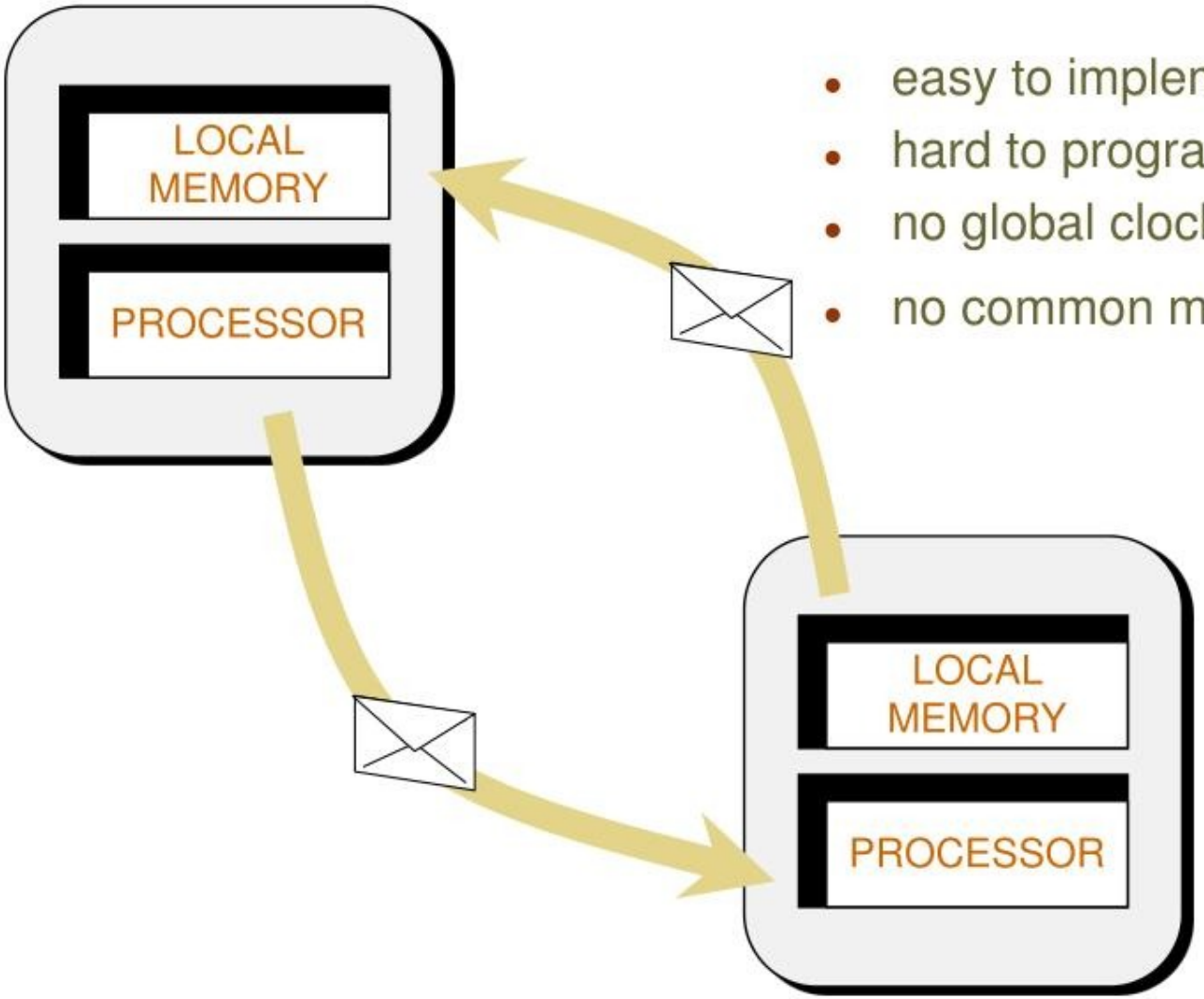
Distributed Shared Memory (DSM)

Distributed Shared Memory

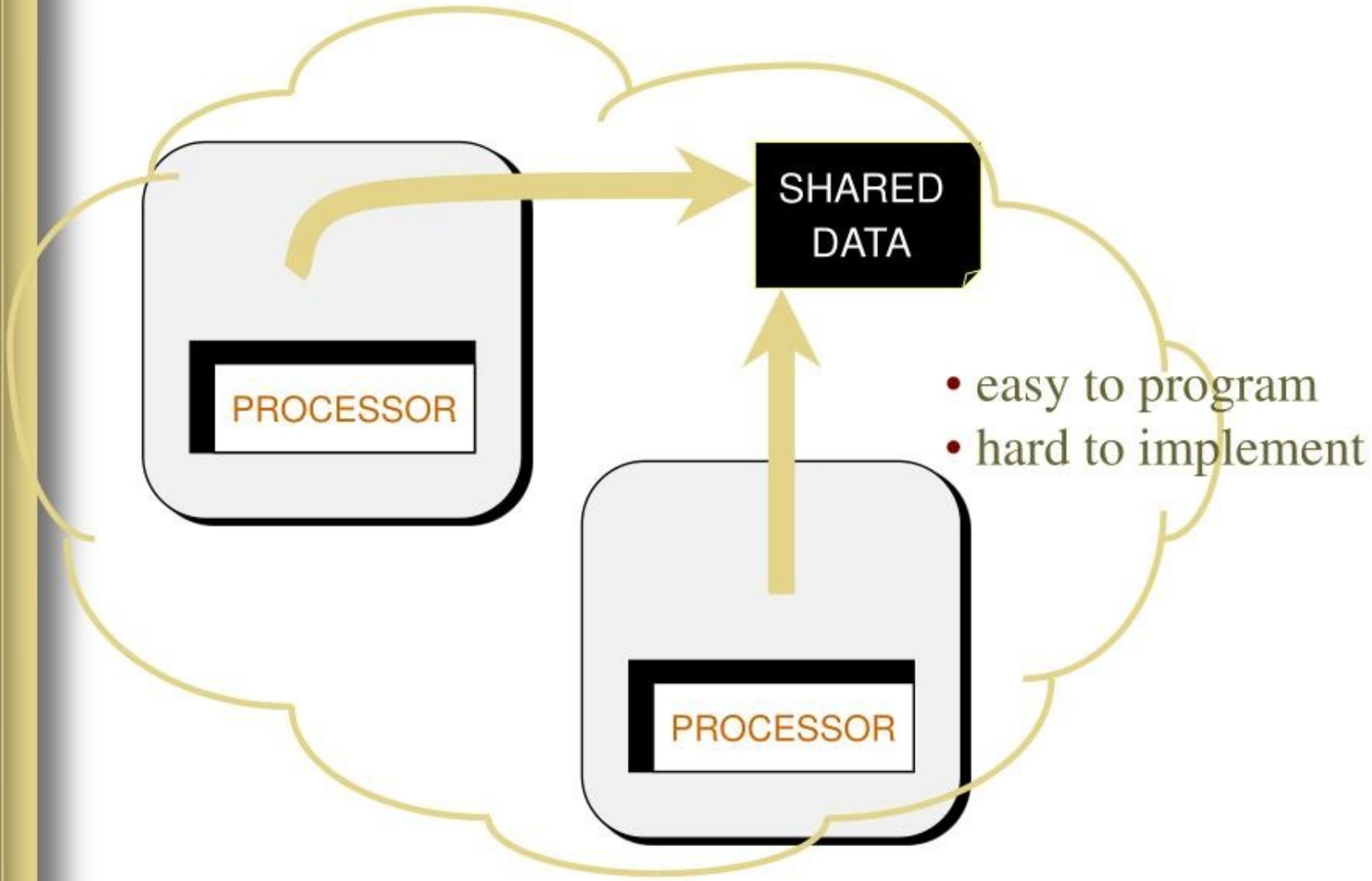
- Message Passing Paradigm
  - Send (recipient, data)
  - Receive (data)
- Shared Memory Paradigm
  - Data= Read (address)
  - Write (address, data)

Message Passing



- easy to implement
- hard to program
- no global clock
- no common memory

Distributed Shared Memory



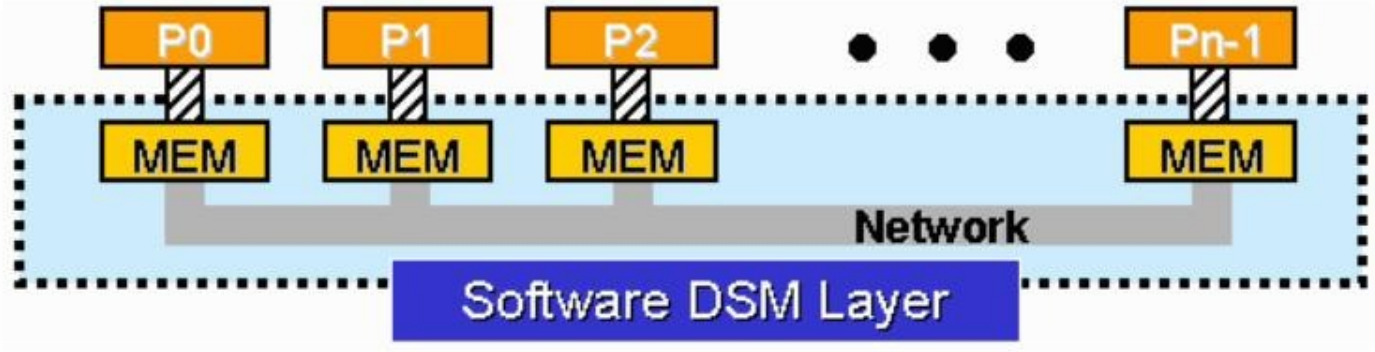
Distributed Shared Memory

- For loosely coupled distributed systems, no physically shared memory is available.
- A software layer is implemented on top of the message passing communication system in loosely coupled distributed-memory system to provide a shared memory abstraction to the programmers.
- It provides a virtual address space among processes on loosely coupled processors.
- Implemented either in an operating system or in runtime library routines with proper kernel support.

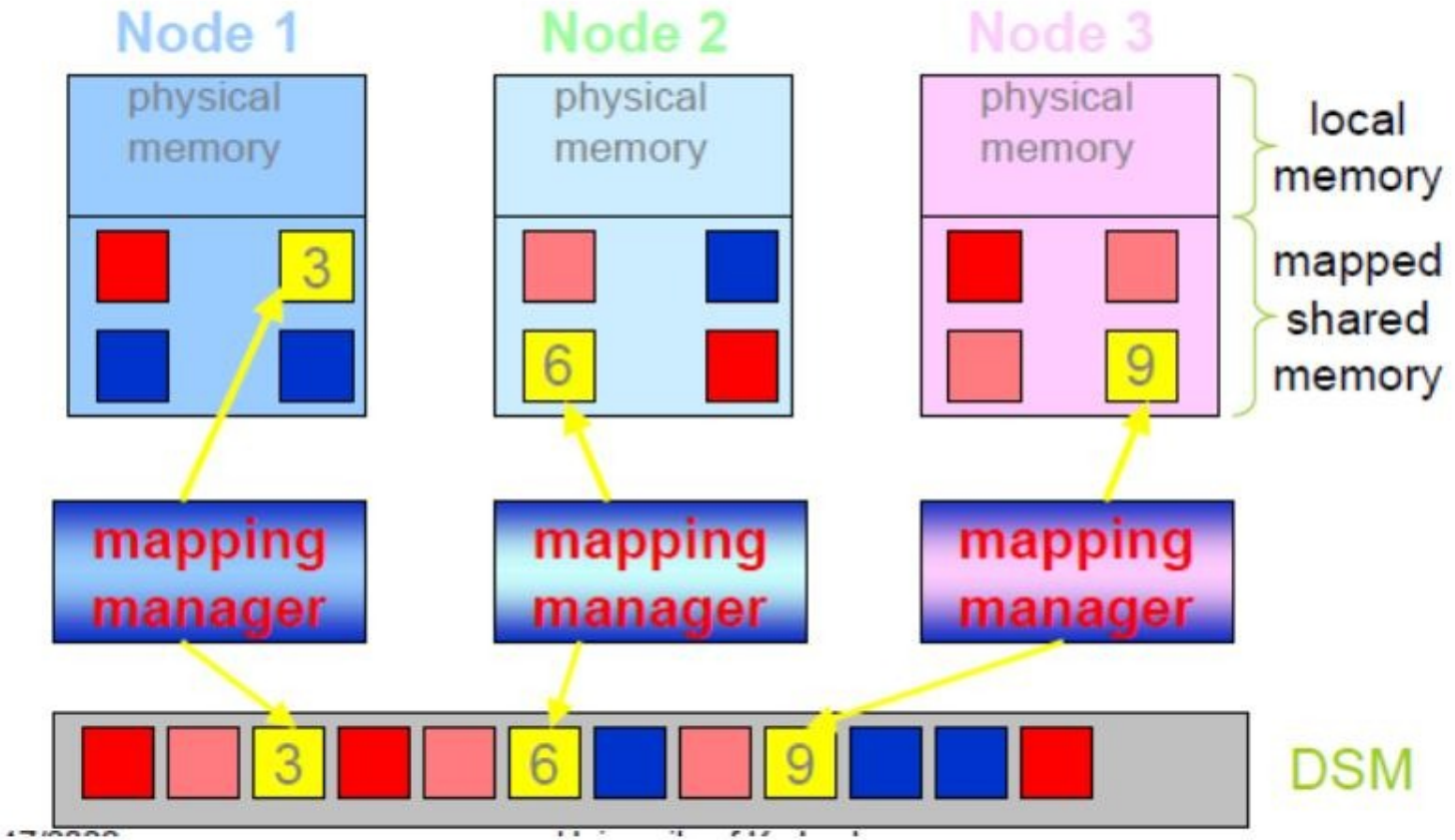
Architecture of DSM Systems

- DSM is an abstraction that integrates the local memory of different machines in a network environment into a single logical entity shared by all co-operating processes executing on multiple sites.

