Caches for Multiprocessor Architectures

Slides adopted from David Patterson (1998, 2001) and David E. Culler (2001), Copyright 1998-2002, University of California Berkeley

David Andrews
Computer Engineering Group
University of Paderborn

dandrews@ittc.ku.edu

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

First, Where Do Multiprocessors Fit ?

- SISD (Single Instruction Single Data)
  - Uniprocessors
- MISD (Multiple Instruction Single Data)
  - ???; multiple processors on a single data stream
- SIMD (Single Instruction Multiple Data)
  - Historical Examples: Illiac-IV, CM-2
    - Simple programming model
    - Low overhead
    - Flexibility
    - All custom integrated circuits
  - Recent Example: IBM Blue Gene, Cell Processor
- MIMD (Multiple Instruction Multiple Data)
  - Historical Examples: Cray T3D, SGI Origin
    - Flexible
    - *Use off-the-shelf micros*
  - Recent Example: Virtex 2 P30 with 2 PPC 405's
- Within General Computing MIMD current winner:
  - Clusters, x Core PC's

Major MIMD Styles

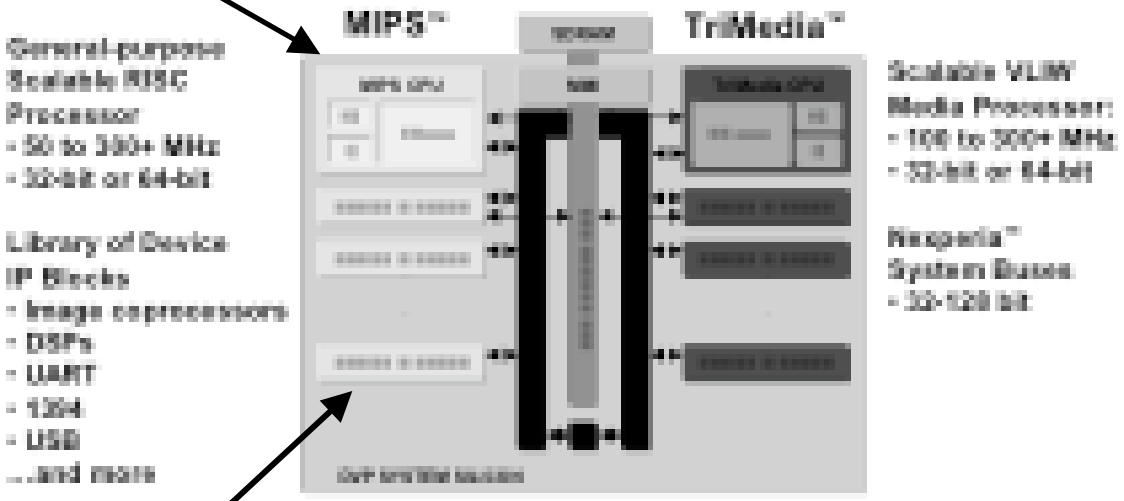
- Generalized Taxonomies Along Memory Access Lines
- Centralized Shared Memory
  - Uniform Memory Access or Shared Memory Processor
  - Global Address Space for All. Constant Access Time from Anywhere
- Decentralized memory (memory module with CPU)
    - get more memory bandwidth, lower memory latency
    - Drawback: Longer communication latency
    - Drawback: Software model more complex

- Multiprocessor Systems on Chip Taxonomies Include
  - Variability of “Cores”
    - Homogeneous: All Processors Identical
    - Heterogeneous: Different Cores {CPU + DSP}

