Lecture 17: Introduction to Multiprocessors

Professor David A. Patterson

Computer Science 252

Fall 1996

Review: Networking

- Protocols allow hetereogeneous networking
  - Protocols allow operation in the presense of failures
  - Internetworking protocols used as LAN protocols => large overhead for LAN
- Integrated circuit revolutionizing networks as well as processors
  - Switch is a specialized computer
  - Faster networks and slow overheads violate of Amdahl’s Law

Review: Phil Karn visit

- Wireless means we'll always need protocols to guarantee reliable communication
  - Advantages when portable, remote site, or broadcast
- An Internetworking Protocol (IP) will be required because we'll never have single LAN/WAN network
  - >1 for office to office, city to city, nation to nation
- Telephony has dominated WAN decisions
  - will the Internet dominate WAN in the future?
- Internet continue as flat access fee? Local calls too?
- Cable provides interesting opportunities
  - Compete with local companies for local phone calls?
  - Direct Broadcast Satelites free cable bandwidth?
  - Broadcast popular WWW pages? Locality of WWW?

Parallel Computers

- Definition: "A parallel computer is a collection of processiong elements that cooperate and communicate to solve large problems fast."

  Almasi and Gottlieb, *Highly Parallel Computing* ,1989

- Questions about parallel computers:
  – How large a collection?
  – How powerful are processing elements?
  – How do they cooperate and communicate?
  – How are data transmitted?
  – What type of interconnection?
  – What are HW and SW primitives for programmer?
  – Does it translate into performance?

Parallel Processors Religion

- The dream of computer architects for 30 years: replicate processors to add performance vs. design a faster processor
- Led to innovative organization tied to particular programming models since uniprocessors can't keep going
  - e.g., uniprocessors must stop getting faster due to limit of speed of light: 1972,…, 1989
  - Borders religious fervor at times: you must believe!
  - Fervor damped some when companies went out of business: Thinking Machines, Kendall Square, ...
- Argument instead is the "pull" of the opportunity of scalable performance vs. the "push" of uniprocessor plateau

Opportunities: Scientific Computing

- Nearly Unlimited Demand (Grand Challenge):

App	*Perf (GFLOPS)*	*Memory (GB)*
48 hour weather	**0.1**	**0.1**
72 hour weather	**3**	**1**
Pharmaceutical design	**100**	**10**
Global Change, Genome	**1000**	**1000**

- **Success in real industries:**
  - Petrolium: reservoir modeling
  - Automotive: crash simulation, drag analysis, engine
  - Aeronautics: airflow analysis, engine, structural mechanics
  - Pharmaceuticals: molecular modeling
  - Entertainment: full length movies (“Toy Story”)

Example: Scientific Computing

- Molecular Dynamics on Intel Paragon with 128 processors
  - (see Chapter 1, Figure 1-3, page 22 of Culler, Sighn, Gupta [CSG96])
- Improve over time: load balancing, other
- 128 processor Intel Paragon = 406 MFLOPS
- C90 vector = 145 MFLOPS
  (or ≈ 45 Intel processors)

Opportunities: Commercial Computing

- •Transaction processing & TPC-C bencmark
  - – (see Chapter 1, Figure 1-4, page 23 of [CSG96])
  - – small scale parallel processors to large scale
- •Througput (Transactions per minute) vs. Time

•Speedup:	1	4	8	16	32	64	112
IBM RS6000	735	1438	3119				
	1.00	*1.96*	*4.24*				
Tandem Himilaya				3043	6067	12021	20918
				1.00	*1.99*	*3.95*	*6.87*

  - – IBM performance hit 1=>4, good 4=>8
  - – Tandem scales: 112/16 = 7.0
- •Others: eletronic CAD simulation, multiple processes

What level Parallelism?

- Bit level parallelism: 1970 to ≈1985
  – 4 bits, 8 bit, 16 bit, 32 bit microprocessors
- Instruction level parallelism (ILP): 1985 through today
  – Pipelining
  – Superscalar
  – VLIW
  – Out-of-Order execution
  – Limits to benefits?
- Process Level or Thread level parallelism???
  – Servers are parallel (see Fig. 1-9, p. 32 of [CSG96])
  – Highend Desktop dual processor PC soon? (or the sell the socket?)

Whither Supercomputing?

- **Linpack (dense linear algebra) for Vector Supercomputers vs. Microprocessors**
- **“Attack of the Killer Micros”**
  - **(see Chapter 1, Figure 1-11, page 34 of [CSG96])**
  - **100 x 100 vs. 1000 x 1000**
- **MPPs vs. Supercomputers when rewrite linpack to get peak performance**
  - **(see Chapter 1, Figure 1-12, page 35 of [CSG96])**
- **500 fastest machines in the world: parallel vector processors (PVP), bus based shared memory (SMP), and MPPs**
  - **(see Chapter 1, Figure 1-13, page 36 of [CSG96])**

CS 252 Administrivia

- Homework on Chapter 7 due Monday 11/4 at 5 PM in 252 box, done in pairs:
  - Exercises 7.1, 7.3, 7.10
- Next reading is Chapter 8 of CA:AQA 2/e and Sections 1.1-1.4, Chapter 1 of upcoming book by Culler, Singh, and Gupta:

`www.cs.berkeley.edu/~culler/`

- Remzi Arpaci will talk Fri. 11/8 on Networks of Workstations and world record sort
- Dr. Dan Lenowski, architect of SGI Origin, talk in Systems Seminar Thur. 11/14 at 4PM in 306 Soda
- Next project review: survey due Mon. 11/11; 20 min. meetings moved to Fri. 11/15; signup Wed. 11/6

Parallel Architecture

- Parallel Architecture extends traditional computer architecture with a communication architecture
  - abstractions (HW/SW interface)
  - organizational structure to realize abstraction efficiently

