# CS230: Digital Logic Design and Computer Architecture

## Lecture 23: DRAM Controller and Cache Coherence

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html

https://www.cse.iitb.ac.in/~biswa/

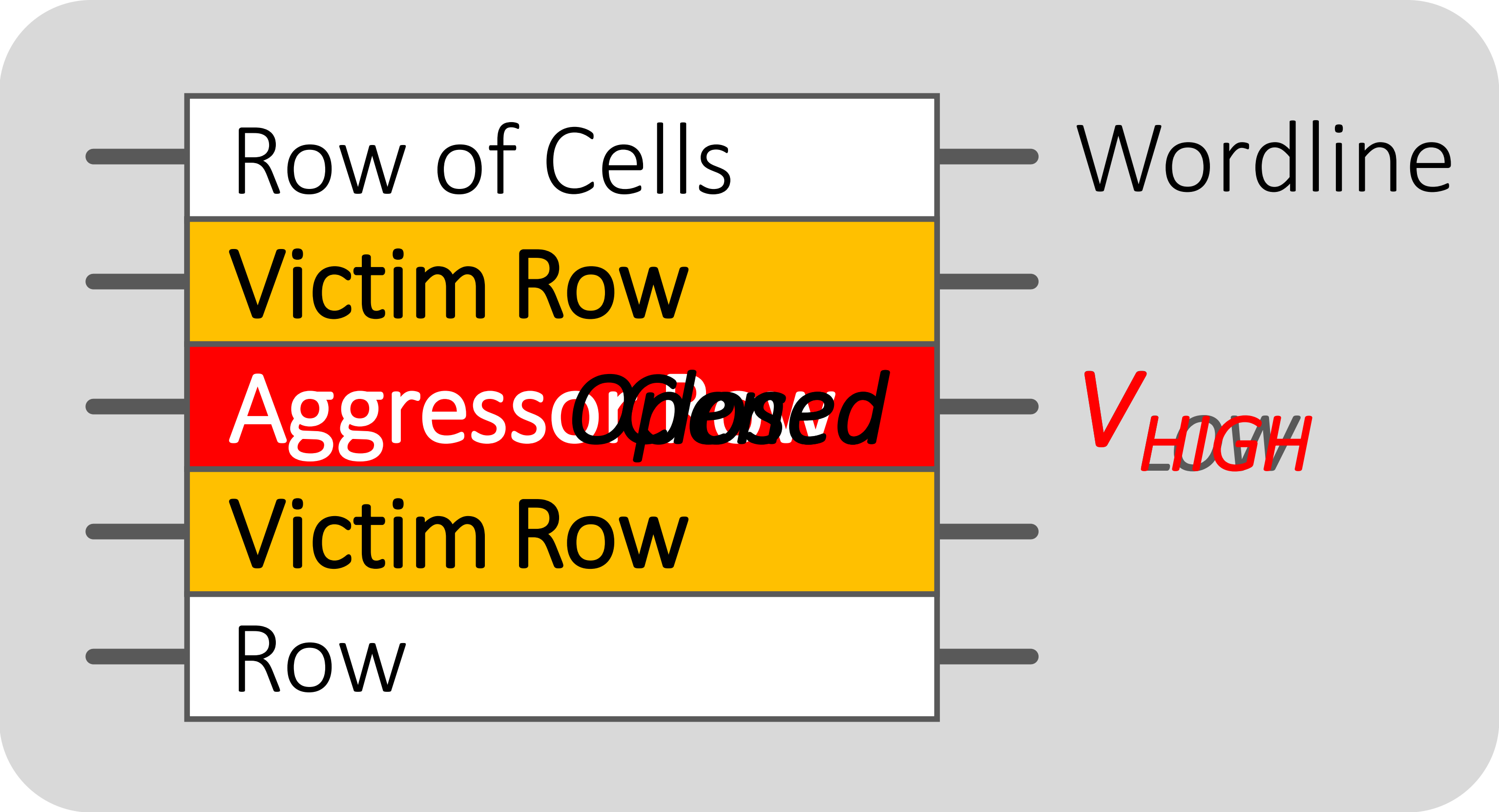
# 10K view on the latency

Page Empty: ACT + CAS

Page Hit: CAS

Page Miss: PRE+ACT+CAS

# Rowhammer



Row of Cells

Wordline

Victim Row

Aggressor Row

Opened

Closed

$V_{LOW}$

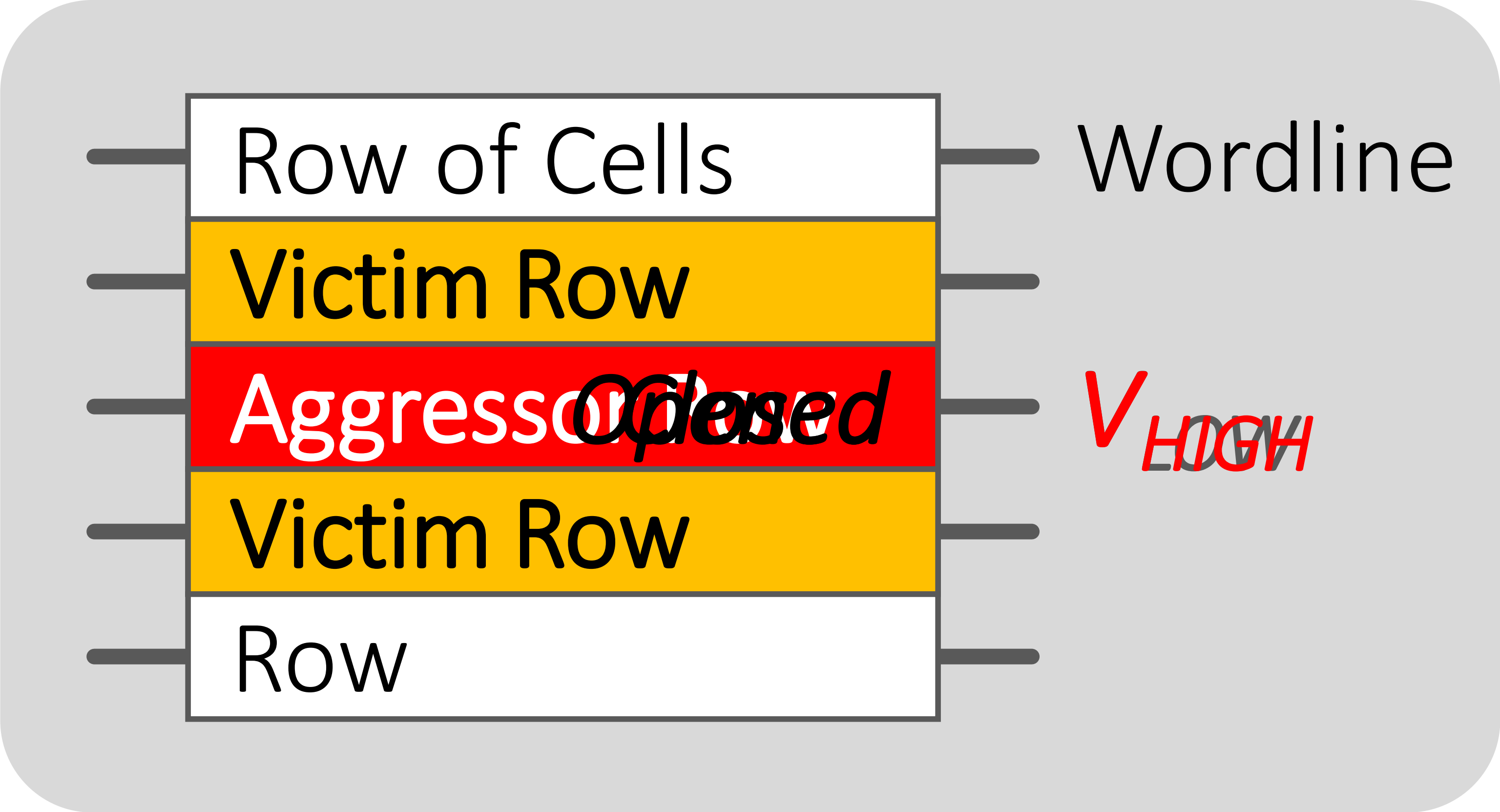
$V_{HIGH}$

Victim Row

Row

*Repeatedly opening and closing a row induces* ***disturbance errors*** *in adjacent rows*

