



(51) International Patent Classification:  
G06F 9/50 (2006.01)

(21) International Application Number:  
PCT/US2019/067560

(22) International Filing Date:  
19 December 2019 (19.12.2019)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **GOOGLE LLC** [US/US]; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(72) Inventors: **LEVIN, Alex**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **BROCH, Todd Alan**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(74) Agent: **COLBY, Michael K.**; 291 E. Shore Drive, Suite 200, Eagle, Idaho 83616 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME,

MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

**Declarations under Rule 4.17:**

— as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))

**Published:**

— with international search report (Art. 21(3))

(54) Title: RESOURCE MANAGEMENT UNIT FOR CAPTURING OPERATING SYSTEM CONFIGURATION STATES AND MEMORY MANAGEMENT

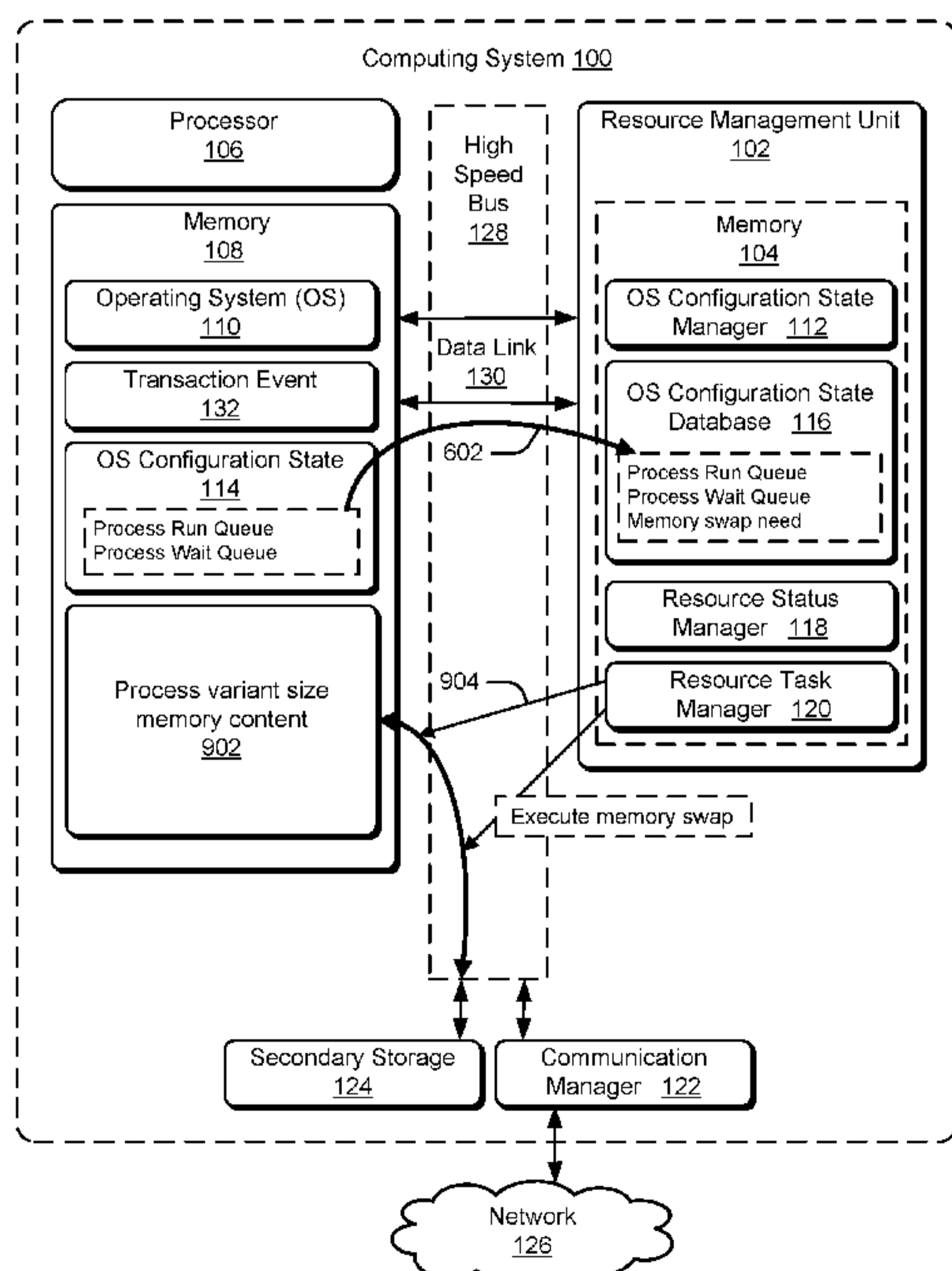


FIG. 9

(57) Abstract: This disclosure describes methods, devices, systems, and procedures in a computing system (100; 300) for capturing (112; 116) a configuration state (114) of an operating system (110) executing on a central processing unit (CPU) (106), and offloading memory management tasks, based on the configuration state, to a resource management unit (102) such as a system-on-a-chip (SoC). The resource management unit identifies a status of a resource (118) requiring memory swapping based on the captured configuration state of the operating system. The resource management unit then swaps the memory (902; 904) to alleviate the CPU from processing the swap thereby improving overall computing system performance.

## **RESOURCE MANAGEMENT UNIT FOR CAPTURING OPERATING SYSTEM CONFIGURATION STATES AND MEMORY MANAGEMENT**

### **BACKGROUND**

[0001] Computer systems generally include a central processing unit (CPU), memory, input/output (I/O) devices, and other electronic components, as well as software, firmware, and data. Operating system (OS) software stored in memory and executed on the CPU manages the computer system's core functions, such as scheduling tasks, executing software applications, managing memory, and controlling I/O and peripheral devices. Additional hardware or other integrated circuit components, such as a memory management unit (MMU), may be included on a motherboard to offload processing tasks from the CPU in effort to improve performance of the system. These efforts to improve performance of the system with additional hardware or other components, however, often fail to effectively or sufficiently improve system performance.

### **SUMMARY**

[0002] This document describes an example computing system and method that leverages a resource management unit for offloading memory management tasks from a central processing unit (CPU). The resource management unit captures a configuration state of an operating system (OS) executing on the CPU, identifies opportunities for memory management, and then, based on the captured configuration state and the identified opportunities, offloads memory swapping tasks from the CPU to the resource management unit.

[0003] The computing system includes the CPU, the resource management unit, and memory having instructions that, responsive to execution by the CPU or the resource management unit, cause the resource management unit to capture the configuration state of the operating system, identify the opportunity for memory management, and execute a memory swap. In other aspects, the computing system includes a hypervisor or virtual machine, and the resource management unit captures



the configuration state of the operating system of the hypervisor or virtual machine, identifies an opportunity for memory management, and executes a memory swap. In additional aspects, the resource management unit may swap using a variant size of the content in the memory without a set page-size (*e.g.*, block-size) limitation.

[0004] The details of one or more methods, devices, systems, and procedures for capturing operating system configuration states with a resource management unit for identifying memory management opportunities and executing memory swapping tasks are set forth in the accompanying drawings and the following description. Other features and advantages will be apparent from the description, drawings, and claims. This summary is provided to introduce subject matter that is further described in the detailed description and drawings. Accordingly, this summary should not be considered to limit the scope of the claimed subject matter.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

[0005] The details of one or more aspects of a resource management unit for capturing operating system configuration states and memory management are described with reference to the following drawings. The use of the same reference numbers in different instances in the description and the figures may indicate like elements.

[0006] FIG. 1 is a block diagram illustrating an example computing system configured to capture operating system configuration states for offloading tasks to a resource management unit (*e.g.*, scaling clock rates, memory management, processing, managing vulnerabilities).

[0007] FIG. 2 is a block diagram illustrating another example computing system, including a hypervisor and virtual machine, configured to capture operating system configuration states for offloading tasks to a resource management unit.

[0008] FIG. 3 illustrates an example device diagram for a user equipment that can implement various aspects of capturing operating system configuration states for offloading tasks to a resource management unit.

[0009] FIG. 4 illustrates an example method for capturing operating system configuration states for offloading tasks to a resource management unit in a computing system.

[0010] FIG. 5 illustrates an example method with additional detail for capturing operating system configuration states for offloading tasks to a resource management unit in a computing system.

[0011] FIG. 6 is a block diagram illustrating an example computing system depicting an example operating system configuration state as captured by a resource management unit for offloading operating system tasks to the resource management unit.

[0012] FIG. 7 is a block diagram illustrating an example computing system configured for capturing operating system configuration states by a resource management unit for dynamically scaling clock rates.

[0013] FIG. 8 illustrates an example method for capturing operating system configuration states by a resource management unit in a computing system for dynamically scaling clock rates.

[0014] FIG. 9 is a block diagram illustrating an example computing system configured for capturing operating system configuration states by a resource management unit for performing memory management tasks.

[0015] FIG. 10 illustrates an example method for capturing operating system configuration states by a resource management unit in a computing system for managing memory swapping.

[0016] FIG. 11 is a block diagram illustrating an example computing system configured for capturing operating system configuration states by a resource management unit for detecting and managing malware and software vulnerabilities.

[0017] FIG. 12 illustrates an example method for capturing operating system configuration states by a resource management unit in a computing system for detecting and managing malware and software vulnerabilities.

### **DETAILED DESCRIPTION**

[0018] This disclosure describes methods and systems for capturing operating system configuration states to offload memory management tasks to a resource management unit. A computing system leverages a resource management unit, such as a system-on-a-chip (SoC), to capture a configuration state of an operating system (OS)

executing on a central processing unit (CPU) of the computing system. Based on the configuration state of the operating system, the resource management unit identifies a status of a process resource in the computing system requiring memory management. The resource management unit then executes the memory management, such as a memory swapping, to alleviate the CPU from processing the task, thus improving overall computing system performance. Given the resource management unit's view of the operating system configuration state and resources, the resource management unit may swap using a variant size of the content in the memory without a set page-size (*e.g.*, block-size) limitation.

**[0019]** This disclosure generally describes capturing an operating system configuration state with a resource management unit and, based on the captured state, offloading tasks. Various ways in which to offload tasks are described, including processing tasks, memory management, malware and vulnerability management, and dynamically scaling clock rates. While these examples are described in subsections below, these subsections are not intended to limit these examples to being only described in those subsections, their coexistence with other examples, or their independent operation.

#### CAPTURING OPERATING SYSTEM CONFIGURATION STATES AND OFFLOADING TASKS TO A RESOURCE MANAGEMENT UNIT

**[0020]** Hardware components of a computer system are often configured to assist an operating system or CPU with processing tasks. These processing tasks, however, remain managed by the operating system, which is executed through the CPU. Further, these components impose their own limitations on the operating system and CPU, such as application programming interfaces (APIs) in the form of cyclic buffers, memory mapped input/output (MMIO), shared-memory limitations, and sizing restrictions for swapping memory pages or I/O blocks. These limitations continue to burden the CPU, often causing delay or reduced performance of the computing system, even from a user's perspective. For example, as the operating system manages the system's resources, such as scheduling tasks, pausing execution of a process, allocating memory, swapping memory, starting execution of another process, and so forth, a user's input or

expected output in response to that input may be delayed. This may occur even if some of these functions are assigned to, and completed with the help of, additional hardware because the operating system remains responsible for assigning the tasks or executing certain functions related to these hardware components through the CPU, thus burdening the CPU.

**[0021]** In contrast, the resource management unit of this disclosure avoids these limitations by capturing the operating system configuration state, storing it into a database (*e.g.*, memory) associated with or accessible by the resource management unit, identifying from the database a status of resources in the system, and independently processing tasks associated with the resources. This independent processing by the resource management unit effectively alleviates the processor from having to perform the tasks thus improving overall system performance.

**[0022]** Although the described resource management unit can be an SoC, other hardware configurations may similarly be used, either alone or in combination with firmware or software, that provides comparable functionality to the resource management unit of this disclosure.

**[0023]** FIG. 1 is a block diagram illustrating an example computing system 100 having a resource management unit 102, *e.g.*, a system-on-a-chip (SoC) and associated memory 104, as well as a processor 106 (*e.g.*, the CPU) and its associated memory 108. The memory 104 and 108 are computer-readable storage media, and can include various implementations of random-access memory (RAM), read-only memory (ROM), flash memory, cache memory, and other types of storage memory in various memory device configurations, including a shared memory configuration. The memory 108 includes the operating system 110 and instructions that execute on the processor 106.

**[0024]** The resource management unit 102 includes an OS configuration state manager 112 configured to capture or obtain (used interchangeably herein) an OS configuration state 114 of the operating system 110. The OS configuration state 114 reflects a momentary state or status of the operating system at a given point in time. As an example, and for simplicity of this discussion, the OS configuration state 114 reflects a status of the operating system upon a resource transaction event 132 occurring in the



operating system 110. However, other events or times may similarly be selected to capture the OS configuration state.

**[0025]** The OS configuration state manager 112 captures and stores the OS configuration state 114 into an OS configuration state database (database) 116. The database 116 may be located in the memory 104 or any other fast memory associated with the resource management unit 102, either fabricated on or as part of the resource management unit or located elsewhere in the computing system 100 and accessible by the resource management unit 102 via low-latency access. The OS configuration state database 116 may be any standard database structure, such as a table, tree, hierarchical, relational (*e.g.*, MySQL, SQLite, Oracle, Sybase), NoSQL or object-oriented scheme, configured to capture indicators of the operating system configuration state as described in this disclosure.

**[0026]** The OS configuration state 114 captured into the OS configuration state database 116 may include information associated with a process run queue, a process wait queue, active devices, and virtual memory tables. These queues include process data relevant to specific processes in each queue. The process data includes data defining the status of that process as stored in a data structure, such as a process control block or similar structure, as described further below for defining and tracking processes and activity in operating systems. The virtual memory tables define the storage allocation scheme in which a secondary storage 124 (*e.g.*, memory) can be addressed as though it were part of the memory 108. Other example aspects of the operating system captured may include process priority data, process run-time remaining, resource scheduling data, resource usage data, memory usage data, memory mapping data, storage management data, active device data, a hypervisor status, a virtual machine (VM) status, a VM guest operating system status, or resources used by the hypervisor or virtual machine. Additional operating system aspects captured may be similar to those identified using an operating system analysis tool, such as a Linux operating system event-oriented observability tool, *e.g.*, “perf.” This allows for tracking performance counters, events, trace points, tasks, workload, control-flow, cache misses, page faults, and other profiling aspects of the operating system to provide a robust reference perspective of the operating system configuration state.

