

**UNITED STATES PATENT AND TRADEMARK OFFICE**

---

**PATENT TRIAL AND APPEAL BOARD**

---

Capital One, National Association,  
Petitioner

v.

Implicit, LLC,  
Patent Owner

---

CASE: IPR2024-00878  
U.S. PATENT NO. 7,774,740

---

**DECLARATION OF EREZ ZADOK, PH.D., IN SUPPORT OF PETITION  
FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 7,774,740**

## TABLE OF CONTENTS

|       |                                                                            |    |
|-------|----------------------------------------------------------------------------|----|
| I.    | Introduction.....                                                          | 1  |
| II.   | Qualifications.....                                                        | 2  |
| III.  | Understanding of Relevant Legal Principles.....                            | 13 |
|       | A. Claim Construction.....                                                 | 13 |
|       | B. Obviousness.....                                                        | 15 |
| IV.   | Technical Background.....                                                  | 17 |
|       | A. General Computer Operations.....                                        | 17 |
|       | B. Web Technologies.....                                                   | 23 |
|       | C. Caching Technologies.....                                               | 34 |
|       | D. Programming Languages, Compilers and Interpreters,<br>Optimization..... | 43 |
|       | E. Transforming Data and Files.....                                        | 52 |
|       | F. Intermediate Compiler Stages.....                                       | 56 |
| V.    | U.S. Patent No. 7,774,740.....                                             | 60 |
|       | A. Summary of the '740 Patent.....                                         | 60 |
|       | B. Prosecution History.....                                                | 63 |
|       | C. Person of Ordinary Skill in the Art.....                                | 68 |
| VI.   | Overview of the Prior Art.....                                             | 69 |
|       | A. Fowlow (EX1009).....                                                    | 69 |
|       | B. Kimera (EX1008).....                                                    | 74 |
|       | C. Parthasarathy (EX1007).....                                             | 77 |
|       | D. Angles (EX1010).....                                                    | 81 |
| VII.  | Overview of the Claims.....                                                | 83 |
| VIII. | The Challenge Grounds.....                                                 | 86 |
|       | A. Ground 1: Claims 11-19 are Obvious in view of Fowlow.....               | 86 |
|       | 1. Independent Claim 11.....                                               | 86 |

|    |                                                                                                                                                                                                       |     |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| a. | [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising: .....                           | 86  |
| b. | [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and .....                               | 89  |
| c. | [11b] in response to receiving the request: compiling the application into a compiled form;.....                                                                                                      | 91  |
| d. | [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and ..... | 94  |
| e. | [11d] sending the transformed form of the application to the client computer. ....                                                                                                                    | 102 |
| 2. | Dependent Claims 12-19.....                                                                                                                                                                           | 104 |
| a. | [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.....                                                                       | 104 |
| b. | [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.....                                                                                           | 105 |
| c. | [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header. ....                                                                                        | 107 |
| d. | [15] The method of claim 11 wherein the identified form of the application is a web page.....                                                                                                         | 108 |
| e. | [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.....                                                                                            | 109 |
| f. | [17] The method of claim 11 wherein transforming further comprises verification of the compiled form....                                                                                              | 110 |
| g. | [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form..                                                                                                | 112 |
| h. | [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response. ....                                                                  | 114 |

|    |                                                                                                                                                                                                       |     |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| B. | Ground 2: Claims 11-21 are Obvious in view of Fowlow in view of Kimera.....                                                                                                                           | 115 |
| 1. | Motivation to Combine Fowlow and Kimera .....                                                                                                                                                         | 115 |
| 2. | Independent Claim 11 .....                                                                                                                                                                            | 124 |
| a. | [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising: .....                           | 124 |
| b. | [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and .....                               | 124 |
| c. | [11b] in response to receiving the request: compiling the application into a compiled form;.....                                                                                                      | 127 |
| d. | [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and ..... | 133 |
| e. | [11d] sending the transformed form of the application to the client computer. ....                                                                                                                    | 140 |
| 3. | Dependent Claims 12-21.....                                                                                                                                                                           | 141 |
| a. | [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.....                                                                       | 141 |
| b. | [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.....                                                                                           | 141 |
| c. | [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header. ....                                                                                        | 141 |
| d. | [15] The method of claim 11 wherein the identified form of the application is a web page.....                                                                                                         | 141 |
| e. | [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.....                                                                                            | 142 |
| f. | [17] The method of claim 11 wherein transforming further comprises verification of the compiled form....                                                                                              | 142 |

|      |                                                                                                                                                                                                                   |     |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| g.   | [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form..                                                                                                            | 144 |
| h.   | [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response. ....                                                                              | 146 |
| i.   | [20pre] The method of claim 11, further comprising:..                                                                                                                                                             | 146 |
| i.   | [20a] storing a plurality of available forms of the application; .....                                                                                                                                            | 146 |
| ii.  | [20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and .....                     | 148 |
| iii. | [20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming. .... | 149 |
| j.   | [21pre] The method of claim 11, further comprising:..                                                                                                                                                             | 152 |
| i.   | [21a] storing a plurality of available forms of the application; .....                                                                                                                                            | 152 |
| ii.  | [21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and .....                     | 153 |
| iii. | [21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling. ....                  | 153 |
| C.   | Ground 3: Claims 11-12 and 14-21 are Obvious in view of Parthasarathy in view of Kimera.....                                                                                                                      | 154 |
| 1.   | Motivation to Combine Parthasarathy and Kimera .....                                                                                                                                                              | 154 |
| 2.   | Independent Claim 11 .....                                                                                                                                                                                        | 161 |
| a.   | [11pre] A method operating on a computer system, having a client computer and a server computer, for                                                                                                              |     |

|    |                                                                                                                                                                                                       |     |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|    | managing requests to the server computer, the method comprising: .....                                                                                                                                | 161 |
| b. | [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and .....                               | 164 |
| c. | [11b] in response to receiving the request: compiling the application into a compiled form;.....                                                                                                      | 168 |
| d. | [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and ..... | 170 |
| e. | [11d] sending the transformed form of the application to the client computer. ....                                                                                                                    | 177 |
| 3. | Dependent Claims 12-21.....                                                                                                                                                                           | 179 |
| a. | [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.....                                                                       | 179 |
| b. | [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header. ....                                                                                        | 181 |
| c. | [15] The method of claim 11 wherein the identified form of the application is a web page.....                                                                                                         | 182 |
| d. | [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.....                                                                                            | 183 |
| e. | [17] The method of claim 11 wherein transforming further comprises verification of the compiled form....                                                                                              | 184 |
| f. | [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form..                                                                                                | 186 |
| g. | [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response. ....                                                                  | 188 |
| h. | [20pre] The method of claim 11, further comprising:..                                                                                                                                                 | 190 |
| i. | [20a] storing a plurality of available forms of the application; .....                                                                                                                                | 190 |

|      |                                                                                                                                                                                                                   |     |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ii.  | [20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and .....                     | 193 |
| iii. | [20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming. .... | 195 |
| i.   | [21pre] The method of claim 11, further comprising:..                                                                                                                                                             | 197 |
| i.   | [21a] storing a plurality of available forms of the application; .....                                                                                                                                            | 197 |
| ii.  | [21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and .....                     | 197 |
| iii. | [21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling. ....                  | 198 |
| D.   | Ground 4: Claim 13 is Unpatentable Under 35 U.S.C. §103(a) as Obvious over Parthasarathy in view of Kimera and Angles .....                                                                                       | 199 |
| 1.   | Motivation to Combine .....                                                                                                                                                                                       | 199 |
| 2.   | Dependent Claim 13 .....                                                                                                                                                                                          | 203 |
| a.   | 13. The method of claim 11 wherein the identified form of the application is specified using a cookie.....                                                                                                        | 203 |
| IX.  | Conclusion .....                                                                                                                                                                                                  | 205 |

### LIST OF EXHIBITS

| Exhibit No. | Description                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1001        | U.S. Patent No. 7,774,740 (the “’740 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, <u>Slim Binaries</u> , 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, <u>Kimera: A System Architecture for Networked Computers</u> , 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        | <i>Reserved</i>                                                                                                                                                                                                                                     |
| 1012        | Affidavit of Nathaniel E. Frank-White, dated 2/9/2024                                                                                                                                                                                               |
| 1013        | <i>Reserved</i>                                                                                                                                                                                                                                     |
| 1014        | Microsoft Computer Dictionary, 3rd ed., 1997                                                                                                                                                                                                        |
| 1015        | Hypertext Transfer Protocol – HTTP/1.1, RFC 2068, The Internet Engineering Task Force (IETF), January 1997.<br>[Download from <a href="https://www.rfc-editor.org/rfc/pdf/rfc2068.txt.pdf">https://www.rfc-editor.org/rfc/pdf/rfc2068.txt.pdf</a> ] |

| Exhibit No. | Description                                                                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1016        | HTML 4.0 Specification, The World Wide Web Consortium (W3C) Recommendation, December 18, 1997. [Download from <a href="https://www.w3.org/TR/REC-html40-971218/html40.pdf">https://www.w3.org/TR/REC-html40-971218/html40.pdf</a> ]        |
| 1017        | HTTP State Management Mechanism, RFC 2109, The Internet Engineering Task Force (IETF), February 1997. [Download from <a href="https://www.rfc-editor.org/rfc/pdf/rfc2109.txt.pdf">https://www.rfc-editor.org/rfc/pdf/rfc2109.txt.pdf</a> ] |
| 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. [ <a href="https://www.rfc-editor.org/rfc/pdf/rfc2187.txt.pdf">https://www.rfc-editor.org/rfc/pdf/rfc2187.txt.pdf</a> ]                                 |
| 1020        | GNU C Manual Page documentation, 1993                                                                                                                                                                                                      |
| 1021        | Persistent Client State HTTP Cookies, October 1996                                                                                                                                                                                         |
| 1022        | <i>Reserved</i>                                                                                                                                                                                                                            |
| 1023        | <i>Reserved</i>                                                                                                                                                                                                                            |
| 1024        | <i>Reserved</i>                                                                                                                                                                                                                            |
| 1025        | <i>Implicit, LLC v. Trend Micro, Inc.</i> , No. 6:16-cv-00080-JRG, Dkt. No. 115, Markman Order (Mar 19, 2017)                                                                                                                              |

## 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. 7,774,740 (“the ’740 Patent”).

2. I was asked to review the following:

- U.S. Patent No. 7,774,740 to Balassanian titled “Application server” (“the ’740 Patent”) (EX1001);
- Prosecution History of U.S. Patent Application No. 11/241,985 (EX1004);
- 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
- U.S. Patent No. 5,933,811 to Angles et al. (“Angles”) (EX1010).

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 '740 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<sup>1</sup> (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.

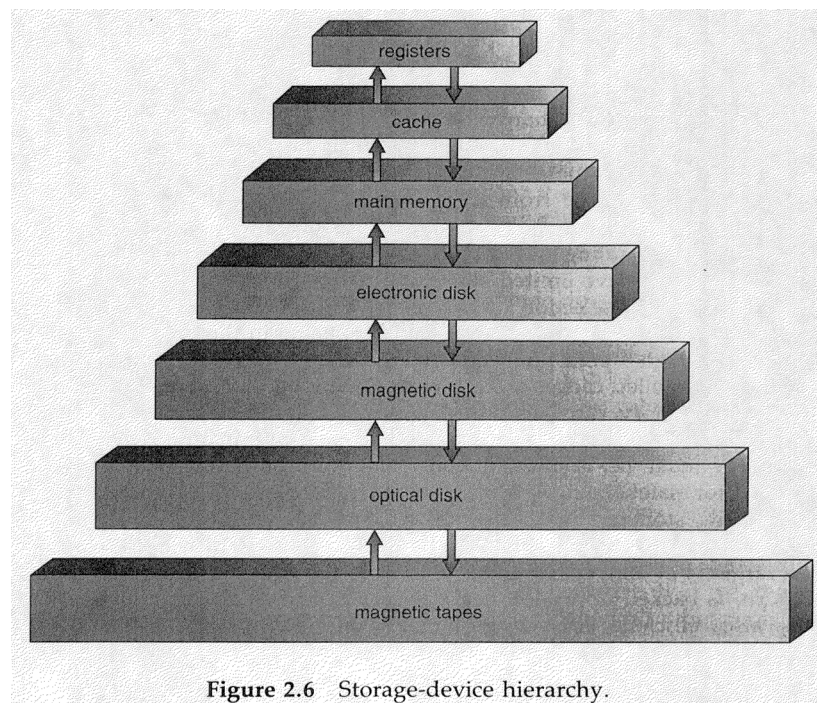


Figure 2.6 Storage-device hierarchy.

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 43-44.

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<sup>1</sup>. 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 in 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\\_\(software\)](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.

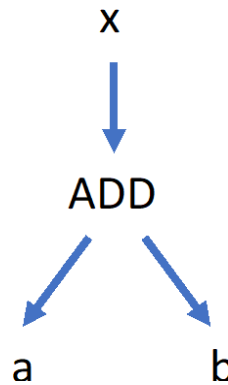
105. In stage 3, the “assembler” is “[a] program that converts assembly language programs, which are understandable by humans, into executable machine

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. 7,774,740**

148. The '740 Patent is titled "Application server" and was filed on October 4, 2005 and issued on August 10, 2010. EX1001 at cover.

149. I understand the '740 Patent is a continuation of U.S. Patent Application Ser. No. 09/968,704 (now U.S. Patent No. 6,976,248). 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 '740 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 '740 Patent do not change whether the '740 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 '740 Patent.

**A. Summary of the '740 Patent**

151. The '740 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 2:2:17-21. The requested and executed applets are described in the '740 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 3:13-18.

152. The '740 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 1:11-13. The '740 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 3:46-55.

153. The '740 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 3:60-65. 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 3:66-4:3.

154. If the applet cannot be built from local resources, the '740 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 4:11-19. “The applet server manager 22 ... may request the local resources 26 and transformers 28 it needs to complete building the applet itself.” *Id.* at 4:15-19. “After the applet server manager 22 builds the applet, the network interface 20 transmits the applet to the requesting client computer A12.” *Id.* at 4:8-10.

155. Figure 1 of the '740 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.

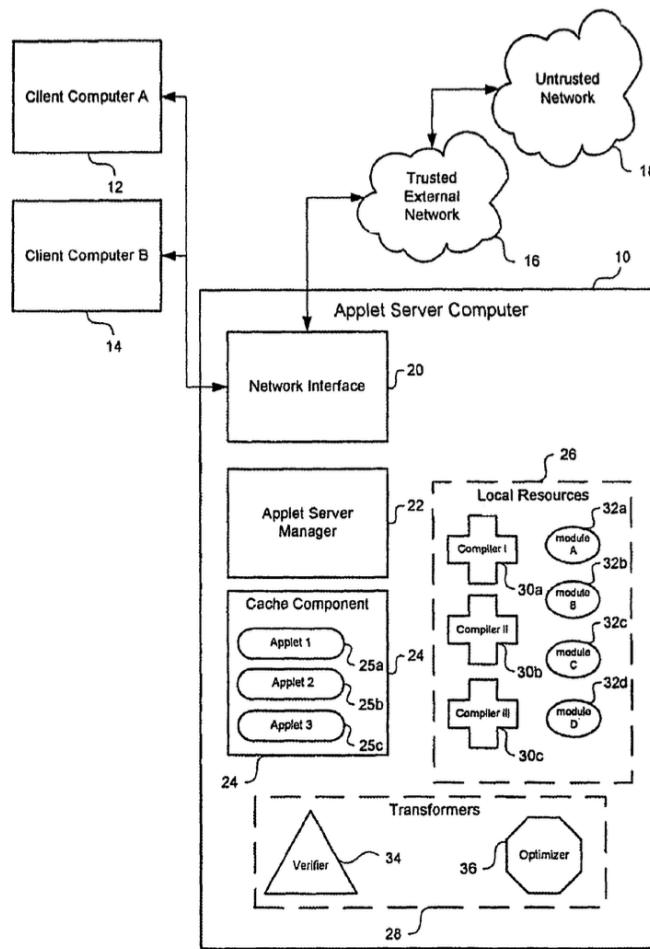


**Fig. 1**

*Id.* at Fig. 1, 3:46-4:19.

156. This applet building and downloading scheme was known before the priority date of the '740 Patent, as I discuss in more detail below.

## **B. Prosecution History**

157. The '740 Patent was originally filed with a single claim, which was rejected as anticipated by United States Patent No. 6,253,228 to Ferris ("Ferris"). EX1004 ('740 Prosecution History) at 20, 40-43. That claim differed substantially

from claim 11 challenged herein, and was cancelled in response to the rejection. *Id.* at 53.

158. In its place, the applicant filed new claims 2-34. *Id.* at 53-57. Before the next review, the applicant amended certain of these new claims, and added claims 35-46. *Id.* at 72-81. Among these claims was original claim numbers 13-18, 20-21, and 36-38, which ultimately issued as claims 11-21. *Id.* at 205.

159. Original claim 13 was rejected over Ferris. There appears to have been some confusion as to the basis for the rejection. In introducing Ferris, the examiner included the heading “Claim Rejections – 35 USC § 103” and quoted a portion of 35 U.S.C. § 103(a) before stating “Claims 2-46 are rejected under 35 U.S.C. 102(e) as being unpatentable over Ferris.” *Id.* at 86. After this introduction, the examiner cited Ferris as teaching “at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms,” “in response to receiving the request: generating the identified form of the application from an intermediate [or] a source form of the applications,” “transforming the application into a transformed version of the generated form of the application; and sending the transformed form of the application to the client computer.” *Id.* at 86-89. The examiner also cited Ferris in rejecting each of the other original claims. *Id.* at 86.

