US 20160210053A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2016/0210053 A1**

(43) **Pub. Date: Jul. 21, 2016**

FRANK et al.

(54) **DISTRIBUTED INDEX FOR FAULT TOLERANT OBJECT MEMORY FABRIC**

(71) Applicant: **ULTRATA LLC**, VIENNA, VA (US)

(72) Inventors: **STEVEN FRANK**, BOULDER, CO (US); **LARRY REBACK**, VIENNA, VA (US)

(73) Assignee: **ULTRATA LLC**, VIENNA, VA (US)

(21) Appl. No.: **15/001,451**

(22) Filed: **Jan. 20, 2016**

Related U.S. Application Data

(60) Provisional application No. 62/105,602, filed on Jan. 20, 2015.

Publication Classification

(51) **Int. Cl.**

G06F 3/06 (2006.01)

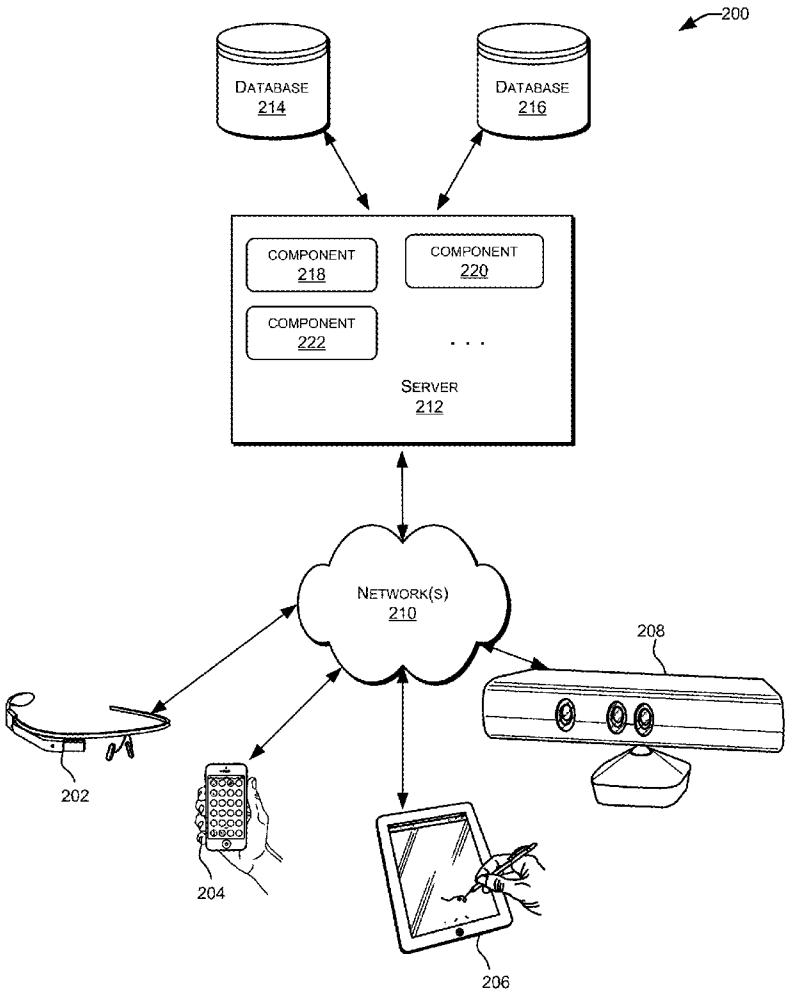
H04L 29/08 (2006.01)

(52) **U.S. Cl.**

CPC ***G06F 3/0604*** (2013.01); ***H04L 67/1097*** (2013.01); ***G06F 3/0644*** (2013.01); ***G06F 3/0659*** (2013.01); ***G06F 3/067*** (2013.01)

(57) **ABSTRACT**

Embodiments of the invention provide systems and methods for managing processing, memory, storage, network, and cloud computing to significantly improve the efficiency and performance of processing nodes. Embodiments can implement an object memory fabric including object memory modules storing memory objects created natively within the object memory module and may be a managed at a memory layer. The memory module object directory may index all memory objects within the object memory module. A hierarchy of object routers communicatively coupling the object memory modules may each include a router object directory that indexes all memory objects and portions contained in object memory modules below the object router in the hierarchy. The hierarchy of object routers may behave in aggregate as a single object directory communicatively coupled to all object memory modules and to process requests based on the router object directories.



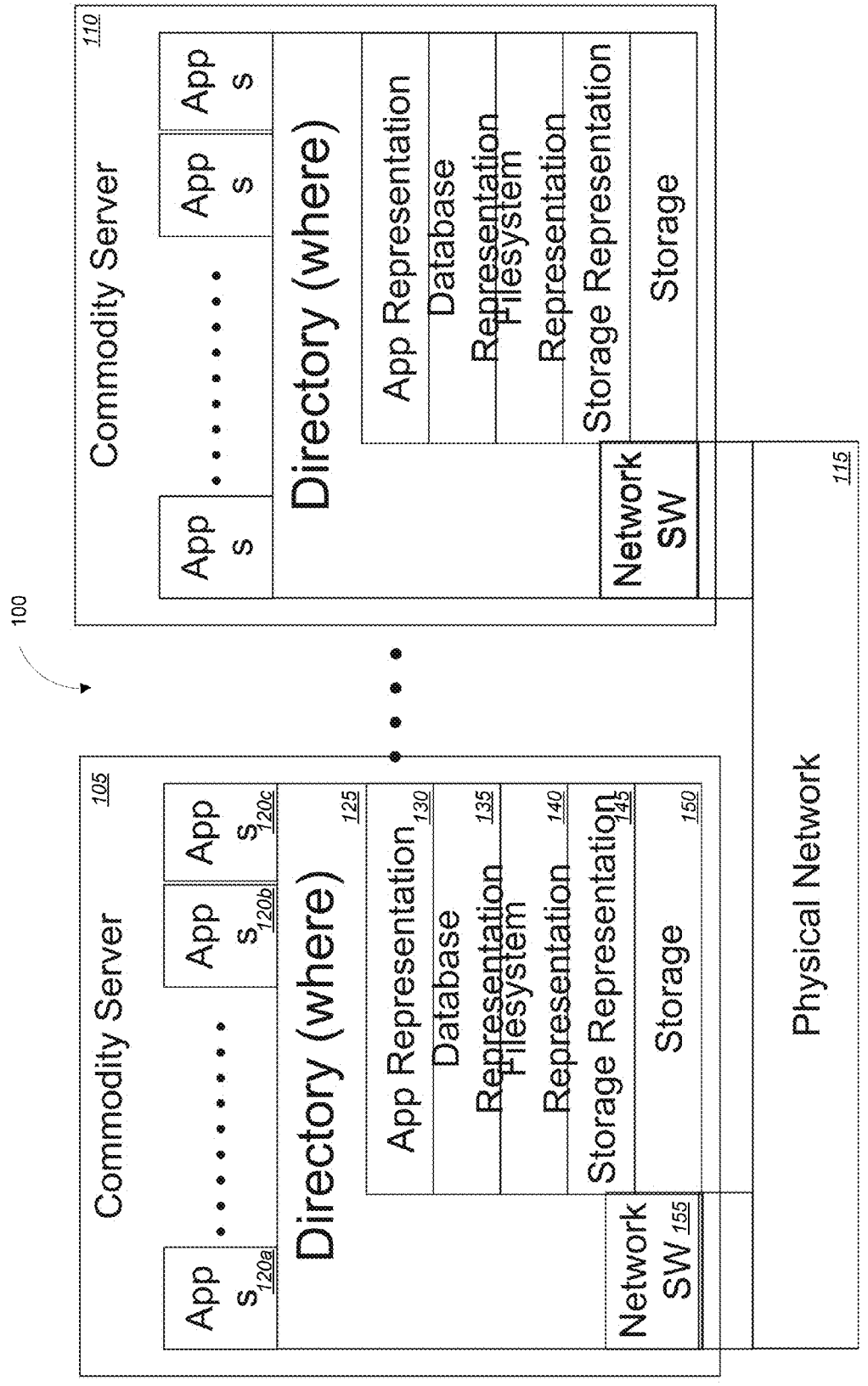


FIG. 1
(Prior Art)

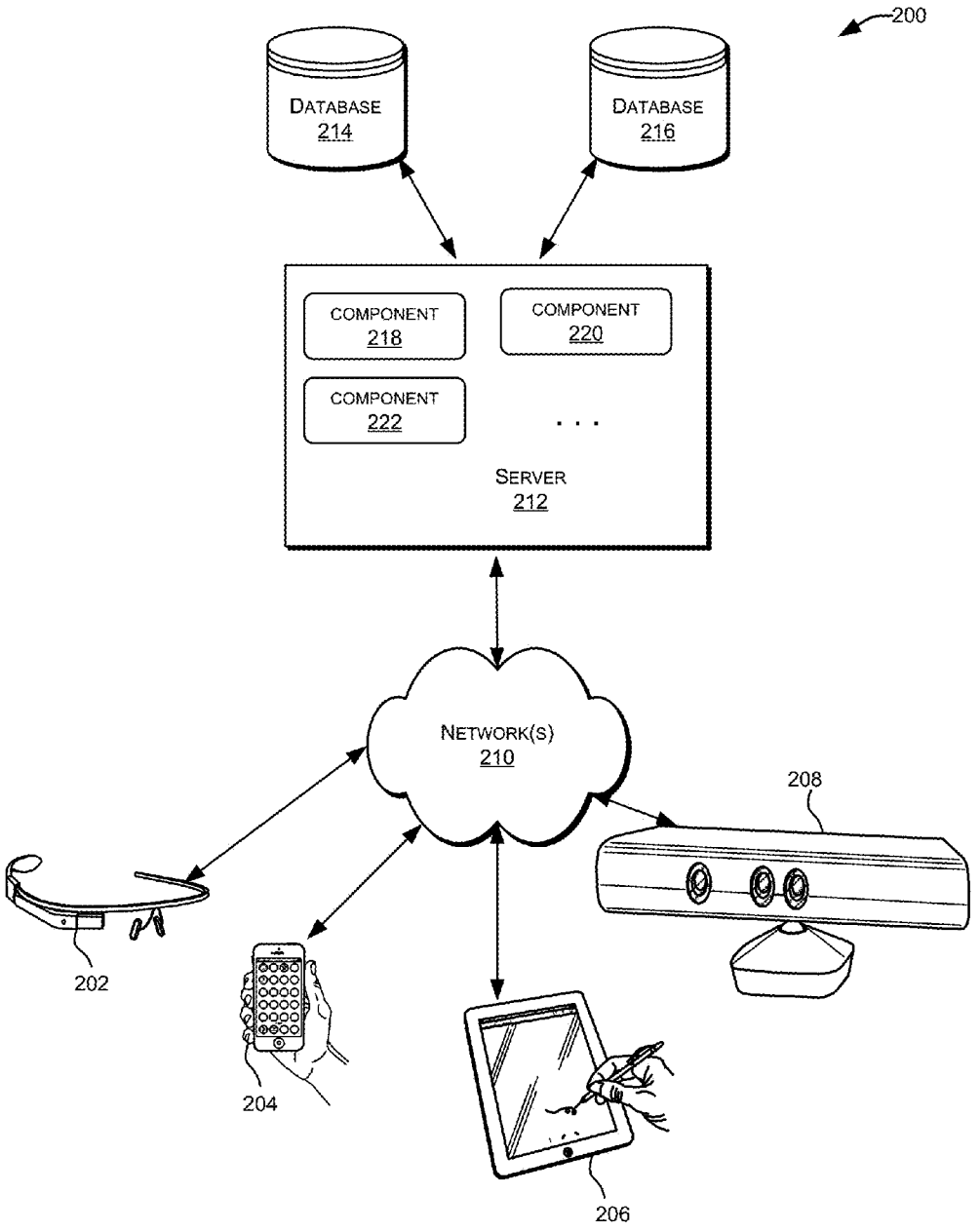

200
DATABASE
214
DATABASE
216
COMPONENT
218
COMPONENT
220
COMPONENT
222
. . .
SERVER
212
NETWORK(S)
210
208
202
204
206

FIG. 2

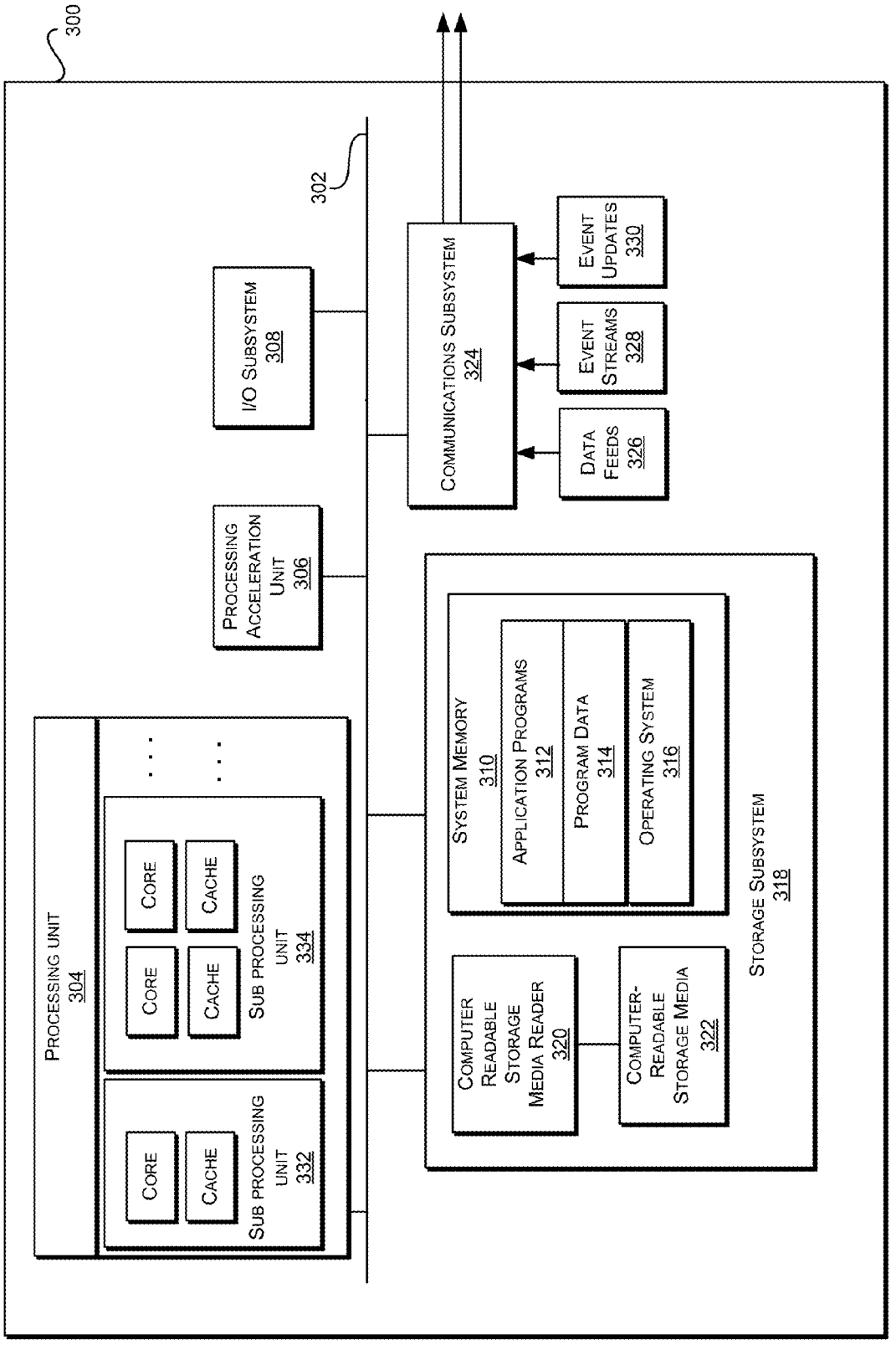

300
302
I/O Subsystem 308
Communications Subsystem 324
Event Updates 330
Event Streams 328
Data Feeds 326
Processing Acceleration Unit 306
Processing Unit 304
Core
Cache
Sub Processing Unit 332
Core
Cache
Sub Processing Unit 334
System Memory 310
Application Programs 312
Program Data 314
Operating System 316
Computer Readable Storage Media Reader 320
Computer-Readable Storage Media 322
Storage Subsystem 318

FIG. 3

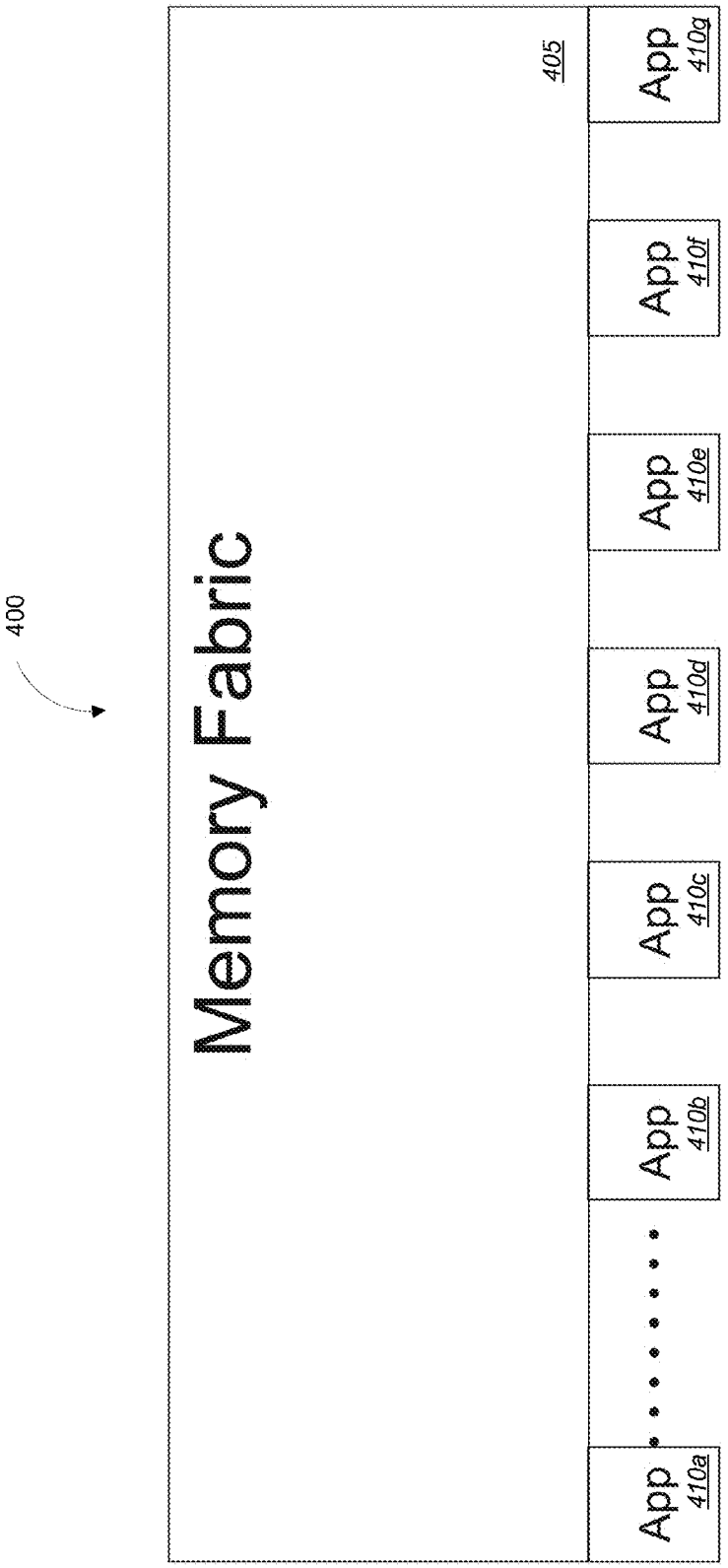

400
Memory Fabric
405
App 410a
App 410b
App 410c
App 410d
App 410e
App 410f
App 410g

FIG. 4

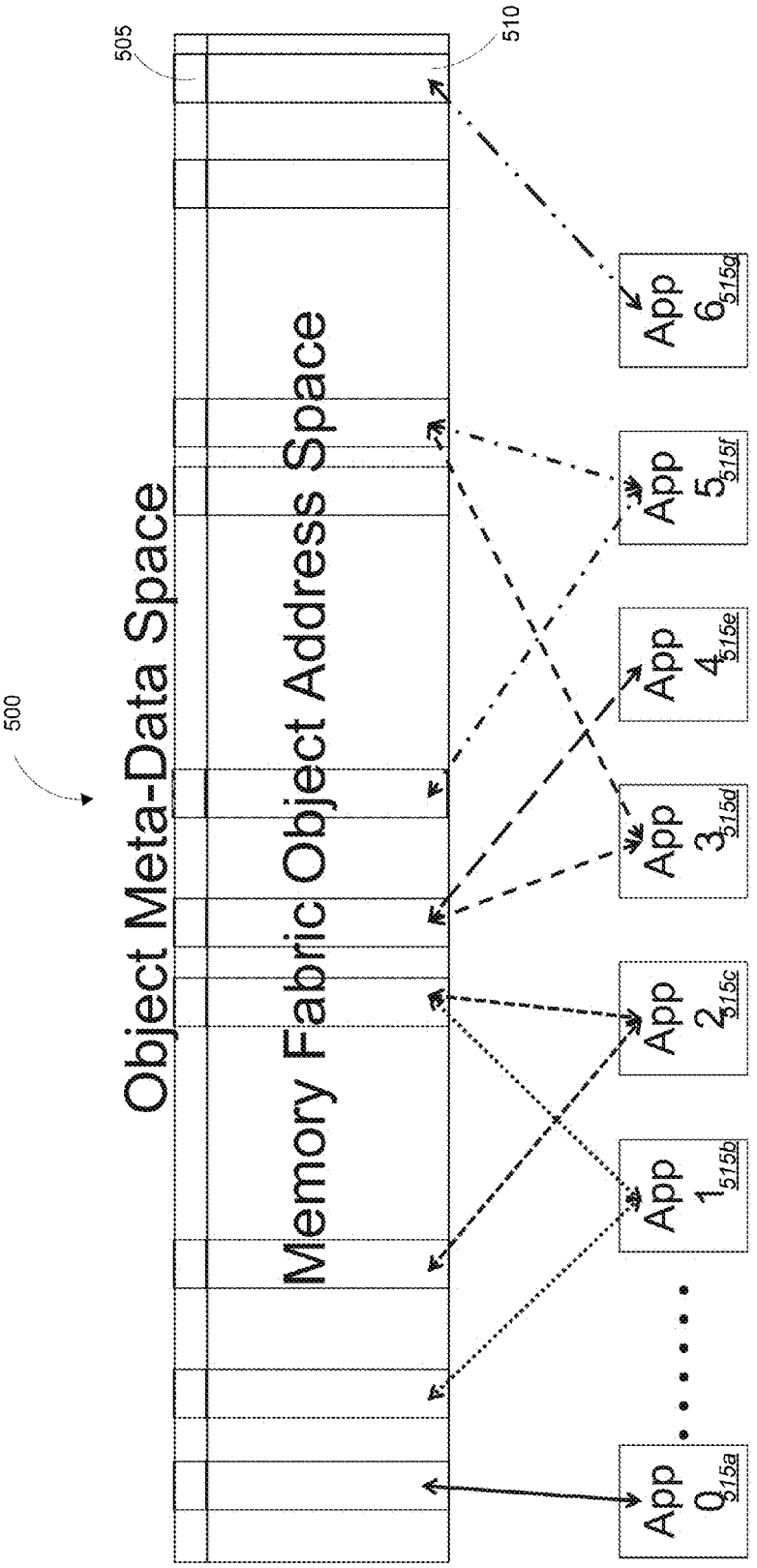

500
505
510
Object Meta-Data Space
Memory Fabric Object Address Space
App 6 515g
App 5 515f
App 4 515e
App 3 515d
App 2 515c
App 1 515b
App Q 515a

