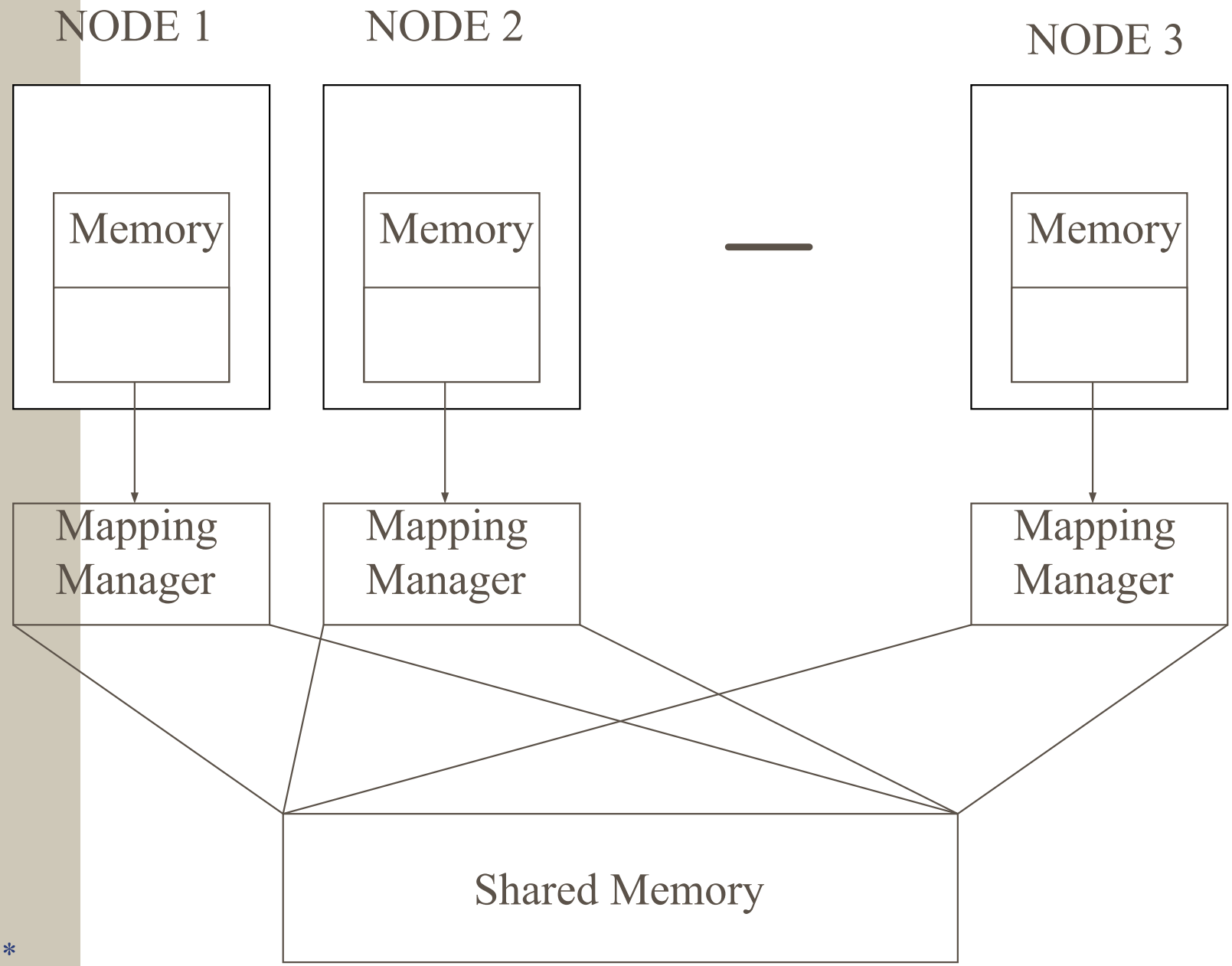
Distributed Shared Memory

- DSM provides a virtual address space that is shared among all nodes in the distributed system.
- Programs access DSM just as they do locally.
- An object/data can be *owned* by a node in DSM. Initial owner can be the creator of the object/data.
- Ownership can change when data moves to other nodes.
- A process accessing a shared object gets in touch with a *mapping manager* who maps the shared memory address to the physical memory.
- Mapping manager: a layer of software, perhaps bundled with the OS or as a runtime library routine. When a process accesses data in the shared address space, the mapping manager maps shared memory address to physical memory (local or remote)

Distributed Shared Memory...



