Computer Architecture(EECC551)

Cache Coherence Protocols

Presentation By: Sundararaman Nakshatra

Cache Coherence Protocols Overview

- **Multiple processor system**
  - System which has two or more processors working simultaneously
  - Advantages
- **Multiple Processor Hardware**
  - Types based on memory (Distributed, Shared and Distributed Shared Memory)
- **Need for CACHE**
  - Functions and Advantages
- **Problem when using cache for Multiprocessor System**
- **Cache Coherence Problem** (assuming write back cache)
- **Cache Coherence Solution**
- **Bus Snooping Cache Coherence Protocol**
- **Write Invalidate Bus Snooping Protocol**
  - For write through
  - For write back
  - Problems with write invalidate
- **Write Update or Write Invalidate?**
  - A Comparison
- **Some other Cache Coherence Protocols**
- **Enhancements in Cache Coherence Protocols**
- **References**

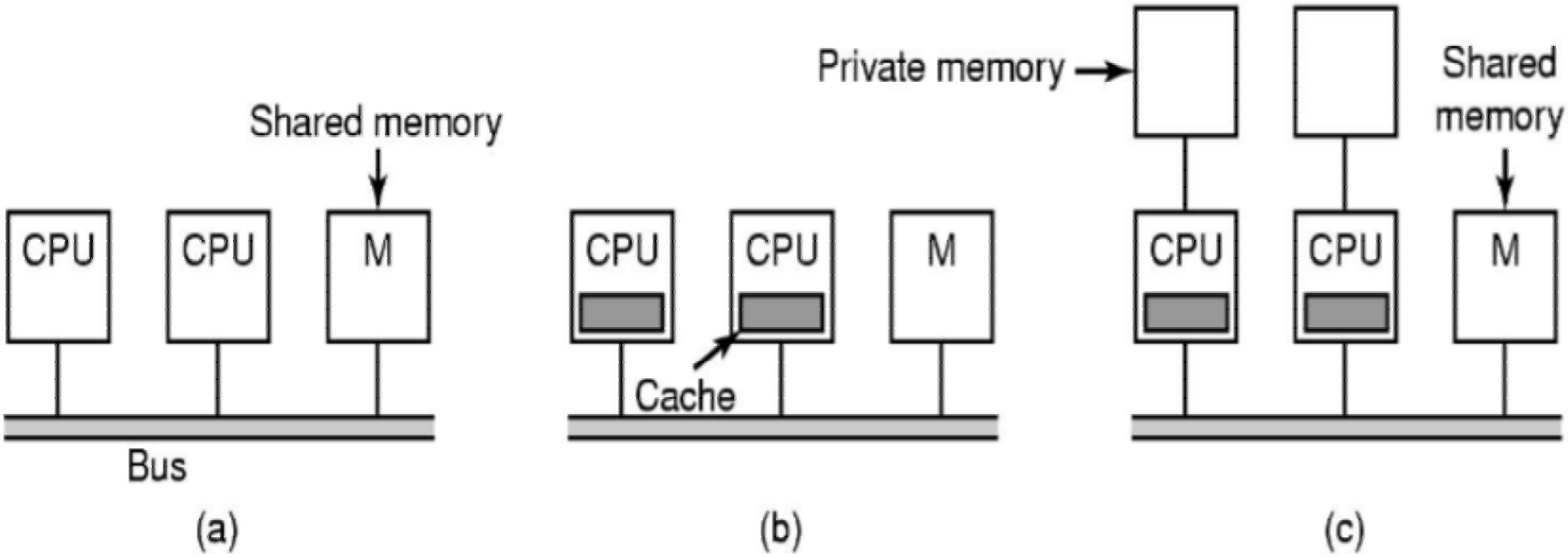
Multiple Processor System

A computer system which has two or more processors working simultaneously and sharing the same hard disk, memory and other memory devices.

Advantages:

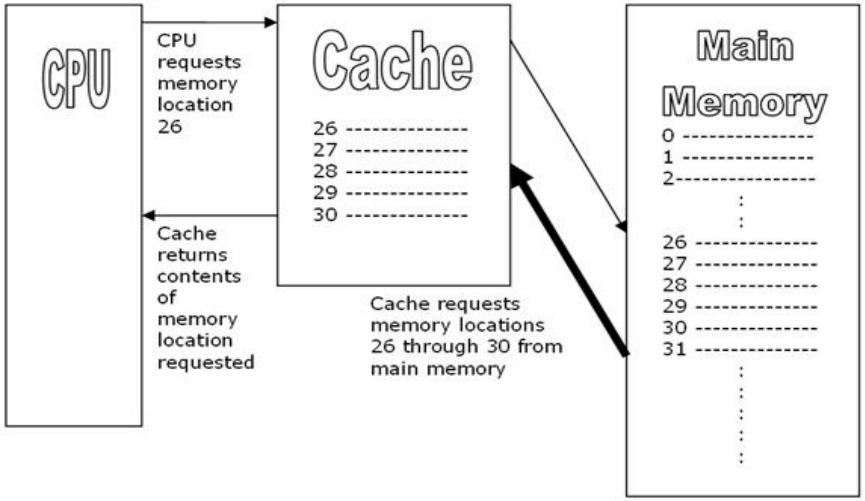
- **Reduced Cost:** Multiple processors share the same resources (like power supply and mother board).
- **Increased Reliability:** The failure of one processor does not affect the other processors though it will slow down the machine provided there is no master and slave processor.
- **Increased Throughput:** An increase in the number of processes completes the work in less time.