# Rowhammer

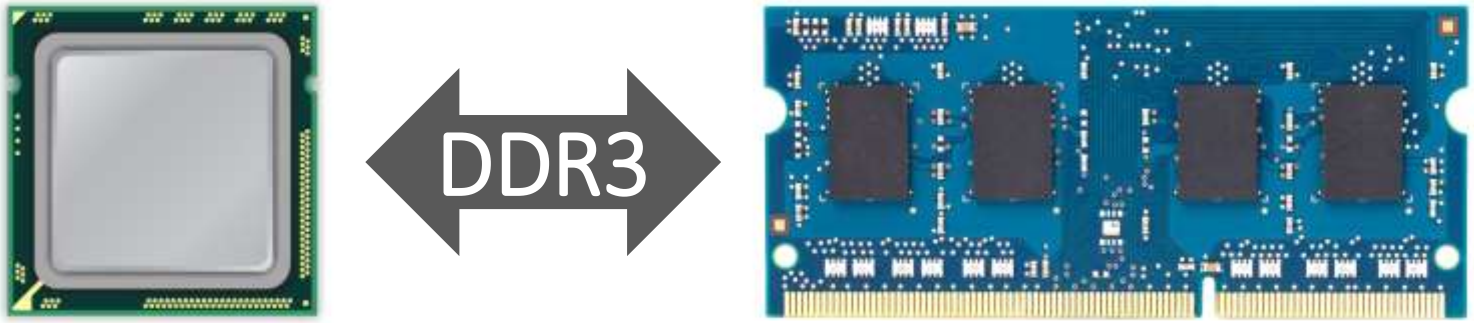


*"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice*

# The Code Please

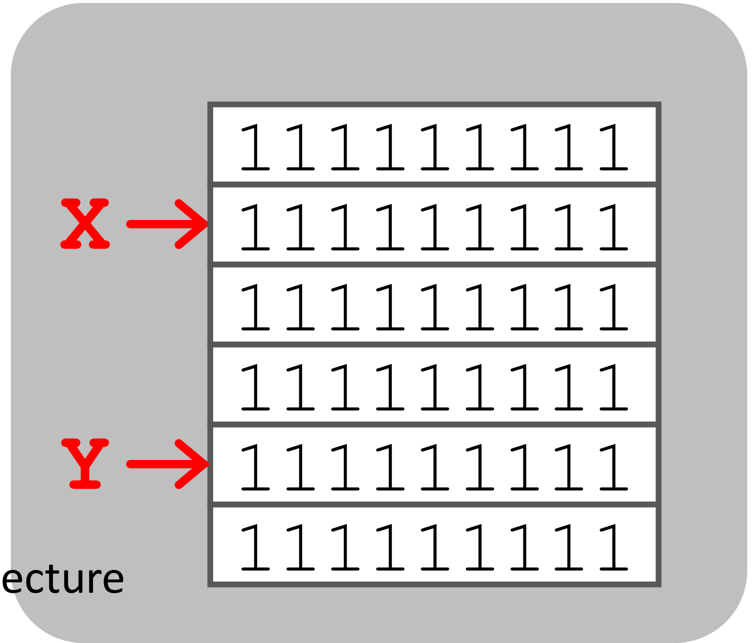
x86 CPU

DRAM Module



1. Avoid *cache hits*
   - Flush **X** from cache

2. Avoid *row hits* to **X**
   - Read **Y** in another row



111111111

X → 111111111

111111111

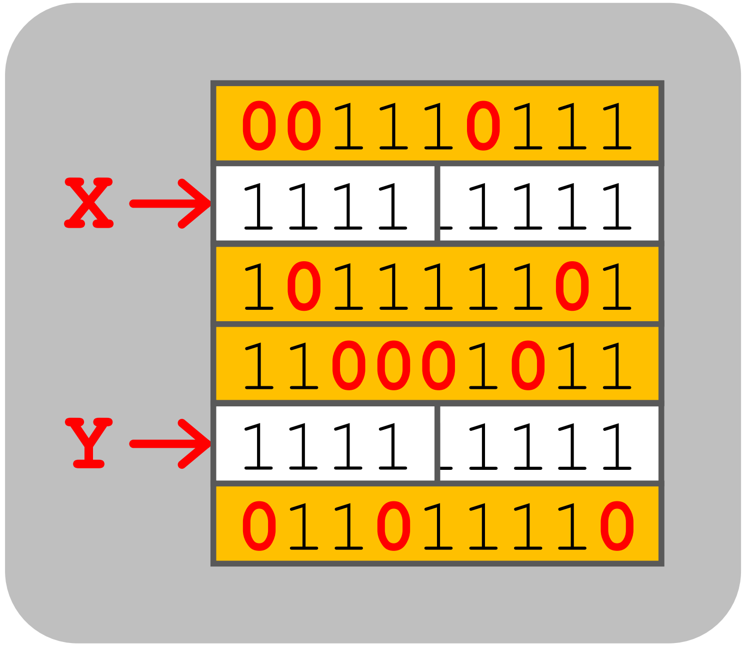
111111111

Y → 111111111

111111111

# The code please

```
loop:
   mov  (X), %eax
   mov  (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



# Solution:

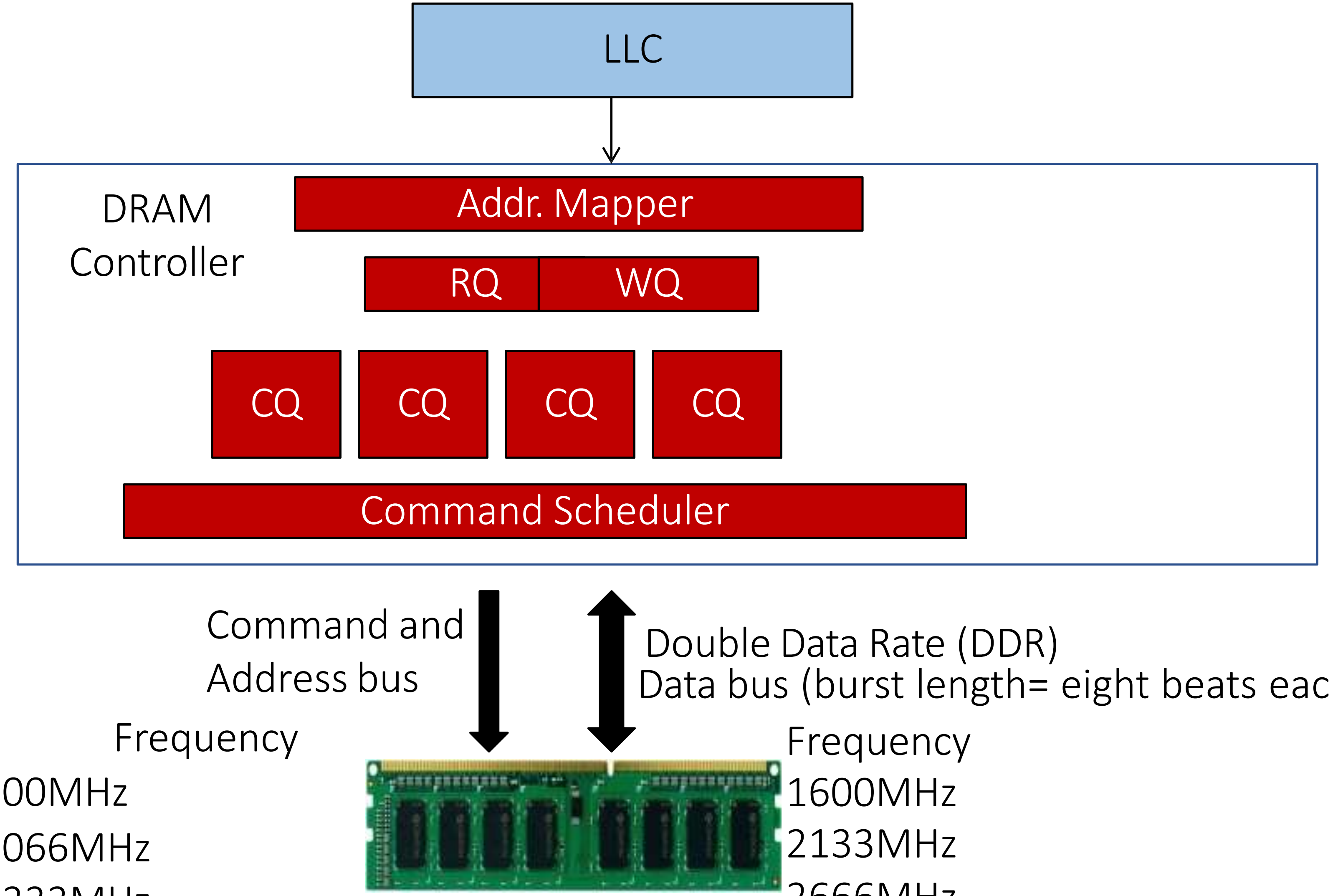
# DRAM Refresh

- But intelligently ☺

- Baseline: Once in 63ms

- Need for a Rowhammer tracker, and then mitigator

