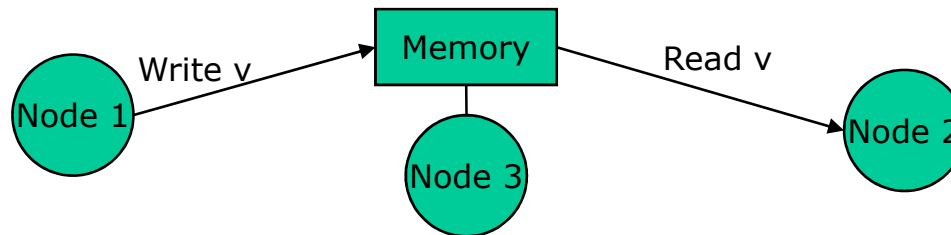


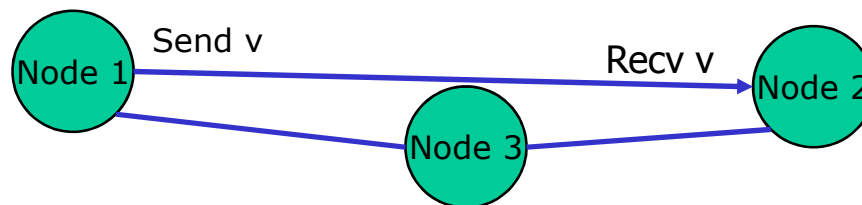
Distributed Shared Memory

Information Sharing

- Using a *Shared Memory*
 - A node **writes** information in the *memory*
 - Another node **reads** information from the *memory*



- Using *Message Passing*
 - A node **sends** a *message* to another node
 - The second node **receives** the *message* from the other

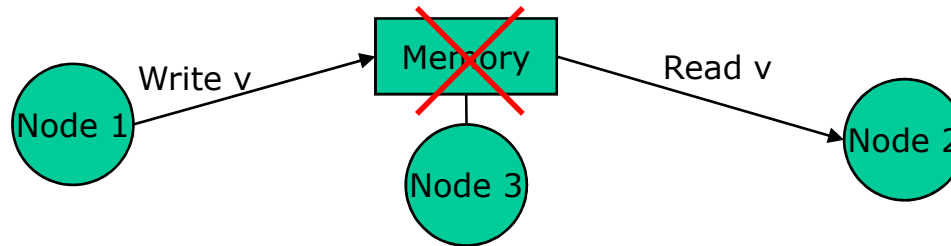


Information Sharing

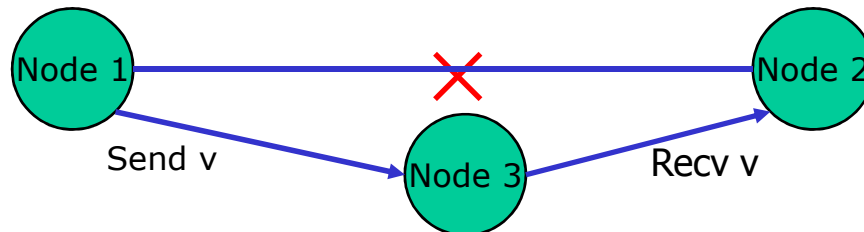
- *Shared Memory* is **easy** to use
 - If information is **written**, collaboration progresses!
- *Message Passing* is **difficult** to use
 - To which node the information should be **sent**?

Information Sharing

- *Shared Memory* is **failure-prone**
 - Communication trusts on memory availability



