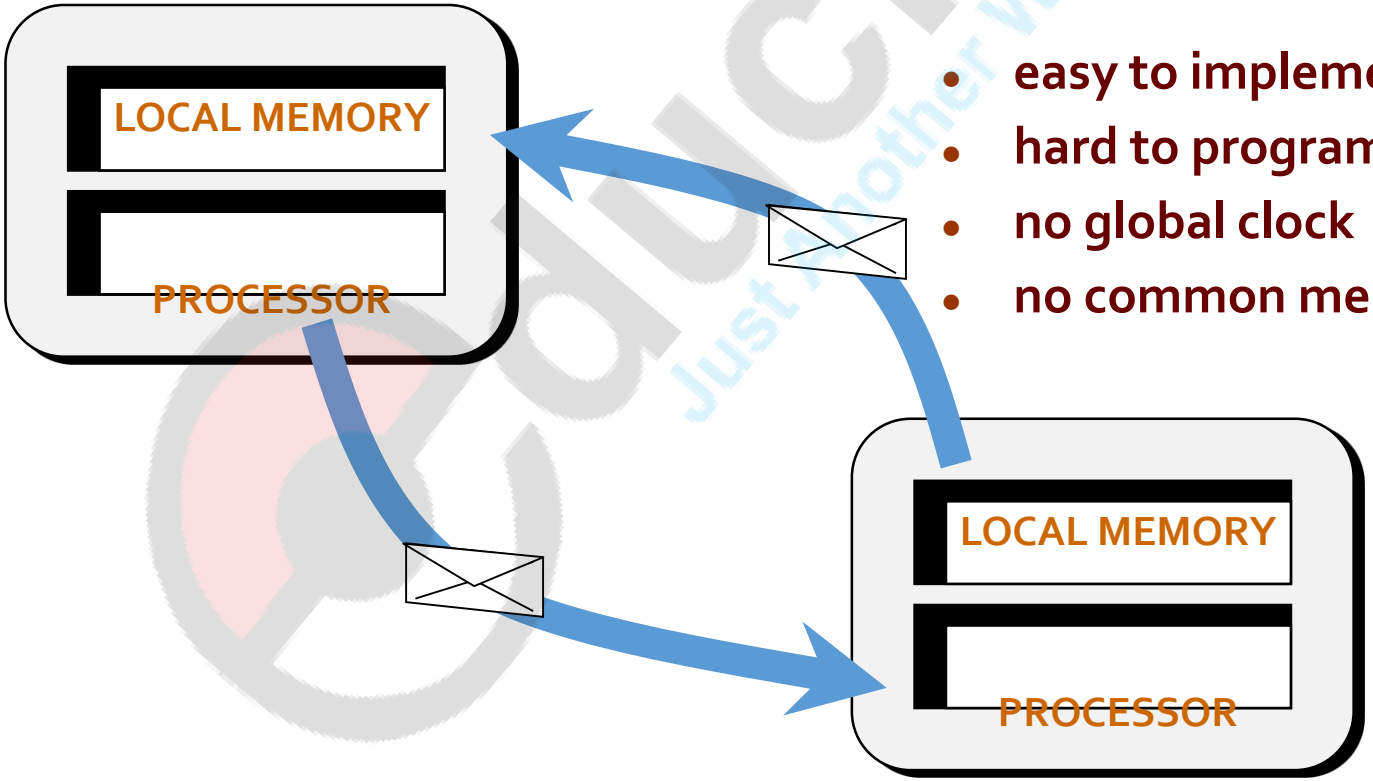
DISTRIBUTED SHARED MEMORY

Distributed Shared Memory

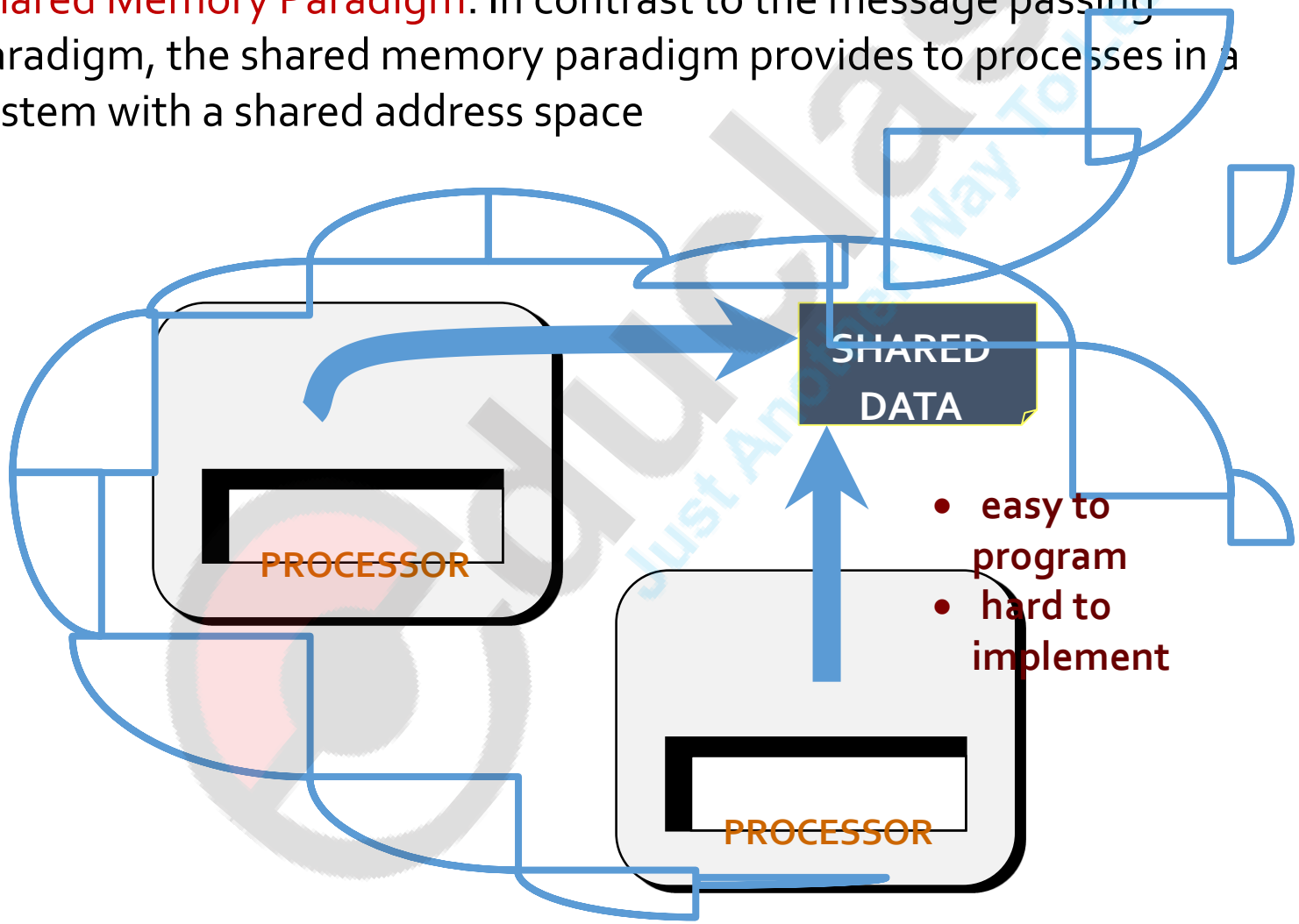
- We have mentioned earlier that there are two basic paradigms or IPC mechanisms namely:
- Message Passing Paradigm: Message passing systems or systems supporting RPC adhere to the message passing paradigm and it has two basic primitives for IPC



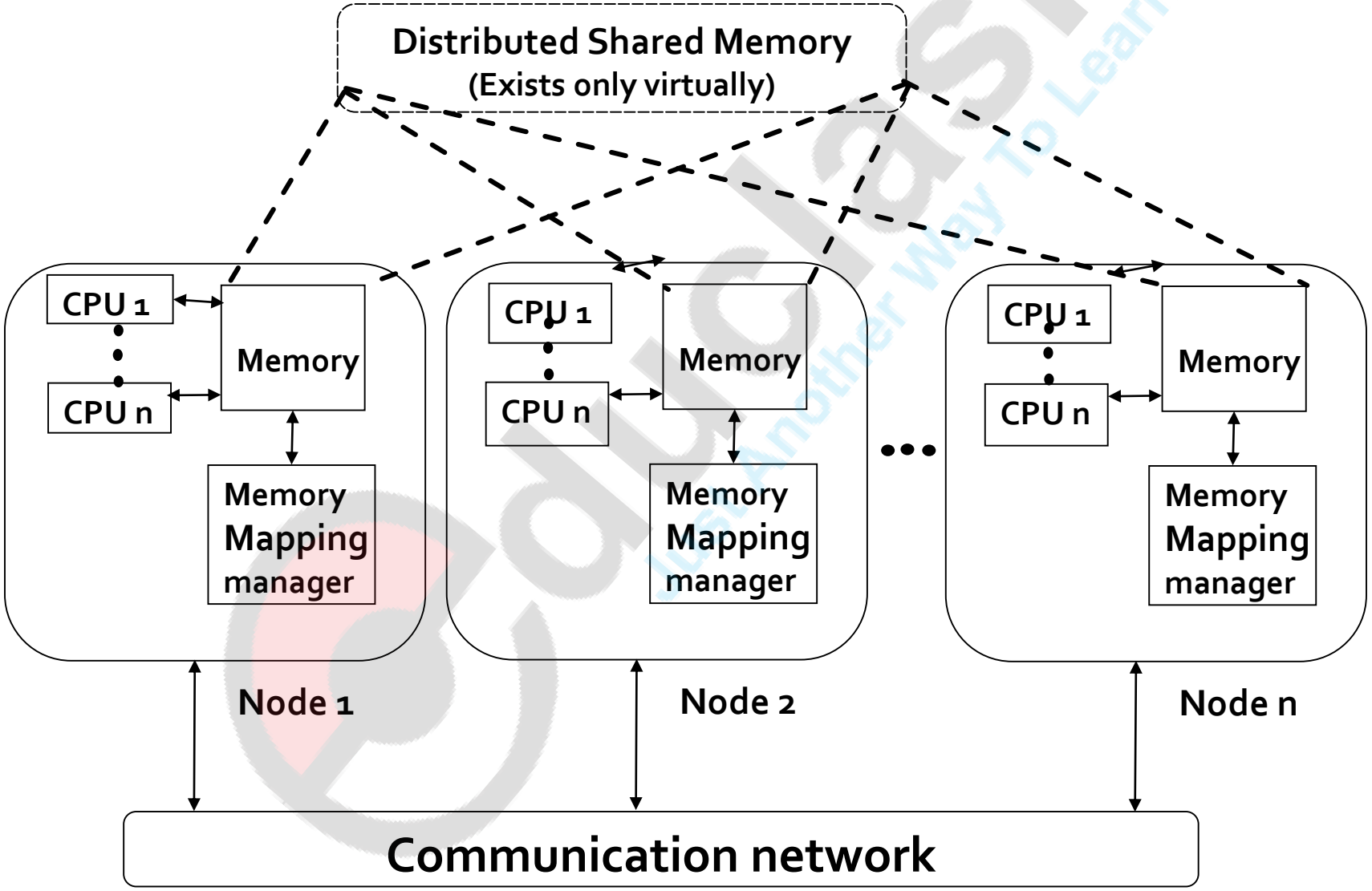
- **easy to implement**
- **hard to program**
- **no global clock**
- **no common memory**

Distributed Shared Memory(Cont'd)

- Shared Memory Paradigm: In contrast to the message passing paradigm, the shared memory paradigm provides to processes in a system with a shared address space



