Lecture 10

Cache Coherence

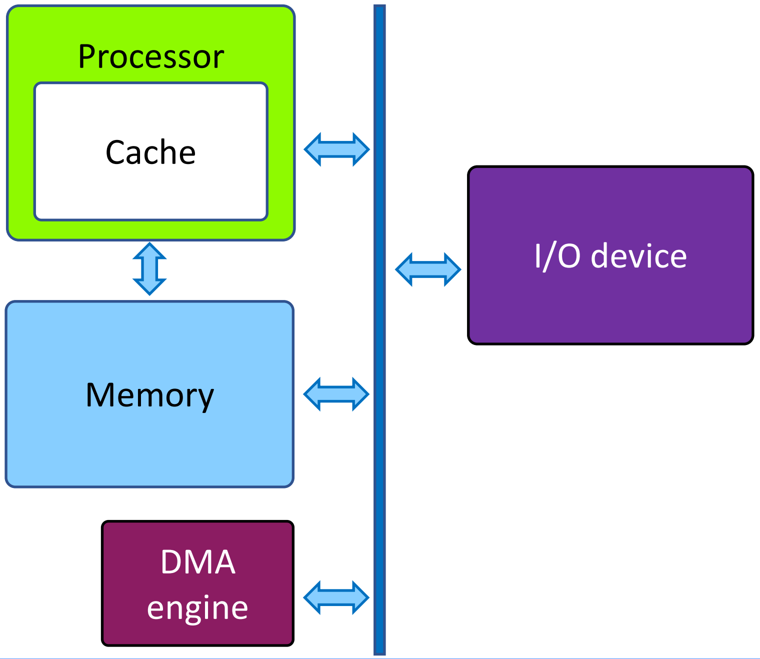
이재진
서울대학교 데이터사이언스대학원
서울대학교 공과대학 컴퓨터공학부
http://aces.snu.ac.kr/~jlee

Cache Coherence Problem

- Caching is vital to reducing memory latency in multiprocessor systems
- Private caches in a multiprocessor system may create a coherence problem
  - Copies of a variable can be present in multiple caches
- We expect that a read by any processor to return the most up-to-date value
- A write by one processor may not become visible to others
  - The result of the write are not observed by others
  - Stale values

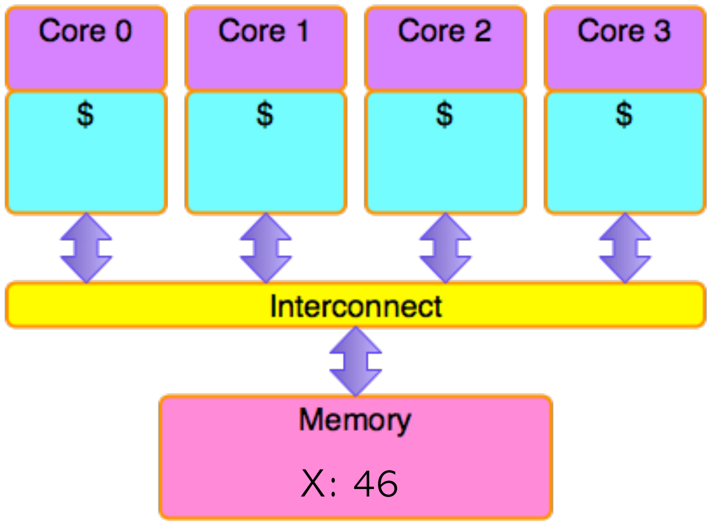
Cache Coherence Problem (cont'd)

- Easy in uniprocessors except I/O
- Coherence problems between I/O devices and the processor caches
  - Uncacheable memory region, uncacheable operations, flushing pages, passing I/O data through caches, etc.