# DRAM Controller



LLC

DRAM Controller

Addr. Mapper

RQ

WQ

CQ

CQ

CQ

CQ

Command Scheduler

Command and Address bus

Double Data Rate (DDR) Data bus (burst length= eight beats each of 64-bits)

Frequency
800MHz
1066MHz
1333MHz

Frequency
1600MHz
2133MHz
2666MHz

# Reads and Writes(Writebacks)

Reads are critical to performance

Write Queue stores writes and the writes are serviced after # writes reach a threshold

*Why?*

The direction of the data bus changes from reads to writes. So ??

DRAM controller creates DRAM commands from based on the requests at read Q and write Q

# DRAM Scheduling

Based on
Row-buffer locality,
Source of the request,
Loads/Stores
Load criticality

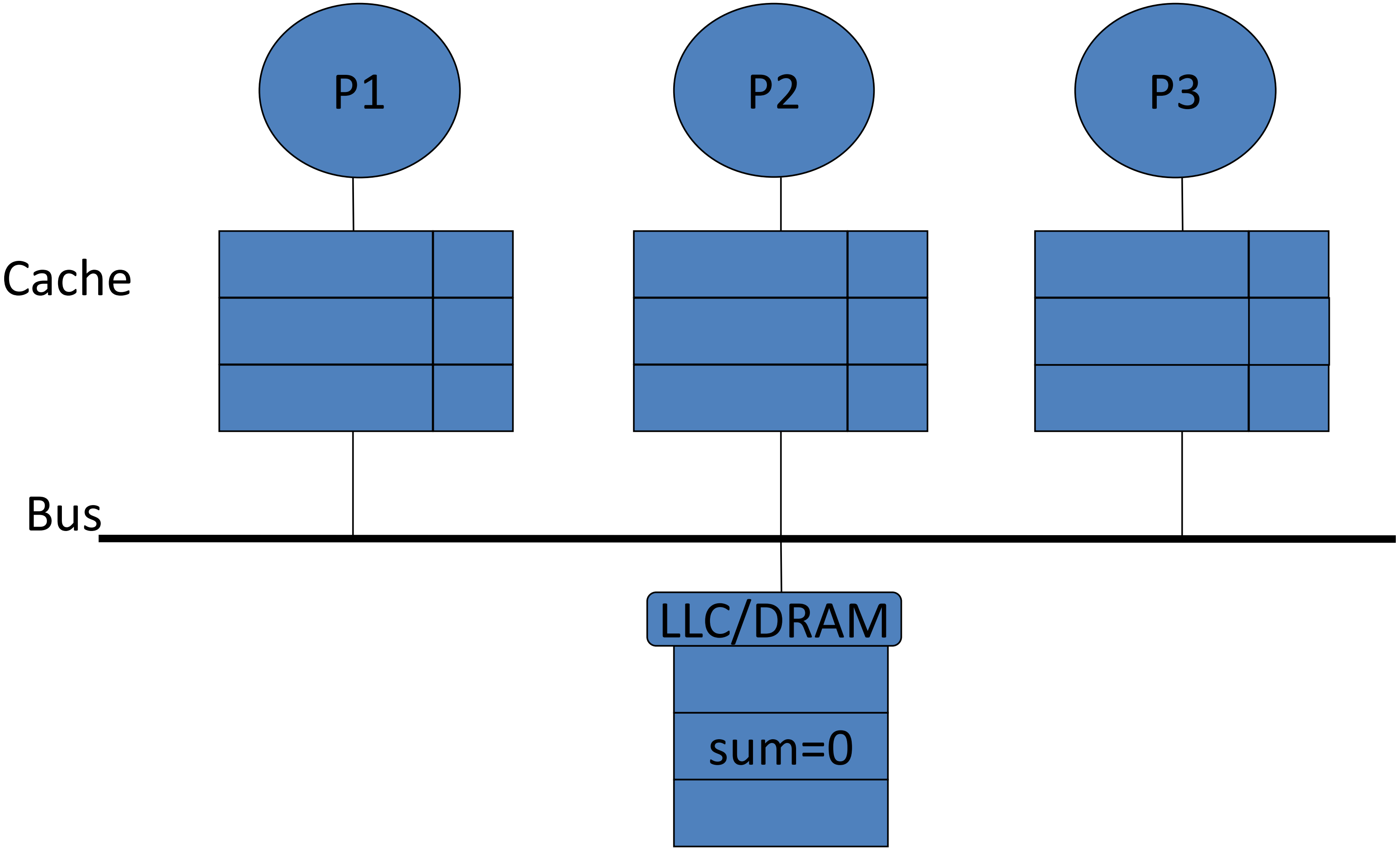
