Cache Coherence

Prof. Seokin Hong

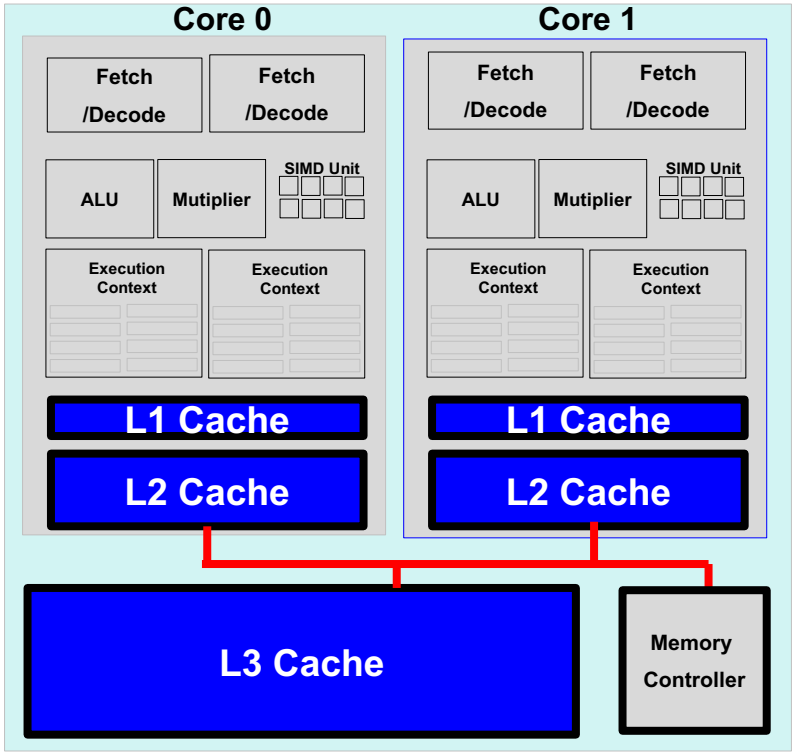
Next ?

- **Cores share memory (shared-memory multiprocessor model)**

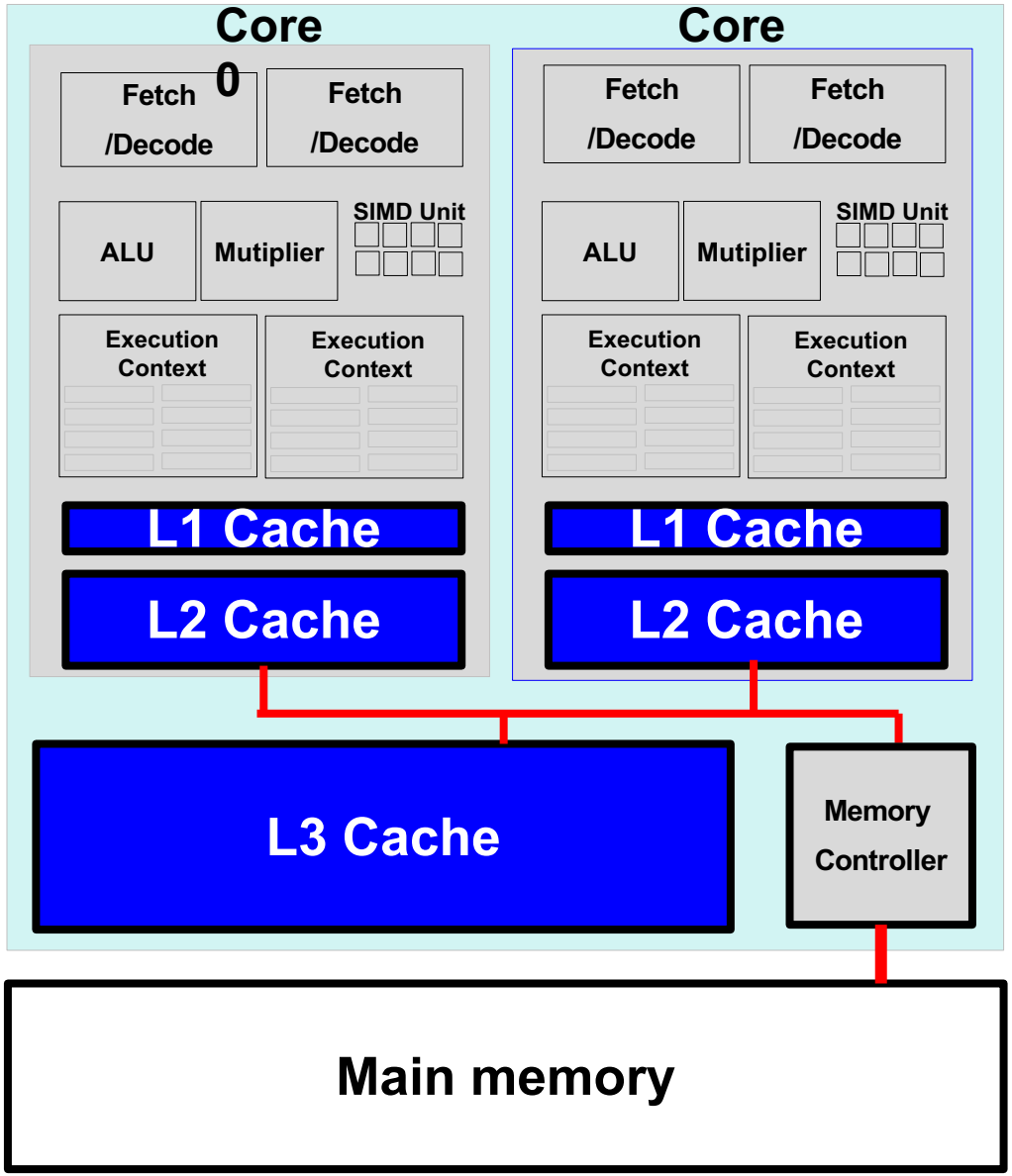
  → Cache Coherence and Memory Consistency

