CS 110
Computer Architecture

Cache Coherence

Guest Instructor:
Prof. Shu Yin

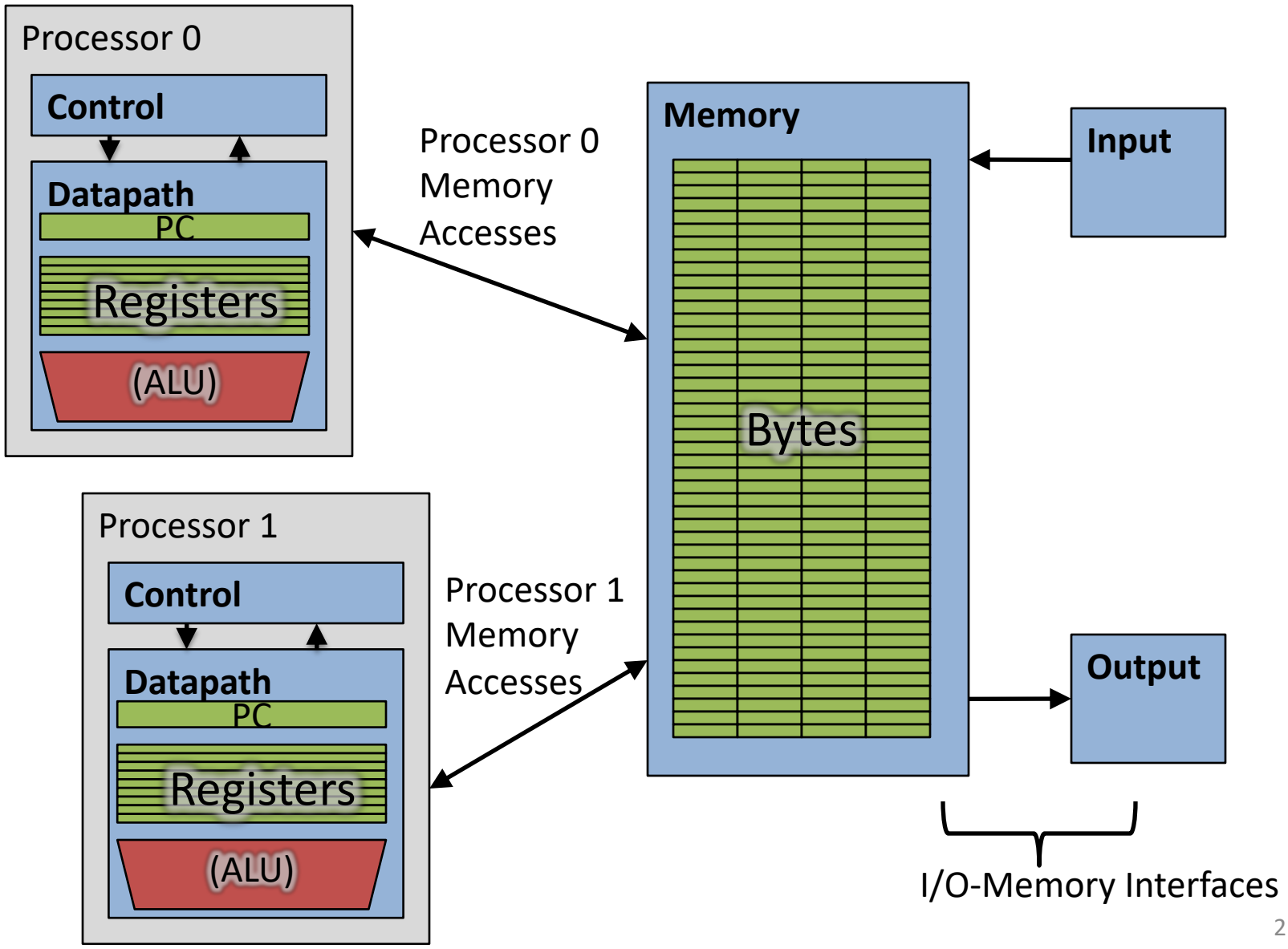
https://robotics.shanghaitech.edu.cn/courses/ca

School of Information Science and Technology SIST

ShanghaiTech University

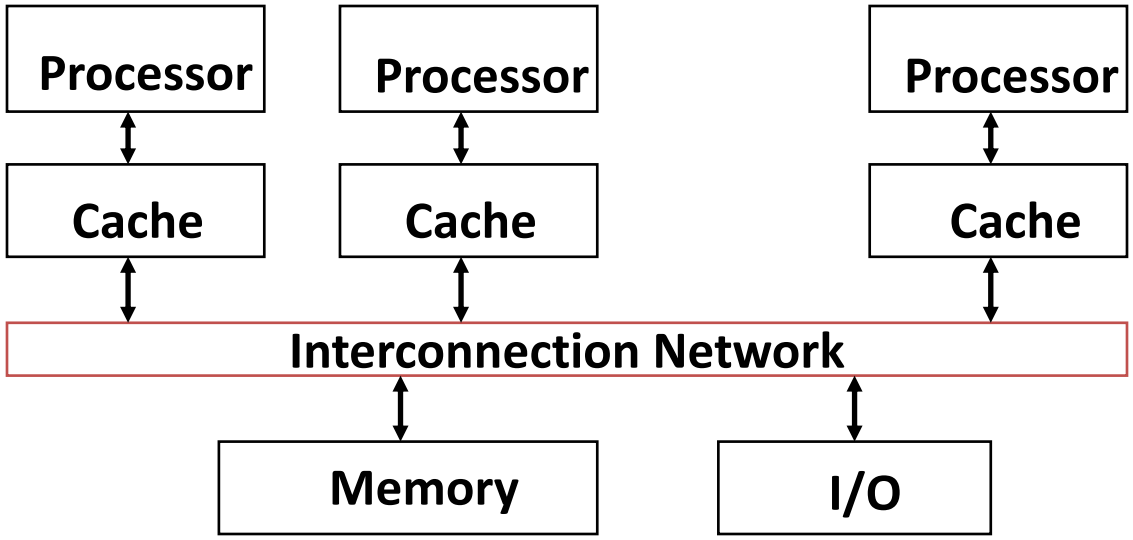
Slides based on UC Berkley's CS61C, CS152 and CS252

Review: Simple Multi-core Processor



Review: Multiprocessor Caches

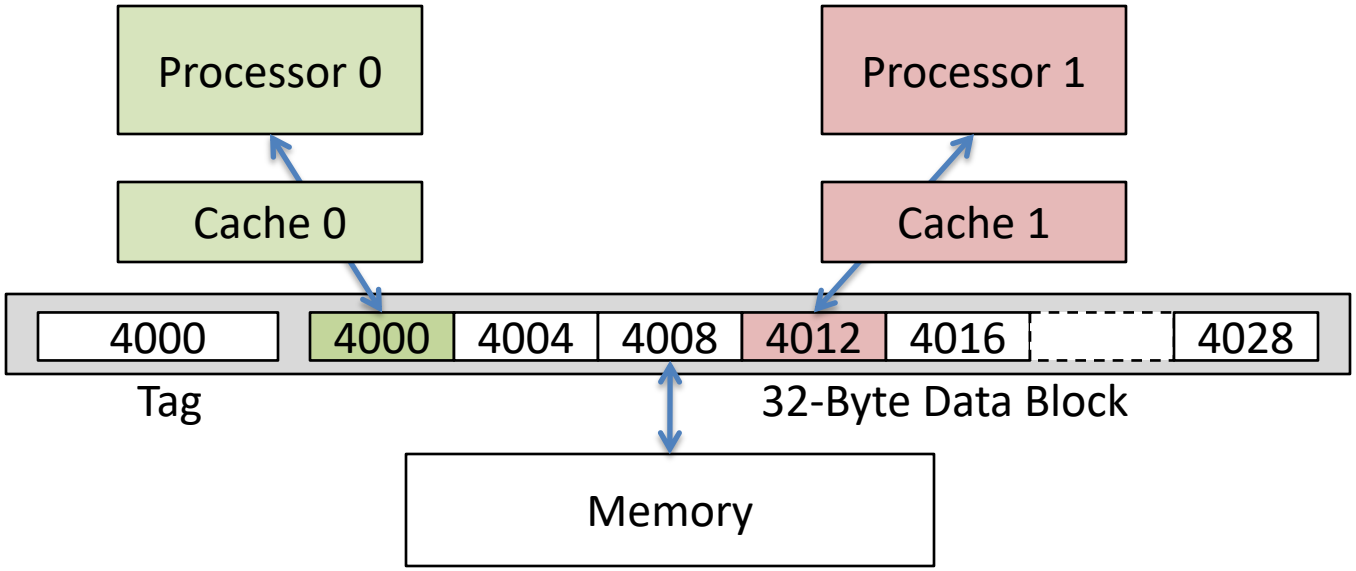
- Memory is a performance bottleneck even with one processor
- Use caches to reduce bandwidth demands on main memory
- Each core has a local private cache holding data it has accessed recently
- Only cache misses have to access the shared common memory



