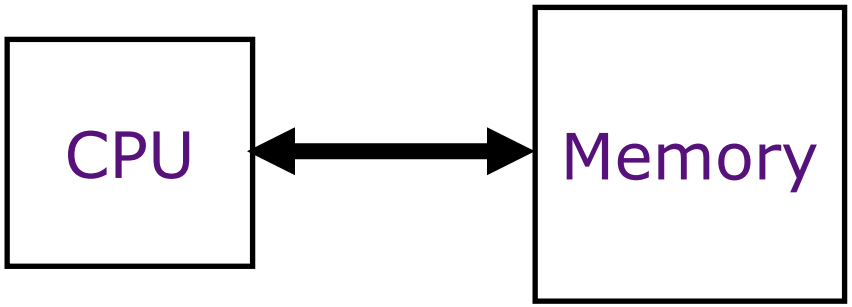
CSE 486/586 Distributed Systems
Cache Coherence

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Storage to Memory

- We've looked at storage consistency.
- The same consistency models are equally applicable to memory.
  - Think multiple threads accessing the same memory addresses
- But a memory system can have another form of consistency mainly for managing caches. We'll look at this today.
  - In a multi-core system, there are many caches, and they need to be synchronized.

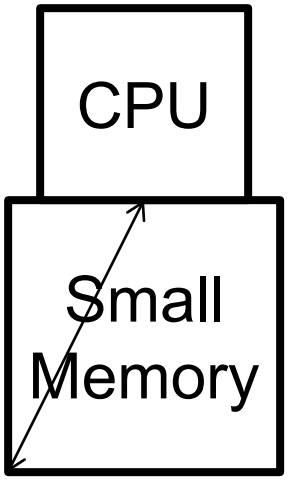
Caching Basics: CPU-Memory Bottleneck

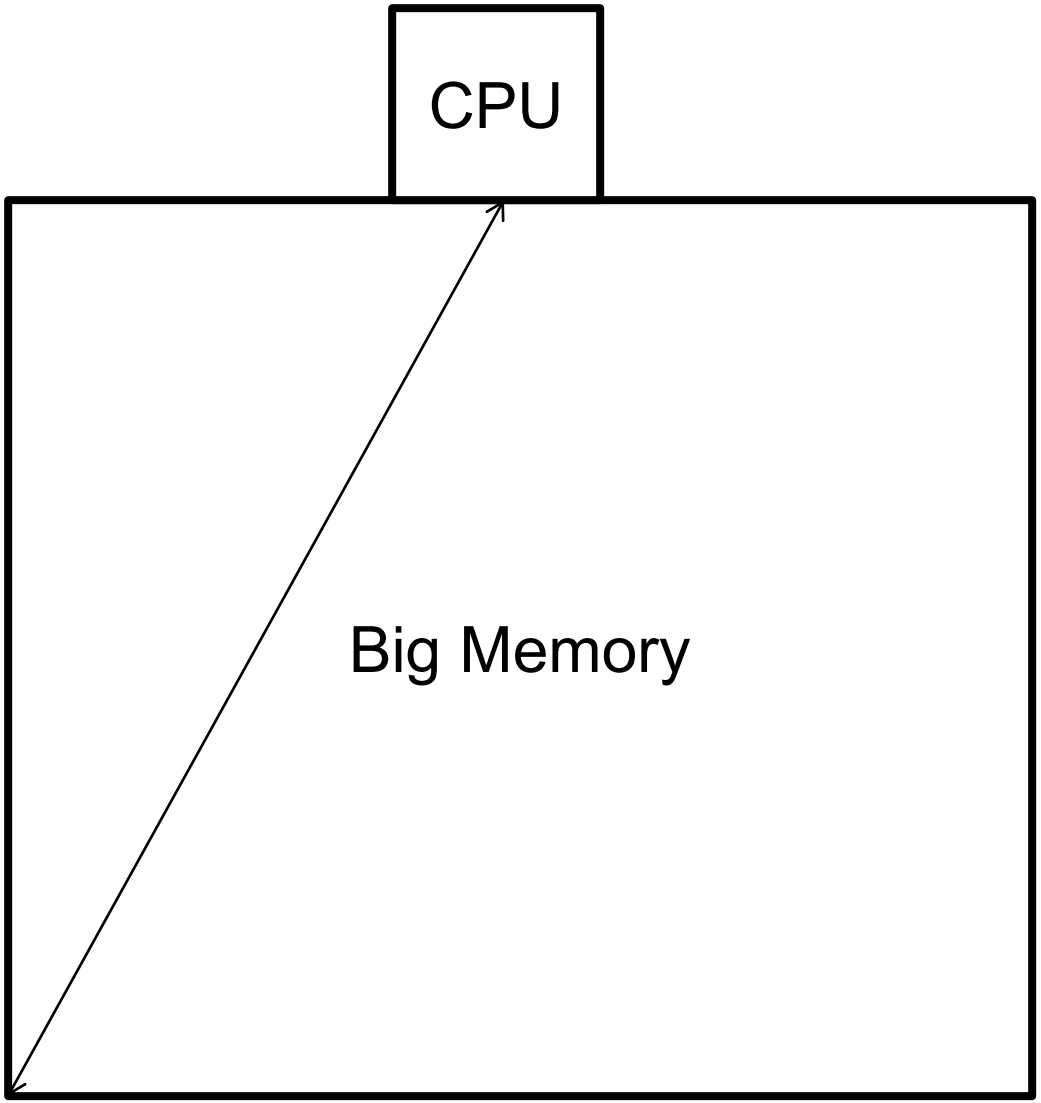


Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- Latency (time for a single access)
  - Memory access time >> Processor cycle time
  - Problematic
- Bandwidth (number of accesses per unit time)
  - Increase the bus size, etc.
  - Usually OK

