Distributed Systems

16 Distributed Shared Memory

July-15-2009

Gerd Liefländer
System Architecture Group

Schedule of Today

- Motivation & Introduction
- Potential Problems with DSM
- Design of DSM
  - Single versus Multiple Copy DSM
  - Structure of DSM
  - Synchronization Model
  - Consistency Model
  - Update Propagation
- Implementation of DSM
- Examples of DSM
- Literature

Distributed Shared Memory

See textbook Coulouris et al.:

"Distributed Systems" Ch. 16 or 18

(depending on the edition)

Distributed Shared Memory

- Distributed Shared Memory (DSM) allows applications running on separate computers to share data or address ranges without the programmer having to deal with message passing
- Instead the underlying technology (HW or MW) will send the messages to keep the DSM consistent (or relatively consistent) between computer nodes
- DSM allows applications that used to operate on the same computer to be easily adapted to operate on multiple computers

What is a DSM?

- DSM is a special kind of a DDS, because this time main memory parts are distributed and shared
- Applications should see no difference between a local or remote memory access (except for further delays)
- Processes should see all writes by other processes (as fast as possible)

⇒ DSM design and implementation must provide access transparency

- DSM is not suitable for all situations, e.g. client server applications

Motivation

Why DSM? (compare: why shared memory in local systems?)

- Some programmers want one programming concept for distributed applications without all this IPC stuff

What's easier?

- Sharing data no longer requires explicit IPC (even though a DSM is based upon IPC)
- History has shown: Distributed applications based on application IPCs tend to have more program bugs and have larger code than DSM applications
- Shared memory was fastest collaboration in local systems, as long as there are only few conflicting operations

Why DSM?

- Better portability of distributed application programs
  - Natural transition from sequential to distributed application
- Better performance of some applications
  - Data locality, on-demand data movement, and larger RAMs reduce network traffic due to remote paging
  - However, ping-pong paging due to false sharing etc. must be avoided
- Flexible communication environment
  - Sender and receiver must not know each other
  - No need that they do coexist at the same time
- Ease of process migration
  - Migration is completed only by transferring the corresponding PCB (including its ASCB) to the destination

DSM Implementations

- Hardware
  - Mainly used by SMPs. HW 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 (2)

- 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.

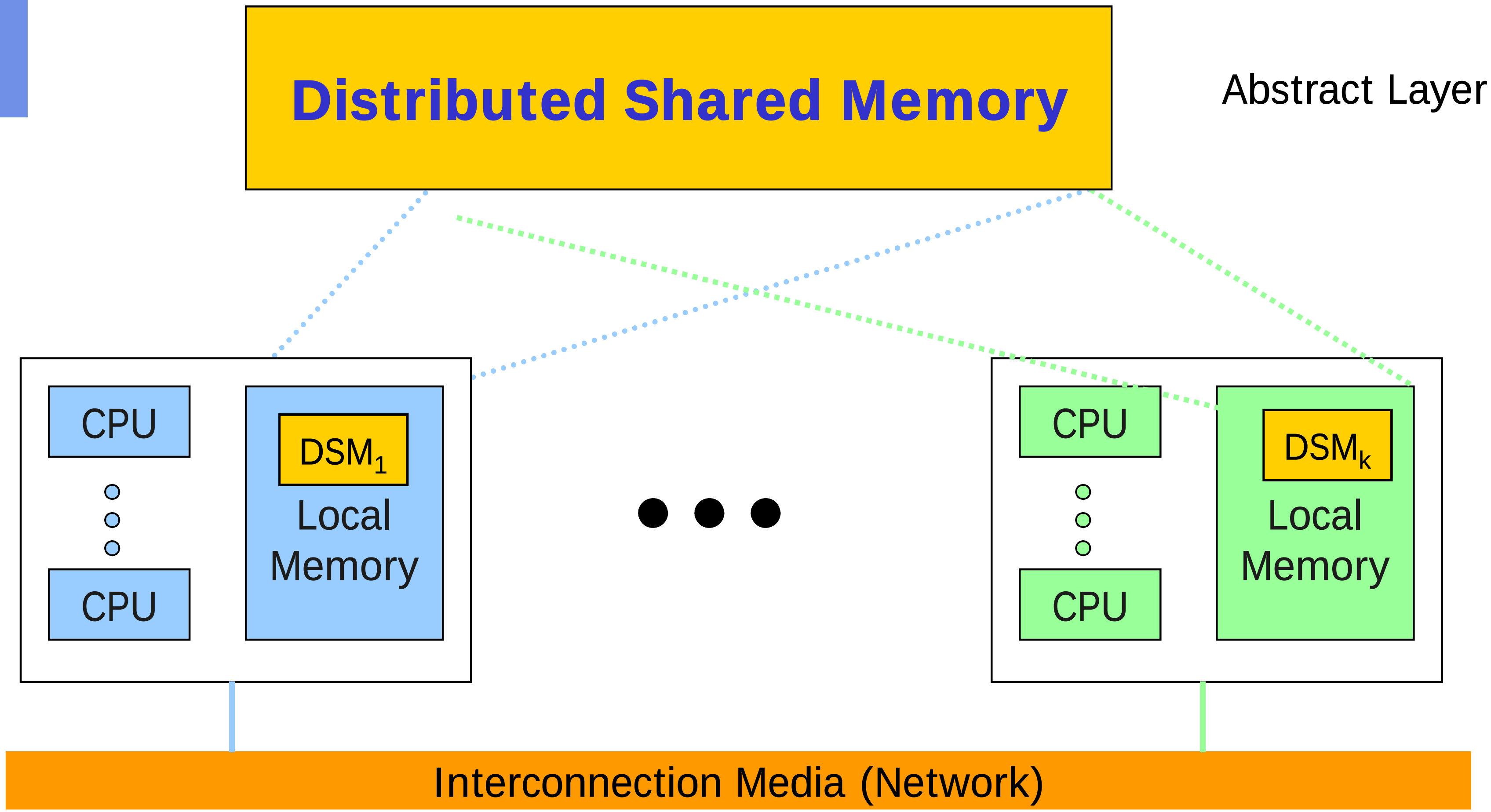
Typical Applications of DSM

- Multiple processes sharing
  - memory mapped files (already in MULTICS)
  - large global data, e.g. matrices etc. in parallel numeric applications
- DSM has to track
  - how many replicas currently exist and
  - where the current replicas are mapped
- Some DSMs offer
  - one copy of each read only page and of each read/write page
  - one copy of read/write page, but at least replicated read-only pages

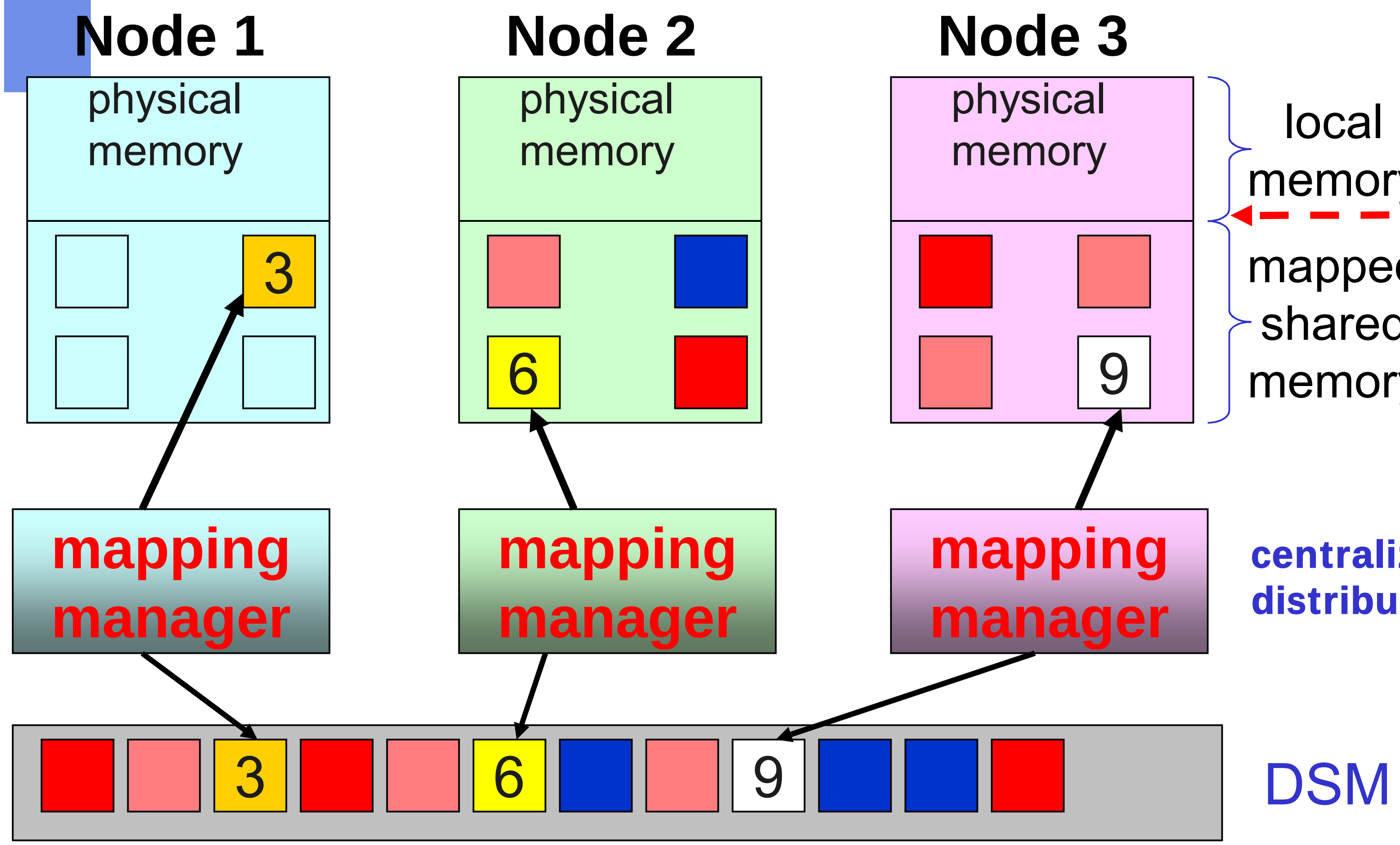
Efficiency of DSM

- DSM systems can perform almost as well as equivalent message-passing programs for systems that run on N~ 10 or less nodes
- There are many factors that affect the efficiency of a DSM, e.g.
  - implementation
  - design approach
  - memory consistency model

Architecture of a DSM



Page Based DSM



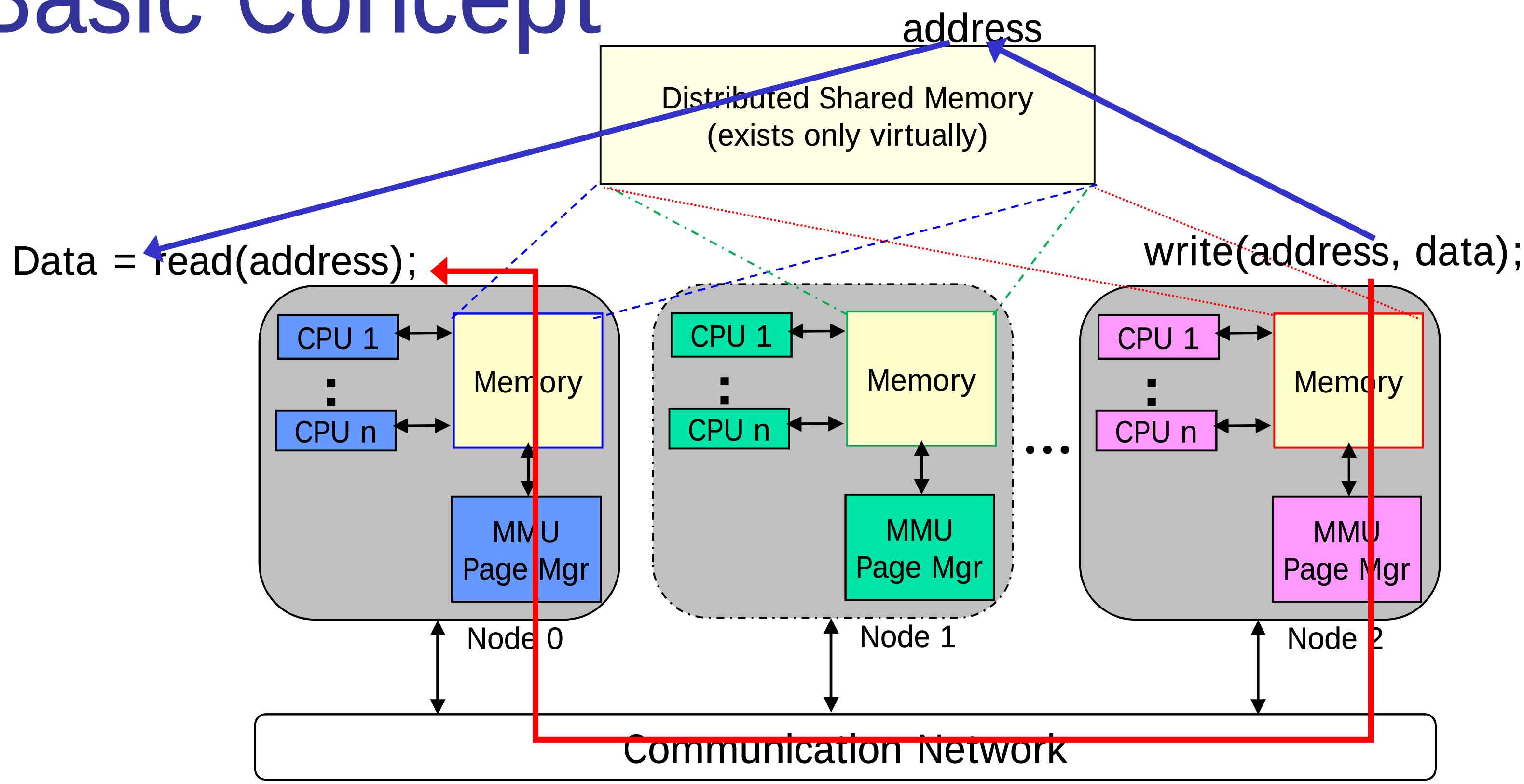
Who is Sharing Memory in a DSM?

