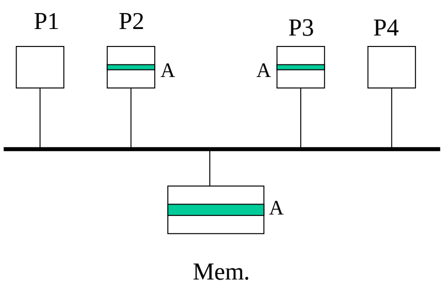
Cache Coherence (controllers snoop on bus transactions)

Initial state: P2 reads A; P3 reads A

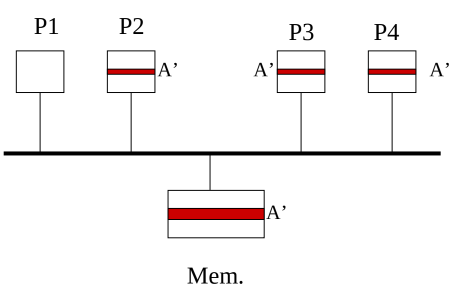


Cache coherence (cont'd)

- Now P2 wants to write A
- Two choices:
  - Broadcast the new value of A on the bus; value of A snooped by cache of P3: Write-update (or write broadcast) protocol (resembles write-through). Memory is also updated.
  - Broadcast an invalidation message with the address of A; the address snooped by cache of P3 which invalidates its copy of A: Write-invalidate protocols. Note that the copy in memory is not up-to-date any longer (resembles write-back)
- If instead of P2 wanting to write A, we had a write miss in P4 for A, the same two choices of protocol apply.

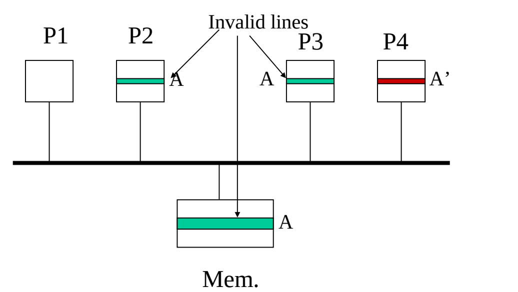
Write-update

P2 and P3 have read line A; P4 has a write miss on an element of line A



Write-invalidate

P2 and P3 have read line A; P4 has a write miss on an element of line A



Snoopy Cache Coherence Protocols

- Associate states with each cache block; for example:
  - Invalid
  - Clean (one or more copies are up to date)
  - Dirty (modified; exists in only one cache)
- Fourth state (and sometimes more) for performance purposes

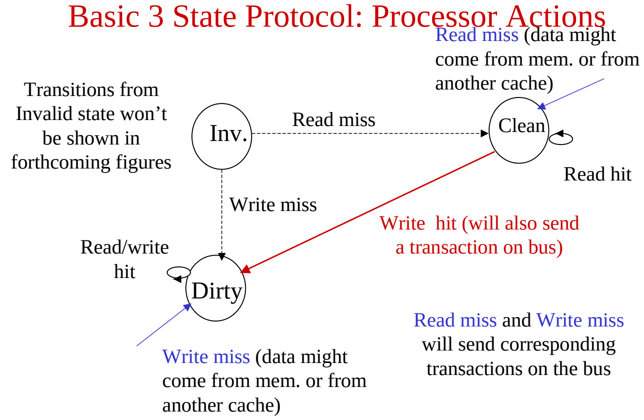
State Transitions for a Given Cache Block

- Those incurred as answers to processor associated with the cache
  - Read miss, write miss, write on clean block
- Those incurred by snooping on the bus as result of other processor actions, e.g.,
  - Read miss by Q might make P's block transit from dirty to clean
  - Write miss by Q might make P's block transit from dirty/clean to invalid (write invalidate protocol)

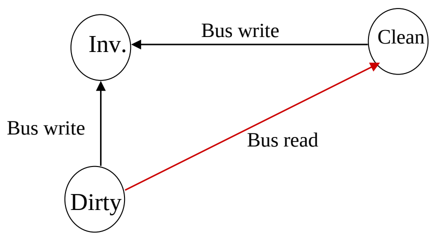
Basic Write-invalidate Protocol (write-back write-allocate caches)

- Needs 3 states associated with each cache block
  - Invalid
  - Clean (read only – can be shared) – also called Shared
  - Dirty (only valid copy in the system) – also called Modified
- Need to decompose state transitions into those:
  - Induced by the processor attached to the cache
  - Induced by snooping on the bus

Basic 3 State Protocol: Processor Actions



Basic 3 State Protocol: Transitions from Bus Snooping



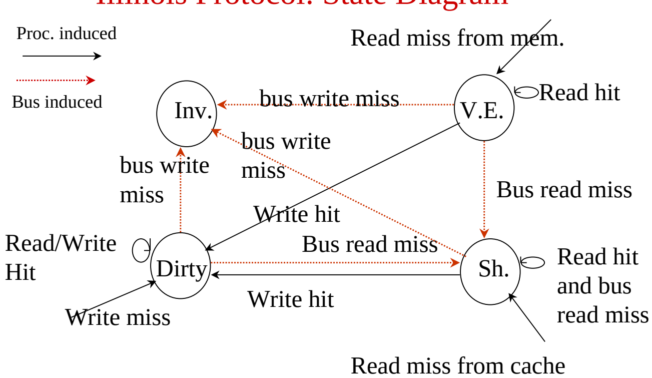
An Example of Write-invalidate Protocol: the Illinois Protocol

