Protocol Design Space of Snooping Cache Coherent Multiprocessors

Recap

- Snooping cache coherence
  - solve difficult problem by applying extra interpretation to naturally occuring events
    - » state transitions, bus transactions
  - write-thru cache
    - » 2-state: invalid, valid
    - » no new transaction, no new wires
    - » coherence mechanism provides consistency, since all writes in bus order
    - » poor performance
- Coherent memory system
- Sequential Consistency

Sequential Consistency

- Memory operations from a proc become visible (to itself and others) in program order
- There exist a total order, consistent with this partial order - i.e., an interleaving
  - the position at which a write occurs in the hypothetical total order should be the same with respect to all processors
- Sufficient Conditions
  - every process issues mem operations in program order
  - after a write operation is issued, the issuing process waits for the write to complete before issuing next memory operation
  - after a read is issued, the issuing process waits for the read to complete and for the write whose value is being returned to complete (gloabaly) befor issuing its next operation

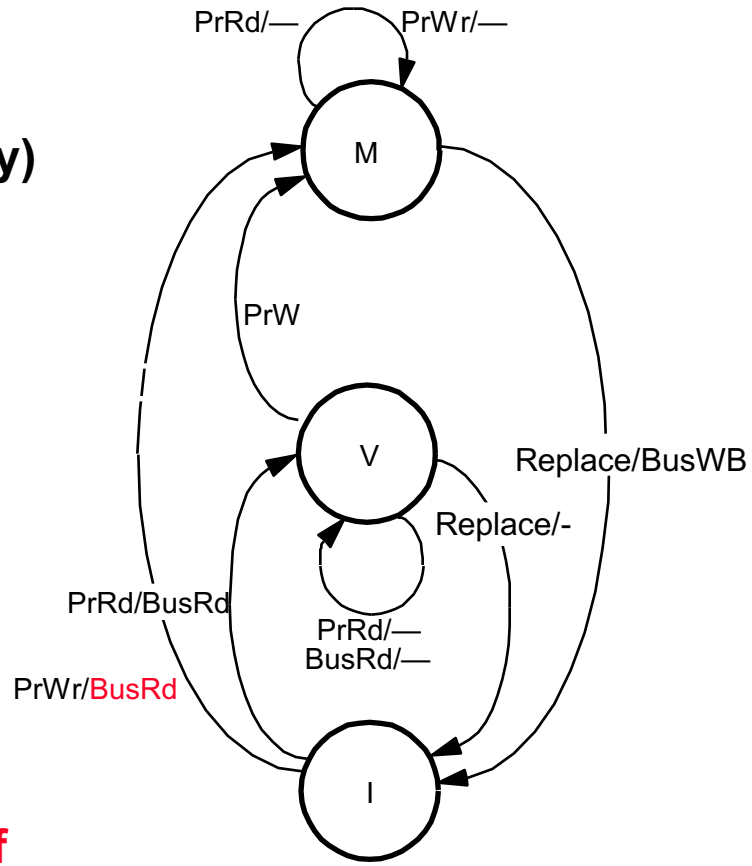
Write-back Caches

- 2 processor operations
  - PrRd, PrWr
- 3 states
  - invalid, valid (clean), modified (dirty)
  - ownership: who supplies block
- 2 bus transactions:
  - read (BusRd), write-back (BusWB)
  - only cache-block transfers

=> treat Valid as "shared" and Modified as "exclusive"

