上海科技大学
ShanghaiTech University

CS211
Advanced Computer Architecture

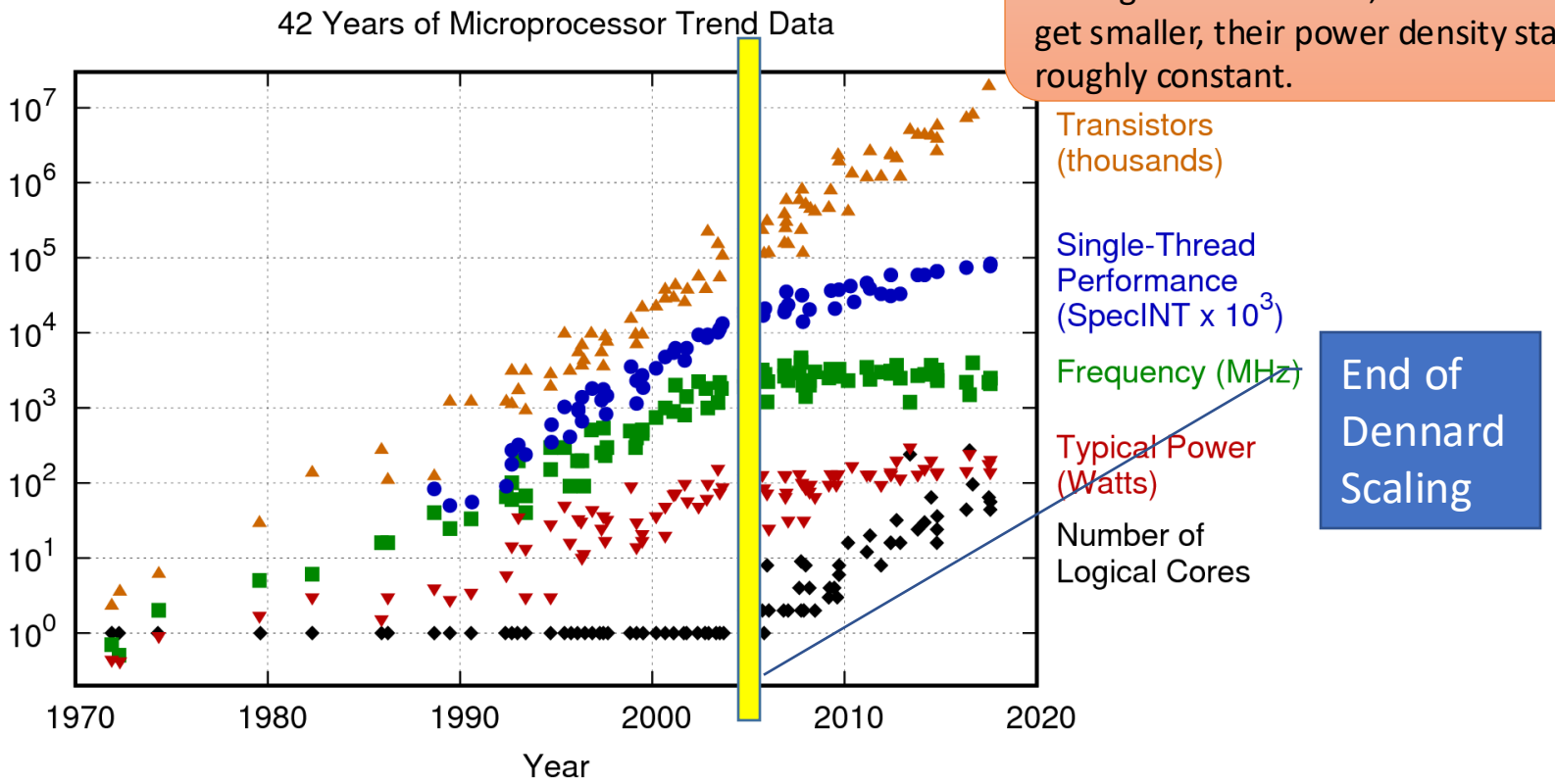
L15 Cache Coherence

Chundong Wang

November 19th, 2025

The Shift to Multicore

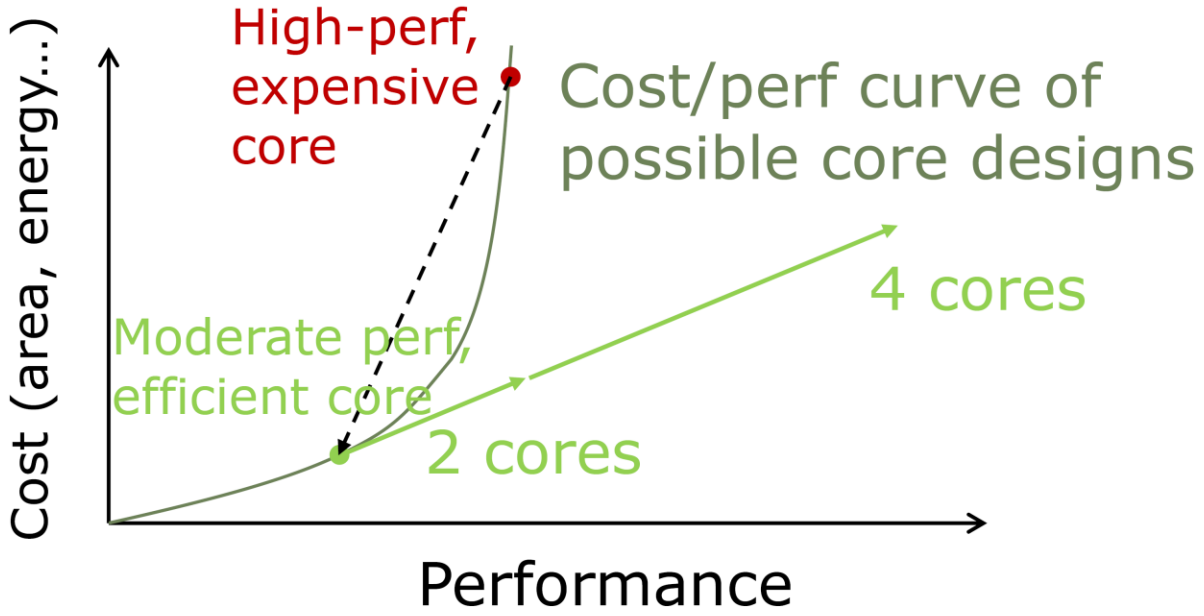
Dennard scaling refers to the reduction of MOS supply voltage in concert with the scaling of feature sizes, so that as transistors get smaller, their power density stays roughly constant.



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp [https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/]

- Since 2005, improvements in system performance mainly due to increasing cores per chip
- Why? Technology scaling
  Limited instruction-level parallelism

Multicore Performance



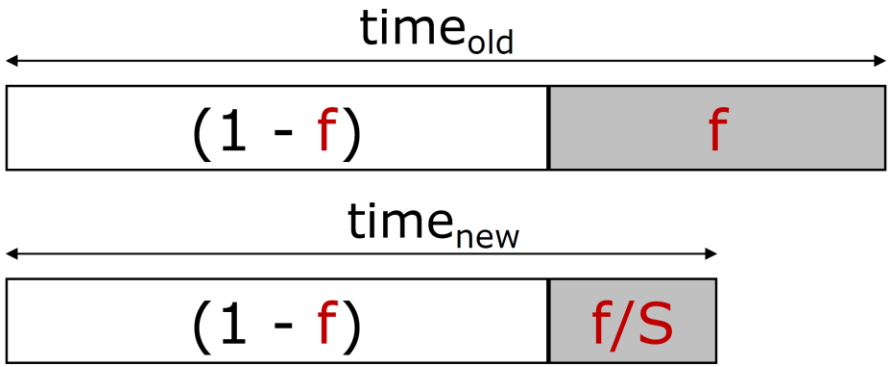
- *What factors may limit multicore performance?*
  - Limited application parallelism
  - Memory accesses and inter-core communication
  - Programming complexity

Amdahl's Law

- Speedup= $time_{without\ enhancement}$ / $time_{with\ enhancement}$
- Suppose an enhancement speeds up a fraction *f* of a task by a factor of *S*

$$time_{new} = time_{old} \cdot ((1-f) + f/S) \rightarrow$$

$$S_{overall} = 1 / ((1-f) + f/S)$$



Corollary: Make the common case fast

Amdahl's Law and Parallelism

- Say you write a program that can do 90% of the work in parallel, but the other 10% is sequential
- What is the maximum speedup you can get by running on a multicore machine?