160. The applicant responded to the rejection, noting “[s]ince *Ferris* is the only reference in the *Final Office Action*, and since the prior rejection [to original claim 1] was based upon 35 U.S.C. § 102(e) and the same reference, Applicant assumes that the current rejection is under 35 U.S.C. § 102(e) as well.” *Id.* at 100 (emphasis in original). The applicant did not substantively amend its claims in response to the rejection, instead amended certain claims “to correct typographical errors.” *Id.*

161. The applicant then argued that the ’740 Patent and *Ferris* were different, stating “the present invention is an applet *server* which accepts requests for applets from *client computers*, and fulfills those requests.” *Id.* at 100-101 (emphasis in original). “By contrast,” the applicant argued, “*Ferris* integrates applets running on a *client* with application logic for applications running on the *server*, the reverse situation, to synchronize states and to recognize user actions in a browser (i.e., input data).” *Id.* at 101. Put differently, it appears the applicant argued *Ferris* addresses applets already running on a client, and therefore *Ferris* does not anticipate delivering applets to the client. *Id.*

162. The applicant further argued that *Ferris* instead involves “a client [that] contacts the HTTP server and requests to receive or transmit ***data***, not an application....” *Id.* at 102 (emphasis in original). Regarding the step of “in response to receiving the request, generating the identified form of the application from a

source form of the application,” the applicant argued that Ferris “simply talks about synchronizing *information* between the browser and server, using applets to retrieve user input with an applet controller managing the communication.” *Id.* This “has nothing to do with identifying a requested application and the form of the application” and, according to the applicant, “[t]he Examiner never addresses the step of generating the identified form of the application from a source form of the application.” *Id.* at 102-103. Because in the applicant’s view Ferris did not teach the generating step, “there can be no transforming of the generated form of the application,” as recited in the transforming step. *Id.* at 103.

163. Next, the applicant argued that the examiner “has confused the ‘form’ of the application in the present invention, i.e., whether it is source code, binary code or intermediate code, with the FORM element of HTML discussed in *Ferris*.” *Id.* at 101. Per the applicant, “the FORM element used in HTML provides the ability to create a fill-in form for the client and permits the user to enter information through a limited number of components including checkboxes, radio boxes, pull-down lists, text windows, and menus.” *Id.* The applicant argued “[t]hese are clearly not the same.” *Id.*

164. Following a telephone interview in February 3, 2010, the applicant included substantive amendments to the claims on February 10, 2010. *Id.* at 133, 144. In the interview, the examiner noted, among other things, that if “the limitations

of claims 19 and 22 [were] brought into claim 13, ... the case would be allowable.” *Id.* at 144. The applicant obliged, amending claim 13 to include “wherein transforming comprises [1] execution and [2] compression of the compiled form,” which were originally claimed in original claims 19 and 22, respectively. *Id.* at 136, 144. The applicant also made what it calls “minor amendments” to original claims 13, 18, 20-21, and 36. *Id.* at 144-145, 136 (original claim 13), 137 (original claims 18, 20, and 21), 139 (original claim 36).

165. Another telephone interview occurred on February 17, 2010 which included an examiner’s amendment to some of the claims. *Id.* at 154. Claim 13 was not amended in this examiner’s amendment, while the “minor amendments” from the applicant’s February 10<sup>th</sup> response were included. *See* 156, 157. The examiner also deemed these amended claims to be allowable subject matter, and issued a notice of allowance. *Id.* at 160, 153.

166. After the notice of allowance, the applicant proposed additional “minor amendments” to best represent “the agreement reached between Applicant’s counsel and the Examiner in the February 17, 2010, interview.” *Id.* at 178. None of the claims challenged in this declaration were amended in this final amendment.

167. A second notice of allowance issued on March 17, 2010 and the ’740 Patent issued on August 10, 2010. *Id.* at 190; EX1001 at cover.

**C. Person of Ordinary Skill in the Art**

168. I understand that an assessment of claims of the '740 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.

169. In my opinion, a POSA at the time of the priority date of the '740 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.

170. 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 '740 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. Fowlow (EX1009)**

171. 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.*

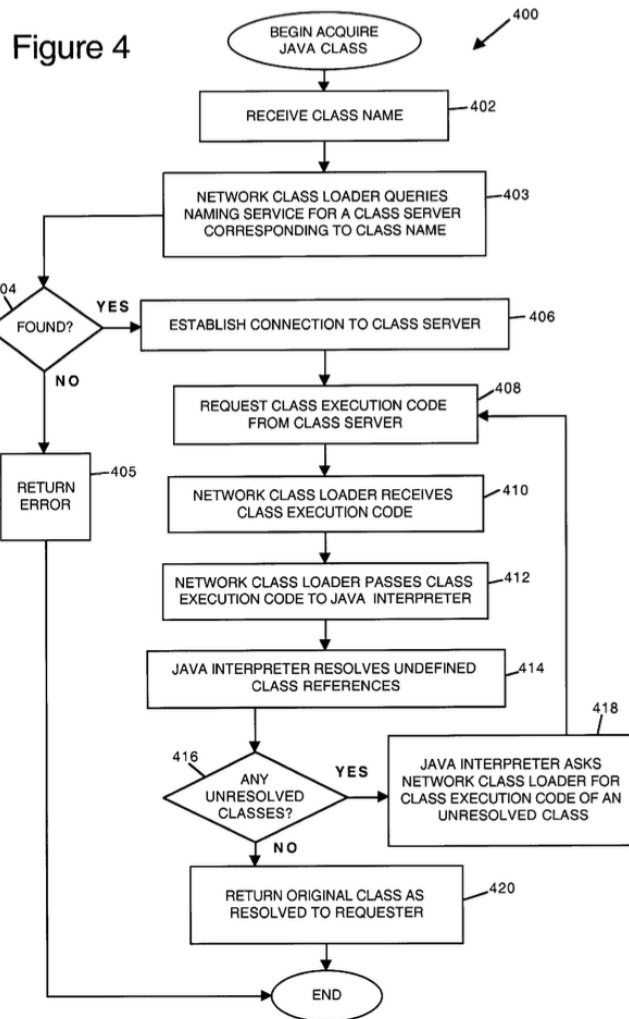
172. The original assignee of Fowlow was Sun Microsystems, Inc., which invented and developed the Java programming language. *Id.* at cover, 1:39-43. Fowlow describes "a client enabled to load and run Java applets in a distributed

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.*

173. 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.

174. 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.

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

176. 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.

177. 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.

178. 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.*

179. 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.

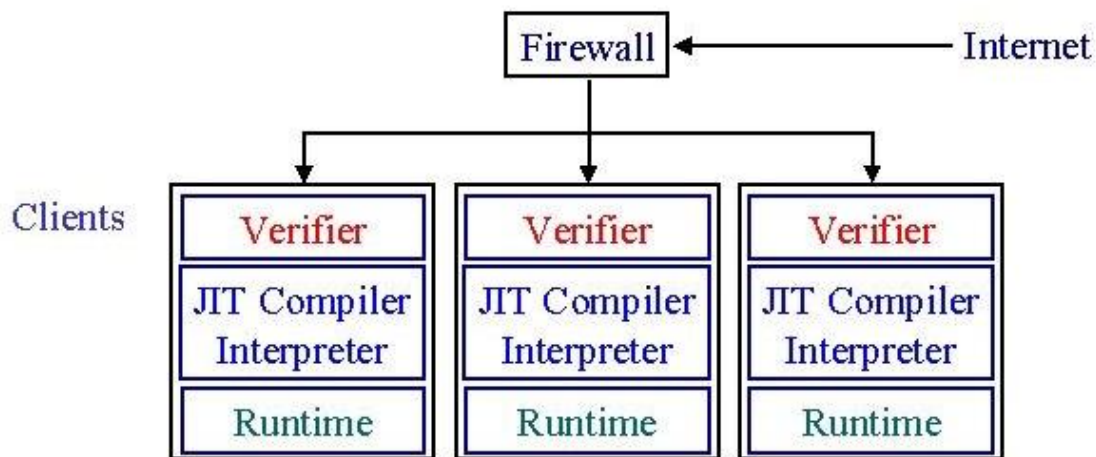
**B. Kimera (EX1008)**

180. 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 '740 Patent.

181. 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 “make[] 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.

182. 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 more 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.

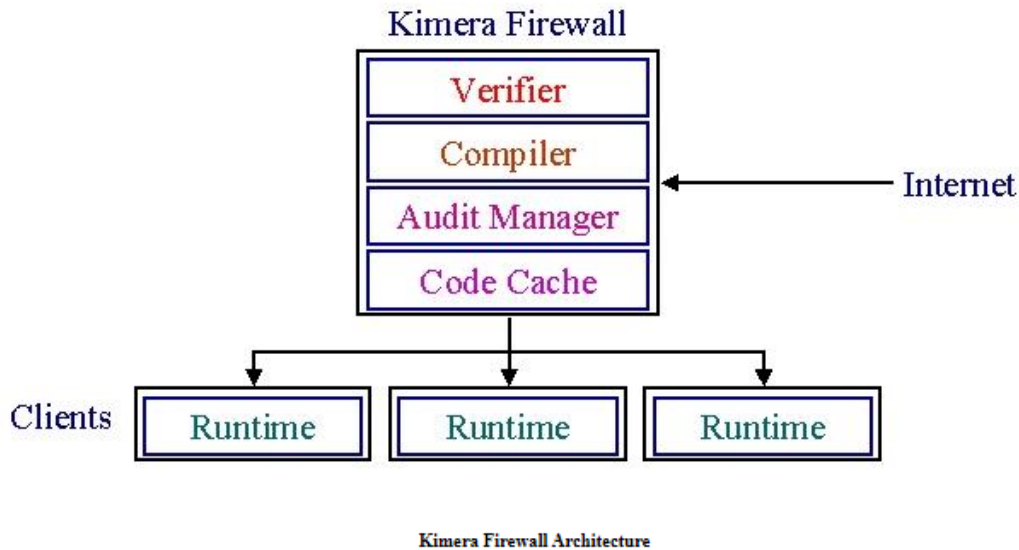
*Id.* at 1 (“Kimera Figure 1”).<sup>1</sup>

183. 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

---

<sup>1</sup> 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”.

evident; the system is “easy to administer, provides enterprise-wide security, supports safe auditing, and reduces Java client resource requirements.” *Id.*



*Id.* at 1 (“Kimera Figure 2”).

184. 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.*

185. 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.*

186. 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.

**C. Parthasarathy (EX1007)**

187. 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 on December 12, 1996. *Id.*

188. 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.

189. 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.

190. 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:

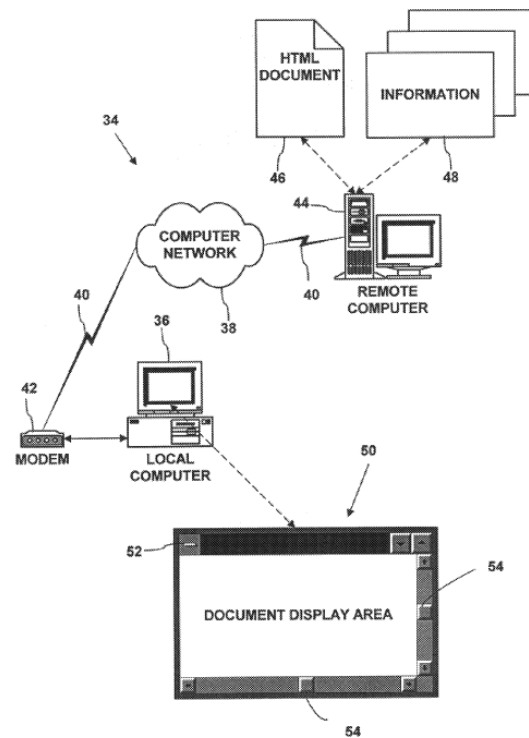


FIG. 2

*Id.* at Fig. 2.

191. 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.

192. 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.

193. 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.

194. 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.”).

**D. Angles (EX1010)**

195. U.S. Patent 5,933,811 to Angles et al. (“Angles”) is titled “System and method for delivering customized advertisements within interactive communication systems.” EX1010 (Angles) at cover. Angles was applied for on August 20, 1996 and issued on August 3, 1999. *Id.*

196. Angles is generally directed towards “a system and method for delivering customized electronic advertisements in an interactive communication system” such as the Internet. *Id.* at Abstract. “The customized advertisements are selected based on consumer profiles and are then integrated with offerings maintained by different content providers.” *Id.*

197. “Focusing now on the consumer computer 12 as illustrated in FIG. 4, the consumer computer 12 is a device which allows a consumer to interact with the communication medium 20. In the preferred embodiment, the consumer computer 12 is a conventional computer which is equipped with a conventional modem.” *Id.* at 10:20-25. “In the preferred embodiment, the consumer computer 12 utilizes several operational modules including a consumer browser module 40, a consumer member code 22 and an advertising storage medium 44.” *Id.* at 10:43-46. “The consumer member code module 22 stores a code which uniquely identifies each consumer. ... A copy of the consumer member code 22 is [] stored on the consumer computer 12 in a ‘cookie’.” *Id.* at 10:60-11:4.

198. “A ‘cookie’ is a small piece of information which a web server (via a CGI script) can store with a web browser and later read back from that browser. This is useful for having the browser remember some specific information across several pages; for example, when the consumer browses through a ‘virtual shopping mall’ and add items to his ‘shopping cart,’ a list of the items he has picked up is kept in

the consumer browser's cookie file so that the consumer can pay for all the items at once he has finished shopping.” *Id.* at 11:5-13.

199. “To create a cookie,” according to Angles, “a web server sends a ‘Set-Cookie’ HTTP header line in response to a URL access from a browser.” *Id.* at 11:13-15. “Whenever the browsing software sends a HTTP request for a URL on a server for which it has stored cookies, it includes a line in the form: Cookie: NAME=VALUE; NAME=VALUE; ... which lists all cookies that apply to that particular URL.” *Id.* at 11:27-31. A POSA would have recognized that while Angles discloses the use of cookies in Web systems for advertisements, they were already well known, standardized, and used in various contexts by 1998. See §IV.B above.

## VII. OVERVIEW OF THE CLAIMS

200. I have been asked to give opinions about whether the elements of claims 11-21 of the '740 Patent are disclosed, taught, and/or suggested by Fowlow, Kimera, Parthasarathy, and Angles. Below, I have reproduced those claims in their entirety:

[11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising:

[11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and

- [11b] in response to receiving the request: compiling the application into a compiled form;
  - [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and
  - [11d] sending the transformed form of the application to the client computer.
- [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.
- [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.
- [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header.
- [15] The method of claim 11 wherein the identified form of the application is a web page.
- [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.
- [17] The method of claim 11 wherein transforming further comprises verification of the compiled form.

[18] The method of claim 11 wherein transforming further comprises optimization of the compiled form.

[19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response.

[20pre] The method of claim 11, further comprising:

[20a] storing a plurality of available forms of the application;

[20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and

[20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming.

[21pre] The method of claim 11, further comprising:

[21a] storing a plurality of available forms of the application;

[21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and

[21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling.

## VIII. THE CHALLENGE GROUNDS

201. 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 | Fowlow                            | 11-19        |
| 2      | Obviousness | Fowlow and Kimera                 | 11-21        |
| 3      | Obviousness | Parthasarathy and Kimera          | 11-12, 14-21 |
| 4      | Obviousness | Parthasarathy, Kimera, and Angles | 13           |

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

### A. Ground 1: Claims 11-19 are Obvious in view of Fowlow

203. Independent claim 11 and dependent claims 12-19 are obvious over Fowlow. Below I explain how each element of claims 11-19 are disclosed, taught, and/or suggested by Fowlow.

#### 1. Independent Claim 11

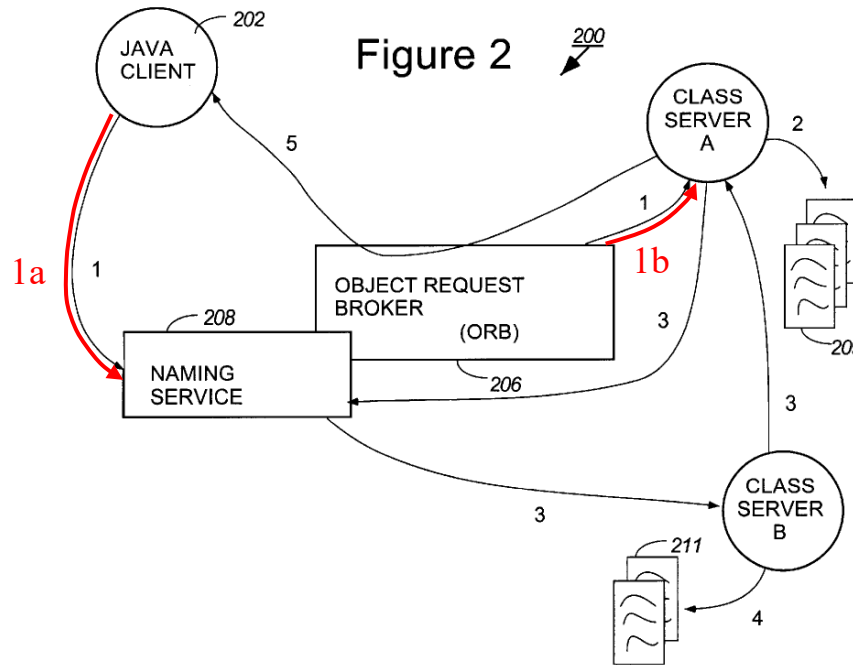
- a. [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising:*

204. 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 11 is limiting, it is my opinion that it is disclosed by Fowlow.

205. 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.

