Invalidate and Update Protocols

Lecture 14
(Chapter 7, cont.)

E. F. Gehringer,
based on slides by Yan Solihin

Lecture 15 Outline

- **MSI protocol**
  - **State diagram**
  - **Animations**
- **MESI protocol**
- **Dragon protocol**
- **Firefly protocol**

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

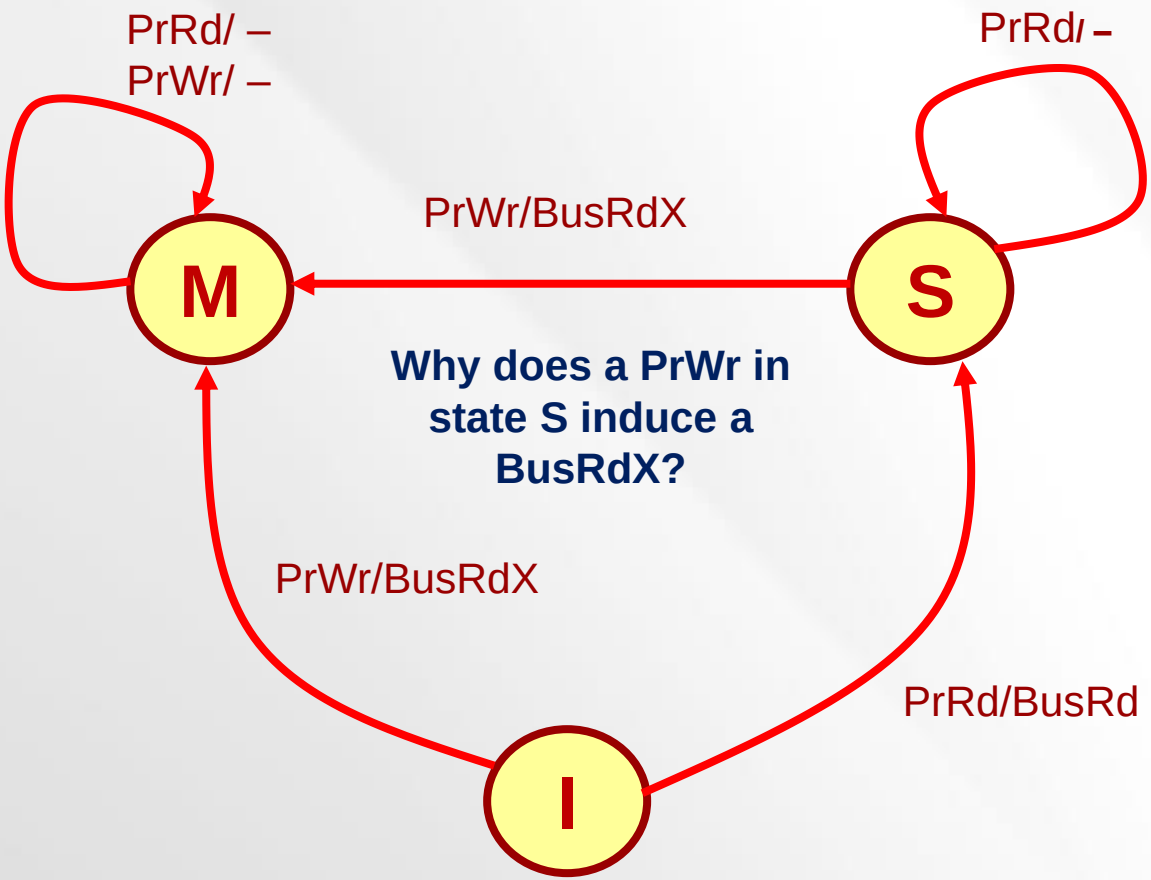
Basic MSI Writeback Invalidation Protocol

- States
  - Invalid (I)
  - Shared (S): one or more copies, and memory copy is up-to-date
  - Dirty or Modified (M): only one copy
- Processor Events:
  - PrRd (read), PrWr (write)
- Bus Transactions
  - BusRd: asks for copy with no intent to modify
  - BusRdX: asks for copy with intent to modify (instead of BusWr)
  - Flush: updates memory
- Actions
  - Update state, perform bus transaction, flush value onto bus

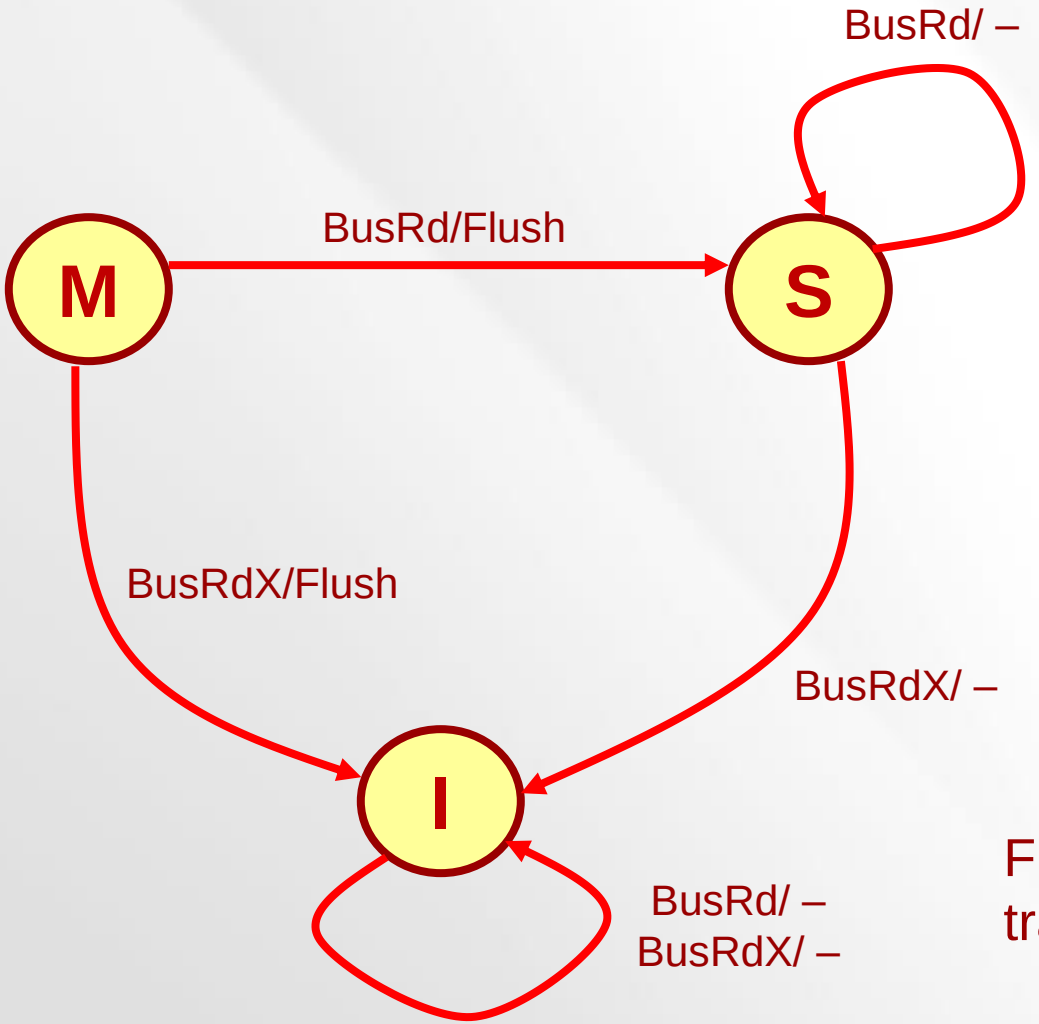
State-Transition Diagrams

- On the following slides, we will display the state-transition diagrams
  - for processor-initiated transactions
  - for bus-initiated transactions
- We will see transitions of the following form:
  - Invalidation: ⟨Any⟩ → I
  - Intervention: {Exclusive, Modified} → Shared

MSI: Processor-Initiated Transactions



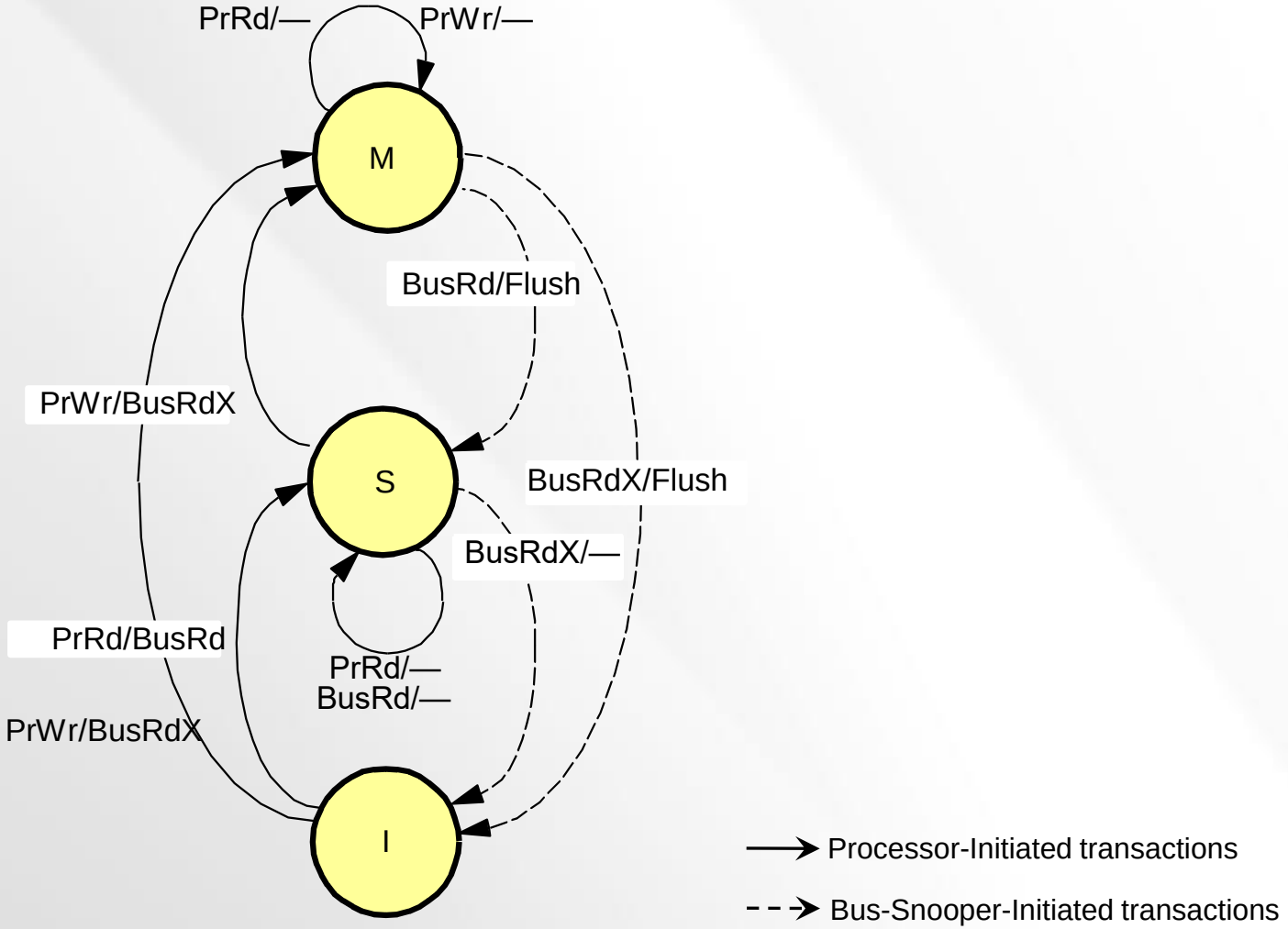
MSI: Bus-Initiated Transactions



Thus, valid data must be supplied by memory

Fill in the last two transitions here.

MSI State Transition Diagram



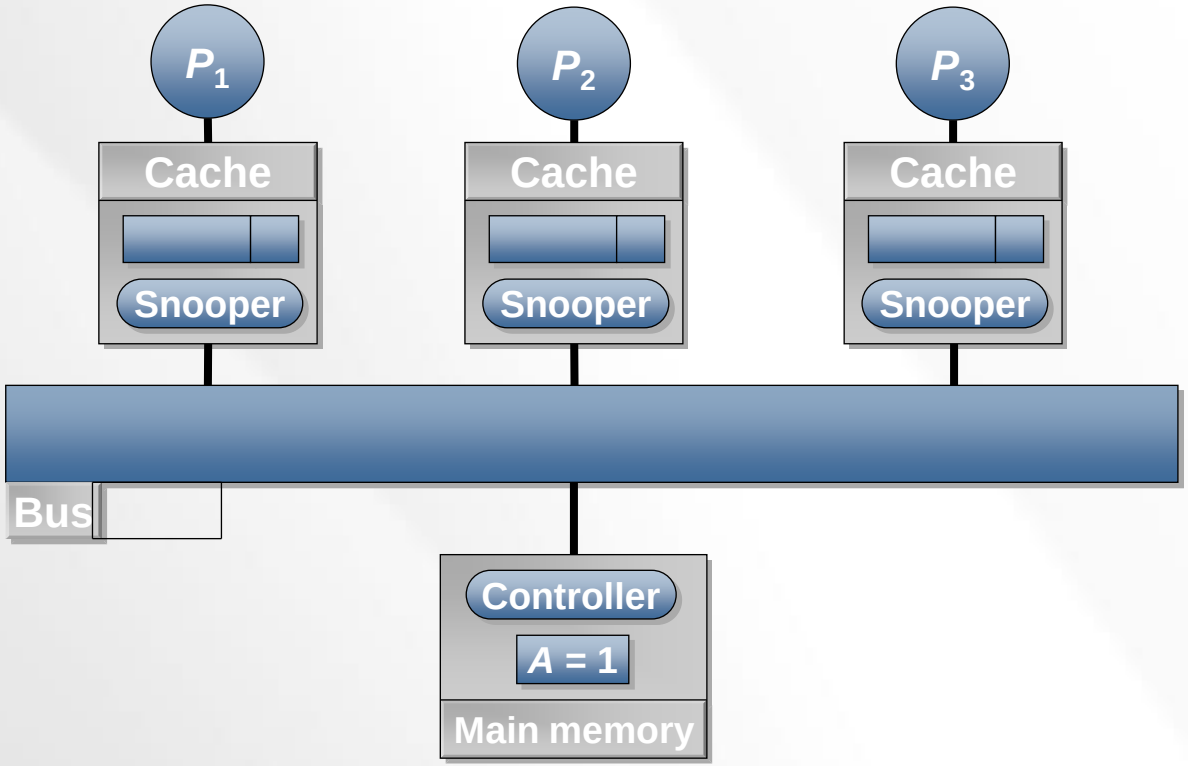
Lecture 15 Outline

- **MSI protocol**
  - **State diagram**
  - **Animations**
- **MESI protocol**
- **Dragon protocol**
- **Firefly protocol**

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

MSI Visualization – Start State

Start state. All caches empty and main memory has $A = 1$.

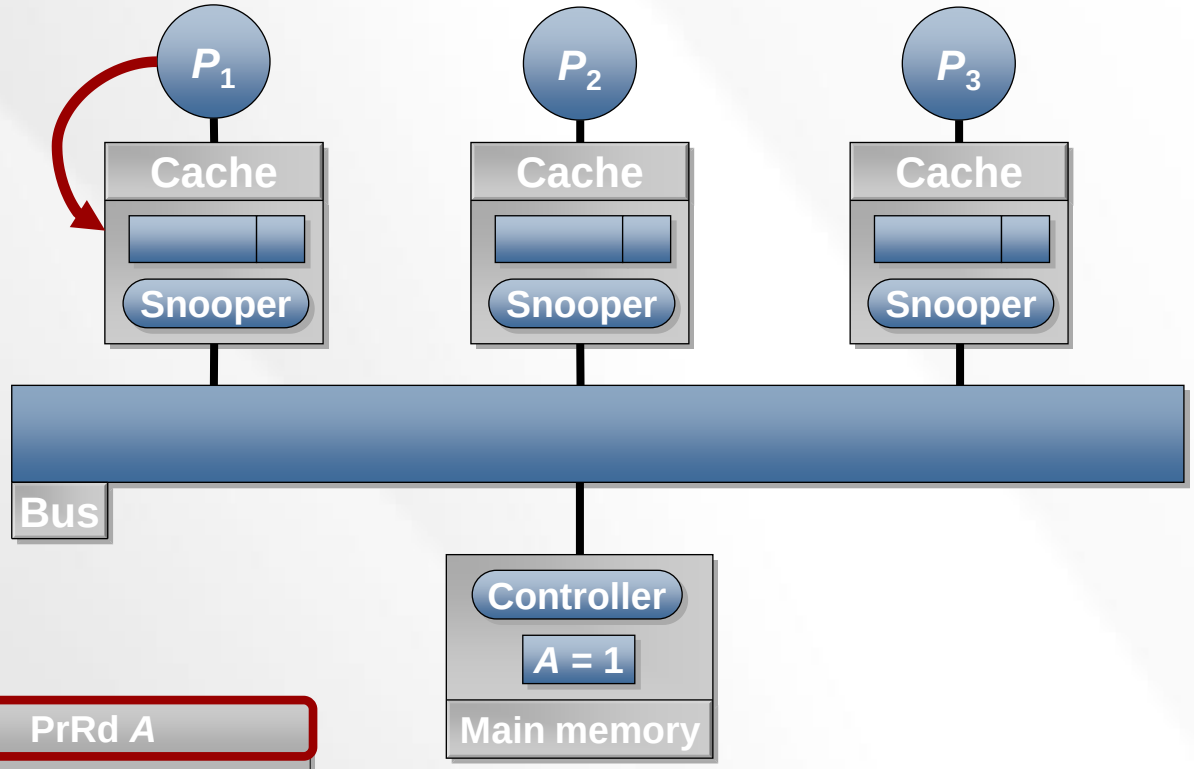


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Processor P_1 attempts to read A from its cache.



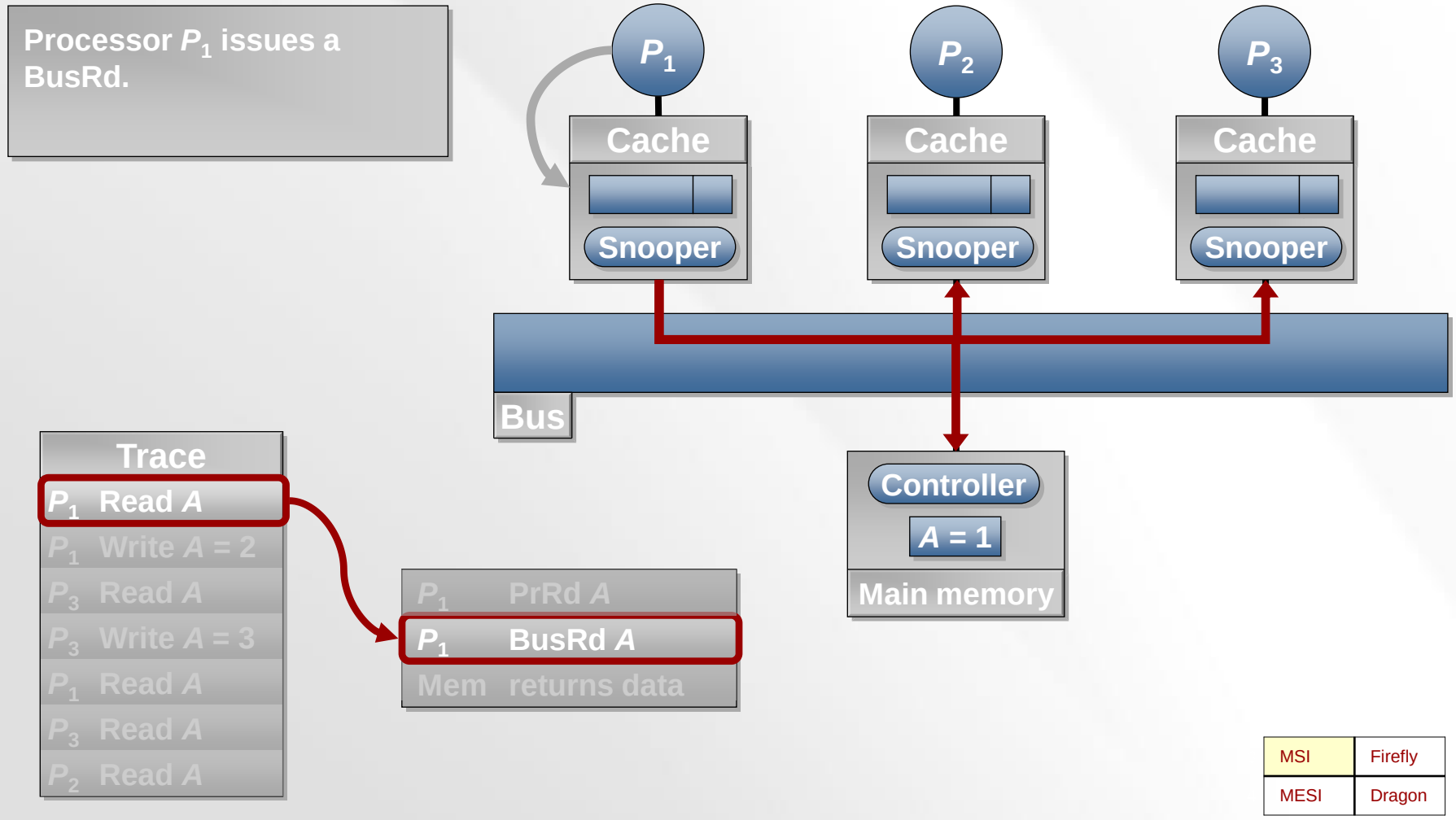
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

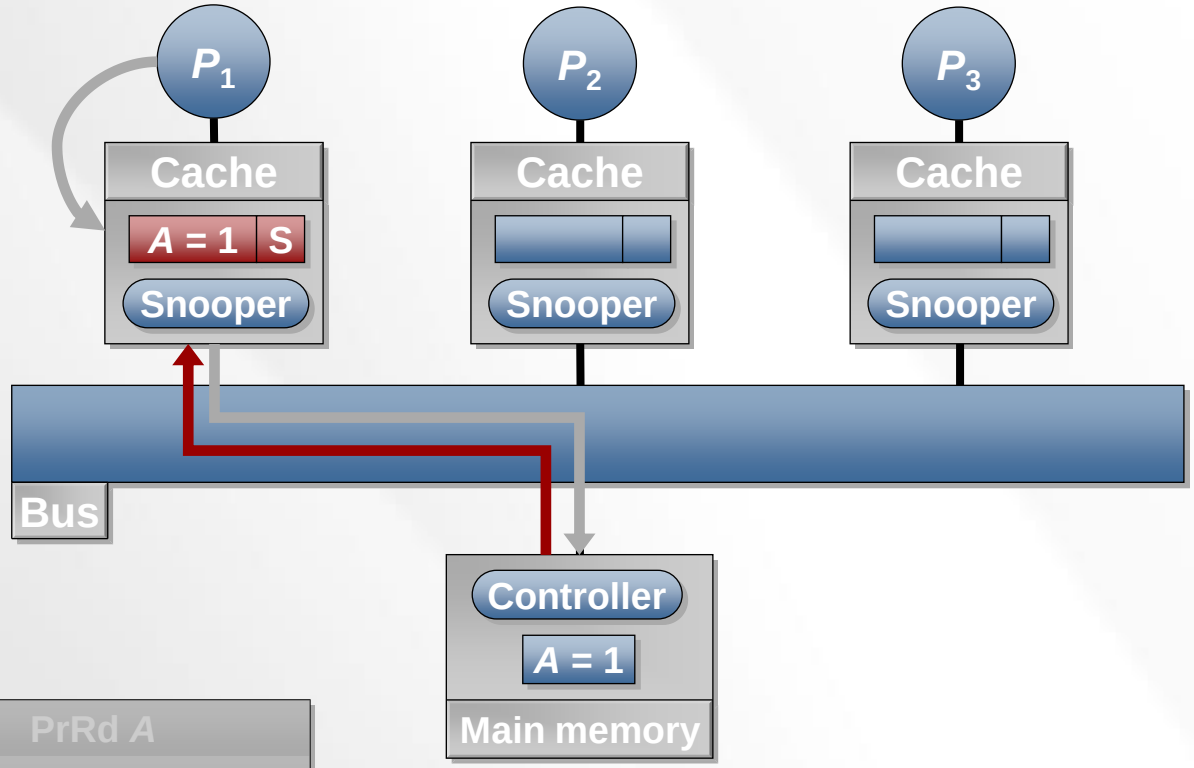
Processor P_1 issues a BusRd.



MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Main memory returns data to processor P_1 which updates its cache.



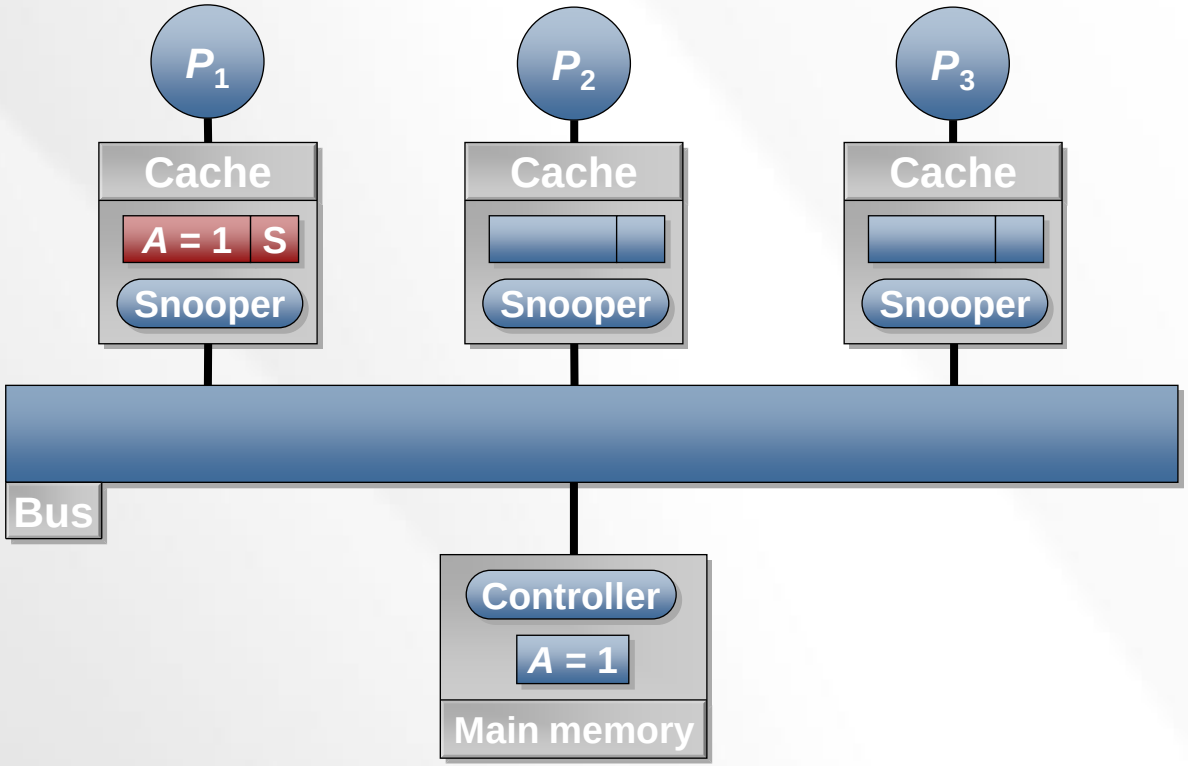
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Read operation completes.

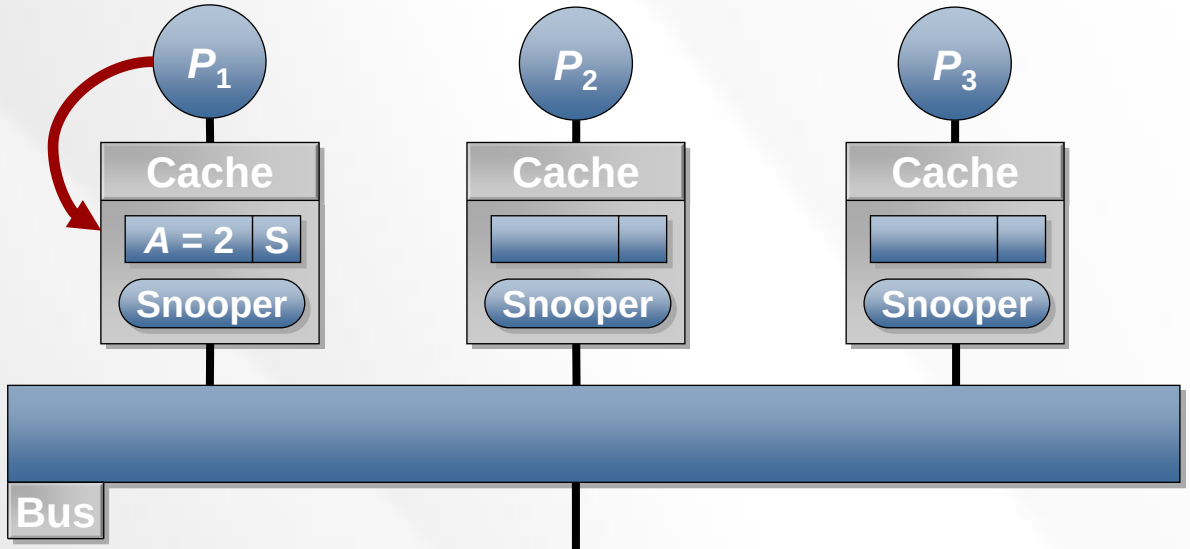


Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Writes $A = 2$

Processor P_1 writes to its cache.



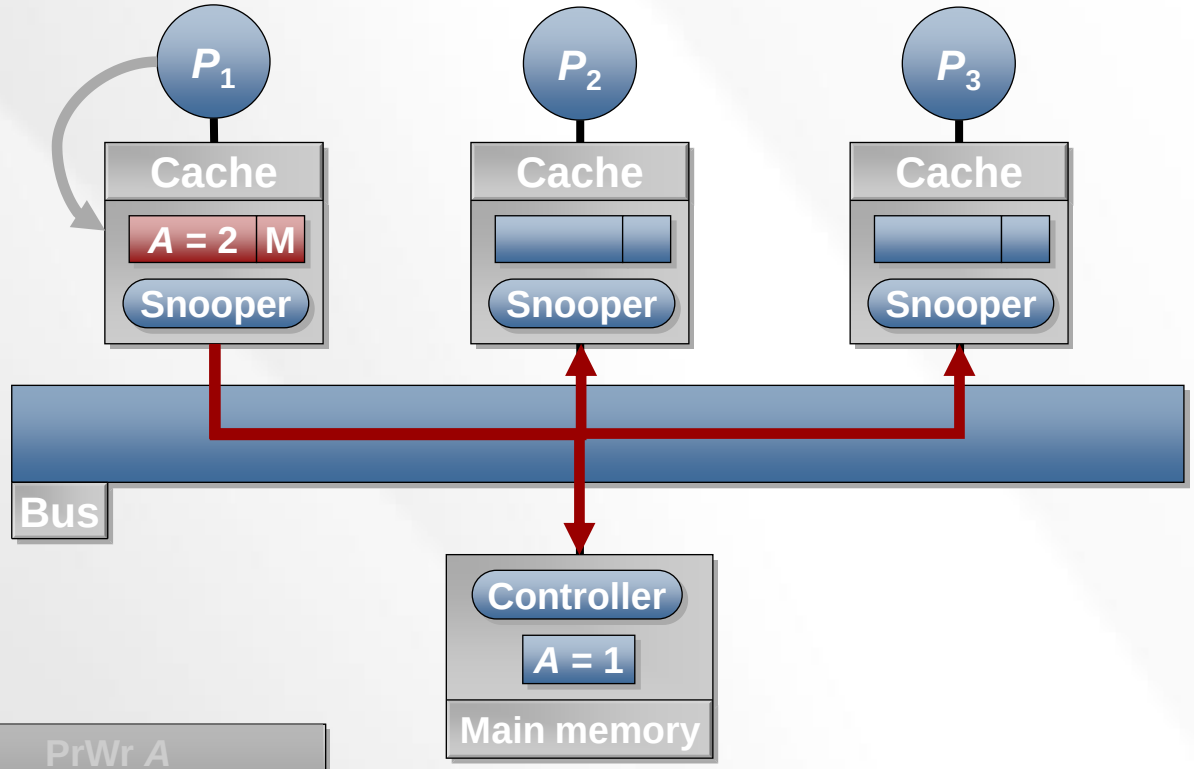
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrWr A
P_1	BusRdX

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Writes $A = 2$

Processor P_1 issues a BusRdX request.



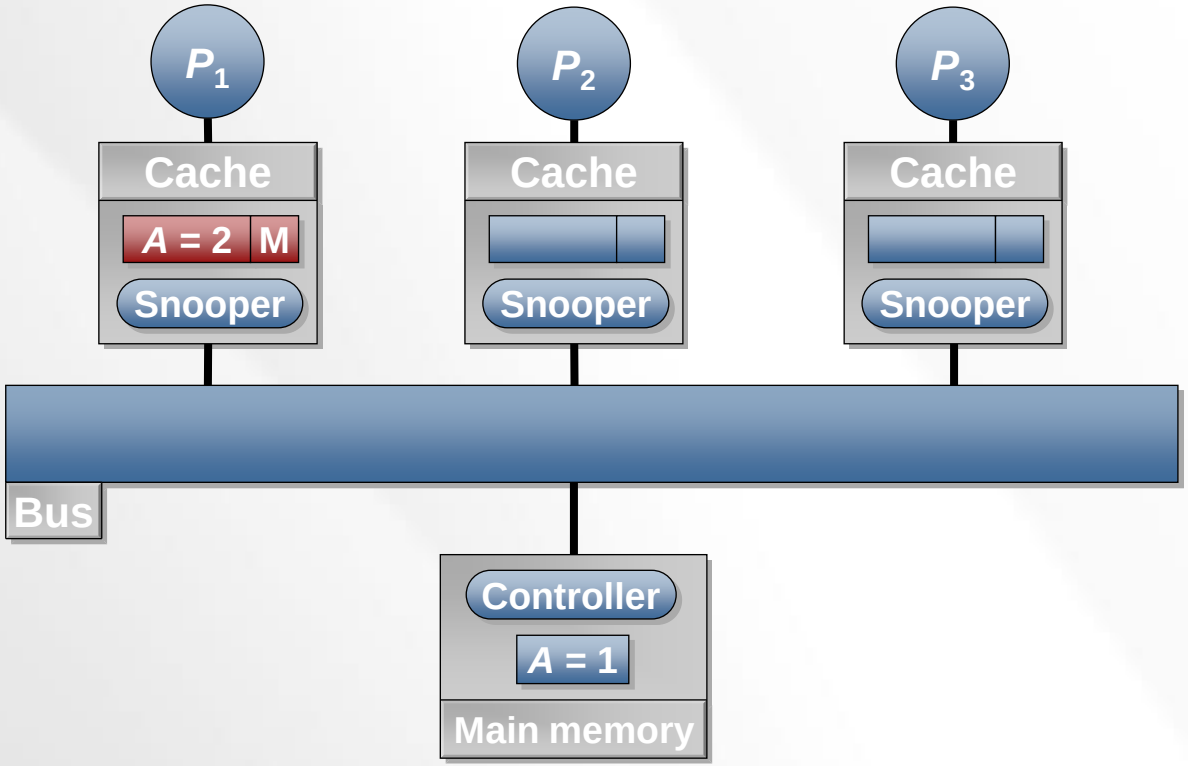
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrWr A
P_1	BusRdX

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Writes $A = 2$

Write operation completes.

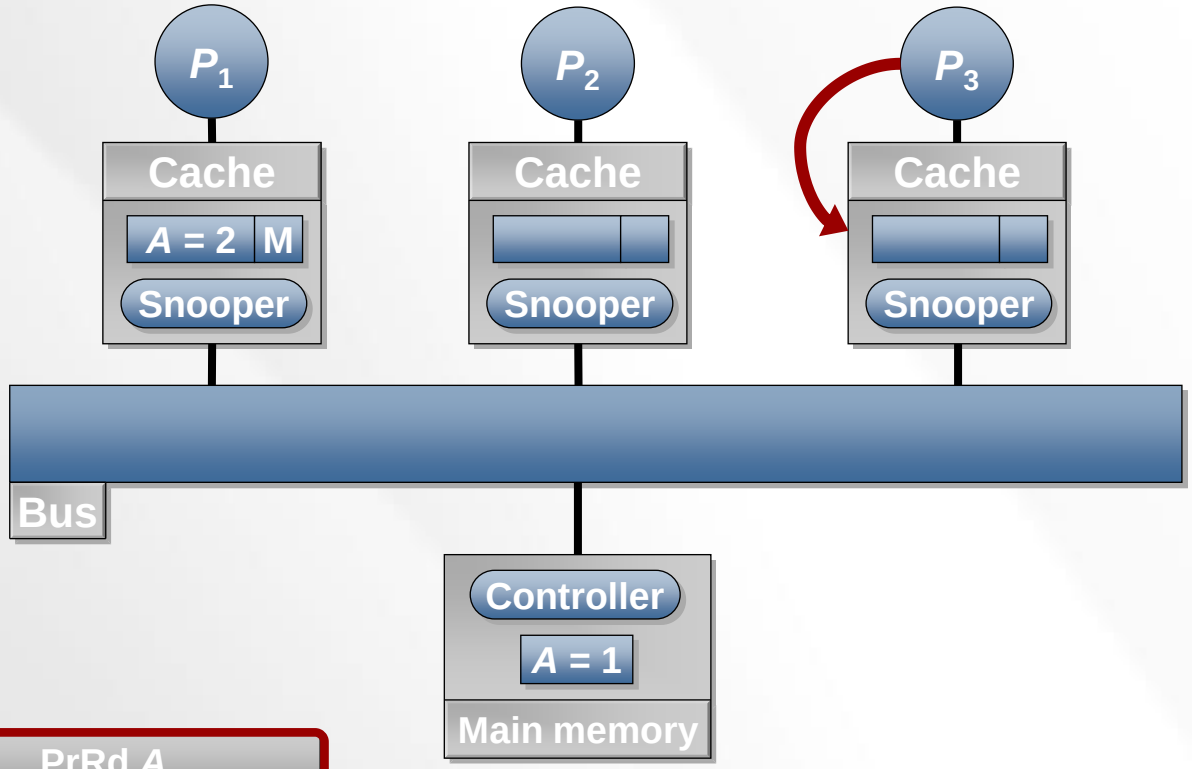


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_3 attempts to read A from its cache.



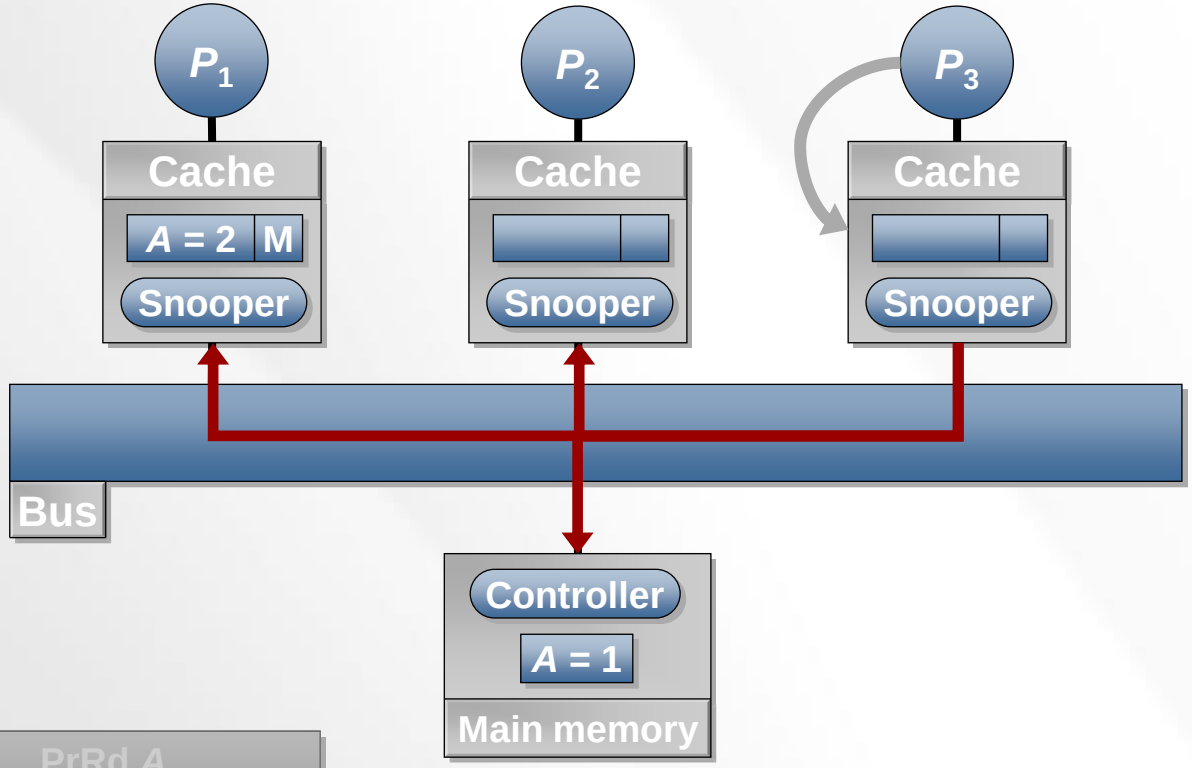
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_3 issues a BusRd.



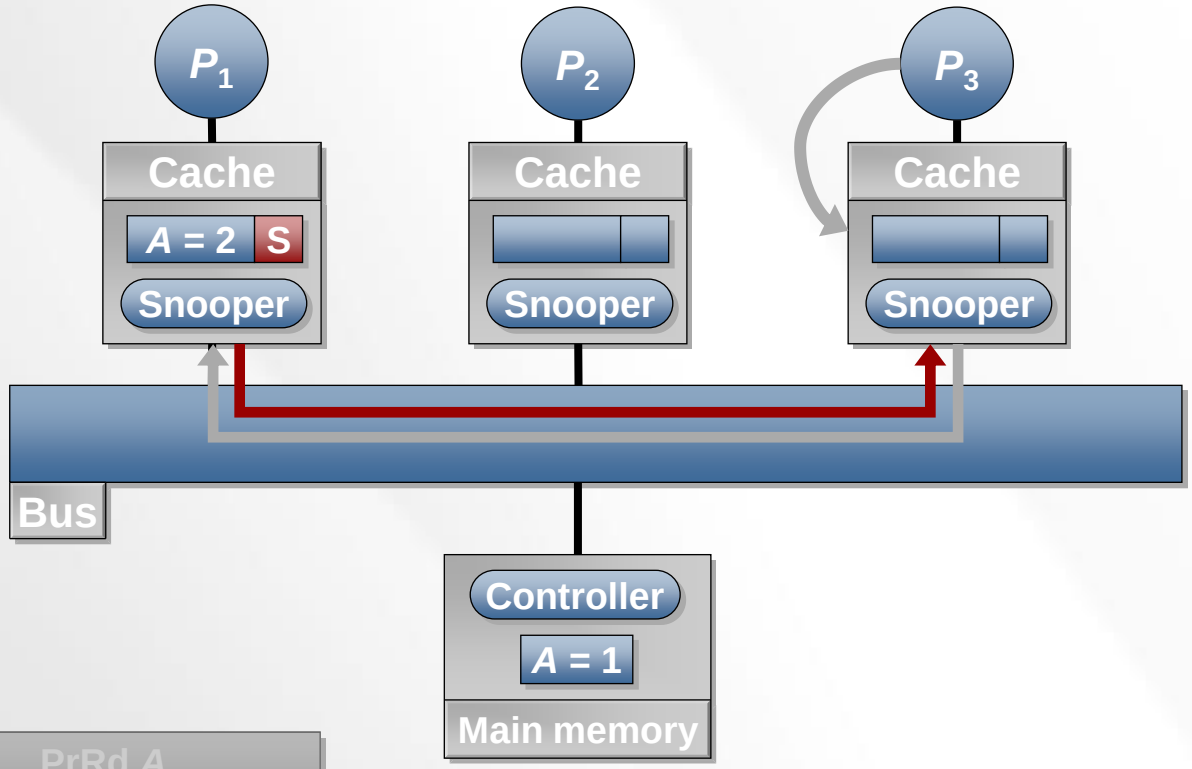
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_1 snoops the BusRd from processor P_3.