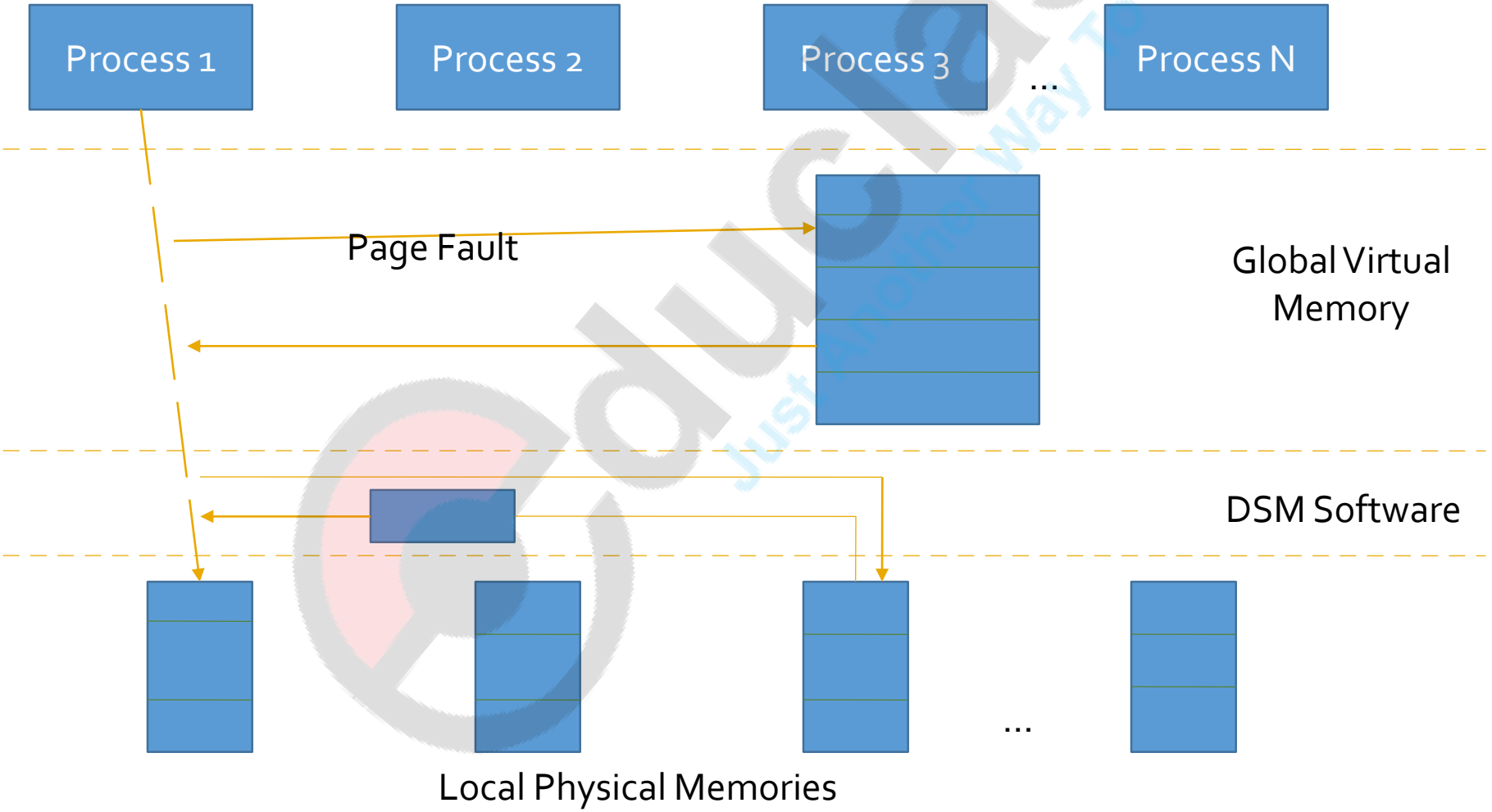
Distributed Shared Memory



Types of DSMs

- DSM systems can be classified on the basis of:
  - DSM Implementation (where and how)
  - DSM Algorithm (reflects adopted strategy)
  - Organization of DSM management
- Based on first criteria , systems are classified as :
  - Hardware-level DSM: these system uses smaller unit of sharing such as cache block.HDS is often used in high-end machines where performance is more important than cost but these implementations are not much scalable.

- Software-level DSM: this implementation is based on level of programming language, where the compiler detects shared accesses and inserts calls for synchronization and coherence into the code.
- Implementation is shown on next page
- If a processor does not find the page in its local memory, it triggers a page fault to the DSM runtime software. It locates the page and transfers it to the local memory of the requesting processor.


Process 1
Process 2
Process 3
...
Process N
Page Fault
Global Virtual Memory
DSM Software
...
Local Physical Memories

- Hybrid-level DSM: Hybrid level DSM lies between hardware and software DSM systems. Typical example for this is NUMA machines. Like a multiprocessor, each NUMA processor can access each word in the common virtual address space by reading or writing to it.

Advantages of DSM

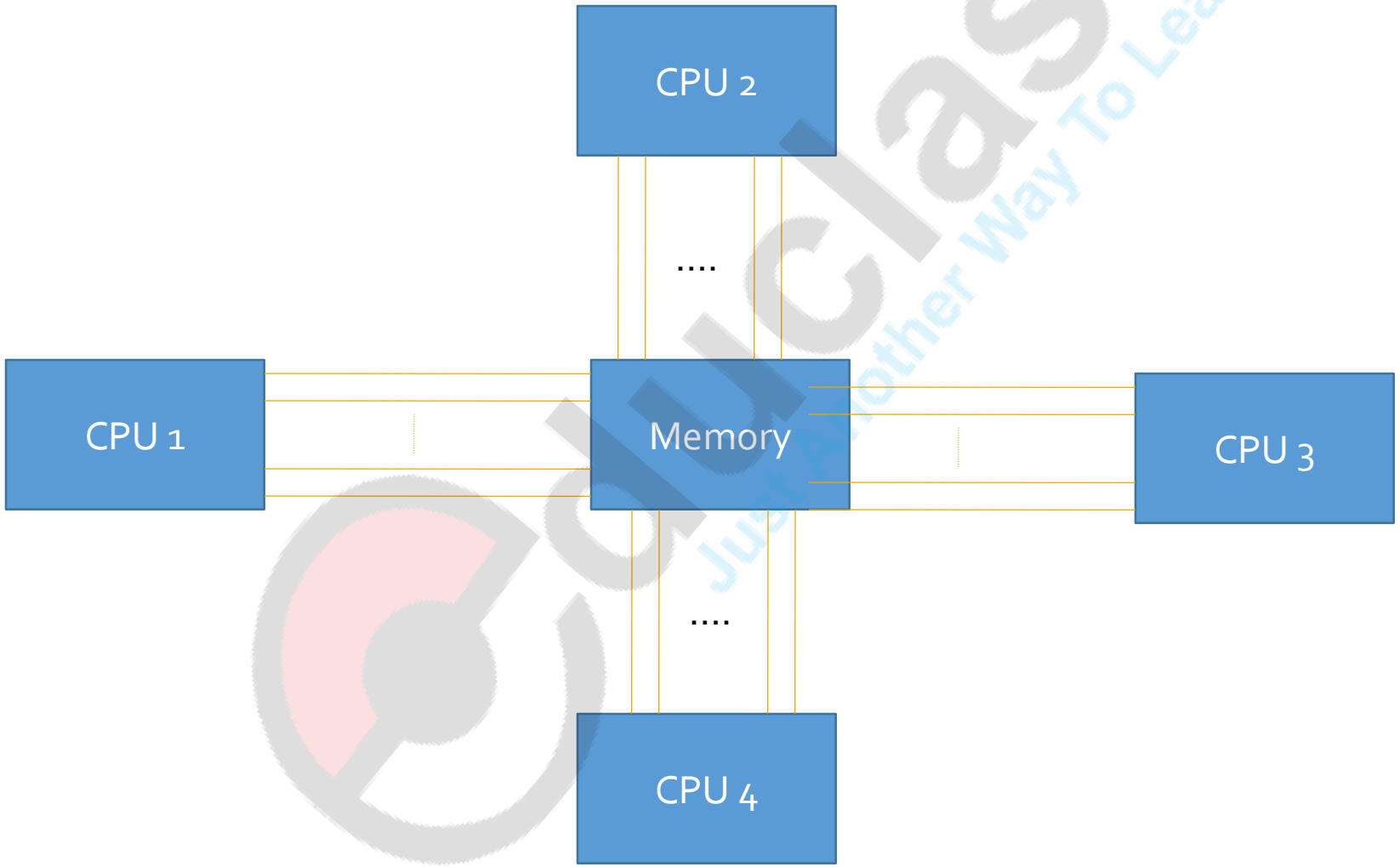
- Simpler abstraction
  - Shields programmers from low level concerns
- Better portability of distributed application program
  - Distributed application programs written for shared memory processor can be executed on DSM system without any change
- Better performance of some application
  - Ongoing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

Advantages of DSM

- Flexible communication environment
  - No formal IPC required, communication through shared space
- 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

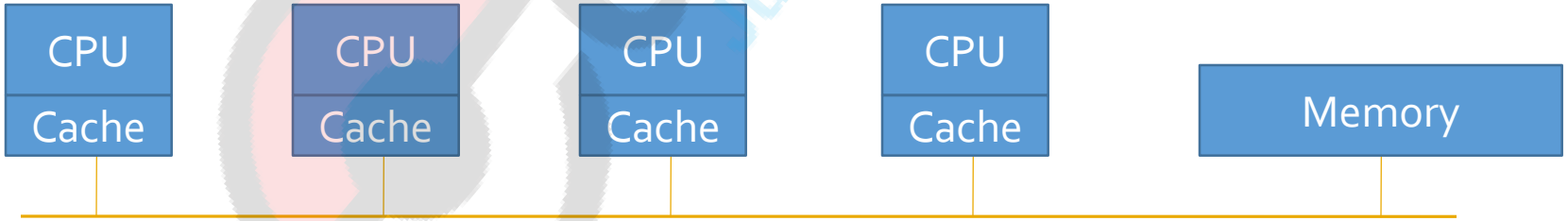
Hardware DSM

- There are few hardware architectures discussed here:
  - On-chip Memory DSM: CPUs with little functionality and on-chip memory are widely used in cars, appliances and toys. The address and the data lines are directly connected from all CPUs to the single shared memory. Practically, it is expensive and impossible to build a set of hundred CPUs with single shared memory.


CPU 2
....
CPU 1
Memory
CPU 3
....
CPU 4

- Bus-based Multiprocessor:

