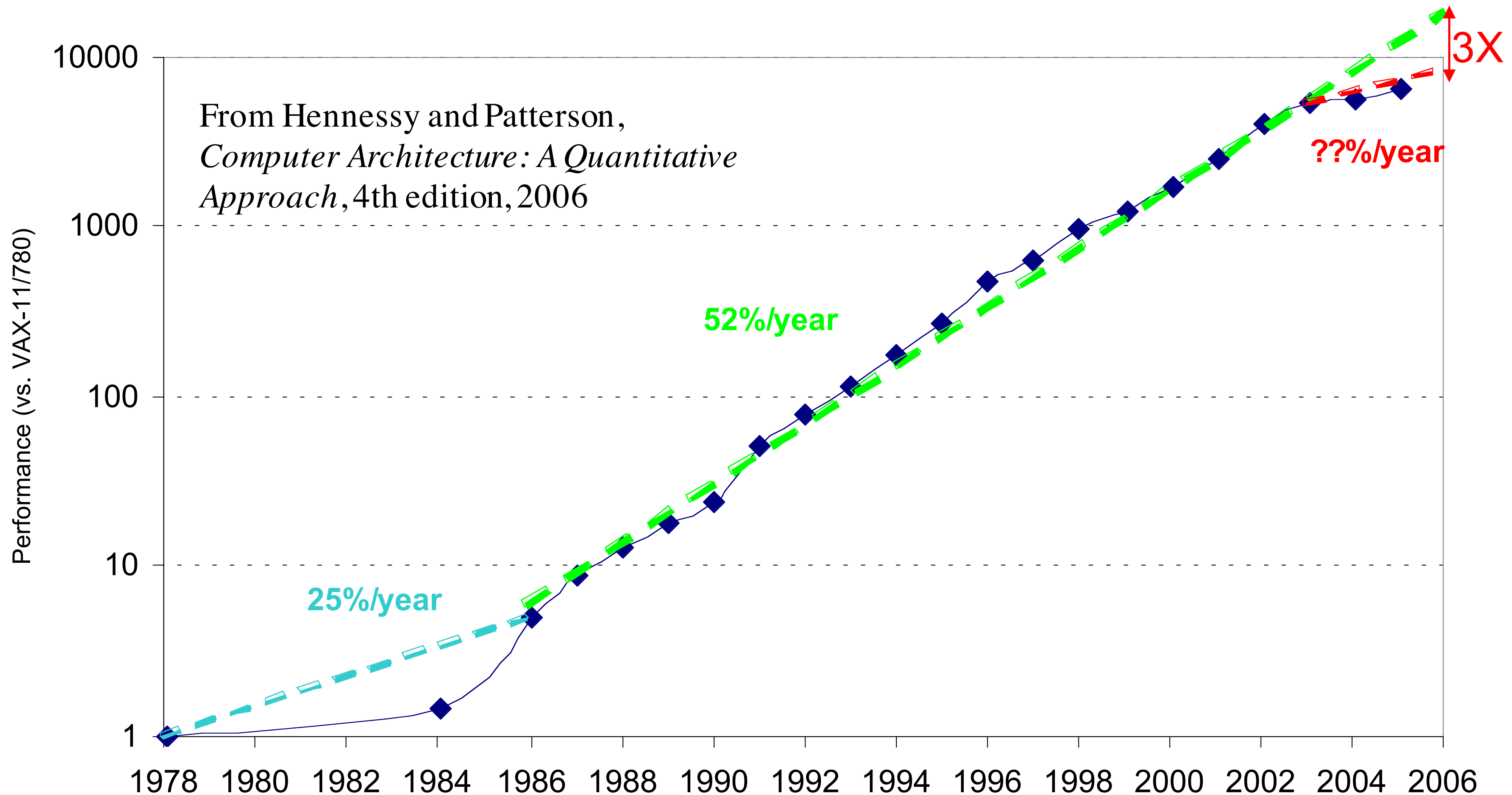
CS 152 Computer Architecture and Engineering

Lecture 18: Multi-Processors - Snoopy Caches

John Wawrzynek
Electrical Engineering and Computer Sciences
University of California, Berkeley

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

Uniprocessor Performance (SPECint)