Multiple Processor Hardware



Bus-based multiprocessors

Why do we need cache?



Cache Memory : *"A computer memory with very short access time used for storage of frequently used instructions or data" – webster.com*

Cache memory function:

- an **extremely fast memory** that is built into a computer's central processing unit (CPU), or located next to it on a separate chip.
- The CPU uses cache memory to **store instructions** that are repeatedly required to run programs.
- When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache

Cache has various **advantages** in all shared memory multiprocessor system

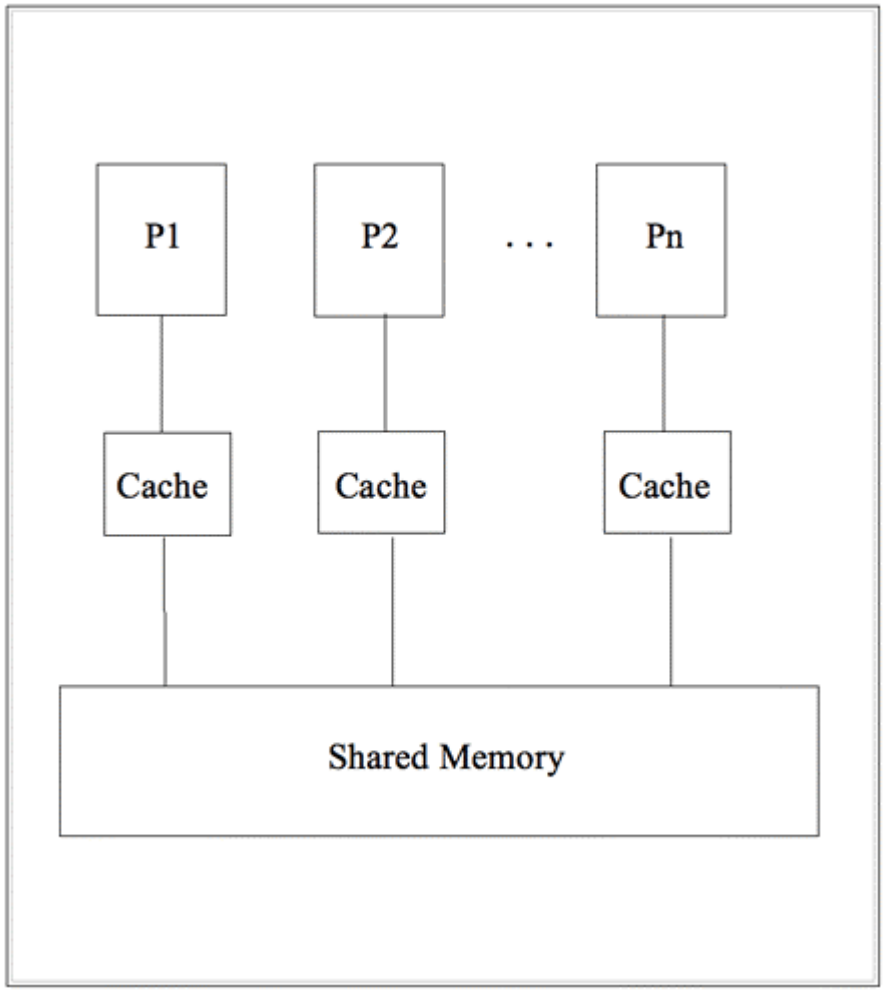
– To Reduce average data access time: The CPU does not have to use the motherboard's system bus for data transfer thereby improving the overall system speed.
– Reduced bandwidth demands placed on shared interconnect
–Volatile memory so its reusable and doesn't occupy much space.

(This diagram above illustrates level 2 cache. Level 1 cache is where the cache memory is built into the CPU. This diagram is referred from http://www.sqa.org.uk/e-learning/CompArch02CD/page_28.htm)