- What f do you need to use a 1000-core machine well?

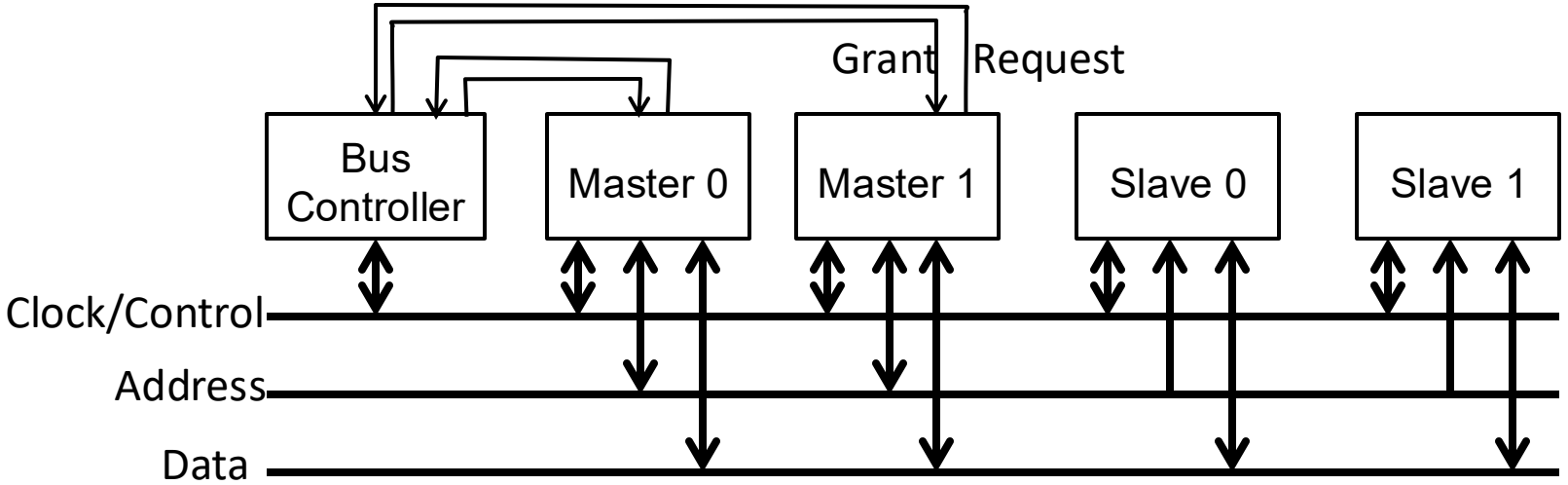
Coherence & Consistency

- Shared memory systems:
  - Have multiple private caches for performance reasons
  - Need to provide the illusion of a single shared memory
- Intuition: A read should return the most recently written value
  - What is "*most recent*"?
- Formally:
  - Coherence: What values can a read return?
    - Concerns reads/writes to a single memory location
  - Consistency: When do writes become visible to reads?
    - Concerns reads/writes to multiple memory locations

Implementing Cache Coherence

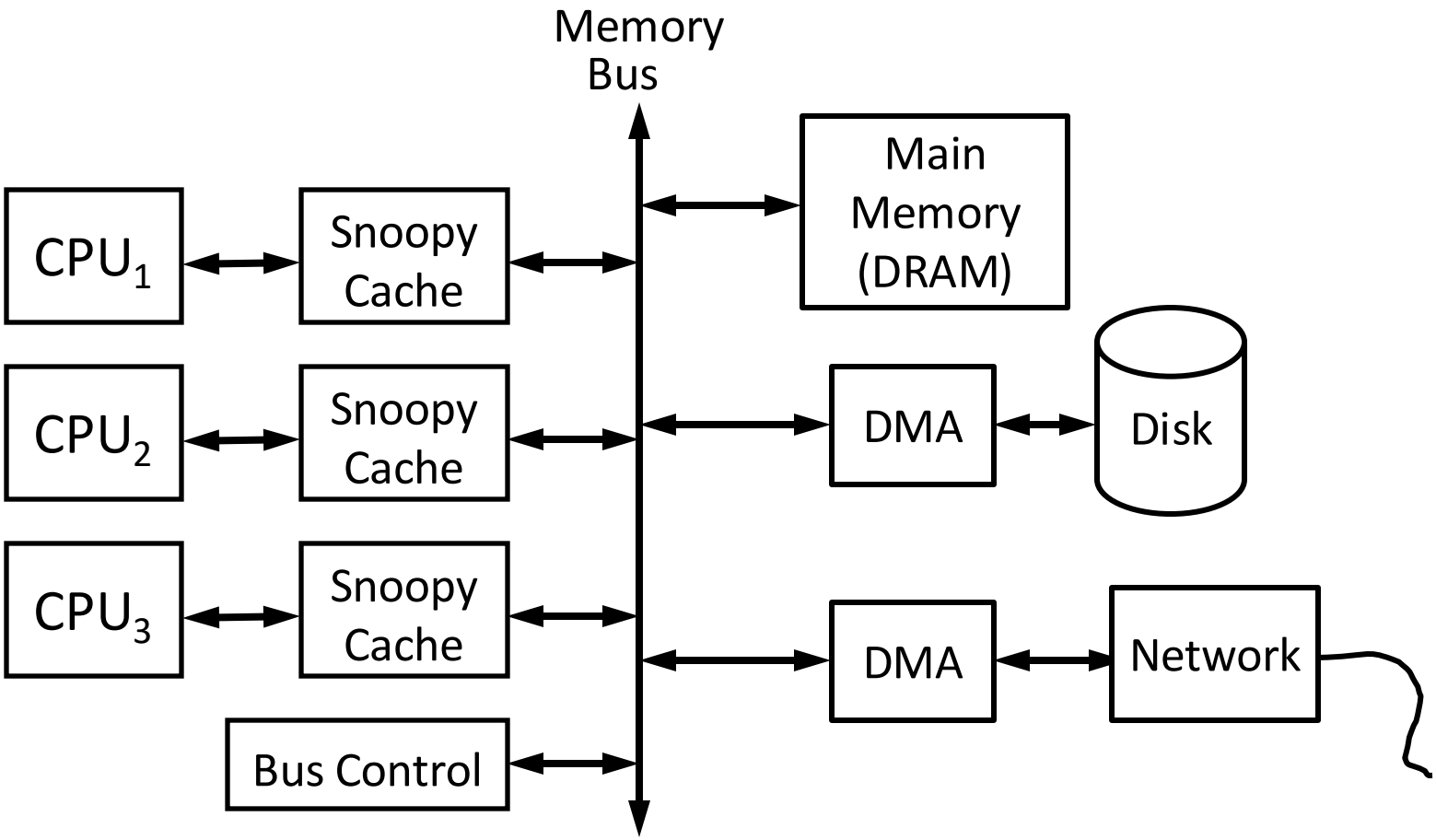
- Coherence protocols must enforce two rules:
  - Write propagation: Writes eventually become visible to all processors
  - Write serialization: Writes to the same location are serialized (all processors see them in the same order)
- How to ensure write propagation?
  - Write-invalidate protocols: Invalidate all other cached copies before performing the write
  - Write-update protocols: Update all other cached copies after performing the write
- How to track sharing state of cached data and serialize requests to the same address?
  - Snooping-based protocols: All caches observe each other's actions through a shared bus (bus is the serialization point)
  - Directory-based protocols: A coherence directory tracks contents of private caches and serializes requests (directory is the serialization point)

