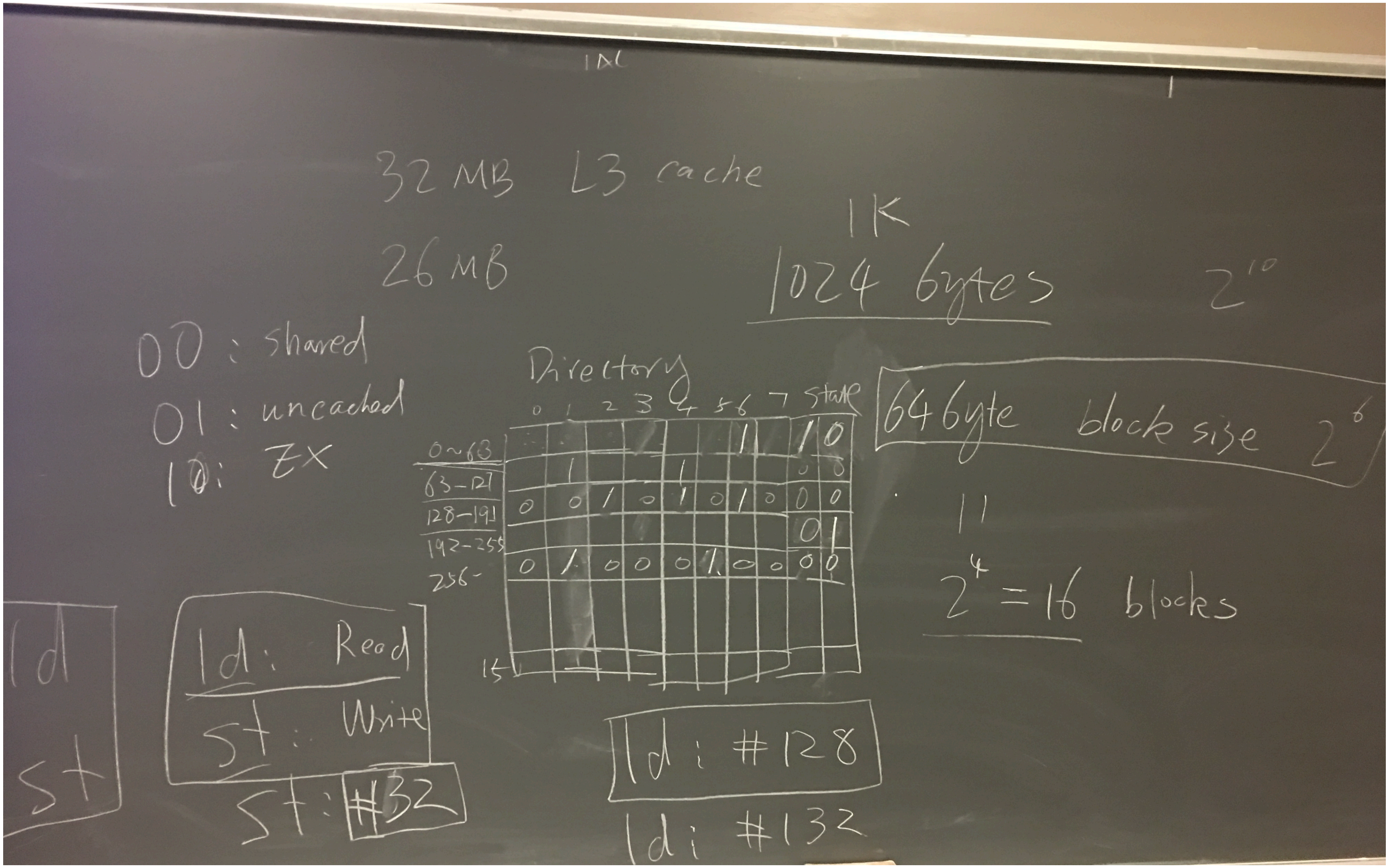
Things

- Uploaded assignment #5 and final exam used in 2016
  - Will update study topic page later today
- No audio in the video of last lecture→ re-record