Problems when using Cache for Multiprocessor System

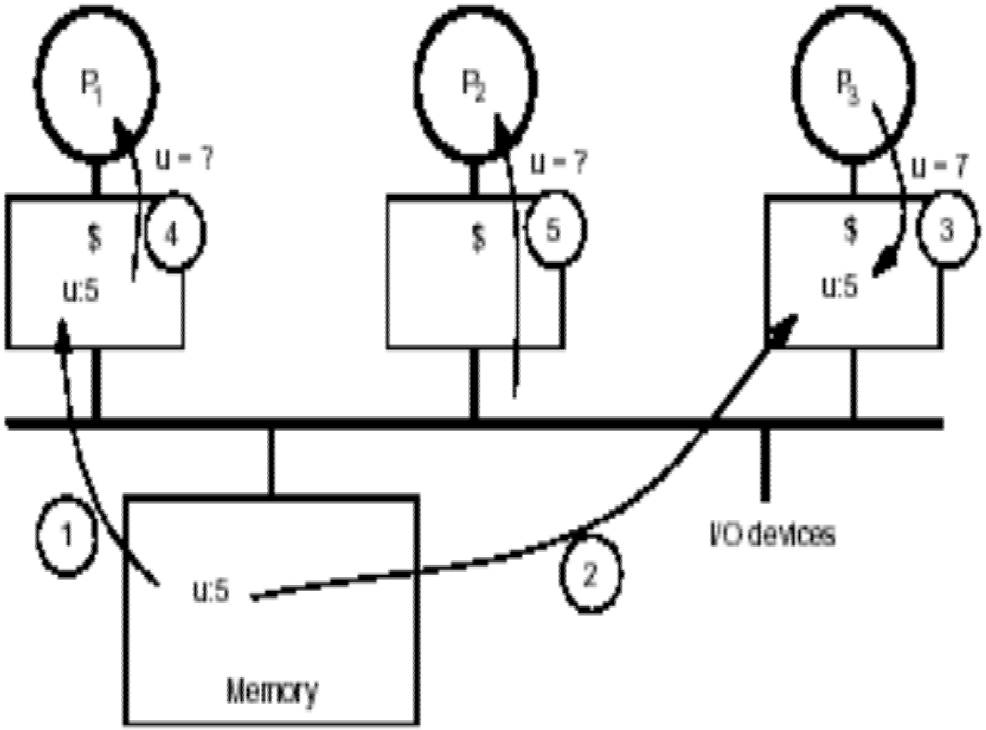
- Since all the processors share the same address space, it is possible for more than one processor to cache an address at the same time.(***coherence issue***)

- If one processor updates the data item without informing the other processor, inconsistency may result and cause incorrect execution.(***consistency issue***)



Cache Coherence Problem

(Write Back Caches Assumed)



1. P1 reads u=5 from memory

2. P3 reads u=5 from memory

3. P3 writes u=7 to local P3 cache

4. P1 reads old u=5 from local P1 cache

5. P2 reads old u=5 from memory

- – Processors see different values for u after event 3.
- – With write back caches, a value updated in cache may not have been written back to memory:
- – Processes even accessing main memory may see very stale value.
- – Unacceptable: leads to incorrect program execution.

Cache Coherence Solution

• **Bus-Snooping Protocols: (Not scalable)** Used in **bus-based systems** where all the processors observe memory transactions and take proper action to **invalidate or update the local cache** content if needed.

Snoopy

–**Send** all **requests for data** to all processors
–Processors **snoop** to see if they have a copy and respond accordingly
–Requires **broadcast**, since caching information is at processors
–Works well with **bus** (natural broadcast medium)
–Dominates for **small scale machines** (most of the market)

• **Directory Schemes:** Used in **scalable cache-coherent distributed memory multiprocessor systems** where cache directories are used to keep a record on where copies of cache blocks reside.

–**Keep track** of what is being shared in **1 centralized place** (logically)
–**Distributed memory** => distributed directory for scalability
(avoids bottlenecks)
–Send **point-to-point requests** to processors via network
–Scales **better** than Snooping
–Actually **existed BEFORE** Snooping-based schemes

Bus-Snooping Cache Coherence Protocols

Key Features:

- Transactions on bus are visible to all processors.
- Processors or bus-watching (bus snoop) mechanisms can snoop (monitor) the bus and take action on relevant events (e.g. change state) to ensure data consistency among private caches and shared memory.

Basic Protocol Types:

- *Write-invalidate:*

Invalidate all remote copies of cache when a local cache block is updated.

- *Write-update:*

When a local cache block is updated, the new data block is broadcast to all caches containing a copy of the block for updating them.

Write-invalidate Bus-Snooping Protocol

(For Write-Through Caches)

The state of a cache block copy of local processor can take one of the two states :

Valid State:

- All processors can read safely.
- Local processor can also write

Invalid State: (not in cache)

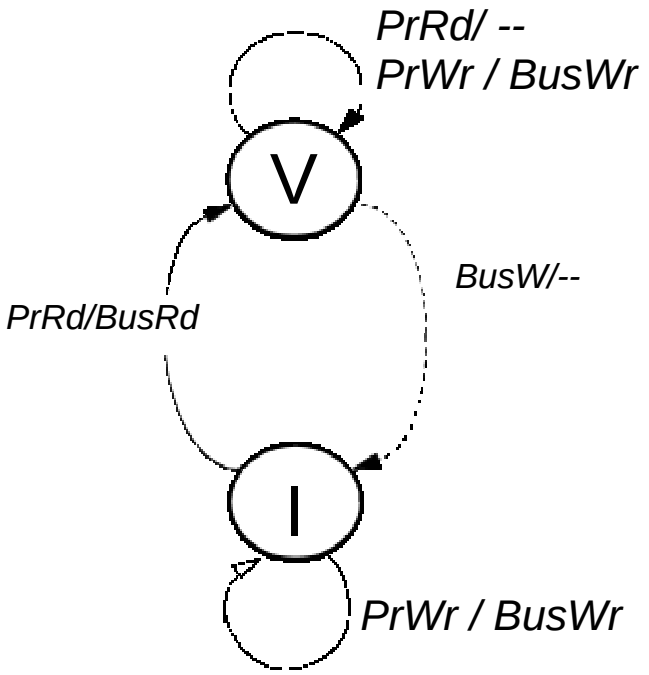
- Block being invalidated.
- Block being replaced
- When a remote processor writes to its cache copy, all other cache copies become invalidated.