- A multi-threaded task of KLTs, whose KLTs have migrated to n>1 nodes of the DS
  - Thread programmers know about the shared data and have to avoid write/write conflicts as usual using critical sections
  - Challenge: we have to provide that each KLT can do its read- & write operations on shared data without too much delay, i.e. the handling of a remote page fault should not take significantly more time than handling a local page fault
- Multi-process applications that have one or more data segments in common
  - It is convenient when a programmer can specify how specific parts of his segments are implemented with respect to sharing

No Usage of DSM

- Typical client/server system applications do not profit so much from DSM, because the clients often see the resources offered by a server as an abstract data type that can be used using RPC or RMI
- Furthermore, a server is often not interested that a unknown (malicious) clients access its data, i.e. in this case sharing might be too dangerous due to security reasons

Basic Concept



- Local pager must know the current location of an unmapped page
- Local pager must know the location of a centralized super-pager responsible for the tracking of all page/frame locations of the DSM

Potential Problems with DSM

Main Issues

- Memory coherence and access synchronization
  - Strict, Sequential, Causal, Weak, and Release Consistency models
- Data location and access
  - Broadcasting, centralized data locator, fixed distributed data locator, and dynamic distributed data locator
- Replacement strategy
  - LRU or FIFO, and using secondary store or the memory space of other nodes (COMA)
- Thrashing (due to false sharing, i.e. ping-pong effect)
  - *How to prevent a block from being exchanged back and forth between two nodes over and over again*

Granularity

Granularity = amount of data sent with each update

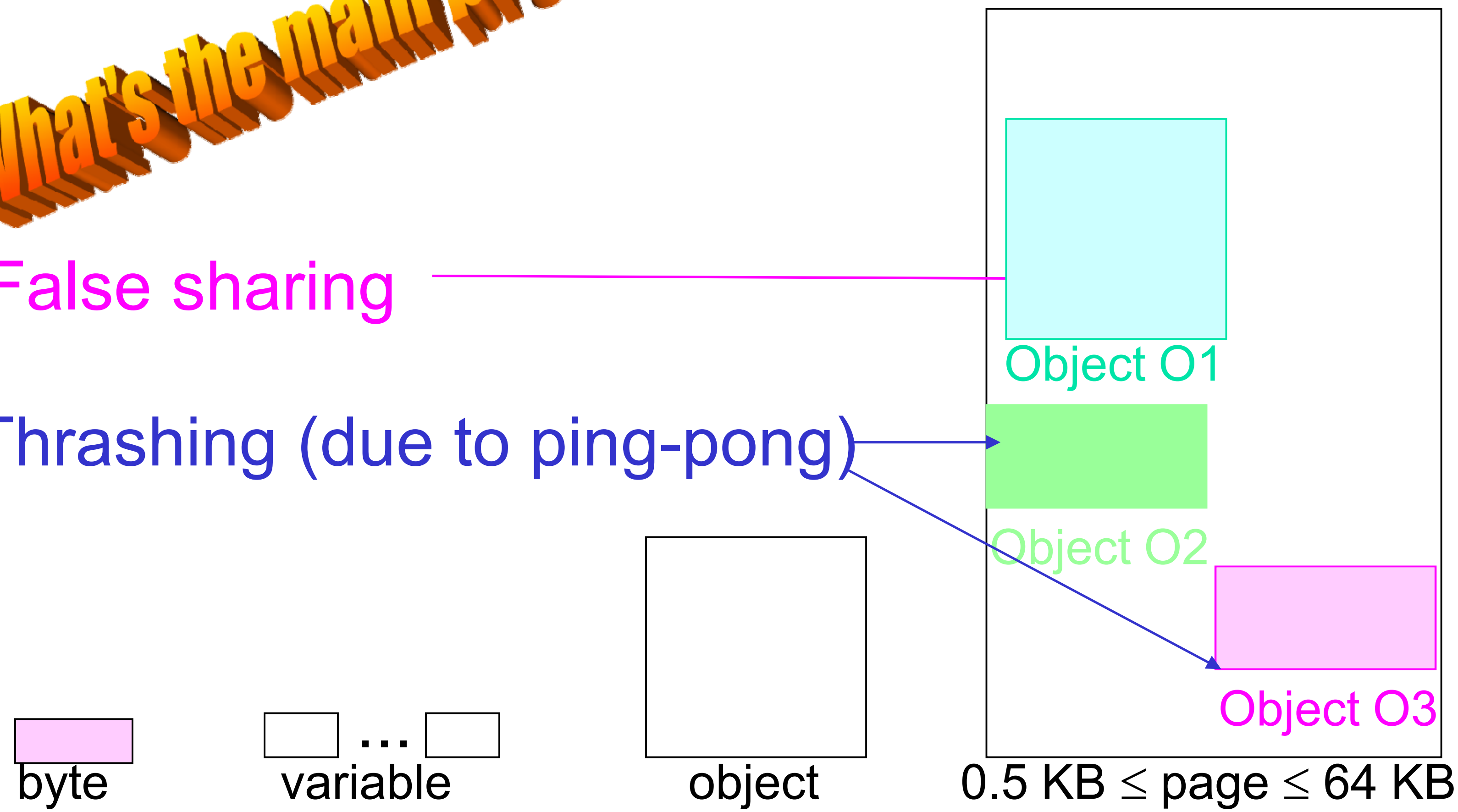
- If granularity is too small and a large amount of contiguous data is updated, the overhead of sending many small update-messages can reduce efficiency
  - Fine (less false sharing but more network traffic, e.g. object in Orca & Linda)

- If granularity is too large, a whole page (or more) would be sent for an update to a single byte, thus reducing efficiency
  - Coarse (more false sharing but less network traffic, e.g. page in Ivy)

Granularity Problem

What's the main problem?

1. False sharing
2. Thrashing (due to ping-pong)



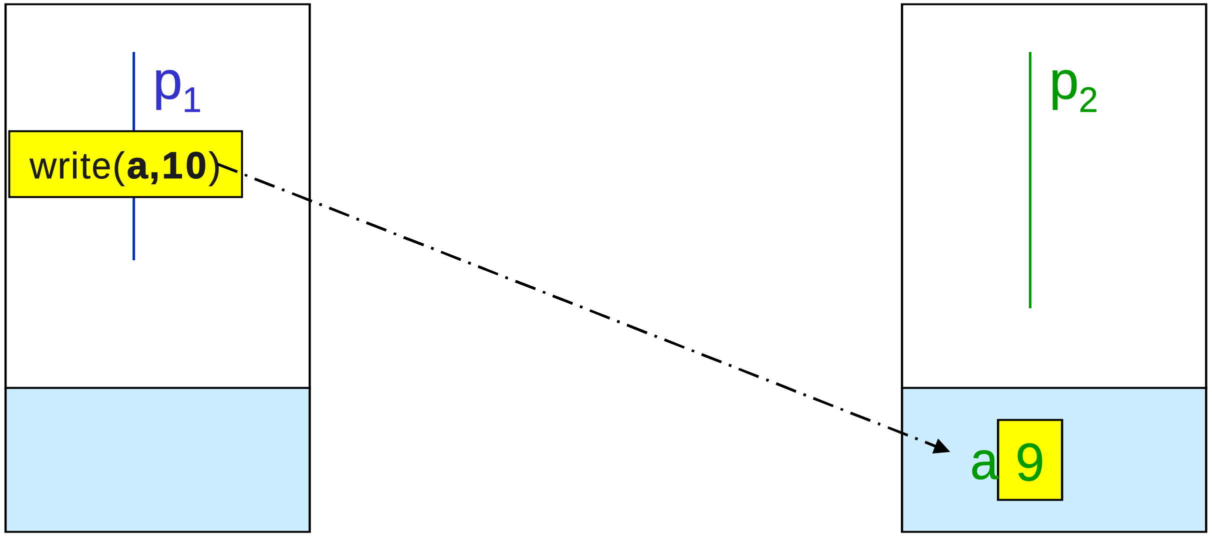
Granularity in a Page-Based DSM

- Typical standard page size 4 KB might be too small to host a typical shared data object
- However, using super pages might not pay off
  - Migrating a super page requires bandwidth and it might be difficult, to find a fitting memory hole for the super page
  - Furthermore, the larger the super page size the larger the potential internal fragmentation
- In a DS there are some applications that might run faster when using smaller than 4KB pages
- A 4 KB page might contain too many different objects $\Rightarrow$ false sharing, i.e. the ping-pong paging effect due to conflicting activities at different nodes

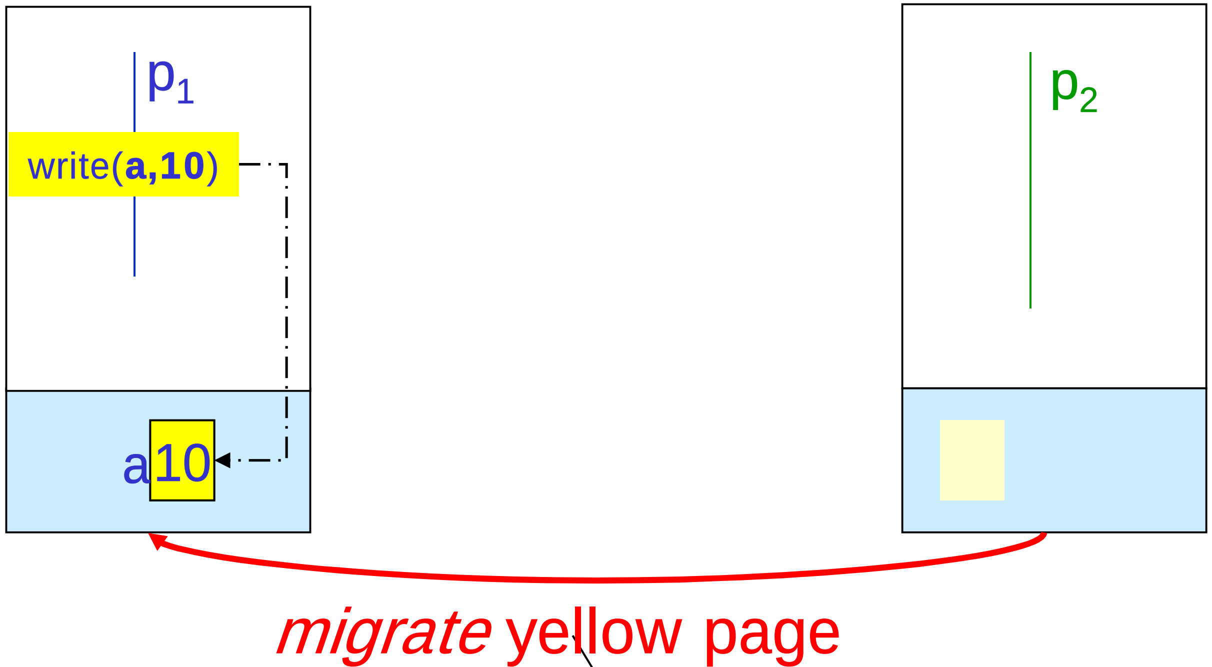
Thrashing in Single Copy DSM

Example:

∃ 2 processes on different nodes sharing one page

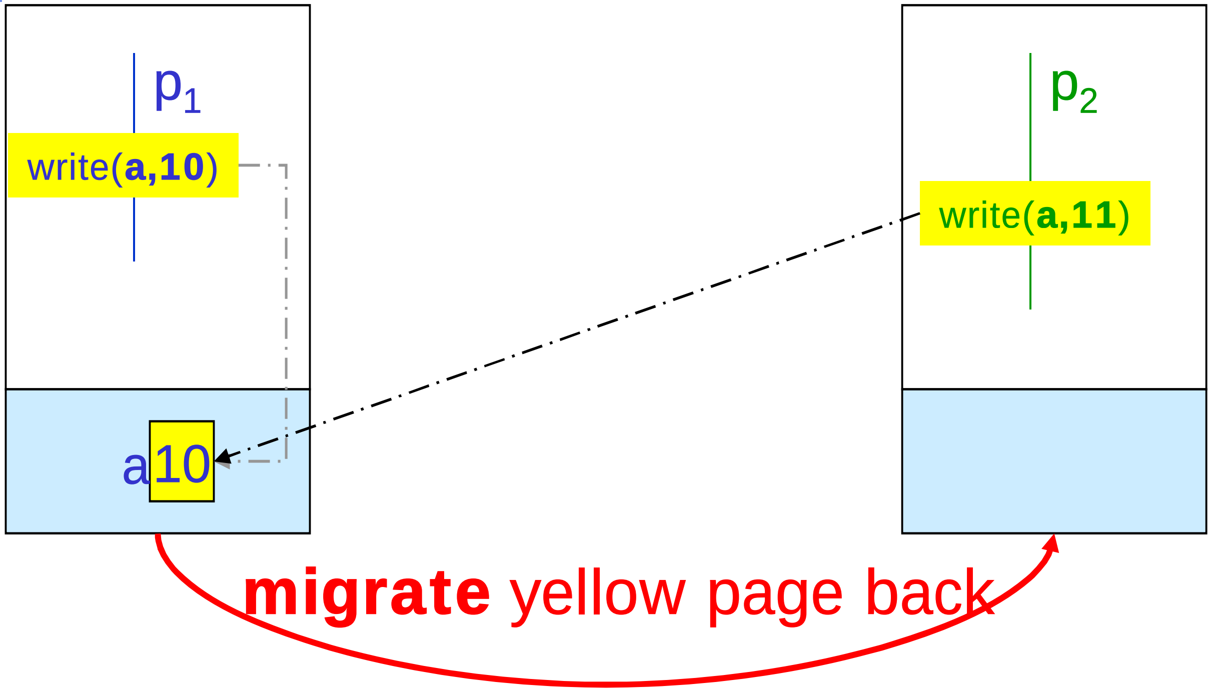


Thrashing in Single Copy DSM



Instead of copying “10” from node 1 to node 2 we migrate the complete page, i.e. we handle a “remote page fault”

Thrashing in Single Copy DSM

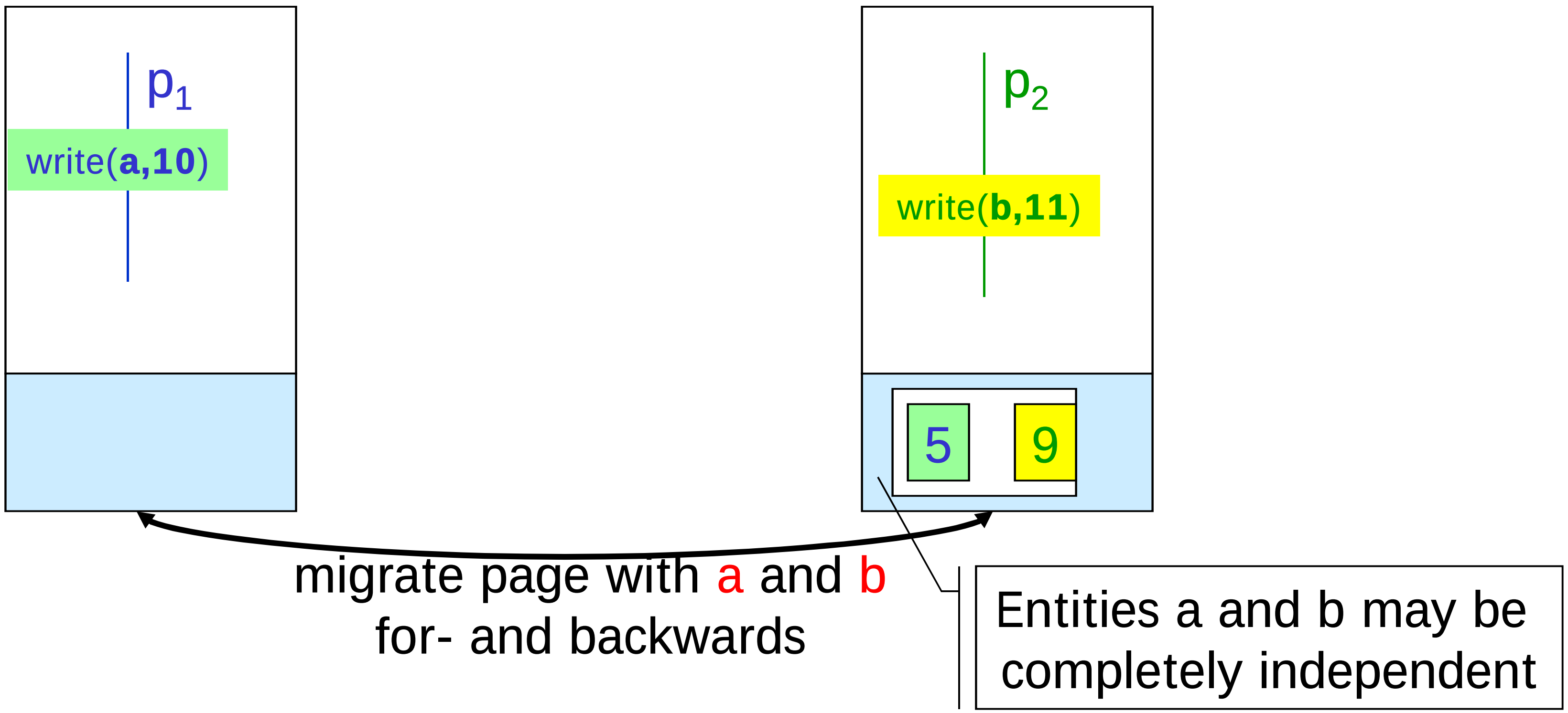


Thrashing in a Single Copy DSM

Thrashing in Single Copy DSM

Example:

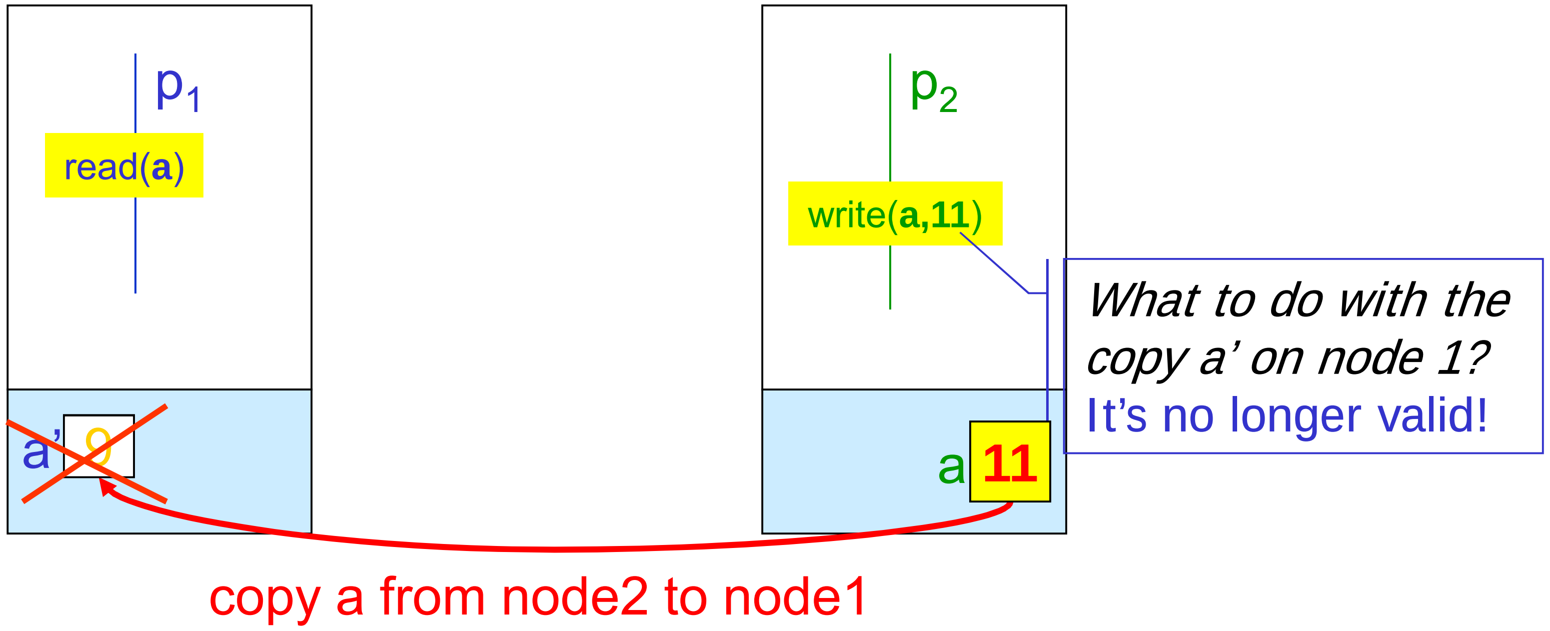
p_1 only writes to object a, p_2 only writes to object b, however, both objects are in the same mapping/migration unit (e.g. page).



Thrashing in Multi Copy DSM

Example:

p_1 reads a, p_2 writes a, both nodes have replicas of a.

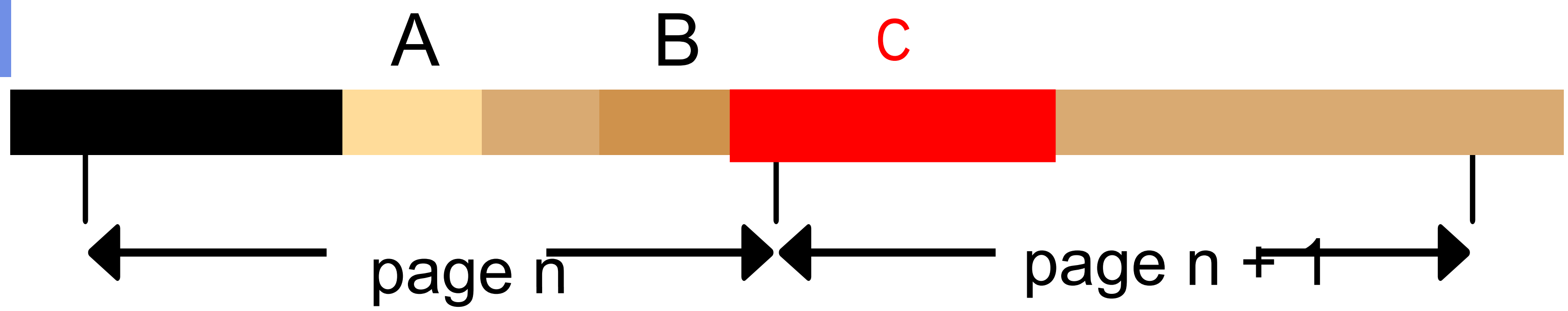


Remark:

Before writing to a data item in a replicated page, we must invalidate all replicas

We must solve similar problems as with coherent caches in a SMP

Data Items Laid out over Pages



- Danger of false sharing when process1 accesses data item A, and process2 accesses data item B concurrently
- Danger of two page faults in case of data item C is located on two different pages

Design of DSM

Single Copy versus Multiple Copy DSM

Structure of DSM

Synchronization Model

Consistency Model

Update Options

Two DSM Principles

- Single copy, i.e. without replication
  - If entity = page, ⇒ implement **remote paging**, i.e. instead of swapping to and from a local disk, swap via network or from a (local/)remote disk

- Multiple copies, i.e. with replication
  - If entity = page, no problems with replicated read-only pages, but we must deal with reader/writer-problems
    1. Single copy for read-write pages
    2. N>1 copies of read-write pages with additional owner bit, i.e. we must enforce that all writes are done on all copies in the same order

Structure of DSM

- Byte oriented
  - Access to a part of a byte-oriented DSM corresponds to an access to a virtual memory, e.g. Ivy & Mether
- Object oriented
  - DSM is a collection of objects
  - Operations are the methods of the object type, e.g. Orca serializes automatically all methods of the same object
- Constant data
  - No updates, but new versions

Synchronization Model

- To enable synchronization on byte-oriented DSM or synchronized methods in object-oriented DSM we have to provide solutions enabling mutual exclusion
  - Centralized lock manager
  - Token manager
  - Distributed CS managers

Update Propagation

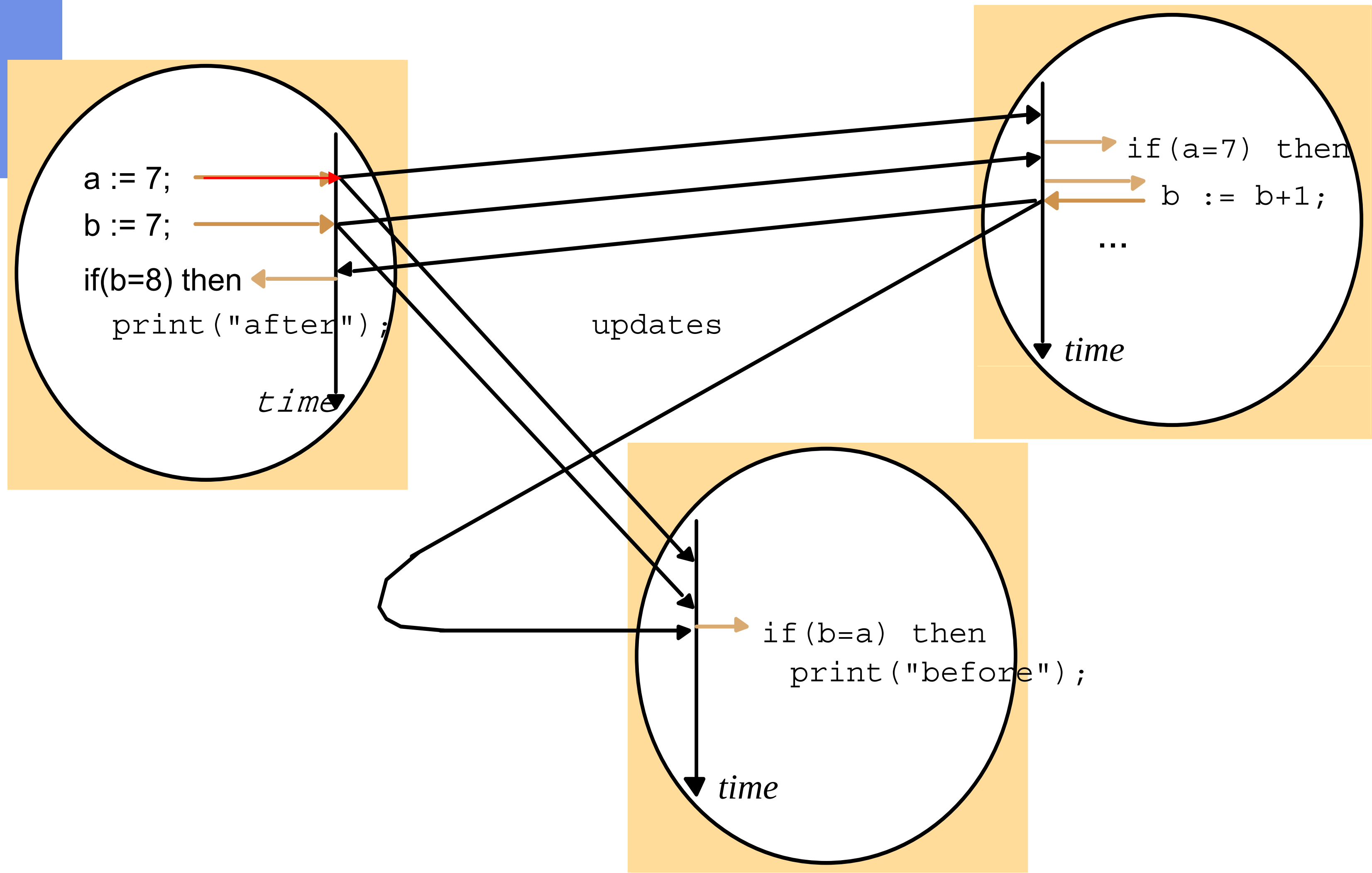
Write-Update

Write-Invalidate

Write-Update

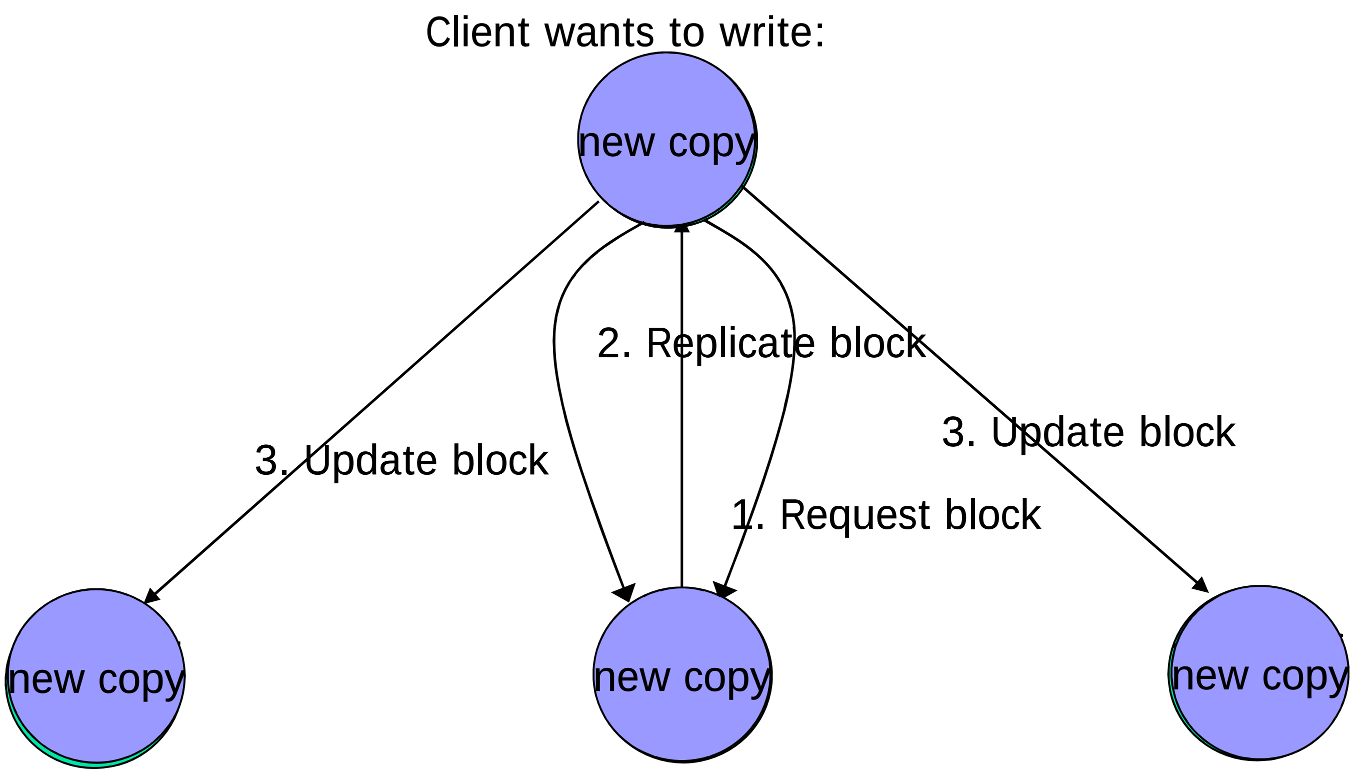
- Suppose a process has write permission for a page. It updates a “data item” on it locally
- Updates are propagated via multicast to all replicas that currently have a copy of this “data item”, i.e. the page
- Replica-managers update the corresponding data items in order to allow as consistent reads as possible
- In practice you try to allow multiple writes to a page in a row by the same process, otherwise too much overhead
- Furthermore, if possible you just propagate the updates differences of the page to the other replicas

