ECE 669

Parallel Computer Architecture

Lecture 17

Memory Systems

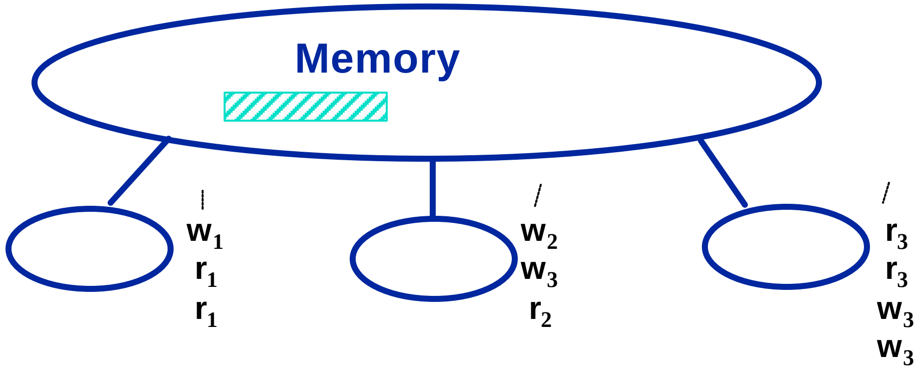
Memory Characteristics

- Caching performance important for system performance
- Caching tightly integrated with networking
- Physcial properties
    - Consider topology and distribution of memory
- Develop an effective coherency strategy
- Limitless approach to caching
    - Allow scalable caching

Perspectives

° **Programming model and caching.**

or: the meaning of shared memory



Sequential consistency: Final state (of memory) is as if all RDs and WRTs were executed in some given serial order (per processor order maintained)

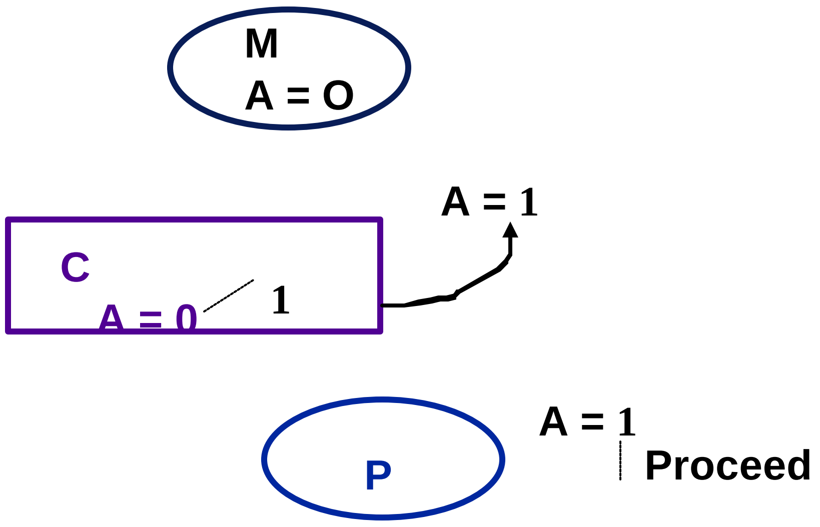
-Lamport

$r_1\ r_2\ r_1\ w_2\ w_2\ w_3$ ....

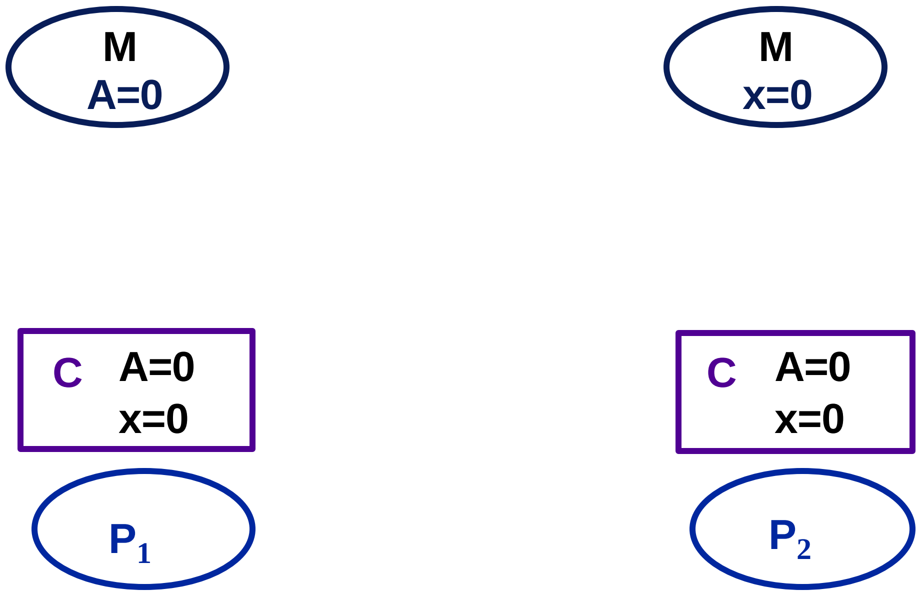
[This notion borrows from similar notions of sequential consistency in transaction processing systems.]

