CS302

Hardware Cache Coherence

Googled “Incoherent”

Spring 2025

Arkaprava Basu & Babak Falsafi

parsa.epfl.ch/course-info/cs302

Adapted from slides originally developed by Profs. Falsafi, Patterson, Wenisch, Fatahalian of CMU/EPFL, Michigan, UC Berkeley, and CMU

Where are We?

M	T	W	T	F
17-Feb	18-Feb	19-Feb	20-Feb	21-Feb
24-Feb	25-Feb		27-Feb	28-Feb
3-Mar	4-Mar	5-Mar	6-Mar	7-Mar
10-Mar	11-Mar	12-Mar	13-Mar	14-Mar
17-Mar	18-Mar	19-Mar	20-Mar	21-Mar
24-Mar	25-Mar	26-Mar	27-Mar	28-Mar
31-Mar	1-Apr	2-Apr	3-Apr	4-Apr
7-Apr	8-Apr	9-Apr	10-Apr	11-Apr
14-Apr	15-Apr	16-Apr	17-Apr	18-Apr
21-Apr	22-Apr	23-Apr	24-Apr	25-Apr
28-Apr	29-Apr	30-Apr	1-May	2-May
5-May	6-May	7-May	8-May	9-May
12-May	13-May	14-May	15-May	16-May
19-May	20-May	21-May	22-May	23-May
26-May	27-May	28-May	29-May	30-May

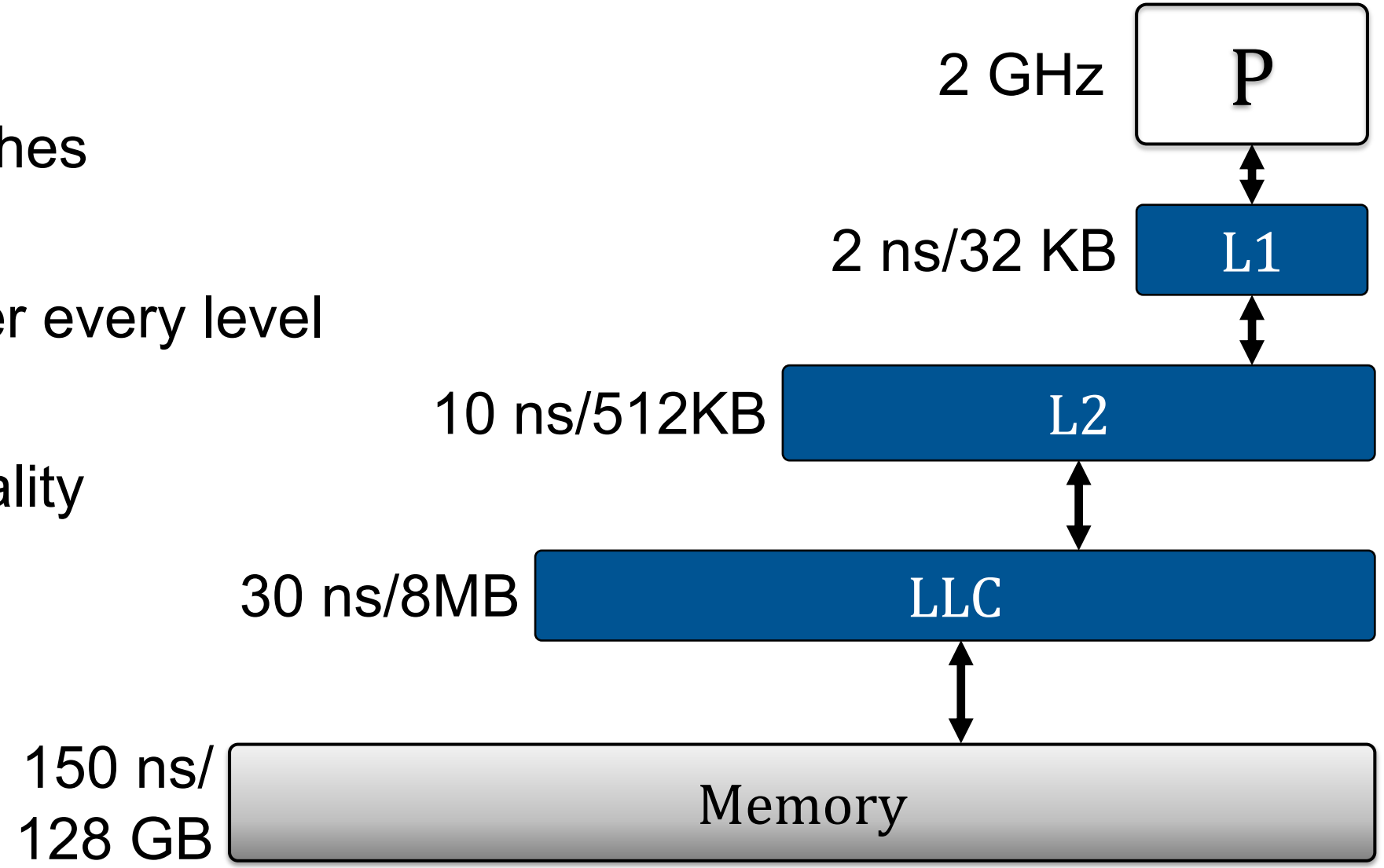
- Cache Coherence
  - Coherence protocols
  - Scaling to many cores
    - Directories
- Exercise session
  - Example OpenMP programs
- Next Tuesday
  - Optimizing SW for caches and coherence

Cache Coherence and Assignment 1

- Today's lecture explains how multiprocessor caches share data
  - Review of basics + advanced topics
- Next week we will see
  - How such sharing can reduce performance
  - How we can optimize sharing in software
- Part 2 of Assignment 1 requires such software optimizations for increasing the performance of parallel programs

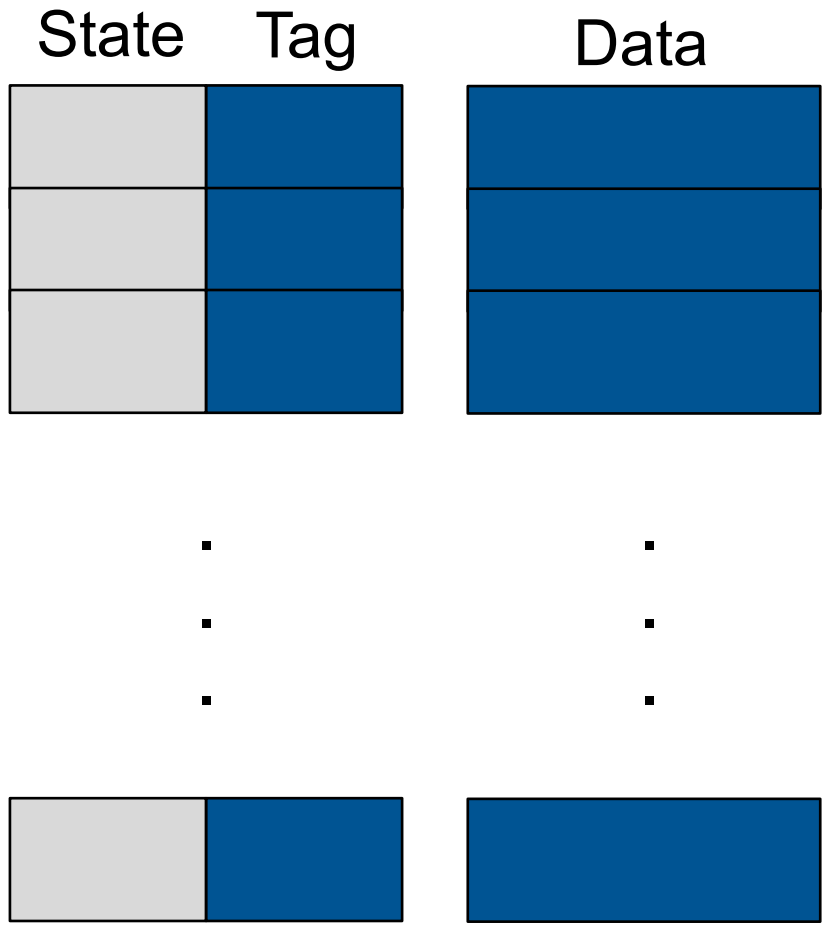
Review: Uniprocessor Memory Hierarchy

- Hierarchy of caches
- Slower but bigger every level
- Principles of locality



Review: Inside a cache

- Data array keeps cache blocks
- Tag array keeps
  - Tags ~ addresses
  - State of block
- State
  - Valid/Invalid
  - Dirty for writeback caches



Assumptions

- All memory accesses are atomic & in program order
- All cores see all accesses immediately (for now)
- We will just look at coherence in L1 Data Caches
  - In reality you have coherence in L1 Data, Instructions, L2, L3, L4,..
  - All coherence is maintained at the granularity of a cache block

Review: What is Cache Coherence?

Incoherence is intrinsically due to sharing data

- More specifically, updating a value and reading a different copy

Shared Memory Expectation:

- Any core P, reading address X, will get the last value written to X
  - Fairly intuitive for sequential programs
  - Difficult for parallel programs, may have intervening writes

Review: What is Cache Coherence?

- Caches don’t change program semantics
  - They just exist for performance optimization (fast data access)

- Goal of cache coherence is to be invisible
  - Given a program executing on cores with their own caches
  - Make it operate as if the caches are not present, providing the illusion of simple global memory

Mechanism for Basic Coherence

- Assume a bus connects cores and memory
  - Allows global visibility of all memory operations
- Add independent “cache controllers” to each core
  - Each controller is a Finite State Machine (FSM)
  - The controller sees all remote processor actions
  - Takes coherence actions without stopping the processor
  - Actions depend on block’s state, and protocol

The MESI Protocol

- **M**odified, **E**xclusive, **S**hared, **I**nvalid

- Modified
  - This cache has updated the value and holds the authoritative copy
  - Memory is stale, needs to be updated on eviction
- Exclusive
  - This cache has an exclusive copy
  - It can write to the block (and move to M) without telling anyone
- Shared
  - One or more caches hold the value, read permission only
- Invalid

The MESI Protocol

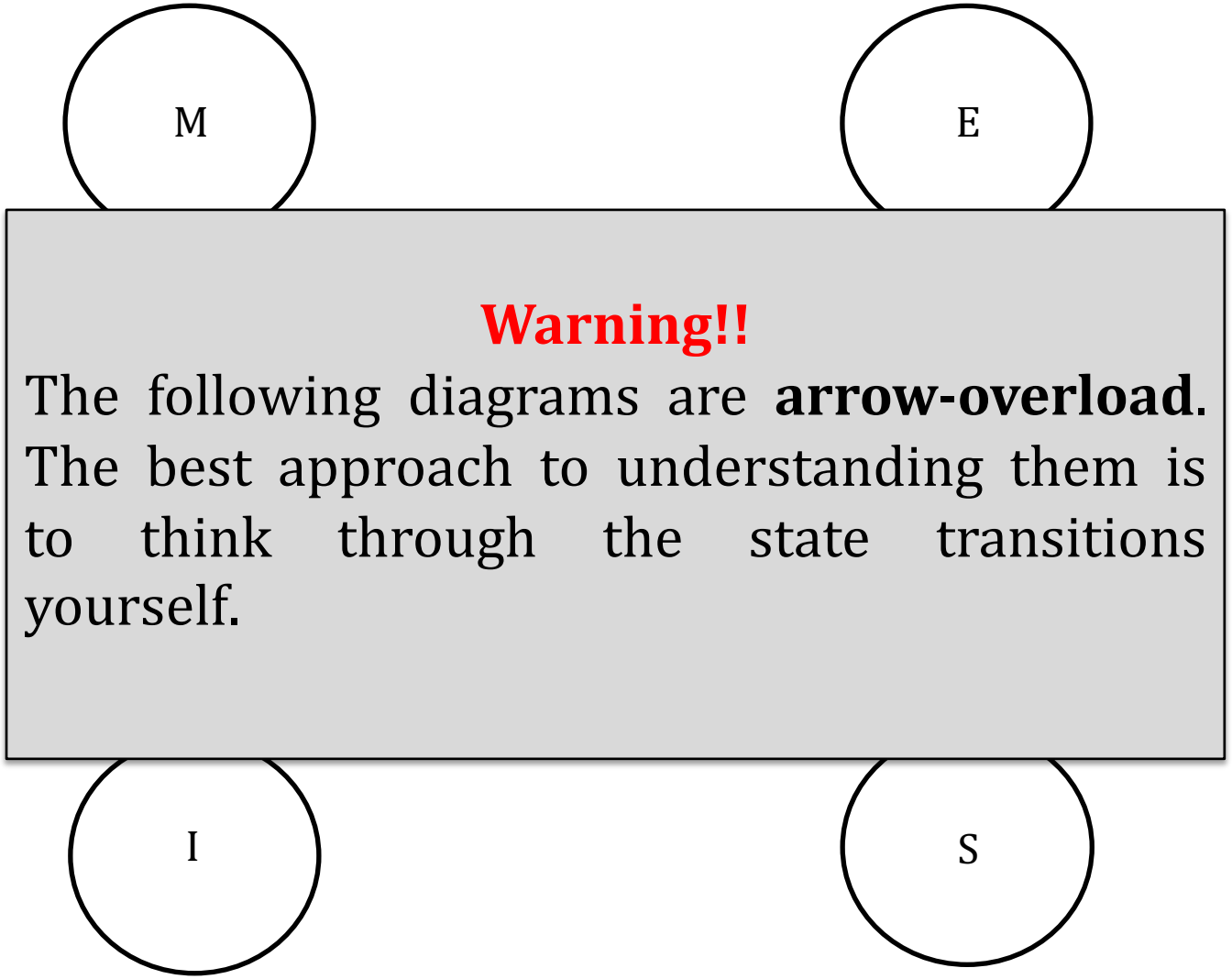
- Core Side
  - Must obey permissions implied by cache state
  - e.g., Cannot write in **S**

- Cache Controller Side
  - Distinguish between read and ”read with intent to modify”
    - Load → read → Shared
    - Store → “read to modify” → Modified
  - On a write to a block in S, send an **invalidate** to other caches
  - When seeing a read to a block in S, raise the “copies” line
  - When seeing a read to a block in M, put local data on the bus

MESI Protocol State Diagram

- Terminology for all coherence diagrams:
  - { Event → Response }
- Core Actions
  - PRd – Processor attempts to read a line
  - PWr – Processor attempts to write a line
- Remote Actions
  - BusRd – Obtain copy of a line with no intent to modify
  - BusRd**X** – Obtain copy of a line **with** intent to modify
  - BusInv – Get rid of other copies
  - DataWB – Put updated value of a line on the bus

