5DV118
Computer Organization and Architecture
Umeå University
Department of Computing Science

Stephen J. Hegner

Topic 5: The Memory Hierarchy

Part C: Cache Coherence in Multicores

These slides are mostly taken verbatim, or with minor changes, from those prepared by

Mary Jane Irwin (www.cse.psu.edu/~mji)

of The Pennsylvania State University

[Adapted from *Computer Organization and Design, 4th Edition*,

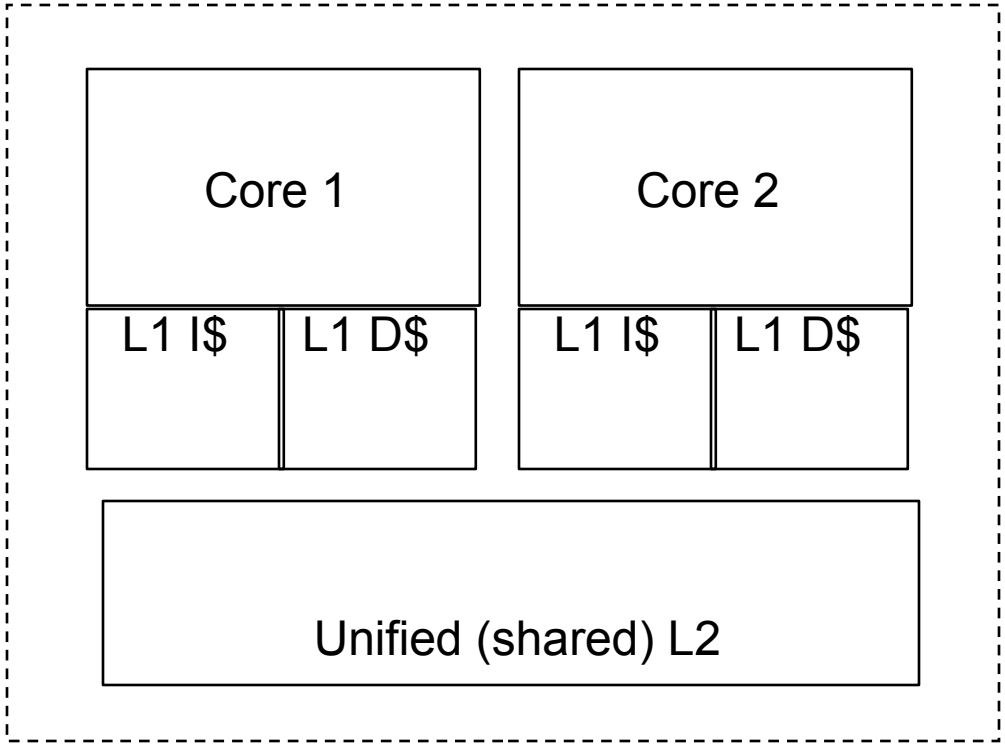
Patterson & Hennessy, © 2008, MK]

Key to the Slides

- The source of each slide is coded in the footer on the right side:
  - Irwin CSE331 = slide by Mary Jane Irwin from the course CSE331 (Computer Organization and Design) at Pennsylvania State University.
  - Irwin CSE431 = slide by Mary Jane Irwin from the course CSE431 (Computer Architecture) at Pennsylvania State University.
  - Hegner UU = slide by Stephen J. Hegner at Umeå University.

