CMPEN 431
Introduction to Computer Architecture
Fall 2022

Multiprocessor (1): Cache Coherence (1)

Kiwan Maeng

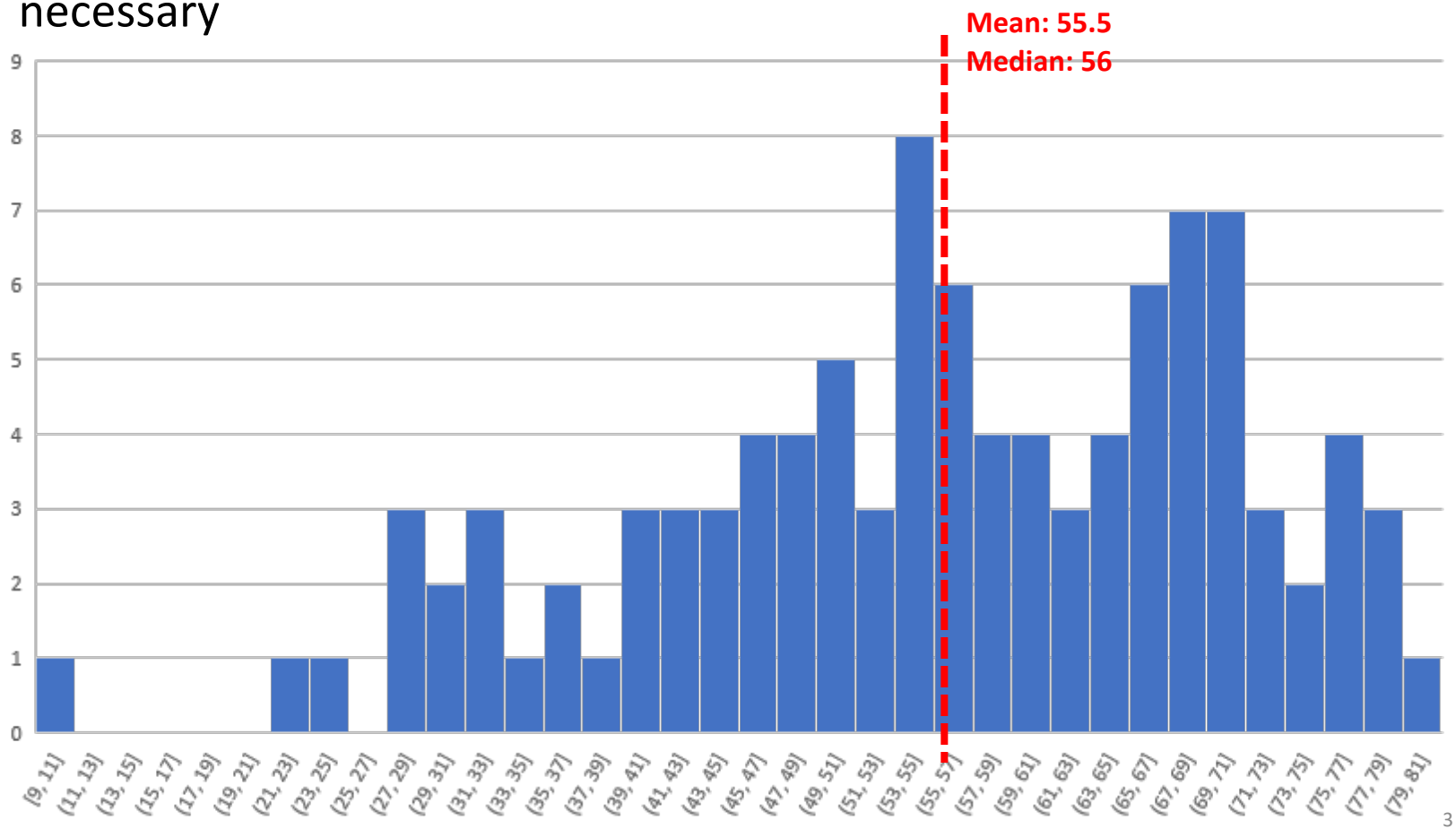
Slides adapted from ECE6100 @ GeorgiaTech
by Hsien-Hsin Sean Lee &
Computer Organization and Design, 5th Edition,
Patterson & Hennessy, © 2014, MK

Exam 2: Partial Answer Change

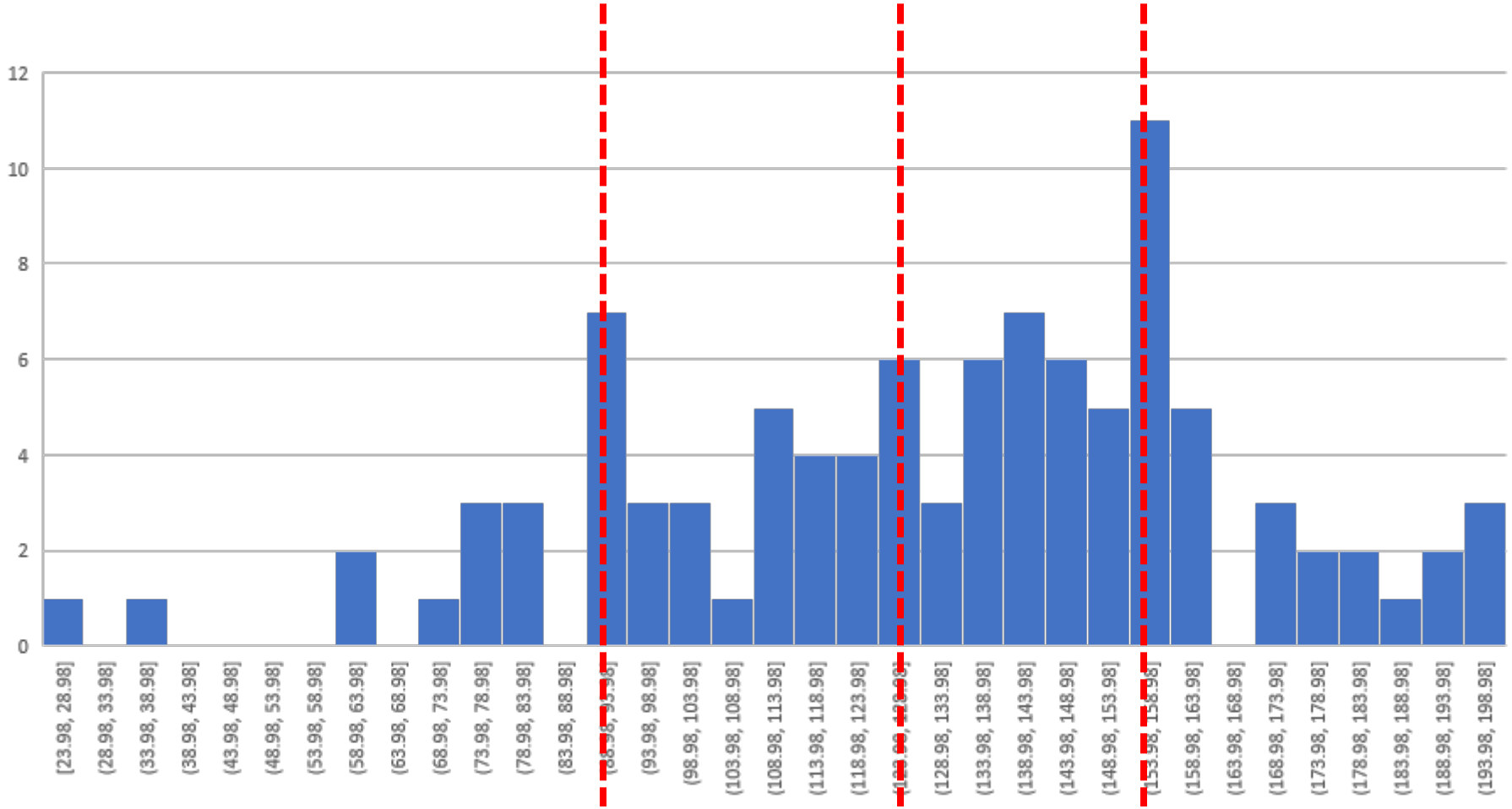
- Q2-2: What is the disadvantage of loop unrolling?
- The answer I taught during class (and the one I was expecting)
  - Binary size increase (use up more memory, increased I-cache misses, …)
- The answer many students wrote
  - Increased use of register file
  - **I decided to give full credit to this answer as well.**
  - I never said this was a disadvantage
  - This is usually less of a problem because many other optimizations alleviate this problem
  - But there can be a case where this can be a problem… so I am giving full credit
  - Keep in mind that a much commonly discussed issue is the increase of binary size, which leads to increased instruction memory use + increased I-cache misses

Exam 2 Results Are Out!

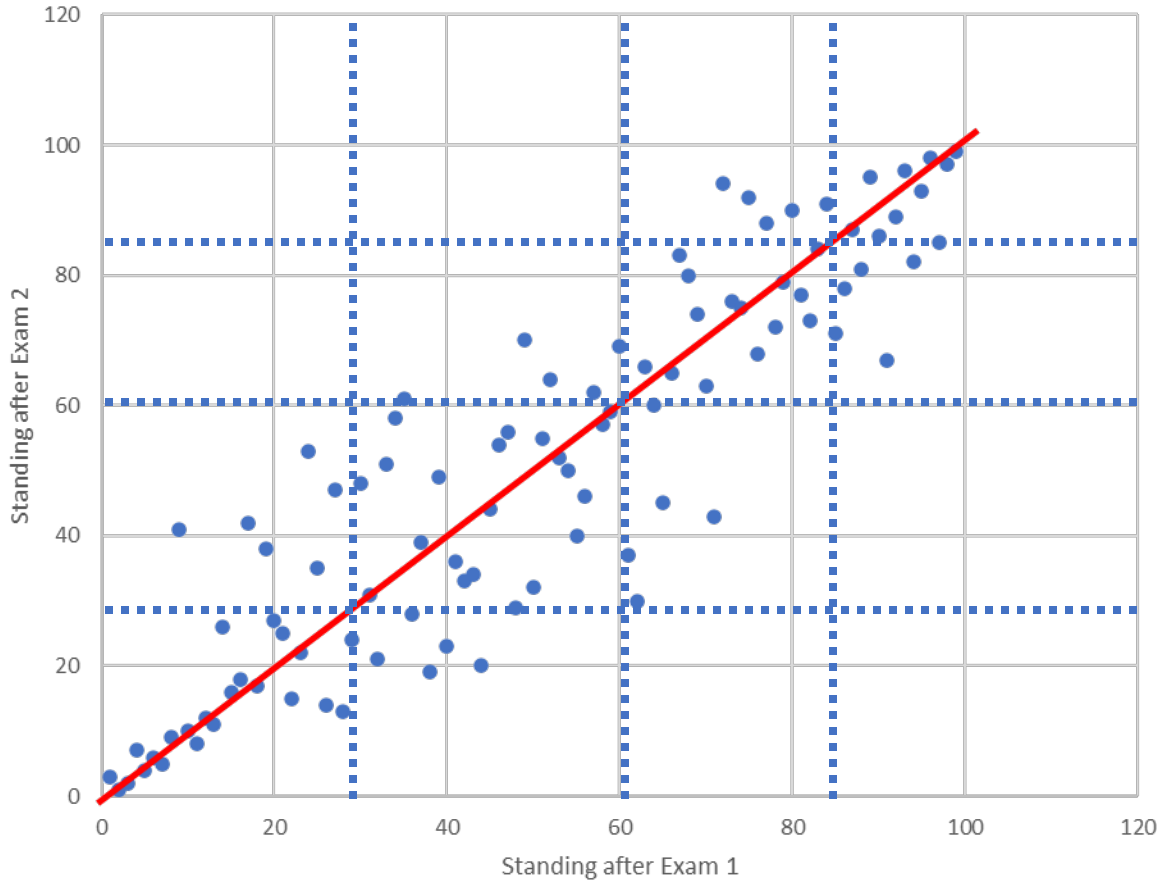
- Please check the solution & submit regrade request
- Submit regrade request by **11/10 (Thurs)** so that we can late-drop if necessary



Exam 1 + 2 (each exam scaled to 100)

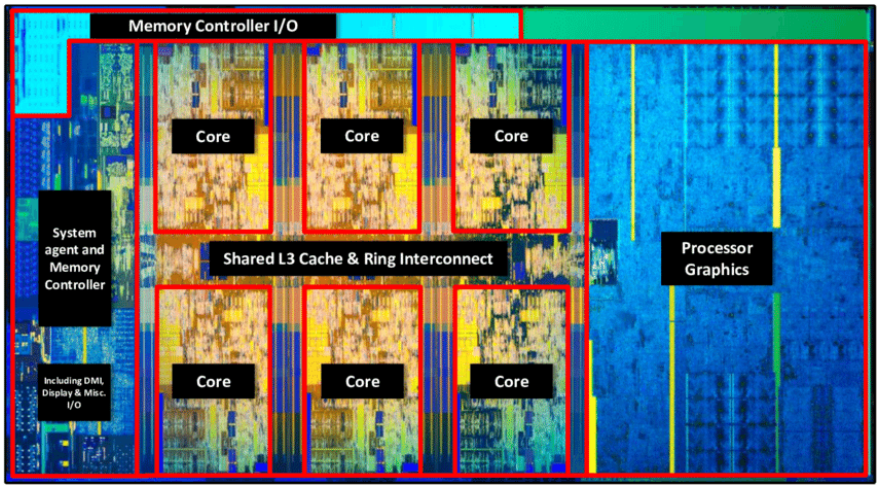


Exam 1 + 2: Studying (or not studying) Actually Makes a Difference!