Review: MESI Protocol State Diagram

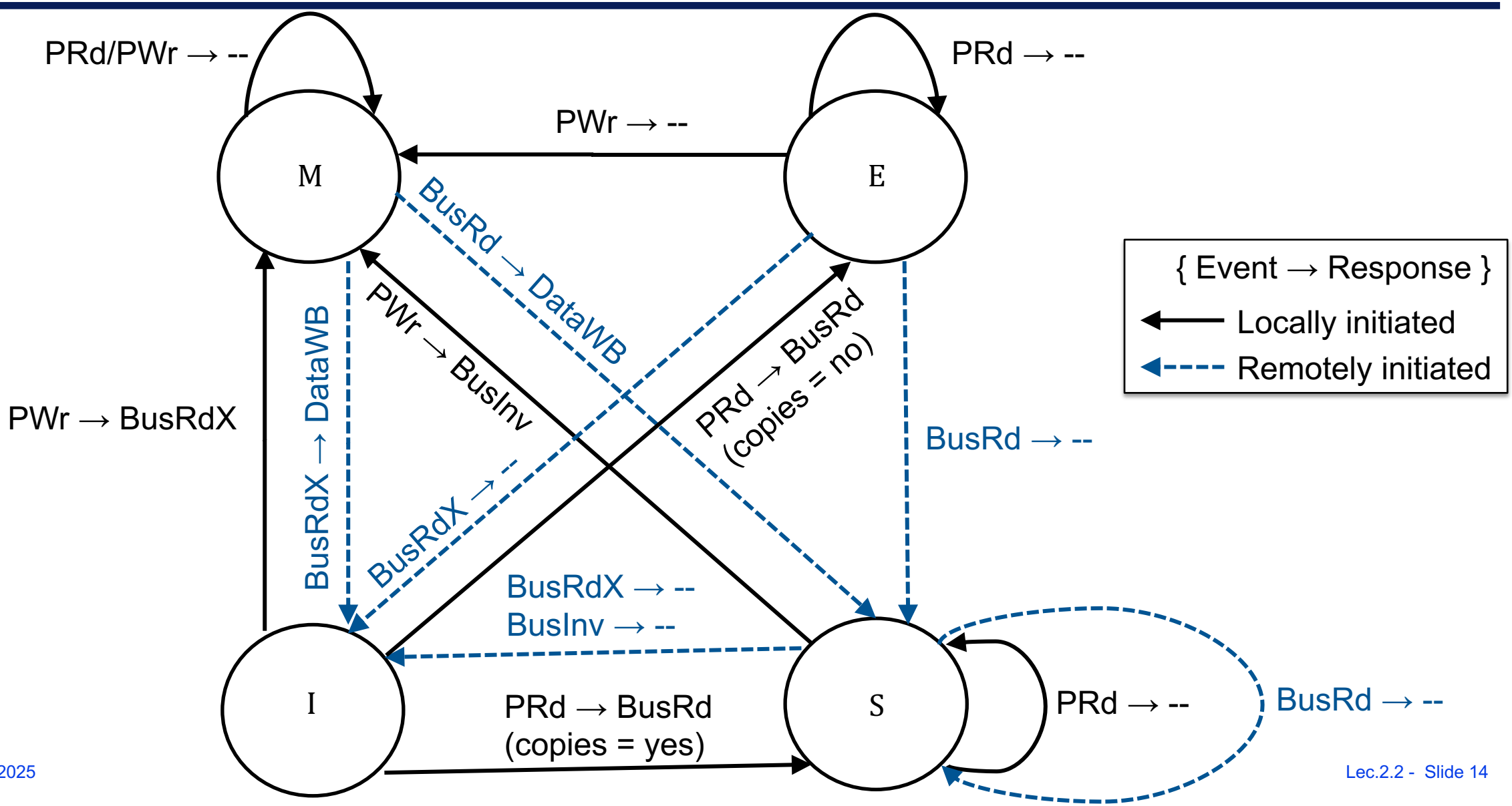


{ Event → Response }

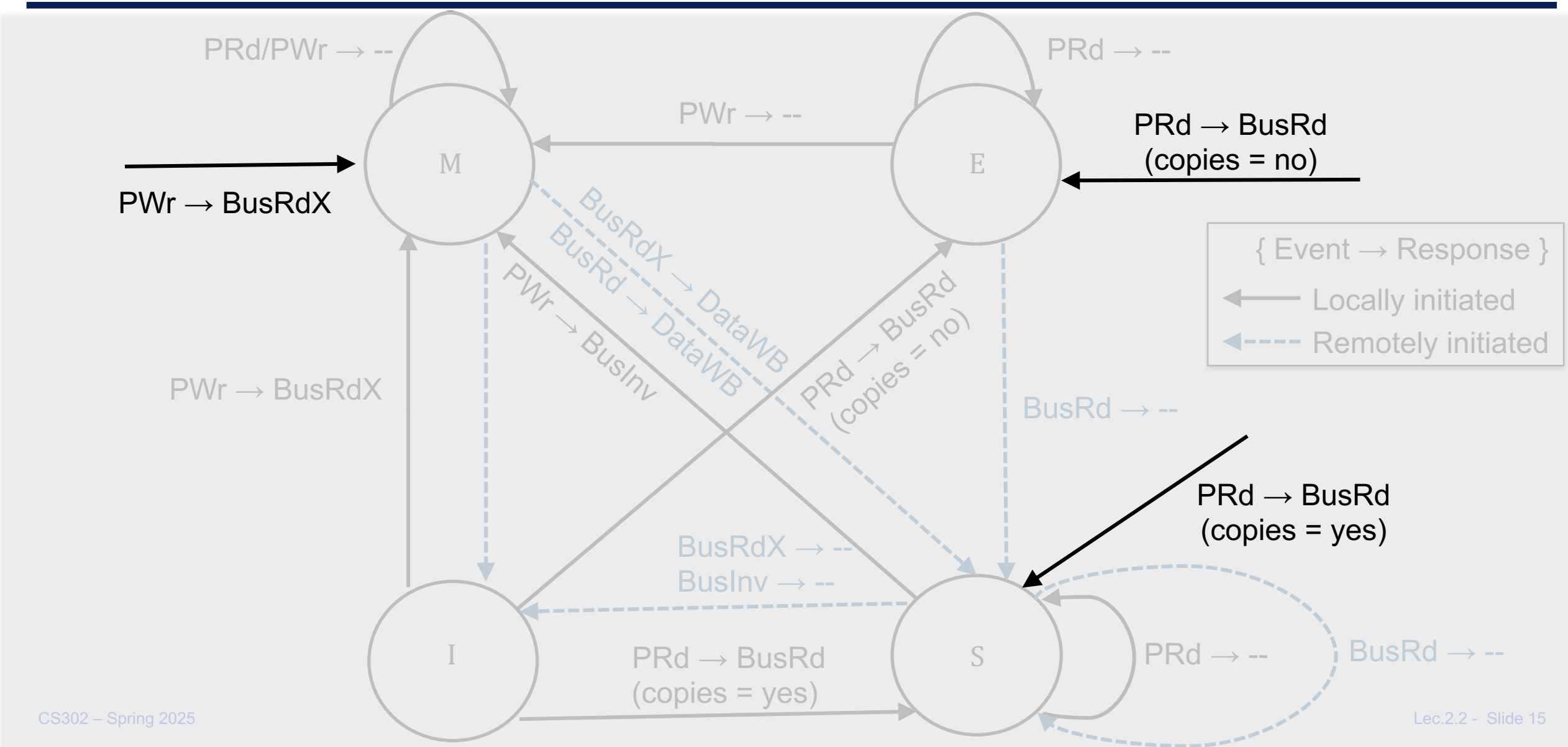
Locally initiated

Remotely initiated

Review: MESI Protocol State Diagram

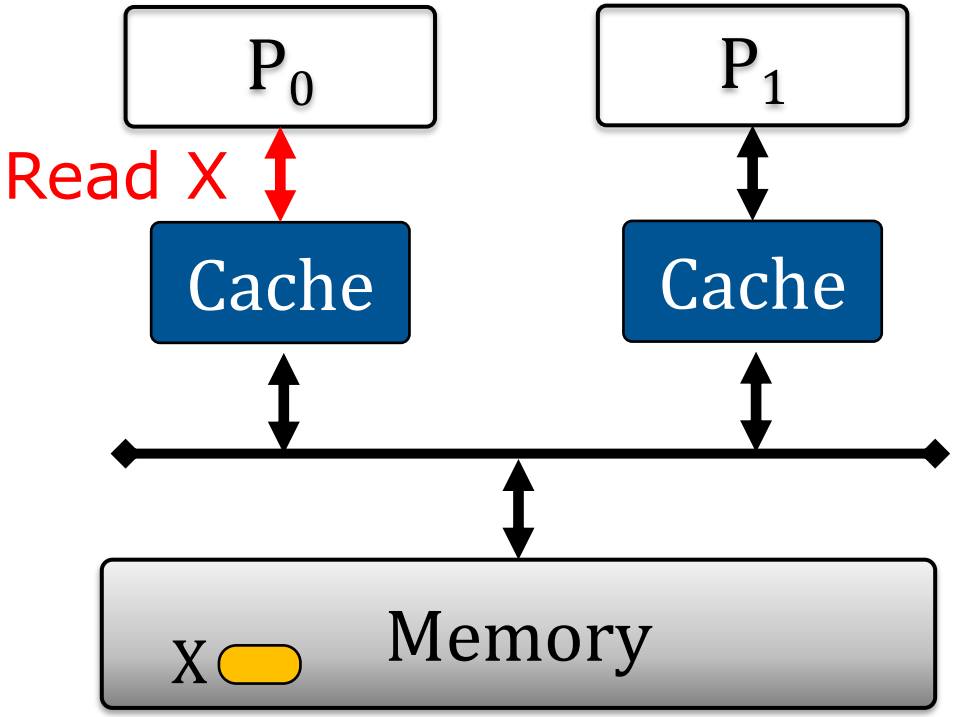


Review: Remaining Arcs (when the block is not in the cache)



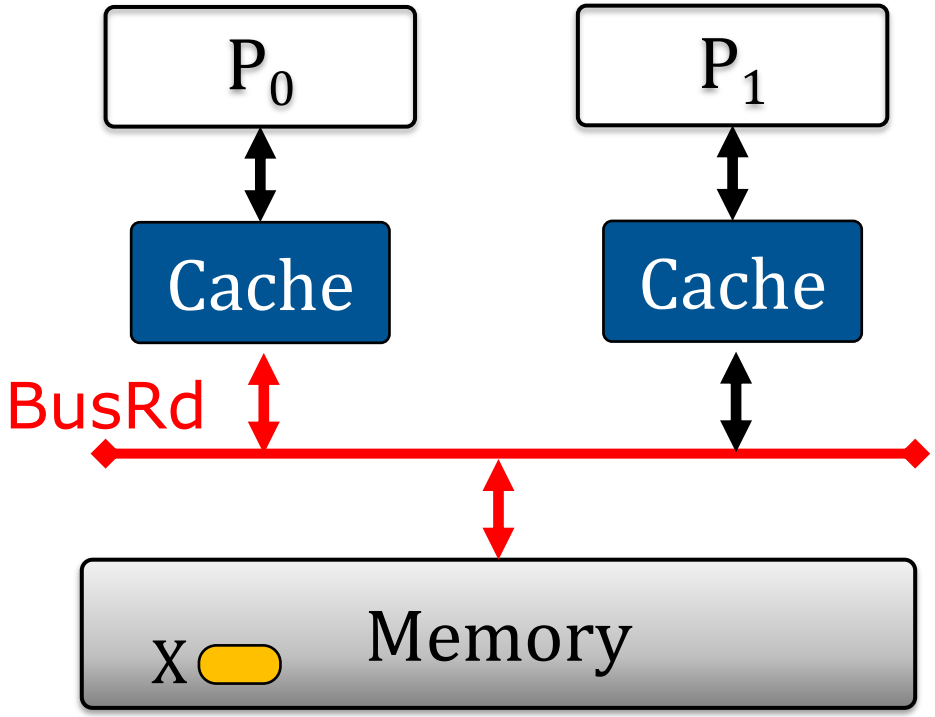
Example: MESI Protocol (Read)

1. P_0 Read X



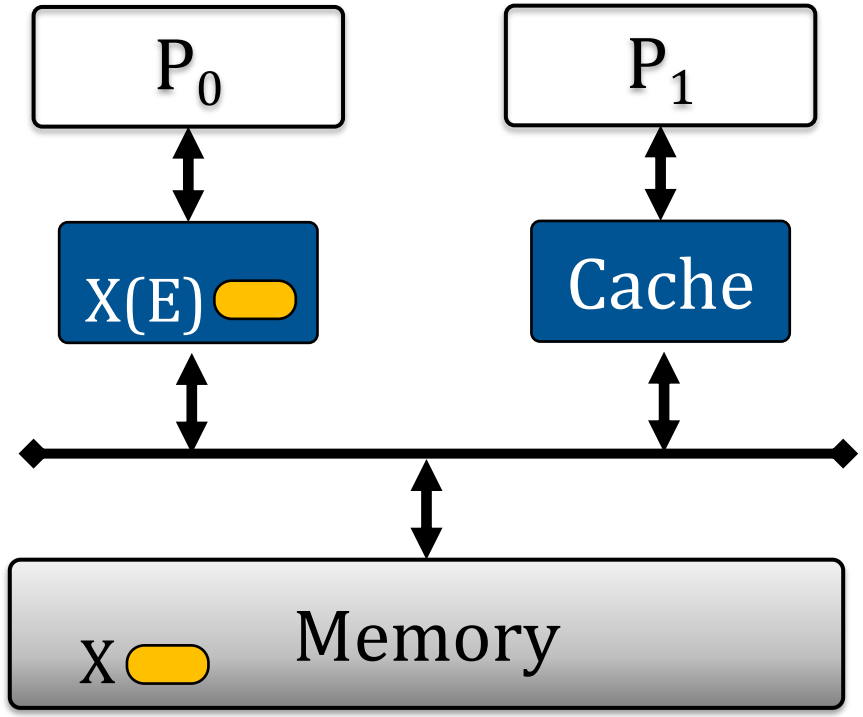
Example: MESI Protocol (Read)

1. P_0 Read X
   a) Cache controller issues a read on bus



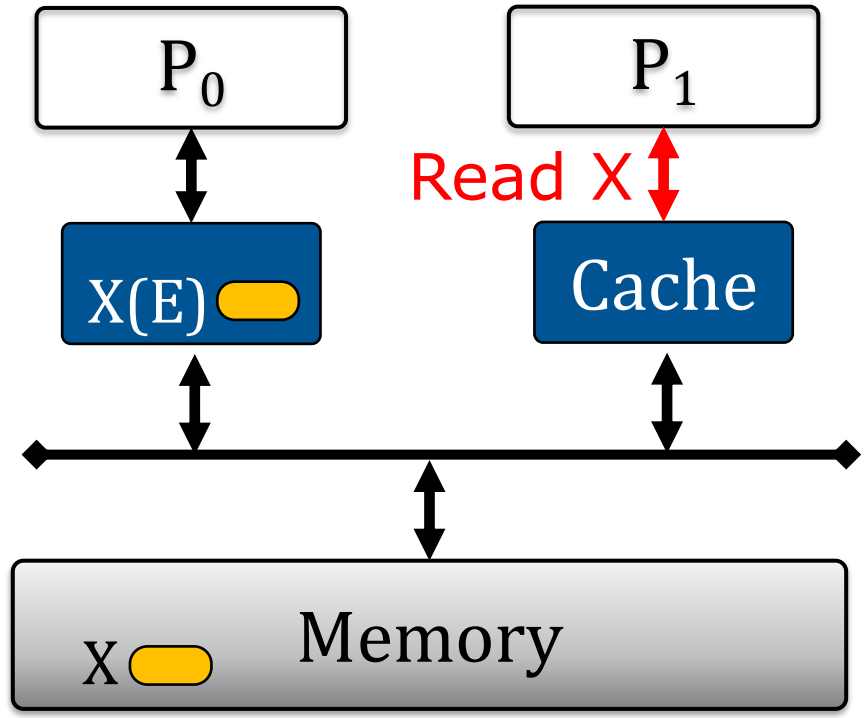
Example: MESI Protocol (Read)

1. P_0 Read X
   a) Cache controller issues a read on bus
   b) X returned (copies=no) → **E**



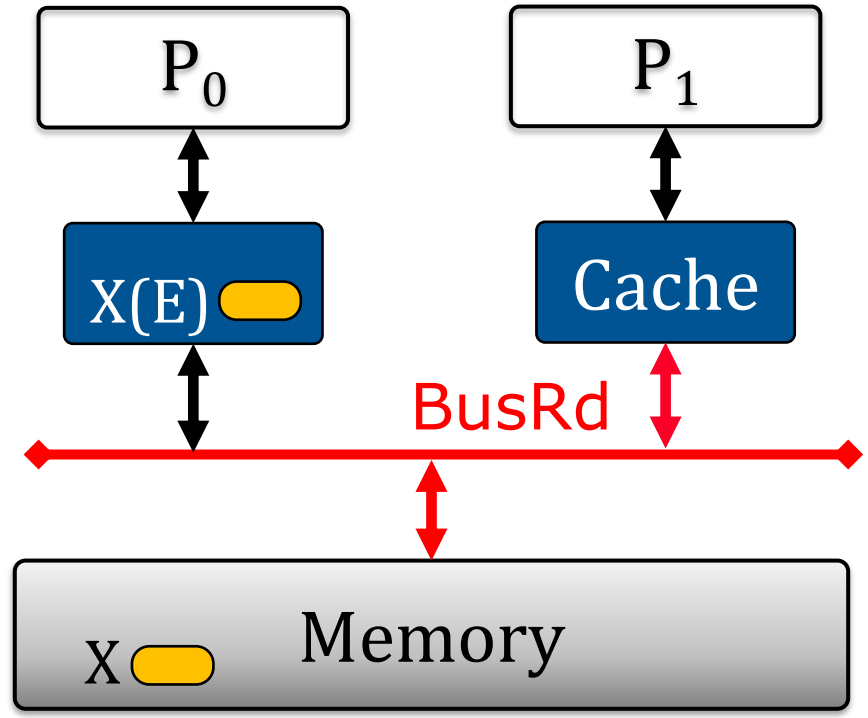
Example: MESI Protocol (Read)

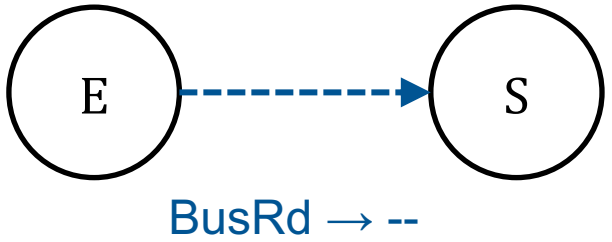
1. P_0 Read X → **E**
2. P_1 Read X



Example: MESI Protocol (Read)

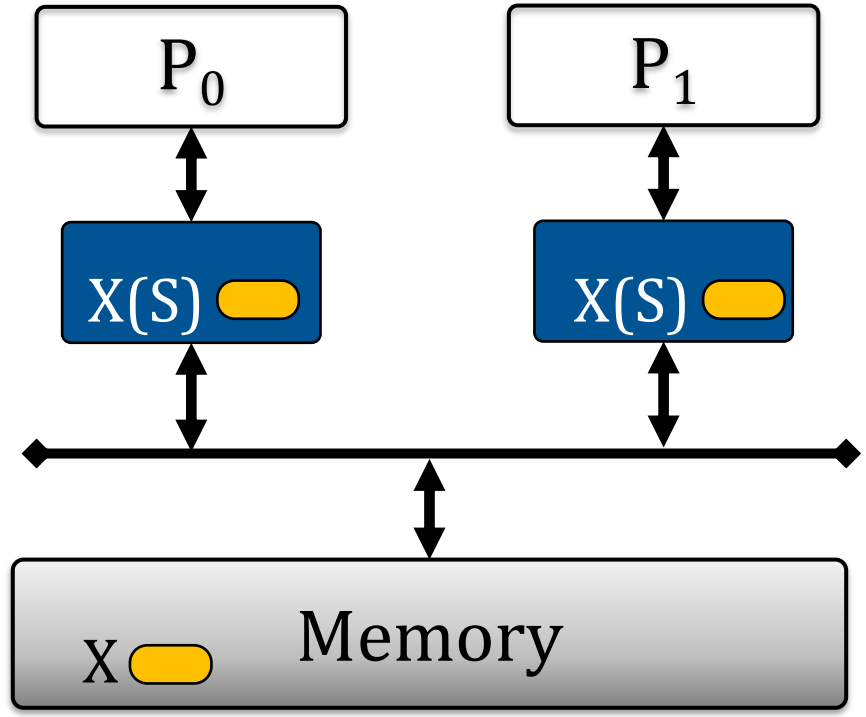
1. P_0 Read X → **E**
2. P_1 Read X
   a) Cache controller issues a read on bus





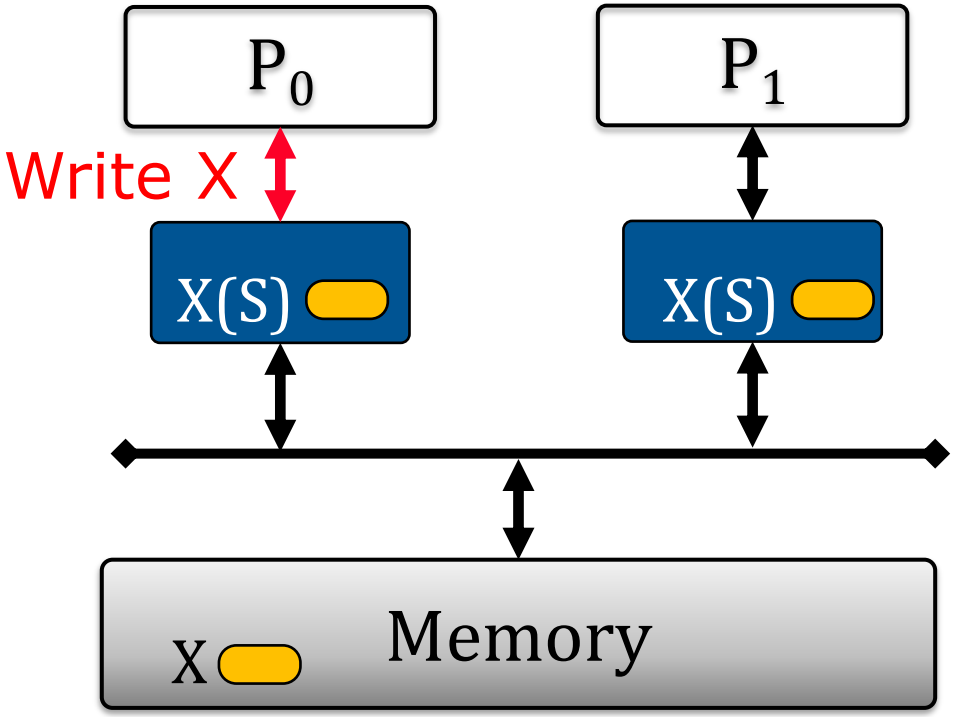
Example: MESI Protocol (Read)

1. P_0 Read X → **E**
2. P_1 Read X
   a) Cache controller issues a read on bus
   b) X returned (copies=yes) → **S**



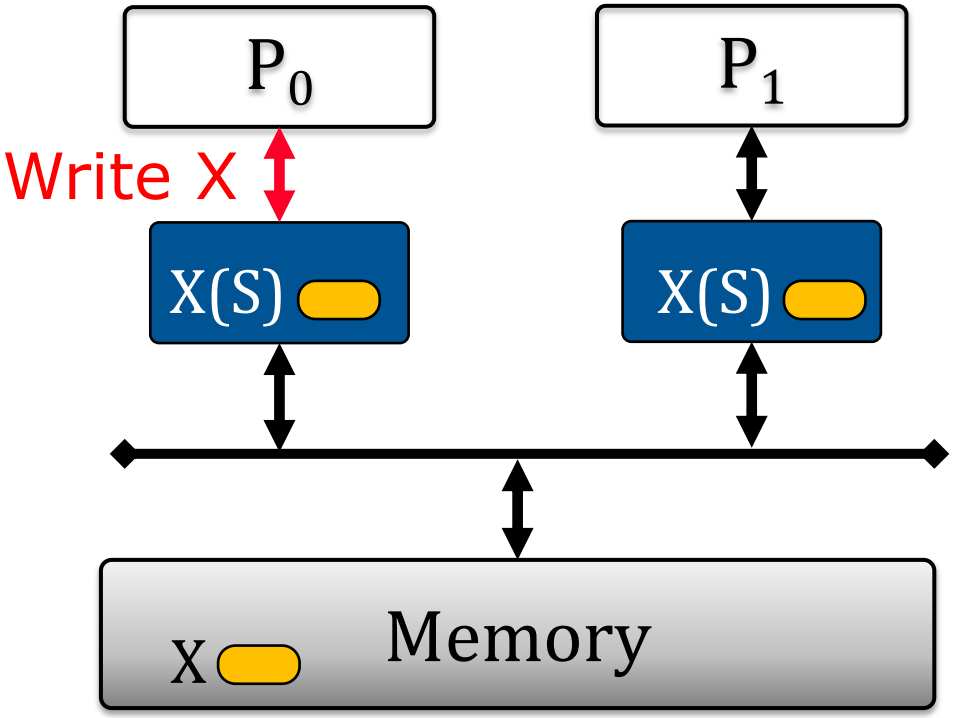
Example: MESI Protocol (Write)

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X



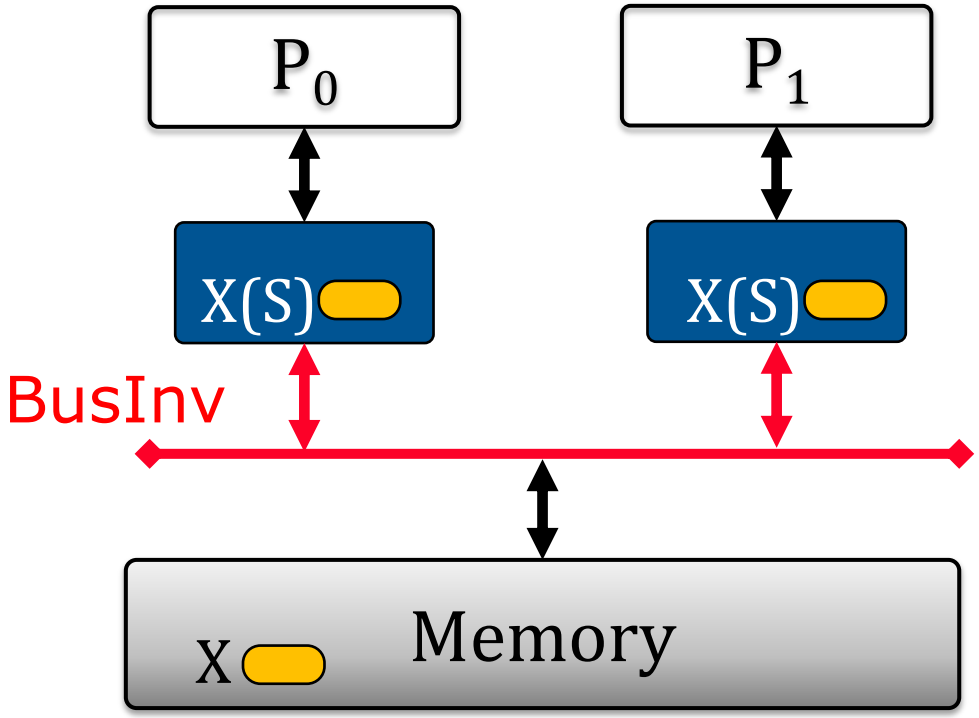
Example: MESI Protocol (Write)

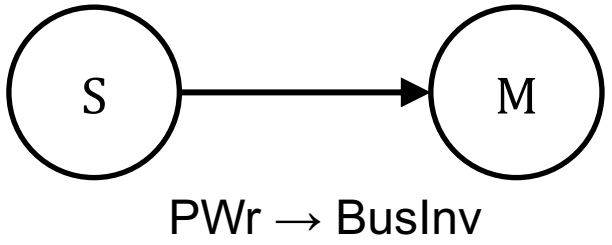
1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Cannot write, as we do not have permissions.