- **Cores are connected each other through intra-chip network**

  → Interconnection Network

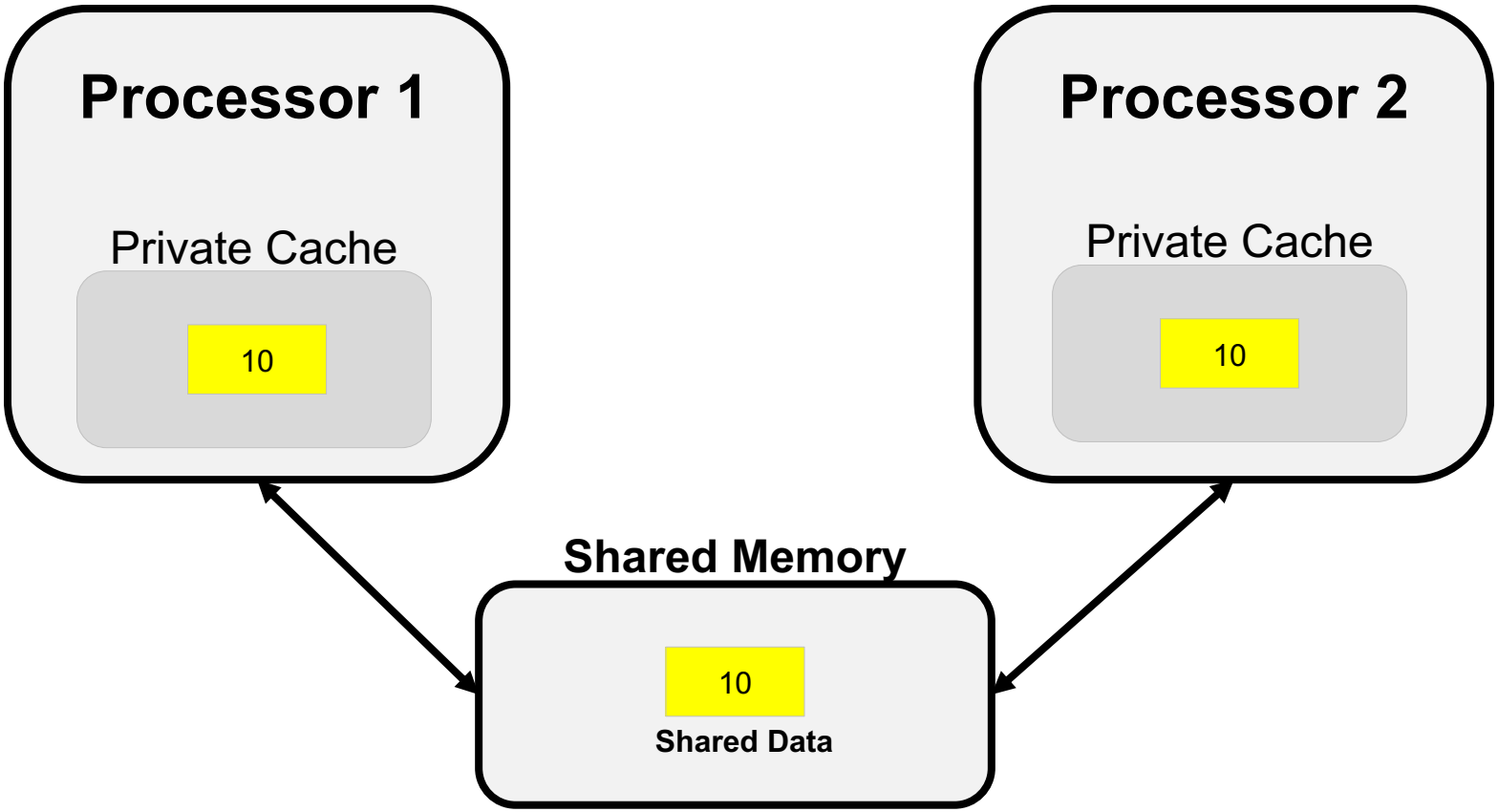


Shared and Private Memories



Cache Coherence

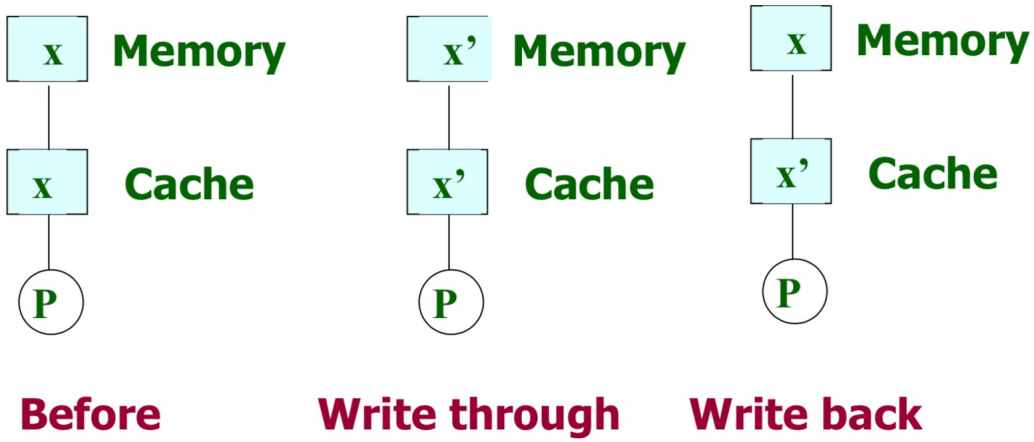
- A problem might happen in a multiprocessor system



What happen if processor 1 updates the shared data!!

Cache Coherence (Cont'd)

- Write-Through vs Write-Back (quick review)
  - **Write-Through policy**
    - The memory is updated every time the cache is updated
  - **Write-back policy**
    - The memory is updated only when the cache block in the cache is being replaced.



From Hesham El-Rewini & Mostafa Abd-El-Barr slide

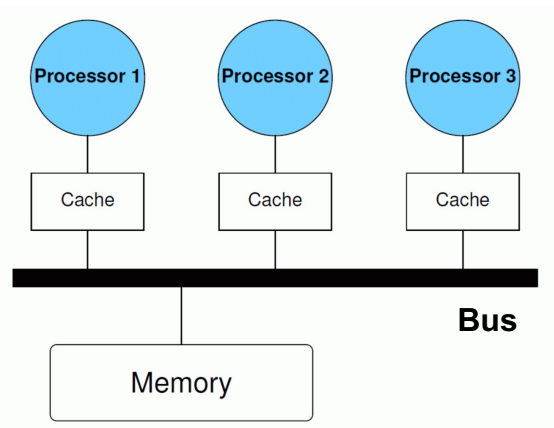
Cache Coherence (Cont'd)

- Memory System is Coherent if..
  - **P** writes **X**, **P** reads **X**
    - X returns value written by **P**
  - **Q** writes **X**, <sufficient time period>, **P** reads **X**
    - X returns value written by **Q**
  - **Writes are serialized**
    - Written values seen in the same order by all processors

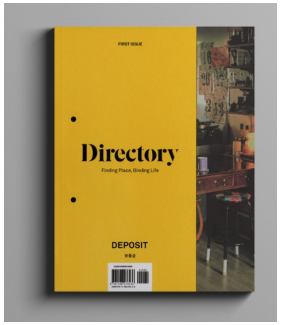
- **Coherence** defines **<u>what</u>** values can be returned by a read
- **Consistency** defines **<u>when</u>** a written value is returned by a read
  - If processor P writes to location A and then to B, then any processor Q that sees the new value in B must also see the new value in A

Cache Coherence (Cont'd)

- Coherent Caches provide:
  - **Migration**: movement of data
  - **Replication**: multiple copies of data



- **Cache coherence protocols**
  - **Directory-based**
    - Shared status of each cache line (a.k.a. cache block) kept in one location
    - Single directory for small core counts, or distributed directories for many cores



  - **Snooping-based**
    - Each core tracks the **state** of each cache line
    - Watch bus activity and carry out the appropriate coherency commands when necessary
    - Requires broadcast link (e.g., data bus)

