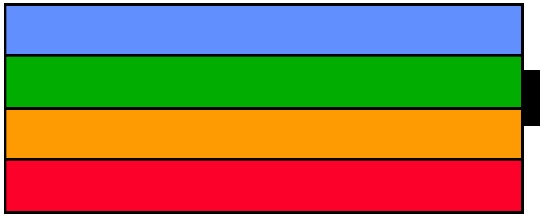
Lecture 18: Snooping vs. Directory Based Coherency

Professor David A. Patterson

Computer Science 252

Fall 1996

Review: Parallel Framework

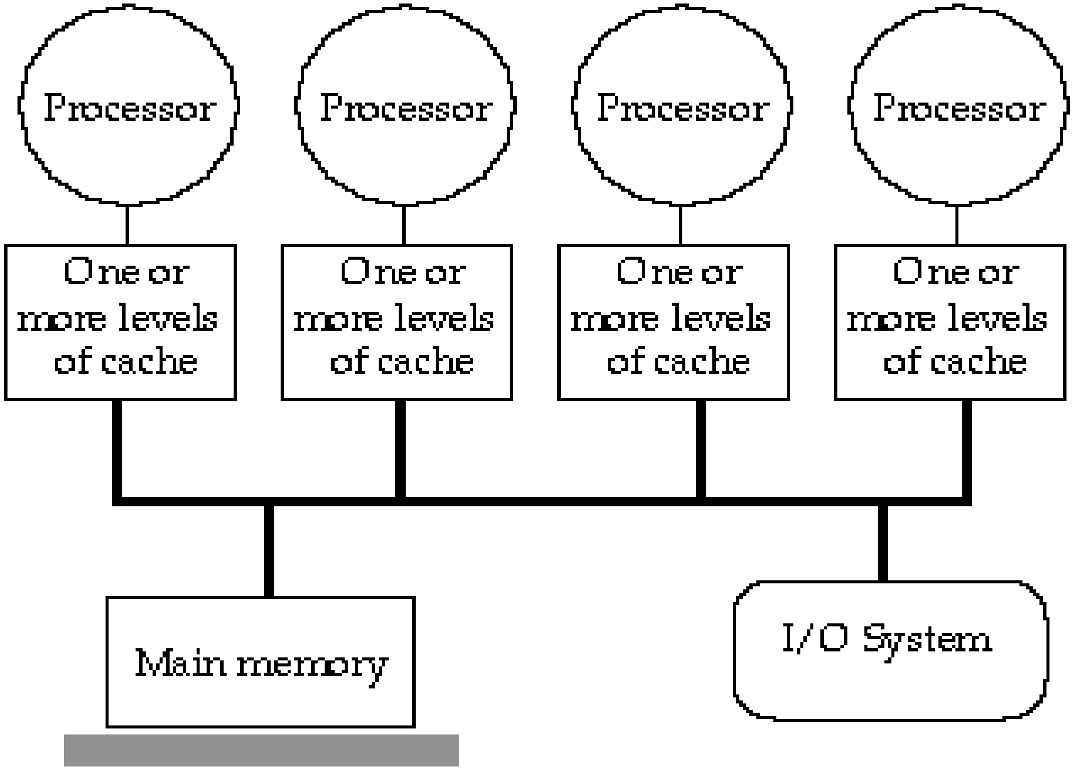


Programming Model
Communication Abstraction
Interconnection SW/OS
Interconnection HW

- Layers:
  - 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

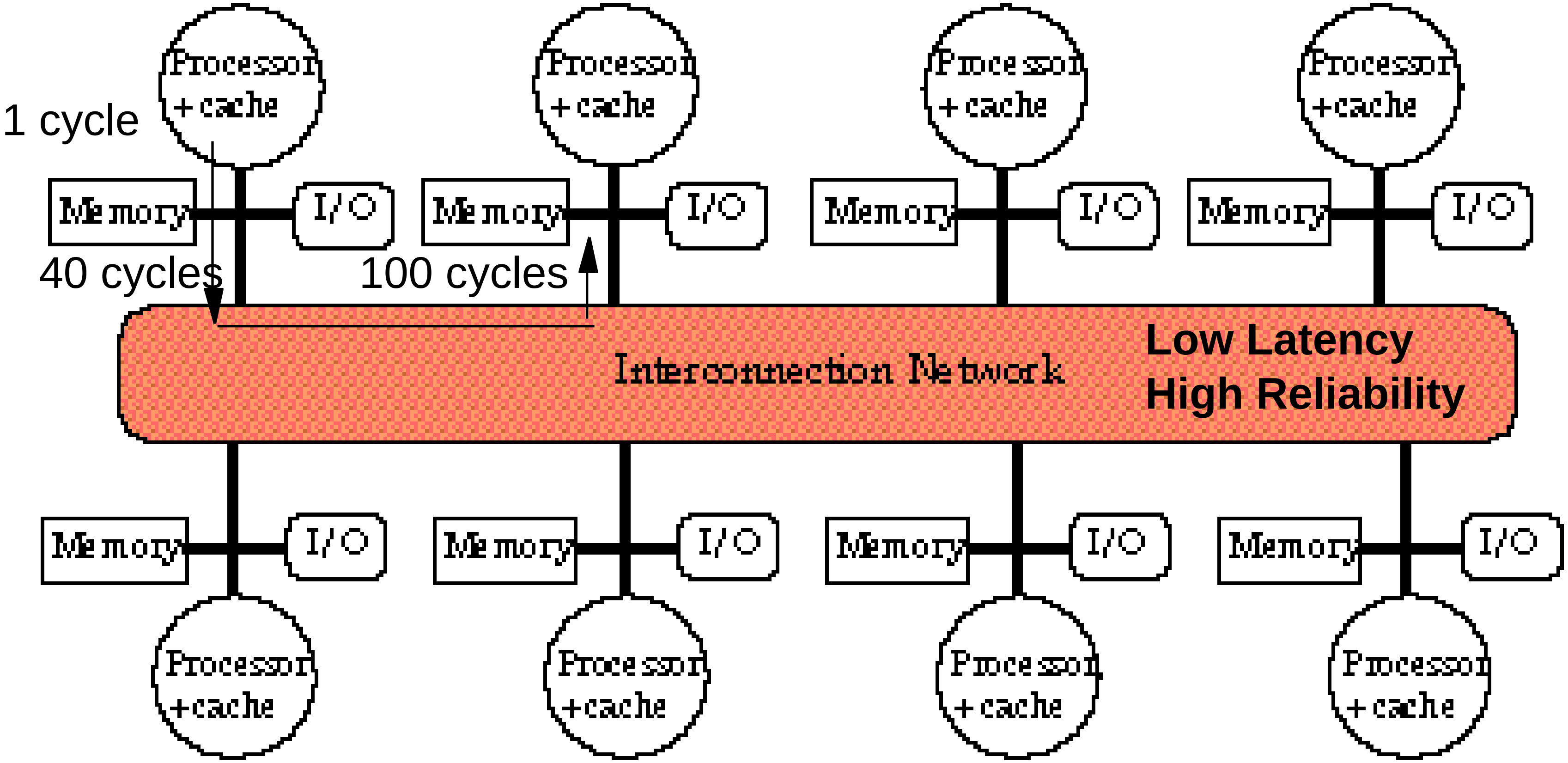
Review: Small-Scale MP Designs

- **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

Review: 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])**



1 cycle

40 cycles

100 cycles

Low Latency
High Reliability

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

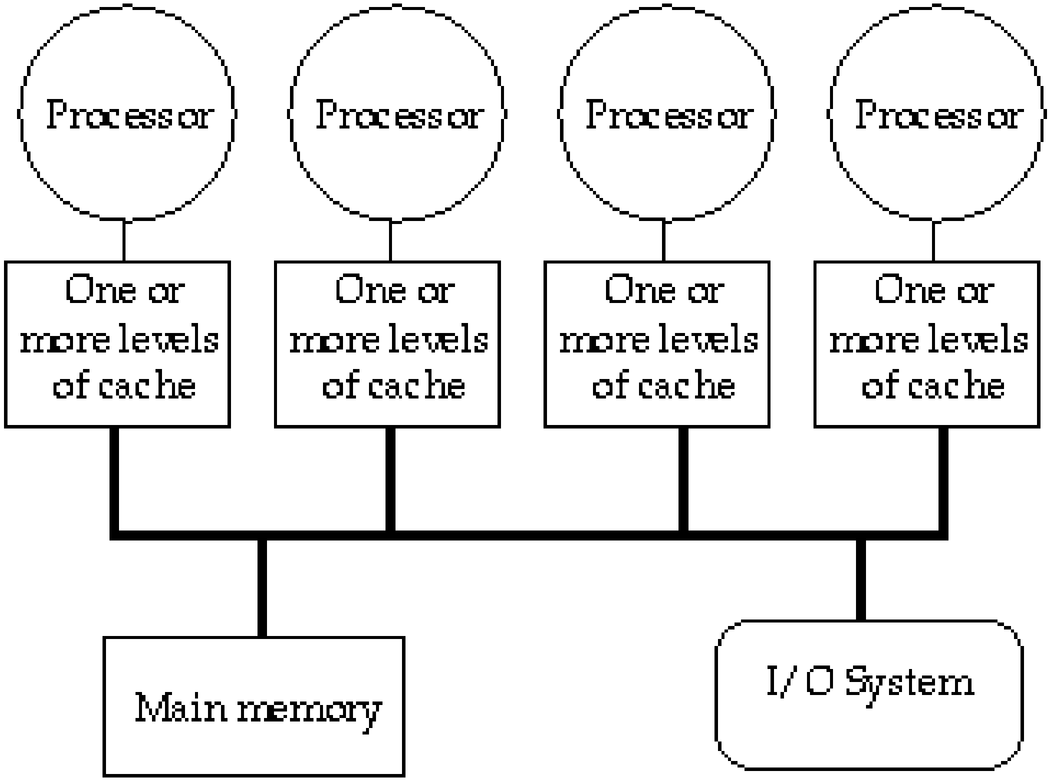
- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address => additional operations to explicitly coordinate: e.g., write a flag, awaken a thread, interrupt a processor