Example: MESI Protocol (Write)

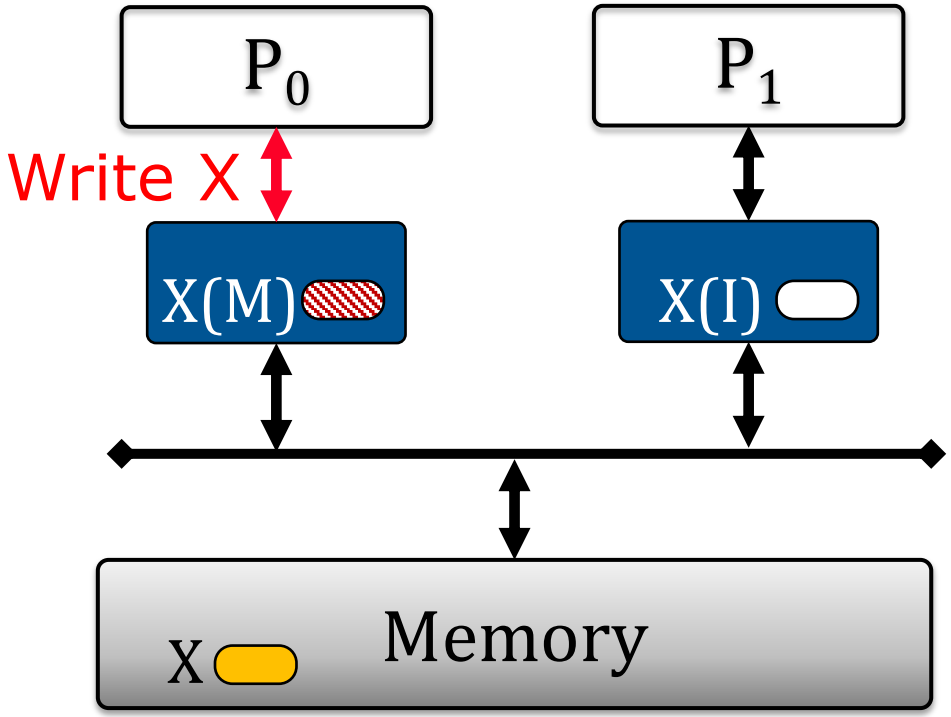
1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Cannot write, as we do not have permissions.
   b) Send BusInv





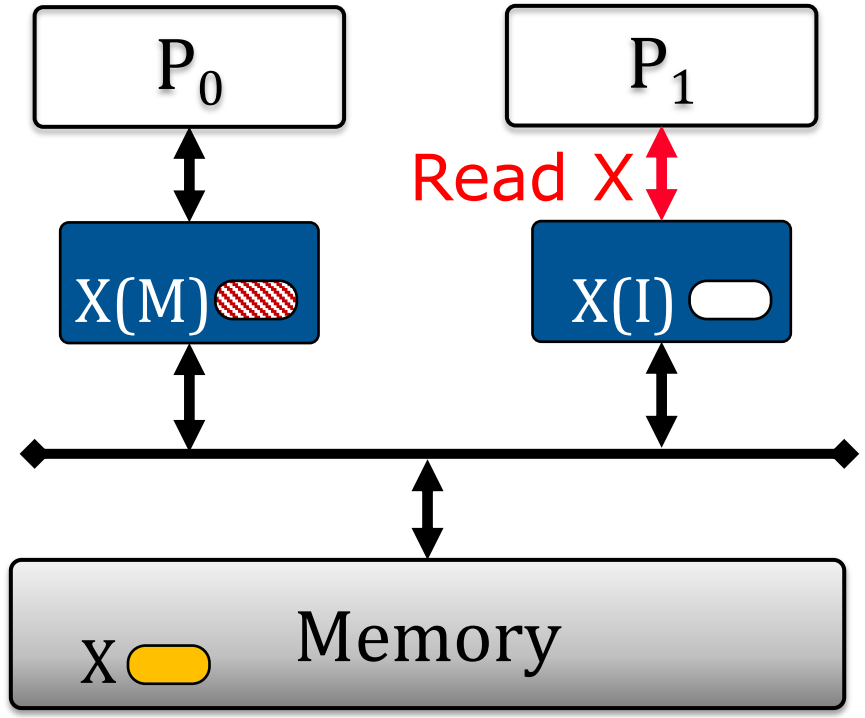
Example: MESI Protocol (Write)

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Cannot write, as we do not have permissions
   b) Send BusInv → **M**
   c) Other caches remove the block → **I**
   d) Allow write to proceed



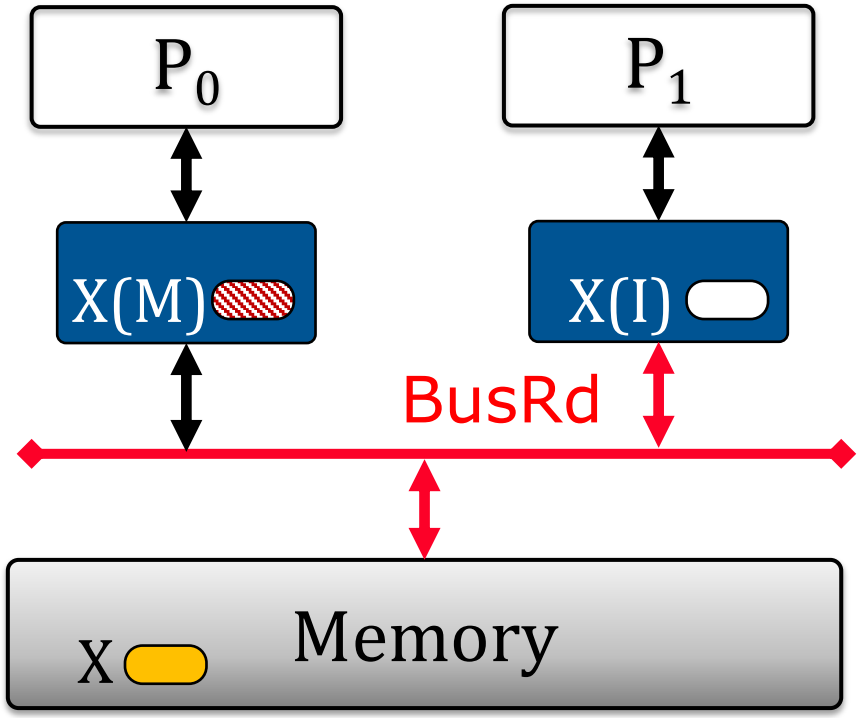
Example: MESI Protocol (Writeback)

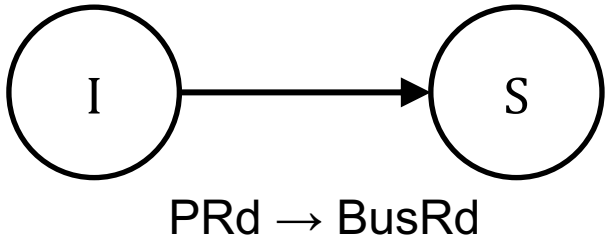
1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X



Example: MESI Protocol (Writeback)

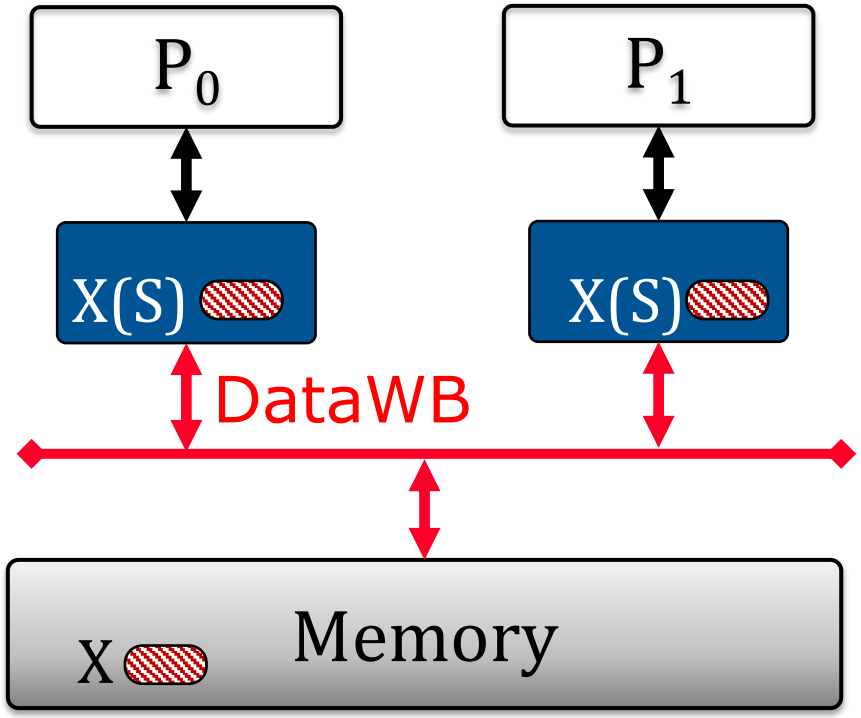
1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X
   a) Issue read on the bus





Example: MESI Protocol (Writeback)

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X
   a) Issue read on the bus
   b) P_0 sends new value of X → **S**
   c) Memory is updated
   d) P_1 caches the value → **S**



BusRd → DataWB

Summary

- Coherence
  - Keeps values at an address up-to-date across parallel processors
  - Do not confuse with consistency: order of loads/stores to multiple addresses
- MESI protocol
  - Optimizes interconnect traffic, uses write-back caches
  - More complex cache controller and protocol
- 3Cs for misses are now 4Cs (coherence)

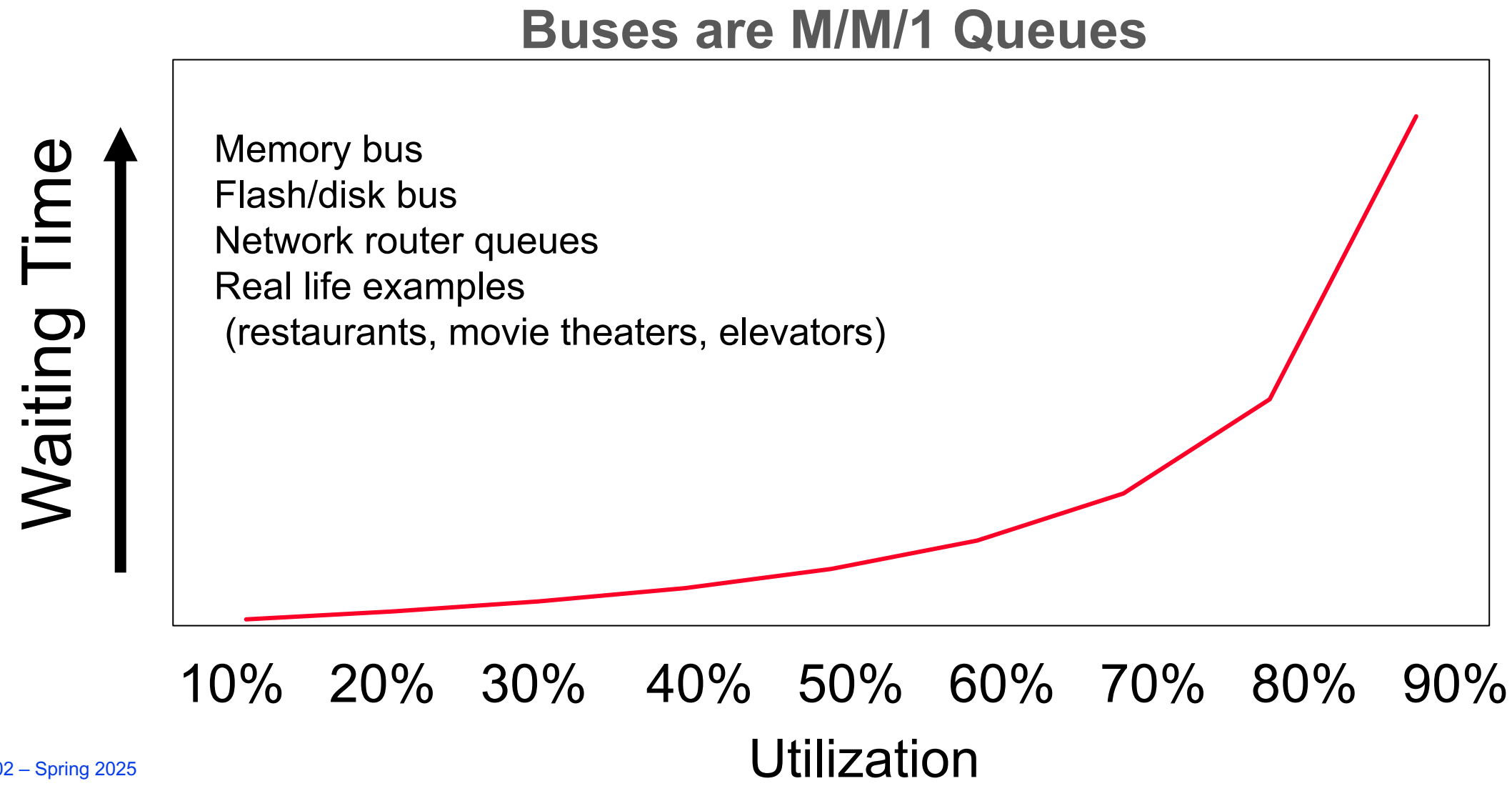
Limitations on Interconnect Scaling

- MESI protocol reduces bus contention by eliminating unnecessary BusInv messages
- But the bus is a globally shared substrate
  - Tech. limitations mean it can only support a handful of cores
  - e.g., Physical size, electrical capacitance, conflict resolution

Exercise: Buses & Cores

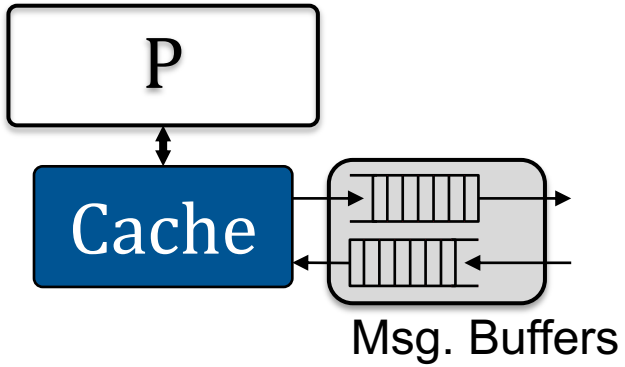
- Buses have long wires and high capacitance
- Bus cycle times could be 3x core cycle time
- Assume:
  - 4.8 GB/s traffic on the backside of L1 (L1 miss traffic)
  - 64-byte blocks
  - 2 GHz cores
- How many cores can the bus support, assuming 40% maximum utilization?
  - Answer: Bus clock is 0.5ns x 3 = 1.5ns
  - Bus bandwidth = 64 bytes/1.5 ns = 43 GB/s, at 40% = 17.06 GB/s
  - # of cores = 17.06/4.8 ~ 3 cores

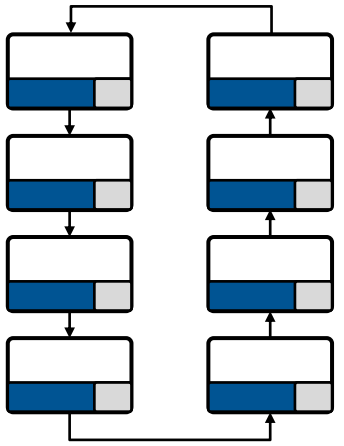
Centralized Structures w/ Utilization are a Bottleneck!



Point to Point Network Organization

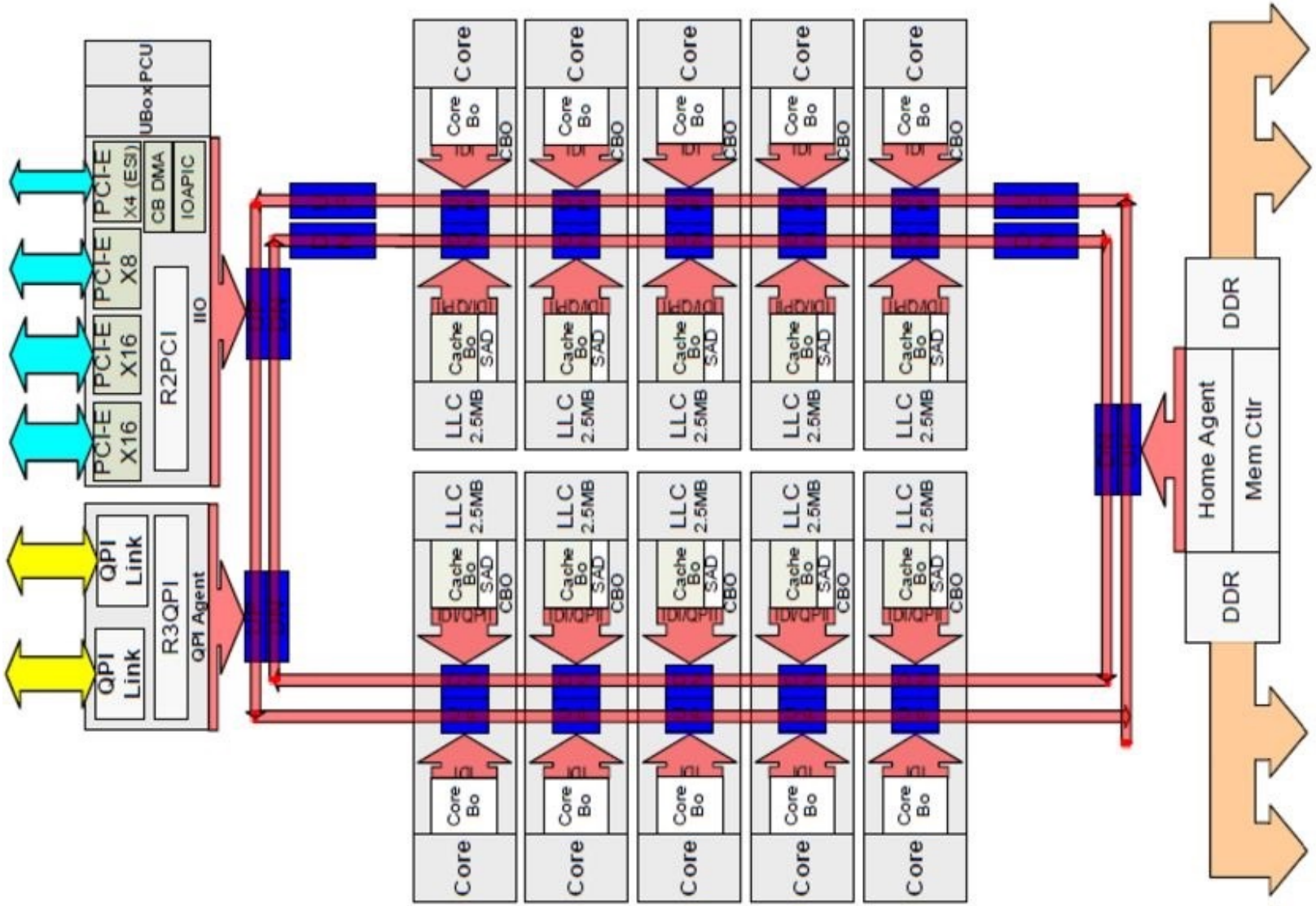
- More scalable systems use point-to-point network
  - Messages have an explicit source and destination
  - e.g., Intel Skylake uses a ring





- Problem for our coherence protocols
  - Each core no longer sees all of the messages
  - Need to introduce another protocol controller into the system…

Point to Point Network Organization



Directory-Based Protocol

- Previously, the bus served as a global “ordering point” for all coherence activity
- Instead, we use a **coherence directory**
  - The directory tracks all caches that contain a particular block
  - All messages leaving the core are first sent to the directory
  - The directory takes the appropriate actions, and then responds to the original cache’s request
  - The directory serves as a per-block address “ordering point”
- For now, directory is just another node in the interconnection network (other structures later)

