COS3043
System Fundamentals

Lecture 8

Topics

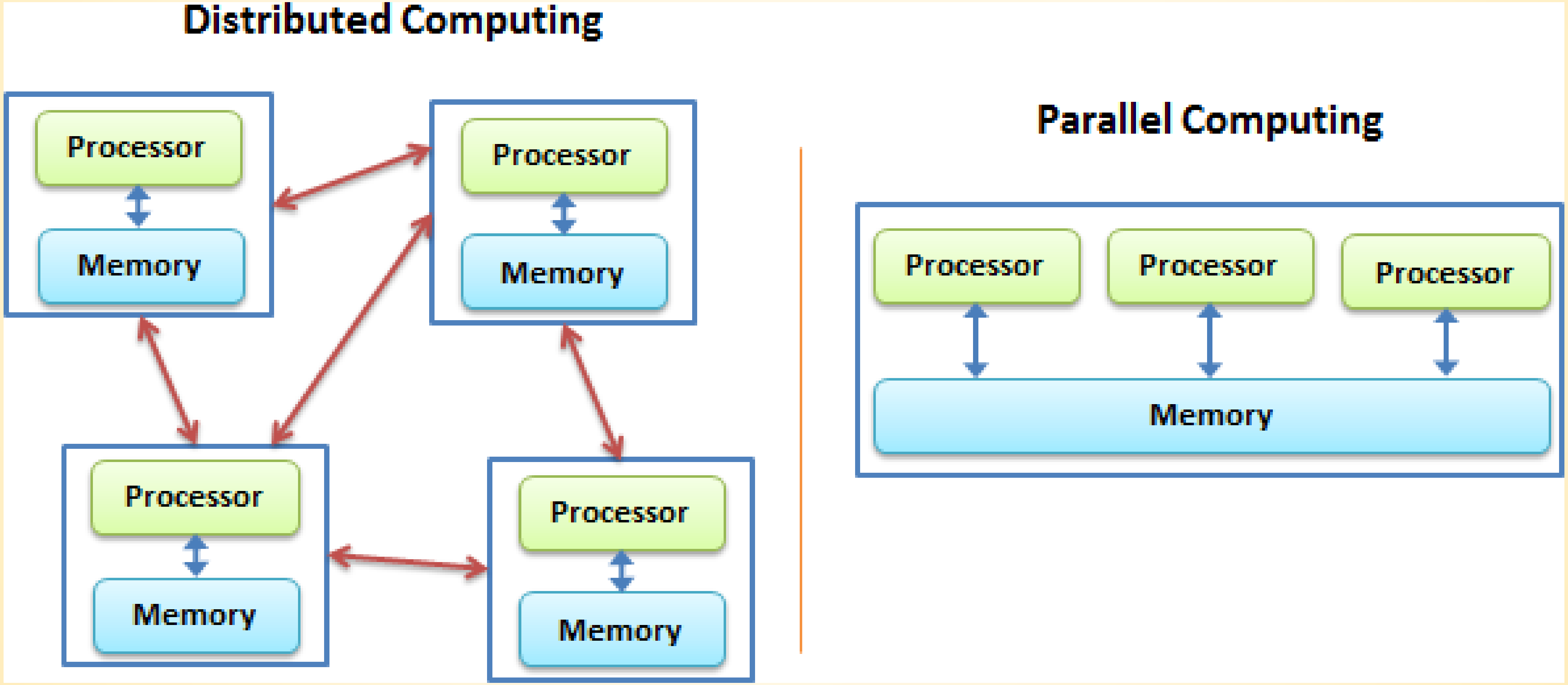
1.	Abstractions 1.1 Hardware Resources 1.2 OS Functionality 1.3 Managing the CPU and Memory
2.	OS Structure 2.1 SPIN Approach 2.2 Exokernel Approach 2.3 L3/L4 Micro-Kernel Approach
3.	Virtualization 3.1 Intro to Virtualization 3.2 Memory Virtualization 3.3 CPU and Device Virtualization
4.	Parallelism 4.1 Shared Memory Machines 4.2 Synchronization 4.3 Communication 4.4 Scheduling
5.	Distributed Systems 5.1 Definitions 5.2 Lamport Clocks 5.3 Latency Limit

6.	Distributed Object Technology 6.1 Spring Operating System 6.2 Java RMI 6.3 Enterprise Java Beans
7.	Design and Implementation of Distributed Services 7.1 Global Memory System 7.2 Distributed Shared Memory 7.3 Distributed File System
8.	System Recovery 8.1 Lightweight Recoverable Virtual Memory 8.2 Rio Vista 8.3 Quicksilver
9.	Internet Scale Computing 9.1 GiantScale Services 9.2 Content Delivery Networks 9.3 MapReduce
10.	Real-Time and Multimedia 10.1 Persistent Temporal Streams

List of Discussion

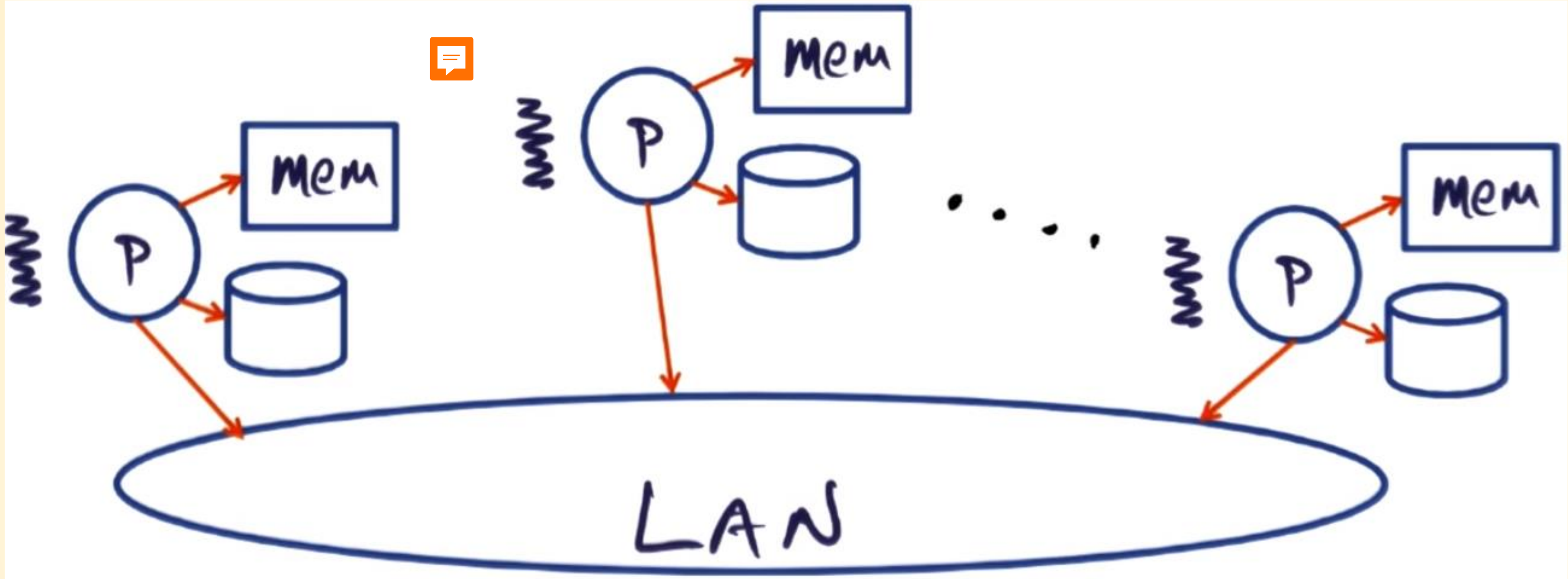
- Global Memory Systems
- Distributed Shared Memory
- Distributed File Systems

Distributed Systems Versus Parallel Systems (Let's Recall This Again)



Global Memory Systems

Context for Global Memory Systems



- Memory Pressure:
  - ➢ Different for each node
  - ➢ How to use idle cluster memory?
  - ➢ Remote memory access faster than disk.

- Normal OS:

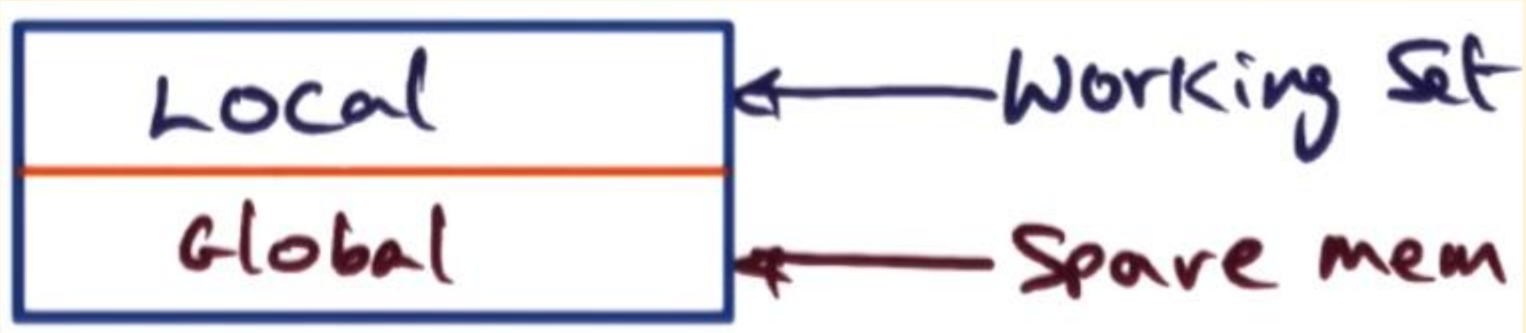
  VA => PA or Disk

- GMS:

  VA => PA or Cluster Memory
  or Disk

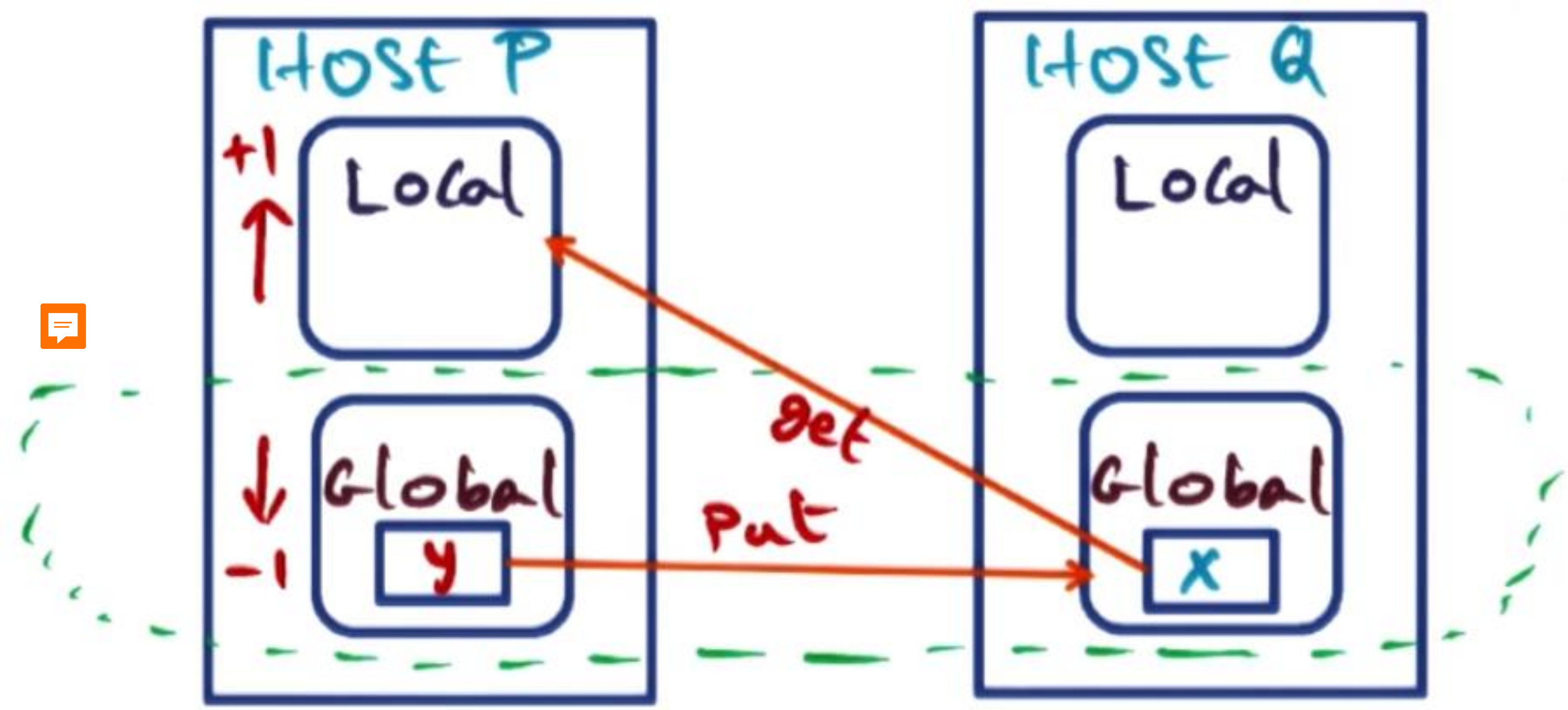
GMS Basics

- "Cache" refers to physical memory (i.e. DRAM) not processor cache.
- Sense of "community" to handle page faults at a node.
- The main purpose of GMS is for managing page faults.
- Physical memory at a node:



Handling Page Faults – Case 1

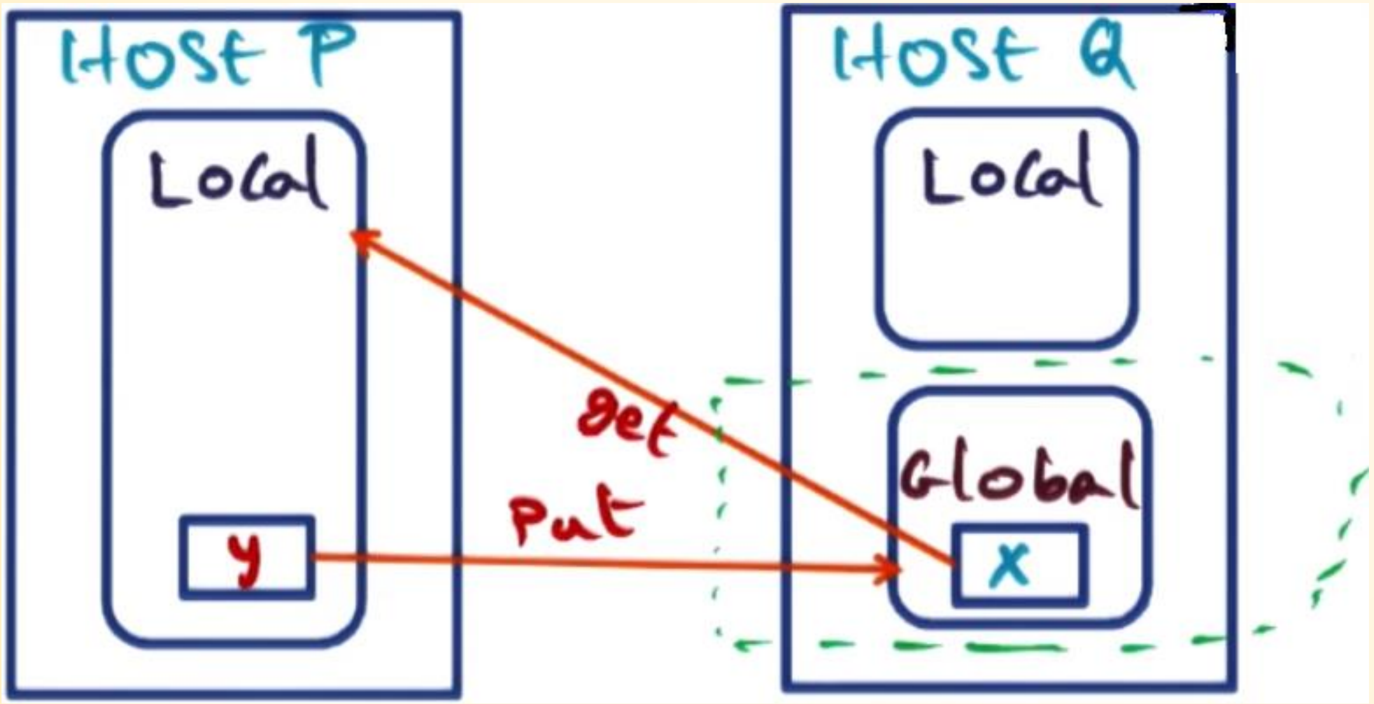
- Page fault for x at node P.
- Hit in global cache of node Q.
- y is the oldest page on P.