Fundamental Issues: Latency and Bandwidth

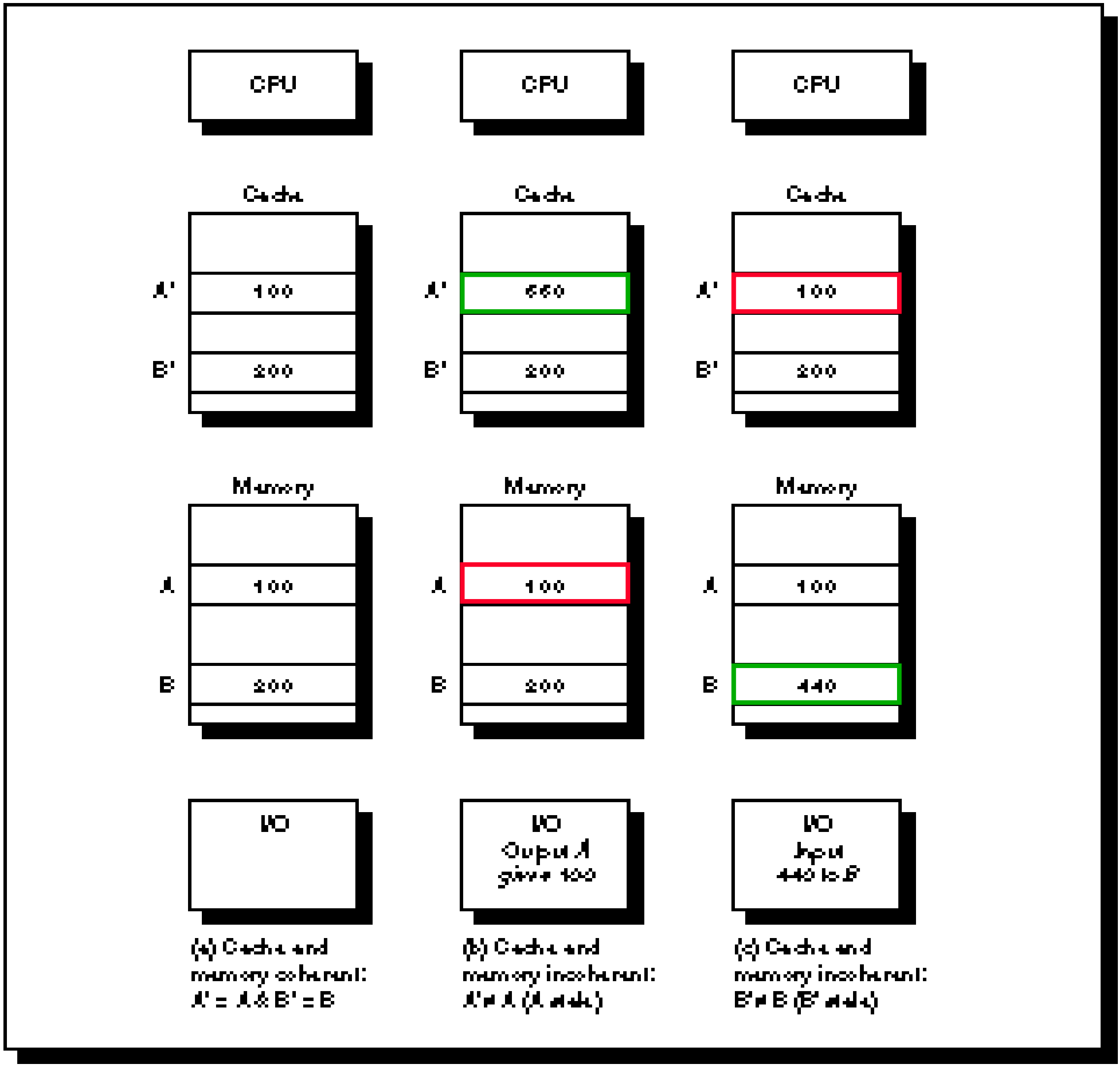
- Bandwidth
  - Need high bandwidth in communication
  - Cannot scale, but stay close
  - Make limits in network, memory, and processor match
  - Overhead to communicate is a problem in many machines
- Latency
  - Affects performance, since processor may have to wait
  - Affects ease of programming, 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



(a) Cache and memory coherent: A' = A & B' = B

(b) Cache and memory incoherent: A' ≠ A (A stale)

(c) Cache and memory incoherent: B' ≠ B (B' stale)

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 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**
    - » **Otherewise could see writes in illogical order (could see older value after a newer value)**

CS 252 Administrivia

- 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

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
MESI	Write invalidate	Write back	Pentium, PowerPC

Snoop Cache Variations

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

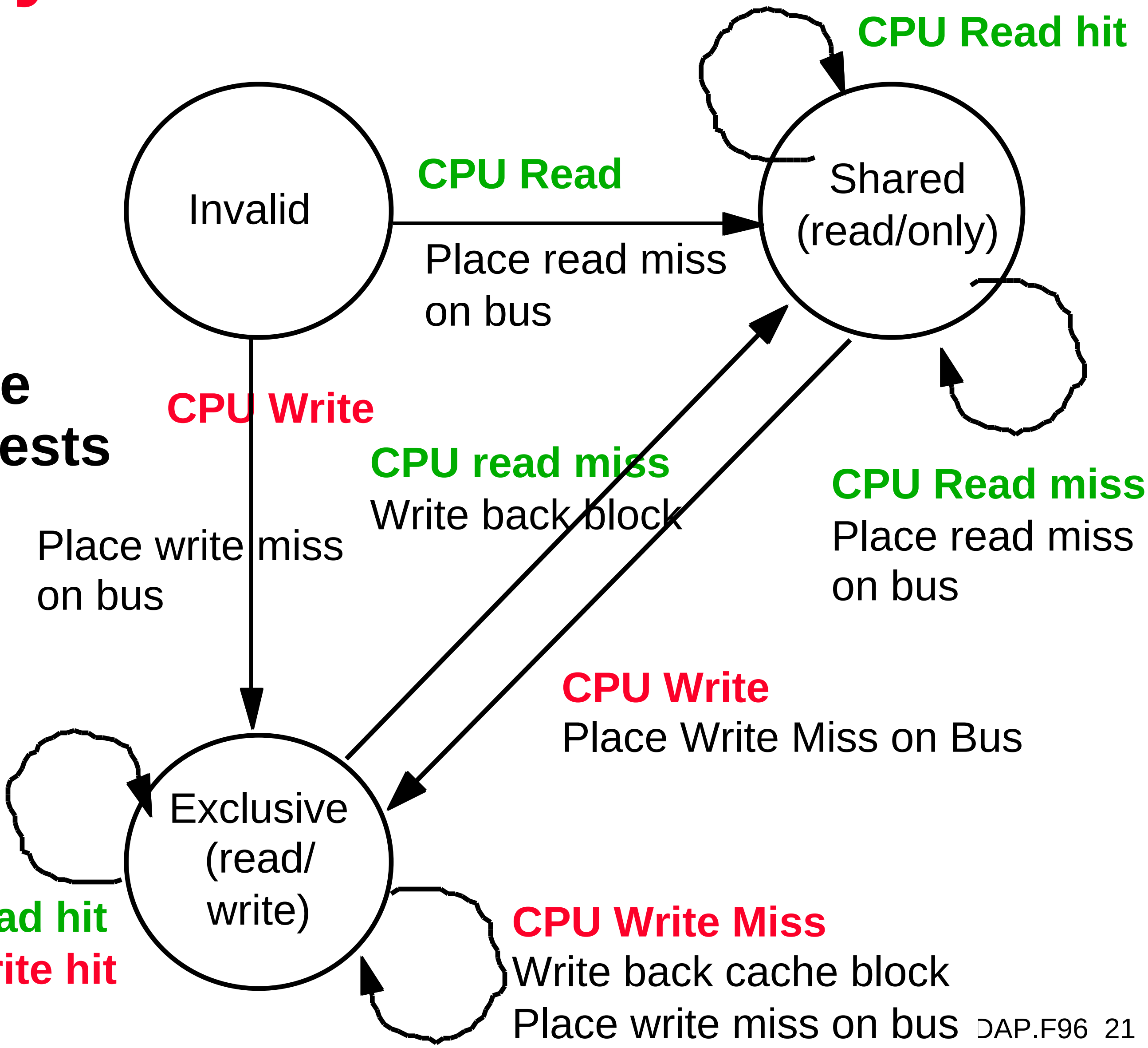
An Example Snoopy Protocol

- 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:
  - 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**

