REMARKS

I. Introduction

Claims 1-8, 10-17, and 19-22 are pending in the present reexamination of U.S. Patent No. 7,426,720 ("the '720 Patent"). The '720 Patent is directed to a system, apparatus, and method for dynamic preloading of classes. This is achieved through the following unique combination of elements as set forth in exemplary claim 1:

- A **"class preloader"**[1] to obtain a representation of at least one class from a source definition provided as object-oriented program code;
- A **"master runtime system process"** to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process;
- A **"runtime environment"** to clone the memory space as a child runtime system process responsive to a process request; and
- A **"copy-on-write process cloning mechanism"** to copy references to the memory space of the master runtime system process into a separate memory space of the child runtime system process and to defer copying of the master's memory space until the child needs to modify the referenced memory space.

The Office during the original examination found this combination of elements--specifically the **"runtime environment"** and **"copy-on-write cloning mechanism"**--to be patentable over the prior art. (Notice of Allowance at 2-3.)

The art of the present reexamination does not compel a different conclusion. None of the cited references teaches or suggests the claimed combination of elements. First, the Office cited Dike and Steinberg as disclosing the claimed combination. (Action at 16 (Grounds 2 and 3).) Dike and Steinberg, however, disclose the operation of Linux processes, including user mode Linux

[1] Throughout this Response, recited claim language is in quotes and boldface.

kernels, which have no relationship to, nor disclose in any way, **"to obtain a representation of at least one class from a source definition provided as object-oriented program code,"** and **"to interpret and to instantiate the representation."** Thus, Dike and Steinberg fail to meet the base element of a **"class preloader,"** let alone the elements of a **"master runtime system process,"** a **"runtime environment,"** and a **"copy-on-write process cloning mechanism."** The combination of Srinivasan and Bach (Ground 6) has similar deficiencies.

Second, while the Office recognizes that the remaining references are deficient as to one or more claimed elements, many of the asserted combinations simply do not address the recognized deficiency. To establish *prima facie* obviousness, all the claimed elements must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974); *KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. 398 (2007); MPEP 2141. For example, the Office acknowledges that Sexton fails to disclose the element of a **"copy-on-write process cloning mechanism"** and cites Bugnion and Johnson to meet this deficiency. (Action at 44, 53-54 (Grounds 7 and 8).) But Bugnion does not disclose the **"copy-on-write process cloning mechanism"** with a **"master runtime system process"** and a **"child runtime system process"** that references the memory space of the master runtime system process. Johnson is just as deficient as Bugnion.

Third, even if assuming an asserted combination in the Action meets all the recited elements, the combination is based on impermissible hindsight. For example, in Grounds 4 and 5, the Office combines Bryant with Bach or Traut based on their alleged disclosures of a **"copy-on-write process cloning mechanism."** (Action at 26, 32.) However, "[a] critical step in analyzing the patentability of claims pursuant to section 103(a) is casting the mind back to the time of invention, to consider the thinking of one of ordinary skill in the art, guided only by the prior art references and the then-accepted wisdom in the field." *In re Kotzab*, 217 F.3d 1365, 1369 (Fed. Cir. 2000). The skilled person guided by Bryant, Bach, and Traut would not have reached the claimed invention. Bryant teaches away from Bach and Traut's copy-on-write, because it expressly points to

substantial memory usage in a web server environment having extensive memory resources. Thus, rather than cast its mind back to the time of invention to be guided by what the references teach, the Office has simply used the claimed elements as a "blueprint" or "road map" to find them in the prior art. See, e.g., *Interconnect Planning Corp. v. Feil*, 774 F.3d 1132, 1139 (Fed. Cir. 1985).

Indeed, the secondary considerations affirm that the claimed combination of elements in the '720 Patent would not have been obvious to the skilled person at the time of the invention. The '720 Patent taught a new approach to virtual machine memory management, such as in Java virtual machines. Others, including the cited references, developed different, elaborate, complicated, and expensive schemes for virtual machine memory management. The claimed invention thus satisfied a long-felt need in virtual machine memory management and was immediately copied, including by Requester Google.

Accordingly, Patent Owner submits that all of the rejections should be withdrawn.

II. Brief Summary of the Invention

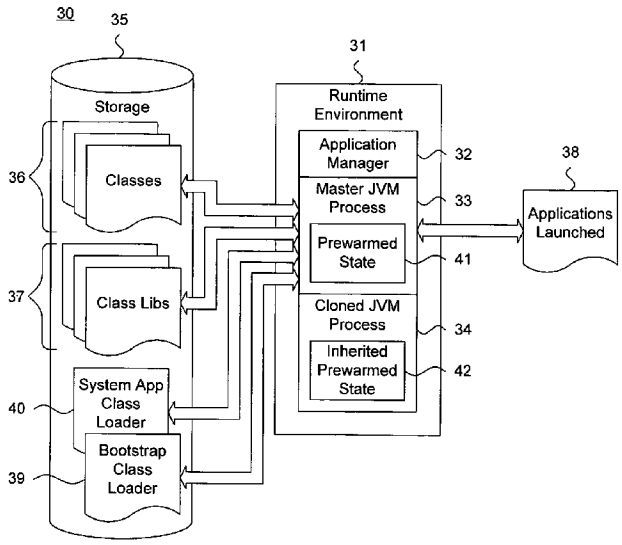
The breakthrough of the '720 Patent was inventing a new approach to virtual machine memory management and startup by using copy-on-write with process cloning of virtual machines. (Appendix A, Goldberg Declaration, ¶9.) This breakthrough is expressly reflected in the claims as a unique combination of elements: a **"class preloader,"** a **"master runtime system process,"** a **"runtime environment to clone,"** and a **"copy-on-write process cloning mechanism."** This unique combination of elements is fully described in the specification and drawings of the '720 Patent.

FIG. 2 of the '720 Patent, reproduced below, is an illustrative example of the claimed combination. With respect to the element of a **"class preloader,"** FIG. 2 illustrates a bootstrap class loader 39 and system application class loader 40 that can obtain representations of classes 36 and classes defined in class libraries 37 from storage 35. The classes 36 and the classes defined in class libraries 37 can be stored as object-oriented program code. With respect to the element of a **"master**

runtime system process," master Java virtual machine (JVM) process 33 can interpret and instantiate the object-oriented-coded classes 36 and classes defined in the class libraries 37 to create class instances and can store the instances in the memory space of the master process as prewarmed state 41. (5:7-6:66.)

As for the element of a **"runtime environment to clone,"** runtime environment 31 can create cloned (or child) JVM process 34 as a new instance of the master JVM process 33, in response to launched application 38, where the cloned process can inherit the prewarmed state 41 of the master process as inherited prewarmed state 42 stored in the cloned process's memory space. The class instances forming the inherited prewarmed state 42 can be the class instances that formed the prewarmed state 41 from interpreted and instantiated object-oriented program code. (*Id.*)

Fig. 2.



Finally, using the copy-on-write variant of process cloning in the element of a **"copy-on-write process cloning mechanism,"** the memory space of the cloned JVM process 34, at creation, can include references to the memory space of the master JVM process 33, e.g., references to memory space segments containing classes instantiated from object-oriented program code.

Segments of the referenced memory space of the master JVM process 33 can be copied to the memory space of the cloned JVM process 34 upon an attempt by the cloned process to modify the master process's memory space. That is, a segment of the memory space--such as one containing classes instantiated from object-oriented program code--can be copied to the cloned process's memory space when the cloned process tries to modify the data associated with the segment. (*Id.*)

Use of the **"copy-on-write process cloning mechanism"** in virtual machines provides several advantages. The cloned process can inherit preloaded classes from the master process, thereby making the cloned process more quickly available for execution. The cloned process can execute as an independent process, making data isolation, process invocation and termination, and resource management cleaner. User applications executing the cloned process can initialize faster and exhibit more deterministic runtime behavior. Deferred copying of non-shareable master process memory space can reduce memory usage and lessen the performance impact of immediate and/or unnecessary copying. (3:21-40.) (Goldberg Declaration, ¶9.)

