Lecture 30: Multiprocessors— Flynn Categories, Large vs. Small Scale, Cache Coherency

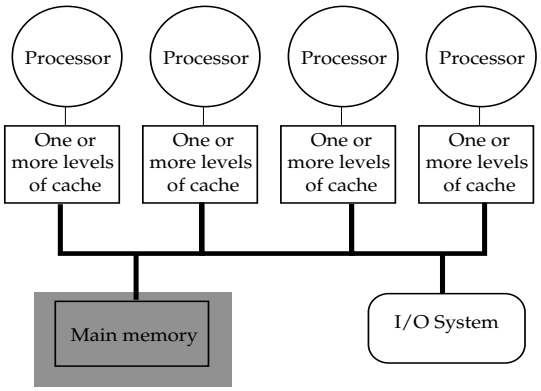
Professor Randy H. Katz
Computer Science 252
Spring 1996

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**
- **MIMD (Multiple Instruction Multiple Data)**
  - **Examples: SPARCCenter, T3D**
    - » **Flexible**
    - » ***Use off-the-shelf micros***

Small-Scale MIMD Designs

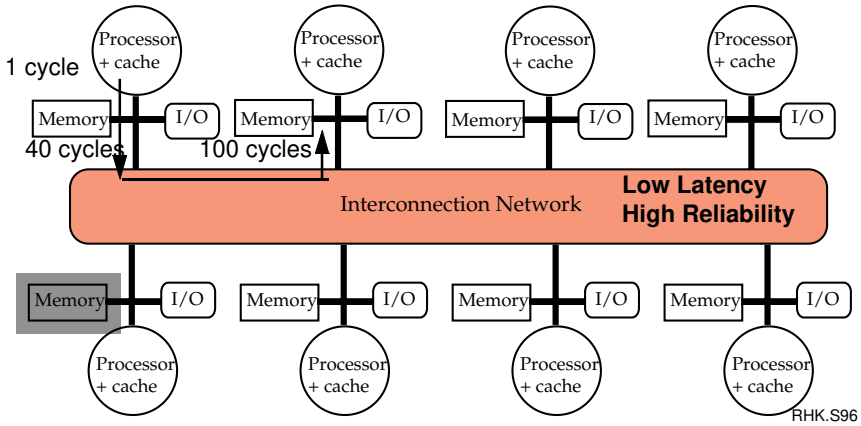
- **Memory: centralized with uniform access time ("uma") and bus interconnect**
- **Examples: SPARCCenter, Challenge, SystemPro**



RHK.S96 3

Large-Scale MIMD Designs

- **Memory: distributed with nonuniform access time ("numa") and scalable interconnect (distributed memory)**
- **Examples: T3D, Exemplar, Paragon, CM-5**



RHK.S96 4

Communication Models

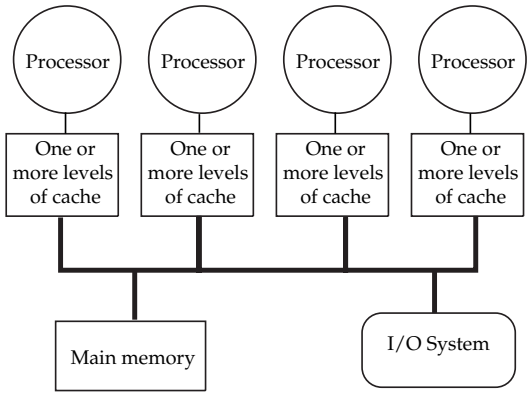
- **Shared Memory**
  - Processors communicate with shared address space
  - Easy on small-scale machines
  - Advantages:
    - » Model of choice for uniprocessors, small-scale MPs
    - » Ease of programming
    - » Lower latency
    - » Easier to use hardware controlled caching
- **Message passing**
  - 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**

Important Communication Properties

- **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

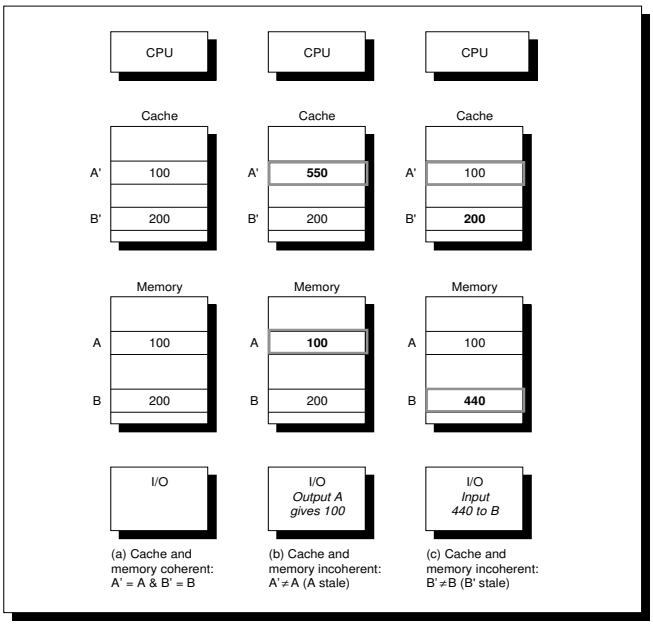
Small-Scale—Shared Memory

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



RHK.S96 7

The Problem of Cache Coherency



RHK.S96 8

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
    - » Otherwise 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) tradeoff