Cache Block State



Invalid

CPU Read

Place read miss on bus

Shared (read/only)

CPU Read hit

CPU Write

Place write miss on bus

CPU read miss

Write back block

CPU Read miss

Place read miss on bus

CPU Write

Place Write Miss on Bus

Exclusive (read/ write)

CPU read hit

CPU write hit

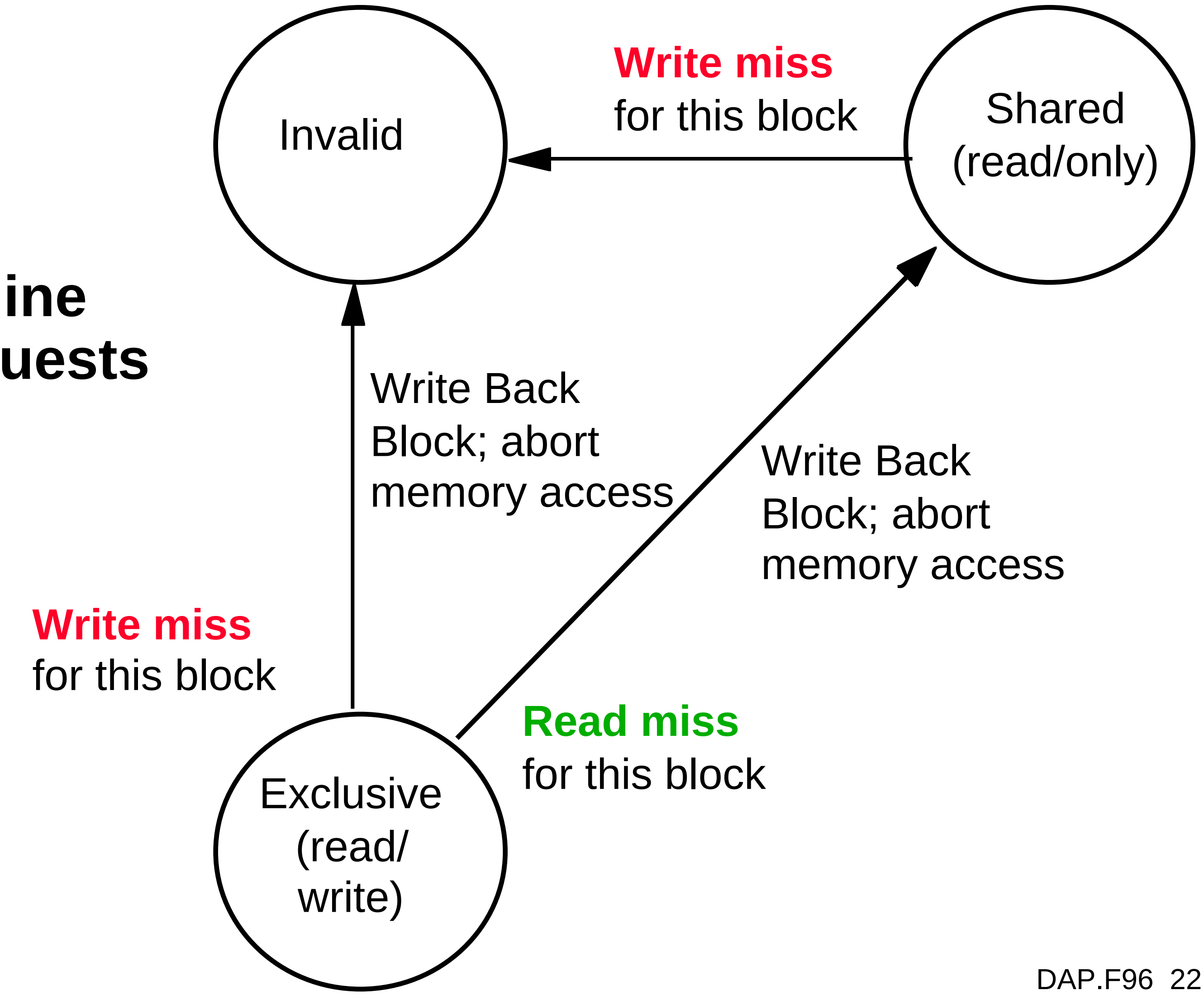
CPU Write Miss

Write back cache block

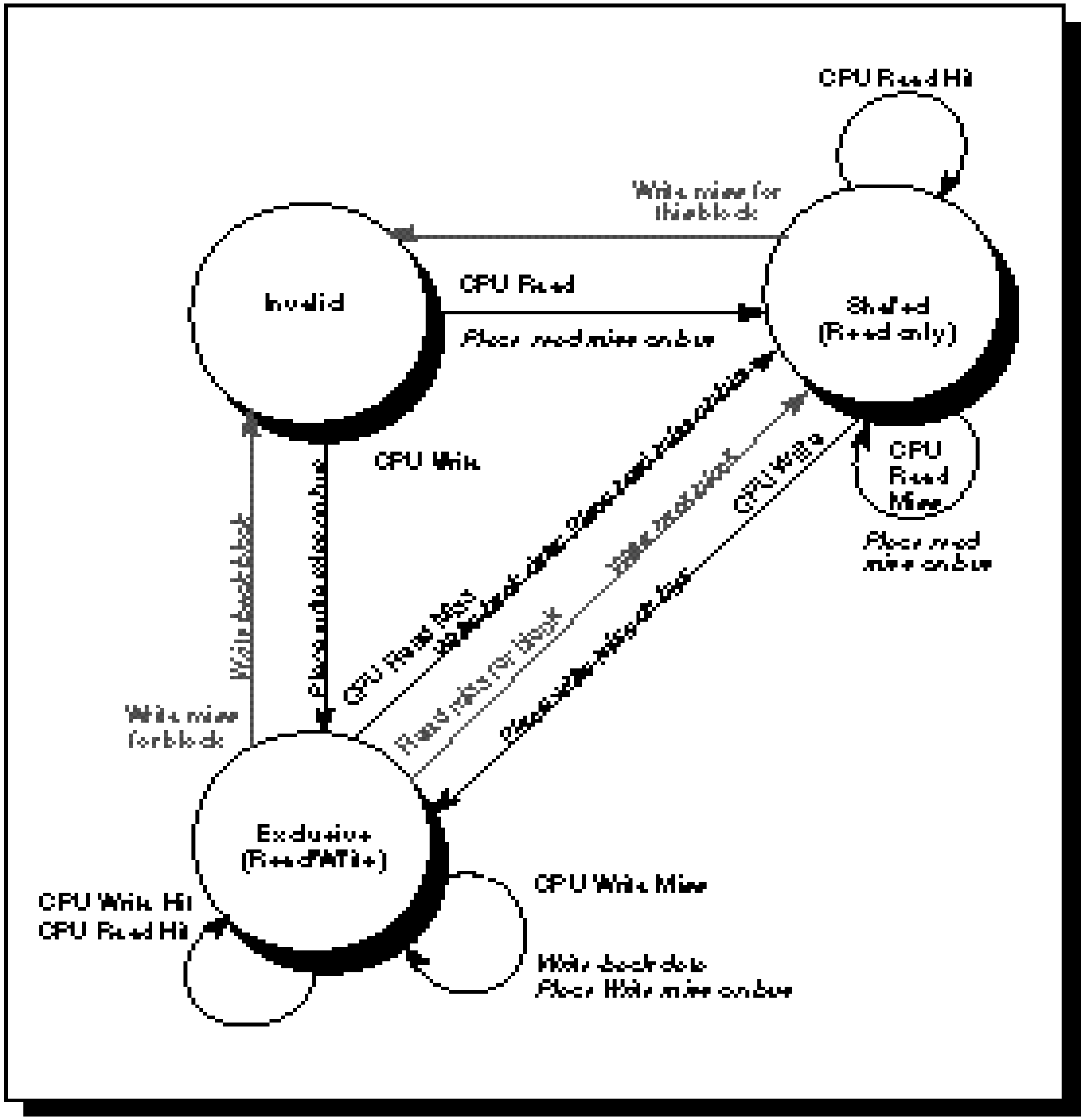
Place write miss on bus

Snoopy-Cache State Machine-II

- **State machine for *bus* requests**


Invalid
Shared (read/only)
Exclusive (read/write)
Write miss for this block
Write Back Block; abort memory access
Write Back Block; abort memory access
Write miss for this block
Read miss for this block