Importantly, during the original examination, the Office did not allow the claims based on any given element, such as the **"copy-on-write process cloning mechanism."** Rather, in the Notice of Allowance, the Office conferred patentability for the combination of elements, i.e., using a **"runtime environment to clone"** and a **"copy-on-write process cloning mechanism"** in the context of a virtual machine:

> [T]he prior art of record fails to teach and/or suggest a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process and a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process as recited in independent claims. (emphasis added.)

As discussed in detail below, the cited references in the pending reexamination also fail to teach or suggest the novel combination.

III. The Obviousness Rejection Based on the Combination of Webb, Kuck, and Bach Should Be Withdrawn (Ground 1)

The Office relies on three different references to meet the combination of elements. Webb is relied on for meeting the elements of **"class preloader"** and **"master runtime system process."** (Action at 5-8.) Kuck and Bach are respectively cited for the elements of a **"runtime environment to clone"** and a **"copy-on-write process cloning mechanism."** (*Id.* at 8-10.)

This combination resurrects a nearly identical rejection made and overcome in the original examination. There, the Office also cited Webb for the elements of a **"class preloader"** and **"master runtime system process"** and Kuck for a **"runtime environment to clone."** (Office Action dated April 27, 2007, at 3-4.) Bach was not cited, but it was described in the background of the '720 Patent and portions were incorporated by reference. (4:53-58; 5:37-40.) Indeed, the discussion relating to copy-on-write process cloning (4:66-5:6) was not incorporated, but rather expressly placed in the body of the specification for the reader to read.

Notwithstanding its knowledge of Webb, Kuck, and Bach, the Office allowed the claims over them. In this reexamination, the Office should reach the same conclusion as there are a number of reasons described below why the claims are patentable over this combination of references. First, the Webb, Kuck, and Bach combination does not disclose the **"runtime environment."** In particular, Kuck, cited for this element, lacks the requisite cloning, i.e., **"to clone,"** in a **"runtime environment."** As stated in the Introduction, to establish *prima facie* obviousness, all the claimed elements must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974); *KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. 398 (2007); MPEP 2141. Second, Webb, Kuck, and Bach are not combinable because: (a) Webb teaches away from the combination regarding cloning; (b) Kuck teaches away from the combination regarding copy-on-write; (c) the combination changes the principle of operation of Webb from a single reusable virtual machine to

multiple independent virtual machines; (d) the combination changes the principle of operation of Kuck from simple shared memory to copy-on-write memory; and (e) the reasons for the combination are insufficient to support a *prima facie* case of obviousness. Third, the combination does not disclose the dependent claims.

A. The Combination of Webb, Kuck, and Bach Does Not Disclose or Suggest the Claimed Runtime Environment

Claim 1[2] recites, *inter alia*:

> a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process. (emphasis added.)

The Office relies solely on Kuck, specifically ¶¶[0064]-[0065], for this element. (Action at 8.) The cited paragraphs describe reducing initialization overhead when initializing a new process attachable virtual machine (PAVM) by copying the memory block of a master PAVM into the memory block of the new PAVM. Patent Owner submits that the Office has mistakenly equated "initialization" with **"to clone."** Kuck's initialization is performed after creating the new PAVM to set initial values for the new PAVM. Therefore, Kuck's initialization of a new PAVM based on an already-initialized "master" PAVM does not actually create the new PAVM when the memory block is copied. The new PAVM already exists, having been created as described in ¶¶[0062]-[0063]. Rather, Kuck's initialization merely copies initialization data from the master PAVM to the new PAVM. Kuck thus cannot be considered to disclose that the new PAVM is cloned. (Goldberg Declaration, ¶10.)

Because Kuck does not disclose cloning, Kuck does not disclose the **"runtime environment to clone."** These deficiencies of Kuck are not corrected by either Webb or Bach. The Office acknowledges that Webb does not disclose the **"runtime environment"** and does not address this

---

[2] Throughout this Response, Patent Owner refers to exemplary independent claim 1. The analysis as to claim 1 applies to the other independent claims (claims 10 and 20) unless otherwise noted.

element with respect to Bach. (Action at 8.) Patent Owner notes that Bach also does not disclose this element.

B. There Would Have Been No Reason for the Webb-Kuck-Bach Combination

Webb, Kuck, and Bach are not properly combinable for at least the following reasons.

1. Webb Teaches Away from the Combination

Webb discloses the desirability of running successive applications on the same Java virtual machine, by reinitializing classes in the Java virtual machine between applications. (1:39-41; 2:5-25; 4:8-11; claim 1.) It is clear that Webb leads the skilled person to reuse the same Java virtual machine for multiple applications. This is fundamentally different from Kuck, which is directed to creating multiple Java machines (Kuck Abstract), and Bach, which is directed to creating multiple processes (Bach, Chapter 7). As such, Webb teaches away from using any process, such as Bach's cloning, that would create multiple Java virtual machines. (Goldberg Declaration, ¶11.) "'Teaching away' does not require that the prior art foresaw the specific invention that was later made, and warned against taking that path." *Spectralytics, Inc., v. Cordis Corp.* (Fed. Cir. June 13, 2011). Rather, the design of the prior art device itself can teach away from the invention. (*Id.*) See also, e.g., MPEP 2145(X)(D), which describes teaching away as an improper rationale for combining references.

2. Kuck Teaches Away from the Combination

Kuck discloses the allocation of a memory block for a new PAVM: "a PAVM is generated and initialized for that user session (502). That can include allocating a block of memory for the PAVM…After a block of memory has been allocated to the PAVM, the PAVM can be stored in the memory block." (¶¶[0062]-[0063].) This is fundamentally different from Bach's copy-on-write process, in which no allocation of a memory block is performed when a new process is forked. As such, Kuck teaches away from using any process, such as copy-on-write, that would avoid or defer until necessary memory block allocation for a new virtual machine. (Goldberg Declaration, ¶13.)

3. The Combination Changes the Principle of Operation of Webb

Modifying Webb to implement cloning would eviscerate its principle of operation. The cloning would provide separate Java virtual machines. As a result, there would no longer be a need to reinitialize classes on the same virtual machine; hence, no need for Webb. (MPEP 2143.01(VI).) (Goldberg Declaration, ¶12.)

4. The Combination Changes the Principle of Operation of Kuck

Similarly, modifying Kuck to implement cloning would eviscerate its principle of operation. It would provide for generating Java virtual machines upon demand. As a result, there would no longer be a need to generate and reuse persistent PAVMs; hence, no need for Kuck. Moreover, copy-on-write would replace a simple shared memory with a copy-on-write memory. (Goldberg Declaration, ¶14.)

5. Requester's Reasons to Combine Are Insufficient

Given the above discussion, it is clear that Requester's reasons for the combination of Webb, Kuck, and Bach, adopted by the Office, are insufficient to establish a *prima facie* case of obviousness. The Action, at 10, states that Webb and Kuck could be combined "to provide avoiding the overhead and crashing the process and enabling the server to run robustly."

Patent Owner disagrees. First, Kuck does not disclose the **"runtime environment"** to clone a memory space, as described above. However, even if the **"runtime environment"** were added to Kuck, the skilled person reading Webb would not be motivated to modify Webb to include cloning because Webb specifically teaches the desirability of using a single Java virtual machine (1:39-41; 2:5-25; 4:8-11; claim 1), thereby eliminating the need for cloning to create multiple virtual machines. Second, adding cloning would increase system overhead and increase the likelihood of crashing processes because of the increased number of virtual machines running the processes, negating any alleged benefit from Kuck.

