Multiprocessors Coherence

Review: Small-Scale—Shared Memory

- Caches serve to:
  - Increase bandwidth versus bus/memory
  - Reduce latency of access
  - Valuable for both private data and shared data
- What about cache consistency?

Time	Event	$A	$B	X (memory)
0				1
1	CPU A: R x	1		1
2	CPU B: R x	1	1	1
3	CPU A: W x,0	0	1	0

What Does Coherency Mean?

- Informally:
  - "Any read of a data item must return the most recently written value"
  - this definition includes both coherence and consistency
    - coherence: what values can be returned by a read
    - consistency: when a written value will be returned by a read
- Memory system is coherent if
  - a read(X) by P1 that follows a write(X) by P1, with no writes of X by another processor occurring between these two events, always returns the value written by P1
  - a read(X) by P1 that follows a write(X) by another processor, returns the written value if the read and write are sufficiently separated and no other writes occur between
  - writes to the same location are serialized: two writes to the same location by any two CPUs are seen in the same order by all CPUs

Potential HW Coherence Solutions

- Snooping Solution (Snoopy Bus):
  - every cache that has a copy of the data also has a copy of the sharing status of the block
  - 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 (discuss later)
  - 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

Basic Snoopy Protocols

- Write Invalidate Protocol
  - A CPU has exclusive access to a data item before it writes that item
  - Write to shared data: an invalidate is sent to all caches which snoop and invalidate any copies
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy
- Write Update Protocol (typically write through):
  - Write to shared data: broadcast on bus, processors snoop, and update any copies
  - Read miss: memory is always up-to-date
- Write serialization: bus serializes requests!
  - Bus is single point of arbitration

Write Invalidate versus Update

- Multiple writes to the same word with no intervening reads
  - Update: multiple broadcasts
- For multiword cache blocks
  - Update: each word written in a cache block requires a write broadcast
  - Invalidate: only the first write to any word in the block requires an invalidation
- Update has lower latency between write and read

Snooping Cache Variations

Basic Protocol	Berkeley Protocol	Illinois Protocol	MESI Protocol
Exclusive	Owned Exclusive	Private Dirty	Modfied (private,!=Memory)
Shared	Owned Shared	Private Clean	eXclusive (private,=Memory)
Invalid	Shared	Shared	Shared (shared,=Memory)
	Invalid	Invalid	Invalid

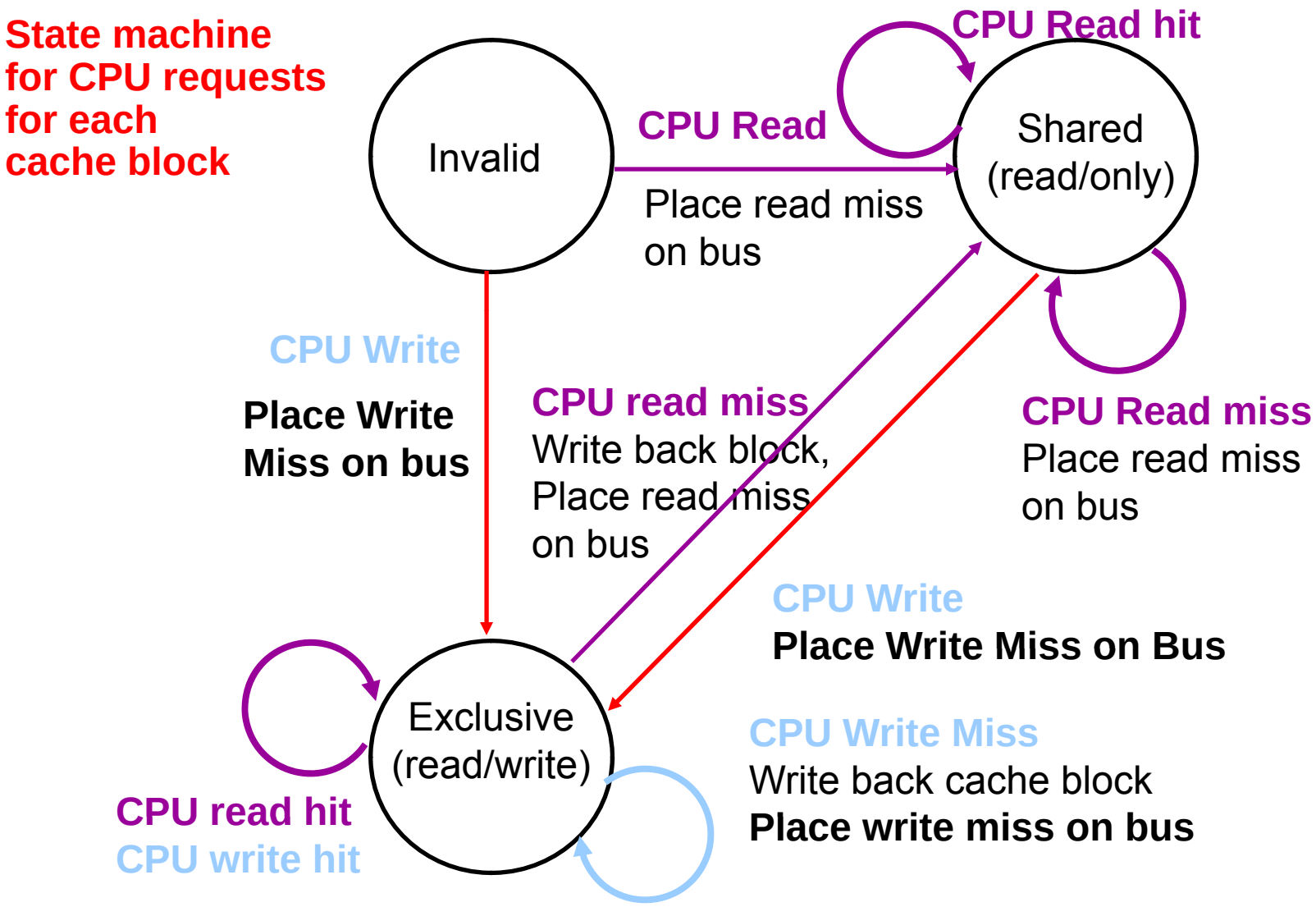
Owner can update via bus invalidate operation
Owner must write back when replaced in cache

If read sourced from memory, then Private Clean
if read sourced from other cache, then Shared
Can write in cache if held private clean or dirty