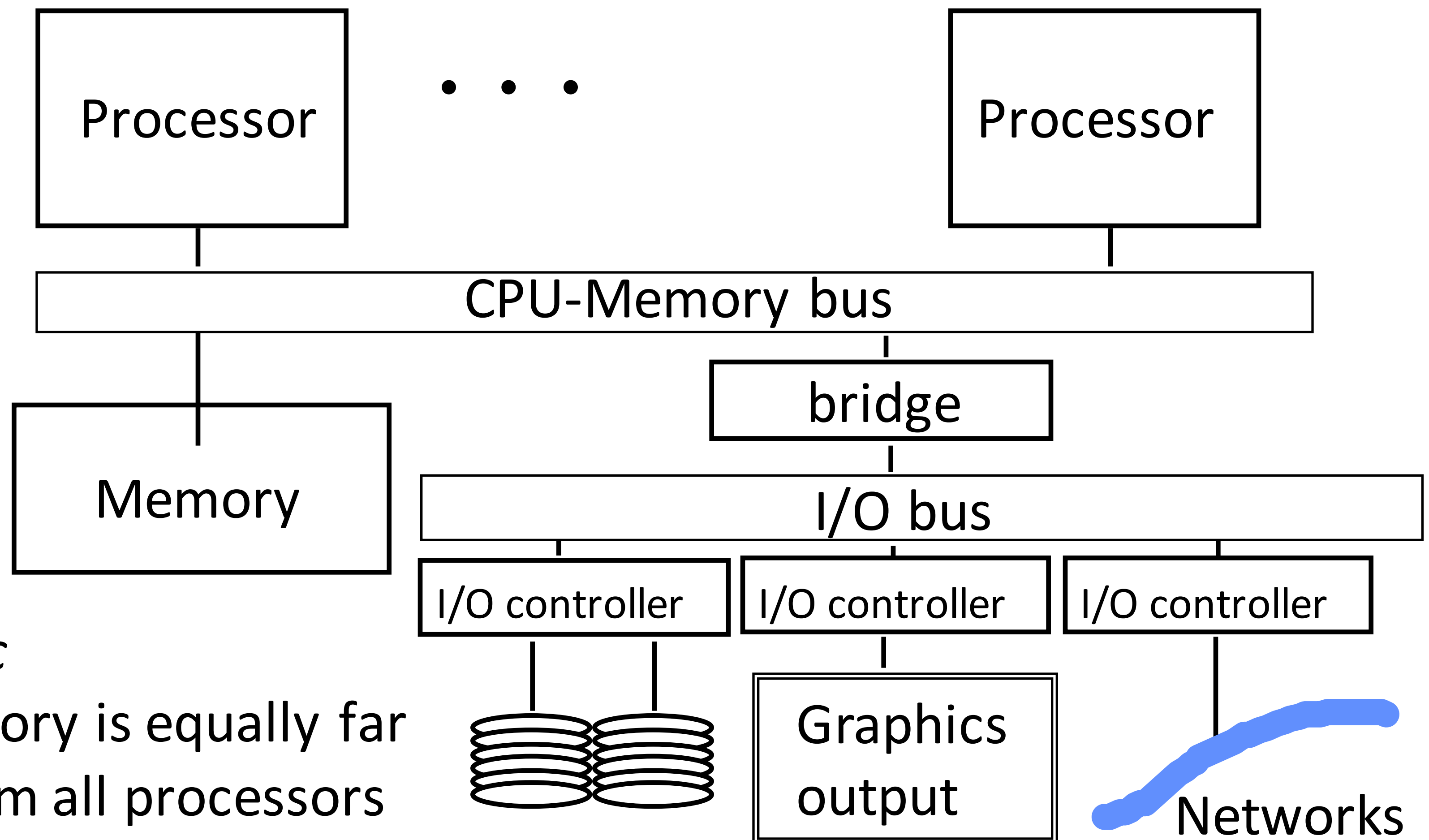
- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

CS152-Spring'09

Parallel Processing: Déjà vu all over again?

- "… today's processors … are nearing an impasse as technologies approach the speed of light.."
  - David Mitchell, The Transputer: The Time Is Now (1989)
- Transputer had bad timing (Uniprocessor performance↑) ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"
  - Paul Otellini, President, Intel (2005)
- All microprocessor companies switch to MP (2+ CPUs/2 yrs) ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
- Even handheld systems moved to multicore
  - Nintendo 3DS, iPhone6 has two cores each (plus additional specialized cores), Android Qualcomm Snapdragon 805 has four cores. Playstation Portable Vita has four cores.

Symmetric Multiprocessors (SMPs)



symmetric

- All memory is equally far away from all processors
- Any processor can do any I/O (set up a DMA transfer)

Local caches at processors makes it practical!

Synchronization

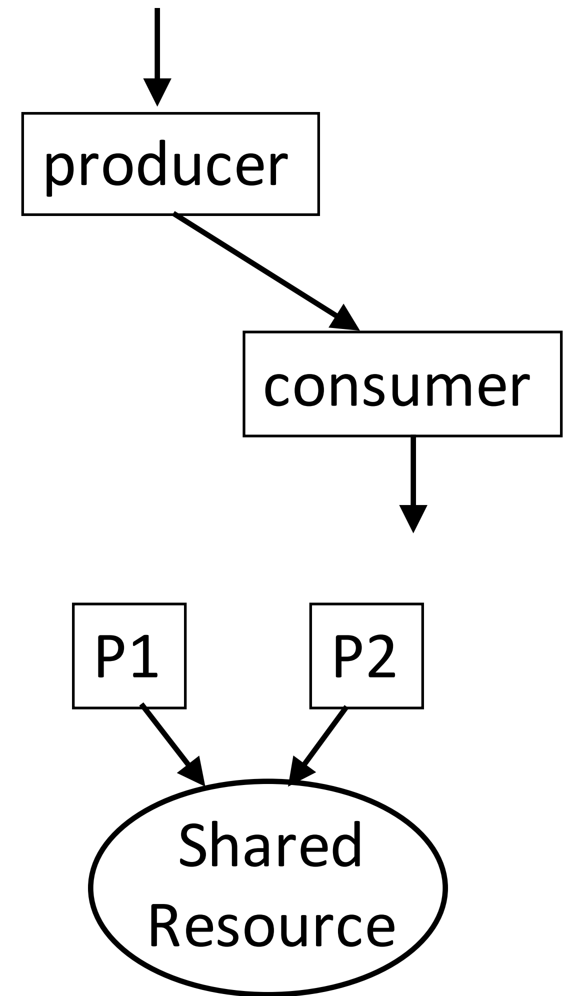
The need for synchronization arises whenever there are concurrent processes in a system cooperating on some task

(even in a uniprocessor system)

Two classes of synchronization:

