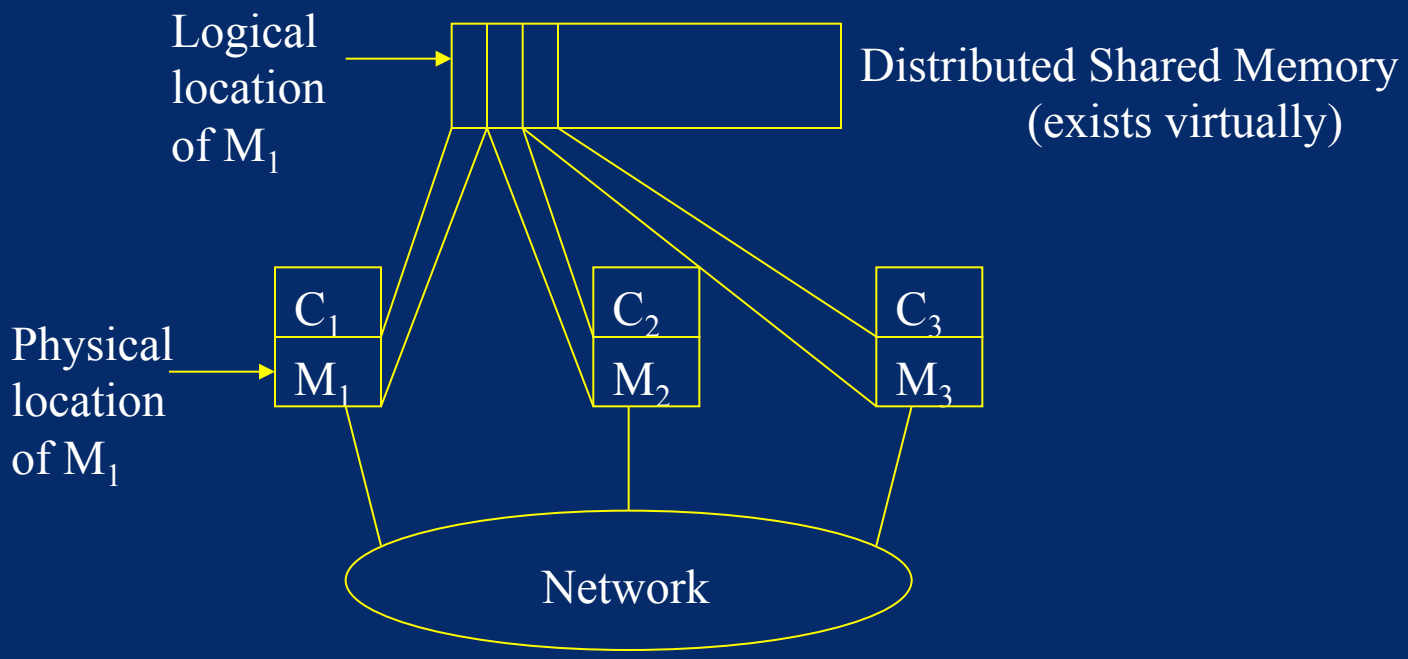
Distributed Shared Memory (DSM)

- Introduction
- Shared Memory Systems
- Distributed Shared Memory Systems
- Advantage of DSM Systems

Distributed Shared Memory Systems



Distributed Shared Memory (DSM)

Main Issues (cont)

- Granularity and structure
  - granularity refers to the size of sharing unit that can be uniform chunks of memory or data structures: byte, page or complex data structure
  - structure refers to the arrangement of shared data
    - most systems view DSM as a linear array of words
  - small pages: increased parallelism -> increase in directory size
  - large pages: reduce paging overhead, but increase sharing overhead

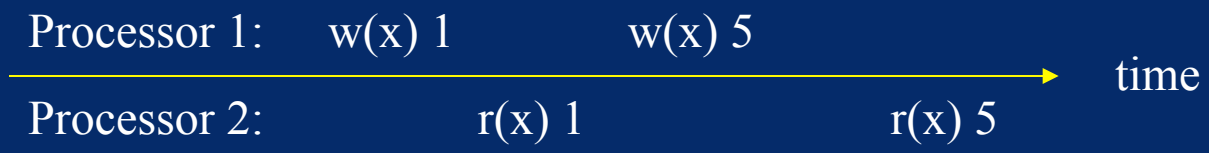
Main Issues (cont)

- Replacement Strategies
  - Similar to caching mechanisms in MP
  - In cache systems, LRU is often used
  - In DSM, shared pages need to be given higher priority than exclusively owned pages => they could be replaced first
- Synchronization Primitives:
  - Coherence protocols must ensure the consistency of shared data
  - DSM must allow simultaneous access to shared data on different machines. (single writer, multiple readers, etc.)

2) *Memory Coherence, Access Synchronization*

- *Strict Consistency Model*

  any read to a certain memory location returns the value stored by most recent write operation to that address, irrespective of the locations of the processors performing the read and the write operation.



Behavior of strict consistence model

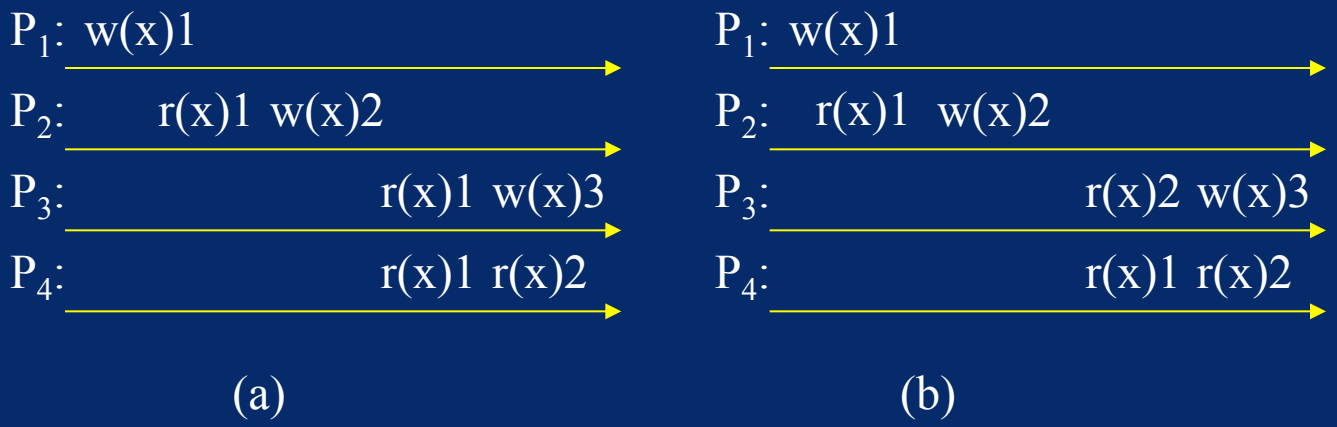
- *Sequential Consistency Model*

  if the result of any execution is the same as if the operations of all processors were executed in the same sequential order, and the operations of each individual processor appear in this sequence in the order specified by the program.

- *Causal Consistency Model*

  writes that are potentially causally related must be seen by all processors in the same order, writes that are not potentially causally related may be seen in a different order on different machines.

- *Causal Consistency Model (cont.)*



(a) A violation of causal memory.
(b) A correct sequence of events in causal memory.

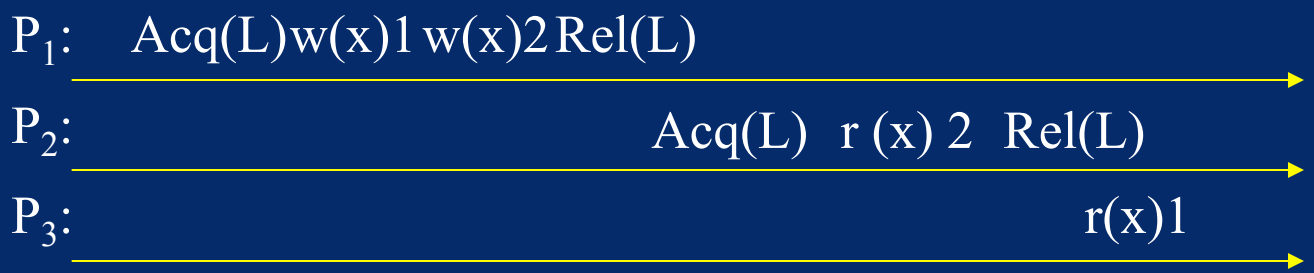
- *Processor Consistency Model*

  writes done by a single processor are seen by all other processors in the order in which they were written on that processor, but writes from different processors may be seen in a different order by different processors.

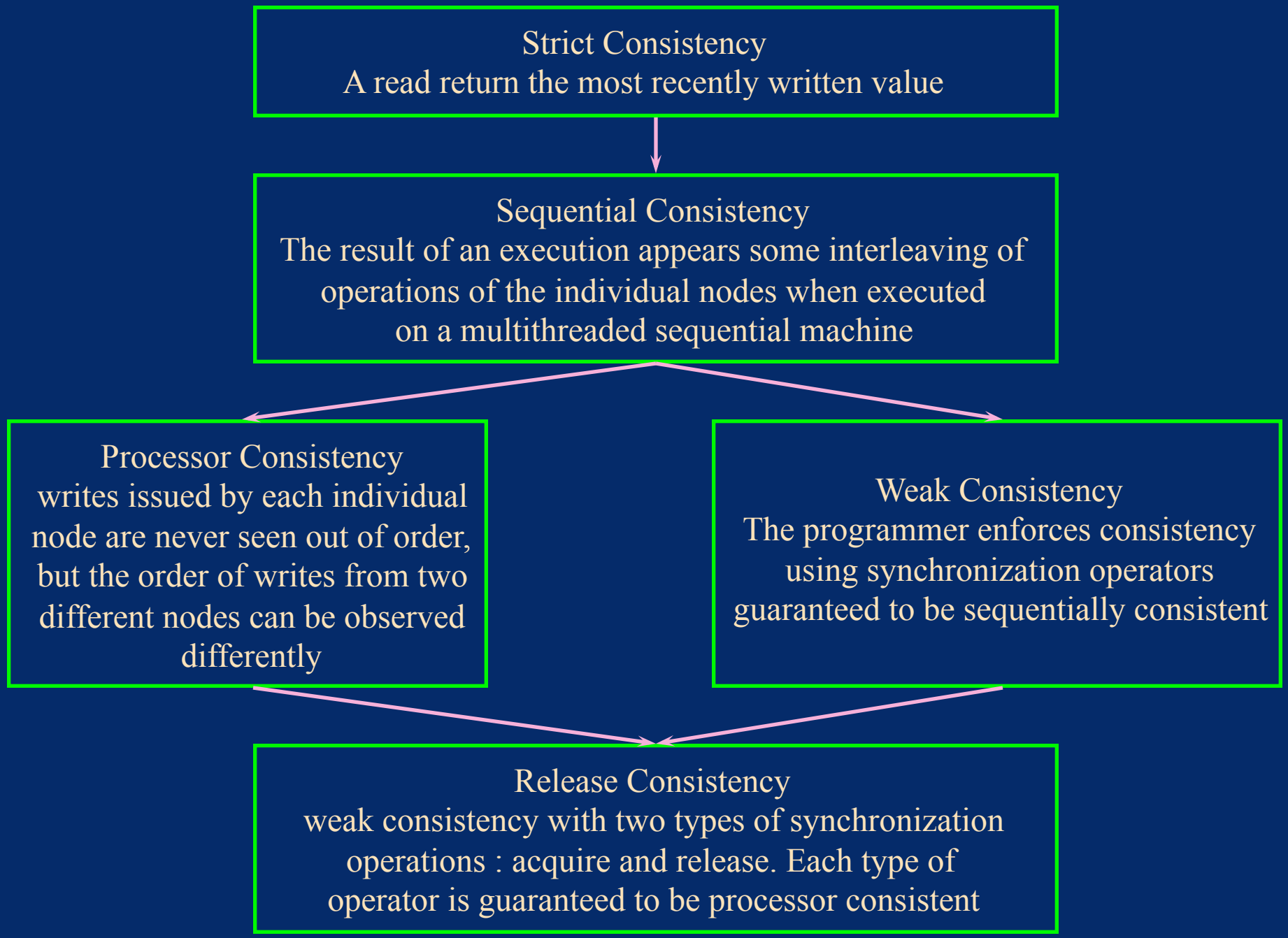
  if w_{11} and w_{12} are two writes performed by processor 1 in that order, and w_{21} and w_{22} are performed by processor 2 in that order.

- A processor consistency model guarantees that all procesors see the write in the order on which written on that processor, i.e., $[(w_{11}, w_{12}), (w_{21}, w_{22})]$

- *Release Consistency Model*



A valid event sequence for release consistency.


Strict Consistency
A read return the most recently written value
Sequential Consistency
The result of an execution appears some interleaving of operations of the individual nodes when executed on a multithreaded sequential machine
Processor Consistency
writes issued by each individual node are never seen out of order, but the order of writes from two different nodes can be observed differently
Weak Consistency
The programmer enforces consistency using synchronization operators guaranteed to be sequentially consistent
Release Consistency
weak consistency with two types of synchronization operations : acquire and release. Each type of operator is guaranteed to be processor consistent

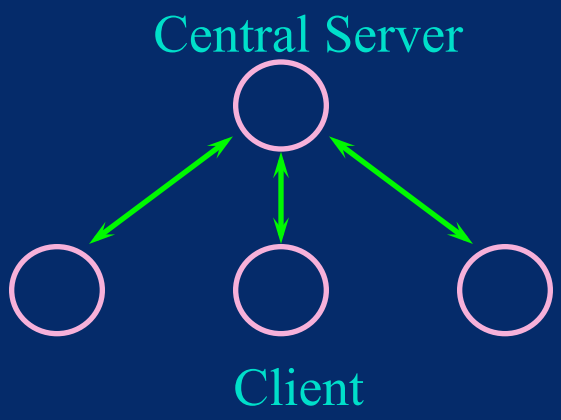
Advantages of Distributed Shared Memory

- a simpler abstraction that is well understood by programmer
- the shared memory system hides the remote communication mechanism and allows complex structures to be passed by reference
- in message passing model, the programmer must be aware of
  - data movement between all processes
  - **it is difficult to pass complex data structures**
  - **in general, distributed shared memory application runs slower than message passing based applications**

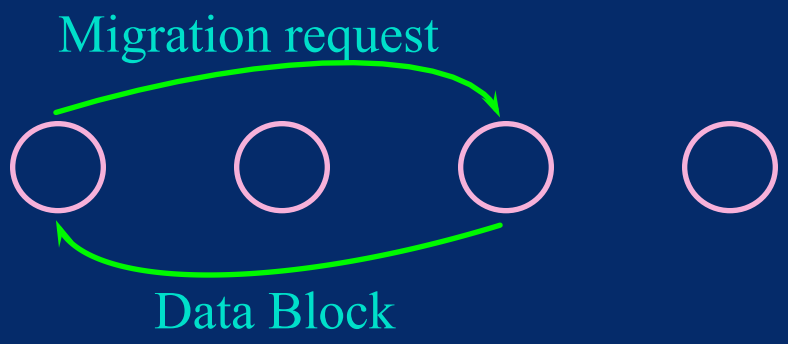
Similar Systems

- CPU Cache memories in shared memory multiprocessors
- local memories in shared memory multiprocessors with nonuniform memory access (NUMA) times
- distributed caching in network file systems
- distributed databases
- all these system attempt to minimize the access time to potentially shared data that needs to be kept consistent
- for performance analysis, communication costs are abstracted in terms of number of messages sent and the number of packet events

