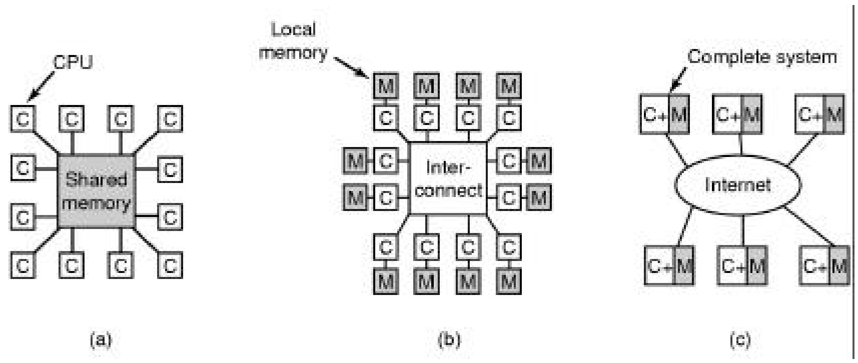
Lecture Overview

- Multiple processors
  - Multiprocessors
    - UMA versus NUMA
    - Hardware configurations
    - OS configurations
    - Process scheduling
  - Multicomputers
    - Interconnection configurations
    - Network interface
    - User-level communication
    - Distributed shared memory
    - Load balancing
  - Distributed Systems

Multiple Processors



Continuous need for faster computers

a) Shared memory multiprocessor
b) Message passing multicomputer
c) Wide-area distributed system

Multiprocessor System

Definition

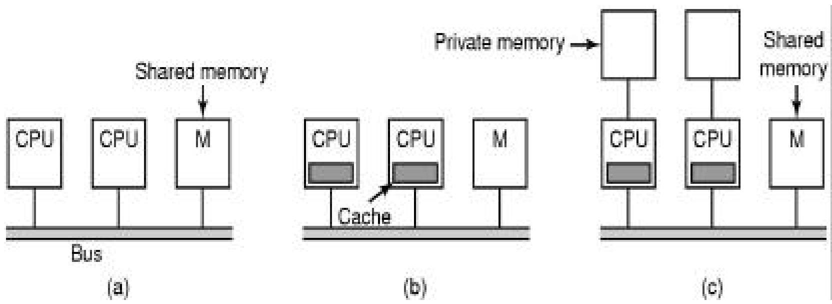
A computer system in which two or more CPUs share full access to a common RAM

Multiprocessor System

Two types of multiprocessor systems

- *Uniform Memory Access (UMA)*
  - All memory addresses are reachable as fast as any other address
- *Non-uniform Memory Access (NUMA)*
  - Some memory addresses are slower than others

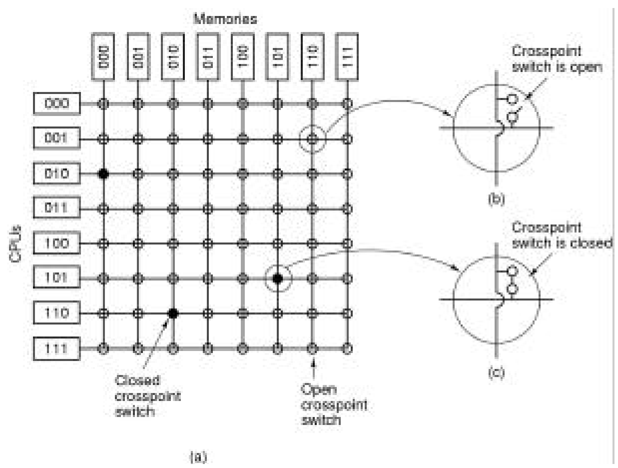
UMA Multiprocessor Hardware



UMA bus-based multiprocessors

a) CPUs communicate via bus to RAM
b) CPUs have a local cache to reduce bus access
c) CPUs have private memory, shared memory access via bus

UMA Multiprocessor Hardware

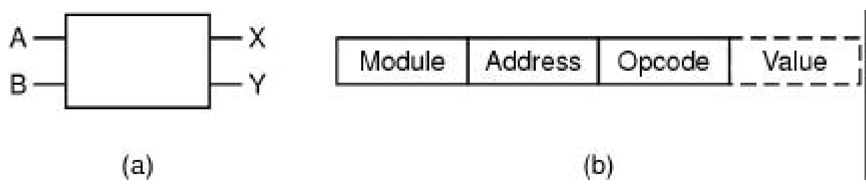


UMA multiprocessor using a crossbar switch

– Alleviates bus access problems, but is expensive (grows as n^2)