*

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.

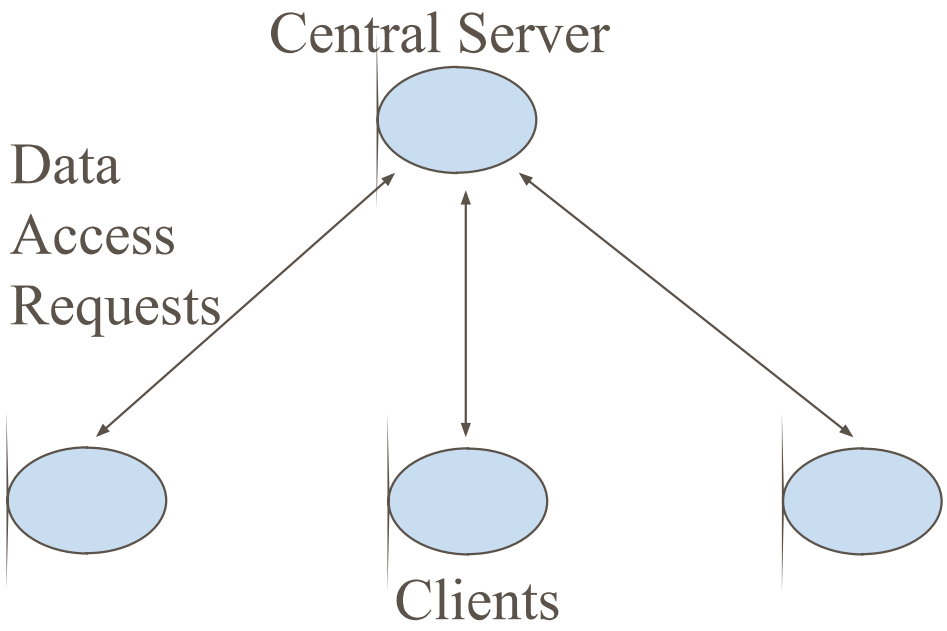
DSM Advantages…

- 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.

DSM Algorithms

- 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.
- Types of algorithms:
  - Central-server
  - Data migration
  - Read-replication
  - Full-replication

Central Server Algorithm

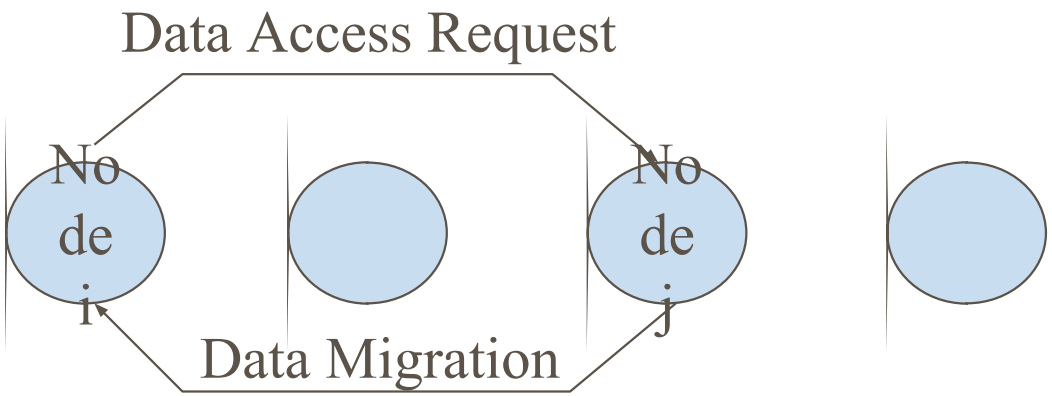


- Central server maintains all the shared data.
- It services read/write requests from clients, by returning data items.
- Timeouts and sequence numbers can be employed for retransmitting requests (which did not get responses).
- Simple to implement, but central server can become a bottleneck.