An Example Snoopy Protocol

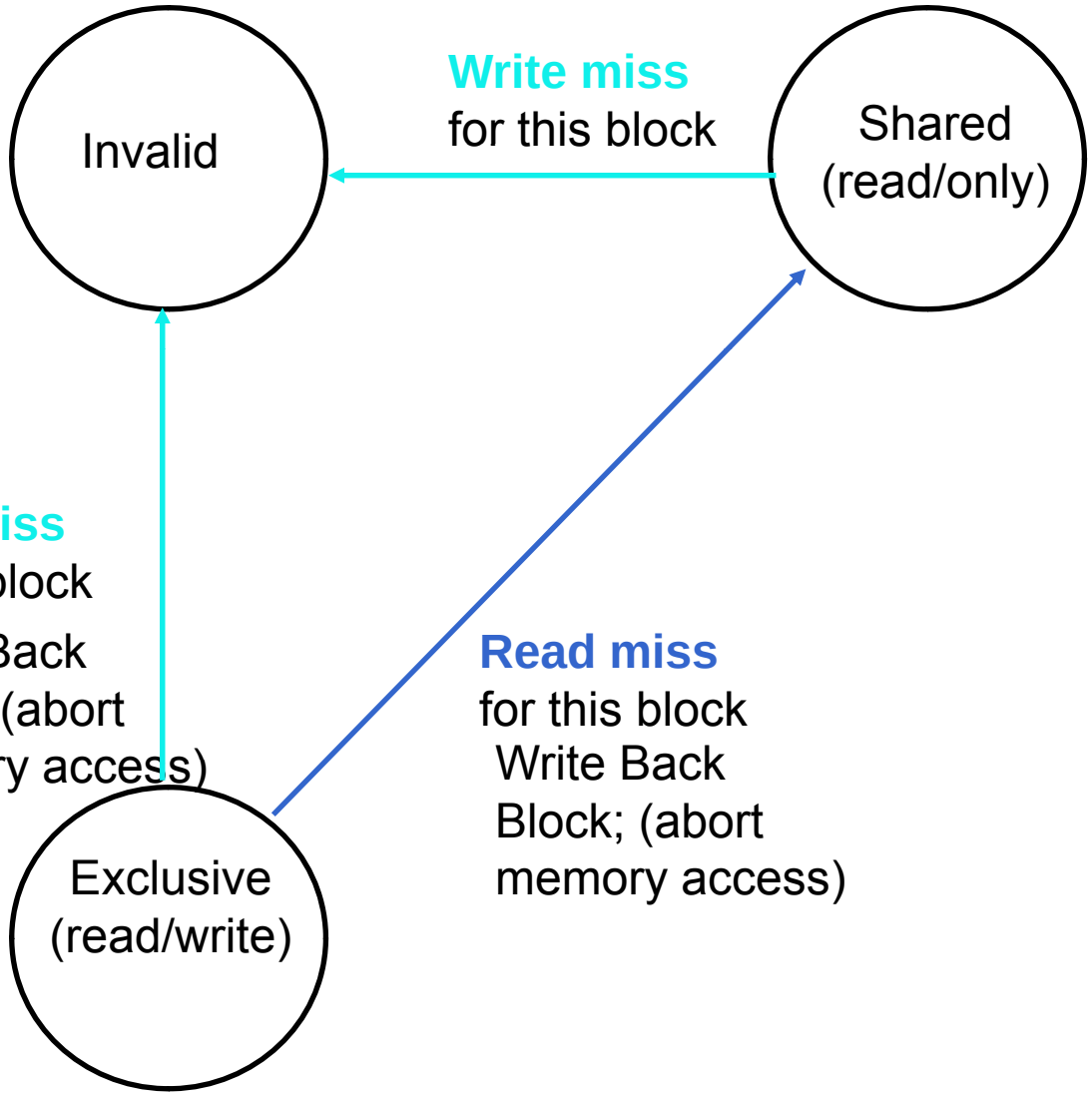
- Invalidation protocol, write-back cache
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Exclusive)
  - OR Not in any caches
- Each cache block is in one state (track these):
  - Shared : block can be read
  - OR Exclusive : cache has only copy,
    its writeable, and dirty
  - OR Invalid : block contains no data
- Read misses: cause all caches to snoop bus
- Writes to clean line are treated as misses