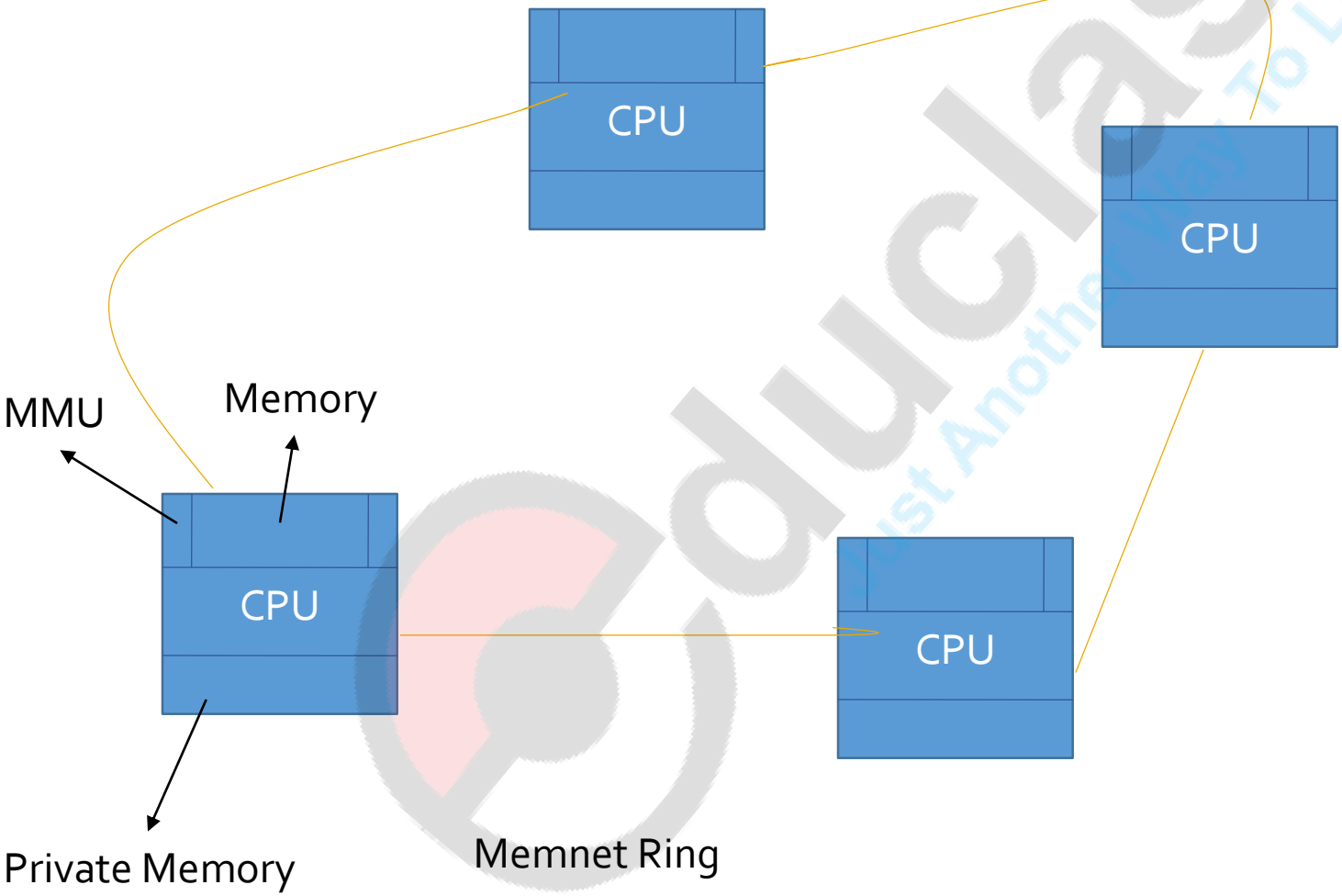
Bus-based multiprocessor



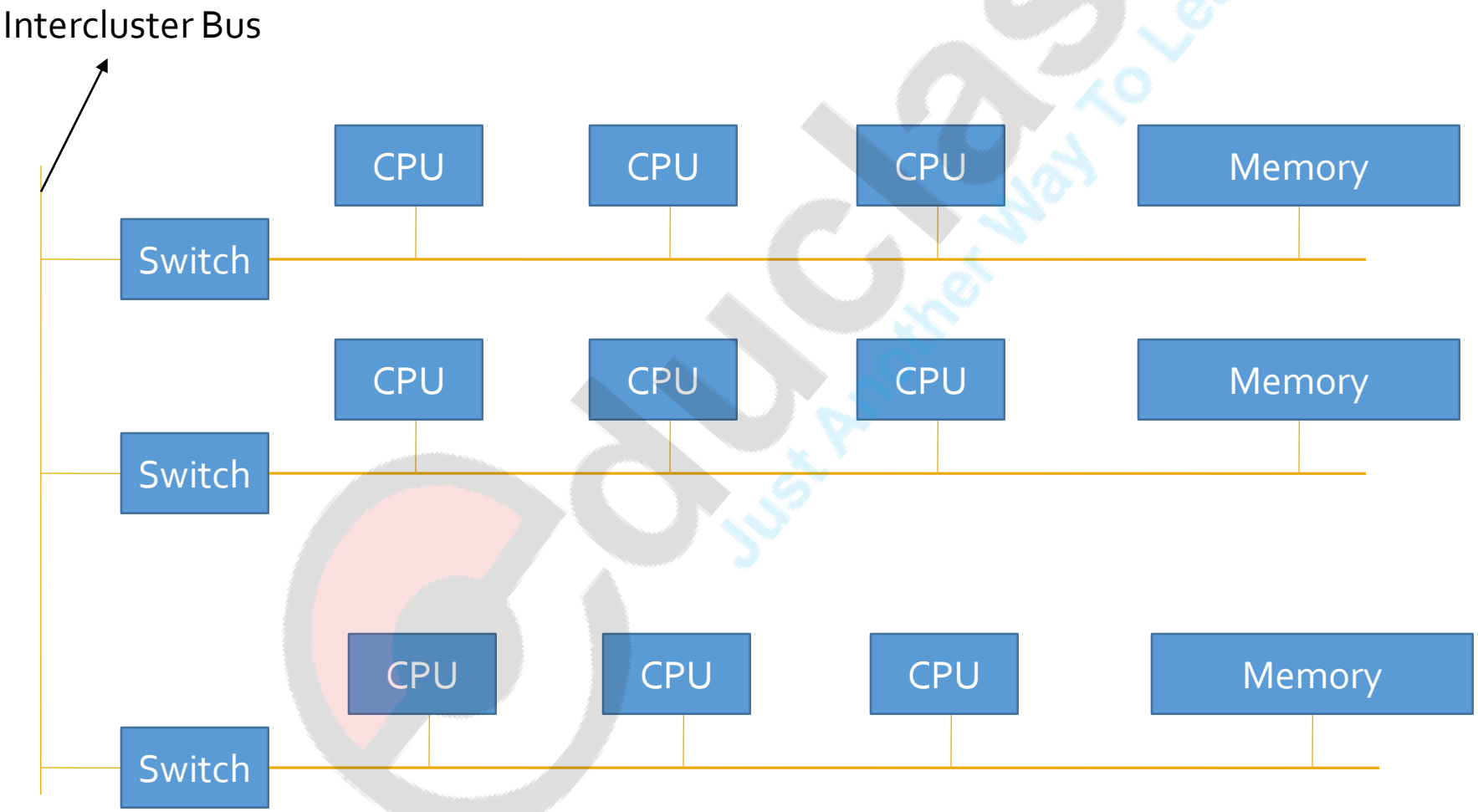
Bus-based multiprocessor with caching

- To maintain consistency amongst caches write-through cache consistency protocol is used.
- Advantages:
  - Consistency is achieved since all caches snoop.
  - Protocol is built into MMU.
  - The algorithm is performed in one memory cycle.

- Ring-based Multiprocessor: these are implemented in Memnet DSM. A single shared memory is divided into private and shared areas.
  - Private space is allocated per machine to accommodate unshared data and code.
  - Hardware is used to keep the data in shared spaces consistent


CPU
CPU
MMU
Memory
CPU
CPU
Private Memory
Memnet Ring

- Switched Multiprocessor: both bus and ring bandwidth saturate and reduce system performance.
- To overcome the problem, we can either reduce amount of communication by caching or increase the communication capacity by adding more than one bus.


Intercluster Bus
CPU
CPU
CPU
Memory
Switch
CPU
CPU
CPU
Memory
Switch
CPU
CPU
CPU
Memory
Switch

Design and Implementation Issues of DSM

1. Granularity:
   - **Refers to block size of a DSM system, the unit sharing and data transfer across the network when network block fault occurs**
   - Block size is a measure of the granularity of a DSM
   - It is an important part of the design of DSM as it determines the amount of network traffic generated on network on block fault
2. Structure of shared memory space:
   - Layout of the shared data in memory
   - Its structure is normally dependent on the applications that the DSM system intend to support
   - Can be unstructured, structured as datatypes, structured as database.

Design and Implementation Issues of DSM (Cont'd)

3. Memory coherence and access synchronization
   - In a DSM system that allows replication of shared data items, copies of shared data items may simultaneously be available to main memories of number of nodes
   - In this case the main problem is the memory coherence that deals with the consistency of a piece of shared data lying in the main memories of two or more nodes
   - This problem is similar to the problem in shared multiple processor systems that have multiple caches
   - In a DSM system concurrent accesses to shared data is possible
   - Hence just memory coherence protocol is not sufficient to maintain consistency of shared data
   - Synchronization primitives, such as semaphores, event count and lock are needed to synchronize concurrent accesses to shared data

Design and Implementation Issues of DSM (Cont'd)

4. Data location and access
   - To share data in a DSM system, it should be possible to locate and retrieve data accessed by a user process
   - Hence some form of data block location mechanism to service network data block faults must be implemented by the DSM.
5. Replacement strategy
   - If the local memory of a node is full, a cache miss at that node implies not only a fetch of the accessed data block from a remote node, but also a replacement
   - i.e., a data block of the local memory that must be replaced
   - Hence a cache replacement strategy has to be in place

Design and Implementation Issues of DSM (Cont'd)

6. Thrashing
   - In DSM the data blocks migrate between nodes on demand
   - Data block gets transferred back & forth at a high rate if two nodes compete for write access to a single data item
   - This will result in heavy network traffic without really achieving any real work done
   - Hence DSM must use a policy to avoid this situation known as thrashing
7. Heterogeneity
   - In a network containing heterogeneous systems, one needs to look into individual memory access modalities and architecture, before building a DSM system

Granularity

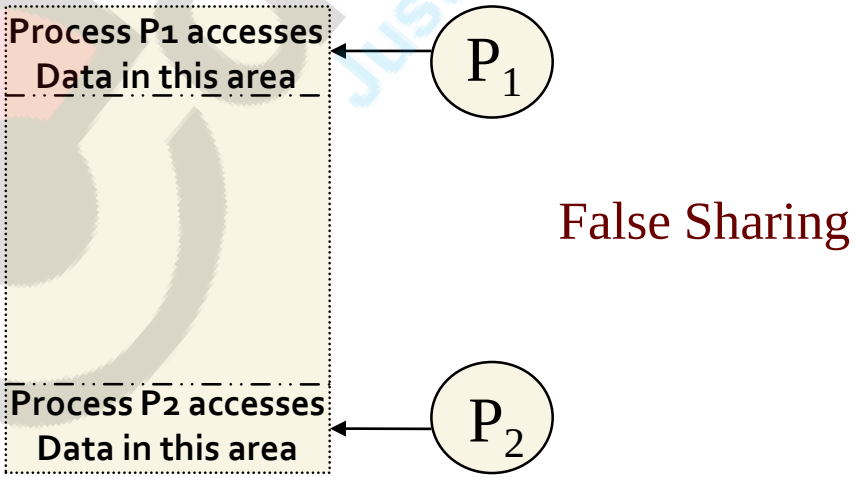
- One of most important factor to be chosen in the design of a DSM system is the block size
- Factors influencing block size selection
  - Paging overhead
    - Less for large block size because of locality of reference as it is well known that a process is likely to access a large region of the address space in a small interval of time
    - Hence paging overhead is less for large block sizes
  - Directory size
    - Larger block size, smaller directory information of the blocks in a system to be maintained

Granularity (Cont'd)

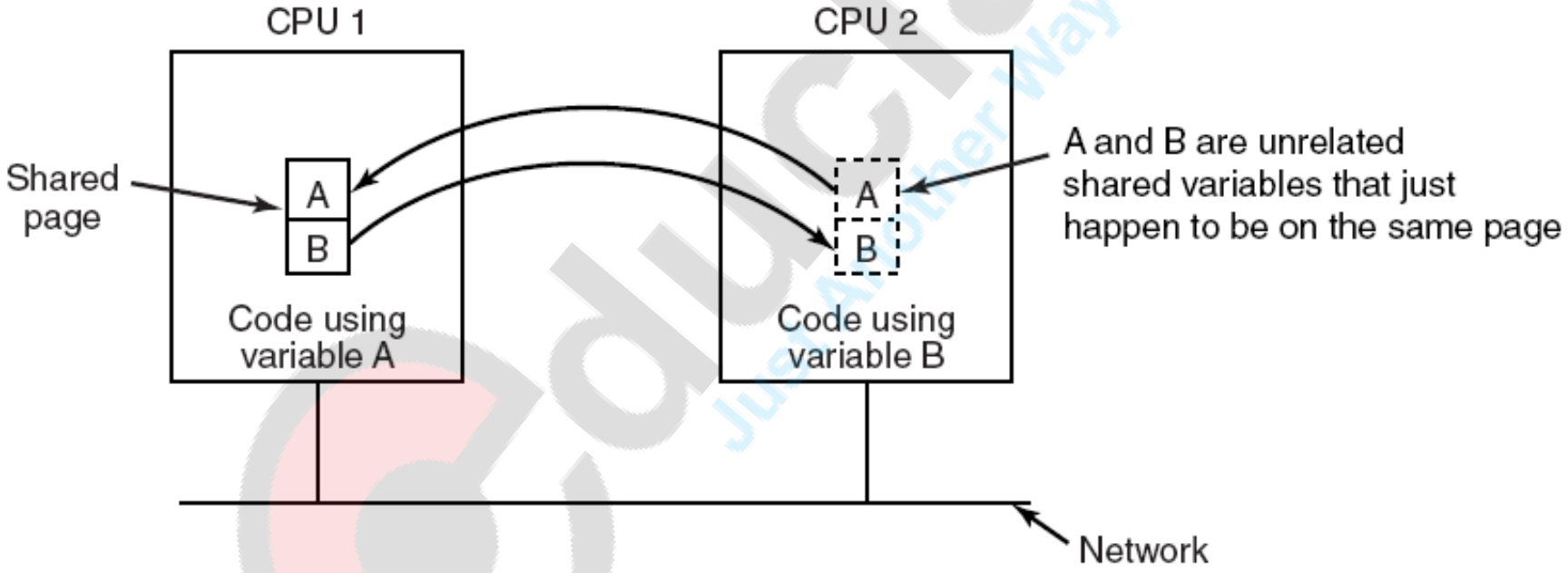
- Thrashing
  - Thrashing occurs when data items in the same block are being updated by multiple nodes at the same time, resulting large number of block transfers among nodes without much progress in application execution
  - Though thrashing can occur in any block size, it is more likely in large block sizes
  - As different data in same data block may be updated by processes in different nodes which may not be necessary if the block size was smaller
  - Another condition in which thrashing occurs is when two variables in different blocks are accessed repeatedly and one of the two pages is swapped out of a system by the page replacement algorithm again and again

Granularity (Cont'd)

- False sharing
  - False sharing occurs when two different processes access two unrelated variables that reside in same data block
  - In such a case even though the original variables are not shared, the data block appears to be shared by two processes
  - Larger is the block size, higher is probability of false sharing
  - Also note this can lead to thrashing



False Sharing



Block sizing

- Use page size as block size
  - Relative advantages & disadvantages of small and big block size, makes it difficult for a DSM designer to decide on the block size
  - It has been shown that using a same block size as the page size of the individual system has the following advantages
  - Allows the use of existing page-fault scheme
  - Allows access right control to be integrated into memory management unit of the system
  - If page can fit into packet, it does not impose undue communication overhead at the time of page fault
  - Experience has shown that page size is a suitable data entity for memory contention

CONSISTENCY MODELS

Consistency Models

- Consistency requirements of a DSM varies from application to application
- A consistency model basically refers to the degree of consistency that has to be maintained for the shared memory data for the memory to work correctly for a certain set of applications
- 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
- It is defined as a set of rules that applications must obey if they want the DSM system to provide the degree of consistency guaranteed by the consistency model

Consistency Models (Cont'd)

- A system supports stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true
- Several consistency models have been proposed
- Let us consider some of the main ones

Strict (Atomic) Consistency Model

- It is the 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) irrespective of the locations of the processes performing read & write operations
  - i.e. all write instantaneously visible to all processes
- Absolute synchronization of clocks of all the nodes of a distributed system is not possible, so is the single global time
- Hence an implementation of strict consistency model in DSM is practically not possible
- Possible only on uniprocessor or shared memory systems

Sequential Consistency Model

- Weaker than strict consistency model and experience shows that programmers can often manage well with weaker models
- When processes run concurrently (possibly) on different machines, any valid interleaving of read and write operations is acceptable behavior, but all processes see the same interleaving operations
- A shared memory system is said to support the sequential consistency model if all processes see the same order of all memory access operations on the shared memory
- 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 in that order, 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

Causal Consistency Model

- The casual consistency model relaxes the requirement of sequential consistency model for better concurrency
- In this model all processes see only those memory reference operations in the same (correct) order that are potentially casually related
- Memory reference operations that are not potentially casually related writes may be seen in a different order on different machines
- A memory reference operation (read/write) is said to be potentially causally related to one another memory reference operation if the first one might have influenced the second one in any way .
- For example, if a process performs a read operation followed by write operation, the write operation is potentially causally related to read operation, because the computation of the value written may have some way depend on the value obtained by the read operation

Causal Consistency Model (Cont'd)

- On the other hand, a write operation performed by one process is not causally related to a write operation performed by another process, if the first process had not read either the value written by the second process or any memory variable that was directly or indirectly derived from the value written by the second process

Pipelined Random Access Memory (PRAM) Consistency Model

- PRAM consistency is 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})

PRAM consistency Model (Cont'd)

- All write operations performed by a single process are pipelined i.e., the process does not have to stall waiting for each one to complete before starting next one and all writes by different processes are concurrent
- It leads to better performance than the previous models because a process need not wait for a write operation performed by it to complete before starting the next one, as all write operations of a single process are pipelined

Processor Consistency Model

- Processor Consistency Model is 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*
- Memory coherence means all processes agree on the same order of all write operations to that location
- 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 P_1 executes w_{11} & w_{12}