Parallel Framework

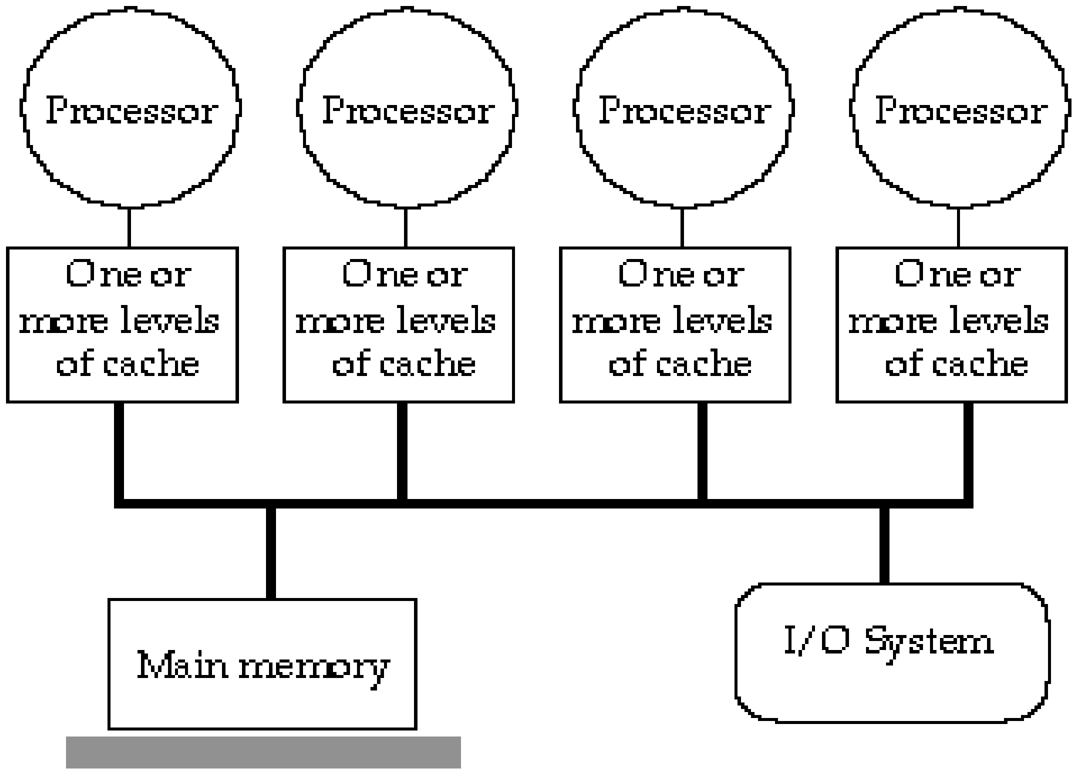
- **Layers:**
  - **(see Chapter 1, Figure 1-14, page 37 of [CSG96])**
  - **Programming Model:**
    - » **Multiprogramming : lots of jobs, no communication**
    - » **Shared address space: communicate via memory**
    - » **Message passing: send and recieve messages**
    - » **Data Parallel: several 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, recieve library calls**
    - » **Debate over this topic (ease of programming, scaling) => many hardware designs 1:1 programming model**

Shared Address/Memory Multiprocessor Model

- Communicate via Load and Store
  - Oldest and most popular model
- Based on timesharing: processes on multiple processors vs. sharing single processor
- process: a virtual address space and ≥ 1 thread of control
  - Multiple processes can overlap (share), but ALL threads share a process address space
- Writes to shared address space by one thread are visible to reads of other threads
  - Usual model: share code, private stack, some shared heap, some private heap

Small-Scale MP Designs

- **Memory: centralized with uniform access time ("uma") and bus interconnect**
- **Examples: Sun Enterprise 5000 , SGI Challenge, Intel SystemPro**

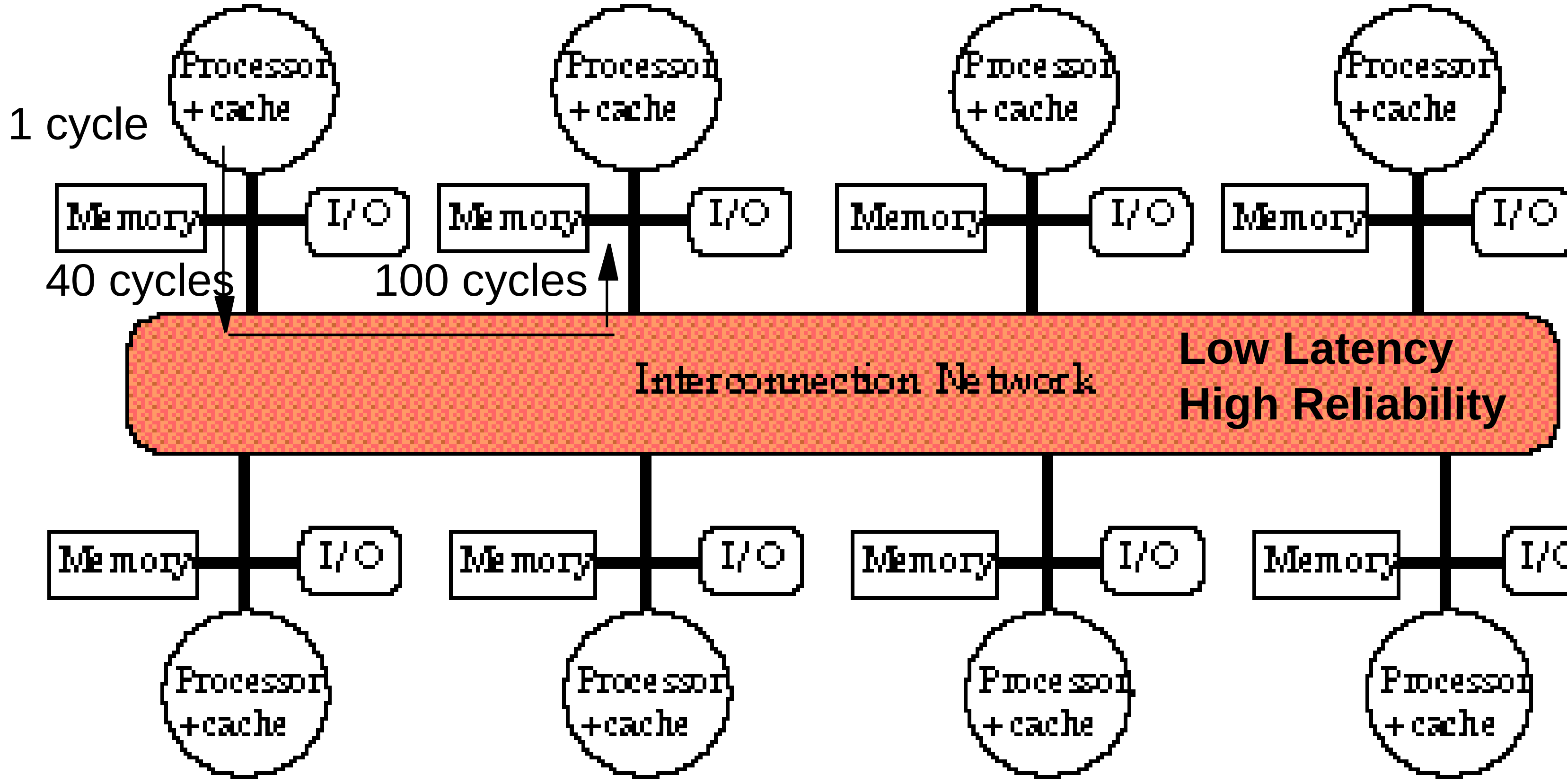

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O System

SMP Interconnect

- Processors to Memory AND to I/O
- Bus based: all memory locations equal access time so SMP = “Symmetric MP”
  - Sharing limited BW as add processors, IO
  - (see Chapter 1, Figs 1-18/19, page 42-43 of [CSG96])
- Crossbar: expensive to expand
- Multistage network (less expensive to expand than crossbar with more BW)
- “Dance Hall” designs: All processors on the left, all memories on the right

