specific operation such as establishing network * connections, modifying files, and many more. In general, these methods simply * return if they allow the application to perform the operation; if an * operation is not allowed, then they throw a {@link SecurityException}. The * only exception is {@link #checkTopLevelWindow(Object)}, which returns a * boolean to indicate permission. */ public class SecurityManager { … /** * Checks whether the calling thread is allowed to access the resource being * guarded by the specified permission object.

The ’476 Patent	Infringed By
	<pre> * * @param permission * the permission to check. * @throws SecurityException * if the requested {@code permission} is denied according to * the current security policy. */ public void checkPermission(Permission permission) { try { inCheck = true; AccessController.checkPermission(permission); } finally { inCheck = false; } }</pre> *See also* libcore\security-kernel\src\main\java\java\security\AccessController.java: <pre> /** * The real implementation of doPrivileged() method. It pushes the passed * context into this thread's contexts stack, and then invokes * <code>action.run()</code>. The pushed context is then investigated in the * {@link #getContext()} which is called in the {@link #checkPermission}. */ private static <T> T doPrivilegedImpl(PrivilegedExceptionAction<T> action, AccessControlContext context) throws PrivilegedActionException { Thread currThread = Thread.currentThread(); ArrayList<AccessControlContext> a = null; try { // currThread==null means that VM warm up is in progress</pre>

pa-1432801

The '476 Patent	Infringed By
	<pre> if (currThread != null && contexts != null) { synchronized (contexts) { a = contexts.get(currThread); if (a == null) { a = new ArrayList<AccessControlContext>(); contexts.put(currThread, a); } } a.add(context); } return action.run(); } catch (Exception ex) { // Errors automagically go through - they are not catched by this // block // Unchecked exceptions must pass through without modification if (ex instanceof RuntimeException) { throw (RuntimeException) ex; } // All other (==checked) exceptions get wrapped throw new PrivilegedActionException(ex); } finally { if (currThread != null) { // No need to sync() here, as each given 'a' will be accessed // only from one Thread. 'contexts' still need sync() however, // as it's accessed from different threads simultaneously if (a != null) { // it seems I will never have here [v.size() == 0] a.remove(a.size() - 1);</pre>

The ’476 Patent	Infringed By
	} } } }
the step of determining whether said action is authorized includes determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said thread.	*See* Claim 1-b, *supra*. getContext() returns AccessControlContext, which “encapsulates the ProtectionDomains on which access control decisions are based.” *See* Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html. public void checkPermission (Permission perm) Checks the specified permission against the vm's current security policy. The check is based on this AccessControlContext as opposed to the checkPermission(Permission) method which performs access checks based on the context of the current thread. This method returns silently if the permission is granted, otherwise an AccessControlException is thrown. A permission is considered granted if every ProtectionDomain in this context has been granted the specified permission. If privileged operations are on the call stack, only the ProtectionDomains from the last privileged operation are taken into account. If inherited methods are on the call stack, the protection domains of the declaring classes are checked, not the protection domains of the classes on which the method is invoked. Parameters perm the permission to check against the policy Throws AccessControlException if the specified permission is not granted NullPointerException if the specified permission is null

The ’476 Patent	Infringed By
	See Also checkPermission(Permission) Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html. *See also*: public void checkPermission (Permission permission) Checks whether the calling thread is allowed to access the resource being guarded by the specified permission object. Parameters permission the permission to check. Throws SecurityException if the requested permission is denied according to the current security policy. public void checkPermission (Permission permission, Object context) Checks whether the specified security context is allowed to access the resource being guarded by the specified permission object. Parameters permission the permission to check. context the security context for which to check permission. Throws SecurityException if context is not an instance of AccessControlContext or if the requested permission is denied for context according to the current security policy. *See also* Android Developer Tools available at

The ’476 Patent	Infringed By
	http://developer.android.com/reference/java/lang/SecurityManager.html#checkPermission(java.security.Permission) *See also* libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: // An AccessControlContext inherited by the current thread from its parent private AccessControlContext inherited; …. /** * Package-level ctor which is used in AccessController. * ProtectionDomains passed as <code>stack</code> is then passed into * {@link #AccessControlContext(ProtectionDomain[])}, therefore: * <il> * it must not be null * duplicates will be removed * null-s will be removed * * * @param stack - array of ProtectionDomains * @param inherited - inherited context, which may be null */ AccessControlContext(ProtectionDomain[] stack, AccessControlContext inherited) { this(stack); // removes dups, removes nulls, checks for stack==null this.inherited = inherited; } … /** * Checks the specified permission against the vm's current security policy. * The check is based on this {@code AccessControlContext} as opposed to the * {@link AccessController#checkPermission(Permission)} method which * performs access checks based on the context of the current thread. This * method returns silently if the permission is granted, otherwise an

The ’476 Patent	Infringed By
	`` * {@code AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * this context has been granted the specified permission. * <p> * If privileged operations are on the call stack, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * If inherited methods are on the call stack, the protection domains of the * declaring classes are checked, not the protection domains of the classes * on which the method is invoked. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessController#checkPermission(Permission) * @since Android 1.0 */ public void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("Permission cannot be null"); } for (int i = 0; i < context.length; i++) { if (!context[i].implies(perm)) { throw new AccessControlException("Permission check failed " + perm, perm); } ```

The ’476 Patent	Infringed By
	} if (inherited != null) { inherited.checkPermission(perm); } }

The ’476 Patent	Infringed By
3. The method of claim 1, wherein:	*See* Claim 1, *supra*.
the calling hierarchy includes a first routine; and	*See* Claim 1-b, *supra*. Android calls the hierarchy, which includes a first routine, e.g., thread. *See, e.g.*: http://developer.android.com/reference/java/security/AccessController.html static AccessControlContext getContext() Returns the `AccessControlContext` for the current `Thread` including the inherited access control context of the thread that spawned the current thread (recursively). Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html. *See also* libcore\security-kernel\src\main\java\java\security\AccessController.java: /** * Returns the {@code AccessControlContext} for the current {@code Thread} * including the inherited access control context of the thread that spawned * the current thread (recursively). * <p> * The returned context may be used to perform access checks at a later * point in time, possibly by another thread.

The ’476 Patent	Infringed By
	`` * * @return the {@code AccessControlContext} for the current {@code Thread} * @see Thread#currentThread * @since Android 1.0 */ public static AccessControlContext getContext() { // duplicates (if any) will be removed in ACC constructor ProtectionDomain[] stack = getStackDomains(); Thread currThread = Thread.currentThread(); if (currThread == null || contexts == null) { // Big boo time. No need to check anything ? return new AccessControlContext(stack); } ArrayList<AccessControlContext> threadContexts; synchronized (contexts) { threadContexts = contexts.get(currThread); } AccessControlContext that; if ((threadContexts == null) || (threadContexts.size() == 0)) { // We were not in doPrivileged method, so // have inherited context here that = SecurityUtils.getContext(currThread); } else { // We were in doPrivileged method, so // Use context passed to the doPrivileged() that = threadContexts.get(threadContexts.size() - 1); } ```

pa-1432801

The ’476 Patent	Infringed By
	`if (that != null && that.combiner != null) {` `    ProtectionDomain[] assigned = null;` `    if (that.context != null && that.context.length != 0) {` `        assigned = new ProtectionDomain[that.context.length];` `        System.arraycopy(that.context, 0, assigned, 0, assigned.length);` `    }` `    ProtectionDomain[] allpds = that.combiner.combine(stack, assigned);` `    if (allpds == null) {` `        allpds = new ProtectionDomain[0];` `    }` `    return new AccessControlContext(allpds, that.combiner);` `}` `return new AccessControlContext(stack, that);` `}` `}` *See also*, *e.g.*, dalvik\vm\native\java_security_AccessController.c.
the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.	Android further includes determining whether a permission is encompassed by at least one permission associate the first routine. *See* Claim 1-b, *supra*. *See*, *e.g.*, Android Developer Tools available at: http://developer.android.com/reference/java/security/AccessController.html: **public static T doPrivileged (PrivilegedAction<T> action, AccessControlContext context)** Returns the result of executing the specified privileged action. The `ProtectionDomain`

pa-1432801

The ’476 Patent	Infringed By
	of the direct caller of this method, the `ProtectionDomains` of all subsequent classes in the call chain and all `ProtectionDomains` of the given context are checked to be granted the necessary permission if access checks are performed. If an instance of `RuntimeException` is thrown during the execution of the `PrivilegedAction#run()` method of the given action, it will be propagated through this method. **Parameters** **action** the action to be executed with privileges **context** the `AccessControlContext` whose protection domains are checked additionally **Returns** the result of executing the privileged action *See* libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: // An AccessControlContext inherited by the current thread from its parent private AccessControlContext inherited; …. /** * Package-level ctor which is used in AccessController. * ProtectionDomains passed as <code>stack</code> is then passed into * {@link #AccessControlContext(ProtectionDomain[])}, therefore: * <il> * it must not be null * duplicates will be removed * null-s will be removed * * * @param stack - array of ProtectionDomains

The ’476 Patent	Infringed By
	`` * @param inherited - inherited context, which may be null */ AccessControlContext(ProtectionDomain[] stack, AccessControlContext inherited) { this(stack); // removes dups, removes nulls, checks for stack==null this.inherited = inherited; } … /** * Checks the specified permission against the vm's current security policy. * The check is based on this {@code AccessControlContext} as opposed to the * {@link AccessController#checkPermission(Permission)} method which * performs access checks based on the context of the current thread. This * method returns silently if the permission is granted, otherwise an * {@code AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * this context has been granted the specified permission. * <p> * If privileged operations are on the call stack, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * If inherited methods are on the call stack, the protection domains of the * declaring classes are checked, not the protection domains of the classes * on which the method is invoked. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException ```

pa-1432801

The ’476 Patent	Infringed By
	`*            if the specified permission is {@code null}` `* @see AccessController#checkPermission(Permission)` `* @since Android 1.0` `*/` `public void checkPermission(Permission perm) throws AccessControlException {` `    if (perm == null) {` `        throw new NullPointerException("Permission cannot be null");` `    }` `    for (int i = 0; i < context.length; i++) {` `        if (!context[i].implies(perm)) {` `            throw new AccessControlException("Permission check failed "` `                    + perm, perm);` `        }` `    }` `    if (inherited != null) {` `        inherited.checkPermission(perm);` `    }` `}` libcore\security\src\main\java\java\security.

The ’476 Patent	Infringed By
4. The method of claim 1, wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.	Android further determines whether a permission required to perform the action is encompassed by a permission associated with each routine in the calling hierarchy. *See* Claim 1, *supra*. *See* Claim 1-b, *supra*. *See*, *e.g.*, Android Developer Tools available at http://developer.android.com/reference/java/security/AccessController.html:

The ’476 Patent	Infringed By
	public static T doPrivileged (PrivilegedAction<T> action, AccessControlContext context) Returns the result of executing the specified privileged action. The `ProtectionDomain` of the direct caller of this method, the `ProtectionDomains` of all subsequent classes in the call chain and all `ProtectionDomains` of the given context are checked to be granted the necessary permission if access checks are performed. If an instance of `RuntimeException` is thrown during the execution of the `PrivilegedAction#run()` method of the given action, it will be propagated through this method. **Parameters** **action** the action to be executed with privileges **context** the `AccessControlContext` whose protection domains are checked additionally **Returns** the result of executing the privileged action (highlighting added). *See also* libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: // An AccessControlContext inherited by the current thread from its parent private AccessControlContext inherited; …. /** * Package-level ctor which is used in AccessController. * ProtectionDomains passed as <code>stack</code> is then passed into * {@link #AccessControlContext(ProtectionDomain[])}, therefore:

The ’476 Patent	Infringed By
	` * <il>` ` * <li>it must not be null` ` * <li>duplicates will be removed` ` * <li>null-s will be removed` ` * </li>` ` *` ` * @param stack - array of ProtectionDomains` ` * @param inherited - inherited context, which may be null` ` */` `AccessControlContext(ProtectionDomain[] stack,` `        AccessControlContext inherited) {` `    this(stack); // removes dups, removes nulls, checks for stack==null` `    this.inherited = inherited;` `}` `…` `/**` ` * Checks the specified permission against the vm's current security policy.` ` * The check is based on this {@code AccessControlContext} as opposed to the` ` * {@link AccessController#checkPermission(Permission)} method which` ` * performs access checks based on the context of the current thread. This` ` * method returns silently if the permission is granted, otherwise an` ` * {@code AccessControlException} is thrown.` ` * <p>` ` * A permission is considered granted if every {@link ProtectionDomain} in` ` * this context has been granted the specified permission.` ` * <p>` ` * If privileged operations are on the call stack, only the {@code` ` * ProtectionDomain}s from the last privileged operation are taken into` ` * account.` ` * <p>` ` * If inherited methods are on the call stack, the protection domains of the` ` * declaring classes are checked, not the protection domains of the classes`

<table>
<tr><th>The ’476 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>* on which the method is invoked.
*
* @param perm
* the permission to check against the policy
* @throws AccessControlException
* if the specified permission is not granted
* @throws NullPointerException
* if the specified permission is {@code null}
* @see AccessController#checkPermission(Permission)
* @since Android 1.0
*/
public void checkPermission(Permission perm) throws AccessControlException {
if (perm == null) {
throw new NullPointerException("Permission cannot be null");
}
for (int i = 0; i < context.length; i++) {
if (!context[i].implies(perm)) {
throw new AccessControlException("Permission check failed "
+ perm, perm);
}
}
if (inherited != null) {
inherited.checkPermission(perm);
}
}
libcore\security\src\main\java\java\security.</td></tr>
</table>

The ’476 Patent	Infringed By
5. A method for providing security, the method comprising the steps of:	See corresponding elements of claim 1, *supra*.
detecting when a request for an	See corresponding elements of claim 1, *supra*.

The ’476 Patent	Infringed By
action is made by a principal,	
determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal;	See corresponding elements of claim 1, *supra*.
wherein each routine of said plurality of routines is associated with a class; and	Each routine of the plurality of routines in the calling hierarchy in Android is associated with a class of one or more objects. *See e.g.:* dalvik\vm\native\java_lang_VMClassLoader.c: /* * java.lang.VMClassLoader */ … /* * static Class defineClass(ClassLoader cl, String name, * byte[] data, int offset, int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. */ static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; StringObject* nameObj = (StringObject*) args[1]; const u1* data = (const u1*) args[2]; int offset = args[3]; int len = args[4]; Object* pd = (Object*) args[5]; char* name = NULL;

The ’476 Patent	Infringed By
	`` name = dvmCreateCstrFromString(nameObj); LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n", loader, name, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); free(name); RETURN_VOID(); } /* * static Class defineClass(ClassLoader cl, byte[] data, int offset, * int len, ProtectionDomain pd) * throws ClassFormatError * * Convert an array of bytes to a Class object. Deprecated version of * previous method, lacks name parameter. */ static void Dalvik_java_lang_VMClassLoader_defineClass2(const u4* args, JValue* pResult) { Object* loader = (Object*) args[0]; const u1* data = (const u1*) args[1]; int offset = args[2]; int len = args[3]; Object* pd = (Object*) args[4]; LOGE("ERROR: defineClass(%p, %p, %d, %d, %p)\n", loader, data, offset, len, pd); dvmThrowException("Ljava/lang/UnsupportedOperationException;", "can't load this type of class file"); ```

The ’476 Patent	Infringed By
	RETURN_VOID(); } *See, e.g.*, PolicyEntry.java: /** * This class represents an elementary block of a security policy. It associates * a CodeSource of an executable code, Principals allowed to execute the code, * and a set of granted Permissions. * * @see org.apache.harmony.security.fortress.DefaultPolicy */ *See generally, e.g.*: • dalvik\vm\native\InternalNative.c • dalvik\vm\native\java_security_AccessController.c • dalvik\vm\native\java_lang_VMClassLoader.c • source code files in libcore\security\src\main\java\java\security • source code files in libcore\security-kernel\src\main\java\java\security • libcore\security\src\main\java\org\apache\harmony\security
wherein said association between permissions and said plurality of routines is based on a second association between classes and protection domains.	The association between permissions and routines in Android is based on an association between classes and protection domains. *See* Claim 1-b, *supra*. *See* Claim 1-c, *supra*. Android associates classes and protection domains when loading classes. *See, e.g.*, ProtectionDomain.java: * {@code ProtectionDomain} represents all permissions that are granted to a * specific code source. The {@link ClassLoader} associates each class with the * corresponding {@code ProtectionDomain}, depending on the location and the

The ’476 Patent	Infringed By
	* certificates (encapsulates in {@link CodeSource}) it loads the code from. * * A class belongs to exactly one protection domain and the protection domain * can not be changed during the lifetime of the class. * * * @since Android 1.0 */ public class ProtectionDomain { // CodeSource for this ProtectionDomain private CodeSource codeSource; // Static permissions for this ProtectionDomain private PermissionCollection permissions; // ClassLoader private ClassLoader classLoader; // Set of principals associated with this ProtectionDomain private Principal[] principals; // false if this ProtectionDomain was constructed with static // permissions, true otherwise. private boolean dynamicPerms; *See* libcore\security\src\main\java\java\security\ProtectionDomain.java. *See also* PolicyEntry.java: /** * This class represents an elementary block of a security policy. It associates

The ’476 Patent	Infringed By
	* a CodeSource of an executable code, Principals allowed to execute the code, * and a set of granted Permissions. * * @see org.apache.harmony.security.fortress.DefaultPolicy */ public class PolicyEntry { // Store CodeSource private final CodeSource cs; // Array of principals private final Principal[] principals; // Permissions collection private final Collection<Permission> permissions; /** * Returns unmodifiable collection of permissions defined by this * PolicyEntry, may be <code>null</code>. */ public Collection<Permission> getPermissions() { return permissions; } *See* dalvik\libcore\security\src\main\java\org\apache\harmony\security.

The ’476 Patent	Infringed By
6. A method for providing security, the method comprising the steps of:	See corresponding elements of claim 1, *supra*.
detecting when a request for an	See corresponding elements of claim 1, *supra*.

