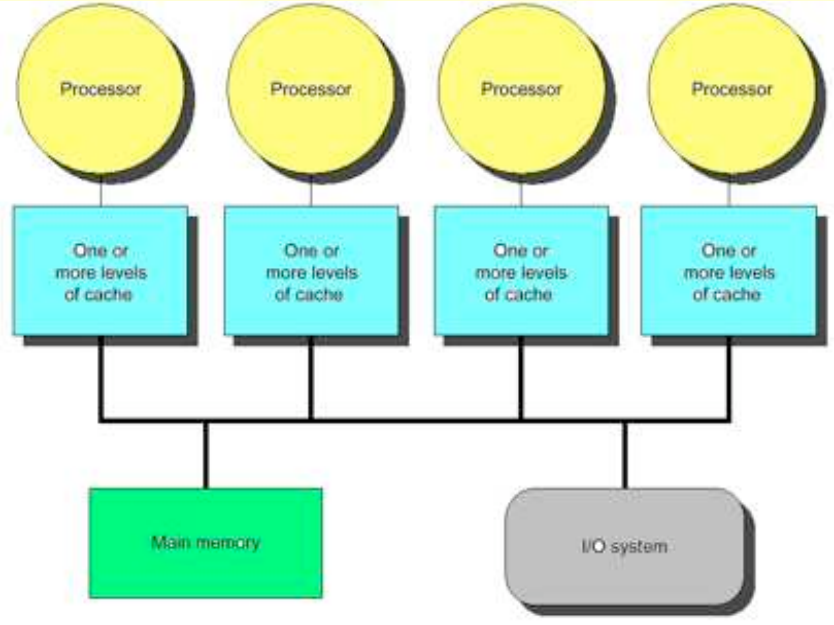
CMSC 611: Advanced Computer Architecture

Distributed & Shared Memory

Centralized Shared Memory MIMD

Processors share a single centralized memory through a bus interconnect

- Feasible for small processor count to limit memory contention
- Caches serve to:
  - Increase bandwidth versus bus/memory
  - Reduce latency of access
  - Valuable for both private data and shared data
  - Access to shared data is optimized by replication
    - Decreases latency
    - Increases memory bandwidth
    - Reduces contention
    - Replication introduces the problem of cache coherence



Cache Coherency

A cache coherence problem arises when the cache reflects a view of memory which is different from reality

Time	Event	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

A memory system is coherent if:

- P reads X, P writes X, no other processor writes X, P reads X
  - Always returns value written by P
- P reads X, Q writes X, P reads X
  - Returns value written by Q (provided sufficient W/R separation)
- P writes X, Q writes X
  - Seen in the same order by all processors

Potential HW Coherency Solutions

Snooping Solution (Snoopy Bus)

- 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-Based Schemes

- Keep track of what is being shared in one centralized place
- 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

* Slide is partially a courtesy of Dave Patterson

Basic Snooping Protocols

Write Invalidate Protocol:

- Write to shared data: an *invalidate* is sent to all caches which snoop and invalidate any copies
- Cache invalidation will force a cache miss when accessing the modified shared item
- For multiple writers only one will win the race ensuring serialization of the write operations
- Read Miss:
  - Write-through: memory is always up-to-date
  - Write-back: snoop in caches to find most recent copy

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

* Slide is partially a courtesy of Dave Patterson

Basic Snooping Protocols

Write Broadcast (Update) Protocol (typically write through):

- Write to shared data: broadcast on bus, processors snoop, and update any copies
- To limit impact on bandwidth, track data sharing to avoid unnecessary broadcast of written data that is not shared
- Read miss: memory is always up-to-date
- Write serialization: bus serializes requests!

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Write broadcast of X	1	1	1
CPU B reads X		1	1	1

* Slide is partially a courtesy of Dave Patterson

Invalidate vs. Update

Write-invalidate has emerged as the winner for the vast majority of designs

Qualitative Performance Differences :

- Spatial locality
  - WI: 1 transaction/cache block;
  - WU: 1 broadcast/word
- Latency
  - WU: lower write–read latency
  - WI: must reload new value to cache

Invalidate vs. Update

Because the bus and memory bandwidth is usually in demand, write-invalidate protocols are very popular

Write-update can causes problems for some memory consistency models, reducing the potential performance gain it could bring

The high demand for bandwidth in write-update limits its scalability for large number of processors

An Example Snoopy Protocol

Invalidation protocol, write-back cache

Each block of memory is in one state:

- Clean in all caches and up-to-date in memory (Shared)
- OR Dirty in exactly one cache (Exclusive)
- OR Not in any caches

Each cache block is in one state (track these):

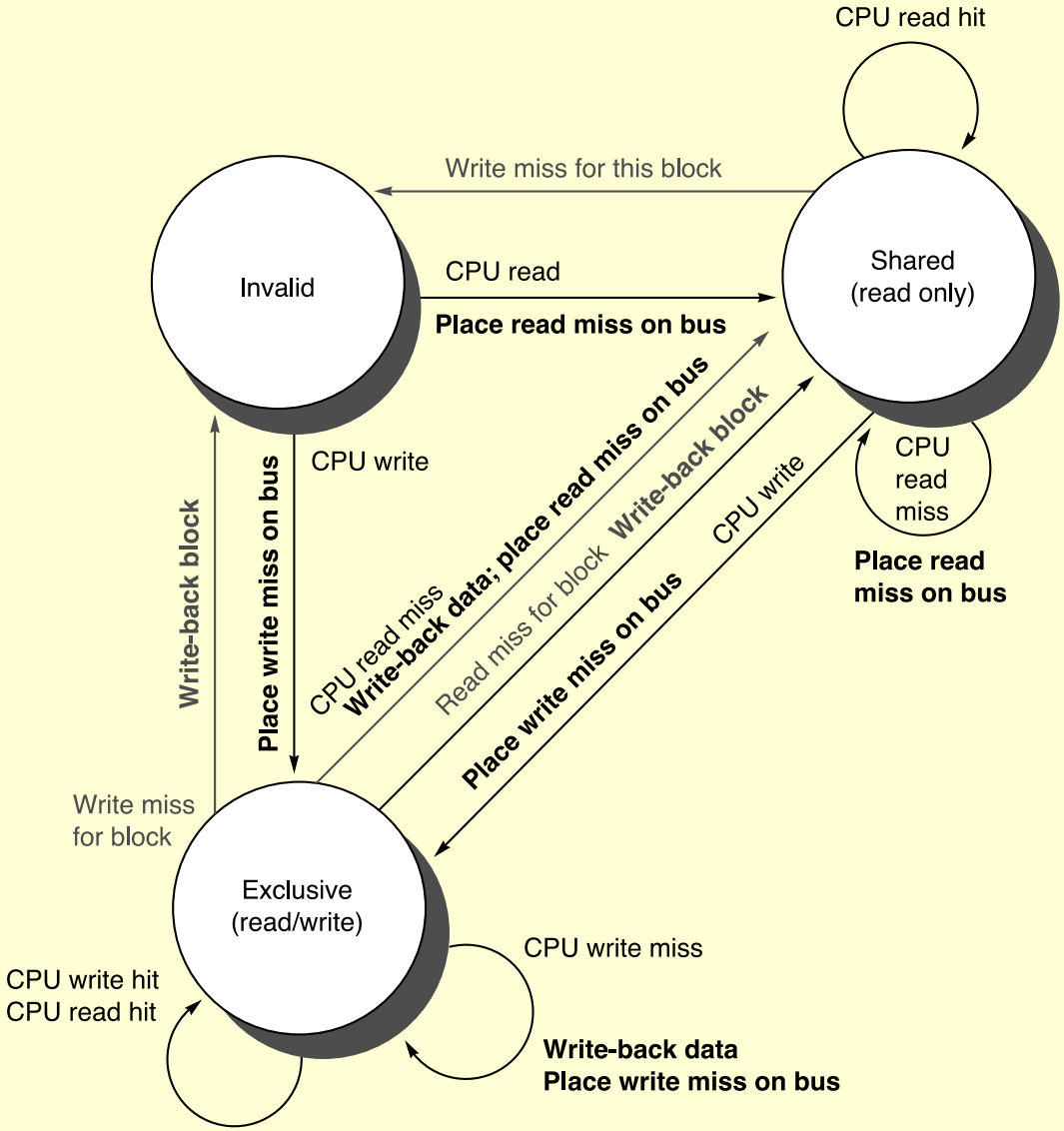
- Shared : block can be read
- OR Exclusive : cache has only copy, it is write-able, and dirty
- OR Invalid : block contains no data

