

CSE 120

Summer, 2005

Day 8

Multiple Processor Systems

Instructor: Neil Rhodes

Why?

Limits on improving CPU speed

- Speed of light: 1 foot/nanosecond

Add more CPUs working together

Some problems can't be split

- If it takes 9 months for 1 woman to have a baby, can adding more women to the job speed it up?
 - Some problems inherently sequential

Overview

Multiprocessors: Multiple CPUs with shared memory

- Hardware
- Operating Systems
- Synchronization
- Scheduling

Multicomputer: Tightly-coupled CPUs with local memory

- Interconnection
- Communication software
- Distributed Shared Memory (DSM)
- Scheduling

Distributed Systems: Loosely coupled computers

- Networking
- Remote File Systems
- Shared Objects
- Coordination

Multiprocessors

Model:

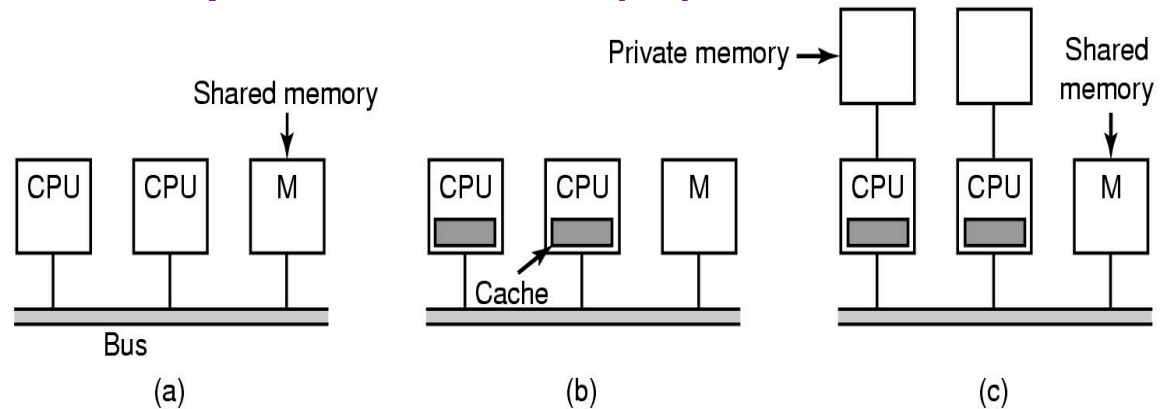
- Multiple processors sharing a common memory
- Each processor can read/write to any memory location
 - One oddity: CPU can write a value to memory location, and read back a different value

Speed of accessing memory

- **Uniform Memory Access (UMA)**: every memory word can be read as fast as every other memory word by any CPU
- **Nonuniform Memory Access (NUMA)**: some memory is closer to some CPUs than others

UMA Multiprocessors (1)

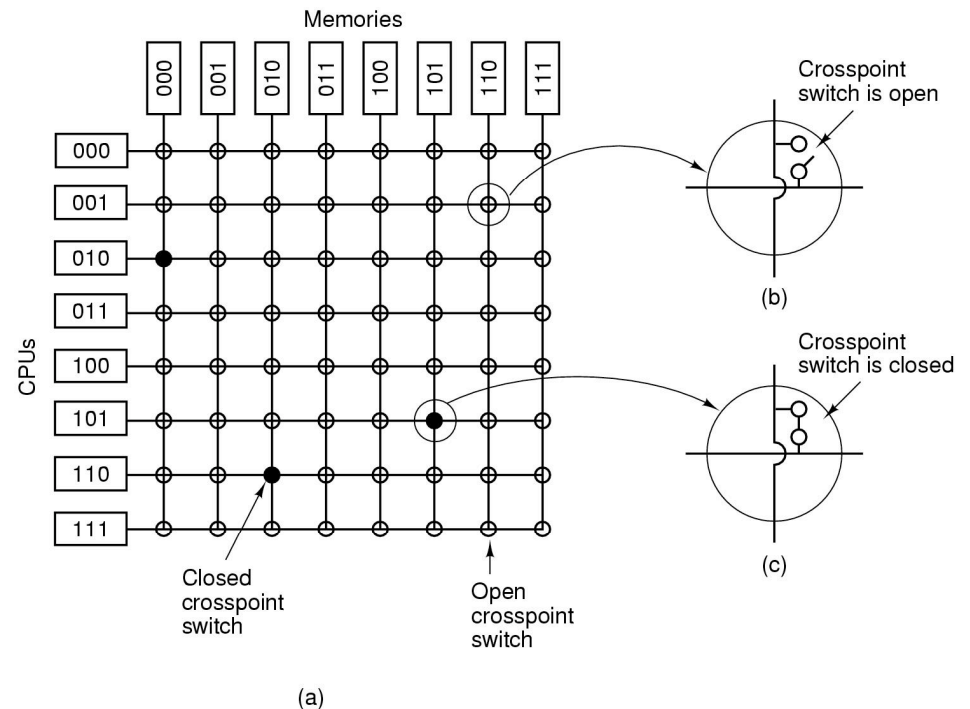
Bus



- Local cache
 - Cache not just a single word, but a block of 32 or 64
 - Cache marked dirty or clean
 - When writing, must check with other caches to see whether they have same block in cache
 - If yes and dirty, other CPU must write dirty block to writing CPU
 - If yes and clean, other CPU must release block from cache
- Local memory
 - Used only for read-only data
 - Code, constants
 - Plus local variables
 - Data that need not be shared
- Problems: bandwidth of the single bus can be the bottleneck
 - Each CPU access memory every X instructions.

UMA Multiprocessors (2)

Crossbar switches



- n^2 switches between memory and CPU
- Non-blocking: CPU 1 can access memory block 1 while CPU 3 is accessing memory block 2
- Downside: large number of switches

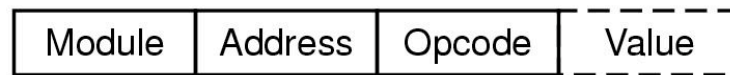
UMA Multiprocessors (3)

Switching network

- Can build up a network out of 2x2 switches



(a)

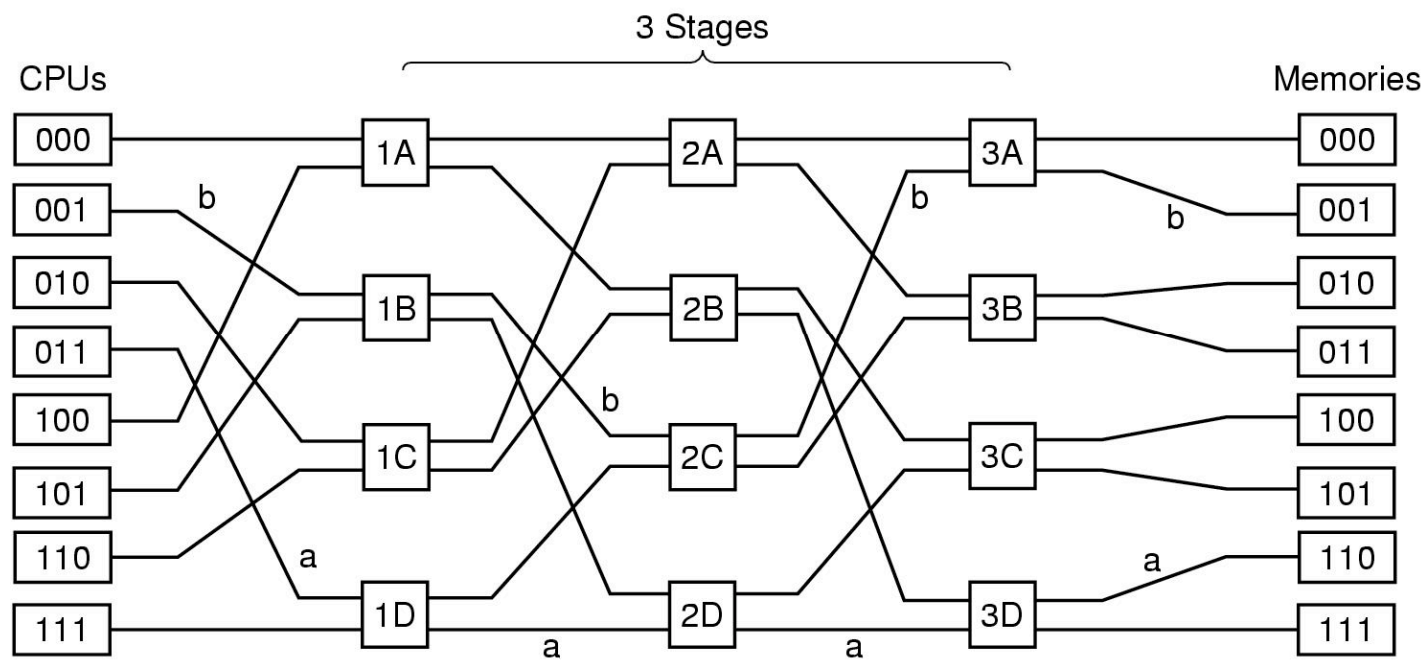


(b)