Directory-Based Protocol Messages

Bus-based: Bus Actions

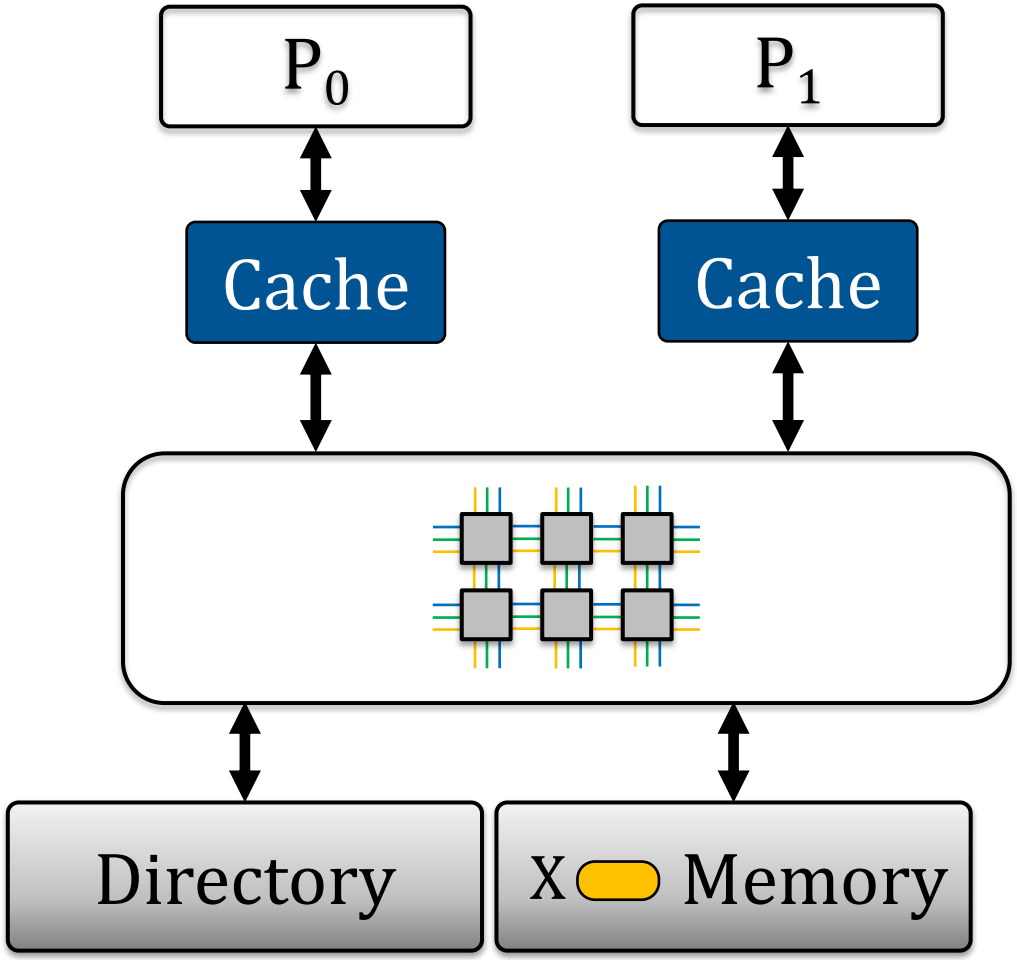
- BusRd
- BusRd**X**
- BusInv
- DataWB

Directory-based: Messages

- Cache to directory:
  - GetRd – Get copy to read
  - GetWr – Get copy to modify
  - UpGrd – Get rid of other copies
  - DataWB – Write back a line
  - Ack – Acknowledge message
- Directory to cache
  - DwnGrd – Go from E/M → S (if in M, write back a copy)
  - Inv – Get rid of your copy
  - Fill – Here is a copy of the line
  - Ack – Acknowledge message

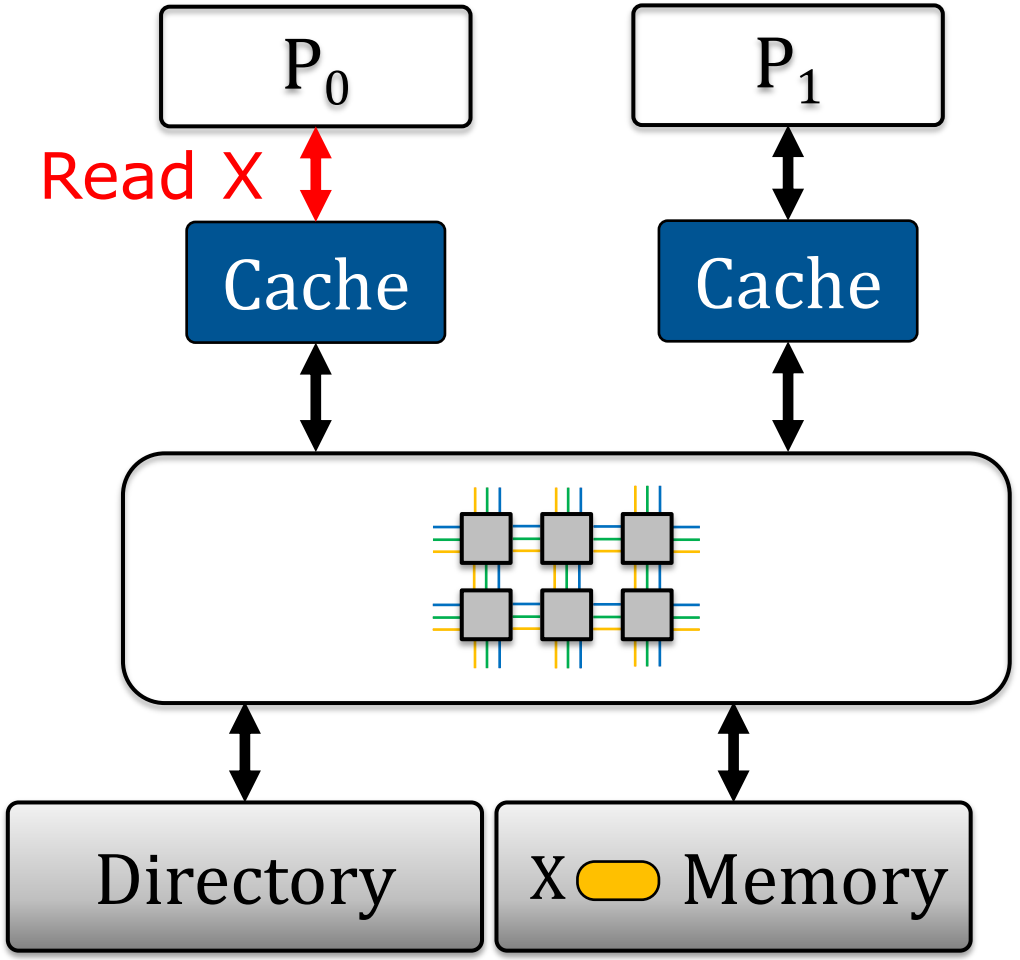
Example: Directory-Based MESI Protocol

- Bus replaced by a generic point-to-point network
- Directory sits near memory
  - We will just show the state for X…
  - But it stores entries for all cache blocks in the cores



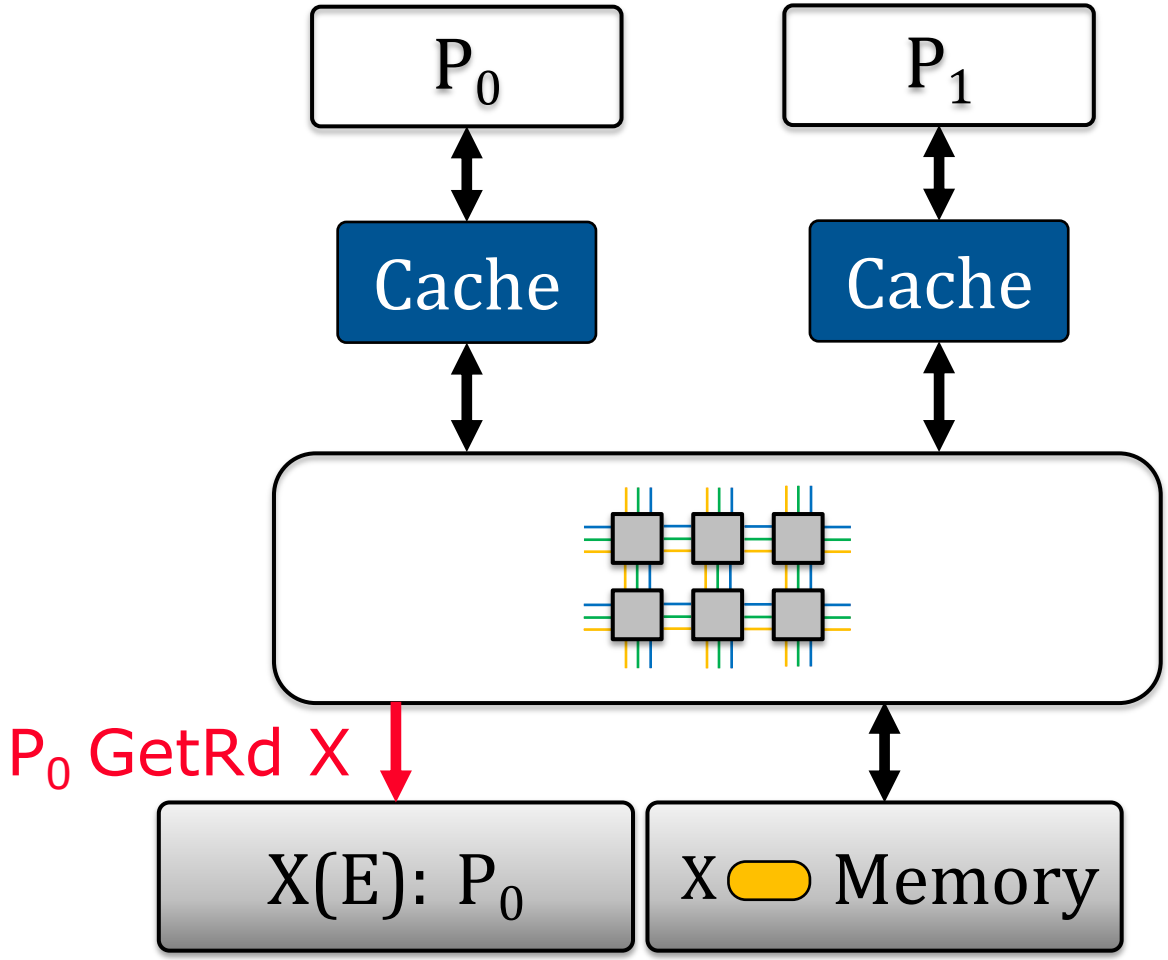
Example: Directory-Based MESI Protocol

1. P_0 Read X



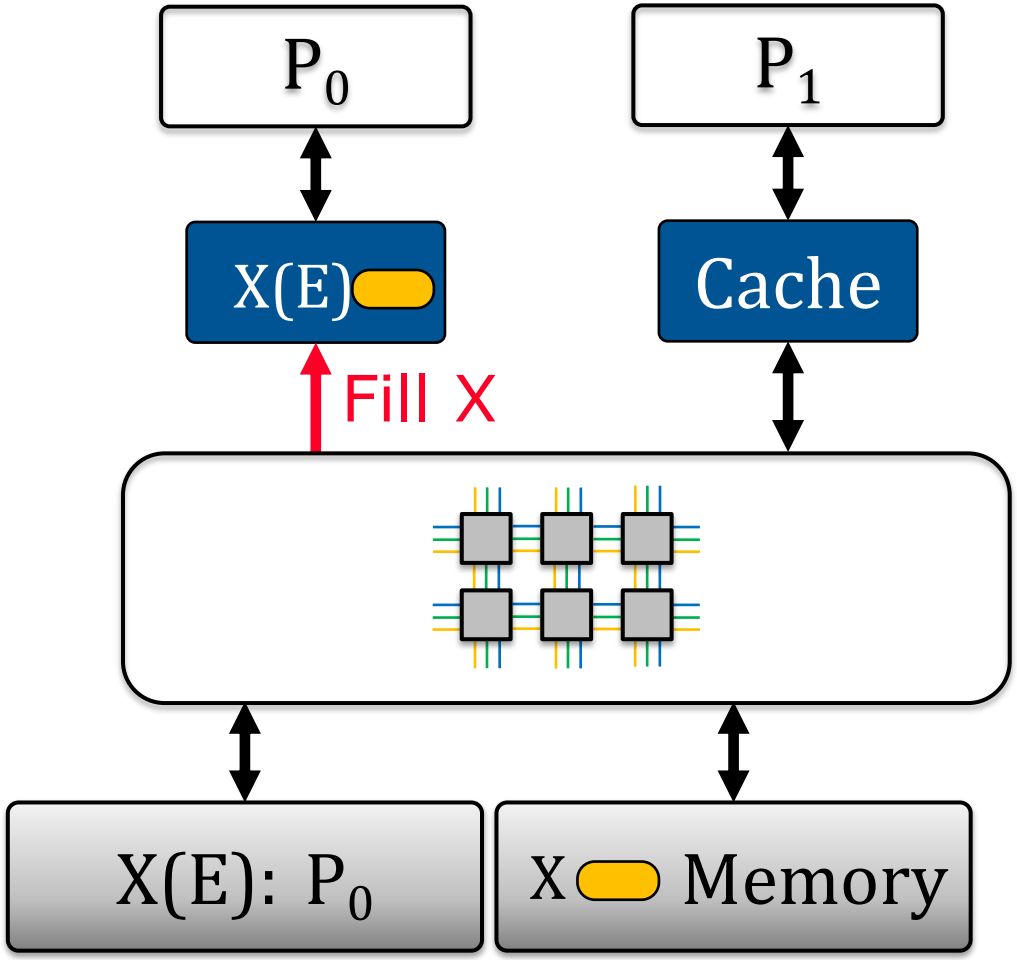
Example: Directory-Based MESI Protocol

1. P_0 Read X
   a) Cache controller forwards to directory



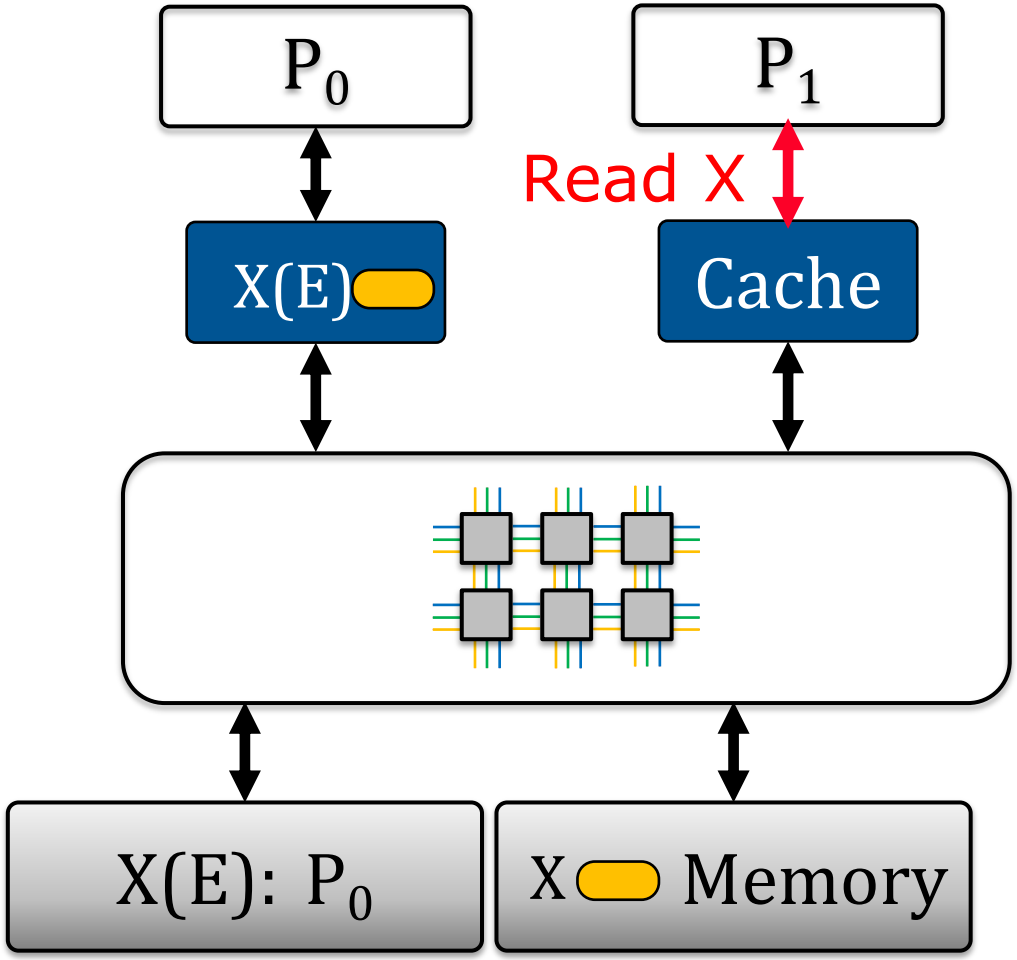
Example: Directory-Based MESI Protocol

1. P_0 Read X
   a) Cache controller forwards to directory
   b) Directory responds with a **fill** message, which has Data X → **E**



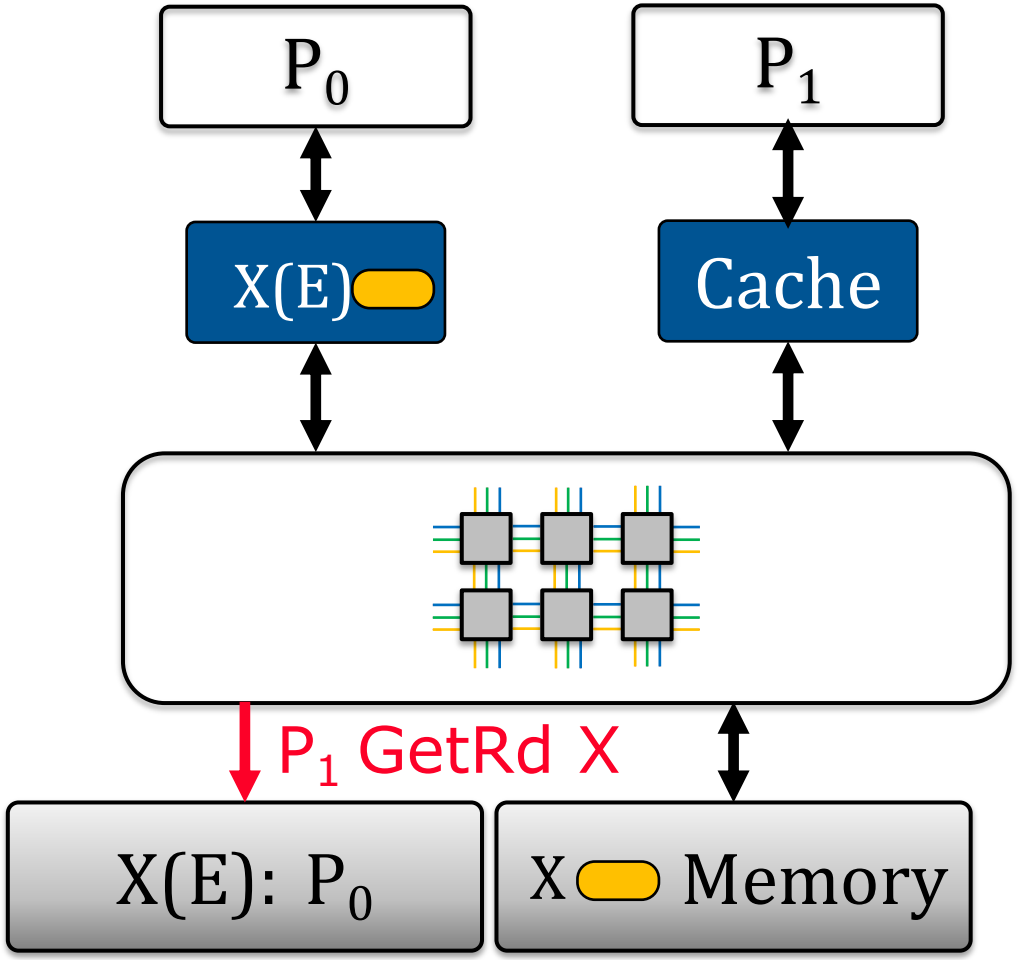
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X