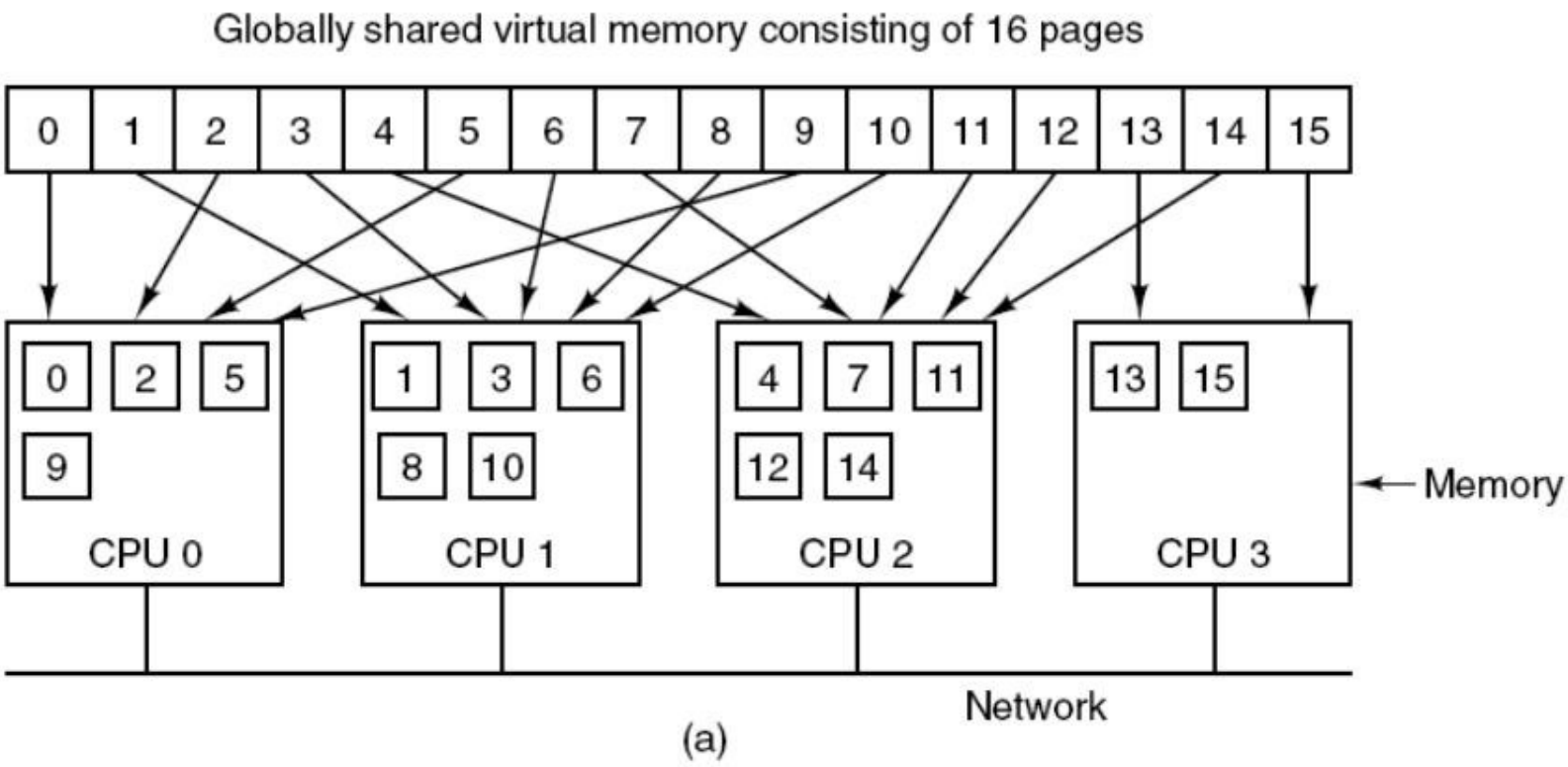
- Also known as Distributed Shared Virtual memory (DSVM).



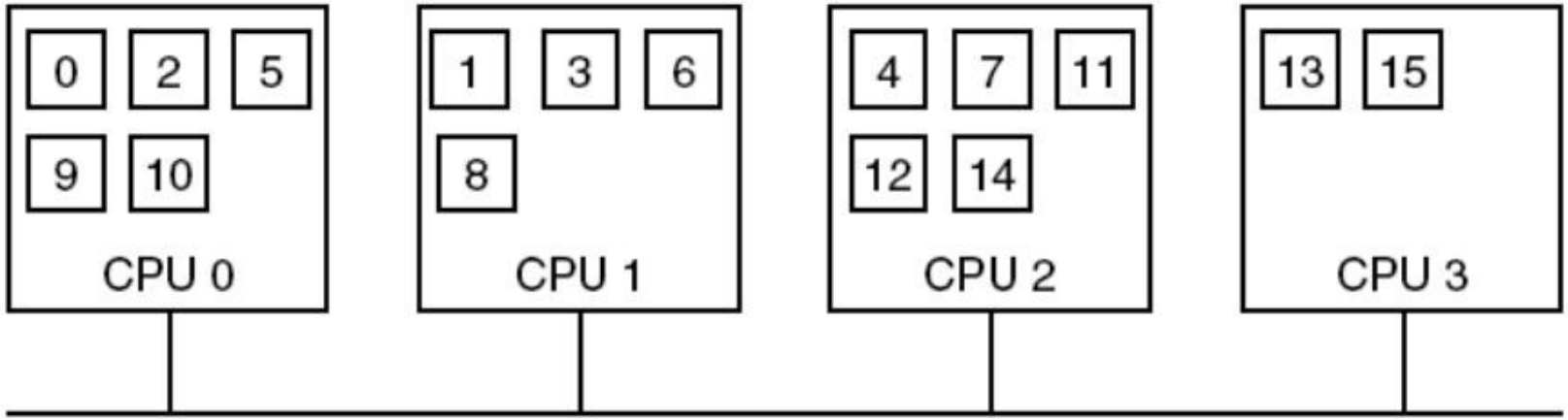
DSM Systems


Node 1
Node 2
Node 3
physical memory
physical memory
physical memory
local memory
mapped shared memory
3
6
9
mapping manager
mapping manager
mapping manager
3
6
9
DSM

- Each node of the system consists of one or more CPU and a memory unit. Main memory of each node is used to cache pieces of the shared memory space.
- A software memory-mapping manager routine in each node maps the local memory on to a shared virtual memory.
- When process node makes a request , memory mapping manager takes charge of it. If required data block is not present in local memory, network page fault is generated & control is passed to operating system. The missing block is migrated from remote node to client's node & is mapped into application's address space.
- Data caching is used to reduce the network latency.
- The basic unit of data transfer or caching is a memory block.
- High degree of locality of data accesses reduces network traffic.

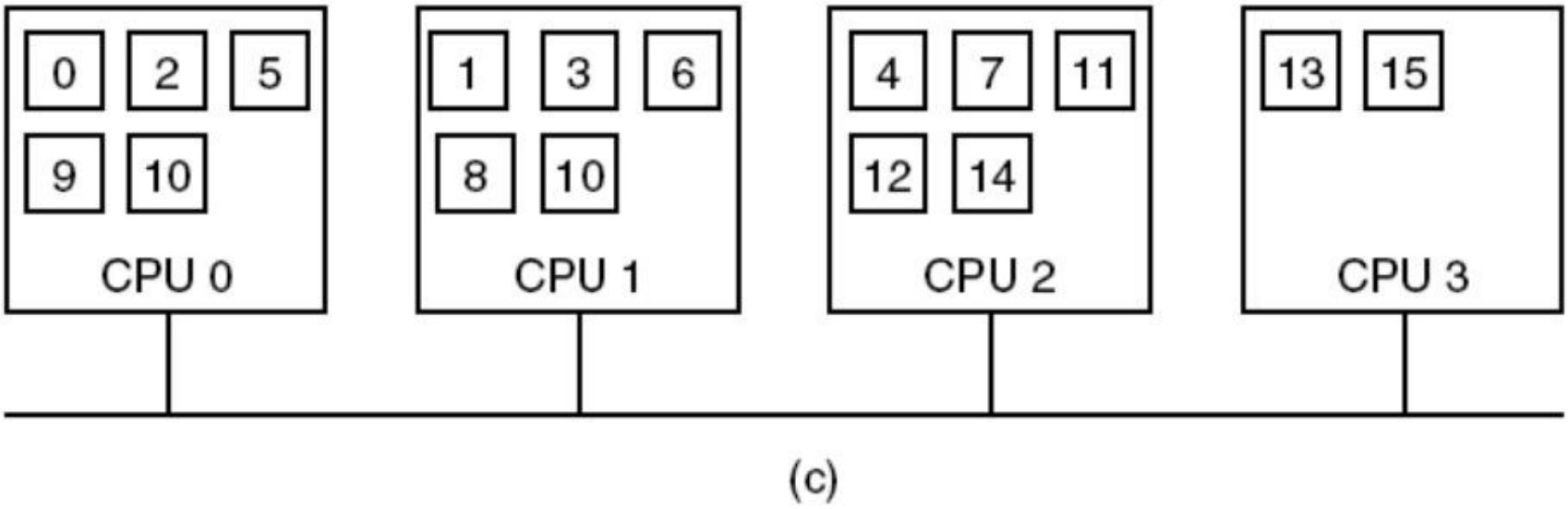


Pages of the address space distributed among four machines.



(b)

CPU 1 references page 10 and the page is moved there.



(c)

Page 10 is read only and replication is used.

Advantages of DSM

- Simpler abstraction
  - Shields programmers from low level concerns
- Better portability of distributed application program
  - Distributed application programs written for shared memory processor can be executed on DSM system without any change.
- Better performance of some application
  - Locality of data
  - Ongoing On-demand data movement rather than separate data-exchange phase
  - Larger memory space

- Flexible communication environment
  - Co existence of sender & receiver not necessary
  - Does not require recipient information

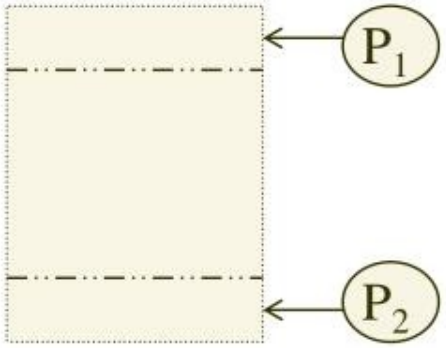
- Ease of process management
  - Migrant process can leave its address space on its old node at the time of migration & fetch required pages from its new node on demand at time of accessing.