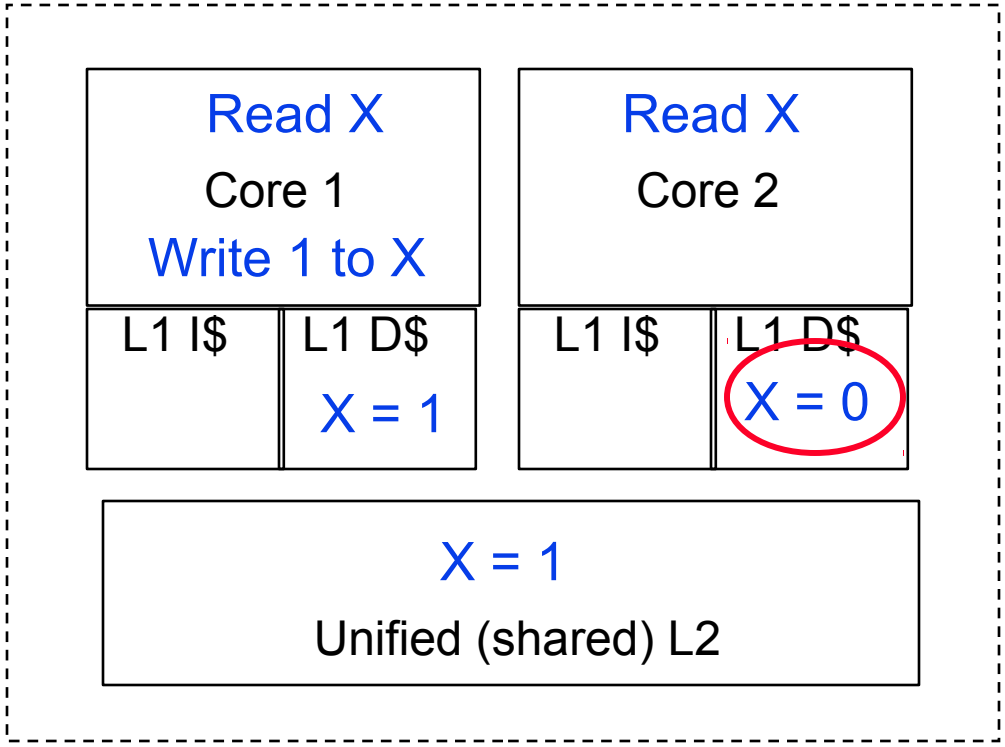
Cache Coherence in Multicores

- In future multicore processors its likely that the cores will *share* a common physical address space, causing a cache coherence problem



Cache Coherence in Multicores

- In future multicore processors its likely that the cores will *share* a common physical address space, causing a cache coherence problem



A Coherent Memory System

- Any read of a data item should return the most recently written value of the data item
  - Coherence – defines what values can be returned by a read
    - Writes to the same location are serialized (two writes to the same location must be seen in the same order by all cores)
  - Consistency – determines when a written value will be returned by a read

- To enforce coherence, caches must provide
  - Replication of shared data items in multiple cores' caches
    - Replication reduces both latency and contention for a read shared data item
  - Migration of shared data items to a core's local cache
    - Migration reduced the latency of the access the data and the bandwidth demand on the shared memory (L2 in our example)

Coherence and Consistency Clarified

- The consistency of data in multiple caches is defined according to a consistency model.
- In the caches of processors, the sequential or serialized model of consistency is typically used.
  - In this model, writes must be seen in the same order by all processes, regardless of which caches they access.
- A cache-coherence protocol is an algorithm for maintaining the consistency of a multi-cache system according to a specified consistency model.
  - Cache coherence thus refers to the process of maintaining consistency (according to a specific consistency model) of the data which are distributed amongst the various caches.
- Bottom line: It is necessary to start with a consistency model and then define a cache-coherence protocol with respect to that model.

Cache Coherence Protocols

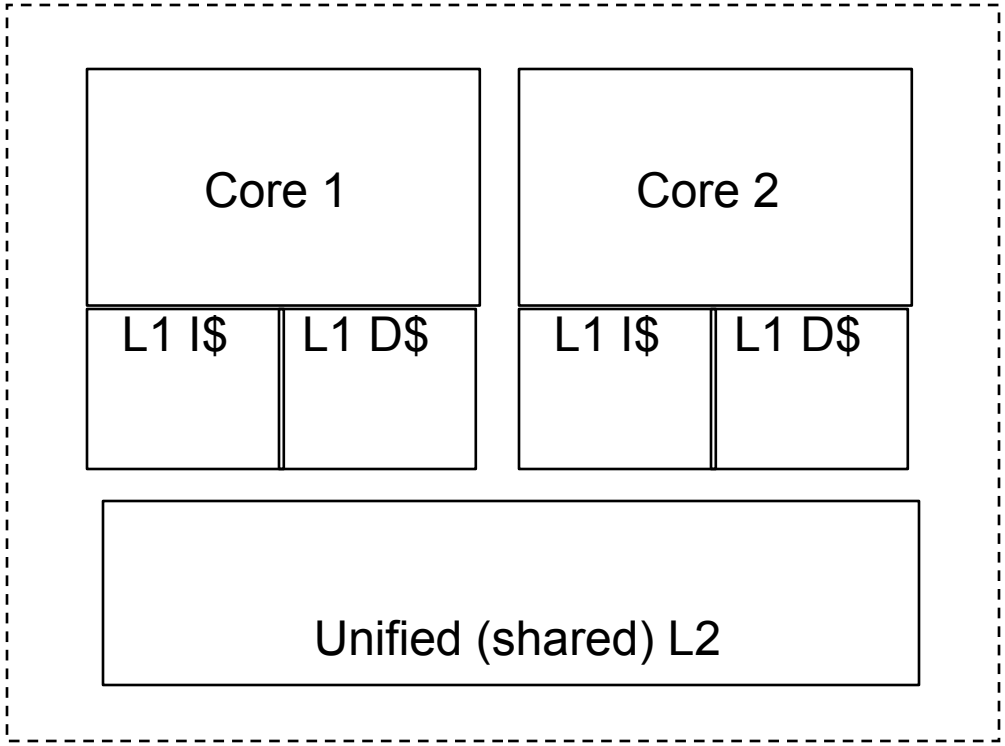
- Need a hardware protocol to ensure cache coherence the most popular of which is snooping
  - The cache controllers monitor (snoop) on the broadcast medium (e.g., bus) with duplicate address tag hardware (so they don't interfere with core's access to the cache) to determine if their cache has a copy of a block that is requested
- Write invalidate protocol – writes require exclusive access and invalidate *all* other copies
  - Exclusive access ensure that no other readable or writable copies of an item exists
- If two processors attempt to write the same data at the same time, one of them wins the race causing the other core's copy to be invalidated. For the other core to complete, it must obtain a new copy of the data which must now contain the updated value – thus enforcing write serialization