The ’476 Patent	Infringed By
action is made by a principal; and	
in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal,	See corresponding elements of claim 1, *supra*.
wherein a first routine in said calling hierarchy is privileged; and	A routine in the calling hierarchy may be privileged. *See, e.g.*, Android Developer Tools available at http://developer.android.com/reference/java/security/PrivilegedAction.html: **Class Overview** `PrivilegedAction` represents an action that can be executed privileged regarding access control. Instances of `PrivilegedAction` can be executed on `AccessController.doPrivileged()`. *See also* http://developer.android.com/reference/java/security/AccessController.html: **public static T doPrivileged (PrivilegedAction<T> action, AccessControlContext context)** Returns the result of executing the specified privileged action. The `ProtectionDomain` of the direct caller of this method, the `ProtectionDomains` of all subsequent classes in the call chain and all `ProtectionDomains` of the given context are checked to be granted the necessary permission if access checks are performed. (highlighting added).
wherein the step of determining whether said action is authorized further includes determining	Android further determines whether a permission required to perform the action is encompassed by a permission associated with each routine in the calling hierarchy between and including a first routine and a second routine in said calling hierarchy, wherein said second routine is

The ’476 Patent	Infringed By
whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and a second routine in said calling hierarchy, wherein said second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.	invoked after said first routine. *See* Claim 1-b, *supra*. *See*, *e.g.*, Android Developer Tools available at http://developer.android.com/reference/java/security/AccessController.html: **public static T doPrivileged (PrivilegedAction<T> action, AccessControlContext context)** Returns the result of executing the specified privileged action. The `ProtectionDomain` of the direct caller of this method, the `ProtectionDomains` of all subsequent classes in the call chain and all `ProtectionDomains` of the given context are checked to be granted the necessary permission if access checks are performed. If an instance of `RuntimeException` is thrown during the execution of the `PrivilegedAction#run()` method of the given action, it will be propagated through this method. **Parameters** **action** the action to be executed with privileges **context** the `AccessControlContext` whose protection domains are checked additionally **Returns** **the result of executing the privileged action** (highlighting added). *See also* libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: // An AccessControlContext inherited by the current thread from its parent

The ’476 Patent	Infringed By
	`` private AccessControlContext inherited; … /** * Package-level ctor which is used in AccessController. * ProtectionDomains passed as <code>stack</code> is then passed into * {@link #AccessControlContext(ProtectionDomain[])}, therefore: * <il> * it must not be null * duplicates will be removed * null-s will be removed * * * @param stack - array of ProtectionDomains * @param inherited - inherited context, which may be null */ AccessControlContext(ProtectionDomain[] stack, AccessControlContext inherited) { this(stack); // removes dups, removes nulls, checks for stack==null this.inherited = inherited; } … /** * Checks the specified permission against the vm's current security policy. * The check is based on this {@code AccessControlContext} as opposed to the * {@link AccessController#checkPermission(Permission)} method which * performs access checks based on the context of the current thread. This * method returns silently if the permission is granted, otherwise an * {@code AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * this context has been granted the specified permission. * <p> ```

The ’476 Patent	Infringed By
	` * If privileged operations are on the call stack, only the {@code` ` * ProtectionDomain}s from the last privileged operation are taken into` ` * account.` ` * <p>` ` * If inherited methods are on the call stack, the protection domains of the` ` * declaring classes are checked, not the protection domains of the classes` ` * on which the method is invoked.` ` *` ` * @param perm` ` *            the permission to check against the policy` ` * @throws AccessControlException` ` *            if the specified permission is not granted` ` * @throws NullPointerException` ` *            if the specified permission is {@code null}` ` * @see AccessController#checkPermission(Permission)` ` * @since Android 1.0` ` */` `public void checkPermission(Permission perm) throws AccessControlException {` `    if (perm == null) {` `        throw new NullPointerException("Permission cannot be null");` `    }` `    for (int i = 0; i < context.length; i++) {` `        if (!context[i].implies(perm)) {` `            throw new AccessControlException("Permission check failed "` `                    + perm, perm);` `        }` `    }` `    if (inherited != null) {` `        inherited.checkPermission(perm);` `    }` `}`

The ’476 Patent	Infringed By
7. The method of claim 6, wherein the step of determining whether said permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and said second routine further includes the steps of:	*See* claim 2 and 6, *supra*.
determining whether said permission required is encompassed by at least one permission associated with said second routine; and	See claim 3, *supra*.
in response to determining said permission required is encompassed by at least one permission associated with said second routine, then performing the steps of:	See claim 4, *supra*.
A) selecting a next routine from said plurality of routines in said calling hierarchy,	Android selects a next routine from a plurality of routines in a calling hierarchy. *See, e.g.*, libcore\security-kernel\src\main\java\java\security\AccessController.java: * Checks the specified permission against the vm's current security policy. * The check is performed in the context of the current thread. This method * returns silently if the permission is granted, otherwise an {@code * AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * the current execution context has been granted the specified permission. * If privileged operations are on the execution context, only the {@code * ProtectionDomain}s from the last privileged operation are taken into

The ’476 Patent	Infringed By
	* account. * <p> * This method delegates the permission check to * {@link AccessControlContext#checkPermission(Permission)} on the current * callers' context obtained by {@link #getContext()}. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessControlContext#checkPermission(Permission) * * @since Android 1.0 */ public static void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("permission can not be null"); } getContext().checkPermission(perm); }
B) if said permission required is not encompassed by at least one permission associated with said next routine, then transmitting a message indicating that said permission required is not authorized, and	Android checks whether the permission required is or is not encompassed by a permission associated with the next routine and transmits a message if not encompassed by the permission associated with the next routine. *See, e.g.*, libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: /** * Checks the specified permission against the vm's current security policy. * The check is based on this {@code AccessControlContext} as opposed to the

The ’476 Patent	Infringed By
	* {@link AccessController#checkPermission(Permission)} method which * performs access checks based on the context of the current thread. This * method returns silently if the permission is granted, otherwise an * {@code AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * this context has been granted the specified permission. * <p> * If privileged operations are on the call stack, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * If inherited methods are on the call stack, the protection domains of the * declaring classes are checked, not the protection domains of the classes * on which the method is invoked. * * @param perm * the permission to check against the policy * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessController#checkPermission(Permission) * @since Android 1.0 */ public void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("Permission cannot be null"); } for (int i = 0; i < context.length; i++) { if (!context[i].implies(perm)) { throw new AccessControlException("Permission check failed "

The ’476 Patent	Infringed By
	`` + perm, perm); } } if (inherited != null) { inherited.checkPermission(perm); } } ```
C) repeating steps A and B until: said permission required is not authorized by at least one permission associated with said next routine, there are no more routines to select from said plurality of routines in said calling hierarchy, or determining that said next routine is said first routine.	Android repeats steps A and B detailed above until the permission required is not authorized or there are no more routines. *See, e.g.*, libcore\security-kernel\src\main\java\java\security\AccessControlContext.java: ``` /** * Checks the specified permission against the vm's current security policy. * The check is based on this {@code AccessControlContext} as opposed to the * {@link AccessController#checkPermission(Permission)} method which * performs access checks based on the context of the current thread. This * method returns silently if the permission is granted, otherwise an * {@code AccessControlException} is thrown. * <p> * A permission is considered granted if every {@link ProtectionDomain} in * this context has been granted the specified permission. * <p> * If privileged operations are on the call stack, only the {@code * ProtectionDomain}s from the last privileged operation are taken into * account. * <p> * If inherited methods are on the call stack, the protection domains of the * declaring classes are checked, not the protection domains of the classes * on which the method is invoked. * * @param perm * the permission to check against the policy ```

The ’476 Patent	Infringed By
	`` * @throws AccessControlException * if the specified permission is not granted * @throws NullPointerException * if the specified permission is {@code null} * @see AccessController#checkPermission(Permission) * @since Android 1.0 */ public void checkPermission(Permission perm) throws AccessControlException { if (perm == null) { throw new NullPointerException("Permission cannot be null"); } for (int i = 0; i < context.length; i++) { if (!context[i].implies(perm)) { throw new AccessControlException("Permission check failed " + perm, perm); } } if (inherited != null) { inherited.checkPermission(perm); } } ```

The ’476 Patent	Infringed By
8. The method of claim 7, wherein:	*See* claim 7, *supra.*
the method further includes the step of setting a flag associated with said first routine to indicate that said first routine is	The process includes setting a flag associated with the first routine that indicates that the first routine is privileged, where the flag value is a “true” or “false” return value. *See, e.g.:* platform/dalvik.git/vm/native/java_security_AccessController.c: /*

The ’476 Patent	Infringed By
privileged; and	` * Generate a list of ProtectionDomain objects from the frames that` ` * we're interested in.  Skip the first two methods (this method, and` ` * the one that called us), and ignore reflection frames.  Stop on the` ` * frame *after* the first privileged frame we see as we walk up.` ` *` ` * We create a new array, probably over-allocated, and fill in the` ` * stuff we want.  We could also just run the list twice, but the` ` * costs of the per-frame tests could be more expensive than the` ` * second alloc.  (We could also allocate it on the stack using C99` ` * array creation, but it's not guaranteed to fit.)` ` *` ` * The array we return doesn't include null ProtectionDomain objects,` ` * so we skip those here.` ` */` `Object** subSet = (Object**) malloc((length-2) * sizeof(Object*));` `if (subSet == NULL) {` `    LOGE("Failed to allocate subSet (length=%d)\n", length);` `    free(methods);` `    dvmThrowException("Ljava/lang/InternalError;", NULL);` `    RETURN_VOID();` `}` `int idx, subIdx = 0;` `for (idx = 2; idx < length; idx++) {` `    const Method* meth = methods[idx];` `    Object* pd;` `    if (dvmIsReflectionMethod(meth))` `        continue;` `    if (dvmIsPrivilegedMethod(meth)) {` `        /* find nearest non-reflection frame; note we skip priv frame */` `        //LOGI("GSD priv frame at %s.%s\n", meth->clazz->name, meth->name);`

The ’476 Patent	Infringed By
	<pre> while (++idx < length && dvmIsReflectionMethod(methods[idx])) ; length = idx; // stomp length to end loop meth = methods[idx]; } /* get the pd object from the method's class */ assert(gDvm.offJavaLangClass_pd != 0); pd = dvmGetFieldObject((Object*) meth->clazz, gDvm.offJavaLangClass_pd); //LOGI("FOUND '%s' pd=%p\n", meth->clazz->name, pd); if (pd != NULL) subSet[subIdx++] = pd; } //LOGI("subSet:\n"); //for (i = 0; i < subIdx; i++) // LOGI(" %2d: %s\n", i, subSet[i]->clazz->name); /* * Create an array object to contain "subSet". */ ClassObject* pdArrayClass = NULL; ArrayObject* domains = NULL; pdArrayClass = dvmFindArrayClass("[Ljava/security/ProtectionDomain;", NULL); if (pdArrayClass == NULL) { LOGW("Unable to find ProtectionDomain class for array\n"); goto bail; } domains = dvmAllocArray(pdArrayClass, subIdx, kObjectArrayRefWidth, ALLOC_DEFAULT); if (domains == NULL) {</pre>

The ’476 Patent	Infringed By
	<pre> LOGW("Unable to allocate pd array (%d elems)\n", subIdx); goto bail; } /* copy the ProtectionDomain objects out */ Object** objects = (Object**) domains->contents; for (idx = 0; idx < subIdx; idx++) *objects++ = subSet[idx]; bail: free(subSet); free(methods); dvmReleaseTrackedAlloc((Object*) domains, NULL); RETURN_PTR(domains); } const DalvikNativeMethod dvm_java_security_AccessController[] = { { "getStackDomains", "()[Ljava/security/ProtectionDomain;", Dalvik_java_security_AccessController_getStackDomains }, { NULL, NULL, NULL }, };</pre> *See also, e.g.:* platform/dalvik.git/vm/native/InternalNative.c <pre> #define NUM_DOPRIV_FUNCS 4 /* * Determine if "method" is a "privileged" invocation, i.e. is it one * of the variations of AccessController.doPrivileged(). * * Because the security stuff pulls in a pile of stuff that we may not * want or need, we don't do the class/method lookups at init time, but</pre>

The ’476 Patent	Infringed By
	<pre> * instead on first use. */ bool dvmIsPrivilegedMethod(const Method* method) { int i; assert(method != NULL); if (!gDvm.javaSecurityAccessControllerReady) { /* * Populate on first use. No concurrency risk since we're just * finding pointers to fixed structures. */ static const char* kSignatures[NUM_DOPRIV_FUNCS] = { "(Ljava/security/PrivilegedAction;)Ljava/lang/Object;", "(Ljava/security/PrivilegedExceptionAction;)Ljava/lang/Object;", "(Ljava/security/PrivilegedAction;Ljava/security/AccessControlContext;)Ljava/lang/Object;", "(Ljava/security/PrivilegedExceptionAction;Ljava/security/AccessControlContext;)Ljava/lang/ Object;", }; ClassObject* clazz; clazz = dvmFindClassNoInit("Ljava/security/AccessController;", NULL); if (clazz == NULL) { LOGW("Couldn't find java/security/AccessController\n"); return false; } assert(NELEM(gDvm.methJavaSecurityAccessController_doPrivileged) == NELEM(kSignatures));</pre>

The ’476 Patent	Infringed By
	`/* verify init */` `for (i = 0; i < NUM_DOPRIV_FUNCS; i++) {` `    gDvm.methJavaSecurityAccessController_doPrivileged[i] =` `        dvmFindDirectMethodByDescriptor(clazz, "doPrivileged", kSignatures[i]);` `    if (gDvm.methJavaSecurityAccessController_doPrivileged[i] == NULL) {` `        LOGW("Warning: couldn't find java/security/AccessController"` `            ".doPrivileged %s\n", kSignatures[i]);` `        return false;` `    }` `}` `/* all good, raise volatile readiness flag */` `gDvm.javaSecurityAccessControllerReady = true;` `}` `for (i = 0; i < NUM_DOPRIV_FUNCS; i++) {` `    if (gDvm.methJavaSecurityAccessController_doPrivileged[i] == method) {` `        //LOGI("+++ doPriv match\n");` `        return true;` `    }` `}` `return false;` `}`
the step of determining that said next routine is said first routine includes determining that a flag associated with said next routine indicates said next routine is privileged.	*See* above element of claim 8.

<table>
<tr><th>The ’476 Patent</th><th>Infringed By</th></tr>
<tr><td>9. The method of claim 8, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread.</td><td>See claim 8, supra.

libcore\security-kernel\src\main\java\java\security\AccessController.java:
/**
* Returns the {@code AccessControlContext} for the current {@code Thread}
* including the inherited access control context of the thread that spawned
* the current thread (recursively).
* <p>
* The returned context may be used to perform access checks at a later
* point in time, possibly by another thread.
*
* @return the {@code AccessControlContext} for the current {@code Thread}
* @see Thread#currentThread
* @since Android 1.0
*/
public static AccessControlContext getContext() {

// duplicates (if any) will be removed in ACC constructor
ProtectionDomain[] stack = getStackDomains();

Thread currThread = Thread.currentThread();
if (currThread == null || contexts == null) {
// Big boo time. No need to check anything ?
return new AccessControlContext(stack);
}

ArrayList<AccessControlContext> threadContexts;
synchronized (contexts) {
threadContexts = contexts.get(currThread);
}</td></tr>
</table>

The ’476 Patent	Infringed By
	`` AccessControlContext that; if ((threadContexts == null) || (threadContexts.size() == 0)) { // We were not in doPrivileged method, so // have inherited context here that = SecurityUtils.getContext(currThread); } else { // We were in doPrivileged method, so // Use context passed to the doPrivileged() that = threadContexts.get(threadContexts.size() - 1); } if (that != null && that.combiner != null) { ProtectionDomain[] assigned = null; if (that.context != null && that.context.length != 0) { assigned = new ProtectionDomain[that.context.length]; System.arraycopy(that.context, 0, assigned, 0, assigned.length); } ProtectionDomain[] allpds = that.combiner.combine(stack, assigned); if (allpds == null) { allpds = new ProtectionDomain[0]; } return new AccessControlContext(allpds, that.combiner); } return new AccessControlContext(stack, that); } } ``` (highlighting added).

The ’476 Patent	Infringed By
10. A computer-readable medium	The Accused Instrumentalities include devices that store, distribute, or run Android or the

The ’476 Patent	Infringed By
carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of:	Android SDK, including websites, servers, and mobile devices. These encompass a computer readable medium carrying one or more sequences of one or more instructions. See corresponding elements of claim 1, *supra*.
detecting when a request for an action is made by a principal; and	See corresponding elements of claim 1, *supra*.
in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.	See corresponding elements of claims 1 and 5, *supra*.

The ’476 Patent	Infringed By
11. The computer-readable medium of claim 10, wherein:	*See* corresponding elements of claims 1 and 10, *supra*.
the step of detecting when a request for an action is made includes detecting when a request for an action is made by a thread; and	*See* corresponding elements of claims 1, 2, and 10, *supra*.
the step of determining whether said action is authorized includes	*See* corresponding elements of claims 1, 2, and 10, *supra*.

The ’476 Patent	Infringed By
determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said thread.	

The ’476 Patent	Infringed By
12. The computer readable medium of claim 10, wherein:	*See* corresponding elements of claims 1 and 10, *supra*.
the calling hierarchy includes a first routine; and	*See* corresponding elements of claims 1, 3, and 10, *supra*.
the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.	*See* corresponding elements of claims 1, 3, and 10, *supra*.

The ’476 Patent	Infringed By
13. The computer readable medium of claim 10, wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.	*See* corresponding elements of claims 1 and 10, *supra*.

The ’476 Patent	Infringed By
14. A computer-readable medium bearing instructions for providing security, the instructions including instructions for performing the steps of:	*See* corresponding element of claim 1, *supra*.
detecting when a request for an action is made by a principal;	*See* corresponding element of claim 1, *supra*.
determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal;	*See* corresponding element of claim 1, *supra*.
wherein each routine of said plurality of routines is associated with a class; and	*See* corresponding element of claim 5, *supra*.
wherein said association between permissions and said plurality of routines is based on a second association between classes and protection domains.	*See* corresponding element of claim 5, *supra*.

The ’476 Patent	Infringed By
15. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of:	*See* corresponding element of claim 1, *supra*.

The ’476 Patent	Infringed By
detecting when a request for an action is made by a principal; and	*See* corresponding element of claim 1, *supra*.
in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein a first routine in said calling hierarchy is privileged; and	*See* corresponding elements of claims 1 and 6, *supra*.
wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and a second routine in said calling hierarchy, wherein said second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.	*See* corresponding elements of claims 1 and 6, *supra*.

The ’476 Patent	Infringed By
16. The computer readable medium of claim 15, wherein the step of determining whether said permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and said second routine further includes the steps of:	*See* claims 7 and 15, *supra*.
determining whether said permission required is encompassed by at least one permission associated with said second routine; and	*See* claims 7 and 15, *supra*.
in response to determining said permission required is encompassed by at least one permission associated with said second routine, then performing the steps of:	*See* claims 7 and 15, *supra*.
A) selecting a next routine from said plurality of routines in said calling hierarchy,	*See* claims 7 and 15, *supra*.
B) if said permission required is not encompassed by at least one permission associated with said next routine, then transmitting a message indicating that said permission required is not authorized, and	*See* claims 7 and 15, *supra*.
C) repeating steps A and B until: said permission required is not authorized by at least one	*See* claims 7 and 15, *supra*.

The ’476 Patent	Infringed By
permission associated with said next routine, there are no more routines to select from said plurality of routines in said calling hierarchy, or determining that said next routine is said first routine.	

The ’476 Patent	Infringed By
17. The computer readable medium of claim 16, wherein:	*See* claim 16, *supra*.
the computer readable medium further comprises one or more instructions for performing the step of setting a flag associated with said first routine to indicate that said first routine is privileged; and	*See* claims 8 and 15, *supra*.
the step of determining that said next routine is said first routine includes determining that a flag associated with said next routine indicates said next routine is privileged.	*See* claims 8 and 15, *supra*.

The ’476 Patent	Infringed By
18. The computer readable medium of claim 17, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread.	*See* claims 8, 9, and 15, *supra*.

The ’476 Patent	Infringed By
19. A computer system comprising:	*See* corresponding element of claim 1, *supra*.
a processor;	*See* corresponding element of claim 1, *supra*.
a memory coupled to said processor;	*See* corresponding element of claim 1, *supra*.
said processor being configured to detect when a request for an action is made by a principal; and	*See* corresponding element of claim 1, *supra*.
said processor being configured to respond to detecting the request by determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.	*See* corresponding element of claim 1, *supra*.

The ’476 Patent	Infringed By
20. The computer system of claim 19, wherein:	*See* claims 1and 19, *supra*.
the calling hierarchy includes a first routine; and	*See* claims 1, 3, and 19, *supra*.
said processor is configured to determine whether said action is authorized by determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.	*See* claims 1, 3, and 19, *supra*.