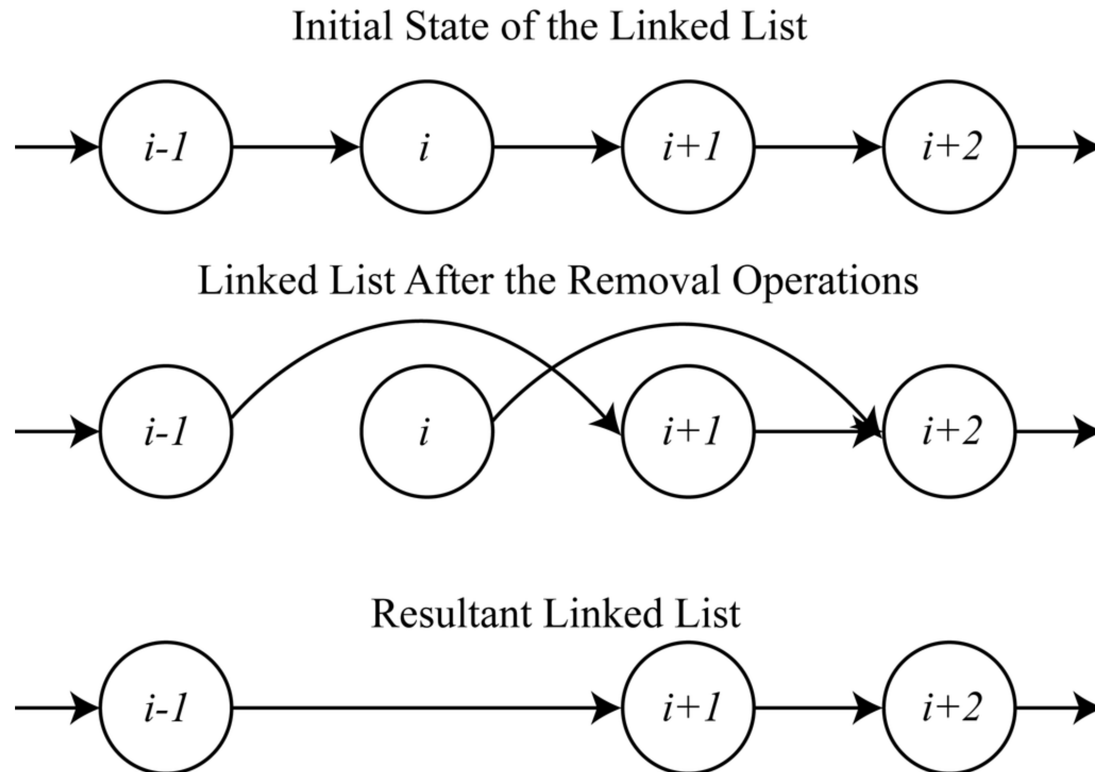
Producer-Consumer: A consumer process must wait until the producer process has produced data

Mutual Exclusion: Ensure that only one process uses a resource at a given time

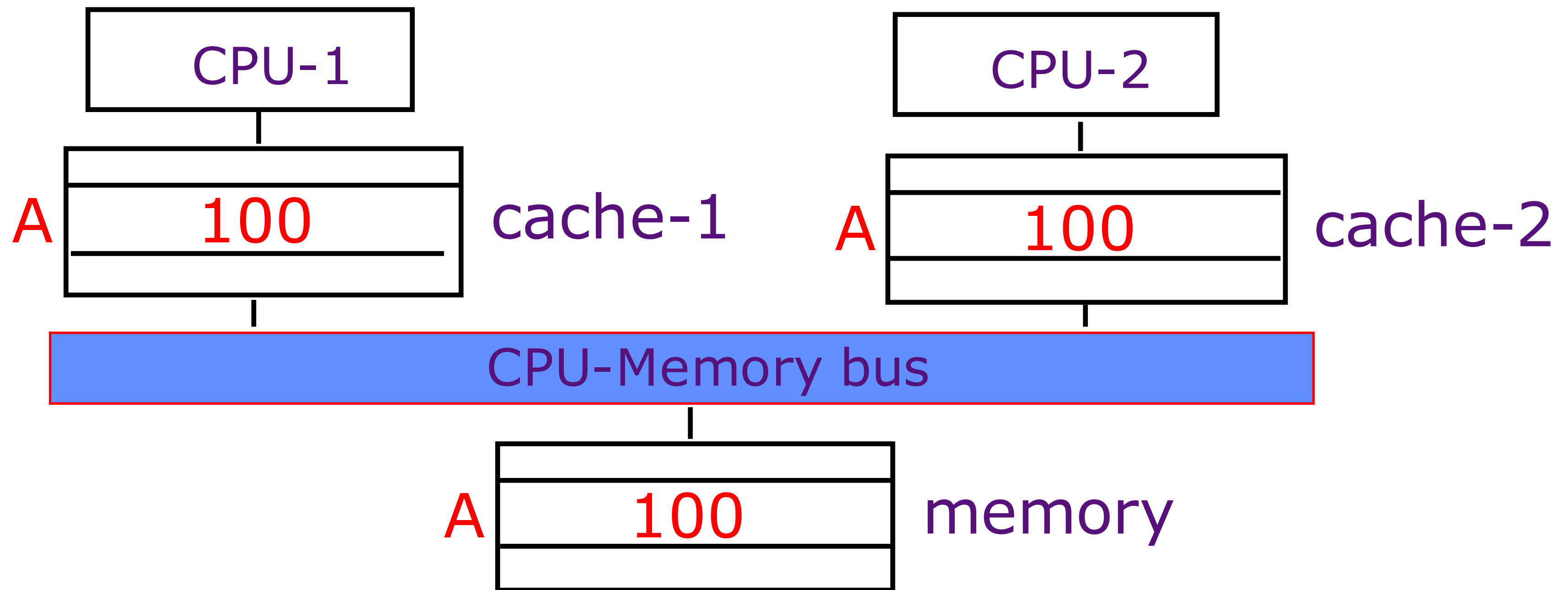


Need for Mutual Exclusion

- Example (wikipedia): shared linked list management
- Two nodes, *i* and *i* + 1, being removed simultaneously results in node *i* + 1 not being removed.