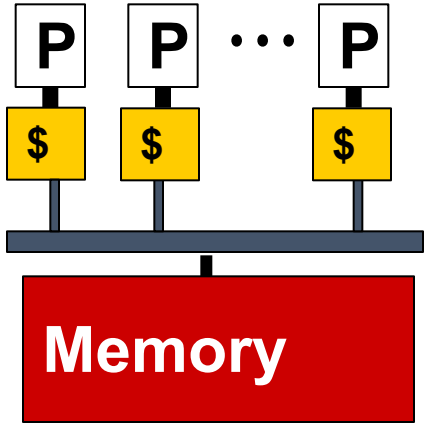


- Many students moved between the A-range and B-range
- Many students moved between the C-range and D-range

Shared-memory Multiprocessors (SMP)



Intel Coffelake Processor
6 cores + graphics co-processor,
Shared LLC

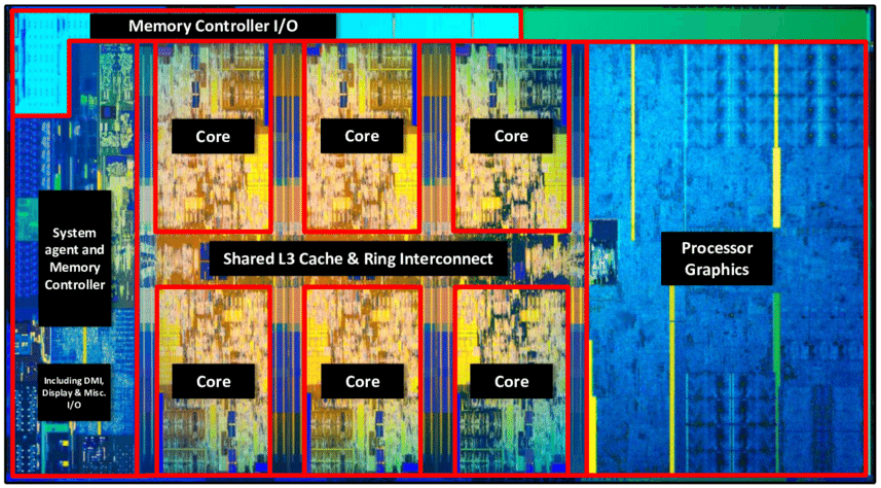


Conceptual diagram

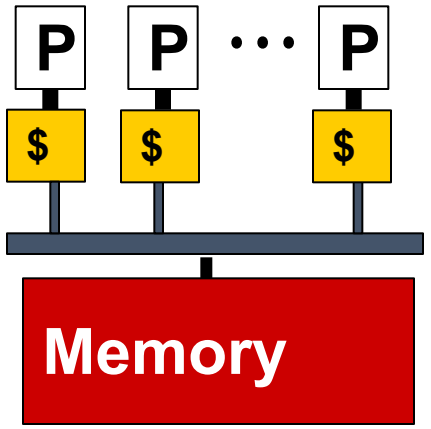
- Moore's Law: transistor counts kept increasing
- Pipelining / OoO was hitting the wall… what should we do?

  → Multicore/multiprocessors!
- Shared-memory multiprocessors (SMP): a single physical address space across all processors

Shared-memory Multiprocessors (SMP)



Intel Coffelake Processor
6 cores + graphics co-processor,
Shared LLC

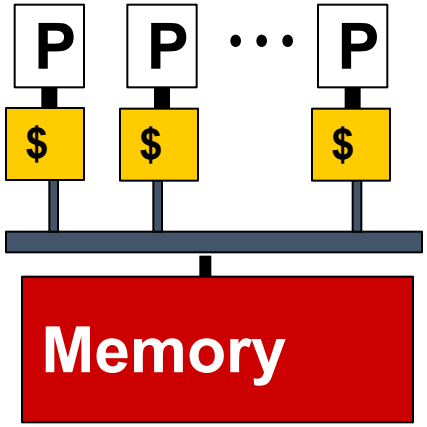


Conceptual diagram

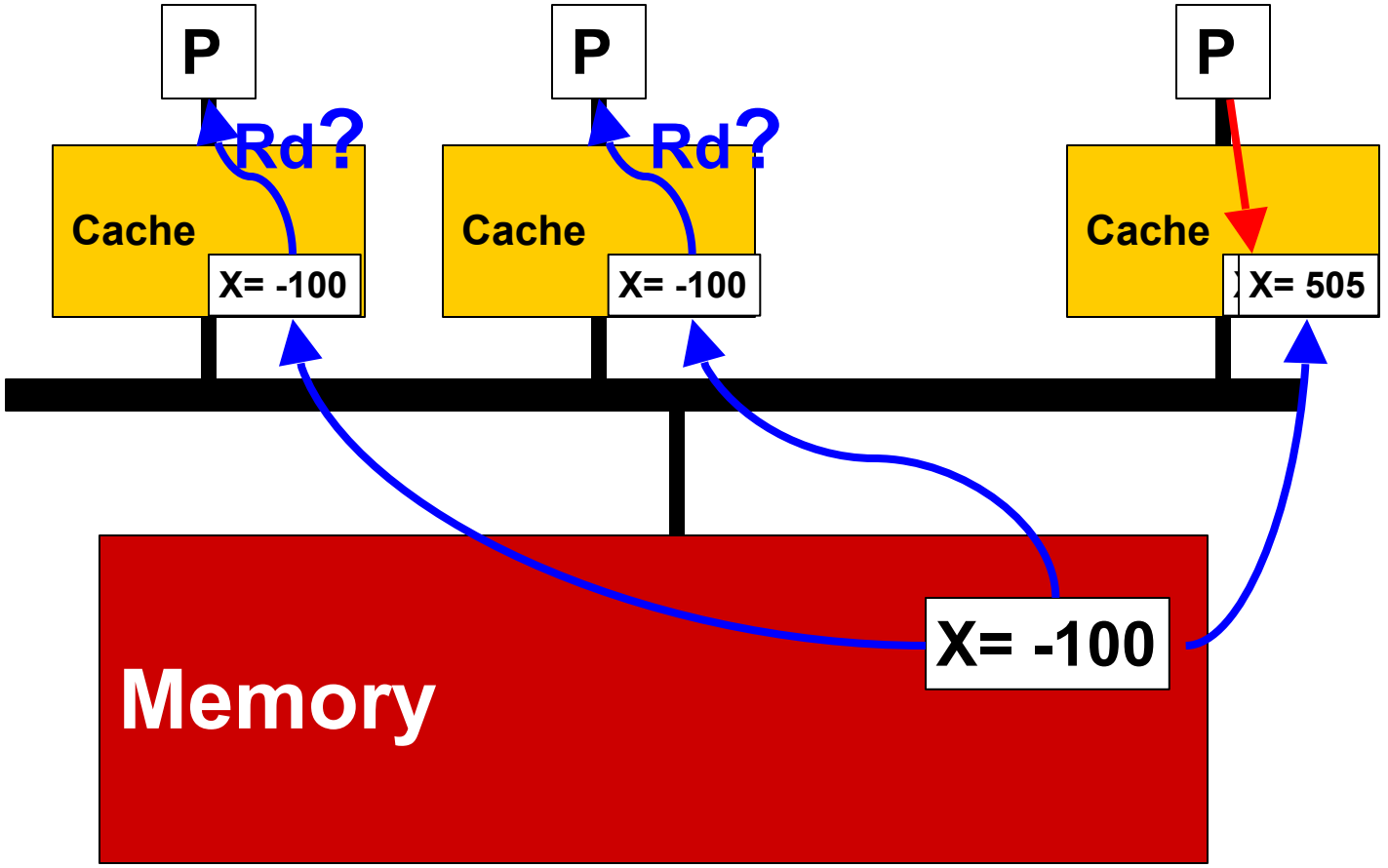
Complexities of an SMP
- **Cache coherence:** defines the ordering of writes to a single address location
- Memory consistency: defines the ordering of reads and writes to all memory location
- Synchronization: allowing only one processor to access data at a time

Cache Coherence

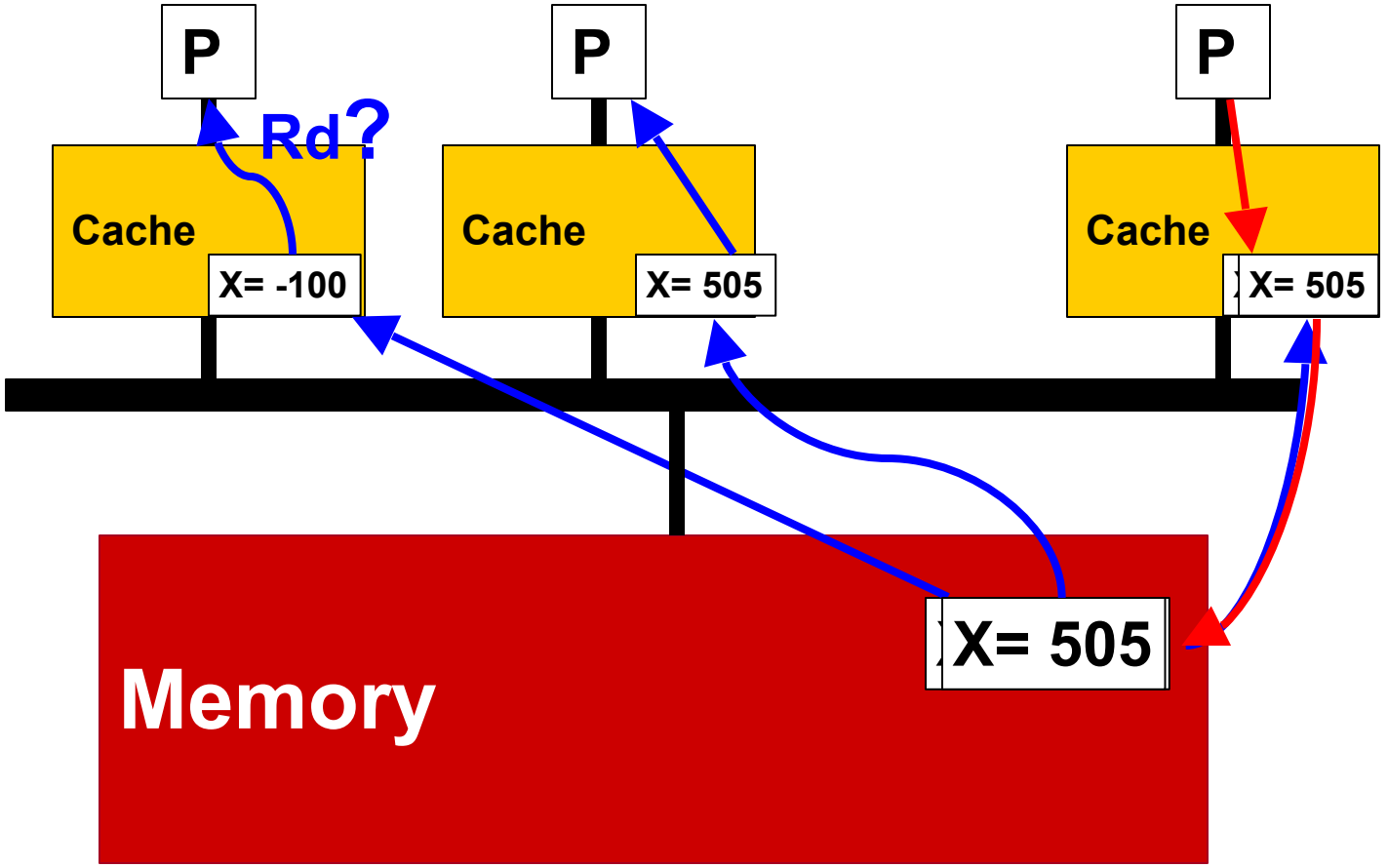
- If P writes to a location X and reads X, if there were no other processors written to X in between, P should read the value it wrote.
- If P1 writes to a location X and P2 reads X, if there were no other writes to X in between and if the two were sufficiently separated in time, P2 should read the value P1 wrote.
- Two writes to the same location by any two processors are seen in the same order by all processors (writes are serializes).
- Seem obvious?