The Action, at 10, further states that Bach could be combined with Webb and Kuck "to reduce memory usage and overhead impact when instantiating and executing virtual machines."

Patent Owner again disagrees. First, the combination of Webb and Kuck does not provide all the claimed elements, i.e., the **"runtime environment,"** as described above. Second, even if the **"runtime environment"** were added to Kuck, implementing cloning, even with copy-on-write, would increase the memory usage and overhead impact of Webb's operation by creating and using multiple Java virtual machines, rather than reusing the same virtual machine. Lastly, because Kuck's operation already performs efficiently and effectively by creating and storing PAVMs for later use in the disclosed manner, there would have been no need to implement copy-on-write. As stated during prosecution of the '720 Patent (December 18, 2007 Amendment), Webb and Kuck were concerned with reducing the overhead needed when executing a virtual machine (e.g., Webb, 2:5-16; Kuck, ¶¶ [0064]-[0065]), but did not attempt to reduce the memory usage for the virtual machine, as provided by copy-on-write.

Moreover, given that copy-on-write technology was known as early as 1986, when Bach was published and at least fourteen years before Webb and Kuck were filed as patent applications, it clearly was not obvious to use copy-on-write "to reduce memory usage and overhead impact when instantiating and executing virtual machines." Otherwise, either Webb or Kuck would have done so. See also Goldberg Declaration, ¶¶10-14.

C. The Rejection Regarding the Dependent Claims Should Be Withdrawn

The dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. For example, claims 4 and 13 directed to a **"class resolver"** are patentable at least by virtue of the patentability of their independent claims. Claims 6 and 15 directed to a **"process cloning mechanism"** are patentable because Kuck, cited for this element, merely discloses the initialization of a new PAVM based on an already-initialized master PAVM, and does not disclose that the new PAVM is created as the result of **"cloning."** Claims 21 and 22 directed to a **"resource controller"** are also patentable because Kuck's PAVMs cannot be considered to be cloned and therefore cannot provide the requisite **"child runtime system process"**

on which the **"resource controller"** operates. For the reasons described above, Webb and Bach are not combinable with Kuck to correct its deficiencies.

IV. The Anticipation and Obviousness Rejections Based on the Combination of Dike and Steinberg Should Be Withdrawn (Grounds 2 and 3)

The combination of Dike and Steinberg neither anticipates nor renders obvious the claimed invention for the reasons described below. As an initial matter, the anticipation rejection fails to satisfy the requirements for using multiple references as a basis for anticipation under MPEP 2131. Additionally, Dike and Steinberg are not combinable because: (a) the combination fails to disclose any of the claimed elements as the combination discloses a Linux environment executing C and C++ programs which lack the base element, i.e., the **"class preloader,"** and hence the remaining claimed elements devised therefrom; and (b) Dike teaches away from the **"copy-on-write process cloning mechanism"** by explicitly changing from copy-on-write memory segments to shared segments. The combination also does not disclose the dependent claims.

A. The Anticipation Rejection Is Not Proper

The Office rejects the claims as being anticipated by Dike in view of Steinberg. (Action at 10.) While multiple references may be proper for an anticipation rejection in certain circumstances (see MPEP 2131), neither the Office nor Requester describes how any of those circumstances are applicable here. In fact, it appears from the Action, the anticipation rejection (Ground 2) simply collapses into the obviousness rejection (Ground 3). Accordingly, Patent Owner will address the two rejections together.

B. The Combination Does Not Disclose or Suggest Each and Every Element of the Claims

As described in detail below, Dike is directed to a different technology than the '720 Patent. This renders Dike and its combination with Steinberg deficient on numerous grounds.

1. The Combination Does Not Disclose the Claimed Class Preloader

Claim 1 recites, *inter alia*:

> a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code. (emphasis added.)

The Office asserts that Dike discloses this element in §2.2, §3. (Action at 16.) On the contrary, neither these nor any other sections of Dike disclose or even suggest a **"class preloader"** or any other preloader that **"obtain[s] a representation of at least one class from a source definition provided as object-oriented program code."** This is because Dike discloses a Linux environment, which executes C and C++ programs that do not provide a representation that can be interpreted and instantiated as a class definition. Rather C and C++ programs are simply compiled into machine code that is then executed. On the other hand, the **"class preloader"** provides **"a representation of at least one class from a source definition provided as object-oriented program code"** that is interpreted and instantiated as a class definition. A Linux virtual machine in machine code compiled from C and C++ has no need to be prewarmed, i.e., preloaded with the class data. Thus, the combination's Linux virtual machine does not lead to a **"class preloader."** (Goldberg Declaration, ¶15.)

In §2.2, Dike discloses the initialization of a system in preparation for booting up a user-mode kernel. However, there is nothing in this system initialization that involves a **"class preloader."** Dike's mention of the initialization being "analogous to the boot loader on a physical machine" is not the same as the **"class preloader."** The boot loader loads the programs that run the physical machine, without also performing preloading that **"obtain[s] a representation of at least one class from a source definition provided as object-oriented program code."**

In §3, Dike discloses various applications, including daemons and services, that are started when the user-mode kernel boots up. However, none of the various applications is started by a **"class preloader"** or includes **"a representation of at least one class from a source definition provided as object-oriented program code."**

The deficiencies of Dike are not corrected by Steinberg. In Steinberg, Fiasco-UX can include C++ code and can link to glibc, a C library for Linux (p.14). However, the C++ code does

not provide the **"class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code."** Steinberg's brief disclosures of C++ code and glibc are not associated with a **"class preloader."** (Goldberg Declaration, ¶15.)

2. The Combination Does Not Disclose the Claimed Master Runtime System Process

Dike and Steinberg are deficient as to the next element of claim 1 as well:

> a master runtime system process to interpret and to instantiate the representation [of at least one class] as a class definition in a memory space of the master runtime system process. (emphasis added.)

The Office asserts that Dike discloses this element in §2.2, §2.3, §3. (Action at 16.) On the contrary, neither these nor any other sections of Dike disclose or even suggest that a parent process in the user-mode kernel has the ability **"to interpret and to instantiate the representation [of at least one class] as a class definition"** in the parent's address space. There can be no **"master runtime system process"** if there is no **"class preloader"** to provide the **"representation [of at least one class] as a class definition"** for interpreting and instantiation. (Goldberg Declaration, ¶15.) Moreover, because C and C++ are not interpreted languages, Dike's Linux virtual machine in C or C++ cannot **"interpret and [] instantiate the representation [of at least one class] as a class definition."**

In §2.3, Dike simply discloses how the fork or clone system call is implemented in a Linux user-mode kernel. However, this has nothing to do with a **"class definition."** That is, Dike does not disclose that the parent's address space contains an interpreted and instantiated **"representation [of a class] as a class definition."** Indeed, Dike does not disclose what the contents of the parent's address space are.

Dike §2.2, §3 are described above. For the same reasons that these sections do not disclose the **"class preloader,"** they also do not disclose the **"master runtime system process."**

The deficiencies of Dike are not corrected by Steinberg because Steinberg's brief disclosures of C++ code and glibc are not associated with a **"master runtime system process."** (Goldberg Declaration, ¶15.)

3. The Combination Does Not Disclose the Claimed Runtime Environment

Dike and Steinberg are further deficient regarding this element of claim 1, which recites:

> a runtime environment to clone the memory space [of the master runtime system process] as a child runtime system process responsive to a process request and to execute the child runtime system process. (emphasis added.)