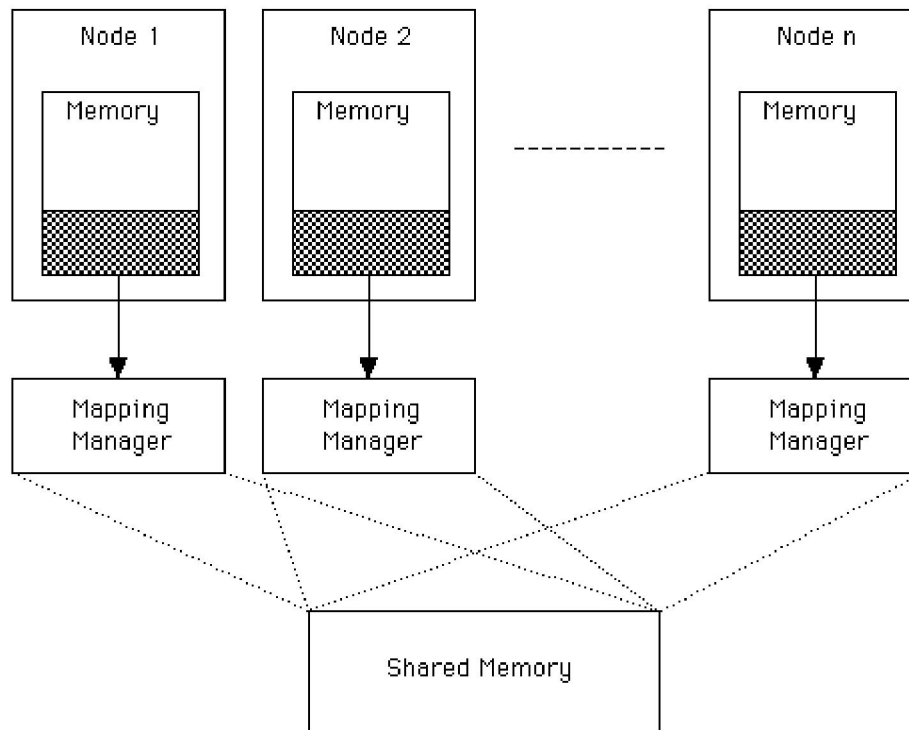
- *Message-Passing* is **fault-tolerant**
 - As long as there is a way to route a message



Introduction

- In 1986, Kai Li published his PhD dissertation entitled, “Shared Virtual Memory on Loosely Coupled Microprocessors,” thus opening up the field of research that is now known as Distributed Shared Memory (DSM) systems.
- DSM provides a virtual address space that is shared among all nodes in the distributed system. All the nodes perceive the same illusion of a single address space.
- Programs access DSM just as they do locally.
- A process accessing a shared object gets in touch with a *Mapping Manager/Memory Mapping Manager(MMM)* who maps the shared memory address to the physical memory.