The ’476 Patent	Infringed By
21. The computer system of claim 19, wherein	*See* claim 19, *supra*.
said processor is configured to determine whether said action is authorized by determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.	*See* claims 1, 3, and 19, *supra*.

EXHIBIT F
Preliminary Infringement Contentions for the ’520 Patent

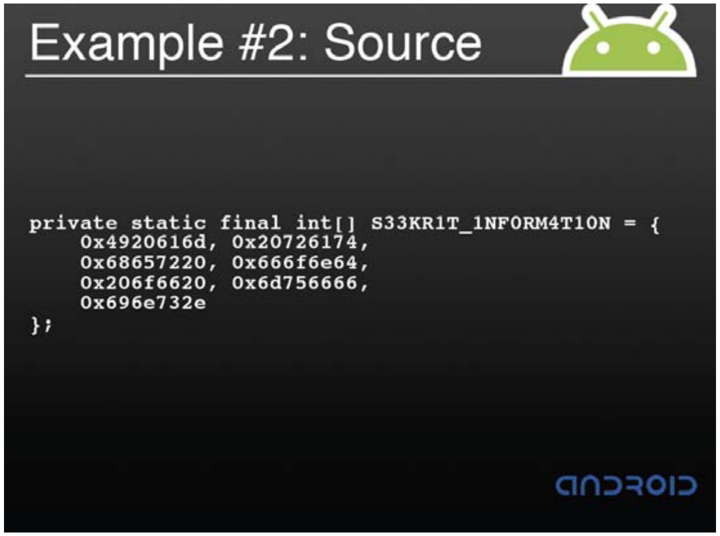
NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’520 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

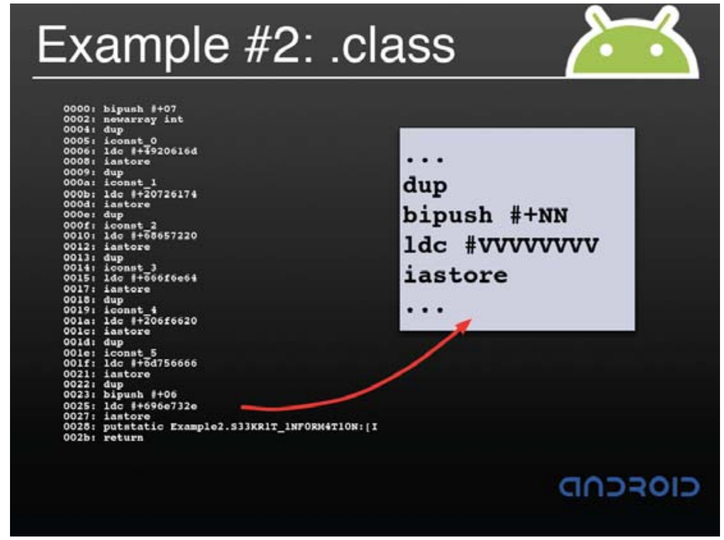
It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

’520 Patent	Infringed By
1. A method in a data processing system for statically initializing an array, comprising the steps of:	Android and its development environment are both stored on computer-readable media containing instructions for controlling a data processing system to perform a method (Android is stored on a computer usable medium, e.g., RAM of a device or computer running Android). Android operates in a data processing system and operates to statically initialize an array as recited by claim 1. *See* Google I/O 2008 Video entitled “*Google I/O 2008 - Dalvik Virtual Machine Internals*,” presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM (“Dalvik Video”), at time 1:50 to 2:30 and 29:50 to 32:00, which describes that instructions are translated from Java (.class bytecode) to a form (.dex bytecode and .dex files) executable or run by the dalvik VM, and includes adding elements to a static array to initialize a data array. *See also* Google I/O 2008 Presentation Slides, entitled, “*Dalvik Virtual Machine Internals, Google I/O 2008*,” presented by Dan Bornstein (“Dalvik Presentation”) at slides 5-7 and 41-45, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0.
compiling source code containing the array with static values	The javac tool compiles source code containing the array to generate a class file with a clinit method containing the .dex bytecode to statically initialize the array to the static values. For example, the Dalvik Video describes compiling source code and .class files (slides 41 and 42) for initialization. *See, e.g.*, time 29:50-32:00 and Dalvik presentation, slides 41-45:

<table>
<tr><th>’520 Patent</th><th>Infringed By</th></tr>
<tr><td>to generate a class file with a clinit method containing byte codes to statically initialize the array to the static values;</td><td>



(Slide 41)



(Slide 42)

See also, e.g., http://developer.android.com/guide/basics/what-is-android.html:

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included “dx” tool.

</td></tr>
<tr><td>receiving the class file into a preloader;</td><td>

As described above, the class file is received by the Android dx tool. This is described, for example, in the Dalvik Video with respect to at least slides 42 and 43, where source and .class files are received for initialization. *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 41-45.

See also:

</td></tr>
</table>

’520 Patent	Infringed By

dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java;

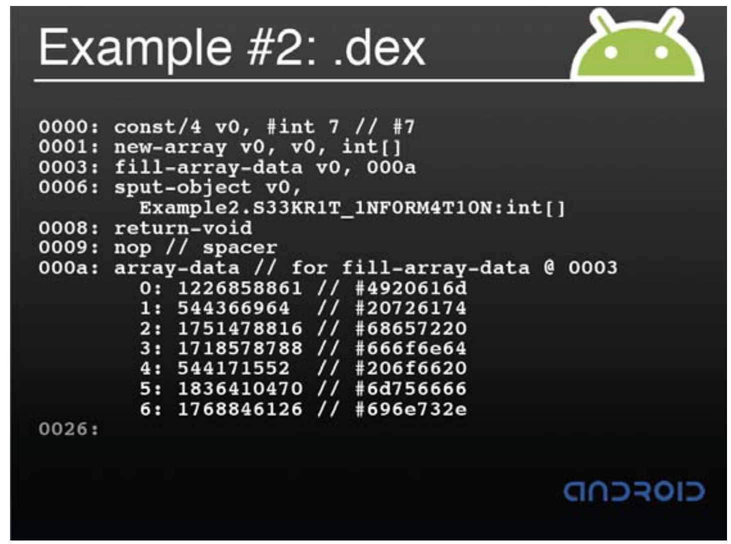
dalvik/dx/src/com/android/dx/cf/code/Simulator.java;

dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and

dalvik/vm/interp/Interp.c.

Android also includes the dx tool, which includes converting/translating .class files to .dex files (slide 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 41-45:

(Slide 43)

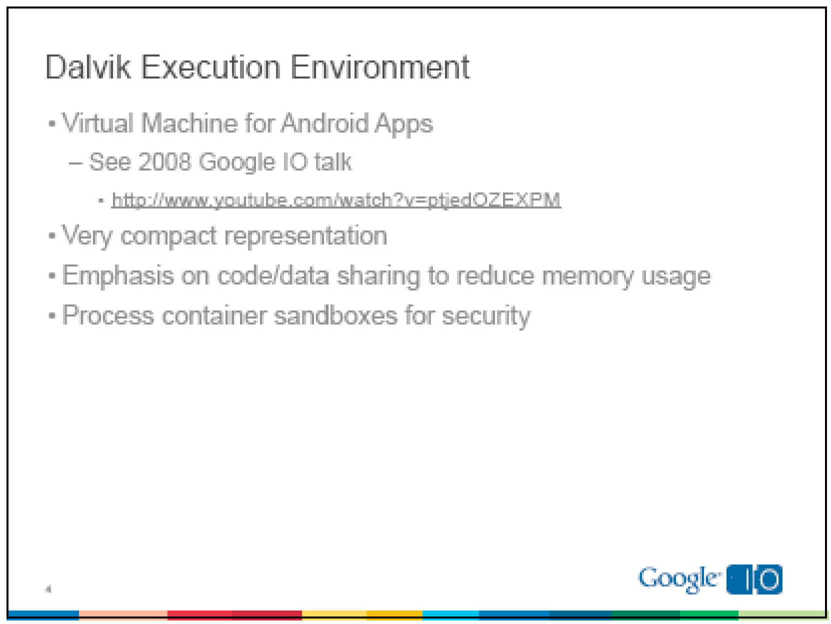


(Slide 44)

See also:

“dx

The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories

’520 Patent	Infringed By
	to Dalvik executable format (.dex) files, so that they can run in the Android environment.” Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html. *See also*: “You write your applications in the Java programming language and they get translated after compilation into a form that runs on the Dalvik virtual machine.” Google I/O 2008 Video, Google I/O 2008 Video, entitled “Dalvik Virtual Machine Internals,” presented by Dan Bornstein (Google Android Project), available at http://developer.android.com/videos/index.html#v=ptjedOZEXPM, at time 1:45. *See also*:  “You are most likely going to write it in the Java programming language and then push the source code through the SDK. And what pops out at the end is an executable targeted to the Dalvik virtual machine.” Google I/O 2010 Video,

’520 Patent	Infringed By
	entitled “A JIT Compiler for Android’s Dalvik VM,” presented by Ben Cheng and Bill Buzbee (Google’s Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo, at time 2:03. *See also*, [platform/sdk.git] / eclipse / plugins / com.android.ide.eclipse.adt / src / com / android / ide / eclipse / adt / internal / build / PostCompilerBuilder.java public class PostCompilerBuilder extends BaseBuilder { ... // build() returns a list of project from which this project depends for future compilation. @SuppressWarnings({“unchecked”}) @Override protected IProject[] build(int kind, Map args, IProgressMonitor monitor) throws CoreException { // get a project object IProject project = getProject(); ... // check classes.dex is present. If not we force to recreate it. if (mConvertToDex == false) { tmp = outputFolder.findMember(SdkConstants.FN_APK_CLASSES_DEX); if (tmp == null || tmp.exists() == false) { mConvertToDex = true; mBuildFinalPackage = true; } } ... // then we check if we need to package the .class into classes.dex if (mConvertToDex) { try { helper.executeDx(javaProject, osBinPath, osBinPath + File.separator + SdkConstants.FN_APK_CLASSES_DEX, referencedJavaProjects, mResourceMarker); } catch (DexException e) { String message = e.getMessage(); AdtPlugin.printErrorToConsole(project, message);

’520 Patent	Infringed By
	`` BaseProjectHelper.markResource(project,AndroidConstants.MARKER_PACKAGING, message, IMarker.SEVERITY_ERROR); Throwable cause = e.getCause(); if (cause instanceof NoClassDefFoundError || cause instanceof NoSuchMethodError) { AdtPlugin.printErrorToConsole(project,Messages.Incompatible_VM_Warning, Messages.Requires_1_5_Error); } // dx failed, we return return allRefProjects; } // build has been done. reset the state of the builder mConvertToDex = false; // and store it saveProjectBooleanProperty(PROPERTY_CONVERT_TO_DEX, mConvertToDex); } ```
	See also [platform/sdk.git] / eclipse / plugins / com.android.ide.eclipse.adt / src / com / android / ide / eclipse / adt / internal / build / PostCompilerHelper.java: /** * Helper with methods for the last 3 steps of the generation of an APK. * * {@link #packageResources(IFile, IProject[], String, int, String, String)} packages the * application resources using aapt into a zip file that is ready to be integrated into the apk. * * {@link #executeDx(IJavaProject, String, String, IJavaProject[])} will convert the Java byte * code into the Dalvik bytecode. * {@link #finalPackage(String, String, String, boolean, IJavaProject, IProject[], IJavaProject[], String,boolean)}

’520 Patent	Infringed By
	* will make the apk from all the previous components. * * This class only executes the 3 above actions. It does not handle the errors, and simply sends * them back as custom exceptions. * * Warnings are handled by the {@link ResourceMarker} interface. * * Console output (verbose and non verbose) is handled through the {@link AndroidPrintStream} passed * to the constructor. * */ public class PostCompilerHelper { ... /** * Execute the Dx tool for dalvik code conversion. * @param javaProject The java project * @param osBinPath the path to the output folder of the project * @param osOutFilePath the path of the dex file to create. * @param referencedJavaProjects the list of referenced projects for this project. * * @throws CoreException * @throws DexException */ public void executeDx(IJavaProject javaProject, String osBinPath, String osOutFilePath, IJavaProject[] referencedJavaProjects, ResourceMarker resMarker) throws CoreException, DexException { IAndroidTarget target = Sdk.getCurrent().getTarget(mProject); AndroidTargetData targetData = Sdk.getCurrent().getTargetData(target); if (targetData == null) { throw new CoreException(new Status(IStatus.ERROR, AdtPlugin.PLUGIN_ID, Messages.ApkBuilder_UnableBuild_Dex_Not_loaded)); }

’520 Patent	Infringed By
	`` // get the dex wrapper DexWrapper wrapper = targetData.getDexWrapper(); if (wrapper == null) { throw new CoreException(new Status(IStatus.ERROR, AdtPlugin.PLUGIN_ID, Messages.ApkBuilder_UnableBuild_Dex_Not_loaded)); } try { // get the list of libraries to include with the source code String[] libraries = getExternalJars(resMarker); // get the list of referenced projects output to add String[] projectOutputs = getProjectOutputs(referencedJavaProjects); String[] fileNames = new String[1 + projectOutputs.length + libraries.length]; // first this project output fileNames[0] = osBinPath; // then other project output System.arraycopy(projectOutputs, 0, fileNames, 1, projectOutputs.length); // then external jars. System.arraycopy(libraries, 0, fileNames, 1 + projectOutputs.length, libraries.length); // set a temporary prefix on the print streams. mOutStream.setPrefix(CONSOLE_PREFIX_DX); mErrStream.setPrefix(CONSOLE_PREFIX_DX); int res = wrapper.run(osOutFilePath, fileNames, mVerbose, mOutStream, mErrStream); ```

’520 Patent	Infringed By
	mOutStream.setPrefix(null); mErrStream.setPrefix(null); if (res != 0) { // output error message and marker the project. String message = String.format(Messages.Dalvik_Error_d, res); throw new DexException(message); } } catch (DexException e) { throw e; } catch (Throwable t) { String message = t.getMessage(); if (message == null) { message = t.getClass().getCanonicalName(); } message = String.format(Messages.Dalvik_Error_s, message); throw new DexException(message, t); } }
simulating execution of the byte codes of the clinit method against a memory without executing the byte codes to identify the static initialization	The dx tool steps through and translates the Java .class files to simulate execution of the bytecodes against a memory without executing the byte codes to identify the static initialization of the array by the preloader. For example, Android does not run Java .class files directly; instead, Java .class files are identified and translated into .dex files using the dx utility. *See, e.g.*, dalvik\dx\src\com\android\dx\dex\cf\CfTranslator.java. This process identities the static initialization of the array by the dx tool. The Dalvik Video further describes source .class files (slides 41 and 42) for initialization, which includes converting/translating to .dex files (slide 44) and adding elements to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 41-45. *See e.g.*, dalvik/dx/src/com/android/dx/cf/code/Simulator.java:

’520 Patent	Infringed By
of the array by the preloader;	/** * Class which knows how to simulate the effects of executing bytecode. * * Note: This class is not thread-safe. If multiple threads * need to use a single instance, they must synchronize access explicitly * between themselves. */ public class Simulator { /** * {@code non-null;} canned error message for local variable * table mismatches */ private static final String LOCAL_MISMATCH_ERROR = “This is symptomatic of .class transformation tools that ignore “ + “local variable information.”; /** {@code non-null;} machine to use when simulating */ private final Machine machine; /** {@code non-null;} array of bytecode */ private final BytecodeArray code; /** {@code non-null;} local variable information */ private final LocalVariableList localVariables; /** {@code non-null;} visitor instance to use */ private final SimVisitor visitor; /** * Constructs an instance. * * @param machine {@code non-null;} machine to use when simulating * @param method {@code non-null;} method data to use

’520 Patent	Infringed By
	*/ public Simulator(Machine machine, ConcreteMethod method) { if (machine == null) { throw new NullPointerException(“machine == null”); } if (method == null) { throw new NullPointerException(“method == null”); } this.machine = machine; this.code = method.getCode(); this.localVariables = method.getLocalVariables(); this.visitor = new SimVisitor(); } /** * Simulates the effect of executing the given basic block. This modifies * the passed-in frame to represent the end result. * * @param bb {@code non-null;} the basic block * @param frame {@code non-null;} frame to operate on */ public void simulate(ByteBlock bb, Frame frame) { int end = bb.getEnd(); visitor.setFrame(frame); try { for (int off = bb.getStart(); off < end; /*off*/) { int length = code.parseInstruction(off, visitor); visitor.setPreviousOffset(off); off += length; }

’520 Patent	Infringed By
	`` } catch (SimException ex) { frame.annotate(ex); throw ex; } } /** * Simulates the effect of the instruction at the given offset, by * making appropriate calls on the given frame. * * @param offset {@code >= 0;} offset of the instruction to simulate * @param frame {@code non-null;} frame to operate on * @return the length of the instruction, in bytes */ public int simulate(int offset, Frame frame) { visitor.setFrame(frame); return code.parseInstruction(offset, visitor); } /** * Constructs an “illegal top-of-stack” exception, for the stack * manipulation opcodes. */ private static SimException illegalTos() { return new SimException(“stack mismatch: illegal “ + “top-of-stack for opcode”); } /** * Bytecode visitor used during simulation. */ private class SimVisitor implements BytecodeArray.Visitor { /** * {@code non-null;} machine instance to use (just to avoid excessive ```

’520 Patent	Infringed By
	<pre> * cross-object field access) */ private final Machine machine; /** * {@code null-ok;} frame to use; set with each call to * {@link Simulator#simulate} */ private Frame frame; /** offset of the previous bytecode */ private int previousOffset; /** * Constructs an instance. */ public SimVisitor() { this.machine = Simulator.this.machine; this.frame = null; }</pre>*See also* dalvik/dx/src/com/android/dx/cf/code/Simulator.java. *See*, *e.g.*, dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java:<pre> /** * Helper to deal with {@code newarray}. * * @param offset the offset to the {@code newarray} opcode itself * @param visitor {@code non-null;} visitor to use * @return instruction length, in bytes */ private int parseNewarray(int offset, Visitor visitor) {</pre>

’520 Patent	Infringed By
| | ```
int value = bytes.getUnsignedByte(offset + 1);
CstType type;
switch (value) {
   case ByteOps.NEWARRAY_BOOLEAN: {
      type = CstType.BOOLEAN_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_CHAR: {
      type = CstType.CHAR_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_DOUBLE: {
      type = CstType.DOUBLE_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_FLOAT: {
      type = CstType.FLOAT_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_BYTE: {
      type = CstType.BYTE_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_SHORT: {
      type = CstType.SHORT_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_INT: {
      type = CstType.INT_ARRAY;
      break;
   }
   case ByteOps.NEWARRAY_LONG: {
      type = CstType.LONG_ARRAY;
      break;
``` |
```

’520 Patent	Infringed By
	`` } default: { throw new SimException(“bad newarray code ” + Hex.u1(value)); } } // Revisit the previous bytecode to find out the length of the array int previousOffset = visitor.getPreviousOffset(); ConstantParserVisitor constantVisitor = new ConstantParserVisitor(); int arrayLength = 0; /* * For visitors that don’t record the previous offset, -1 will be * seen here */ if (previousOffset >= 0) { parseInstruction(previousOffset, constantVisitor); if (constantVisitor.cst instanceof CstInteger && constantVisitor.length + previousOffset == offset) { arrayLength = constantVisitor.value; } } /* * Try to match the array initialization idiom. For example, if the * subsequent code is initializing an int array, we are expecting the * following pattern repeatedly: * dup * push index * push value * *astore * ```

