(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 April 2012 (19.04.2012)

PCT

(10) International Publication Number
WO 2012/050934 A2

(51) International Patent Classification:
G06F 13/14 (2006.01) *G06F 13/42* (2006.01)
G06F 13/16 (2006.01)

(21) International Application Number:
PCT/US201 1/053792

(22) International Filing Date:
28 September 201 1 (28.09.201 1)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/387,400 28 September 2010 (28.09.2010) US

(71) Applicant *(for all designated States except US):* **FUSION-IO, INC.** [US/US]; 2855 E. Cottonwood Parkway Box 100, Salt Lake City, Utah 84121 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only):* **NELLANS, David** [US/US]; 1120 S. 1300 E., Salt Lake City, Utah 84105 (US). **WIPFEL, Robert** [US/US]; 2229 Kodiak Court, Draper, Utah 84020 (US).

(74) Agents: **HILTON, Scott** et al; 8 East Broadway Suite 600, Salt Lake City, Utah 841 11 (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, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ,

[Continued on next page]

(54) Title: APPARATUS, SYSTEM, AND METHOD FOR A DIRECT INTERFACE BETWEEN A MEMORY CONTROLLER AND NON-VOLATILE MEMORY USING A COMMAND PROTOCOL

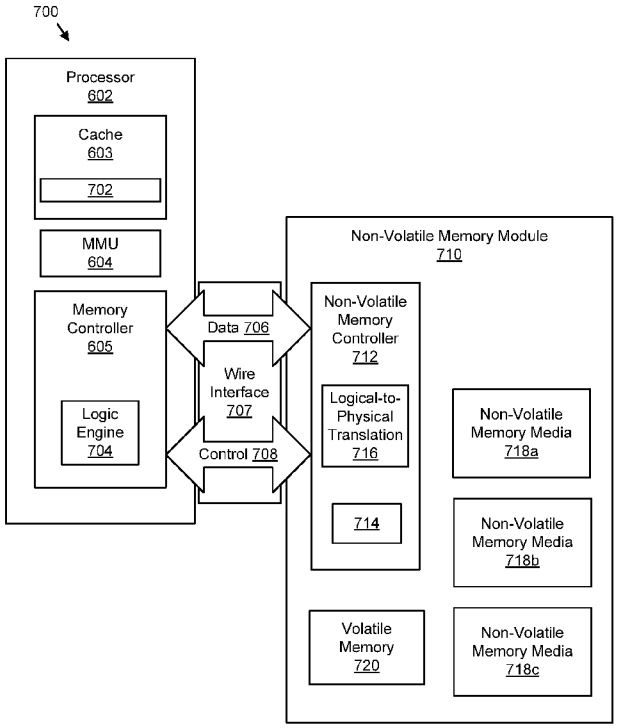

FIG. 4

(57) Abstract: A method for a direct interface between a memory controller 605 and a non-volatile memory controller 712 using a command protocol includes receiving a command from a memory controller 605 to a non-volatile memory controller 712 over a wire interface 707 by way of a command protocol. The memory controller 605 is coupled to one or more processors 602 and the non-volatile memory controller 712, in one embodiment, is coupled to non-volatile memory media 718. The command protocol includes a control path 708 that enables the memory controller 605 to distinguish among different memory modules. The non-volatile memory controller 712 stores data sequentially on the non-volatile memory media 718 to preserve an ordered sequence of memory operations performed on the non-volatile memory media 718. The method includes executing the command within the non-volatile memory controller 712 in response to determining that the non-volatile memory controller 712 is capable of satisfying the command.

WO 2012/050934 A2

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, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

APPARATUS, SYSTEM, AND METHOD FOR A DIRECT INTERFACE BETWEEN A MEMORY CONTROLLER AND A NON-VOLATILE MEMORY CONTROLLER USING A COMMAND PROTOCOL

FIELD OF THE INVENTION

This application relates to an interface between a processor memory controller and a non-volatile memory controller and more particular relates to a direct interface between a memory controller and a non-volatile memory controller using a command protocol

DESCRIPTION OF THE RELATED ART

In typical computing devices, main memory includes volatile memory such as dynamic random access memory ("DRAM") and static random access memory ("SRAM"). A processor typically communicates with the main memory over a wire interface using a low-level wire protocol such as the Joint Electron Devices Engineering Council ("JEDEC") protocol, the industry standard for processor - DRAM interfaces. The JEDEC standard assumes that physically addressable media is synchronous, heavily parallel, reliable and implements a design structure that is known to a processor memory controller. Consequently, JEDEC uses a series of distinct commands that cause the DRAM devices to execute known operations in hardware.

Recent significant development of flash-based devices enable use of non-volatile memory as a main memory replacement. However, typical non-volatile main memory solutions continue to provide communication between the processor and the non-volatile main memory using a low-level wire protocol such as JEDEC.

SUMMARY OF THE INVENTION

The present invention has been developed in response to the present state of the art, and in particular, in response to the problems and needs in the art that have not yet been fully solved by current memory controllers and main memory solutions. Accordingly, the present invention has been developed to provide an apparatus, system, and method for a direct interface between a memory controller and a non-volatile memory controller using a command protocol that overcome many or all of the above-discussed shortcomings in the art.

A method is presented for a direct interface between a memory controller and a non-volatile memory controller using a command protocol. In one embodiment, the method includes receiving a command from a memory controller to a non-volatile memory controller over a wire interface by way of a command protocol. The memory controller, in one embodiment, is coupled to one or more processors and the non-volatile memory controller, in one embodiment, is coupled to non-volatile memory media. The command protocol, in one embodiment, includes a control path that enables the memory controller to distinguish among different memory

modules. The non-volatile memory controller, in one embodiment, stores data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media. In one embodiment, the method includes executing the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

In a further embodiment, the command received includes a synchronous command and the method further includes executing the command asynchronously in response to determining that the non-volatile memory controller is capable of satisfying the command asynchronously. In one embodiment, the method includes signaling to the memory controller, using the command protocol, that the command will not be executed in response to determining that the non-volatile memory controller is not capable of satisfying the command.

In one embodiment, the memory controller communicates with volatile memory by way of a second protocol. The command protocol and the second protocol, in one embodiment, are different protocols. In a further embodiment, the method includes notifying the memory controller of memory attributes of the non-volatile memory. The memory controller, in one embodiment, directs data to the non-volatile memory, the volatile memory, a memory division on the non-volatile memory, or a memory division on the volatile memory based on the memory attributes based on the memory attributes.

In one embodiment, the second memory controller communicates with volatile memory by way of a second protocol. The second memory controller, in one embodiment, is coupled to one or more processors. The command protocol and the second protocol, in one embodiment, are different protocols. In a further embodiment, the second memory controller is coupled to a second processor and the second processor is coupled to the processor such that memory accessible to the second memory controller is accessible to the processor and the second processor and memory accessible to the memory controller is accessible to the processor and the second processor.

In one embodiment, the method includes associating sequence indicators with the data on the non-volatile memory media. The sequence indicators, in one embodiment, determine an ordered sequence of memory operations performed on the non-volatile memory media. In one embodiment, the method includes associating checkpoint information with the data on the non-volatile memory media, wherein the checkpoint information is organized in an ordered sequence of memory checkpoint operations performed on the non-volatile memory media.

In one embodiment, the method includes storing the data in a format that associates the data with respective logical memory addresses on the non-volatile memory media. In a further

embodiment, the method includes maintaining an index of associations between logical memory addresses of the data and physical storage memory locations comprising the data on the non-volatile memory media. In yet a further embodiment, the method includes reconstructing the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media. Reconstructing the index, in one embodiment, includes replaying a sequence of changes made to the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media. In a further embodiment, reconstructing the index includes rolling back a sequence of changes made to the index from a last change back to a valid checkpoint indicator using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media.

In one embodiment, the command includes a read command, a write command, or a memory management command. The memory management command, in one embodiment, includes a query command, a directive command, and/or a hint command. In one embodiment, the method includes operating one or more memory maintenance functions on the non-volatile memory to optimize non-volatile memory performance. The non-volatile memory controller, in one embodiment, operates the one or more memory maintenance functions independent of the memory controller or in response to receiving memory management commands from the memory controller.

An apparatus for a direct interface between a memory controller and a non-volatile memory controller using a command protocol is provided with a plurality of modules. These modules in the described embodiments include one or more of a receiving module, an execution module, a notification module, a sequence indicator module, a checkpoint module, a storage module, an index module, and an index reconstruction module.

In one embodiment, the receiving module receives a command from a memory controller to a non-volatile memory controller over a wire interface by way of a command protocol. The memory controller, in one embodiment, is coupled to one or more processors. The non-volatile memory controller, in one embodiment, is coupled to non-volatile memory media. The command protocol, in one embodiment, includes a control path that enables the memory controller to distinguish among different memory modules. The non-volatile memory controller, in one embodiment, stores data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media. In one embodiment, the execution module executes the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

In a further embodiment, the memory controller communicates with volatile memory by way of a second protocol. The command protocol and the second protocol, in one embodiment, are different protocols. In one embodiment, the notification module notifies the memory controller of memory attributes of the non-volatile memory. The memory controller, in one embodiment, directs data to the non-volatile memory, the volatile memory, a memory division on the non-volatile memory, or a memory division on the volatile memory based on the memory attributes based on the memory attributes.

In one embodiment, the checkpoint module associates checkpoint information with the data on the non-volatile memory media. The checkpoint information, in one embodiment, is organized in an ordered sequence of memory checkpoint operations performed on the non-volatile memory media.

In one embodiment, the storage module stores the data in a format that associates the data with respective logical memory addresses on the non-volatile memory media. In a further embodiment, the index module maintains an index of associations between logical memory addresses of the data and physical storage locations including the data on the non-volatile memory media. In a further embodiment, the index reconstruction module reconstructs the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media. The index reconstruction module, in one embodiment, replays a sequence of changes made to the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media. In one embodiment, the storage module further associates sequence indicators with the data on the non-volatile memory media. The sequence indicators, in one embodiment, determine an ordered sequence of memory operations performed on the non-volatile memory media.

A system is also presented for a direct interface between a memory controller and a non-volatile memory controller using a command protocol. The system, in one embodiment, is embodied by one or more processors, a memory controller coupled to the one or more processors, and a non-volatile memory controller coupled to non-volatile memory media. The memory controller, in one embodiment, generates a command in response to a request from a client. In one embodiment, the memory controller communicates the command over a wire interface by way of a command protocol. In one embodiment, the command protocol includes a control path that enables the memory controller to distinguish among different memory modules. In one embodiment, the non-volatile memory controller receives the command from the memory controller and stores data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media. In one

embodiment, the non-volatile memory controller executes the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

Reference throughout this specification to features, advantages, or similar language does not imply that all of the features and advantages that may be realized with the present invention should be or are in any single embodiment of the invention. Rather, language referring to the features and advantages is understood to mean that a specific feature, advantage, or characteristic described in connection with an embodiment is included in at least one embodiment of the present invention. Thus, discussion of the features and advantages, and similar language, throughout this specification may, but do not necessarily, refer to the same embodiment.

These features and advantages of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the advantages of the invention will be readily understood, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments that are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings, in which:

Figure 1 is a schematic block diagram illustrating one embodiment of a solid-state storage system in accordance with the present invention;

Figure 2 is a schematic block diagram illustrating one embodiment of a logical representation of a solid-state storage controller with a logical-to-physical translation layer in accordance with the present invention;

Figure 3 is a schematic block diagram illustrating one embodiment of a computing device in accordance with the present invention;

Figure 4 is a schematic block diagram illustrating one embodiment of a system with direct interface between a memory controller and non-volatile memory in accordance with the present invention;

Figure 5 is a schematic block diagram illustrating a logical representation of one embodiment of a plurality of communication layers between a client and non-volatile memory in accordance with the present invention;

Figure 6 is a schematic block diagram illustrating one embodiment of a system with a

plurality of memory controllers communicating with a plurality of memory modules in accordance with the present invention;

Figure 7A is a schematic block diagram illustrating one embodiment of a system with a memory controller communicating with a plurality of memory modules in accordance with the present invention;

Figure 7B is a schematic block diagram illustrating one embodiment of a system with a plurality of memory controllers in a processor communicating with a plurality of memory modules in accordance with the present invention;

Figure 8 is a schematic block diagram illustrating one embodiment of a non-volatile memory controller in accordance with the present invention;

Figure 9 is a schematic block diagram illustrating another embodiment of a non-volatile memory controller in accordance with the present invention;

Figure 10 is a schematic flow chart diagram illustrating one embodiment of a method for a direct interface between a memory controller and a non-volatile memory controller using a command protocol in accordance with the present invention; and

Figure 11 is a schematic flow chart diagram illustrating another embodiment of a method for a direct interface between a memory controller and a non-volatile memory controller using a command protocol in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

SOLID-STATE STORAGE SYSTEM

Figure 1 is a schematic block diagram illustrating one embodiment of a system 100 for improving performance in a solid-state storage device in accordance with the present invention. The system 100 includes a solid-state storage device 102, a solid-state storage controller 104, a write data pipeline 106, a read data pipeline 108, a solid-state storage 110, a computer 112, a client 114, and a computer network 116, which are described below.

The system 100 includes at least one solid-state storage device 102. In another

embodiment, the system 100 includes two or more solid-state storage devices 102. Each solid-state storage device 102 may include non-volatile, solid-state storage 110, such as flash memory, nano random access memory ("nano RAM or NRAM"), magneto-resistive RAM ("MRAM"), dynamic RAM ("DRAM"), phase change RAM ("PRAM"), etc. In further embodiments, the data storage device 102 may include other types of non-volatile and/or volatile data storage, such as dynamic RAM ("DRAM"), static RAM ("SRAM"), magnetic data storage, optical data storage, and/or other data storage technologies.

The solid-state storage device 102 is depicted in a computer 112 connected to a client 114 through a computer network 116. In one embodiment, the solid-state storage device 102 is internal to the computer 112 and is connected using a system bus, such as a peripheral component interconnect express ("PCI-e") bus, a Serial Advanced Technology Attachment ("serial ATA") bus, or the like. In another embodiment, the solid-state storage device 102 is external to the computer 112 and is connected, a universal serial bus ("USB") connection, an Institute of Electrical and Electronics Engineers ("IEEE") 1394 bus ("FireWire"), or the like. In other embodiments, the solid-state storage device 102 is connected to the computer 112 using a peripheral component interconnect ("PCI") express bus using external electrical or optical bus extension or bus networking solution such as Infiniband or PCI Express Advanced Switching ("PCIe-AS"), or the like.

In various embodiments, the solid-state storage device 102 may be in the form of a dual-inline memory module ("DIMM"), a daughter card, or a micro-module. In another embodiment, the solid-state storage device 102 is an element within a rack-mounted blade. In another embodiment, the solid-state storage device 102 is contained within a package that is integrated directly onto a higher level assembly (e.g. mother board, lap top, graphics processor). In another embodiment, individual components comprising the solid-state storage device 102 are integrated directly onto a higher level assembly without intermediate packaging.

The solid-state storage device 102 includes one or more solid-state storage controllers 104, each may include a write data pipeline 106 and a read data pipeline 108 and each includes a solid-state storage 110. In one embodiment, the storage controller 104 sequentially writes data on the solid-state storage media 110 in a log structured format and within one or more physical structures of the storage elements, the data is sequentially stored on the solid-state storage media 110. Sequentially writing data involves the storage controller 104 streaming data packets into storage write buffers for storage elements, such as a chip (a package of one or more dies) or a die on a circuit board. When the storage write buffers are full, the data packets are programmed to a designated virtual or logical page ("LP"). Data packets then refill the storage write buffers and,

when full, the data packets are written to the next LP. The next virtual page may be in the same bank or another bank. This process continues, LP after LP, typically until a virtual or logical erase block ("LEB") is filled.

In another embodiment, the streaming may continue across LEB boundaries with the process continuing, LEB after LEB. Typically, the storage controller 104 sequentially stores data packets in an LEB by order of processing. In one embodiment, where a write data pipeline 106 is used, the storage controller 104 stores packets in the order that they come out of the write data pipeline 106. This order may be a result of data segments arriving from a requesting device mixed with packets of valid data that are being read from another storage location as valid data is being recovered from another LEB during a recovery operation.

The sequentially stored data, in one embodiment, can serve as a log to reconstruct data indexes and other metadata using information from data packet headers. For example, in one embodiment, the storage controller 104 may reconstruct a storage index by reading headers to determine the data structure to which each packet belongs and sequence information to determine where in the data structure the data or metadata belongs. The storage controller 104, in one embodiment, uses physical address information for each packet and timestamp or sequence information to create a mapping between the physical locations of the packets and the data structure identifier and data segment sequence. Timestamp or sequence information is used by the storage controller 104 to replay the sequence of changes made to the index and thereby reestablish the most recent state.

In one embodiment, erase blocks are time stamped or given a sequence number as packets are written and the timestamp or sequence information of an erase block is used along with information gathered from container headers and packet headers to reconstruct the storage index. In another embodiment, timestamp or sequence information is written to an erase block when the erase block is recovered.

In a read, modify, write operation, data packets associated with the logical structure are located and read in a read operation. Data segments of the modified structure that have been modified are not written to the location from which they are read. Instead, the modified data segments are again converted to data packets and then written to the next available location in the virtual page currently being written. Index entries for the respective data packets are modified to point to the packets that contain the modified data segments. The entry or entries in the index for data packets associated with the same logical structure that have not been modified will include pointers to original location of the unmodified data packets. Thus, if the original logical structure is maintained, for example to maintain a previous version of the logical

structure, the original logical structure will have pointers in the index to all data packets as originally written. The new logical structure will have pointers in the index to some of the original data packets and pointers to the modified data packets in the virtual page that is currently being written.