206. 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 files 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:



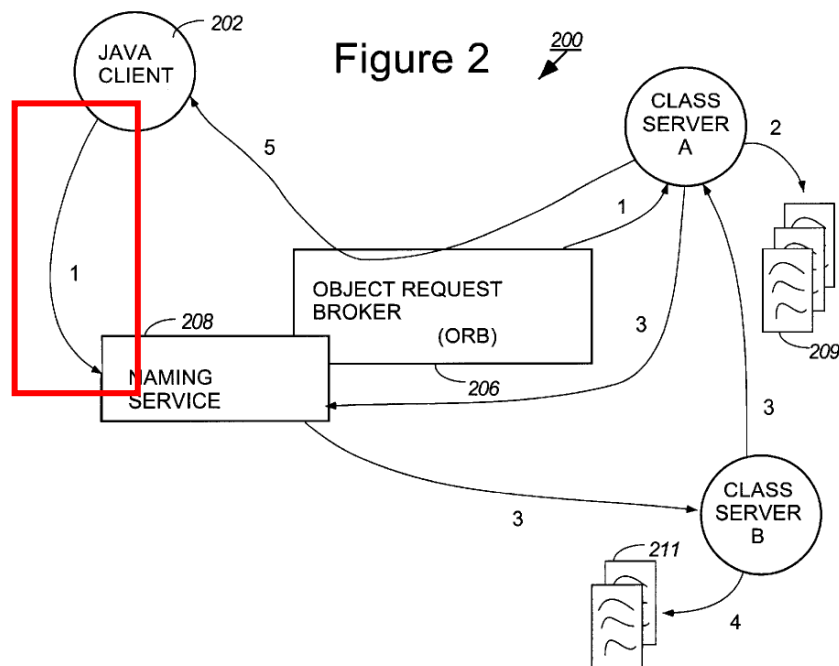
*Id.* at Fig. 2 (annotated).

207. 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).

208. In sum, in my opinion, to the extent the preamble is limiting, Fowlow teaches a method operating on a computer system [*e.g.*, distributed object system], having a client computer [*e.g.*, Java client 202] and a server computer [*e.g.*, ORB 206, Class Servers A, B], for managing requests [*e.g.*, requests for class code] to the server computer.

- b. [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and

209. 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 in a location independent manner.” *Id.* at 10:18-34, Abstract, 4:3-15, Fig. 2. For every request, a POSA would have understood the ORB as receiving a request from the client for an application.



*Id.* at Fig. 2 (annotated).

210. Each request in Fowlow also identifies the form of the desired application. 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 211 within a distributed object system 200.” *Id.* at 11:21-25. A POSA would have recognized that in this example, in addition to identifying an application, a Java client would be identifying a form of the application (*e.g.*, Java classes).

211. The identified form of the application, such as the language of the native machine, could be understood from the request itself. 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 3:19-23 (“Binary format

refers to processor specific machine instructions suitable for running natively on a given computing platform.”).

212. A POSA would have also understood that web clients report their architecture and other useful client-side properties to web servers as part of the HTTP protocol. This is important so servers could return code that can be executed and properly displayed given the specific client’s operating system, architecture, display size and resolution, etc. *See* §IV.B above.

213. In sum, in my opinion, Fowlow teaches at the server computer [*e.g.*, ORB 206], receiving a request from the client computer [*e.g.*, client 202], the request identifying an application [*e.g.*, “particular applet execution code”] and identifying a form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes].

*c. [11b] in response to receiving the request: compiling the application into a compiled form;*

214. As an initial matter, compiling software applications in response to a request was well known at the time of the ’740 Patent. *See* §IV.D above. This concept was not novel to the ’740 Patent.

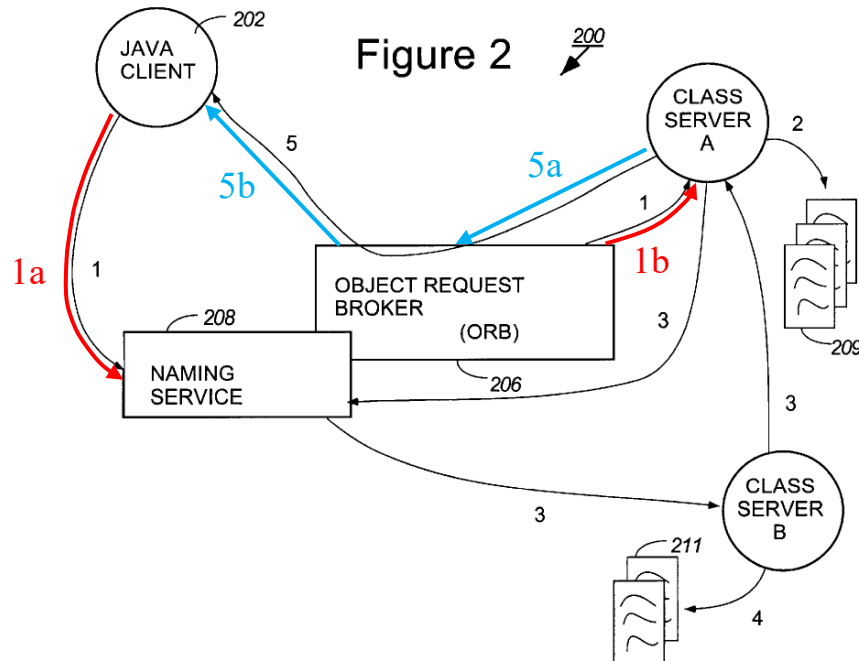
215. Fowlow teaches that “[a]n ‘applet’ may be a collection of byte codes representing classes that are executed by a remote interpreter, such as Java classes ... may also be **compiled** before execution or may even undergo a combination of

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 application.

216. 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.”).

217. Finally, a POSA would have understood that the application would be compiled in response to a request for that application in the identified form: there is little point to attempt and pre-compile all possible applications that all clients may ever request, as that would waste a lot of resources; instead, a POSA would have understood that compilation is more efficiently conducted on an as-needed basis (*i.e.*, on-the-fly). *See also* §IV.C above.

218. 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.” *Id.* at 3:27-42. The request is received at the ORB as shown by arrow 1a below and the application code is retrieved from a class server as shown by arrows 1b and 5a. *See id.* at 11:57-12:9. This application code “may also be **compiled** before execution or may even undergo a combination of interpretation/compilation before execution.” *Id.* at 10:50-56.



*Id.* at Fig. 2 (annotated).

219. “In a first step 402 [of the application retrieval process], the class name that is desired to be loaded [*i.e.*, the request] is received. Next, the class server object

located somewhere within the distributed object system that has access to the desired class name must be determined. ... [After] the class server has been found, the NCL [network class loader] requests the class execution code from the class server that corresponds to the desired class name in step 408.” *Id.* at 13:18-35. “This step is illustrated graphically in FIG. 2 by arrows (1).” *Id.*

220. A POSA would have understood that any compiling would have been dependent on the “first step 402” in which the desired class name is received. Put differently, without first receiving the request illustrated as arrow 1a in the annotated Figure 2 above, nothing would happen.

221. In sum, in my opinion, Fowlow teaches in response to receiving the request [*e.g.*, request from the client 202, Figure 2’s arrow 1a]: compiling the application [*e.g.*, a particular applet’s execution code] into a compiled form [*e.g.*, application code “may be compiled before execution”].

*d. [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and*

222. As an initial matter, within the context of the ’740 Patent, I understand the term “transforming” to include operations that modify code, such as verification, optimization, and compression. EX1001 at 5:18-23 (“Transformers 56 are programs that take in intermediate code and put out intermediate code. Transformers 56 are

generally used for verification and optimization. Other transformers might include compressors that identify portions of code that can be replaced with smaller equivalents.”), 5:29-53 (“The verifier’s role can therefore be summarized as determining where unsafe code exists and possible altering the offending code to render it harmless. ... Verifiers 48 are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure.”).

223. I understand the term “transforming operation,” used in U.S. Patent Nos. 8,856,779 and 9,325,740 (which I understand are continuations of the herein challenged ’740 Patent), was previously construed to mean “operation that modifies code.” EX1025 (*Trend Micro Markman Order*) at 43-45. In my opinion, this construction of “transforming operation” is consistent with my understanding of the plain and ordinary meaning of the term “transforming” as the term is used in claim 11 of this ’740 Patent. Thus, in my opinion, the plain and ordinary meaning of “transforming” includes operations that modify code.

224. Additionally, it is unclear what claim 11 of the ’740 patent means when claiming “wherein transforming comprises execution....” Each time the specification refers to “execution” or “executing,” it is referring to a client executing the received application on the client (*e.g.*, web browser). *See, e.g.*, EX1001 at 1:37-38 (“In order to run a program, the client machine uses an interpreter to execute the

compiled byte-code.”), 2:6-8 (“There is a need for a scalable distributed system architecture that provides a mechanism for **client computers** to request and execute applets in a safe manner”), 2:17-19 (“an applet server architecture is taught which allows **client computers** to request and execute applets”), 6:30-31 (“The client 12 is the **target computer** on which the user wishes to execute an applet.”), 6:43-46 (“Client computers 12 and 14 requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the **client machine**.”).<sup>2</sup>

225. In my opinion, a POSA would have understood that in the ’740 Patent, execution takes place on the client endpoint, as consistent with these disclosures, *i.e.*, a client executes applications. Conversely, it makes little sense to execute an applet or application on a server (*i.e.*, outside the client) because, for example, the results of execution must be locally displayed on the client web browser, cookies have to be stored locally on the client disk drive, and the browser has to interact with the client directly (*e.g.*, mouse and keyboard).

226. One such specification reference describes *preventing* unsafe code from being executed, modifying code to make it safe, or flagging the risk of executing the

---

<sup>2</sup> In this declaration, all bold and underlined text is my own emphasis, unless otherwise noted.

unsafe code at the client computer. *Id.* at 5:34-40 (“Once a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe.”).

227. In my opinion, a POSA would not have understood preventing unsafe code from being executed, modifying code to be safe, and flagging unsafe code to require “execution,” because all such actions can take place on a binary executable file without running it. Moreover, the ’740 Patent lacks any other description of how transforming can comprise execution, and thus this disclosure seems to be the most relevant.

228. Finally, element [11d] recites “sending the transformed form of the application to the client computer.” I understand this means the step of “transforming compris[ing] execution” to create the “transformed form,” as recited in element [11c], must occur at the server computer before the transformed form is sent to the client computer.

229. Thus, in my opinion, in light of the specification of the ’740 Patent, to the extent a POSA could infer a meaning of “execution” in the context of Claim 11, it would include encasing unsafe code that prevents execution at the client, modifying unsafe code to make it safe, and/or flagging to a user that a code fragment is unsafe; and otherwise, actual “execution” takes place on the client side. *Id.*

Various methods to perform these steps were well-known as of March 1998. *See* §§IV.D and IV.E above.

230. Fowlow describes application transforming functionality within the ORB architecture. For example, Fowlow describes filter 40 of ORB 11 and states that this filter “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.

231. In my opinion, this disclosure suggests ORB-based transforming of application code—including compression, encryption, tracing, and debugging—of applets at the ORB. *Id.* at 5:59-6:8. Fowlow explicitly notes that “this object request broker, or ORB, may be implemented as described above with reference to FIGS. 1a, 1b, and 1c.” *Id.* at 11:21-35.

232. In addition, Fowlow teaches that applets are made up of “byte codes representing classes that are **executed by a remote interpreter**, such as Java classes ... [that] may also be **compiled before execution**...” EX1009 (Fowlow) at 10:50-56. In my opinion, a remote interpreter executing and compiling class files teaches processing the applets remotely (*i.e.*, at the ORB). *See id.* at 1:48-50 (“A Java applet contains compiled code that is portable and may be executed on any computer running the Java interpreter.”).

233. In my opinion, each of these functions undertaken by the ORB—compressing, encrypting, tracing, debugging, and compilation—are examples of the ORB transforming applications. *See* §§IV.D and IV.E above.

234. In my opinion, filter 40’s tracing and debugging functionality suggests the claimed transforming during execution. Both tracing and debugging suggest programs that understand the input code, which would have involved either stepping through a program, executing each stage of the program, or executing the whole program and logging or tracing the output. *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” and “tracing” is a technique to “analyze source code and running programs, looking for bottlenecks”). This understanding is also consistent with the disclosure of the ’740 Patent. EX1001 at 5:34-40.

235. Debugging also involve analyzing input code and determining whether that code is genuine and safe. Debugging would involve ensuring the input code would not result in memory leaks, or other concerns that would make the client device vulnerable to failures or attacks. The ’740 Patent defines transforming in a similar way, describing “verifier 48” as “a type of transformer that is able to analyze input code and determine areas that might not be safe.” EX1001 at 5:29-31. “Once a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents

this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment.” *Id.* at 5:34-40.

236. In my opinion, the tracing and debugging functionality of Fowlow’s filter 40 teach transforming the compiled application during execution. *See* §IV.E above.

237. Regarding compression, Fowlow explicitly teaches that filter 40 transforms applications with compression. 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 applied to communications to and from an object.”).

238. Fowlow states that “this object request broker, or ORB, [of Figure 2] may be implemented as described above with reference to FIGS. 1a, 1b, and 1c.” EX1009 (Fowlow) at 11:21-35. The reference to Figs 1a, 1b, and 1c include the disclosure of filter 40. *Id.* at 5:7-13 (“A distributed object system 10 typically includes an Object Request Broker (ORB) 11 as is symbolically illustrated in FIG. 1a. ORB 11 provides all of the location and transport mechanisms and facilities necessary to deliver a call from a client to a servant (target object) and to return a response to the client, as will be discussed below with reference to FIG. 1b.”), 5:57-6:9.

239. In my opinion, to take advantage of these beneficial functions and provide efficient, secure, and reliable applications to the client device, it would have been obvious, and wise, to include these features in the ORB of Figure 2. *Id.* at 6:5-8, 6:43-47.

240. Notably, any transforming actions on the ORB take place before the application is sent to the client. *Id.* at 12:6-9 (“Arrow (5) shows how this class code corresponding to the class name desired by the Java client is finally returned via the ORB 206 to the Java client 202.”).

241. In my opinion, any transforming actions would occur at the ORB after the application has been compiled, but before the application was sent to the client, because it would otherwise be impractical and wasteful to transform whole applications before compilation—to debug and trace through the application in its entirety as opposed to individual files. This would be beneficial, for example, to catch bugs and other issues stemming from the compilation, such as locating and remedying unresolved dependencies. *Id.* at 4:3-15 (“In another aspect, once 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. These unresolved references may take the form of classes used by loaded execution code that are not yet defined or loaded. If there are unresolved references, the client requests additional applet execution code

corresponding to these unresolved references from the first applet server found.”),  
13:43-62.

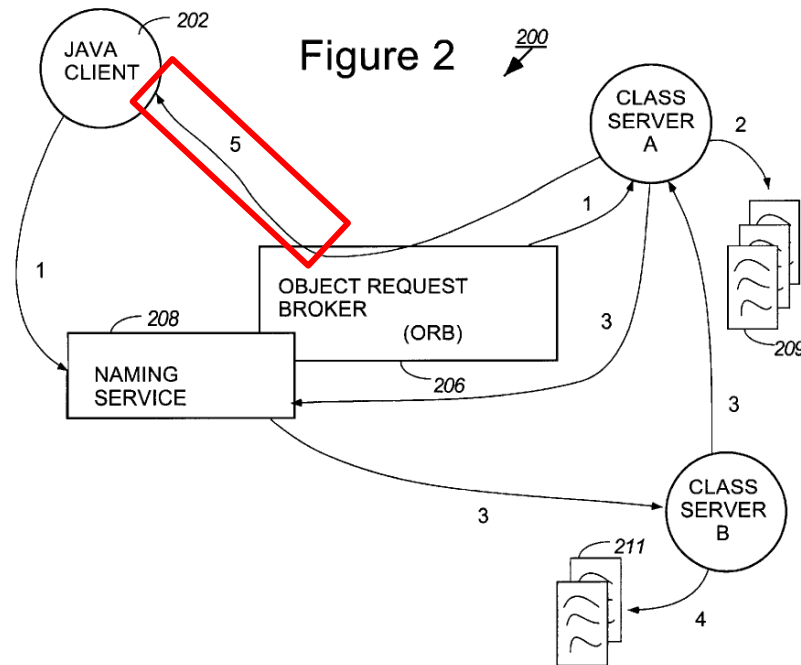
242. Finally, a POSA would have understood that it would also be impractical and wasteful to send untransformed (*e.g.*, non-debugged and uncompressed) code to the client, then return that same code to the ORB for transforming (*e.g.*, debugging and compression) by the filter 40. This would waste network resources, memory, and processing, by sending “useless” untransformed data—only then to subsequently send the actual transformed data.

243. In sum, in my opinion, Fowlow teaches transforming [*e.g.*, filter 40’s “compression, encryption, tracing, or debugging”] the compiled application [*e.g.*, application code “may be compiled before execution”] into a transformed form [*e.g.*, debugged, traced, all dependencies resolved, and compressed] of the compiled form of the application, wherein transforming comprises execution [*e.g.*, filter 40’s debugging and tracing, and the dependency resolution process] and compression [*e.g.*, filter 40’s compression] of the compiled form.

*e. [11d] sending the transformed form of the application to the client computer.*

244. In the end of Fowlow’s steps, “class code corresponding to the class name desired by the Java client is finally returned [to the client] via the ORB.”  
EX1009 (Fowlow) at 12:6-9. As seen in annotated Figure 2 below, “[a]rrow (5)

shows how this class code corresponding to the class name desired by the Java client is finally returned via the ORB 206 to the Java client 202.” EX1009 (Fowlow) at 11:63-12:9.



*Id.* at Fig. 2 (annotated).

245. While I do not express an opinion regarding the order of the steps, and reserve the right to address the order of steps if the Patent Owner addresses order of steps, it is my opinion that the application or applications returned to the client device would be the transformed form of the compiled form of the application. Specifically, as detailed in §VIII.A.1.d above, it would be impractical and wasteful to send untransformed (*e.g.*, non-debugged and uncompressed) code to the client, then return

that same code to the ORB for transforming (*e.g.*, debugging and compression) by the filter 40, before finally sending the transformed form to the client.

246. In sum, in my opinion, Fowlow teaches sending [*e.g.*, returned via the ORB, Figure 2's arrow 5] the transformed form of the application [*e.g.*, debugged, traced, all dependencies resolved, and compressed] to the client computer [*e.g.*, client 202].

247. In my opinion, Fowlow discloses, or at least teaches, each limitation of claim 11 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow.