Satisfy all the timing constraints. Around 60

FR-FCFS

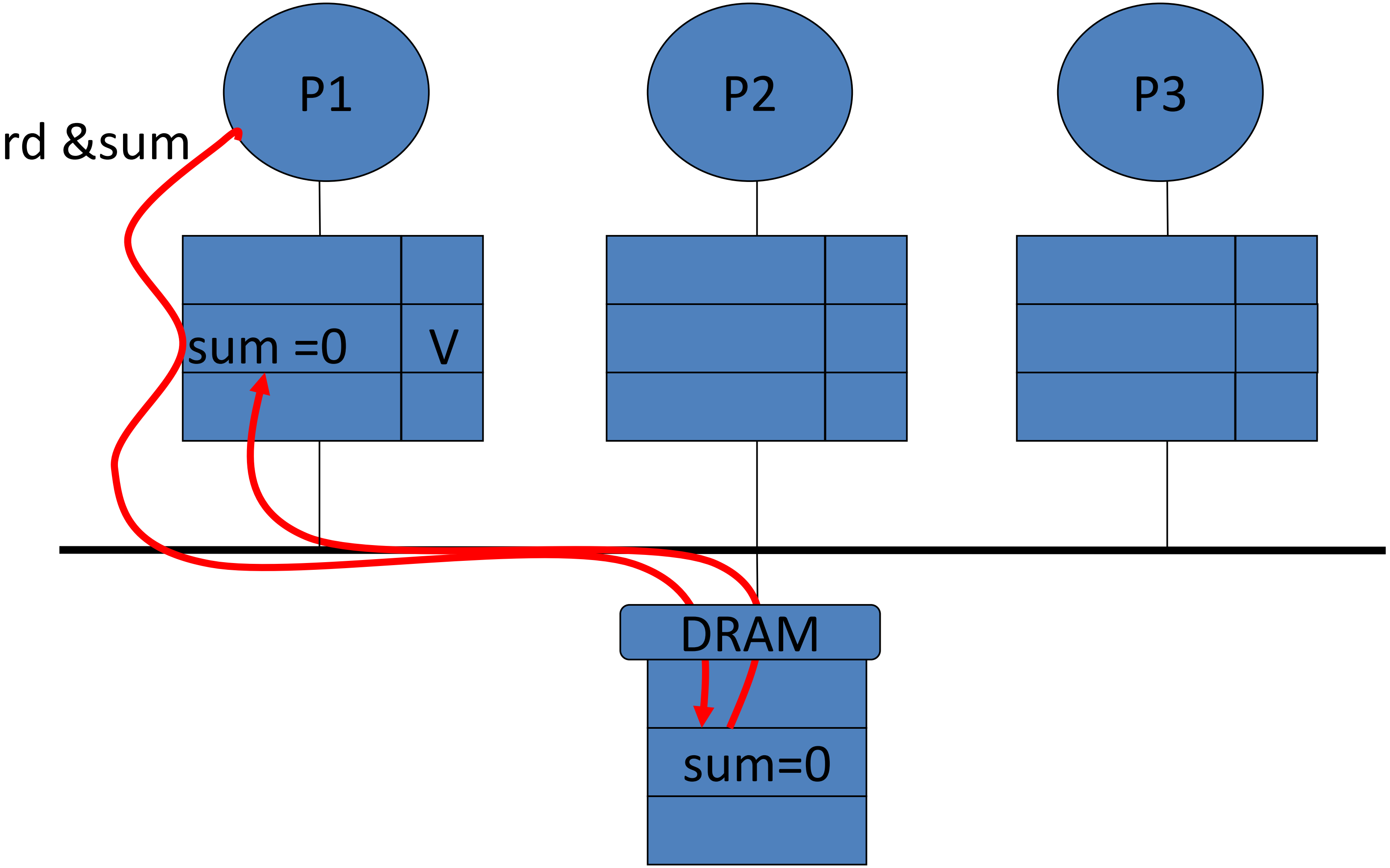
Prefers requests with Row hits (column-first) FR: First Ready

| Names | Memory clock | I/O bus clock | Transfer rate | Theoretical bandwidth |
|---|---|---|---|---|
| DDR-200, PC-1600 | 100 MHz | 100 MHz | 200 MT/s | 1.6 GB/s |
| DDR-400, PC-3200 | 200 MHz | 200 MHz | 400 MT/s | 3.2 GB/s |
| DDR2-800, PC2-6400 | 200 MHz | 400 MHz | 800 MT/s | 6.4 GB/s |
| DDR3-1600, PC3-12800 | 200 MHz | 800 MHz | 1600 MT/s | 12.8 GB/s |
| DDR4-2400, PC4-19200 | 300 MHz | 1200 MHz | 2400 MT/s | 19.2 GB/s |
| DDR4-3200, PC4-25600 | 400 MHz | 1600 MHz | 3200 MT/s | 25.6 GB/s |
| DDR5-4800, PC5-38400 | 300 MHz | 2400 MHz | 4800 MT/s | 38.4 GB/s |