Bus Management



- A “bus” is a collection of shared wires
  - Newer “buses” 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 buses only allow one master (in which case, it’s also the controller)

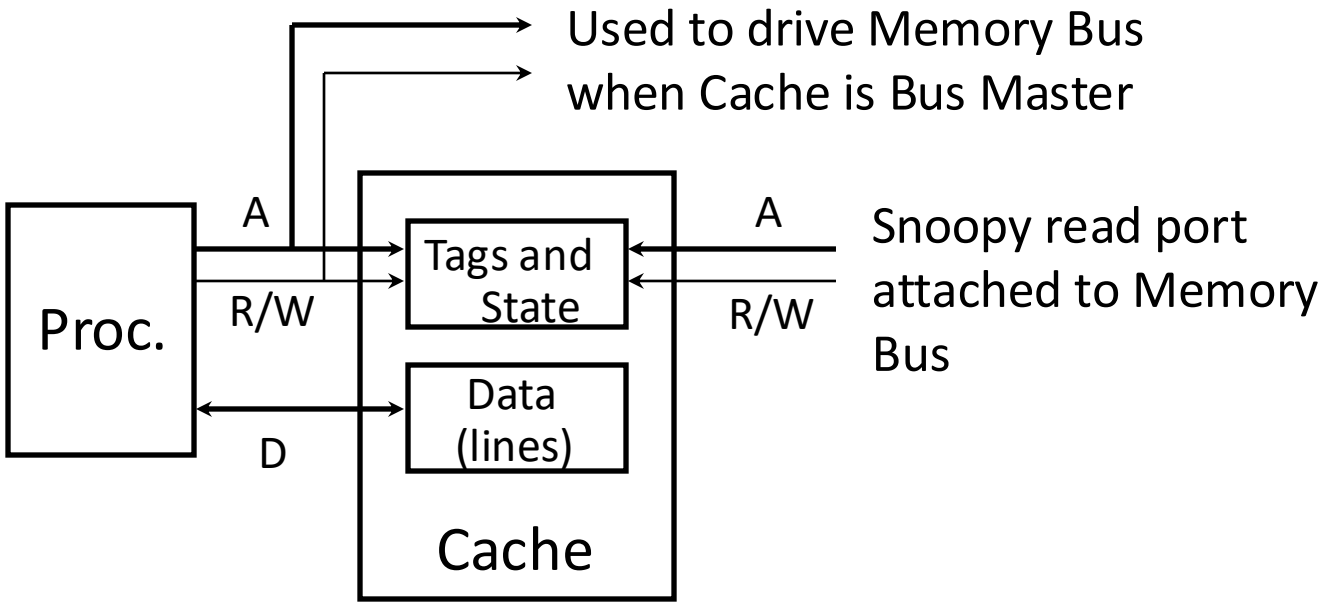
Shared-Memory Multiprocessor



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

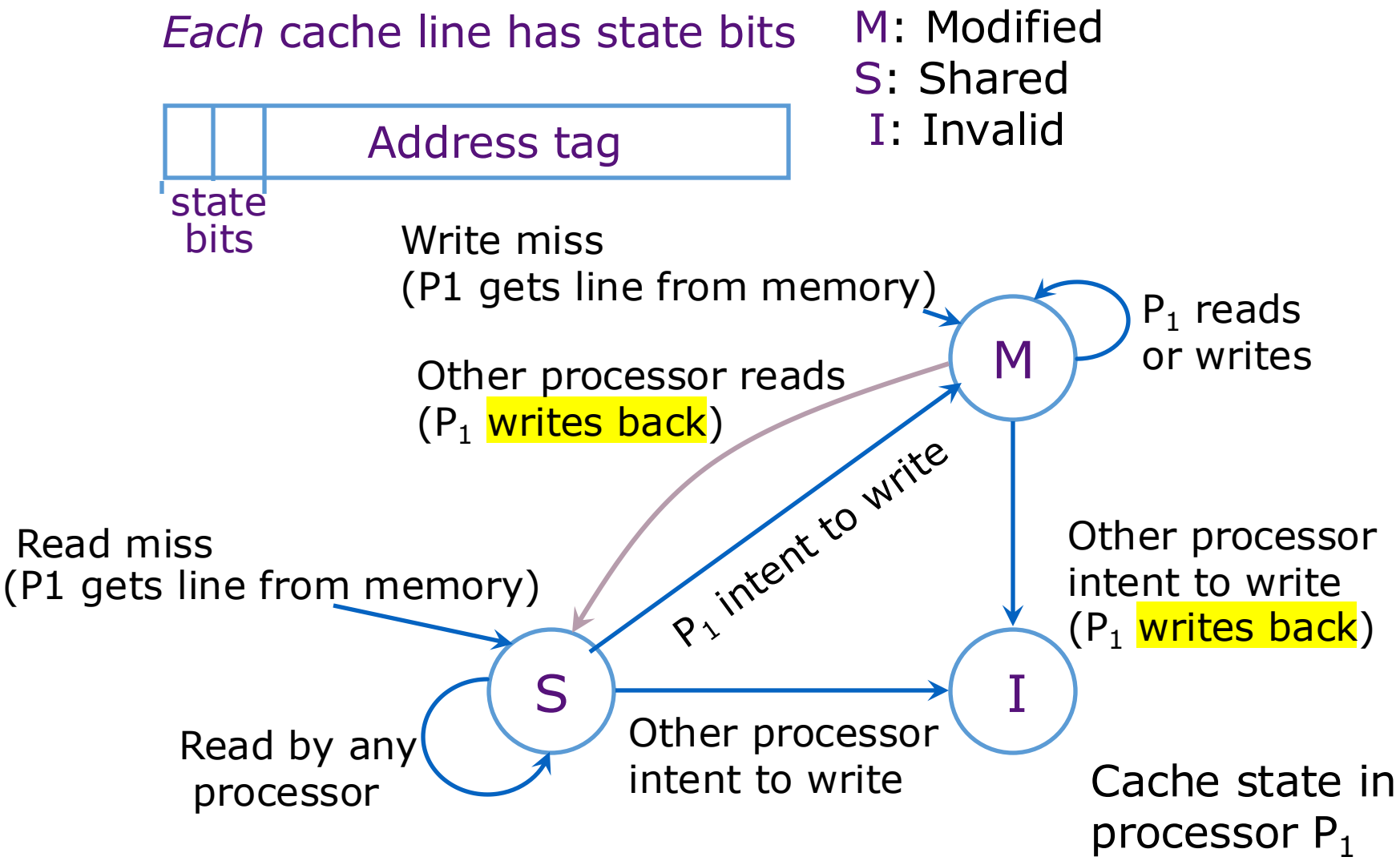


Snoopy Cache-Coherence Protocols

- 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