Review: Keeping Multiple Caches Coherent

- Architect's job: shared memory
  => keep cache values coherent
- Idea: When any processor has cache miss or writes, notify other processors via interconnection network
  - If only reading, many processors can have copies
  - If a processor writes, invalidate any other copies
- Write transactions from one processor, other caches "snoop" the common interconnect checking for tags they hold
  - Invalidate any copies of same address modified in other cache

Review: Cache Coherency Tracked by Block



- Suppose block size is 32 bytes
- Suppose Processor 0 reading and writing variable X, Processor 1 reading and writing variable Y
- Suppose in X location 4000, Y in 4012
- What will happen?

Review: Understanding Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to block, impossible to avoid; small effect for long-running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity (not compulsory and…)
  - Cache cannot contain all blocks accessed by the program ***even with perfect replacement policy in fully associative cache***
  - Solution: increase cache size (may increase access time)
- Conflict (not compulsory or capacity and…):
  - Multiple memory locations map to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)
  - Solution 3: improve replacement policy, e.g.. LRU

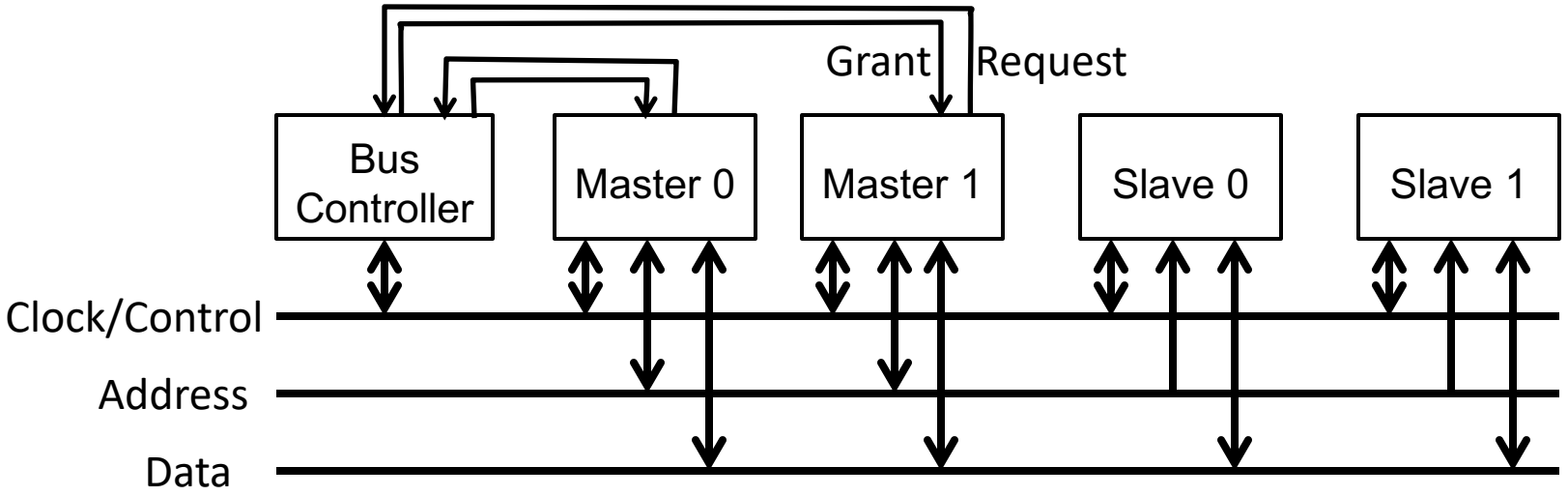
Review: Coherency Tracked by Cache Block

- Block ping-pongs between two caches even though processors are accessing disjoint variables
- Effect called *false sharing*
- How can you prevent it?

Review: Fourth "C" of Cache Misses: *Coherence* Misses

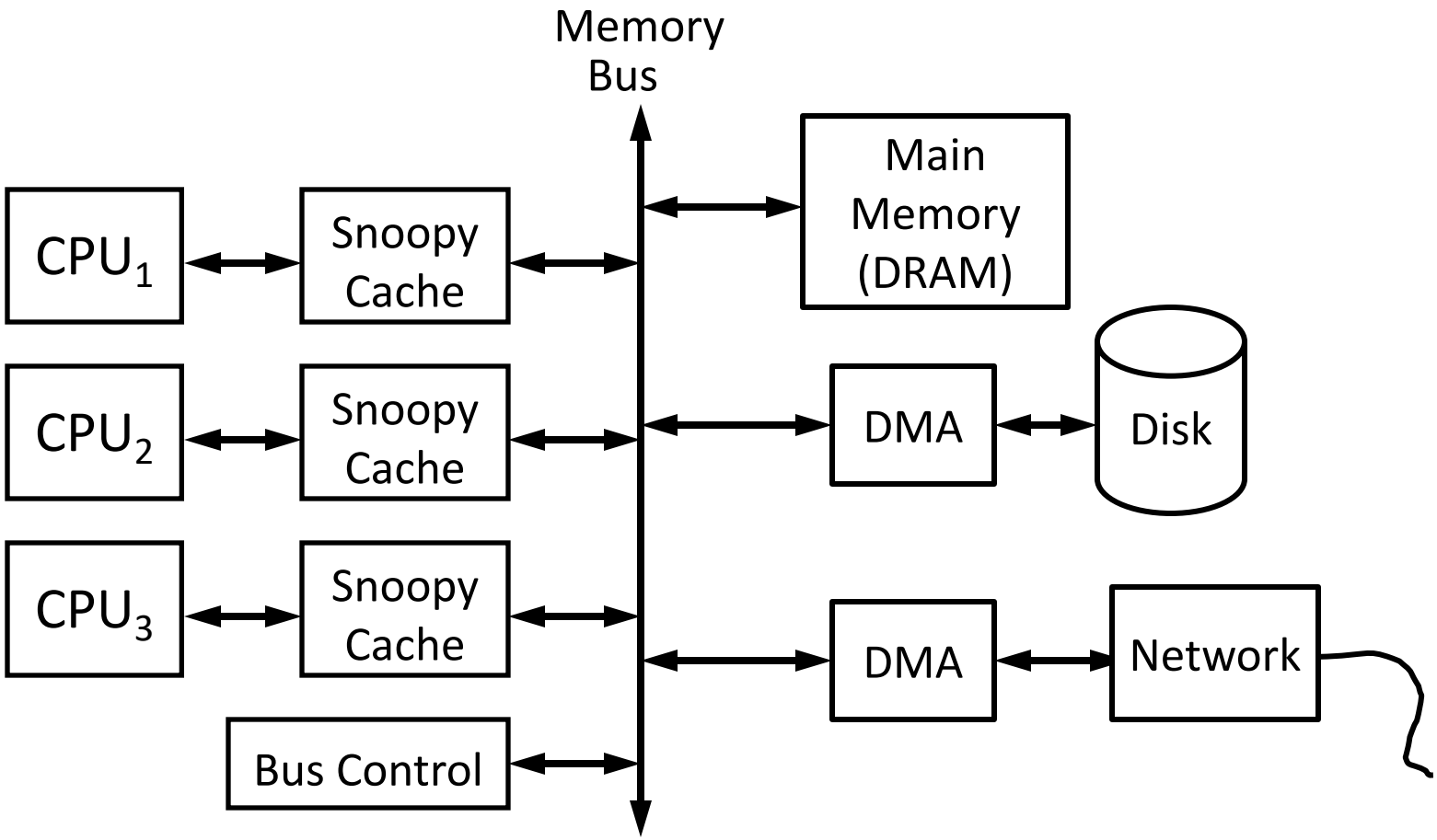
- Misses caused by coherence traffic with other processor
- Also known as *communication* misses because represents data moving between processors working together on a parallel program
- For some parallel programs, coherence misses can dominate total misses