Design and Implementation Issue of DSM

- Granularity
  - Block size
- Structure of shared memory space
- Memory coherence and access synchronization
  - Consistency of a shared data lying in main memories of two or more nodes & dealing with concurrent accesses
- Data location and access
- Replacement strategy
- Thrashing
  - Data block gets transferred back & forth at a high rate if two nodes compete for write access to it.
- Heterogeneity

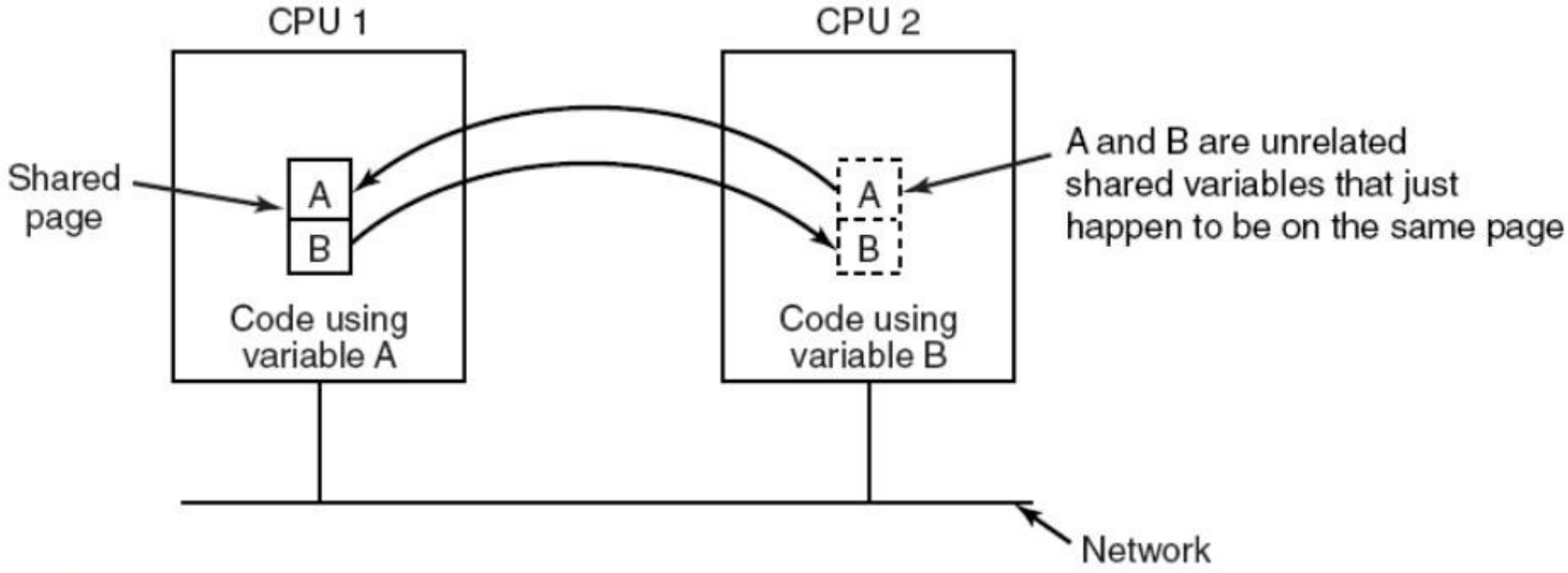
Granularity

- Factors influencing block size selection
  - Paging overhead
    - Less for large block size because of locality of reference
  - Directory size
    - Larger block size, smaller directory
  - Thrashing
    - More in larger blocks, as data in same data block may be updated by multiple nodes at same time
  - False sharing
    - More in larger blocks, when two processes access two unrelated variables that reside in same data block



False Sharing

False Sharing



- Use page size as block size
  - Allows the use of existing page-fault scheme.
  - Allows access right control to be integrated into memory management unit.
  - If page can fit into packet, it does not impose undue communication overhead.
  - Suitable data entity for memory contention.

Structure of Shared Memory Space

- Structure defines the abstract view of the shared – memory space to be presented to the application programmers of a DSM system.

- No structuring
  - Linear array of words
  - Can choose fixed grain size for all applications
  - DSM simple

- Structuring by data type
  - Collection of objects or variables in source language
  - Granularity is object or variable
  - DSM complicated as grain size can be variable

- Structuring as a database
  - Ordered as a associative memory called tuple space
  - Requires programmers to use special access functions to access shared memory space, thus, data access is non transparent.

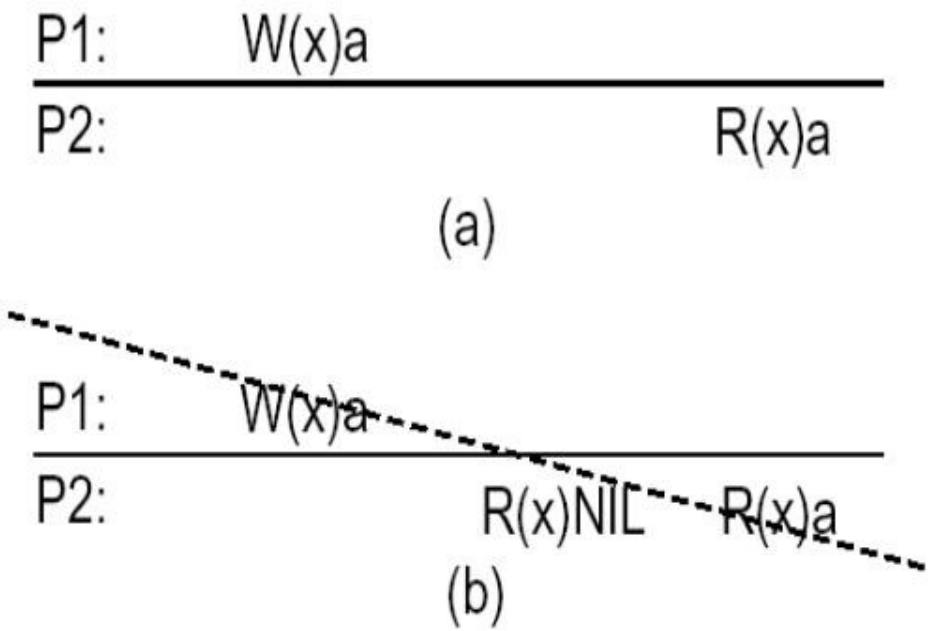
Consistency Models

- To improve performance, DSM systems rely on replicating shared data items and allowing concurrent access at many nodes. However, if the concurrent accesses are not carefully controlled, memory accesses may be executed in an order different from that which the programmer expected.
- A consistency model basically refers to the degree of consistency that has to be maintained for the shared memory data.
- It is defined as a set of rules that applications must obey to ensure consistency of memory.
- Better concurrency can be maintained by relaxing consistency requirement.

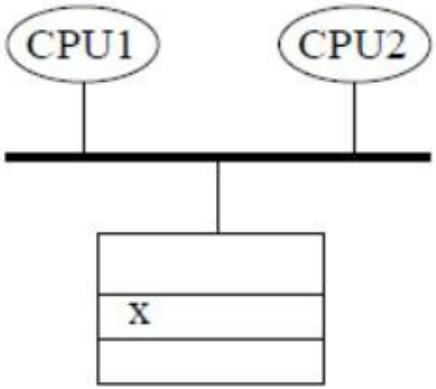
Strict (Atomic) Consistency Model

- Strongest form of memory coherence.
  - Any read to a memory location X returns the value stored by the most recent write operation to X (changes are instantaneous).
- Implementation requires the existence of an absolute global time.
- Possible only on uniprocessor.

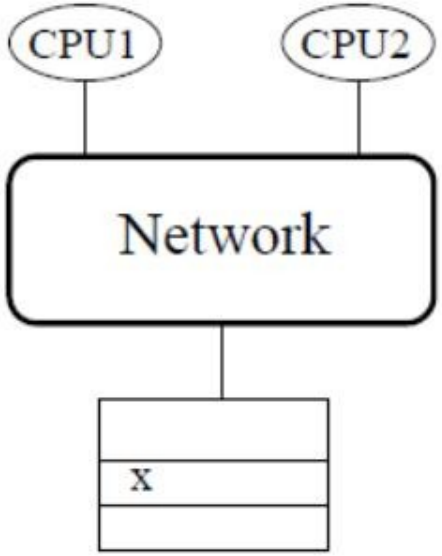
Strict Consistency Example



Strict Consistency



Natural Strict Consistency



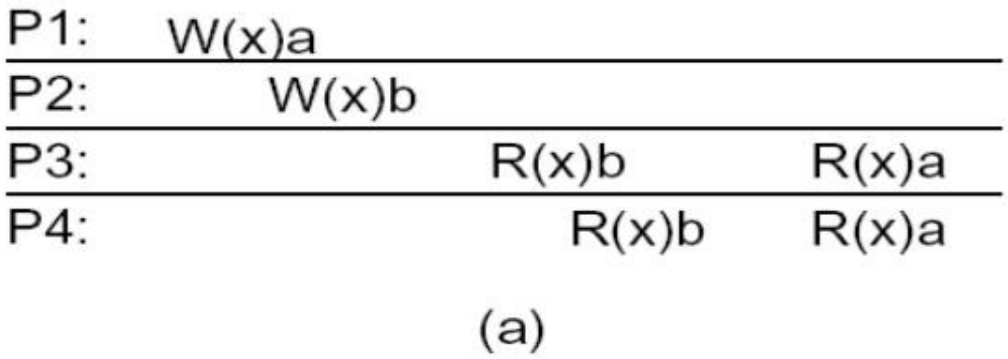
Difficult to Implement as -

- Network latency
- Difficult to determine global time
- No global synchronization

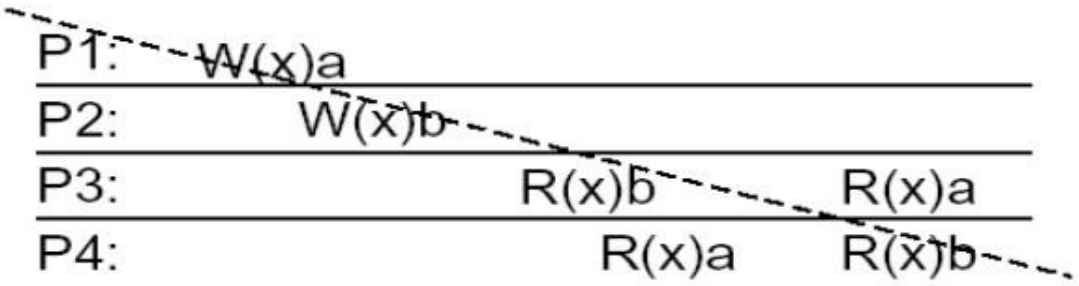
Sequential Consistency Model

- Weaker than strict consistency model.
- Every node of the system sees the operations on the same memory part in the same order, although the order may be different from the order of issuing the operations.
- The exact order in which the memory access operation are interleaved does not matter.
- If three operations r_1, w_1, r_2 are performed on a memory address, any ordering (r_1, w_1, r_2), (r_2, w_1, r_1), (w_1, r_2 , r_1), (r_2, r_1, w_1) is acceptable provided all processors see same ordering.
- Implementation require to ensure that no memory operation is started until all the previous one have been completed.
- Provides single copy semantics as all processes sharing a memory location always see exactly the same contents stored in it.

Sequential Consistency Example



(a)



(b)

Causal Consistency Model

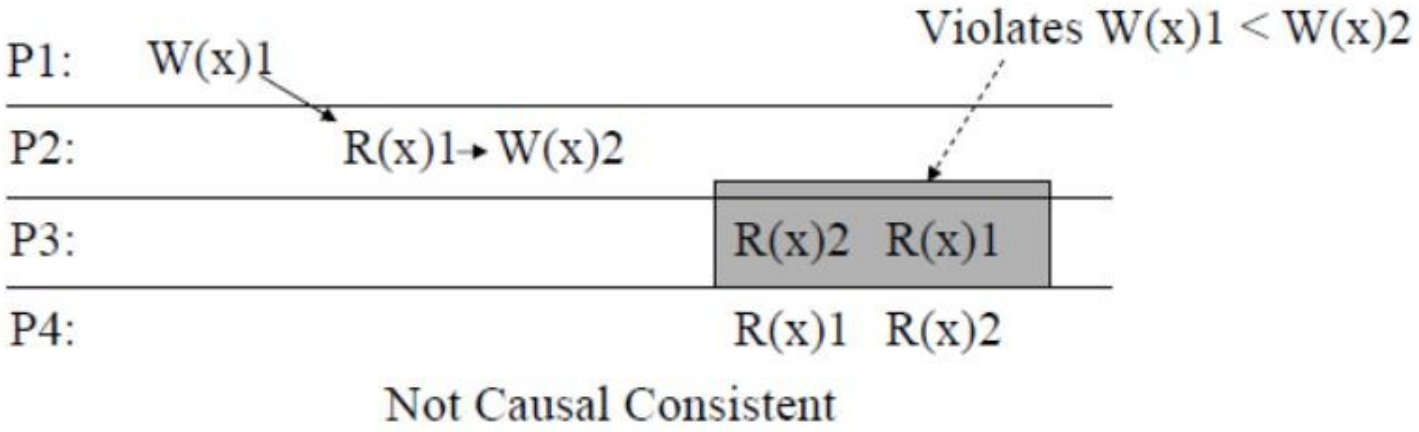
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent (Operations that are not causally related) writes may be seen in a different order on different machines.