- Bonus Questions
  - Extra and optional questions for total and max 15 points to your final percentage grade
  - Due date:
    - » 12/07 Friday Assignment #5
    - » 12/12 Wednesday Final Exam
    - » 12/16 Sunday Bonus Questions. Strict cutoff, no any extension
- Today and Monday
  - Finish chapter 5, TLP
  - Possibly Introducing Chapter 7
- Next Wednesday
  - More on Chapter 7
  - Recap and review

Pictures Used During the Lecture



Lecture 24: Thread Level Parallelism -- Distributed Shared Memory and Directory-based Coherence Protocol

CSCE 513 Computer Architecture

Department of Computer Science and Engineering
Yonghong Yan
yanyh@cse.sc.edu
https://passlab.github.io/CSCE513

Topics for Thread Level Parallelism (TLP)

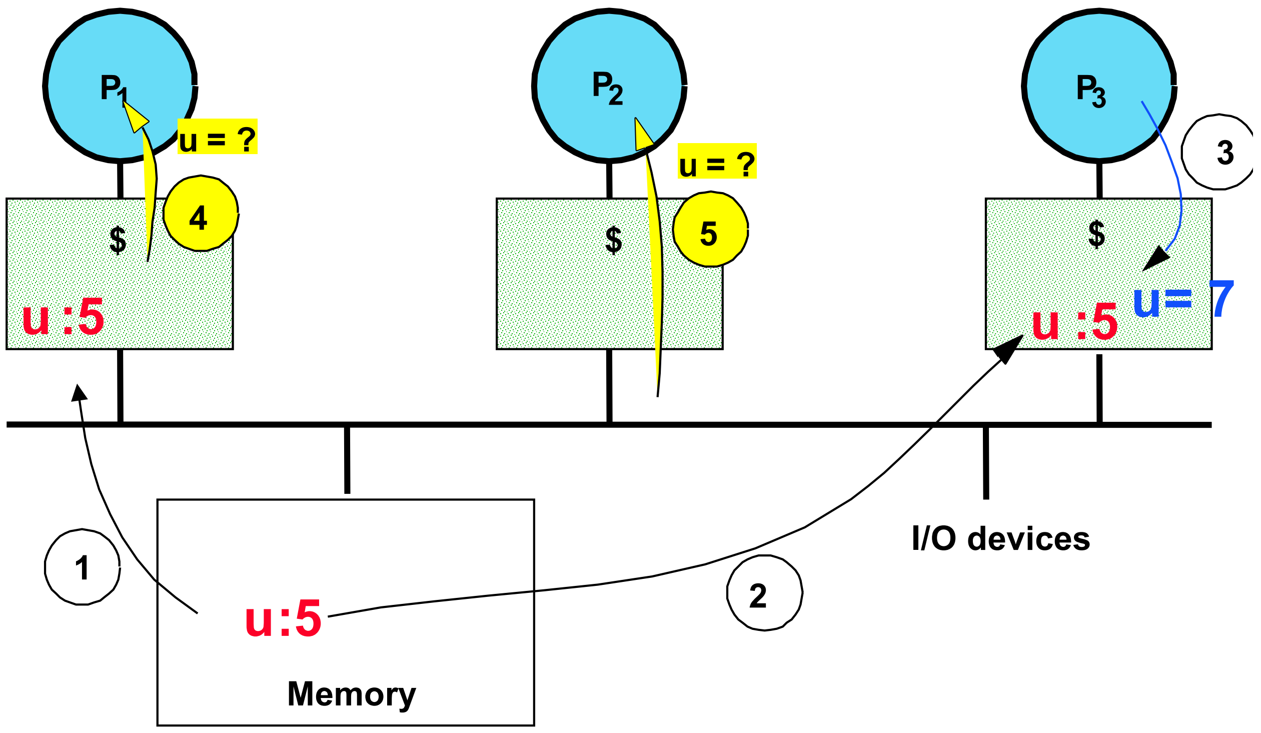
- Parallelism (centered around …)
  - Instruction Level Parallelism
  - Data Level Parallelism
  - Thread Level Parallelism