(Bus write cycles are higher than bus read cycles due to request invalidations to remote caches)

Write-Invalidate Bus-Snooping Protocol

(For Write-Through Caches)

- Two states per block in each cache
  - States similar to a uniprocessor cache
  - Hardware state bits associated with blocks that are in the cache
  - Other blocks can be seen as being in invalid (not-present) state in that cache
- Writes invalidate all other caches
  - No local change of state
  - Multiple simultaneous readers of block, but write invalidates them



V = Valid
I = Invalid
A/B means if A is observed B is generated.
Processor Side Requests:
read (PrRd)
write (PrWr)
Bus Side or snooper/cache controller Actions:
bus read (BusRd)
bus write (BusWr)

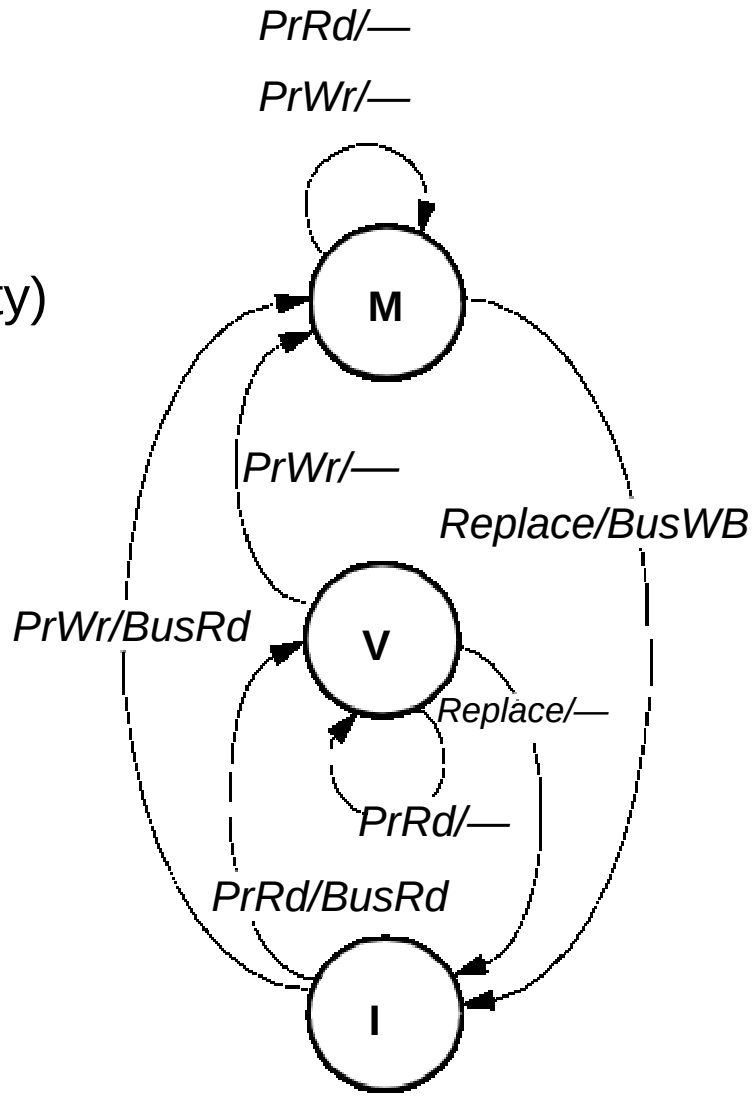
Problems with Write Through cache

- **High bandwidth requirements:**
  - Every write from every processor goes to shared bus and memory.
  - Consider 200MHz, 1 CPI processor, and 15% of the instructions are
    8-byte stores.
  - Each processor generates 30M stores or 240MB data per second.
  - 1GB/s bus can support only about 4 processors without saturating.
  - Write-through especially is unpopular for SMPs.

- **Write-back caches absorb most writes as cache hits:**
  - Write hits don**'t go on bus.**
  - But now how do we ensure write propagation and serialization?
    - **Requires more sophisticated coherence protocols.**

Basic Write-Invalidate Bus-Snooping Protocol (For Write-Back Caches)