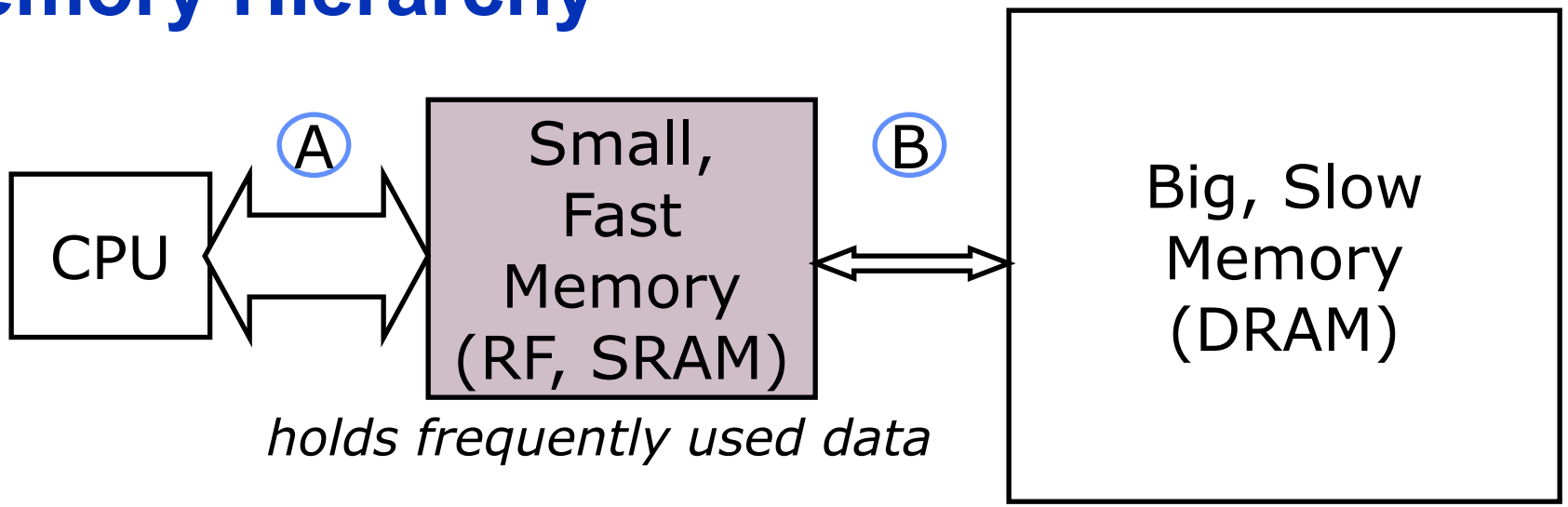
Physical Size Affects Latency





- Signals have further to travel
- Fan out to more locations

Memory Hierarchy



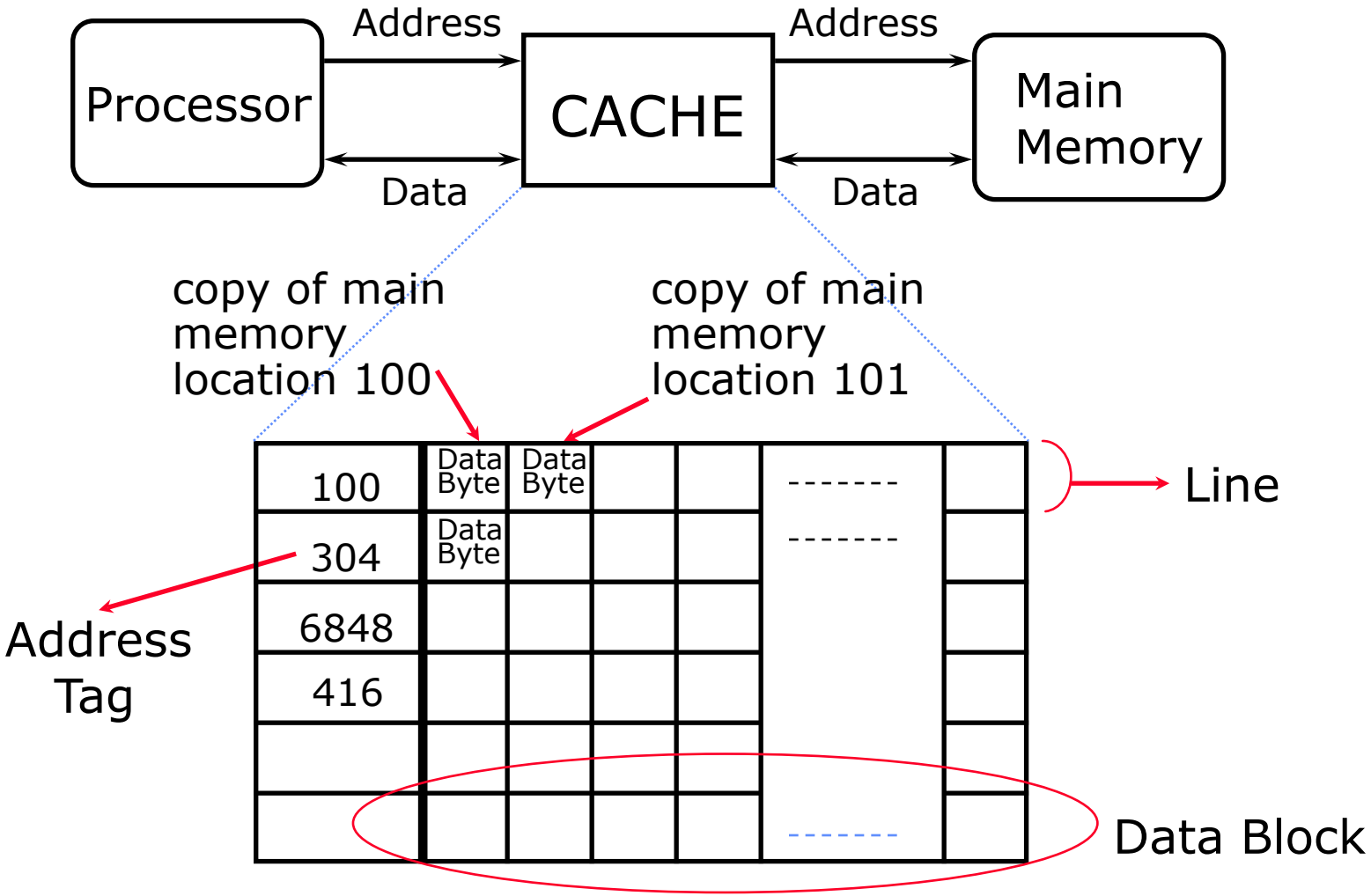
- *capacity*: Register << SRAM << DRAM *(cost)*
- *latency:* Register << SRAM << DRAM *(size)*
- *bandwidth:* on-chip >> off-chip *(delays)*

On a data access:

if data $\in$ fast memory $\Rightarrow$ low latency access *(SRAM)*

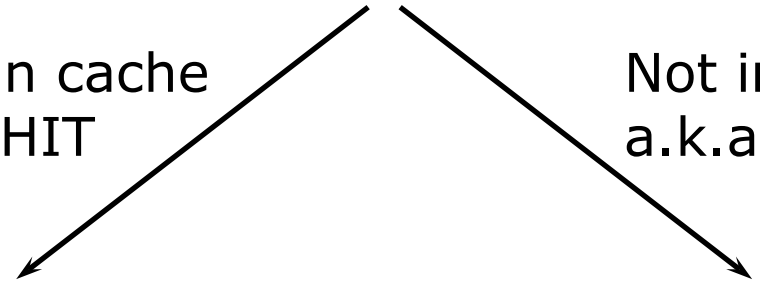
If data $\notin$ fast memory $\Rightarrow$ long latency access *(DRAM)*

Inside a Cache



Cache Read

Look at Processor Address, search cache tags to find match. Then either



Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

Read block of data from Main Memory

Wait …

Return data to processor and update cache
(Use a replacement algorithm to select a line to replace)