[0027] For purposes of this disclosure, a process is an instance of a program (*e.g.*, binary program file) in execution on a processor 106 (or a core or cores of the processor). A process may be defined by how it is referenced in memory. Examples of a process definition in memory may include elements of text (*e.g.*, compiled program code), data (*e.g.*, global and static variables), heap (*e.g.*, for dynamic memory allocation), stack (*e.g.*, local variables), and information related to program counters and registers. A state of a process (process state) in the operating system may include: (i) a new process being created, (ii) a ready process that has its resources ready but not yet running on the processor, (iii) a running process that is being executed by the processor, (iv) a waiting process that is not running but is waiting for an event to occur or resource to become available, and (v) a terminated process that has completed execution on the processor. Additionally, for each process there is a data structure, referenced as a process control block in this disclosure (*e.g.*, see FIG. 6), that stores the process-specific information. The process control block may include data indicative of: (i) the process identification (ID), (ii) the process state, (iii) processor registers and program counter (*e.g.*, for swapping the process in and out of the processor), (iv) processor scheduling information (*e.g.*, priority information), (v) memory management data (*e.g.*, page tables), (vi) accounting data (*e.g.*, processor time consumed, limits), and (vii) input/output status data (*e.g.*, devices used, open files).

[0028] The resource management unit 102 further includes a resource status manager 118 and a resource task manager 120. The resource status manager 118 identifies a status of a resource in the computing system, as stored in the database 116, based on the OS configuration state 114. For example, a status of a process is detected by referencing the process control block of that process, thereby detecting the process state, registers, scheduling, priority, memory management, and other process-related information. The resource task manager 120 processes a task associated with the status of the resource, as identified by the resource status manager 118, to alleviate the processor 106 from performing the task and thereby improving overall performance of the computing system 100. For example, if the process control block indicates a process is in a waiting state and a memory resource is needed, the resource task manager 120 processes a task of freeing memory so the process may move from the wait state to the



ready state for execution. Although the OS configuration state manager 112, resource status manager 118, and resource task manager 120 are shown in the diagram as being stored as executable instructions (*e.g.*, firmware or software) in the memory 104, the same may alternatively be formed in digital logic hardware blocks on the resource management unit 102, or enabled using a combination of firmware, software, and hardware blocks.

**[0029]** A communication manager 122 enables communications external to the computing system 100, such as to a networked computing environment, the Internet, or a cloud or multi-cloud network computing environment, generally referred to herein as a network 126. The resource management unit 102 and processor 106 together form a processor complex in the computing system 100, and the memory 104, 108 include data or instructions stored thereon to cause the processor complex 102, 106 to perform the methods and functions described throughout this document.

**[0030]** A secondary storage 124 provides additional storage (*e.g.*, memory) functionality for the computing system 100. Examples of the secondary storage 124 include non-volatile memory, fixed and removable media devices, and any suitable memory device or electronic data storage that maintains executable instructions and/or supporting data. The secondary storage 124 can include various implementations of RAM, ROM, or other types of non-volatile memory such as a solid-state drive (SSD) or a hard disk drive (HDD), or combinations thereof. The secondary storage 124 may be used for storing software applications or data, or for memory management purposes.

**[0031]** The resource management unit 102 may include other components not shown in this example, such as processing units (*e.g.*, central processing units, graphics processing units, artificial intelligence processing units, display processors, video processors), communication units (*e.g.*, modems), input/output controllers, and sensor hubs (see FIG. 3 for additional examples).

**[0032]** The resource management unit 102 and the processor 106 communicate using low-latency communication and write transactions, either by way of the resource management unit being implemented on a same integrated circuit die as the processor and its memory 108, or by using a high-speed interface bus 128 and data link 130. The high-speed bus 128 may be any fast bus standard that creates a seamless interface

between the resource management unit 102 and the processor 106, such as the Peripheral Component Interconnect Express bus, otherwise known as PCI Express or PCIe, or other similar open or proprietary bus standard with low-latency performance. The resource management unit 102 may also have a direct memory access (DMA) channel to the memory 108 via the data link 130 for transferring data to/from the memory 108 without passing it through the processor 106.

**[0033]** In one example, capturing an OS configuration state 114 of the operating system 110, in coordination with the OS configuration state manager 112 on the resource management unit 102, includes detecting resource transaction events 132 indicative of operational status information of the operating system and its managed resources. Examples of resource transaction events include events regarding a process run queue, a process wait queue, process priority data, process run-time remaining, resource scheduling data, resource usage data, active device data, memory usage data, memory mapping data, virtual memory tables, storage management data, a hypervisor status, a virtual machine (VM) status, a VM guest operating system, or resources used by the hypervisor or virtual machine.

**[0034]** In one example, a resource transaction event 132 is detected and the OS configuration state 114 is captured by using an OS abstraction layer of instructions 134. These instructions 134 are executed by the processor 106 to push the OS configuration state to the resource management unit 102. Alternatively, or in combination with the abstraction layer, the OS configuration state manager 112 on the resource management unit 102 is configured to capture or pull the OS configuration state 114 upon detecting resource transaction events 132. In some cases these resource transaction events 132 are detected through monitoring write transactions to the high-speed bus 128 interface and data link 130, such as through API queues and interrupt lines.

**[0035]** The OS configuration state 114, captured via the OS configuration state manager 112 in the resource management unit 102, is stored in the OS configuration state database 116 associated with the resource management unit memory 104 so that the resource status manager 118 may identify a status of any particular resource by accessing the database. Examples of resources for which a status may be identified include a process run queue, process wait queue, process priority data, process run-time

remaining, resource scheduling data, active device data, memory usage data, memory mapping data, virtual memory tables, storage management data, a hypervisor, a virtual machine (VM), a VM guest operating system, and resources used by the hypervisor or virtual machine.

**[0036]** Based on an understanding of the OS configuration state database 116 and its depiction of operating system resources, the resource management unit resource task manager 120 processes a task associated with the status of a resource. The resource management unit 102 identifies and acts on a resource task in advance of what the processor 106 may identify and process. The resource management unit can increase processing priority or performance factors for a specific task, or address a security aspect of the computing system relative to a particular process, or process an overhead management activity of the computing system. All these aspects can result in improved overall system performance and also an improved user's perception of responsive processing. This reduced latency, both real and as perceived by a user, is enabled because the processor 106 can focus on user-perceivable activities or other priorities while the resource management unit performs overhead system management work and other processing tasks.

**[0037]** FIG. 2 is a block diagram illustrating another example of a computing system 100 generally including the resource management unit 102, operating system 110, memory 108, and processor 106. However, in this example the computing system 100 is configured with a hypervisor 136 and virtual machine 138 running a guest operating system. Although only one virtual machine is depicted, the computing system 100 may include multiple virtual machines and/or guest operating systems as managed by the hypervisor 136.

**[0038]** The components, functionality, and methods previously discussed with respect to FIG. 1 perform similarly for managing the hypervisor and virtual machine components included in FIG. 2. For example, the OS configuration state manager 112 of the resource management unit 102 stores the OS configuration state 114 into an OS configuration state database 116. The OS configuration state 114, however, may include operating system aspects relating to the hypervisor 136 and virtual machine 138. The resource status manager 118 again identifies a status of a resource in the computing



system, as stored in the database 116 based on the configuration state 114 of the operating system 110, hypervisor 136, and virtual machine 138. The resource task manager 120 processes a task associated with the status of the resource, as identified by the resource status manager 118, to alleviate the processor 106 from performing the task, thus improving overall performance of the computing system.

**[0039]** Similar to the aspects discussed in reference to FIG. 1, capturing the OS configuration state 114 of the operating system 110, the hypervisor 136, and the virtual machine 138, includes obtaining operational status information of the operating system and its managed resources. This information may be obtained by monitoring and detecting the OS resource transaction events 132. Such information may be captured by the resource management unit 102 by using virtualization technologies 140, such as single root input/output virtualization (SR-IOV) or multi-root input/output virtualization (MR-IOV), to allow isolation of the bus (*e.g.*, PCIe) by the hypervisor or virtual machine for manageability and performance. Example operational status information of the managed resources may relate to a process run queue, a process wait queue, process priority data, process run-time remaining, resource scheduling data, resource usage data, active device data, memory usage data, memory mapping data, virtual memory tables, storage management data, a hypervisor, a virtual machine, a virtual machine guest operating system, or resources used by the hypervisor or virtual machine.

**[0040]** FIG. 3 is a conceptual diagram illustrating an example computing system 300 as, or integrated into, a computing device 302 configured with components and functionality for capturing OS configuration states for offloading tasks to a resource management unit (*e.g.*, a resource management unit 102). The computing device 302 is an example computing environment or application for the computing system 100 of FIG. 1. As some non-limiting examples, the computing device 302 may be a mobile phone 302-1, a tablet device 302-2, a laptop computer 302-3, a television/display or desktop or server computer 302-4, a computerized watch 302-5, or other wearable device 302-6, a game controller 302-7, a networked multimedia or voice assistant system 302-8, or an appliance 302-9.

**[0041]** The computing device 302 includes the resource management unit 102, the memory 104, and the OS configuration state manager 112 configured to capture or obtain the OS configuration state 114 of the operating system 110 stored in the memory 108 and executing on the processor 106. The OS configuration state manager 112 of the resource management unit 102 stores the OS configuration state 114 into an OS configuration state database 116 associated with the resource management unit 102. The resource status manager 118 identifies a status of a resource in the computing device, as stored in the database 116 based on the OS configuration state 114. The resource task manager 120 processes a task associated with the status of the resource, as identified by the resource status manager 118, to alleviate the processor 106 from performing the task and thereby improving overall performance of the computing device.

**[0042]** The computing device 302 also includes communication components 304, input/output components 306, communication interfaces 310, and input/output interfaces 312. These interfaces and components may leverage the memory 108 and processor 106, and/or be incorporated into the resource management unit 102 to leverage its operations.

**[0043]** The resource management unit 102 may also include processing units 308. The processing units 308 process computer-executable instructions to perform operations and execute functions of the computing device 302. The processing units 308 may include any combination of controllers, microcontrollers, processors, microprocessors, hardware processors, hardware processing units, digital-signal-processors, graphics processors, graphics processing units, video processors, video processing units, and the like.

**[0044]** The memory 104 associated with the resource management unit 102, and the memory 108, store information and instructions that are executed by the processor 106 and/or processing units 308 to perform operations and execute functions. These memories are configured to provide the computing device 302 with persistent and/or non-persistent storage of executable instructions (*e.g.*, firmware, recovery firmware, software, applications, modules, programs, functions, and the like) and data (*e.g.*, user data, operational data, scan results) to support execution of the executable instructions.

Examples of these memories include volatile memory and non-volatile memory, fixed and removable media devices, and any suitable memory device or electronic data storage that maintains executable instructions and supporting data. These memories can include various implementations of RAM, ROM, flash memory, cache memory, and other types of storage memory in various memory device configurations, or may be a solid-state drive (SSD), a hard disk drive (HDD), or combination thereof. These memories exclude propagating signals.

[0045] The communication components 304 enable wired and/or wireless communication of device data between the computing device 302 and other devices, computing systems, and network 126. The communication components 304 can include receivers, transmitters, and transceivers for various types of wired and wireless communications. Communication interfaces 310 handle messaging and protocols associated with communications being transmitted and received using the communication components 304.

[0046] The input/output components 306 provide connectivity to the computing device 302. For example, the input/output components 306 may include a user interface device that manages a user interface of the computing device 302, or may provide additional connectivity, beyond just the user interface device. The input/output components 306 can also include data interfaces or data input ports for receiving data, including user inputs. The processor 106 and/or processing units 308 may tailor operations according to input information obtained by the input/output interfaces 312 from the input/output components 306. Likewise, based on information obtained by the communication interfaces 310 from the communication components 304, the processor 106 and/or processing units 308 tailor operations according to incoming or outgoing communications.

[0047] FIGs. 4-5 are flow diagrams 400, 500 illustrating high-level operations of example methods for capturing operating system configuration states in a computing system for offloading tasks to a resource management unit. The operations of methods 400, 500, and other methods described herein, may be embodied as programming instructions stored on a non-transitory, machine-readable (*e.g.*, computer/processor-readable) medium, such as a RAM or ROM memory or other storage device for



execution in a computing device or devices, or may be performed by hardware blocks, or combinations thereof. In some examples, implementing the operations of the methods can be achieved by a processor complex, including a main processor and a resource management unit, reading and executing the programming instructions stored in the memory, and/or functioning in combination with hardware blocks of the resource management unit. In some examples, implementing the operations of the methods can be achieved using a resource management unit such as a system-on-chip (SoC), and/or other hardware components either alone or in combination with programming instructions executable by a processor or processors in a computing device.

**[0048]** The example methods described in this disclosure may include more than one implementation, and different implementations of the methods may not employ every operation presented in the respective flow diagrams, or may employ additional operations not shown. Therefore, while the operations of the methods are presented in a particular order within the flow diagram(s), the order of their presentations is not intended to be a limitation as to the order in which the operations may actually be implemented, or as to whether all of the operations may be implemented. For example, one implementation of the methods might be achieved through the performance of a number of initial operations, without performing subsequent operations, while another implementation of the methods might be achieved through the performance of all of the operations.

**[0049]** Referring now to example method 400 of FIG. 4, a first operation 402 includes posting an operating system resource transaction event in a memory of a computing system responsive to the operating system executing on a processor in the computing system. An operating system transaction event is indicative of when a configuration state of the operating system may be captured because an operating system event is occurring or has occurred. Capturing a configuration state of the operating system includes obtaining operational status information of the operating system and its managed resources. Examples of such resources and operating system operational status information include a process run queue, a process wait queue, process priority data, process run-time remaining, resource scheduling data, resource usage data, active device data, memory usage data, memory mapping data, virtual

memory tables, storage management data, hypervisor activity, virtual machine activity, virtual machine guest operating system activity, and other related process and system activities, statuses, and data.

**[0050]** At 404, a resource management unit, such as an SoC, obtains the configuration state of the operating system and stores it in a database (*e.g.*, memory) associated with the resource management unit. For example, the resource management unit 102 described above obtains the configuration state by communicating with the processor and memory using low-latency communication and read/write transactions, either by way of the resource management unit being implemented on a same integrated circuit die as the processor and its memory, or by using a high-speed interface bus and data link. The resource management unit may also have a direct memory access (DMA) channel to the memory for transferring data to/from memory without passing it through the processor in the computing system.

**[0051]** At 406, the resource management unit identifies a status of a resource in the computing system based on the operating system configuration state stored in the database, and identifies a task to be performed that is associated with the resource. For example, a status of a process resource in the process run queue may be identified as a process that is next up for execution time on the processor, and that a memory resource of some number of bytes is needed for the process to execute. As another example, a status of a process resource in the process wait queue may be identified as one that is waiting for a resource-related event to occur. An example event may be that the process needs a virtual file system (VFS) inode describing a directory in the file system, but that inode may not be in the cache. Because of this, the process must wait for that inode to be fetched from the physical media containing the file system before it can continue executing.