Handling Writes

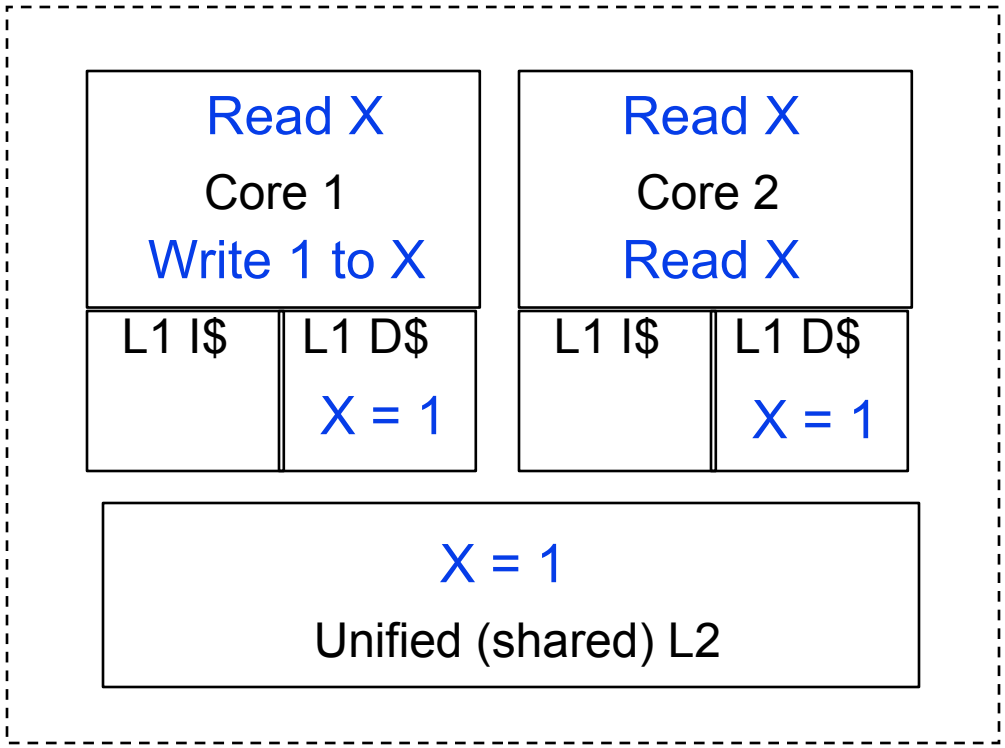
Ensuring that all other processors sharing data are informed of writes can be handled two ways:

1. Write-update (write-broadcast) – writing processor broadcasts new data over the bus, all copies are updated
   - All writes go to the bus → higher bus traffic
   - Since new values appear in caches sooner, can reduce latency
2. Write-invalidate – writing processor issues invalidation signal on bus, cache snoops check to see if they have a copy of the data, if so they invalidate their cache block containing the word (this allows multiple readers but only one writer)
   - Uses the bus only on the first write → lower bus traffic, so better use of bus bandwidth