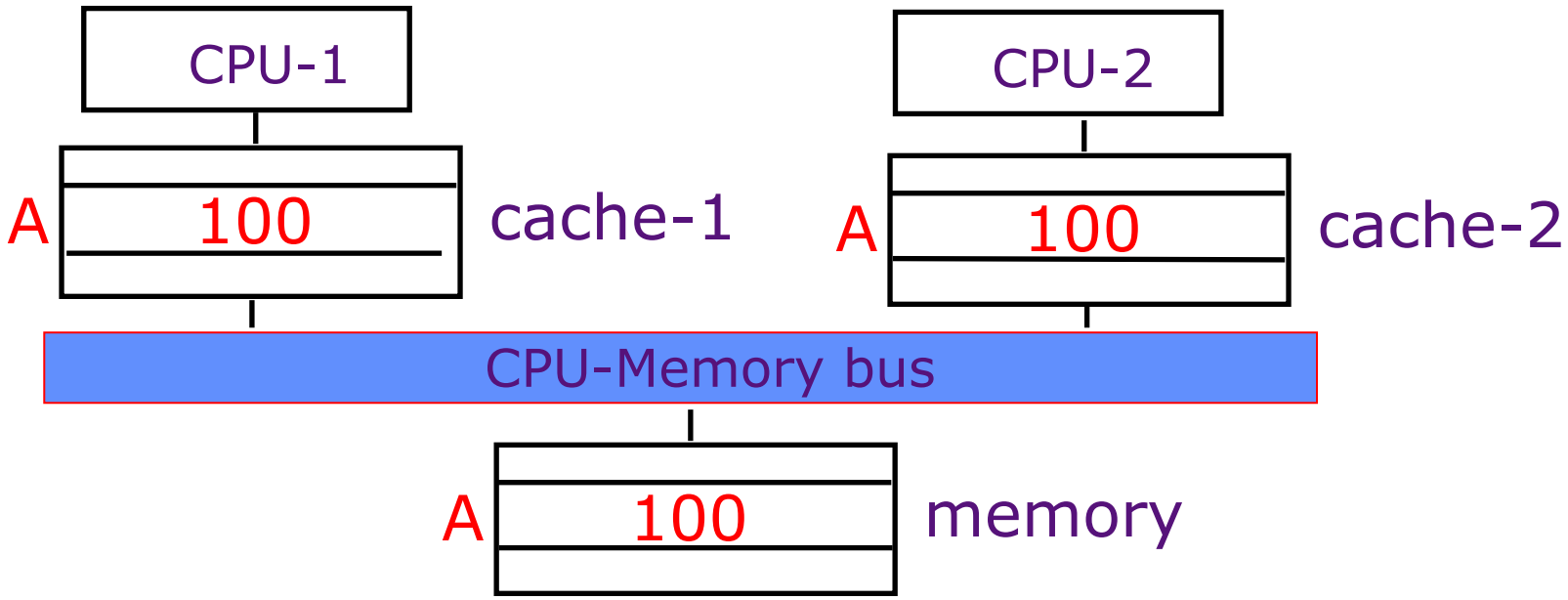
Cache Write

- Cache hit:
  - ***write through:*** write both cache & memory
  - ***write back:*** write cache only, memory is written only when the entry is evicted
- Cache miss:
  - ***no write allocate:*** only write to main memory
  - ***write allocate*** *(aka fetch on write):* fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate

Administrivia

- PA3 grading still going on
- This Friday, no recitation, undergrad office hours from 2 pm – 4 pm & general office hours from 4 pm – 5 pm

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 kind of guarantee do you get?

Cache Coherence

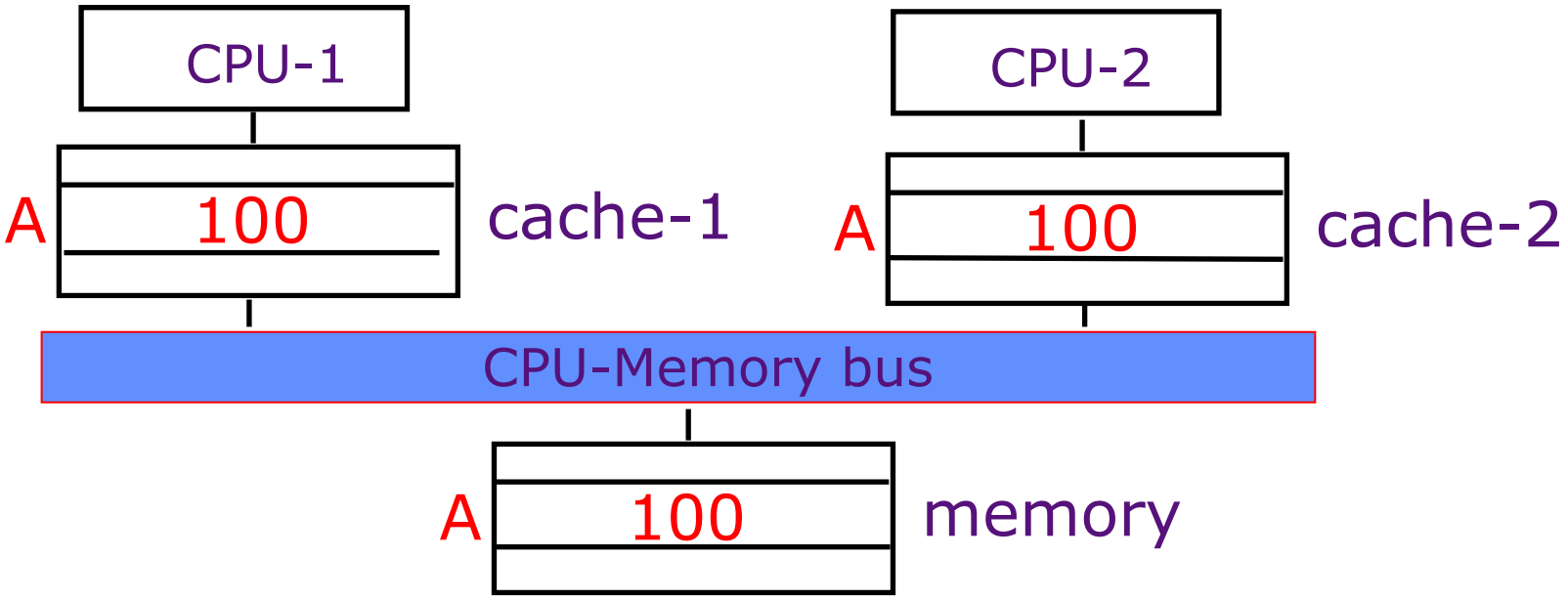
- A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors
  - i.e., updates are not lost
  - You can consider this a hardware-based update propagation mechanism for distributed caches.