- Suppose that process **P1** writes a variable **X**. Then **P2** reads **X** and writes **Y**. Reading of **X** and writing of **Y** are potentially causally related because the computation of **Y** may have depended on the value of **X** read by **P2**.

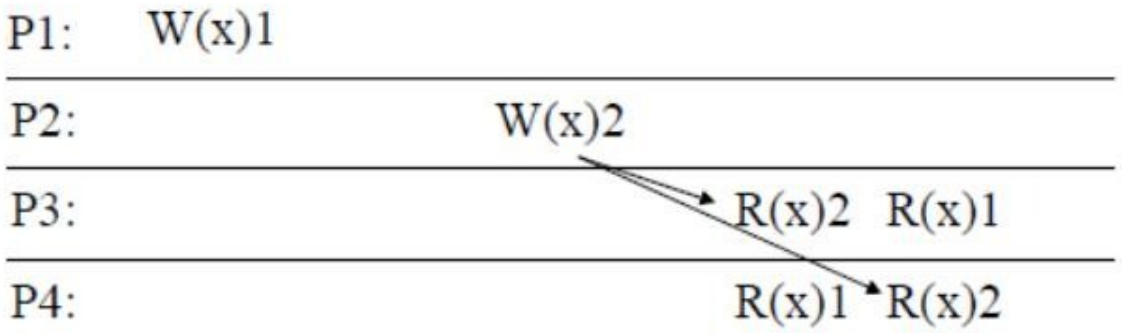
- If a write operation w_1 is casually related to w_2 then (w_1, w_2) is acceptable but not (w_2, w_1) .

- Implementation require to keep track of dependent memory reference using dependency graph.

Causal Consistency Model



Not Causal Consistent

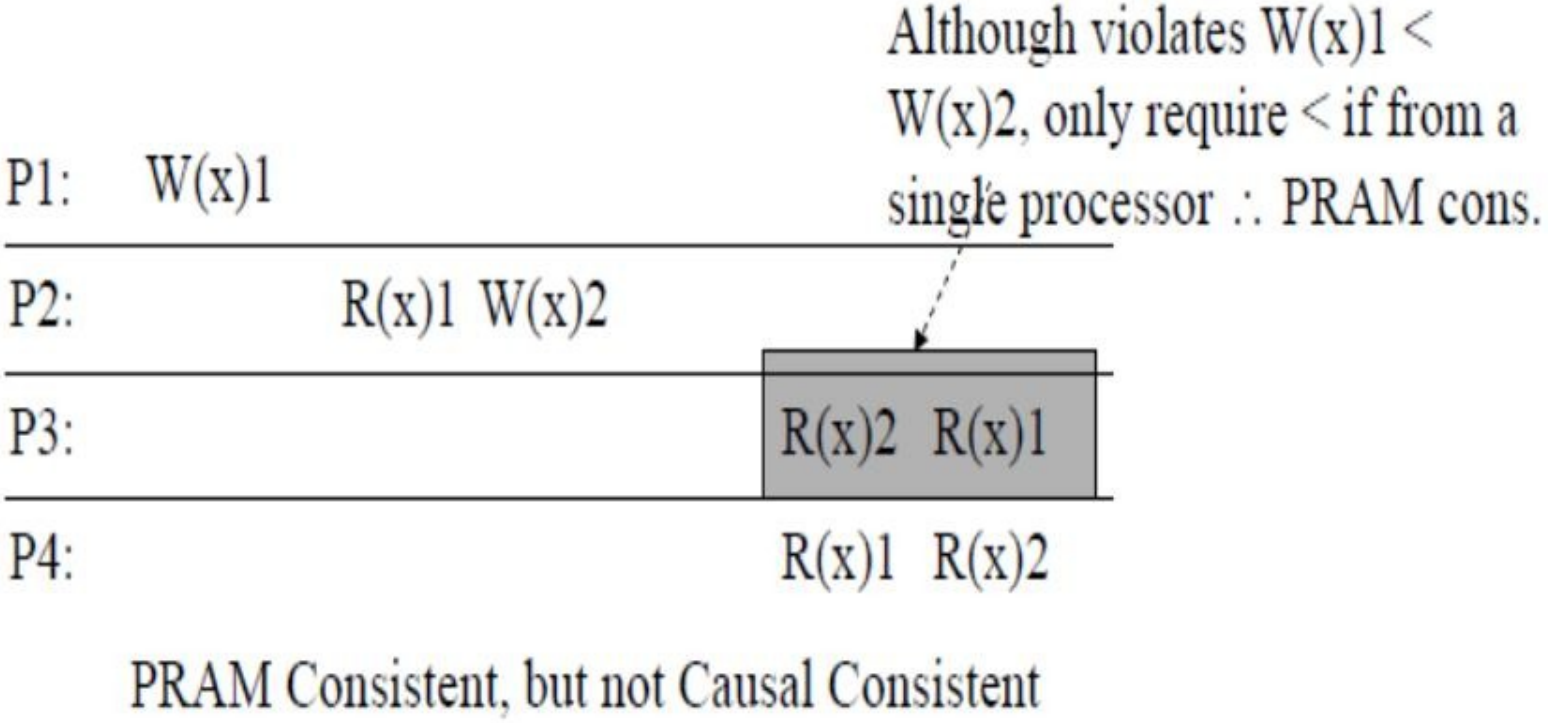


Causal Consistent

Pipelined Random Access Memory (PRAM) Consistency Model

- Also known as FIFO consistency.
- Writes done by a single process are received by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.
- Ex. Process P1 executes w_{11} & w_{12}
  Process P2 executes w_{21} & w_{22}
  P3 sees it as $(w_{11}, w_{12})(w_{21}, w_{22})$
  P4 sees it as $(w_{21}, w_{22})(w_{11}, w_{12})$
- All write operations performed by single process are in pipeline.
- Can be implemented by sequencing the write operation of each node independently.

PRAM Consistency Model



PRAM Consistent, but not Causal Consistent

Processor Consistency Model

- PRAM consistent with additional restriction of memory coherence, i.e. for every memory location x, there be a global agreement about order of writes to x.

- All write operations performed on the same memory location (no matter by which process they are performed) are seen by all processes in the same order.

- Ex. Process P1 executes w_{11} & w_{12}
  Process P2 executes w_{21} & w_{22}
  P3 & P4 both see it as $(w_{11}, w_{12})(w_{21}, w_{22})$ or $(w_{21}, w_{22})(w_{11}, w_{12})$ if they are writes to different memory locations.

Weak Consistency Model

- Based on facts
  - Results of several write operations can be combined & sent to other processes only when they need it. Ex. Critical section. Memory has no way of knowing when a process is in critical section, so it has to propagate all writes to all memories in the usual way.
  - Isolated accesses to shared variables are rare.

- It ensures that consistency is enforced on a group of memory operation rather on individual memory reference operation.

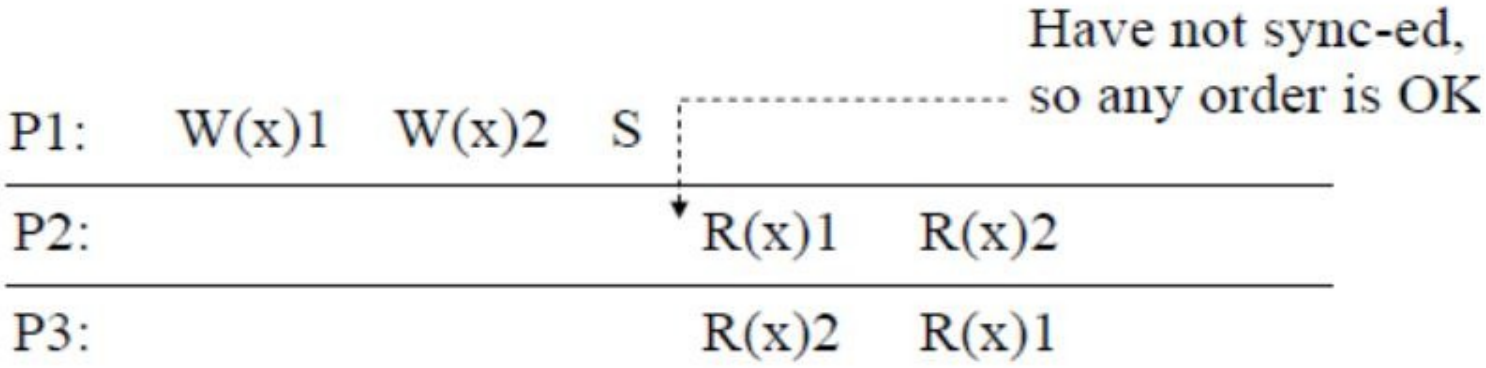
- Responsibility of programmer to decide when to reflect changes in all processes, but better performance.

- Uses synchronization variable. When it is accessed by a process , the entire memory is synchronized by making all changes to the memory made by all processes visible to all other processes.

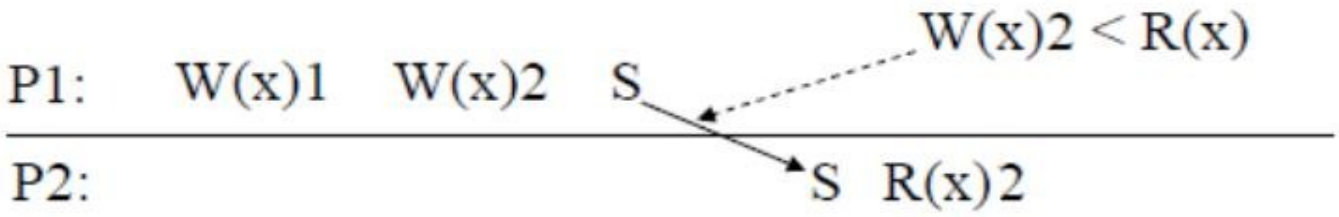
- When a synchronization completes, all writes done on that machine are propagated outward & all writes done on other machine are brought in.

- Meets following requirements:-
  - Accesses to synchronization variables are sequential consistent. Access of a synchronization variable is broadcast, so no other synchronization variable can be accessed in any other process until this one is finished everywhere.
  - All previous write operations must be complete before access to a synchronization variable.
  - All previous accesses to synchronization variables must be complete before access to a non-synchronization variable, so that a process can be sure of getting the most recent values.

Weak Consistency Example



Weak Consistent Example



Weak Consistent

Release Consistency Model

- Process exits critical section
  - All changes made to the memory by the process are propagated to other nodes.
- Process enters critical section
  - All changes made to the memory by other processes are propagated from other nodes to the process's node.

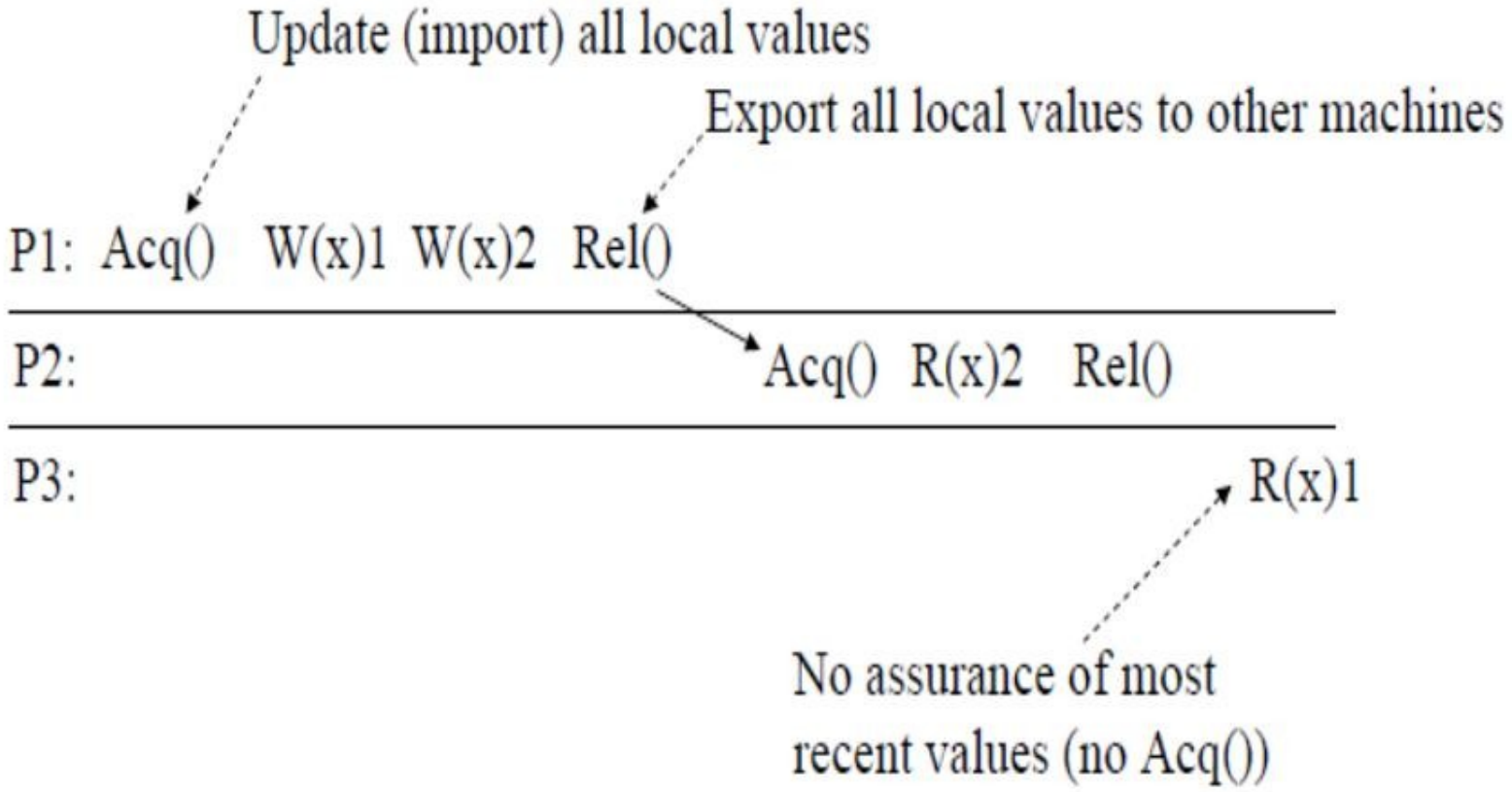
- Uses two synchronization variables
  - Acquire used to enter critical section.
  - Release used to exit critical section.