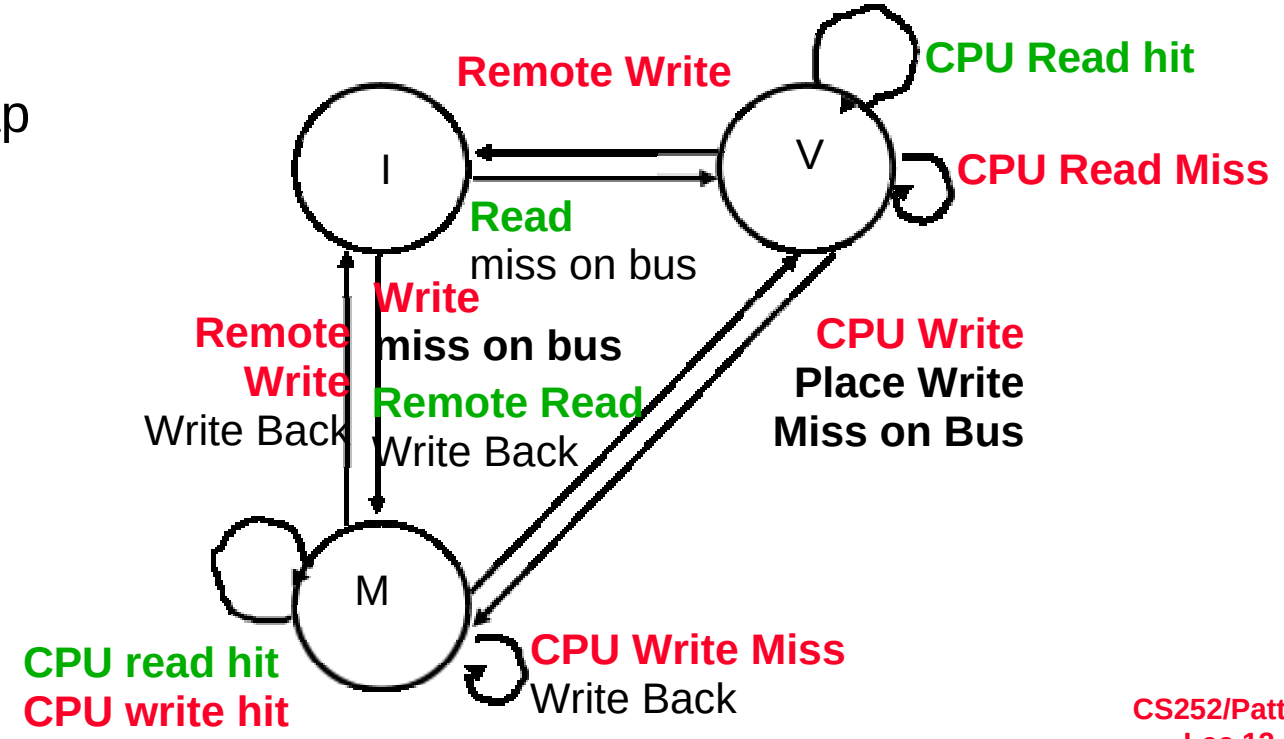
- Processor / Cache Operations
  - **PrRd**, **PrWr**, block **Replace**
- States
  - **Invalid**, **Valid** (clean), **Modified** (dirty)
- Bus Transactions
  - Bus Read (**BusRd**), Write-Back (**BusWB**)
  - Only cache-block are transfered
- Can be adjusted for cache coherence
  - Treat Valid as **Shared**
  - Treat Modified as **Exclusive**
- Introduce one new bus transaction
  - **Bus Read-eXclusive (BusRdX)**
  - For purpose of modifying (read-to-own)



Example

	Processor 1			Processor 2			Bus				Memory	
	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2

