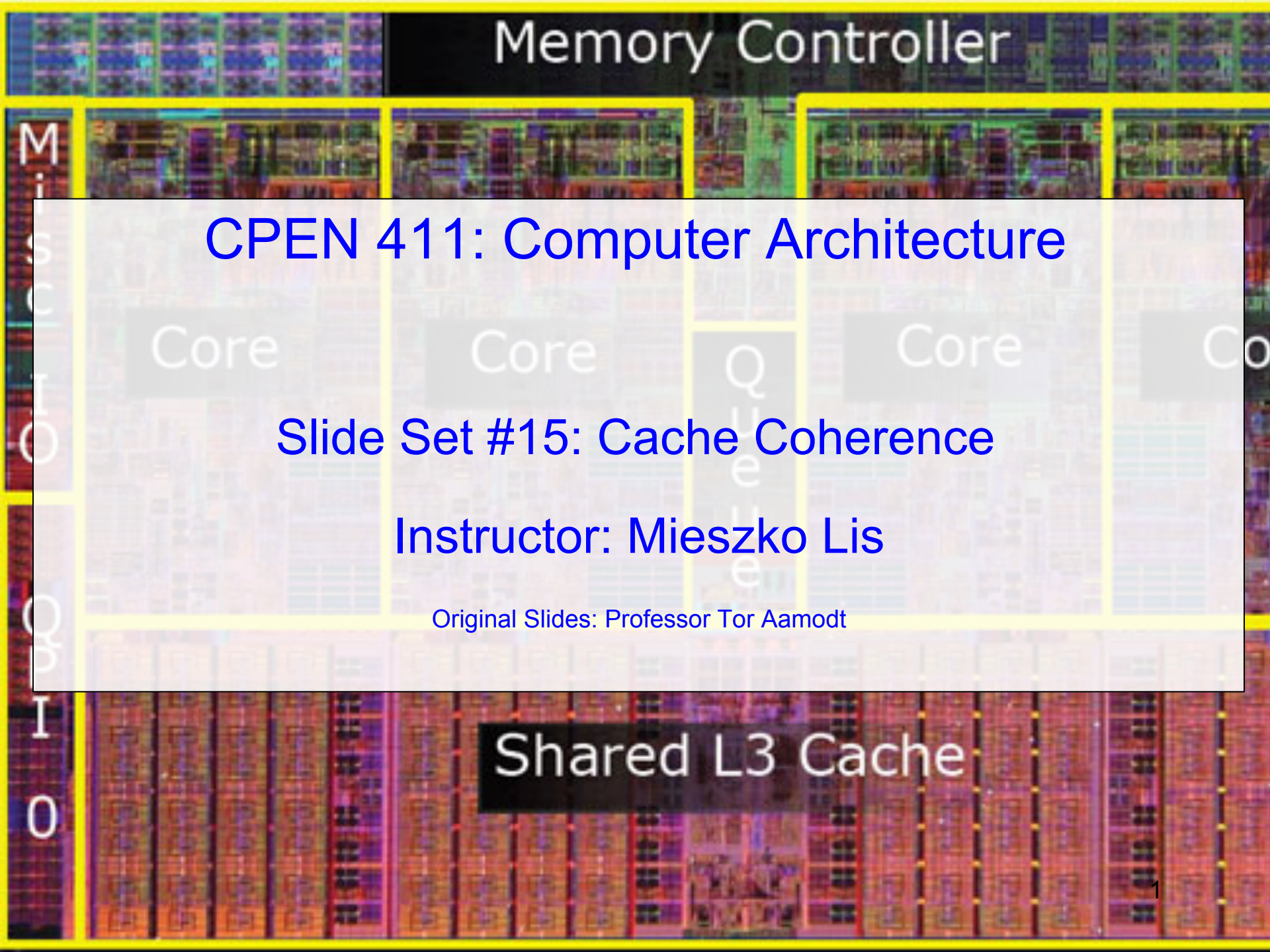


CPEN 411: Computer Architecture

Slide Set #15: Cache Coherence

Instructor: Mieszko Lis

Original Slides: Professor Tor Aamodt

Introduction to Slide Set 15

- In the prior slide set we considered **memory consistency** or the order that memory accesses to multiple locations from one thread may become visible to another thread running on a different processor.