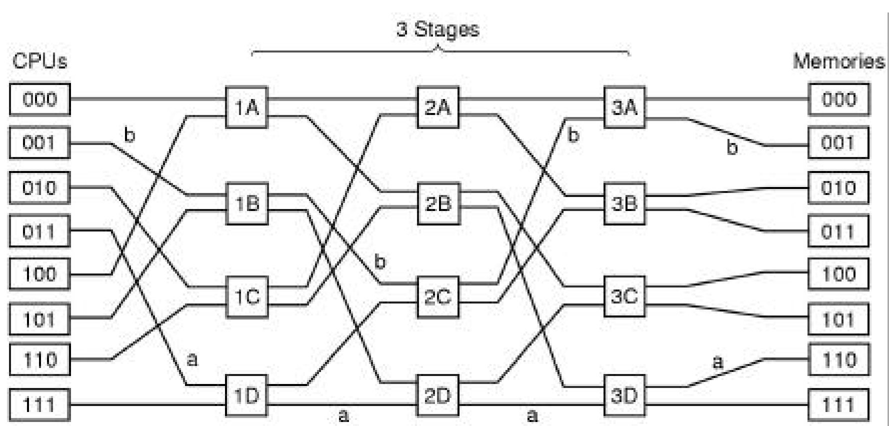
UMA Multiprocessor Hardware

- UMA multiprocessors using multistage switching networks can be built from 2 × 2 switches
  - Input to switches in in the form of a message



a) 2 × 2 switch b) Message format

UMA Multiprocessor Hardware



UMA omega switching network

- Less costly than crossbar switch

NUMA Multiprocessor Hardware

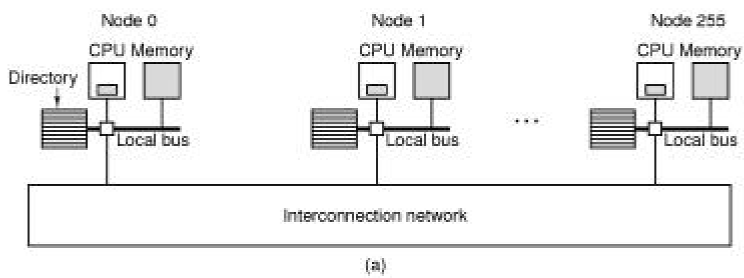
NUMA Multiprocessor Characteristics

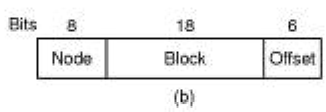
- Single address space visible to all CPUs
- Access to remote memory via commands
  - `LOAD`
  - `STORE`
- Access to remote memory slower than to local

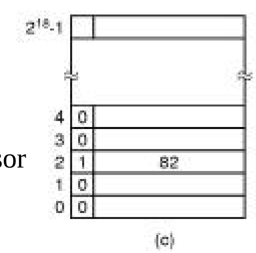
NC-NUMA versus CC-NUMA

– No cache versus cache-coherent

NUMA Multiprocessor Hardware

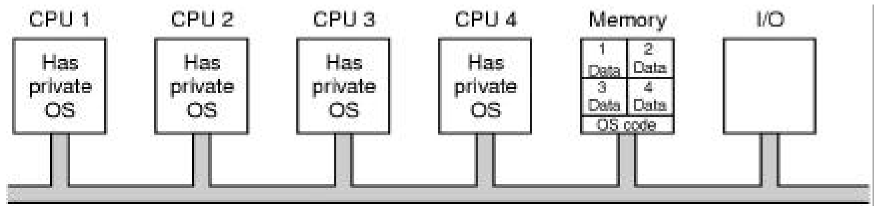






a) 256-node directory-based multiprocessor
b) Fields of 32-bit memory address
c) Directory at node 36

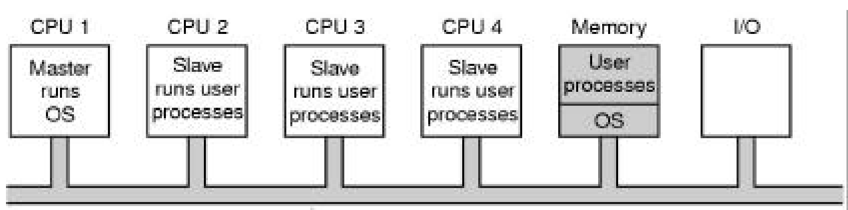
Multiprocessor OS Paradigms



Each CPU has its own operating system

- Allows sharing of devices
- Efficient process communication via shared memory
- Since each OS is independent
  - No process sharing among CPUs
  - No page share among CPUs
  - Makes disk buffering very difficult