Example of Snooping Invalidation

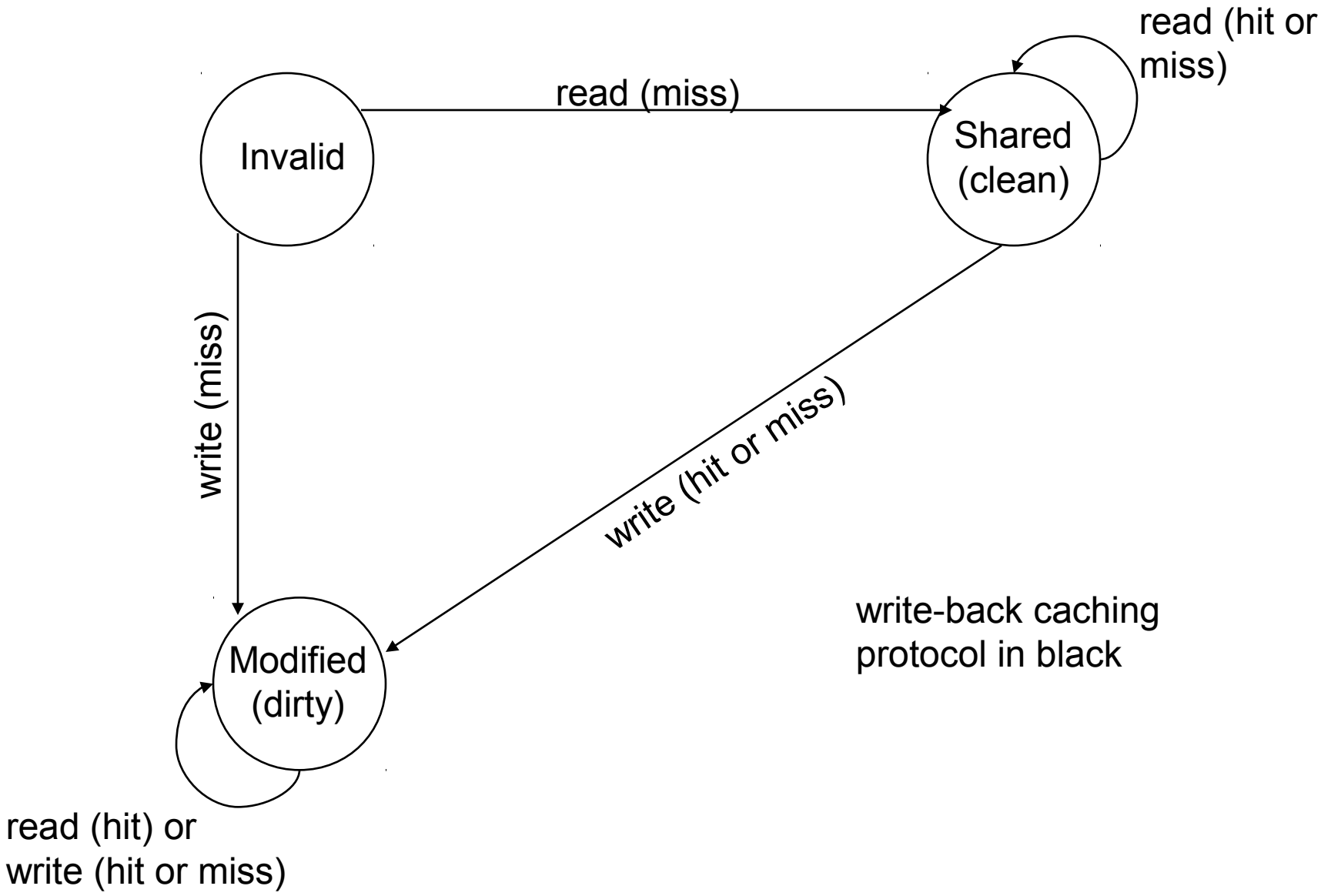


Example of Snooping Invalidation

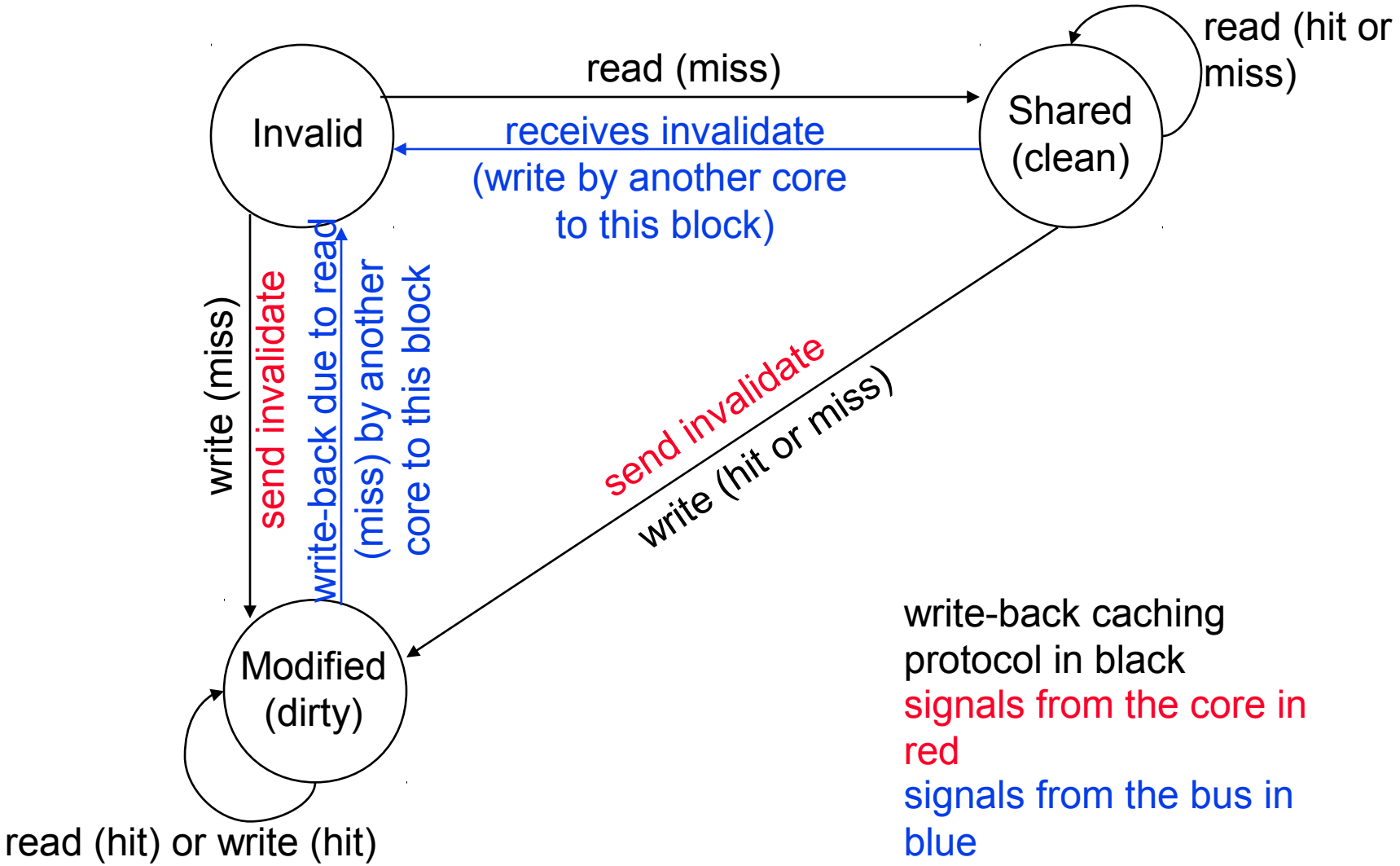


- When the second miss by Core 2 occurs, Core 1 responds with the value canceling the response from the L2 cache (and also updating the L2 copy)