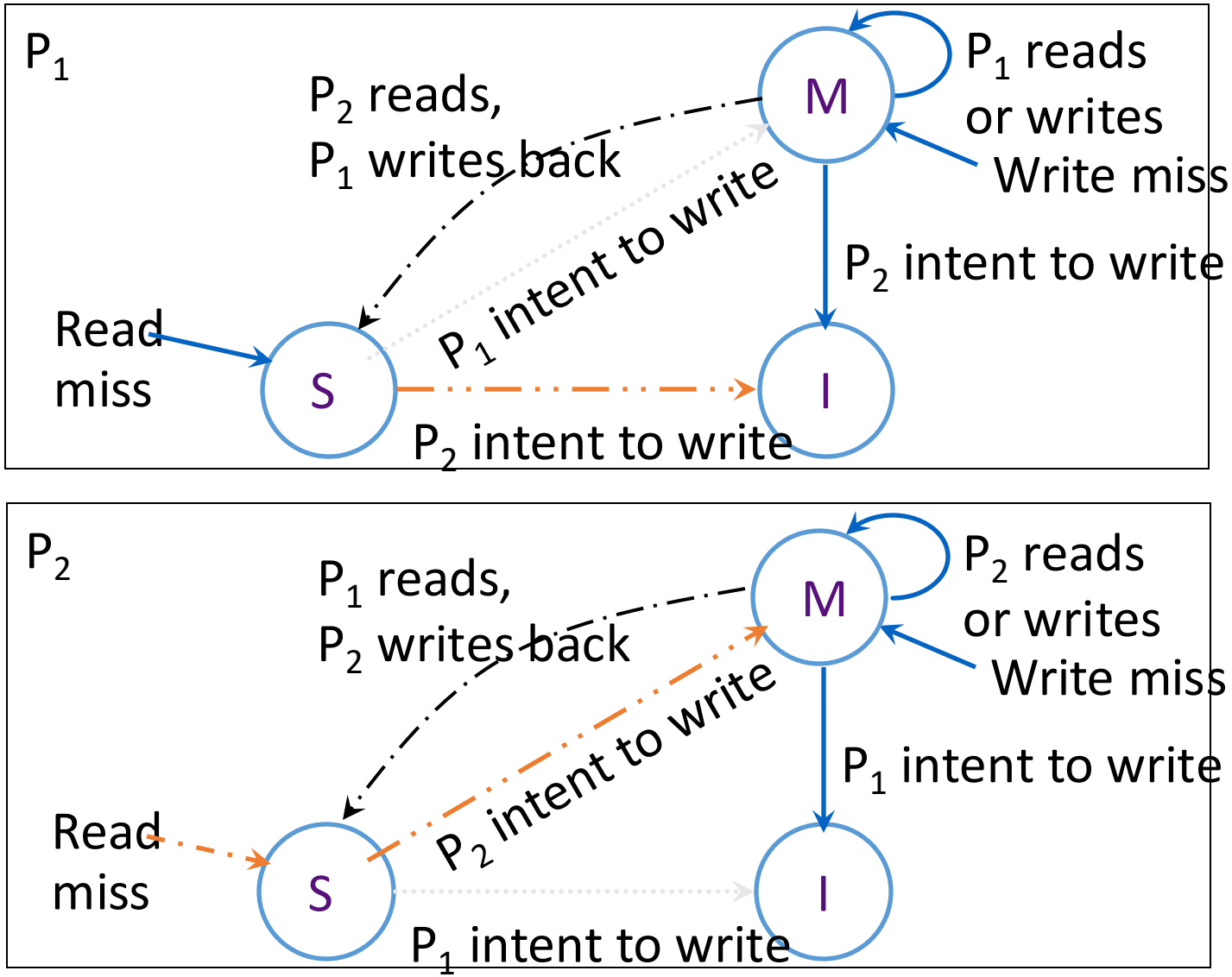
Cache State-Transition Diagram

The MSI protocol

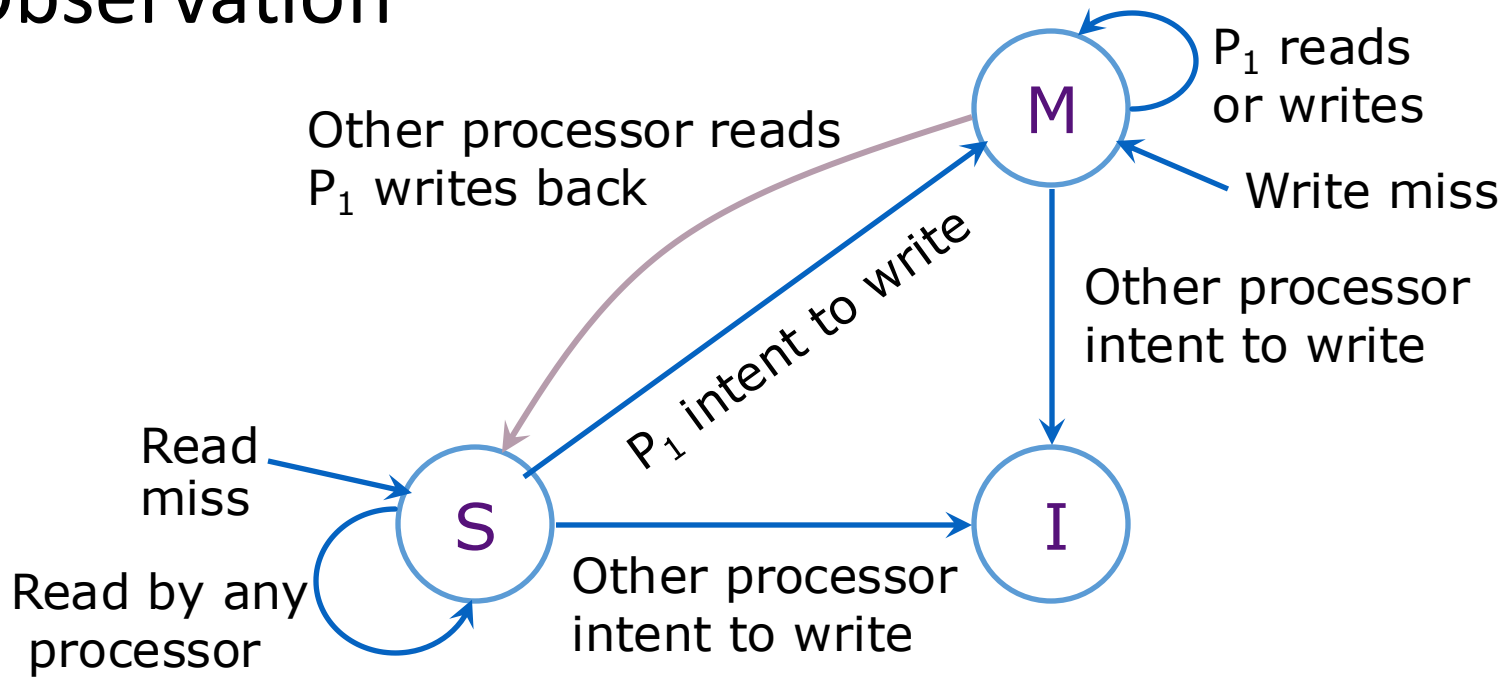


Two-Processor Example (Reading and writing the same cache line)

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

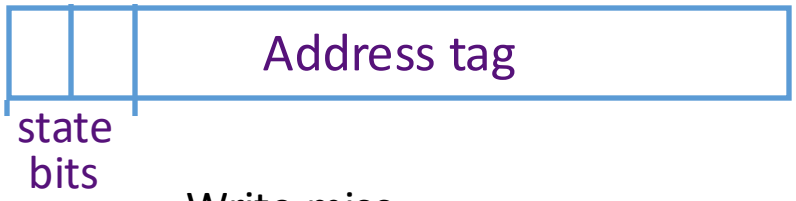


- If a line is in the M state then no other cache can have a valid copy of the line!
- Memory stays coherent, multiple differing copies cannot exist

