UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT TRIAL AND APPEAL BOARD

Capital One, National Association; Hulu, LLC
Petitioner

v.

Implicit, LLC,
Patent Owner

CASE: IPR2024-00787
U.S. PATENT NO. 6,976,248

DECLARATION OF EREZ ZADOK, PH.D., IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 6,976,248

TABLE OF CONTENTS

I. Introduction....1
II. Qualifications....3
III. Understanding of Relevant Legal Principles....13
A. Claim Construction....14
B. Obviousness....15
IV. Technical Background....18
A. General Computer Operations....18
B. Web Technologies....24
C. Caching Technologies....35
D. Programming Languages, Compilers and Interpreters, Optimization....44
E. Transforming Data and Files....52
F. Intermediate Compiler Stages....56
V. U.S. Patent No. 6,976,248....60
A. Summary of the '248 Patent....61
B. Prosecution History....63
C. Person of Ordinary Skill in the Art....66
VI. Overview of the Prior Art....67
A. Johnson (EX1011)....67
B. Parthasarathy (EX1007)....73
C. Kimera (EX1008)....77
D. Fowlow (EX1009)....81
VII. Overview of the Claims....86
VIII. The Challenge Grounds....89
A. Ground 1: Claims 1 and 9 are Obvious in view of Johnson....90
1. Independent Claim 1....90

a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:.....90

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 95

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and 96

d. [1c] c) transmitting said applet in said selected form to said client computer.....100

2. Independent Claim 9 101

a. [9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising: 101

b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 102

c. [9b] b) compiling an applet program module into an intermediate form program module;.....102

d. [9c] c) transmitting said applet in said selected form to said client computer.....104

B. Ground 2: Claims 1-13 are Unpatentable Under 35 U.S.C. §103(a) as Obvious over Parthasarathy in view of Kimera.....104

1. Motivation to Combine Parthasarathy and Kimera 104

2. Independent Claim 1 112

a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an

applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:.....112

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 118

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and 120

d. [1c] c) transmitting said applet in said selected form to said client computer. 122

3. Dependent Claims 2-4.....123

a. 2. The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.....123

b. 3. The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local esource, and directing said external resource to transmit said applet in said selected form to said server computer. 125

c. 4. The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to tansmit said applet in said selected form to said server computer.....130

4. Independent Claim 5 131

a. [5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprisig:.....131

b. [5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms; 131

c. [5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer; 132

d. [5c] c) if said applet is not stored in said selected form in said local cache, ...133

e. [5d] compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and 134

f. [5e] transmitting said applet in said selected form to said client computer. 134

5. Dependent Claims 6-8.....135

a. 6. The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.....135

b. 7. The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local esource, and directing said external resource to transmit said applet in said form to said server computer. 135

c. 8. The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local esources, and directing said external resource to transmit said applet in said selected form to said server computer. 136

6. Independent Claim 9 136

a. [9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising: 136

b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 137

c. [9b] b) compiling an applet program module into an intermediate form program module;.....137

d. [9c] c) transmitting said applet in said selected form to said client computer. 138

7. Dependent Claims 10-13.....139

a. 10. The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program. 139

b. 11. The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring comuter programs and encrypting computer programs. 142

c. 12. The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler. 145

d. 13. The method of claim 10, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler. 147

C. Ground 3: Claims 1-13 are Unpatentable Under 35 U.S.C. §103(a) as Obvious over Fowlow in view of Kimera 147

1. Motivation to Combine 147

2. Independent Claim 1 156

a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:.....156

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 159

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and 161

d. [1c] c) transmitting said applet in said selected form to said client computer. 168

3. Dependent Claims 2-4.....170

a. 2. The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.....170

b. 3. The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local esource, and directing said external resource to transmit said applet in said selected form to said server computer. 171

c. 4. The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to tansmit said applet in said selected form to said server computer.....175

4. Independent Claim 5 176

a. [5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:.....176

b. [5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms; 177

c. [5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer; 177

d. [5c] c) if said applet is not stored in said selected form in said local cache,.....179

e. [5d] compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and 181

f. [5e] transmitting said applet in said selected form to said client computer. 181

5. Dependent Claims 6-8.....181

a. 6. The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.....181

b. 7. The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local esource, and directing said external resource to transmit said applet in said form to said server computer.....182

c. 8. The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not

stored in said local cache and said applet can not be compiled from said local esources, and directing said external resource to transmit said applet in said selected form to said server computer. 182

6. Independent Claim 9 182

   a. [9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising: 182

   b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms; 183

   c. [9b] b) compiling an applet program module into an intermediate form program module;.....183

   d. [9c] c) transmitting said applet in said selected form to said client computer. 184

7. Dependent Claims 10-13.....185

   a. 10. The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.....185

   b. 11. The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring computer programs and encrypting computer programs. 189

   c. 12. The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler. 192

   d. 13. The method of claim 10, further comprising the step of: compiling said transformed intermediate form

program module into machine specific binary code with a target compiler. 193

IX. Conclusion 193

LIST OF EXHIBITS

Exhibit No.	Description
1001	U.S. Patent No. 6,976,248 (the "'248 Patent")
1002	Declaration of Erez Zadok, Ph.D. (this document).
1003	Curriculum Vitae of Erez Zadok, Ph.D.
1004	Prosecution History of U.S. Patent Application No. 11/241,985
1005	PCT App. No. WO 1998/006033 to Agranat et al. ("Agranat")
1006	Michael Franz & Thomas Kistler, Slim Binaries, Communications of the ACM, Dec. 1997 ("Franz")
1007	U.S. Pat. No. 6,347,398 to Parthasarathy et al. ("Parthasarathy")
1008	Emin Gün Sirer & Sean McDirmid & Brian Bershad, Kimera: A System Architecture for Networked Computers, Univ. of Wash., 1997 ("Kimera")
1009	U.S. Patent No. 6,260,078 to Fowlow ("Fowlow")
1010	U.S. Patent No. 5,933,811 to Angles et al. ("Angles")
1011	U.S. Patent No. 5,923,885 to Johnson ("Johnson")
1012	Affidavit of Nathanial E. Frank-White, dated 2/9/2024
1013	Prosecution History of U.S. Patent Application No. 08/764,040
1014	Microsoft Computer Dictionary, 3rd ed., 1997
1015	Hypertext Transfer Protocol – HTTP/1.1, RFC 2068, The Internet Engineering Task Force (IETF), January 1997. [Download from https://www.rfc-editor.org/rfc/pdfrfc/rfc2068.txt.pdf]

Exhibit No.	Description
1016	HTML 4.0 Specification, The World Wide Web Consortium (W3C) Recommendation, December 18, 1997. [Download from https://www.w3.org/TR/REC-html40-971218/html40.pdf]
1017	HTTP State Management Mechanism, RFC 2109, The Internet Engineering Task Force (IETF), February 1997. [Download from https://www.rfc-editor.org/rfc/pdfrfc/rfc2109.txt.pdf]
1018	Abraham Silberschatz and Peter B. Galvin, Operating Systems Concepts, 4th Edition, 1994, Addison-Wesley Publishing
1019	Application of Internet Cache Protocol (ICP), version 2. RFC-2187. September 1997. [https://www.rfc-editor.org/rfc/pdfrfc/rfc2187.txt.pdf]
1020	GNU C Manual Page documentation, 1993
1021	Persistent Client State HTTP Cookies, October 1996
1022	Original claims of U.S. Patent Application No. 11/933,161
1023	Non-final Rejection of U.S. Patent Application No. 11/933,161 (Feb. 14, 2011)
1024	Applicant Response to Non-final Rejection of U.S. Patent Application No. 11/933,161 (Aug. 11, 2011)

I. INTRODUCTION

1. I, Erez Zadok, have been retained as an independent technical expert on behalf of Petitioner Capital One National Association related to *Inter Partes* Review ("IPR") of U.S. Patent No. 6,976,248 ("the '248 Patent").

2. I was asked to review the following:

- U.S. Patent No. 6,976,248 to Balassanian titled "Application server facilitating with client's computer for applets along with various formats" ("the '248 Patent") (EX1001);
- Prosecution History of U.S. Patent Application No. 09/968,704 (EX1004);
- U.S. Patent No. 5,923,885 to Johnson ("Johnson") (EX1011);
- Non-final Rejection of U.S. Patent Application No. 11/933,161 (Feb. 14, 2011) (EX1022);
- Applicant Response to Non-final Rejection of U.S. Patent Application No. 11/933,161 (Aug. 11, 2011) (EX1023);
- U.S. Patent No. 6,347,398 to Parthasarathy et al. ("Parthasarathy") (EX1007);
- Emin Gün Sirer & Sean McDirmid & Brian Bershad, Kimera: A System Architecture for Networked Computers, Univ. of Wash., 1997 ("Kimera") (EX1008);
- U.S. Patent No. 6,260,078 to Fowlow et al. ("Fowlow") (EX1009); and

3. I have also reviewed and rely on any other documents cited and referenced herein, as well as my own personal knowledge and experience.

4. The opinions and comments formulated during this assessment are based on observations and information available at the time of the investigation. I am not offering any conclusions as to the ultimate determinations that I understand the Board will make in this proceeding. I am simply providing my opinion on the technical aspects of the documents.

5. In formulating my opinions, I have relied on my knowledge, training, and experience in the relevant field, which I will summarize below. A more detailed summary of my background, education, experience, and publications is set forth in my curriculum vitae (CV), which is submitted as EX1003.

6. I have personal knowledge of the facts and opinions set forth in this declaration and believe them to be true. If called upon to do so, I would testify competently thereto. I have been warned that willful false statements and the like are punishable by fine or imprisonment, or both.

7. My consulting company, Zadoks Consulting, LLC, is being compensated for my time at my standard consulting rate. I am also being reimbursed for expenses that I may incur during the course of this work. My compensation is not contingent upon the results of my study and analysis, the substance of my opinions, or the outcome of any proceeding involving the Challenged Claims. I have

no financial interest in the outcome of this matter or in any litigation involving the '248 patent.

II. QUALIFICATIONS

8. I am a Professor in the Computer Science Department at Stony Brook University (part of the State University of New York ("SUNY") system). I direct the File-systems and Storage Lab (FSL) at Stony Brook's Computer Science Department. My research interests include file systems and storage systems, operating systems, transactional systems including database technologies, information technology and system administration, security and information assurance, networking, energy efficiency, performance and benchmarking, virtualization, cloud systems, compilers, applied machine learning, and software engineering.

9. I studied at a professional high school in Israel, focusing on electrical engineering ("EE"), and graduated in 1982. I spent one more year at the high school's college division, receiving a special Certified Technician's degree in EE. I then went on to serve in the Israeli Defense Forces for three years (1983-1986). I received my Bachelor of Science degree in computer science ("CS") in 1991, my Master's degree in CS in 1994, and my PhD in CS in 2001—all from Columbia University in New York.

10. When I began my undergraduate studies at Columbia University, I also started working as a student assistant in the various campus-wide computer labs, eventually becoming an assistant to the head labs manager, who was managing all public computer labs on campus. During that time, I also became more involved with research within the CS Department at Columbia University, conducting research on operating systems, file and storage systems, distributed and networked systems, security, and other topics. I also assisted the CS department's computer administrators in managing the department's computers, which included storage, IT, networking, and cyber-security related duties.

11. In 1991, I joined Columbia University's CS department as a full-time systems administrator, studying towards my MS degree part-time. My MS thesis topic is related to file system reliability, fault tolerance, replication, and failover in mobile networked storage systems using file virtualization. My main duties as a systems administrator involved installing, configuring, and managing many networked servers, proxies, and desktops running several operating systems, as well as network devices setup; this included many software and hardware upgrades, device upgrades, and BIOS firmware/chipset updates/upgrades. My duties also included ensuring reliable, secure, authenticated access to networked systems/storage and licensed software, as well as software updates, security and bug fixes. Examples of servers and their protocols included email (SMTP), file transfer

(FTP), domain names (DNS), network file systems (NFS), network news systems (NNTP), and Web (HTTP).

12. In 1994, I left my systems administrator position to pursue my doctoral studies at Columbia University. My PhD thesis topic was on versatile file system development using stackable (virtualized) file systems, with examples in the fields of security and encryption, efficiency, reliability, and failover. I continued to work part-time as a systems administrator at the CS department, and eventually I was asked to serve as manager to the entire information technology ("IT") staff. From 1991 to 2001, I was also a member of the faculty-level Facilities Committee that oversaw all IT operations at the CS department.

13. As part of my PhD studies at Columbia, I collaborated on projects to develop advanced AI-like techniques to detect previously unknown viruses (a.k.a. "zero-day malware"), using data mining and rule-based detection. This work led to several highly cited papers (over 1,600 citations for one of the papers alone), and two patents. I also became a Teaching Assistant (TA) for a first-ever Computer Security course given at Columbia University's CS department with Dr. Matt Blaze as instructor.

14. From 1990 to 1998, I consulted for SOS Corporation and HydraWEB Technologies, as a systems administrator and programmer, managing data storage use and backup/restore duties, databases, web servers, as well as information

assurance and cyber-security (*e.g.*, malware protection, software licensing). From 1994 to 2000, I led projects at HydraWEB Technologies, and then became the Director of Software Development-overseeing the development of several products and appliances such as stateful firewalls and HTTP load-balancers, utilizing network-virtualization and high-availability techniques. From 2009 to 2019, I have consulted for Packet General Networks, a startup specializing in secure, virtualized, network storage and applications' data security in the cloud.

15. In 2001, I joined the faculty of Stony Brook University, a position I have held since that time. In 2002, I joined the Operations Committee, which oversees the IT operations of the CS department at Stony Brook University. From 2006 to 2010, I was the Director of IT Operations of the CS department; my day-to-day duties included setting policies regarding computing, hiring and training new staff, assisting any staff with topics of my specialty, defining requirements for new software/hardware, and purchasing. From 2010 to 2015, I had served as the Co-Chair to the Operations Committee. From 2016 to 2019, I oversaw the IT Operations as the Chair of the Operations Committee. A significant component of these duties included defining and helping implement policies for data management, so as to ensure the security of users and their data, and data reliability and availability, while minimizing the inconvenience and performance impact to users. I personally helped setup and maintain an initial virtual-host infrastructure in the department. Since late

2019, I've been a member of the department's Executive Committee that also oversees all IT operations.

16. In 2017, I became the department's Graduate Academic Adviser, advising all Master students (over 400 annually on average) and many other graduate students on an assortment of academic matters.

17. Since 2001, I have personally configured and managed my own research lab's network. This includes setting up and configuring multiple storage systems (*e.g.*, NFS, CIFS/SMB, NAS), virtual and physical environments, applications such as database (*e.g.*, MySQL, Postgresql), Web servers (*e.g.*, Apache), and mail servers; user access control (*e.g.*, NIS, LDAP), backups and restores, snapshot policies, and more. I've personally installed, configured, changed, replaced parts, and upgraded components in numerous devices including mobile devices, laptops, desktops, and servers, both physical and virtual.

18. Since 1995, I have taught courses on operating systems, storage and file systems, advanced systems programming in Unix/C, systems administration, data structures, data/software security, and more. My courses often use storage, file systems, distributed systems, and system/network security as key teaching principles and practical examples for assignments and projects. I have taught these concepts and techniques to my students, both to my direct advisees as well as in my courses. For example, in my graduate Operating Systems course, I often cover Linux's kernel

mechanisms to protect users, applications, and data files, virtual file systems, as well as distributed storage systems (*e.g.*, NFS). And in the System Administration undergraduate course, I covered many topics such as networking, storage, backups, and configuring complex applications such as mail, web, and database servers.

19. My research often investigates computer systems from many angles: security, efficiency, energy use, scalability, reliability, portability, survivability, usability, ease-of-use, versatility, flexibility, and more. My research gives special attention to balancing five often-conflicting aspects of computer systems: performance, reliability, energy use, security, and ease-of-use.

20. Since joining Stony Brook University in 2001, my group in the File-systems and Storage Lab (FSL) has developed many file systems and operating system extensions; examples include a highly-secure cryptographic file system, a portable copy-on-write (COW) versioning file system, a tracing file system useful to detect intrusions, a replaying file system useful for forensics, a snapshotting and sandboxing file system, a namespace unification file system (that uses stackable, virtualized, file-based COW), an anti-virus file system, an integrity-checking file system, a load balancing and replication/mirroring file system, network file system extensions for security and performance, distributed secure cloud-based storage systems, transactional key-value stores and file systems, OS-level embedded databases, a compiler to convert user-level C code to in-kernel efficient yet safe

code, GCC plugins, stackable file system templates, and a Web-based backup system. Many of these projects used one form of virtualization or another (storage, network, host, etc.). I continue to maintain and release newer versions of some of these file systems and software.

21. I have published over 120 refereed publications (in ACM, IEEE, USENIX, and more). To date, my publications have been cited more than 10,000 times (as per Google Scholar as of February 18, 2024). My papers cover a wide range of related technologies such as file systems, storage systems, transactional systems, security, clouds and virtualization, performance benchmarking and optimization, energy efficiency, system administration, web systems, and more. I also published a book titled "Linux NFS and Automounter Administration" (Sybex, 2001), covering systems administration topics related to network storage and data security.

22. Some of my research has led to public software releases that have been used worldwide. I have publicly maintained the Amd Berkeley Automounter in a package called "am-utils" since 1992; this software helps administrators manage the multitude of file system mounts on dozens of different Unix systems, especially helping to automate access to multiple NFS/NAS storage volumes. Since 1997, I have maintained and released several stackable (virtualized) file system software projects for Linux, FreeBSD, and/or Solaris, in a package called FiST. One of my

stackable file system encryption projects, called Cryptfs, became the basis for IBM's public release of eCryptfs, now part of Linux. Packet General Networks, for whom I have provided consulting services between 2009 and 2019, licensed another encryption file system called Ncryptfs. Another popular file system released in 2003, called Unionfs, offers virtual namespace unification, transparent shadow copying (a.k.a. copy-on-write or COW), file system snapshotting (*e.g.*, useful for forensics and disaster recovery), and the ability to save disk space by sharing a read-only copy of data among several computers, among other features.

23. My research and teaching make extensive use of data security features. For example, each time I taught the graduate operating system course, the first homework assignment includes the creation of a new system call that performs new or added functionality, often for encrypting a file or verifying its integrity; many of my other assignments cover topics of user/process access control, anti-virus filtering, and more. Since 2001, over 1,000 graduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

24. Moreover, in an undergraduate course titled "Advanced Systems Programming in Unix/C," I cover many topics of system security and vulnerabilities, such as the structure of UNIX processes, and memory segments such as the heap and stack. This course covers details of several hundred Linux system calls. Often, the first assignment for this course is to develop a tool to encrypt/decrypt files using

advanced ciphers, use digital signatures to certify the cipher keys used, and reliably recover files in case of failures. Since 2001, several hundred undergraduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

25. In another undergraduate course, System Administration, I taught network configuration, security, and storage configuration and reliability. In a special topics course on Storage Systems, I covered many topics such as data deduplication, RAID, transactional storage, storage hardware including modern Flash based ones, virtual storage, backup/restore, snapshots and continuous data protection (CDP), NAS and SAN, and NFS.

26. Overall, in addition to the aforementioned experience, my technical experience relevant to the challenged patent(s) at the time of the alleged invention(s) included the following: I developed, configured, and maintained Web servers, Web firewalls and load balancers, CGI scripts and other dynamic Web code (*e.g.*, C, Perl) with backend databases; configured and used Web proxies such as Squid; written, developed, debugged, and/or maintained code in various programming languages (*e.g.*, C, C++, Lisp/Scheme, Java, and scripting languages (*e.g.*, sh, ksh, csh, bash, Perl, PHP); used, developed, studied, and optimized compiled code (*e.g.*, cc, gcc, g++); debugged and traced programs and code (*e.g.*, dbx, gdb, strace); used and developed compression software and file systems (*e.g.*, gzip, compress, bzip2, zip,

gzipfs, Deflate algorithm); used and developed software for encryption, integrity checking, and digital signatures (*e.g.*, cryptfs, DES, MD5, SHA); used logging and tracing of programs, network packets, and storage systems; written code and used systems to debug memory errors in programs and operating systems (*e.g.*, buffer overflow/underflow, pointer errors), type safety errors, and more; developed code on multiple operating systems: Windows (*e.g.*, NT, 95, 98) and Unix variants (*e.g.*, SunOS, Solaris, IBM AIX, SGI Irix, HPUX, Linux, DEC Ultrix and OSF/1, and many more), on multiple architectures (*e.g.*, SPARC, Intel, PPC, Alpha); and studied, developed, and maintained caching systems (including inside operating system kernels).

27. My research has been supported by many federal and state grants as well as industry awards, including an NSF CAREER award, two IBM Faculty awards, two NetApp Faculty awards, a Western Digital award, a Facebook award, several Dell-EMC awards, and several equipment gifts. I received the 2008 SUNY Chancellor's Excellence in Teaching award, and the 2022 SUNY Chancellor's Award for Excellence in Scholarship and Creative Activities (both awards can be given only once in a lifetime). In 2021 I was named an ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

28. My service record to the community includes serving as the co-chair for the USENIX Annual Technical Conference in 2020 (ATC'20); serving as the co-

chair for USENIX File and Storage Technologies (FAST'15) in 2015 and on the FAST Conference Steering Committee from 2015 to 2023; serving on the ACM HotStorage Steering Committee since 2021; and serving as the co-chair in 2012 and on the Steering Committee of the ACM SYSTOR conference since 2012. I have served as an Associate Editor to the ACM Transactions on Storage (TOS) journal from 2009 to 2022; in 2022, I was named the Editor-in-Chief for ACM's TOS journal.

29. I am a named inventor on four patents, two titled "Systems and Methods for Detection of New Malicious Executables" (U.S. Patent No. 7,487,544, issued February 3, 2009; and U.S. Patent No. 7,979,907, issued July 12, 2011); and two more titled "Multi-Tier Caching," (U.S. Patent No. 9,355,109, issued May 31, 2016; and U.S. Patent 9,959,279, issued May 1, 2018).

30. I have been disclosed as a testifying expert in 18 cases (including *inter partes* review (IPR) proceedings) in the past four years. I have been deposed 13 times and testified in trial twice.

III. UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES

31. I am not a lawyer, and I will not provide any legal opinions. Although I am not a lawyer, I have been advised certain legal standards are to be applied by technical experts in forming opinions regarding the meaning and validity of patent claims.

A. Claim Construction

32. I understand that claim terms are generally given their ordinary and customary meaning, which is the meaning that the term would have to a person of ordinary skill in the art (referred to as a POSA) in question at the time of the invention, *i.e.*, as of the earliest priority date of the patent application. I further understand that the POSA is deemed to read the claim term not only in the context of the particular claim in which a claim term appears, but in the context of the entire patent, including the specification and file history.

33. I am informed by counsel that the patent specification, under the legal principles, has been described as the best guide to the meaning of a claim term, and is thus highly relevant to the interpretation of claim terms. I further understand that for terms that do not have a customary meaning within the art, the specification usually supplies the best context of understanding the meaning of those terms. I also understand that claim terms should be understood in the context of the claim as a whole.

34. I understand that the prosecution history can further inform the meaning of the claim language by demonstrating how the inventors understood the invention and whether the inventors limited the invention in the course of prosecution, making the claim scope narrower than it otherwise would be. Extrinsic evidence may also

be consulted in construing the claim terms, such as my experience and expert testimony.

35. I have not been asked to provide any specific definitions for any of the terms in the claims I have analyzed. If asked, I would undertake such an endeavor. Accordingly, I have treated each claim term as it would be understood to have its plain and ordinary meaning to a POSA, as outlined below.

36. I understand that some claims are independent, and that these claims are complete by themselves. Other claims refer to these independent claims and are "dependent" from those independent claims. The dependent claims include all the limitations of the claims on which they depend.

B. Obviousness

37. It is my understanding that a patent claim is invalid as obvious if all the claimed elements were known in the prior art, including the knowledge of a POSA, and a POSA would have combined known methods in a way that teaches the elements as claimed in the challenged patent, and the combination would have yielded nothing more than predictable results to a POSA.

38. Counsel has informed me that the petitioner has the burden in an IPR, such as this matter, to show invalidity by a preponderance of the evidence. Counsel has told me that a preponderance of the evidence means more likely than not.

39. Counsel also instructed me that an invalidity inquiry may involve assessing the motivation of a POSA to modify a prior art reference.

40. I understand that I am to perform the task referenced in the preceding paragraph without using "hindsight" reasoning. Instead, I was asked to consider the motivation to modify a particular prior art reference through the eyes of a POSA as of around March, 1998.

41. I understand that relevant considerations for a motivation to modify a prior art reference includes at least the following:

(a) Combining prior art elements according to known methods to yield predictable results;

(b) Simple substitution of one known element for another to obtain predictable results;

(c) Use of known techniques to improve similar devices, methods, or products in the same way;

(d) Applying a known technique to a known device, method, or product ready for improvement to yield predictable results;

(e) "Obvious to try," which is choosing from a finite number of identified, predictable solutions with a reasonable expectation of success;

(f) Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to a POSA; and

(g) Some teaching, suggestion, or motivation in the prior art that would have led a POSA to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

42. I have also been informed that certain factors called "secondary considerations" can impact an obviousness analysis. These factors can include (1) the invention's commercial success, (2) long felt but unresolved needs, (3) the failure of others, (4) skepticism by experts, (5) praise by others, (6) teaching away by others, (7) recognition of a problem, (8) copying of the invention by competitors, and (9) other relevant factors. I am not aware of any such "secondary considerations." However, should any be presented, I reserve my right to address those unknown factors.

43. I have been informed that, in considering obviousness, hindsight reasoning derived from the patent-at-issue may not be used.

44. I have kept these considerations in mind when offering the opinions below regarding combinability, as well as when interpreting the scope and content of the prior art.

IV. TECHNICAL BACKGROUND

A. General Computer Operations

45. Computers are hardware and software devices that execute instructions. The hardware often includes a central processing unit (CPU), typically called "processor," volatile memory (*e.g.*, RAM (random-access memory), or DRAM (dynamic random-access memory)), non-volatile (persistent) storage (*e.g.*, a hard disk or flash drive), and input/output (I/O) devices (*e.g.*, network interface card, mouse, keyboard, display). RAM, hard disks, and flash drives are considered, generally, "storage" or "memory" devices because they can store and hold bits of information (whether persistently or not). Computers can have multiple storage devices. Processors are coupled to main memory using a memory bus and to peripherals such as hard disks and network interface cards using one or more I/O busses.

46. Memory and storage devices have considerable differences in their speeds, capacities, and costs. Non-volatile storage devices such as hard disks have the largest capacity, are least expensive per gigabyte, and are slowest (typically operating at millisecond speeds). Volatile DRAM devices are smaller, more expensive, and faster than hard disks (typically operating at microsecond speeds). CPUs' own memory caches and operations are the smallest, most expensive, and

fastest (typically operating at nanosecond speeds). Overall, CPUs are about 1,000× faster than DRAM, which in turn is about 1,000× faster than hard disks.

47. Computer programs are often written in a high-level human-readable language (*e.g.*, C, C++, Java, Perl, PHP, Python), then translated using a compiler or script processor to machine instructions understood by the CPU (*e.g.*, Intel or AMD processor). Machine instructions—generally called "software"—are stored in files on persistent media, loaded into DRAM, and then loaded into the CPU where they can be executed. All modern computers operate in this manner, regardless of their purpose: a small laptop, a server in a data center, a mainframe computer, a personal workstation, a proxy device, a gateway, a firewall, a router, an appliance, a virtual machine, a handheld or mobile device, smart card, etc. Computer systems may run multiple software programs, from background servers and daemons to user applications.

48. Users (including administrators) can login to one or more computers (*e.g.*, desktop, laptop, server) and execute one or more applications (*e.g.*, word processing, web browsing, backup tool).

49. A Local Area Network (LAN) is a typically a network at a local site where network latencies are short and physical distances between computers are small. A LAN and its computers are often protected from the rest of the internet using a firewall. EX1014 (Microsoft Computer Dictionary) at 276 ("**LAN** *n*.

Acronym for local area network. A group of computers and other devices dispersed over a relatively limited area and connected by a communications link that enables any device to interact with any other on the network. LANs commonly include microcomputers and shared resources such as laser printers and large hard disks. The devices on a LAN are known as nodes, and the nodes are connected by cables through which messages are transmitted. *See also* baseband network, broadband network, bus network, collision detection, communications protocol, contention, CSMA/CD, network, ring network, star network, token bus network, token passing, token ring network. *Compare* wide area network.").

50. Another form of a local area network is one that operates inside a single computer. A "localhost" is "[t]he name that is used to represent the same computer on which a TCP/IP message originates. An IP packet sent to localhost has the IP address 127.0.0.1 and does not actually go out to the Internet. *See also* IP address, packet (definition 1), TCP/IP." EX1014 (Microsoft Computer Dictionary) at 287. Often, when users/processes want to communicate with other users/processes, using TCP/IP communications on the same machine, they can use the localhost address: it is considered a trusted, high-speed internal network.

51. Conversely, a Wide-Area Network (WAN) refers to computers across the entire Internet, who are distant from each other, exhibit longer latencies, and afford little protection. EX1014 (Microsoft Computer Dictionary) at 507 ("**wide**

area network *n*. A communications network that connects geographically separated areas. *Acronym*: WAN.")