FIG. 5

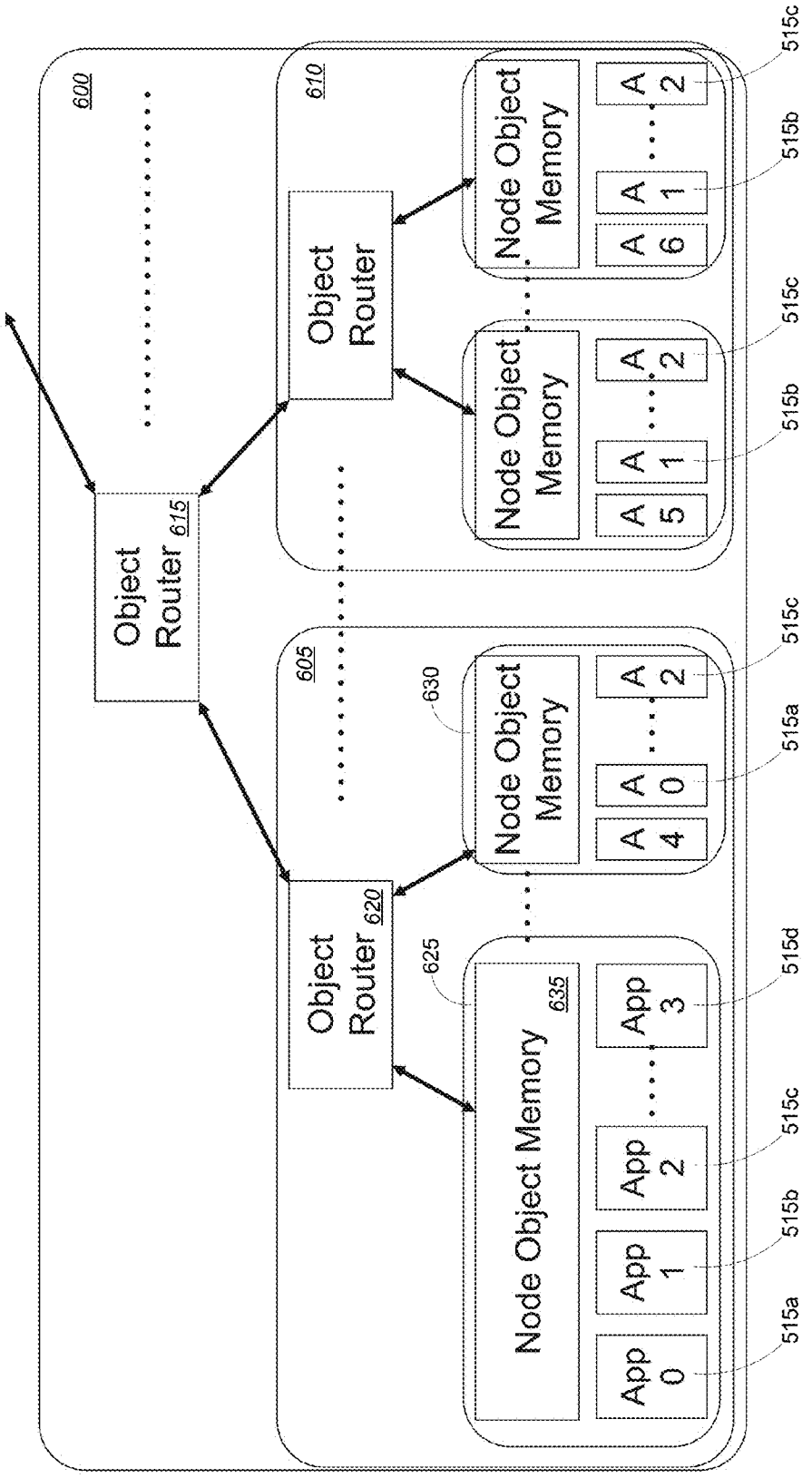

600
Object Router 615
605
610
Object Router 620
Object Router
Object Router
630
625
635
Node Object Memory
Node Object Memory
Node Object Memory
Node Object Memory
App 0
App 1
App 2
App 3
A 0
A 1
A 2
A 4
A 5
A 6
515a
515b
515c
515d

FIG. 6

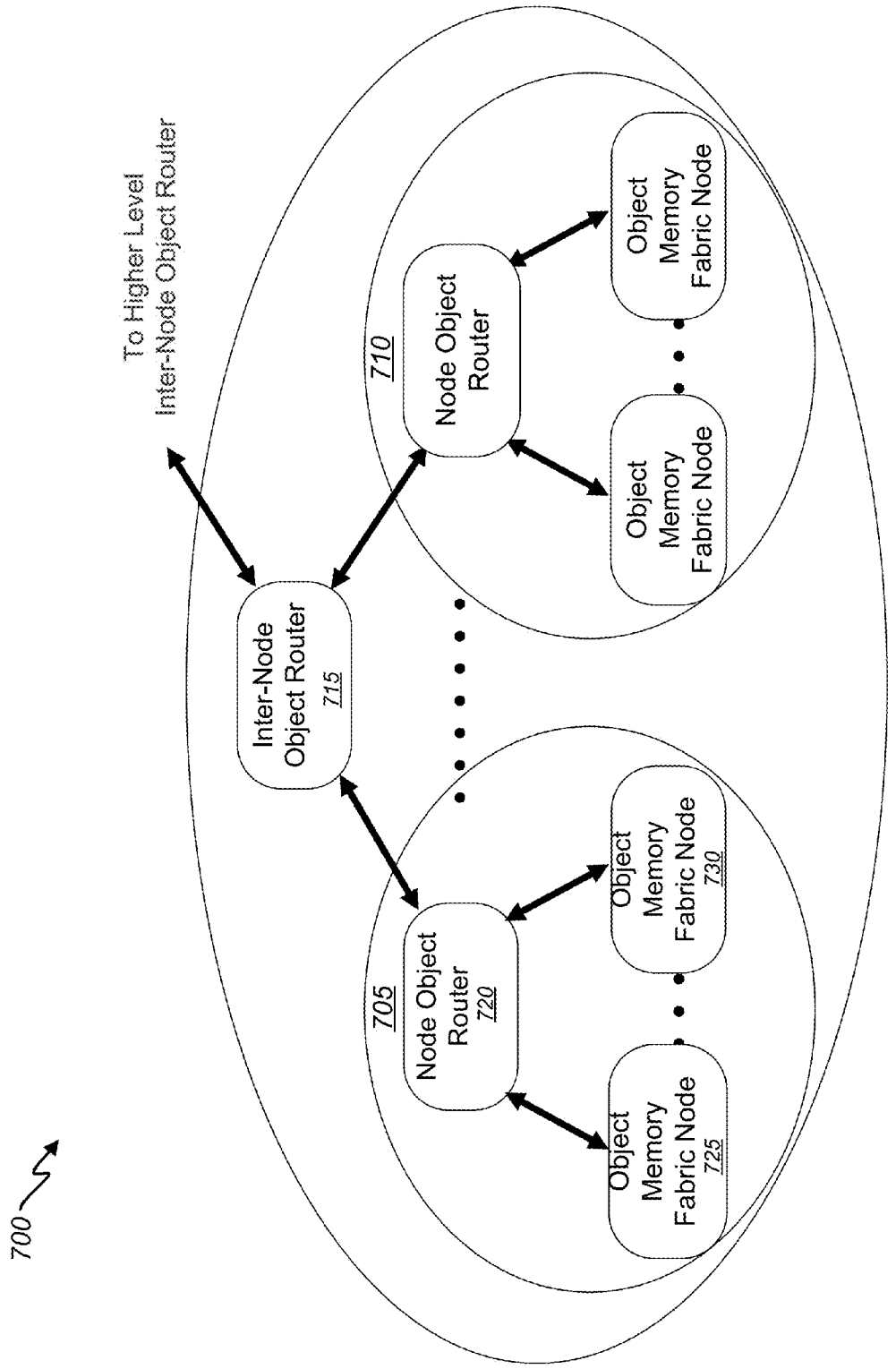

700
To Higher Level
Inter-Node Object Router
Inter-Node
Object Router
715
705
Node Object
Router
720
Object
Memory
Fabric Node
725
Object
Memory
Fabric Node
730
710
Node Object
Router
Object
Memory
Fabric Node
Object
Memory
Fabric Node

FIG. 7

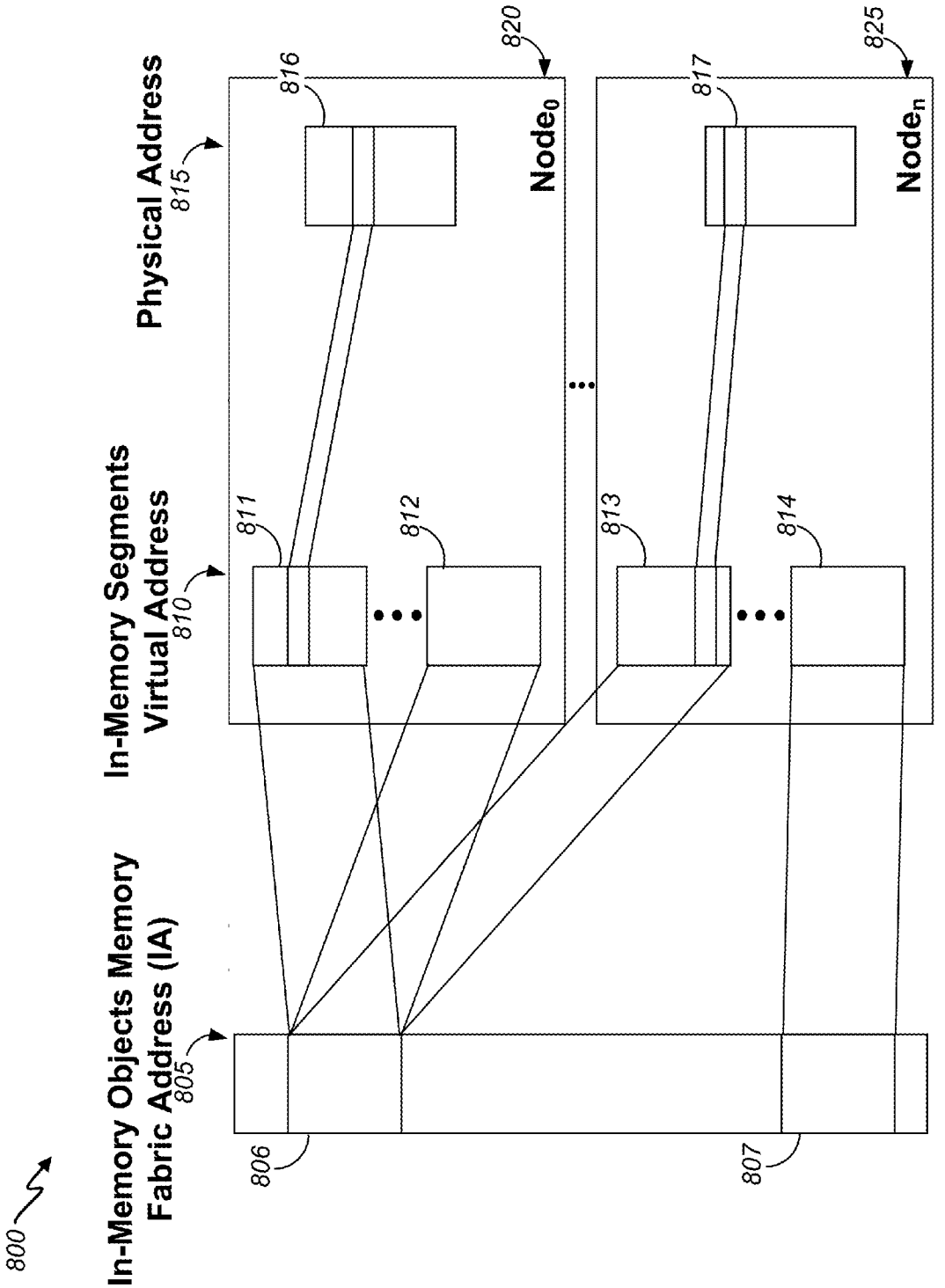

800
In-Memory Objects Memory Fabric Address (IA)
805
In-Memory Segments Virtual Address
810
Physical Address
815
Node0
820
Noden
825
806
807
811
812
813
814
816
817

FIG. 8

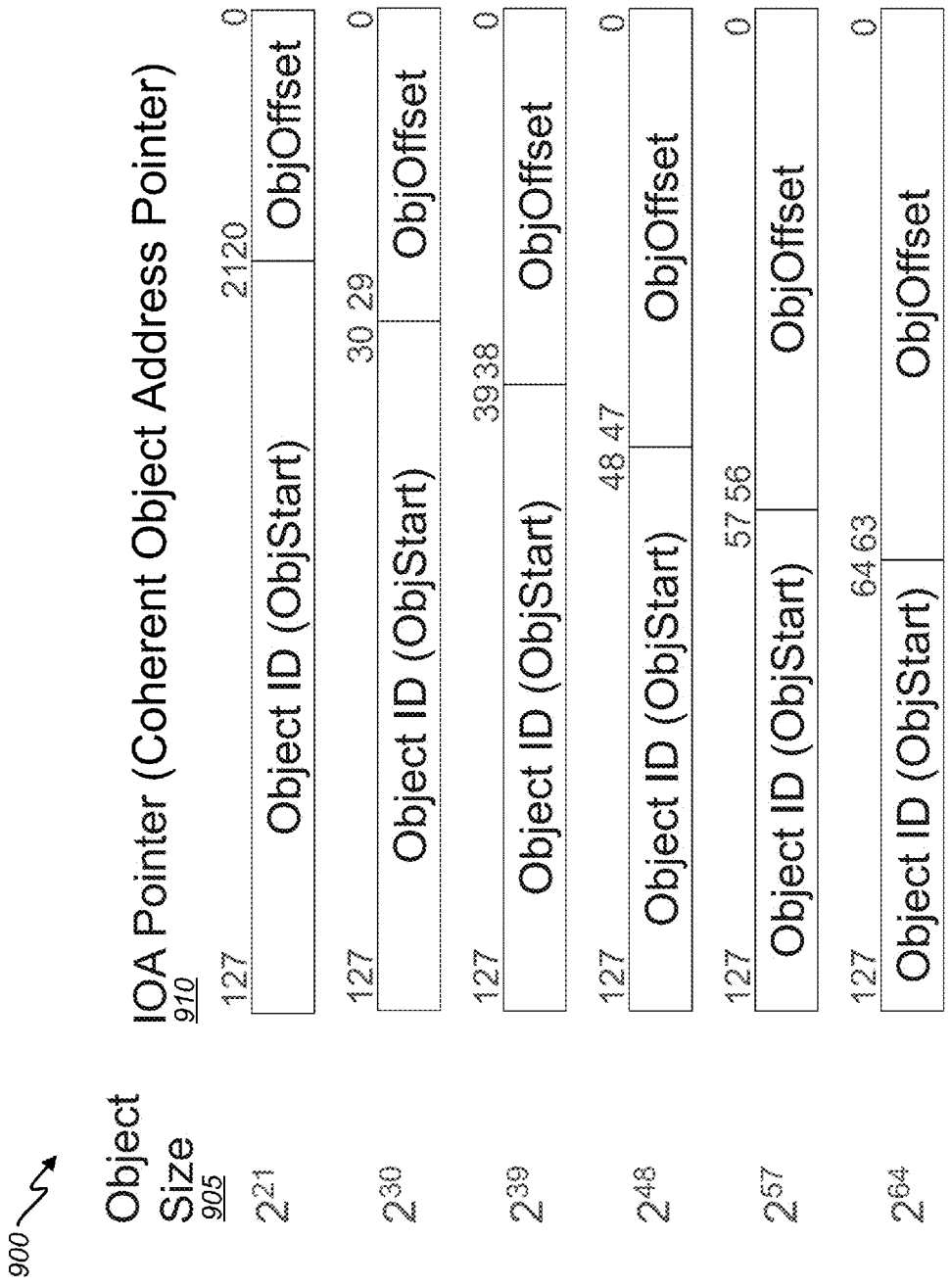

900
Object Size 905
IOA Pointer (Coherent Object Address Pointer) 910
2^{21}
2^{30}
2^{39}
2^{48}
2^{57}
2^{64}
127
0
21 20
30 29
39 38
48 47
57 56
64 63
Object ID (ObjStart)
ObjOffset

FIG. 9

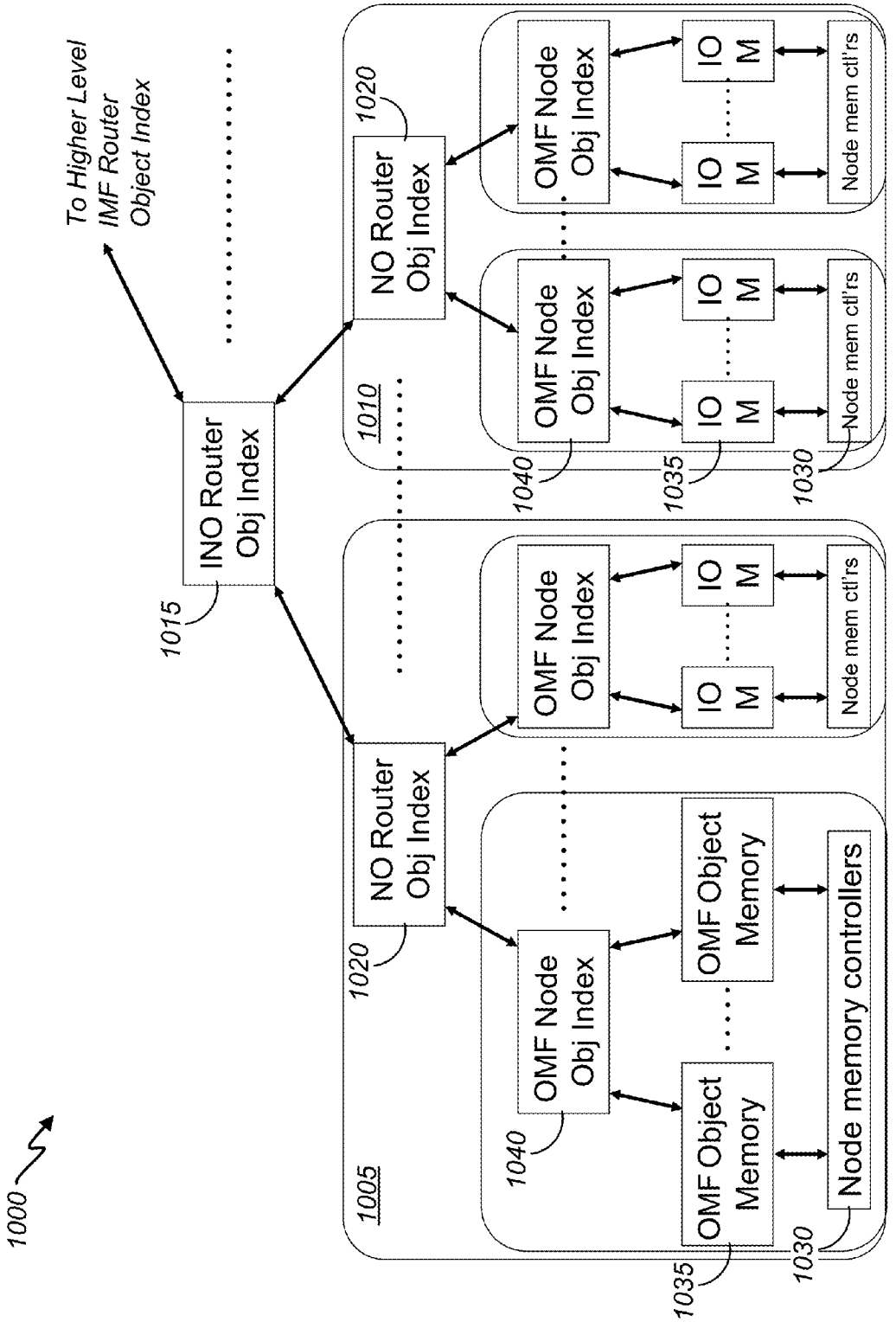

1000
To Higher Level IMF Router Object Index
INO Router Obj Index
1015
NO Router Obj Index
1020
1005
1010
OMF Node Obj Index
1040
OMF Object Memory
1035
IO M
Node memory controllers
Node mem ctl'rs
1030

FIG. 10

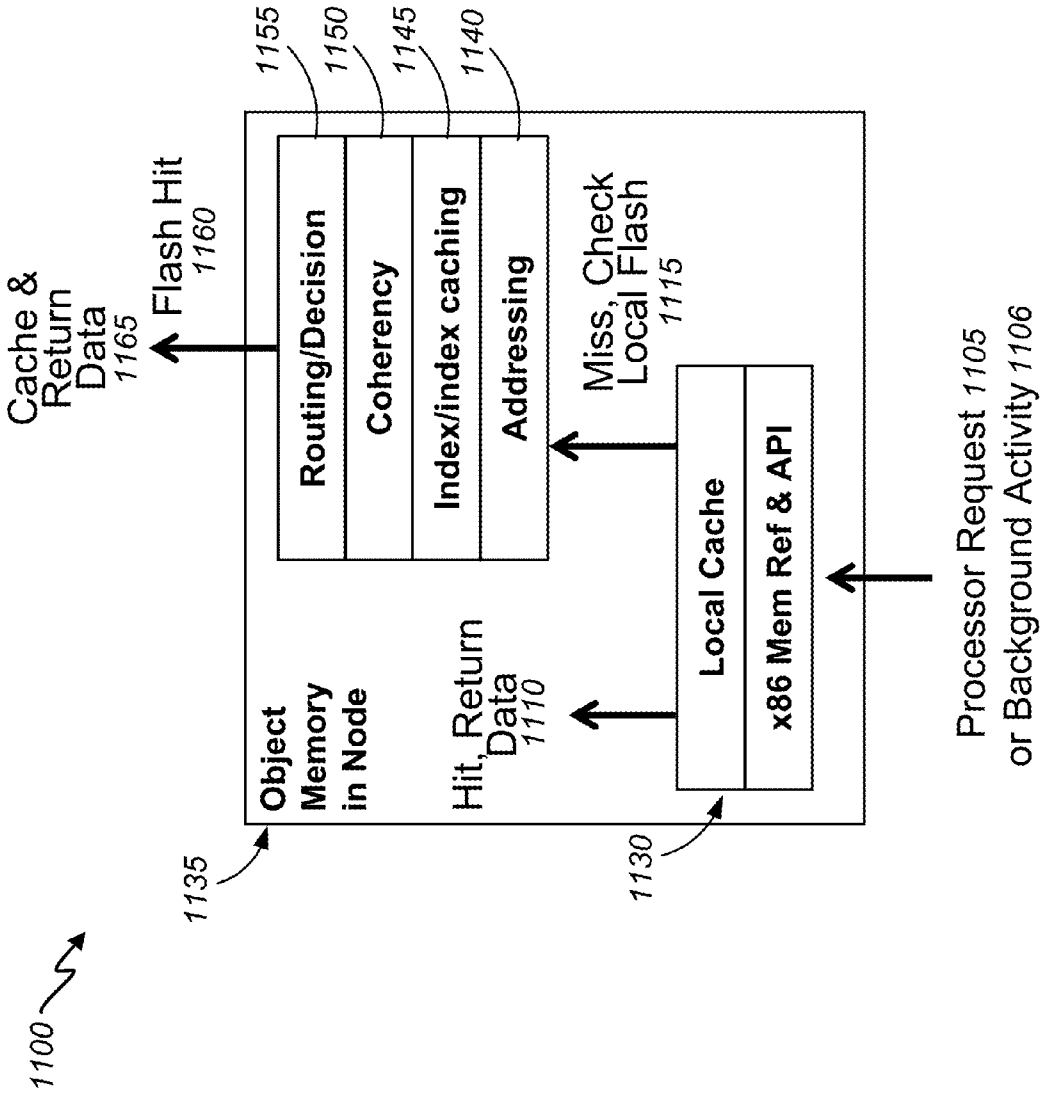

1100
1135
1130
Object Memory in Node
Routing/Decision
Coherency
Index/index caching
Addressing
Local Cache
x86 Mem Ref & API
1155
1150
1145
1140
Cache & Return Data 1165
Flash Hit 1160
Miss, Check Local Flash 1115
Hit, Return Data 1110
Processor Request 1105
or Background Activity 1106