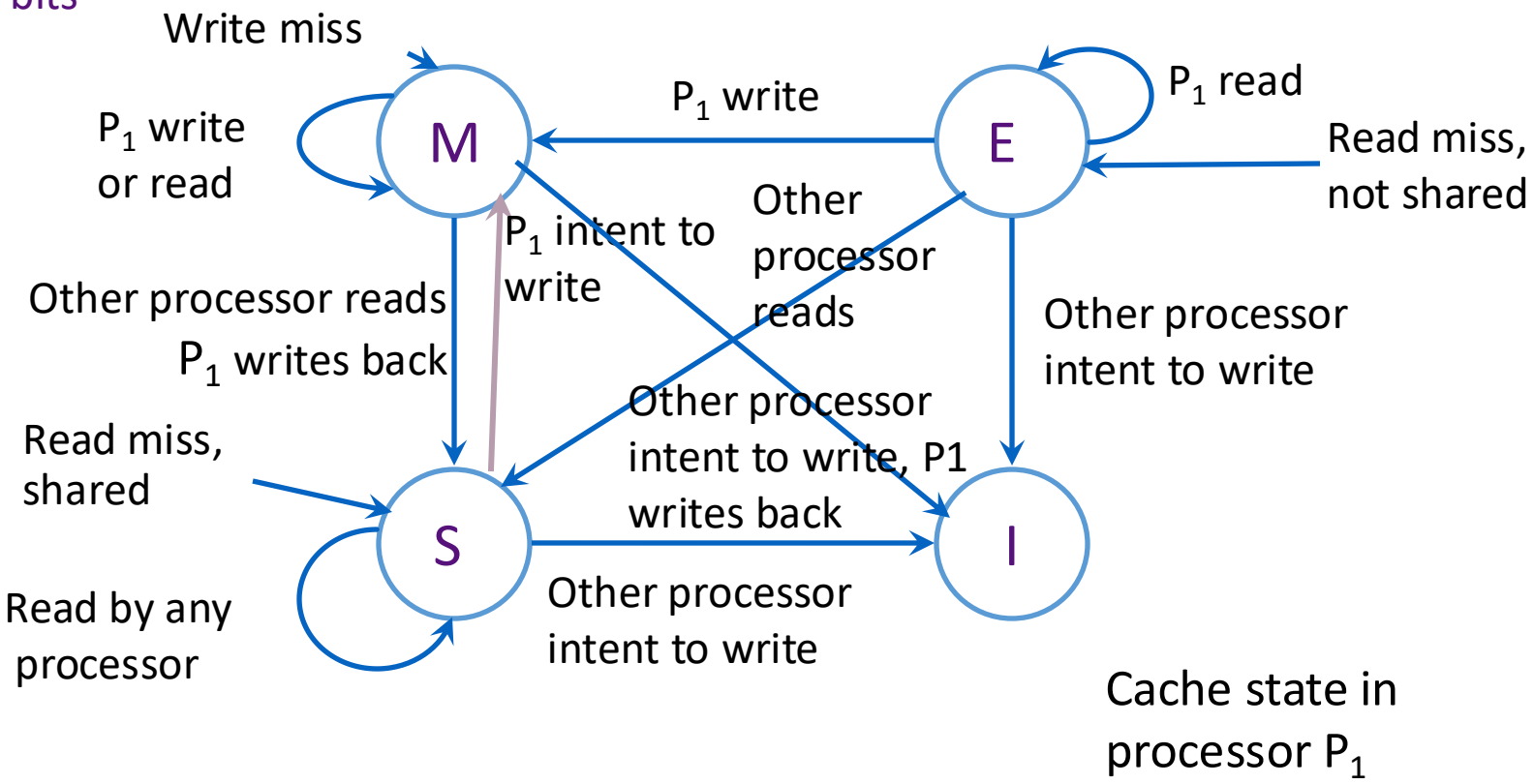
MESI: An Enhanced MSI protocol

increased performance for private data

Each cache line has a tag



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



MESI vs. MSI

- Advantage?
  - If a block is in the E state, it can be written without generating any invalidates, which optimizes the case where a block is read by a single cache before being written by that same cache.
    - A subsequent write to a block in the exclusive state by the same core need not acquire bus access or generate an invalidate, since the block is known to be exclusively in this local cache; the processor merely changes the state to modified.
- Disadvantage?
  - Complexity

The Problem with MESI

- Observation: Shared state requires the data to be clean
  - i.e., all caches that have the block have the up-to-date copy and so does the memory
- Problem: Need to write the block to memory when BusRd happens when the block is in Modified state

- Why is this a problem?
  - Memory can be updated unnecessarily → some other processor may want to write to the block again

Improving on MESI

- Idea 1: Do not transition from M→S on a BusRd. Invalidate the copy and supply the modified block to the requesting processor directly without updating memory

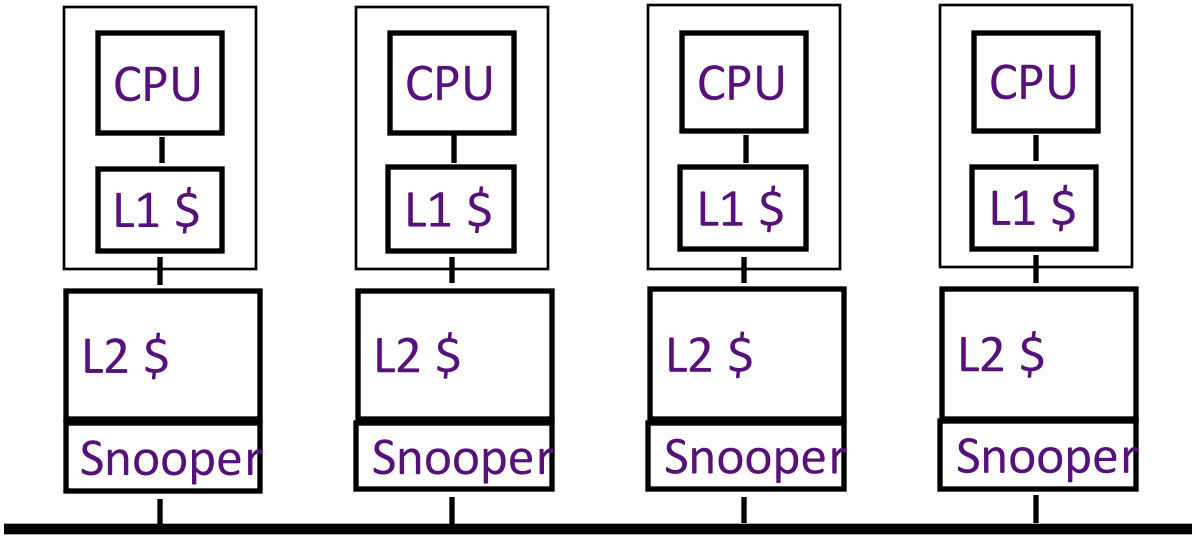
- Idea 2: Transition from M→S, but designate one cache as the owner (O), who will write the block back when it is evicted
  - Now "Shared" means "Shared and potentially dirty"
  - This is a version of the MOESI protocol

Tradeoffs in Sophisticated Cache Coherence Protocols

- The protocol can be optimized with more states and prediction mechanisms to
  - + Reduce unnecessary invalidates and transfers of blocks