Coherent Cache Implementation

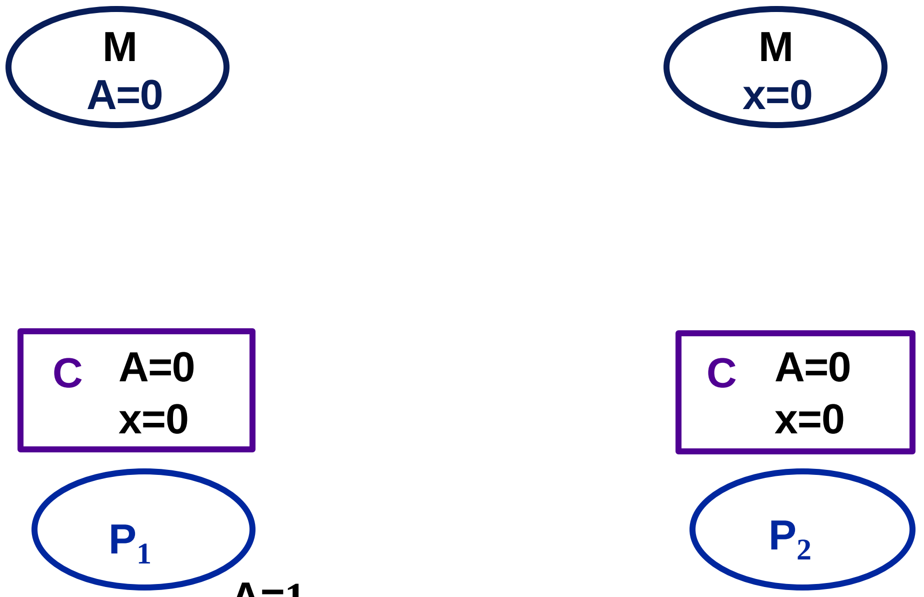
- **Twist:**
  - On write to shared location
    - Invalidation sent in background
    - Processor proceeds



Does caching violate this model?



Does caching violate this model?

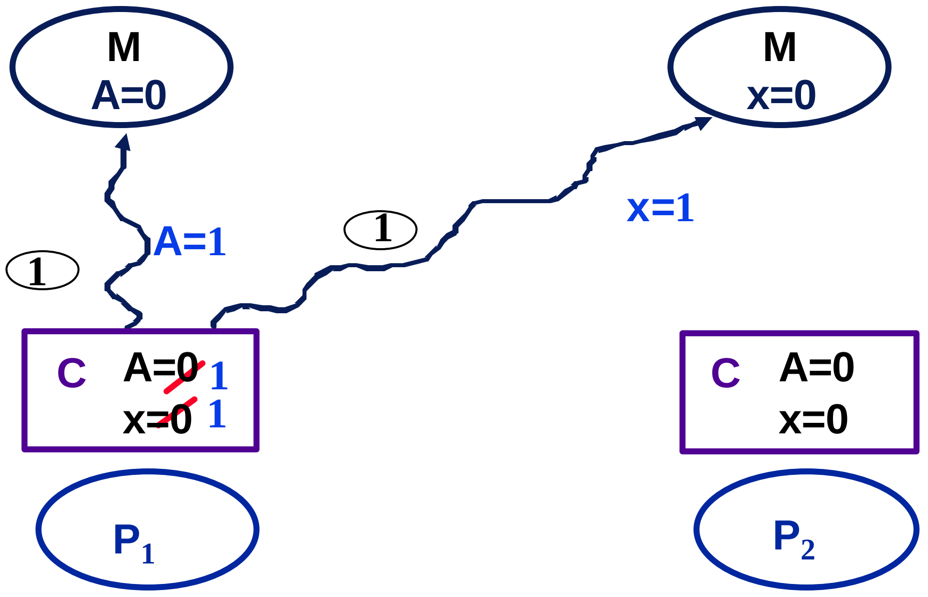


A=1
x=1

LOOP: If (x= = 0) GOTO LOOP;
b=A

If b = = 0 at the end, sequential consistency is violated

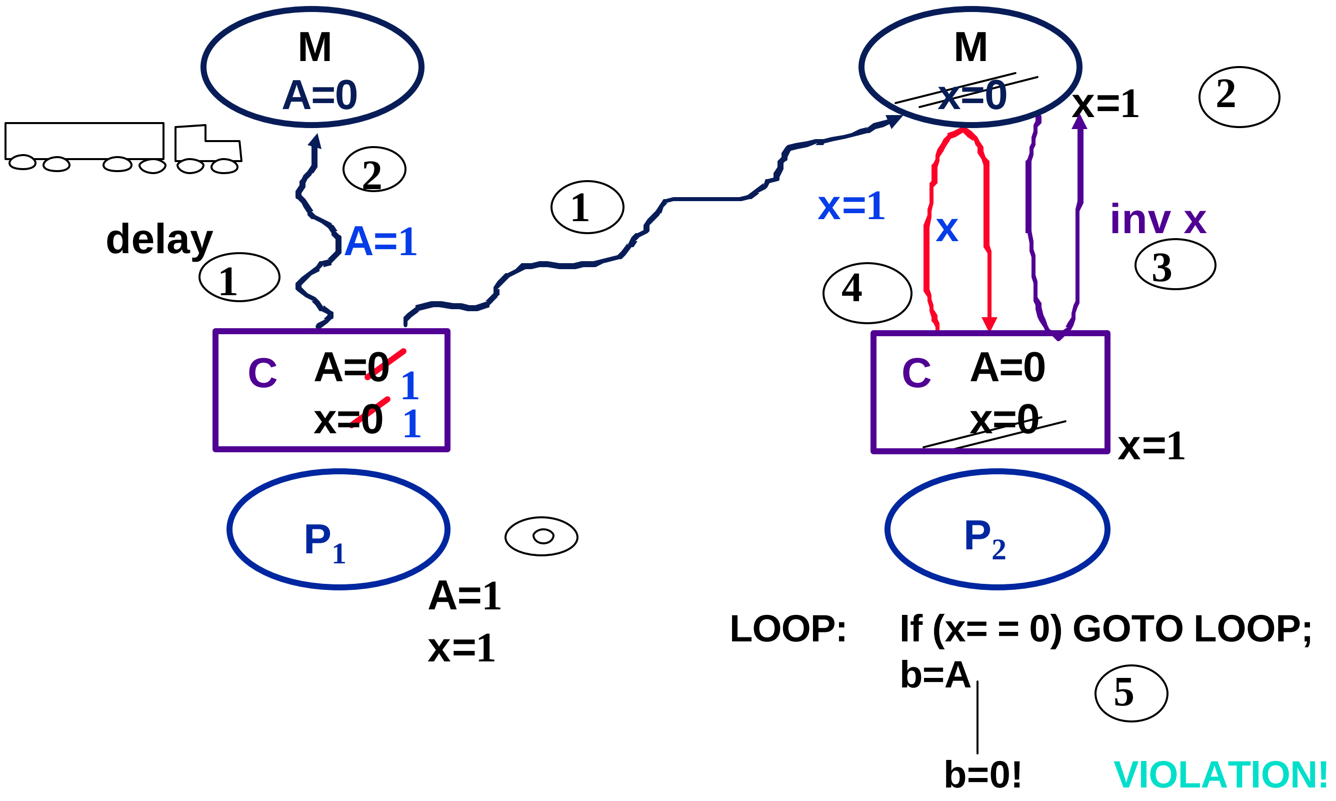
Does caching violate this model?



```
LOOP:    If (x= = 0) GOTO LOOP;
         b=A
```