UMA Multiprocessors (3)

Switching network

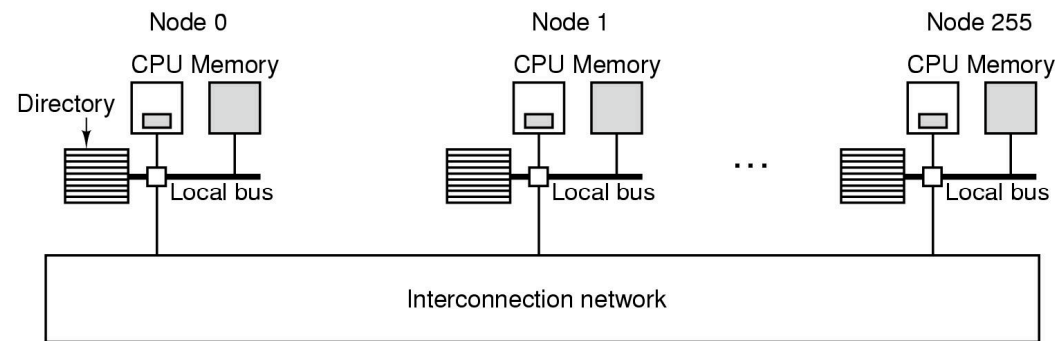
- Omega network:
 - Usually use low-order bits of memory address as module index (spreads out memory access across memory modules)



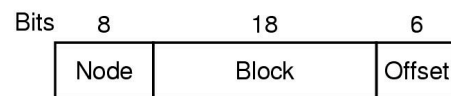
NUMA Multiprocessors

Characteristics

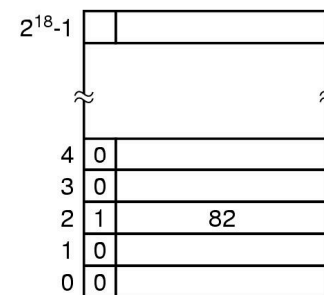
- Single address space visible to all CPUs
- Access to remote memory is via 2 instructions
 - Load
 - Store
- Access to remote memory is slower than to local memory



(a)



(b)

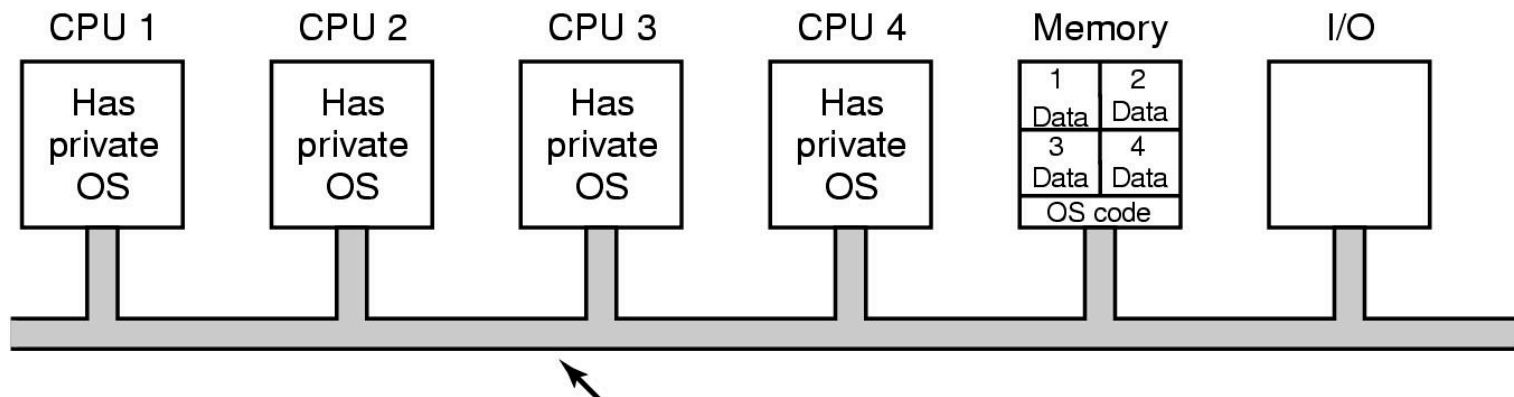


Directory at node 36

(c)

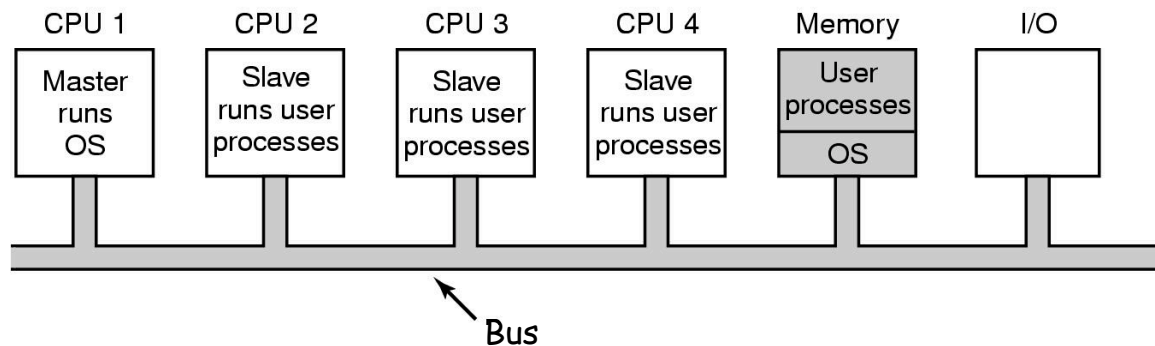
Multiprocessor Operating System Types (1)

Each CPU has its own copy of the OS



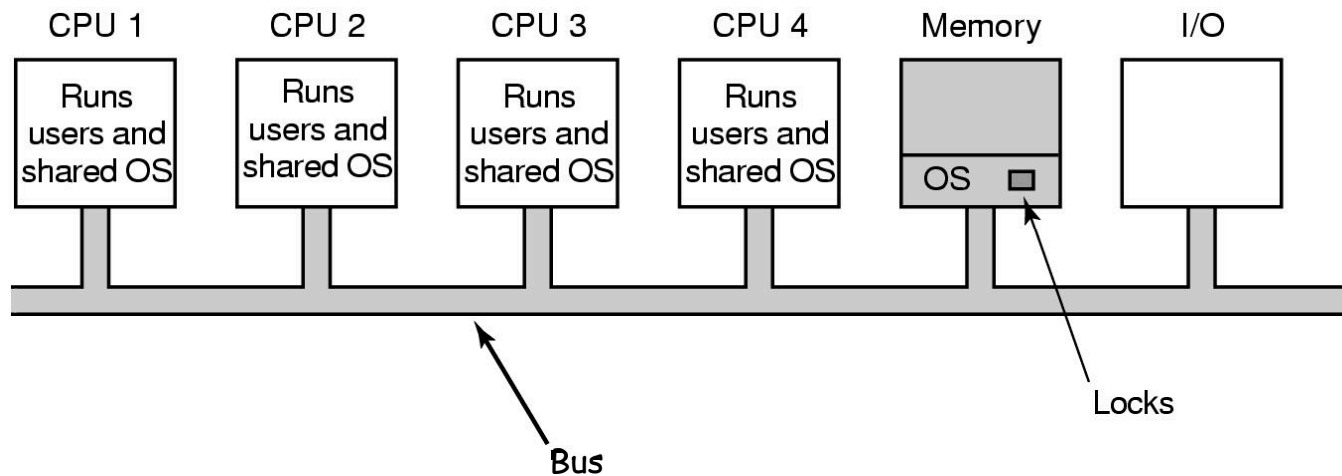
Multiprocessor Operating System Types (2)