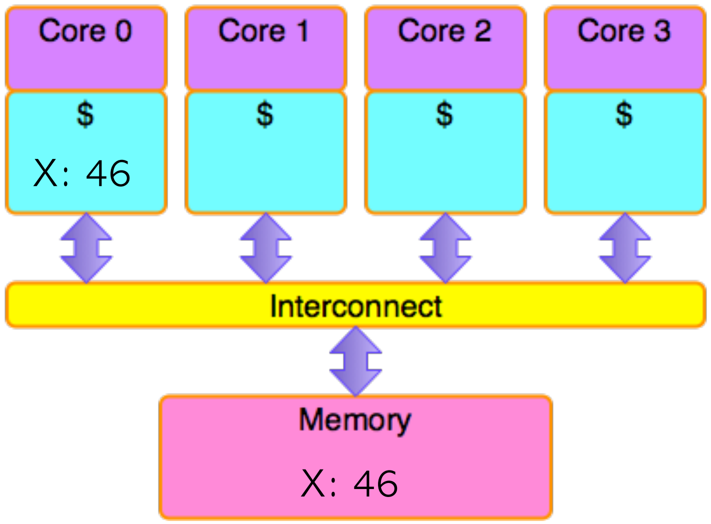
Cache Coherence Problem

- Assume write back, write allocate caches



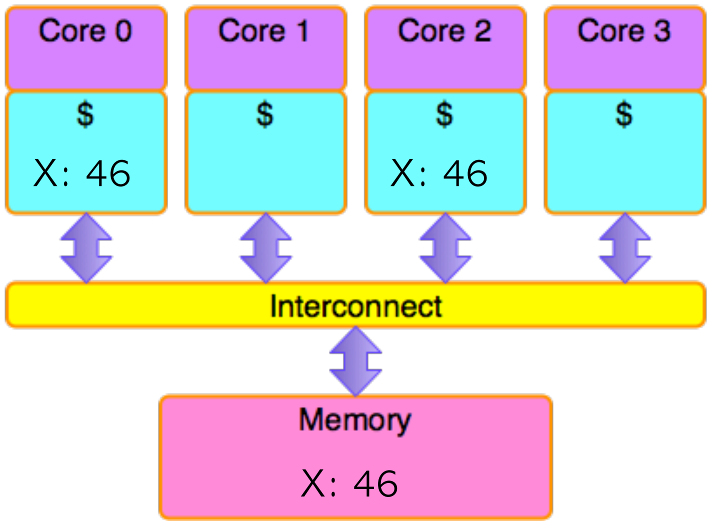
Cache Coherence Problem

- Assume write back, write allocate caches
- Core 0 reads location X



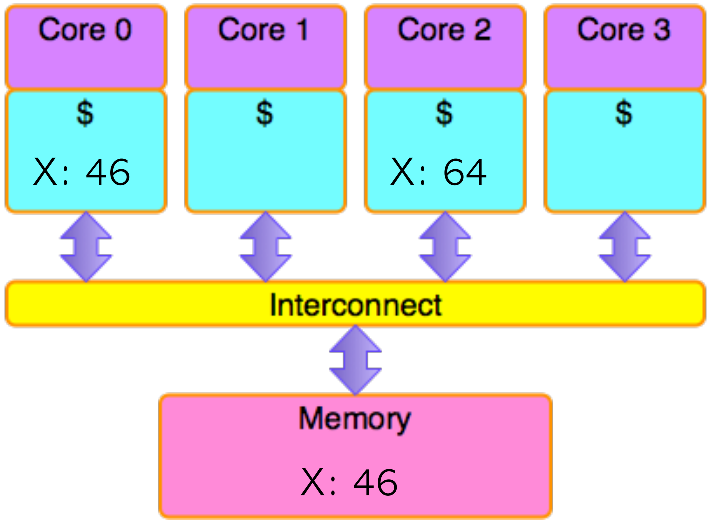
Cache Coherence Problem

- Assume write back, write allocate caches
- Core 2 reads location X



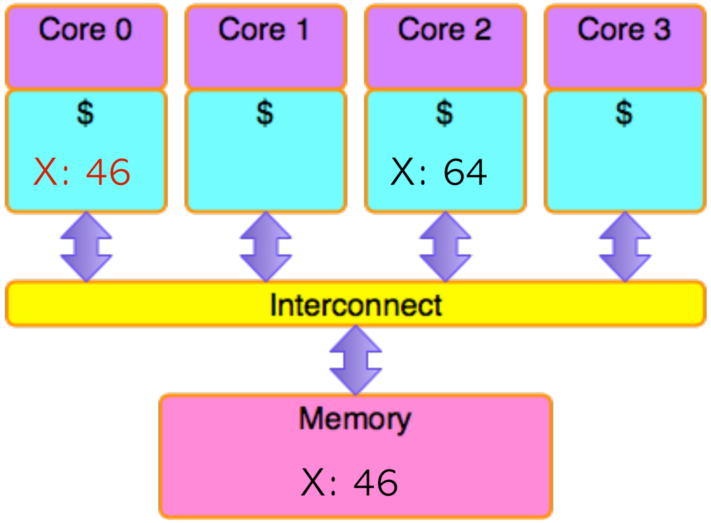
Cache Coherence Problem (cont’d)