Read misses: cause all caches to snoop bus

Writes to clean line are treated as misses

* Slide is partially a courtesy of Dave Patterson

Snoopy-Cache Controller



Complications

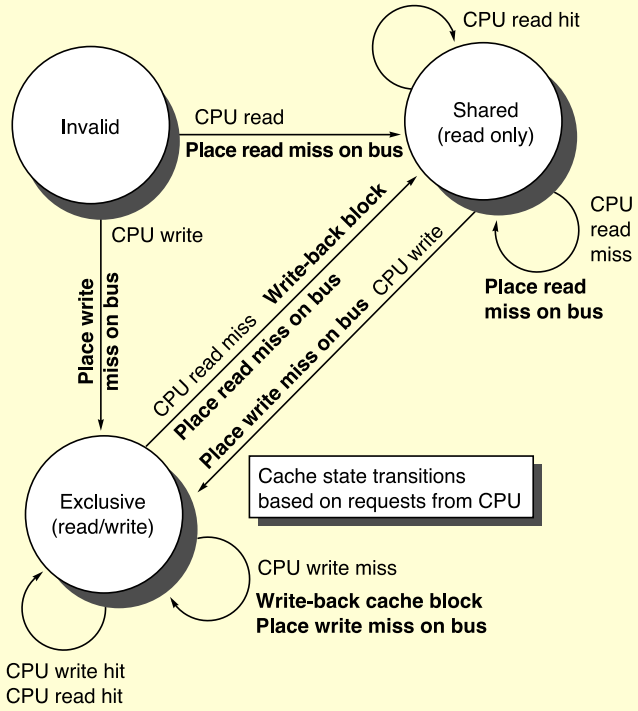
- Cannot update cache until bus is obtained
- Two step process:
  - Arbitrate for bus
  - Place miss on bus and complete operation
- Split transaction bus:
  - Bus transaction is not atomic
  - Multiple misses can interleave, allowing two caches to grab block in the Exclusive state
  - Must track and prevent multiple misses for one block

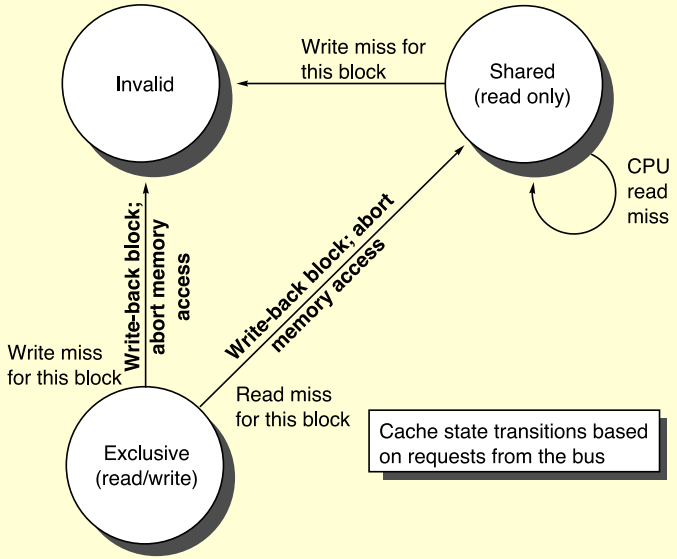
* Slide is a courtesy of Dave Patterson

Example

	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 memory blocks A1 and A2 map to same cache block, initial cache state is invalid