Snoopy-Cache State Machine-I

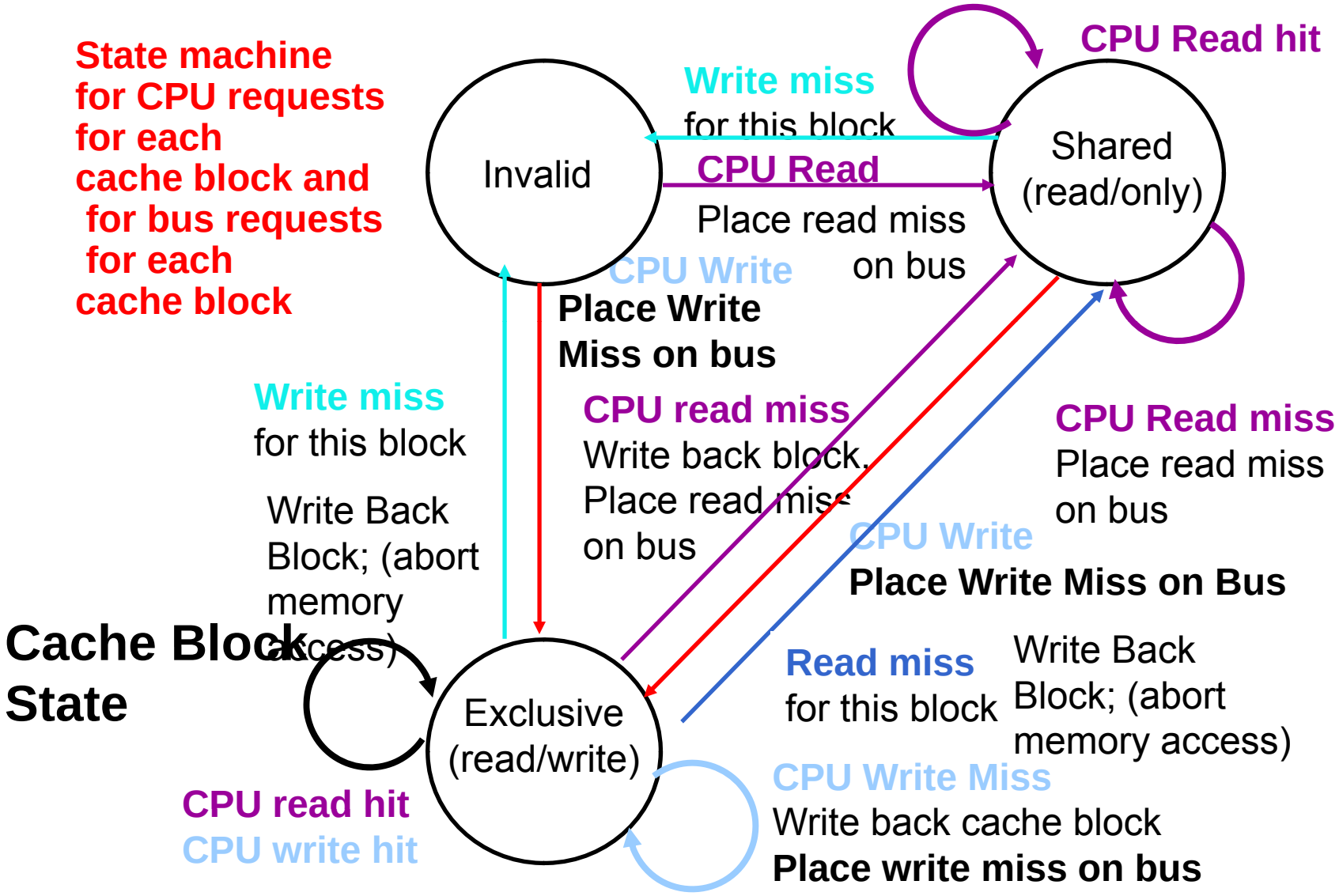


Snoopy-Cache State Machine-II

State machine for bus requests for each cache block



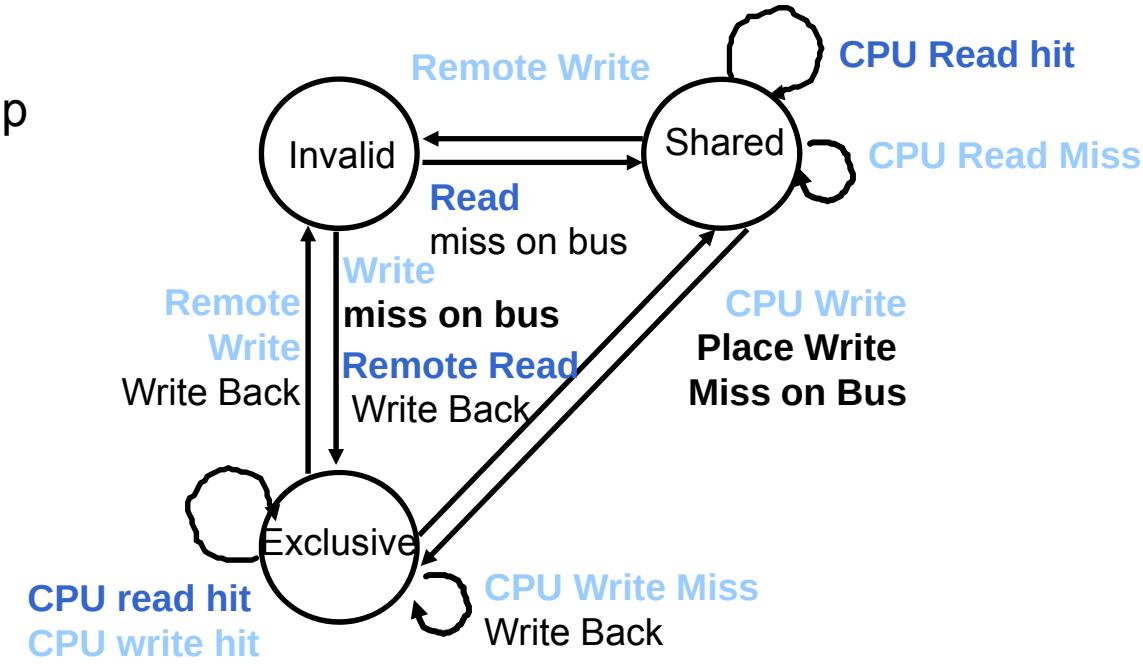
Snoopy-Cache State Machine-III



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

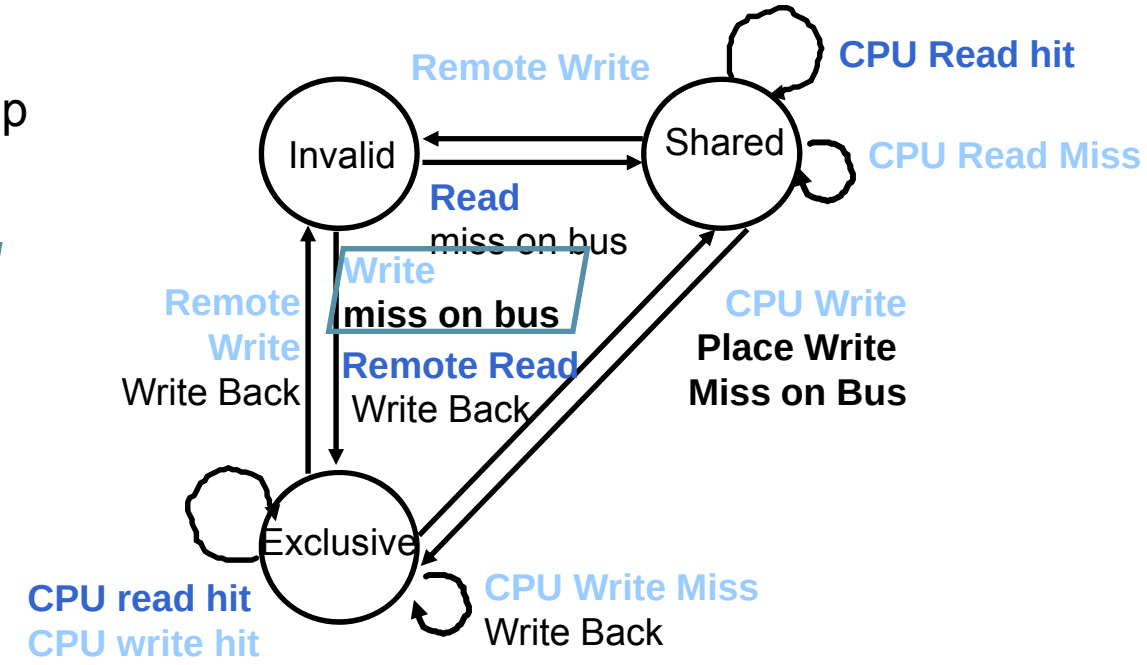


Example: Step 1

	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.