- 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

Cache Coherence

- A memory system is coherent if:
- A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.
- A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
- Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors.
- (Coherence provides per-location sequential consistency).

One Design: Snoopy Cache



- Cache controllers work together to maintain cache coherence.
- Cache controllers send commands to the bus.
- Each cache controller snoops on the bus traffic to catch various commands and follow them.

Snoopy Cache Coherence Protocol

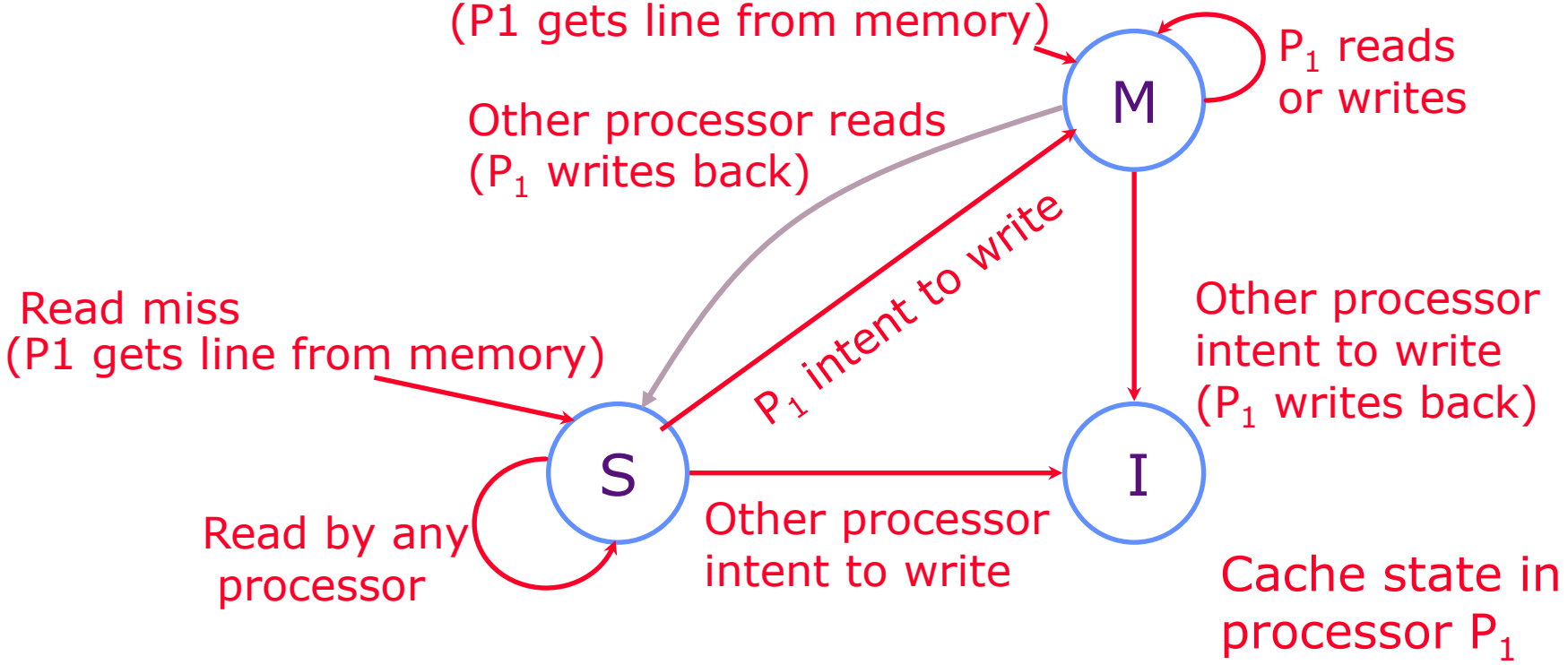
- Each cache line has a state:
  - M (modified): no other cache has a copy and the processor can write to it.
  - S (shared): other caches might have a copy as well.
  - I (invalid): the data is no longer valid.
- If a cache line is in S, then only read is possible.
- If a cache line is in M, then write is possible as well.
- Writing to a cache line
  - If it's M, the cache controller does the write.
  - If it is not M, it sends an invalidation request to other caches, switches the state to M, and does the write.
  - Other cache controllers switch the state to I.
- Reading a memory address
  - If it's a hit, read it.
  - If it's not a hit, read it from memory, and other cache controllers switch the state to S.