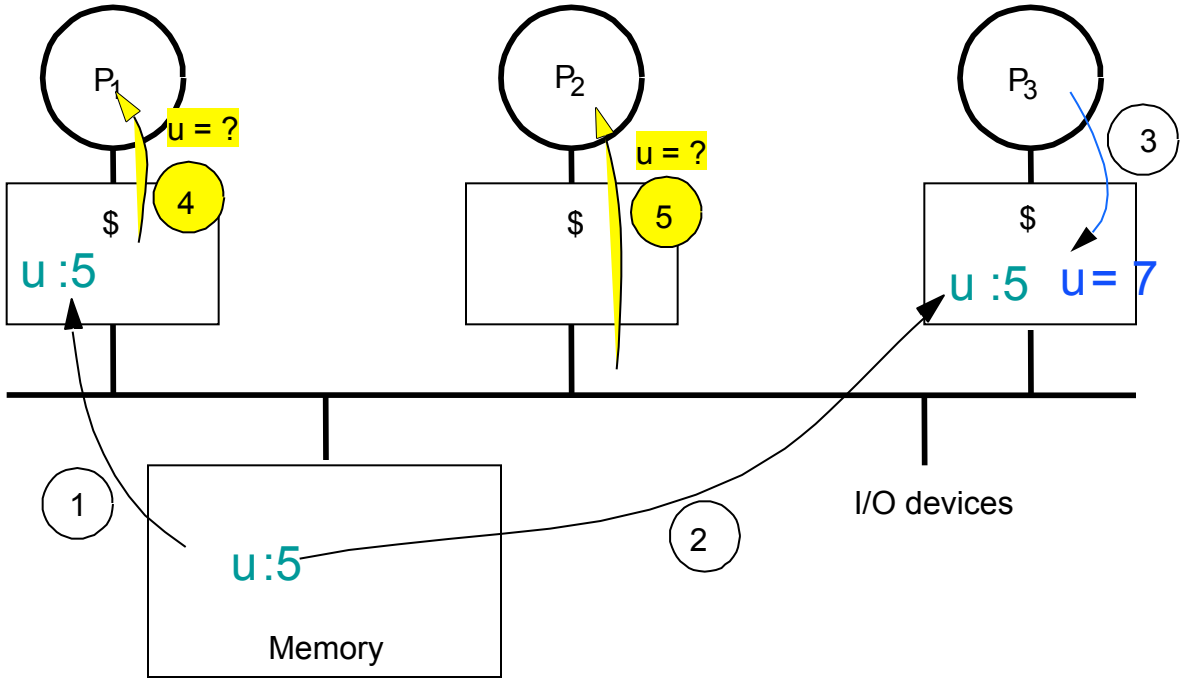
- In this slide we consider **cache coherence** or how an update to a single location on one core becomes visible to threads running on a different core.

Learning Objectives

By the time we finish talking about this slide set in lectures you should be able to:

- Describe the snoop based approach to implementing shared memory.
- Evaluate which events are generated when loads and stores operate with a simple snoop cache coherence protocol for writeback caches.
- Describe how a directory based cache coherence protocol differs from a snooping cache coherence protocol
- Evaluate which events are generated when loads and stores operate with a simple directory cache coherence protocol

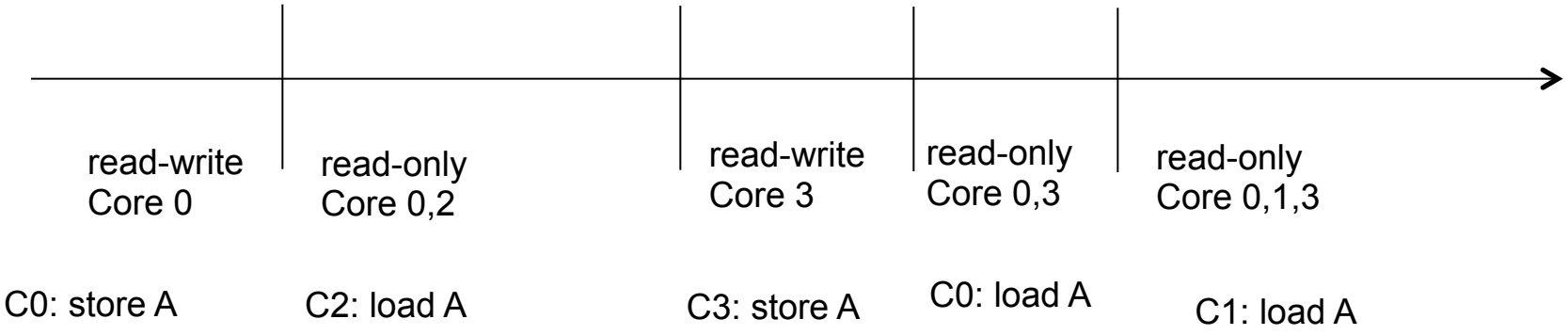
Recall: Cache Coherence Problem



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on order of which cache writes back value first
- Unacceptable situation for programmers