Introduction



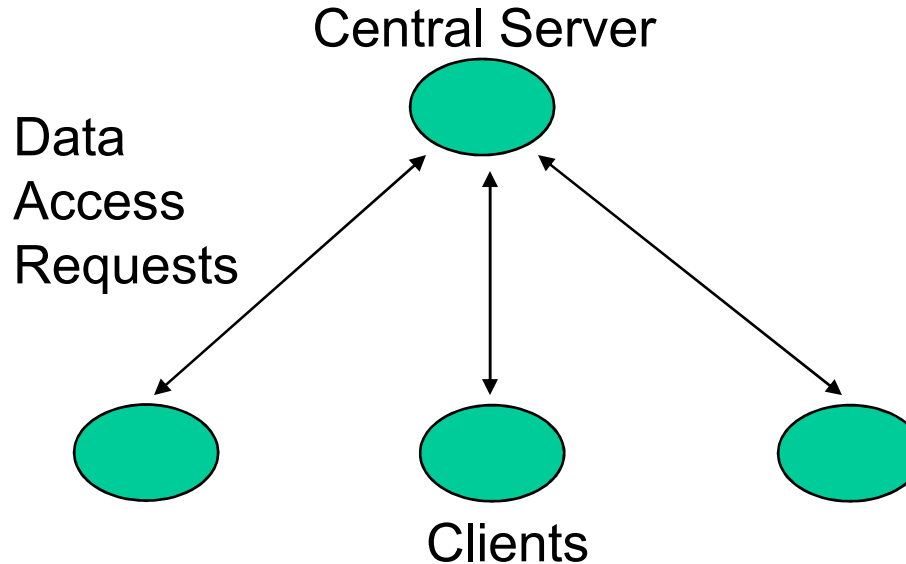
Introduction

- **Mapping manager**: a layer of software, perhaps bundled with the OS or as a runtime library routine.
- Other than mapping, their chief responsibility is to **keep the address space consistent** at all times; that is, the value returned by a read operation is always the same as the value written by the **most recent write operation** to the same address.
- Any processor can access any memory location in the address space directly.
- DSM offers the power of **parallel computing** using multiple processors as well as a single system memory view which makes the programming task easy.
- To improve performance and get correct result of computation, distributed shared memory systems designers should choose the proper paradigm of memory coherence semantics and consistency protocols.

DSM Implementation Algorithms

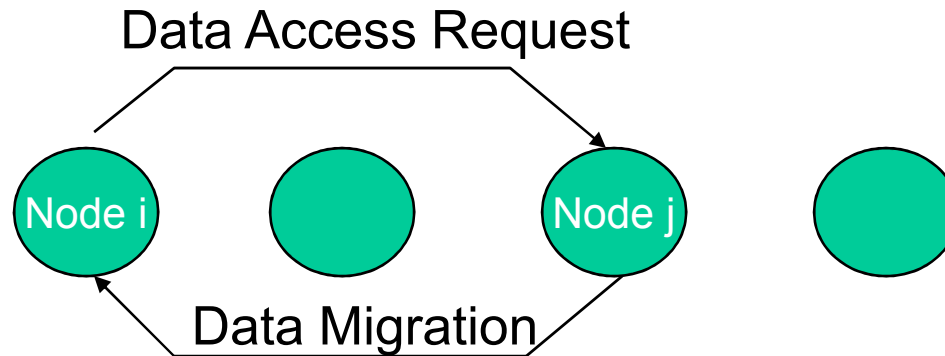
- DSM Implementation Issues:
 - Keeping track of remote data locations
 - Overcoming/reducing communication delays and protocol overheads when accessing remote data.
 - Making shared data concurrently available to improve performance.
- DSM Implementation algorithms:
 - Central-server
 - Data migration
 - Read-replication
 - Full-replication

Central Server Algorithm



- Central server maintains all the shared data.
- It services read/write requests from clients, by returning data items.
- Timeouts and sequence numbers can be employed for retransmitting requests (which did not get responses).
- Simple to implement, but central server can become a bottleneck.

Migration Algorithm



- Data is shipped to the requesting node, allowing subsequent accesses to be done locally.
- Typically, whole block/page migrates to help access to *other* data.
- Susceptible to *thrashing*: page migrates between nodes while serving only a few requests.
- Alternative: Mirage system – uses tunable parameter that determines the duration for which a node can possess a shared data item before it can be migrated to other node(e.g., the Mirage system).