- Acquires & releases on different locks occur independently of one another.

- Can achieved by using barrier mechanism also. A barrier is a synchronization mechanism that prevents any process from starting phase n+1 of a program until all processes have finished phase n.

- When a process arrives at a barrier, it must wait until all other processes get there too. When the last process arrives, all shared variables are synchronized & then all processes are resumed.

Release Consistency Example



- Requirements to be met:-
  - All previous acquires done by the process must have completed successfully before an access to a shared variable is performed.
  - All previous reads and writes done by the process must have completed before a release is allowed to be performed.
  - The acquire and release accesses must be processor consistent (sequential consistency is not required).

- Also known as eager release consistency.

Lazy Release Consistency Model

- Modifications are not sent to other nodes at the time of release.
- When a process does an acquire, all modifications of other nodes are acquired by the process's node, thus getting the most recent values.
- No network traffic generated until another process does acquire.
- Beneficial especially when critical region is located inside loop.

Strong Const. models

Weak const. models

Model	Description
Strict	Absolute time ordering of all shared accesses
Seq.	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited

Models do not use synch. operations

Models use synch. operations

Summary of Consistency Models

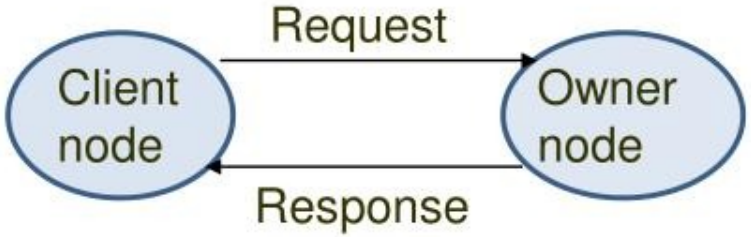
- Strict consistency not practical.
- Sequential consistency preferred, but suffers low concurrency.
- Casual, PRAM, Processor consistency do not provide memory coherence because different processes see different sequence of operations.
- Weak & release consistency provide better concurrency but impose some burden on programmers.

Implementing Sequential Consistency Model

- Nonreplicated , nonmigrating blocks (NRNMBs)
- Nonreplicated , migrating blocks (NRMBs)
- Replicated , migrating blocks (RMBs)
- Replicated , nonmigrating blocks (RNMBs)

Nonreplicated , Nonmigrating Blocks

- Simplest strategy.
- Each block of the shared memory has a single copy whose location is always fixed.
- All access request to a block from any node are sent to the owner node of the block, which has only one copy.

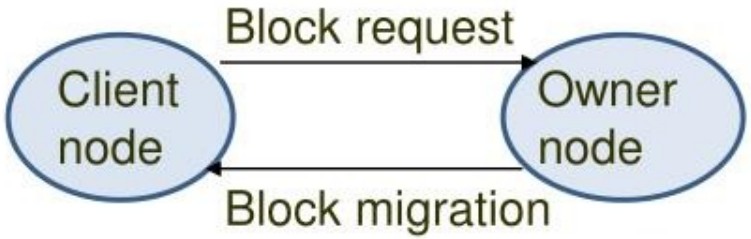


- Drawbacks
  - Serializing data access creates a bottle neck
  - Parallelism is not possible

- Data Locating uses mapping function
  - There is a singe copy of each block in the system
  - The location of the block never changes
  - Use mapping function to find block

Nonreplicated, Migrating Blocks

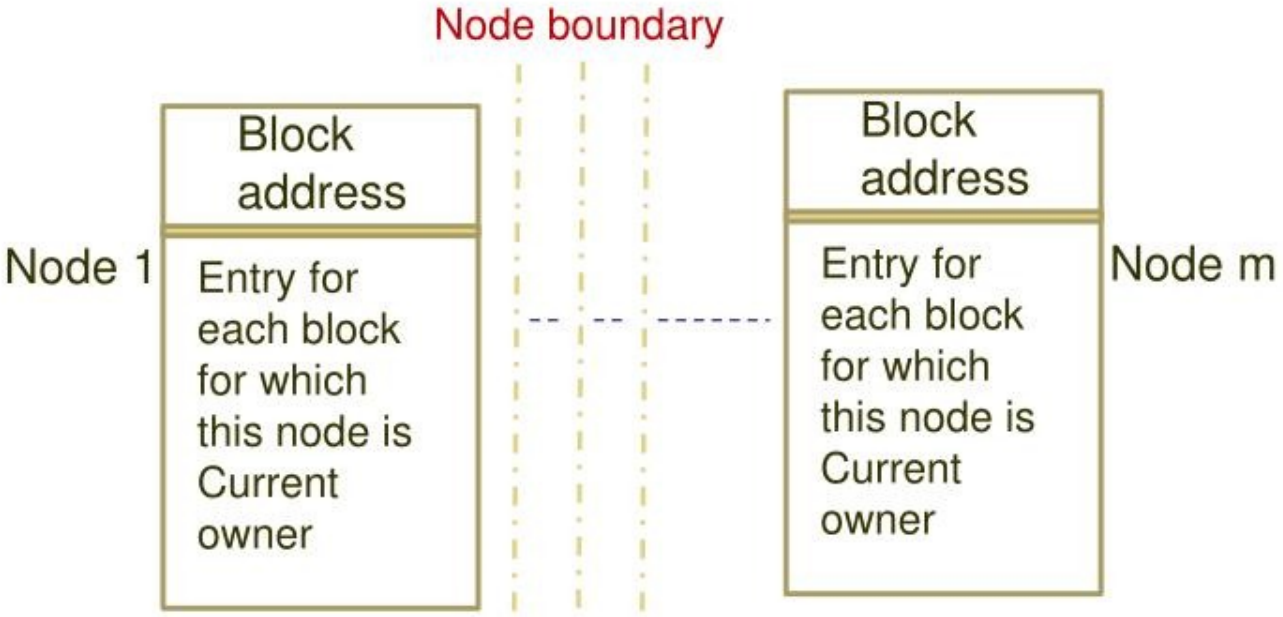
- Each block of the shared memory has a single copy in the entire system.
- Each access to a block causes the block to migrate from its current node to the node from where it is accessed, thus changing its owner.
- At a given time data can be accessed only by processes of current owner node. Ensures sequential consistency.



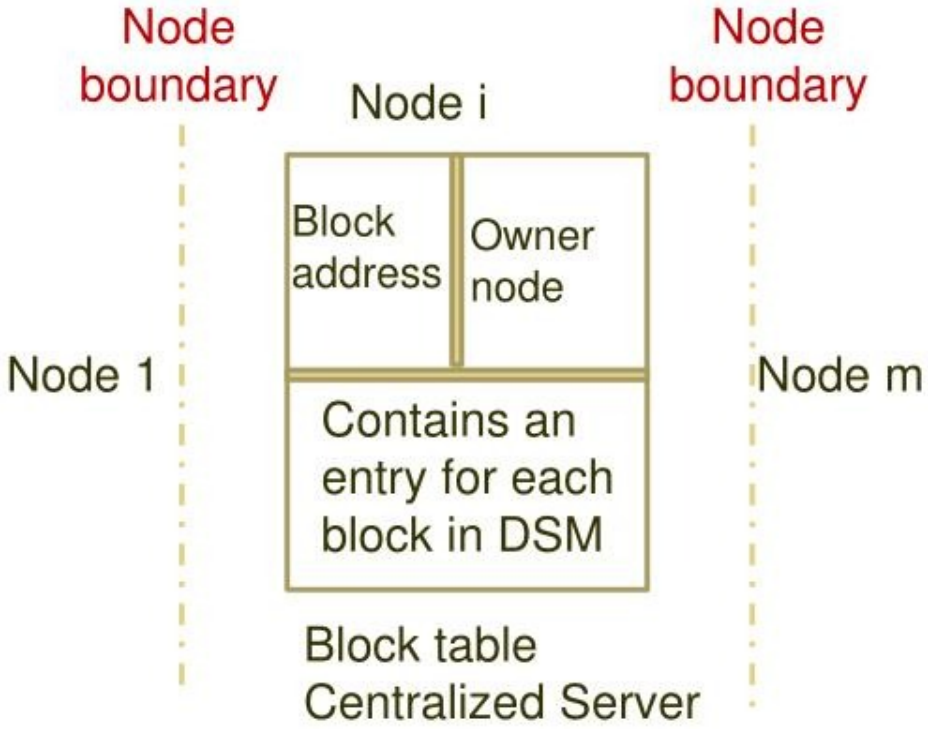
- Advantages
  - Data located locally so no communication cost
  - High locality of reference so cost of multiple accesses reduced
- Disadvantages
  - Prone to thrashing
  - Parallelism not possible

Data Locating

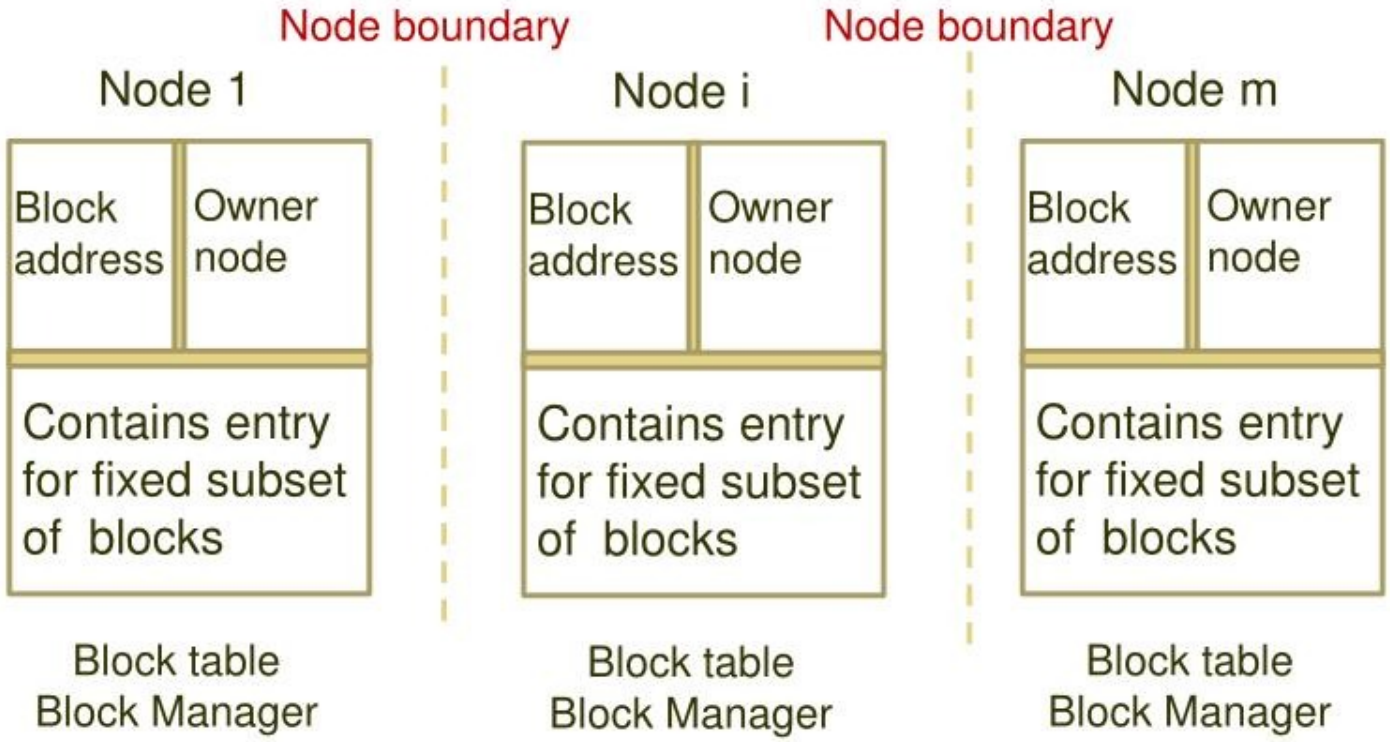
- Broadcasting
  - Fault handler of faulting node broadcasts a read/ write request on network, to which the current owner responds by sending the block.
  - All nodes must process broadcast request – communication bottleneck
  - Network latency