In a copy operation, the index includes an entry for the original logical structure mapped to a number of packets stored on the solid-state storage media 110. When a copy is made, a new logical structure is created and a new entry is created in the index mapping the new logical structure to the original packets. The new logical structure is also written to the solid-state storage media 110 with its location mapped to the new entry in the index. The new logical structure packets may be used to identify the packets within the original logical structure that are referenced in case changes have been made in the original logical structure that have not been propagated to the copy and the index is lost or corrupted. In another embodiment, the index includes a logical entry for a logical block.

Beneficially, sequentially writing packets facilitates a more even use of the solid-state storage media 110 and allows a solid-storage device controller to monitor storage hot spots and level usage of the various virtual pages in the solid-state storage media 110. Sequentially writing packets also facilitates a powerful, efficient garbage collection system, which is described in detail below. One of skill in the art will recognize other benefits of sequential storage of data packets.

The system 100 may comprise a log-structured storage system or log-structured array similar to a log-structured file system and the order that data is stored may be used to recreate an index. Typically an index that includes a logical-to-physical mapping is stored in volatile memory. If the index is corrupted or lost, the index may be reconstructed by addressing the solid-state storage media 110 in the order that the data was written. Within a logical erase block ("LEB"), data is typically stored sequentially by filling a first logical page, then a second logical page, etc. until the LEB is filled. The solid-state storage controller 104 then chooses another LEB and the process repeats. By maintaining an order that the LEBs were written to and by knowing that each LEB is written sequentially, the index can be rebuilt by traversing the solid-state storage media 110 in order from beginning to end. In other embodiments, if part of the index is stored in non-volatile memory, such as on the solid-state storage media 110, the solid-state storage controller 104 may only need to replay a portion of the solid-state storage media 110 to rebuild a portion of the index that was not stored in non-volatile memory. One of skill in the art will recognize other benefits of sequential storage of data packets.

The system 100 includes one or more computers 112 connected to the solid-state storage

device 102. A computer 112 may be a host, a server, a storage controller of a storage area network ("SAN"), a workstation, a personal computer, a laptop computer, a handheld computer, a supercomputer, a computer cluster, a network switch, router, or appliance, a database or storage appliance, a data acquisition or data capture system, a diagnostic system, a test system, a robot, a portable electronic device, a wireless device, or the like. In another embodiment, a computer 112 may be a client and the solid-state storage device 102 operates autonomously to service data requests sent from the computer 112. In this embodiment, the computer 112 and solid-state storage device 102 may be connected using a computer network, system bus, Direct Attached Storage (DAS) or other communication means suitable for connection between a computer 112 and an autonomous solid-state storage device 102.

In one embodiment, the system 100 includes one or more clients 114 connected to one or more computer 112 through one or more computer networks 116. A client 114 may be a host, a server, a storage controller of a SAN, a workstation, a personal computer, a laptop computer, a handheld computer, a supercomputer, a computer cluster, a network switch, router, or appliance, a database or storage appliance, a data acquisition or data capture system, a diagnostic system, a test system, a robot, a portable electronic device, a wireless device, or the like. The computer network 116 may include the Internet, a wide area network ("WAN"), a metropolitan area network ("MAN"), a local area network ("LAN"), a token ring, a wireless network, a fiber channel network, a SAN, network attached storage ("NAS"), ESCON, or the like, or any combination of networks. The computer network 116 may also include a network from the IEEE 802 family of network technologies, such Ethernet, token ring, WiFi, WiMax, and the like.

In a further embodiment, instead of being connected directly to the computer 112 as DAS, the data storage device 102 may be connected to the computer 112 over a data network. For example, the data storage device 102 may include a storage area network ("SAN") storage device, a network attached storage ("NAS") device, a network share, or the like. In one embodiment, the system 100 may include a data network, such as the Internet, a wide area network ("WAN"), a metropolitan area network ("MAN"), a local area network ("LAN"), a token ring, a wireless network, a fiber channel network, a SAN, a NAS, ESCON, or the like, or any combination of networks. A data network may also include a network from the IEEE 802 family of network technologies, such Ethernet, token ring, Wi-Fi, Wi-Max, and the like. A data network may include servers, switches, routers, cabling, radios, and other equipment used to facilitate networking between the computer 112 and the data storage device 102.

The computer network 116 may include servers, switches, routers, cabling, radios, and other equipment used to facilitate networking computers 112 and clients 114. In one

embodiment, the system 100 includes multiple computers 112 that communicate as peers over a computer network 116. In another embodiment, the system 100 includes multiple solid-state storage devices 102 that communicate as peers over a computer network 116. One of skill in the art will recognize other computer networks 116 comprising one or more computer networks 116 and related equipment with single or redundant connection between one or more clients 114 or other computer with one or more solid-state storage devices 102 or one or more solid-state storage devices 102 connected to one or more computers 112. In one embodiment, the system 100 includes two or more solid-state storage devices 102 connected through the computer network 116 to a client 114 without a computer 112.

In one embodiment, the data storage device 102 has a block device interface that support block device commands. For example, the first data storage device 102 may support the ATA interface standard, the ATA Packet Interface ("ATAPI") standard, the small computer system interface ("SCSI") standard, and/or the Fibre Channel standard which are maintained by the InterNational Committee for Information Technology Standards ("INCITS").

LOGICAL-TO-PHYSICAL TRANSLATION AND DEALLOCATION

Figure 2 is a schematic block diagram illustrating a logical representation 500 of a solid-state storage controller 506 with a logical-to-physical translation layer 512 in accordance with the present invention. The storage controller 506 may be similar, in certain embodiments, to the solid-state storage controller 104 depicted in Figure 1 and may include one or more solid-state storage controllers 104. The depicted embodiment shows a user application 502 in communication with a storage client 504. The storage client 504 is in communication with a storage controller 506 that includes the logical-to-physical translation layer 512, an ECC correction module 514, a read data pipeline 516, and a write data pipeline 518.

The storage controller 506 manages a solid-state storage array 522. The storage controller 506 may include various hardware and software controllers, drivers, and software, such as the depicted hardware controllers 520.

In one embodiment, the user application 502 is a software application operating on or in conjunction with the storage client 504. The storage client 504 manages files and data and utilizes the functions and features of the storage controller 506 and associated solid-state storage array 522. Representative examples of storage clients include, but are not limited to, a server, a file system, an operating system, a database management system ("DBMS"), a volume manager, and the like. The storage client 504 is in communication with the storage controller 506. In one embodiment, the storage client 504 communicates through an Input/Output (I/O) interface represented by a block I/O emulation layer 508.

Certain conventional block storage devices divide the storage media into volumes or partitions. Each volume or partition may include a plurality of sectors. One or more sectors are organized into a logical block. In certain storage systems, such as those interfacing with the Windows® operating systems, the logical blocks are referred to as clusters. In other storage systems, such as those interfacing with UNIX, Linux, or similar operating systems, the logical blocks are referred to simply as blocks. A logical block or cluster represents a smallest physical amount of storage space on the storage media that is managed by the storage manager. A block storage device may associate n logical blocks available for user data storage across the storage media with a logical block address, numbered from 0 to n. In certain block storage devices, the logical block addresses may range from 0 to n per volume or partition. In conventional block storage devices, a logical block address maps directly to a particular logical block. In conventional block storage devices, each logical block maps to a particular set of physical sectors on the storage media.

However, certain storage devices 102 do not directly or necessarily associate logical block addresses with particular physical blocks. These storage devices 102 may emulate a conventional block storage interface to maintain compatibility with block storage clients 504.

When the storage client 504 communicates through the block I/O emulation layer 508, the storage device 102 appears to the storage client 504 as a conventional block storage device. In one embodiment, the storage controller 506 provides a block I/O emulation layer 508 which serves as a block device interface, or API. In this embodiment, the storage client 504 communicates with the storage device 102 through this block device interface. In one embodiment, the block I/O emulation layer 508 receives commands and logical block addresses from the storage client 504 in accordance with this block device interface. As a result, the block I/O emulation layer 508 provides the storage device 102 compatibility with block storage clients 504.

In one embodiment, a storage client 504 communicates with the storage controller 506 through a direct interface layer 510. In this embodiment, the storage device 102 directly exchanges information specific to non-volatile storage devices. A storage device 102 using direct interface 510 may store data on the solid-state storage media 110 as blocks, sectors, pages, logical blocks, logical pages, erase blocks, logical erase blocks, ECC chunks, logical ECC chunks, or in any other format or structure advantageous to the technical characteristics of the solid-state storage media 110. The storage controller 506 receives a logical address and a command from the storage client 504 and performs the corresponding operation in relation to the non-volatile solid-state storage media 110. The storage controller 506 may support a block I/O

emulation layer 508, a direct interface 510, or both a block I/O emulation layer 508 and a direct interface 510.

As described above, certain storage devices, while appearing to a storage client 504 to be a block storage device, do not directly associate particular logical block addresses with particular physical blocks, also referred to in the art as sectors. Such storage devices may use a logical-to-physical translation layer 512. The logical-to-physical translation layer 512 provides a level of abstraction between the logical block addresses used by the storage client 504, and the physical block addresses at which the storage controller 506 stores the data. The logical-to-physical translation layer 512 maps logical block addresses to physical block addresses of data stored on solid-state storage media 110. This mapping allows data to be referenced in a logical address space using logical identifiers, such as a logical block address. A logical identifier does not indicate the physical location of data on the solid-state storage media 110, but is an abstract reference to the data.

The storage controller 506 manages the physical block addresses in the physical address space. In one example, contiguous logical block addresses may in fact be stored in non-contiguous physical block addresses as the logical-to-physical translation layer 512 determines the location on the solid-state storage media 110 to perform data operations.

Furthermore, in one embodiment, the logical address space is substantially larger than the physical address space. This "thinly provisioned" or "sparse address space" embodiment, allows the number of logical identifiers for data references to greatly exceed the number of possible physical addresses.

In one embodiment, the logical-to-physical translation layer 512 includes a map or index that maps logical block addresses to physical block addresses. The map may be in the form of a B tree, a content addressable memory ("CAM"), a binary tree, and/or a hash table, and the like. In certain embodiments, the logical-to-physical translation layer 512 is a tree with nodes that represent logical block addresses and comprise corresponding physical block addresses.

As stated above, in conventional block storage devices, a logical block address maps directly to a particular physical block. When a storage client 504 communicating with the conventional block storage device deletes data for a particular logical block address, the storage client 504 may note that the particular logical block address is deleted and can re-use the physical block associated with that deleted logical block address without the need to perform any other action.

Conversely, when a storage client 504, communicating with a storage controller 104 with a logical-to-physical translation layer 512 (a storage controller 104 that does not map a logical

block address directly to a particular physical block), deletes a logical block address, the corresponding physical block address remains allocated because the storage client 504 does not communicate the change in used blocks to the storage controller 506. The storage client 504 may not be configured to communicate changes in used blocks (also referred to herein as "data block usage information"). Because the storage client 504 uses the block I/O emulation 508 layer, the storage client 504 may erroneously believe that the storage controller 506 is a conventional storage controller that would not utilize the data block usage information. Or, in certain embodiments, other software layers between the storage client 504 and the storage controller 506 may fail to pass on data block usage information.

Consequently, the storage controller 104 preserves the relationship between the logical block address and a physical address and the data on the storage device 102 corresponding to the physical block. As the number of allocated blocks increases, the performance of the storage controller 104 may suffer depending on the configuration of the storage controller 104.

Specifically, in certain embodiments, the storage controller 506 is configured to store data sequentially, using an append-only writing process, and use a storage space recovery process that re-uses non-volatile storage media storing deallocated/unused logical blocks. Specifically, as described above, the storage controller 506 may sequentially write data on the solid-state storage media 110 in a log structured format and within one or more physical structures of the storage elements, the data is sequentially stored on the solid-state storage media 110.

As a result of storing data sequentially and using an append-only writing process, the storage controller 506 achieves a high write throughput and a high number of I/O operations per second ("IOPS"). The storage controller 506 includes a storage space recovery, or garbage collection process that re-uses data storage cells to provide sufficient storage capacity. The storage space recovery process reuses storage cells for logical blocks marked as deallocated, invalid, unused, or otherwise designated as available for storage space recovery in the logical-physical translation layer 512.

As described above, the storage space recovery process determines that a particular section of storage may be recovered. Once a section of storage has been marked for recovery, the storage controller 506 may relocate valid blocks in the section. The storage space recovery process, when relocating valid blocks, copies the packets and writes them to another location so that the particular section of storage may be reused as available storage space, typically after an erase operation on the particular section. The storage controller 506 may then use the available storage space to continue sequentially writing data in an append-only fashion. Consequently, the

storage controller 104 expends resources and overhead in preserving data in valid blocks. Therefore, physical blocks corresponding to deleted logical blocks may be unnecessarily preserved by the storage controller 104, which expends unnecessary resources in relocating the physical blocks during storage space recovery.

Some storage devices 102 are configured to receive messages or commands notifying the storage device 102 of these unused logical blocks so that the storage device 102 may deallocate the corresponding physical blocks. As used herein, to deallocate a physical block includes marking the physical block as invalid, unused, or otherwise designating the physical block as available for storage space recovery, its contents on storage media no longer needing to be preserved by the storage controller 506. Data block usage information, in reference to the storage controller 506, may also refer to information maintained by the storage controller 506 regarding which physical blocks are allocated and/or deallocated/unallocated and changes in the allocation of physical blocks and/or logical-to-physical block mapping information. Data block usage information, in reference to the storage controller 506, may also refer to information maintained by the storage controller 506 regarding which blocks are in use and which blocks are not in use by a storage client. Use of a block may include storing of data in the block on behalf of the client, reserving the block for use by a client, and the like.

While physical blocks may be deallocated, in certain embodiments, the storage controller 506 may not immediately erase the data on the storage media. An erase operation may be performed later in time. In certain embodiments, the data in a deallocated physical block may be marked as unavailable by the storage controller 506 such that subsequent requests for data in the physical block return a null result or an empty set of data.

One example of a command or message for such deallocation is the "Trim" function of the "Data Set Management" command under the T13 technical committee command set specification maintained by INCITS. A storage device, upon receiving a Trim command, may deallocate physical blocks for logical blocks whose data is no longer needed by the storage client 504. A storage controller 506 that deallocates physical blocks may achieve better performance and increased storage space, especially storage controllers 506 that write data using certain processes and/or use a similar data storage recovery process as that described above.

Consequently, the performance of the storage controller 506 is enhanced as physical blocks are deallocated when they are no longer needed such as through the Trim command or other similar deallocation commands issued to the storage controller 506.

DIRECT INTERFACE BETWEEN A MEMORY CONTROLLER AND NON-VOLATILE MEMORY

Figure 3 illustrates one embodiment of a computing device 600 in accordance with the present invention. The computing device 600 may be one embodiment of the computer 112 depicted in Figure 1. The computing device 600 includes a processor 602, a memory module 608, an IO module 610, a basic input/output system ("BIOS") module 612, a network module 614, a peripheral component interconnect express ("PCIe") module 616, and a storage module 618. One of skill in the art will recognize that other configurations of a computing device 600 may be employed with the embodiments described herein.

The processor 602 executes computer readable programs stored on the memory module 604 as is well known to those skilled in the art. The processor 602 may include a cache 603 to reduce the average time to access the memory module 608. In one embodiment, the processor 602 comprises a multiprocessor having one or more cores (independent processing units). The cache 603 may store copies of instructions and data from frequently used locations in the memory module 608. The processor 602 may include a memory management unit ("MMU") 604 that translates logical memory addresses from a client (such as an operating system) to physical memory addresses that, in conventional computing devices, may correspond to physical locations on the storage media of the memory module 608. For example, in conventional computing devices, an operating system may send a data read request to the processor 602 along with logical addresses for a page stored in the memory module 608. The MMU 604 may translate the logical addresses to physical media addresses corresponding to locations of page data on the memory module 608.

In addition, the processor 602 includes a memory controller 605 that manages data communication between the processor 602 and the memory module 608. In conventional computing devices, the memory controller 605, when reading or writing data to the memory module 608, may send a physical memory address (from the MMU 604) to the memory module 608 to read a "word" of data. The size of the word is platform dependent: for example, a 64-bit computing device may have a word size of 8 bytes. Furthermore, the size of the memory address sent by the memory module 608 is also platform dependent. A memory controller 605 in a 64-bit computing device may request a word of data with a 64-bit address. Furthermore, the memory controller 608 may receive the requested word of data from the memory module 608 in a cache line of data, which may comprise the smallest unit of memory transferred between main memory of the memory module 608 and the cache 603. For example, the cache line may be 64 bytes of data (e.g. include the word and include a portion of contiguous surrounding data).

The processor 602 may communicate with the memory module 608 over a wire interface. In conventional computing devices, the wire interface may support a low-level wire protocol as

described below. In embodiments of the present invention, the wire interface may comprise a QuickPath Interconnect ("QPI") point-to-point processor interface by Intel® with a plurality of point-to-point data links. The wire interface, in another embodiment, is a HyperTransport point-to-point processor interface.

The processor 602 may communicate with the IO module 610. The IO module 610 may support and communicate with the BIOS module 612, the network module 614, the PCIe module 616, the storage module 618, and other components as is known in the art. The BIOS module 612 may communicate instructions through the IO module 610 to boot the computing device 600. Alternatively, the BIOS module 612 may comprise a coded program embedded on a chipset that recognizes and controls various devices that make up the computing device 600. The network module 614 may communicate with the IO module 610 to allow the computing device 600 to communicate with other devices over a network.