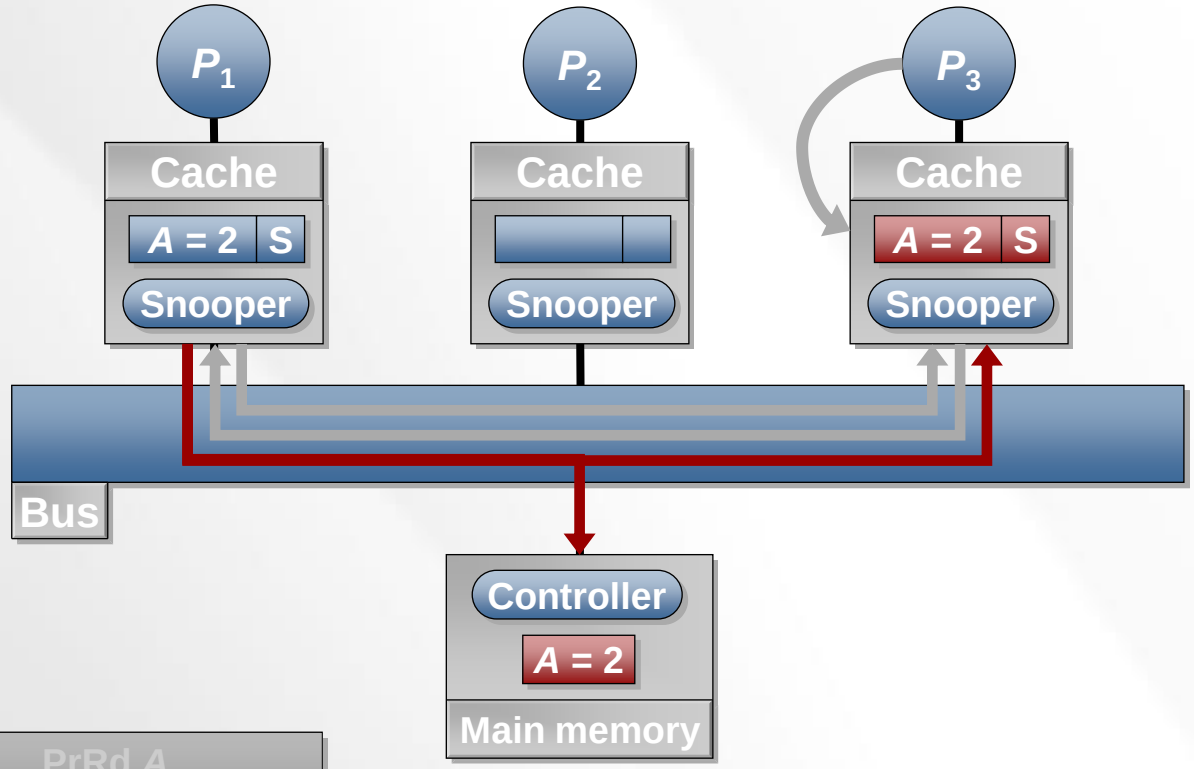
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_1 flushes A, sending updated data to P_3 and main memory.



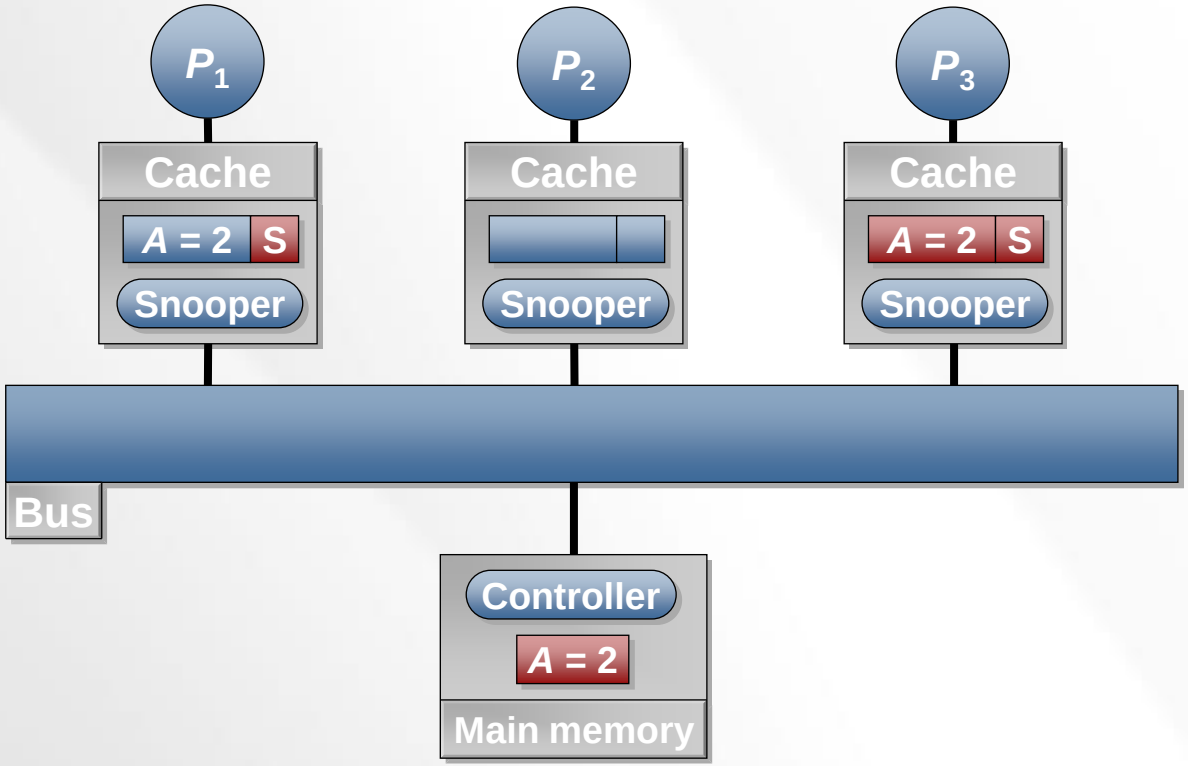
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Read operation completes.

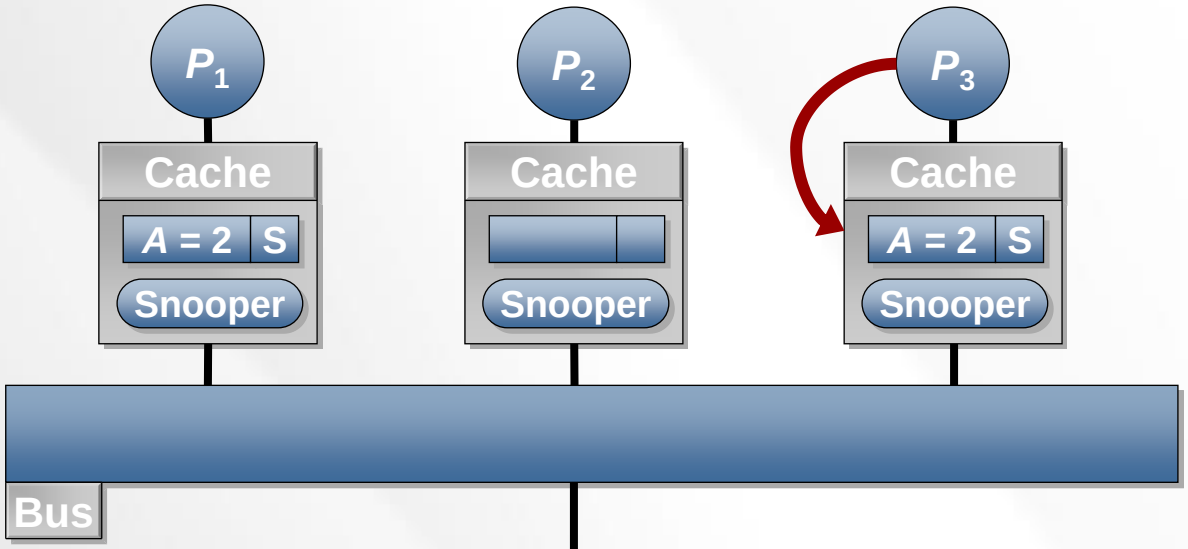


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Writes $A = 3$

Processor P_3 writes to its cache.



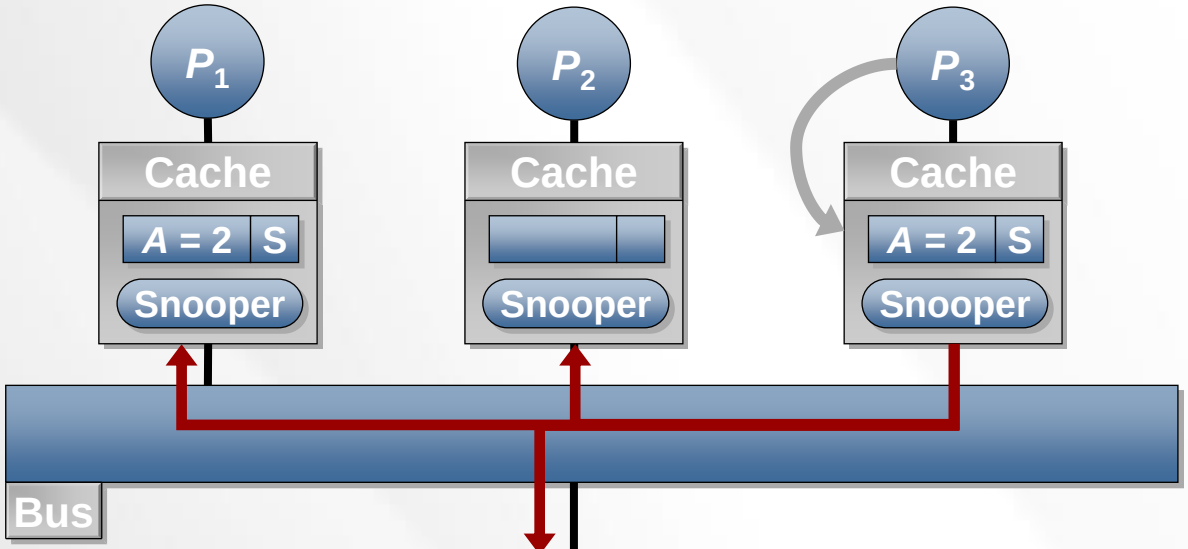
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrWr A
P_3	BusRdX
P_1	snoops BusRdX

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Writes $A = 3$

Processor P_3 issues a BusRdX request.



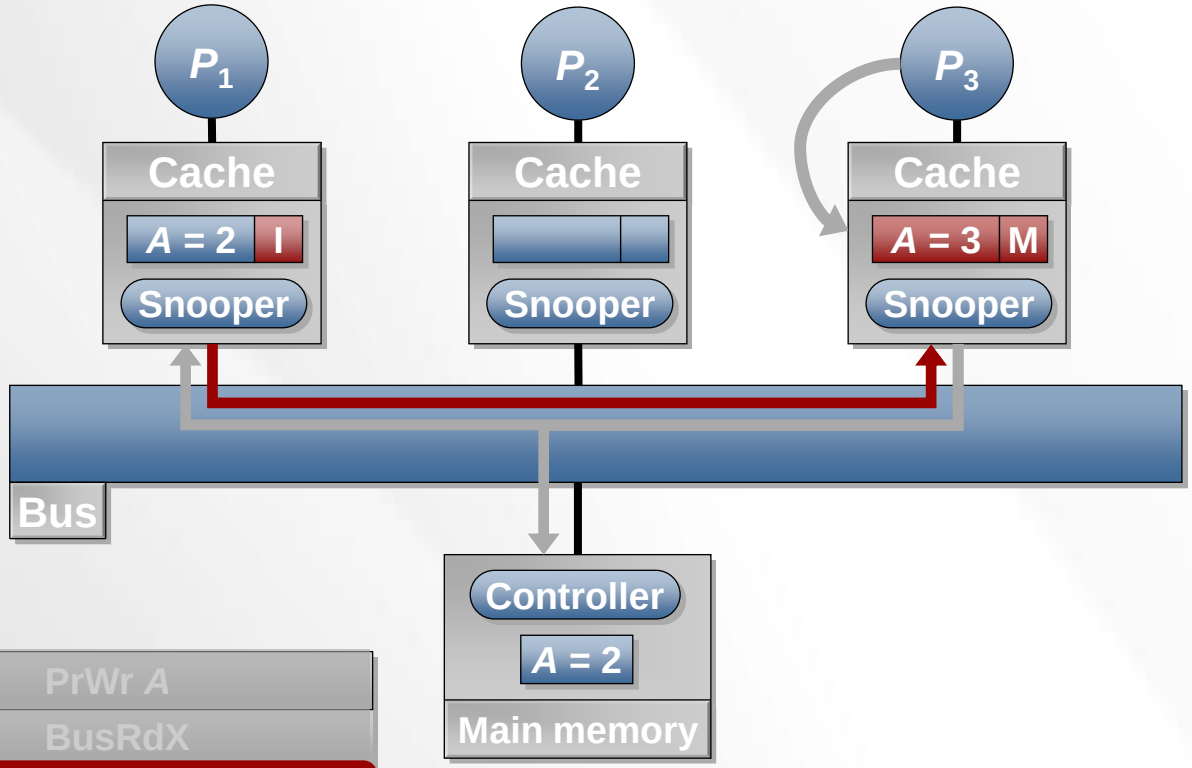
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrWr A
P_3	**BusRdX**
P_1	snoops BusRdX

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Writes $A = 3$

Processor P_1 snoops the BusRd and invalidates its cache.



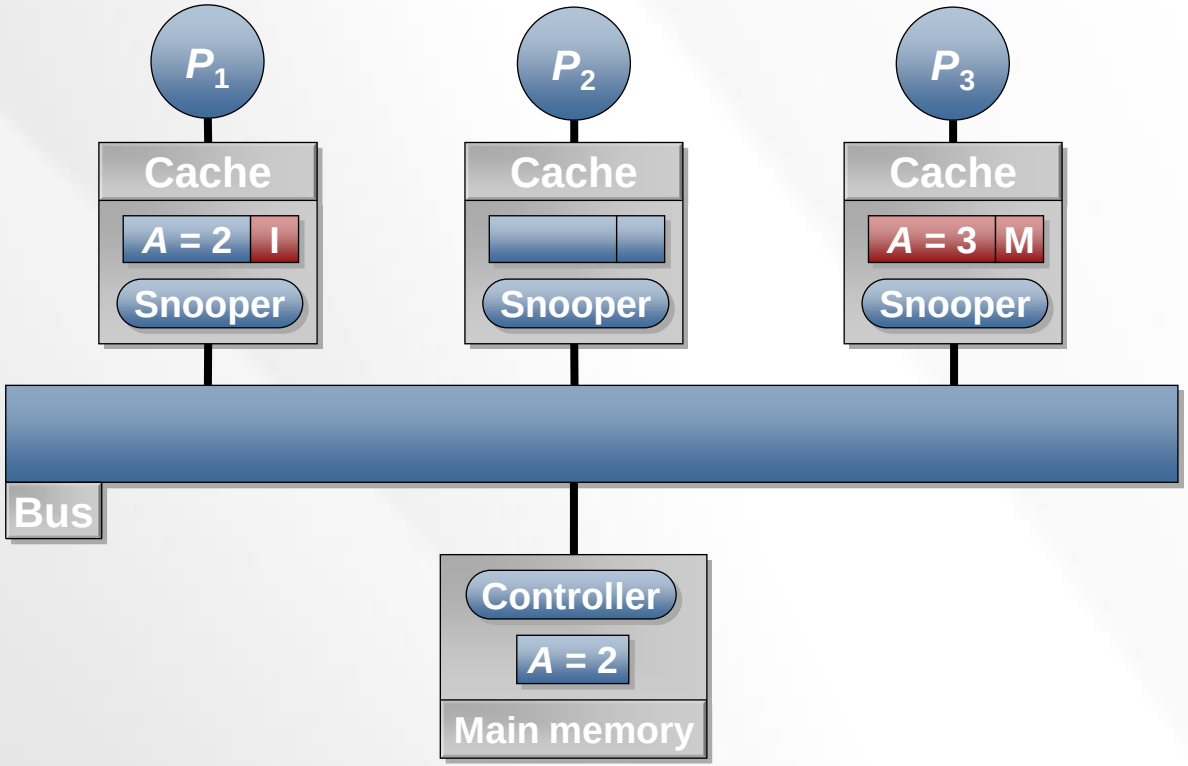
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A
P_3	BusRdX
P_1	snoops BusRdX

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Writes $A = 3$

Write operation completes.

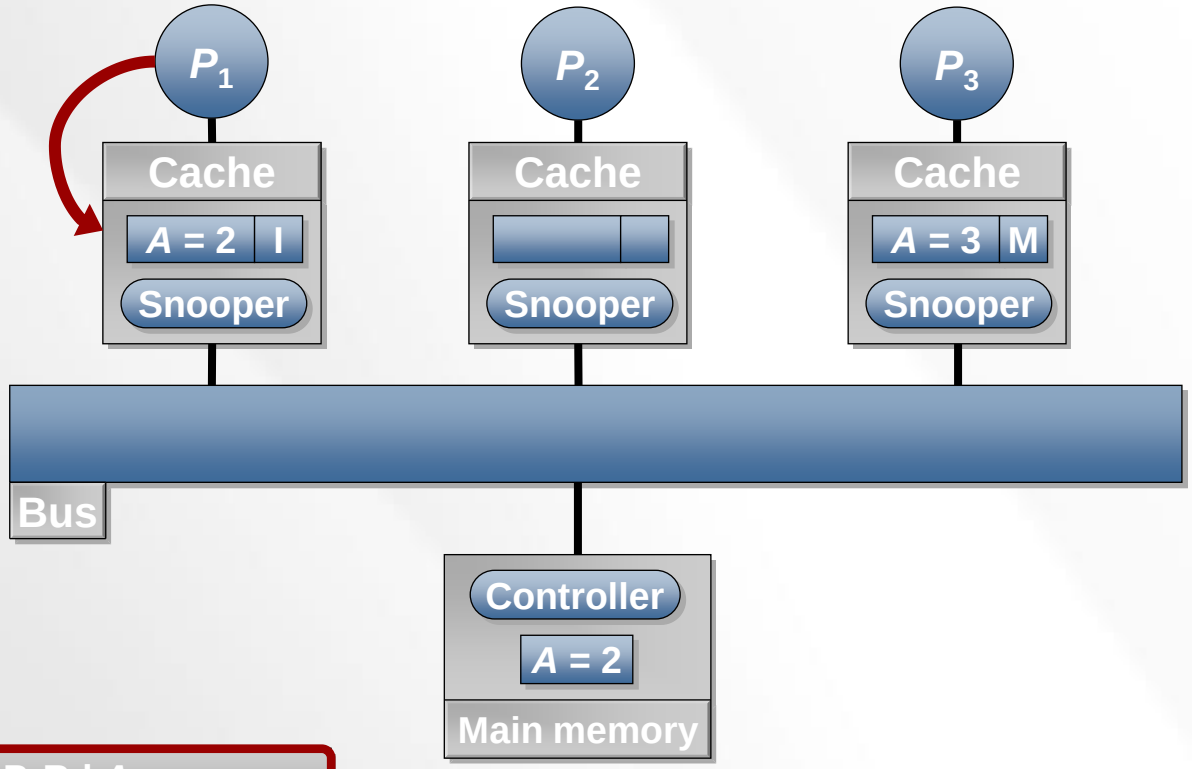


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Processor P_1 reads from its cache.



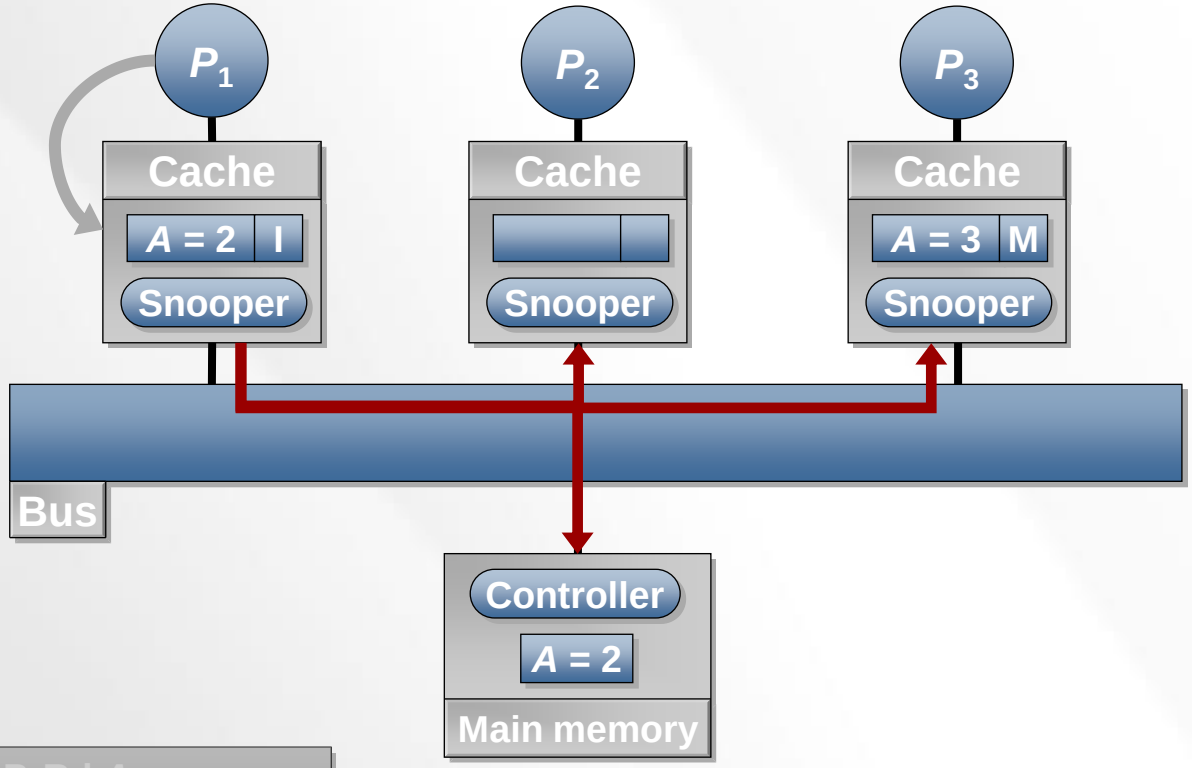
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	**PrRd A**
P_1	BusRd A
P_3	snoops BusRd
P_3	Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Processor P_1 issues a BusRd request.



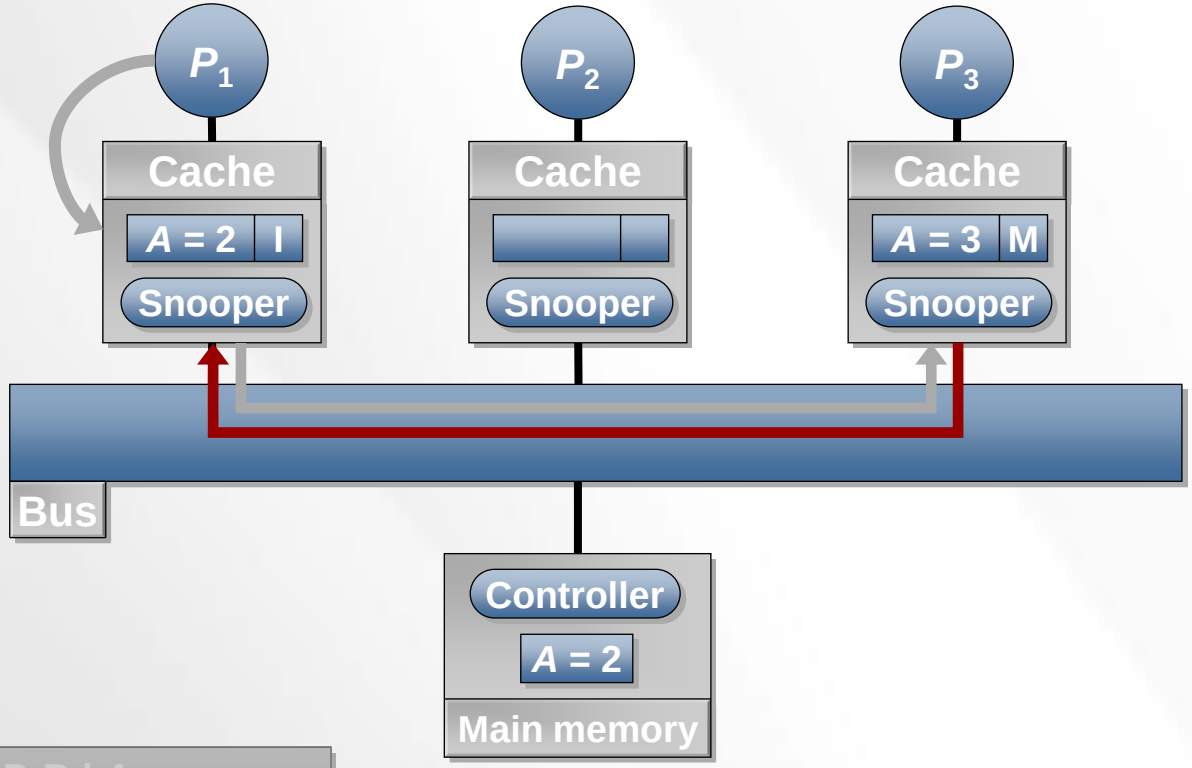
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrRd A
P_1 BusRd A
P_3 snoops BusRd
P_3 Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Processor P_3 snoops the BusRd.



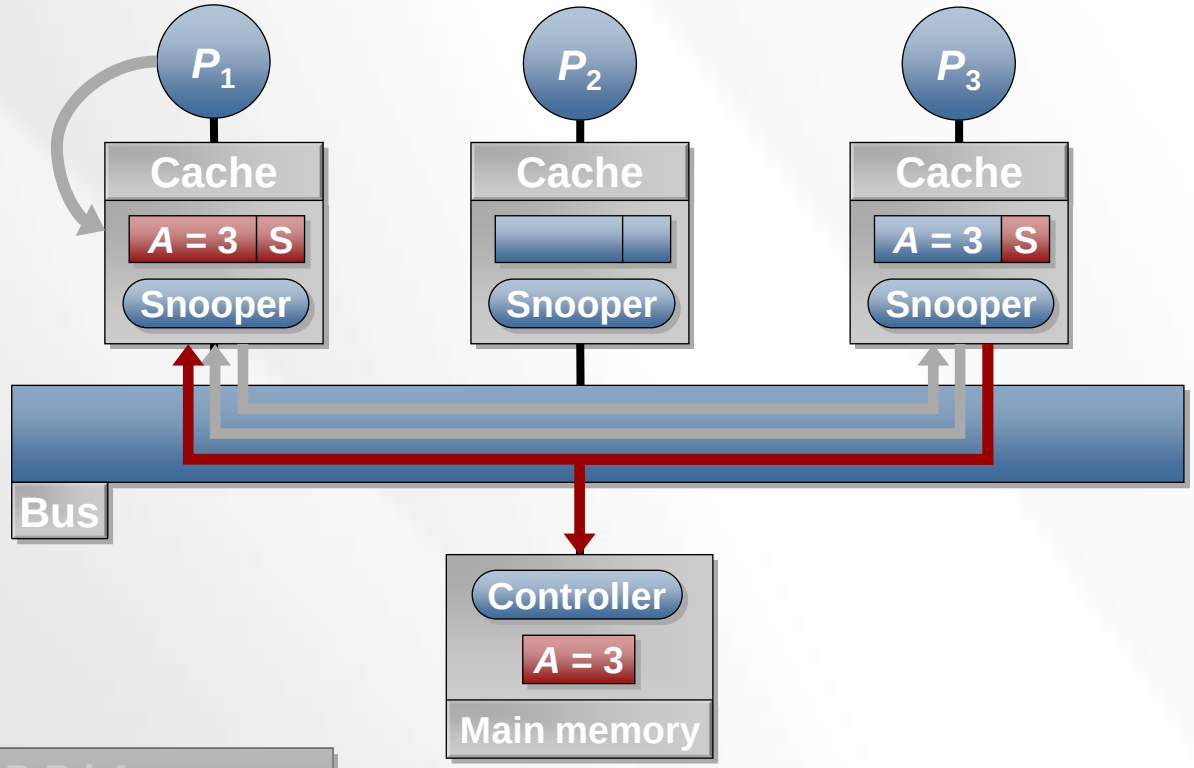
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrRd A
P_1 BusRd A
P_3 snoops BusRd
P_3 Flush

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Processor P_3 flushes, updating processor P_1, main memory and its own cache state.



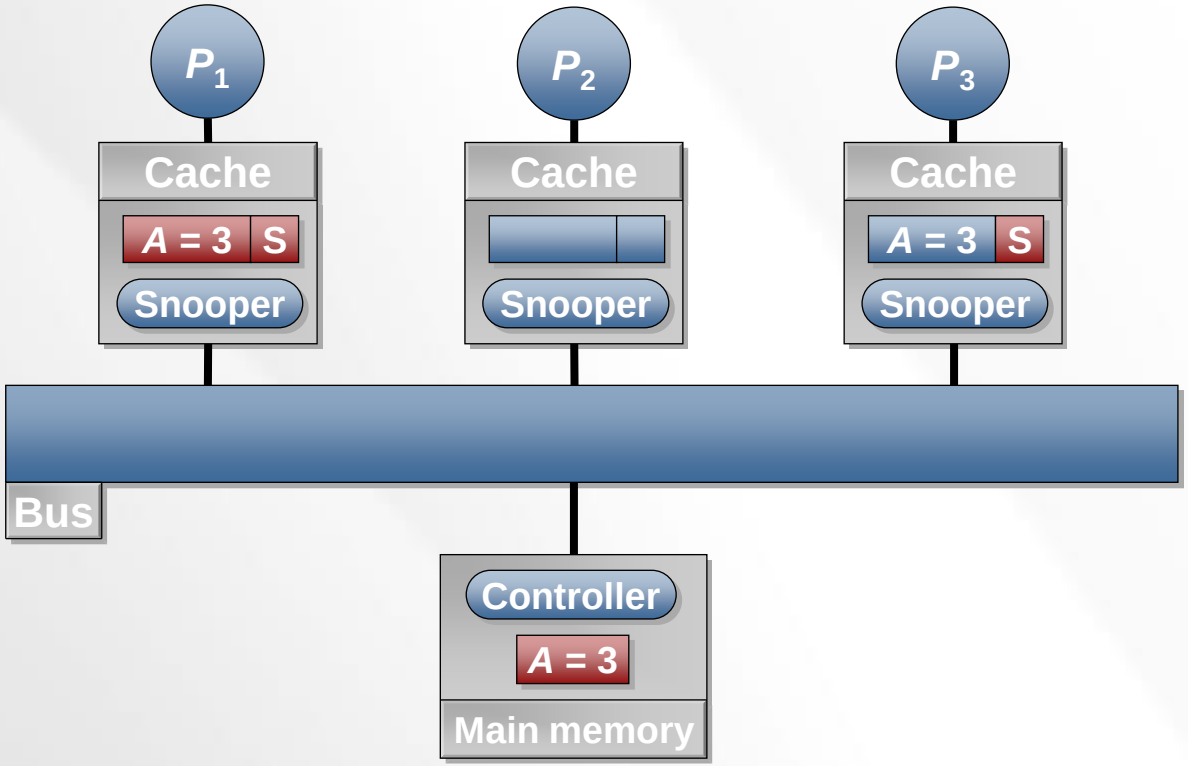
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
P_3	snoops BusRd
P_3	**Flush**

MSI	Firefly
MESI	Dragon

MSI: Processor P_1 Reads A

Read operation completes.

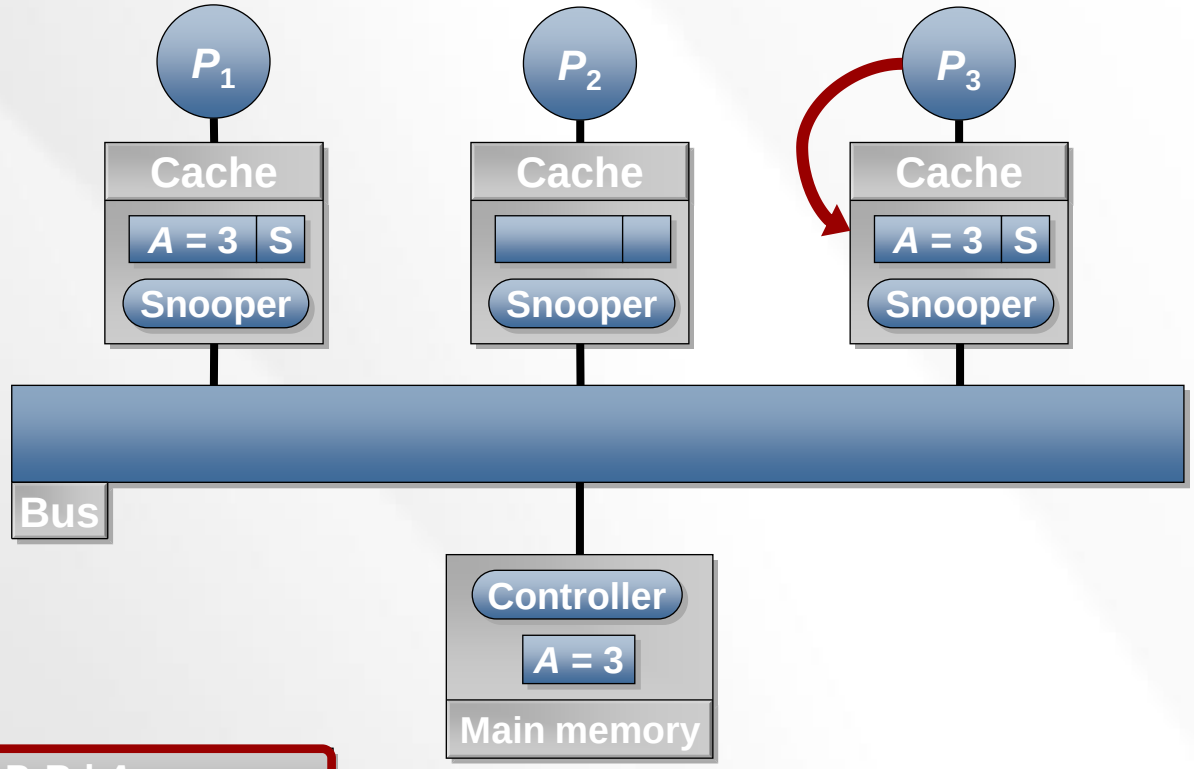


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_3 reads from its cache.



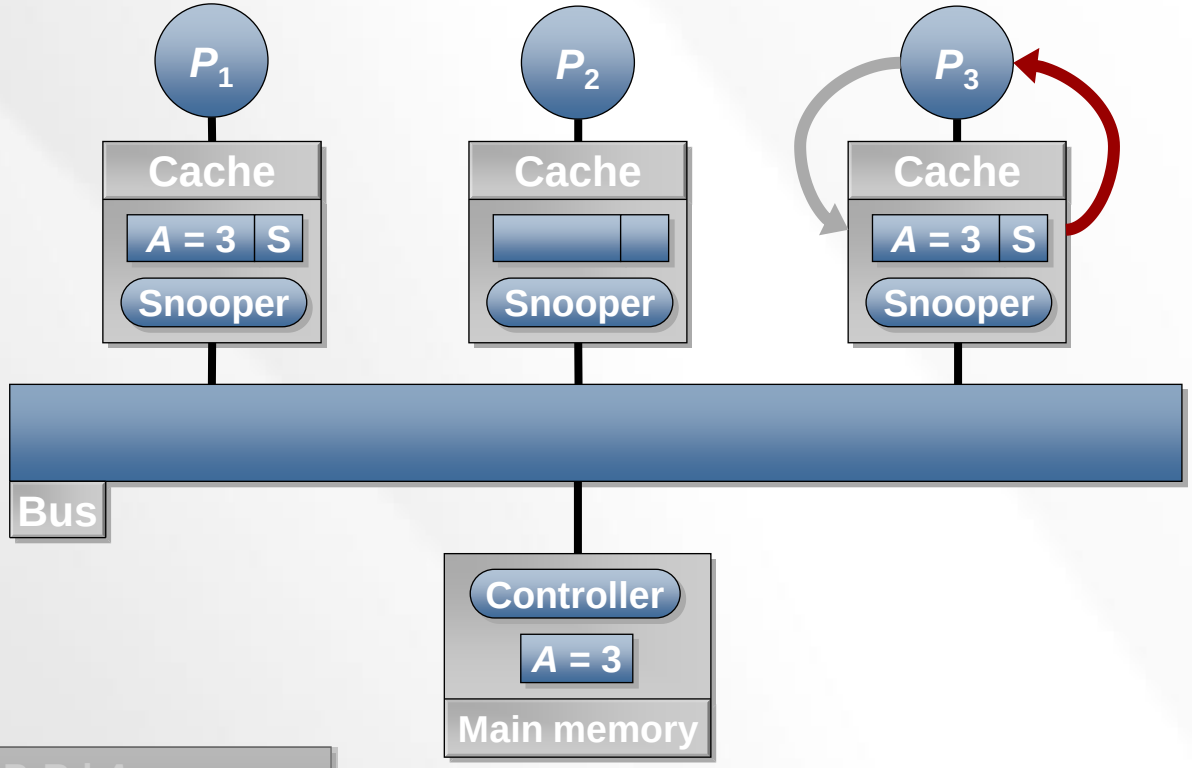
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3 PrRd A
P_3 returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Processor P_3 returns valid data from its cache.



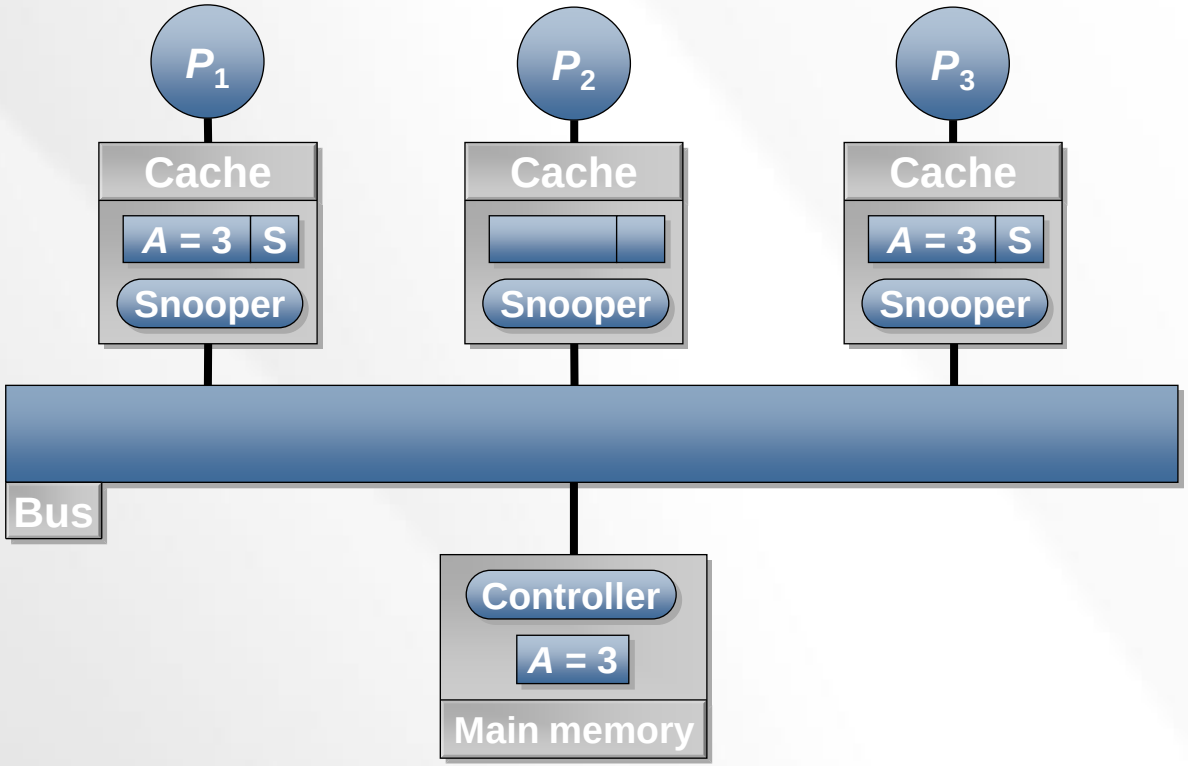
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3 PrRd A
P_3 returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_3 Reads A

Read operation completes.

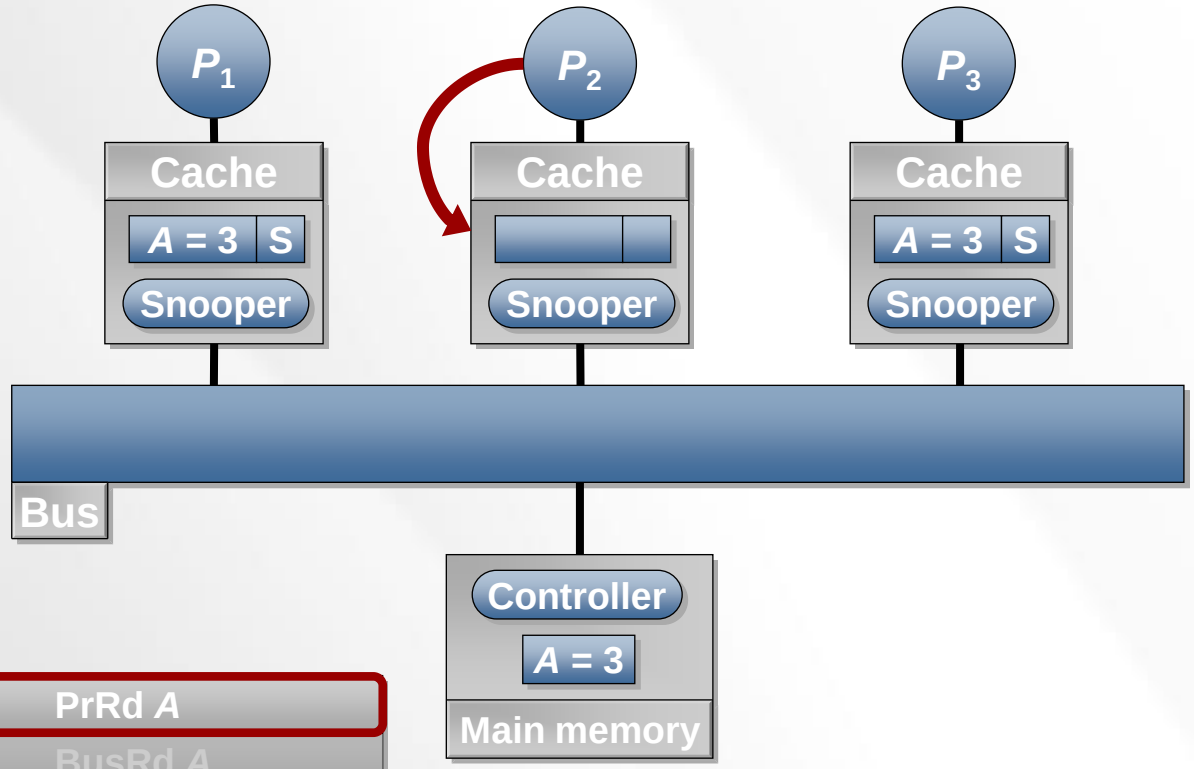


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI: Processor P_2 Reads A

Processor P_2 reads from its cache.



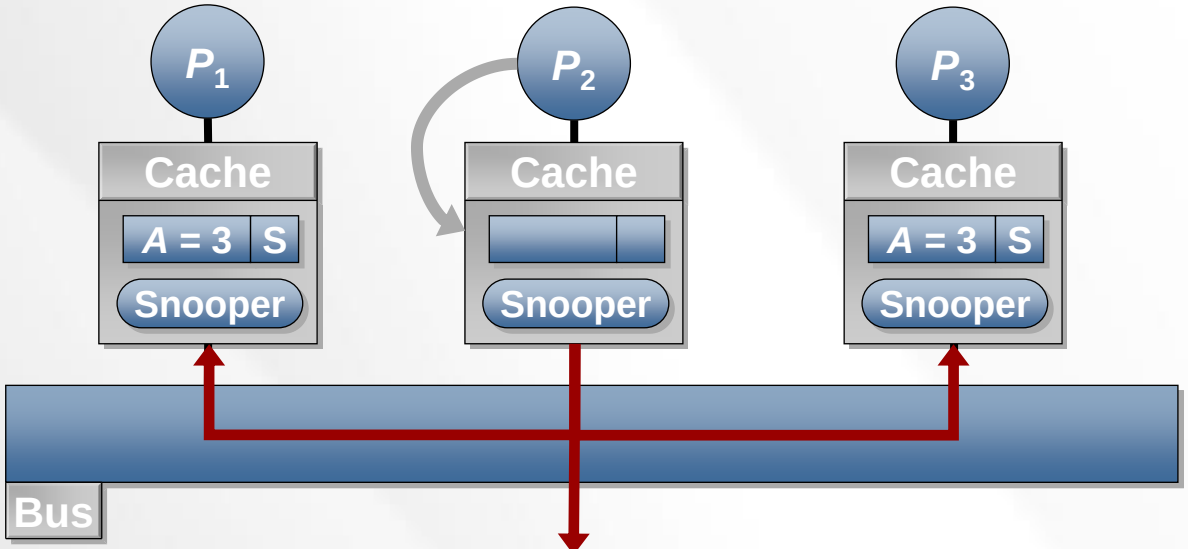
Trace	
P_1	Read A
P_1	Write A = 2
P_3	Read A
P_3	Write A = 3
P_1	Read A
P_3	Read A
P_2	Read A

P_2	PrRd A
P_2	BusRd A
MemCntr	observes BusRd
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_2 Reads A

Processor P_2 issues a BusRd request.