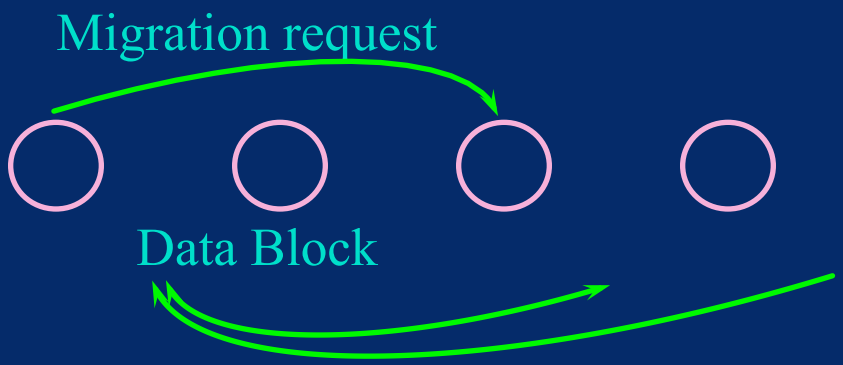
Distributed Shared Memory Algorithms (cont)



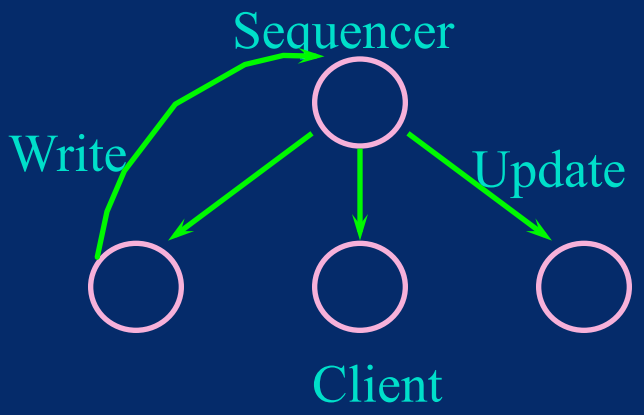
Central Server



Migration



Read replication



Full replication

Central-Server Algorithm

- this algorithm requires two messages for each data access
  - one from the process requesting the access
  - the second contains the data server's response
  - each data access requires four packet events
  - central server might become the bottleneck
    - load can be reduced by distributing shared data over multiple servers
    - a simpler method is to partition data based on server address

Migration Algorithm

- the data is always migrated to the site where it is accessed (SRSW protocol)
  - to reduce costs, blocks are migrated
  - if access behavior does not follow locality of reference property, thrashing can occur between hosts
- it can be integrated with local virtual memory system if the block size chosen equal to that of the local virtual memory
  - access to a remote page triggers a page fault so that page fault handler can bring requested pages from other hosts
    - to improve performance, one can assign managers to locate certain data blocks

Read-Replication Algorithm

- the problem with the previous techniques is the sequential access to the data block
- replication can reduce the average cost of read operations
  - write operations might be more expensive since replicas may have to be invalidated or updated to maintain consistency
  - it is okay if the ratio of read to write is high
- replication can be added to migration algorithm which results (MRSW)
  - for a read operation to a remote block, node needs first to acquire read-only copy of the requested block
  - for a write operation to a block that is not local or node does not have write access to, all replica blocks must be invalidated before write can proceed

Coherent Protocols

- Write-Invalidate Protocol:
  - a write to a shared data causes the invalidation of all copies except one before the write can proceed.
  - once invalidated, copies are no longer accessible
  - disadvantage: irrespective of whether all other nodes will use this data or not
- Write-Update Protocol:
  - a write to a shared data causes all copies to that data to be updated.
  - more difficult to implement because a new value has to be sent instead of invalidation.

Full-Replication Algorithm

- multiple readers/multiple writers (MRMW) protocol
- access to data must properly sequenced or controlled to ensure consistency
  - needs to globally sequence the write operations
  - intended modifications are sent to the sequencer that assigns the next sequence number and multicasts the modification with this sequence number
    - each site processes broadcast write operations in sequence number order

Performance measure

- it needs to take into account the cost of accessing local and remote data blocks
- comparative analysis
  - we do pair-wise comparisons to illustrate the conditions under which one algorithm might outperform another
  - we equate their cost to derive a curve along which they yield similar performance

Performance Analysis

- the parameters that characterize the costs of shared data access are
  - p: cost of sending or receiving a short packet
  - P: cost of sending or receiving a data block
    - assume P/p equal to 20
  - S: number of sites participating in distributed shared memory
  - r: Read/Write ratio
  - f: probability of an access fault on a nonreplicated data block
  - f `: probability of an access fault on replicated data blocks

Performance Analysis-Cont.

- Simplified Assumptions:
  - the message traffic will not cause network congestion so we can ignore network bandwidth occupied by messages
  - server congestion is not a serious to significantly delay remote access
  - the cost of accessing local data is negligible when compared to that associated with remote data access
  - message passing is reliable so the cost of retransmission can be ignored

Performance Measures

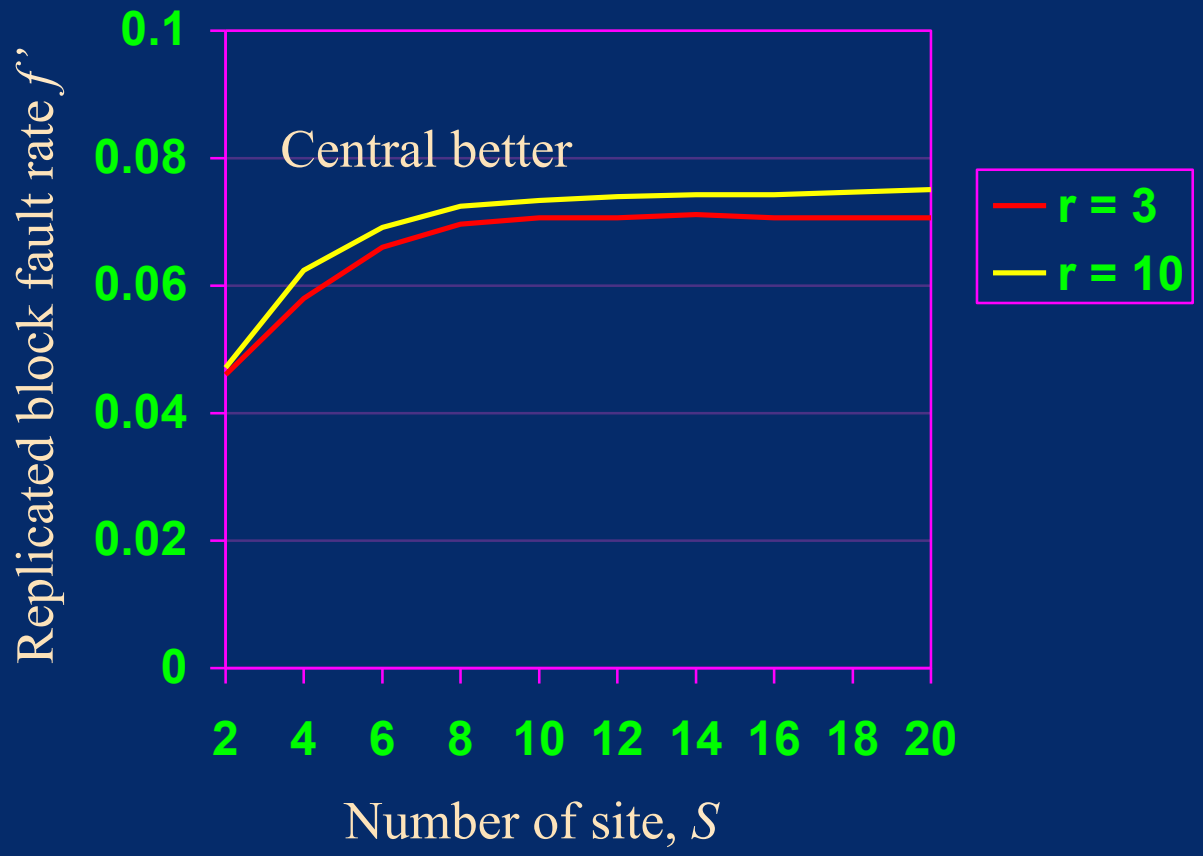
$$C_c = \left(1 - \frac{1}{S}\right) \times 4p$$

$$C_m = f \times (2P + 4p)$$

$$C_{rr} = f' \times \left[2P + 4p + \frac{Sp}{r+1}\right]$$

$$C_{fr} = \frac{1}{r+1} \times (S+2)p$$

Central Server v.s. Read Replication



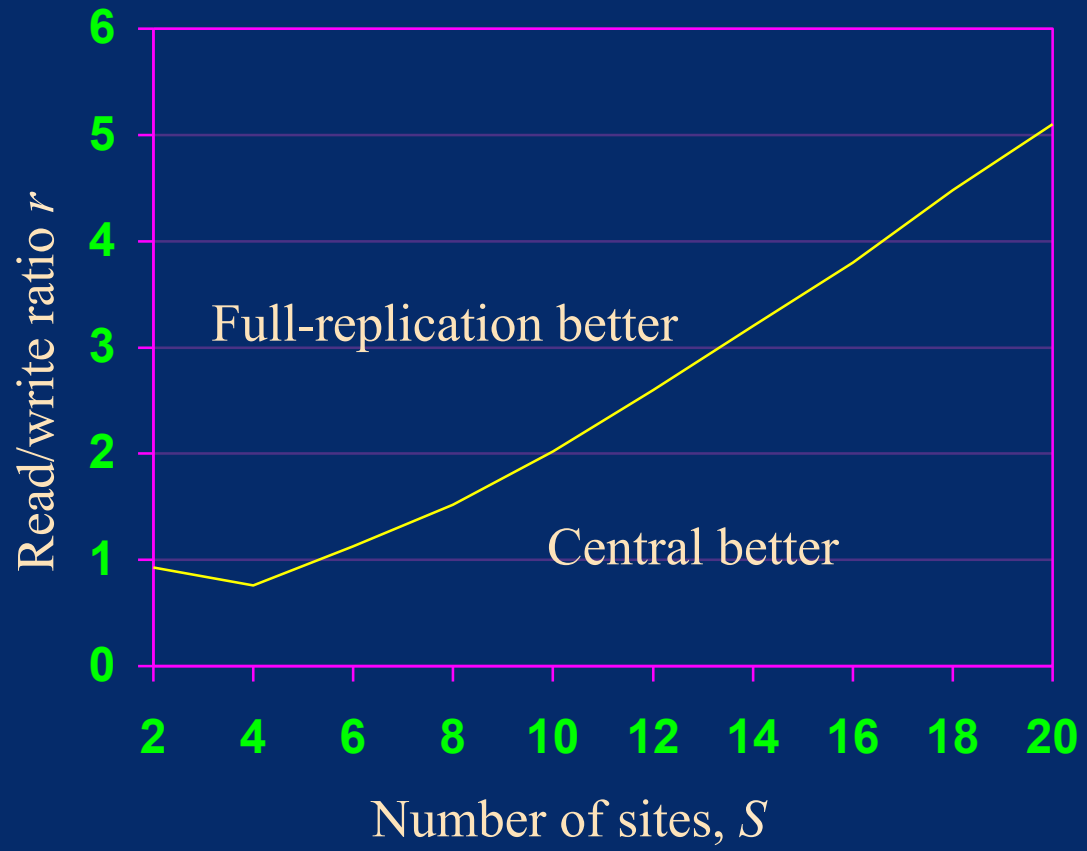
$$f' = \frac{4\left(1 - \frac{1}{S}\right)}{44 + \frac{S}{r+1}}$$

Central Server v.s. Read Replication- Cont.

- for small number of sites and high read/write ratio, read replication performs better
- If the number of sites increases, the update cost increases and that makes the central server better.

- Comments:
  - no single algorithm is good for all applications
  - algorithms need to be adaptive to application characteristics

Central Server v.s. Full Replication

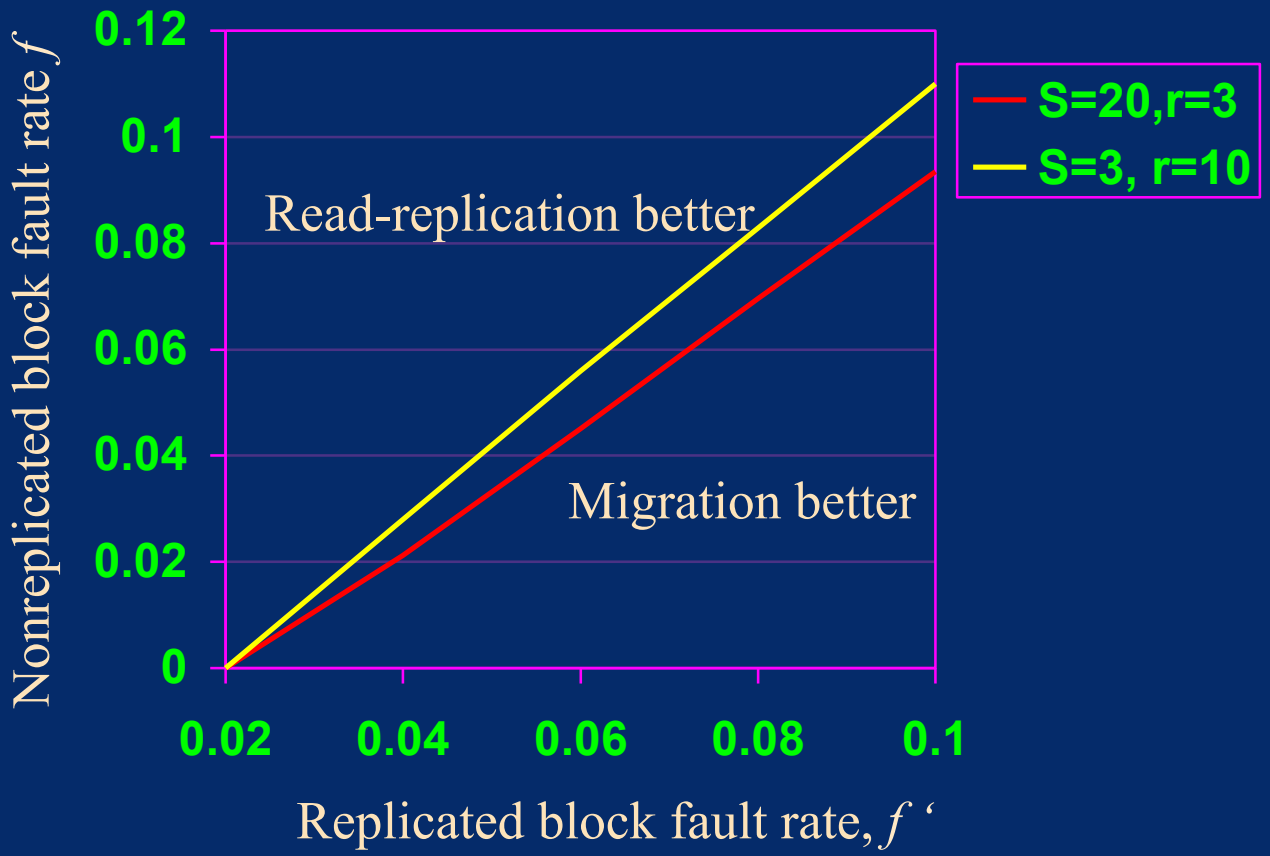


$$r = \frac{1}{4}\left[(S-1) + \frac{3}{S-1}\right]$$

Central Server versus Full Replication

- these represent the two extremes: one is completely centralized, the other is completely distributed and replicated
- for values of S up to about 20, full replication is better as long as r is 5 or higher

Migration v.s. Read Replication



$$f = f'\left[1 + \frac{S}{44(r+1)}\right]$$

Migration v.s. Read Replication-Cont.

