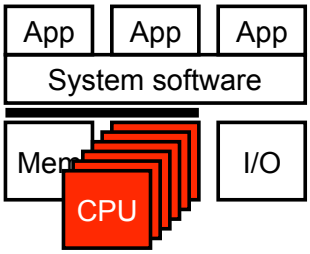
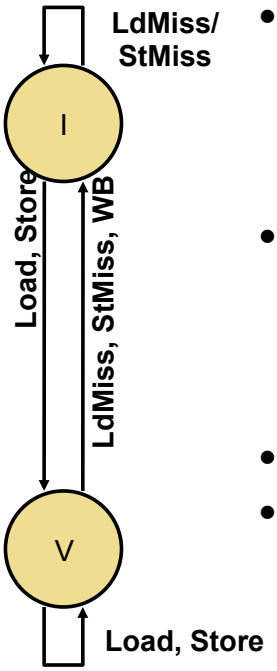
Roadmap Checkpoint



- Multicore vs. Hardware multihreading
- Cache coherence
- Memory consistency models

Review: VI Coherence Protocol



- **VI (valid-invalid) protocol**:
  - Two states (per block in cache)
    - **V (valid)**: own the block
    - **I (invalid)**: don't own block

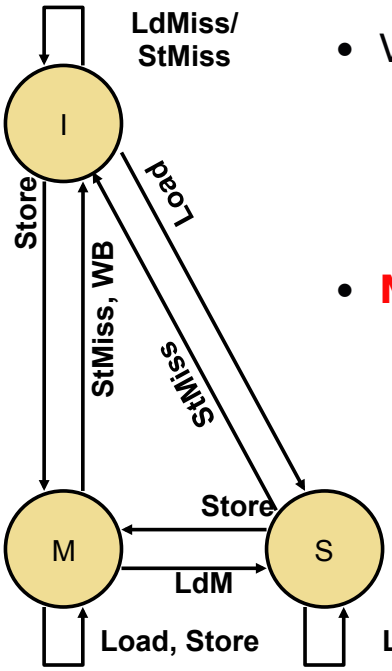
    + Can implement with "valid bit"
- Protocol state transition (the left figure)
  - Summary
    - If anyone wants to read/write block
    - Give it up: transition to **I** state
    - Write-back if your own copy is dirty
- This is an **invalidate protocol**
- VI protocol is inefficient
  - Only one cached copy allowed in entire system
  - Multiple copies can't exist even if read-only
    - Not a problem in example
    - Big problem in reality

Review: VI Protocol State Transition Table

State	*This Processor* Load	Store	*Other Processor* Load Miss	Store Miss
Invalid (I)	Load Miss ➔ V	Store Miss ➔ V	---	---
Valid (V)	Hit	Hit	Send Data ➔ I	Send Data ➔ I

- Rows are "states"
  - I vs V
- Columns are "events"
- Writeback events not shown
- Memory controller not shown
- **Memory sends data when no processor responds**

Review: MSI summary



- VI protocol is inefficient
  - Only one cached copy allowed in entire system
  - Multiple copies can't exist even if read-only
    - Not a problem in example
    - Big problem in reality
- **MSI (modified-shared-invalid)**
  - Fixes problem: splits "V" state into two states
    - **M (modified)**: local dirty copy
    - **S (shared)**: local clean copy
  - Allows **either**
    - Multiple read-only copies (S-state) **--OR--**
    - Single read/write copy (M-state)