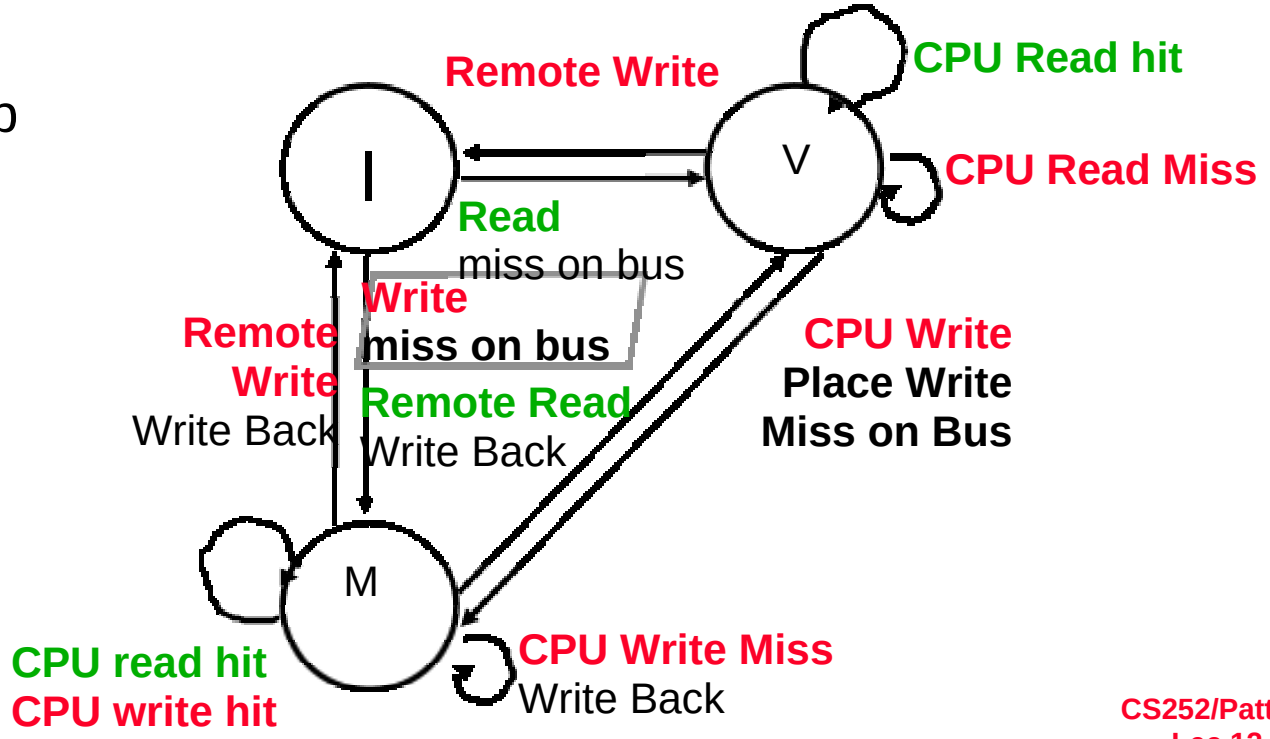


Example: Step 1

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.