- read replication reduces block fault rate
- read replication can outperform migration for a vast majority of applications

Read Replication v.s. Full Replication

$$f' = \frac{S+2}{S+44(r+1)}$$

Read Replication v.s. Full Replication

- Their performance depends on:
  - degree of replication
  - read/write ratio
  - degree of locality in read applications
- Generally speaking, full replication performs poorly for large systems and high update frequency (low r)

Distributed Shared Memory Algorithms

- Central server: one server responsible for serving all accesses to shared data and maintains the only copy of the shared data.
- Migration (Single reader / single writer (SRSW)): Data is migrated to the site where it is accessed.
- Read Replication: replication is done by allowing either
  - one site read /write or
  - multiple sites of read copies of a block
- Full Replication:
  - allows data blocks to be replicated while being written to (MWMR) protocol
  - use a single gap-free sequencer for write operation

Observations

- central server is simple to implement
  - it is sufficient for infrequent access to shared data especially if R/W is low
- locality of reference and high block hit ratio is usually high
  - block migration and replication becomes advantageous
- read replication seems a good compromise and work fine in most applications

DSM Classification

- Implementation Level
  - Hardware or software or hybrid
- Architecture Configuration
  - describes the system on which the DSM is running
- Shared Data Organization
  - Structured/non-structured data , objects, language data type
- Granularity of Coherence unit
  - Word, cache block, page, or data structure (object)
- DSM Algorithm (SRSW, MRSW, and MRMW
- Management Responsibility
  - Centralized or distributed
- Consistency Model
- Coherence Control Protocol
  - Write invalidate or write-update coherence protocol

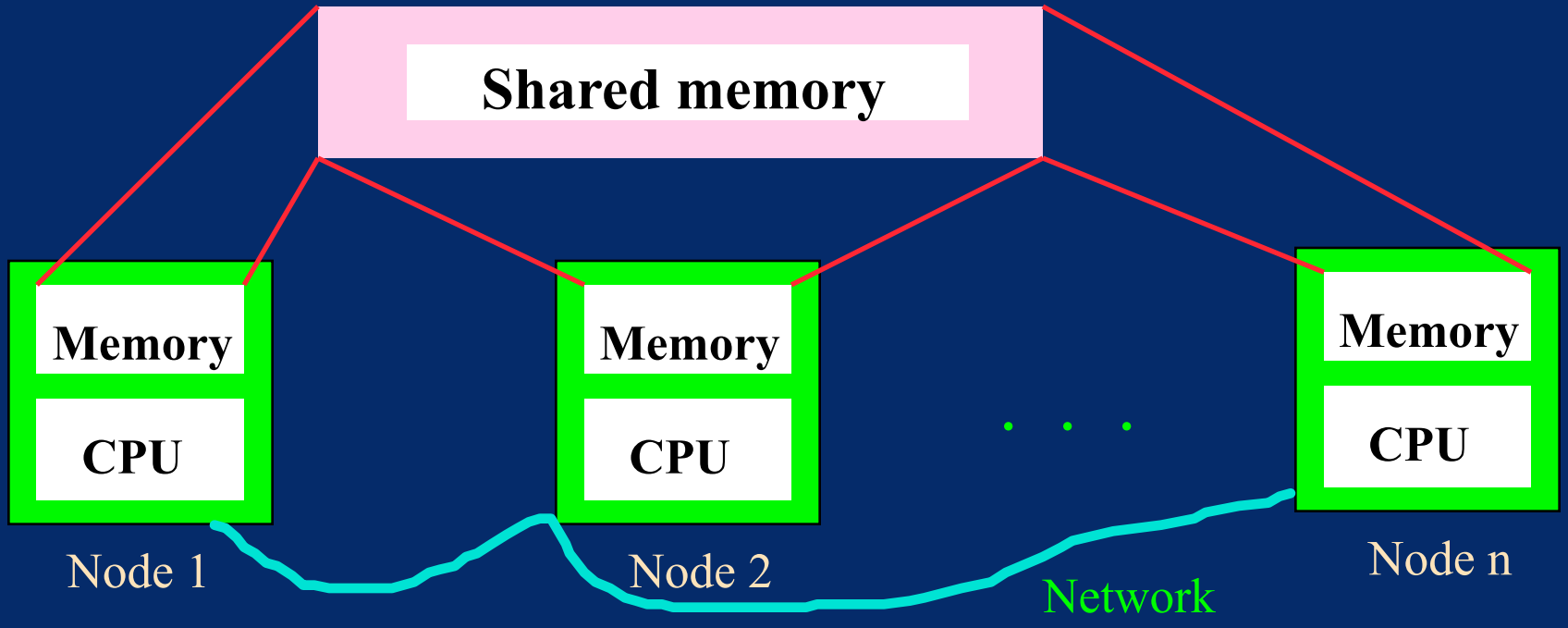
Distributed Shared Memory (DSM)

- Introduction
- Shared Memory Systems
- Distributed Shared Memory Systems
- Advantage of DSM Systems

Distributed Shared Memory (DSM) Systems

References:

- Nitzberg Bill and Virginia Lo. "*Distributed Shared Memory: A survey of issues and algorithms.*" IEEE Computer August 1991
- Stumm, Michael and Songnian Zhou. *"Algorithms Implementing Distributed Shared Memory."* IEEE Computer May 1990



Distributed Shared Memory Systems

- Page-based Distributed Shared Memory Systems, such as IVY, CVM
- Shared Variable Distributed Shared Memory Systems, such as Munin
- Object-based Distributed Shared Memory Systems, such as Linda, Orca

Case study: IVY

- IVY system (Integrated shared Virtual Memory at Yale)
  - first DSM implementations with strict consistence and invalidate protocol(MRSW)
  - algorithms are based on centralized and distributed techniques for solving coherence problem
  - a prototype is implemented on Appolo ring

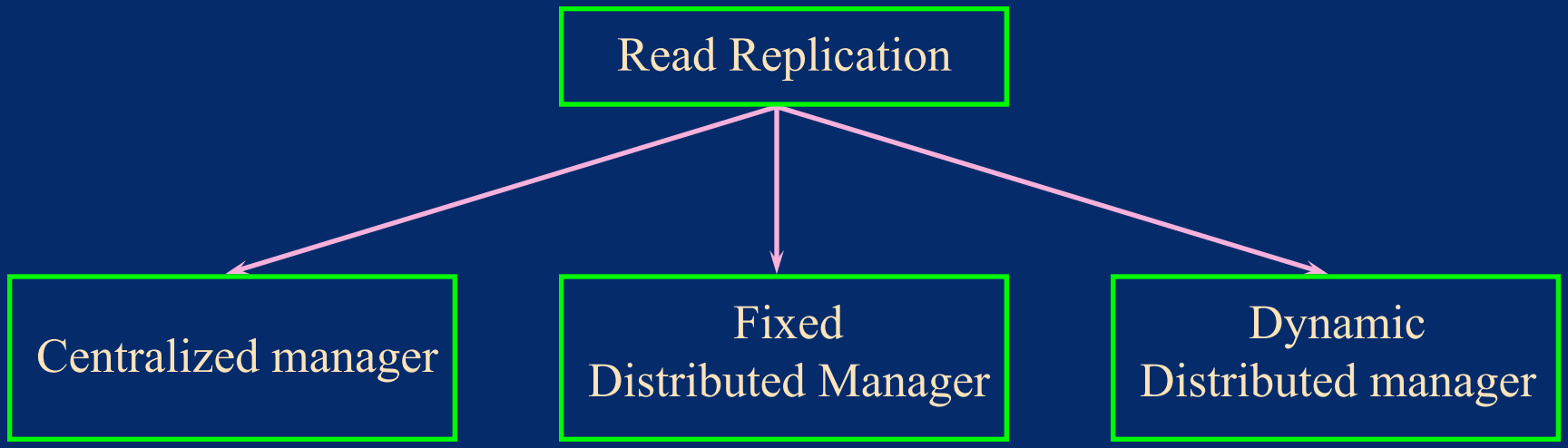
Case Study:IVY-Cont.

- Development algorithms:
  1. Page synchronization:
     - write-invalidate is chosen in IVY
     - write-update: it is not feasible because of required HW support and high network latency.
  2. Page ownership
     - fixed: suitable for algorithms that do not migrate data
     - dynamic: it is chosen in IVY system

=> Class belong to Migration and Replication.

Case study: IVY (cont)

- Read Replication Algorithms



Case study: IVY (cont)

Page contains information:

- access: page accessibility
- copy set: processor number who has read copy
- lock: synchronize multiple page faults

1. Centralized Manager:
   - maintains a table INFO which has entry for each page: entry fields:
     (1) owner
     (2) copy set: list of all processors with Read copies
     (3) lock
   - other processors have two fields: access and lock
   - on Read fault: the manager is contacted to get copy
   - on Write fault: similar way, but the manager invalidates the owner copy.

Case study: IVY (cont)

2. Fixed distributed Managers:
   - every processor is given a number of pages to manage.
3. Dynamic Distributed Managers:
   - keep track of ownership of all pages in each processor's local page table.

- IVY: Page based DSM system (cont.)
  - emulate the cache of a multiprocessor.
  - run multiprocessor program without modification.
  - implemented with sequential consistency model and invalidation protocol (MRSW).
  - the basic unit passing through the network is page.
  - access to remote data is detected by MMU.
  - fixed or dynamic page manager.
  - maintain a copy set.
  - use lock to synchronize multiple page fault.
  - the only problem is the performance.

Case study: PLUS

- The PLUS system employs the write-update protocol
- A memory coherent manager (MCM) per node.
- MCM is responsible for maintaining the consistency.
- A virtual page in the PLUS system corresponds to a list of replicas of a page.
- One replica is designated to be the Master copy
- The MCM is made aware of the other replicas through a distributed linked-list called copy-list

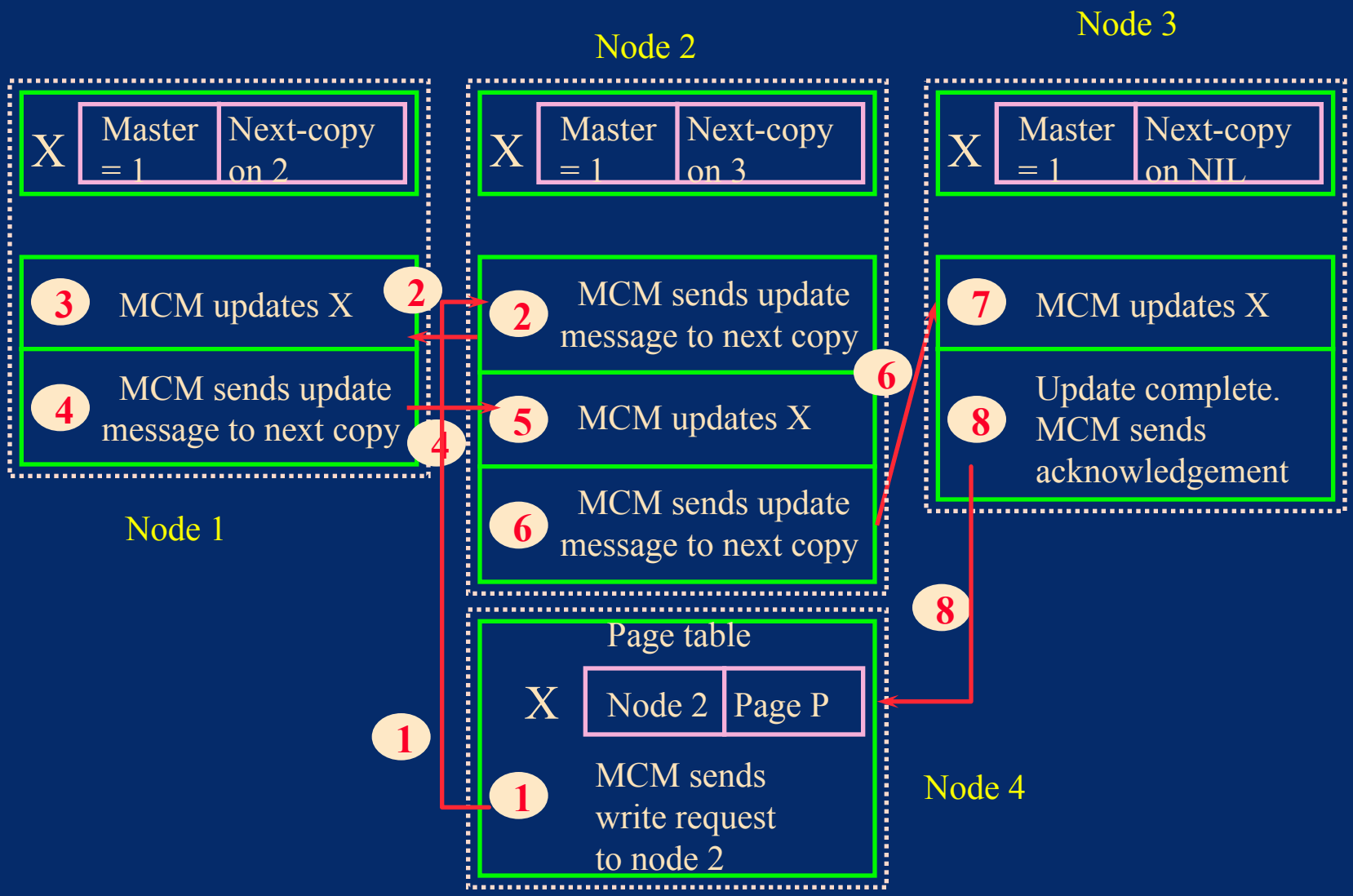
Case Study:PLUS-Cont.

- Read operation:
  - if the address indicates local memory, the local memory is read
  - otherwise the local MCM sends a read request to its counterpart MCMs
  - The data is returned by the remote MCM

Case study: PLUS (cont)

- Write Operation:
  - The writes are always performed first on the master copy and are then propagated to the copies linked by the copy-list.
  - on a write fault, generated by local copy (not the master copy), then the update request is sent to the node containing the master copy
  - once done, further update propagation is performed.