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])**



Shared Address Model Summary

- **Each processor can name every physical location in the machine**
- **Each process can name all data it shares with other processes**
- **Data transfer via load and store**
- **Data size: byte, word, ... or cache blocks**
- **Uses virtual memory to map virtual to local or remote physical**
- **Memory hierarchy model applies: now communication moves data to local processor cache (as load moves data from memory to cache)**
  - **Latency, BW, scalability when communicate?**

Message Passing Model

- Whole computers (CPU, memory, I/O devices) communicate as explicit I/O operations
  - Essentially NUMA but integrated at I/O devices vs. memory system
- Send specifies local buffer + receiving process on remote computer
- Receive specifies sending process on remote computer + local buffer to place data
  - Usually send includes process tag and receive has rule on tag: match 1, match any
  - Synch: when send completes, when buffer free, when request accepted, receive wait for send
- Send+receive => memory-memory copy, where each each supplies local address, AND does pairwise sychronization

Message Passing Model

- Send+receive => memory-memory copy, sychronization on OS even on 1 processor
- History of message passing:
  - Network topology important because could only send to immediate neighbor
  - Typically synchronouns, blocking send & receive
  - Later DMA with non-blocking sends, DMA for receive into buffer until processor does receive, and then data is tranfered to local memory
  - Later SW libraries to allow arbitrary communication
- Example: IBM SP-2, RS6000 workstations in racks
  - Network Inteface Card has Intel 960
  - 8X8 Crossbar wtich building block
  - 40 MByte/sec per link

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 model on either HW base

Flynn Categories

- SISD (Single Instruction Single Data)
  - Uniprocessors
- MISD (Multiple Instruction Single Data)
  - ???
- SIMD (Single Instruction Multiple Data)
  - Examples: Illiac-IV, CM-2
    - » Simple programming model
    - » Low overhead
    - » Flexibility
    - » All custom integrated circuits
- MIMD (Multiple Instruction Multiple Data)
  - Examples: Sun Enterprise 5000, Cray T3D, SGI Origin
    - » Flexible
    - » *Use off-the-shelf micros*

Data Parallel Model

- Operations can be performed in parallel on each element of a large regular data structure, such as an array
- 1 Control Processsor broadcast to many PEs (see Ch. 1, Figs 1-26, page 51 of [CSG96])
  - When computers were large, could amortize the control portion of many replicated PEs
- Data distributed in each memory
- Condition flag per PE so that can skip
- Early 1980s VLSI => SIMD rebirth: 32 1-bit PEs + memory on a chip was the PE
- Data parallel programming languages lay out data to processor

Data Parallel Model

- Vector processors have similar ISAs, but no data placement restriction
- Advancing VLSI led to single chip FPUs and whole fast µProcs
- SIMD programming model led to Single Program Multiple Data (SPMD) model
  - All processors execute identical program
- Data parallel programming languages still useful, do communication all at once: "Bulk Synchronous" phases in which all communicate after a global barrier

Convergence in Parallel Architecture

- Complete computers connected to scalable network via communication assist
  - (see Ch. 1, Fig. 1-29, page 57 of [CSG96])
- Different programming models place different requirements on communication assist
  - Shared address space: tight integration with memory to capture memory events that interact with others + to accept requests from other nodes
  - Message passing: send messages quickly and respond to incoming messages: tag match, allocate buffer, transfer data, wait for receive posting
  - Data Parallel: fast global synchronization
- HPF shared-memory, data parallel; PVM, MPI message passing libraries; both work on many machines, different implementations

Fundamental Issues: 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**
- **Choice of naming affects replication of data; via load in cache memory hierachy or via SW replication and consistency**

Fundamental Issues: Naming

- Global physical address space: any processor can generate and 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: if locations are named <process number, address> uniformly for all processes of the parallel program

Fundamental Issues: Synchronization

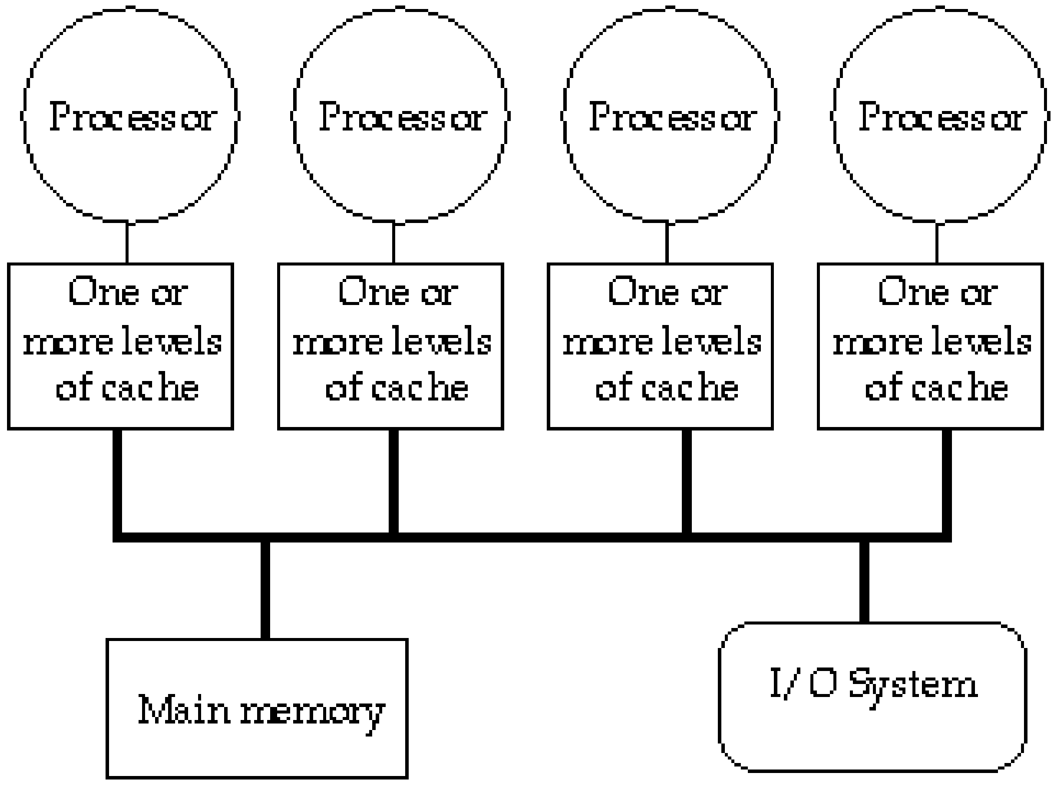
- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address => additional operations to explicitly coordinate: e.g., write a flag, awaken a thread, interrupt a processor