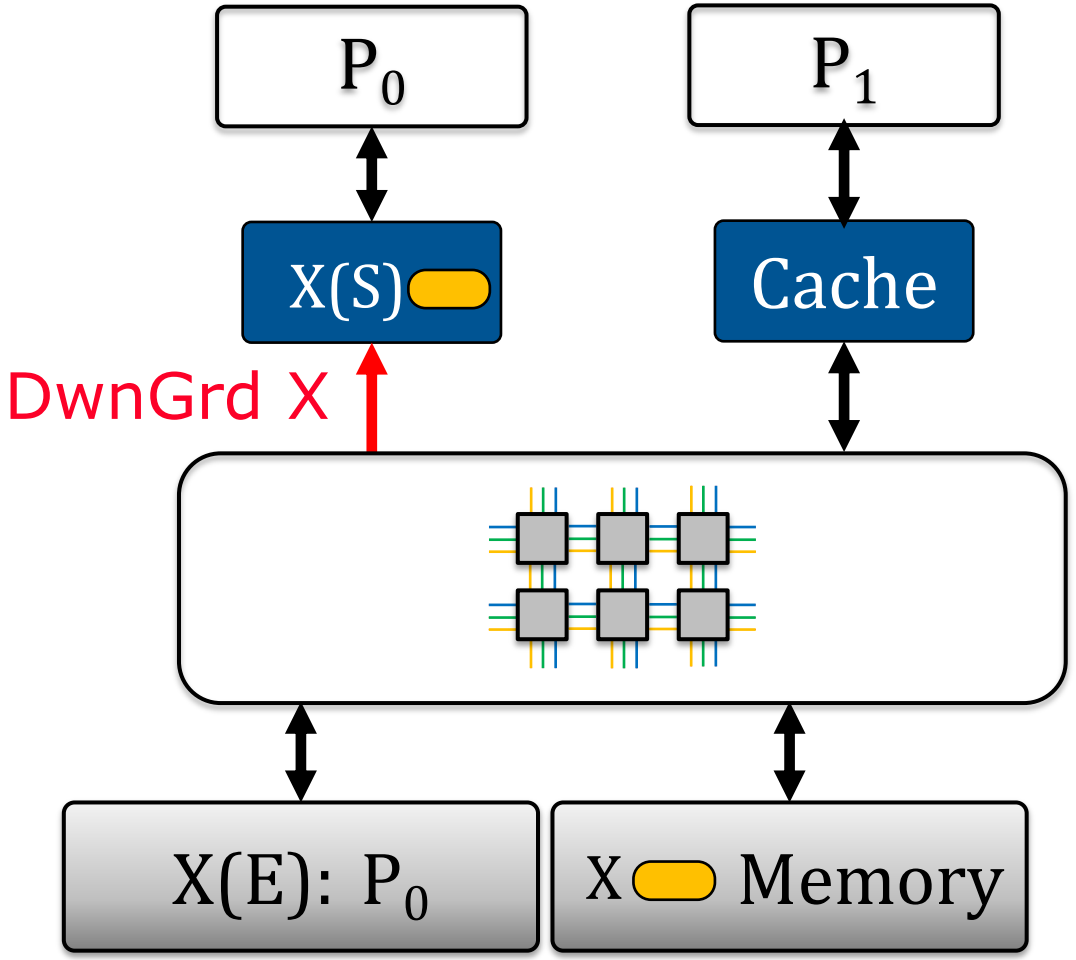
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X
   a) Cache controller forwards to directory



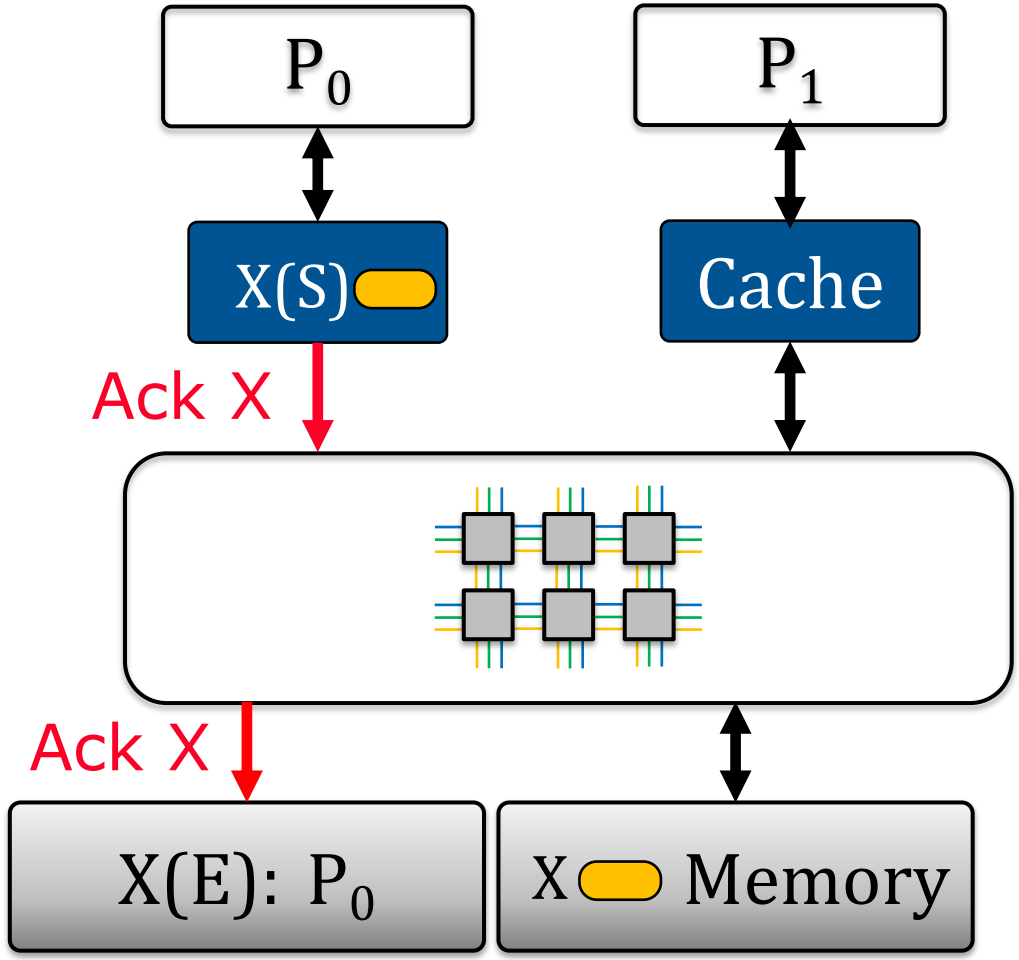
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X
   a) Cache controller forwards to directory
   b) Directory responds by downgrading P0 to **S**



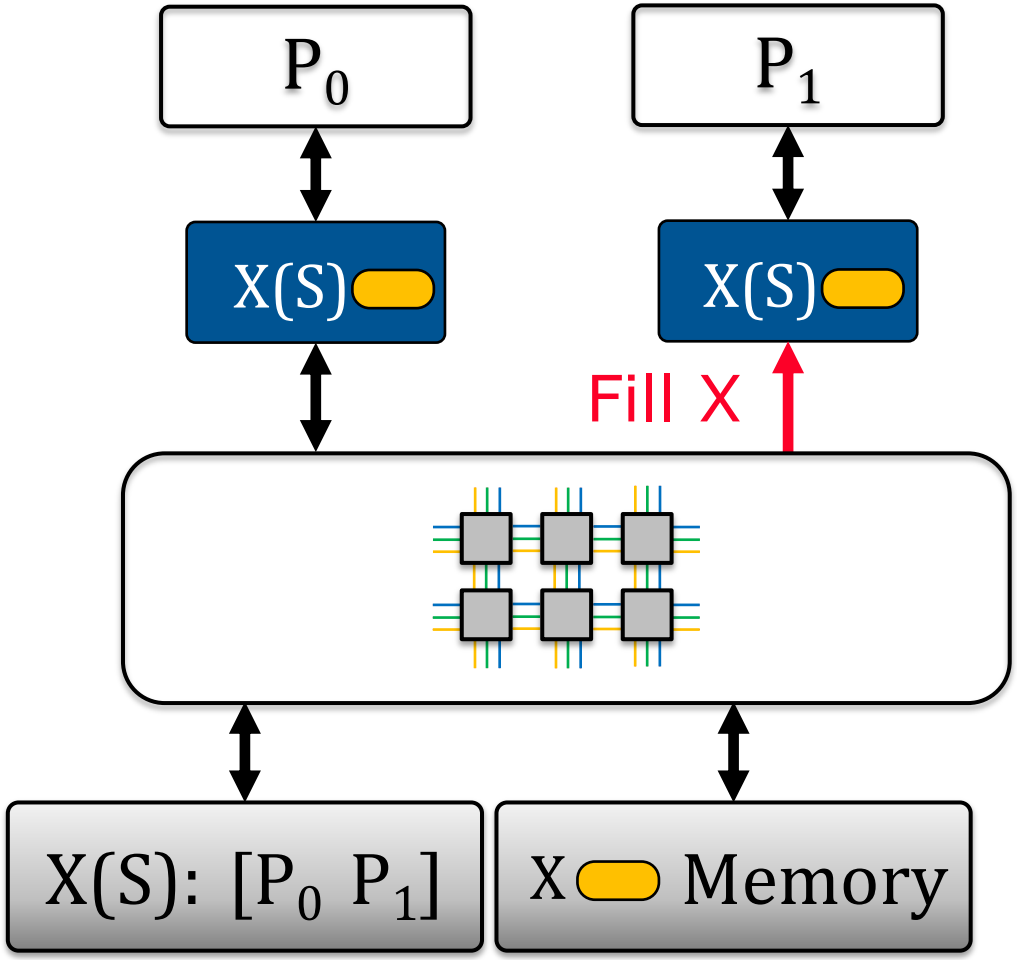
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X
   a) Cache controller forwards to directory
   b) Directory responds by downgrading P0 to **S**



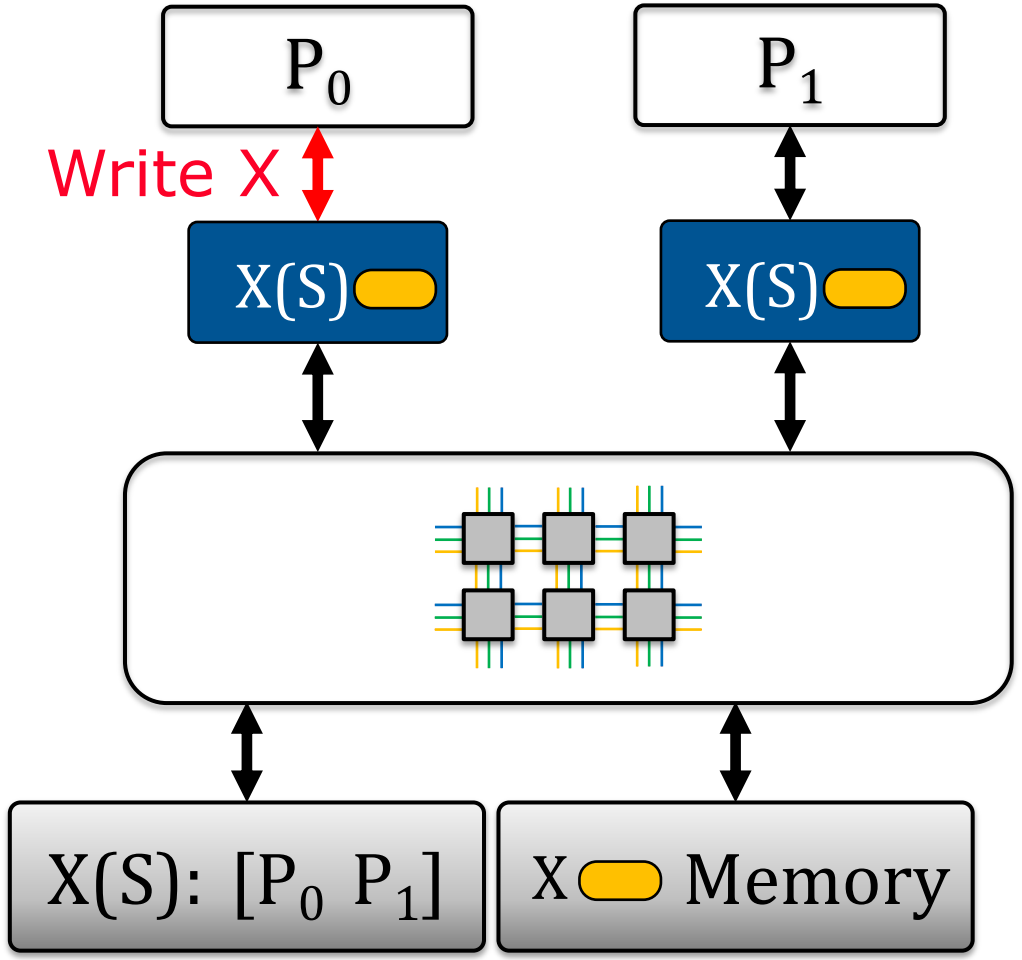
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
   a) Cache controller forwards to directory
   b) Directory responds by downgrading P0 to **S**
   c) Directory responds with a **fill** message, which has Data X → **S**



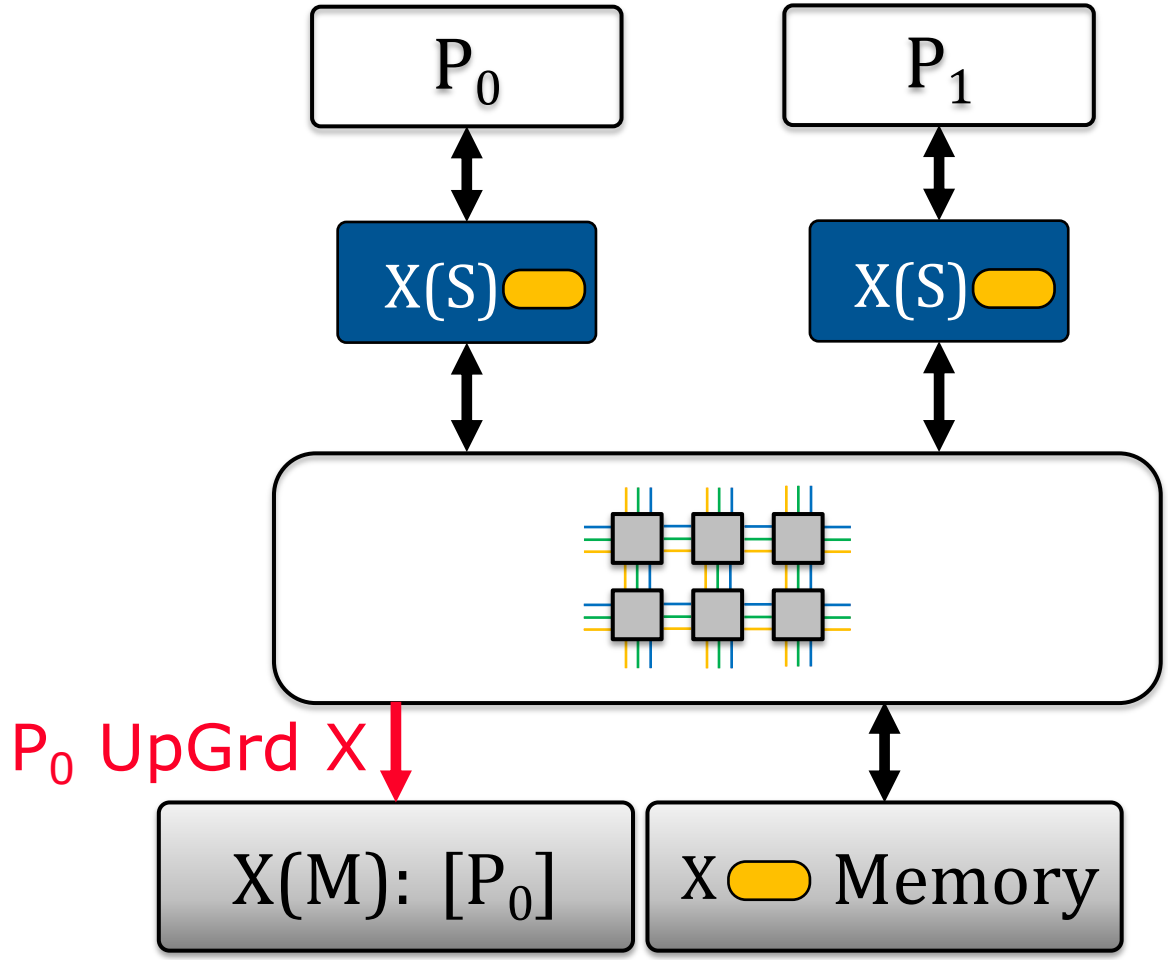
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Forward the write miss to directory



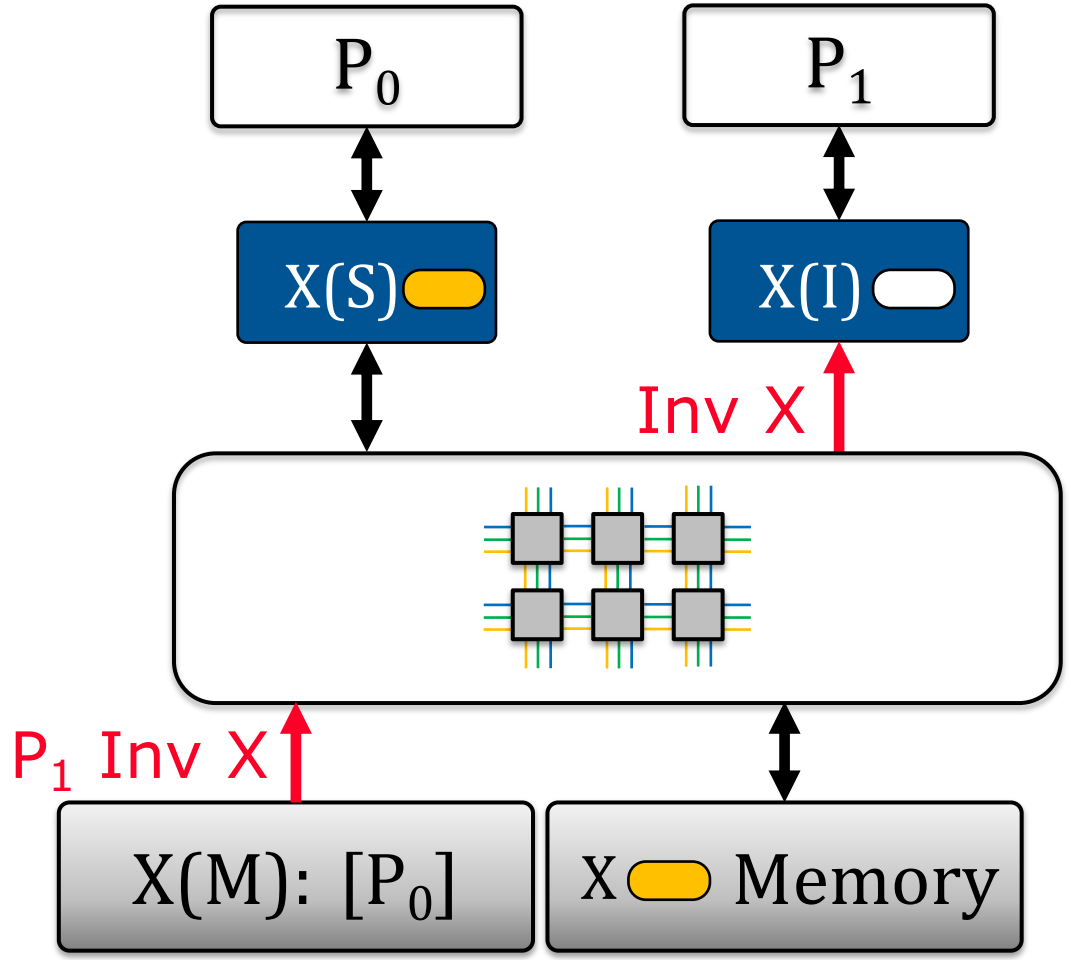
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Forward the write miss to directory
   b) Directory sends inv's to other sharers → **M**