- TLP Introduction
  - 5.1
- SMP and Snooping Cache Coherence Protocol
  - 5.2
- Distributed Shared-Memory and Directory-Based Coherence
  - 5.4
- Synchronization Basics and Memory Consistency Model
  - 5.5, 5.6
- Others

Example Cache Coherence Problem

```
int count = 5;
int * u= &count;
….
```

T1 (P1)	T2 (P2)	T3 (P3)
a1 = *u;		
		a3 = *u;
		*u = 7;
b1 = *u		
	a2 = *u	



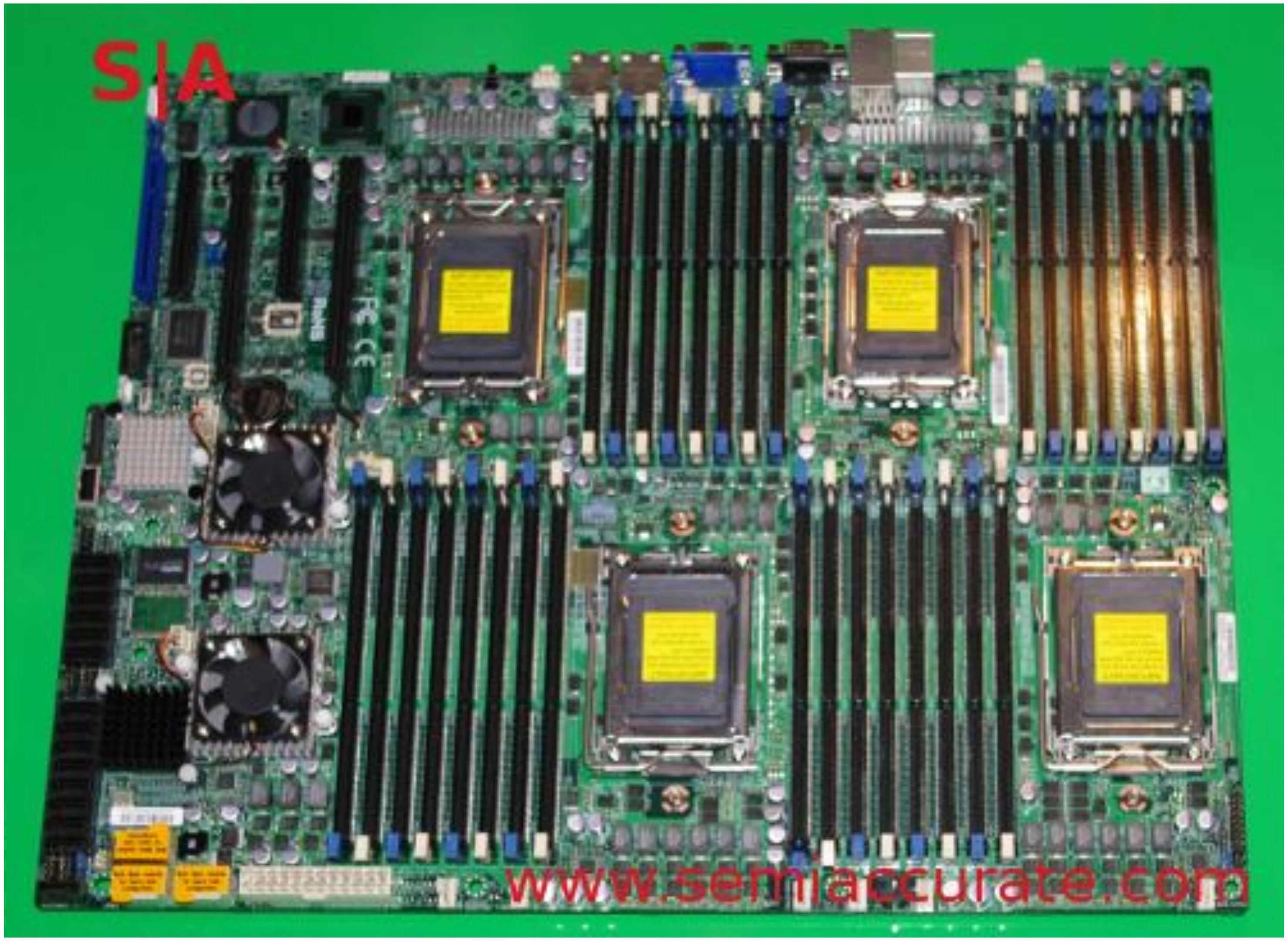
Things to note:

- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value and when
  - Processes accessing main memory may see very stale value
- Unacceptable to programs, and frequent!