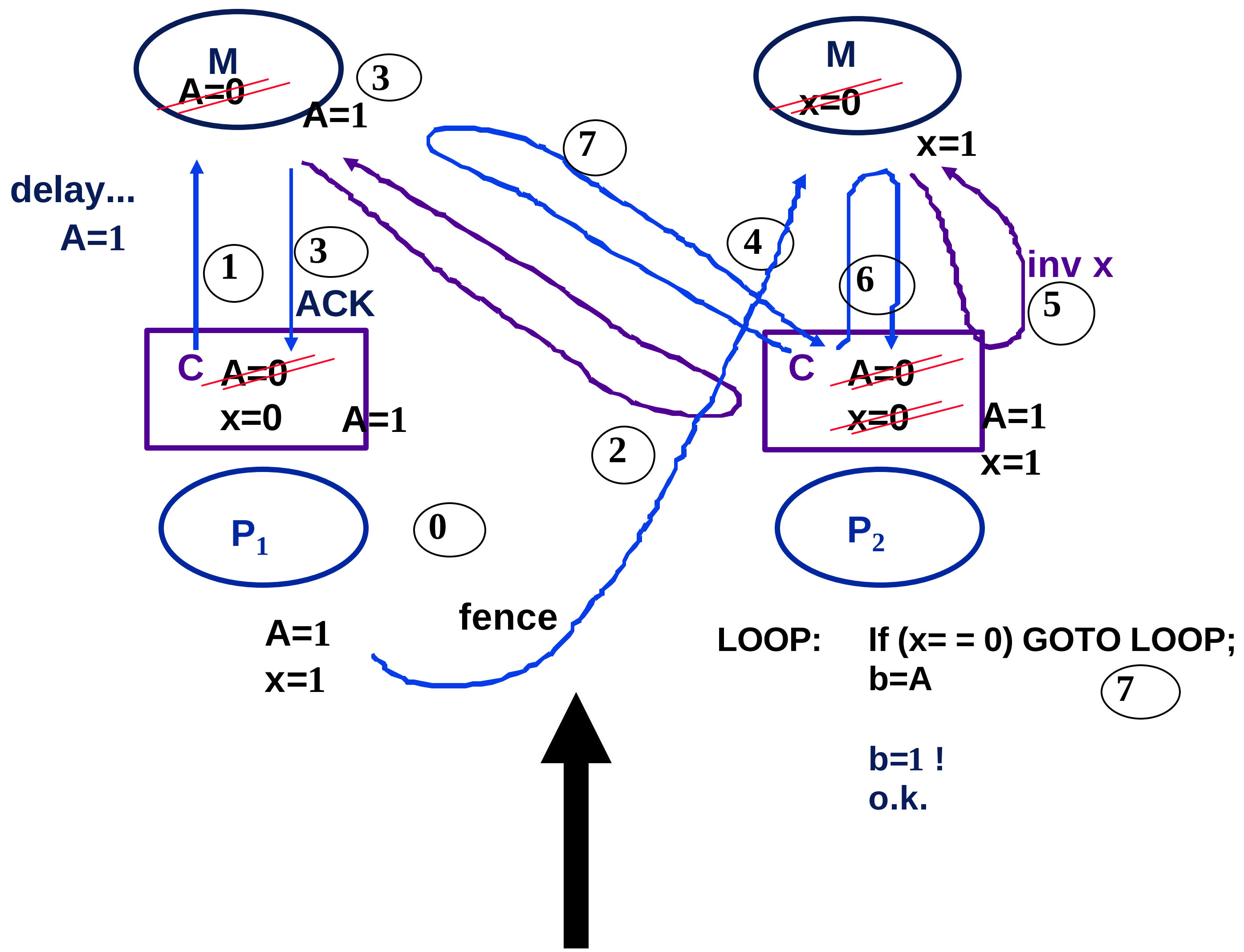
If b = = 0 at the end, sequential consistency is violated

Does caching violate this model?



If b = = 0 at the end, sequential consistency is violated

Does caching violate this model?



```
LOOP:   If (x= = 0) GOTO LOOP;
        b=A                    7

        b=1 !
        o.k.
```

Does caching violate this model?