  Process P_2 executes w_{21} & w_{22}

  P_3 & P_4 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 same memory location

Weak Consistency Model

- The weak consistency model is based on following facts
  - It is not necessary to show the change in memory done by every write operation to other processes
  - The results of several write operations can be combined & sent to other processes only when they need it; e.g., when executes in Critical section
  - Other processes need not see changes made by a process until that process exits from critical section
  - Isolated accesses to shared variables are rare; i.e., in many applications, a process makes several accesses to a set of shared variables and then no access at all to the variables in this set for long time
- These characteristics imply that better performance can be achieved if consistency is enforced on a group of memory operations rather on individual memory reference operations

Weak Consistency Model (Cont'd)

- This is the basic idea behind weak consistency model
- Responsibility of programmer is to decide when to reflect changes in all processes, but for better performance
- For this, DSM supporting weak consistency model uses a special variable called synchronization variable and operations on it can synchronize memory
- When it is accessed by a process, the entire (shared) 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

Weak Consistency Model (Cont'd)

- The following requirements must be met:-
  - All Accesses to synchronization variables associated with a data store 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 completed everywhere, before access to a synchronization variable
  - Synchronization "flushes the pipeline, by forcing all the writes operations to be completed everywhere
  - 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

Release Consistency Model

- In weak consistency model the entire (shared) memory is synchronized when a synchronization variable is accessed by a process and memory synchronization involves
  - All changes made to the memory by the process are propagated to other processes (which is required only on exiting critical section).
  - All changes made to the memory by other processes are propagated from other nodes to the process's node (required only before entering CS).
- In order to facilitate this Release consistency model was proposed

Release Consistency Model (Cont'd)

- It uses two synchronization variables
  - *Acquire* used to enter critical section
  - *Release* used to exit critical section
- *Acquires* & *releases* on different locks occur independent of each other
- The programmer is responsible for inserting these two variables in the code, explicitly by calling library procedures
- Can be achieved by using barrier mechanism also instead of critical regions

Release Consistency Model (Cont'd)

- 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
- **This also known as eager release consistency**

Lazy Release Consistency Model

- A different implementation of release consistency is lazy release consistency
- Usually, when a release is done, the process doing the release pushes all the modified data to all processes that already have a copy of the data and thus might potentially need it and there is no way to tell if they actually need it, making the system inefficient
- In Lazy release model, modifications are not sent to other nodes at the time of release
- When a process does an acquire, it has to get the most recent values of the data from the process or processes holding them
- No network traffic generated until another process does acquire

Summary of Consistency Models

- 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

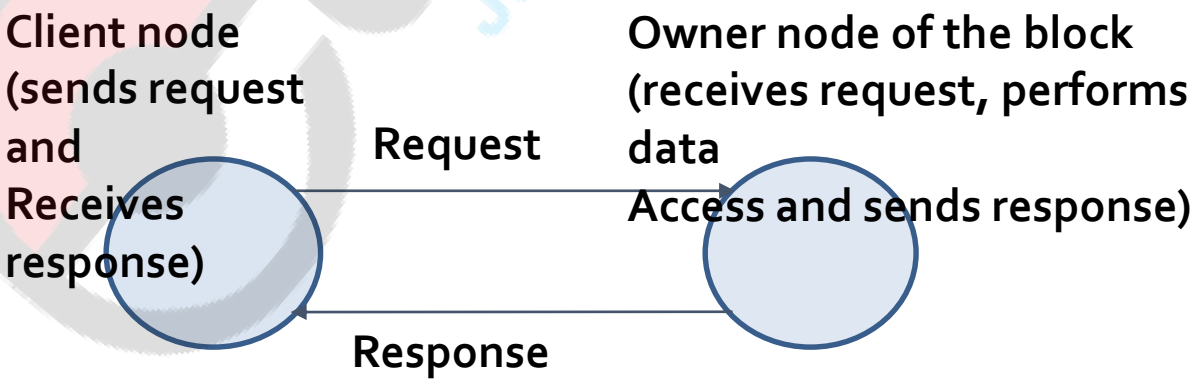
Implementation of Sequential Models

Implementing Sequential Consistency Model

- We have seen our earlier discussion is that the most commonly used consistency model in DSM systems is the sequential consistency model
- A protocol for implementing the sequentially consistent DSM systems is presented here
- These protocols to a great extent depend on **whether DSM allows replication and/or migration of shared memory data blocks**
- The design may choose among the following replication and migration strategies
  - Nonreplicated , nonmigrating blocks (NRNMBs)
  - Nonreplicated , migrating blocks (NRMBs)
  - Replicated , migrating blocks (RMBs)
  - Replicated , nonmigrating blocks (RNMBs)

Nonreplicated, Nonmigrating Blocks

- This is the Simplest strategy
- Each block of the shared memory has a single copy whose location is always fixed
- All access requests to a block from any node are sent to the *owner node* of the block, which has only copy of the block
- On receiving the request from a client node, the memory management unit (MMU) and Operating System S/W of the owner node perform the access request on the block and return a response to the client

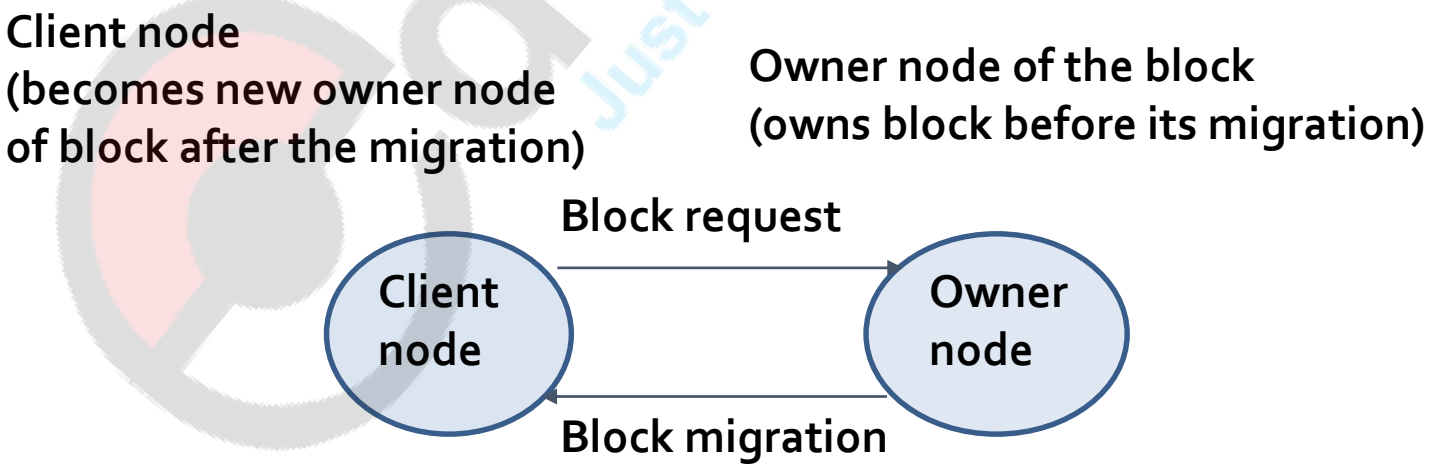


Nonreplicated, Nonmigrating Blocks

- Although the method is simple and easy to implement, it suffers from the following Drawbacks
  - Serializing data access creates a bottle neck
  - Parallelism, which is a major advantage of DSM is not possible
- Data Locating in the NRNMB Strategy has the following characteristics
  - There is a single copy of each block in the system
  - The location of the block never changes
  - Use mapping function to map a block to a node
  - When the fault occurs, the fault handler of the faulting node uses the mapping function to get the location of the accessed block and forwards the access request to that node

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



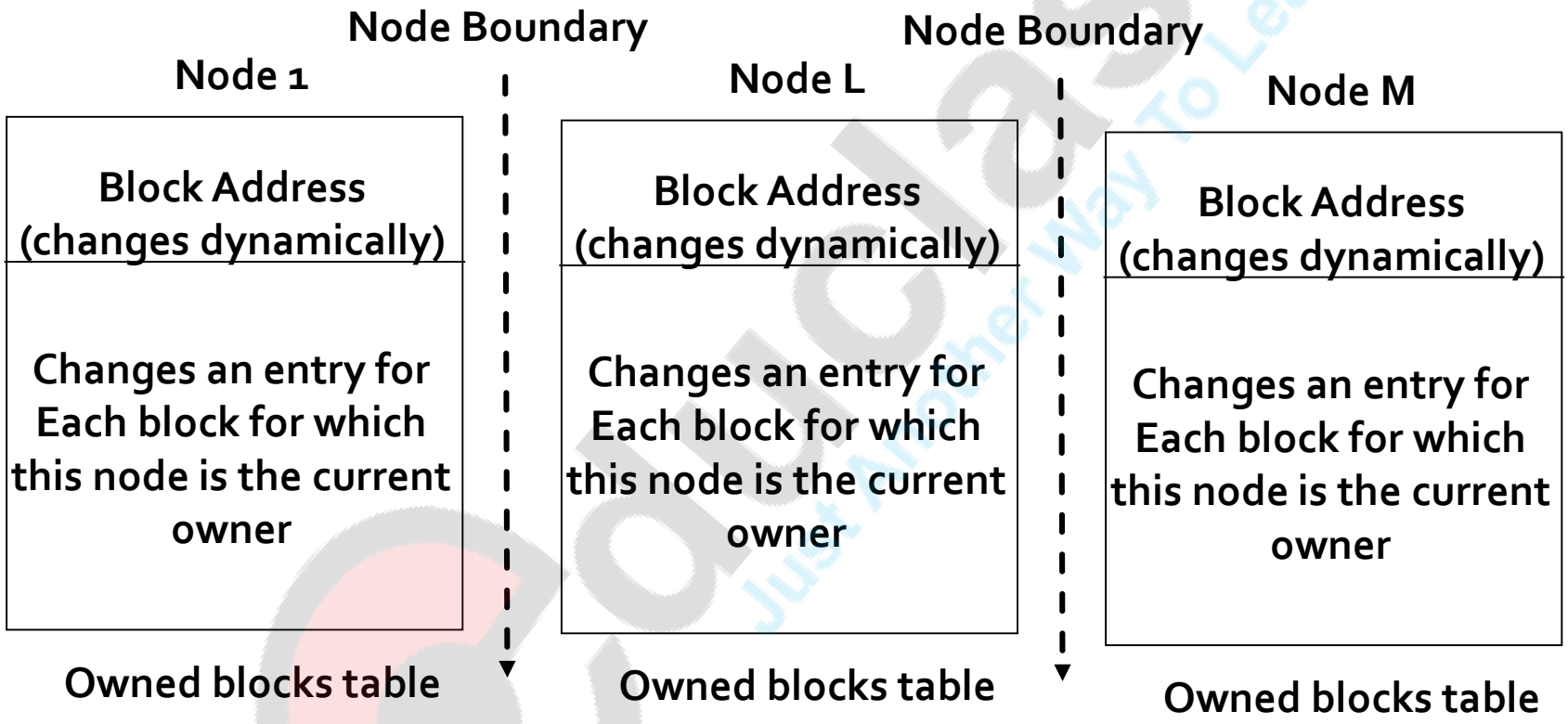
Nonreplicated, Migrating Blocks

- The method has the following advantages
  - Data located locally so no communication cost
  - Cost of multiple accesses reduced if high locality of reference.
- Disadvantages
  - Prone to thrashing problem
    - The block may keep migrating frequently from one node to another, resulting in few memory accesses between migration and thereby poor performance
  - The advantage of Parallelism can not be availed in this method also
- In the NRMB strategy, although there is a single copy of each block, the location of block keeps on changing dynamically

Data Locating in NRMB Strategy

- Hence one of following method can be used to locate a block
- Broadcasting
  - Each node maintains an owned blocks table that contains an entry for each block for which the node is the current owner (fig in next slide)
  - When page fault occurs, the fault handler of faulting node broadcasts a read/ write request on network, to which the current owner responds by sending the block
  - The major disadvantage of this algorithm is that it does not scale well
  - When a request is broadcast all nodes must process broadcast request leading to communication bottleneck
  - Network latency may also lead to accesses taking long time

