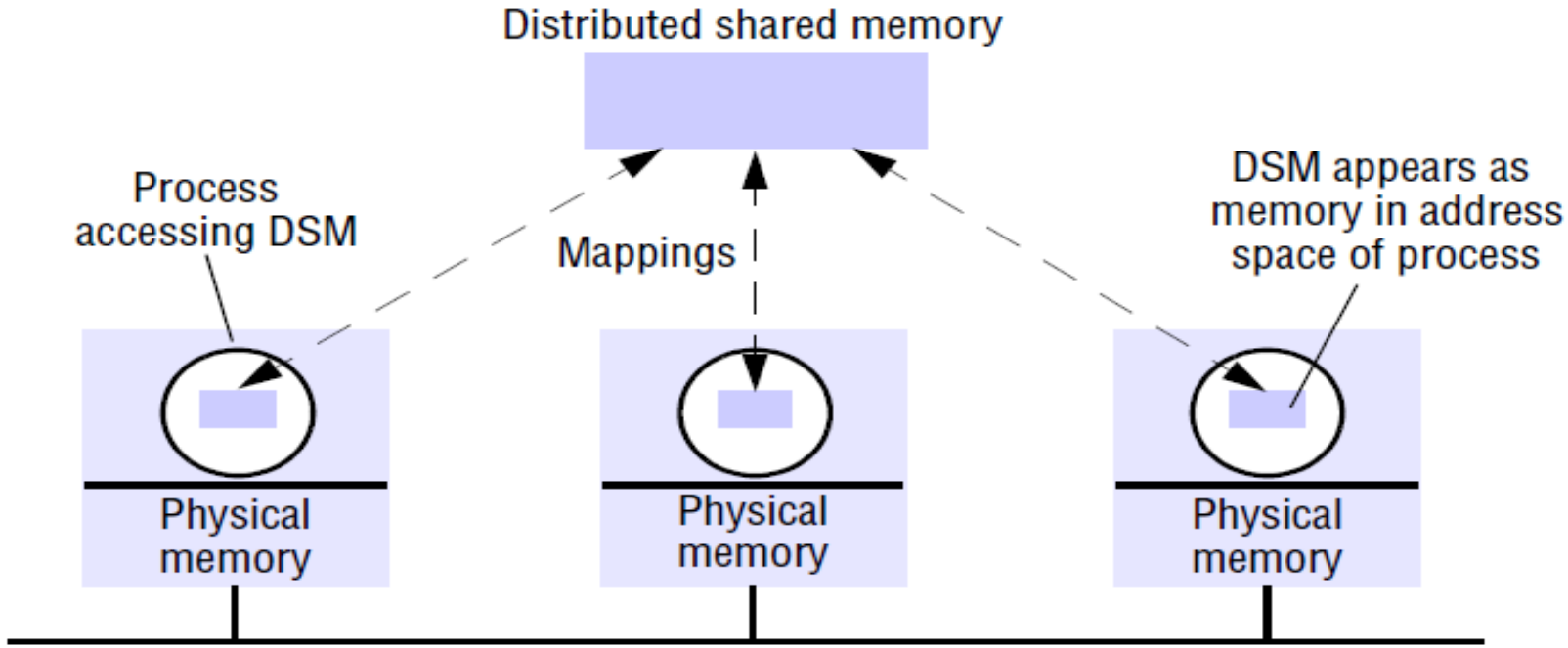
Distributed shared memory

Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory.

Processes access DSM by reads and updates to what appears to be ordinary memory within their address space. However, an underlying runtime system ensures transparently that processes executing at different computers observe the updates made by one another.

It is as though the processes access a single shared memory, but in fact the physical memory is distributed.

The distributed shared memory abstraction



The main point of DSM is that it spares the programmer the concerns of message passing when writing applications that might otherwise have to use it.

DSM is primarily a tool for parallel applications or for any distributed application or group of applications in which individual shared data items can be accessed directly.

DSM is in general less appropriate in client-server systems, where clients normally view server-held resources as abstract data and access them by request (for reasons of modularity and protection).

Message passing cannot be avoided altogether in a distributed system: in the absence of physically shared memory, the DSM runtime support has to send updates in messages between computers.

DSM systems manage replicated data: each computer has a local copy of recently accessed data items stored in DSM, for speed of access

Advantages:

1.Ease of Programming: DSM abstracts away the complexities of distributed systems, making it easier for programmers to develop parallel applications. Programmers can use familiar shared-memory programming models, like threads or processes communicating via shared memory.

2.Scalability: DSM systems can scale effectively as they allow multiple processors to access a shared memory space. Adding more nodes to the network can increase computational power without significant changes to the programming model.

3.Data Sharing: DSM allows for seamless data sharing between processes running on different nodes. This simplifies data exchange and communication in distributed applications, enhancing collaboration and efficiency.