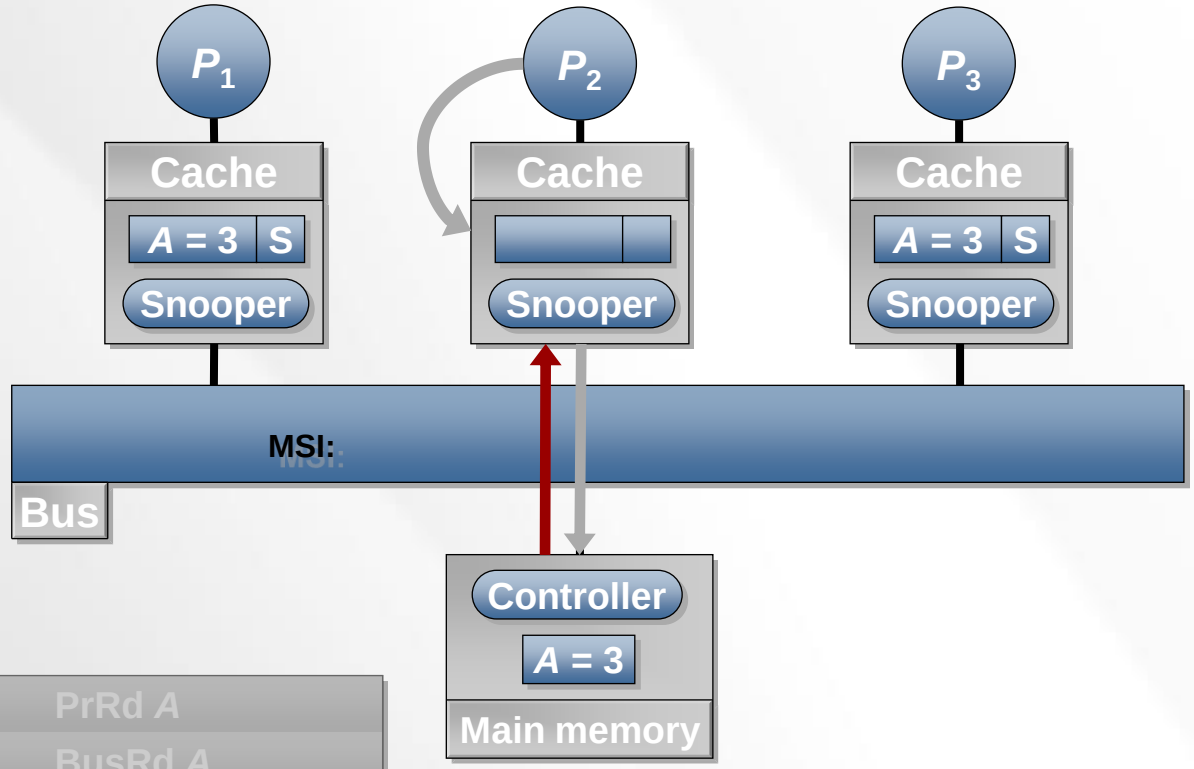
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_2	PrRd A
P_2	BusRd A
MemCntr	observes BusRd
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_2 Reads A

Main memory controller observes the BusRd.



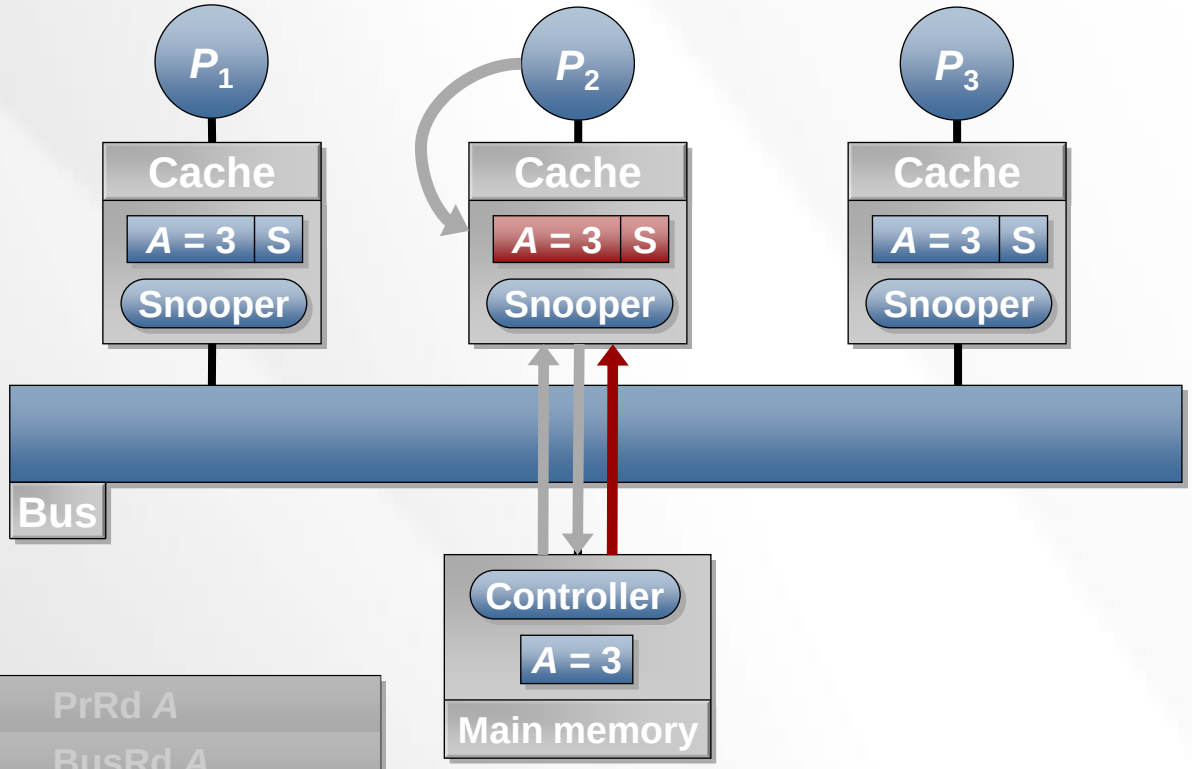
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_2	PrRd A
P_2	BusRd A
MemCntr	observes BusRd
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_2 Reads A

Main memory returns valid data.



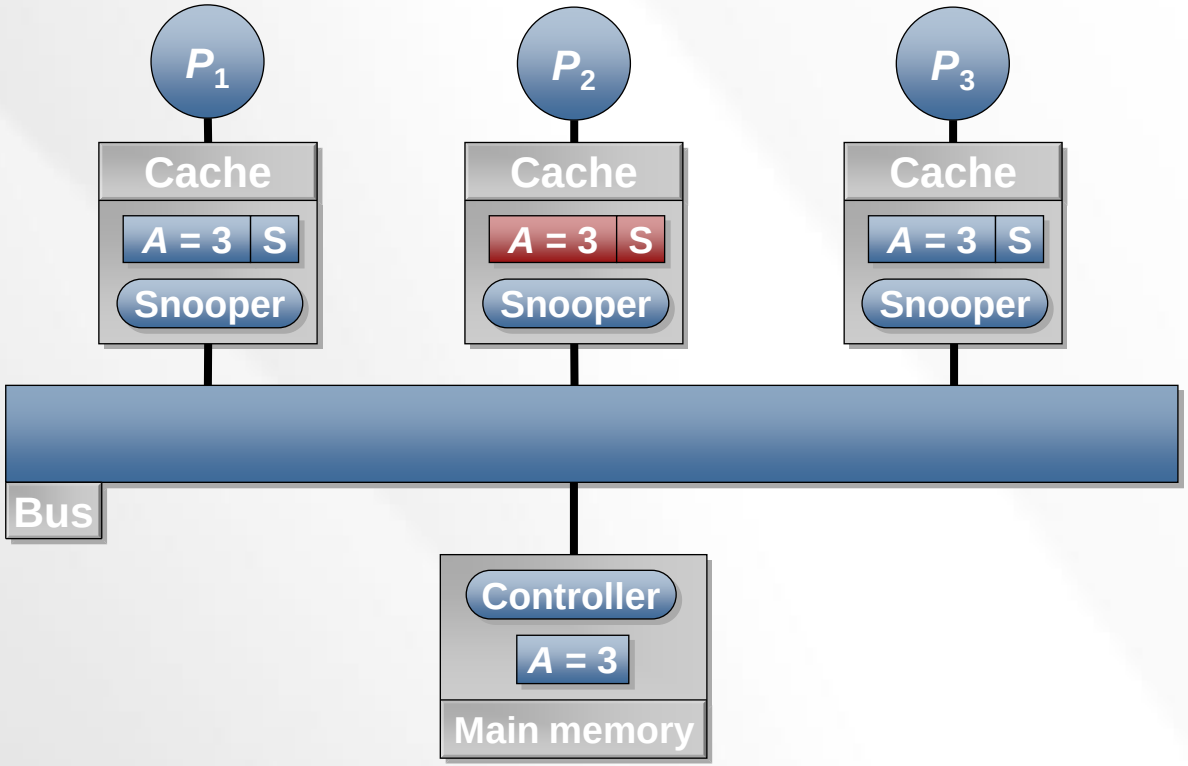
Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_2	PrRd A
P_2	BusRd A
MemCntr	observes BusRd
Mem	returns data

MSI	Firefly
MESI	Dragon

MSI: Processor P_2 Reads A

Operation completes.



Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

MSI	Firefly
MESI	Dragon

MSI Example: Rd/Wr to a single line

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	S	–	–	BusRd	Mem
W1	M	–	–	BusRdX*	Mem
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX*	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own Cache
R2	S	S	S	BusRd	Mem

*or, BusUpgr (data from own cache)

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush?

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush? Because it is a read with intention to write, as opposed to write.
  - Thus, there is a possibility for a read before the write is performed.
  - In addition, the write could be to a different word in the line (so the whole line needs to be flushed).

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush? Because it is a read with intention to write, as opposed to write.
  - Thus, there is a possibility for a read before the write is performed.
  - In addition, the write could be to a different word in the line (so the whole line needs to be flushed).
- In case of a write to a shared block:
  - Cache already has latest data; can use upgrade (BusUpgr) instead of BusRdX
- Replacement changes state of two blocks: outgoing and incoming
- Flush has to modify both caches and main memory

Note: Coherence granularity is *u* (a single line). What happens when all the reads go to word 0 on line *u*, but write by P3 goes to word 1 on line *u*? False-sharing miss on the 2nd R1

MSI: Coherence and SC

- Coherence:
  - Write propagation:
    - through invalidation, and flush on subsequent BusRds
  - Write serialization?
    - Writes (BusRdX) that go to the bus appear in bus order (and handled by snoopers in bus order!)
    - Writes that do not go to the bus?
      - Only happen when the line state is M, i.e. when I am the only processor holding the line. Local writes are only visible to me, so they are serialized.
- To enforce SC:
  - Program order: enforced by following the bus transaction order
    - All writes appear on the bus
    - All local writes (within 1 processor) can follow program order
  - Write completion: Occurs when write appears on bus
  - Write atomicity: A read returns the latest value of a write. At that time, the value is visible to all others (on a bus transaction, or on a local write).

Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

Lower-Level Protocol Choice

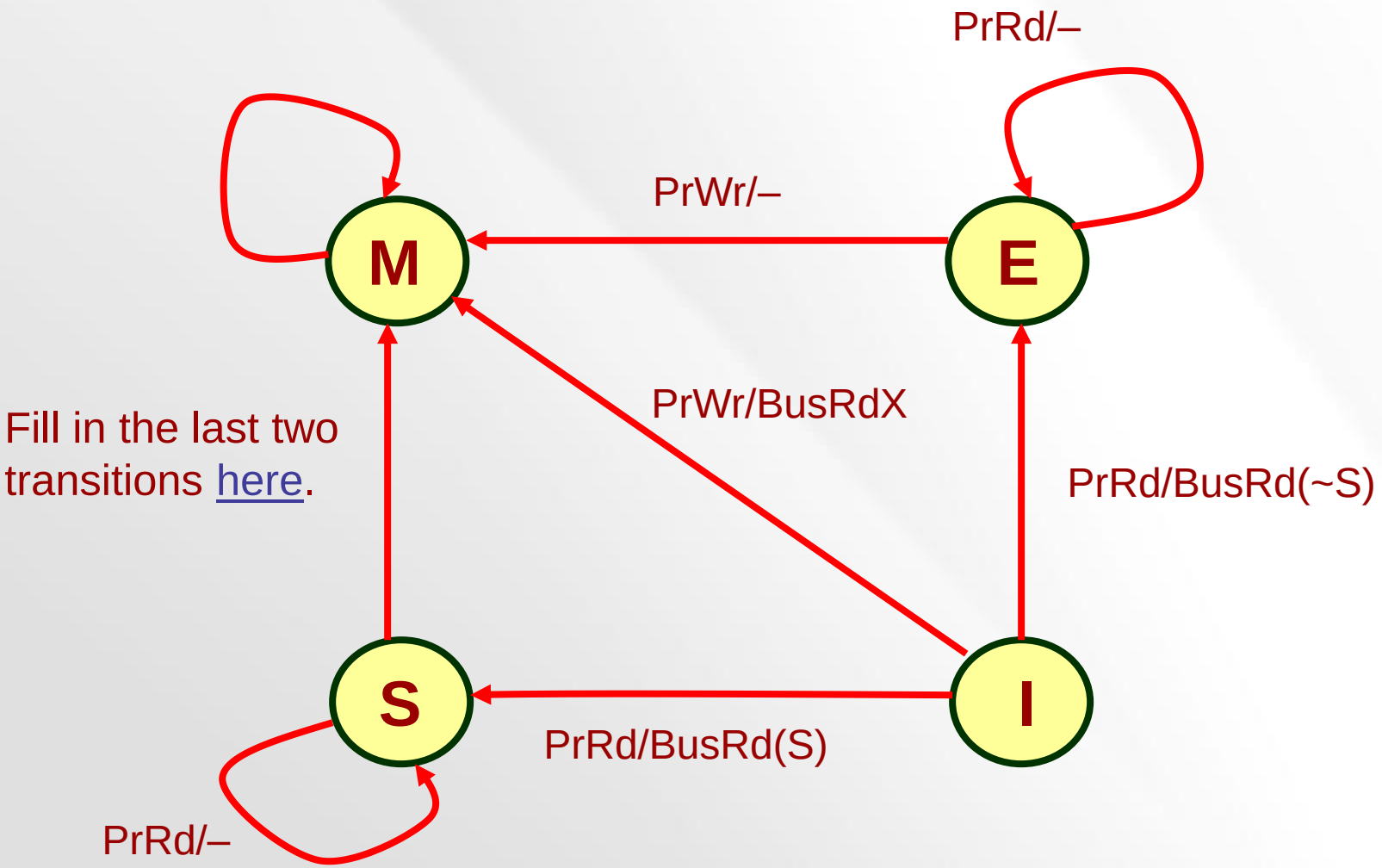
- What transition should occur when a BusRd is observed in state M?
  - Should the state change to S or to I?

MESI (4-state) Invalidation Protocol

- Here's a problem with the MSI protocol:
  - A {Rd, Wr} sequence causes two bus transactions
    - BusRd (I → S) followed by BusRdX or BusUpgr (S → M)
    - even when no one is sharing (e.g., serial program!)
    - *In general, coherence traffic from serial programs is unacceptable*
- To avoid this, add a fourth state, Exclusive:
    - Invalid
    - Modified (dirty)
    - Shared (two or more caches may have copies)
    - Exclusive (only this cache has clean copy, same value as in memory)

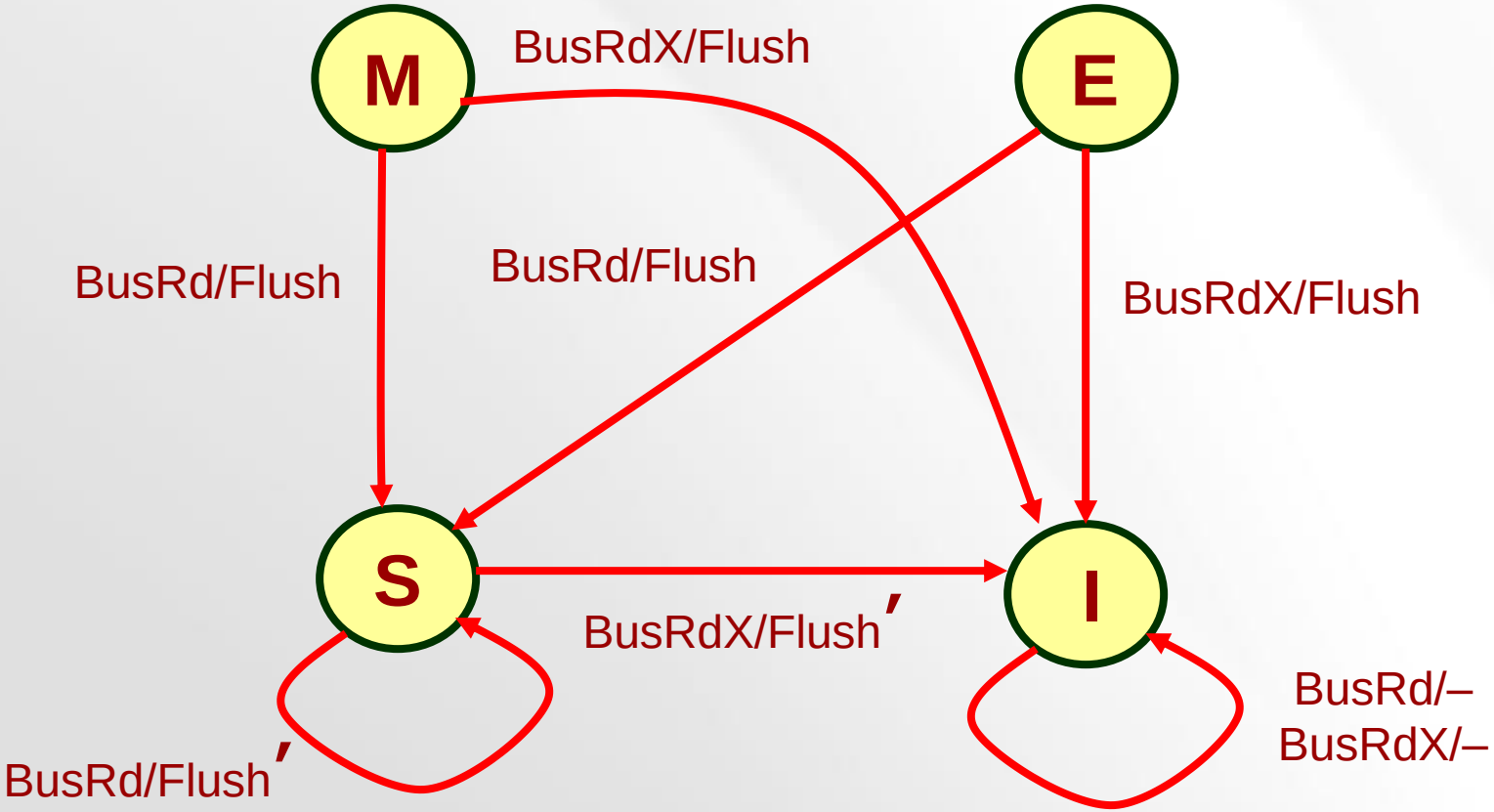
- How does the protocol decide whether I → E or I → S?
  - Need to check whether someone else has a copy
  - "Shared" signal on bus: wired-or line asserted in response to BusRd

MESI: Processor-Initiated Transactions

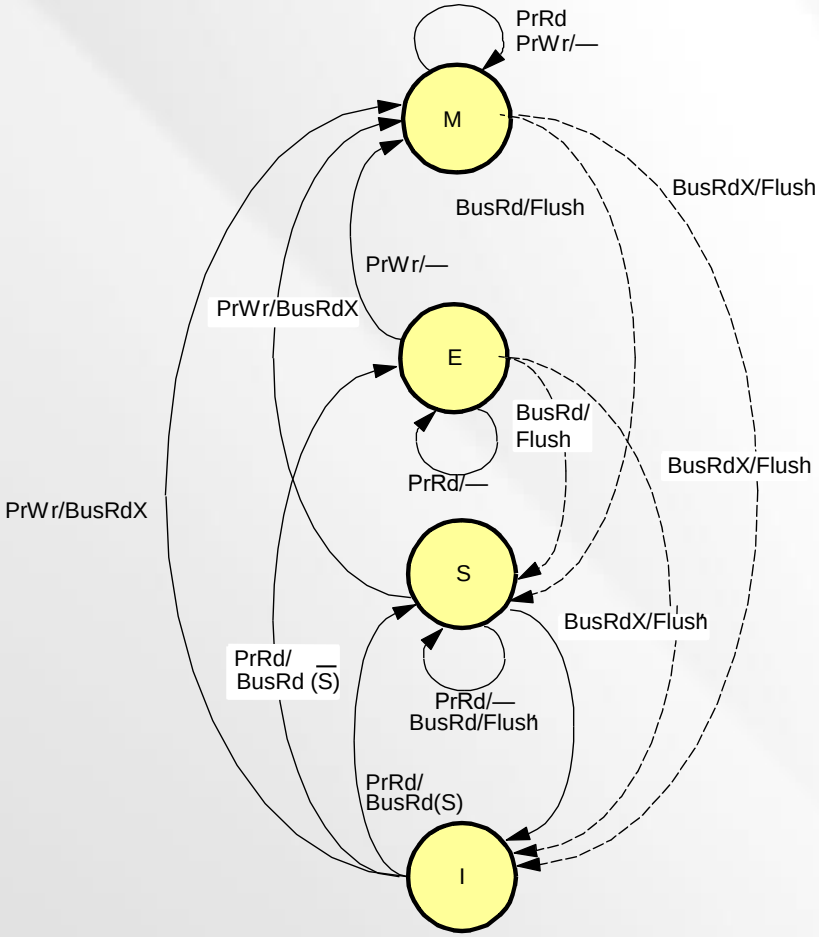


MESI: Bus-Initiated Transactions

Flush′ means flush only if cache-to-cache sharing is used; only the cache responsible for supplying the data will do a flush.



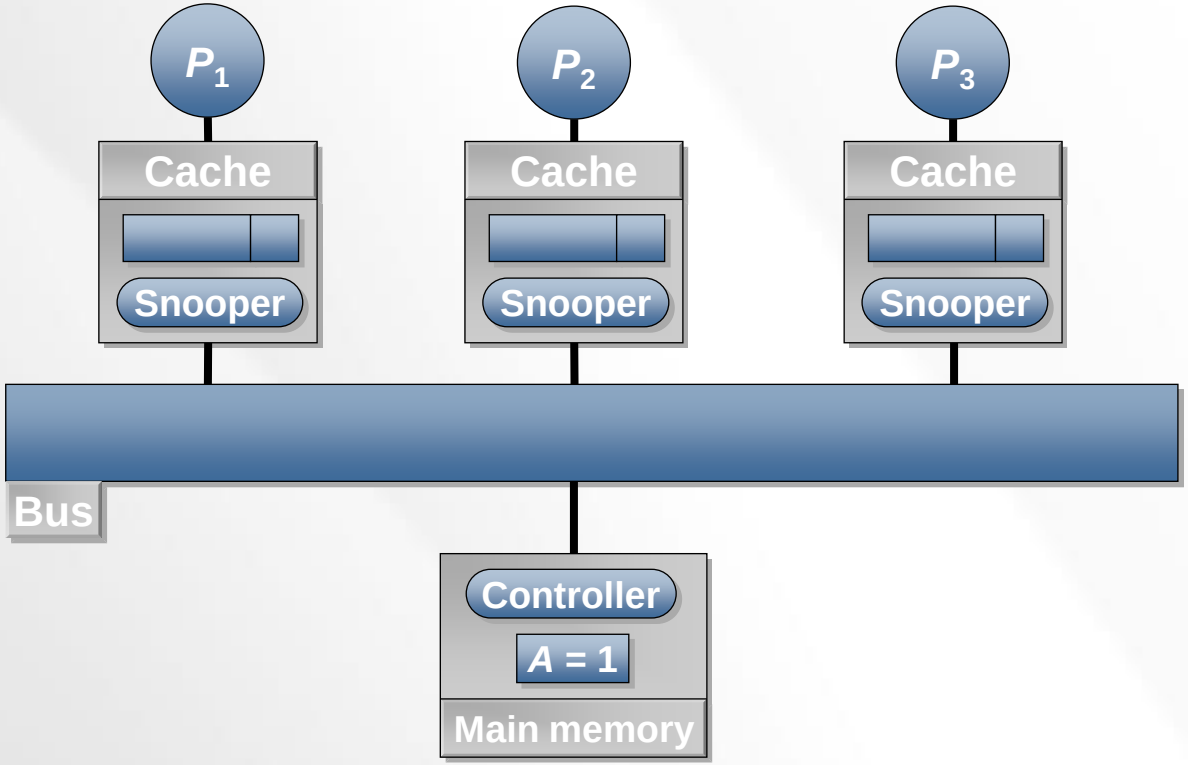
MESI State Transition Diagram



- BusRd(S) means shared line asserted on BusRd transaction

MESI Visualization

Start state. All caches empty and main memory has $A = 1$.

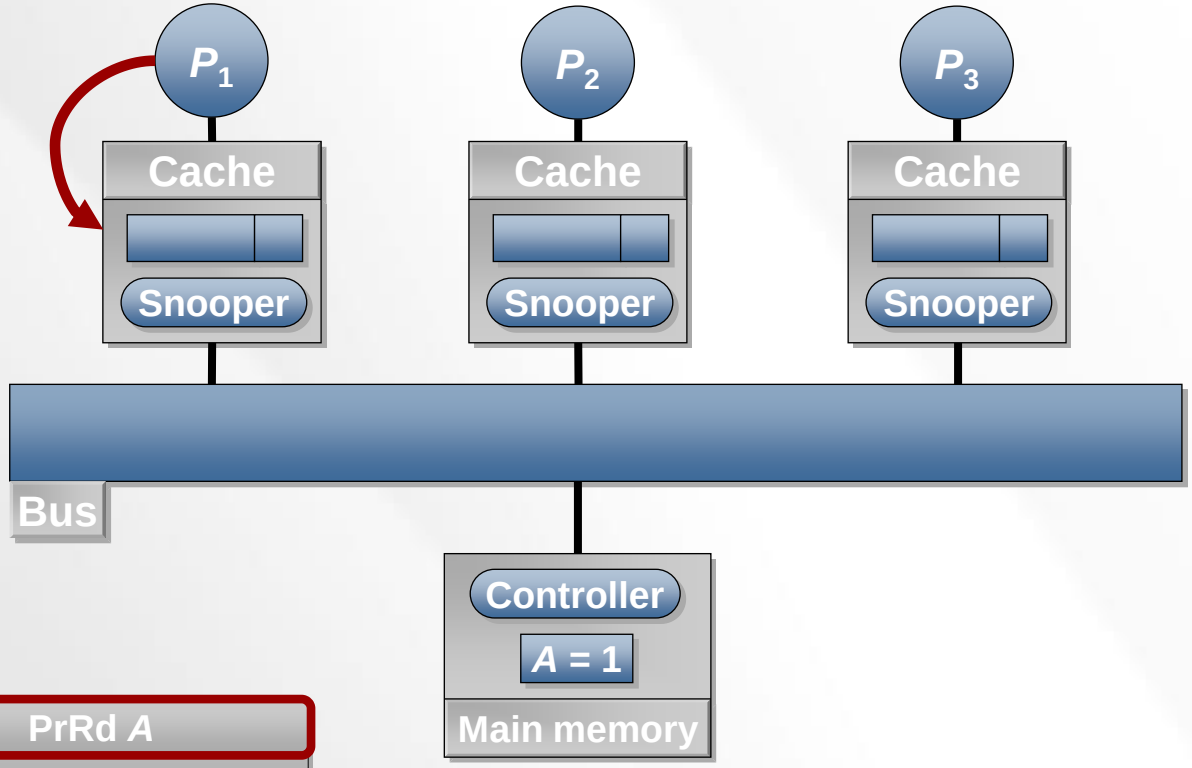


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 attempts to read A from its cache.



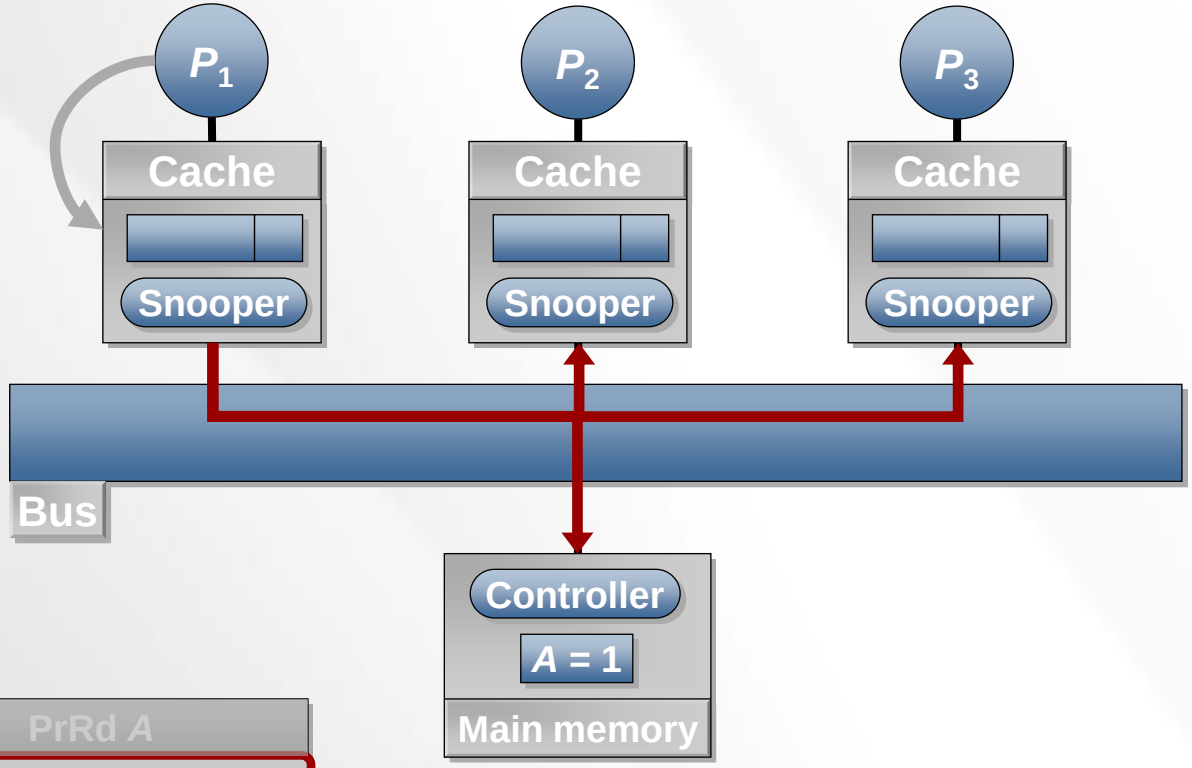
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 issues a BusRd.



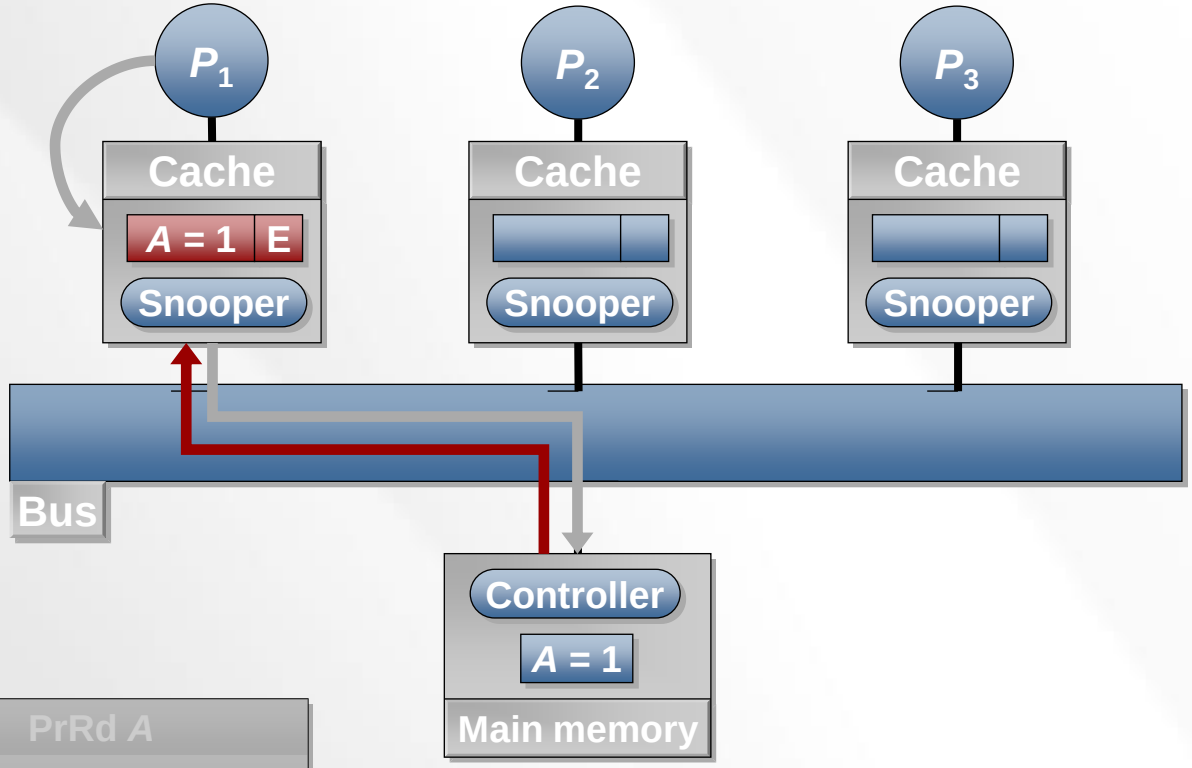
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Main memory returns data to processor P_1 which updates its cache.



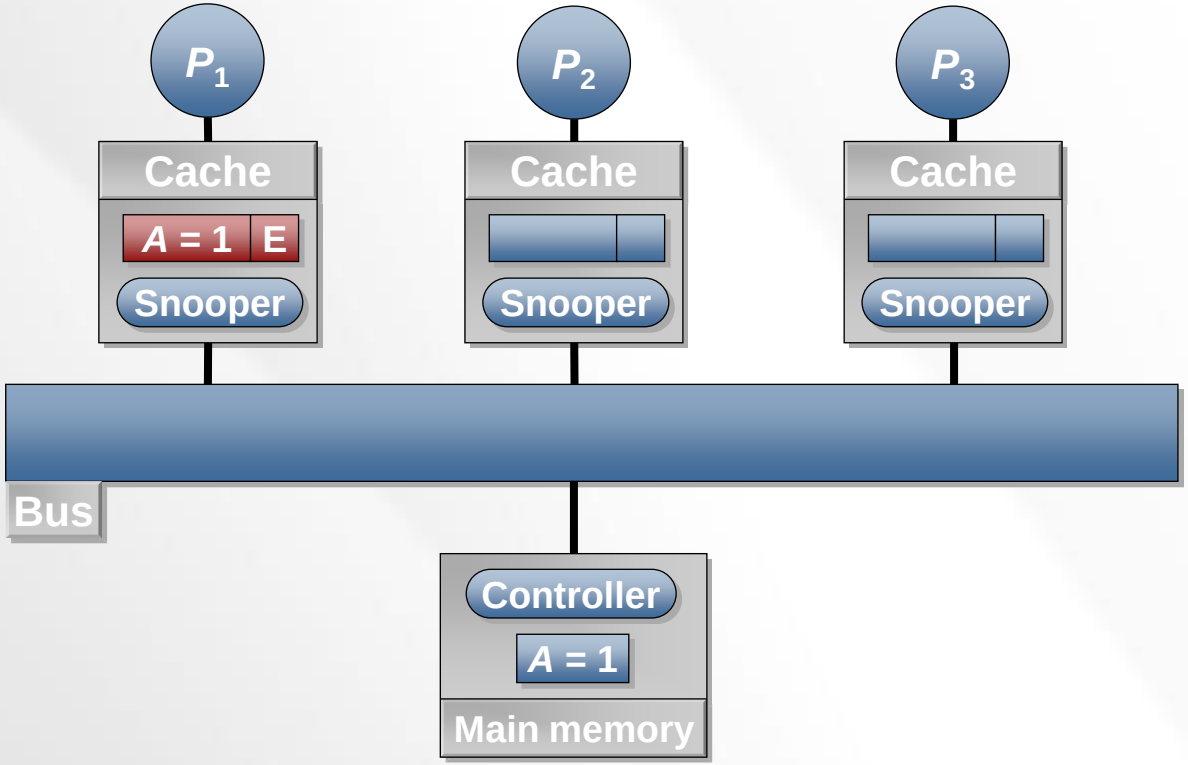
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Read operation completes.

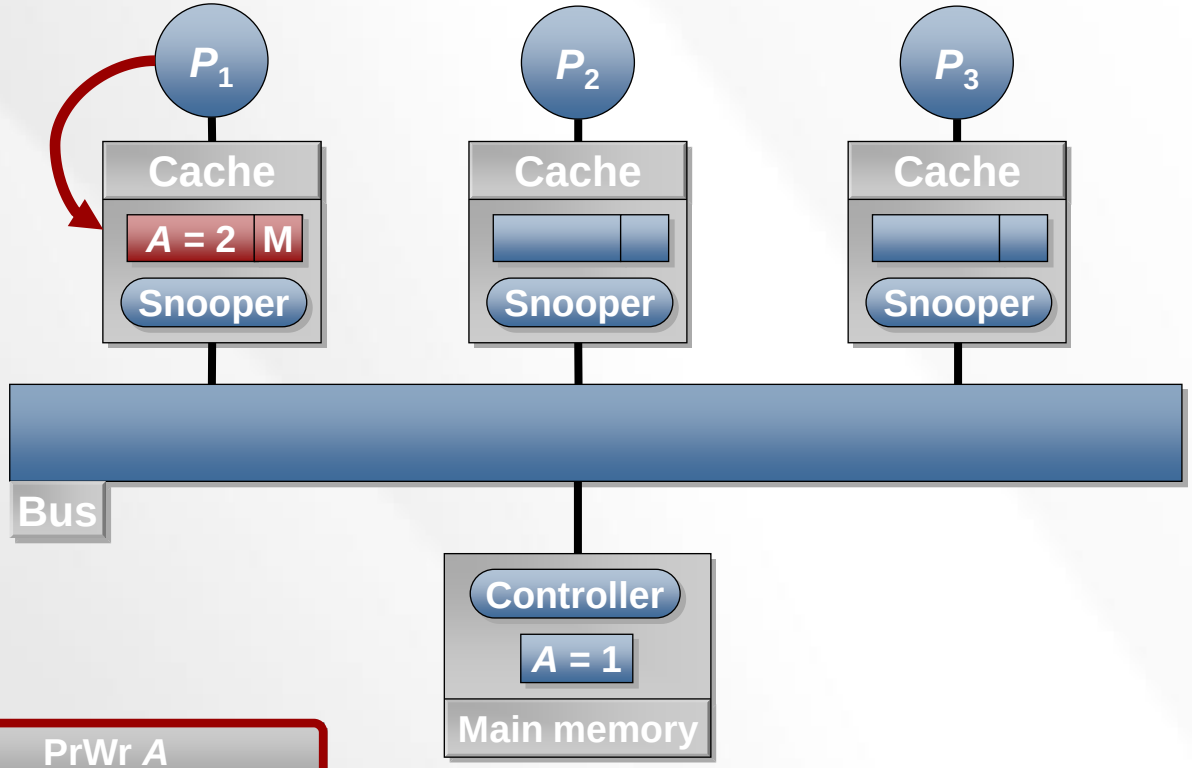


Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Processor P_1 writes to its cache.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

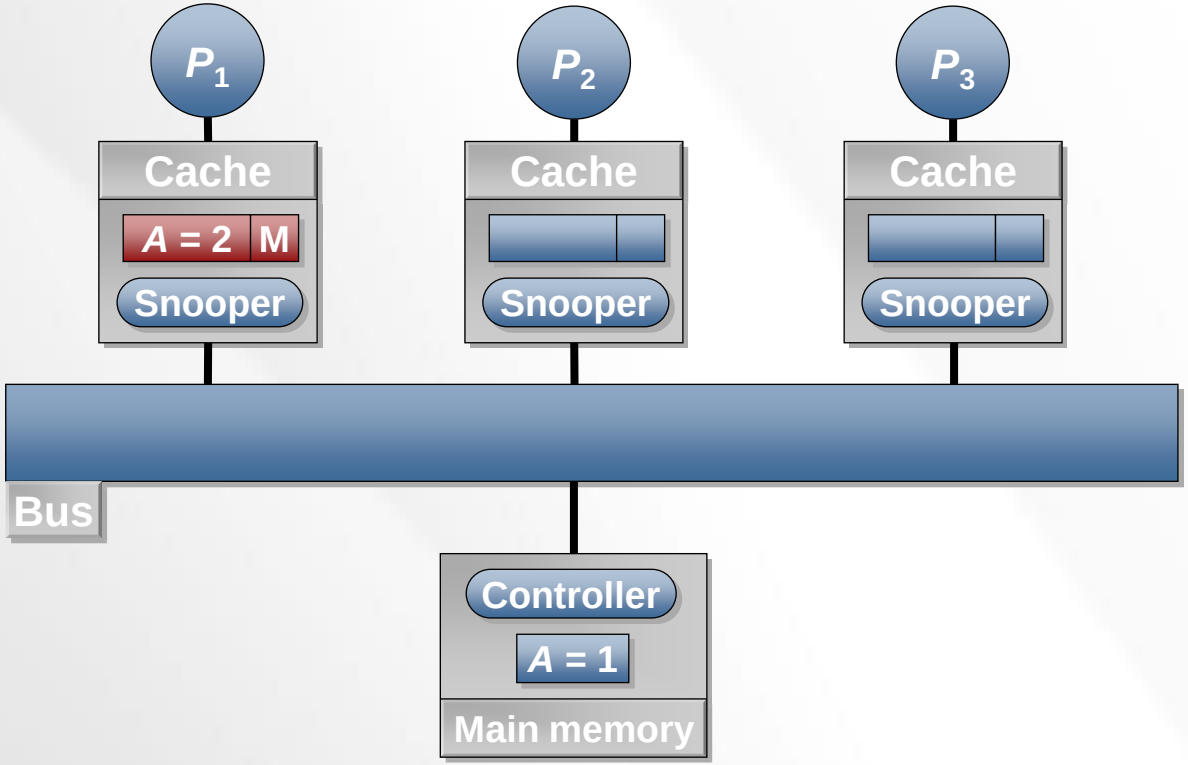
P_1 PrWr A

One less bus request due to Exclusive state, esp. for serial programs

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Write operation completes.

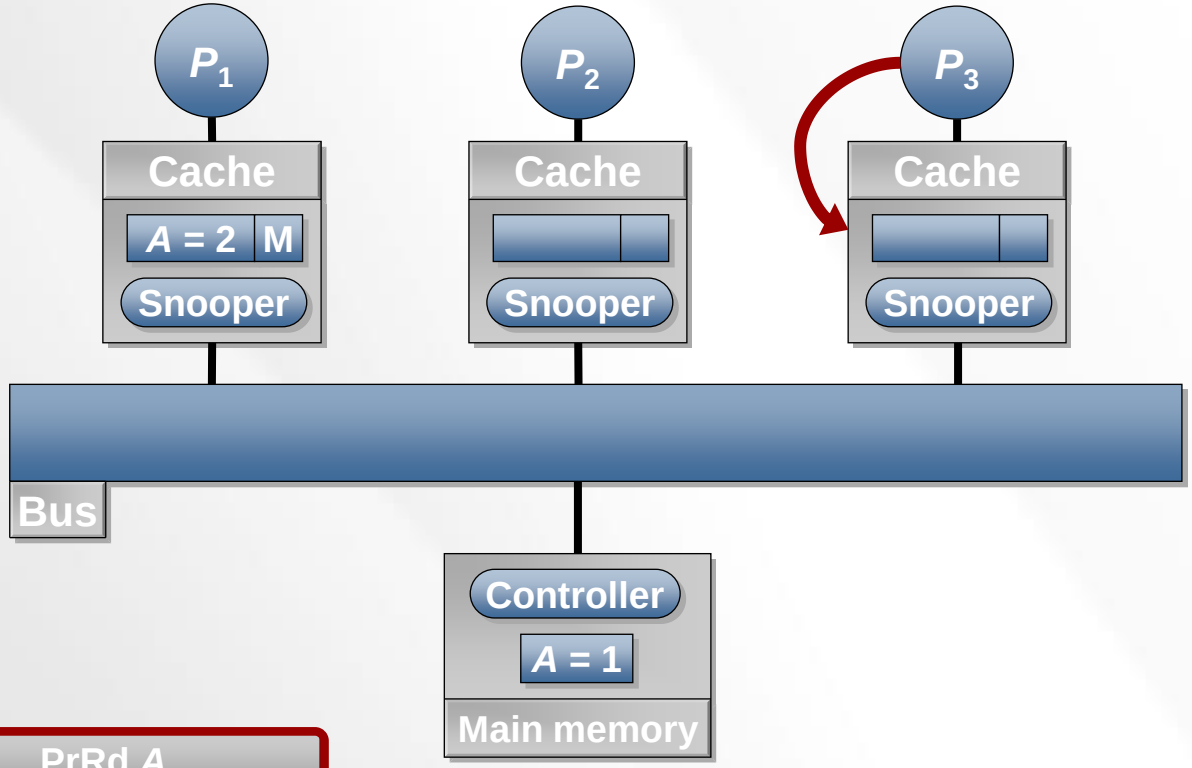


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 attempts to read A from its cache.