Review: MSI Protocol State Transition Table

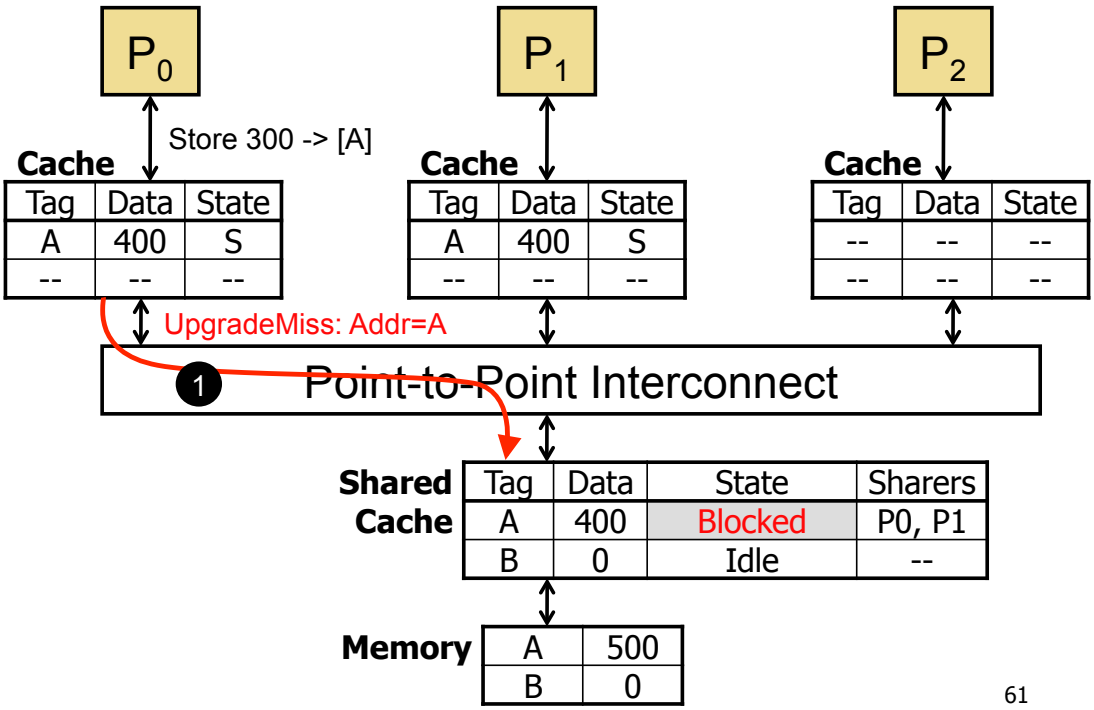
State	*This Processor* Load	Store	*Other Processor* Load Miss	Store Miss
Invalid (I)	Load Miss ➔ S	Store Miss ➔ M	---	---
Shared (S)	Hit	Upgrade Miss ➔ M	---	➔ I
Modified (M)	Hit	Hit	Send Data ➔ S	Send Data ➔ I

- M ➔ S transition also updates memory
- After which memory willl respond (as all processors will be in S)

Classifying Misses: 3C Model

- Divide cache misses into three categories
  - **Compulsory (cold)**: never seen this address before
    - **Would miss even in infinite cache**
  - **Capacity**: miss caused because cache is too small
    - **Would miss even in fully associative cache**
    - Identify? Consecutive accesses to block separated by access to at least N other distinct blocks (N is number of frames in cache)
  - **Conflict**: miss caused because cache associativity is too low
    - Identify? **All other misses**
- One more case of cache misses
  - **(COHERENCE): MISS DUE TO EXTERNAL INVALIDATIONS**
    - **ONLY IN SHARED MEMORY MULTIPROCESSORS**