4.Load Balancing: In DSM systems, tasks can be distributed among nodes dynamically, allowing for better load balancing. This ensures optimal resource utilization and improves system performance.

Disadvantages:

1. **Latency**: Accessing shared data in a distributed environment can incur higher latency compared to local memory access. This latency arises from the need to communicate over the network and synchronize access to shared memory.
2. **Consistency and Coherence**: Maintaining consistency and coherence of shared data across distributed nodes is challenging. Ensuring that all nodes have consistent views of the shared memory requires complex protocols, which can introduce overhead and reduce performance.
3. **Scalability Limits**: While DSM systems can scale to a certain extent, adding more nodes may eventually lead to diminishing returns or even reduced performance due to increased communication overhead and contention for shared resources.
4. **Fault Tolerance**: Distributed systems are susceptible to various types of failures, such as network partitions or node failures. Ensuring fault tolerance in DSM systems requires sophisticated mechanisms for data replication, recovery, and fault detection, adding complexity to the system.

5. Programming Complexity: While DSM abstracts away some of the complexities of distributed systems, it introduces its own set of challenges, particularly regarding synchronization, consistency, and data access patterns. Programmers need to carefully design and manage their applications to avoid performance bottlenecks and ensure correctness.

Distributed shared memory(DSM) system is a resource management component of distributed operating system that implements shared memory model in distributed system which have no physically shared memory.

The shared memory model provides a virtual address space which is shared by all nodes in a distributed system.

The central issues in implementing DSM are:

how to keep track of location of remote data.

how to overcome communication overheads and delays involved in execution of communication protocols in system for accessing remote data.

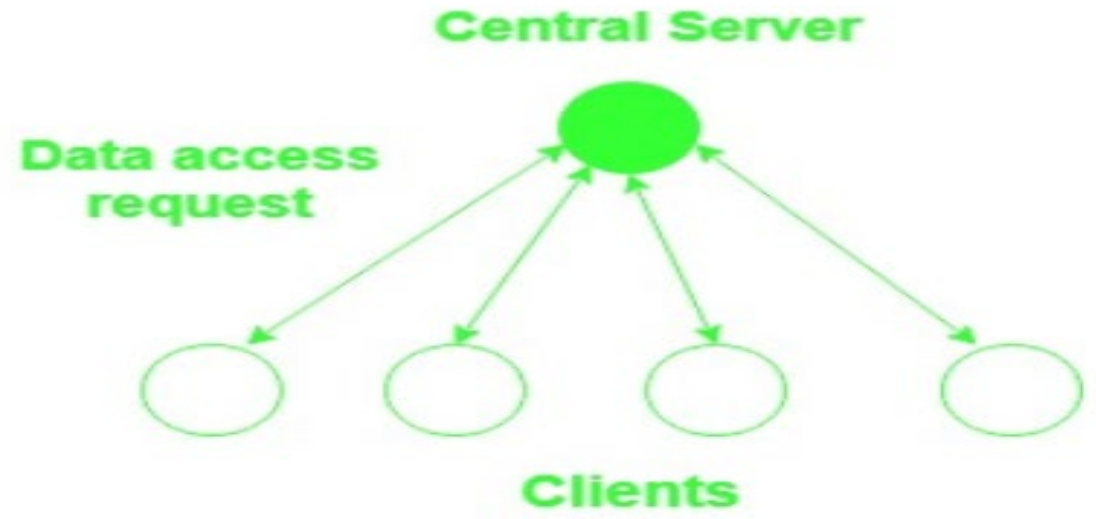
how to make shared data concurrently accessible at several nodes to improve performance.

Algorithms to implement DSM

• **Central Server Algorithm:** In this, a central server maintains all shared data. It services read requests from other nodes by returning the data items to them and write requests by updating the data and returning acknowledgement messages.

•Time-out can be used in case of failed acknowledgement while sequence number can be used to avoid duplicate write requests.

•It is simpler to implement but the central server can become bottleneck and to overcome this shared data can be distributed among several servers. This distribution can be by address or by using a mapping function to locate the appropriate server.

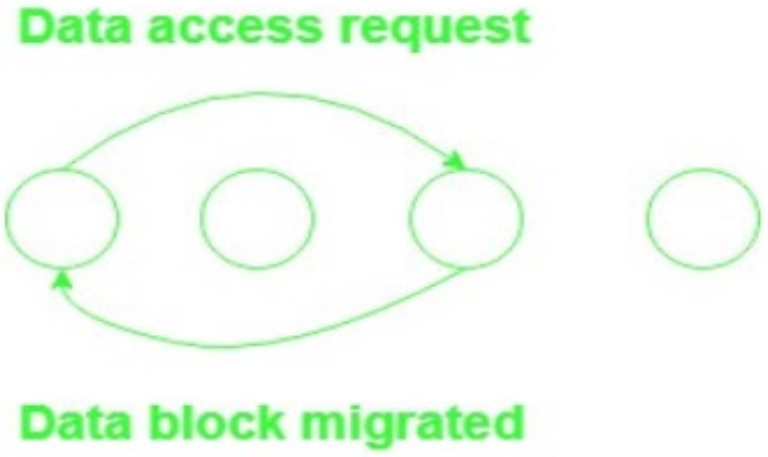


•2. **Migration Algorithm:** In contrast to central server algo where every data access request is forwarded to location of data while in this data is shipped to location of data access request which allows subsequent access to be performed locally.