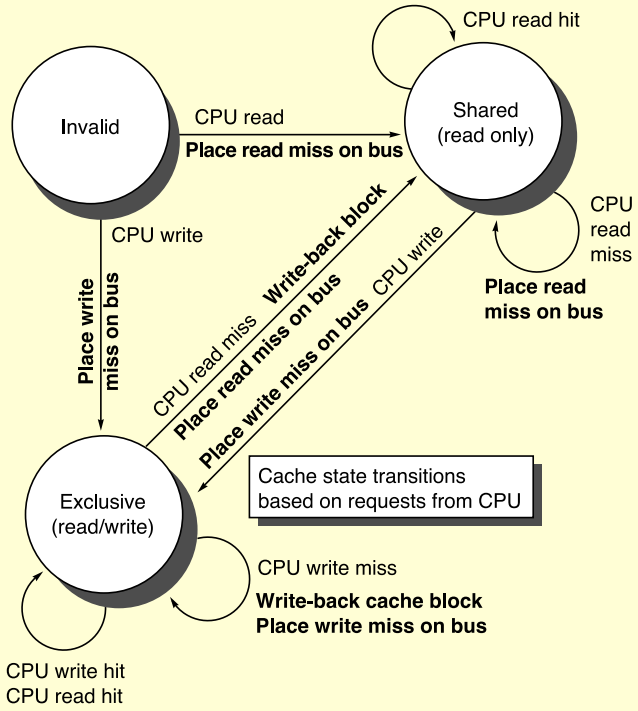


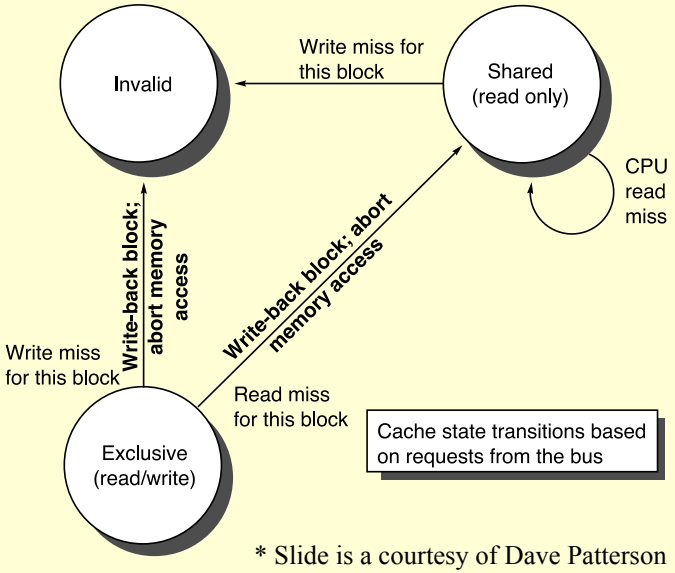
* Slide is a courtesy of Dave Patterson

Example

	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 memory blocks A1 and A2 map to same cache block



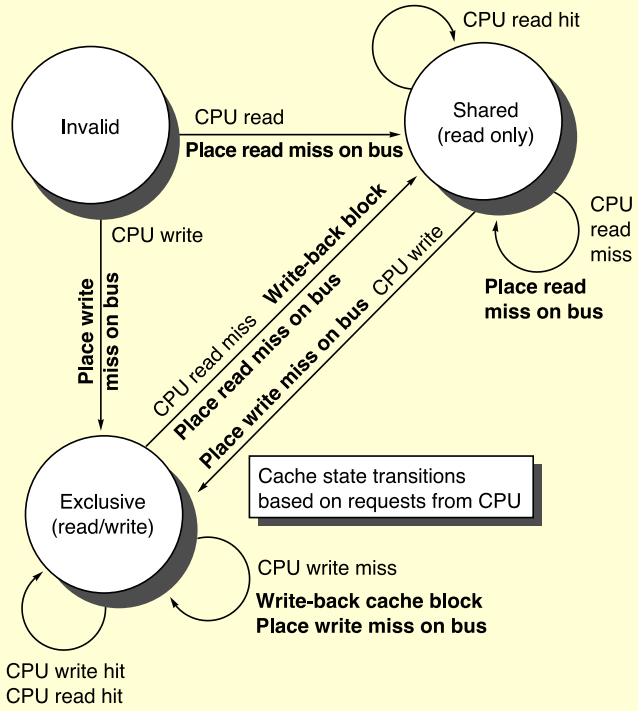


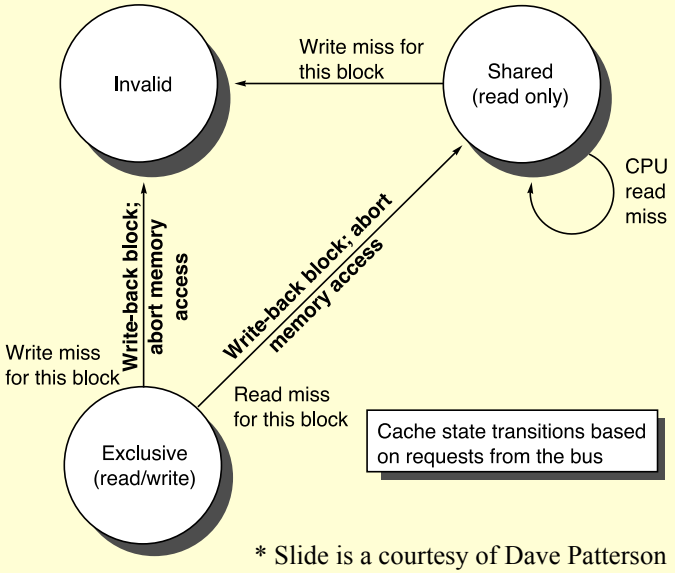
* Slide is a courtesy of Dave Patterson

Example

	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 memory blocks A1 and A2 map to same cache block



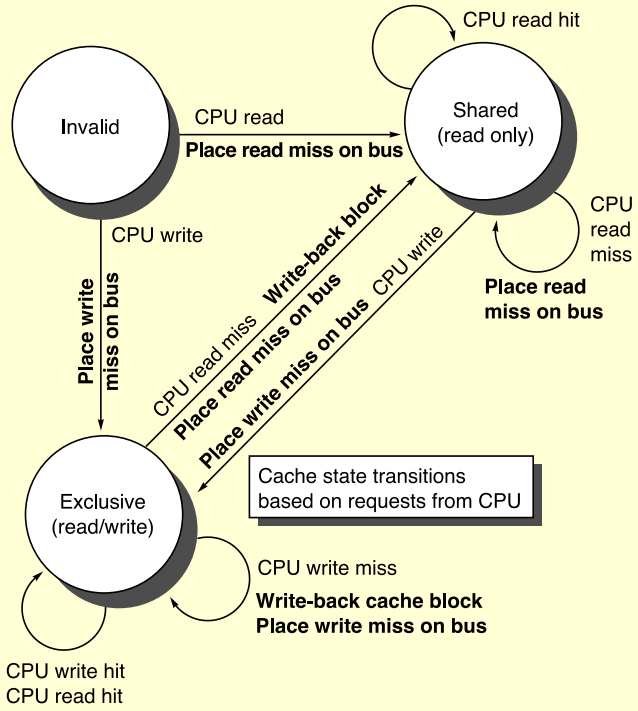


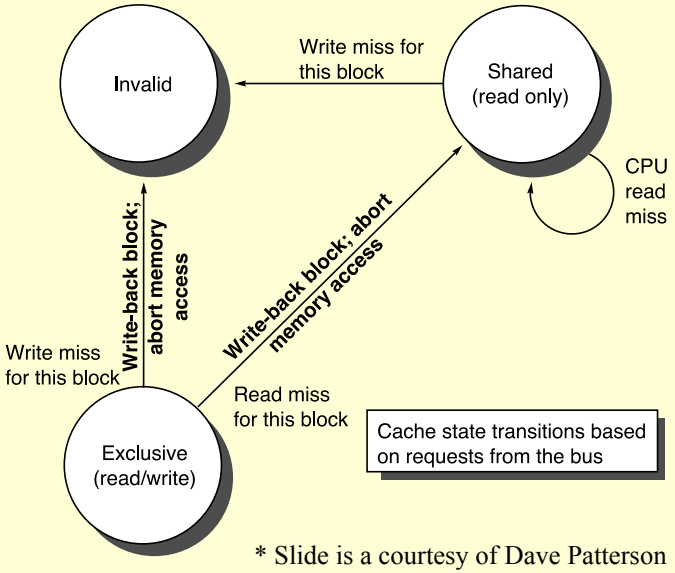
* Slide is a courtesy of Dave Patterson

Example

	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		10
				Shar.	A1	10	RdDa	P2	A1	10		10
P2: Write 20 to A1												10
P2: Write 40 to A2												10
												10