| DDR5-6400, PC5-51200 | 400 MHz | 3200 MHz | 6400 MT/s | 51.2 GB/s |

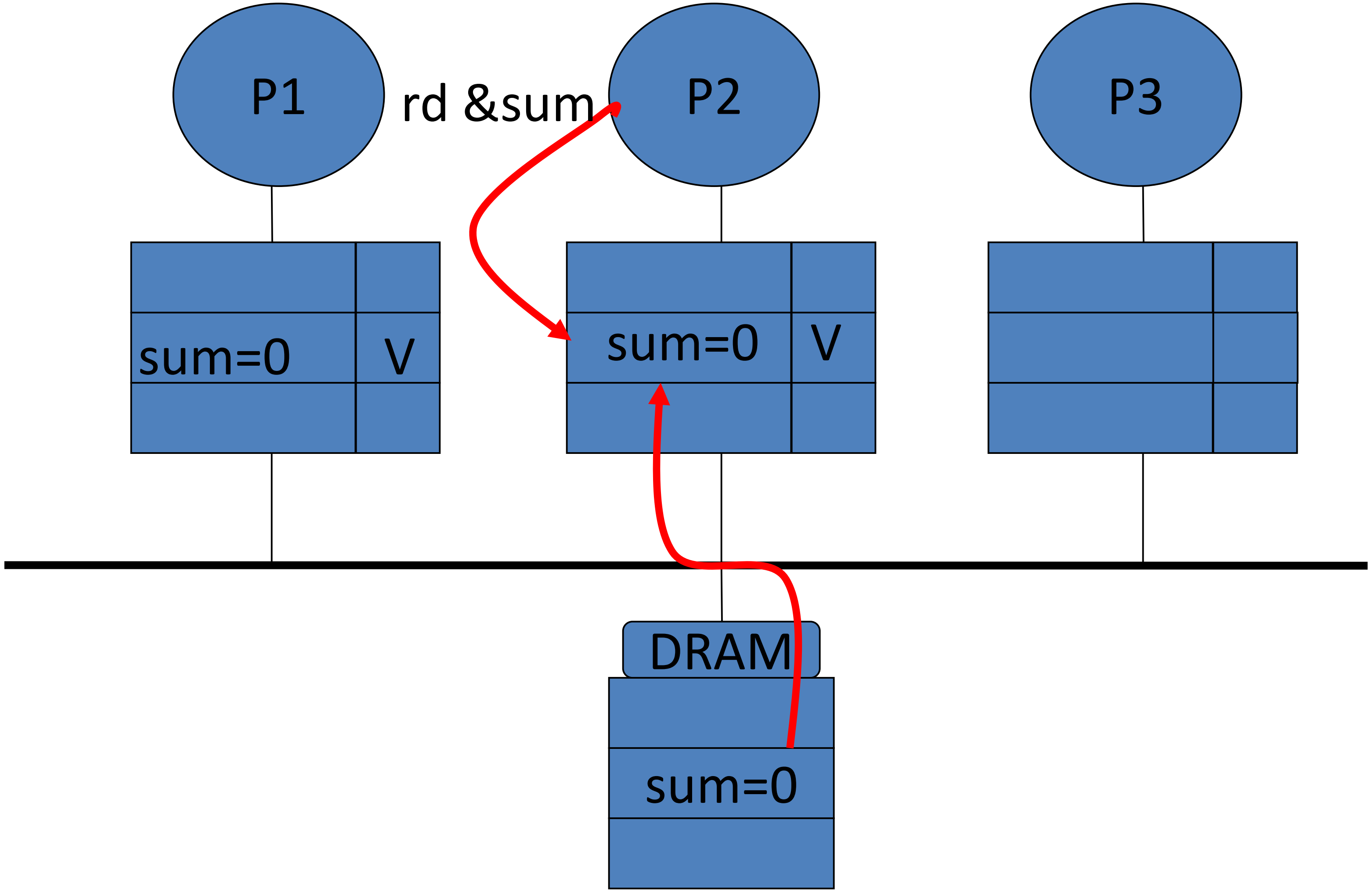
# Cache Coherence



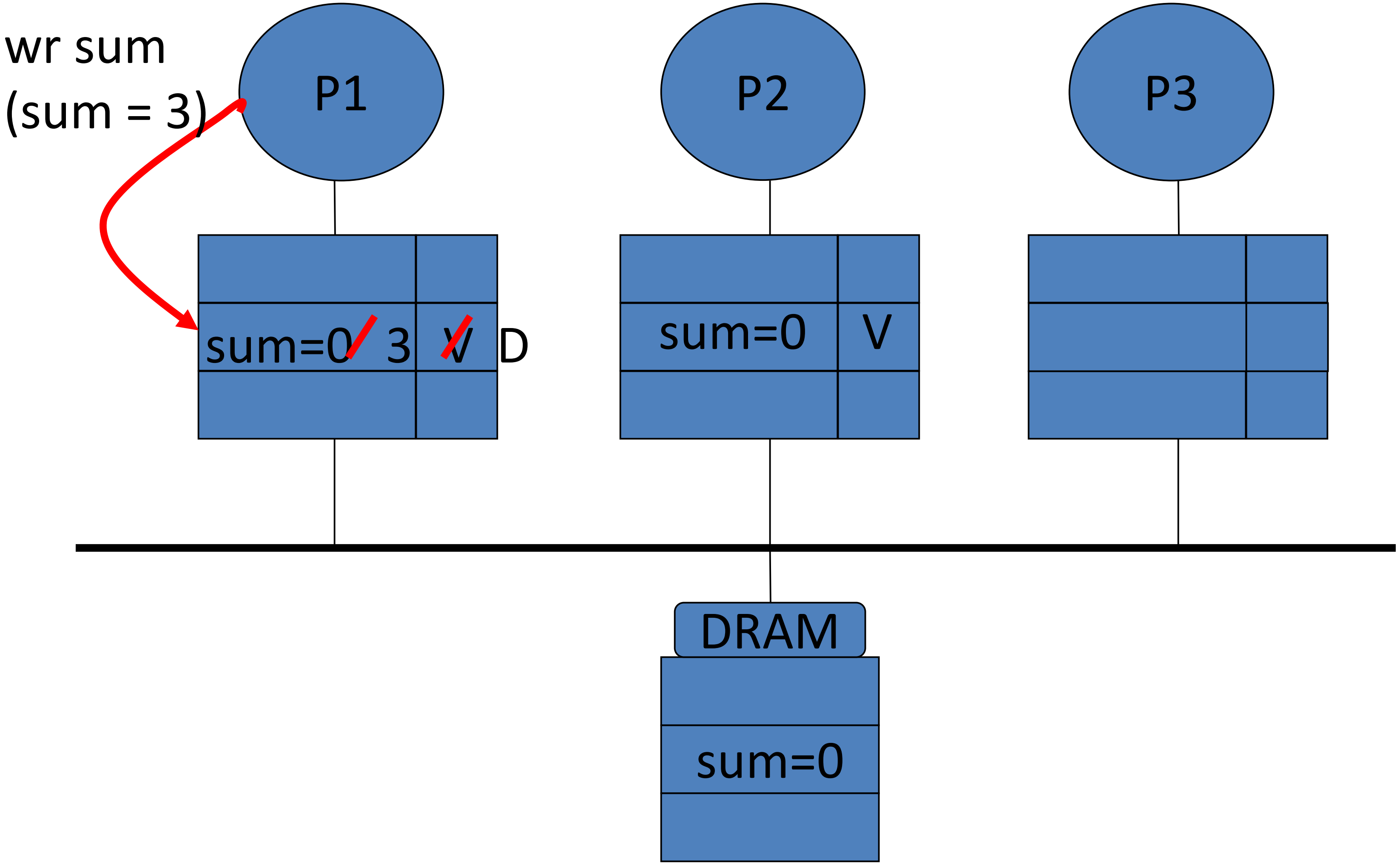
# Cache Coherence



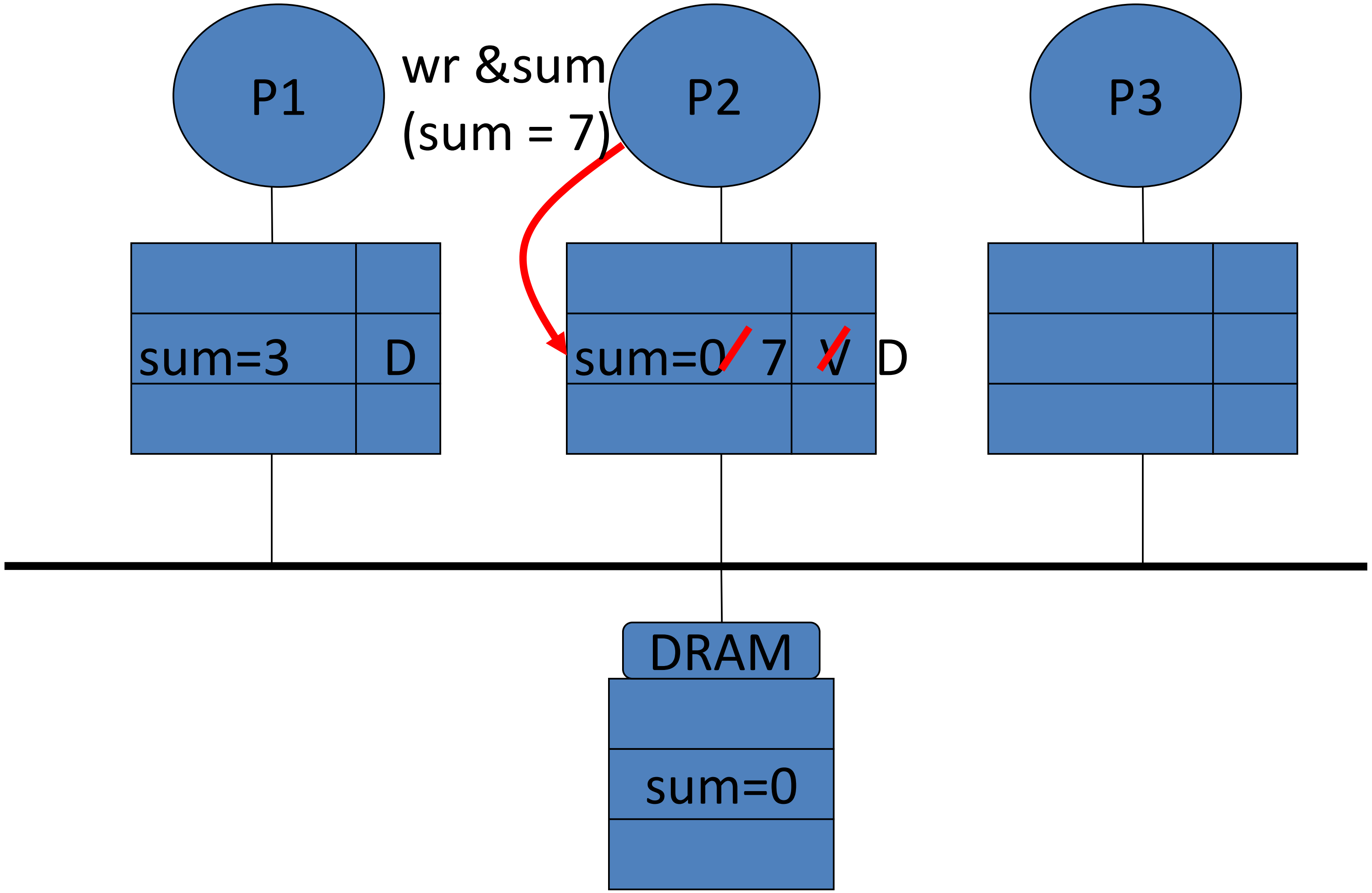
# Cache Coherence



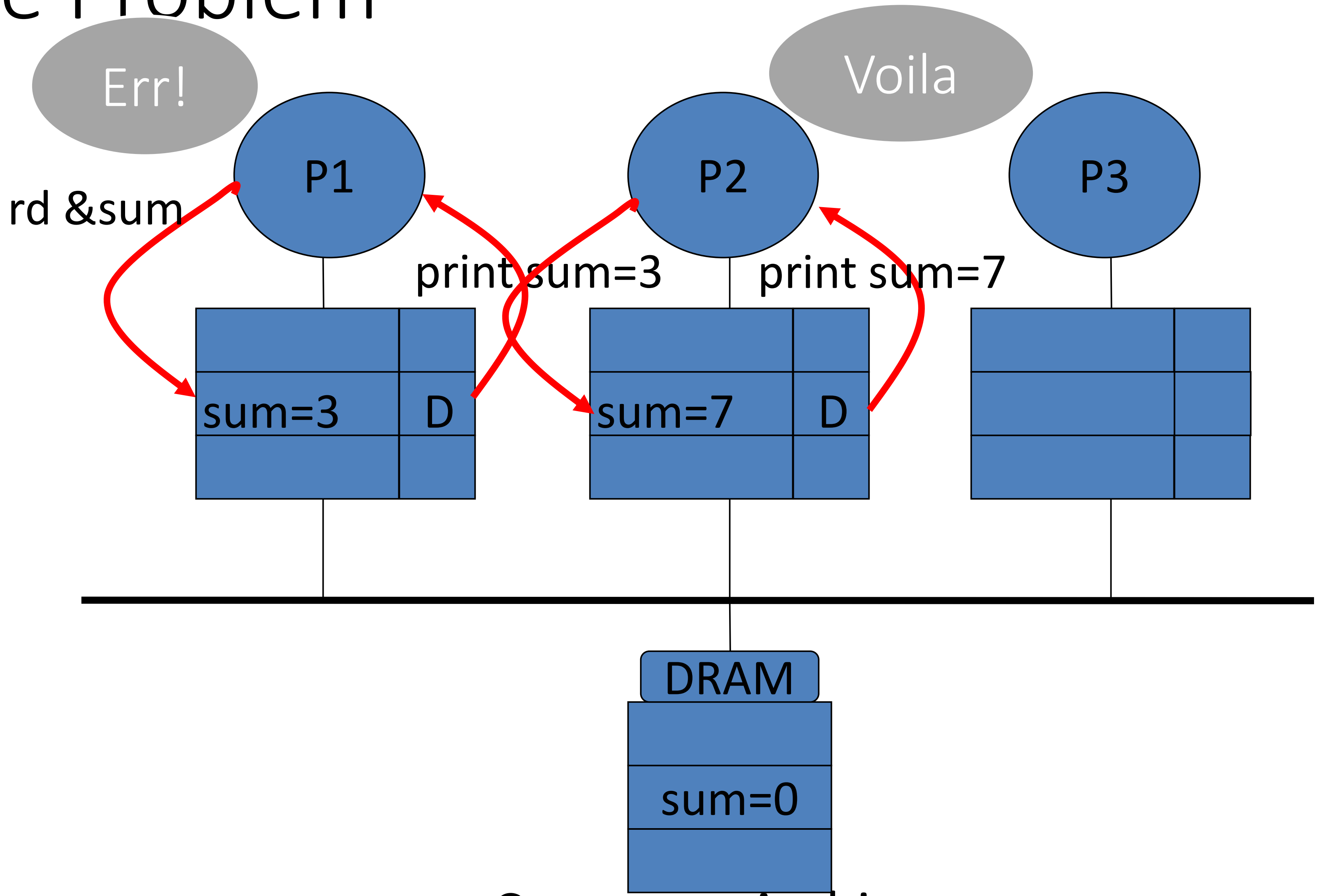
# Cache Coherence



# Cache Coherence



# The Problem


Err!
Voila
P1
P2
P3
rd &sum
print sum=3
print sum=7
sum=3
D
sum=7
D
DRAM
sum=0

# The Problem

If multiple cores cache the same block, how do they ensure they all see a consistent state?

Solution:

1. **Write Propagation**: All writes eventually become visible to other cores.
2. **Write Serialization**: All cores see write to a cache line in same order.