•It allows only one node to access a shared data at a time and the whole block containing data item migrates instead of individual item requested.

•It is susceptible to thrashing where pages frequently migrate between nodes while servicing only a few requests.

•This algo provides an opportunity to integrate DSM with virtual memory provided by operating system at individual nodes.

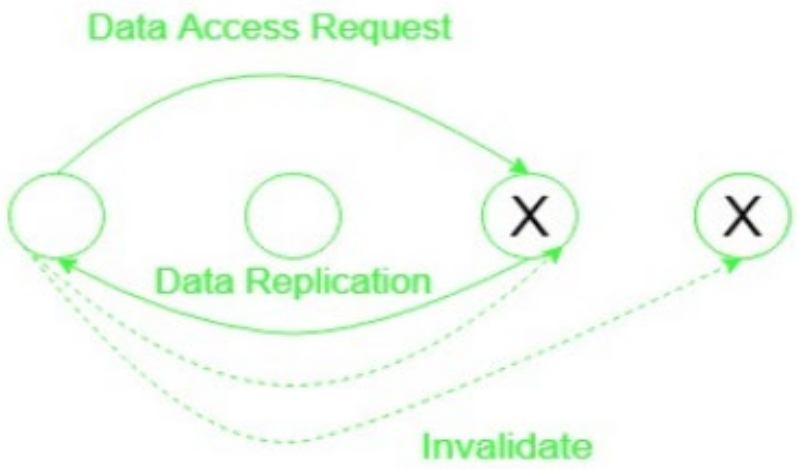


In operating systems, "thrashing" refers to a situation where the system is excessively busy swapping data between main memory (RAM) and disk storage, resulting in a significant decrease in system performance.

Insufficient Physical Memory: If the system does not have enough physical memory to accommodate the active processes' working sets, it relies heavily on virtual memory and paging mechanisms. When the demand for memory exceeds available physical memory, the system constantly swaps pages between RAM and disk, leading to thrashing.

3. Read Replication Algorithm: This extends the migration algorithm by replicating data blocks and allowing multiple nodes to have read access or one node to have both read write access.

- It improves system performance by allowing multiple nodes to access data concurrently.
- The write operation in this is expensive as all copies of a shared block at various nodes will either have to invalidated or updated with the current value to maintain consistency of shared data block.
- DSM must keep track of location of all copies of data blocks in this.

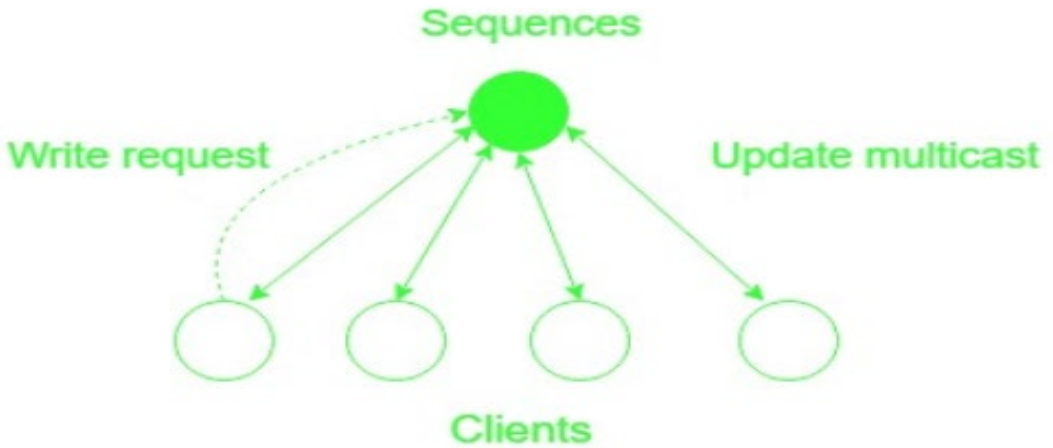


Write operation in Read Replication algorithm

4.Full Replication Algorithm: It is an extension of read replication algorithm which allows multiple nodes to have both read and write access to shared data blocks.

•Since many nodes can write shared data concurrently, the access to shared data must be controlled to maintain it's consistency.

•To maintain consistency, it can use a gap free sequences in which all nodes wishing to modify shared data will send the modification to sequencer which will then assign a sequence number and multicast the modification with sequence number to all nodes that have a copy of shared data item.



Write operation in Full Replication algorithm