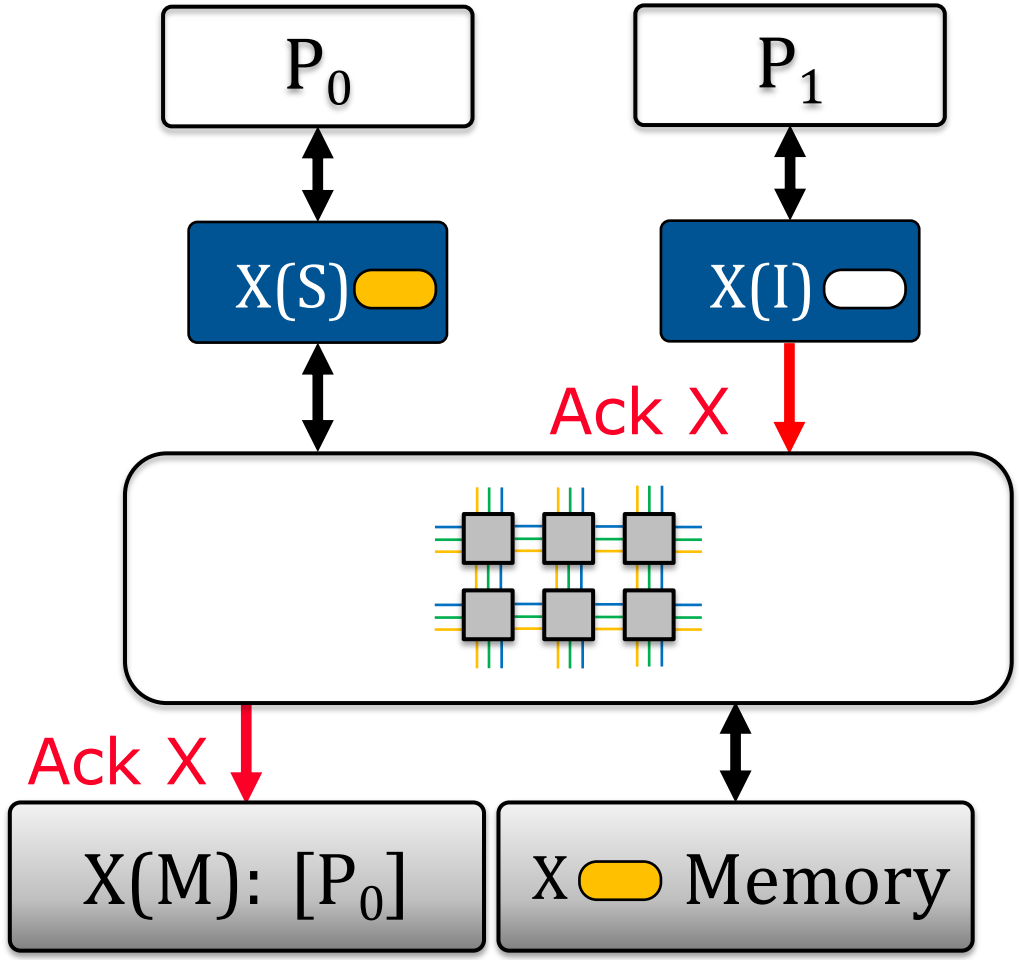
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Forward the write miss to directory
   b) Directory sends inv's to other sharers → **M**
   c) P_1 inv's block X → **I**



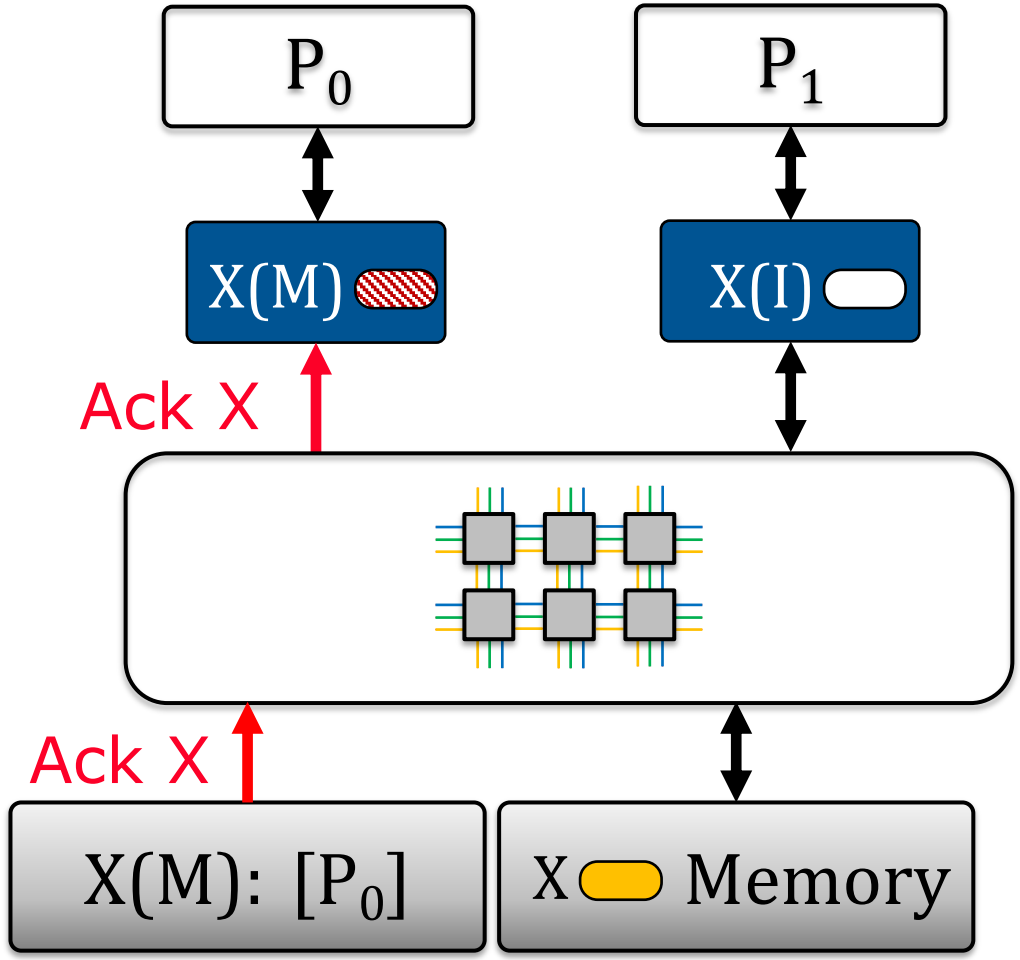
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Forward the write miss to directory
   b) Directory sends inv's to other sharers → **M**
   c) P_1 inv's block X → **I**
   d) P_1 acknowledges invalidate



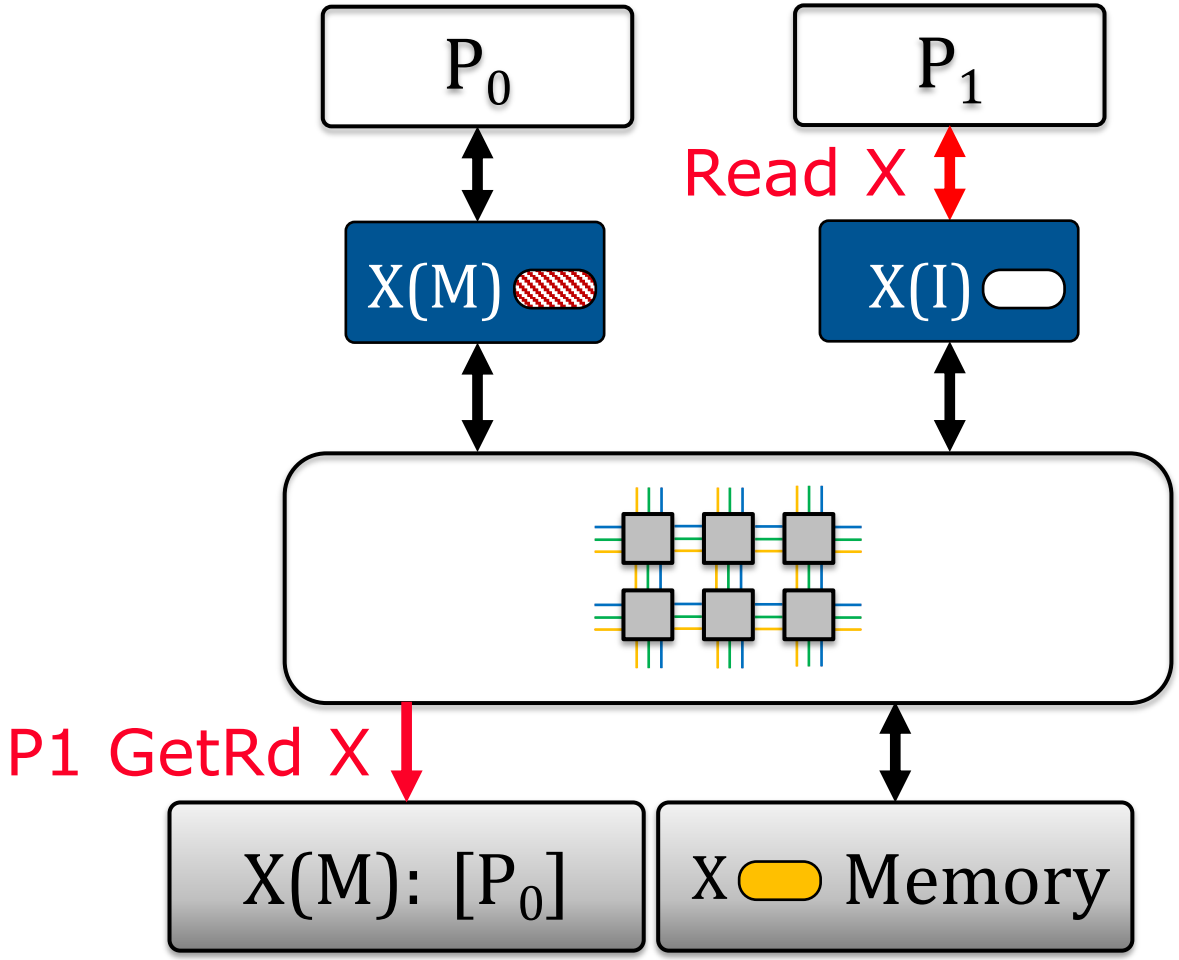
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X
   a) Forward the write miss to directory
   b) Directory sends inv's to other sharers → **M**
   c) P_1 inv's block X → **I**
   d) P_1 acknowledges invalidate
   e) Directory responds to P_0, transitions state → **M**



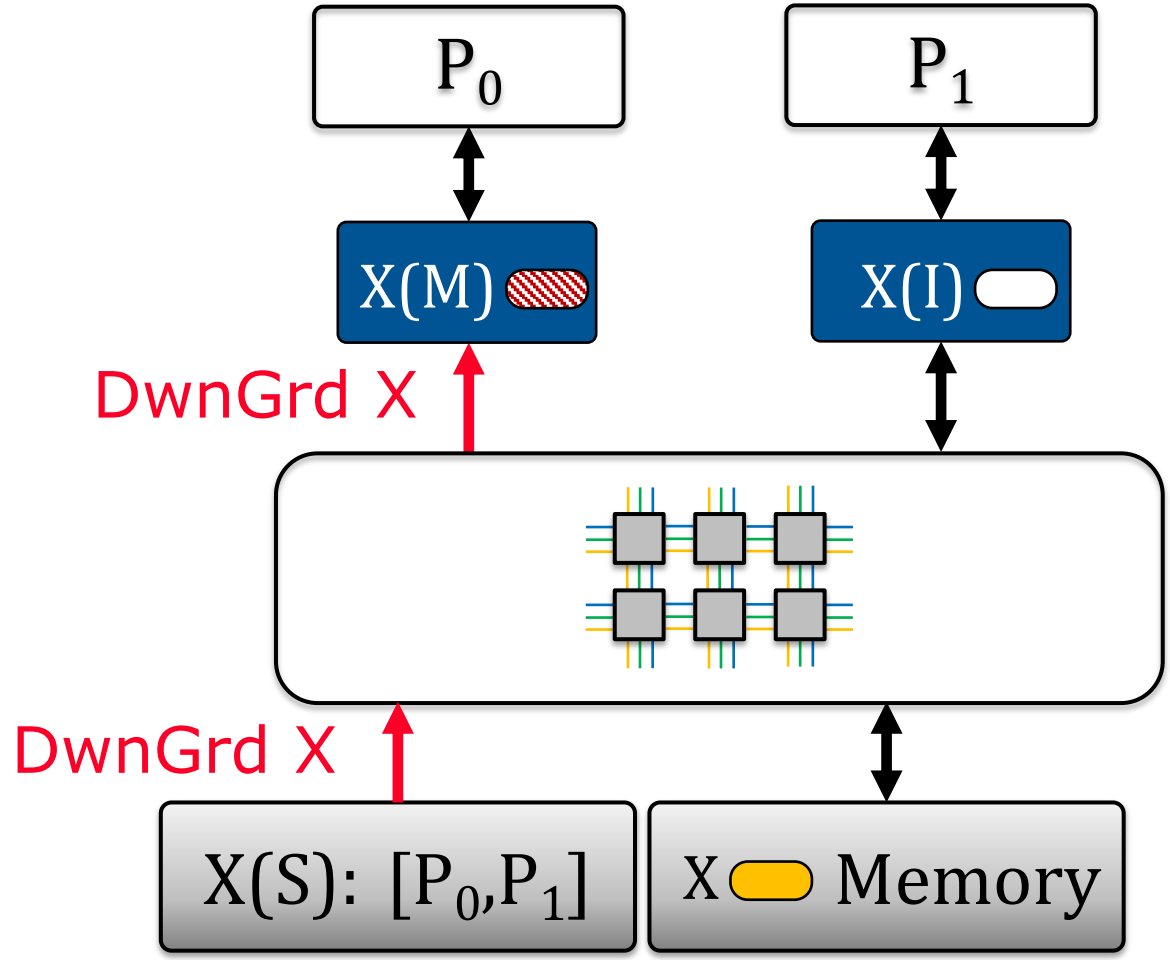
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X
   a) Send request to directory



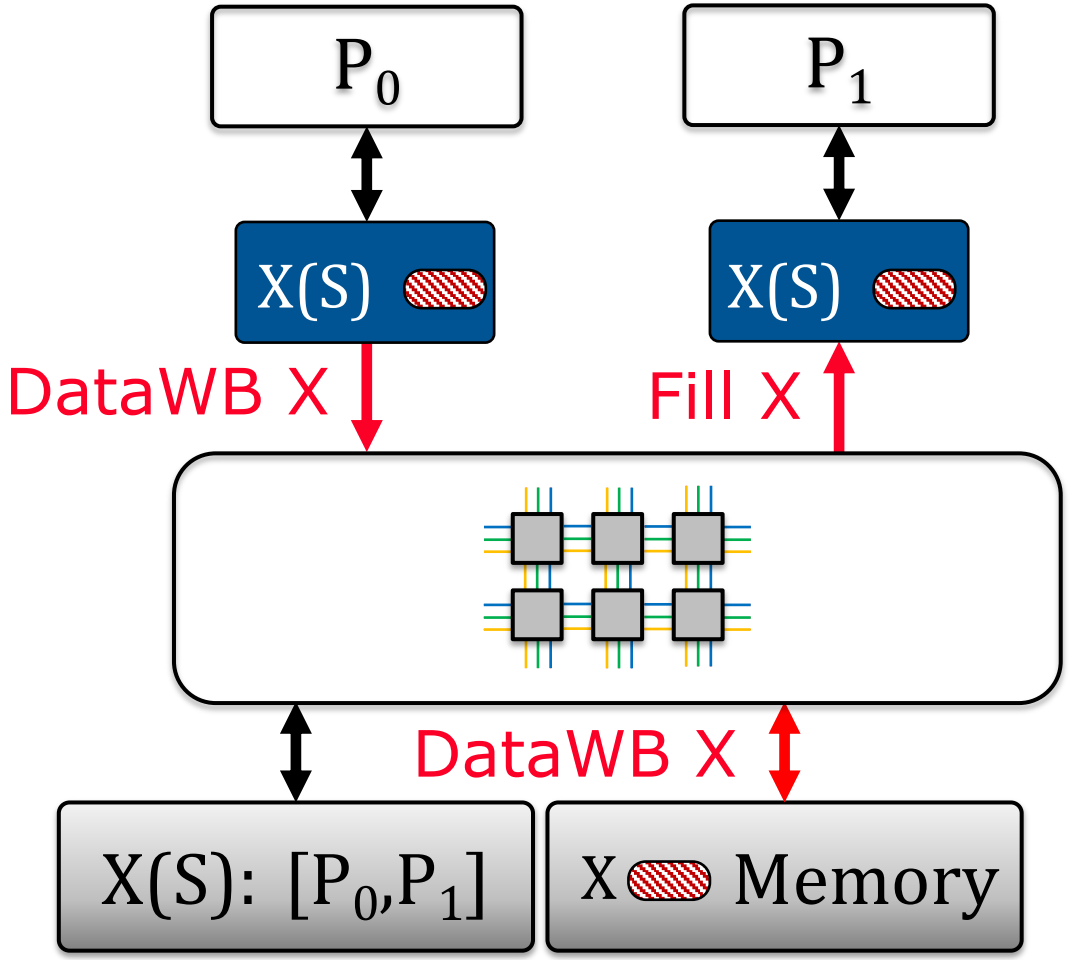
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X
   a) Send request to directory
   b) Directory sees P_0 has the most updated copy, and forwards P_1's request → **S**



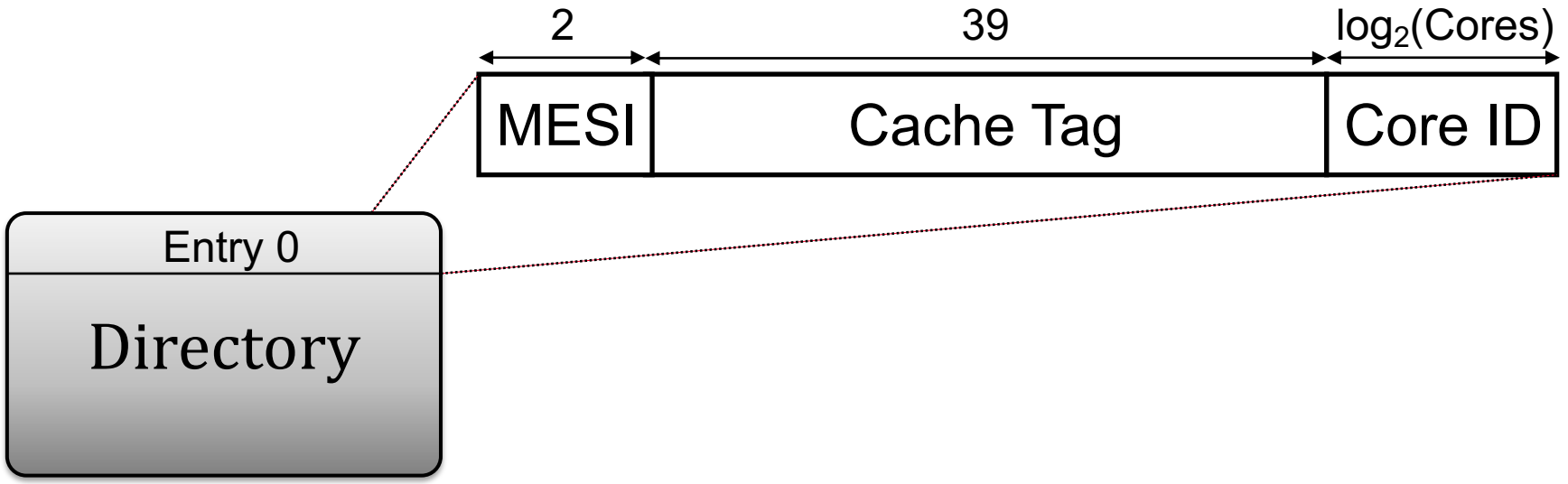
Example: Directory-Based MESI Protocol

1. P_0 Read X → **E**
2. P_1 Read X → **S**
3. P_0 Write X → **M**
4. P_1 Read X
   a) Send request to directory
   b) Directory sees P_0 has the most updated copy, and forwards P_1's request → **S**
   c) P_0 sends P_1 the data → **S** and writes back to memory



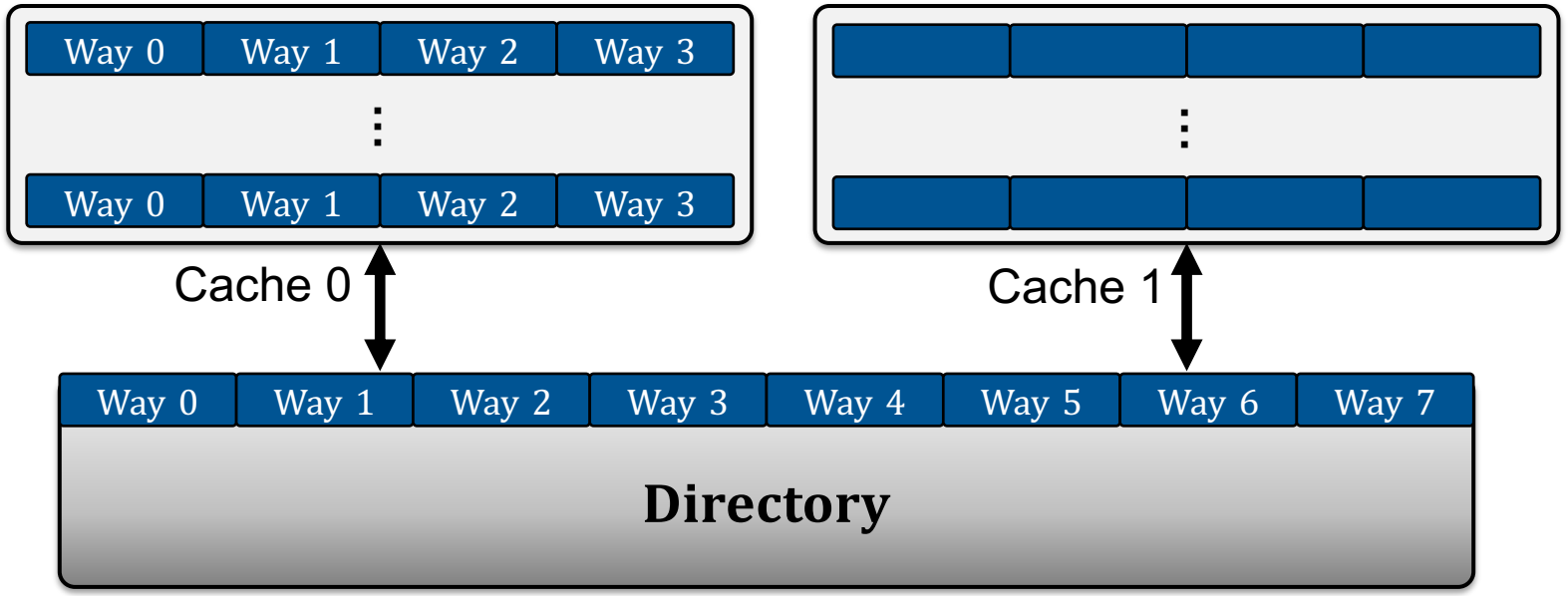
Anatomy of a Directory Entry

- Each entry stores:
  1. Bits storing the MESI coherence state
  2. Cache tag
  3. Bits identifying the core associated with this entry