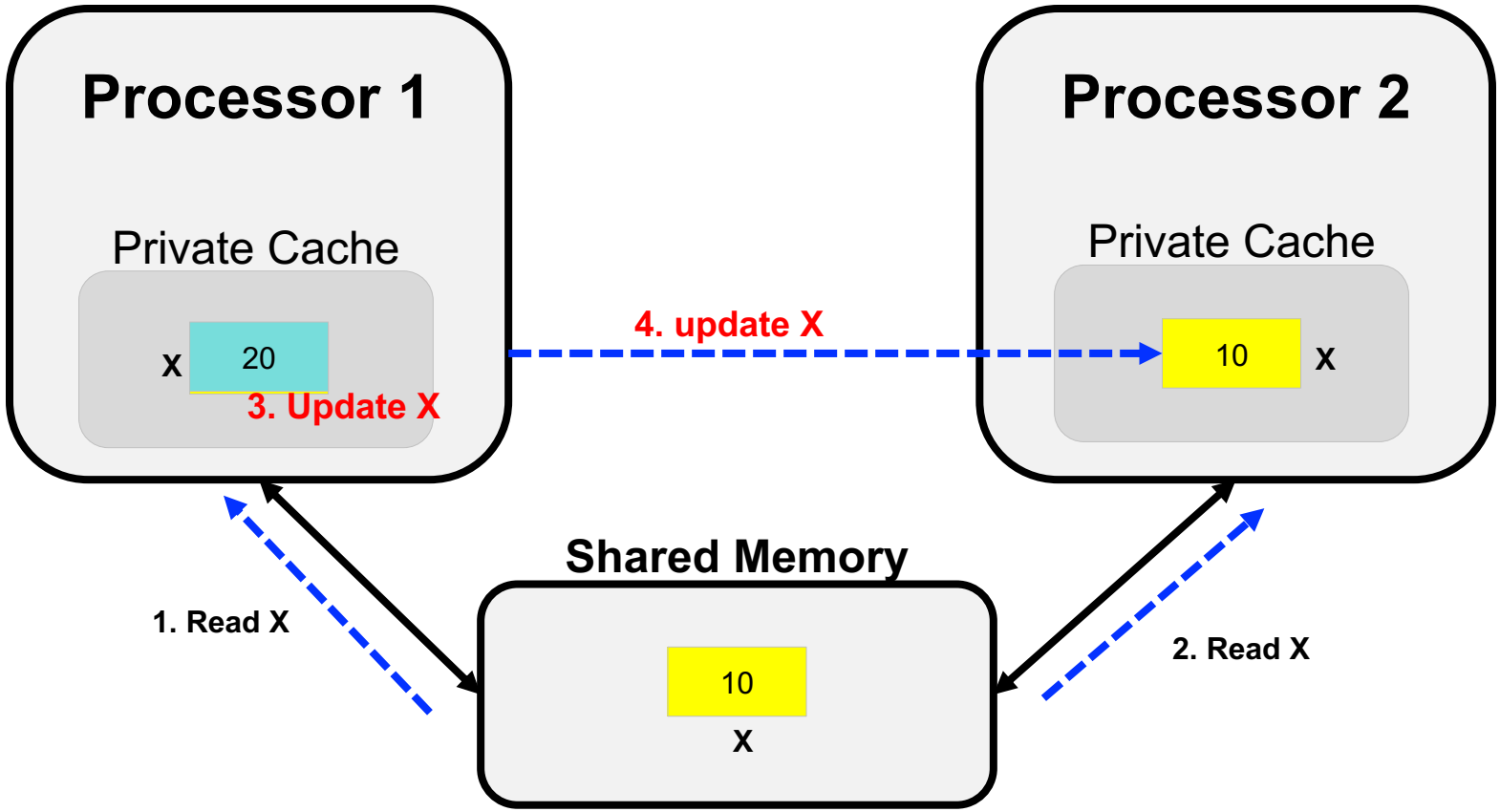
Snooping Cache Protocols

- **Write-update Protocol (write broadcast)**
  - Upon write, make sure all copies are the same
  - Requires more bandwidth

- **Write-invalidate protocol (2-state invalidate protocol)**
  - Upon write, make sure there is only one valid copy
  - Most common protocol

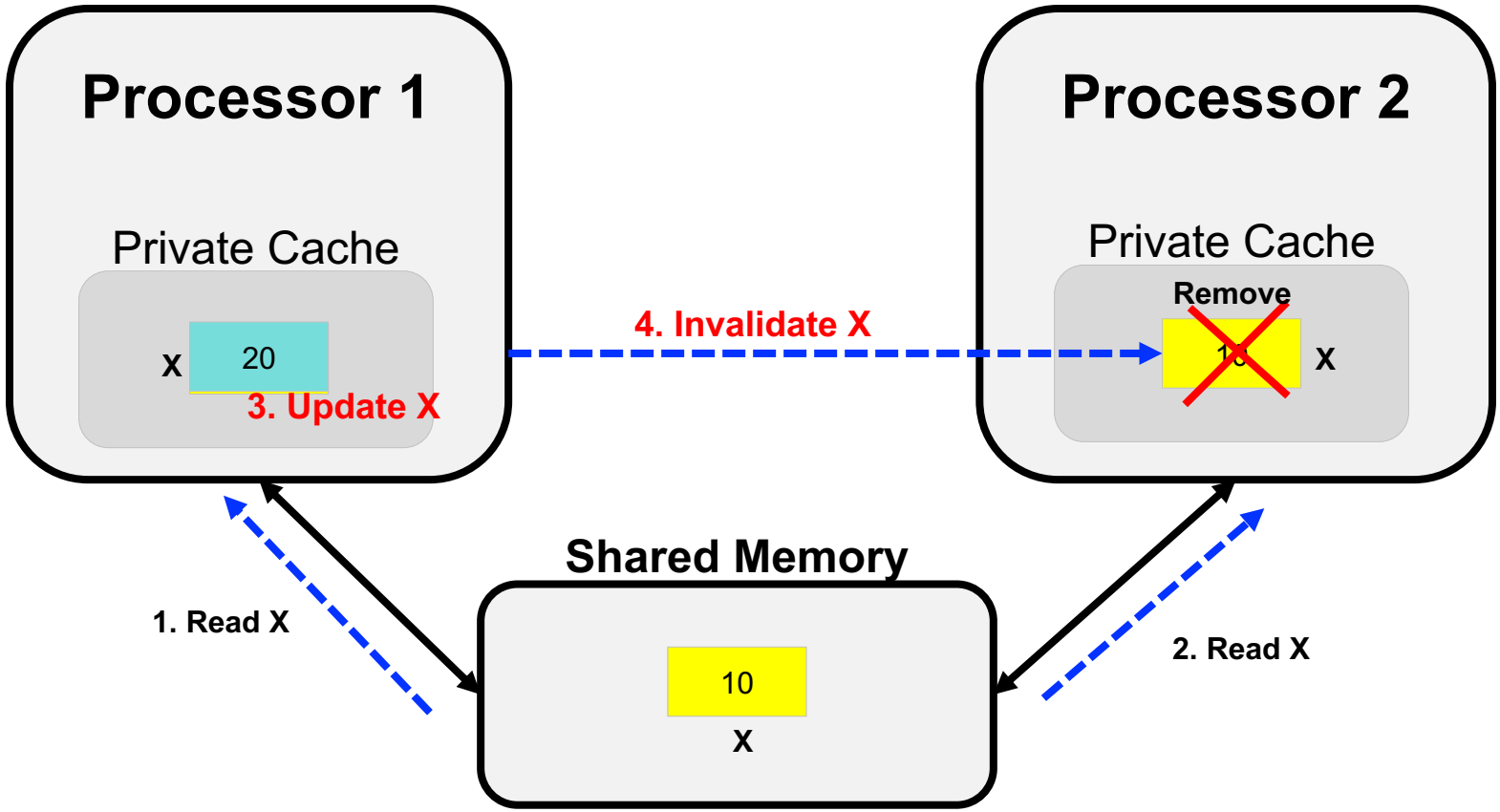
Snooping Cache Protocols (Cont'd)

- **Write-update Protocol**



Snooping Cache Protocols (Cont'd)

- **Write-invalidate Protocol**



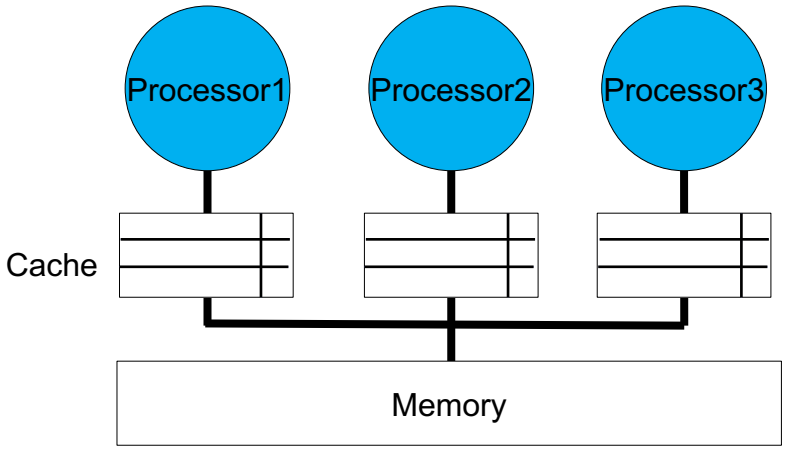
Snooping Cache Protocols (Cont'd)