Heterogeneous MPSoC

A MPSoC Example: Nexperia™ DVP

General Purpose CPU



SIMD Array

Classic Communication Models

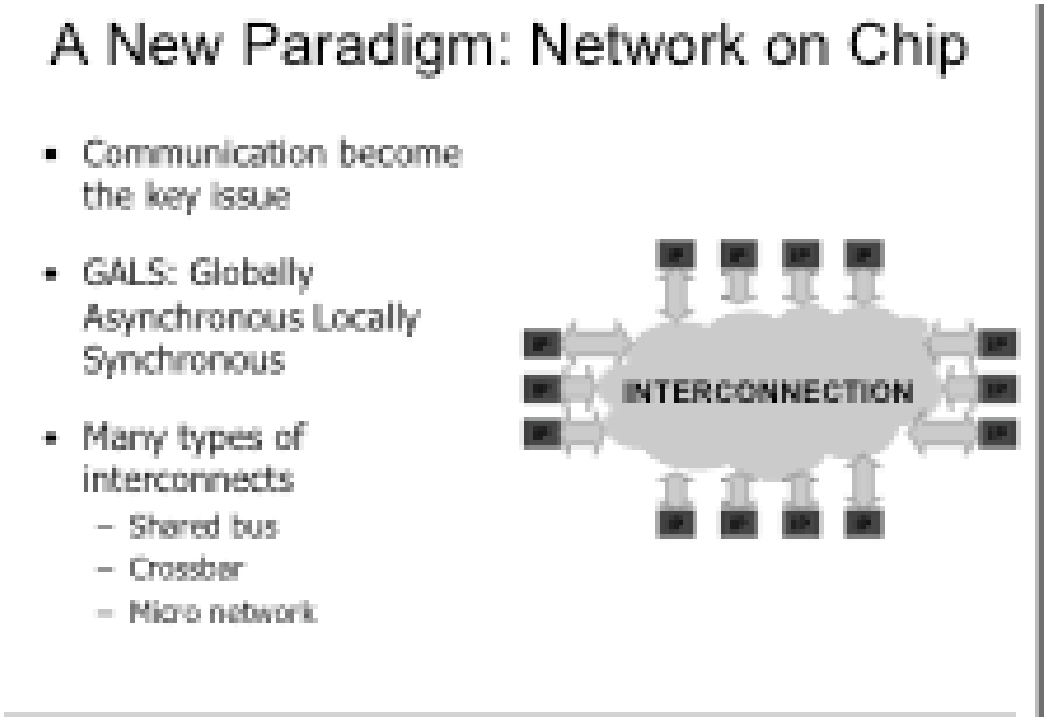
- Shared Memory
  - Processors communicate with shared address space
  - Easy on small-scale machines
  - Advantages:
    - Model of choice for uniprocessors, small-scale MPs
    - Ease of programming
    - Lower latency
    - Easier to use hardware controlled caching
- Message passing
  - Processors have private memories, communicate via messages
  - Advantages:
    - Less hardware, easier to design
    - Focuses attention on costly non-local operations
- Can support either SW model on either HW base

SMP—Shared Memory Organization

- Caches serve to:
  - Increase bandwidth versus bus/memory
  - Reduce latency of access
  - Valuable for both private data and shared data
- I/O & Memory Global Access

Blob Interconnect

- Processors to Memory and I/O
  - Important for Embedded Systems
    - MPSoC: The “oC” on Chip Presents Interesting Studies….