The Office asserts that Dike discloses this element in §2.3, described above. (Action at 16.) But this assertion fails because Dike does not disclose the **"class preloader,"** the **"master runtime system process"** and its **"memory space,"** e.g., its memory space contents, as described above. (Goldberg Declaration, ¶15.) Rather, as stated above, Dike (§2.3) simply discloses how the fork or clone system call is implemented. It has nothing to do with a **"class definition."** Dike also does not disclose that the new process is the **"child runtime system process"** whose copied address space is a copy of the **"memory space [of the master runtime system process]"** that includes a **"class definition."** Rather Dike makes no mention of the contents of that address space in this regard. Because Dike does not disclose the above elements, Dike does not disclose the **"runtime environment"** that recites them.

The deficiencies of Dike are not corrected by Steinberg because Steinberg's brief disclosures of C++ code and glibc are not associated with a **"runtime environment."**

4. The Combination Does Not Disclose the Claimed Copy-On-Write Process Cloning Mechanism

Dike and Steinberg are also deficient regarding this last element of claim 1, which recites:

> a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime

> system process needs to modify the referenced memory space of the master runtime system process. (emphasis added.)

The Office asserts that Dike discloses this element in §2.1, §2.3. (Action at 17.) But this assertion similarly fails because Dike does not disclose the **"class preloader,"** the **"master runtime system process,"** the **"child runtime system process,"** and their respective **"memory spaces,"** e.g., the memory space contents, as described above. (Goldberg Declaration, ¶15.) In §2.1, Dike describes the address space having a shared memory segment that is converted from a copy-on-write memory segment in order to share kernel data. However, this is a disclosure not of using a **"copy-on-write process cloning mechanism,"** but of the opposite. Dike's shared segment does not **"defer copying"** of the parent's memory space until the new process **"needs to modify"** the memory space. Dike prevents such deferring by making the address space a shared one. Moreover, Dike's shared segment is not the **"memory space of the master runtime system process"** because the address space's kernel data does not include a **"class definition."** Dike does not disclose the kernel data contents.

Dike §2.3 is described above. For the same reasons that this Dike section does not disclose the other claim elements, it also does not disclose the **"copy-on-write process cloning mechanism."**

The deficiencies of Dike are not corrected by Steinberg because Steinberg's brief disclosures of C++ code and glibc are not associated with a **"copy-on-write process cloning mechanism."** (Goldberg Declaration, ¶15.)

C. There Would Have Been No Reason for the Dike-Steinberg Combination

Dike and Steinberg are not properly combinable because the combination teaches away from the claimed invention. The Office points out that the Dike-Steinberg combination is "directed to virtual machines in a Linux environment." (Action at 17.) As such, the skilled person reading Dike and Steinberg would have been led toward a Linux environment, which executes C and C++

programs that do not provide representations of a class that can be interpreted and instantiated as a class definition, and away from the '720 Patent.

Additionally, Dike teaches away from the **"copy-on-write process cloning mechanism."** The only reference to copy-on-write in Dike (§2.1) states that the copy-on-write memory segment is converted to a shared segment. As such, the skilled person reading Dike would have been led toward shared memory and away from copy-on-write memory. See also Goldberg Declaration, ¶15.

D. The Rejection Regarding the Dependent Claims Should Be Withdrawn

The dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. For example, claims 4 and 13 directed to a **"class resolver"** are patentable because Dike fails to disclose the **"class preloader"** that obtains a **"class definition"** and therefore cannot disclose the **"class resolver"** to resolve the absent class definition. Claims 6 and 15 directed to a **"process cloning mechanism"** are also patentable because Dike's address space is a shared memory space, not a copied **"memory space."** Claims 21 and 22 directed to a **"resource controller"** are further patentable because Dike does not disclose the requisite **"child runtime system process,"** as described above, on which the **"resource controller"** operates. The deficiencies of Dike are not corrected by Steinberg for at least the reasons described above.

V. The Obviousness Rejection Based on the Combination of Bryant and Bach Should Be Withdrawn (Ground 4)

The Office found that the combination of the claimed elements was novel and nonobvious during the original prosecution of the '720 Patent. See Sections II and III, above. Webb, Kuck, and Bach were before the Office and were specifically indicated as not being combinable. Bryant is cumulative of what the Office alleged during the original prosecution and would not have been combined with Bach for the same reasons.

Additionally, the Bryant and Bach combination does not render obvious the claimed invention for the reasons described below. Bryant and Bach are not combinable because: (a) Bryant teaches away from the combination regarding copy-on-write cloning; (b) the combination changes

the principle of operation of Bryant regarding how the fork system call is implemented; (c) the reasons for the combination are insufficient to support a *prima facie* case of obviousness; and (d) the combination is based on impermissible hindsight that uses the '720 Patent as a road map to find the claimed elements in the prior art. The combination also does not disclose the dependent claims.

A. There Would Have Been No Reason for the Bryant-Bach Combination

1. Bryant Teaches Away from the Combination

Bryant discloses a Java server preloading all the classes that would be potentially needed by requesting applications. (2:46-54.) Bryant also discloses a child Java server selecting a subset of the preloaded classes when a particular Java application executes. (*Id.*) Clearly, by loading all potentially needed classes, Bryant is not concerned about memory usage reduction, having been implemented in a web server environment having substantial memory resources. Rather, Bryant is directed to faster startup at the expense of memory usage. (2:32-36, 46-63; 7:36-40.) As such, Bryant teaches away from or at least fails to provide any motivation to combine Bryant with any process, such as copy-on-write, that would reduce memory usage. (Goldberg Declaration, ¶16.)

Moreover, the description in Bach (pp. 289-290) of the motivation for copy-on-write teaches away from a combination with Bryant. Bach reads: "Traditionally, the kernel of a swapping system makes a physical copy of the parent's address space, usually a wasteful operation, because processes often call *exec* soon after the *fork* call and immediately free the memory just copied." (Underline added.) This is fundamentally different from Bryant because, in the method of Bryant, the use of a *fork* call will not be followed by a call to an *exec*. (Goldberg Declaration, ¶16.)

2. The Combination Changes the Principle of Operation of Bryant

Modifying Bryant to implement a copy-on-write memory would change the principle of operation of Bryant. The modification would require the operating system to change how it implements the fork system call. (Goldberg Declaration, ¶17.)

3. Requester's Reasons to Combine Are Insufficient

Given the above discussion, it is clear that Requester's reasons for the combination of Bryant and Bach, which were adopted by the Office, are insufficient to establish a *prima facie* case of obviousness. The Action, at 27, states that Bryant and Bach could be combined "to streamline and accelerate a Java machine."

However, Bryant already discloses streamlining and accelerating a Java machine by loading all potentially needed classes for faster startup and then selecting the actually needed classes therefrom. Given that copy-on-write technology was known as early as 1986, when Bach was published and twelve years before Bryant was filed as a patent application, it clearly was not obvious to use copy-on-write "to streamline and accelerate a Java machine." Otherwise, Bryant would have done so. See also Goldberg Declaration, ¶¶16-18.

4. The Combination of Bryant and Bach Is Based on Impermissible Hindsight.

As discussed in the Introduction, the skilled person guided by Bryant and Bach would not have reached the claimed invention for at least the following reasons. First, Bryant teaches away from copy-on-write as stated above. Second, computer scientists at the time did not think about using copy-on-write technology with process cloning for Java virtual machines. (Goldberg Declaration, ¶18.) Rather, as evidenced by the cited references, other schemes were used to manage memory for multiple Java virtual machines or multiple Java processes. Thus, rather than cast its mind back to the time of invention to be guided by what the references teach, the Office has simply used the claimed elements as a blueprint or road map to find them in the prior art. This is impermissible. (MPEP 2145(X)(A).) *Interconnect Planning Corp. v. Feil*, 774 F.3d 1132, 1139.