MSI Coherence Example: Step #7



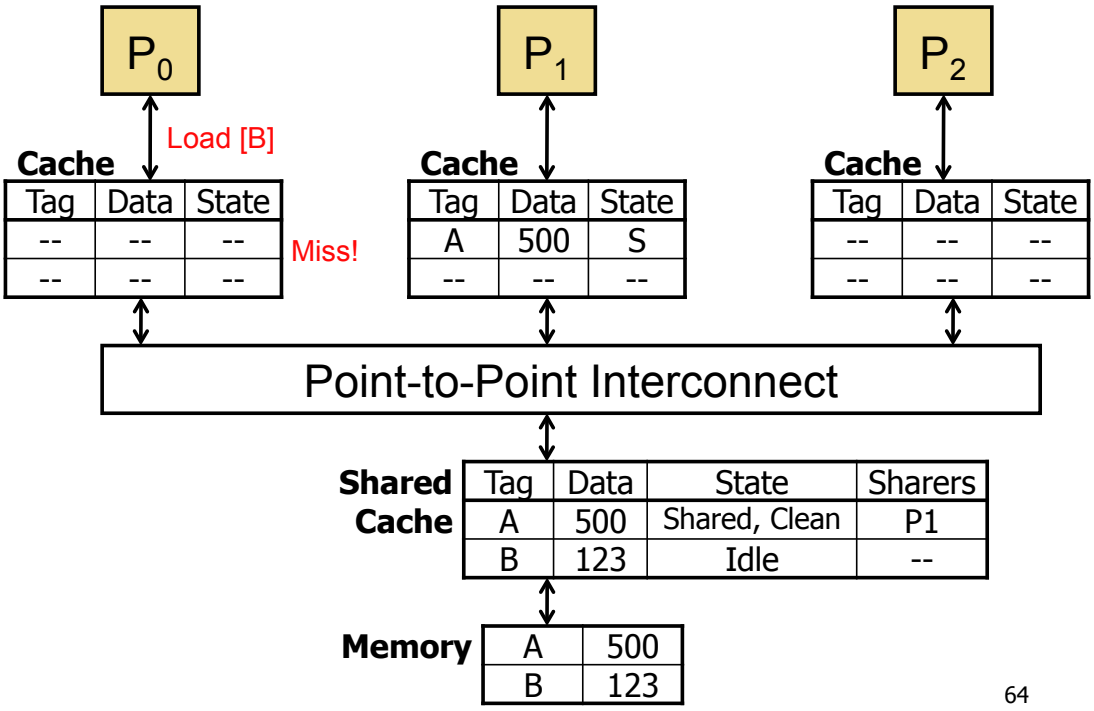
MESI Cache Coherence

- Ok, we have read-only and read/write with MSI
- But consider load & then store of a block by same core
  - Under coherence as described, **this would be two misses: "Load miss" plus an "upgrade miss"…**
  - … even if the block isn't shared!
  - Consider programs with 99% (or 100%) private data
    - Potentially doubling number of misses (bad)
- Solution:
  - Most modern protocols also include **E (exclusive)** state
  - Interpretation: "I have the only cached copy, and it's a **clean** copy"
    - Has read/write permissions
    - Just like "Modified" but "clean" instead of "dirty".