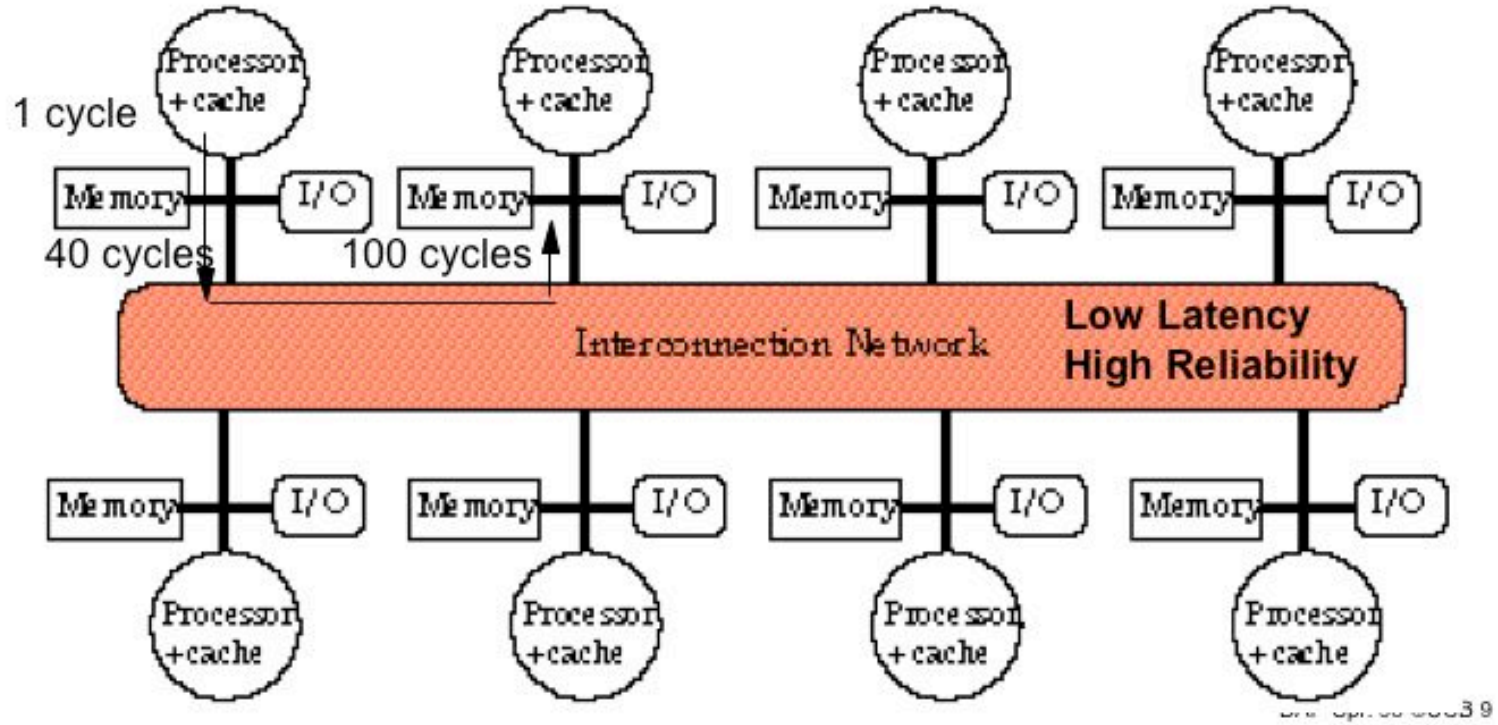
Picture from Massimo Poncino’s Slides

SMP Interconnect

- All Memory Locations Equal Access Time so SMP = Symmetric Multiprocessor
  - Sharing Limited Bandwidth as Processors and I/O Added
  - Crossbar: Eliminates Contention but Expensive
  - Multistage Interconnection: Less Expensive than Crossbar with More BW than common bus
  - “Dance Hall” designs: All Processors on Left, Memories on Right
- Today, We’ll Simplify For Cache Discussions on Single Bus Based Interconnections
  - Makes Coherency Easier
  - Good For Small Numbers of Processors

Large-Scale MP Designs

- **Memory: distributed with nonuniform access time ("numa") and scalable interconnect (distributed memory)**
- **Examples: T3E: (see Ch. 1, Figs 1-21, page 45 of [CSG96])**