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

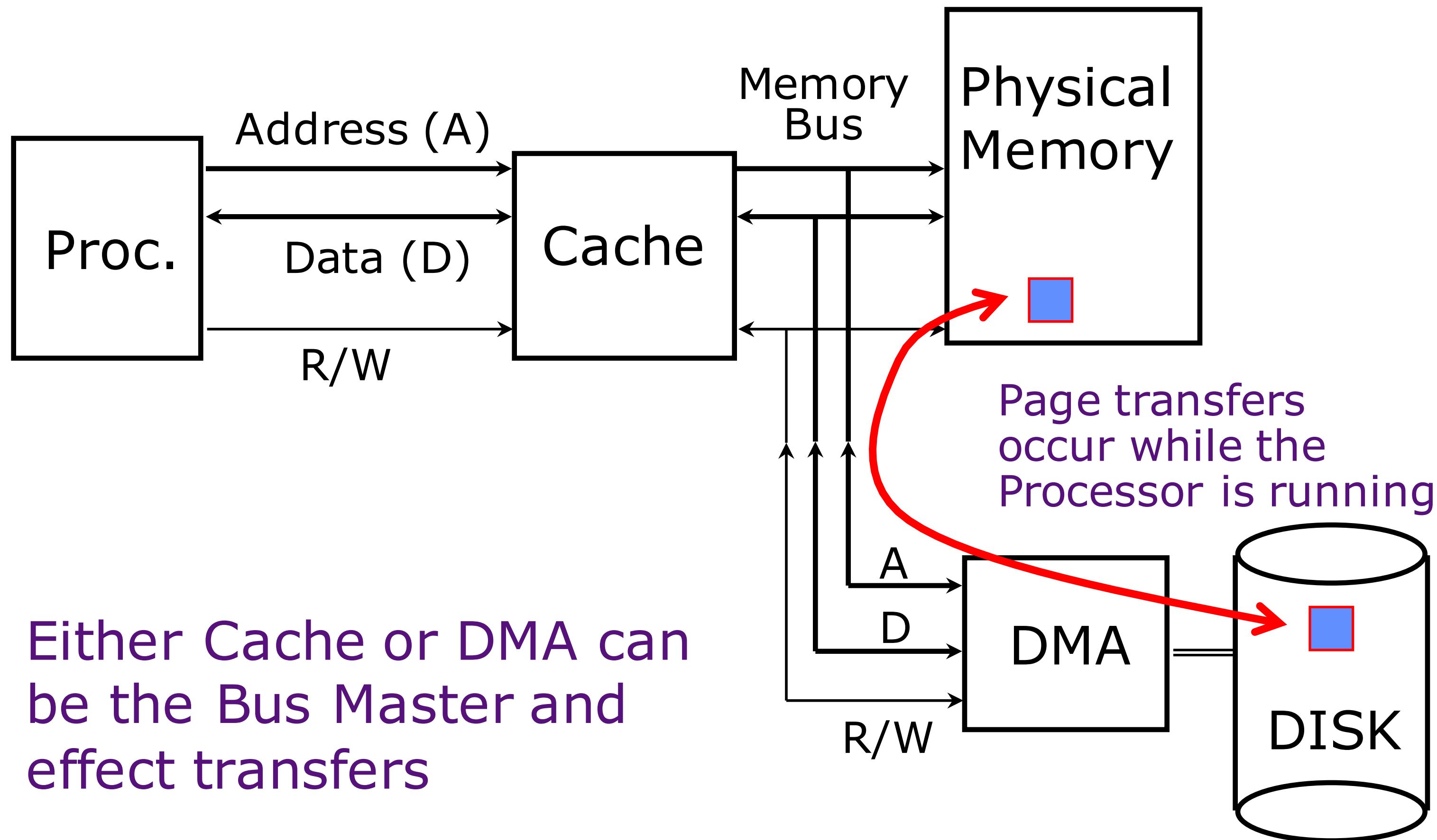
Do these stale values matter?
What is the view of shared memory for programming?

Maintaining Cache Coherence

- A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors, for one memory address
  - i.e., updates are not lost
- 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

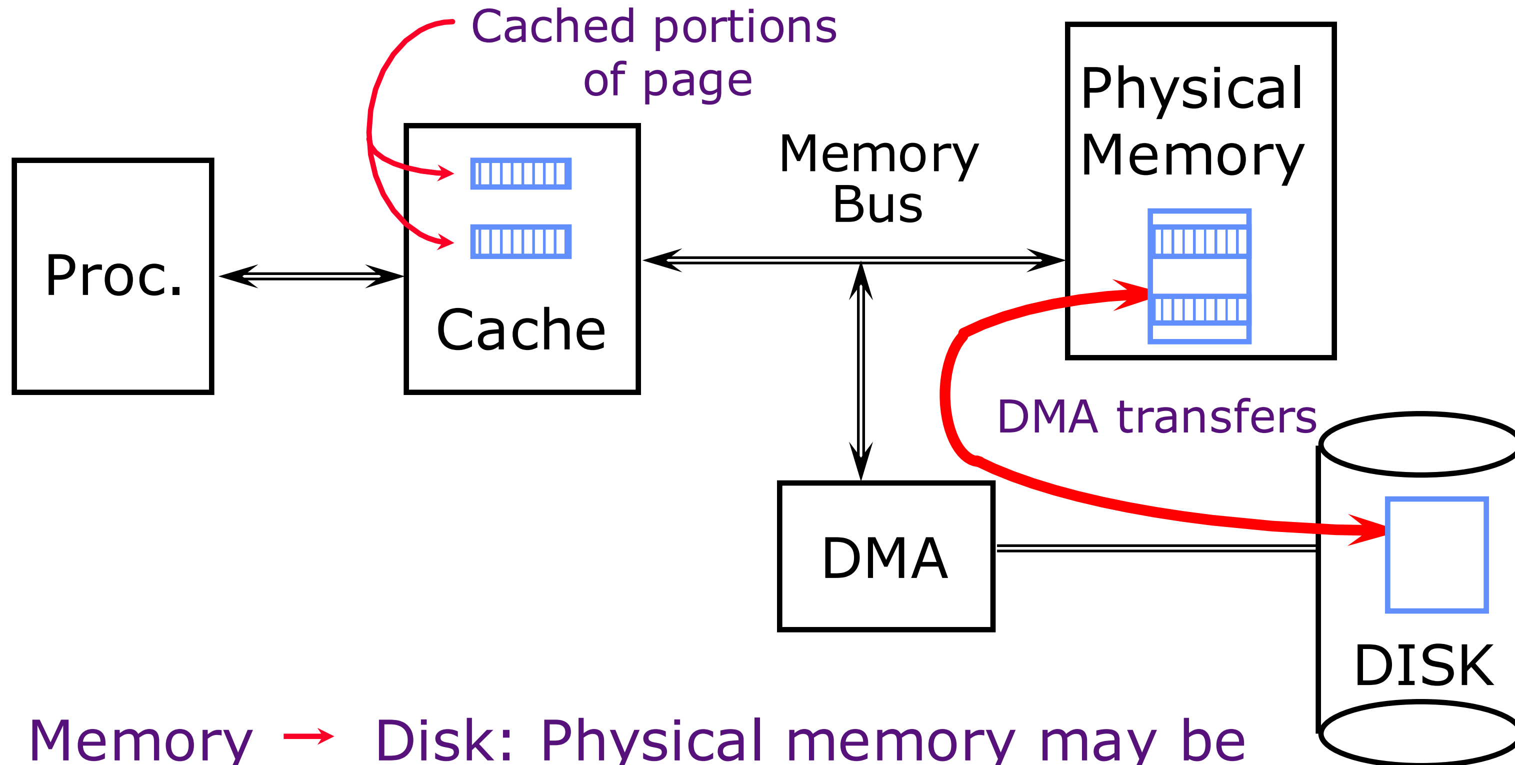
  $\Rightarrow$ cache coherence protocols