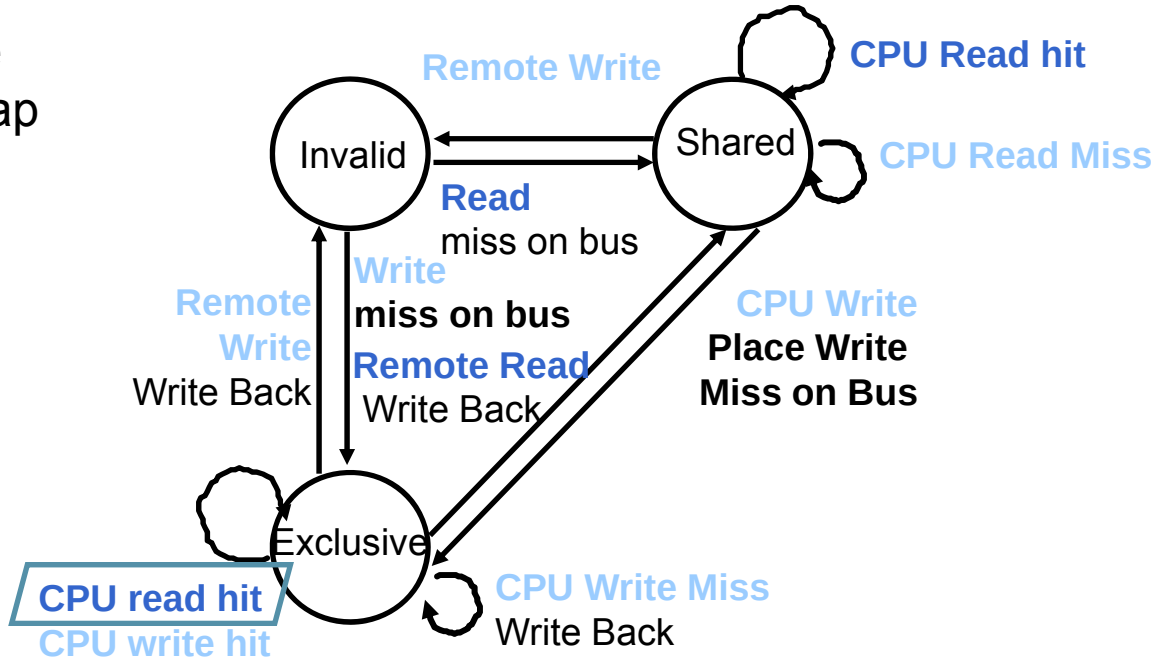
Active arrow =



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												

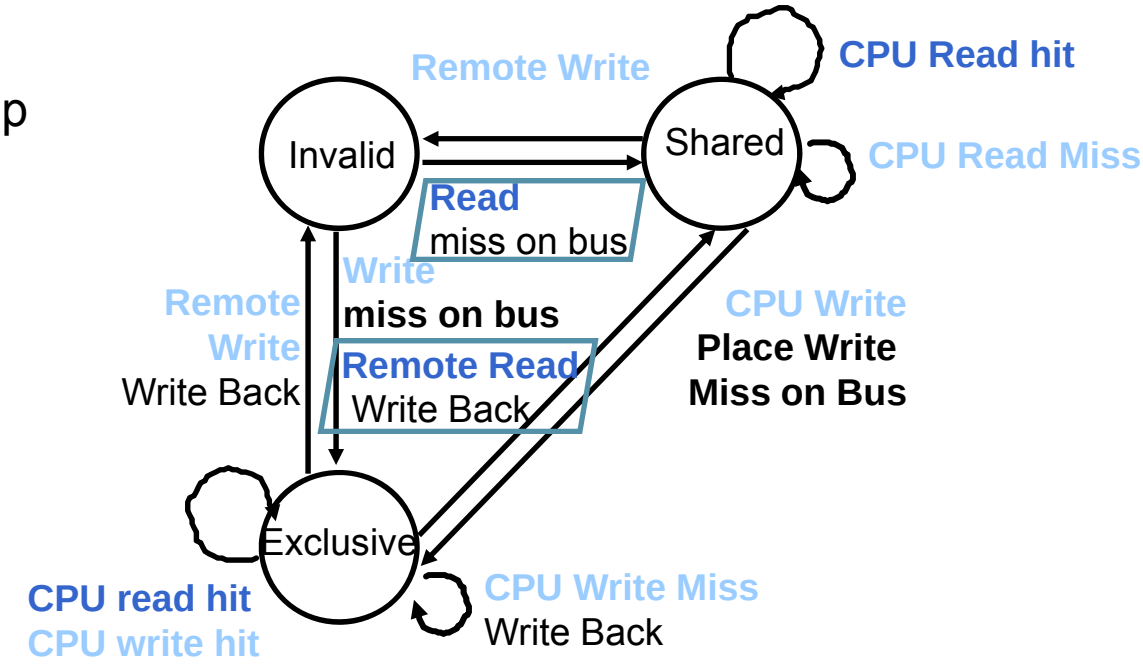
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



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												

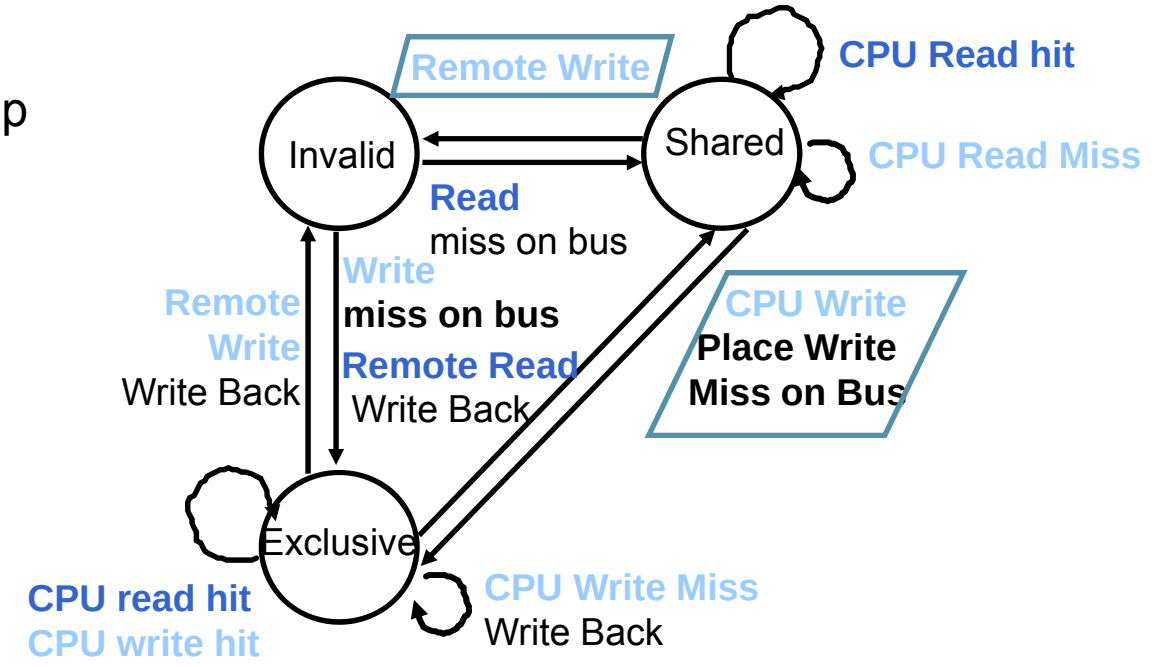
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.