52. A firewall typically intercepts all traffic between computers in a LAN and the WAN, monitors and controls access, and can filter out undesired traffic. Often, LAN computers are far more trusted than WAN computers; therefore, a firewall can be said to separate the trusted network (LAN or localhost) from the untrusted network (WAN). EX1014 (Microsoft Computer Dictionary) at 197 ("**firewall** *n*. A security system intended to protect an organization's network against external threats, such as hackers, coming from another network, such as the Internet. A firewall prevents computers in the organization's network from communicating directly with computers external to the network and vice versa. Instead, all communication is routed through a proxy server outside of the organization's network, and the proxy server decides whether it is safe to let a particular message or file pass through to the organization's network.").

53. A server is a computer or software that provides a service to other computers, called clients. There are many servers on the Internet, for example, Web servers and Mail servers. EX1014 (Microsoft Computer Dictionary) at 430 ("**server** *n*. 1. On a local area network (LAN), a computer running administrative software that controls access to the network and its resources, such as printers and disk drives, and provides resources to computers functioning as workstations on the network. 2.

On the Internet or other network, a computer or program that responds to commands from a client. For example, a file server may contain an archive of data or program files; when a client submits a request for a file, the server transfers a copy of the file to the client. *See also* client/server architecture. *Compare* client (definition 3)."

54. A client is typically an end-point computer or software, sometimes used by a user, that communicates across LAN and WAN networks, and may contact servers to request some service. A user's desktop, laptop, and handheld device are examples of clients. (Note that computer can be both a client and a server.) EX1014 (Microsoft Computer Dictionary) at 92 ("**client** *n*. 1. In object-oriented programming, a member of a class (group) that uses the services of another class to which it is not related. *See also* inheritance (definition 1). 2. A process, such as a program or task, that requests a service provided by another program—for example, a word processor that calls on a sort routine built into another program. The client process uses the requested service without having to 'know' any working details about the other program or the service itself. *Compare* child (definition 1), descendant (definition 2). 3. On a local area network or the Internet, a computer that accesses shared network resources provided by another computer (called a server). *See also* client/server architecture, server.").

55. A client/server architecture describes how client computers interact with server computers. EX1014 (Microsoft Computer Dictionary) at 92

("**client/server architecture** *n*. An arrangement used on local area networks that makes use of distributed intelligence to treat both the server and the individual workstations as intelligent, programmable devices, thus exploiting the full computing power of each. This is done by splitting the processing of an application between two distinct components: a 'front-end' client and a 'back-end' server. The client component is a complete, stand-alone personal computer (not a 'dumb' terminal), and it offers the user its full range of power and features for running applications. The server component can be a personal computer, a minicomputer, or a mainframe that provides the traditional strengths offered by minicomputers and mainframes in a time-sharing environment: data management, information sharing between clients, and sophisticated network administration and security features. The client and server machines work together to accomplish the processing of the application being used. Not only does this increase the processing power available over older architectures but it also uses that power more efficiently. The client portion of the application is typically optimized for user interaction, whereas the server portion provides the centralized, multiuser functionality. *See also* distributed intelligence.").

B. Web Technologies

56. HTTP is an "[a]cronym for Hypertext Transfer Protocol. The client/server protocol used to access information on the World Wide Web. *See also* URL." EX1014 (Microsoft Computer Dictionary) at 238.

57. HTML is an "[a]cronym for Hypertext Markup Language. The markup language used for documents on the World Wide Web. HTML is an application of SGML that uses tags to mark elements, such as text and graphics, in a document to indicate how Web browsers should display these elements to the user and should respond to user actions such as activation of a link by means of a key press or mouse click. HTML 2.0, defined by the Internet Engineering Task Force (IETF), includes features of HTML common to all Web browsers as of 1995 and was the first version of HTML widely used on the World Wide Web. Future HTML development will be carried out by the World Wide Web Consortium (W3C). HTML 3.2, the latest proposed standard, incorporates features widely implemented as of early 1996. Most Web browsers, notably Netscape Navigator and Internet Explorer, recognize HTML tags beyond those included in the present standard. *See also* .htm, .html, SGML, tag (definition 3), Web browser." EX1014 (Microsoft Computer Dictionary) at 238. *See also*, generally, EX1016 (HTML 4.0 Specification).

58. A "Web server" or "HTTP Server" is "Server software that uses HTTP to serve up HTML documents and any associated files and scripts when requested

by a client, such as a Web browser. The connection between client and server is usually broken after the requested document or file has been served. HTTP servers are used on Web and Intranet sites. *Also called* Web server. *See also* HTML, HTTP, server (definition 2). 2. Any machine on which an HTTP server program is running." EX1014 (Microsoft Computer Dictionary) at 239.

59. A "Web browser" is "[a] client application that enables a user to view HTML documents on the World Wide Web, another network, or the user's computer; follow the hyperlinks among them; and transfer files. Text-based Web browsers, such as Lynx, can serve users with shell accounts but show only the text elements of an HTML document; most Web browsers, however, require a connection that can handle IP packets but will also display graphics that are in the document, play audio and video files, and execute small programs, such as Java applets or ActiveX controls, that can be embedded in HTML documents. Some Web browsers require helper applications or plug-ins to accomplish one or more of these tasks. In addition, most current Web browsers permit users to send and receive e-mail and to read and respond to newsgroups. *Also called* browser. *See also* ActiveX controls, helper application, hyperlink, Internet Explorer, Java applet, Lynx, Mosaic, Netscape Navigator, plug-in." EX1014 (Microsoft Computer Dictionary) at 505.

60. A "Uniform Resource identifier" (URI) is "[a] character string used to identify a resource (such as a file) from anywhere on the Internet by type and

location. The set of Uniform Resource Identifiers includes Uniform Resource Names (URNs) and Uniform Resource Locators (URLs). *Acronym*: URI (U-R-I). *See also* relative URL, Uniform Resource Name, URL." EX1014 (Microsoft Computer Dictionary) at 484.

61. A "URL" is an "[a]cronym for Uniform Resource Locator," and is a special form of a URI. A URL is "[a]n address for a resource on the Internet. URLs are used by Web browsers to locate Internet resources. A URL specifies the protocol to be used in accessing the resource (such as http: for a World Wide Web page or ftp: for an FTP site), the name of the server on which the resource resides (such as //www.whitehouse.gov), and, optionally, the path to a resource (such as an HTML document or a file on that server). *See also* FTP[1] (definition 1), HTML, HTTP, path (definition 1), server (definition 2), virtual path (definition 1), Web browser." EX1014 (Microsoft Computer Dictionary) at 487.

62. Like many client/server architectures (see IV.A above), clients send request to servers, which process the requests and return responses and resulting status codes. "HTTP status codes" are "[t]hree-digit codes sent by an HTTP server that indicate the results of a request for data. Codes beginning with 1 respond to requests that the client may not have finished sending; with 2, successful requests; with 3, further action that the client must take; with 4, requests that failed because

of client error; and with 5, requests that failed because of server error. *See also* 400, 401, 402, 403, 404, HTTP." EX1014 (Microsoft Computer Dictionary) at 239.

63. Many Internet protocols specify the format of the messages exchanged between clients and servers. Such messages often have a header and a body. A "message header" is "[a] sequence of bits or bytes at the beginning of a message that usually provides a timing sequence and specifies such aspects of the message structure as its length, data format, and block identification number. *See also* header (definition 2)." EX1014 (Microsoft Computer Dictionary) at 305.

64. HTTP is no different: it defines and uses many types of message headers. "HTTP messages consist of requests from client to server and responses from server to client. … Both types of message consist of a start-line, one or more header fields (also known as 'headers'), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and an optional message-body." EX1015 (HTTP RFC 2068) at 30. "HTTP header fields, which include general-header (section 4.5), request-header (section 5.3), response-header (section 6.2), and entity-header (section 7.1) fields, follow the same generic format as that given in §3.1 of RFC 822 [9]." EX1015 (HTTP RFC 2068) at 31.

65. The HTTP specification describes many other header types: "General Header Fields" (EX1015 (HTTP RFC 2068) at 34), "Response Header Fields" (*Id.* at 41), "Entity Header Fields" (*Id.* at 41-42), "Non-modifiable Headers" (*Id.* at 88-

89), "Combining Headers" (*Id.* at 89), a whole section on "Header Field Definitions" (*Id.* at 94-139), "Security Considerations" related to headers (*Id.* at 143-144), "HTTP Header Fields in Multipart Body-Parts" (*Id.* at 153), and "[a]dditional Header Field Definitions" (*Id.* at 157-160).

66. A "cookie" is, "On the World Wide Web, a block of data that a Web server stores on a client system. When a user returns to the same Web site, the browser sends a copy of the cookie back to the server. Cookies are used to identify users, to instruct the server to send a customized version of the requested Web page, to submit account information for the user, and for other administrative purposes." EX1014 (Microsoft Computer Dictionary) at 119 (definition 2).

67. Cookies are in fact special HTTP headers designed to manage arbitrary, persistent state between HTTP clients and servers. The cookies feature "specifies a way to create a stateful session with HTTP requests and responses. It describes two new headers, Cookie and Set-Cookie, which carry state information between participating origin servers and user agents." EX1017 (HTTP State Management Mechanism RFC 2109) at 1.

68. The cookie standard "describes a way to create stateful sessions with HTTP requests and responses. Currently, HTTP servers respond to each client request without relating that request to previous or subsequent requests; the technique allows clients and servers that wish to exchange state information to place

HTTP requests and responses within a larger context, which we term a 'session'. This context might be used to create, for example, a 'shopping cart', in which user selections can be aggregated before purchase, or a magazine browsing system, in which a user's previous reading affects which offerings are presented. There are, of course, many different potential contexts and thus many different potential types of session." EX1017 (HTTP State Management Mechanism RFC 2109) at 2.

69. Cookies are useful to provide a better user experience. For example, suppose a user reads one article on a news Web site; the cookie can record which articles the user had already read, so next time they browse the same news site, that site's Web server could inform the user which articles s/he had read and which are still unread. Similarly, cookies can be used to track and construct a user's online shopping cart. Without cookies, the Web would lack the necessary "memory" to let users have an experience that persists over time (*e.g.*, a user would have to complete all shopping or news reading in one sitting and would not be able to stop on one day and resume the next).

70. Web clients can provide information to Web servers that helps the Web server produce content appropriate for that client. When clients initially contact a server, they specify in a special header certain properties and capabilities such as screen resolution and size, running operating system, processor architecture, and more. "[t]he User-Agent request-header field contains information about the user

agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain multiple product tokens (section 3.8) and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application." EX1015 (HTTP RFC 2068) at 134.

71. Both web clients and servers can exchange headers that contain one or more of these "product" keywords. For example, "Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products which form a significant part of the application to be listed, separated by whitespace. By convention, the products are listed in order of their significance for identifying the application. … Examples: User-Agent: CERN-LineMode/2.15 libwww/2.17b3 [] Server: Apache/0.8.4". EX1015 (HTTP RFC 2068) at 28.

72. Web servers serve Web pages, written in HTML, over the HTTP protocol. The "Web" is "[a] set of interlinked documents in a hypertext system. The user enters the web through a home page. *See also* World Wide Web." EX1014 (Microsoft Computer Dictionary) at 505.

73. This "World Wide Web" (or "WWW") represents "[t]he total set of interlinked hypertext documents residing on HTTP servers all around the world. Documents on the World Wide Web, called pages or Web pages, are written in HTML (Hypertext Markup Language), identified by URLs (Uniform Resource Locators) that specify the particular machine and pathname by which a file can be accessed, and transmitted from node to node to the end user under HTTP (Hypertext Transfer Protocol). Codes, called tags, embedded in an HTML document associate particular words and images in the document with URLs so that a user can access another file, which may be halfway around the world, at the press of a key or the click of a mouse. These files may contain text (in a variety of fonts and styles), graphics images, movie files, and sounds as well as Java applets, ActiveX controls, or other small embedded software programs that execute when the user activates them by clicking on a link. A user visiting a Web page also may be able to download files from an FTP site and send messages to other users via e-mail by using links on the Web page. The World Wide Web was developed by Timothy Berners-Lee in 1989 for the European Laboratory for Particle Physics (CERN). *Acronym*: WWW (WWW). *Also called* w3, W3, Web. *See also* ActiveX controls, HTML, HTTP, HTTP server (definition 2), Java applet, URL." EX1014 (Microsoft Computer Dictionary) at 511-512.

74. Of note, HTML pages can include one or more "HREFs", which are "[s]hort for hypertext reference. An attribute in an HTML document that defines a link to another document on the Web. *See also* HTML." EX1014 (Microsoft Computer Dictionary) at 237. Users can click on one HREF in one Web page, and get to another, then another, in a never-ending sequence. This permits Web pages to embed content (directly or recursively) for other Web pages.

75. HTML Web pages are typically represented and written as plain text. However, Web content that can change dynamically and be interactive is often more desired as it can adapt to changing conditions (*e.g.*, Web pages showing the latest stock market status for a user-specified stock, or top sports news of the day). Therefore, a "dynamic Web page" is "[a] Web page that has fixed form but variable content, allowing it to be tailored to a customer's search criteria." EX1014 (Microsoft Computer Dictionary) at 167.

76. Dynamic Web content can be produced by executing code on the Web server or the Web client/browser. Such code can be written by any programming or scripting language. A "CGI script" is "[s]hort for Common Gateway Interface script. An external application that is executed by an HTTP server machine in response to a request by a client, such as a Web browser. Generally, the CGI script is invoked when the user clicks on some element in a Web page, such as a link or an image. Communication between the CGI script and the server is carried out via the CGI

specification. CGI scripts can be written in many programming languages, including C, C++, and Visual Basic. However, the most commonly used language for CGI scripts is Perl, because it is a small but robust language and it is common on UNIX, which is the platform on which the majority of Web sites run. CGI scripts don't necessarily need to be scripts; they can also be batch programs or compiled programs. CGI scripts are used to provide interactivity in a Web page, including such features as providing a form that users can fill out, image maps that contain links to other Web pages or resources, and links that users can click on to send e-mail to a specified address. … *See also* CGI (definition 1), cgi-bin, image map, Perl. *Compare* ActiveX controls, Java applet." EX1014 (Microsoft Computer Dictionary) at 85.

77. As noted, dynamic content can also be produced by running directly on the Web client/browser, using several technologies. "ActiveX controls and Java applets can provide much the same functionality as CGI scripts, through different means." EX1014 (Microsoft Computer Dictionary) at 85 ("CGI script" definition").

78. An "applet" is "[a] small piece of code that can be transported over the Internet and executed on the recipient's machine. The term is especially used to refer to such programs as they are embedded in line as objects in HTML documents on the World Wide Web." EX1014 (Microsoft Computer Dictionary) at 27.

79. Microsoft ActiveX is "[a] set of technologies that enables software components to interact with one another in a networked environment, regardless of the language in which the components were created. ActiveX, which was developed as a proposed standard by Microsoft in the mid 1990s and is currently administered by the Open Group, is built on Microsoft's Component Object Model (COM). Currently, ActiveX is used primarily to develop interactive content for the World Wide Web, although it can be used in desktop applications and other programs. ActiveX controls can be embedded in Web pages to produce animation and other multimedia effects, interactive objects, and sophisticated applications. *See also* ActiveX controls, COM. *Compare* applet, plug-in (definition 2)." EX1014 (Microsoft Computer Dictionary) at 15.

80. Similarly, a "Java applet" is "[a] Java class that is loaded and run by an already-running Java application such as a Web browser or an applet viewer. Java applets can be downloaded and run by any Web browser capable of interpreting Java, such as Internet Explorer, Netscape Navigator, and Hot-Java. Java applets are frequently used to add multimedia effects and interactivity to Web pages, such as background music, real-time video displays, animations, calculators, and interactive games. Applets can be activated automatically when a user views a page, or they may require some action on the part of the user, such as clicking on an icon in the

Web page. *See also* applet, Java." EX1014 (Microsoft Computer Dictionary) at 268.

C. Caching Technologies

81. Computers store the 0 and 1 values of bits in memory devices. Memory devices can be volatile (lose data when device power is off) or non-volatile (data is not lost when power is off). There are many storage media in existence: tapes, hard disks, floppies, Flash technologies (*e.g.*, SSDs), optical (*e.g.*, CD-ROM and DVDs), DRAM (*e.g.*, system memory), CPU registers, and more. All these devices can be placed on a specific point or range in a 3-dimensional continuum comprising their (a) capacity, (b) performance, and (c) price. Almost universally, the faster performing devices (*e.g.*, DRAM) are smaller and cost more per unit storage, whereas the larger-capacity devices (*e.g.*, HDD) are slower and cheaper per gigabyte of storage capacity. The difference in speed/cost/capacity among devices is very significant and ranges along at least 9 orders of magnitude—or a billion times bigger, faster, or costlier. See §IV.A above.

82. Therefore, "[t]he wide variety of storage systems in a computer system can be organized in a hierarchy (Figure 2.6) according to their speed and their cost. The higher levels are expensive, but are fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases. This tradeoff is reasonable; if a given storage system were both faster and less expensive than another — other

properties being the same — then there would be no reason to use the slower, more expensive memory." EX1018 (Silberschatz) at 42, Fig. 2.6.

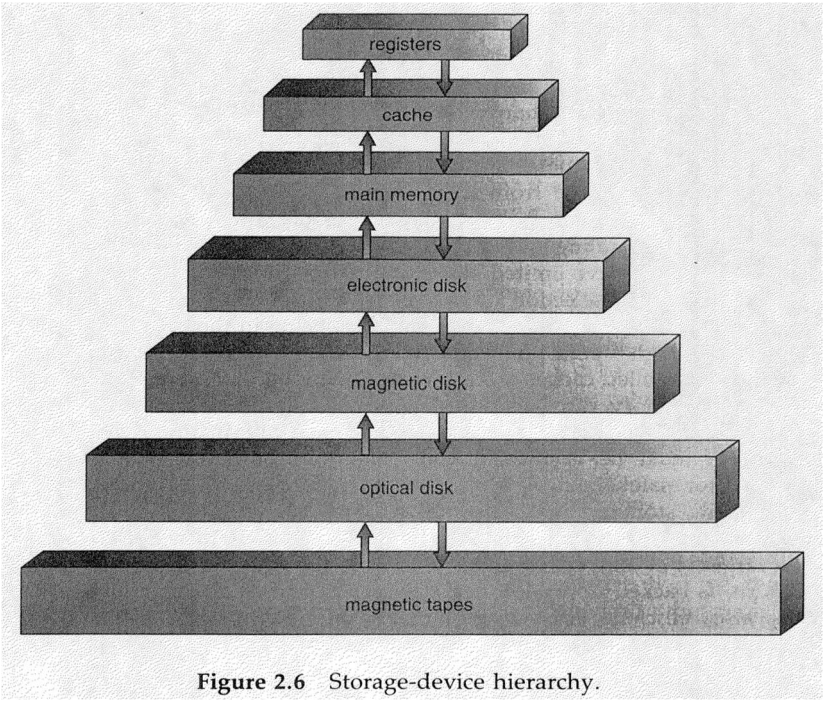


Figure 2.6 Storage-device hierarchy.

83. Managing all these factors is challenging: cost, speed, access speed/latency, size, and volatility. "The design of a complete memory system attempts to balance all these factors: It uses only as much expensive memory as necessary, while providing as much inexpensive, nonvolatile, memory as possible. Cache s can be installed to ameliorate performance differences where there is a large access-time or transfer-rate disparity between two components". EX1018 (Silberschatz) at 42-43.

84. "*Caching* is an important principle of computer systems, in both hardware and software. Information is normally kept in some storage system (such

as main memory). As it is used, it is copied into faster storage system, the cache, on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we used the information directly from the cache; if it is not, we use the information from the main storage system, putting a copy in the cache under the assumption that there is a high probability that it will be needed again". EX1018 (Silberschatz) at 43.

85. A "cache" is "[a] special memory subsystem in which frequently used data values are duplicated for quick access. A memory cache stores the contents of frequently accessed RAM locations and the addresses where these data items are stored. When the processor references an address in memory, the cache checks to see whether it holds that address. If it does hold the address, the data is returned to the processor; if it does not, a regular memory access occurs. A cache is useful when RAM accesses are slow compared with the microprocessor speed, because cache memory is always faster than main RAM memory. *See also* disk cache, wait state." EX1014 (Microsoft Computer Dictionary) at 72.

86. Caches can exist at many levels. A "CPU cache" is "[a] section of fast memory linking the central processing unit (CPU) and main memory that temporarily stores data and instructions the CPU needs to execute upcoming commands and programs. Considerably faster than main memory, the CPU cache contains data that is transferred in blocks, thereby speeding execution. The system

anticipates the data it will need through algorithms. *Also called* cache memory, memory cache. *See also* cache, central processing unit, VCACHE." EX1014 (Microsoft Computer Dictionary) at 122. CPUs internally can have 2 or more caches, marked "level 1" or L1, "level 2", etc. *See also* EX1014 (Microsoft Computer Dictionary) at 276 ("L1 cache" and "L2 cache" definitions).

87. A "disk cache" "[a] portion of a computer's random access memory (RAM) set aside for temporarily holding information read from disk. A disk cache does not hold entire files, as does a RAM disk (a portion of memory that acts as if it were a disk drive). Instead, a disk cache is used to hold information that either has recently been requested from disk or has previously been written to disk. If the required information remains in a disk cache, access time is considerably faster than if the program must wait for the disk drive mechanism to fetch the information from disk. *See also* cache. *Compare* disk buffer." EX1014 (Microsoft Computer Dictionary) at 151.

88. A "RAM cache" is "[c]ache memory that is used by the system to store and retrieve data from the RAM. Frequently accessed segments of data may be stored in the cache for quicker access compared with secondary storage devices such as disks. *See also* cache, RAM." EX1014 (Microsoft Computer Dictionary) at 395.

89. In any cache hierarchy, typically the cache (above) is faster and smaller than the backend location (below) from where the data was retrieved. Therefore,

caches must be managed appropriately. If a data item is found in a cache it is called a "cache hit"—otherwise it is a "cache miss" that requires retrieving the backend data and copying it into the cache. A "hit" is "1. A successful retrieval of data from a cache rather than from the slower hard disk or RAM. *See also* cache, hard disk, RAM. 2. A successful retrieval of a record matching a query in a database. *See also* query (definition 1), record[1]. 3. Retrieval of a document, such as a home page, from a Web site." EX1014 (Microsoft Computer Dictionary) at 234-235.

90. Cached data is typically stored temporarily. At some point a cache space fills up, and the system must determine which items to "evict"; often the oldest, least used items are evicted first. "Since caches have limited size, *cache management* is an important design problem. Careful selection of the cache size and of a replacement policy can result in 80 to 99 percent of all accesses being in the cache, resulting in extremely high performance. Various replacement algorithms for software-controlled caches are discussed in Chapter 9." EX1018 (Silberschatz) at 44.

91. Such caching techniques are particularly effective when considering users' work habits and workload patterns. Studies have shown that users and applications often operate on a small set of data files (called a "working set") within a narrow time period. For example, while drafting this very declaration, I am editing the same Word document repeatedly over a period of several days and weeks. Thus,

my OS (and any OS prior to 1998) recognizes this access pattern and keeps this Word file cached in DRAM for long time periods. Conversely, it makes less sense for the OS to waste precious DRAM to cache documents I've not worked on for months. However, if, for any reason, I try to access a document I've not opened in a long time, and the document is not already cached in DRAM, then indeed my OS will have to incur a longer latency—and I will have to wait for however long it takes to retrieve the older document from the slower HDD (and cache it into DRAM). Cache management considers when to "admit" data into the cache from backend sources and when to "evict" older data in the cache to make room for new items. One of the most popular and successful cache eviction algorithms is the Least-Recently-Used (LRU), a heuristic stating that items used more recently are more likely to be used in the near future, and therefore should be kept—while lesser used items should be evicted. *See* EX1018 (Silberschatz) at 319-325.

92. Admitting an item from a backend into a cache, or retrieving it from the cache itself, can be seen as a "get" (or read) operation. Conversely, an application inserting an item into a cache, is also called "put" (or write) operation. When new items are inserted into a cache by an application, they are said to be "dirty" because their content is newer than the corresponding backend item. Such dirty items must be "flushed" or written to the backend before they are removed from the cache. A "dirty bit" is "[a] bit used to mark modified data in a cache so that the modifications

may be carried over to primary memory. *See also* bit, cache." EX1014 (Microsoft Computer Dictionary) at 149.

93. To "flush" is "[t]o clear a portion of memory. For example, to flush a disk file buffer is to save its contents on disk and then clear the buffer for filling again." EX1014 (Microsoft Computer Dictionary) at 202 (definition 2). In some caches, items in the cache could become "stale" if the corresponding backend item could have been updated: in that case, some caches attempt to revalidate any cached item before returning it to an application, and as needed remove stale items.

94. Caching is applicable anywhere there's room in one memory device to access data much faster than its source location. Other than RAM-HDD caches, one can cache items on a local storage device that would otherwise take even longer to retrieve than the HDD's access speeds (*e.g.*, network storage, distant servers). A "proxy server" is "[a] firewall component that manages Internet traffic to and from a local area network (LAN) and can provide other features, such as document caching and access control. A proxy server can improve performance by supplying frequently requested data, such as a popular Web page, and can filter and discard requests that the owner does not consider appropriate, such as requests for unauthorized access to proprietary files. *See also* firewall." EX1014 (Microsoft Computer Dictionary) at 387.

95. As soon as the Web began, many Web servers were launched. Still, the number of clients/browsers grew much faster than the number of servers. Therefore, Web servers in the 1990s began getting overloaded with many Web requests, large spikes of activity, and many Web servers even crashed under the heavy load. At the same time, network speeds were much slower in the 1990s than they are today (many were still using dialup modem lines). Worse, most users and companies paid for each byte of data they transmitted over their network.

96. In response, Web server providers kept upgrading the hardware, adding more servers, and replicating Web content into a "farm" of Web servers to increase the aggregate throughput. (I personally consulted for a startup in the 1990s that developed one of the first "Web load balancers". See II above.)

97. Similarly, companies were faced with rising costs: if each of their employees browsed the same Web sites, the load on the company's network increased multi-fold, with a commensurate increase is network-usage costs. The community therefore created Web caching proxies to address this problem. A Web cache is a proxy device installed at the company's network perimeter, in close proximity to the company firewall (or even integrated with one). All users' Web browsing traffic was redirected (proxied) via the Web cache device. The Web cache device would redirect HTTP requests to their actual Internet Web servers, retrieve the HTML pages, and cache (copy) them on the proxy's local storage (*e.g.*, an HDD).

When users subsequently try to browse the same Web page, the proxy device would return that Web page's HTML content much faster—speeding up Web browsing (which also makes users happier), and saving network traffic and costs (which makes managers happier).

98. One of the first and most popular Web proxy cache was the "squid" cache, first released in 1996 (which I personally used, see §II above). See "https://en.wikipedia.org/wiki/Squid_(software)" (last viewed on February 16, 2024).

99. The issue of Web performance quickly dominated the Web, with many researchers investigating techniques and multiple companies offering products and services. The Internet Engineering Task Force (IETF), recognizing the importance of Internet content caching, began defining related standards.

100. The IETF defined the "Internet Cache Protocol" (ICP) and released a specification document outlining "the application of ICPv2 (Internet Cache Protocol version 2, RFC2186) to Web caching. ICPv2 is a lightweight message format used for communication among Web caches. Several independent caching implementations now use ICP[3,5], making it important to codify the existing practical uses of ICP for those trying to implement, deploy, and extend its use. ICP queries and replies refer to the existence of URLs (or objects) in neighbor caches. Caches exchange ICP messages and use the gathered information to select the most

appropriate location from which to retrieve an object. A companion document (RFC2186) describes the format and syntax of the protocol itself. In this document we focus on issues of ICP deployment, efficiency, security, and interaction with other aspects of Web traffic behavior." EX1019 (Internet Cache Protocol RFC-2187) at 1.

D. Programming Languages, Compilers and Interpreters, Optimization

101. As discussed in §IV.A above, computer software can be written in several programming or scripting languages, and eventually executed on a processor. A "compiler" is "1. Any program that transforms one set of symbols into another by following a set of syntactic and semantic rules. 2. A program that translates all the source code of a program written in a high-level language into object code prior to execution of the program. *See also* assembler, compile, high-level language, interpreted language, language processor, object code." EX1014 (Microsoft Computer Dictionary) at 105-106.

102. A compiler for the C programming language goes through several stages: a pre-processor, syntax and lexical analyzer, assembler, and linker. In fact, the compiler itself is often an umbrella program that takes source code input, passes it through one stage, which transforms the input into another form; then the compiler

takes the first stage output, passes it as input to the second stage transformer, and so on.

103. In stage 1, a "preprocessor" is "[a] device or routine that performs preliminary operations on input before passing it on for further processing. *See also* front-end processor (definition 1). *Compare* post-processor." EX1014 (Microsoft Computer Dictionary) at 378.

104. In stage 2, the "syntax checker" is "[a] program for identifying errors in syntax for a programming language. *See also* syntax, syntax error." EX1014 (Microsoft Computer Dictionary) at 457.

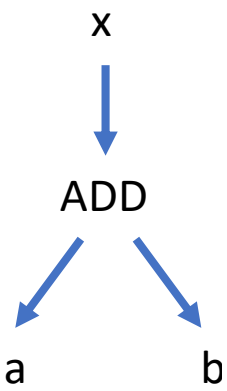
105. In stage 3, the "assembler" is "[a] program that converts assembly language programs, which are understandable by humans, into executable machine language. *See also* assemble, assembly language, assembly listing, compiler (definition 2), machine code." EX1014 (Microsoft Computer Dictionary) at 32.