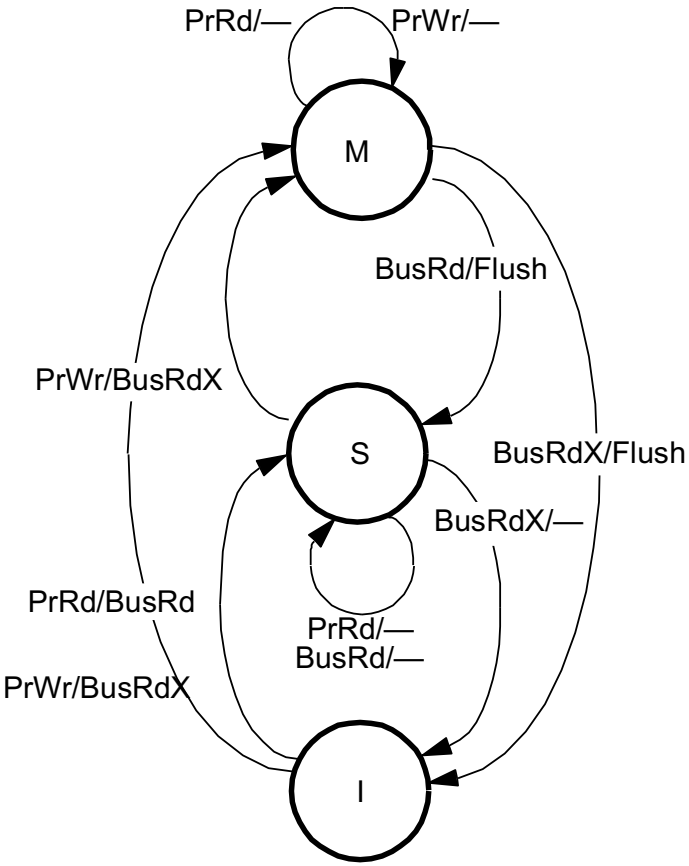
=> introduce one new bus transaction

  - read-exclusive: read for purpose of modifying (read-to-own)

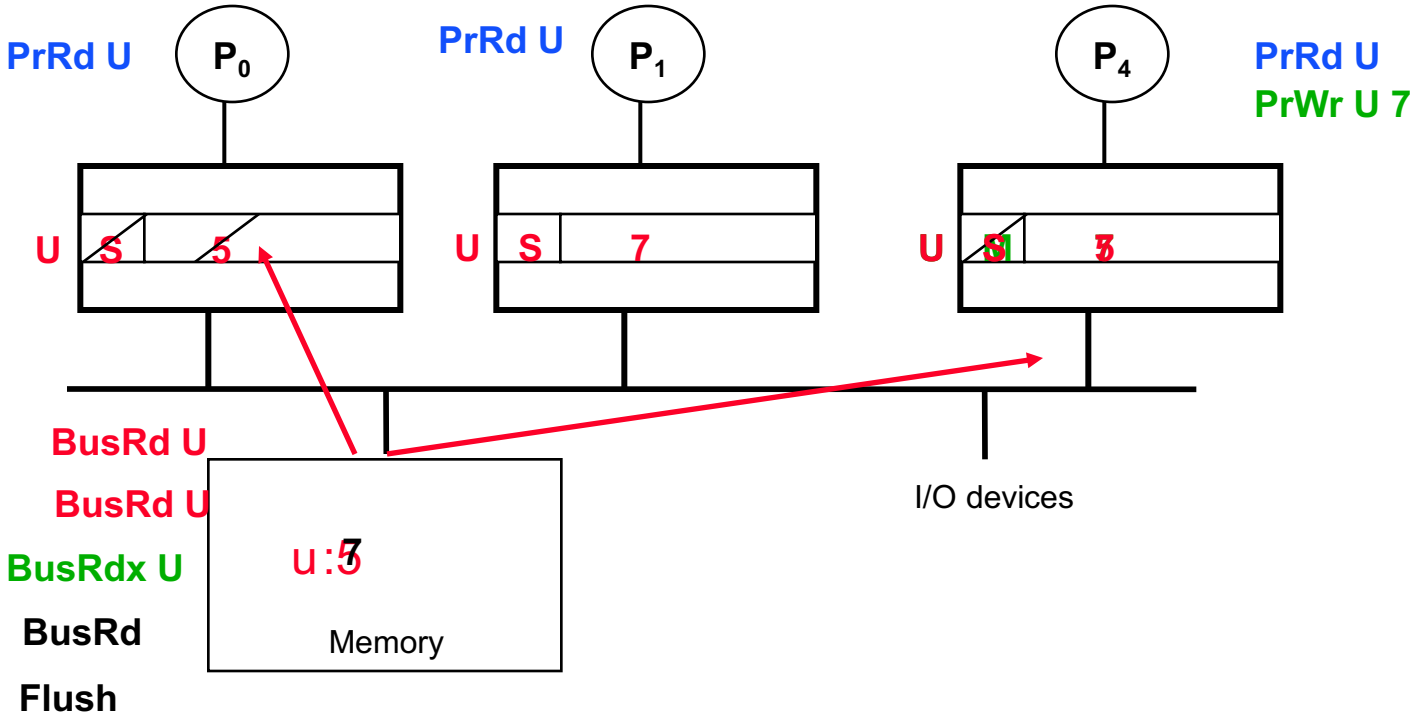


MSI Invalidate Protocol

- **Read obtains block in "shared"**
  - **even if only cache copy**
- **Obtain exclusive ownership before writing**
  - **BusRdx causes others to invalidate (demote)**
  - **If M in another cache, will flush**
  - **BusRdx even if hit in S**
    - » **promote to M (upgrade)**
- **What about replacement?**
  - **S->I, M->I as before**