FIG. 11

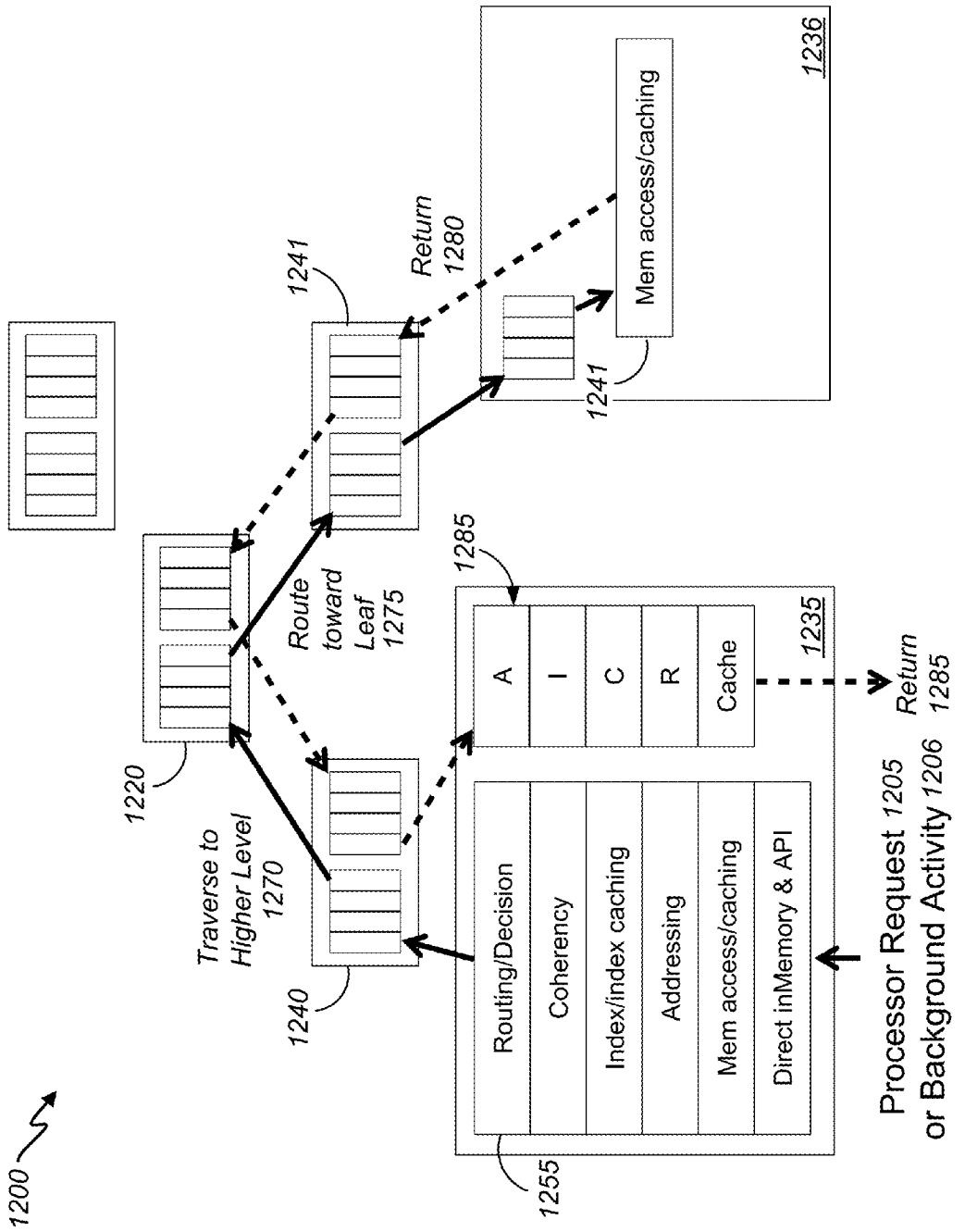

1200
1220
Traverse to Higher Level 1270
1240
Route toward Leaf 1275
1241
Return 1280
1241
Mem access/caching
1236
1285
A
I
C
R
Cache
1235
Routing/Decision
Coherency
Index/index caching
Addressing
Mem access/caching
Direct inMemory & API
1255
Processor Request 1205 or Background Activity 1206
Return 1285

FIG. 12

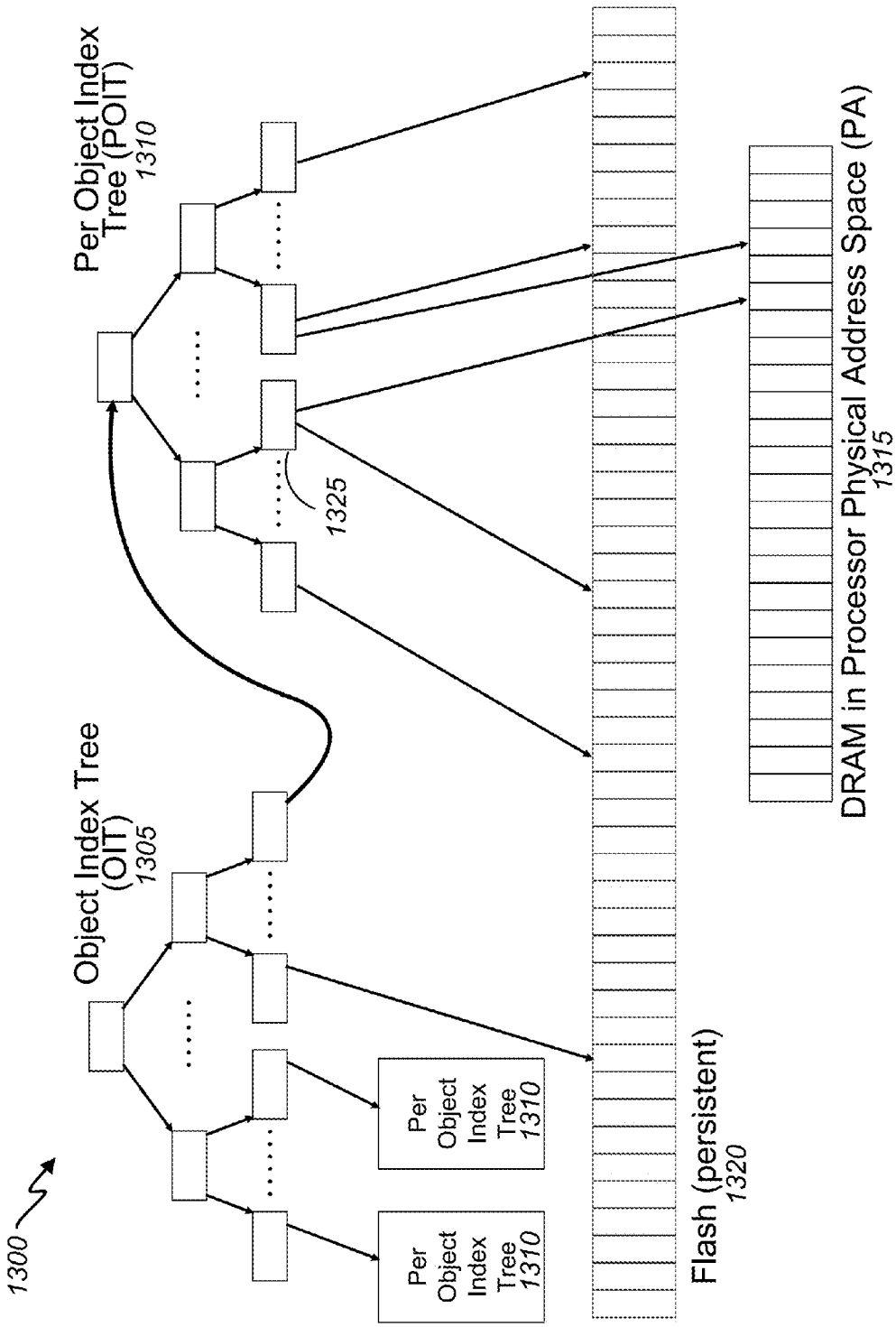

1300
Object Index Tree (OIT) 1305
Per Object Index Tree (POIT) 1310
Per Object Index Tree 1310
Per Object Index Tree 1310
1325
Flash (persistent) 1320
DRAM in Processor Physical Address Space (PA) 1315

FIG. 13

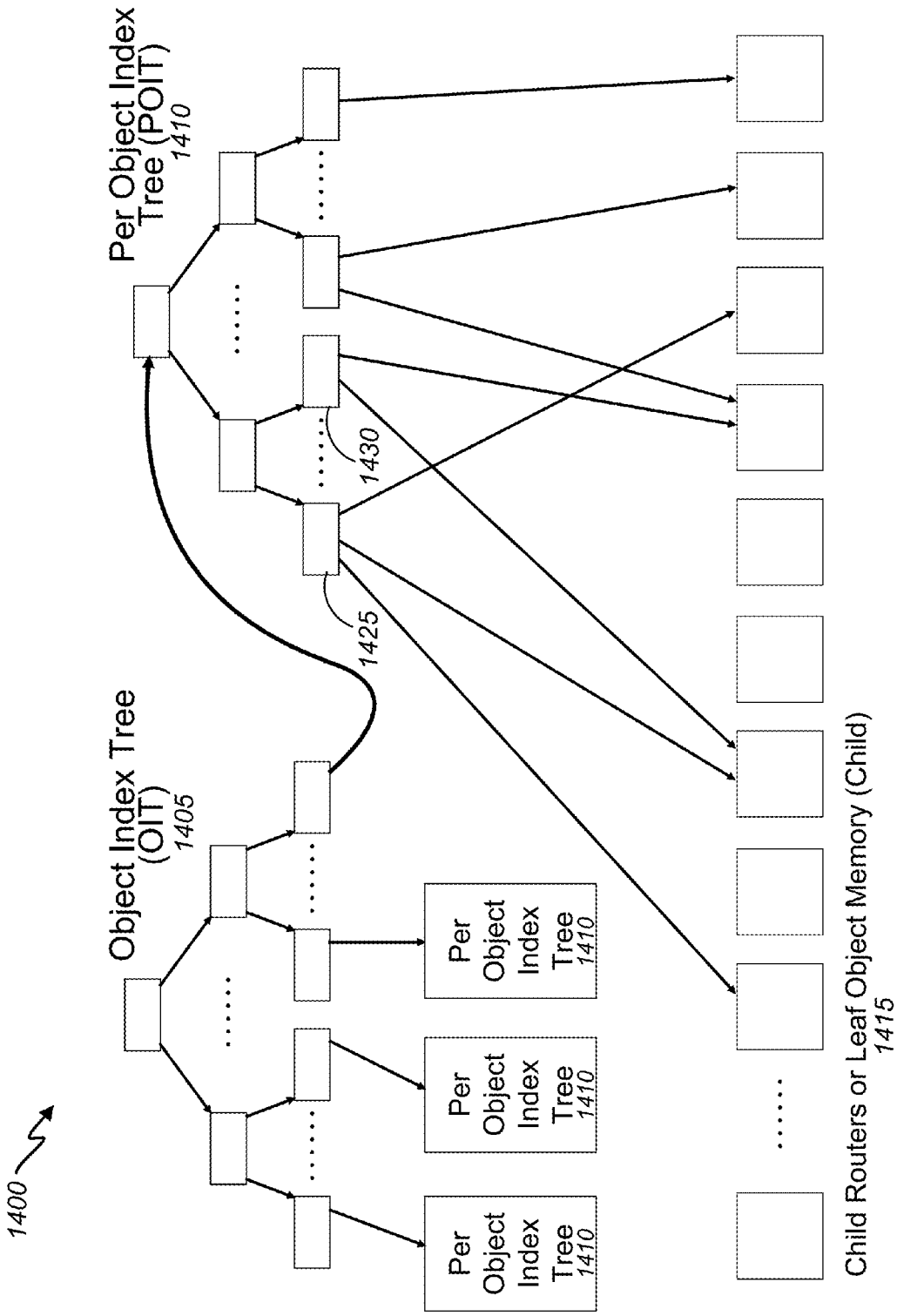

1400
Object Index Tree (OIT) 1405
Per Object Index Tree (POIT) 1410
Per Object Index Tree 1410
Per Object Index Tree 1410
Per Object Index Tree 1410
1425
1430
Child Routers or Leaf Object Memory (Child) 1415

FIG. 14

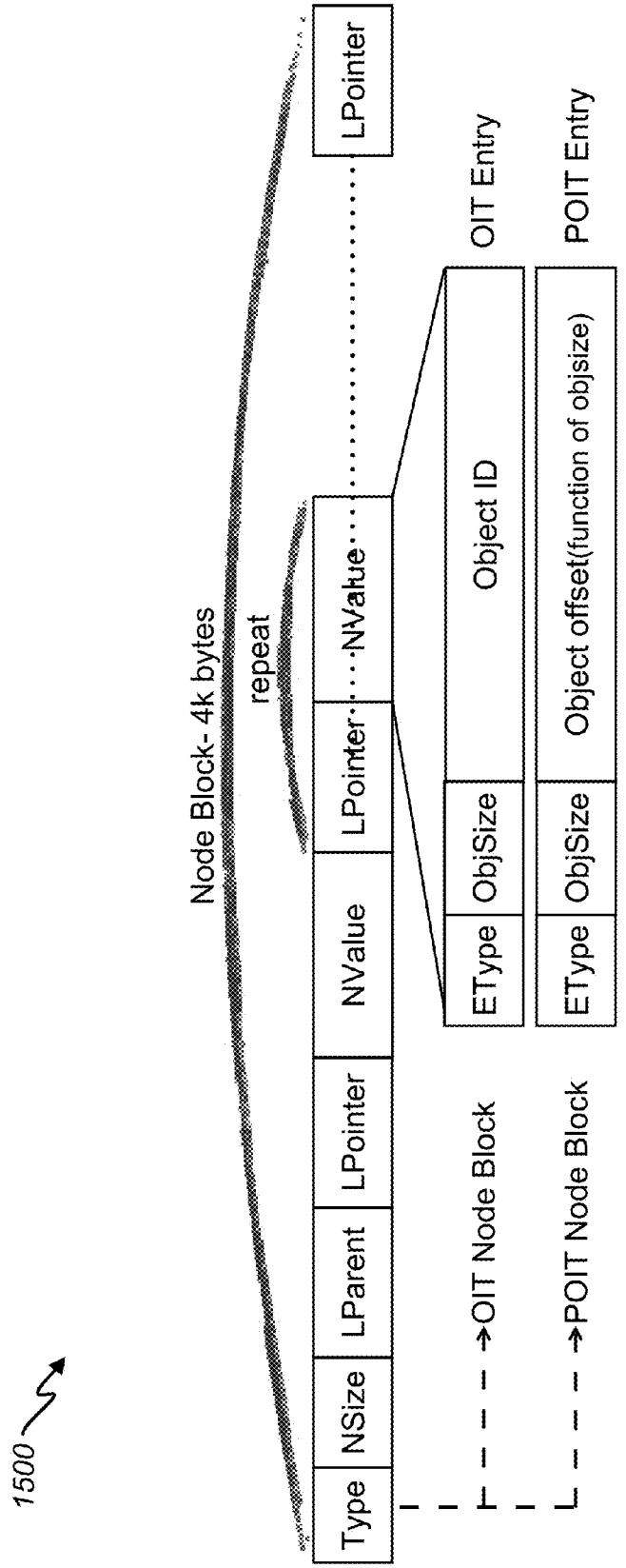

1500
Node Block- 4k bytes
repeat
Type
NSize
LParent
LPointer
NValue
LPointer
NValue
LPointer
OIT Node Block
POIT Node Block
EType
ObjSize
Object ID
OIT Entry
EType
ObjSize
Object offset(function of objsize)
POIT Entry

FIG. 15A

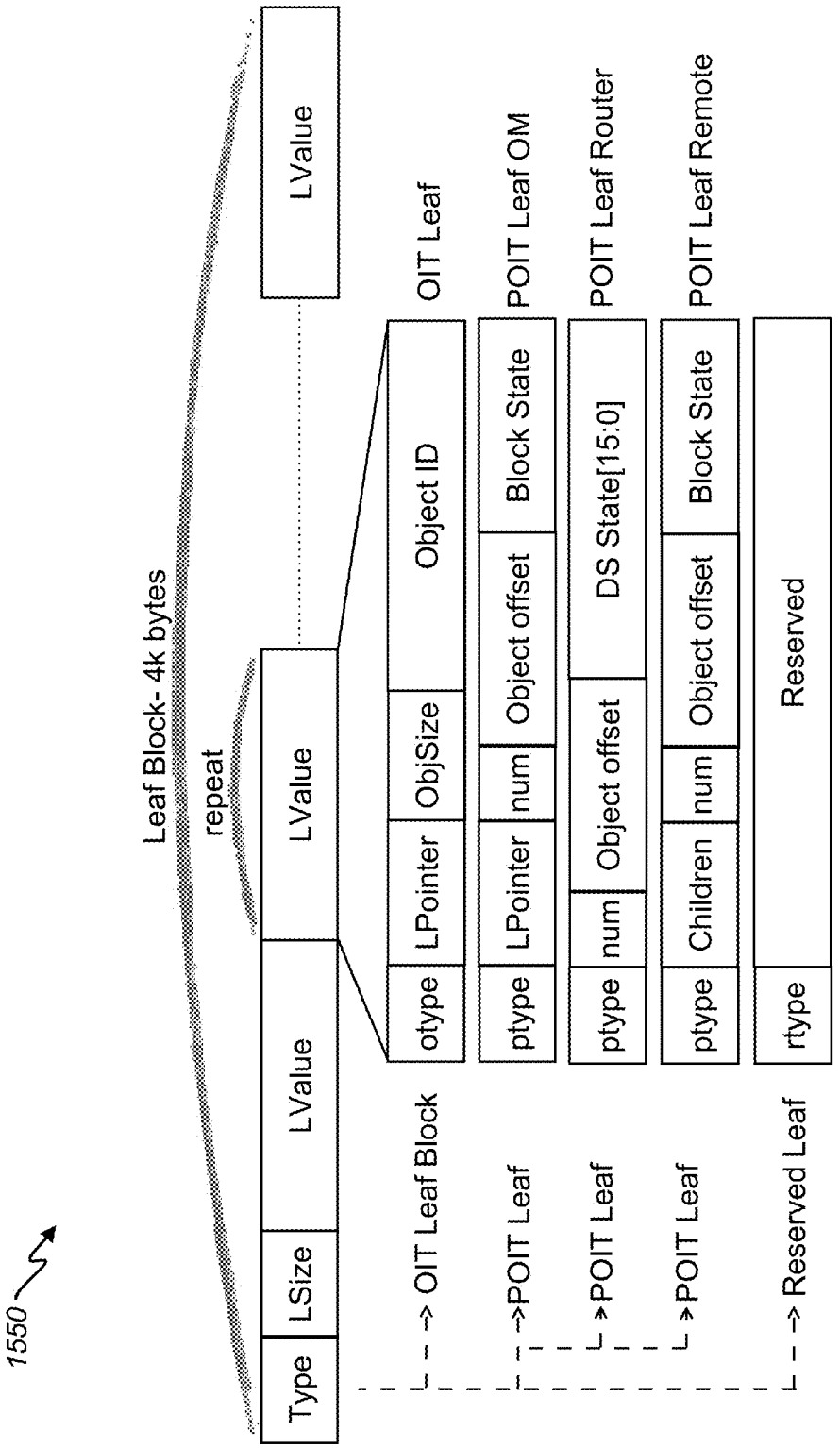

1550
Leaf Block- 4k bytes
Type
LSize
LValue
repeat
LValue
LValue
--> OIT Leaf Block
otype
LPointer
ObjSize
Object ID
OIT Leaf
-->POIT Leaf
ptype
LPointer
num
Object offset
Block State
POIT Leaf OM
-->POIT Leaf
ptype
num
Object offset
DS State[15:0]
POIT Leaf Router
-->POIT Leaf
ptype
Children
num
Object offset
Block State
POIT Leaf Remote
--> Reserved Leaf
rtype
Reserved

FIG. 15B

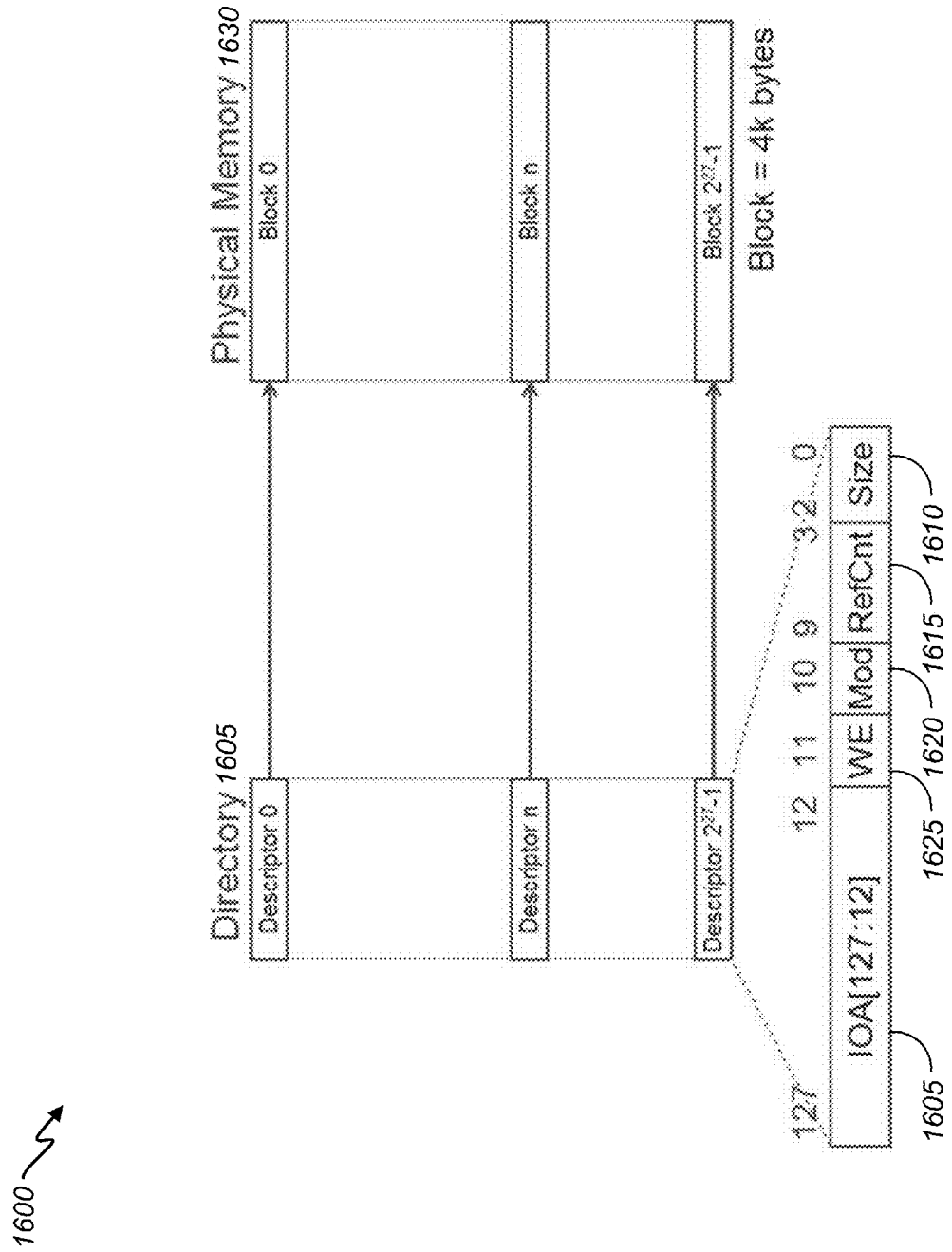

1600
Physical Memory 1630
Block 0
Block n
Block 2^27-1
Block = 4k bytes
Directory 1605
Descriptor 0
Descriptor n
Descriptor 2^27-1
127
12 11 10 9
32 0
IOA[127:12]
WE
Mod
RefCnt
Size
1605
1625
1620
1615
1610