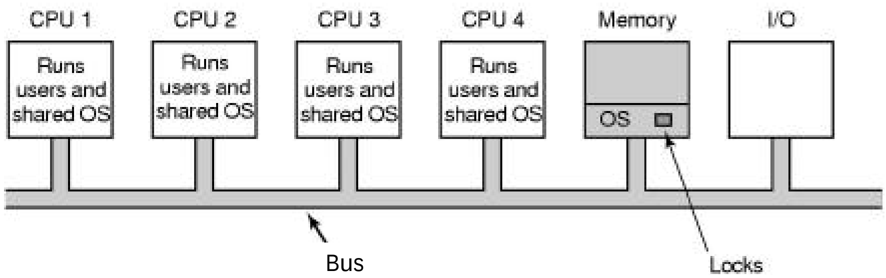
Multiprocessor OS Paradigms



Master-slave multiprocessors

- OS and all tables are on one CPU
  - Process sharing, so no idle CPUs
  - Page sharing
  - Disk buffers
- Master CPU becomes a bottleneck

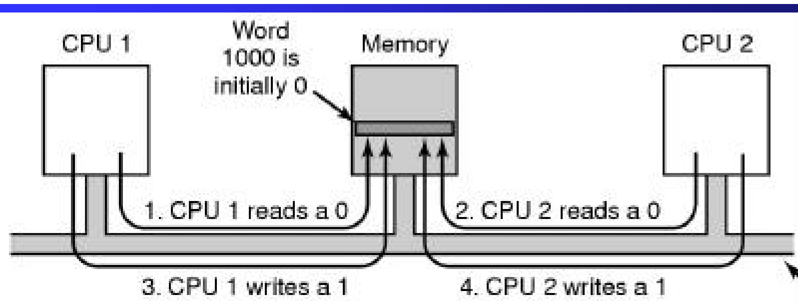
Multiprocessor OS Paradigms



Symmetric multiprocessors

- One copy of OS, but any CPU can run it
- Balances processes and memory dynamically
- Eliminates bottleneck
- More complicated requires reasonably fine-grained synchronization to avoid bottleneck, deadlock issues

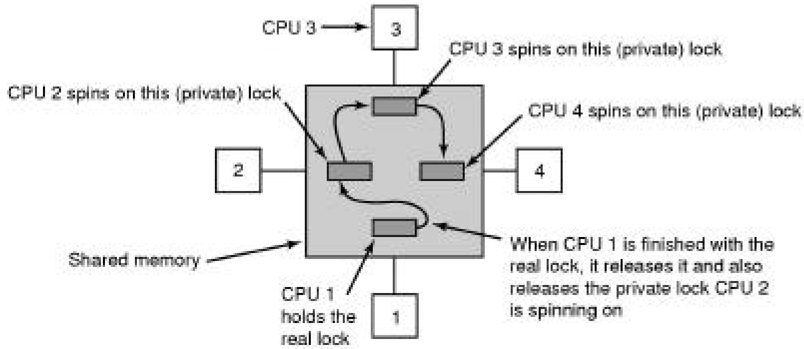
Multiprocessor Synchronization



Locking

- Test-and-set instructions fail is bus cannot be locked
- Create contention for bus, caching doesn't help since test-and-set is a write instruction
- Could read first, before test-and-set; also use exponential back-off

Multiprocessor Synchronization



Locking with multiple private locks

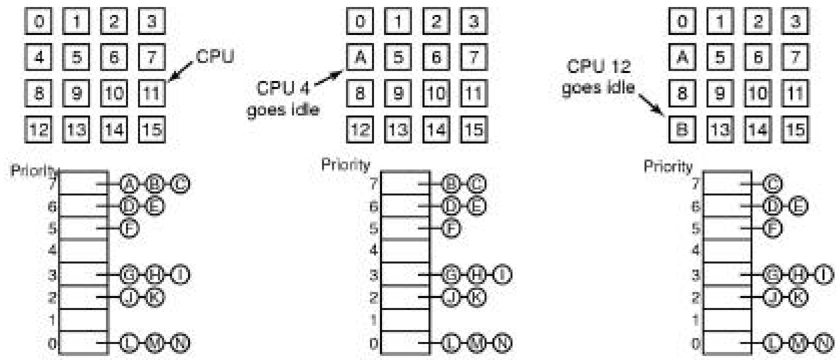
- Try to lock first, if failed, create private lock and put it at the end of a list of CPUs waiting for lock
- Lock holder releases original lock and frees private lock of first CPU on waiting list