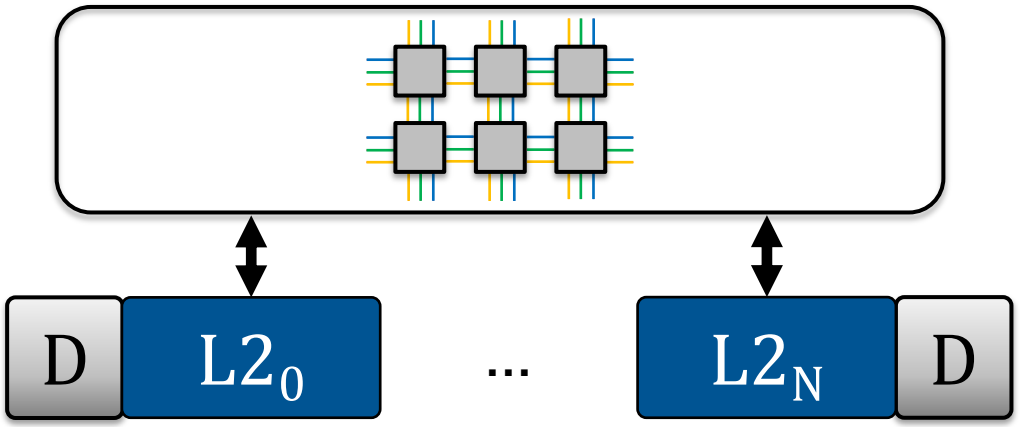
Duplicate Tag Directory

- Stores all tags that are present in the L1 caches
  - Means that the set-mapping functions must be identical
  - e.g., 2 processors, with 4-way associative caches, means that each directory set must be 8-way associative



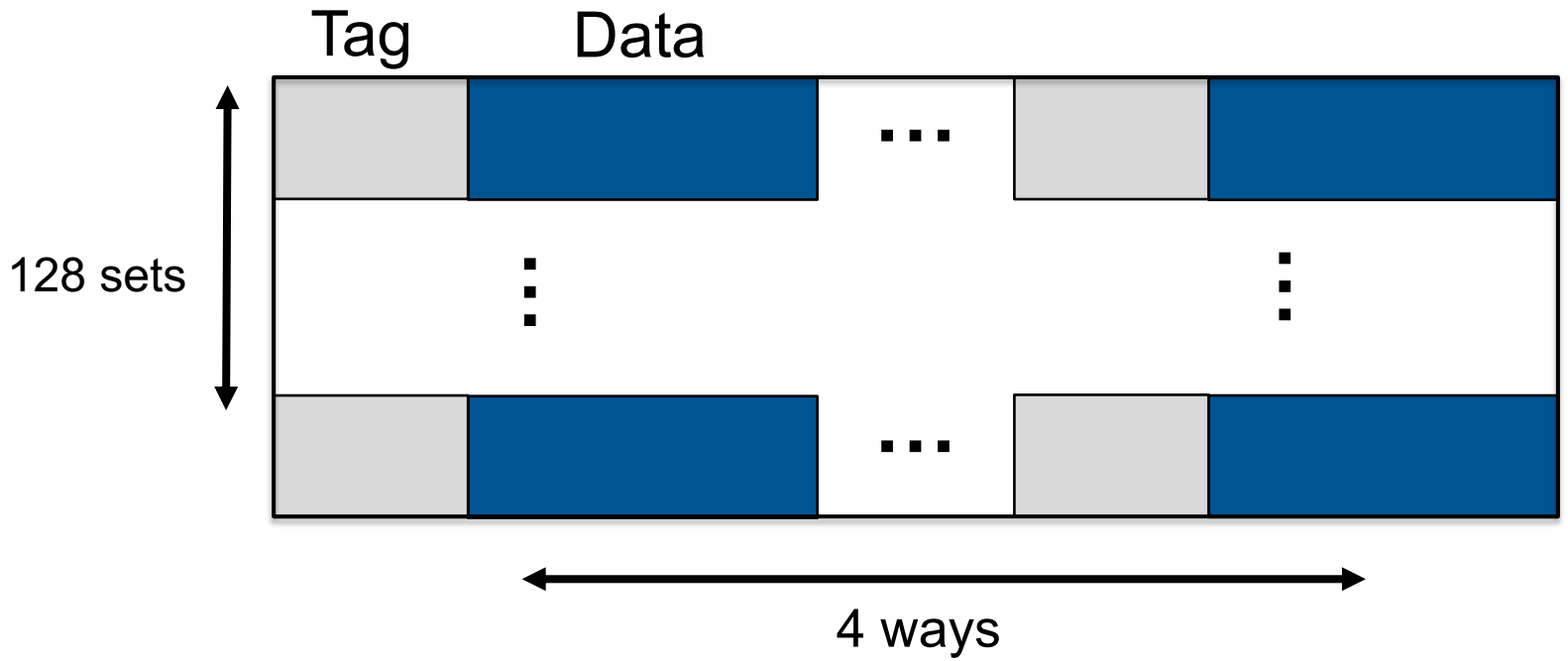
Distributed Duplicate Tag Directories

- Centralized directory can also be a source of contention, just like a bus
  - Insight: L2 caches are multi-banked and distributed across chip
  - Solution: Distribute the directory with the L2 banks
    - Each block has a home node determined by address interleaving
  - Each directory still requires associativity equal to sum of L1 caches



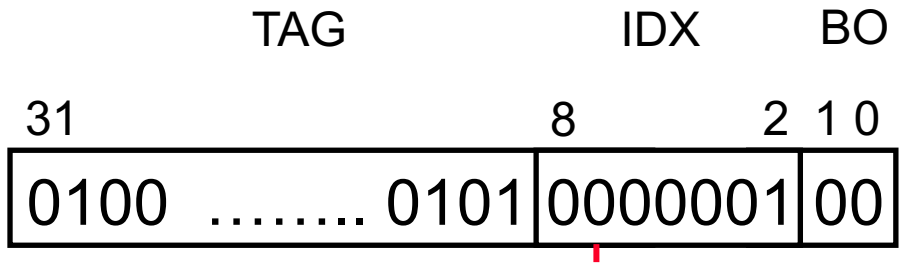
Reminder: Set Mapping in Caches

- Assume a 32-bit address space
- 2KB cache, 4-way associative, 4-byte blocks
  - 2KB with 4B per block = 512 blocks in the cache
  - Organization is 128 sets x 4 ways

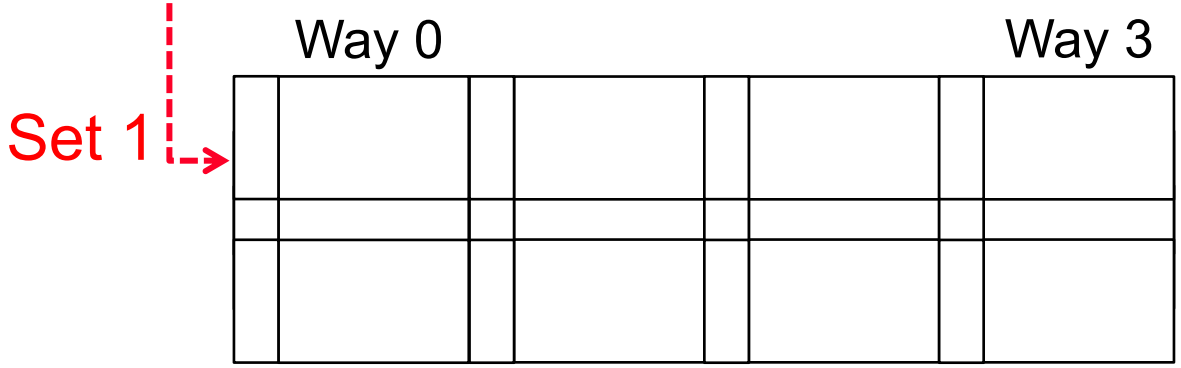


Reminder: Set Mapping in Caches

- Assume a 32-bit address space
- 2KB cache, 4-way associative, 4B blocks

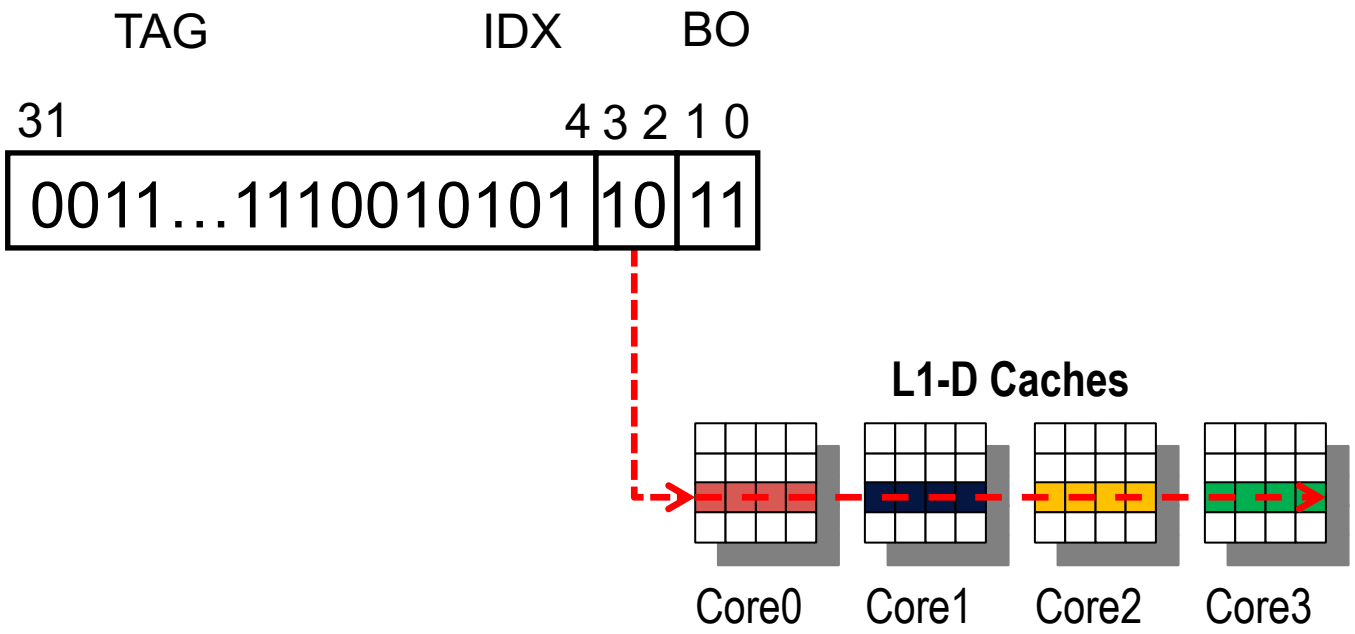


4B block, B. Offset = 2 bits
128 sets, IDX = 7 bits
TAG = rest of the bits



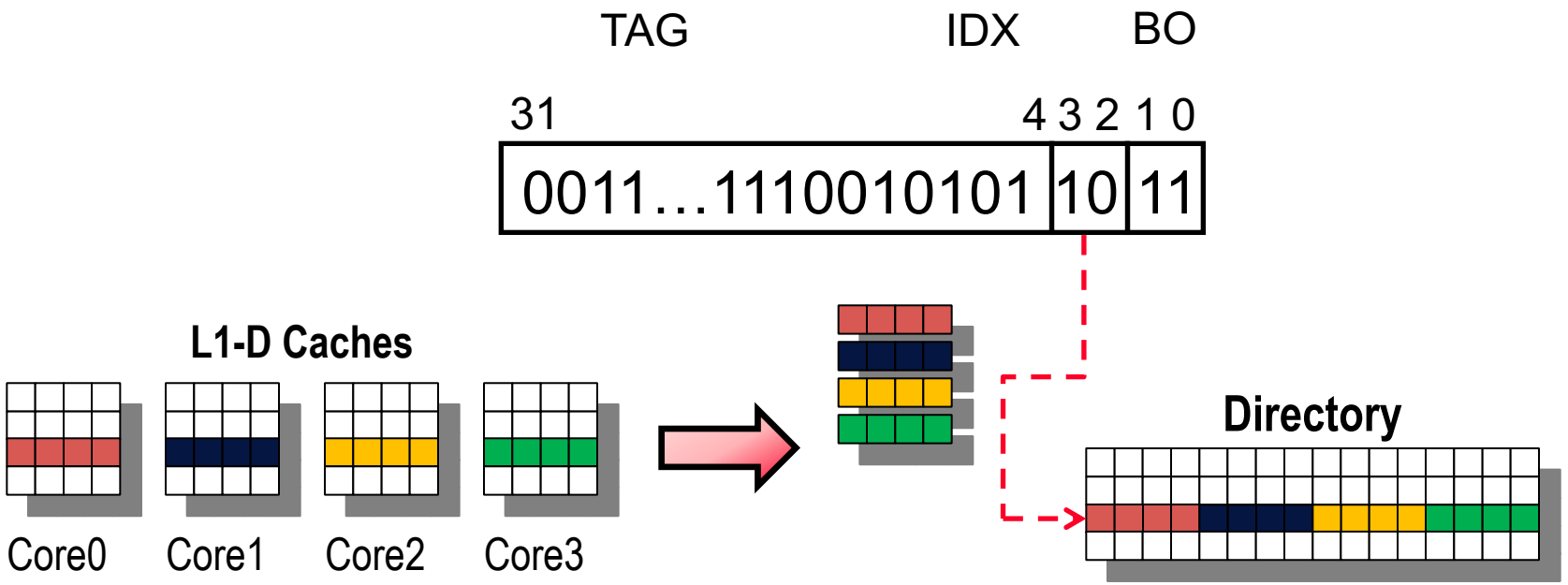
How are the Caches Accessed?

- Simpler example for illustrative purposes
- L1 Caches: 4B blocks, 4-way associative, 4 sets
- Note: Different address than previous slide



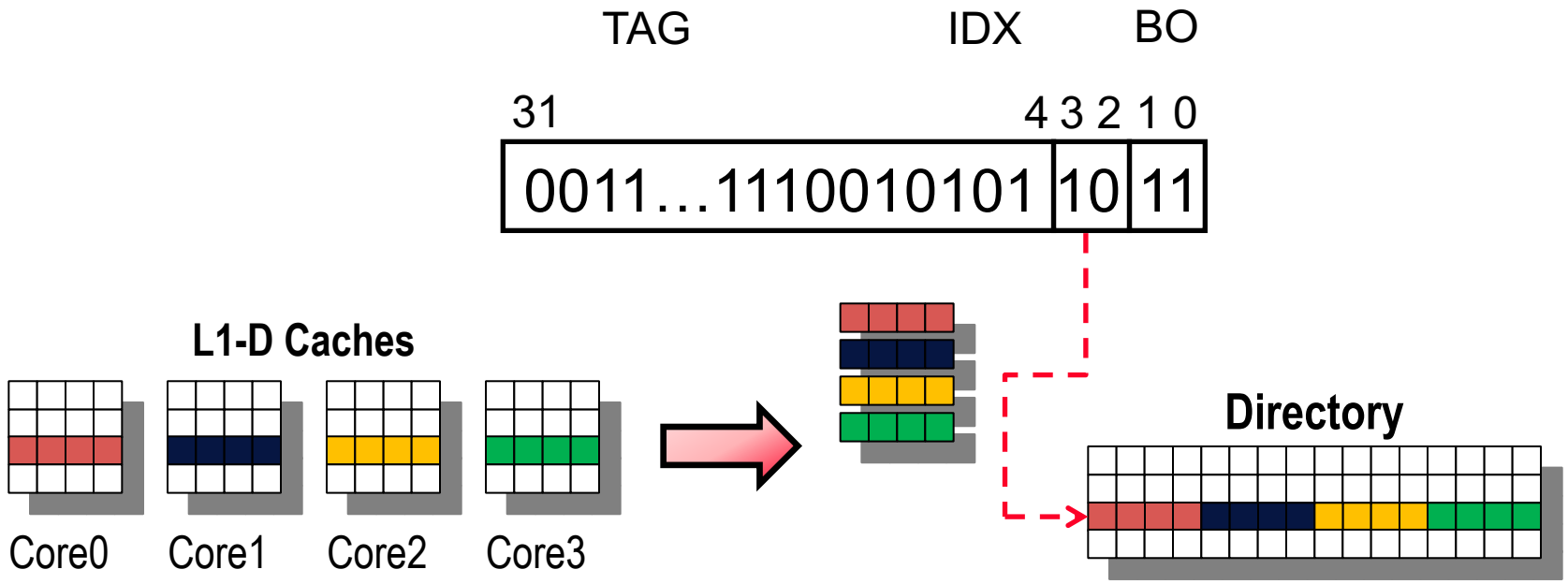
How is the Directory Accessed?

- Assume a centralized directory
- Simple solution:
  - Use the same set mapping index to pick the directory set



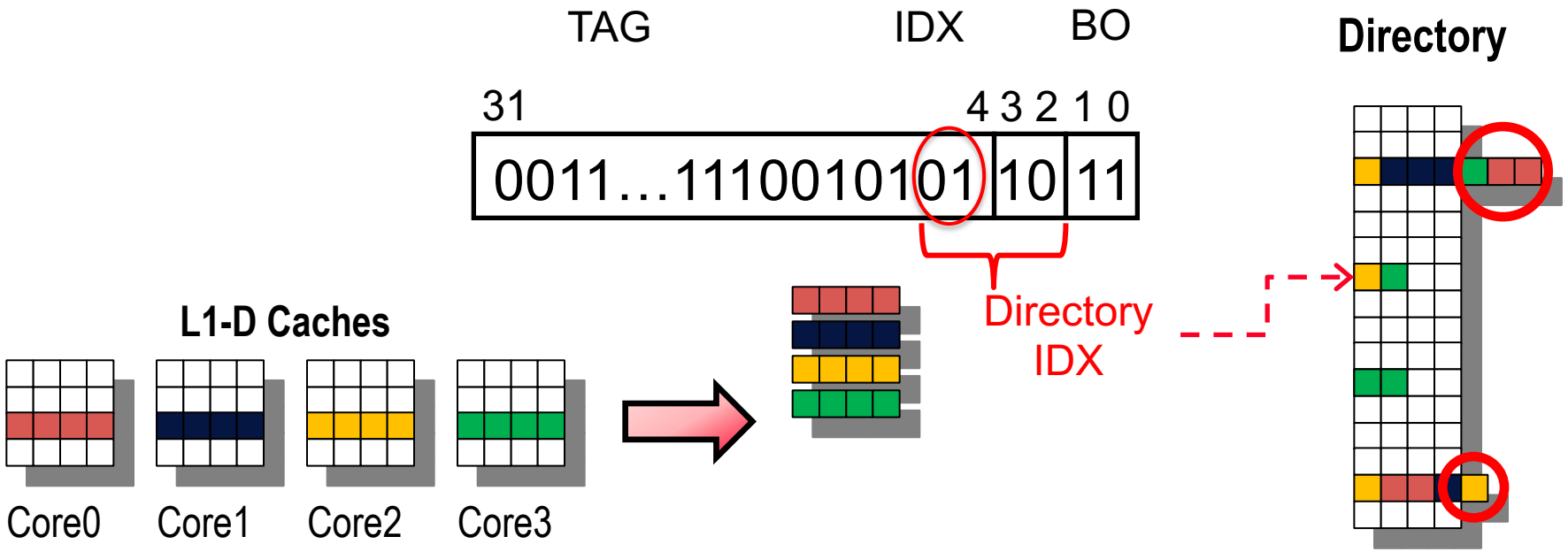
How is the Directory Accessed?

- But now 16-way (4 cores x 4-way)
- Associativity is bad for latency & power!