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			
	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												

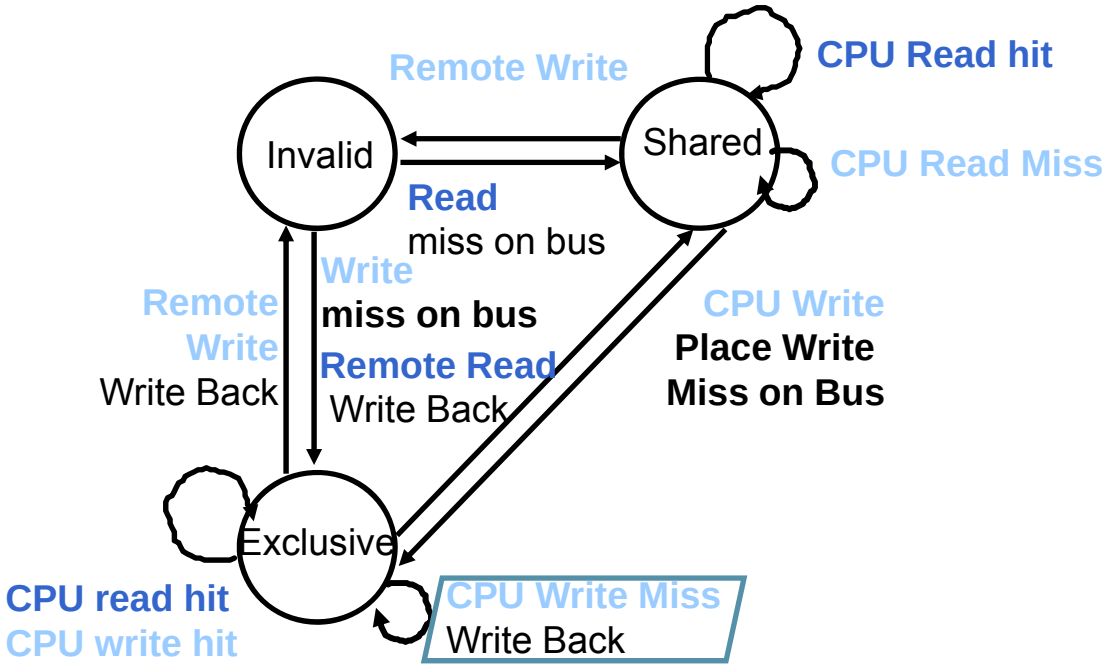
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



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			
	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	A1	20

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



Implementation Complications

- Write Races:
  - Cannot update cache until bus is obtained
    - Otherwise, another processor may get bus first, and then write the same cache block!
  - Two step process:
    - Arbitrate for bus
    - Place miss on bus and complete operation
  - If miss occurs to block while waiting for bus, handle miss (invalidate may be needed) and then restart
  - Split transaction bus:
    - Bus transaction is not atomic: can have multiple outstanding transactions for a block
    - Multiple misses can interleave, allowing two caches to grab block in the Exclusive state
    - Must track and prevent multiple misses for one block
- Must support interventions and invalidations

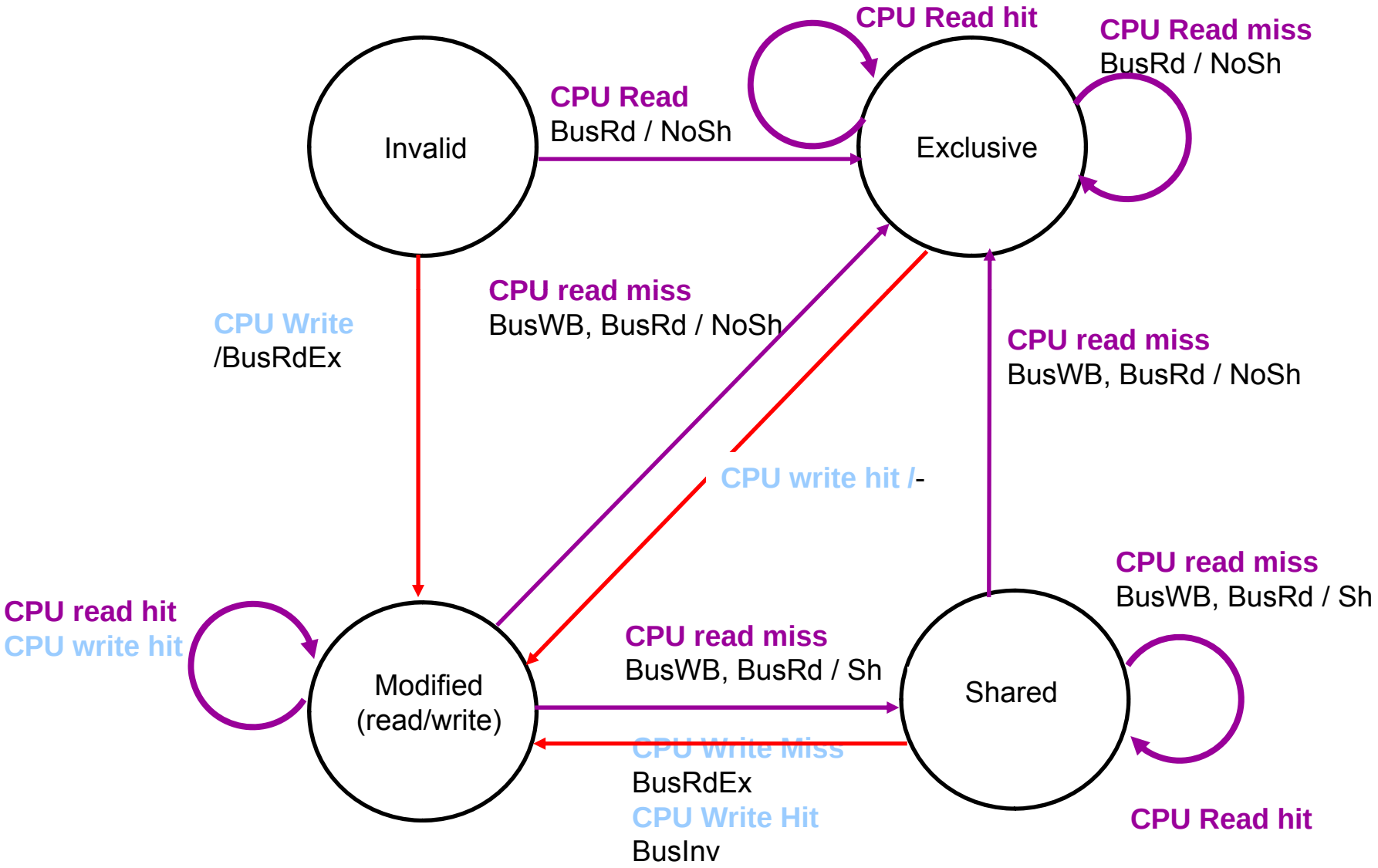
Implementing Snooping Caches

- Multiple processors must be on bus, access to both addresses and data
- Add a few new commands to perform coherency, in addition to read and write
- Processors continuously snoop on address bus
  - If address matches tag, either invalidate or update
- Since every bus transaction checks cache tags, could interfere with CPU just to check:
  - solution 1: duplicate set of tags for L1 caches just to allow checks in parallel with CPU
  - solution 2: L2 cache already duplicate, provided L2 obeys inclusion with L1 cache
    - block size, associativity of L2 affects L1