The Central Server Algorithm

- -Central server maintains all shared data
  - Read request: returns data item
  - Write request: updates data and returns acknowledgement message
- -Implementation
  - A timeout is used to resend a request if acknowledgment fails
  - Associated sequence numbers can be used to detect duplicate write requests
  - If an application's request to access shared data fails repeatedly, a failure condition is sent to the application
- -Issues: performance and reliability
- -Possible solutions
  - Partition shared data between several servers
  - Use a mapping function to distribute/locate data

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: minimum duration or minimum accesses *before* a page can be migrated (e.g., the Mirage system).

Migration Algorithm …

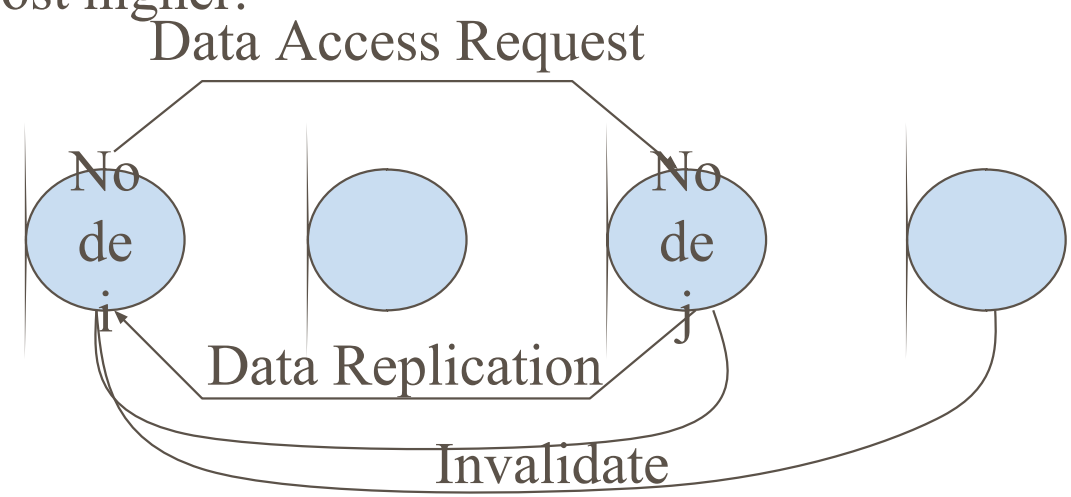
- Migration algorithm can be combined with virtual memory.
- (e.g.,) if a page fault occurs, check memory map table.
- If map table points to a remote page, migrate the page before mapping it to the requesting process's address space.
- Several processes can share a page at a node.
- Locating remote 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

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

Write Operation in Read-replication Algorithm :



DSM has to keep track of locations of all copies of data objects.
Examples of implementations:

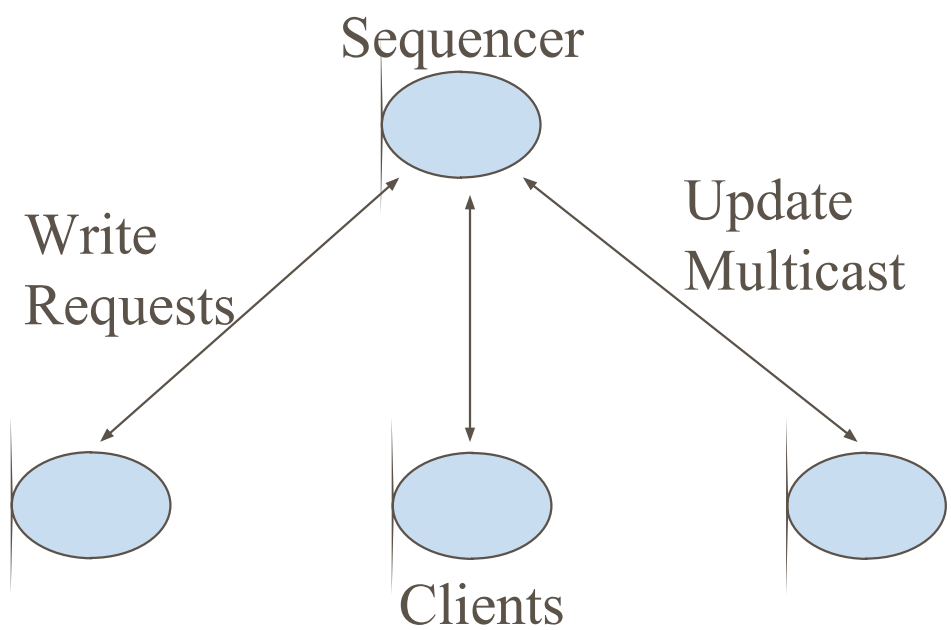
IVY: owner node of data object knows all nodes that have copies

* PLUS: distributed linked-list tracks all nodes that have copies

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.

Memory Coherence

- DSM are based on
  - Replicated shared data objects
  - Concurrent access of data objects at many nodes
- Coherent memory: when value returned by read operation is the expected value (e.g., value of most recent write)
- Mechanism that control/synchronizes accesses is needed to maintain memory coherence
- Sequential consistency: A system is sequentially consistent if
  - The result of any execution of operations of all processors is the same as if they were executed in sequential order, and
  - The operations of each processor appear in this sequence in the order specified by its program
- General consistency:
  - All copies of a memory location (replicas) eventually contain same data when all writes issued by every processor have completed

Memory Coherence…

- Processor consistency:
  - Operations issued by a processor are performed in the order they are issued
  - Operations issued by several processors may not be performed in the same order (e.g. simultaneous reads of same location by different processors may yields different results)
- Weak consistency:
  - Memory is consistent only (immediately) after a synchronization operation
  - A regular data access can be performed only after all previous synchronization accesses have completed
- Release consistency:
  - Further relaxation of weak consistency
  - Synchronization operations must be consistent which each other only within a processor
  - Synchronization operations: Acquire (i.e. lock), Release (i.e. unlock)
  - Sequence: Acquire
    Regular access
    Release

*

Coherence Protocols

- Issues
  - How do we ensure that all replicas have the same information
  - How do we ensure that nodes do not access stale data

1. Write-invalidate protocol
   - A write to shared data invalidates all copies except one before write executes
   - Invalidated copies are no longer accessible
   - Advantage: good performance for
     - Many updates between reads
     - Per node locality of reference
   - Disadvantage
     - Invalidations sent to all nodes that have copies
     - Inefficient if many nodes access same object
   - Examples: most DSM systems: IVY, Clouds, Dash, Memnet, Mermaid, and Mirage

2. Write-update protocol
   - A write to shared data causes all copies to be updated (new value sent, instead of validation)
   - More difficult to implement

*

Granularity & Replacement

- Granularity: size of the shared memory unit.
  - For better integration of DSM and local memory management: DSM page size can be multiple of the local page size.
  - Integration with local memory management provides built-in protection mechanisms to detect faults, to prevent and and recover from inappropriate references.
  - Larger page size:
    - More locality of references.
    - Less overhead for page transfers.
    - Disadvantage: more contention for page accesses.
  - Smaller page size:
    - Less contention. Reduces false sharing that occurs when 2 different data items are not shared by 2 different processors but contention occurs as they are on same page.