Multiprocessor Synchronization

Spinning versus Switching

- In some cases CPU must wait
  - For example, must wait to acquire ready list
- In other cases a choice exists
  - Spinning wastes CPU cycles
  - Switching uses up CPU cycles also
  - Possible to make separate decision each time locked mutex encountered

Multiprocessor Scheduling

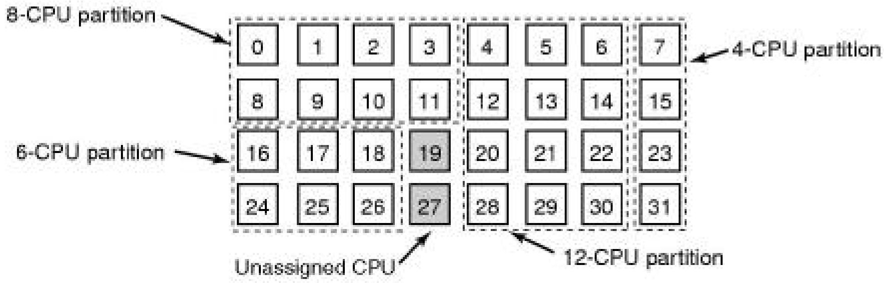


- Timesharing
  - Non-related processes
  - Note use of single data structure for scheduling
    - Provides automatic load balancing
    - Contention for process list

Multiprocessor Scheduling

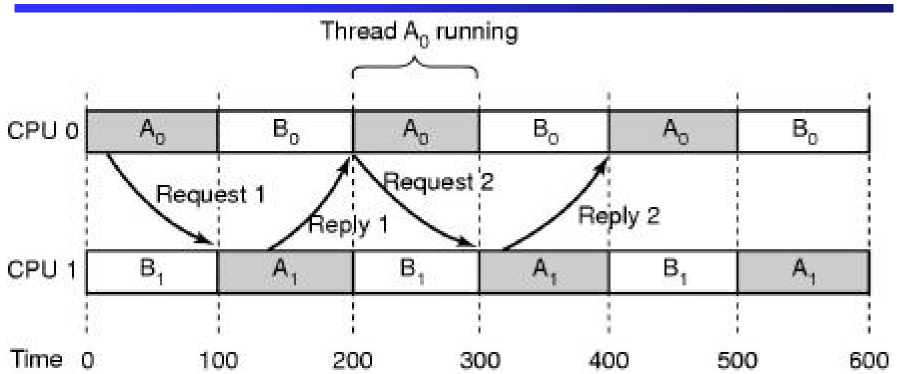
- What about processes holding a spin lock?
  - Does not make sense to block such a process
- In general, all CPUs are equal, but some are more equal than others
  - The CPU cache may have cached blocks of process that was previously running on it
  - The CPU TLB may have cached pages of a process that was previously running on it
  - Use *affinity scheduling* to try to keep processes on the same CPU

Multiprocessor Scheduling



- Space sharing
  - Related processes/threads
  - Multiple threads at same time across multiple CPUs
    - A group of threads is created and assigned to CPUs as a block
    - Group runs until completion
  - Eliminates multiprogramming and context switches
  - Potentially wastes CPU time, when CPUs left idle

Multiprocessor Scheduling



- Potential communication problem when scheduling threads independently
  - A_0 and A_1 both belong to process A
  - Both running out-of-phase

Multiprocessor Scheduling

- Need to avoid wasting idle CPUs and out-of-phase thread communication
- Solution: Gang Scheduling
  - Groups of related threads scheduled as a unit (a gang)
  - All members of gang run simultaneously on different timeshared CPUs
  - All gang members start and end time slices together

Multiprocessor Scheduling

CPU

Time slot	0	1	2	3	4	5
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

Gang scheduling

- All CPUs scheduled synchronously
- Still has some idle time and out-of-phase, but reduced