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

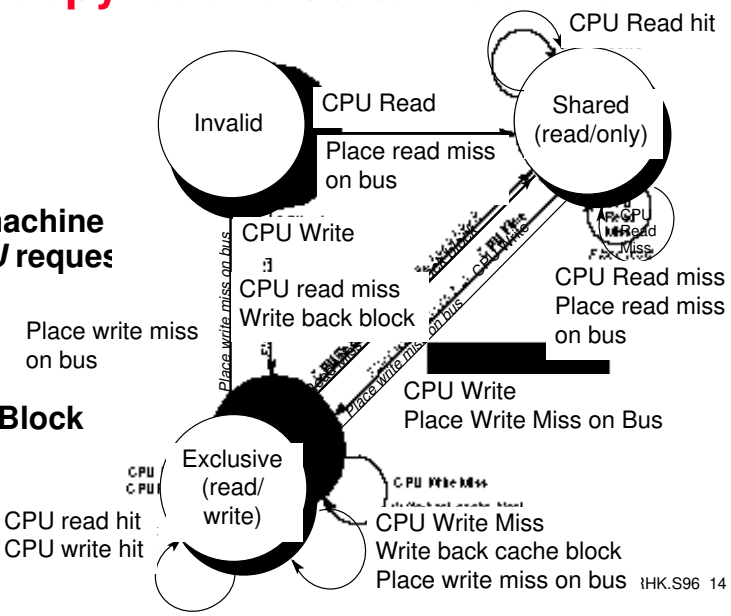
An Example Snoopy Protocol

- **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

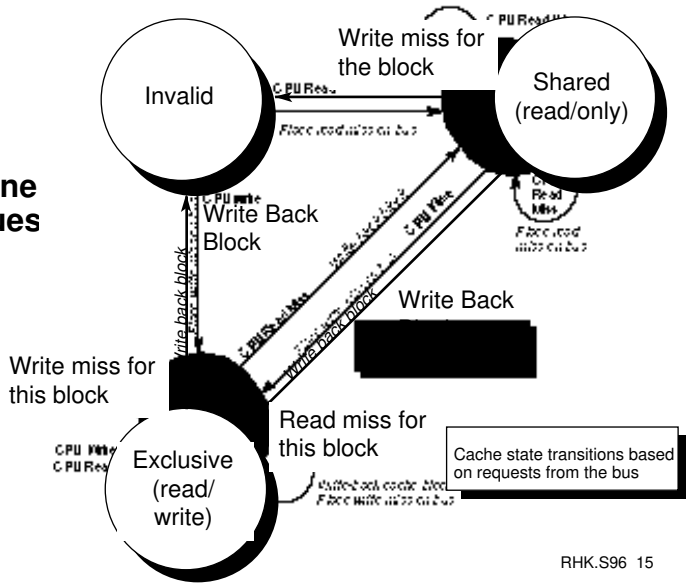
- **State machine for *CPU* reques**

Cache Block State



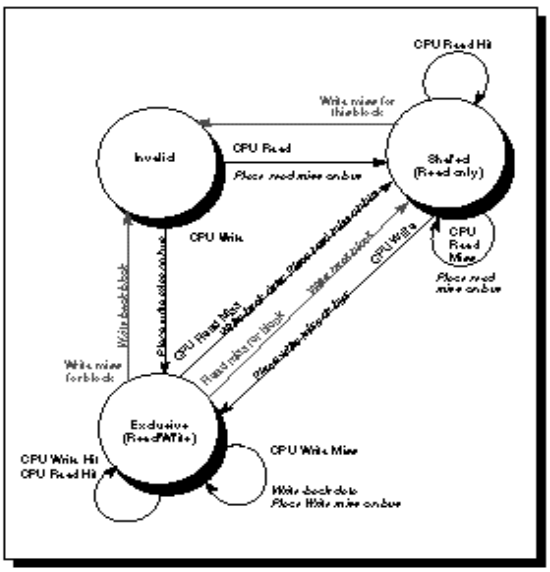
Snoopy-Cache State Machine-II

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



RHK.S96 15

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

RHK.S96 16

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**