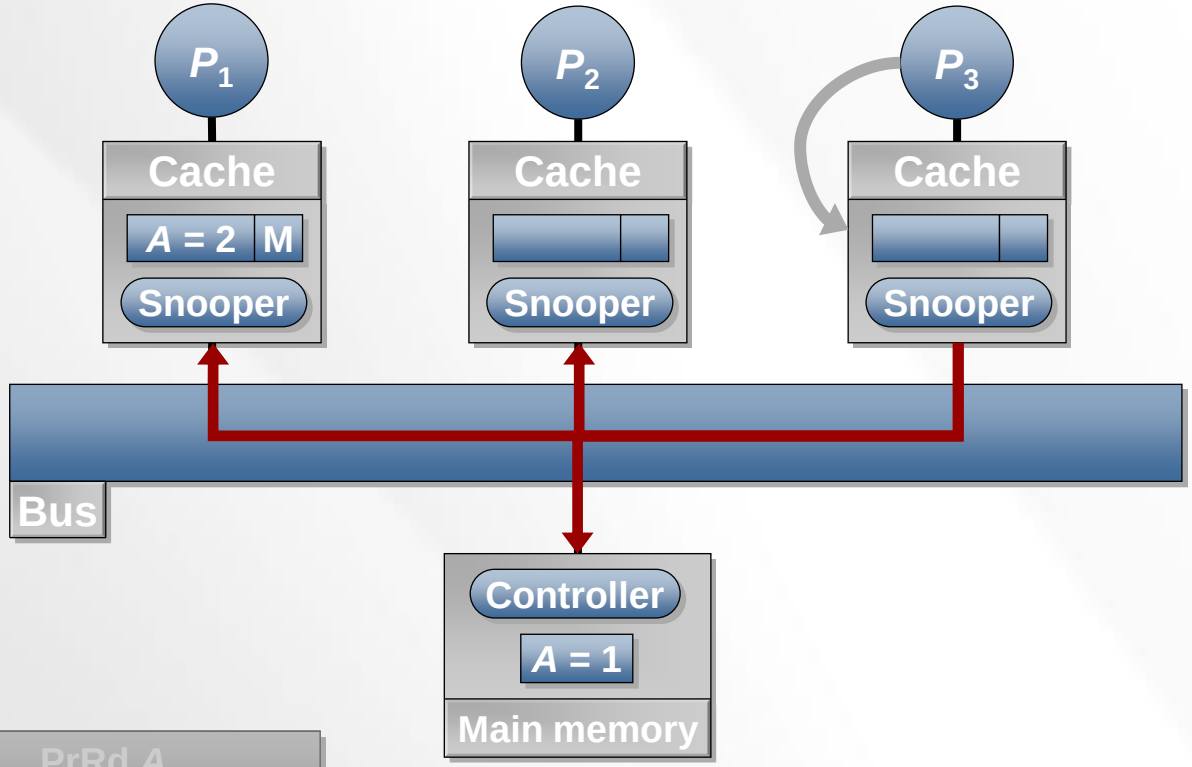
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 issues a BusRd.



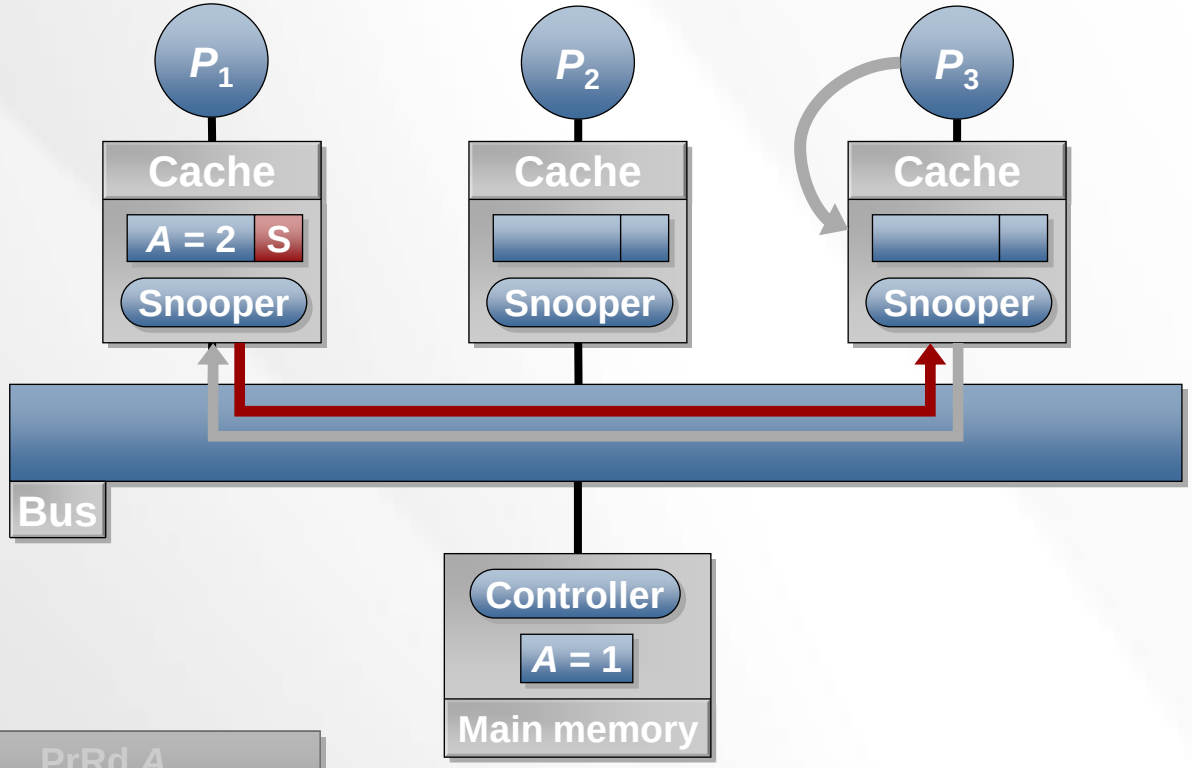
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_1 snoops the BusRd from processor P_3.



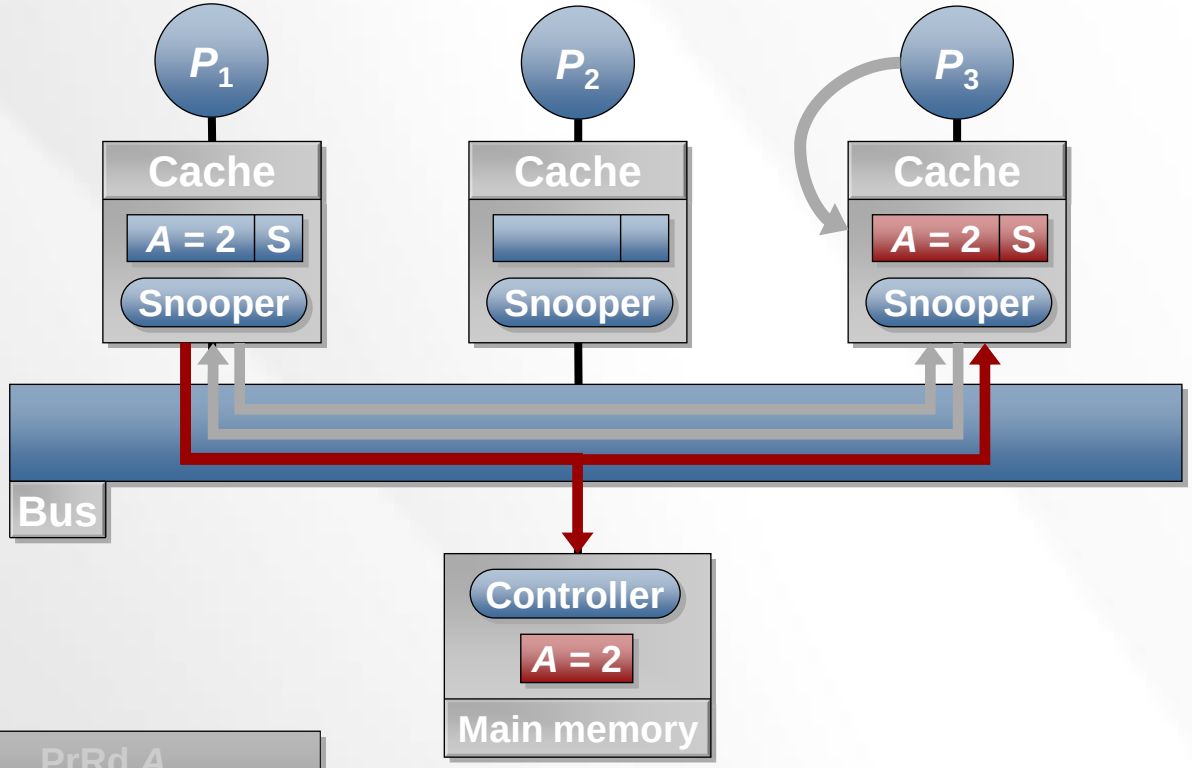
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_1 flushes, sending updated data to P_3 and main memory.



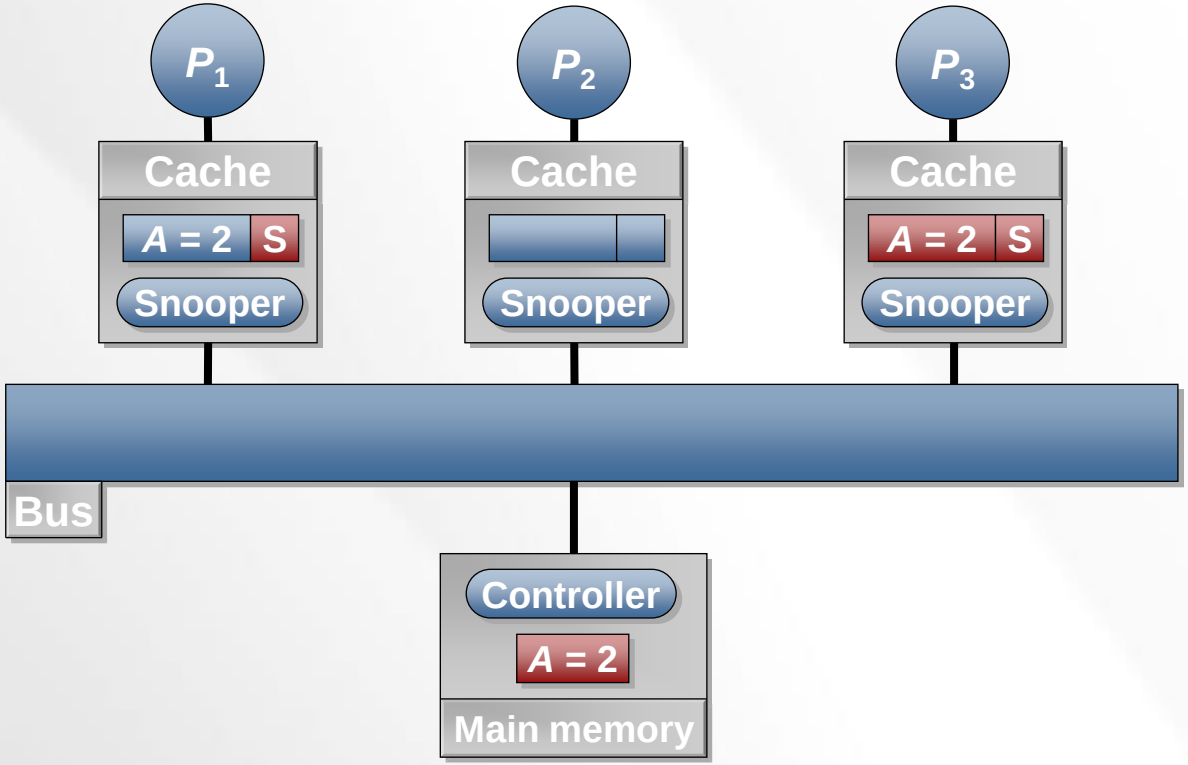
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Read operation completes.

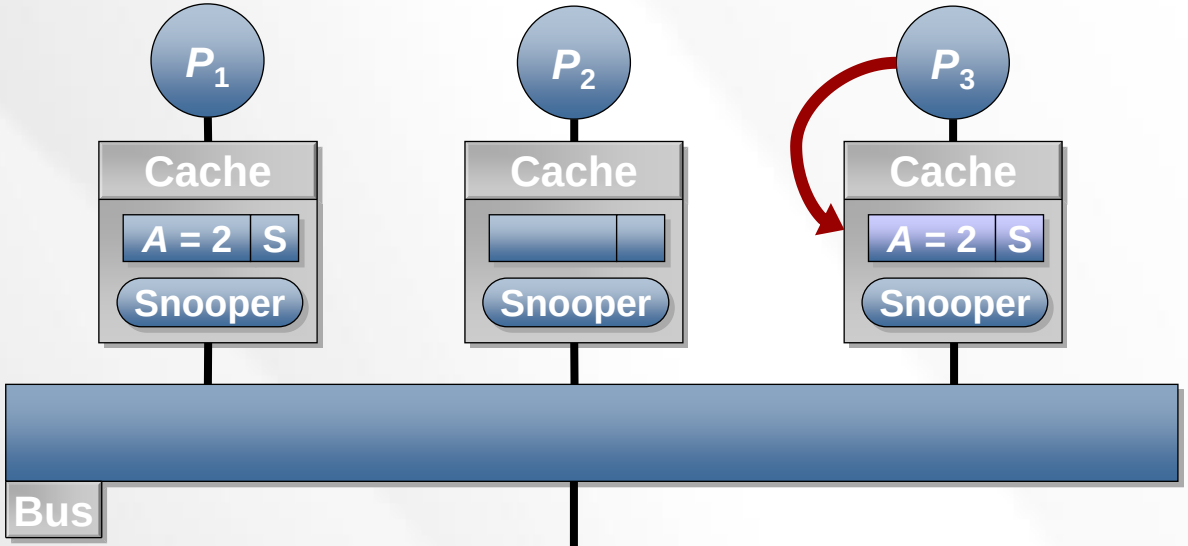


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 writes to its cache.



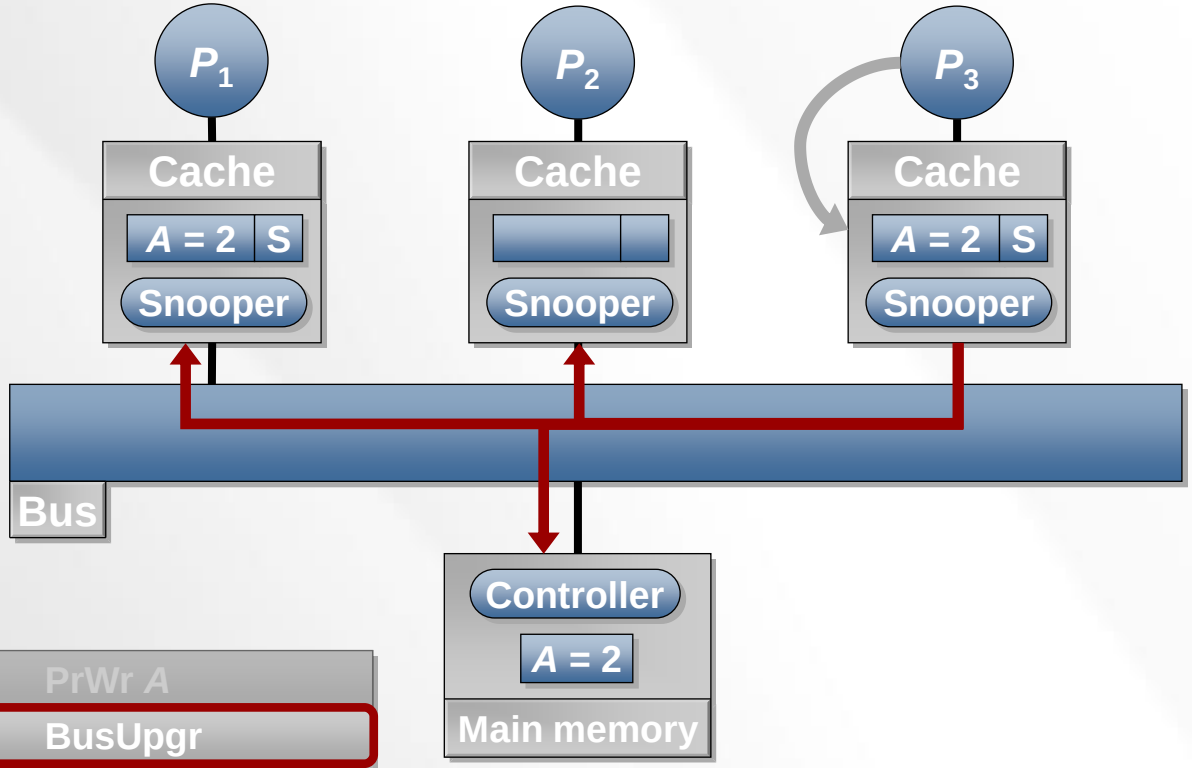
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A	
P_3	BusUpgr	
P_1	snoops BusUpgr	

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 issues a BusUpgr request.



Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

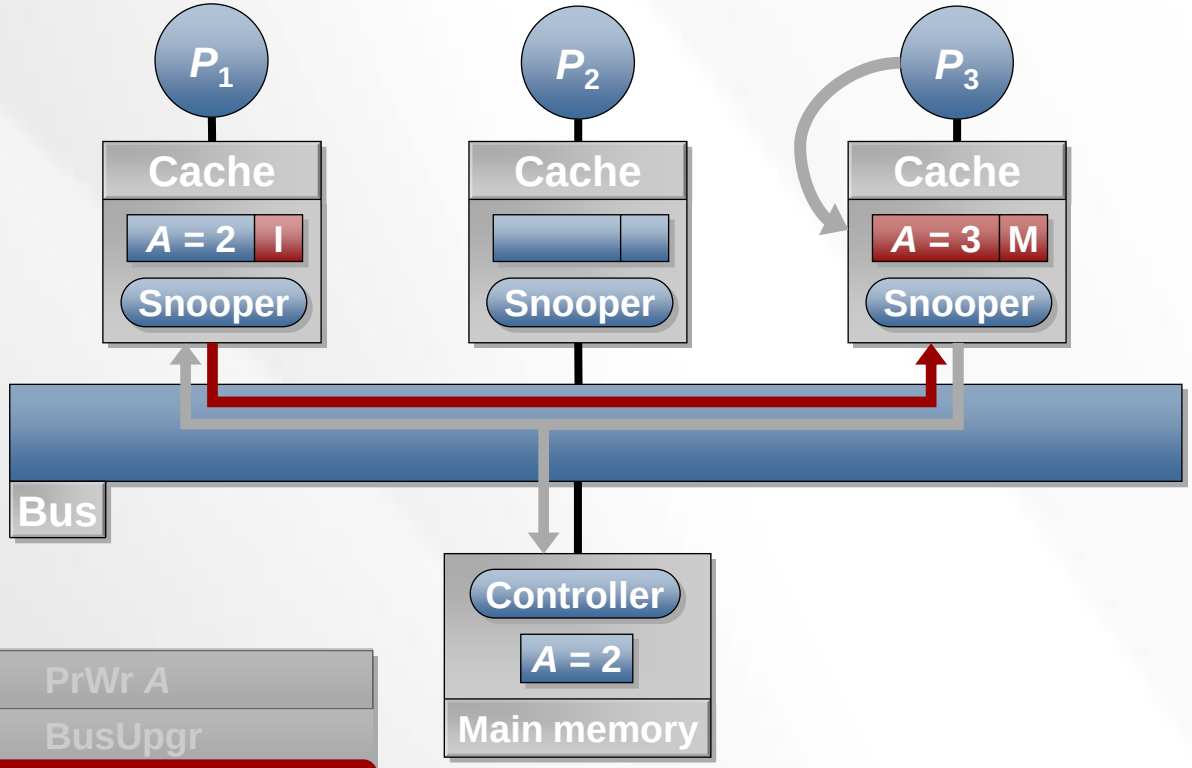
P_3	PrWr A
P_3	BusUpgr
P_1	snoops BusUpgr

Note: BusUpgr used instead of BusRdX

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_1 snoops the BusRd and invalidates its cache.



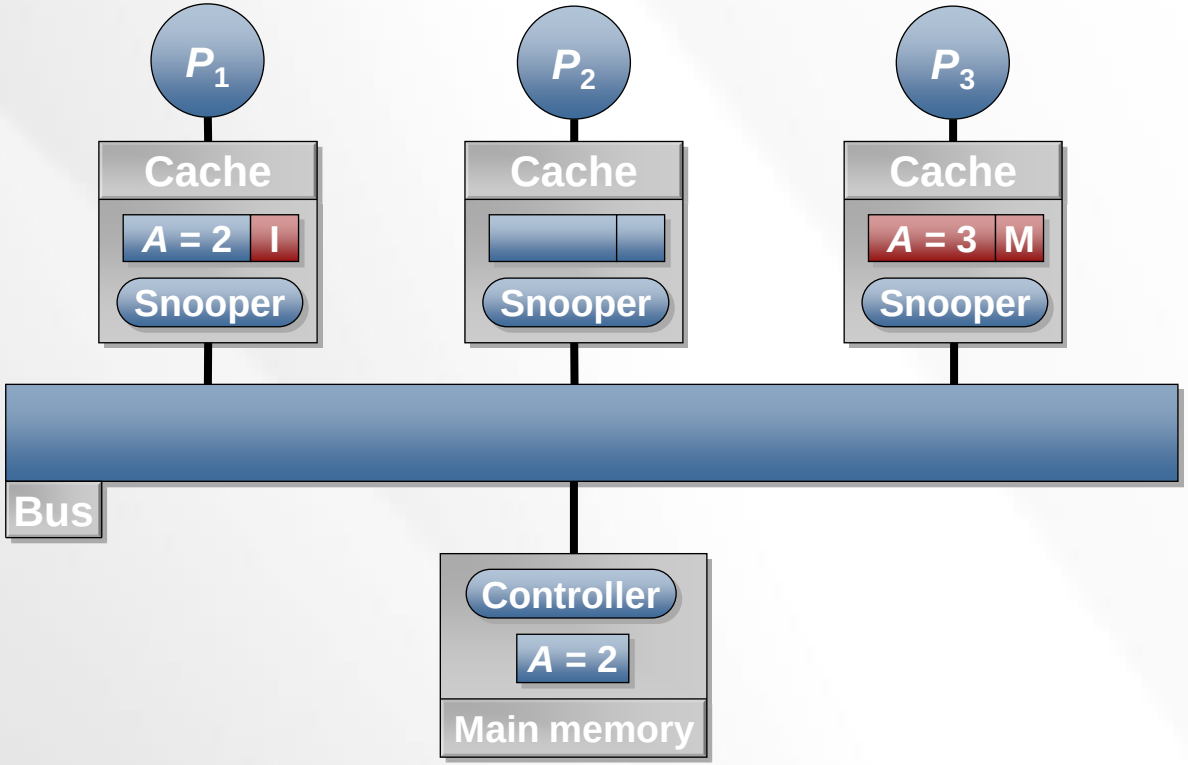
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A
P_3	BusUpgr
P_1	snoops BusUpgr

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Write operation completes.

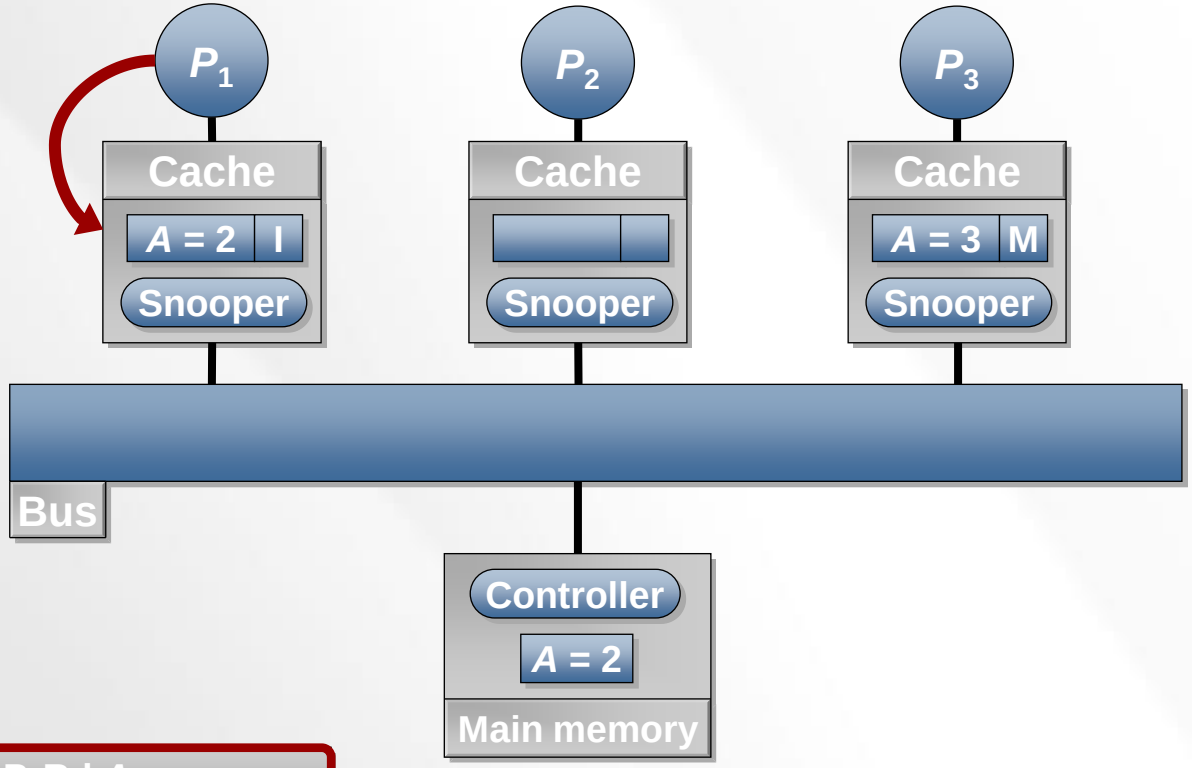


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 reads from its cache.



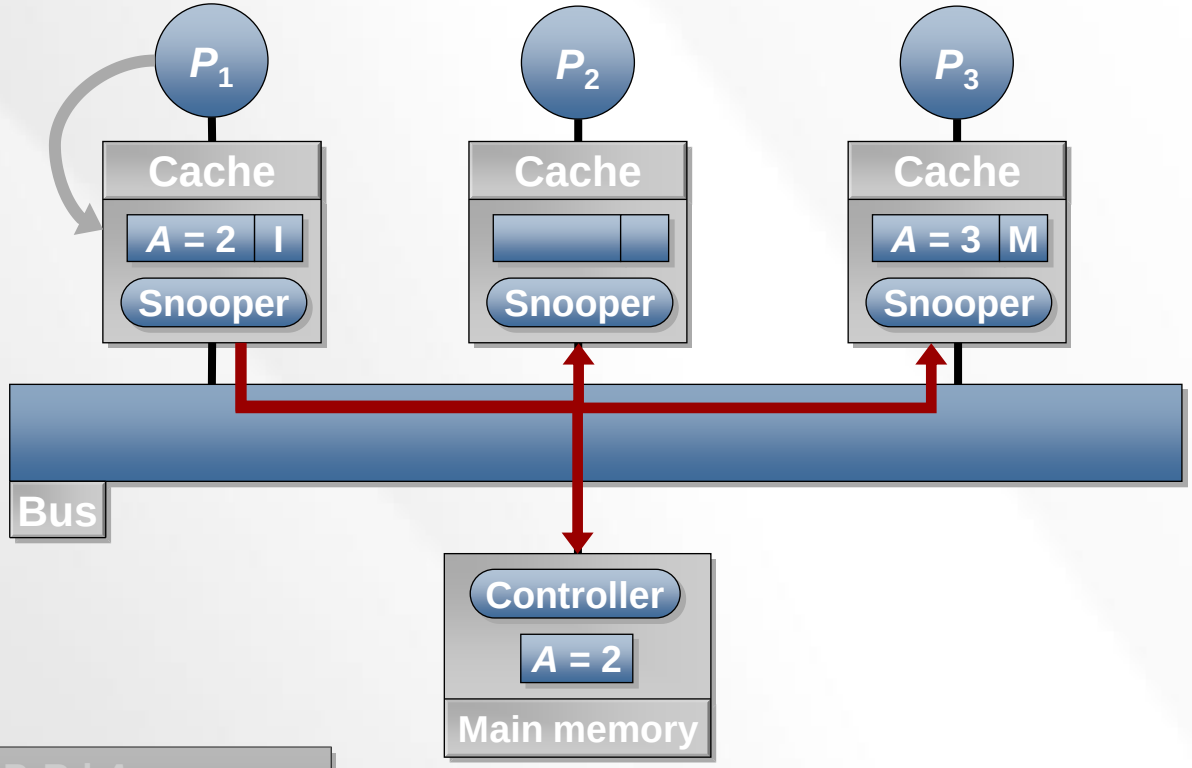
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	**PrRd A**
P_1	BusRd A
P_3	snoops BusRd
P_3	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 issues a BusRd request.



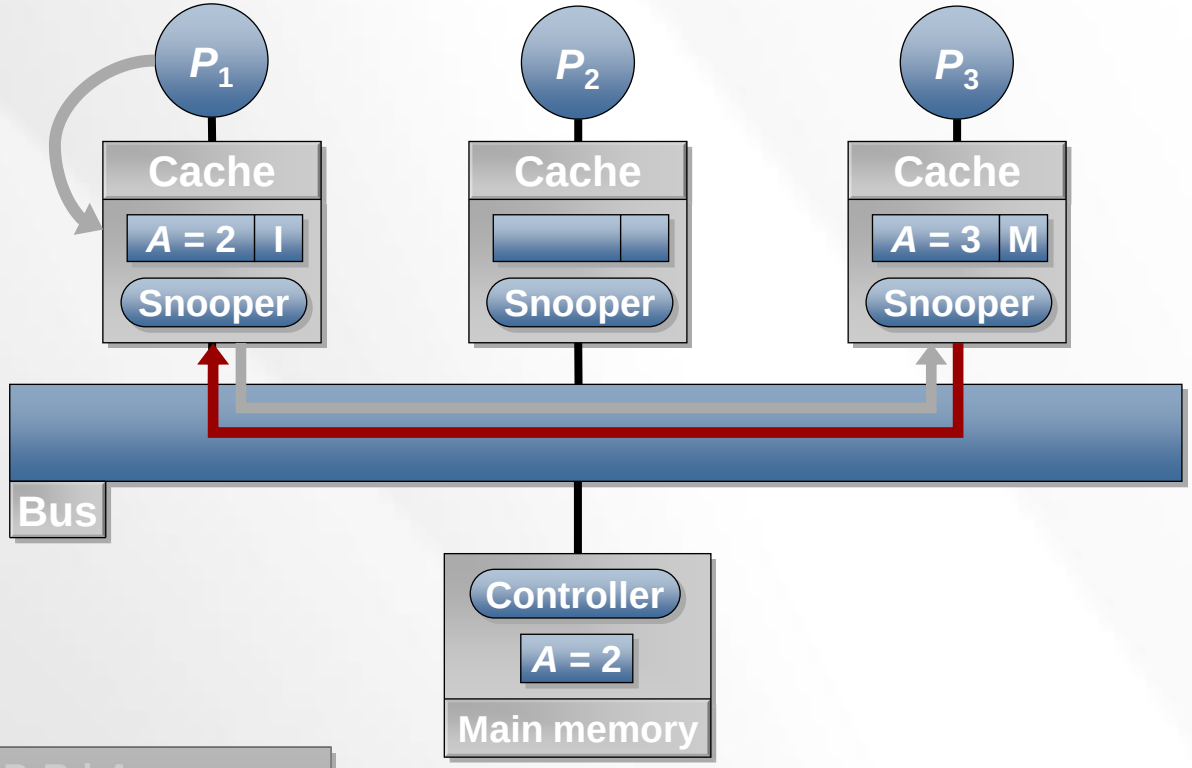
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrRd A
P_1 BusRd A
P_3 snoops BusRd
P_3 Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_3 snoops the BusRd.



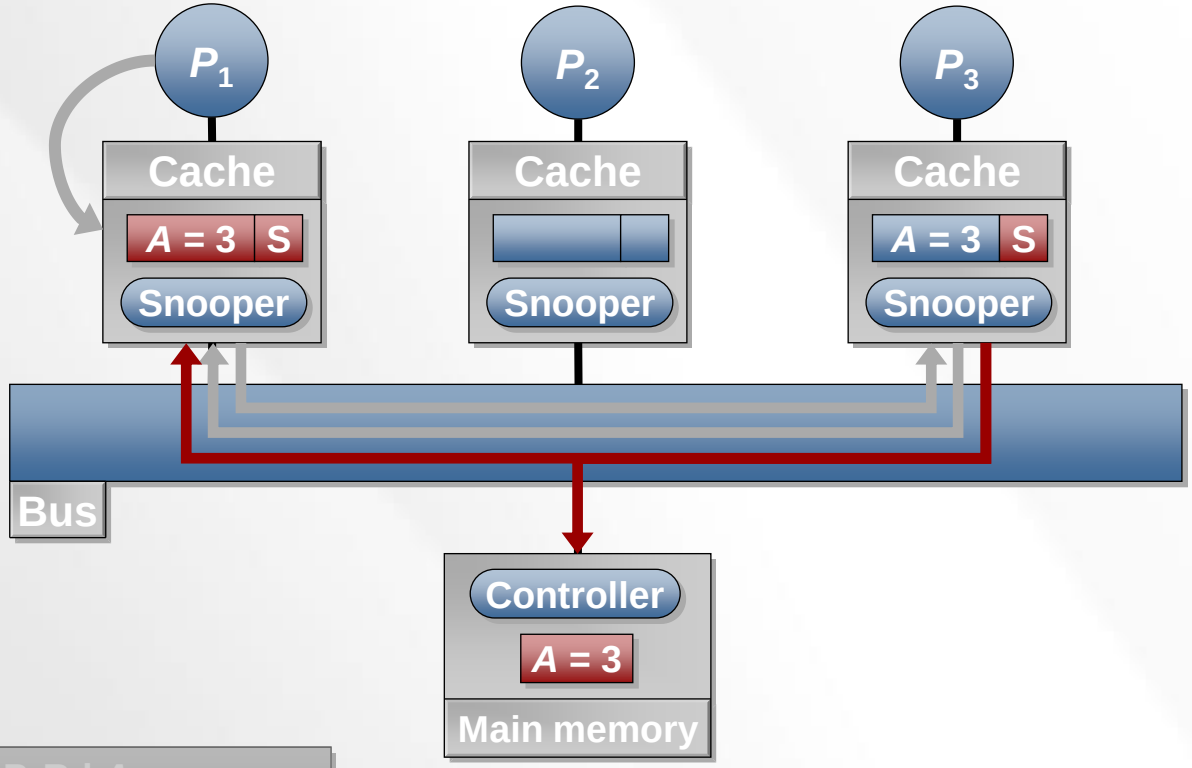
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
P_3	**snoops BusRd**
P_3	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_3 flushes, updating processor P_1, main memory and its own cache state.



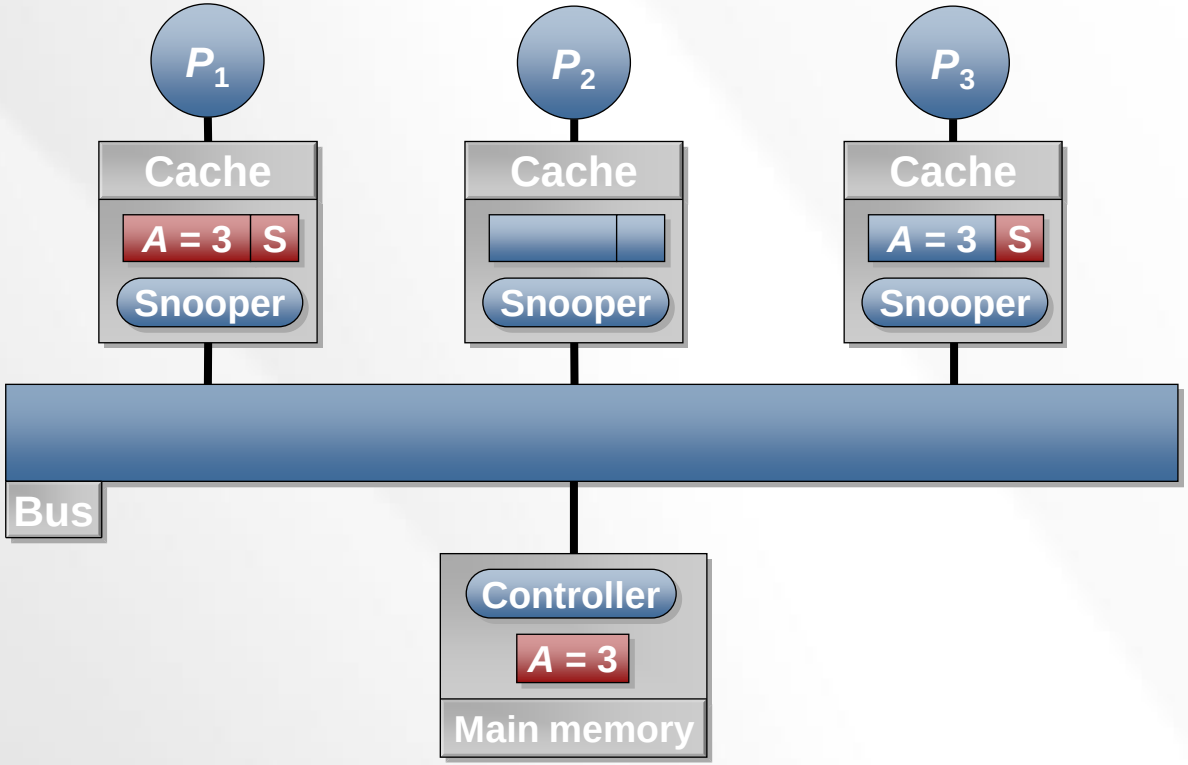
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
P_3	snoops BusRd
P_3	**Flush**

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Read operation completes.

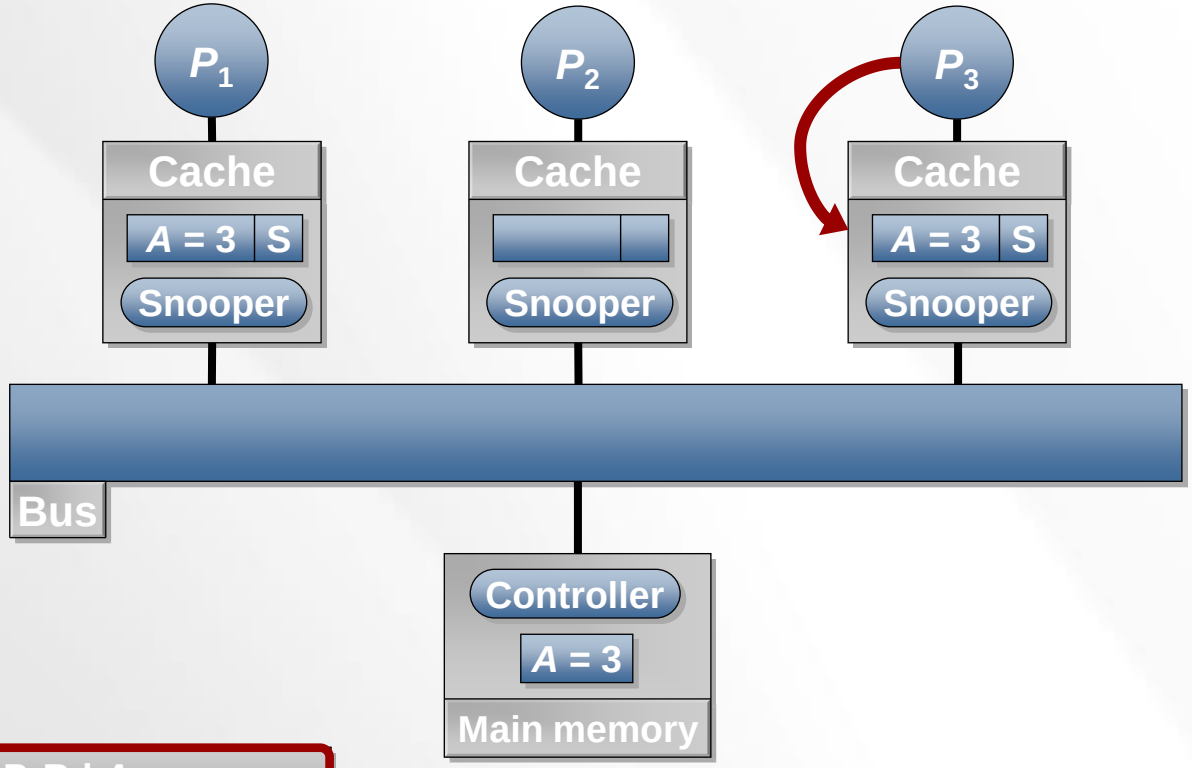


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 reads from its cache.



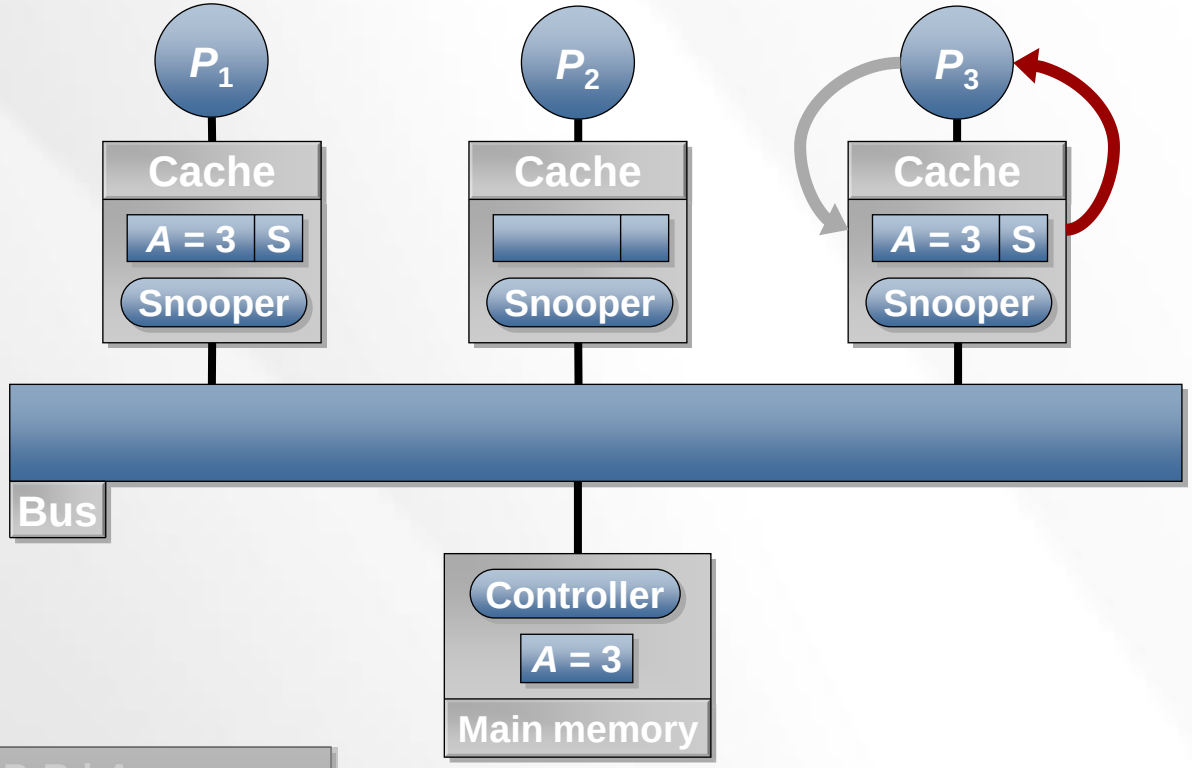
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 returns valid data from its cache.



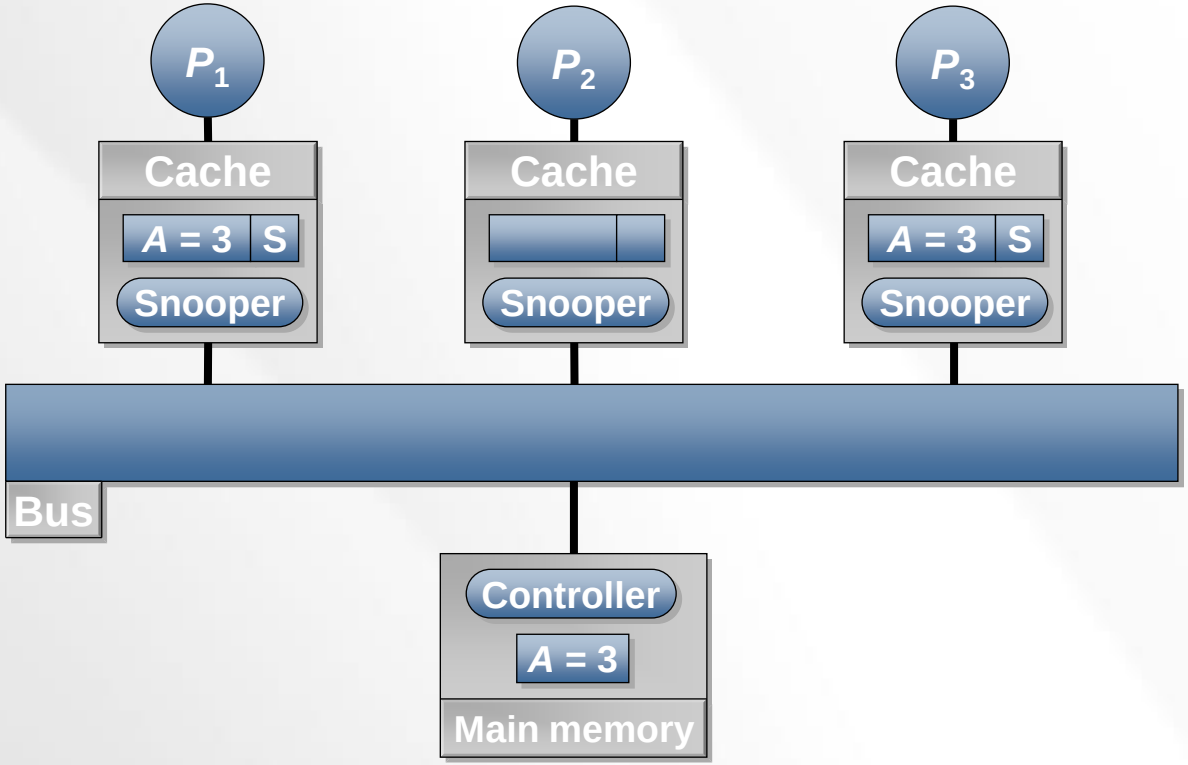
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3 PrRd A
P_3 returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Read operation completes.