- **Write-invalidate Protocol (Cont'd)**

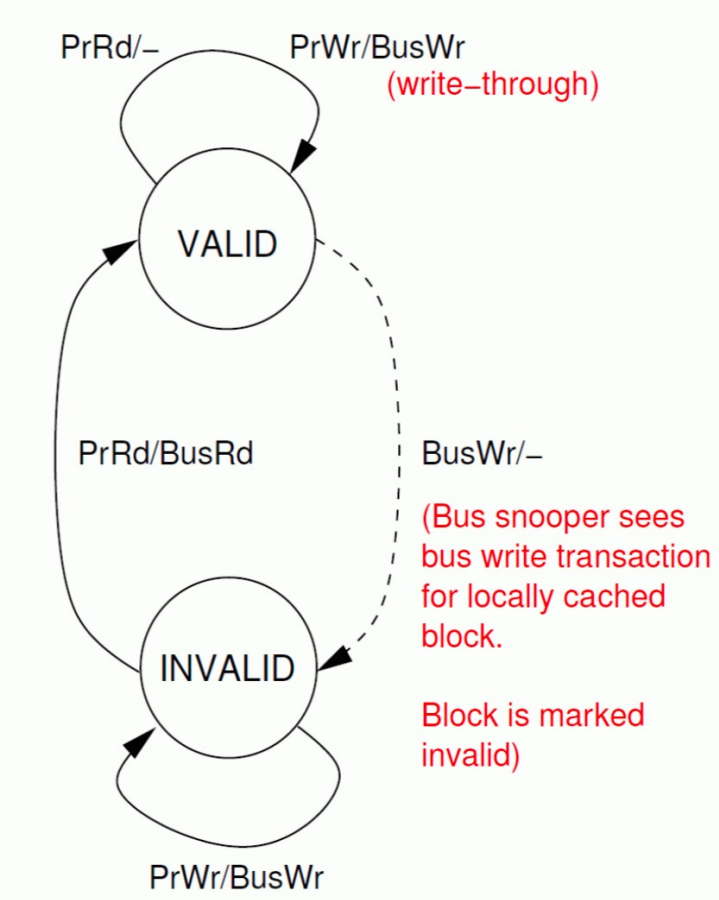
  = 2-state invalidate protocol

  - **Each cache controller receives**
    - Memory requests from a local processor
    - memory **requests** from bus (snooping)

  - **In response, the controller may**
    - update the state of a particular cache line
    - initiate a new bus transaction

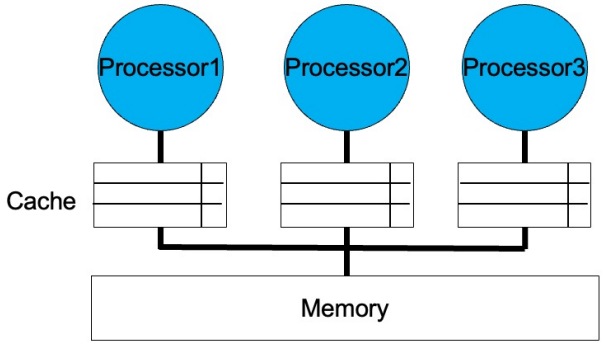


State transition diagram for a cache line

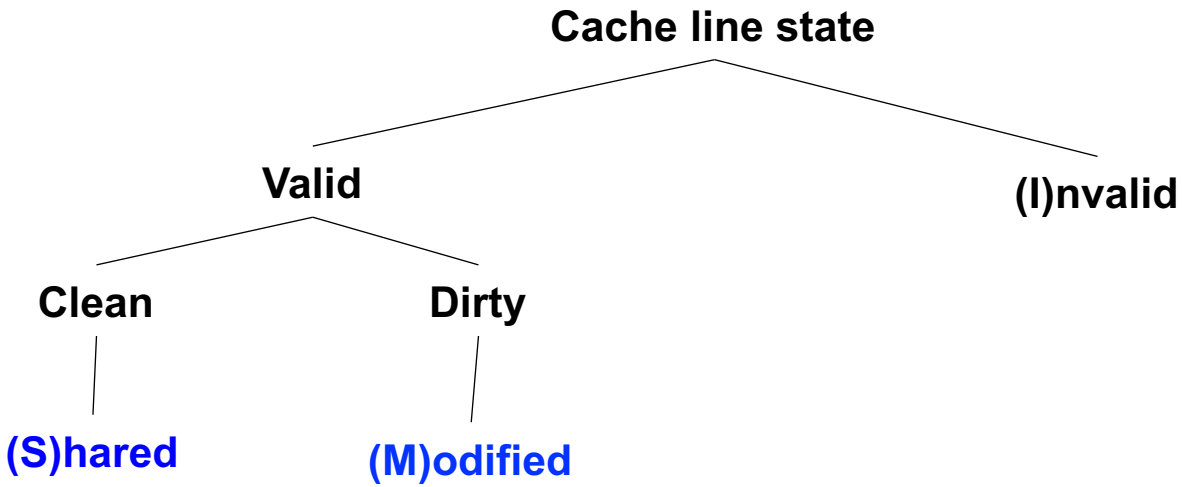


PrRd : Read request from processor
PrWr : Write request from processor
BusRd: Read request from bus (other processors)
BusWr: Write request from bus (other processors)

Snooping Cache Protocols (Cont

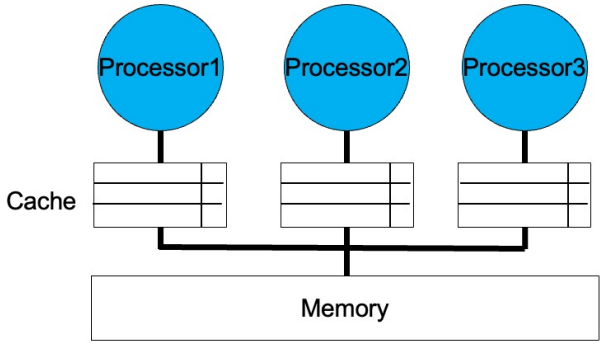


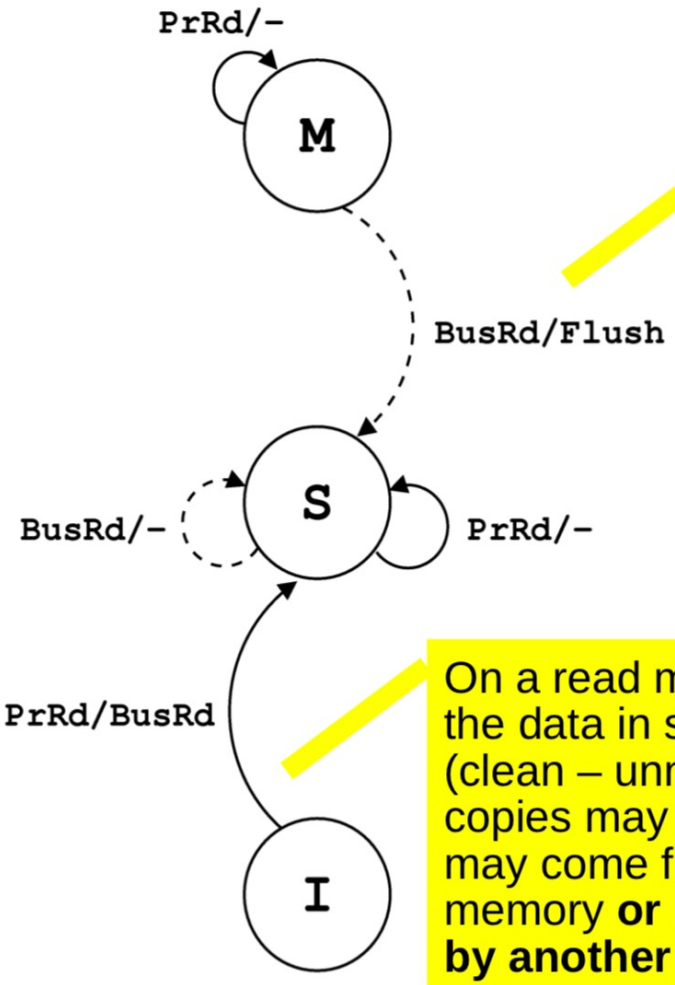
- **MSI (Modified-Shared-Invalid) Protocol**
  - **Cache line state**
    - **(S) shared**
      - The block is present in an unmodified state in this cache
      - The block is up-to-date in main memory
      - Zero or more up-to-date copies exist in other caches
    - **(M)odified**
      - Only this cache has a valid copy, **the copy in memory and other cache is stale**