- Core 2 writes location X



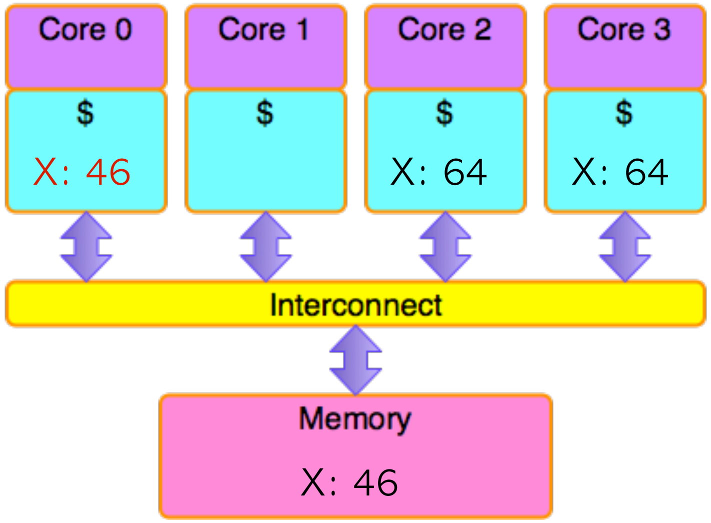
Cache Coherence Problem (cont’d)

- Core 0 has a stale copy of X
- Some action must be taken
  - Update or invalidate (more common)



Cache Coherence Problem (cont'd)

- Core 3 reads location X
- Core 2 supplies the updated copy of X to Core 3



Solutions to the Cache Coherence Problem

- Software based vs. hardware based
- Software-based:
  - Compiler or/and runtime
  - With or without hardware support
  - Perfect memory access information is needed
    - Aliasing and parallelism
- Hardware-based solutions are more common

Solutions to the Cache Coherence Problem (cont'd)

- Hardware cache coherence protocol ensures that requests for a certain data item always return the most recent value
  - Snoopy and directory-based protocols
  - MSI, MESI, MOESI, etc.
- Coherence misses
  - Misses due to invalidations when using an invalidation-based cache coherence protocol
- Shared caches do not have a coherence problem

Snooping

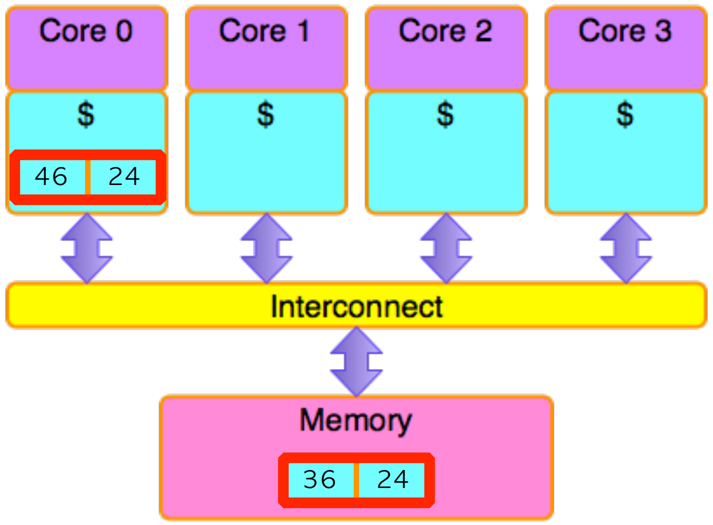
- Each processor snoops every address placed on the bus when a processor requests a read from memory
- If a processor has a dirty copy of the requested cache block, provide that cache block to the requestor and abort the memory access

Coherence and Write-through Caches

- All processor writes result in update of the local cache and a global bus write
  - Updates main memory
  - Needs to invalidate/update all other caches
- Simple
- Memory contains the most up-to-date value