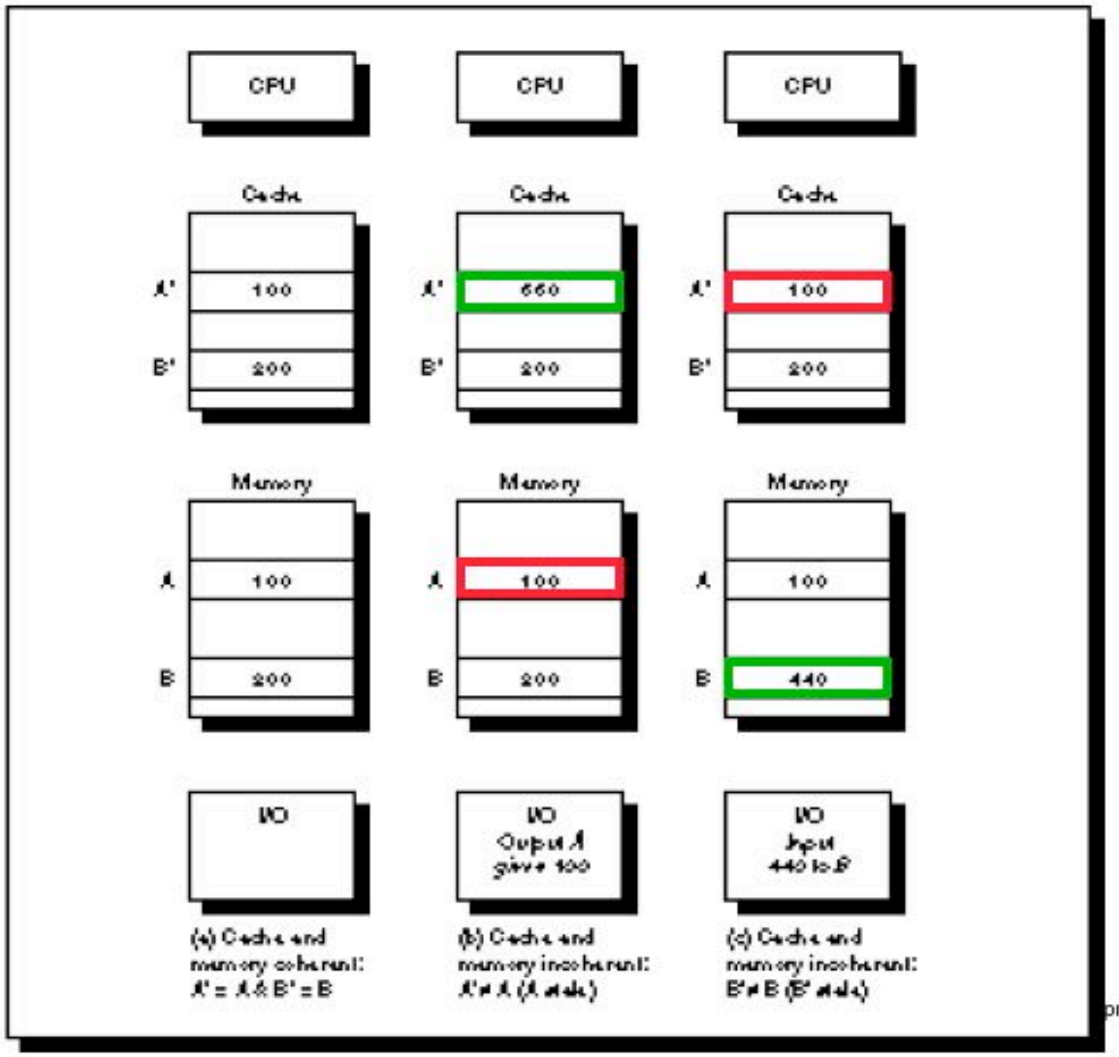


Classic Cache Coherency For Multiprocessors

-SMP Architectures := Snoopy Cache

-NUMA Architectures := Global Directory Cache

The Problem of Cache Coherency



What Does Coherency Mean?

- Informally:
  - "Any read must return the most recent write"
  - Too strict and too difficult to implement
- Better:
  - "Any write must eventually be seen by a read"
  - All writes are seen in proper order ("serialization")
- Two rules to ensure this:
  - "If P writes x and P1 reads it, P's write will be seen by P1 if the read and write are sufficiently far apart"
  - Writes to a single location are serialized: seen in one order
    - Latest write will be seen
    - Otherwise could see writes in illogical order (could see older value after a newer value)

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 (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