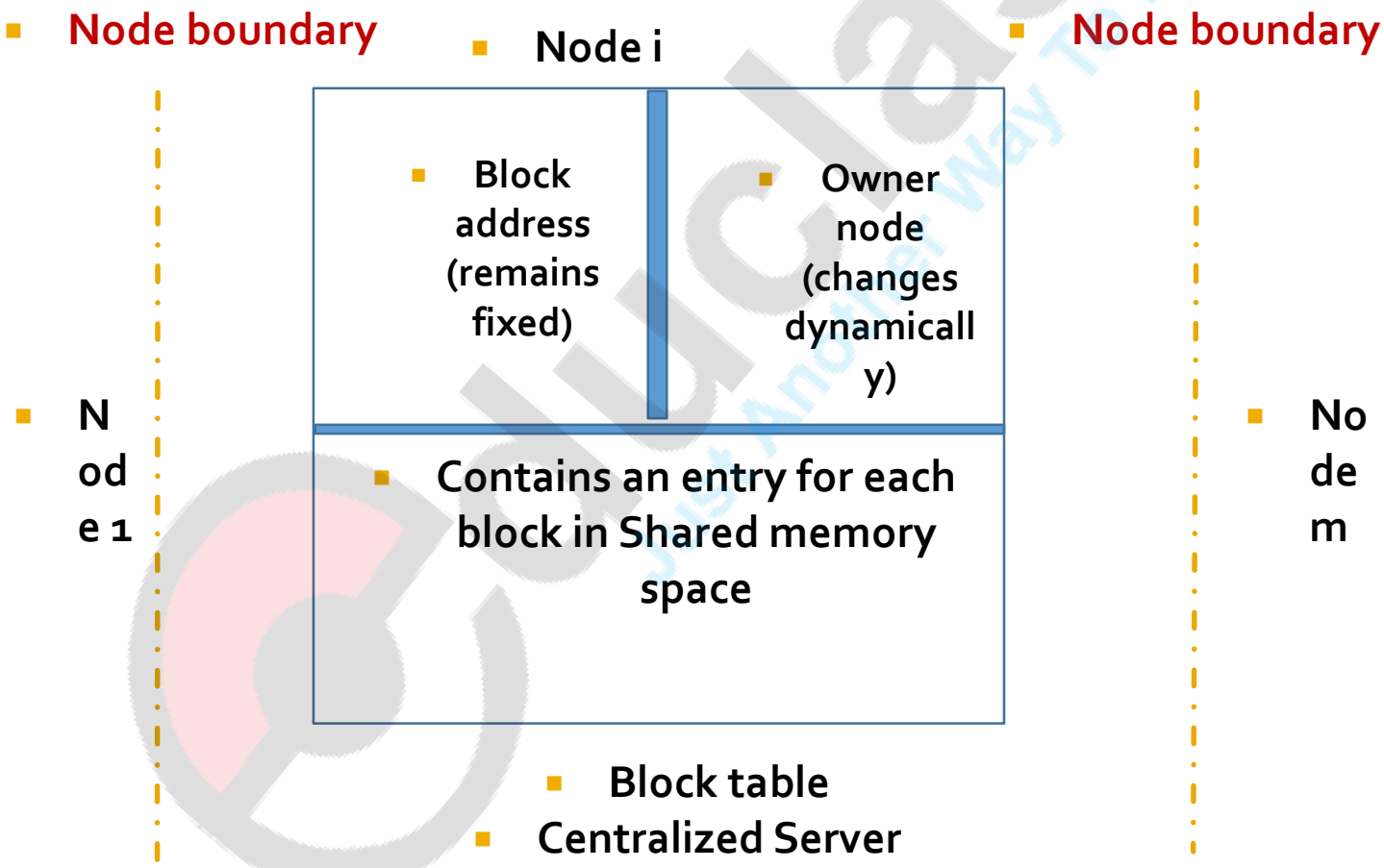
Data Locating in NRMB Strategy



Centralized Server Algorithm

- A centralized server maintains a block table that contains the location information for all blocks in the shared-memory space (fig. in next slide)
- The location and identity of the centralized server is well known to all nodes
- When a fault occurs 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
- The main drawbacks are
  - Centralized server serializes location queries, reducing parallelism
  - Centralized server is bottleneck and its failure will make the entire DSM to halt

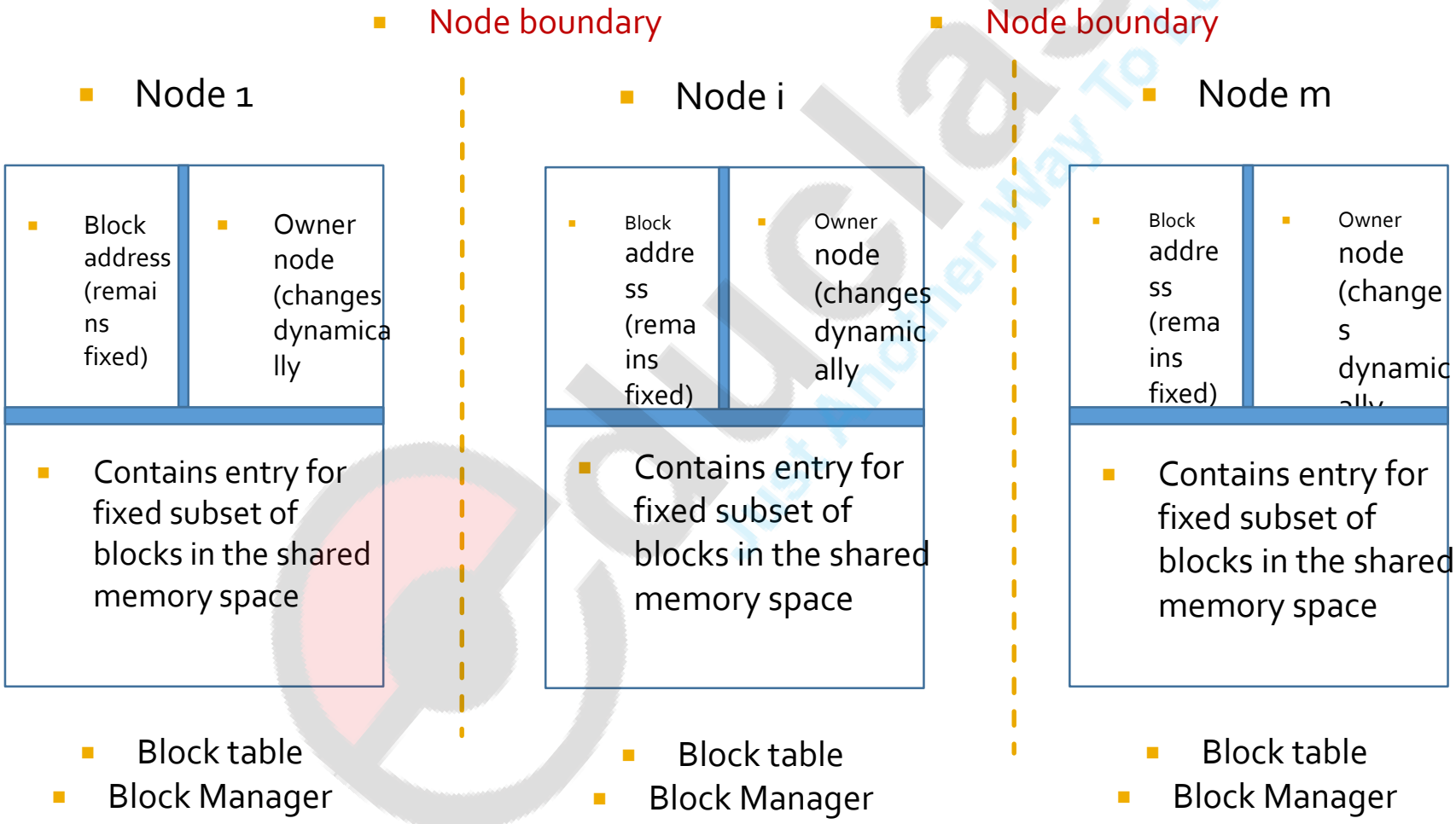
Centralized Server Algorithm



Fixed distributed server algorithm

- Is a direct extension of the centralized-server scheme
- It overcomes the problems of centralized server scheme by distributing the role of the centralized server
- In the scheme there is block manager on several nodes and each block manager is given a predetermined subset of data blocks to manage
- A mapping function describes the mapping data blocks to block manager and their corresponding nodes
- When a fault occurs, the mapping function is used by the fault handler to find out node whose block manager manages currently accessed block
- Then a request for the block is sent to the block manager of that node, block manager handles request in the same manner as in central server

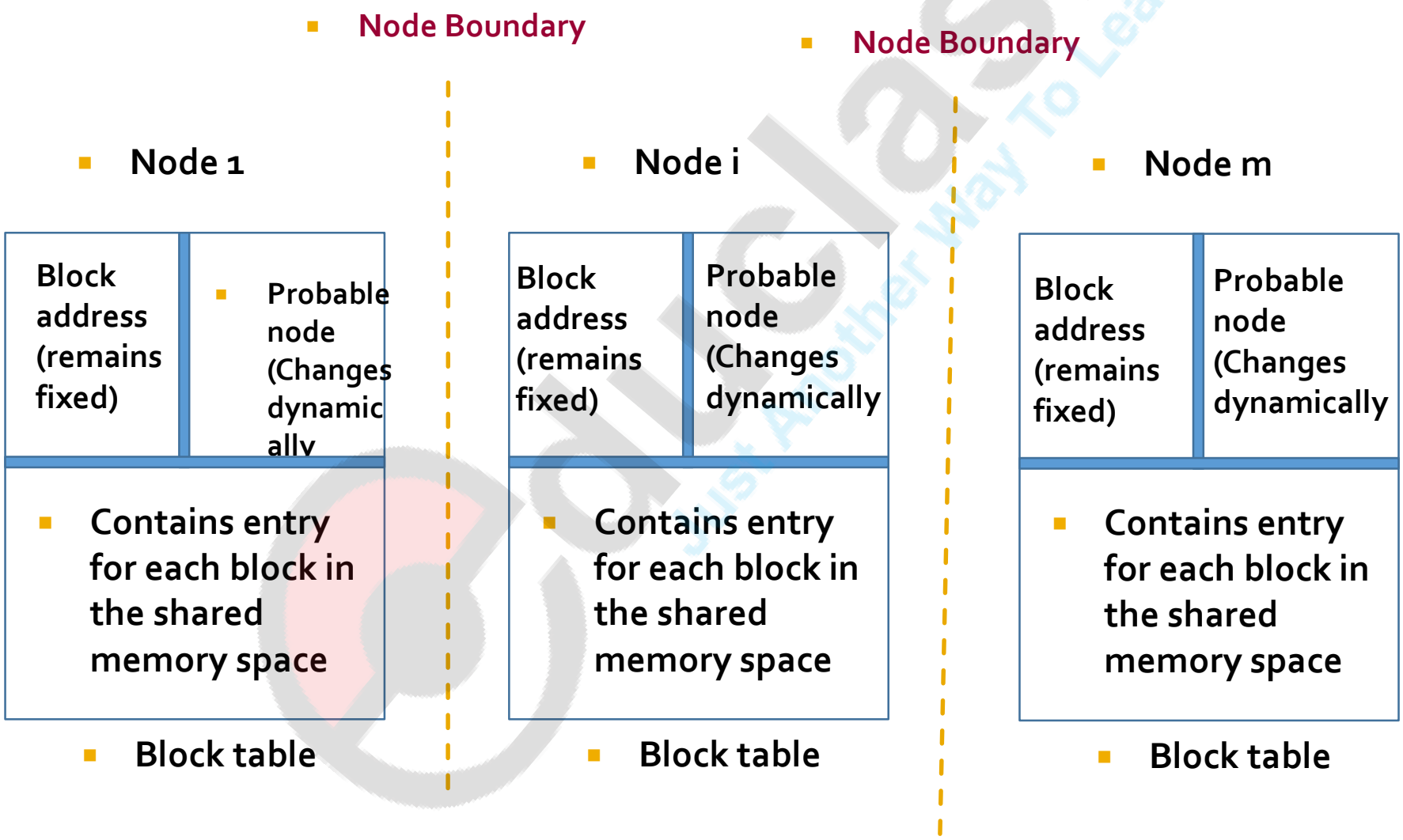
Fixed distributed server algorithm



Dynamic distributed server algorithm

- This scheme does not keep any block manager and attempts to keep track of the ownership information of all blocks in each node
- Each node has a block table that contains the ownership information for all the blocks in shared memory space
- However, the information contained in the ownership field is not necessarily correct at all times, but if incorrect, it at least provides the beginning of a sequence of nodes to be traversed to read the true owner node of a block; Hence it is called probable owner table
- When fault occurs, the faulting node (N) extracts from its local block table the node information stored in the probable owner field of the accessed block
- If that node is the true owner of the block, it transfers the block to node N and updates its local block table to node N
- Else it looks up its local block table and forwards the request to the entry in its block table and updates its block table to node N
- When node N receives the block, it becomes the new owner of the block

Dynamic distributed server algorithm



Replicated, Migrating Blocks (RMB)

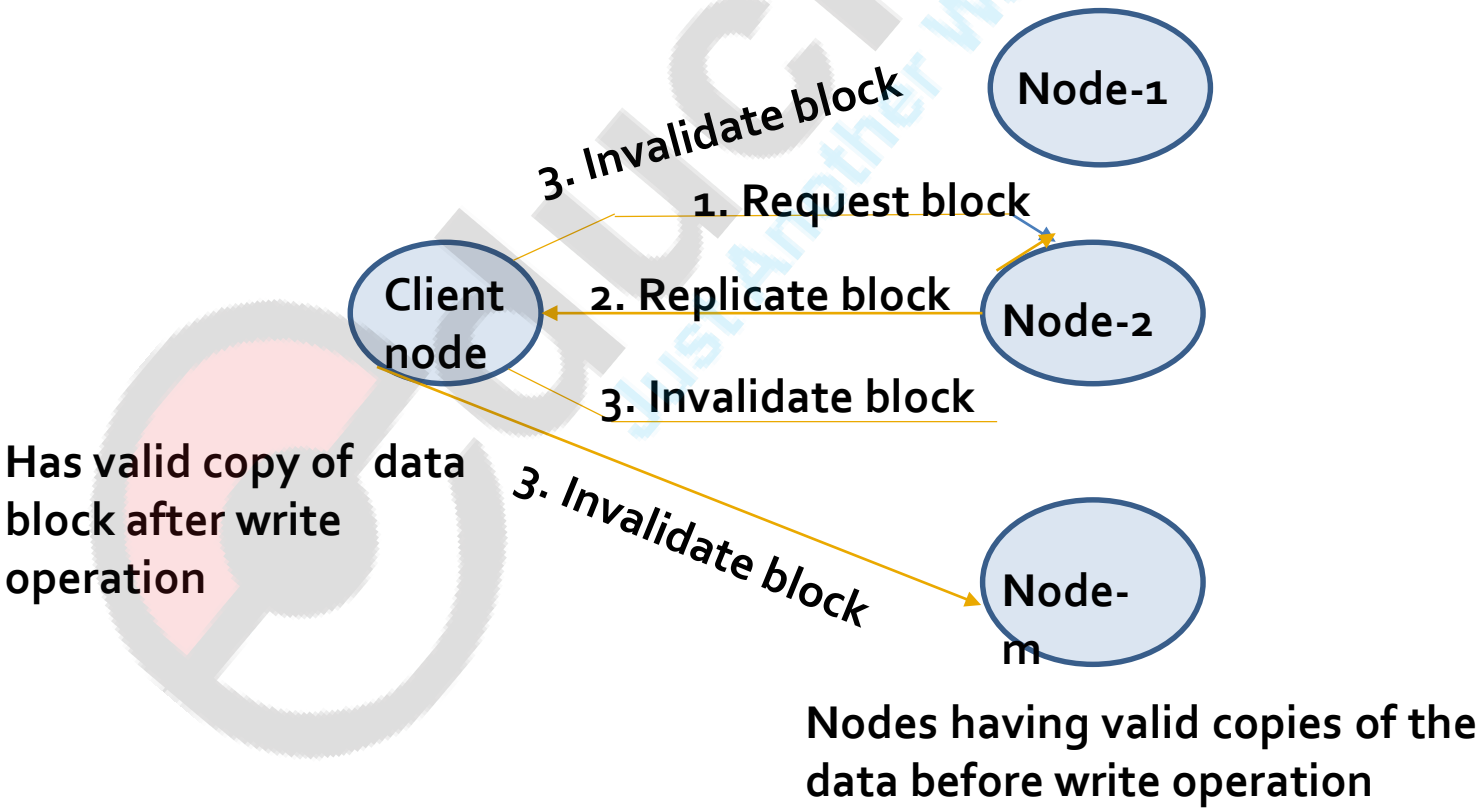
- Major issue with non-replication strategies is lack of parallelism because only the processes on one block can access data contained in a block at any given time
- To increase parallelism, virtually all DSM systems replicate blocks
- Replication increases parallelism but complicates memory coherence due to requirement of keeping multiple copies of the block consistent

