CS 152 Computer Architecture and Engineering

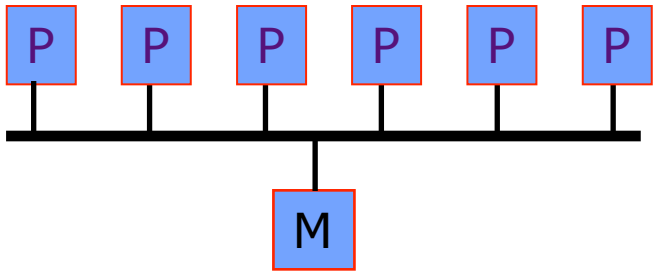
Lecture 20: Snoopy Caches

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.cs.berkeley.edu/~cs152

Recap: Sequential Consistency

A Memory Model



" A system is *sequentially consistent* if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program"

Leslie Lamport

Sequential Consistency =

arbitrary *order-preserving interleaving*
of memory references of sequential programs

Recap: Sequential Consistency

Sequential consistency imposes more memory ordering constraints than those imposed by uniprocessor program dependencies (⟶)

What are these in our example ?

T1:

Store (X), 1 *(X = 1)*

Store (Y), 11 *(Y = 11)*

T2:

Load R_1, (Y)

Store (Y'), R_1 *(Y'= Y)*

Load R_2, (X)

Store (X'), R_2 *(X'= X)*

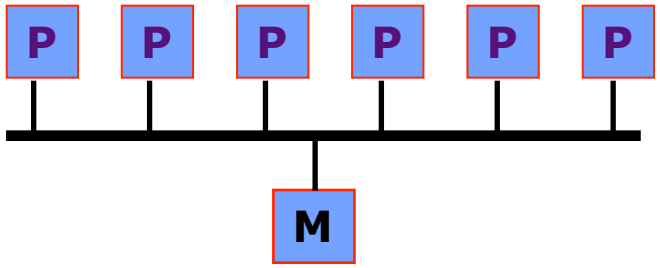
⟶ additional SC requirements

Recap: Mutual Exclusion and Locks

Want to guarantee only one process is active in a critical section

- Blocking atomic read-modify-write instructions
  e.g., Test&Set, Fetch&Add, Swap
  vs
- Non-blocking atomic read-modify-write instructions
  e.g., Compare&Swap, Load-reserve/Store-conditional
  vs
- Protocols based on ordinary Loads and Stores

Issues in Implementing Sequential Consistency



Implementation of SC is complicated by two issues

- *Out-of-order execution capability*

Load(a); Load(b)	*yes*
Load(a); Store(b)	*yes if* a ≠ b
Store(a); Load(b)	*yes if* a ≠ b
Store(a); Store(b)	*yes if* a ≠ b

- *Caches*

  Caches can prevent the effect of a store from being seen by other processors

SC complications motivate architects to consider *weak* or *relaxed* memory models

Memory Fences

Instructions to sequentialize memory accesses

Processors with *relaxed or weak memory models* (i.e., permit Loads and Stores to different addresses to be reordered) need to provide *memory fence* instructions to force the serialization of memory accesses

Examples of processors with relaxed memory models:

Sparc V8 (TSO,PSO): Membar

Sparc V9 (RMO):

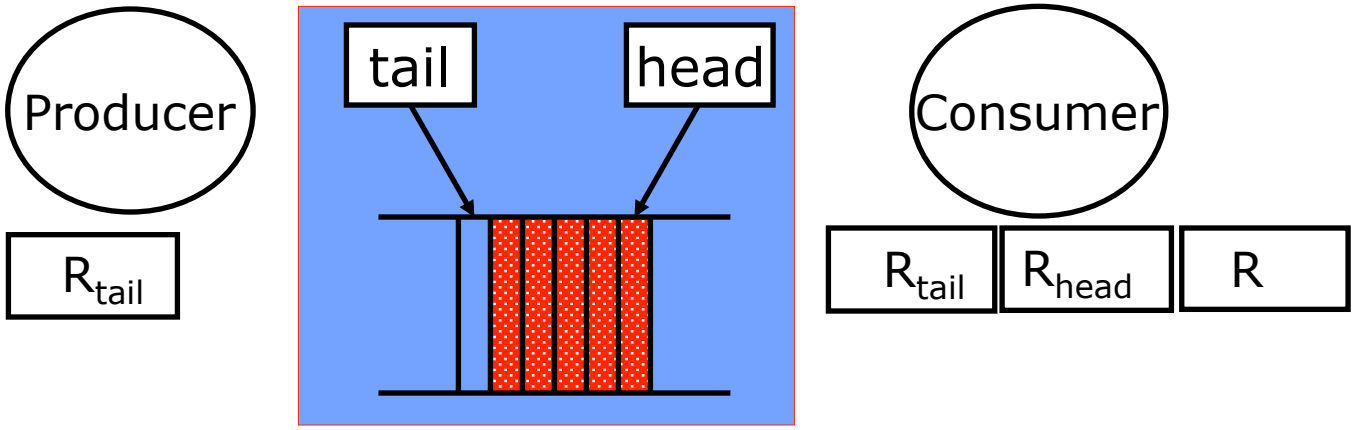
Membar #LoadLoad, Membar #LoadStore

Membar #StoreLoad, Membar #StoreStore

PowerPC (WO): Sync, EIEIO

Memory fences are expensive operations, however, one pays the cost of serialization only when it is required

Using Memory Fences



Producer posting Item x:

```
Load R_tail, (tail)
Store (R_tail), x
Membar_SS
R_tail=R_tail+1
Store (tail), R_tail
```

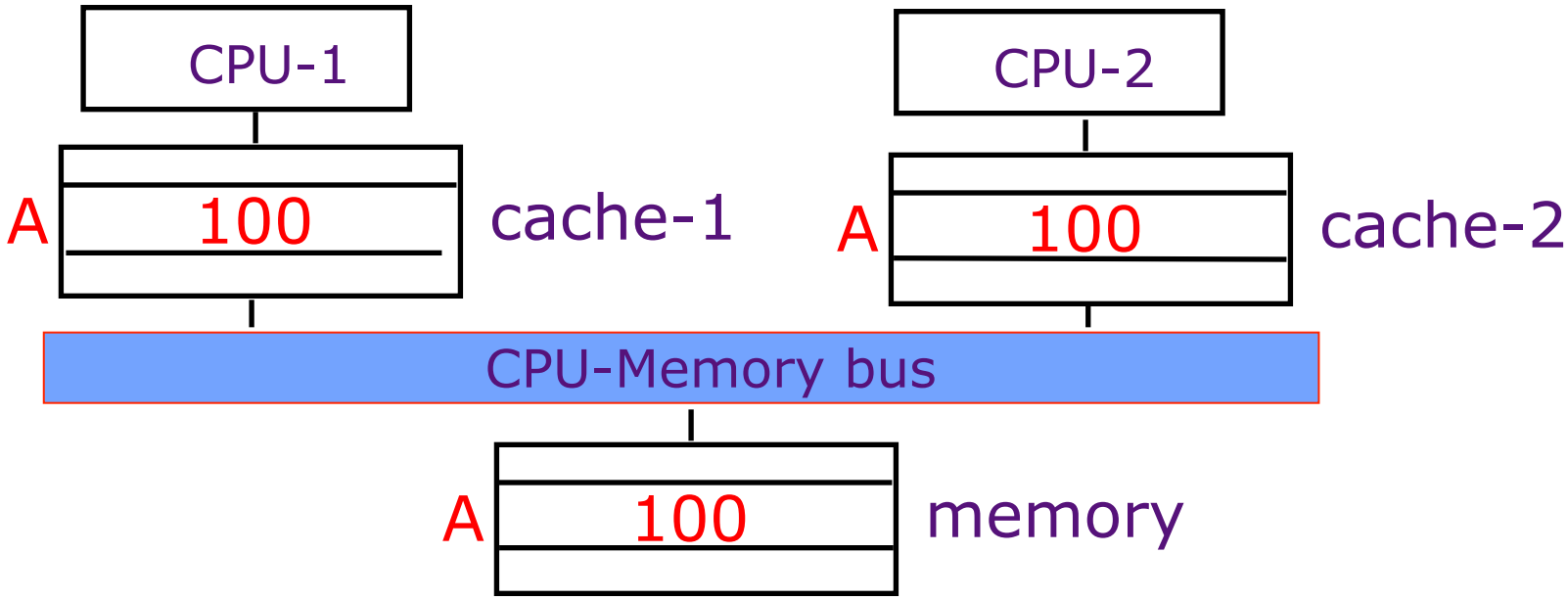
ensures that tail ptr is not updated before x has been stored