Assumes memory blocks A1 and A2 map to same cache block



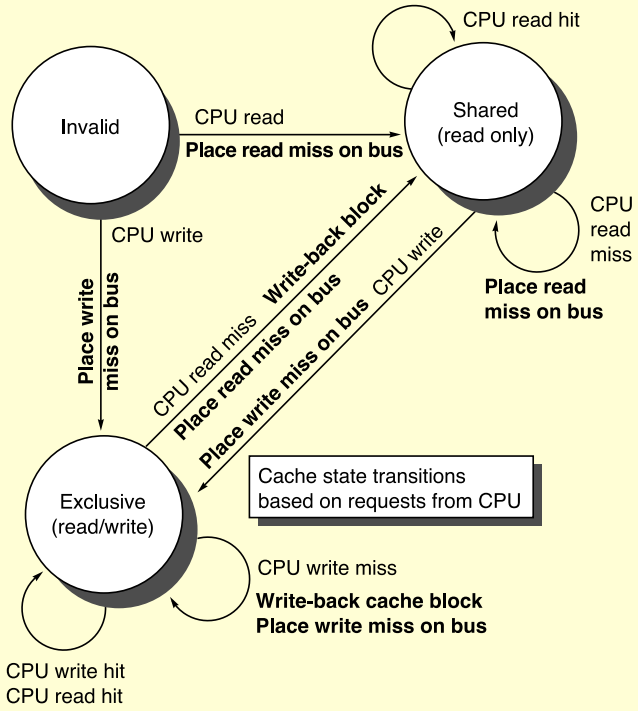


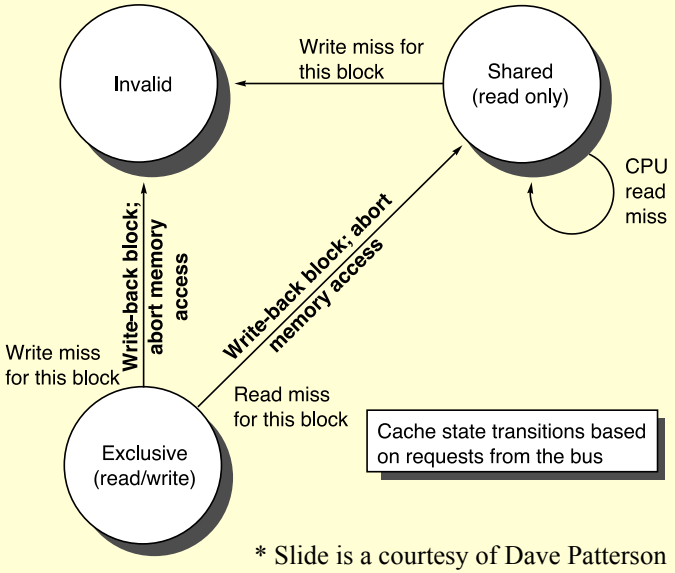
* Slide is a courtesy of Dave Patterson

Example

	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		10
				Shar.	A1	10	RdDa	P2	A1	10		10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1			10
P2: Write 40 to A2												10
												10

Assumes memory blocks A1 and A2 map to same cache block



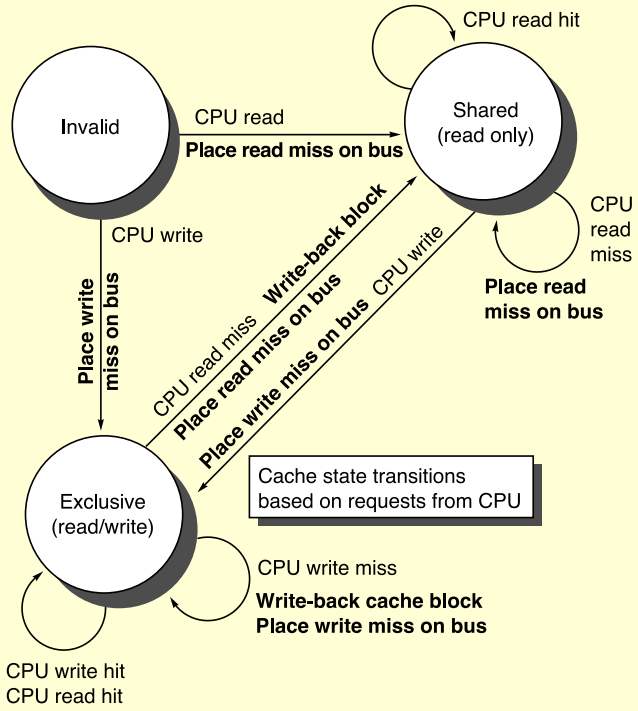


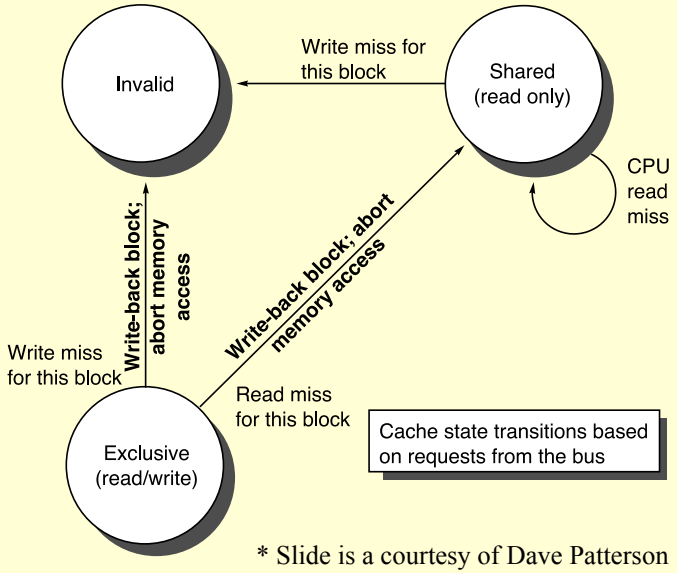
* Slide is a courtesy of Dave Patterson

Example

	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		10
				Shar.	A1	10	RdDa	P2	A1	10	*A1*	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1			10
P2: Write 40 to A2							WrMs	P2	A2			10
				Excl.	A2	40	WrBk	P2	A1	20	*A1*	20