Snooping Cache Protocols (Cont

- **MSI (Modified-Shared-Invalid) Protocol**





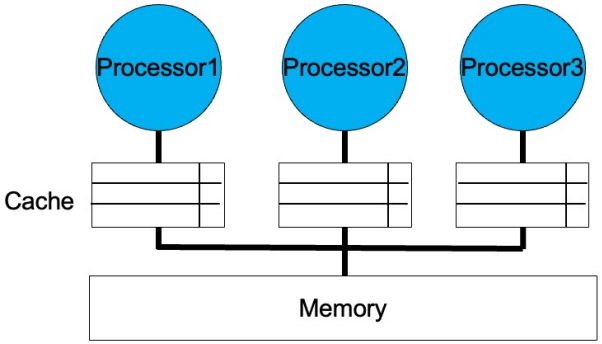
If another cache needs the block that we have in state M (the block is dirty/modified), we must flush the block to memory and provide the data to the requesting cache (over the bus)

On a read miss, we get the data in state S (clean – unmodified, other copies may exist). Data may come from main memory **or be provided by another cache**

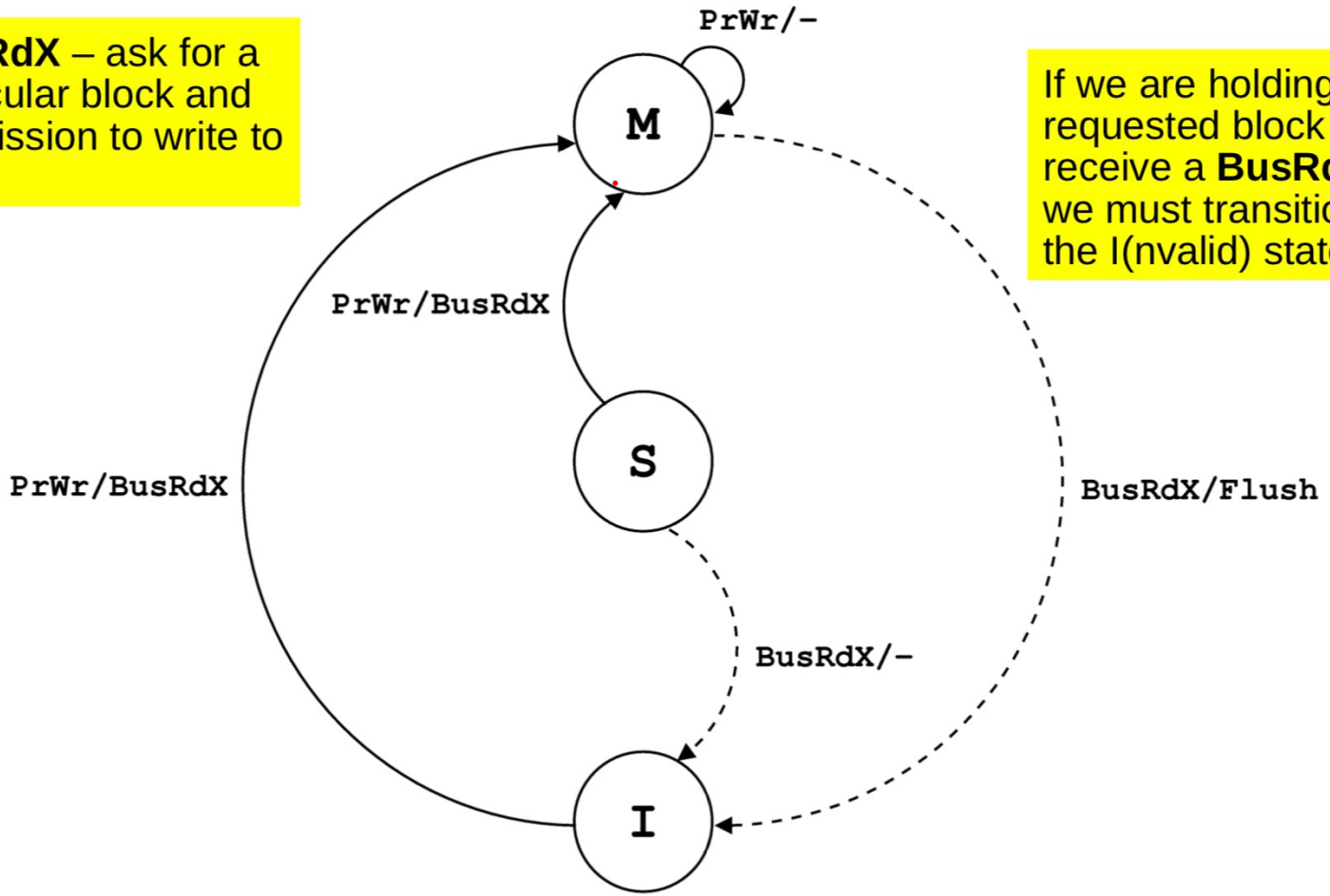
From Robert Mullins's slide

Snooping Cache Protocols (Cont

- **MSI (Modified-Shared-Invalid) Protocol**



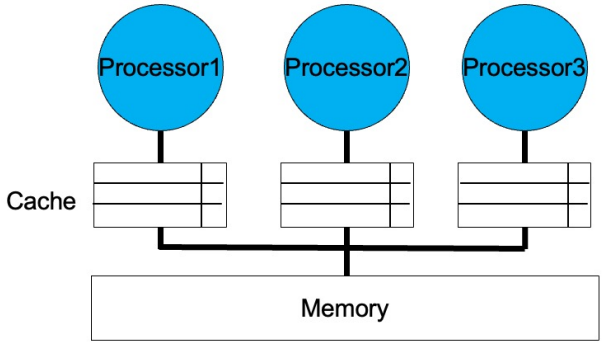
BusRdX – ask for a particular block and permission to write to it.



If we are holding the requested block and receive a **BusRdX**, we must transition to the I(nvalid) state