Need a protocol to ensure (1) and (2) called *cache coherence protocol*

# Invalidate/Update

**Invalidate** –

- Write to a cache line, and simultaneously broadcast invalidation of address to all others
- Other cores clear/invalidate their cache lines

**Update** –

- Write to a cache line, and simultaneously broadcast written data to all others
- Other cores update their caches if data was present

# A Simple MSI protocol

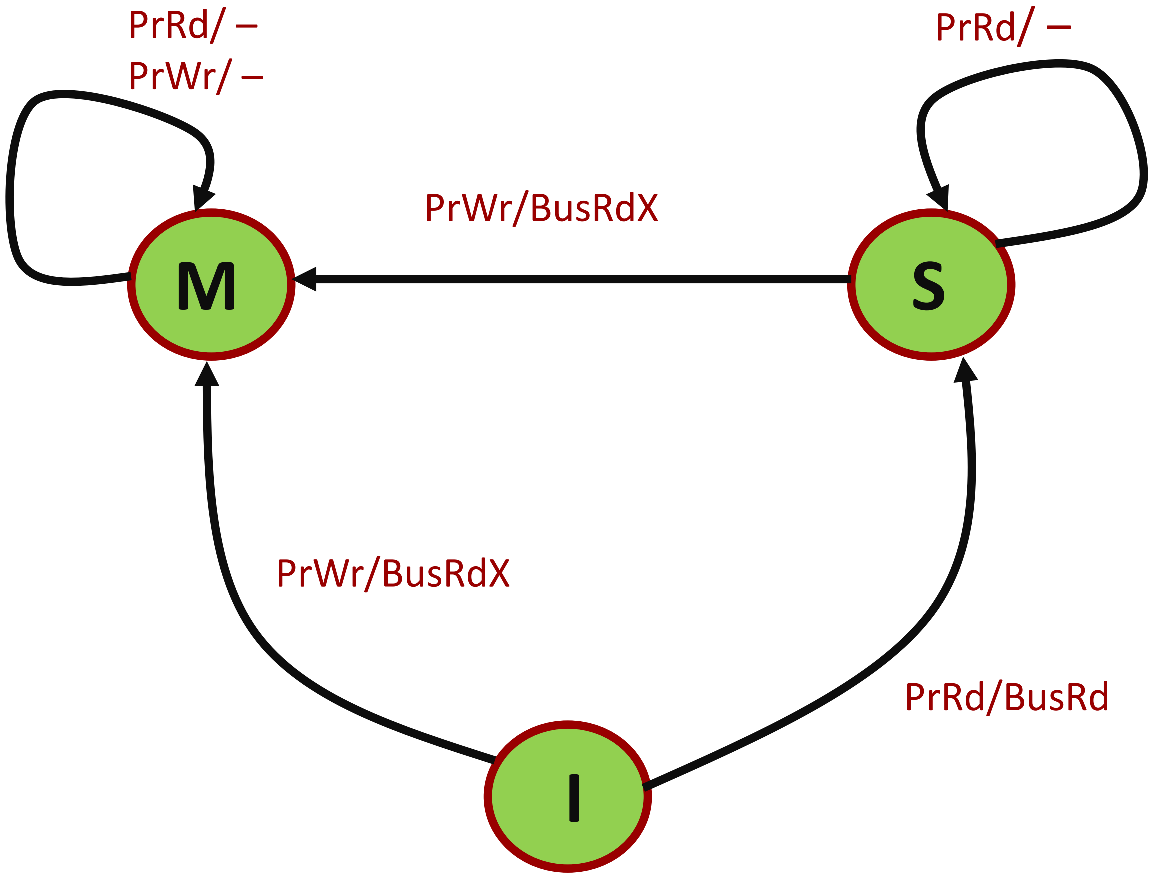
Extend single valid bit per line to three states:

- **M**(odified): only one cache, memory not updated.
- **S**(hared): one or more caches, and memory copy is up-to-date
- **I**(nvalid): not present

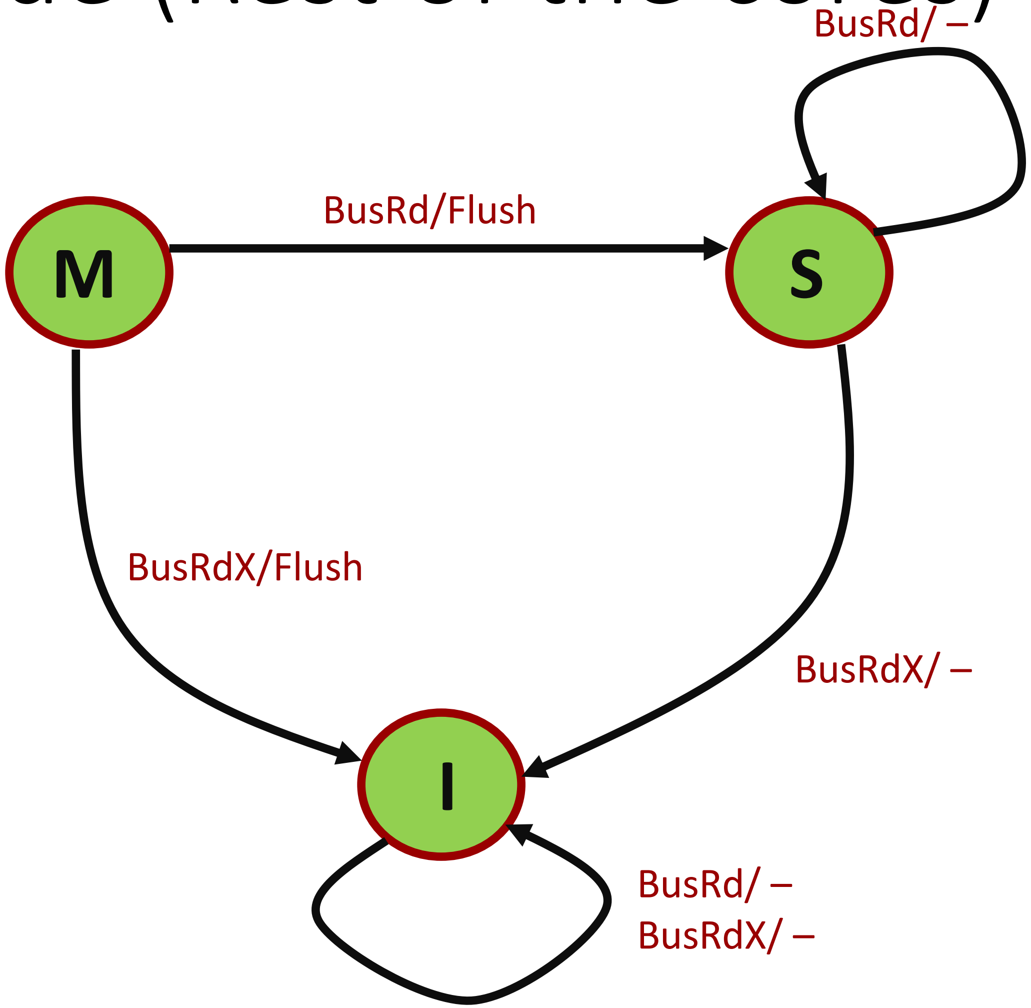
Read - *Read* request on bus, transitions to **S**