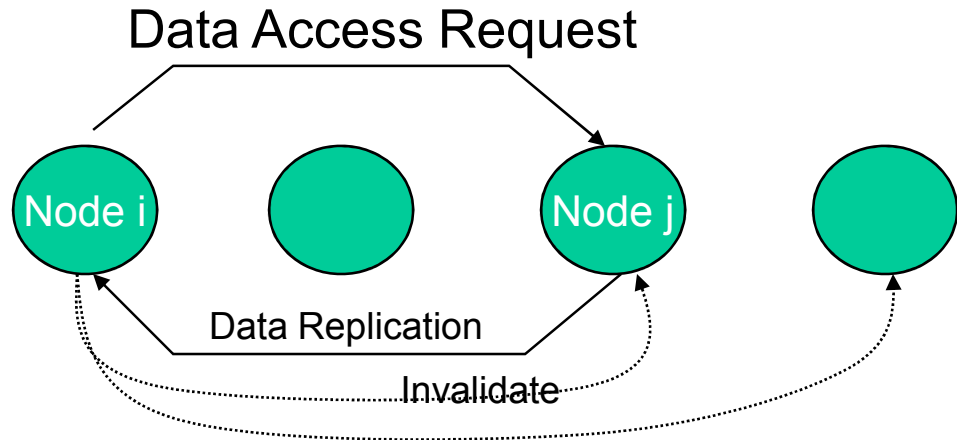
Migration Algorithm

- Migration algorithm can be combined with virtual memory.
- (e.g.,) if a page fault occurs, check memory map table.
- If map table points to a remote page, migrate the page before mapping it to the requesting process's address space.
- Several processes can share a page at a node.
- Locating remote page:
 - Use a server that tracks the page locations.
 - Use *hints* maintained at nodes. Hints can direct the search for a page toward the node holding the page.
 - Broadcast a query to locate a page.

Read-replication Algorithm

- Extend migration algorithm: replicate data at multiple nodes for read access.
- Write operation:
 - invalidate all copies of shared data at various nodes.
 - (or) update with modified value

Write Operation in Read-replication Algorithm :

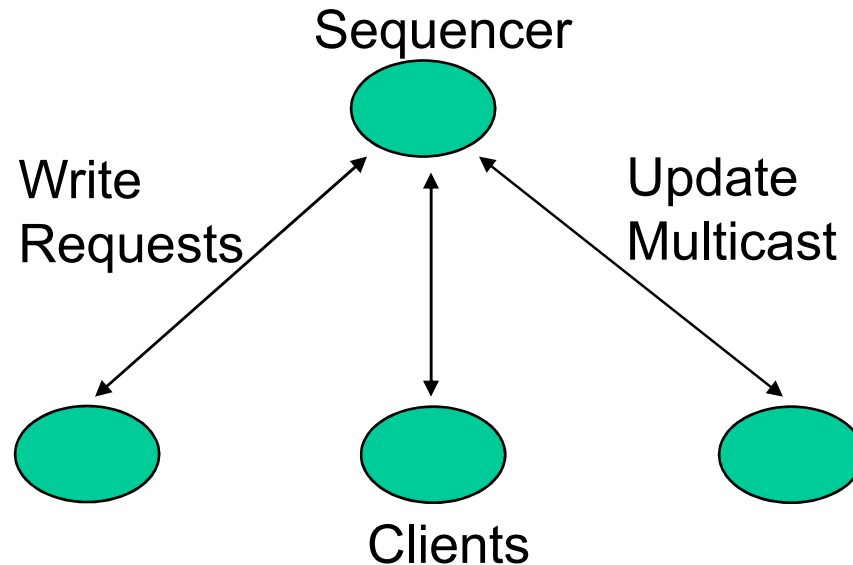


- DSM must keep track of the location of all the copies of shared data.
- Read cost low, write cost higher.

Full-replication Algorithm

- Extension of read-replication algorithm: allows multiple sites to have both read and write access to shared data blocks.
- One mechanism for maintaining consistency: gap-free sequencer.

Write Operation in Full-replication Algorithm :



Full-replication Sequencer

- Nodes modifying data send request (containing modifications) to sequencer.
- Sequencer assigns a sequence number to the request and multicasts it to all nodes that have a copy of the shared data.
- Receiving nodes process requests in order of sequence numbers.
- Gap in sequence numbers? (modification requests might be missing)
- Missing requests will be reported to sequencer for retransmission.

Design Issues of DSM

- Granularity
- Structure of shared-memory space
- Memory coherence and access synchronization
- Data location and access – easy to locate and retrieve the data
- Replacement strategy – which block and where to place?
- Thrashing – can be addressed using application controlled locks, share block for min amount of time, use coherence algorithms
- Heterogeneity – data conversion(byte ordering, floating point representation), block size selection

Granularity

- Granularity refers to the block size of a DSM system, the unit of sharing and the unit of data transfer across the network.
- Possible units are a few words, a page, or a few pages.

Factors influencing block size selection

- **Paging overhead:** a process is likely to access a large region of its shared address space in a small amount of time. So paging overhead is less for large block sizes as compared to the paging overhead for small block sizes.
- **Directory Size:** Information regarding blocks is maintained in directory. The larger the block size, the smaller the directory which results in reduced directory management overhead.