Consumer:

```
        Load R_head, (head)
spin:   Load R_tail, (tail)
        if R_head==R_tail goto spin
        Membar_LL
        Load R, (R_head)
        R_head=R_head+1
        Store (head), R_head
        process(R)
```

ensures that R is not loaded before x has been stored

Memory Coherence in SMPs



Suppose CPU-1 updates A to 200.

write-back: memory and cache-2 have stale values

write-through: cache-2 has a stale value

Do these stale values matter?

What is the view of shared memory for programming?

Write-back Caches & SC

- T1 is executed
- cache-1 writes back Y
- T2 executed
- cache-1 writes back X
- cache-2 writes back X′ & Y′

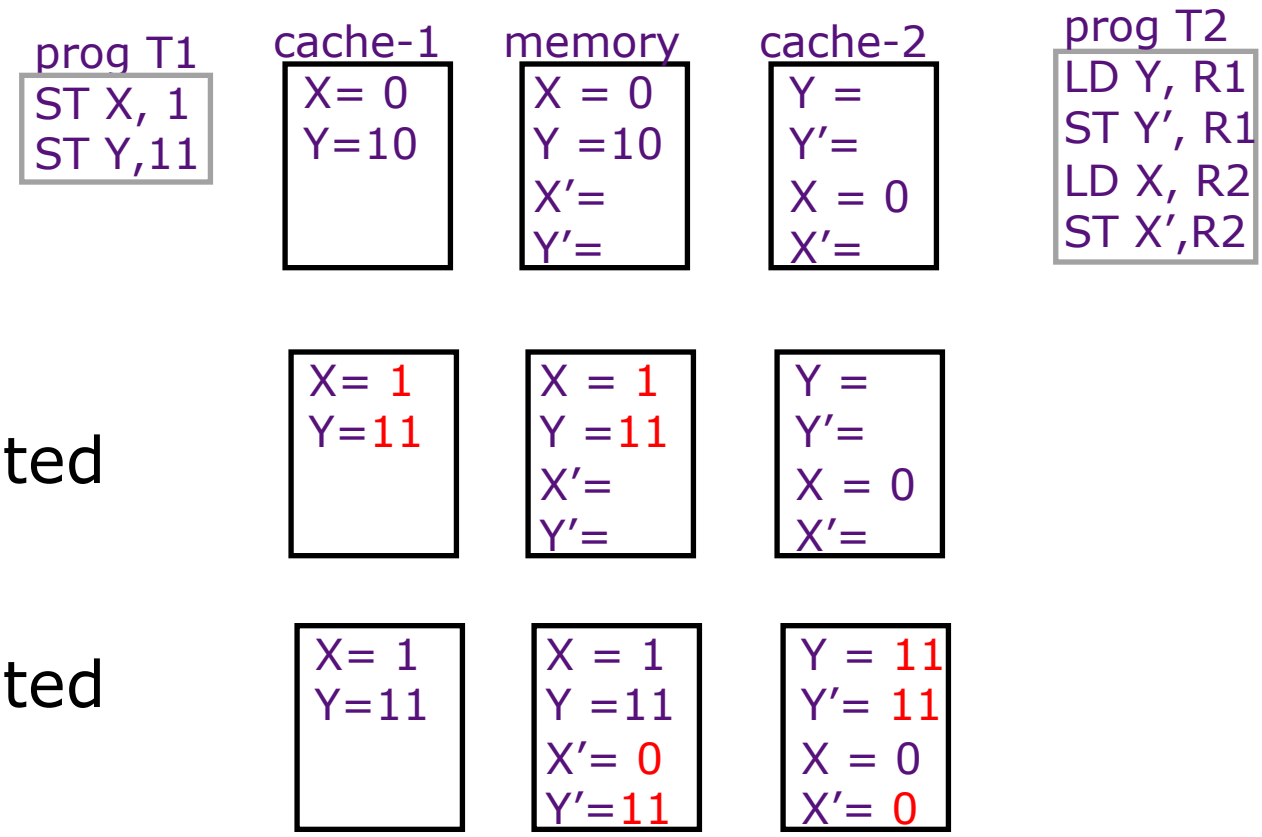
prog T1	cache-1	memory	cache-2	prog T2

ST X, 1
ST Y,11
	X= 1
	Y=11
		X = 0
		Y =10
		X′=
		Y′=
			Y =
			Y′=
			X =
			X′=
				LD Y, R1
				ST Y′, R1
				LD X, R2
				ST X′,R2

	X= 1 Y=11	X = 0 Y =11 X′= Y′=	Y = Y′= X = X′=	
	X= 1 Y=11	X = 0 Y =11 X′= Y′=	Y = 11 Y′= 11 X = 0 X′= 0	
	X= 1 Y=11	X = 1 Y =11 X′= Y′=	Y = 11 Y′= 11 X = 0 X′= 0	
	X= 1 Y=11	X = 1 Y =11 X′= 0 Y′=11	Y =11 Y′=11 X = 0 X′= 0	

inconsistent

Write-through Caches & SC



- T1 executed
- T2 executed

Write-through caches don't preserve sequential consistency either

Cache Coherence vs. Memory Consistency

- A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors
  - i.e., updates are not lost
- A memory consistency model gives the rules on when a write by one processor can be observed by a read on another
  - Equivalently, what values can be seen by a load
- A cache coherence protocol is not enough to ensure sequential consistency
  - But if sequentially consistent, then caches must be coherent
- Combination of cache coherence protocol plus processor memory reorder buffer implements a given machine's memory consistency model