Cache Coherence Protocols

- **Snooping Protocols**
  - **Send all requests for data to all processors, the address**
  - **Processors snoop a bus 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 centralized shared memory machines**
- **Directory-Based Protocols**
  - **Keep track of what is being shared in centralized location**
  - **Distributed memory => distributed directory for scalability (avoids bottlenecks)**
  - **Send point-to-point requests to processors via network**
  - **Scales better than Snooping**
  - **Commonly used for distributed shared memory machines**

Distributed Shared Memory Systems



Distributed Directory MPs

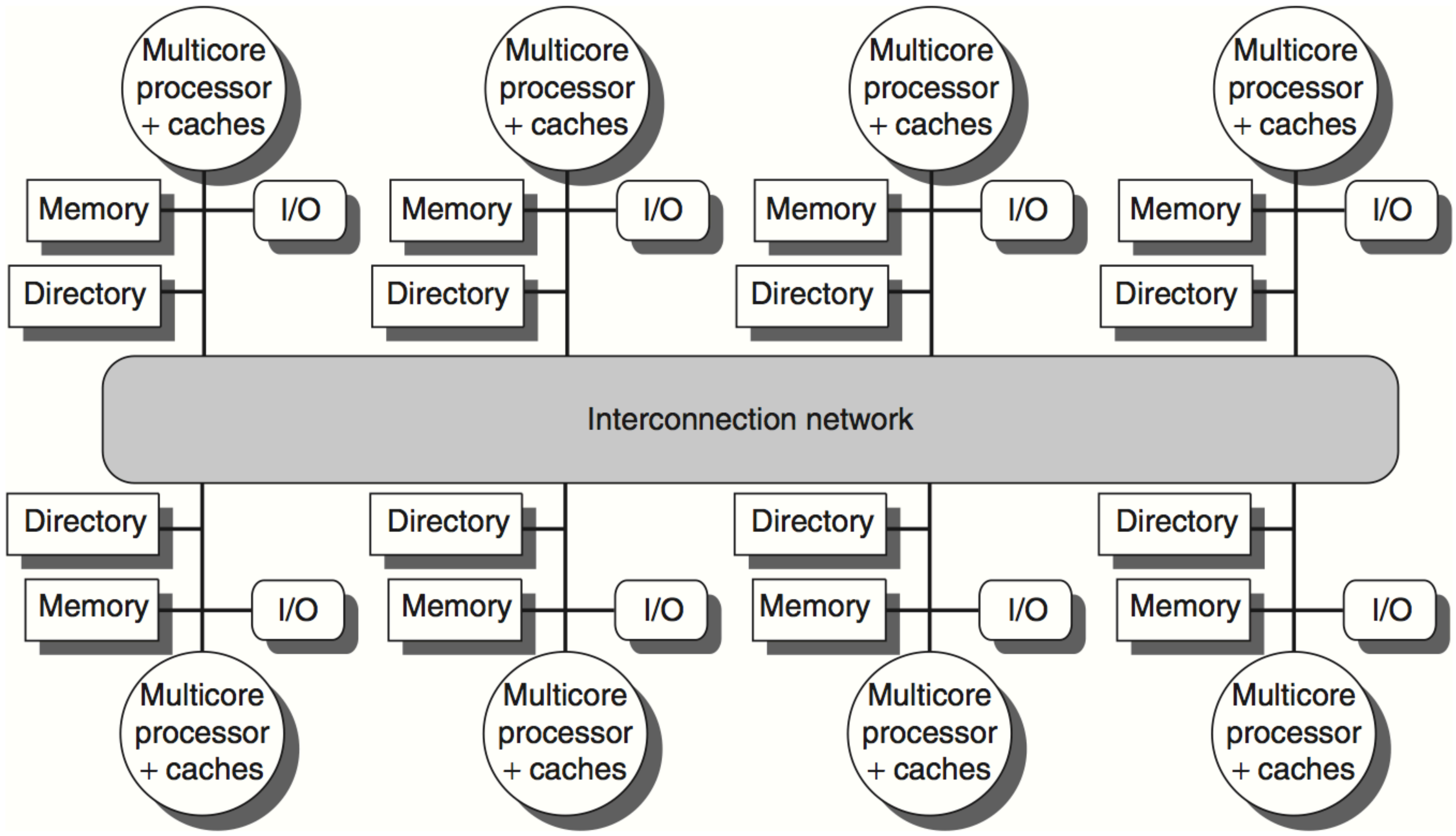


Figure 5.20 A directory is added to each node to implement cache coherence in a distributed-memory multiprocessor. In this case, a node is shown as a single multicore chip, and the directory information for the associated memory may reside either on or off the multicore. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The coherence mechanism would handle both the maintenance of the directory information and any coherence actions needed within the multicore node.

Directory for Distributed Shared Memory Systems

- Typically Distributed Shared Memory Systems
  - E.g. SGI UV 3000 up to 256 CPU sockets
  - Local or remote memory access via memory controller
- Directory per block 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 *ƒ*(memory size) vs. *ƒ*(cache size)
- Implement in shared L3 cache
  - Keep bit vector of size = # cores for each block in L3
  - Not scalable beyond shared L3
- Prevent directory as bottleneck?
  - Distribute directory entries with memory, each keeping track of which processors have copies of their blocks

Directory-based Cache Coherence Protocol

- Similar to Snoopy Protocol: Three states of each block
  - 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)
  - The ownership is also tracked
- Assumptions:
  - 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
- Handling two operations
  - Write to shared block
  - Read/write miss
- Example messages on next slide:
  P = processor number, A = address

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

Example Directory Protocol

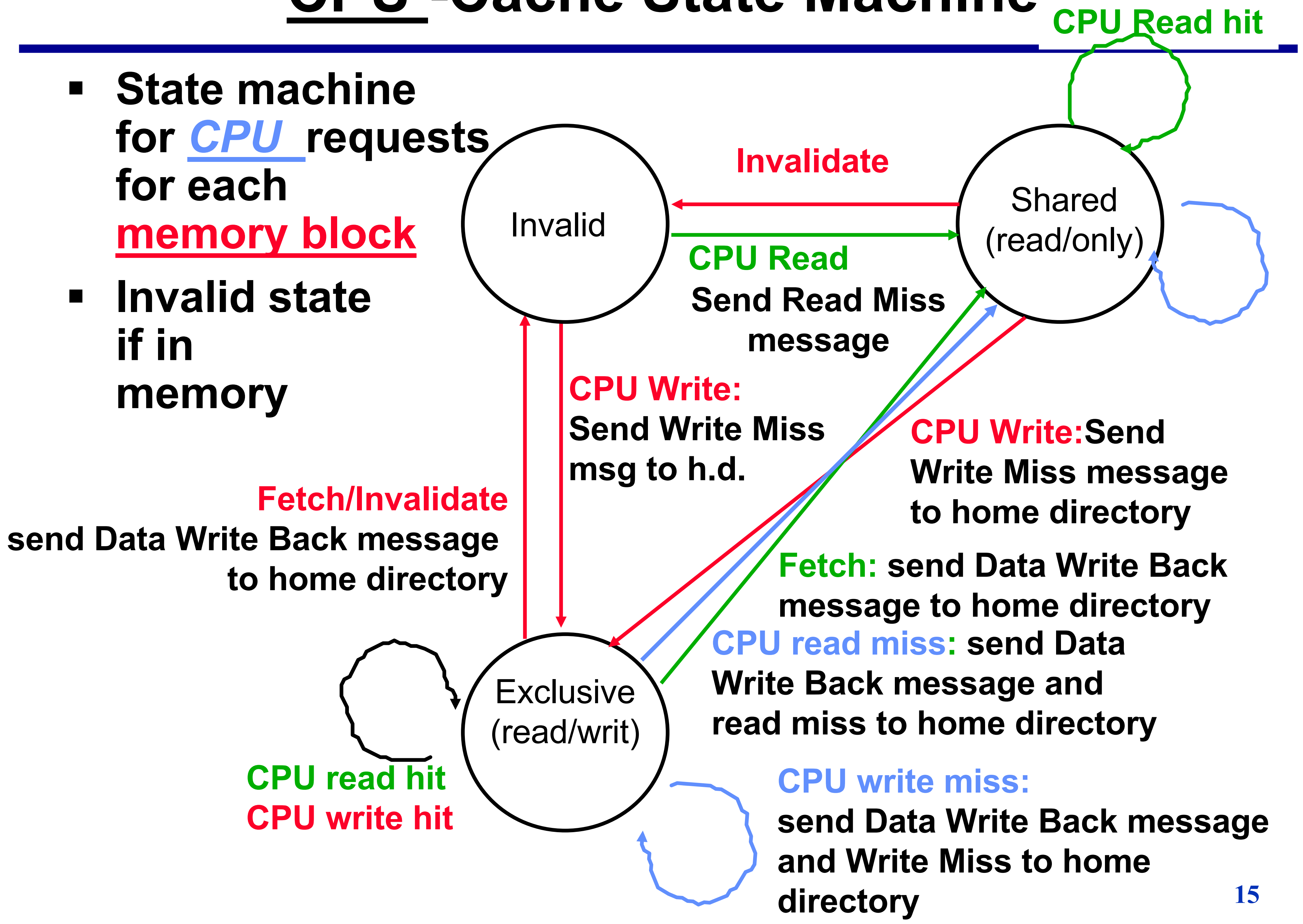
- **Message sent to directory causes two actions:**
  - **Update the directory**
  - **More messages to satisfy request**
- **Block is in Uncached state: the copy in memory is the current value; only possible requests for that block are:**
  - **Read miss: requesting processor is sent data from memory & is 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.