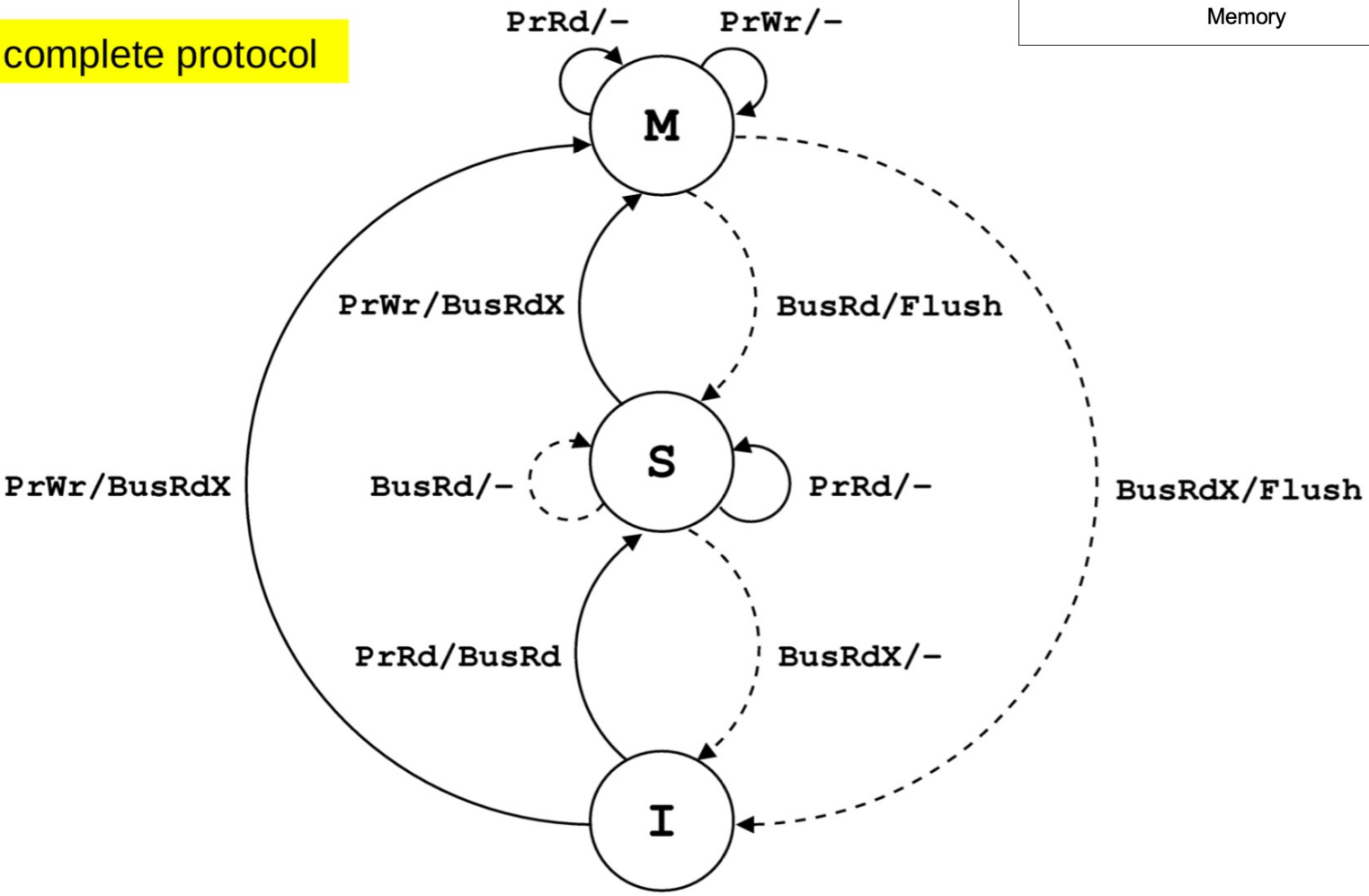
From Robert Mullins's slide

Snooping Cache Protocols (Cont



- **MSI (Modified-Shared-Invalid) Protocol**

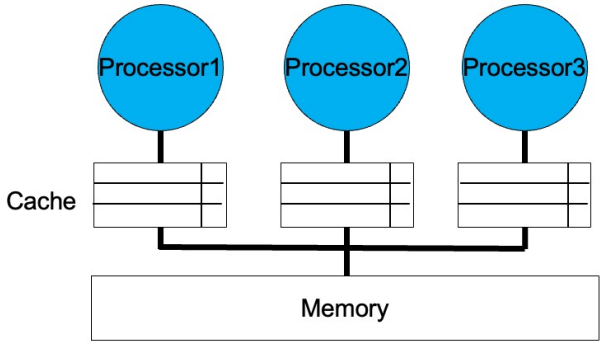
The complete protocol

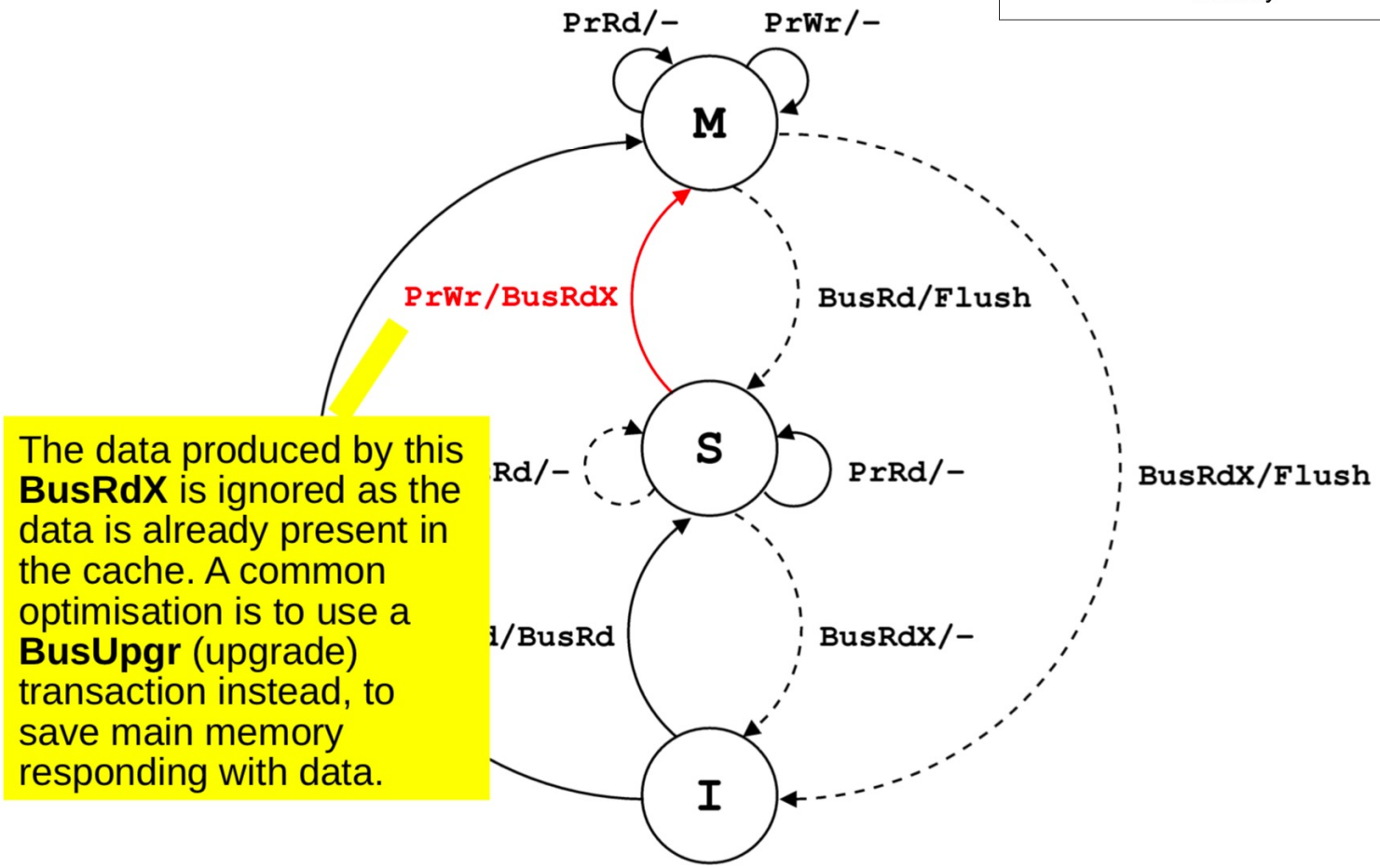


From Robert Mullins's slide

Snooping Cache Protocols (Cont

- **MSI (Modified-Shared-Invalid) Protocol**



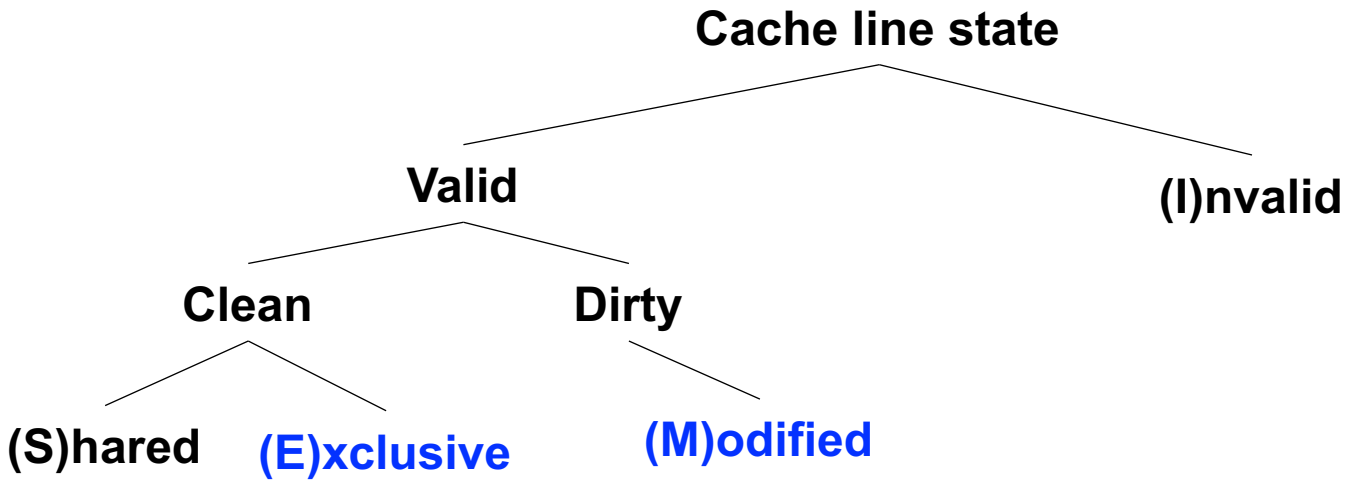


The data produced by this **BusRdX** is ignored as the data is already present in the cache. A common optimisation is to use a **BusUpgr** (upgrade) transaction instead, to save main memory responding with data.

From Robert Mullins's slide

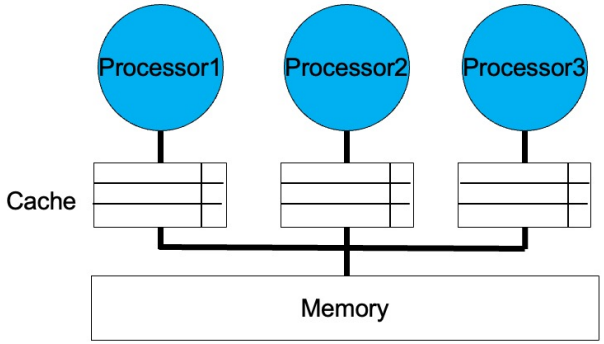
Snooping Cache Protocols (Cont'd)

- **MESI Protocol**
  - Let's consider what happens if we read a block and then subsequently modify it
    - This will require two bus transactions in the MSI protocol
    - But if we know that we have the only copy of the block, the transaction (BusUpgr) required to transition from state S to M is really unnecessary
      - We could safely, and silently, transition from S to M
    - This will be a common event in parallel programs