Example Write-Update



Implementing Sequential Consistency

Write Update

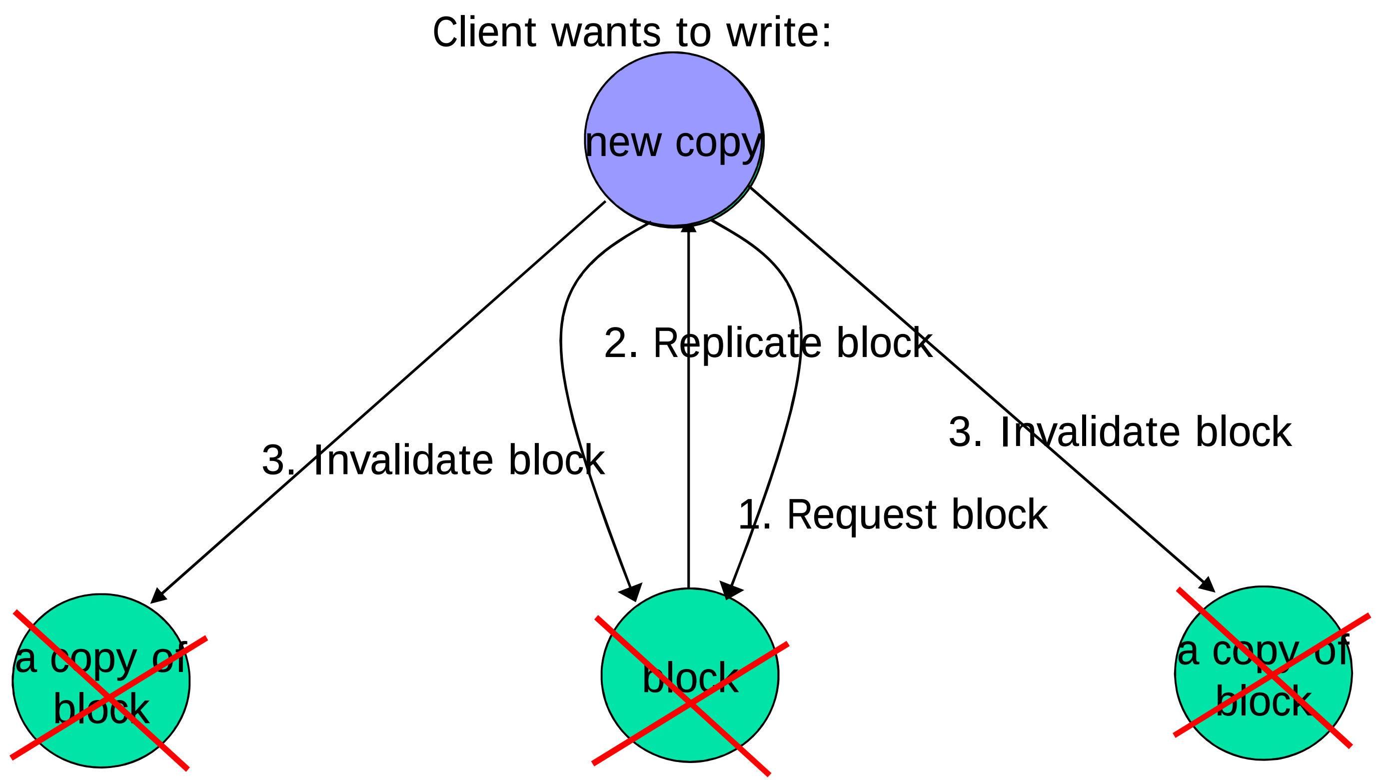


Write-Invalidate

- Before writing to a data item the process multicast an invalidation message to all replicas that currently host that data item, announcing the upcoming update
- As long as the process is writing, all other processes accessing that data item, will be “blocked”
- Updates are sent whenever a process wants to read a data item that had been invalidated in the past
- Reading valid local data items occurs with no delay

Implementing Sequential Consistency

Write Invalidation



Implement “Consistent” DSM

Single Copy DSM

Multiple Copy DSM

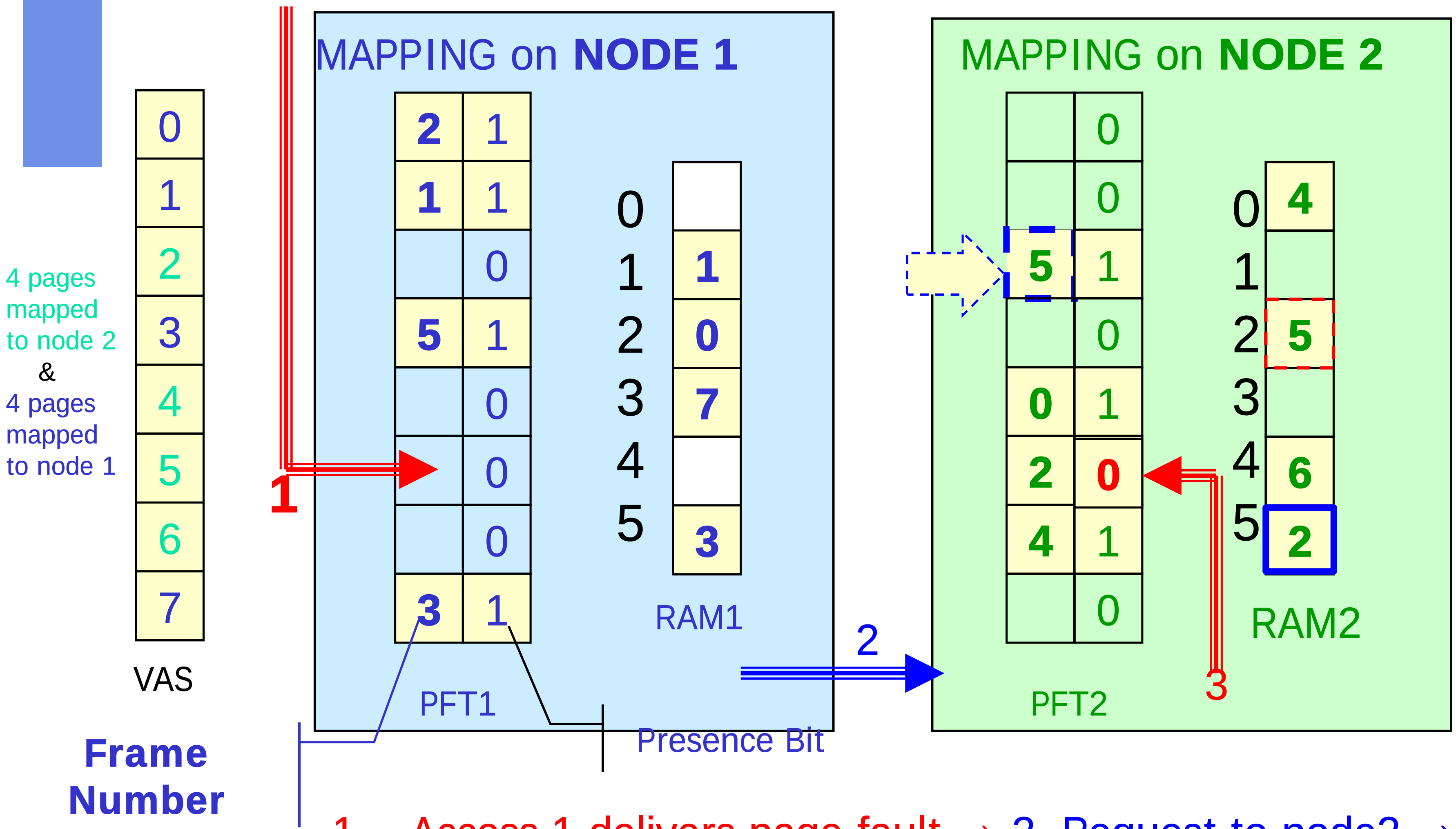
Implement a Single Copy DSM

- Page based virtual memory management
- MMU with page-based address transformation
- Shared memory segment(s), i.e. its(their) virtual address range(s) can be mapped at different nodes
- Page is never mapped to more than one node

Implement a Single Copy DSM

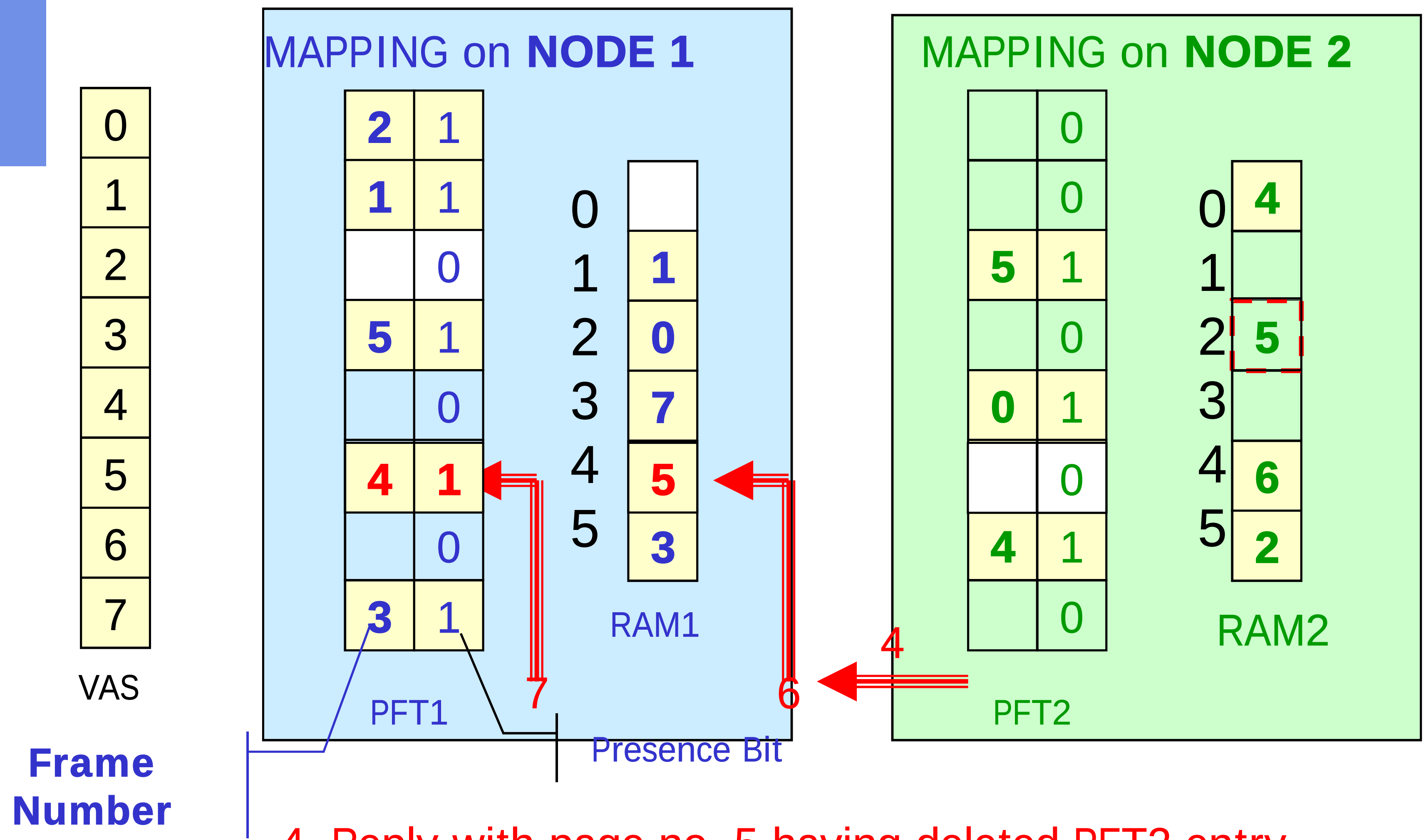
- Local access:
  - ∃ mapped page with presence bit = set in the corresponding local page-frame table (PFT)
  - Perform read/write accesses only to local RAM
- Remote access:
  - Presence bit in local PFT is empty
  - Remote access ⇒ page fault
    - Pager gets page from remote node
    - Set presence bit
  - Repeat memory access
- DSM is coherent if
  - Page transfer operations are atomic
  - No node crashed occur

Simple Map Protocol in a DSM



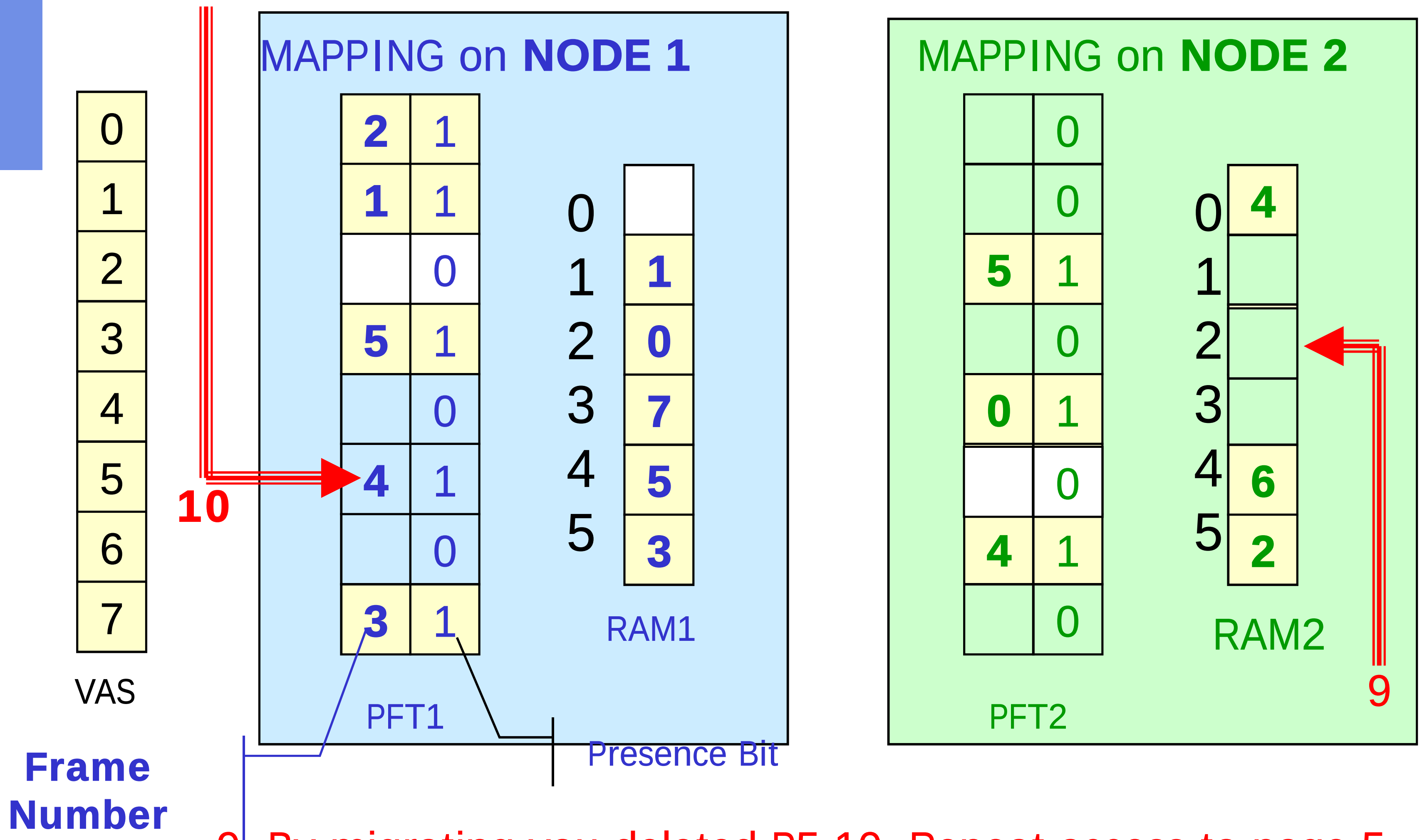
1. Access 1 delivers page fault ⇒ 2. Request to node2 ⇒
3. Delete presence bit of page 5 in node 2

Mapping Protocol in a DSM



4. Reply with page no. 5 having deleted PFT2 entry
6. Migrate that page into RAM1 ⇒ 7. Map into PFT1,

Mapping Protocol in a DSM



9. By migrating you deleted P5 10. Repeat access to page 5
11. Page fault commit message to node 2

Summary Single Copy DSM

- Linear consistency if we use a central coordinator sequencing all accesses, however poor performance
- No concurrent read access to the same page
  - Ping-Pong paging between nodes occurs too often, e.g. especially if code of different threads is located on the same page in case of a distributed task
- False Sharing
  - 2 data objects in the same page used by different activities
  - Mutual page stealing when threads write to that page
- *What can we do?*
  - Reducing the consistency requirements
  - Implement multiple copy DSM to support concurrent reads

Multi Copy DSM

Linear Consistency?