## **2. Dependent Claims 12-19**

- a. [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.*

248. Fowlow confirms it was known that “the location of these Java classes within the computer network are identified using a Universal Resource Locator (URL) address.” EX1009 (Fowlow) at 2:13-32. The Java classes identified by a URL make up an applet. *Id.* at 10:50-54 (“an applet may be a collection of byte codes representing classes..., such as Java classes”). In addition, Fowlow explains that “HTML page data that a user may acquire through Netscape contains references to applet classes that may be used to execute small programs in parts of the page. Thus, Netscape would be directed to locate and download the code for the applet to

run in the frame that the page defines. This code is found by reference to a specific URL address that identifies a particular computer.” *Id.* at 2:37-54. *See also* §IV.B above.

249. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein the identified form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes] is specified using a Universal Resource Locator [*e.g.*, URLs accessed in browser such as Netscape].

*b. [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.*

250. As I discussed in §IV.B above, cookies are integral to web technologies. Cookies are in fact special HTTP headers designed to manage arbitrary, persistent state between HTTP clients and servers. Moreover, web clients (browsers) and servers often exchange information about their own properties (*e.g.*, operating system and version running, screen capabilities, and more).

251. Fowlow teaches “the form of the request [for an application or applet] will depend to a large extent upon the nature of the programming language used to create the client.” EX1009 (Fowlow) at 7:11-23, 10:28-34. Relatedly, Fowlow explains that the ORB has an implementation repository that “is used to store information relating to object servers” including “information such as ... any

arguments to the [server] program, and any environment variables to pass to the [server] program, etc.” *Id.* at 6:61-67.

252. A POSA would have understood that such “arguments” and “variables” that are passed to the server can be accomplished via standardized cookies among other forms. *See* EX1014 (Microsoft Computer Dictionary) at 119 (“cookie” definition 2) (“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.”).

253. In my opinion, it would have been obvious to include the information stored on the implementation repository to identify an application and a form of the application on the server. *Id.*; *see also* §IV.B above. In my opinion, this stored information functions as a cookie, which is not otherwise defined or described in the ’740 Patent.

254. For example, the implementation server could store information relating to the programming language used to create the client. *See* EX1009 (Fowlow) at 7:11-23 (“The form of the request will depend to a large extent upon the nature of the programming language used to create the client. By way of example, if the client is written in the C++ language, the request may take the form of a C++ method call 62.”).

255. This information could then be passed to the server as an argument or environment variable to specify which form or language of an applet is needed. *Id.* at 6:61-67 (“For example, implementation repository 50 stores information such as the location of the server program, any arguments to the program, and any environment variables to pass to the program, etc.”). In my opinion, this would achieve one of the goals outlined in Fowlow, that the client “is mercifully unaware of all of the machinations behind the scenes required to find other classes used or needed” to enable the applet to run (*e.g.*, download a form of an applet suitable for running the specific client’s operating system, version, and architecture). *Id.* at 10:31-34.

256. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein the identified form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes] is specified using a cookie [*e.g.*, information stored on the implementation repository].

*c. [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header.*

257. As an initial matter, I note that this claim merely requires that the already identified form of the application be specified using an HTTP header. But HTTP headers were well known, heavily used (*e.g.*, for cookies), and standardized

well before the '740 Patent. *See* §IV.B above; *see also generally* EX1017 (HTTP State Management Mechanism RFC 2109).

258. Fowlow discloses that “a browser [*i.e.*, a client] typically uses the hypertext transfer or ‘http’ protocol” to request data, such as Java classes, stored on servers. EX1009 (Fowlow) at 2:22-25, 7:11-23, 10:49-50. As I describe in §IV.B above, most protocols include headers and HTTP is replete with multiple types of headers (*e.g.*, headers used in requests, responses, cookie based headers, and more). In my opinion, a POSA would have found it obvious to specify a form of an application using an HTTP header, because using HTTP headers to convey information was well known at the time of the '740 Patent. *See* EX1014 (Microsoft Computer Dictionary) at 305.

259. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein the identified form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes] is specified in a HTTP header [*e.g.*, because using the HTTP protocol and headers was “typical”].

*d. [15] The method of claim 11 wherein the identified form of the application is a web page.*

260. Fowlow describes locating and downloading applets onto a client machine, which are made up of “a wide variety of types of execution code.” EX1009 (Fowlow) at Abstract, 10:15-65. These applets include Java code that “can be

embedded in another application such as an HTML document in order to provide interactive, executable content on a **Web page**.” *Id.* at 10:56-62.

261. In my opinion, applets that make up content on a web page and applets that make up an entire web page were both known at the time of the ’740 Patent. *Id.* at 2:33-40 (“Typically, Web pages are described in a hypertext markup language (HTML) that defines how the Web page will appear and perform when downloaded to the user’s computer.”). *See also* §IV.B above.

262. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein the identified form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes or an HTML document] is a web page [*e.g.*, HTML documents].

*e. [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.*

263. As described in §VIII.A.2.d, Fowlow teaches a method of locating and downloading applets that “can be embedded in another application such as an HTML document in order to provide interactive, **executable content on a Web page**.” *Id.* at Abstract, 10:56-62. More specifically, “the HTML page data that a user may acquire through Netscape contains references to **applet classes** that may be used to execute small programs **in parts of the page**. Thus, Netscape would be directed to

locate and download the code for the applet to run in the frame that the page defines.”

*Id.* at 2:37-47, 4:3-15.

264. According to Fowlow, the “applet [to run in the frame that the page defines] may mean in particular a small Java program that can be embedded in another application such as an HTML document in order to provide interactive, executable content on a Web page. ... The presentation of a particular object may be used to run an executable program in a small window within a larger window.” *Id.* at 10:59-11:4. *See also* §IV.B above.

265. In sum, in my opinion, Fowlow teaches the method of claim 11 Wherein the identified form of the application [*e.g.*, the form corresponding to the native language of the client, such as Java classes or an HTML document] is a portion of a web page [*e.g.*, a small Java program embedded in an HTML document].

*f. [17] The method of claim 11 wherein transforming further comprises verification of the compiled form.*

266. The filter 40 in Fowlow’s ORB “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. In my opinion, this filter 40 teaches or suggests verification of the compiled form of the application.

267. Specifically, in my opinion, the encryption and debugging functionalities of filter 40 are a form of verification. Both encryption and debugging

involve analyzing input code and determining whether that code is genuine and safe. Encryption accomplishes this by determining if a requisite level of security is met, and that the code is genuine and cannot be tampered in an unauthorized manner. Encryption is a foundational technology for verifying privacy and authenticity of digital data.

268. Debugging would involve ensuring that the input code would not result in (for example) memory leaks, or other concerns that would make the client device vulnerable to failures 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.” EX1009 (Fowlow) at 6:43-47. In my opinion, this Secure Protocol functions to verify input code and applet data transmitted through the transport layer 38. The ’740 Patent defines verification in a similar way: verification includes “analyz[ing] input code and determin[ing] areas that might not be safe.” EX1001 at 5:29-31. In my opinion, filter 40 and Secure Protocol 42 teach the “verifying” step in claim 1. See §IV.E above.

269. Notably, any transforming operations, including the encryption and debugging operations on the ORB, takes place before the applet is sent to the client. *Id.* at 12:6-9 (“Arrow (5) shows how this class code corresponding to the class name desired by the Java client is finally returned via the ORB 206 to the Java client

202.”). In my opinion, any processing would occur at the ORB while the ORB was returning the applet to the client because it would be the most efficient and logical way to process data the ORB already has: otherwise it would be impractical and wasteful to send unprocessed code to the client, then return that same code to the ORB for processing by the filter 40 or Secure Protocol 42.

270. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein transforming [*e.g.*, debugging, tracing, resolving dependencies, and compressing] further comprises verification [*e.g.*, encryption and debugging] of the compiled form [*e.g.*, application code “may be compiled before execution”].

*g. [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form.*

271. The filter 40 in Fowlow’s ORB “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. In my opinion, this filter 40 teaches or suggests optimizing the compiled form of the application.

272. The ’740 Patent describes “[o]ptimizers 46 [that] analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers 46 take in intermediate code 44 and put out transformed intermediate code 50. The transformed intermediate code 50 is functionally equivalent to the source intermediate code 44 in that they share the same

structure.” EX1001 at 6:1-8. A POSA would have understood that the ’740 patent merely describes standard optimization techniques that, for example, compilers have used long before the patent. *See* §§IV.D-IV.F above.

273. In my opinion, filter 40’s tracing and debugging functionality suggests the claimed optimizing the compiled form of the application. Both tracing and debugging suggest programs that understand the input code; both have been used to analyze programs and source code for the purpose of optimizations.

274. In my opinion, tracing and debugging result in better running code, and both are well-known components and precursors for more specific optimizer functions. For example, before one can optimize a piece of code, it is useful to trace and debug the code to find performance bottlenecks (*e.g.*, inefficient loops, bloated data structures).

275. 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.).

276. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein transforming [*e.g.*, debugging, tracing, resolving dependencies, and compressing] further comprises optimization [*e.g.*, tracing and debugging] of the compiled form [*e.g.*, application code “may be compiled before execution”].

*h. [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response.*

277. As an initial matter, I note that this claim merely requires that the already transformed form of the application be returned to the client computer as an HTTP response. But HTTP requests and responses were well known, heavily used (*e.g.*, GET and POST calls), and standardized well before the '740 Patent. In fact, most Internet protocols (even before HTTP) inherently use a request-response behavior. *See* §IV.B above; *see also generally* EX1016 (HTML 4.0 Specification).

278. As discussed in §VIII.A.2.c above in relation to claim 14, Fowlow teaches using HTTP requests. EX1009 (Fowlow) at 2:22-25, 7:11-23, 10:49-50. In the HTTP protocol (and in fact in most Internet protocols), any data returned to the client is sent in an HTTP response. Such client/server protocols are designed to send a request and receive a response in return. When an application is requested by the

client, Fowlow renders obvious sending the transformed application to the client computer in an HTTP response. *See also* §§IV.A and IV.B above.

279. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein the transformed form of the application [*e.g.*, debugged, traced, all dependencies resolved, and compressed] is sent to the client computer in an HTTP response [*e.g.*, because browsers typically used the standardized HTTP protocol, using standard request-response protocols].

**B. Ground 2: Claims 11-21 are Obvious in view of Fowlow in view of Kimera**

280. Independent claim 11 and dependent claims 12-21 are obvious over Fowlow combined with Kimera. Below I explain how each element of claims 11-21 are disclosed, taught, and/or suggested by Fowlow in combination with Kimera.

**1. Motivation to Combine Fowlow and Kimera**

281. 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.

282. 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).

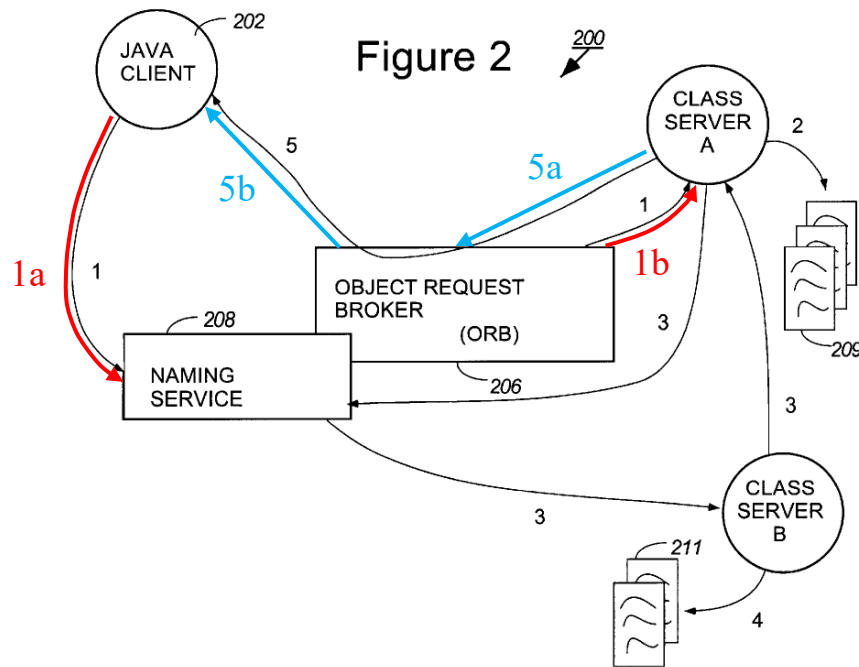
283. Second, both Fowlow and Kimera suggest and explain the benefits of storing software in more locations than just a single server. Fowlow describes “[o]ne advantage to using an ORB is that these class files and their associated class server may be located anywhere within the distributed object system 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.

284. 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.”).

285. 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.

286. 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:

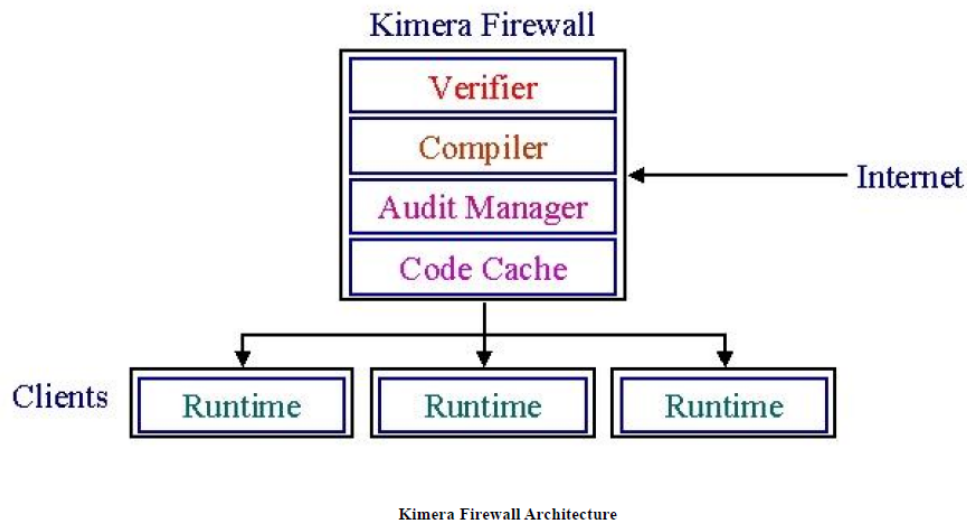


EX1009 (Fowlow) at Fig. 2 (annotated).

287. 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,

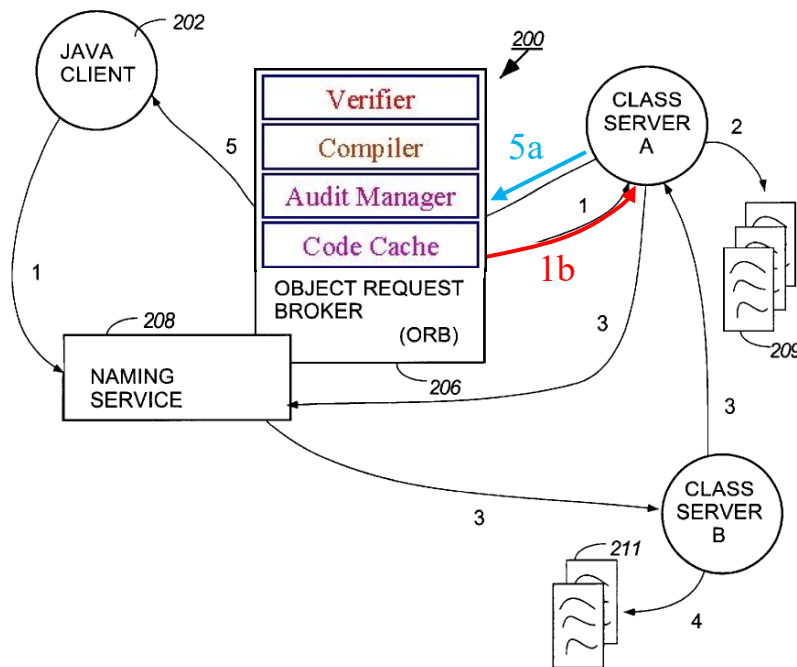
and verified) at the ORB before being sent to the client (arrow 5b). *Id.*; *see also Id.* at 6:5-8. If the base class that is returned to the client is found to have any missing dependencies, then the *entire loop* must be completed again to locate and download those files. *Id.* at Abstract, 4:3-15,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

288. 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:



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

289. 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.

290. 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.

291. 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

292. 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.

293. 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.

294. 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 11**

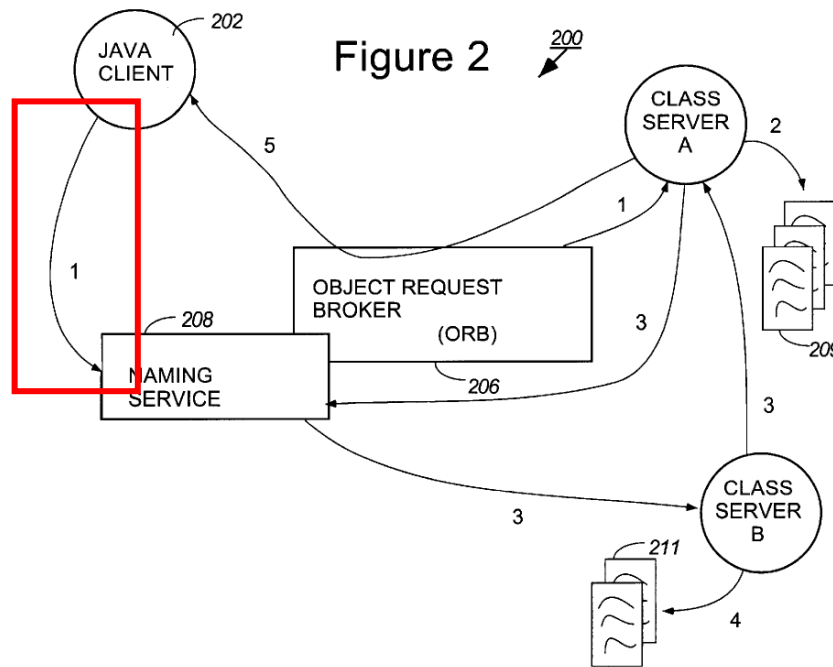
- a. [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising:*

295. 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 11 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.A.1.a above. *See also* EX1009 (Fowlow) at Abstract, 3:27-42, 5:34-36, 7:11-52, 10:15-34, Fig. 2.