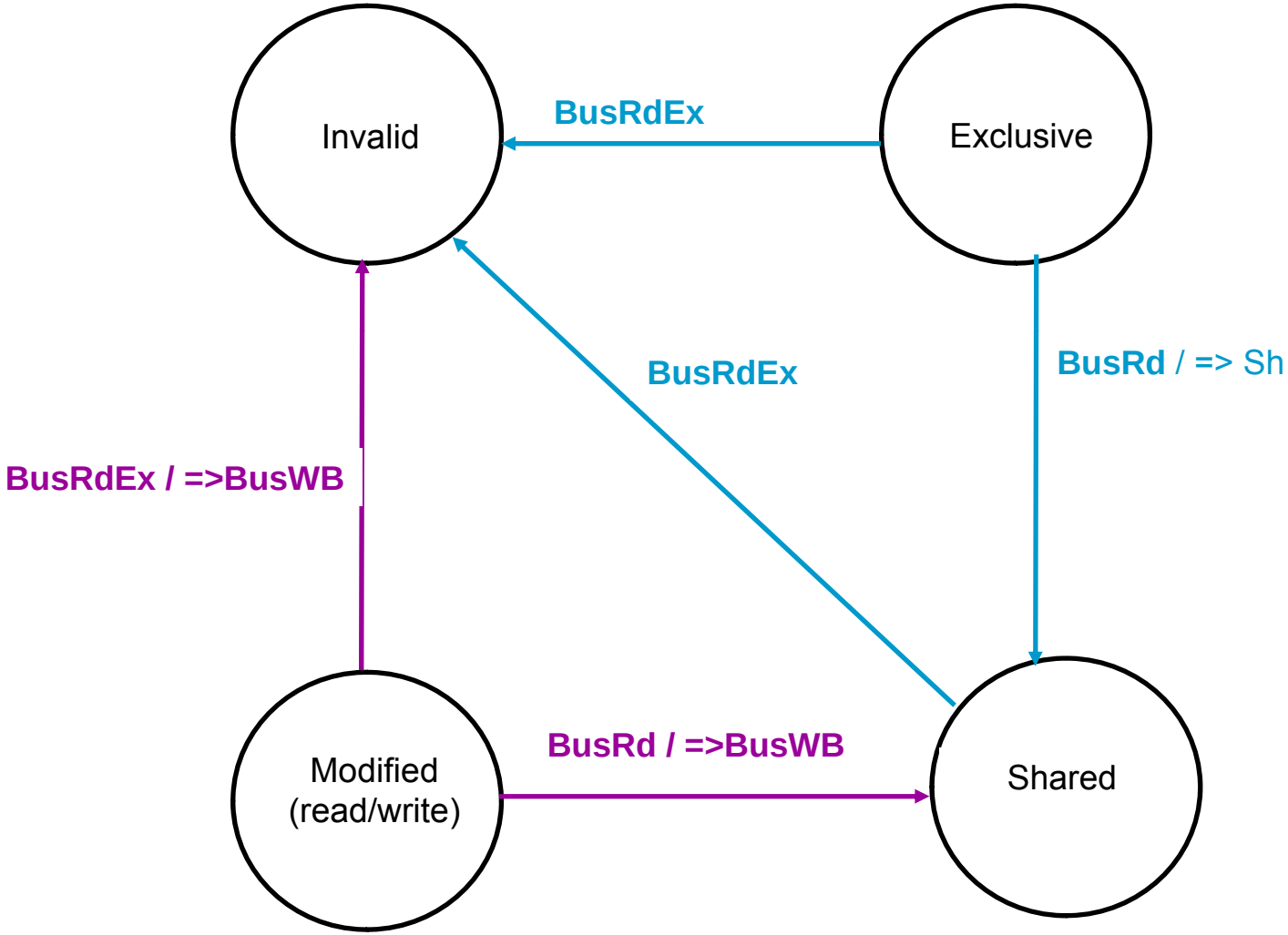
Implementing Snooping Caches

- Bus serializes writes, getting bus ensures no one else can perform memory operation
- On a miss in a write back cache, may have the desired copy and its dirty, so must reply
- Add extra state bit to cache to determine shared or not
- Add 4th state (MESI)

MESI: CPU Requests



MESI: Bus Requests



Fundamental Issues

- 3 Issues to characterize parallel machines
  - 1) Naming
  - 2) Synchronization
  - 3) Performance: Latency and Bandwidth (covered earlier)

Fundamental Issue #1: Naming

- Naming: how to solve large problem fast
  - what data is shared
  - how it is addressed
  - what operations can access data
  - how processes refer to each other
- Choice of naming affects code produced by a compiler; via load where just remember address or keep track of processor number and local virtual address for msg. passing
- Choice of naming affects replication of data; via load in cache memory hierarchy or via SW replication and consistency

Fundamental Issue #1: Naming

- Global physical address space:
  any processor can generate,
  address and access it in a single operation
  - memory can be anywhere:
    virtual addr. translation handles it
- Global virtual address space: if the address space of each process can be configured to contain all shared data of the parallel program
- Segmented shared address space:
  locations are named
  <process number, address>
  uniformly for all processes of the parallel program

Fundamental Issue #2: Synchronization

- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address
  => additional operations to explicitly coordinate:
  e.g., write a flag, awaken a thread, interrupt a processor