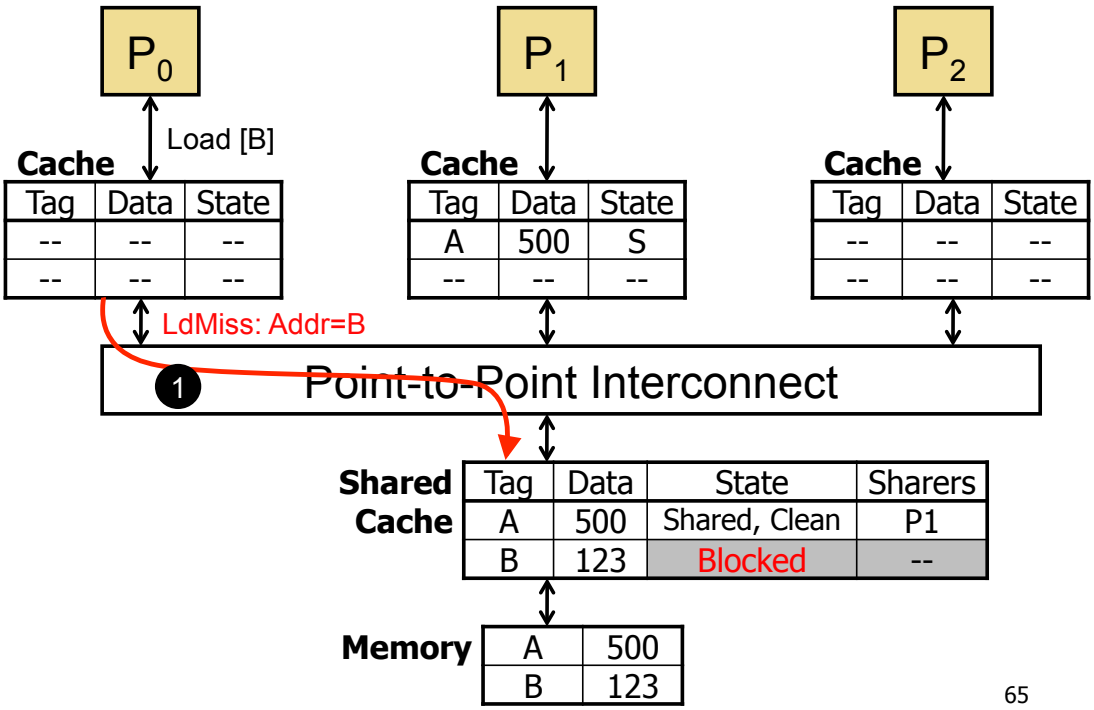
MESI Operation

- Goals:
  - Avoid "upgrade" misses for non-shared blocks
  - While not increasing eviction (aka writeback or replacement) traffic
- Two cases on a load miss to a block…
  - **Case #1**: … with no current sharers
    (that is, no sharers in the set of sharers)
    - Grant requester "Exclusive" copy with read/write permission
  - **Case #2**: … with other sharers
    - As before, grant just a "Shared" copy with read-only permission
- A store to a block in "Exclusive" changes it to "Modified"
  - **Instantaneously & silently** (no latency or traffic)
- On block eviction (aka writeback or replacement)…
  - If "Modified", block is dirty, must be written back to next level
  - If "Exclusive", writing back the data is not necessary
    (but notification may or may not be, depending on the system)