Fundamental Issues: Latency and Bandwidth

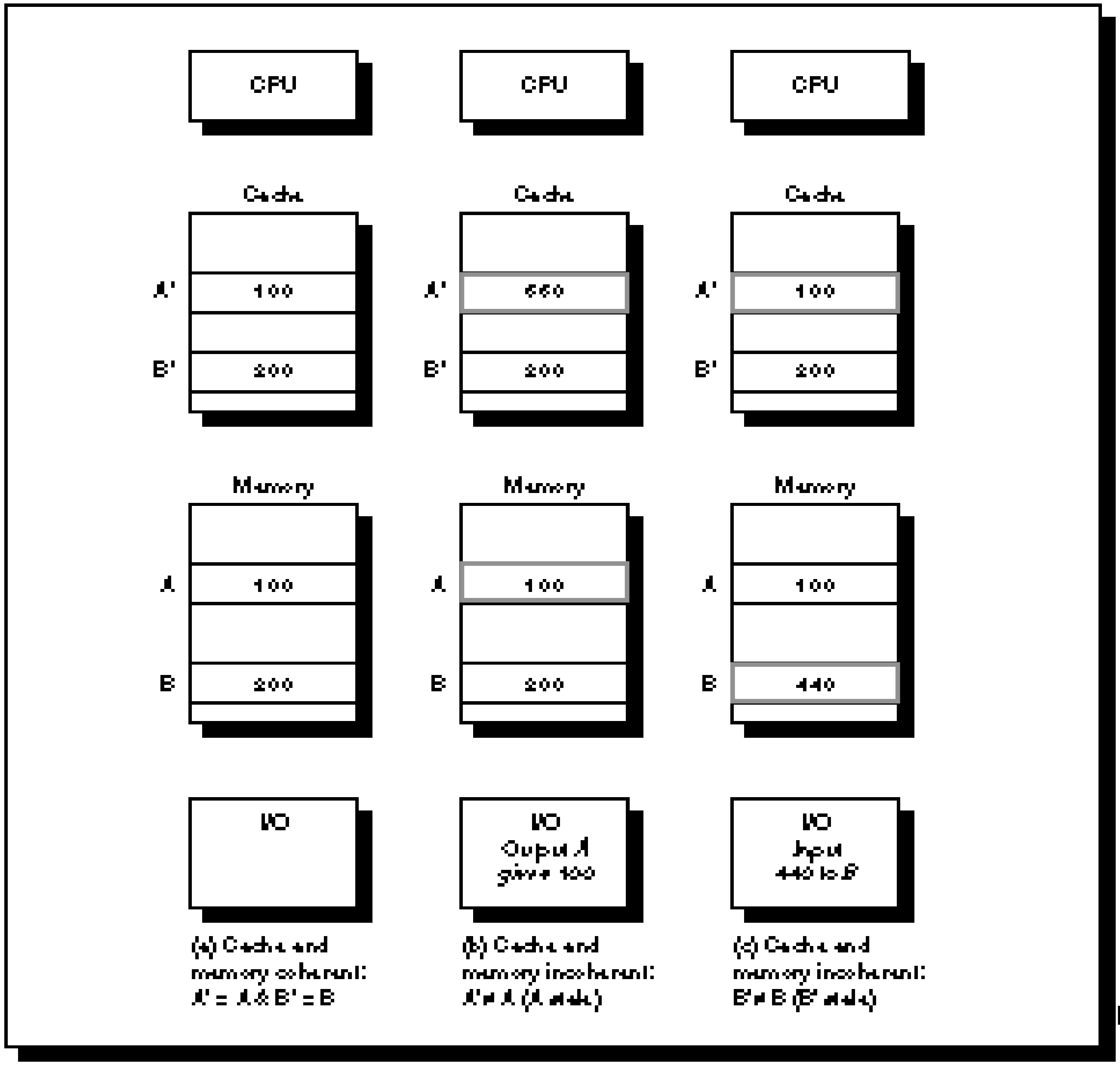
- **Bandwidth**
  - Need high bandwidth in communication
  - Cannot scale, but stay close
  - Make limits in network, memory, and processor
  - Overhead to communicate is a problem in many machines
- **Latency**
  - Affects performance, since processor may have to wait
  - Affects ease of programming, since requires more thought to overlap communication and computation
- **Latency Hiding**
  - How can a mechanism help hide latency?
  - Examples: overlap message send with computation, prefetch

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?**


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O System

The Problem of Cache Coherency



What Does Coherency Mean?