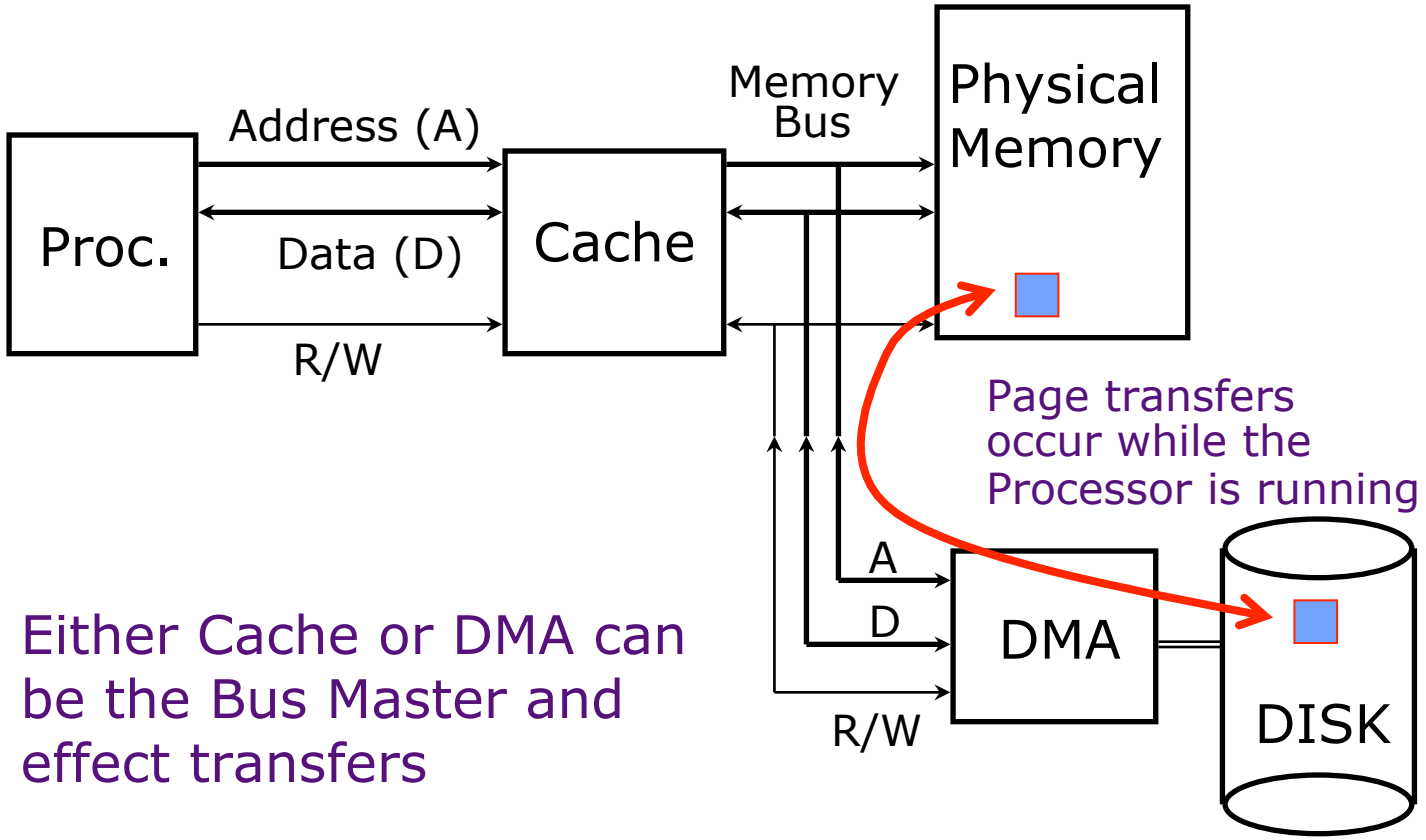
Maintaining Cache Coherence

Hardware support is required such that

- only one processor at a time has write permission for a location
- no processor can load a stale copy of the location after a write

$\Rightarrow$ *cache coherence protocols*

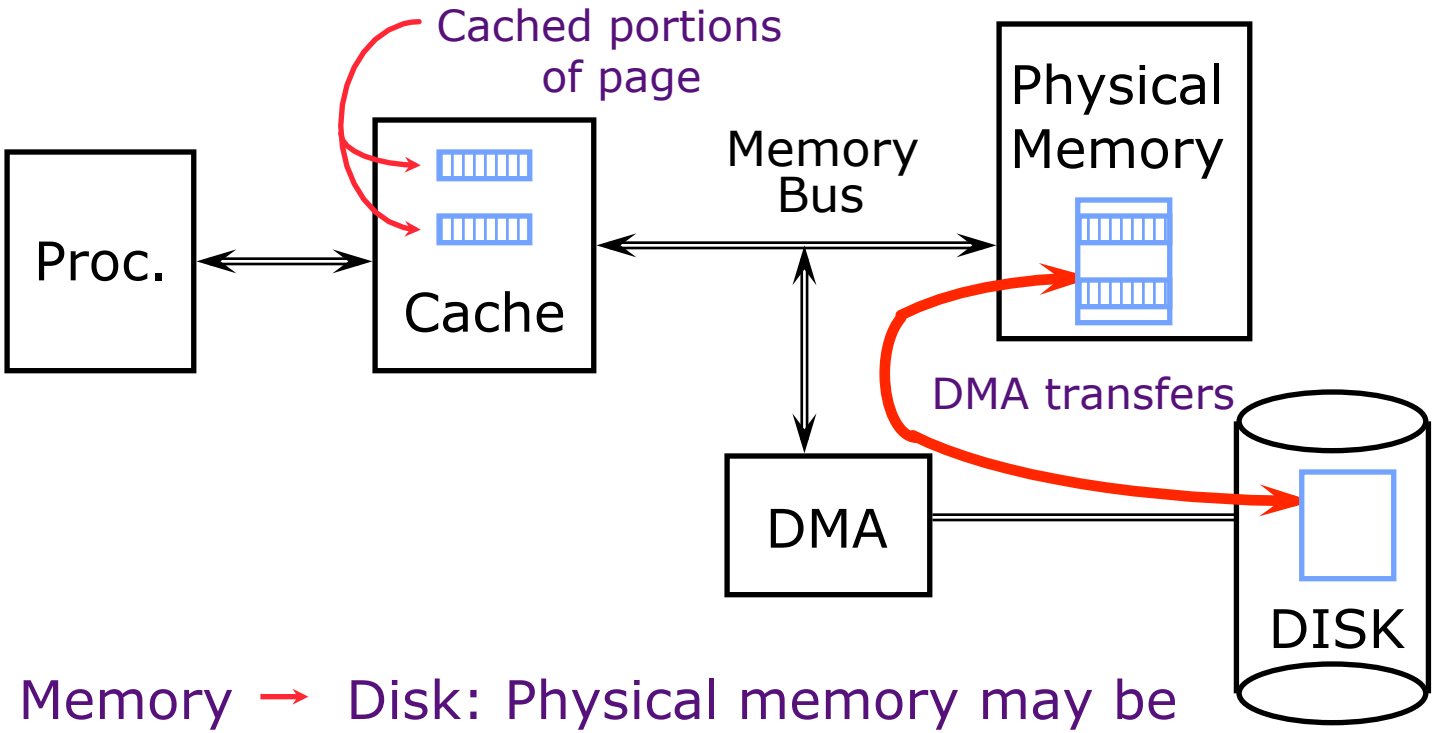
Warmup: Parallel I/O



Either Cache or DMA can be the Bus Master and effect transfers

(DMA stands for Direct Memory Access, means the I/O device can read/write memory autonomous from the CPU)