Summary: Parallel Framework

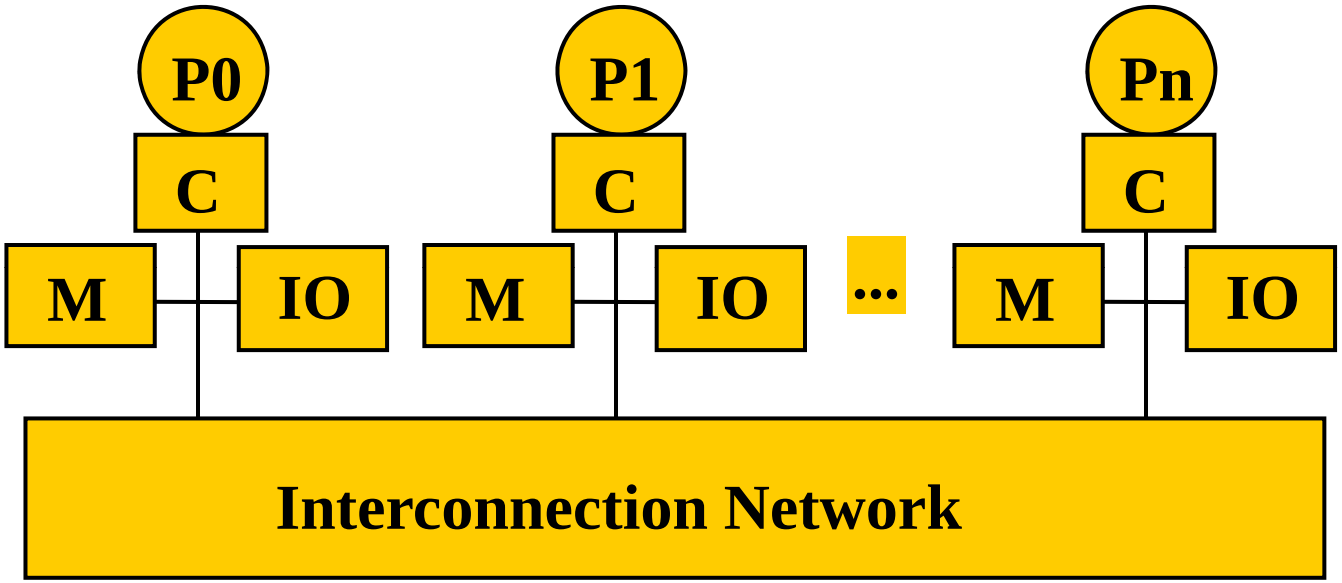
- Layers:
  - Programming Model:
    - Multiprogramming : lots of jobs, no communication
    - Shared address space: communicate via memory
    - Message passing: send and receive messages
    - Data Parallel: several agents operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing)
  - Communication Abstraction:
    - Shared address space: e.g., load, store, atomic swap
    - Message passing: e.g., send, receive library calls
    - Debate over this topic (ease of programming, scaling) => many hardware designs 1:1 programming model

Programming Model
Communication Abstraction
Interconnection SW/OS
Interconnection HW

Larger MPs

- Separate Memory per Processor
- Local or Remote access via memory controller
- One Cache Coherency solution: non-cached pages
- Alternative: directory per cache that tracks state of every block in every cache
  - Which caches have a copies 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)
- Prevent directory as bottleneck?
  distribute directory entries with memory, each keeping track of which Procs have copies of their blocks

Distributed Directory MPs



C - Cache

M - Memory

IO - Input/Output

Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached (no processor has it; not valid in any cache)
  - Exclusive: 1 processor (owner) has data;
    memory out-of-date
- In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
- 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

Directory Protocol

- No bus and don't want to broadcast:
  - interconnect no longer single arbitration point
  - all messages have explicit responses
- 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
- Example messages on next slide:
  P = processor number, A = address

Directory Protocol Messages

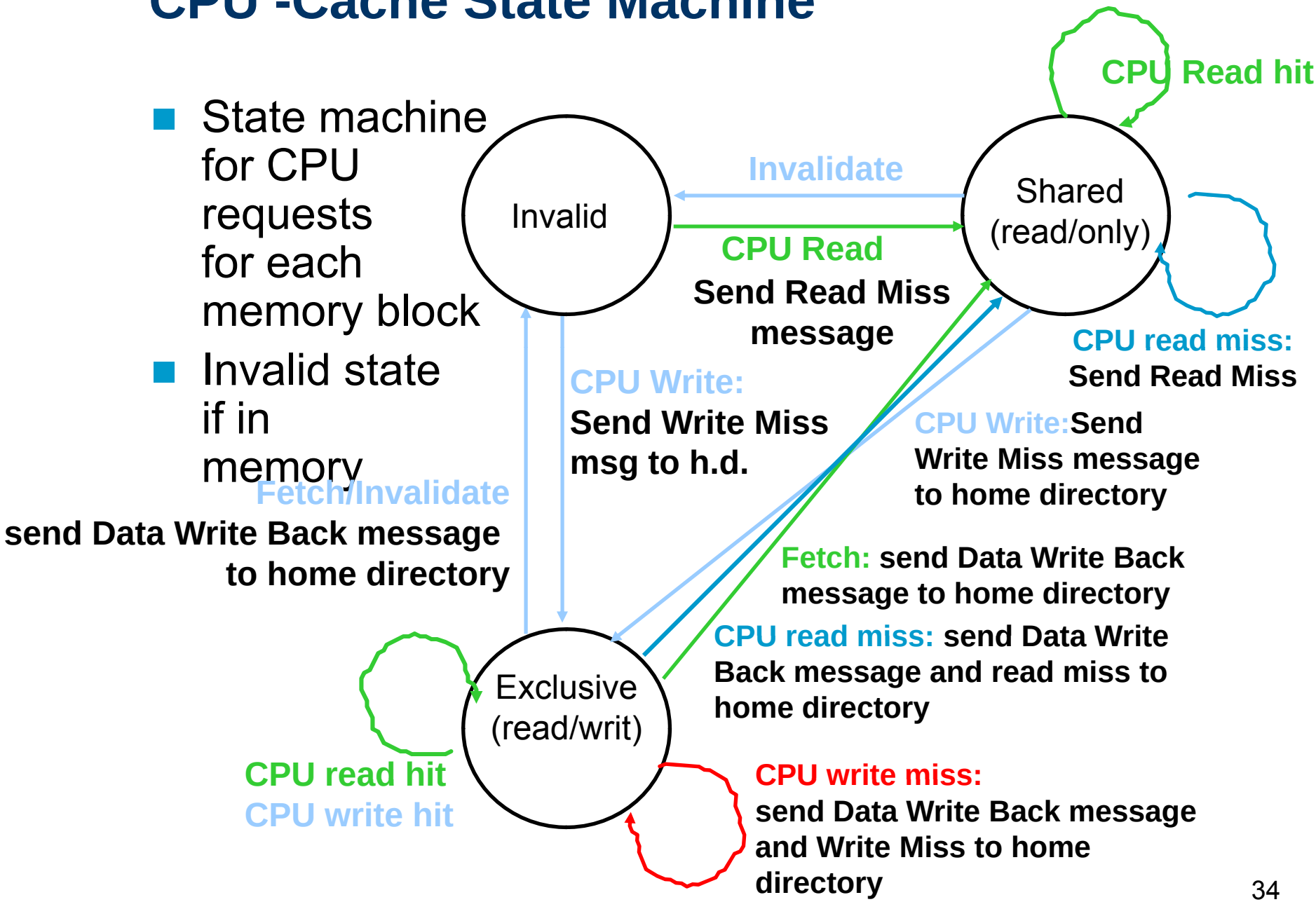
Message type	Source	Destination	Msg
Read miss	Local cache	Home directory	P, A
Processor P reads data at address A; make P a read sharer and arrange to send data back			
Write miss	Local cache	Home directory	P, A
Processor P writes data at address A; make P the exclusive owner and arrange to send data back			
Invalidate	Home directory	Remote caches	A
Invalidate a shared copy at address A.			
Fetch	Home directory	Remote cache	A
Fetch the block at address A and send it to its home directory			
Fetch/Invalidate	Home directory	Remote cache	A
Fetch the block at address A and send it to its home directory; invalidate the block in the cache			
Data value reply	Home directory	Local cache	Data
Return a data value from the home memory (read miss response)			
Data write-back	Remote cache	Home directory	A, Data
Write-back a data value for address A (invalidate response)			