Master-slave multiprocessors



Multiprocessor Operating System Types (3)

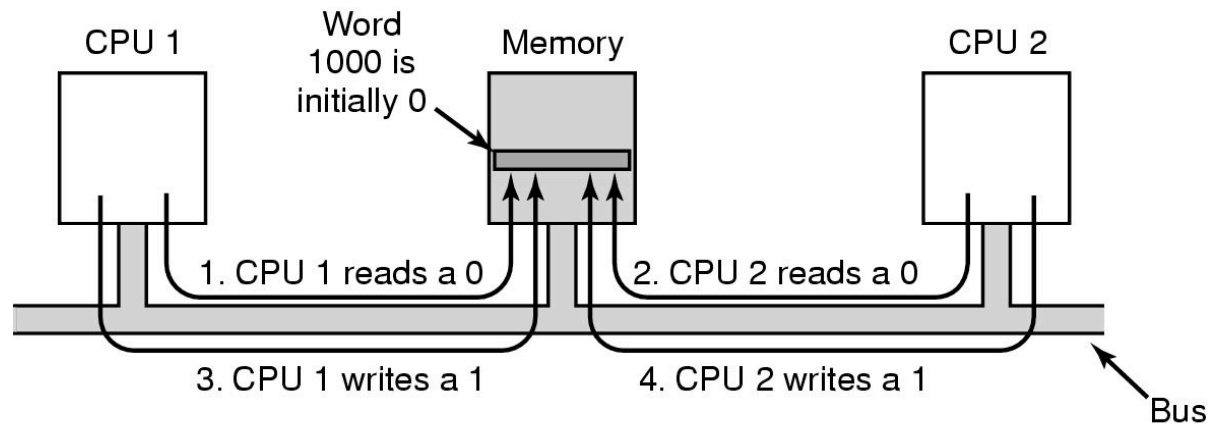
Symmetric multiprocessors (SMP)



Multiprocessor Synchronization

Implementing mutex

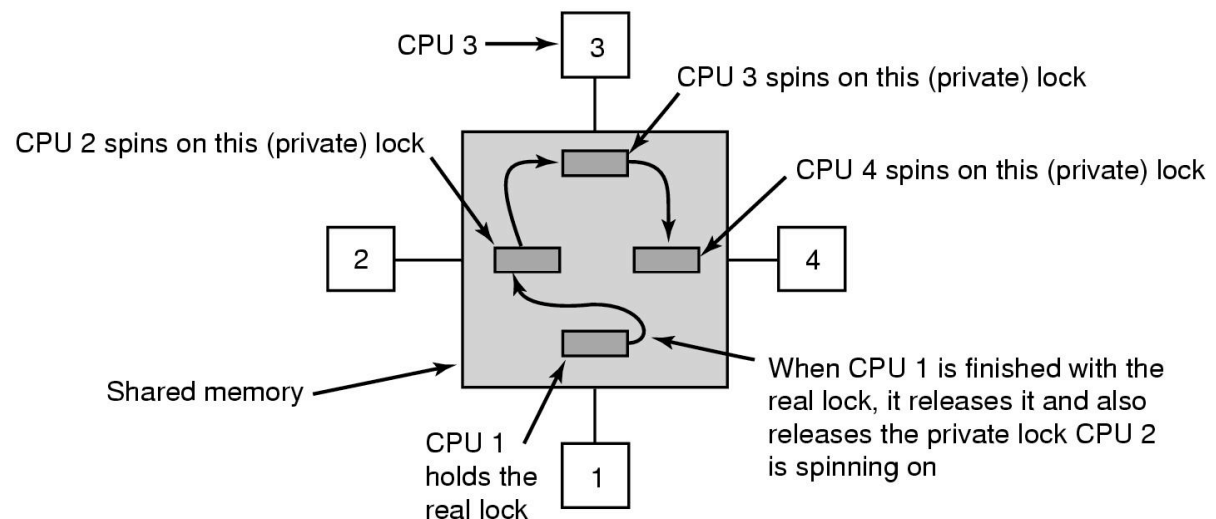
- Can't disable interrupts
 - only affects current CPU
- Test-and-set requires two bus cycles
 - Another CPU may use bus in the middle
- Solution:
 - Lock bus
 - Do 2 memory accesses
 - Release bus



Multiprocessor Synchronization

Problem with Test and Set

- Waiting CPU constantly testing
- Potential solution: cache
 - Problem: waiting CPU does write; gets cache block containing lock
 - using CPU uses memory in cache block. When it reads, it gets cache block
- Better solution: read first
 - Only do Test and Set if it appears to be free
 - All reads can be cached
- Another solution: exponential backoff
 - If Test and Set fails to acquire lock, delay before trying again
 - Increase delay by a factor of two after each failure (up to some maximum)
- Yet another solution:
private locks



Multiprocessor Synchronization

Spinning vs. switching

- When waiting for a lock, can either:
 - spin, waiting for lock
 - switch to another thread/process
- Tradeoffs
 - Spinning: wastes waste CPU time waiting for lock
 - Switching: wastes time:
 - saving current state
 - Acquiring ready list lock
 - Loading new state
 - Also, cache misses
- Best choice: whichever one wastes less time
 - How to know?
 - Spin for a while, then switch?
 - Look at history (how long have recent spin times on this lock been)

Multiprocessor Scheduling

Uniprocessor

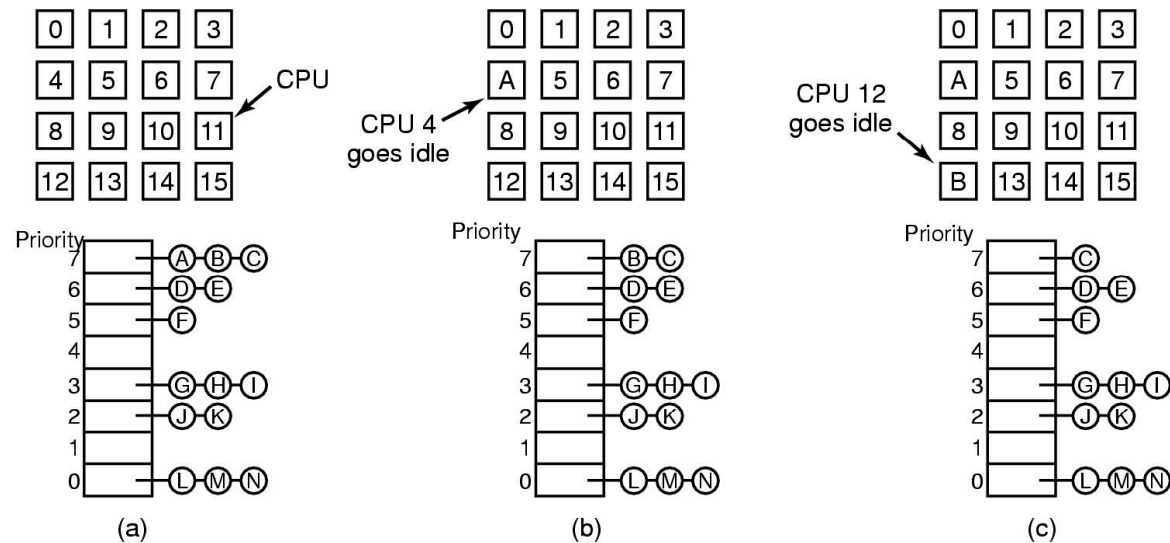
- What process to run?

Multiprocessor

- What process to run?
- Where to run it?