Snoop Cache: State Machine



Extensions:

- **Fourth State: Ownership**
- **Clean-> dirty, need invalidate only (upgrade request), don't read memory** **Berkeley Protocol**
- **Clean exclusive state (no miss for private data on write)** **MESI Protocol**
- **Cache supplies data when shared state (no memory access)** **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	A1	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	A1	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**

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

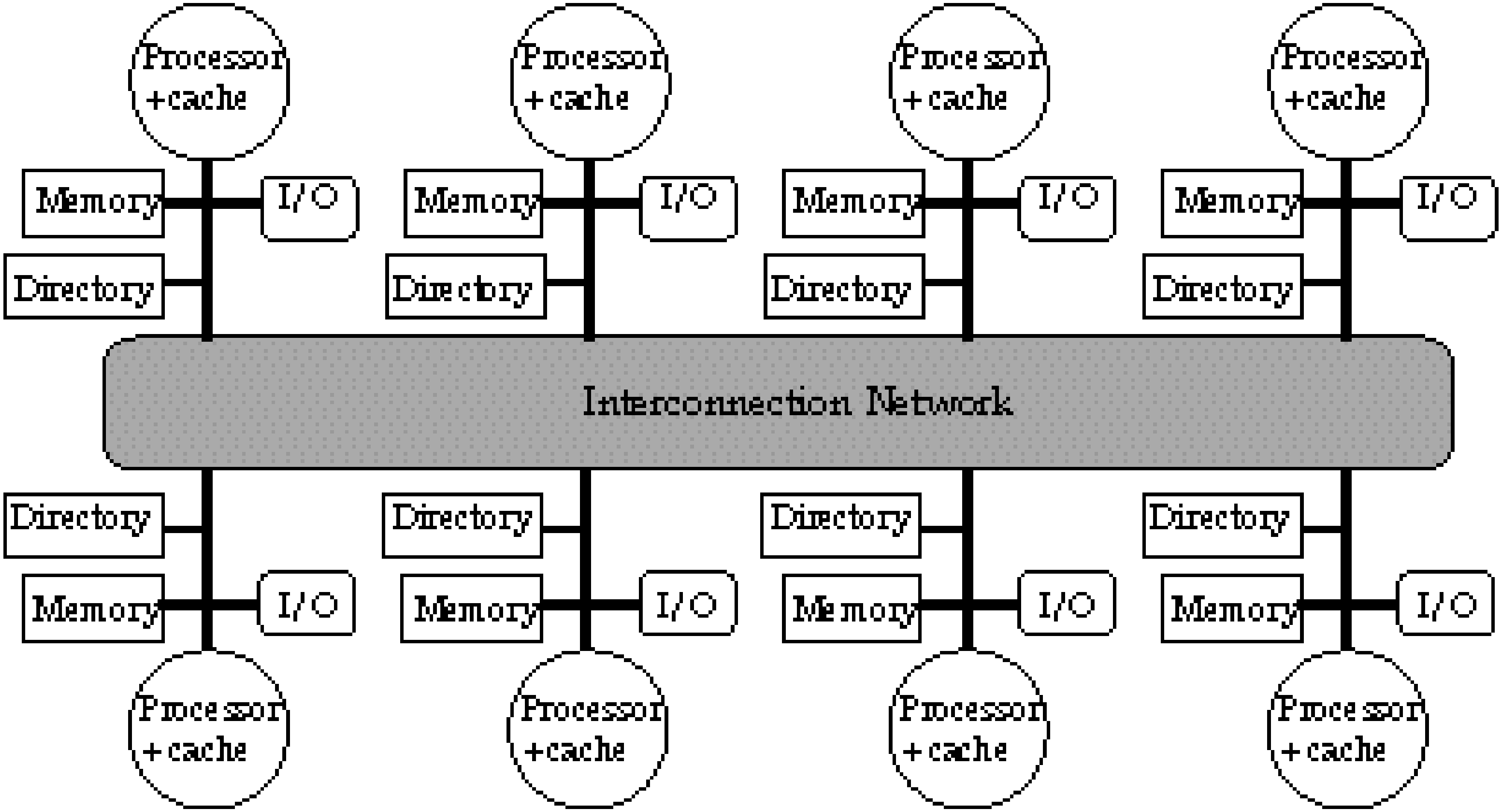
Implementing Snooping Caches

- **Bus serializes writes, getting bus ensures no one else can perform 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**
- **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 that obeys inclusion with L1 cache**

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 (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)
- 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