- Each read should deliver the value of the latest write operation
- <u>Problem</u>
  - No synchronized exact global time
    - No unambiguous sequence, if clock synchronization is too coarse, but we can achieve linear consistency
  - We must resolve read/write and write/write conflicts between concurrent processes
    - A memory access is far shorter than the minimal time deviation

~ Strict Consistent DSM?

- Use a Single Copy DSM (see slides before)
  - Whenever a page is accessed, it first must migrate to the accessing node, but there are no read/write or write/write conflicts
  - However, many additional page migrations, e.g. with concurrent reads from the same page
- *How to know where a page is currently located?*
- Every node must know the current location of each mapped page or you use a central super pager
  - Use a shadow page table
  - Whenever the mapping of a page changes, you have to change the corresponding page tables at all involved nodes

~ Strict Consistent DSM?

- *Where to migrate a page when there are concurrent accesses?*
  - Need a consensus on the sequence of operations (easy with a central coordinator, otherwise additional overhead)
  - Real parallel operations only on different pages
  - No distinction between read/write(RW) & read-only(RO) pages, no support for concurrent reads from different RO-pages

Multi Copy DSM

- Assume: Non modifying code $\Rightarrow$
- Code pages are similar to Read-Only Pages, i.e. their content will never change
- Once copied to the needed location, they can stay their until application has finished without any additional overhead
- Changes might only happen when a thread migrates to another location

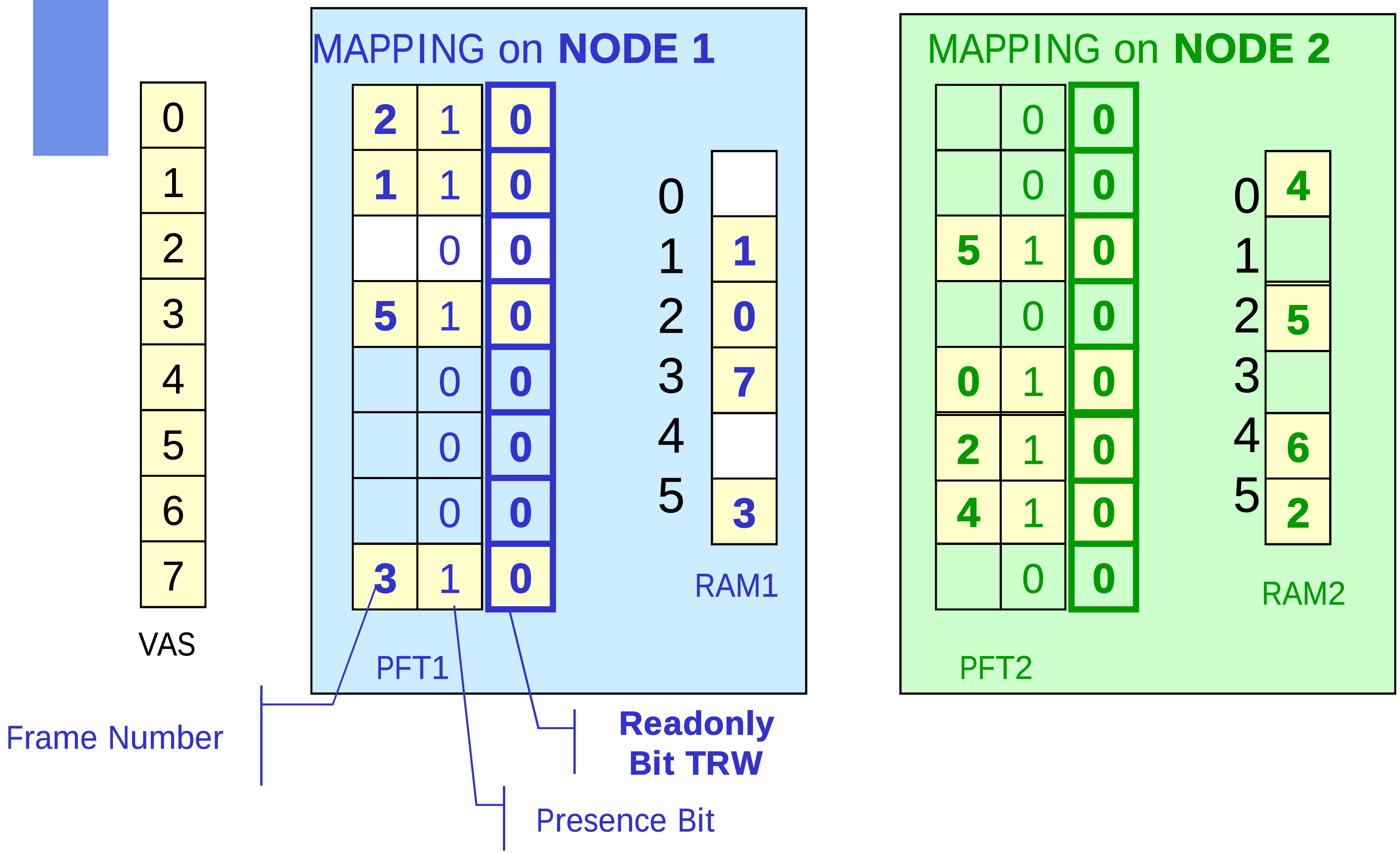
Multi Copy DSM

- In the following we focus on potentially shared read-write data-pages
  - To distinguish between READ-ONLY and READ-WRITE pages there is a permanent control-bit PRW per page
  - If PRW-Bit ==1, a write to a READ-ONLY page will cause an exception of type: address violation
- The very first time, a potentially READ-WRITE page is mapped, it is initiated with a „temporal“ control-bit TRW = 1, indicating that at the node where this page is mapped, each process/KLT can write to this page
- TRW == 0 means, that at the involved node no write access to P is temporarily allowed

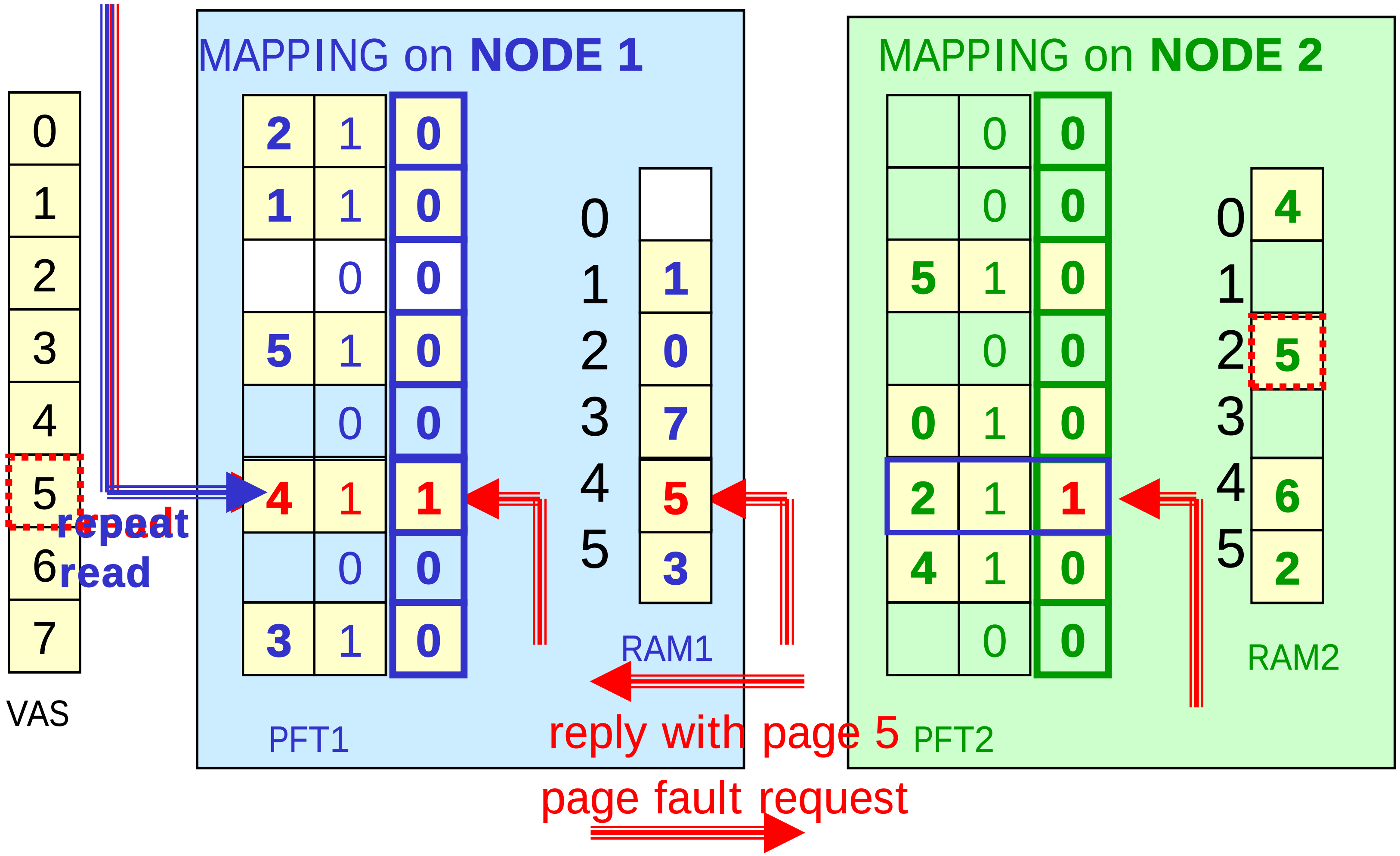
Multi Copy Consistent DSM

- A remote read (to a non local page) → page fault
  - **Copy** page from current page owner, i.e. don't delete page at owner's RAM
  - Prevent writes on both sides, set TRW=0, i.e. any new write → page-fault exception
- Whenever some node L_j tries to write to page P
  - **Copy** P from the replica with TRW == 1, which must be the one with the most recent writes
    - Invalidate all replicas, i.e. delete their PFT entries & empty the corresponding mapped page-frame.
  - If there is no such replica with TRW==1 all replicas (also your local one) are identical and up to date
  - Set local TRW = 1 and repeat your write operation at L_j

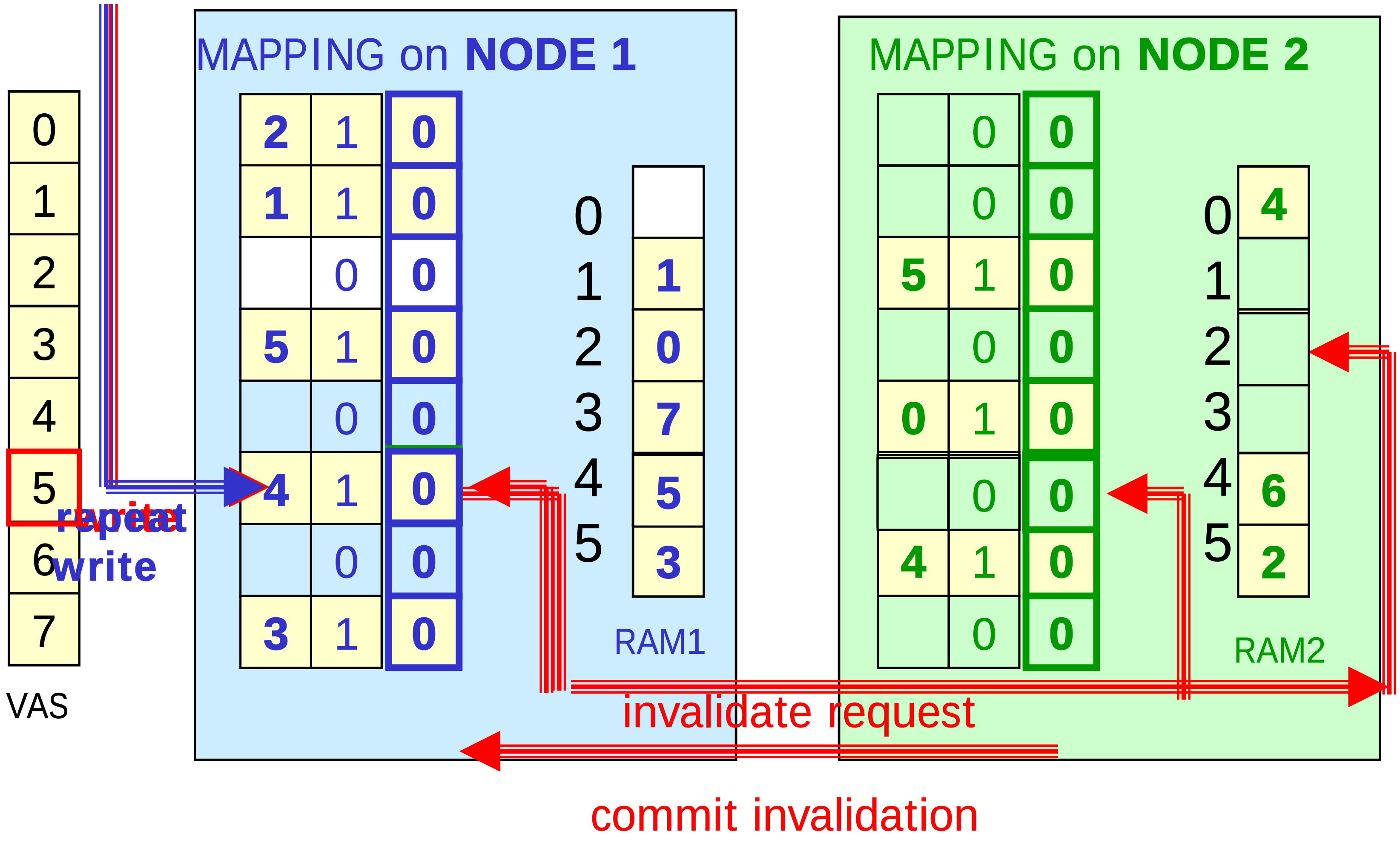
Multi Copy Consistent DSM



Implement Consistent DSM



Implement Consistent DSM



Analysis of Linear Consistent DSM

- *How did we achieve ~strict consistency?*
- Only one process/node can write to the same page at the same instant of time
- Only works efficiently when writes are rare and multiple writes at one side are collected, otherwise ping-pong paging

Sequential Consistent DSM

- Problem
  - No linkage between real time and operations
  - Find some sequential ordering of the operations on all nodes
    - Ordering might conflict with application, e.g. a process awaiting a certain value of a coordination variable

- Implementation
  - Write operations have to be visible in all processes in the same order
  - Duration of a write
  - Example: Forge the latest write in the next write
  - Add an owner flag per PFT entry

Implement Sequential Consistent DSM

MAPPING on **NODE 1**

VAS / PFT1

VAS	Frame Number	Presence Bit	Readonly Bit	Owner Bit
0	**2**	1	0	**1**
1	**1**	1	0	**1**
2		0	0	**0**
3	**5**	1	0	**1**
4		0	0	**0**
5		0	0	**0**
6		0	0	**0**
7	**3**	1	0	**1**