Processes

- Related/unrelated



Multiprocessor Scheduling

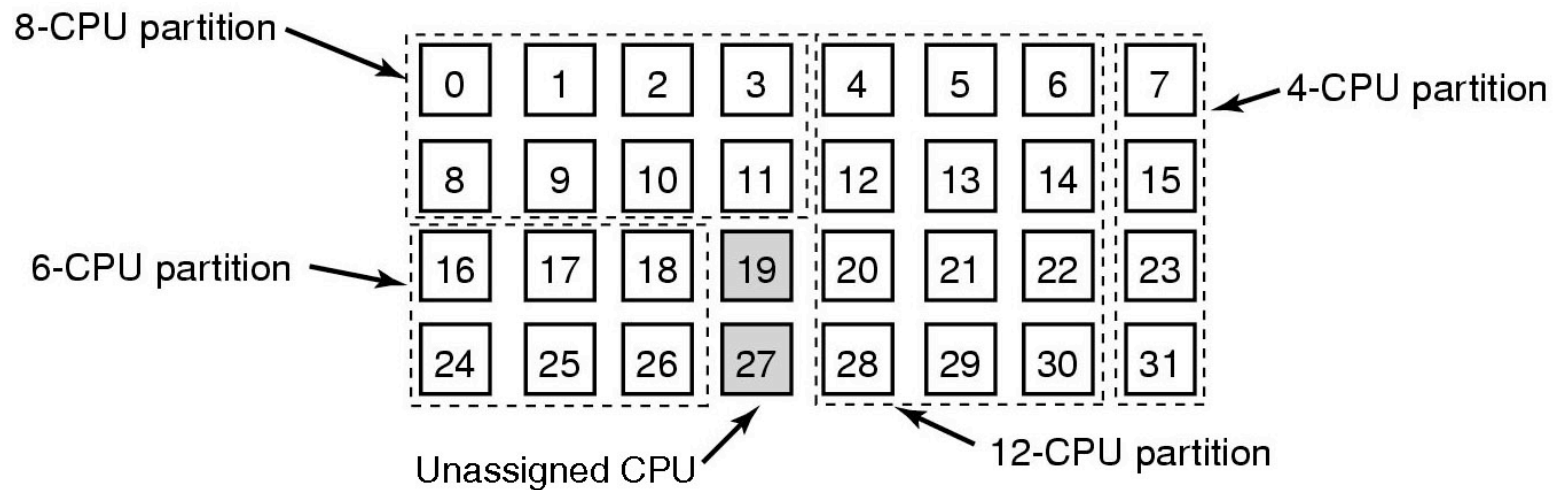
Problems:

- Contention for single data structure
- Interrupted process may still hold spin lock
 - Smart scheduling: process sets a flag showing it holds spin lock
 - System doesn't interrupt process holding spin lock, but wait a little more time to release the spin lock
- Caching
 - If process A ran on machine B last time, some of its data may still be in B's cache
 - Prefer to rerun on same machine
- Two-level scheduling
 - When process is created, assign to a machine (stays there forever)
 - Each machine has its own ready queue
 - Good for caching
 - Contention for single ready list gone
 - If ready queue is empty, grab a process from another machine

Multiprocessor Scheduling

Space Sharing

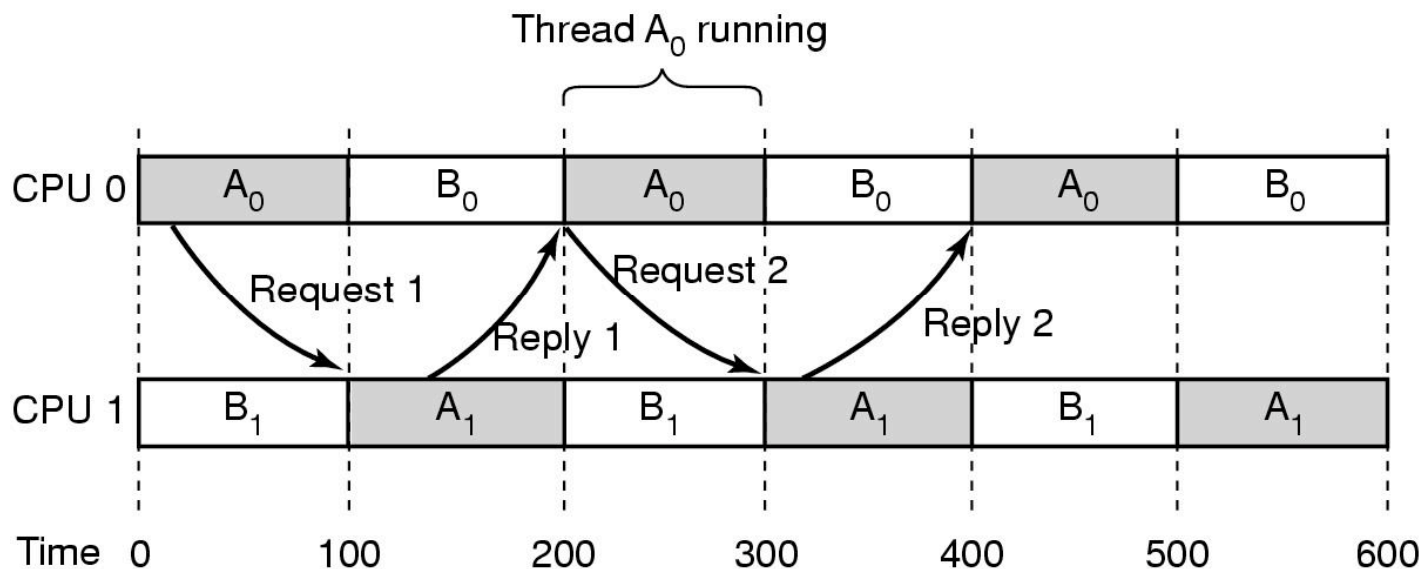
- Related threads are scheduled together across multiple machines
 - Stay on machine until done
 - Machine idle if thread blocks on I/O



Multiprocessor Scheduling

Gang Scheduling

- Groups of related threads scheduled as a *gang*
- All members of a gang run simultaneously (on different CPUs)
- All gang members start and stop their time slices together



Multiprocessor Scheduling

Gang Scheduling

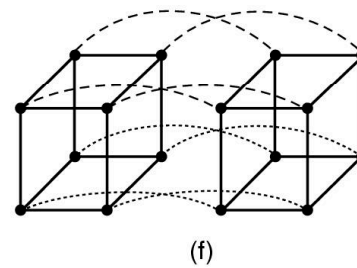
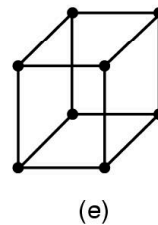
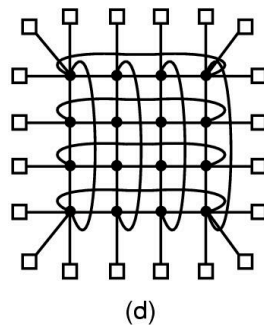
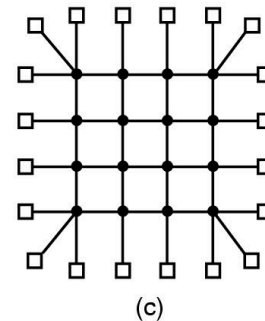
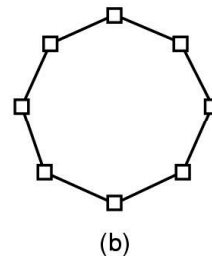
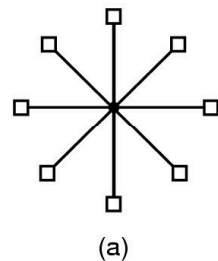
		CPU					
		0	1	2	3	4	5
Time slot	0	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
	1	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
	2	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
	3	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆
	4	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
	5	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
	6	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
	7	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆

Multicomputers

Definition: tightly-coupled CPUs that do not share memory

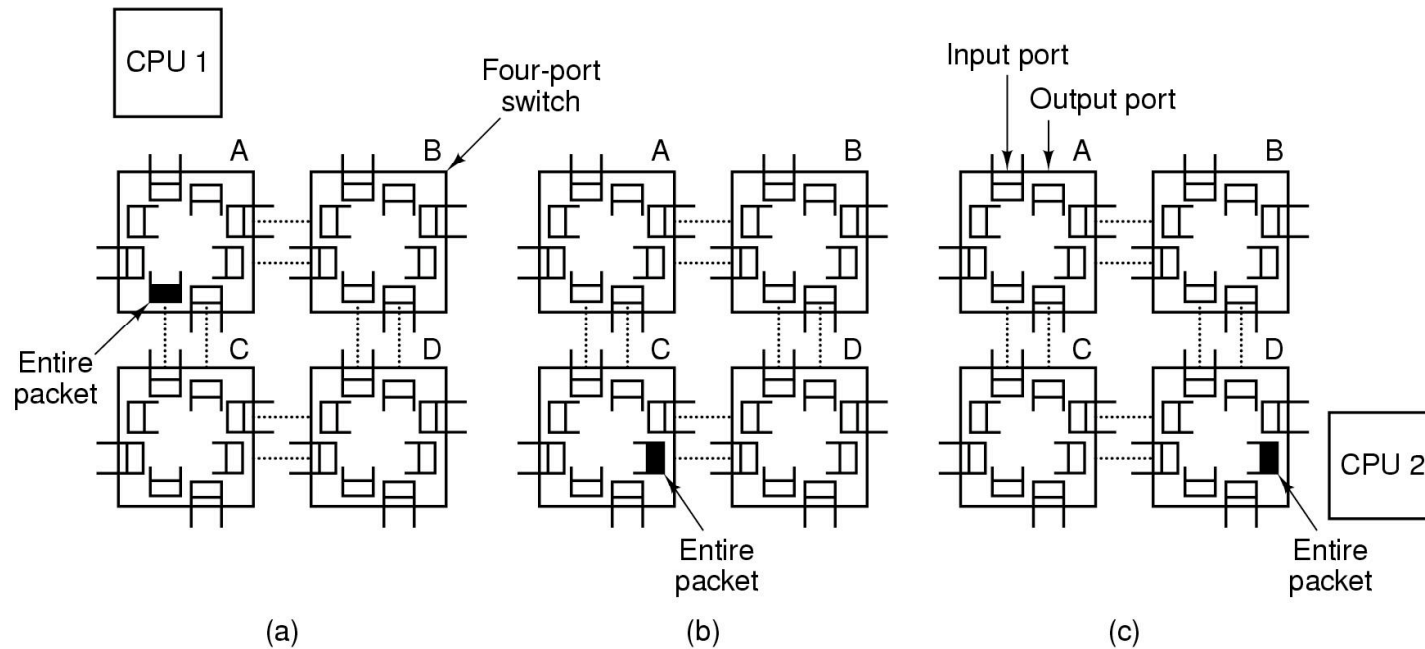
- Also called:
 - Cluster Computers
 - Cluster of Workstations (COWS)

Topologies



Multicomputer Hardware

Store-and-forward packet switching:

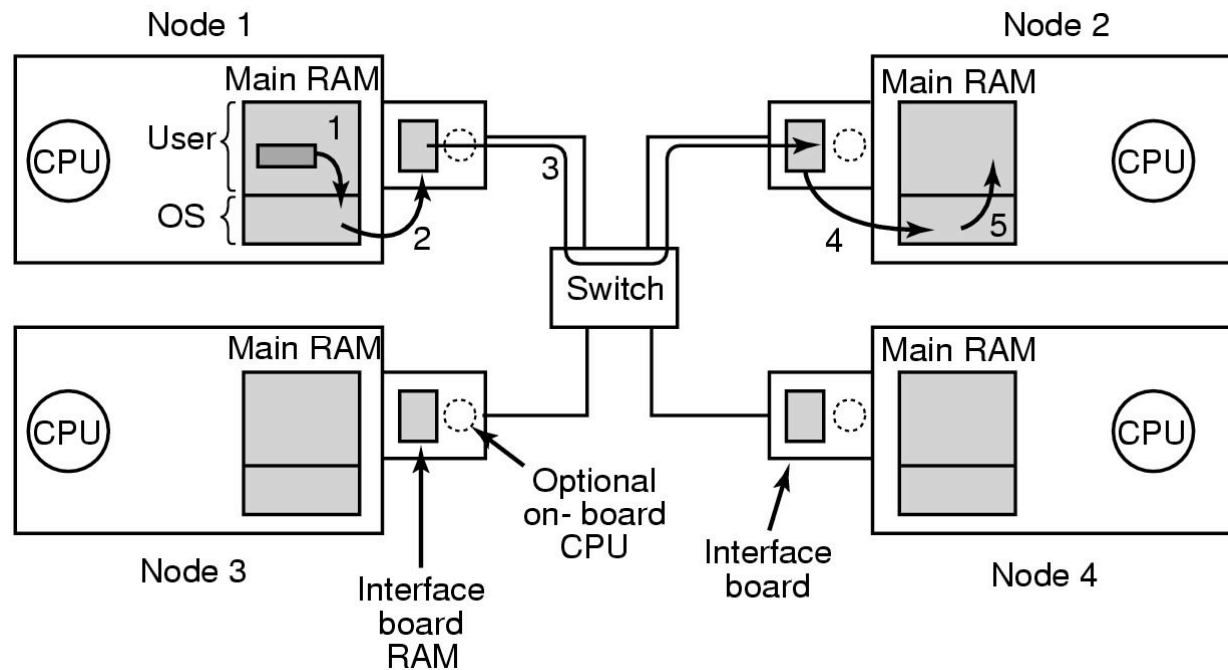


Circuit switching:

- Establish circuit from CPU 1 to CPU 2, then send bits through circuit

Multicomputer Hardware

Network Interface Card (NIC)



Low-level Communication Software

If a user process needs to send packets

- Map network interface card to address space of process
 - (to avoid buffer copying)

If multiple user processes need to send packets

- Map NIC to address space of all processes
- Add mutex to mediate access to NIC buffers

If kernel also needs to send/receive packets

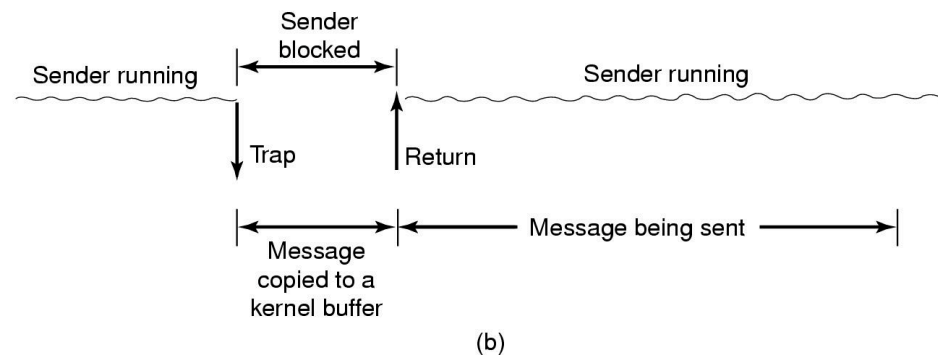
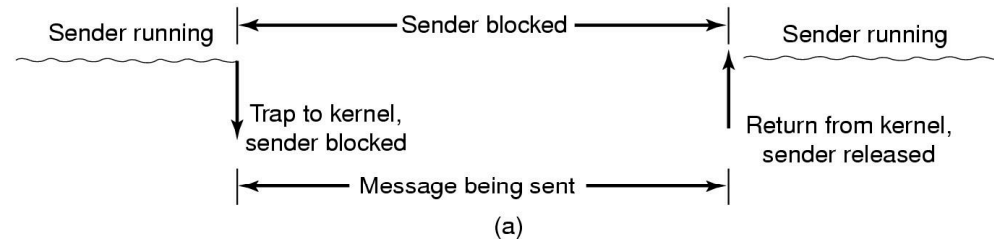
- Have a second NIC reserved for the kernel