- b. [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and*

296. A POSA would find Fowlow renders this element obvious, as explained in §VIII.A.1.b above. *See also* EX1009 (Fowlow) at 5:59-65, 6:5-11, 10:45-56, 11:21-35; EX1001 at 3:21-24.

297. Specifically, 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. For every request, a POSA would have understood the ORB as receiving a request from the client for an applet.



*Id.* at Fig. 2 (annotated).

298. Each request in Fowlow also identifies the form of the desired application. For example, Fowlow explains “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.

299. 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.”).

300. The language of the native machine code could be understood from the request itself, such that the specific form of the application 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.

301. 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 (“Compilation from Java bytecode into native machine code is another service that is best performed at a central location. ... Caching the results of recent compilations in the central compilation server can be used to further reduce the latency of binary translation.”).

302. In sum, in my opinion, the Fowlow-Kimera combination teaches at the server computer [*e.g.*, Fowlow’s ORB 206, Kimera’s centralized server], receiving a request from the client computer [*e.g.*, Fowlow’s client 202], the request identifying an application [*e.g.*, Fowlow’s “particular applet execution code”] and identifying a form of the application [*e.g.*, the form corresponding to the native language of the client, such as Fowlow’s Java classes or Kimera’s “native machine code”].

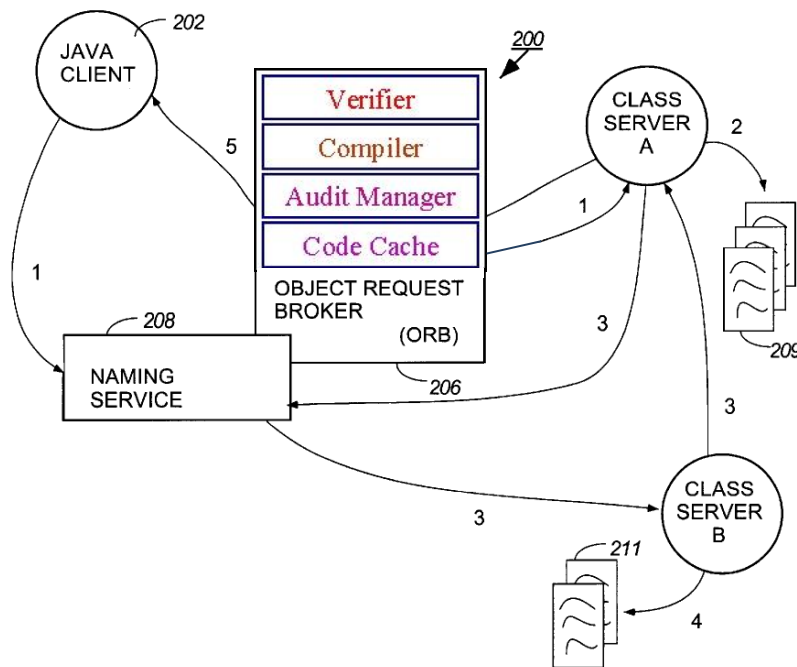
c. *[11b] in response to receiving the request: compiling the application into a compiled form;*

303. A POSA would find that Fowlow renders this element obvious, as explained in §VIII.A.1.c above. *See also* EX1009 (Fowlow) at 5:59-65, 6:5-11, 10:50-56, 11:21-35.

304. Like Fowlow, Kimera 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.* *See also* §IV.D above.

305. 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.B.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.

306. Kimera’s Firewall/server includes a compiler that compiles “Java bytecode into native machine code” and a code cache which “cach[es] the results of recent compilations in the central compilation server.” *Id.* In my opinion, a POSA would have understood the “recent compilations” addressed in Kimera would be compilations processed in response to a client request for a particular applet. This would make sense because (1) it would be highly inefficient to consider pre-compiling all possible applets from the Internet on Kimera’s firewall and (2)

caching, as in Kimera, is a technique for storing recently processed items on-the-fly, or in demand due to recent requests. *See also* §IV.C above.

307. 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 identified form (*e.g.*, Java programs or native machine code). EX1008 (Kimera) at 2; EX1009 (Fowlow) at 11:21-25.

308. The applicability of Kimera's Firewall to Fowlow's ORB is even more evident when the combination is viewed next to the '740 Patent. For example, the language and figures describing Kimera's Firewall/server and the '740 Patent's applet server computer 10 are fairly similar, as seen in the comparison table and side-by-side annotated figures below:

| Kimera                                                                                                                                                                                                                                                                                                        | '740 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 3:40-45, 4:40-43. |

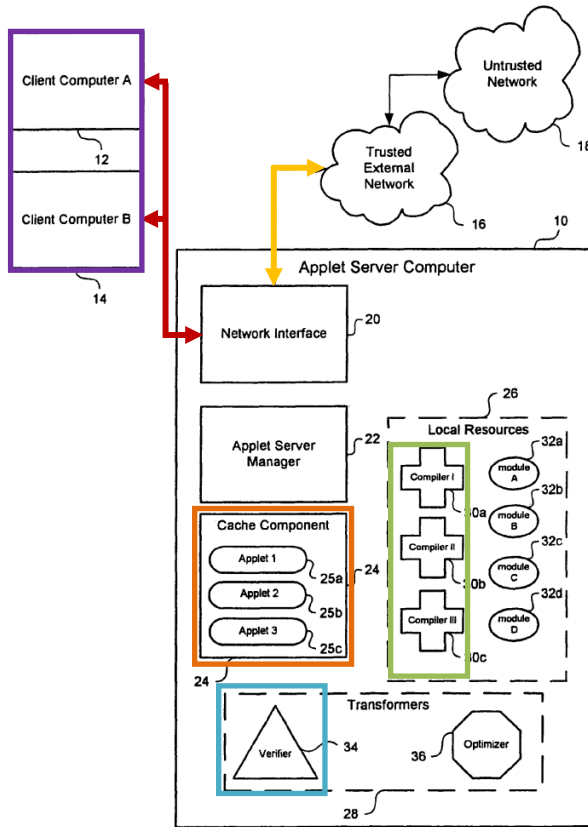
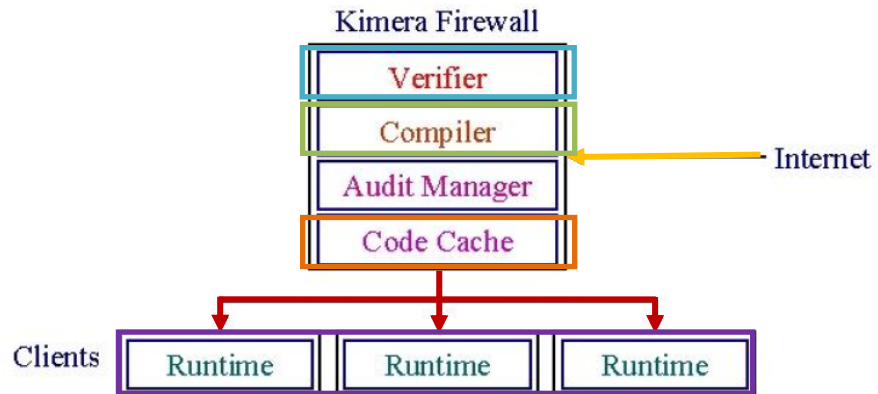


**Fig. 1**



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                                  | '740 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                                                         |

309. The similarities between Kimera and the '740 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 '740 Patent. EX1001 at Abstract, 3:53-55, 4:58-67, Fig. 1.

310. In sum, in my opinion, the Fowlow-Kimera combination teaches in response to receiving the request [*e.g.*, Fowlow’s request from the client 202, Fig. 2 (annotated), arrow 1a]: compiling the application [*e.g.*, Fowlow’s particular applet

execution code, Kimera's Java bytecode] into a compiled form [e.g., Fowlow's application code "may be compiled before execution," Kimera's compiling into "native machine code"].

*d. [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and*

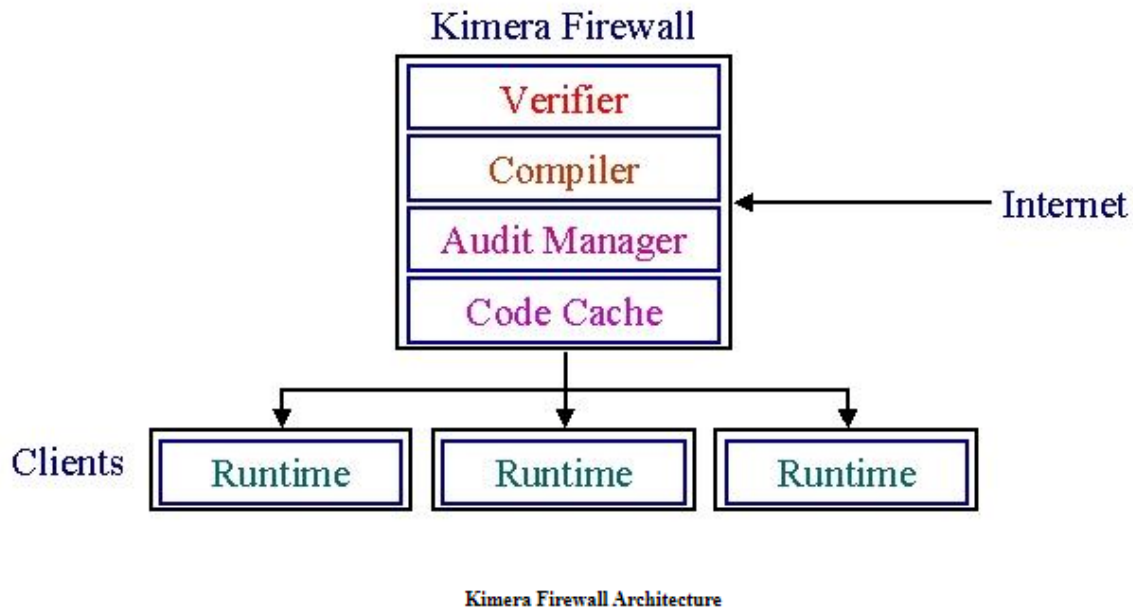
311. A POSA would find Fowlow renders this element obvious, as explained in §VIII.A.1.d above. *See also* EX1009 (Fowlow) at 5:59-65, 6:5-11, 10:50-56, 11:21-35.

312. Additionally, for the reasons discussed in §VIII.A.1.d above, to the extent a POSA could infer a meaning of "execution" in the context of Claim 11, in my opinion that meaning would include encasing unsafe code that prevents execution at the client, modifying unsafe code to make it safe, and/or flagging to a user that a code fragment is unsafe; and actual execution takes place on the client. EX1001 at 5:34-40. Various methods to perform these steps were well-known as of March 1998. *See* §§IV.D and IV.E above.

313. Kimera includes additional disclosure of transforming applications at the server. More specifically, Kimera discloses a "way of structuring Java systems, in which verification and compilation are performed in centralized servers." EX1008 (Kimera) at 1-2.

314. This Firewall/server-side transforming actions allows processing-specific hardware and software, such as verifiers, compilers, and memory resources, to be moved from client devices to the Kimera Firewall/server. For example, Kimera's Firewall/server includes a compiler that compiles "Java bytecode into native machine code" and a code cache which "cach[es] the results of recent compilations in the central compilation server." *Id.*

315. In my opinion, a POSA would have understood the "recent compilations" addressed in Kimera would be compilations processed in response to a client request for a particular applet. This would make sense because (1) it would be highly inefficient to consider pre-compiling all possible applets from the Internet on Kimera's firewall and (2) caching, as in Kimera, is a technique for storing recently processed items on-the-fly, or in demand due to recent requests. *See also* §IV.C above.



EX1008 (Kimera) at 1.

316. Kimera’s Firewall/server also includes a “verification service” that “performs checking of incoming code....” *Id.* In my opinion, in order to take advantage of Kimera, a client would request code (such as the Java applet class files described in Fowlow (*see* EX1009 (Fowlow) at Abstract (“A client enabled to load and run Java applets in a distributed object computing system retrieves the needed Java classes in a location-independent manner...”), 3:32-36, 13:1-3)) located on the Internet, receive the code into the ORB as modified by the Kimera Firewall/server, transform the code with the centralized compiler and verifier, and then transmit the compiled and verified code to the client. *Id.* at 1, 2.

317. In my opinion, a POSA would have understood Kimera’s verification service would have likely executed the code while checking incoming code. In my

opinion, execution of the code would have been obvious to ensure the recently compiled code functions “without introducing security vulnerabilities.” *Id.* at 2. A POSA would have understood that in order to verify that programs are safe, often there is no other choice but to run the programs themselves, with debugging and/or tracing safeguarding functionality enabled to catch and prevent bugs right before they happen (*e.g.*, memory/pointer safety violations). *See also* §§IV.E and IV.F above.

318. More specifically, as programs are executed, a verification service like Kimera’s can verify the safety of the applications and check “for errors such as type mismatches, overflows and underflows, dangling or illegal pointer references, and memory-use inconsistencies.” EX1014 (Microsoft Computer Dictionary) at 179-180 (“error control”). This mirrors the ’740 Patent, which states “[o]nce a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment.” EX1001 at 5:34-40.

319. A POSA would have also understood Kimera’s auditing service to execute compiled applications, which could have been used in a way to trace and log the output of the execution. EX1014 (Microsoft Computer Dictionary) at 36 (“audit

trail”). In my opinion, between verifying and auditing the application code, Fowlow-Kimera teaches transforming the compiled form of applications while executing the application code.

320. Additionally, Kimera recognizes the need to conserve resources in environments that have limited capabilities. Kimera complains that “[t]he resource requirements for a Java client are consequently very high” and this “makes Java unsuitable for small mobile devices.” EX1008 (Kimera) at 1. Kimera’s “approach to Java system structure is to break the trusted computing base into small, secure components and locate them in security firewalls.” *Id.* Therefore “[s]uch a design enables timely security upgrades, especially for small, mobile clients....” *Id.* at 2.

321. A POSA would have understood Kimera to at least suggest the usefulness of compression, which is known to reduce data sizes to better fit in smaller storage/memory components, as well as reduce network transmission costs. *See* §IV.E above; *see also* EX1014 (Microsoft Computer Dictionary) at 107 (definition 2) (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.”).

322. Kimera also discusses the dangers of high communication overhead between client a server. EX1008 (Kimera) at 2 (“the communication overhead between the client and the compilation server would dominate any potential

benefits”). In my opinion, it would have been obvious to include compression on all applet files sent to the client device to reduce network bandwidth use and increase download speeds to users.

323. A POSA would have recognized that the transforming operations would likely occur after the application had been compiled, but before the application was sent to the client computer. In particular, a POSA would have understood that it would be impractical and wasteful to transform applications before compilation, as the execution and compression applied in this step would be applied to the application in its entirety as opposed to individual files. Executing or compressing individual files before compilation would making the compilation step difficult as some files may be compressed while others are not. In addition, a POSA would have understood that compressing before compilation could lead to inefficient compression.

324. Worse, compilation operates on well-structured data (*e.g.*, plain text source code, high level assembler instructions, machine instructions), as described in §§IV.C-IV.F above. Compression generally changes low-entropy inputs into high-entropy binary outputs. Compressing such intermediate data would change the very format of data being processed, causing that intermediate compiler stage to fail. Thus, in my opinion, the transforming operations occur after compilation.

325. Finally, a POSA would have understood that it would also be impractical and wasteful to send untransformed (*e.g.*, unverified, pre-audited, and uncompressed) code to the client, then return that same code to the centralized server for transforming (*e.g.*, verification, auditing, and compression). This would waste network resources, memory, and processing, by sending “useless” untransformed data—only then to subsequently send the actual transformed data.

326. In my opinion, a POSA would have been motivated to combine Fowlow and Kimera for the reasons stated in §VIII.B.1 above. In addition, Kimera explains that there are several disadvantages resulting from relying on client computers for processing incoming data. EX1008 (Kimera) at 1. These disadvantages include a higher risk for security breaches and “very high” client system requirements, making client devices more expensive. *Id.* Thus, a POSA would have been motivated to combine Fowlow and Kimera in order to simplify client devices to take advantage of Kimera’s articulated benefits; moreover, a POSA would have achieved this by modifying or supplementing Fowlow’s ORB with the more comprehensive Kimera Firewall/server. The resulting system would be faster, cheaper, more efficient, and more secure. Additional motivations to combine these references are discussed in §VIII.B.1 above.

327. In sum, in my opinion, Fowlow combined with Kimera teaches transforming [*e.g.*, Fowlow’s filter 40’s “compression, encryption, tracing, or

debugging,” Kimera’s verification and auditing services, and client resource reduction] the compiled application [e.g., Fowlow’s application code “may be compiled before execution”, Kimera’s compilation service] into a transformed form [e.g., Fowlow’s debugged, traced, all dependencies resolved, and compressed, Kimera’s verified, audited, and compressed software] of the compiled form of the application, wherein transforming comprises execution [e.g., Fowlow’s filter 40’s debugging and tracing, and the dependency resolution process, Kimera’s verification and auditing service] and compression [e.g., Fowlow’s filter 40’s compression, Kimera’s client resource reduction] of the compiled form.

*e. [11d] sending the transformed form of the application to the client computer.*

328. A POSA would find Fowlow renders this element obvious, as explained in §VIII.A.1.e above. *See also* EX1009 (Fowlow) at 12:6-9, Fig. 2.

329. In sum, in my opinion, Fowlow-Kimera teaches sending [e.g., returned via the ORB as shown in Fowlow’s Figure 2’s arrow 5, Kimera’s arrows from Firewall to Clients] the transformed form of the application [e.g., Fowlow’s debugged, traced, all dependencies resolved, and compressed, Kimera’s verified and audited software for clients with reduced resources] to the client computer [e.g., Fowlow’s client 202, Kimera’s Clients].