Granularity

- **Thrashing:** When data items in the same data block are being updated by multiple nodes at the same time, no real work can get done. A DSM must use a policy to avoid this situation (usually known as thrashing).
- **False sharing:** False sharing occurs when two unrelated variables (each used by different processes) are placed in the same page. The page appears shared, even though the original variables were not.
 - Conventional programming usually requires processes to gain exclusive access to a page before it starts modification.
 - False sharing leads to a **race condition** where multiple processors seek for ownership of a page while actually they are modifying totally different sets of data.


Structure of shared-memory space

- It defines the abstract view of the shared-memory space to be presented to the application programmers of a DSM system
 - It may appear as a storage for words or for data objects or for database
- 1. No structuring**
 - Linear array of words
 - Easy to choose any page size
 - Used for all applications
 - Simple and easy to design
 - 2. Structure by data type**
 - shared-memory is structured either as a collection of objects or as a collection of variables
 - 3. Structuring as a database**
 - Memory is ordered as an associative memory (addressed by content instead of name/address) called tuple space

Consistency Problems in Distributed Shared Memory

- Cache coherence problem:
- It occurs when processors get different view of memory when accessing and updating at different time.
- For example, if two processors, X and Y, cache two different variables, A and B, locally, the value in the cache may not be coherent when one of them modified the value of the variable and the other processor is not notified.
- This **memory inconsistency** may leads to serious computation problems.

Consistency Problems in Distributed Shared Memory (Contd..)

Time	Processor X	Processor Y
	$A = 0$	$B = 0$
	$A = 1$	$B = 1$
	$A = A + B$	
		$B = A + B$

- 1) Variables A and B are initialized to 0.
- 2) A and B are updated to the value 1
- 3) Because of the lack of memory coherency mechanism, X still thinks that B is 0 and Y still thinks that A is 0
- 4) Finally, X and Y will both think A and B to be 1, which is the wrong answer. The correct answer for A and B should be 2

Memory Coherence

- Memory coherence problem:
- Value returned by a read operation is the one expected by a programmer (e.g., the value of the latest write operation).
- 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.
- A consistency model is essentially a **contract between the software and the memory**. **If the software agrees to obey certain rules, the memory promises to work correctly.**

Consistency Models

- Consistency requirements vary from application to application.
- Applications that depend on a stronger consistency model may not perform correctly if executed in a system that supports only weaker consistency model.
- If a system supports the stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true.

Strict Consistency Model

- This is the strongest form of memory coherence.
- Value returned by a **read operation** on a memory address is always the same as the **value written** by the **most recent write operation** to that address. All writes become visible to all processes.
- Implementation of the strict consistency model requires the existence of an **absolute global time** so that memory read/write operations can be **correctly ordered** to make the meaning of “most recent” clear.
- Absolute synchronization of clocks of all the nodes of a distributed system is not possible.

Sequential Consistency Model

- All processes see the **same order** of all **memory access operations** on the shared memory.
- The exact order in which the memory access operations are interleaved does not matter.
- Example: If the three operations read (r1), write (w1), read (r2) are performed on a memory address in that order, any of the orderings (r1, w1, r2), (r1, r2, w1), (w1, r1, r2), (w1, r2, r1), (r2, r1, w1), (r2, w1, r1) **of three operations is acceptable provided all processes see the same ordering.**
- This model is weaker than that of the strict consistency model.

Sequential Consistency Model

- It provides **one-copy/single-copy** semantics because all the processes sharing a memory location always see exactly the same contents stored in it.
- Disadvantage - it does not guarantee that a read operation on a particular memory address always **returns the same value as written by the most recent write operation** to that address.

Causal Consistency Model

- A shared memory system is said to support the casual consistency model if all **write operations** that are potentially related are seen by all processors in the same (correct) order.
- **Write operations that are not potentially causally related may be seen by different processes in different orders.**
 - **Example:** if a write operation (w_2) is causally related to another write operation (w_1), the acceptable order is (w_1, w_2) because the value written by w_2 might have been influenced in some way by the value written by w_1 .
 - (w_2, w_1) is not an acceptable order.