- **Not if we are careful.**
  - **Ensure that at time instant *t*, no two processors see different values of a given variable.**
  - **On a write:**
    - **Lock datum**
    - **Invalidate all copies of datum**
    - **Update central copy of datum**
    - **Release lock on datum**
    - **Do not proceed till write completes (ack got)**
  - **How do we implement an update protocol?**
  - **Hard!**
    - **Lock central copy of datum**
    - **Mark all copies as unreadable**
    - **Update all copies --- release read lock on each copy after each update**
    - **Unlock central copy**

Writes are looooong -- latency ops.

- **Solutions -**

  1. **Build latency tolerant processors - Alewife**

see later

  2. **Change shared-memory semantics [solve a different problem!]**
  3. **Notion of weaker memory semantics**

     Basic idea - Guarantee completion of write only on "fence" operations

     Typical fence is synchronization point

     (or programmer puts fences in)

  Use:

  - **Modify shared data only within critical sections**
  - **Propagate changes at end of critical section, before releasing lock**

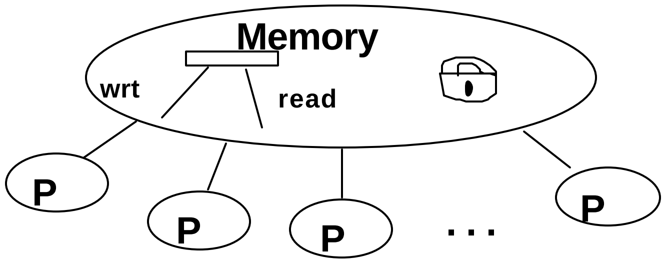
  Higher level locking protocols must guarantee that others do not try to read/write an object that has been modified and read by someone else.