User-Level Communication Software

Messaging

- `Send(dest, &messagePtr)`
- `Receive(addr, &messagePtr)`

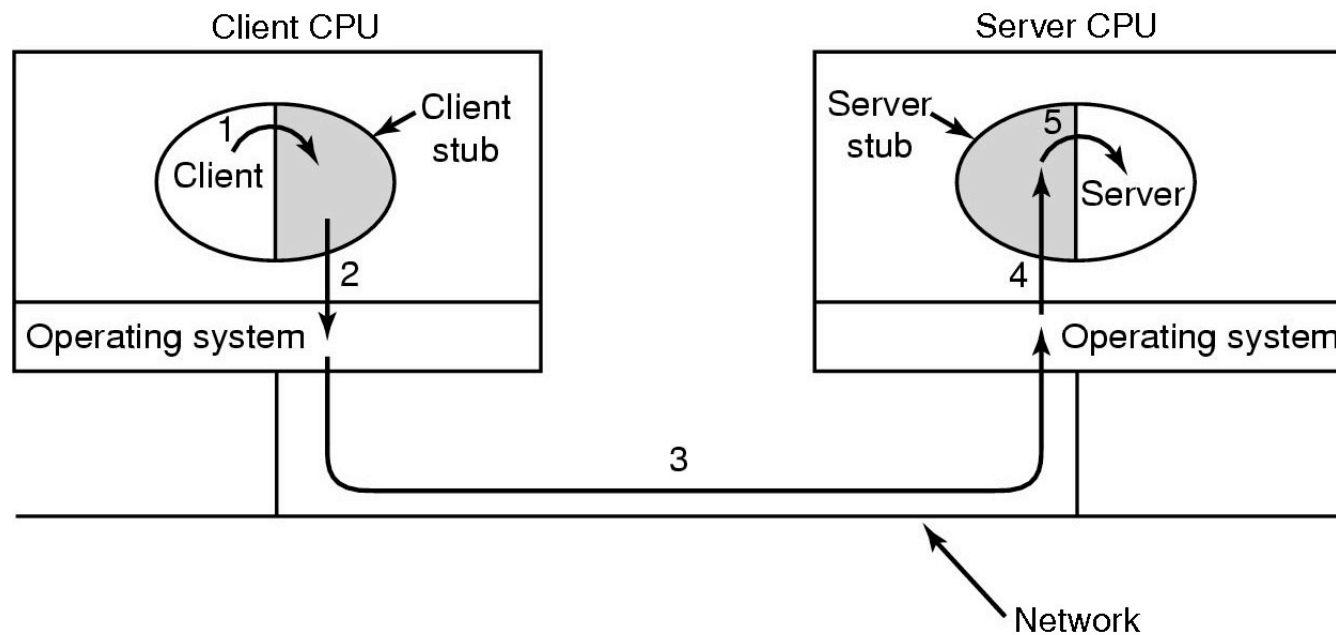
Blocking vs. nonblocking



User-level Communication Software

Remote Procedure Calls (RPC)

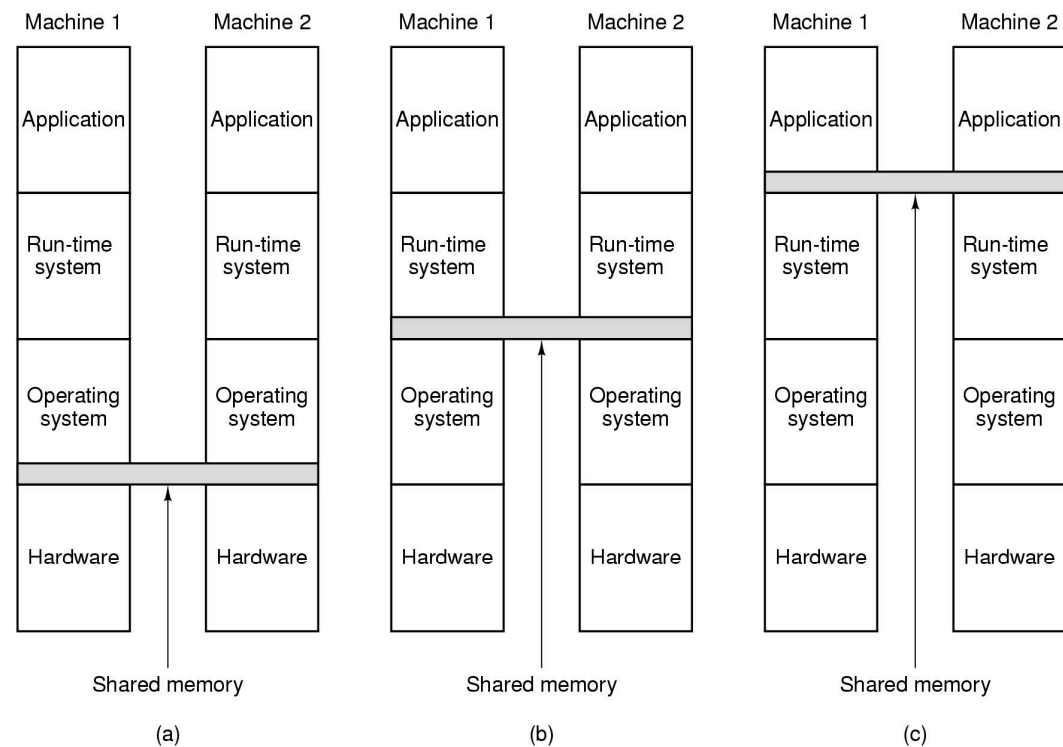
- Call a procedure on a remote machine



- Implementation issues:
 - Pointers/references (call-by-reference turns into copy-restore)
 - Sizes of arrays, etc in weakly-typed languages
 - Types of parameters (printf, for example)
 - Global variables
 - copy
 - don't use

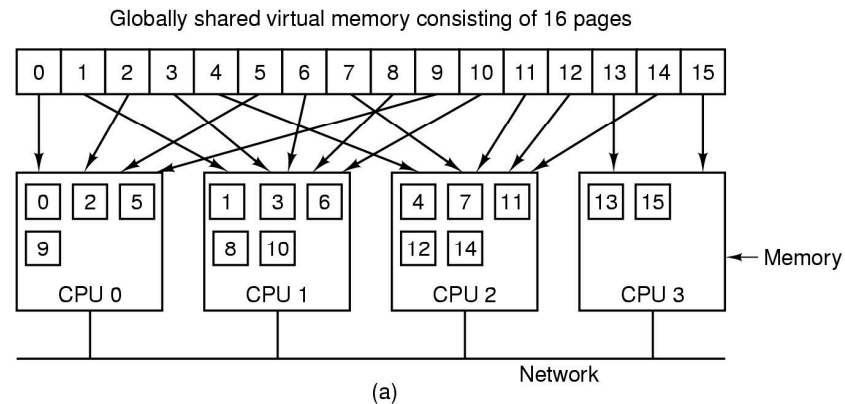
Distributed Shared Memory

When referencing non-local page, trap causes DSM to retrieve page from machine that has it.

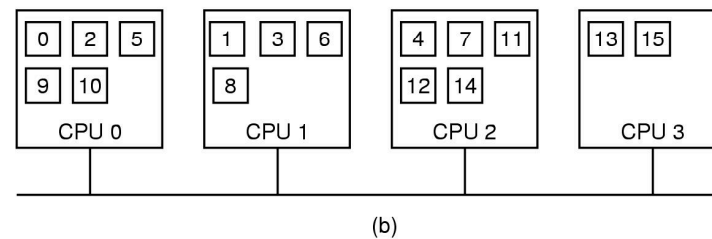


Distributed Shared Memory

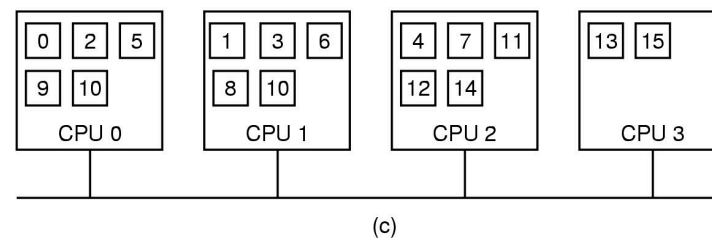
Pages distributed among 4 machines



After CPU 1 references page 10



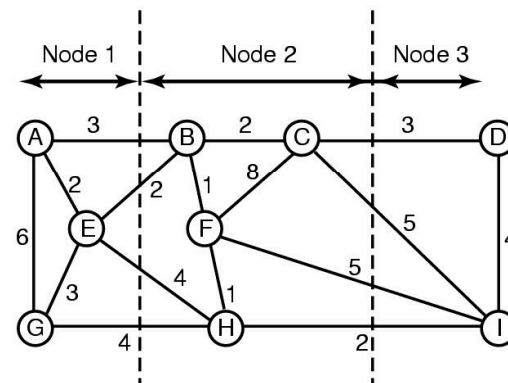
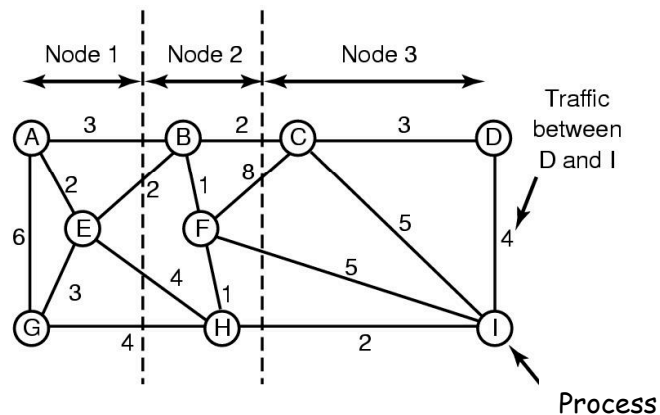
If page 10 is read-only and replication is used