Computer readable programs may be stored in non-volatile storage on the storage module 618. The storage module 618 may include a hard disk drive, an optical storage device, a holographic storage device, a micromechanical storage device, a solid-state storage device 102 described above in relation to Figures 1-2, and the like. A solid-state storage device 102 such as that described above, may also communicate with the IO module 610 through the PCIe module 616 using a PCIe bus.

In conventional computing devices, the memory module 608, or "main memory," includes volatile memory such as dynamic random access memory ("DRAM") and static random access memory ("SRAM"). Specifically, the memory module 608 may include one or more storage media, such as one or more dual in-line memory modules ("DIMM"s) of volatile memory. Each DIMM may comprise a series of volatile memory integrated circuits. As stated above, the processor 602 communicates with the memory module 608 (with the volatile memory) over a wire interface by way of a protocol 606. Furthermore, in conventional computing devices, the processor 602 may communicate with the volatile memory by way of a low-level wire protocol 606 such as the Joint Electron Devices Engineering Council ("JEDEC") protocol.

JEDEC has been the industry standard for processor - DRAM interfaces. The JEDEC standard assumes that physically addressable media is synchronous, heavily parallel, reliable and implements a design structure that is known to the memory controller 605. Consequently, JEDEC uses a series of distinct commands, PRECHARGE, RAS, CAS, that cause the DRAM devices to execute known operations in hardware.

Even with some of the new, disruptive non-volatile memory technologies like Flash

Memory, Phase Change Memory ("PCM") and Spin-Torque Transfer Memory ("STT-RAM") in play for replacing DRAM, or other volatile memory, as main memory, the JEDEC standard is still being considered the standard of choice for accessing main memory that includes devices of these memory types. This means that although the unique properties of the device may lend itself to a different physical or board level arrangement (due to access times, byte/block addressability, read/write/erase capabilities, and wear out properties) the arrangement of these macroscale devices must still comply with the sub-optimal JEDEC standard. The assumption of the JEDEC standard has tied most studies for main memory replacements to technologies that have performance and reliability characteristics close to traditional DRAM.

Recently, there has been significant development of flash-based devices—a technology generally considered too unreliable for a main memory replacement—that are capable of operating on the PCIe bus, such as the solid-state storage device 102 described above. While these devices may export a traditional block device interface such as the block I/O emulation layer 508 described above in relation to Figure 2, advances in the logical-to-physical translation layer 512 no longer require these devices to be connected to the low speed, long latency DMI and SATA busses which currently limit bandwidth to a maximum of 2GB/s. In enterprise systems, the PCIe bus is moving further and further away from the processor 602, in terms of wire length, and communication between the processor 602 and the PCIe bus involves multi-hop protocols utilizing external DMA transfer engines, PCIe lane sharing which multiplexes multiple devices onto a single lane, and virtualization that requires additional hardware protection modules. While bandwidth rates are maintained, current NAND flash devices have an access time of approximately 50µ, but even the highest performing devices have access latency of approximately 250µ due to the multi-hop protocols required to reach the device.

This 4:1 overhead to access current NAND and other non-volatile memory technology is due primarily as a result of using legacy software interfaces within the operating system that require heavyweight operations such as context switching away from the user application into the kernel, traversing a heavyweight DMA setup process and then waiting for the asynchronous DMA operations to occur, a multi-hop operation requiring a minimum of 6 on/off chip accesses.

Conventional DRAM devices have an access time of approximately 10ns. Yet even with only a single socket addressing the DRAM subsystem, the total round trip time for a processor 602 to access main memory is on the order of 100-200ns. Control logic overhead, queuing delays, and off-chip wire delay are more expensive operations than the actual device latency. In multi-socket Non-Uniform Memory Access ("NUMA") memory models, this disparity becomes even greater. NUMA memory models involve multiprocessors in which each processor has a

local main memory but can also access the main memory local to other processors (e.g. a memory controller of one processor is connected to a memory controller of another processor via Hypertransport or QPI links). As a result, main memory access time from a particular processor depends on the distance between the particular processor and the main memory it accesses, with faster access times being those for processors physically closer to main memory.

NUMA memory model computer systems with large amounts of main memory are currently available such as 1024 cache-coherent non-uniform memory access ("CC-NUMA") machines where lterabyte ("TB") of DRAM is standard and accessible via the traditional memory controller interface. In these machines, a single memory access can traverse as many as 64 Hypertransport or QPI links before arriving at the on chip memory controller which is physically attached to the DRAM device to be accessed. As a result, the wire delay to access main memory can grow into the 10's of μ scale, while actual DRAM device access time remains only 10ns.

Certain processors have recently incorporated multiple memory controllers ("MC"s), with each memory controller controlling a distinct subset of the total main memory physical address space. As described below, a computing device may use multiple "on-chip" memory controllers to deviate from the JEDEC standard and access high bit density technologies, such as the non-volatile memory types described above, that reduce the need for complex long wire traversals in exchange for local, albeit slower, amounts of local memory. Because the main memory interface is synchronous, latency, not bandwidth is often the limiting factor to application throughput.

Figure 4 illustrates one embodiment of a system 700 for a direct interface between a memory controller 605 and non-volatile memory in accordance with the present invention. Figure 4 refers to elements of Figure 3, like numbers referring to like elements. Figure 4 includes a processor 602 in communication with a non-volatile memory module 712. The processor includes a cache 603 with a cache line 702, an MMU 604, and a memory controller 605 with a logic engine 704. The memory controller 605 communicates, by way of a data path 706 and a control path 708 of a wire interface 707, with a non-volatile memory controller 712 in the non-volatile memory module 710. The non-volatile memory controller 712 includes a logic engine 714 and a logical-to-physical translation layer 716. In addition, the non-volatile memory module 710 includes non-volatile memory media 718a-c and a volatile memory buffer 720.

As described above, the processor 602 may be a multiprocessor having one or more cores. The processor 602 includes a cache 603 as described above. The cache 603, in the depicted embodiment, includes a cache line 702. In certain embodiments, the processor 602

includes multiple caches 603 and may include multi-level caches with such as a Level 0 ("LO") cache, Level 1 ("LI") cache, Level 2 ("L2") cache, Level 3 ("L3") cache, and the like. As is known in the art, with a multi-level cache, the processor 602 may sequentially check each cache level for cached data starting with the smallest and fastest cache (such as LI cache). As described above, the processor 602 may include an MMU 604 that translates logical memory addresses from a client to physical memory addresses that, in conventional computing devices, correspond to physical locations on one or more of the storage media of main memory. The physical memory addresses from the MMU 604 may be communicated to the non-volatile memory module 710 by the memory controller 605, which is described below.

The non-volatile memory module 710 may support, house, and/or provide access to one or more non-volatile memory media devices 718a-c. In one embodiment, the non-volatile memory module 710 includes all or a portion of the functionality described above in relation to the solid-state storage device 102 for reading data, writing data, storage space recovery, and the like, except that the non-volatile memory module 710 reads and writes data in a main memory context, not a storage context. For example, the non-volatile memory module 710 through the non-volatile memory controller 712 may be configured to operate memory maintenance functions on the non-volatile memory media 718 to optimize non-volatile memory module 710 performance. These memory maintenance functions may include, but are not limited to storage space recovery, error correcting code ("ECC"), log-based sequential storage, and wear-leveling as described above in relation to the solid-state storage device 102 and solid state storage controller 104 and as described below specific to using the non-volatile memory module 710 for memory. The non-volatile memory module 710 includes a logical-to-physical translation layer 716 that is similar in concept to the logical-to-physical translation layer 512 described above in relation to Figure 2, except that the logical-to-physical translation layer 716 depicted in Figure 4 treats physical memory addresses from the MMU 604 (e.g. what the MMU 604 assumes are physical addresses pointing to locations on the media) as logical memory addresses, translating these into physical media addresses specifying locations on the non-volatile memory media. The logical-to-physical translation layer 716 is described in further detail below.

The non-volatile memory module 710 includes non-volatile memory media 718. The non-volatile memory media may include flash memory, nano random access memory ("NRAM"), magneto-resistive RAM ("MRAM"), dynamic RAM ("DRAM"), phase change RAM ("PRAM"), Racetrack memory, Memristor memory, nanocrystal wire-based memory, silicon-oxide based sub-10 nanometer process memory, graphene memory, Silicon-Oxide-Nitride-Oxide-Silicon ("SONOS"), Resistive random-access memory ("RRAM"), programmable

metallization cell ("PMC"), and conductive-bridging RAM ("CBRAM").

Non-volatile memory, such as flash, has substantially better bit density than DRAM and average power characteristics. For example, 1TB of DRAM based main memory organized as 512x2GB DIMMS will consume almost 1KW of power (4.7W/DIMM). As a result, CC-NUMA machines must distribute these modules across a large rackspace to satisfy power and cooling constraints. Modern NAND flash devices can deliver 1.28TB of storage in as little as 25 watts.

In addition, conventional computer systems are architected such that all storage must be first loaded into main memory (typically via DMA) before the processor 602 can access that data. This is done in chunks as large as Megabytes, but never any smaller than the native block size of the device, typically 512B or larger. This initial copy (called a Von Neumann copy) to a volatile memory structure is inherently an energy inefficient method of accessing data, if not all 512B are going to be used. Providing direct access to large volumes of persistent storage removes the need for the implicit copying that occurs in Von Neumann architectures.

Current non-volatile memory technologies have inherently lower bandwidth than DRAM devices. Current state of the art Flash (such as NAND) based designs can saturate a lGB/s link with 24 pipelined transactions. To achieve bandwidth on parity with modern DRAM systems, in one embodiment, multiple non-volatile memory devices may be used in a channel based arrangement similar to DRAM devices today. Advantageously, these lower bandwidth links also require less on chip pins, so the absolute number of off-chip pins may drop or at least remain unchanged.

In addition, non-volatile memory technologies are inherently asymmetric. Read and Write (a combination of erase+write in many cases) have very different latency and power properties. While write buffering and other techniques reduce this impact, too many writes to the non-volatile memory subsystem, such as media 718, could lead to performance degradation. However, as described below, non-volatile memory modules 710 may co-exist alongside traditional volatile memory modules where a client, such as the operating system, may chose to map read only pages onto the non-volatile memory physical address space. Over 95% of an application's footprint in memory may be read only even in 8 and 16 core multi processor systems.

As described above, after a limited number of writes to a non-volatile memory physical memory cell, the cell can no longer be erased and re-written. For modern NAND flash MLC cells, this number is currently about 3000. To avoid wearing out the physical media, memory device manufactures may implement a logical to physical mapping layer as described above, and use over-provisioning and other techniques. This translation can happen at near line-speeds and

is effective enough that even by constantly writing to a modern flash device at a full lGB/s, many flash device manufacturers guarantee that a flash device will not wear out in any less than 5 years.

Furthermore, future technologies are expected to have improved wear-out capabilities, increasing future data integrity with non-volatile memory behind a near line-speed flash translation layer. Operating system intelligence in placing write-heavy pages on volatile memory modules for performance reasons will have the added benefit of improving the non-volatile memory module's 710 wear-out prevention capabilities.

In addition, the non-volatile memory module 710 may include spare non-volatile memory devices. As described below, the non-volatile memory module 710 may signal the memory controller 605, by communicating memory attributes as described below, that the non-volatile memory media 718 currently in use has reached a predetermined amount of wear and/or is exhibiting certain performance characteristics. The memory controller 605 may signal the operating system, which may alert a user and/or refrain from performing subsequent memory operations (e. g. wear intensive operations such as writing or erasing) on the non-volatile memory module 710. In the meantime, the non-volatile memory module 710 may copy the data from the affected non-volatile memory media 718 to the spare memory devices. In some embodiments, the user may switch out the worn memory devices for spare memory devices. Additionally, memory devices of the non-volatile memory module 710 may be swappable, interchangeable, and/or replaceable. The non-volatile memory module 710 may have one or more open memory device slots. A user may plug-in an additional memory device into an open slot. The non-volatile memory module 710 may copy data from a selected memory device (such as a worn memory device) to the additional memory device. The user may then remove the selected memory device. In one embodiment, memory devices of the non-volatile memory media 718 in the non-volatile memory module 710 are "hot swappable," meaning that the non-volatile memory module 710 and/or the processor 602 does not need to be shut down before adding or removing memory devices.

The non-volatile memory module 710 may be in communication with the processor 602 over the wire interface 707. In certain embodiments, instead of only connecting processor memory controllers together using QPI point-to-point processor interfaces or HyperTransport point-to-point processor interfaces, the wire interface 707 between the memory controller 605 and the non-volatile memory module 710 may include a QPI point-to-point processor interface, a HyperTransport point-to-point processor interface, or other similar wire interface. Consequently, in one embodiment, the wire interface 707, in addition to supporting a low level, wire interface

specific communication as conventional wire interfaces, the wire interface 707 includes a data path 706 and a control path 708 to communicate data and commands between the memory controller 605 of the processor 602 and the non-volatile memory controller 712.

The memory controller 605 communicates with, is coupled to, and/or integrated with the processor 602. The memory controller 605 manages data communication between the processor 602 and the non-volatile memory module 710. For example, the memory controller 605 may send a physical memory address to the non-volatile memory module 710 as part of a read request. As described above, in conventional computing devices, the memory controller 605 may communicate with a volatile memory module using the JEDEC protocol. However, in the depicted embodiment, the memory controller 605 includes a logic engine 704 that supports communication over the wire interface 707 to non-volatile memory. In one embodiment, the logic engine 704 includes logic to enable the memory controller 605 to communicate over the wire interface 707 by way of a command protocol. The command protocol enables greater communication flexibility in comparison to a wire protocol, providing for a wider variety of information that the wire interface 707 may communicate and providing for more methods of communication. Specifically, the memory controller 605 may take advantage of wire interface technology (e.g. QPI or Hypertransport) typically used to connect a processor memory controller to another processor memory controller, to, instead of communicating bit addresses and signals as part of the JEDEC standard, prepare data operation and control commands and then communicate (send and receive) data operation commands and control commands, communicate variables, and/or communicate memory management metadata by way of the command protocol as described below.

While the JEDEC standard includes commands, PRECHARGE, Row Address Strobe (RAS), Column Address Strobe (CAS), JEDEC combines commands with a physical wire protocol into one specification such that limitations of the wire interface are inherently embedded into the JEDEC standard. For example, the coupling of commands with the wire interface under JEDEC protocols requires that a memory controller and the memory device that the memory controller communicates with adhere to specific physical connection parameters. For example, the memory device (such as a DRAM DIMM) must operate at a given value or range of values for voltage, frequency, and impedance defined by the JEDEC protocol for the wire interface.

Additionally, because of this inter-relationship between the wire interface and the JEDEC commands, memory devices that satisfy the JEDEC memory communication protocol suffer from the STUB electronics problem. The STUB electronics problem limits the maximum

number of memory devices that can be connected over the wire interface (e.g. the JEDEC standard bus) to avoid noise interference due to each added memory devices operating like an antenna.

In addition, because the operation of the connected memory device has become so standard and well understood during the development of the JEDEC standard memory communication protocol, even certain JEDEC commands have been influenced by the electronic characteristics of the memory devices. For example, the PRECHARGE command is a command that initiates a voltage charge in the memory device and has a set minimum latency threshold that dictates when a subsequent command can be sent.

In contrast, the memory controller 605 and non-volatile memory controller 712 keep the command protocol and the physical wire protocol separate, which provides improved flexibility. The physical limitations of the wire interface do not affect the command protocol. Specifically, the same command protocol may support multiple wire interfaces 707 or alternative wire interfaces 707 and because the memory controller 605 and non-volatile memory controller 712 may report-back/query the performance and/or characteristics of the wire interface 707, no assumptions about the limitations of the wire interface are inherently embedded in the command protocol. The logic engine 704 may interpret and/or generate data bit patterns according to the command protocol. The logic engine 704 comprises the logic needed to decouple the command protocol and the wire interface 707. For example, the logic engine 704 may be configured to handle re-transmission of commands, out-of-order command processing, or other higher level communication tasks that enable the data path 706, control path 708 and wire interface 707 to be de-coupled.

The non-volatile memory controller 712 may manage, facilitate, and/or perform memory operations on the non-volatile memory media 718a-c in the non-volatile memory module 710. The non-volatile memory controller 712 may be coupled to, in communication with, and/or integrated with the non-volatile memory media 718a-c. The non-volatile memory controller 712 may implement all or a portion of the solid-state storage device 102 functionality as described above, including operating memory maintenance functions such as storage space recovery, ECC, and wear-leveling described in more detail below. In certain embodiments, the non-volatile memory controller 712 may be configured to detect and replace certain storage media that fails to meet a performance threshold. Specifically, the non-volatile memory controller 712 may copy data from the certain storage media to one or more additional storage media such as a parity storage element (e. g. a device or chip), a reserved storage element, a backup or redundant storage element, and the like. The non-volatile memory controller 712 may then map storage

operations to the replacement storage media instead of the certain storage media.

In addition, the nonvolatile memory controller 712 may include similar data storing functionality as the solid-state storage controller 104. Specifically, in one embodiment, the non-volatile memory controller 712 is configured to store data on the non-volatile memory media 718 using a sequential log-based storage format described below. The non-volatile memory controller 712 may also implement, maintain, and/or manage the logical-to-physical translation layer 716 described below. The functionality of the non-volatile memory controller 712 may be implemented in firmware, hardware logic, and/or software in the non-volatile memory module 710.