Coherence Invariants

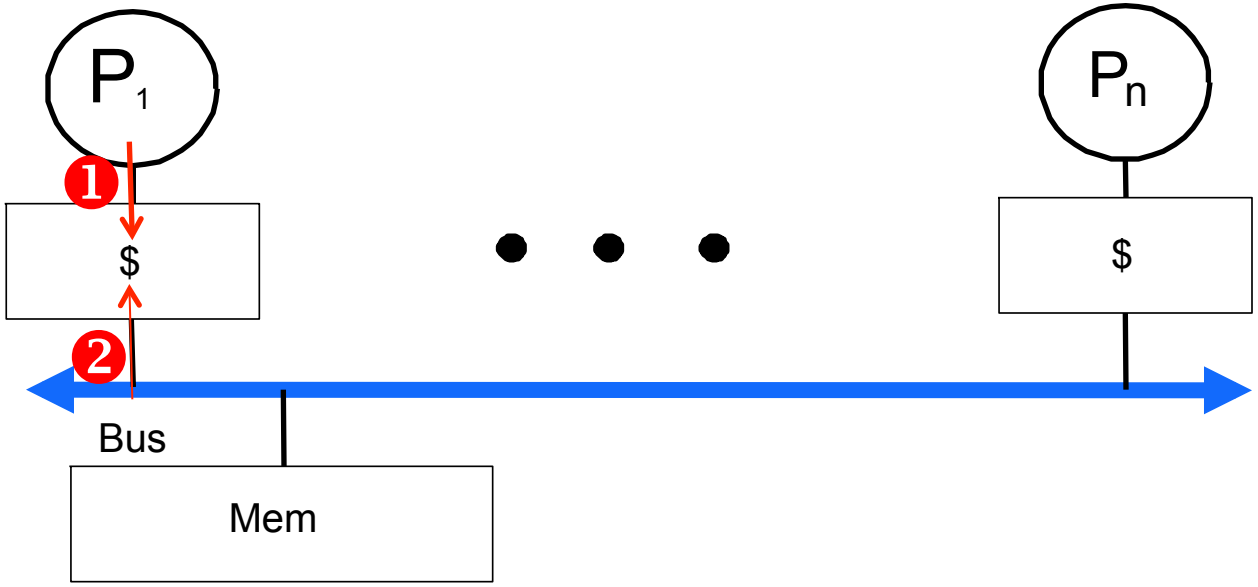
1. **Single-Writer, Multiple-Reader (SWMR) Invariant**



2. **Data-Value Invariant.** The value of the memory location at the start of an epoch is the same as the value of the memory location at the end of its last read-write epoch.

Coherence States

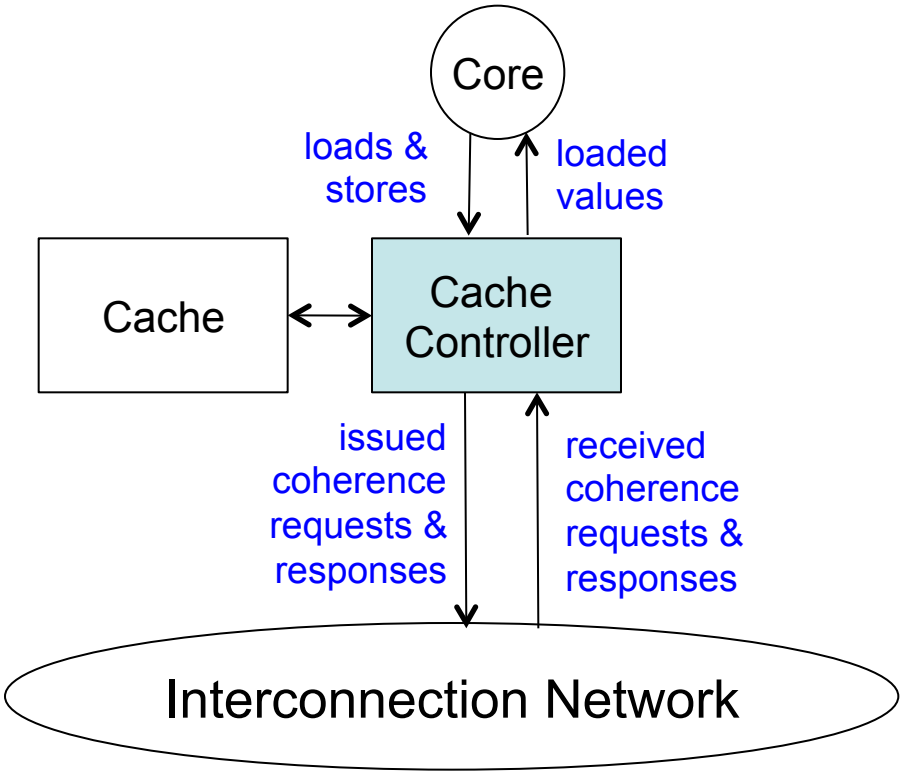
- How to design system satisfying invariants?
- Track “state” of memory block copies and ensure states changes satisfy invariants.
- Typical states: “modified”, “shared”, “invalid”.
- Mechanism for updating block state called a coherence protocol.



Cache can be accessed by CPU (❶) or Bus (❷).

We will consider effect of each starting with CPU.

Implementing Cache Coherence



Requests received from both core and network:

Processor request:

- receive load/store from core
- update state of block
- issue coherence request
- receive response
- update state of block
- reply to core

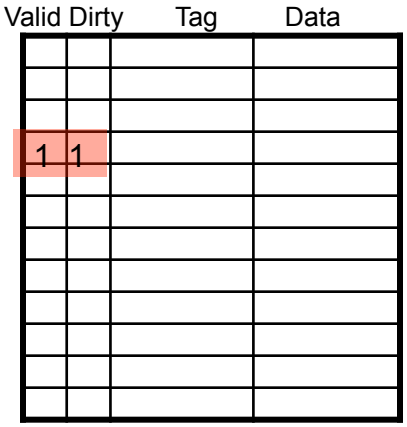
Network request:

- receive coherence request
- update state of block
- issue coherence response

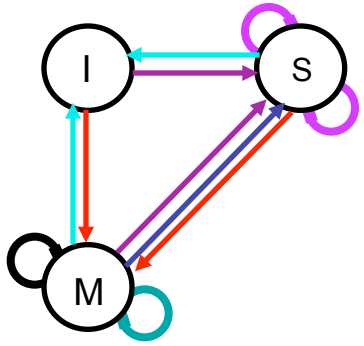
Coherence, Simplified

- Initially, assume all actions triggered by a memory request take place atomically. Only consider “stable” coherence states.
- This view ignores time for messages to be sent and received.
- In practice add “transient” states to enable multiple memory requests to occur in parallel.