PipelinedRAM Consistency Model

- It only ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline.
- But Write operations performed by different processes may be seen by different processes in different order.
- Example: If w_{11} and w_{12} are two write operations performed by a process p_1 in that order, and w_{21} and w_{22} are two write operations performed by a process p_2 in that order, process p_3 may see them in the order $[(w_{11}, w_{12}), (w_{21}, w_{22})]$ and another process p_4 may see them in the order $[(w_{21}, w_{22}), (w_{11}, w_{12})]$.
- This model proves a weaker consistency semantics than the consistency models described so far.

Processor Consistency Model

- It is similar to PRAM model with an additional restriction of **memory coherence**
- For any memory location all processes agree on the same order of all write operations to that location. **In effect, processor consistency ensures that all write operations performed on the same memory location are seen by all processes in the same order.**
- This means that for the earlier example, if w_{12} and w_{22} are write operations for writing to the same memory locations x , all processes must see them in the same order – w_{12} before w_{22} or w_{22} before w_{12} .

Weak Consistency Model

- It is designed to take advantage of the following characteristics common to many applications.
- It is not necessary to show the change in memory done by every write operation to other processes. The result of several write operations can be combined and send to other processes when they need it.
- A DSM system that supports the weak consistency model uses a special variable called a synchronization variable.
- When a synchronization variable is accessed by a process, (1) all changes made to the memory by the process are propagated to other nodes,(when process exits the critical section) (2) all changes made to the memory by other processes are propagated from other nodes to the process's node (when process enters a critical section)

Release Consistency Model

- To overcome the disadvantage of weak consistency model, this is designed.
- It provides a mechanism to clearly tell the system whether a process is entering a critical section or exiting from a critical section so that the system can decide and perform only either the first or the second operation when a synchronization variable is accessed by a process.
- Two synchronization variables, acquire and release, are used.

Discussion of Consistency Models

- It is difficult to grade the consistency models based on **performance** because quite different results are usually obtained for **different applications**.
- Therefore, in the design of a DSM system, the choice of consistency model usually depends on several other factors, **such as how easy is it to implement, how much concurrency does it allow**, and **how easy is it to use**.
- *Strict consistency model is never used in the design of a DSM system because its implementation is practically impossible.*
- Sequential consistency model is commonly used.
- Others are the main choices in the weaker category.
- Last two provides better concurrency.

Coherence Protocols

- Write-invalidate Protocol: invalidate all copies except the one being modified *before* the write can proceed.
 - Once invalidated, data copies cannot be used.
 - Disadvantages:
 - invalidation sent to all nodes regardless of whether the nodes will be using the data copies.
 - Inefficient if many nodes frequently refer to the data: after invalidation message, there will be many requests to copy the updated data.
 - Used in several systems including those that provide strict consistency.
- Write-update Protocol: causes all copies of shared data to be updated. More difficult to implement, guaranteeing consistency may be more difficult. (reads may happen in between write-updates).

DSM Advantages

- Parallel algorithms can be written in a transparent manner using DSM. Using message passing (e.g., send, receive), the parallel programs might become even more complex.
- Difficult to pass complex data structures with message passing primitives.
- Entire block/page of memory along with the reference data/object can be moved. This can help in easier referencing of associated data.
- DSM cheaper to build compared to tightly coupled multiprocessor systems.

DSM Advantages

- Fast processors and high speed networks can help in realizing large sized DSMs.
 - Programs using large DSMs may not need as many disk swaps as in the case of local memory usage.
 - This can offset the overhead due to communication delay in DSMs.
- Tightly coupled multiprocessor systems access main memory via a *common bus*. So the number of processors limited to a few tens. No such restriction in DSM systems.
- Programs that work on multiprocessor systems can be ported or directly work on DSM systems.