Common case

Handling Page Faults – Case 2

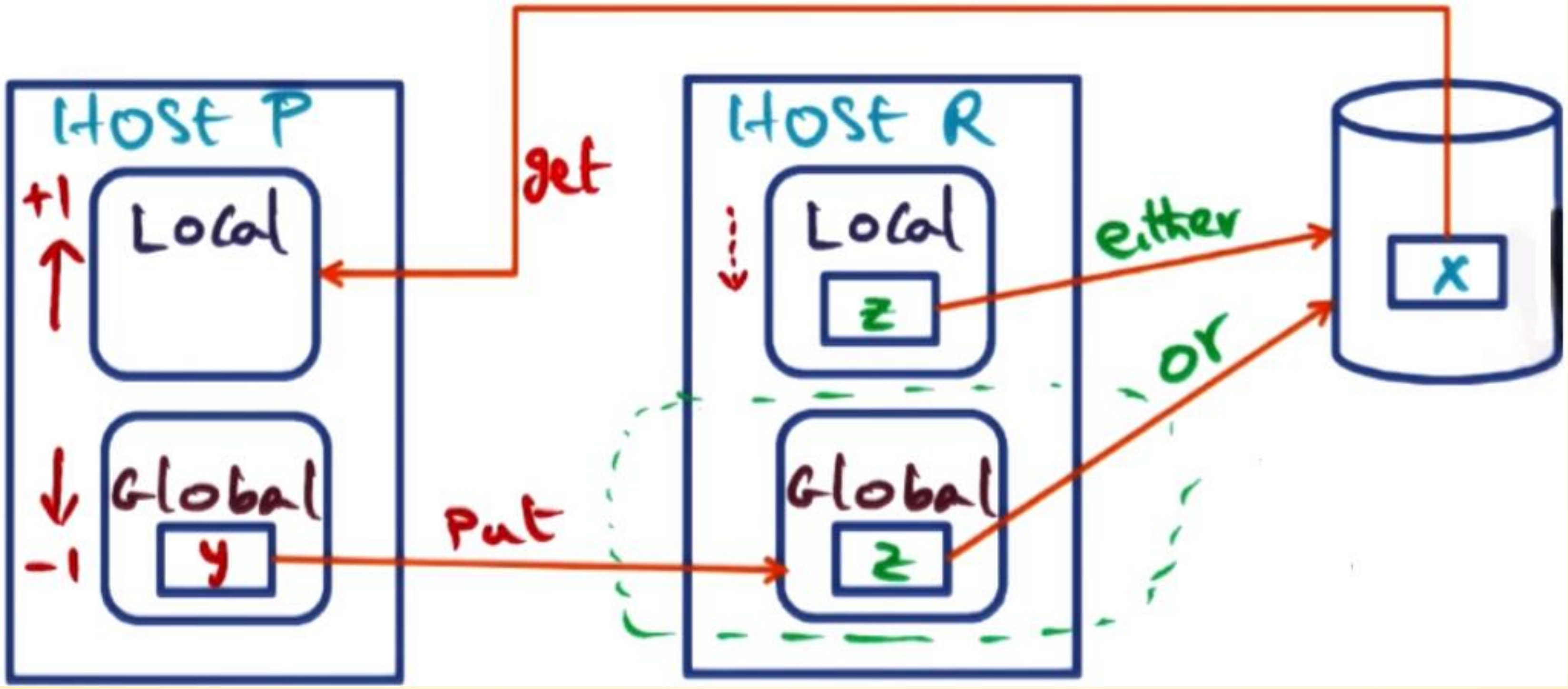
- Page fault for x at node P.
- Swap LRU page y for x
- LRU = least recently used



Common case with memory pressure on P

Handling Page Faults – Case 3

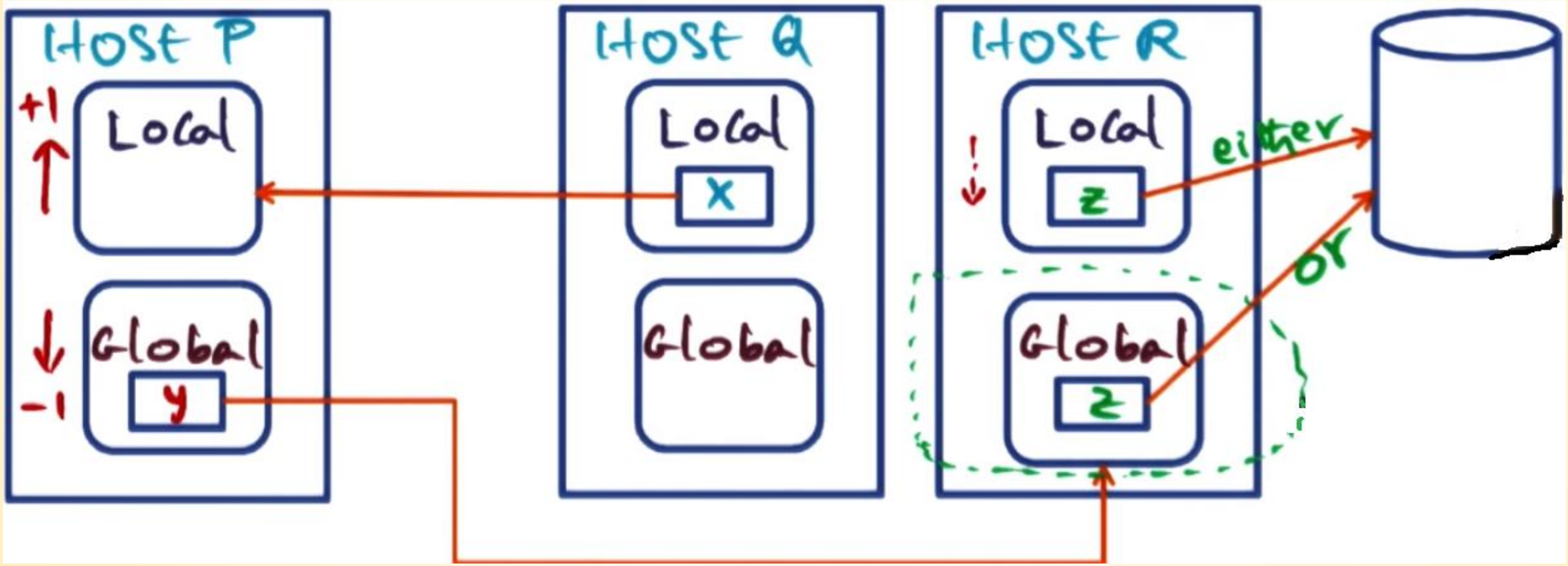
- Page fault for x at node P.
- Page not in cluster



Faulting page on disk

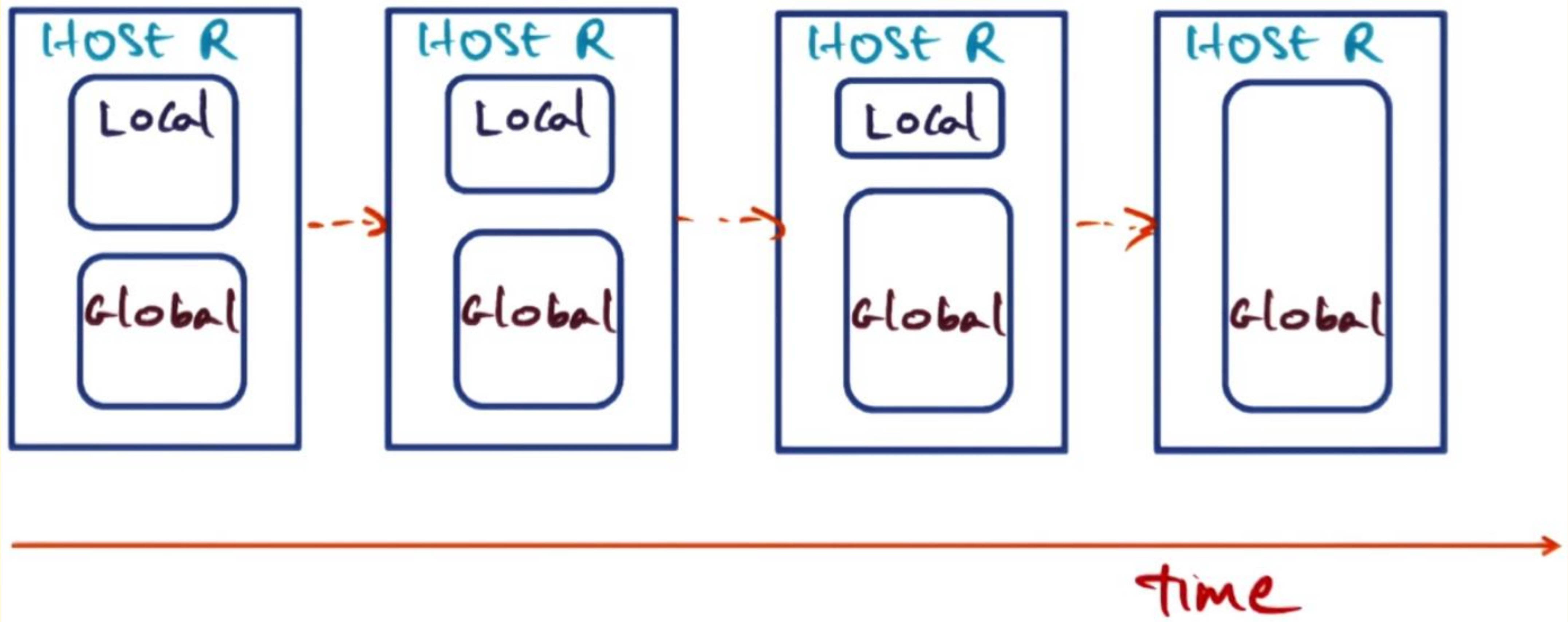
Handling Page Faults – Case 4

- Page fault for x at node P.
- Page is at peer node Q's local cache



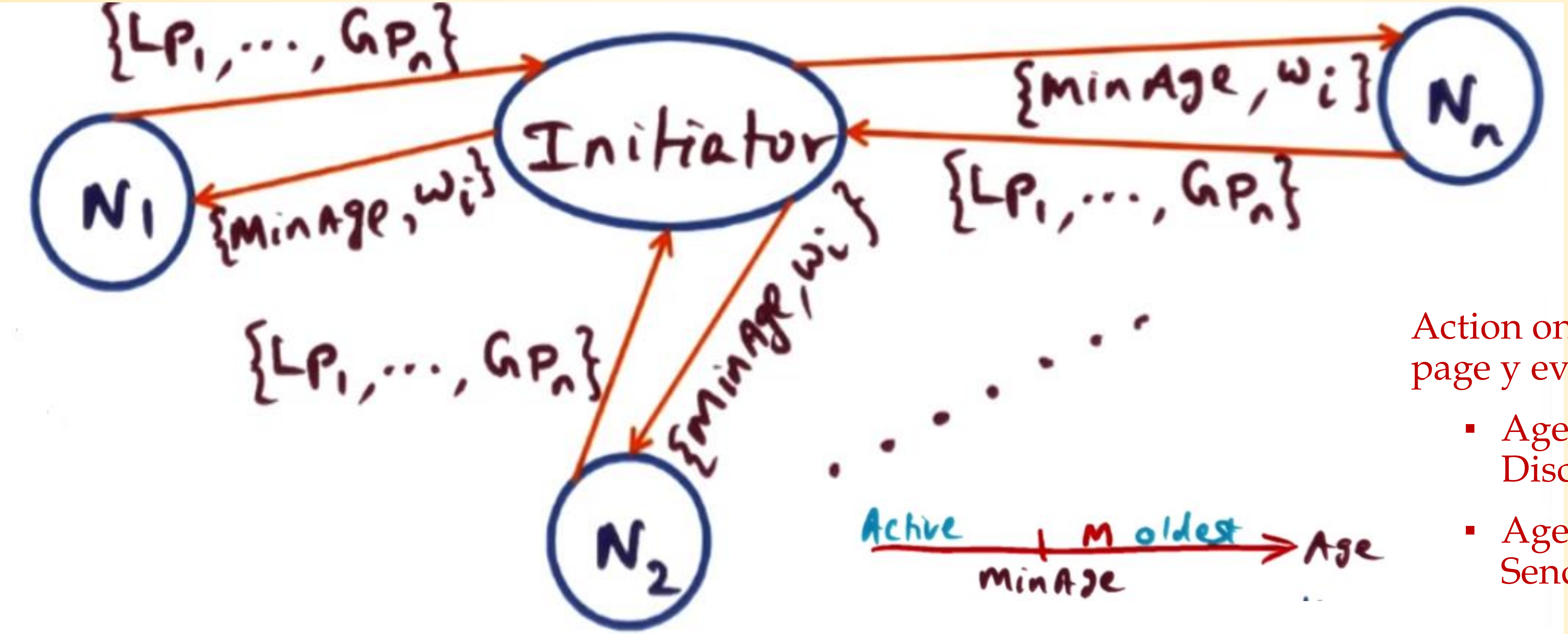
Faulting page actively shared

Behavioral of Algorithm



- When R become idle, R becomes memory server for peers on the cluster.
- When R becomes active again, Global memory can shrink back to allow Local to grow.

Geriatrics Management (Age Management)



Action on page fault, page y eviction candidate:

- Age(y) > minAge => Discard
- Age(y) < minAge => Send to peer N_i

Epoch Parameters

- T maximum duration
- M maximum replacement

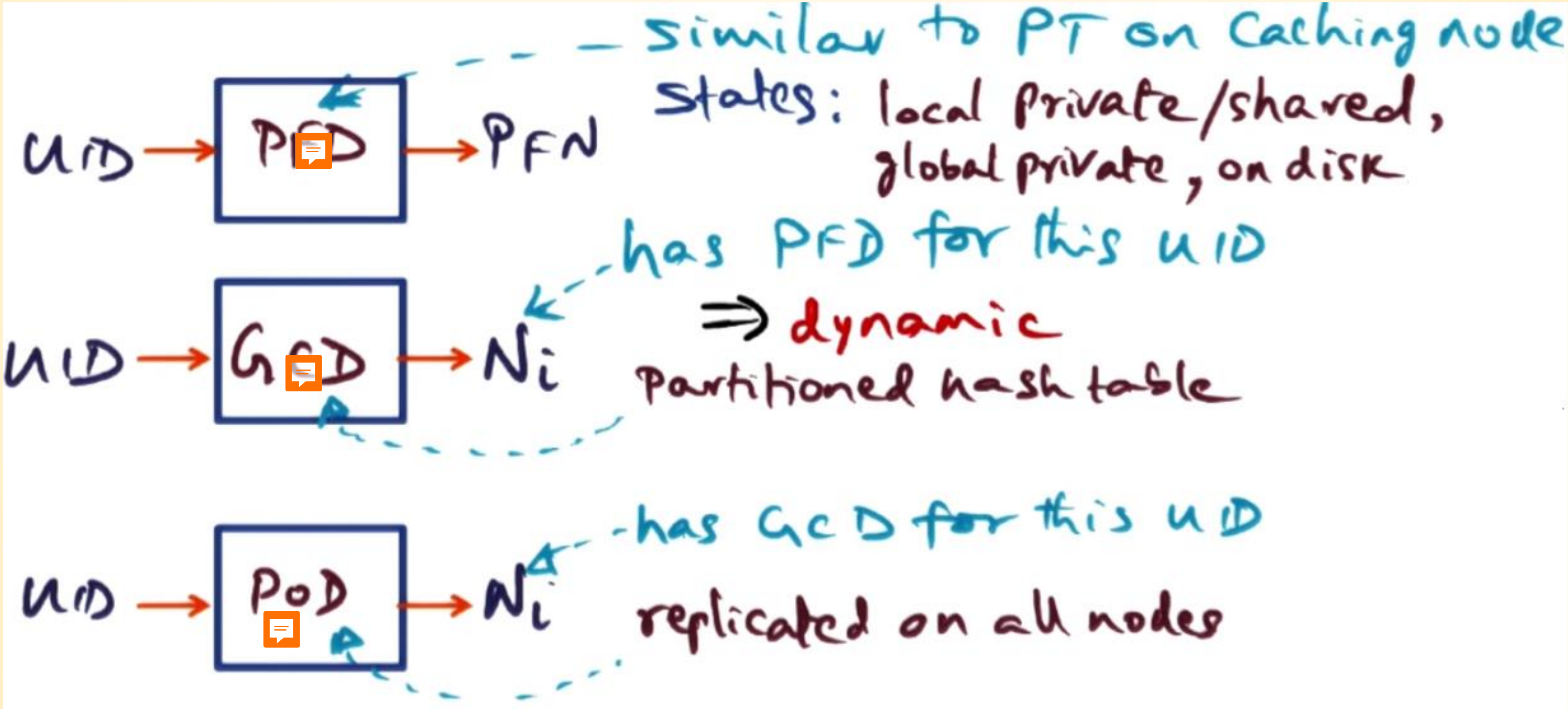
Each Epoch:

- Send age info to initiator
- Receive {minAge, W_i}

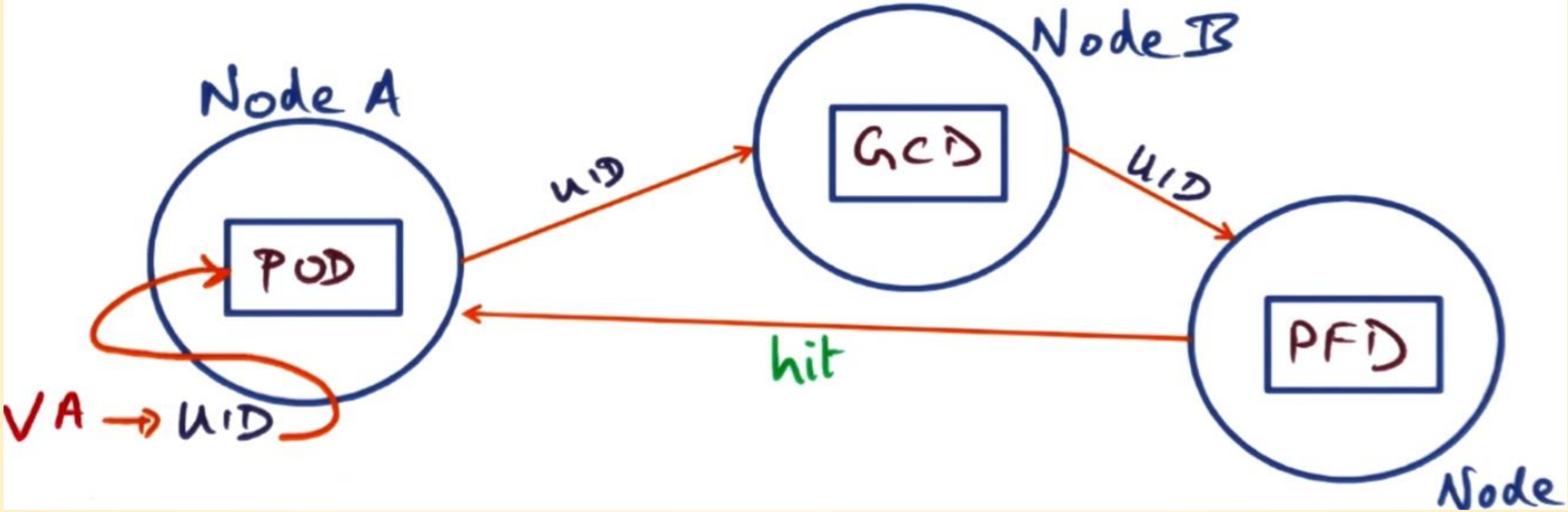
Initiator for next Epoch:

- Node with max W_i

GMS Data Structures



Putting Data Structures to Work



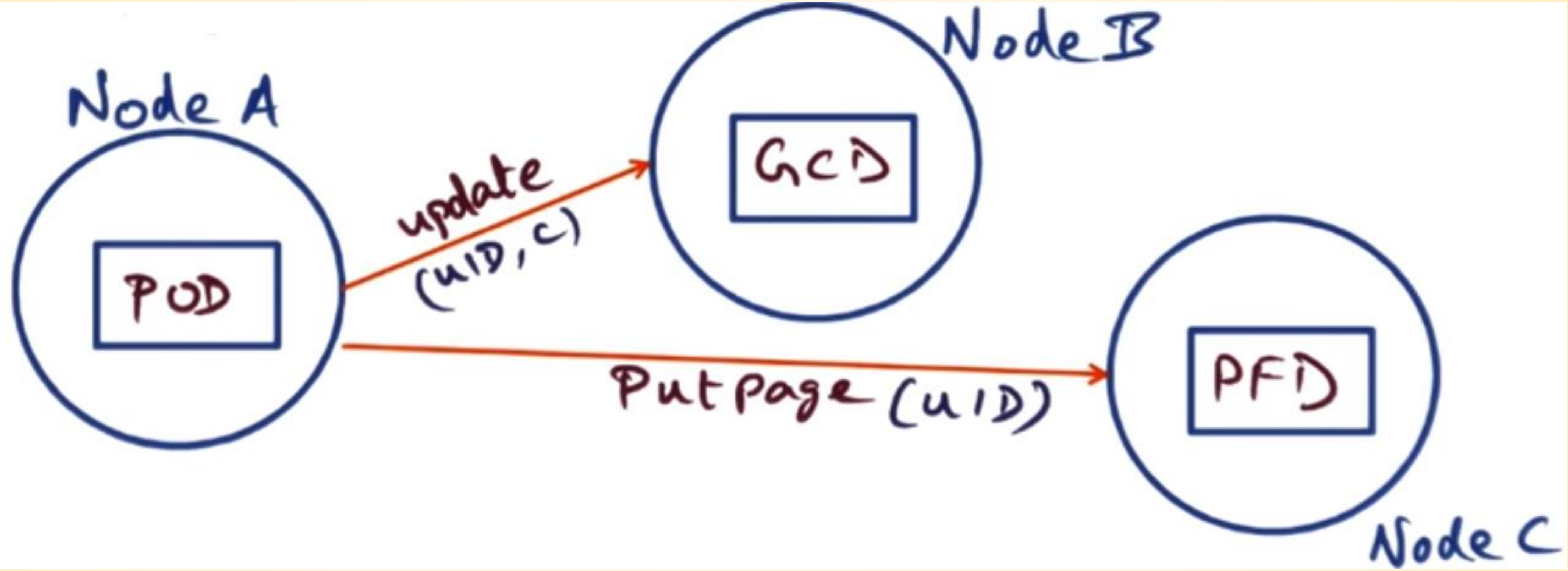
Common Case:

- Page non-shared
  =>A & B same node
- Page fault service quick

Miss?

- PFD changed – evicted out the page to other node
- POD changed – addition/deletion of nodes (uncommon case)

Putting Data Structures to Work – Page Eviction



Paging Daemon:

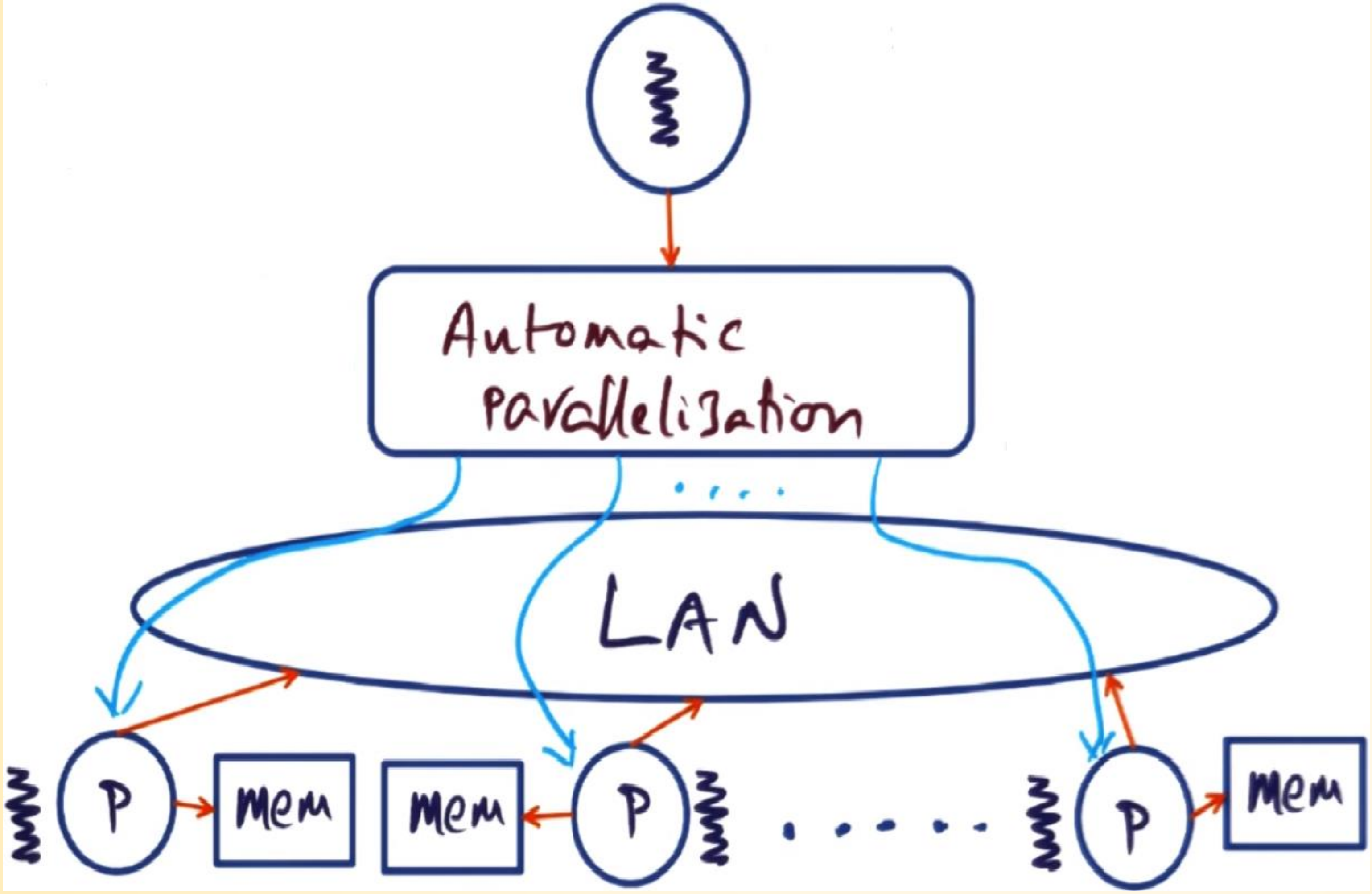
- Free list below threshold
- PutPage oldest pages
- Update GCD, PFD for the UID

Distributed Shared Memory

Cluster as Parallel Machine - Different Approaches

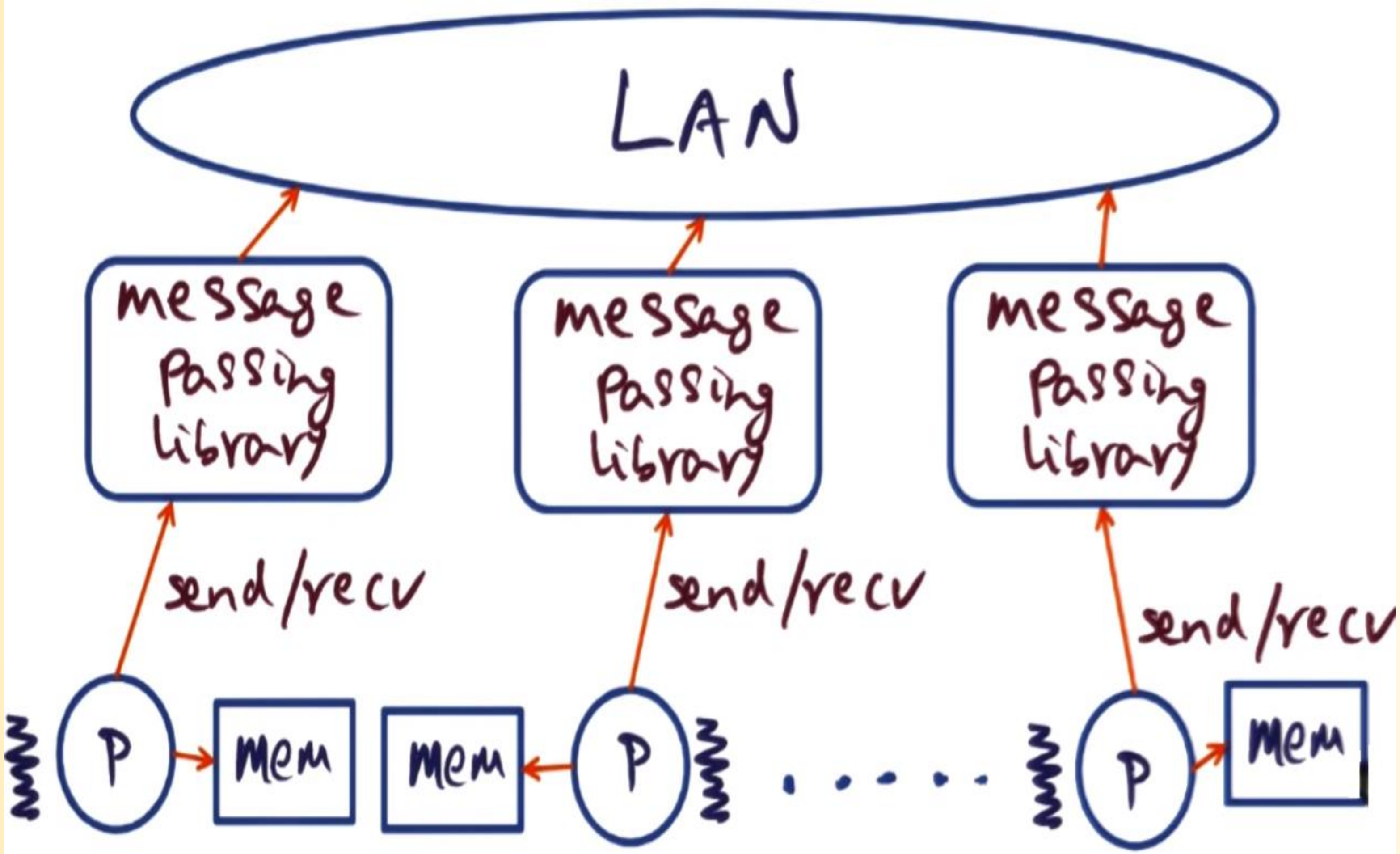
- Sequential Program
- Message Passing
- Distributed Shared Memory (DSM)

Sequential Program



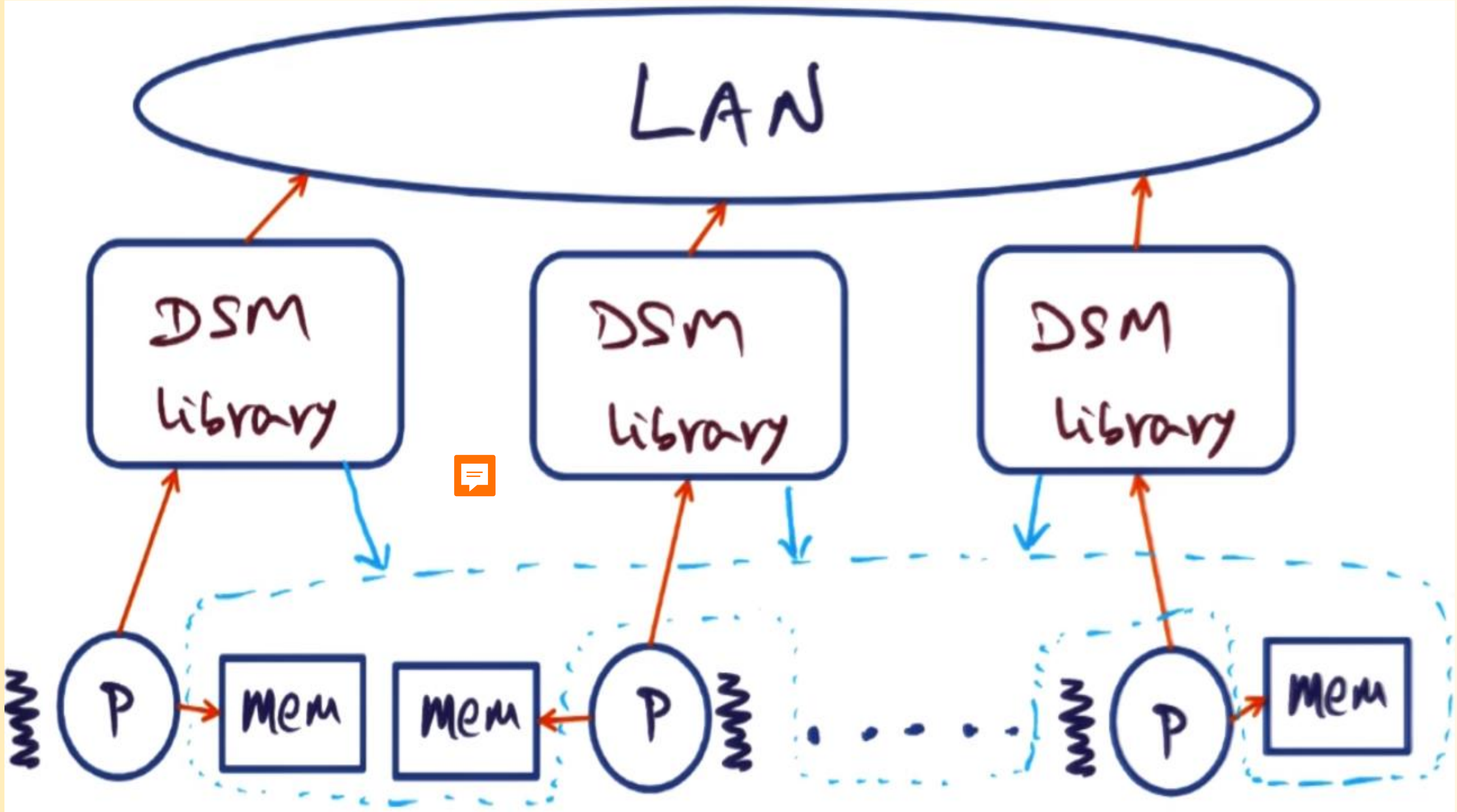
- Directives for data/computation distribution
- Data parallel programs
- Limited potential for exploiting available parallelism

Message Passing



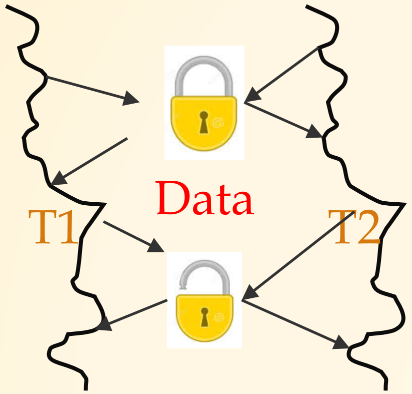
- Processor can't directly access to the memory that is not associated within.
- It has to send/receive message to access other processor's memory.
- Difficult to implement as utilizing message passing to access memory.