Assumes memory blocks A1 and A2 map to same cache block





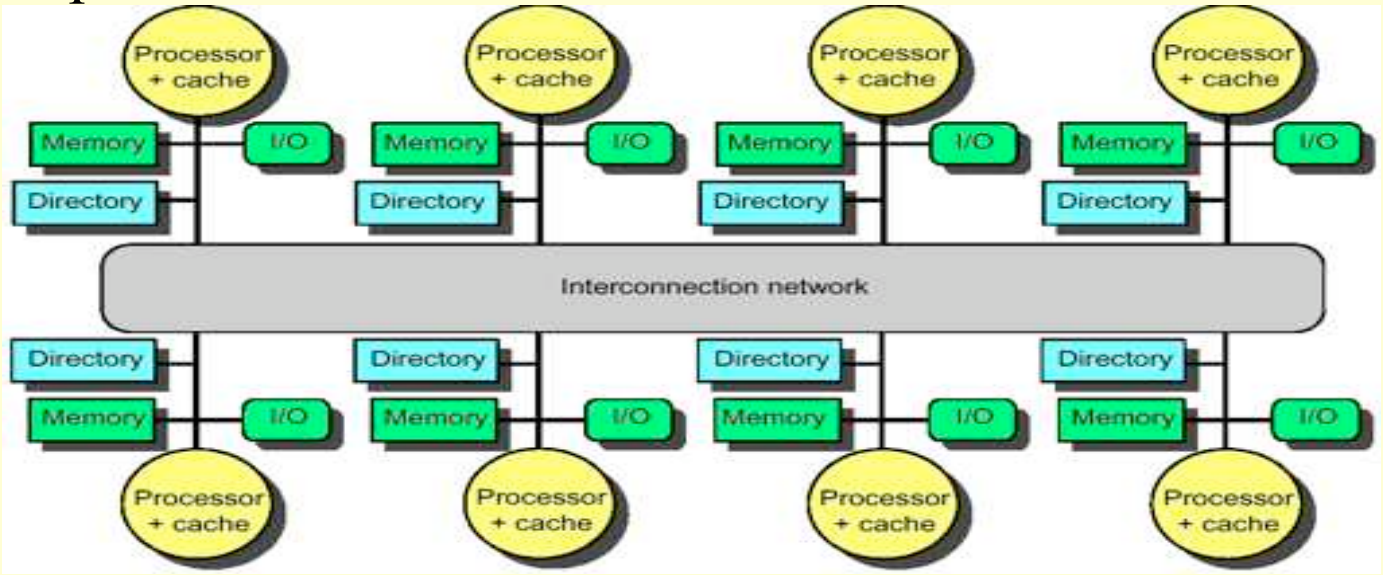
* Slide is a courtesy of Dave Patterson

Distributed Directory Multiprocessors

Directory per cache that tracks state of every block in every cache

- Which caches have a copy of block, dirty vs. clean, ...
- Info per memory block vs. per cache block?
  - PLUS: In memory => simpler protocol (centralized/one location)
  - MINUS: In memory => directory is *f*(memory size) vs. *f*(cache size)

To prevent directory from being a bottleneck, distribute directory entries with memory, each keeping track of which processor have copies of their blocks



* Slide is partially a courtesy of Dave Patterson

Directory Protocol

Similar to Snoopy Protocol: Three states

- Shared: Multiple processors have the block cached and the contents of the block in memory (as well as all caches) is up-to-date
- Uncached No processor has a copy of the block (not valid in any cache)
- Exclusive: Only one processor (owner) has the block cached and the contents of the block in memory is out-to-date (the block is dirty)

In addition to cache state, must track which processors have data when in the shared state

- usually bit vector, 1 if processor has copy

* Slide is a courtesy of Dave Patterson

Directory Protocol

Keep it simple(r):

- Writes to non-exclusive data => write miss
- Processor blocks until access completes
- Assume messages received and acted upon in order sent

Terms: typically 3 processors involved

- Local node where a request originates
- Home node where the memory location of an address resides
- Remote node has a copy of a cache block, whether exclusive or shared

No bus and do not want to broadcast:

- interconnect no longer single arbitration point
- all messages have explicit responses

Example Directory Protocol

Message sent to directory causes two actions:

- Update the directory
- More messages to satisfy request

We assume operations atomic, but they are not; reality is much harder; must avoid deadlock when run out of buffers in network