**[0052]** At 408, the resource management unit processes the task associated with the status of the resource to alleviate the processor from performing that task and to improve the performance of the computing system. For example, in the context described above (that of a task needing a memory resource of some number of bytes of memory), the resource management unit may execute the task of making that memory resource available, such as by swapping out memory, so that sufficient memory is

available when the process is actually executed again on the processor. As another example, in the context described above regarding the wait queue waiting for an inode to be fetched, the resource management unit may execute the task to fetch the inode in advance of when the operating system and processor may be able to do it. This allows processes to avoid waiting to execute relative to if they had waited for the processor (not the resource management unit device's processor) to fetch the inode. Other examples include the resource management unit modifying processing priority or performance factors for a specific process, addressing a security aspect of the computing system relative to a particular process, and processing other overhead management activities of the computing system.

**[0053]** Referring now to FIG. 5, example method 500 depicts additional details for capturing operating system configuration states for offloading tasks to a resource management unit in a computing system. At 502, an operating system stored in a memory executes on a processor in a computing system. At 504, during execution of operating system instructions, the operating system posts a resource transaction event to memory indicative of a resource event occurring and a new configuration state of the operating system. As an example, an abstraction layer of the operating system software posts resource transaction event data indicative of, for example, a process execution status, resource usage and scheduling, memory usage and mapping, hypervisor status, virtual-machine status, and/or any other transaction event that the operating system is managing for the computing system.

**[0054]** At 506, a resource management unit communicates with the processor and memory via a low-latency communication data link. Such low-latency communication data link may be a high-speed bus (*e.g.*, PCIe) or via cache if the resource management unit is implemented on a same integrated circuit die as the processor. At 508 the resource management unit obtains (*e.g.*, captures) the configuration state of the operating system and stores it to a database associated with the resource management unit. Note that after a transaction event is posted 504, a return execution flow arrow 514 in the diagram illustrates that the operating system continues executing on the processor 502, even while the resource management unit communicates 506 with the memory and processor. This allows the operating system



to continue performing its operations, even while the resource management unit is performing its functions 508, 510, 512, for overall improved system performance.

**[0055]** At 510, the resource management unit identifies a status of a resource from the database based on the configuration state of the operating system. For example, a status of a resource may be identified as a process that needs additional time to execute, or a process that will need to access certain I/O resources that need to be retrieved from storage, or a process that may pose a security issue if executed, or a process or system that may benefit from modifying a clock rate/signal frequency for a clocked component.

**[0056]** At 512, the resource management unit processes a task associated with the status of the resource. For example, memory is swapped in advance so the process waiting next in queue has sufficient memory available when it begins to execute on the processor as described above, or a resource is fetched as described above, or additional memory is allocated for the process waiting next in queue, or a potential security issue is resolved concerning a process, or a component signal frequency is adjusted to meet processing demands. This awareness of a resource status enables the resource management unit to act on a task in advance of when the processor or operating system may eventually perform the task.

**[0057]** After the task is processed for the resource at 512, execution control returns, at 516, to allow the resource management unit to identify 510 a status of a next resource from the database as needed, and process another task 512 that is associated with that next resource. This processing is repeated as needed for resources identified in the database. As this resource status identification 510 and task processing 512 is performed by the resource management unit, the operating system continues executing 502 and posting resource transaction events 504, and the resource management unit continues communicating 506 with the processor and memory and capturing 508 a configuration state of the operating system and storing it in the database. This example method of capturing operating system configuration states for offloading tasks from a processor in a computing system to a resource management unit provides overall improved computing system performance.

[0058] FIG. 6 is a block diagram illustrating an example OS configuration state 114 in a memory 108 of a computing system 100 as captured by a resource management unit 102 into a memory 104 and a database 116. Similar to the discussion with respect to FIGs. 1-3, the computing system 100 generally includes the resource management unit 102, the memory 108, the operating system 110, and the main processor 106. This example depicts operational status information in the memory 108 for exemplary resources as indicative of the OS configuration state 114 at a time that a resource transaction event is detected (*e.g.*, the resource transaction event 132). In this example, OS configuration state 114 also depicts a process control block 142 representative of a status of an example process in the process run queue or wait queue as described above in reference to FIG. 1. Although only a single process control block is depicted, a process control block exists for each process in the process run queue and wait queue. Upon detecting the resource transaction event 132, the OS configuration state 114 is copied 602 (pushed or pulled) into the OS configuration state database 116 on the resource management unit 102. Although the diagram illustrates an example OS configuration state identifying a number of example resources, other operating system configuration state resources, resource references, or status indicators may similarly be referenced, identified, and captured by the resource management unit 102 into the database 116 in coordination with the OS configuration state manager 112.

[0059] With the database 116 having the OS configuration state 114, the resource management unit 102 may then process a task associated with a status of any given resource identified in the database to improve the performance of the computing system. For example, the resource management unit 102 may increase processing priority or performance criteria for a specific process, or address a security aspect of the computing system relative to a particular process, or process an overhead management activity of the computing system, or modify a signal frequency of a clocked component. An example of processing an overhead management activity, such as memory allocation or data fetch, is described above in reference to FIG. 4 and further below in reference to FIG. 9. Having the resource management unit process each of these tasks results in improved overall system performance and may also improve a user's perception of responsive processing in the computing system because the processor 106 can focus on

user perceivable activities or other needed activities while the resource management unit 102 performs the overhead system management work.

#### DYNAMICALLY SCALING CLOCK RATES WITH A RESOURCE MANAGEMENT UNIT

**[0060]** FIG. 7 is a block diagram illustrating the example computing system 100 configured for capturing operating system configuration states by a resource management unit for dynamically scaling clock rates. Dynamically scaling clock rates or clock frequency (used interchangeably herein) of clocked components in a computing system achieves benefits including performance boost, power optimizations, thermal dynamic constraints, and reduced processing time. Enabling a resource management unit, rather than a CPU, to dynamically scale clock rates in the system responsive to operating system configuration states, enables more-responsive and relevant scaling to active events, and avoids the CPU wasting resources on scaling tasks so that it may address other computing system operations.

**[0061]** The operating system 110 executes on the processor 106 from the memory 108, and upon detecting an OS transaction event 132, the OS configuration state 114 is captured 602 by the OS configuration state manager 112 of the resource management unit 102 and stored into the OS configuration state database 116 of the memory 104. For simplicity and drawing space limitations, only two example operating system resources are shown in the OS configuration state 114 and the database 116, a process run queue and a process wait queue, although other OS configuration state resources, status indicators, or data may similarly be represented, referenced, identified, and captured by the resource management unit 102 into the database 116.

**[0062]** With the process run queue and process wait queue resources represented in the database 116, the resource management unit resource status manager 118 may then identify the status of a specific resource 116 in those queues and process a task 120 associated with the specific resource to improve the performance of the computing system. In this example, the resource status manager 118 identifies the status of a process resource from the run queue in the database 116 that indicates it requires or would benefit from modifying the clock rate for a clocked component in the computing system. The benefits include to maintain or improve performance of the process, overall



system performance, power usage, or thermal dynamics of the computing system 100. In one example, the status of the process resource is identified by referencing the process control block. Factors in the process control block considered may include the process state, scheduling information, memory management data, and accounting data that indicates significant processing time is needed or consumed or that significant memory is needed. By knowing the status of the process, such as when it is going to execute, the resources it needs, and the status of other resources in the computing system, the resource management unit may dynamically scale clock rates accordingly. Modifying the clock rate may include modifying a performance of the processor or access speed of the memory to address load operating conditions, performance, power consumption, thermal dynamics in the computing system, or combinations thereof.

**[0063]** With the captured OS configuration state available 116 that identifies the status of the process, and optionally with other process related metrics available, the resource management unit 102 can identify processing needs and clock-rate modification benefits in advance of what the operating system 110 may actually detect and execute on the processor 106. For example, the OS configuration state, along with associated process metrics or a detected or assigned process score representing process metrics, can show that (i) the identified process needs more memory resources than normal, (ii) the process requires processor intensive activity, (iii) the process may work with reduced memory resources or reduced processor activity, (iv) a particular core in the processor 106 or a memory bank in the memory 108 may be powered down (*e.g.*, clock rate terminated) for a short time, or (v) a hypervisor or virtual machine requires increased or reduced processor activity or memory usage for improved overall system performance.

**[0064]** Where it is determined by the resource status manager 118 that a clock rate of a clocked component in the computing system, such as the processor 106 or the memory 108, may be modified to improve aspects of the process or the computing system 100, then the resource task manager 120 initiates the task of modifying the clock rate for the relevant clocked component. In this depicted example, based on information identified about a process in the OS configuration state database 116, the resource task manager 120 may modify a clock rate 702 for a clock 704 which modifies the clock rate

signal 706 to the memory 108. Similarly, in this depicted example, the resource task manager 120 may modify the clock rate 702 for a clock 708 which modifies the clock rate signal 710 to the processor 106. The resource task manager 120 may similarly modify the clock rate 702 to any number of other clocks in the system 100 to modify the clock rate signal of other respective clocked components not shown in this example (*e.g.*, a graphics processing unit (GPU), MMU, bus, specific memory bank).

**[0065]** To determine whether a process or the overall system may benefit from modifying the clock rate, any number of factors relating to the process, its process control block, and the OS configuration state 114 may be considered including, for example, memory usage, memory need, scheduling, process run queue, process wait queue, process priority data, process run-time remaining data, process scheduling data, hypervisor status, virtual-machine status, or combinations thereof. Alternatively, or in combination, processing metrics associated with a process (process metrics) may be considered that are indicative of resources used and system metrics associated with execution of the process. Process metrics may be captured, assigned a score, and stored in the database 116, or the score may be detected or obtained if it was previously assigned to the process. Process metrics may be captured and obtained by monitoring the process during execution on the processor 106 or, if available, retrieved from the database 116 or a storage resource 712 external to the computing system 100 and accessible via communication manager 122 and the network 126.

**[0066]** Process metrics that reflect resources used and related processing data may be obtained by using an operating system analysis tool, such as a Linux operating system event-oriented observability tool, *e.g.*, “perf.” This allows for tracking performance counters, events, trace points, tasks, workload, control-flow, cache misses, page faults, and other profiling aspects of the operating system. Metrics of the process that are captured using an observability or profiling tool, or obtained from the external storage resource 712, are maintained in the database 116 for the resource status manager 118 to reference in determining whether modifying a clock rate of a component may be advantageous for the process or system overall. Modifying the clock rate based on process metrics may be advantageous in any number of events including, for example, increasing the clock rate for the memory 108 if the process score or metrics indicate

that the process requires heavy memory usage, or increasing the clock rate for the processor 106 if the process score or metrics indicate that the process requires heavy processing, or reducing the clock rate for the processor to reduce thermal issues if the process score or metrics indicate heavy processing that creates increased thermal issues.

[0067] Because the process score represents the processing metrics associated with the process, it enables the resource status manager 118 to quickly and easily identify whether the process or the system would benefit from modifying a clock rate of a clocked component. The score also enables the resource status manager 118 to easily compare the process with scores of other processes in the system. A score may be obtained from the external storage resource 712 if the process is one that has been previously associated with a score. For example, a process that represents a game played by users on a computing device may be assigned a score relative to its known processing metrics as used in other computing devices or systems. That score may then be stored in the external storage resource 712 as associated with that particular process and leveraged by the resource management unit 102 to quickly and easily determine benefits of modifying clock rates for the process in the computing system 100.

[0068] The resource task manager 118 may also detect other scores associated with other process statuses stored in the OS configuration state database 116, or assign other scores to those other processes based on processing metrics detected, or obtain other scores of other processes from the storage resource 712, all representative of processing metrics associated with those other processes and resources used by those processes. The resource task manager may then compare the process score to the scores of the other processes to determine if, when, and how to modify clock rates for clocked components in the computing system 100 for improved system performance.

[0069] For example, the process score may be compared with respect to a score of another process, or to a combination of scores of multiple other processes in the process run queue, and in an event that the score of the other process or a combined score of the multiple other processes meets a given threshold indicative for modifying the clock rate of the component, then the clock rate of that component may be modified. As one example, if the process score is weighted very heavy on processor usage, and there are no other process scores in the process run queue that are weighted processor



heavy, then the resource task manager 118 may not need to modify the clock rate for the process. On the other hand, if there are other process scores that are weighted processor heavy in the process run queue, then the resource task manager 118 may increase the clock rate for the process relative to the processor 106, the memory 108, or other clocked component.

**[0070]** FIG. 8 illustrates an example method 800 for capturing operating system configuration states by a resource management unit for dynamically scaling clock rates. At 802, the resource management unit captures a configuration state of the operating system in memory that is executing on the processor in the computing system, and stores the configuration state to a database associated with the resource management unit. At 804, the resource management unit identifies from the database a status of a process resource in the computing system based on the configuration state of the operating system. In this example for dynamically scaling clock rates, the process may be a process in a run queue waiting to execute on the processor, or a process in the wait queue that has partially executed but is paused and waiting for an event to occur before beginning to execute again.

**[0071]** The resource management unit examines the process status to determine whether the process or the computing system would benefit from modifying a clock rate of a clocked component in the computing system, such as the processor or memory, to maintain or improve performance of the process, overall system performance, power consumption, or thermal dynamics of the computing system. For example, the resource management unit examines the process control blocks of processes in the run queue or wait queue. If a process state, scheduling information, memory management data, and/or accounting data indicate significant processing time still needed or consumed, or significant memory needed, then the processor clock rate or memory clock rate may be increased by the resource management unit to address the need. Another example, at 806, is for the resource management unit to detect whether a process score is associated with the process. The process score represents overall processing metrics associated with the process, and resources used by the process in the computing system and operating system, or in another computing system and operating system external to the computing system with the resource management unit. The process score enables

the resource management unit to easily identify whether the process or the system would benefit from modifying a clock rate of a clocked component.

[0072] If a process score is not detected 806 in association with the process, then a process score may be obtained 808 from an operating system profiling tool or an external storage resource. While the process is executing in the computing system the process score may be calculated or obtained using standard operating system profiling tools known for tracking performance counters, events, trace points, tasks, workload, control-flow, cache misses, page faults, and other profiling aspects of the operating system. Alternatively, the process score may be obtained from an external storage resource.

[0073] At 810, if the OS configuration state or process score meets a threshold, then the resource management unit modifies the clock rate 812 for a clocked component associated with the process or otherwise beneficial to the computing system. If the threshold is not met, then execution control returns 814 for the resource management unit to identify 804 a status of another process resource in the computing system.