Bus Management



- A “bus” is a collection of shared wires
  - Newer “busses” use point-point links
- Only one “master” can initiate a transaction by driving wires at any one time
- Multiple “slaves” can observe and conditionally respond to the transaction on the wires
  - slaves decode address on bus to see if they should respond (memory is most common slave)
  - some masters can also act as slaves
- Masters arbitrate for access with requests to bus “controller”
  - Some busses only allow one master (in which case, it’s also the controller)

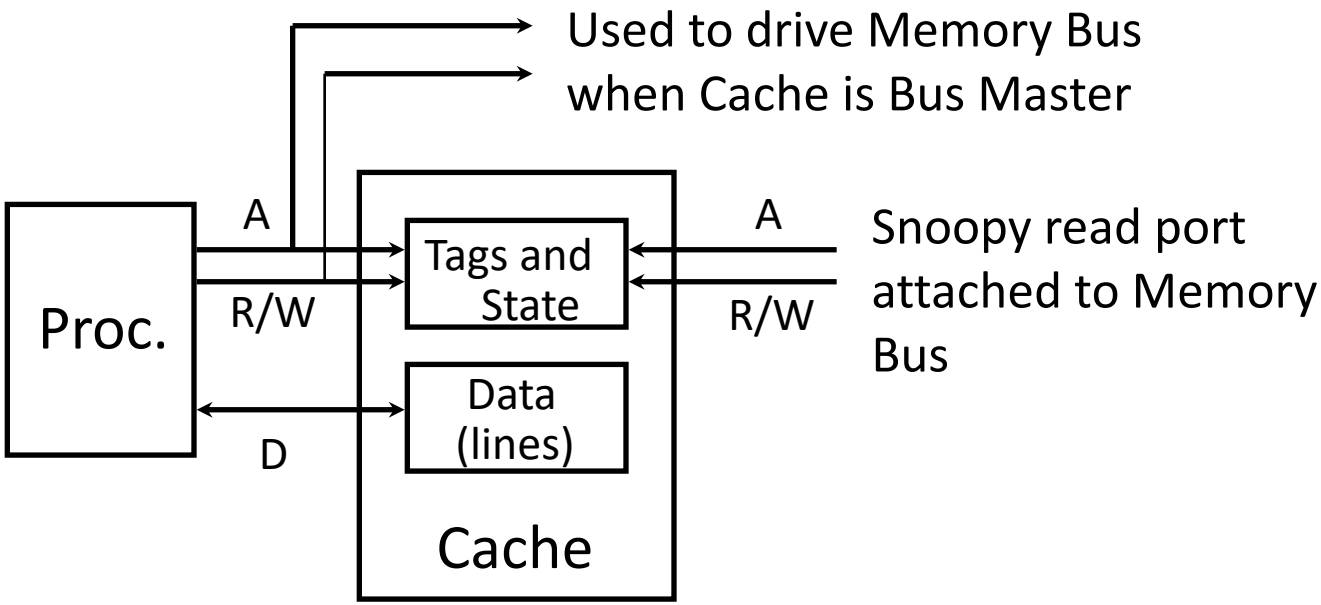
Shared-Memory Multiprocessor



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

Snoopy Cache, *Goodman 1983*

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



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

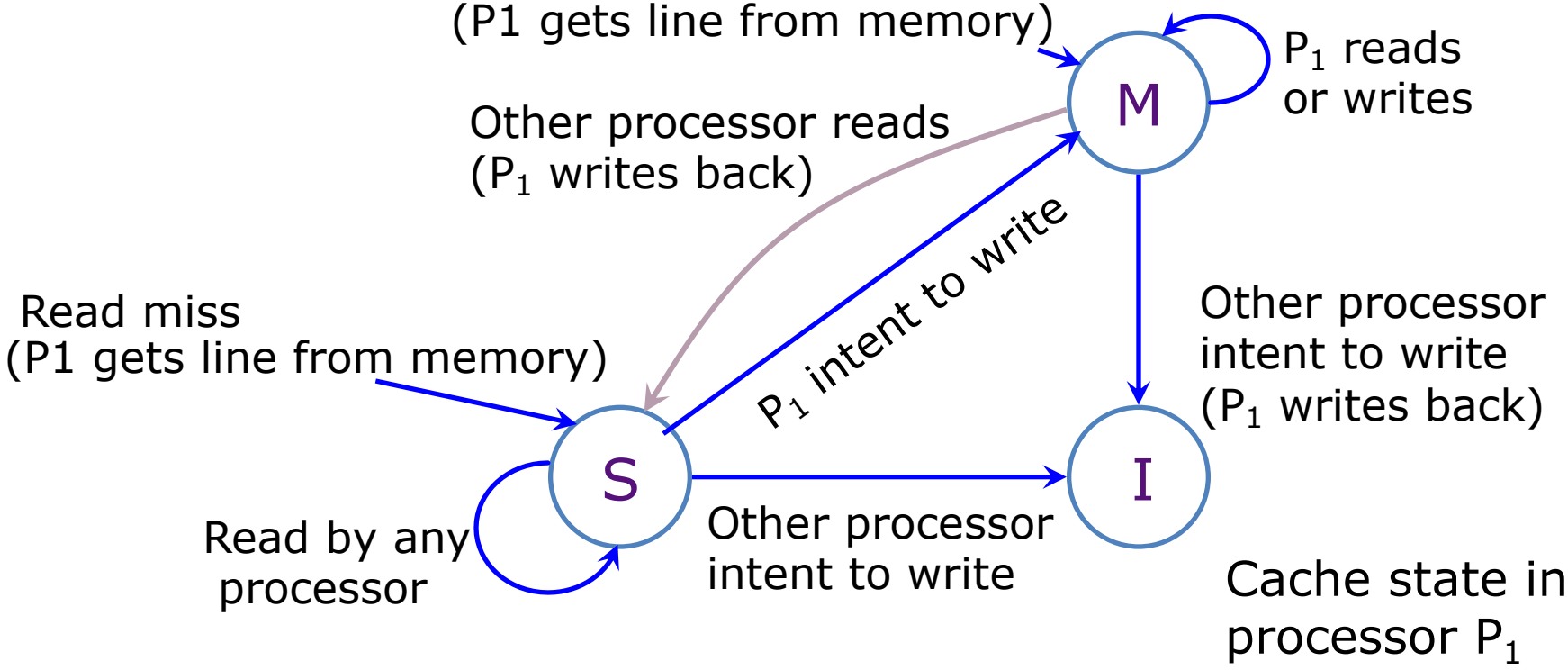
Cache State-Transition Diagram

The MSI protocol

Each cache line has state bits

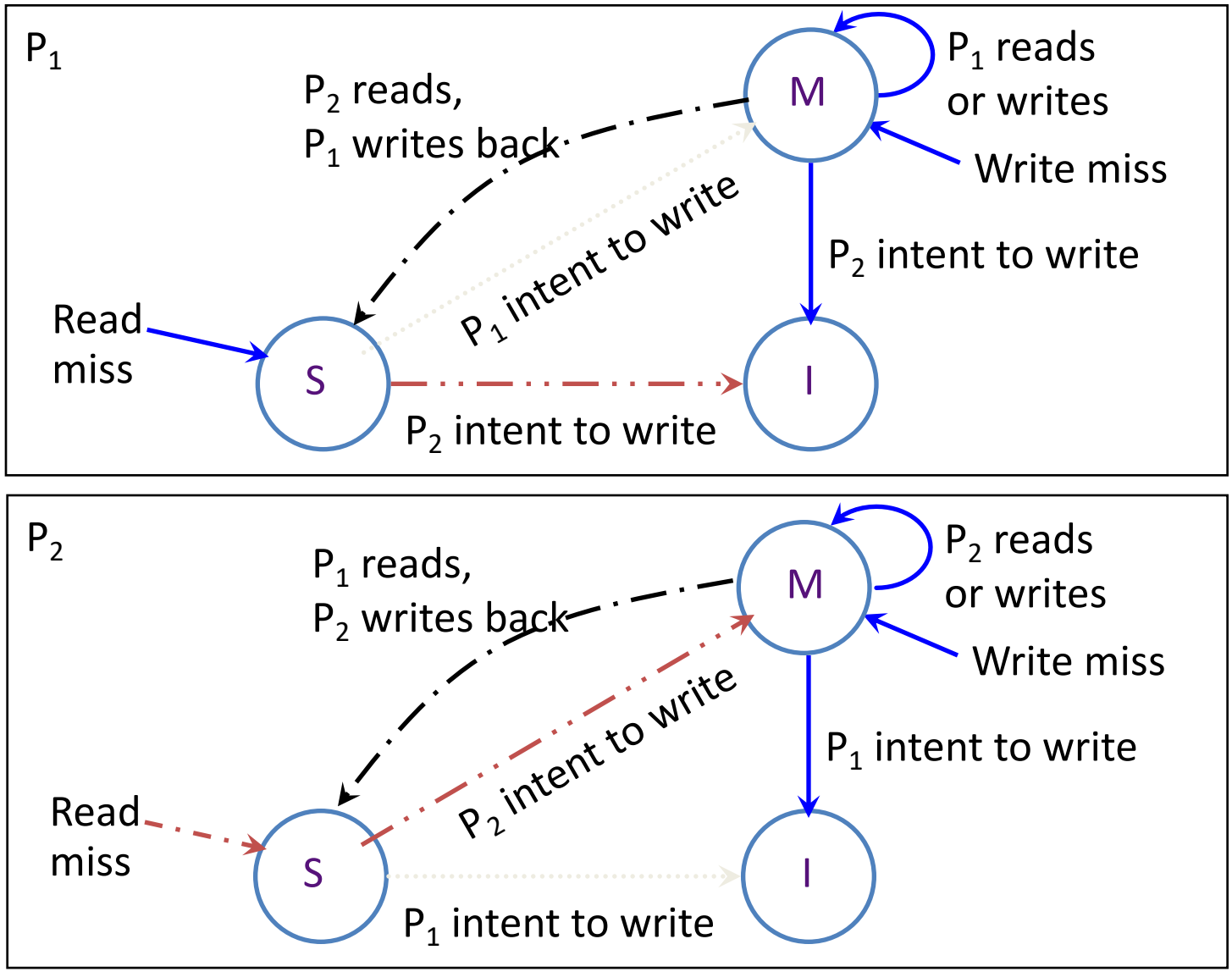
state bits

M: Modified
S: Shared
I: Invalid

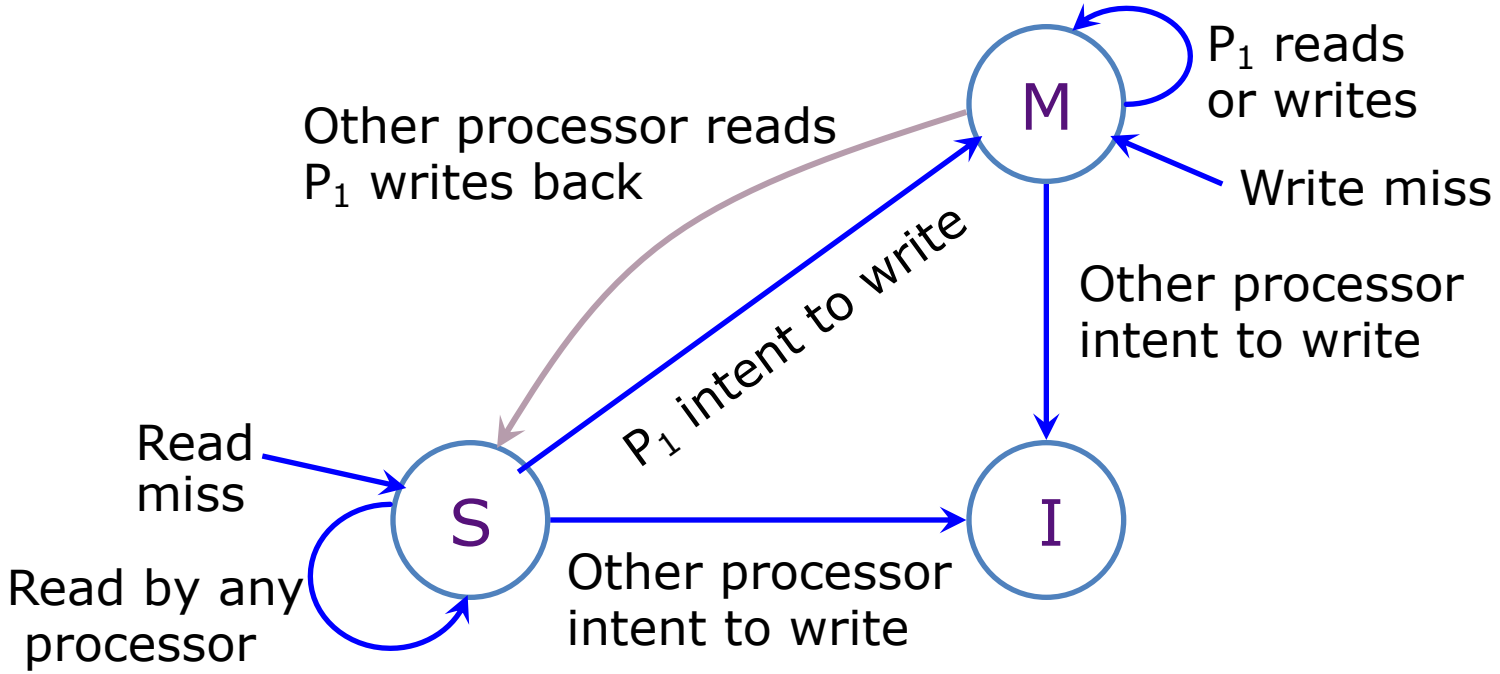


Two-Processor Example (Reading and writing the same cache line)