FIG. 16

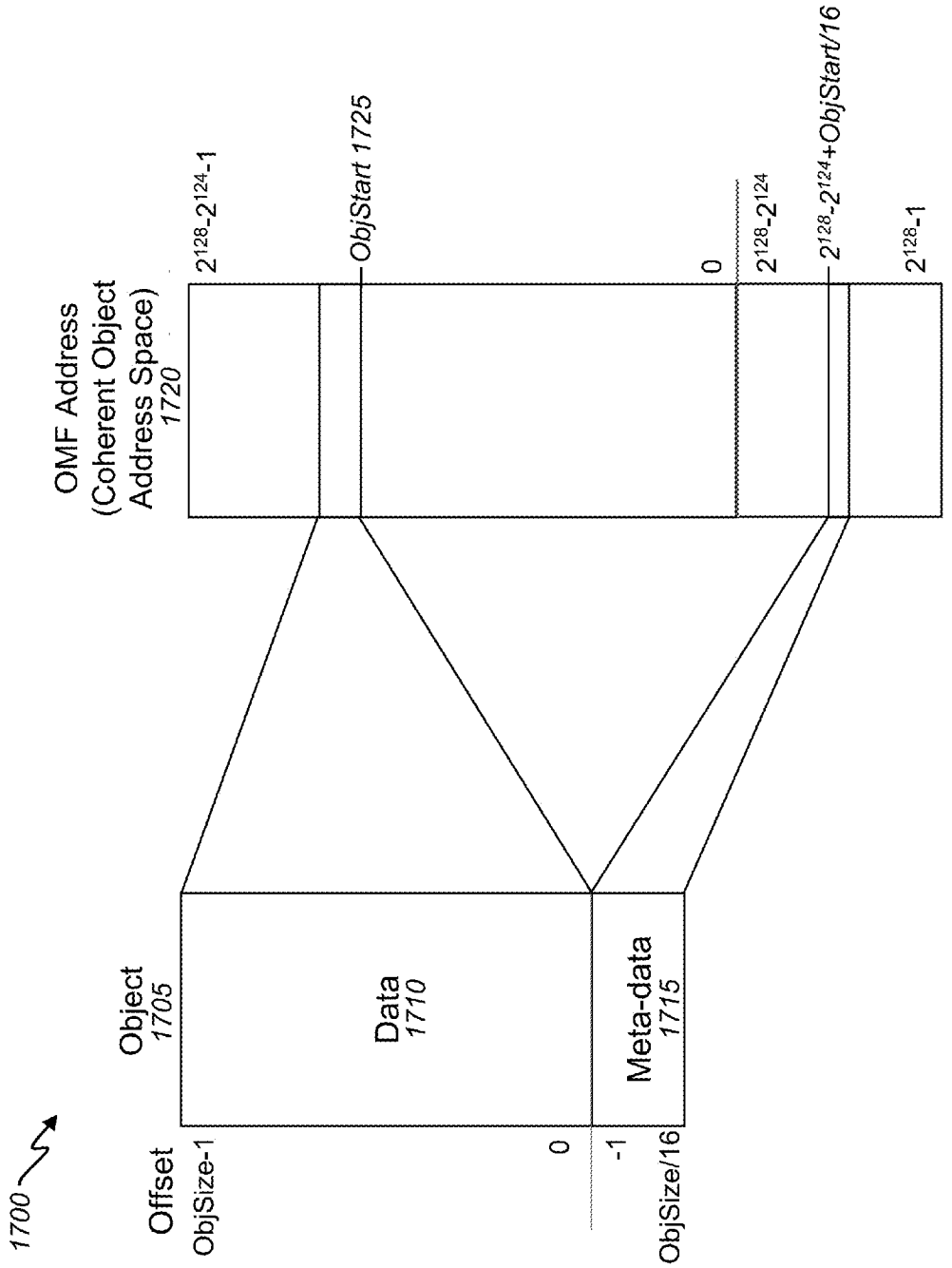

1700
Offset
ObjSize-1
0
-1
ObjSize/16
Object
1705
Data
1710
Meta-data
1715
OMF Address
(Coherent Object
Address Space)
1720
2128-2124-1
ObjStart 1725
0
2128-2124
2128-2124+ObjStart/16
2128-1

FIG. 17A

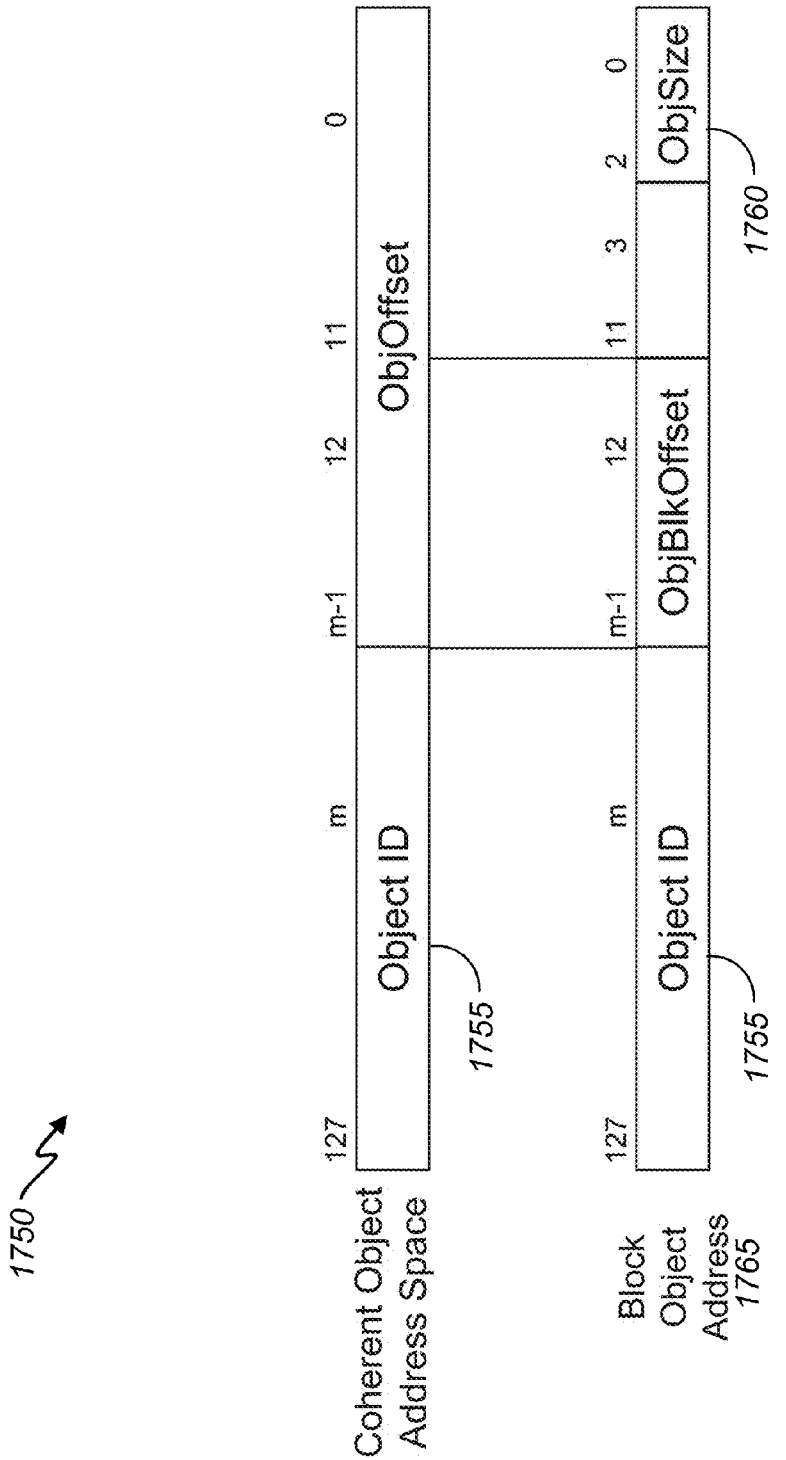

1750
Coherent Object Address Space
127
m
m-1
12
11
0
Object ID
ObjOffset
1755
Block Object Address 1765
127
m
m-1
12
11
3
2
0
Object ID
ObjBlkOffset
ObjSize
1760
1755

FIG. 17B

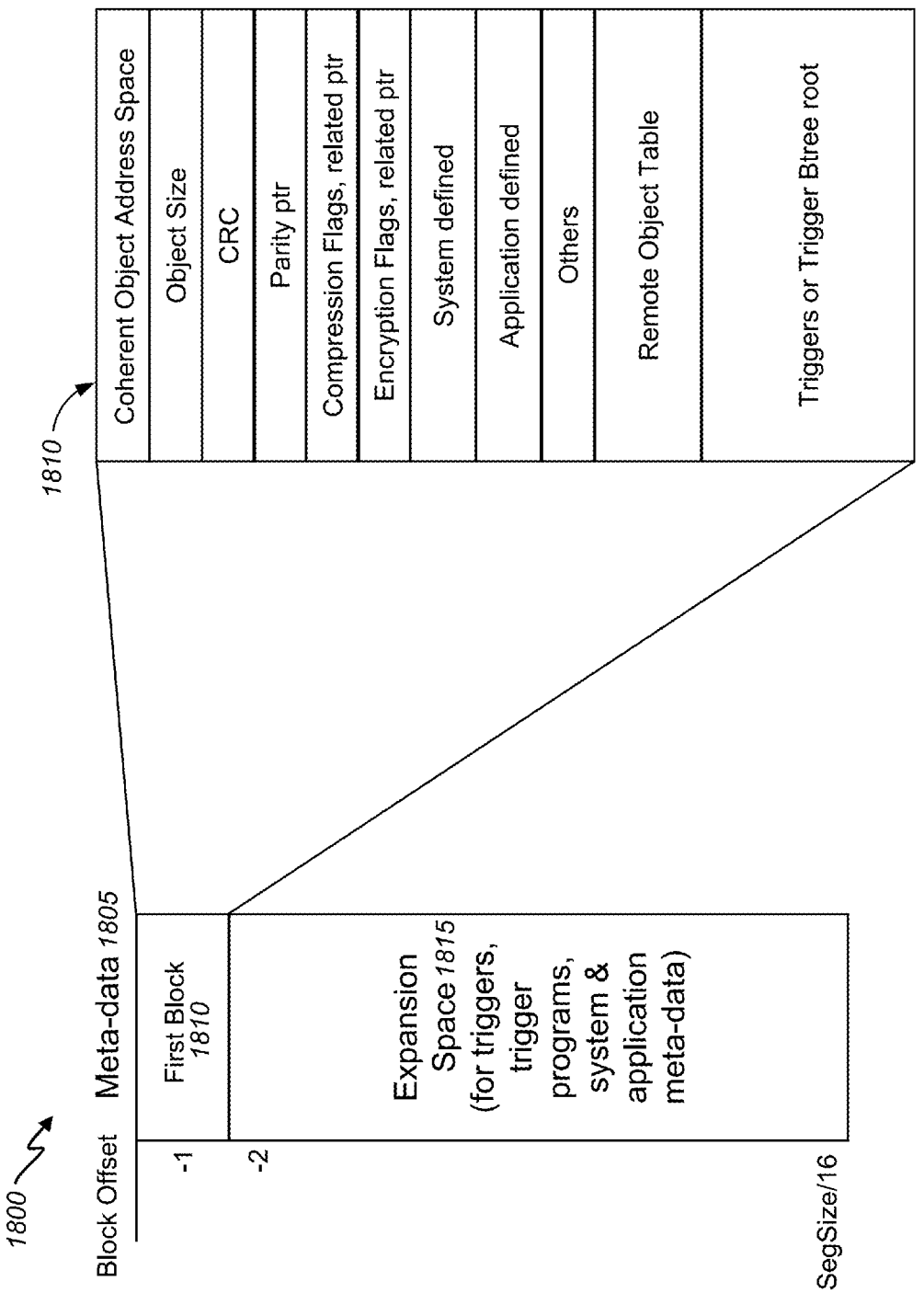

1800
Block Offset
Meta-data 1805
First Block 1810
-1
-2
Expansion Space 1815 (for triggers, trigger programs, system & application meta-data)
SegSize/16
1810
Coherent Object Address Space
Object Size
CRC
Parity ptr
Compression Flags, related ptr
Encryption Flags, related ptr
System defined
Application defined
Others
Remote Object Table
Triggers or Trigger Btree root

FIG. 18

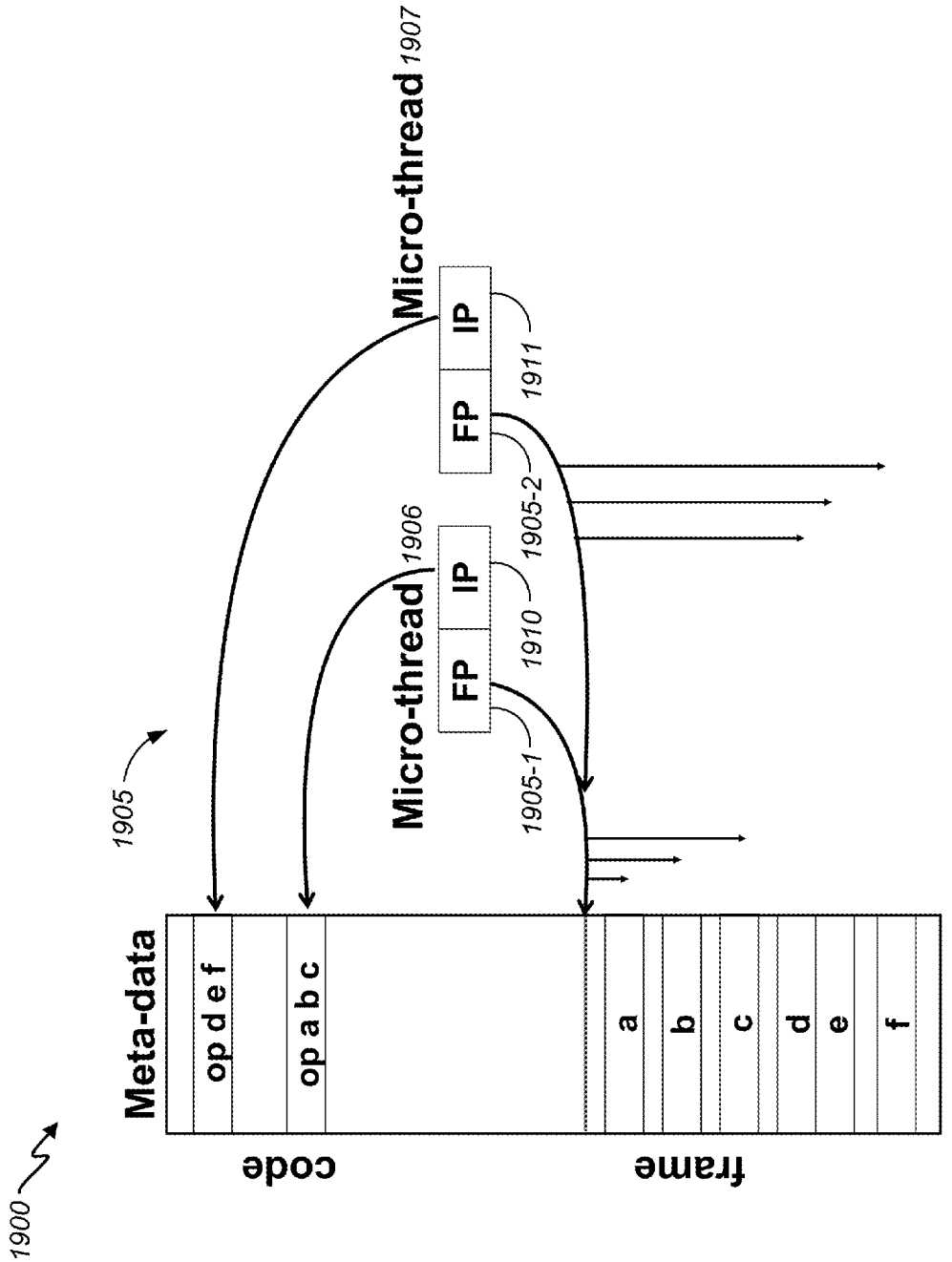

1900
1905
Meta-data
op d e f
op a b c
code
frame
a
b
c
d
e
f
Micro-thread 1907
FP
IP
1911
1905-2
Micro-thread 1906
FP
IP
1910
1905-1

FIG. 19

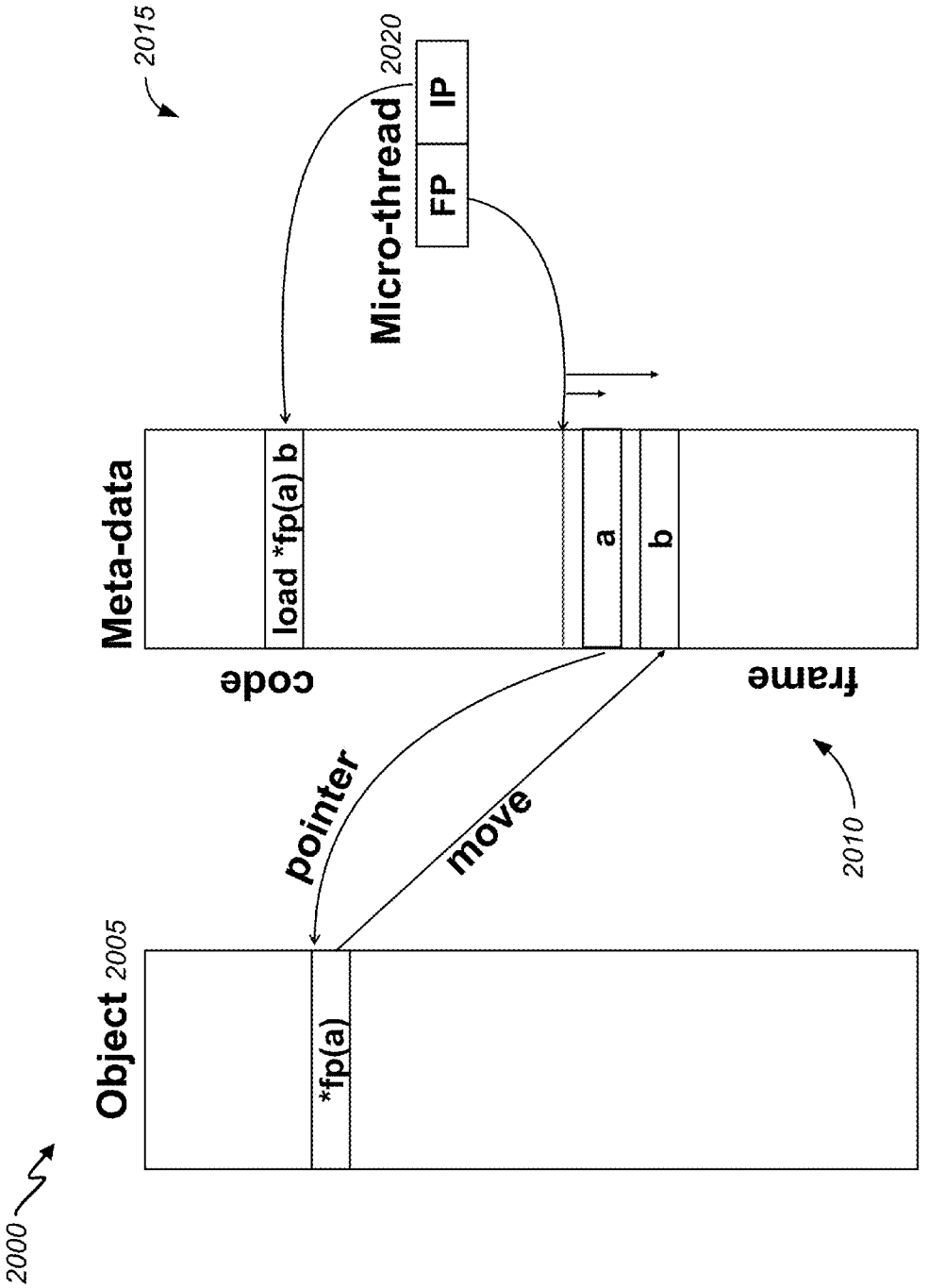

2015
2020
Micro-thread
IP
FP
Meta-data
load *fp(a) b
a
b
code
frame
pointer
move
2010
Object 2005
*fp(a)
2000

FIG. 20

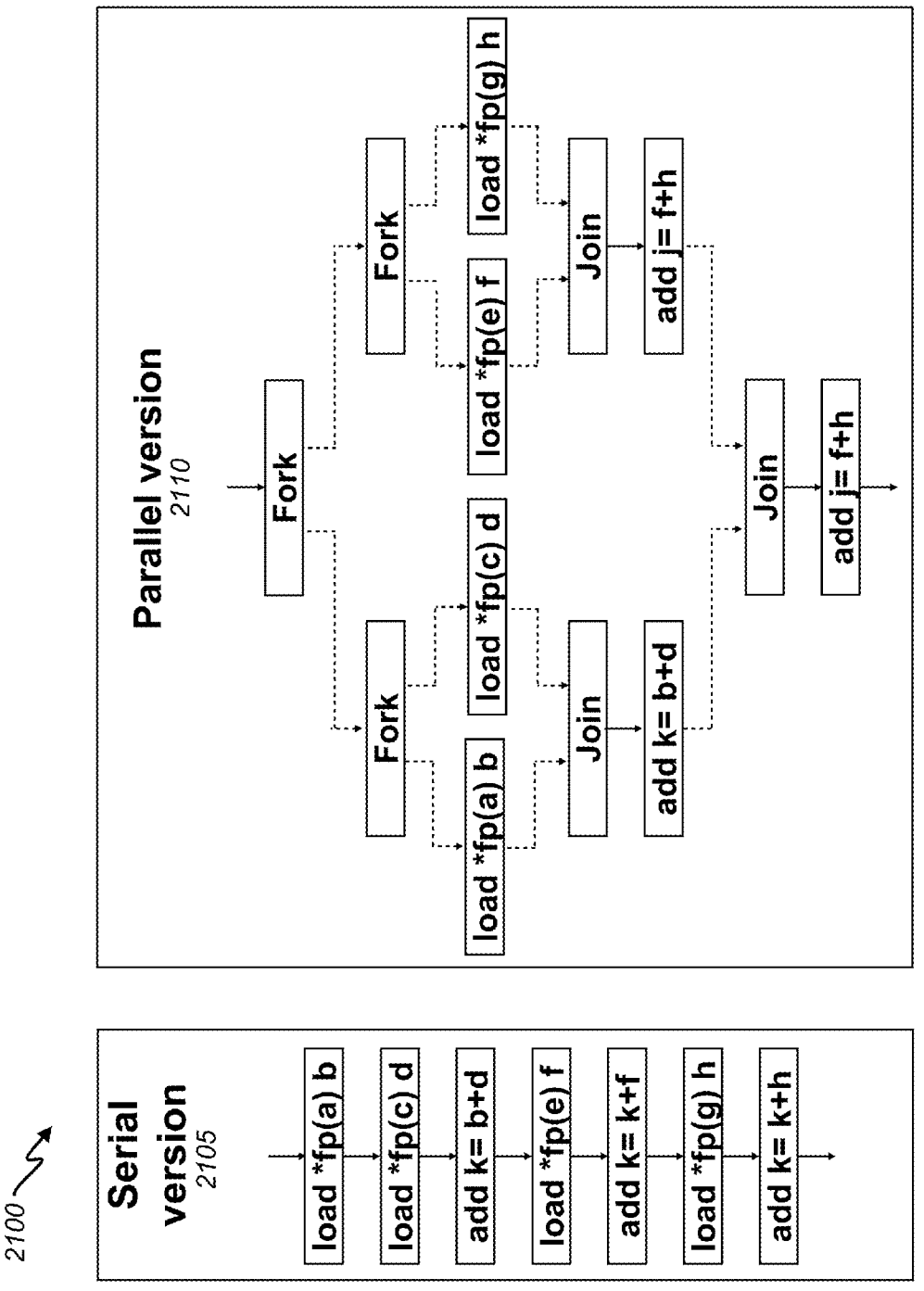

2100
Serial version
2105
load *fp(a) b
load *fp(c) d
add k= b+d
load *fp(e) f
add k= k+f
load *fp(g) h
add k= k+h
Parallel version
2110
Fork
Fork
Fork
load *fp(a) b
load *fp(c) d
load *fp(e) f
load *fp(g) h
Join
Join
Join
add k= b+d
add j= f+h
add j= f+h

FIG. 21

DISTRIBUTED INDEX FOR FAULT TOLERANT OBJECT MEMORY FABRIC

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims benefit under 35 USC 119(e) of U.S. Provisional Application No. 62/105,602, filed on Jan. 20, 2015, by Frank et al. and entitled "Infinite Memory Fabric Data Types and Metadata Architecture," of which the entire disclosure is incorporated herein by reference for all purposes.