Coherence Protocol States



Direct mapped,
write-back,
write allocate



Valid	Dirty	State
0	X	Invalid
1	0	Shared
1	1	Modified

- Logically, each memory block has a state.
- Typically 2 to 5 states for each block.

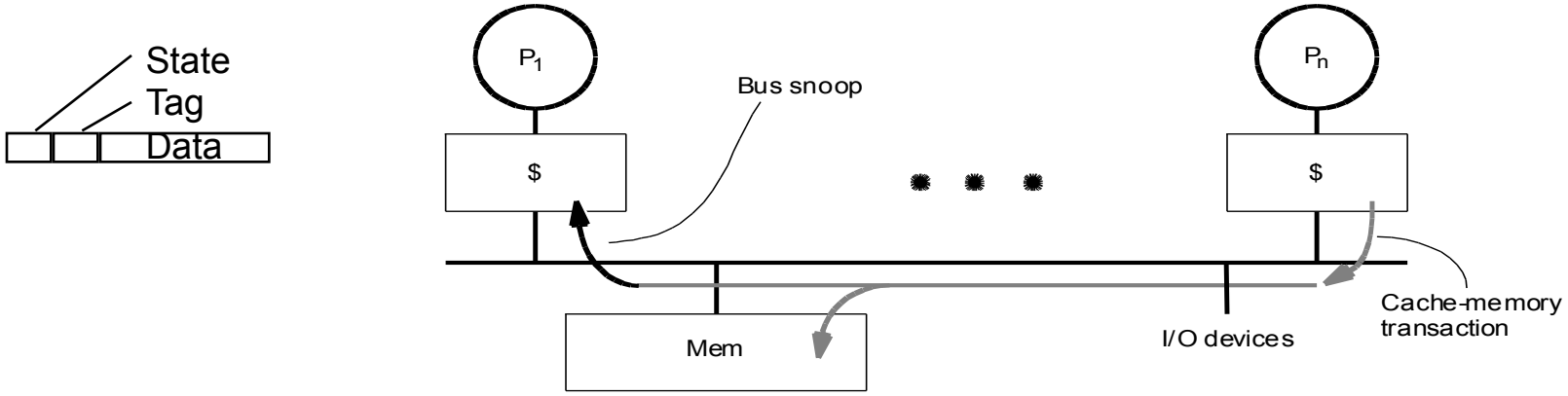
Directory vs. Snoop Protocols

- Directory Coherence Protocol
  - Summarize state of block in all caches in single location called a directory.
  - Advantages: Scalability
  - Disadvantage: Latency, Complexity

- Snoop Coherence
  - For each action ask all caches for state of block
  - Advantages: Fast and simple
  - Disadvantage: Not scalable

- Stable states at L1 cache same for both

Snooping Cache-Coherence Protocols



- Cache Controller "snoops" all transactions on the shared medium (bus or switch)
  - Relevant transaction if for a block it contains
  - Take action to ensure coherence
    - invalidate or supply value
  - Action depends on state of the block and the protocol
- Get exclusive access before write via write invalidate

MSI Snoop Protocol

- Each memory block is in one state (state is implicit)
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Modified)
  - OR Not in any caches

- Each cache block is in one state (state is explicit):
  - Shared : block can be read
  - OR Modified: cache has only copy, its writeable, and dirty
  - OR Invalid : block contains stale data or never accessed

- Read misses: cause all caches to snoop bus in case they have a modified copy of data (memory may be out of date)

- Write hits to “shared” blocks: treated as misses since we need to tell other processor caches about the write so they can invalidate any shared copy of the data.

Writes need to know whether any other copies of the block is cached

Write hit:

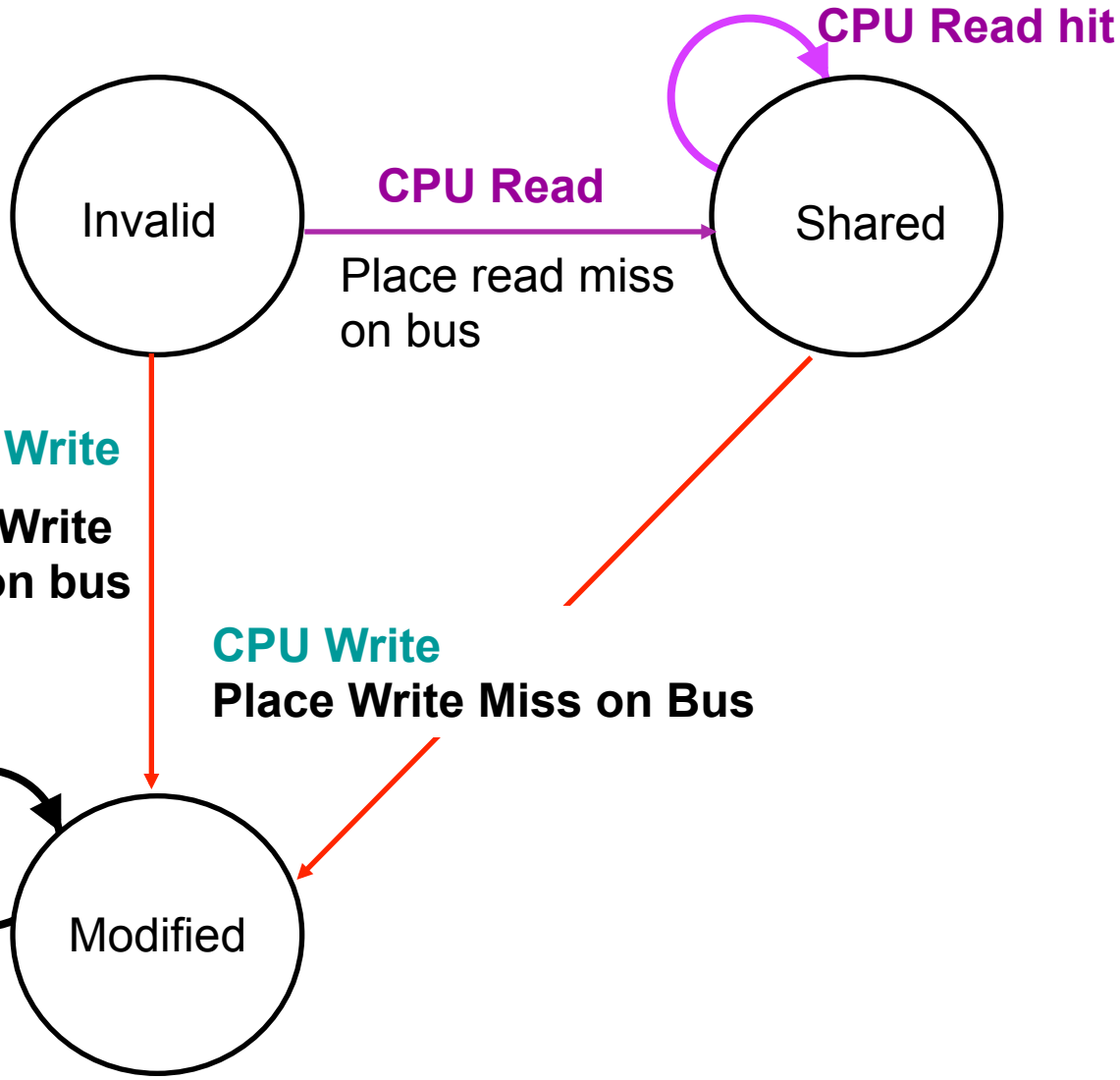
- Block in modified state ⇒ No other copies ⇒ No need to place invalidate on bus for writeback cache
- Block in shared state ⇒ may be copies ⇒ Need to place invalidate on bus (will cause other processors to invalidate their copy)

Write miss:

- Block in invalid state ⇒ write miss ⇒ Need to read block from memory (as in uniprocessor case) or another processor (if modified state in that other processor's cache).