Cache State Transition Diagram

The MSI protocol

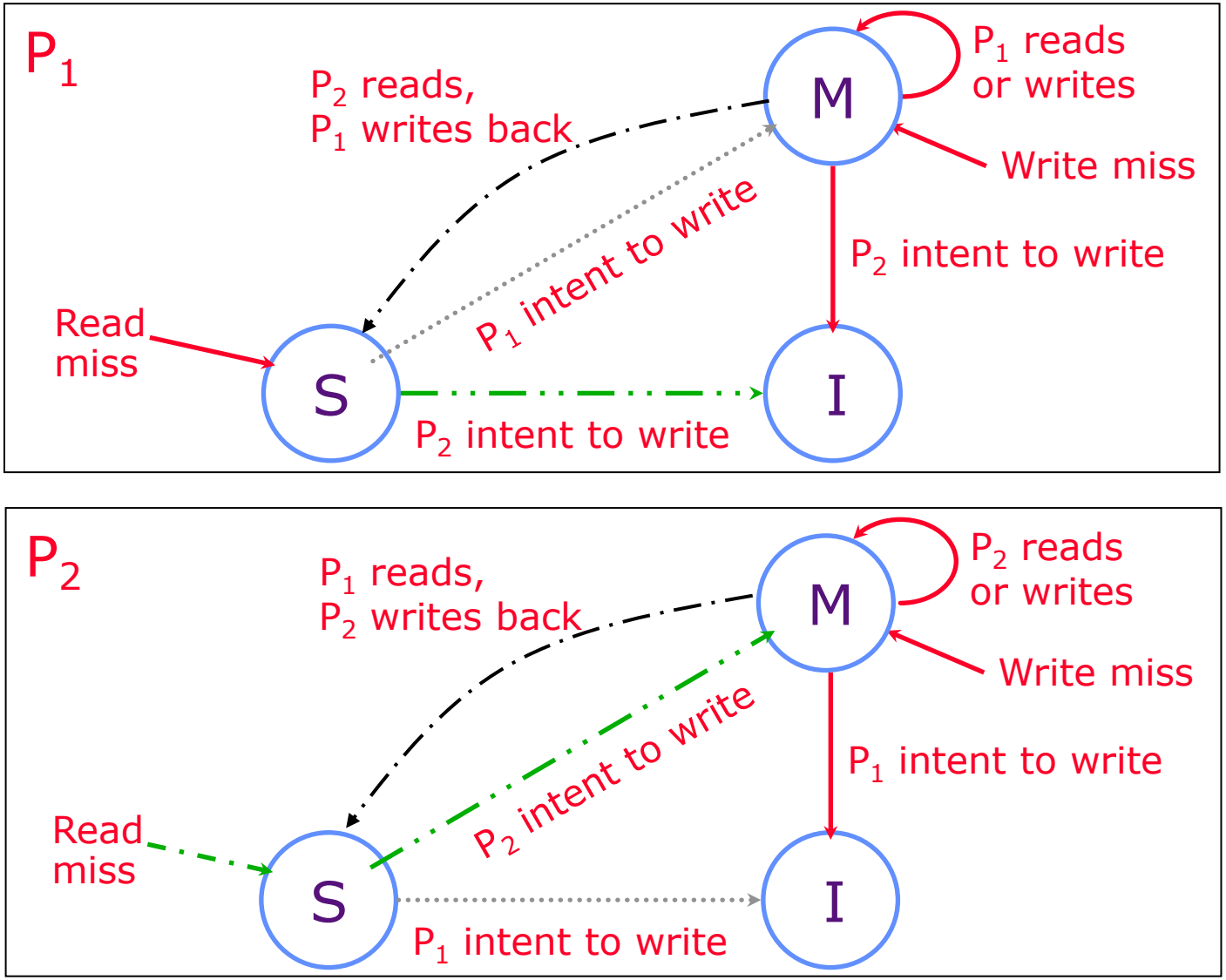
Each cache line has state bits

M: Modified
S: Shared
I: Invalid

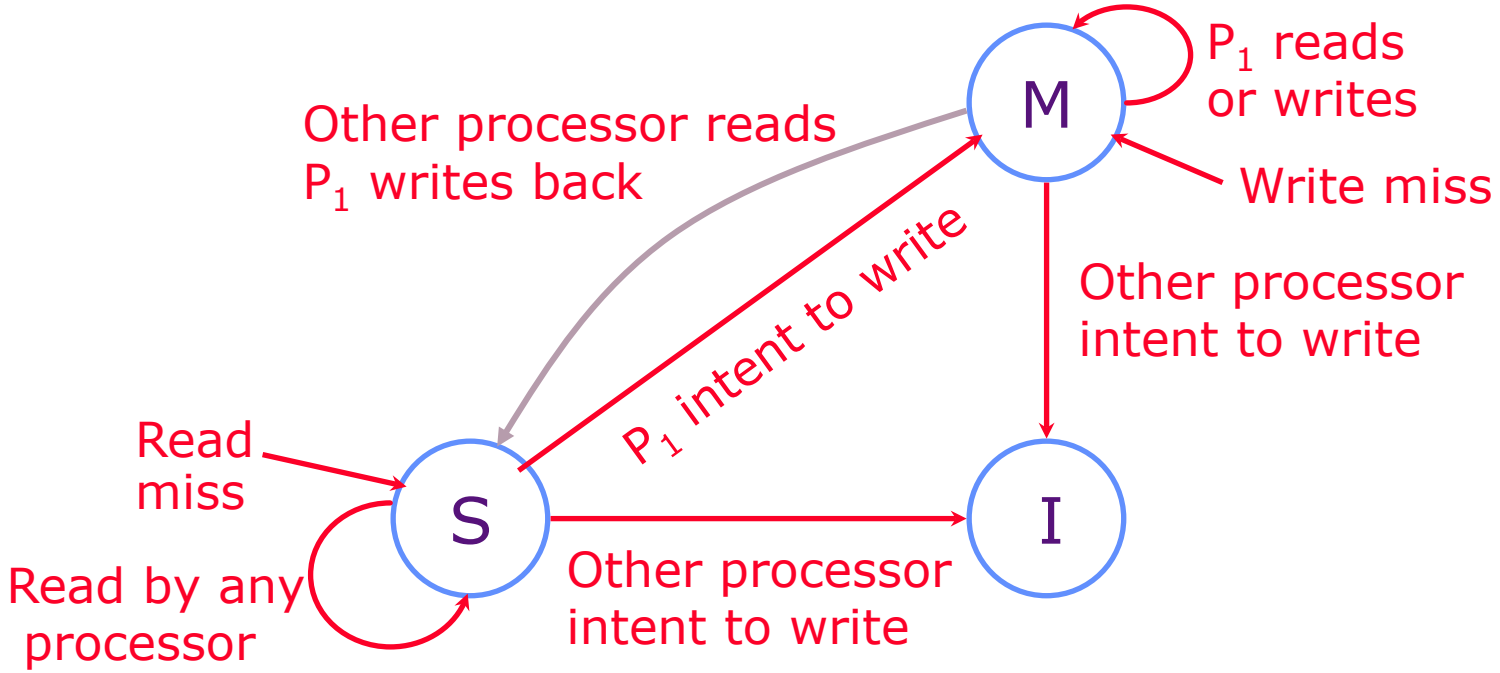


Two Processor Example
(Reading and writing the same cache line)

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

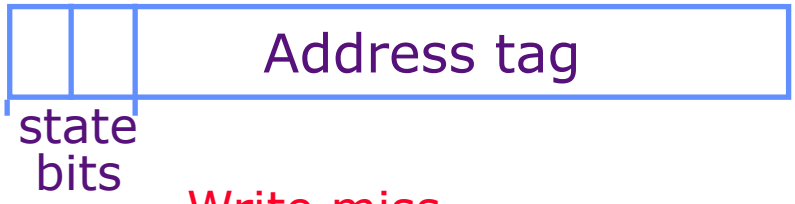


- 2 bits used for 3 states
  - There's room for one more state
- S indicates that sharing is possible, but not definite.
  - From S to M, there's always invalidation requests, even when it's not actually shared.