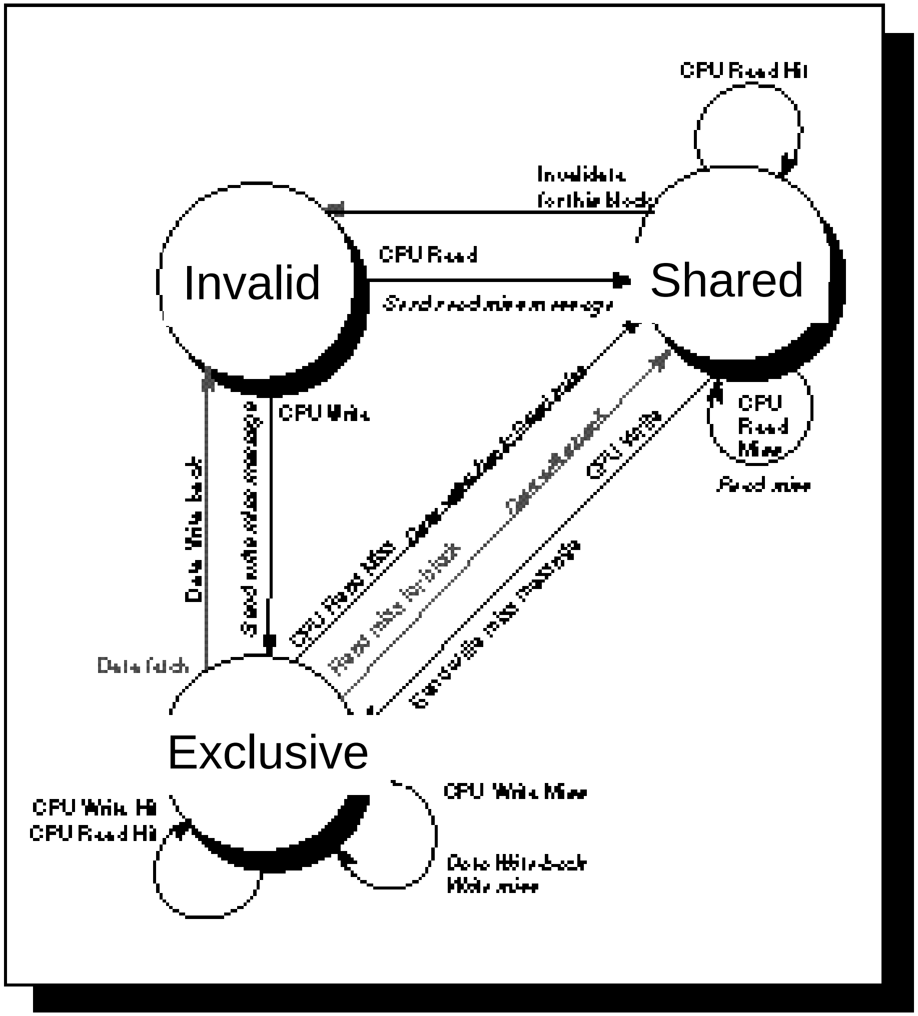
Directory Protocol

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

Directory Protocol Messages

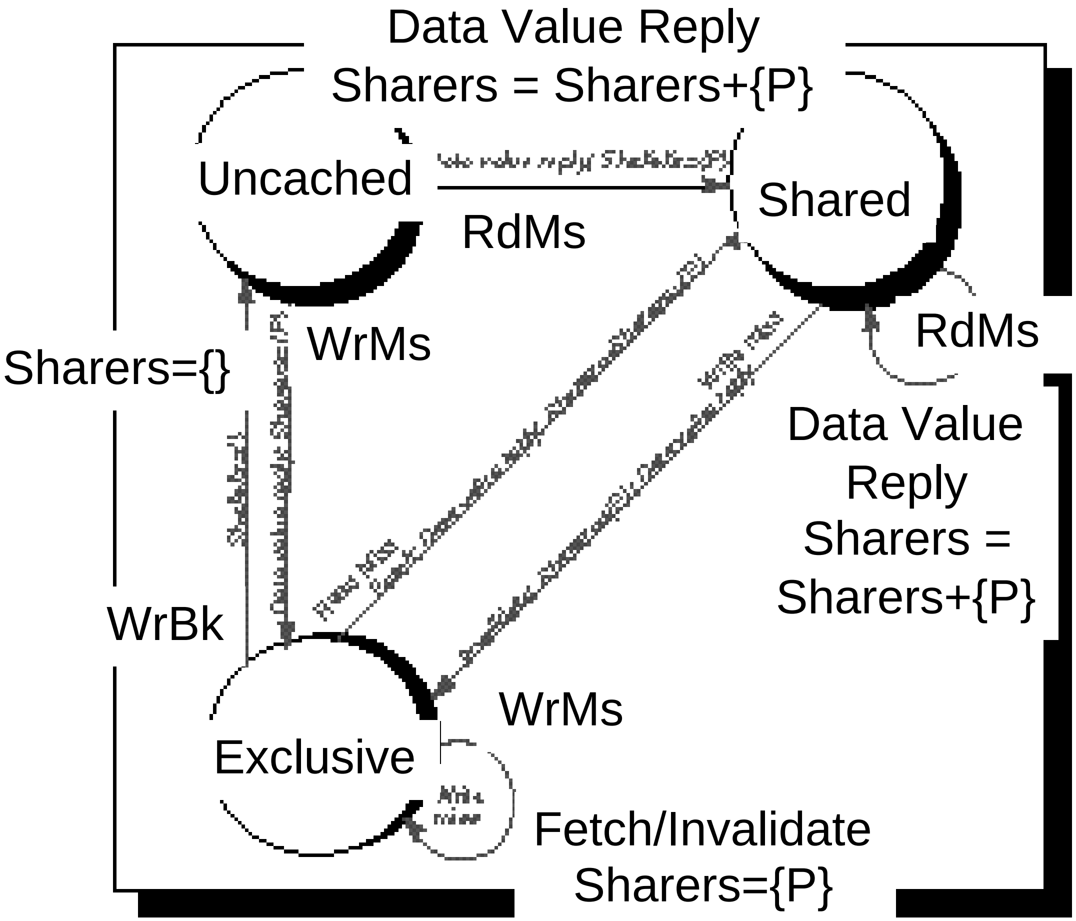
<table>
<tr><th><i>Message type</i></th><th><i>Source</i></th><th><i>Destination</i></th><th><i>Msg</i></th></tr>
<tr><td><b>Read miss</b></td><td><b>Local cache</b></td><td><b>Home directory</b></td><td><b>P, A</b></td></tr>
<tr><td colspan="4"><b><i>– Processor P reads data at address A;
send data and make P a read sharer</i></b></td></tr>
<tr><td><b>Write miss</b></td><td><b>Local cache</b></td><td><b>Home directory</b></td><td><b>P, A</b></td></tr>
<tr><td colspan="4"><b><i>– Processor P writes data at address A;
send data and make P the exclusive owner</i></b></td></tr>
<tr><td><b>Invalidate</b></td><td><b>Home directory</b></td><td><b>Remote caches</b></td><td><b>A</b></td></tr>
<tr><td colspan="4"><b><i>– Invalidate a shared copy at address A.</i></b></td></tr>
<tr><td><b>Fetch</b></td><td><b>Home directory</b></td><td><b>Remote cache</b></td><td><b>A</b></td></tr>
<tr><td colspan="4"><b><i>– Fetch the block at address A and send it to its home directory</i></b></td></tr>
<tr><td><b>Fetch/Invalidate</b></td><td><b>Home directory</b></td><td><b>Remote cache</b></td><td><b>A</b></td></tr>
<tr><td colspan="4"><b><i>– Fetch the block at address A and send it to its home directory;
invalidate the block in the cache</i></b></td></tr>
<tr><td><b>Data value reply</b></td><td><b>Home directory</b></td><td><b>Local cache</b></td><td><b>Data</b></td></tr>
<tr><td colspan="4"><b><i>– Return a data value from the home memory</i></b></td></tr>
<tr><td><b>Data write-back</b></td><td><b>Remote cache</b></td><td><b>Home directory</b></td><td><b>A, Data</b></td></tr>
<tr><td colspan="4"><b><i>– Write-back a data value for address A</i></b></td></tr>
</table>

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



- States identical to snoopy case; transactions very similar.
- Tranistions caused by read misses, write misses, invalidates, data fetch req.
- Generates read miss & write miss msg to home directory.
- Write misses that were broadcast on the bus => explicit invalidate & data fetch requests.

State Transition Diagram for the Directory



Data Value Reply
Sharers = Sharers+{P}

Uncached

Shared

RdMs

WrMs

Sharers={}

RdMs

Data Value
Reply
Sharers =
Sharers+{P}

WrBk

Exclusive

WrMs

Fetch/Invalidate
Sharers={P}

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

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, 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 & 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).**
  - **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 owner, and state of block is made Exclusive.**

Implementing a Directory

- We assume operations atomic, but they are not; reality is much harder; must avoid deadlock when run out of bufffers in network (see Appendix E)
- 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

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														
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														
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	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														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

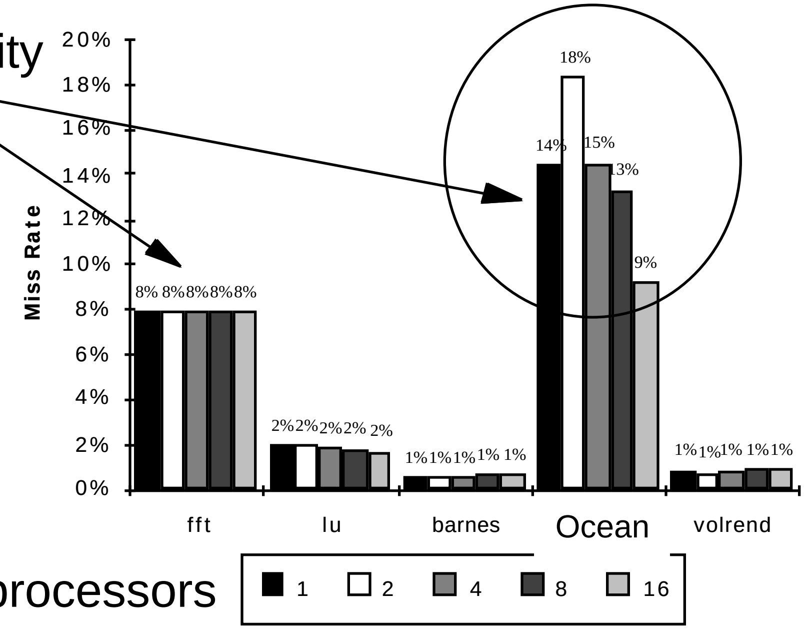
Miss Rates for Snooping Protocol

- 4th C: Conflict, Capacity, Compulsory and Coherency Misses
- More processors: increase coherency misses while decreasing capacity misses since more cache memory (for fixed problem size)
- Cache behavior of Five Parallel Programs:
  - FFT Fast Fourier Transform: Matrix transposition + computation
  - LU factorization of dense 2D matrix (linear algebra)
  - Barnes-Hut n-body algorithm solving galaxy evolution probem
  - Ocean simluates influence of eddy & boundary currents on large-scale flow in ocean: dynamic arrays per grid