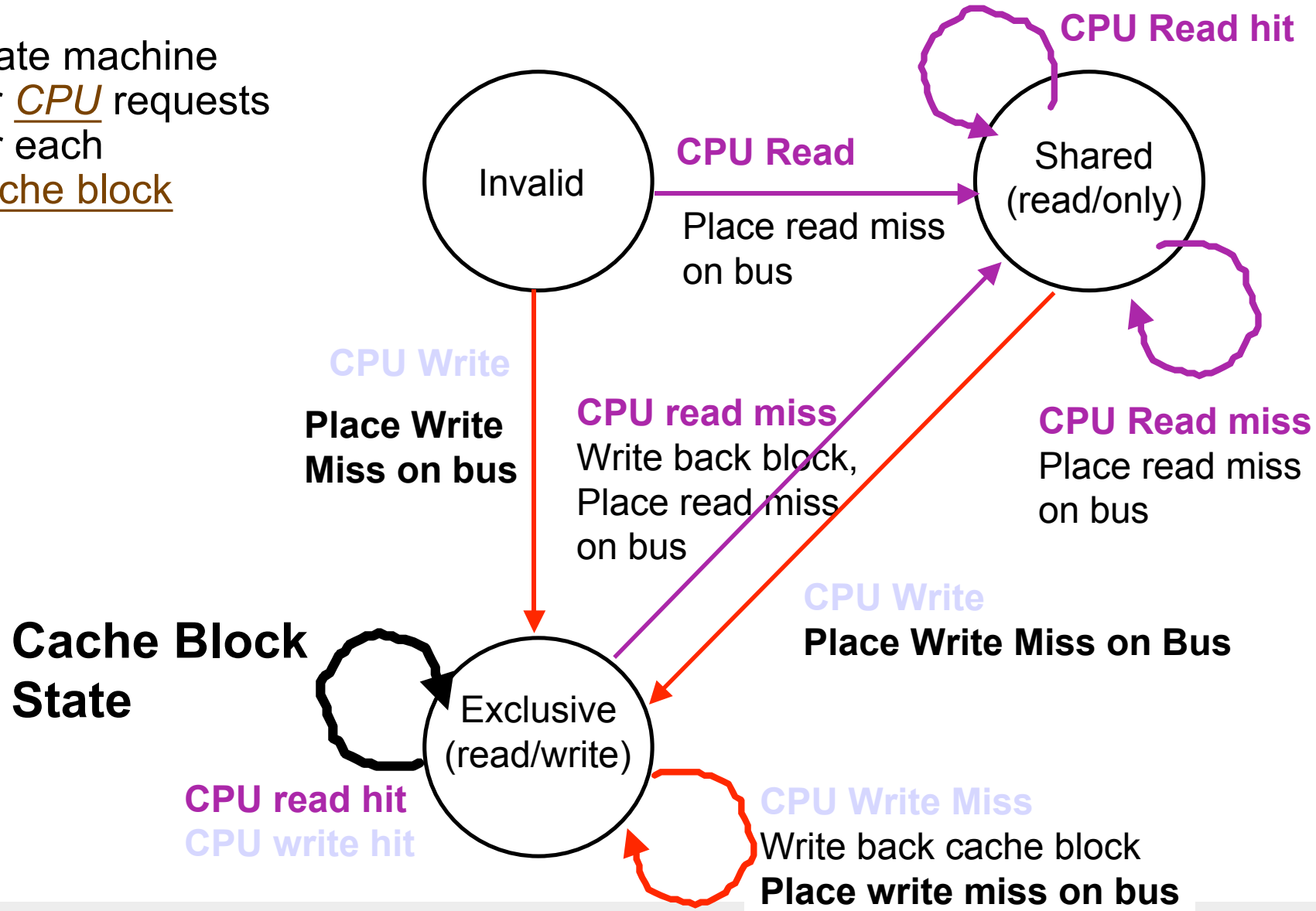
- Write Invalidate Protocol:
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches which snoop and *invalidate* any copies
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy
- Write Broadcast 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