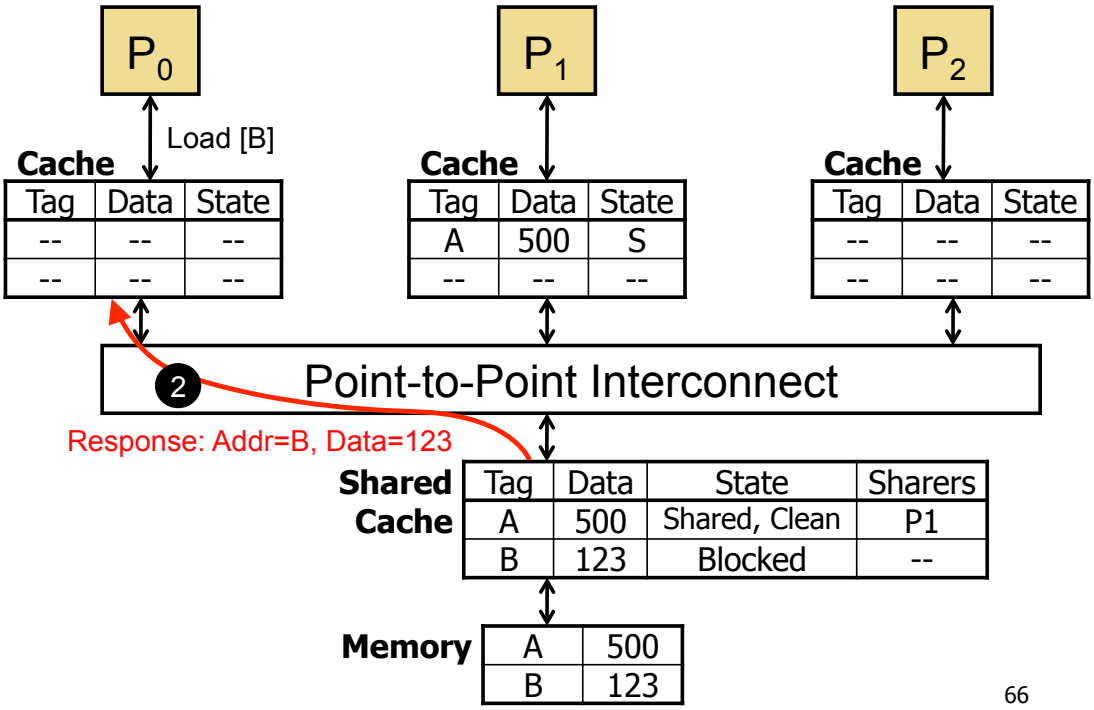
MESI Coherence Example: Step #1



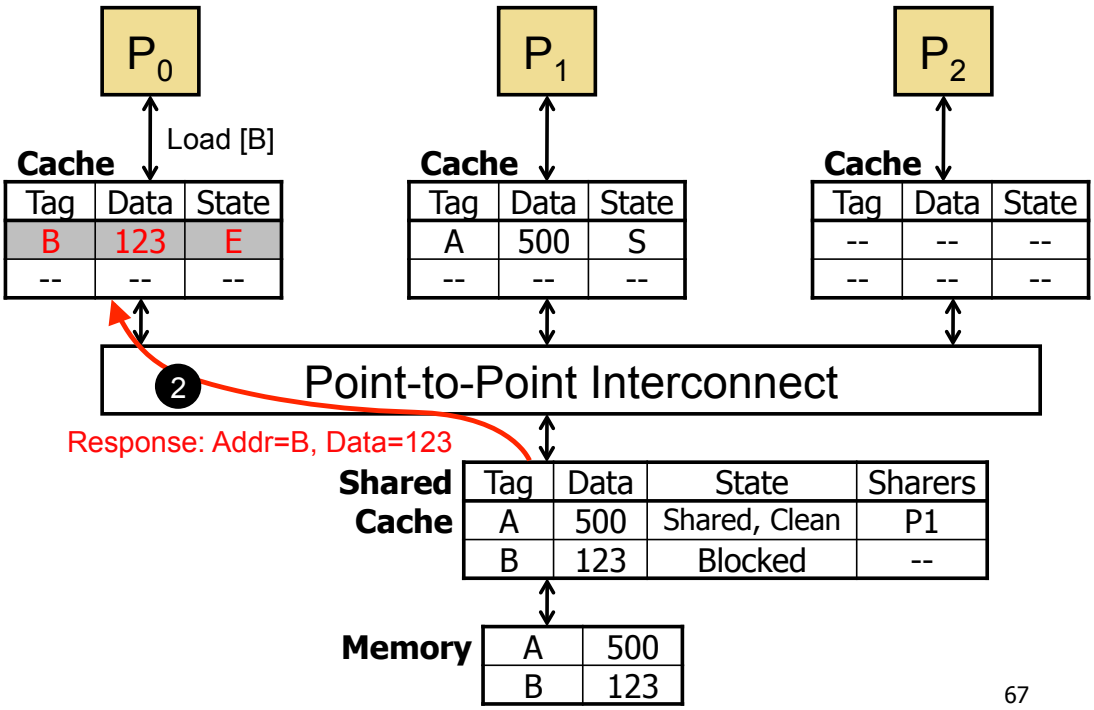
MESI Coherence Example: Step #2



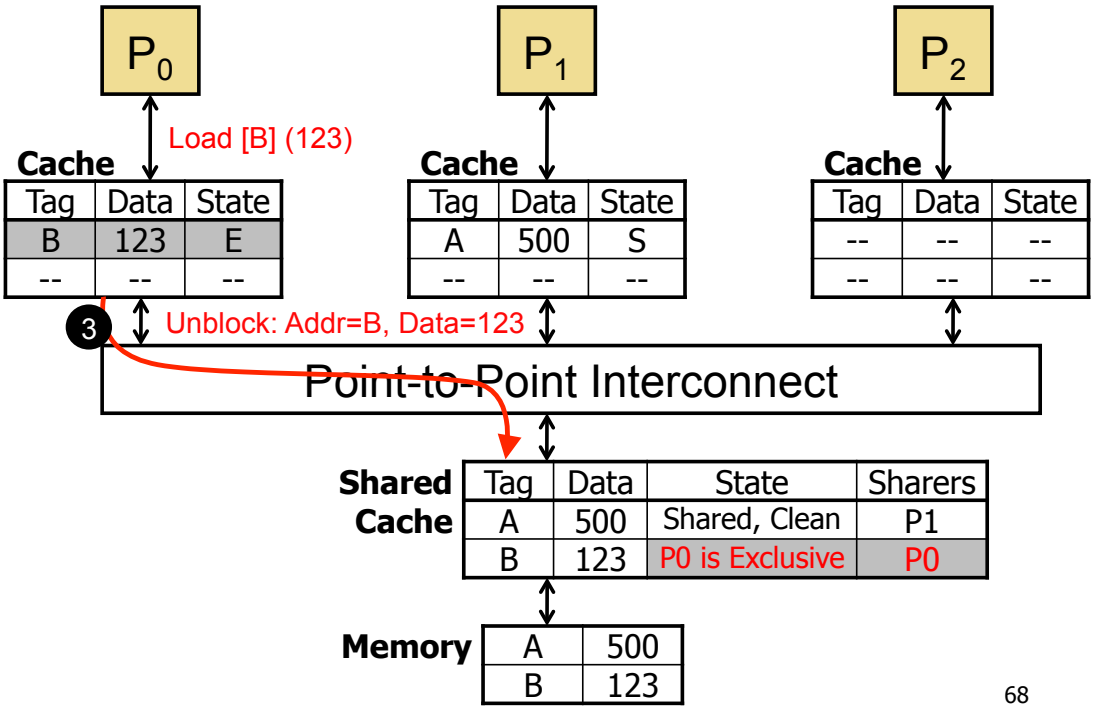
MESI Coherence Example: Step #3