[0074] To determine whether an OS configuration state meets the threshold 810 for modifying the clock rate, factors associated with the process status are considered including, for example, memory usage, memory need, scheduling, process run queue, process wait queue, process priority data, process run-time remaining data, process scheduling data, hypervisor status, virtual machine status, or combinations thereof. On the other hand, the process score allows the resource management unit to easily compare it to a threshold score 810, or to compare it with scores of other processes in the system to determine if an overall threshold score is met 810. Scores of other processes in the system may be obtained or calculated from a profiling tool while the other processes execute in the computing system, or from an external storage resource if those other processes have been previously associated with a score. If the OS configuration state and/or process score meets the threshold 810, then the resource management unit modifies the clock rate 812 for respective clocked components in the computing system, and execution control returns 816 to identify 804 a status of another process and act on it.

[0075] Modifying the clock rate based on the operating system configuration state, process status, and metrics may be advantageous in any number of events. For example, increasing the clock rate for memory may occur if the metrics suggest that the process requires heavy memory usage, or increasing the clock rate for the processor, or a hypervisor or virtual machine may occur if the metrics suggest that any requires heavy processing. On the other hand, reducing the clock rate for the processor may occur to reduce thermal dynamics if the metrics suggest heavy processing that typically creates increased thermal issues, or even terminating a clock rate briefly may occur in certain components if the metrics suggest no activity in those components.

#### OFFLOADING MEMORY MANAGEMENT TO A RESOURCE MANAGEMENT UNIT

[0076] FIG. 9 is a block diagram illustrating an example computing system 100 configured with a resource management unit 102 for capturing operating system configuration states and managing memory swapping. The operating system 110 executes on the processor 106 from the memory 108. Upon detecting an OS transaction event 132, the OS configuration state 114 is captured 602 by the OS configuration state manager 112 of the resource management unit 102 and stored into the OS configuration state database 116 of the memory 104. For simplicity of discussion and drawing limitations, only two example operating system resources are shown in the database 116, a process run queue and a process wait queue, although other operating system configuration state resources, status indicators, or data may similarly be represented, referenced, identified, and captured by the resource management unit 102 into the database 116.

[0077] With the process run queue and process wait queue resources represented in the database 116, in this example the resource management unit resource status manager 118 identifies a process from the run queue in the database 116 that requires a swap of memory resources 902 associated with the process to maintain or improve overall system performance. The process status and resources associated with the process, or relative to the process, such as identified in the process control block, are identified from the OS configuration state 114 of the operating system 110 as stored in the database 116 by considering any number of aspects. For example, aspects may



include memory usage, memory need, secondary memory usage, scheduling, process run queue, process wait queue, process priority data, process run-time remaining data, process scheduling data, status of a hypervisor or virtual machine, or combinations thereof.

[0078] The term memory swapping may refer to copying an entire process address space out to a swap device, such as the secondary storage 124, or back, as a single event. And the term paging may refer to copying, in or out, one or more same-size blocks of the address space, at a much finer grain, such as 4K bytes per block, referred to as a page-size. However, for simplicity of discussion in this disclosure, paging and swapping will be used interchangeably to reference the copying of memory content to or from the secondary storage 124.

[0079] With the captured OS configuration state available in the database 116, the resource management unit 102 can identify memory swapping requirements for a process in advance of what the operating system will detect and execute on the processor 106, thus avoiding memory page faults and improving computing system performance. For example, the OS configuration state can show that the identified process requires more memory resources to execute than what is currently available in the memory 108, or the process recently completed execution and doesn't require the memory resources. The resource status manager 118 reviews the process in the database 116 to see if it is a good candidate for swapping, *e.g.*, if it has pages that can be swapped in, or discarded from, memory, considering factors such as aging, whether or not the page is locked in memory, or whether the page is shared in memory.

[0080] When the resource status manager 118 determines that the process requires memory content 902 to be swapped in, or may be swapped out, the resource task manager 120 executes the memory swap 904 between the memory 108 and the secondary storage 124. If the process needs more memory to execute than what is available in the memory 108, then the resource management unit 102 may move other memory content out to the secondary storage 124 that is not needed for execution at the current time, and the resource management unit swaps in content from the secondary storage 124 that is needed for the process to execute. On the other hand, if the process has completed execution and doesn't need to use the memory content 902, then the

resource management unit 102 may move the memory content 902 out to the secondary storage 124 to free up the memory 108.

**[0081]** In another example, the memory content 902 is swapped between the memory 108 and the secondary storage 124 using a variant size of the memory content without a page-size calculation or page-size processing of the content. For example, rather than using a standard set page-size (*e.g.*, 1K, 2K or 4K bytes) to swap the memory content 902, thereby requiring multiple processing tasks of swapping the multiple pages, and potentially using memory less efficiently, the resource management unit 102 may swap the memory content 902 out to the secondary storage 124 in a specific amount needed, either some or all of the memory content 902, or swap it into the memory 108 in the specific amount needed, in one data-stream swap. For example, if the size of memory that needs swapping is 1.5K, then the resource management unit 102 swaps 1.5K. Thus, there is no need to process the swap using a traditional page-size calculation or multi-staged page-processing activity to swap some or all of the memory content 902.

**[0082]** This memory management and swapping can be performed because the resource management unit has an understanding of the OS configuration state 116 and, for example, since both wait queues and scheduling priorities are known, the resource management unit knows what memory each process will use, what memory is free, how much memory is needed to be swapped, how much memory will be needed by a next process in the run queue to execute, and other factors that guide usage of the memory for improved performance in the computing system. The resource management unit 102 can, using its own direct memory access (DMA) and without pausing the processor 106, pre-swap pages for each process before it runs, in the order the process will consume those pages (*e.g.*, using stack and heap pointers for predicting this) and in case there is insufficient memory 108 for those pages. The resource management unit also knows the last process that was executed on the processor 106, and which has the most time left until it is back to a running state, so the resource management unit can evict that last process pages to make more room in the memory 108 for other processing needs.

[0083] Similarly, in the context of a hypervisor 136 or virtual machine 138 running in the computing system 100, this memory management processing by the resource management unit 102 for the hypervisor and virtual machine improves their effectiveness, *e.g.*, the hypervisor swapping out or migrating virtual machines. The resource management unit knows which virtual machine will run the latest, and which will be scheduled next, so the resource management unit can fetch the virtual machine to the memory 108 when needed and swap it back to secondary storage after it is done executing, making the hypervisor more effective.

[0084] FIG. 10 illustrates an example method 1000 for capturing operating system configuration states by a resource management unit in a computing system for managing memory swapping. At 1002, the resource management unit captures a configuration state of the operating system executing on the processor in the computing system, and stores the configuration state to a database (*e.g.*, memory) associated with the resource management unit. At 1004, the resource management unit identifies from the database a status of a process resource in the computing system, as may be referenced in the process control block, based on the configuration state of the operating system. As an example, the process may be a process in a run queue but waiting to execute on the processor, or a process that has partially executed but is paused and waiting in the wait queue for an event to occur before beginning to execute again.

[0085] In this example, the resource management unit identifies a process from the run queue in the database as requiring to swap memory resources to maintain process execution or improve system performance. The status of the process, and resources associated with the process or relevant to the process, are identified from the configuration state of the operating system by considering factors such as its memory usage, memory need, secondary memory usage, scheduling, process run queue, process wait queue, process priority data, process run-time remaining data, process scheduling data, hypervisor or virtual machine activity, or combinations thereof.

[0086] At 1006, if it is determined that a memory swap is not required, then execution control returns 1012 to identify another process 1004. If it is determined 1006 that the process requires memory swapped based on the captured operating system configuration state, then the resource management unit determines a memory swap size



1008. In this example, the memory swap size is based on the variant size of the content to be swapped in the memory for improved system performance and efficient memory usage. Given the captured operating system configuration state, the resource management unit knows the details of the process and other system activities so that it can perform the memory swap based on the variant size of the content, rather than having to perform standard set page-size transfers of the content.

[0087] At 1010, the memory content is swapped to or from a secondary storage using, in this example, a swap size based on the variant size of the memory content without using a set page-size limitation. For example, the resource management unit may swap the memory content out to secondary storage, or swap it into the memory from secondary storage, in one data-stream swap using the actual size of the memory content, rather than a set page-size, thereby avoiding multiple processing tasks of swapping multiple pages. After the memory content is swapped 1010, execution control returns 1014 to identify a status of another process 1004.

[0088] While the resource management unit is processing all these functions of capturing the operating system configuration state 1002, identifying a process that may require a memory swap 1004, determining whether a memory swap is required 1006, determining the memory swap size 1008, and then actually swapping the memory content 1010, the operating system separately and in parallel continues executing its operations on the processor.

#### MANAGING MALWARE AND VULNERABILITIES WITH A RESOURCE MANAGEMENT UNIT

[0089] FIG. 11 is a block diagram illustrating an example computing system 100 configured to capture operating system configuration states with a resource management unit 102 for detecting and managing malware and software vulnerabilities. The operating system 110 is executing on the processor 106 from the memory 108, and upon detecting an operating system transaction event 132, the OS configuration state 114 is captured 602 by the OS configuration state manager 112 of the resource management unit 102 and stored into the OS configuration state database 116 of the memory 104. For simplicity of discussion and drawing space limitations, only two

example operating system resources are shown in the database 116, a process run queue and a process wait queue, although other operating system configuration state resources, data, or status indicators 114 may similarly be represented, referenced, identified, and captured by the resource management unit 102 into the database 116.

**[0090]** With the process run queue and process wait queue resources represented in the database 116, the resource management unit may then identify the status of a resource 118 and process a task 120 associated with the resource to improve the performance of the computing system. In this example, the resource management unit resource status manager 118 identifies a first process from the run queue in the database 116 as ready to be executed (*e.g.*, by referencing the process control block). Prior to execution of the first process, a fingerprint of the first process is identified (*e.g.*, generated) 1102 by the resource task manager 120 to subsequently compare to a plurality of fingerprints of flagged processes 1104 stored in the memory 104. In other aspects, the fingerprint of the first process is identified at multiple points in time, such as before, during, and/or immediately after execution, to capture a potential rolling (*e.g.*, changing) fingerprint of the process.

**[0091]** For this disclosure, a fingerprint represents one or more aspects, characteristics, schemes, traits, or perspectives that identify the process or its binary or source files. For example, the fingerprint may relate to, or be generated from, (i) the binary program file of the process, (ii) the source program file of the process, (iii) a perspective of the process in memory while the process executes (*e.g.*, as the process code may dynamically morph in memory while executing, including rolling or scattered execution changes in memory such as from a code segment to a data segment), and/or (iv) a perspective of the process in memory, or its binary file, or its source file, or some combination thereof, as generated by a machine learning algorithm. Accordingly, a single process may have multiple fingerprints, and the multiple fingerprints may be linked for association purposes.

**[0092]** The fingerprint of the first process may be identified by executing a hash function on the first process (*e.g.*, by executing a hash function on the binary program file or source program file of the first process, or on code or data of the process as it exists in memory for execution by the processor). For example, a rolling hash function

may be applied where the process data is hashed in a window that moves through the data, or a Rabin fingerprinting scheme may be applied using polynomials. The fingerprint may also be identified by passing the first process through a search filter, such as a regular expression (*e.g.*, regex or regexp) engine. This may be accomplished by applying the search filter to the (i) binary program file of the process, (ii) source program file of the process, (iii) code or data of the process as it exists in memory, and/or (iv) data, memory or devices the process is accessing.. Such regular expression functionality may be applied recursively, or iteratively, using varying parameters to progressively narrow the scope and results. The fingerprint generation method simply aligns with the fingerprint generation method of the plurality of fingerprints of flagged processes 1104 stored in the memory 104 of the resource management unit so that an accurate comparison may be made.

**[0093]** The plurality of fingerprints of flagged processes 1104 are indicative of processes, executable files, programs, applications, operating systems, or other software that include executable instructions or are binary program files that have a known vulnerability or have been identified as malware. Examples of malware may be in the form of a virus, trojan, worm, adware spyware, or ransomware, or any software that is specifically designed to disrupt, damage, or gain unauthorized access to or control of the computing system, or to steal or corrupt data. A vulnerability is any weakness in the software or in the operating system that may be exploited to disrupt, damage, or gain unauthorized access to or control of the computing system, or compromise or cause unintended behavior of the computing system.

**[0094]** The fingerprints of the flagged processes 1104 may be obtained periodically (meaning occasionally or continuously as system parameters allow) by the resource task manager 120 from an external storage 1106 through the resource management unit communication manager 122 and the network 126. The external storage 1106 may be a database on a local network external to the computing system, or a database located in a remote computing environment, such as a cloud computing environment or multi-cloud environment, accessed via the Internet. The external storage 1106 maintains a list of fingerprints of known malware and vulnerable software.

The resource management unit accesses the external storage 1106 periodically to update its own memory of fingerprints of flagged processes 1104.

**[0095]** The resource management unit 102 compares the fingerprint of the first process 1102 to the fingerprints of the flagged processes 1104 stored in the memory 104 of the resource management unit. Alternatively, or in combination with comparing to the fingerprints of the flagged processes 1104 stored in the memory 104, the resource management unit may compare the fingerprint of the first process to the fingerprints of the flagged processes as stored in the external storage 1106.

**[0096]** If there is not a match of the fingerprint of the first process 1102 to any of the fingerprints of the flagged processes 1104, then no action is taken, and the resource management unit 102 and the processor 106 continue executing their normal commands and functions. However, if there is a match of the fingerprint of the first process 1102 to at least one of the fingerprints of the flagged processes 1104, then the first process is recognized as malware or a vulnerable process, as the case may be, and the resource task manager 120 takes action to protect the computing system from unwanted effects of the first process. Such action may include any number of separately actionable or coordinated events between the resource management unit, the processor, and/or the operating system to protect the computing system. For example, such action may include notifying a user of the computing system, prohibiting execution of the first process, suspending scheduling of the first process, limiting execution of the first process, stopping execution of the first process, archiving the first process in a secure vault, removing the first process completely, stopping certain processing in the computing system, or combinations thereof.

**[0097]** While the resource management unit is processing all these described functions of capturing the OS configuration state 114 and managing malware and vulnerabilities, the operating system 110 separately and in parallel continues executing its operations on the processor 106. This leaves the processor and operating system unburdened by the activities of the resource management unit unless a vulnerable process or malware is detected and acted on.