106. Finally, in stage 4, the "linker" is "[a] program that links compiled modules and data files to create an executable program. A linker can also have other functions, such as creating libraries. *See also* library, link (definition 1), program creation." EX1014 (Microsoft Computer Dictionary) at 285.

107. For different compiled languages (*e.g.*, C, C++), the first two stages differ, but the latter two stages (assembler and linker) can be shared. The reason is that in the middle, most compilers convert the language-specific syntax into a

generic syntax of code, often organized as a tree. An "abstract syntax tree" (AST) is "[a] treelike representation of programs used in many integrated programming environments and structure-oriented editors." EX1014 (Microsoft Computer Dictionary) at 11.

108. Consider the statement to add two variables and store the result into another. In C, this would be represented as "x=a+b", but in the Pascal programming language, this will be represented as "x:=a+b". Either way, the uniform AST representation of these statements would be a small tree such as:



109. A "native compiler" is "[a] compiler that produces machine code for the computer on which it is running, as opposed to a cross-compiler, which produces code for another type of computer. Most compilers are native compilers. *See also* compiler (definition 2), cross-compiler." EX1014 (Microsoft Computer Dictionary) at 324.

110. Code compiled for one operating system and processor architecture can only run on that combination of operating system and architecture. For example,

Linux code compiled for Intel CPUs cannot be expected to run on BSD or Windows systems even if they run on the same Intel CPUs. Similarly, SunOS/Solaris code compiled for the SPARC architecture cannot run on the same OS running on an Intel architecture. Therefore, code must be compiled for specific operating systems (and versions) as well as the processor architecture.

111. A "cross-compiler" "[a] compiler that executes on one hardware platform but generates object code for another. *See also* assembler, compiler, cross-assembler, cross development." EX1014 (Microsoft Computer Dictionary) at 123. Cross compilers are useful for example when the target environment where the code may run is not so convenient to work with. For example, it is more convenient to develop code for a handheld device on a regular, full size (Intel) workstation, but generate code for the handheld's own architecture and operating system. Another example where a cross-compiler is useful is in a proxy device that holds generic source code: clients and users can contact the proxy device (which runs a specific operating system and architecture) to request executable versions suitable for running the client's own system.

112. An "interpreter" is "[a] program that translates and then executes each statement in a program written in an interpreted language. *See also* compiler, interpreted language, language processor." EX1014 (Microsoft Computer Dictionary) at 261.

113. An "interpreted language" is "[a] language in which programs are translated into executable form and executed one statement at a time rather than being translated completely (compiled) before execution. Basic, LISP, and APL are generally interpreted languages, although Basic can also be compiled. *See also* compiler. *Compare* compiled language." EX1014 (Microsoft Computer Dictionary) at 261. Additional examples of interpreted languages (that I personally used in the 1990s, see §II above) include shells (*e.g.*, sh, ksh, csh, and bash) and Perl.

114. Making programs run faster had always been desired, especially in the 1990s when computer processors, memory, storage, and networks were considered to be too slow to handle the rapid growth of the Internet (aka the "dot-com" boom). Therefore, researchers and programmers began to develop various forms of optimization.

115. An "optimizing compiler" is "[a] compiler that analyzes its output (assembly language or machine code) to produce more efficient (smaller or faster) instruction sequences." EX1014 (Microsoft Computer Dictionary) at 343.

116. An "optimization" is, "1. In programming, the process of producing more efficient (smaller or faster) programs through selection and design of data structures, algorithms, and instruction sequences. 2. The process of a compiler or assembler in producing efficient executable code. *See also* optimizing compiler." EX1014 (Microsoft Computer Dictionary) at 343.

117. For example, I personally used the GNU C compiler (gcc), passing it the optimization flag "-O". (See §II above.) C compilers can have several levels of optimizations, from -O1 to -O3, with "-O" defaulting to level 1. EX1020 (GNU C Manual Page) at 16-17. The gcc compiler also supports many fine-grained optimization options, for example -funroll-loops, an optimization that converts small loops into linear instruction sequences, leading to more efficient CPU cache utilization at the expense of slightly larger binaries. *Id.* at 2. C compilers I personally used in the late 1990s added additional optimization levels (up to -O6).

118. Applying compiler optimizations produces more efficient code. However, the compilation process itself can take substantially longer, and may produce larger executables. Therefore, developers have to tradeoff the level of optimization against the compilation speed. Programmers could also help the compiler build more efficient code by, for example, providing hints that certain critical variables should be stored in (fast) CPU registers, for example by including the keyword "register" when declaring an integer variable (*e.g.*, "register int x").

119. Optimizers can operate at various stages of the compiler, especially the aforementioned AST encoding stage. *See* EX1006 (Franz) at 4, 13 ("a tree-based encoding has several important advantages over a linear stream of byte-codes for some virtual machine. It preserves the control-flow structure of the original program, which makes it much easier to subsequently perform code optimization.").

Consider for example a code fragment with two statements: "x=a+b" and "y=x+c". A non-optimizing compiler will turn each statement into an addition of two variables followed by an assignment to a third, overall using at least five CPU registers (to hold the contents of the variables a, b, c, x, and y) and two AST nodes. An optimizing compiler, however, will employ more sophisticated parsing and processing: it will notice that the assignment to variable 'y' can be embedded directly into the assignment for 'x', effectively merging the two statements into one that reads "x=a+b+c". This would need only four registers and one AST node for the 3-variable assignment.

120. Programmers need a suitable system and environment for developing, testing, and debugging code. An "application development environment" is "[a]n integrated suite of programs for use by software developers. Typical components of application development environments include a compiler, file browsing system, debugger, and text editor for use in creating programs." EX1014 (Microsoft Computer Dictionary) at 28. Similarly, an "integrated development environment" is "[a] set of integrated tools for developing software. The tools are generally run from one user interface and consist of a compiler, an editor, and a debugger, among others. *Acronym*: IDE." EX1014 (Microsoft Computer Dictionary) at 255.

121. With a growing complexity of the Internet and the rising popularity of the WWW, a new need arose for developing secure code more quickly, such that it

can run on Web servers/browsers with greater ease. "Java" is "[a]n object-oriented programming language, developed by Sun Microsystems, Inc. Similar to C++, Java is smaller, more portable, and easier to use than C++ because it is more robust and it manages memory on its own. Java was also designed to be secure and platform-neutral (meaning that it can be run on any platform) through the fact that Java programs are compiled into byte-codes, which are similar to machine code and are not specific to any platform. This makes it a useful language for programming Web applications, since users access the Web from many types of computers. Currently, the most widespread use of Java is in programming small applications, or applets, for the World Wide Web. *See also* byte-code, Java applet, object-oriented programming." EX1014 (Microsoft Computer Dictionary) at 268.

122. In Java, "bytecode" is "[a]n encoding of a computer program that a compiler produces when the original source code is processed. This encoding is in an abstract, processor-independent form that cannot be directly executed by most CPUs but is highly suitable for further analysis (for example, compiler optimization), for processing by interpreters (for example, executing Java applets within Web browsers), or for use in generation of binary instructions for the target computer's CPU. Intermediate bytecode production is a feature of the compilers for the Pascal and Java programming languages. *See also* compiler (definition 2),

central processing unit, interpreter, Java, Java applet, Pascal." EX1014 (Microsoft Computer Dictionary) at 69.

123. Java bytecode cannot be executed directly on a specific processor: it has to be converted into native processor instructions by the Java Virtual Machine (JVM). The JVM verifies the bytecode's safety, memory compliance, and more. It then converts the bytecode into executable form and runs it. JIT, or "just-in-time" is a process that "2. Describes a compiler that compiles Java on the fly. *See also* Java, on the fly." EX1014 (Microsoft Computer Dictionary) at 271.

124. A "Java-compliant browser" is a "[a] Web browser with support for the Java programming language built into it. Most current Web browsers are java-compliant. *See also* java, Web browser." As discussed in §IV.B above, Java applets were designed specifically to be small pieces of code that could run inside Web browser's JVM. The browser's JVM would compile the bytecode on the fly and execute it inside the browser.

E. Transforming Data and Files

125. As discussed in §IV.D above, compilers transform source code, through multiple stages, into a final working executable. Abstract Syntax Trees (ASTs) are one such transformation stage; optimizations at each compilation stage are another form of transformations.

126. Another transformation is used to save storage space and network bandwidth through compression. To "compress" is "[t]o reduce the size of a set of data, such as a file or a communications message, so that it can be stored in less space or transmitted with less bandwidth. Data can be compressed by removing repeated patterns of bits and replacing them with some of summary that takes up less space; restoring the repeated patterns decompresses the data. Lossless compression methods must be used for text, code, and numeric data files; lossy compression may be used for video and sound files. *See also* lossless compression, lossy compression." EX1014 (Microsoft Computer Dictionary) at 107 (definition 2).

127. A "data compression" represents "[a] means of reducing the amount of space or bandwidth needed to store or transmit a block of data, used in data communications, facsimile transmission, and CD-ROM publishing. *Also called* data compaction." EX1014 (Microsoft Computer Dictionary) at 130. Just about any data can be compressed: user files, images, video, audio, binaries, applets, etc. Compression techniques and tools have been in use since well before 1998.

128. Another common transformation, useful to protect user's privacy, is "encryption", or "[t]he process of encoding data to prevent unauthorized access, especially during transmission. Encryption is usually based on a key that is essential for decoding. The U.S. National Bureau of Standards created a complex encryption standard, Data Encryption Standard (DES), which provides almost unlimited ways

to encrypt documents. *See also* DES." EX1014 (Microsoft Computer Dictionary) at 175.

129. There are many forms of verification that can be applied to data files. One example are the syntax checks performed by compilers (see §IV.D above).

130. Another is to verify the integrity of data using cryptographic hashing algorithms such as SHA. "SHA" is an "[a]cronym for Secure Hash Algorithm. A technique that computes a 160-bit condensed representation of a message or data file, called a message digest. The SHA is used by the sender and the receiver of a message in computing and verifying a digital signature, for security purposes. *See also* algorithm, digital signature." EX1014 (Microsoft Computer Dictionary) at 432. The term "data integrity" represents "[t]he accuracy of data and its conformity to its expected value, especially after being transmitted or processed." EX1014 (Microsoft Computer Dictionary) at 131.

131. Users can also verify the authenticity of files' sources using digital signatures. A "digital signature" is "[a] personal authentication method based on encryption and secret authorization codes used for 'signing' electronic documents." EX1014 (Microsoft Computer Dictionary) at 145.

132. Digital signatures often combine cryptographic techniques and hashing algorithms to enable bi-directional authentication and secure communications. "public key cryptography" or "public key encryption" is "[a]n asymmetric scheme

that uses a pair of keys for encryption: the public key encrypts data, and a corresponding secret key decrypts it. For digital signatures, the process is reversed: the sender uses the secret key to create a unique electronic number that can be read by anyone possessing the corresponding public key, which verifies that the message is truly from the sender. *See also* private key, public key." EX1014 (Microsoft Computer Dictionary) at 388-389.

133. When programs run, they can verify their own safety; the operating system and system libraries often perform additional verifications. An "error control" is "1. The section of a program, procedure, or function that checks for errors such as type mismatches, overflows and underflows, dangling or illegal pointer references, and memory-use inconsistencies. 2. The process of anticipating program errors during software development." EX1014 (Microsoft Computer Dictionary) at 179-180.

134. A "debugger" is "[a] program designed to aid in debugging another program by allowing the programmer to step through the program, examine the data, and monitor conditions such as the values of variables. *See also* bug (definition 1), debug." EX1014 (Microsoft Computer Dictionary) at 135. I personally used several debuggers starting in the late 1980s (*e.g.*, dbx, gdb). See §II above.

135. Producing an "audit trail" is, "[i]n reference to computing, a means of tracing all activities affecting a piece of information, such as a data record, from the

time it is entered into a system to the time it is removed. An audit trail makes it possible to document, for example, who made changes to a particular record and when." EX1014 (Microsoft Computer Dictionary) at 36. Tracing is sometimes called "monitoring" or "logging." Programmers have used such techniques to analyze source code and running programs, looking for bottlenecks, which in turn allow them to find and fix bugs, as well as to optimize the code (*e.g.*, to run faster, to consume less memory).

F. Intermediate Compiler Stages

136. As discussed in §IV.D above, there are many possible stages of converting source code—through multiple intermediate stages—to a final executable that can be run on a specific machine/CPU architecture: at least a pre-processor, syntax checker, assembler, and linker. And as further discussed in §IV.E above, there are many transformations that can be applied at any stage, including compression, encryption and integrity verification, debugging, tracing and profiling, and many optimizations.

137. But these are not absolutes, and many additional variants exist. For example, the shell discussed in §IV.D above, scripts execute shell-script code (a human written "source code") directly, without first transforming/compiling the shell script code into binary. Some shell script processors (*e.g.*, Perl) internally compile and optimize the source code before executing it; and all shell script

processors perform some form of syntax checking. As further discussed in §IV.D above, Java and the like include another stage, called bytecode, that produces yet another intermediate representation that is architecture agnostic; bytecodes can be transformed to a specific architecture and executed there (*e.g.*, using the Java Virtual Machine, or JVM).

138. Indeed, an "intermediate language" is "[a] computer language used as an intermediate step between the original source language, usually a high-level language, and the target language, usually machine code. Some high-level compilers use assembly language as an intermediate language. *See also* compiler (definition 2), object code." EX1014 (Microsoft Computer Dictionary) at 257.

139. Yet even source code, often thought of as being exclusively written by humans, can be automatically generated. For example, there are meta-tools that use a higher-level language (written by humans) which in turn produce source code (*e.g.*, C/C++). (I personally used tools such as 'yacc' and 'lex' to generate C code for parsing custom syntax.) A "metacompiler" is "[a] compiler that produces compilers. The UNIX utility yacc (Yet Another Compiler-Compiler) is a metacompiler. If it is given a language specification, yacc produces a compiler for that language. *See also* compiler (definition 2)." EX1014 (Microsoft Computer Dictionary) at 305.

140. There are multiple reasons why one would want to store a program in one or more intermediate stages, and not only in its final binary executable form.

The Java bytecode is one such example, as it allows the separation of high-level architecture-agnostic executable instructions from architecture-specific ones.

141. A user may want to use an optimizer that's designed to operate on a specific stage (*e.g.*, syntax, high-level assembly, bytecode). It is often easier and better to develop custom optimizers that are designed to operate on a specific type of data or compiler stage. One may want to run an optimizer after one intermediate stage, produce code for the next stage, then invoke another optimizer for that new stage, and so on. Optimizations can indeed be cumulative.

142. Debugging is another popular reason to operate on one or more intermediate stages. One may insert source-code assertions into the human readable source code or after the pre-processing stage, to verify certain properties. An "assertion" is "[a] Boolean statement used in a program to test a condition that, if the program is operating correctly, should always evaluate as true; otherwise the program will typically terminate with an appropriate error message. Assertions are used for debugging programs and for documenting how a program should operate." EX1014 (Microsoft Computer Dictionary) at 32. Debugging assertions are a form of verification: they can be inserted at any stage, to test for different conditions.

143. The GNU C compiler can automatically insert debugging statements using the '-g' flag, which can be used by symbolic debuggers (*e.g.*, gdb, dbx). See EX1020 (GNU C Manual Page) at 2 (disclosing several "Debugging Options"). But

since debugging statements increase the binary size and slow it down, tools exist (*e.g.*, UNIX's 'strip' program) to remove such debugging symbols and statements from a binary. Therefore, adding and removing debugging symbols represents two more possible (intermediate) stages in the compiler tool-chain.

144. Profiling is yet another popular reason to operate at multiple intermediate stages. Profiling is often used to trace and/or optimize a program. While compilers can perform some amount of automated optimization at different stages, often programmers have to be involved directly to evaluate the performance of their programs, find bottlenecks, and optimize them.

145. For example, suppose a program invokes a certain function many times a second, and that function uses a sub-optimal data structure (*e.g.*, an unsorted linked list). A "profiler" can detect and report, at runtime, which functions are taking up an unusual percentage of execution time. A programmer can then investigate the bottleneck functions and rewrite the code (*e.g.*, switch to a more efficient hash-table or binary-tree data structure).

146. Profiling (like debugging) is another form of assistive code instrumentation that inserts special instructions at specific stages of the compilation, to effectuate a specific purpose. For example, the GNU C compiler (gcc) includes several profiling flags. For example, the '-p' flag will "[g]enerate extra code to write

profile information suitable for the analysis program **prof**." EX1020 (GNU C Manual Page) at 15. *See also Id.* ('-pg' and '-a' profiling flags).

147. In sum, users always desired to have as much control to transform code in various intermediary stages to another, for multiple reasons. The complexity and versatility of compilers that resulted is apparent, as they can control and transform data in many ways, at various stages. That is why, for example, gcc supports hundreds of compile-time flags. *See, generally*, EX1020 (GNU C Manual Page).

V. U.S. PATENT NO. 6,976,248

148. The '248 Patent is titled "Application server facilitating with client's computer for applets along with various formats" and was filed on October 1, 2001 and issued on December 13, 2005. EX1001 at cover.

149. I understand the '248 Patent is a continuation of U.S. Patent Application Ser. No. 09/040,972 (now U.S. Patent No. 6,324,685). The first application in this chain, U.S. Patent Application Ser. No. 09/040,972, was filed on March 18, 1998.

150. I have been asked by counsel to use March 18, 1998, as the priority date of the '248 Patent. The opinions I offer below regarding the prior art, the combinations set forth in my declaration, or the applicability of those combinations to certain claims of the '248 Patent do not change whether the '248 Patent is entitled to a priority date different to March 18, 1998. I do not offer an opinion, as I have

not been asked to offer such an opinion, as to the proper priority date of the '075 Patent.

A. Summary of the '248 Patent

151. The '248 Patent discloses server architecture that delivers applications requested by a client computer. EX1001 at Abstract. This applet server architecture "allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code." *Id.* at 2:21-24. The requested and executed applets are described in the '248 Patent as "any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self contained program, or it can be a code fragment associated with a larger application." *Id.* at 3:20-26.

152. The '248 Patent further notes that it "relates to computer operating systems and, in particular to a server architecture providing application caching and security verification." *Id.* at 1:13-15. The '248 Patent specifically states that "[i]n operation, client computer A 12 transmits a request to an applet server computer 10 requesting an applet in a particular form. … The request may also specify that the applet be verified or have some other transformation operation preformed upon it. Verification, optimization and compression are examples of types of transformation operations." *Id.* at 3:55-64.

153. The ’248 Patent also includes a cache that “stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache 24 … the applet server manager 22 instructs the network interface 20 to transmit the applet to [the] requesting client computer A 12.” *Id.* at 4:3-8. If the applet is not available in the cache, then “the applet server manager 22 will attempt to build the requested applet from local resource 26 and one or more transformation operations performed by one or more transformers 28.” *Id.* at 4:9-13.

154. If the applet cannot be built from local resources, the ’248 Patent states that “the applet server manager 22 will pass a request for the requested applet to external network 16 and/or untrusted network 18.” *Id.* at 4:21-28. “The applet server manager 22 … may request the local resources 26 and transformers 28 it needs to complete building the applet itself.” *Id.* at 4:25-28. “After the applet server manager 22 builds the applet, the network interface 20 transmits the applet to the requesting client computer A12.” *Id.* at 4:17-20.

155. Figure 1 of the ’248 patent, reproduced below, shows an applet server computer 10 with an on-board applet server manager 22 that coordinates requests from clients and builds applets not stored in the applet server’s cache.

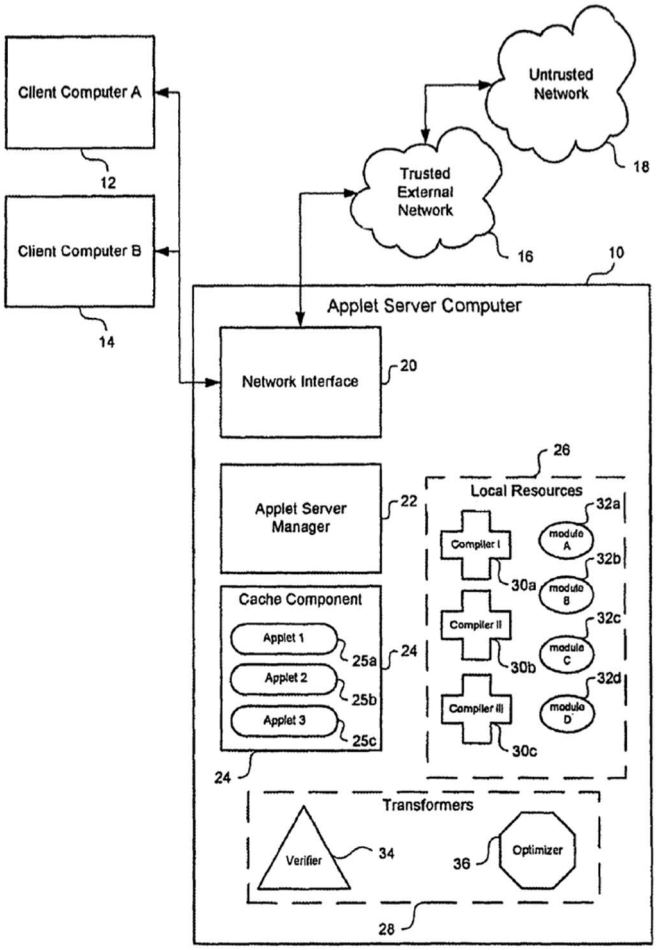


Fig. 1

Id. at Fig. 1, 3:55-4:28.

156. This applet building and downloading scheme was known before the priority date of the '248 Patent, as I discuss in more detail below.

B. Prosecution History

157. The '248 Patent was originally filed with thirteen claims (1-13), including three independent claims (1, 5, and 9) which the Examiner rejected as obvious over U.S. Patent No. 5,805,829 ("Cohen") in view of U.S. Patent No. 5,828,840 ("Cowan"). EX1004 ('248 Prosecution History) at 45-47.

158. In response, the applicant did not amend the claims (*id.* at 60-63), and instead argued that "Cohen does not teach, disclose or suggest receiving on the server a request … in a form selected from a plurality of forms. The applet loader blindly accepts a name containing the URL and transmits it to the server. As such, Cohen is silent regarding formatting the request, in a 'form selected from a plurality of forms' as recited in independent claim 1." *Id.* at 65-66. The applicant continued that "Cohen does not disclose the ability to compile an applet according to a form selected from a plurality of forms. Cohen's server simply transmits 'intermediate code' to the server." *Id.* at 66. Accordingly, according to the applicant, "the server disclosed in Cohen does not transmit applets according to a selected format chosen from a plurality of formats." *Id.*

159. The applicant also attempted to distinguish Cowan, saying "the Cowan reference does not remedy the deficiencies identified in Cohen regarding independent claim 1." *Id.* Instead, the applicant argued, "Cowan simply discloses that the server may determine whether a client can execute Java applications directly (a network terminal) or not (non-network terminals). Based on the determination, the server executes the application for non-network terminals, were [sic] as for network terminals it appears as though the client executes the code." *Id.* at 66-67. Thus, per the applicant, "Cowan does not receive a request … in a form selected from a plurality of forms, as recited in independent claim 1."

160. Finally, "Applicant submit[ted] that independent claims 1, 5 and 9 are patentably distinct from the cited references for at least these reasons" and that the dependent claims "for at least similar reasons." *Id.* at 67.

161. The Examiner allowed the claims in response, finding "claims 1, 5 and 9 are directed to managing requests to a server computer for applets in a client server environment, compiling them and transmitting them to client's system in selected form and byte-code verification are server based and thereby providing more efficient use of resources and a flexible mechanism for instituting enterprise-wise security policies. Further the server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network." *Id.* at 85.

162. In the grandchild patent application, U.S. Patent Application No. 11/933,161 (the "'161 Application") which became U.S. Patent No. 8,056,075, the applicant originally filed a single claim, which is a verbatim copy of claim 1 of the '248 patent. *Compare* EX1022 (Original Claim 1 of the '161 Application) *with* EX1001 at 7:20-34. The '075 Patent examiner rejected this claim as anticipated by U.S. Patent No. 5,923,885 to Johnson et al. ("Johnson"). EX1023 (Feb. 14, 2011, Non-Final Rejection of '161 Application) at 3-7. It does not appear the '248 Patent examiner ever considered Johnson during examination of the application leading to

the ’248 Patent. *See* EX1004 (’248 Prosecution History) at 87-89. Below, I opine that at least Johnson renders claims 1 and 9 of the ’248 Patent obvious.

C. Person of Ordinary Skill in the Art

163. I understand that an assessment of claims of the ’248 Patent should be undertaken from the perspective of a Person of Ordinary Skill in the Art (“POSA”) as of the priority date, which I have been asked to apply as March 18, 1998. I have also been advised that to determine the appropriate level of a person having ordinary skill in the art, the following factors may be considered: (1) the types of problems encountered by those working in the field and prior art solutions thereto; (2) the sophistication of the technology in question, and the rapidity with which innovations occur in the field; (3) the educational level of active workers in the field; and (4) the educational level of the inventor.

164. In my opinion, a POSA at the time of the priority date of the ’248 Patent would have had a Bachelor’s Degree in computer science, computer engineering, or equivalent degree, and approximately two years of experience working in the computer science or engineering field. Additional experience might substitute for education and vice versa. Such a POSA would have had knowledge of design considerations known in the industry and would have been familiar with then-existing products and solutions. To that end, a POSA in March 1998 would have

been familiar with compilers, programming languages, networking, and web systems and applications.

165. My opinions regarding the level of ordinary skill in the art are based on, among other things, my more than 35 years of experience in client-server architecture design, my understanding of the basic qualifications that would be relevant to an engineer or computer scientist tasked with investigating and designing client-server architectures and components therefor, my knowledge and experience with the capabilities of such systems by 1998, and my familiarity with the backgrounds of colleagues, both past and present. Although my qualifications and experience exceed those of the hypothetical person having ordinary skill in the art defined above, my analysis and opinions regarding the '075 Patent have been based on the perspective of a person of ordinary skill in the art as of March 1998. I am able to take on that perspective based on my years of experience.

VI. OVERVIEW OF THE PRIOR ART

A. Johnson (EX1011)

166. U.S. Patent No. 5,923,885 to Johnson et al. ("Johnson") is titled "Acquisition and operation of remotely loaded software using applet modification of browser software." EX1011 (Johnson) at cover. Johnson was applied for on October 31, 1996 and was issued July 13, 1999. *Id.*

167. Johnson details the rise in popularity of the personal computer, Internet, and World Wide Web, which "has prompted many organizations to form internal computer networks—the so-called 'intranets'." *Id.* at 1:30-61. "Using the World Wide Web, users can view text, graphics and hear sounds from sites all over the globe. In addition users can also download new software, or software capable of modifying programs already installed on the client computers. These same features available to users of the World Wide Web on the Internet can also be provided to users of a local network through an intranet." *Id.* at 2:13-22.

168. "However, the use of modern networks for such a widespread distribution of information and software points to a drawback regarding the highly decentralized nature of personal computers. Because users have so much control over their computing environments, and because of the need to maintain individual copies of software packages on individual computers, the management of computer networks is becoming increasingly difficult. With so many remote computers connected to each other over computer networks, administrators are finding it difficult to insure [sic] that each user has a computer system that is configured to operate reliably and efficiently." *Id.* at 2:23-34.

169. According to Johnson, a particular challenge "is the need to support software that runs on different computer platforms, such as UNIX, Macintosh, and Windows, as several platforms are often used in organizations to perform different

functions. … This forces organizations to purchase multiple copies of the same software application to accommodate those uses that are common to all platforms (e.g., word-processing or database access)." *Id.* at 2:40-54.

170. Accordingly, Johnson teaches "a method, system, and software for installing and operating selected software applications on a client computer that is in communication with a server computer on a computer network. … More importantly, the methods, systems, and apparatus provided by the present invention allow for the distribution of software that is platform-independent; thereby freeing network and system administrators—and users—from the frustrations and difficulties created by the need to manage various platform-dependent versions of software." *Id.* at 2:66-3:14, Abstract.

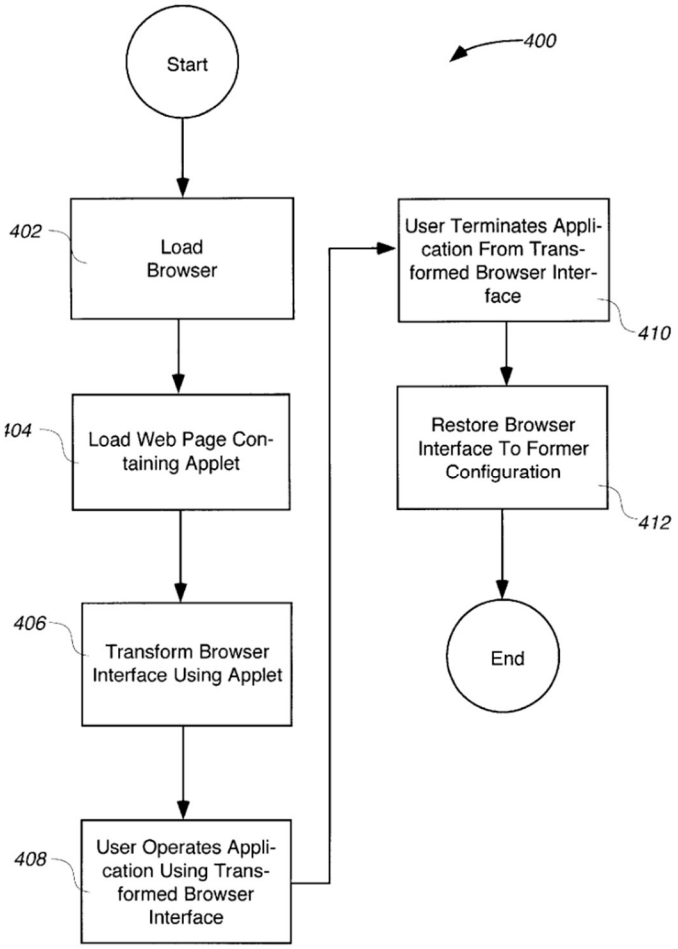
171. "In one embodiment, browser software is provided on the client computer that includes a browser user interface and is effective to identify and download selected software applications from the server onto the client computer for execution thereby." *Id.* at Abstract, 3:15-25. "In another embodiment, the selected software is an applet, and, in a more specific embodiment, an applet written in the Java programming language." *Id.* at 3:40-56.