[0002] The present application is also related to the following co-pending and commonly assigned U.S. Patent Applications:

[0003] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967319 (000100US)) filed concurrent herewith by Frank and entitled "Object Based Memory Fabric;"

[0004] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967320 (000110US)) filed concurrent herewith by Frank and entitled "Trans-Cloud Object Based Memory;"

[0005] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967321 (000120US)) filed concurrent herewith by Frank and entitled "Universal Single Level Object Memory Address Space;"

[0006] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967322 (000130US)) filed concurrent herewith by Frank and entitled "Object Memory Fabric Performance Acceleration;"

[0007] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967324 (000210US)) filed concurrent herewith by Frank and entitled "Implementation of an Object Memory Centric Cloud;"

[0008] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967325 (000220US)) filed concurrent herewith by Frank and entitled "Managing Meta-Data in an Object Memory Fabric;"

[0009] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967326 (000230US)) filed concurrent herewith by Frank and entitled "Utilization of a Distributed Index to Provide Object Memory Fabric Coherency;"

[0010] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967327 (000300US)) filed concurrent herewith by Frank and entitled "Object Memory Data Flow Instruction Execution;"

[0011] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967329 (000310US)) filed concurrent herewith by Frank and entitled "Object Memory Data Flow Triggers;" and

[0012] U.S. patent application Ser. No. _____ (Attorney Docket Number 097704-0967328 (000320US)) filed concurrent herewith by Frank and entitled "Object Memory Instruction Set," of which the entire disclosure of each is incorporated herein by reference for all purposes.

BACKGROUND OF THE INVENTION

[0013] Embodiments of the present invention relate generally to methods and systems for improving performance of processing nodes in a fabric and more particularly to changing the way in which processing, memory, storage, network, and cloud computing, are managed to significantly improve the efficiency and performance of commodity hardware.

[0014] As the size and complexity of data and the processes performed thereon continually increases, computer hardware is challenged to meet these demands. Current commodity hardware and software solutions from established server, network and storage providers are unable to meet the demands of Cloud Computing and Big Data environments. This is due, at least in part, to the way in which processing, memory, and storage are managed by those systems. Specifically, processing is separated from memory which is turn is separated from storage in current systems and each of processing, memory, and storage is managed separately by software. Each server and other computing device (referred to herein as a node) is in turn separated from other nodes by a physical computer network, managed separately by software and in turn the separate processing, memory, and storage associated with each node are managed by software on that node.

[0015] FIG. **1** is a block diagram illustrating an example of the separation data storage, memory, and processing within prior art commodity servers and network components. This example illustrates a system **100** in which commodity servers **105** and **110** are communicatively coupled with each other via a physical network **115** and network software **155** as known in the art. Also as known in the art, the servers can each execute any number of one or more applications **120***a*, **120***b*, **120***c* of any variety. As known in the art, each application **120***a*, **120***b*, **120***c* executes on a processor (not shown) and memory (not shown) of the server **105** and **110** using data stored in physical storage **150**. Each server **105** and **110** maintains a directory **125** mapping the location of the data used by the applications **120***a*, **120***b*, **120***c*. Additionally, each server implements for each executing application **120***a*, **120***b*, **120***c* a software stack which includes an application representation **130** of the data, a database representation **135**, a file system representation **140**, and a storage representation **145**.

[0016] While effective, there are three reasons that such implementations on current commodity hardware and software solutions from established server, network and storage providers are unable to meet the increasing demands of Cloud Computing and Big Data environments. One reason for the shortcomings of these implementations is their complexity. The software stack must be in place and every application must manage the separation of storage, memory, and processing as well as applying parallel server resources. Each application must trade-off algorithm parallelism, data organization and data movement which is extremely challenging to get correct, let alone considerations of performance and economics. This tends to lead to implementation of more batch oriented solutions in the applications, rather than the integrated real-time solutions preferred by most businesses. Additionally, separation of storage, memory, and processing, in such implementations also creates significant inefficiency for each layer of the software stack to find, move, and access a block of data due to the required instruction execution and latencies of each layer of the software stack and between the layers. Furthermore, this inefficiency limits the economic scaling possible and limits the data-size for all but the most extremely parallel algorithms. The reason for the latter is that the efficiency with which servers (processors or threads) can interact limits the amount of parallelism due to Amdahl's law. Hence, there is a need for improved methods and systems for managing processing, memory, and storage to significantly improve the performance of processing nodes.

BRIEF SUMMARY OF THE INVENTION

[0017] Embodiments of the invention provide systems and methods for managing processing, memory, and storage to significantly improve the performance of processing nodes. Certain embodiments may include an object memory fabric distributed object memory and index. With the distributed index, individual nodes may index local objects and blocks of objects on a per-object basis. Certain embodiments of object memory fabric distributed object memory and index may be based at least in part on overlaying an independent local index function on top of a fat-tree hierarchical network.

[0018] In one aspect, an object memory fabric is disclosed. The object memory fabric may include a plurality of object memory modules, each object memory module including object storage storing one or more memory objects, memory object meta-data, and a memory module object directory. Each memory object and/or portion of memory objects may be created natively within the object memory module and may be a managed by the object memory module at a memory layer. The memory module object directory may index all memory objects and/or portions of memory objects within the object memory module. The object memory fabric may include a hierarchy of object routers communicatively coupling the plurality of object memory modules. Each object router of the hierarchy of object routers may include a router object directory. The router object directory may index all memory objects and/or portions of memory objects contained in object memory modules below the object router along a line of descent in the hierarchy stemming from the object router. The hierarchy of object routers, based at least in part on the router object directories, may be adapted to behave in aggregate as a single object directory communicatively coupled to all object memory modules and to process requests based at least in part on the router object directories.

[0019] The hierarchy of object routers may operate according to a hierarchical tree network. The object memory modules below the object router along the line of descent in the hierarchy stemming from the object router may include object memory modules directly communicatively coupled to the respective object router toward a leaf of the hierarchical tree network. Toward the leaf of the hierarchical tree network may correspond to a most direct path between the object router away from a root of the hierarchical tree network and an object memory module at the leaf of the hierarchical tree network.

[0020] The behaving in aggregate as the single object directory communicatively coupled to all the object memory modules and the processing the requests may include: responsive to each request of the requests, at least one of the object routers looking up an object corresponding to the request with a respective router object directory, and: consequent to identifying a reference to the object in the respective router object directory, forwarding the object toward a leaf in the hierarchy; consequent to identifying the reference to the object is not in the respective router object directory, forwarding the first request toward a root in the hierarchy.

[0021] At least one of the requests may be received from an application layer. Each object memory module may further include an object index tree that indexes local memory objects. The object index tree may include node blocks and leaf blocks each differentiated by a type attribute, where one or more of the leaf blocks point to locations of the local memory objects in persistent memory and/or faster memory. The object index tree may include a pointer to a per object index tree for each local memory object, where the per object index tree for each local memory object indexes, on a block basis, memory object data and memory object meta-data locally present for the local memory object. The per object index tree may include node blocks and leaf blocks each differentiated by a type attribute, where one or more of the leaf blocks point to locations of the memory object data and memory object meta-data in persistent memory and/or faster memory. The hierarchy of object routers adapted to behave in aggregate as the single object directory may be used to manage at least one memory object across multiple object memory modules of the plurality of object memory modules where a single object memory module of the multiple object memory modules does not have storage space required to store all blocks of the at least one memory object.

[0022] In another aspect, a hardware-based processing node is disclosed. The hardware-based processing node may include an object memory module including object storage storing one or more memory objects, memory object meta-data, and a memory module object directory. Each memory object and/or portion of memory objects may be created natively within the object memory module and may be a managed by the object memory module at a memory layer. The memory module object directory may index all memory objects and/or portions of memory objects within the object memory module. The object memory module may process one or more requests based at least in part on the one or more object directories.

[0023] The processing the one or more requests may include: processing an object identifier corresponding to a first request of the one or more requests; determining whether at least one object of the one or more memory objects corresponds to the object identifier; and consequent to a determination that at least one of the one or more memory objects does correspond to the object identifier, generating a response to the first request based at least in part on the at least one object. An object router may be communicatively coupled to the object memory module that, consequent to a determination that at least one of the one or more memory objects does not correspond to the object identifier, routes the first request to an additional node. The routing the first request to the additional node may be based at least in part on the object router determining a location of a memory object corresponding to the object identifier. The routing the first request to the additional node may be based at least in part on the object router determining that a location of a memory object corresponding to the object identifier is unknown. The first request may be directed toward a root node. After the routing the first request to the additional node, the memory module may generate a response to the first request based at least in part on a received response from the additional node. The object memory module may further include an object index tree that indexes local memory objects. The object index tree may include node blocks and leaf blocks each differentiated by a type attribute, where one or more of the leaf blocks point to locations of the local memory objects in persistent memory and/or faster memory. The object index tree may include a pointer to a per object index tree for each local memory object, where the per object index tree for each local memory object indexes, on a block basis, memory object data and memory object meta-data locally present for the local memory object. The per object index tree may include node blocks and leaf blocks each differentiated by a type attribute, where one or more of the leaf blocks point to locations of the

memory object data and memory object meta-data in persistent memory and/or faster memory. At least one of the one or more requests may be received from an application layer. At least one of the one or more requests may be received from one or more additional nodes. The node may be configured to utilize a set of algorithms to operatively couple to one or more additional nodes to operate as a set of nodes independently of a scale of the set of nodes, where the set of nodes operates so that all memory objects of the set of nodes are accessible by any node of the set of nodes.

[0024] In yet another aspect, a method for storing and managing one or more memory objects in an object memory fabric is disclosed. The method may include any one or combination of the following. One or more memory objects, memory object meta-data, and a memory module object directory may be stored in object storage of an object memory module. Each memory object and/or portion of memory objects may be created natively within the object memory module and may be managed by the object memory module at a memory layer. The memory module object directory may index all memory objects and/or portions of memory objects within the object memory module. The object memory module may process one or more requests based at least in part on the one or more object directories.

[0025] The processing the one or more requests may include: processing an object identifier corresponding to a first request of the one or more requests; determining whether at least one object of the one or more memory objects corresponds to the object identifier; and consequent to a determination that at least one of the one or more memory objects does correspond to the object identifier, generating a response to the first request based at least in part on the at least one object. The processing the one or more requests may include, consequent to a determination that at least one of the one or more memory objects does not correspond to the object identifier, routing, by an object router communicatively coupled to the object memory module, the first request to an additional node.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] FIG. **1** is a block diagram illustrating an example of the separation data storage, memory, and processing within prior art commodity servers and network components.

[0027] FIG. **2** is a block diagram illustrating components of an exemplary distributed system in which various embodiments of the present invention may be implemented.

[0028] FIG. **3** is a block diagram illustrating an exemplary computer system in which embodiments of the present invention may be implemented.

[0029] FIG. **4** is a block diagram illustrating an exemplary Infinite Object memory fabric (object memory fabric) architecture according to one embodiment of the present invention.

[0030] FIG. **5** is a block diagram illustrating an exemplary object memory fabric object memory according to one embodiment of the present invention.

[0031] FIG. **6** is a block diagram illustrating an exemplary object memory dynamics and physical organization according to one embodiment of the present invention.

[0032] FIG. **7** is a block diagram illustrating aspects of object memory fabric hierarchy of object memory, which localizes working sets and allows for virtually unlimited scalability, according to one embodiment of the present invention.

[0033] FIG. **8** is a block diagram illustrating aspects of an example relationship between object address space, virtual address, and physical address, according to one embodiment of the present invention.

[0034] FIG. **9** is a block diagram illustrating aspects of an example relationship between object sizes and object address space pointers, according to one embodiment of the present invention.

[0035] FIG. **10** is a block diagram illustrating aspects of an example object memory fabric distributed object memory and index structure, according to one embodiment of the present invention.

[0036] FIG. **11** illustrates aspects of an object memory hit case that executes completely within the object memory, according to one embodiment of the present invention.

[0037] FIG. **12** illustrates aspects of an object memory miss case and the distributed nature of the object memory and object index, according to one embodiment of the present invention.

[0038] FIG. **13** is a block diagram illustrating aspects of an example of leaf level object memory in view of the object memory fabric distributed object memory and index structure, according to one embodiment of the present invention.

[0039] FIG. **14** is a block diagram illustrating aspects of an example of object memory fabric router object index structure, according to one embodiment of the present invention.

[0040] FIGS. **15**A and **15**B are block diagrams illustrating aspects of example index tree structures, including node index tree structure and leaf index tree, according to one embodiment of the present invention.

[0041] FIG. **16** is a block diagram illustrating aspects of an example physical memory organization, according to one embodiment of the present invention.

[0042] FIG. **17**A is a block diagram illustrating aspects of example object addressing, according to one embodiment of the present invention.

[0043] FIG. **17**B is a block diagram illustrating aspects of example object memory fabric pointer and block addressing, according to one embodiment of the present invention.

[0044] FIG. **18** is a block diagram illustrating aspects of example object metadata, according to one embodiment of the present invention.

[0045] FIG. **19** is a block diagram illustrating aspects of an example micro-thread model, according to one embodiment of the present invention.

[0046] FIG. **20** is a block diagram illustrating aspects of an example relationship of code, frame, and object, according to one embodiment of the present invention.

[0047] FIG. **21** is a block diagram illustrating aspects of an example of micro-thread concurrency, according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0048] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of various embodiments of the present invention. It will be apparent, however, to one skilled in the art that embodiments of the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

[0049] The ensuing description provides exemplary embodiments only, and is not intended to limit the scope, applicability, or configuration of the disclosure. Rather, the

ensuing description of the exemplary embodiments will provide those skilled in the art with an enabling description for implementing an exemplary embodiment. It should be understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope of the invention as set forth in the appended claims.

[0050] Specific details are given in the following description to provide a thorough understanding of the embodiments. However, it will be understood by one of ordinary skill in the art that the embodiments may be practiced without these specific details. For example, circuits, systems, networks, processes, and other components may be shown as components in block diagram form in order not to obscure the embodiments in unnecessary detail. In other instances, well-known circuits, processes, algorithms, structures, and techniques may be shown without unnecessary detail in order to avoid obscuring the embodiments.

[0051] Also, it is noted that individual embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a data flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed, but could have additional steps not included in a figure. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a function, its termination can correspond to a return of the function to the calling function or the main function.

[0052] The term "machine-readable medium" includes, but is not limited to portable or fixed storage devices, optical storage devices, wireless channels and various other mediums capable of storing, containing or carrying instruction(s) and/or data. A code segment or machine-executable instructions may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc. Various other terms used herein are now defined for the sake of clarity.

[0053] Virtual memory is a memory management technique that gives the illusion to each software process that memory is as large as the virtual address space. The operating system in conjunction with differing degrees of hardware manages the physical memory as a cache of the virtual address space, which is placed in secondary storage and accessible through Input/Output instructions. Virtual memory is separate from, but can interact with, a file system.

[0054] A single level store is an extension of virtual memory in which there are no files, only persistent objects or segments which are mapped into a processes' address space using virtual memory techniques. The entire storage of the computing system is thought of as a segment and address within a segment. Thus at least three separate address spaces, i.e., physical memory address/node, virtual address/process, and secondary storage address/disk, are managed by software.

[0055] Object storage refers to the way units of storage called objects are organized. Every object consists of a container that holds three things: actual data; expandable metadata; and a globally unique identifier referred to herein as the object address. The metadata of the object is used to define contextual information about the data and how it should be used and managed including relationship to other objects.

[0056] The object address space is managed by software over storage devices, nodes, and network to find an object without knowing its physical location. Object storage is separate from virtual memory and single level store, but can certainly inter-operate through software.

[0057] Block storage consists of evenly sized blocks of data with an address based on a physical location and without metadata.

[0058] A network address is a physical address of a node within an IP network that is associated with a physical location.

[0059] A node or processing node is a physical unit of computing delineated by a shared physical memory that be addressed by any processor within the node.

[0060] Object memory is an object store directly accessible as memory by processor memory reference instructions and without implicit or explicit software or Input/Output instructions required. Object capabilities are directly provided within the object memory to processing through memory reference instructions.

[0061] An object memory fabric connects object memory modules and nodes into a single object memory where any object is local to any object memory module by direct management, in hardware, of object data, meta-data and object address.

[0062] An object router routes objects or portions of objects in an object memory fabric based on an object address. This is distinct from a conventional router which forwards data packets to appropriate part of a network based on a network address.

[0063] Embodiments may be implemented by hardware, software, firmware, middleware, microcode, hardware description languages, or any combination thereof. When implemented in software, firmware, middleware or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium. A processor(s) may perform the necessary tasks.

[0064] Embodiments of the invention provide systems and methods for managing processing, memory, storage, network, and cloud computing to significantly improve the efficiency and performance of processing nodes. Embodiments described herein can be implemented in a set of hardware components that, in essence, change the way in which processing, memory, and storage, network, and cloud computing are managed by breaking down the artificial distinctions between processing, memory, storage and networking in today's commodity solutions to significantly improve the efficiency and performance of commodity hardware. For example, the hardware elements can include a standard format memory module, such as a (DIMM) and a set of one or more object routers. The memory module can be added to commodity or "off-the-shelf" hardware such a server node and acts as a big data accelerator within that node. Object routers can be used to interconnect two or more servers or other nodes adapted with the memory modules and help to manage processing, memory, and storage across these different servers. Nodes can be physically close or far apart.

Together, these hardware components can be used with commodity servers or other types of computing nodes in any combination to implement the embodiments described herein.

[0065] According to one embodiment, such hardware components can implement an object-based memory which manages the objects within the memory and at the memory layer rather than in the application layer. That is, the objects and associated properties are implemented and managed natively in memory enabling the object memory system to provide increased functionality without any software and increasing performance by dynamically managing object characteristics including, but not limited to persistence, location and processing. Object properties can also propagate up to higher application levels.

[0066] Such hardware components can also eliminate the distinction between memory (temporary) and storage (persistent) by implementing and managing both within the objects. These components can eliminate the distinction between local and remote memory by transparently managing the location of objects (or portions of objects) so all objects appear simultaneously local to all nodes. These components can also eliminate the distinction between processing and memory through methods of the objects to place the processing within the memory itself.

[0067] According to one embodiment, such hardware components can eliminate typical size constraints on memory space of the commodity servers imposed by address sizes. Rather, physical addressing can be managed within the memory objects themselves and the objects can in turn be accessed and managed through the object name space.

[0068] Embodiment described herein can provide transparent and dynamic performance acceleration, especially with big data or other memory intensive applications by reducing or eliminating overhead typically associated with memory management, storage management, networking and data directories. Rather, management of the memory objects at the memory level can significantly shorten the pathways between storage and memory and between memory and processing, thereby eliminating the associated overhead between each. Various additional details of embodiments of the present invention will be described below with reference to the figures.

[0069] FIG. **2** is a block diagram illustrating components of an exemplary distributed system in which various embodiments of the present invention may be implemented. In the illustrated embodiment, distributed system **200** includes one or more client computing devices **202**, **204**, **206**, and **208**, which are configured to execute and operate a client application such as a web browser, proprietary client, or the like over one or more network(s) **210**. Server **212** may be communicatively coupled with remote client computing devices **202**, **204**, **206**, and **208** via network **210**.

[0070] In various embodiments, server **212** may be adapted to run one or more services or software applications provided by one or more of the components of the system. In some embodiments, these services may be offered as web-based or cloud services or under a Software as a Service (SaaS) model to the users of client computing devices **202**, **204**, **206**, and/or **208**. Users operating client computing devices **202**, **204**, **206**, and/or **208** may in turn utilize one or more client applications to interact with server **212** to utilize the services provided by these components. For the sake of clarity, it should be noted that server **212** and database **214**, **216** can correspond to server **105** described above with reference to FIG. **1**. Network **210** can be part of or an extension to physical network **115**. It should also be understood that there can be any number of client computing devices **202**, **204**, **206**, **208** and servers **212**, each with one or more databases **214**, **216**.

[0071] In the configuration depicted in the figure, the software components **218**, **220** and **222** of system **200** are shown as being implemented on server **212**. In other embodiments, one or more of the components of system **200** and/or the services provided by these components may also be implemented by one or more of the client computing devices **202**, **204**, **206**, and/or **208**. Users operating the client computing devices may then utilize one or more client applications to use the services provided by these components. These components may be implemented in hardware, firmware, software, or combinations thereof. It should be appreciated that various different system configurations are possible, which may be different from distributed system **200**. The embodiment shown in the figure is thus one example of a distributed system for implementing an embodiment system and is not intended to be limiting.

[0072] Client computing devices **202**, **204**, **206**, and/or **208** may be portable handheld devices (e.g., an iPhone®, cellular telephone, an iPad®, computing tablet, a personal digital assistant (PDA)) or wearable devices (e.g., a Google Glass® head mounted display), running software such as Microsoft Windows Mobile®, and/or a variety of mobile operating systems such as iOS, Windows Phone, Android, BlackBerry 10, Palm OS, and the like, and being Internet, e-mail, short message service (SMS), Blackberry®, or other communication protocol enabled. The client computing devices can be general purpose personal computers including, by way of example, personal computers and/or laptop computers running various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems. The client computing devices can be workstation computers running any of a variety of commercially-available UNIX® or UNIX-like operating systems, including without limitation the variety of GNU/Linux operating systems, such as for example, Google Chrome OS. Alternatively, or in addition, client computing devices **202**, **204**, **206**, and **208** may be any other electronic device, such as a thin-client computer, an Internet-enabled gaming system (e.g., a Microsoft Xbox gaming console with or without a Kinect® gesture input device), and/or a personal messaging device, capable of communicating over network(s) **210**.

[0073] Although exemplary distributed system **200** is shown with four client computing devices, any number of client computing devices may be supported. Other devices, such as devices with sensors, etc., may interact with server **212**.

[0074] Network(s) **210** in distributed system **200** may be any type of network familiar to those skilled in the art that can support data communications using any of a variety of commercially-available protocols, including without limitation TCP/IP (Transmission Control Protocol/Internet Protocol), SNA (Systems Network Architecture), IPX (Internet Packet Exchange), AppleTalk, and the like. Merely by way of example, network(s) **210** can be a Local Area Network (LAN), such as one based on Ethernet, Token-Ring and/or the like. Network(s) **210** can be a wide-area network and the Internet. It can include a virtual network, including without limitation a Virtual Private Network (VPN), an intranet, an extranet, a Public Switched Telephone Network (PSTN), an

infra-red network, a wireless network (e.g., a network operating under any of the Institute of Electrical and Electronics (IEEE) 802.11 suite of protocols, Bluetooth®, and/or any other wireless protocol); and/or any combination of these and/or other networks. Elements of such networks can have an arbitrary distance, i.e., can be remote or co-located. Software Defined Networks (SDNs) can be implemented with a combination of dumb routers and software running on servers.

[0075] Server **212** may be composed of one or more general purpose computers, specialized server computers (including, by way of example, Personal Computer (PC) servers, UNIX® servers, mid-range servers, mainframe computers, rack-mounted servers, etc.), server farms, server clusters, or any other appropriate arrangement and/or combination. In various embodiments, server **212** may be adapted to run one or more services or software applications described in the foregoing disclosure. For example, server **212** may correspond to a server for performing processing described above according to an embodiment of the present disclosure.

[0076] Server **212** may run an operating system including any of those discussed above, as well as any commercially available server operating system. Server **212** may also run any of a variety of additional server applications and/or mid-tier applications, including HyperText Transport Protocol (HTTP) servers, File Transfer Protocol (FTP) servers, Common Gateway Interface (CGI) servers, JAVA® servers, database servers, and the like. Exemplary database servers include without limitation those commercially available from Oracle, Microsoft, Sybase, International Business Machines (IBM), and the like.

[0077] In some implementations, server **212** may include one or more applications to analyze and consolidate data feeds and/or event updates received from users of client computing devices **202**, **204**, **206**, and **208**. As an example, data feeds and/or event updates may include, but are not limited to, Twitter® feeds, Facebook® updates or real-time updates received from one or more third party information sources and continuous data streams, which may include real-time events related to sensor data applications, financial tickers, network performance measuring tools (e.g., network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like. Server **212** may also include one or more applications to display the data feeds and/or real-time events via one or more display devices of client computing devices **202**, **204**, **206**, and **208**.