Example (Writeback Cache)



Example (Write-through Cache)



Defining Coherence

- An MP is coherent if the results of any execution of a program can be reconstructed by a hypothetical *serial order*

Implicit definition of coherence

- Write propagation
  - Writes are visible to other processes
- Write serialization
  - All writes to the same location are seen in the same order by all processes (to "all" locations called write atomicity)
  - E.g., w1 followed by w2 seen by a read from P_1, will be seen in the same order by all reads by other processors P_i

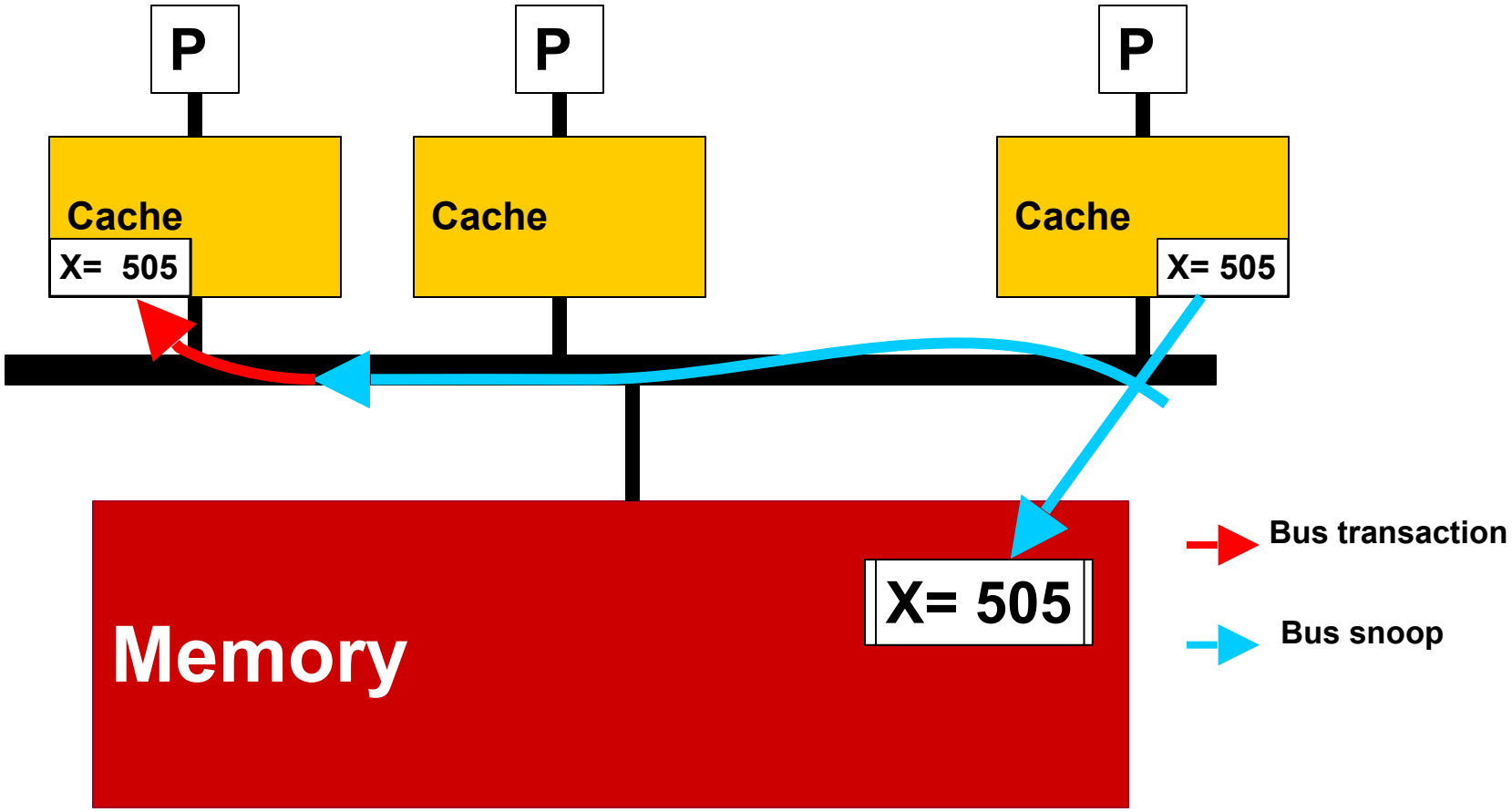
Bus Snooping based on Write-Through Cache

- All the writes will be shown as a transaction on the shared bus to memory

- Two protocols
  - Update-based Protocol
  - Invalidation-based Protocol

Bus Snooping

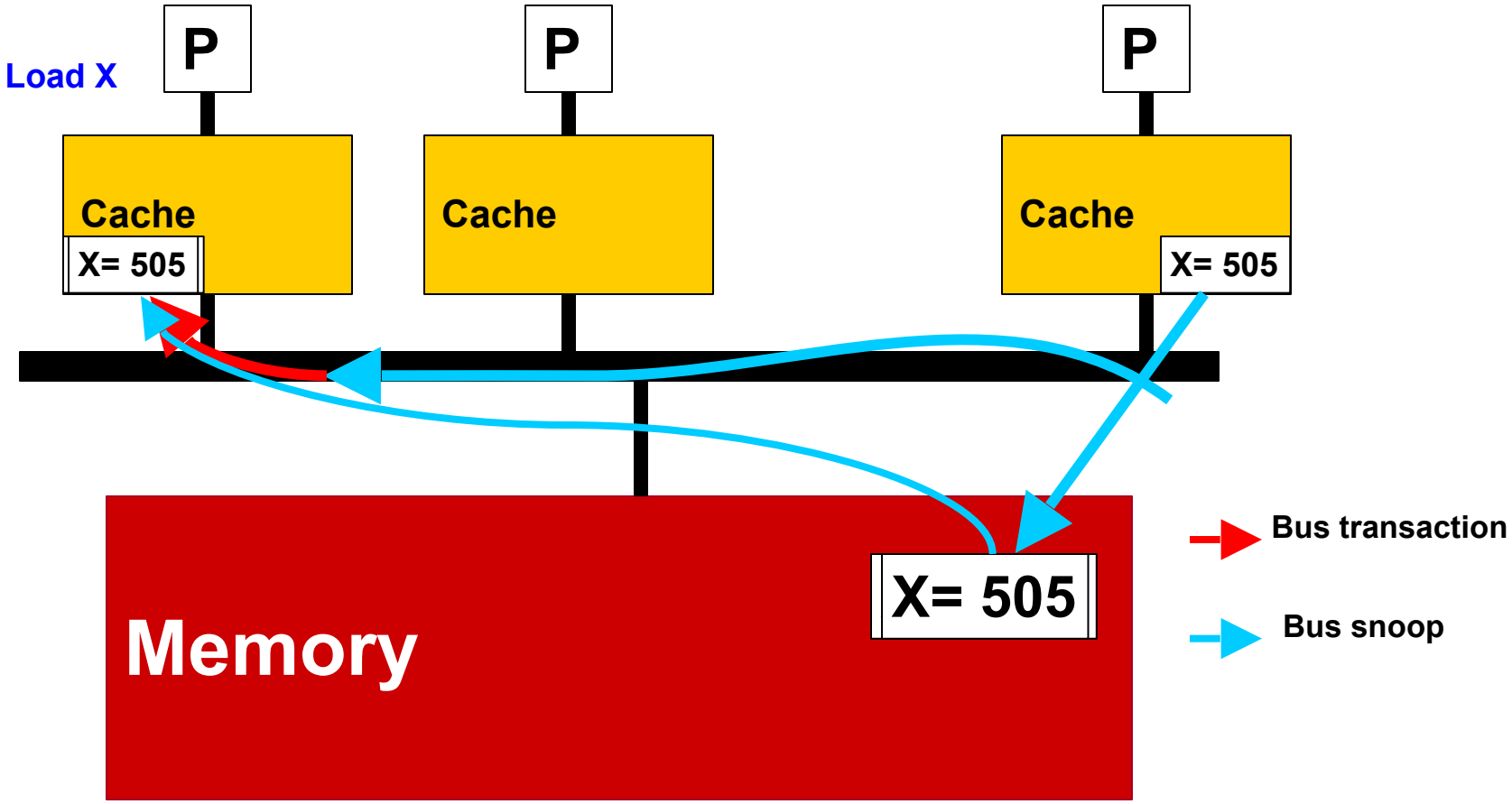
(Update-based Protocol on Write-Through cache)



- Each processor's cache controller constantly snoops on the bus
- Update local copies upon snoop hit

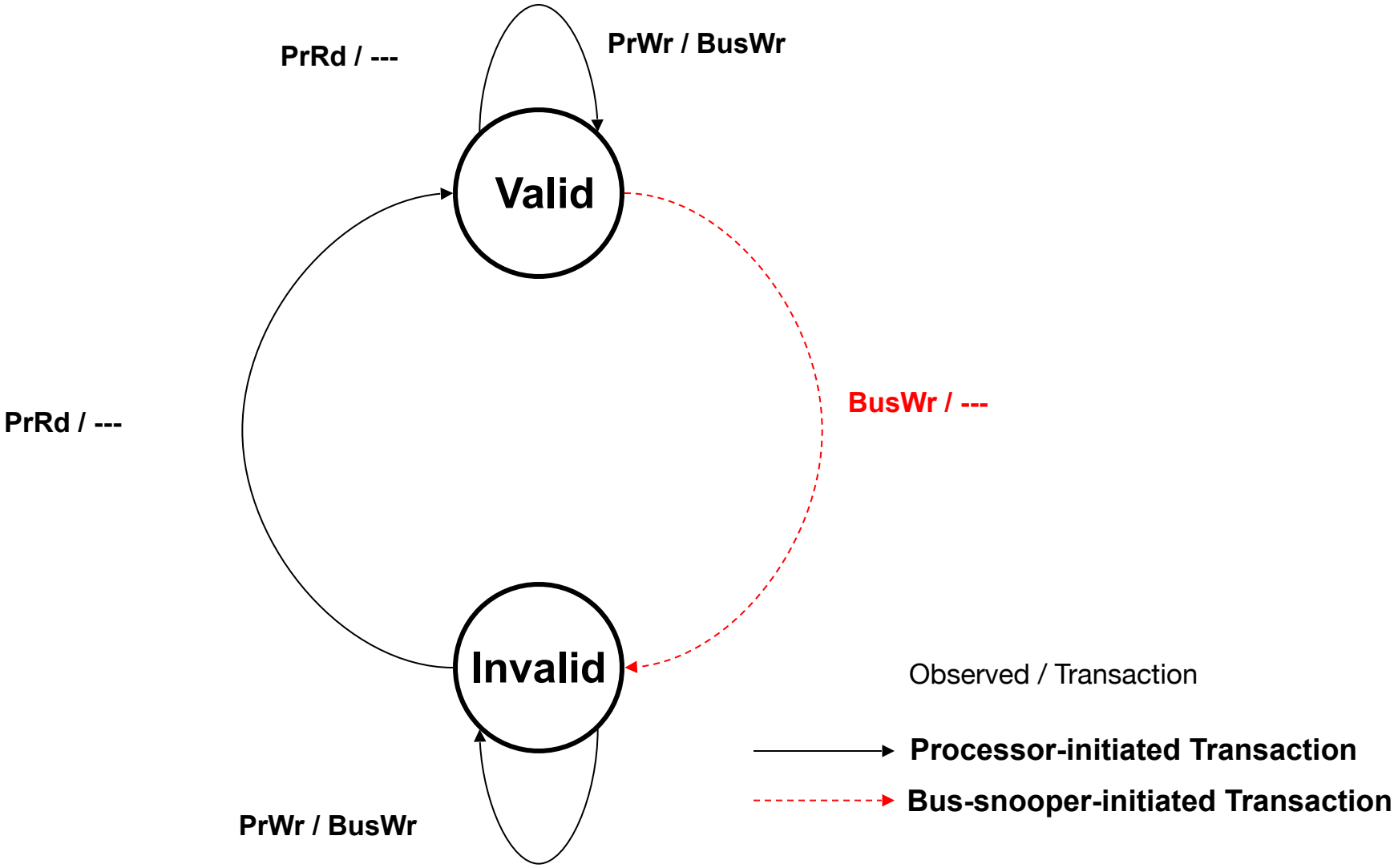
Bus Snooping

(Invalidation-based Protocol on Write-Through cache)