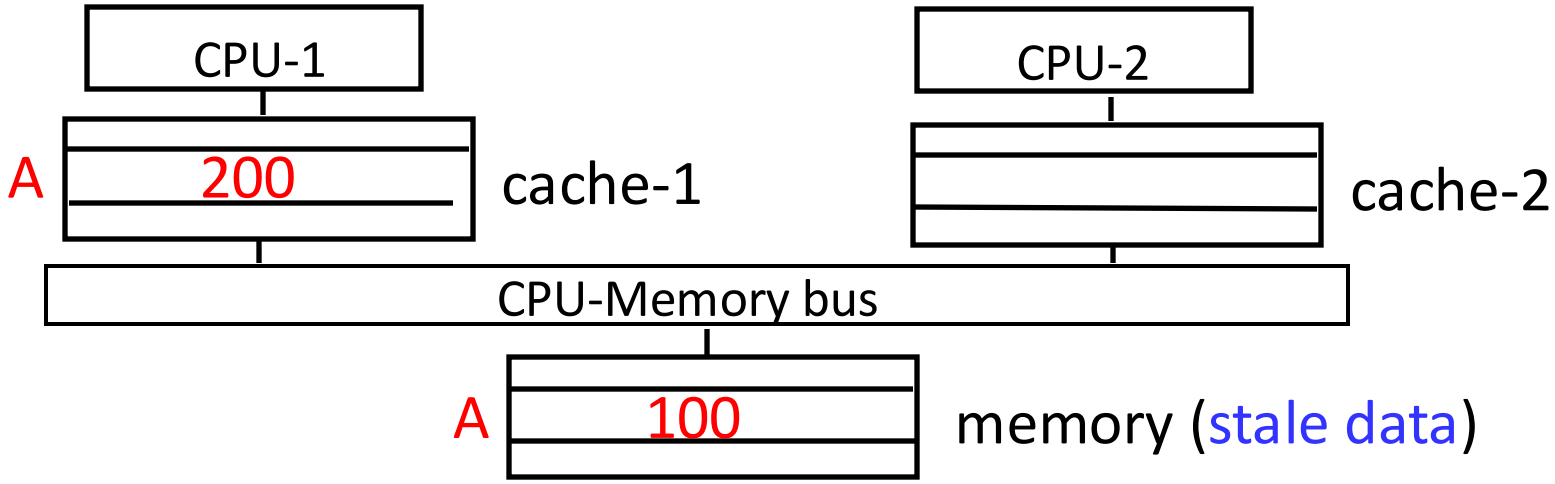
- However, more states and optimizations
  - -- Are more difficult to design and verify (lead to more cases to take care of, race conditions)
  - -- Provide diminishing returns

Optimized Snoop with Level-2 Caches



- Processors with two-level caches
  - small L1, large L2 (usually both on chip now)
- Inclusion property: entries in L1 must be in L2
  - Miss in L2 ⇒ Not present in L1
  - Only if invalidation hits in L2 ⇒ probe and invalidate in L1
- Snooping on L2 does not much 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 $i \neq k$ but both words have the same line address.

What can happen?

Performance of Symmetric Multiprocessors (SMPs)

Cache performance is combination of:

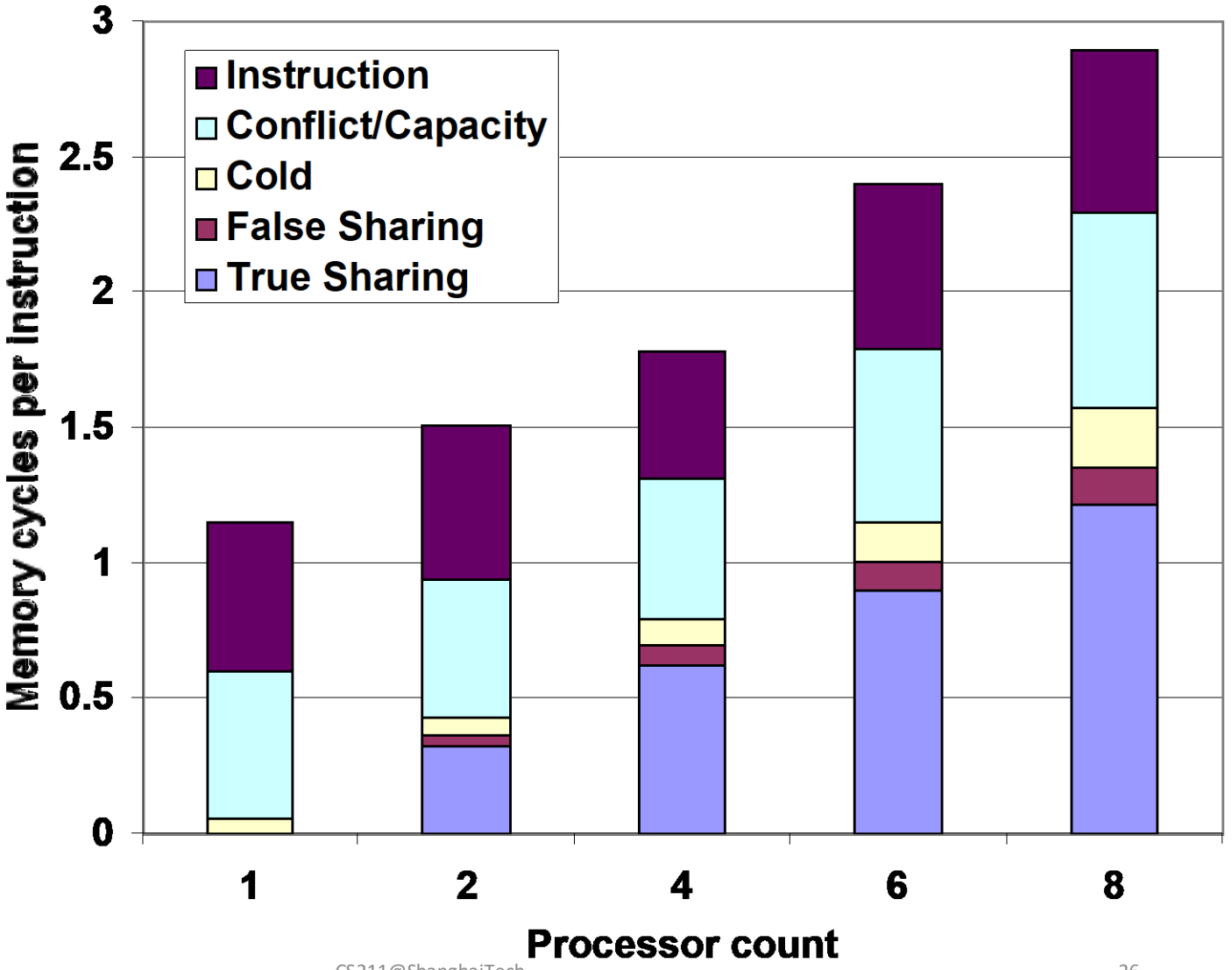
- Uniprocessor cache miss traffic
- Traffic caused by communication
  - Results in invalidations and subsequent cache misses
- Coherence misses
  - Sometimes called a Communication miss
  - 4th C of cache misses along with Compulsory, Capacity, & Conflict.