* Slide is a courtesy of Dave Patterson

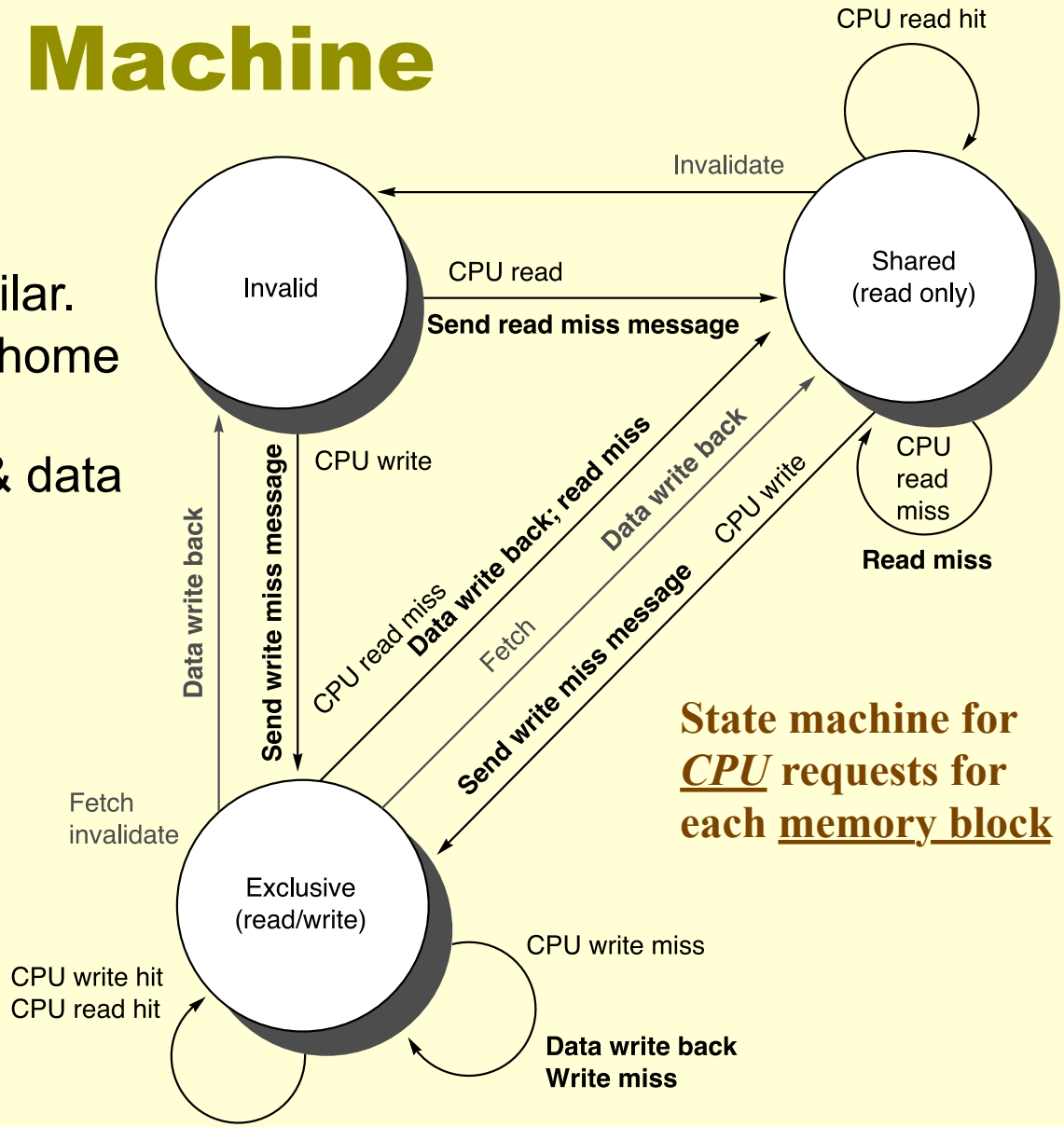
Directory Protocol Messages

Type	SRC	DEST	MSG
Read miss	local cache	home directory	P,A
P has read miss at A; request data and make P a read sharer			
Write miss	local cache	home directory	P,A
P has write miss at A; request data and make P exclusive owner			
Invalidate	home directory	remote cache	A
Invalidate shared data at A			
Fetch	home directory	remote cache	A
Fetch block A home; change A remote state to shared			
Fetch/invalidate	home directory	remote cache	A
Fetch block A home; invalidate remote copy			
Data value reply	home directory	local cache	D
Return data value from home memory			
Data write back	remote cache	home directory	A,D
Write back data value for A			

Cache Controller State Machine

States identical to snoopy case

- Transactions very similar.
  - Miss messages to home directory
  - Explicit invalidate & data fetch requests



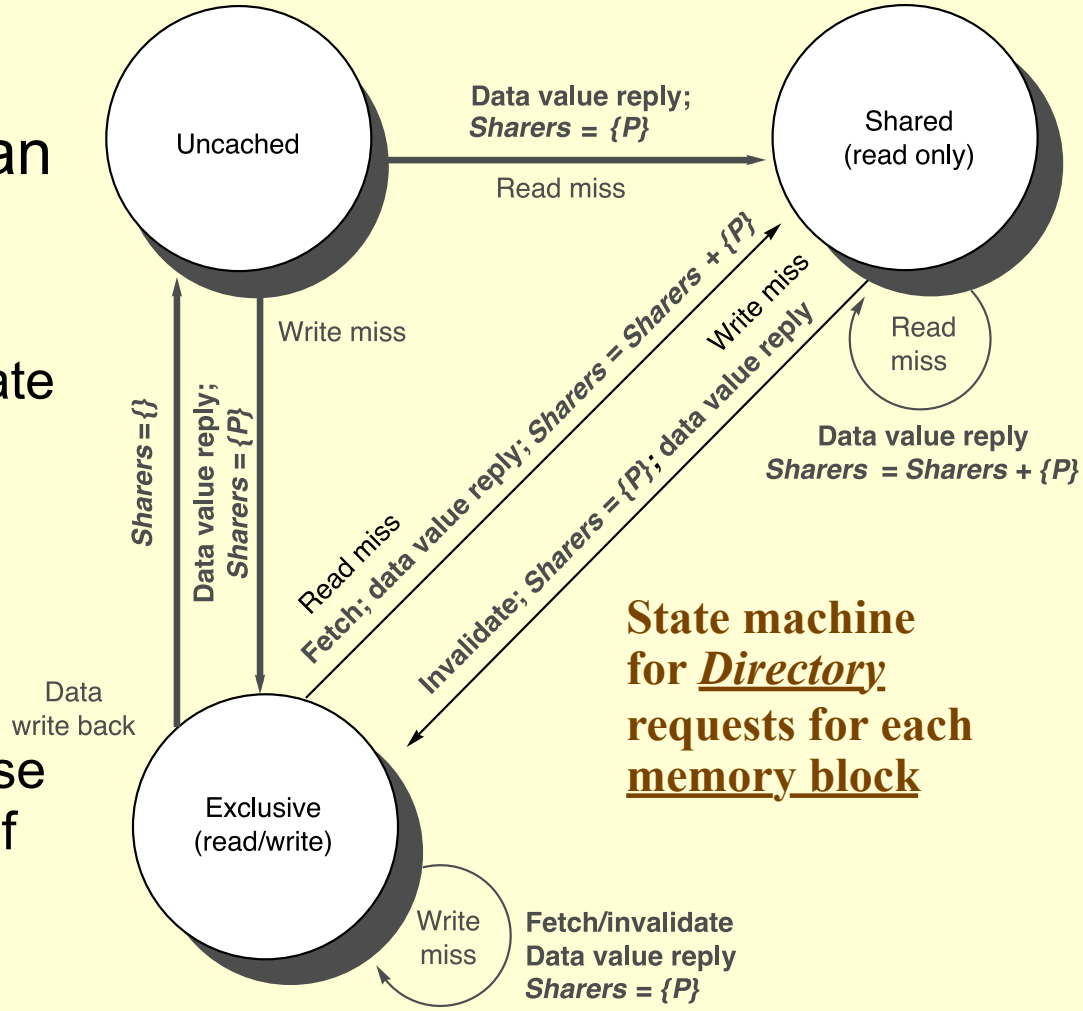
State machine for *CPU* requests for each memory block

* Slide is partially a courtesy of Dave Patterson

Directory Controller State Machine

Same states and structure as the transition diagram for an individual cache

- Actions:
  - update of directory state
  - send messages to satisfy requests
- Tracks all copies of each memory block
  - Sharers set implementation can use a bit vector of a size of # processors for each block



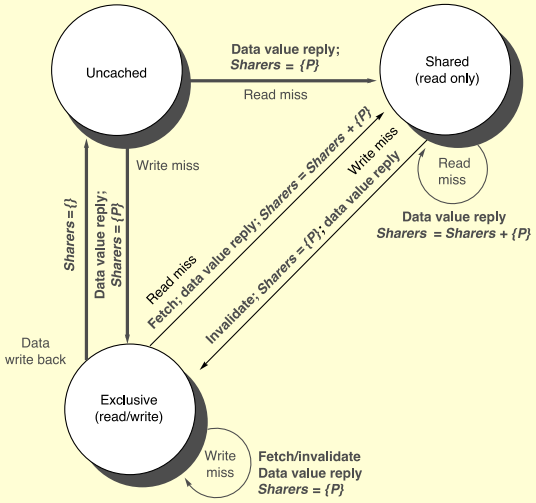
State machine for *Directory* requests for each memory block