A Write-Invalidate CC Protocol

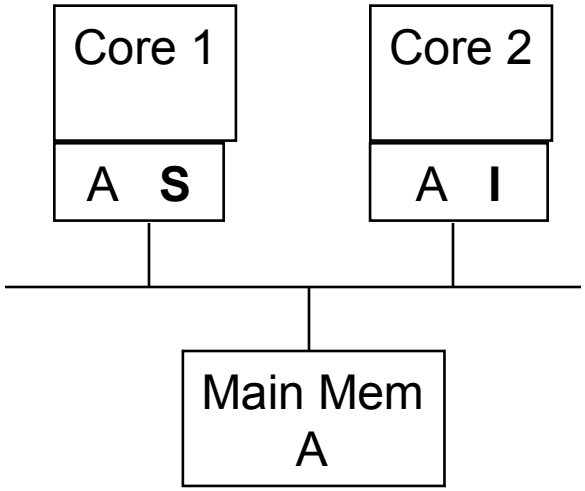


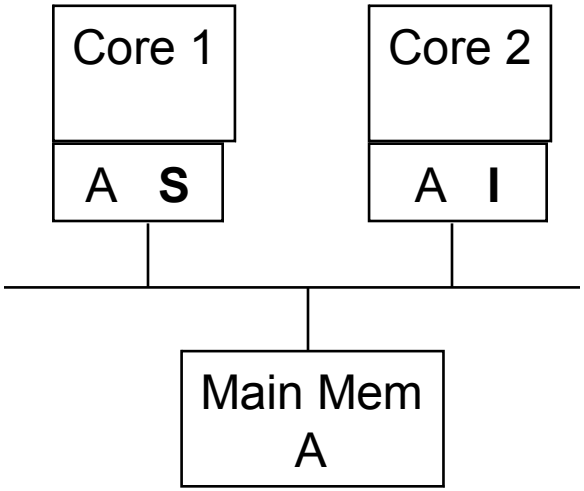
A Write-Invalidate CC Protocol

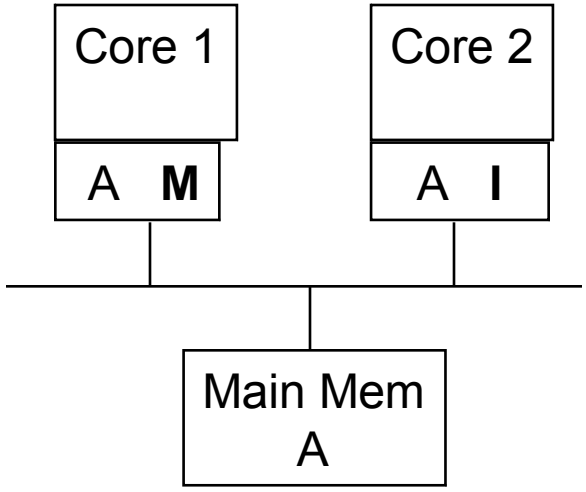


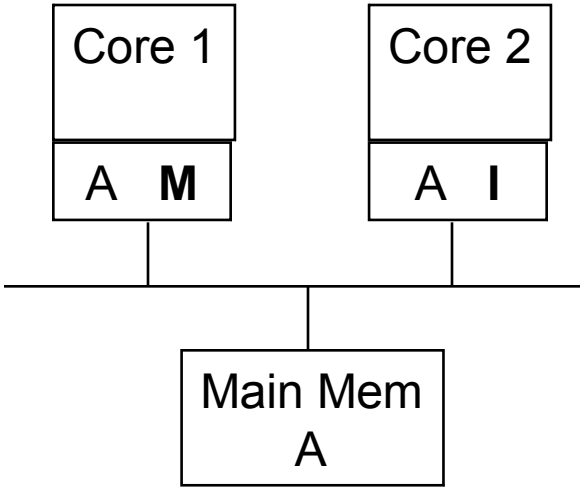
Write-Invalidate CC Examples

- I = invalid (many), S = shared (many), M = modified (only one)