Multicomputers

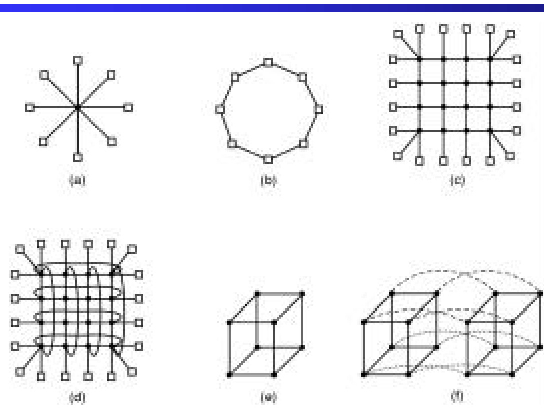
Definition

Tightly-coupled CPUs that do not share memory

Also known as

- Cluster computers
- Clusters of workstations (COWs)

Multicomputer Interconnection



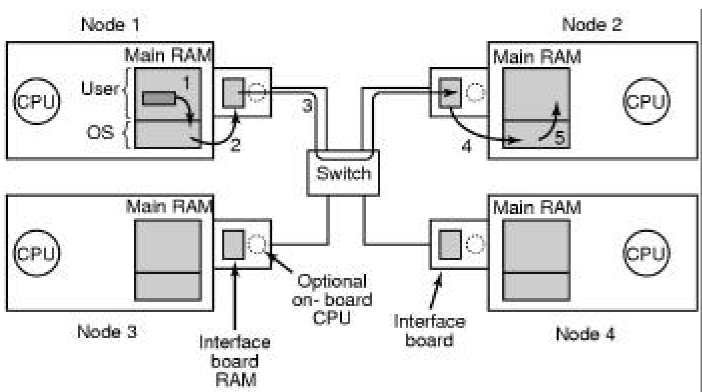
Interconnection topologies

a) single switch
b) ring
c) grid
d) double torus
e) cube
f) hypercube

Multicomputer Interconnection

- Switching schemes
  - Store-and-forward packet switching
    - Send a complete packet to first switch
    - Complete packet is received and forward to next switch
    - Repeated until it arrives at destination
    - Increases latency due to all the copying
  - Circuit switching
    - Establishes a path through switches (i.e., a circuit)
    - Pumps packet bits non-stop to destination
    - No intermediate buffer
    - Requires set-up and tear-down time

Multicomputer Network Interface

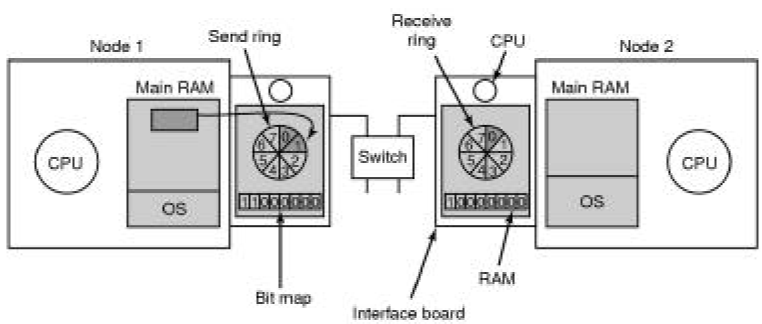


- Interface boards usually contain buffer for packets
  - Needs to control flow onto interconnection network when sending and receiving packets
- Interface boards can use DMA to copy packets into main RAM

Multicomputer Network Interface

- Must avoid unnecessary copying of packets
  - Problematic if interface board is mapped into kernel memory
- Map interface board into process memory
- If several processes are running on node
  - Each needs network access to send packets …
  - Must have sharing/synchronization mechanism
- If kernel needs access to network …
- One possible solution is to use two network boards
  - One for user space, one for kernel space

Multicomputer Network Interface



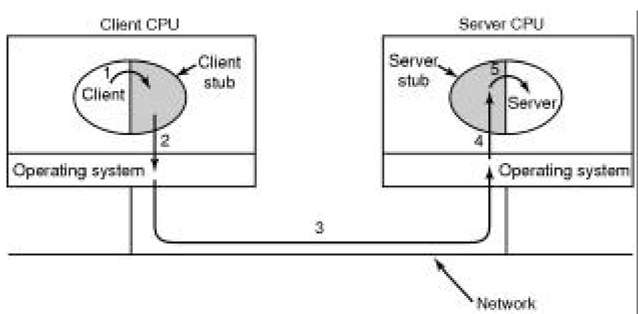
Node to network interface communication

- Complicated when user is controlling DMA
- If interface has its own CPU, then must coordinate with man CPU
  - Use send & receive rings