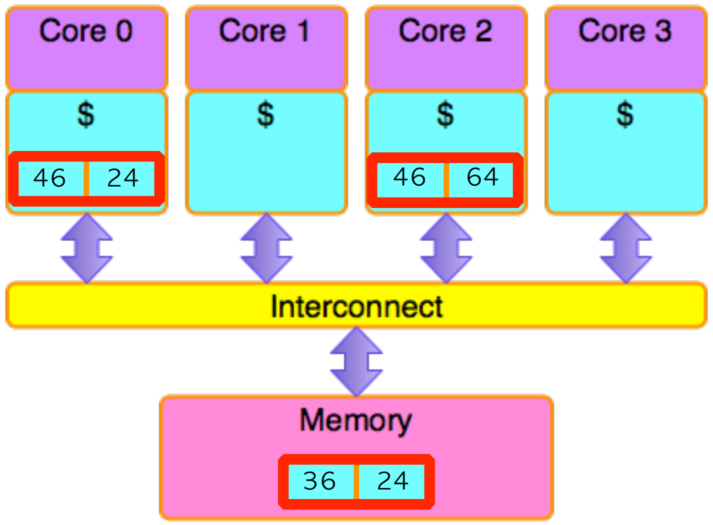
False Sharing

- Core 0 writes to the left half of a cache block



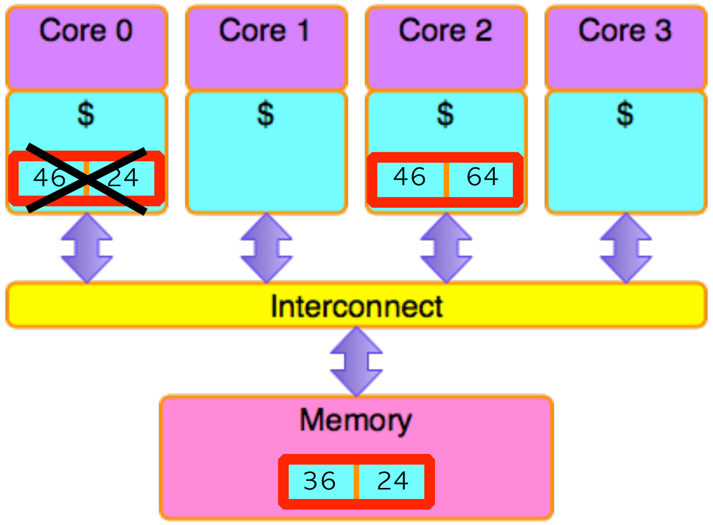
False Sharing (cont'd)

- Core 2 writes to the right half of the cache block



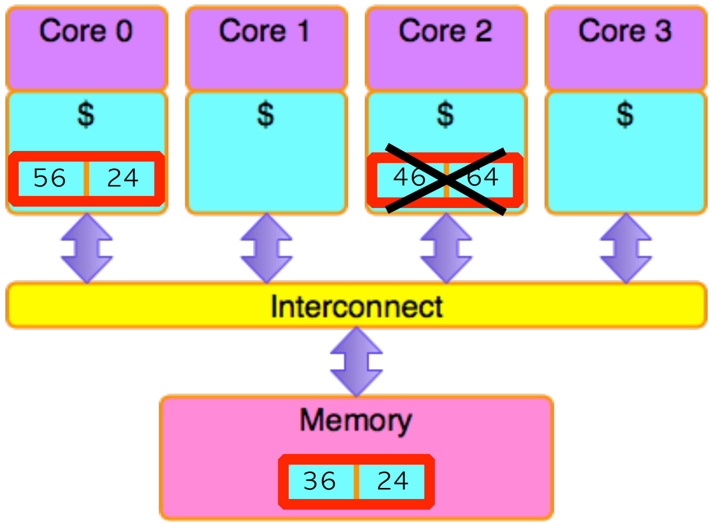
False Sharing (cont'd)

- The cache block of Core 0 is invalidated



False Sharing (cont'd)

- Core 0 writes to the left half of the cache block again



False Sharing (cont'd)

- Invalidations can lead to problems with false sharing
  - Different cores write to different locations in the same cache block repeatedly
  - Force the cache line to ping-pong back and forth between the two cores

- Not occur in update-based protocols