PLUS Write-Update

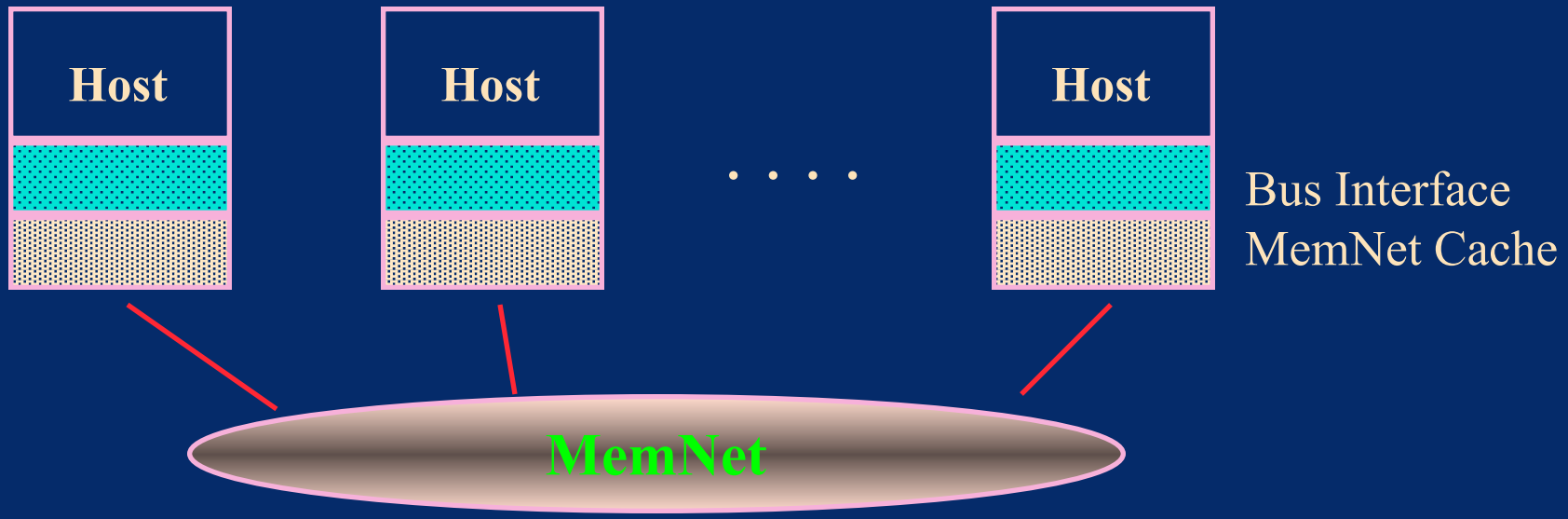


Case Study: MemNet (cont)

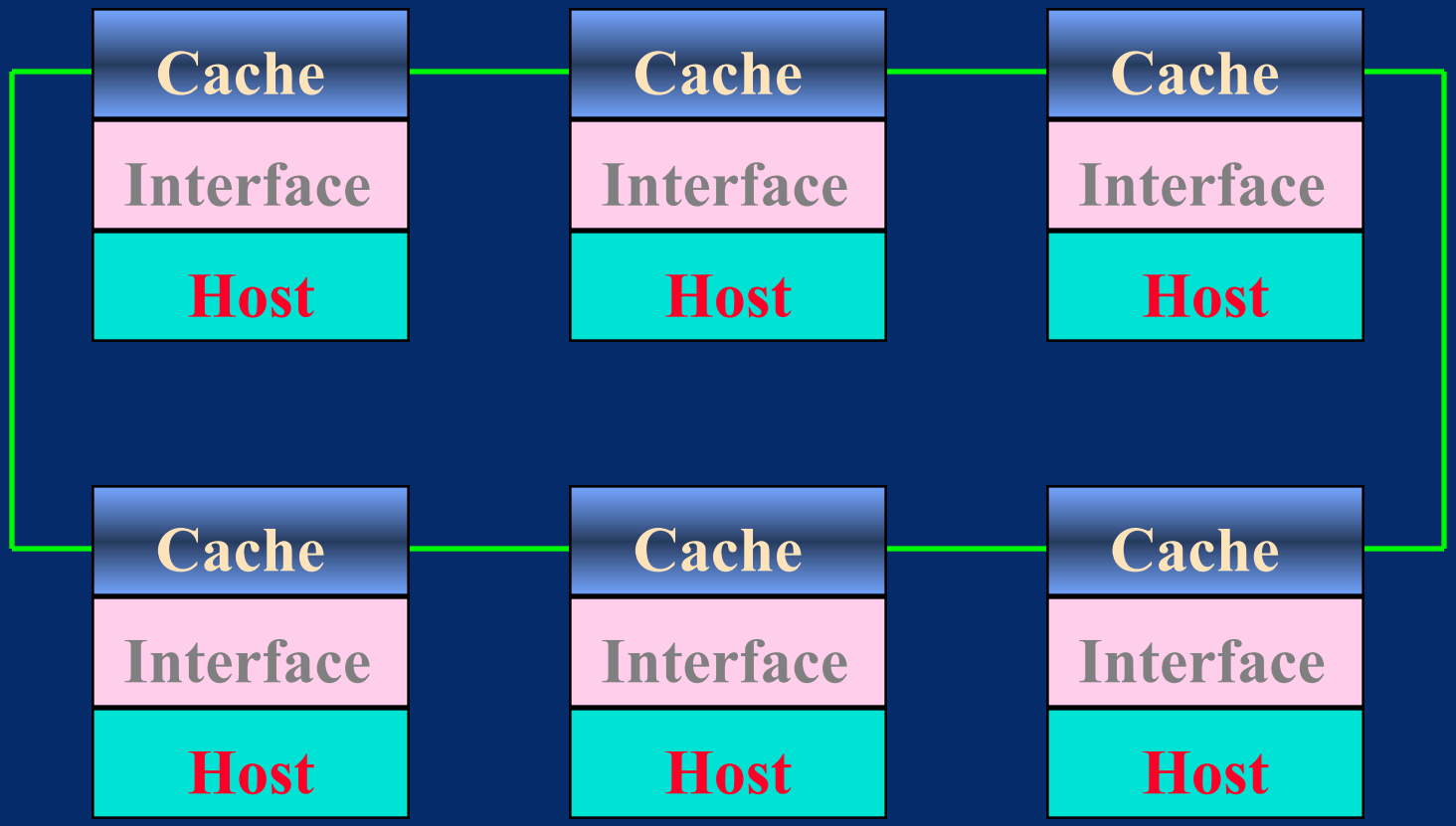
- on Read Request:
  - a message request travels around the token-ring until it reaches the MemNet devices that has the memory
  - The request message is then converted into a data message that has the requested data.
- on Write Request:
  - similar to Read request, but non-owner nodes invalidate their copies.
- Invalidation Message:
  - when one node needs to write to a page, it needs to invalidate other copies.
- Replacement Policy: Random: it has a large amount of memory.

Case Study: MemNet

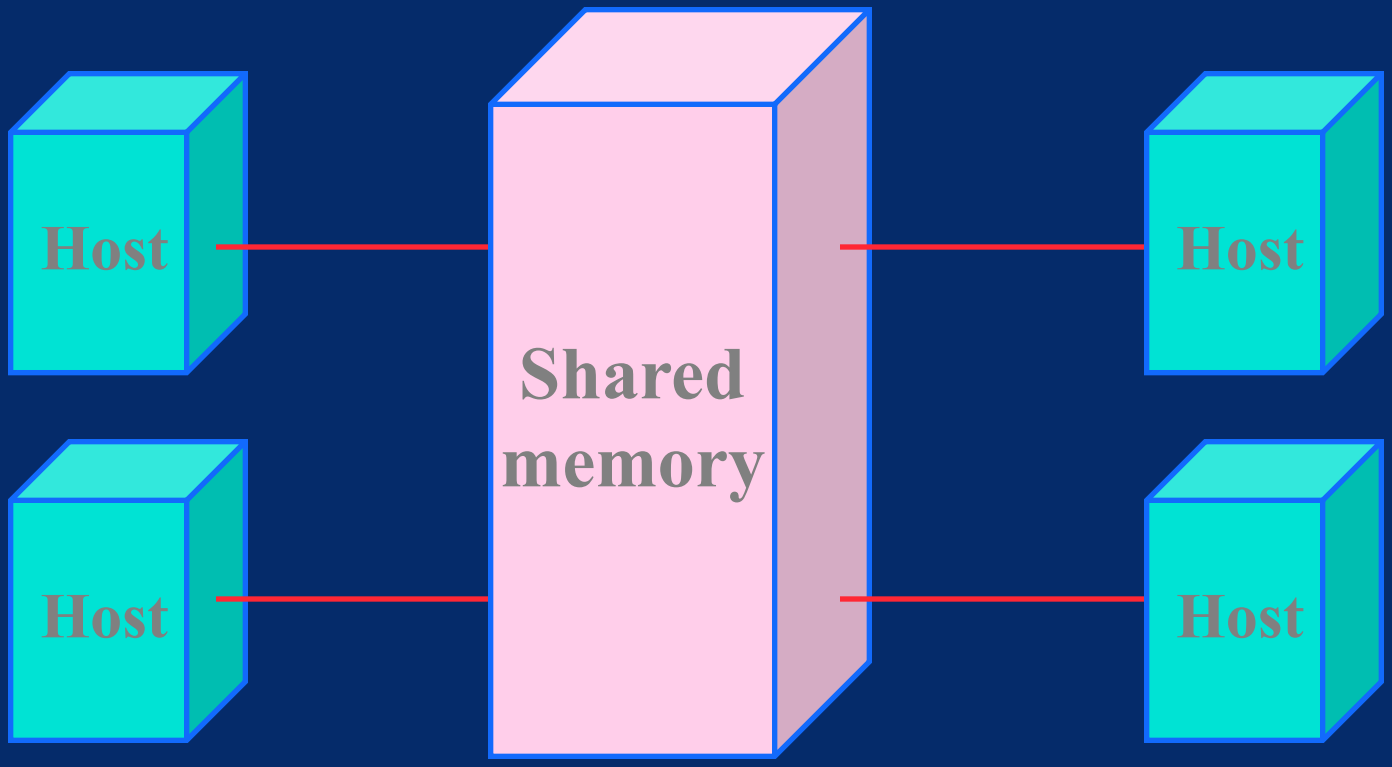
- to improve the performance of data migration by using hardware
- techniques
- The machines are connected to a MemNet device
- The device receives memory request (32-byte block).