Problems with Parallel I/O

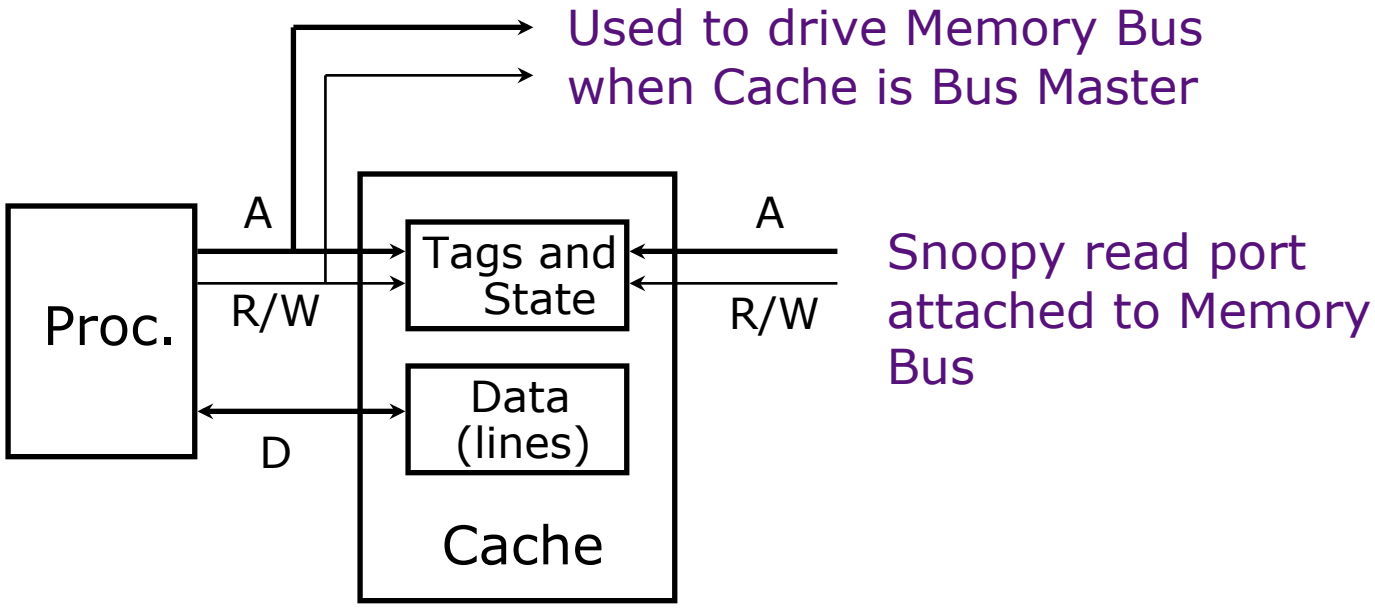


Memory → Disk: Physical memory may be stale if cache copy is dirty

Disk → Memory: Cache may hold stale data and not see memory writes

Snoopy Cache *Goodman 1983*

- Idea: Have cache watch (or snoop upon) DMA transfers, and then "do the right thing"
- Snoopy cache tags are dual-ported

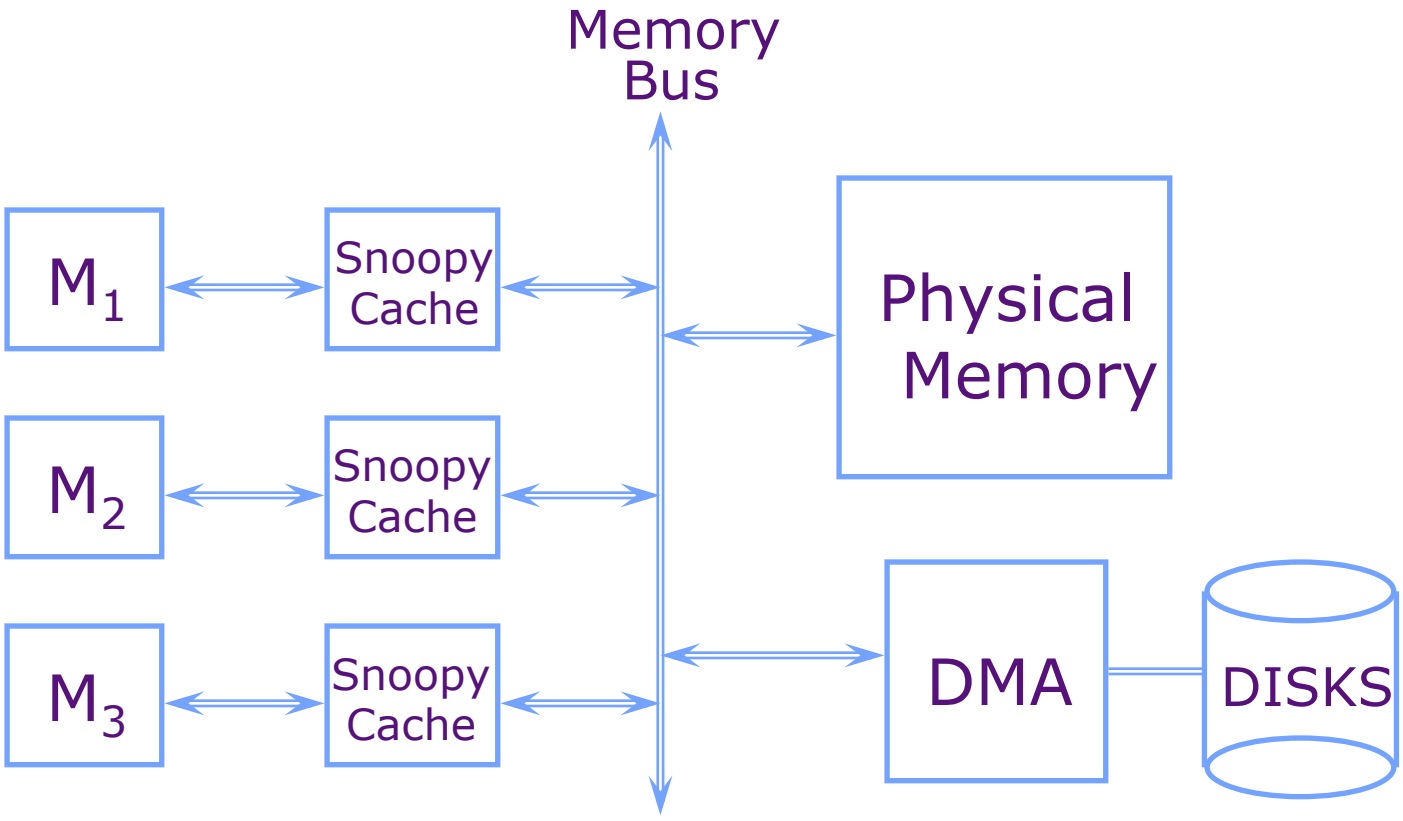


Snoopy Cache Actions for DMA

Observed Bus Cycle	Cache State	Cache Action
DMA Read Memory → Disk	Address not cached	No action
	Cached, unmodified	No action
	Cached, modified	Cache intervenes
DMA Write Disk → Memory	Address not cached	No action
	Cached, unmodified	Cache purges its copy
	Cached, modified	???