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

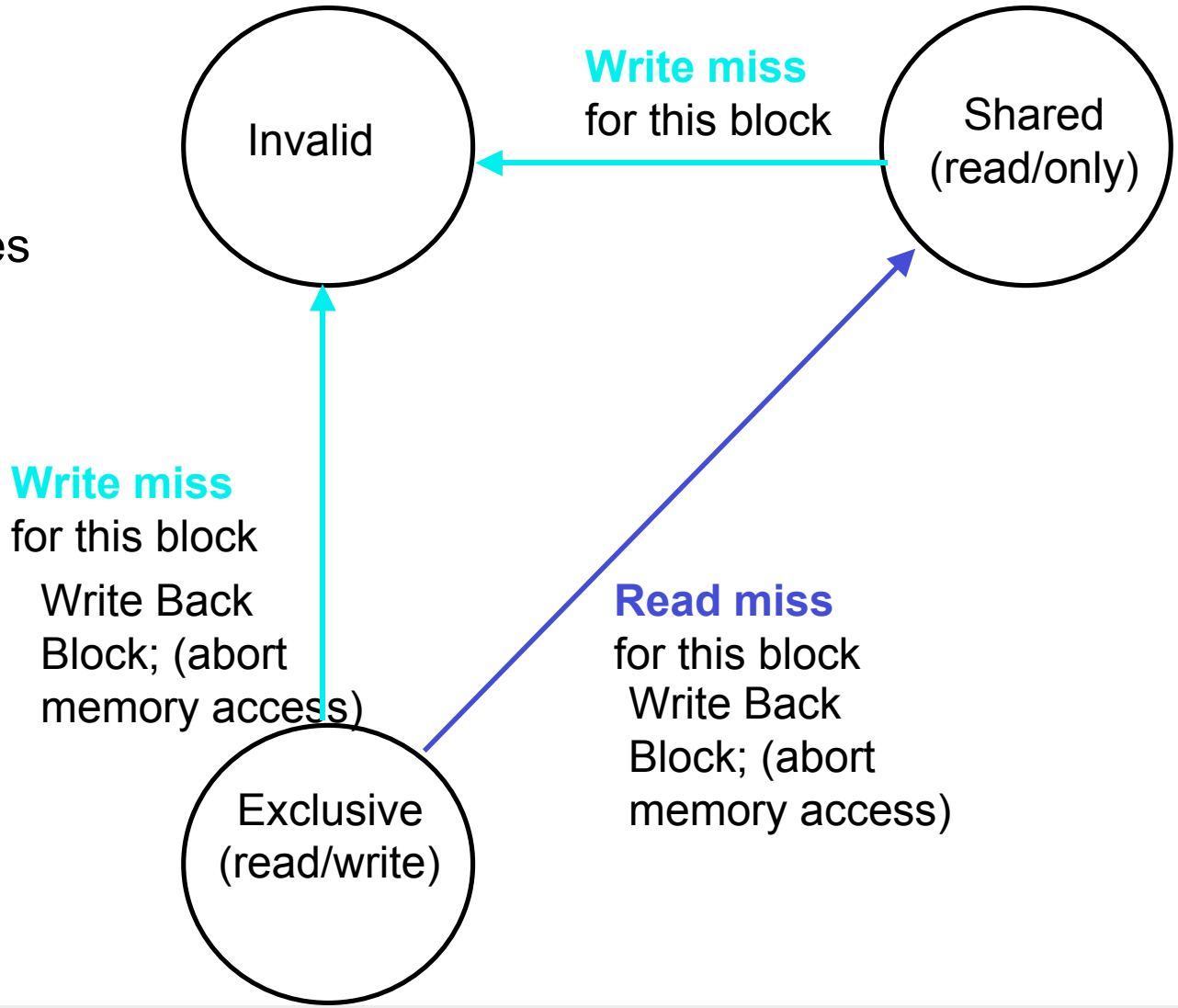
Snoopy-Cache State Machine-I

- State machine for *CPU* requests for each cache block



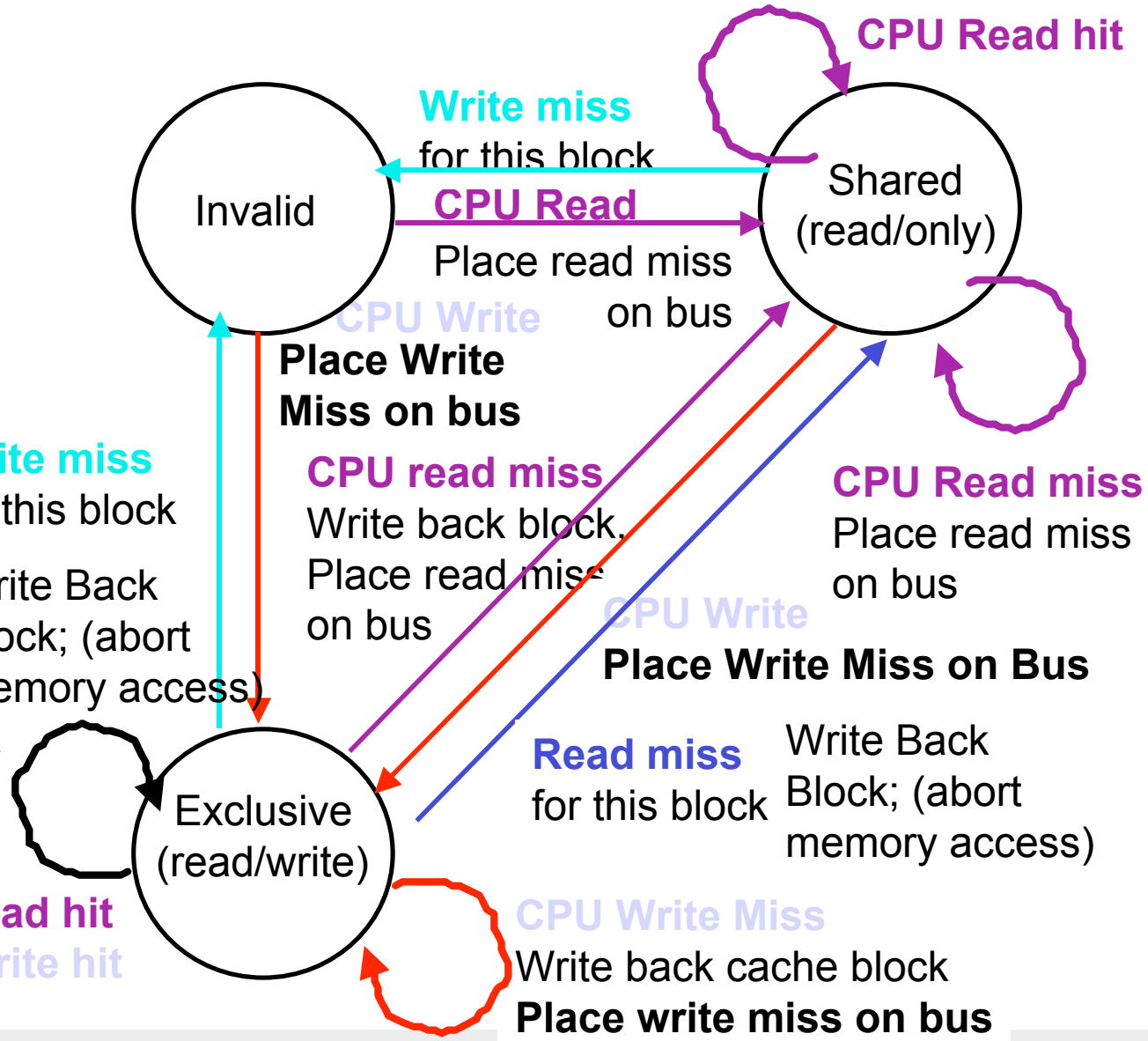
Snoopy-Cache State Machine-II

- State machine for *bus* requests for each cache block
- Appendix E? gives details of bus requests



Snoopy-Cache State Machine-III

- State machine for *CPU* requests for each cache block **and** for *bus* requests for each cache block



Example

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

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

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 A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory 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 A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory 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 A1 and A2 map to same cache block

Example

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory 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 A1 and A2 map to same cache block

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	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 A1 and A2 map to same cache block,
but A1 != A2