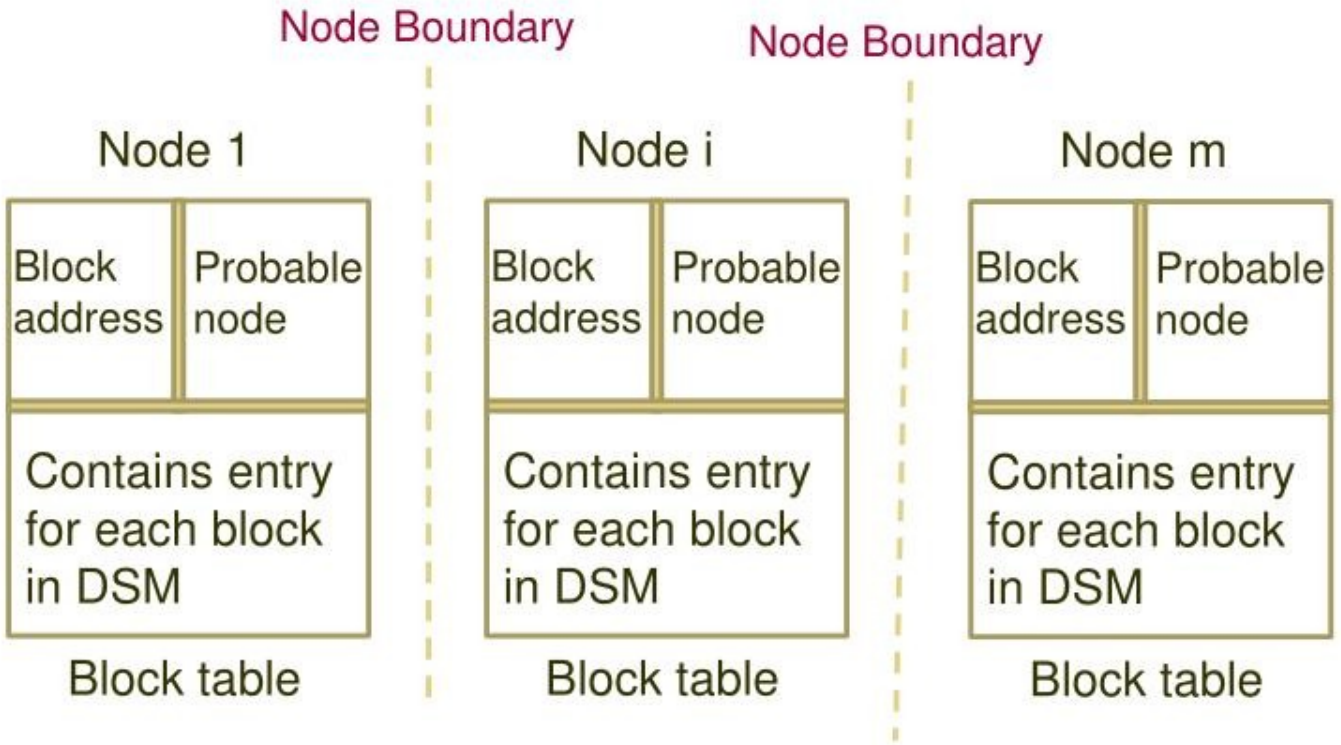
- Centralized Server Algorithm
  - Fault handler of faulting node sends request to centralized server, which forwards the request to current owner. Block is transferred & current node information also changed.
  - Centralized server serializes location queries, reducing parallelism
  - Centralized entity issues



- Fixed distributed server algorithm
  - Has a block manager on several nodes, and each block manager is given a predetermined subset of data blocks to manage.
  - Mapping function used to find out node whose block manager manages currently accessed node.



- Dynamic distributed server algorithm
  - Each node has a block table that contains the ownership information of all blocks (probable owner)
  - Chain of probable nodes might have to be traversed to reach true owner.

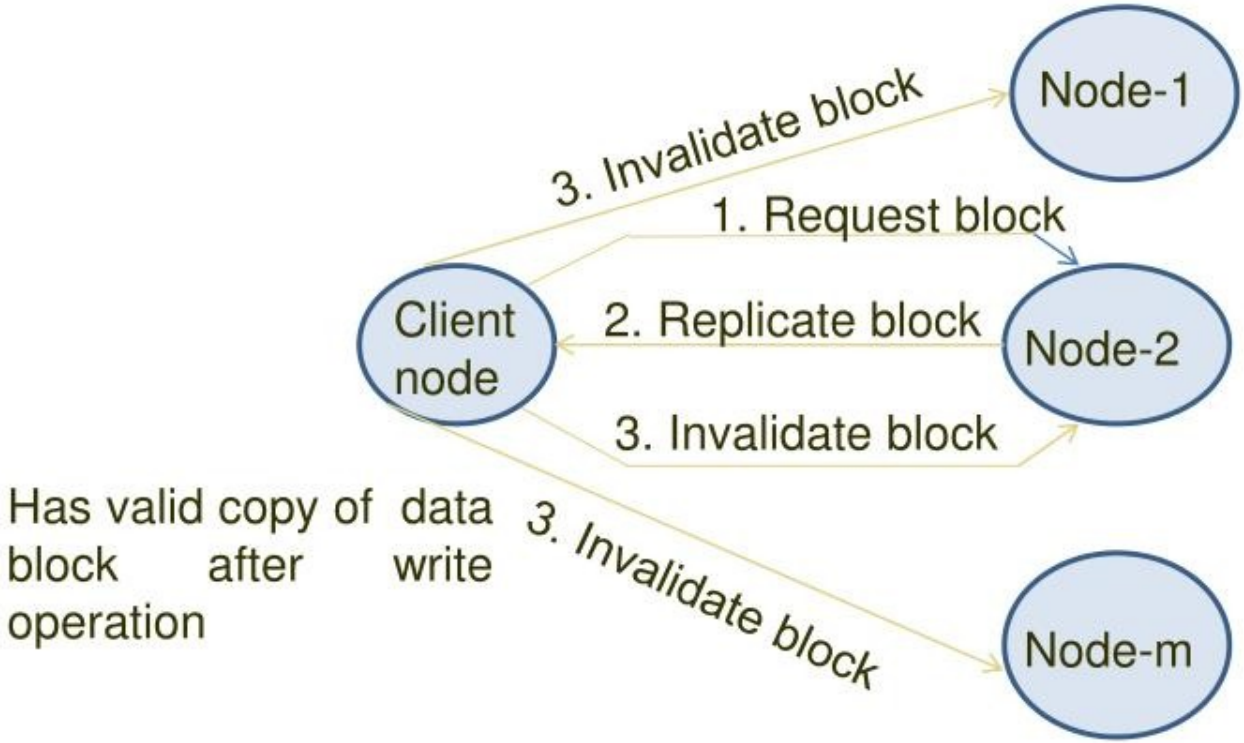


Replicated, Migrating Blocks

- Replication increases parallelism but complicates memory coherence
- Replication reduces reading cost but increases writing cost
- Protocols ensuring sequential consistency
  - Write invalidate
  - Write update

Write Invalidate

- If a node that had a copy of block before invalidation tries to access block after invalidation, a cache miss will occur & fault handler will fetch block again.
- Updates only propagated when data is read.