CPU -Cache State Machine

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



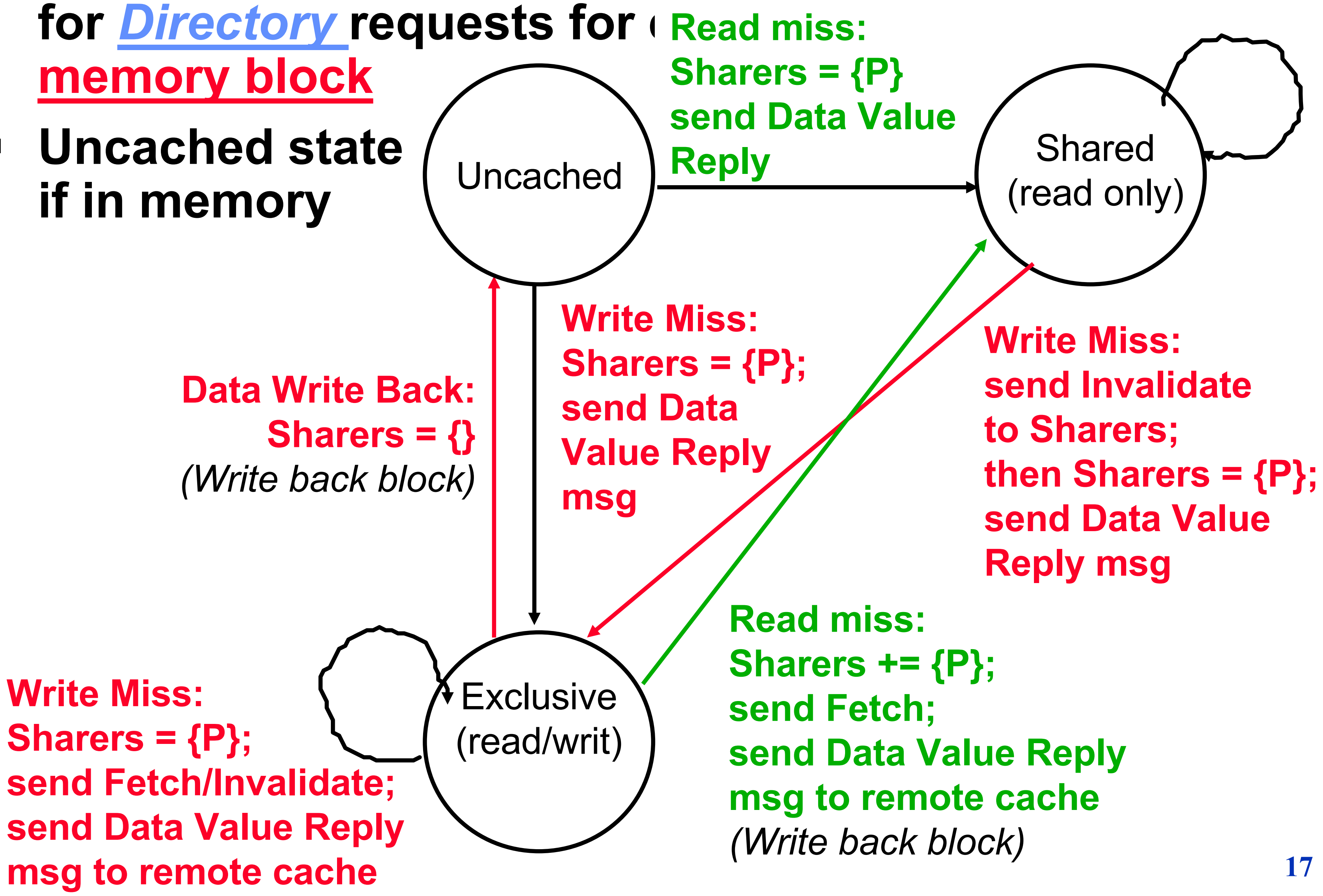
CPU Read hit
Invalid
Shared (read/only)
Exclusive (read/writ)
Invalidate
CPU Read
Send Read Miss message
CPU Write:
Send Write Miss msg to h.d.
CPU Write:Send Write Miss message to home directory
Fetch/Invalidate
send Data Write Back message to home directory
Fetch: send Data Write Back message to home directory
CPU read miss: send Data Write Back message and read miss to home directory
CPU read hit
CPU write hit
CPU write miss:
send Data Write Back message and Write Miss to home directory

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 satisfy requests**
- **Tracks all copies of memory block.**
- **Also indicates an action that updates the sharing set, Sharers, as well as sending a message.**