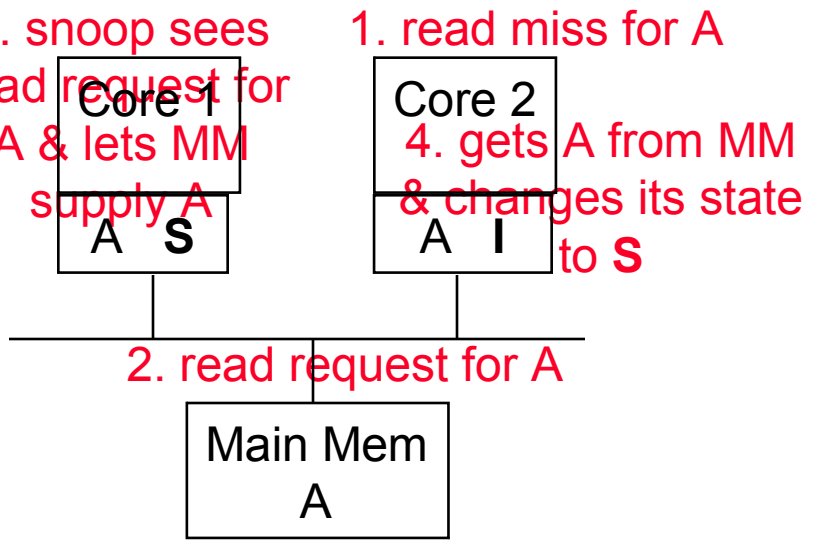


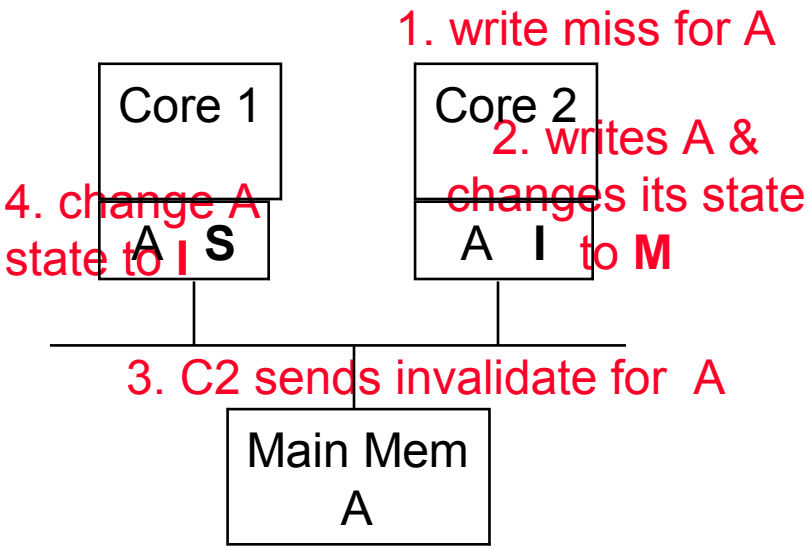


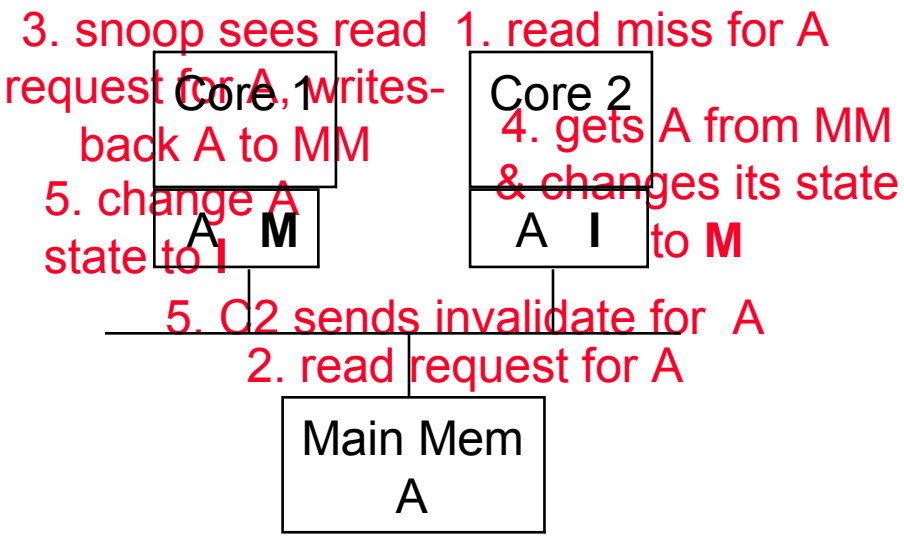


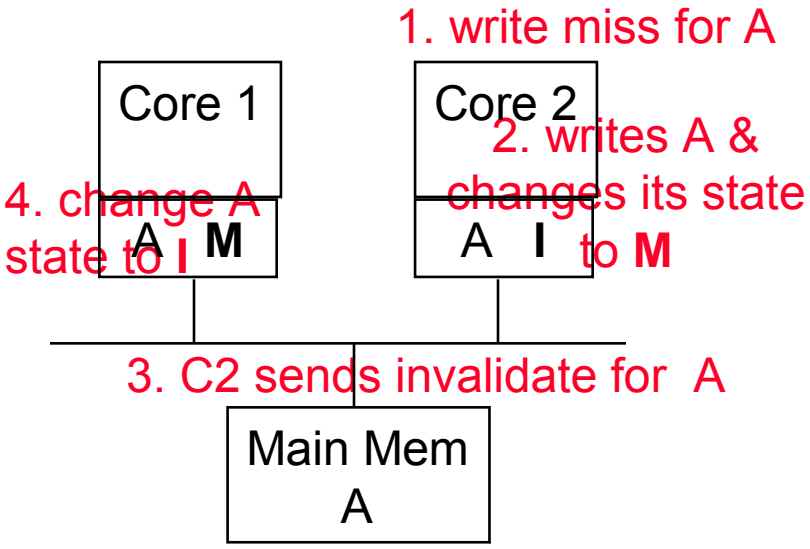
Write-Invalidate CC Examples

- I = invalid (many), S = shared (many), M = modified (only one)