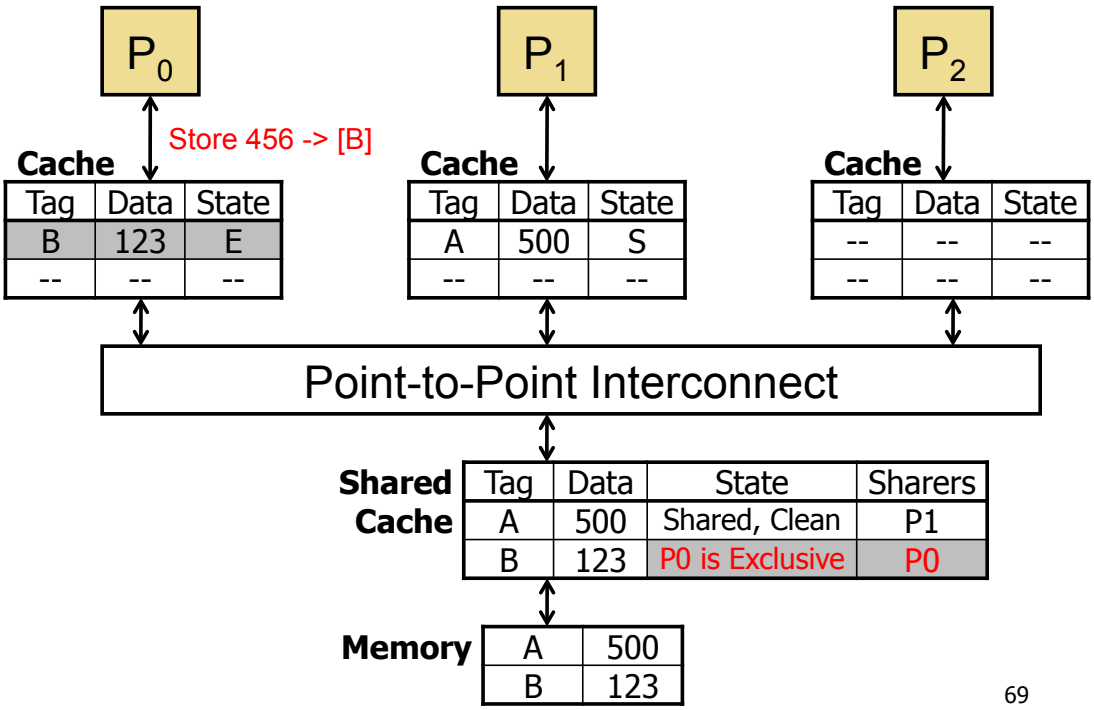
MESI Coherence Example: Step #4



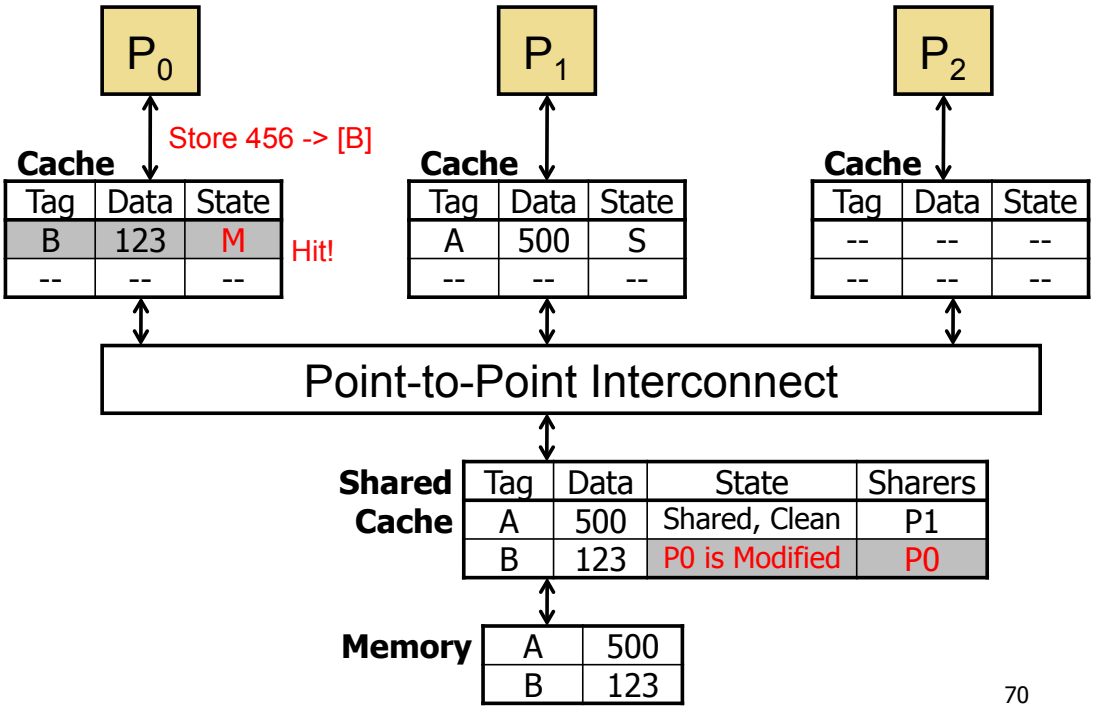
MESI Coherence Example: Step #5



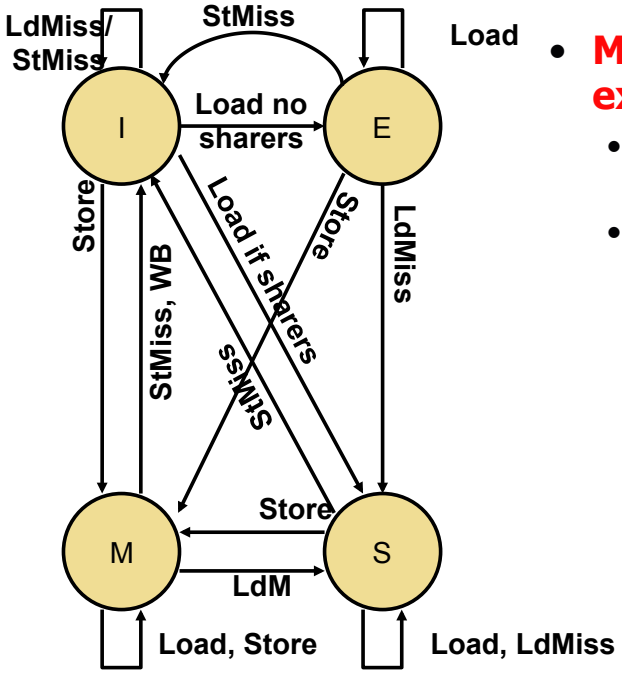
MESI Coherence Example: Step #6



MESI Coherence Example: Step #7



MSI → MESI



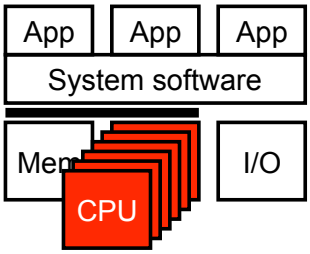
- **MESI (modified-exclusive-shared-invalid)**
  - Eliminates the cost of coherence when there's no sharing
  - Keeps single-threaded programs fast on multicores