Granularity & Replacement…

- Page replacement needed as physical/main memory is limited.
- Data may be used in many modes: shared, private, read-only, writable,…
- Least Recently Used (LRU) replacement policy cannot be directly used in DSMs supporting data movement. Modified policies more effective:
  - Private pages may be removed ahead of shared ones as shared pages have to be moved across the network
  - Read-only pages can be deleted as owners will have a copy
- A page to be replaced should not be lost for ever.
  - Swap it onto local disk.
  - Send it to the owner.
  - Use *reserved memory* in each node for swapping.

Case Studies

- Cache coherence protocol:
  - PLUS System
  - Munin System
- General Distributed Shared Memory:
  - IVY (Integrated Shared Virtual Memory at Yale)

PLUS System

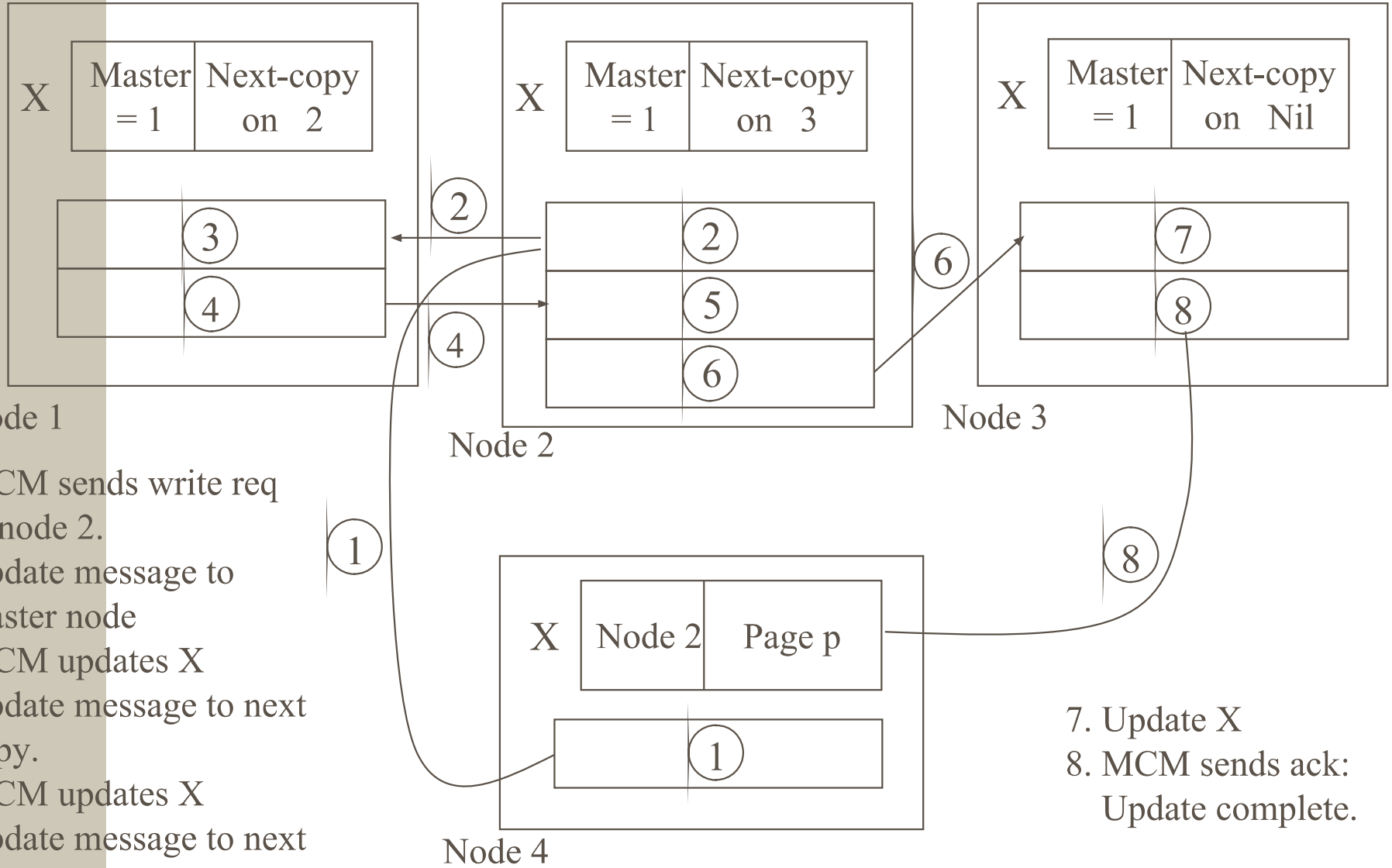
- PLUS system: write-update protocol and supports general consistency.
- Memory Coherence Manager (MCM) manages cache.
- Unit of replication: a page (4 Kbytes); unit of memory access and coherence maintenance: one 32-bit word.
- A virtual page corresponds to a list of replicas of a page.
- One of the replica is designated as *master copy*.
- Distributed link list (*copy-list*) identifies the replicas of a page. Copy-list has 2 pointers
  - Master pointer
  - Next-copy pointer