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



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

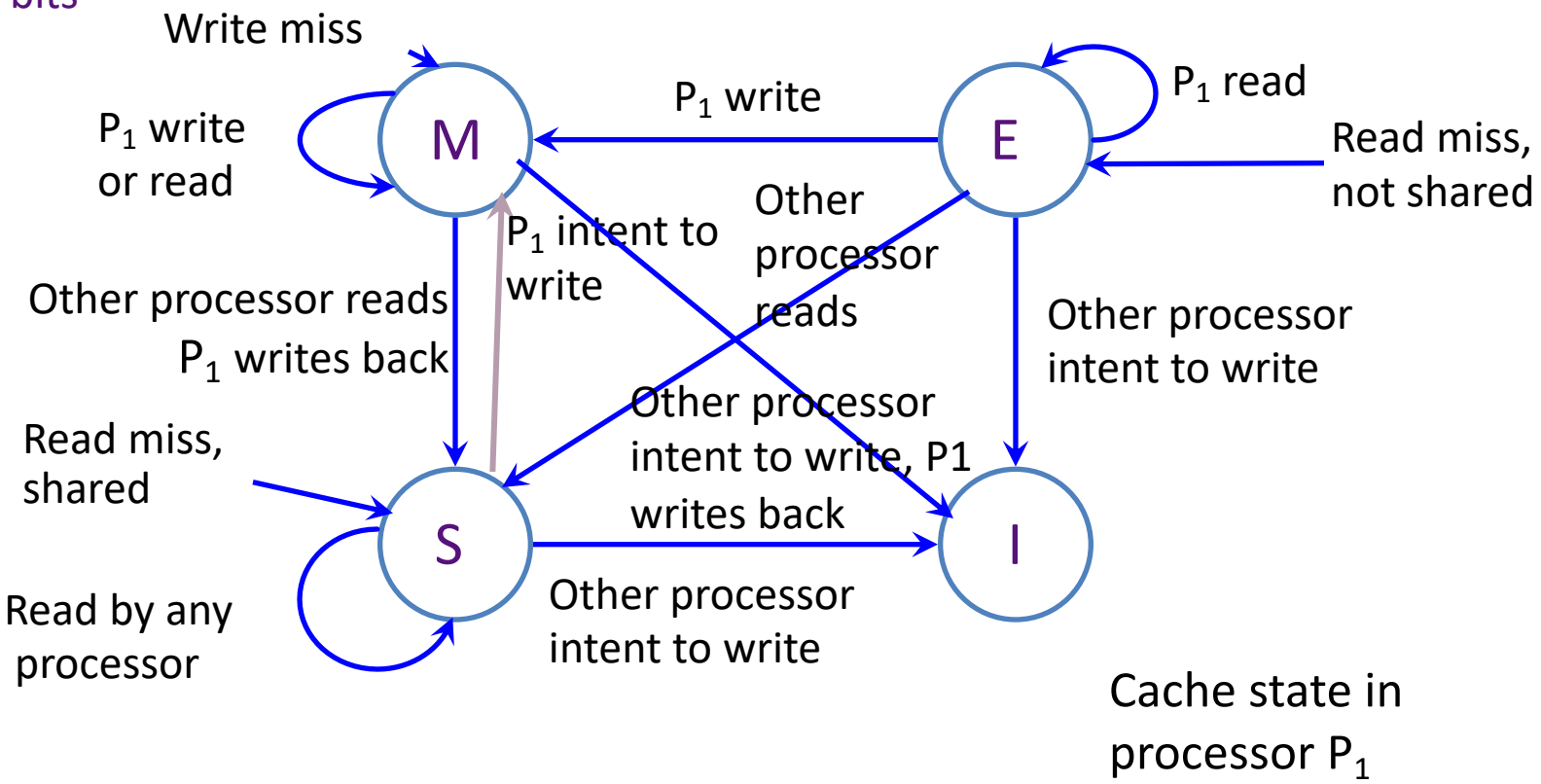
MESI: An Enhanced MSI protocol

increased performance for private data

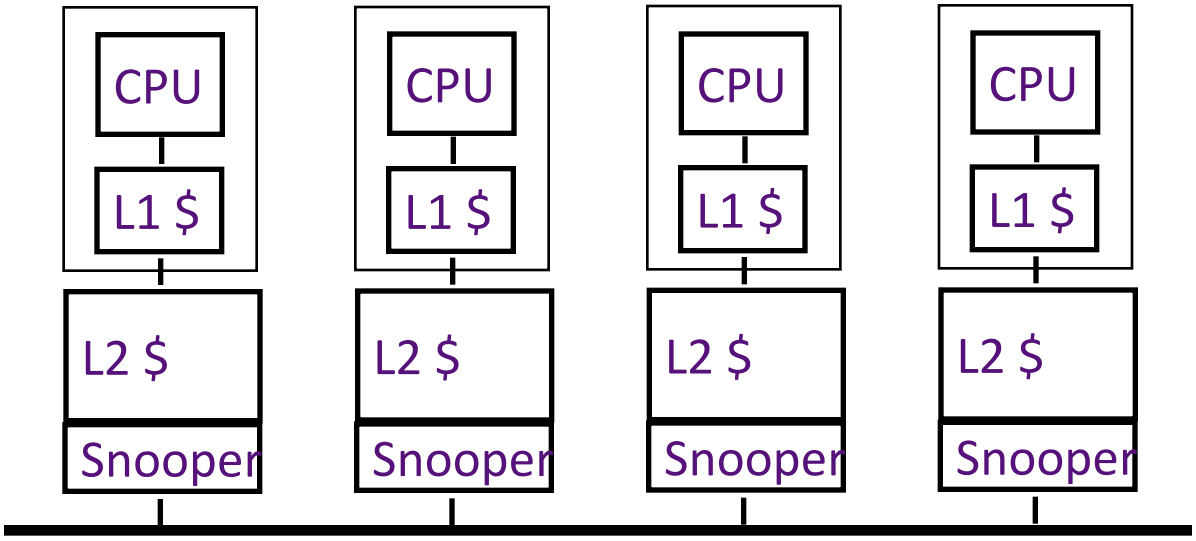
Each cache line has a tag

state bits

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

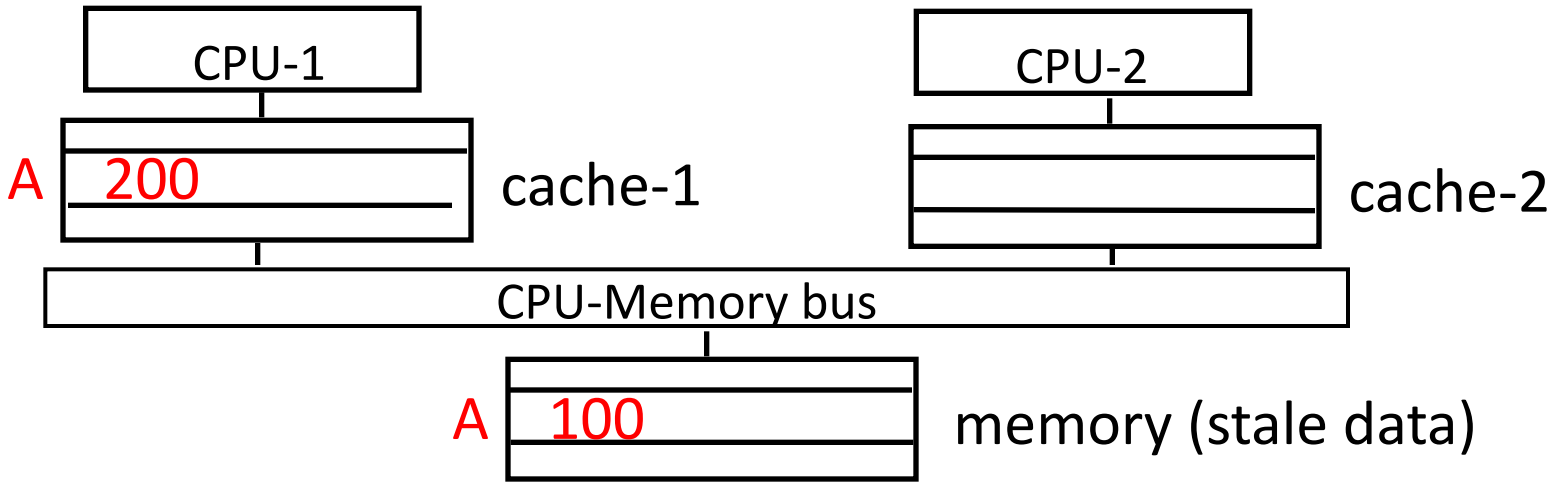


Optimized Snoop with Level-2 Caches



- 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

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

A cache line contains more than one word

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

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

What can happen?