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

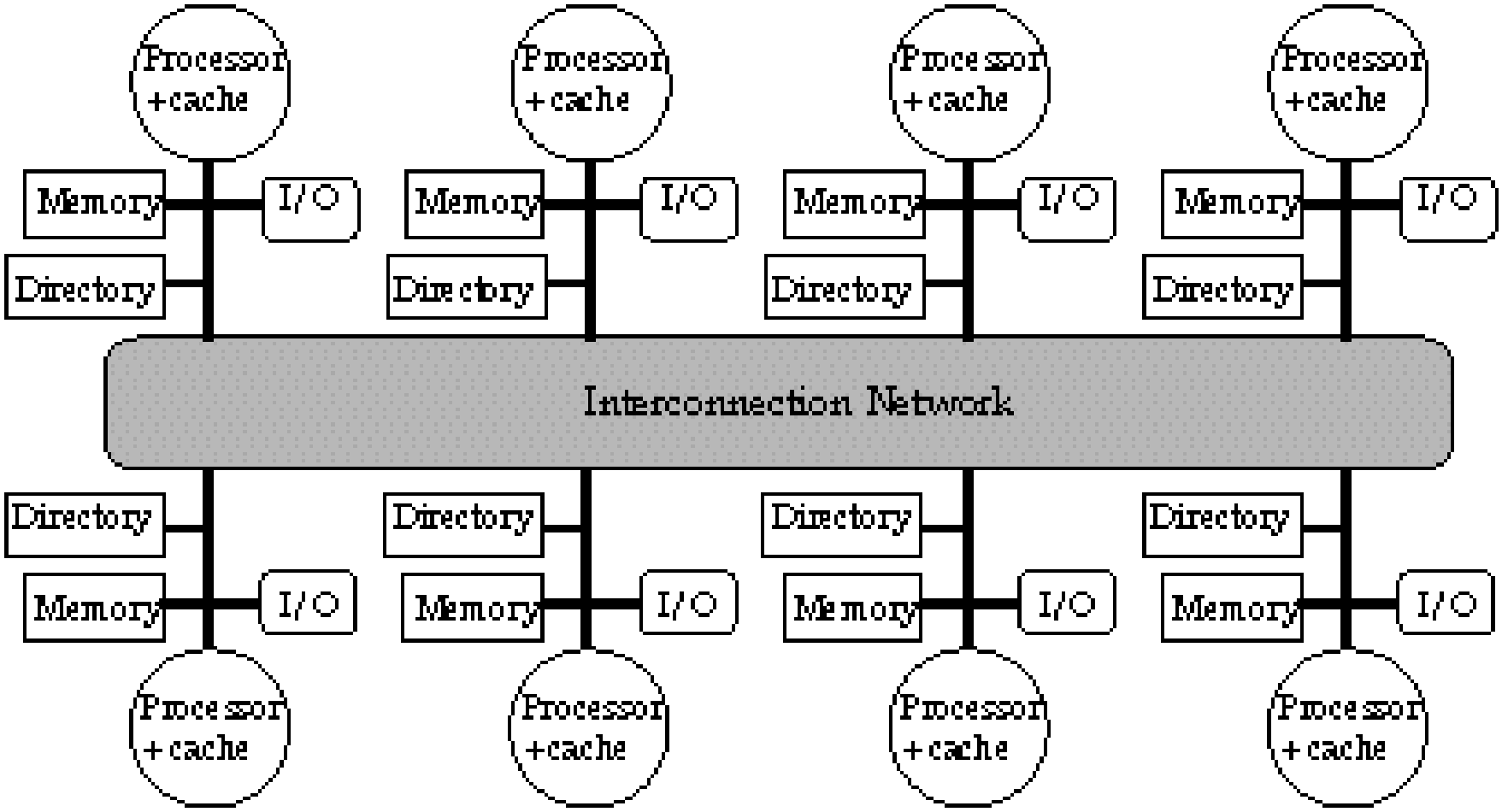
Implementing Snooping Caches

- 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)

Larger MPs

- Separate Memory per Processor
- Local or Remote access via memory controller
- 1 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



Network Examples

- Bi-directional Ring – EX: HP V Class
- 2-D Mesh and Hypercube – SGI Origin and Cray T3E
- Crossbar and Omega Network – SMPs, IBM SP3, and IP Routers
- Clusters using ethernet, Gigabit ethernet, Myrinet, etc.

Properties of various networks will be discussed later

CC-NUMA Multiprocessor: Directory Protocol

- What is Cache Coherent Non-Uniform Memory Access (CC-NUMA)?
- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached (no processor hasit; 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)
- Directory Size: Big => Limited Directory Schemes (Not to be discussed)

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

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.

Rough Timing Analysis

The previous Example looked at the bandwidth demands. The other key issue for a parallel program is remote memory access time, or latency. To get insight into this, we use a simple example of a directory-based machine. Figure 8.30 shows the parameters we assume for our simple machine. It assumes that the time to first word for a local memory access is 25 cycles and that the path to local memory is 8 bytes wide, while the network interconnect is 2 bytes wide. This model ignores the effects of contention, which are probably not too serious in the parallel benchmarks we examine, with the possible exception of FFT, which uses all-to-all communication. Contention could have a serious performance impact in other work loads.

Characteristic	Number of processor clock cycles
Cache hit	1
Cache miss to local memory	$25 + \frac{\text{block size in bytes}}{8}$
Cache miss to remote home directory	$75 + \frac{\text{block size in bytes}}{2}$
Cache miss to remotely cached data (3-hop miss)	$100 + \frac{\text{block size in bytes}}{2}$

FIGURE 8.30 Characteristics of the example directory-based machine. Misses can be serviced locally (including from the local directory), at a remote home node, or using the services of both the home node and another remote node that is caching an exclusive copy. This last case is called a 3-hop miss and has a higher cost because it requires interrogating both the home directory and a remote cache. Note that this simple model does not account for invalidation time. These network latencies are typical of what can be achieved in 1995–96 in an MPP-style network interfaced in hardware to each node and assuming moderately fast processors (150–200 MHz).