  For most parallel programs -- no problem

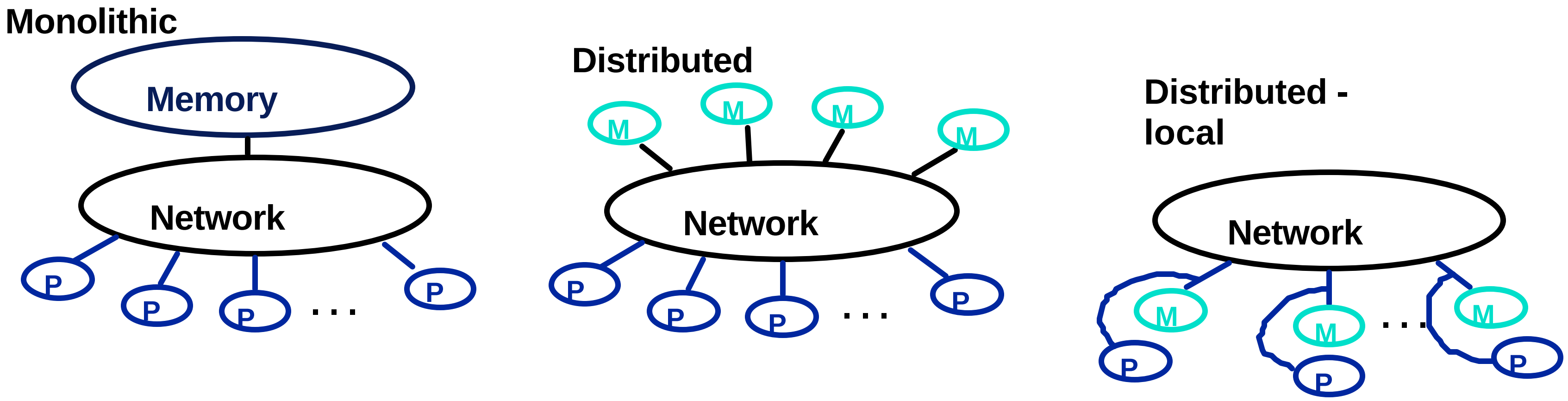
Memory Systems

- Memory storage
- Communication
- Processing

° **Programmer's view**

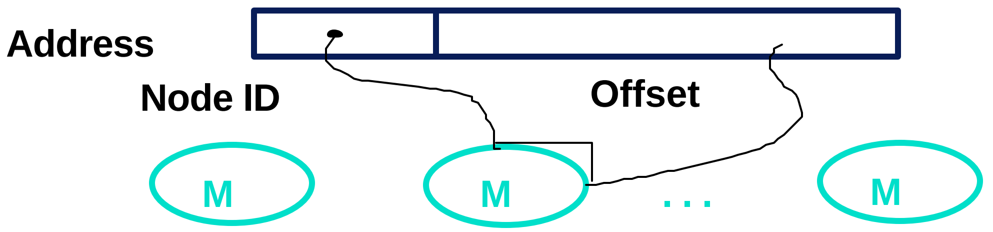


° **Physically,**



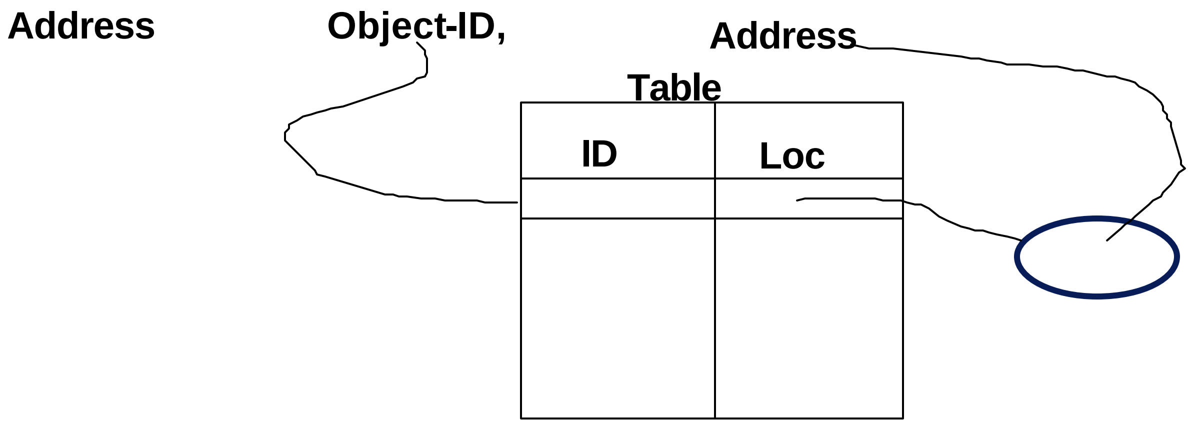
Addressing

- I. Like uniprocessors



Could include a translation phase for virtual memory systems

- II. Object-oriented models



Issues in virtual memory (also naming)

- **Goals:**
  - Illusion of a lot more memory than physically exists.