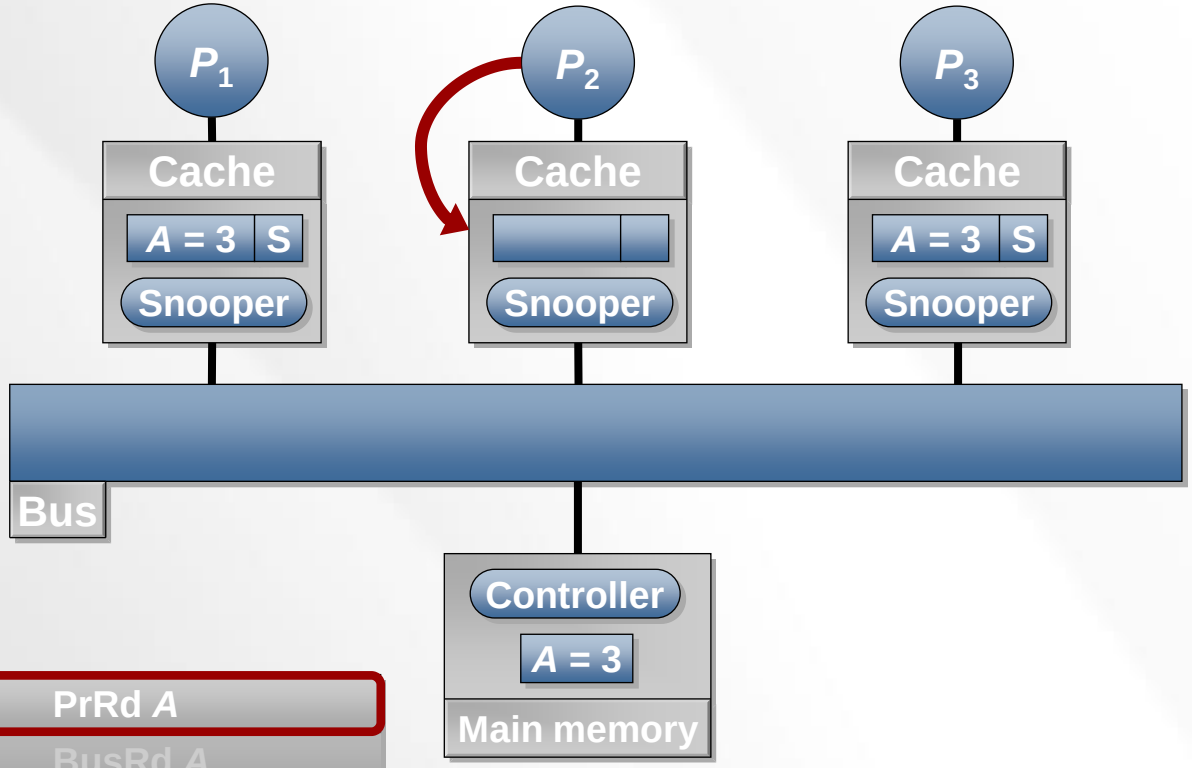


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 reads from its cache.



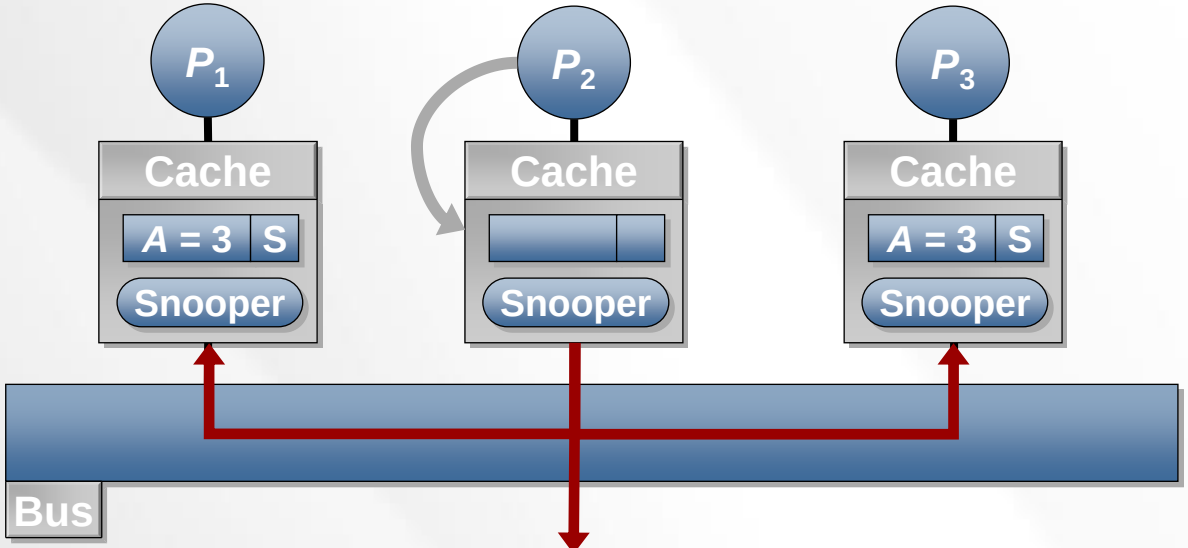
Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_2	PrRd A
P_2	BusRd A
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 issues a BusRd request.



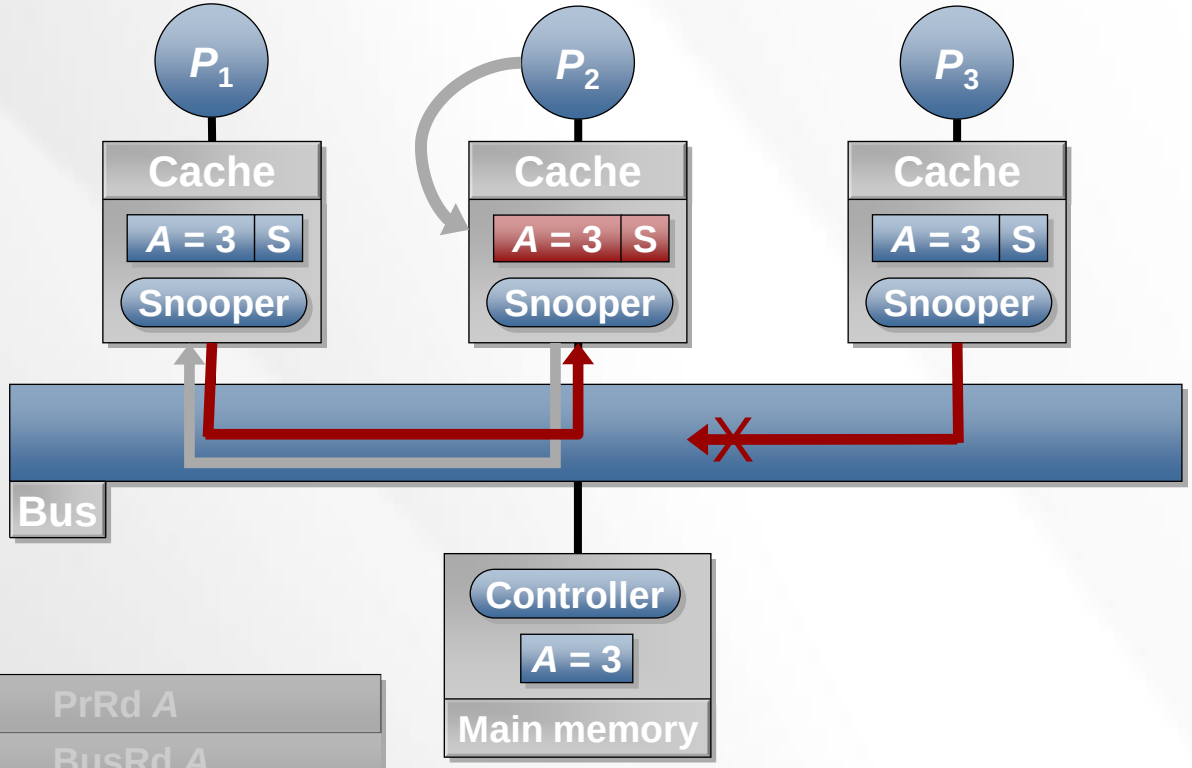
Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_2	PrRd A
P_2	BusRd A
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Main memory controller observes the BusRd.



Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

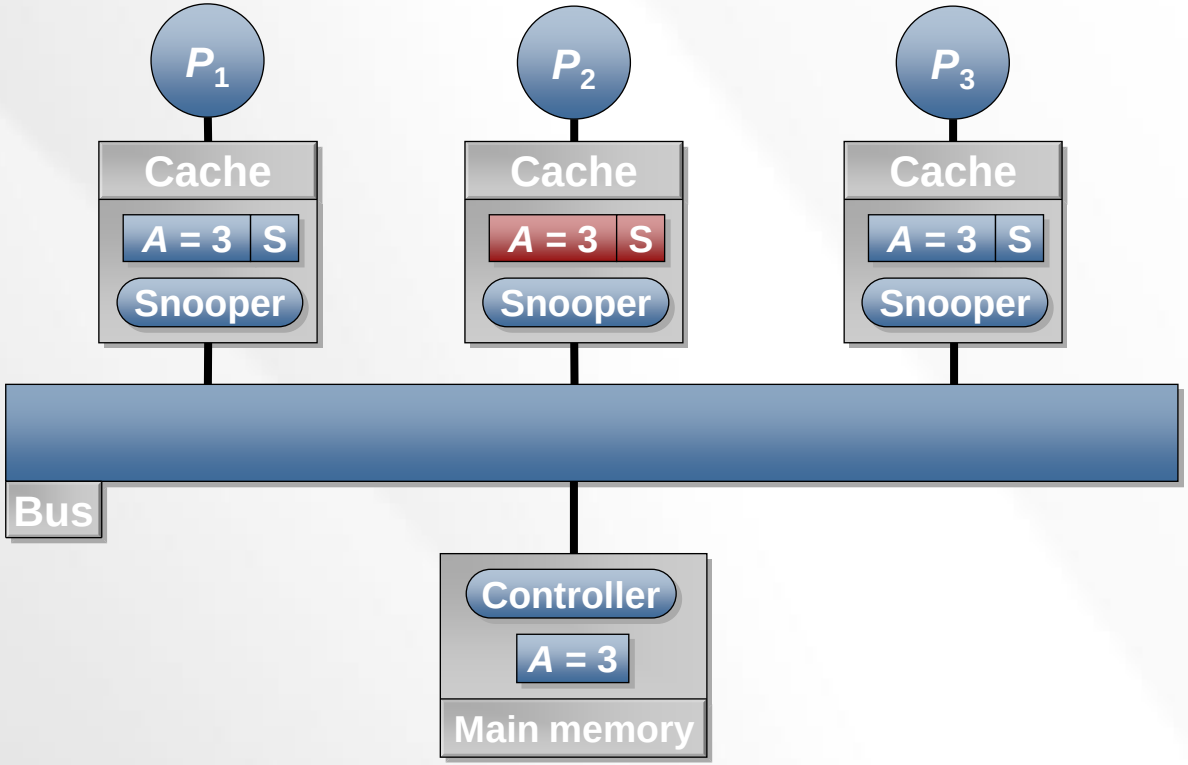
P_2	PrRd A
P_2	BusRd A
P_1	Flush

Referred to as cache-to-cache transfer in Illinois MESI protocol

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Operation completes.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

MESI Example (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

MESI Example (Cache-to-Cache Transfer+BusUpgr)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	-	-	BusRd	Mem
W1	M	-	-	-	Own cache
R3	S	-	S	BusRd/Flush	P1 cache
W3	I	-	M	BusUpgr	Own cache
R1	S	-	S	BusRd/Flush	P3 cache
R3	S	-	S	-	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

MESI Example (Cache-to-Cache Transfer+BusUpgr)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	-	-	BusRd	Mem
W1	M	-	-	-	Own cache
R3	S	-	S	BusRd/Flush	P1 cache
W3	I	-	M	BusUpgr	Own cache
R1	S	-	S	BusRd/Flush	P3 cache
R3	S	-	S	-	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

Lower-Level Protocol Choices

- Who supplies data on miss when not in M state: memory or cache?
- Original, *Illinois* MESI: cache
  - assumes cache is faster than memory (*cache-to-cache transfer*)
  - Not necessarily true
- Adds complexity
  - How does memory know it should supply data? (must wait for caches)
  - A selection algorithm is needed if multiple caches have valid data.
- Useful in a distributed-memory system
  - May be cheaper to obtain from nearby cache than distant memory
  - Especially when constructed out of SMP nodes (Stanford DASH)

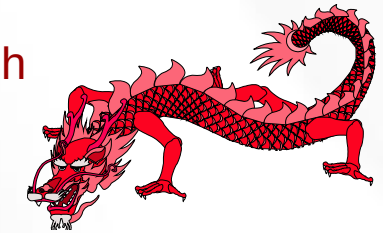
Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

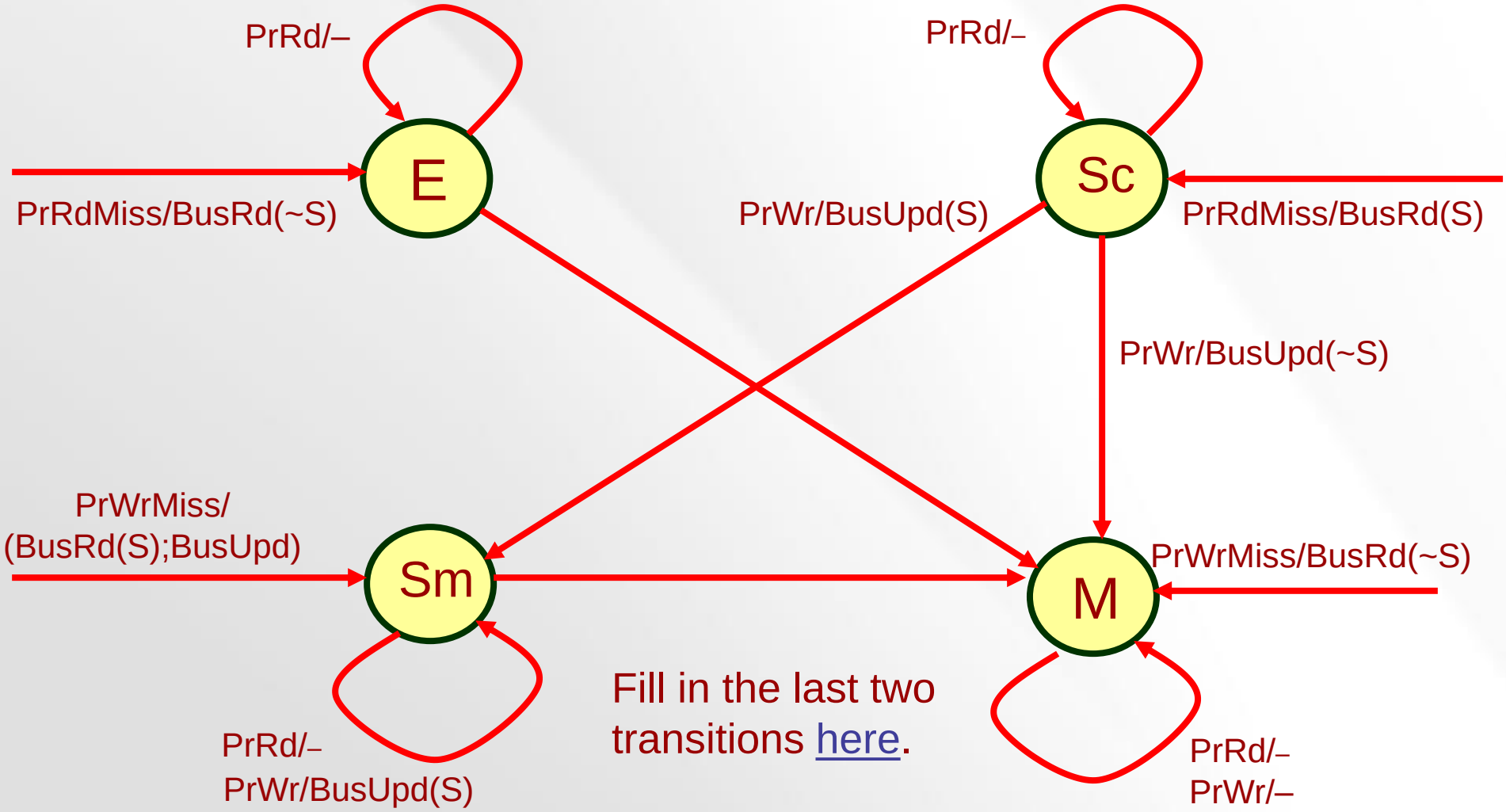
	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

Dragon Writeback Update Protocol

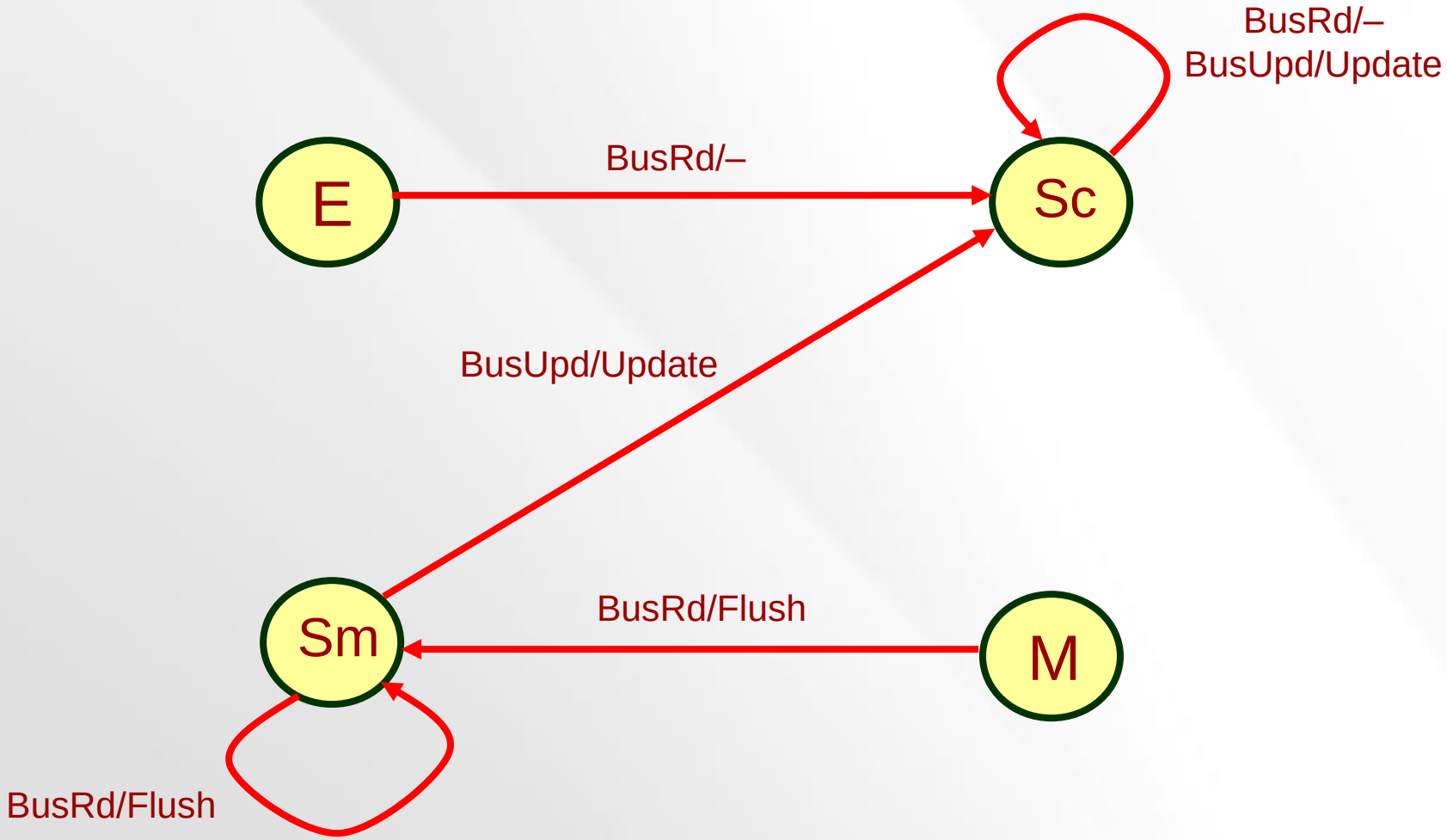
- Four states
  - **Exclusive-clean** (E): Memory and I have it
  - **Shared clean** (Sc): I, others, and maybe memory, but I'm not owner
  - **Shared modified** (Sm): I and others but not memory, and I'm the **owner**
    - Sm and Sc can coexist in different caches, with at most one Sm
  - **Modified** or dirty (M): I and, no one else
  - On replacement: Sc can silently drop, Sm has to flush
- No invalid state
  - If in cache, cannot be invalid
  - If not present in cache, can view as being in not-present or invalid state
- New processor events: PrRdMiss, PrWrMiss
  - Introduced to specify actions when block not present in cache
- New bus transaction: BusUpd
  - Broadcasts single word written on bus; updates other relevant caches



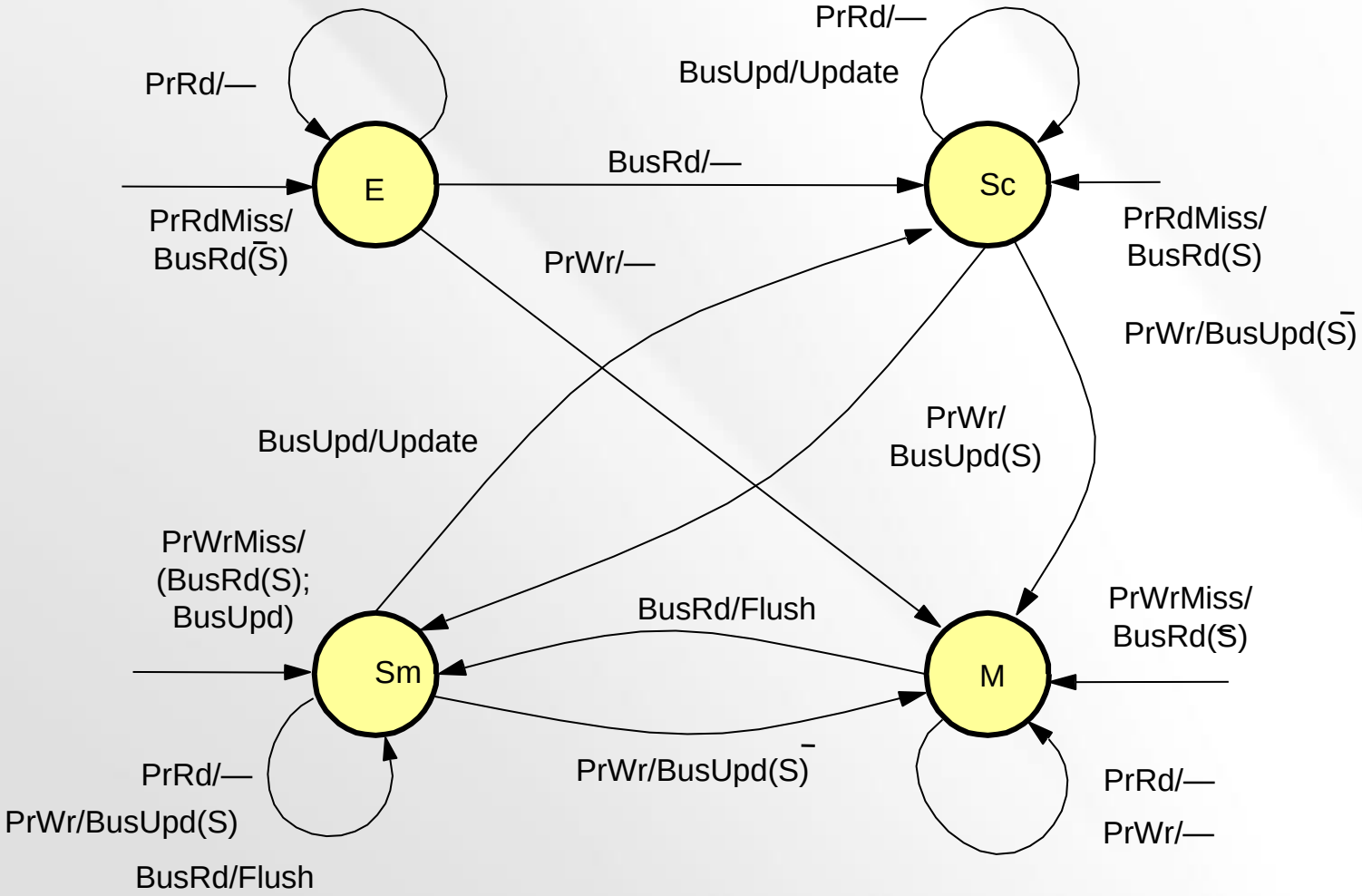
Dragon: Processor-Initiated Transactions



Dragon: Bus-Initiated Transactions

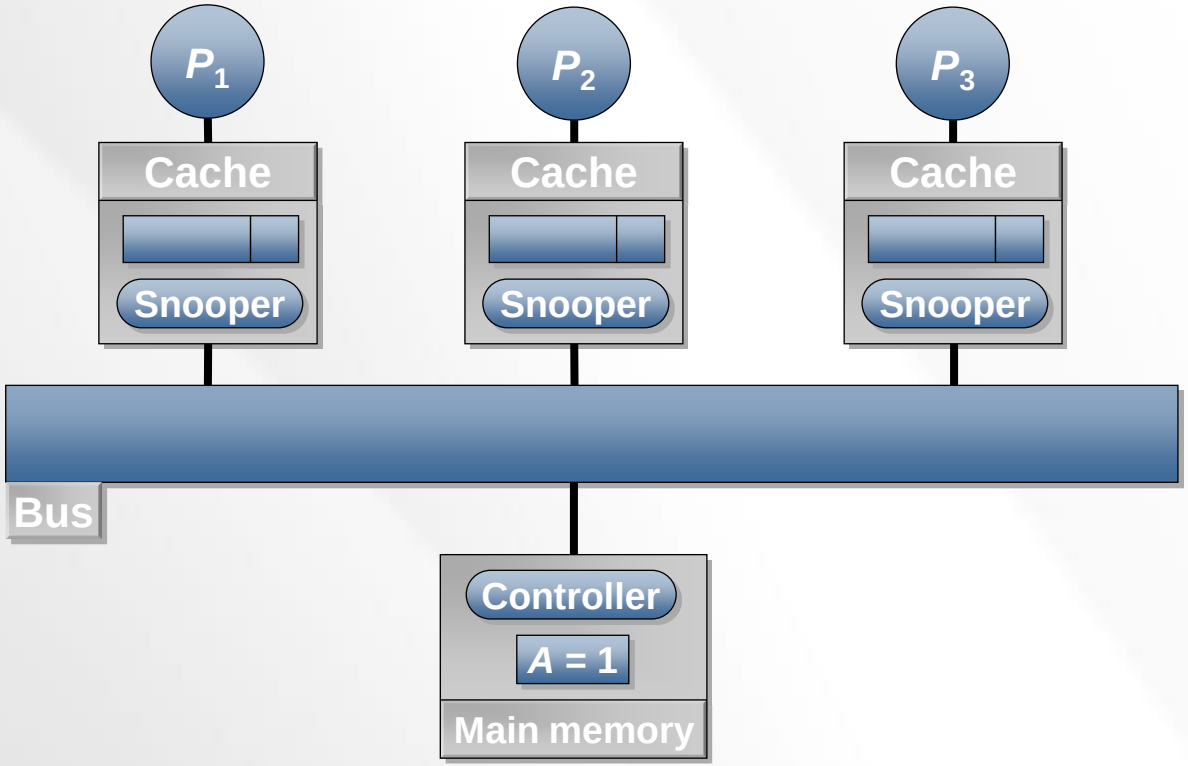


Dragon State Transition Diagram



Dragon Visualization

Start state. All caches empty and main memory has $A = 1$.

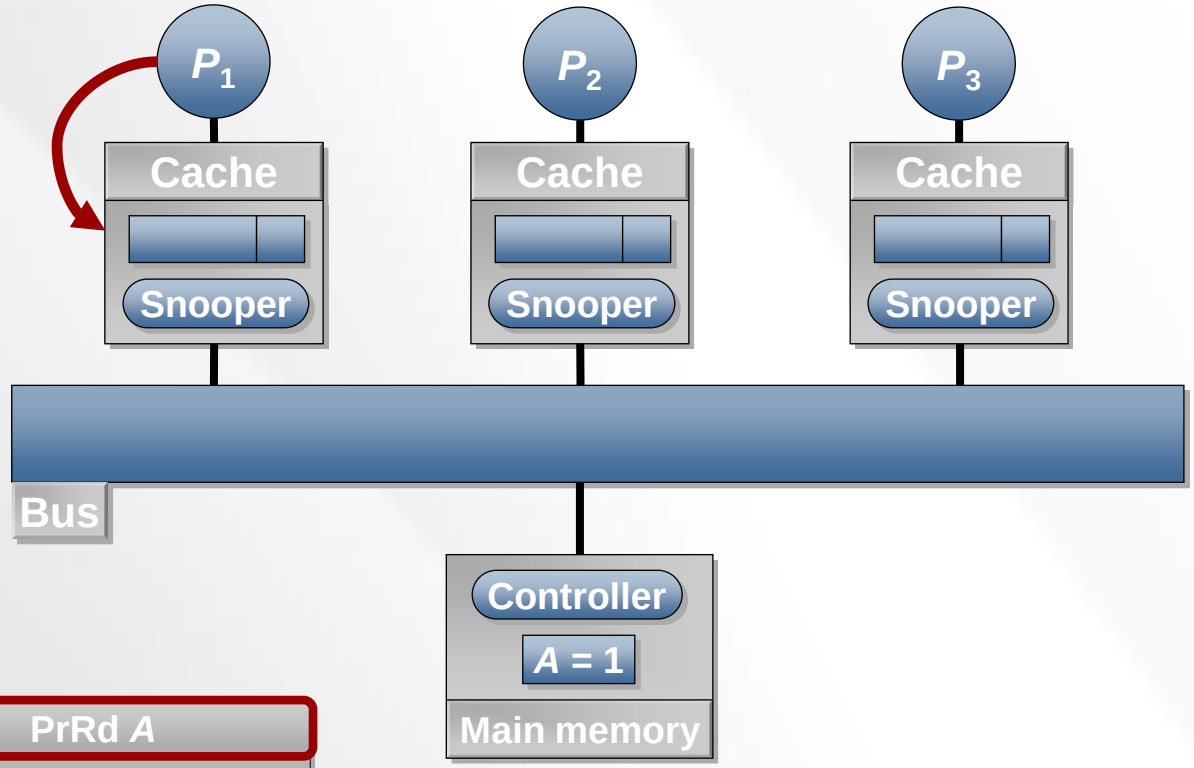


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 attempts to read A from its cache.



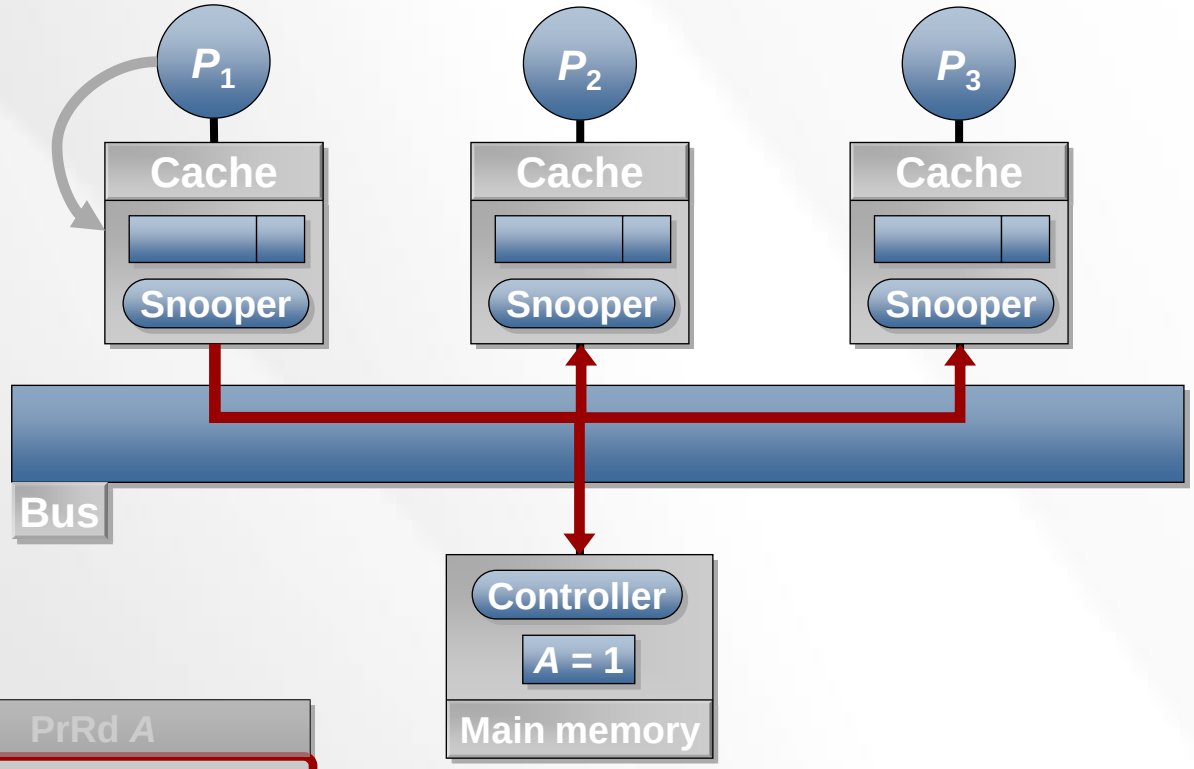
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 issues a BusRd.



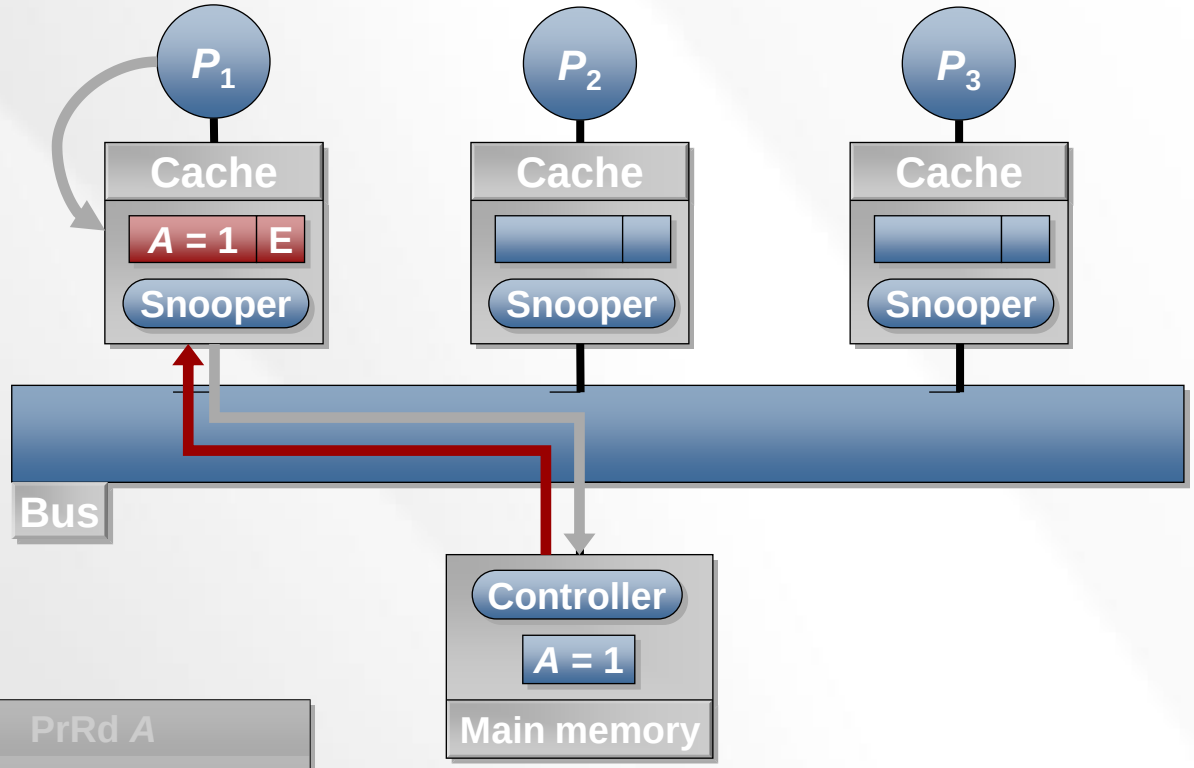
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Main memory returns data to processor P_1 which updates its cache.



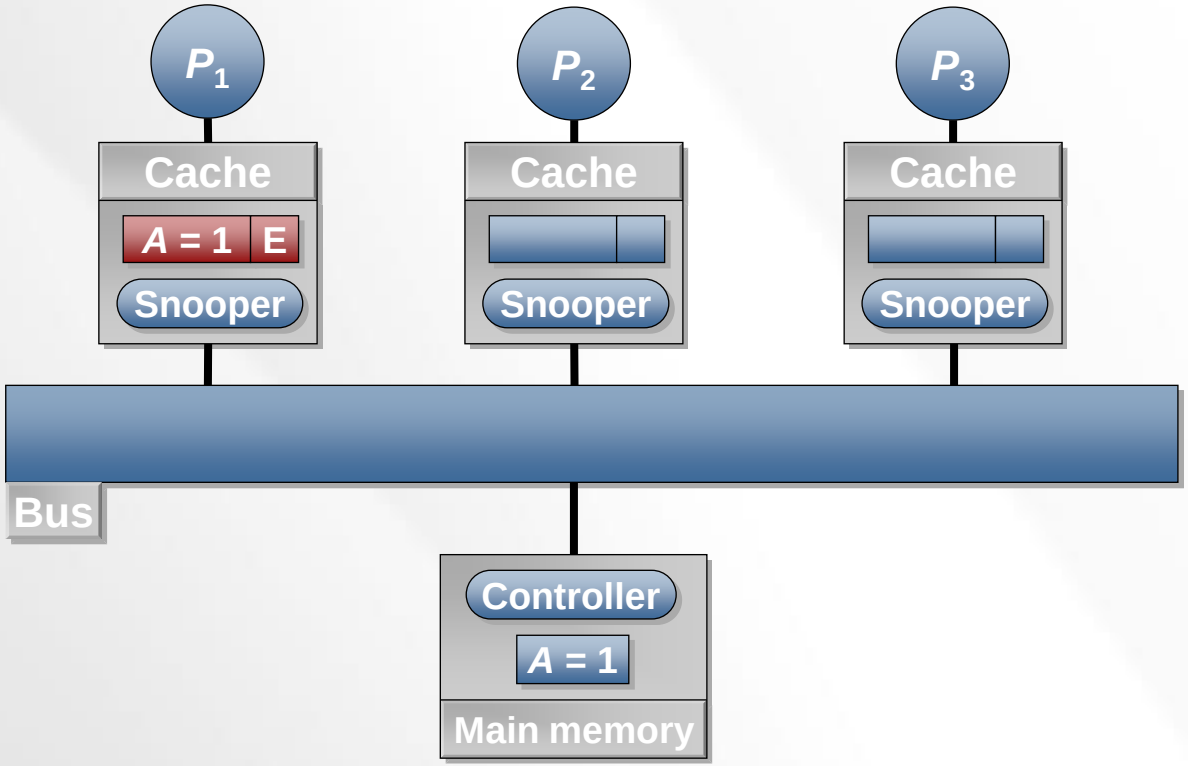
Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_1	PrRd A
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Read operation completes.

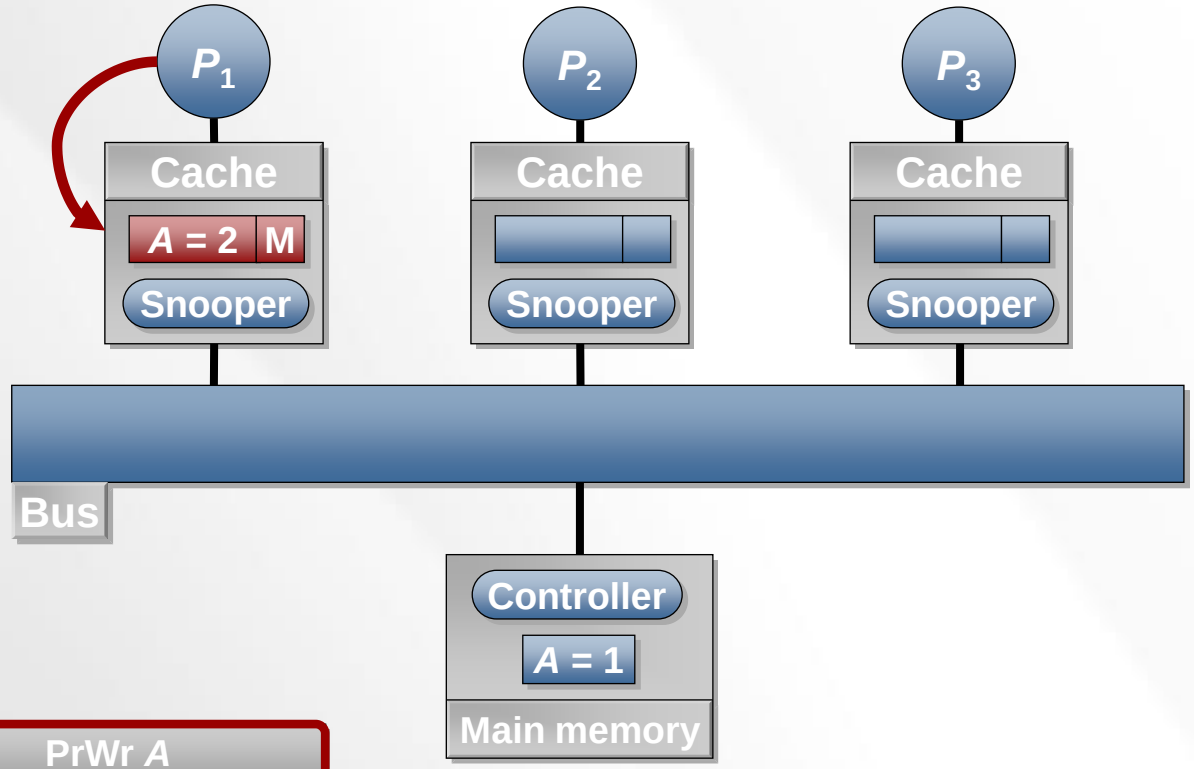


Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Processor P_1 writes to its cache.



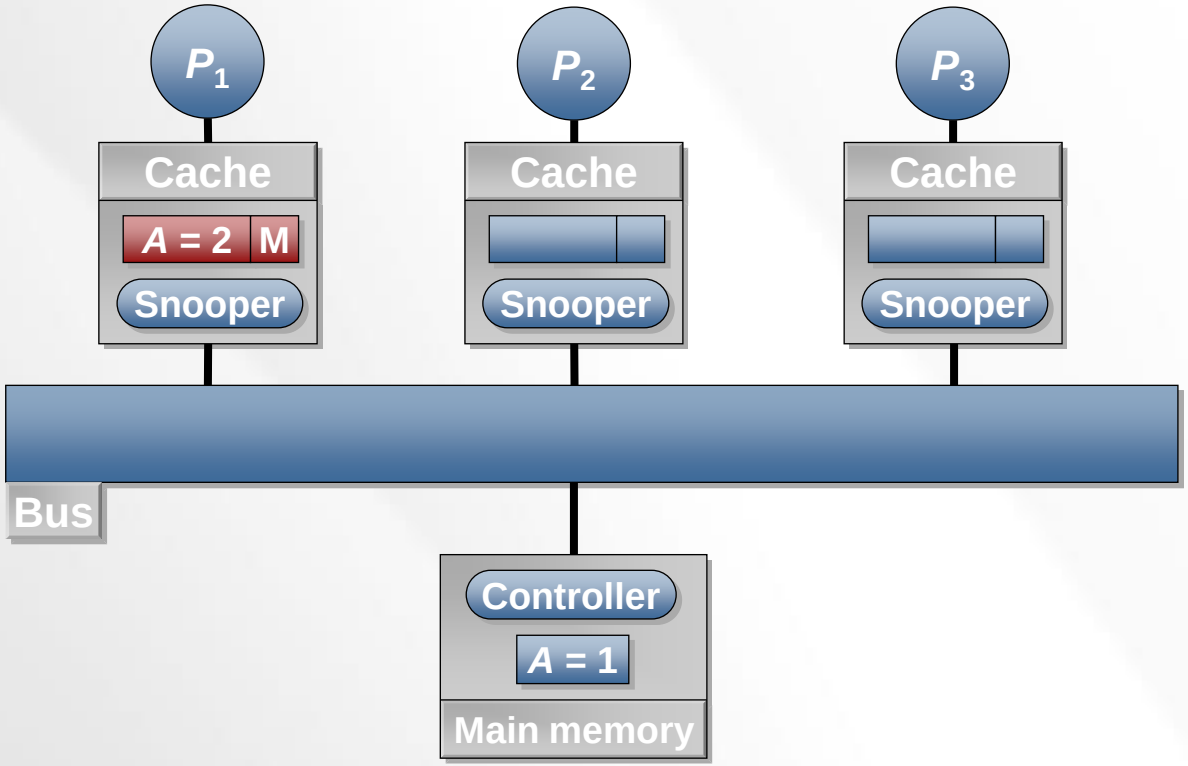
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrWr A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Write operation completes.