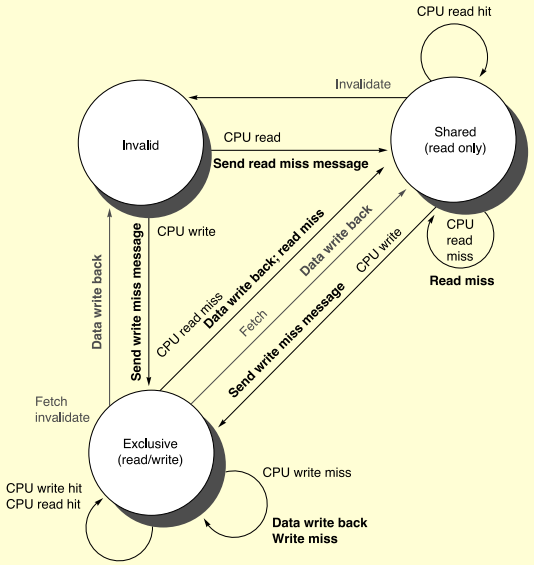
* Slide is partially a courtesy of Dave Patterson

Example

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

Assumes memory blocks A1 and A2 map to same cache block



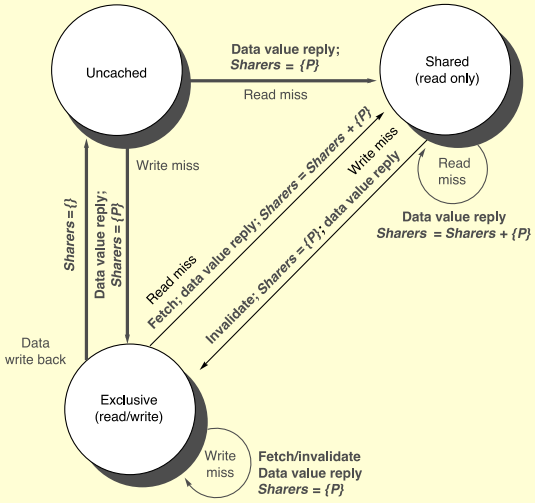


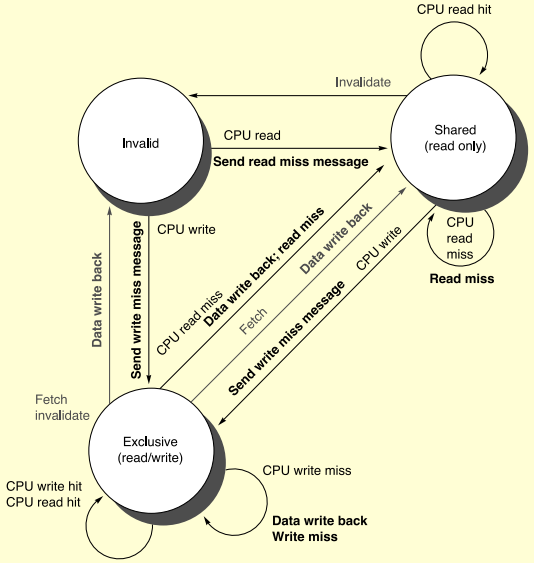
* Slide is a courtesy of Dave Patterson

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1														
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