Warmup: Parallel I/O



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

Problems with Parallel I/O

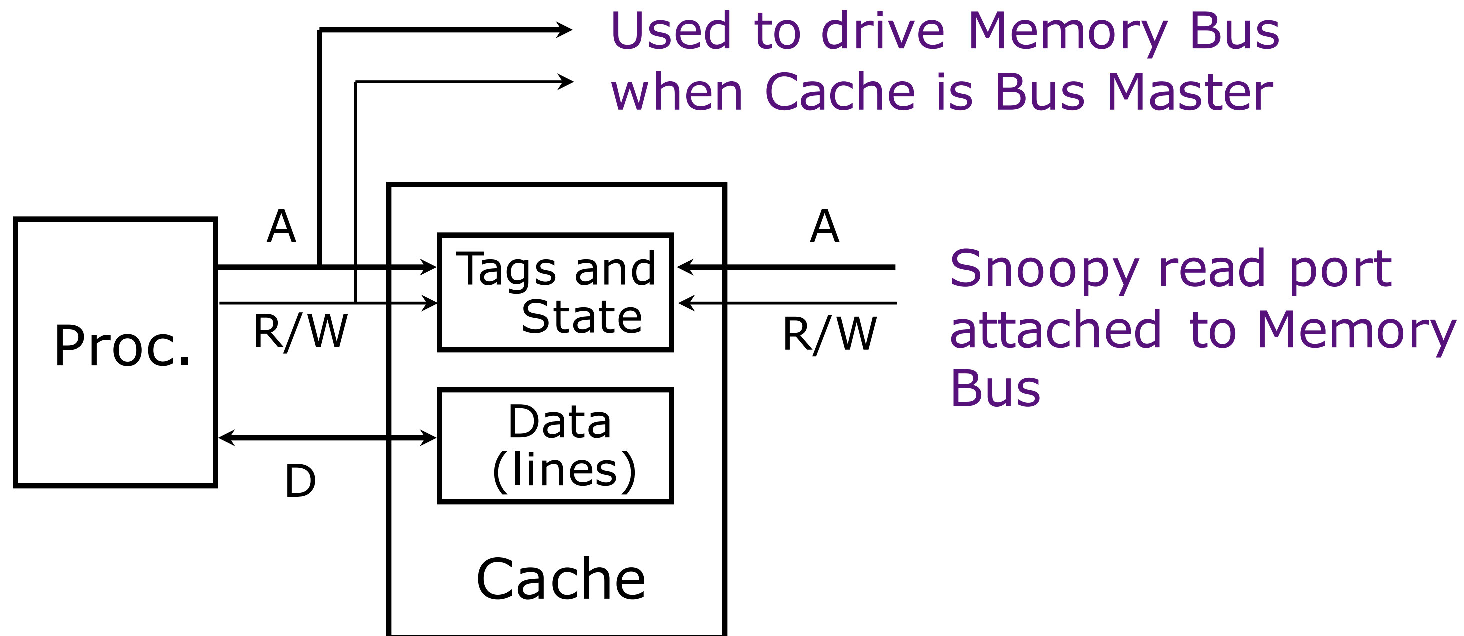


Memory → Disk: Physical memory may be stale if cache copy is dirty

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

Snoopy Cache, *Goodman 1983*

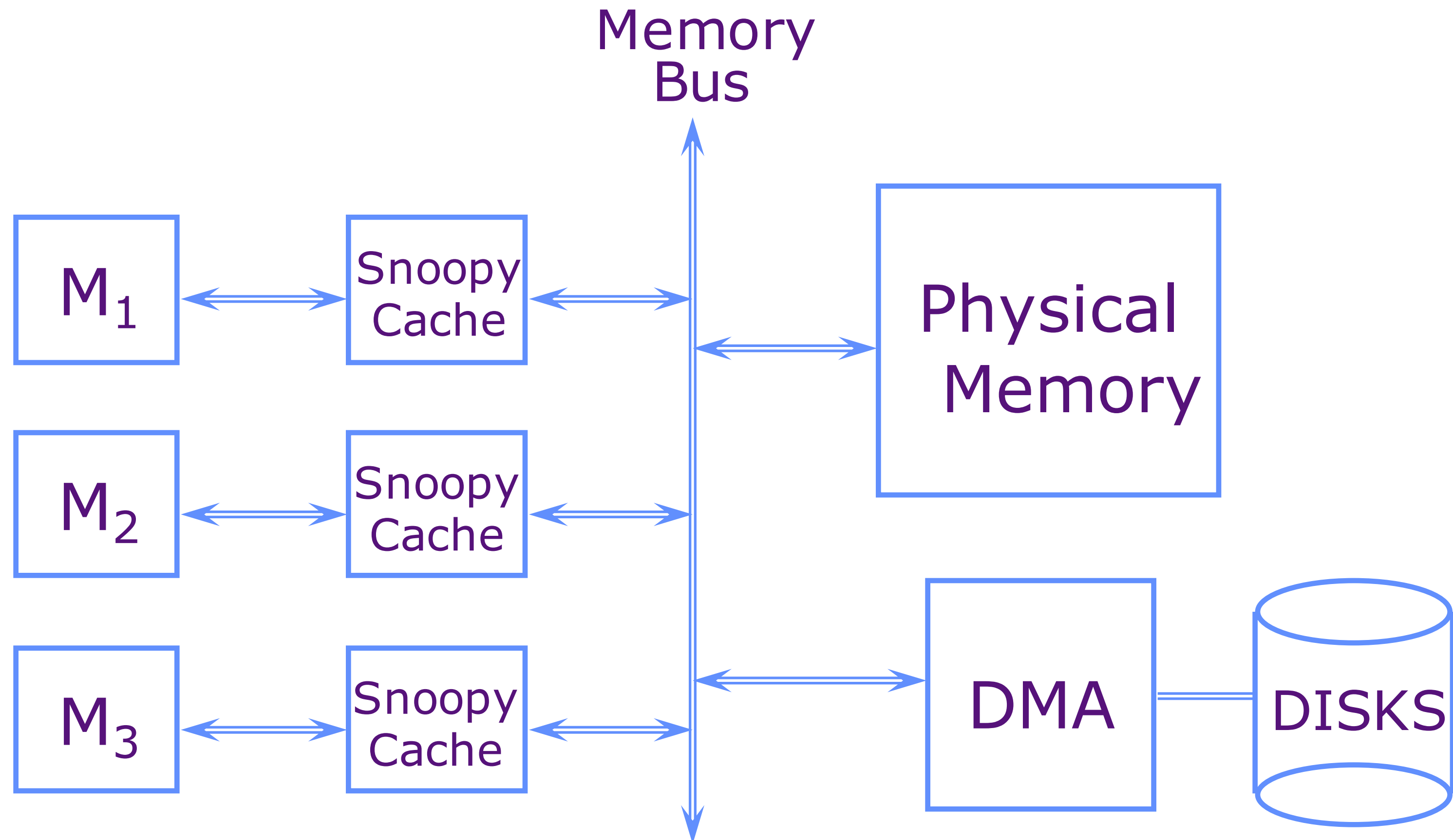
- Idea: Have cache watch (or snoop upon) DMA transfers, and then "do the right thing"
- Snoopy cache tags are dual-ported