Replicated, Migrating Blocks (Cont'd)

- Replication tend to increase the cost of write operations because for a write to a block, all its replicas must be invalidated or updated to maintain consistency
- The two basic protocols used for ensuring sequential consistency
  - Write invalidate: all copies of a piece of data except one are invalidated before a write can be performed on it
  - When write fault occurs at a node, its fault handler copies the accessed block from one of the block's current nodes to its own node, invalidates all other copies of the block by sending an invalidate message containing the block address to the nodes having a copy of the block, then updates the block
  - The node owns that block until the block ownership is relinquished to some other node

Write Invalidate

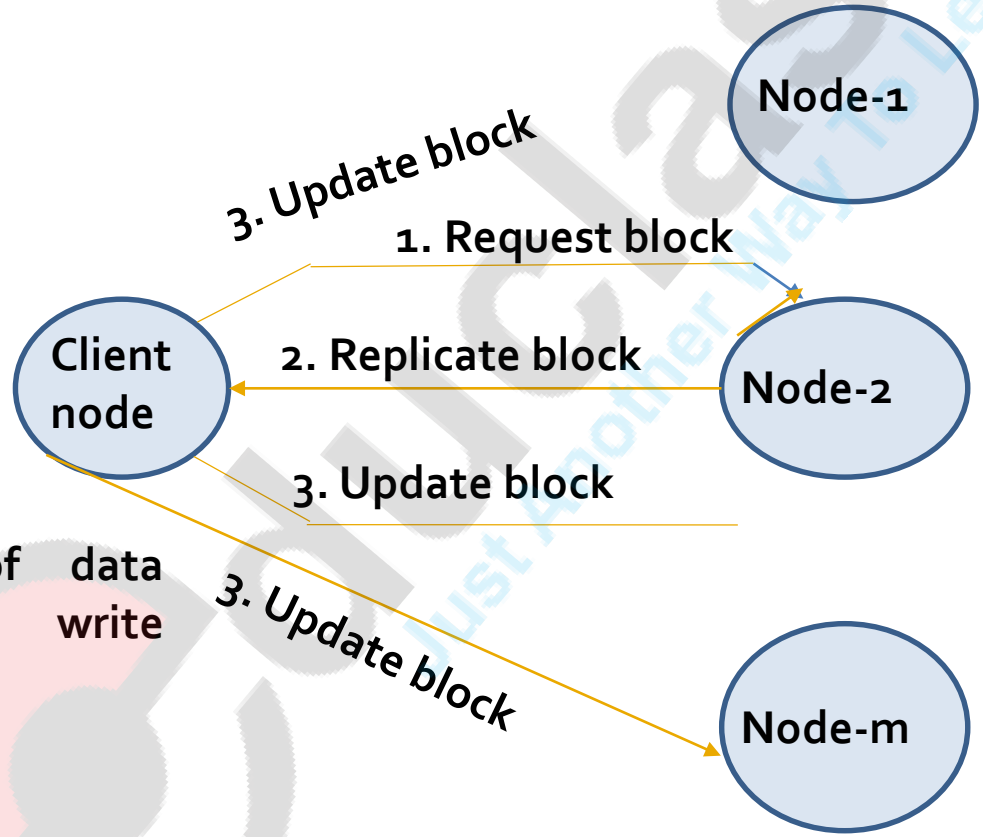
- If one of the nodes with invalid copy of a block, tries to perform memory access operation, a cache miss will occur and the fault handler of that node will have to fetch a valid copy of the block
- Therefore the scheme achieves sequential consistency



Write update

- In this scheme, a write operation is carried out by updating all copies of the data on which the write is performed
- 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, only then returns to the faulting instruction, but making it an expensive approach (fig. in next slide)
- It maintains the sequential consistency and it can be achieved by using a mechanism to totally order the write operations of all the nodes so that all processes agree on the order of writes
- One method is to use a global sequencer to sequence the write operations of all nodes

Write update



Has valid copy of data block after write operation

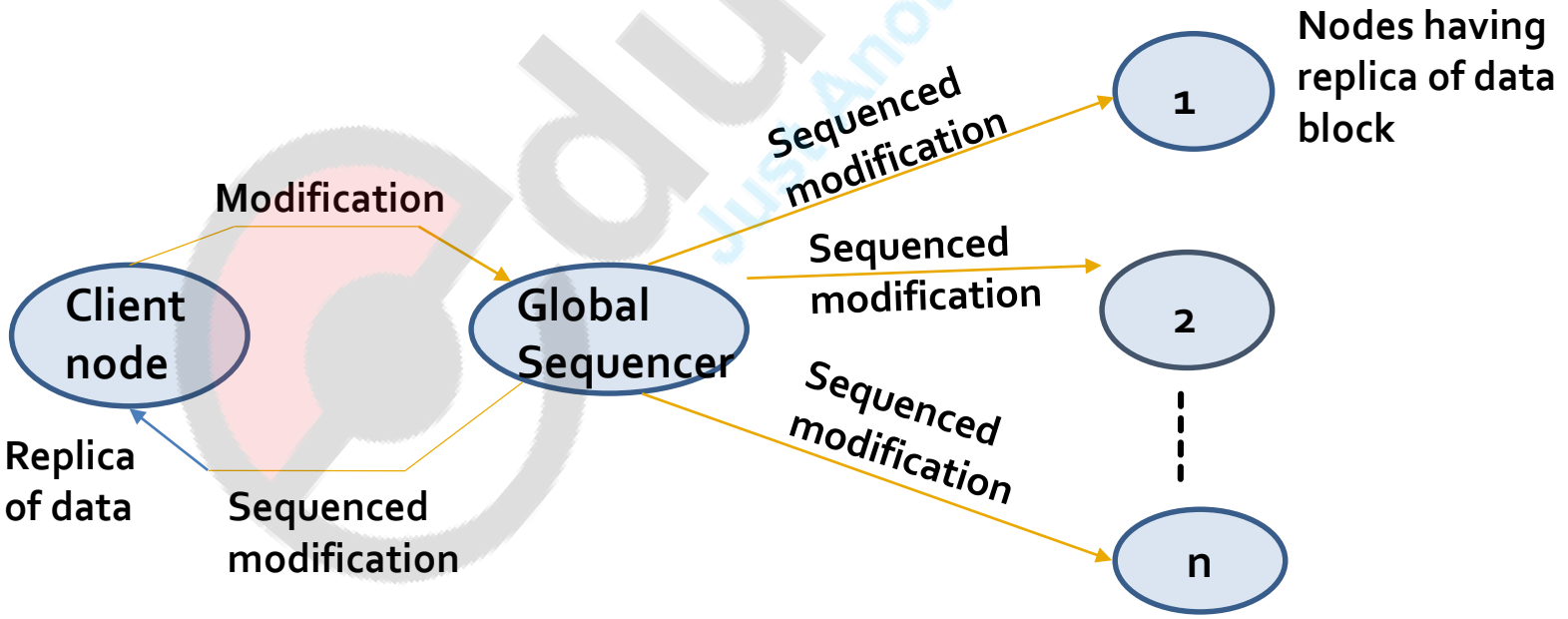
Nodes have valid copies of data before & after write operation

Global Sequencer

- Intended modification of each write operation goes to sequencer which assigns it the next sequence number and multicasts the modification with this sequence no. to all the nodes where a replica of the data block to be modified is located (fig. in next slide)
- When a new modification arrives at a node, its sequence number is verified with next expected one
- If the verification fails, either a modification was missed or received out of order, in which case it requests the sequencer for a retransmission of the missing modification
- The write update approach is very expensive for use with loosely coupled DSMs because it requires a network access on every write operation and updates all copies of the modified block

Global Sequencer (Cont'd)

- Write invalidate updates are done only when required. Hence most DSM systems use write-invalidate protocol
- In this there is status tag with each block indicating whether the block is valid, it is shared, read only, writable. Using this status information, read and write requests are carried out



Write Invalidate

- Using the status information, read and write operations are done as follows
  - Read Request
    - If local block available, data is valid request is satisfied by the local data
    - Otherwise the fault handler of the requesting node generates a read fault and obtains a copy of the block from a node having a valid copy of the block
  - Write request
    - If there is a local block containing the data and if it is valid and writable, the request is immediately satisfied
    - Otherwise fault handler of the requesting node generates a write fault and obtains valid copy of the block and changes its status to writable
    - A write fault causes invalidation of all other copies of the block

Data Locating in the RMB strategy

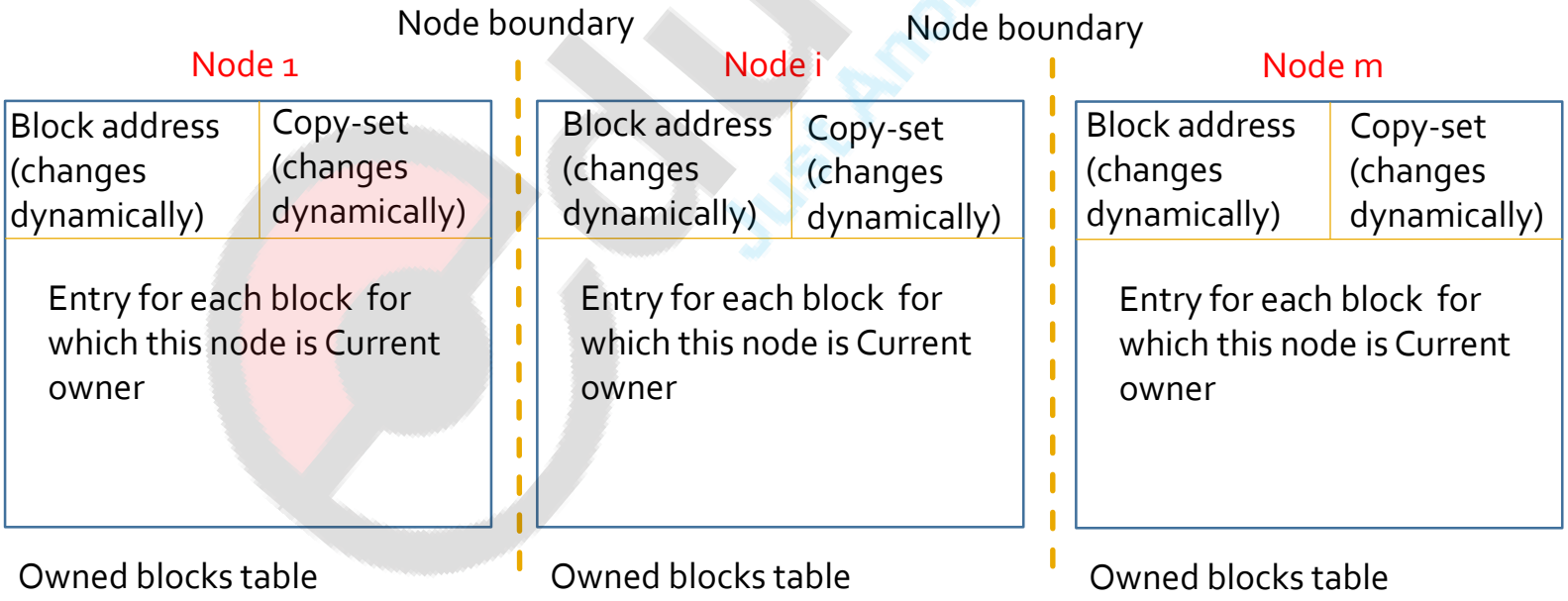
- Issues involved are
  - Locating the owner of a block
  - The owner of a block is the node that owns block namely, the most recent node to have write access to it
  - Keeping track of the nodes that currently have a valid copy of the block
- One of the following Algorithms may be used to address these two issues
- Broadcasting
  - Each node has an owned blocks table (fig. on next slide) and each entry of this table has Copy-set field that contains a list of nodes that currently have a valid copy of the corresponding block