’520 Patent	Infringed By
	`` * where the index value will be incrimented sequentially from 0 up. */ int nInit = 0; int curOffset = offset+2; int lastOffset = curOffset; ArrayList<Constant> initVals = new ArrayList<Constant>(); if (arrayLength != 0) { while (true) { boolean punt = false; // First check if the next bytecode is dup int nextByte = bytes.getUnsignedByte(curOffset++); if (nextByte != ByteOps.DUP) break; // Next check if the expected array index is pushed to the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstInteger) || constantVisitor.value != nInit) break; // Next, fetch the init value and record it curOffset += constantVisitor.length; // Next find out what kind of constant is pushed onto the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstLiteralBits)) break; curOffset += constantVisitor.length; initVals.add(constantVisitor.cst); ```

’520 Patent	Infringed By
	`nextByte = bytes.getUnsignedByte(curOffset++);` `// Now, check if the value is stored to the array properly` `switch (value) {` `    case ByteOps.NEWARRAY_BYTE:` `    case ByteOps.NEWARRAY_BOOLEAN: {` `        if (nextByte != ByteOps.BASTORE) {` `            punt = true;` `        }` `        break;` `    }` `    case ByteOps.NEWARRAY_CHAR: {` `        if (nextByte != ByteOps.CASTORE) {` `            punt = true;` `        }` `        break;` `    }` `    case ByteOps.NEWARRAY_DOUBLE: {` `        if (nextByte != ByteOps.DASTORE) {` `            punt = true;` `        }` `        break;` `    }` `    case ByteOps.NEWARRAY_FLOAT: {` `        if (nextByte != ByteOps.FASTORE) {` `            punt = true;` `        }` `        break;` `    }` `    case ByteOps.NEWARRAY_SHORT: {` `        if (nextByte != ByteOps.SASTORE) {` `            punt = true;` `        }` `        break;`

’520 Patent	Infringed By
	<pre> } case ByteOps.NEWARRAY_INT: { if (nextByte != ByteOps.IASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_LONG: { if (nextByte != ByteOps.LASTORE) { punt = true; } break; } default: punt = true; break; } if (punt) { break; } lastOffset = curOffset; nInit++; } } /* * For singleton arrays it is still more economical to * generate the aput. */ if (nInit < 2 || nInit != arrayLength) { visitor.visitNewarray(offset, 2, type, null); return 2; } else { visitor.visitNewarray(offset, lastOffset - offset, type, initVals);</pre>

’520 Patent	Infringed By
	return lastOffset - offset; } } dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java. *See also:* dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c.
storing into an output file an instruction requesting the static initialization of the array; and	The dx tool stores an instruction requesting the static initialization of the array. This process is described, for example, in the Dalvik Video at time 29:50-32:00 and Dalvik Presentation slides 41-45. *See also:* * Java library array constructor. */ Type componentType = ((CstType) cst).getClassType(); for (int i = 0; i < sourceCount; i++) { componentType = componentType.getComponentType(); } RegisterSpec classReg = RegisterSpec.make(dest.getReg(), Type.CLASS); if (componentType.isPrimitive()) { /* * The component type is primitive (e.g., int as opposed * to Integer), so we have to fetch the corresponding * TYPE class. */ CstFieldRef typeField = CstFieldRef.forPrimitiveType(componentType);

’520 Patent	Infringed By
	`` insn = new ThrowingCstInsn(Rops.GET_STATIC_OBJECT, pos, RegisterSpecList.EMPTY, catches, typeField); } else { /* * The component type is an object type, so just make a * normal class reference. */ insn = new ThrowingCstInsn(Rops.CONST_OBJECT, pos, RegisterSpecList.EMPTY, catches, new CstType(componentType)); } insns.add(insn); // Add a move-result-pseudo for the get-static or const rop = Rops.opMoveResultPseudo(classReg.getType()); insn = new PlainInsn(rop, pos, classReg, RegisterSpecList.EMPTY); insns.add(insn); /* * Add a call to the “multianewarray method,” that is, * Array.newInstance(class, dims). Note: The result type * of newInstance() is Object, which is why the last * instruction in this sequence is a cast to the right * type for the original instruction. */ RegisterSpec objectReg = RegisterSpec.make(dest.getReg(), Type.OBJECT); insn = new ThrowingCstInsn(Rops.opInvokeStatic(MULTIANEWARRAY_METHOD.getPrototype()), pos, RegisterSpecList.make(classReg, dimsReg), ```

’520 Patent	Infringed By
	catches, MULTIANEWARRAY_METHOD); insns.add(insn); // Add a move-result. rop = Rops.opMoveResult(MULTIANEWARRAY_METHOD.getPrototype() .getReturnType()); insn = new PlainInsn(rop, pos, objectReg, RegisterSpecList.EMPTY); insns.add(insn); /* * And finally, set up for the remainder of this method to * add an appropriate cast. */ opcode = ByteOps.CHECKCAST; sources = RegisterSpecList.make(objectReg); } else if (opcode == ByteOps.JSR) { // JSR has no Rop instruction hasJsr = true; return; } else if (opcode == ByteOps.RET) { try { returnAddress = (ReturnAddress)arg(0); } catch (ClassCastException ex) { throw new RuntimeException(“Argument to RET was not a ReturnAddress”, ex); } // RET has no Rop instruction. return; } ropOpcode = jopToRopOpcode(opcode, cst); rop = Rops.ropFor(ropOpcode, destType, sources, cst);

’520 Patent	Infringed By
	`` Insn moveResult = null; if (dest != null && rop.isCallLike()) { /* * We’re going to want to have a move-result in the next * basic block. */ extraBlockCount++; moveResult = new PlainInsn(Rops.opMoveResult(((CstMethodRef) cst).getPrototype() .getReturnType()), pos, dest, RegisterSpecList.EMPTY); dest = null; } else if (dest != null && rop.canThrow()) { /* * We’re going to want to have a move-result-pseudo in the * next basic block. */ extraBlockCount++; moveResult = new PlainInsn(Rops.opMoveResultPseudo(dest.getTypeBearer()), pos, dest, RegisterSpecList.EMPTY); dest = null; } if (ropOpcode == RegOps.NEW_ARRAY) { /* * In the original bytecode, this was either a primitive * array constructor “newarray” or an object array * constructor “anewarray”. In the former case, there is * no explicit constant, and in the latter, the constant * is for the element type and not the array type. The rop * instruction form for both of these is supposed to be ```

’520 Patent	Infringed By
	<pre> * the resulting array type, so we initialize / alter * “cst” here, accordingly. Conveniently enough, the rop * opcode already gets constructed with the proper array * type. */ cst = CstType.intern(rop.getResult()); } else if ((cst == null) && (sourceCount == 2)) { TypeBearer lastType = sources.get(1).getTypeBearer(); if (lastType.isConstant() && advice.hasConstantOperation(rop, sources.get(0), sources.get(1))) { /* * The target architecture has an instruction that can * build in the constant found in the second argument, * so pull it out of the sources and just use it as a * constant here. */ cst = (Constant) lastType; sources = sources.withoutLast(); rop = Rops.ropFor(ropOpcode, destType, sources, cst); } } SwitchList cases = getAuxCases(); ArrayList<Constant> initValues = getInitValues(); boolean canThrow = rop.canThrow(); blockCanThrow |= canThrow; if (cases != null) { if (cases.size() == 0) { // It’s a default-only switch statement. It can happen! insn = new PlainInsn(Rops.GOTO, pos, null,</pre>

’520 Patent	Infringed By
	`` RegisterSpecList.EMPTY); primarySuccessorIndex = 0; } else { IntList values = cases.getValues(); insn = new SwitchInsn(rop, pos, dest, sources, values); primarySuccessorIndex = values.size(); } } else if (ropOpcode == RegOps.RETURN) { /* * Returns get turned into the combination of a move (if * non-void and if the return doesn’t already mention * register 0) and a goto (to the return block). */ if (sources.size() != 0) { RegisterSpec source = sources.get(0); TypeBearer type = source.getTypeBearer(); if (source.getReg() != 0) { insns.add(new PlainInsn(Rops.opMove(type), pos, RegisterSpec.make(0, type), source)); } } insn = new PlainInsn(Rops.GOTO, pos, null, RegisterSpecList.EMPTY); primarySuccessorIndex = 0; updateReturnOp(rop, pos); returns = true; } else if (cst != null) { if (canThrow) { insn = new ThrowingCstInsn(rop, pos, sources, catches, cst); catchesUsed = true; primarySuccessorIndex = catches.size(); } else { insn = new PlainCstInsn(rop, pos, dest, sources, cst); ```

’520 Patent	Infringed By
	`` } } else if (canThrow) { insn = new ThrowingInsn(rop, pos, sources, catches); catchesUsed = true; if (opcode == ByteOps.ATHROW) { /* * The op athrow is the only one where it’s possible * to have non-empty successors and yet not have a * primary successor. */ primarySuccessorIndex = -1; } else { primarySuccessorIndex = catches.size(); } } else { insn = new PlainInsn(rop, pos, dest, sources); } insns.add(insn); if (moveResult != null) { insns.add(moveResult); } /* * If initValues is non-null, it means that the parser has * seen a group of compatible constant initialization * bytecodes that are applied to the current newarray. The * action we take here is to convert these initialization * bytecodes into a single fill-array-data ROP which lays out * all the constant values in a table. */ if (initValues != null) { extraBlockCount++; ```

’520 Patent	Infringed By
	`` insn = new FillArrayDataInsn(Rops.FILL_ARRAY_DATA, pos, RegisterSpecList.make(moveResult.getResult()), initValues, cst); insns.add(insn); } } ``` dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java. *See, e.g.*, dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java: ``` /* * Try to match the array initialization idiom. For example, if the * subsequent code is initializing an int array, we are expecting the * following pattern repeatedly: * dup * push index * push value * *astore * * where the index value will be incrimented sequentially from 0 up. */ int nInit = 0; int curOffset = offset+2; int lastOffset = curOffset; ArrayList<Constant> initVals = new ArrayList<Constant>(); if (arrayLength != 0) { while (true) { boolean punt = false; // First check if the next bytecode is dup int nextByte = bytes.getUnsignedByte(curOffset++); if (nextByte != ByteOps.DUP) ```

’520 Patent	Infringed By
	`` break; // Next check if the expected array index is pushed to the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstInteger) || constantVisitor.value != nInit) break; // Next, fetch the init value and record it curOffset += constantVisitor.length; // Next find out what kind of constant is pushed onto the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstLiteralBits)) break; curOffset += constantVisitor.length; initVals.add(constantVisitor.cst); nextByte = bytes.getUnsignedByte(curOffset++); // Now, check if the value is stored to the array properly switch (value) { case ByteOps.NEWARRAY_BYTE: case ByteOps.NEWARRAY_BOOLEAN: { if (nextByte != ByteOps.BASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_CHAR: { if (nextByte != ByteOps.CASTORE) { punt = true; ```

’520 Patent	Infringed By
	} break; } case ByteOps.NEWARRAY_DOUBLE: { if (nextByte != ByteOps.DASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_FLOAT: { if (nextByte != ByteOps.FASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_SHORT: { if (nextByte != ByteOps.SASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_INT: { if (nextByte != ByteOps.IASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_LONG: { if (nextByte != ByteOps.LASTORE) { punt = true; } break; } default:

’520 Patent	Infringed By
	punt = true; break; } if (punt) { break; } lastOffset = curOffset; nInit++; } } /* * For singleton arrays it is still more economical to * generate the aput. */ if (nInit < 2 || nInit != arrayLength) { visitor.visitNewarray(offset, 2, type, null); return 2; } else { visitor.visitNewarray(offset, lastOffset - offset, type, initVals); return lastOffset - offset; } } dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java. *See also:* dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c.
interpreting the instruction by a virtual	The Dalvik virtual machine interprets the instruction to perform the static initialization of the array. This process is described, for example, in the Dalvik Video at time 29:50-32:00 and Dalvik Presentation slides 41-45. *See also:*

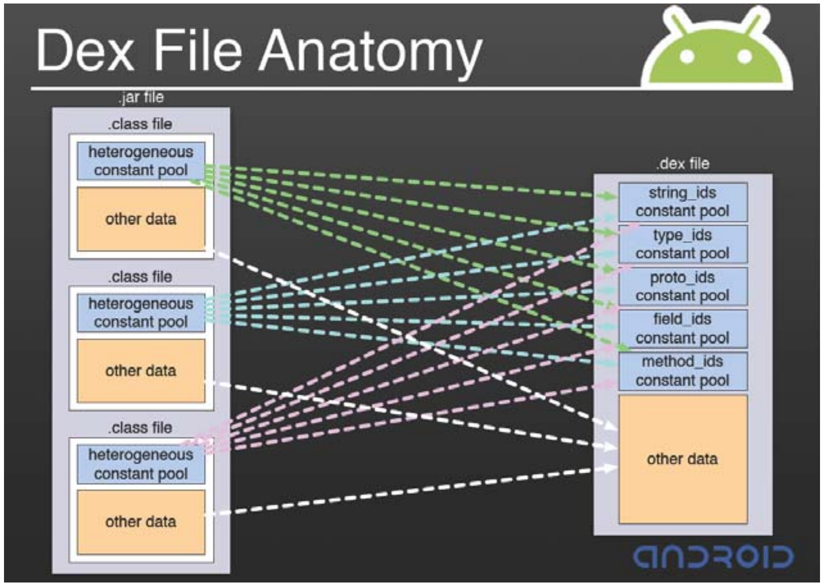
’520 Patent	Infringed By
machine to perform the static initialization of the array.	<pre>/* * Fill the array with predefined constant values. * * Returns true if job is completed, otherwise false to indicate that * an exception has been thrown. */ bool dvmInterpHandleFillArrayData(ArrayObject* arrayObj, const u2* arrayData) { u2 width; u4 size; if (arrayObj == NULL) { dvmThrowException(“Ljava/lang/NullPointerException;”, NULL); return false; } /* * Array data table format: * ushort ident = 0x0300 magic value * ushort width width of each element in the table * uint size number of elements in the table * ubyte data[size*width] table of data values (may contain a single-byte * padding at the end) * * Total size is 4+(width * size + 1)/2 16-bit code units. */ if (arrayData[0] != kArrayDataSignature) { dvmThrowException(“Ljava/lang/InternalError;”, “bad array data magic”); return false; } width = arrayData[1]; size = arrayData[2]	(((u4)arrayData[3]) << 16);</pre>

’520 Patent	Infringed By
	<pre> if (size > arrayObj->length) { dvmThrowException(“Ljava/lang/ArrayIndexOutOfBoundsException;”, NULL); return false; } copySwappedArrayData(arrayObj->contents, &arrayData[4], size, width); return true; } /* * Find the concrete method that corresponds to “methodIdx”. The code in * “method” is executing invoke-method with “thisClass” as its first argument. * * Returns NULL with an exception raised on failure. */ Method* dvmInterpFindInterfaceMethod(ClassObject* thisClass, u4 methodIdx, const Method* method, DvmDex* methodClassDex) { Method* absMethod; Method* methodToCall; int i, vtableIndex; /* * Resolve the method. This gives us the abstract method from the * interface class declaration. */ absMethod = dvmDexGetResolvedMethod(methodClassDex, methodIdx); if (absMethod == NULL) { absMethod = dvmResolveInterfaceMethod(method->clazz, methodIdx); if (absMethod == NULL) { LOGV(“+ unknown method\n”); return NULL; } }</pre>

’520 Patent	Infringed By
	`` /* make sure absMethod->methodIndex means what we think it means */ assert(dvmIsAbstractMethod(absMethod)); /* * Run through the “this” object’s iftable. Find the entry for * absMethod’s class, then use absMethod->methodIndex to find * the method’s entry. The value there is the offset into our * vtable of the actual method to execute. * * The verifier does not guarantee that objects stored into * interface references actually implement the interface, so this * check cannot be eliminated. */ for (i = 0; i < thisClass->iftableCount; i++) { if (thisClass->iftable[i].clazz == absMethod->clazz) break; } if (i == thisClass->iftableCount) { /* impossible in verified DEX, need to check for it in unverified */ dvmThrowException(“Ljava/lang/IncompatibleClassChangeError;”, “interface not implemented”); return NULL; } assert(absMethod->methodIndex < thisClass->iftable[i].clazz->virtualMethodCount); vtableIndex = thisClass->iftable[i].methodIndexArray[absMethod->methodIndex]; assert(vtableIndex >= 0 && vtableIndex < thisClass->vtableCount); methodToCall = thisClass->vtable[vtableIndex]; #if 0 /* this can happen when there’s a stale class file */ ```

’520 Patent	Infringed By
	`` if (dvmIsAbstractMethod(methodToCall)) { dvmThrowException(“Ljava/lang/AbstractMethodError;”, “interface method not implemented”); return NULL; } #else assert(!dvmIsAbstractMethod(methodToCall) || methodToCall->nativeFunc != NULL); #endif LOGVV(“+++ interface=%s.%s concrete=%s.%s\n”, absMethod->clazz->descriptor, absMethod->name, methodToCall->clazz->descriptor, methodToCall->name); assert(methodToCall != NULL); return methodToCall; } ``` dalvik/vm/interp/Interp.c.

‘520 Patent	Infringed By
2. The method of claim 1 wherein the storing step includes step of:	*See* Claim 1, *supra*.
storing a constant pool entry into the constant pool.	The Android dx tool forms a shared table of the duplicated elements from the plurality of class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 15-20. The Dalvik Presentation shows the elements of the class files combining into a shared constant

‘520 Patent	Infringed By
	pool (shared tables) in the .dex file.  (Dalvik Presentation, slide 15) In the illustration above, each of “string_ids,” “type_ids” and “method_ids” are examples of the shared tables (or, equivalently, a collective shared table). In the Android source code, *see also generally*: “Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type

‘520 Patent	Infringed By
	* com.android.dx.util” dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java, dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java from Android 2.2.

‘520 Patent	Infringed By
3. The method of claim 1 wherein the play executing step includes the steps of:	*See* claim 1, *supra*. The class files are stack based, and as such, the dx tool play executes java bytecode stacks, thereby performing stack manipulation on the allocated stack as recited.
allocating a stack;	*See* claim 1, *supra*. The class files are stack based, and as such, the dx tool allocates a stack for play execution. *See also, e.g.,* http://developer.android.com/guide/basics/what-is-android.html: **Android Runtime** Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a

‘520 Patent	Infringed By
	Java language compiler that have been transformed into the .dex format by the included “dx” tool. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.
reading a byte code from the clinit method that manipulates the stack; and	*See* claim 1, *supra*. The dx tool reads byte code from the clinit method that manipulates the stack. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.
performing the stack manipulation on the allocated stack.	*See* claim 1, *supra*. The dx tool performs stack manipulation of the allocated stack. *See also*:

‘520 Patent	Infringed By
	dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.