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	???

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:

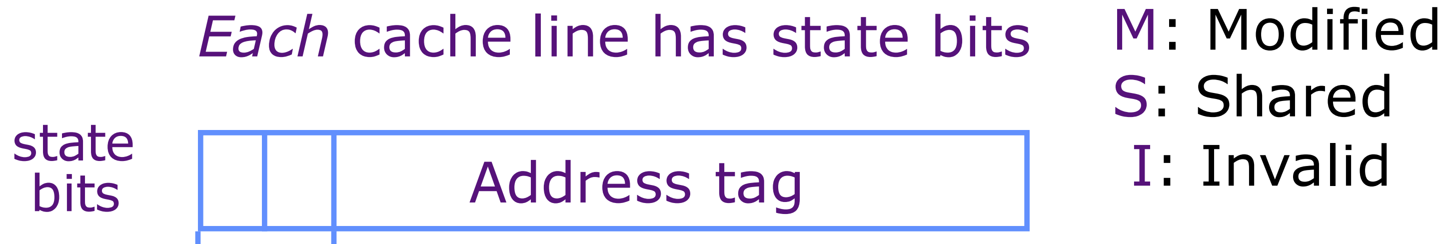
if a dirty copy is found in some cache, a write-back is performed before the memory is read

The MSI protocol

M: Modified
S: Shared
I: Invalid

- **M**odified: The block has been modified in the cache. The data in the cache is then inconsistent with the backing store (e.g. memory). A cache with a block in the "M" state has the responsibility to write the block to the backing store when it is evicted. *A block in the Modified state is exclusive (it can't be in any other cache).*
- **S**hared: This block is unmodified and exists in read-only state in at least one cache. The cache can evict the data without writing it to the backing store.
- **I**nvalid: This block is either not present in the current cache or has been invalidated by a bus request, and must be fetched from memory if the block is to be stored in this cache.

The MSI protocol



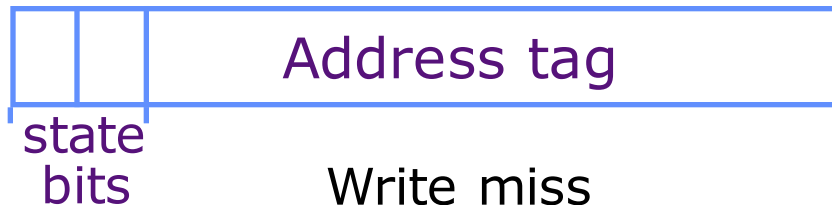
M: Modified
S: Shared
I: Invalid

- A read miss to a block in a cache, C1, generates a bus transaction – if another cache,C2, has the block in M state ("exclusively"), it has to write back the block before memory supplies it. C1 gets the data from the bus and the block becomes "shared" in both caches.
- A write hit to a shared block in C1 forces a write back – all other caches that have the block should invalidate that block – the block becomes "exclusive" in C1.
- A write hit to a modified (exclusive) block does not generate a write back or change of state.
- A write miss (to an invalid block) in C1 generates a bus transaction
  - If a cache, C2, has the block as "shared", C2 invalidates it's copy
  - If a cache, C2, has the block in "modified (exclusive)", it writes back the block and changes it state in C2 to "invalid".
  - If no cache supplies the block, the memory will supply it.
  - When C1 gets the block, it sets its state to "modified (exclusive)"

Cache State Transition Diagram

The MSI protocol

Each cache line has state bits



M: Modified
S: Shared
I: Invalid

Write miss
(P1 gets line from memory)

P_1 reads or writes

Other processor reads
(P_1 writes back)

M

P_1 intent to write

Other processor intent to write
(P_1 writes back)

Read miss
(P_1 gets line from memory)