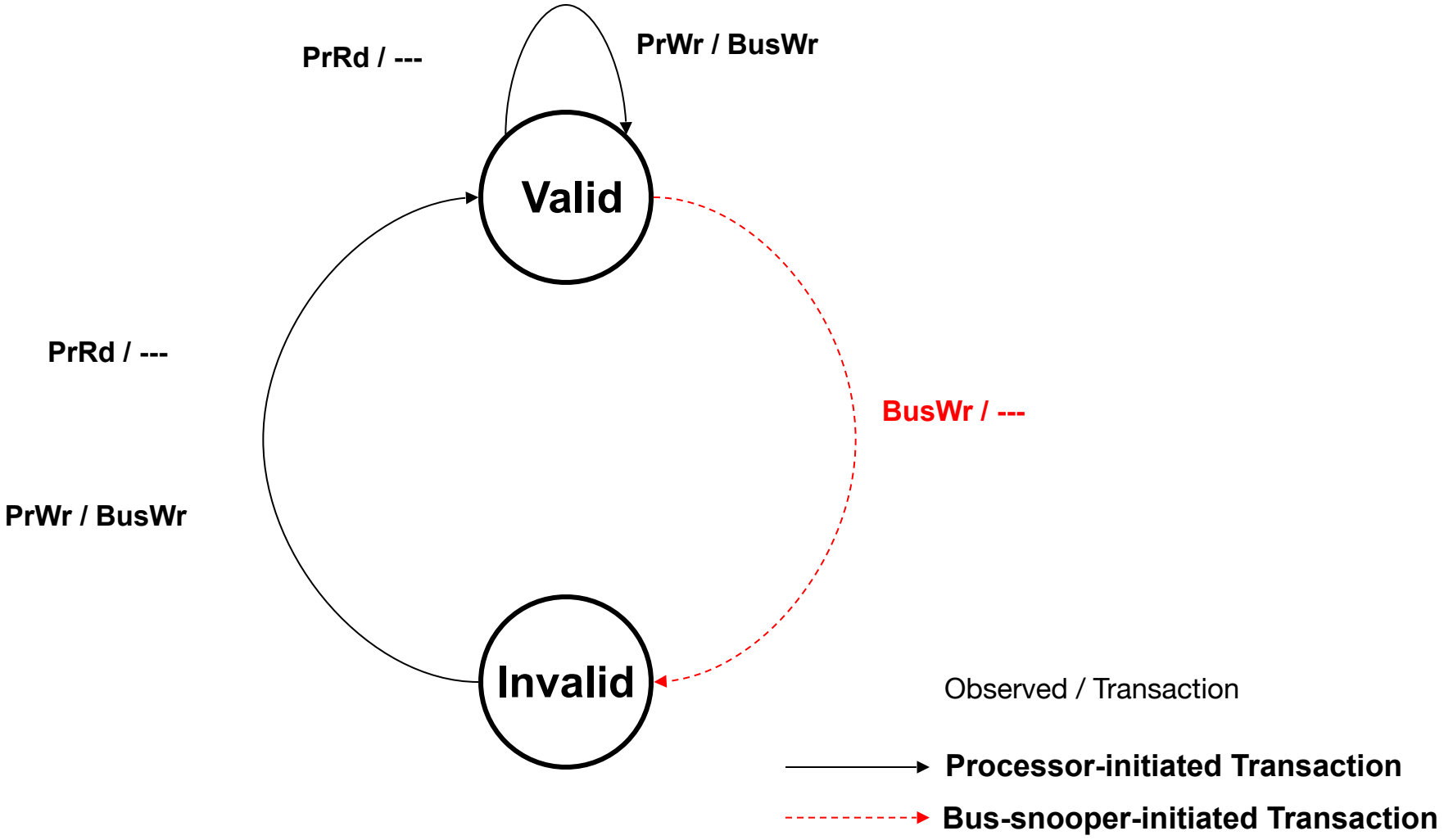


- Each processor's cache controller constantly snoops on the bus
- Invalidate local copies upon snoop hit

A Simple Invalidation-based Coherence Protocol for a WT, No Write-Allocate Cache

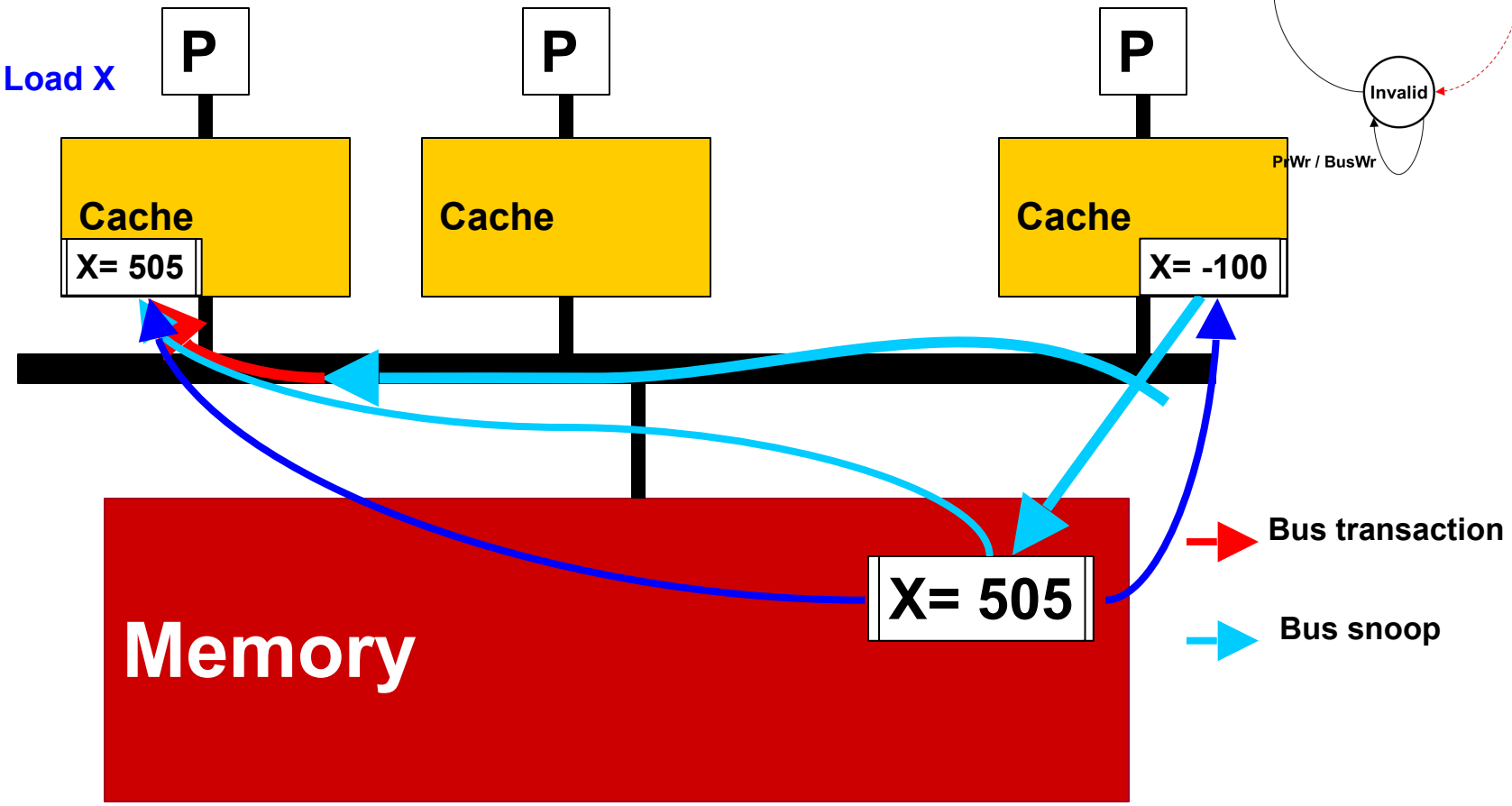


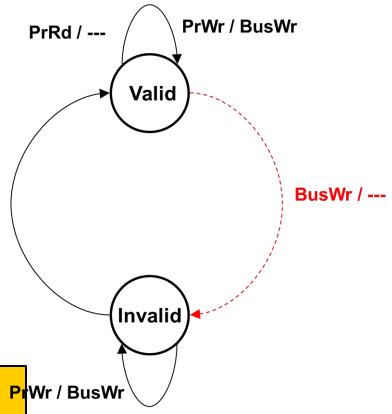
A Simple Invalidation-based Coherence Protocol for a WT, Write-Allocate Cache



Bus Snooping

(Invalidation-based Protocol on Write-Through cache)





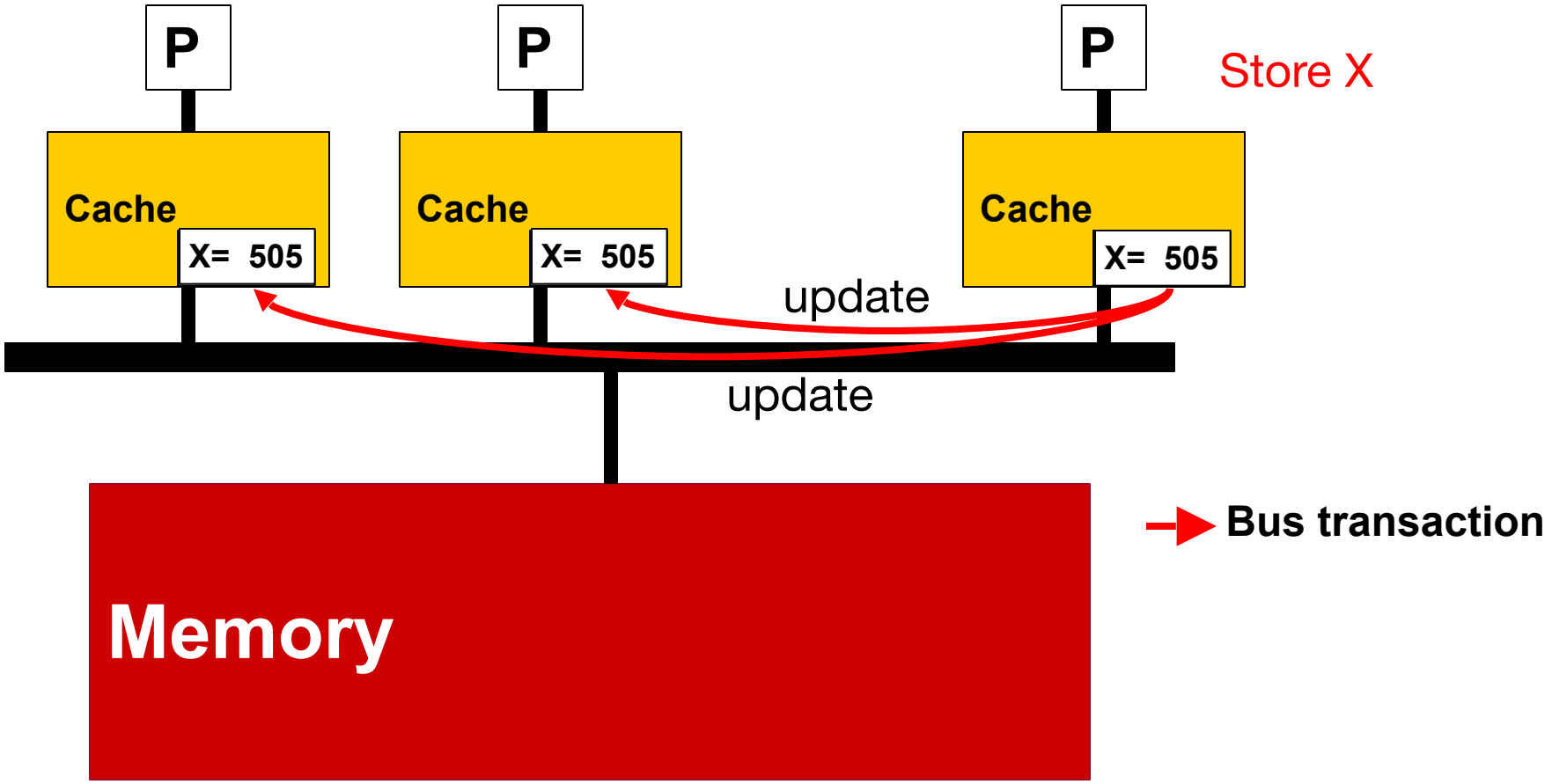
- Each processor's cache controller constantly snoops on the bus
- Invalidate local copies upon snoop hit

How about Writeback Cache?