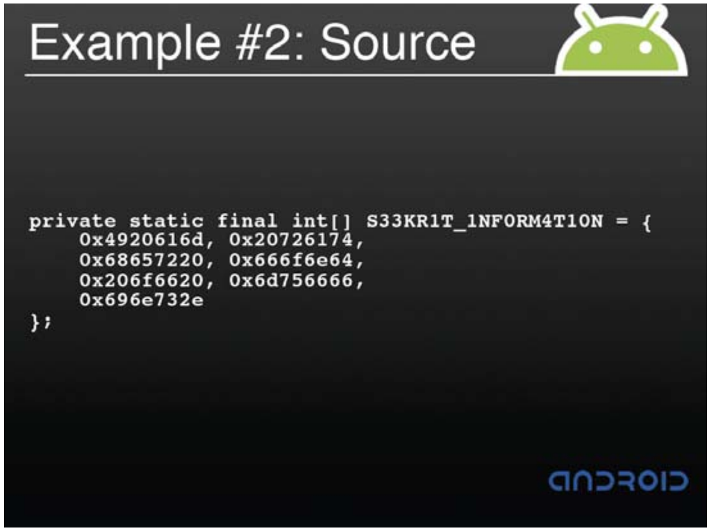
‘520 Patent	Infringed By
4. The method of claim 1 wherein the play executing step includes the steps of:	*See* claim 1, *supra*. The dx tool play executes java bytecode stacks, including allocating variables (which are registers), reading byte code from the clinit method, manipulating local variables, and manipulating the local variables on the allocated variables.
allocating variables;	*See* claim 1, *supra*. The dx tool allocates variables (which are registers). *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.
reading a byte code from the clinit method that manipulates local	*See* claim 1, *supra*. The dx tool reads byte code from the clinit method that manipulates local

‘520 Patent	Infringed By
variables of the clinit method; and	variables. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.
performing the manipulation of the local variables on the allocated variables.	*See* claim 1 *supra*. The dx tool performs the manipulation of the local variables on the allocated variables. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. Android also includes the dx tool, which includes converting/translating .class files to .dex files (slides 15-20, 44) and adding elements (e.g., static values) to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00 and Dalvik Presentation, slides 15-20, 41-45.

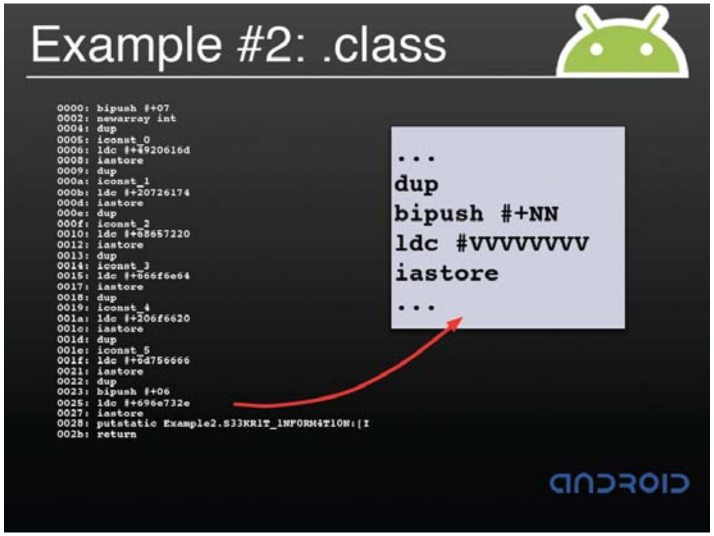
‘520 Patent	Infringed By

'520 Patent	Infringed By
6. A method in a data processing system, comprising the steps of:	Android and its development environment are both stored on computer-readable media containing instructions for controlling a data processing system to perform a method (Android is stored on a computer usable medium, e.g., RAM of a device or computer running Android). This analysis applies the claimed methods to the conversion of .class bytecodes into .dex bytecodes. An Android runtime environment executes instructions created by operation of the method of claim 6. *See* Google I/O 2008 Video entitled "*Google I/O 2008 - Dalvik Virtual Machine Internals*," presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM ("Dalvik Video"), at time 1:50 to 2:30, which describes that instructions are translated from Java (.class bytecode) to a form (.dex bytecode) executable or run by the dalvik VM. *See also* Google I/O 2008 Presentation Slides, entitled, "*Dalvik Virtual Machine Internals, Google I/O 2008*," presented by Dan Bornstein ("Dalvik Presentation") at slides 5-7, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0.
receiving code to be run on a processing component to perform an operation;	The Android dx tool receives Java .class files containing bytecode instructions. **"dx** The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories to Dalvik executable format (.dex) files, so that they can run in the Android environment." Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html
play executing the code without running the code on the processing	The dx tool steps through and translates the Java .class files to simulate execution of the bytecode without running the code on the processing component to identify its operation if it were run on the processing component. For example, Android does not run Java .class files directly; instead, Java .class files are identified and translated into .dex files using the dx utility. *See, e.g.*, dalvik\dx\src\com\android\dx\dex\cf\CfTranslator.java. The Dalvik Video further describes source .class files (slides 41 and 42) for initialization, which includes

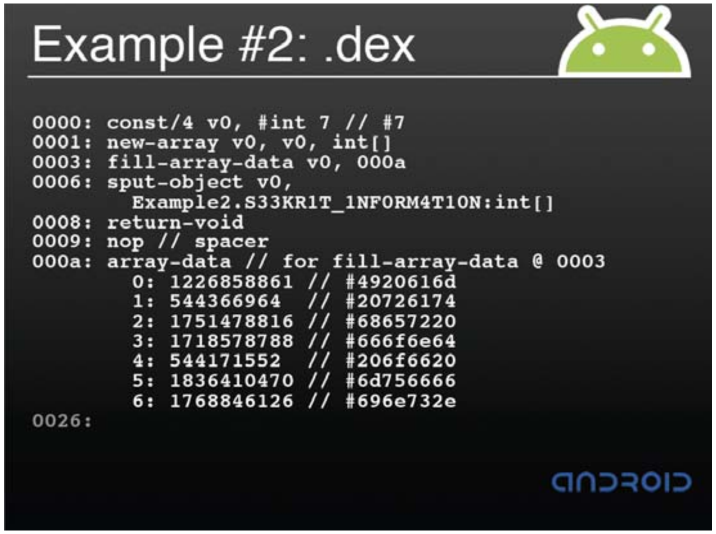
‘520 Patent	Infringed By
component to identify the operation if the code were run by the processing component; and	converting/translating to .dex files (slides 15-20, 44) and adding elements to a static array to initialize the data (slide 45). *See, e.g.*, time 29:50-32:00, and Dalvik Presentation slides 15-20, 41-45.



(Slide 41)



(Slide 42)



‘520 Patent	Infringed By
	(Slide 43) (Slide 44) *See e.g.*, dalvik/dx/src/com/android/dx/cf/code/Simulator.java: /** * Class which knows how to simulate the effects of executing bytecode. * * Note: This class is not thread-safe. If multiple threads * need to use a single instance, they must synchronize access explicitly * between themselves. */ public class Simulator { /** * {@code non-null;} canned error message for local variable * table mismatches */ private static final String LOCAL_MISMATCH_ERROR = “This is symptomatic of .class transformation tools that ignore “ + “local variable information.”; /** {@code non-null;} machine to use when simulating */ private final Machine machine; /** {@code non-null;} array of bytecode */ private final BytecodeArray code; /** {@code non-null;} local variable information */ private final LocalVariableList localVariables; /** {@code non-null;} visitor instance to use */ private final SimVisitor visitor; /**

‘520 Patent	Infringed By
	* Constructs an instance. * * @param machine {@code non-null;} machine to use when simulating * @param method {@code non-null;} method data to use */ public Simulator(Machine machine, ConcreteMethod method) { if (machine == null) { throw new NullPointerException(“machine == null”); } if (method == null) { throw new NullPointerException(“method == null”); } this.machine = machine; this.code = method.getCode(); this.localVariables = method.getLocalVariables(); this.visitor = new SimVisitor(); } /** * Simulates the effect of executing the given basic block. This modifies * the passed-in frame to represent the end result. * * @param bb {@code non-null;} the basic block * @param frame {@code non-null;} frame to operate on */ public void simulate(ByteBlock bb, Frame frame) { int end = bb.getEnd(); visitor.setFrame(frame); try { for (int off = bb.getStart(); off < end; /*off*/) {

‘520 Patent	Infringed By
	<pre>int length = code.parseInstruction(off, visitor); visitor.setPreviousOffset(off); off += length; } } catch (SimException ex) { frame.annotate(ex); throw ex; } } /** * Simulates the effect of the instruction at the given offset, by * making appropriate calls on the given frame. * * @param offset {@code >= 0;} offset of the instruction to simulate * @param frame {@code non-null;} frame to operate on * @return the length of the instruction, in bytes */ public int simulate(int offset, Frame frame) { visitor.setFrame(frame); return code.parseInstruction(offset, visitor); } /** * Constructs an “illegal top-of-stack” exception, for the stack * manipulation opcodes. */ private static SimException illegalTos() { return new SimException(“stack mismatch: illegal “ + “top-of-stack for opcode”); } /** * Bytecode visitor used during simulation.</pre>

‘520 Patent	Infringed By
	`` */ private class SimVisitor implements BytecodeArray.Visitor { /** * {@code non-null;} machine instance to use (just to avoid excessive * cross-object field access) */ private final Machine machine; /** * {@code null-ok;} frame to use; set with each call to * {@link Simulator#simulate} */ private Frame frame; /** offset of the previous bytecode */ private int previousOffset; /** * Constructs an instance. */ public SimVisitor() { this.machine = Simulator.this.machine; this.frame = null; } ``` dalvik/dx/src/com/android/dx/cf/code/Simulator.java. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c.
creating an instruction	The dx tool rewrites the .class bytecodes into .dex bytecodes (stored in .dex files) for the processing component to perform the identified operation.

‘520 Patent	Infringed By
for the processing component to perform the operation.	dx The dx tool rewrites .class bytecode into Android bytecode (stored in .dex files.) Android Developer Tools available at http://developer.android.com/guide/developing/tools/index.html. **“dx** The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories to Dalvik executable format (.dex) files, so that they can run in the Android environment.” Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html. *See also*: dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c. *See, e.g.*, dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java: /* * Try to match the array initialization idiom. For example, if the * subsequent code is initializing an int array, we are expecting the * following pattern repeatedly: * dup * push index * push value * *astore *

‘520 Patent	Infringed By
	`` * where the index value will be incrimented sequentially from 0 up. */ int nInit = 0; int curOffset = offset+2; int lastOffset = curOffset; ArrayList<Constant> initVals = new ArrayList<Constant>(); if (arrayLength != 0) { while (true) { boolean punt = false; // First check if the next bytecode is dup int nextByte = bytes.getUnsignedByte(curOffset++); if (nextByte != ByteOps.DUP) break; // Next check if the expected array index is pushed to the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstInteger) || constantVisitor.value != nInit) break; // Next, fetch the init value and record it curOffset += constantVisitor.length; // Next find out what kind of constant is pushed onto the stack parseInstruction(curOffset, constantVisitor); if (constantVisitor.length == 0 || !(constantVisitor.cst instanceof CstLiteralBits)) break; curOffset += constantVisitor.length; initVals.add(constantVisitor.cst); ```

‘520 Patent	Infringed By
	nextByte = bytes.getUnsignedByte(curOffset++); // Now, check if the value is stored to the array properly switch (value) { case ByteOps.NEWARRAY_BYTE: case ByteOps.NEWARRAY_BOOLEAN: { if (nextByte != ByteOps.BASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_CHAR: { if (nextByte != ByteOps.CASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_DOUBLE: { if (nextByte != ByteOps.DASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_FLOAT: { if (nextByte != ByteOps.FASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_SHORT: { if (nextByte != ByteOps.SASTORE) { punt = true; } break;

'520 Patent	Infringed By
	`` } case ByteOps.NEWARRAY_INT: { if (nextByte != ByteOps.IASTORE) { punt = true; } break; } case ByteOps.NEWARRAY_LONG: { if (nextByte != ByteOps.LASTORE) { punt = true; } break; } default: punt = true; break; } if (punt) { break; } lastOffset = curOffset; nInit++; } } /* * For singleton arrays it is still more economical to * generate the aput. */ if (nInit < 2 || nInit != arrayLength) { visitor.visitNewarray(offset, 2, type, null); return 2; } else { visitor.visitNewarray(offset, lastOffset - offset, type, initVals); ```

‘520 Patent	Infringed By
	return lastOffset - offset; } } dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java. *See also:* dalvik/dx/src/com/android/dx/cf/code/Simulator.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c.

‘520 Patent	Infringed By
7. The method of claim 6 wherein the operation initializes a data structure, and wherein the play executing step includes the step of:	The dx tool operates to initialize a data structure by converting the target files to .dex files. *See, e.g.*, dalvik\dx\src\com\android\dx\dex\cf\CfTranslator.java.
play executing the code to identify the initialization of the data structure.	The dx tool further play executes the code to identify the initialization of the data structure by stepping through the .class files. *See e.g.*, dalvik\dx\src\com\android\dx\dex\cf\CfTranslator.java. *See e.g.*, dalvik/dx/src/com/android/dx/cf/code/Simulator.java: /** * Class which knows how to simulate the effects of executing bytecode. * * Note: This class is not thread-safe. If multiple threads * need to use a single instance, they must synchronize access explicitly * between themselves. */ public class Simulator {

'520 Patent	Infringed By
	<pre>/** * {@code non-null;} canned error message for local variable * table mismatches */ private static final String LOCAL_MISMATCH_ERROR = "This is symptomatic of .class transformation tools that ignore " + "local variable information."; /** {@code non-null;} machine to use when simulating */ private final Machine machine; /** {@code non-null;} array of bytecode */ private final BytecodeArray code; /** {@code non-null;} local variable information */ private final LocalVariableList localVariables; /** {@code non-null;} visitor instance to use */ private final SimVisitor visitor; /** * Constructs an instance. * * @param machine {@code non-null;} machine to use when simulating * @param method {@code non-null;} method data to use */ public Simulator(Machine machine, ConcreteMethod method) { if (machine == null) { throw new NullPointerException("machine == null"); } if (method == null) { throw new NullPointerException("method == null"); }</pre>

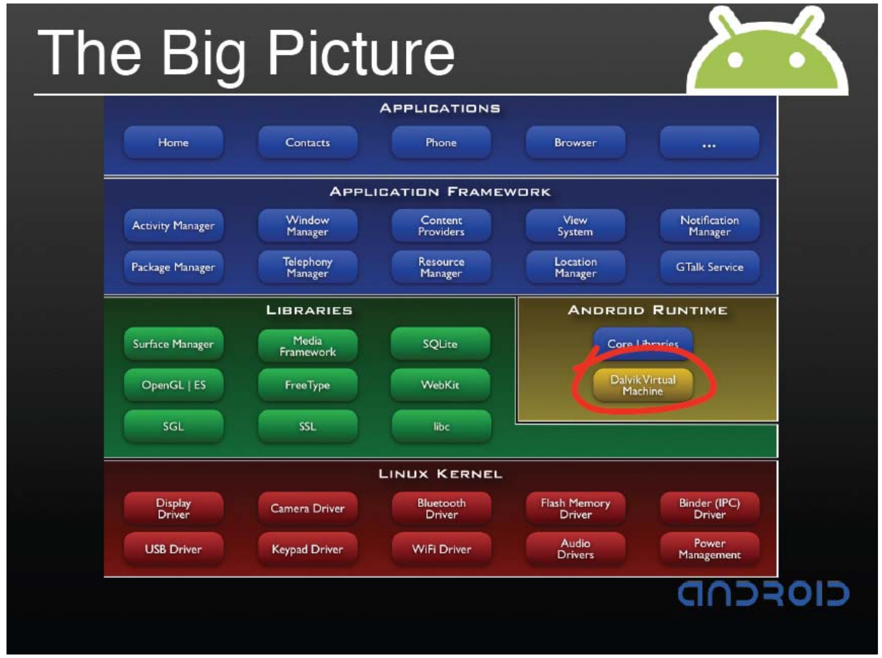
‘520 Patent	Infringed By
	<pre> this.machine = machine; this.code = method.getCode(); this.localVariables = method.getLocalVariables(); this.visitor = new SimVisitor(); } /** * Simulates the effect of executing the given basic block. This modifies * the passed-in frame to represent the end result. * * @param bb {@code non-null;} the basic block * @param frame {@code non-null;} frame to operate on */ public void simulate(ByteBlock bb, Frame frame) { int end = bb.getEnd(); visitor.setFrame(frame); try { for (int off = bb.getStart(); off < end; /*off*/) { int length = code.parseInstruction(off, visitor); visitor.setPreviousOffset(off); off += length; } } catch (SimException ex) { frame.annotate(ex); throw ex; } } /** * Simulates the effect of the instruction at the given offset, by * making appropriate calls on the given frame.</pre>

‘520 Patent	Infringed By
	<pre> * * @param offset {@code >= 0;} offset of the instruction to simulate * @param frame {@code non-null;} frame to operate on * @return the length of the instruction, in bytes */ public int simulate(int offset, Frame frame) { visitor.setFrame(frame); return code.parseInstruction(offset, visitor); } /** * Constructs an “illegal top-of-stack” exception, for the stack * manipulation opcodes. */ private static SimException illegalTos() { return new SimException(“stack mismatch: illegal “ + “top-of-stack for opcode”); } /** * Bytecode visitor used during simulation. */ private class SimVisitor implements BytecodeArray.Visitor { /** * {@code non-null;} machine instance to use (just to avoid excessive * cross-object field access) */ private final Machine machine; /** * {@code null-ok;} frame to use; set with each call to * {@link Simulator#simulate} */ private Frame frame;</pre>

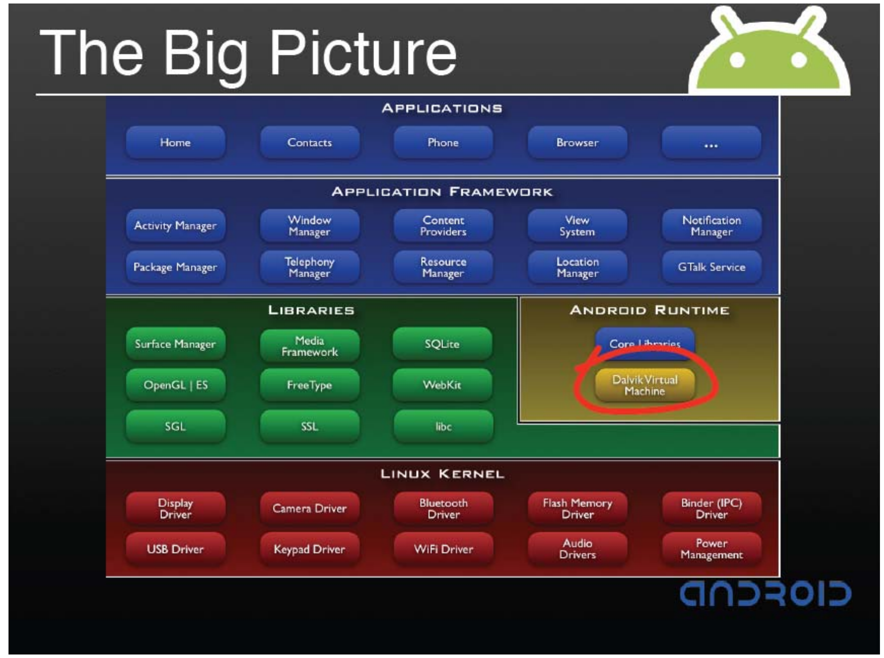
‘520 Patent	Infringed By
	`/** offset of the previous bytecode */` `private int previousOffset;` `/**` ` * Constructs an instance.` ` */` `public SimVisitor() {` `    this.machine = Simulator.this.machine;` `    this.frame = null;` `}` dalvik/dx/src/com/android/dx/cf/code/Simulator.java. *See also:* dalvik/dx/src/com/android/dx/cf/code/BytecodeArray.java; dalvik/dx/src/com/android/dx/cf/code/RopperMachine.java; and dalvik/vm/interp/Interp.c.