S

I

Read by any processor

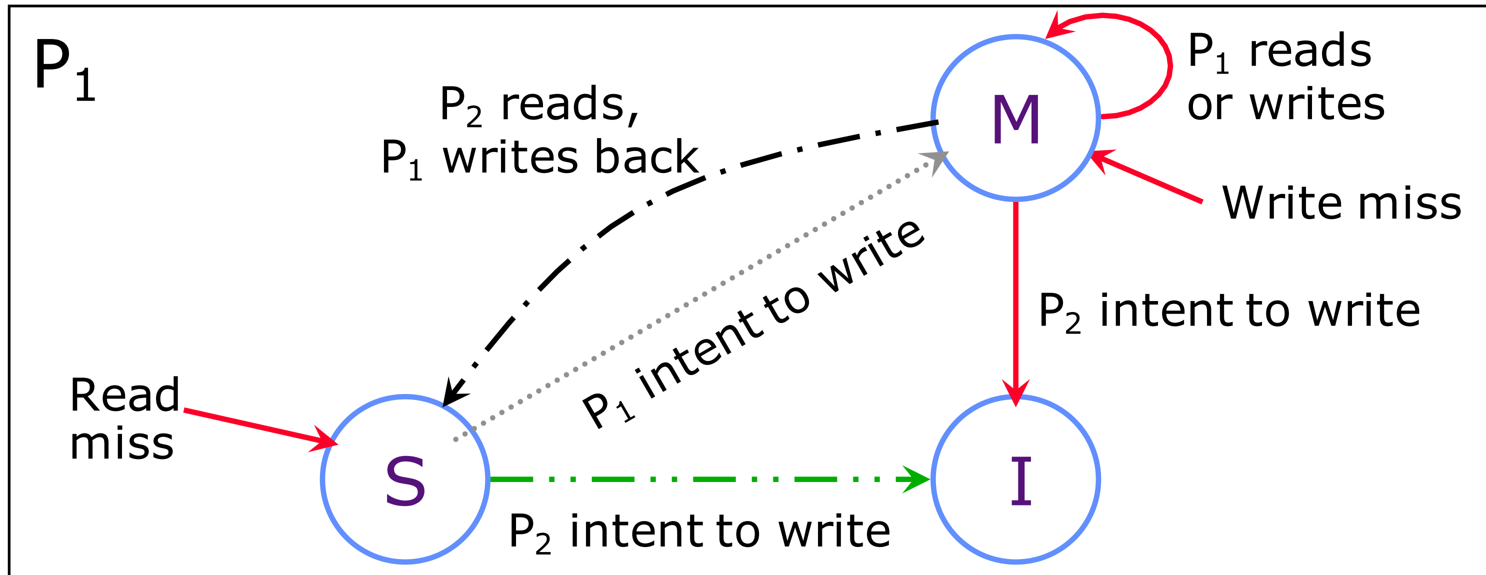
Other processor intent to write

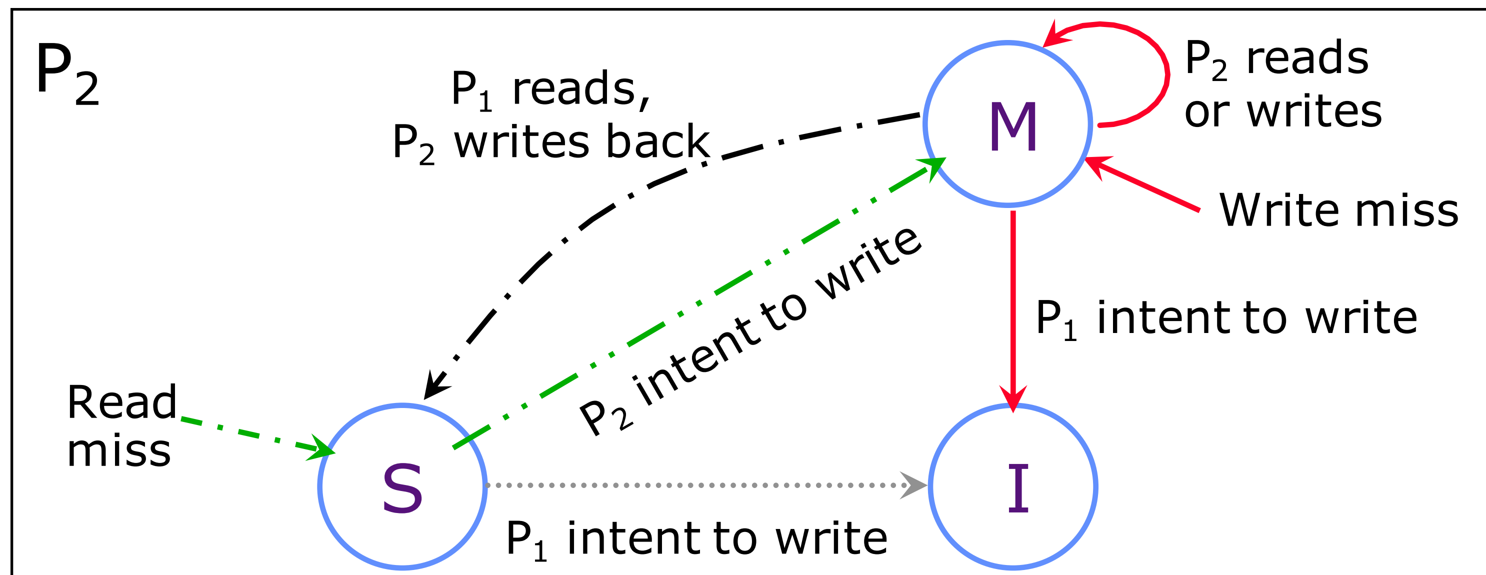
Cache state in processor P_1

Two Processor Example

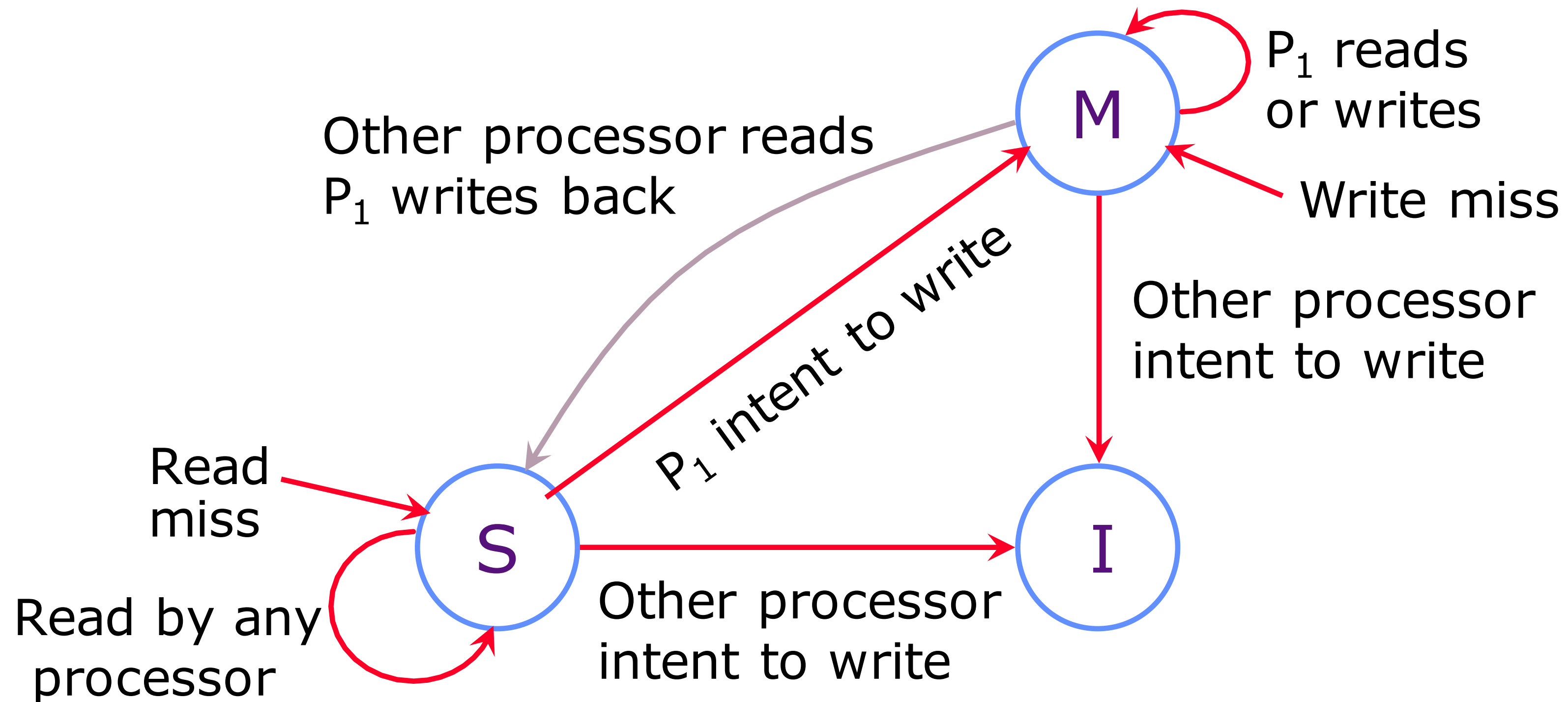
(Reading and writing the same cache line)

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





Observations



- 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
- **A write to a line in the S state causes a writeback (even if no other cache has a copy!)**

MESI: An Enhanced MSI protocol

increased performance for private data

Each cache line has a tag

M: Modified Exclusive
E: Exclusive but unmodified
S: Shared