- Informally:
  - Any read must return the most recent write
  - Too strict and very difficult to implement
- Better:
  - Any write must eventually be seen by a read
  - All writes are seen in order (“serialization”)
- Two rules to ensure this:
  - If P writes x and P1 reads it, P’s write will be seen 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
    - » Otherewise could see writes in illogical order (could see older value after a newer value)

Potential 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 (avoids bottlenecks)
  - Send point-to-point requests to processors
  - Scales better than Snoop
  - Actually existed BEFORE Snoop-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:**
  - **Write to shared data: broadcast on bus, processors snoop, and *update* copies**
  - **Read miss: memory is always up-to-date**
- **Write serialization: bus serializes requests**
  - **Bus is single point of arbitration**

Basic Snoopy Protocols

- **Write Invalidate versus Broadcast:**
  - Invalidate requires one transaction per write-run
  - Invalidate uses spatial locality: one transaction per block
  - Broadcast has lower latency between write and read
  - Broadcast: BW (increased) vs. latency (decreased)

Name	Protocol Type	Memory-write policy	Machines using
Write Once	Write invalidate	Write back after first write	First snoopy protocol.
Synapse N+1	Write invalidate	Write back	1st cache-coherent MPs
Berkeley	Write invalidate	Write back	Berkeley SPUR
Illinois	Write invalidate	Write back	SGI Power and Challenge
“Firefly”	Write broadcast	Write back private, Write through shared	SPARCCenter 2000

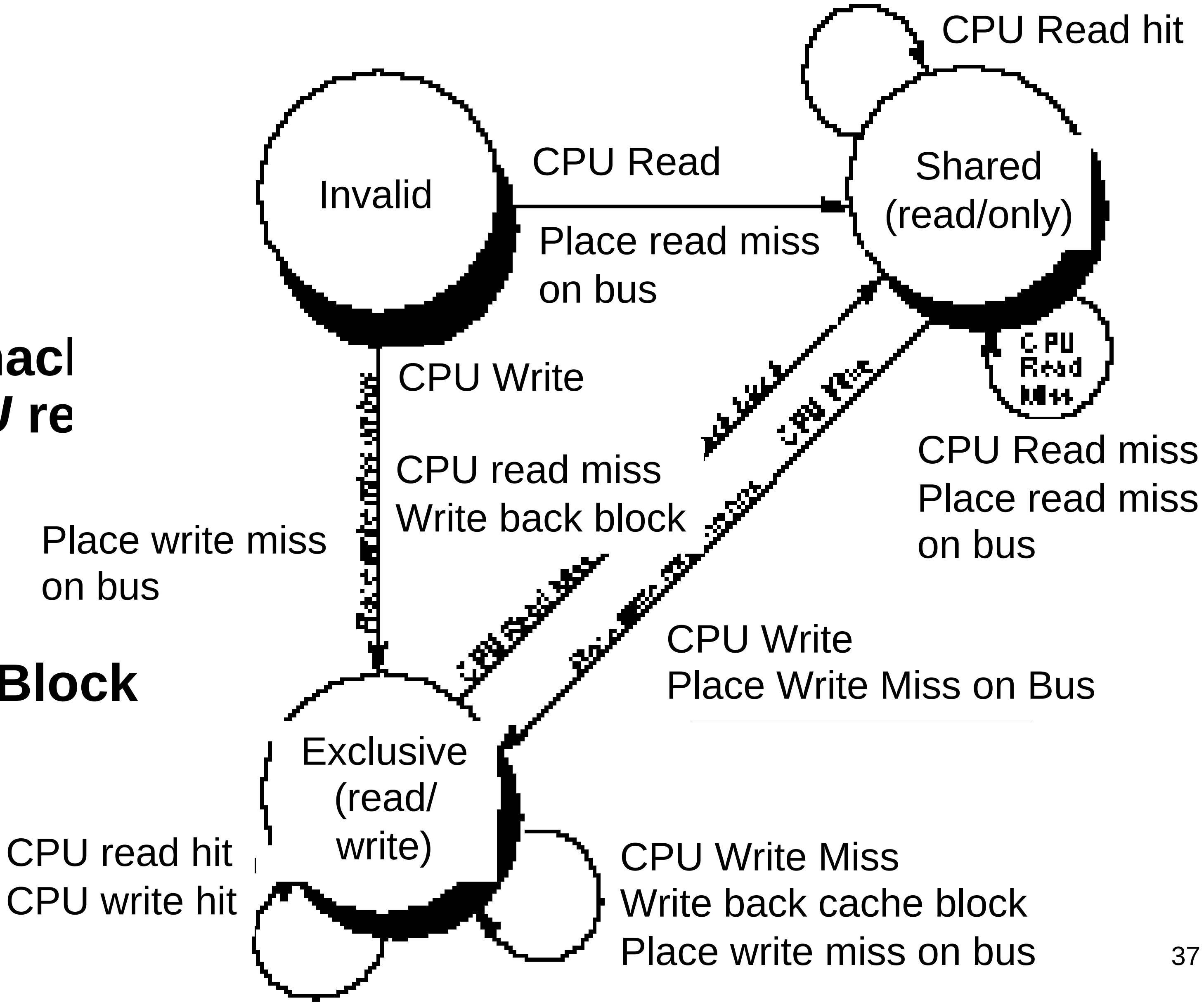
An Example Snoopy Protocol

- Invalidation protocol, write-back cache
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory
  - OR Dirty in exactly one cache
  - OR Not in any caches
- Each cache block is in one state:
  - 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
- Writes to clean line are treated as misses