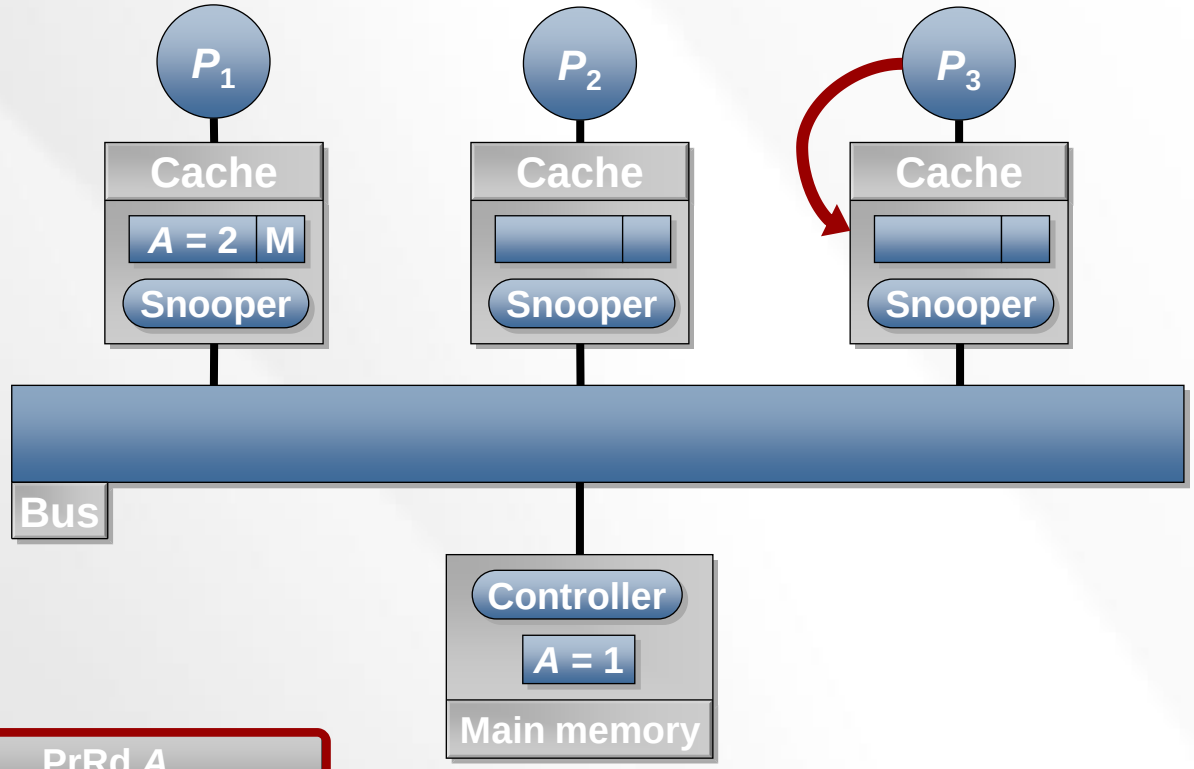


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 attempts to read A from its cache.



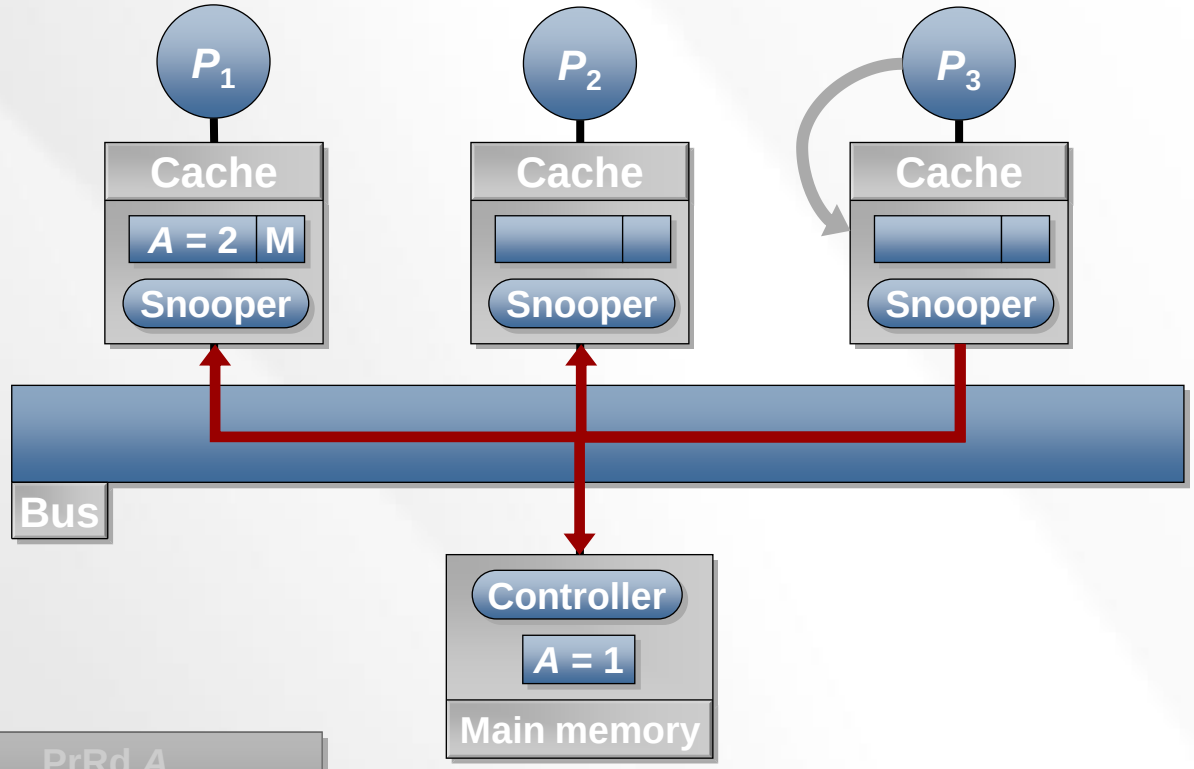
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 issues a BusRd.



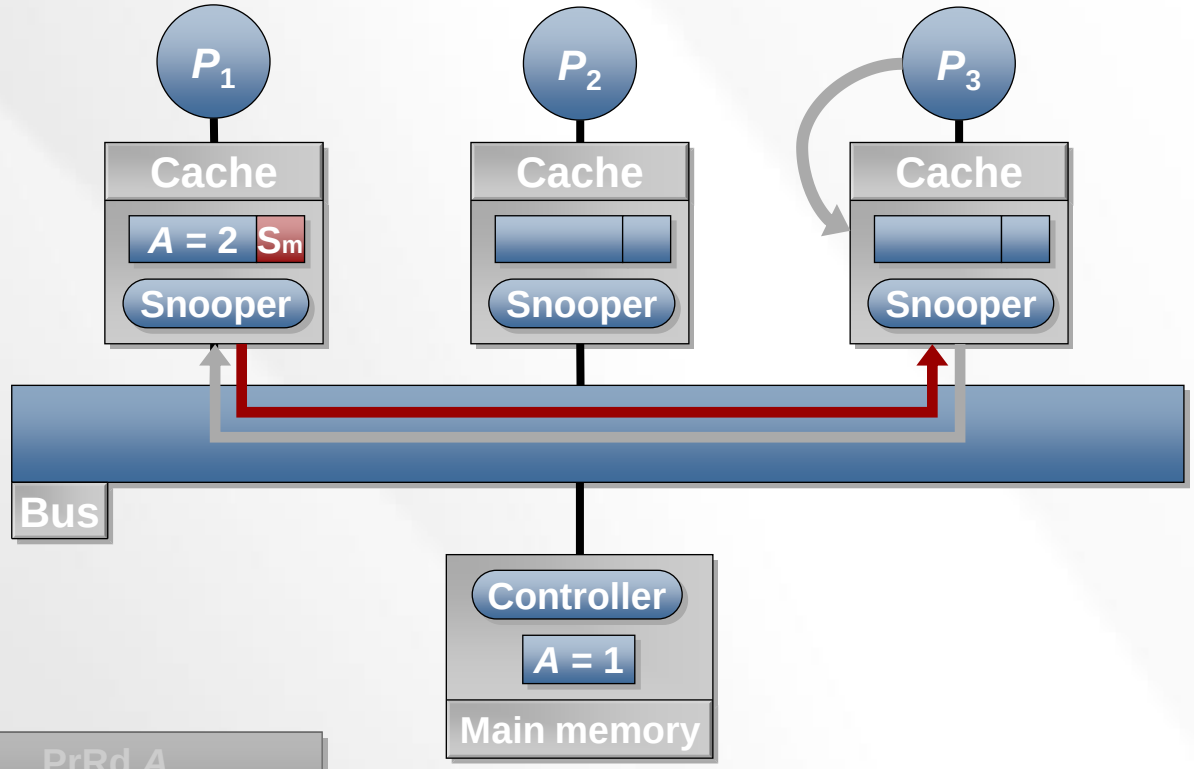
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_1 snoops the BusRd from processor P_3.



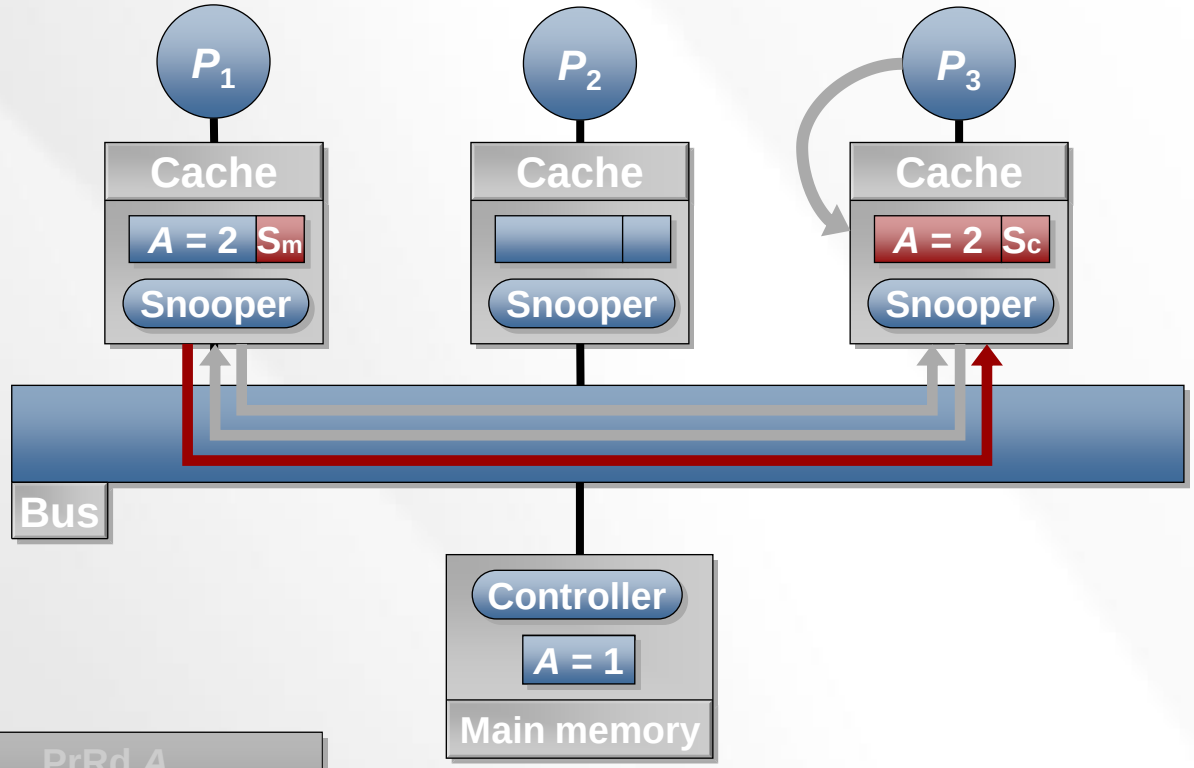
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

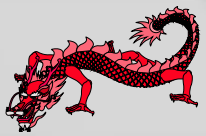
Processor P_1 flushes, sending updated data to P_3 and main memory.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

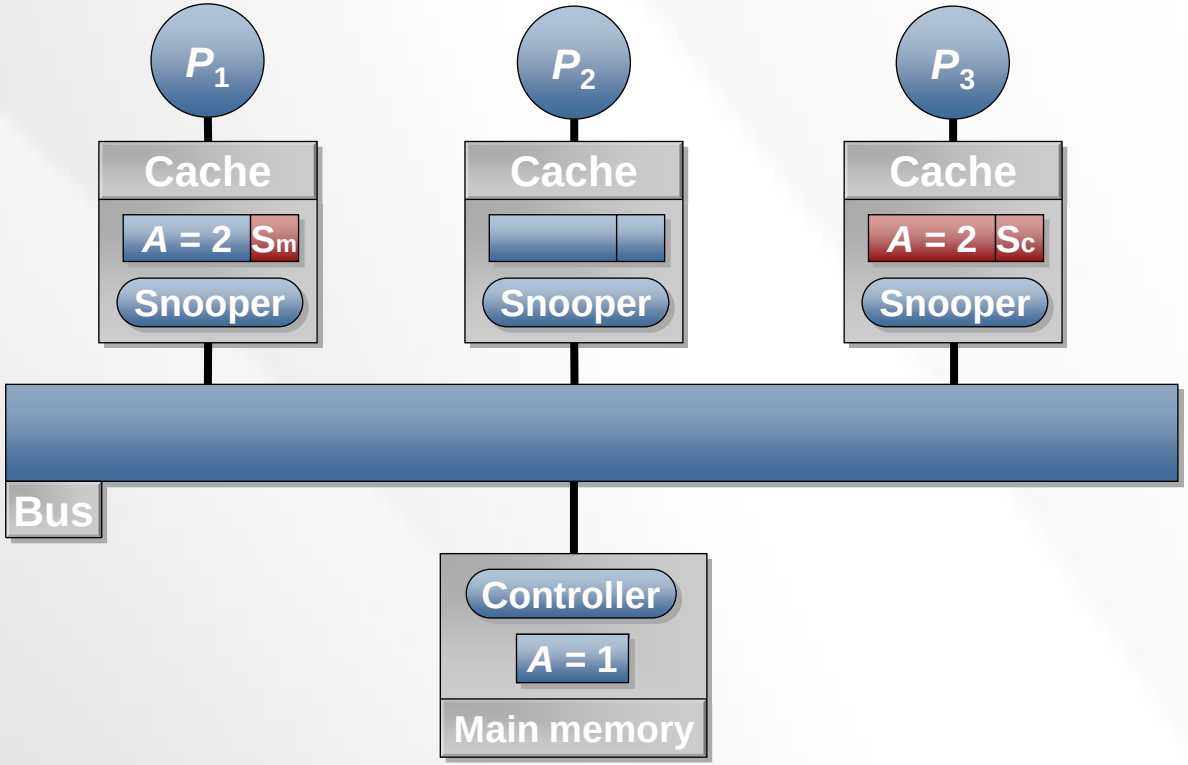
P_3	PrRd A
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon



Processor P_3 Reads A

Read operation completes.

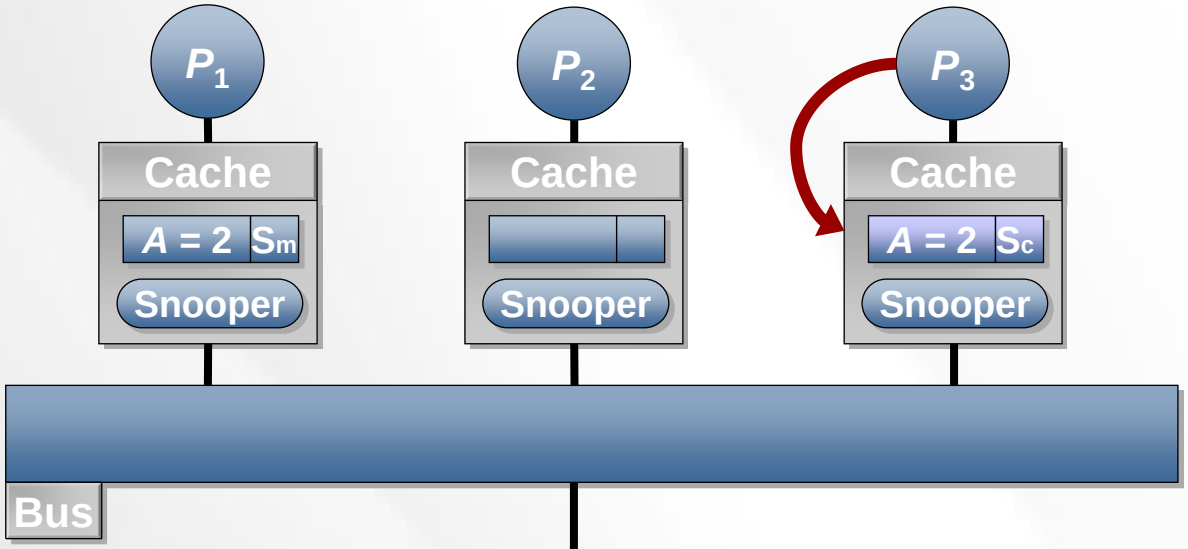


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 writes to its cache.



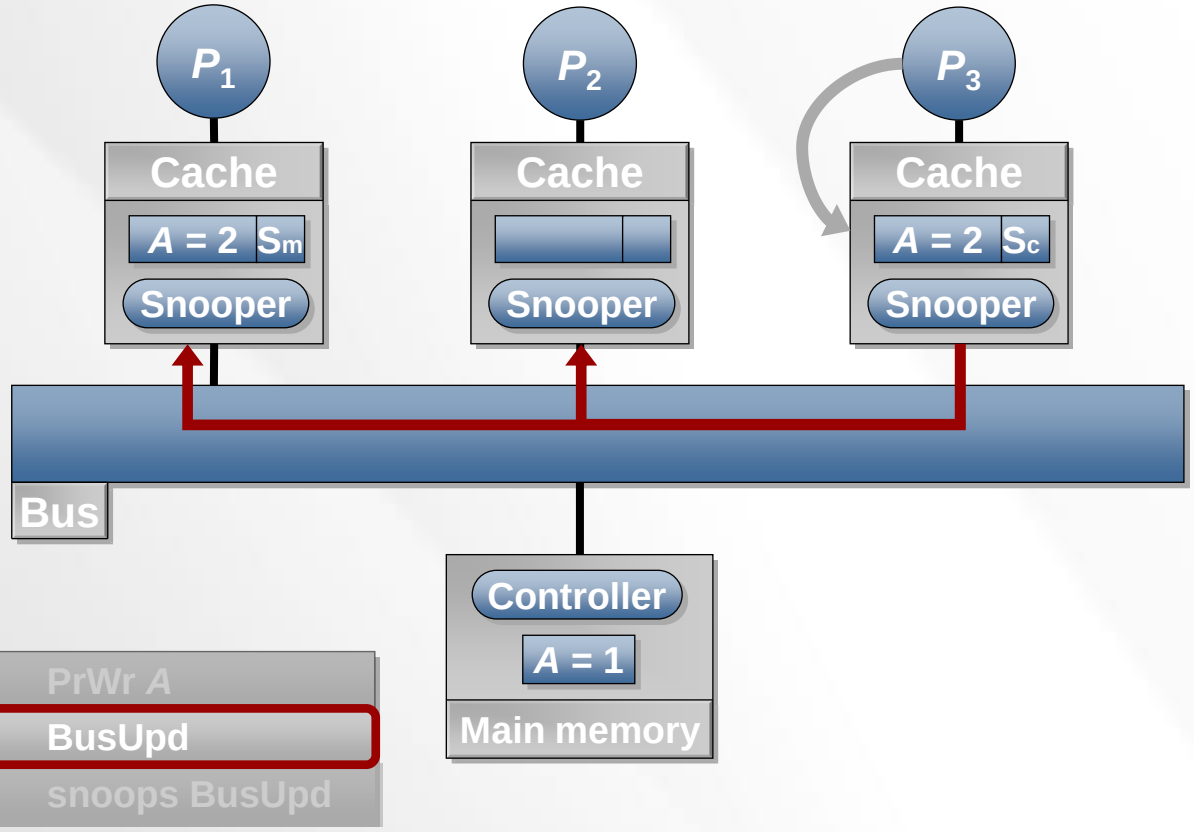
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A	
P_3	BusUpd	
P_1	snoops BusUpd	

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 issues a BusUpd request.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

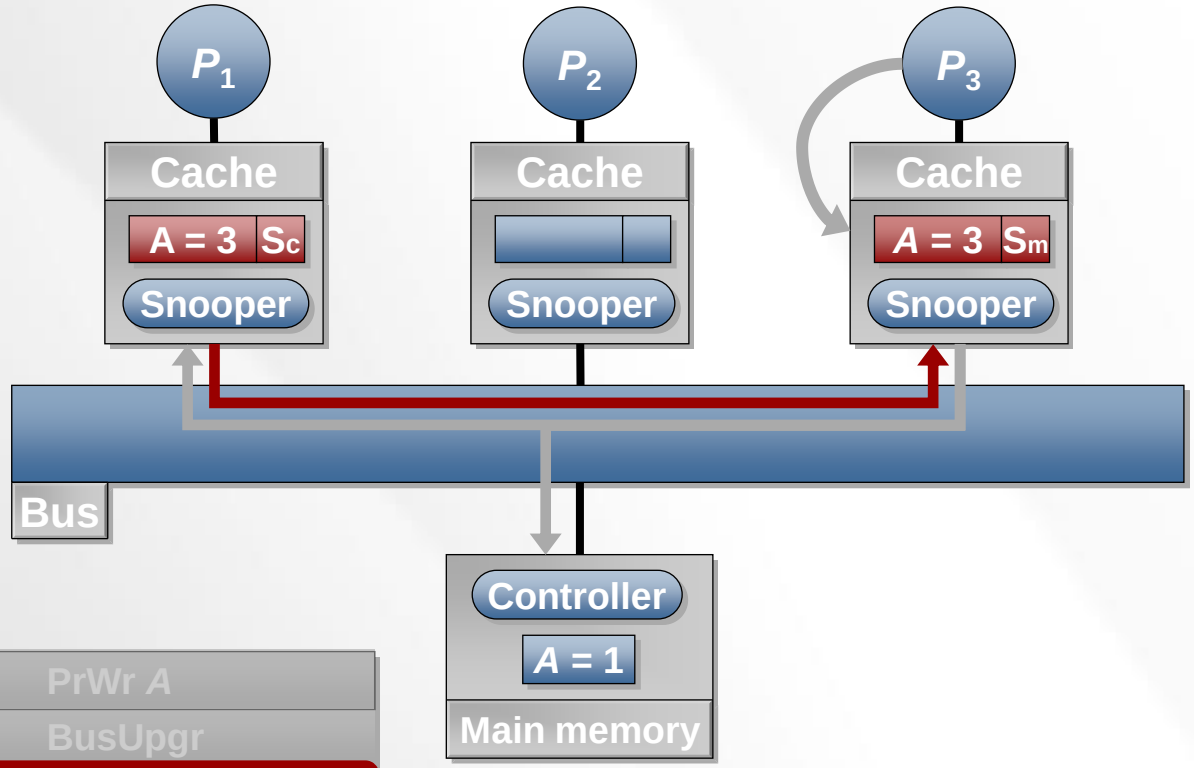
P_3	PrWr A
P_3	BusUpd
P_1	snoops BusUpd

Note: BusUpdate instead of BusUpgr (no invalidation is performed)

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_1 snoops the BusUpd and updates its cache.



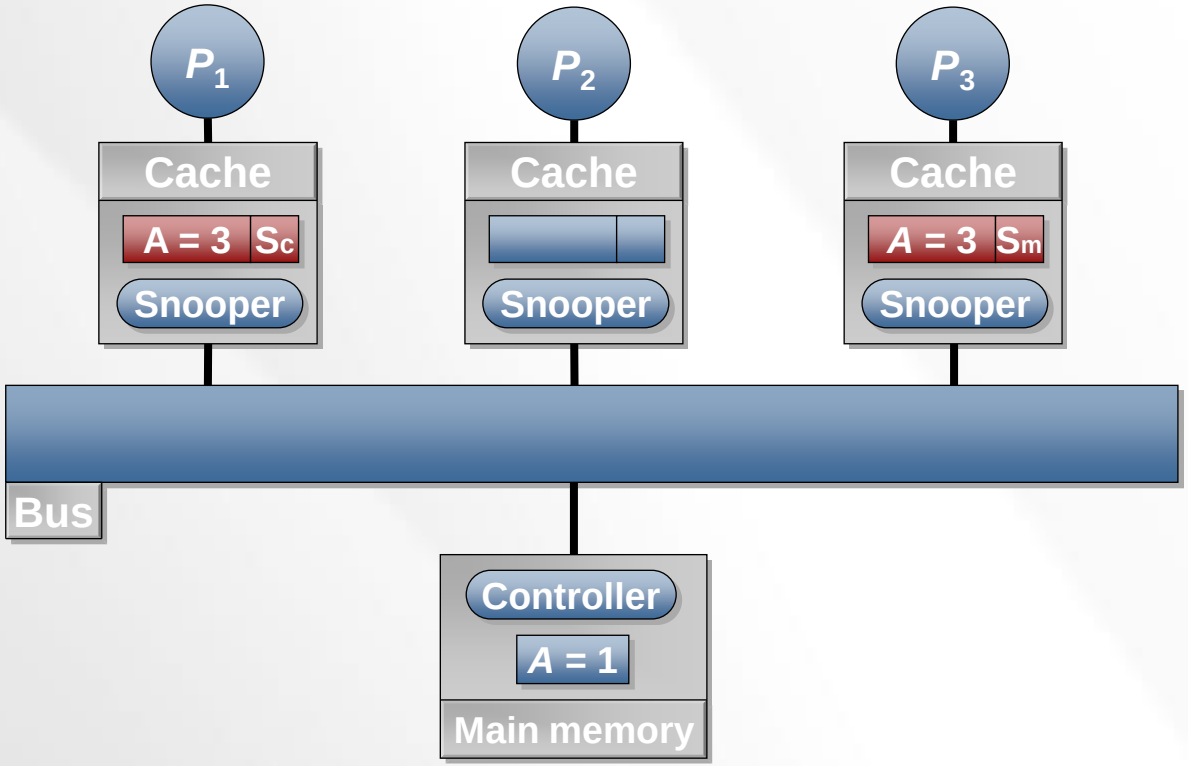
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A
P_3	BusUpgr
P_1	snoops BusUpd

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Write operation completes.

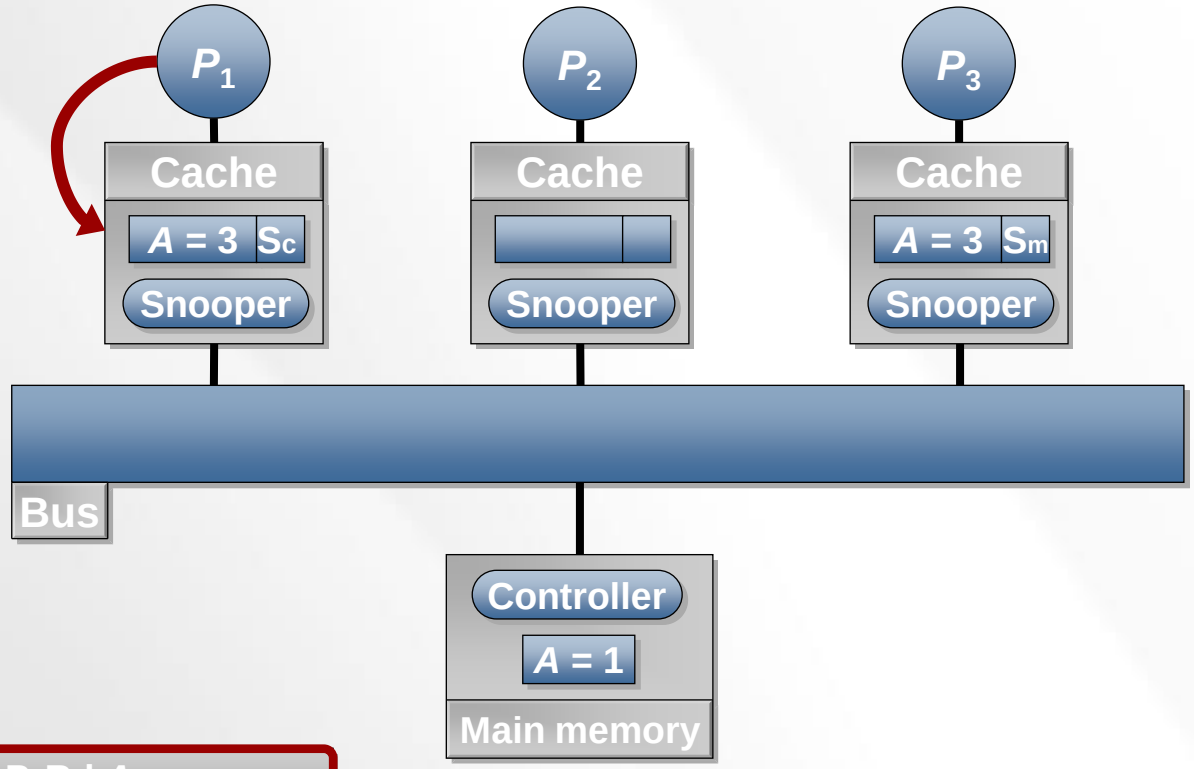


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 reads from its cache.



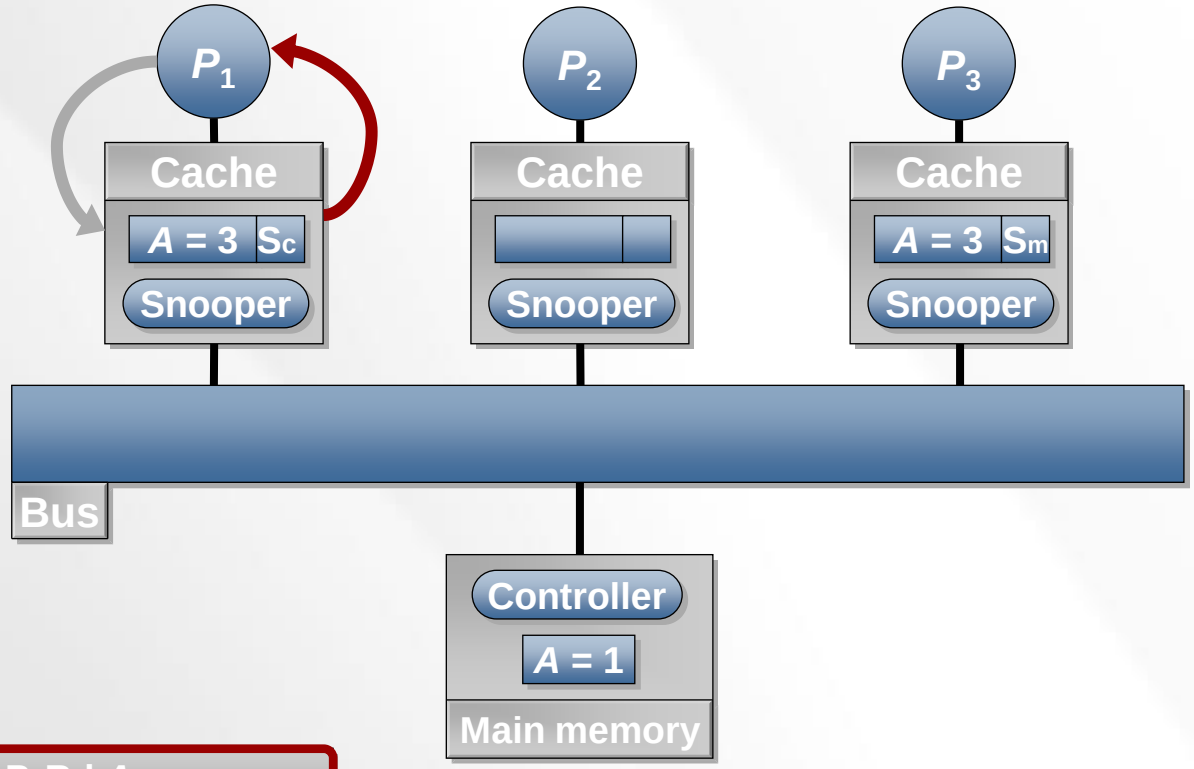
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrRd A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 reads from its cache.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

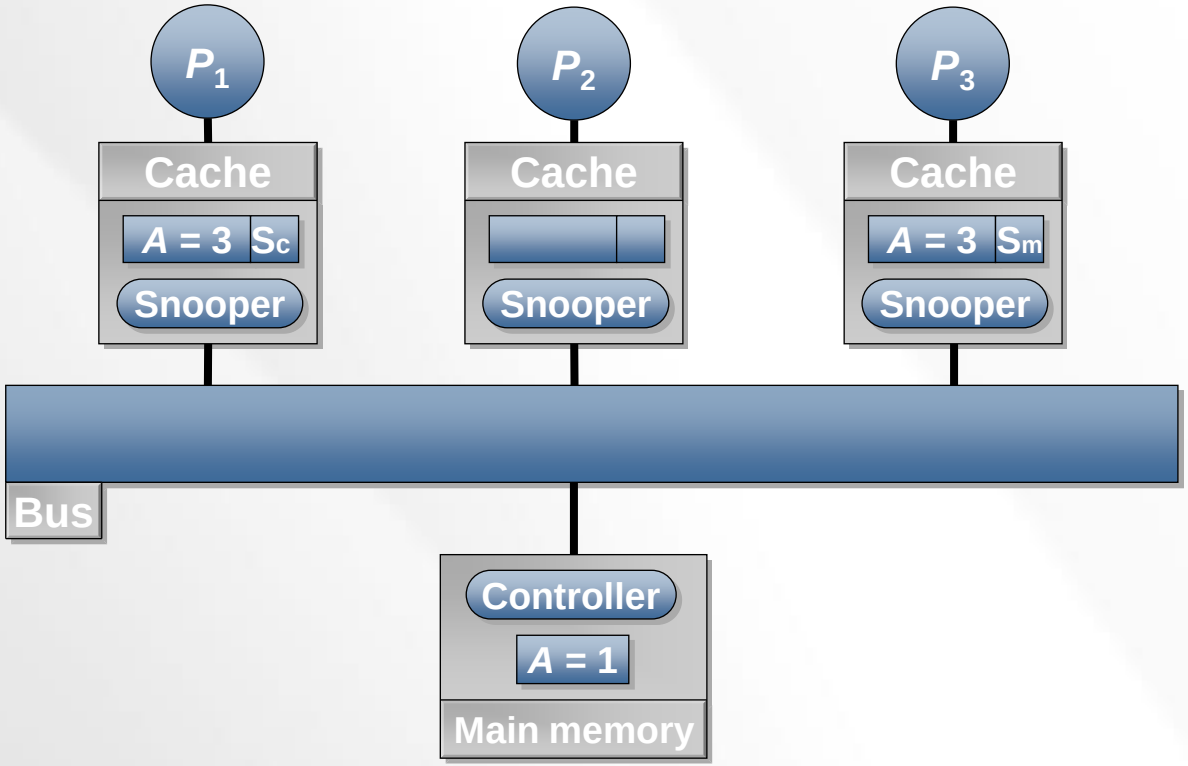
P_1 PrRd A

This is a miss in the MESI and MSI protocols.

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Read operation completes.

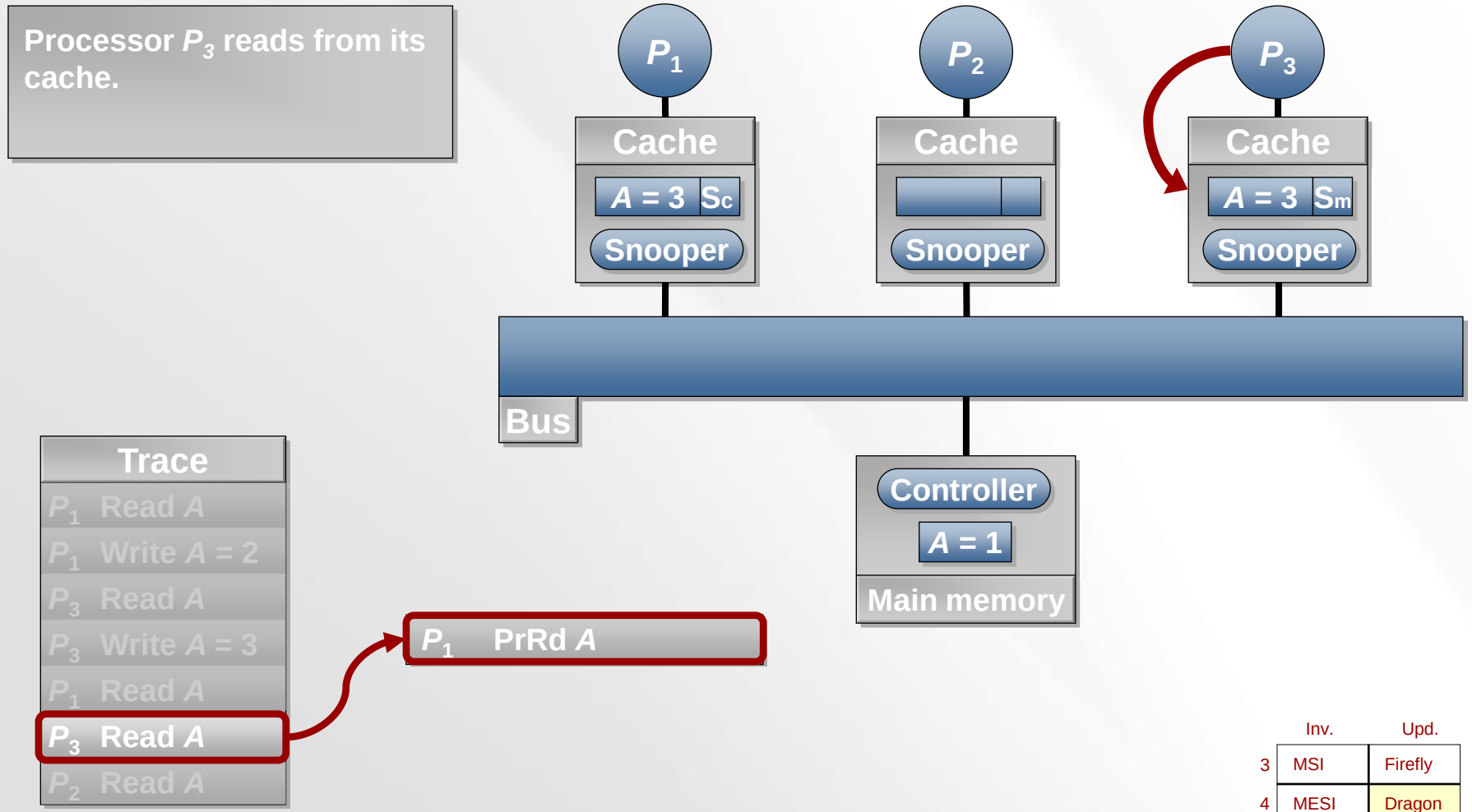


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

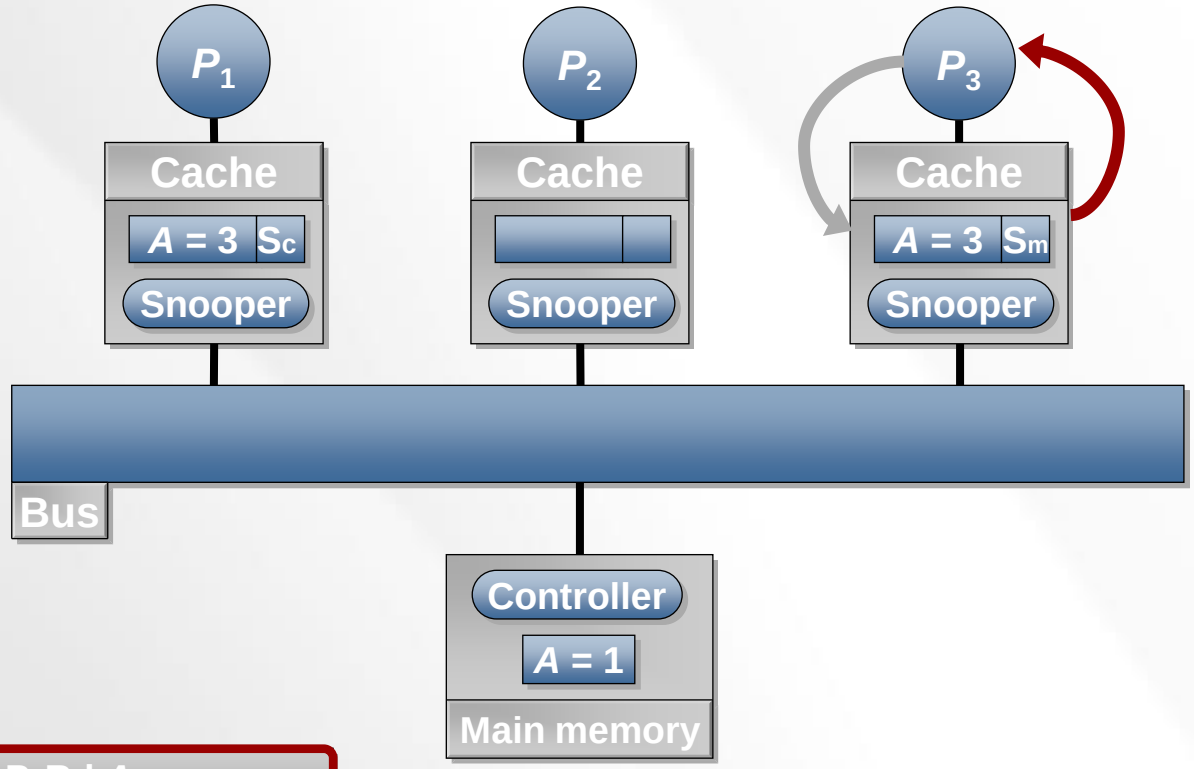
Processor P_3 reads from its cache.



	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 reads from its cache.



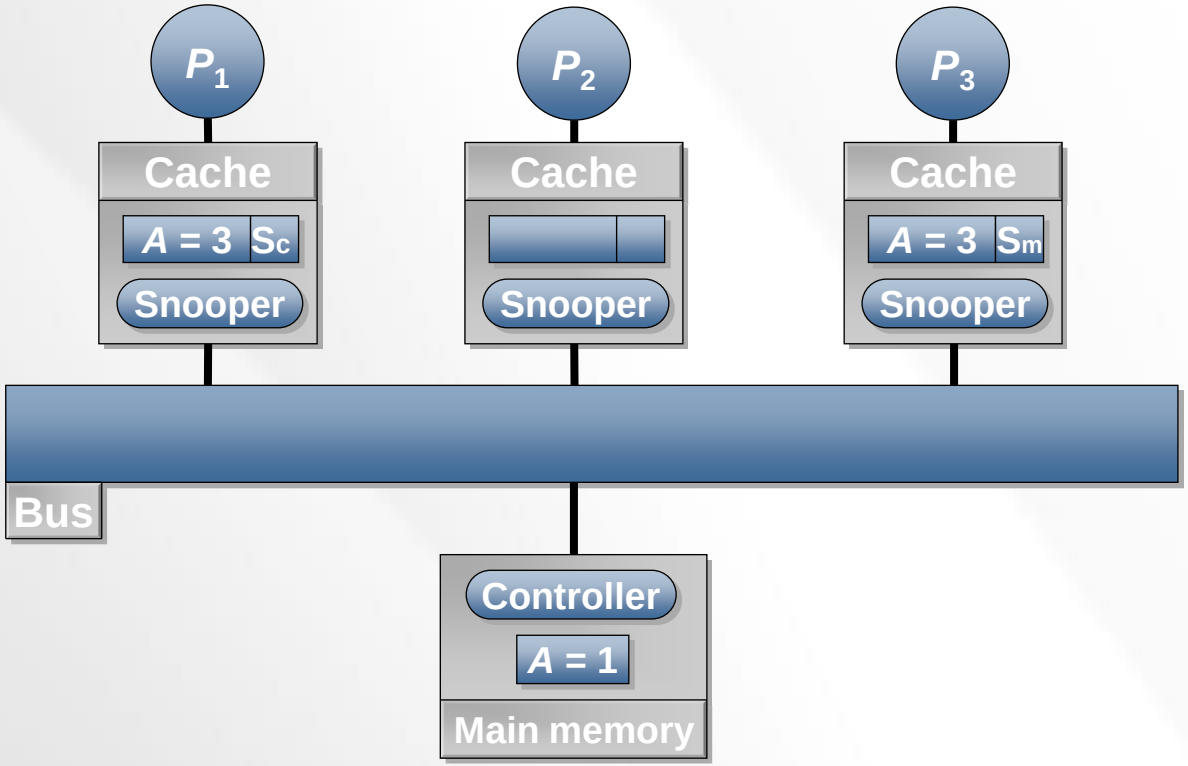
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrRd A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Read operation completes.