B. The Rejection Regarding the Dependent Claims Should Be Withdrawn

The dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. Claims 4 and 13 directed to a **"class resolver"** are patentable because Bryant merely discloses that classes are preloaded and does not disclose that the classes are

resolved. Claims 6 and 15 directed to a **"process cloning mechanism"** are patentable at least by virtue of the patentability of their independent claims. Claims 21 and 22 directed to a **"resource controller"** are also patentable because Bryant's disclosure of the child Java server receiving information though a pipe connection from the Java server has nothing to do with a resource controller **"to set operating system level resource management parameters."** Bryant would not have been combined with Bach to correct its deficiencies for at least the reasons described above.

VI. The Obviousness Rejection Based on the Combination of Bryant and Traut Should Be Withdrawn (Ground 5)

The rejection based on Bryant and Traut is cumulative of the rejection based on Bryant and Bach. As with Bach, the Office cites Traut for the **"copy-on-write process cloning mechanism."** (Action at 32.) The Bryant and Traut combination does not render obvious the claimed invention for the reasons described below. Bryant and Traut are not combinable because: (a) Bryant teaches away from the combination regarding copy-on-write cloning; (b) the combination changes the principle of operation of Bryant regarding how the fork system call is implemented; (c) the reasons for the combination are insufficient to support a *prima facie* case of obviousness; (d) the combination is based on impermissible hindsight that uses the '720 Patent as a road map to find these elements in the prior art; (e) Traut teaches away from the **"copy-on-write process cloning mechanism"** and toward a copy-on-access cloning mechanism for its child virtual machine; and (f) Traut and Bryant's virtual machines are too different to reasonably teach the skilled person how to transfer Traut's teachings to Bryant's virtual machine. The combination also does not disclose the dependent claims.

A. There Would Have Been No Reason for the Bryant-Traut Combination

In addition to the reasons described above regarding Bryant and Bach, Traut has the following additional deficiencies that prevent Bryant and Traut from being combinable.

1. Traut Teaches Away from the Claimed Copy-On-Write Process Cloning Mechanism

Traut teaches away from using a **"copy-on-write process cloning mechanism"** when a child process modifies a memory space of a parent process. While Traut generally describes the copy-on-write technique, Traut describes its own system as using copy-on-write when the parent process modifies a page, not when the child process does so. (¶¶ [0026], [0030].) Rather, Traut describes the child using an advantageous copy-on-access technique to get a copy of a page as soon as the child accesses the page. (¶¶ [0031], [0034]-[0035]; FIG. 3.) It is clear that Traut leads the skilled person to using copy-on-access for its child virtual machine and, therefore, away from using copy-on-write for the child virtual machine. This is because Traut's goal is to copy all memory to the child either immediately or over time (¶[0032], [0039]) and copy-on-access achieves that goal. This is fundamentally different from the '720 Patent, in which the goal is to copy as needed. (Goldberg Declaration, ¶20.) Had Traut intended that copy-on-write be used with the child virtual machine, Traut would certainly have done so, particularly in view of the parent virtual machine using copy-on-write.

2. The Virtual Machines of Bryant and Traut Are Very Different

Traut's virtual machine is very different from Bryant's Java virtual machine. Traut's virtual machine emulates a hardware machine (¶[0006]), whereas Bryant's Java virtual machine emulates a software machine. (1:61-65.) As an example, Traut's virtual machine interprets machine code, whereas Bryant's Java virtual machine interprets Java bytecodes. The skilled person looking to implement a **"copy-on-write process cloning mechanism"** in a Bryant Java virtual machine would not have looked to Traut to do so. (Goldberg Declaration, ¶21.)

B. The Rejection Regarding the Dependent Claims Should Be Withdrawn

For the reasons that claims 4, 6, 13, 15, 21, and 22 are not disclosed in Bryant and Bach, they are also not disclosed in Bryant and Traut. The remaining dependent claims are patentable over the combination for at least the same reasons as their respective independent claims.

VII. The Obviousness Rejection Based on the Combination of Srinivasan and Bach Should Be Withdrawn (Ground 6)

The combination of Srinivasan and Bach does not render obvious the claimed invention for the reasons described below. Srinivasan is basically a programming manual to show the skilled person how to program various discrete concepts in Perl, a scripting language. Srinivasan and Bach are not combinable because: (a) the combination fails to disclose any of the claimed elements, as the combination merely discloses simple Perl program code that describes how to create and use a Perl package, without disclosing the base element, i.e., the **"class preloader,"** and hence the remaining claimed elements devised therefrom; and (b) the reasons for the combination are insufficient to provide a *prima facie* case of obviousness. Even if the claimed elements were added to Srinivasan, the skilled person would have no idea how to cobble them together to achieve the claimed invention, except by impermissibly using the '720 Patent as a road map. These disparate sections mark a sum total of 10 pages scattered throughout different and sometimes "stand alone" chapters of a 400-page manual, with no teaching or suggestion how each of the separate elements should be used together. The combination also does not disclose the dependent claims.

A. The Combination of Srinivasan and Bach Does Not Disclose or Suggest the Claimed Invention

As described in detail below, Srinivasan is directed to Perl programming code. This renders Srinivasan and its combination with Bach deficient on numerous grounds.

1. The Combination Does Not Disclose the Claimed Class Preloader

Claim 1 recites, *inter alia*:

> a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code. (emphasis added.)

The Office asserts that this element is disclosed in Srinivasan, pp. 389-390, 98, 99, 101, 323-324, 370. (Action at 34.) On the contrary, these sections of Srinivasan do not disclose or suggest a **"class preloader to obtain a representation of at least one class from a source definition**

provided as object-oriented program code." (Goldberg Declaration, ¶22.) Srinivasan, pp. 389-390, merely discloses an example of Perl program code used to create and use an object package. Srinivasan does not disclose that the Perl program code includes **"a representation of at least one class from a source definition provided as object-oriented program code**." Neither does Srinivasan disclose that the Perl program code includes a **"class preloader."**

Srinivasan, p. 98, discloses differences between a Perl package and a Java package; p. 99, that Perl can be used to build objects; and p. 101, that Perl objects of certain types can belong to a class. These sections have nothing to do with a **"class preloader."** The Perl package and object include Perl program code, not **"a representation of at least one class from a source definition provided as object-oriented program code."**

Srinivasan, pp. 323-324, discloses that a Perl translator can convert a Perl script into opcodes that can be executed by a virtual machine. Srinivasan, p. 370, discloses that a Perl compiler can translate a Perl script into C code. These sections of Srinivasan merely disclose various uses of a Perl script by components of a Perl system, but provide no disclosure of a **"class preloader."**

2. The Combination Does Not Disclose the Claimed Master Runtime System Process

Srinivasan and Bach are deficient as to the next element that claim 1 recites, as well:

> a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process. (emphasis added.)

The Office asserts that this element is disclosed in Srinivasan, pp. 193-194, 321, 323. (Action at 35.) But this assertion fails because Srinivasan does not disclose the **"class preloader"** and the **"representation"** obtained thereby, as described above. (Goldberg Declaration, ¶22.) Rather, Srinivasan, pp. 193-194, merely shows example Perl program code that executes a Unix fork system call. However, this section of Srinivasan has nothing to do with a **"representation as a class definition."** The parent "environment" and "open file descriptors" in the parent's memory space are not a **"representation as a class definition."**

Srinivasan, pp. 321, 323, discloses multiple Perl interpreters. However, this section of Srinivasan does not disclose how the multiple interpreters are created or that any of the interpreters is a **"master"** interpreter **"to interpret and to instantiate the representation as a class definition in a memory space"** of that interpreter. The interpreters' main and loaded packages are not the **"memory space"** contents. Because the above elements are absent from Srinivasan, the **"master runtime system process"** that recites them is also absent.

3. The Combination Does Not Disclose the Claimed Runtime Environment

Srinivasan and Bach are further deficient regarding this element of claim 1, which recites:

> a runtime environment to clone the memory space [of the master runtime system process having an interpreted and instantiated representation of a class definition provided as object-oriented program code] as a child runtime system process request and to execute the child runtime system process. (emphasis added.)