Multicomputer Scheduling

Load balancing: Assigning processors to nodes

- Given process that communicate with one another, find clusters of processes that communicate mostly within the cluster and not outside the cluster



Load Balancing

Sender-initiated Distributed Heuristic

- Newly-created process starts on same node that created it
- But, if overloaded (by some metric)
 - Randomly ask another node its load
 - If low, send the process to it
 - If not, choose another node
 - Eventually, give up and keep the process

Receiver-initiated Distributed Heuristic

- When process finishes, system checks load
- If underloaded (by some metric)
 - Randomly ask another node its load
 - If high, receive a process from it
 - If not, choose another node
 - Eventually, give up and be underloaded

Bidding

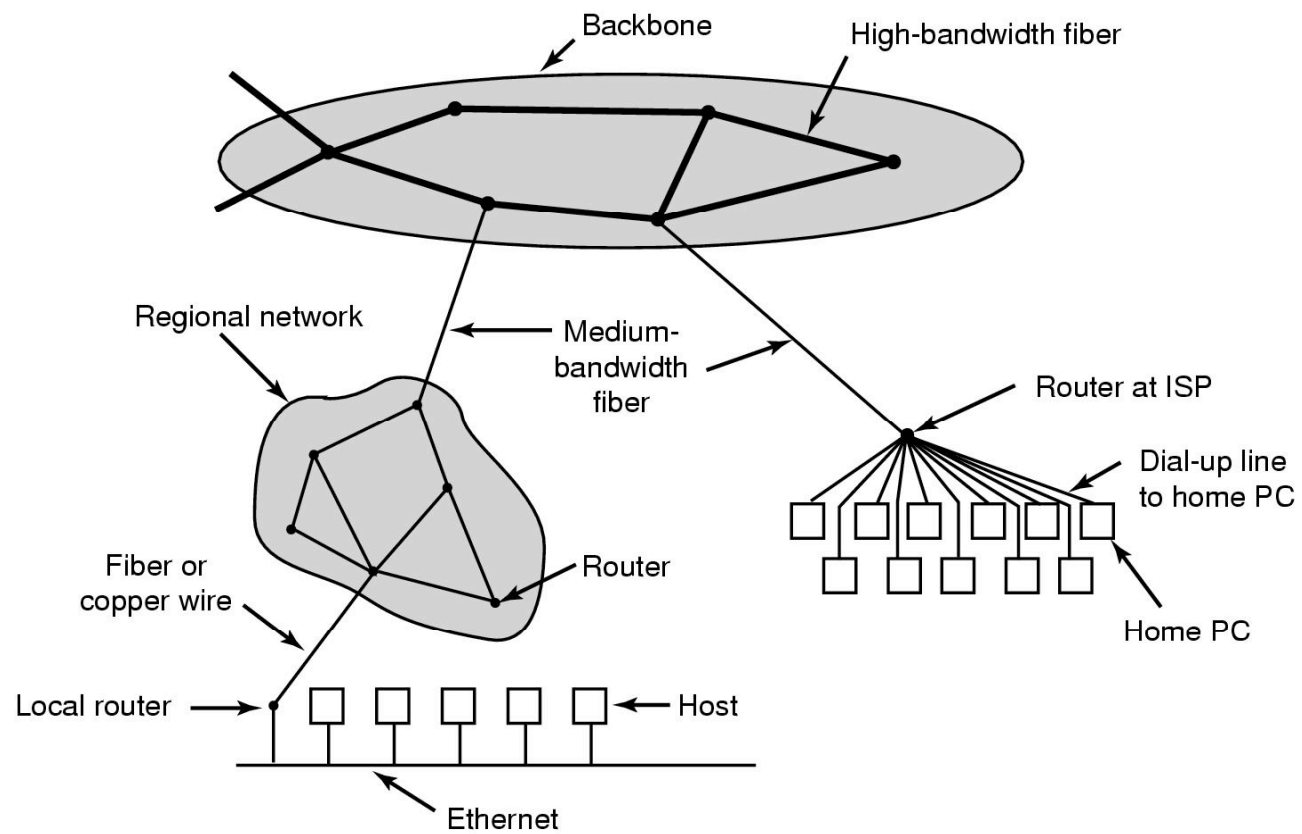
- Processes buy CPU time
 - When creating process, get bids from nodes and choose best node
- Nodes auction off CPU time (specifying speed, features, etc).

Distributed Systems

Definition: loosely coupled computers that don't share memory

Item	Multiprocessor	Multicomputer	Distributed System
Node configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly all different
File systems	One, shared	One, shared	Each node has own
Administration	One organization	One organization	Many organizations

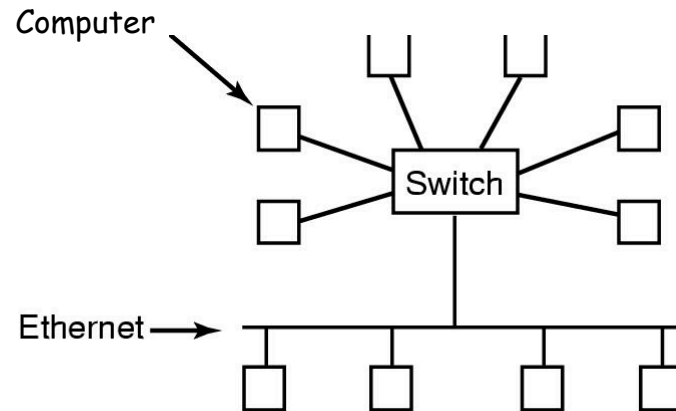
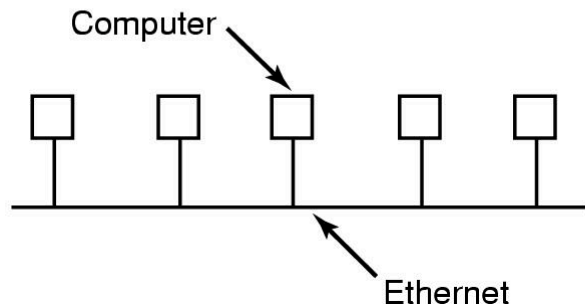
The Internet



Network Hardware

Ethernet

- Signals propagate through the *ether*
- Carrier-Sense Multiple Access with Collision Detect (CSMA/CD)
- All NICs connected electrically
- When want to send, listen to see if carrier in use
- If not, start transmitting
 - If two machines transmit at once, each notices collision, waits random time (within range) and tries again
 - If still collision, exponential backoffs