Distributed Shared Memory (DSM)

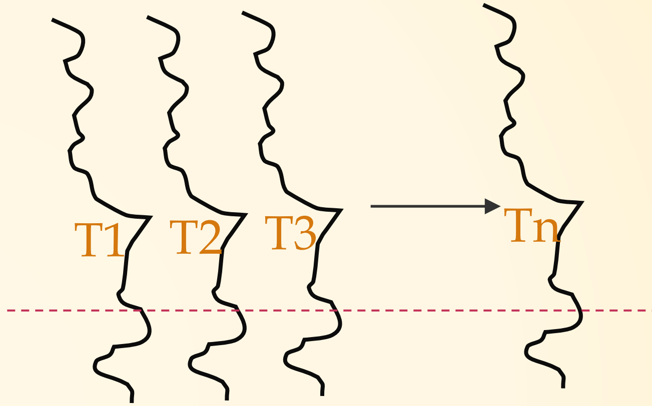


- DSM library gives an illusion to the processes that the entire memories of the cluster are shared among nodes.

Shared Memory Programming



Lock



Barrier

Two Types of Memory Access:

- Normal R/W to shared data
- R/W to synchronization variables

Memory Consistency and Cache Coherence

- There must be strong relationship/partnership between the memory consistency (software) and cache coherence (hardware).

- Memory consistency: what's the model presented to the programmer?
  - Is a contract between the app programmer and the system
  - Answering the **"when"** question.
  - How soon the change going to be made visible to other processes that have the same memory location in their respective cache.

- Cache coherence: how is the system implementing the model in the presence of private cache?
  - Answering the **"how"** question
  - How the system (software + hardware working together) implementing the contract of the memory consistency model.
    - CC: hardware will make sure the cache is coherent => Hardware Cache Coherence.
    - NCC: No hardware cache coherent, let system software to ensure cache is coherent.

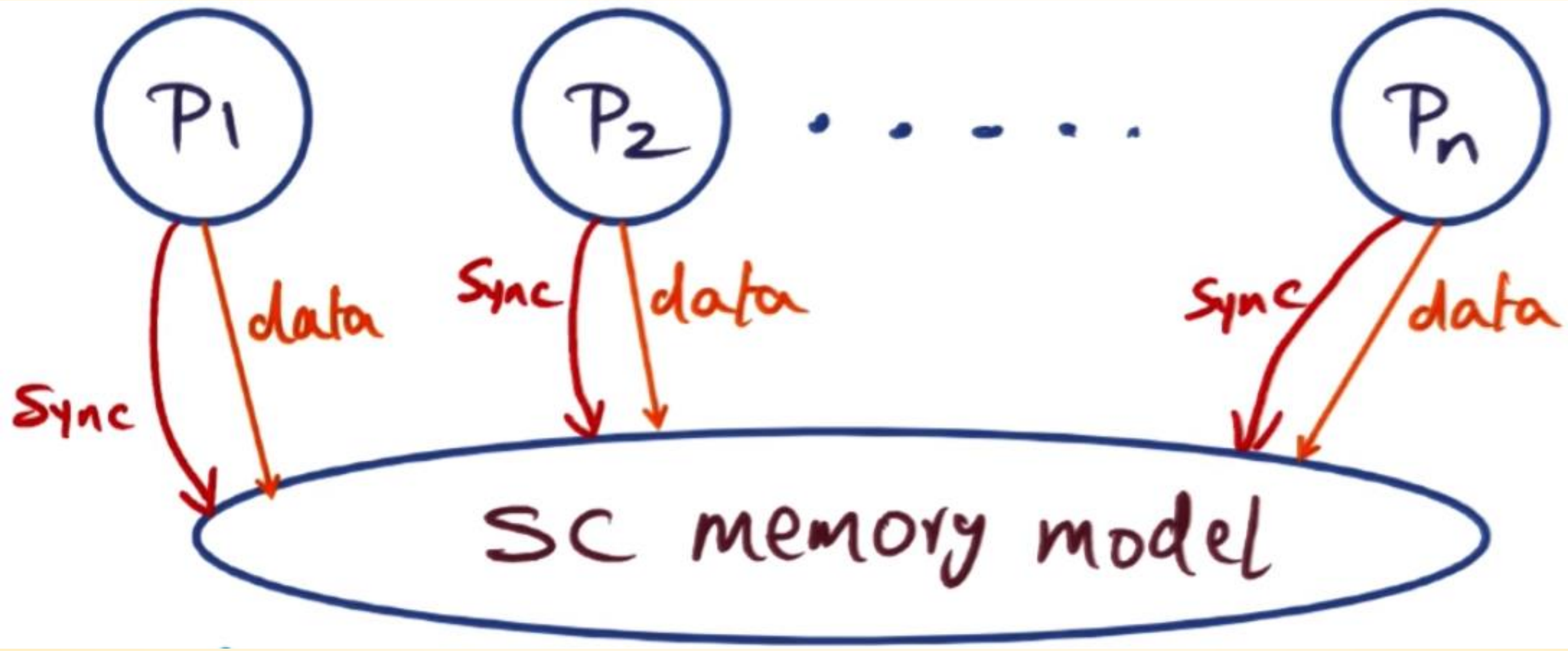
Sequential Consistency

Process P 1:
R (a)
W (b)
W (c)
R (b)

Process P 2:
W (a)
R (b)
W (c)

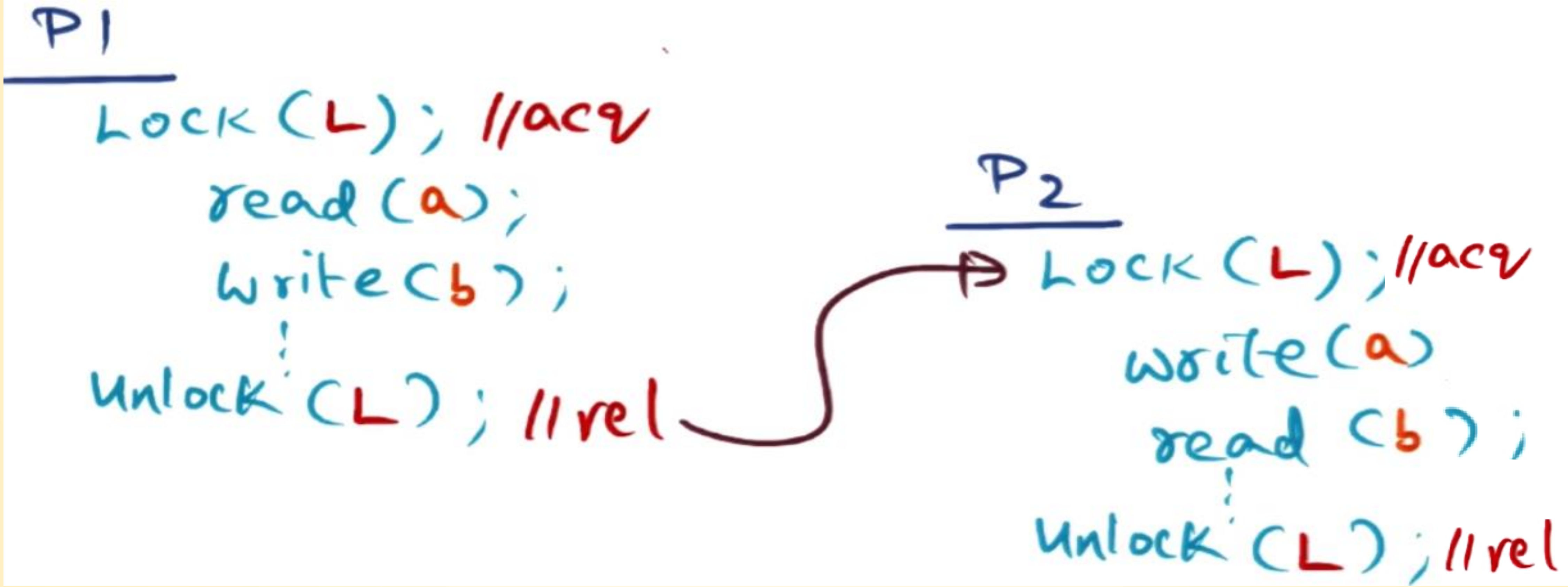
Program order +
arbitrary interleaving

Sequential Consistency (SC) Memory Model



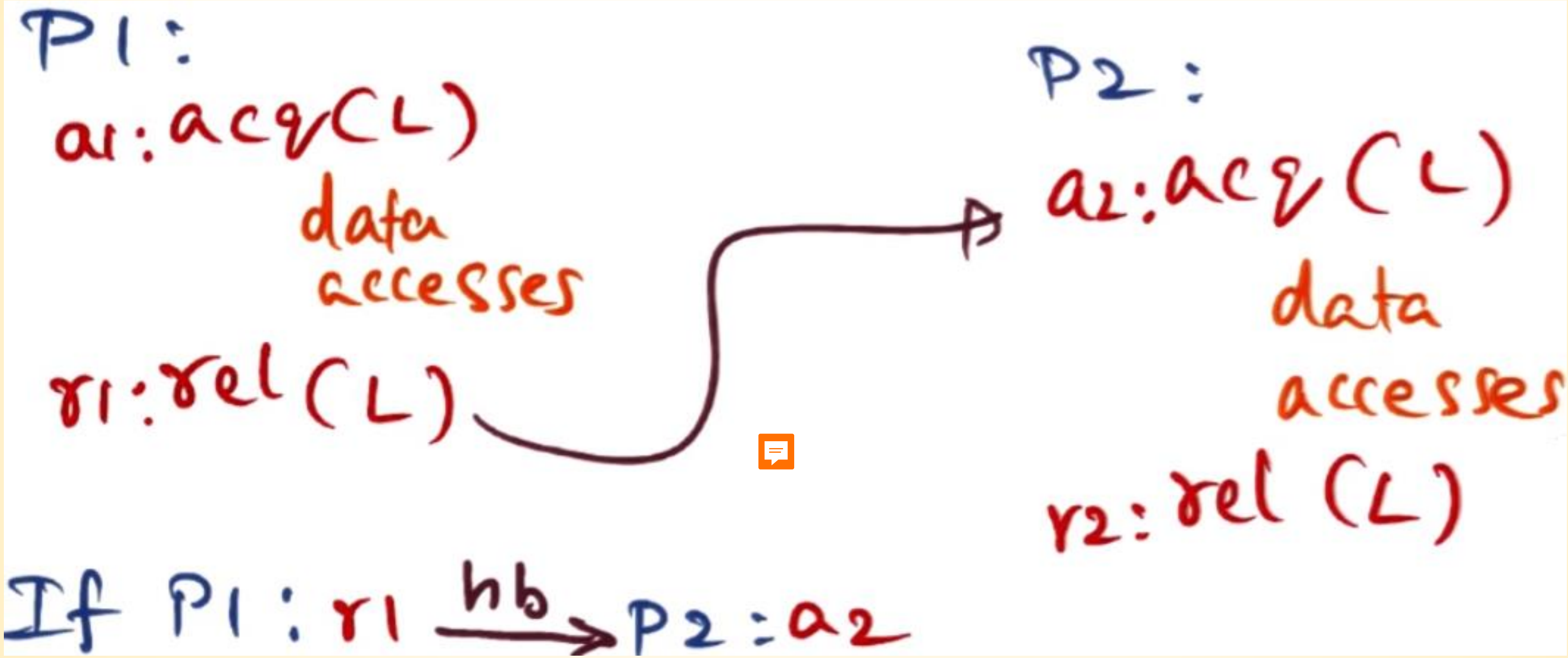
- SC doesn't distinguish between data r/w and synchronization r/w
- Upshot: coherent actions happen on every r/w access

Typical Parallel Program



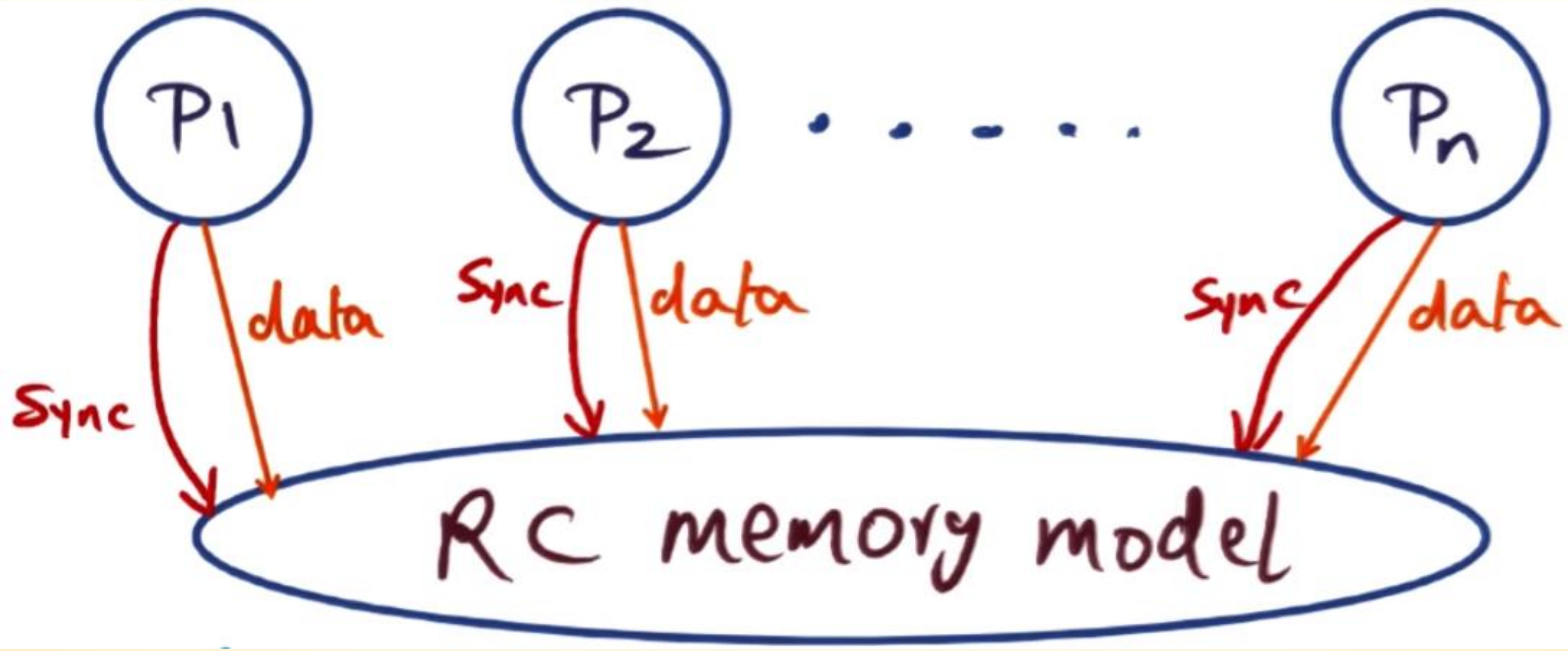
- P2 doesn't access data until P1 releases L
- Thus, coherent actions are not needed until P1 releases L

Release Consistency



- All coherent actions prior to P1:r1, should be completed before P2:a2.