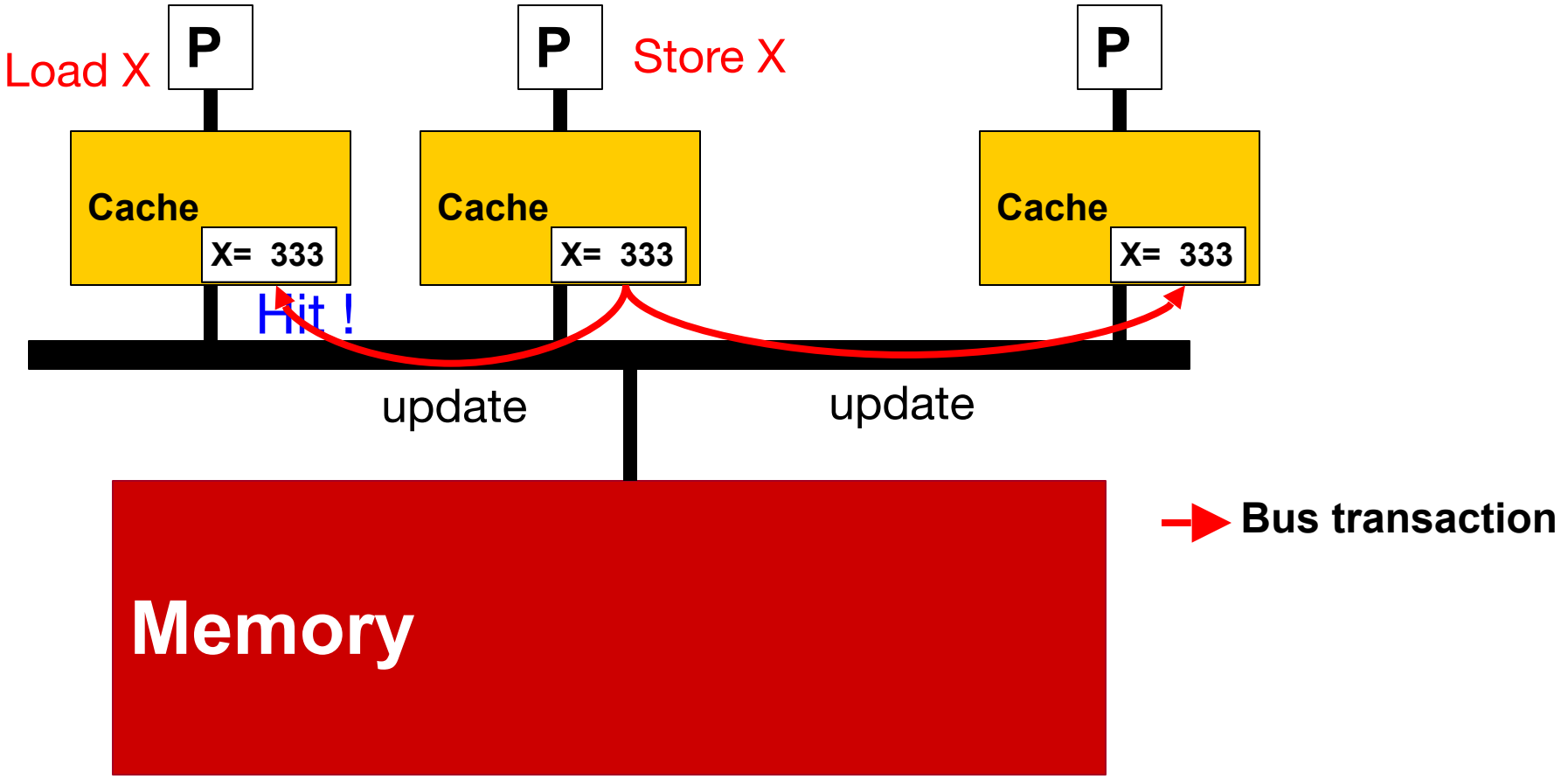
- Write-through wastes a lot of bus bandwidth (every update creates a traffic)
- WB cache to reduce bandwidth requirement
- The majority of local writes are hidden behind the processor nodes
- How to snoop?
- Write ordering

Cache Coherence Protocols for WB caches

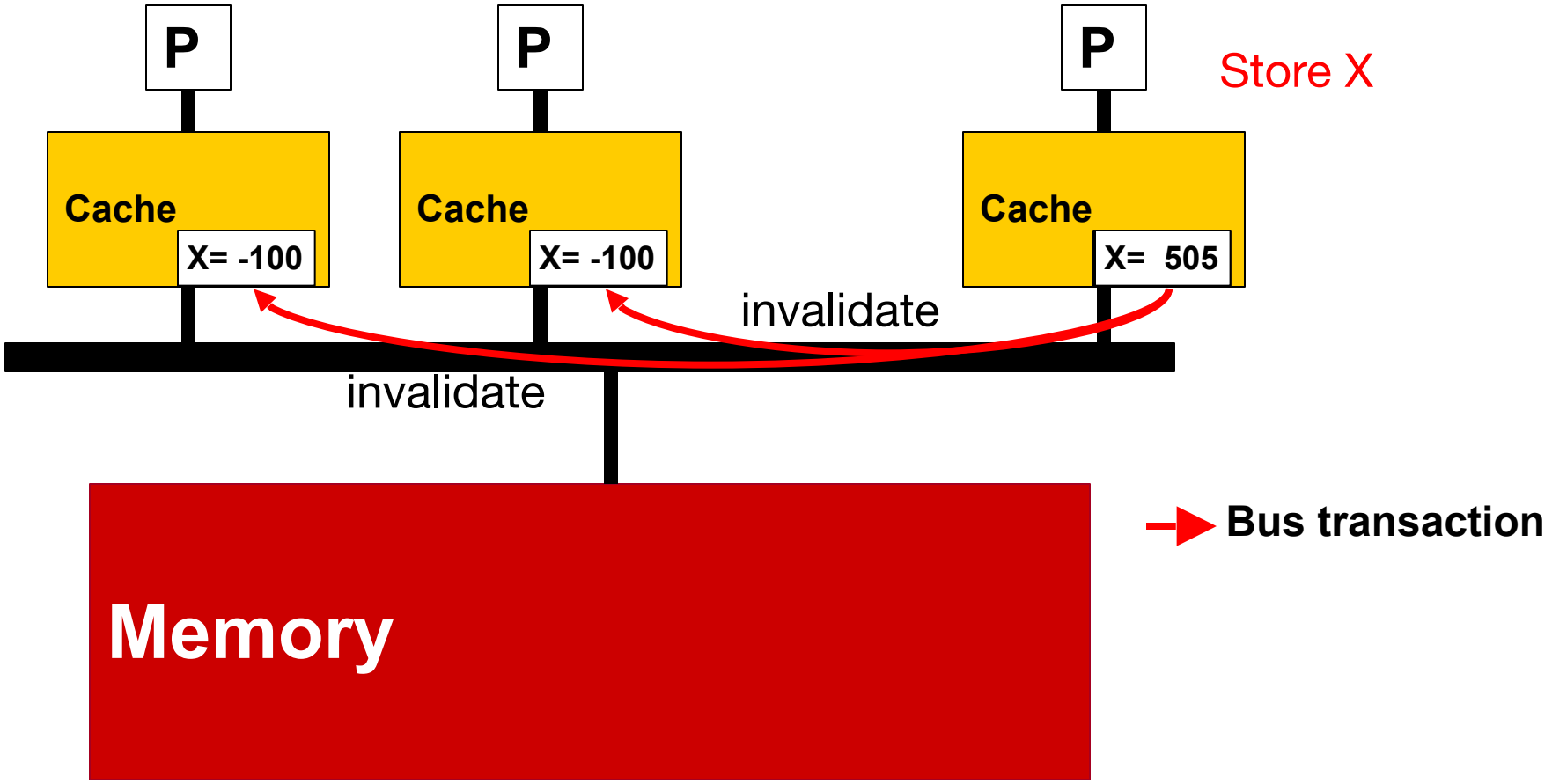
- A cache has an *exclusive* copy of a line if
  - It is the only cache having a valid copy
  - Memory may or may not have it
- Modified (dirty) cache line
  - The cache having the line is the *owner* of the line, because it must supply the block



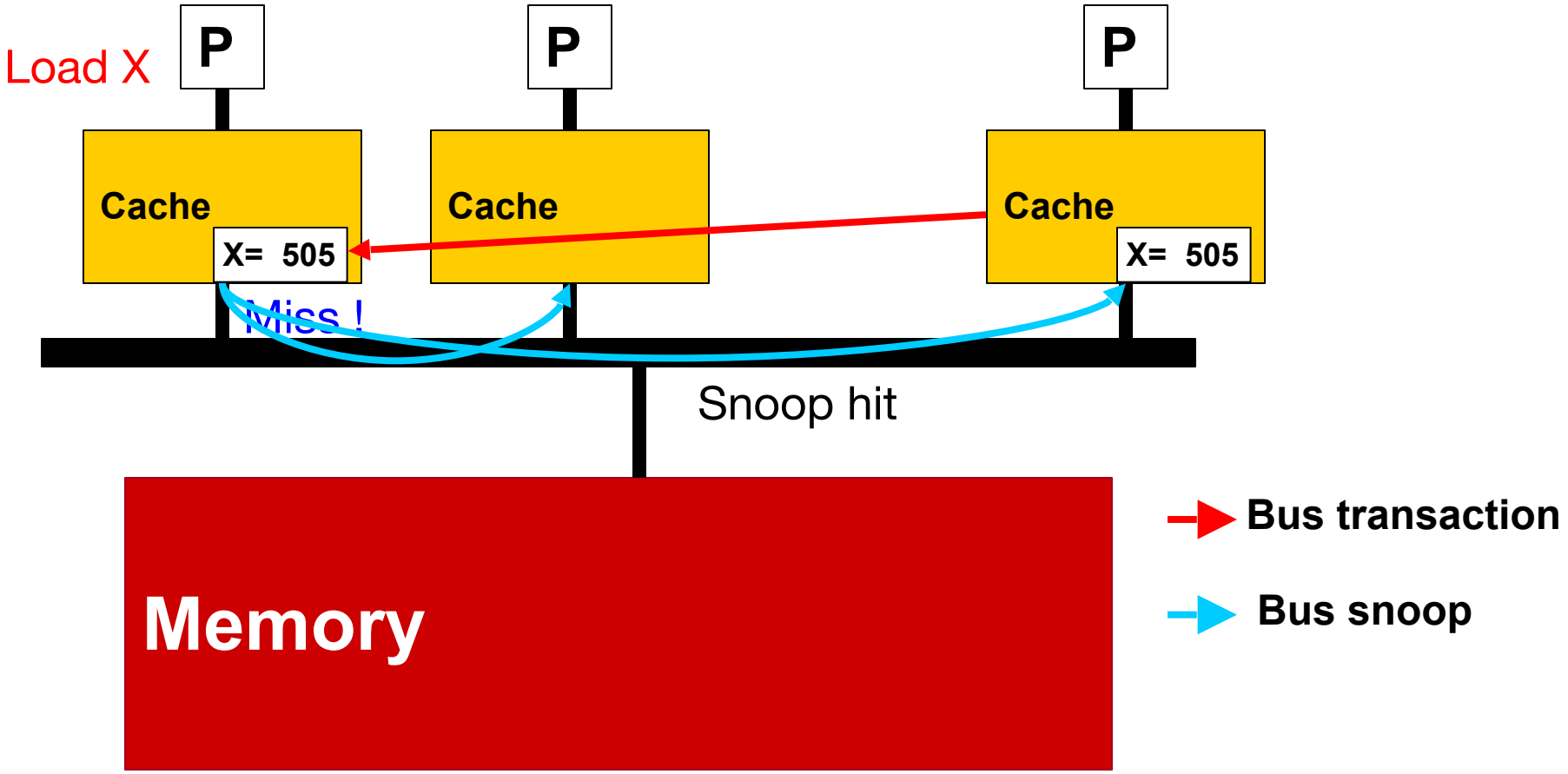
- Update data for all processor nodes who share the same data
- For a processor node keeps updating the memory location, a lot of traffic will be incurred