MemNet System



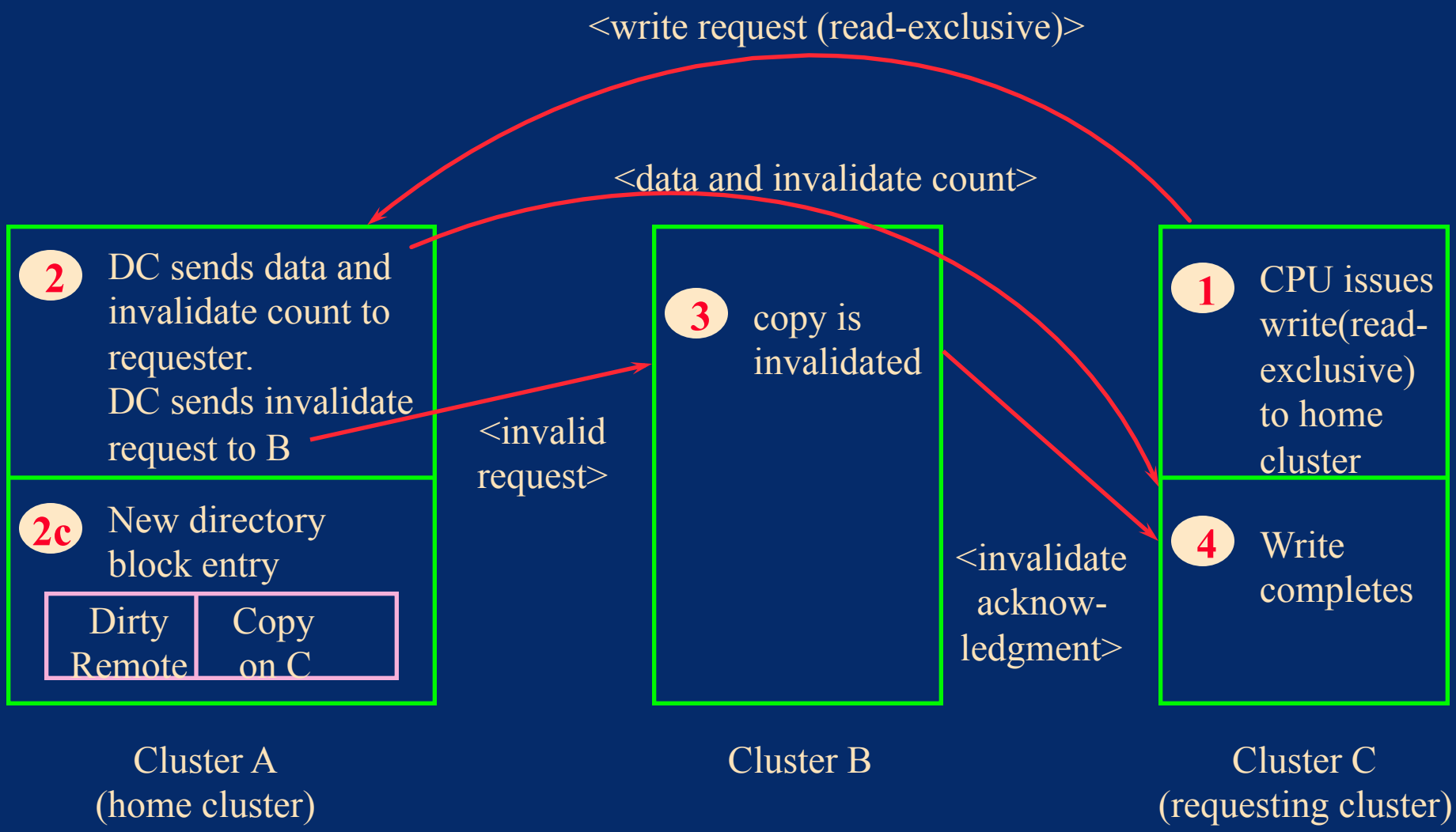
MemNet Project



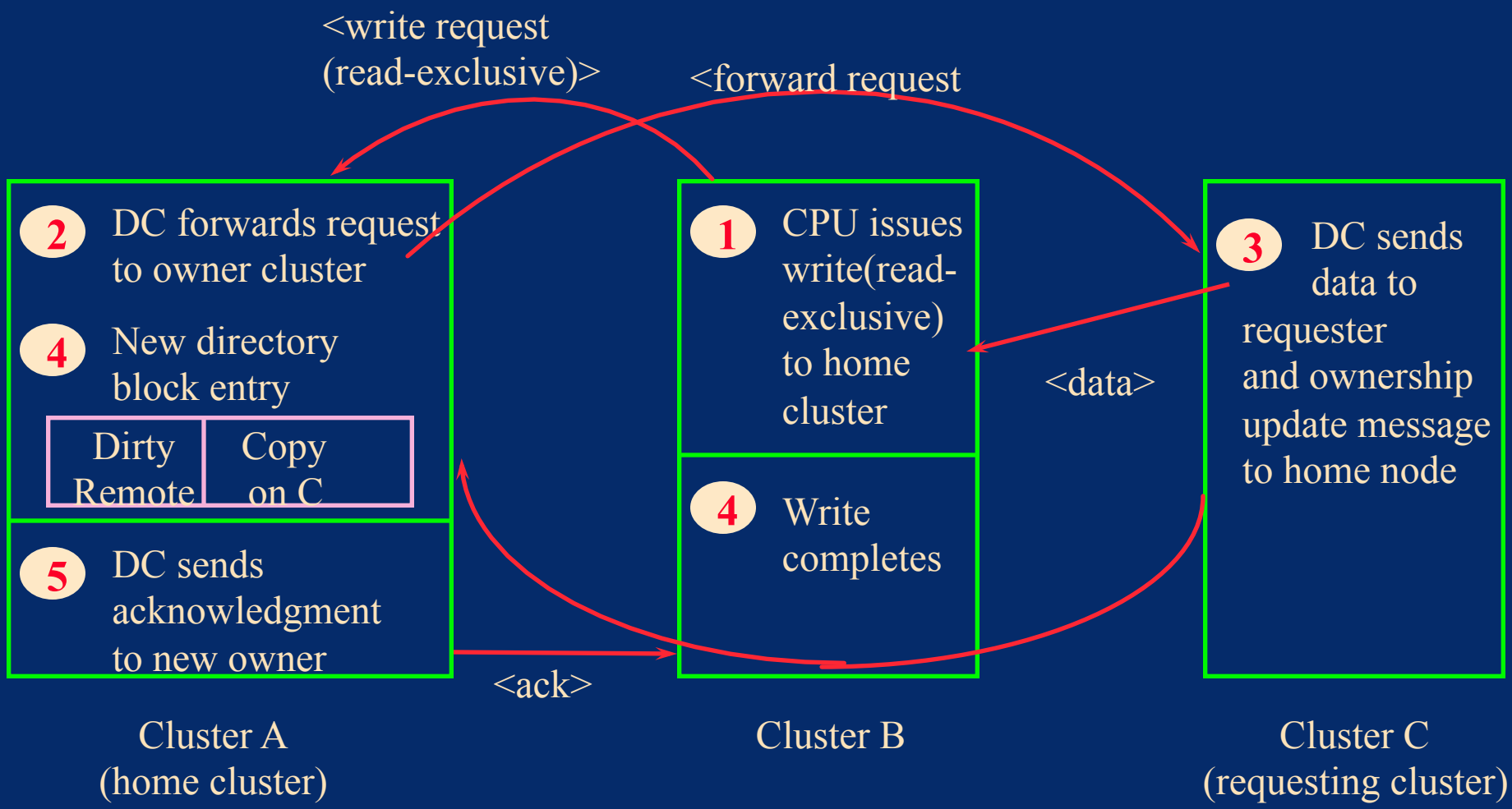
CASE STUDIES- DASH

- Stanford University Project
- A Hardware implementation of DSM
  - A directory based coherence protocol
  - Realease Consistency semantics

DASH DSM: Shared Remote Data



DASH DSM: Shared Dirty Data



METHER DSM

- It is a DSM implemented on Sun Workstations
- Processes share read, write, and execute access
- Mether project objectives were:
  - to demonstrate that DSM is practical even if page faults are handled in software
  - better understanding the applications interface to DSM
  - build a DSM on a NOW using conventional comm. protocols

First Implementation of Mether V0

- V0 was operational in November 1988
- it is a software MemNet: strong consistency and replicated only pages
- Problems observed were:
  - many programs used shared memory variables (locks, semaphores, etc.) for their synchronization
  - synchronization traffic affect network performance
  - programs spent significant amount of time checking unchanged variables
  - packet deliver was unreliable

Resolving the Problems with V0

- Inconsistent memory
  - a process may request the consistent copy, causing the uptodate copy to be transmitted over the network
  - the process holding the consistent copy, sends the new version via a system call (network refresh)
  - the local inconsistent copy will be discarded if it stays inconsistent for more than 5 seconds
  - the next time it needs that purged page, it fetches the page from the network

Resolving the Problems with V0- Cont.

- Short Pages
  - it is only 32 bytes to store important state variables
  - page faults cause only 32 bytes overhead as opposed to 8192 byte page
- User-driven page propagation
  - pages can be out-of-date, Mether provides mechanisms to propagate new copies of a page
    - It supports user-driven propagation; discard local inconsistent copy to force page fault during the next access
    - In systems supporting multicast, a writer can cause its copy to be broadcast to all holders of inconsistent copies; network refresh

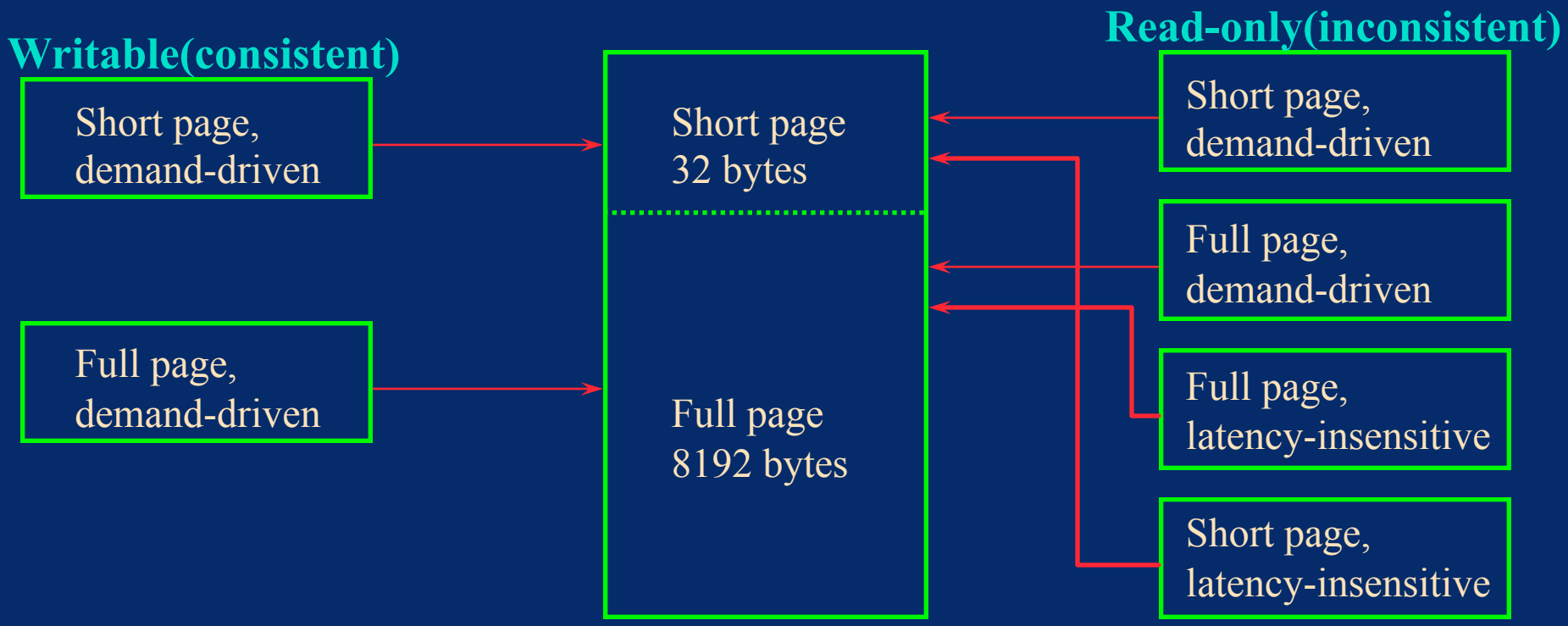
Resolving the Problems with V0- Cont

- Latency-incenstive Address Space
  - Mether provides an address space that is latency insensitive
  - it is used to support data-driven page fault
  - it is used to experiment with high latency communications environment

Resolving the Problems with V0- Cont

- Data Driven page Faults
  - in DSM, a page fualt always results in a request over the network for a page
  - in data-driven page fault, one process takes an action that causes another process's page fault to be satisfied
    - one process request a read, another process responds with a network refresh

METHER DSM



Notes : (1) The choice of the read-only space or the writable space is made when the application maps in the Mether address space

(2) The consistent space can be demand-driven only

(3) Teh choice of full or short page and demand- or data-driven are determened by two address bits in the Mether address space

(4) If further applications demand it, we may opt for four different page sizes - one more bit of address space

Mether DSM-Cont.

Operation	Rule for subsets	Rule for supersets
Mapping a page in	All subsets must be present	Supersets need not be present
Pagein from the network	All subsets paged in	No supersets paged in
Pageout	All subsets paged in	All supersets left paged in but unmapped
Lock	All subsets must be present; if all are present all are locked; otherwise, the lock fails and any nonpresent subsets are marked wanted	No supersets locked but must be present; all are unmapped; nonpresent supersets are marked wanted
Page fault	All subsets must be present	Supersets need not be present
Purge	All consistent subsets are purged	Supersets are not affected

CapNet- A distributed Shared Memory for WANS

- There are important differences between LANs and WANs
  - WANs have much larger latency
  - WANs can not effectively support broadcast
  - WANs have traditionally been bandwidth constrained
- Since broadcast is expensive, directory of page locations should be maintained by some page manager
  - Owner is defined as the host that made the last modifications to a given page
  - an owner honor a read request by sending the page and updating its copyset
  - a write request is honored by transfering ownership to the requesting host

CapNet Page-Location Scheme

- augment the packet switches with the information required to locate pages
- by distributing page table into network switches, the network can route a page request to the owner directly
- Each switch has a page table that indicates the outgoing hob leading to the owner of the page; similar to the routing table
- a host requesting a page, it is sent over the network that finds the page and transfers it to the requester; only two messages are used

Distributed Shared Memory

Coularis, Dollimore and Kindberg, *Distributed Systems, Concepts and Design,* Chapter 18

prepared by James Deak

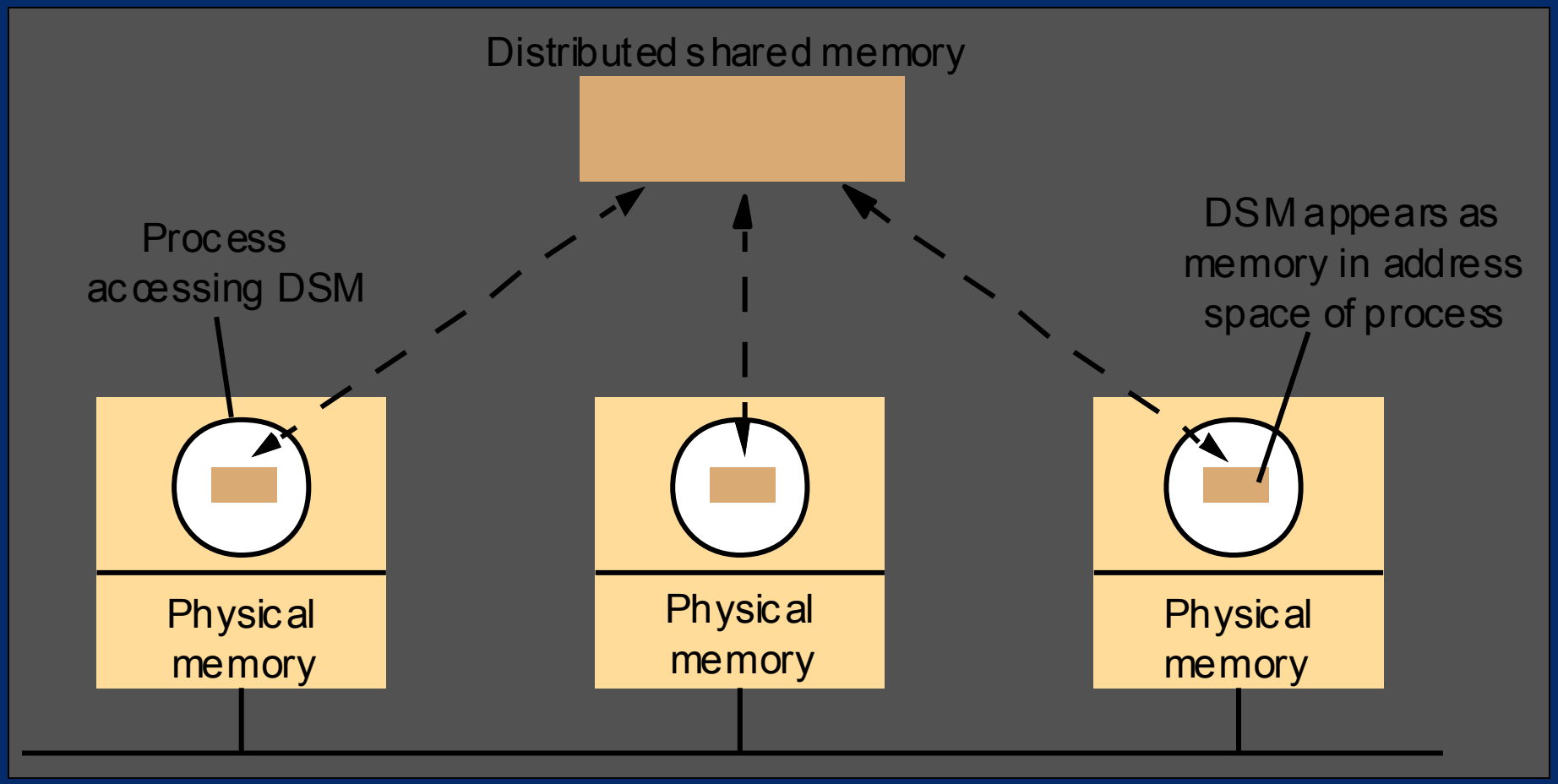
Distributed Shared Memory

- Distributed Shared Memory (DSM) allows programs running on separate computers to share data without the programmer having to deal with sending messages.
- Instead underlying technology will send the messages to keep the DSM consistent (or relatively consistent) between computers.
- DSM allows programs that used to operate on the same computer to be easily adapted to operate on separate computers.

Introduction

- Programs access what appears to them to be normal memory.
- Hence, programs that use DSM are usually shorter and easier to understand than programs that use message passing.
- However, DSM is not suitable for all situations. Client-server systems are generally less suited for DSM, but a server may be used to assist in providing DSM functionality for data shared between clients.

Figure 18.1 The distributed shared memory abstraction



DSM History

- Memory mapped files started in the MULTICS operating system in the 1960s.
- One of the first DSM implementations was Apollo. One of the first system to use Apollo was Integrated shared Virtual memory at Yale (IVY).
- DSM developed in parallel with shared-memory multiprocessors.