Multicomputer User-Level Communication

- Bare minimum, `send` and `receive`
  - Blocking versus non-blocking
    - Choices
      - Blocking send (CPU idle during message transmission)
      - Non-blocking send with copy (CPU time waste for extra copy)
      - Non-blocking send with interrupt (makes programming difficult)
      - Copy on write (extra copy eventually)
  - Pop-up thread
    - Creates a thread spontaneously when a message arrives
  - Active messages
    - Message handler code is run directly in the interrupt handler

Multicomputer User-Level Communication



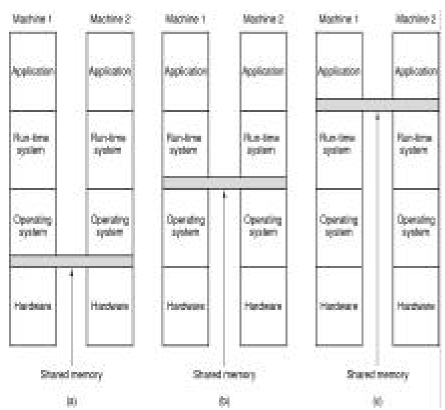
- The `send`/`receive` primitives are wrong paradigm
- Remote procedure call (RPC) maintains procedural paradigm
  - Breaks a procedure into client and server

Multicomputer User-Level Communication

RPC implementation issues

- Cannot pass pointers
  - Call by reference becomes copy-restore (but might fail)
- Weakly typed languages
  - Client stub cannot determine size
- Not always possible to determine parameter types
  - Think about `printf(…)` with variable parameters
- Cannot use global variables
  - May get moved to remote machine

Multicomputer Distributed Shared Memory



- Layers where shared memory can be implemented
  - Hardware (multiprocessors)
  - Operating system