**[0098]** FIG. 12 illustrates an example method 1200 for a resource management unit, such as a SoC, to capture operating system configuration states of an operating



system executing on a processor in a computing system, and to detect and manage a vulnerable process or malware in the computing system. At 1202, a plurality of fingerprints of flagged processes are stored to a memory of the resource management unit. The flagged processes are indicative of processes, executable files, programs, applications, operating systems, or other software that include executable instructions that have a known vulnerability or have been identified as malware. The plurality of fingerprints of the flagged processes may be obtained periodically from an external storage which maintains a list of fingerprints of known malware and vulnerabilities. The resource management unit accesses the external storage periodically to update its own memory of fingerprints of flagged processes.

**[0099]** At 1204, the resource management unit obtains a configuration state of the operating system executing on the main processor in the computing system, and stores the configuration state to a database associated with the resource management unit. At 1206, the resource management unit identifies from the database a status of a first process resource in the computing system based on the configuration state of the operating system. As an example, the first process may be a process in a run queue but waiting to execute on the processor, or a process that has partially executed but is paused and waiting in the wait queue for an event to occur before beginning to execute again.

**[0100]** At 1208, the resource management unit identifies a fingerprint of the first process. The fingerprint may be identified, for example, by executing a hash function on the first process to generate a fingerprint, or by passing the first process through a search filter, or some combination thereof. Multiple fingerprints of the first process may also be identified (*e.g.*, generated), at different points in time relative to execution, and linked for association with the process.

**[0101]** At 1210, the resource management unit compares the fingerprint of the first process to the fingerprints of the flagged processes stored in the memory of the resource management unit. Alternatively, or in combination with comparing to the fingerprints of the flagged processes stored in the memory, the resource management unit may compare the fingerprint of the first process to the fingerprints of the flagged processes as stored in the external storage.

[0102] If there is not a match 1212 of the fingerprint of the first process to any of the fingerprints of the flagged processes, then execution control returns 1216 for the resource management unit to repeat the steps of identifying another process 1206, identifying a fingerprint 1208, and comparing the fingerprint 1210 to fingerprints of flagged processes. If there is a match 1212 of the fingerprint of the first process to one of the fingerprints of the flagged processes, then the first process is recognized as malware or a vulnerable process, as the case may be, and the resource management unit takes action 1214 to protect the computing system from unwanted effects of the first process. Such action may include any number of actions as described above in reference to FIG. 11.

[0103] Notably, while the resource management unit is processing all these functions depicted in the method 1200, and also while the resource management unit repeats processing these functions against different process resources identified in the computing system based on capturing the configuration state of the operating system, the operating system separately and in parallel continues executing its operations on the processor in the computing system for overall improved computing system performance.

[0104] While this disclosure has been described with respect to example embodiments outlined above, it is evident that alternatives, modifications, and variations will be apparent to those skilled in the art. Accordingly, the described and depicted embodiments of the present disclosure are intended to be illustrative, not limiting, and the subject of the appended claims is not necessarily limited to the specific features or methods described in this disclosure.

#### EXAMPLES

[0105] In the following paragraphs, some examples are described.

[0106] Example 1: A method of managing a computing system (100; 300), the method including: capturing (112; 404; 508), by a resource management unit (102) and into a first memory (104), a configuration state (114) of an operating system (110) in a second memory (108), the operating system (110) executing on a processor (106) of the computing system; identifying (118; 406; 1004), by the resource management unit

(102), a status of a process (116) in the computing system (100; 300) based on the configuration state (114) of the operating system (110), the status of the process indicative of requiring to swap a memory resource in the computing system; and responsive to identifying the status of the process indicative of requiring to swap the memory resource, swapping, by the resource management unit (102), the memory resource (120; 904; 1010) effective to alleviate the processor (106) from swapping the memory resource.

**[0107]** Example 2: The method of example 1 wherein the status of the process indicative of requiring to swap the memory resource is indicative of the process needing the memory resource to execute on the processor, or the process completed execution and not needing the memory resource.

**[0108]** Example 3: The method of any preceding example wherein swapping the memory resource includes swapping a content of the memory resource between the second memory and a secondary storage.

**[0109]** Example 4: The method of any preceding example wherein swapping the memory resource includes swapping a content of the memory resource using a variant size of the content of the memory resource without a page-size or block-size calculation of the content.

**[0110]** Example 5: The method of any preceding example wherein swapping the memory resource includes swapping a content of the memory resource from a secondary storage into the second memory before execution of the process by the processor for the process to use the content in the second memory during execution of the process by the processor.

**[0111]** Example 6: The method of any preceding example wherein swapping the memory resource includes swapping a content of the memory resource from the second memory to a secondary storage after execution of the process by the processor for a second process to use the second memory upon execution of the second process by the processor.

**[0112]** Example 7: The method of any preceding example further including the resource management unit communicating with the processor or the second memory for capturing the configuration state of the operating system using a low-latency

communication data link or a high-speed interface bus, or by being integrated with a same integrated circuit die as the processor.

**[0113]** Example 8: The method of any preceding example wherein capturing the configuration state of the operating system includes an abstraction layer of instructions coordinating with the operating system as executed by the processor for detecting a resource transaction event associated with the operating system and pushing the configuration state of the operating system to the first memory of the resource management unit responsive to the resource transaction event.

**[0114]** Example 9: The method of any preceding example wherein capturing the configuration state of the operating system includes the resource management unit detecting a resource transaction event associated with the operating system and pulling the configuration state of the operating system to the first memory of the resource management unit responsive to the resource transaction event.

**[0115]** Example 10: The method of any preceding example wherein identifying the status of the process includes identifying a memory usage, memory need, secondary storage, scheduling data, process run queue, process wait queue, process priority data, process run-time remaining, hypervisor, virtual machine, or combinations thereof.

**[0116]** Example 11: The computing system (100; 300) of any of examples 1 to 10 including: the processor (106); the resource management unit (102); and the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of examples 1 to 10.

**[0117]** Example 12: The computing system (100; 300) of any of examples 1 to 10 including: the processor (106); the resource management unit (102) including a system-on-a-chip, an application-specific integrated circuit, or an application-specific standard product; and the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of examples 1 to 10.

**[0118]** Example 13: The computing system (100; 300) of any of examples 1 to 10 including: a hypervisor (136) or a virtual machine (138); the processor (106); the



resource management unit (102); and the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of examples 1 to 10.

**[0119]** Example 14: The first or second memory (104, 108) of any of examples 1 to 10 including instructions that, when executed by the processor (106) or the resource management unit (102), cause the resource management unit (102) to perform the method of any of examples 1 to 10.

**[0120]** Example 15: The computing system (100; 300) of any of examples 1 to 10, further including means to perform the method of any of examples 1 to 10.

**[0121]** Example 16: The method of any of examples 1 to 10 wherein capturing the configuration state of the operating system includes obtaining operational status information of the operating system concerning at least one of a process run queue, a process wait queue, process priority data, process run-time remaining data, resource scheduling data, resource usage data, active device data, memory usage data, memory mapping data, virtual memory tables, storage management data, a status of the hypervisor, a status of the virtual machine, a status of a guest operating system in the virtual machine, resources of the computing system used by the hypervisor, resources of the computing system used by the virtual machine, or combinations thereof.

**[0122]** Example 17: The method of any of examples 1 to 10 wherein identifying the status of the process in the computing system includes accessing the first memory and identifying a status of at least one of a process run queue, process wait queue, process priority data, process run-time remaining data, resource scheduling data, active device data, memory usage data, memory mapping data, virtual memory tables, storage management data, hypervisor, virtual machine, the guest operating system in the virtual machine, resources of the computing system used by the hypervisor, resources of the computing system used by the virtual machine, or combinations thereof.

**[0123]** Example 18: The first and second memory (104, 108) of any of examples 1 to 9 comprising a shared memory and instructions that, when executed by the processor (106) or the resource management unit (102), cause the resource management unit (102) to perform the method of any of claims 1 to 10.

**CLAIMS**

1. A method of managing a computing system (100; 300), the method comprising:

capturing (112; 404; 508), by a resource management unit (102) and into a first memory (104), a configuration state (114) of an operating system (110) in a second memory (108), the operating system (110) executing on a processor (106) of the computing system;

identifying (118; 406; 1004), by the resource management unit (102), a status of a process (116) in the computing system (100; 300) based on the configuration state (114) of the operating system (110), the status of the process indicating a requirement to swap a memory resource in the computing system; and

responsive to identifying the status of the process indicating the requirement to swap the memory resource, swapping, by the resource management unit (102), the memory resource (120; 904; 1010) effective to alleviate the processor (106) from swapping the memory resource.

2. The method of claim 1, wherein swapping the memory resource comprises swapping a content of the memory resource between the second memory and a secondary storage.

3. The method of claim 2, wherein swapping the memory resource comprises swapping the content of the memory resource using a variant size of the content of the memory resource without a page-size calculation of the content.

4. The method of claims 2 or 3, wherein swapping the memory resource comprises swapping the content of the memory resource from the secondary storage into the second memory before execution of the process by the processor, the swapping for the process to use the content in the second memory during execution of the process by the processor.

5. The method of claim 1, wherein swapping the memory resource comprises swapping a content of the memory resource from the second memory to a secondary storage after execution of the process by the processor for a second process to use the second memory upon execution of the second process by the processor.

6. The method of any preceding claim further comprising the resource management unit communicating with the processor or the second memory for capturing the configuration state of the operating system using a low-latency communication data link or a high-speed interface bus, or by being integrated with a same integrated circuit die as the processor.

7. The method of any preceding claim wherein capturing the configuration state of the operating system comprises an abstraction layer of instructions coordinating with the operating system as executed by the processor for detecting a resource transaction event associated with the operating system and pushing the configuration state of the operating system to the first memory of the resource management unit responsive to the resource transaction event.

8. The method of any preceding claim wherein capturing the configuration state of the operating system comprises the resource management unit detecting a resource transaction event associated with the operating system and pulling the configuration state of the operating system to the first memory of the resource management unit responsive to the resource transaction event.

9. The method of any preceding claim wherein identifying the status of the process comprises identifying a memory usage, memory need, secondary storage, scheduling data, process run queue, process wait queue, process priority data, process run-time remaining, hypervisor, virtual machine, or combinations thereof.

10. The computing system (100; 300) of any of claims 1 to 9 comprising:  
the processor (106);  
the resource management unit (102); and  
the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of claims 1 to 9.



11. The computing system (100; 300) of any of claims 1 to 9 comprising:  
the processor (106);  
the resource management unit (102) comprising a system-on-a-chip, an application-specific integrated circuit, or an application-specific standard product; and  
the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of claims 1 to 9.

12. The computing system (100; 300) of any of claims 1 to 9 comprising:  
a hypervisor (136) or a virtual machine (138);  
the processor (106);  
the resource management unit (102); and  
the first or second memory (104; 108) having instructions stored thereon that, responsive to execution by the processor (106) or the resource management unit (102), cause the resource management unit (102) to execute the method of any of claims 1 to 9.

13. The first or second memory (104, 108) of any of claims 1 to 9 comprising instructions that, when executed by the processor (106) or the resource management unit (102), cause the resource management unit (102) to perform the method of any of claims 1 to 9.

14. The computing system (100; 300) of any of claims 1 to 9, further comprising means to perform the method of any of claims 1 to 9.

15. The first and second memory (104, 108) of any of claims 1 to 9 comprising a shared memory and instructions that, when executed by the processor (106) or the resource management unit (102), cause the resource management unit (102) to perform the method of any of claims 1 to 9.