The non-volatile memory controller 712 may receive commands from the memory controller 605. As described above, in conventional computing devices, the memory controller 605 communicates with main memory using the JEDEC protocol. However, in the depicted embodiment, the non-volatile memory controller 712, like the memory controller 605, includes logic 714 (that may be embodied in certain embodiments by the modules described below) that supports communication over the wire interface 707 to the memory controller 605 by way of the command protocol.

In one embodiment, the data path 706 communicates data between the memory controller 605 and the non-volatile memory controller 712. Specifically, the data path 706 may communicate a 64-byte cache line of data containing a word of data requested by the memory controller 605. By separating the control path 708 and the data path 706 the bandwidth for the data path 706 over the wire interface 707 is maximized. In certain embodiments, the separation of these paths 706, 708 permits the data to flow on the data path at line-speed (the same speed as the clock cycles that operate the wire interface 707). In one embodiment, the data path 706 and control path 708 may co-exist on the wire interface 707 using a multiplexor to distinguish between information related to the data path 706 and information related to the control path 708.

In one embodiment, the control path 708 communicates data operation commands between the memory controller 605 and the non-volatile memory controller 712. Data operation commands may include, but are not limited to data read requests, data write requests, and the like. In addition, a data operation command, in one embodiment, contains a physical memory address from the MMU (treated as a logical memory address by the non-volatile memory controller 712). For example, the memory controller 605 may send a 64-bit memory address over the control path 708 in a data operation command to request a word of data from the non-volatile memory module 710. Similarly, the control path 708 may also communicate a response from the non-volatile memory controller 712. Specifically, the response may include a

confirmation of a successful write.

For example, the control path 708 may communicate a data operation command from the memory controller 605 requesting a word of data and containing a 64-bit physical memory address. The data path 706 may subsequently communicate data with the requested word embedded in a 64-byte cache line. In one embodiment, the memory controller 712 requests a word of data at a time. However, the command protocol, in some embodiments, allows the memory controller 712 to request and receive amounts of data not limited to a word. In one embodiment, the memory controller 712 may request a plurality of words of data, or another unit of data.

The control path 708 may also communicate control commands between the memory controller 605 and the non-volatile memory controller 712. Control commands may include memory management commands such as deallocation commands, described below, which notify the non-volatile memory controller 712 that certain data does not need to be preserved (similar to the Trim command). Memory management commands may also include requests for memory attributes from the non-volatile memory module 710. Memory attributes provided by the non-volatile memory module 710 may include attributes, characteristics, and/or a status of the non-volatile memory media 718a-c. Memory attributes may also include memory performance attributes and memory wear-out attributes. For example, the memory attributes may specify that a particular non-volatile memory module 710, a particular non-volatile memory media 718, or a particular section (e.g. an LBA) on non-volatile memory media 718 is experiencing excessive wear. In addition, certain non-volatile memory media devices 718a-c may have differing characteristics from other non-volatile memory media devices 718a-c. The memory controller 605 may distinguish among different non-volatile memory media devices 718a-c based on memory attribute information conveying characteristics of each non-volatile memory media devices 718a-c.

As described above, in one embodiment, the non-volatile memory module 710 includes spare (such as for example replacement) non-volatile memory devices. Spare non-volatile memory devices may already be installed in the non-volatile memory module 710 and may not be used until needed. Alternatively, spare non-volatile memory devices may be installed in the non-volatile memory module 710 either in open slots or to replace existing non-volatile memory devices when needed. Specifically, in certain embodiments, the non-volatile memory controller 712 may be configured to detect and replace certain storage media that fails to meet a performance threshold. The non-volatile memory controller 712 may communicate to the memory controller 605, through the control path 708, that a memory device has reached a

predetermined amount of wear and/or is exhibiting certain performance characteristics. The memory controller 605 may signal the operating system, which may alert a user and/or refrain from performing subsequent memory operations on the non-volatile memory module 710. In the meantime, the non-volatile memory module 710 may copy the data from the affected memory devices to the spare/replacement memory devices. The non-volatile memory controller 712 may then map future storage operations to the spare/replacement storage media instead of the worn memory devices. In some embodiments, the user may switch out the worn memory devices for the replacement memory devices. Specifically, as described above, memory devices 718a-c of the non-volatile memory module 710 may be swappable, interchangeable, and/or replaceable. The non-volatile memory module 710 may have one or more open memory device slots. A user may plug-in an additional memory device 718c into an open slot. The non-volatile memory controller 712 may copy data from a selected memory device 718a (such as a worn memory device) to the additional memory device 718c.

In one embodiment, the non-volatile memory controller 710 sends memory attributes in response to a query command requesting attributes from the memory controller 605. For example, a client such as an operating system may request a wear-out status of a non-volatile memory module 710. The operating system communicates the request to the memory controller 605, which then communicates the request over the control path 708. The non-volatile memory controller 710 may respond with a status of the non-volatile memory media 718a-c in the form of memory attributes communicated over the control path 708. In one embodiment, the non-volatile memory controller 712 communicates memory management metadata at a predetermined interval. The depicted embodiment depicts a distinct data path 706 and control path 708 (the data and the data and control commands may transmit over different wires or groups of wires). In one embodiment, the data and the commands communicate over distinct paths for performance reasons. For example, control commands may have a lower priority than data. In other embodiments, the wire interface 707 may communicate the data and the commands over a common path (a common wire or groups of wires) and the data and commands may be distinguishable by way of a multiplexor.

The total time to access a traditional non-volatile memory device is dominated by both the software overheads of using legacy block device interfaces, and the wire delay to traverse multiple bridges and the long wires required that allow physical access to the off-chip devices. Removing the trap to a software routine and allowing these devices to be accessed as part of the 64-bit physical address space, rather than within the storage address space, will remove as much as 50% of the total device access latency. Moving access to these devices from the long and

complex PCIe bus, to a processor direct connected interface can reduce the physical propagation delay by as much as 30%. Overall, moving to a processor direct interface has the potential of reducing total access latency to non-volatile memory technology by over 75%.

The non-volatile memory module 710 includes a volatile memory buffer 720 that, in one embodiment, stores a copy of data structures used to implement the logical to physical translation layer 716 described below.

Figure 5 illustrates a logical representation of one embodiment of a plurality of communication layers 800 between a client 802 and non-volatile memory media 604 in accordance with the present invention. Figure 5 refers to elements of Figure 3 and Figure 4, like numbers referring to like elements. Figure 5 depicts a client 802, the memory controller 706, a wire interface 804, a command protocol 806 over the wire interface 804, the non-volatile memory controller 712, the logical-to-physical translation layer 716, and the non-volatile memory media 718. The client 802 initiates data reads and writes to the processor 602 for the non-volatile memory media 718. Representative examples of clients 802 include, but are not limited to, one of or a combination of, a server, a file system, an operating system, an MMU 604, a processor 602 (multiprocessor), and the like. The wire interface 804 may be similar to the wire interface 707 described in relation to Figure 4, comprising a QPI or HyperTransport point-to-point interface. The wire interface 707 may also support a data path 706 and a control path 708 as described in Figure 4. The command protocol 806 may also be similar to the command protocol described above. The logical-to-physical translation layer 716 may map logical memory addresses (typically assumed by the memory controller 605 to be physical memory addresses) contained in the commands from the memory controller 706 to physical media addresses indicating physical storage memory locations in the non-volatile memory media 718.

As described above, in conventional computing devices, the address used at the memory controller 706 and used to access DRAM is called the physical address. Therefore, the physical address sent by the memory controller 706 to the main memory references a physical location on the DRAM. Due to wear-out issues with non-volatile memory, in various embodiments, a client 802 such as the operating system, the memory controller 706, and/or the non-volatile memory module 710 may control wear-out properties. In one embodiment, wear leveling and other advanced techniques to improve the raw performance of these non-volatile memory is controlled at the device level in the non-volatile memory module 710. The non-volatile memory module 710 may present a linear contiguous address space to the client, underneath which a second layer of translation may occur before accessing the non-volatile memory media 718.

The logical-to-physical translation layer 716 may be similar in concept to the logical-to-

physical translation layer 512 described above for the storage controller 506 in relation to Figure 2. Specifically, the logical-to-physical translation layer 716 provides a level of abstraction between the physical memory addresses used by the memory controller 706 (obtained from the MMU 604), and physical media addresses, or addresses that specify physical locations at which the non-volatile memory controller 712 stores the data. Consequently, a logical address sent by the client 802 may undergo two address translations: being translated at the MMU 604 in the processor 602 to a physical memory address (used to locate data on the physical media in conventional computing devices), and then being translated again at the logical-to-physical translation layer 716 from the physical memory address to a physical media address indicating the location of the data on the non-volatile memory media 718.

This mapping allows data to be referenced in a logical address space on the non-volatile memory media 718, treating the physical memory addresses as logical identifiers. A logical identifier does not indicate the physical location of data on the non-volatile memory media 604, but is an abstract reference to the data.

The non-volatile memory controller 712 manages the actual physical address space and may divide the physical address space into smaller units such as logical pages ("LP"s), logical erase blocks ("LEB"s), ECC chunks, packets, blocks, sectors, and the like, similar to the solid-state storage device 110 as described above. In one embodiment, the non-volatile memory controller 712 divides the physical address space into units of the same size as the cache line 702. For example, the non-volatile memory controller 712 may maintain LEBs of cache-line sized memory units such that a memory unit, at its smallest form, may does not span LEBs or other similar logical structures.

Furthermore, in one embodiment, like the logical address space presented by the solid-state storage device 102, the logical address space presented by the non-volatile memory controller 712, is substantially larger than the actual physical address space. This "thinly provisioned" or "sparse address space" embodiment, allows the number of logical identifiers for data references to greatly exceed the number of possible physical addresses. Specifically, the non-volatile memory controller 712 may be configured to store data sequentially, using a log-based append-only writing process, similar to the storage controller 506 described above in relation to Figure 2. Specifically, the non-volatile memory controller 712 may sequentially write data on the non-volatile memory media 718 in a log structured format and use a storage space recovery process that re-uses data storage cells as described below to provide sufficient storage capacity.

In one embodiment, the logical-to-physical translation layer 716 includes a map or index,

similar to index of the solid-state storage device 102, except that it may map physical memory addresses to physical media address. The map may be in the form of a B tree, a content addressable memory ("CAM"), a binary tree, and/or a hash table, and the like described below. In certain embodiments, the logical-to-physical translation layer 716 is a tree with nodes that represent physical memory addresses, associating corresponding physical media addresses. In one embodiment, each node represents a contiguous range of physical memory addresses to minimize tree size.

As stated above, in computing devices with conventional main memory, a physical memory address maps directly to a particular physical location in the main memory. A client 802 communicating with this conventional main memory typically deletes data in main memory by overwriting data for a particular physical memory address. In addition, due to the volatile nature of conventional main memory, when the main memory is powered off, the data is cleared.

Conversely, when a client 802, such as a processor 602, communicating with a non-volatile memory module 710 having a logical-to-physical translation layer 716, no longer needs data, the corresponding physical media addresses may remain allocated on the non-volatile memory media 718. For example, a processor 602 executing an application using one or more pages stored on the non-volatile memory media 604, may close the application. The data corresponding to the pages stored on the non-volatile memory media 604 are no longer needed. However, the non-volatile memory controller 712 may still preserve the relationship between physical memory addresses for the pages and the corresponding physical media addresses.

In certain embodiments, the non-volatile memory controller 712, similar to the storage controller 506 above, is configured to store data on the non-volatile memory using log-based storage and configured to recover storage space on the non-volatile memory media 718 using a storage space recovery process. Like the storage controller 506, the non-volatile memory controller 712 may include a similar storage space recovery, or garbage collection process that re-uses data storage cells to provide sufficient storage capacity. The storage space recovery process reuses storage cells for storage units marked as deallocated, invalid, unused, or otherwise designated as available for storage space recovery in the logical-to-physical translation layer 716. The storage space recovery process may also preserve storage units that the non-volatile memory controller 712 specifies as valid.

Similar to the storage controller 506, the non-volatile memory controller 712 expends resources and overhead in preserving data in valid memory units. Therefore, memory units corresponding to deleted pages may be unnecessarily preserved by the non-volatile memory controller 712. Furthermore, due to the inherent transient nature of data in main memory, the

performance of the non-volatile memory module 710 may suffer if it retains unneeded memory units.

As stated above, some storage devices 102 are configured to receive messages or commands, such as the Trim command, notifying the storage device 102 of unused logical blocks so that the storage device 102 may deallocate the corresponding physical blocks. In one embodiment, the non-volatile memory controller 712 is configured to receive a deallocation command similar in concept to the Trim command. Specifically, the memory controller 706 may communicate a deallocation command over the control path 708 by way of the command protocol 806, notifying the non-volatile memory controller 712 of physical memory addresses that the client no longer needs. In one embodiment, the client 802 issues the deallocation command, notifying the memory controller 706 of the memory addresses for one or more pages of data that may be deallocated. For example, the client 802 may evict one or more pages of data from cache. The MMU 604 translates these page memory addresses into the corresponding physical memory addresses and communicates the deallocation command over the control path 708 with the corresponding physical memory addresses. The non-volatile memory controller 712 may then deallocate the corresponding memory units for those addresses by removing nodes in the logical-to-physical translation layer map, tree or index. Specifically, in one embodiment, the logical-to-physical translation layer 716 comprises a tree with nodes representing contiguous physical memory addresses with each node providing an indicator of the location of corresponding data on the non-volatile memory media 718. In this embodiment, the presence of a node including a particular physical memory address indicates that the physical memory address has data stored on the non-volatile memory media 718. Similarly, if a physical memory address has no corresponding node, the physical memory address is not assigned. Therefore, by removing and/or modifying a node in the tree to eliminate a physical memory address, the data corresponding to the physical memory address is freed.

Figure 6 is a schematic block diagram illustrating one embodiment of a system 900 with a plurality of memory controllers 904, 908 communicating with a plurality of memory modules 906, 710 in accordance with the present invention. Specifically, Figure 6 depicts a system 900, as mentioned above, in which volatile memory modules 906 co-exist with non-volatile memory modules 710. In the depicted embodiment, a first plurality of memory controllers 904 communicate with volatile memory modules 906 over a plurality of wire interfaces 905. Each wire interface 905 may be similar to the wire interface 606 described in relation to Figure 3. In one embodiment, the first plurality of memory controllers 904 communicate by way of a protocol optimized for volatile memory module management such as the JEDEC protocol or a

Rambus (RDRAM or Direct RD-RAM) protocol. The volatile memory module 906 may house and provide access to volatile memory media. The volatile memory media may include DRAM, SRAM, and the like.

In the depicted embodiment, a second plurality of memory controllers communicate 908 with non-volatile memory modules 710 over a plurality of wire interfaces 905. Each of the second plurality of memory controllers 908 may be similar to the memory controller 605 in Figure 4. Likewise, each of the non-volatile memory modules 710 may be similar to the non-volatile memory module 710 in Figure 4. In addition, the second plurality of memory controllers 908 may communicate by way of the command protocol 806 described above in relation to Figures 4 and 5. In one embodiment, the amount of second memory controllers 908 is less than the amount of first memory controllers 904, and similarly the amount of non-volatile memory modules 910 is less than the amount of volatile memory modules 906.

In the depicted embodiment, the processors 902 are in communication with one another over an inter-processor interface 907 between processors 902. The inter-processor interface 907 may also allow communication between memory controllers 904. Consequently, memory accessible to a particular memory controller 904a is also accessible to other processors 902b, 902c. In one embodiment, the inter-processor interface comprises a QuickPath Interconnect ("QPI") or HyperTransport point-to-point processor interface.

In one embodiment, the first memory controllers 904 and the second memory controllers 908 present a single logical address space to a client 802. In some embodiments, the client 802, such as an operating system, is aware of the boundaries in the address space that divide volatile memory modules 906 and non-volatile memory modules 710. In another embodiment, the first memory controllers 904 may present a first logical address space and the second memory controllers 908 may present a second logical address space. The client 802 may differentiate the type of main memory using the separate address spaces. Furthermore, the client 802 may direct data to memory divisions in the volatile memory and/or the non-volatile memory or to volatile or non-volatile memory modules based on various memory attributes. The non-volatile memory media 718 may be divided into memory divisions, which are physical or logical units of memory including pages, erase blocks, media banks, channels or the like. For example, in one embodiment, memory attributes may include a writing frequency and/or a reading frequency of data pages directed toward main memory, power use characteristics, persistence characteristics, and/or performance characteristics. These memory characteristics may be communicated by way of the command protocol 806 on the control path 708 (See Fig. 7).

Currently, the support of operating systems and memory controllers for NUMA systems

is rudimentary. Support exists simply by attempting to map memory pages being requested via a processor onto the physical NUMA node that exists "nearest" to that processor socket. In the depicted embodiment, in addition to the memory controller utilization being a first class scheduling primitive, an operating system may be aware of the exported access times and wear out properties from the non-volatile memory modules 910. For example, most non-volatile memory technologies have write times that have 2-10x longer latency than their respective read access times. For optimal page placement to occur, this imbalance may be recognized by a client such as the operating system, so that write heavy pages are migrated away from write-unfriendly non-volatile memory modules 910 onto write-friendly volatile memory modules 906. Similarly some pages may be very sensitive to read latency and page-migration between non-volatile memory modules 910 may optimize application throughput. By far, the majority of operating system pages allocated are read only. In one embodiment, the high density (read-optimized) non-volatile memory NUMA capacity is optimized against the (write-friendly) volatile NUMA capacity. For example, in one embodiment, the ratio of volatile memory modules 906 to non-volatile memory modules 910 is 1:2. In another embodiment, the ratio is 1:3, 1:4, 1:5 or the like. Any suitable ratio may be used.