[0078] Distributed system **200** may also include one or more databases **214** and **216**. Databases **214** and **216** may reside in a variety of locations. By way of example, one or more of databases **214** and **216** may reside on a non-transitory storage medium local to (and/or resident in) server **212**. Alternatively, databases **214** and **216** may be remote from server **212** and in communication with server **212** via a network-based or dedicated connection. In one set of embodiments, databases **214** and **216** may reside in a Storage-Area Network (SAN). Similarly, any necessary files for performing the functions attributed to server **212** may be stored locally on server **212** and/or remotely, as appropriate. In one set of embodiments, databases **214** and **216** may include relational databases that are adapted to store, update, and retrieve data in response to commands, e.g., MySQL-formatted commands. Additionally or alternatively, server **212** can provide and support big data processing on unstructured data including but not limited to Hadoop processing, NoSQL databases, graph databases etc. In yet other implementations, server **212** may perform non-database types of bog data applications including but not limited to machine learning.

[0079] FIG. **3** is a block diagram illustrating an exemplary computer system in which embodiments of the present invention may be implemented. The system **300** may be used to implement any of the computer systems described above. As shown in the figure, computer system **300** includes a processing unit **304** that communicates with a number of peripheral subsystems via a bus subsystem **302**. These peripheral subsystems may include a processing acceleration unit **306**, an I/O subsystem **308**, a storage subsystem **318** and a communications subsystem **324**. Storage subsystem **318** includes tangible computer-readable storage media **322** and a system memory **310**.

[0080] Bus subsystem **302** provides a mechanism for letting the various components and subsystems of computer system **300** communicate with each other as intended. Although bus subsystem **302** is shown schematically as a single bus, alternative embodiments of the bus subsystem may utilize multiple buses. Bus subsystem **302** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. For example, such architectures may include an Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, Peripheral Component Interconnect (PCI) bus, which can be implemented as a Mezzanine bus manufactured to the IEEE P1386.1 standard, or PCI enhanced (PCIe) bus.

[0081] Processing unit **304**, which can be implemented as one or more integrated circuits (e.g., a conventional microprocessor or microcontroller), controls the operation of computer system **300**. One or more processors may be included in processing unit **304**. These processors may include single core or multicore processors. In certain embodiments, processing unit **304** may be implemented as one or more independent processing units **332** and/or **334** with single or multicore processors included in each processing unit. In other embodiments, processing unit **304** may also be implemented as a quad-core processing unit formed by integrating two dual-core processors into a single chip.

[0082] In various embodiments, processing unit **304** can execute a variety of programs in response to program code and can maintain multiple concurrently executing programs or processes. At any given time, some or all of the program code to be executed can be resident in processor(s) **304** and/or in storage subsystem **318**. Through suitable programming, processor(s) **304** can provide various functionalities described above. Computer system **300** may additionally include a processing acceleration unit **306**, which can include a Digital Signal Processor (DSP), a special-purpose processor, and/or the like.

[0083] I/O subsystem **308** may include user interface input devices and user interface output devices. User interface input devices may include a keyboard, pointing devices such as a mouse or trackball, a touchpad or touch screen incorporated into a display, a scroll wheel, a click wheel, a dial, a button, a switch, a keypad, audio input devices with voice command recognition systems, microphones, and other types of input devices. User interface input devices may include, for example, motion sensing and/or gesture recognition devices

such as the Microsoft Kinect® motion sensor that enables users to control and interact with an input device, such as the Microsoft Xbox® 360 game controller, through a natural user interface using gestures and spoken commands. User interface input devices may also include eye gesture recognition devices such as the Google Glass® blink detector that detects eye activity (e.g., 'blinking' while taking pictures and/or making a menu selection) from users and transforms the eye gestures as input into an input device (e.g., Google Glass®). Additionally, user interface input devices may include voice recognition sensing devices that enable users to interact with voice recognition systems (e.g., Siri® navigator), through voice commands.

[0084] User interface input devices may also include, without limitation, three dimensional (3D) mice, joysticks or pointing sticks, gamepads and graphic tablets, and audio/visual devices such as speakers, digital cameras, digital camcorders, portable media players, webcams, image scanners, fingerprint scanners, barcode reader 3D scanners, 3D printers, laser rangefinders, and eye gaze tracking devices. Additionally, user interface input devices may include, for example, medical imaging input devices such as computed tomography, magnetic resonance imaging, position emission tomography, medical ultrasonography devices. User interface input devices may also include, for example, audio input devices such as MIDI keyboards, digital musical instruments and the like.

[0085] User interface output devices may include a display subsystem, indicator lights, or non-visual displays such as audio output devices, etc. The display subsystem may be a Cathode Ray Tube (CRT), a flat-panel device, such as that using a Liquid Crystal Display (LCD) or plasma display, a projection device, a touch screen, and the like. In general, use of the term "output device" is intended to include all possible types of devices and mechanisms for outputting information from computer system **300** to a user or other computer. For example, user interface output devices may include, without limitation, a variety of display devices that visually convey text, graphics and audio/video information such as monitors, printers, speakers, headphones, automotive navigation systems, plotters, voice output devices, and modems.

[0086] Computer system **300** may comprise a storage subsystem **318** that comprises software elements, shown as being currently located within a system memory **310**. System memory **310** may store program instructions that are loadable and executable on processing unit **304**, as well as data generated during the execution of these programs.

[0087] Depending on the configuration and type of computer system **300**, system memory **310** may be volatile (such as Random Access Memory (RAM)) and/or non-volatile (such as Read-Only Memory (ROM), flash memory, etc.) The RAM typically contains data and/or program modules that are immediately accessible to and/or presently being operated and executed by processing unit **304**. In some cases, system memory **310** can comprise one or more Double Data Rate fourth generation (DDR4) Dual Inline Memory Modules (DIMMs). In some implementations, system memory **310** may include multiple different types of memory, such as Static Random Access Memory (SRAM) or Dynamic Random Access Memory (DRAM). In some implementations, a Basic Input/Output System (BIOS), containing the basic routines that help to transfer information between elements within computer system **300**, such as during start-up, may typically be stored in the ROM. By way of example, and not limitation, system memory **310** also illustrates application programs **312**, which may include client applications, Web browsers, mid-tier applications, Relational Database Management Systems (RDBMS), etc., program data **314**, and an operating system **316**. By way of example, operating system **316** may include various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems, a variety of commercially-available UNIX® or UNIX-like operating systems (including without limitation the variety of GNU/Linux operating systems, the Google Chrome® OS, and the like) and/or mobile operating systems such as iOS, Windows® Phone, Android® OS, BlackBerry® 10 OS, and Palm® OS operating systems.

[0088] Storage subsystem **318** may also provide a tangible computer-readable storage medium for storing the basic programming and data constructs that provide the functionality of some embodiments. Software (programs, code modules, instructions) that when executed by a processor provide the functionality described above may be stored in storage subsystem **318**. These software modules or instructions may be executed by processing unit **304**. Storage subsystem **318** may also provide a repository for storing data used in accordance with the present invention.

[0089] Storage subsystem **300** may also include a computer-readable storage media reader **320** that can further be connected to computer-readable storage media **322**. Together and, optionally, in combination with system memory **310**, computer-readable storage media **322** may comprehensively represent remote, local, fixed, and/or removable storage devices plus storage media for temporarily and/or more permanently containing, storing, transmitting, and retrieving computer-readable information.

[0090] Computer-readable storage media **322** containing code, or portions of code, can also include any appropriate media known or used in the art, including storage media and communication media, such as but not limited to, volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of information. This can include tangible computer-readable storage media such as RAM, ROM, Electronically Erasable Programmable ROM (EEPROM), flash memory or other memory technology, CD-ROM, Digital Versatile Disk (DVD), or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or other tangible computer readable media. This can also include nontangible computer-readable media, such as data signals, data transmissions, or any other medium which can be used to transmit the desired information and which can be accessed by computing system **300**.

[0091] By way of example, computer-readable storage media **322** may include a hard disk drive that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive that reads from or writes to a removable, nonvolatile magnetic disk, and an optical disk drive that reads from or writes to a removable, nonvolatile optical disk such as a CD ROM, DVD, and Blu-Ray® disk, or other optical media. Computer-readable storage media **322** may include, but is not limited to, Zip® drives, flash memory cards, Universal Serial Bus (USB) flash drives, Secure Digital (SD) cards, DVD disks, digital video tape, and the like. Computer-readable storage media **322** may also include, Solid-State Drives (SSD) based on non-volatile memory such as flash-memory based SSDs, enterprise flash drives, solid state ROM, and the like, SSDs based on volatile memory such as

solid state RAM, dynamic RAM, static RAM, DRAM-based SSDs, Magnetoresistive RAM (MRAM) SSDs, and hybrid SSDs that use a combination of DRAM and flash memory based SSDs. The disk drives and their associated computer-readable media may provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for computer system **300**.

[0092] Communications subsystem **324** provides an interface to other computer systems and networks. Communications subsystem **324** serves as an interface for receiving data from and transmitting data to other systems from computer system **300**. For example, communications subsystem **324** may enable computer system **300** to connect to one or more devices via the Internet. In some embodiments communications subsystem **324** can include Radio Frequency (RF) transceiver components for accessing wireless voice and/or data networks (e.g., using cellular telephone technology, advanced data network technology, such as 3G, 4G or Enhanced Data rates for Global Evolution (EDGE), WiFi (IEEE 802.11 family standards, or other mobile communication technologies, or any combination thereof), Global Positioning System (GPS) receiver components, and/or other components. In some embodiments communications subsystem **324** can provide wired network connectivity (e.g., Ethernet) in addition to or instead of a wireless interface. In some cases, communications subsystem **324** can be implemented in whole or in part as one or more PCIe cards.

[0093] In some embodiments, communications subsystem **324** may also receive input communication in the form of structured and/or unstructured data feeds **326**, event streams **328**, event updates **330**, and the like on behalf of one or more users who may use computer system **300**.

[0094] By way of example, communications subsystem **324** may be configured to receive data feeds **326** in real-time from users of social networks and/or other communication services such as Twitter® feeds, Facebook® updates, web feeds such as Rich Site Summary (RSS) feeds, and/or real-time updates from one or more third party information sources.

[0095] Additionally, communications subsystem **324** may also be configured to receive data in the form of continuous data streams, which may include event streams **328** of real-time events and/or event updates **330**, that may be continuous or unbounded in nature with no explicit end. Examples of applications that generate continuous data may include, for example, sensor data applications, financial tickers, network performance measuring tools (e.g. network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like.

[0096] Communications subsystem **324** may also be configured to output the structured and/or unstructured data feeds **326**, event streams **328**, event updates **330**, and the like to one or more databases that may be in communication with one or more streaming data source computers coupled to computer system **300**.

[0097] Computer system **300** can be one of various types, including a handheld portable device (e.g., an iPhone® cellular phone, an iPad® computing tablet, a PDA), a wearable device (e.g., a Google Glass® head mounted display), a PC, a workstation, a mainframe, a kiosk, a server rack, or any other data processing system.

[0098] Due to the ever-changing nature of computers and networks, the description of computer system **300** depicted in the figure is intended only as a specific example. Many other configurations having more or fewer components than the system depicted in the figure are possible. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, firmware, software (including applets), or a combination. Further, connection to other computing devices, such as network input/output devices, may be employed. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments.

[0099] As introduced above, embodiments of the invention provide systems and methods for managing processing, memory, storage, network, and cloud computing to significantly improve the efficiency and performance of processing nodes such as any of the servers or other computers or computing devices described above. Embodiments described herein can be implemented in a set of hardware components that, in essence, change the way in which processing, memory, storage, network, and cloud are managed by breaking down the artificial distinctions between processing, memory, storage and networking in today's commodity solutions to significantly improve the performance of commodity hardware. For example, the hardware elements can include a standard format memory module, such as a Dual Inline Memory Module (DIMM), which can be added to any of the computer systems described above. For example, the memory module can be added to commodity or "off-the-shelf" hardware such a server node and acts as a big data accelerator within that node. The components can also include one or more object routers. Object routers can include, for example, a PCI express card added to the server node along with the memory module and one or more external object routers such as rack mounted routers, for example. Object routers can be used to interconnect two or more servers or other nodes adapted with the memory modules and help to manage processing, memory, and storage across these different servers Object routers can forward objects or portions of objects based on object addresses and participate in operation of the object memory fabric. Together, these hardware components can be used with commodity servers or other types of computing nodes in any combination to implement an object memory fabric architecture.

[0100] FIG. **4** is a block diagram illustrating an exemplary object memory fabric architecture according to one embodiment of the present invention. As illustrated here, the architecture **400** comprises an object memory fabric **405** supporting any number of applications **410***a-g*. As will be described in greater detail below, this object memory fabric **405** can comprise any number of processing nodes such as one or more servers having installed one or more memory modules as described herein. These nodes can be interconnected by one or more internal and/or external object routers as described herein. While described as comprising one or more servers, it should be noted that the processing nodes of the object memory fabric **405** can comprise any of a variety of different computers and/or computing devices adapted to operate within the object memory fabric **405** as described herein.

[0101] According to one embodiment, the object memory fabric **405** provides an object-based memory which manages memory objects within the memory of the nodes of the object memory fabric **405** and at the memory layer rather than in the application layer. That is, the objects and associated properties can be implemented and managed natively in the nodes of

the object memory fabric **405** to provide increased functionality without any software and increasing efficiency and performance by dynamically managing object characteristics including, but not limited to persistence, location and processing. Object properties can also propagate to the applications **410***a-g*. The memory objects of the object memory fabric **405** can be used to eliminate typical size constraints on memory space of the commodity servers or other nodes imposed by address sizes. Rather, physical addressing can be managed within the memory objects themselves and the objects can in turn be accessed and managed through the object name space. The memory objects of the object memory fabric **405** can also be used to eliminate the distinction between memory (temporary) and storage (persistent) by implementing and managing both within the objects. The object memory fabric **405** can also eliminate the distinction between local and remote memory by transparently managing the location of objects (or portions of objects) so all objects appear simultaneously local to all nodes. The memory objects can also eliminate the distinction between processing and memory through methods of the objects to place the processing within the memory itself. In other words, embodiments of the present invention provide a single-level memory that puts the computes with the storage and the storage with the computes, directly and thereby eliminating numerous levels of software overhead communicating across these levels and the artificial overhead of moving data to be processed.

[0102] In these ways, embodiments of the object memory fabric **405** and components thereof as described herein can provide transparent and dynamic performance acceleration, especially with big data or other memory intensive applications by reducing or eliminating overhead typically associated with memory management, storage management, networking, data directories, and data buffers at both the system and application software layers. Rather, management of the memory objects at the memory level can significantly shorten the pathways between storage and memory and between memory and processing, thereby eliminating the associated overhead between each.

[0103] Embodiments provide coherent, hardware-based, infinite memory managed as memory objects with performance accelerated in-memory, spanning all nodes, and scalable across all nodes. This enables transparent dynamic performance acceleration based on the object and end application. Using an architecture according to embodiments of the present invention, applications and system software can be treated the same and as simple as a single, standard server but additionally allowing memory fabric objects to capture heuristics. Embodiments provide multiple dimensions of accelerated performance including locality acceleration. According to one embodiment, object memory fabric metadata associated with the memory objects can include triggers which enable the object memory fabric architecture to localize and move data to fast dram memory ahead of use. Triggers can be a fundamental generalization that enables the memory system to execute arbitrary functions based on memory access. Various embodiments can also include an instruction set which can provide a unique instruction model for the object memory fabric based on the triggers defined in the metadata associated with each memory object and that supports core operations and optimizations and allows the memory intensive portion of applications to be more efficiently executed in a highly parallel manner within IMF.

[0104] Embodiments can also decrease software pathlength by substituting a small number of memory references for a complex application, storage and network stack. This can be accomplished when memory and storage is directly addressable as memory under embodiments of the present invention. Embodiments can additionally provide accelerated performance of high level memory operations. For many cases, embodiments of the object memory fabric architecture can eliminate the need to move data to the processor and back to memory, which is extremely inefficient for today's modern processors with three or more levels of caches.

[0105] FIG. **5** is a block diagram illustrating an exemplary memory fabric object memory according to one embodiment of the present invention. More specifically, this example illustrates an application view of how memory fabric object memory can be organized. Memory fabric object address space **500** can be a 128 bit linear address space where the object ID corresponds to the start of the addressable object. Objects **510** can be variable size from 2^{12} to 2^{64} bytes. The address space **500** can efficiently be utilized sparsely within and across objects as object storage is allocated on a per block basis. The size of the object space **500** is meant to be large enough that garbage collection is not necessary and to enable disjoint systems to be easily combined.

[0106] Object metadata **505** associated with each object **510** can be transparent with respect to the object address space **500** and can utilize the object memory fabric to manage objects and blocks within objects and can be accessible at appropriate privilege by applications **515***a-g* through Application Program Interfaces (APIs) of the object memory fabric. This API provides functions for applications to set up and maintain the object memory fabric, for example by using modified Linux libc. With a small amount of additional effort applications such as a SQL database or graph database can utilize the API to create memory objects and provide and/or augment object metadata to allow the object memory fabric to better manage objects. Object metadata **505** can include object methods, which enable performance optimization through dynamic object-based processing, distribution, and parallelization. Metadata can enable each object to have a definable security policy and access encapsulation within an object.

[0107] According to embodiments of the present invention, applications **515***a-g* can now access a single object that captures it's working and/or persistent data (such as App**0** **515***a*) or multiple objects for finer granularity (such as App**1** **515***b*). Applications can also share objects. Object memory **500** according to these embodiments can physically achieves this powerfully simple application view with a combination of physical organization, which will be described in greater detail below with reference to FIG. **6**, and object memory dynamics. Generally speaking, the object memory **500** can be organized as a distributed hierarchy that creates hierarchical neighborhoods for object storage and applications **515***a-g*. Object memory dynamics interact and leverage the hierarchal organization to dynamically create locals of objects and applications (object methods) that operate on objects. Since object methods can be associated with memory objects, as objects migrate and replicate on the memory fabric, object methods naturally gain increased parallelism as object size warrants. The hierarchy in conjunction with object dynamics can further create neighborhoods of neighborhoods based on the size and dynamics of the object methods.

[0108] FIG. **6** is a block diagram illustrating an exemplary object memory dynamics and physical organization according to one embodiment of the present invention. As illustrated in this example, an object memory fabric **600** as described above can include any number of processing nodes **605** and **610** communicatively coupled via one or more external object routers **615**. Each node **605** and **610** can also include an internal object router **620** and one or more memory modules. Each memory module **625** can include a node object memory **635** supporting any number of applications **515***a-g*. Generally speaking, the memory module **625**, node object router **620** and inter-node object router **615** can all share a common functionality with respect to the object memory **635** and index thereof. In other words, the underlying design objects can be reused in all three providing a common design adaptable to hardware of any of a variety of different form factors and types in addition to those implementations described here by way of example.

[0109] More specifically, a node can comprise a single node object router **620** and one or more memory modules **625** and **630**. According to one embodiment, a node **605** can comprise a commodity or "off-the-shelf" server, the memory module **625** can comprise a standard format memory card such as a Dual-Inline Memory Module (DIMM) card, and the node object router **620** can similarly comprise a standard format card such as a Peripheral Component Interconnect express (PCIe) card. The node object router **620** can implement an object index covering the objects/blocks held within the object memory(s) **635** of the memory modules **625** and **630** within the same node **605**. Each memory module **625** and **630** can hold the actual objects and blocks within objects, corresponding object meta-data, and object index covering objects currently stored local to that memory module. Each memory module **625** and **630** can independently manage both dram memory (fast and relatively expensive) and flash memory (not as fast, but much less expensive) in a manner that the processor (not shown) of the node **605** thinks that there is the flash amount of fast dram. The memory modules **625** and **630** and the node object router **620** can both manage free storage through a free storage index implemented in the same manner as for other indexes. Memory modules **625** and **630** can be directly accessed over the standard DDR memory bus by processor caches and processor memory reference instructions. In this way, the memory objects of the memory modules **625** and **630** can be accessed using only conventional memory reference instructions and without implicit or explicit Input/Output (I/O) instructions.

[0110] Objects within the object memory **635** of each node **625** can be created and maintained through an object memory fabric API (not shown). The node object router **620** can communicate with the API through a modified object memory fabric version of libc and an object memory fabric driver (not shown). The node object router **620** can then update a local object index, send commands toward a root, i.e., towards the inter-node object router **615**, as required and communicate with the appropriate memory module **625** or **630** to complete the API command locally. The memory module **625** or **630** can communicate administrative requests back to the node object router **620** which can handle them appropriately.

[0111] According to one embodiment, the internal architecture of the node object router **620** can be very similar to the memory module **625** with the differences related to routing functionality such as managing a node memory object index and routing appropriate packets to and from the memory modules **625** and **630** and the inter-node object router **615**. That is, the node object router **620** can have additional routing functionality but does not need to actually store memory objects.

[0112] The inter-node object router **615** can be considered analogous to an IP router. However, the first difference is the addressing model used. IP routers utilize a fixed static address per each node and routes based on the destination IP address to a fixed physical node. However, the inter-node object router **615** of the object memory fabric **600** utilizes a memory fabric object address (OA) which specifies the object and specific block of the object. Objects and blocks can dynamically reside at any node. The inter-node object router **615** can route OA packages based on the dynamic location(s) of objects and blocks and track object/block location dynamically in real time. The second difference is that the object router can implement the object memory fabric distributed protocol which provides the dynamic nature of object/block location and object functions, for example including, but not limited, to triggers. The inter-node object router **615** can be implemented as a scaled up version of node object router **620** with increased object index storage capacity, processing rate and overall routing bandwidth. Also, instead of connecting to a single PCIe or other bus or channel to connect to memory modules, inter-node object router **615** can connect to multiple node object routers and/or multiple other inter-node object routers. According to one embodiment, a node object router **620** can communicate with the memory modules **625** and **630** with direct memory access over PCIe and the memory bus (not shown) of the node **605**. Node object routers of different nodes **605** and **610** can in turn connect with one or more inter-node object routers **615** over a high-speed network (not shown) such as 25/100GE fiber that uses several layers of Gigabit Ethernet protocol or object memory fabric protocol tunneled through standard IP, for example. Multiple inter-node object routers can connect with the same network.

[0113] In operation, the memory fabric object memory can physically achieve its powerfully simple application view described above with reference to FIGS. **4** and **5** with a combination of physical organization and object memory dynamics. According to one embodiment and as introduced above with reference to FIG. **5**, the memory fabric object memory can be organized as a distributed hierarchy that creates hierarchical neighborhoods for object storage and applications **515***a-g*. The node object routers can keep track of which objects and portions of objects are local to a neighborhood. The actual object memory can be located on nodes **605** or **610** close to applications **515***a-g* and memory fabric object methods.

[0114] Also as introduced above, object memory dynamics can interact and leverage the hierarchal organization to dynamically create locals of objects and applications (object methods) that operate on objects. Since object methods can be associated with objects as objects migrate and replicate across nodes, object methods naturally gain increased parallelism as object size warrants. This object hierarchy, in conjunction with object dynamics, can in turn create neighborhoods of neighborhoods based on the size and dynamics of the object methods.

[0115] For example, App**0** **515***a* spans multiple memory modules **625** and **630** within a single level object memory fabric neighborhood, in this case node **605**. Object movement can stay within that neighborhood and its node object router **620** without requiring any other communication links or rout-

ers. The self-organizing nature along the hierarchy defined neighborhoods provides efficiency from a performance and minimum bandwidth perspective. In another example, App**1** (A**1**) **515***b* can have the same characteristic but in a different neighborhood, i.e., in node **610**. App**2** (A**2**) **515***c* can be a parallel application across a two-level hierarchy neighborhood, i.e., nodes **605** and **610**. Interactions can be self-contained in the respective neighborhood.

[0116] As noted above, certain embodiments may include a data types and metadata architecture certain embodiments can also include a data types and metadata architecture that facilitate multiple advantages of the present invention. With respect to the architecture, the following description discloses various aspects of: object memory fabric address spaces; an object memory fabric coherent object address space; an object memory fabric distributed object memory and index; an object memory fabric index; object memory fabric objects; and an extended instruction execution model. Various embodiments may include any one or combination of such aspects.