‘520 Patent	Infringed By
8. The method of claim 6 wherein the operation statically initializes an array and wherein the play executing step includes the step of:	*See* claim 1, *supra*.
play executing the code to identify the static initialization of the array.	*See* claim 1, *supra*.

'520 Patent	Infringed By
9. The method of claim 6 further including the step of:	*See* claim 1, *supra*.
running the created instruction on the processing component to perform the operation.	An Android runtime environment executes instructions created by operation of the method of claim 6. *See* Google I/O 2008 Video entitled "*Google I/O 2008 - Dalvik Virtual Machine Internals*," presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM ("Dalvik Video"), at time 1:50 to 2:30, which describes that instructions are translated from Java (.class bytecode) to a form (.dex bytecode) executable or run by the dalvik VM. *See also* Google I/O 2008 Presentation Slides, entitled, "*Dalvik Virtual Machine Internals, Google I/O 2008*," presented by Dan Bornstein ("Dalvik Presentation") at slides 5-7, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0.

‘520 Patent	Infringed By
	 (Dalvik Presentation, slide 5)

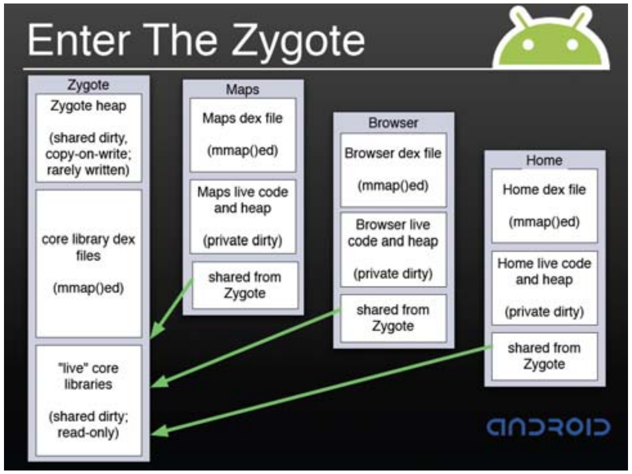
‘520 Patent	Infringed By
10. The method of claim 6 further including the step of:	
interpreting the created instruction by a virtual machine to perform the operation.	Android’s dalvik virtual machine interprets the .dex bytecode instructions (stored in .dex files) created by operation of the method of claim 6. Dalvik Video at time 1:50 to 2:30 describes that instructions are translated from Java (.class bytecode) to a form (.dex bytecode) executable or run by the dalvik VM. *See also*, Dalvik Presentation, slides 5-7.

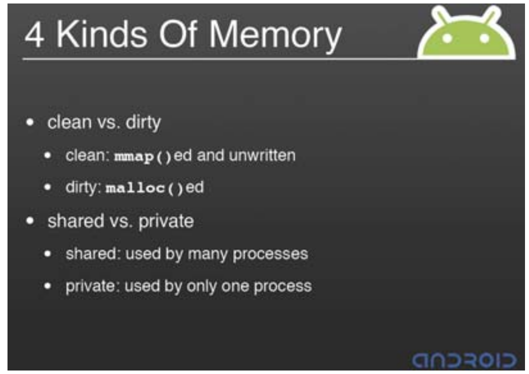
<table>
<tr><th>‘520 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>


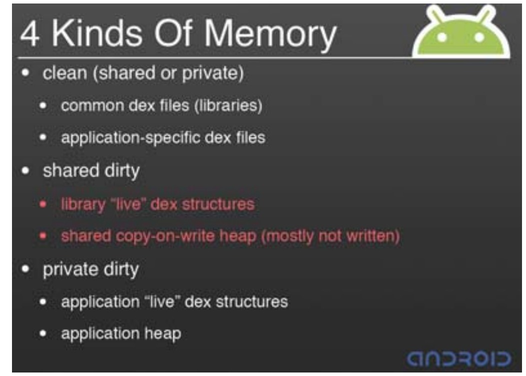
(Dalvik Presentation, slide 5)

See also claim 1, supra.
</td></tr>
</table>

‘520 Patent	Infringed By
11. The method of claim 6 wherein the operation has an effect on memory, and wherein the play executing step includes the step of:	The operation of the instruction by the Dalvik VM has an effect on memory. This is explained in the Dalvik Video at time 13:40-15:45 and Dalvik Presentation, slides 25-27.

'520 Patent	Infringed By
	 (Dalvik Presentation, Slide 26) dx tool maps in .dex files and the Zygote creates heap, dirty memory for classes/methods (when commanded to start application), starts fork, and makes heap of objects, library dex structure, and shares memory with the zygote. Note that the child processes in Slide 26 ("Maps," "Browser," and "Home"), have references to the memory space of the Zygote, as shown by the green arrows. *See also* corresponding Dalvik Video at 13:50-15:20. In the above example, the "Maps live code and heap", "Browser live code and heap" and "Home live code and heap" are stored as private dirty memory, defined as:

<table>
<tr><th>‘520 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>




(Dalvik Presentation, Slides 23 and 27)
</td></tr>
<tr><td>play executing the code to identify the effect on the memory.</td><td>See CfTranslator.java, which updates statistics on .dex files. This method may be used to identify the effect on the memory of optimizing methods.</td></tr>
</table>

<table>
<tr><th>‘520 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td><pre><code>/**
 * Helper that updates the dex statistics.
 */
private static void updateDexStatistics(CfOptions args,
        RopMethod optRmeth, RopMethod nonOptRmeth,
        LocalVariableInfo locals, int paramSize, int originalByteCount) {
    /*
     * Run rop->dex again on optimized vs. non-optimized method to
     * collect statistics. We have to totally convert both ways,
     * since converting the "real" method getting added to the
     * file would corrupt it (by messing with its constant pool
     * indices).
     */

    DalvCode optCode = RopTranslator.translate(optRmeth,
            args.positionInfo, locals, paramSize);
    DalvCode nonOptCode = RopTranslator.translate(nonOptRmeth,
            args.positionInfo, locals, paramSize);

    /*
     * Fake out the indices, so code.getInsns() can work well enough
     * for the current purpose.
     */

    DalvCode.AssignIndicesCallback callback =
        new DalvCode.AssignIndicesCallback() {
            public int getIndex(Constant cst) {
                // Everything is at index 0!
                return 0;
            }
        };

    optCode.assignIndices(callback);
    nonOptCode.assignIndices(callback);

    CodeStatistics.updateDexStatistics(nonOptCode, optCode);
    CodeStatistics.updateOriginalByteCount(originalByteCount);
}
}</code></pre></td></tr>
</table>

‘520 Patent	Infringed By
12. A data processing system comprising:	Any device or computer which can run the Android dx tool.

‘520 Patent	Infringed By
a storage device containing:	A storage memory, e.g., RAM, of the device or computer running Android.
a program with source code that statically initializes a data structure; and	*See* claim 1, *supra*.
class files, wherein one of the class files contains a clinit method that statically initializes the data structure;	*See* claim 1, *supra*.
a memory containing:	A storage memory, e.g., RAM, storing Android.
a compiler for compiling the program and generating the class files; and	*See* claim 1, *supra*.
a preloader for consolidating the class files, for play executing the clinit method to determine the static initialization the clinit method performs, and for creating an instruction to perform the static initialization; and	*See* claim 1, *supra*.
a processor for running the compiler and the preloader.	The processor of the device running Android runs the dx tool.

‘520 Patent	Infringed By
13. The data processing system of claim 12 wherein the preloader includes a mechanism for generating an output file	*See* claim 1, *supra*.

‘520 Patent	Infringed By
containing the created instruction.	

‘520 Patent	Infringed By
14. The data processing system of claim 13 wherein the memory further includes a virtual machine that interprets the created instruction to perform the static initialization.	*See* claim 1, *supra*.

‘520 Patent	Infringed By
15. The data processing system of claim 12, wherein the data structure is an array.	*See* claim 1, *supra*.

‘520 Patent	Infringed By
16. The data processing system of claim 12 wherein the clinit method has byte codes that statically initialize the data structure.	*See* claim 1, *supra*.

‘520 Patent	Infringed By
17. The data processing system of claim 12 wherein the created	*See* claim 2, *supra*.

‘520 Patent	Infringed By
instruction includes an entry into a constant pool.	

‘520 Patent	Infringed By
18. A computer-readable medium containing instructions for controlling a data processing system to perform a method, comprising the steps of:	*See* claim 6, *supra*.
receiving code to be run on a processing component to perform an operation;	*See* claim 6, *supra*.
simulating execution of the code without running the code on the processing component to identify the operation if the code were run by the processing component; and	*See* claim 6, *supra*.
creating an instruction for the processing component to perform the operation.	*See* claim 1, *supra*.

‘520 Patent	Infringed By
19. The computer-readable medium of claim 18 wherein the operation initializes a data structure, and wherein the simulating step includes the step	*See* claim 7, *supra*.

‘520 Patent	Infringed By
of:	
simulating execution of the code to identify the initialization of the data structure.	*See* claim 7, *supra*.

‘520 Patent	Infringed By
20. The computer-readable medium of claim 18 wherein the operation statically initializes an array and wherein the simulating step includes the step of:	*See* claims 1 and 8, *supra*.
simulating execution of the code to identify the static initialization of the array.	*See* claims 1 and 8, *supra*.

‘520 Patent	Infringed By
21. The computer-readable medium of claim 18 further including the step of:	*See* claim 9, *supra*.
running the created instruction on the processing component to perform the operation.	*See* claim 9, *supra*.

‘520 Patent	Infringed By

‘520 Patent	Infringed By
22. The computer-readable medium of claim 18 further including the step of:	*See* claim 10, *supra*.
interpreting the created instruction by a virtual machine to perform the operation.	*See* claim 10, *supra*.

‘520 Patent	Infringed By
23. The computer-readable medium of claim 18 wherein the operation has an effect on memory, and wherein the simulating step includes the step of:	*See* claim 11, *supra*.
simulating execution of the code to identify the effect on the memory.	*See* claim 11, *supra*.

EXHIBIT G

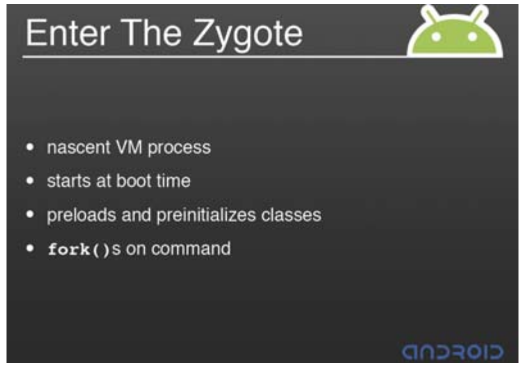
Preliminary Infringement Contentions for US 7,426,720 (’720 Patent)

NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2 and current versions of Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’720 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), and 2.2 (“Froyo”).

The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c).

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

The ’720 Patent	Infringed By
1.pre. A system for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	A system running Android for dynamic preloading of classes through memory space cloning of a master runtime system process. An example of a master runtime system process is a zygote process, which creates a Dalvik virtual machine instance and which forks upon request to create new Dalvik virtual machine instances for various applications.
1.a. A processor;	A processor of a computer or smartphone running Android.
1.b. A memory	A memory of a computer or smartphone running Android.
1.c. a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code;	Android includes a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code. *See* Presentation slides corresponding to the Dalvik Video: “Dalvik Virtual Machine Internals, Google I/O 2008,” by Dan Bornstein, http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf (“Dalvik Presentation”), at slide 25; and corresponding Video: “Google I/O 2008 - Dalvik Virtual Machine Internals,” by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM (“Dalvik Video”), at time 13:50-15:20.

The ’720 Patent	Infringed By
	 (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: “What we do with the zygote, as its name implies, it’s, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and methods ….” *See also* Presentation slides corresponding to the Android Video: “Anatomy and Physiology of an Android, Google I/O 2008,” by Patrick Brady, http://sites.google.com/site/io/anatomy--physiology-of-an-android/Android-Anatomy-GoogleIO.pdf (“Android Presentation”), at slide 82; and corresponding Video: “Google I/O 2008 – Anatomy and Physiology of an Android,” by Patrick Brady, http://developer.android.com/videos/index.html#v=G-36noTCaiA (“Android Video”), at time 43:15-49:00.  (Android Presentation, Slide 82) Corresponding Android Video at 44:30: “The init process starts up a really neat process called zygote. As its name

The ’720 Patent	Infringed By
	implies, zygote is really just the beginning of all of the rest of the Android platform. And so zygote is a nascent VM process that initializes a Dalvik VM and preloads a lot of its libraries….” Example source code files in, *e.g.*, base\preloaded-classes, base\core\java\com\android\internal\os\ZygoteInit.java *See*, *e.g.*, base\preloaded-classes. *# Classes which are preloaded by com.android.internal.os.ZygoteInit.* *# Automatically generated by frameworks/base/tools/preload/WritePreloadedClassFile.java.* *# MIN_LOAD_TIME_MICROS=1250* *android.R$styleable* *android.accounts.AccountManager* … *dalvik.system.Zygote* *java.beans.PropertyChangeEvent* *java.beans.PropertyChangeListener* … *See*, *e.g.*, base\core\java\com\android\internal\os\ZygoteInit.java. */*** ** Performs Zygote process initialization. Loads and initializes* ** commonly used classes.* *** ** Most classes only cause a few hundred bytes to be allocated, but* ** a few will allocate a dozen Kbytes (in one case, 500+K).* **/* *private static void preloadClasses() {* *final VMRuntime runtime = VMRuntime.getRuntime();* *InputStream is = ZygoteInit.class.getClassLoader().getResourceAsStream(* *PRELOADED_CLASSES);* *if (is == null) {* *Log.e(TAG, “Couldn’t find “ + PRELOADED_CLASSES + “.”);* *} else {* *Log.i(TAG, “Preloading classes...”);* … *try {* *BufferedReader br* *= new BufferedReader(new InputStreamReader(is), 256);* *int count = 0;* *String line;*

The ’720 Patent	Infringed By
	String missingClasses = null; *while ((line = br.readLine()) != null) {* *// Skip comments and blank lines.* *line = line.trim();* *if (line.startsWith(“#”) || line.equals(““)) {* *continue;* *}* *try {* *if (Config.LOGV) {* *Log.v(TAG, “Preloading “ + line + “...”);* *}* *Class.forName(line);* *if (Debug.getGlobalAllocSize() > PRELOAD_GC_THRESHOLD) {* *if (Config.LOGV) {* *Log.v(TAG,* *“ GC at “ + Debug.getGlobalAllocSize());* *}* *runtime.gcSoftReferences();* *runtime.runFinalizationSync();* *Debug.resetGlobalAllocSize();* *}* *count++;* **…** *}* *}* *}*
1.d. a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process;	Android includes a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process. *See*  (Android Presentation, Slide 80) Corresponding Android Video at 43:28: “Like any Linux-based or Unix-based system, at startup, the bootloader is gonna boot Linux and it’s gonna kick off the init process. This is similar to how any Linux system really starts up.”

<table>
<tr><th>The ’720 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

(Android Presentation, Slide 81)

Corresponding Android Video at 43:41:
“The first thing init is going to do on Android is start some low level, ah, processes called Linux daemons. And these are typically used to handle things like low level hardware interfaces, um, and they would sit on top of the abstraction layer and run and listen on sockets for things like USB connections or, you know, Android Debug Bridge or ADB connections, the Debugger connections and also the Radio Interface Layer daemon, which will sit on top of, um, on top of the radio baseband and interface with the baseband modem.”

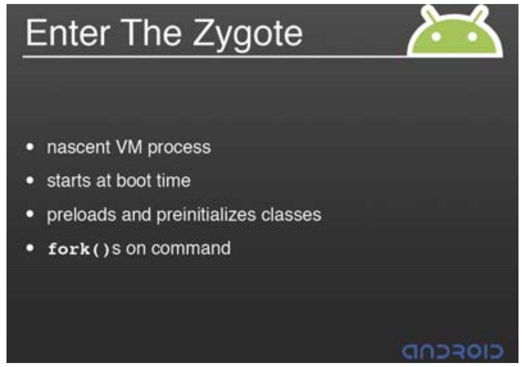
(Android Presentation, Slide 82)

Corresponding Android Video at 44:25:
“Ah, after starting up the Linux daemons, and we’ll collapse those in the corner of the screen here to save some space, the init process starts up a really neat process called zygote. And as its name implies, zygote is really just the beginning of all of the rest of the Android platform. And so zygote is a nascent, ah, VM process that initializes a Dalvik VM and preloads a lot of these libraries….”

See also

</td></tr>
</table>

The ’720 Patent	Infringed By



(Dalvik Presentation, Slide 25)

Corresponding Dalvik Video at 13:48:
“What we do with the zygote, as its name implies, it’s, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and methods ….”

Example source code files in
base\core\jni\AndroidRuntime.cpp,
base\cmds\app_process\app_main.cpp,
base\core\java\com\android\internal\os\ZygotInit.java

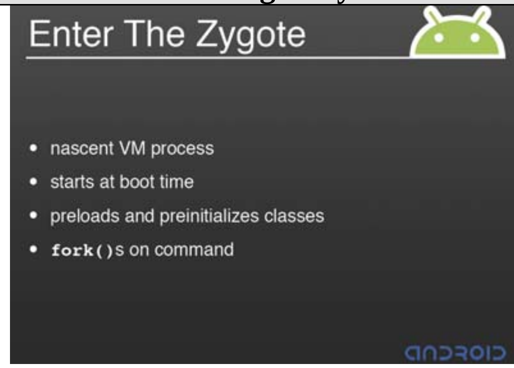
Example code call chain,
Class AppRuntime in app_main.cpp passes ZygoteInit class name to AndroidRuntime::startVm,
AndroidRuntime::start(className) calls startVm,
AndroidRuntime::startVm calls JNI_CreateJavaVM(),
AndroidRuntime::start calls CallStaticVoidMethod(ZygoteInit className.main)

See, *e.g.*, base\core\java\com\android\internal\os\ZygoteInit.java.

```
/**
 * Startup class for the zygote process.
 *
 * Pre-initializes some classes, and then waits for commands on a UNIX domain
 * socket. Based on these commands, forks of child processes that inherit
 * the initial state of the VM.
 *
 * Please see {@link ZygoteConnection.Arguments} for documentation on the
 * client protocol.
 *
```

The ’720 Patent	Infringed By
	` * @hide` ` */` **…** *public static void main(String argv[]) {* *try {* *// Start profiling the zygote initialization.* *SamplingProfilerIntegration.start();* *registerZygoteSocket();* *EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_START,* *SystemClock.uptimeMillis());* *preloadClasses();* *//cacheRegisterMaps();* *preloadResources();* *EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_END,* *SystemClock.uptimeMillis());* *if (SamplingProfilerIntegration.isEnabled()) {* *SamplingProfiler sp = SamplingProfiler.getInstance();* *sp.pause();* *SamplingProfilerIntegration.writeZygoteSnapshot();* *sp.shutDown();* *}* *// Do an initial gc to clean up after startup* *gc();* *// If requested, start system server directly from Zygote* *if (argv.length != 2) {* *throw new RuntimeException(argv[0] + USAGE_STRING);* *}* *if (argv[1].equals(“true”)) {* *startSystemServer();* *} else if (!argv[1].equals(“false”)) {* *throw new RuntimeException(argv[0] + USAGE_STRING);* *}* *Log.i(TAG, “Accepting command socket connections”);* *if (ZYGOTE_FORK_MODE) {* *runForkMode();* *} else {* *runSelectLoopMode();* *}* *closeServerSocket();* *} catch (MethodAndArgsCaller caller) {* *caller.run();* *} catch (RuntimeException ex) {* *Log.e(TAG, “Zygote died with exception”, ex);* *closeServerSocket();* *throw ex;* *}* *}*
1.e. a runtime environment to clone the memory space as a child runtime system process responsive	Android includes a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process. *See*

The ’720 Patent	Infringed By
to a process request and to execute the child runtime system process; and	 (Android Presentation, Slide 55) Corresponding Android Video at 33:40: “So we’ve covered the native libraries, we’ve covered everything down to the Linux kernel, and the real magic of the Android platform happens in the layers above this. And that’s what we’ll go into now, starting with the Android runtime. The Android runtime sits on top of the libraries and Linux kernel and it provides (1) the Dalvik virtual machine and the core libraries, here written in blue, because they are exposed through the Java programming languages.”  (Android Presentation, Slide 56) Corresponding Android Video at 34:04: “So Dalvik virtual machine. Remember Android is not Linux. We don’t have a native windowing system. All of the applications and services that you run, will be running inside a virtual environment powered by the Dalvik virtual machine….” *See also*

<table>
<tr><th>The ’720 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>


(Dalvik Presentation, Slide 25)

Corresponding Dalvik Video at 13:48:
“What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this.”