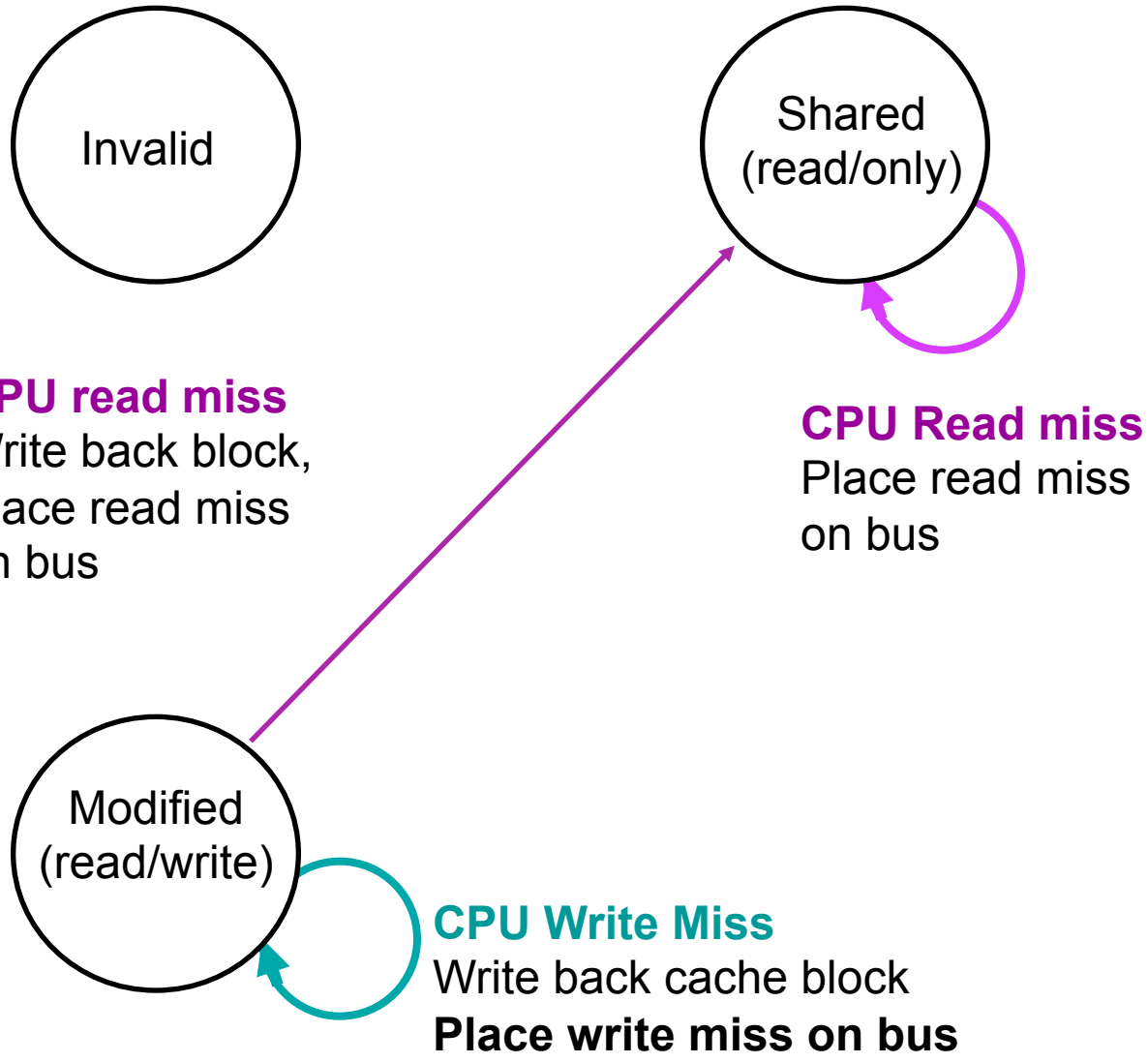
Write-Back State Machine – CPU

- State machine for *CPU* requests for each cache block
- Non-resident blocks invalid