In addition, memory attributes may also include wear-out characteristics and volatility characteristics. As stated above, in one embodiment, the client, such as an operating system, does not explicitly manage the wear out properties of lossy devices that may change through time and vary substantially between process generations. As stated above, the non-volatile memory module 710 may include a logical-to-physical translation layer 718 that allows wear leveling, redundancy, ECC, and other features. This logical-to-physical translation layer 718 allows a reliable, non-lossy interface to be provided on top of an inherently lossy medium. However, in one embodiment, the non-volatile memory modules 710 do not manage this lossy nature in isolation. Specifically, in one embodiment, the non-volatile memory modules 710 may provide wear-out characteristics or other memory characteristics as part of a wear-out rate feedback mechanism to the client 802 so that it can make intelligent decisions about what write-level to drive into the memory modules 906, 910. For example, a non-volatile memory module 710 may export, by way of the control path 708, projected wear-out durations for predetermined time periods such as the 5 min, 60min, 24 hour, and 7 day average traffic patterns. If a wear-out duration of 3 years is deemed to be too low, the client 802 may dynamically shift its write pattern (through page migration) to a more write-friendly device, such as a volatile memory module 906. If the wear-out duration of 10 years is deemed high enough, the client 802 may chose to dynamically shift more pages to the non-volatile memory module 906 to improve overall system

energy efficiency.

Conventional non-volatile memory research focuses on the use of non-volatile memory module 906 as pin compatible solutions with JEDEC conforming devices. Conventional non-volatile memory research avoids non-volatile memory technologies such as Flash based technologies (NAND/NOR) because other memory researchers are unable to compensate for the wear-out properties of these devices.

The combination of non-volatile memory modules 710 and volatile memory modules 906 in a single system 900 as described above embraces the heterogeneous nature of new technology. Allowing new interfaces and devices to connect to the memory controller exports a richer set of interface properties and uses low power high density non-volatile memory when possible. The interplay between page migration, copy-on-write semantics, and application throughput may be leveraged to obtain the optimal power efficiency in systems while still maintaining a lower bound on application throughput.

Figure 7A illustrates one embodiment of a system 1000 with a memory controller 1004 communicating with a plurality of memory modules 710, 1008 in accordance with the present invention. Specifically, in one embodiment, a single memory controller 1004 may support two or more communication protocols. The depicted embodiment depicts a processor 1002 and a memory controller 1004 communicating with a non-volatile memory module 710a over a first interface 1005a and communicating with a volatile memory module 1008a over a second interface 1005b. In one embodiment, the memory controller 1004 communicates with the non-volatile memory with a first protocol and communicates with the volatile memory with a second protocol. In one embodiment, the first protocol and the second protocol are different protocols. For example, the first protocol may be the command protocol 806 described above and the second protocol may be the JEDEC protocol. In one embodiment, the memory controller 1004 includes multiple logic engines (to support multiple protocols) or a logic engine configured to support multiple protocols. The memory controller 1004 may exist in a system of multiple memory controllers and memory modules as described above in relation to Figure 6.

Figure 7B illustrates one embodiment of a system 1009 with a plurality of memory controllers 1012 in a processor communicating with a plurality of memory modules 710b, 1008b in accordance with the present invention. In the depicted embodiment, a first memory controller 1012 and a second memory controller 1014 are coupled to a common processor 1010. Specifically, a computing system supporting the depicted embodiment may allow multiple memory controllers 1012 per processor socket. The depicted embodiment shows the first memory controller 1012 communicating with a non-volatile memory module 710b over a first

interface 1015a and the second memory controller 1014 communicating with a volatile memory module 1008b over a second interface 1015b. In one embodiment, the first memory controller 1012 communicates with the non-volatile memory with a first protocol and the second memory controller communicates with the volatile memory with a second protocol. In one embodiment, the first protocol and the second protocol are different protocols. For example, the first protocol may be the command protocol described above and the second protocol may be the JEDEC protocol.

In one embodiment, the client 802 may extend the NUMA factor from multi-socket to multi-memory controller even within a socket. In one embodiment, a computing system with non-volatile memory modules and multi-memory controller sockets may provide a lower NUMA factor than a widely distributed DRAM based system because the bit density of the non-volatile memory system can provide hundreds of GB of main memory within a power and space budget that is over lOOx better. This results in decreased wire delays, a dominant portion of access time in modern NUMA systems. NUMA factors may no longer be tied to physical locality of memory controllers, so process scheduling due to the NUMA factor may be taken into account making memory controller utilization a first class scheduling primitive.

Figure 8 is a schematic block diagram illustrating one embodiment of the non-volatile memory controller 712. In the depicted embodiment, the non-volatile memory controller 712 includes a receiving module 1102 and an execution module 1104.

The receiving module 1102 receives commands from the memory controller 605 over the wire interface 707 by way of the command protocol as described above. The command protocol includes a control path 708 that enables the memory controller 605 to distinguish among different memory modules. Specifically, the memory controller 605 may obtain memory attributes from various memory modules 710 over the control path 708. The control path 708 also provides for more commands than a standard JEDEC protocol.

In one embodiment, commands received from the memory controller 605 may include read commands, write commands, or memory management commands. Memory management commands may be further classified as query commands, directive commands, hint commands, or checkpoint commands, which are described below.

In one embodiment, a hint command is a deallocation command or discard command identifying to the non-volatile memory controller 712 one or more memory units that have been deallocated, and thus no longer need to be preserved.

A hint command may also include an "F-advise" command. An application executing on the processor 602 may signal that it will be using a certain memory address range heavily. The

memory controller 605 may issue an F-advise command notifying the non-volatile memory controller 712 to keep memory associated with the certain address range on the non-volatile memory media 718, in one embodiment, and to keep context metadata, such as forward mapping metadata as described below, for that memory address range in the volatile memory buffer 720 for ready access. In one embodiment, the non-volatile memory controller 712 may send an acknowledgment after receiving a hint command and/or executing the hint command.

A directive command may also identify to the non-volatile storage controller 712 one or more memory units for deallocation. However, in one embodiment, the directive may require the non-volatile storage controller 712 to erase the non-volatile storage media comprising the memory units and/or destroy data of the memory units such that the data is unusable. In one embodiment, the non-volatile storage controller 712 returns an acknowledgment in response to executing the directive command.

A query command, in one embodiment, queries the non-volatile memory controller 712 for memory attributes including characteristics of the non-volatile memory media 718 and/or information such as how much memory capacity is available, latency information, error correcting code ("ECC") latency, endurance information, status information, throttle rights (how much bandwidth, power, and the like is allocated to certain non-volatile memory modules 710) and the like. A checkpoint command may also be a type of directive or hint command that causes the checkpoint module 1214 to associate checkpoint information with the data on the non-volatile memory media 718 as described below.

The execution module 1104 executes a command within the non-volatile memory controller 712. In one embodiment, the execution module 1104 executes the command in response to the receiving module receiving the command. In one embodiment, the execution module 1104 determines whether the non-volatile memory controller 712 is capable of satisfying the command. In some embodiments, the non-volatile memory controller 712 may not be capable if memory allocation is above a certain level and/or if the non-volatile memory controller 712 needs to perform storage space recovery, if too many solid-state storage memory cells have worn out and are now unusable, and/or the like. The execution module 1104 executes the command in response to determining that the non-volatile memory controller 712 is capable of satisfying the command. As described below, if the execution module 1104 determines that the non-volatile memory controller 712 is incapable of executing the command, the execution module 1104 may trigger a signal to the memory controller 605 indicating such.

In one embodiment, the command received by the non-volatile memory controller 712 is a synchronous command, however the execution module 1104 may determine that the non-

volatile memory controller 712 may need to execute the command asynchronously (for example to allow time for some memory space recovery such as garbage collection). The execution module 1104 may then execute the command asynchronously in response to determining that the memory controller 605 will accept satisfying the command asynchronously. For example, the non-volatile memory controller 712 may be able to execute the command asynchronously if the memory controller 605 can tolerate allowing time to perform storage space recovery, ECC correction, LEB retirement, and/or the like.

Executing the command may include reading data in response to a read command, writing data in response to a write command, and performing a memory management function or returning memory attributes in response to a memory management command.

Figure 9 depicts another embodiment of the non-volatile memory controller 712 including the receiving module 1102 and the execution module 1104, and further including a notification module 1202, a signal module 1204, a memory maintenance module 1206, a storage module 1208, an index module 1210, an index reconstruction module 1212, a checkpoint module 1214, a map sync module 1216, and a memory space recovery module 1218.

The notification module 1202 notifies the memory controller 605 of memory attributes of the non-volatile memory, thus allowing the memory controller 605 to distinguish among different memory modules 710, 1008. These attributes may include, but are not limited to performance attributes, memory wear-out attributes, and quality of service attributes. writing frequency, reading frequency, power use characteristics, performance, persistence characteristics, volatility characteristics, wear-out characteristics, and the like. Instead of memory with assumed characteristics (and in fact characteristics that are implicitly required due to a communication protocol such as JEDEC), the memory controller 605 may obtain attributes of different memory modules 710, 1008 with which it communicates. In one embodiment, the memory controller 605 directs data to specific types of memory modules (volatile or non-volatile, single-level cell ("SLC") or multi-level cell ("MLC"), Phase Change Memory (PCM), Resistive Random-access Memory (RRAM), modules comprising a combination of non-volatile memory types, and the like) and/or specific memory divisions (such as physical pages, logical pages, physical erase blocks, logical erase blocks or the like) on the non-volatile memory media and/or the volatile memory media based on the memory attributes. In one embodiment, the notification module 1202 communicates memory attributes to the memory controller 605 over the control path 708 in response to a memory management command.

The signal module 1204 signals to the memory controller 605, using the command protocol, that a command will not be executed in response to the execution module 1104

determining that the non-volatile memory controller 712 is not capable of satisfying the command. As described above, if the non-volatile memory controller 712 is, for some reason, incapable of executing a command, the signal module 1204 may signal to the memory controller 605 that the command will not be executed. Consequently, the memory controller 605 may request execution of the command on another memory module. However, the command may be a read command for data located on the non-volatile memory media 718 of the non-volatile memory controller 712 that received the command. In such an embodiment, the memory controller 605 may have to wait until the non-volatile memory controller 712 can service the read command. In another embodiment, the signal module 1204 signals to the memory controller 605 indicating that the execution module 1104 can execute the command asynchronously rather than synchronously. If the memory controller 712 indicates that asynchronous execution is acceptable (e.g. such as through a command on the control path 708) the execution module 1104 may execute the command asynchronously. Of course in other embodiments, the initial synchronous command from the memory controller 712 may include and indicator that asynchronous execution is permissible.

The memory maintenance module 1206 operates and/or triggers one or more memory maintenance functions for the non-volatile memory module 710 to optimize non-volatile memory performance. Memory maintenance functions may include but are not limited to storage space recovery, power management, thermal management, scanning (evaluating sections of the non-volatile memory media 718 such as physical pages, logical pages, physical erase blocks, logical erase blocks or the like to determine if they should be retired), wear-leveling, defragmentation, and the like. The memory maintenance module 1206 may perform memory maintenance independent of the memory controller 605. For example, the memory maintenance module 1206 may perform memory maintenance functions in response to detecting available free memory media 718 falling below a predetermined threshold

In one embodiment, the memory maintenance module 1206 receives memory management commands from the memory controller 605 (e.g. through the control path 708) instructing the memory maintenance module 1206 to operate and/or trigger memory maintenance functions. For example, the processor 602, an application, host, or the like may detect a period of time in which the processor 602 will not be sending commands to the non-volatile memory controller 712 (e.g. the user sets the host computer into a sleep mode) and may instruct the memory maintenance module 1206 to perform one or more memory maintenance functions during the period.

The storage module 1208 stores data on the non-volatile memory media 718. In one

embodiment, the storage module 1208, stores data sequentially on the non-volatile memory media 718 to preserve an ordered sequence of memory operations performed on the non-volatile memory media 718. Likewise, in one embodiment, the storage module 1208 associates sequence indicators with the data on the non-volatile memory media 718, wherein the sequence indicators determine an ordered sequence of memory operations performed on the non-volatile memory media 718.

For example, the storage module 1208 may store the data of memory operations to the non-volatile memory media 718 sequentially by appending the data to an append point of a sequential, log-based, cyclic writing structure of the non-volatile memory media 718, in the order that the receiving module receives the commands. The log-based structure may optionally record indicators of various commands (e.g. read commands, write commands, memory management commands) received by the receiving module 1102. For example, the log may include a record that the receiving module 1102 received a memory management command at a certain order and point in time with respect to other commands. In another embodiment, the organization of the log-based, cyclic writing structure implicitly captures the order of certain memory operations. For example, if the log writes data in a predetermined order, and the non-volatile memory media 718 is a write-out-of place media, then blocks of data for a common logical memory address occurring multiple times in the log indicate multiple write operations for that logical memory address and the last time data for this logical memory address was written will be the most current version of the data.

Likewise, in one embodiment, the storage module 1208 stores data in a format that associates the data with respective logical memory addresses on the non-volatile memory media 718. Specifically, the storage module 1208 may store the respective logical memory addresses of the data on the non-volatile memory media 718 in association with the corresponding data on the log-based writing structure. For example, the storage module 1208 may store a logical memory address in metadata space or in a packet header with the data. The storage module 1208 may also store a numerical sequence indicator as metadata with data of a command, may use the sequential order of a log-based writing structure as a sequence indicator, or the like. By storing sequence indicators and logical memory addresses of data with the data on the non-volatile memory media 718, the storage module 1208 enables the index reconstruction module 1212 (described below) to reconstruct, rebuild, and/or recover entries in a mapping structure using the stored sequence indicators and logical memory addresses.

The index module 1210 maintains an index of associations between logical memory addresses of the data and physical storage memory locations comprising the data on the non-

volatile memory media 718. In one embodiment, the index module 1210 maps logical memory addresses (e.g. memory addresses from the memory controller 605) to actual physical addresses and/or locations on the non-volatile memory media 718 using the index. In a further embodiment, the index module 1210 uses a single mapping structure as the index to map logical memory addresses of the memory controller 605 to physical addresses specifying actual locations on the non-volatile memory media 718.

The index, in various embodiments, may include a B-tree, B*-tree, B+-tree, a CAM, a binary tree, a hash table, an index, an array, a linked-list, a look-up table, or another mapping data structure. Use of a B-tree as the index in certain embodiments, is particularly advantageous where the logical address space presented to the memory controller 605 is a very large address space (2^64 addressable blocks - which may or may not be sparsely populated). Because B-trees maintain an ordered structure, searching such a large space remains very fast. For example, in one embodiment, the index includes a B-tree with multiple nodes and each node may store several entries. In the example embodiment, each entry may map a variable sized range or ranges of logical memory addresses to a location on the non-volatile memory media 718. Furthermore, the number of nodes in the B-tree may vary as the B-tree grows wider and/or deeper.

In one embodiment, the index of the index module 1210 only includes a node or entry for logical memory addresses that are associated with currently stored data in the non-volatile memory media 718. In this embodiment, membership in the index represents membership in/presence on the non-volatile memory media 718. The index module 1210, in one embodiment, adds entries, nodes, and the like to the index as data is stored on the non-volatile memory media 718 and removes entries, nodes, and the like from the index in response to data being cleared, trimmed, or otherwise deallocated from memory. Similarly, membership in the index may represent valid allocated memory units (such as data pages) on the non-volatile memory media 718. The storage module 1208, in one embodiment, adds entries, nodes, and the like to the index as data is stored on the non-volatile memory media 718 and removes entries, nodes, and the like from the index in response to data being invalidated cleared, trimmed, or otherwise removed from the non-volatile memory media 718.

The non-volatile memory module 710 may include a volatile memory buffer 720. In one embodiment, the index module 1210 stores the index in this volatile memory 720. The index module 1210, in one embodiment, stores at least one copy of some or all of the mapping structure to the non-volatile memory media 718 periodically. By storing the index on the non-volatile memory media 718, in a further embodiment, the mapping of logical memory addresses

of the memory controller 605 to the locations on the non-volatile memory media 718 is persisted, even if the non-volatile memory module 710 undergoes an unexpected or improper shutdown, power loss, or the like. The volatile memory buffer 720 may comprise dynamic random access memory ("DRAM"), static random access memory ("SRAM"), buffer random access memory ("BRAM"), or other suitable volatile memory.

In one embodiment, the index module 1210 uses the index to identify one or more physical addresses of data of a data segment. The physical addresses are identified from one or more logical memory addresses of the data segment, which are identified in commands directed to the non-volatile memory media 718 from the memory controller 605.

The logical memory addresses correspond to one or more data segments relating to the data stored in the non-volatile memory media 718. The one or more logical memory addresses typically include discrete addresses within a logical memory address space where the logical memory addresses sparsely populate the logical memory address space.

Often logical memory addresses used to identify stored data represent a very small number of logical memory addresses that are possible within a name space or range of possible logical memory addresses. Searching this sparsely populated space may be cumbersome. For this reason, the index is typically a data structure that facilitates quickly traversing the index to find a physical address based on a logical memory address. For example, the index may include a B-tree, a content addressable memory ("CAM"), a binary tree, a hash table, or other data structure that facilitates quickly searching a sparsely populated space or range.