172. Johnson explains "[t]he construction, maintenance, and operation of the server, router, and various client machines will be well known to those of skill in the art." *Id.* at 6:6-14. That said, "the server provides data and software that is

accessible to the various clients which are in communication with the server, either directly or through a device such as a router." *Id.* "In particular, … the server 208 will be configured to provide data that is compatible with browser software such as that used to view data on the World Wide Web as described above." *Id.*

173. "In one embodiment, the server and clients are configured to exchange not only data but computer software applications and, more particularly, software in the form of 'applets', such as those written in the Java programming language available from Sun Microsystems of Mountain View, Calif." *Id.* at 6:14-21. In Johnson, the term "'applet' … refers to a software code that is executed in the context of software already running on the remote computer." *Id.* at 6:22-24. *See also* IV.B above.

174. "One embodiment of a method by which the aforementioned transformation is accomplished is shown at 400 in FIG. 4." *Id.* at 8:13-15.



Id. at Fig. 4.

175. Johnson describes that the browser software can either be "installed on the local client machine and [] executed by the user directly on the client" or "the browser can be provided remotely across the network in which case the client machine is configured to respond more like a terminal than a stand-alone computing platform." *Id.* at 8:16-21. "Upon loading the browser, the user loads a web page which contains the desired applet to be executed at step 404. The loading of such web pages is performed using methods that are familiar to those of skill in the art.

Upon loading the desired applet, the browser interface is transformed at step 406 such that a new interface is provided to the user which includes functionalities and displays that are consistent with the functionality of the applet as illustrated above with respect to FIGS. 3A and 3B." *Id.* at 8:24-32.

176. "At step 502 the applet which is now executing on the client computer, having been loaded from the remote server using standard techniques, acquires new properties settings that will be used to define the new functionalities to be presented in the transformed browser interface. The settings are typically stored on the server on which the applet is stored. … At step 504 the applet communicates the new properties settings to the browser as will be described in greater detail below. Upon the communication of the new properties settings to the browser, at step 506 the applet requests the browser to update its interface using the new settings. At step 508 the browser transforms its interface as it updates its interface through the new properties settings that have been provided by the applet. This transformation relies on the same mechanisms used to construct the original browser interface and therefore will not be described in greater detail as such operations will be familiar of those of skill in the art." *Id.* at 8:43-65.

177. "At step 702 a determination is made as to whether the applet or the browser is responsible for maintaining a copy of the original browser interface settings. If the applet has the original settings, then those settings are communicated

to the browser, e.g., using the methods just described with respect to FIG. 6, or by redirecting the browser to a copy of the original properties table that has been placed in memory at a location different from the modified original table. At step 706 the applet then sends a[n] 'update interface' message to the browser to cause the browser to consult the copy of the original table to thereby restore its original settings and interface." *Id.* at 9:50-10:5.

178. "A data transfer communication link is established between the client and server computers across the network, a desired software application is selected using the browser, and the desired software application is transmitted across the network from the server to the client." *Id.* at Abstract, Figs. 4, 5, 7.

B. Parthasarathy (EX1007)

179. U.S. Patent 6,347,398 to Parthasarathy ("Parthasarathy") is titled "Automatic software downloading from a computer network." EX1007 (Parthasarathy), cover. Parthasarathy was applied for on November 8, 1999, and claims priority to U.S. Patent Application No. 08/764,040 filed December 12, 1996. *Id.*

180. According to Parthasarathy, "[a]s the content of information stored on the Internet and World Wide Web evolves, it is desirable to insert dynamic multimedia components (e.g., animated buttons, scrolling banners, blinking lights, bouncing or spinning objects, high quality music, and full motion video) and

interactive multimedia components (e.g., a computer game sent over the Internet and played against one or more other users, or a remote computer) into a HTML document" which is accompanied by "a number of problems currently associated with using dynamic and pseudo-interactive multimedia components in a HTML document." *Id.* at 1:60-2:7, Abstract. To enable dynamic content, "a user typically must request the dynamic multimedia interaction by selecting a remote link with input from a keyboard, electronic mouse or other pointing device." *Id.* at 2:5-14, 1:60-2:2.

181. Traditionally, browsers have utilized Common Gateway Interface or CGI "to invoke a remote software application to provide pseudo-interactive multimedia content." *Id.* at 2:43-57. Executing CGI's applications, as Parthasarathy explains, are "typically a slow and cumbersome interface process that also requires input (e.g., a mouse click) from a user … and are executed on a remote computer." *Id.* at 2:54-57. Thus, Parthasarathy appreciates that a POSA would recognize the desire to reduce the load on remote (Web) servers. *Id.* at 2:49-57.

182. To enable dynamic content, Parthasarathy provides a method to "locate, download, verify, install, register, and display a computer software component from a computer network like the Internet." *Id.* at 2:65-67. An example network environment for this software distribution is provided below:

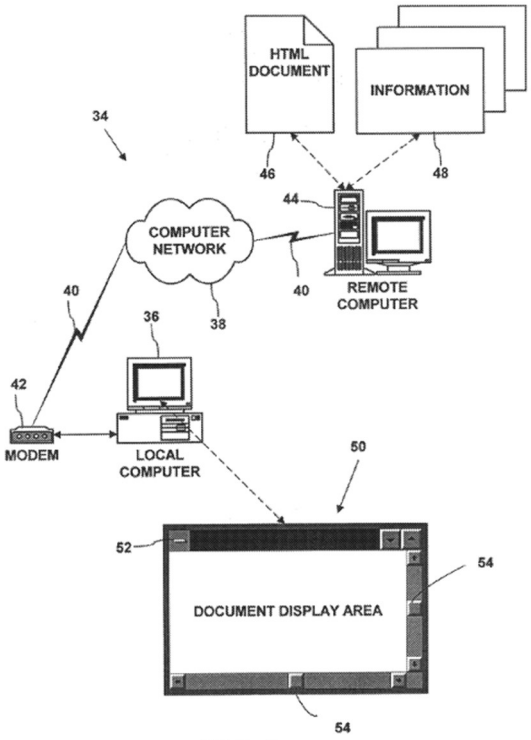


FIG. 2

Id. at Fig. 2.

183. The network environment includes "a local computer 36 (e.g., computer system 10 shown in FIG. 1) [that] runs software, referred to herein as a 'browser,' for unified browsing of electronic documents and other data from local sources (e.g., the memory system 16 of FIG. 1) and from a computer network 38 (e.g., the Internet, an intranet)." *Id.* at 6:44-47. "Documents for browsing with the illustrated browser can reside as files of a file system stored in the computer's secondary storage 32 (FIG. 1), or reside as resources at a remote computer 44 (also referred to as a 'site' or 'store') connected to the computer network 38, such as a World Wide Web site on the Internet." *Id.* at 7:3-8.

184. One type of network application program, network browsers, use various protocols such as Hyper Text Transfer Protocol ("HTTP") to access data on the World-Wide-Web ("Web"). *Id.* at 1:33-46 ("There are many types of client network applications known in the art …. These network browsers send network requests via the File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Hyper Text Transfer Protocol (HTTP), Gopher document protocol and others."). "The World Wide Web is an information service on the Internet containing documents created in the Hyper Text Markup Language (HTML)." *Id.* at 1:46-56. The software components that are downloaded in Parthasarathy can be written and developed as "an ActiveX Control, a Netscape ONE Plugin, a Java class library," or other formats and can work on a wide variety of operating systems such as Windows, Mac OS, or UNIX. *Id.* at 3:15-21, 5:22-26, 18:52-56.

185. Parthasarathy's process of acquiring software components is described as follows: "When a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local computer 36. If the desired software component is already installed on the local computer 36, then the desired software component on the local computer 34 is checked to see if a more recent version is available. If a more recent version of the desired software component is available, or if the desired computer software

component is not available on the local computer 34, then it is downloaded in an asynchronous manner by the data download module 60. The software component download module 58 uses a Uniform Resource Locator (URL) or a network search path to locate a desired software component." *Id.* at 8:19-34. The software is automatically located, downloaded, verified, installed, registered, and displayed by a software download control module. *Id.* at 2:65-67, 3:5-14, 4:1-2, 8:19-29.

186. The software that is downloaded is verified on the local computer before being installed using digital signatures or other forms of verification. *Id.* at 8:36-47, 9:36-38 ("However, security measures other than digital signatures and digital certificates could also be used to verify a software component.").

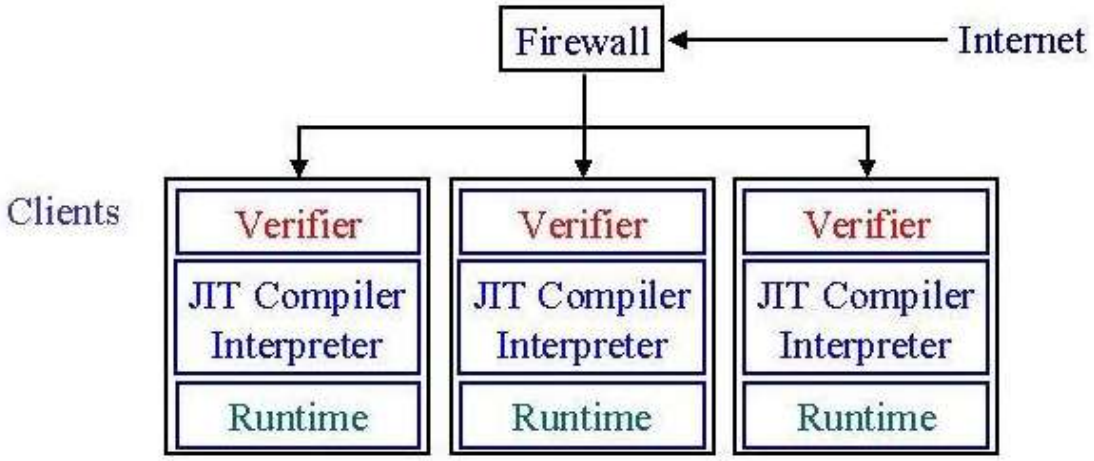
C. Kimera (EX1008)

187. Kimera is a publication of a system architecture developed by Emin Gün Sirer, Sean McDirmid, and Brian Bershad, who were academic researchers in the Department of Computer Science and Engineering at the University of Washington. EX1008 (Kimera), 2. Kimera lists a copyright date of 1997. *Id.* I have been asked to assume Kimera is prior art to the '075 Patent.

188. Kimera describes "an alternative way of structuring Java systems, in which verification and compilation are performed in centralized servers." *Id.* at 1. Kimera recognizes that a modern-day computing system (as of 1997) "has to provide its own verifier, interpreter and optionally a just-in-time compiler. The resource

requirements for a Java client are consequently very high." *Id.* The high resource requirements "makes Java unsuitable for small mobile devices." *Id.* In my opinion, this would have also made client systems slower because each client needs to compile its own resources, even if different clients across the system were compiling the same data.

189. In addition, including this functionality at the individual client device level exposed the system to security vulnerabilities, because "security is discretionary under such a point-to-point model" and "any individual within an organization may disable some or all of the checks performed on imported code." *Id.* "Furthermore, the current structure of Java based systems makes security management at the level of an enterprise almost impossible." *Id.* If any one end-node compromises the verification or security, that "Java end-node becomes a beachhead for further security attacks." *Id.* In my opinion, this concern would have been amplified as users in 1997-1998 would have often been willing to disable security functionality to gain performance.

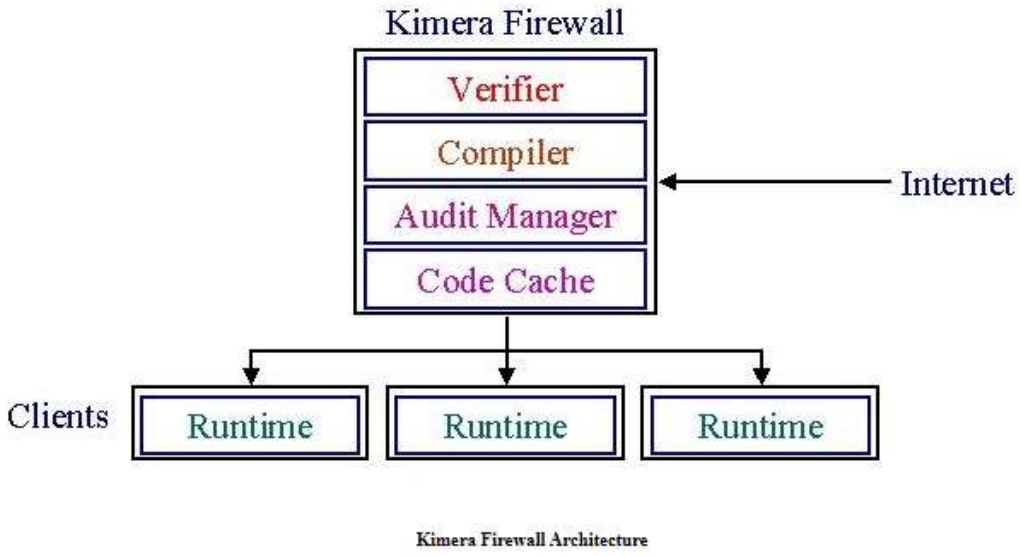


Current Java architectures are point-to-point, leaving crucial system services up to the discretion of clients.

Id. at 1 ("Kimera Figure 1").[1]

190. To overcome these shortfalls, Kimera proposes "an alternative way of structuring Java systems, in which verification and compilation are performed in centralized servers." *Id.* According to Kimera, the benefits of this architecture are evident; the system is "easy to administer, provides enterprise-wide security, supports safe auditing, and reduces Java client resource requirements." *Id.*

[1] Kimera does not number its two figures. For ease of discussion in this entire declaration, I refer to the first figure (current architecture) as "Kimera Figure 1" and the second figure (Kimera architecture) as "Kimera Figure 2".



Kimera Firewall Architecture

Id. at 1 ("Kimera Figure 2").

191. In addition, hosting verification services at a centralized point "enables timely security upgrades, especially for small, mobile clients where the Java engine may be hard-coded in ROM" and where the "security upgrade affects only the central firewall, and the change does not require any end-user cooperation." *Id.* at 1, 2. Kimera then teaches that "[c]aching the results of recent compilations in the central compilation server can be used to further reduce the latency of binary translation." *Id.* at 2. In my opinion, this would have also sped up the whole system because the benefits of "reduc[ing] the latency of binary translation" would be realized for all clients connected to the Firewall/server and because the "results of recent compilations" would be available to all of the clients across the network. *Id.* at 2. The net result is a vastly simplified "Java end-point [that] would consist of only a

minimal runtime environment" because the centralized Firewall/server eliminates "the need for a JIT compiler, verifier, [and] interpreter" on each client computer. *Id.*

192. Kimera states that "techniques such as just-in-time compilation are not suitable" to this architecture and that "[i]nstead, compiling incoming Java code ahead of time would reduce communication requirements and enable foreign code to run natively." *Id.* In my opinion, just-in-time compilation of Java code is fairly slow, and as a result, to avoid excessive communication between client and server, all compilation in Kimera of incoming code (from remote web servers) would have been finished before the compiled code was sent to the client. Incoming code is simply code that is received at the Firewall/server in response to a client request. It would have been impossible, or at least near impossible, to identify and compile every resource across the Internet, and then cache each compiled resource locally.

D. Fowlow (EX1009)

193. U.S. Patent 6,260,078 to Fowlow ("Fowlow") is titled "Using a distributed object system to find and download Java-based applications." EX1009 (Fowlow), cover. Fowlow was filed on July 3, 1996, and issued on July 10, 2001. *Id.*

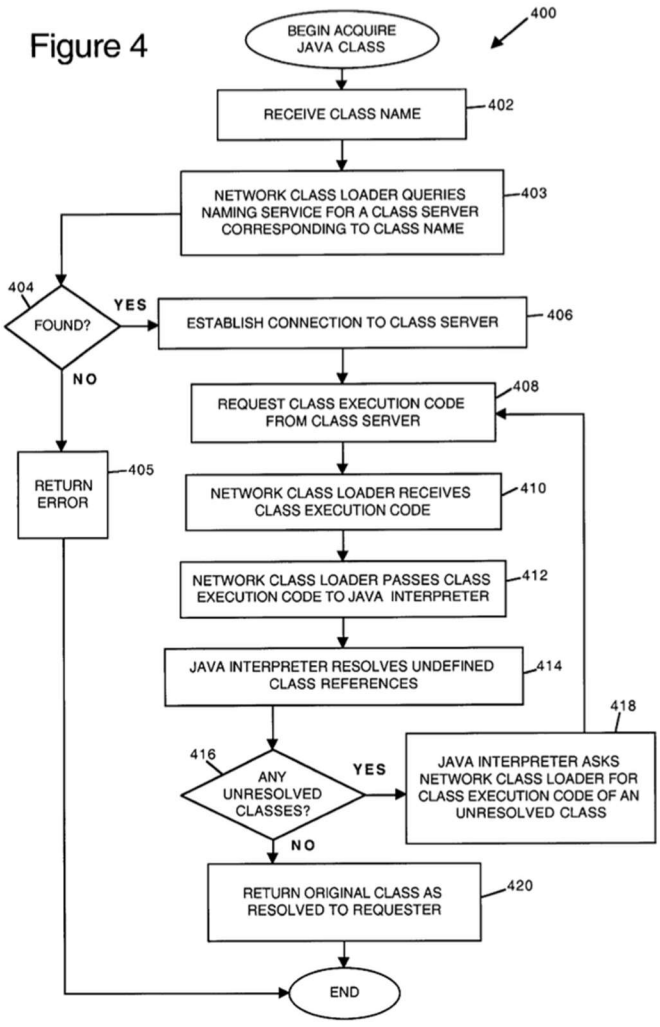
194. The original assignee of Fowlow was Sun Microsystems, Inc., which invented and developed the Java programming language. *Id.* at cover, 1:39-43. Fowlow describes "a client enabled to load and run Java applets in a distributed

object computing system." *Id.* at Abstract, Fig. 2, 10:15-65, 11:21-27. A POSA reading Fowlow would have been well aware of a wide variety of aspects of Java-based systems, including but not limited to: the Java programming language, Java applets, Java interpreters on Java-enabled client devices, just-in-time compilation for Java classes, and the like. *Id.*

195. Fowlow is directed to a method of "retriev[ing] needed Java classes in a location-independent manner from various class servers in the system" in response to "the client request[ing] a portion of this applet execution code from a found applet server by using [an] object request broker." *Id.* at Abstract, 3:35-41, 10:15-65, Figs. 2, 4. The applets and class files that make up the applets are stored on servers in the distributed object system. *Id.* at 11:40-46. Fowlow "provides a mechanism by which a requesting client may acquire the classes it needs within a distributed object system in order to run a particular applet. In one embodiment, this mechanism uses an object request broker (ORB) in order to communicate with one or more class servers that provide access to the class files needed. Thus, an ORB of a distributed object system is used to find the class files needed in a location independent manner." *Id.* at 10:15-36, 11:21-33.

196. Client devices can download applets or class files from the server or servers via parallel or series request. *Id.* at 13:18-14:3 ("if there are one or more unresolved classes, then in step 418 the Java interpreter asks the NCL for the

execution code of a first unresolved class. From step 418 the procedure loops back to step 408 in which the NCL requests from the class server the appropriate class execution code. In this fashion, this portion of FIG. 4 may loop through steps 408 to 418 until all execution code has been retrieved for all unresolved classes."). More specifically, a client requests a particular application that is made up of a variety of class files that "may be located on any machine within a distributed object system. *Id.* at 11:40-46, 3:32-38 ("initially a client queries the object request broker to determine if there is an applet server available within the system that may be used to obtain particular applet execution code"), 10:15-36, 11:21-33. The "applet server is then able to return this portion of applet execution code to the client by way of the object request broker." *Id.* at 3:40-42, 13:43-48. "[O]nce the portion of applet execution code is returned to the client, the client determines whether the portion of code contains any unresolved references to any other portion of the applet execution code." *Id.* at 4:3-6. "If there are unresolved references, the client requests additional applet execution code corresponding to these unresolved references." *Id.* at 4:8-11, 13:48-14:3. These missing classes are called "unresolved classes" or "unresolved dependencies." *Id.*; *Id.* at Abstract, 1:67-2:3. The applet download process taught in Fowlow is depicted below:



Id. at Fig. 4; *see also Id.* at 11:57-12:11.

197. In my opinion, this automatic, serial-request process enables client devices to download Java applications in a faster, more efficient manner.

198. Fowlow's applet retrieval process indicates that downloading applets, such as Java applets, happens in response to a client request. *Id.* at 3:32-35 ("In a method aspect, initially a client queries the object request broker to determine if there

is an applet server available within the system that may be used to obtain particular applet execution code."), 10:15-36, 11:21-33, 11:40-46.

199. Fowlow uses an intermediary broker, which acts as a proxy of sorts, to mediate the requests and responses between client devices and the many servers hosting Java classes and other applet data. *Id.* at 3:30-36 ("In another step, the client requests a portion of this applet execution code from a found applet server by using the object request broker."), 10:15-36 ("Thus, an ORB of a distributed object system is used to find the class files needed in a location independent manner. One advantage to using an ORB is that these class files and their associated class server may be located anywhere within the distributed object system yet may still be found by the client."), 11:21-33, 13:43-48. *See also* §IV.A above.

200. Fowlow explains that Java code may be "executed by a remote interpreter" and that this code "may also be compiled before execution, or may even undergo a combination of interpretation/compilation before execution." *Id.* at 10:50-56. As noted in §IV.DI above, a POSA would have been aware of just-in-time or JIT compilation in addition to the remote compilation described in Fowlow. *Id.*

201. Fowlow also notes that "Java may be running on a single computer or on a number of computers within a network." *Id.* at 1:55-57.

VII. OVERVIEW OF THE CLAIMS

202. I have been asked to give opinions about whether the elements of claims 1-13 of the '248 Patent are disclosed, taught, and/or suggested by Johnson, Parthasarathy, Kimera, and Fowlow. Below, I have reproduced those claims in their entirety:

[1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

[1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

[1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and

[1c] c) transmitting said applet in said selected form to said client computer.

[2] The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.

[3] The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said selected form to said server computer.

[4] The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to transmit said applet in said selected form to said server computer

[5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:

[5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms;

[5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer;

[5c] c) if said applet is not stored in said selected form in said local cache,

[5d] compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and

[5e] transmitting said applet in said selected form to said client computer.

[6] The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.

[7] The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said form to said server computer.

[8] The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resources, and directing said external resource to transmit said applet in said selected form to said server computer.

[9] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

[9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

[9b] b) compiling an applet program module into an intermediate form program module;

[9c] c) transmitting said applet in said selected form to said client computer.

[10] The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.

[11] The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring computer programs and encrypting computer programs.

[12] The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

[13] The method of claim 10, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

VIII. THE CHALLENGE GROUNDS

203. Below is a table showing the references and combinations of references I have considered in forming my opinion, and the claims that I have analyzed for each:

Ground	Basis	Relied-On References	Claim(s)
1	Obviousness	Johnson	1, 9

Ground	Basis	Relied-On References	Claim(s)
2	Obviousness	Parthasarathy and Kimera	1-13
3	Obviousness	Fowlow and Kimera	1-13

204. As discussed, §III.B above sets forth my understanding of obviousness.

A. Ground 1: Claims 1 and 9 are Obvious in view of Johnson

205. Independent claims 1 and 9 are obvious over Johnson. Below I explain how each element of claims 1 and 9 are disclosed, taught, and/or suggested by Johnson.

1. Independent Claim 1

206. Claim 1 is obvious in view of Johnson. Below I explain how each element of claim 1 is disclosed, taught, and/or suggested by Johnson.

> *a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:*

207. I have been asked to provide an analysis of this claim element, which I understand is the preamble. However, I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 1 is limiting, it is my opinion that it is disclosed by Johnson. I understand that the USPTO has also found claim 1 is obvious over Johnson. *See* §V.B above.

208. Johnson discloses "[m]ethods, systems, and software for installing and operating selected software applications on a client computer that is in communication with a server computer on a computer network." EX1011 (Johnson) at Abstract, 2:66-3:14. Johnson describes that [o]ne embodiment of an intranet … is provided in FIG. 2 at 200. A typical intranet includes a server 202 which is coupled with clients 204 and 206." *Id.* at 5:57-6:5, Fig. 2.

209. "Typically the server provides data and software that is accessible to the various clients which are in communication with the server." *Id.* at 6:6-32. "In one embodiment, the server and clients are configured to exchange not only data but computer software applications and, more particularly, software in the form of 'applets', such as those written in the Java programming language." *Id.* at 6:16-21. Johnson describes an applet as "a software code that is executed in the context of software already running on the remote computer." *Id.* at 6:22-24.

210. "Using the method system and software provided by the present invention, users can select software [including applets] from central sources and execute that software on their local machines as if that software was installed originally on those local machines." *Id.* at 12:3-14, 3:14-25 ("According to one embodiment of the method of the invention, browser software is provided on the computer. The browser software includes a browser user interface and is effective to identify and download selected software applications for execution. **A desired**

software application is selected using the browser."). A POSA would have understood that Johnson's network manages requests to a server computer for applets in a client server environment because users on a remote computer select and download applets from a remote computer. *See also* §IV.B above.

211. Johnson further discloses that the request for software or an applet specifies one form of the applet out of a plurality of forms of the applet. Specifically, Johnson discloses multiple "computer platforms," noting that "the use of modern networks for such a widespread distribution of information and software points to a drawback regarding the highly decentralized nature of personal computers." EX1011 (Johnson) at 2:23-54. This leads to "[p]articular problems includ[ing] the maintenance of current software revisions, modifications made to the operating system and software by the users, the installation of extraneous software on the local computer which is not required for performing the central tasks of the user, and the need to buy multiple copies of a single software package even if that package is used only sparingly. More important is the need to support software that runs on different computer platforms, such as UNIX, Macintosh, and Windows, as several platforms are often used in organizations to perform different functions." *Id.*

212. In addition to different computer platforms, a POSA would have known that different browsers can exist on those different platforms. Johnson lists "HotJava from JavaSoft of Cupertino, Calif., Netscape Navigator available from Netscape

Communications, Inc. of Mountain View, Calif. and Microsoft Explorer available from Microsoft Corporation of Redmond, Wash" as example browsers that can be used. EX1011 (Johnson) at 7:12-19.

213. A POSA would have also known that executable code compiled for one computer platform and browser combination would not execute on another computer platform or browser without being re-compiled for that specific platform-browser combination. *See* §§IV.D aboveIV.F above. "Although only a few embodiments of the present invention have been described, it should be understood that the present invention may be embodied in many other specific forms without departing from the spirit or scope of the present invention. By way of example, although the invention has been described in terms of a Java GUI and a HotJava presentation engine, any suitable GUI and presentation engine which are platform-independent may be implemented." *Id.* at 14:3-11.

214. Therefore, because the requested applet would be specific to a platform and browser combination, in my opinion a POSA would have understood that the request for that applet would specify the specific form of the applet capable of being executed on the requesting machine. It would make little sense for, say, an Internet Explorer browser running on Windows on an Intel i386 architecture, to request an applet for a Netscape Navigator browser, running on SunOS/Solaris using a SPARC processor—it will not run.

215. Finally, Johnson's claim 27 and its dependent claims recite "A method for installing and operating a selected software application on a client computer, said software application having a corresponding software application user interface, and said software application residing on a server computer that is coupled with said client computer through a computer network, the method comprising the steps of: providing browser software on said client computer, said browser software including a browser user interface and said browser software being effective to identify and download a selected applet onto said client computer for execution, said selected applet having an application user interface; … selecting said selected applet using said browser software … ." *Id.* at 17:29-52.

216. Johnson's claim 28 adds that "said applet is an applet written in the Java programming language." *Id.* at 17:53-54.

217. A POSA would have understood this claimed method as teaching a user on a client computer, having a particular operating system/platform and browser combination, requesting an applet from a server computer, and specifying the form of the applet (*e.g.*, an applet for the specific computer platform/browser combination, or a platform-independent form of the applet such as a Java applet).

218. In sum, in my opinion, to the extent the preamble is limiting, Johnson teaches a method operating on a computer system [*e.g.*, a network for installing and operating selected software applications] for managing requests to a server computer

[*e.g.*, server computer 208] for applets [*e.g.*, applets or "software code that is executed in the context of software already running on the remote computer"] in a client server environment [*e.g.*, the user request from a client computer to a remote server computer for a software application] wherein each request for an applet specifies one form of the applet [*e.g.*, the version capable of running on the specific client platform and browser] out of a plurality [of] forms of the applet [*e.g.*, the versions for each computer platform and/or browser combination].