DSM implementations

- Hardware: Mainly used by shared-memory multiprocessors. The hardware resolves LOAD and STORE commands by communicating with remote memory as well as local memory.
- Paged virtual memory: Pages of virtual memory get the same set of addresses for each program in the DSM system. This only works for computers with common data and paging formats. This implementation does not put extra structure requirements on the program since it is just a series of bytes.

DSM Implementations (continued)

- Middleware: DSM is provided by some languages and middleware without hardware or paging support. For this implementation, the programming language, underlying system libraries, or middleware send the messages to keep the data synchronized between programs so that the programmer does not have to.

Efficiency

- DSM systems can perform almost as well as equivalent message-passing programs for systems that run on about 10 or less computers.
- There are many factors that affect the efficiency of DSM, including the implementation, design approach, and memory consistency model chosen.

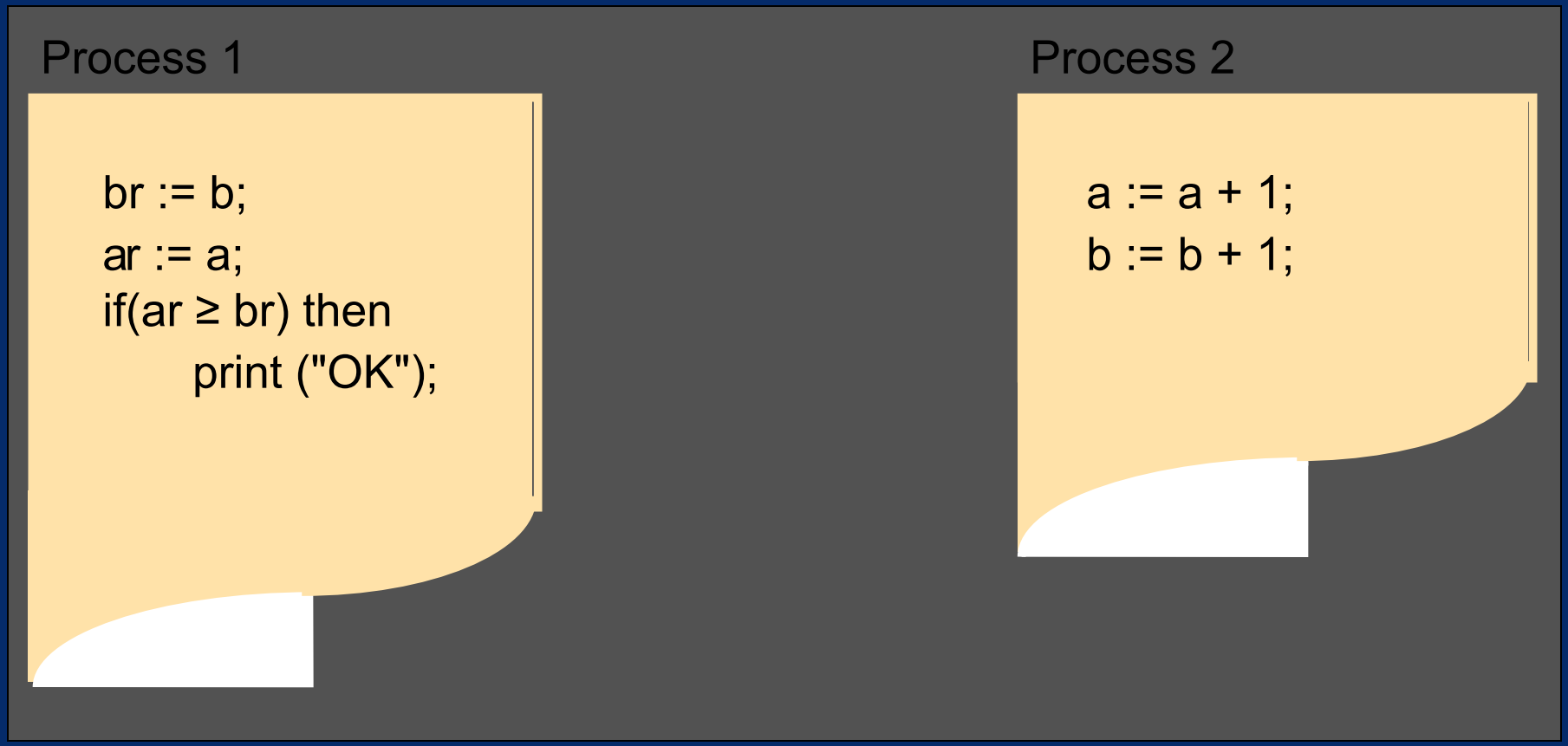
Design approaches

- Byte-oriented: This is implemented as a contiguous series of bytes. The language and programs determine the data structures.
- Object-oriented: Language-level objects are used in this implementation. The memory is only accessed through class routines and therefore, OO semantics can be used when implementing this system.
- Immutable data: Data is represented as a group of many tuples. Data can only be accessed through read, take, and write routines.

Memory consistency

- To use DSM, one must also implement a distributed synchronization service. This includes the use of locks, semaphores, and message passing.
- Most implementations, data is read from local copies of the data but updates to data must be propagated to other copies of the data.
- Memory consistency models determine when data updates are propagated and what level of inconsistency is acceptable.

Figure 18.3 Two processes accessing shared variables



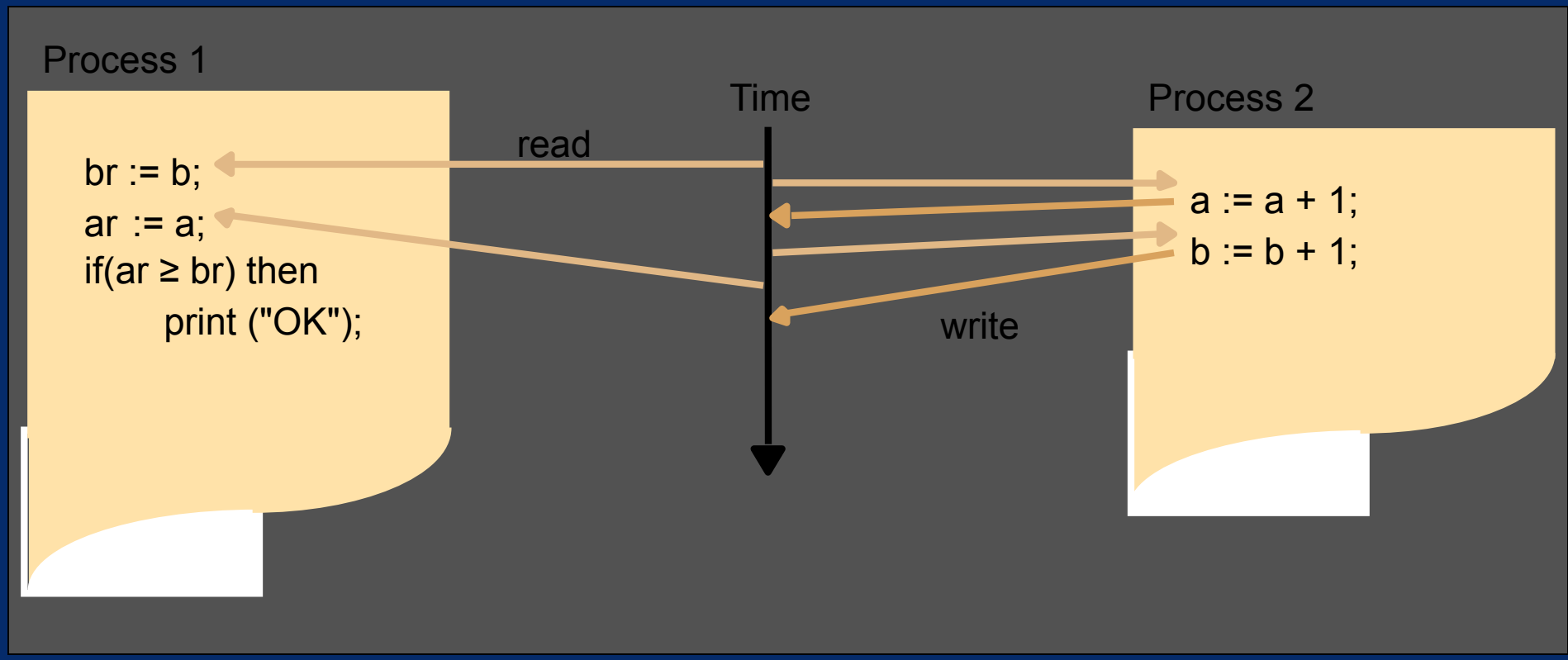
Memory consistency models

- Linearizability or atomic consistency is the strongest model. It ensures that reads and writes are made in the proper order. This results in a lot of underlying messaged being passed.
- Sequential consistency is strong, but not as strict. Reads and writes are done in the proper order in the context of individual programs.

Memory consistency models (continued)

- Coherence has significantly weaker consistency. It ensures writes to individual memory locations are done in the proper order, but writes to separate locations can be done in improper order.
- Weak consistency requires the programmer to use locks to ensure reads and writes are done in the proper order for data that needs it.

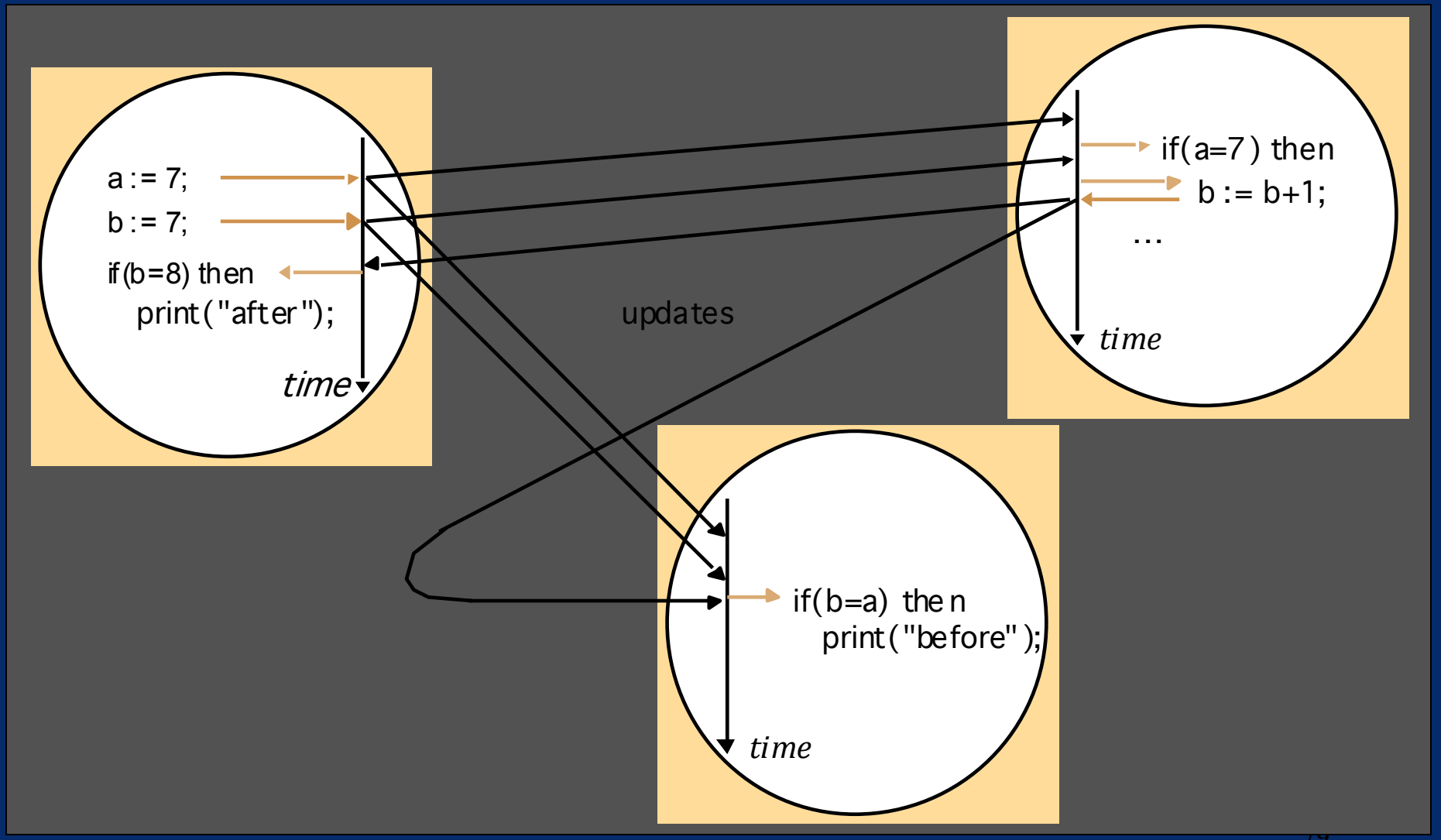
Figure 18.4 Interleaving under sequential consistency



Update options

- Write-update: Each update is multicast to all programs. Reads are performed on local copies of the data.
- Write-invalidate: A message is multicast to each program invalidating their copy of the data before the data is updated. Other programs can request the updated data.

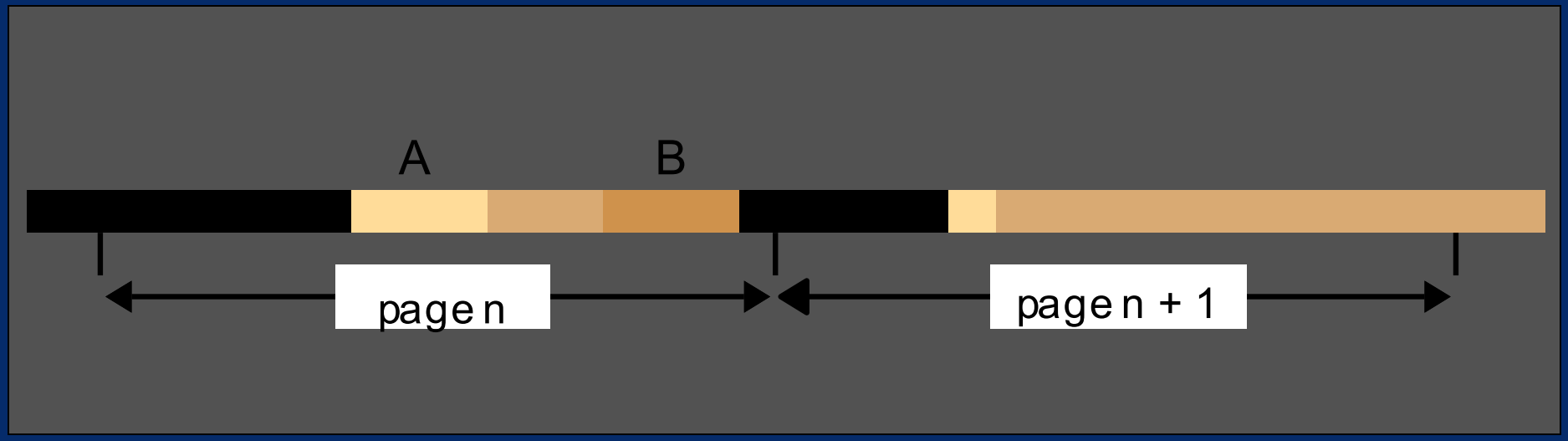
Figure 18.5
DSM using write-update



Granularity

- Granularity is the amount of data sent with each update.
- If granularity is too small and a large amount of contiguous data is updated, the overhead of sending many small messages leads to less efficiency.
- If granularity is too large, a whole page (or more) would be sent for an update to a single byte, thus reducing efficiency.