Performance of Symmetric Multiprocessors (SMPs)

Cache performance is combination of:

- Uniprocessor cache miss traffic
- Traffic caused by communication
  - Results in invalidations and subsequent cache misses
- Coherence misses
  - Sometimes called a Communication miss
    - Read miss: remote core write
    - Write miss: remote core write or read
  - 4th C of cache misses along with Compulsory, Capacity, & Conflict.

Coherency Misses

- True sharing misses arise from the communication of data through the cache coherence mechanism
  - Invalidates due to 1st write to shared line
  - Reads by another CPU of modified line in different cache
  - Miss would still occur if line size were 1 word
- False sharing misses when a line is invalidated because some word in the line, other than the one being read, is written into
  - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
  - Line is shared, but no word in line is actually shared
    $\Rightarrow$ miss would not occur if line size were 1 word

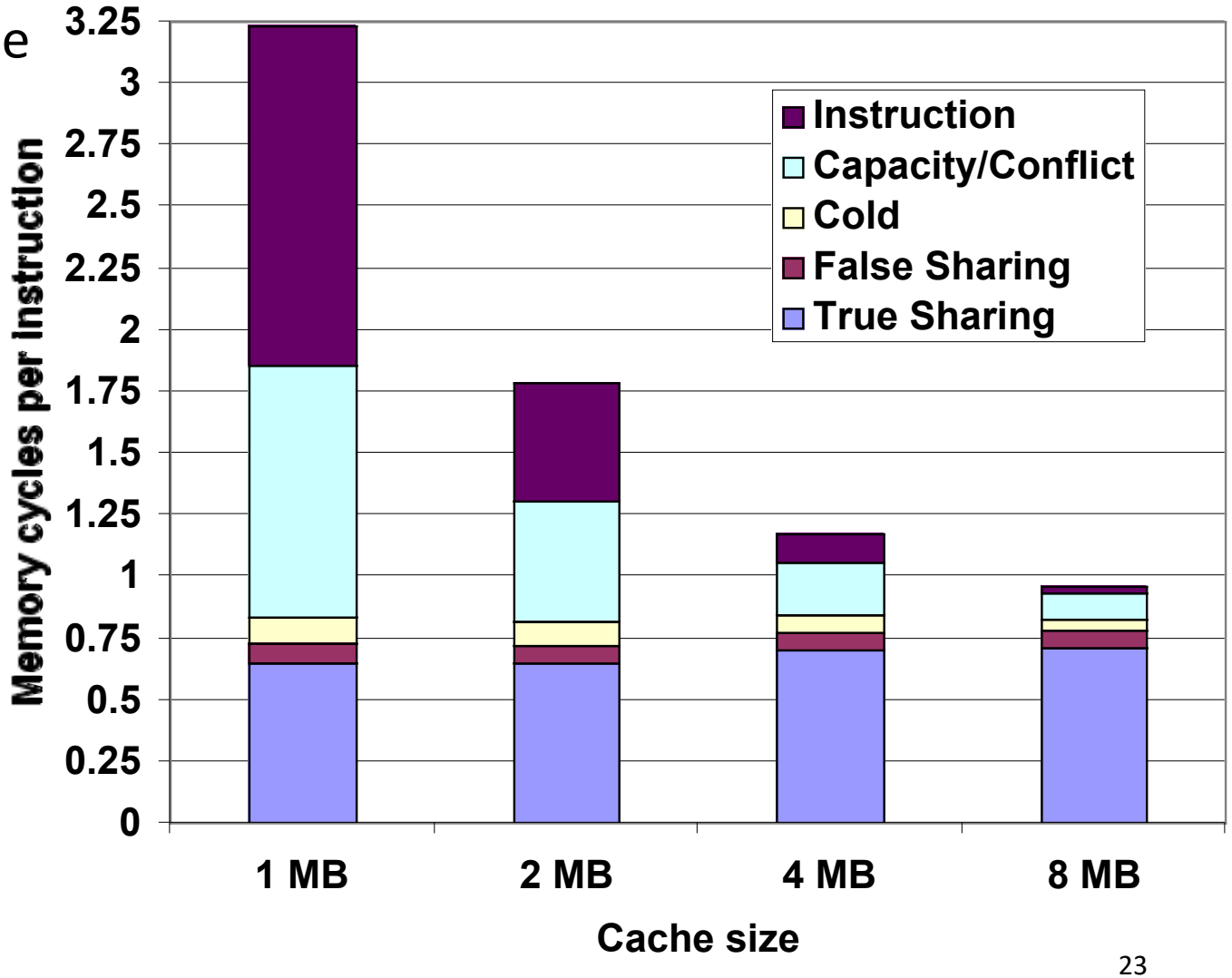
Example: True v. False Sharing v. Hit?

- Assume x1 and x2 in same cache line.
  P1 and P2 both read x1 and x2 before.

Time	P1	P2	True, False, Hit? Why?
1	Write x1		True miss; invalidate x1 in P2
2		Read x2	False miss; x1 irrelevant to P2
3	Write x1		False miss; x1 irrelevant to P2
4		Write x2	True miss; x2 not writeable
5	Read x2		True miss; invalidate x2 in P1

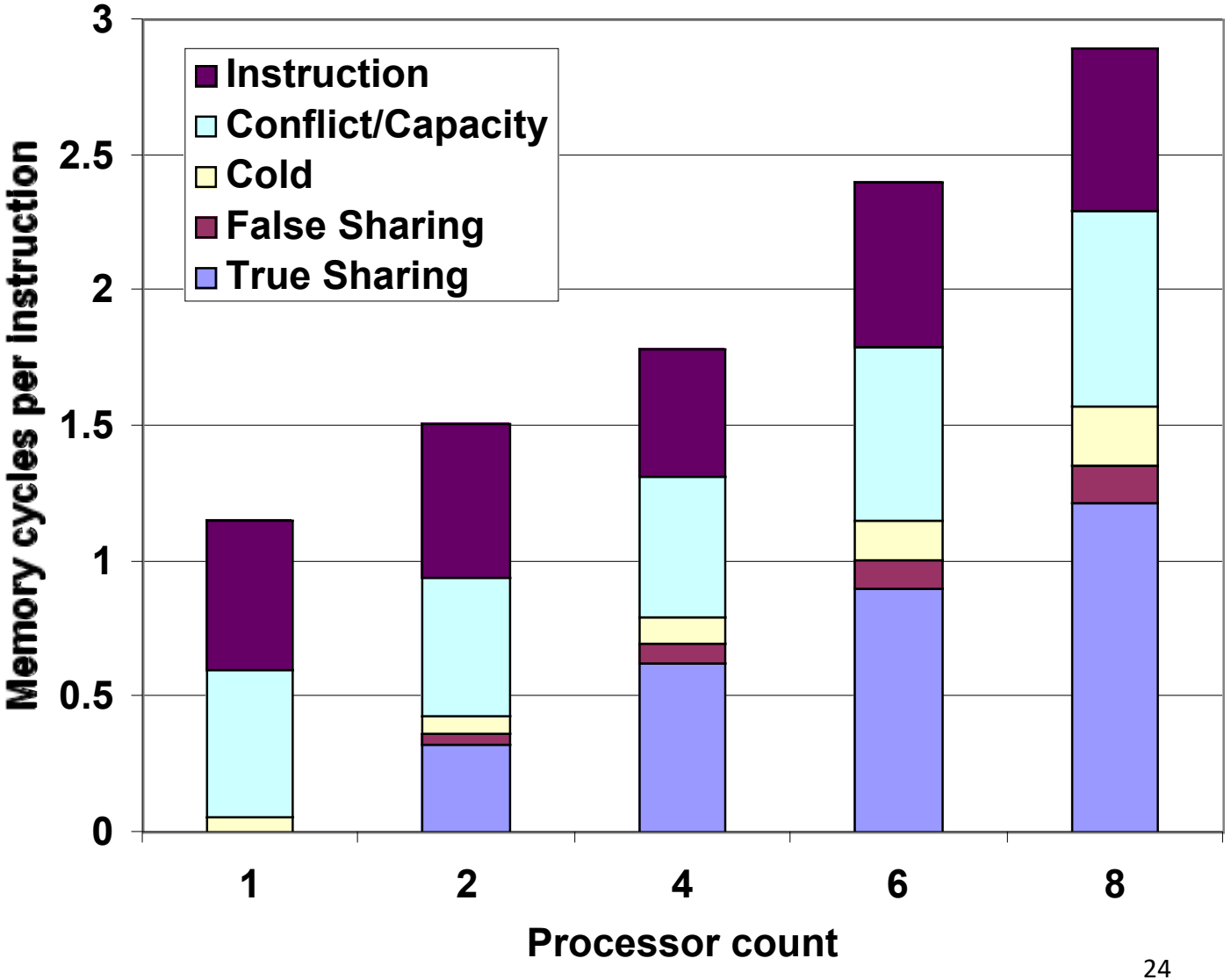
MP Performance 4-Processor Commercial Workload: OLTP, Decision Support (Database), Search Engine