MESI Protocol State Transition Table

State	*This Processor* Load	*This Processor* Store	*Other Processor* Load Miss	*Other Processor* Store Miss
Invalid (I)	Miss → **S or E**	Miss → M	---	---
Shared (S)	Hit	Upg Miss → M	---	→ I
Exclusive (E)	Hit	Hit → M	Send Data → S	Send Data → I
Modified (M)	Hit	Hit	Send Data → S	Send Data → I

- Load misses lead to "E" if no other processors is caching the block

Roadmap Checkpoint



- Hardware multihreading
- Cache coherence
- Memory consistency models

Memory consistency

Memory Consistency

- **Cache coherence**
  - Creates globally uniform (consistent) view of memory
  - Of **a single memory location** (in other words: cache blocks)
  - Not enough
    - Some optimizations skip coherence

- **Memory consistency model**
  - Specifies the semantics of shared memory operations
  - i.e., what value(s) a load may return

- Who cares? Programmers
  - Globally inconsistent memory creates mystifying behavior

3 Classes of Memory Consistency Models

- **Sequential consistency (SC)** (MIPS, PA-RISC)
  - **Typically what programmers expect**
  - 1. Processors see their own loads and stores in program order
  - 2. Processors see others' loads and stores in program order
  - 3. All processors see same global load/store ordering
  - Corresponds to some sequential interleaving of uniprocessor orders
  - **Indistinguishable from multi-programmed uni-processor**
- **Total Store Order (PC)** (**x86**, SPARC)
  - Allows an in-order (FIFO) store buffer
    - Stores can be deferred, but must be put into the cache in order
- **Release consistency (RC)** (**ARM**, Itanium, **PowerPC**)
  - Allows an un-ordered coalescing store buffer
    - Stores can be put into cache in any order
  - Loads re-ordered, too.

Memory operation ordering

- A program defines a sequence of loads and stores (this is the "program order" of the loads and stores)

- Four types of memory operation orderings
  - W→R: write must complete before subsequent read *
  - R→R: read must complete before subsequent read
  - R→W: read must complete before subsequent write
  - W→W: write must complete before subsequent write

* To clarify: "write must complete before subsequent read" means:
When a write comes before a read in program order, the write must complete (its results are visible) when the read occurs.

Sequential consistency

- A sequentially consistent memory system maintains all four memory operation orderings
- A parallel system is sequentially consistent if the result of any parallel execution is the same as if all the memory operations were executed in some sequential order, and the memory operations of any one processor are executed in program order.

Quick example

Thread 1 (on P1)	Thread 2 (on P2)
`A = 1;`	`B = 1;`
`if (B == 0)`	`if (A == 0)`
`  print "Hello";`	`  print "World";`

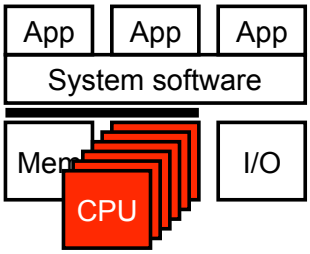
- Assume:
  - A and B are initialized to 0, writes propagate immediately
  - Sequential consistency
- **Question: Imagine threads 1 and 2 are being run simultaneously on a two core system. What will get printed?**

Quick example

Thread 1 (on P1)	Thread 2 (on P2)
`A = 1;`	`B = 1;`
`if (B == 0)`	`if (A == 0)`
`  print "Hello";`	`  print "World";`

- Sequential consistency =>
  - P1: the write to A has to complete before the read of B can begin
  - P2: the write to B has to complete before the read of A can begin
- Answer: Code will either print "hello" or "world" or nothing, but not both.

Summary: Multicore



- Hardware multihreading
- Cache coherence
- Memory consistency models