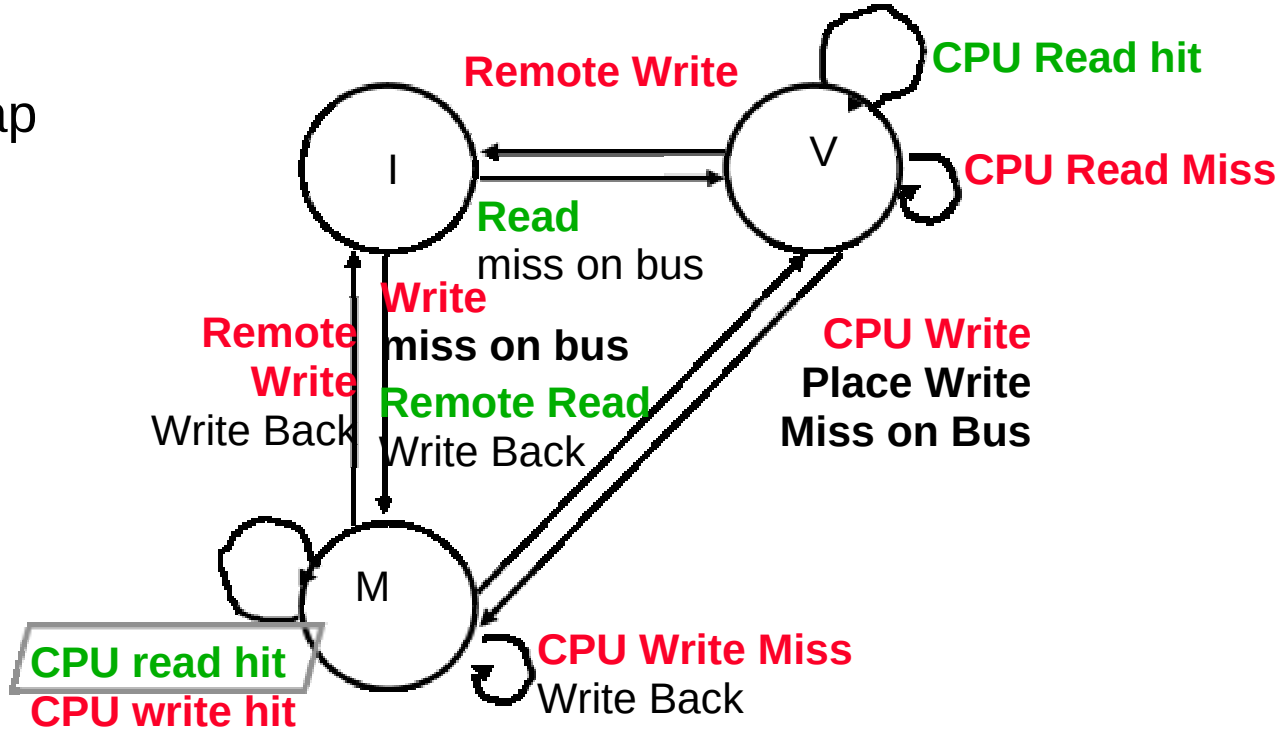
Active arrow =



Example: Step 2

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

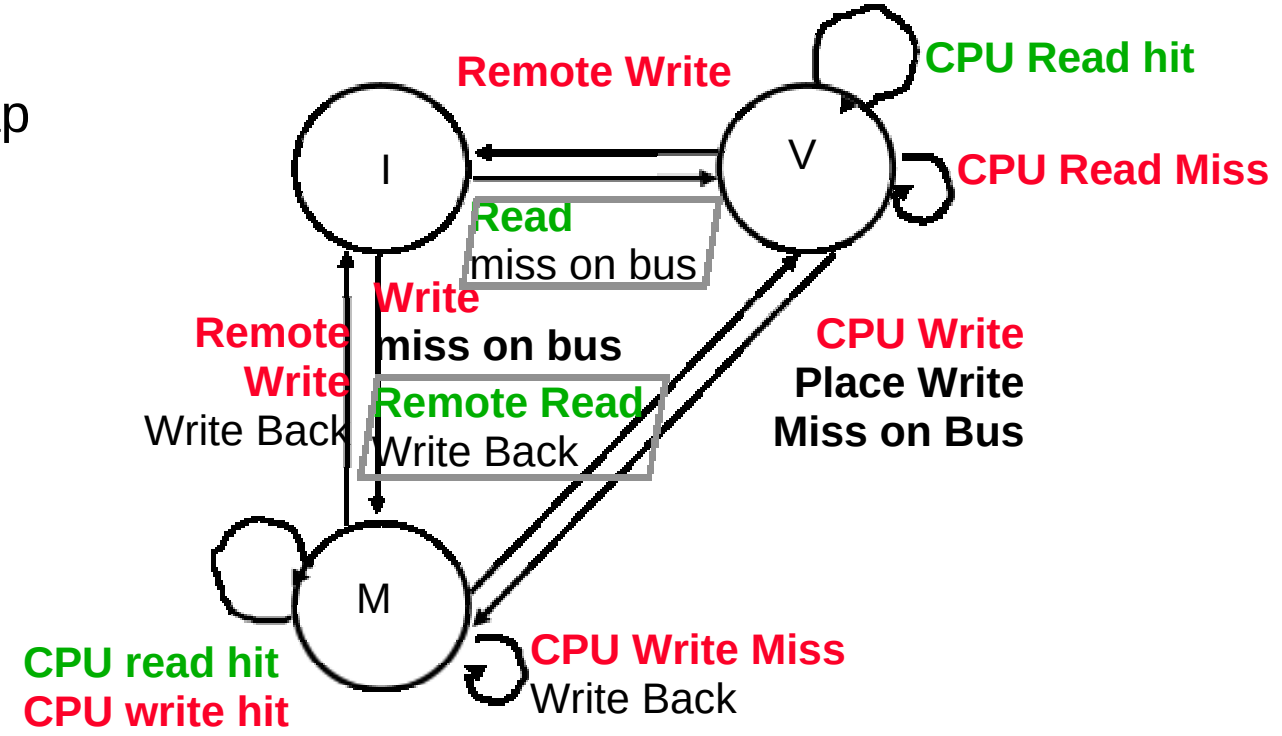
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



Example: Step 3

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

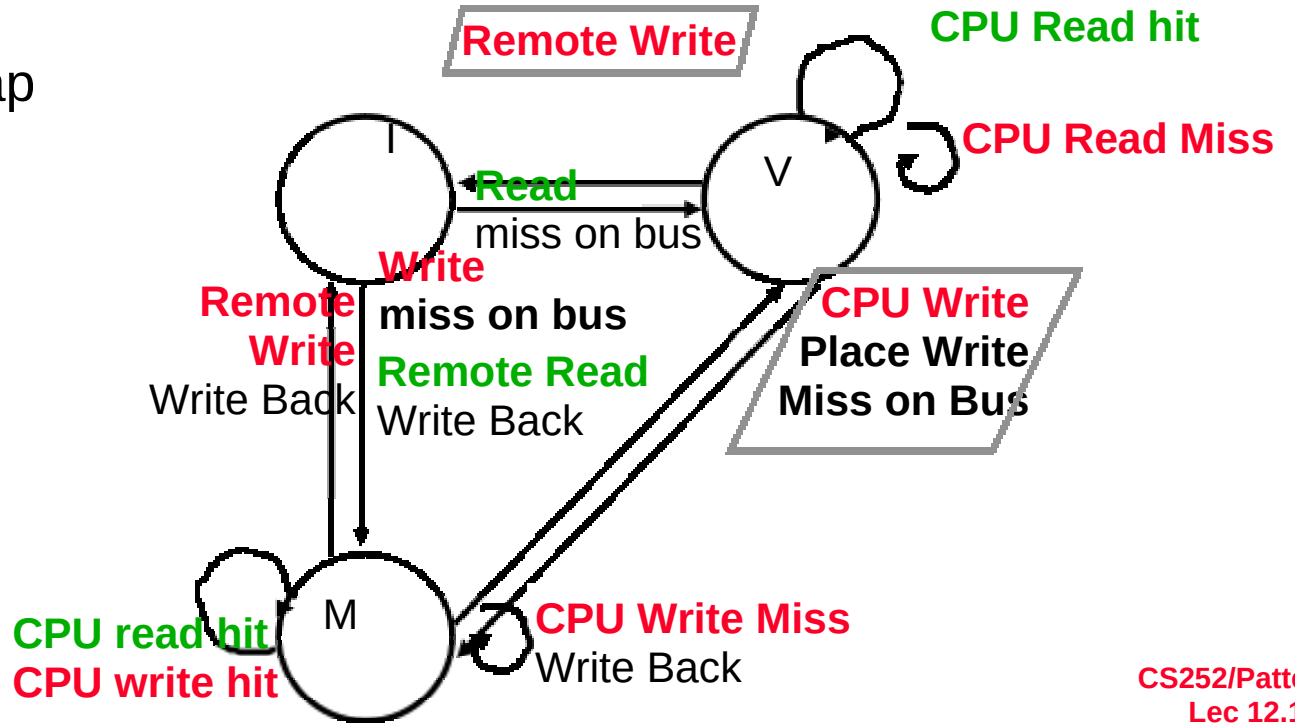
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.