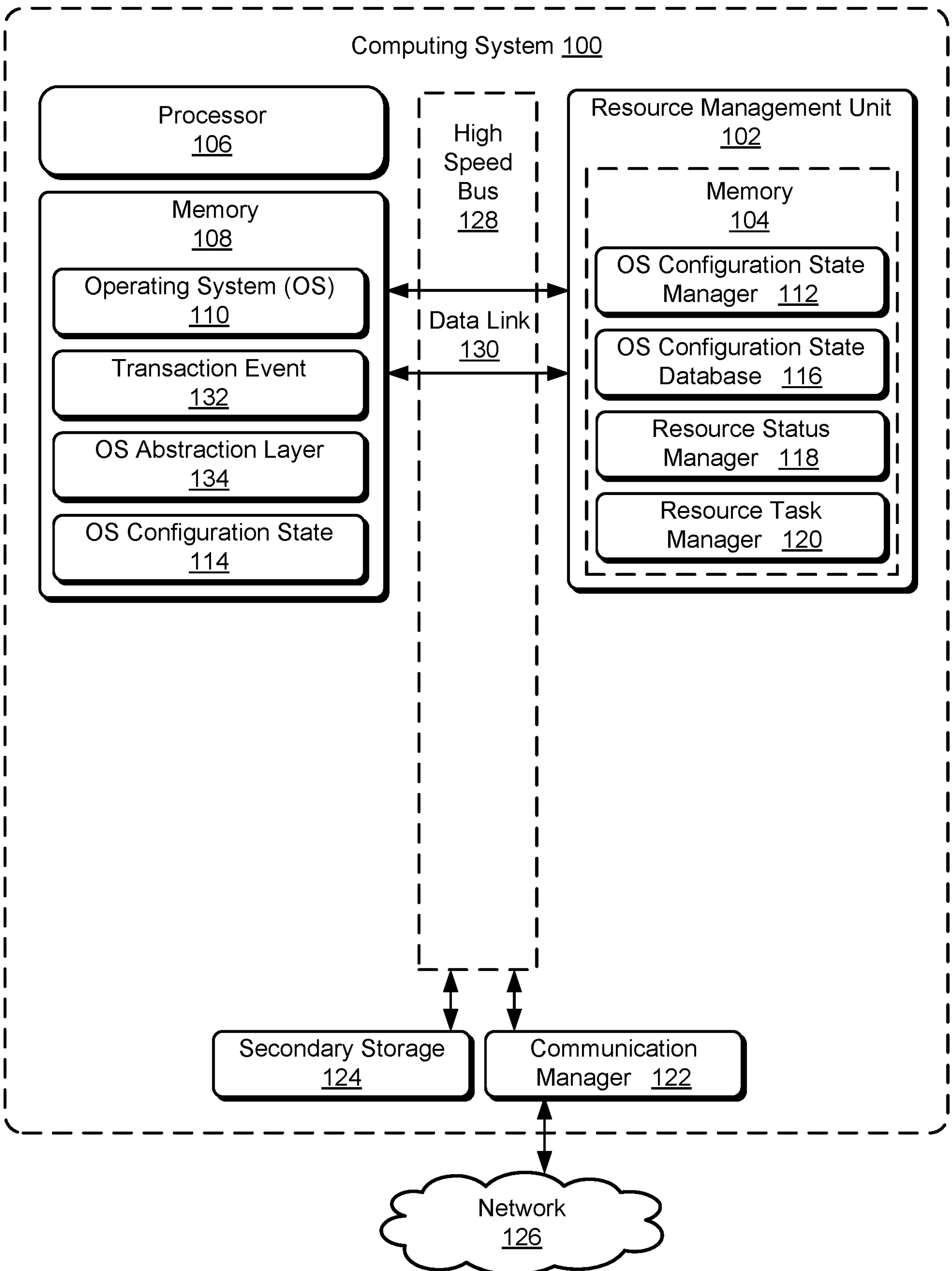


FIG. 1

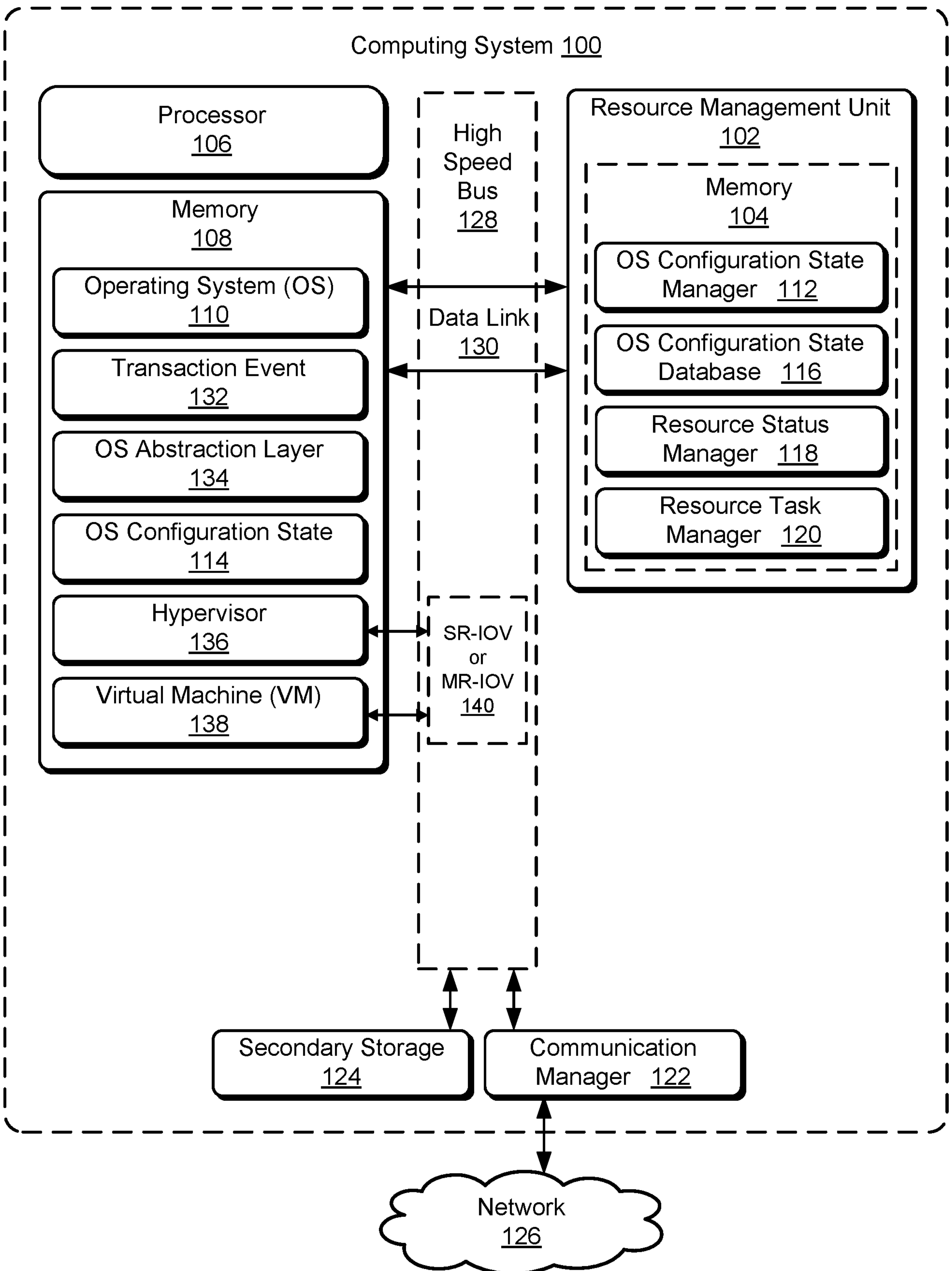


FIG. 2



300 →

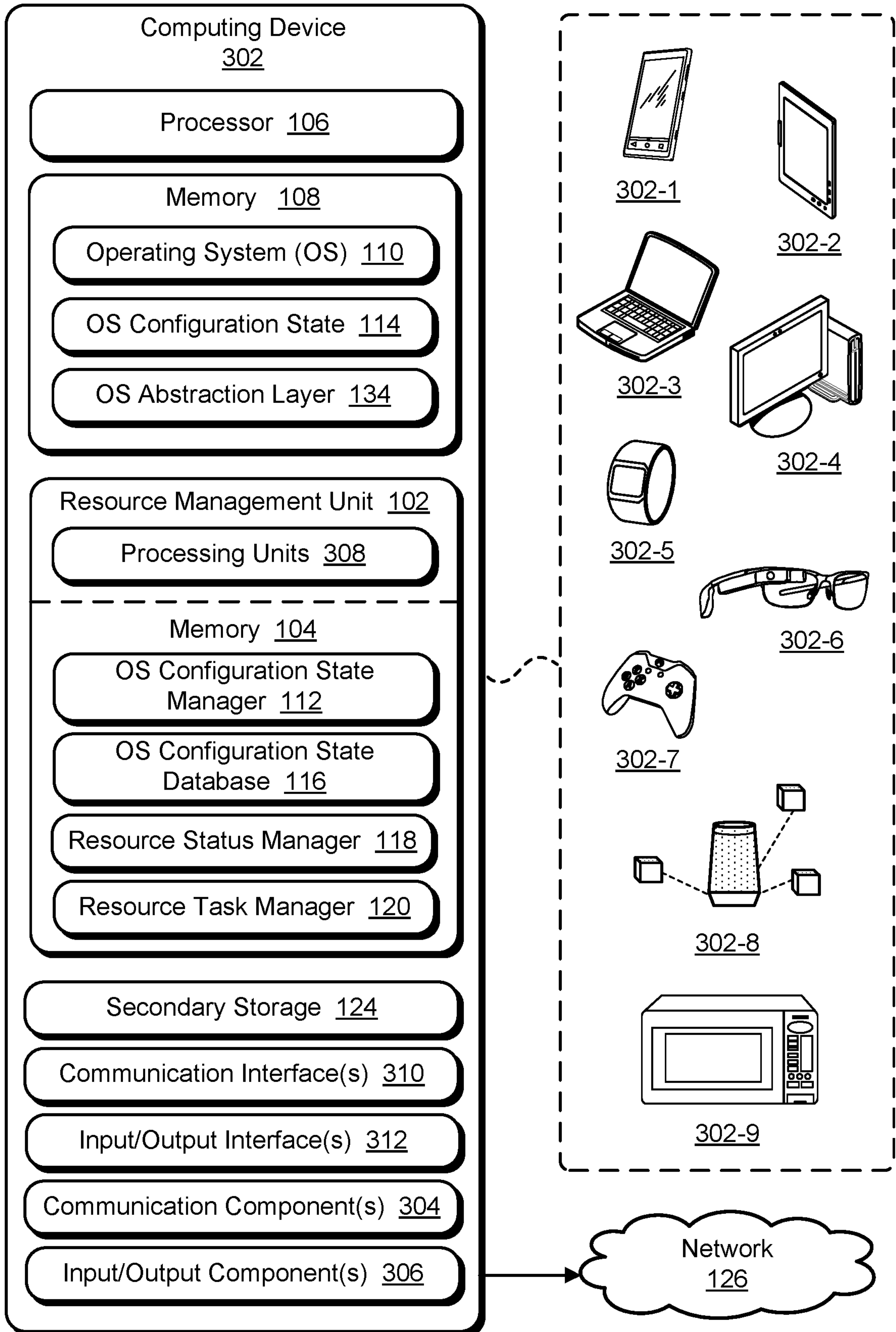
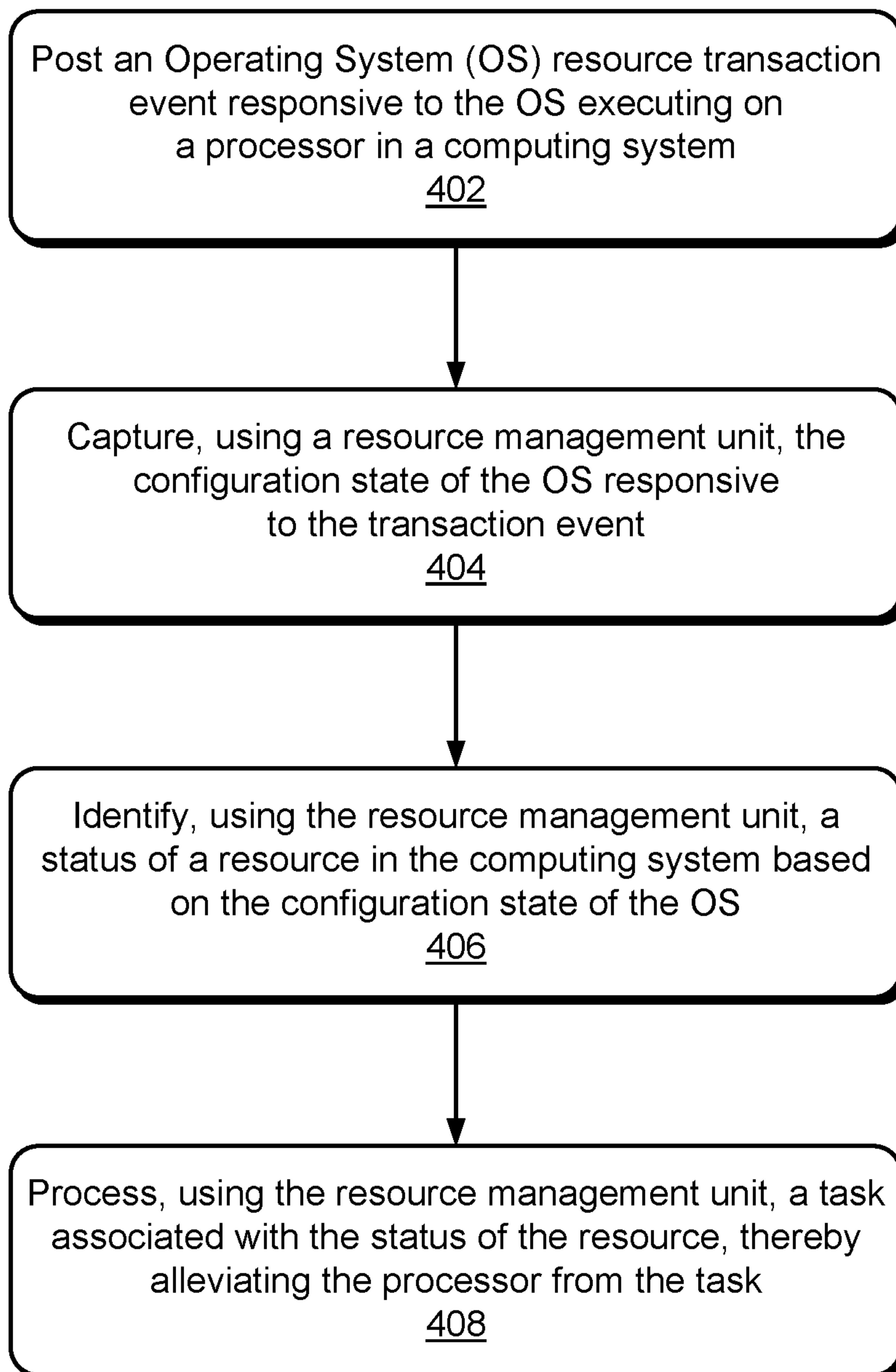