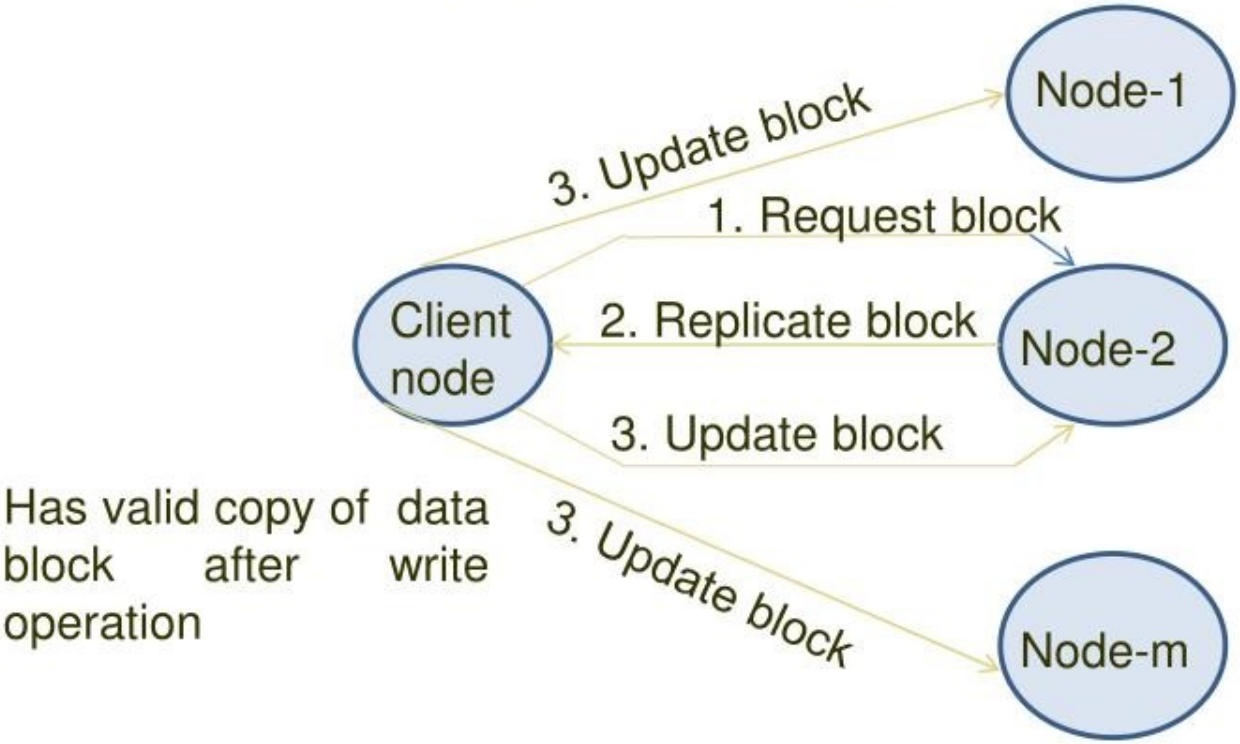


Has valid copy of data block after write operation

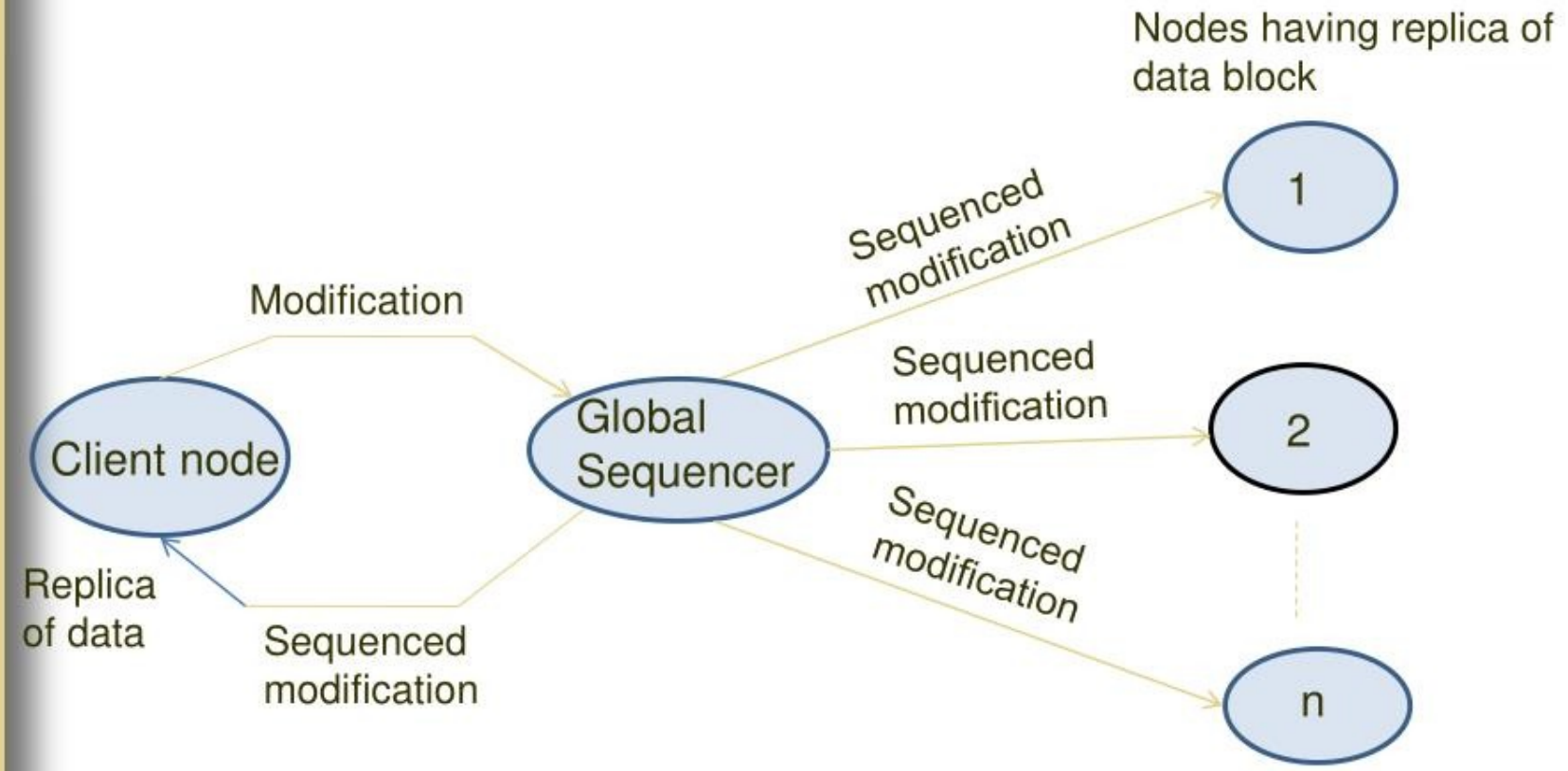
Nodes having valid copies of the data before write operation

Write Update

- Fault handler after modifying block, sends address of modified memory location & its new value to nodes having its copy.
- Write operation completes only after all copies of block have been successfully updated, making it an expensive approach.



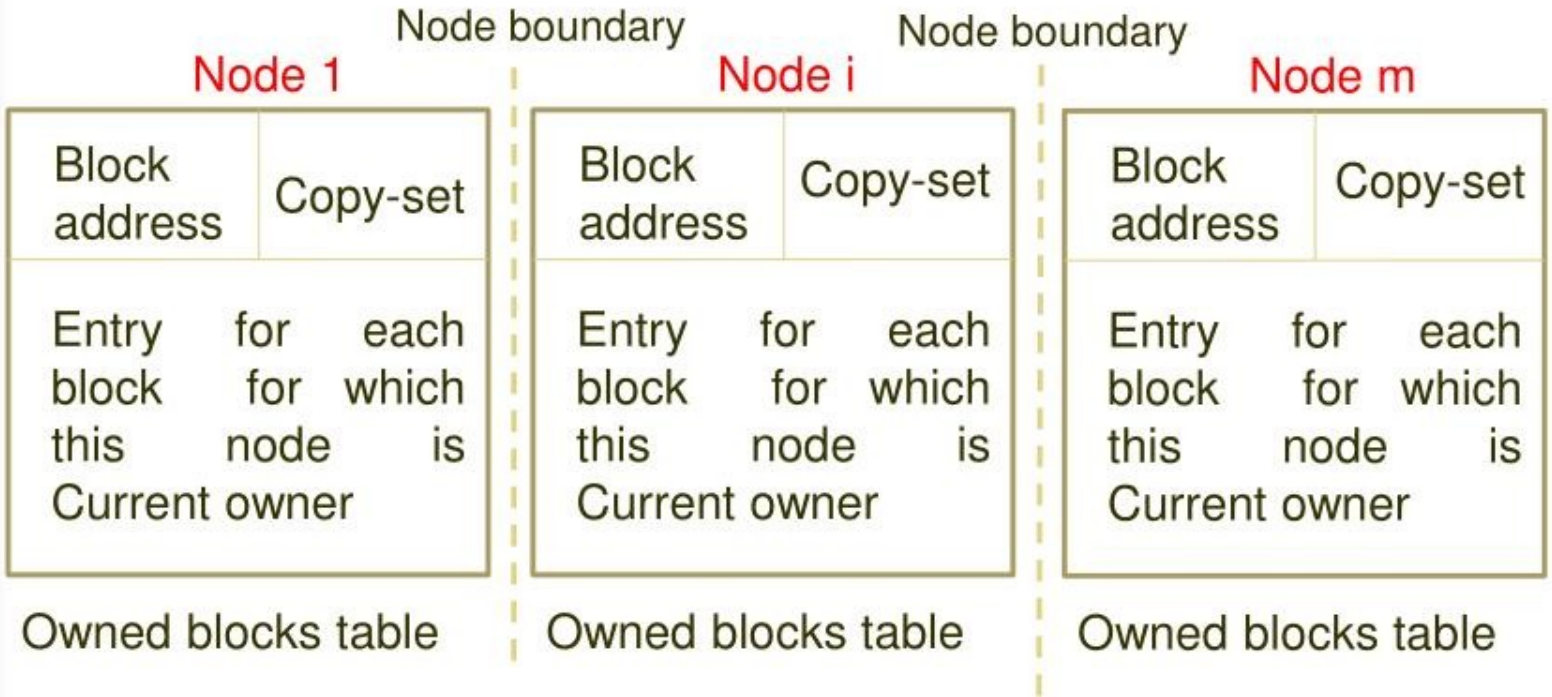
- Use a global sequencer to totally order write operations of all node to ensure sequential consistency.
- Intended modification of each write operation goes to sequencer which assigns it the next sequence number.
- When a new modification arrives at a node, its sequence number is verified with next expected one.



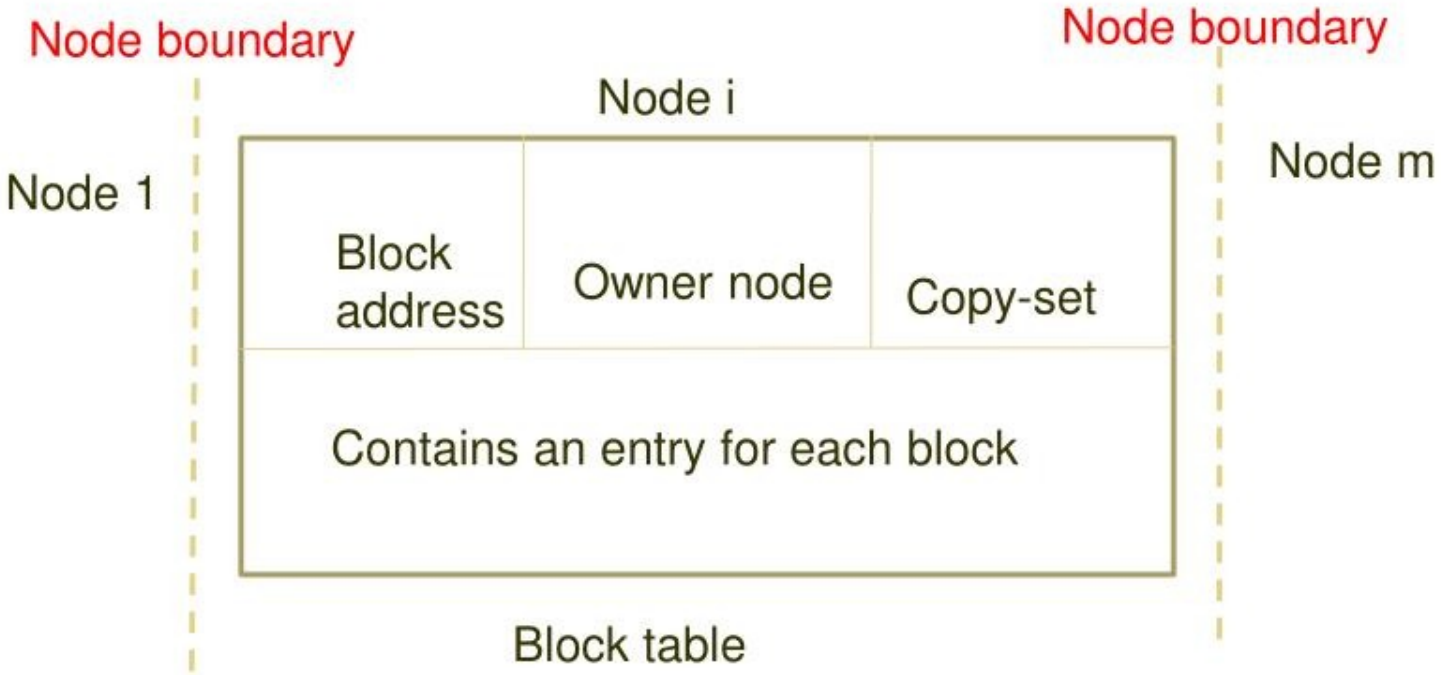
Data Locating

- Issues
  - Locating the owner of the block, the most recent node to have write access to it.
  - Keeping track of the nodes that currently have a valid copy of the block.

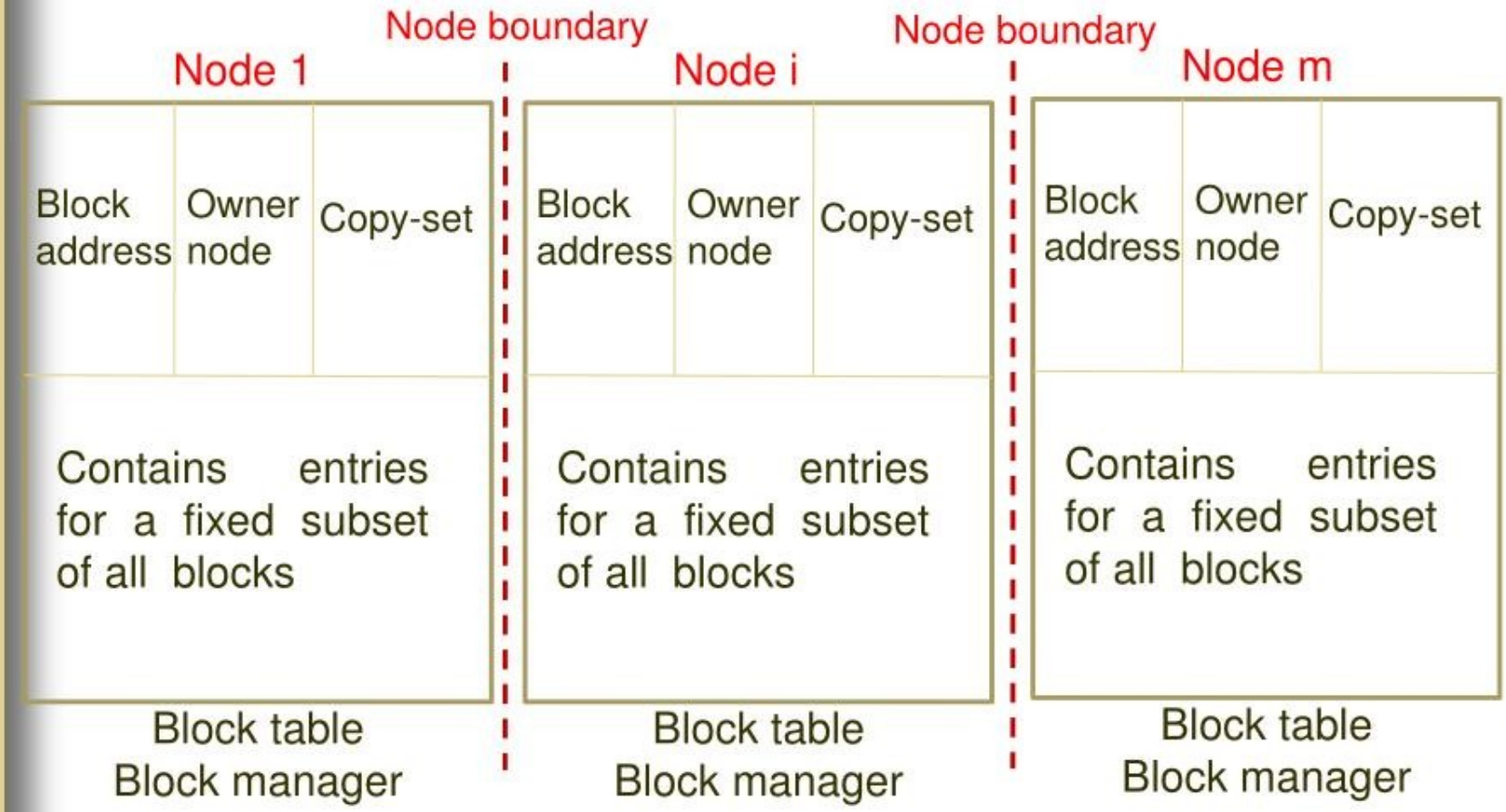
- Broadcasting
- Copy-set - list of nodes that currently have a valid copy of the corresponding block
- Read fault
  - Faulting node N broadcasts to find owner. Owner sends block to node N & adds node N to its copy set
- Write fault
  - Owner sends block & copy set to node N. N sends invalidation message to all nodes of copy set & initializes copy set to Null.