CS152 Administrivia

Shared Memory Multiprocessor



Use snoopy mechanism to keep all processors' view of memory coherent

Snoopy Cache Coherence Protocols

write miss:

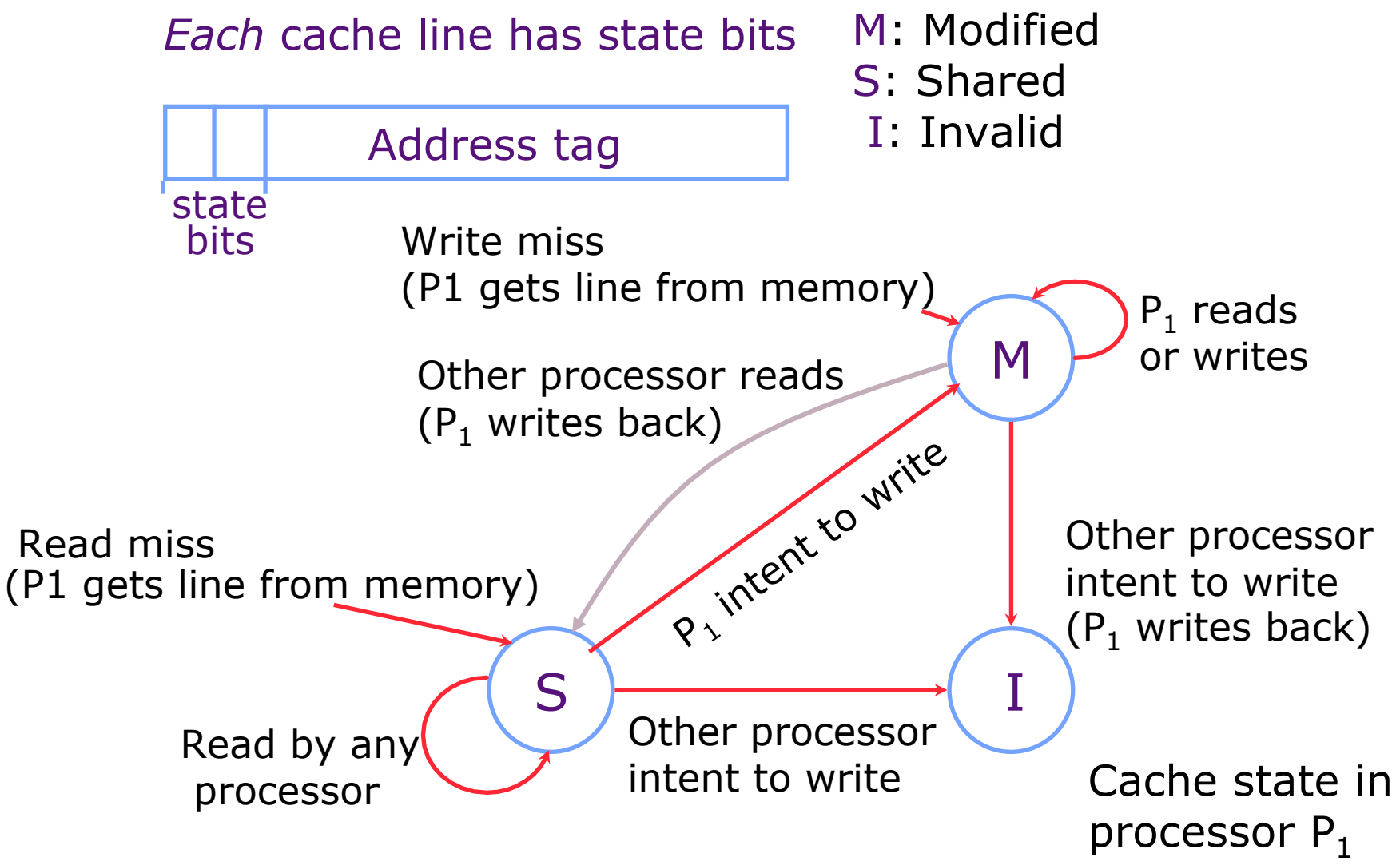
the address is *invalidated* in all other caches *before* the write is performed

read miss:

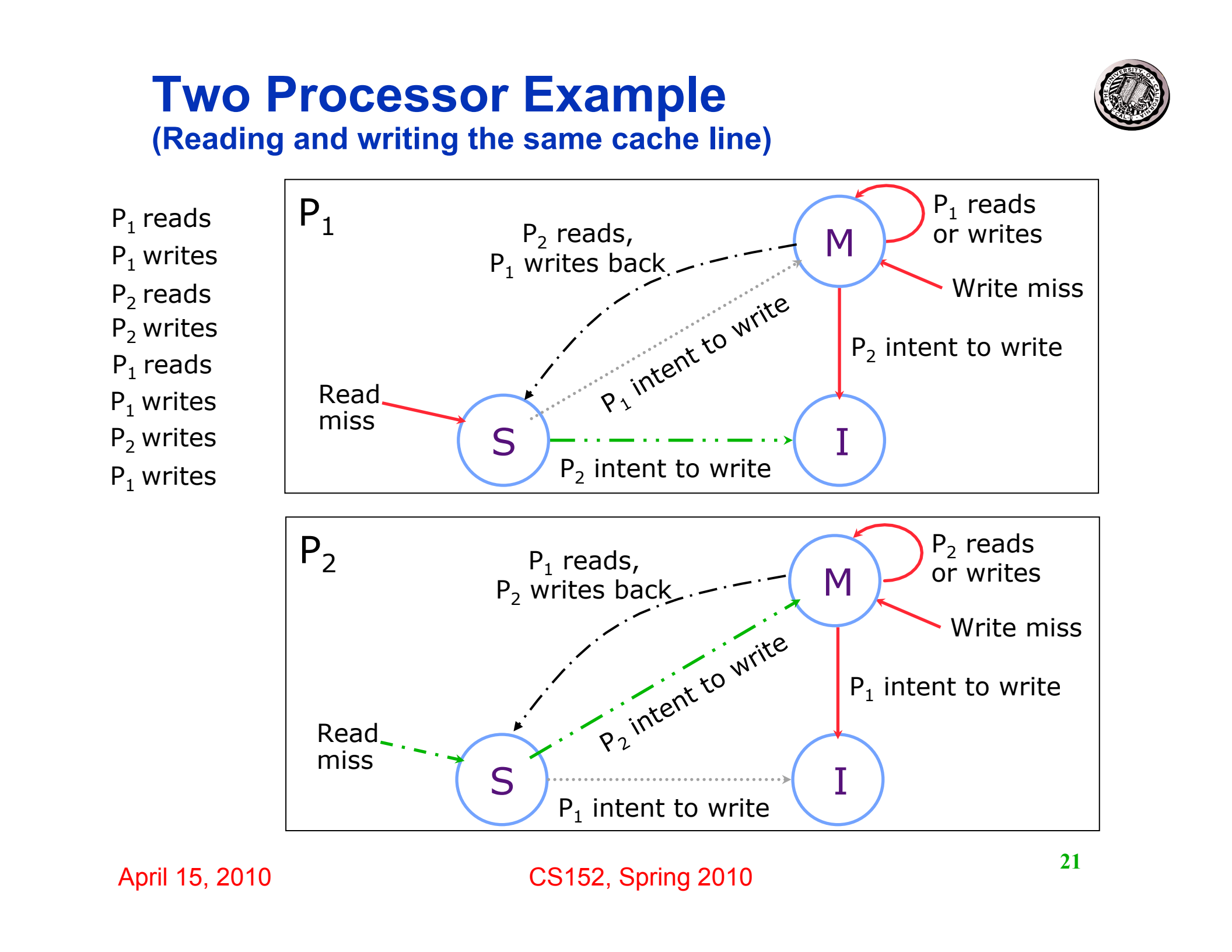
if a dirty copy is found in some cache, a write-back is performed before the memory is read