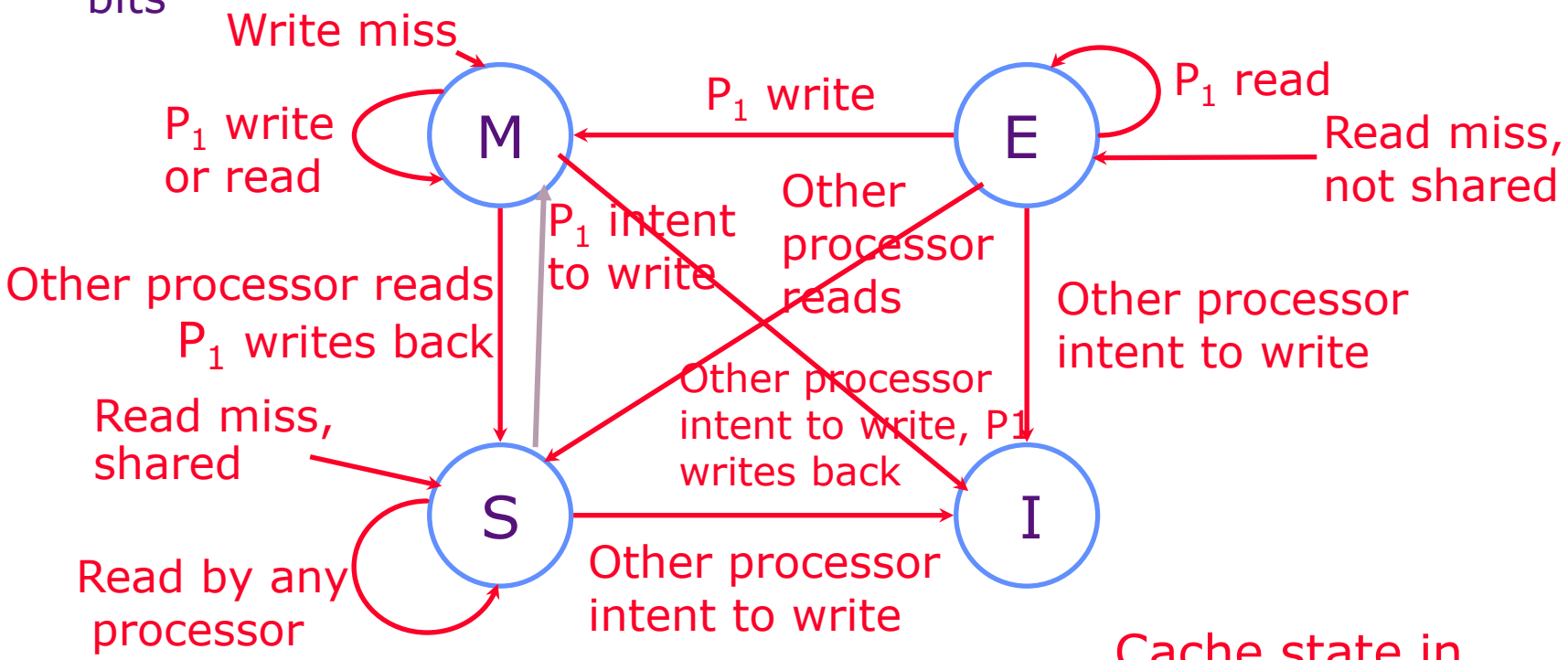
MESI: An Enhanced MSI protocol

increased performance for private data

Each cache line has a tag



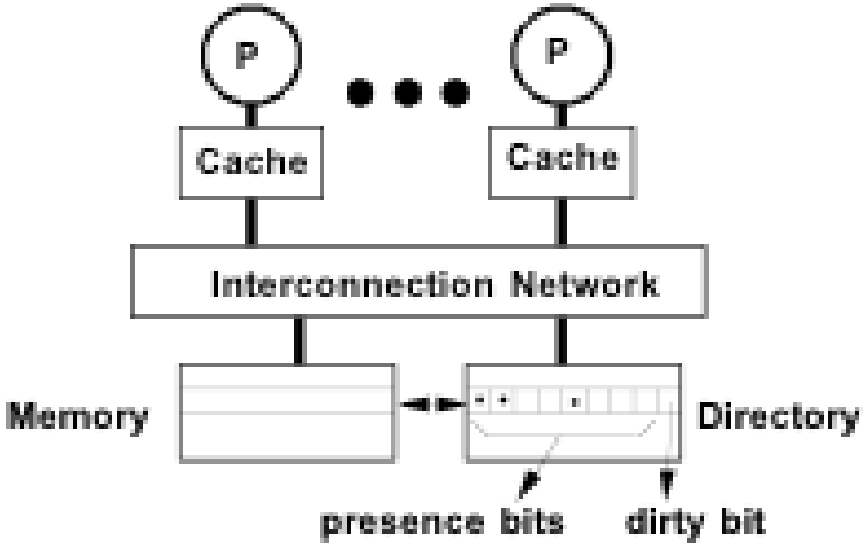
M: Modified Exclusive
E: Exclusive but unmodified
S: Shared
I: Invalid



Cache state in processor P_1

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
  - Many alternatives for organizing directory information



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

Summary

- Cache coherence
  - Making sure that caches do not contain stale copies.
- Snoopy cache coherence
  - MSI
  - MESI
- Directory-based
  - Uses a directory per memory block

Acknowledgements

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

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