Broadcasting

- Read Fault
  - When Read fault occurs, the faulting node N sends a broadcast read request on the network to find the owner of the required block
  - Owner sends block to node N & adds node N to its copy set
- Write fault
  - On write fault, the faulting node sends a broadcast write request on the network
  - Owner of the block relinquishes its ownership to node N and sends block & copy set to node N
  - On receipt of the block, becomes the new owner of the block, and N sends invalidation message to all nodes in the copy set & initializes copy set to Null indicating no other copies of block (all are invalidated)

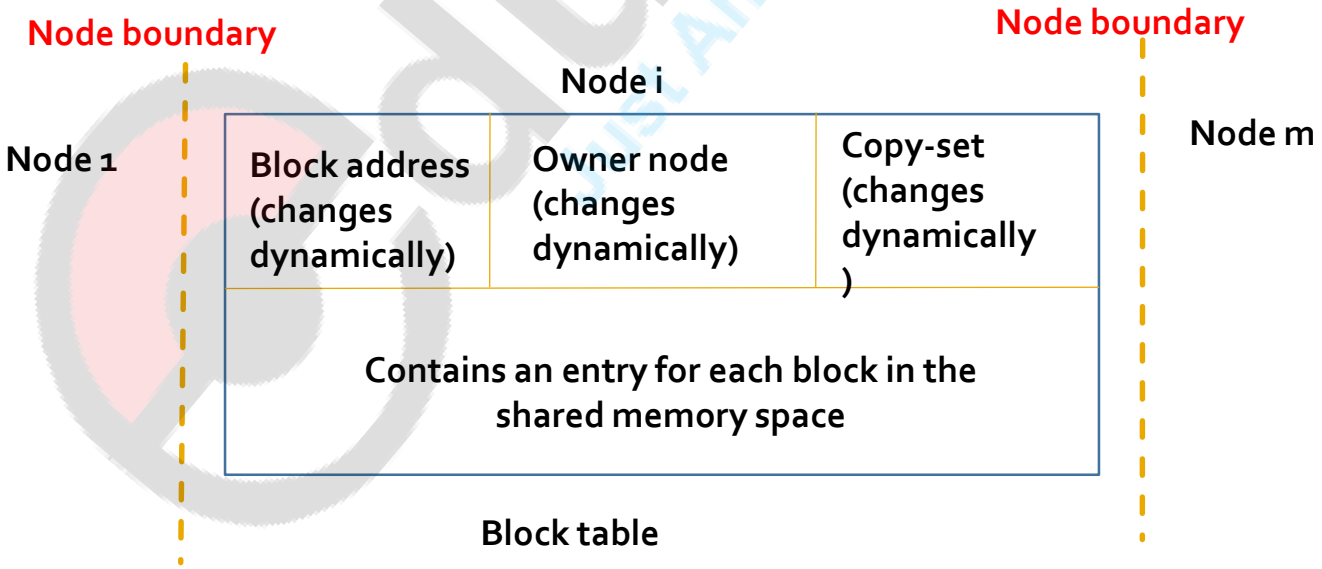
Broadcasting

- Disadvantages
  - The major disadvantage of this algorithm is that it does not scale well
  - When a request is broadcast all nodes must process broadcast request leading to communication bottleneck
  - Network latency may also lead to accesses taking long time



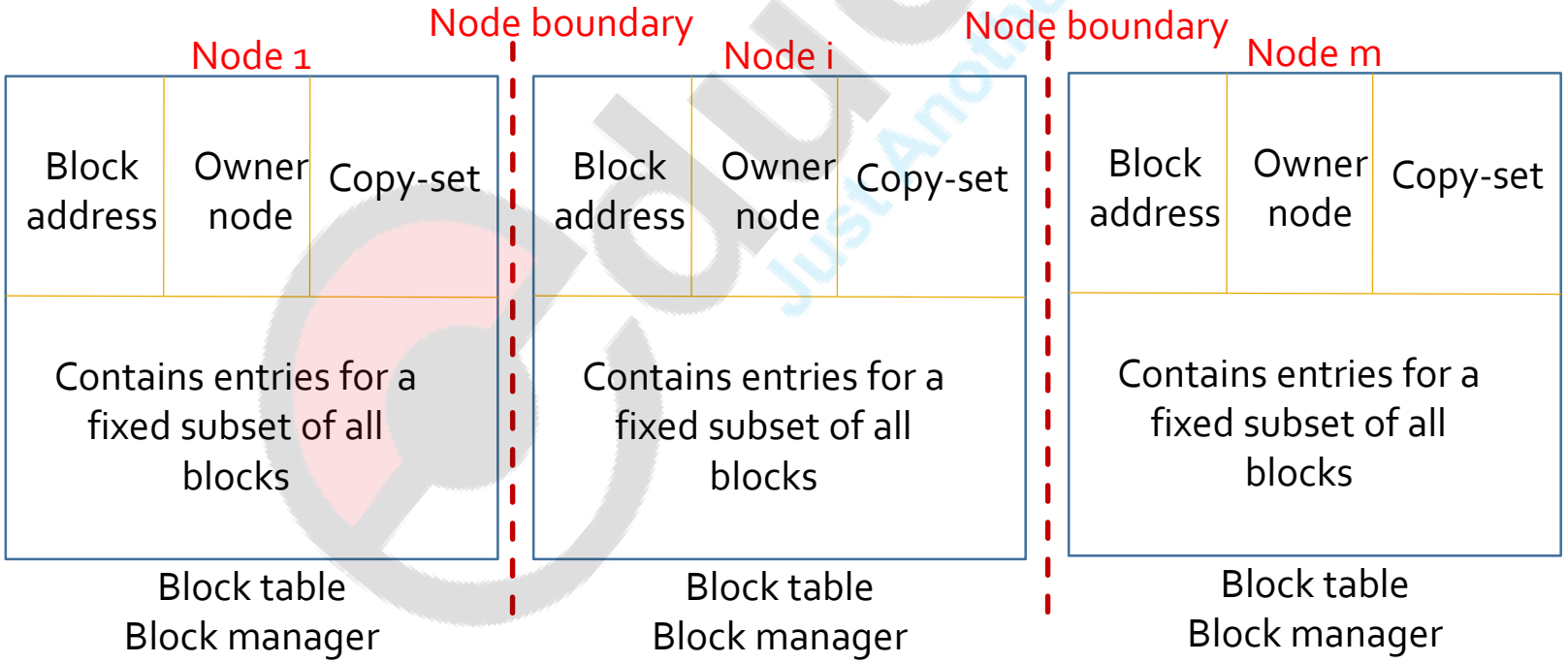
Centralized Server algorithm

- Similar to NRMB Strategy But the central table has an owner node field
- Read fault
  - Server 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

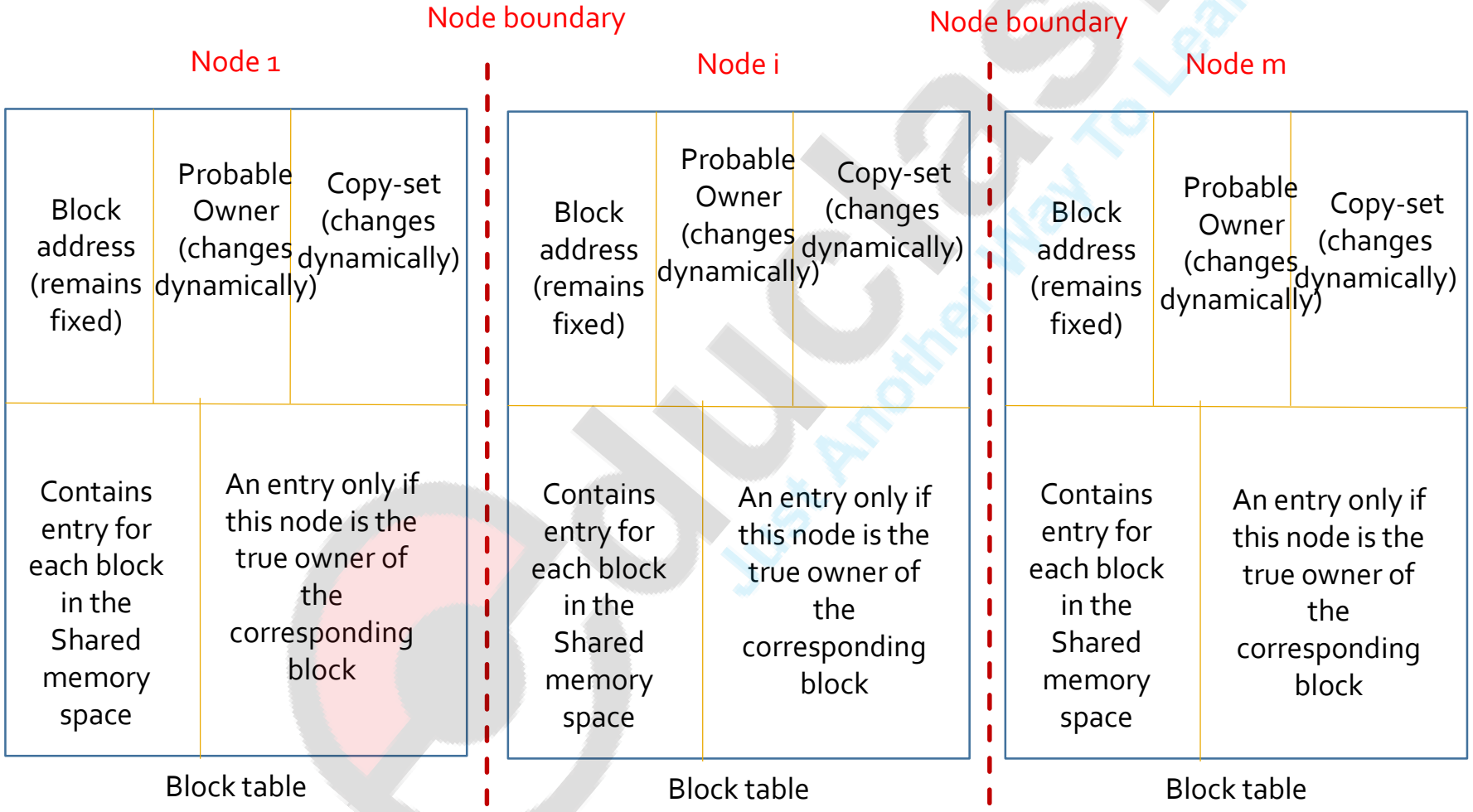
- Extension of centralized server with a block manager on several nodes and each managing predetermined subset of blocks
- A mapping function is used to map a block to a particular block manager
- When a fault occurs the mapping function is used to locate the block manager and is processed same as centralized server algorithm



Dynamic distributed server algorithm

- Works similar to dynamic distributed server algorithm of NRMB strategy
- Each node has block table with an entry for all blocks in the shared memory space (fig in next slide)
- Each entry in the table has a probable owner field gives the node hint of the owner of the corresponding block
- It also has a copy set field that has the list of nodes having a valid copy of the block
- 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

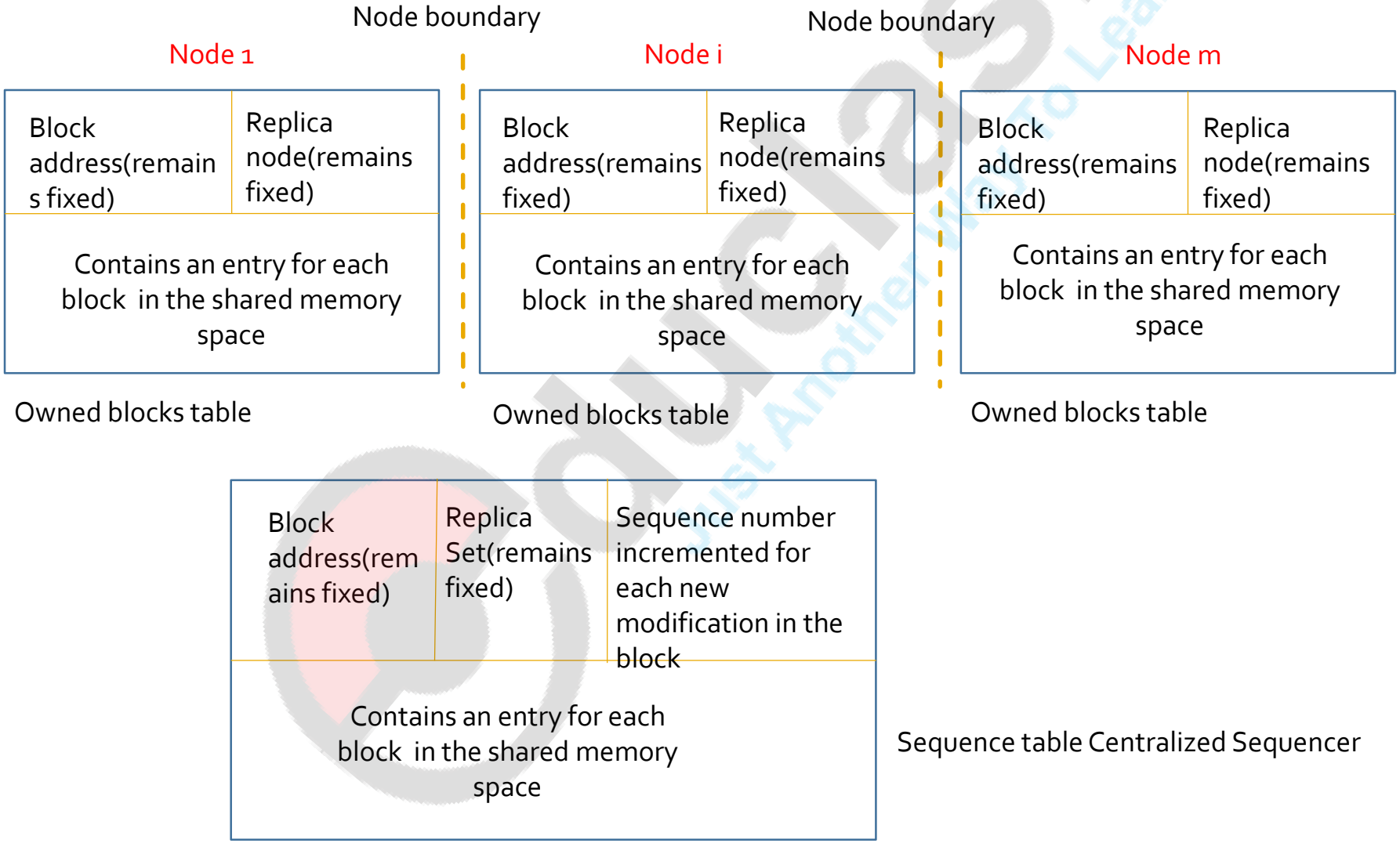
Dynamic distributed server algorithm



Replicated, Nonmigrating Blocks

- A shared memory block may be replicated at multiple nodes of the system, but the location of each replica is fixed
- A read / write access to a memory address is carried out by sending the access request to one of the nodes having a replica of the block containing the memory address
- All replicas of a block are sequential consistent by the use of Write-update protocol and a global sequencer
- Data locating characteristics
  - The location of replica is fixed
  - All replicas are kept consistent
  - Read request serviced by any node having replica of block.
  - All write request are sent to global sequencer to ensure sequential consistency. Sequencer multicasts modification with sequence number to all nodes of replica set.