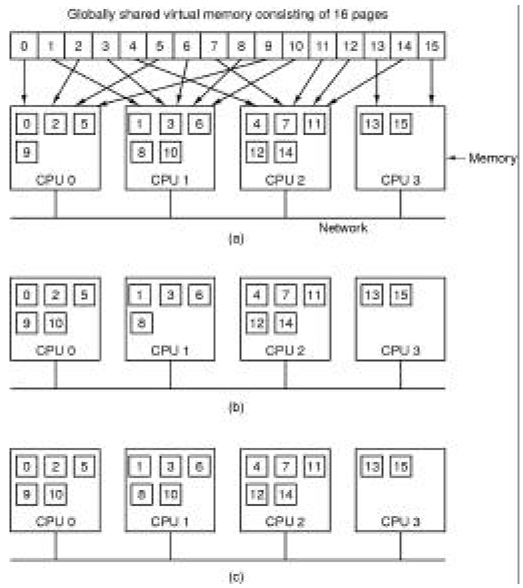
Multicomputer Distributed Shared Memory

Replication

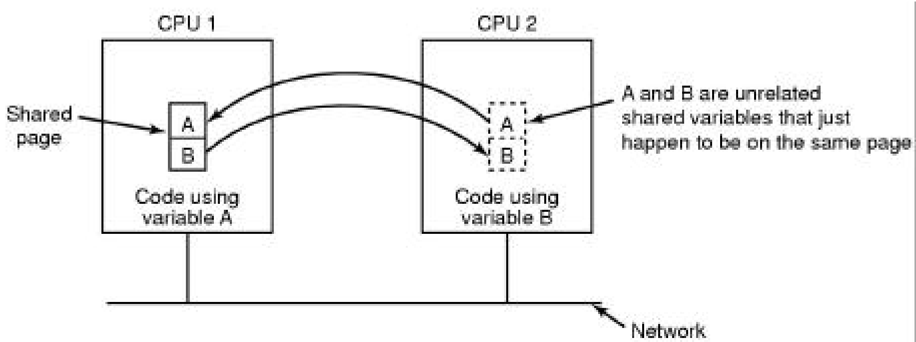
a) Pages distributed on 4 machines

b) CPU 0 reads page 10

c) CPU 1 reads page 10



Multicomputer Distributed Shared Memory

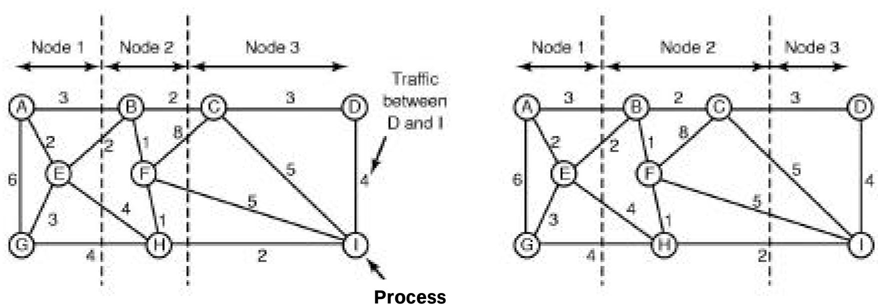


- False Sharing
- Must also achieve sequential consistency (i.e., cache coherency problem)

Multicomputer Process Scheduling

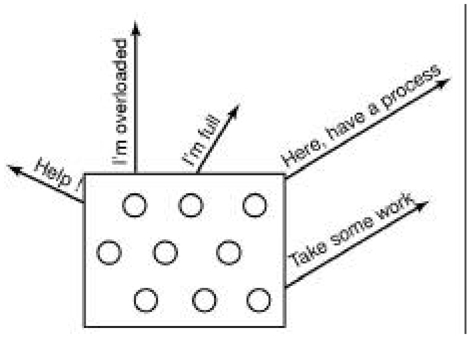
- On a multicomputer, each node has its own memory and its own set of processes
  - This is very similar to a uniprocessor, so process scheduling can use similar algorithms
    - Unless you have multiprocessors as nodes
- The critical aspect of multicomputer scheduling is allocating processes to processors
  - Processor allocation algorithms
    - Use various metrics to determine process "load" and how to properly allocate processes to processors
  - These are "load" balancing algorithms

Multicomputer Load Balancing



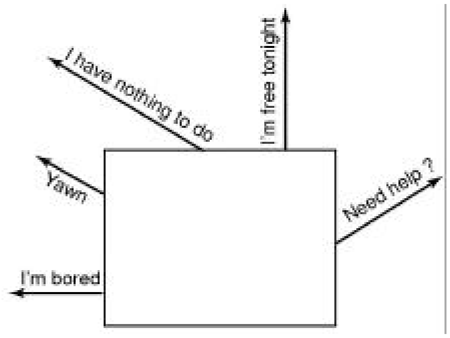
- Graph-theoretic deterministic algorithm
  - Know processes, CPU and memory requirements, and average communication traffic among processes
  - Partition graph to minimize network traffic and to meet constraints on CPU and memory

Multicomputer Load Balancing



- Sender-initiated distributed heuristic algorithm
  - Overloaded sender probes for underloaded node
  - Searches come during heavy loads, which adds more load

Multicomputer Load Balancing



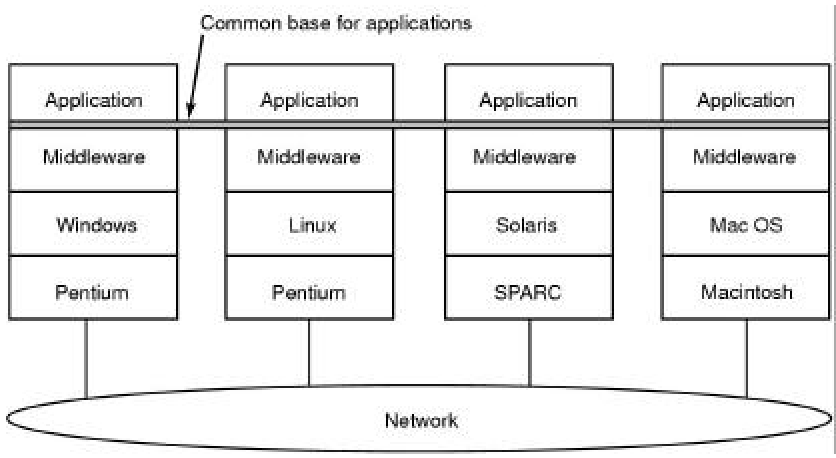
- Receiver-initiated distributed heuristic algorithm
  - Under loaded sender probes for overloaded node
  - Searches come during lower loads

Distributed Systems

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

Comparison of three kinds of multiple CPU systems

Distributed System Middleware



Achieving uniformity with middleware