Snoopy-Cache State Machine-I

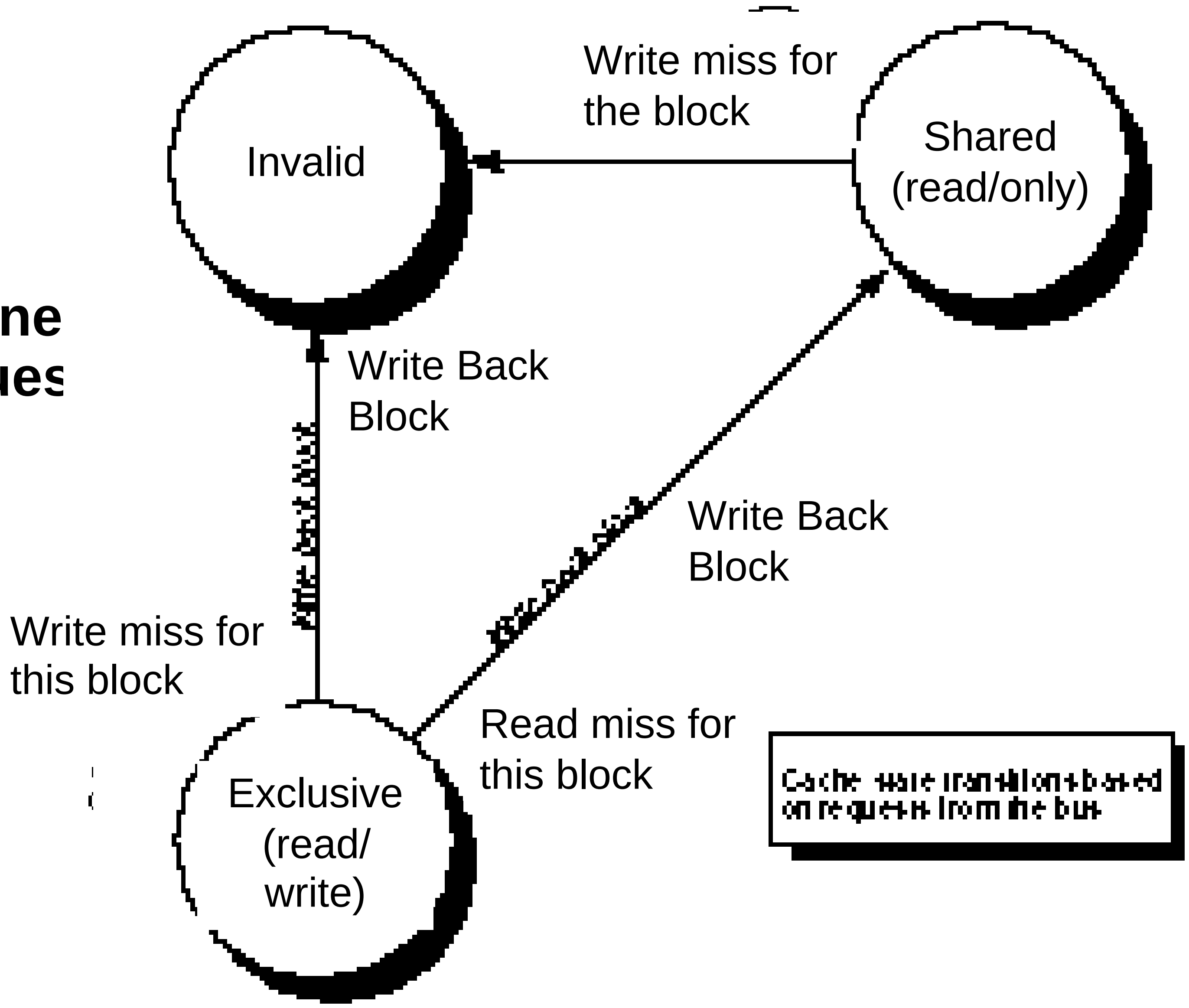
- **State macl**
  for *CPU* re

Cache Block State

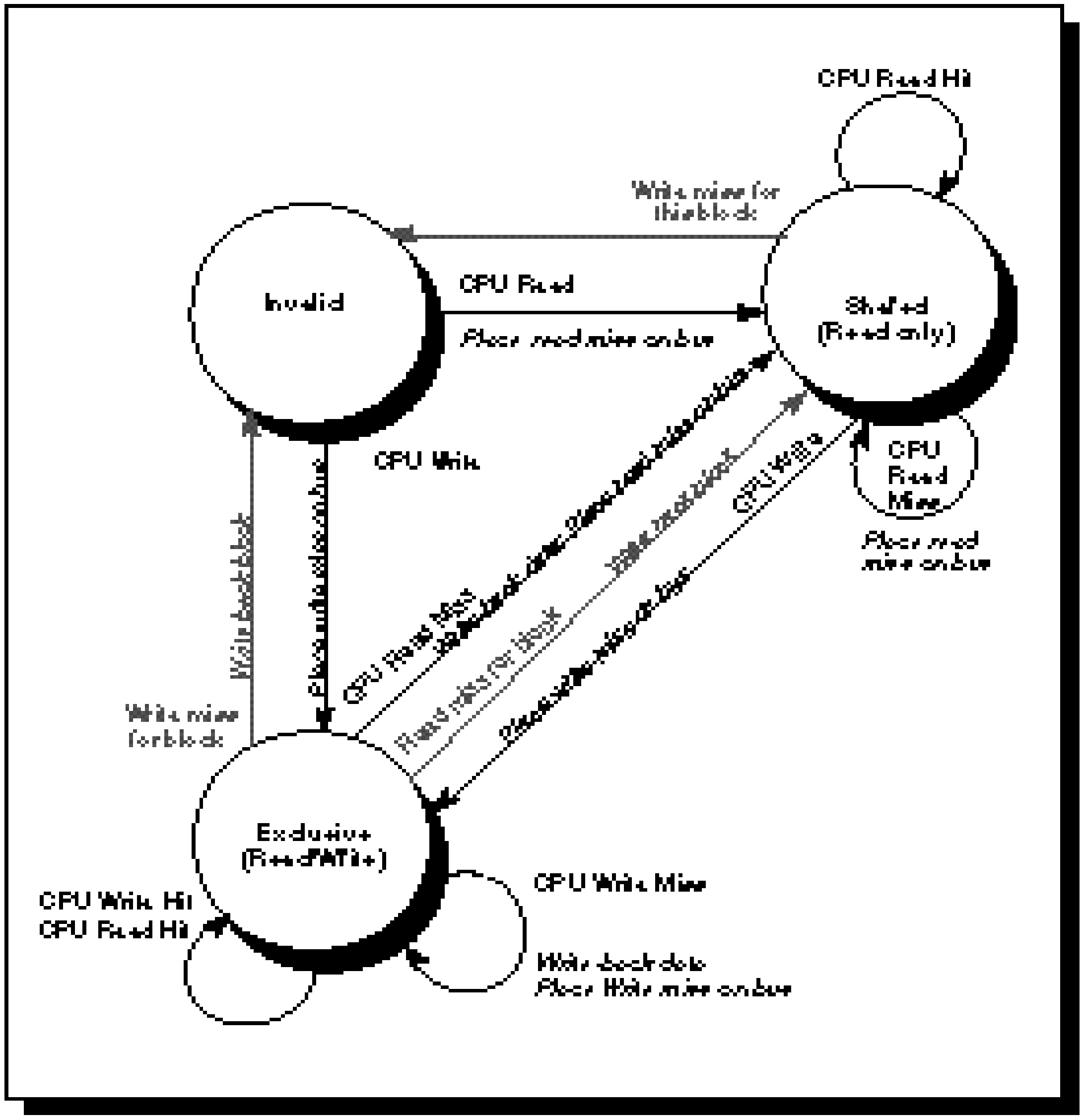


Snoopy-Cache State Machine-II

- **State machine for *bus* reques**



Snoop Cache: State Machine



Extensions:

- Fourth State: Ownership
- Clean-> dirty, need invalidate only (upgrade request) **Berkeley Protocol**
- Clean exclusive state (no miss for private data on write) **Illinois Protocol**

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

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		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 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		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							WrMs	P2	A2			10
				Excl.	A2	40	WrBk	P2	A1	20		20

Assumes A1 and A2 map to same cache block

Implementation Complications

- **Write Races:**
  - **Cannot update cache until bus is obtained**
    - » **Otherwise, another processor may get bus first, and 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**