Miss Rates for Snooping Protocol

High Capacity Misses

Miss Rate

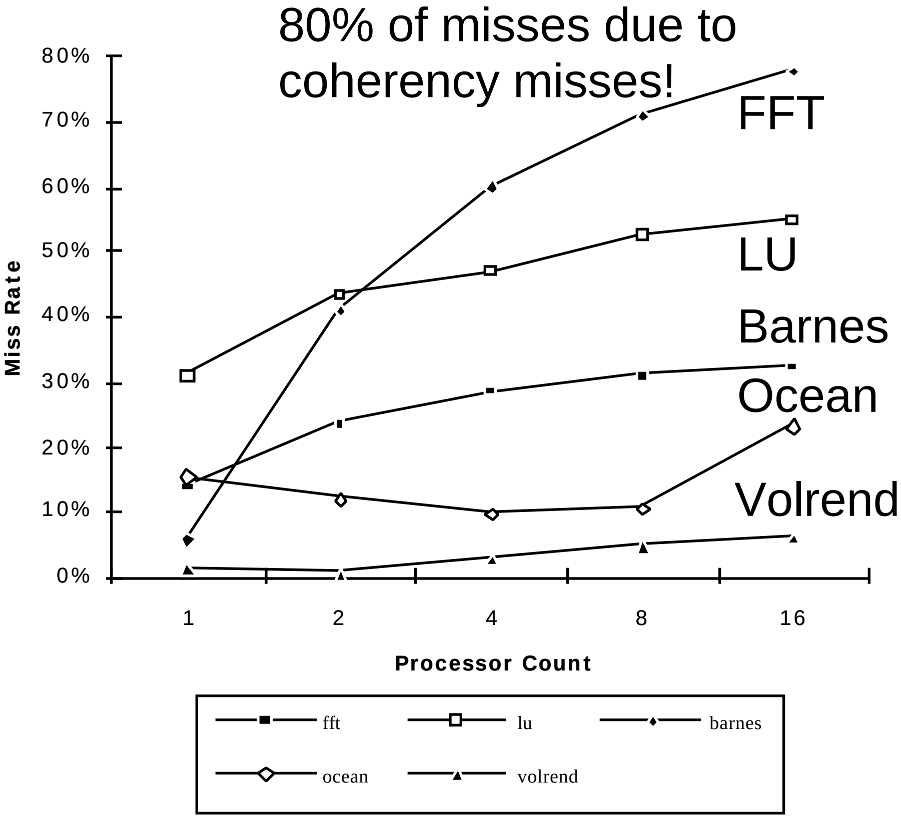


# of processors

Big differences in miss rates among the programs

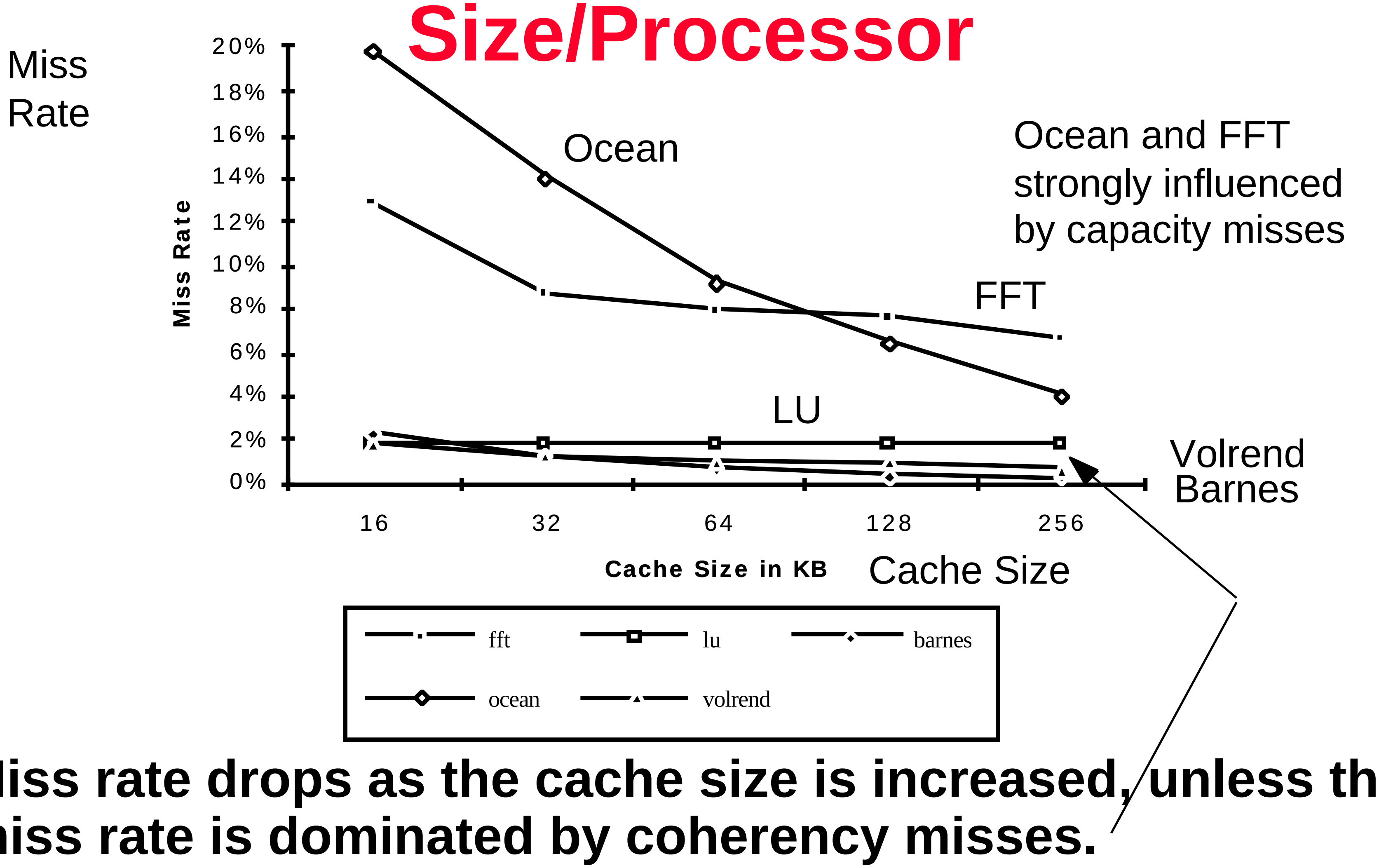
- – **Cache size is 64KB, 2-way set associative, with 32B blocks.**
- – **Misses in these applications are generated by accesses to data that is potentially shared.**
- – **Except for Ocean, data is heavily shared; in Ocean only the boundaries of the subgrids are shared, though the entire grid is treated as a shared data object. Since the boundaries change as we increase the processor count (for a fixed size problem), different amounts of the grid become shared. The anamolous increase in miss rate for Ocean in moving from 1 to 2 processors arises because of conflict misses in accessing the subgrids.**

% Misses Caused by Coherency Traffic vs. # of Processors


80% of misses due to coherency misses!

- **% cache misses caused by coherency transactions typically rises when a fixed size problem is run on more processors.**
- **The absolute number of coherency misses is increasing in all these benchmarks, including Ocean. In Ocean, however, it is difficult to separate out these misses from others, since the amount of sharing of the grid varies with processor count.**
- **Invalidations increases significantly; In FFT, the miss rate arising from coherency misses increases from nothing to almost 7%.**

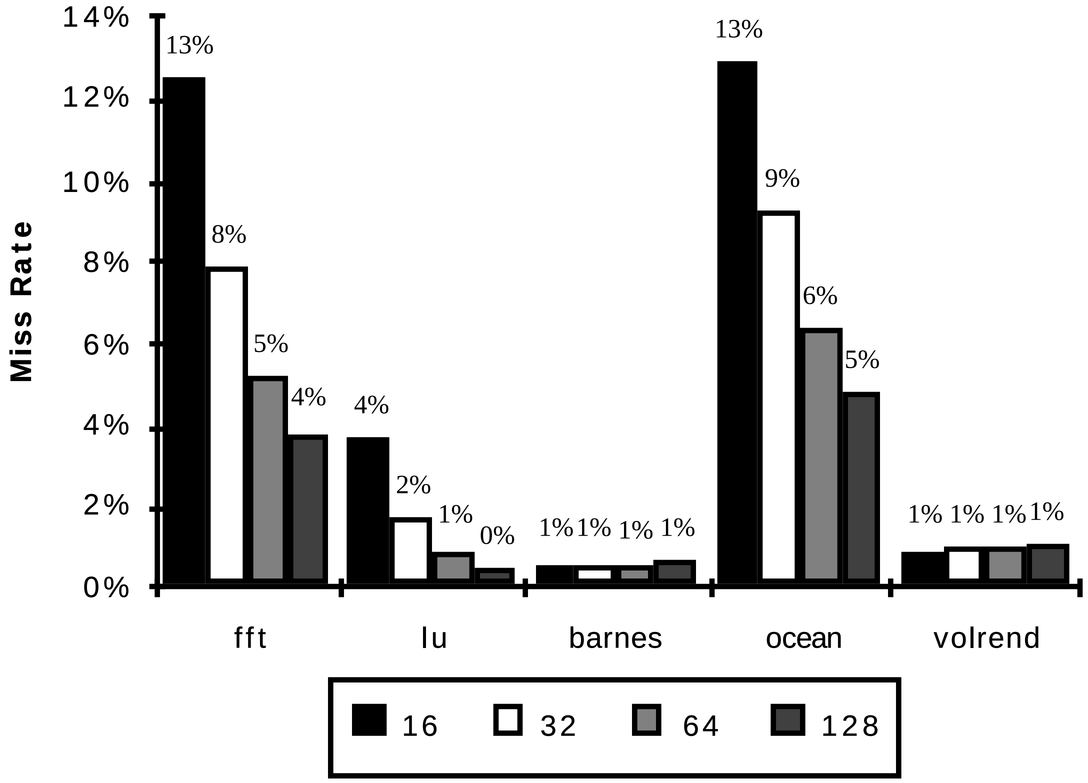
Miss Rates as Increase Cache Size/Processor