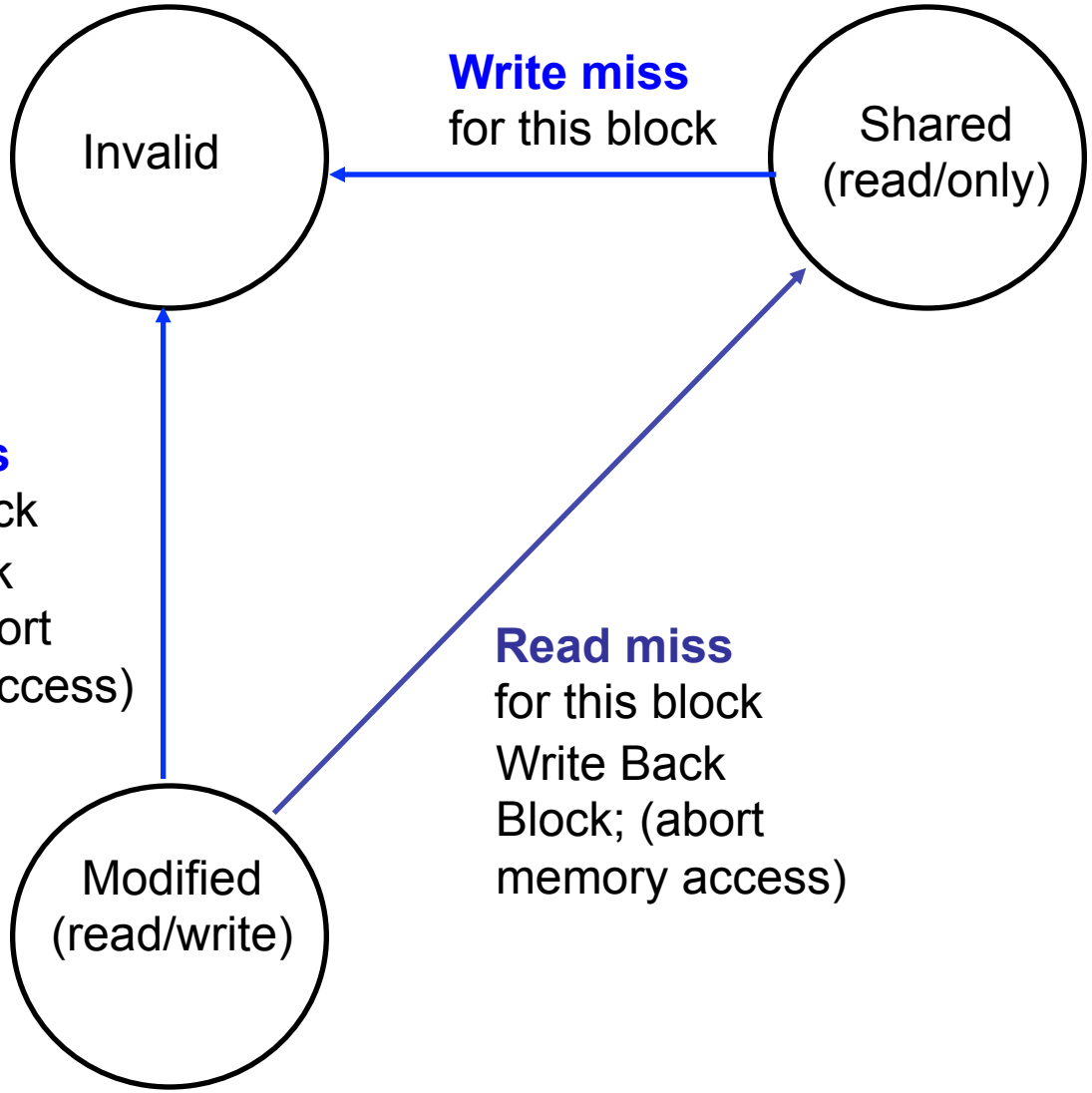
Block-replacement

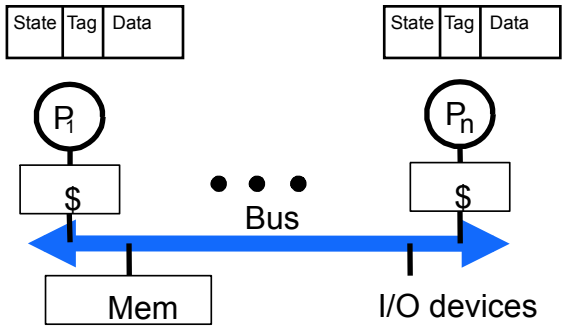
- State machine for *CPU* requests for each cache block



Write-Back State Machine- Bus request

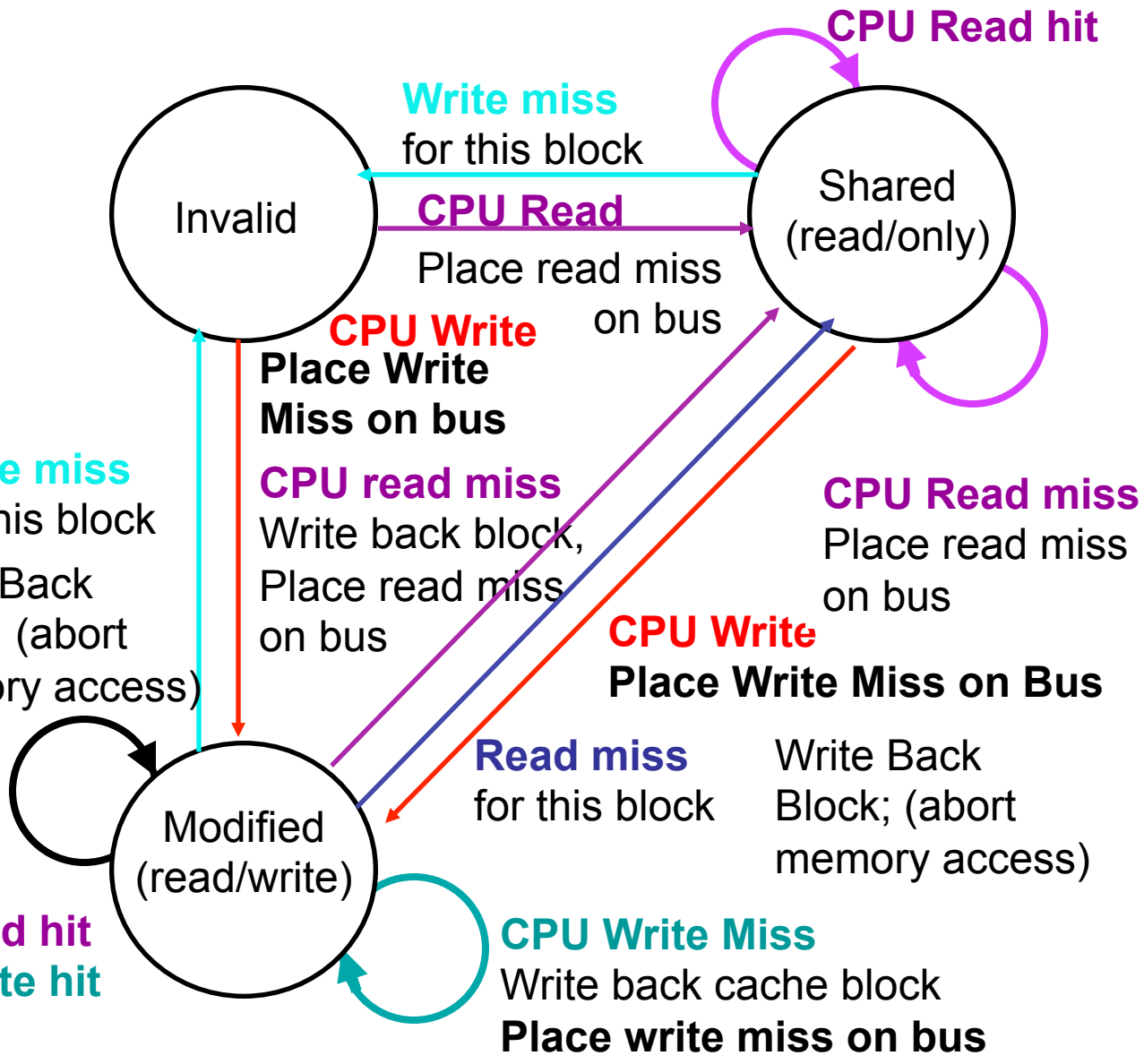
- State machine for *bus* requests for each cache block