Example: Write-Back Protocol



Lower-level Protocol Choices

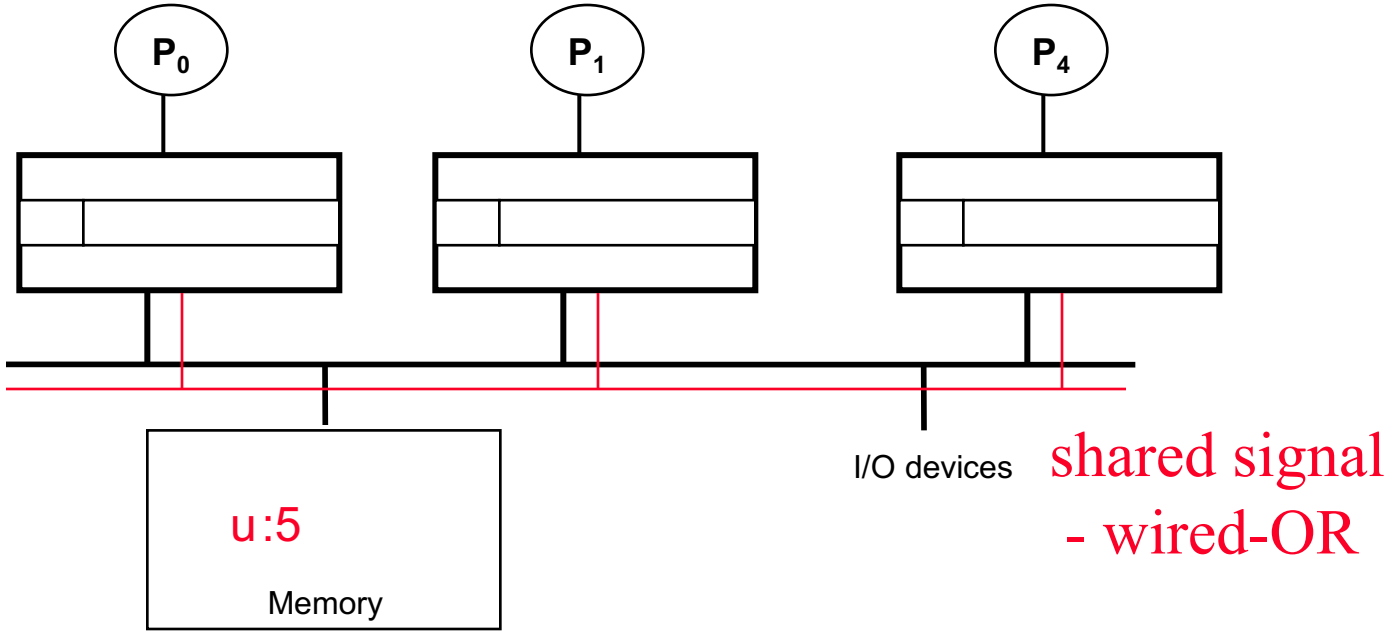
- **BusRd observed in M state: what transitition to make?**
  - **M ----> I**
  - **M ----> S**
  - **Depends on expectations of access patterns**
- **How does memory know whether or not to supply data on BusRd?**
- **Problem: Read/Write is 2 bus xactions, even if no sharing**
  - » **BusRd (I->S) followed by BusRdX or BusUpgr (S->M)**
  - » **What happens on sequential programs?**

MESI (4-state) Invalidation Protocol

- **Add *exclusive* state**
  - **distinguish exclusive (writable) and owned (written)**
  - **Main memory is up to date, so cache not necessarily owner**
  - **can be written locally**
- **States**
  - **invalid**
  - **exclusive or *exclusive-clean* (only this cache has copy, but not modified)**
  - **shared (two or more caches may have copies)**
  - **modified (dirty)**
- **I -> E on PrRd if no cache has copy**

  => How can you tell?

Hardware Support for MESI



- All cache controllers snoop on BusRd
- Assert ‘shared’ if present (S? E? M?)
- Issuer chooses between S and E
  - how does it know when all have voted?

MESI State Transition Diagram

- **BusRd(S) means shared line asserted on BusRd transaction**
- **Flush': if cache-to-cache xfers**
  - **only one cache flushes data**
- **MOESI protocol: Owned state: exclusive but memory not valid**