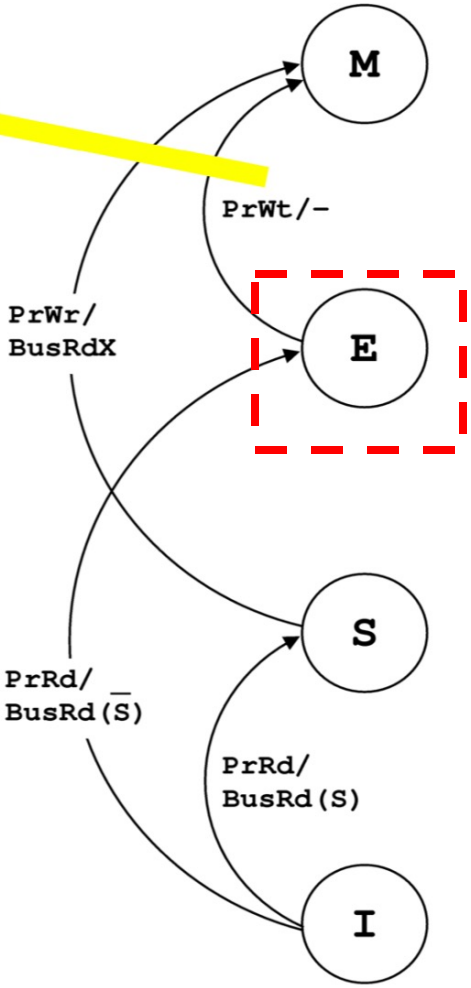
From Robert Mullins's slide

Snooping Cache Protocols (Cont



- **MESI Protocol (Cont'd)**

If we are in state E and need to write, we require no bus transaction to take place



The shared signal (S) is used to determine if any caches currently hold the requested data on a PrRd.

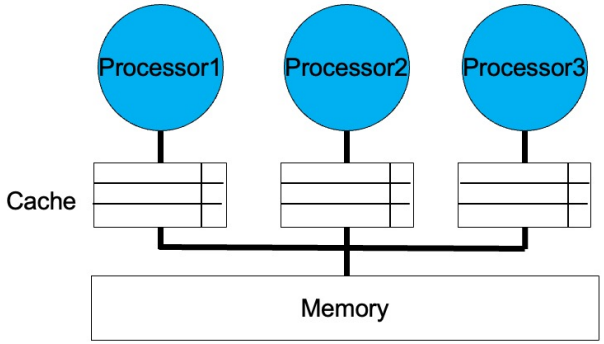
BusRd(S) means the bus read transaction caused the shared signal to be asserted (another cache has a copy of the data). **BusRd(Not S)** means no cache has the data.

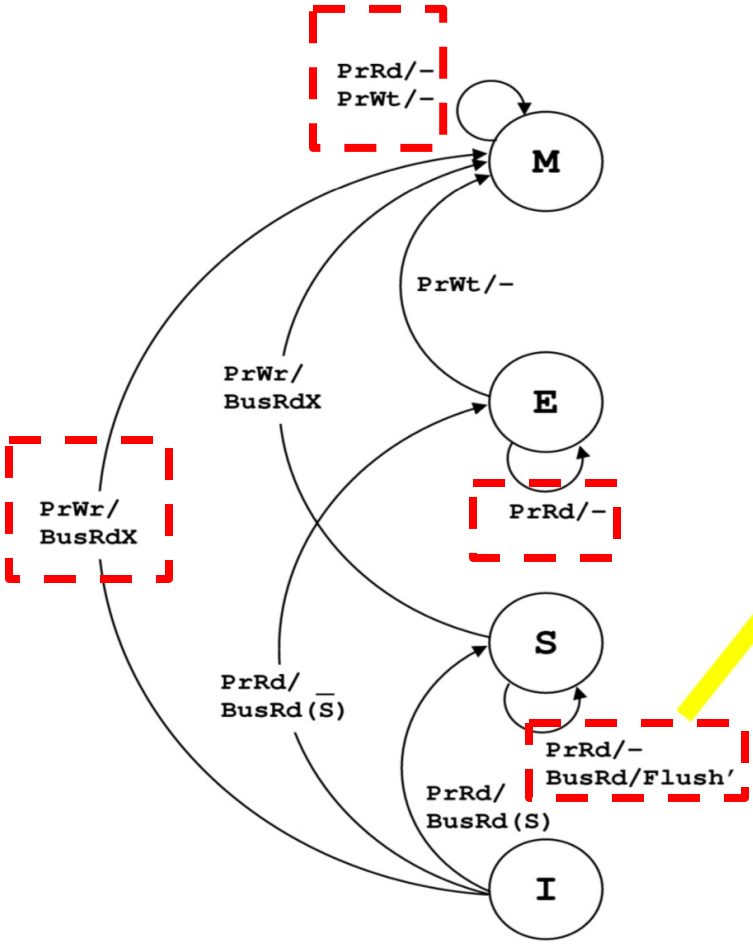
This signal is used to decide if we transition from state I to S or E

From Robert Mullins's slide

Snooping Cache Protocols (Cont

- **MESI Protocol (Cont'd)**





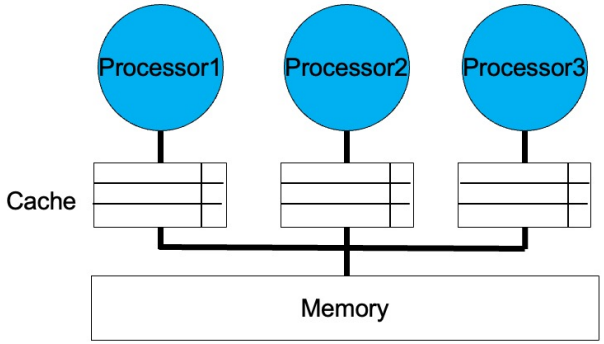
BusRd/Flush'

To enable cache-to-cache sharing we must be sure to select only one cache to provide the required block. Flush' is used to indicate that only **one** cache will flush the data (in order to supply it to the requesting cache). This is easy in a bus-based system of course as only one transaction can take place at a time.