[0117] FIG. **7** is a block diagram illustrating an aspect of object memory fabric hierarchy of object memory, which localizes working sets and allows for virtually unlimited scalability, according to one embodiment of the present invention. As disclosed herein, certain embodiments may include core organization and data types that enable the object memory fabric to dynamically operate to provide the object memory application view. The core organization and data types facilitate the fractal-like characteristics of the system which allow the system to behave identically in a scale-independent fashion. In the depicted example, an object memory fabric **700** as disclosed herein can include any number of processing nodes **705** and **710** communicatively coupled at higher levels via one or more external object routers, such as object router **715**, which may in turn be coupled to one or more higher level object routers.

[0118] Specifically, the system may be a fat-tree built from nodes, from leaf nodes to root node(s). According to certain embodiments, each node may just understand whether its scope encompasses an object and based on that whether to route a request/response toward the root or leaf. Putting these nodes together enables a system to dynamically scale to any capacity, without impacting the operation or perspective of any node. In some embodiments, the leaf node may be a DIMM built from standard memory chips, plus object memory fabric **700** implemented within an FPGA. In some embodiments, standard memory chips could have object memory fabric **700** imbedded. In various embodiments, implementations may have remote nodes such as mobile phones, drones, cars, internet of things components, and/or the like.

[0119] To facilitate various advantageous properties of object memory fabric **700**, certain embodiments may employ coherent object memory fabric address spaces. Table 1 below identifies non-limiting examples of various aspects of address spaces, in accordance with certain embodiments of the present disclosure. All nodes that are connected to a single object memory fabric **700**, local or distributed, can be considered part of a single system environment according to certain embodiments. As indicated in Table 1, object memory fabric **700** can provide a coherent object address space. In some embodiments, a 128-bit object address space may be provided. However, other embodiments are possible. There are several reasons for a large object address space, including the following. The object address space is to directly uniquely address and manage all memory, storage across all nodes within an object memory fabric system, and provide a unique address for conventional storage outside of an object memory fabric system. The object address space can allow an address to be used once and never garbage collect, which is a major efficiency. The object address space can allow a distinction between allocating address space and allocating storage. In other words, the object address space can be used sparsely as an effective technique for simplicity, performance, and flexibility.

[0120] As further indicated in Table 1, the object memory fabric **700** can directly support per-process virtual address spaces and physical address spaces. With some embodiments, the per-process virtual address spaces and physical address spaces may be compatible with x86-64 architecture. In certain embodiments, the span of a single virtual address space may be within a single instance of Linux OS, and may be usually coincident with a single node. The object memory fabric **700** may enable the same virtual address space to span more than a single node. The physical address space may be the actual physical memory addressing (e.g., within an x86-64 node in some embodiments).

TABLE 1

Address Spaces			
Parameter	Object memory fabric Object Address Space	Virtual Address	Physical Address
Description	Object memory fabric address	Process address handle to object memory fabric address	Cache of object memory fabric
Scope	Global	Per process, can be shared	Per node
Size	2^{128}	2^{64} (2^{48} Haswell)	2^{46} (Haswell)
Object Support	Yes, object memory fabric object index tree and per object index	Yes, page tables	Yes, object memory fabric metadata
Object Sizes		$2^{\{12\|21\|30\|39\|48\|57\|64\}}$	
Address Space Allocation	Sparse - with or without storage, object units	Sparse - with or without storage, object units	Sparse - page
Storage Allocation	Object or block (page)	Based on object memory fabric	Page
Security (Access)	Through virtual address, operating system, and file system	Operating system	Operating system/ object memory fabric

[0121] FIG. **8** is a block diagram illustrating an example relationship **800** between object address space **805**, virtual addresses **810**, and physical addresses **815**, in accordance with certain embodiments of the present disclosure. With object address space **805**, a single object can range in size. By way of example without limitation, a single object can range in size from 2 megabytes (2^{21}) to 16 petabytes (2^{64}). Other ranges are possible. Within the object memory fabric **700**, object address space **805** may be allocated on an object granularity basis in some embodiments. In some embodiments, storage may be allocated on a 4k byte block basis (e.g., blocks **806**, **807**). Thus, the object address space block **806**, **807** in

some embodiments may correspond to the 4k byte page size within x86-64 architecture. When the object address space **805** is created, only the address space and object metadata may exist. When storage is allocated on a per block basis, there can be data stored in the corresponding block of the object. Block storage can be allocated in a sparse or non-sparse manner and pre and/or demand allocated. For example, in some embodiments, software can use an object as a hash function and only allocate physical storage for the valid hashes.

[0122] Referring to the example of FIG. **8**, within a node **820**, **825**, which could be a conventional server in some embodiments, physical pages corresponding to physical addresses **815** may be allocated on a dynamic basis corresponding to the virtual addresses **810**. Since object memory fabric **700** actually provides the physical memory within a node **820**, **825***by* way of the object memory fabric DIMM, when a virtual address segment **811**, **812**, **813**, **814** is allocated, an object address space **805** object which corresponds to the particular segment **811**, **812**, **813**, **814** can also be created. This enables the same or a different virtual address **810** across nodes **820**, **825** to address and access the same object. The actual physical address **815** at which a block/page within an object resides within a node **820**, **825** can vary over time within or across nodes **820**, **825**, transparently to application software.

[0123] Certain embodiments of the object memory fabric **700** may provide key advantages: embodiments of object memory fabric **700** may provide integrated addressing, objects with transparent invariant pointers (no swizzling required), and methods to access a large address space across nodes—a with certain embodiments being compatible with x84-64, Linux, and applications. Normally, systems have numerous different addresses (e.g., for memory address with separate address space, sectors, cylinders, physical disks, database systems, file systems, etc.), which requires significant software overhead for converting, buffering, and moving objects and blocks between different layers of addresses. Using integrated addressing to address objects, and blocks within objects, and using the object namespace eliminates layers of software by having single-level addressing invariant across all nodes/systems. With a sufficiently large address space, one address system is not invariant with particular database application and all these systems working together.

[0124] Thus, a node may include a memory module may store and manage one or more memory objects, where physical address of memory and storage is managed with each of the one or more memory objects based at least in part on an object address space that is allocated on a per-object basis with a single-level object addressing scheme. The node may be configured to utilize the object addressing scheme to operatively couple to one or more additional nodes to operate as a set of nodes of an object memory fabric, where the set of nodes operates so that all memory objects of the set of nodes are accessible based at least in part on the object addressing scheme, the object addressing scheme defining invariant object addresses for the one or more memory objects that are invariant with respect to physical memory storage locations and storage location changes of the one or more memory objects within the memory module and across all modules interfacing the object memory fabric. Accordingly, the object addresses are invariant within a module and across all modules that interface to object memory fabric, regardless of whether the objects are in a single server or not. Even though the objects can be stored on any or all modules, the object addresses are still invariant no matter at which physical memory locations the objects are currently or will be stored. The following provides details of certain embodiments that may provide such advantages through the object address space and object address space pointers.

[0125] Certain embodiments of object memory fabric **700** may support multiple, various pointer formats. FIG. **9** is a block diagram illustrating an example relationship **900** between object sizes **905** and object address space pointers **910**, in accordance with certain embodiments of the present disclosure. Table 1Table 2 below identifies non-limiting examples of aspects of the object address space pointer **910**, in accordance with certain embodiments of the present disclosure. As indicated by Table 1Table 2, some example embodiments can support three pointer formats. The object address space format may be an object memory fabric native 128 bit format and can provide a single pointer with full addressability for any object and offset within object. Object memory fabric **700** can support additional formats, for example, two additional formats in 64 bit format to enable direct compatibility with x86-64 virtual memory and virtual address. Once a relationship between an object memory fabric object and a virtual address segment is established by object memory fabric API (which can be handled transparently to the application in Linux libc, in some embodiments), standard x86 Linux programs can directly reference data within an object (x86 segment) efficiently and transparently utilizing the x86-64 addressing mechanisms.

TABLE 2

Object Address Space Pointer Formats

Pointer Type	Object memory fabric Pointer	Object Address Space Generation	Trans-formation to Virtual Address	Virtual Address Format
Object memory fabric Address	128 bit Storage	Direct	None	None
Object Relative	Offset (64 bit)	ObjStart + ObjOffset	None	virtual address base + offset address mode
Object Virtual Address	Offset (64 bit)	ObjStart + ObjOffset	Add virtual address base to offset	48 bit virtual address with 64 bit data type

[0126] Table 1Table 2Table 3 below identifies non-limiting examples of aspects of the object address space pointers in relation to object sizes, in accordance with certain embodiments of the present disclosure. Embodiments of object address space can supports multiple segment sizes, for example, six segment sizes from 2^{21} to 2^{64} as illustrated in Table 1Table 2Table 3 below. The object sizes correspond to the x86-64 virtual memory segment and large page sizes. Objects can start on a modulo 0 object size boundary. Object address space pointers **910** may be broken into ObjStart and ObjOffset fields, the sizes of which are dependent on the object size as shown in the example below. The Obj Start field corresponds to the object address space start of the object and also the ObjectID. The Obj Offset is an unsigned value in a range from zero to (ObjectSize−1) with specifies the offset within an object. Object metadata can specify the object size and object memory fabric interpretation of the object address

space pointer **910**. Objects of arbitrary size and sparseness can be specified by only allocating storage for blocks of interest within an object.

[0127] Because of the nature of most applications and object nature of object memory fabric **700**, most addressing can be relative to an object. In some embodiments, all the object memory fabric address pointer formats can be natively stored and loaded by the processor. Object Relative and Object Virtual Address can work directly with x86-64 addressing modes in some embodiments. Object Virtual Address pointer can be or include a process virtual address that works within the x86-64 segment and corresponding object memory fabric object. Object memory fabric object address space can be calculated by using the Object Virtual Address as an object offset. Object Relative pointer can be or include an offset into an x86-64 virtual address segment, thus base plus index addressing mode works perfectly. Object memory fabric object address space can be calculated by using the Object Relative as an object offset. Table 3 below identifies non-limiting examples of details of generating a 128 bit object address space from an Object Virtual Address or Object Relative pointer as a function of object size, in accordance with certain embodiments of the present disclosure.

TABLE 3

Object Address Space Generation

Object Size	Object Address Space Generation from Object Relative and Object Virtual Address Pointers
2^{21}	IA[127:00] = (ObjBase[127:21], zero[20:0]) + (zero[127:21], ObjOffset[20,0])
2^{30}	IA[127:00] = (ObjBase[127:30], zero[29:0]) + (zero[127:30], ObjOffset[29,0])
2^{39}	IA[127:00] = (ObjBase[127:39], zero[38:0]) + (zero[127:39], ObjOffset[38,0])
2^{48}	IA[127:00] = (ObjBase[127:48], zero[47:0]) + (zero[127:48], ObjOffset[47,0])
2^{57}	IA[127:00] = (ObjBase[127:57], zero[56:0]) + (zero[127:57], ObjOffset[56,0])
2^{64}	IA[127:00] = (ObjBase[127:21], zero[20:0]) + (zero[127:21], ObjOffset[20,0])

[0128] As disclosed herein, certain embodiments may include an object memory fabric distributed object memory and index. With the distributed index, individual nodes may index local objects and blocks of objects on a per-object basis. Certain embodiments of object memory fabric distributed object memory and index may be based at least in part on an intersection concept of cellular automata and fat trees. Prior distributed hardware and software systems with real-time dynamic indices used two approaches: a centralized index or a distributed single conceptual index. Embodiments of object memory fabric may use a new approach which overlays an independent local index function on top of a fat-tree hierarchical network.

[0129] FIG. **10** is a block diagram illustrating an example object memory fabric distributed object memory and index structure **1000**, in accordance with certain embodiments of the present disclosure. At leaves of the structure **1000** are any number of processing nodes **1005** and **1010** object memories **1035**. These object memories **1035** may each have an object index that describes the objects and portions of objects currently stored locally in the object memories **1035**. A number of object memories **1035**, which in some embodiments may be DDR4-DIMM interface compatible cards within a single node are logically connected with an object memory fabric node object index **1040**. The object memory fabric node object indices **1040** may each have an object index that describes the objects and portions of objects currently stored locally and/or currently stored in the object memories **1035**. In some embodiments, the object memory fabric node object index **1040** can be instantiated as a PCIe card. With some embodiments, the object memory fabric object memory DDR4-DIMM and object memory fabric node object index PCIe card can communicate over PCIe and memory bus.

[0130] In some embodiments, the object memory fabric node object index **1040** works identically to the object index within the object memory **1035**, except that the object memory fabric node object index **1040** tracks all objects and portions of objects that are within any of the connected object memories **1035** and maps the objects and portions of objects to particular object memory **1035**. The next level up in the tree is an node object router object index **1020** that may be provided by an object memory fabric router that performs the same object index function for all the object memory fabric node object indices **1040** to which it is connected. The node object router object indices **1020** may each have an object index that describes the objects and portions of objects currently stored locally in lower levels (e.g., at **1040**, **1035**). Thus, according to some embodiments, router modules may have directory and router functions, whereas memory modules may have directory and router functions, as well as memory functions to store memory objects. However, other embodiments are possible, and, in alternative embodiments, the router modules may additionally have memory functions to store memory objects.

[0131] The pattern may illustrated by the structure **1000** may continue to another higher level inter-node object router object index **1015** that may be provided by an object memory fabric router that performs the same object index function for all the object memory fabric node object indices to which it is connected, and so on to the root of the tree. Thus, in certain embodiments, each object index and each level may perform the same function, independently, but, the aggregate of object indices and levels as a tree network may provide a real time dynamic distributed index, with great scalability properties, that efficiently tracks and localizes memory objects and blocks. An additional property may be that the combination of tree, distributed indices, and caching enable a significant reduction in bandwidth requirements. This may be illustrated by the hierarchically indicated neighborhoods that are delineated by object memory fabric router to leafs (down in this case). As the level of the defined hierarchy increases, so does the aggregate object memory caching capacity. So, as an application working set fits within the aggregate capacity of a given level, the bandwidth requirement at the level toward the root may go to zero.

[0132] As disclosed herein, each processing node is configured to utilize a set of algorithms to operatively couple to one or more additional processing nodes to operate as a set of processing nodes independently of a scale of the set of processing nodes. The set of nodes may operate so that all memory objects of the set of nodes are accessible by any node of the processing set of nodes. At the processing nodes, object memory modules may store and manage memory objects, each instantiated natively therein and managed at a memory layer, and object directories that index the memory objects and blocks thereof on a per-object basis. A memory module

may process requests based at least in part on the one or more object directories, which requests may be received from an application layer. In some case, the requests may be received from one or more additional processing nodes. Responsive to the requests, a given memory module may process an object identifier corresponding to a given request and may determine whether the memory module has requested object data. If the memory module has the requested object data, the memory module may generate a response to the request based at least in part on the requested object data. If the memory module does not have the requested object data, an object routing module may routes the first request to another node in the tree. The routing of the request may be based at least in part on the object routing module making a determination about a location of object data responsive to the request. If the object routing module identifies the location based at least in part on the object routing module's directory function, the object routing module may rout the request down toward the location (i.e., a lower level node possessing the requested object data). However, if the object routing module determines that the location is unknown, the object routing module may rout the request toward a root node (i.e., to one or more higher level object routers—inter-node object routers) for further determinations at each level until the requested object is located, accessed, and returned to the original memory module.

[0133] In addition, as disclosed herein, triggers may be defined for objects and/or blocks within objects in object metadata. The object-based triggers may predict what operations will be needed and may provide acceleration by performing the operations ahead of time. When a node receives a request that specifies an object (e.g., with a 128-bit object address), the node uses an object directory to determine if the node has any part of the object. If so, the object directory points to a per-object tree (a separate one, where the size is based on the size of the object) which may be used to locate local the blocks of interest. There could be additional trigger metadata that indicates, for the particular blocks of interest, to interpret the particular addresses in a predefined manner as the blocks are transferred to/through the memory module. The triggers may specify one or more pre-defined hardware and/or software actions on a per-block basis with respect to one or more data blocks within an object (e.g., fetch a particular address, run a more complicated trigger program, perform pre-fetching, calculate these other three blocks and send signal to software, etc.). Triggers may correspond to a hardware way to dynamically move data and/or perform other actions ahead of when such actions are needed as objects flow through any memory module of the object memory fabric. Accordingly, such actions may be effected when a particular memory object having one or more trigger is located at a respective memory module and accessed as part of the respective memory module processing one or more other requests.

[0134] FIGS. **11** and **12** are block diagrams illustrating examples at a logical level of how the distributed nature of the object index operates and interoperates with the object memory fabric protocol layering, in accordance with certain embodiments of the present disclosure. Certain embodiments of object memory fabric protocol layering may be similar to, but have important differences from, a conventional layered communication protocol. A communications protocol may be essentially stateless, but embodiments of the object memory fabric protocol may maintain object state and directly enable distributed and parallel execution—all without any centralized coordination.

[0135] FIG. **11** illustrates an object memory hit case **1100** that executes completely within the object memory **1135**, in accordance with certain embodiments of the present disclosure. Object memory **1135** may receive a processor request **1105** or background trigger activity **1106**. The object memory **1135** may manage the local DRAM memory as a cache **1130**, based on processor physical address. The most frequent case may be that the requested physical address is present and it gets immediately returned to the processor, as indicated at **1110**. The object memory **1135** may use triggers to transparently move data from slower flash memory into the fast DRAM memory, as indicated at **1115**.

[0136] For the case of a miss **1115** or background trigger activity **1106**, some embodiments may include one or a combination of the following. In some embodiments, an object memory fabric object address may be generated from the physical address, as indicated by block **1140**. The object index may generate the location in local flash memory from the object address space, as indicated by block **1145**. Object index lookup can be accelerated by two methods: (1) a hardware-based assist for index lookup; and (2) results of the object index lookup being locally cached. Object memory fabric cache coherency may be used to determine whether the local state is sufficient of the intended operation, as indicated by block **1150**. Based on the index, a lookup may be performed to determine whether the object and/or block within object are local, as indicated by block **1155**. In the case of a hit **1160**, the data corresponding to request **1105** or trigger activity **1106** may be transferred, as indicated by **1165**. And, in some embodiments, when the cache state is sufficient, a decision may be made to cache the block into DRAM.

[0137] FIG. **12** illustrates an object memory miss case **1200** and the distributed nature of the object memory and object index, in accordance with certain embodiments of the present disclosure. The object memory **1235** may go through steps described previously, but the routing/decision stage **125** may determine that the object and/or block is not local. As a result, the algorithm may involve the request traversing **1270** up the tree toward the root, until the object/block is found. Any number of levels and corresponding node elements may be traversed until the object/block is found. In some embodiments, at each step along the path, the same or similar process steps may be followed to independently determine the next step on the path. No central coordination is required. Additionally, as disclosed herein, object memory fabric API and triggers normally get executed in the leafs, but can be executed in a distributed manner at any index.

[0138] As a simplified example, in the case depicted the request traverses **1270** up from the object memory fabric node object index **1240** corresponding to object memory **1235** to the object router **1220**. The object router **1220**, with its an object router object index, may identify the request object/block as being down the branch toward object memory fabric node object index **1241**. Hence, at the index of object router **1220**, the request may then be routed **1275** toward the leaf(s) that can supply the object/block. In the example depicted, the object memory **1236** can supply the object/block. At the object memory **1236**, memory access/caching **1241** may be performed (which may include previously described process steps for a hit case being performed), and the object/block may be returned **1280** back to the original requesting leaf

1235 for the ultimate return **1290**. Again, in some embodiments, at each step along the path, the same or similar process steps may be followed to independently determine the next step on the path. For example, the original requesting leaf **1235** may perform previously described process steps **1285** for a hit case, and then return **1290** the requested data.

[0139] As disclosed herein, the operation of a single object memory fabric index structure, the object memory fabric index structure may be based on several layers of the same tree implementation. Certain embodiments employing tree structure may have several uses within object memory fabric as described in Table 4 below. However, various other embodiments are possible.

TABLE 4

Tree Structure Uses

Use	Object Memory	Node Object Index	Object Memory Fabric Router
Determine local location of objects and blocks comprising objects as function of object address space	Yes		
Determine which children hold objects, and blocks comprising objects, as a function of object address space		Yes	Yes
Generate object address space as function of local physical address (single level)	Yes		
Object virtual address to object address space		Yes	
Application defined	Yes		

[0140] FIG. **13** is a block diagram illustrating an example of leaf level object memory structure **1300** in view of the object memory fabric distributed object memory and index structure, in accordance with certain embodiments of the present disclosure. In some embodiments, the leaf level object memory structure **1300** may include a nested set of B-trees. The root tree may be the object index tree (OIT) **1305**, which may index objects locally present. The index for the object index tree **1305** may be the object memory fabric object address, since objects start at object size modulo zero. There may be one object index tree **1305** for each object that has at least a single block stored locally within the object memory.

[0141] The object index tree **1305** may provide one or more pointers (e.g., local pointers) to one or more per object index trees (POIT) **1310**. For example, every local object may have a per object index tree **1310**. A per object index tree **1310** may index object metadata and blocks belonging to the object that are locally present. The per object index tree **1310** leaves point to the corresponding metadata and blocks (e.g., based on offset within object) in DRAM **1315** and flash **1320**. A leaf for a specific block can point to both DRAM **1315** and flash **1320**, as in the case of leaf **1325**, for example. Organization of object metadata and data is disclosed further herein.

[0142] The tree structure utilized may be a modified B-tree that is copy-on-write (COW) friendly. COW is an optimization strategy that enables multiple tasks to share information efficiently without duplicating all storage where most of the data is not modified. COW stores modified blocks in a new location which works well for flash memory and caching. In certain embodiments, the tree structure utilized may be similar to that of the open source Linux file system btrfs, with major differences being utilization for a single object/memory space, hardware acceleration, and the ability of independent local indices to aggregate as described previously. By utilizing multiple layers of B-trees, there can be a higher degree of sharing and less rippling of changes. Applications, such as file systems and database storage managers, can utilize this underlying efficient mechanism for higher level operation.

[0143] FIG. **14** is a block diagram illustrating an example of object memory fabric router object index structure **1400**, in accordance with certain embodiments of the present disclosure. With some embodiments, the object memory fabric router object index and the node object index may use an almost identical structure of object index trees **1405** and per object index trees **1410** for each object. The object index trees **1405** may index objects locally present. Each object described in an object index tree **1405** may have a per object index tree **1410**. The per object index trees **1410** may index blocks and segments that are locally present.

[0144] The object memory fabric router object index and the node object index may index objects and blocks within objects that are present in the children **1415** within the tree structure **1400**, namely child router(s) or leaf object memory. An entry within a leaf in the per object index tree **1410** has the ability to represent multiple blocks within the object. Since blocks of an object may tend to cluster together naturally and due to background housekeeping, each object tends be represented much more compactly in object indices that are closer to the tree root. The object index trees **1405** and per object index trees **1410** may enable reduplication at the object and block level, since multiple leafs can point to the same blocks, as in the case of leaves **1425** and **1430**, for example. Index Copy-On-Write (COW) support enables, for example, only modified blocks to be updated for an object.

[0145] FIGS. **15**A and **15**B are block diagrams illustrating non-limiting examples of index tree structures, including node index tree structure **1500** and leaf index tree **1550**, in accordance with certain embodiments of the present disclosure. Further non-limiting examples of various aspects of index tree fields are identified in Table 5 below. Other embodiments are possible. An individual index tree may include node blocks and leaf blocks. Each node or leaf block may include of a variable number of entries based on the type and size. Type specifies type of node, node block, leaf, and/or leaf block.