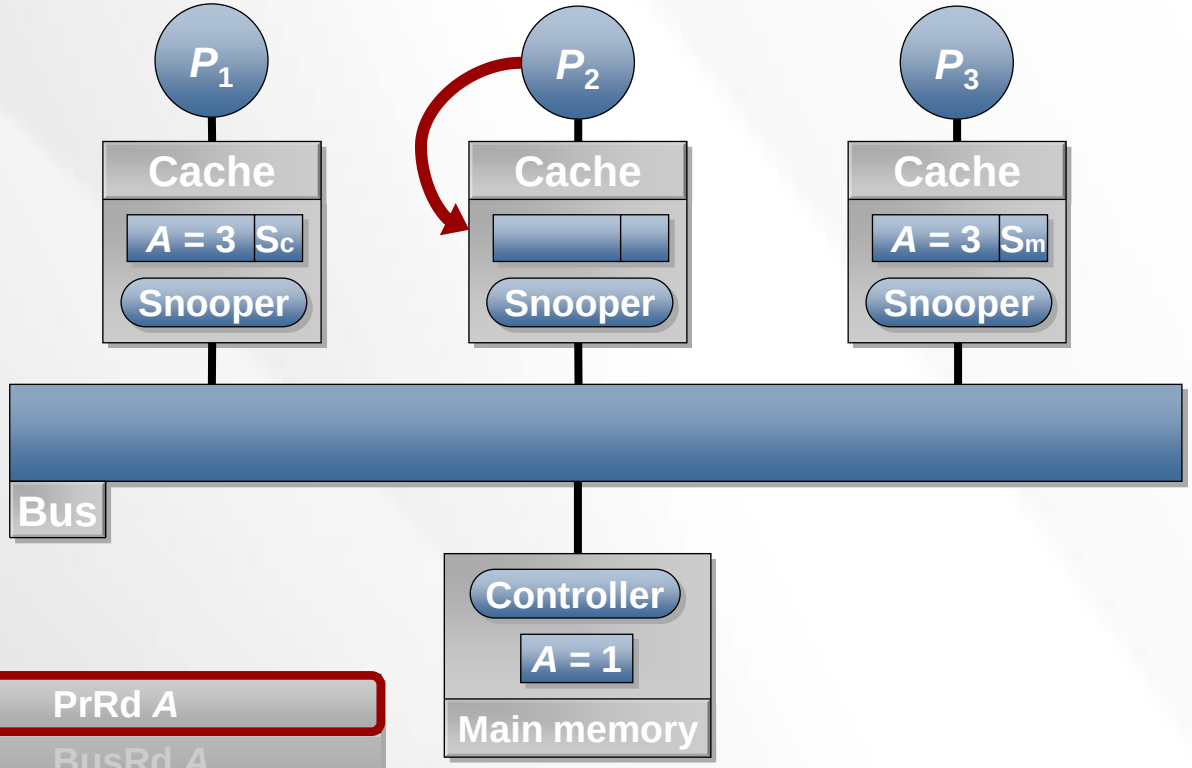


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 reads from its cache.



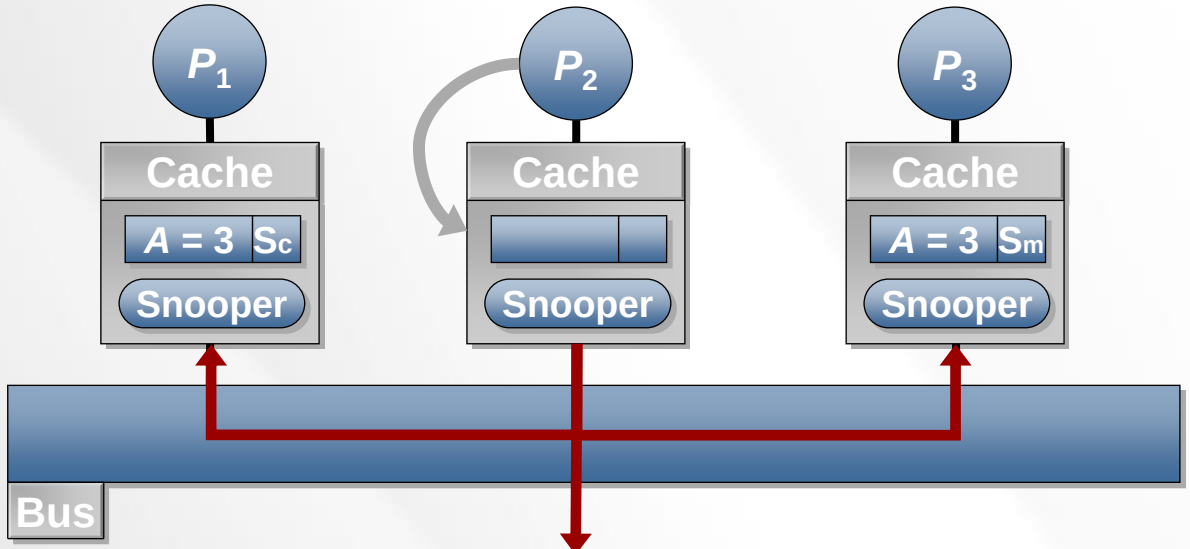
Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_2	PrRd A
P_2	BusRd A
P_3	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 issues a BusRd request.



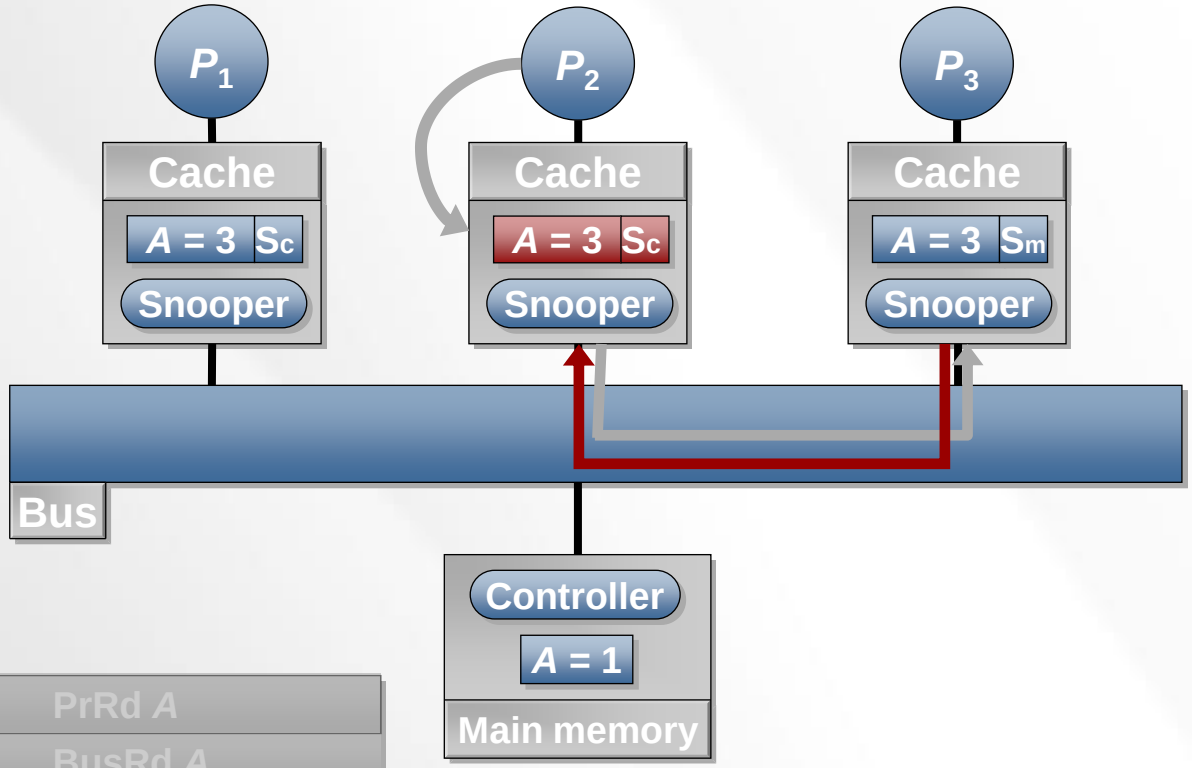
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_2	PrRd A
P_2	BusRd A
P_3	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Main memory controller observes the BusRd.



Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

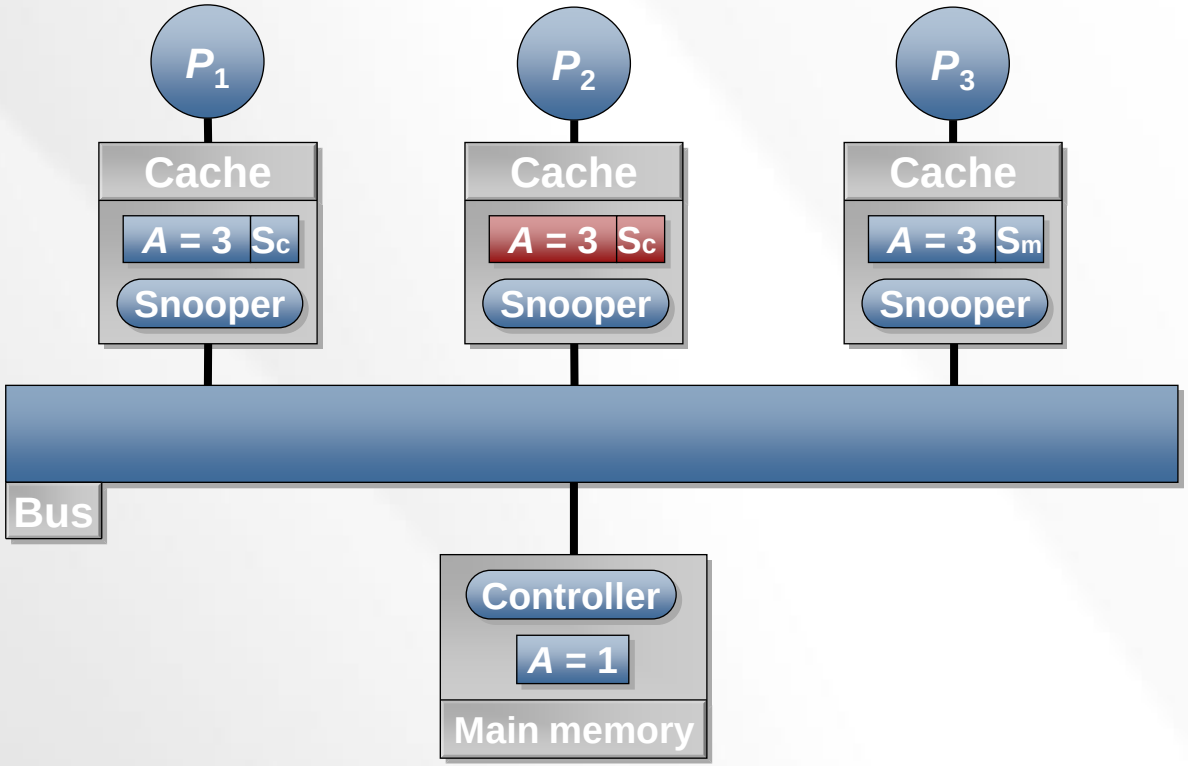
P_2	PrRd A
P_2	BusRd A
P_3	Flush

Note: Only the cache in state Sm is responsible for cache-to-cache transfer

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Operation completes.

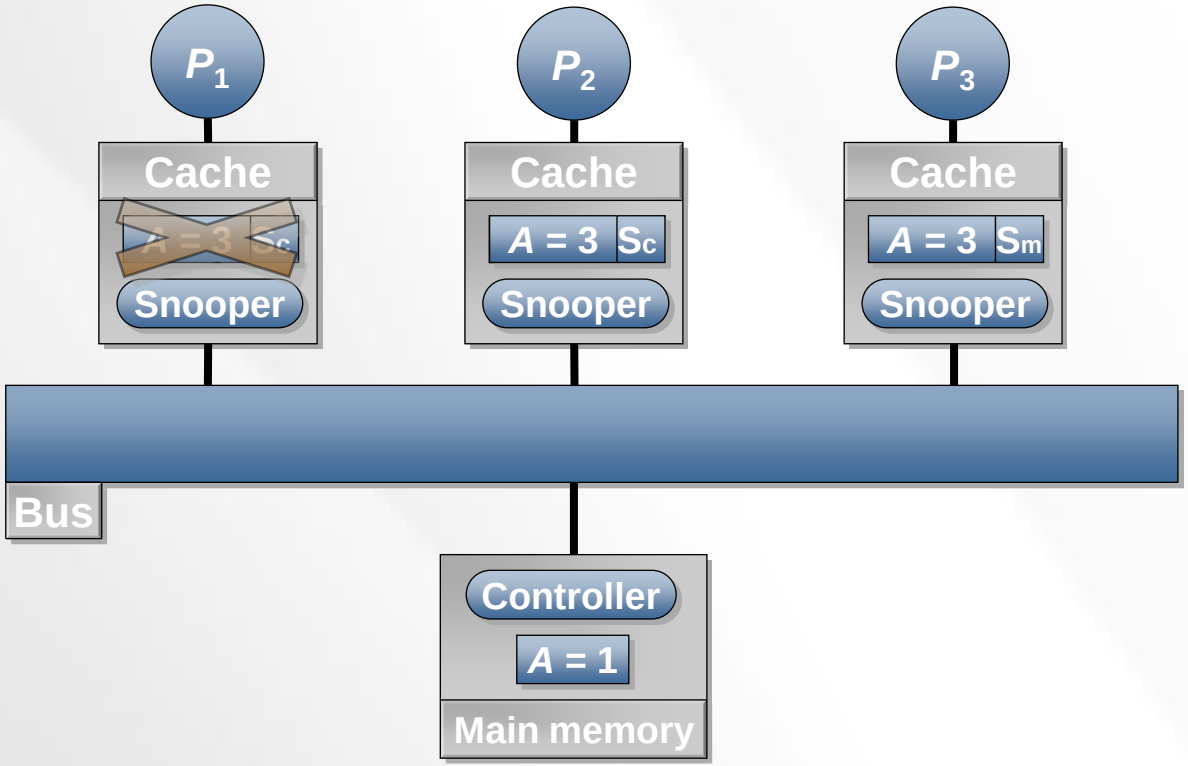


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

P_1 Replaces A

A evicted from P_1

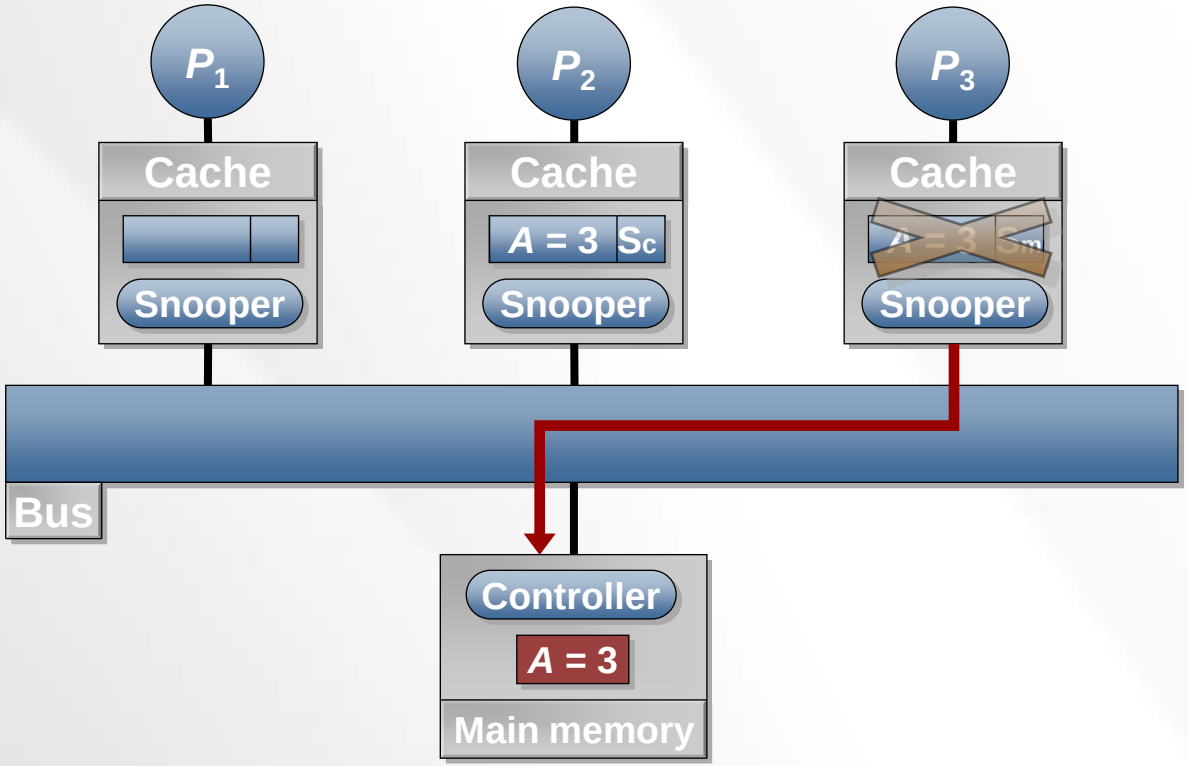


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

P_1 Replaces A

A evicted from P_3.

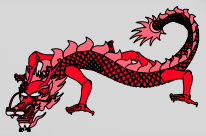


Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P3 replaces X
Owner responsible
for writing back to memory

vs. MSI or MESI where
write-back only when
the line is in M state

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon



Dragon Example

Proc Action	State P1	State P2	State P3	Bus Action	Data from
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	Sm	–	Sc	BusRd/Flush	P1 cache
W3	Sc	–	Sm	BusUpd/Upd	Own cache
R1	Sc	–	Sm	–	Own cache
R3	Sc	–	Sm	–	Own cache
R2	Sc	Sc	Sm	BusRd/Flush	P3 cache

Lower-Level Protocol Choices

- Can shared-modified state be eliminated?
  - If memory is updated too on BusUpd transactions (DEC Firefly)
  - Dragon protocol doesn't (assumes DRAM memory slow to update)
- Should replacement of an Sc block be broadcast?
  - Would allow last copy to go to Exclusive state and not generate updates
  - Replacement bus transaction isn't in critical path, but later update may be
- Shouldn't update local copy on write hit before controller gets bus
  - Can mess up serialization
- Coherence, consistency considerations much like write-through case

- In general, there are many subtle race conditions in protocols.

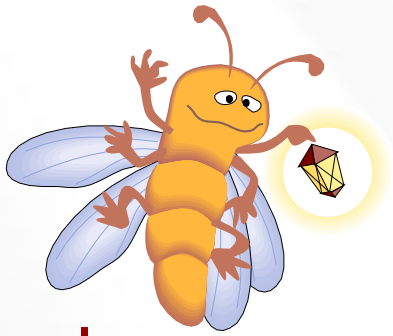
Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

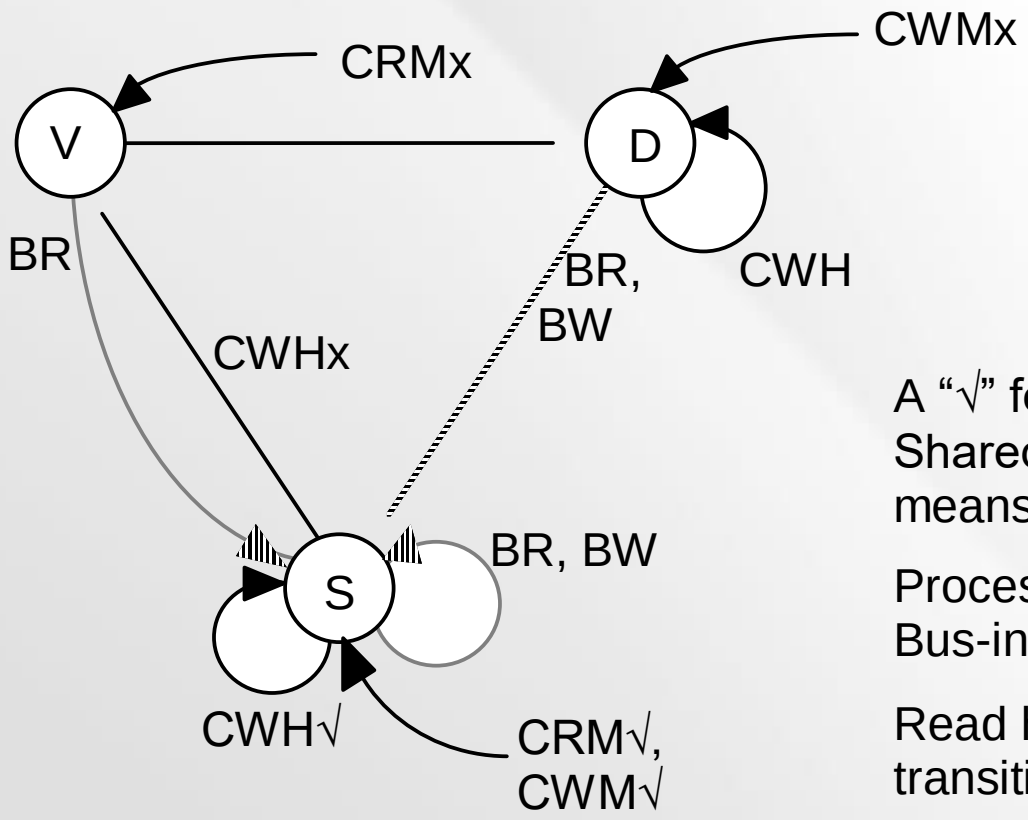
A Three-State Update Protocol

- Whenever a bus update is generated, suppose that main memory—as well as the caches—updates its contents.
- Then which state don't we need?
- What's the advantage, then, of having the fourth state?



- The Firefly protocol, named after a multiprocessor workstation developed by DEC, is an example of such a protocol.

Firefly State-Transition Diagram



Key:

CRM — CPU read miss
CWM — CPU write miss
CWH — CPU write hit
BR — bus read
BW — bus write

A "√" following a transition means SharedLine was asserted. An "x" means it was not.

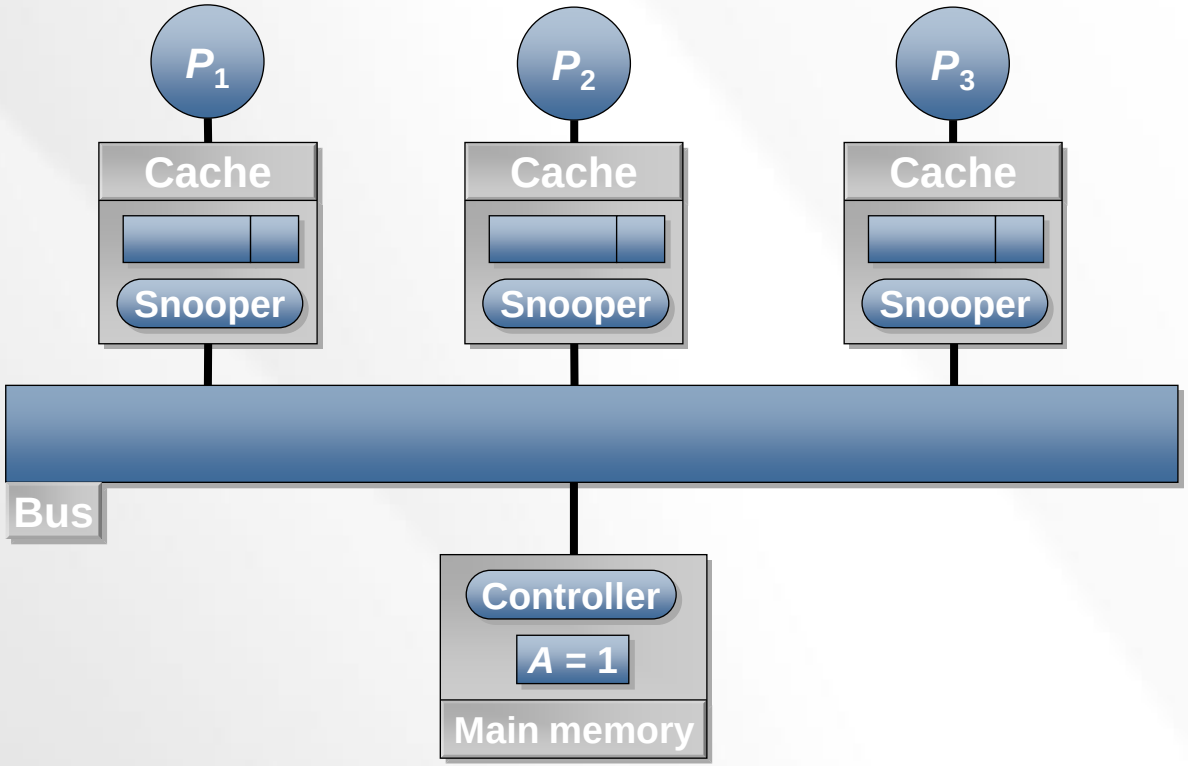
Processor-induced transitions ——
Bus-induced transitions .....

Read hits do not cause state transitions and are not shown.

- Answer some questions about this diagram.

Firefly Visualization

Start state. All caches empty and main memory has $A = 1$.

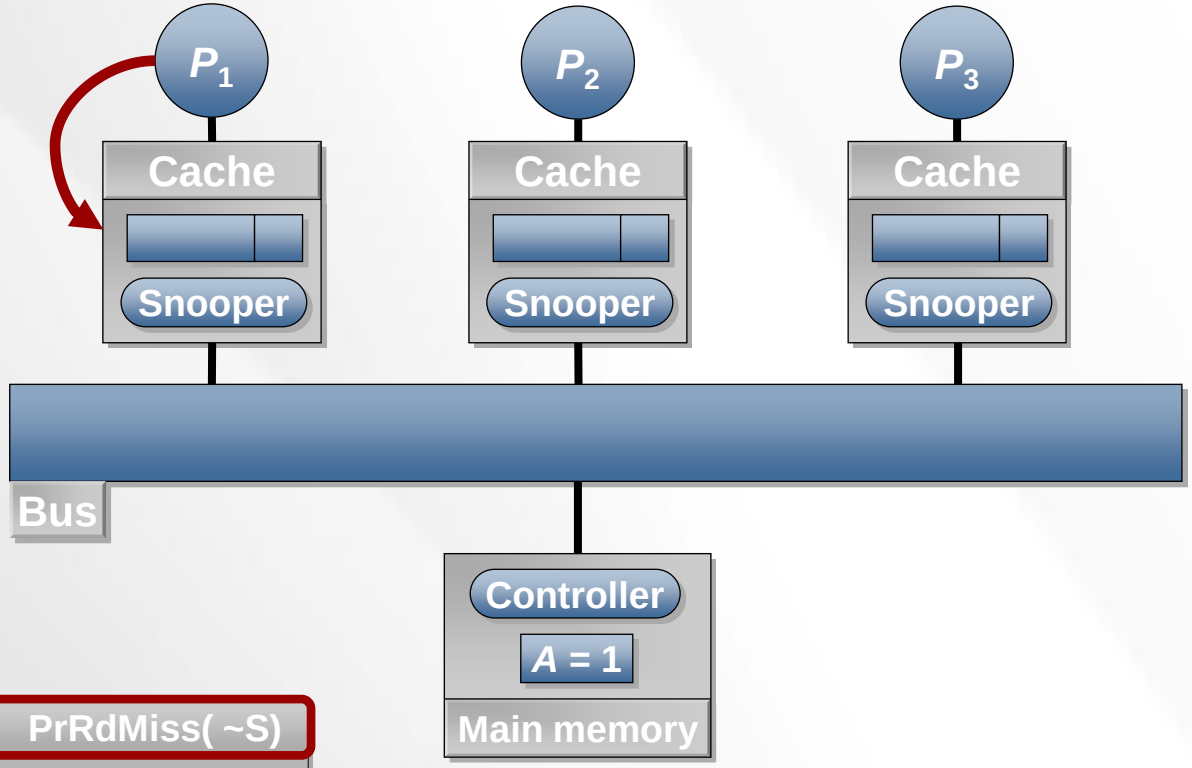


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 attempts to read A from its cache.



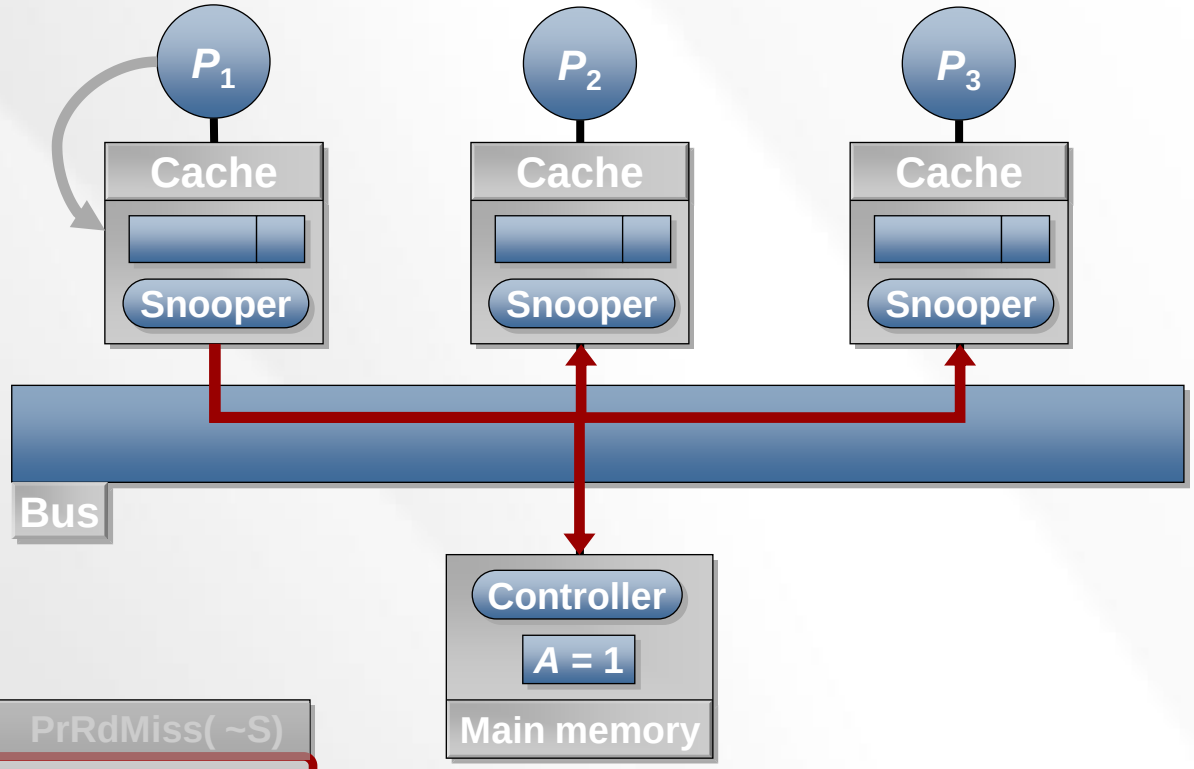
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRdMiss(~S)
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 issues a BusRd.



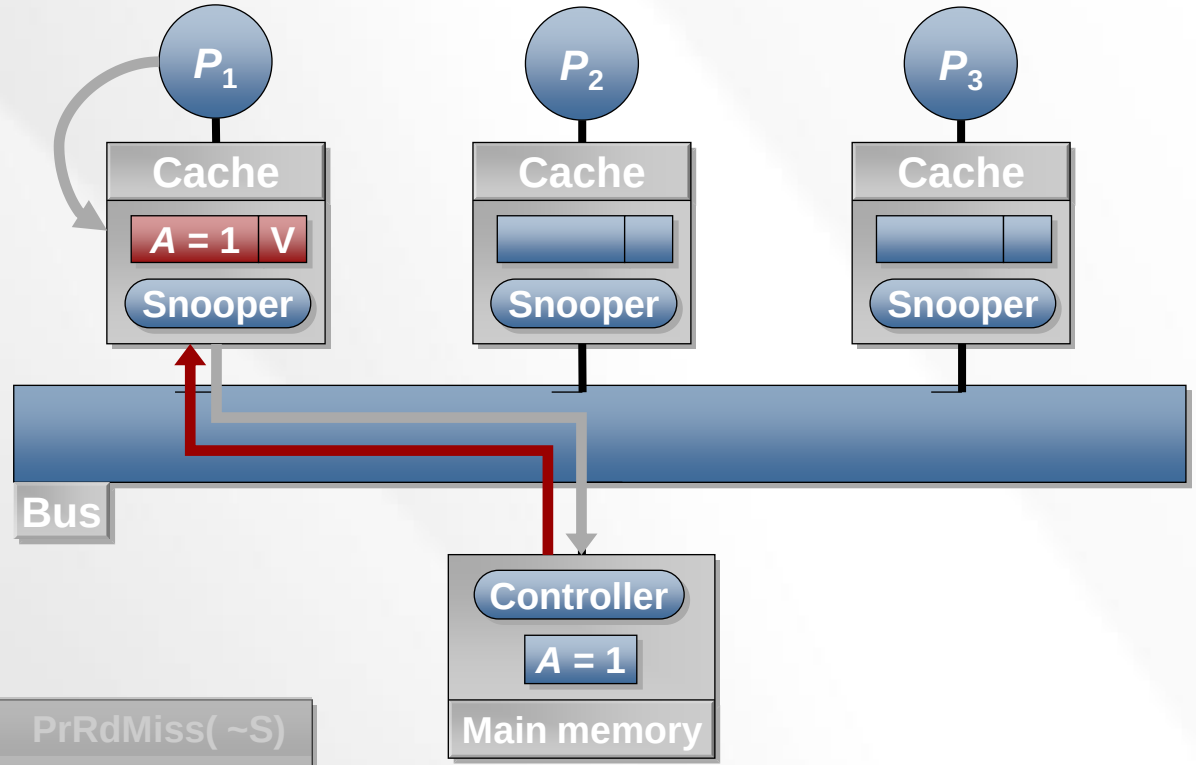
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRdMiss(~S)
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Main memory returns data to processor P_1 which updates its cache.



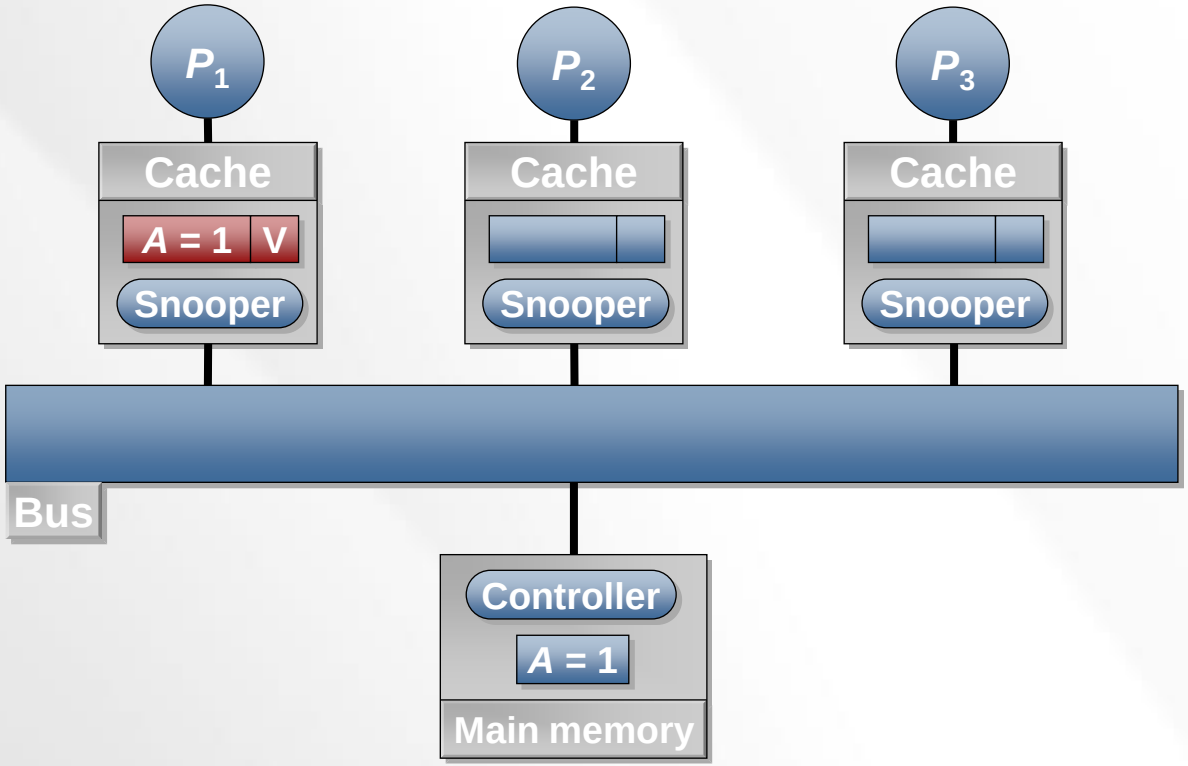
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_1	PrRdMiss(~S)
P_1	BusRd A
Mem	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Read operation completes.

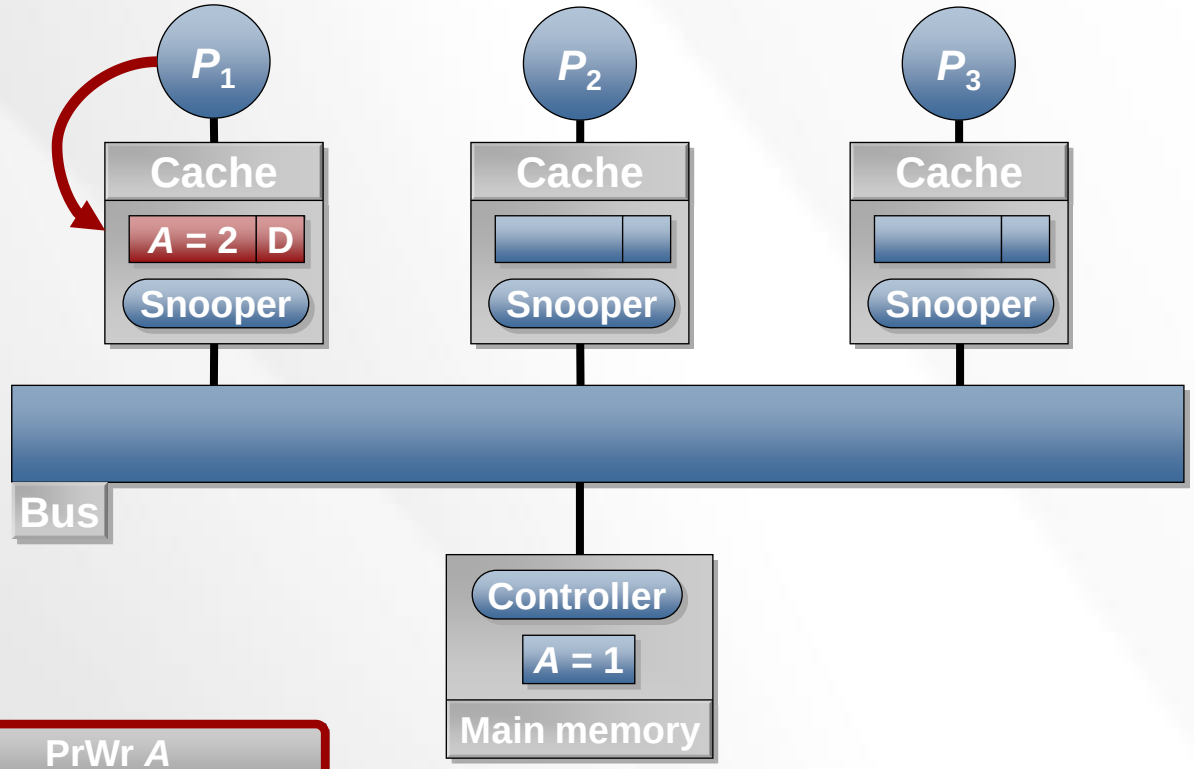


Trace	
P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Processor P_1 writes to its cache.



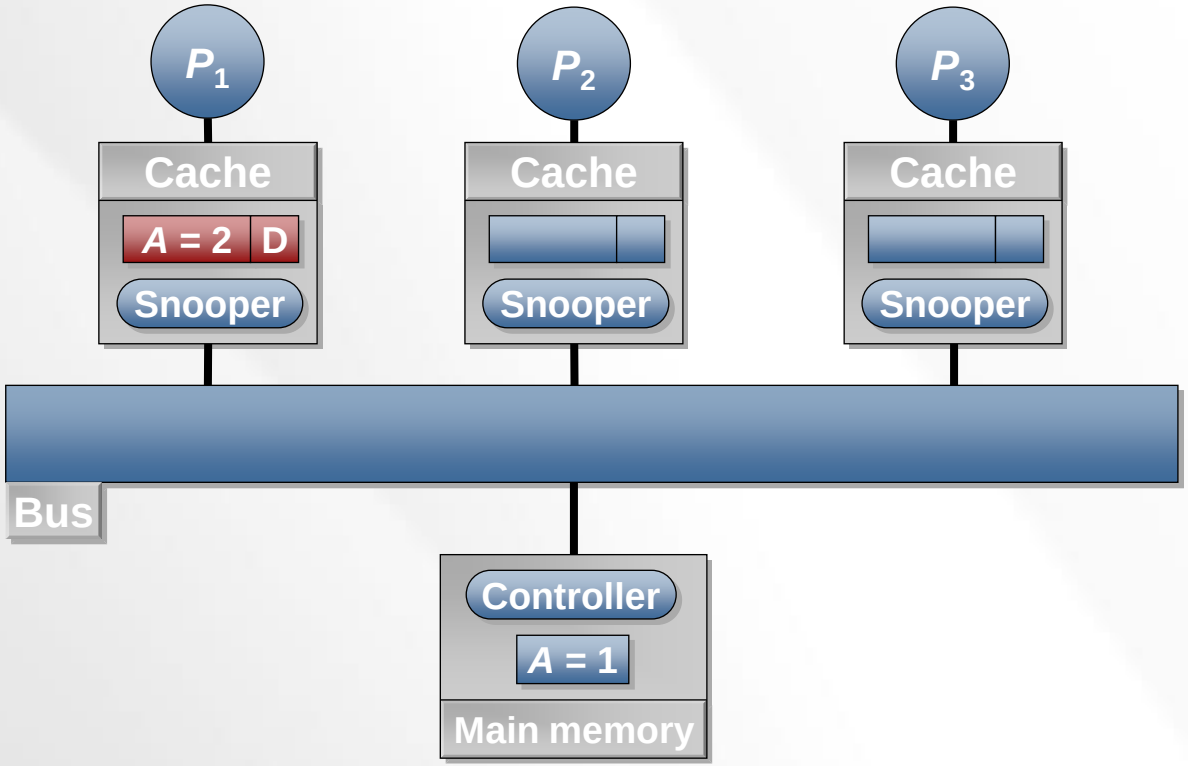
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_1 PrWr A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Writes $A = 2$

Write operation completes.

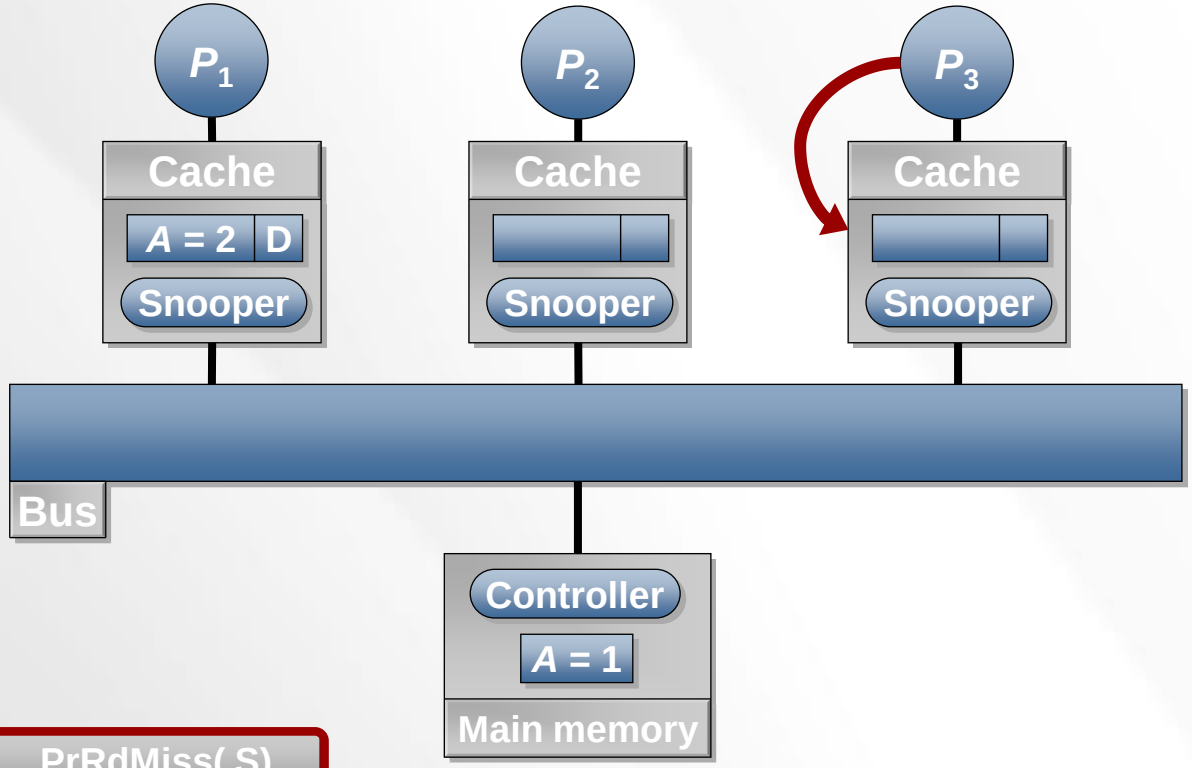


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 attempts to read A from its cache.