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

219. Johnson discloses receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms. Johnson's network functions over an intranet having "a server 202 which is coupled with clients 204 and 206." EX1011 (Johnson) at 5:57-6:5, Fig. 2. "[T]he server and clients are configured to exchange not only data but computer software applications and, more particularly, software in the form of 'applets.'" *Id.* at 6:17-24. Using Johnson's network, "users can select software from central sources and execute that software on their local machines." *Id.* at 12:5-7, 3:15-25.

220. A POSA would have understood that when a user selects a desired applet, the client computer sends a request to the server (which is received by the server) for the specified applet. Further, as explained in §VIII.A.1.a above, the user

request specifies the form of the applet (the version capable of running on the specific client platform and browser) from of a plurality of forms. *Id.* at 2:23-54 (listing different computer platforms), 7:12-19 (listing various browsers).

221. In sum, in my opinion, Johnson teaches receiving on said server computer a request [*e.g.*, a software request] from a client computer [*e.g.*, client computers 204, 206, user 214] for an applet in a form [*e.g.*, the version capable of running on the specific client platform and browser] selected from a plurality [of] forms [*e.g.*, the versions for each computer platform and/or browser combination].

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and

222. As an initial matter, in my opinion, the term "source module" should not be limited merely to human written text-based source code, but to any original or intermediate forms that can pass through a compiler tool-suite to produce another form (and possibly a final binary executable, *i.e.*, the requested form). As I explained in §IV.D above, original source code is transformed using multiple stages by a compiler "wrapper," which in turn invokes sub-programs to take an input from one stage, process it, and produce an output suitable for the next stage of the compiler suite. Each of those intermediate stages can meet the "source module" that a "compiler … acts on."

223. Johnson discloses compiling said applet into said selected form from a local resource. "[T]he transfer of the selected software application from the remote computer is effective to transmute the user interface of the browser into the user interface of the software application." EX1011 (Johnson) at 4:29-32. Then, "the software application is executed by the computer." *Id.* at 4:12-17.

224. A POSA would have understood that the process of transmuting (transforming) the code is compiling it from one form to another, namely, from source code to executable code. Moreover, a POSA would have understood that for a compiler to successfully work, it necessarily must have the right local resources such as memory, CPU, an installed operating system environment, libraries, and an assortment of auxiliary tools and sub-programs. *See* §IV.D above.

225. Notably, Johnson teaches that "[i]n one embodiment, the browser software is installed on the local client machine and is executed by the user directly from the client. Alternatively, the browser can be provided remotely across the network in which case the client machine is configured to respond more like a terminal than a stand-alone computing platform." *Id.* at 8:16-21. A POSA would have understood that in both embodiments, the server is compiling software into executable applets using resources local to the server. In the first embodiment, the browser software is compiled into a browser-executable form that is unique to the

requesting platform and browser, which is then sent to the client where it is executed by the browser. *Id.*

226. In the alternative, terminal-like embodiment, a POSA would have understood that the server which hosts the browser would have compiled the applet into the selected form (*e.g.*, a server-executable form) using compilation resources at (*i.e.*, local to) the server and then executing the applet at that server for the client to display. *Id.* Here, the browser is running on the server itself, but transmits display commands to a (client) terminal used by the user. Such terminals are not full-fledged computers or workstations; they are often called "thin clients" or "thin displays" because they require little resources and can run more efficiently (at the cost of a lower ability to run independently, as thin clients require an operating server). A "thin client" is, "[i]n a client/server architecture, a client computer that performs little or no data processing. The processing is instead performed by the server. *See also* client/server architecture, fat server, thin server. *Compare* fat client." EX1014 (Microsoft Computer Dictionary) at 466.

227. In my opinion, this terminal-like or "thin client" embodiment shows that a POSA would have known how to include processing capability, including the ability to compile applets, at the server. Specifically, in this embodiment, the server would have been performing compilation services in order to prepare the applet for execution, and to execute the applet at the server before sending the display output

to the thin client. A POSA would have known that these compilation services could have also been used to compile applets into the form specific to the requesting client-browser architecture, which would have been the form required by the first embodiment, or a fat client. Thus, compiling applets into the form selected (*e.g.*, the specific form for a particular browser and client architecture combination) from a plurality of forms at the server would have been known, or a trivial modification, to Johnson. A POSA would have also recognized an additional benefit of using thin clients: once the applets are compiled on the server, they can be cached there for quicker reuse by other thin clients of the same architecture, thus improving overall performance.

228. In sum, in my opinion, Johnson teaches compiling [*e.g.*, transmuting] said applet into said selected form [*e.g.*, the version capable of running on the specific client platform and browser] from a local resource comprising at least one source module [*e.g.*, applet in source form stored on the server computer] and one compiler [*e.g.*, necessary tools, libraries, and a working environment for compilers to function] which acts on said source module to produce said selected form [*e.g.*, the version capable of running on the specific client platform and browser].

d. [1c] c) transmitting said applet in said selected form to said client computer.

229. Johnson discloses transmitting said applet in said selected form to said client computer. Johnson discloses "[m]ethods, systems, and software for installing and operating selected software applications on a client computer that is in communication with a server computer on a computer network…." EX1011 (Johnson) at Abstract. The methods, systems, and software disclosed include a "server [that] provides data and software that is accessible to the various clients which are in communication with the server, either directly or through a device such as a router." *Id.* 6:6-24. "In particular, however, the server 208 will be configured to provide data that is compatible with browser software" including software applications and applets. *Id.*

230. "[T]he data provided by the server will be in the form of pages of data that can be examined using typical browser software. In one embodiment, the server and clients are configured to exchange not only data but computer software applications and, more particularly, software in the form of 'applets', such as those written in the Java programming language available from Sun Microsystems of Mountain View, Calif." *Id.* A POSA would have understood that any form of browser-executable applet could be transmitted to the client, not only applets written in the Java programming language (*e.g.*, ActiveX). *See* §IV.B above.

231. In sum, in my opinion, Johnson teaches transmitting said applet in said selected form [*e.g.*, the version capable of running on the specific client platform and browser] to said client computer [*e.g.*, client computers 204, 206, user 214].

232. In my opinion, Johnson discloses, or at least teaches, each limitation of claim 1 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Johnson.

2. Independent Claim 9

233. Claim 9 is obvious in view of Johnson. Below I explain how each element of claim 9 is disclosed, taught, and/or suggested by Johnson.

> *a. [9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:*

234. I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 9 is limiting, it is my opinion that a POSA would have found that Johnson renders this limitation obvious, as explained in §VIII.A.1.a above. EX1011 (Johnson) at Abstract, 2:23-54, 2:66-3:25, 5:57-6:32, 7:12-19, 12:3-14, 14:3-11, 17:29-54, Fig. 2.

b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

235. A POSA would find Parthasarathy-Kimera renders this limitation obvious, as explained in §VIII.A.1.b above. EX1011 (Johnson) at 3:15-25, 5:57-6:5, 6:17-24, 12:5-7, Fig. 2.

c. [9b] b) compiling an applet program module into an intermediate form program module;

236. A POSA would find Parthasarathy-Kimera renders this limitation obvious, as explained in §VIII.A.1.c above. EX1011 (Johnson) at 4:12-17, 4:29-32, 8:16-21. As I explained in §IV.D above, original source code is transformed using multiple stages by a compiler "wrapper," which in turn invokes sub-programs to take an input from one stage, process it, and produce an output suitable for the next intermediate stage of the compiler suite.

237. The '248 Patent describes that "[t]he intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary for each potential target computer." EX1001 at 2:43-46. "The intermediate byte-code [can but] need not necessarily be Java byte-code." *Id.* at 3:38-39. A POSA would have understood the intermediate form to refer to platform-independent code that could be adapted to any number of different computer architectures by compiling the intermediate form into binary executable form. *Id.*

at 5:23-26 ("A target compiler 52 compiles intermediate applet code 44 into an applet 54 in a processor specific format (binary) suitable for running natively on a given computing platform."); 3:27-31. *See also* §IV.D above.

238. This intermediate form program module is rendered obvious by "the methods, systems, and apparatus provided by the present invention [of Johnson, which] allow for the distribution of software that is platform-independent; thereby freeing network and system administrators—and users—from the frustrations and difficulties created by the need to manage various platform-dependent versions of software." EX1011 (Johnson) at 3:8-14, 2:40-54 (noting that organizations utilize the UNIX, Macintosh, and Windows platforms to perform different functions).

239. "Because applets, such as Java applets or an applet written in the Java programming language, can function in a platform-independent manner (Flannagan 1996) it will be appreciated that the present invention provides a way of distributing software in a platform independent manner that allows users to execute software without having that software installed on their local machines." *Id.* at 6:42-49. Thus, a POSA would have found that compiling into an intermediate form program module is obvious. *See* §IV.D above (discussing multi-stage and cross-compilers).

240. In sum, in my opinion, Johnson teaches compiling an applet program module [*e.g.*, applets or "software code that is executed in the context of software

already running on the remote computer"] into an intermediate form program module [*e.g.*, "Java applets or an applet written in the Java programming language"].

d. [9c] c) transmitting said applet in said selected form to said client computer.

241. A POSA would find that Johnson renders this limitation obvious, as explained in §VIII.A.1.d above. EX1011 (Johnson) at Abstract, 6:6-24, Fig. 2.

242. In my opinion, Johnson discloses, or at least teaches, each limitation of claim 9 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Johnson.

B. Ground 2: Claims 1-13 are Unpatentable Under 35 U.S.C. §103(a) as Obvious over Parthasarathy in view of Kimera

243. Independent claims 1, 5, 9, and dependent claims 2-4, 6-8, and 10-13 are obvious over Parthasarathy combined with Kimera. Below I explain how each element of claims 1-13 are disclosed, taught, and/or suggested by Parthasarathy in combination with Kimera.

1. Motivation to Combine Parthasarathy and Kimera

244. In my opinion, there is strong motivation to combine Parthasarathy and Kimera, including at least six discrete reasons, and incorporating the functionality of Kimera's Firewall/server into Parthasarathy's client-server interface would have been relatively easy.

245. First, in my opinion a POSA would be motivated to combine Parthasarathy and Kimera because both discuss methods and systems for downloading software, including Java class libraries and applets. *Compare* EX1007 (Parthasarathy) at 3:15-21 ("Almost any software component can be downloaded, verified, and installed via the described method and system, whether it is an ActiveX™ Control, a Netscape™ ONE Plugin, a Java™ class library … ."), 31:63-65 ("The method of claim 13 where the file of data includes HTML data, Visual Basic programming language data, or Java programming language data.") *with* EX1008 (Kimera) at 1 ("Java enables mobile programs, called applets, to be shipped from a web server to a client's machine, where they execute locally and are integrated into the client's environment."). In my opinion, a POSA would be motivated to improve one Java software system (Parthasarathy) with the functionality of another Java software system (Kimera).

246. Second, both Parthasarathy and Kimera are similarly concerned with improving overall system performance and security. Parthasarathy teaches "[a] code verification module [that] verifies that the downloaded computer software components are safe to install on a local computer by verifying a digital signature and digital certificate sent with the downloaded computer software component." EX1007 (Parthasarathy) at 3:5-12; *see also* EX1007 (Parthasarathy) at 2:65-3:27, 8:19-35, 8:66-9:3. Kimera similarly describes a "verification service" at the

Firewall/server that "performs checking of incoming code for typesafety and conformance to interfaces." EX1008 (Kimera) at 2. A POSA would have been motivated to improve Parthasarathy's code verification module, that verifies digital signatures and certificates, with Kimera's "verification service," that checks incoming code for typesafety and conformance.

247. Third, a POSA would be motivated to combine Parthasarathy and Kimera to centralize the software locating, downloading, and verifying functionality to improve system efficiency. Parthasarathy discloses methods and systems, and more specifically, modules for "downloading, verifying, installing, registering and displaying computer software components." EX1007 (Parthasarathy) at Abstract. Kimera describes a similar architecture, where software is acquired from the Internet into "centralized servers," were "verification, access control, auditing, and compilation services" are located. EX1008 (Kimera) at 1.

248. Parthasarathy's main module that accomplishes the tasks of "downloading, verifying, installing, registering and displaying computer software components" is the "software component download module 58" that manages "a data download module 60, a code verification module 62, and a code installation module 64." EX1007 (Parthasarathy) at 2:67-3:14, 8:13-19, Fig. 3. While it is not expressly defined in Parthasarathy, it is my opinion that Parthasarathy's software component download module resides on the local computer. *See Id.* at 8:20-31

(noting that the software component download module checks to see "[i]f the desired software component is already installed on the local computer 36").

249. The software component download module would need to be in constant communication with remote computers as well, such as to check "[i]f a more recent version of the desired software component is available." *Id.* at 8:20-31. Kimera explains why constant communication between client and server only serves to hamper the software download process. EX1008 (Kimera) at 2 (noting "communication overhead between the client and the compilation server would dominate any potential benefits").

250. Thus, in my opinion, a POSA considering Parthasarathy would have been motivated to look to Kimera for guidance on how to structure the system in a way that avoids overly burdensome communication between client and server (*e.g.*, reduced network resource usage). Accordingly, I believe a POSA would have been motivated to include the Kimera centralized architecture, otherwise called the Kimera Firewall, to centralize Parthasarathy's modules onto a common access point and reduce communication overhead between client and server. This would accomplish Kimera's goal of making systems "faster, cheaper, and without introducing security vulnerabilities." EX1008 (Kimera) at 1.

251. The resulting combination would include the applet locating and software downloading component module functionality of Parthasarathy loaded into

Kimera's centralized processing architecture, as demonstrated in the combined and annotated figure below:

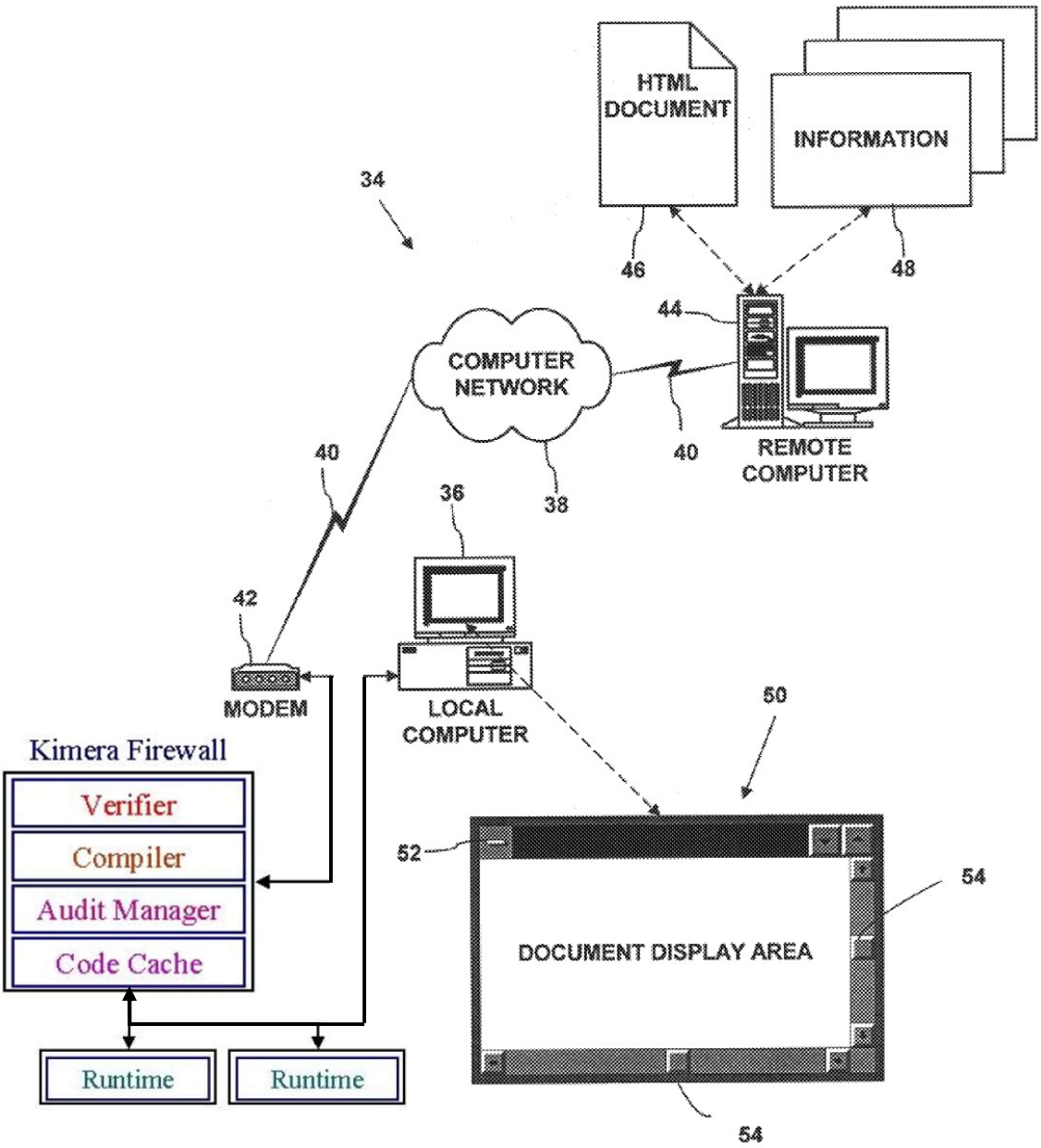


FIG. 2

EX1007 (Parthasarathy) at Fig. 2 (annotated, combined with the Kimera firewall (Kimera Fig. 2) inserted where Internet communications are in Parthasarathy, between the modem 42 and local computer 36); EX1008 (Kimera) at 1.

252. Fourth, in my opinion, this same centralization logic leading to better performance and security would apply to determining whether the desired software component is stored in the code cache. That is, the Kimera Firewall/server would assume the role of Parthasarathy's "software component download module 58," "data download module 60, [] code verification module 62, and [] code installation module 64." *Id.* at 2:67-3:14, 8:13-19, Fig. 3; *see also Id.* at 8:66-9:8, 9:33-42, Fig. 4. According to Kimera, moving these functions to a central server "enables the same services to be performed faster, cheaper, and without introducing security vulnerabilities." EX1008 (Kimera) at 1-2.

253. Fifth, in my opinion a POSA would have been motivated to combine Parthasarathy and Kimera in order to speed up a client's access to requested applets by taking advantage of Kimera's centralized "Code Cache." This centralized Code Cache caches the results of recent compilations "to further reduce the latency of binary translation" in response to future requests for the same code. *Id.* at 2. In Parthasarathy's current architecture, if N local computers (clients) ask for the same software component from a remote computer, each requested download is handled

individually and the remote computer (or the network at large) has to retrieve that code N times. This is exactly the "waste" that Kimera tried to address. *Id.* at 1.

254. In my opinion, storing compiled code in a Code Cache on Kimera's Firewall/server placed between Parthasarathy's Internet modem and any local computer would make the system even more efficient and quick, as the software components would be stored on the cache and accessed from the cache on second requests from the local computer, or even requests from other local computers in the network. In other words, including Kimera's Code Cache eliminates the external network altogether for additional requests for the same software component when that software component is stored in the Code Cache. *See* §IV.C above.

255. When a local computer in this arrangement desires a software component, it would send the request to the Kimera Firewall/server, which would check the local Code Cache for the software component, retrieve non-cached software from remote computers over the network, verify received software components, and ultimately return the software to the local computers. *See* EX1007 (Parthasarathy) at 8:20-30 (describing how "the software component download module 58 determines if the desired software component is already installed on the local computer 36.").

256. In my opinion, incorporating Kimera's Firewall/server (which already has processing functionality) into the Parthasarathy system architecture, and

particularly adding it between the modem and local computers, would be achieved without undue experimentation. Both Parthasarathy and Kimera would function nearly the same as their original disclosures (and in fact, better and faster), and the overall combination would incorporate the beneficial aspects of each reference, while minimizing the negative concerns, specifically outlined in Kimera. For example, the centralized verification scheme ensures that permanent and rapidly modifiable security is applied to all local machines. EX1008 (Kimera) at 2.

257. Sixth, even though the local computer of Parthasarathy does not need to be modified to incorporate Kimera's centralized Firewall/server, some or all local computers could be stripped to a more bare-bones runtime environment, lowering the over architecture cost in the process. *Id.* Thus, combining Parthasarathy and Kimera would result in faster, cheaper, and more secure systems, and would be accomplished without undue burden. *Id.* at 1. That is, Kimera's firewall can be "dropped in" between clients and the Internet, as it acts as a transparent proxy. *See also* §§IV.A above, IV.C above, and IV.D above.

258. It is therefore my opinion that there is strong motivation to combine these references, and that the combination would be accomplished without undue burden.

2. Independent Claim 1

a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

259. I have been asked to provide an analysis of this claim element, which I understand is the preamble. However, I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 1 is limiting, it is my opinion that it is disclosed by Parthasarathy in view of Kimera.

260. Parthasarathy teaches "a network browsing environment." EX1007 (Parthasarathy) at Abstract, 2:65-67, 4:1-2. The network browsing environment provides "a method and system to automatically locate, download, verify, install, register and display computer software components from a computer network." *Id.* The method operates in this way: "a request is made [from a local computer] to download a desired software component stored on a remote computer." *Id.* at 8:19-20. The local (client) computer is connected to the remote (server) computer through a computer network 38 (*e.g.*, the Internet). *Id.* at 4:44-47, 7:3-8, Figure 2. The method first determines if the requested software component is stored on the local computer, and if it is not, "it is downloaded in an asynchronous manner [from the remote computer] by the data download module 60." *Id.* at 8:26-31.

261. A POSA would have understood that Parthasarathy's process of locating and downloading a desired software component by a local computer from a remote computer discloses a method operating on a computer system for managing requests to a server computer for applets in a client server environment. *See* §IV.B above.

262. A POSA would have further understood that in the Parthasarathy-Kimera combination, all requests would pass through Kimera's centralized server. EX1008 (Kimera) at 1 ("Our approach to Java system structure is to break the trusted computing base into small, secure components and locate them in security firewalls. In particular, locating verification, access control, auditing, and compilation services in centralized servers enables the same services to be performed faster, cheaper, and without introducing security vulnerabilities.").

263. In addition, a POSA would have understood that, if the code cache of the centralized server had the requested software components, it could transmit those components to the client directly from the code cache. EX1008 (Kimera) at 2 ("Caching the results of recent compilations in the central compilation server can be used to further reduce the latency of binary translation."). *See also* §IV.C above.

264. Parthasarathy discloses that "[w]hen a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is

already installed on the local computer 36. If the desired software component is already installed on the local computer 36, then the desired software component on the local computer 34 is checked to see if a more recent version is available." EX1007 (Parthasarathy) at 8:20-27. "If a more recent version of the desired software component is available, or if the desired computer software component is not available on the local computer 34, then it is downloaded in an asynchronous manner by the data download module 60. The software component download module 58 uses a Uniform Resource Locator (URL) or a network search path to locate a desired software component." *Id.* at 8:27-33. A POSA would understand that in the Parthasarathy-Kimera combination, the scope of local memory would include the contents of the centralized code cache. *See also* §IV.A above.

265. Parthasarathy discloses that the request for an applet specifies the form of the applet from a plurality of forms in order to achieve platform independence. For example, Parthasarathy teaches "the CoGetClassObjectFrom URL interface using method 96 will first try locating 68 (FIG. 4) the desired control file using URL_1 through URL_m (98-102)." *Id.* at 14:25-27. Parthasarathy teaches "computer software components (e.g., a control file) are automatically located 68 using one of two designators: a URL or a network search path" and that "[t]he control file is used to determine the location of the software components on one or more remote computers 44 to download to the local computer 36." *Id.* at 9:9-21.

266. "The control file located 68 on a remote computer can be, for example, a portable executable file, a cabinet file, or an initialization file." *Id.* at 15:48-50. "The portable executable is typically an OLE control or ActiveX™ (network activation) control file (e.g., OCX), a Dynamic Link Library file (e.g., .DLL), or a executable file (.EXE), OLE control, ActiveX™, Dynamic Link Library, and executable files are known to those skilled in art." *Id.* at 15:52-57.

267. Another form of the control file, a cabinet file, "contains one or more files, all of which are downloaded together in a compressed 'cabinet.' One file in the cabinet is an initialization file (e.g., INF) providing further installation information." *Id.* at 15:56-16:4. "The initialization file also provides platform independence for HTTP and non-HTTP servers which store the desired software components. It is possible to create platform-independent setup scripts that pull desired software components from different locations depending on the desired platform. A sample platform-independent .INF file would include a text such as the following:

```
[circ3.ocx]
; lines below specify that the specified circ3.ocx (clsid, version) needs
to be installed
        on
; the system. If doesn't exist already, can be downloaded from the given
location (a
        .CAB)
file-win32-x86=file://products/release/circ3/x86/circ3.cab
file-win32-mips=file://products/release/circ3/mips/circ3.cab
```

file-mac-ppc=ignore
; the 'ignore' keyword means that this file is not needed for this platform
clsid={9DBAFCCF-592F-101B-85CE-00608CEC297B}
FileVersion=1,2,0,0"

Id. at 17:61-18:20.

268. As Parthasarathy explains, the "[t]he 'file-x-y' syntax used in the .INF file is expanded to 'file-%opersys%-%cpu%', (e.g., 'file-win32-mips' where %opersys%='win32' and %cpu%='mips') allowing the .INF file to specify multiple locations where various platform dependent modules can be found and downloaded." *Id.* at 18:15-20. In other words, "[t]he %opersys% and %cpu% attribute values … will ***specify the operating system and CPU*** for the desired platform desired software components will be executed on." *Id.* at 18:35-43. Various "valid values for %opersys% and %cpu%" are provided:

Valid values for %opersys%	Meaning
win32	32-bit Windows ® operating systems (Windows95 or Windows NT)
mac	Macintosh ® operating system
<other>	will be defined as necessary
Valid values for %cpu%	**Meaning**
x86	Intel ® x86 family of processors
ppc	Motorola ® PowerPC architecture
mips	MIPS ® architecture processors
alpha	DEC ® Alpha architecture

Id. at 18:45-57.

269. Platform independence is also achieved without initialization files via HTTP headers. Specifically, "[a]n HTTP ACCEPT header MIME request type is used to provide platform independence. The MIME request type specifies which platform the code is to run on, thus allowing platform independence when the CODEBASE attribute or the network search path is used." *Id.* at 18:64-19:3.

270. In my opinion, a POSA would have understood that the different software variations depending on the platform render obvious the claimed "plurality of forms" of the applet because each platform-specific, or operating-system-dependent version of applet is a distinct form of that applet. Parthasarathy's platform-independence disclosure is similar to Kimera's teaching that the central server "[c]ompil[es] from Java bytecode to native machine code" because a POSA would know that the native machine code needed for that platform and/or operating system is specific to that platform/operating system. EX1008 (Kimera) at 2.

271. In sum, in my opinion, to the extent the preamble is limiting, Parthasarathy-Kimera teaches a method operating on a computer system [*e.g.*, network browsing environment] for managing requests [*e.g.*, requests from a local computer for a software component] to a server computer [*e.g.*, Kimera's centralized server or Parthasarathy's remote computer] for applets [*e.g.*, a software component] in a client server environment wherein each request for an applet specifies one form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine

code] of the applet out of a plurality [of] forms of the applet [*e.g.*, Parthasarathy's forms for various operating systems and/or CPU architectures].

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

272. The Parthasarathy-Kimera combination teaches receiving a request on said server computer from a client computer for an applet in a form selected from a plurality of forms. Parthasarathy teaches "[w]hen a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local computer 36." EX1007 (Parthasarathy) at 8:20-24.

273. As discussed in §§VIII.B.1 and VIII.B.2.a above, a POSA would have understood that the request would first be received at Kimera's centralized server to determine whether the requested applet, in the requested form, was stored in the centralized server's code cache. EX1008 (Kimera) at 2. If so, the requested applet in the selected form would be transmitted to the client from the centralized server in order to "further reduce latency" in the applet download process. *Id.*

274. "If a more recent version of the desired software component is available, or if the desired computer software component is not available on the local computer 34" or in the code cache, a POSA would have understood that the request

would be passed to a network-connected server computer, such as remote computer 44 of Parthasarathy. EX1007 (Parthasarathy) at 8:20-33.

275. As discussed in §VIII.B.2.a above, Parthasarathy utilizes an initialization file or HTTP header that "specif[ies] the operating system and CPU for the desired platform desired software components will be executed on." *Id.* at 15:48-16:4, 17:61-18:20, 18:35-57, 18:64-19:3. In specifying the "operating system and CPU for the desired platform," a POSA would have understood the request as specifying a form of the applet from plurality of forms, namely, applets for a variety of operating system and CPU architectures. *Id.* at 18:35-57.

276. Further, a POSA would have understood the same platform-specifying information would accompany requests to Kimera's centralized server, to ensure the correct form of the applet was received in response to the request. *See* EX1008 (Kimera) at 2 ("Compilation from Java bytecode into native machine code is another service that is best performed at a central location."). To compile into *native* machine code, Kimera's server would need to know the architecture of the requesting machine in order to compile machine code capable of natively executing on that machine. *Id.* *See also* §IV.B above.

277. In sum, in my opinion, the Parthasarathy-Kimera combination teaches receiving on said server computer [*e.g.*, Kimera's centralized server or Parthasarathy's remote computer] a request from a client computer for an applet in

a form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] selected from a plurality [of] forms [*e.g.*, Parthasarathy's forms for various operating systems and/or CPU architectures].