Replicated, Nonmigrating Blocks (Cont'd)



Replicated, Nonmigrating Blocks (Cont'd)

- The block table of a node has an entry for each block in the shared memory
- The sequence table also has an entry for each block in the shared memory space
- Each entry in the sequence table has three fields – a field containing the block address, a replica set field containing a list of nodes having a replica of the block, and a sequence number field that is incremented by 1 for every new modification performed on the block

Replicated, Nonmigrating Blocks (Cont'd)

- For performing a read operation on a block, the replica location of the block is extracted from the local block table and the read request is directly sent to that node
- A write operation on a block is sent to the sequencer
- The sequencer assigns the next sequence number to the requested modification
- It then multicasts the modification with the sequence number to all the nodes listed in replica list field of entry for the block
- The write operations are performed at each node in sequence number order
- Note to prevent all read operations on a block getting service by a same replica node, as for as practicable, the block table of different nodes should have different replica locations in the entry corresponding to the block
- This will help in evenly distributing read operations on the same block emerging from different nodes

MUNIN DSM-
THE RELEASE CONSISTENCY MODEL

MUNIN : Release Consistent DSM

- The release consistent is also promising and attractive for use in DSM systems
- Hence, as a case study let us consider Munin system based on release consistency architecture
- Structure of Shared Memory Space
  - Shared memory space of Munin is structured as a collection of shared variables including program data structures
  - Shared variables are declared with keyword shared so that compiler can recognize them
  - A programmer can annotate shared variable with one of the standard annotation types (annotation types discussed later)

MUNIN : Release Consistent DSM (Cont'd)

- Implementation of Release Consistency
  - In the discussion of release consistency applications we have seen that they must be modeled around critical sections
  - Therefore DSM system that supports release consistency must have mechanisms and programming language constructs for critical sections
  - Munin provides two such synchronization mechanisms namely a locking mechanism and barrier mechanism
  - Lock mechanism uses lock synchronization variables with *acquireLock* and *releaseLock* as its primitives for accessing these variables
  - The *acquireLock* primitive with a lock variable as its parameter is executed by a process to enter a critical section and *releaseLock* primitive with the same lock variable as its parameter is executed by the process to exit from the critical section

MUNIN : Release Consistent DSM (Cont'd)

- To ensure release consistency, write operations on shared variables must be performed only within critical sections
- But read operations of shared variables can be done inside or outside a critical section
- Modifications done to shared variables within critical sections are sent to other nodes having a replica of the shared variable only when the process making update exits from the critical section
- Whether the lock is on local or remote node, if it is free it is granted to the requesting process, else it is added to end of the queue of processes waiting to acquire a lock variable
- When the present owner releases the lock, it is given to next process in queue

MUNIN : Release Consistent DSM (Cont'd)

- The Barrier mechanism uses *barrier* synchronization variables with a *waitAtBarrier* primitive for accessing these variables
- Barriers are implemented by using centralized barrier server mechanism

Annotations for Shared Variable

- For further improvement in performance, the Munin defines several standard annotations for shared variables and uses a different consistency protocol for each type that is most suitable for that type
- i.e., the consistency protocols in this approach are applied at the granularity of individual data items
- The standard annotations and consistency protocol for each type of variables are as follows

Annotations for Shared Variable (Cont'd)

- Read only
  - Shared data variables annotated as read only are immutable data items
  - These variable are read but never written after initialization and hence the question of consistency does not arise
  - As these variable are never modified, their average read cost can be reduced drastically by freely replicating them on all nodes from where they are accessed without consistency problem to reduce the cost of communication
  - When reference to such a variable causes a network page fault, the page having the variable is copied to the faulting node from one of the nodes already having the copy of the page
  - Read only variable are protected by MMU hardware and an attempt to write to such a variable causes a fatal error

Annotations for Shared Variable (Cont'd)

- Migratory
  - Shared variables that are accessed in phases, where each phase corresponds to a series of accesses by a single process can be annotated as migratory variables
  - The locking mechanism is used to keep migratory variables consistent
  - These are protected by lock synchronization variables and are used within critical sections
  - To access a migratory variable, a process first acquires a lock for the variable for some time, and then releases the lock, when it has finished using it
  - At a time, the system allows only a single process to acquire a lock for migratory variable

Annotations for Shared Variable (Cont'd)

- If a page fault occurs while waiting for lock for a migratory variable, the page is migrated to the faulting node from the node that is its current owner
- NRMB strategy used in this case. i.e., pages migrate from one node to another on a demand basis, but pages are not replicated
- Hence only one copy of the page containing migratory variable exists in the system

Annotations for Shared Variable (Cont'd)

- Write Shared
  - A programmer may use this annotation with a shared variable to indicate to the system that the variable is updated concurrently by multiple processes, but the different processes do not update the same parts of the variable
  - For example in a matrix, different processes may be updating concurrently different rows / columns elements, with each process updating only the elements of one row/column
  - Munin avoids false sharing problem of write shared variable by allowing them to be concurrently updated by multiple processes
  - A write shared variable is replicated on all nodes where a process sharing is located, i.e., when access to such a variable causes a network page fault to occur, the page having the variable is copied in to the faulting node from one of its current nodes

Annotations for Shared Variable (Cont'd)

- If the access is write access, the system first makes a copy of the page (called twin page), then updates original page
- When the page is released, the system performs a word-by-word comparison of the original page and twin page and sends the difference to all the nodes having a replica of the page
- When a node receives the differences of the modified page, the system checks if the local copy of the page was also modified.
- If not local copy of the page is updated by incorporating the received differences in it

Annotations for Shared Variable (Cont'd)

- Producer Consumer
  - Shared variables written by only one process & read by a fixed set of processes can be annotated as producer-consumer type
  - Munin uses "eager object movement" mechanism for this type of variable
  - In this mechanism a variable is moved from the producer's node to the nodes of the consumers in advance of when they are required so that no network page fault occurs
  - The producer may send several updates together by using locking mechanism, where it acquires a synchronization lock, makes several updates on the variable and then releases the lock when the variable or the updates are sent to the nodes of consumer processes

Annotations for Shared Variable (Cont'd)

- Result
  - This is just the opposite of producer-consumer variables as they are written by multiple processes but read by only one process
  - Different process write to different parts of the variable that do not conflict
  - Variable is read only when all parts written
  - For example in an application there may be different "worker" processes to generate and fill the elements of each row/column of a matrix and once the matrix is complete, it may be used by the "master" process for further processing

Annotations for Shared Variable (Cont'd)

- Reduction
  - Shared variable that must be atomically modified may be annotated to be of reduction type
  - For example in parallel computation application, a global minimum must be automatically fetched and modified if it is greater than the local minimum
  - In Munin the reduction variables are always modified, by being locked →Acquire lock, read, update, release lock
  - For better performance, a reduction variable is stored at a fixed owner that receives updates to the variable from other processes, synchronizes the updates received from different processes, performs the updates on the variable and propagates the updated variable to its replica locations

Annotations for Shared Variable (Cont'd)

- Conventional
  - Shared variables which do not fall into any of the above class are conventional variables
  - Release consistency of Munin is used to maintain the consistency of replicated conventional variables
  - The invalidation protocol is used in this case to ensure that no process ever reads a stale version of a conventional variable

REPLACEMENT STRATEGY

Replacement Strategy

- In DSM systems that allow shared-memory blocks to be dynamically migrated/replicated the following issues need to be addressed when the available space for caching shared data fills up at a node
  - Which block should be replaced to make space for a newly required block
  - Where should the replaced block be placed
- Which block to replace?
- This has been studied extensively in shared memory multiprocessor systems they fall into two categories
  - Usage based versus non-usage based (LRU/ FIFO)
    - Usage based algorithms keep track of the history of usage of a cache line or page and use this information to make replacement decisions

Replacement Strategy (Cont'd)

- LRU fall in this category
- Non usage based algorithms are like FIFO and Rand

- Fixed space versus variable cache space
  - Fixed space algorithms assume that the cache size is fixed while the variable-space algorithms are based on the assumption that the cache size can be changed dynamically depending on the requirement
  - Usage based-fixed space algorithms are more suitable for DSM when compared to variable-space algorithms
  - However DSM uses some type of priority mechanism instead of simple LRU system

Replacement Strategy (Cont'd)

- In DSM each memory block of node is classified into one of the following five types
  - Unused - free memory block that is not currently being used
  - Nil - Invalidated block
  - Read-only – Node has read access right only
  - Read-owned - owner of the Node with only read access right
  - Writable – Node has write access permission
- Based on these the following replacement priority is used
  - Both unused and nil blocks have the highest replacement priority
    - The nil block may have been recently accessed block, i.e., the reason simple LRU is not sufficient
    - i.e., they will be replaced first if a block is required

Replacement Strategy (Cont'd)

- The read only blocks have the next replacement priority as the owner of the block will a copy of this block
- Read-owned and writeable blocks for which replica(s) exist on some other node have the next replacement priority as it is sufficient to pass the ownership to one of replica nodes
- Read-owned and writable block for which only this node has a copy have the lowest priority as it involves transfer of the ownership and the block to some other node

- Where to place a replaced block
  - Once memory block is selected for replacement, it should be ensured that if there is some useful information in the block, it should not be lost
  - For example read only or nil data can be discarded without the loss of data

Replacement Strategy (Cont'd)

- However, discarding read-owned or writeable block can lead to loss of data
- Hence care must be taken to store them some where before discarding them
- Using secondary store (local disk): the block is simply transferred to the local disk
- Using the memory space of other nodes may be more efficient

THRASHING

Thrashing

- Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another, compared to time spent doing the useful work of executing application processes
- Thrashing can occur in the following situations
  - When interleaved data accesses made by processes on two or more nodes causes a data block to move back and forth from one node to another in quick succession (ping-pong effect)
  - When blocks with read-only permission are repeatedly invalidated soon after they are replicated
    - These situations indicate poor (node) locality in reference
  - If not handled properly, thrashing degrades system performance considerably

Thrashing (Cont'd)

- Providing application-controlled locks
  - Locking data to prevent other nodes from accessing data for short period of time can reduce thrashing
  - An application controlled lock can be associated with each data block to implement this method
- Nailing a block to a node for a minimum amount of time t
  - Disallow a block to be taken away from a node until a minimum amount of time t elapses after it has been allocated to the node
  - The time t can be fixed statically or dynamically chosen on the basis of past access patterns
  - The main drawback of this scheme is to fix the value for t
  - One way is to tune the value of t based on past access pattern of the block
  - Another factor that may be used for deciding the value of t for a block is the length of the queue of processes waiting to access that block

Thrashing (Cont'd)

- Tailoring the coherence algorithm to the shared data usage patterns
  - Thrashing can also be minimized by using different coherence protocols for shared data having different characteristics
  - e.g., The coherence protocol used in Munin for write shared variables avoids the false sharing problem, which ultimately results in the avoidance of thrashing

University Questions

- Explain various data locating strategies used in DSM system that uses replicated migration blocks (RMB) strategy.
- Thrashing (Short Note)
- Differentiate between Strong Consistency Model and Casual Consistency Model.
- Name different consistency models. How is the sequential consistency model implemented in DSM?
- False Sharing (Short Note)
- Explain the types of strong consistency models. How they differ from weak consistency models?
- How does a centralized manager algorithm finds data location in Distributed Shared Memory(DSM)?
- What is Memory Consistency in a Distributed Shared Memory(DSM)?
- What is difference between sequential consistency and release consistency? State their relative advantages?
- Release Consistency (Short Note)

- What is false sharing? When is it likely to occur? Can it be completely eliminated?
- Give relative advantages and disadvantages of using large block size and small block size in the design of block-based Distributed Memory System.
- Differentiate between Strong Consistency and Casual Consistency Model.
- Differentiate between PRAM Consistency model and Processor Consistency Model.
- What are the differences between Replication and Caching.
- Explain various data locating strategies used in DSM system that uses Non-replicated migration blocks (NRMB) strategy.
- Compare and contrast replicated migrating and non migrating block strategies used in DSM systems.
- Munin DSM (Short Note)
- Explain different annotations for shared variables in Munin DSM.