While the index may be optimized, or at least designed, for quickly determining a physical address from a logical memory address, typically the index is not optimized for locating all of the data within a specific region of the non-volatile memory media 718. For this reason, the index module 1210 may include a reverse map to determine a logical memory address of a data segment from a physical address. The reverse map is used to map the one or more physical addresses to one or more logical memory addresses and can be used by the index module 1210 or other process to determine a logical memory address from a physical address. The reverse map beneficially maps the non-volatile memory media 718 into erase regions such that a portion of the reverse map spans an erase region of the non-volatile memory media 718 erased together during a memory space recovery operation. The memory space recovery operation (or garbage collection operation) recovers erase regions for future storage of data. By organizing the reverse map by erase region, the memory space recovery module described below can efficiently identify an erase region for memory space recovery and identify valid data.

The index reconstruction module 1212 reconstructs the index using the logical memory

addresses and the sequence indicators associated with the data on the non-volatile memory media 718. In one embodiment, reconstructing the index includes replaying a sequence of changes made to the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media 718.

The index reconstruction module 1212, in one embodiment, reconstructs the index and included entries by scanning data on the non-volatile memory media 718, such as a sequential log-based writing structure or the like, and extracting logical memory addresses, sequence indicators, and the like from data at physical locations on the non-volatile memory media 718. For example, as described above, in certain embodiments the storage module 1208 stores data of commands in a format that associates the data with sequence indicators for the data and with respective logical memory addresses for the data. If the index becomes lost or corrupted, the index module 1210 may use the physical address or location of data on the non-volatile memory media 718 with the associated sequence indicators, logical memory addresses, and/or other metadata stored with the data, to reconstruct entries of the index.

Where data is stored on the non-volatile memory media 718 sequentially, by keeping track of the order in which erase regions in the non-volatile memory media 718 were filled and by storing logical memory addresses with the data, the non-volatile memory media 718 becomes a sequential log. The index reconstruction module 1212 replays the log by sequentially reading data packets stored on the non-volatile memory media 718. Each physical address and a data packet length of associated data is paired with the logical memory address found in each data packet to recreate the forward and reverse maps.

In one embodiment, reconstructing the index includes rolling back a sequence of changes made to the index from a last change back to a valid checkpoint indicator using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media 718.

The checkpoint module 1214 associates checkpoint information with the data on the non-volatile memory media 718. This checkpoint information determines an ordered sequence of memory checkpoint operations performed on the non-volatile memory media 718 and includes, in some embodiments, a copy of the index or a portion of the index stored onto the non-volatile memory media 718 from the volatile memory buffer 720. The checkpoint module 1214 may store information, such as the index or portions of the index and the reverse map, where the checkpoint is related to a point in time or state of the non-volatile memory module (e.g. a stable state that the map sync module 1216 may roll back to as described below). The stored checkpoint information is sufficient to restore the index and the reverse map to a consistent and

stable state without having to replay the entire log from the beginning. For example, the stored information may include storing the index and reverse maps in non-volatile storage, such as on the non-volatile storage media 718, along with some identifier indicating a state or time checkpoint. Checkpoint information, in one embodiment, is stored in the log itself at the checkpoint. In one embodiment, the checkpoint module 1214 creates a checkpoint in response to the receiving module 1102 receiving a memory management checkpoint command.

In one embodiment, the map sync module 1216 updates the index and the reverse map by replaying the log from the checkpoint, using the checkpoint information as a base. The replaying of the log updates the index mappings to reflect memory operations that occurred since the checkpoint. Beneficially the map sync module 1216 restores the index and reverse map from a checkpoint up to a current state, rather than starting from scratch and replaying the entire contents of the non-volatile memory media 718. The map sync module 1216 uses the checkpoint to go to a data packet stored just after the checkpoint and then replays data packets from that point to a last current stable state of the non-volatile memory media 718. The map sync module 1216 typically takes less time to restore the forward and reverse maps than the index reconstruction module 1212.

The memory space recovery module 1218, upon recovering a section of the non-volatile memory media 718, allows the non-volatile memory controller 712 to re-use the section of the non-volatile memory media 718 to store different data. In one embodiment, the memory space recovery module 1218 adds the recovered section of non-volatile memory media 718 to an available storage pool. The memory space recovery module 1218, in one embodiment, erases existing data in a recovered section. In a further embodiment, the memory space recovery module 1218 allows the non-volatile memory controller 712 to overwrite existing data in a recovered section, where the non-volatile memory media allows overwriting existing data at the same location, without first requiring an erase operation.

In one embodiment, the memory space recovery module 1218 relocates valid data that is in a section of the non-volatile memory media 718 that the memory space recovery module 1218 is recovering to preserve the valid data. In one embodiment, the memory space recovery module 1218 is part of an autonomous garbage collector system that operates within the non-volatile memory module 710. This allows the non-volatile memory module 710 to manage data so that data is systematically spread throughout the non-volatile memory media 718, or other physical memory media, to improve performance, data reliability and to avoid overuse and underuse of any one location or area of the non-volatile memory media 718 and to lengthen the useful life of the non-volatile memory media 718.

The memory space recovery module 1218, upon recovering a section of the non-volatile memory media 718, allows the non-volatile memory controller 712 to re-use the section of the non-volatile memory media 718 to store different data. In one embodiment, the memory space recovery module 1218 adds the recovered section of non-volatile memory media 718 to an available storage pool. The memory space recovery module 1218, in one embodiment, erases existing data in a recovered section. In a further embodiment, the memory space recovery module 1218 allows the non-volatile memory controller 712 to overwrite existing data in a recovered section, where the non-volatile memory allows in-place modification to overwrite existing data of the same location.

In one embodiment, the memory space recovery module 1218 uses the reverse map to identify valid data in an erase region prior to an operation to recover the erase region. The identified valid data is moved to another erase region prior to the recovery operation. By organizing the reverse map by erase region, the memory space recovery module 1218 can scan through a portion of the reverse map corresponding to an erase region to quickly identify valid data or to determine a quantity of valid data in the erase region. An erase region may include an erase block, a fixed number of pages, etc. erased together. The reverse map may be organized so that once the entries for a particular erase region are scanned, the contents of the erase region are known.

By organizing the reverse map by erase region, searching the contents of an erase region is more efficient than searching a B-tree, binary tree, or other similar structure used for logical-to-physical address searches. Searching forward map in the form of a B-tree, binary tree, etc. is cumbersome because the B-tree, binary tree, etc. would frequently have to be searched in its entirety to identify all of the valid data of the erase region. The reverse may include a table, data base, or other structure that allows entries for data of an erase region to be stored together to facilitate operations on data of an erase region.

Figure 10 depicts one embodiment of a method 1300 for a direct interface between a memory controller 605 and a non-volatile memory controller 712 using a command protocol. The method 1300 begins and the receiving module 1102 receives 1302 a command from the memory controller 605. The command may be a read command, a write command, or a memory management command. Next, the execution module 1104 executes 1304 the command within the non-volatile memory controller 712 in response to determining that the non-volatile memory controller 712 is capable of satisfying the command. Then, the method 1300 ends.

Figure 11 depicts one embodiment of a method 1400 for a direct interface between a memory controller 605 and a non-volatile memory controller 712 using a command protocol.

The method 1400 begins and the receiving module 1102 awaits 1402 commands from the memory controller 605. When the receiving module 1102 receives a command, in one embodiment, the execution module 1104 determines 1404 whether the non-volatile memory controller 712 is capable of satisfying the command (e.g the non-volatile memory controller 712 has sufficient resources, and the like). If the execution module 1104 determines 1404 that the non-volatile memory controller 712 is incapable of satisfying the command, the signal module 1204 signals 1406 the memory controller 605 informing the memory controller 605 that this particular non-volatile storage module 710 cannot execute the command and/or informing the memory controller 605 that the non-volatile storage module 710 may execute the command asynchronously.

If the execution module 1104 determines 1404 that the non-volatile memory controller 712 is capable of satisfying the command or if the receiving module 1102 receives 1407 an indication that asynchronous execution of the command is acceptable, the execution module 1104 executes the command. Specifically, if the command is a write command 1408, the execution module 1104 signals the non-volatile memory controller 712 to store data associated with the command. The data is received by the non-volatile memory controller 712 over the data path 706. If the command is a read command 1412, the execution module 1104 signals the non-volatile memory controller 712 to read data requested by the command. Otherwise, if the command is a management command 1416, the execution module 1104 signals the non-volatile memory controller 712 to return memory attributes, perform memory maintenance functions, perform a trim, and/or other functions depending on the type of memory management command over the control path 708. The receiving module 1102 continues to monitor for commands from the memory controller 605.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

CLAIMS

1. A method for a direct interface between a memory controller and a non-volatile memory controller using a command protocol, the method comprising:

receiving a command from a memory controller to a non-volatile memory controller over a wire interface by way of a command protocol, the memory controller coupled to one or more processors, the non-volatile memory controller coupled to non-volatile memory media, the command protocol comprising a control path that enables the memory controller to distinguish among different memory modules, the non-volatile memory controller storing data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media; and

executing the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

2. The method of claim 1, wherein the command received comprises a synchronous command and further comprising executing the command asynchronously in response to determining that the non-volatile memory controller is capable of satisfying the command asynchronously.

3. The method of claim 1, further comprising signaling to the memory controller, using the command protocol, that the command will not be executed in response to determining that the non-volatile memory controller is not capable of satisfying the command.

4. The method of claim 1, wherein the memory controller communicates with volatile memory by way of a second protocol, the command protocol and the second protocol comprising different protocols.

5. The method of claim 4, further comprising notifying the memory controller of memory attributes of the non-volatile memory, the memory controller directing data to one of the non-volatile memory, the volatile memory, a memory division on the non-volatile memory, and a memory division on the volatile memory based on the memory attributes.

6. The method of claim 1, wherein a second memory controller communicates with volatile memory by way of a second protocol, the second memory controller coupled to one or more processors, the command protocol and the second protocol comprising different protocols.

7. The method of claim 6, wherein the second memory controller is coupled to a second

processor and the second processor is coupled to the processor such that memory accessible to the second memory controller is accessible to the processor and the second processor and memory accessible to the memory controller is accessible to the processor and the second processor.

8. The method of claim 1, further comprising associating sequence indicators with the data on the non-volatile memory media, wherein the sequence indicators determine an ordered sequence of memory operations performed on the non-volatile memory media.

9. The method of claim 1, further comprising associating checkpoint information with the data on the non-volatile memory media, wherein the checkpoint information is organized in an ordered sequence of memory checkpoint operations performed on the non-volatile memory media.

10. The method of claim 1, further comprising:

storing the data in a format that associates the data with respective logical memory addresses on the non-volatile memory media;

maintaining an index of associations between logical memory addresses of the data and physical storage memory locations comprising the data on the non-volatile memory media; and

reconstructing the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media, wherein reconstructing the index comprises replaying a sequence of changes made to the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media.

11. The method of claim 10, wherein reconstructing the index comprises rolling back a sequence of changes made to the index from a last change back to a valid checkpoint indicator using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media.

12. The method of claim 1, wherein the command comprises one of a read command, a write command, and a memory management command, the memory management command comprising one or more of a query command, a directive command, and a hint command.

13. The method of claim 1, further comprising operating one or more memory maintenance functions on the non-volatile memory to optimize non-volatile memory performance, wherein the non-volatile memory controller operates the one or more memory maintenance functions according to one of independent of the memory controller and in

response to receiving memory management commands from the memory controller.

14. An apparatus for a direct interface between a memory controller and a non-volatile memory controller using a command protocol, the apparatus comprising:

a receiving module configured to receive a command from a memory controller to a non-volatile memory controller over a wire interface by way of a command protocol, the memory controller coupled to one or more processors, the non-volatile memory controller coupled to non-volatile memory media, the command protocol comprising a control path that enables the memory controller to distinguish among different memory modules, the non-volatile memory controller storing data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media; and

an execution module configured to execute the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

15. The apparatus of claim 14, wherein the memory controller communicates with volatile memory by way of a second protocol, the command protocol and the second protocol comprising different protocols.

16. The apparatus of claim 15, further comprising a notification module configured to notify the memory controller of memory attributes of the non-volatile memory, the memory controller directing data to locations on one of the non-volatile memory and the volatile memory based on the memory attributes.

17. The apparatus of claim 14, further comprising a checkpoint module configured to associate checkpoint information with the data on the non-volatile memory media, wherein the checkpoint information is organized in an ordered sequence of memory checkpoint operations performed on the non-volatile memory media.

18. The apparatus of claim 14, further comprising:

a storage module configured to store the data in a format that associates the data with respective logical memory addresses on the non-volatile memory media;

an index module configured to maintain an index of associations between logical memory addresses of the data and physical storage locations comprising the data on the non-volatile memory media; and

an index reconstruction module configured to reconstruct the index using the

logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media, wherein the index reconstruction module replays a sequence of changes made to the index using the logical memory addresses and the sequence indicators associated with the data on the non-volatile memory media.

19. The apparatus of claim 18, wherein the storage module is further configured to associate sequence indicators with the data on the non-volatile memory media, wherein the sequence indicators determine an ordered sequence of memory operations performed on the non-volatile memory media.

20. A system for a direct interface between a memory controller and a non-volatile memory controller using a command protocol, the system comprising:

one or more processors;

a memory controller coupled to the one or more processors, the memory controller configured to:

generate a command in response to a request from a client;

communicate the command over a wire interface by way of a command protocol, the command protocol comprising a control path that enables the memory controller to distinguish among different memory modules; and

a non-volatile memory controller coupled to non-volatile memory media, the non-volatile memory controller configured to:

receive the command from the memory controller, the non-volatile memory controller storing data sequentially on the non-volatile memory media to preserve an ordered sequence of memory operations performed on the non-volatile memory media; and

execute the command within the non-volatile memory controller in response to determining that the non-volatile memory controller is capable of satisfying the command.

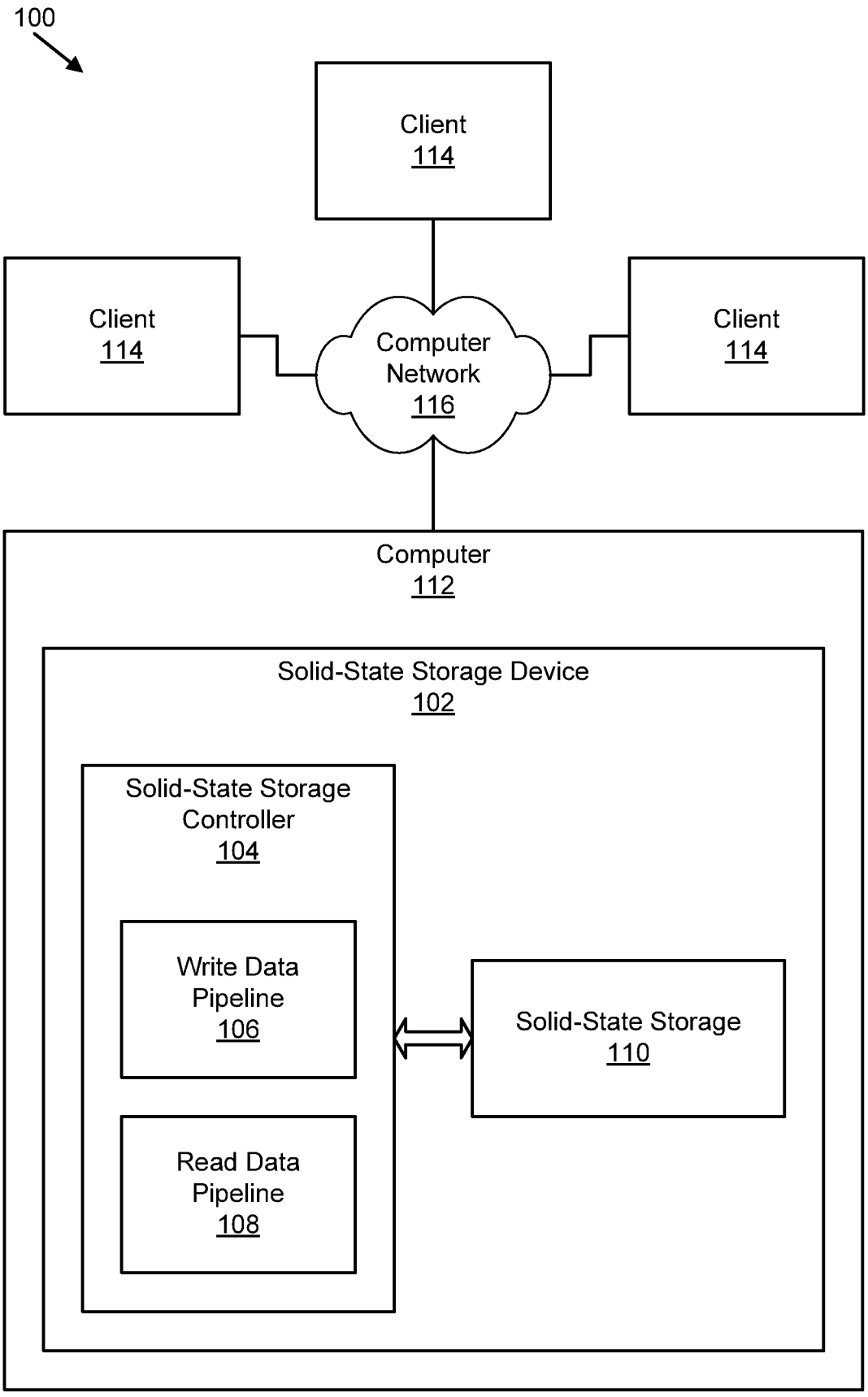
100
Client
114
Client
114
Client
114
Computer
Network
116
Computer
112
Solid-State Storage Device
102
Solid-State Storage
Controller
104
Write Data
Pipeline
106
Read Data
Pipeline
108
Solid-State Storage
110