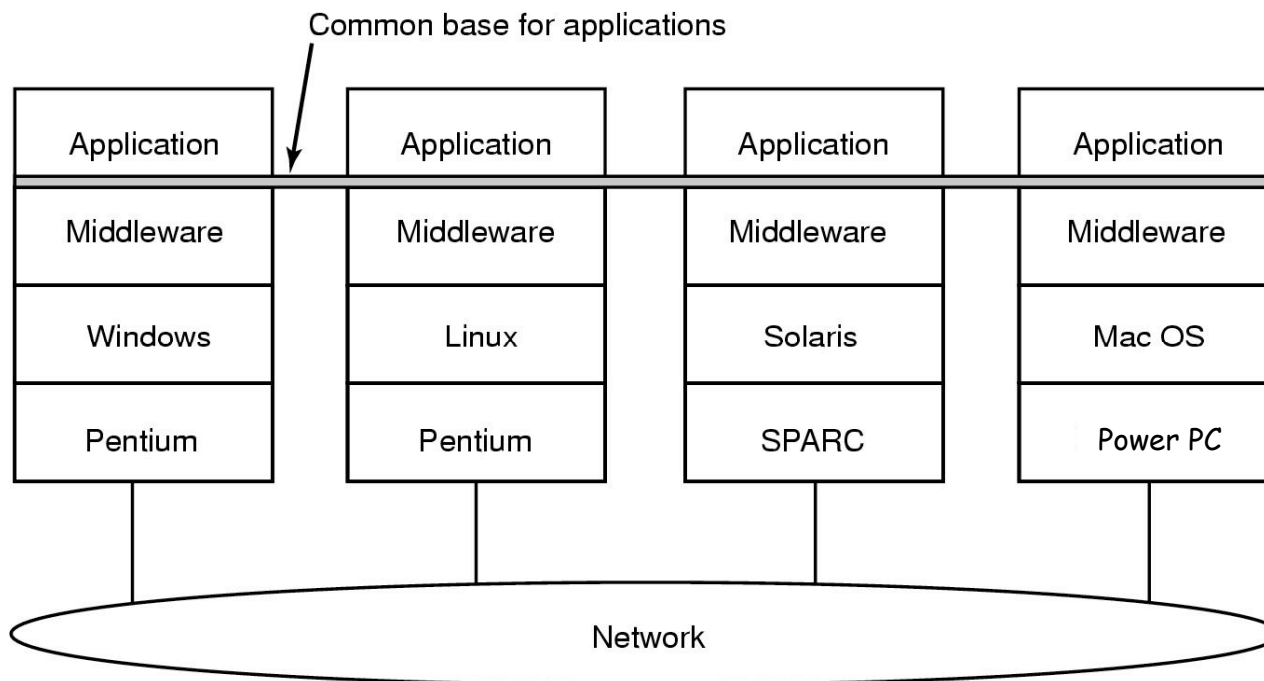
Network Services

		Service	Example
Connection-oriented	{	Reliable message stream	Sequence of pages of a book
		Reliable byte stream	Remote login
		Unreliable connection	Digitized voice
Connectionless	{	Unreliable datagram	Network test packets
		Acknowledged datagram	Registered mail
		Request-reply	Database query

Distributed Systems

Middleware: a layer of software on top of the OS

- Provide consistency across multiple OS types

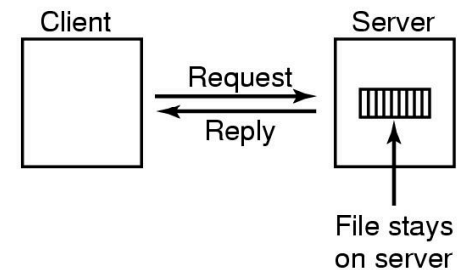
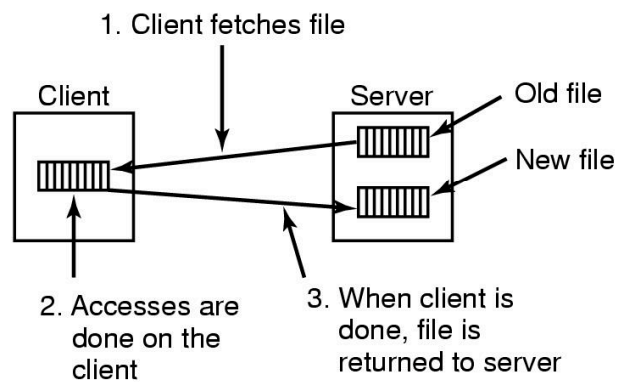


File System-based Middleware

Looks like great big filesystem

Transfer model

- upload/download
- Remote access

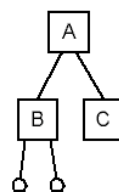


File System-based Middleware

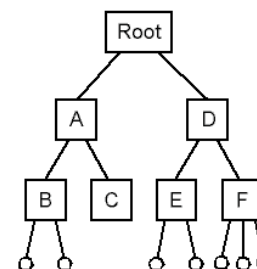
Naming

- Machine + path naming
 - /machine/path
- Mounting remote file system on local file hierarchy
- Single name space looks same on all machines

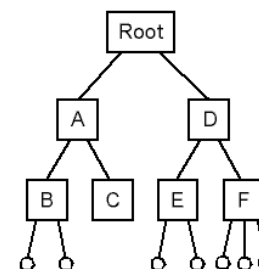
File server 1



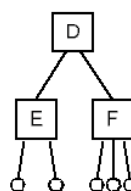
Client 1



Client 1

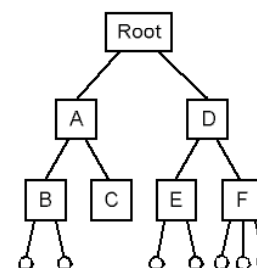


File server 2



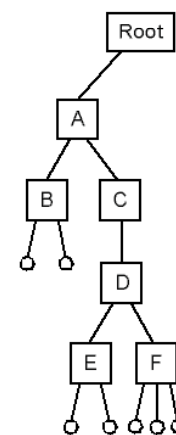
(a)

Client 2



(b)

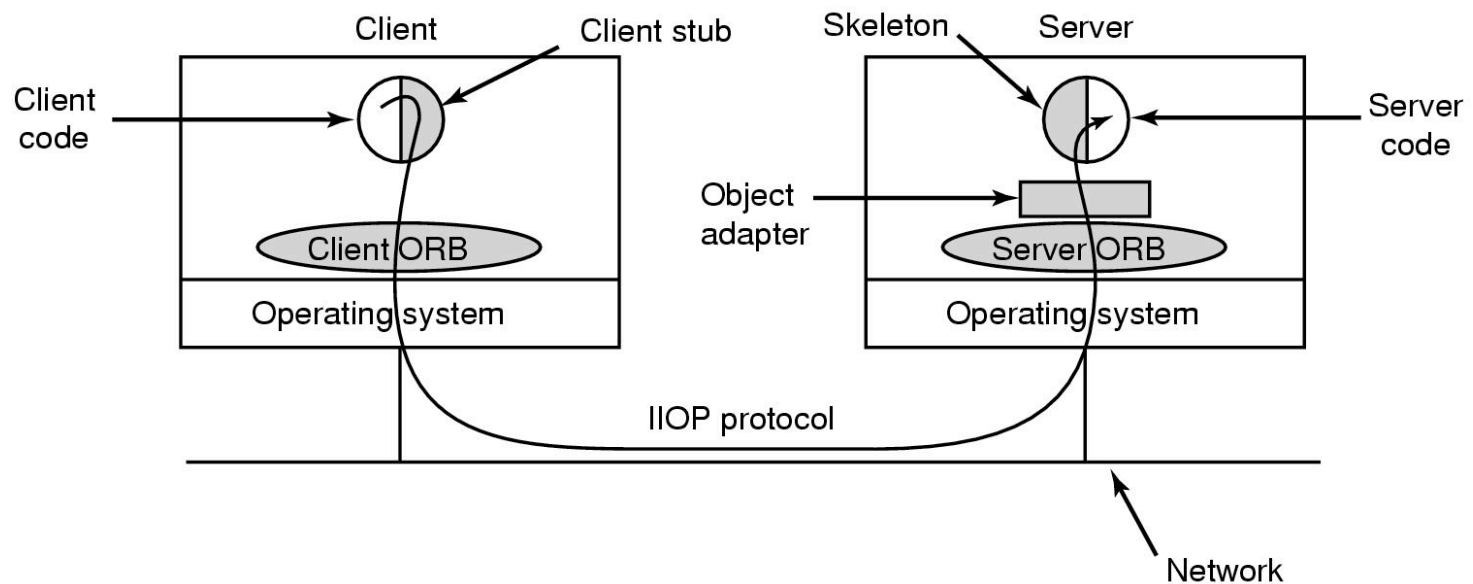
Client 2



(c)

Shared Object-based Middleware

Common Object Request Broker Architecture (CORBA)



Coordination-based Middleware

Linda

- processes communicate via *tuple-space*
- tuple, ordered list of typed data
 - ("abc", 2, 5)
 - ("matrix-1", 1, 6, 3.14)
 - ("family", "is-sister", "Stephany", "Roberta")
- Operations:
 - out: puts tuple into tuple space
 - in, retrieves tuple from tuple space and removes it (atomic, can use wildcards)
 - block if required tuple not present
 - read, like in, but doesn't remove tuple from tuple space
 - eval: used for creating parallel processes

Coordination-based Middleware

Jini

- Devices join network and register services
- Same idea as Linda of tuple-spaces: JavaSpaces
 - Write: put a new entry into the JavaSpace
 - Read: copy an entry that matches a template
 - Take: copy and remove entry that matches a template
 - Notify: notify caller when a matching entry is written
- Read and take can have timeout