- States:
  - Invalid (aka Invalid)
  - Valid-Exclusive (clean, only copy, aka Exclusive)
  - Shared (clean, possibly other copies, aka Shared)
  - Dirty (modified, only copy, aka Modified)
  - In the MOESI notation, a MESI protocol
    - O stands for ownership

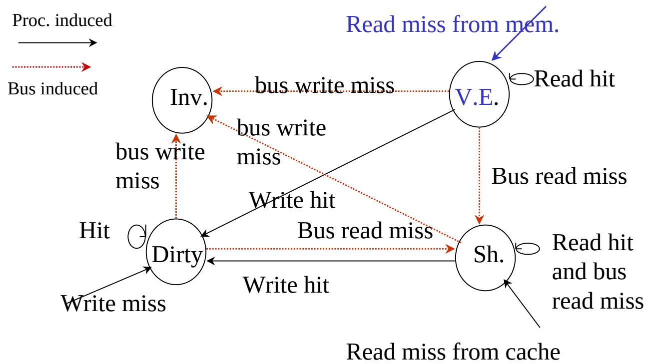
Illinois Protocol: Design Decisions

- The Valid-Exclusive state is there to enhance performance
  - On a write to a block in V-E state, no need to send an invalidation message (occurs often for private variables).
- On a read miss with no cache having the block in dirty state
  - Who sends the data: memory or cache (if any)?
    - Answer: cache for that particular protocol; other protocols might use the memory
  - If more than one cache, which one?
    - Answer: the first to grab the bus (tri-state devices)

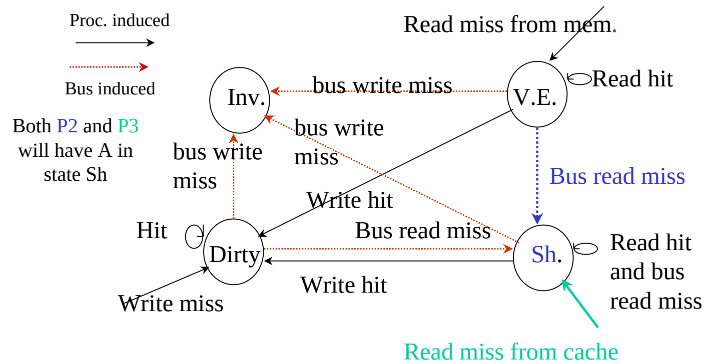
Illinois Protocol: State Diagram



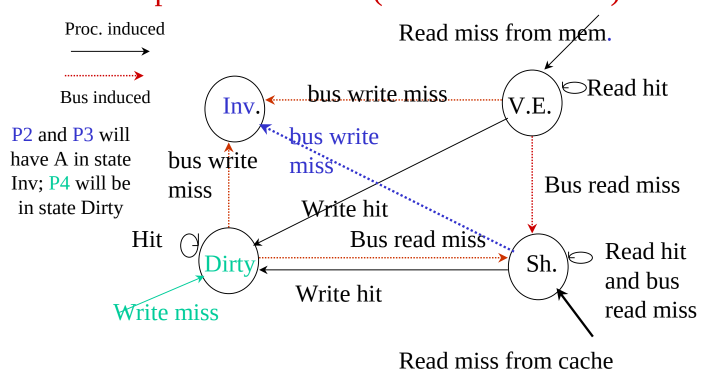
Example: P2 reads A (A only in memory)



Example: P3 reads A (A comes from P2)



Example: P4 writes A (A comes from P2)



Cache Parameters for Multiprocessors

- In addition to the 3 C's types of misses, add a 4th C: coherence misses
- As cache sizes increase, the misses due to the 3 C's decrease but coherence misses increase
- Shared data has been shown to have less spatial locality than private data; hence large block sizes could be detrimental
- Large block sizes induce more false sharing
  - P1 writes the first part of line A; P2 writes the second part. From the coherence protocol viewpoint, both look like "write A"

Performance of Snoopy Protocols

- Protocol performance depends on the length of a write run
- Write run: sequence of write references by 1 processor to a shared address (or shared block) uninterrupted by either access by another processor or replacement
  - Long write runs better to have write invalidate
  - Short write runs better to have write update
- There have been proposals to make the choice between protocols at run time
  - Competitive algorithms

What About Cache Hierarchies?

- Implement snoopy protocol at L2 (board-level) cache
- Impose multilevel inclusion property
  - Encode in L2 whether the block (or part of it if blocks in L2 are longer than blocks in L1) is in L1 (1 bit/block or subblock)
  - Disrupt L1 on bus transactions from other processors only if data is there, i.e., L2 shields L1 from unnecessary checks
  - Total inclusion might be expensive (need for large associativity) if several L1's share a common L2 (like in clusters). Instead use partial inclusion (i.e., possibility of slightly over invalidating L1)