- Uniprocessor cache misses improve with cache size increase (Instruction, Capacity/Conflict, Compulsory)
- True sharing and false sharing unchanged going from 1 MB to 8 MB (L3 cache)



MP Performance 2MiB Cache Commercial Workload: OLTP, Decision Support (Database), Search Engine

- True sharing, false sharing increase going from 1 to 8 CPUs



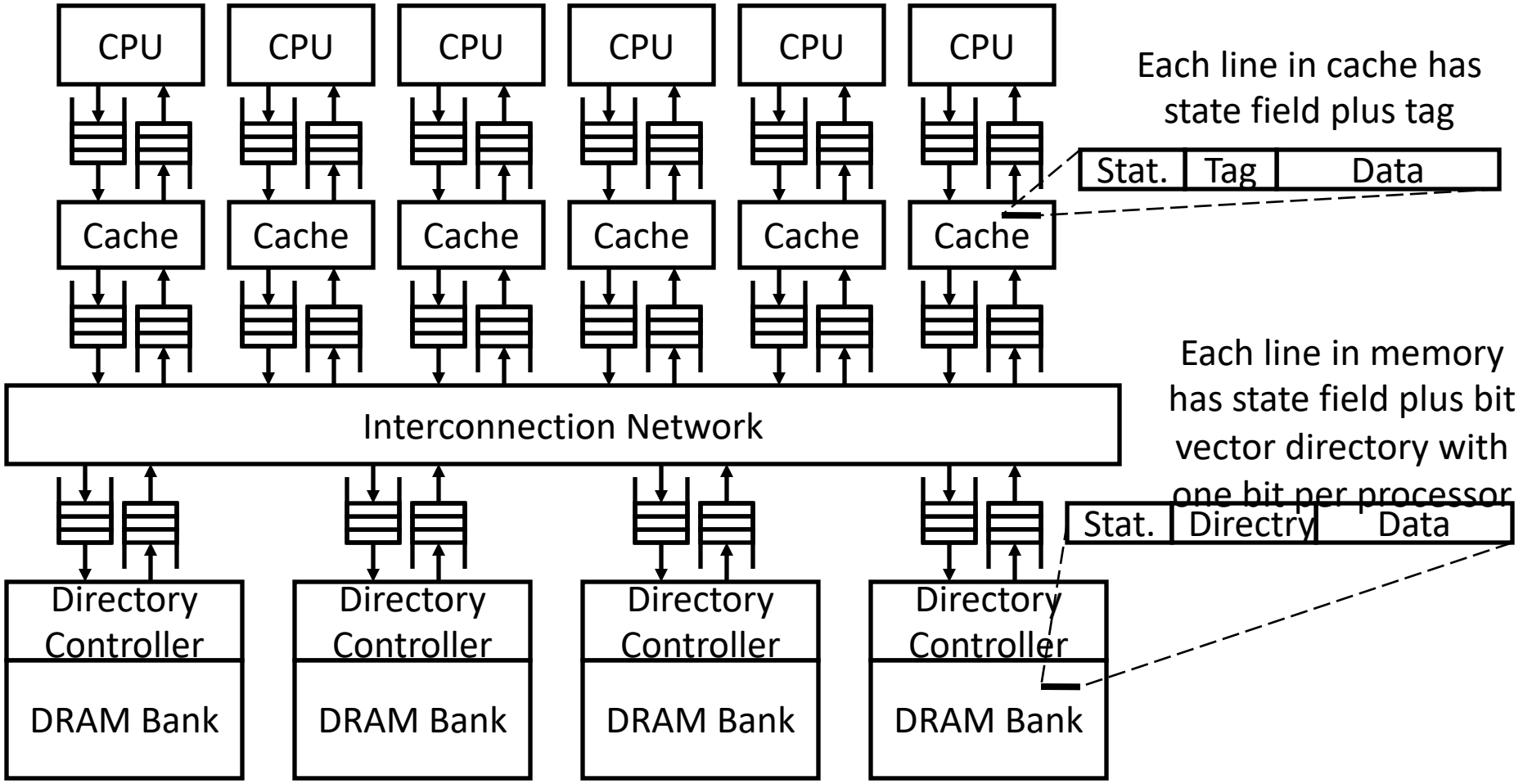
Scaling Snoopy/Broadcast Coherence

- When any processor gets a miss, must probe every other cache
- Scaling up to more processors limited by:
  - Communication bandwidth over bus
  - Snoop bandwidth into tags
- Can improve bandwidth by using multiple interleaved buses with interleaved tag banks
  - E.g, two bits of address pick which of four buses and four tag banks to use – (e.g., bits 7:6 of address pick bus/tag bank, bits 5:0 pick byte in 64-byte line)
- Buses don't scale to large number of connections, so can use point-to-point network for larger number of nodes, but then limited by tag bandwidth when broadcasting snoop requests.
- **Insight**: Most snoops fail to find a match!

Scalable Approach: Directories

- Can use point-to-point network for larger number of nodes, but then limited by tag bandwidth when broadcasting snoop requests
- Every memory line has associated directory information
  - keeps track of copies of cached lines 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

Directory Cache Protocol



- Assumptions: Reliable network, FIFO message delivery between any given source-destination pair