- **Miss rate drops as the cache size is increased, unless the miss rate is dominated by coherency misses.**
- **The block size is 32B & the cache is 2-way set-associative. The processor count is fixed at 16 processors.**

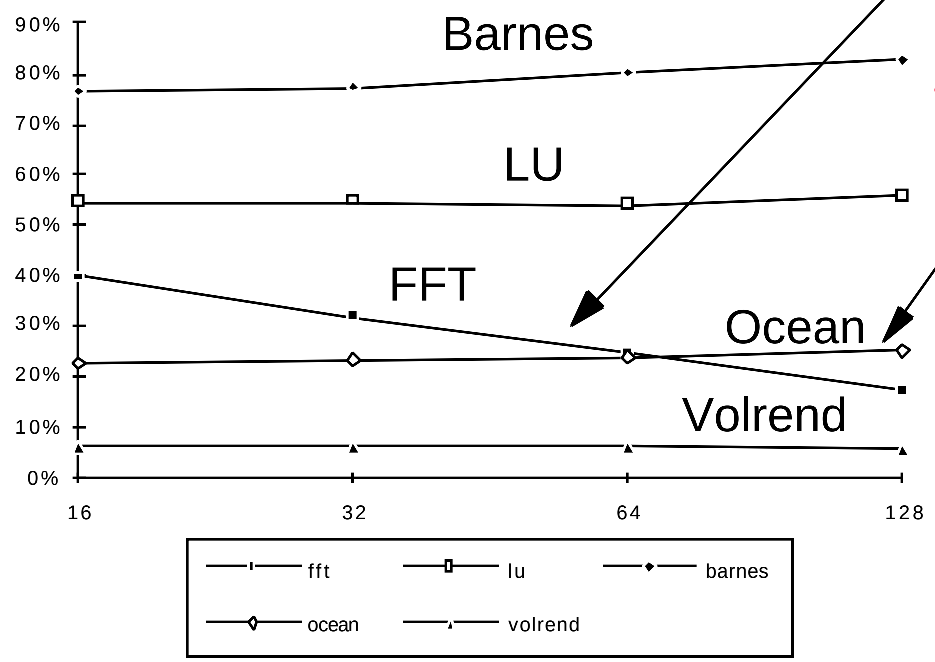
Miss Rate vs. Block Size

- Since cache block hold multiple words, may get coherency traffic for unrelated variables in same block
- False sharing arises from the use of an invalidation-based coherency algorithm. It occurs when a block is invalidated (and a subsequent reference causes a miss) because some word in the block, other than the one being read, is written into.



miss rates mostly fall with increasing block size

% Misses Caused by Coherency Traffic vs. Block Size

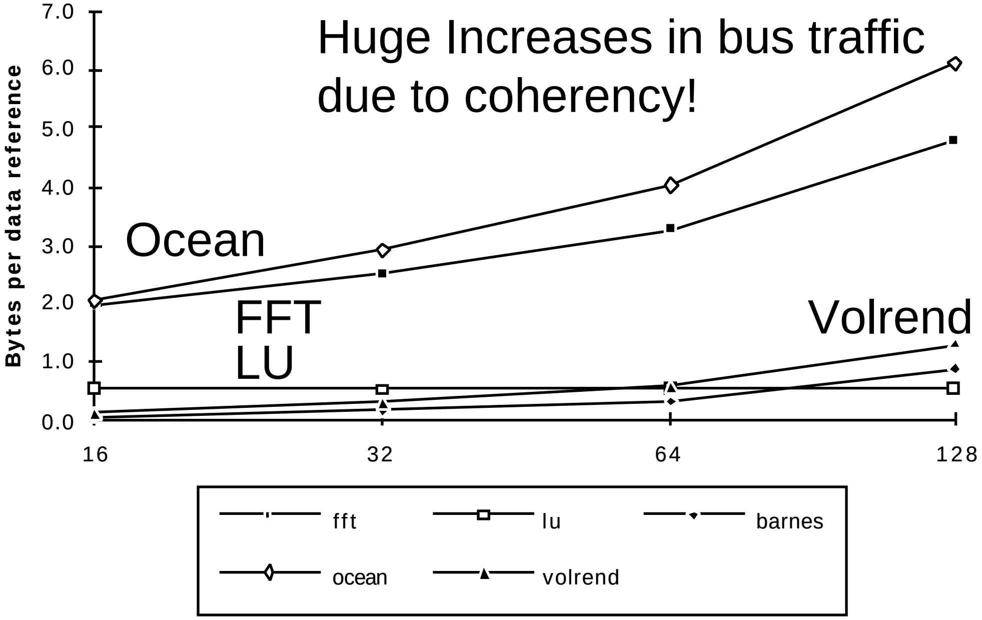


Behavior tracks cache size behavior
FFT: Coherence misses reduced faster than capacity misses!

- **FFT communicates data in large blocks & communication adapts to the block size (it is a parameter to the code); makes effective use of large blocks.**
- **Ocean competing effects that favor different block size**
  - **Accesses to the boundary of each subgrid, in one direction the accesses match the array layout, taking advantage of large blocks, while in the other dimension, they do not match. These two effects largely cancel each other out leading to an overall decrease in the coherency misses as well as the capacity misses.**

Bus Traffic as Increase Block Size

Bytes per data ref

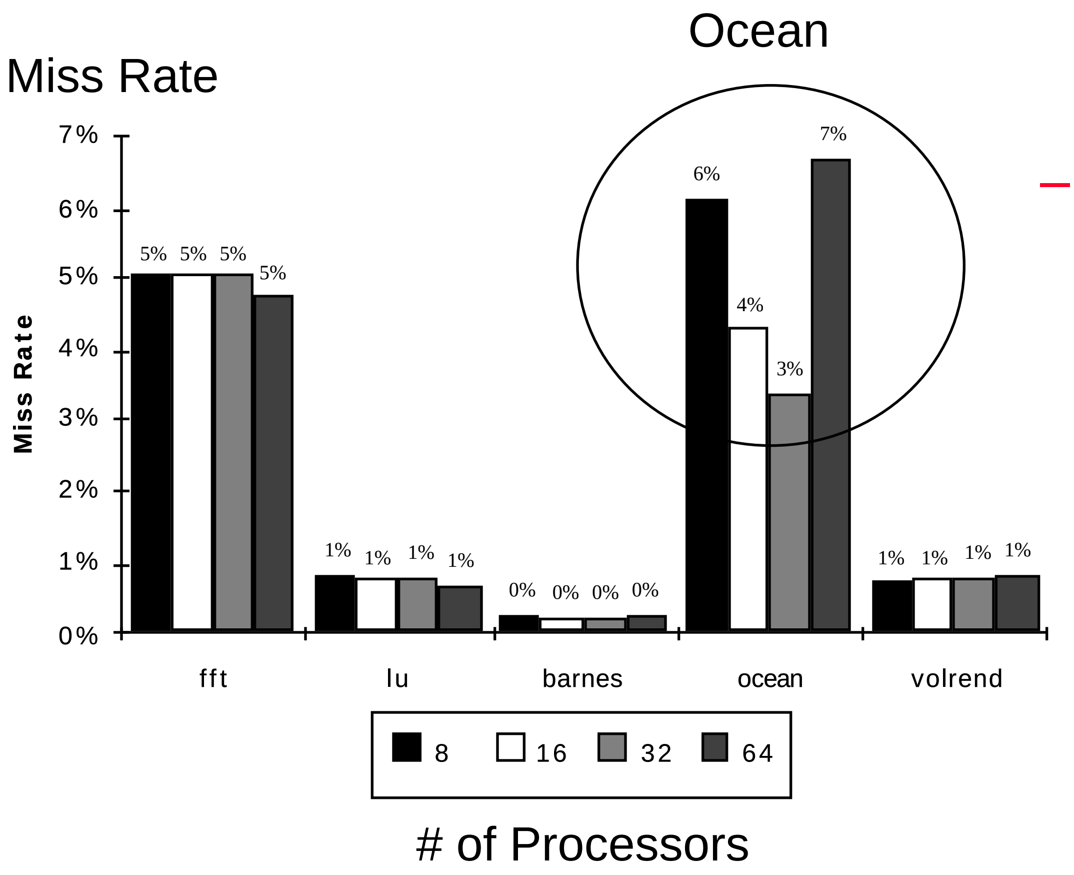


Huge Increases in bus traffic due to coherency!

- Bus traffic climbs steadily as the block size is increased.
- The factor of 3 increase in traffic for Ocean is the best argument against larger block sizes.
- Remember that our protocol treats ownership misses the same as other misses, slightly increasing the penalty for large cache blocks: in both Ocean and FFT this effect accounts for less than 10% of the traffic.