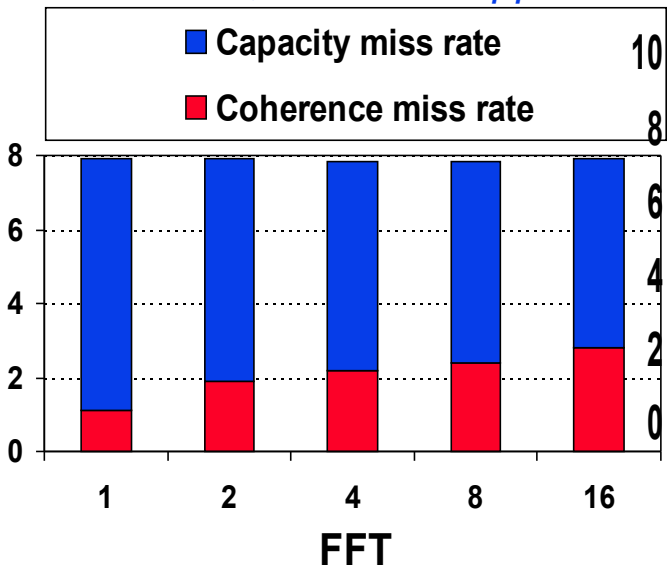


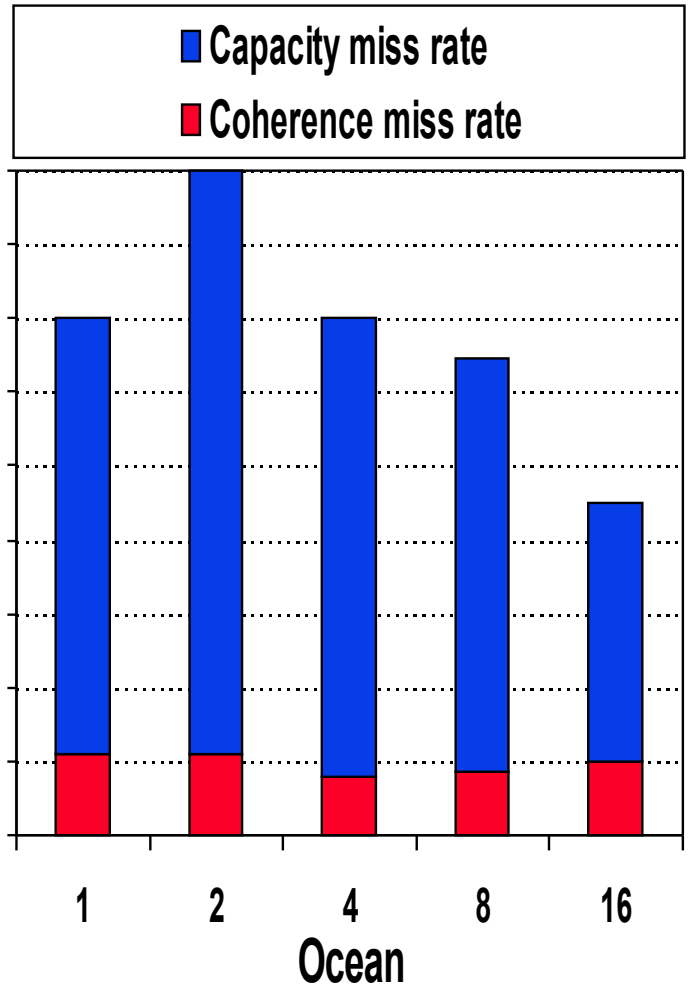
Data Miss Rates

- Shared data has lower spatial and temporal locality
  - Share data misses often dominate cache behavior even though they may only be 10% to 40% of the data accesses

64KB 2-way set associative data cache with 32B blocks

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach*

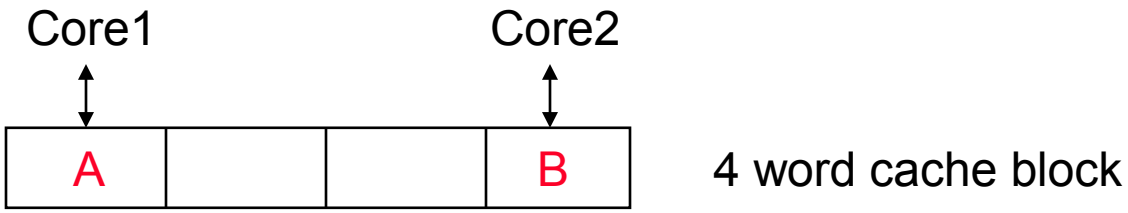




Block Size Effects

- Writes to one word in a multi-word block mean that the full block is invalidated

- Multi-word blocks can also result in false sharing: when two cores are writing to two different variables that happen to fall in the same cache block
  - With write-invalidate false sharing increases cache miss rates



- Compilers can help reduce false sharing by allocating highly correlated data to the same cache block

Other Coherence Protocols

- There are many variations on cache coherence protocols

- Another write-invalidate protocol used in the Pentium 4 (and many other processors) is MESI with four states:
  - Modified – same
  - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
    - Since there is only one copy of the block, write hits don't need to send invalidate signal
  - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
  - Invalid – same

MESI Cache Coherency Protocol