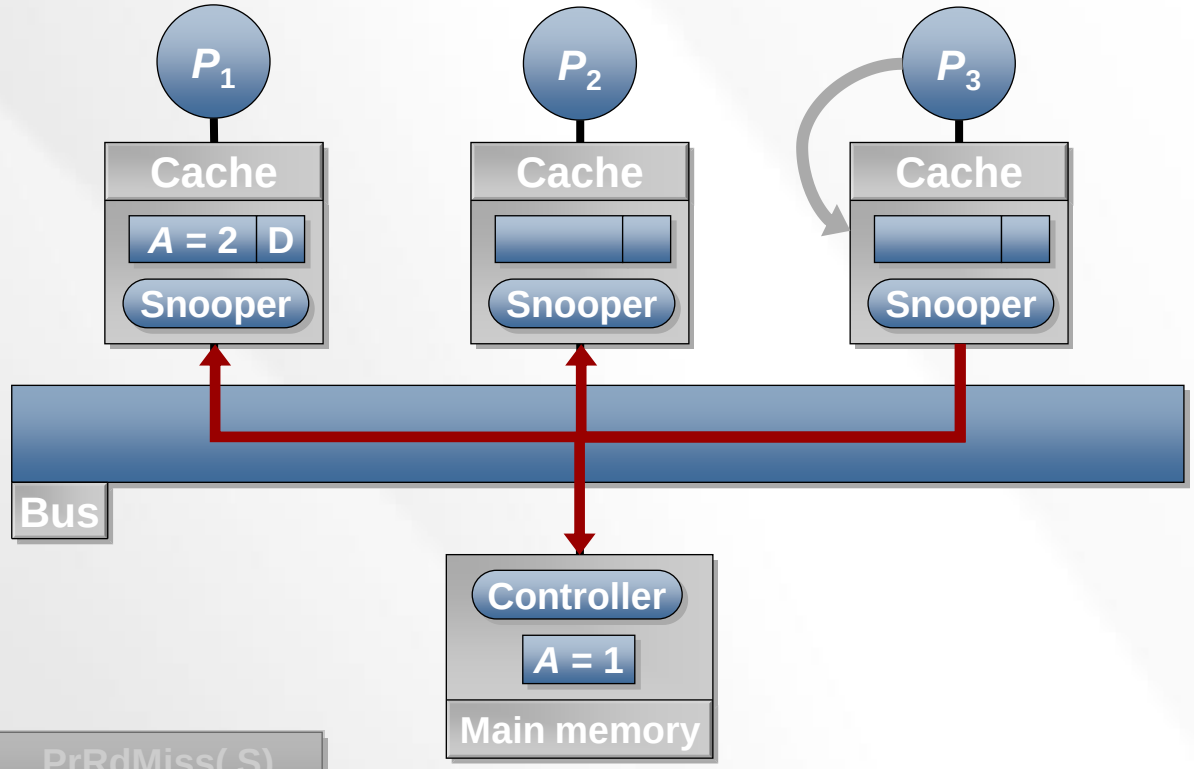
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRdMiss(S)
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 issues a BusRd.



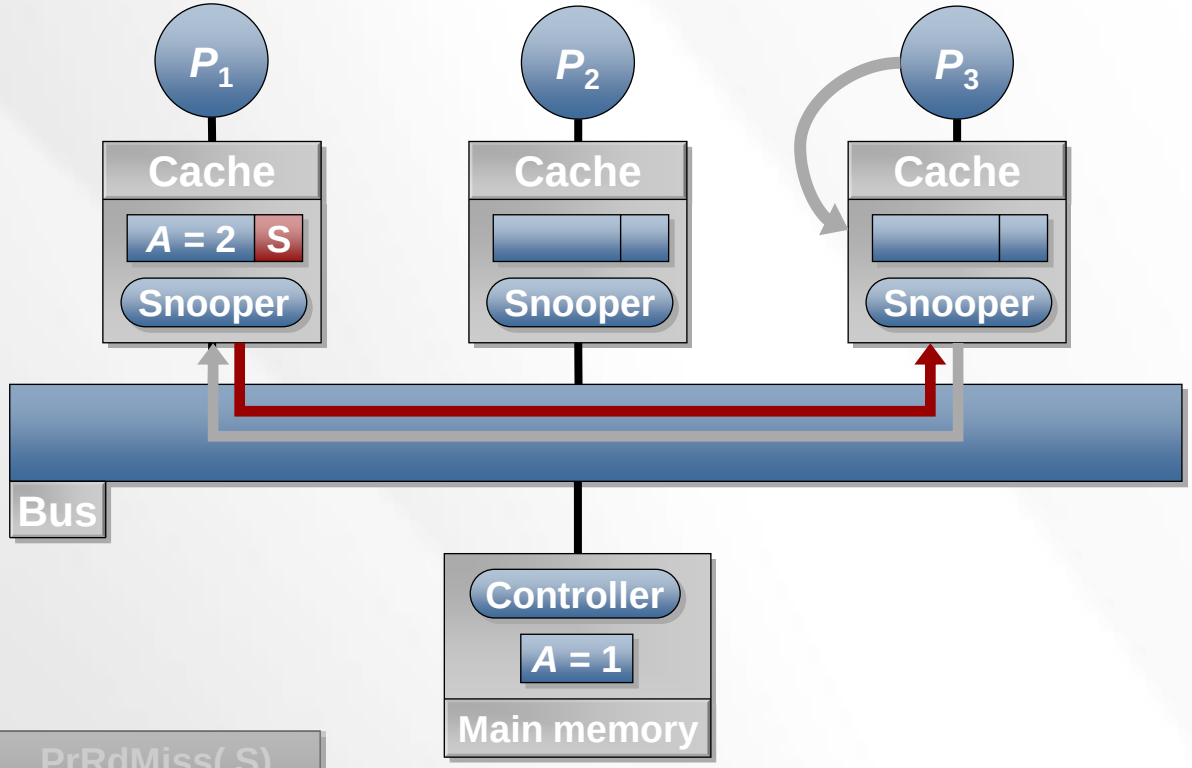
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRdMiss(S)
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_1 snoops the BusRd from processor P_3.



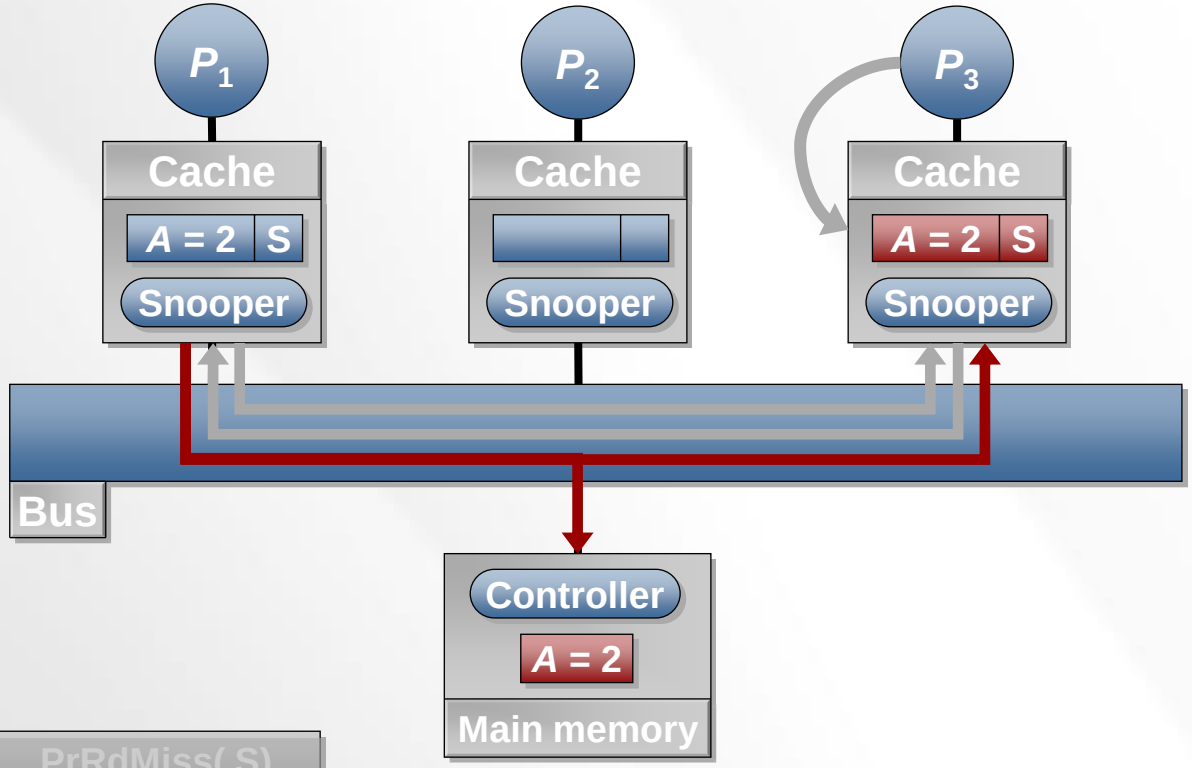
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRdMiss(S)
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_1 flushes, sending updated data to P_3 and main memory.



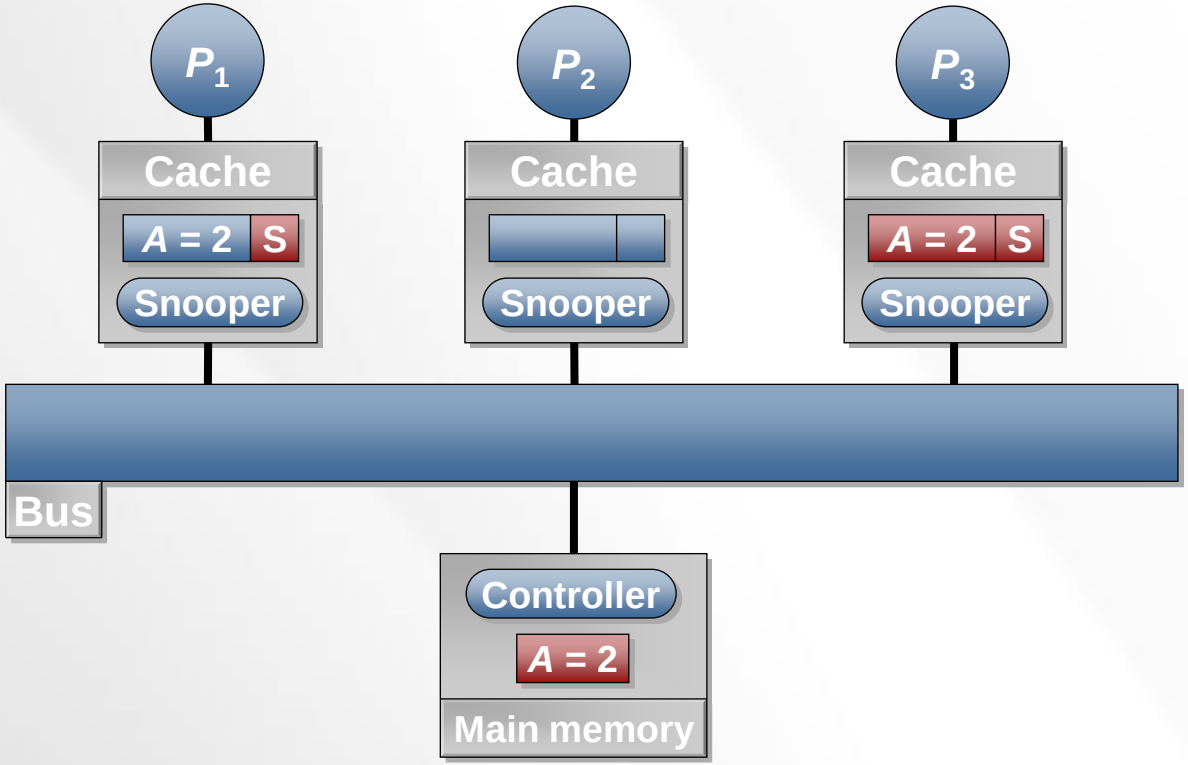
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrRdMiss(S)
P_3	BusRd A
P_1	snoops BusRd
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Read operation completes.

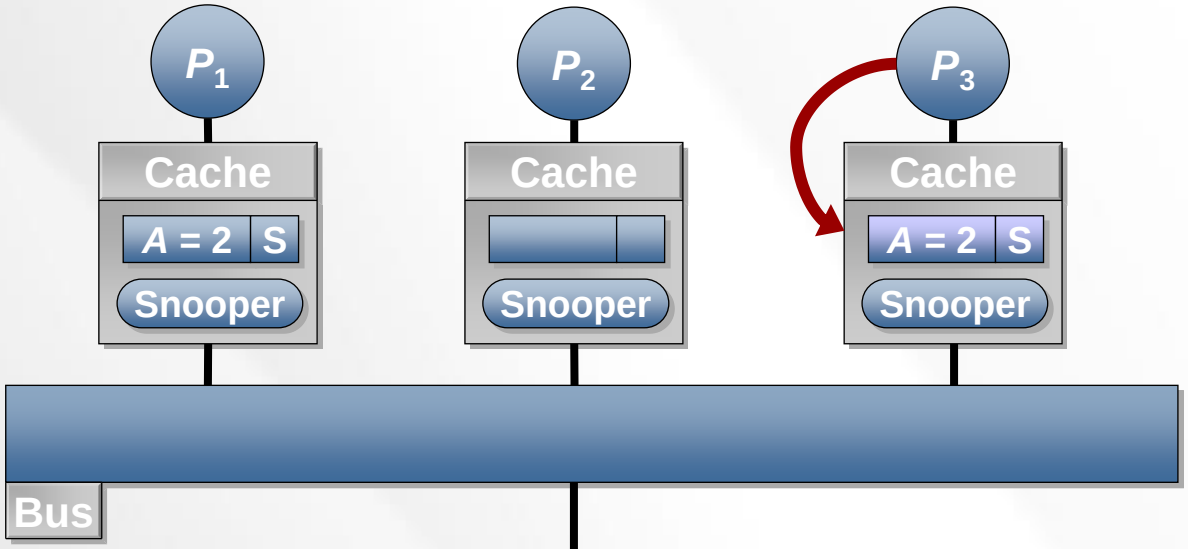


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 writes to its cache.



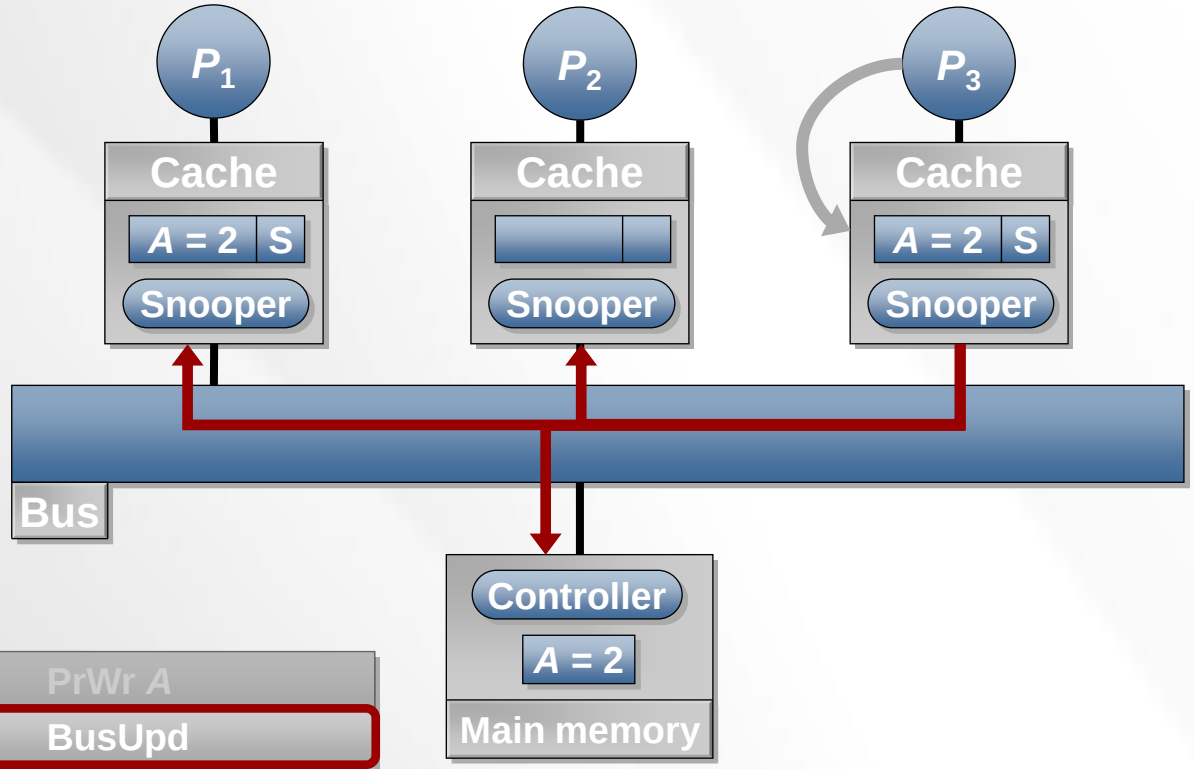
Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_3	PrWr A	
P_3	BusUpd	
P_1	snoops BusUpd	

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_3 issues a BusUpd request.



Trace		
P_1	Read A	
P_1	Write $A = 2$	
P_3	Read A	
P_3	Write $A = 3$	
P_1	Read A	
P_3	Read A	
P_2	Read A	

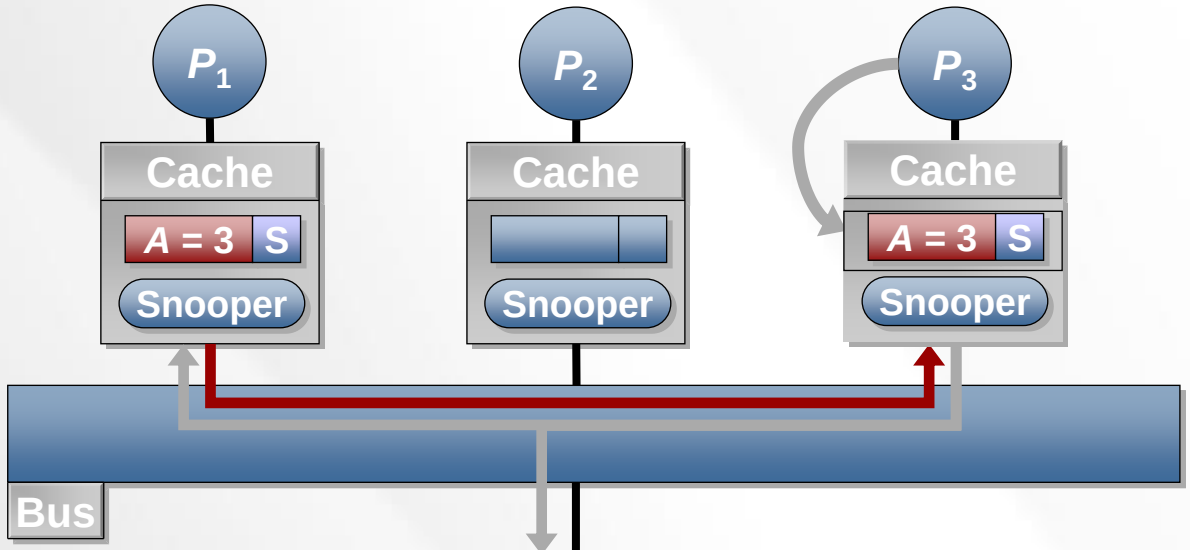
P_3	PrWr A	
P_3	BusUpd	
P_1	snoops BusUpd	

Note: BusUpd

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Processor P_1 snoops the BusUpd and updates its cache.



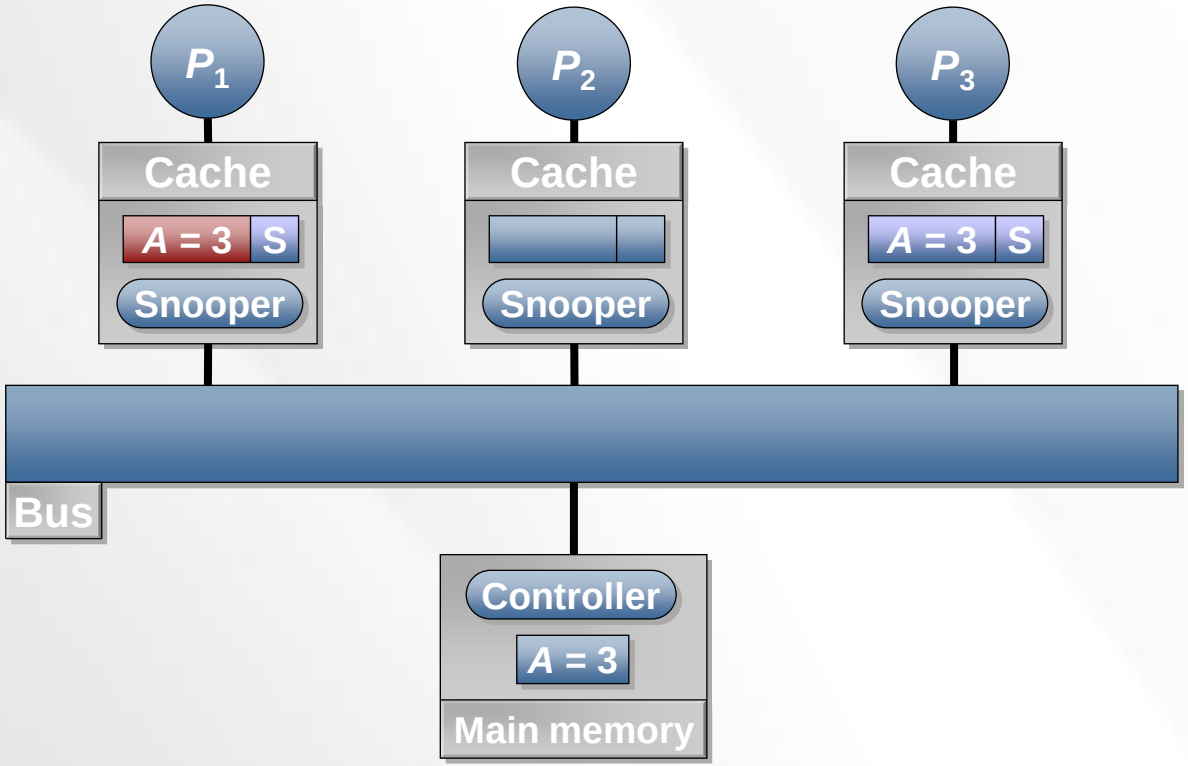
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	PrWr A
P_3	BusUpd
P_1	snoops BusUpd

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Writes $A = 3$

Write operation completes.

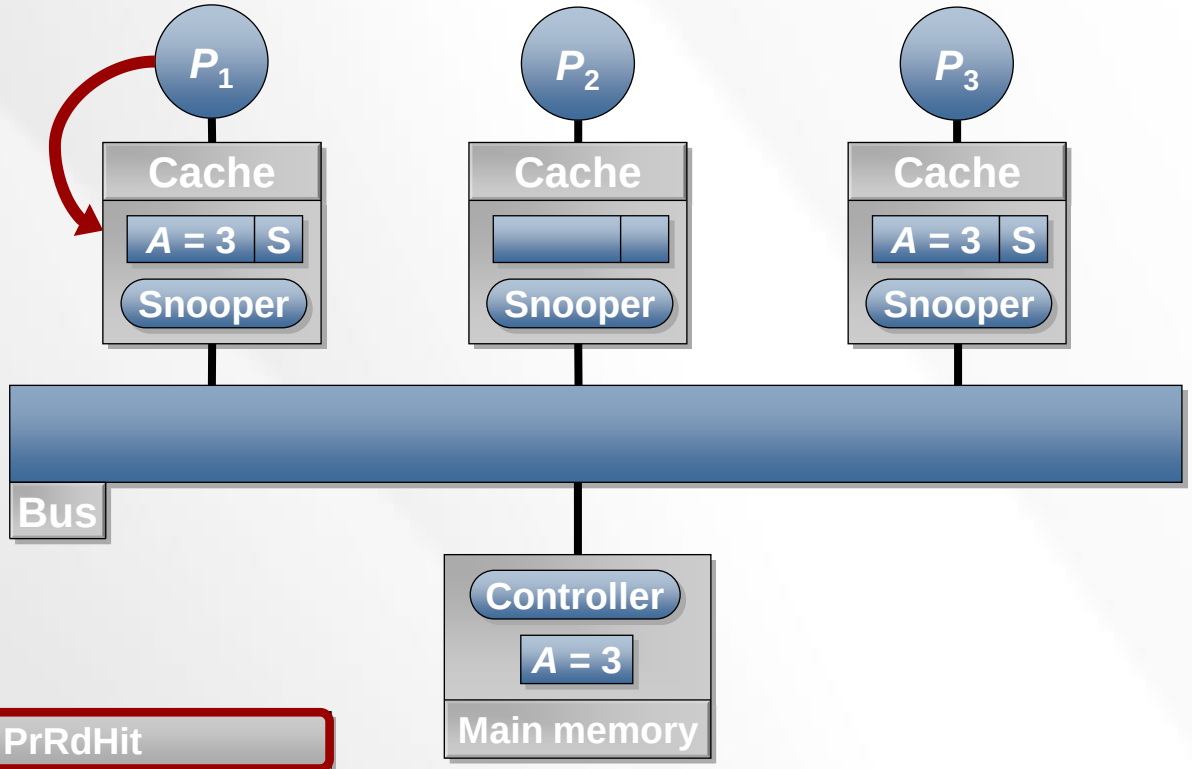


Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 reads from its cache.



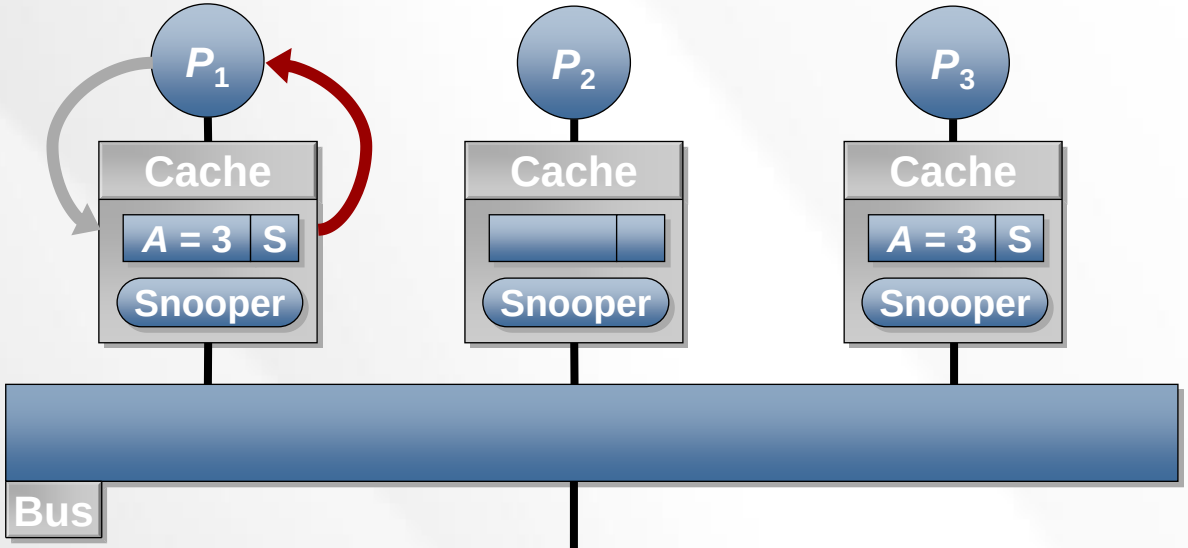
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	**PrRdHit**
P_3	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_1 Reads A

Processor P_1 reads from its cache.



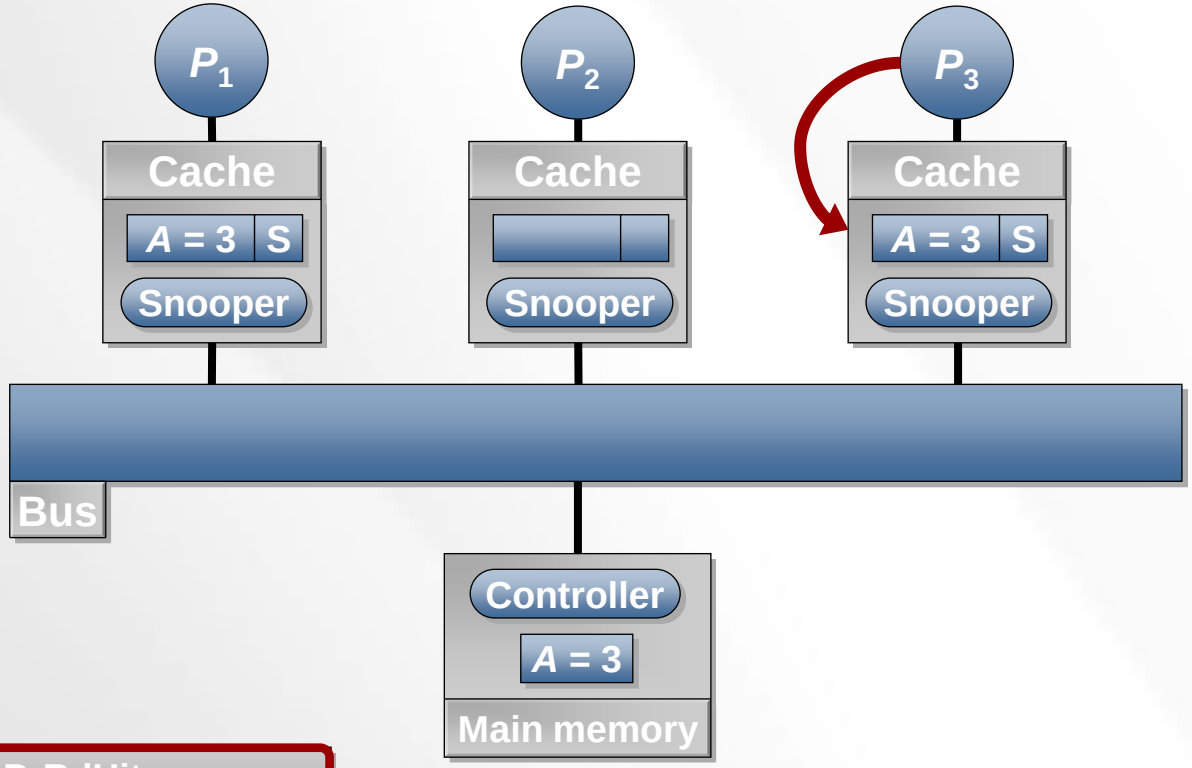
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3 PrRdHit
P_3 returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 reads from its cache.



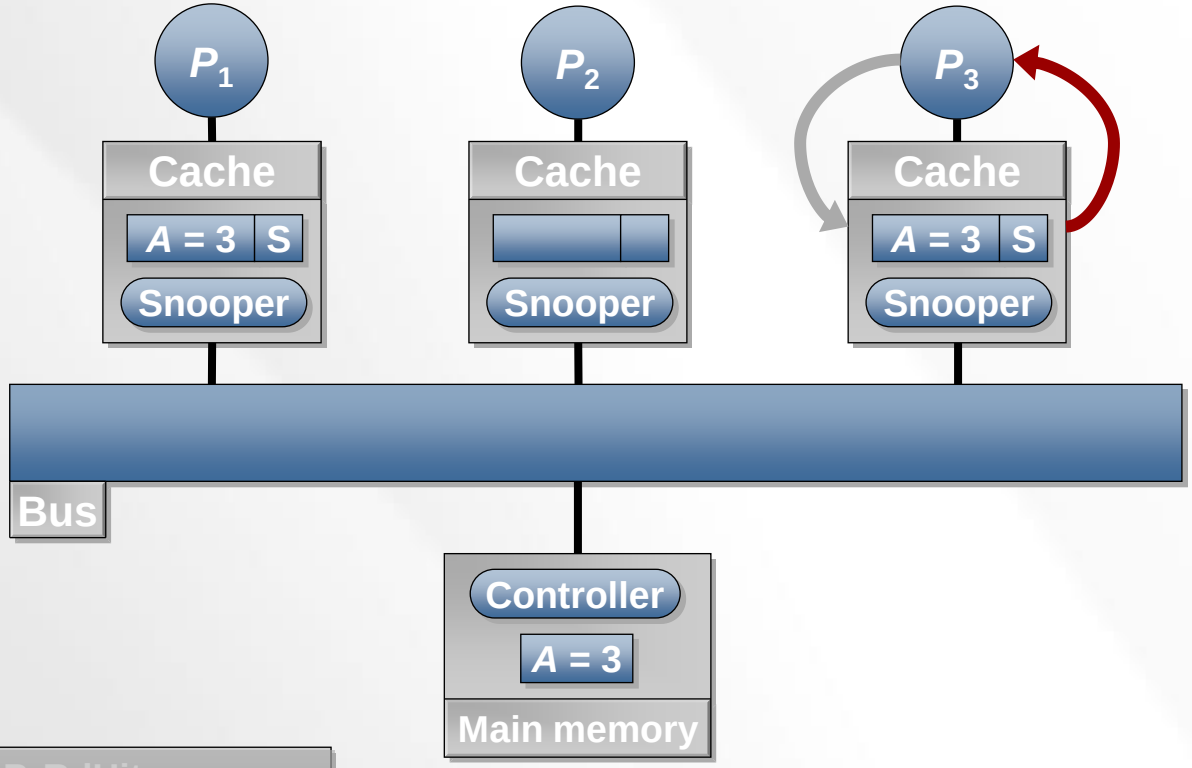
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3	**PrRdHit**
P_3	returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_3 Reads A

Processor P_3 returns valid data from its cache.



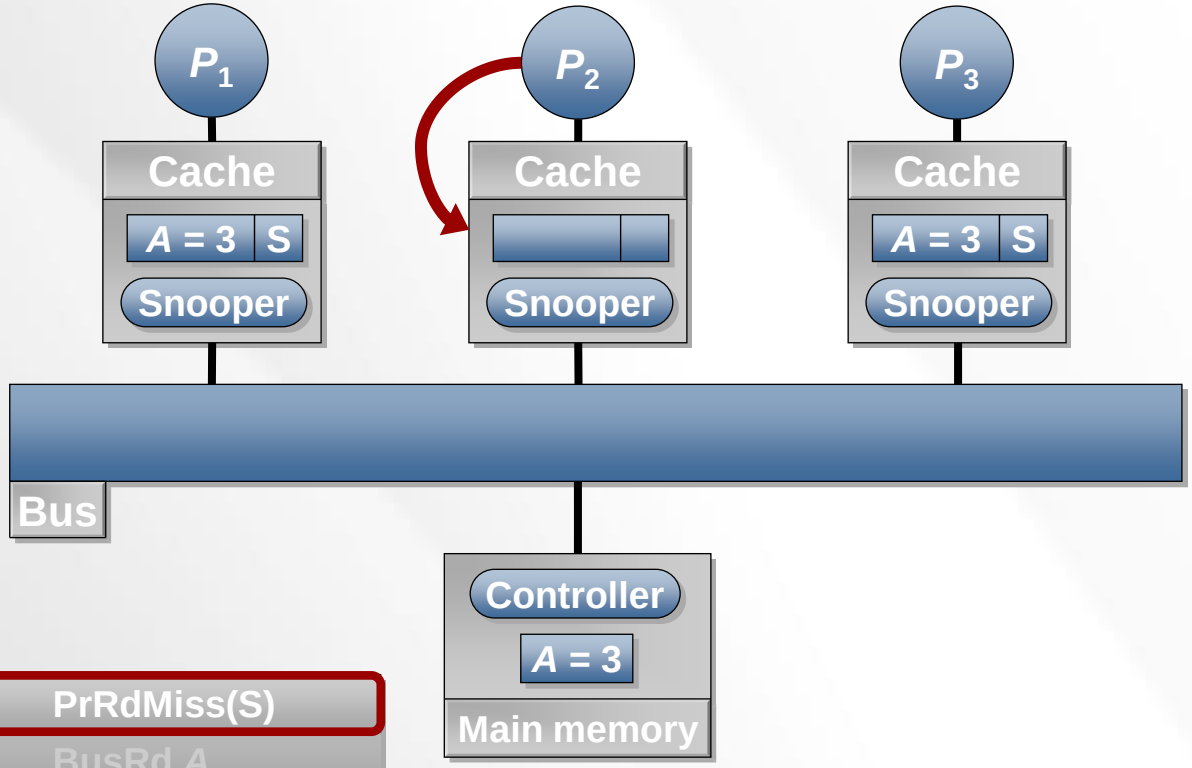
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_3 PrRdHit
P_3 returns data

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 reads from its cache.



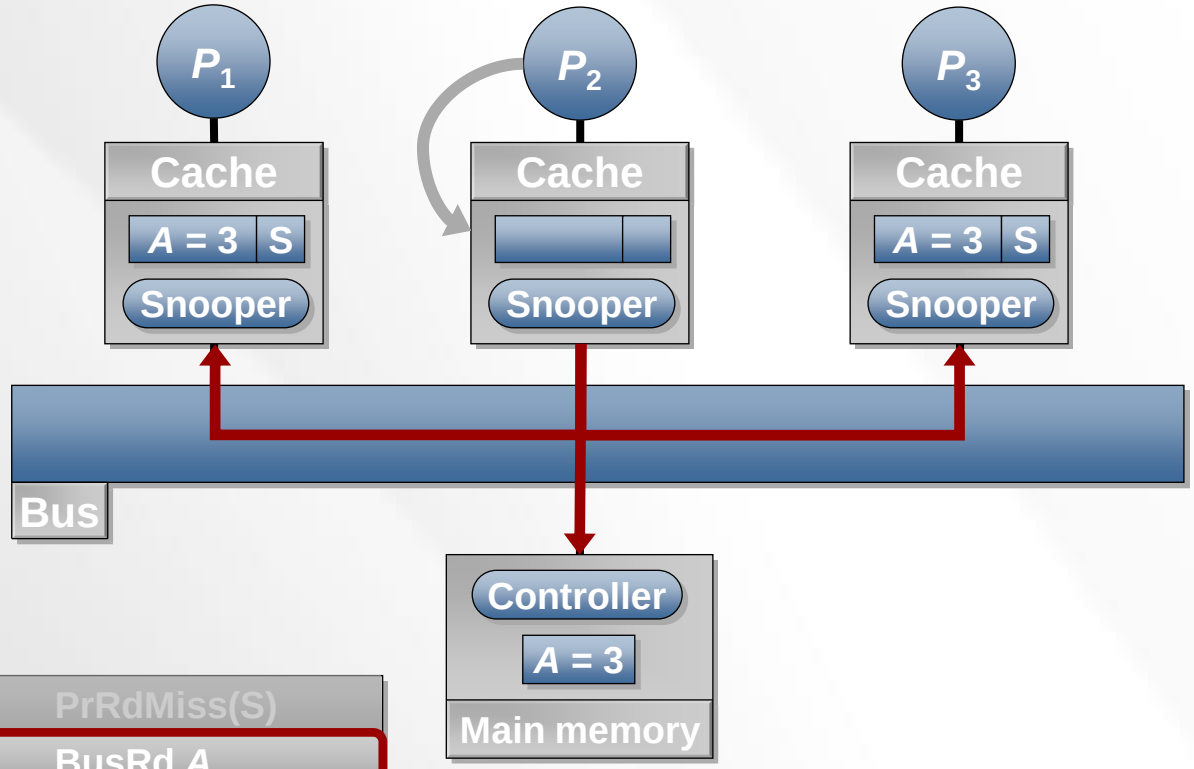
Trace		
P_1	Read A	
P_1	Write A = 2	
P_3	Read A	
P_3	Write A = 3	
P_1	Read A	
P_3	Read A	
P_2	Read A	

P_2	PrRdMiss(S)	
P_2	BusRd A	
P_1	Flush	

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Processor P_2 issues a BusRd request.



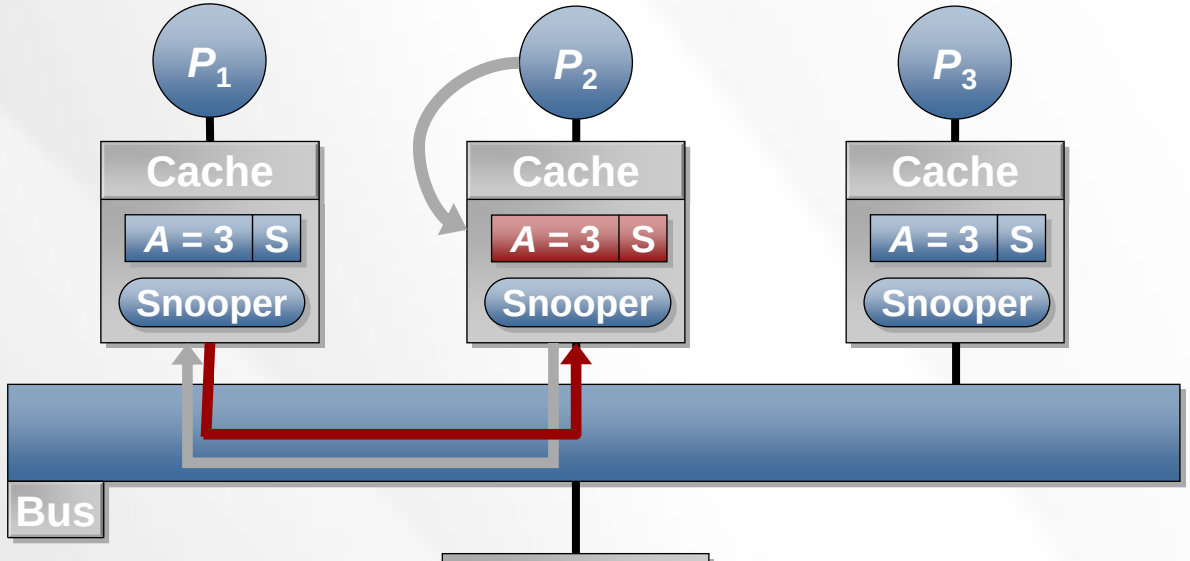
Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

P_2	PrRdMiss(S)
P_2	BusRd A
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Main memory controller observes the BusRd.



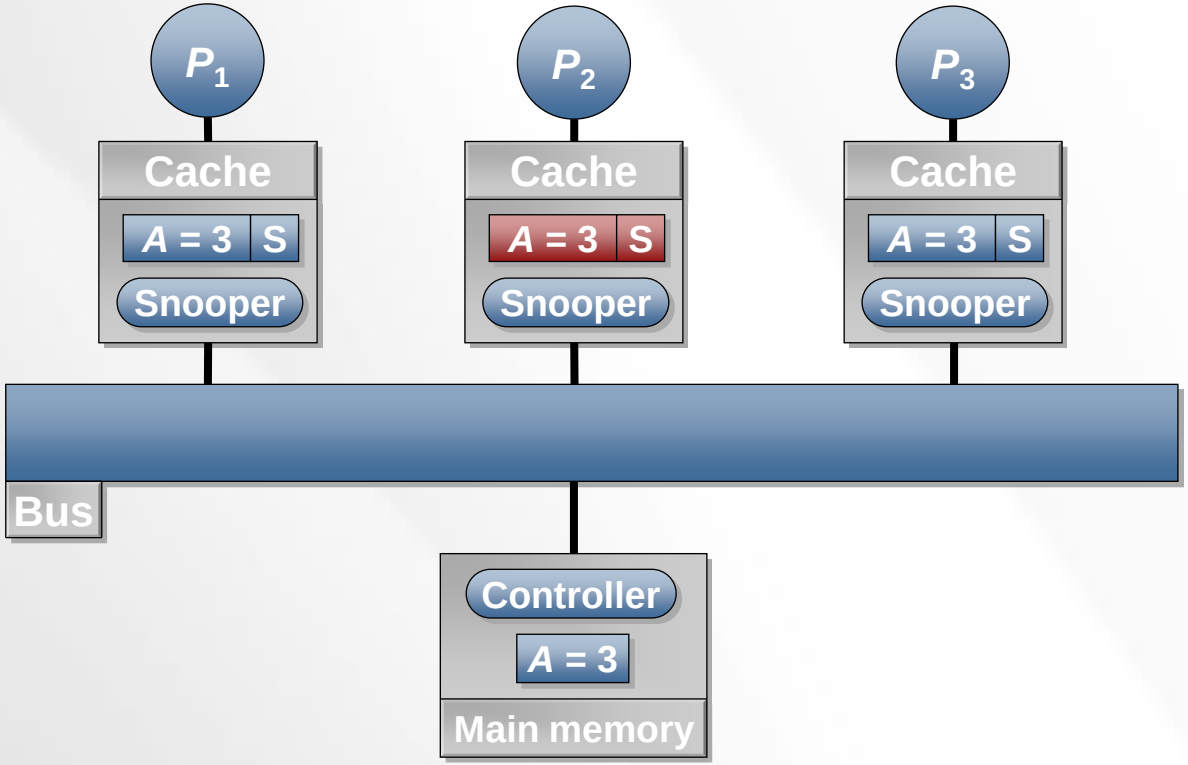
Trace
P_1 Read A
P_1 Write A = 2
P_3 Read A
P_3 Write A = 3
P_1 Read A
P_3 Read A
P_2 Read A

P_2	PrRdMis(S)
P_2	BusRd A
P_1	Flush

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Processor P_2 Reads A

Operation completes.



Trace
P_1 Read A
P_1 Write $A = 2$
P_3 Read A
P_3 Write $A = 3$
P_1 Read A
P_3 Read A
P_2 Read A

	Inv.	Upd.
3	MSI	Firefly
4	MESI	Dragon

Firefly Example

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	*V*	–	–	BusRd	Mem
W1	*D*	–	–	–	Own cache
R3	S	–	*S*	BusRd/Flush	P1 cache
W3	*S*	–	S	BusUpd	Own cache
R1	*S*	–	S	-	Own cache
R3	*S*	–	S	–	Own cache
R2	*S*	*S*	S	BusRd/Flush	P1 Cache

Firefly Example

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	*V*	–	–	BusRd	Mem
W1	*D*	–	–	–	Own cache
R3	S	–	*S*	BusRd/Flush	P1 cache
W3	*S*	–	S	BusUpd	Own cache
R1	*S*	–	S	-	Own cache
R3	*S*	–	S	–	Own cache
R2	*S*	*S*	S	BusRd/Flush	P1 Cache

Assessing Protocol Tradeoffs

- In the next lecture, we will look at results of comparing protocols by simulation.
- Methodology:
  - Use simulator; default 1MB, 4-way cache, 64-byte block, 16 processors. Some runs use 64K cache.
  - Focus on frequencies, not end performance for now
    - transcends architectural details, but not what we're really after
  - Use idealized memory performance model to avoid changes of reference interleaving across processors with machine parameters
    - Cheap simulation: no need to model contention