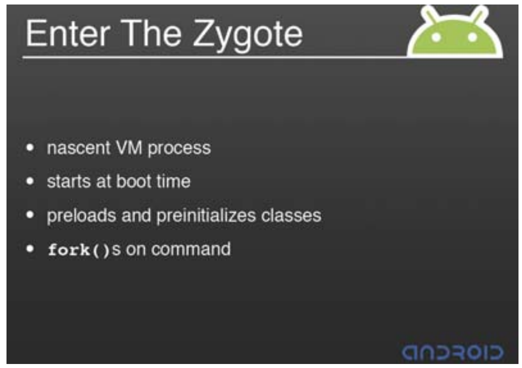
See also claim 1.f. below.
</td></tr>
<tr><td>1.f. a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.</td><td>
Android includes a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

See

(Android Presentation, Slide 82)

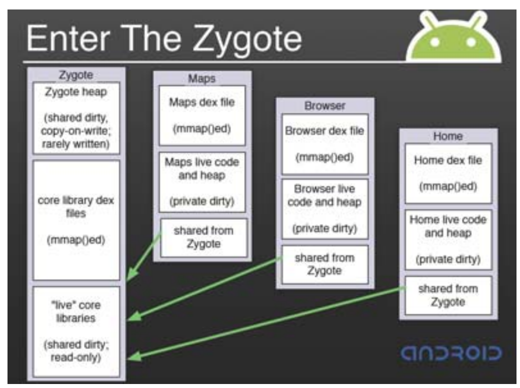
Corresponding Android Video at 44:30:
“The init process starts up a really neat process called zygote….It uses copy-on-write to maximize re-use and minimize footprint so that data structures are shared and it won’t do a full copy unless some of those data structures are to be modified.”
</td></tr>
</table>

The ’720 Patent	Infringed By
	See also



(Dalvik Presentation, Slide 25)

Corresponding Dalvik Video at 13:48:
“What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this.”



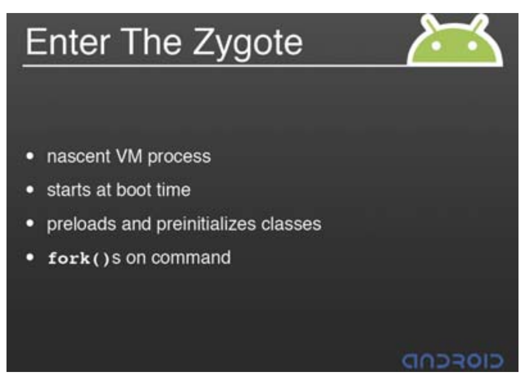
(Dalvik Presentation, Slide 26)

Corresponding Dalvik Video at 14:40:
“So the zygote, again, has made, has made this heap of objects, it’s made this live dex structure and then each application that then starts up, instead of having its own memory for those things, it just shares it with the zygote and also with any other app that’s also on the system.”

See also http://developer.android.com/guide/basics/what-is-android.html.
“Android Runtime
…The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

The ’720 Patent	Infringed By
	Linux Kernel Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.” *See also*, Lowe, Robert, Linux Kernel Process Management, April 15, 2005. Sample Chapter is provided courtesy of Sams, http://www.informit.com/articles/article.aspx?p=370047&seqNum=2&rll=1. “Copy-on-Write …In Linux, fork() is implemented through the use of copy-on-write pages. Copy-on-write (or COW) is a technique to delay or altogether prevent copying of the data. Rather than duplicate the process address space, the parent and the child can share a single copy. The data, however, is marked in such a way that if it is written to, a duplicate is made and each process receives a unique copy.” Example source code files in libcore\dalvik\src\main\java\dalvik\system\Zygote.java, dalvik\vm\native\dalvik_system_Zygote.c, linux-2.6\kernel\fork.c. *See, e.g.*, libcore\dalvik\src\main\java\dalvik\system\Zygote.java. */*** ** Forks a new Zygote instance, but does not leave the zygote mode.* ** The current VM must have been started with the -Xzygote flag. The* ** new child is expected to eventually call forkAndSpecialize()* *** ** @return 0 if this is the child, pid of the child* ** if this is the parent, or -1 on error* **/* *native public static int fork();* */*** ** Forks a new VM instance. The current VM must have been started* ** with the -Xzygote flag. NOTE: new instance keeps all* ** root capabilities. The new process is expected to call capset().* *** ** @param uid the UNIX uid that the new process should setuid() to after* ** fork()ing and and before spawning any threads.* ** @param gid the UNIX gid that the new process should setgid() to after* ** fork()ing and and before spawning any threads.* ** @param gids null-ok; a list of UNIX gids that the new process should* ** setgroups() to after fork and before spawning any threads.*

The ’720 Patent	Infringed By
	<pre>* @param debugFlags bit flags that enable debugging features. * @param rlimits null-ok an array of rlimit tuples, with the second * dimension having a length of 3 and representing * (resource, rlim_cur, rlim_max). These are set via the posix * setrlimit(2) call. * * @return 0 if this is the child, pid of the child * if this is the parent, or -1 on error. */ native public static int forkAndSpecialize(int uid, int gid, int[] gids, int debugFlags, int[][] rlimits);</pre> *See, e.g.*, dalvik\vm\native\dalvik_system_Zygote.c. <pre>/* native public static int forkAndSpecialize(int uid, int gid, * int[] gids, int debugFlags); */ static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args, JValue* pResult) { pid_t pid; pid = forkAndSpecializeCommon(args); RETURN_INT(pid); } … /* * Utility routine to fork zygote and specialize the child process. */ static pid_t forkAndSpecializeCommon(const u4* args) { pid_t pid; uid_t uid = (uid_t) args[0]; gid_t gid = (gid_t) args[1]; ArrayObject* gids = (ArrayObject *)args[2]; u4 debugFlags = args[3]; ArrayObject *rlimits = (ArrayObject *)args[4]; if (!gDvm.zygote) { dvmThrowException(“Ljava/lang/IllegalStateException;”, “VM instance not started with -Xzygote”); return -1; } if (!dvmGcPreZygoteFork()) { LOGE(“pre-fork heap failed\n”); dvmAbort(); } setSignalHandler(); dvmDumpLoaderStats(“zygote”); pid = fork(); if (pid == 0) { int err; /* The child process */ …. } else if (pid > 0) {</pre>

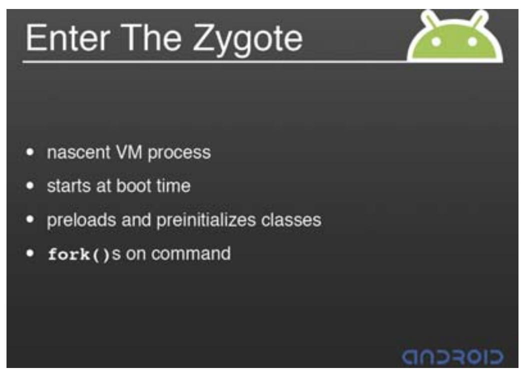
The ’720 Patent	Infringed By
	/* the parent process */ } return pid; } *See*, *e.g.*, linux-2.6\kernel\fork.c. /* * Ok, this is the main fork-routine. * * It copies the process, and if successful kick-starts * it and waits for it to finish using the VM if required. */ long do_fork(unsigned long clone_flags, unsigned long stack_start, struct pt_regs *regs, unsigned long stack_size, int __user *parent_tidptr, int __user *child_tidptr) { struct task_struct *p; int trace = 0; long nr; … p = copy_process(clone_flags, stack_start, regs, stack_size, wake_up_new_task(p, clone_flags); … tracehook_report_clone_complete(trace, regs, clone_flags, nr, p); … return nr; }
2. A system according to claim 1, further comprising: a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process.	Android includes a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process. *See*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48:

pa-1435310

The ’720 Patent	Infringed By
	“What we do with the zygote, as its name implies, it’s, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and methods….” *See, e.g.*, dalvik\vm\oo\Class.c. */* * Find the named class (by descriptor), using the specified * initiating ClassLoader. * * The class will be loaded and initialized if it has not already been. * If necessary, the superclass will be loaded. * * If the class can’t be found, returns NULL with an appropriate exception * raised. */ ClassObject* dvmFindClass(const char* descriptor, Object* loader) { ClassObject* clazz; clazz = dvmFindClassNoInit(descriptor, loader); if (clazz != NULL && clazz->status < CLASS_INITIALIZED) { /* initialize class */ if (!dvmInitClass(clazz)) { /* init failed; leave it in the list, marked as bad */ assert(dvmCheckException(dvmThreadSelf())); assert(clazz->status == CLASS_ERROR); return NULL; } } return clazz; }* */* * Find the named class (by descriptor), using the specified * initiating ClassLoader. * * The class will be loaded if it has not already been, as will its * superclass. It will not be initialized. * * If the class can’t be found, returns NULL with an appropriate exception * raised. */ ClassObject* dvmFindClassNoInit(const char* descriptor, Object* loader) { assert(descriptor != NULL); //assert(loader != NULL); LOGVV(“FindClassNoInit ‘%s’ %p\n”, descriptor, loader);*

<table>
<tr><th>The ’720 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>if (*descriptor == ‘[‘) {
/*
* Array class. Find in table, generate if not found.
*/
return dvmFindArrayClass(descriptor, loader);
} else {
/*
* Regular class. Find in table, load if not found.
*/
if (loader != NULL) {
return findClassFromLoaderNoInit(descriptor, loader);
} else {
return dvmFindSystemClassNoInit(descriptor);
}
}
}</td></tr>
<tr><td>3. A system according to claim 2, further comprising: a class locator to locate the source definition if the instantiated class definition is unavailable in the local cache.</td><td>Android includes a class locator to locate the source definition if the instantiated class definition is unavailable in the local cache.

See



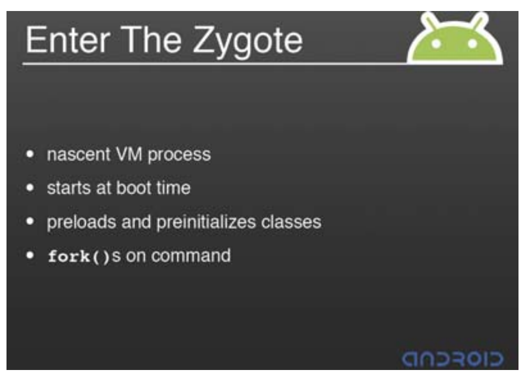
(Dalvik Presentation, Slide 25)

Corresponding Dalvik Video at 13:48:
“What we do with the zygote, as its name implies, it’s, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and methods….”

Example source code files in
dalvik\vm\oo\Class.c.

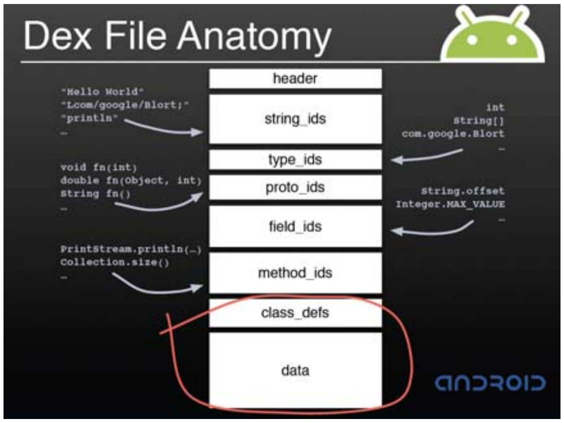
Example code call chain for application classloader,
Class.forName calls Class.classForName,
Class.classForName calls dvmFindClassByName,
dvmFindClassByName calls dvmFindClass,</td></tr>
</table>

The ’720 Patent	Infringed By
	dvmFindClass calls dvmFindClassNoInit, dvmFindClassNoInit calls findClassFromLoaderNoInit, findClassFromLoaderNoInit calls dvmLookupClass, If dvmLookupClass returns NULL, calls ClassLoader.loadClass, Else dvmLookupClass returns class from gDvm.loadedClasses (a table of loaded classes) Example code call chain for boot classloader, Class.forName calls Class.classForName, Class.classForName calls dvmFindClassByName, dvmFindClassByName calls dvmFindClass, dvmFindClass calls dvmFindClassNoInit, dvmFindClassNoInit calls dvmFindSystemClassNoInit, dvmFindSystemClassNoInit calls findClassNoInit, findClassNoInit calls dvmLookupClass, If dvmLookupClass returns NULL, calls ClassLoader.loadClass, Else dvmLookupClass returns class from gDvm.loadedClasses (a table of loaded classes) *See*, *e.g.*, dalvik\vm\oo\Class.c. */* * Find the named class (by descriptor), using the specified * initiating ClassLoader. * * The class will be loaded and initialized if it has not already been. * If necessary, the superclass will be loaded. * * If the class can’t be found, returns NULL with an appropriate exception * raised. */ ClassObject* dvmFindClass(const char* descriptor, Object* loader) { ClassObject* clazz; clazz = dvmFindClassNoInit(descriptor, loader); if (clazz != NULL && clazz->status < CLASS_INITIALIZED) { /* initialize class */ if (!dvmInitClass(clazz)) { /* init failed; leave it in the list, marked as bad */ assert(dvmCheckException(dvmThreadSelf())); assert(clazz->status == CLASS_ERROR); return NULL; } } return clazz; } /* * Find the named class (by descriptor), using the specified * initiating ClassLoader.*

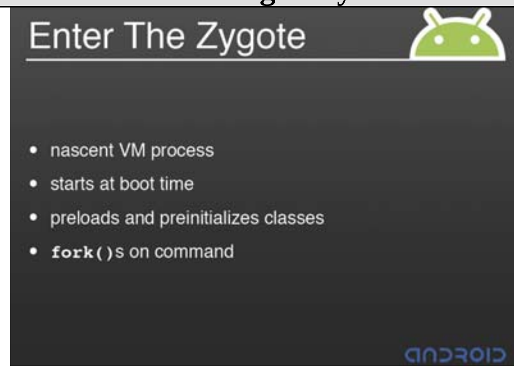
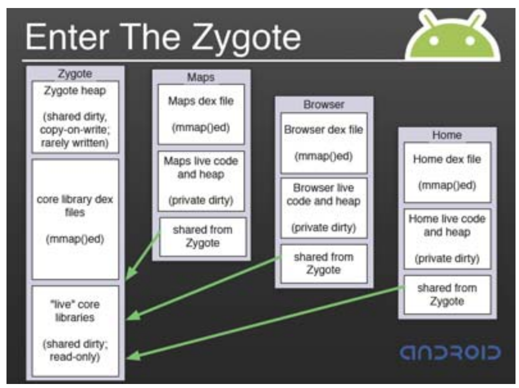
The ’720 Patent	Infringed By
	* * The class will be loaded if it has not already been, as will its * superclass. It will not be initialized. * * If the class can’t be found, returns NULL with an appropriate exception * raised. */ ClassObject* dvmFindClassNoInit(const char* descriptor, Object* loader) { assert(descriptor != NULL); //assert(loader != NULL); LOGVV(“FindClassNoInit ‘%s’ %p\n”, descriptor, loader); if (*descriptor == ‘[‘) { /* * Array class. Find in table, generate if not found. */ return dvmFindArrayClass(descriptor, loader); } else { /* * Regular class. Find in table, load if not found. */ if (loader != NULL) { return findClassFromLoaderNoInit(descriptor, loader); } else { return dvmFindSystemClassNoInit(descriptor); } } }
4. A system according to claim 1, further comprising: a class resolver to resolve the class definition.	Android includes a class resolver to resolve the class definition. *See*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: “What we do with the zygote, as its name implies, it’s, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and

The ’720 Patent	Infringed By
	methods….” *See*, *e.g.*, dalvik\vm\oo\Class.c. ** * Link (prepare and resolve). Verification is deferred until later. * * This converts symbolic references into pointers. It’s independent of * the source file format. * * If “classesResolved” is false, we assume that superclassIdx and * interfaces[] are holding class reference indices rather than pointers. * The class references will be resolved during link. (This is done when * loading from DEX to avoid having to create additional storage to pass * the indices around.) * * Returns “false” with an exception pending on failure. */ bool dvmLinkClass(ClassObject* clazz, bool classesResolved) { u4 superclassIdx = 0; bool okay = false; bool resolve_okay; int numInterfacesResolved = 0; int i; if (gDvm.verboseClass) LOGV(“CLASS: linking ‘%s’...\n”, clazz->descriptor); /* “Resolve” the class. * * At this point, clazz’s reference fields contain Dex * file indices instead of direct object references. * We need to translate those indices into real references, * while making sure that the GC doesn’t sweep any of * the referenced objects. * * The GC will avoid scanning this object as long as * clazz->obj.clazz is gDvm.unlinkedJavaLangClass. * Once clazz is ready, we’ll replace clazz->obj.clazz * with gDvm.classJavaLangClass to let the GC know * to look at it. */ assert(clazz->obj.clazz == gDvm.unlinkedJavaLangClass); /* It’s important that we take care of java.lang.Class * first. If we were to do this after looking up the * superclass (below), Class wouldn’t be ready when * java.lang.Object needed it. * * Note that we don’t set clazz->obj.clazz yet. */ if (gDvm.classJavaLangClass == NULL) { if (clazz->classLoader == NULL && strcmp(clazz->descriptor, “Ljava/lang/Class;”) == 0) {*