RAM1

0	
1	**1**
2	**0**
3	**7**
4	
5	**3**

MAPPING on **NODE 2**

PFT2

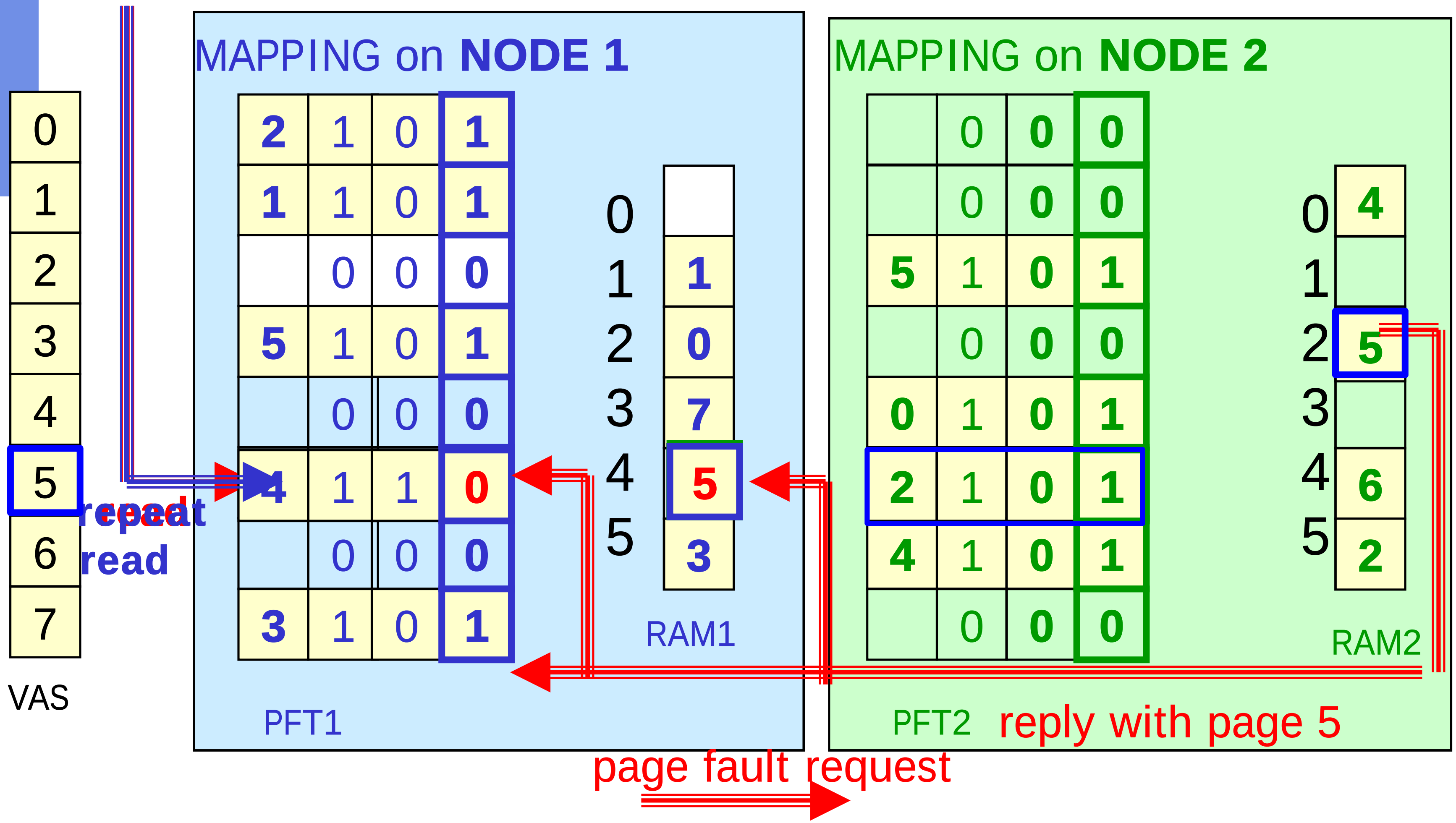
VAS	Frame Number	Presence Bit	Readonly Bit	Owner Bit
0		0	**0**	**0**
1		0	**0**	**0**
2	**5**	1	**0**	**1**
3		0	**0**	**0**
4	**0**	1	**0**	**1**
5	**2**	1	**0**	**1**
6	**4**	1	**0**	**1**
7		0	**0**	**0**

RAM2

0	**4**
1	
2	**5**
3	
4	**6**
5	**2**

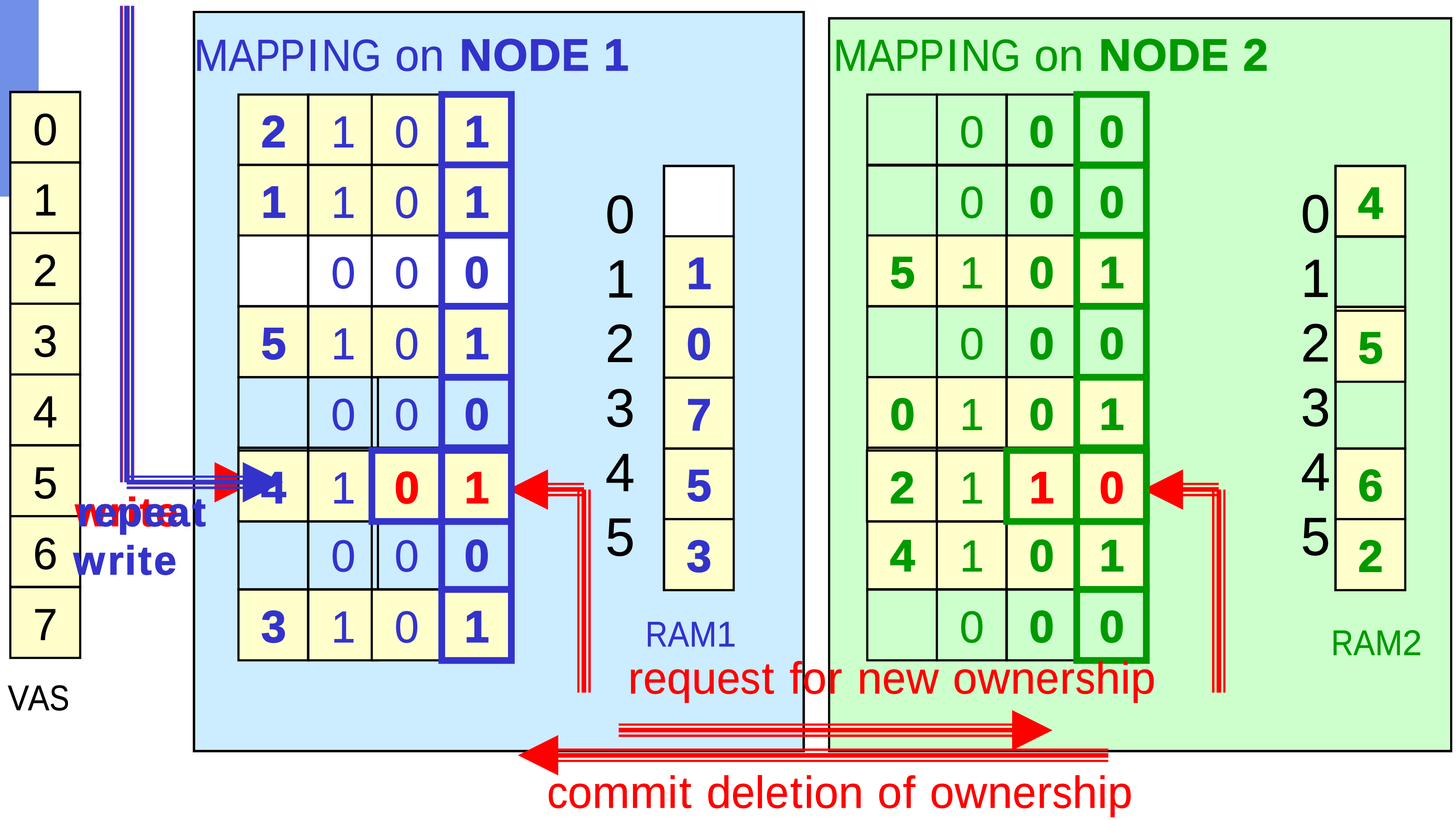
- each node = owner of its 4 present pages

Implement Sequential Consistent DSM



- owner dos not change its PFT
- new replica is not owner, i.e. its owner bit is not set

Implement Sequential Consistent DSM



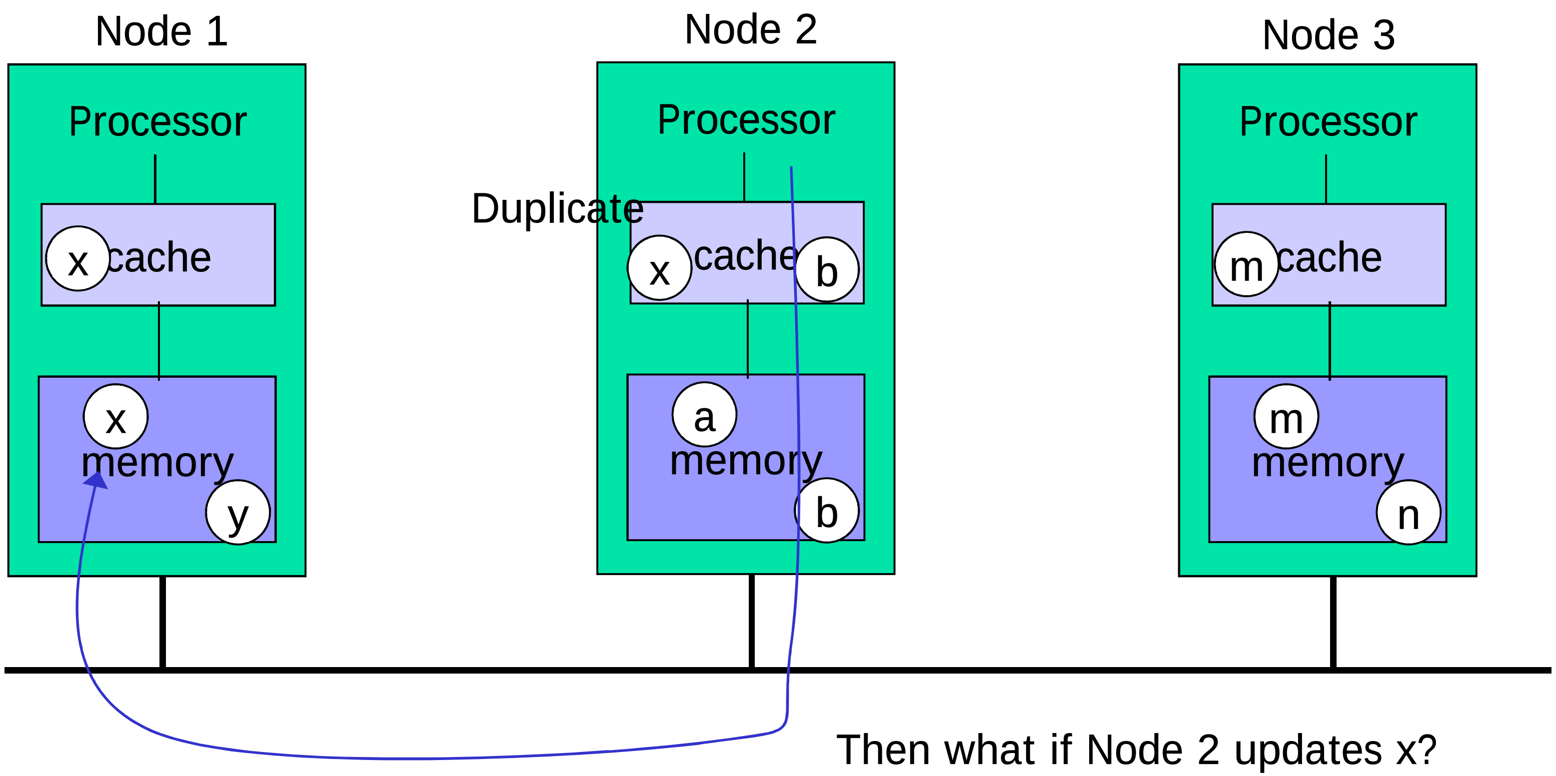
- Change old owners read only bit and owner flag
- Give ownership to writer

Drawbacks Sequential Consistent DSM

- *What to do when 2 concurrent writes are initiated?*
  - See assignments
- Overhead per access still expensive
  - Per first read you must copy remote page to target
  - Per write on a not-owner node you must delete the ownership of the owner node and shift it to the writer node, having copied the page before
- *How to propagate the updates of the owner's side to all other outdated copies?*
- *How to prevent from reading staled data?*

Implementing Sequential Consistency

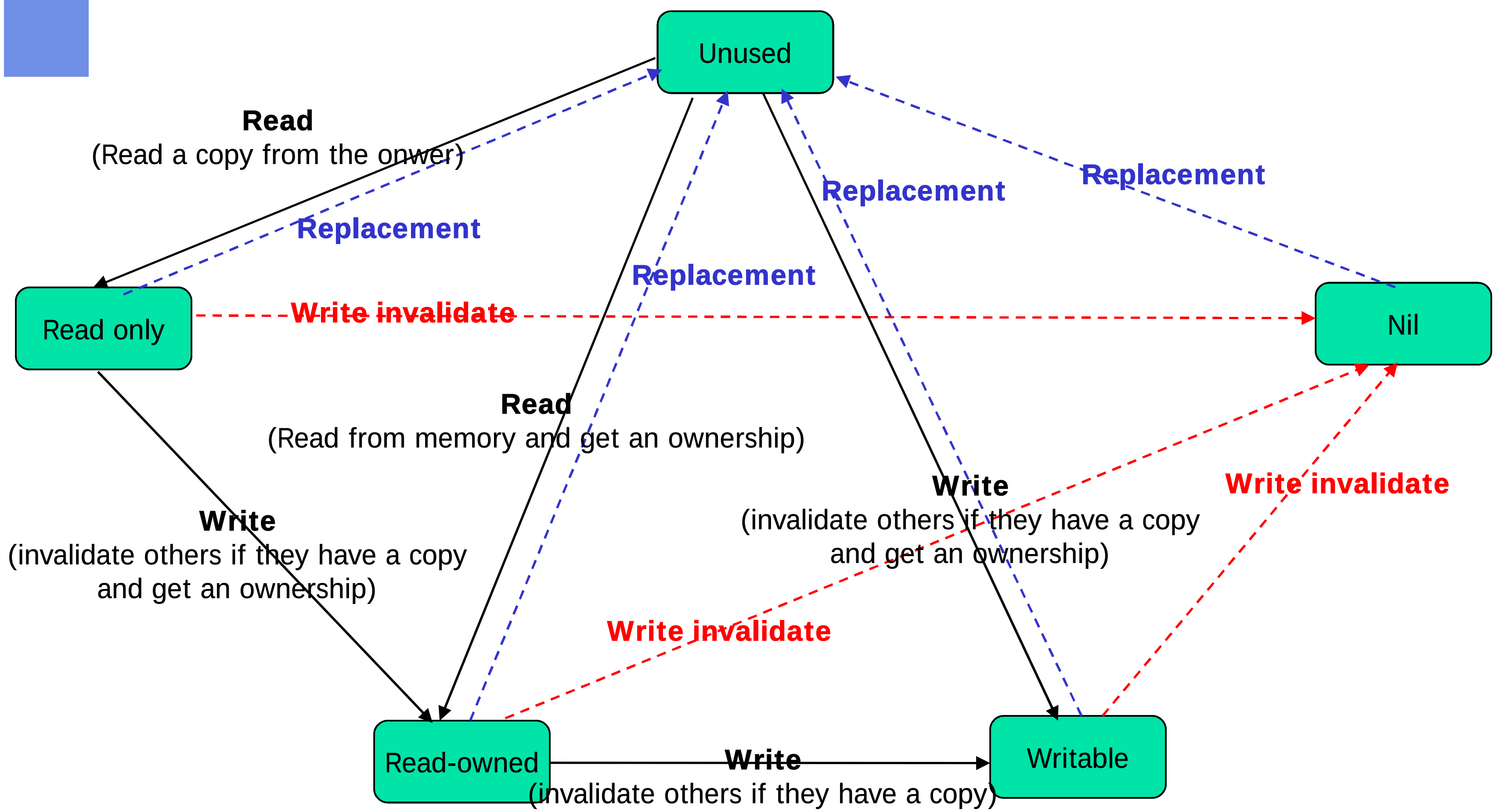
Replicated and Migrating Data Blocks



Then what if Node 2 updates x?