Cache State Transition Diagram

The MSI protocol

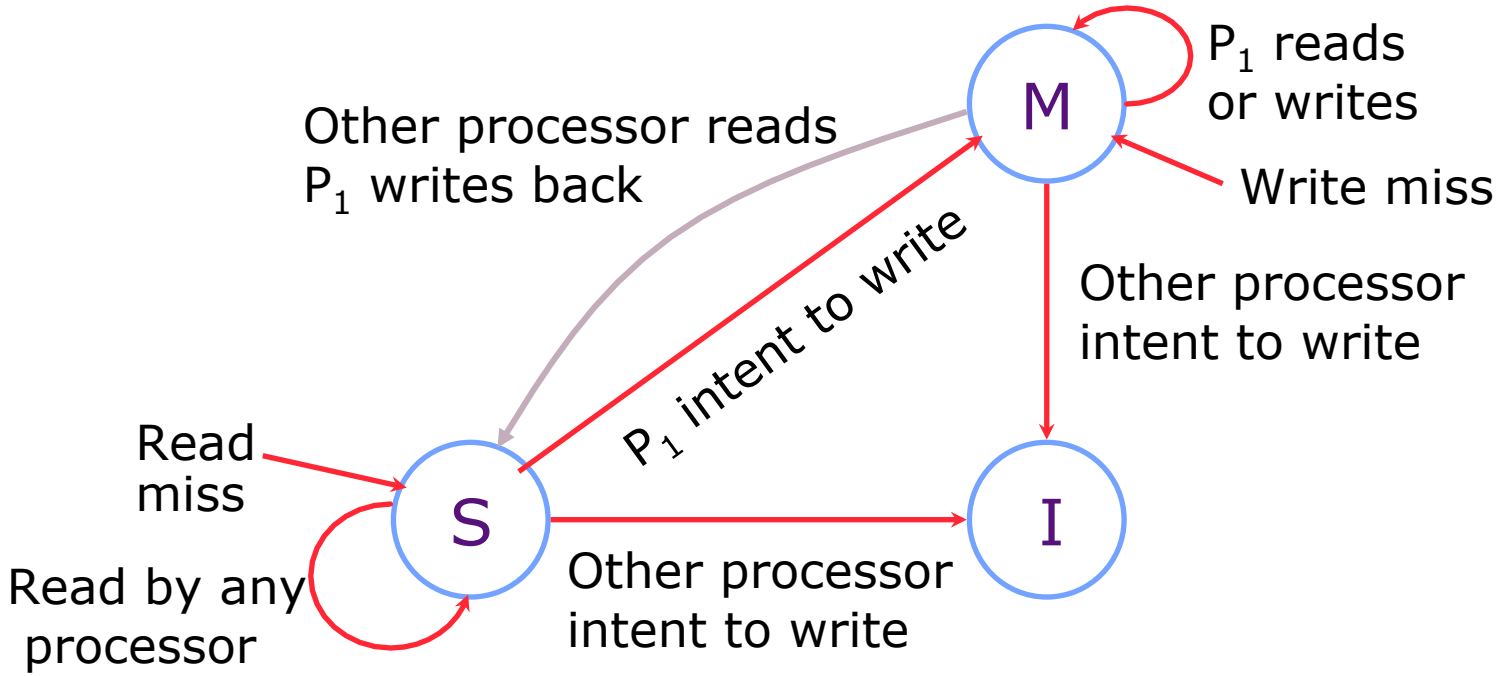


Two Processor Example
(Reading and writing the same cache line)



P_1 reads
P_1 writes
P_2 reads
P_2 writes
P_1 reads
P_1 writes
P_2 writes
P_1 writes