*

PLUS: RW Operations

- Read fault?:
  - If address points to local memory, read it. Otherwise, local MCM sends a read request to its counterpart at the specified remote node.
  - Data returned by remote MCM passed back to the requesting processor.
- Write operation:
  - Always on master copy then propagated to copies linked by the copy-list.
  - Write fault: memory location for write, not local.
  - On write fault: update request sent to the remote node pointed to by MCM.
  - If the remote node does not have the master copy, update request sent to the node with master copy and for further propagation.

PLUS Write-update Protocol



1. MCM sends write req to node 2.
2. Update message to master node
3. MCM updates X
4. Update message to next copy.
5. MCM updates X
6. Update message to next copy
7. Update X
8. MCM sends ack: Update complete.

*

PLUS: Protocol

- Node issuing write is not blocked on write operation.
- However, a read on that location (being written into) gets blocked till the whole update is completed. (i.e., remember pending writes).
- Strong ordering within a single processor independent of replication (in the absence of concurrent writes by other processors), but not with respect to another processor.
- *write-fence* operation: strong ordering with synchronization among processors. MCM waits for previous writes to complete.

Munin System

- Use application-specific semantic information to classify shared objects. Use class-specific handlers.
- Shared object classes:
  - *Write-once objects*: written at the start, read many times after that. Replicated on-demand, accessed locally at each site. Large object? Portions can be replicated instead of whole object.
  - *Private objects*: accessed by a single thread. Not managed by coherence manager unless accessed by a remote thread.
  - *Write-many objects*: modified by multiple threads between synchronization points. Munin employs delayed updates. Updates are propagated only when thread synchronizes. Weak consistency.
  - *Result objects*: Assumption is concurrent updates to different parts of a result object will not conflict and object is not read until all parts are updated -> delayed update can be efficient.
  - *Synchronization objects*: (e.g.,) distributed locks for giving exclusive access to data objects.

Munin System…

- *Migratory objects*: accessed in phases where each phase is a series of accesses by a single thread: lock + movement, i.,e., migrate to the node requesting lock.
- *Producer-consumer objects*: written by 1 thread, read by another. Strategy: move the object to the reading thread in advance.
- *Read-mostly object*: i.e., writes are infrequent. Use broadcasts to update cached objects.
- *General read-write objects*: does not fall into any of the above categories: Use Berkeley ownership protocol supporting strict consistency. Objects can be in states such as:
  - *Invalid*: no useful data.
  - *Unowned*: has valid data. Other nodes have copies of the object and the object cannot be updated without first acquiring ownership.
  - *Owned exclusively*: Can be updated locally. Can be replicated on-demand.
  - Owned non-exclusively: Cannot be updated before invalidating other copies.