FIG. 1

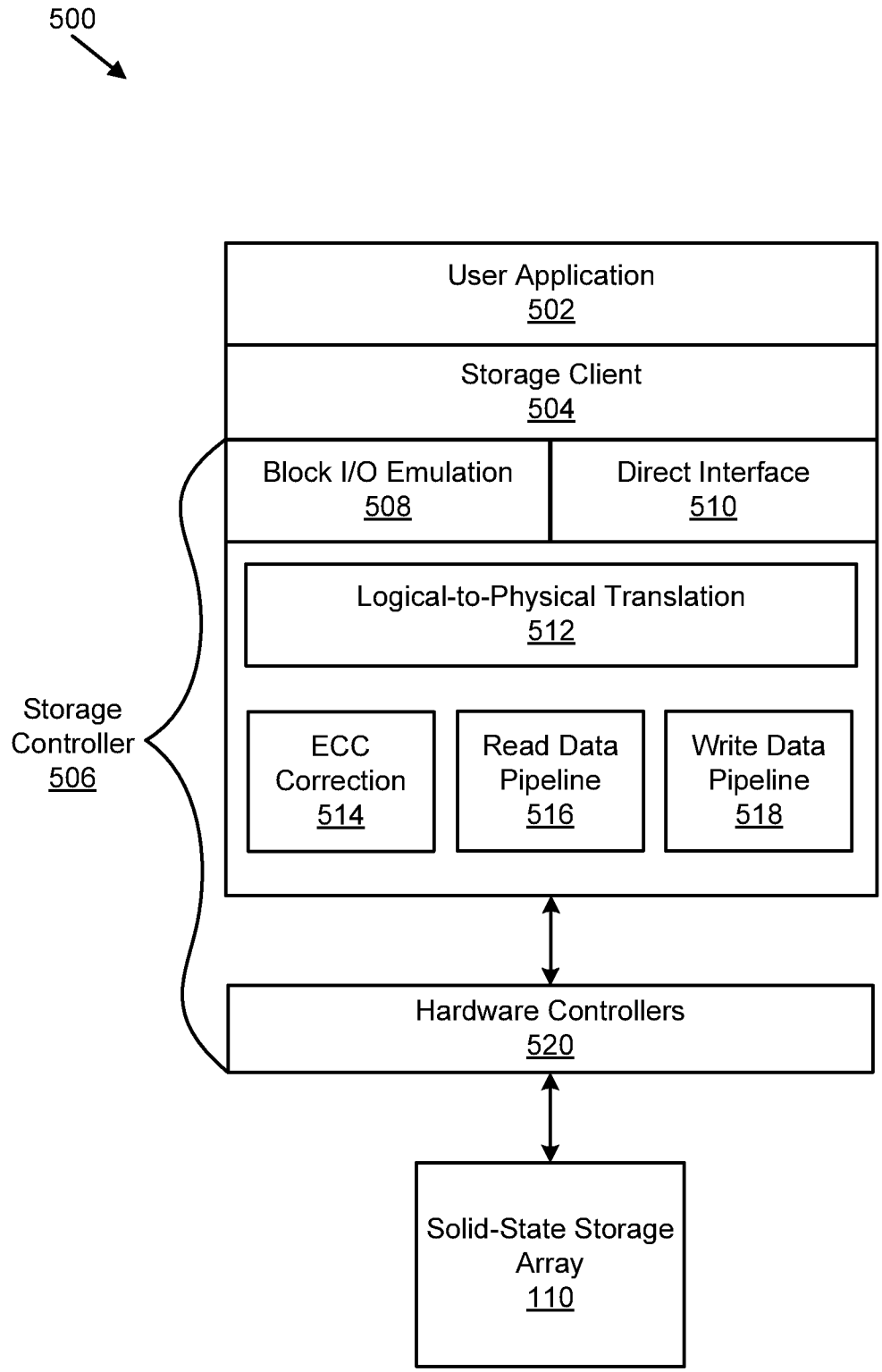
500
User Application
502
Storage Client
504
Block I/O Emulation
508
Direct Interface
510
Logical-to-Physical Translation
512
Storage
Controller
506
ECC
Correction
514
Read Data
Pipeline
516
Write Data
Pipeline
518
Hardware Controllers
520
Solid-State Storage
Array
110

FIG. 2

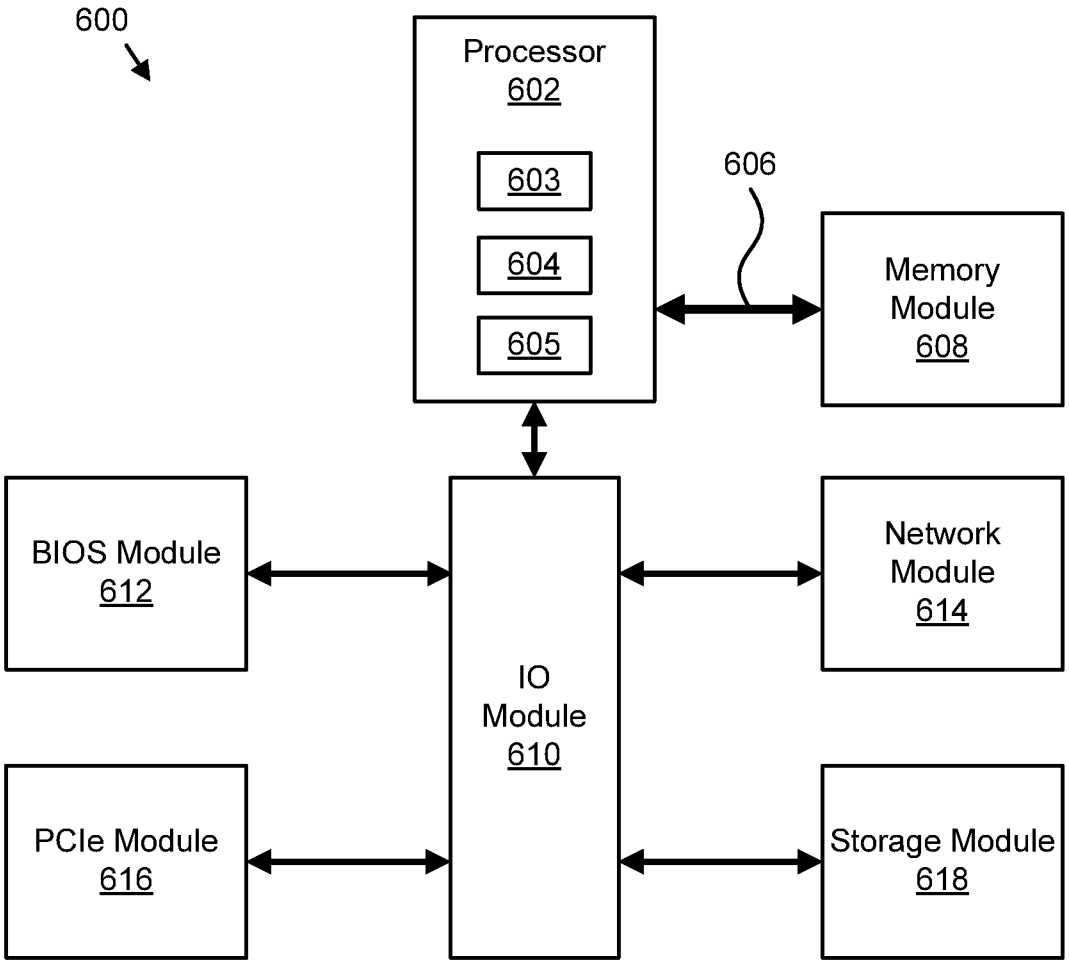
600
Processor
602
603
604
605
606
Memory
Module
608
BIOS Module
612
IO
Module
610
Network
Module
614
PCIe Module
616
Storage Module
618

FIG. 3

PAGE 4/11

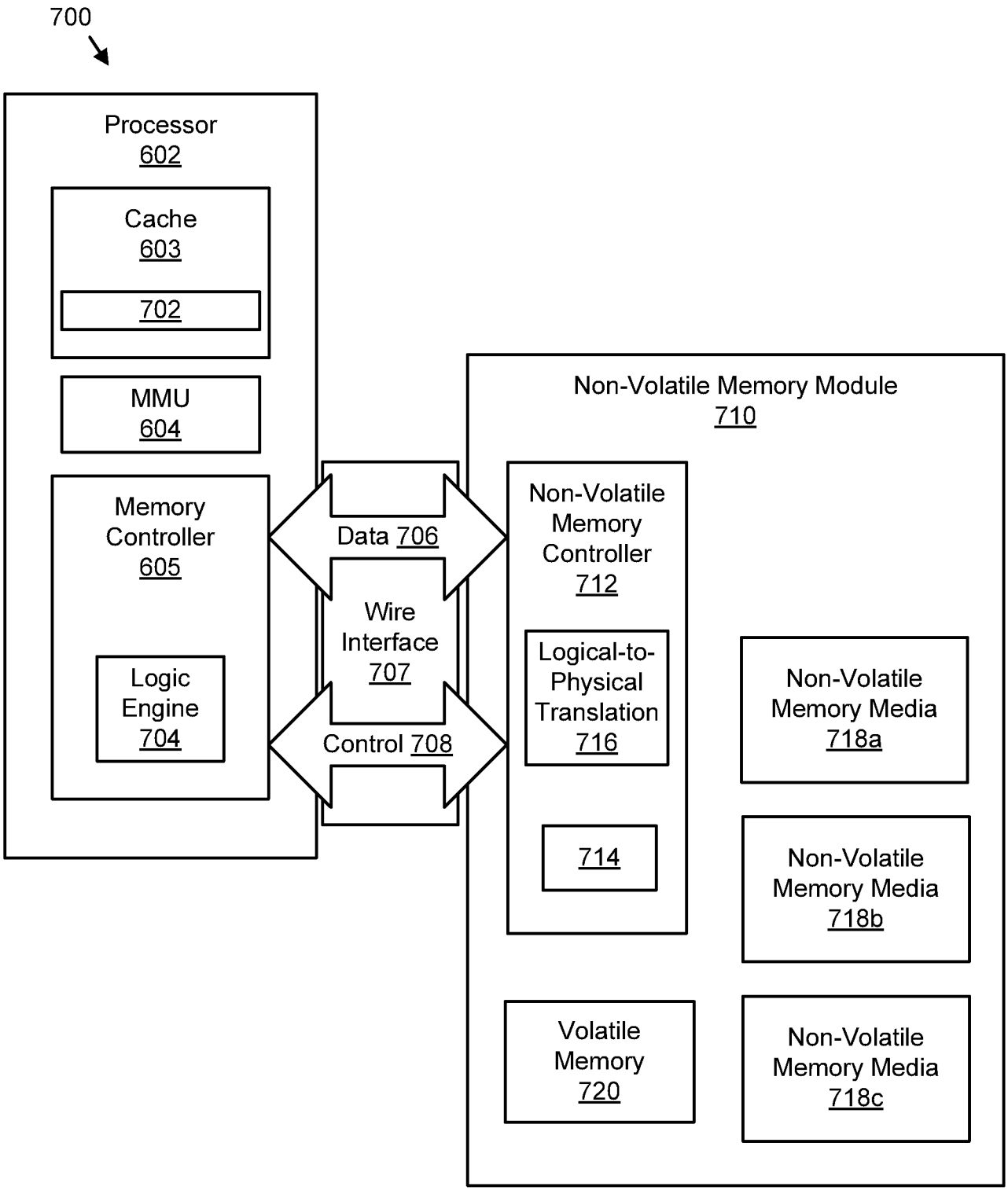

FIG. 4

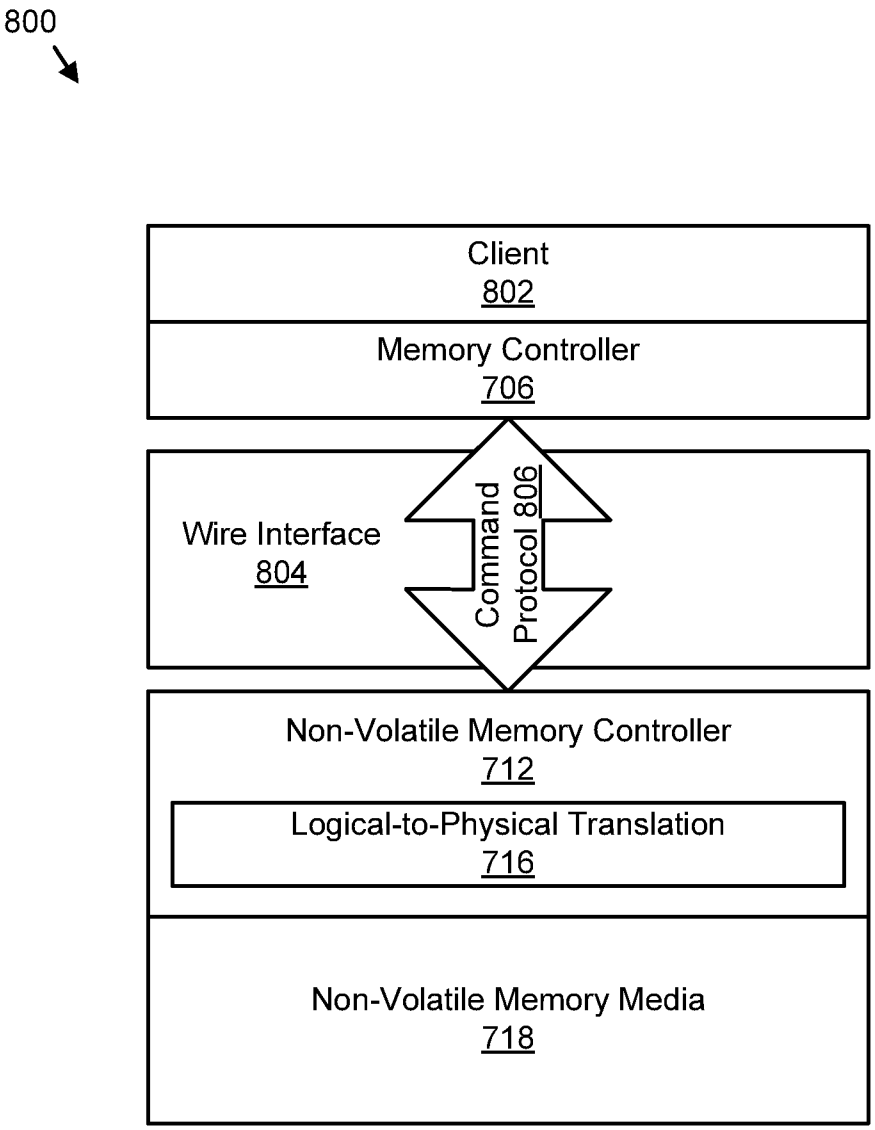
800
Client
802
Memory Controller
706
Wire Interface
804
Command Protocol 806
Non-Volatile Memory Controller
712
Logical-to-Physical Translation
716
Non-Volatile Memory Media
718

FIG. 5

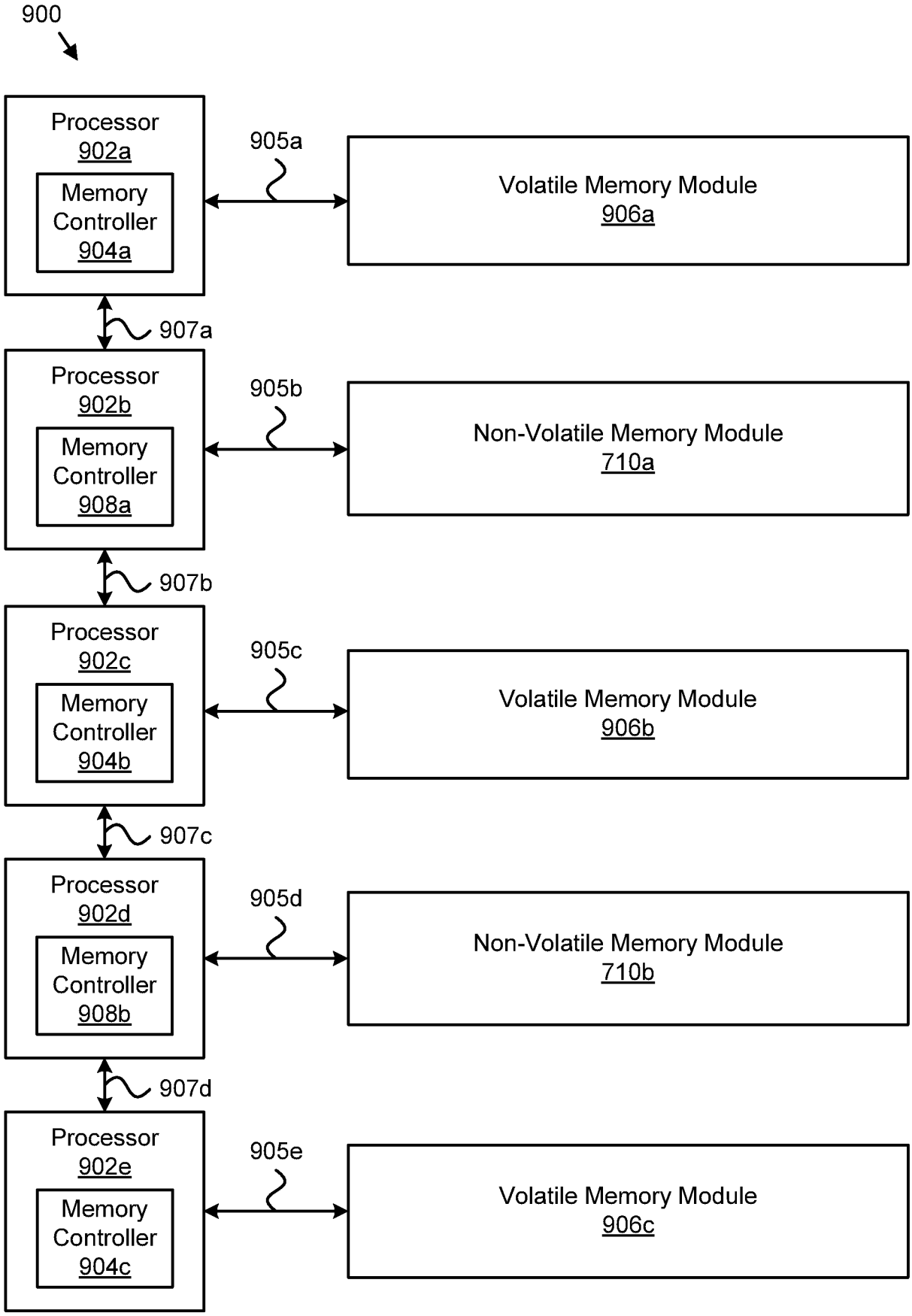
900
Processor
902a
Memory
Controller
904a
905a
Volatile Memory Module
906a
907a
Processor
902b
Memory
Controller
908a
905b
Non-Volatile Memory Module
710a
907b
Processor
902c
Memory
Controller
904b
905c
Volatile Memory Module
906b
907c
Processor
902d
Memory
Controller
908b
905d
Non-Volatile Memory Module
710b
907d
Processor
902e
Memory
Controller
904c
905e
Volatile Memory Module
906c