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

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

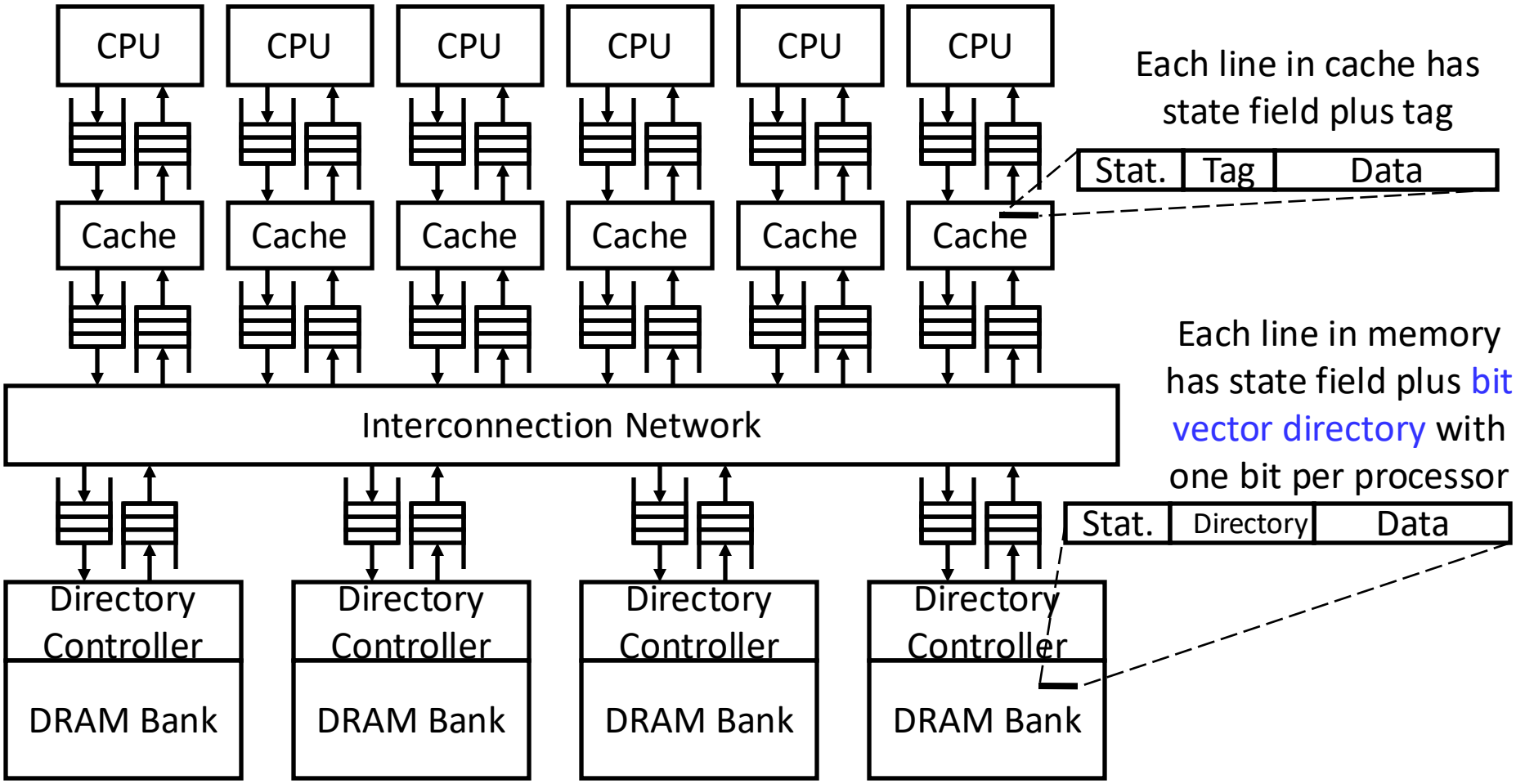
- When any processor gets a miss, must probe every other cache
- Scaling up to more processors limited by:
  - Communication bandwidth over bus
  - Snoop bandwidth into tags
- Can improve bandwidth by using multiple interleaved buses with interleaved tag banks
  - E.g, two bits of address pick which 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!

e.g., Intel QuickPath Interconnect

Scalable Approach: Directories

- 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

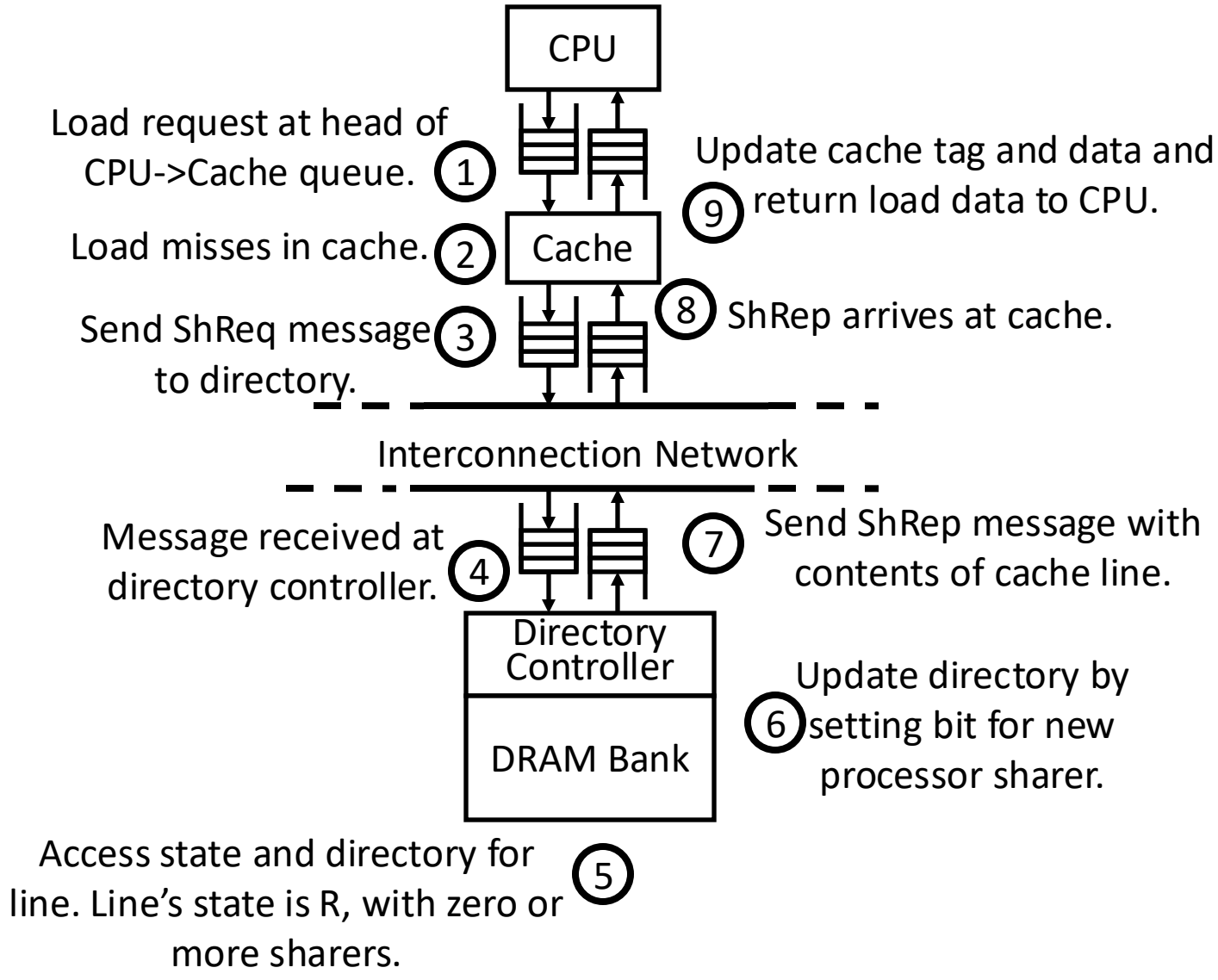
- For each cache line, there are 4 possible states:
  - **C-invalid** (= Nothing): The accessed data is not resident in the cache.
  - **C-shared** (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.
  - **C-modified** (= Ex): The 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