Assumes memory blocks A1 and A2 map to same cache block



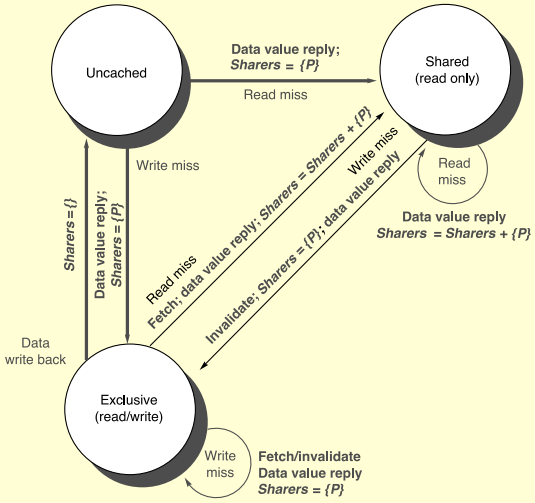


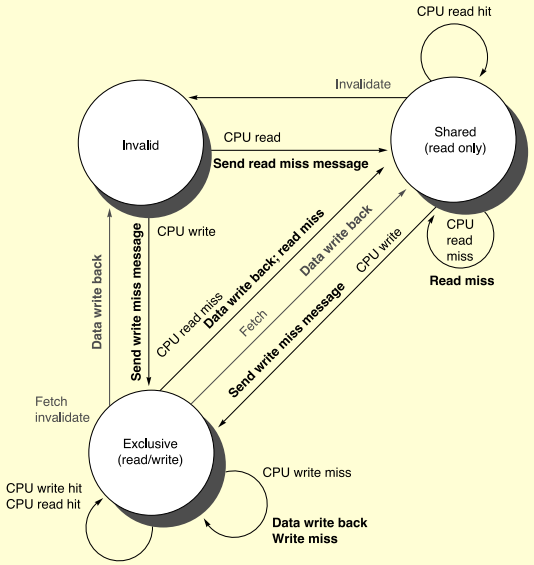
* Slide is a courtesy of Dave Patterson

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

Assumes memory blocks A1 and A2 map to same cache block





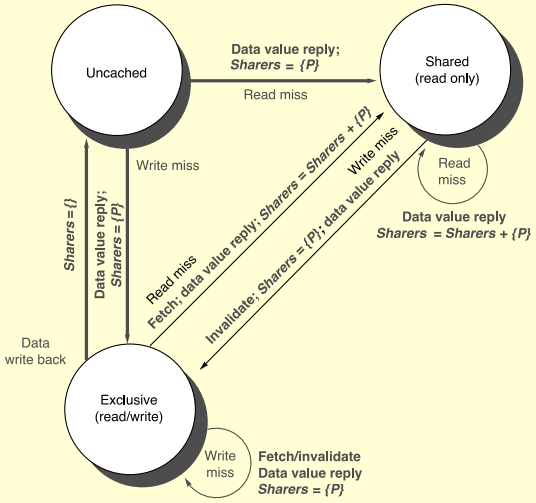
* Slide is a courtesy of Dave Patterson

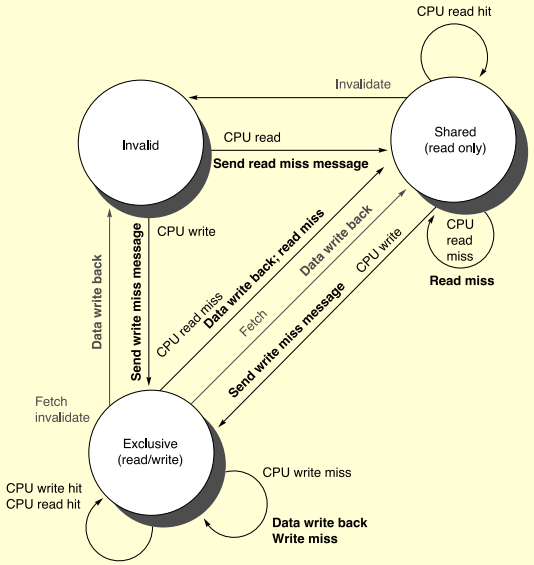
Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				*Shar.*	*A1*		*RdMs*	P2	A1					
	Shar.	A1	10				*Ftch*	P1	A1	10			*A1*	*10*
				Shar.	A1	*10*	*DaRp*	P2	A1	10	A1	*Shar.*	*{P1,P2}*	10
P2: Write 20 to A1														
P2: Write 40 to A2														

Write Back

Assumes memory blocks A1 and A2 map to same cache block



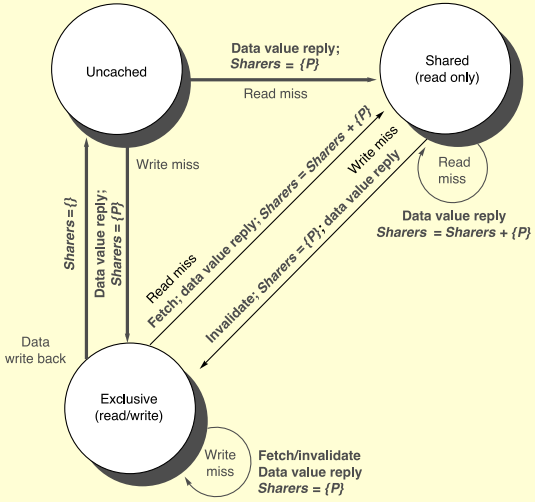


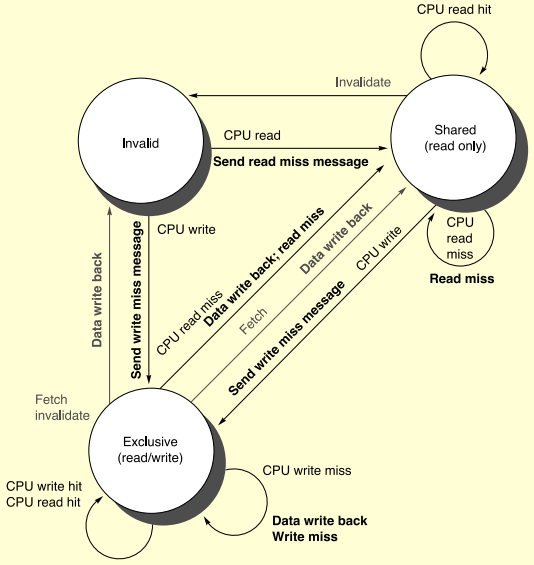
* Slide is a courtesy of Dave Patterson

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				*Shar.*	*A1*		*RdMs*	P2	A1					
	Shar.	A1	10				*Ftch*	P1	A1	10			*A1*	*10*
				Shar.	A1	*10*	*DaRp*	P2	A1	10	A1	*Shar.*	*{P1,P2}*	10
P2: Write 20 to A1				Excl.	A1	*20*	*WrMs*	P2	A1					10
	Inv.						*Inval.*	P1	A1		A1	*Excl.*	*{P2}*	10
P2: Write 40 to A2														

Assumes memory blocks A1 and A2 map to same cache block



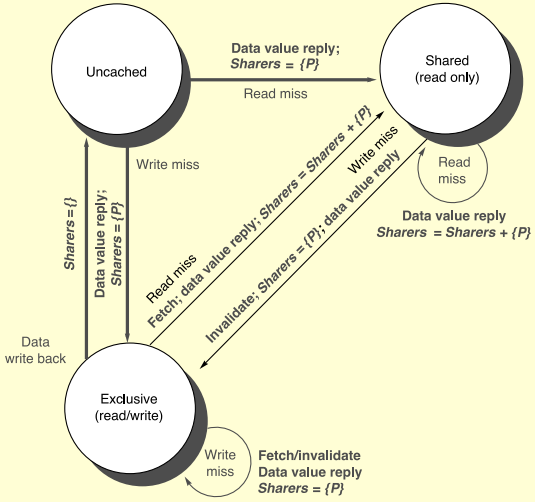


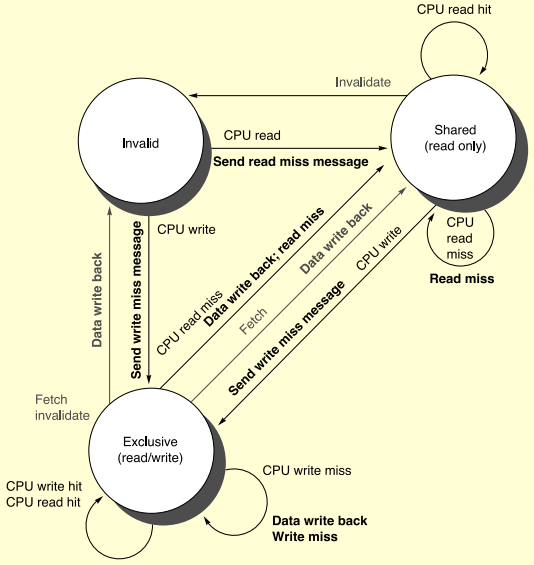
* Slide is a courtesy of Dave Patterson

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1				*Shar.*	*A1*		*RdMs*	P2	A1					
	Shar.	A1	10				*Ftch*	P1	A1	10			*A1*	*10*
				Shar.	A1	*10*	*DaRp*	P2	A1	10	A1	*Shar.*	*{P1,P2}*	10
P2: Write 20 to A1				Excl.	A1	*20*	*WrMs*	P2	A1					10
	Inv.						*Inval.*	P1	A1		A1	*Excl.*	*{P2}*	10
P2: Write 40 to A2							*WrMs*	P2	A2		*A2*	*Excl.*	*{P2}*	0
							WrBk	P2	A1	20	*A1*	*Unca.*	*{}*	*20*
				Excl.	*A2*	*40*	*DaRp*	P2	A2	0	A2	Excl.	{P2}	0

Assumes memory blocks A1 and A2 map to same cache block





* Slide is a courtesy of Dave Patterson