I: Invalid

Write miss

P_1 write
or read

P_1 write

P_1 read

Read miss,
not shared

P_1 intent
to write

Other
processor
reads

Other processor reads
P_1 writes back

Other processor
intent to write

Read miss,
shared

Other processor
intent to write, P1
writes back

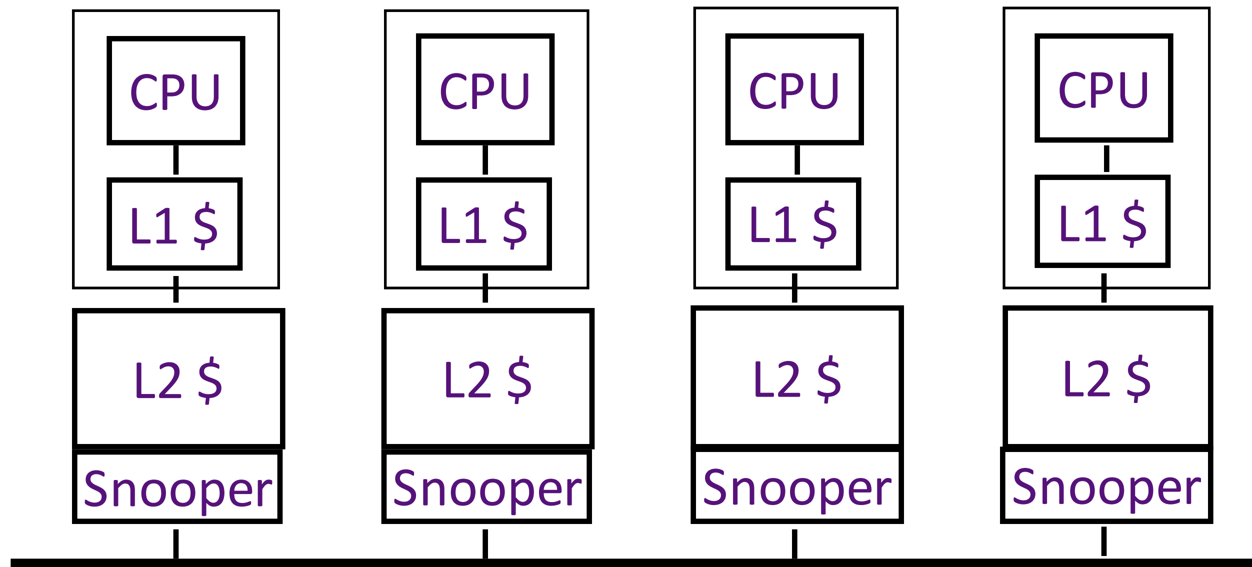
Read by any
processor

Other processor
intent to write

Cache state in
processor P_1

Write to a Exclusive line doesn't cause a writeback.

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 $\Rightarrow$ invalidation in L1
- Snooping on L2 does not affect CPU-L1 bandwidth

What problem could occur?

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