Though not entirely clear from the Action, the Office appears to assert that this element is disclosed in Srinivasan, pp. 193-195. (Action at 35.) But this assertion similarly fails because Srinivasan does not disclose the **"class preloader,"** the **"master runtime system process,"** and its **"memory space,"** e.g., its memory space contents, as described above. (Goldberg Declaration, ¶22.) As stated above, this section of Srinivasan discloses Perl program code to execute a fork system call. It has nothing to do with a **"class definition"** that would be contained in the forked memory space. That is, this section of Srinivasan does not disclose that the program code **"clone[s] the memory space [of the master runtime system process having an interpreted and instantiated representation of a class definition]."** Instead, the program code shows forking a process containing a socket, as written in Perl. Because the above elements are not disclosed by Srinivasan, the **"runtime environment"** that recites them is not disclosed.

4. The Combination Does Not Disclose the Claimed Copy-On-Write Process Cloning Mechanism

Srinivasan and Bach are also deficient regarding this last element of claim 1, which recites:

> a copy-on-write process cloning mechanism to instantiate the child

> runtime system process by copying references to the memory space of the master runtime system process [having an interpreted and instantiated representation of a class definition provided as object-oriented program code] into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. (emphasis added.)

The Office asserts that this element is disclosed in Srinivasan, pp. 193-195, and Bach, pp. 192, 287, 289-290. (Action at 36.) But this assertion similarly fails because Srinivasan does not disclose the **"class preloader,"** the **"master runtime system process,"** its **"memory space,"** e.g., its memory space contents, and the **"runtime environment,"** as described above. (Goldberg Declaration, ¶22.) As stated above, this section of Srinivasan discloses Perl program code to execute a fork system call. It has nothing to do with a **"class definition"** that would be contained in the forked memory space. That is, this section of Srinivasan does not disclose the **"memory space of the master runtime system process [having an interpreted and instantiated representation of a class definition]."** Instead, the program code shows forking a process containing a socket, as written in Perl. This deficiency is not corrected by Bach because Bach is directed to the forking process, not to the contents of the forked memory space. Because the above elements are absent from the combination, the **"copy-on-write process cloning mechanism"** that recites them is also absent.

B. The Combination Is Based on Impermissible Hindsight Reasoning

The cited sections of Srinivasan are found in a sum total of 10 pages throughout different and sometimes "stand alone" chapters of the 400-page manual. They describe disparate, discrete Perl concepts and do not indicate to the skilled person how or why they should be brought together in the manner the Action, at 34-37, asserts. The leap that would have to be made to cobble together these sections could not be done without guidance from the '720 Patent. (Goldberg Declaration, ¶23.) For example, the Action picks piecemeal from a Srinivasan section that describes Perl syntax (e.g., Appendix B, pp. 389-390), a section that describes object orientation (e.g., Chapter 7, pp. 99,

101), and a section that describes a Perl system (e.g., Chapter 19, pp. 323-324) in an attempt to find a disclosure of the **"class preloader."** The Action similarly picks from a Srinivasan section that describes Perl program code for networking (e.g., Chapter 12, pp. 193-194) and a section that describes a Perl system (e.g., Chapter 19, pp. 323-324) in an attempt to find a disclosure of the **"master runtime system process."** Similar attempts are made regarding the **"runtime environment"** and the **"copy-on-write process cloning mechanism"** as well. Yet Srinivasan provides no guidance on which sections to pick and why.

Moreover, even if the **"class preloader," "master runtime system process," "runtime environment,"** and **"copy-on-write process cloning mechanism"** were added to Srinivasan, the skilled person could only know to combine the cited sections to achieve the claimed invention from Patent Owner's disclosure. Such knowledge comes from impermissible hindsight and is not allowed as the basis for rejection. The Office cannot use the claimed invention as a blueprint to pick and choose from the prior art without some guidance therefrom about how to put the chosen pieces together.

C. Requester's Reasons to Combine Are Insufficient

Given the above discussion, it is clear that Requester's reasons for the combination of Srinivasan and Bach, adopted by the Office, are insufficient to establish a *prima facie* case of obviousness. The Action, at 37, states that Srinivasan and Bach could be combined "to further streamline the impact on system memory."

Given that copy-on-write technology was known as early as 1986, when Bach was published and eleven years before Srinivasan was published, it clearly was not obvious to use copy-on-write "to further streamline the impact on system memory" in Perl systems. Otherwise, Srinivasan would have done so. Moreover, there is no disclosure in either Srinivasan or Bach to suggest that the fork system call "commonly used in conjunction with the copy-on-write mechanism" could be used **"to clone the memory space of a master runtime system process [having an interpreted and**

instantiated representation of a class definition provided as object-oriented program code]." See also Goldberg Declaration, ¶¶22-23.

D. The Rejection Regarding the Dependent Claims Should Be Withdrawn

The dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. For example, claims 4 and 13 directed to a **"class resolver"** are patentable because Srinivasan fails to disclose the **"class preloader"** that obtains a **"class definition"** and therefore cannot disclose the **"class resolver"** to resolve the absent class definition. Claims 6 and 15 directed to a **"process cloning mechanism"** are also patentable because Srinivasan fails to disclose the **"memory space,"** e.g., its interpreted and instantiated representation of a class definition, as described above, and therefore cannot disclose the **"process cloning mechanism"** that copies the absent memory space. Claims 21 and 22 directed to a **"resource controller"** are similarly patentable because Srinivasan does not disclose the requisite **"child runtime system process,"** as described above, on which the **"resource controller"** operates. The deficiencies of Srinivasan are not corrected by Bach for at least the reasons described above.

VIII. The Obviousness Rejection Based on the Combination of Sexton and Bugnion Should Be Withdrawn (Ground 7)

The combination of Sexton and Bugnion does not render obvious the claimed invention for the reasons described below. The combination does not disclose all of the claimed elements, in particular, (a) the **"runtime environment"** because Sexton, cited for this element, lacks the requisite cloning, i.e., **"to clone,"** in a **"runtime environment"**; and (b) the **"copy-on-write process cloning mechanism"** because Bugnion, cited for this element, lacks the requisite copy-on-write cloning. Additionally, Sexton and Bugnion are not combinable because: (a) Sexton teaches away from the combination regarding copy-on-write; (b) the combination changes the principle of operation of Sexton from segmented memory to copy-on-write memory; and (c) the reasons for the combination are insufficient to support a *prima facie* case of obviousness. The combination also fails to disclose the dependent claims.

A. The Combination of Sexton and Bugnion Does Not Disclose or Suggest the Claimed Invention

1. The Combination Does Not Disclose the Claimed Runtime Environment

Claim 1 recites, *inter alia*:

> a runtime environment to clone the memory space [of the master runtime system process] as a child runtime system process responsive to a process request and to execute the child runtime system process. (emphasis added.)

The Office asserts that Sexton discloses this element in 7:61-8:18. (Action at 43.) Patent Owner submits that the Office has mistakenly equated "spawned" with **"to clone."** Sexton clearly describes "spawned" to mean instantiation, in which a virtual machine instance is created from a virtual machine class (i.e., a "VM data structure"). (8:3-5.) Sexton does not disclose that the "spawned" Java virtual machines are either all clones of an undesignated master Java virtual machine, or one of the virtual machines is the undesignated master and the others are clones. Sexton does not disclose virtual machine cloning. (Goldberg Declaration, ¶24.)

These deficiencies are not corrected by Bugnion. Bugnion also does not describe virtual machine cloning. Rather, Bugnion merely describes the use of multiple virtual machines (8:56-65; FIG. 1) without describing how the virtual machines are created. (Goldberg Declaration, ¶24.)

Because the combination does not disclose the above elements, the **"runtime environment"** that recites them is also not disclosed.