  - Protection - allows multiprogramming
  - Mobility of data: indirection allows ease of migration
- **Premise:**
  - Want a large, virtualized, single address space
  - But, physically distributed, local

Memory Performance Parameters

- Size (per node)
- Bandwidth (accesses per second)
- Latency (access time)

° **Size:**

- Issue of cost.

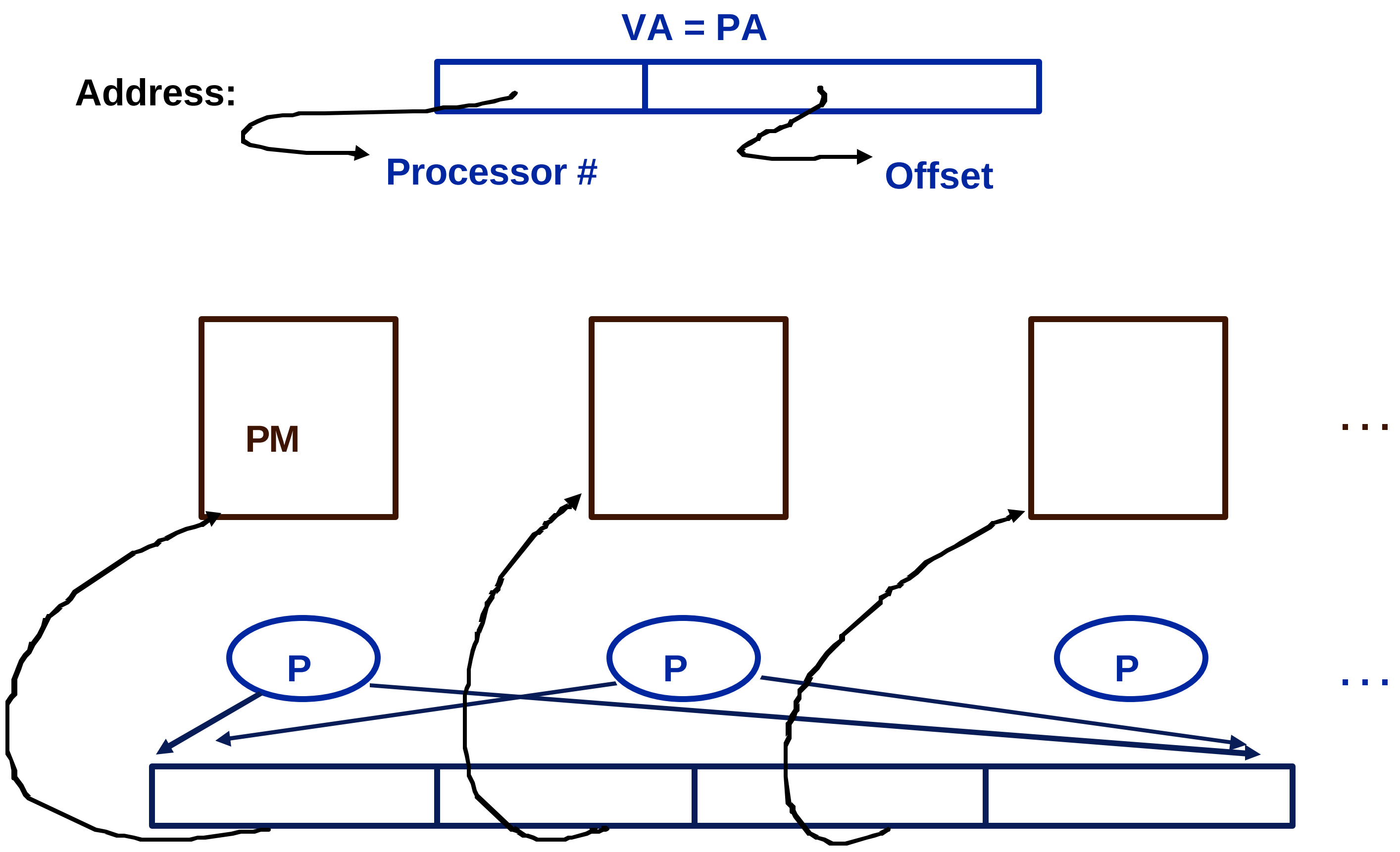
• Uniprocessors	1 MByte per MIPS
• Multiprossors?	Raging debate
Eg. Alewife	1/8 MByte memory per MIPS
Firefly	2 MByte per MIPS

What affects memory size decision?