Release Consistency (RC) Memory Model

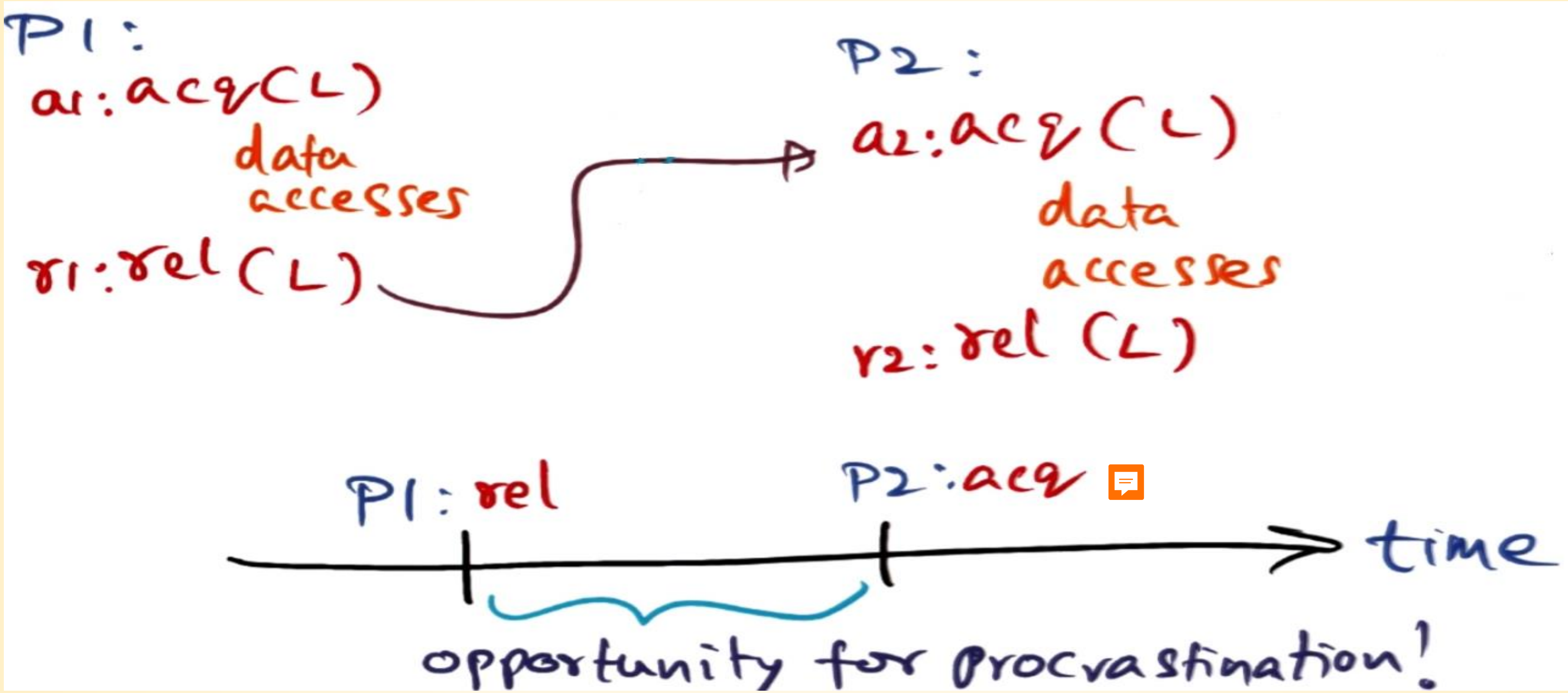


- RC distinguishes between data r/w and synchronization r/w
- Upshot: coherent actions happen only when lock is released

Advantage of RC Over SC

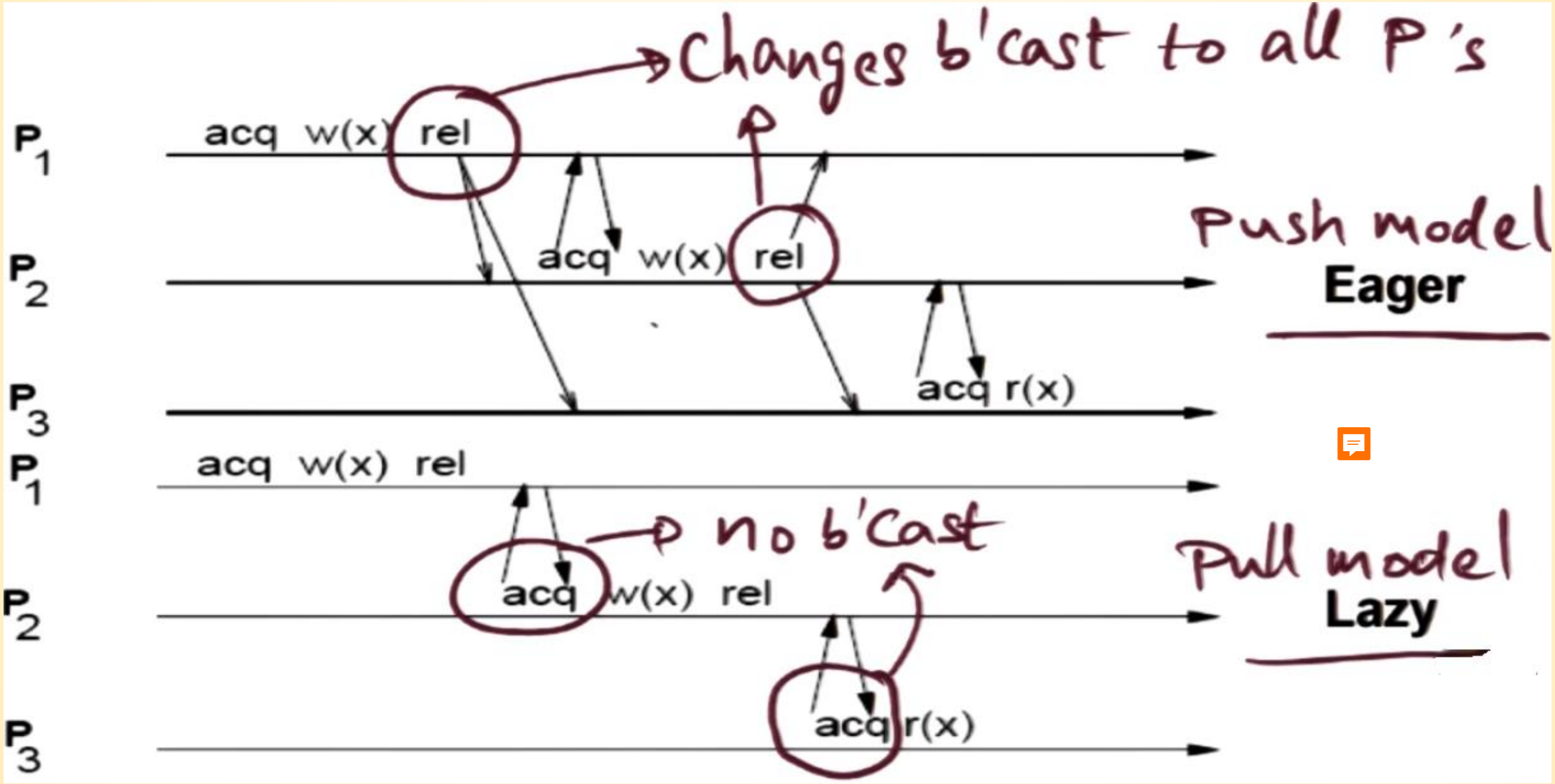
- No waiting for coherent actions on every memory access:
  - $\Rightarrow$ Overlap computation with communication
  - $\Rightarrow$ Better performance for RC over SC

"Lazy" Release Consistency (Lazy RC)

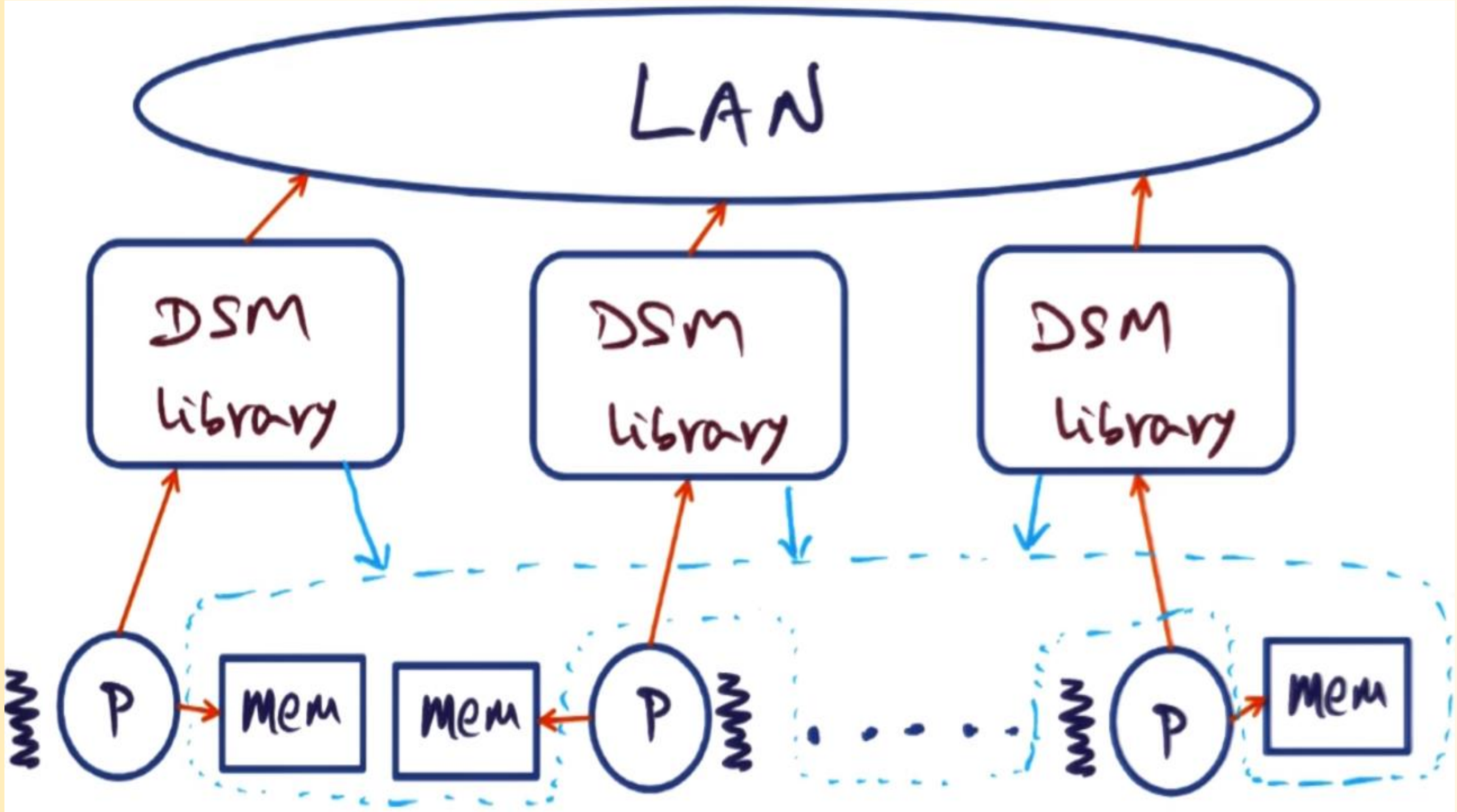


- Coherent actions happen at acquire rather than at release.

"Eager" RC vs "Lazy" RC



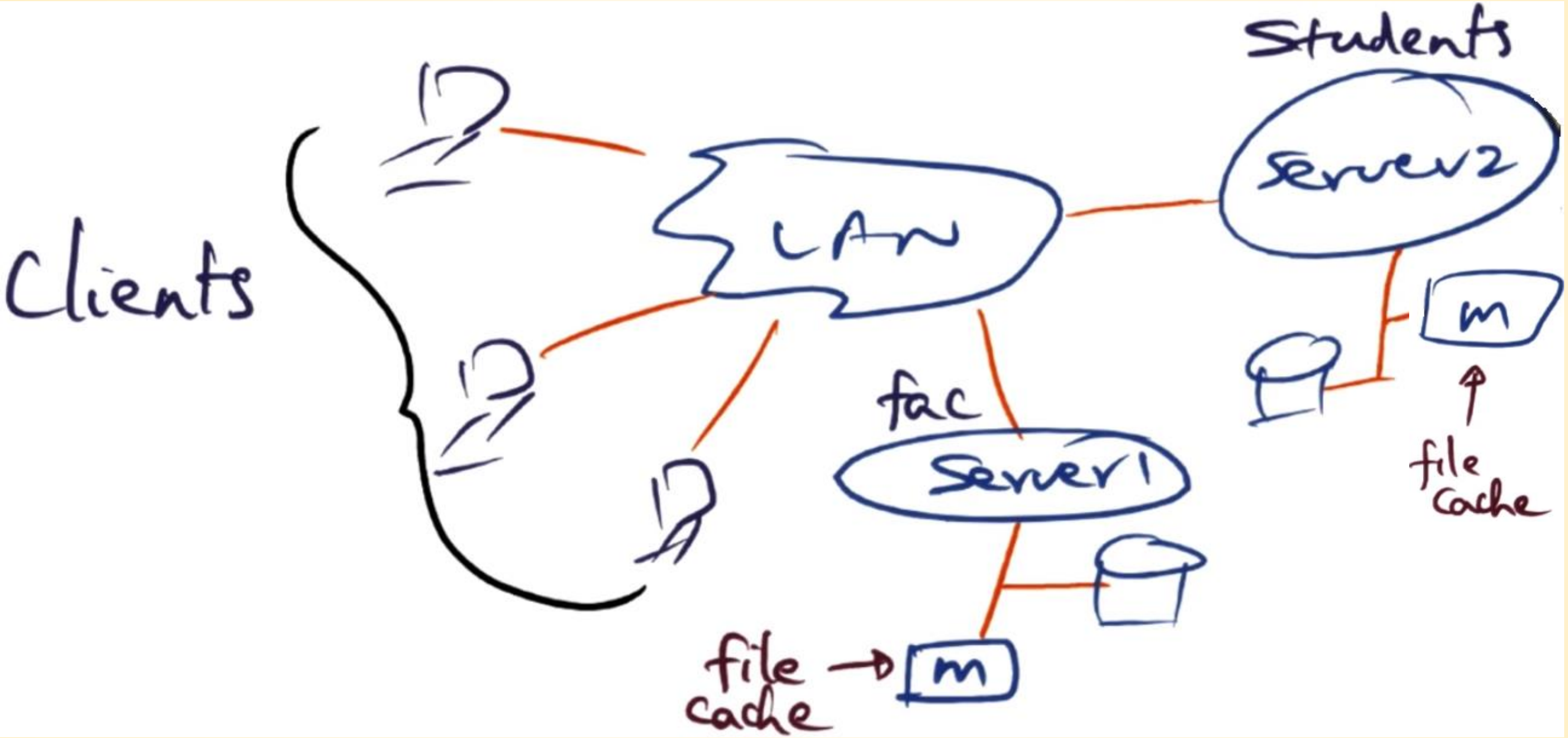
Distributed Shared Memory (DSM)



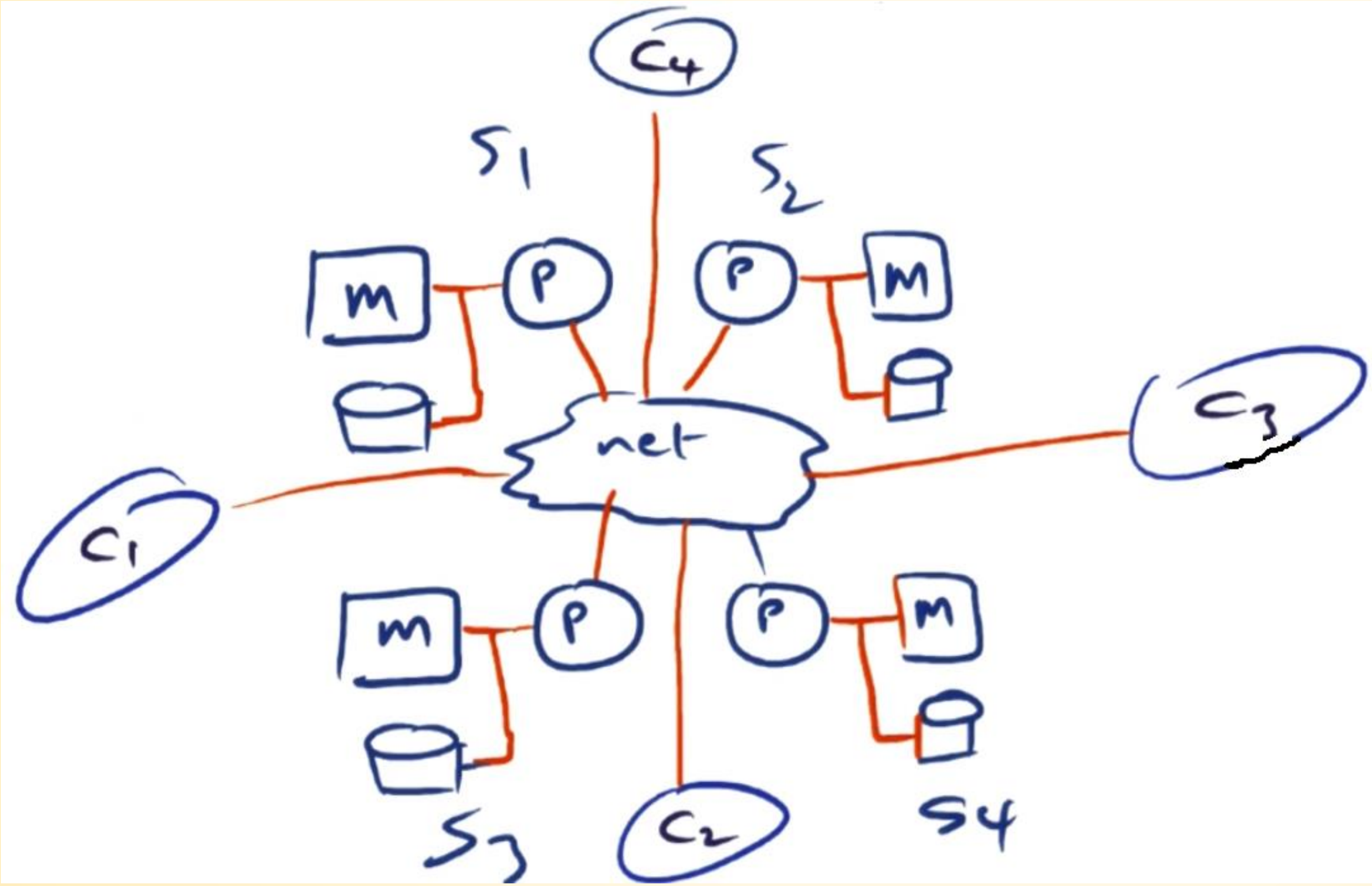
- Software DSM
- Lazy RC with Multi Writer Coherence Protocol