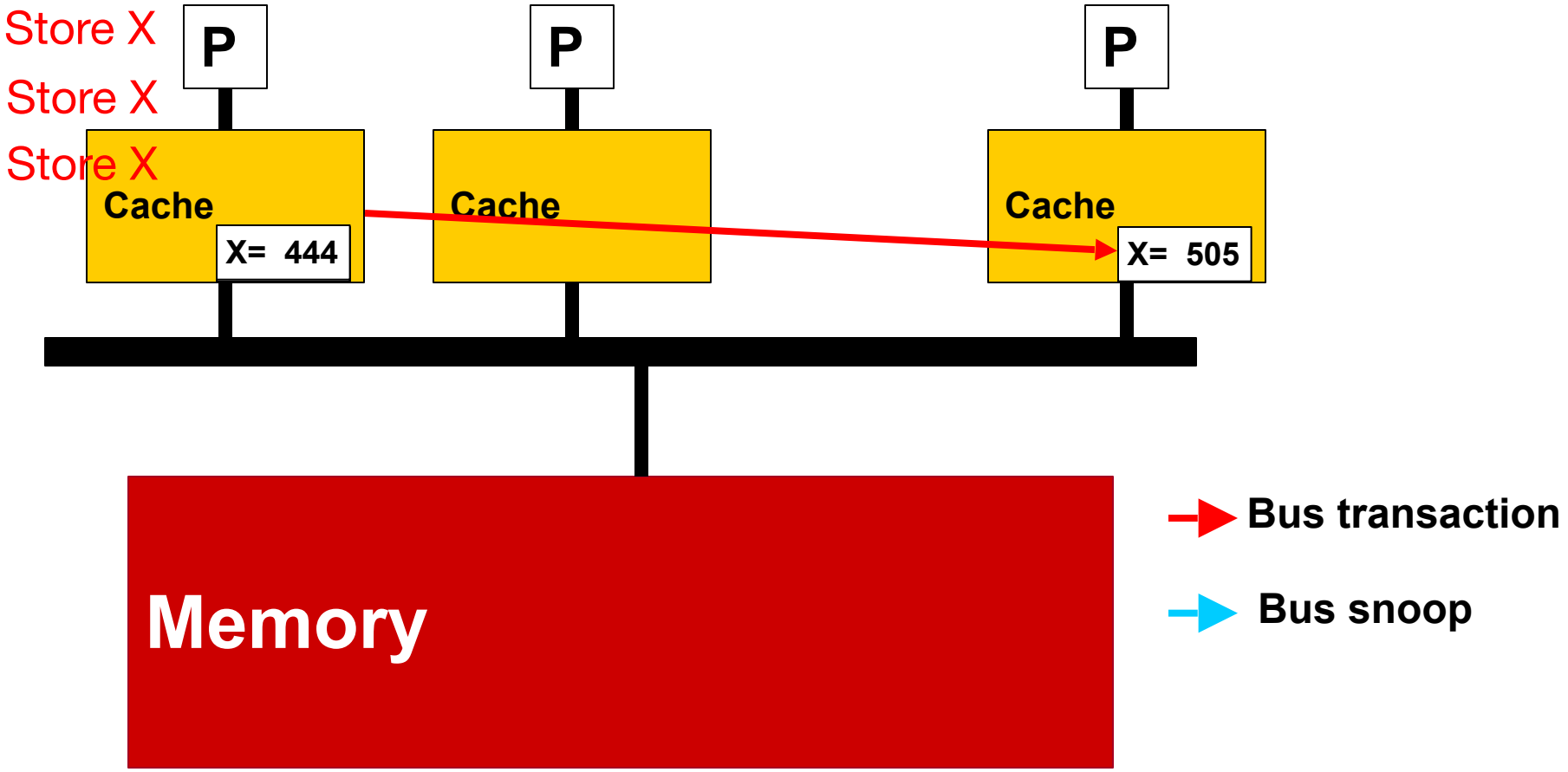
- Update data for all processor nodes who share the same data
- For a processor node keeps updating the memory location, **a lot of traffic** will be incurred



- Invalidate the data copies for the sharing processor nodes
- Reduced traffic when a processor node keeps updating the same memory location



- Invalidate the data copies for the sharing processor nodes
- Reduced traffic when a processor node keeps updating the same memory location



- Invalidate the data copies for the sharing processor nodes
- Reduced traffic when a processor node keeps updating the same memory location

MSI Writeback Invalidation Protocol

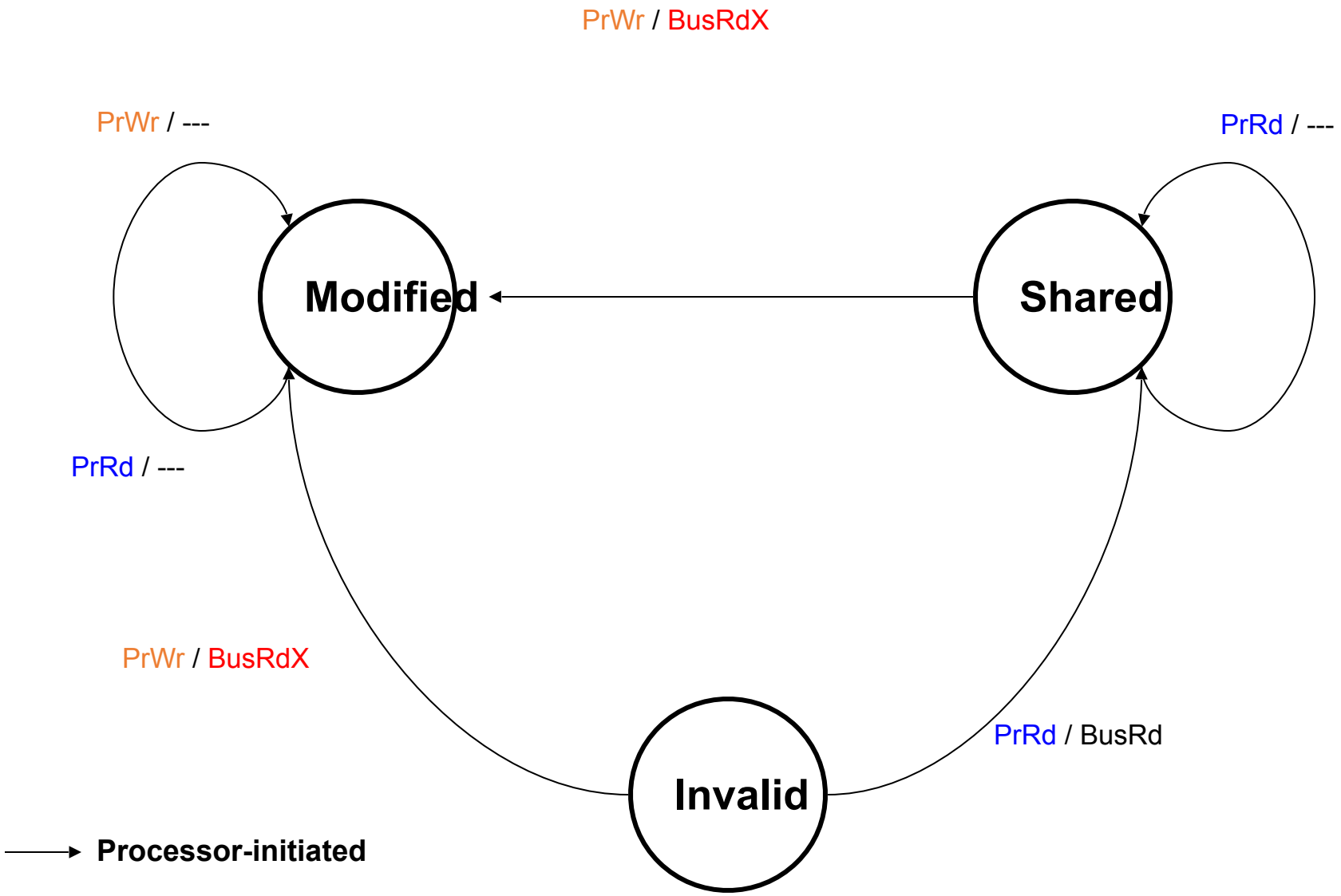
- **M**odified
  - Dirty
  - Only this cache has a valid copy
- **S**hared
  - Memory is consistent
  - One or more caches have a valid copy
- **I**nvalid

- Writeback protocol: A cache line can be written multiple times before the memory is updated.

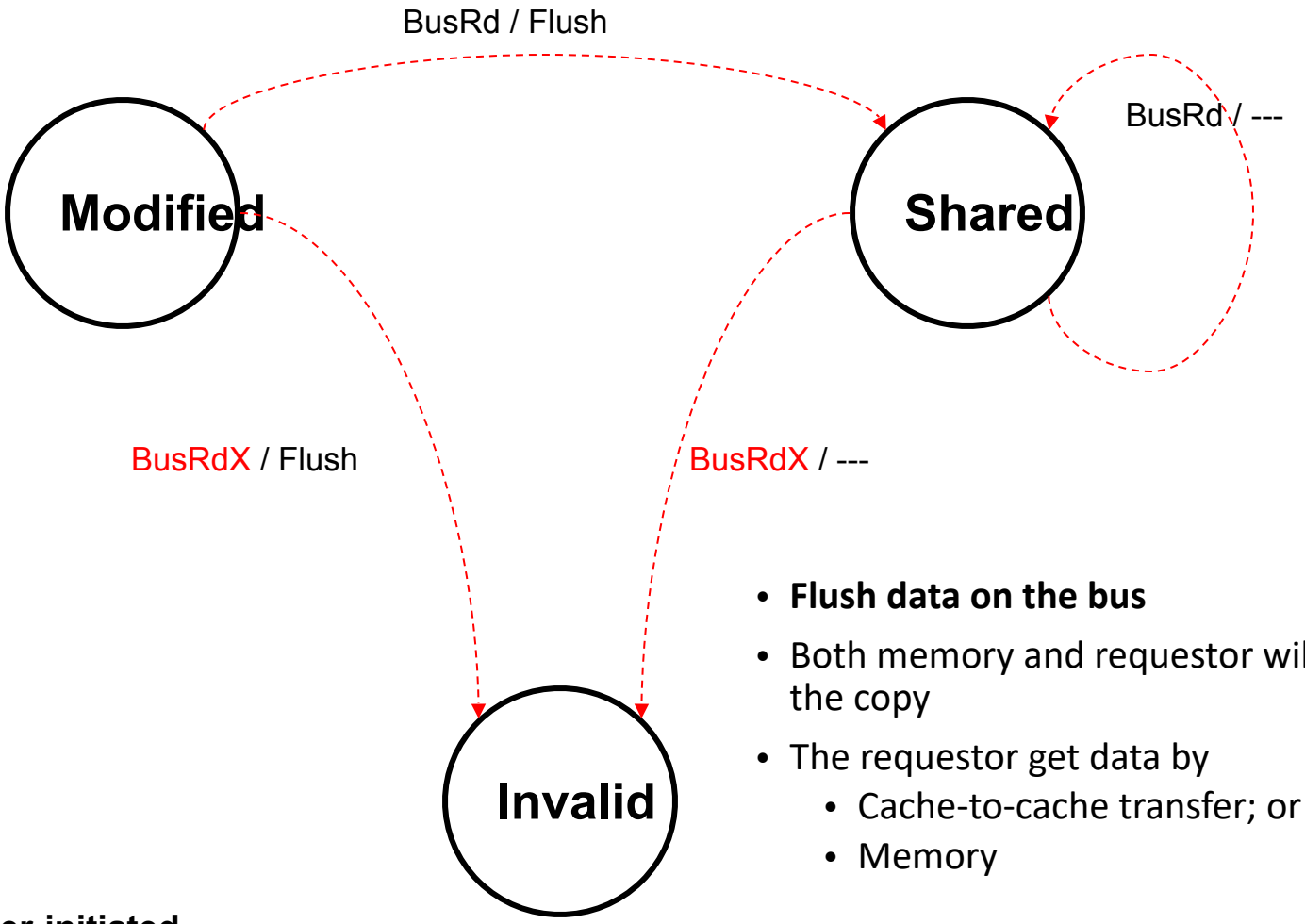
MSI Writeback Invalidation Protocol

- Two types of request from the **processor**
  - PrRd
  - PrWr
- Three types of **bus transactions** post by cache controller
  - **BusRd**
    - PrRd misses the cache
    - Memory or another cache supplies the line
  - **BusRd** e**X**clusive (Read-to-own)
    - PrWr is issued to a line which is **not** in the **M**odified state
  - **BusWB**
    - Writeback due to replacement
    - Processor does not directly involve in initiating this operation