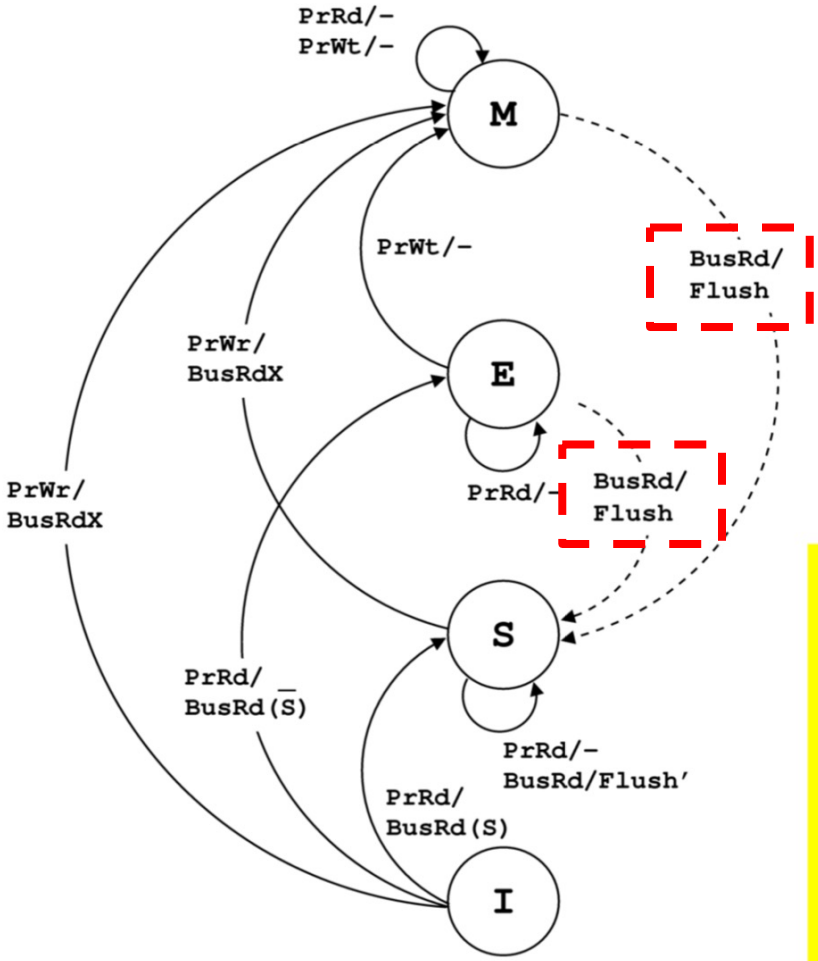
Of course a cache will normally be able to provide the block faster than main memory.

From Robert Mullins's slide

Snooping Cache Protocols (Cont



- **MESI Protocol (Cont'd)**



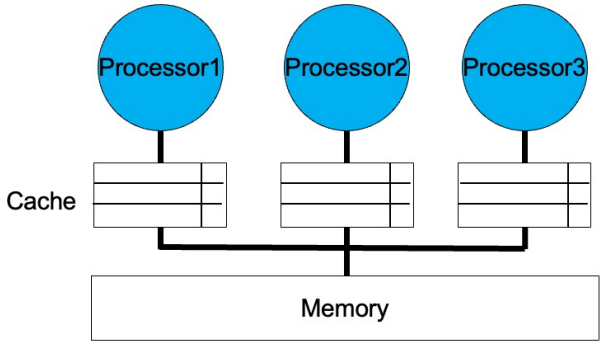
BusRd/Flush

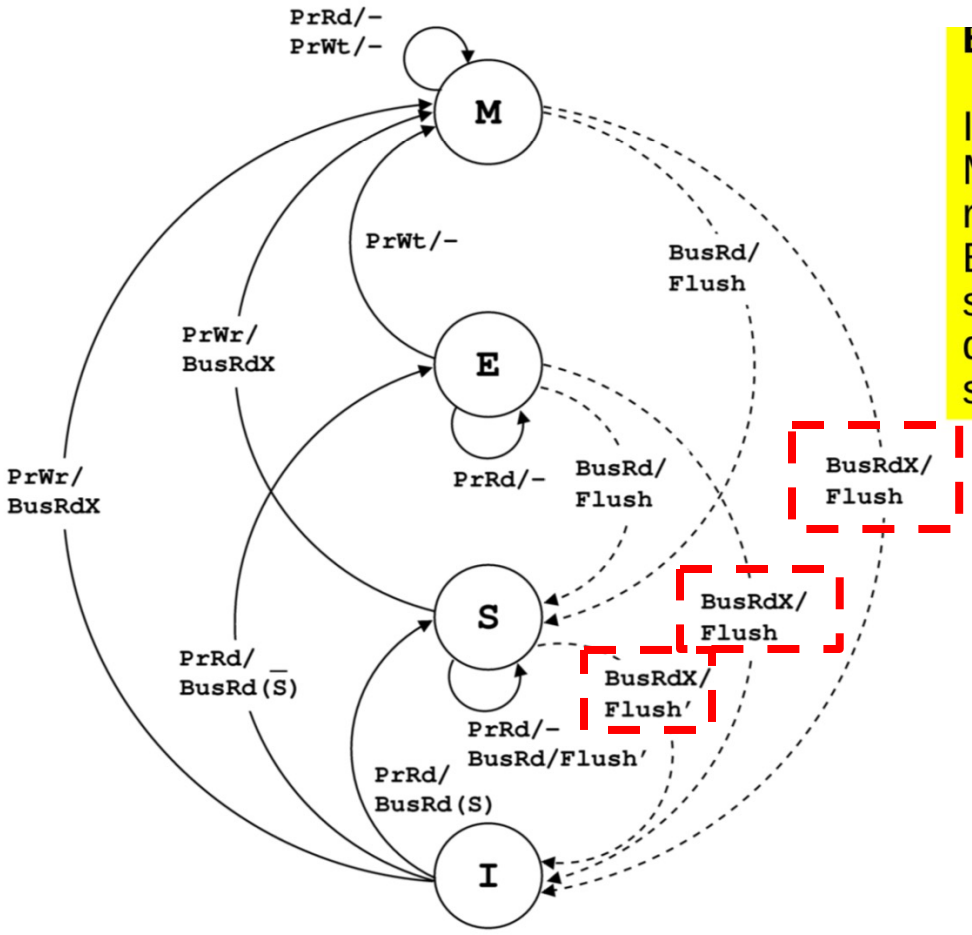
If we are in state M or E and other cache requests the block, we provide the data (we have the only copy) and move to state S (clean & zero or more copies)

From Robert Mullins's slide

Snooping Cache Protocols (Cont

- **MESI Protocol (Cont'd)**





BusRdX/Flush

If we are in states M, E or S and receive a BusRdX we simply flush our data and move to state I

From Robert Mullins's slide

Directory-Based Coherence

- Scalability bottleneck: snooping bandwidth
  - Snooping broadcast are a major drain of cache bus bandwidth
  - Practical scalability limit: 8 cores

- Solution: add a **directory**
  - Directory keeps
    - The state of every cache line
    - Which caches have a copy of the block
  - Directory also must be distributed