Snoop Cache Variations

Berkeley Protocol	Basic Protocol	Illinois Protocol
Owned Exclusive	Exclusive	Private Dirty
Owned Shared	Shared	Private Clean
Shared	Invalid	Shared
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

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**

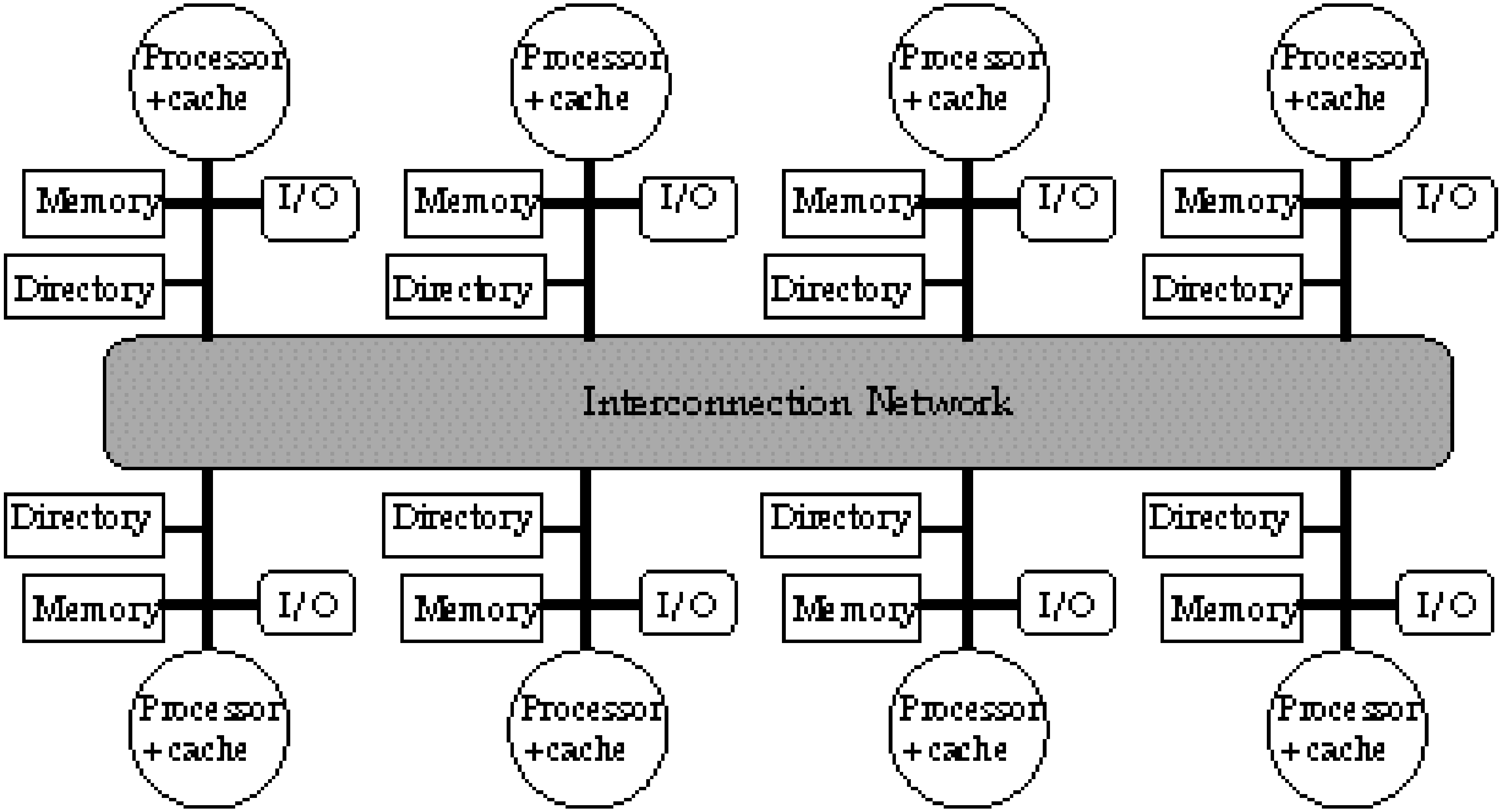
Implementing Snooping Caches

- Bus serializes writes, getting bus ensures no one else can perform 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
- Since every bus transaction checks cache tags, could interfere with CPU just to check: solution is a duplicate set of tags just to allow checks in parallel with CPU or second level cache that obeys inclusion

Larger MPs

- Separate Memory per Processor
- Local or Remote access via memory controller
- 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 *ƒ*(memory size) vs. *ƒ*(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



Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached
  - 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
- Terms:
  - Local node is the node where a request originates
  - Home node is the node where the memory location of an address resides
  - Remote node is the node that has a copy of a cache block, whether exclusive or shared.

Directory Protocol Messages

Message type	*Source*	*Destination*	*Msg*
Read miss	**Local processor**	**Home directory**	**P, A**
– ***Processor P reads data at address A; send data and make P a read sharer***			
Write miss	**Local processor**	**Home directory**	**P, A**
– ***Processor P writes data at address A; send data and make P the exclusive owner***			
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***			
Data write-back	**Remote cache**	**Home directory**	**A, Data**