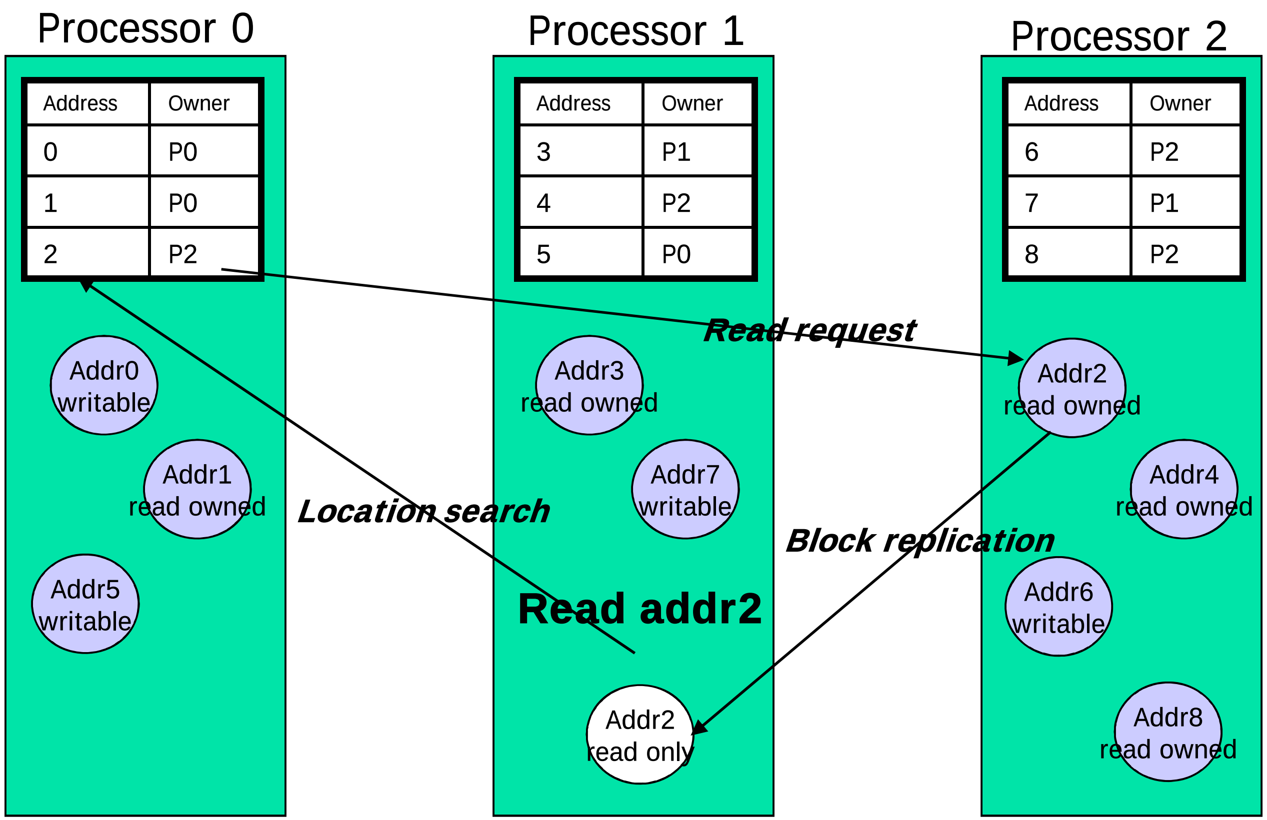
Implementing Sequential Consistency

Read/Write Request



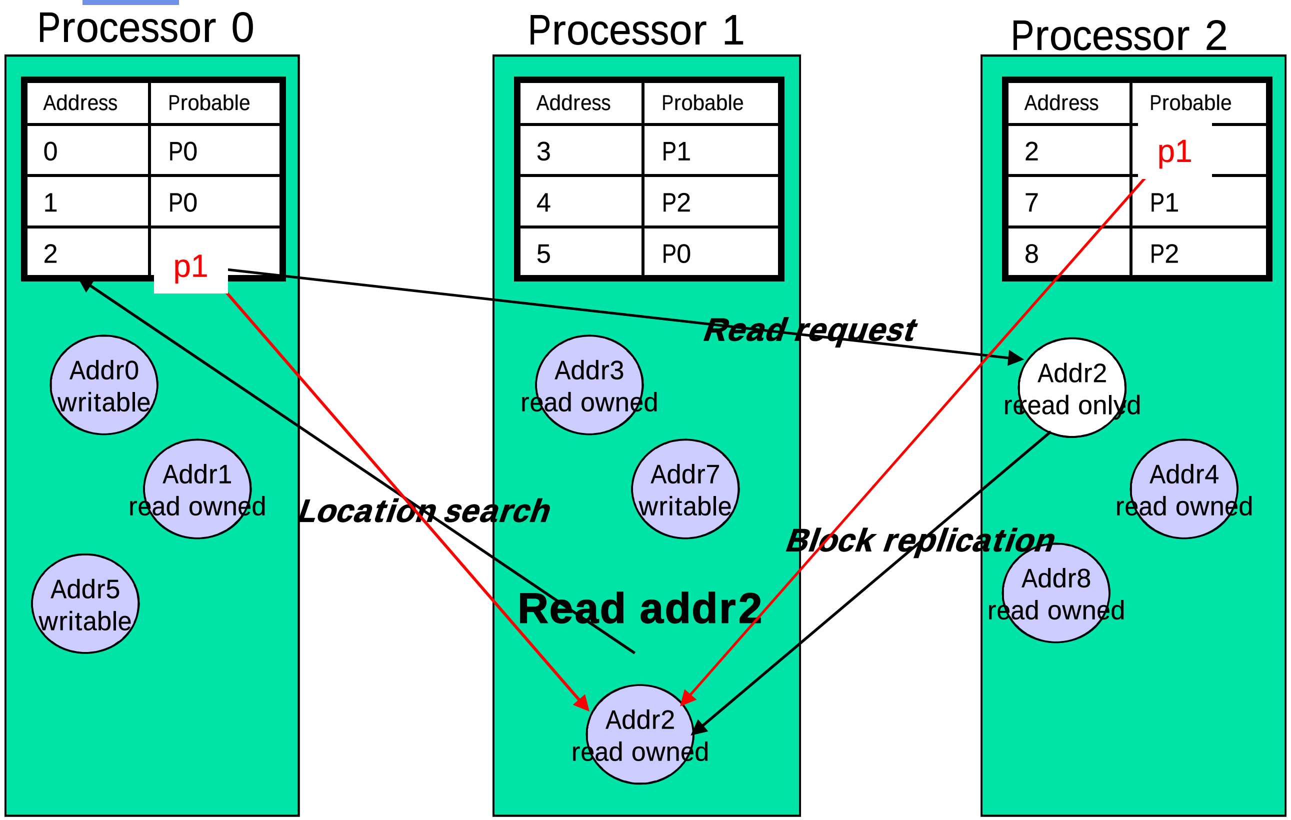
Implementing Sequential Consistency

Locating Data –Fixed Distributed-Server Algorithms



Implementing Sequential Consistency

Locating Data – Dynamic Distributed-Server Algorithms



- Breaking the chain of nodes:
  - When the node receives an invalidation
  - When the node relinquishes ownership
  - When the node forwards a fault request
- The node points to a new owner

Replacement Strategy

- Which block to replace
  - Non-usage based (e.g. FIFO)
  - Usage based (e.g. LRU)
  - Mixed of those (e.g. Ivy)
    - Unused/Nil: replaced with the highest priority
    - Read-only: the second priority
    - Read-owned: the third priority
    - Writable: the lowest priority and LRU used.
- Where to place a replaced block
  - Invalidating a block if other nodes have a copy.
  - Using secondary store
  - Using the memory space of other nodes

Thrashing

- Thrashing:
  - Two or more processes try to write the same shared block.
  - An owner keeps writing its block shared by two or more reader processes.
  - The larger a block, the more chances of false sharing that causes thrashing.
- Solutions:
  - Allow a process to prevent a block from being accessed by other processes, using a lock.
  - Allow a process to hold a block for a certain amount of time.
  - Apply a different coherence algorithm to each block.
- *What do those solutions require users to do?*
- *Are there any perfect solutions?*

Literature

- B. Bershad et al: “The Midway DSM System, IEEE 1993
- N. Carreiro, D. Gelernter: “The S/Net’s Linda Kernel”, ACM Trans. On Comp. Sys., 1986
- J. Cordsen: “Virtueller gemeinsamer Speicher”, PhD TU Berlin, 1996
- M. Dubois et al.: “Synchronization, Coherence and Event Ordering in Multiprocessors”, IEEE Computer, 1988
- K. Li: “Shared Virtual Memory on Loosley Coupled Multiprocessors”, PhD Yale, 1986
- D. Mosberger: “Memory Consistency Models”, Tech. Report, Uni. Of Arizona, 1993
- B. Nitzberg: “DSM: A Survey of Issues and Algorithms”, IEEE Comp. Magazine, 1993

Appendix: Review Consistency Models

Another Notation

See Colouris et al

Processes Accessing Shared Data

Process 1

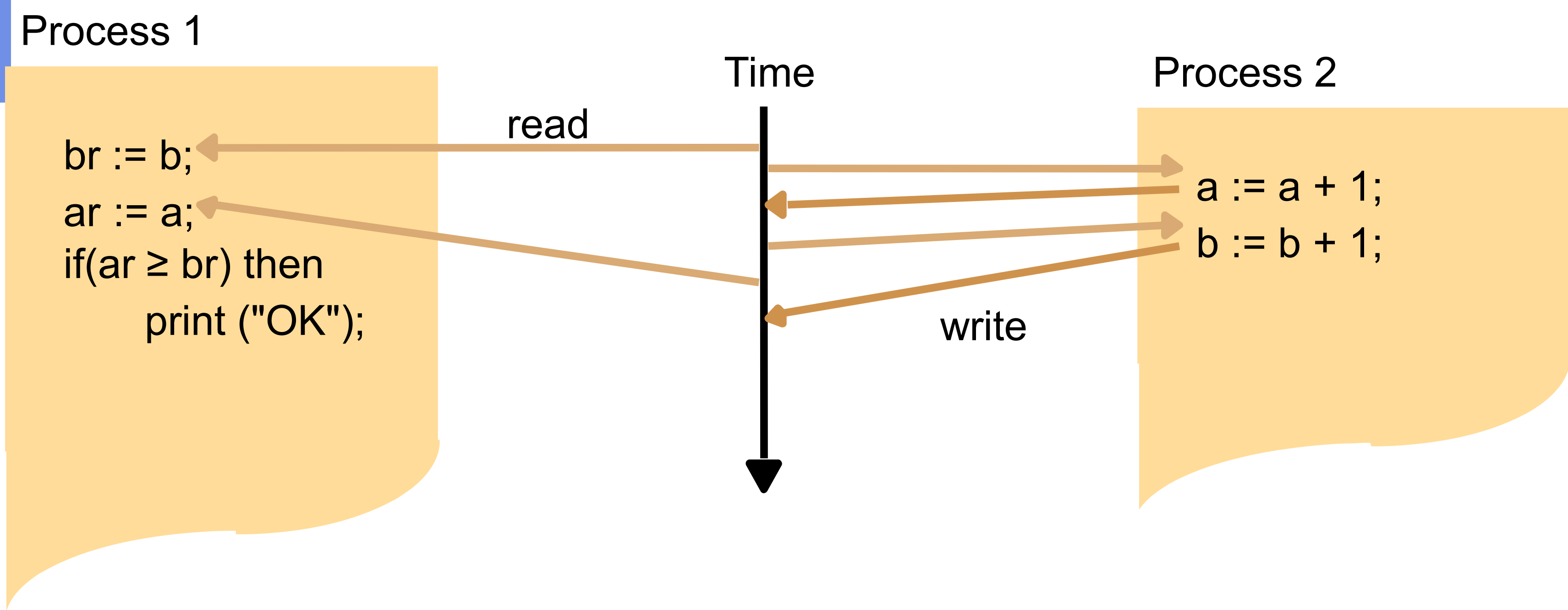
```
br := b;
ar := a;
if(ar ≥ br) then
      print ("OK");
```

Process 2

```
a := a + 1;
b := b + 1;
```

- a & b are initialized with 0
- Suppose, process 2 runs first, then process 1
- We expect that process1 always prints OK
- However, the update propagation of the DSM might send the updates to process1 in reverse order, i.e. ar = k, but br =k+1