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and

278. As an initial matter, in my opinion, the term "source module" should not be limited merely to human written text-based source code, but to any original or intermediate forms that can pass through a compiler tool-suite to produce another form (and possibly a final binary executable). As I explained in §IV.D above, original source code is transformed using multiple stages by a compiler "wrapper," which in turn invokes sub-programs to take an input from one stage, process it, and produce an output suitable for the next stage of the compiler suite. Each of those intermediate stages can meet the "source module" that a "compiler … acts on."

279. Moreover, a POSA would have understood that for a compiler to successfully work, it necessarily must have the right local resources such as memory, CPU, an installed operating system environment, libraries, and an assortment of auxiliary tools and sub-programs. *See* §IV.D above.

280. Kimera discloses "[c]ompilation from Java bytecode into native machine code is another service that is best performed at a central location."

EX1008 (Kimera) at 2. A POSA would have understood that the compiler of Kimera would have been used to compile platform-agnostic source material (*e.g.*, Java bytecode) into a platform-specific form of that applet or code (*e.g.*, native machine code or otherwise platform-specific code) in order for that applet to function properly on each client computer. *Id.*; *see also* §IV.D above.

281. Kimera's compiler, which acts on Java bytecode and produces the requested, platform-specific version of the applet, is remarkably similar to the description of the "local resources" described in the '248 patent. Specifically, the '248 patent explains that "[l]ocal resources 26 are comprised of program modules 32 (applets in source form, not the requested form) and compilers 30. The program modules 32 are run through the compilers 30 in order to produce applets in the requested form. … Program modules 32 are program code used to build applets. Program modules 32 may be stored in local resources 26 in source, binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers 30 upon one or more program modules 32." EX1001 at 4:46-57.

282. In sum, in my opinion, the Parthasarathy-Kimera combination teaches compiling said applet [*e.g.*, Parthasarathy's software component] into said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] from a local resource comprising at least one source module [*e.g.*, Kimera's Java bytecode] and one compiler [*e.g.*, Kimera's compilation service]

which acts on said source module to produce said selected form [*e.g.*, Kimera's disclosure of "[c]ompilation from Java bytecode into native machine code"].

d. [1c] c) transmitting said applet in said selected form to said client computer.

283. "When a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local computer 36. … If a more recent version of the desired software component is available, or if the desired computer software component is not available on the local computer 34, then it is downloaded in an asynchronous manner by the data download module 60." EX1007 (Parthasarathy) at 8:19-31. In my opinion, Parthasarathy's software component would be downloaded to Kimera's Firewall/server, compiled and cached, and then the "software components are returned 78 to the client application (e.g., the network browser) which made the request for the computer software components." *Id.* at 8:19-34, 29:13-20; *see also* EX1008 (Kimera) at 2.

284. Further, a POSA would have understood that the applet that was returned to the client computer would have been in the platform-specific form needed to run on the specific requesting client's operating system and CPU architecture.

285. In sum, in my opinion, the Parthasarathy-Kimera combination teaches transmitting [*e.g.*, Parthasarathy's returning] said applet [*e.g.*, Parthasarathy's software component] in said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] to said client computer.

286. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 1 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

3. Dependent Claims 2-4

a. 2. The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.

287. As an initial matter, copying data to a cache if it is not already in the cache, is exactly how caches operate. *See* §IV.C above.

288. Parthasarathy discloses "an illustrative embodiment of the present invention, [in which] most of the downloaded software components are installed in a permanent cache store in an operating system directory called "windows\occache." EX1007 (Parthasarathy) at 24:40-44. "Furthermore, when downloading files, the ModuleUsage registry information is used as an efficient shortcut for verifying whether a file needs to be overwritten because it is an out-of-date version." *Id.* at 27:15-25. Parthasarathy's permanent cache storage is used in conjunction with

ModuleUsage registry information (1) "as an efficient shortcut for verifying whether a file needs to be overwritten because it is an out-of-date version," and if so (2) the "ModuleUsage registry information [] allows caching of downloaded code in the permanent cache store." *Id.*

289. This is similar to Kimera's disclosure regarding "[c]aching the results of recent compilations in the central compilation server…." EX1008 (Kimera) at 2. The recent compilations were the "[c]ompilation from Java bytecode into native machine code." *Id.* The "native machine code" would be a platform-specific form of the code, and would therefore be the selected form of the applet. A POSA would have understood that Kimera's disclosure of caching the results of recent native-machine code compilations in the central compilation server is the same as "copying said applet in said selected form to a local cache after compiling said applet from said local resource." *Id.*

290. A POSA would have further understood that the compilation results would only be copied to cache memory "if said cache does not contain a copy of said applet in said selected form" because there would be no need to overwrite an applet stored in memory with the exact same applet. *See* EX1007 (Parthasarathy) at 27:15-25. Further, if the selected applet was already present in the Code Cache, it would be transmitted to the local (client) computer from there in order to "further reduce the latency of binary translation," as taught by Kimera. *Id.* Returning cached

applets would reduce the burden on the network considerably, as well as return results to users/clients much faster. *See* §IV.C above; *see also* EX1018 (Silberschatz) at 43.

291. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 1, further comprising the step of: copying said applet [*e.g.*, Parthasarathy's software component] in said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] to a local cache [*e.g.*, Parthasarathy's permanent cache storage or Kimera's Code Cache] after compiling said applet from said local resource [*e.g.*, Kimera's compilation service] if said cache does not contain a copy of said applet in said selected form.

b. *3. The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said selected form to said server computer.*

292. As an initial matter, retrieving data to be cached from a remote source, when the data is not already in the cache, is exactly how caches operate. *See* §IV.C above.

293. "When a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local

computer 36." EX1007 (Parthasarathy) at 8:20-24. "If a more recent version of the desired software component is available, or if the desired computer software component is not available on the local computer 34, then it is downloaded in an asynchronous manner by the data download module 60." *Id.* at 8:26-31. In other words, a POSA would have understood that "[i]f the desired software component is not registered and installed on the local computer, or if the version number is incorrect, then the desired software component is obtained from the remote computer 44." *Id.* at 13:45-48.

294. Further, given the arrangement of Kimera's Firewall/server in the Parthasarathy-Kimera combined system, a POSA would know that requests for software components not present on the local computer would pass through Kimera's centralized server before being passed to the broader network. As requests pass through the Firewall/server, a POSA would have understood the "central compilation server" would determine whether the requested code was stored in this Code Cache. *Id.* If the code was present in the Code Cache, it would be served to the local client computer from there. Returning recently cached results would reduce the burden on the network considerably, as well as return results to users/clients much faster. *See* §IV.C above.

295. Conversely, if the Firewall/server determined that the requested applet in a selected form was not present in the Code Cache, then the Firewall/server would

be configured to communicate with external networks and resources to locate and download the desired software components. *See* EX1007 (Parthasarathy) at 8:26-31. Programming like this on the Firewall/server is contemplated in Parthasarathy, which notes "executable programs" may "reside at the remote computer 44 or other remote computers" connected to the computer network 38. *Id.* at 7:3-26.

296. As Kimera's Firewall/server is a remote computer (in affect, a proxy device), it is my opinion that executable programs to determine software availability would have been an obvious use case for this type of proxy service. A POSA would have recognized this combined architecture to be nothing more than a traditional network proxy. *See* §IV.A above.

297. Parthasarathy teaches that "[w]hen a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local computer 36"; in my opinion, a POSA would have understood that Kimera's Code Cache would also be checked along with the memory of the local computer before "the desired software component is obtained from the remote computer 44." EX1007 (Parthasarathy) at 8:20-24, 13:45-48.

298. Specifically, Kimera's centralized server would determine if the requested applet in the selected form existed in the Code Cache or if it could be compiled with resources local to the centralized server. *Id.*; EX1008 (Kimera) at 2

("Caching the results of recent compilations in the central compilation server can be used to further reduce the latency of binary translation."). Only if the desired applet in the selected form was not among the "recent compilations" stored in the Code Cache, and could not be compiled from resources on the centralized server, would the centralized server pass the request on to the external network so that "the desired software component is obtained from the remote computer 44." EX1008 (Kimera) at 2; EX1007 (Parthasarathy) at 8:20-30, 13:45-48. Thus, in my opinion, Kimera's Code Cache renders obvious the claimed "local cache."

299. Further, a POSA would have understood that Kimera's goal of "reduc[ing] the latency of binary translation" would be accomplished by compiling a form of an applet with resources stored in the Code Cache, wherever possible. EX1008 (Kimera) at 2. For example, Parthasarathy explains "[i]t is possible to create platform-independent setup scripts that pull desired software components from different locations depending on the desired platform." EX1007 (Parthasarathy) at 17:63-66. In the example .INF file, Parthasarathy provides three examples:

> "file-win32-x86=file://products/release/circ3/x86/circ3.cab
> file-win32-mips=file://products/release/circ3/mips/circ3.cab
> file-mac-ppc=ignore"

Id. at 18:9-12.

300. In these examples, the Macintosh-PowerPC applet utilizes "the 'ignore' keyword [which] means that this file is not needed for this platform." *Id.* Thus, a POSA would have understood that if a client requests the Windows-MIPS applet, and only a Windows32-x86 form of the applet was recently compiled and stored in Kimera's Code Cache, then the code could be re-compiled to produce and cache a new form of the applet, Windows-MIPS. EX1008 (Kimera) at 2.

301. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 2, further comprising the step of: transmitting a request to an external resource [*e.g.*, Parthasarathy's remote computer 44] for said applet [*e.g.*, Parthasarathy's software component] in said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] if a copy of said applet is not stored in said local cache [*e.g.*, Parthasarathy's permanent cache storage or Kimera's Code Cache] and said applet can not be compiled from said local resource [*e.g.*, Kimera's compilation service], and directing said external resource to transmit said applet in said selected form to said server computer [*e.g.*, Kimera's Firewall/server].

c. *4. The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to transmit said applet in said selected form to said server computer.*

302. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.3.b above. EX1007 (Parthasarathy) at 7:3-26, 8:20-24, 8:26-31, 13:45-48, 17:63-66, 18:9-12; EX1008 (Kimera) at 2.

303. Specifically, claim 4 similarly recites a portion of claim 3, claiming only transmitting the request to an external resource "if said applet can not be compiled from said local resource" as opposed to claim 3's broader transmitting the request "if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource."

304. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 1, further comprising the step of: transmitting a request to an external resource [*e.g.*, Parthasarathy's remote computer 44] for supplying said applet [*e.g.*, Parthasarathy's software component] in said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] if said applet can not be compiled from said local resource [*e.g.*, Kimera's

compilation service] and directing said external resource to transmit said applet in said selected form to said server computer [*e.g.*, Kimera's Firewall/server].

4. Independent Claim 5

a. [5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:

305. I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 5 is limiting, it is my opinion that a POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.a above. EX1007 (Parthasarathy) at Abstract, 2:65-67, 4:1-2, 4:44-47, 7:3-8, 8:19-33, 9:9-21, 14:25-27, 15:52-16:4, 17:61-18:20, 18:35-57, 18:64-19:3; EX1007 (Kimera) at 1, 2.

b. [5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms;

306. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.b above. EX1007 (Parthasarathy) at 8:20-33, 15:48-16:4, 17:61-18:20, 18:35-57, 18:64-19:3; EX1008 (Kimera) at 2.

c. [5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer;

307. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.3.b above. EX1007 (Parthasarathy) at 7:3-26, 8:20-31, 13:45-48, 17:63-66, 18:9-12; EX1008 (Kimera) at 2.

308. As Parthasarathy teaches, "[w]hen a request is made to download a desired software component stored on a remote computer 44, the software component download module 58 determines if the desired software component is already installed on the local computer 36." EX1007 (Parthasarathy) at 8:20-24. "If a more recent version of the desired software component is available, or if the desired computer software component is not available on the local computer 34, then it is downloaded in an asynchronous manner by the data download module 60." *Id.* at 8:26-31. *See also* §IV.C above.

309. Given the arrangement of Kimera's Firewall/server in the Parthasarathy-Kimera combined system, a POSA would have known that requests for software components not present on the local computer 36 would pass through Kimera's centralized server before being passed to the broader network. As requests pass through the Firewall/server, a POSA would have understood that the "central compilation server" would determine whether the requested code was stored in this

Code Cache. *Id.* If the code was present in the Code Cache, it would be served to the local client computer from there. *See* EX1008 (Kimera) at 2. Returning recently cached results would reduce the burden on the network considerably, as well as return results to users/clients much faster. *See* §IV.C above.

310. In sum, in my opinion, the Parthasarathy-Kimera combination teaches determining whether said applet is stored in said specified form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] in a local cache [*e.g.*, Kimera's Code Cache] and, if so, transmitting said applet in said specified form to said client computer.

d. [5c] c) if said applet is not stored in said selected form in said local cache,

311. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.3.b above. EX1007 (Parthasarathy) at 7:3-26, 8:20-31, 13:45-48, 17:63-66, 18:9-12; EX1008 (Kimera) at 2.

312. Specifically, As Parthasarathy teaches, "[i]f the desired software component is not registered and installed on the local computer [or stored in cache memory], or if the version number is incorrect, then the desired software component is obtained from the remote computer 44." EX1007 (Parthasarathy), 13:45-48, 8:26-30.

313. In sum, in my opinion, the Parthasarathy-Kimera combination teaches determining if said applet is not stored in said selected form [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] in said local cache [*e.g.*, Kimera's Code Cache].

e. [5d] compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and

314. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.c above. EX1008 (Kimera) at 2; EX1001 at 4:46-57.

f. [5e] transmitting said applet in said selected form to said client computer.

315. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.d above. EX1007 (Parthasarathy) at 8:19-34, 29:13-20; EX1008 (Kimera) at 2.

316. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 5 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

5. Dependent Claims 6-8

a. 6. The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.

317. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.3.a above. EX1007 (Parthasarathy) at 24:40-44, 27:15-25; EX1008 (Kimera) at 2; EX1018 (Silberschatz) at 43.

b. 7. The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said form to said server computer.

318. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.3.b above. EX1007 (Parthasarathy) at 7:3-26, 8:20-31, 13:45-48, 17:63-66, 18:9-12; EX1008 (Kimera) at 2.

c. *8. The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resources, and directing said external resource to transmit said applet in said selected form to said server computer.*

319. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.3.c above. EX1007 (Parthasarathy) at 7:3-26, 8:20-31, 13:45-48, 17:63-66, 18:9-12; EX1008 (Kimera) at 2.

6. Independent Claim 9

a. *[9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:*

320. I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 9 is limiting, it is my opinion that a POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.a above. EX1007 (Parthasarathy) at Abstract, 2:65-67, 4:1-2, 4:44-47, 7:3-8, 8:19-33, 9:9-21, 14:25-27, 15:48-16:4, 17:61-18:20, 18:35-57, 18:64-19:3; EX1008 (Kimera) at 1, 2.

b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

321. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.b above. EX1007 (Parthasarathy) at 8:20-33, 15:48-16:4, 17:61-18:20, 18:35-57, 18:64-19:3; EX1008 (Kimera) at 2.

c. [9b] b) compiling an applet program module into an intermediate form program module;

322. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.c above. EX1008 (Kimera) at 2; EX1001 at 4:46-57.

323. Kimera teaches compiling Java bytecode into native machine code at the centralized Firewall/server. EX1008 (Kimera) at 2. A POSA would have understood that "an intermediate form program module" could include Java bytecode as well as other intermediate representations, because the '248 patent describes "[t]he intermediate byte-code need not necessarily be Java byte-code." EX1001 at 3:38-39.

324. Further, a POSA would have found it obvious to compile code into Java bytecode to the extent a client required Java bytecode, such as for experimental or debugging purposes or for use in a Java-compliant browser. *See* §IV.D above; *see*

also EX1014 (Microsoft Computer Dictionary), "bytecode" definition at 69. A POSA would have further understood that code can exist in multiple intermediate stages, and at each stage one may apply a variety of transformations such as debugging, optimizations, profiling, and more. *See* §IV.F above for a detailed discussion of the types of transformations that can be applied from one intermediate stage of the compiler to another.

325. In sum, in my opinion, the Parthasarathy-Kimera combination teaches compiling an applet program module [*e.g.*, Parthasarathy's software components] into an intermediate form program module [*e.g.*, Kimera's Java bytecode or any other intermediate form a POSA would consider].

d. [9c] c) transmitting said applet in said selected form to said client computer.

326. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.B.2.d above. EX1007 (Parthasarathy) at 8:19-34, 29:13-20; EX1008 (Kimera) at 2.

327. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 9 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

7. Dependent Claims 10-13

a. 10. The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.

328. As an initial matter, a POSA would have understood that code can exist in multiple intermediate stages, and at each stage one may apply a variety of transformations such as debugging, optimizations, profiling, verification, and more—to transform the code from one intermediate form to another. When considering a proxy device that serves a variety of users and clients of different operating systems and architectures, it is imperative for such a device to be as versatile as possible, so it can serve every diverse need. *See* §IV.F above for a detailed discussion of the types of transformations that can be applied from one intermediate stage of the compiler to another.

329. The Parthasarathy-Kimera system includes various transformer programs that are used to transform code program modules for the client computer. For example, Parthasarathy discloses a "code verification module 62 [that] checks a digital signature in the downloaded software component to ensure the downloaded software component is sage (e.g., computer virus and corruption free)…." EX1007 (Parthasarathy) at 8:35-44, 3:9-21.

330. Kimera similarly discloses a "verification service [that] performs checking of incoming code for typesafety and conformance to interfaces." EX1008 (Kimera) at 2. Kimera's verification service applies "[t]amper-proof signatures, using a cryptographically strong hash function such as MD5…." *Id.* This verification service would be performed at the centralized Firewall/server to "enable[] the same services to be performed faster, cheaper, and without introducing security vulnerabilities." EX1008 (Kimera) at 1.

331. In my opinion, a POSA would have understood that software components that had been checked by the verification service and to which an MD5 hash function had been applied would constitute a transformed intermediate program module, ready to be returned to the client device.

332. Parthasarathy also describes transforming files through compression. For example, Parthasarathy teaches a "cabinet file contains one or more files, all of which are downloaded together in a compressed 'cabinet.'" EX1007 (Parthasarathy) at 15:65-16:1. File compression in software retrieval was well known at the time of the '248 patent, and it would have been obvious to a POSA to utilize compression for software components retrieved from computer networks and sent to client computers. *See* §IV.E above; *see also* EX1014 (Microsoft Computer Dictionary), "compress" definition at 107 (definition 2).

333. In my opinion, it would have been obvious to transform software components via compression to further satisfy the goals and limitations described in Kimera: namely, reducing client system requirements and avoiding excessive communication between server computers and client computers. EX1008 (Kimera) at 1, 2. Compressing applets sent to the client computer enables the system to "make[] a sizeable reduction in computation and memory requirements of the end-points" because the received, compressed applets can be more quickly received and processed on the client computers.

334. "Mak[ing] a sizeable reduction in computation and memory requirements of end-points," as taught by Kimera, also suggests transforming applets via optimization. EX1008 (Kimera) at 2. Specifically, optimization techniques were well known as of the time of the '248 patent, and a POSA would have been motivated to incorporate any known optimization techniques to allow for the maximum reduction is system requirements, as Kimera describes. *See* §IV.D above; *see also* EX1014 (Microsoft Computer Dictionary), "optimization" and "optimizing compiler" definitions at 343; *see also* EX1020 (GNU C Manual Page) at 16-17.

335. Finally, Kimera teaches that "[c]ompilation from Java bytecode into native machine code is another service that is best performed at a central location." EX1008 (Kimera) at 2. In my opinion, a POSA would have understood that a compiler is a transformer program that acts on Java bytecode to produce a

transformed intermediate program module, such as native machine code. *Id.*; *see also* §§IV.DIV.F above.

336. In sum, in my opinion, Parthasarathy-Kimera teaches the method of claim 9, further comprising the step of: transforming [*e.g.*, via Parthasarathy's code verification module, Kimera's verification service, Parthasarathy's compression scheme, Kimera's optimization, or Kimera's compilation services] said intermediate form program module [*e.g.*, Kimera's Java bytecode or any other intermediate data] into a transformed intermediate form program module [*e.g.*, Kimera's MD5-signed applet, Parthasarathy's compressed files, Kimera's native machine code, or another intermediate stage] with at least one transformer program [*e.g.*, Parthasarathy's code verification module, Kimera's verification service, or Kimera's compiler].

b. 11. The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring computer programs and encrypting computer programs.

337. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.7.a above. EX1007 (Parthasarathy) at 3:9-21, 8:35-44, 15:65-16:1; EX1008 (Kimera) at 1, 2; *see also* EX1014 (Microsoft Computer Dictionary), "compress" definition at 107 (definition 2), 343; *see also* EX1020 (GNU C Manual Page) at 16-17. Moreover, a

POSA would have known that tools to compile, verify, optimize, compress, debug, and profile for usage monitoring were commonplace and standard on many operating systems, as detailed in §IV.F above.

338. Nevertheless, I note that this claim can be met by just one of the transformer programs because the claim only requires transformer program to be at least one of the programs selected from the following list of computer programs: a verifying computer program, an optimizing computer program, a compressing computer program, a debugging computer program, a usage monitoring computer program, or an encrypting computer program. Regardless, Parthasarathy-Kimera renders obvious nearly all of the listed transformer programs.

339. For example, both Parthasarathy and Kimera teach verifier computer programs, namely, Parthasarathy's code verification module 62 and Kimera's verification service. EX1007 (Parthasarathy) at 8:35-44 ("Any software components downloaded to the local computer 36 are verified with the code verification module 62. The code verification module 62 checks a digital signature in a digital certificate included in the downloaded software component to ensure the downloaded software component is safe (e.g., computer virus and corruption free) on the local computer 34."), 3:9-21; EX1008 (Kimera) at 2 ("The verification service performs checking of incoming code for typesafety and conformance to interfaces.").

340. Kimera also suggests taking advantage of optimizing programs to take advantage of Kimera's client with reduced computation and memory requirements. EX1008 (Kimera) at 1 ("A centralized security architecture is easy to administer, provides enterprise-wide security, supports safe auditing, and **reduces Java client resource requirements**. … In particular, locating verification, access control, auditing, and compilation services in centralized servers enables the same services to be performed faster, **cheaper**, and without introducing security vulnerabilities.").

341. Parthasarathy describes taking advantage of compression protocols, such as Lempel-Ziv compression. EX1007 (Parthasarathy) at 16:1-16:9 ("Using a cabinet file requires authoring of a .INF and packaging of a CAB file, but in return it provides file compression. File compression is completed with Lempel–Ziv compression, which is known in the art. However, other compression formats could also be used.").

342. Finally, Kimera describes the use of encryption-related techniques, including the application of cryptographically strong hash functions such as MD5. EX1008 (Kimera) at 2 ("Locating the verification service at a separate server requires that the code verified at the central service remain intact after verification. Tamper-proof signatures, using a cryptographically strong hash function such as MD5, can be utilized to ensure the safety of the code after it leaves the verifier.").

343. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs [*e.g.*, Parthasarathy's code verification module or Kimera's verification service], optimizing computer programs [*e.g.*, known optimization programs to enable Kimera's reduction in client computer computation requirements], compressing computer programs [*e.g.*, Parthasarathy's Lempel-Ziv compression], debugging computer programs, usage monitoring computer programs and encrypting computer programs [*e.g.*, Kimera's cryptographic hashing].

> *c. 12. The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.*

344. Kimera discloses "[c]compilation from Java bytecode into native machine code is another service that is best performed at a central location." EX1008 (Kimera) at 2. Thus, a POSA would have understood that Kimera teaches using the Firewall/server's compiler to compile the transformed intermediate form program module (*e.g.*, Java bytecode) into machine specific binary code (*e.g.*, native machine code). This is similar to Parthasarathy's use of "[t]he %opersys% and %cpu% attribute values … [to] ***specify the operating system and CPU*** for the desired platform desired software components will be executed on." EX1007

(Parthasarathy) at 18:35-43; *see also* EX1007 (Parthasarathy) at 17:61-18:20, 18:45-57, 18:64-19:3. Thus, a POSA would recognize the ultimate result of the compilation is a machine-specific, or platform-dependent, software executable or applet.

345. Kimera's disclosure matches that of the '248 patent, which states "[a] target compiler 52 compiles intermediate applet code 44 into an applet 54 in a processor specific format (binary) suitable for running natively on a given computing platform." EX1001 at 5:11-26. The "intermediate form program module" may be Java byte code, according to the '248 patent. *Id.* at 3:38-39 ("The intermediate byte-code need not necessarily be Java byte-code.").

346. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 11, further comprising the step of: compiling [*e.g.*, via Kimera's compiler] said transformed intermediate form program module [*e.g.*, Kimera's Java bytecode, as transformed by a verifier, optimizer, compression, or encryption transformer] into machine specific binary code [*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] with a target compiler [*e.g.*, Kimera's compiler].

d. 13. The method of claim 10, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

347. A POSA would find that the Parthasarathy-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.B.7.c above. EX1007 (Parthasarathy) at 17:61-18:20, 18:35-43, 18:45-57, 18:64-19:3; EX1008 (Kimera) at 2; EX1001 at 3:38-39, 5:11-26.

C. Ground 3: Claims 1-13 are Unpatentable Under 35 U.S.C. §103(a) as Obvious over Fowlow in view of Kimera

348. Independent claims 1, 5, 9, and dependent claims 2-4, 6-8, and 10-13 are obvious over Fowlow combined with Kimera. Below I explain how each element of claims 1-13 are disclosed, taught, and/or suggested by Fowlow in combination with Kimera.

1. Motivation to Combine

349. In my opinion, there is strong motivation to combine Fowlow and Kimera, including at least five discrete reasons, and incorporating the functionality of Kimera's Firewall/server into Fowlow's ORB would have been relatively easy.

350. First, both Fowlow and Kimera are concerned with downloading software such as Java applets. Fowlow teaches "[a] client enabled to load and run Java applets in a distributed object computing system retrieves needed Java classes in a location-independent manner from various class servers in the system." EX1009

(Fowlow) at Abstract. Kimera similarly describes how "Java enables mobile programs, called applets, to be shipped from a web server to a client's machine, where they execute locally and are integrated into the client's environment." EX1008 (Kimera) at 1. In my opinion, a POSA would be motivated to improve one Java software system (Fowlow) with the functionality of another Java software system (Kimera).

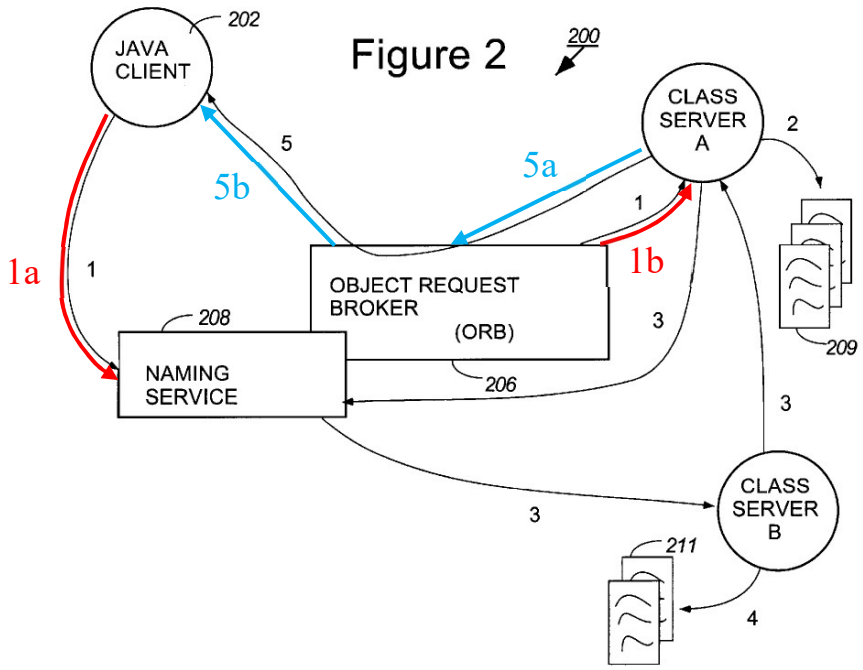
351. Second, both Fowlow and Kimera suggest and explain the benefits of storing software in more locations than just a single server. Fowlow describes "[o]ne advantage to using an ORB is that these class files and their associated class server may be located anywhere within the distributed object system yet may still be found by the client." EX1009 (Fowlow) at 10:23-26. Thus, Fowlow suggests that Java applet code can be stored anywhere in the distributed object system. In my opinion, a POSA would be motivated to include the ORB itself in the list of places on the distributed object system that can store Java class files because the ORB would already be storing code temporarily as the code was transmitted from server to client.

352. A POSA would have understood that storing code in multiple nodes has the benefit that the code can be retrieved by clients more efficiently (*e.g.*, download from the closest node) as well as more reliably (*e.g.*, if some but not all nodes are down). *See also* §IV.A above; *see also* EX1014 (Microsoft Computer Dictionary) at 276 (defining a "LAN" as "A group of computers and other devices dispersed

over a relatively limited area and connected by a communications link that enables any device to interact with any other on the network.").

353. Similarly, storing code at the centralized ORB mirrors the teaching of the Code Cache of Kimera's Firewall/server, such that a POSA would have been motivated to use Kimera's teachings to improve the ORB in a way to store code segments. EX1008 (Kimera) at 1 (Kimera Fig. 2), 2. The Kimera firewall caches code on the local network, which provides similar benefits to the distributed ORB system of Fowlow: code can be retrieved by local clients more efficiently (*e.g.*, download from the local Kimera firewall) as well as more reliably (*e.g.*, download cached code from Kimera's firewall even if the source servers are down). *See also* §IV.C above.