*

IVY

- IVY: Integrated Shared Virtual Memory at Yale. On Apollo DOMAIN environment on a token ring network.
- Granularity of access page: 1KByte.
- Address space of a processor: shared virtual memory + private space.
- On a page fault: if the page is not local, it is acquired through remote memory request and made available to other processes in the node as well.
- Coherence protocol: multiple readers-single writer semantics Using writer-invalidation protocol, all read-only copies are invalidated before writing.

Write-invalidation Protocol

- Processor *i* has a write-fault on page *p*:
  - *i* finds owner of *p*
  - *p*'s owner sends the page + copyset (i.e., the processors having read-only copies). Owner marks its page table entry of *p* as nil.
  - *i* sends invalidation messages to all processors in the copyset.
- Processor *i* has a read fault to a page *p*:
  - *i* finds the owner of *p*.
  - *p*'s owner sends *p* to *i* and adds *i* to the copyset of *p*.
- Implementation schemes:
  - Centralized manager scheme
  - Fixed distributed manager scheme
  - Dynamic distributed manager scheme
- Above schemes differ on how the owner of a page is identified

*

Centralized Manager

- Central manager on a node maintains all page ownership information.
- Page faulting processor contacts the central manager and requests a page copy.
- Central manager forwards the request to the owner. For write operations, central manager updates the page ownership information to the requestor.
- Page owner sends a page copy to the faulting processor.
  - For reads, the faulting processor is added to copyset.
  - For writes, owner is faulting processor now.
- Central manager requires 2 messages to locate the page owner.