Distributed File Systems

Network File System (NFS)

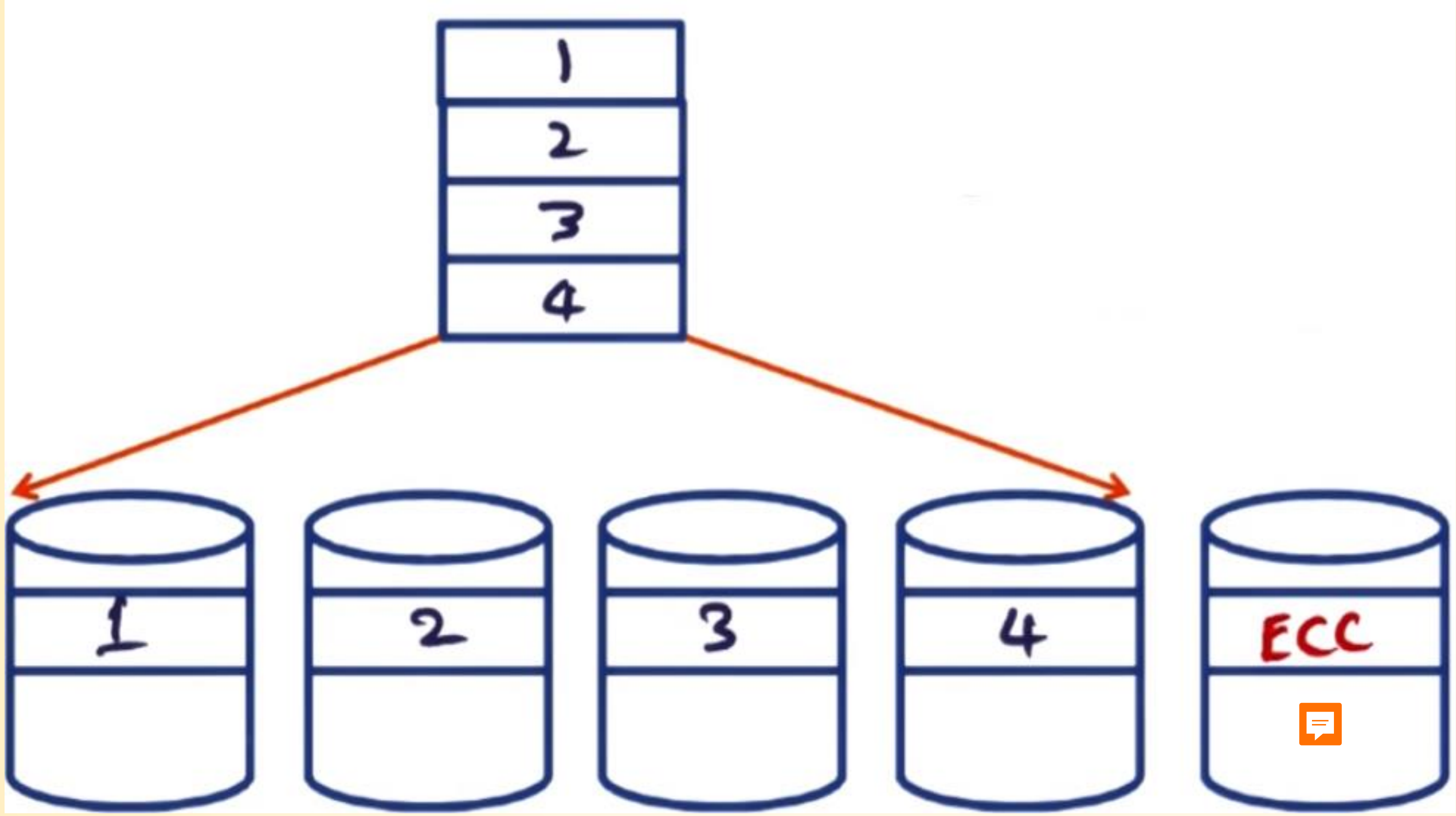


Distributed File System (DFS)



- No central server.
- Each file distributed across several servers (nodes).
- Implemented all disks in the network.

Preliminaries: Stripping a File to Multiple Disks



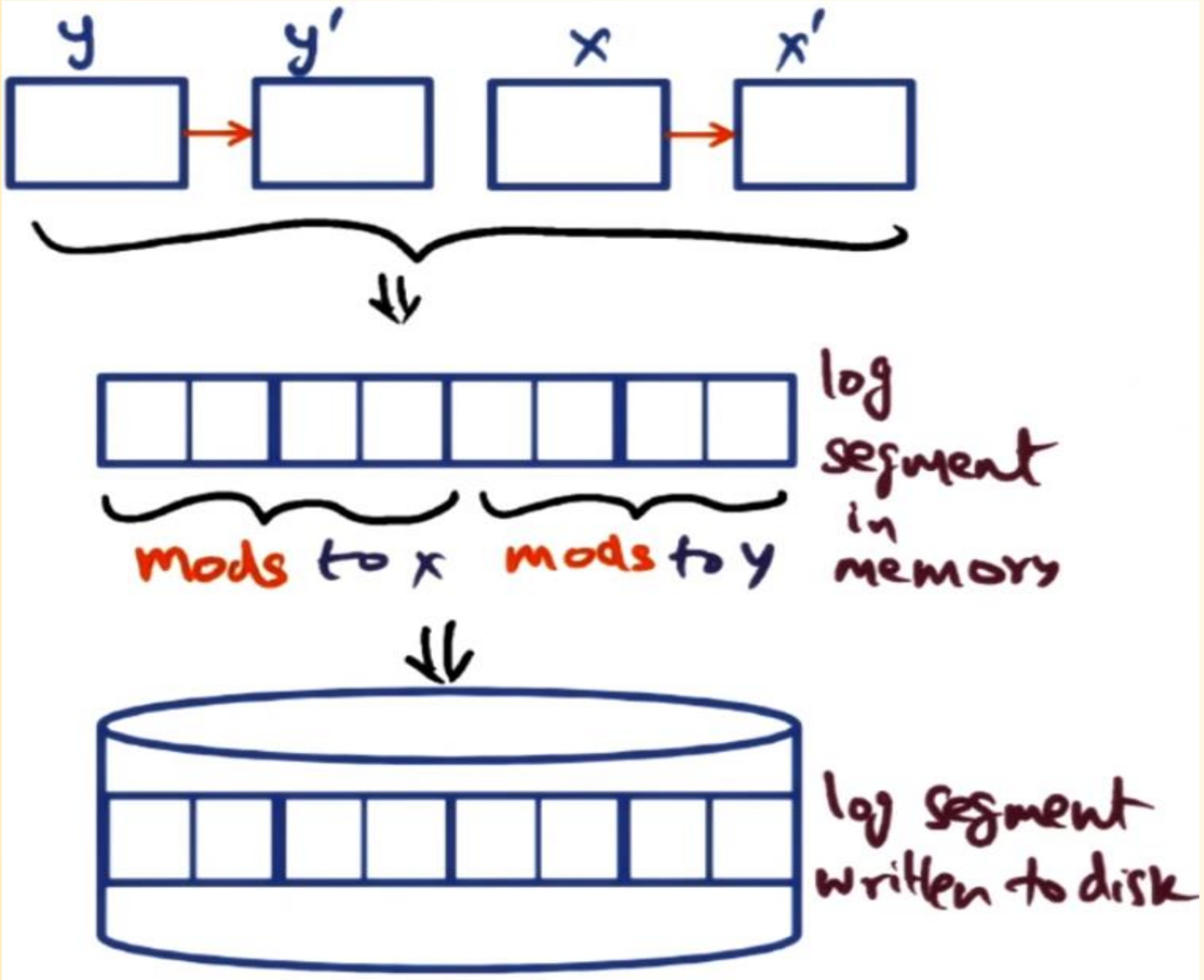
- Increase I/O bandwidth by striping to parallel disks.
- Failure protection by ECC

- Drawbacks:

  ⇒Cost

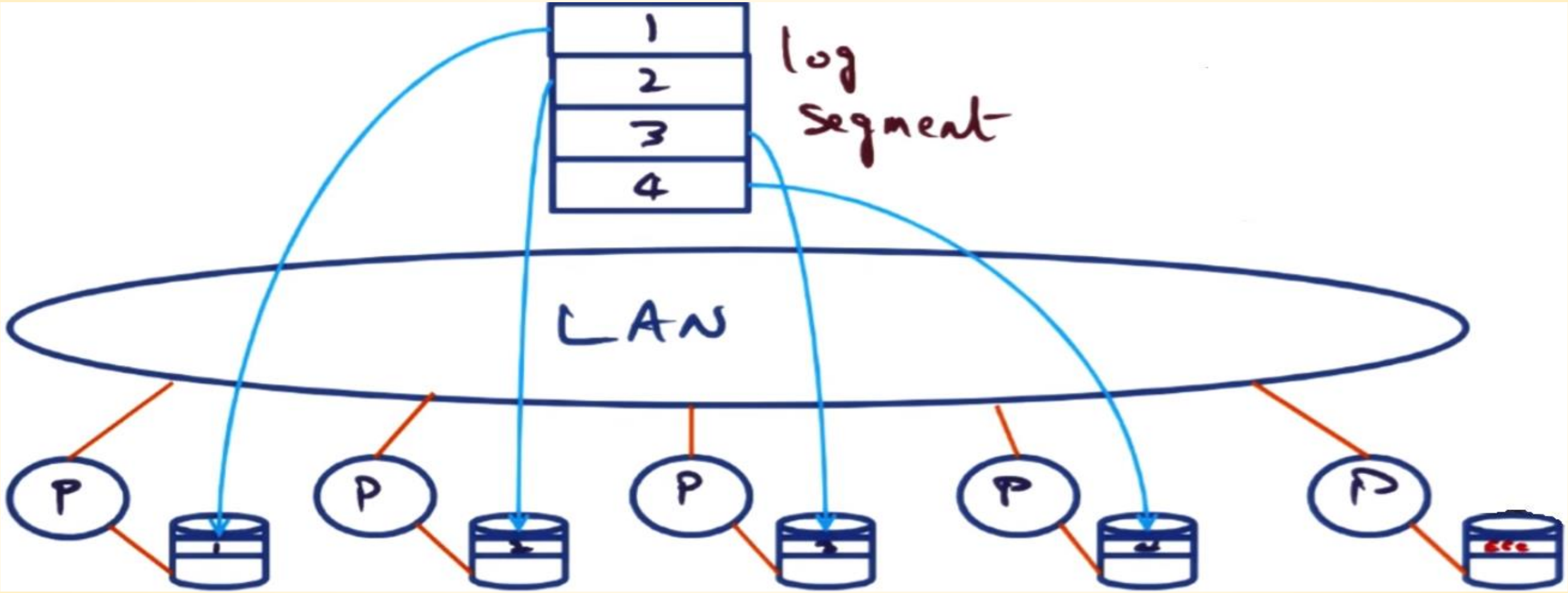
  ⇒Small write problem

Preliminaries: Log Structured File System



- Buffer changes to multiple files in one contiguous log segment data structures.
- Flush log segment to disk once it fills up or periodically

Preliminaries Software (RAID)

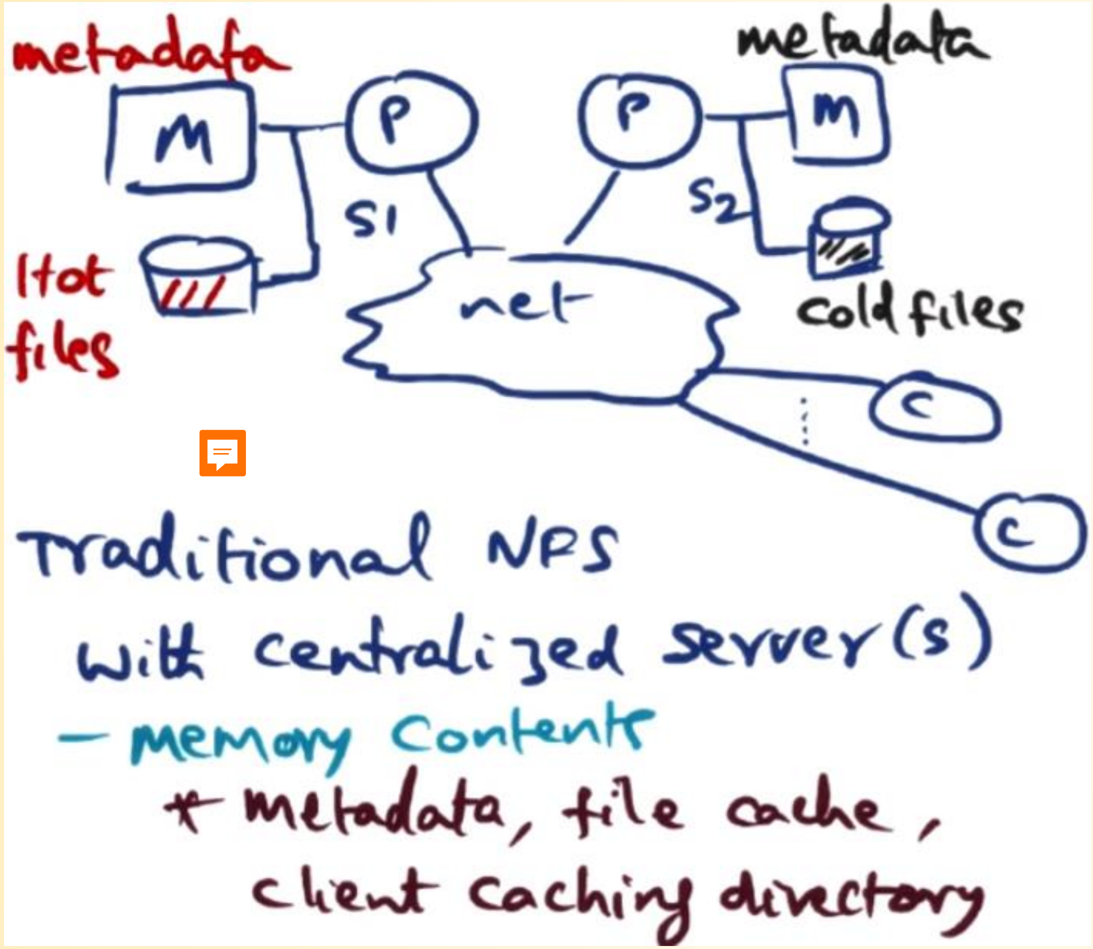


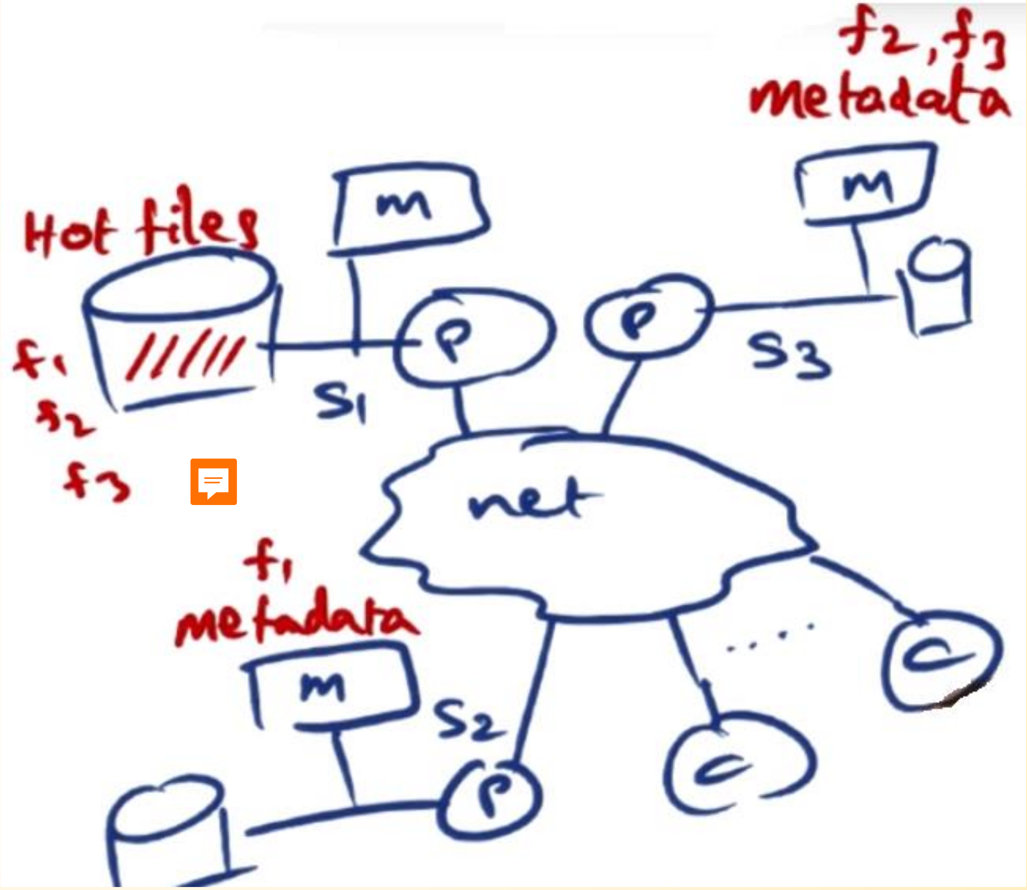
- Combines LFS + RAID
- Stripe log segment on multiple nodes' disks in software