354. Third, in my opinion a POSA would be motivated to combine Fowlow and Kimera to simplify the dependency resolution process. For example, in Fowlow, the unresolved dependency resolution process is somewhat long and convoluted, which can be seen in the annotated version of Figure 2 shown below:

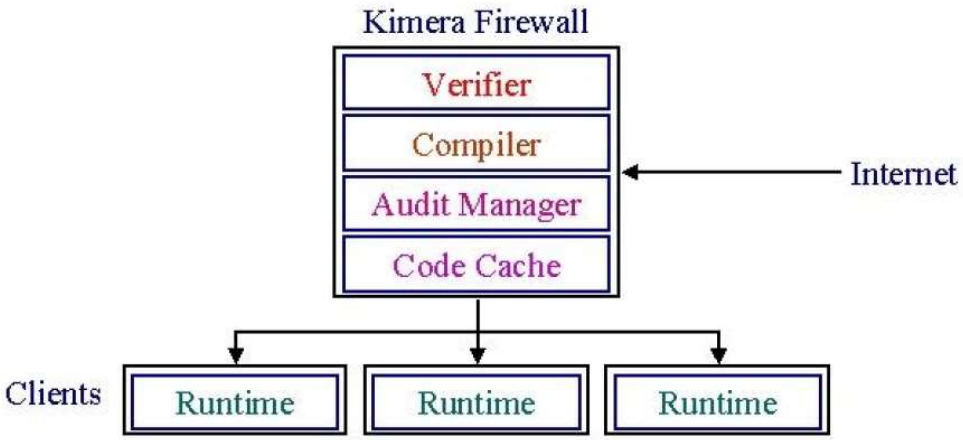


EX1009 (Fowlow) at Fig. 2 (annotated).

355. A Fowlow client first requests a base applet by querying the ORB (arrow 1a). *Id.* at Abstract, 3:31-42, 3:58-4:15, 13:18-67. The ORB then performs whatever pre-processing is required to communicate with the class servers. *See Id.* at 6:5-8 (describing performing "a variety of tasks, such as compression, encryption, tracing, or debugging, that are to be applied to communications to and from an object"), 8:27-31 (describing converting "the object reference of the request [] into protocols appropriate to a given domain."). The ORB then passes the request to the servers to locate the base class files (arrow 1b). *Id.* at Abstract, 3:31-42, 3:58-4:15, 13:18-67. Once code is located on the class servers, it is downloaded to the ORB (arrow 5a). *Id.* The base class files are then processed (*e.g.*, compressed, optimized,

and verified) at the ORB before being sent to the client (arrow 5b). *Id.*; *see also Id.* at 6:5-8. If the base class that is returned to the client is found to have any missing dependencies, then the *entire loop* must be completed again to locate and download those files. *Id.* at Abstract, 4:3-15, 13:18-67.

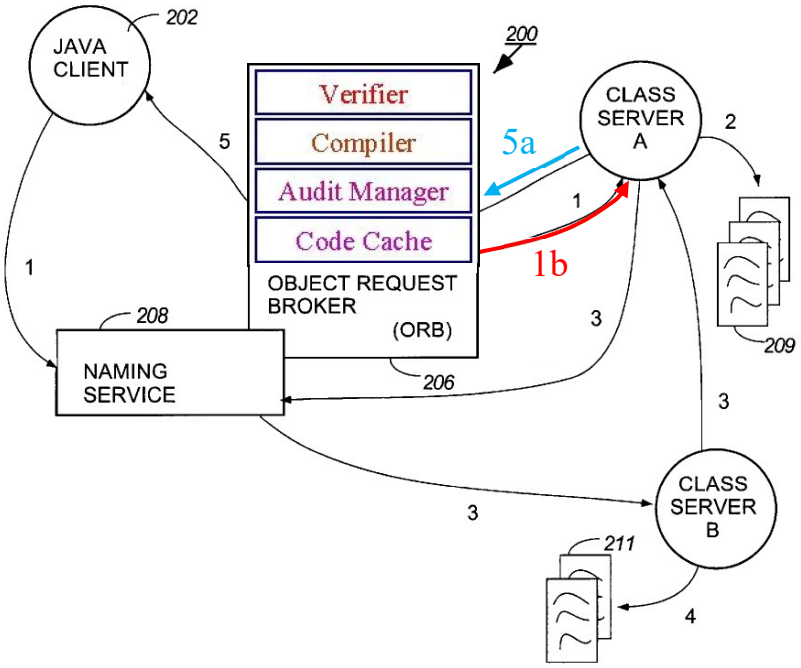
356. Kimera highlights the problem with Fowlow's dependency resolution process, noting that the full dependency resolution loop would be accompanied by "communication overhead between the client and the compilation server [that] would dominate any potential benefits." EX1008 (Kimera) at 1, 2. Moving the processing functionality to the centralized Firewall/server "could deliver high performance, as it is financially viable to locate the shared firewall service on a machine faster than any endpoint." *Id.* at 2. This includes, for example, moving verification, compilation, and code caching to the Firewall/server. *Id.* ("The same arguments for ease of upgrades and uniformity of enforcement that apply to the verification service apply equally well to the compilation service."). That way, as code arrives from servers across the Internet, it can be verified, compiled, and cached at the centralized Firewall/server. *Id.* See Kimera Figure 2 annotated below:



Kimera Firewall Architecture

Id. at 1 (Kimera Figure 2).

357. In my opinion, moving compilation and verification services to the centralized Firewall/server, as opposed to client machines, would significantly reduce the communication overhead between client and server, especially in resolving unresolved dependencies. In the modified system, which would include adding Kimera's Firewall/server functionality to Fowlow's ORB, the verifier would check the downloaded base class for any missing dependencies, recognize missing dependencies, and be able to go back to the servers to locate and download the dependent code. This effectively relieves the client from the dependency resolution process, which would consist of only arrows 1b and 5a as shown below:



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

358. The benefit of this modified arrangement is apparent: communication overhead between client and server is drastically reduced, resulting in faster applet load times. By moving the verifier and compilation functionality from client devices to the Kimera firewall, and further caching information on the Kimera firewall itself, it is my opinion that these features could be removed from client devices (because Kimera says this is possible), which would make client terminals cheaper as well. EX1008 (Kimera) at 1-2. Finally, as Kimera notes, moving this functionality to a centralized location allows for easier and faster security updates, making the system

more secure as well as more efficient (*e.g.*, thanks to caching and saving network messages) and less expensive. *Id.* at 2.

359. Fourth, Fowlow recognizes another motivation to move resolution dependency functionality to the ORB: that the client "knows a base class that it wishes to load, but is mercifully unaware of all the machinations behind the scenes required to find other classes used or needed by this base class." EX1009 (Fowlow) at 10:26-34. Moving the dependency resolution process to the ORB would further make the client mercifully unaware of the "machinations behind the scenes required to find other classes used or needed by [the] base class." *Id.* at 10:33-34

360. Fifth, in my opinion a POSA would have been motivated to combine Fowlow and Kimera in order to speed up a client's access to requested applets by taking advantage of Kimera's centralized "Code Cache." This centralized Code Cache caches the results of recent compilations "to further reduce the latency of binary translation" in response to future requests for the same code. EX1008 (Kimera) at 2. In Fowlow's current architecture, if N clients ask for the same Java program, the ORB has to retrieve that code N times. This is exactly the "waste" that Kimera tried to address. EX1008 (Kimera) at 1.

361. In my opinion, storing compiled code in a Code Cache on Fowlow's ORB modified by Kimera's Firewall/server would make the system even more efficient and quick, as a base class *and the resolved dependencies* would be stored

on the cache and accessed from cache on second requests or even requests from other clients in the network. In other words, while the ORB as modified by Kimera's Firewall/server shortens the dependency resolution loop, the Code Cache eliminates it altogether for secondary requests for the same applet, including requests from other clients connected to the ORB. *See also* §IV.C above.

362. Fowlow's ORB, modified to incorporate the functionality of Kimera's Firewall/server, would continue to serve as the single point of contact for clients sending requests for applets, while bolstering the server-side processing functionality, namely the compilation, verification, and dependency resolution processes. *See* EX1009 (Fowlow) at 5:49-51, 10:18-31, 11:25-33. Because the modifications to the ORB would be minor, and primarily involve the teachings of Fowlow and of Kimera functioning the same way as the original references, it is my opinion that this combination could be made with a high expectation of success. A POSA would know, for example, that inserting a proxy like device such as Kimera's firewall can be accomplished transparently to other clients and servers. It is my opinion that there is strong motivation to combine these references, and that the combination would be accomplished without undue burden.

2. Independent Claim 1

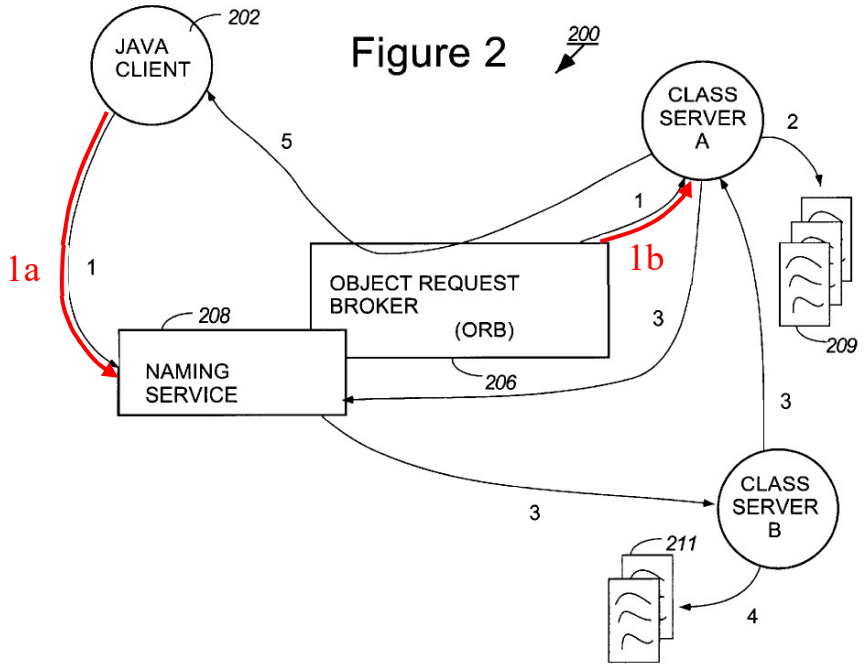
a. [1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

363. I have been asked to provide an analysis of this claim element, which I understand is the preamble. However, I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 1 is limiting, it is my opinion that it is disclosed by Fowlow in view of Kimera.

364. Fowlow teaches a distributed object system where a "client [] requests the code for the base class from the class server by using the object request broker. The class server retrieves the code by either reading a file from its own local file set, or if the code is not local, queries the naming service for another class server that has access to the code for the base class." EX1009 (Fowlow) at Abstract, 3:27-42. "The client determines whether the returned code contains any unresolved classes, i.e., classes that are used but not yet defined or loaded. The client requests code for any unresolved class in a manner as above for the base class." *Id.* at Abstract.

365. The "object request broker" or "ORB of a distributed object system is used to find the class files needed in a location independent manner." *Id.* at 10:15-23. The ORB further "allows a client a single point of access to the distributed object system in order to find the class filed that it needs." *Id.* at 10:26-28. In other words,

every request for an applet is directed to the ORB (see arrow 1a below), which in turn manages all requests to the connected servers (arrow 1b below). *Id.* at 10:18-34, Fig. 2:



Id. at Fig. 2 (annotated).

366. The object request broker is also a server computer. *Id.* at 5:34-36 ("In this example, ORB object implementation 14 is presented as a server as shown by its location in the diagram."), 10:21-23 (noting the ORB is located at the distributed object system).

367. Each request in Fowlow also specifies the form of the desired applet out of a plurality of forms. For example, Fowlow explains "[a]s used herein, the term 'applet' refers to a body of executable program code, which code is effective

to implement a function or facility when used in conjunction with a supporting execution environment. An 'applet' may refer to a wide variety of types of execution code. By way of example, an applet may be a collection of byte codes representing classes that are executed by a remote interpreter, such as Java classes that are executed using a Java language interpreter." *Id.* at 10:45-53. In one embodiment, "a Java client 202 may communicate via a communication mechanism 206 in order to acquire Java class files 209 and 2011 within a distributed object system 200." *Id.* at 11:21-25. A POSA would recognize that in this example, a Java client would be requesting an applet in a specific form, Java classes, out of a plurality of forms. Indeed, any request would need to specify which of the "wide variety of types of execution code" is required on the client computer. *Id.* at 10:49-50.

368. This disclosure is similar to that of Kimera, which clarifies that for certain applets, it is beneficial to compile Java bytecode into native machine code at a centralized server. EX1008 (Kimera) at 2 ("Compilation from Java bytecode into native machine code is another service that is best performed at a central location.").

369. The language of the native machine code could be understood from the request itself, such that the specific form of the applet could be compiled at the centralized server. *Id.*; EX1009 (Fowlow) at 7:11-52 ("the form of the request will depend to a large extent upon the nature of the programming language used to create the client."), 10:28-34. This would be especially important because code capable of

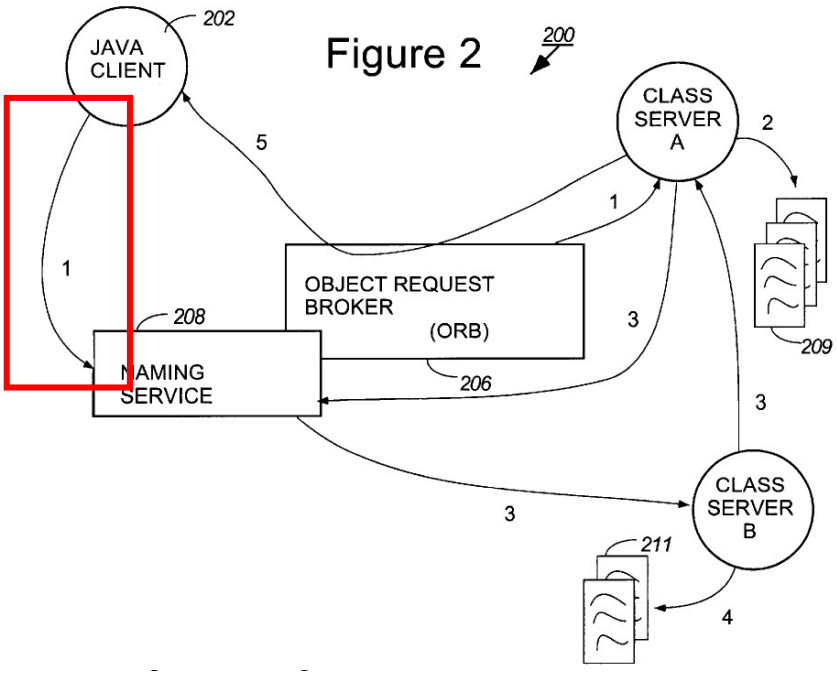
running on one client computer with one architecture could differ from code capable of running on another client computer with a different architecture. *See* EX1001 at 3:27-31 ("Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform."); *see also* EX1008 (Kimera) at 2 (noting different binary translations are compiled by the centralized compiler and the results are cached).

370. In sum, in my opinion, to the extent the preamble is limiting, the Fowlow-Kimera combination teaches a method operating on a computer system [*e.g.*, Fowlow's client-server environment] for managing requests [*e.g.*, Fowlow's requests from a Java client] to a server computer [*e.g.*, Fowlow's ORB or Kimera's centralized server] for applets [*e.g.*, Fowlow's code or "applets"] in a client server environment wherein each request for an applet specifies one form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] of the applet out of a plurality [of] forms of the applet [*e.g.*, Fowlow's "wide variety of types of execution code"].

b. [1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

371. Fowlow teaches that a client requests an applet when it "queries the object request broker to determine if there is an applet server available within the system that may be used to obtain particular applet execution code." EX1009

(Fowlow) at 3:27-42. The ORB "communicate[s] with one or more class servers that provide access to the class files needed. Thus, an ORB of a distributed object system is used to find the class files needed by in a location independent manner." *Id.* at 10:18-34, Abstract, 4:3-15, Fig. 2. For every request, a POSA would understand the ORB as receiving a request from the client for an applet.



Id. at Fig. 2 (annotated).

372. Further, as explained in §VIII.C.2.a above, the request to the ORB specifies the form of the desired applet. In one embodiment, the form of the desired applet is specifically Java classes for a Java client. *Id.* at 11:21-25. Although Java class files are one example of the plurality of forms, "a wide variety of types of

execution code" may be requested, including native machine code as taught by Kimera. *Id.* at 10:45-54; EX1008 (Kimera) at 2.

373. In sum, in my opinion, the Fowlow-Kimera combination teaches receiving on said server computer [*e.g.*, Fowlow's ORB or Kimera's centralized server] a request from a client computer for an applet in a form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] selected from a plurality [of] forms [*e.g.*, Fowlow's "wide variety of types of execution code"].

c. [1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and

374. As an initial matter, in my opinion, the term "source module" should not be limited merely to human written text-based source code, but to any original or intermediate forms that can pass through a compiler tool-suite to produce another form (and possibly a final binary executable). As I explained in §IV.D above, original source code is transformed using multiple stages by a compiler "wrapper," which in turn invokes sub-programs to take an input from one stage, process it, and produce an output suitable for the next stage of the compiler suite. Each of those intermediate stages can meet the "source module" that a "compiler … acts on."

375. Moreover, a POSA would have understood that for a compiler to successfully work, it necessarily must have the right local resources such as memory,

CPU, an installed operating system environment, libraries, and an assortment of auxiliary tools and sub-programs. *See* §IV.D above.

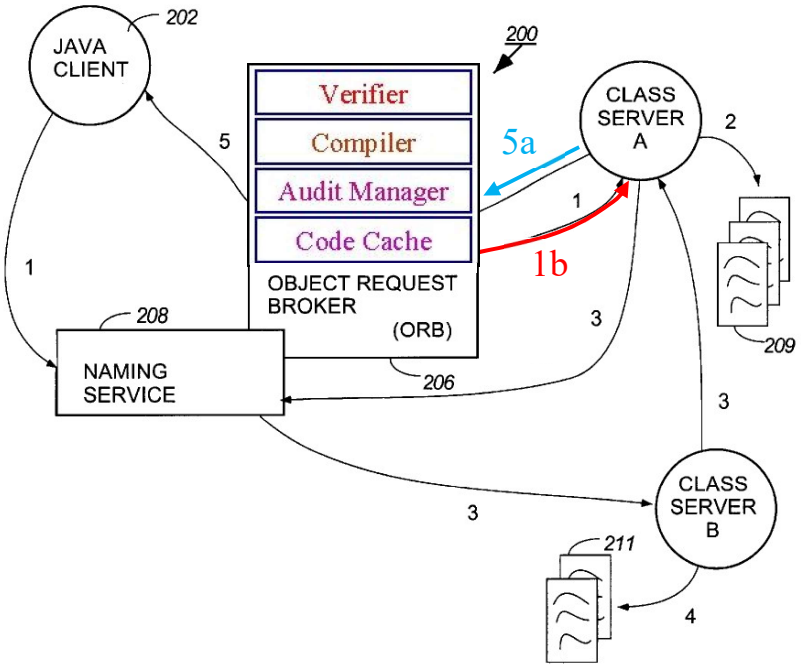
376. Fowlow teaches that "[a]n 'applet' may be a collection of byte codes representing classes that are executed by a remote interpreter, such as Java classes … may also be compiled before execution or may even undergo a combination of interpretation/compilation before execution." EX1009 (Fowlow) at 10:50-56. A POSA would have understood that the collection of classes could be a collection of discrete files, such as Fowlow's class files 209 and 211, that are combined in a single applet.

377. When a Java client requests Java classes, a POSA would have understood that the Java classes returned to the Java client would need to be compiled before the Java client could execute those classes. *Id.* at 11:21-25, 10:56-65. For Java clients, "applet execution code may refer to code that represents an implementation of a Java class." *Id.* A POSA would have understood that this applet execution code may be compiled and sent to the Java client directly. *Id.* at 11:21-25, 10:50-56 ("By way of example, an applet may be a collection of byte codes representing classes that are executed by a remote interpreter, such as Java classes that are executed using a Java language interpreter.").

378. Kimera also describes compiling applets before sending them to client computers. For example, Kimera discloses a "way of structuring Java systems, in

which verification and compilation are performed in centralized servers." EX1008 (Kimera) at 1, 2. Kimera notes that "[c]ompilation from Java bytecode into native machine code is another service that is best performed at a central location," among other benefits. *Id.* at 2. To compile into *native* machine code, Kimera's server would need to know the architecture of the requesting machine in order to compile machine code capable of natively executing on that machine. *Id.*

379. A POSA would have recognized that incorporating Kimera's Firewall/server's functionality into Fowlow's ORB would combine the benefits of the two references, with little to no downside. I have addressed the motivation to combine these references in §VIII.C.1 above. In particular, because Fowlow's ORB manages all requests and responses between clients and servers, including Kimera's verification service, compiler, and Code Cache in the ORB would greatly expand the functionality of the ORB and result in a faster, more secure system.



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

380. In this combined system, Kimera acts on a source module (*e.g.*, Java bytecode 209, Java class 211 from Fowlow's class servers) to produce the selected form (*e.g.*, native machine code). EX1008 (Kimera) at 2; EX1009 (Fowlow) at 11:21-25.

381. The applicability of Kimera's Firewall to Fowlow's ORB is even more evident when the combination is viewed next to the '248 Patent. For example, the language and figures describing Kimera's Firewall/server and the '248 Patent's applet server computer 10 are fairly similar, as seen in the comparison table and side-by-side annotated figures below:

Kimera	**'248 Patent**
"[L]ocating verification, access control, auditing, and compilation services in centralized servers enables the same services to be performed faster, cheaper, and without introducing security vulnerabilities" and allows client devices to "be constructed from cheaper components." EX1008 (Kimera) at 1.	"Having the applets delivered from applet server computer 10 allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers 12 and 14. This reduces the amount of work the client computers 12 and 14 have to do and provides a convenient way to impose global restrictions on code." EX1001 at 3:49-54, 4:50-52.

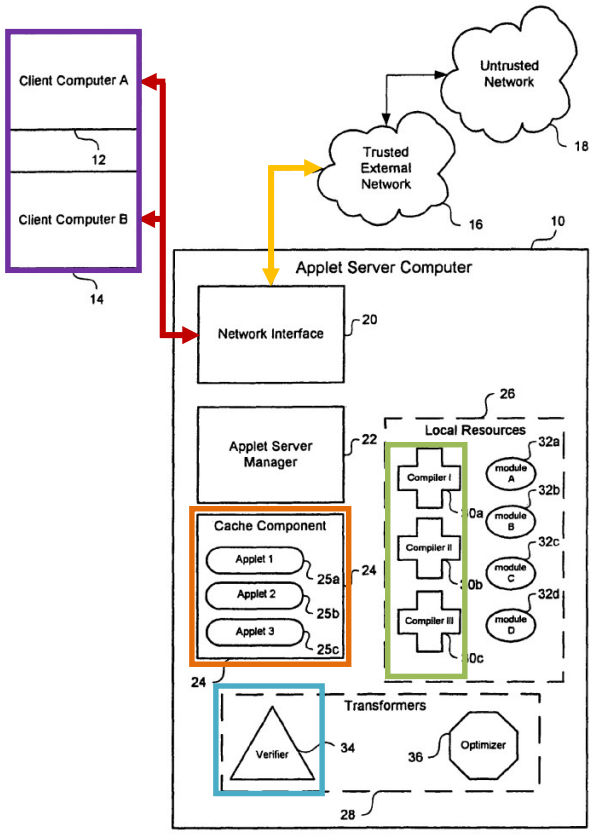
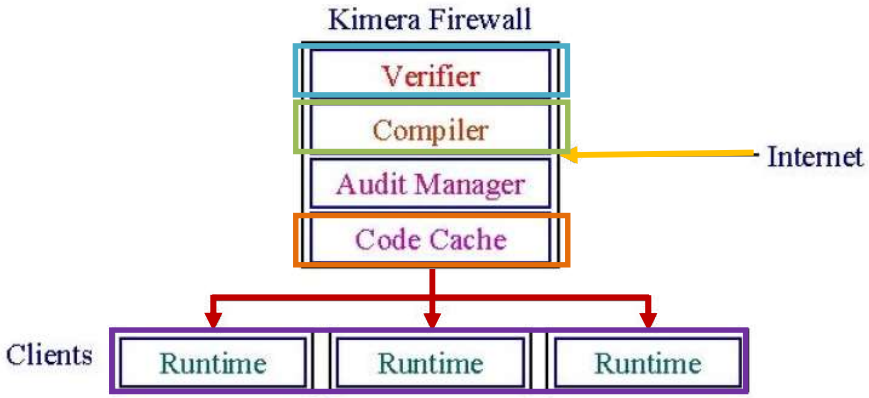


Fig. 1



EX1001 at Fig. 1 (annotated); EX1008 (Kimera) at 1 (Kimera Figure 2, annotated).

The figures have been annotated with similar, corresponding functionality using the same color in each figure, as detailed in the comparison table below:

Color	Kimera	'248 Patent
Red	Connection between Firewall and Clients	Connection between applet server computer 10 and client computers 12, 14
Orange	Code cache	Cache component 25a-c
Yellow	Connection to Internet	Connection to external networks 16, 18
Green	Compiler	Compilers 20a-20c
Blue	Verifier	Verifier 34
Purple	Clients	Client Computers

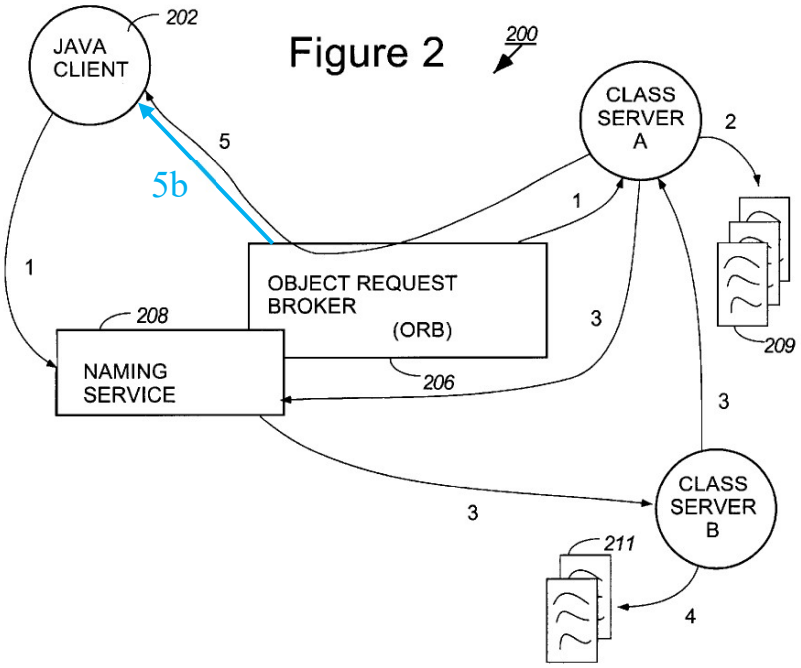
382. The similarities between Kimera and the '248 Patent continue in regard to the particular services performed by the centralized server. Kimera teaches that moving functionality to a centralized server allows for the elimination of "a JIT compiler, verifier, or interpreter" on the client end-point because each of these tasks would instead be performed at the centralized server instead of on the client. EX1008 (Kimera) at 2. This mirrors the compilers 30a-c, verifier 48, and optimizer 46 located in the applet server 10 of the '248 Patent. EX1001 at Abstract, 3:63-64, 5:1-10, Fig. 1.

383. In sum, in my opinion, the Fowlow-Kimera combination teaches compiling said applet [*e.g.*, Fowlow's code or "applets"] into said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine

code] from a local resource comprising at least one source module [*e.g.*, Fowlow's applet class files or Kimera's Java bytecode] and one compiler [*e.g.*, Kimera's compilation service] which acts on said source module to produce said selected form [*e.g.*, Kimera's disclosure of "[c]ompilation from Java bytecode into native machine code"].

d. [1c] c) transmitting said applet in said selected form to said client computer.

384. After locating the requested applet execution class files, Fowlow discloses the "class code corresponding to the class name desired by the Java client is finally returned via the ORB." EX1009 (Fowlow) at 12:6-9, Fig. 2. This can be seen by arrow 5b in the annotated Figure 2 below:



Id. at Fig. 2 (annotated).

385. "Once th[e] execution code for class name has been retrieved, it is delivered to the network class loader within the Java client in step 410. Next, in step 412, the NCL passes this execution code to the Java interpreter." *Id.* at 13:43-46.

386. A POSA would have understood that the incorporation of Kimera's Firewall/server into Fowlow's ORB would not affect the process of transmitting code to client devices. Instead, the Firewall/server would perform any compilation, verification, and/or caching services required, and then transmit the selected form of the requested application to the client device. EX1008 (Kimera) at 1 (Kimera figure 2), 2.