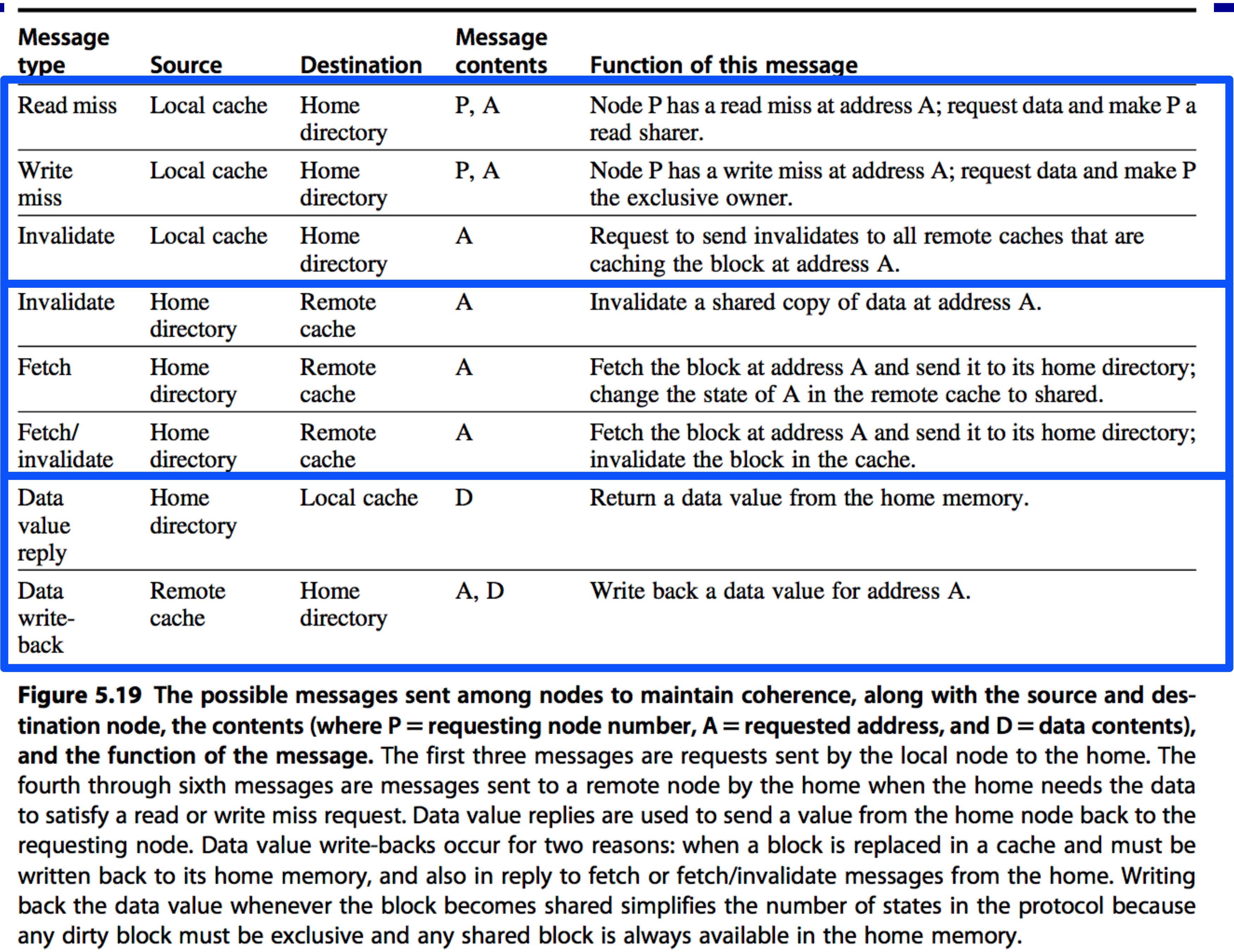
Directory State Machine

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



Uncached
Shared (read only)
Exclusive (read/writ)
Read miss: Sharers = {P} send Data Value Reply
Write Miss: Sharers = {P}; send Data Value Reply msg
Data Write Back: Sharers = {} (Write back block)
Write Miss: send Invalidate to Sharers; then Sharers = {P}; send Data Value Reply msg
Read miss: Sharers += {P}; send Fetch; send Data Value Reply msg to remote cache (Write back block)
Write Miss: Sharers = {P}; send Fetch/Invalidate; send Data Value Reply msg to remote cache

Directory Protocol Messages



Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
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	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write back a data value for address A.

Figure 5.19 The possible messages sent among nodes to maintain coherence, along with the source and destination node, the contents (where P = requesting node number, A = requested address, and D = data contents), and the function of the message. The first three messages are requests sent by the local node to the home. The fourth through sixth messages are messages sent to a remote node by the home when the home needs the data to satisfy a read or write miss request. Data value replies are used to send a value from the home node back to the requesting node. Data value write-backs occur for two reasons: when a block is replaced in a cache and must be written back to its home memory, and also in reply to fetch or fetch/invalidate messages from the home. Writing back the data value whenever the block becomes shared simplifies the number of states in the protocol because any dirty block must be exclusive and any shared block is always available in the home memory.

Example

Processor 1 Processor 2 Interconnect Directory Memory

	P1			*P2*			*Bus*				*Directory*			*Memor*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Actio*	*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*			*Memor*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Actio*	*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			Memor
step	Statε	Addr	Value	Statε	Addr	Value	Actio	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*			*Memor*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Actio*	*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				*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			Memor
step	State	Addr	Value	State	Addr	Value	Actio	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				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			Memor
step	State	Addr	Value	State	Addr	Value	Actio	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				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
- 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