Figure 18.6 Data items laid out over pages



Trashing

- Thrashing occurs when network resources are exhausted, and more time is spent invalidating data and sending updates than is used doing actual work.
- Based on system specifics, one should choose write-update or write-invalidate to avoid thrashing.

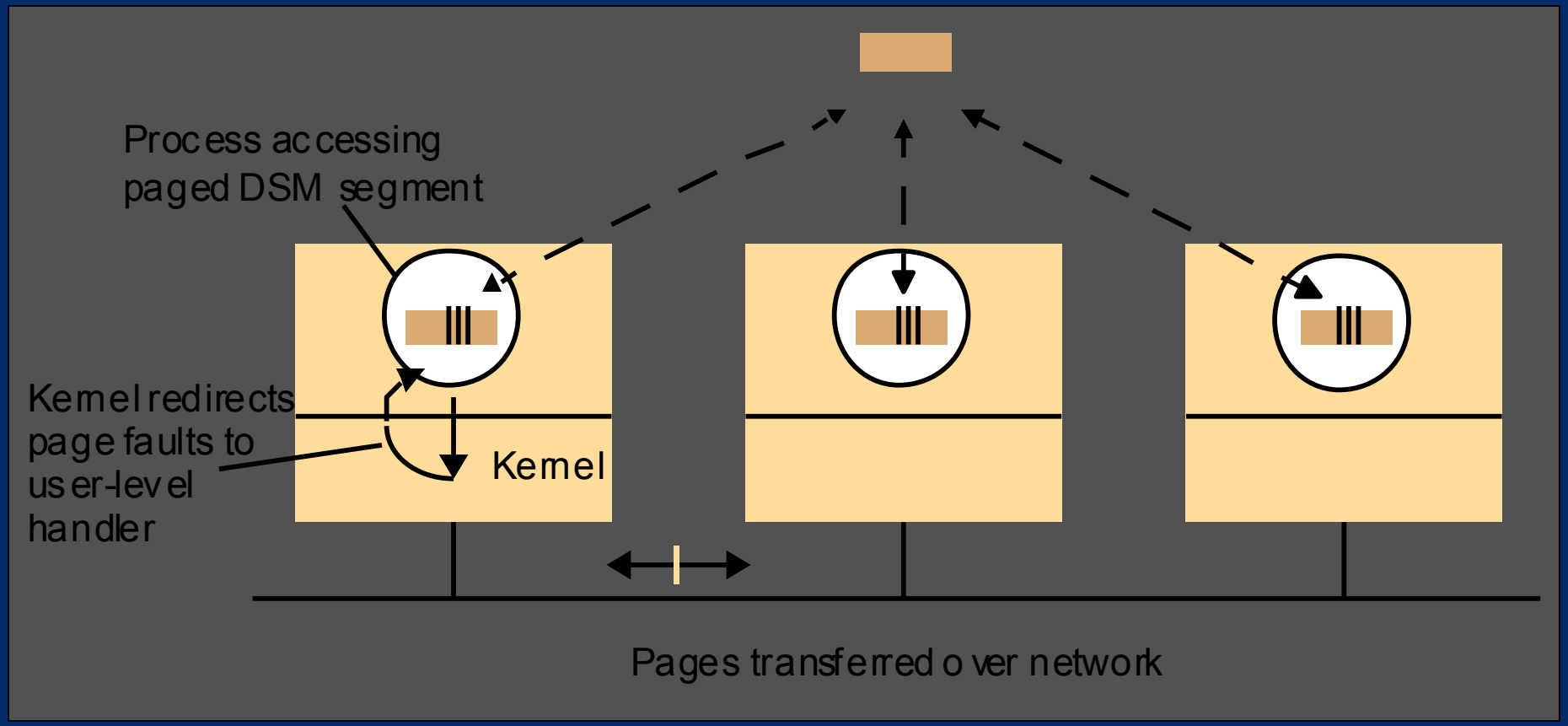
Sequential consistency and Ivy case study

- This model is page-based. A single segment is shared between programs.
- The computers are equipped with a paged memory management unit.
- The DSM restricts data access permissions temporarily in order to maintain sequential consistency.
- Permissions can be none, read-only, or read-write.

Sequential consistency and Ivy case study (continued)

- If a program tries to do more than it has permission for, a page fault occurs and the program is block until the page fault is resolved.
- Since this DSM is page-based, write-update is only used if writes can be buffered. Otherwise several consecutive updates to the same memory location or adjacent memory locations would result in several multicasts of the same page being updated.

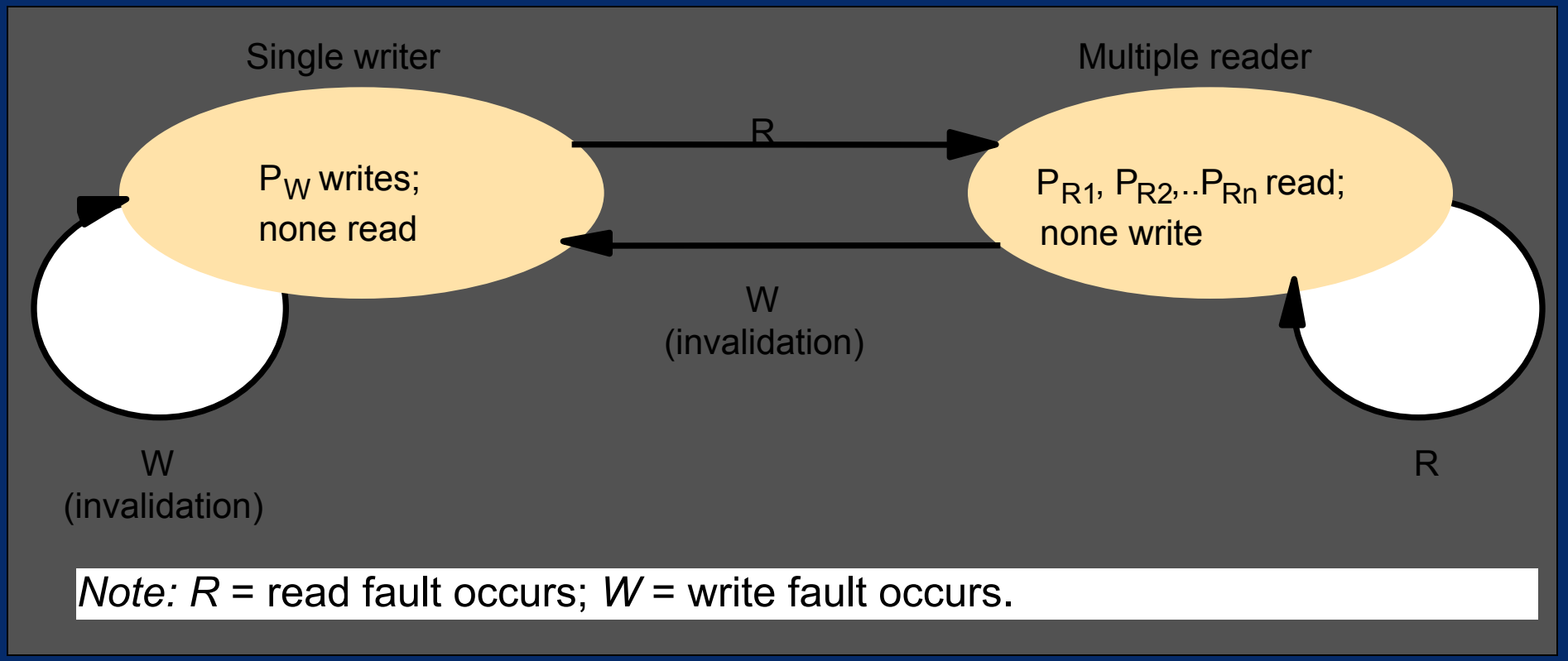
Figure 18.7 System model for page-based DSM



Sequential consistency and Ivy case study (continued)

- If writes cannot be buffered, write-invalidate is used.
- The invalidation message acts as requesting a lock on the data.
- When one program is updating the data it has read-write permissions and everyone else has no permissions on that page.
- At all other times, all have read-only access to the page.

Figure 18.8 State transitions under write-invalidation



Note: R = read fault occurs; *W* = write fault occurs.

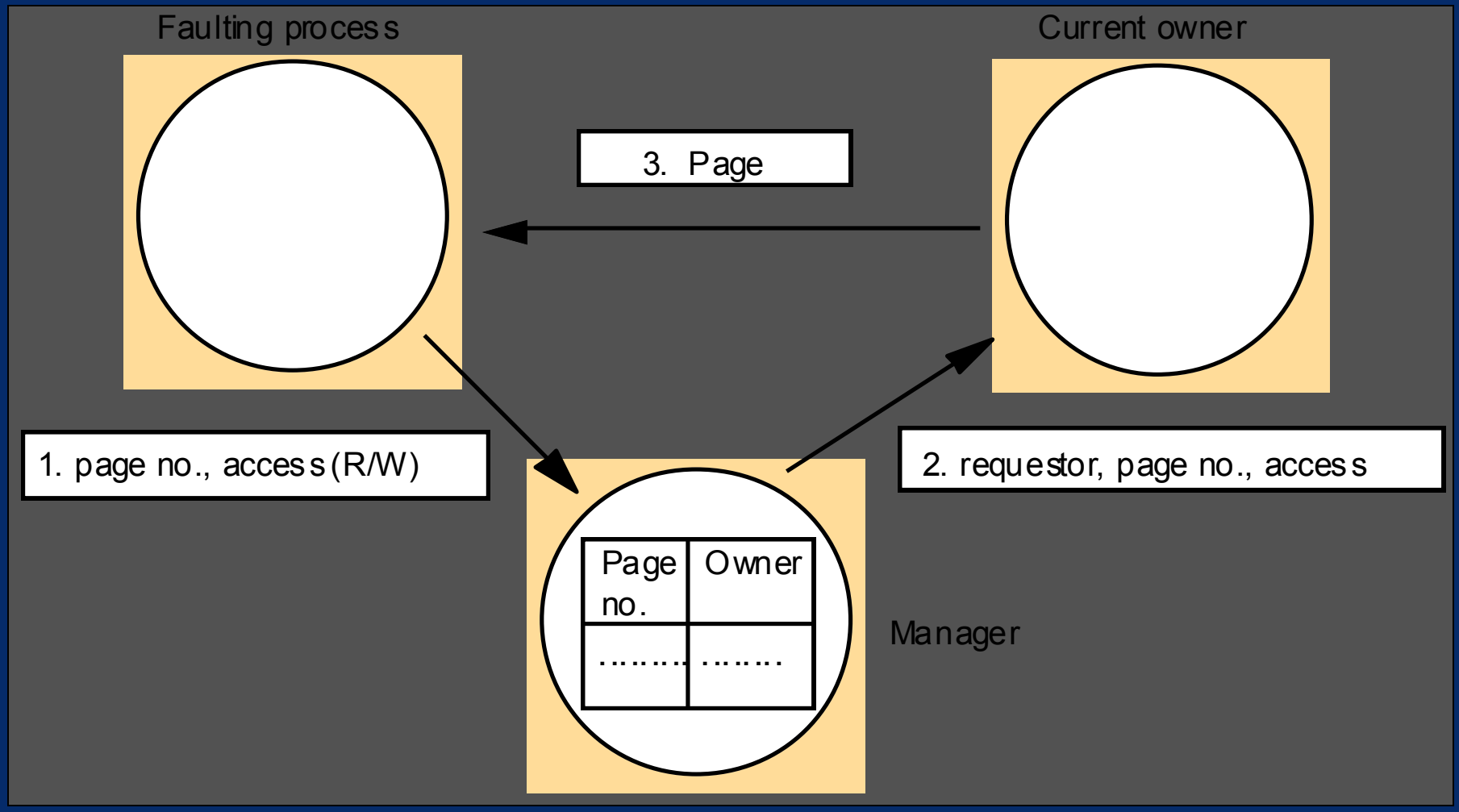
Sequential consistency and Ivy case study: State transitions

- When a program tries to write to a page for which it does not have read-write permission, a page fault occurs. An invalidate message is sent to all other programs. This sets the page permissions for those programs to none, and then the DSM system sets the page permissions for the writing program to read-write and unblocks it from the page fault.
- Two programs might request write access at close to the same time.

Sequential consistency and Ivy case study: State transitions

- If a program attempts to read a page it does not have permissions for a page fault occurs. The DSM system (on behalf of the reading program) will send a message (with the latest sequence number of its copy of the page) to the owner of the page. If the page owner determines the reader's sequence number does not match its sequence number of the page, it will send the whole page to the reading program. It will then grant read access to the page. If the page owner determines it does not need to access the page soon, it may transfer ownership to another program.

Figure 18.9 Central manager and associated messages



Sequential consistency and Ivy case study: Invalidation protocol

- A program must know who is the owner of the page that it needs. For this, they contact the central manager.
- The manager may be just another program in the DSM system, or it may be a separate server.
- When a page fault occurs due to inappropriate permissions, the message requesting access is actually sent to the central manager. The manager determines the page owner and forwards the message requesting access to the page owner. If the request is for a write page fault, the page ownership is transferred by the central manager to the requester.