The ’720 Patent	Infringed By
	gDvm.classJavaLangClass = clazz; *} else {* *gDvm.classJavaLangClass =* *dvmFindSystemClassNoInit(“Ljava/lang/Class;”);* *if (gDvm.classJavaLangClass == NULL) {* */* should have thrown one */* *assert(dvmCheckException(dvmThreadSelf()));* *goto bail;* *}* *}* *}* *assert(gDvm.classJavaLangClass != NULL);* */** ** Resolve all Dex indices so we can hand the ClassObject* ** over to the GC. If we fail at any point, we need to remove* ** any tracked references to avoid leaking memory.* **/* */** ** All classes have a direct superclass, except for java/lang/Object.* **/* *if (!classesResolved) {* *superclassIdx = (u4) clazz->super; /* unpack temp store */* *clazz->super = NULL;* *}* *if (strcmp(clazz->descriptor, “Ljava/lang/Object;”) == 0) {* *assert(!classesResolved);* *if (superclassIdx != kDexNoIndex) {* */* TODO: is this invariant true for all java/lang/Objects,* ** regardless of the class loader? For now, assume it is.* **/* *dvmThrowException(“Ljava/lang/ClassFormatError;”,* *“java.lang.Object has a superclass”);* *goto bail;* *}* */* Don’t finalize objects whose classes use the* ** default (empty) Object.finalize().* **/* *CLEAR_CLASS_FLAG(clazz, CLASS_ISFINALIZABLE);* *} else {* *if (!classesResolved) {* *if (superclassIdx == kDexNoIndex) {* *dvmThrowException(“Ljava/lang/LinkageError;”,* *“no superclass defined”);* *goto bail;* *}* *clazz->super = dvmResolveClass(clazz, superclassIdx, false);* *if (clazz->super == NULL) {* *assert(dvmCheckException(dvmThreadSelf()));* *if (gDvm.optimizing) {* */* happens with “external” libs */* *LOGV(“Unable to resolve superclass of %s (%d)\n”,* *clazz->descriptor, superclassIdx);* *} else {* *LOGW(“Unable to resolve superclass of %s (%d)\n”,*

The ’720 Patent	Infringed By
	clazz->descriptor, superclassIdx); *}* *goto bail;* *}* *}* … *}*
5. A system according to claim 1, further comprising: at least one of a local and remote file system to maintain the source definition as a class file.	Android includes at least one of a local and remote file system to maintain a source definition as a class file. *See*  (Dalvik Presentation, Slide 13) Corresponding Dalvik Video at 6:39: “And then towards the bottom there are a series of class definitions. So a dex file contains multiple classes….” *See, e.g.*, dalvik\vm\analysis\DexOptimize.c. */** ** Return the fd of an open file in the DEX file cache area. If the cache* ** file doesn’t exist or is out of date, this will remove the old entry,* ** create a new one (writing only the file header), and return with the* ** ”new file” flag set.* *** … ** On success, the file descriptor will be positioned just past the ”opt”* ** file header, and will be locked with flock. ”*pCachedName” will point* ** to newly-allocated storage.* **/* *int dvmOpenCachedDexFile(const char* fileName, const char* cacheFileName,u4 modWhen, u4 crc, bool isBootstrap, bool* pNewFile, bool createIfMissing)* *{* *int fd, cc;* *struct stat fdStat, fileStat;* *bool readOnly = false;* **pNewFile = false;* *retry:* */**

The ’720 Patent	Infringed By
	** Try to open the cache file. If we’ve been asked to,* ** create it if it doesn’t exist.* **/* *fd = createIfMissing ? open(cacheFileName, O_CREAT|O_RDWR, 0644) : -1;* *if (fd < 0) {* *fd = open(cacheFileName, O_RDONLY, 0);* *if (fd < 0) {* *if (createIfMissing) {* *LOGE(“Can’t open dex cache ’%s’: %s\n”,* *cacheFileName, strerror(errno));* *}* *return fd;* *}* *readOnly = true;* *}* … *}*
6. A system according to claim 1, further comprising: a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.	Android includes a process cloning mechanism to instantiate a child runtime system process by copying the memory space of a master runtime system process into a separate memory space for the child runtime system process. *See*  (Android Presentation, Slide 82) Corresponding Android Video at 44:30: “The init process starts up a really neat process called zygote….It uses copy-on-write to maximize re-use and minimize footprint so that data structures are shared and it won’t do a full copy unless some of those data structures are to be modified.” *See also*

The ’720 Patent	Infringed By
	 (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: “What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this.”  (Dalvik Presentation, Slide 26) Corresponding Dalvik Video at 14:40: “So the zygote, again, has made, has made this heap of objects, it’s made this live dex structure and then each application that then starts up, instead of having its own memory for those things, it just shares it with the zygote and also with any other app that’s also on the system.” *See also* http://developer.android.com/guide/basics/what-is-android.html. “Android Runtime …The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Linux Kernel Android relies on Linux version 2.6 for core system services such as

The ’720 Patent	Infringed By
	security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.”

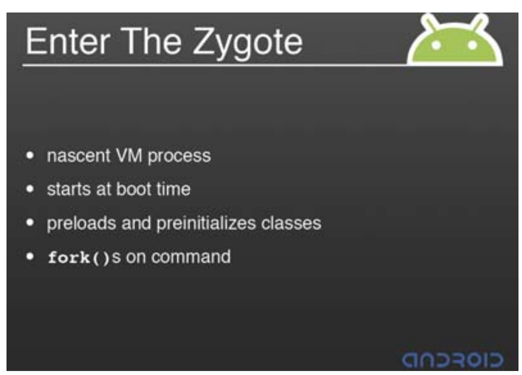
See also, Lowe, Robert, Linux Kernel Process Management, April 15, 2005. Sample Chapter is provided courtesy of Sams, http://www.informit.com/articles/article.aspx?p=370047&seqNum=2&rll=1.
“Copy-on-Write
…In Linux, fork() is implemented through the use of copy-on-write pages. Copy-on-write (or COW) is a technique to delay or altogether prevent copying of the data. Rather than duplicate the process address space, the parent and the child can share a single copy. The data, however, is marked in such a way that if it is written to, a duplicate is made and each process receives a unique copy.”

Example source code files in
libcore\dalvik\src\main\java\dalvik\system\Zygote.java,
dalvik\vm\native\dalvik_system_Zygote.c,
linux-2.6\kernel\fork.c.

See, e.g., libcore\dalvik\src\main\java\dalvik\system\Zygote.java.

```
/**
 * Forks a new Zygote instance, but does not leave the zygote mode.
 * The current VM must have been started with the -Xzygote flag. The
 * new child is expected to eventually call forkAndSpecialize()
 *
 * @return 0 if this is the child, pid of the child
 * if this is the parent, or -1 on error
 */
native public static int fork();

/**
 * Forks a new VM instance.  The current VM must have been started
 * with the -Xzygote flag. <b>NOTE: new instance keeps all
 * root capabilities. The new process is expected to call capset()</b>.
 *
 * @param uid the UNIX uid that the new process should setuid() to after
 * fork()ing and and before spawning any threads.
 * @param gid the UNIX gid that the new process should setgid() to after
 * fork()ing and and before spawning any threads.
 * @param gids null-ok; a list of UNIX gids that the new process should
 * setgroups() to after fork and before spawning any threads.
 * @param debugFlags bit flags that enable debugging features.
 * @param rlimits null-ok an array of rlimit tuples, with the second
```

The '720 Patent	Infringed By
	* dimension having a length of 3 and representing * (resource, rlim_cur, rlim_max). These are set via the posix * setrlimit(2) call. * * @return 0 if this is the child, pid of the child * if this is the parent, or -1 on error. */ native public static int forkAndSpecialize(int uid, int gid, int[] gids, int debugFlags, int[][] rlimits); *See*, *e.g.*, dalvik\vm\native\dalvik_system_Zygote.c. /* native public static int forkAndSpecialize(int uid, int gid, * int[] gids, int debugFlags); */ static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args, JValue* pResult) { pid_t pid; pid = forkAndSpecializeCommon(args); RETURN_INT(pid); } … /* * Utility routine to fork zygote and specialize the child process. */ static pid_t forkAndSpecializeCommon(const u4* args) { pid_t pid; uid_t uid = (uid_t) args[0]; gid_t gid = (gid_t) args[1]; ArrayObject* gids = (ArrayObject *)args[2]; u4 debugFlags = args[3]; ArrayObject *rlimits = (ArrayObject *)args[4]; if (!gDvm.zygote) { dvmThrowException("Ljava/lang/IllegalStateException;", "VM instance not started with -Xzygote"); return -1; } if (!dvmGcPreZygoteFork()) { LOGE("pre-fork heap failed\n"); dvmAbort(); } setSignalHandler(); dvmDumpLoaderStats("zygote"); pid = fork(); if (pid == 0) { int err; /* The child process */ …. } else if (pid > 0) { /* the parent process */ }

The ’720 Patent	Infringed By
	return pid; } *See*, *e.g.*, linux-2.6\kernel\fork.c. /* * Ok, this is the main fork-routine. * * It copies the process, and if successful kick-starts * it and waits for it to finish using the VM if required. */ long do_fork(unsigned long clone_flags, unsigned long stack_start, struct pt_regs *regs, unsigned long stack_size, int __user *parent_tidptr, int __user *child_tidptr) { struct task_struct *p; int trace = 0; long nr; … p = copy_process(clone_flags, stack_start, regs, stack_size, wake_up_new_task(p, clone_flags); … tracehook_report_clone_complete(trace, regs, clone_flags, nr, p); … return nr; }
7. A system according to claim 1, wherein the master runtime system process is caused to sleep relative to receiving the process request.	Android includes a master runtime system process that is caused to sleep relative to receiving a process request. *See*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: “What we do with the zygote, as its name implies, …it sort of sits on a

The ’720 Patent	Infringed By
	socket and it waits for commands….” *See also*  (Android Presentation, Slide 82) Corresponding Android Video at 44:25: “The init process starts up a really neat process called zygote….And so zygote is a nascent VM process that initializes a Dalvik VM and preloads a lot of its libraries and it forks on request to create new VM instances for managed processes….” *See, e.g.*, base\core\java\com\android\internal\os\ZygoteConnection.java. */** * Constructs instance from connected socket. * * @param socket non-null; connected socket * @throws IOException */ ZygoteConnection(LocalSocket socket) throws IOException { mSocket = socket; mSocketOutStream = new DataOutputStream(socket.getOutputStream()); mSocketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()), 256); mSocket.setSoTimeout(CONNECTION_TIMEOUT_MILLIS); try { peer = mSocket.getPeerCredentials(); } catch (IOException ex) { Log.e(TAG, “Cannot read peer credentials”, ex); throw ex; } } /** * Returns the file descriptor of the associated socket. * * @return null-ok; file descriptor*

The ’720 Patent	Infringed By
	*/ FileDescriptor getFileDesciptor() { return mSocket.getFileDescriptor(); } /** * Reads start commands from an open command socket. * Start commands are presently a pair of newline-delimited lines * indicating a) class to invoke main() on b) nice name to set argv[0] to. * Continues to read commands and forkAndSpecialize children until * the socket is closed. This method is used in ZYGOTE_FORK_MODE * * @throws ZygoteInit.MethodAndArgsCaller trampoline to invoke main() * method in child process */ void run() throws ZygoteInit.MethodAndArgsCaller { int loopCount = ZygoteInit.GC_LOOP_COUNT; while (true) { … if (runOnce()) { break; } } } /** * Reads one start command from the command socket. If successful, * a child is forked and a {@link ZygoteInit.MethodAndArgsCaller} * exception is thrown in that child while in the parent process, * the method returns normally. On failure, the child is not * spawned and messages are printed to the log and stderr. Returns * a boolean status value indicating whether an end-of-file on the command * socket has been encountered. * * @return false if command socket should continue to be read from, or * true if an end-of-file has been encountered. * @throws ZygoteInit.MethodAndArgsCaller trampoline to invoke main() * method in child process */ boolean runOnce() throws ZygoteInit.MethodAndArgsCaller { String args[]; Arguments parsedArgs = null; FileDescriptor[] descriptors; try { args = readArgumentList(); descriptors = mSocket.getAncillaryFileDescriptors(); } catch (IOException ex) { Log.w(TAG, “IOException on command socket “ + ex.getMessage()); closeSocket(); return true; } if (args == null) { // EOF reached. closeSocket();

The ’720 Patent	Infringed By
	return true; *}* **…** *int pid;* **…** *pid = Zygote.forkAndSpecialize(parsedArgs.uid, parsedArgs.gid,* *parsedArgs.gids, parsedArgs.debugFlags, rlimits);* *} catch (IllegalArgumentException ex) {* *logAndPrintError (newStderr, “Invalid zygote arguments”, ex);* *pid = -1;* *} catch (ZygoteSecurityException ex) {* *logAndPrintError(newStderr,* *“Zygote security policy prevents request: “, ex);* *pid = -1;* *}* *if (pid == 0) {* *// in child* *handleChildProc(parsedArgs, descriptors, newStderr);* *// should never happen* *return true;* *} else { /* pid != 0 */* *// in parent...pid of < 0 means failure* *return handleParentProc(pid, descriptors, parsedArgs);* *}* *}* **…** */*** ** Reads an argument list from the command socket/* ** @return Argument list or null if EOF is reached* ** @throws IOException passed straight through* **/* *private String[] readArgumentList()* *throws IOException {* */*** ** See android.os.Process.zygoteSendArgsAndGetPid()* ** Presently the wire format to the zygote process is:* ** a) a count of arguments (argc, in essence)* ** b) a number of newline-separated argument strings equal to count* *** ** After the zygote process reads these it will write the pid of* ** the child or -1 on failure.* **/* *int argc;* *try {* *String s = mSocketReader.readLine();* *if (s == null) {* *// EOF reached.* *return null;* *}* *argc = Integer.parseInt(s);* *} catch (NumberFormatException ex) {* *Log.e(TAG, “invalid Zygote wire format: non-int at argc”);* *throw new IOException(“invalid wire format”);* *}* **…**

The ’720 Patent	Infringed By
	String[] result = new String[argc]; *for (int i = 0; i < argc; i++) {* *result[i] = mSocketReader.readLine();* *if (result[i] == null) {* *// We got an unexpected EOF.* *throw new IOException(“truncated request”);* *}* *}* *return result;* *}*
8. A system according to claim 1, wherein the object-oriented program code is written in the Java programming language.	Android includes object-oriented program code that is written in the Java programming language. *See* Google I/O 2010 Video, entitled “A JIT Compiler for Android’s Dalvik VM,” presented by Ben Cheng and Bill Buzbee (Google’s Android Team), available at http://developer.android.com/videos/index.html#v=Ls0tM-c4Vfo (“JIT Video”) at time 1:58. “Now, if you are going to write a program for Android, you are most likely going to write it in the Java programming language and then push the source code through the SDK. And what pops out at the end is an executable targeted to the Dalvik virtual machine.” *See also* http://developer.android.com/guide/basics/what-is-android.html. “What is Android? Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.”
10.pre. A method for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*See* claim 1.pre.
10.a. executing a master runtime system process;	*See* claim 1.c.
10.b. obtaining a representation of at least one class from a source definition provided	*See* claim 1.c.

The ’720 Patent	Infringed By
as object-oriented program code;	
10.c. interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process; and	*See* claim 1.d.
10.d. cloning the memory space as a child runtime system process responsive to a process request and executing the child runtime system process;	*See* claim 1.e.
10.e. wherein cloning the memory space as a child runtime system process involves instantiating the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process; and	*See* claim 1.f.
10.f. wherein copying references to the memory space of the master runtime system process defers copying of the	*See* claim 1.f.

The ’720 Patent	Infringed By
memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	
11. A method according to claim 10, further comprising: determining whether the instantiated class definition is available in a local cache associated with the master runtime system process.	*See* claim 2.
12. A method according to claim 11, further comprising: locating the source definition if the instantiated class definition is unavailable in the local cache.	*See* claim 3.
13. A method according to claim 10, further comprising: resolving the class definition.	*See* claim 4.
14. A method according to claim 10, further comprising: maintaining the	*See* claim 5.

The ’720 Patent	Infringed By
source definition as a class file on at least one of a local and remote file system.	
15. A method according to claim 10, further comprising: instantiating the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.	*See* claim 6.
16. A method according to claim 10, further comprising: causing the master runtime system process to sleep relative to receiving the process request.	*See* claim 7.
17. A method according to claim 10, wherein the object-oriented program code is written in the Java programming language.	*See* claim 8.
19. A computer-readable storage medium holding code for performing the method according to claim 10.	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices.

The ’720 Patent	Infringed By
20.pre. An apparatus for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*See* claim 1.pre.
20.a. A processor;	*See* claim 1.a.
20.b. A memory means for executing a master runtime system process;	*See* claim 1.b.
20.c. means for obtaining a representation of at least one class from a source definition provided as object-oriented program code;	*See* claim 1.c. *See also, e.g.*, ’720 patent, 6:46-54, FIGs. 2, 10: “A set of core Java foundation classes is specified in a bootstrap class loader 39 and application classes in a system application class loader 40. Class loading requires identifying a binary form of a class type as identified by specific name, as further described below with reference to FIG. 10. Depending upon whether the class was previously loaded or referenced, class loading can include retrieving a binary representation from source and constructing a class object to represent the class in memory.”
20.d. means for interpreting and means for instantiating the representation as a class definition in a memory space of the master runtime system process; and	*See* claim 1.d. *See also, e.g.*, ’720 patent, 6:61-67, FIG. 2: “The master JVM process 33 invokes the bootstrap class loader 39 and system application class loader 40 for every class likely to be requested by the applications. Thus, the prewarmed state 41 includes the class loading for applications prior to actual execution and the initialized and loaded classes are inherited by each cloned JVM process 34 as the inherited prewarmed state 42.”
20.e. means for cloning the memory space as a child runtime system process responsive to a process request and means for	*See* claim 1.e. *See also, e.g.*, ’720 patent, 5:33-37, FIG. 2: “The runtime environment 31 executes an application framework that spawns multiple independent and isolated user application process instances by preferably cloning the memory space of a master runtime system process.”

The ’720 Patent	Infringed By
executing the child runtime system process;	
20.f. wherein the means for cloning the memory space is configured to clone the memory space of a child runtime system process using a copy-on-write process cloning mechanism that instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process and that defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	*See* claim 1.f. *See also, e.g.*, ’720 patent, 6:12-19, FIGs. 2, 5A, 5B: “When implemented with copy-on-write semantics, the process cloning creates a logical copy of only the references to the master JVM process context. Segments of the referenced master JVM process context are lazily copied only upon an attempt by the cloned JVM process to modify the referenced context. Therefore as long as the cloned JVM process does not write into a memory segment, the segment remains shared between parent and child processes.”
21. A system according to claim 1, further comprising: a resource controller to set operating system level	Android includes a resource controller to set operating system level resource management parameters on the child runtime system process. *See, e.g.*, libcore\dalvik\src\main\java\dalvik\system\Zygote.java. */*** ** Forks a new VM instance. The current VM must have been started* ** with the -Xzygote flag. NOTE: new instance keeps all*

The ’720 Patent	Infringed By
resource management parameters on the child runtime system process.	* root capabilities. The new process is expected to call capset(). * * @param uid the UNIX uid that the new process should setuid() to after * fork()ing and and before spawning any threads. * @param gid the UNIX gid that the new process should setgid() to after * fork()ing and and before spawning any threads. * @param gids null-ok; a list of UNIX gids that the new process should * setgroups() to after fork and before spawning any threads. * @param debugFlags bit flags that enable debugging features. * @param rlimits null-ok an array of rlimit tuples, with the second * dimension having a length of 3 and representing * (resource, rlim_cur, rlim_max). These are set via the posix * setrlimit(2) call. * * @return 0 if this is the child, pid of the child * if this is the parent, or -1 on error. */ native public static int forkAndSpecialize(int uid, int gid, int[] gids, int debugFlags, int[][] rlimits);
22. A method according to claim 10, further comprising: setting operating system level resource management parameters on the child runtime system process.	*See* claim 21.