2. The Combination Does Not Disclose the Claimed Copy-On-Write Process Cloning Mechanism

Sexton and Bugnion are further deficient as to this element of claim 1, which recites:

> a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. (emphasis added.)

The Office asserts that Bugnion discloses this element in 6:29-36; 14:55-15:35; 15:66-16:1. (Action at 44.) On the contrary, while Bugnion discloses each virtual machine referencing shared data, e.g., the code and buffer cache, in the machine memory and the machine memory having copy-on-write capabilities (14:55-15:25; FIG. 4), Bugnion does not disclose that any of the virtual machines is the result of cloning. (Goldberg Declaration, ¶25.) That is, Bugnion does not disclose the **"master runtime system process"** that owns the machine memory or the **"child runtime system process"** that is cloned and that references and/or needs to modify the memory space of another virtual machine.

Bugnion also does not disclose that the machine memory is the **"memory space"** of any one of the virtual machines. The machine memory is an independent shared memory that each virtual machine can reference (or map to), but not own. (6:29-36.) Neither is a virtual machine's physical memory the claimed **"memory space"** because it is not accessible by any other virtual machine for referencing thereto and/or copying therefrom. (FIG. 4.)

Bugnion further does not disclose the **"copy-on-write process cloning mechanism."** Bugnion merely discloses using copy-on-write disks for machine memory, with no indication that the copy-on-write disks are associated with cloning of the virtual machines. (14:55-15:35; 15:66-16:1.)

Because Bugnion does not disclose the above elements, Bugnion also does not disclose the **"copy-on-write process cloning mechanism"** that recites them. This deficiency is not corrected by Sexton, as acknowledged by the Action. (Action at 44.)

B. There Would Have Been No Reason for the Sexton-Bugnion Combination

Sexton and Bugnion are not properly combinable for at least the following reasons.

1. Sexton Teaches Away from the Combination

Sexton discloses a segmented memory model for the virtual machines, where session and call memories of each virtual machine are set up to store data to be modified by the virtual machine,

and shared memory is set up to store data not to be modified by any virtual machine. (6:59-67; 7:1-11, 31-45; FIGs. 2, 3.) It is clear that Sexton leads the skilled person to provide individual memories, e.g., session memories, upon instantiation of a virtual machine so as to store data that the virtual machine intends to modify and, therefore, away from a shared memory that uses copy-on-write to store such data. This is fundamentally different from Bugnion, which initially sets up all the data (modifiable and unmodifiable) in a global machine memory. Sexton did not consider sharing modifiable data between virtual machines, having set up the session and call memories. Accordingly, Sexton teaches away from the combination. (Goldberg Declaration, ¶26.)

2. The Combination Changes the Principle of Operation of Sexton

Modifying Sexton to implement copy-on-write would change the principle of operation of Sexton. The modification would replace a segmented memory model having a read-only shared memory with a copy-on-write shared memory. (Goldberg Declaration, ¶27.)

3. Requester's Reasons to Combine Are Insufficient

Given the above discussion, it is clear that Requester's reasons for the combination of Sexton and Bugnion, which were adopted by the Office, are insufficient to establish a *prima facie* case of obviousness. The Action, at 46, states that Sexton and Bugnion could be combined for "only copying necessary files."

Patent Owner disagrees. First, the combination does not disclose all of the claimed elements, as described above. Second, even if the **"runtime environment"** and **"copy-on-write process cloning mechanism"** were added to Sexton or Bugnion, the skilled person reading Sexton would not consider modifying Sexton to include copy-on-write because Sexton has already set up a memory model to avoid unnecessary copying by initializing a shared memory area to store "necessary files" of Java classes. There need not be any further copying because the session memories store any data specific to that session's virtual machine. Sexton has reduced the session memory to the extent necessary, such that copy-on-write would be ineffective and unneeded.

Given that copy-on-write technology was known as early as 1986, when Bach was published and fourteen years before Sexton was filed as a patent application, it clearly was not obvious to use copy-on-write for "reducing session memory by sharing data between multiple Virtual Machines" to "only copying necessary files." Otherwise, Sexton would have done so. See also Goldberg Declaration, ¶¶24-27.

C. The Rejection Regarding the Dependent Claims Should Be Withdrawn

The dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. Claims 4 and 13 directed to a **"class resolver"** are patentable because Sexton, cited for this element, merely discloses that class data is shared and does not disclose that classes are resolved. Claims 6 and 15 directed to a **"process cloning mechanism"** are also patentable because neither Sexton nor Bugnion discloses this element. Sexton does not disclose its Java virtual machines being the result of cloning, but merely the result of instantiating a virtual machine data structure. Bugnion does not disclose how its virtual machines are created. Claims 21 and 22 directed to a **"resource controller"** are patentable at least by virtue of the patentability of their independent claims.

IX. The Obviousness Rejection Based on the Combination of Sexton and Johnson Should Be Withdrawn (Ground 8)

The rejection based on Sexton and Johnson is cumulative of the rejection based on Sexton and Bugnion. As with Bugnion, the Office cites Johnson for the **"copy-on-write process cloning mechanism."** (Action at 54.) The Sexton and Johnson combination does not render obvious the claimed invention for the reasons described below. The combination does not disclose all of the claimed elements, in particular, (a) the **"runtime environment"** because Sexton, cited for this element, lacks the requisite cloning, i.e., **"to clone,"** in a **"runtime environment"**; and (b) the **"copy-on-write process cloning mechanism"** because Johnson, cited for this element, lacks the requisite copy-on-write that is applied to the entirety of the child process. Apart from not disclosing all the claimed elements, Sexton and Johnson are not combinable because: (a) Sexton teaches away

from the combination regarding copy-on-write; (b) the combination changes the principle of operation of Sexton from segmented memory to copy-on-write memory; and (c) the reasons for the combination are insufficient to support a *prima facie* case of obviousness. The combination also fails to disclose the dependent claims.

A. The Combination of Sexton and Johnson Does Not Disclose or Suggest the Claimed Invention

1. The Combination Does Not Disclose the Claimed Runtime Environment

Claim 1 recites, *inter alia*:

> a runtime environment to clone the memory space [of the master runtime system process] as a child runtime system process responsive to a process request and to execute the child runtime system process. (emphasis added.)

The deficiencies of Sexton as described above are not corrected by Johnson because Johnson also does not describe virtual machine cloning. (Goldberg Declaration, ¶28.) None of Johnson's virtual machines acts as a **"master"** from which the other virtual machines are cloned. Johnson merely describes the use of multiple virtual machines (18:41-44), without any description of how the multiple virtual machines are created.

Because the combination does not disclose the above elements, the **"runtime environment"** that recites them is also not disclosed.

2. The Combination Does Not Disclose the Claimed Copy-On-Write Process Cloning Mechanism

Sexton and Johnson are further deficient as to this element of claim 1, which recites:

> a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. (emphasis added.)

The Office asserts that Johnson discloses this element in 18:34-44. (Action at 54.) On the

contrary, while Johnson discloses storage having copy-on-write technology (18:34-63), Johnson describes copy-on-write with respect to a particular type of variable, i.e., static variables. This is fundamentally different from the **"copy-on-write process cloning mechanism"** used in the '720 Patent, in which copy-on-write is applied seamlessly to the entirety of the child process. (Goldberg Declaration, ¶29.) Even if the **"copy-on-write process cloning mechanism"** were to replace Johnson's copy-on-write, Johnson appears to describe copy-on-write not being used with respect to the static variables. For example, where a Java virtual machine has multiple class instances that share static variables, any one of the instances can modify the variables without making a separate copy. (18:33-63.) Similarly, multiple Java virtual machines have their own sets of static variables, rather than a shared set, that can be modified without affecting each other. (*Id.*)

Johnson also does not disclose that any of the Java virtual machines is the result of **"process cloning"** where any of the Java virtual machines is **"the master runtime system process"** that has the static variables in its memory space or **"the child runtime system process"** that is cloned and that references and/or needs to modify the memory space of another of the virtual machines. Johnson merely discloses that multiple Java virtual machines can be used (18:41-44), not how the virtual machines are created.