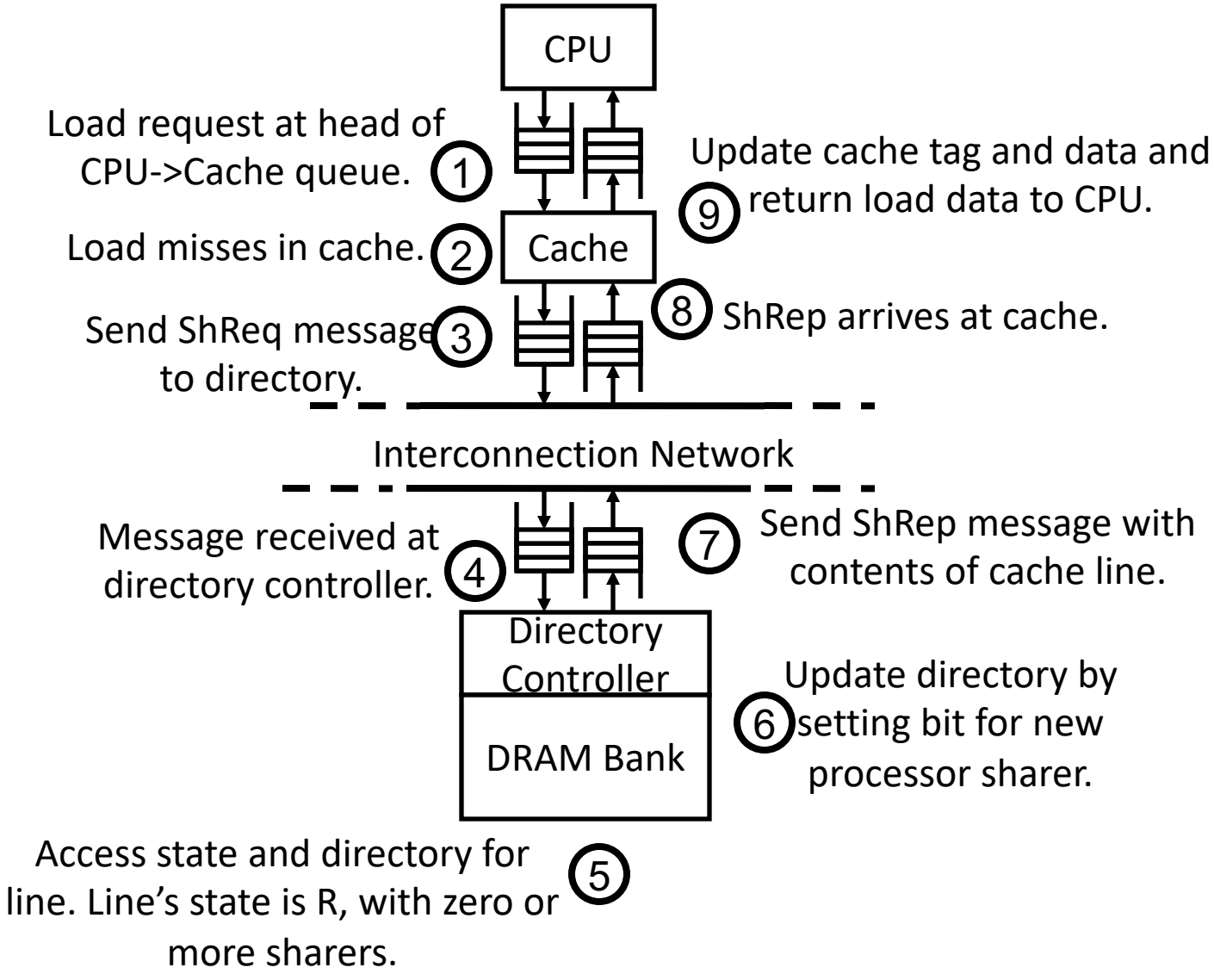
Cache States

- For each cache line, there are 4 possible states:
  - **C-invalid** (= Nothing): The accessed data is not resident in the cache.
  - **C-shared** (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.
  - **C-modified** (= Ex): The accessed data is exclusively resident in this cache, and has been modified. Memory does not have the most up-to-date data.
  - **C-transient** (= Pending): The accessed data is in a transient state (for example, the site has just issued a protocol request, but has not received the corresponding protocol reply).

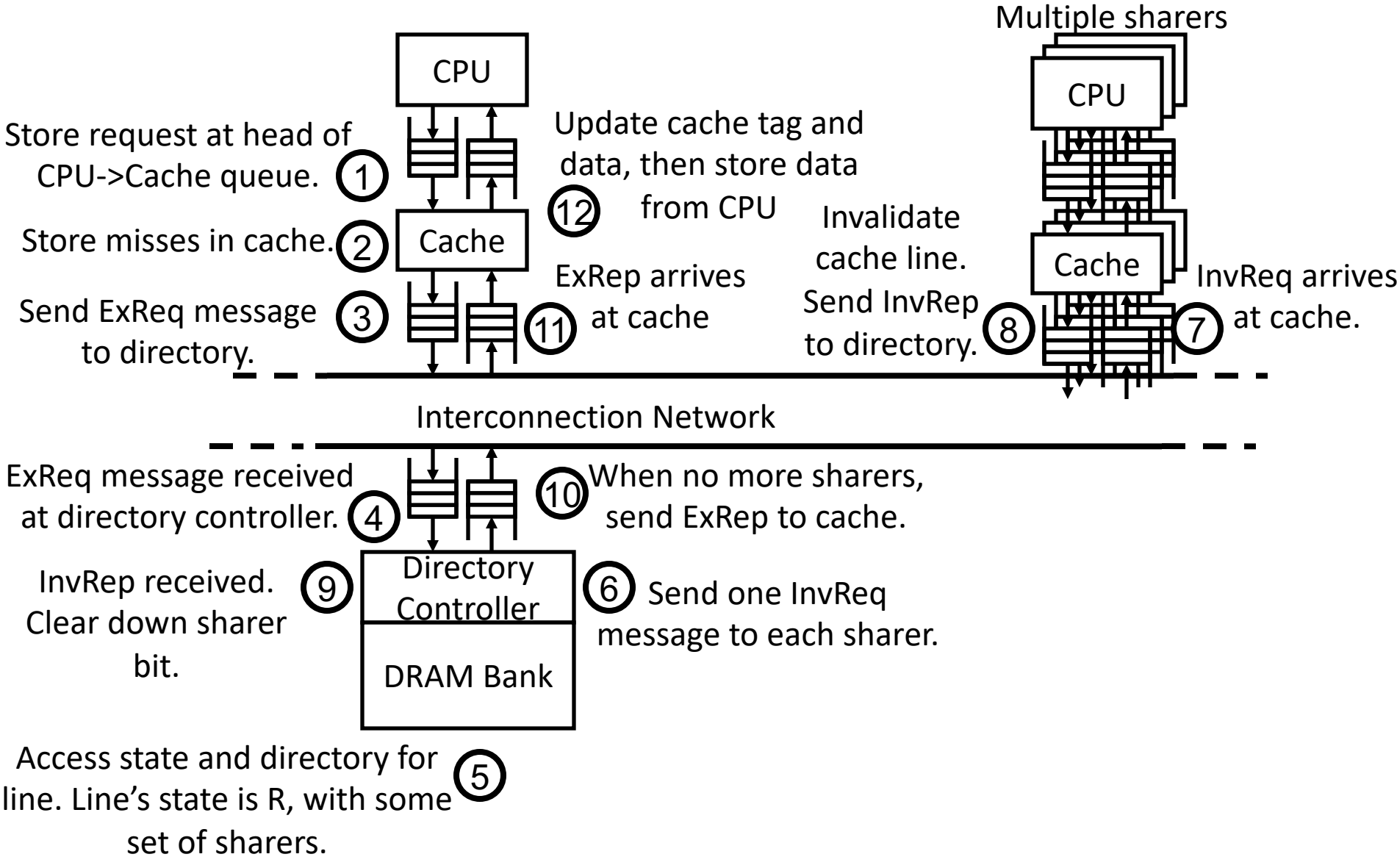
Home directory states

- For each memory line, there are 4 possible states:
  - **R(dir):** The memory line is shared by the sites specified in dir (dir is a set of sites). The data in memory is valid in this state. If dir is empty (i.e., dir = ε), the memory line is not cached by any site.
  - **W(id):** The memory line is exclusively cached at site id, and has been modified at that site. Memory does not have the most up-to-date data.
  - **TR(dir):** The memory line is in a transient state waiting for the acknowledgements to the invalidation requests that the home site has issued.
  - **TW(id):** The memory line is in a transient state waiting for a line exclusively cached at site id (i.e., in C-modified state) to make the memory line at the home site up-to-date.

Read miss, to uncached or shared line



Write miss, to read shared line



Concurrency Management

- Protocol would be easy to design if only one transaction in flight across entire system
- But, want greater throughput and don't want to have to coordinate across entire system
- Great complexity in managing multiple outstanding concurrent transactions to cache lines
  - Can have multiple requests in flight to same cache line!