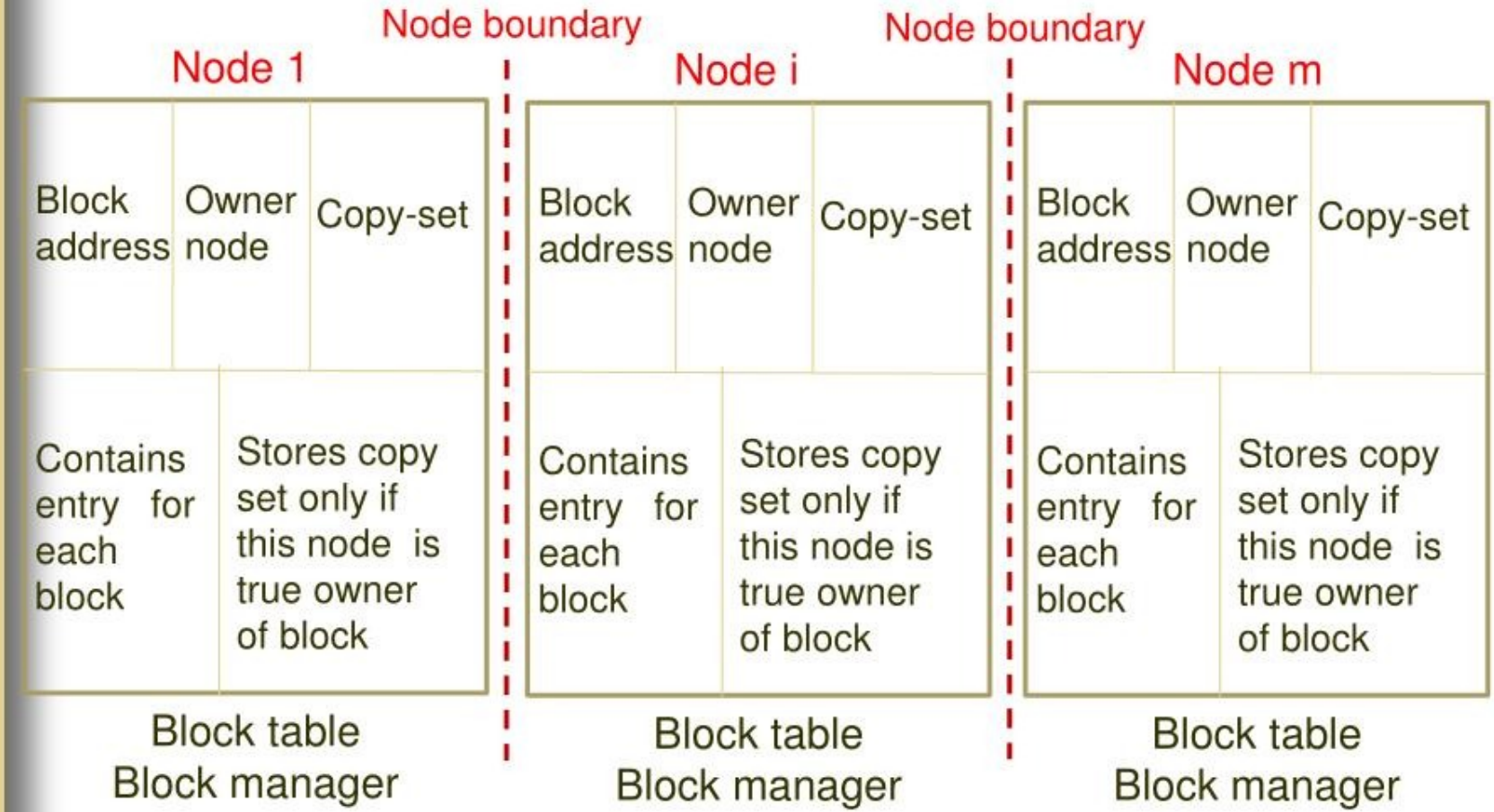
- Centralized Server algorithm
- Read fault
  - Sever adds N to copy set & returns owner information to N.
- Write fault
  - Returns both copy set & owner information to N & initializes copy set to contain only N. Node N on receiving block from owner, sends invalidate message to copy set.



- Fixed distributed server algorithm
  - Extension of centralized server.



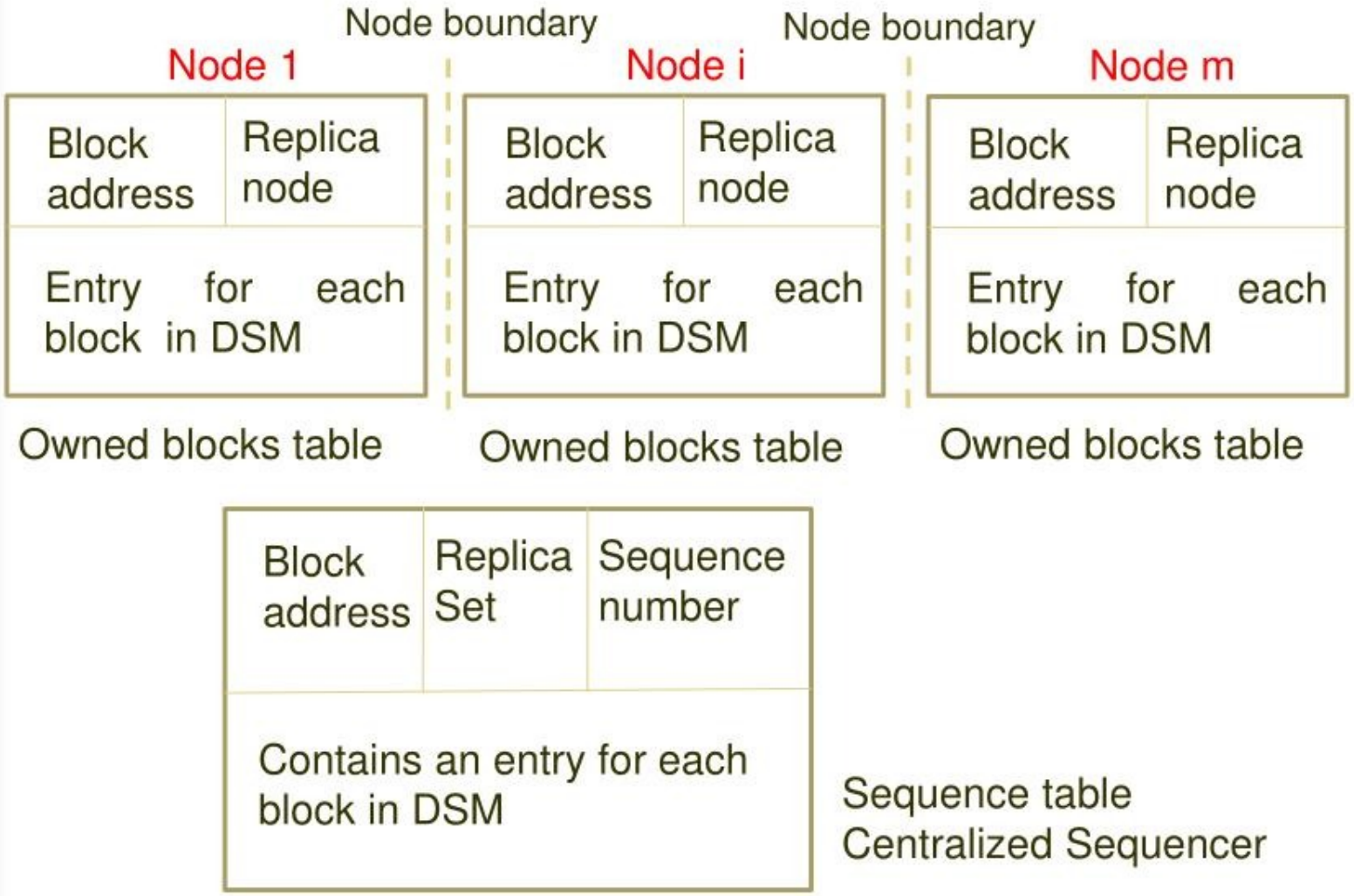
- Dynamic distributed server algorithm
  - Chain of probable nodes might have to be traversed to reach true owner.
- Read fault - Adds N to copy set & sends copy of block to N.
- Write fault - Returns both copy set & block to N & deletes its own copy set. Node N then sends invalidate message to copy set & updates block.



Replicated, Nonmigrating Blocks

- A shared memory block may be replicated at multiple nodes of the system , but the location of each replica is fixed.
- Write-update protocol is used
- Data locating characteristics
  - The location of replica is fixed
  - All replicas are kept consistent.
  - Read request directly by nodes having replica of block. All write request are sent to global sequencer to ensure sequential consistency.

- Write request is sent to sequencer. Sequencer multicasts modification with sequence number to all nodes of replica set



MUNIN : Release Consistent DSM

- Structure of shared memory
  - Structured as a collection of shared variables.
  - Shared variables declared with keyword shared
  - Each shared variable or variables with same annotation placed on separate page
- Synchronization mechanisms
  - Lock – acquirelock, releaselock
  - Barrier – waitAtBarrier primitive, use centralized barrier server
- On page fault, use probable – owner – based dynamic distributed server algorithm.
- Copy set used to keep track of all replicas.

Annotations for Shared Variable

- Read only
  - Read but never written after initialization
  - Can freely replicate them without consistency problem
- Migratory
  - Shared variables that are accessed in phases, where each phase corresponds to a series of accesses by a single process.
  - Used within critical section
  - NRMB strategy used
- Write Shared
  - Variable is updated concurrently by many processes, but different parts.
  - Replicated on all nodes
  - Makes twin page, then updates original. Differences between twin & original page sent to all replicas

- Producer Consumer
  - Shared variables written by only one process & read by a fixed set of processes
  - Write-update used to update replicas
- Result
  - Written by multiple processes (different parts) but read by one.
- Reduction
  - Must be atomically modified
  - Acquire lock, read, update, release lock
- Conventional

Replacement Strategy

- Which block to replace ?
  - Usage based versus non-usage based (LRU/ FIFO)
  - Fixed space versus variable cache space
- Usage based-fixed space algorithms are more suitable for DSM

- In DSM each node is classified into one of the following five types
  - Unused - free memory block
  - Nil - Invalidated block
  - Read-only - only read access right
  - Read-owned - owner with Read access right
  - Writable - write access permission

- Where to place a replaced block
  - Using secondary store (local disk)
  - Using the memory space of other nodes

Thrashing

- Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another , compare to time spent doing the useful work of executing application processes.
  - Providing application-controlled locks
  - Nailing a block to a node for a minimum amount of time t (fixed/ dynamically chosen on basis of past access patterns)
  - Tailoring the coherence algorithm to the shared-data usage patterns

Other approaches to DSM

- Data caching managed by the OS
- Data caching managed by the MMU hardware
  - Used for multiprocessors having hardware cache.
- Data caching managed by the language runtime system
  - DSM is structured as a collection of programming language constructs

Heterogeneous DSM

- Data conversion - Data converted when a block is migrated between two nodes
- Structuring the DSM system as a collection of source language objects
  - Unit of data migration is object
  - Usually objects are scalar data types, making system very inefficient
- Allowing only one type of data in a block
  - Page size is block size
  - DSM page table keeps additional information identifying type of data maintained in that page.
  - Wastage of memory due to fragmentation
  - As size of application level data structures differs for two machines, mapping between pages on two machines would not be one to one.
  - Entire pages are converted even if small portion required
  - Requires users to provide conversion routines.

- Blocksize selection
  - Largest page of all machines
  - Smallest page of all machines
  - Intermediate page size algorithm