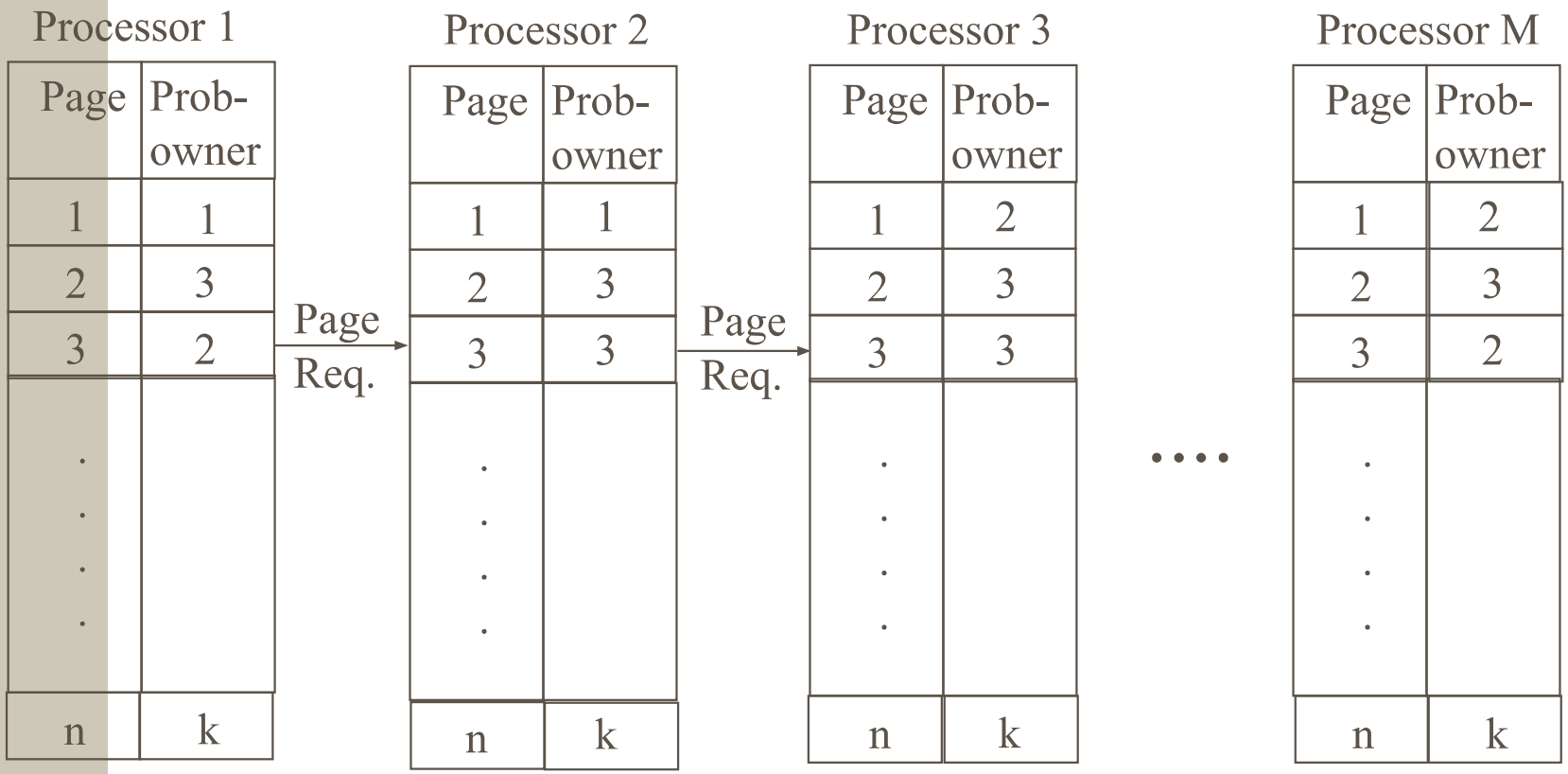
Fixed Distributed Manager

- Every processor keeps track of a pre-determined set of pages (determined by a hashing/mapping function H).
- Processor *i* faults on *p*: *i* contacts processor H(*p*) for a copy of the page.
- Rest of the steps proceeds as in centralized manager.
- In both the above schemes, concurrent access requests to a page are serialized at the site of a manager.

Dynamic Distributed Manager

- Every host keeps track of the page ownership in its local page table.
- Page table has a column *probowner* (probable owner) whose value can either be the true owner or a probable one (i.e., it is used as a hint). Initially set to a default value.
- On a page fault, request is sent to *i* (assume). If *i* is the owner, steps proceed as in the case of central manager.
- If not, request is forwarded to *probowner* of *p* at *i*. This is done until the actual owner is reached.
- *probowner* is updated on receipt of:
  - Invalidation requests, ownership relinquishing messages.
  - Receiving or forwarding of a page:
    - For writes, the receiver is the owner. Forwarder updates the owner as the receiver.

Dynamic Distributed Manager…



*

Dynamic Distributed Manager …

- At most (N-1) messages needed to locate the owner.
- As hints are updated as a side effect, the average number of messages should be lower.
- Double fault:
  - Consider a processor *p* doing a read first and then a write.
  - *p* needs to get the page twice.
- One solution:
  - Use sequence numbers along with page transfers.
  - *p* can send the its page sequence number to the owner. Owner can compare the numbers and decide whether a transfer is needed or not.
  - Still checking with owner is needed, only transfer of whole page is avoided.

Memory Allocation

- Centralized scheme for memory allocation. A central manager allocates and deallocates memory.
- A 2-level procedure may be more efficient:
  - Central manager allocates a large chunk to local processors.
  - Local manager handles local allocations.

Process Synchronization

- Needed to serialize concurrent accesses to a page.
- IVY uses *eventcounts* and provides 4 primitives:
  - Init(ec): initialize an event count.
  - Read(ec): returns the value of an event count.
  - Await(ec, value): calling process waits until *ec* is not equal to the specified value.
  - Advance(ec): increments *ec* by one and wakes up waiting processes.
- Primitives implemented on shared virtual memory:
  - Any process can use eventcount (after initialization) without knowing its location.
  - When the page with eventcount is received by a processor, eventcount operations are local to that processor and any number of processes can use it.

Process Synchronization...

- Eventcount operations are atomic.
  - using test-and-set instructions
  - disallowing transfer of memory pages with eventcounts when primitives are being executed.