FIG. 6

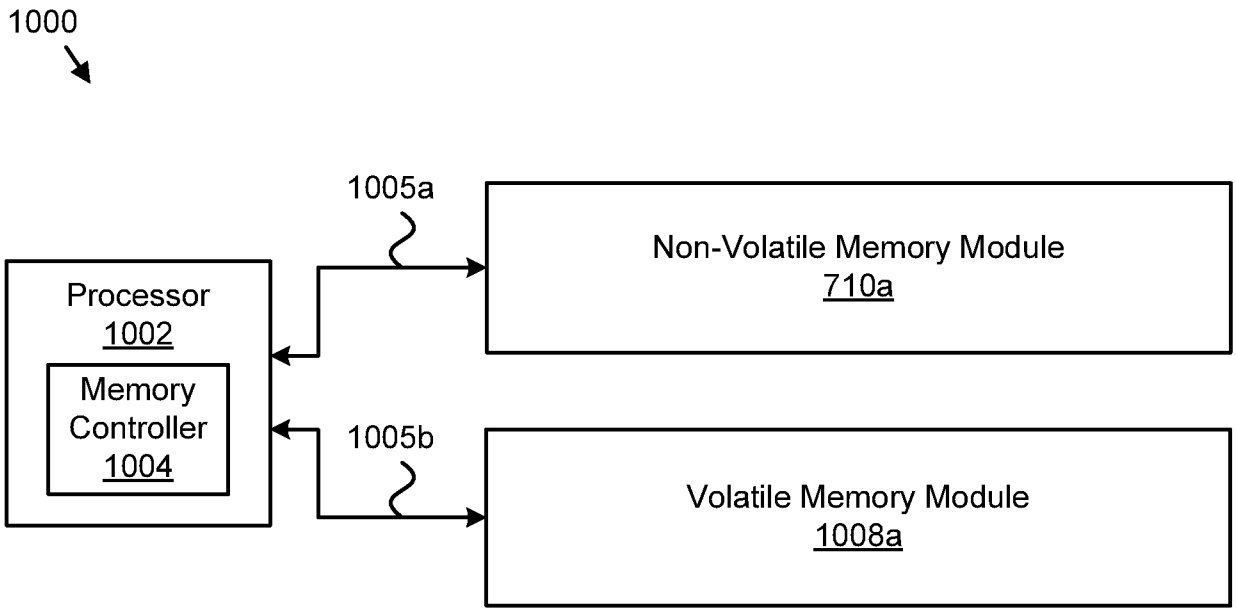
1000
1005a
Processor
1002
Memory
Controller
1004
1005b
Non-Volatile Memory Module
710a
Volatile Memory Module
1008a

FIG. 7A

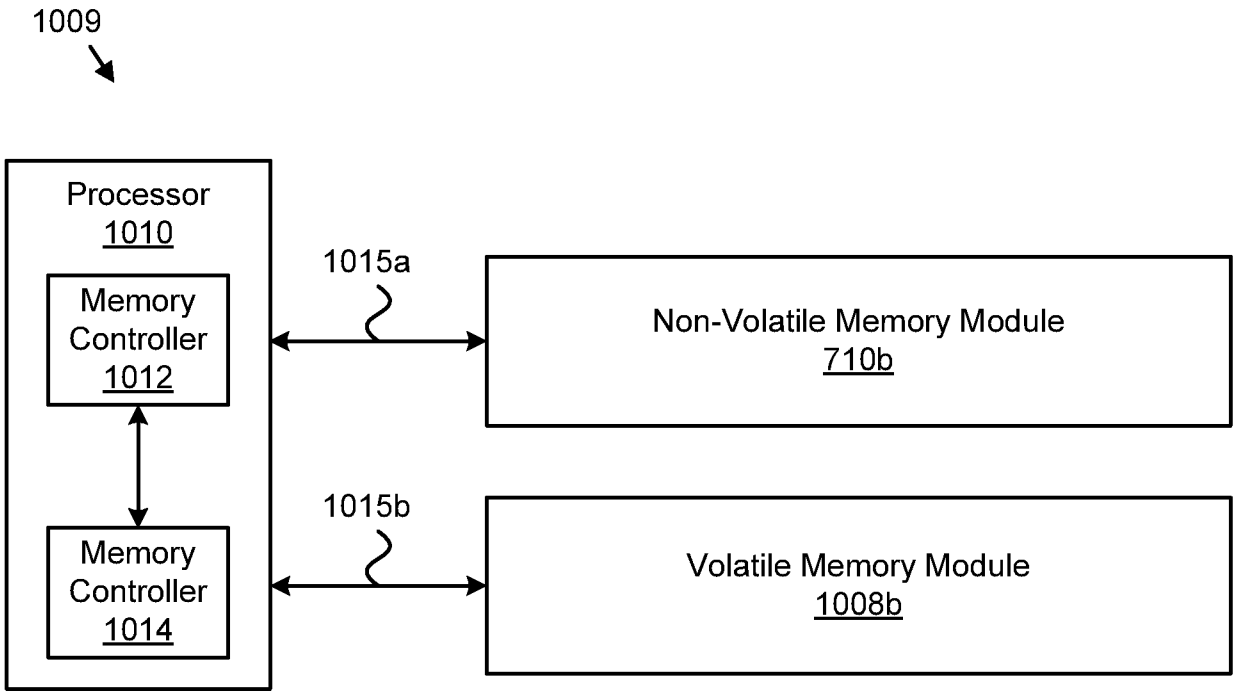
1009
Processor
1010
Memory
Controller
1012
Memory
Controller
1014
1015a
1015b
Non-Volatile Memory Module
710b
Volatile Memory Module
1008b

FIG.7B

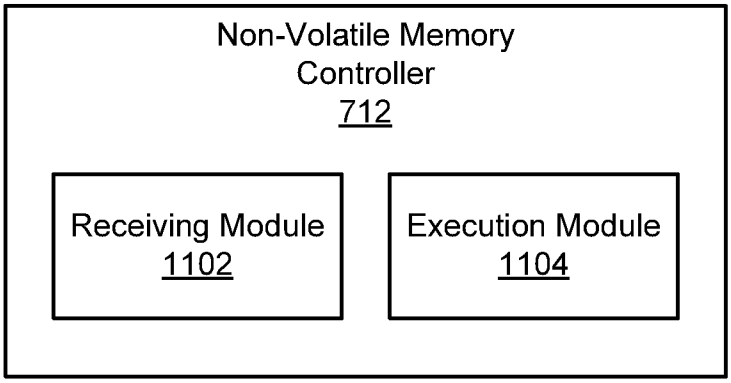
Non-Volatile Memory
Controller
712
Receiving Module
1102
Execution Module
1104

FIG. 8

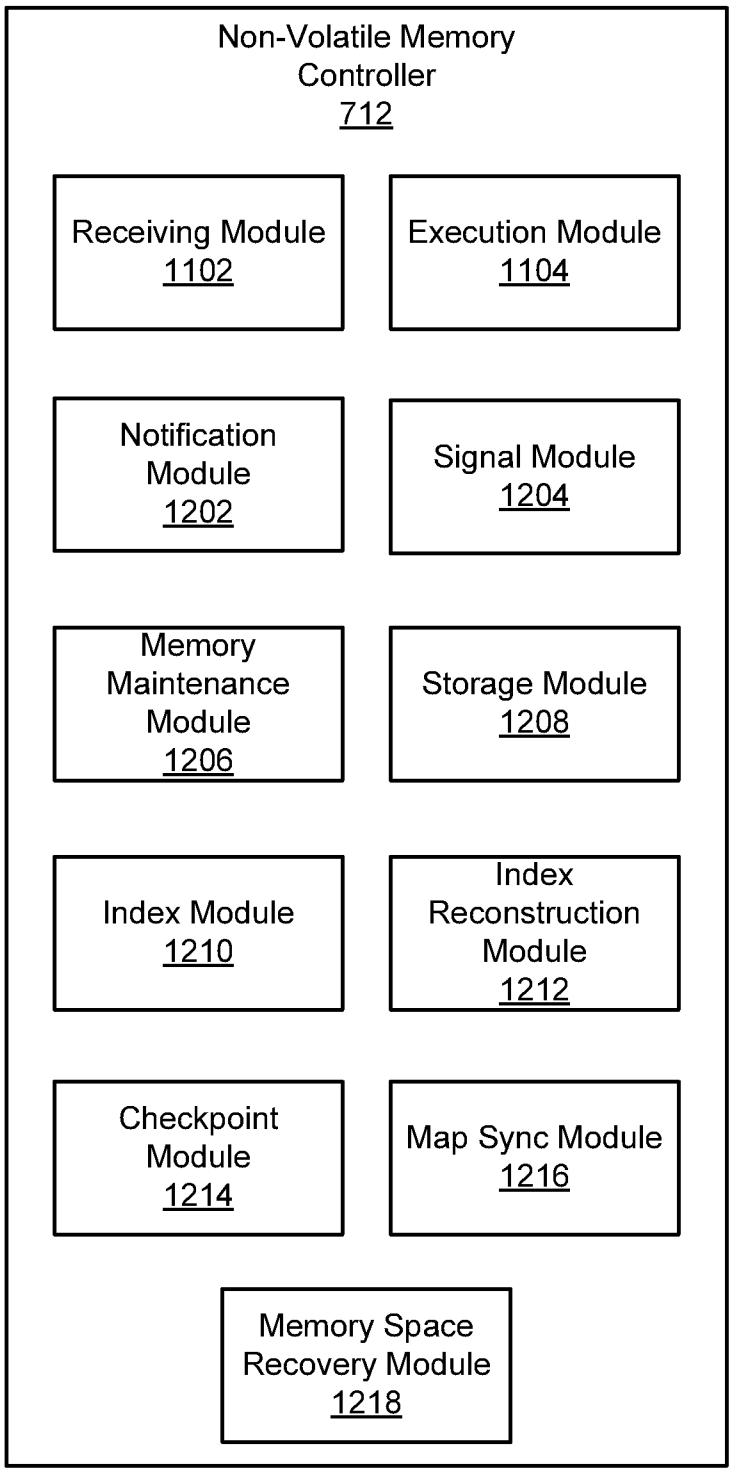
Non-Volatile Memory Controller
712
Receiving Module
1102
Execution Module
1104
Notification Module
1202
Signal Module
1204
Memory Maintenance Module
1206
Storage Module
1208
Index Module
1210
Index Reconstruction Module
1212
Checkpoint Module
1214
Map Sync Module
1216
Memory Space Recovery Module
1218

FIG. 9

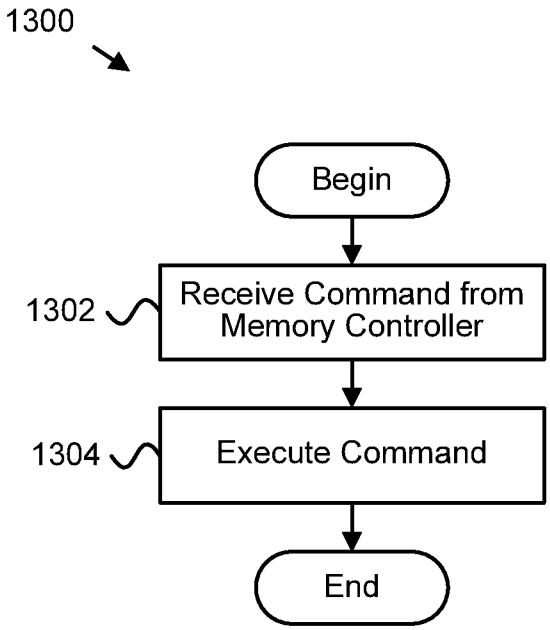
1300
Begin
1302
Receive Command from Memory Controller
1304
Execute Command
End

FIG. 10

PAGE 11/11

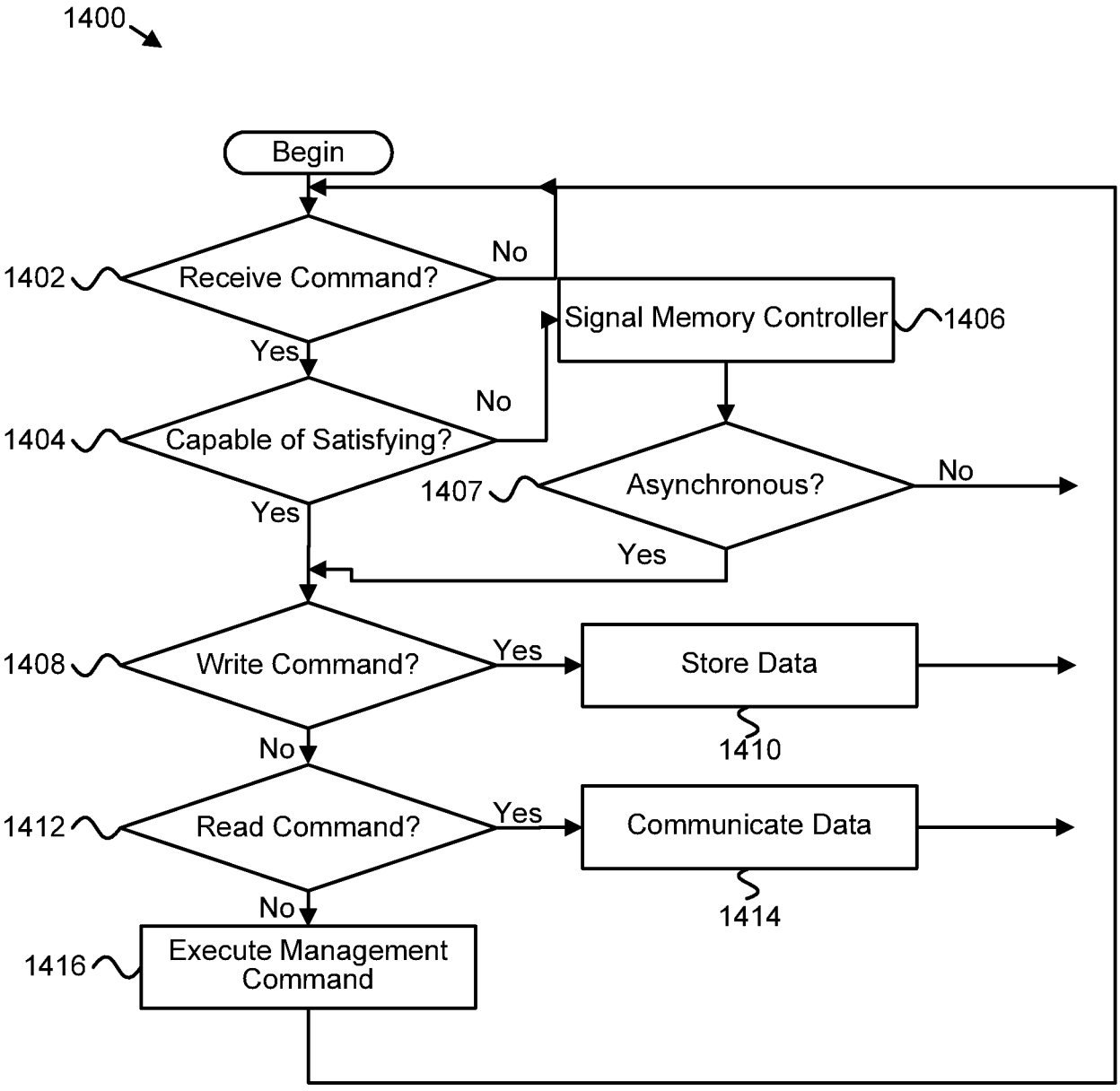

FIG. 11