Key issues Communication bandwidth
memory size tradeoffs

Balanced design --- All components roughly equally utilized

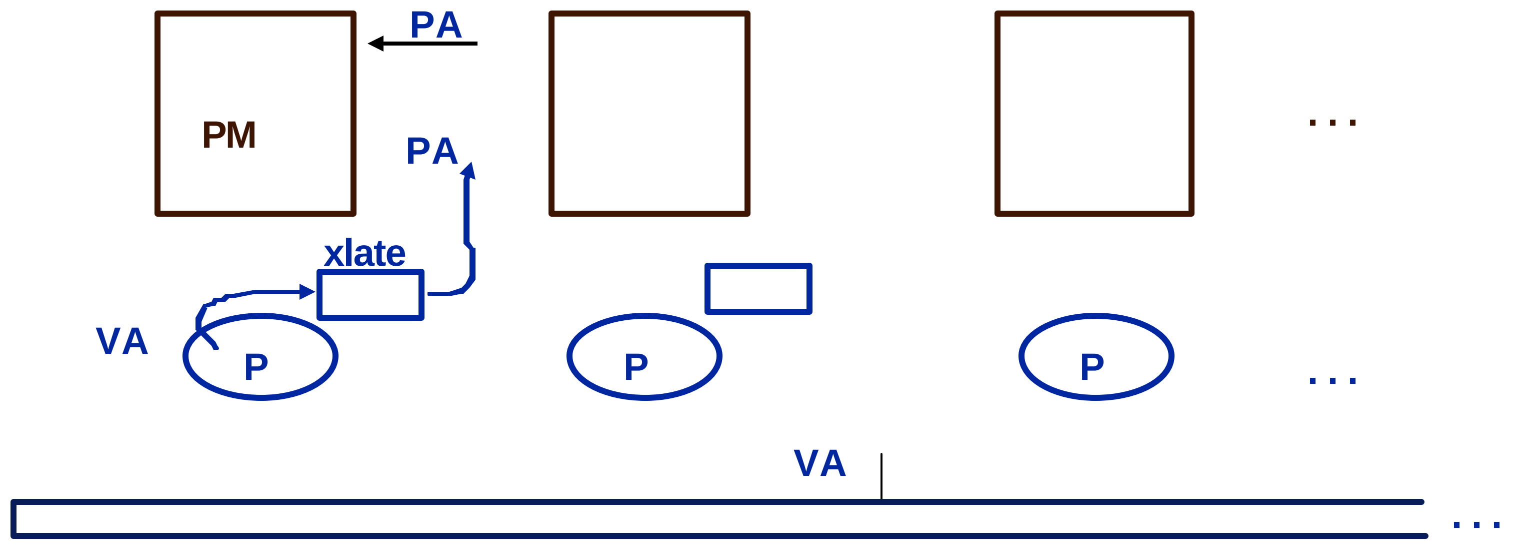
No VM



Relatively small address space

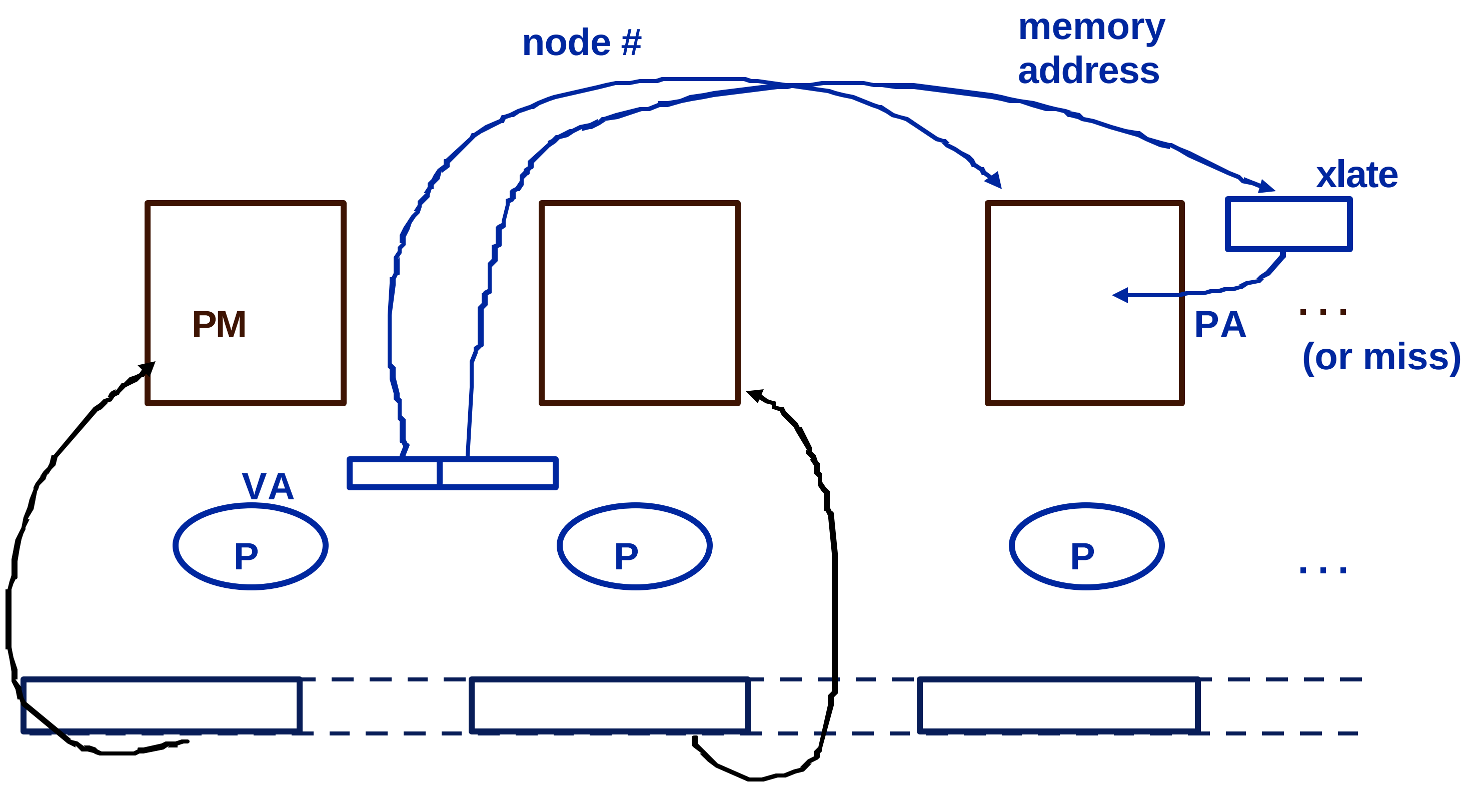
Virtual Memory

At source translation



- Large address space
- Straightforward extension from uniprocessors
- Xlate in software, in cache, or TLBs

VM – At Destination Translation



- On page fault at destination
  - Fetch page/obj from a local disk
  - Send msg to appropriate disk node

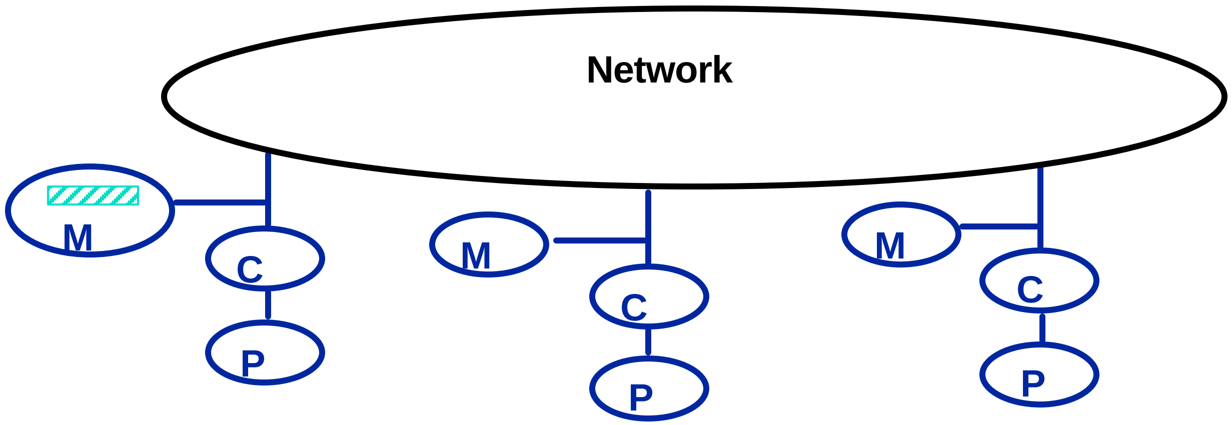
Next, bandwidth and latency

- In the interests of keeping the memory system as simple as possible, and because distributed memory provides high peak bandwidth, we will not consider interleaved memories as in vector processors
- Instead, look at
  - Reducing bandwidth demand of processors
  - Reducing latency of memory