State Transition Diagram for an Individual Cache Block in a Directory Based System

- States identical to snoopy case; transactions very similar
- Transitions caused by read misses, write misses, invalidates, data fetch requests
- Generates read miss & write miss msg to home directory
- Write misses that were broadcast on the bus for snooping => explicit invalidate & data fetch requests
- Note: on a write, a cache block is bigger, so need to read the full cache block

CPU -Cache State Machine

- State machine for CPU requests for each memory block
- Invalid state if in memory

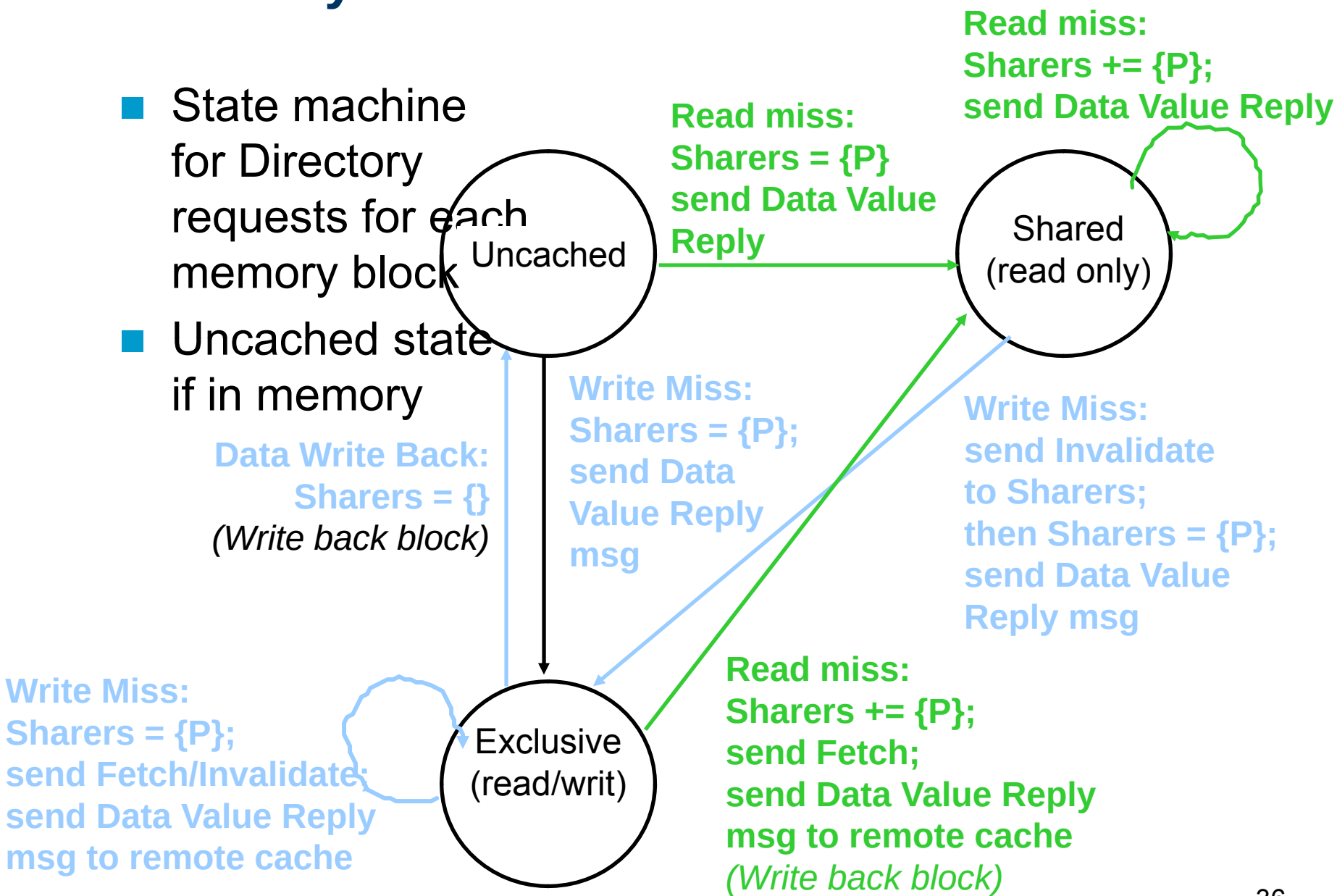


State Transition Diagram for the Directory

- Same states & structure as the transition diagram for an individual cache
- 2 actions: update of directory state & send msgs to statisfy requests
- Tracks all copies of memory block.
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message.

Directory State Machine

- State machine for Directory requests for each memory block
- Uncached state if in memory



Example Directory Protocol

- Message sent to directory causes two actions:
  - Update the directory
  - More messages to satisfy request
- Block is in Uncached state: the copy in memory is the current value; only possible requests for that block are:
  - Read miss: requesting processor sent data from memory &requestor made only sharing node; state of block made Shared.
  - Write miss: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.
- Block is Shared => the memory value is up-to-date:
  - Read miss: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - Write miss: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.

Example Directory Protocol

- Block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) => three possible directory requests:
  - Read miss: owner processor sent data fetch message, causing state of block in owner's cache to transition to Shared and causes owner to send data to directory, where it is written to memory & sent back to requesting processor.
    Identity of requesting processor is added to set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy). State is shared.
  - Data write-back: owner processor is replacing the block and hence must write it back, making memory copy up-to-date
    (the home directory essentially becomes the owner), the block is now Uncached, and the Sharer set is empty.
  - Write miss: block has a new owner. A message is sent to old owner causing the cache to send the value of the block to the directory from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to identity of new owner, and state of block is made Exclusive.

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														

A1 and A2 map to the same cache block

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														

A1 and A2 map to the same cache block

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														

A1 and A2 map to the same cache block

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

A1 and A2 map to the same cache block

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														

A1 and A2 map to the same cache block

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

A1 and A2 map to the same cache block

Implementing a Directory

- We assume operations atomic, but they are not; reality is much harder; must avoid deadlock when run out of buffers in network (see Appendix I) –
  The devil is in the details
- Optimizations:
  - read miss or write miss in Exclusive: send data directly to requestor from owner vs. 1st to memory and then from memory to requestor