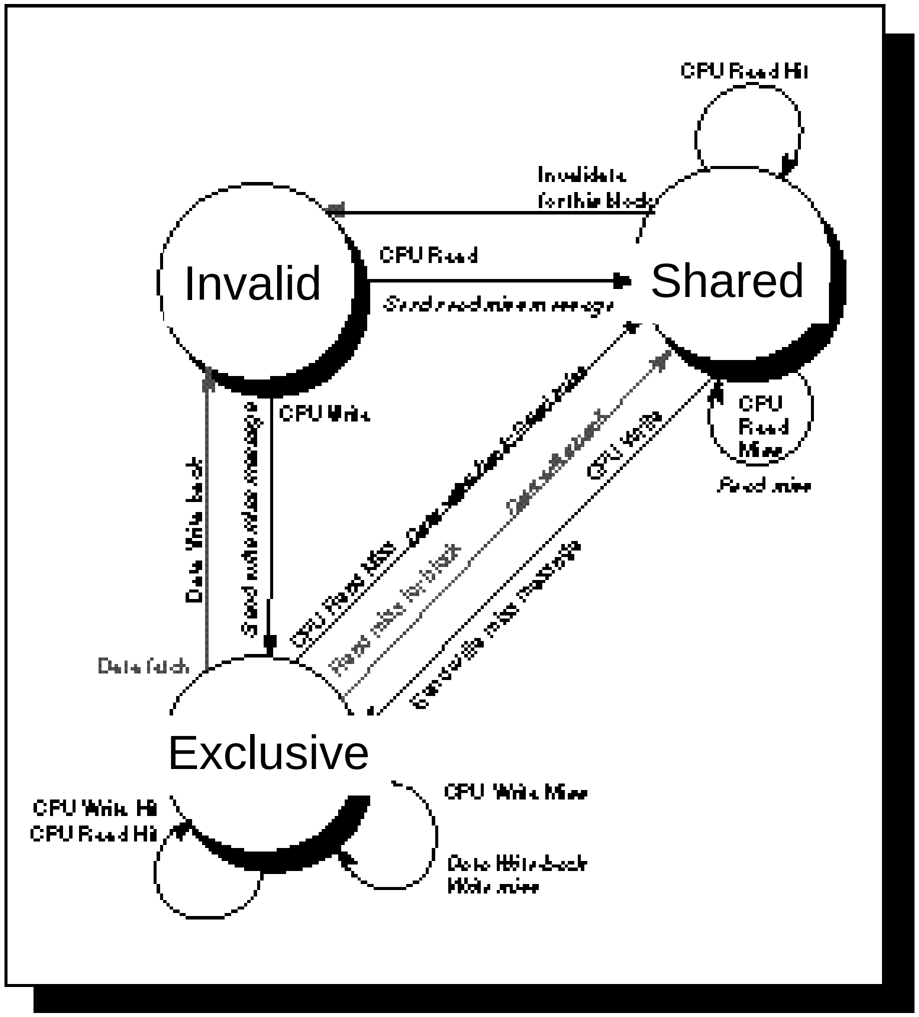
Example Directory Protocol

- 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 back the data from memory and the requestor is the only sharing node. The state of the block is made Shared.
  - Write miss: requesting processor is sent the value and 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

Example Directory Protocol

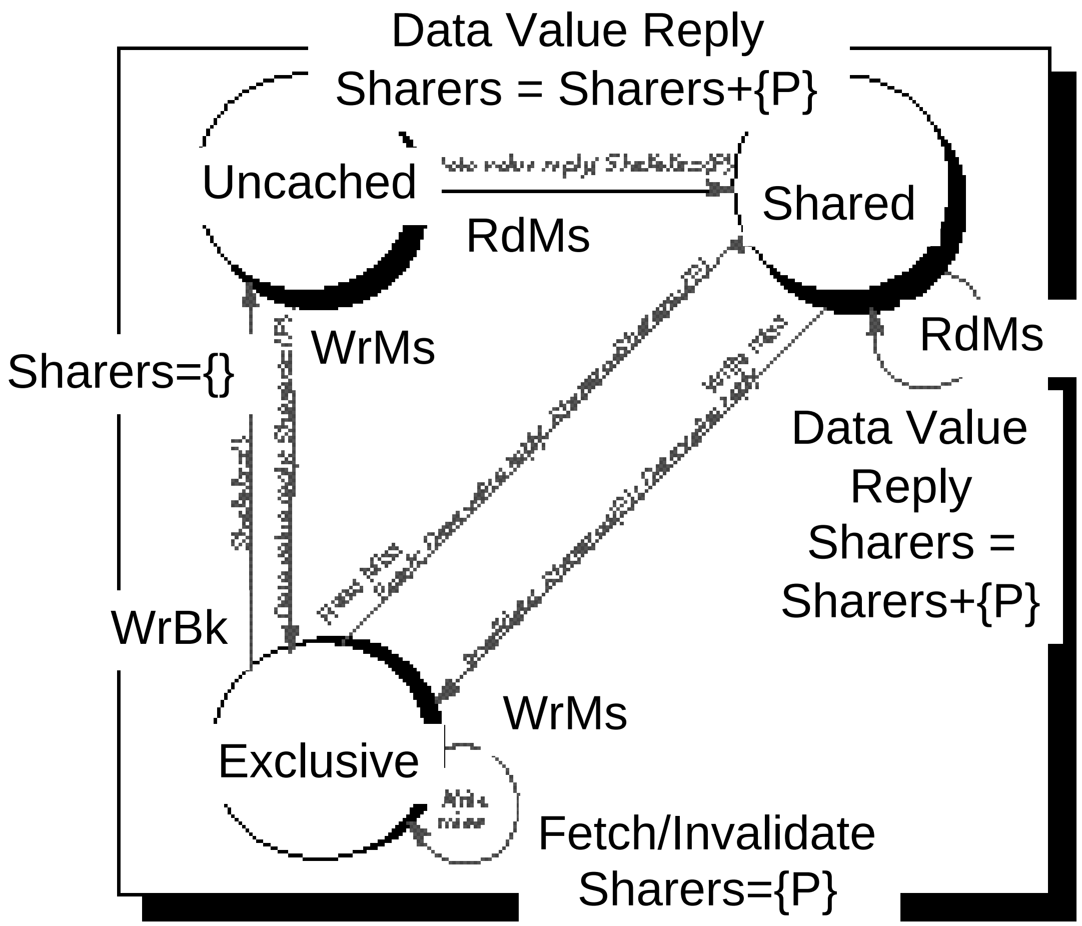
- Block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) & three possible directory requests:
  - Read miss: owner processor is sent a data fetch message, which causes state of block in owner's cache to transition to Shared and causes owner to send data to directory, where it is written to memory and sent back to the 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).
  - Data write-back: owner processor is replacing the block and hence must write it back. This makes the 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

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



- The states are identical to those in the snoopy case, and the transactions are very similar with explicit invalidate and write-back requests replacing the write misses that were formerly broadcast on the bus.

State Transition Diagram for the Directory



- **The same states and structure as the transition diagram for an individual cache**
  - **All actions are in color since they all are externally caused. Italics indicates the action taken the directory in response to the request. Bold italics indicate an action that updates the sharing set, Sharers, as opposed to sending a message.**

Example

	P1			*P2*			*Bus*				*Directory*			*Memor*
step	*State*	*Addr*	*Valu*	*State*	*Add*	*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

	P1			P2			Bus				Directory			Memor
step	State	Addr	Valu	State	Add	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

	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														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

Example

	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														10
														10
P2: Write 40 to A2														10

A1 and A2 map to the same cache block

Example

	P1			P2			Bus				Directory			Memor
step	State	Addr	Valu	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														10

A1 and A2 map to the same cache block

Example

	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