Putting All Together Plus Other Concepts

- XFS – is a DFS:
  - ⇒ Log Based Stripping
  - ⇒ Cooperative Caching
  - ⇒ Dynamic Management of Data + Metadata
  - ⇒ Sub-setting Storage Server
  - ⇒ Distributed Log Cleaning

Dynamic Management

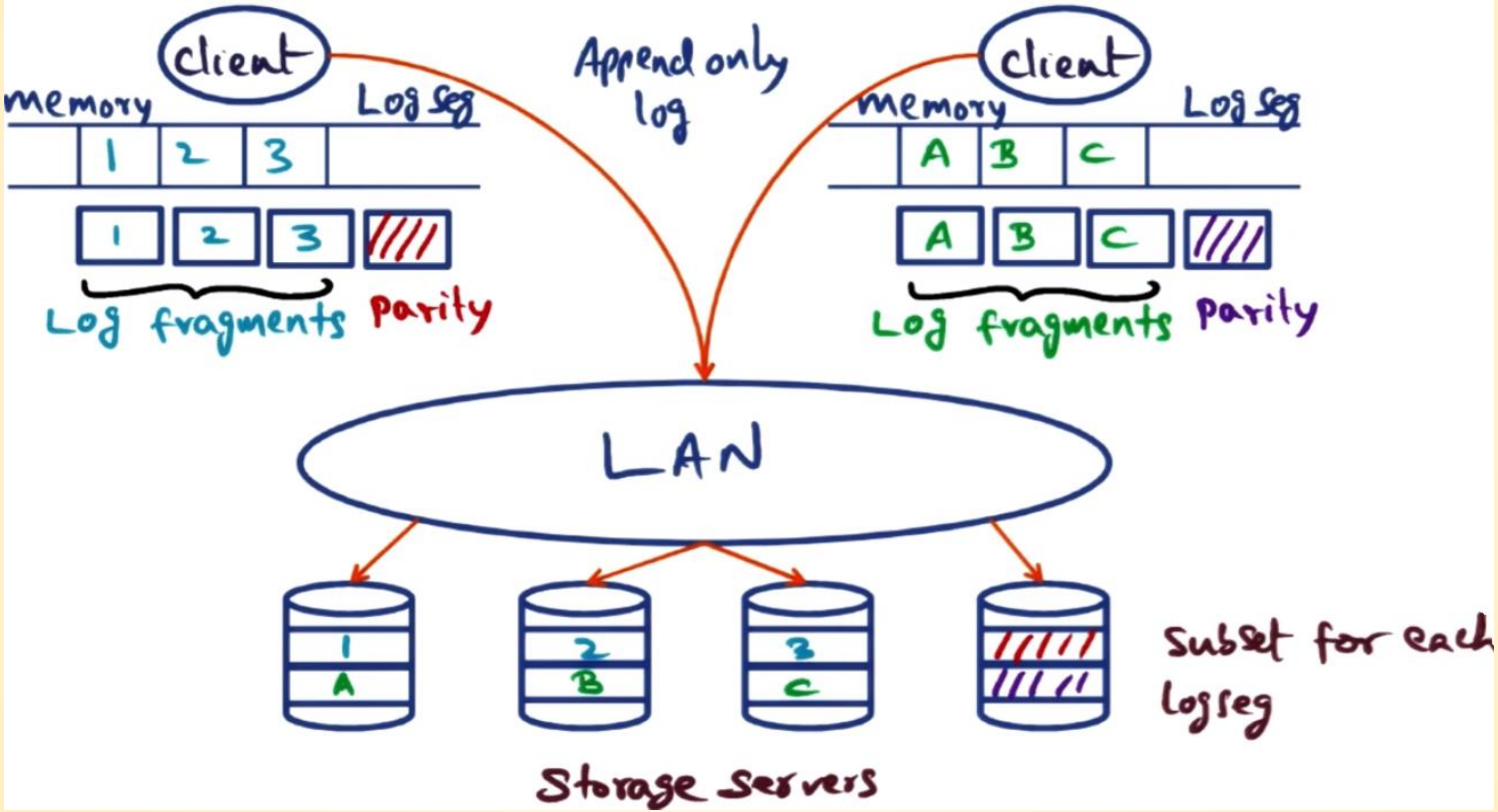




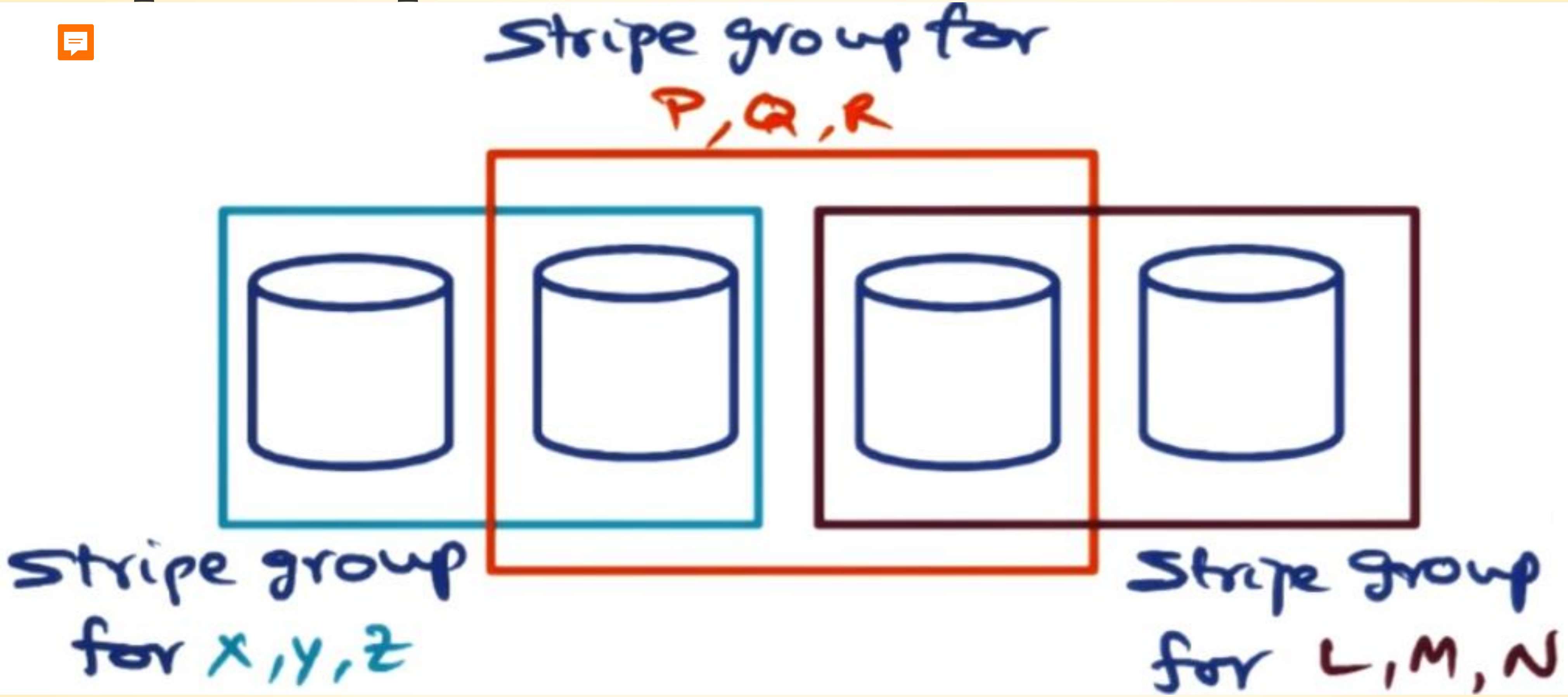
XFS:

- Data management dynamically distributed
- Cooperative client file caching

Log Based Stripping and Stripe Groups

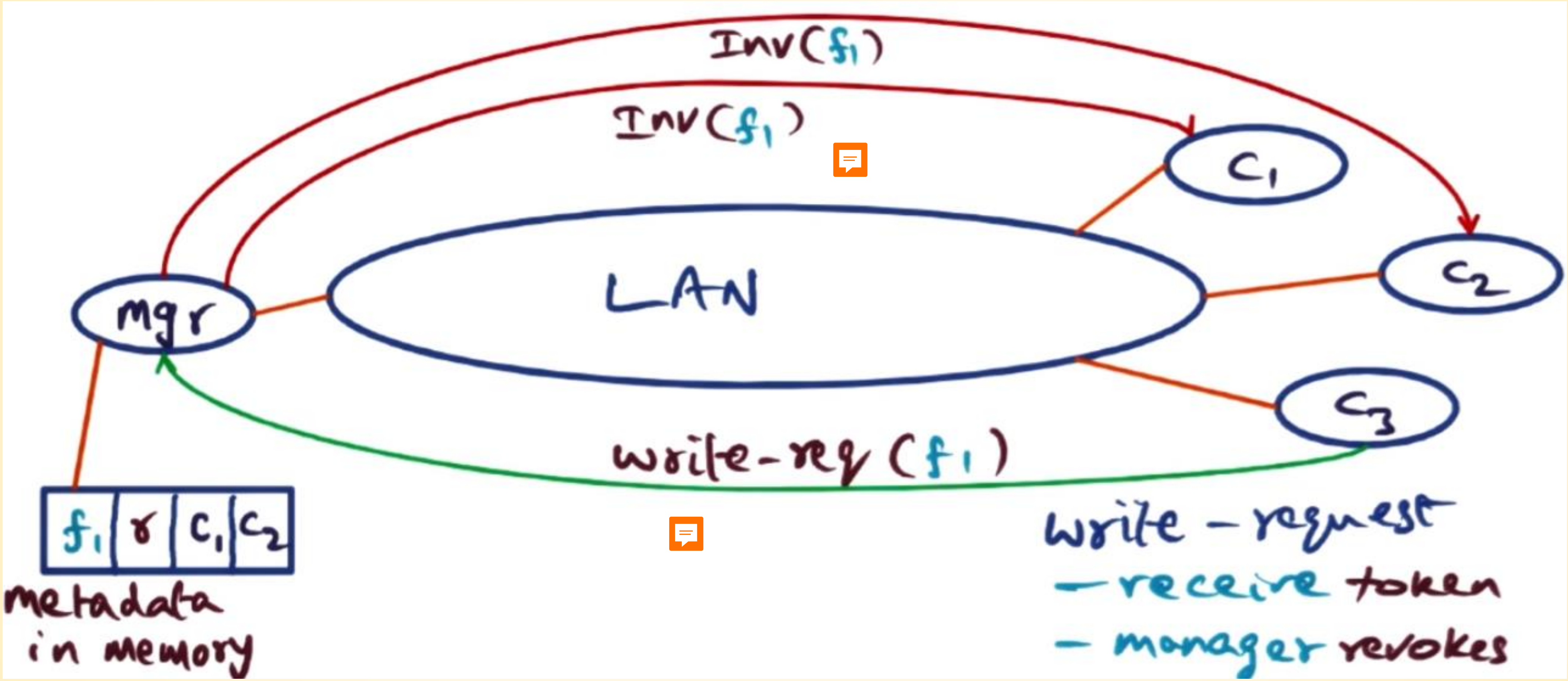


Stripe Group



- Subset servers into stripe groups
- Parallel client activities
- Increased availability
- Efficient log cleaning.

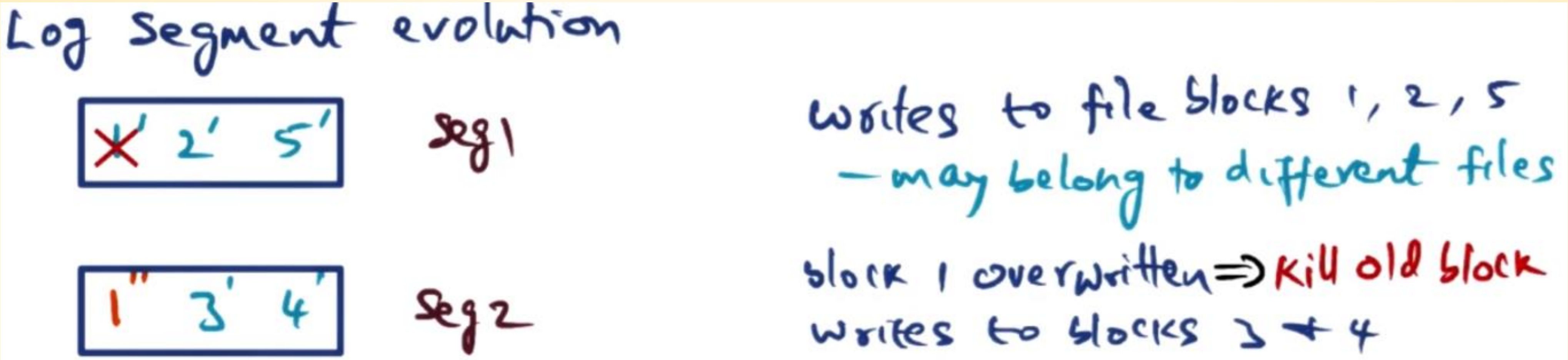
Cooperative Caching

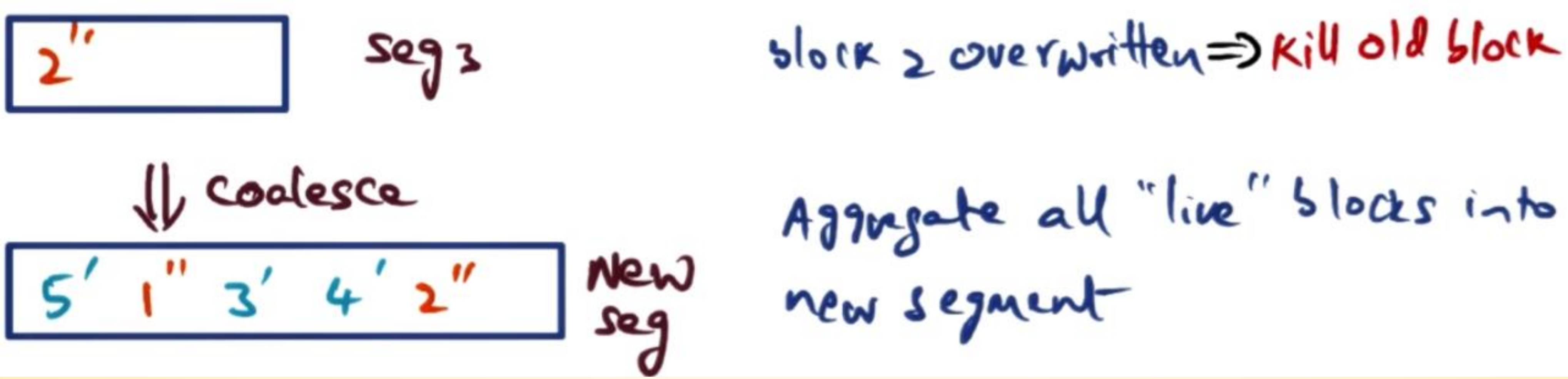


Cache Coherence:

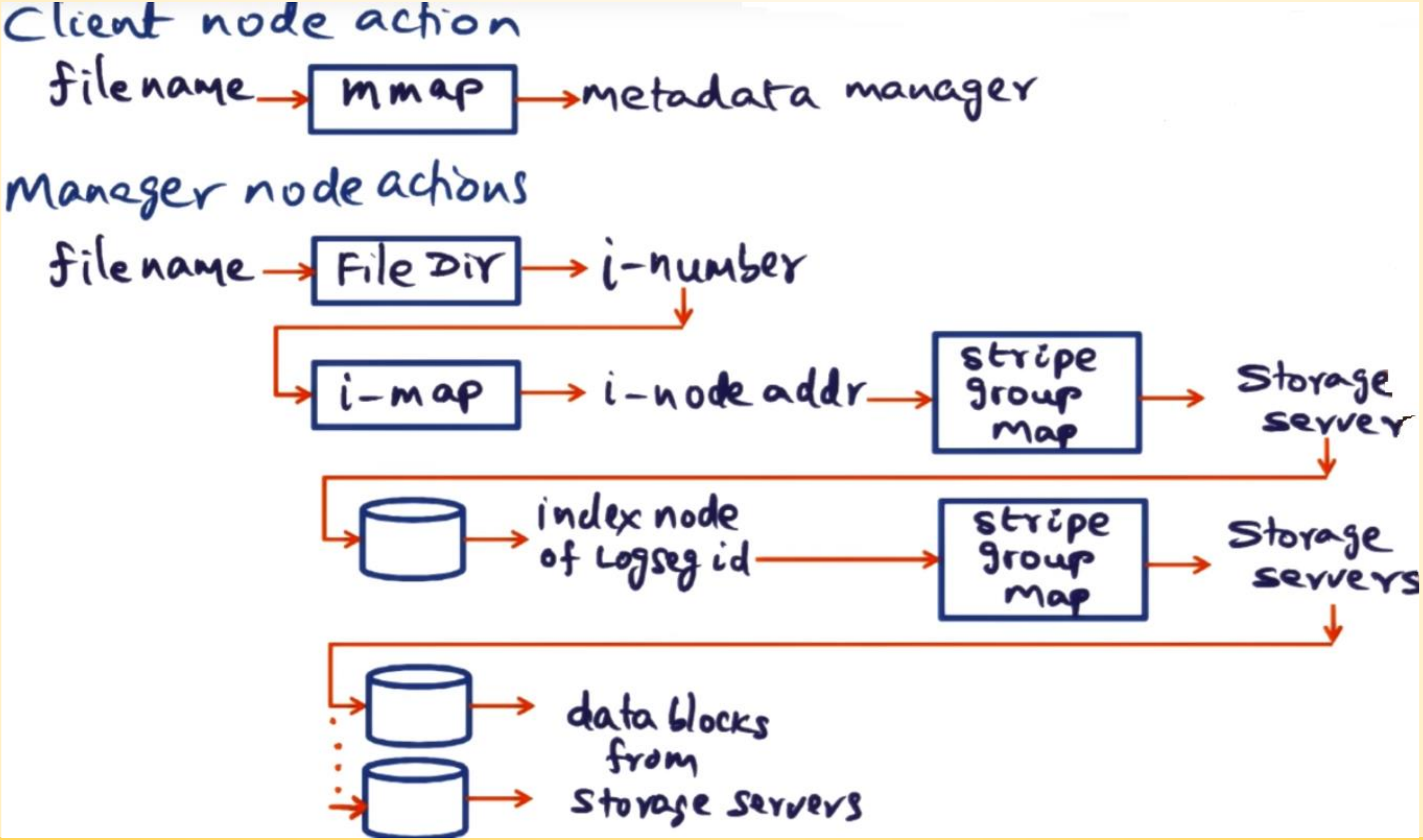
- Single writer, multiple readers
- File block unit of coherence