MSI Writeback Invalidation Protocol (Processor Request)



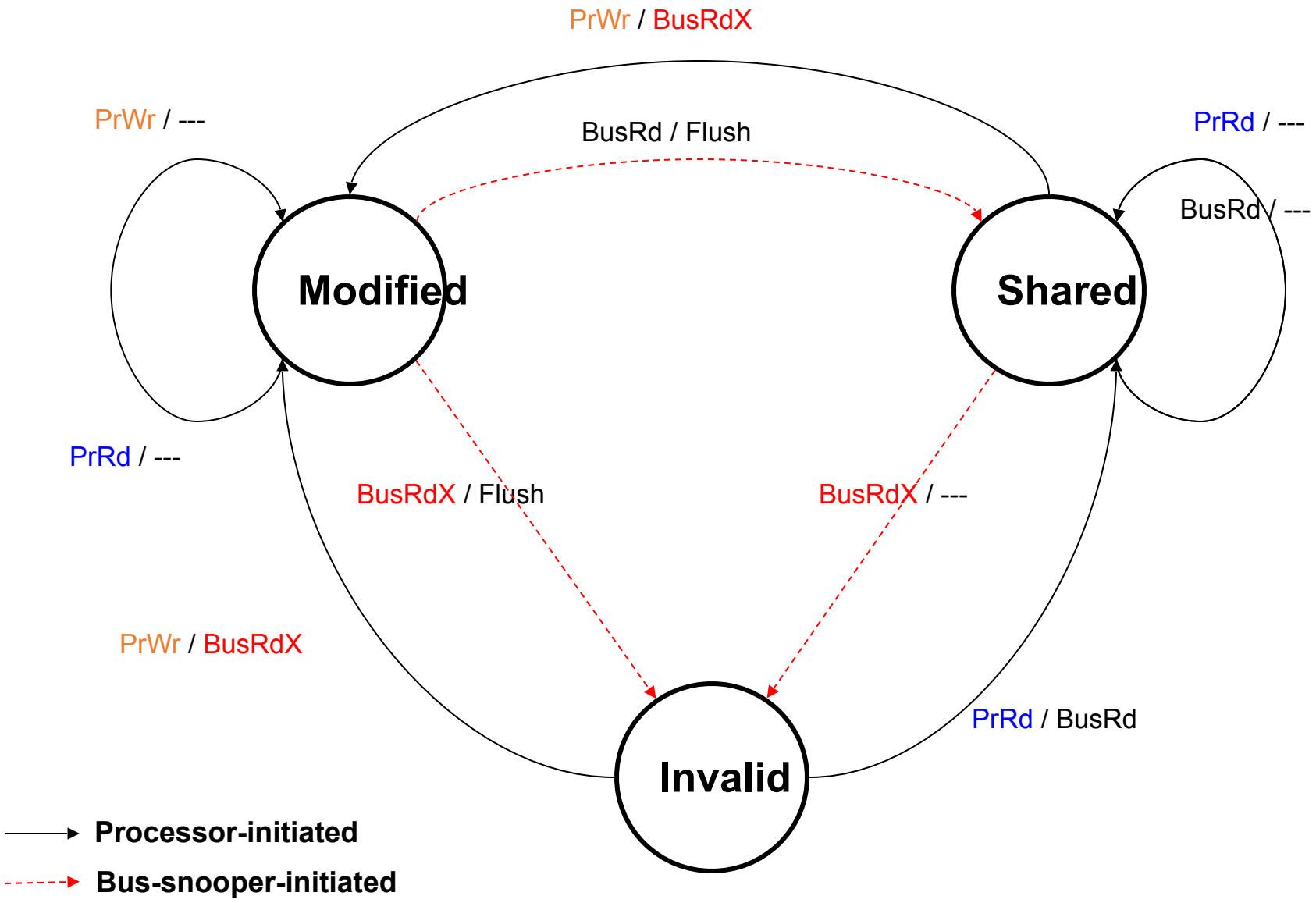
MSI Writeback Invalidation Protocol (Bus Transaction)



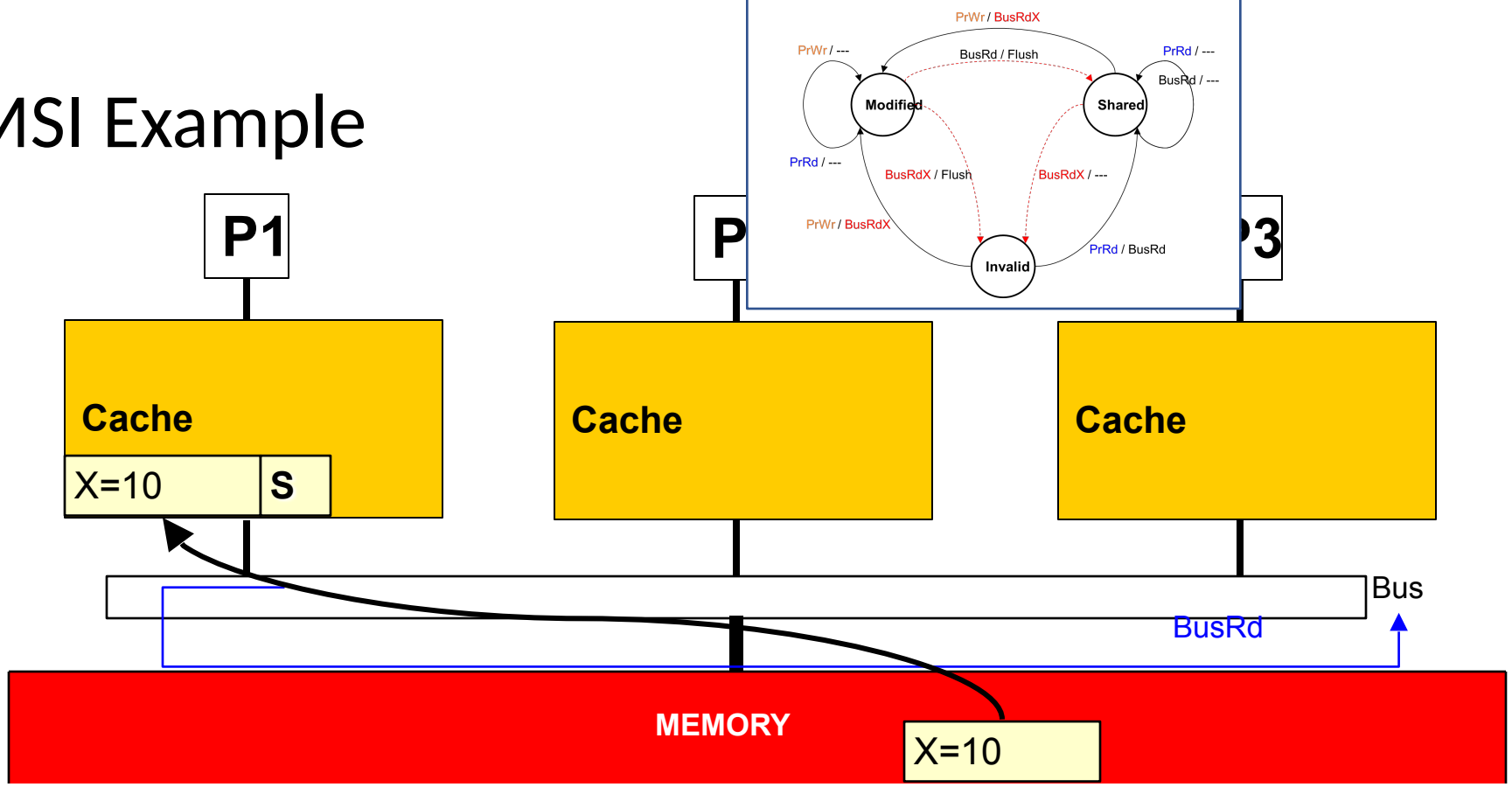
- **Flush data on the bus**
- Both memory and requestor will grab the copy
- The requestor get data by
  - Cache-to-cache transfer; or
  - Memory