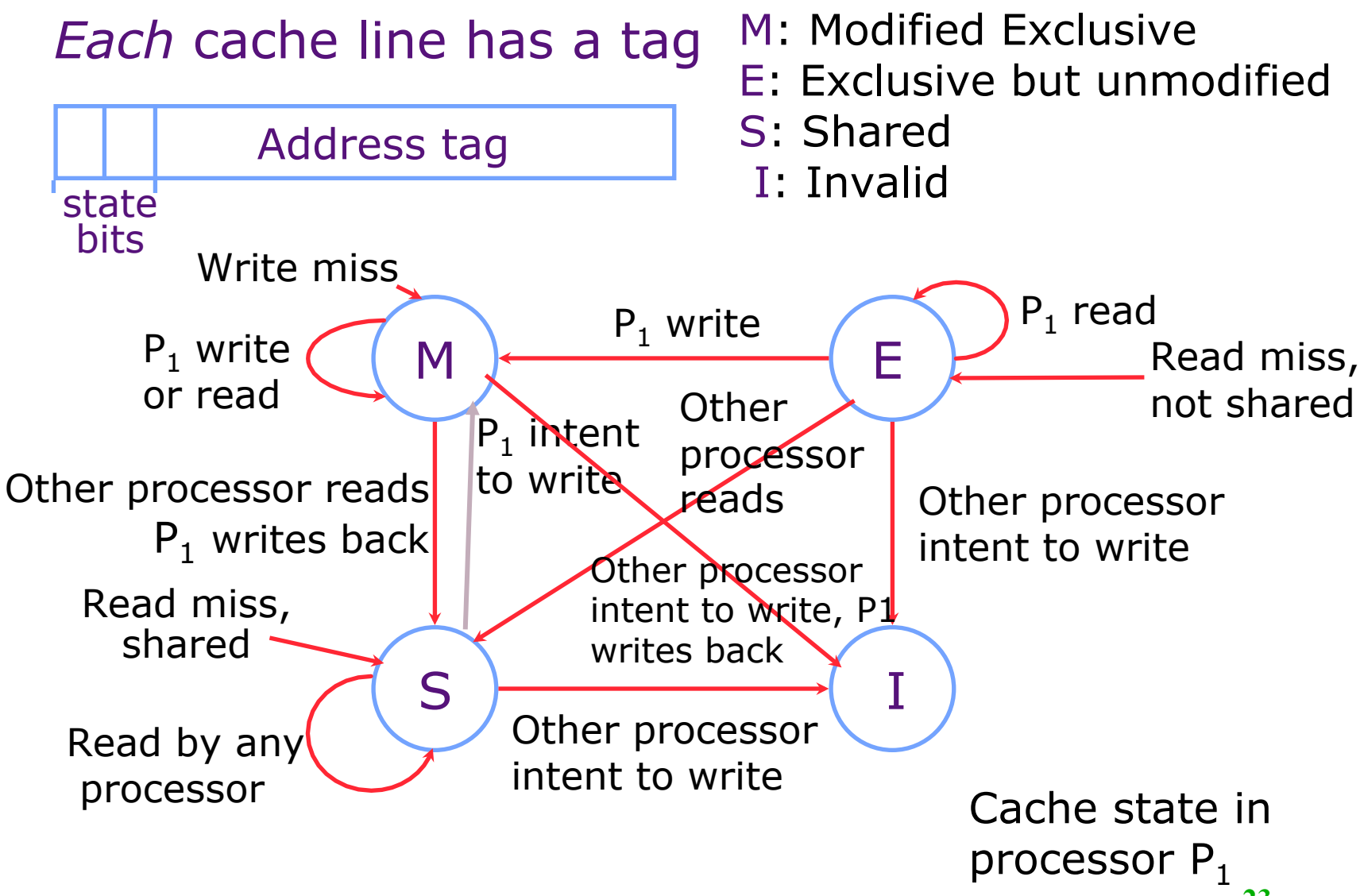
Observation



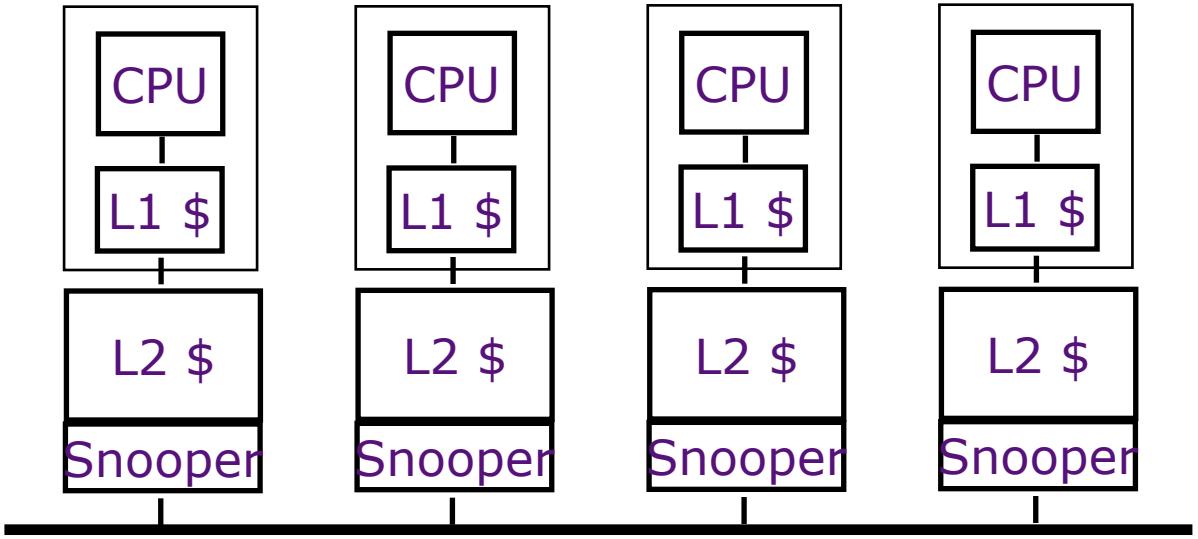
- If a line is in the M state then no other cache can have a copy of the line!
  - Memory stays coherent, multiple differing copies cannot exist

MESI: An Enhanced MSI protocol

increased performance for private data



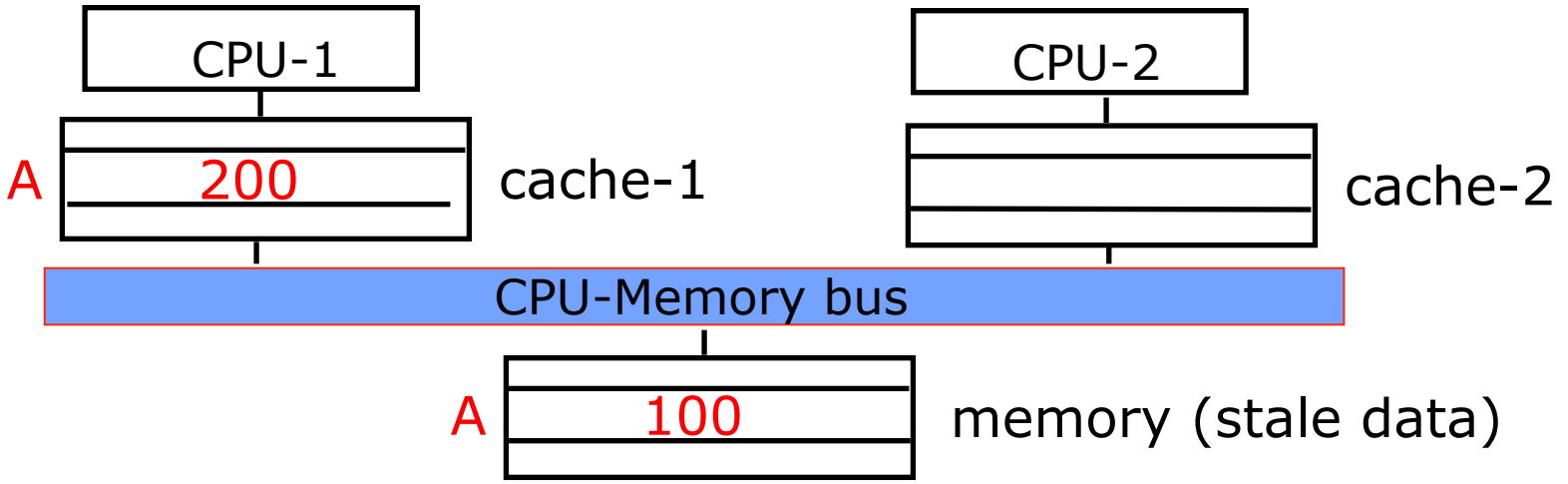
Optimized Snoop with Level-2 Caches



- Processors often have two-level caches
  - small L1, large L2 (usually both on chip now)
- *Inclusion property:* entries in L1 must be in L2
  invalidation in L2 ⇒ invalidation in L1
- Snooping on L2 does not affect CPU-L1 bandwidth

What problem could occur?

Intervention



When a read-miss for A occurs in cache-2, a read request for A is placed on the bus

- Cache-1 needs to supply & change its state to shared
- The memory may respond to the request also!

Does memory know it has stale data?