330. In my opinion, the Fowlow-Kimera combination discloses, or at least teaches, each limitation of claim 11 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow and Kimera.

**3. Dependent Claims 12-21**

*a. [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.*

331. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.a above. *See also* EX1009 (Fowlow) at 2:13-32, 2:37-54.

*b. [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.*

332. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.b above. *See also* EX1009 (Fowlow) at 6:61-67, 7:11-23, 10:28-34.

*c. [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header.*

333. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.c above. *See also* EX1009 (Fowlow) at 2:22-25, 7:11-23, 10:28-34, 10:49-50.

*d. [15] The method of claim 11 wherein the identified form of the application is a web page.*

334. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.d above. *See also* EX1009 (Fowlow) at Abstract, 2:34-4010:15-65.

*e. [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.*

335. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.e above. *See also* EX1009 (Fowlow) at 2:37-47, 4:3-15.

*f. [17] The method of claim 11 wherein transforming further comprises verification of the compiled form.*

336. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.f above. *See also* EX1009 (Fowlow) at 6:5-8, 6:43-47, 12:6-9; EX1001 at 5:29-31; §IV.E above.

337. Kimera includes additional disclosure of transforming applications comprising verification of the compiled form of the application. More specifically, Kimera discloses a “way of structuring Java systems, in which verification and compilation are performed in centralized servers.” EX1008 (Kimera) at 1-2.

338. Verification on Kimera’s Firewall/server occurs through a “verification service” that “performs checking of incoming code....” *Id.* In my opinion, in order to take advantage of Kimera, a client would request code (such as the Java applet class files described in Fowlow (*see* EX1009 (Fowlow) at Abstract (“A client enabled to load and run Java applets in a distributed object computing system retrieves the needed Java classes in a location-independent manner....”), 3:32-36, 13:1-3)) located on the Internet, receive the code into the ORB as modified by the

Kimera Firewall/server, process the code with the centralized compiler and verifier, and then transmit the compiled and verified code to the client. *Id.* at 1, 2.

339. In addition to the verification station checking incoming code, Kimera can also apply “tamper-proof signatures, using a cryptographically strong hash function, such as MD5, ... to ensure the safety of the code after it leaves the verifier.” *Id.* at 2. In my opinion, both (1) “perform[ing] checking of incoming code for typesafety and conformance to interfaces” and (2) applying “tamper-proof signatures” to verified code before it is transmitted to the client confirms that Kimera’s verification service teaches verifying the applets before they are sent to the client.

340. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein transforming [*e.g.*, Fowlow’s debugging, tracing, resolving dependencies, and compressing, Kimera’s verifying, auditing, and compressing software] further comprises verification [*e.g.*, Fowlow’s encryption and debugging, Kimera’s “verification service”] of the compiled form [*e.g.*, Fowlow’s application code “may be compiled before execution”, Kimera’s compilation service].

*g. [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form.*

341. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.g above. *See also* EX1009 (Fowlow) at 6:5-8; EX1001 at 6:1-8; §§IV.D and IV.E above.

342. The '740 Patent describes “[o]ptimizers 46 [that] analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers 46 take in intermediate code 44 and put out transformed intermediate code 50. The transformed intermediate code 50 is functionally equivalent to the source intermediate code 44 in that they share the same structure.” EX1001 at 6:1-8. A POSA would have understood that the '740 patent merely describes standard optimization techniques that, for example, compilers have used long before the patent. *See* §§IV.D-IV.F above. As discussed in §IV.D above, a POSA would have understood optimization to encompass any process of producing more efficient programs through selection and design of data structures, algorithms, and instruction sequences, including through using a compiler to produce more efficient executable code. *See* EX1014 (Microsoft Computer Dictionary) at 343 (“optimization” definition).

343. Next, Kimera details that centralization “enables the same services [e.g., verification, access control, auditing, and compilation services] to be

performed **faster**, cheaper, and without introducing security vulnerabilities.” EX1008 (Kimera) at 1. One benefit of centralization is that it “makes a sizable reduction in computation and memory requirements of the end-points” to the point where “Java enabled network computers [clients] could [] be constructed from cheaper components.” *Id.* at 2. This process of optimizing applets that are ultimately sent to the client devices would be useful because the client may not otherwise be able to run the downloaded applet on its “reduced computation and memory requirement[.]” components. *Id.*

344. The optimized code would be functionally equivalent to the original code but would be optimized to ensure that clients with “reduced computation and memory requirements” could still run the received software. This would be especially important for clients that already had limited resources, such as “small, mobile clients where the Java engine may be hard coded in ROM,” before centralization allowed for even further reduction to the client requirements. *Id.*

345. Moreover, in Kimera, applets are compiled and cached on its centralized Firewall/server. This alone saves a lot of network bandwidth and latency, and hence optimizes the retrieval of applets before they are being sent to the client browsers. *See also* §IV.C above; EX1018 (Silberschatz) at 43-44. A POSA would have understood that reducing latency in the software-acquisition process would be a form of optimization. *See* §§IV.D and IV.E above (describing

optimization as “the process of producing more efficient (smaller or faster) programs through selection and design of data structures, algorithms, and instruction sequences” and the “process of a compiler or assembler in producing efficient executable code”).

346. In sum, in my opinion, Fowlow teaches the method of claim 11 wherein transforming [e.g., Fowlow’s debugging, tracing, resolving dependencies, and compressing, Kimera’s verifying, auditing, and compressing software] further comprises optimization [e.g., Fowlow’s tracing and debugging, Kimera’s faster services and reduced client requirements] of the compiled form [e.g., Fowlow’s application code “may be compiled before execution”, Kimera’s compilation service].

*h. [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response.*

347. A POSA would find Fowlow renders this claim obvious, as explained in §VIII.A.2.h above. *See also* EX1009 (Fowlow) at 2:22-25, 7:11-23, 10:49-50.

*i. [20pre] The method of claim 11, further comprising:*

*i. [20a] storing a plurality of available forms of the application;*

348. As an initial matter, storing data in a cache is exactly how caches operate. *See* §IV.C above.

349. As discussed in §VIII.B.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 more than one recent native machine code compilations in the central compilation server is the same as “storing a plurality of available forms of the application” in the code cache. *Id.* at 1, 2.

350. Storing more than one form of the application would have been obvious in view of Kimera’s description of the purpose of the code cache memory, which is used to “reduce the latency of binary translation.” *Id.* at 2. A POSA would have understood that to reduce the latency of (multiple) future binary translations, more than one binary translation would be stored in the cache. *Id.* (noting “the results of recent **compilations** [plural]” are cached); *see* also §IV.C above.

351. In sum, in my opinion, the Fowlow-Kimera combination teaches storing [*e.g.*, Kimera’s caching] a plurality of available forms [*e.g.*, Kimera’s recent compilations, plural] of the application.

- ii. [20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and

352. As an initial matter, storing and retrieving data in a cache is exactly how caches operate. *See* §IV.C above. A POSA would have readily known that in order to retrieve desired data from cache memory, a determination would first need to be made if the requested data was in the cache.

353. 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.B.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 Kimera's Firewall/server's Code Cache before passing the request on to the broader distributed object system. 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."). Put differently, in my opinion, a POSA would have understood that as the "single point of access to the distributed object system," the modified ORB would have first checked the Firewall/server Code Cache for recent

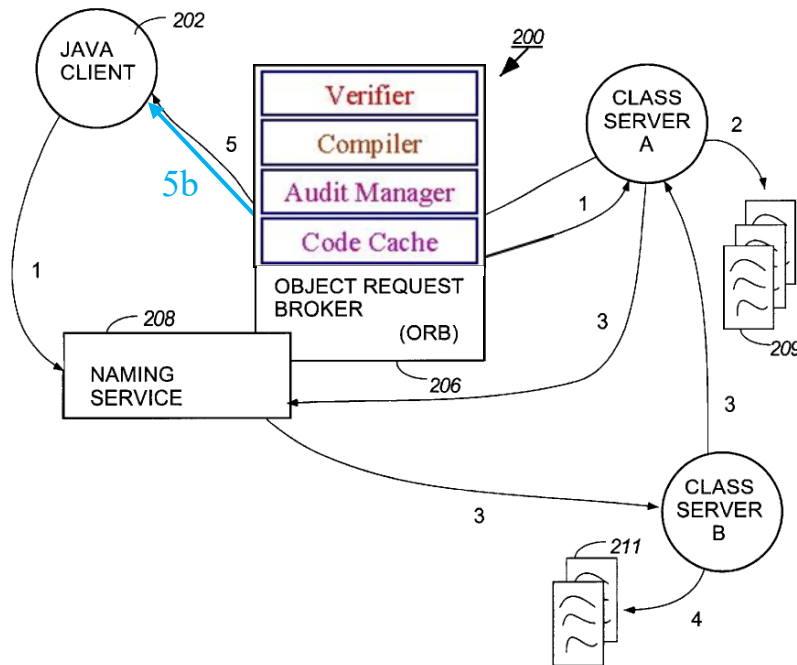
compilations before directing the request to the broader distributed object system.  
EX1009 (Fowlow) at 10:21-27.

354. In sum, in my opinion, the Fowlow-Kimera combination teaches the server, in response to receiving the request, determining if the identified form of the application [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] is one of the stored [*e.g.*, in Kimera's Code Cache] plurality of available forms of the application.

- iii. [20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming.

355. As alluded to in §VIII.B.3.i.ii above, 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. This ORB would decide whether the identified form of the application was stored in the cache memory.

356. If the requested application 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:



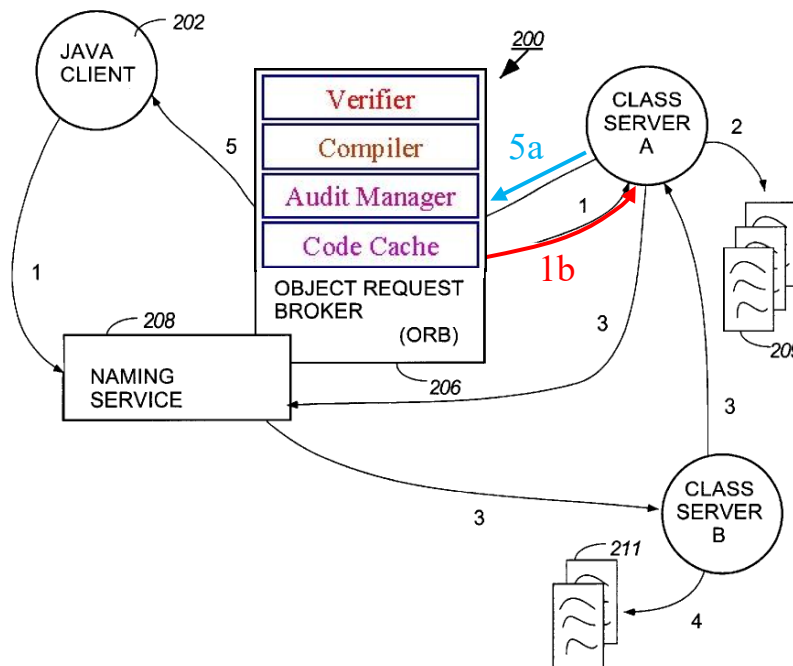
EX1009 (Fowlow) at Fig. 2 (annotated, Fowlow's Figure 2 merged with Kimera firewall Figure 2); EX1008 (Kimera) at 1.

357. In my opinion, when the identified form of the application was stored in Kimera's Code Cache, the application in the identified form could be sent to the client directly without any additional compiling or transforming operations.

358. 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.

359. 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.

360. 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.

361. In sum, in my opinion, the Fowlow-Kimera combination teaches if the identified form of the application [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code] is one of the stored [*e.g.*, in Kimera's Code Cache] plurality of available forms, sending [*e.g.*, arrow 5b] the identified form of the application to the client computer [*e.g.*, client 202] without compiling and transforming [*e.g.*, Fowlow's Java classes, Fowlow's execution code, or Kimera's native machine code, with no additional compilation or transformation].

362. In my opinion, the Fowlow-Kimera combination discloses, or at least teaches, each limitation of claim 20 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow and Kimera.

*j. [21pre] The method of claim 11, further comprising:*

*i. [21a] storing a plurality of available forms of the application;*

363. 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. *See also* EX1008 (Kimera) at 1, 2; §IV.C above.

- ii. [21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and

364. 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. *See also* EX1008 (Kimera) at 1, 2; EX1009 (Fowlow) at 10:21-27; §IV.C above.

- iii. [21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling.

365. 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. *See also* EX1009 (Fowlow) at 10:15-34, Fig. 2; EX1008 (Kimera) at 1, 2; §§IV.A and IV.C above.

366. While claim limitation [20c] recites “sending the identified form of the application to the client computer without compiling *and transforming*,” this limitation only requires sending the application just “without compiling.” In my opinion, removing the transforming operation does not affect the analysis, because one can transmit a previously compiled application exactly the same way, whether the application is also transformed pre- or post-compilation or not; if the application in the identified form was stored in the Code Cache, the application in the identified form could be sent to the client directly without any additional compiling operations.

367. In my opinion, the Fowlow-Kimera combination discloses, or at least teaches, each limitation of claim 21 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Fowlow and Kimera.

**C. Ground 3: Claims 11-12 and 14-21 are Obvious in view of Parthasarathy in view of Kimera**

368. Independent claim 11 and dependent claims 12 and 14-21 are obvious over Parthasarathy combined with Kimera. Below I explain how each element of claims 11-12 and 14-21 are disclosed, taught, and/or suggested by Parthasarathy in combination with Kimera.

**1. Motivation to Combine Parthasarathy and Kimera**

369. 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.

370. 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).

371. 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.

372. 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,” where “verification, access control, auditing, and compilation services” are located. EX1008 (Kimera) at 1.

373. 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”).

374. 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”).

375. 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, in my opinion, 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.

376. 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:

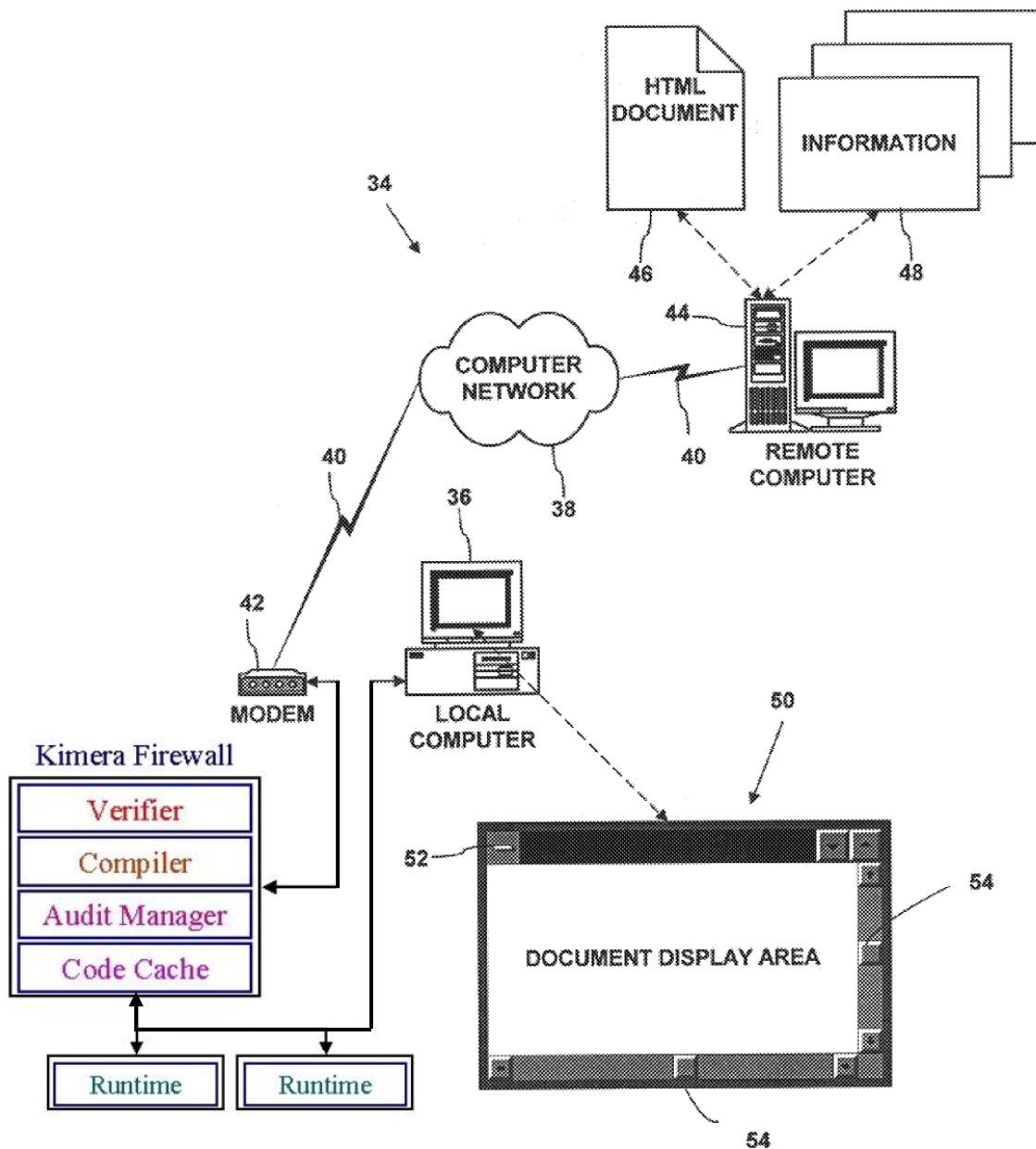


FIG. 2

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.

377. 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.

378. 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.

379. 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.

380. 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.”).

381. 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.

382. 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, IV.C, and IV.D above.

383. 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 11**

- a. [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising:*

384. 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 11 is limiting, it is my opinion that it is taught by Parthasarathy in view of Kimera.

385. 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.

386. 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, having a client computer and a server computer, for managing requests to the server computer. *See* §IV.B above.

387. 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.

388. 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.”). A POSA would have understood 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.