Example: Step 4

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1			
P2: Write 40 to A2												
												10

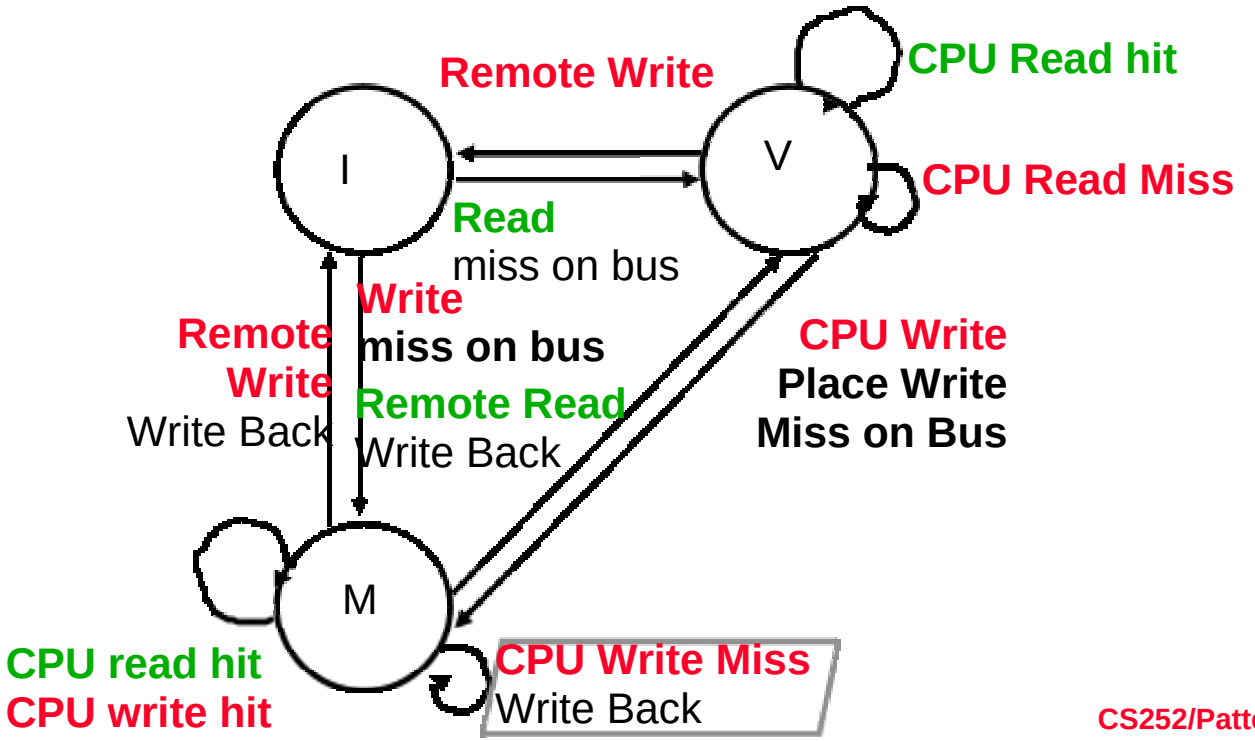
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



Example: Step 5

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20		20

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



Update or Invalidate?

- ***Update looks*** the ***simplest***, most obvious and ***fastest***, but:-
  - Multiple writes to the same word (no intervening read) need only one invalidate message but would require an update for each
  - Writes to same block in (usual) multi-word cache block require only one invalidate but would require multiple updates.

Update or Invalidate?

- Due to both spatial and temporal locality, previous cases occur often.
- Bus bandwidth is a precious commodity in shared memory multi-processors
- Experience has shown that invalidate protocols use significantly less bandwidth.

Other Cache Coherence Protocols

Various models and protocols have been devised for maintaining cache coherence, such as:

MSI Protocol

MESI Protocol *aka Illinois protocol*

MOSI Protocol

MOESI Protocol

MERSI Protocol

MESIF Protocol

Write-once Protocol

Firefly Protocol

Dragon Protocol

Some enhancements in Cache coherence protocol

- **“T” Enhancement of Cache Coherent Protocols.**
- **Using Prediction to Accelerate Coherence Protocols.**
- **Cache coherence protocol with SC-Cache for Multiprocessors.**
- **Snooping and Ordering Ring.**

References

- Effects of cache coherency in Multiprocessors, By Michel Dubois, Member-IEEE, and Faye A. Briggs, Member-IEEE (November 1982)
- Prof. M. Shaaban's EECC 756 Lecture notes on Cache Coherence Problem in Shared Memory Multiprocessor.
- Book: Parallel Computer Architecture (PCA) BY David E. Culler and Jaswinder P. Singh (1999 edition).
- http://parasol.tamu.edu/~rwerger/Courses/654/cachecoherence1.pdf
- CS252 Graduate Computer Architecture. A course by David A. Patterson in CS Department of UC Berkeley.
- Library.rit.edu
- Wikipedia.com
- Google.com

Thank You