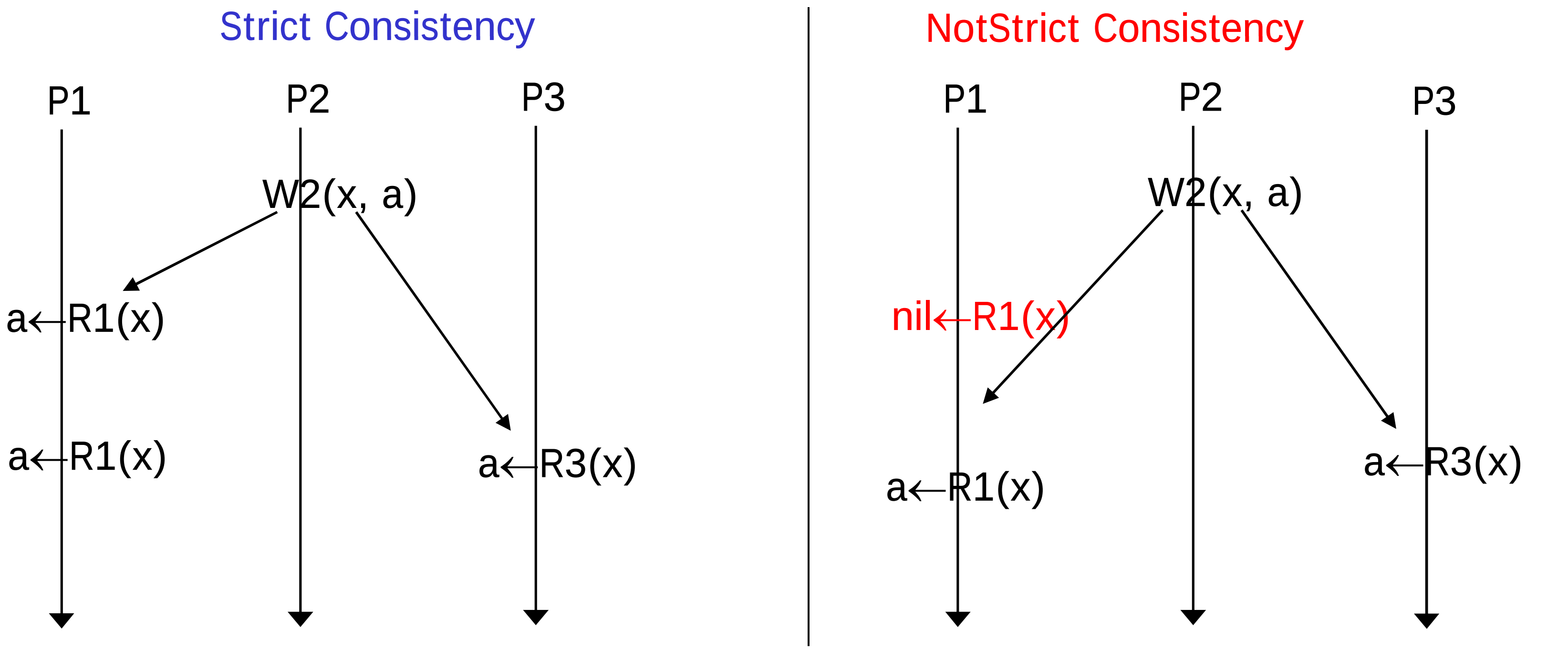
Interleaved Operations



- Allowed interleaving with sequential consistency

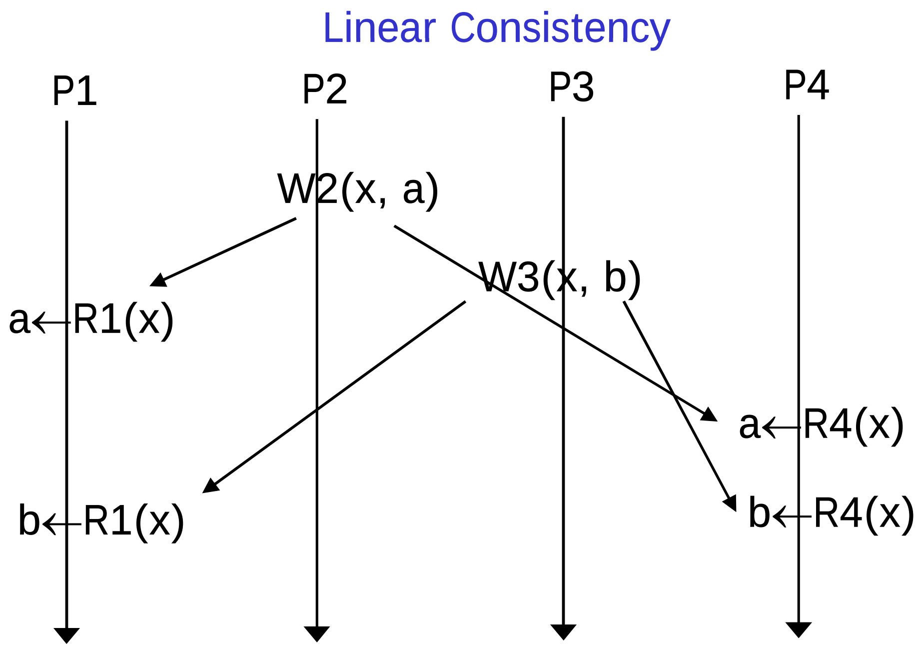
Strict Consistency

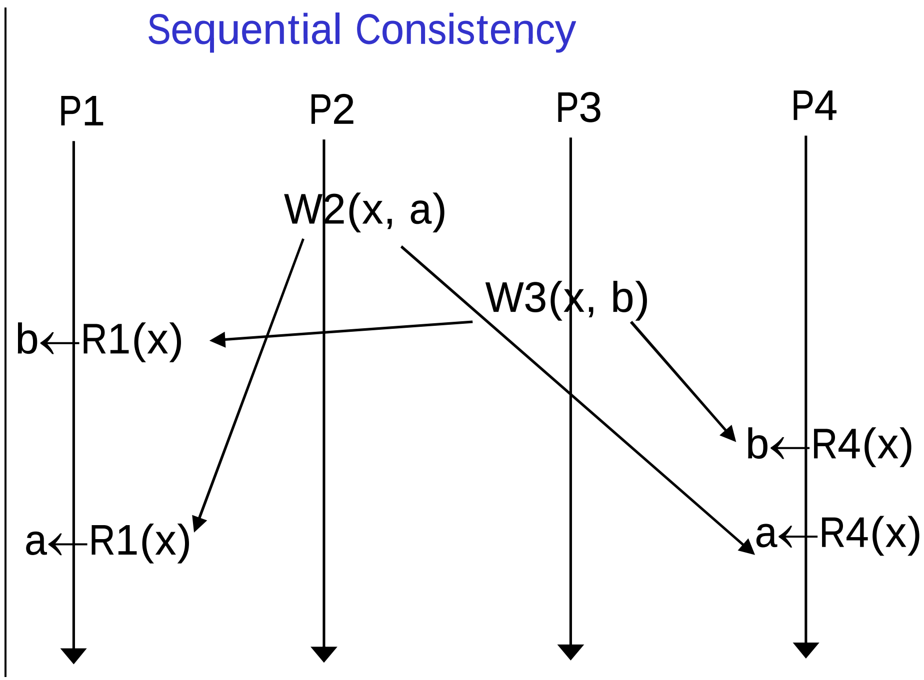
- Wi(x, a): Processor i writes a on variable x.
- b←Ri(x): Processor i reads b from variable x.
- Any read on x must return the value of the most recent write on x.



Linear & Sequential Consistency

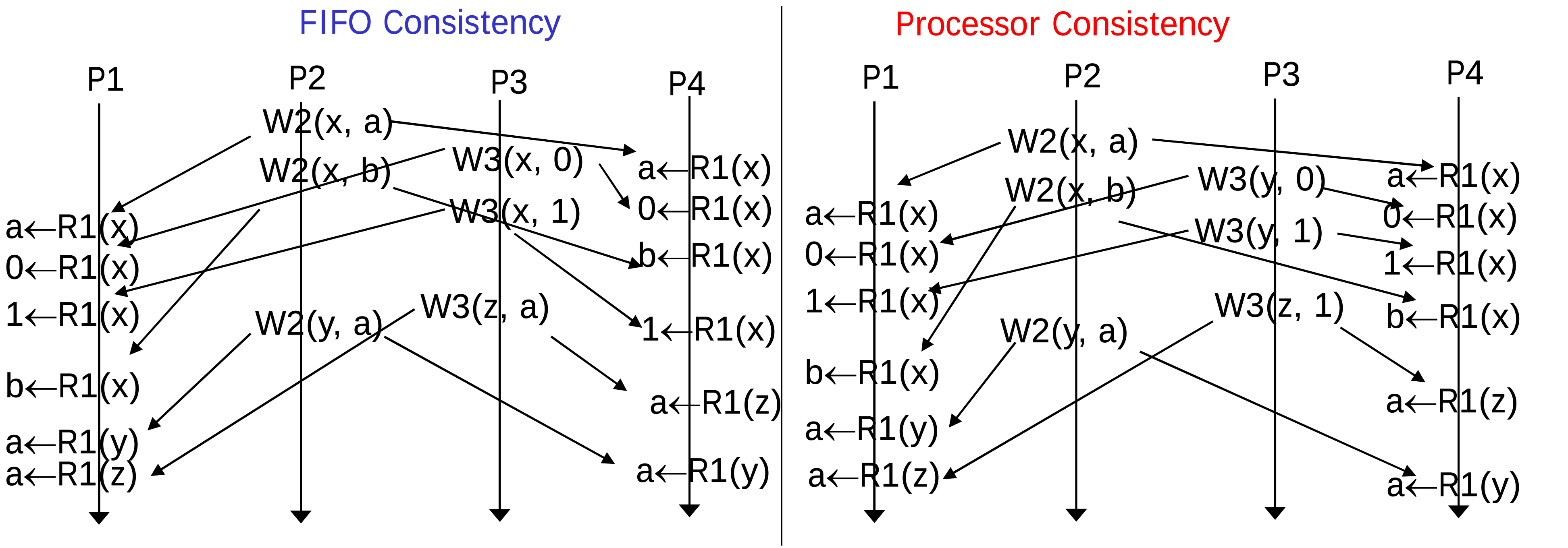
- **Linear Consistency:** Operations of each individual process appear to all processes in the same order as they happen.
- **Sequential Consistency:** Operations of each individual process appear in the same order to all processes.





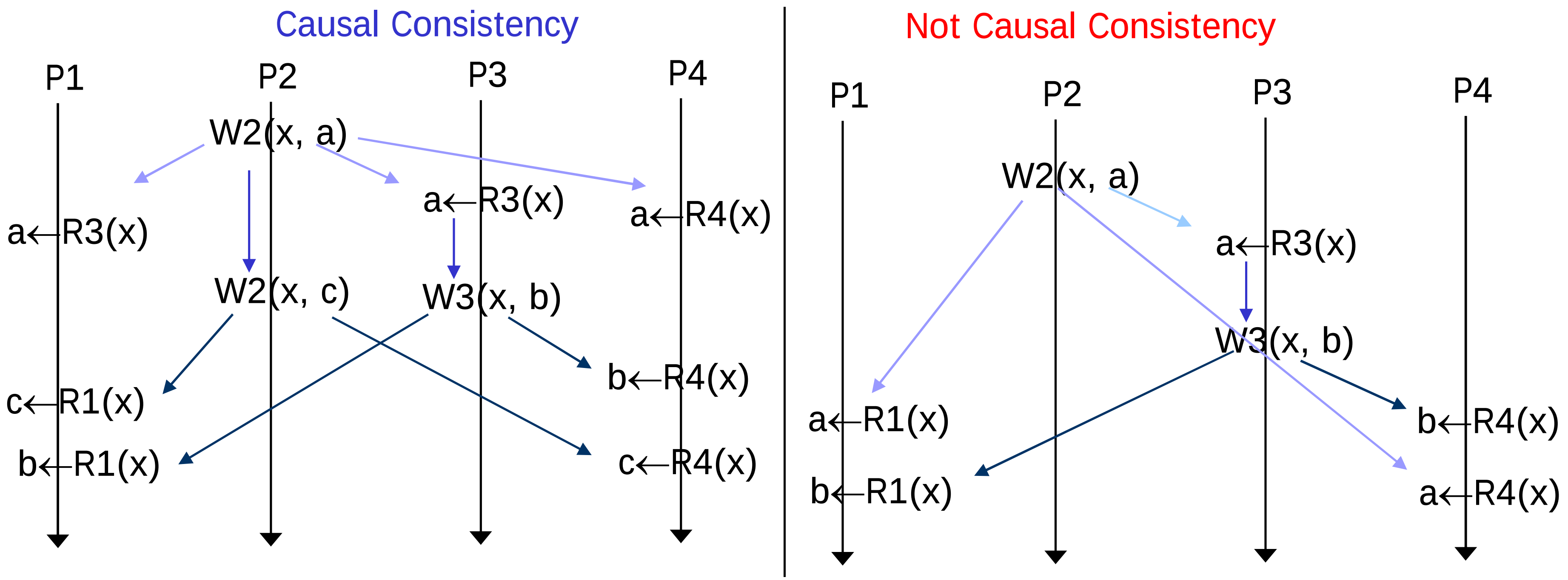
FIFO and Processor Consistency

- **FIFO Consistency:** writes by a single process are visible to all other processes in the order in which they were issued.
- **Processor Consistency:** FIFO Consistency + all write to the same memory location must be visible in the same order.



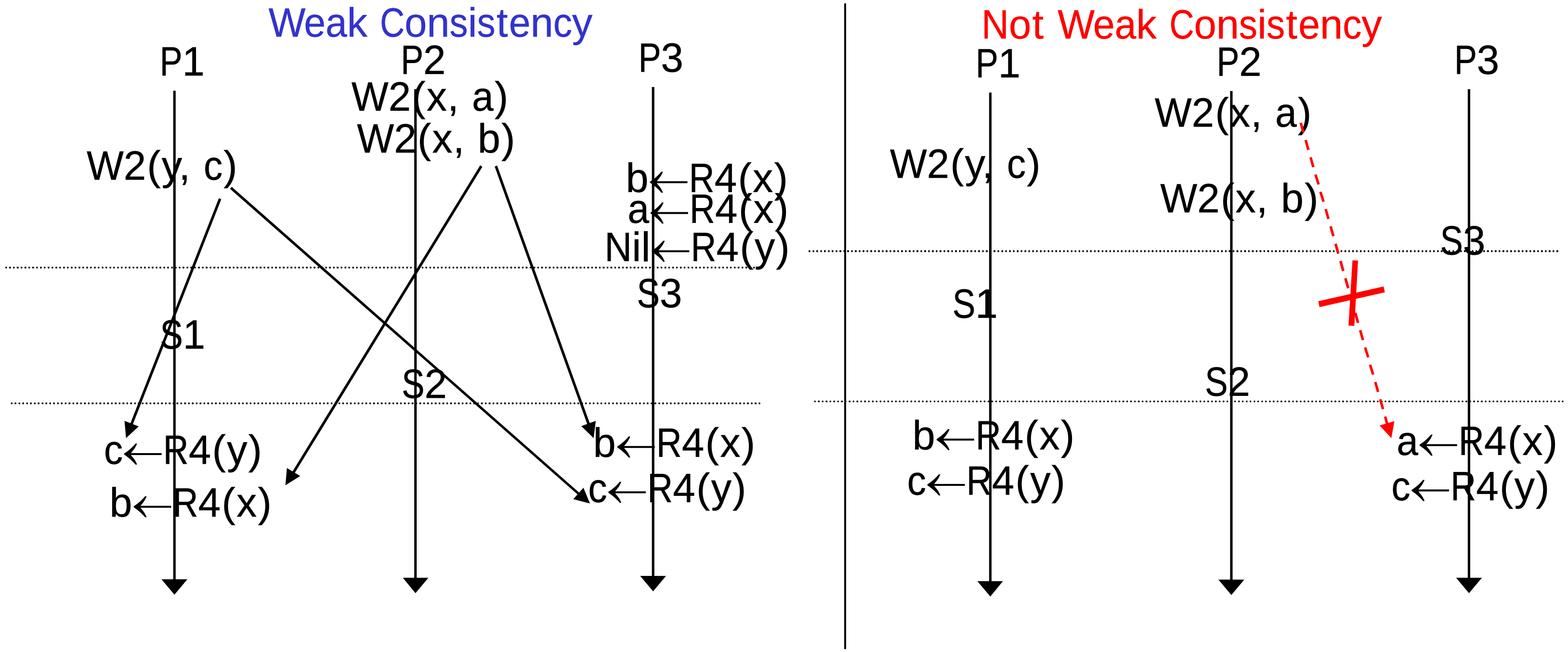
Causal Consistency

- Causally related writes must be visible to all processes in the same order. Concurrent writes may be propagated in a different order.



Weak Consistency

- Accesses to synchronization variables must obey sequential consistency.
- All previous writes must be completed before an access to a synchronization variable.
- All previous accesses to synchronization variables must be completed before access to non-synchronization variable.



Release Consistency

- Access to acquire and release variables obey processor consistency.
- Previous acquires requested by a process must be completed before the process performs a data access.
- All previous data accesses performed by a process must be completed before the process performs a release.