389. In sum, in my opinion, to the extent the preamble is limiting, the Parthasarathy-Kimera combination teaches a method operating on a computer system [*e.g.*, Parthasarathy's network browsing environment], having a client computer [*e.g.*, Parthasarathy's local computer] and a server computer [*e.g.*, Kimera's centralized server or Parthasarathy's remote computer], for managing requests to the server computer.

*b. [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and*

390. The Parthasarathy-Kimera combination teaches at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application. 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.

391. As discussed in §§VIII.C.1 and VIII.C.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.*

392. “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.

393. Thus, “[w]hen a request is made to download a desired software component stored on a remote computer 44,” the request would first be received at the centralized server. *Id.* The request further identifies the “desired software component,” which a POSA would have understood is simply a desired application. *Id.*

394. Parthasarathy discloses that the request for an application also specifies the form of the application 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<sub>1</sub> through URL<sub>m</sub> (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.

395. “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.

396. 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.

397. 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.

398. 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.

399. In sum, in my opinion, the Parthasarathy-Kimera combination teaches at the server computer [*e.g.*, Kimera’s centralized server or Parthasarathy’s remote computer], receiving a request from the client computer [*e.g.*, Parthasarathy’s local computer], the request identifying an application [*e.g.*, Parthasarathy’s “desired software component”] and identifying a form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code].

*c. [11b] in response to receiving the request: compiling the application into a compiled form;*

400. 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* EX1007 (Parthasarathy) at 18:35-57 (“The %opersys% and %cpu% attribute values shown below will specify the operating system and CPU for the desired platform desired software components will be executed on.”); *see also* §IV.D above.

401. Kimera’s Firewall/server also includes a code cache which “cach[es] the results of recent compilations in the central compilation server.” *Id.* In my opinion, a POSA would have understood the “recent compilations” addressed in Kimera would be compilations processed in response to a client request for a particular application. This would make sense because (1) it would be highly inefficient to consider pre-compiling all possible applets from the Internet on Kimera’s firewall and (2) caching, as in Kimera, is a technique for storing recently processed items on-the-fly, or in demand due to recent requests. *See also* §IV.C above.

402. 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 ’740 patent. Specifically, the ’740 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 4:36-47.

403. In sum, in my opinion, the Parthasarathy-Kimera combination teaches in response to receiving the request [Parthasarathy’s “[w]hen a request is made to download a desired software component”]: compiling the application [*e.g.*, Parthasarathy’s software component] into a compiled form [*e.g.*, Kimera’s disclosure of “[c]ompilation from Java bytecode into native machine code”].

*d. [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and*

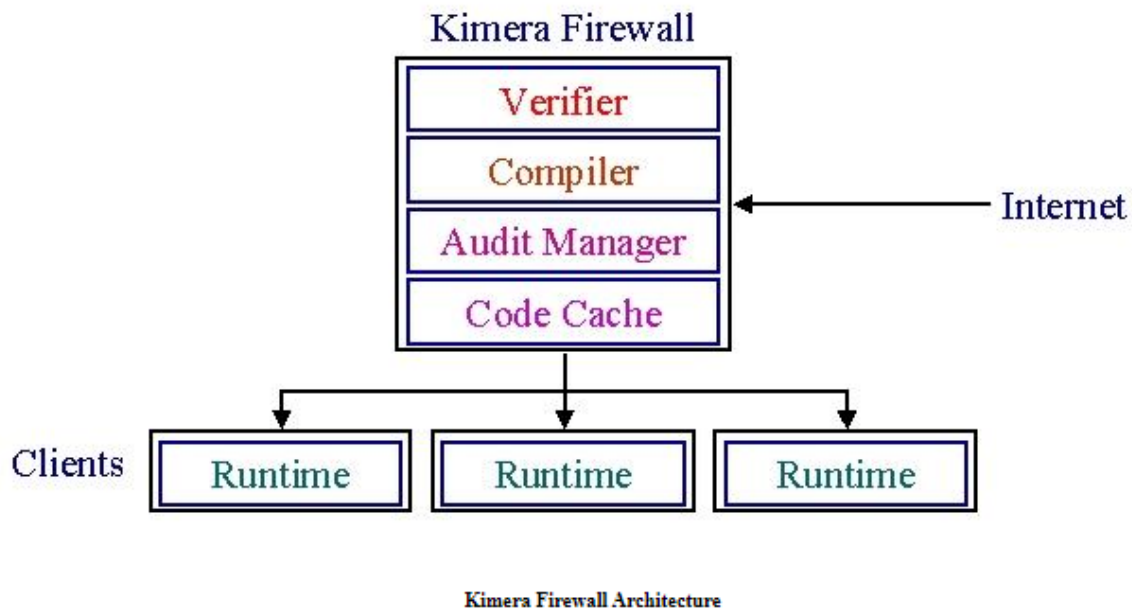
404. As an initial matter, for the reasons discussed in §VIII.A.1.d above, to the extent a POSA could infer a meaning of “execution” in the context of Claim 11, in my opinion that meaning would include encasing unsafe code that prevents execution at the client, modifying unsafe code to make it safe, and/or flagging to a user that a code fragment is unsafe; and otherwise actual “execution” takes place on the client side. EX1001 at 5:34-40. Various methods to perform these steps were well-known as of March 1998. *See* §§IV.D and IV.E above.

405. In Parthasarathy, “[t]he system includes a software component download module for locating computer software components with uniform resource locators (URLs), registering computer software components in a registry database, and displaying computer software components on a local computer. The software component download module also manages other system modules. ... A code verification module 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 2:67-3:12; *see also id.* at 2:65-3:27, 8:19-35, 8:66-9:3. This code verification module, which is managed by the software download control module, demonstrates one way in which the software download control module processes applets.

406. Kimera also describes a verification service at the Firewall/server. EX1008 (Kimera) at 2 (“The verification service performs checking of incoming code for typesafety and conformance to interfaces. Moving this service to a centralized server, and reducing it to a single, small component greatly simplifies security management.”).

407. A POSA would have understood that Parthasarathy’s code verification module be subsumed into Kimera’s verification component on its Firewall. Combining similar processing functionality into Kimera’s centralized server allows

client devices to be simpler and less expensive, because the “compiler, verifier, [and] interpreter” would be moved to the central server. EX1008 (Kimera) at 1, 2. In my opinion, these applet processing functions would be managed more centrally by Parthasarathy’s software download control module on Kimera’s Firewall/server:



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

408. In my opinion, a POSA would have understood Kimera’s verification service would have likely executed the code while checking incoming code. In my opinion, execution of the code would have been obvious to ensure the recently compiled code functions “without introducing security vulnerabilities.” *Id.* at 2. A POSA would have understood that in order to verify that programs are safe, often there is no other choice but to run the programs themselves, with debugging and/or tracing safeguarding functionality enabled to catch and prevent bugs right before

they happen (*e.g.*, memory/pointer safety violations). *See also* §§IV.E and IV.F above.

409. More specifically, as programs are executed, a verification service like Kimera’s can verify the safety of the applications and check “for errors such as type mismatches, overflows and underflows, dangling or illegal pointer references, and memory-use inconsistencies.” EX1014 (Microsoft Computer Dictionary) at 179-180 (“error control”). This mirrors the ’740 Patent, which states “[o]nce a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment.” EX1001 at 5:34-40.

410. A POSA would have also understood Kimera’s auditing service to execute compiled applications, which could have been used in a way to trace and log the output of the execution. EX1014 (Microsoft Computer Dictionary) at 36 (“audit trail”). In my opinion, between verifying and auditing the application code, Parthasarathy-Kimera teaches transforming the compiled form of applications by executing the application code.

411. Additionally, Parthasarathy describes file compression, including Lempel-Ziv compression which was well-known, and specifically notes that “other

compression formats could also be used.” EX1007 (Parthasarathy) at 15:65-16:9. Compression is discussed in Parthasarathy particularly in relation to the cabinet files, which “contain one or more files, all of which are downloaded together in a compressed ‘cabinet.’” *Id.* Transforming applications with compression was, therefore, well-known.

412. Kimera similarly recognizes the need to conserve resources in environments that have limited capabilities. Kimera complains that “[t]he resource requirements for a Java client are consequently very high” and this “makes Java unsuitable for small mobile devices.” EX1008 (Kimera) at 1. Kimera’s “approach to Java system structure is to break the trusted computing base into small, secure components and locate them in security firewalls.” *Id.* Therefore “[s]uch a design enables timely security upgrades, especially for small, mobile clients....” *Id.* at 2. A POSA would have understood Kimera to at least suggest the usefulness of compression, which is known to reduce data sizes to better fit in smaller storage/memory components, as well as reduce network transmission costs. *See also* §IV.E above; *see also* EX1014 (Microsoft Computer Dictionary) at 107 (definition 2) (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.”).

413. Kimera also discusses the dangers of high communication overhead between client a server. EX1008 (Kimera) at 2 (“the communication overhead between the client and the compilation server would dominate any potential benefits”). In my opinion, it would have been obvious to include compression on all applet files sent to the client device to reduce network bandwidth use and increase download speeds to users. *See* EX1014 (Microsoft Computer Dictionary) at 107 (definition 2) (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.”). As Parthasarathy recognizes, any compression format, including Lempel-Ziv, could be used. EX1007 (Parthasarathy) at 15:65-16:9.

414. A POSA would have recognized that the transforming operations would likely occur after the application had been compiled, but before the application was sent to the client computer. In particular, a POSA would have understood that it would be impractical and wasteful to transform applications before compilation, as the execution and compression applied in this step would be applied to the application in its entirety as opposed to individual files. Executing or compressing individual files before compilation would making the compilation step very difficult as some files may be compressed while others are not. In addition, a POSA would have understood that compressing before compilation could lead to inefficient compression.

415. Worse, compilation operates on well-structured data (*e.g.*, plain text source code, high level assembler instructions, machine instructions), as described in §§IV.C-IV.F above. Compression generally changes low-entropy inputs into high-entropy binary outputs. Compressing such intermediate data would change the very format of data being processed, causing that intermediate compiler stage to fail. Thus, in my opinion, the transforming operations occur after compilation.

416. Finally, a POSA would have understood that it would also be impractical and wasteful to send untransformed (*e.g.*, unverified, pre-audited, and uncompressed) code to the client, then return that same code to the centralized server for transforming (*e.g.*, verification, auditing, and compression). This would waste network resources, memory, and processing, by sending “useless” untransformed data—only then to subsequently send the actual transformed data.

417. A POSA would be motivated to incorporate Parthasarathy’s modules, including the software download control module, into the Firewall/server to enable “verification, access control, auditing, and compilation services ... to be performed faster, cheaper, and without introducing security vulnerabilities.” EX1008 (Kimera) at 1. A POSA would have been motivated to combine Parthasarathy and Kimera as described in §VIII.C.1 above. EX1007 (Parthasarathy) at Abstract, 2:67-3:14, 8:13-31, 8:66-9:8, 9:33-42, 23:39-56, Figs. 2, 4; EX1008 (Kimera) at 1, 2.

418. In sum, in my opinion, Parthasarathy in combination with Kimera teaches transforming the compiled application [*e.g.*, Kimera’s disclosure of “[c]ompilation from Java bytecode into native machine code”] into a transformed form [*e.g.*, Kimera’s verified, audited, and compressed software using Parthasarathy’s Lempel-Ziv compression] of the compiled form of the application, wherein transforming comprises execution [*e.g.*, Kimera’s verification and auditing services] and compression [*e.g.*, Parthasarathy’s Lempel-Ziv compression, Kimera’s client resource reduction] of the compiled form.

*e. [11d] sending the transformed form of the application to the client computer.*

419. The local client computer of Parthasarathy can request “to download a desired software component stored on [the] remote computer 44.” EX1007 (Parthasarathy) at 8:19-20. “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 8:26-30. In my opinion, the 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.

420. While I do not express an opinion regarding the order of the steps, and reserve the right to address the order of steps if the Patent Owner addresses order of steps, it is my opinion that the application or applications returned to the client device would be the transformed form of the compiled form of the application. Specifically, as detailed in §VIII.C.2.d above, it would be impractical and wasteful to send untransformed (unverified, pre-audited, and uncompressed) code to the client, then return that same code to the centralized server for transforming (*e.g.*, verification, auditing, and compression), before finally sending the transformed form to the client. This would waste network resources, memory, and processing, by sending “useless” untransformed data—only then to subsequently send the actual transformed data.

421. In sum, in my opinion, Parthasarathy in combination with Kimera teaches sending the transformed form [*e.g.*, Kimera’s verified, audited, and compressed software using Parthasarathy’s Lempel-Ziv compression] of the application [*e.g.*, Parthasarathy’s desired software component] to the client computer [*e.g.*, Parthasarathy’s local computer or Kimera’s clients].

422. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 11 of the ’740 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

### 3. Dependent Claims 12-21

- a. [12] *The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.*

423. In Parthasarathy, “a request is made to download a desired software component stored on a remote computer 44.” EX1007 (Parthasarathy) at 8:19-20. The request “us[es] one of two designators: a URL or a network search path” to identify and locate the computer software component via a control file. *Id.* at 9:9-21. “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 8:30-34. The control file, which is used to locate files, “includes an initialization file that includes one or more *uniform resource locators* pointing to one or more computer software components ... being specified for the operating system or the CPU of the local computer.” *Id.* at cl. 57.

424. As detailed in §VIII.C.2.b above, Parthasarathy’s requests “will **specify the operating system and CPU** for the desired platform desired software components will be executed on.” EX1007 (Parthasarathy) 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.

425. 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.

426. A POSA would have understood that the identified form, *i.e.*, the operating system and CPU architecture, of the desired software component would be communicated to the remote computer in the URL in order to receive the correct application for the client computer. *Id.* at 9:9-21 (“computer software components (e.g., a control file) are automatically located 68 using one of two designators: a URL or a network search path”).

427. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein the identified form of the application [*e.g.*,

Parthasarathy's platform-specific form of the code or Kimera's native machine code] is specified using a Universal Resource Locator [*e.g.*, Parthasarathy's URLs].

*b. [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header.*

428. As an initial matter, I note that this claim merely requires that the already identified form of the application be specified using an HTTP header. But HTTP headers were well known, heavily used (*e.g.*, for cookies), and standardized well before the '740 Patent. *See* §IV.B above; *see also generally* EX1017 (HTTP State Management Mechanism RFC 2109).

429. As I describe in §IV.B above, most protocols include headers and HTTP is replete with multiple types of headers (*e.g.*, headers used in requests, responses, and more).

430. Parthasarathy states “[i]n an illustrative embodiment of the present invention, an Object Store on the network search path is an HTTP server that services requests for a desired control file.” EX1007 (Parthasarathy) at 14:65-67. The desired control file is first located on an Object Store, and once located, “an Object Store will receive an HTTP POST request with data in the format below:

CLSID = {class id}

Version = a, b, c, d

MIMETYPE = mimetype”

*Id.* at 14:67-15:13.

431. “In addition to the HTTP POST data described above, queries to Object Stores may also include **HTTP headers** for ACCEPT (MIME type) and ACCEPT-LANGUAGE, thus specifying the desired platforms and language-localized implementation for a [software] component.” *Id.* at 15:30-34. Thus, the specific applet, *e.g.*, an applet in a specific language, is specified using an HTTP header. Notably, “**HTTP headers** are typically added to all HTTP requests made by method 66 and system 56.” *Id.* at 15:34-36.

432. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein the identified form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code] is specified in a HTTP header [*e.g.*, Parthasarathy’s HTTP headers].

*c. [15] The method of claim 11 wherein the identified form of the application is a web page.*

433. Parthasarathy explains “[d]ocuments 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.” EX1007 (Parthasarathy) at 7:3-8. Additionally, “method 66 and system 56 are not limited to downloading software components for HTML

documents. Almost any type of computer software components can be obtained from a local or remote computer using system 56 and method 66 for many types of files or documents.” EX1008 (Kimera) at 9:3-8. A POSA would have understood that an HTML document is presented as a web page. *Id.* at 7:8-11.

434. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein the identified form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code] is a web page [*e.g.*, Parthasarathy’s “World Wide Web site on the Internet” or HTML document located via URL].

*d. [16] The method of claim 11 Wherein the identified form of the application is a portion of a web page.*

435. As explained in §VIII.C.3.c above, Parthasarathy explains “[d]ocuments 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.” EX1007 (Parthasarathy) at 7:3-8. “[M]ethod 66 and system 56 are not limited to downloading software components for HTML documents. Almost any type of computer software components can be obtained from a local or remote computer using system 56 and method 66 for many types of files or documents.” *Id.* at 9:3-8.

A POSA would have understood an HTML document is presented as a web page or a portion of a web page. *Id.* at 7:8-11. *See also* §IV.B above. *See also* EX1014 (Microsoft Computer Dictionary) at 238. *See also* EX1016 (HTML 4.0 Specification).

436. Additionally, “[t]o use dynamic or interactive multimedia in HTML documents, HTML <OBJECT> tags are used. One or more HTML <OBJECT> tags are embedded into a document written in HTML or some other programing or scripting language which permits HTML references.” *Id.* at 9:63-67. In my opinion, a POSA would have understood that the embedded HTML objects would be portions of the HTML document or portions of the web page.

437. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 Wherein the identified form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code] is a portion of a web page [*e.g.*, Parthasarathy’s embedded objects in an HTML document].

*e. [17] The method of claim 11 wherein transforming further comprises verification of the compiled form.*

438. As an initial matter, compressing downloaded applets, verifying downloaded applets, and optimizing applets were each well-known processes as of March 1998. *See* §§IV.B and IV.E above.

439. Parthasarathy teaches a “method and system for downloading, verifying, installing, registering and displaying computer software components.” EX1007 (Parthasarathy) at Abstract, 1:13-15. Kimera similarly discloses a “verification station” that “performs checking of incoming code for typesafety and conformance to interfaces.” EX1008 (Kimera) at 2.