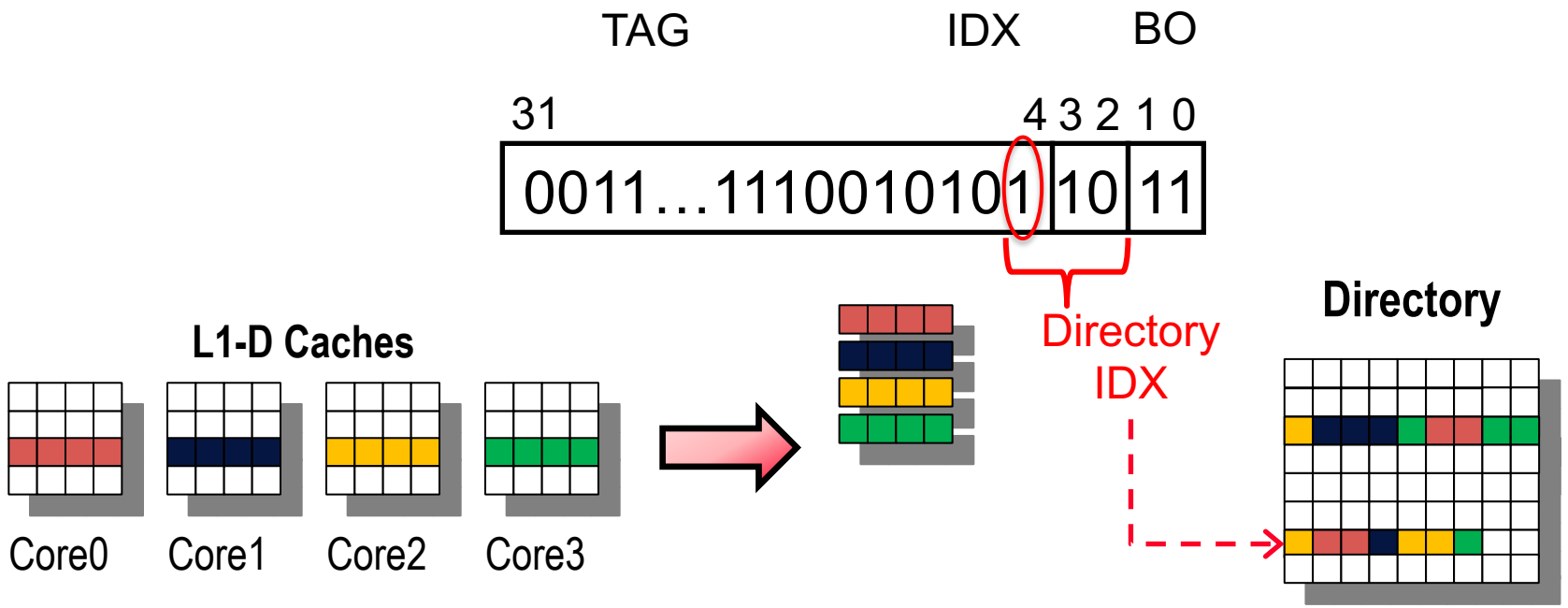
Sparse Directories

- Can continue splitting
- But, will run into low associativity
  - Contention in sets 0010 and 1110



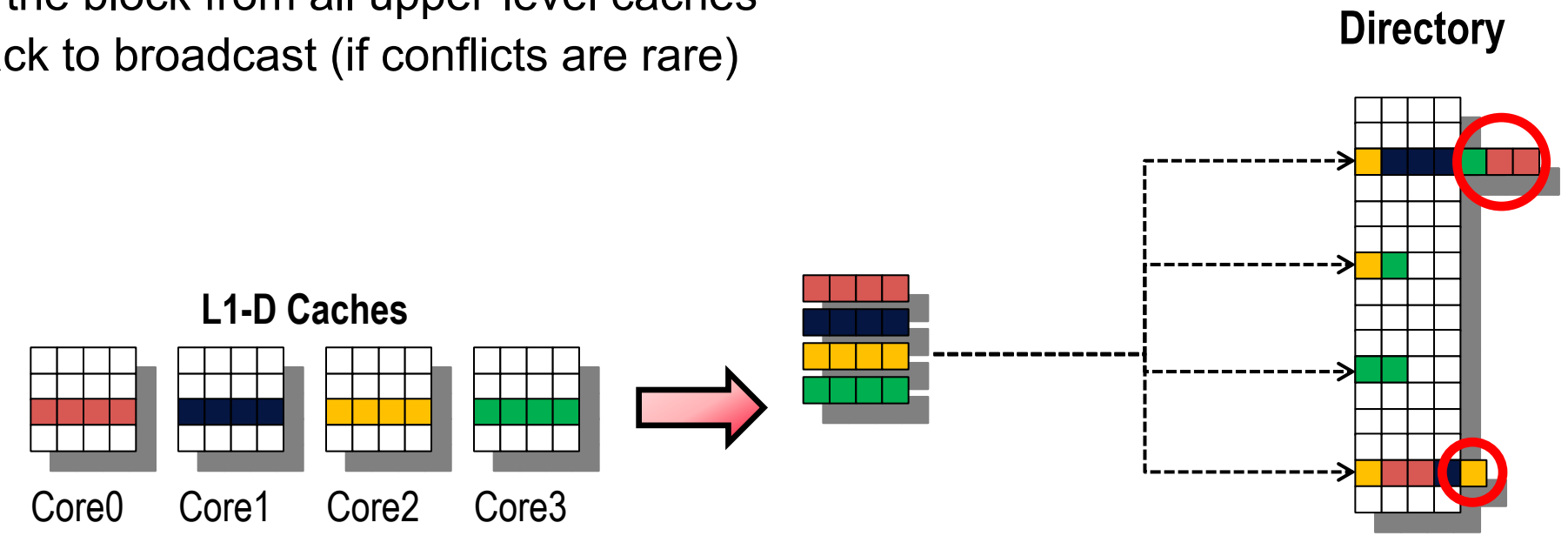
Sparse Directories

- Duplicate tags require massive associativity, equal to the sum of all L1's in the system
  - **Insight:** Split the L1 ways across multiple directory sets, allows us to reduce associativity. Sets are easier to add than ways!



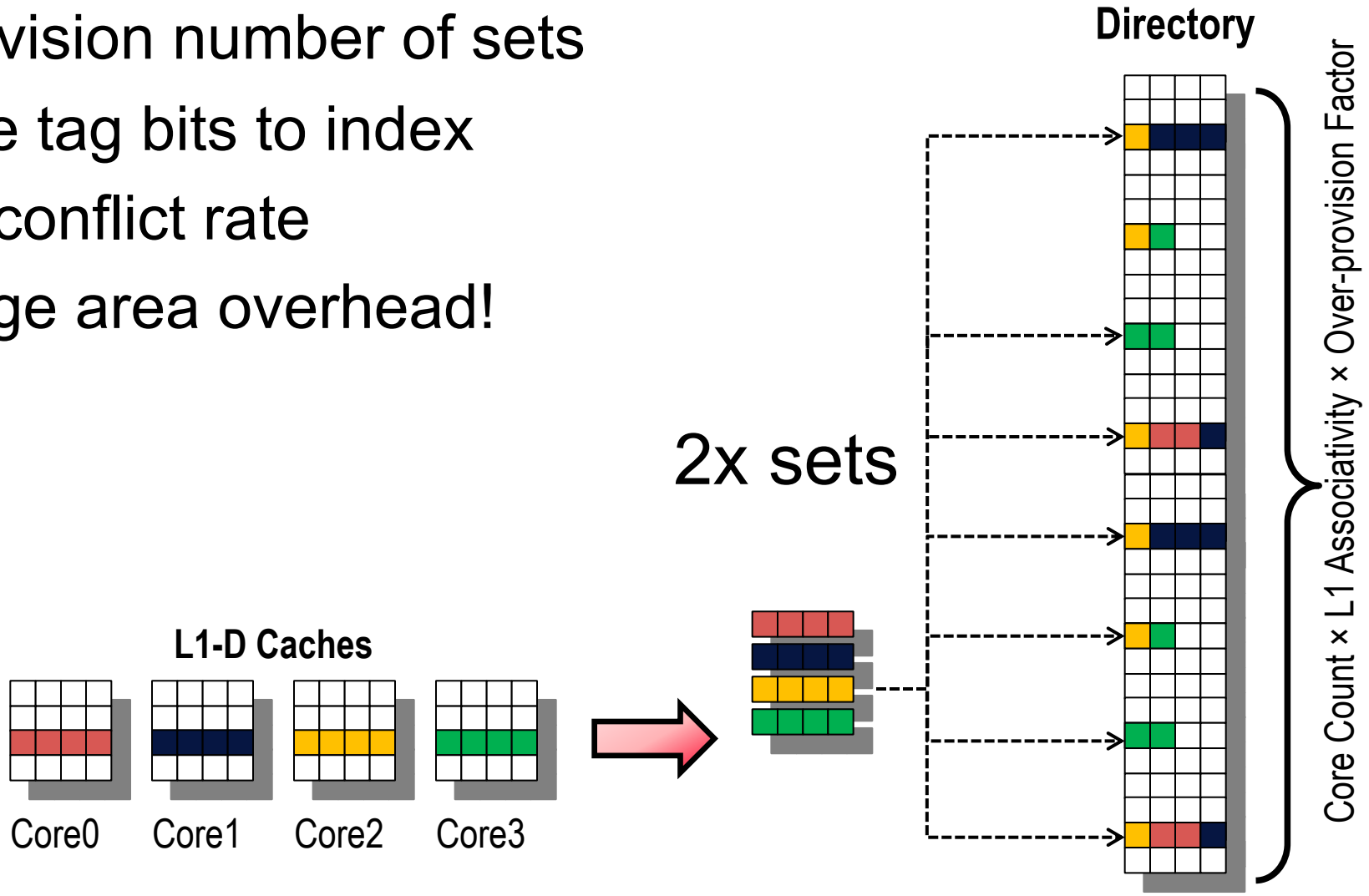
Sparse Directories

- By reducing associativity, we introduce conflicts
  - Eviction policy? e.g., Least recently used (LRU)
- Now we need a mechanism to handle directory conflicts while maintaining coherence
  1. Recall the block from all upper-level caches
  2. Fall back to broadcast (if conflicts are rare)



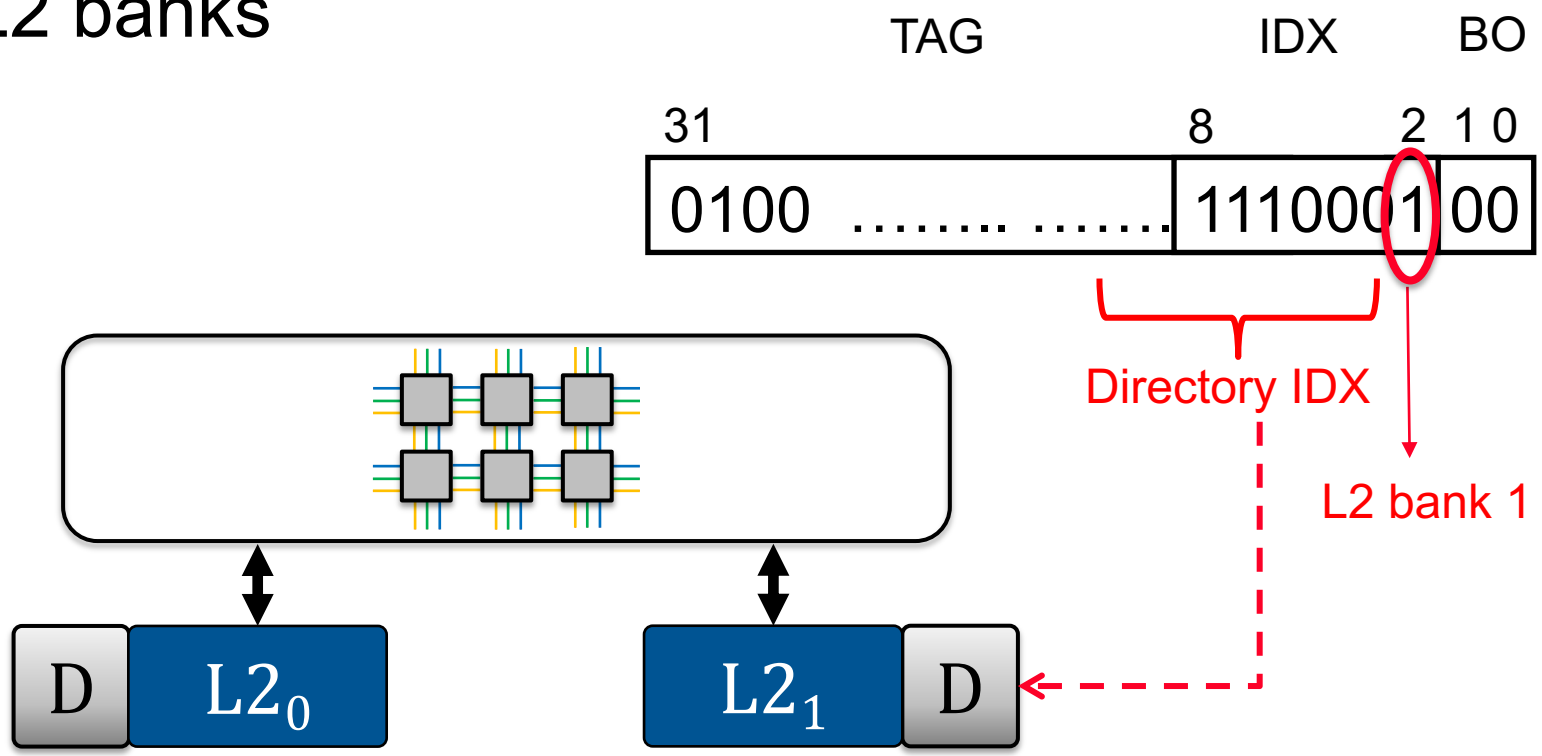
Sparse Directories

- Over-provision number of sets
- Use more tag bits to index
- Pro: low conflict rate
- Con: Large area overhead!



Back to Distributed Directories

- Divide the L1 sets across L2 banks
- Use L1 low-order index bits to find the directory
  - Rest of the L1 index bits + tag bits for directory
- E.g., for two L2 banks



Bit Vector Directory Entries

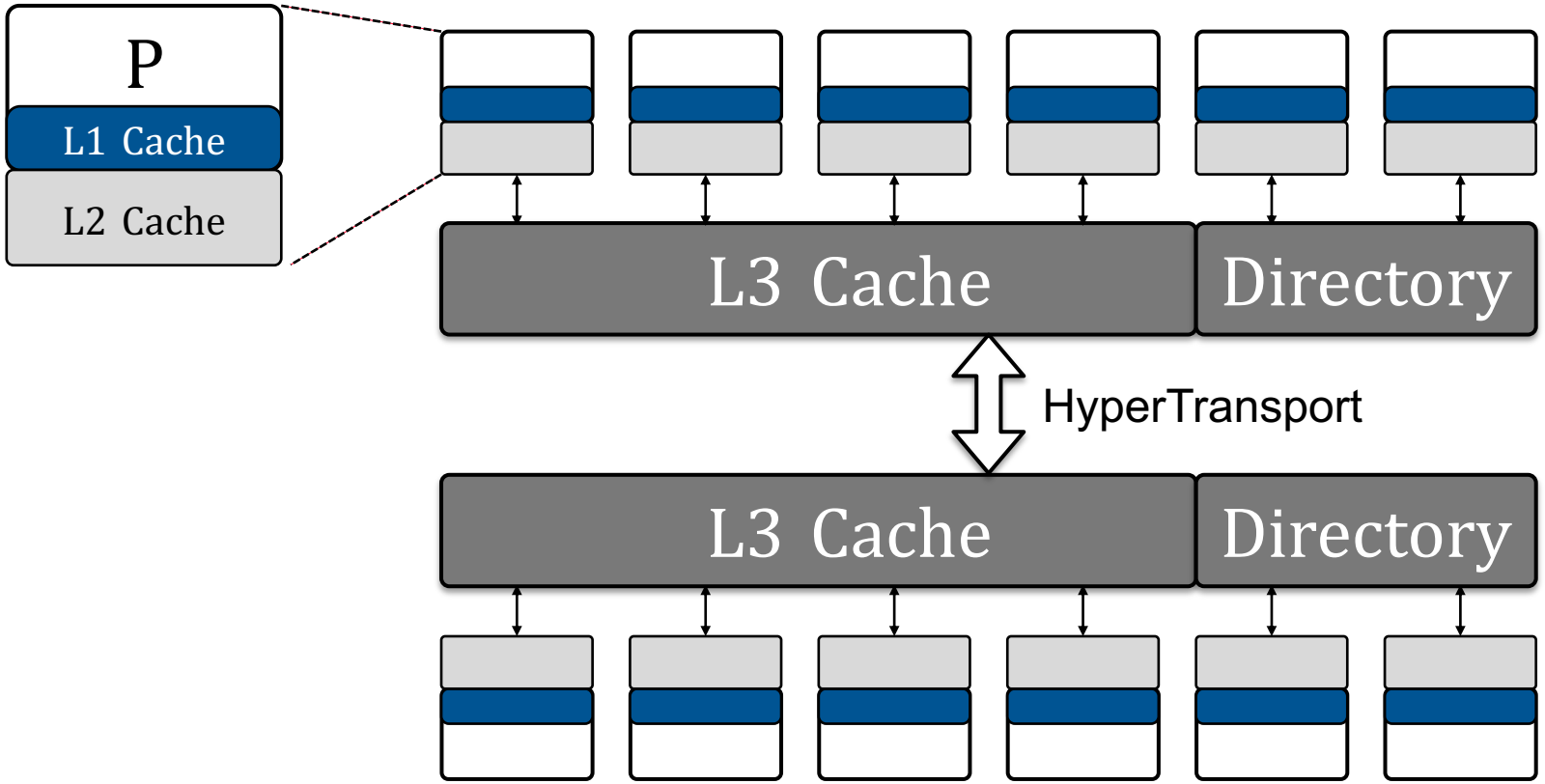
- Before, we stored all of the L1 cache tags
  - Core ID's were explicitly stored

- Instead, store a bit vector (1b per core) w. each tag
  - If a bit is 1, that core shares this cache block

= Shared by C_0,C_2

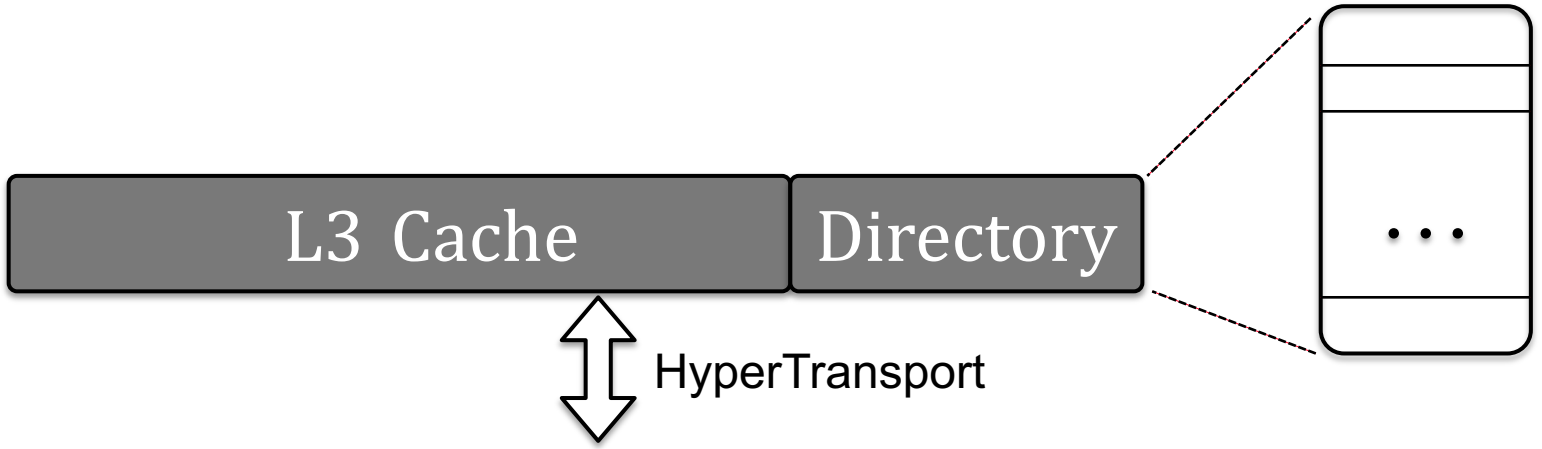
Real-World Directory Structure

- e.g., AMD Opteron “Magny Cours” Processor
  - 4 x 12 core multi-chip modules, with hierarchy shown below



Real-World Directory Structure

- L2 caches:
  - 512kB capacity, 64B lines
  - Inclusive of 64KB L1 caches
- L3 cache tracks which of the L2s share a block
  - 48 possible locations (4 chips, 12 cores each)



Summary

- Cache coherence ensures a unified view of each memory location in isolation
  - All cores will see the same sequence of values
- Sample coherence protocols
  - Bus-based MSI, optimized MESI
- Scalable interconnects demand directory protocol
  - Most common directory is bit vector, use limited ptr. for scalability
  - Scaling exact coherence is an ongoing and active research area!