  Exploit locality

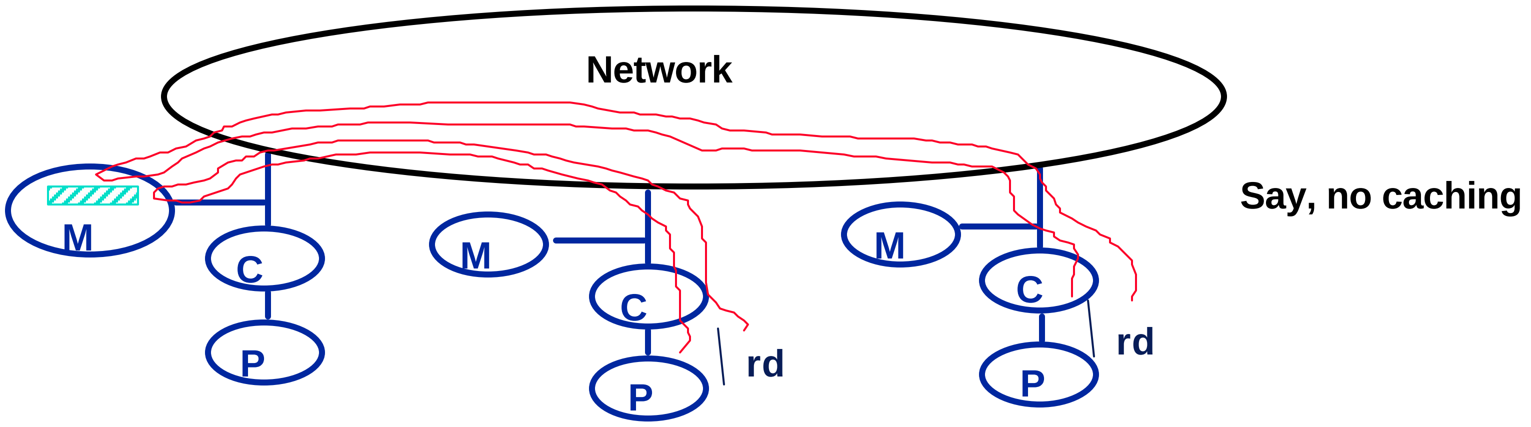
  Property of reuse
- Caches

Caching Techniques for multiprocessors



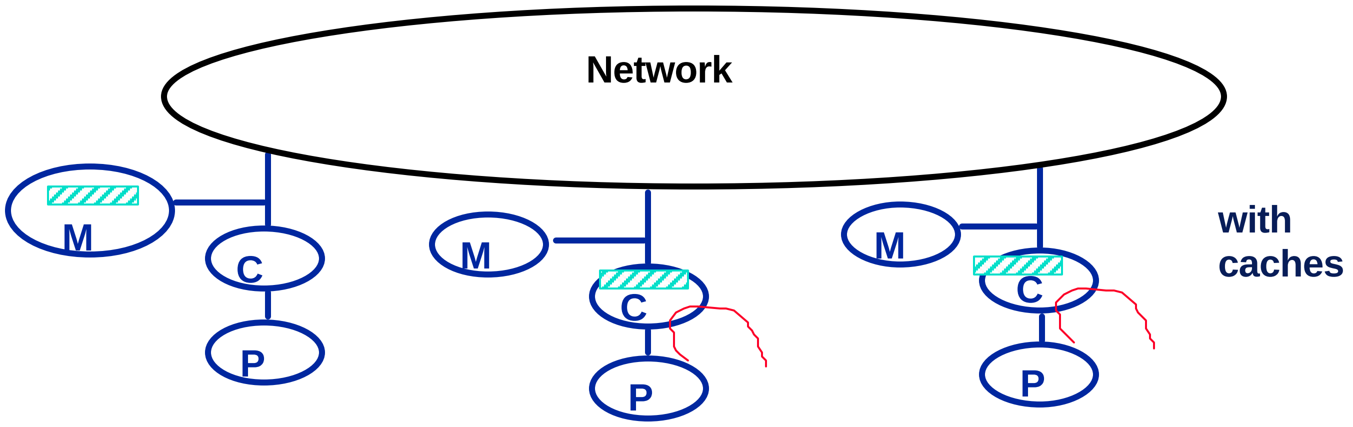
- How are caches different from local memory?
  - Fine-grain relocation of blocks
  - HW support for management, esp. for coherence
  - Smaller, faster, integrable
- Otherwise have similiar properties as local memory

Caching Techniques for multiprocessors



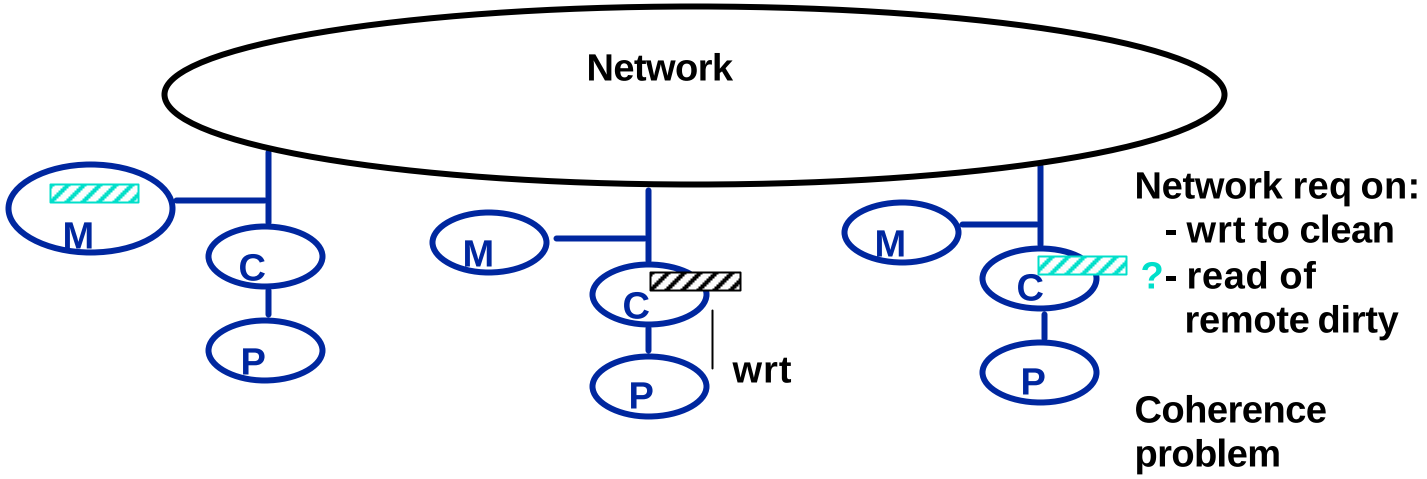
- How are caches different from local memory?
  - Fine-grain relocation of blocks
  - HW support for managment, esp. for coherence
  - Smaller, faster, integrable
- Otherwise have similiar properties as local memory

Caching Techniques for multiprocessors



- How are caches different from local memory?
  - Fine-grain relocation of blocks
  - HW support for managment, esp. for coherence
  - Smaller, faster, integrable
- Otherwise have similiar properties as local memory

Caching Techniques for multiprocessors



- How are caches different from local memory?
  - Fine-grain relocation of blocks
  - HW support for managment, esp. for coherence
  - Smaller, faster, integrable
- Otherwise have similiar properties as local memory

Summary

- Understand how delay affects cache performance
- Maintain sequential consistency
- Physcial properties
  - Consider topology and distribution of memory
- Develop an effective coherency strategy
- Simplicity and software maintenance are keys