Cache-1 needs to intervene through memory controller to supply correct data to cache-2

False Sharing

state	blk addr	data0	data1	...	dataN

A cache block contains more than one word

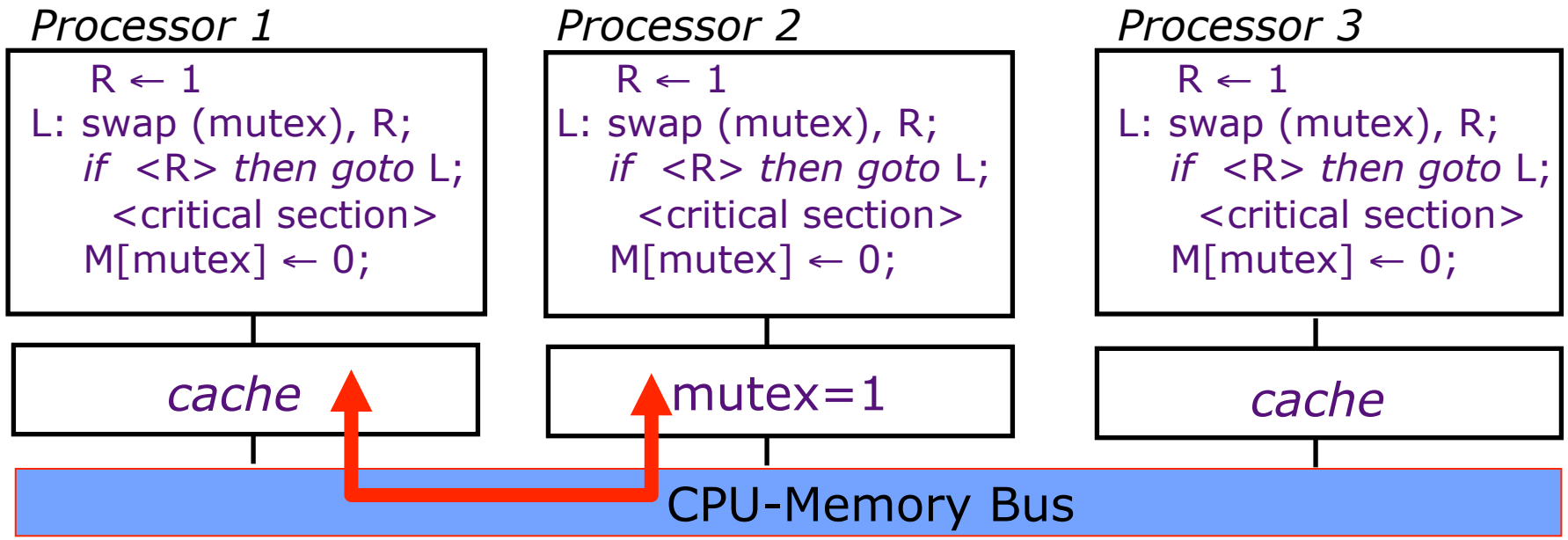
Cache-coherence is done at the block-level and not word-level

Suppose M_1 writes $word_i$ and M_2 writes $word_k$ and both words have the same block address.

What can happen?

Synchronization and Caches:

Performance Issues



Cache-coherence protocols will cause mutex to *ping-pong* between P1's and P2's caches.

Ping-ponging can be reduced by first reading the mutex location *(non-atomically)* and executing a swap only if it is found to be zero.

Load-reserve & Store-conditional

Special register(s) to hold reservation flag and address, and the outcome of store-conditional

```
Load-reserve R, (a):
    <flag, adr> ← <1, a>;
    R ← M[a];
```

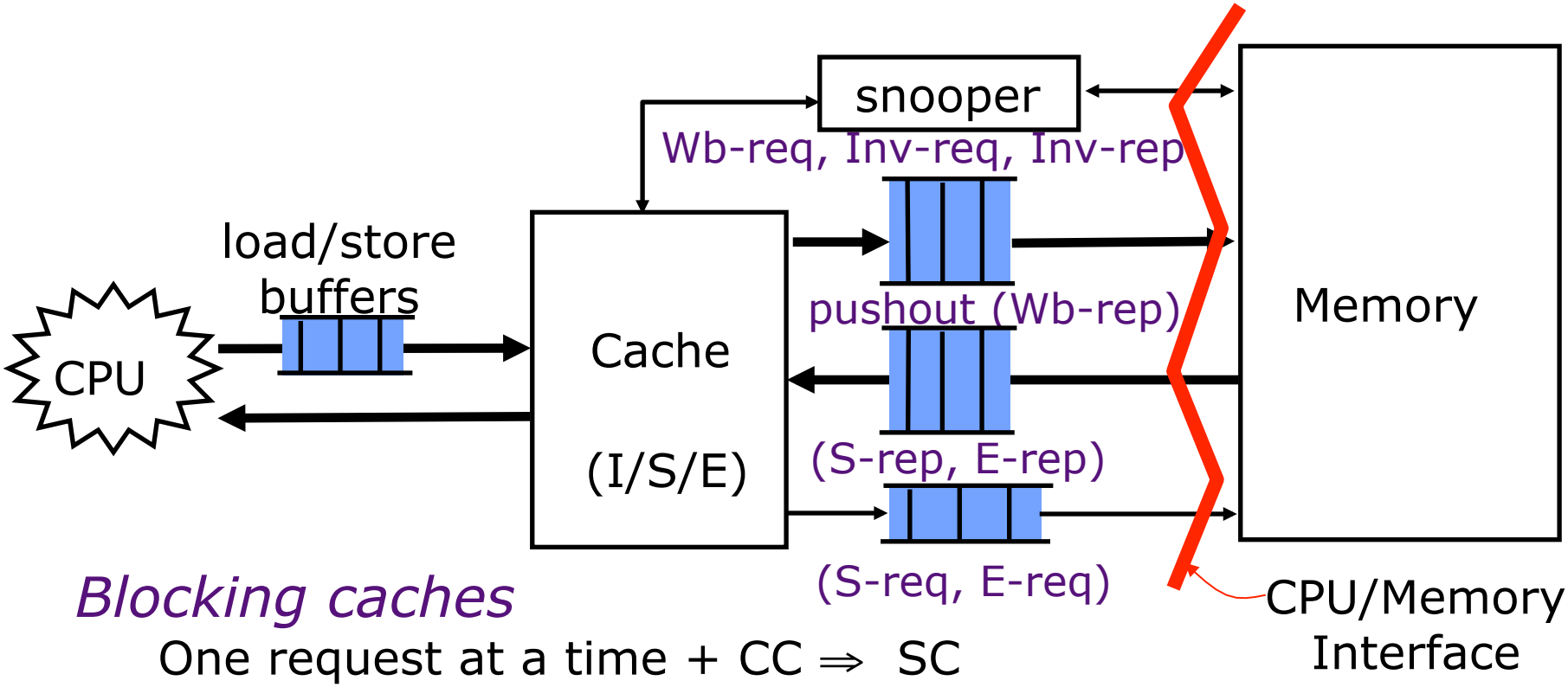
```
Store-conditional (a), R:
    if <flag, adr> == <1, a>
    then  cancel other procs'
          reservation on a;
          M[a] ← <R>;
          status ← succeed;
    else  status ← fail;
```

If the snooper sees a store transaction to the address in the reserve register, the reserve bit is set to 0

- Several processors may reserve 'a' simultaneously
- These instructions are like ordinary loads and stores with respect to the bus traffic

Can implement reservation by using cache hit/miss, no additional hardware required (problems?)

Out-of-Order Loads/Stores & CC



Blocking caches

One request at a time + CC ⇒ SC

Non-blocking caches

Multiple requests (different addresses) concurrently + CC

⇒ Relaxed memory models

CC ensures that all processors observe the same order of loads and stores to an address

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252