387. Indeed, Kimera teaches, eliminating the "JIT compiler, verifier, [and] interpreter" from the client device and moving those processes to the modified ORB allows the "Java end-point [to] consist of only a minimal runtime environment." *Id.* at 2. In order to achieve a "minimal runtime environment," the selected form of the application would need to be prepared before being transmitted to the client computer, so that the client computer could simply display or enable the applet without any additional computational load. *Id.*

388. In sum, in my opinion, the Fowlow-Kimera combination teaches transmitting [*e.g.*, Fowlow's returning, arrow 5b] said applet [*e.g.*, Fowlow's code or "applets"] in said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] to said client computer.

389. In my opinion, the Fowlow-Kimera combination discloses, or at least teaches, each limitation of claim 1 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow and Kimera.

3. Dependent Claims 2-4

a. *2. The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.*

390. As an initial matter, copying data to a cache if it is not already in the cache, is exactly how caches operate. *See* §IV.C above.

391. As discussed in §VIII.C.2.c above, Kimera's Firewall/server compiles Java bytecode to produce native machine code. EX1008 (Kimera) at 2. In addition to compiling native machine code, Kimera teaches "[c]aching the results of recent compilations in the central compilation server…." *Id.* at 2. A POSA would have understood that Kimera's disclosure of caching the results of recent native machine code compilations in the central compilation server is the same as "copying said applet in said selected form to a local cache after compiling said applet from said local resource." *Id.*

392. A POSA would have further understood that the compilation results would only be copied to cache memory "if said cache does not contain a copy of said applet in said selected form" because there would be no need to overwrite an

applet stored in memory with the exact same applet. Indeed, if the selected form of the applet was already stored in memory, a POSA would have recognized that the Firewall/server would simply transmit that applet from cache memory as opposed to re-compiling the applet.

393. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 1, further comprising the step of: copying said applet [*e.g.*, Fowlow's code or "applets"] in said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] to a local cache [*e.g.*, Kimera's Code Cache] after compiling said applet from said local resource [*e.g.*, Kimera's compilation service] if said cache does not contain a copy of said applet in said selected form.

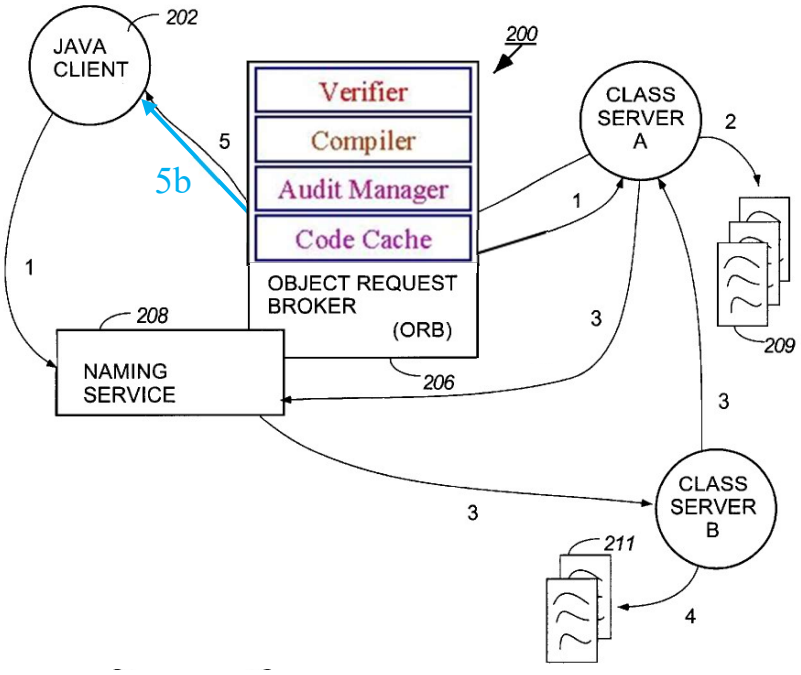
b. 3. The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said selected form to said server computer.

394. As an initial matter, retrieving data to be cached from a remote source, when the data is not already in the cache, is exactly how caches operate. *See* §IV.C above.

395. Fowlow's ORB "allows a client a single point of access to the distributed object system in order to find the class files that it needs." EX1009

(Fowlow) at 10:21-27. Incorporating the functionality of Kimera's Firewall/server into Fowlow's ORB, which a POSA would have been motivated to do as described in §VIII.C.1 above, would result in all requests passing through the modified ORB. A POSA would have recognized several benefits associated with this arrangement, including determining whether the requested code was stored in the Firewall/server's Code Cache before passing the request on to the broader distributed object system. EX1008 (Kimera) at 2.

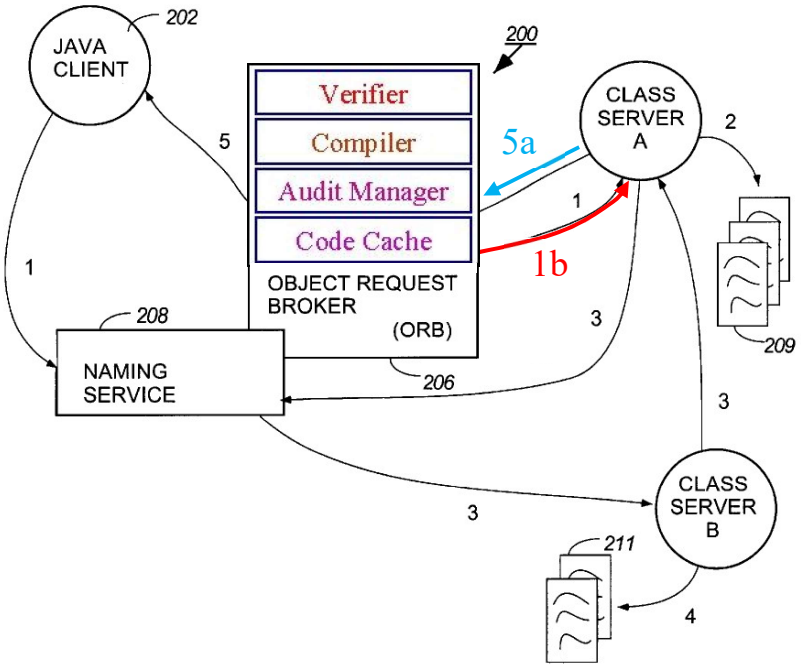
396. If the requested applet execution code was stored in the selected form in the Code Cache, the Firewall/server would simply return this code from cache memory. *Id.* This can be seen as Figure 5b in the annotated figure below:



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

397. A POSA would have readily recognized that this would result in a substantial reduction in the applet retrieval process time by reducing the burden on the network considerably. *See* §IV.C above.

398. On the other hand, if the Firewall/server determined that the requested applet in a selected form was not present in the Code Cache, then the Firewall/server would be configured to communicate with external resources (such as Fowlow's class servers) to locate and download the desired applet classes, as seen by arrows 1b and 5a in the annotated figure below. *See* EX1009 (Fowlow) at 10:15-34 ("In one embodiment, this mechanism uses an object request broker (ORB) in order to communicate with one or more class servers that provide access to the class files needed.").



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

399. As Kimera's Firewall/server is a remote computer (in affect, a proxy device), it is my opinion that executable programs to determine applet code availability would have been an obvious use case for this type of program. A POSA would have recognized this combined architecture to be nothing more than a traditional network proxy. *See* §IV.A above.

400. Further, a POSA would have understood that Kimera's goal of "reduc[ing] the latency of binary translation" would be accomplished by compiling a form of an applet with resources stored in the Code Cache, wherever possible. EX1008 (Kimera) at 2. Thus, if some of the requested applet execution code was

stored in the Code Cache, yet unresolved dependencies existed, Fowlow-Kimera would simply "request[] additional applet execution code corresponding to these unresolved references from the first applet server found." EX1009 (Fowlow) at 4:6-15. This, too, would reduce network latency by minimizing the number of classes for which code is requested from the broader (slower) network.

401. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 2, further comprising the step of: transmitting a request to an external resource [*e.g.*, Fowlow's class servers] for said applet [*e.g.*, Fowlow's code or "applets"] in said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] if a copy of said applet is not stored in said local cache [*e.g.*, Kimera's Code Cache] and said applet can not be compiled from said local resource [*e.g.*, Kimera's compilation service], and directing said external resource to transmit said applet in said selected form to said server computer [*e.g.*, Fowlow's ORB combined with Kimera's Firewall/server].

c. *4. The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to transmit said applet in said selected form to said server computer.*

402. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.3.b above. EX1009

(Fowlow) at 4:6-15, 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2. Specifically, claim 4 similarly recites a portion of claim 3, claiming only transmitting the request to an external resource "if said applet can not be compiled from said local resource" as opposed to claim 3's broader transmitting the request "if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource."

403. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 1, further comprising the step of: transmitting a request to an external resource [*e.g.*, Fowlow's class servers] for supplying said applet [*e.g.*, Fowlow's code or "applets"] in said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] if said applet can not be compiled from said local resource [*e.g.*, Kimera's compilation service] and directing said external resource to transmit said applet in said selected form to said server computer [*e.g.*, Fowlow's ORB combined with Kimera's Firewall/server].

4. Independent Claim 5

a. [5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:

404. I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 5 is limiting, it is my

opinion that a POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.a above. EX1009 (Fowlow) at Abstract, 3:27-42, 5:34-36, 7:11-5210:15-34, 10:45-53, 11:21-25, Fig. 2; EX1008 (Kimera) at 2; EX1001 at 3:21-24.

b. [5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms;

405. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.b above. EX1009 (Fowlow) at Abstract, 3:27-42, 10:18-34, 4:3-15, 10:45-54, 11:21-25, Fig. 2; EX1008 (Kimera) at 2.

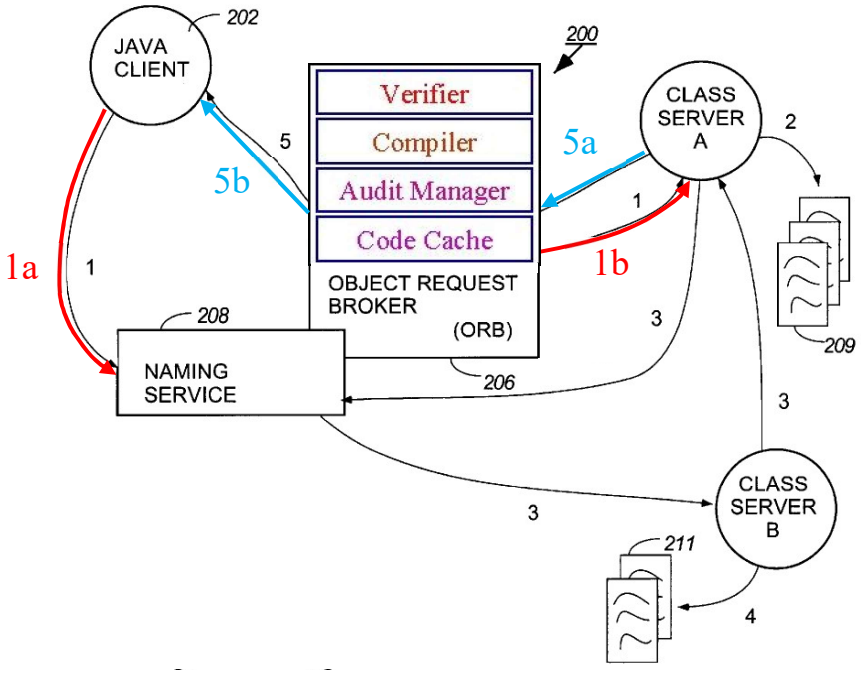
c. [5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer;

406. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.3.b above. EX1009 (Fowlow) at 4:6-15, 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2.

407. As Fowlow teaches, the ORB "allows a client a single point of access to the distributed object system in order to find the class files that it needs." EX1009 (Fowlow) at 10:21-27.

408. Given the arrangement of Kimera's Firewall/server in the Fowlow-Kimera combined system, a POSA would know that requests for applet code would

pass through Kimera's centralized server before being passed to the broader network, such as Fowlow's class servers. As requests pass through the Firewall/server, a POSA would have understood the "central compilation server" would determine whether the requested code was stored in this Code Cache. *Id.* If the code was present in the Code Cache, it would be served to the local computer from there. *See* EX1008 (Kimera) at 2. Returning recently cached results would reduce the burden on the network considerably, as well as return results to users/clients much faster. *See* §IV.C above. Indeed, this would shorten the applet retrieval loop to arrows 1a and 5b in the annotated image below:



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

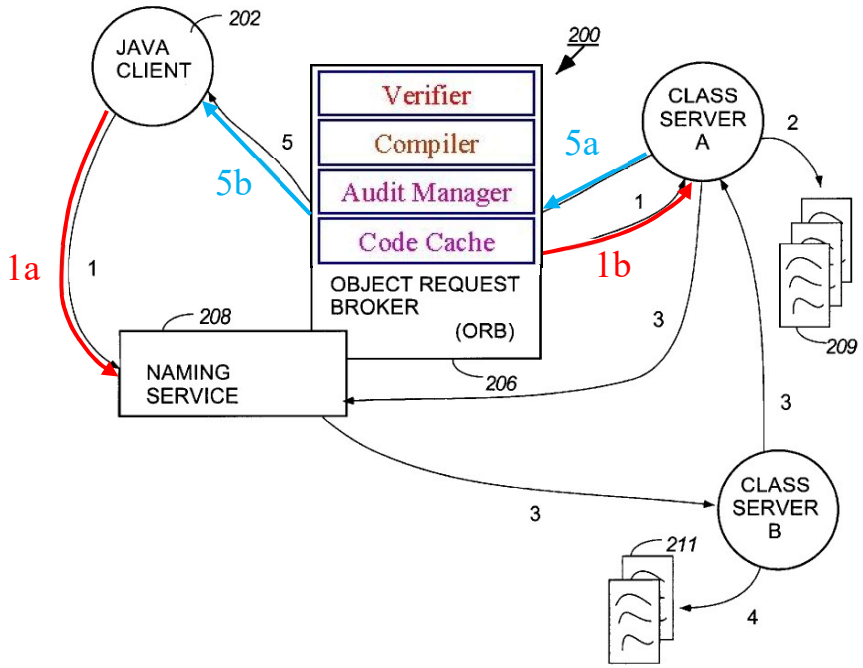
409. In sum, in my opinion, the Fowlow-Kimera combination teaches determining whether said applet is stored in said specified form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] in a local cache [*e.g.*, Kimera's Code Cache] and, if so, transmitting said applet in said specified form to said client computer [*e.g.*, arrow 5b].

d. [5c] c) if said applet is not stored in said selected form in said local cache,

410. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.3.b above. EX1009 (Fowlow) at 4:6-15, 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2.

411. For applet execution code not already stored in the selected form in the Code Cache of the ORB-Firewall/server of the Fowlow-Kimera system, a POSA would recognize that the request would be sent to the class servers as in the traditional Fowlow system. In other words, if the request could not be fulfilled with applet classes stored in Kimera's Code Cache, the request would be forwarded (arrow 1b in the annotated figure below) along to Fowlow's class servers to retrieve the applet in the selected form from the servers (arrow 5a in the annotated figure below). *See* EX1009 (Fowlow) at 3:31-42 ("In a method aspect, initially a client queries the object request broker to determine if there is an applet server available within the system that may be used to obtain particular applet execution code."),

11:57-12:11 ("Arrows (1) show how a Java client 202 may use the naming service 208 and the ORB 206 in order to determine a particular class server A that has access to a given class name that the client desires to be loaded. If this class name is contained within the file set 209 associated with class server A, then arrow (2) shows how class server A retrieves the file containing the class name from its file set 209.").



EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

412. In sum, in my opinion, the Fowlow-Kimera combination teaches determining if said applet is not stored in said selected form [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] in said local cache [*e.g.*, Kimera's Code Cache].

e. [5d] compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and

413. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.c above. EX1009 (Fowlow) at 10:50-56, 11:21-25, Fig. 2; EX1008 (Kimera) at 1, 2; EX1001 at Abstract, 3:49-54, 3:63-64, 4:50-52, 5:1-10, Fig. 1.

f. [5e] transmitting said applet in said selected form to said client computer.

414. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.d above. EX1009 (Fowlow) at 12:6-9, 13:43-46, Fig. 2; EX1008 (Kimera) at 1, 2.

5. Dependent Claims 6-8

a. 6. The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.

415. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.3.a above. EX1008 (Kimera) at 2.

b. *7. The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said form to said server computer.*

416. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.3.b above. EX1009 (Fowlow) at 4:6-15, 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2.

c. *8. The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resources, and directing said external resource to transmit said applet in said selected form to said server computer.*

417. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.3.c above. EX1009 (Fowlow) at 4:6-15, 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2.

6. Independent Claim 9

a. *[9pre] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:*

418. I am not offering an opinion as to whether the preamble of this claim is limiting. Regardless of whether or not the preamble of claim 9 is limiting, it is my

opinion that a POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.2.a above. EX1009 (Fowlow) at Abstract, 3:27-42, 5:34-36, 7:11-5210:15-34, 10:45-53, 11:21-25, Fig. 2; EX1008 (Kimera) at 2; EX1001 at 3:21-24.

b. [9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

419. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.b above. EX1009 (Fowlow) at Abstract, 3:27-42, 10:18-34, 4:3-15, 10:45-54, 11:21-25, Fig. 2; EX1008 (Kimera) at 2.

c. [9b] b) compiling an applet program module into an intermediate form program module;

420. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.c above. EX1009 (Fowlow) at 10:50-56, 11:21-25, Fig. 2; EX1008 (Kimera) at 1, 2; EX1001 at Abstract, 3:49-54, 3:63-64, 4:50-52, 5:1-10, Fig. 1.

421. Kimera teaches compiling Java bytecode into native machine code at the centralized Firewall/server. EX1008 (Kimera) at 2. A POSA would have understood that "an intermediate form program module" would include Java

bytecode, because the '248 patent describes "[t]he intermediate byte-code need not necessarily be Java byte-code." EX1001 at 3:38-39.

422. Further, a POSA would have found it obvious to compile code into Java bytecode to the extent a client required Java bytecode, such as for experimental or debugging purposes or for use in a Java-compliant browser. EX1009 (Fowlow) at 11:21-25; *see also* §§IV.D and IV.E above; *see also* EX1014 (Microsoft Computer Dictionary), "bytecode" definition at 69. Fowlow also teaches "an applet may be a collection of byte codes representing classes that are executed by a remote interpreter." EX1009 (Fowlow) at 10:49-56.

423. In sum, in my opinion, the Fowlow-Kimera combination teaches compiling an applet program module [*e.g.*, Fowlow's "applets" or "applet classes"] into an intermediate form program module [*e.g.*, Kimera's Java bytecode or Fowlow's "collection of byte codes representing classes that are executed by a remote interpreter"].

d. [9c] c) transmitting said applet in said selected form to said client computer.

424. A POSA would find that the Fowlow-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.2.d above. EX1009 (Fowlow) at 12:6-9, 13:43-46, Fig. 2; EX1008 (Kimera) at 1, 2.

425. In my opinion, the Fowlow-Kimera combination discloses, or at least teaches, each limitation of claim 9 of the '248 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow and Kimera.

7. Dependent Claims 10-13

a. *10. The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.*

426. The Fowlow-Kimera system includes various transformer programs that are used to transform code program modules for the client computer. For example, Fowlow discloses "filter 40" on ORB 11, which "may perform a variety of tasks, such as compression, encryption, tracing, or debugging, that are applied to communications to and from an object." EX1009 (Fowlow) at 6:5-8. File compression in software retrieval was well known at the time of the '248 patent, and it would have been obvious to a POSA to utilize compression for software components retrieved from computer networks and sent to client computers. *See* §IV.D above; *see also* EX1014 (Microsoft Computer Dictionary), "optimization" and "optimizing compiler" definitions at 343; *see also* EX1020 (GNU C Manual Page) at 16-17.

427. In my opinion, it would have been obvious to transform software components via compression to further satisfy the goals and limitations described in

Kimera, namely, reducing client system requirements and avoiding excessive communication between server computers and client computers. EX1008 (Kimera) at 1, 2. A POSA would have understood that compressing applets sent to the client computer enables the system to "make[] a sizeable reduction in computation and memory requirements of the end-points" because the received, compressed applets can be more readily processed on the client computers. *Id.* at 2

428. In my opinion, Fowlow's filter 40's performs a variety of transformation operations. "Filter 40, if being used, may perform a variety of tasks, such as compression, encryption, tracing, or debugging, that are to be applied to communications to and from an object." EX1009 (Fowlow) at 6:5-8. A POSA would have understood that Fowlow's filter 40's compression and encryption functions were each transforming operations at least because claim 11 of the '248 Patent recognizes "compressing computer programs" and "encrypting computer programs" as two possible transformer programs that transform intermediate program modules. EX1001 at 8:46-51.

429. Further, Fowlow's filter 40's tracing and debugging functionality suggests transforming the applets through optimization. EX1009 (Fowlow) at 6:5-8. Both tracing and debugging suggest programs that understand the input code; and both have been used to analyze programs and source code for the purpose of optimizations. *See* §IV.E above.

430. In my opinion, tracing and debugging result in better processing code, and these two functionalities are well-known inputs into more discrete optimizer functions. Even if Fowlow does not expressly describe substituting code fragments, a POSA would have understood the tracing and debugging functions of Fowlow suggest this as the logical next step. *See* §§IV.D and IV.E above (noting that a "debugger" "step[s] through the program, examine the data, and monitor conditions such as the values of variables," "tracing" is a technique to "analyze source code and running programs, looking for bottlenecks," and optimization is "the process of producing more efficient (smaller or faster) programs through selection and design of data structures, algorithms, and instruction sequences," which involves a deeper understanding of code, for example to embed variables into other variables, etc.).

431. Fowlow's filter 40 further functions to debug code. EX1009 (Fowlow) at 6:5-8. Debugging would involve ensuring the input code would not result in memory leak, or other concerns that would make the client device vulnerable to issues or attacks. Fowlow also includes Secure Protocol 42, which "secures the internet inter-ORB protocol and helps to transmit information through transport layer 38 in a secure fashion. This may mean integrity protection, confidentiality, etc." *Id.* at 6:43-47.

432. In my opinion, this Secure Protocol functions to verify input code and applet data transmitted through the transport layer 38. The '248 Patent defines

verification in a similar way: verification includes "analyz[ing] input code and determin[ing] areas that might not be safe." EX1001 at 5:38-39. In my opinion, filter 40 and Secure Protocol 42 transform the intermediate code by verifying the input code and applet data transmitted through the transport layer. EX1009 (Fowlow) at 6:43-47. *See also* §IV.E above.

433. Kimera discloses a "verification service [that] performs checking of incoming code for typesafety and conformance to interfaces." EX1008 (Kimera) at 2. Kimera's verification service applies "[t]amper-proof signatures, using a cryptographically strong hash function such as MD5…." *Id.* This verification service would be performed at the centralized Firewall/server to "enable[] the same services to be performed faster, cheaper, and without introducing security vulnerabilities." EX1008 (Kimera) at 1.

434. In my opinion, a POSA would have understood that applet code that had been checked by the verification service, and to which an MD5 hash function had been applied, would be a transformed intermediate program module, ready to be returned to the client device. This is similar to Fowlow's filter 40 performing encryption functions. EX1009 (Fowlow) at 6:5-8.

435. Finally, Kimera teaches that "[c]ompilation from Java bytecode into native machine code is another service that is best performed at a central location." EX1008 (Kimera) at 2. In my opinion, a POSA would have understood that a

compiler is a transformer program that acts on Java bytecode to produce a transformed intermediate program module, such as native machine code. *Id.*; *see also* §IV.D above.

436. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 9, further comprising the step of: transforming [*e.g.*, via Fowlow's filter 40, Kimera's verification service, Kimera's optimization, or Kimera's compilation services] said intermediate form program module [*e.g.*, Kimera's Java bytecode or Fowlow's "collection of byte codes representing classes that are executed by a remote interpreter"] into a transformed intermediate form program module [*e.g.*, Fowlow's compressed, encrypted, traced, or debugged applet, Kimera's MD5-signed applet, or Kimera's native machine code] with at least one transformer program [*e.g.*, Fowlow's filter 40, Kimera's verification service, or Kimera's compiler].

b. *11. The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring computer programs and encrypting computer programs.*

437. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.7.a above. EX1009 (Fowlow) at 6:5-8, 6:43-47; EX1008 (Kimera) at 1, 2; EX1001 at 5:38-39; EX1014

(Microsoft Computer Dictionary), "optimization "and optimizing compiler" definitions at 343; EX1020 (GNU C Manual Page) at 16-17.

438. I note that this claim can be met by just one of the transformer programs because the claim only requires transformer program to be at least one of the programs selected from the following list of computer programs: a verifying computer program, an optimizing computer program, a compressing computer program, a debugging computer program, a usage monitoring computer program, or an encrypting computer program. Regardless, Fowlow-Kimera renders obvious nearly all of the listed transformer programs.

439. For example, both Fowlow and Kimera teach verifier computer programs, namely, Fowlow's filter 40's encryption and debugging features and Kimera's verification service. EX1009 (Fowlow) at 6:5-8 ("Filter 40, if being used, may perform a variety of tasks, such as compression, encryption, tracing, or debugging, that are to be applied to communications to and from an object."); EX1008 (Kimera) at 2 ("The verification service performs checking of incoming code for typesafety and conformance to interfaces.").

440. Both Fowlow and Kimera also suggests taking advantage of optimizing programs to take advantage of Kimera's client's with reduced computation and memory requirements. EX1009 (Fowlow) at 6:5-8 ("Filter 40, if being used, may perform a variety of tasks, such as compression, encryption, tracing, or debugging,

that are to be applied to communications to and from an object."); EX1008 (Kimera) at 1, 2 ("This makes a sizable reduction in computation and memory requirements of the end-points.").

441. Both Fowlow and Kimera describe the use of encryption programs, including the application of encryption-related techniques including the application cryptographically strong hash functions such as MD5. EX1009 (Fowlow) at 6:5-8 ("Filter 40, if being used, may perform a variety of tasks, such as compression, **encryption**, tracing, or debugging, that are to be applied to communications to and from an object."); EX1008 (Kimera) at 2 ("Locating the verification service at a separate server requires that the code verified at the central service remain intact after verification. Tamper-proof signatures, using a cryptographically strong hash function such as MD5, can be utilized to ensure the safety of the code after it leaves the verifier.").

442. Finally, Fowlow's filter 40 teaches applying compression to applets. EX1009 (Fowlow) at 6:5-8 ("Filter 40, if being used, may perform a variety of tasks, such as **compression**, encryption, tracing, or debugging, that are to be applied to communications to and from an object.").

443. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs [*e.g.*, Fowlow's filter 40 or

Kimera's verification service], optimizing computer programs [*e.g.*, Fowlow's filter 40, including known optimization programs to enable Kimera's reduction in client computer computation requirements], compressing computer programs [*e.g.*, Fowlow's filter 40's compression], debugging computer programs [*e.g.*, Fowlow's filter 40 debugging function], usage monitoring computer programs and encrypting computer programs [*e.g.*, Fowlow's filter 40 encryption function or Kimera's cryptographic hashing].

c. 12. The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

444. Kimera discloses "[c]compilation from Java bytecode into native machine code is another service that is best performed at a central location." EX1008 (Kimera) at 2. Thus, a POSA would have understood that Kimera teaches using the Firewall/server's compiler to compile the transformed intermediate form program module (*e.g.*, Java bytecode) into machine specific binary code (*e.g.*, native machine code).

445. Kimera's disclosure matches that of the '248 patent, which states "[a] target compiler 52 compiles intermediate applet code 44 into an applet 54 in a processor specific format (binary) suitable for running natively on a given computing platform." EX1001 at 5:11-26. The "intermediate form program

module" may be Java byte code, according to the '248 patent. *Id.* at 3:38-39 ("The intermediate byte-code need not necessarily be Java byte-code.").

446. In sum, in my opinion, the Fowlow-Kimera combination teaches the method of claim 11, further comprising the step of: compiling [*e.g.*, via Kimera's compiler] said transformed intermediate form program module [*e.g.*, Fowlow or Kimera's Java bytecode, as transformed by a verifier, optimizer, compression, debugger, or encryption transformer] into machine specific binary code [*e.g.*, Kimera's native machine code] with a target compiler [*e.g.*, Kimera's compiler].

d. 13. The method of claim 10, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

447. A POSA would find that the Fowlow-Kimera combination renders this claim obvious, for the same reasons as explained in §VIII.C.7.c above. EX1008 (Kimera) at 2; EX1001 at 3:38-39, 5:11-36.

IX. CONCLUSION

448. In my opinion claim 1-13 of the '248 Patent are obvious over the prior art.

I declare that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code.



April 8, 2024

Date

Erez Zadok, Ph.D.

CLAIM APPENDIX OF THE CHALLENGED CLAIMS

[1pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

[1a] a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

[1b] b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and

[1c] c) transmitting said applet in said selected form to said client computer.

[2] The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.

[3] The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said selected form to said server computer.

[4] The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to transmit said applet in said selected form to said server computer

[5pre] A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:

[5a] a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms;

[5b] b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer;

[5c] c) if said applet is not stored in said selected form in said local cache, compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and transmitting said applet in said selected form to said client computer.

[6] The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.

[7] The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said form to said server computer.

[8] The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resources, and directing said external resource to transmit said applet in said selected form to said server computer.

[9] A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality [of] forms of the applet, comprising:

[9a] a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality [of] forms;

[9b] b) compiling an applet program module into an intermediate form program module;

[9c] c) transmitting said applet in said selected form to said client computer.

[10] The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.

[12] The method of claim 11, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

[13] The method of claim 10, further comprising the step of: compiling said transformed intermediate form program module into machine specific binary code with a target compiler.