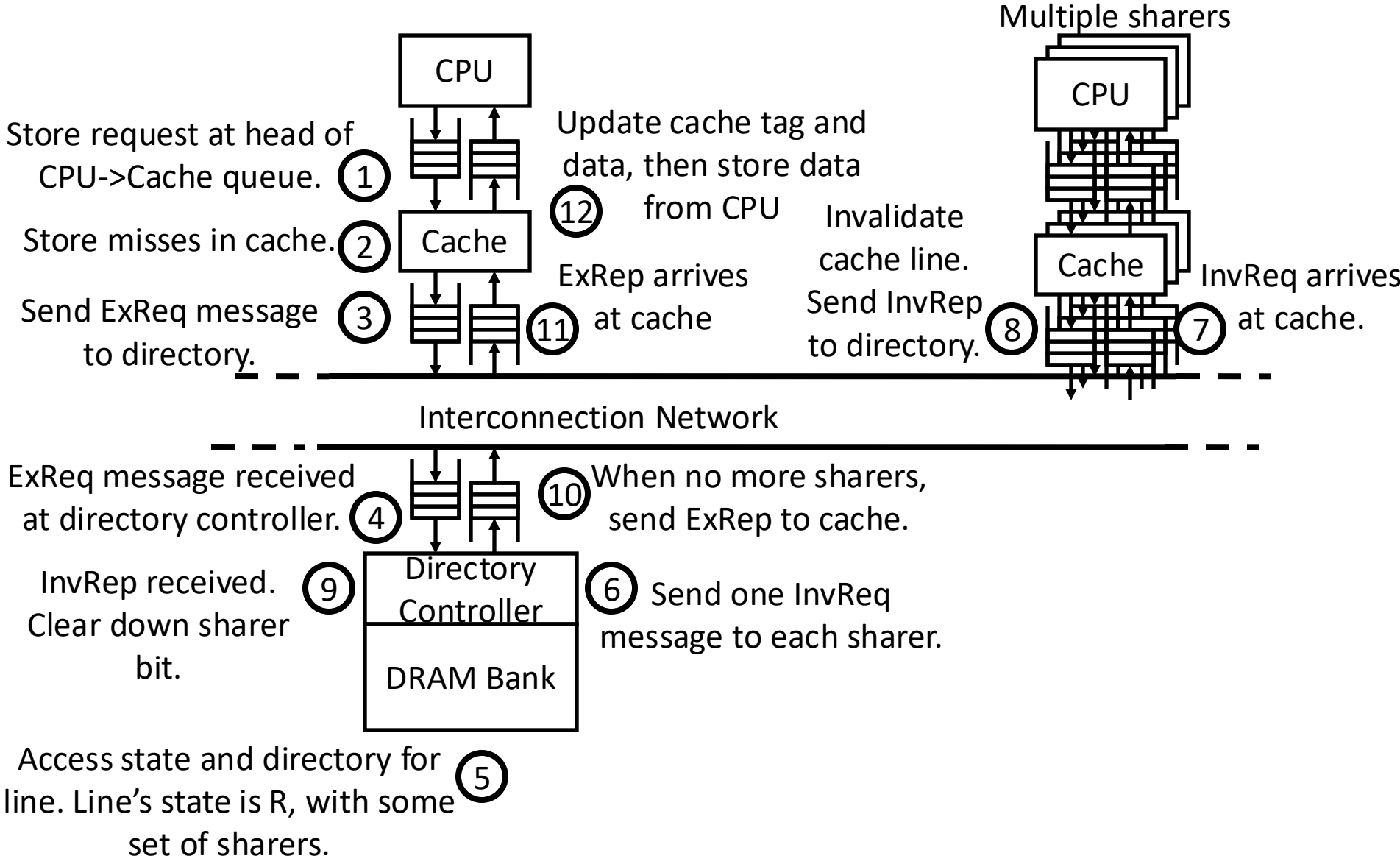
- For each memory line, there are 4 possible states:
  - **R(dir):** The memory line is shared by the sites specified in dir (dir is a set of sites). The data in memory is valid in this state. If dir is empty (i.e., dir = ∅), the memory line is not cached by any site.
  - **W(id):** The memory line is 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.

The home site is the node where the memory location and the directory entry of an address reside

Read miss, to uncached or shared line



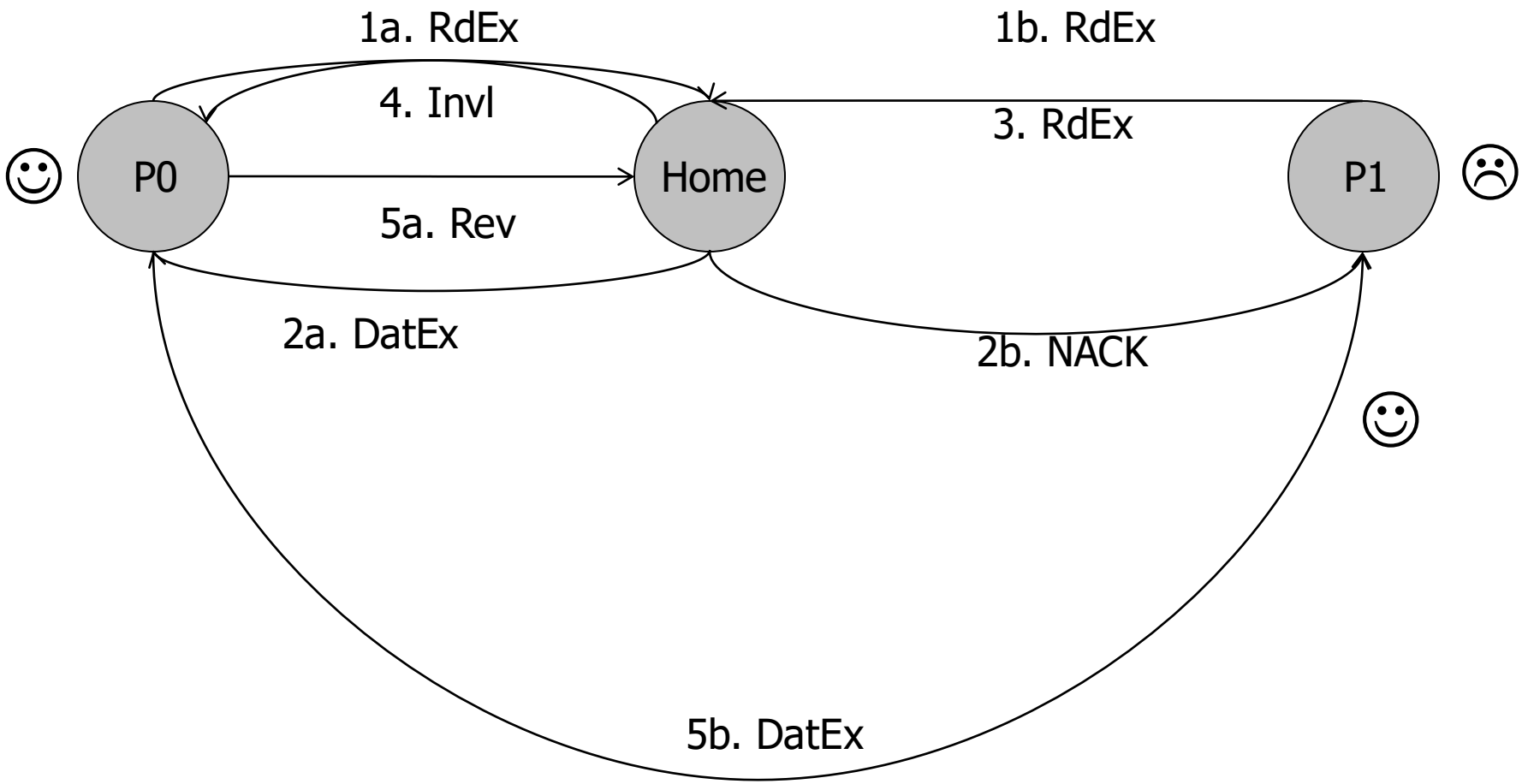
Write miss, to read shared line



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!

Contention Resolution (for Write)



Issues with Contention Resolution

- Need to escape race conditions by:
  - NACKing requests to busy (pending invalidate) entries
    - Original requestor retries
  - OR, queuing requests and granting in sequence
  - (Or some combination thereof)

- Fairness
  - Which requestor should be preferred in a conflict?
  - Interconnect delivery order, and distance, both matter

- Ping-ponging can be reduced w/ protocol optimizations OR better higher-level synchronization
  - With solutions like combining trees (for locks/barriers) and better shared-data-structure design

Scaling the Directory: Some Questions

- How large is the directory?
- How can we reduce the access latency to the directory?
- How can we scale the system to thousands of nodes?
- Can we get the best of snooping and directory protocols?
  - Heterogeneity
  - E.g., token coherence [Martin+, ISCA 2003]

Conclusion

- Snooping-based
- Directory-based

Acknowledgements

- These slides contain materials developed and copyright by:
  - Prof. Krste Asanovic (UC Berkeley)
  - Prof. Mengjia Yan (MIT)
  - Prof. Onur Mutlu (ETHZ)