Write - *ReadEx* request, transitions to **M** state

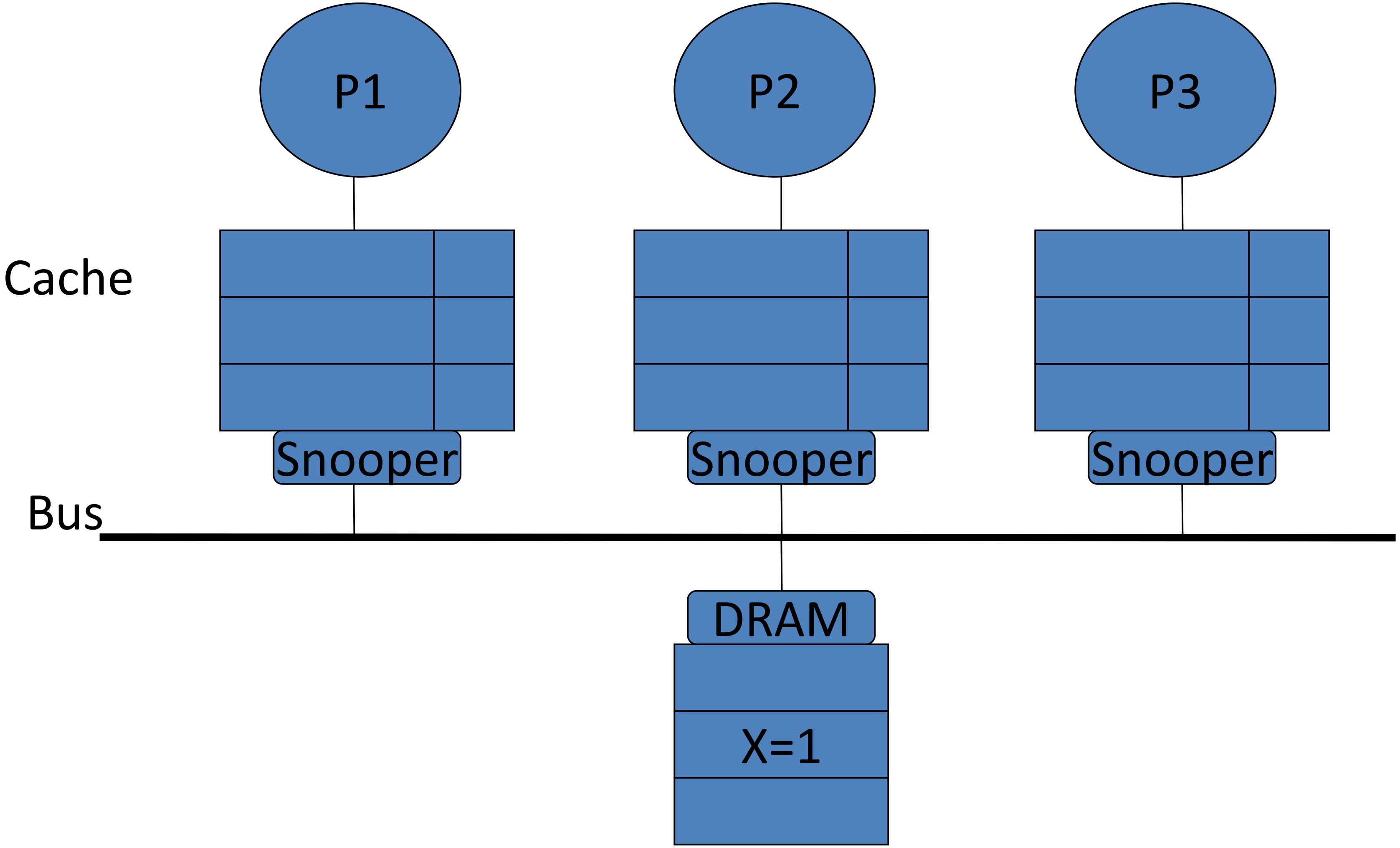
# State-transitions: Processor side (Master core)



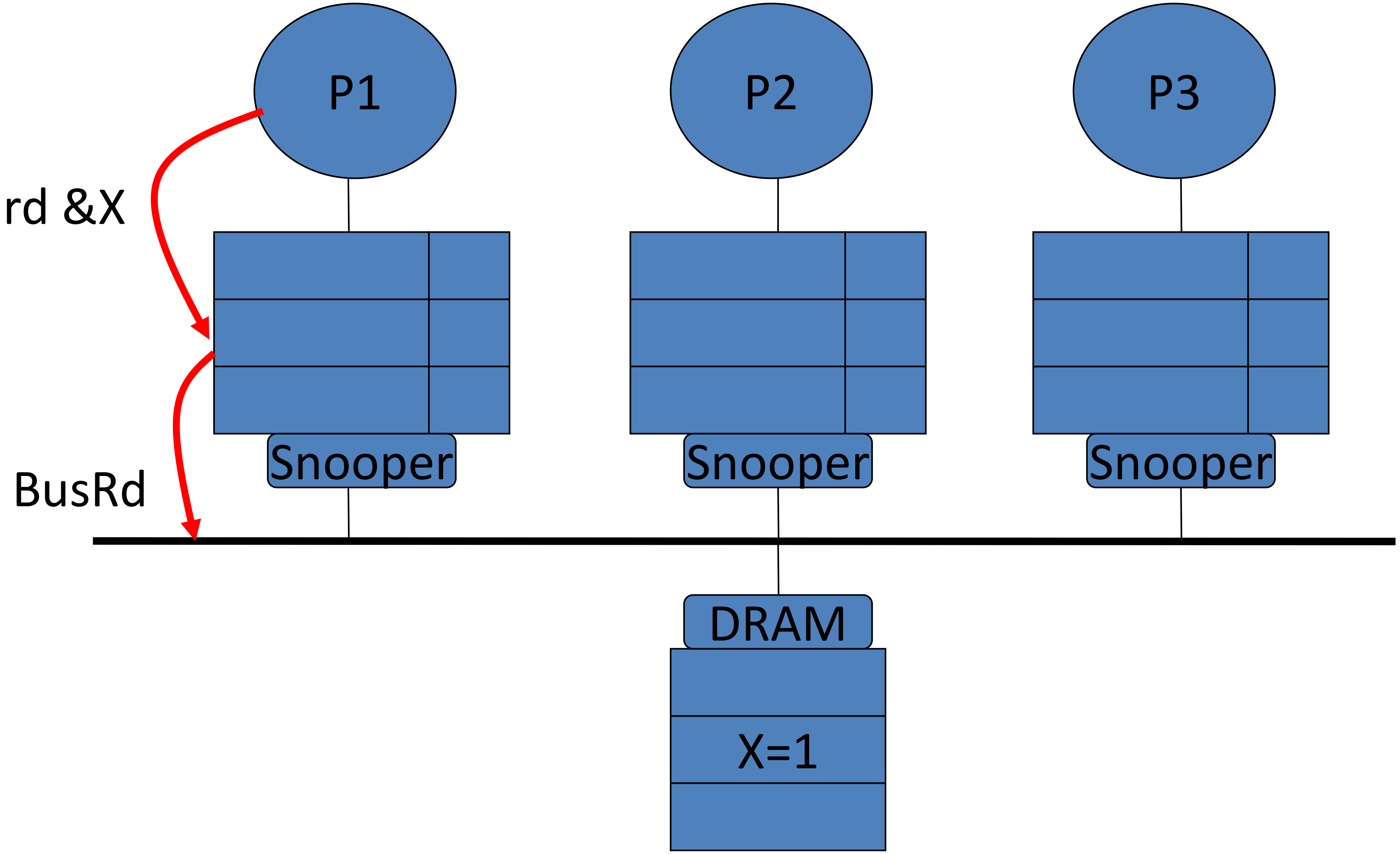
# Bus-side (Rest of the cores)



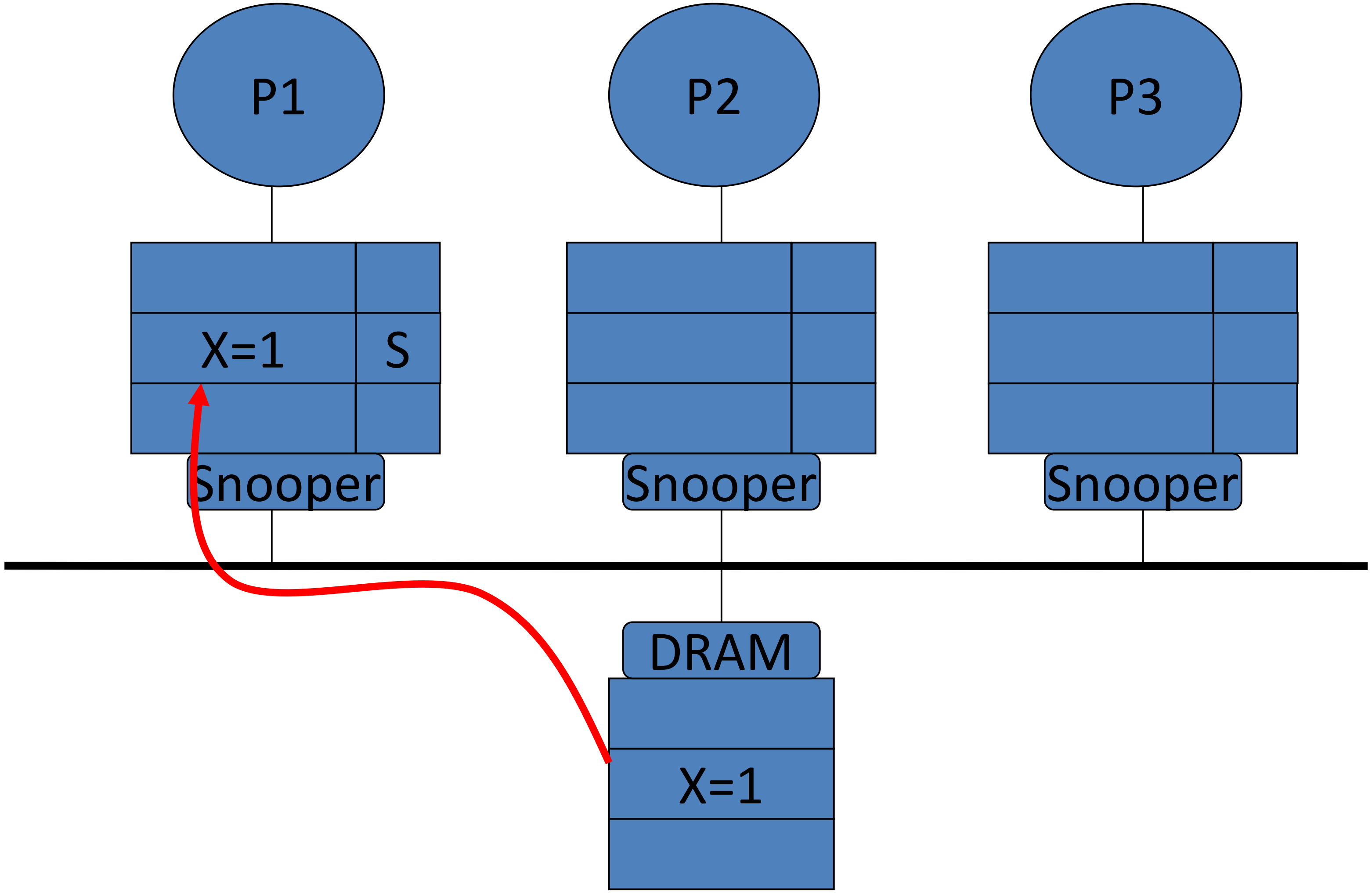
# MSI Protocol



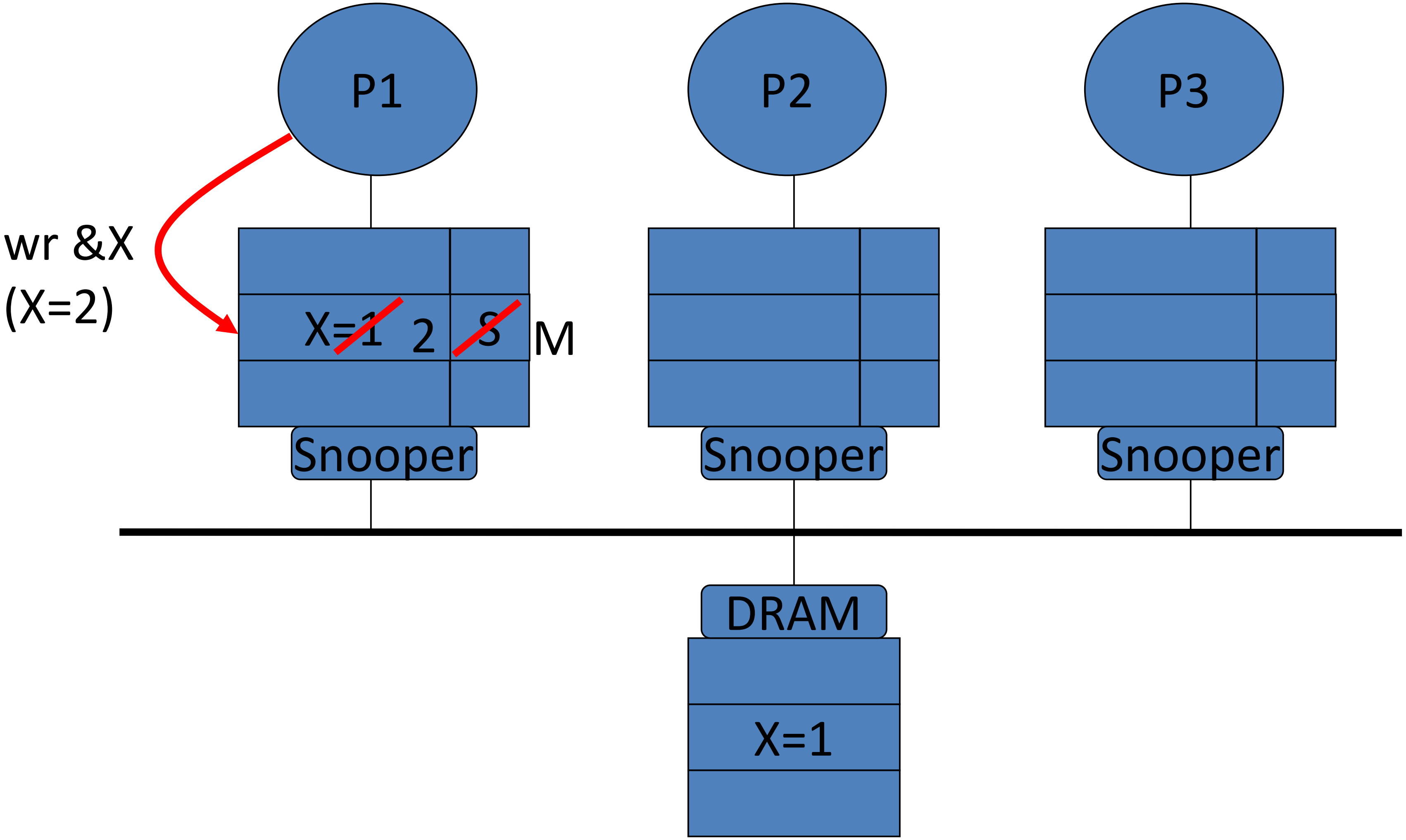
# MSI Protocol



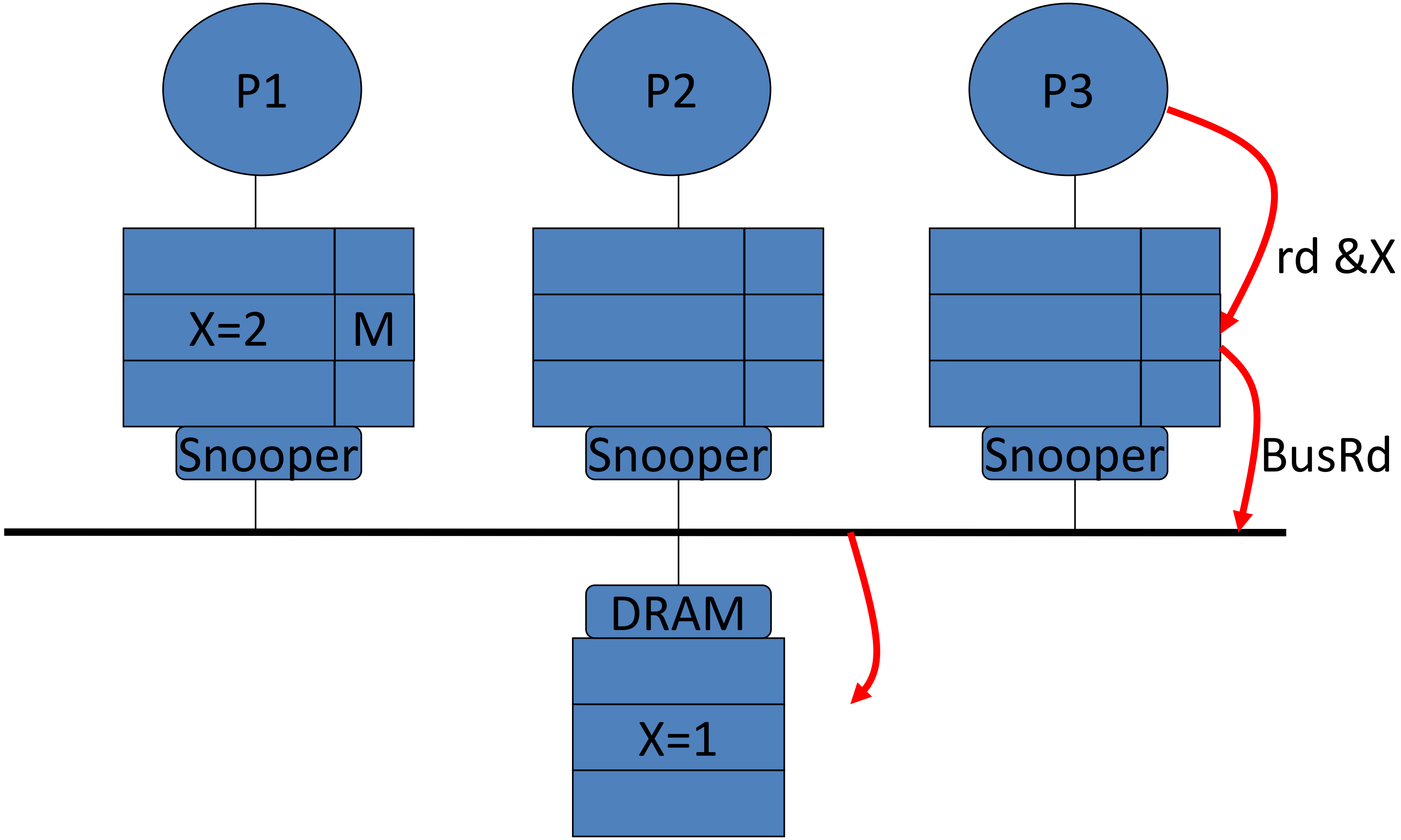
# MSI Protocol



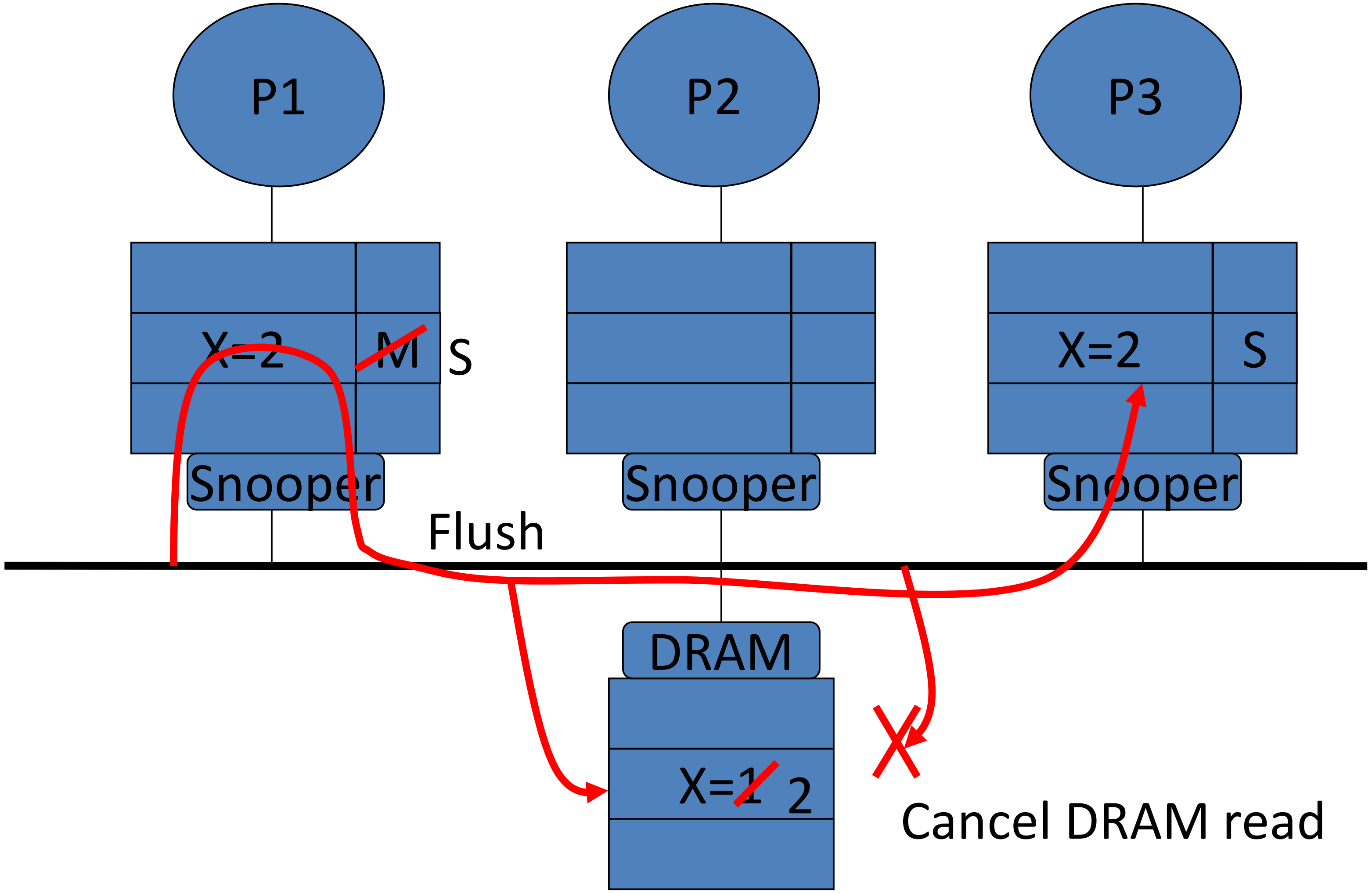
# MSI Protocol



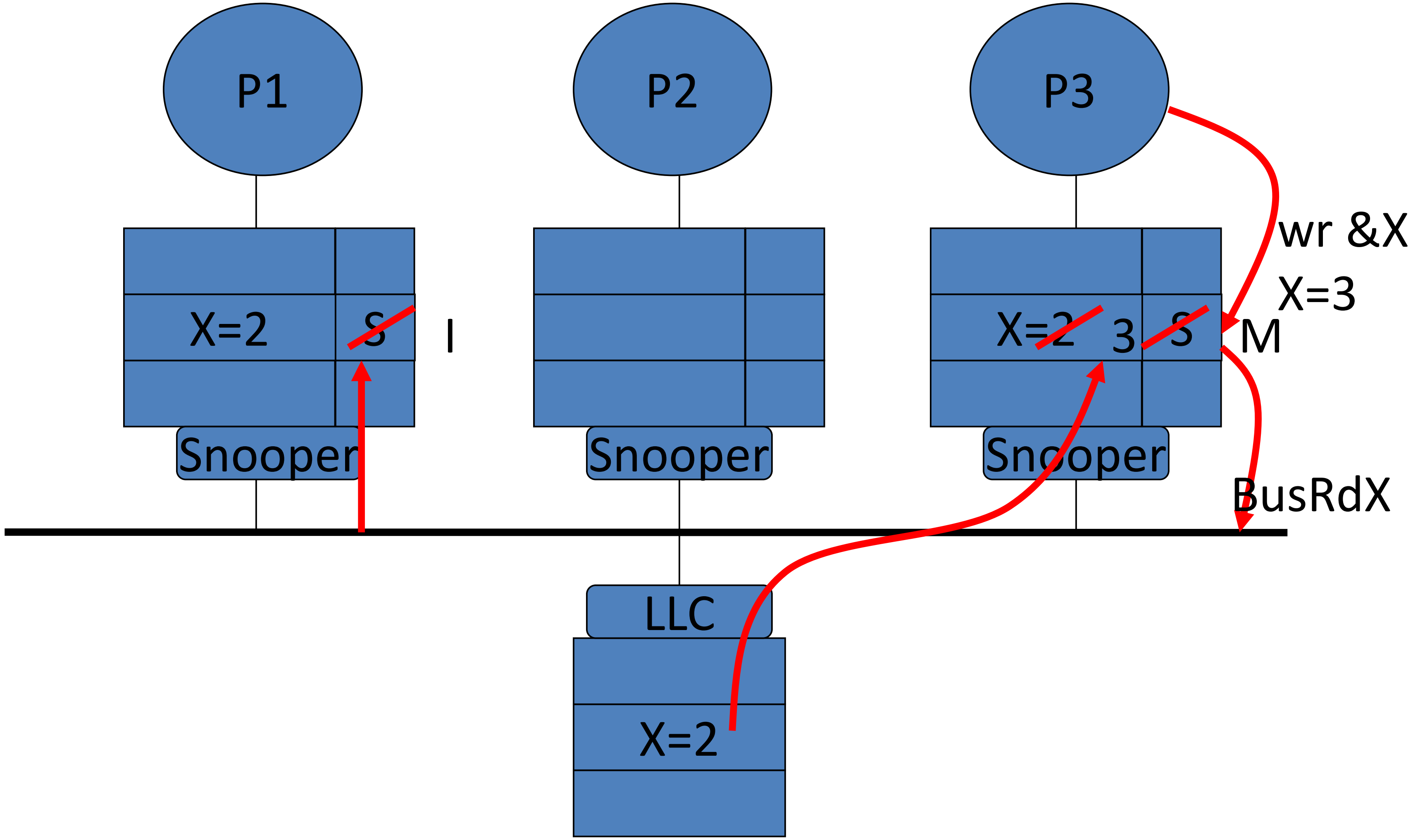
# MSI Protocol



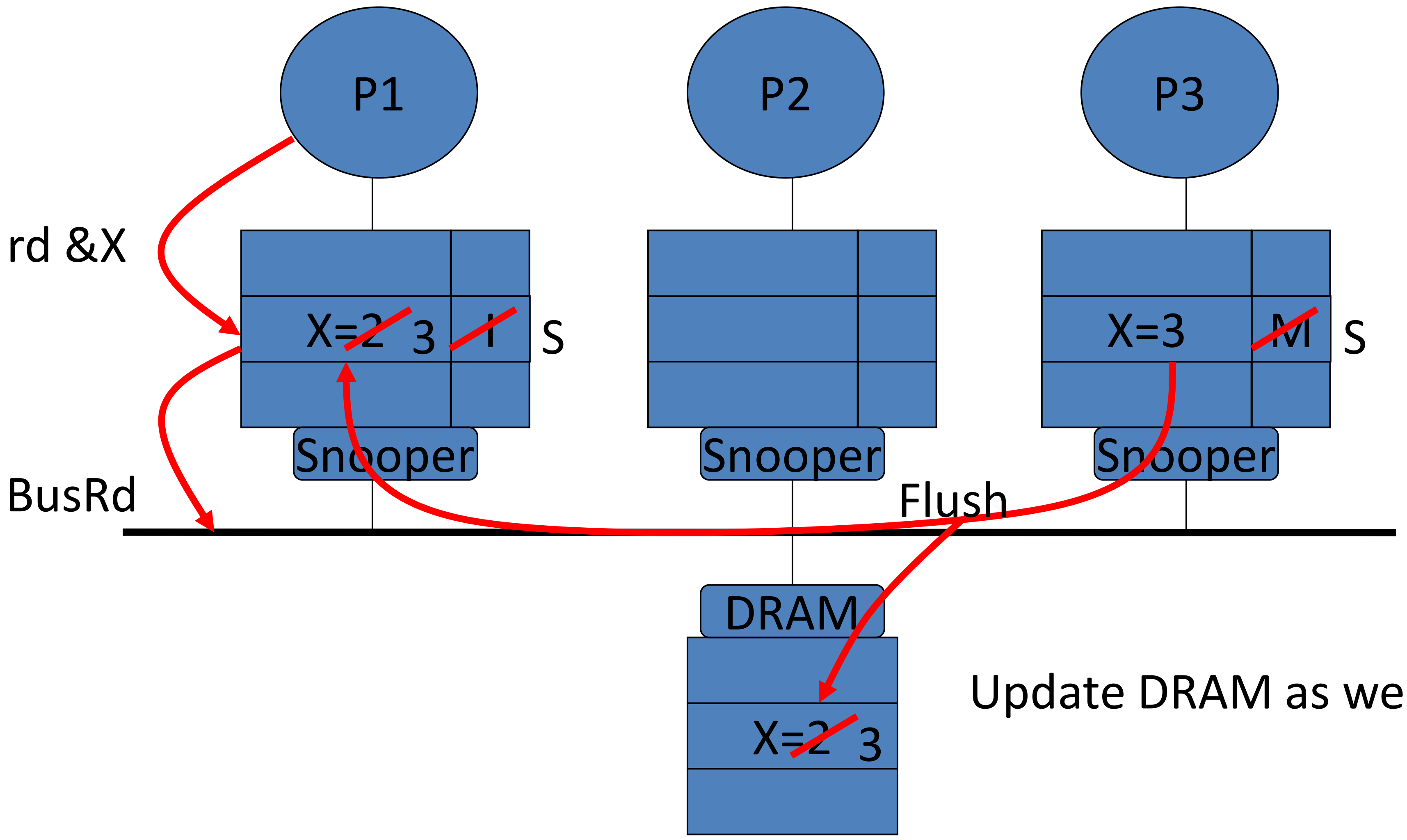
# MSI Protocol



# MSI Protocol

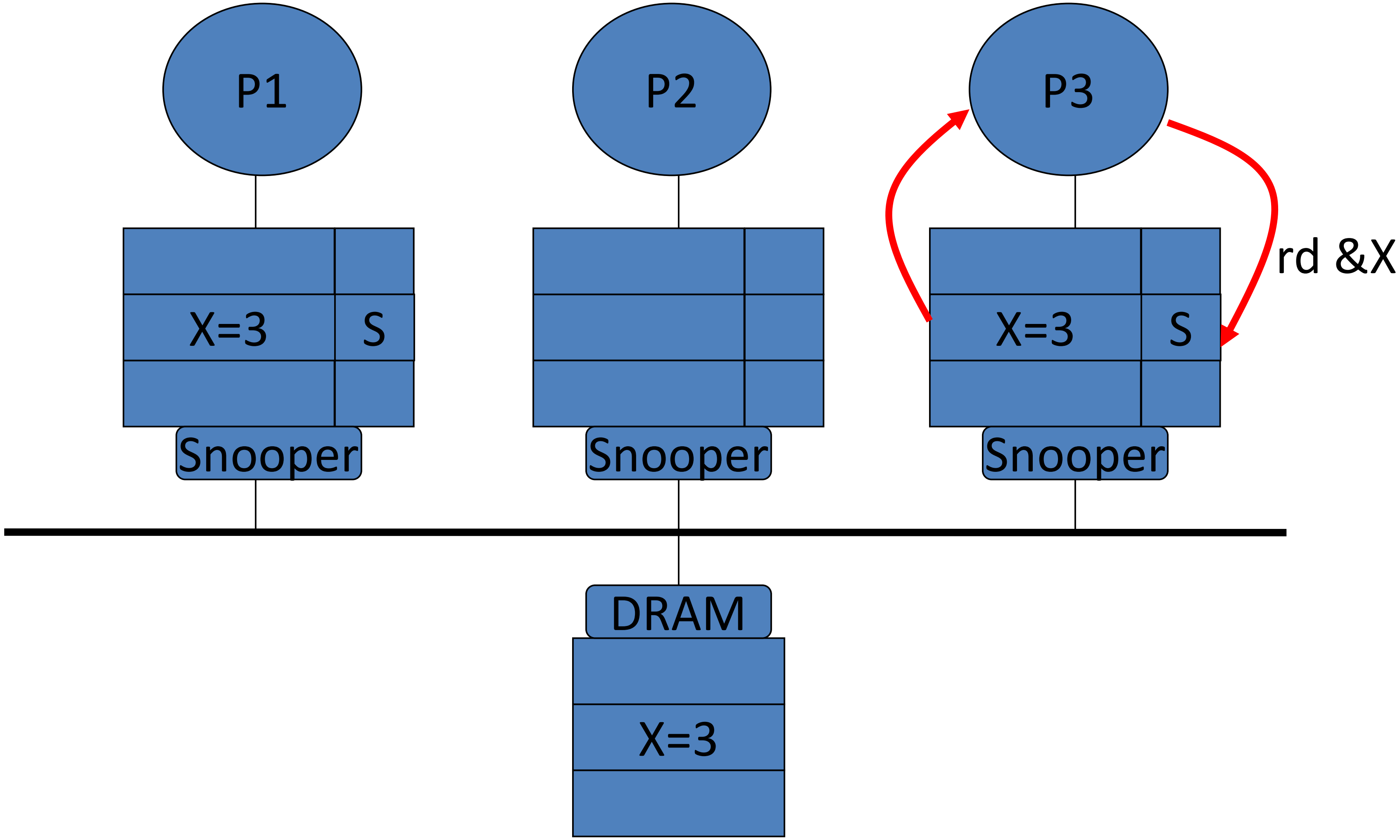


# MSI Protocol



Update DRAM as well

# MSI Protocol



# Summary: For your practice

| Proc Action | State P1 | State P2 | State P3 | Bus Action | Data From |
|---|---|---|---|---|---|
| R1 | - | - | - | BusRd | Mem |
| W1 | - | - | - | BusRdX | Mem |
| R3 | - | - | - | BusRd | P1 cache |
| W3 | - | - | - | BusRdX | Mem |
| R1 | - | - | - | BusRd | P3 cache |
| R3 | - | - | - | - | - |
| R2 | - | - | - | BusRd | Mem |

# Summary: For your practice

| Proc Action | State P1 | State P2 | State P3 | Bus Action | Data From |
|---|---|---|---|---|---|
| R1 | S | - | - | BusRd | Mem |
| W1 | M | - | - | BusRdX | Mem |
| R3 | S | - | S | BusRd | P1 cache |
| W3 | I | - | M | BusRdX | Mem |
| R1 | S | - | S | BusRd | P3 cache |
| R3 | S | - | S | - | - |
| R2 | S | S | S | BusRd | Mem |