Bus-snooper-initiated

MSI Writeback Invalidation Protocol

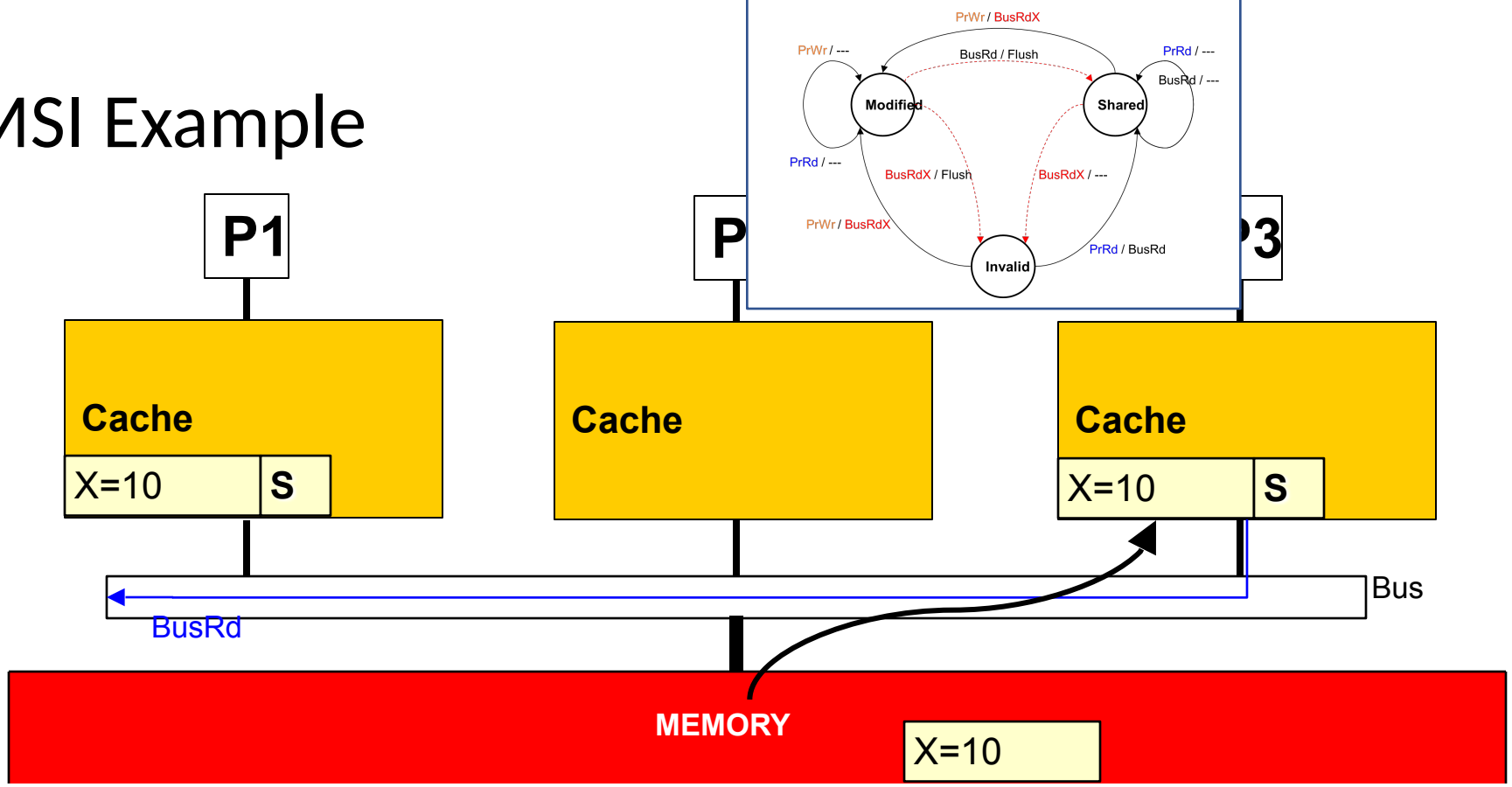


MSI Example



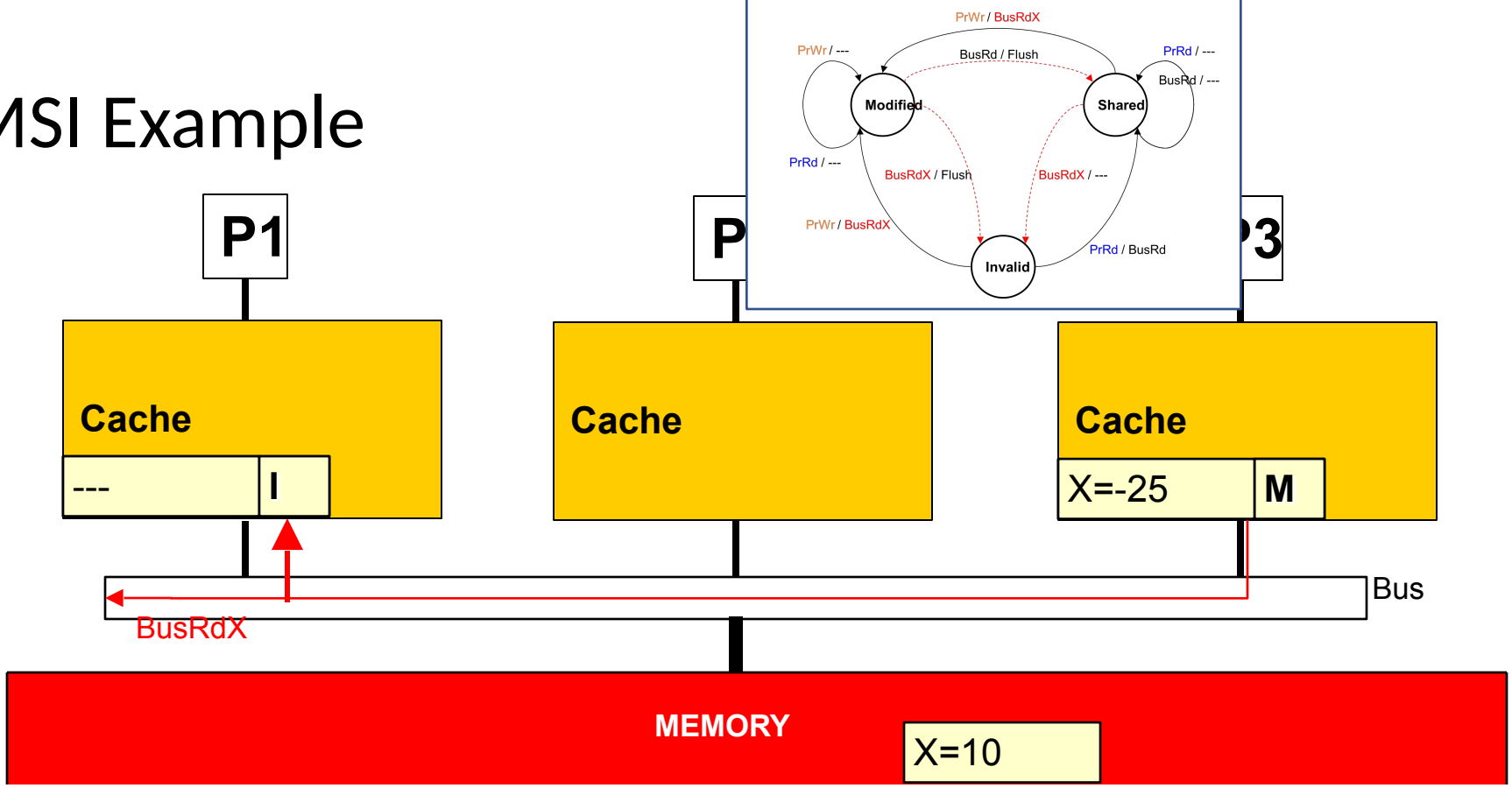
Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory

MSI Example



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory

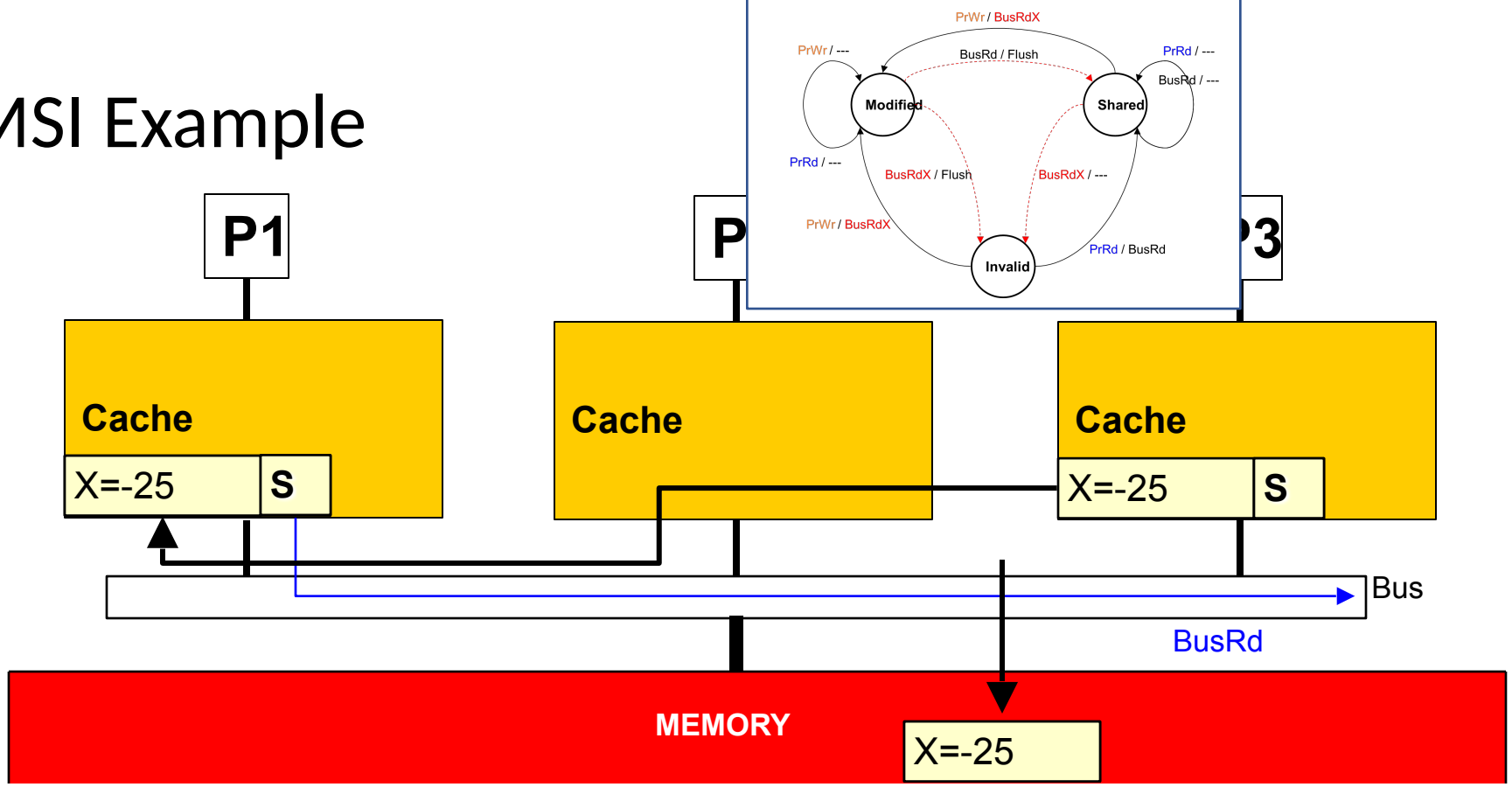
MSI Example



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory
P3 writes X	I	---	M	BusRdX	P3 Cache

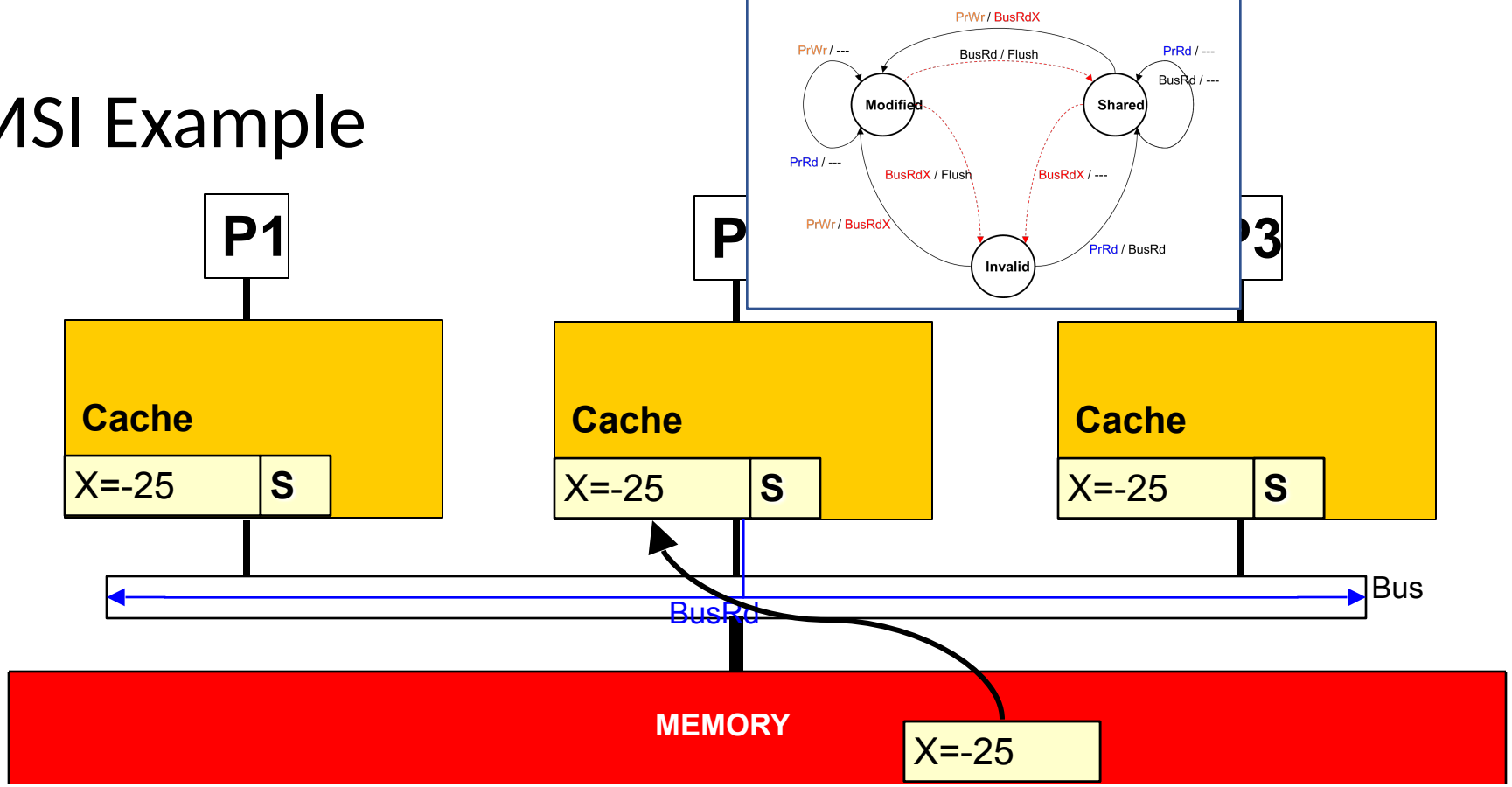
Does not come from memory if having "BusUpgrade"

MSI Example



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory
P3 writes X	I	---	M	BusRdX	P3 Cache
P1 reads X	S	---	S	BusRd	P3 Cache

MSI Example



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory
P3 writes X	I	---	M	BusRdX	P3 Cache
P1 reads X	S	---	S	BusRd	P3 Cache
P2 reads X	S	S	S	BusRd	Memory

MSI Summary

- MSI ensures single-writer-multiple-reader
  - Often called the SWMR (pronounced as “swimmer”) invariant
- We can still do better
  - MESI, MOESI, ….
  - I will not go into further detail.