TABLE 5

Index Tree Fields

Name	Description	Size
NSize	Encoded node size field. Single value for OIT node. Multiple values for POIT node based on object size corresponding to POIT index. Implies the size of NValue field.	3
ObjSize	Encoded Object Size	3
ObjectID	Maximum size object ID	107
Object Offset	4k block Based on Object size corresponding to POIT index (9-52)	52
LPointer (LP)	References local 4k block in flash or dram. Includes 32 bits of pointer and a single bit specifying dram address space.	32
LParent (LPt)	Local Parent references the local 4k block of the parent node in flash or dram. Includes 32 bits of pointer and a single bit specifying dram address space.	33

TABLE 5-continued

Index Tree Fields

Name	Description	Size
LSize	Encoded leaf LValue size.	3
Otype	Type of OIT Leaf	2
Ptype	Type of POIT Leaf	2
Etype	Type of OIT or POIT Entry Node	3
Rtype	Type of reserved Leaf	3
num	May be utilized to increase the size of data that the leaf specifies to increase the efficiency of index tree and storage device. Values may include: 1 block 4 blocks (flash page) 512 blocks (minimum size object, 2 Mbyte)	0
Children	Specifies a remote device number	32
Block State	Encoding of 4k block cache coherency state	8
	Block referenced count (unsigned)	7
	Modified - Indicates that the block has been modified with respect to persistent store. Only valid for blocks while they are present in volatile memory.	1
DS State [15:0]	DownStream State [15:0] - Enumerates the state of for the block within object specified by Object Offset for each of 16 devices.	128

[0146] Size specifies independently the size of the LPointer and IndexVal (or object offset). Within a balanced tree, a single block may point to all node blocks or all leaf blocks. In order to deliver highest performance, the tree may become un-balanced, such as for example where the number of levels for all paths through the tree are equivalent. Node blocks and leaf blocks may provide fields to support un-balanced trees. A background activity may re-balance the trees that are part of other background operations. For example, an interior node (non-leaf) in OIT may include L Pointer and NValue fields. NValue may include object size and object ID. Object ID requires 107 (128-21) bits to specify the smallest possible object. Each LPointer may point to the next level of interior node or a leaf node. LPointer may require enough bits to represent all the blocks within its local storage (approximately 32 bits representing 16 terabytes). For a node in the POIT, the NValue may consist of the object offset based on object size. The object size may be encoded within the NSize field. The size field may enable a node to hold the maximum number of LPointer and NValue fields based on usage. An index tree root node may be stored at multiple locations on multiple flash devices to achieve reliable cold boot of the OIT. Tree root block updates may be alternated among mirrors to provide wear leveling.

[0147] By default, each POIT Leaf entry may point to the location of a single block (e.g., 4k bytes). POIT Leaf OM entry and POIT Leaf Router entry may contain a field to enable support beyond single block to enable more compressed index trees, higher resulting index tree performance and higher persistent storage performance by being able to match the page size for persistent storage.

[0148] Nodes and leafs may be differentiated by the Type field at the start of each 4k block. The NNize field may encode the size of NValue field within a node, and LSize field may encode the size of the LValue field within a leaf. The size of the LPointer field may be determined by the physical addressing of local storage is fixed for a single devices (e.g., RDIMM, node router, or router). The LPointer may be only valid within a single device and not across devices. The LPointer may specify whether the corresponding block is stored in persistent memory (e.g., flash) or faster memory (e.g., DRAM). Blocks that are stored in DRAM may also have storage allocated within persistent memory, so that two entries are present to indicate the two storage locations for a block, node or leaf. Within a single block type, all NValue and/or LValue fields may be a single size.

[0149] The OIT Node may include several node level fields (Type, NSize, and LParent) and entries including OIT Node Entry or OIT Leaf Entry. Since an index tree can be unbalanced at times a node can include both node and leaf entries. The POIT Node may include one or more node level fields (e.g., Type, NSize, and/or LParent) and entries including OIT Leaf Entry. OIT Leaf types may be differentiated by the otype field. OIT Leaf (Object Index Table Leaf) may point to the head of a POIT (Per Object Index Table) that specifies object blocks and object metadata. OIT Leaf R may point to a remote head of an POIT. This may be utilized to reference an object that is residing on a remote device across a network. This leaf may enable the remote device to manage the object.

[0150] POIT Leaf types may be differentiated by the ptype field. POIT Leaf OM may point to a block of object memory or metadata. The Object offset field may be one bit greater than the number of bits to specify the offset for a specific object size to specify metadata. For example, for 2^{21} object size 10 bits may be required (9 plus 1 bits). The implementation can choose to represent the offset in two's complement form (signed form, first block metadata is −1), or in one's complement where the additional bit indicates metadata (first block of metadata is represented by 1, with metadata bit set).

[0151] POIT Leaf Remote may point to an block of object memory or metadata that is remote from the local DIMM. This may be used to reference a block that is residing on a remote device across a network through the stream package interface. For example, this device could be a mobile device. This leaf may enable object memory fabric hardware to manage coherence on a block basis for the remote device.

[0152] POIT Leaf Router may be utilized within node object routers and inter-node object routers to specify the state of the corresponding object memory fabric Block Object Address for each of up to 16 downstream nodes. If within a node object router, up to 16 DIMMs may be specified in some embodiments (or more in other embodiments). If within an inter-node object router up to 16 downstream routers or node object routers (e.g., server nodes) may be specified in some embodiments (or more in other embodiments). The Block Object Address can be present in one or more downstream devices based on valid state combinations.

[0153] Index lookups, index COW updates, and index caching may be directly supported in object memory fabric hardware in Object Memory, node object index, and object memory fabric Router. In addition to the node formats for object memory fabric indices, application-defined indices may be supported. These may be initialized through the object memory fabric API. An advantage of application-defined indices may be that object memory fabric hardware-based index lookup, COW update, index caching, and parallelism may be supported

[0154] Various embodiments may provide for background operations and garbage collection. As each DIMM and Router within object memory fabric may maintain its own directory and storage locally, background operations and garbage collection may be accomplished locally and independently. Each DIMM or Router may have a memory hierarchy for storing index trees and data blocks, that may include

on-chip cache, fast memory (e.g., DDR4 or HMC DRAM) and slower nonvolatile memory (e.g., flash) that it can manage, as well as index trees.

[0155] Each level within the hierarchy may perform the following operations: (1) Tree balancing to optimize lookup time; (2) Reference count and aging to determine when blocks are moved between different storage; (3) Free list updating for each local level of hierarchy as well as keeping a parameters of fill level of the major levels of the local hierarchy; (4) Delivering periodic fill levels to the next level of hierarchy to enable load balancing of storage between DIMMs on a local server and between levels of object memory fabric hierarchy; (5) If a Router, then load balancing between child nodes.

[0156] Block reference count may be utilized object memory fabric to indicate the relative frequency of access. Higher value may indicate more frequent use over time, lower less frequent use. When block reference count is associated with a block in persistent memory, blocks which have lowest values may be candidates to move to another DIMM or node that has more available space. Each time a block is accelerated into volatile memory, the reference count may be incremented. Low frequency background scanning may decrement the value if it is not in volatile memory and increments the value if it is in volatile memory. It may be expected that the scanning algorithm may evolve over time to increment or decrement based or reference value to provide appropriate hysteresis. Blocks that are frequently accelerated into or present in volatile memory may have higher reference count values.

[0157] When a block reference count is associated with a block in volatile memory, blocks which have lowest values may be candidates to move back to persistent memory or memory within another DIMM or node. When a block moves into volatile memory, reference count may be initialized based on the instruction or use case that initiated the movement. For example, a demand miss may set the value to a midpoint, and a speculative fetch may set it to a quarter point. Single use may set it to below the quarter point. Moderate frequency background scanning may decrement the referenced value. Thus, demand fetches may be initially weighted higher than speculative fetches. If a speculative fetch is not utilized, it may quickly fall to the lower referenced values that may be replaced first. Single use may be weighted low to be candidate for replacement sooner than other blocks. Thus, single use and speculative blocks may not replace other frequently accessed blocks.

[0158] FIG. **16** is a block diagrams illustrating an aspect of example physical memory organization **1600**, in accordance with certain embodiments of the present disclosure. Object memory fabric may provide multiple methods to access objects and blocks. For example, a direct method may be based on execution units within object memory fabric or devices that can directly generate full 128-bit memory fabric addresses may have full direct access.

[0159] An associated method may consider convention al servers having limited virtual address and physical address spaces. Object memory fabric may provide an API to dynamically associate objects (e.g., segments) and blocks (e.g., pages) with the larger object memory fabric 128-bit memory fabric address. The associations provided by AssocObj and AssocBlk operations may be utilized by object memory fabric driver (e.g., Linux driver) and object memory fabric system library (Syslib) interfacing with the standard processor memory management to enable object memory fabric to behave transparently to both the operating system and applications. Object memory fabric may provide: (a) an API to associate a processor segment and its range of virtual addresses with an object memory fabric object thus ensuring seamless pointer and virtual addressing compatibility; (b) an API to associate a page of virtual address space and the corresponding object memory fabric block with a page/block of local physical memory within an object memory fabric DIMM (which may ensure processor memory management and physical addressing compatibility); and/or (c) local physical memory divided into standard conventional server DIMM slots, with 512 Gbytes (2^{39} bytes) per DIMM slot. On a per slot basis, object memory fabric may keep an additional directory indexed by physical address of the object memory fabric address of each block that has been associated with the corresponding physical address as illustrated in the following diagram.

[0160] FIG. **16** is a block diagram illustrating an example physical memory organization **1600**, in accordance with certain embodiments of the present disclosure. A physical memory directory **1605** for physical memory **1630** may include: object memory fabric object block address **1610**; object size **1615**; reference count **1620**; a modified field **1625** which may indicate whether the block has been modified with respect to persistent memory; and/or write enable **1630** which may indicate whether local block cache state is sufficient for writing. For example, if the cache state were copy, writes may be blocked, and object memory fabric would may with sufficient state for writing. The physical address range may be assigned to each by system BIOS on boot based object memory fabric DIMM SPD (Serial Presence Detect) configuration.

[0161] FIG. **17**A is a block diagram illustrating an example object addressing **1700**, in accordance with certain embodiments of the present disclosure. FIG. **17**B is a block diagram illustrating example aspects of object memory fabric pointer and block addressing **1750**, in accordance with certain embodiments of the present disclosure. Object memory fabric objects **1705** may include object data **1710** and metadata **1715**, both divided into 4k blocks in some embodiments as one unit of storage allocation, referenced by the object memory fabric address space **1720**. The object starting address may be the ObjectID **1755**. Data **1710** may be accessed as a positive offset from ObjectID **1755**. The largest offset may be based on ObjectSize **1760**.

[0162] Object metadata **1715** may be accessed as a negative offset from ObjectStart **1725** (ObjectID). Metadata **1715** can be also referenced by an object memory fabric address in the top 1/16th of object address space **1720**. The start of a specific objects metadata may be 2^{128}-2^{124}+ObjStart/16. This arrangement may enable the POIT to compactly represent metadata **1715** and the metadata **1715** to have an object address space so it can be managed coherently just like data. Although the full object address space may be allocated for object data **1710** and metadata **1715**, storage may be sparsely allocated on a block basis. At a minimum, an object **1705** has a single block of storage allocated for the first block of metadata **1715**, in some embodiments. Object access privilege may be determined through object memory fabric Filesystem ACL or the like. Since object memory fabric manages objects in units of 4k blocks, addressing within the object memory fabric object memory are block addresses, called Block Object Address **1765** (BOA), which corresponds to object

address space [127:12]. BOA [11:0] may be utilized by the object memory for Obj ectSize (BOA[7:0]) and object metadata indication (BOA[2:0])

[0163] FIG. **18** is a block diagram illustrating example aspects **1800** of object metadata **1805**, in accordance with certain embodiments of the present disclosure. Table 6 below indicates metadata of the first block **1810** of metadata **1805** per certain embodiments. In some embodiments, the first block **1810** of metadata **1805** may hold metadata for an object as depicted.

TABLE 6

Metadata First Block

Name	Description	Size
Object address space	Object ID. Number of significant bits determined by object size	16
Object size	Object Size	
CRC	Reserved for optional object crc	16
Parity pointer	Pointer to pages used for optional object block parity	16
Compression Flags	OID of compression object	16
Encryption Flags	OID of encryption object	16
System Defined	Reserved for software defined OS functions	256
Application Defined	Reserved for software defined owning application functions	256
Others		432
Remote Object Table	Specifies Objects accessible from this object. Specifies 64 OIDs (128 bit). The zero entry is used to specify object or metadata within this	1024
Triggers	Triggers or Trigger B-Tree root	2048
		4096

[0164] System-defined metadata may include any Linux-related data to coordinate use of certain objects seamlessly across servers. Application-defined metadata may include application related data from a file system or database storage manager to enable searches and/or relationships between objects that are managed by the application.

[0165] For an object with a small number of triggers, base triggers may be stored within the first block; otherwise, a trigger B-tree root may reference metadata expansion area for the corresponding object. Trigger B-tree leaf may specify base triggers. A base trigger may be a single trigger action. When greater than a single action is required, a trigger program may be invoked. When trigger programs are invoked, they may reside in the expansion area. The remote object table may specify objects that are accessible from this object by the extended instruction set.

[0166] Certain embodiments may provide for an extended instruction execution model. One goal of the extended execution model may be to provide a lightweight dynamic mechanism to provide memory and execution parallelism. The dynamic mechanism enables a dataflow method of execution that enables a high degree of parallelism combined with tolerance of variation in access delay of portion of objects. Work may be accomplished based on the actual dependencies, not a single access delay holding up the computation.

[0167] Various embodiments may include one or a combination of the following. Loads and memory references may be split transactions, with separate request and response so that the thread and memory path are not utilized during the entire transaction. Each thread and execution unit may be able to issue multiple loads into object memory fabric (local and remote) prior to receiving a response. Object memory fabric may be a pipeline to handle multiple requests and responses from multiple sources so that memory resources can be fully utilized. The execution unit may be able to accept responses in a different order from that the requests were issued. Execution units can switch to different threads to be fully utilized. Object memory fabric can implement policies to dynamically determine when to move objects or portions of objects versus moving a thread versus creating a thread.

[0168] FIG. **19** is a block diagram illustrating aspects of an example micro-thread model **1900**, in accordance with certain embodiments of the present disclosure. A thread may be the basic unit of execution. A thread may be defined at least in part by an instruction pointer (IP) and a frame pointer (FP). The instruction pointer may specify the current instruction that is being executed. The frame pointer may specify the location of the current execution state of the thread.

[0169] A thread can include multiple micro-threads. In the example depicted, the thread **1905** include micro-threads **1906** and **1907**. However, a thread can include greater numbers of micro-threads. The micro-threads of a particular thread may share the same frame pointer but have different instruction pointers. In the example depicted, frame pointers **1905-1** and **1905-2** specify the same location, but instruction pointers **1910** and **1911** specify different instructions.

[0170] One purpose of micro-threads may be to enable data-flow like operation within a thread by enabling multiple asynchronous pending memory operations. Micro-threads may be created by a version of the fork instruction and may be rejoined by the join instruction. The extended instruction set may treat the frame pointer as a top of stack or register set by performing operations on offsets from the frame pointer. Load and store instructions may move data between the frame and the object.

[0171] FIG. **20** is a block diagram illustrating aspects of an example relationship **2000** of code, frame, and object, in accordance with certain embodiments of the present disclosure. Specifically, FIG. **20** illustrates how object data **2005** is referenced through the frame **2010**. The default may be for load and store instructions to reference the object **2005** within local scope. Access to object **2005** beyond local scope can be given in a secure manner by access control and security policies. Once this access is given, objects **2005** within local and non-local scope can be accessed with equal efficiency. Object memory fabric encourages strong security by encouraging efficient object encapsulation. By sharing the frame, micro-threads provide a very lightweight mechanism to achieve dynamic and data-flow memory and execution parallelism, for example, on the order of 10-20 micro-threads or more. The multiple threads enable virtually unlimited memory based parallelism.

[0172] FIG. **21** is a block diagram illustrating aspects of an example of micro-thread concurrency **2100**, in accordance with certain embodiments of the present disclosure. Specifically, FIG. **21** illustrates the parallel data-flow concurrency for a simple example of summing several randomly located values. A serial version **2105** and a parallel version **2110** are juxtaposed, in accordance with certain embodiments of the present disclosure. The parallel version **2110** can be almost n times faster since loads are overlapped in parallel.

[0173] Referring again to FIG. **20**, the approach can be extended to interactive and recursive approaches in a dynamic manner. The advantages of prefetching ahead can now be achieved in cases with minimal locality without using

prefetch. When an object is created, a single default thread **2015** (single micro-thread **2020** is created) may be waiting to start with a start message to the default thread **2015**. The default thread **2015** then can create micro-threads with the thread or use a version of the fork instruction to create a new thread.

[0174] In some embodiments, both the instruction pointer and the frame pointer may be restricted to the expansion metadata region **1815** starting at block two and extending to SegSize/16. As the number of objects, object size, and object capacity increase, the thread and micro-thread parallelism may increase. Since threads and micro-threads may be tied to objects, as objects move and distribute so may the threads and micro-threads. Embodiments of object memory fabric may have the dynamic choice of moving objects or portions of objects to threads or distributing threads to the object(s). This may be facilitated by the encapsulated object methods implemented by the extended execution model.

[0175] In the foregoing description, for the purposes of illustration, methods were described in a particular order. It should be appreciated that in alternate embodiments, the methods may be performed in a different order than that described. It should also be appreciated that the methods described above may be performed by hardware components or may be embodied in sequences of machine-executable instructions, which may be used to cause a machine, such as a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the methods. These machine-executable instructions may be stored on one or more machine readable mediums, such as CD-ROMs or other type of optical disks, floppy diskettes, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, flash memory, or other types of machine-readable mediums suitable for storing electronic instructions. Alternatively, the methods may be performed by a combination of hardware and software.

[0176] While illustrative and presently preferred embodiments of the invention have been described in detail herein, it is to be understood that the inventive concepts may be otherwise variously embodied and employed, and that the appended claims are intended to be construed to include such variations, except as limited by the prior art.

What is claimed is:

1. An object memory fabric comprising:

a plurality of object memory modules, each object memory module comprising object storage storing one or more memory objects, memory object meta-data, and a memory module object directory, wherein:

each memory object and/or portion of memory objects is created natively within the object memory module and is a managed by the object memory module at a memory layer; and

the memory module object directory indexes all memory objects and/or portions of memory objects within the object memory module; and

a hierarchy of object routers communicatively coupling the plurality of object memory modules, where:

each object router of the hierarchy of object routers comprises a router object directory, wherein the router object directory indexes all memory objects and/or portions of memory objects contained in object memory modules below the object router along a line of descent in the hierarchy stemming from the object router; and

the hierarchy of object routers, based at least in part on the router object directories, is adapted to behave in aggregate as a single object directory communicatively coupled to all object memory modules and to process requests based at least in part on the router object directories.

2. The object memory fabric of claim **1**, wherein the hierarchy of object routers operates according to a hierarchical tree network.

3. The object memory fabric of claim **2**, wherein the object memory modules below the object router along the line of descent in the hierarchy stemming from the object router comprises object memory modules directly communicatively coupled to the respective object router toward a leaf of the hierarchical tree network.

4. The object memory fabric of claim **3**, wherein toward the leaf of the hierarchical tree network corresponds to a most direct path between the object router away from a root of the hierarchical tree network and an object memory module at the leaf of the hierarchical tree network.

5. The object memory fabric of claim **1**, wherein the behaving in aggregate as the single object directory communicatively coupled to all the object memory modules and the processing the requests comprises:

responsive to each request of the requests, at least one of the object routers looking up an object corresponding to the request with a respective router object directory, and:

consequent to identifying a reference to the object in the respective router object directory, forwarding the object toward a leaf in the hierarchy;

consequent to identifying the reference to the object is not in the respective router object directory, forwarding the first request toward a root in the hierarchy.

6. The object memory fabric of claim **1**, wherein at least one of the requests is received from an application layer.

7. The object memory fabric of claim **1**, wherein each object memory module further comprises an object index tree that indexes local memory objects.

8. The object memory fabric of claim **7**, wherein the object index tree comprises node blocks and leaf blocks each differentiated by a type attribute, wherein one or more of the leaf blocks point to locations of the local memory objects in persistent memory and/or faster memory.

9. The object memory fabric of claim **7**, wherein the object index tree comprises a pointer to a per object index tree for each local memory object, where the per object index tree for each local memory object indexes, on a block basis, memory object data and memory object meta-data locally present for the local memory object.

10. The object memory fabric of claim **9**, wherein the per object index tree comprises node blocks and leaf blocks each differentiated by a type attribute, wherein one or more of the leaf blocks point to locations of the memory object data and memory object meta-data in persistent memory and/or faster memory.

11. The object memory fabric of claim **1**, wherein the hierarchy of object routers adapted to behave in aggregate as the single object directory is used to manage at least one memory object across multiple object memory modules of the plurality of object memory modules where a single object memory module of the multiple object memory modules does not have storage space required to store all blocks of the at least one memory object.

12. A hardware-based processing node comprising:
an object memory module comprising object storage storing one or more memory objects, memory object meta-data, and a memory module object directory, wherein:
each memory object and/or portion of memory objects is created natively within the object memory module and is a managed by the object memory module at a memory layer; and
the memory module object directory indexes all memory objects and/or portions of memory objects within the object memory module; and
the object memory module to process one or more requests based at least in part on the one or more object directories.

13. The hardware-based processing node of claim **12**, wherein the processing the one or more requests comprises:
processing an object identifier corresponding to a first request of the one or more requests;
determining whether at least one object of the one or more memory objects corresponds to the object identifier; and
consequent to a determination that at least one of the one or more memory objects does correspond to the object identifier, generating a response to the first request based at least in part on the at least one object.

14. The hardware-based processing node of claim **13**, further comprising:
an object router communicatively coupled to the object memory module that, consequent to a determination that at least one of the one or more memory objects does not correspond to the object identifier, routes the first request to an additional node.

15. The hardware-based processing node of claim **14**, wherein the routing the first request to the additional node is based at least in part on the object router determining a location of a memory object corresponding to the object identifier.

16. The hardware-based processing node of claim **14**, wherein:
the routing the first request to the additional node is based at least in part on the object router determining that a location of a memory object corresponding to the object identifier is unknown; and
the first request is directed toward a root node.

17. The hardware-based processing node of claim **14**, after the routing the first request to the additional node, the memory module generates a response to the first request based at least in part on a received response from the additional node.

18. The hardware-based processing node of claim **12**, wherein the object memory module further comprises an object index tree that indexes local memory objects.

19. The hardware-based processing node of claim **18**, wherein the object index tree comprises node blocks and leaf blocks each differentiated by a type attribute, wherein one or more of the leaf blocks point to locations of the local memory objects in persistent memory and/or faster memory.

20. The hardware-based processing node of claim **18**, wherein the object index tree comprises a pointer to a per object index tree for each local memory object, where the per object index tree for each local memory object indexes, on a block basis, memory object data and memory object meta-data locally present for the local memory object.

21. The hardware-based processing node of claim **20**, wherein the per object index tree comprises node blocks and leaf blocks each differentiated by a type attribute, wherein one or more of the leaf blocks point to locations of the memory object data and memory object meta-data in persistent memory and/or faster memory.

22. The hardware-based processing node of claim **12**, wherein at least one of the one or more requests is received from an application layer.

23. The hardware-based processing node of claim **12**, wherein at least one of the one or more requests is received from one or more additional nodes.

24. The hardware-based processing node of claim **12**, wherein the node is configured to utilize a set of algorithms to operatively couple to one or more additional nodes to operate as a set of nodes independently of a scale of the set of nodes, wherein the set of nodes operates so that all memory objects of the set of nodes are accessible by any node of the set of nodes.

25. A method for storing and managing one or more memory objects in an object memory fabric, the method comprising:
storing one or more memory objects, memory object meta-data, and a memory module object directory in object storage of an object memory module, wherein:
each memory object and/or portion of memory objects is created natively within the object memory module and is managed by the object memory module at a memory layer; and
the memory module object directory indexes all memory objects and/or portions of memory objects within the object memory module; and
processing by the object memory module one or more requests based at least in part on the one or more object directories.

26. The method of claim **25**, wherein the processing the one or more requests comprises:
processing an object identifier corresponding to a first request of the one or more requests;
determining whether at least one object of the one or more memory objects corresponds to the object identifier; and
consequent to a determination that at least one of the one or more memory objects does correspond to the object identifier, generating a response to the first request based at least in part on the at least one object.

27. The method of claim **26**, wherein the processing the one or more requests comprises:
consequent to a determination that at least one of the one or more memory objects does not correspond to the object identifier, routing, by an object router communicatively coupled to the object memory module, the first request to an additional node.

* * * * *