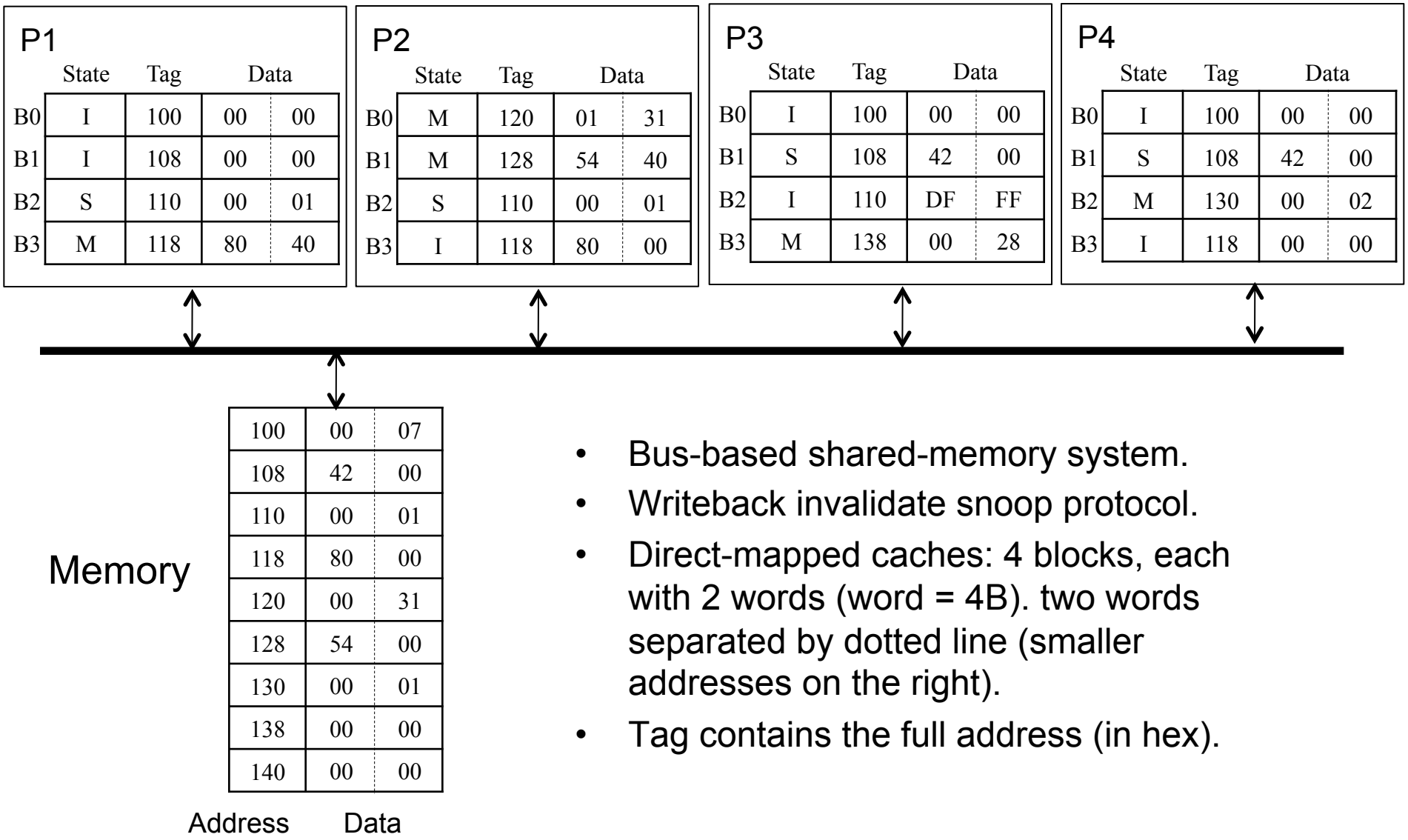


Write-back State Machine-III

- State machine for *CPU* requests for each cache block **and** for *bus* requests for each cache block



Example



- Bus-based shared-memory system.
- Writeback invalidate snoop protocol.
- Direct-mapped caches: 4 blocks, each with 2 words (word = 4B). two words separated by dotted line (smaller addresses on the right).
- Tag contains the full address (in hex).