FIG. 3

4/12

400 *FIG. 4*

5/12

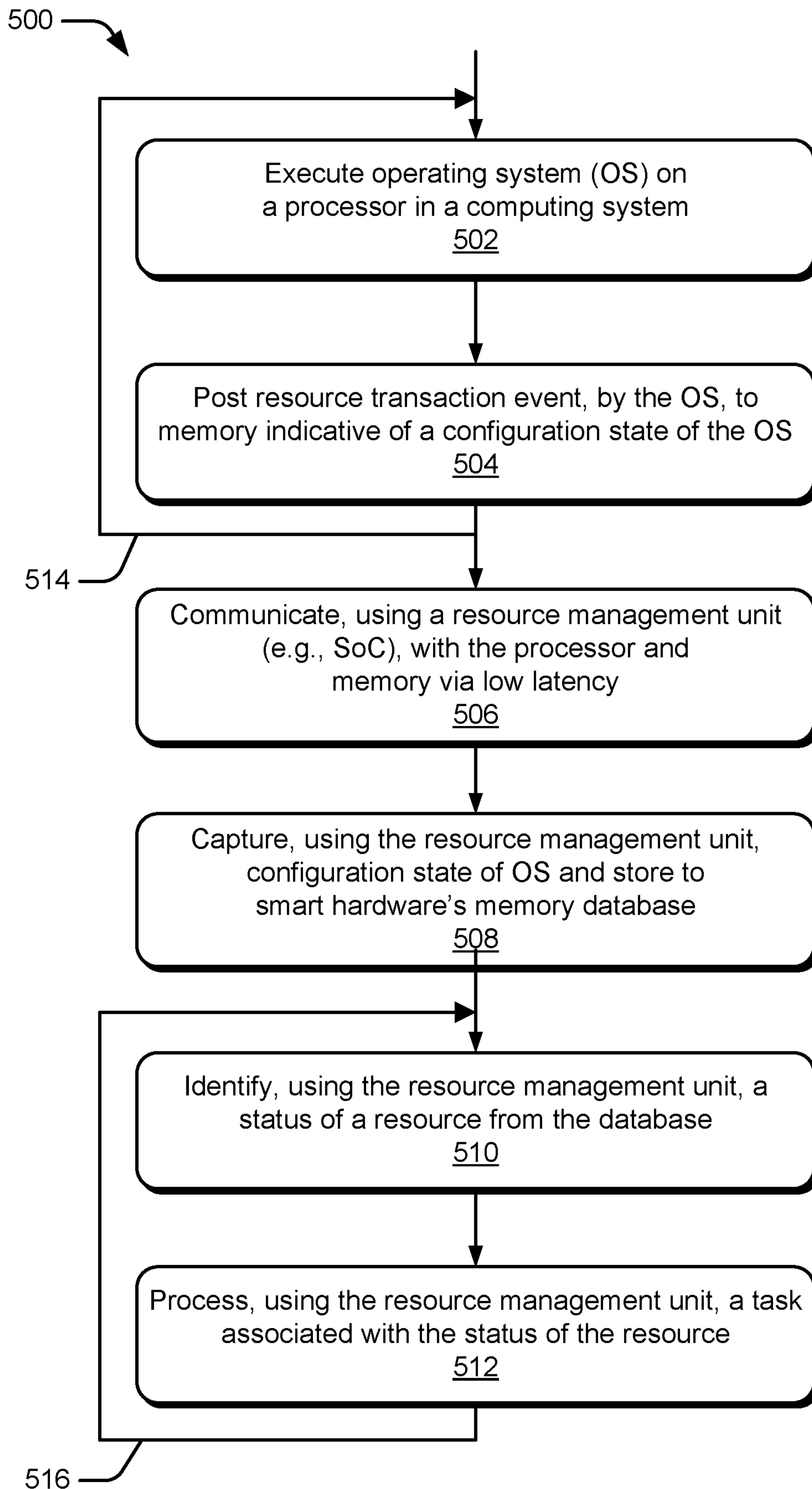


FIG. 5

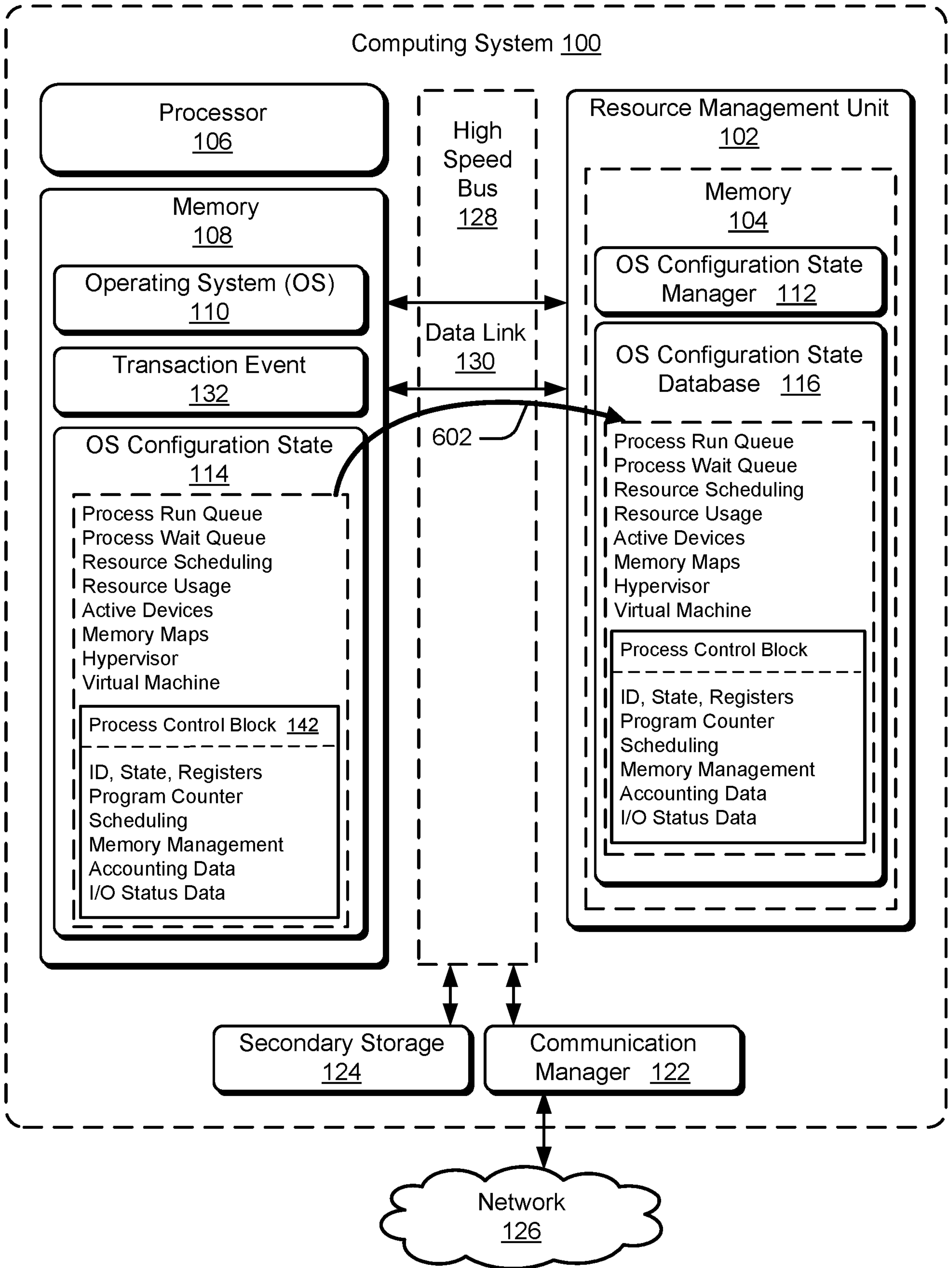


FIG. 6



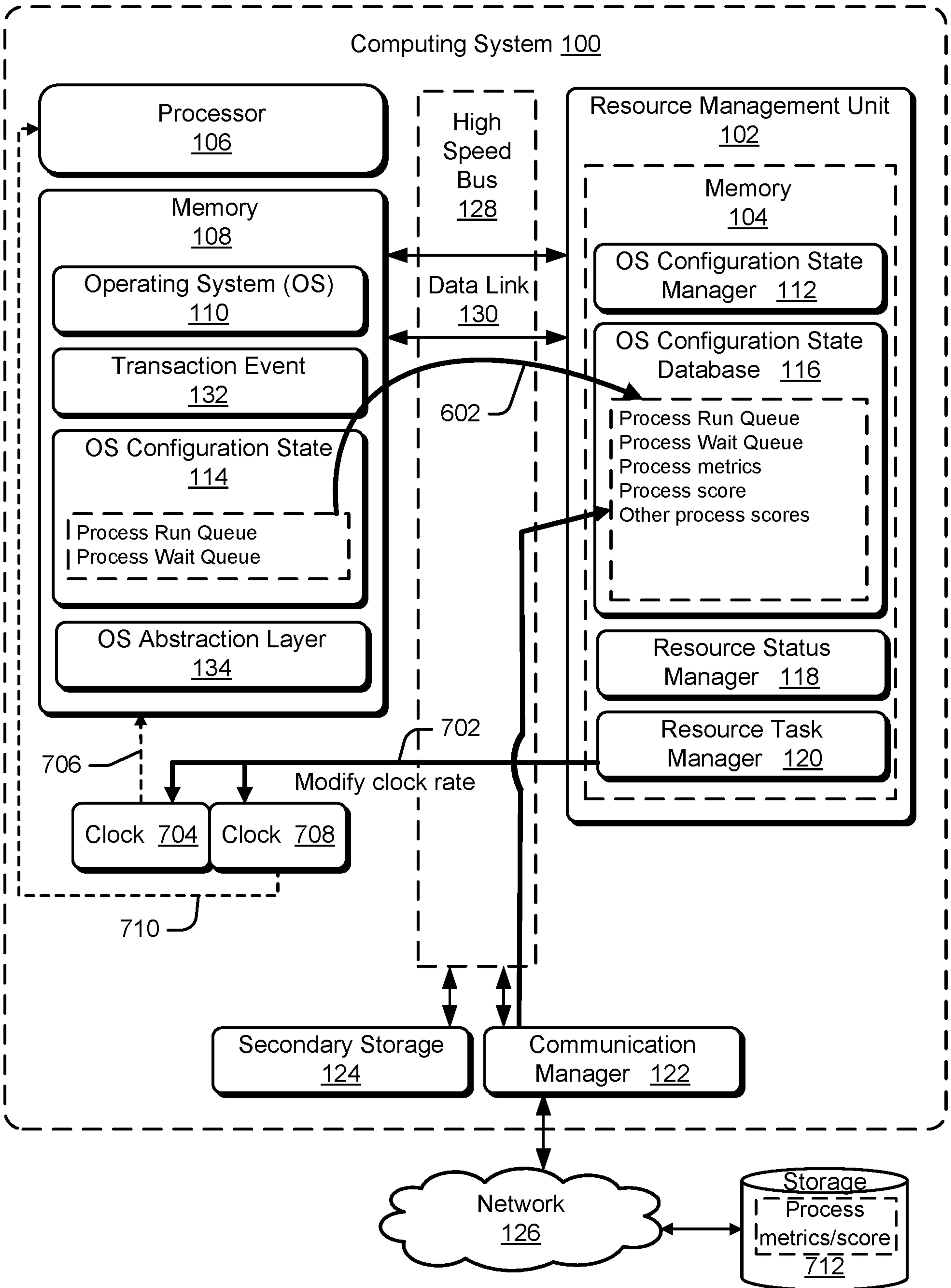


FIG. 7

8/12

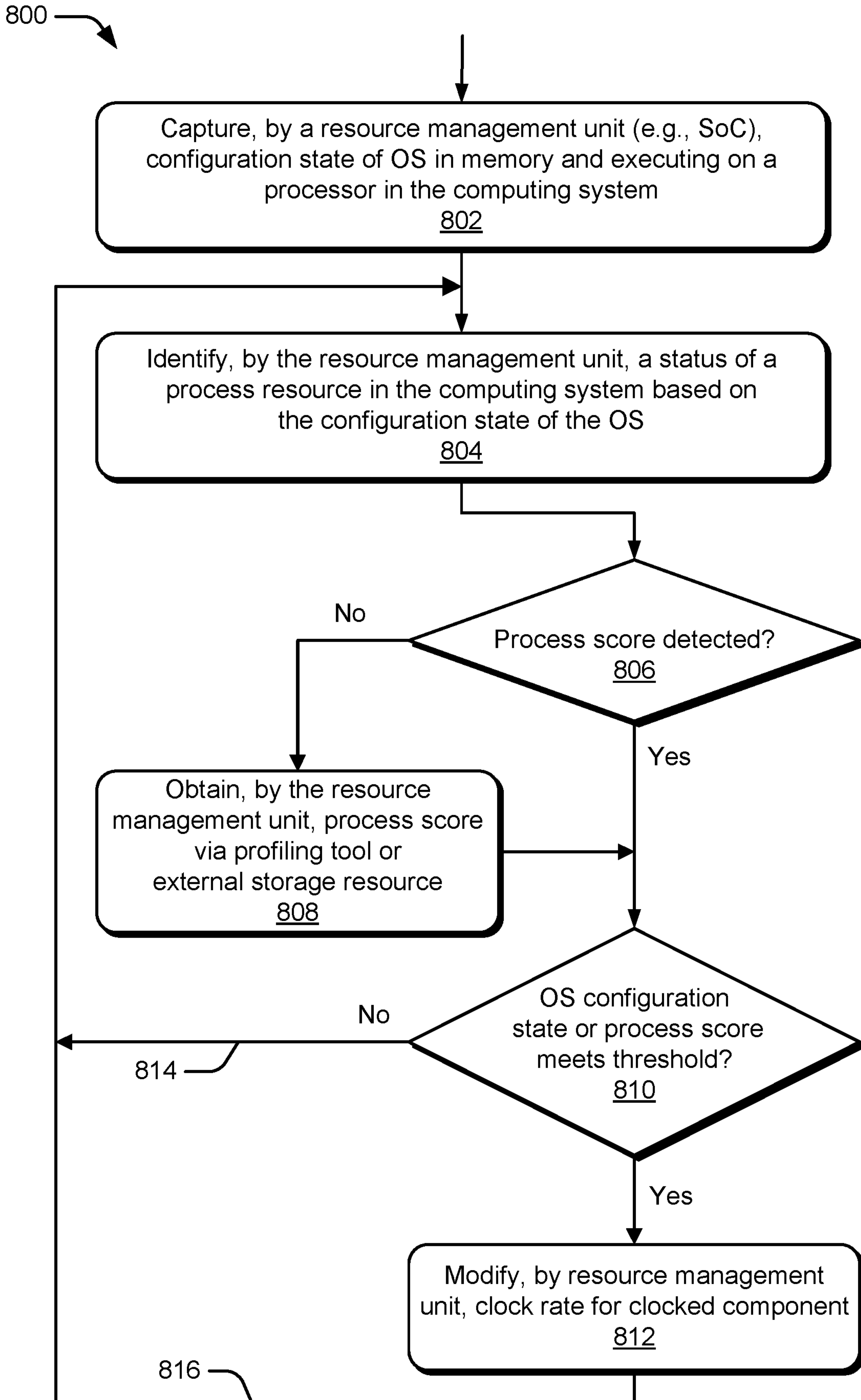


FIG. 8

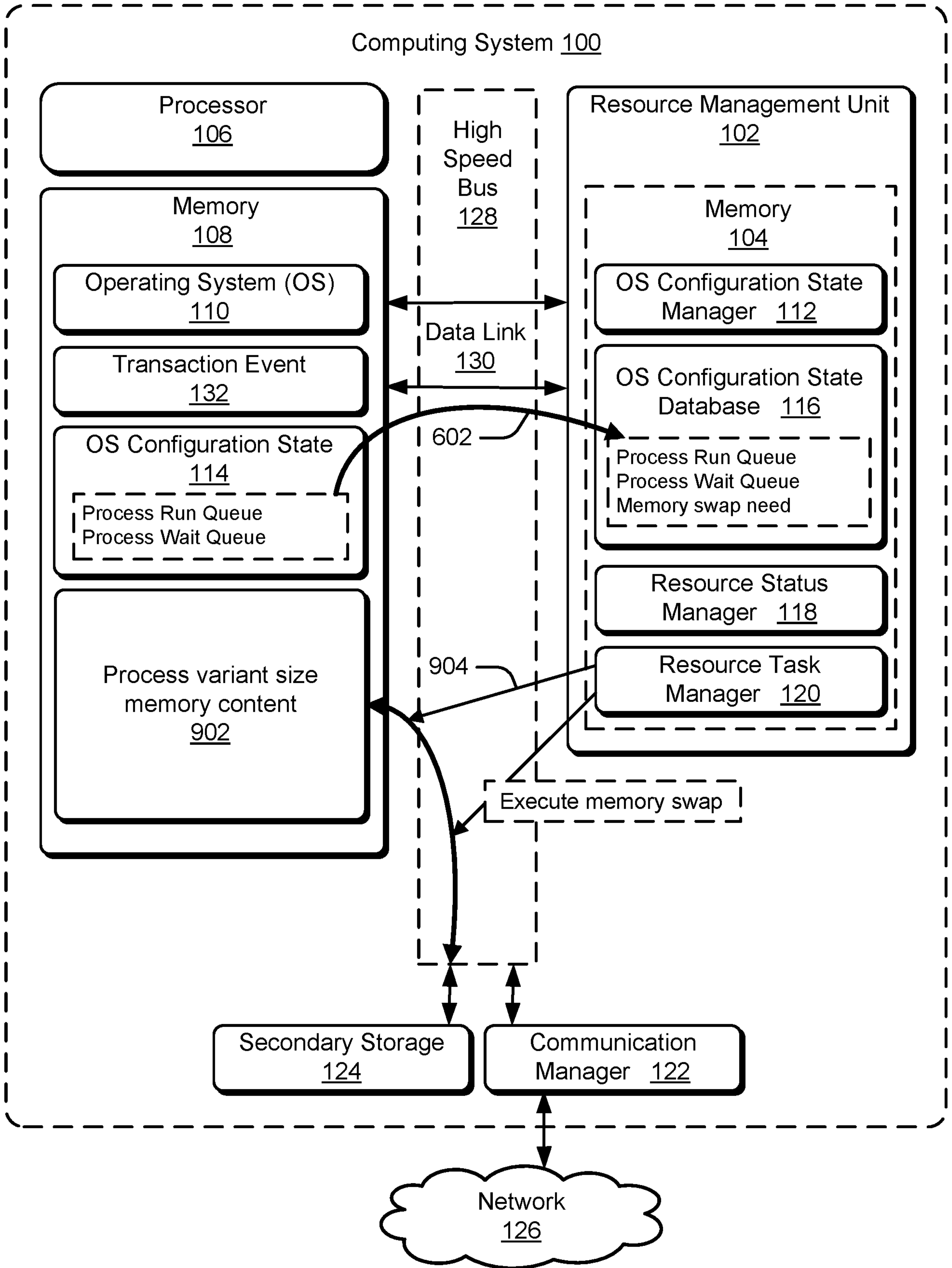
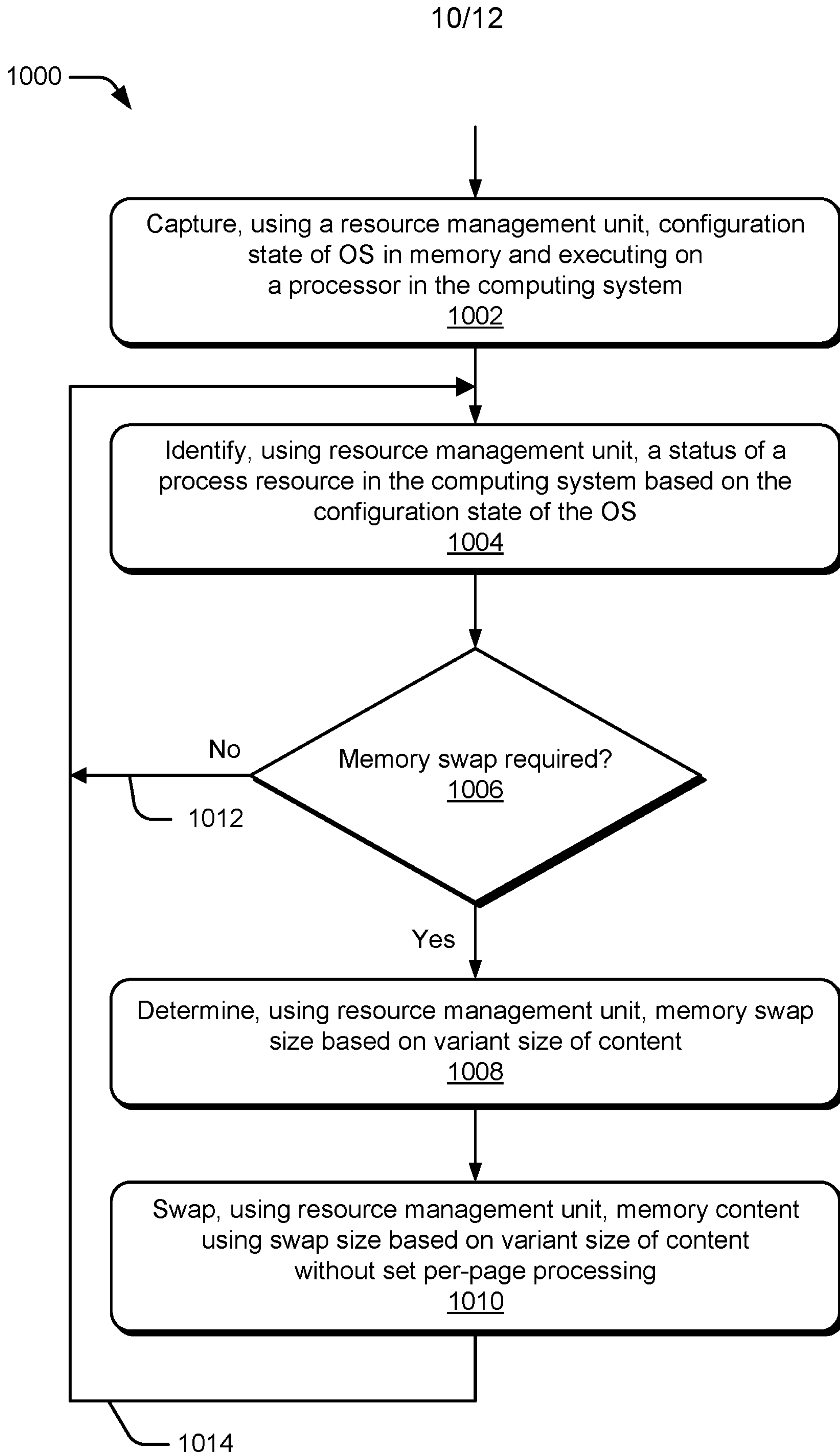


FIG. 9



**FIG. 10**



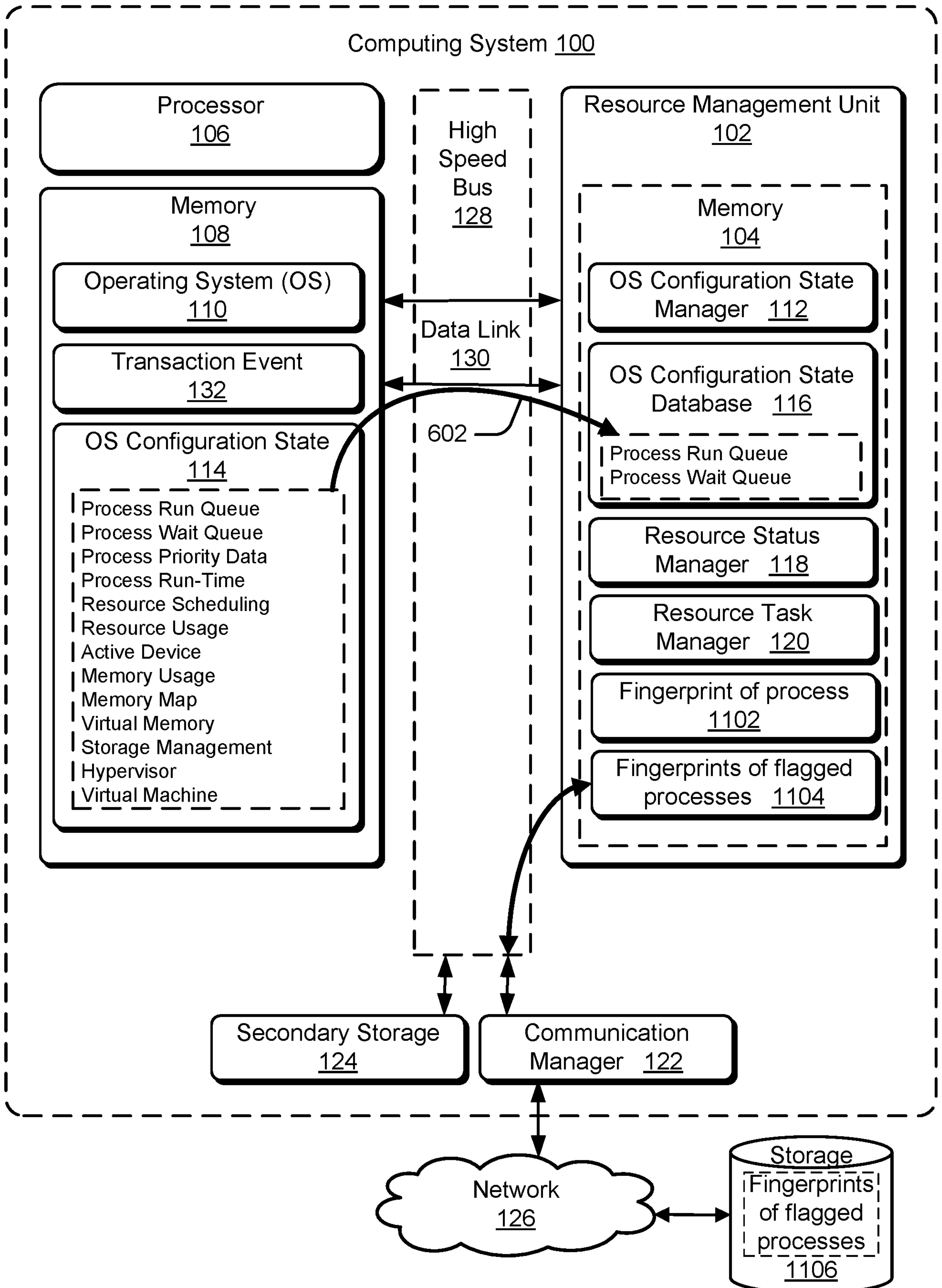
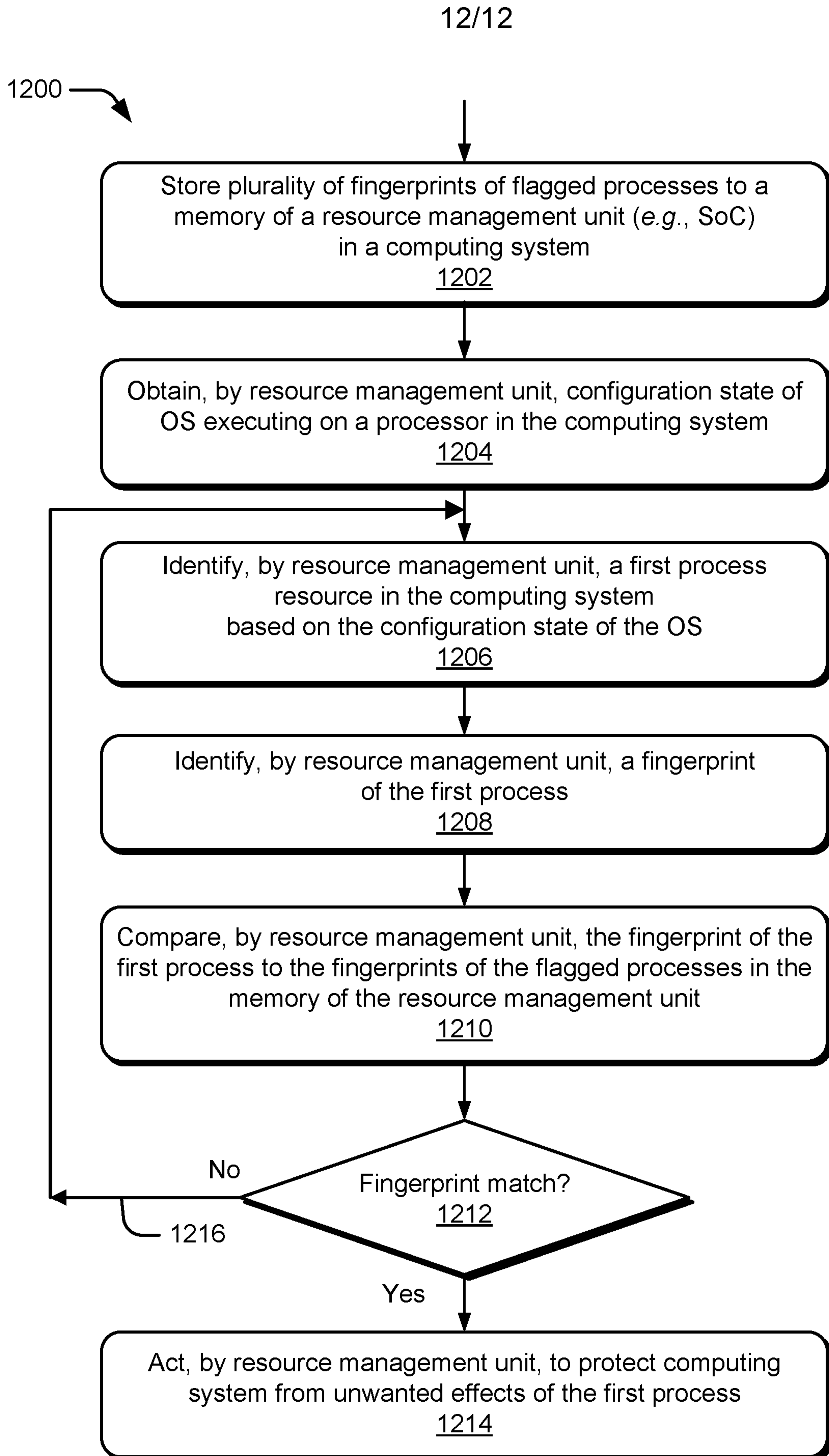


FIG. 11



**FIG. 12**

**INTERNATIONAL SEARCH REPORT**

International application No PCT/US2019/067560
---

**A. CLASSIFICATION OF SUBJECT MATTER**  
 INV. G06F9/50  
 ADD.  
 According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**  
 Minimum documentation searched (classification system followed by classification symbols)  
 G06F  
 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)  
 EPO-Internal, WPI Data

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2018/005826 A1 (AMAZON TECH INC [US]) 4 January 2018 (2018-01-04) paragraph [0029] paragraph [0038] paragraph [0081] paragraph [00124] -----	1-15

Further documents are listed in the continuation of Box C.       See patent family annex.

\* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E" earlier application or patent but published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	"&" document member of the same patent family
"P" document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search  3 September 2020	Date of mailing of the international search report  14/09/2020
---	--

Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer  Dewyn, Torkild
--	--

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2019/067560

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2018005826	A1	04-01-2018	
		AU 2017290252 A1	03-01-2019
		AU 2020202180 A1	16-04-2020
		CN 109564514 A	02-04-2019
		EP 3479224 A1	08-05-2019
		JP 6715356 B2	01-07-2020
		JP 2019525306 A	05-09-2019
		US 2018004539 A1	04-01-2018
		US 2019294446 A1	26-09-2019
		WO 2018005826 A1	04-01-2018

-----