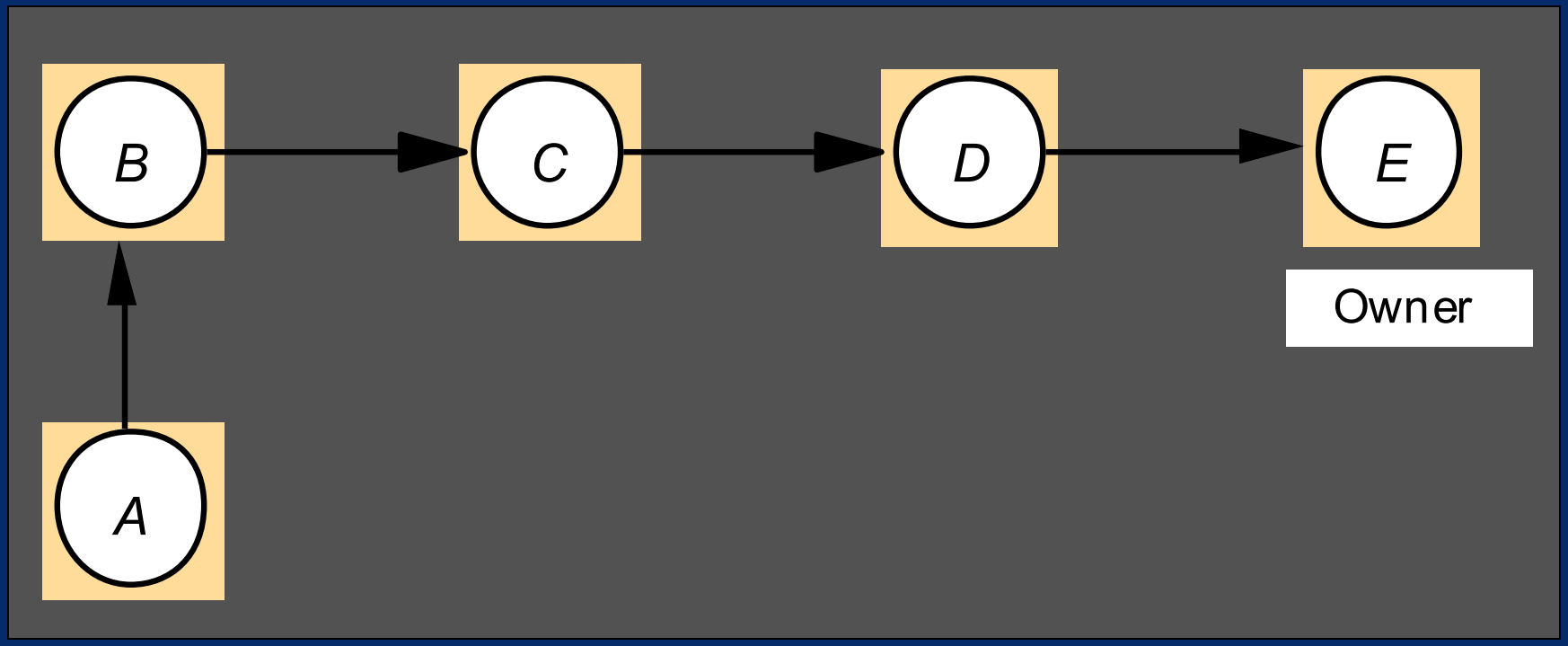
Sequential consistency and Ivy case study: Invalidation protocol

- For a write fault, the page's previous owner sends the page and the page's copy set to the new owner. The new owner performs the invalidation when it receives the page and copy set – it sends the invalidation message to the members of the copy set (excluding the previous owner who invalidate itself), thus revoking their read access to no access.

Sequential consistency and Ivy case study: Invalidation protocol

- A central manager may become a performance bottleneck. There are a few alternatives:
- A fixed distributed page management where on program will manage a set of pages for its lifetime (even if it does not own them).
- A multicast-based management where the owner of a page manages it, read and write requests are multicast, only the owner answers.
- A dynamic distributed system where each program keeps a set of the probable owner(s) of each page.

Figure 18.10 Updating probOwner pointers – slide 1



(a) *probOwner* pointers just before process *A* takes a page fault for a page owned by *E*

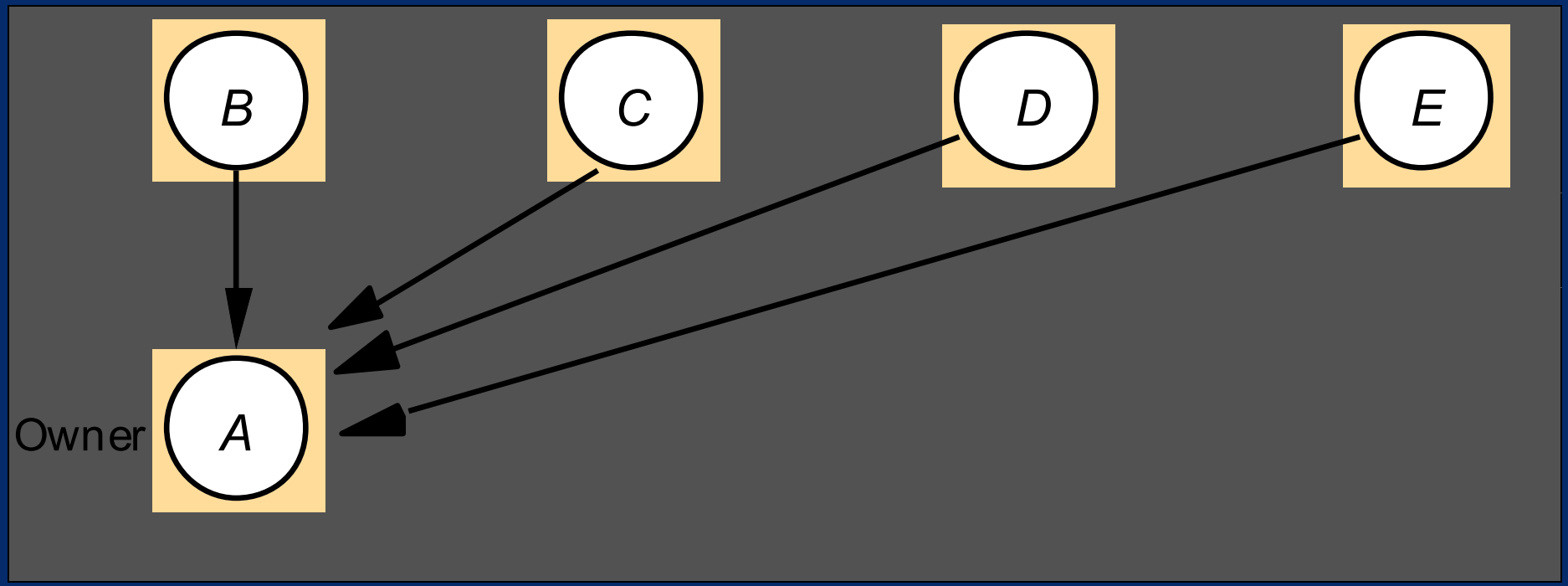
Sequential consistency and Ivy case study: Dynamic distributed manager

- Initially each program receives each pages owner and populates its probable ownership table.
- When an owner transfers ownership, it will update its own probable ownership table with the new owner. (This guarantees at least 2 programs know the correct owner.)
- When a program receives an invalidation message for a page, it updates its table to list the sender of that message as the owner.

Sequential consistency and Ivy case study: Dynamic distributed manager

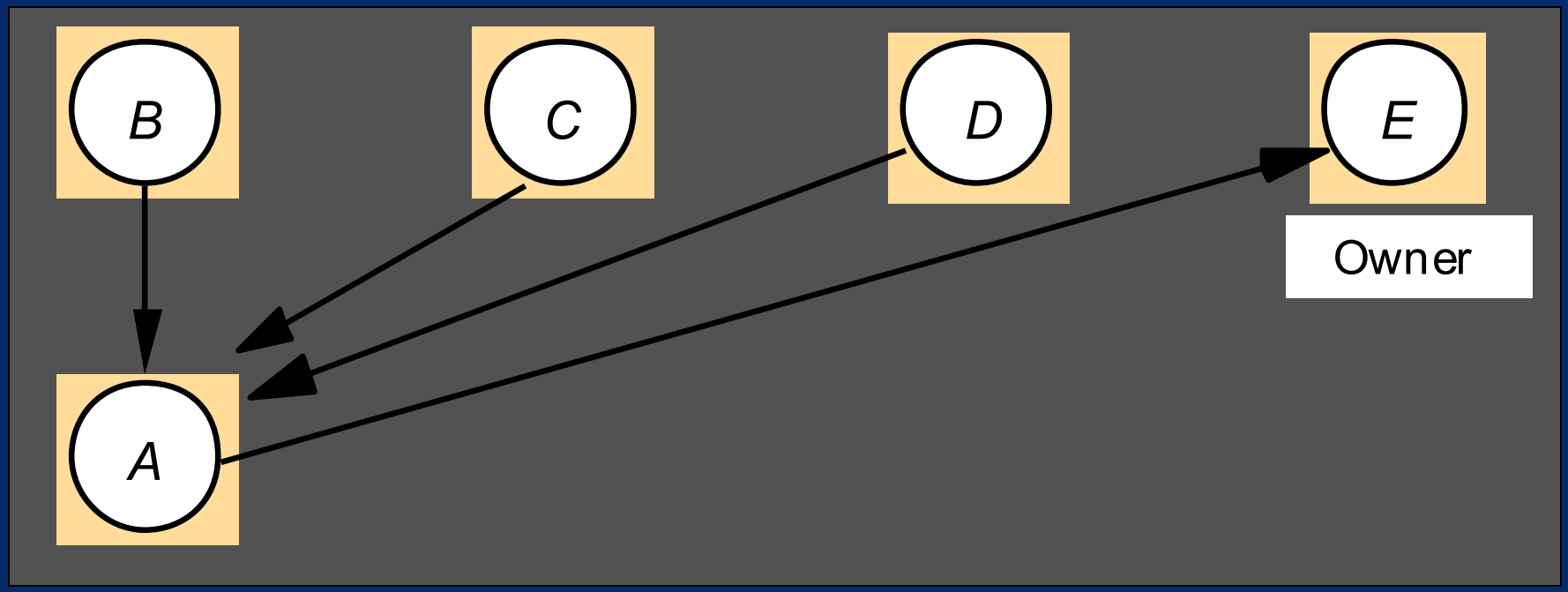
- When a program requests access to a page, it sends the request to whoever is listed in its probable owner table. When it receives the page, it will update its probable owner table with the sender of the page.
- If a program that receives a request for access does not own the page, it will forward the request to whoever is listed for the page in its probable owner table. It will then update its probable owner table to list the requester. Even if the requester does not become the new owner, it is about to find out who the correct owner is. By doing this the number of hops that a request can take before reaching the correct owner is limited.

Figure 18.10 Updating probOwner pointers – slide 2



(b) Write fault: *probOwner* pointers after *A*'s write request is forwarded

Figure 18.10 Updating probOwner pointers – slide 3



(c) Read fault: *probOwner* pointers after *A*'s read request is forwarded

Release consistency and Munin case study

- Release consistency is weaker than sequential consistency, but cheaper to implement.
- Release consistency reduces overhead. It relies on the fact that programmers can use semaphores, locks, and barriers to achieve enough consistency the system may need.

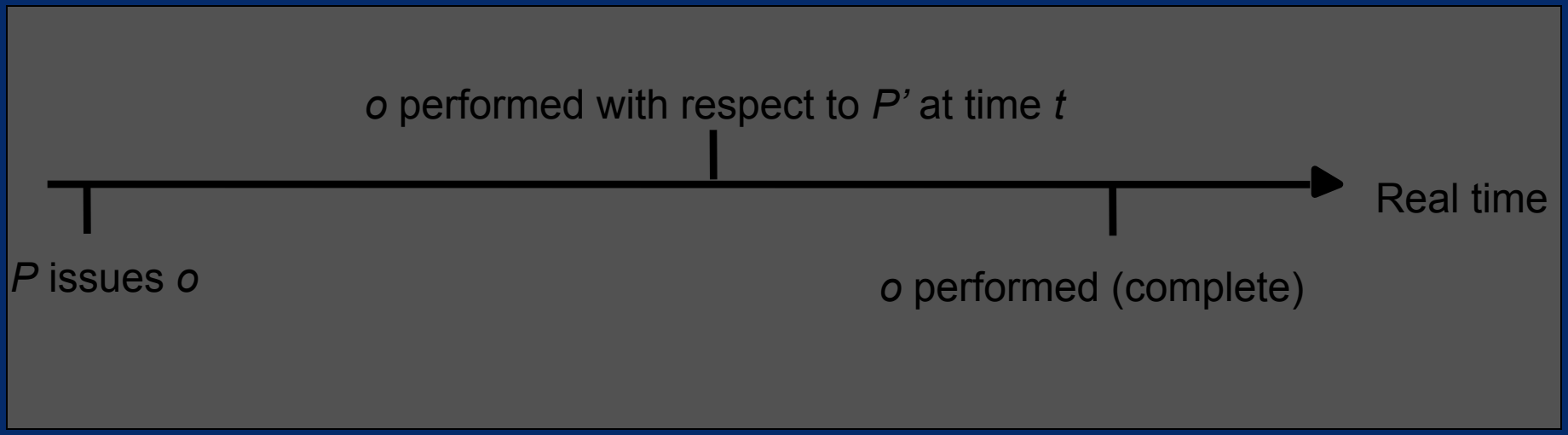
Release consistency and Munin case study: Memory accesses

- Types of memory accesses:
  - Competing accesses
    - They may occur concurrently – there is no enforced ordering between them.
    - At least one is a write
  - Non-competing or ordinary accesses
    - All read-only access, or enforced ordering

Release consistency and Munin case study: Memory accesses

- Competing memory accesses are divided into two categories:
  - Synchronization accesses are concurrent and contribute to synchronization. Examples include releasing a lock or a test-and-set operation.
  - Non-synchronization accesses are concurrent but do not contribute to synchronization.

Figure 18.11 Timeline for performing a DSM read or write operation



Release consistency requirements

- To achieve release consistency, the system must:
  - Preserve synchronization with locks, etc.
  - Gain performance by allowing asynchronous memory operations.
  - Limit the overlap between memory operations.

Release consistency requirements

- One must acquire appropriate permissions before performing memory operations.
- All memory operations must be performed before releasing memory.
- Acquiring permissions and releasing memory

Munin

- Munin had programmers use acquireLock, releaseLock, and waitAtBarrier.
- Munin allows programmers to mark the way data is shared. Munin optimizes DSM based on this. These marks can also pair locks and data, which guarantees the user has the data before accessing it.
- Munin sends updates/invalidations when locks are released. An alternative has the update/invalidation sent when the lock is next acquired

Figure 18.12 Processes executing on a release-consistent DSM

```
Process 1:
    acquireLock();              // enter critical section
    a := a + 1;
    b := b + 1;
    releaseLock();              // leave critical section
Process 2:
    acquireLock();              // enter critical section
    print ("The values of a and b are: ", a, b);
    releaseLock();              // leave critical section
```

Munin: Sharing annotations

- The following are options with Munin on the data item level:
  - Using write-update or write-invalidate.
  - Whether several copies of data may exist.
  - Whether to send updates/invalidate immediately.
  - Whether a data has a fixed owner, and whether that data can be modified by several at once.
  - Whether the data can be modified at all.
  - Whether the data is shared by a fixed set of programs.

Munin : Standard annotations

- Read-only : Initialized, but not allow to be updated.
- Migratory : Programs access a particular data item in turn.
- Write-shared : Programs access the same data item, but write to different parts of the data item.
- Producer-consumer : One program write to the data item. A fixed set of programs read it.
- Reduction : The data is always locked, read, updated, and unlocked
- Result : Several programs write to different parts of one data item. One program reads it.
- Conventional : Data is managed using write-invalidate.

Other consistency models

- Casual consistency – The happened-before relationship can be applied to read and write operations.
- Pipelining RAM – Programs apply write operations through pipelining.
- Processor consistency - Pipelining RAM plus memory coherent.

Other consistency models

- Entry consistency – Every shared data item is paired with a synchronization object.
- Scope consistency – Locks are applied automatically to data objects instead of relying on programmers to apply locks.
- Weak consistency – Guarantees that previous read and write operations complete before acquire or release operations.

Bibliography

- George Coularis, Jean Dollimore and Tim Kindberg, *Distributed Systems, Concepts and Design*, Addison Wesley, Fourth Edition, 2005
- Figures from the Coulouris text are from the instructor's guide and are copyrighted by Pearson Education 2005