440. In addition to the verification station checking incoming code, Kimera can also apply “tamper-proof signatures, using a cryptographically strong hash function, such as MD5, ... to ensure the safety of the code after it leaves the verifier.” *Id.* Utilizing the verification service in the centralized server improves the overall system performance, because the service is “performed faster, cheaper, and without introducing security vulnerabilities.” *Id.* at 1. Hence, a POSA would have been motivated to host the software download control module within centralized Kimera’s Firewall/server.

441. A POSA would have understood the transforming operations, including verification, take place after the application is compiled for the reasons explained in §VIII.C.2.d above.

442. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein transforming [*e.g.*, Kimera’s verifying, auditing, and compressing software using Parthasarathy’s Lempel-Ziv compression] further comprises verification [*e.g.*, using Kimera’s verification service and

Parthasarathy's software component verification] of the compiled form [e.g., Kimera's disclosure of "[c]ompilation from Java bytecode into native machine code"].

*f. [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form.*

443. As an initial matter, compressing downloaded applets, verifying downloaded applets, and optimizing applets were each well-known processes as of March 1998. *See* §§IV.B and IV.E above.

444. The '740 Patent does not expressly define "optimizing the one or more applets before sending the one or more applets to the one or more client computers." Instead, the '740 Patent describes "[o]ptimizers 46 [that] analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers 46 take in intermediate code 44 and put out transformed intermediate code 50. The transformed intermediate code 50 is functionally equivalent to the source intermediate code 44 in that they share the same structure." EX1001 at 6:1-8. A POSA would have understood that the '740 Patent merely describes standard optimization techniques that, for example, compilers have used long before the patent. *See* §§IV.D-IV.F above.

445. Kimera also details that centralization "enables the same services [e.g., verification, access control, auditing, and compilation services] to be performed

**faster**, cheaper, and without introducing security vulnerabilities.” EX1008 (Kimera) at 1. This centralization “makes a sizable reduction in computation and memory requirements of the end-points” to the point where “Java enabled network computers [clients] could [] be constructed from cheaper components.” *Id.* at 2. In my opinion, this suggests optimizing applets that are ultimately sent to the client devices would be necessary because the client may not otherwise be able to run the downloaded applet on its “reduced computation and memory requirement[]” components. *Id.*

446. The optimized code would be functionally equivalent to the original code but would be optimized to ensure that clients with “reduced computation and memory requirements” could still run the received software. This would be especially important for clients that already had limited resources, such as “small, mobile clients where the Java engine may be hard coded in ROM,” before centralization allowed for even further reduction to the client requirements. *Id.*

447. Moreover, in Kimera, applets are compiled and cached on its centralized Firewall/server. This alone saves a lot of network bandwidth and latency, and hence optimizes the retrieval of applets before they are being sent to the client browsers. *See also* §IV.C above; *see also* EX1018 (Silberschatz) at 43-44.

448. A POSA would have understood that reducing latency in the software acquisition process would be a form of optimization. *See* §§IV.D and IV.E above (describing optimization as “the process of producing more efficient (smaller or

faster) programs through selection and design of data structures, algorithms, and instruction sequences” and the “process of a compiler or assembler in producing efficient executable code”).

449. A POSA would have understood that the transforming operations, including optimization, take place after the application is compiled for the reasons explained in §VIII.C.2.d above.

450. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein transforming [*e.g.*, Kimera’s verifying, auditing, and compressing software using Parthasarathy’s Lempel-Ziv compression] further comprises optimization [*e.g.*, to meet Kimera’s client-requirement-reducing goals] of the compiled form [*e.g.*, Kimera’s disclosure of “[c]ompilation from Java bytecode into native machine code”].

*g. [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response.*

451. As an initial matter, I note that this claim merely requires that the already transformed form of the application be returned to the client computer as an HTTP response. But HTTP requests and responses were well known, heavily used (*e.g.*, GET and POST calls), and standardized well before the ’740 Patent. In fact, most Internet protocols (even before HTTP) inherently use a request-response behavior. *See* §IV.B above; *see also generally* EX1016 (HTML 4.0 Specification).

452. As discussed in §VIII.C.3.b above in relation to claim 14, Parthasarathy teaches using HTTP requests. EX1007 (Parthasarathy) at 14:65-67, 14:67-15:13. In the HTTP protocol (and in fact in most Internet protocols), any data returned to the client is sent in an HTTP response. Such client/server protocols are designed to send a request and receive a response in return. When an applet is requested by the client, Parthasarathy and Kimera render obvious sending the applet to the client computer in an HTTP response. *See also* §§IV.A and IV.B above. *See also* EX1014 (Microsoft Computer Dictionary) at 238 (noting that HTTP is an “[a]cronym for Hypertext Transfer Protocol. The client/server protocol used to access information on the World Wide Web”), 239.

453. Parthasarathy states that “HTTP is a protocol used to access data on the World Wide Web.” EX1007 (Parthasarathy) at 1:45-46. “The illustrated and alternative embodiments of the present invention described above assume that all computer software storage servers searched with the network search path are active HTTP servers capable of handling HTTP POST requests and querying an object database to find the desired software components.” *Id.* at 15:39-44. A POSA would have understood that the “active HTTP servers capable of handling HTTP POST requests and querying an object database to find the desired software components” would also serve the software components to the client computer in an HTTP response. *Id.* at 1:45-46.

454. In sum, in my opinion, Parthasarathy in combination with Kimera teaches the method of claim 11 wherein the transformed form of the application [*e.g.*, Kimera’s verified, audited, and compressed software using Parthasarathy’s Lempel-Ziv compression] is sent to the client computer [*e.g.*, Parthasarathy’s local computer or Kimera’s clients] in an HTTP response [*e.g.*, responses to Parthasarathy’s HTTP requests].

*h. [20pre] The method of claim 11, further comprising:*

*i. [20a] storing a plurality of available forms of the application;*

455. As an initial matter, storing data in a cache is exactly how caches operate. *See* §IV.C above.

456. 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.*

457. 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.*

458. 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 already 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 43-44.

459. 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 application because each platform-specific, or operating-system-dependent version of an application is a distinct form of that application. 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. Kimera also confirms the cache stores results of recent compilations, plural, confirming a plurality of forms corresponding to the plurality of recent compilations, would be stored. *Id.*

460. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the method of claim 11, further comprising the step of: storing a plurality of available forms [*e.g.*, Parthasarathy’s platform-specific, or operating-system-dependent versions, Kimera’s results of recent compilations (plural)] of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code].

- ii. [20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and

461. 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.

462. “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.

463. 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.

464. 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, download, and locally store/cache 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.

465. In order to correctly respond to a client request, the Firewall/server's must be capable of determining if the identified form of the application is one of the plurality of available forms of the application stored in the Code Cache.

466. 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.

467. In sum, in my opinion, the Parthasarathy-Kimera combination teaches the server, in response to receiving the request [*e.g.*, Parthasarathy’s request “to download a desired software component stored on a remote computer”], determining if the identified form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code] is one of the stored plurality of available forms of the application [*e.g.*, Parthasarathy’s platform-specific, or operating-system-dependent versions, Kimera’s results of recent compilations (plural)].

- iii. [20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming.

468. 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.

469. 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 (which acts as a proxy) 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.

470. In my opinion, when the identified form of the application was stored in the Code Cache, the application in the identified form could be sent to the client directly without any additional compiling or transforming operations.

471. In sum, in my opinion, the Parthasarathy-Kimera combination teaches if the identified form of the application [*e.g.*, Parthasarathy’s platform-specific form of the code or Kimera’s native machine code] is one of the stored plurality of available forms [*e.g.*, Parthasarathy’s platform-specific, or operating-system-

dependent versions, Kimera's results of recent compilations (plural)], sending the identified form of the application to the client computer [e.g., Parthasarathy's local computer or Kimera's clients] without compiling and transforming [e.g., Parthasarathy's platform-specific form of the code or Kimera's native machine code, with no additional compilation or transformation].

472. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 20 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

*i. [21pre] The method of claim 11, further comprising:*

*i. [21a] storing a plurality of available forms of the application;*

473. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.3.h.i above. *See also* EX1007 (Parthasarathy) at 24:40-44, 27:15-25; EX1008 (Kimera) at 2; EX1018 (Silberschatz) at 43-44.

*ii. [21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and*

474. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.3.h.ii

above. *See also* EX1007 (Parthasarathy) at 7:3-26, 8:20-31, 13:45-48; EX1008 (Kimera) at 2.

- iii. [21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling.

475. A POSA would find that the Parthasarathy-Kimera combination renders this limitation obvious, for the same reasons as explained in §VIII.C.3.h.iii above. *See also* EX1007 (Parthasarathy) at 8:20-31; EX1008 (Kimera) at 2.

476. While claim limitation [20c] recites “sending the identified form of the application to the client computer without compiling *and transforming*,” this limitation only requires sending the application just “without compiling.” In my opinion, removing the transforming operation does not affect the analysis, because one can transmit a previously compiled application exactly the same way, whether the application is also transformed pre- or post-compilation or not; if the application in the identified form was stored in the Code Cache, the application in the identified form could be sent to the client directly without any additional compiling operations.

477. In my opinion, the Parthasarathy-Kimera combination discloses, or at least teaches, each limitation of claim 21 of the '740 Patent. It is, therefore, my opinion that this claim is obvious in view of Parthasarathy and Kimera.

**D. Ground 4: Claim 13 is Unpatentable Under 35 U.S.C. §103(a) as Obvious over Parthasarathy in view of Kimera and Angles**

478. Dependent claim 13 is obvious over Parthasarathy combined with Kimera and Angles. Below I first address the strong motivations for this combination and then I explain how each element of claim 13 is disclosed, taught, and/or suggested by Parthasarathy in combination with Kimera and Angles.

**1. Motivation to Combine**

479. In my opinion, there is strong motivation to combine Parthasarathy, Kimera, and Angles, for at least two discrete reasons: using Angles's cookies (which would have been well-known to a POSA) in Parthasarathy's HTTP requests (1) would have enabled applet requests to be more specific and tailored to each client, and (2) would have enabled otherwise non-existent functionality such as shopping carts on the Web. I explained the motivation to combine Parthasarathy and Kimera in §VIII.C.1 above.

480. In my opinion, there is additional motivation to combine Parthasarathy's HTTP requests and HTTP headers with the persistent cookie disclosure of Angles, to the extent cookies were not already inherent in Parthasarathy's use of standard HTTP (*see* §IV.B above). Incorporating the functionality of Angles cookies into Parthasarathy-Kimera's client-server interface over HTTP would have been trivial.

481. First, Parthasarathy teaches the use of “HTTP headers for ACCEPT (MIME type) and ACCEPT-LANGUAGE [that] specify[] the desired platforms and language-localized implementation for a [software] component.” EX1007 (Parthasarathy) at 15:30-38. Angles further teaches including a line in the “HTTP header line” “whenever the browsing software sends a HTTP request for a URL on a server for which it has stored cookies....” EX1010 (Angles) at 11:13-15, 11:27-31.

482. A POSA would have been motivated to incorporate Angles’s persistent storage of cookies in Parthasarathy’s HTTP header for all requests to a server in order to relay any relevant information about the client or the client’s browsing session to the server. This, for example, permits web servers to optimize code and display formats for the specific clients (*e.g.*, a handheld device with a smaller display would need the web content formatted to fit properly in a smaller screen, compared to a desktop with a larger screen). See §IV.B above.

483. Angles’s cookies are “small piece[s] of information which a web server (via a CGI script) can store with a web browser and later read back from that browser. This is useful for having the browser remember some specific information across several pages....” *Id.* at 11:5-9. One exemplary cookie described in Angles is “the consumer member code 22,” which is “stored on the consumer computer 12

in a ‘cookie’.” *Id.* at 11:2-4. “The consumer member code 22 stores a code which uniquely identified each consumer.” *Id.* at 10:60-61.

484. Angles’s understanding of cookies matches that of POSA. A “cookie” is, “[o]n 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). *See also* §IV.B above.

485. In my opinion, to the extent Parthasarathy’s use of “HTTP headers” that “specify[] the desired platforms and language-localized implementation for a [software] component” (EX1007 (Parthasarathy) at 15:30-38) were not already understood to encompass standard HTTP cookies, a POSA would have been motivated to include Angles’s cookies to “uniquely identified each consumer.” EX1010 (Angles) at 10:60-61. A POSA would have been motivated to incorporate cookies because they are “useful for having the browser remember some specific information across several pages....” EX1010 (Angles) at 11:5-9.

486. Second, a POSA would have been motivated to incorporate Angles’s cookies to enable software requests spanning multiple Web pages or even multiple sessions. One example of this functionality is a “virtual shopping cart” that spans

multiple Web pages. *Id.* at 11:5-12 (“This is useful for having the browser remember some specific information across several pages; for example, when the consumer browses through a ‘virtual shopping mall’ and add items to his ‘shopping cart,’ a list of the items he has picked up is kept in the consumer browser’s cookie file so that the consumer can pay for all the items at once he has finished shopping.”). In my opinion, a POSA would have been motivated to include this functionality as users would find it highly convenient, and even beneficial, to shop for multiple items over various Web pages and pay for all of the items in the shopping cart in a single transaction.

487. In my opinion, A POSA would have recognized the benefits of persistent storage in browsers via cookies and would have been motivated to include cookies in the HTTP headers of requests for software components and other applets as disclosed in Parthasarathy combined with Kimera.

488. Finally, because cookies are simply part of standard HTTP headers, which Parthasarathy already discloses, a POSA would have been able to incorporate this functionality with great ease and a high expectation of success. *See* EX1007 (Parthasarathy) at 15:30-38.

## 2. Dependent Claim 13

- a. 13. The method of claim 11 wherein the identified form of the application is specified using a cookie.*

489. In my opinion, Parthasarathy combined with Kimera teaches the method of claim 11 for the reasons I have set forth in §VIII.C.2 above. As discussed in §VIII.C.3.b above, Parthasarathy teaches “**HTTP headers** are typically added to all HTTP requests made by method 66 and system 56.” EX1007 (Parthasarathy) at 15:34-36. Angles teaches that “whenever the browsing software sends a HTTP request for a URL on a server for which it has stored cookies, it includes a line [in the HTTP header] in the form: Cookie: NAME=VALUE; NAME=VALUE; ... which lists all cookies that apply to that particular URL.” EX1010 (Angles) at 11:16-31.

490. Angles teaches that “[a] ‘cookie’ is a small piece of information which a web server (via a CGI script) can store with a web browser and later read back from that browser. This is useful for having the browser remember some specific information across several pages; for example, when the consumer browses through a ‘virtual shopping mall’ and add items to his ‘shopping cart,’ a list of the items he has picked up is kept in the consumer browser’s cookie file so that the consumer can pay for all the items at once he has finished shopping.” *Id.* at 11:5-12.

491. In my opinion, a POSA would have been motivated to incorporate Angles's cookie functionality into Parthasarathy's HTTP headers in order to enable functionality such as allowing a "consumer [to] browse[] through a 'virtual shopping mall' and add items to his 'shopping cart,' a list of the items he has picked up is kept in the consumer browser's cookie file so that the consumer can pay for all the items at once he has finished shopping." *Id.* at 11:5-12; *see also* EX1017 (HTTP State Management Mechanism RFC 2109) at 2.

492. In my opinion, the same persistent storage structure would also apply to requested applets and applications; when a webpage or software component was requested via an HTTP request (as taught by Parthasarathy), a cookie in the HTTP header would identify the form of the application (*e.g.*, the code language of the desired applet required by the client system). *See* EX1007 (Parthasarathy) at 15:30-38 ("HTTP headers for ACCEPT (MIME type) and ACCEPT-LANGUAGE [that] specify[] the desired platforms and language-localized implementation for a [software] component.").

493. In my opinion, therefore, to the extent Parthasarathy's HTTP headers did not already include cookies, there is a strong motivation to include Angles's cookies to improve the user experience for the reasons detailed in §VIII.D.1 above.

494. In sum, in my opinion, Parthasarathy in combination with Kimera and Angles teaches the method of claim 11 wherein the identified form of the application

[*e.g.*, Parthasarathy's platform-specific form of the code or Kimera's native machine code] is specified using a cookie [*e.g.*, Angles' cookies in standard HTTP headers].

## **IX. CONCLUSION**

495. In my opinion, claim 11-21 of the '740 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.

A handwritten signature in black ink, appearing to read "Erez Zadok". The signature is stylized with a large, looped "Z" and a cursive "K".

May 1, 2024

Date

---

Erez Zadok, Ph.D.

## CLAIM APPENDIX OF THE CHALLENGED CLAIMS

- [11pre] A method operating on a computer system, having a client computer and a server computer, for managing requests to the server computer, the method comprising:
  - [11a] at the server computer, receiving a request from the client computer, the request identifying an application and identifying a form of the application; and
  - [11b] in response to receiving the request: compiling the application into a compiled form;
  - [11c] transforming the compiled application into a transformed form of the compiled form of the application, wherein transforming comprises execution and compression of the compiled form; and
  - [11d] sending the transformed form of the application to the client computer.
- [12] The method of claim 11 wherein the identified form of the application is specified using a Universal Resource Locator.
- [13] The method of claim 11 wherein the identified form of the application is specified using a cookie.
- [14] The method of claim 11 wherein the identified form of the application is specified in a HTTP header.
- [15] The method of claim 11 wherein the identified form of the application is a web page.

- [16] The method of claim 11 wherein the identified form of the application is a portion of a web page.
- [17] The method of claim 11 wherein transforming further comprises verification of the compiled form.
- [18] The method of claim 11 wherein transforming further comprises optimization of the compiled form.
- [19] The method of claim 11 wherein the transformed form of the application is sent to the client computer in an HTTP response.
- [20pre] The method of claim 11, further comprising:
  - [20a] storing a plurality of available forms of the application;
  - [20b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and
  - [20c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling and transforming.
- [21pre] The method of claim 11, further comprising:

- [21a] storing a plurality of available forms of the application;
- [21b] the server, in response to receiving the request, determining if the identified form of the application is one of the stored plurality of available forms of the application; and
- [21c] if the identified form of the application is one of the stored plurality of available forms, sending the identified form of the application to the client computer without compiling.