Log Cleaning

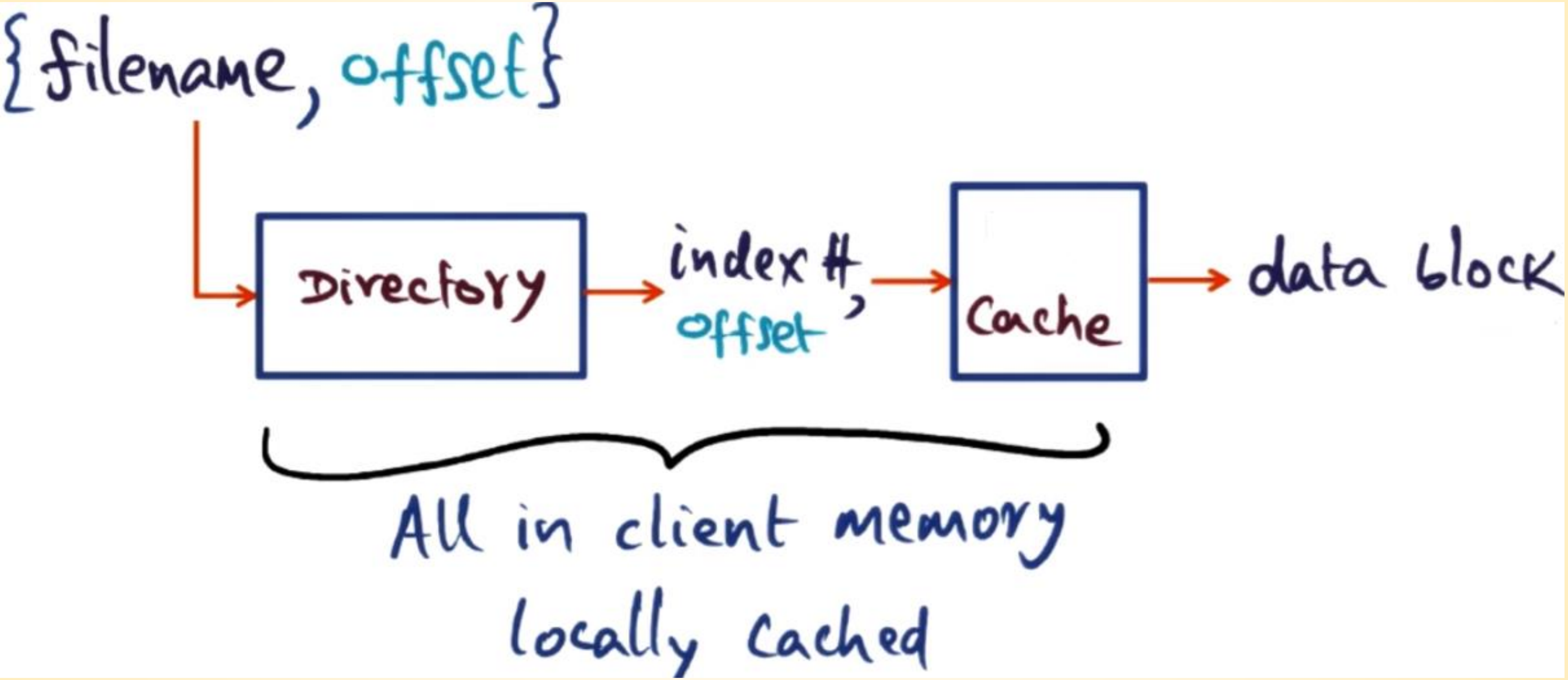




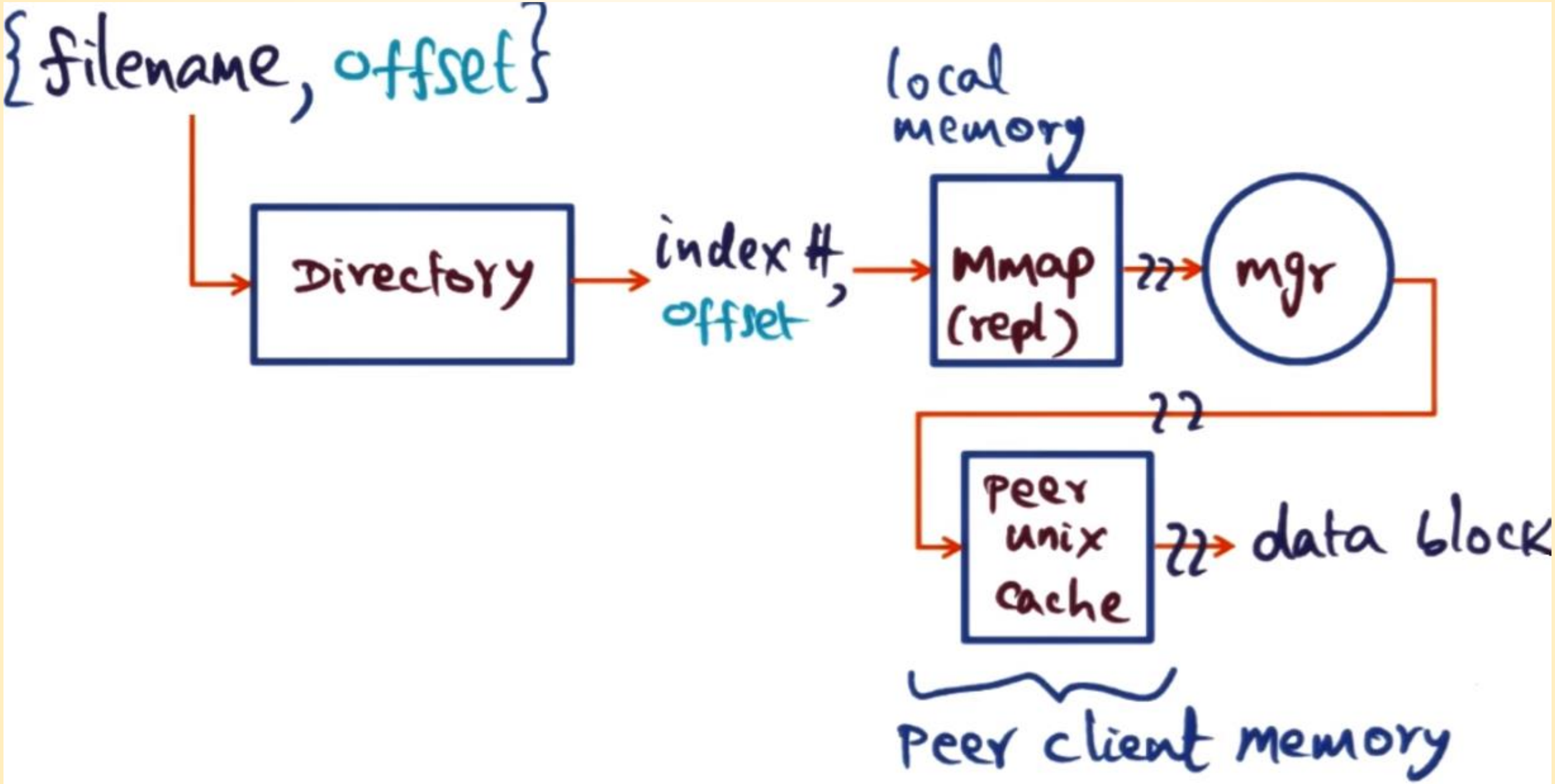
XFS Data Structures



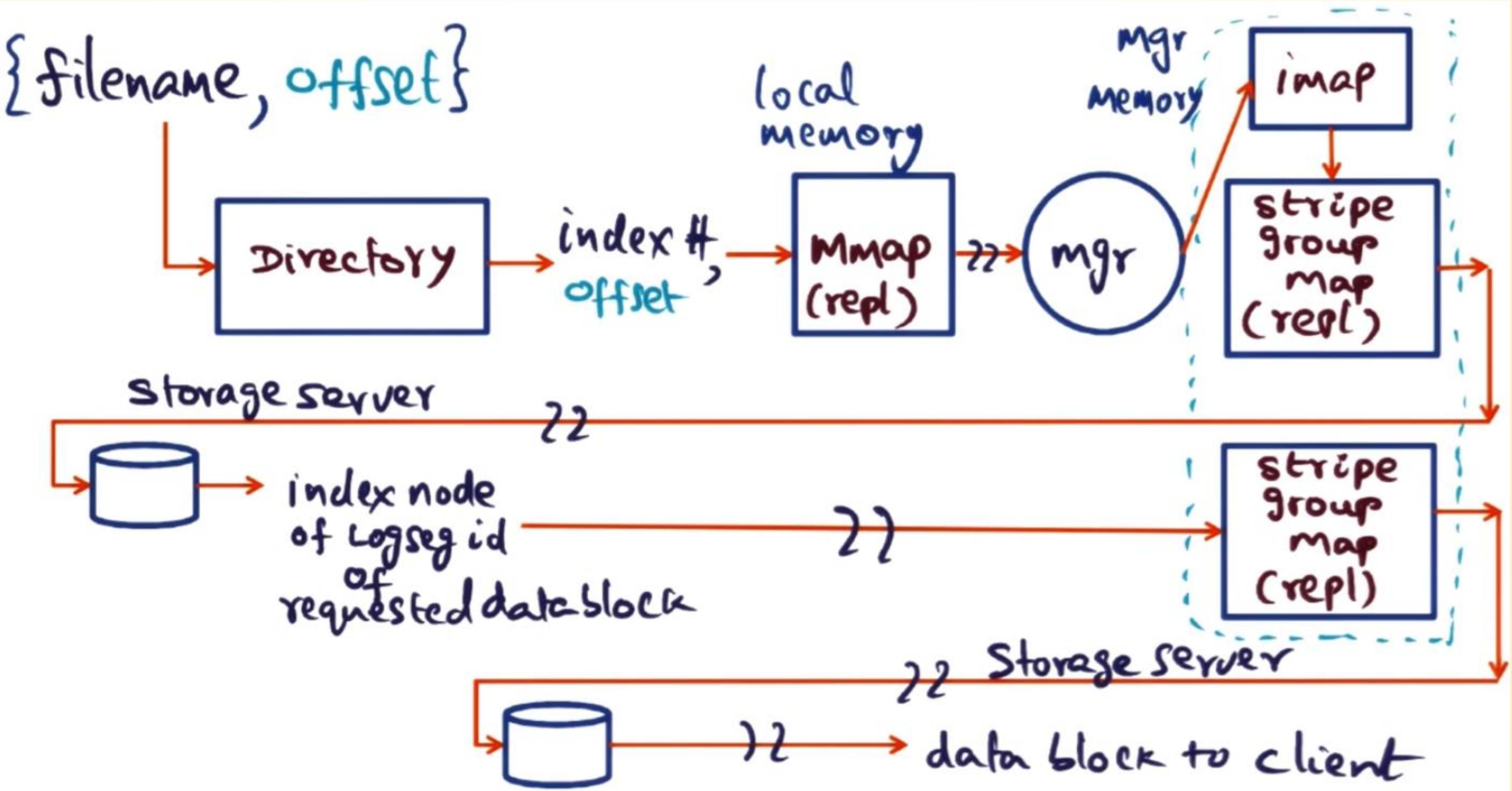
Client Reading A File – Fastest Path



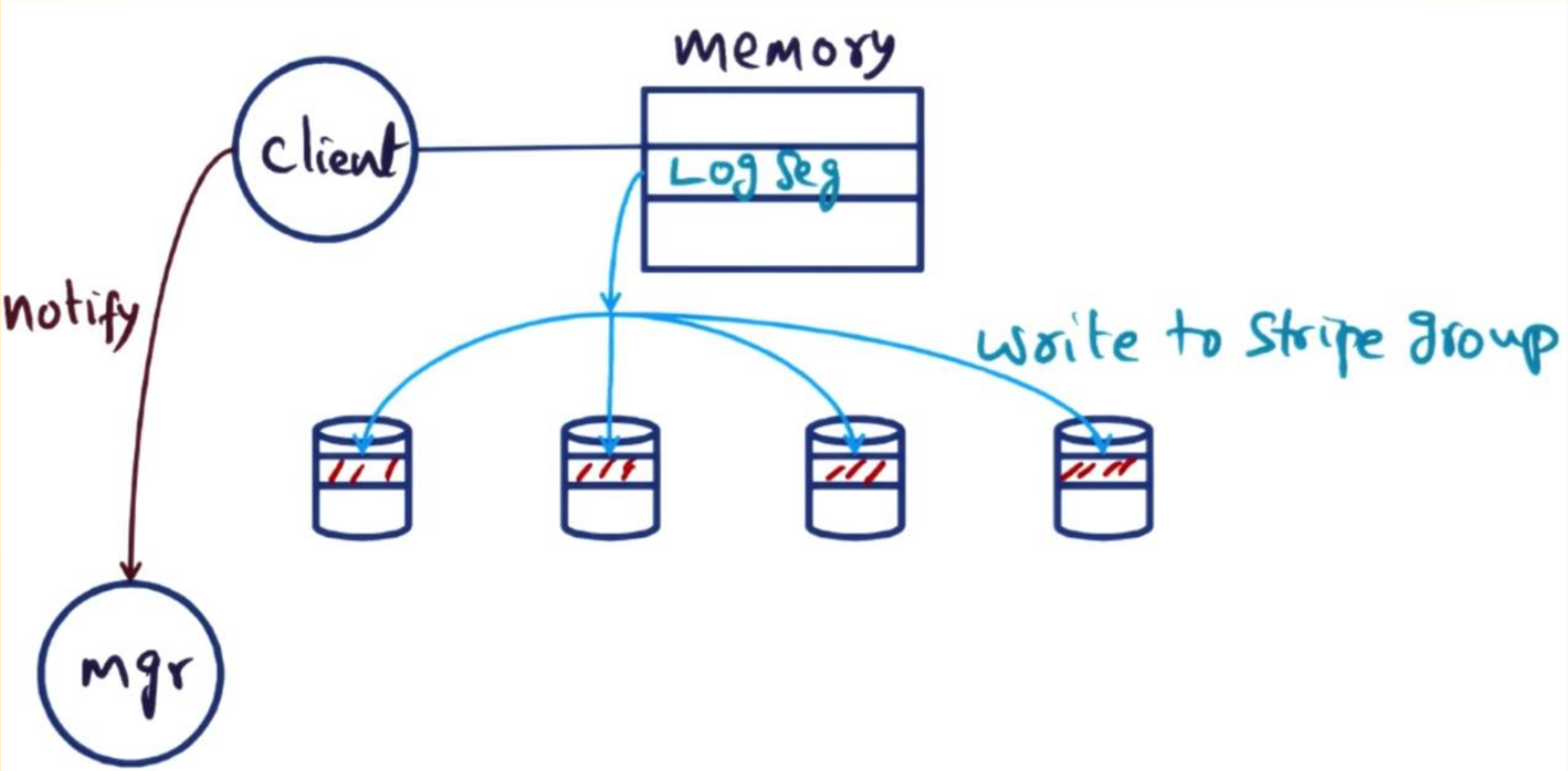
Client Reading A File – Second Best Path



Client Reading A File – The Very Long Way



Client Writing A File



Conclusion

We discussed a lot of concepts pertaining design and implementation of Distributed File System.

In particular, how to make the implementation scalable by removing centralization and utilizing memory that is available in the nodes of a LAN intelligently.

We looked at creative ways in GMS, DSM and DFS, to fully utilize memory in the nodes of a LAN.