Because Johnson fails to disclose the above elements, Johnson also fails to disclose the **"copy-on-write process cloning mechanism"** that recites them. This deficiency is not corrected by Sexton, as acknowledged in the Action at 44, 53.

B. There Would Have Been No Reason for the Sexton-Johnson Combination

Similar to the reasons provided above for Sexton and Bugnion, Sexton and Johnson are also not properly combinable because (a) Sexton teaches away from such a combination, (b) the combination would impermissibly change the principle of operation of Sexton, and (c) the reasons for combination are insufficient to support a *prima facie* case of obviousness. See also Goldberg Declaration, ¶¶28-29.

C. The Rejection Regarding the Dependent Claims Should Be Withdrawn

For the reasons that the Sexton-Bugnion combination does not disclose claims 4, 6, 13, 15, 21, and 22, the combination of Sexton and Johnson also does not disclose these claims. The remaining dependent claims are patentable over the combination for at least the same reasons as their respective independent claims.

X. The Rejections Should Be Withdrawn Because Overwhelming Evidence of Secondary Considerations Demonstrates the Nonobviousness of the Claimed Invention

In addition to all of the reasons set forth above, the '720 Patent is not obvious because of substantial objective evidence of secondary considerations. When evaluating the nonobviousness of a claimed invention, such evidence must be considered, and "may often be the most probative and cogent evidence in the record." *Stratoflex, Inc. v. Aeroquip Corp.*, 713 F.2d 1530, 1538 (Fed. Cir. 1983); *Transocean Offshore Deepwater Drilling, Inc. v. Maersk Contractors USA, Inc.*, 2010 U.S. App. LEXIS 17181 at *14-*15 (Fed. Cir. August 18, 2010). Here, Patent Owner submits that the evidence presented below strongly supports a conclusion of nonobviousness.

A. The Claimed Invention Was Requested by Customers

Patent Owner developed a commercial embodiment of the '720 Patent called Connected Device Configuration-Application Management System (CDC AMS). CDC AMS is a distributed system for launching and managing multiple applications that includes, *inter alia*, virtual machine instances, each application having its own instance, and copy-on-write technology for memory management of the virtual machine instances. (Appendix C, Runtime Guide at 5-1 through 5-12; Appendix B, Porting Guide at 12-1 through 12-12.) Patent Owner also presented CDC AMS at JavaOne, a premier Java conference. (Appendix J, JavaOne Presentation.)

Some customers requested a product embodying the claimed invention, e.g., CDC AMS, because of the performance and memory efficiencies therein. These customers wanted memory usage reduction, as provided by the **"copy-on-write process cloning mechanism,"** for such applications as multi-task printing, e-reader technology, and VoIP telephony, because this reduction

was a good fit for their multi-process environments.

B. The Claimed Invention Satisfied a Long-Felt Need

Since the implementation of Java virtual machines, there has been a need among system developers to have efficient use of memory between multiple virtual machine processes, while providing a robust environment for executing the multiple virtual machine processes concurrently. (Goldberg Declaration, ¶30.) (Appendix F, Shared Memory Blog; Appendix G, Reduced Footprint Blog.) The naïve approach that was developed involved starting multiple instances of a Java virtual machine, each process having and executing its own virtual machine instance and each virtual machine instance having a distinct address space that is physically separate from that of other virtual machines.

There were several drawbacks to this approach, mainly due to the dynamic nature of Java virtual machines. When each process got its own virtual machine, the initialization cost was repeated for each virtual machine, there was no shared memory between the processes, and common libraries were duplicated among processes. As a result, memory usage was inefficient. (Goldberg Declaration, ¶30.)

A later approach was developed to reduce the memory footprint associated with running concurrent Java virtual machines. In this approach, the virtual machines shared some memory between the processes, while maintaining individual memory spaces. This approach had its own drawbacks. Initialization costs were still repeated for each virtual machine and the individual memory spaces were not always necessary. Memory usage improved, but was still inefficient. (*Id.*) See also, Appendix L, Kawachiya Paper.

As smaller devices having limited or otherwise constrained memory resources emerged, these approaches were no longer viable because their memory usage requirements were greater than the device memory resources available. (Goldberg Declaration, ¶30.)

Recognizing this unsatisfied need for efficient memory usage, Patent Owner developed and patented a new approach to virtual machine memory management, as in the claimed invention of the '720 Patent, that used copy-on-write with process cloning to share memory between a master virtual machine and a cloned virtual machine until the cloned virtual machine needed to modify the shared memory. As a result, Patent Owner's approach was well suited for smaller devices having limited memory resources because it shared common libraries between processes, it did not have to repeat initialization costs for each cloned virtual machine, and it shared memory between processes by default. (*Id.*, ¶31.)

C. The Claimed Invention Was Copied by Google

The desirable functionalities of Patent Owner's claimed invention did not go unnoticed by Patent Owner's would-be competitors, which now include Google, the Requester of this reexamination proceeding. As asserted in Patent Owner's patent infringement complaint, Google copied Patent Owner's claimed invention, presumably in order to come up with its competing Android software without having to invest the substantial time and resources that Patent Owner did in the claimed invention. (Appendix H, Complaint at 5-6, 8-9.)

Google's own public disclosures, e.g., websites,[3] presentation videos and slides,[4] and open source code,[5] describe Android software that has the functionality of the '720 Patent, further evidencing copying of the claimed invention. Indeed, a side-by-side comparison of Patent Owner's CDC AMS description and Google's Android description and source code, with respect to claim 1 of the '720 Patent, demonstrates copying by Google. (Appendix J, Copy Chart.)

---

[3] See Appendix I, Zygote. See also, generally, *http://www.developer.android.com*.

[4] See Appendix D, Android Presentation, and corresponding Video, "Google I/O 2008 – Anatomy and Physiology of an Android," by Patrick Brady, *http://developer.android.com/videos/index.html* (follow "Google I/O Sessions" tab; follow "Google I/O 2008 – Anatomy and Physiology of an Android" hyperlink; last visited June 29, 2011). See also Appendix E, Dalvik Presentation, and corresponding Video, "Google I/O 2008 – Dalvik Virtual Machine Internals," by Dan Bornstein, *http://developer.android.com/videos/index.html* (follow "Google I/O Sessions" tab; follow "Google I/O 2008 – Dalvik Virtual Machine Internals" hyperlink; last visited June 29, 2011).

[5] See Appendix J, Copy Chart. See also, generally, *http://android.git.kernel.org*.

XI. Conclusion

For at least these reasons, Patent Owner requests reconsideration and withdrawal of the rejections in the Action and confirmation of the patentability of claims 1-8, 10-17, and 19-22 of the '720 Patent.

In the event that the Office determines that relief or fees (such as payment of a fee under 37 C.F.R. § 1.17 (g)) are required, the Patent Owner petitions for any required relief and authorizes the Commissioner to charge the cost of such petition and/or other fees due in connection with the filing of this document to **Deposit Account No. 03-1952** referencing **154892800700**.

Dated: July 5, 2011

Respectfully submitted,

Electronic Signature: /Cassandra T. Swain/
Cassandra T. Swain
Registration No.: 48,361
MORRISON & FOERSTER LLP
755 Page Mill Road
Palo Alto, California 94304-1018
(650) 813-4295

Electronic Signature: /Christopher B. Eide/
Christopher B. Eide
Registration No.: 48,375
MORRISON & FOERSTER LLP
755 Page Mill Road
Palo Alto, California 94304-1018
(650) 813-5720