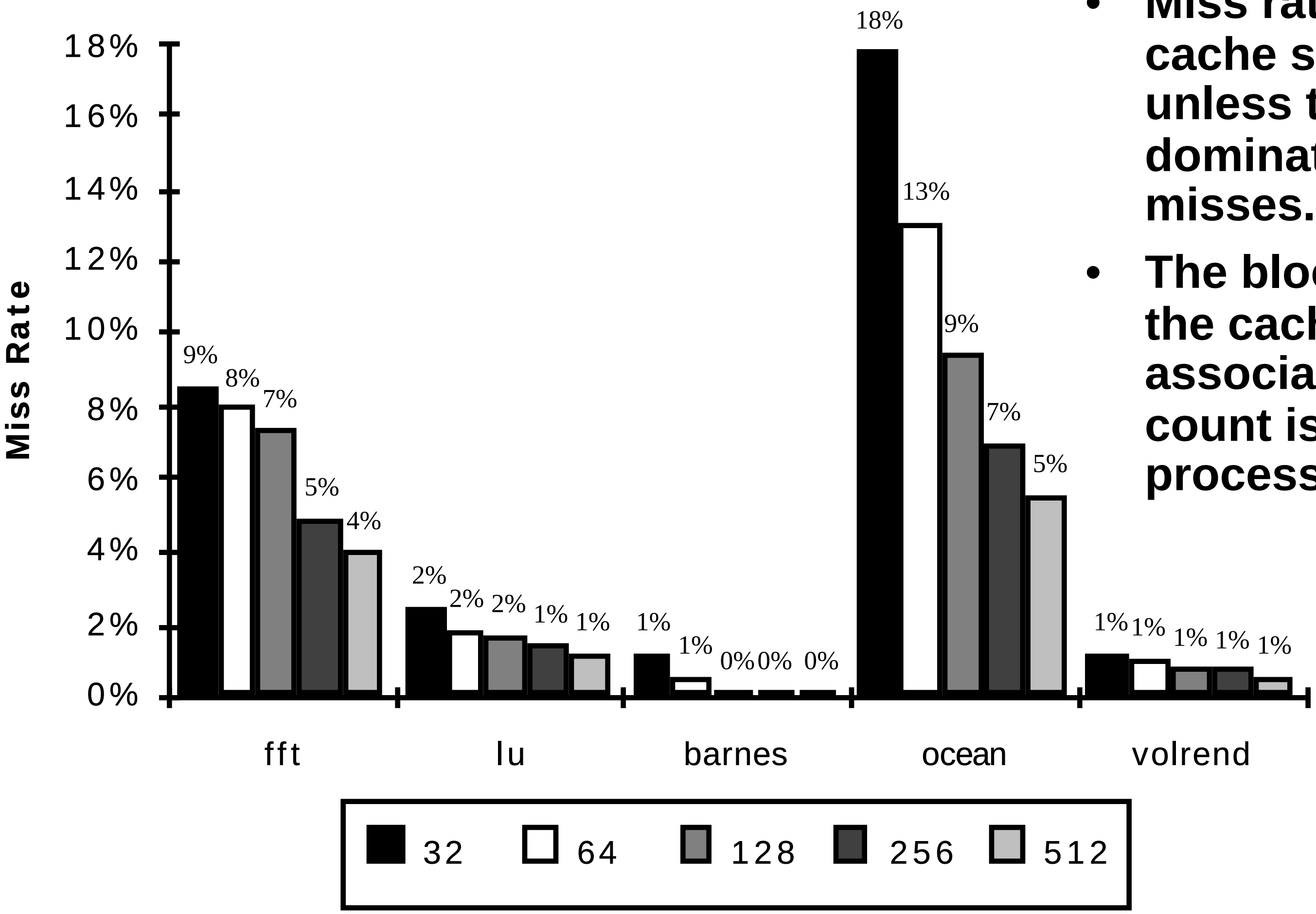
Miss Rates for Directory

Use larger cache to circumvent longer latencies to directories



- – Cache size is 128 KB, 2-way set associative, with 64B blocks (cover longer latency)
- – Ocean: only the boundaries of the subgrids are shared. Since the boundaries change as we increase the processor count (for a fixed size problem), different amounts of the grid become shared. The increase in miss rate for Ocean in moving from 32 to 64 processors arises because of conflict misses in accessing small subgrids & for coherency misses for 64 processors.

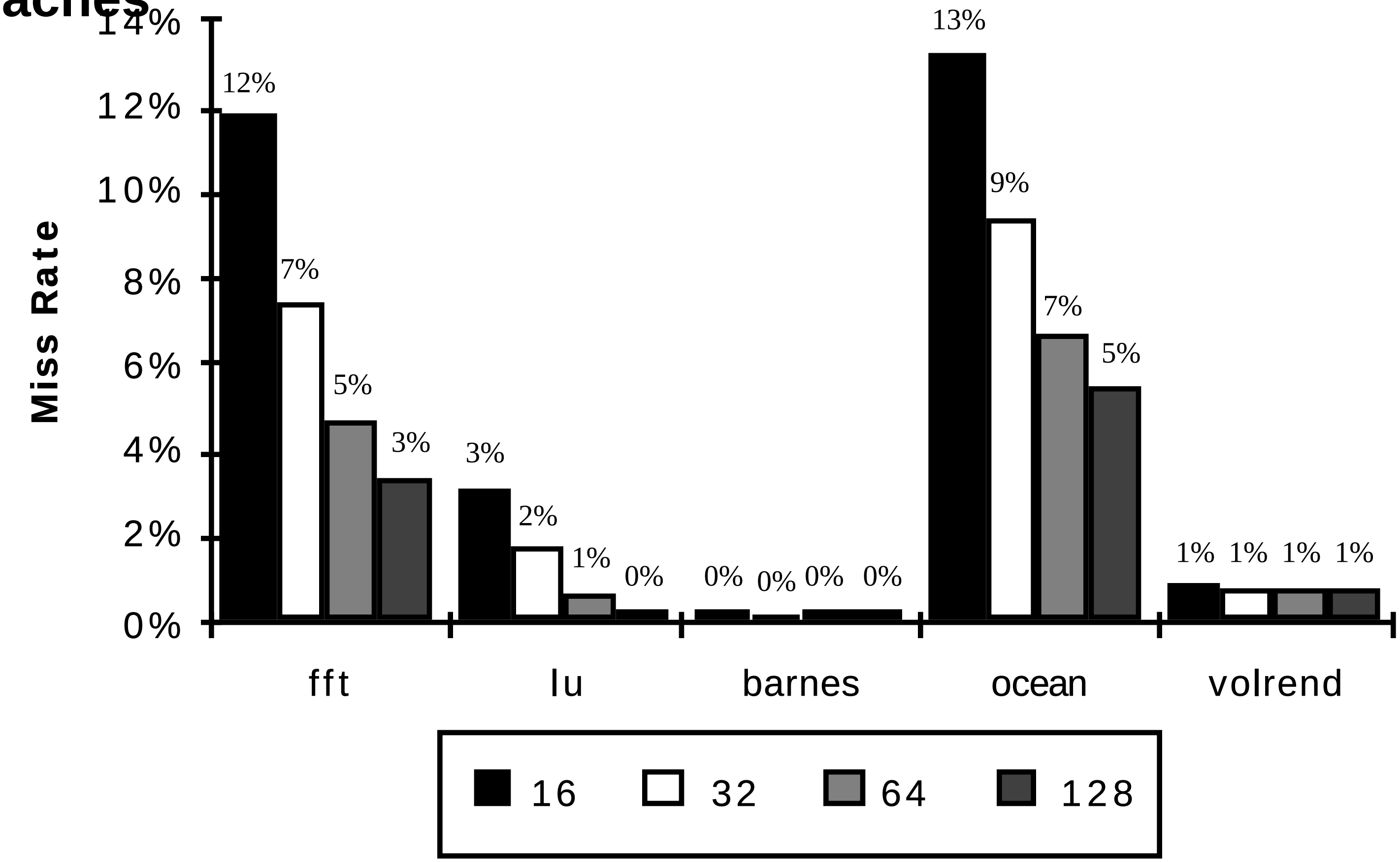
Miss Rates as Increase Cache Size/Processor for Directory



- Miss rate drops as the cache size is increased, unless the miss rate is dominated by coherency misses.
- The block size is 64B and the cache is 2-way set-associative. The processor count is fixed at 16 processors.

Block Size for Directory

- Assumes 128 KB cache & 64 processors
  - Large cache size to combat higher memory latencies than snoop caches



Summary

- **Caches contain all information on state of cached memory blocks**
- **Snooping and Directory Protocols similar; bus makes snooping easier because of broadcast**
- **Directory has extra data structure to keep track of state of all cache blocks**
- **Distributing directory => scalable shared address multiprocessor**