CSX-306
System Programming
B.Tech (6th Semester)

Dr. Mohit Kumar
Dept of IT
NIT Jalandhar

Objectives:

1. Learn basic concepts of operating systems and system software's
2. Learn the functioning of the principal parts of Compiler

3. To gain the basic concepts of Assemblers, Loader, Linkage Editors, Grammar and automation

Course Outcomes:

1. Enumerate and explain the function of common OS kernel routines that are provided by an OS and accessible from a system programming language.

2. Design, write, and test moderately complicated low level programs using a systems programming language.

3. Create a program that measures or simulates performance and use it to analyze behavior.

System Software

- System Software is the type of software which is the interface between application software and system.
- High/low level languages are used to write the system software.
- System Software maintain the system resources and give the path for application software to run.
- System software runs when system is turned on and stop when system is turned off.
- Example of System software are: Operating System, Compiler, Assembler

Application Software:

- Application Software is type of software which runs as per user request.
- It runs on the platform which is provide by system software.
- High level languages are used to write the application software.
- Its a specific purpose software.
- Example of application software are Photoshop, VLC player, web browser etc.

Difference between System program and application Program

- The main difference between System Software and Application Software is that without system software, system can not run on the other hand without application software, system always runs.

Translators

- Compilers
- Interpreters
- Assemblers

Regardless of what language you use, you eventually need to convert your program (written in high or low level) into a language that the computer can understand.

Two ways for doing that: compile the program or interpret the program

Compilers:

- A compiler is a computer program that translates a program in a source language into an equivalent program in a target language.

  or

- Compilers: Translate a source (human-writable) program to an executable (machine-readable) program.

- Translate the entire program.

- Convert the entire program to machine code, when the syntax errors are removed then converted into the object code

Compilers Continue..

- Requires more main memory
- Neither source nor the compiler are required for execution.
- Slow for debugging and testing.
-
- Execution time is less.

Object code and machine code

- **Object code** is a portion of machine **code** that has not yet been linked into a complete program.
- It is the machine **code** for one particular **library** or module that will make up the completed product.
- **Machine code is** binary (1's and 0's) **code** that can be executed directly by the CPU.

Interpreters:

- An **interpreter** is a computer program that translates and executes instructions written in a computer programming language line-by-line, unit by unit etc.,
-
- Interpreters: Convert a source program and execute it at the same time.

- Translate the program line by line.

- each time the program is executed ,every line is checked for syntax error & then converted to equivalent machine code directly

Interpreters Continue..

- An interpreter needs to be able to analyze, or parse, instructions written in the source language.

- Requires less main memory
- Source program and the interpreter are required for execution.

- Good for fast debugging and testing.

- Execution time is more.

Assemblers:

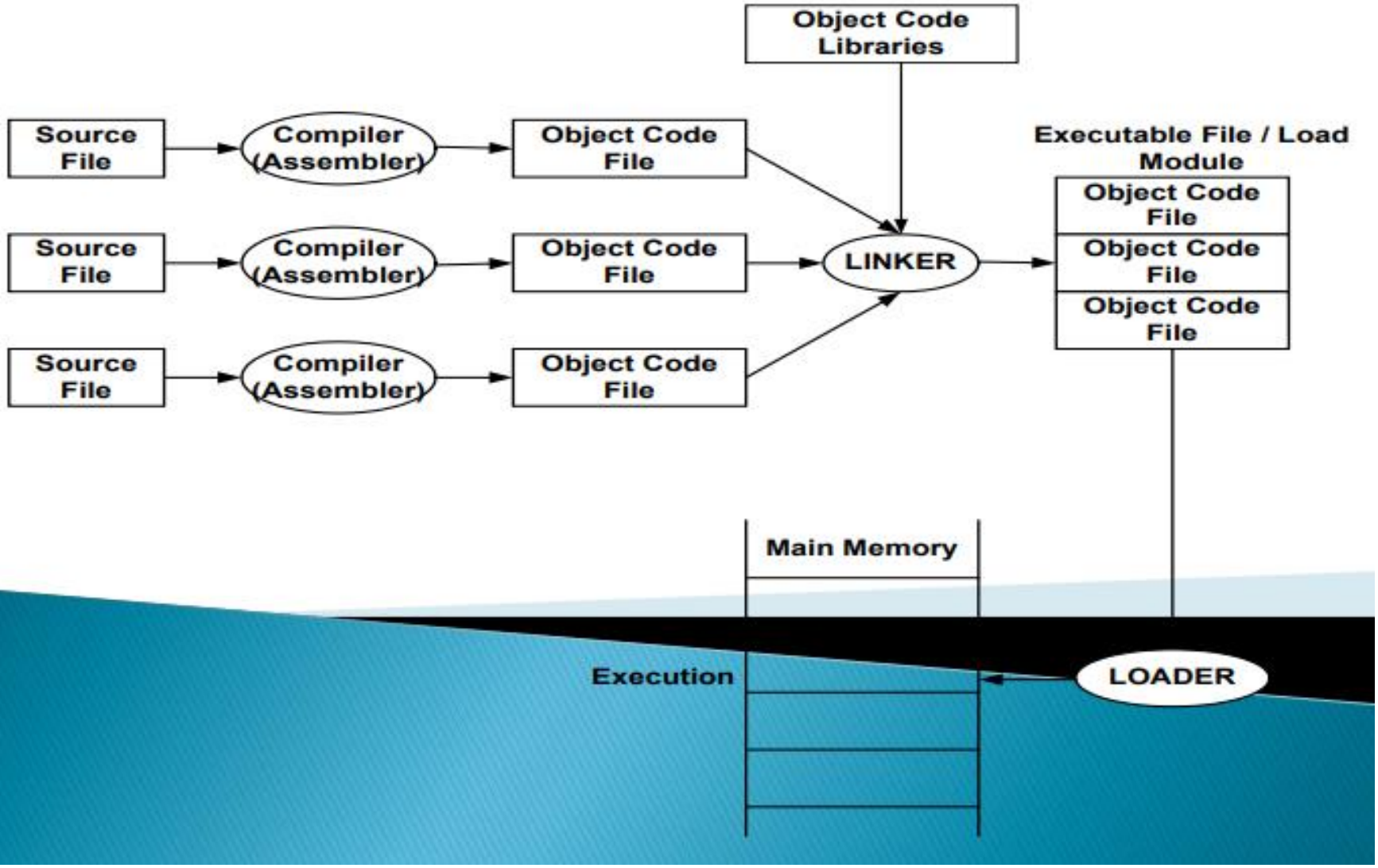
- Programmers found difficult to read and write the program in machine language.
- For a more convenient language they began to use a mnemonic for each machine instruction, which they would translate into machine language.
- Such a mnemonic machine language is now called an assembly language.
- The input to an assembler program is called the source program, the output is a machine language translation (object program).

Linker:

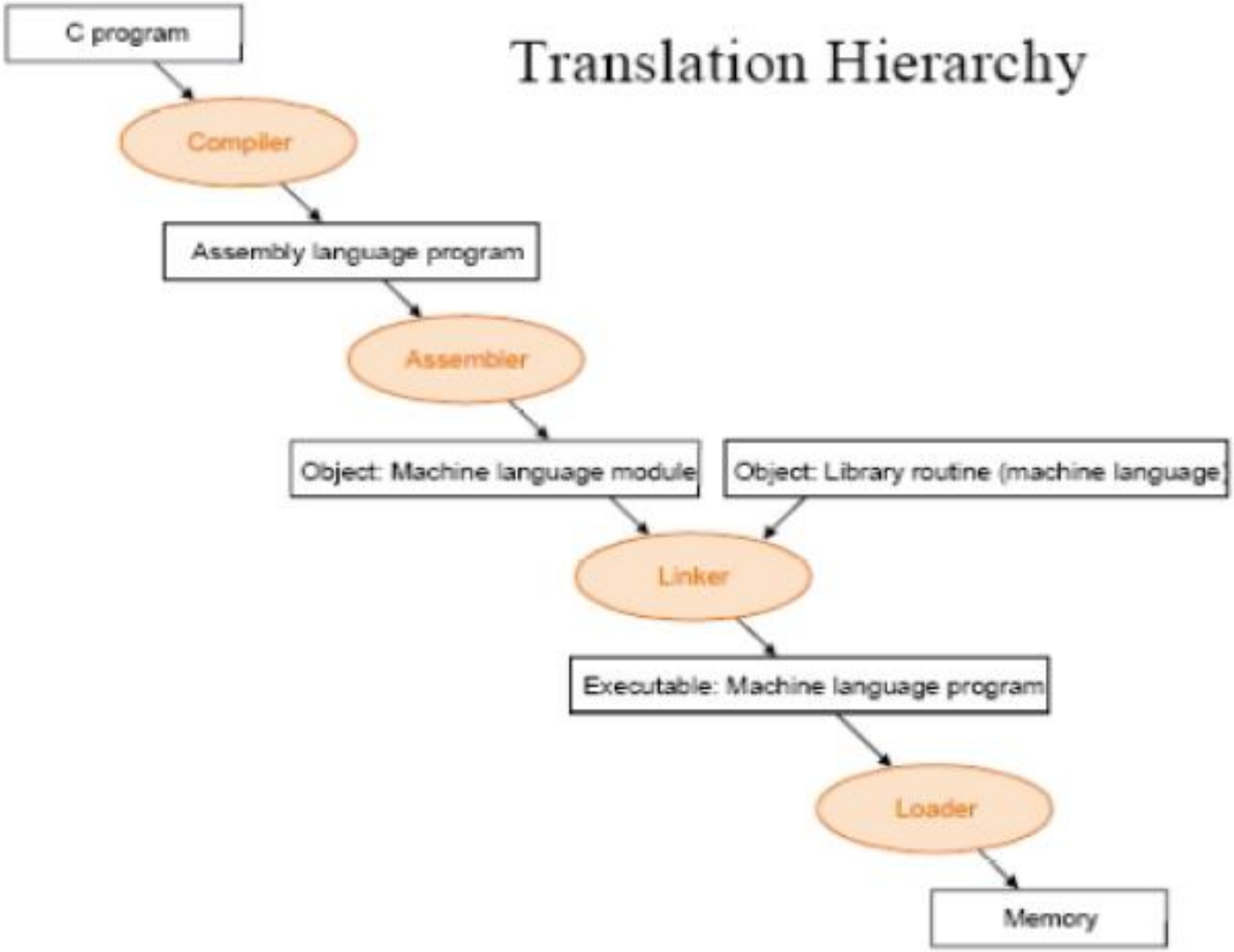
• A program that takes as input the object files of one or more separately compiled program modules, and links them together into a complete executable program, resolving reference from one module to another

Linker/loader

intoducation



Translation Hierarchy



Loader

- A loader is the part of an operating system that is responsible for loading programs into memory, preparing them for execution and then executing them.

- The loader is usually a part of the operating system's kernel and usually is loaded at system boot time and stays in memory until the system is rebooted, shut down, or powered off.

- When the program finished, control must somehow be returned to the operating system.

Loader continue..

• The assembler could place the object program directly in memory and transfer control to it, thereby causing the machine language program to be executed.

•However this would waste memory by leaving the assembler in memory while the user's program was being executed.

•To overcome the problem of wasted memory, system programmers developed another component, called the Loader.

•Loader program is much smaller then the assembler, this makes more memory available to the user's program.

Applications

- Assembly language is typically used in a system's boot code, (BIOS on IBM compatible PC systems and CP/M), the low-level code that initializes and tests the system hardware prior to booting the OS, and is often stored in ROM.

- Some compilers translate high-level languages into assembly first before fully compiling, allowing the assembly code to be viewed for debugging and optimization purposes.

Introduction to Operating Systems

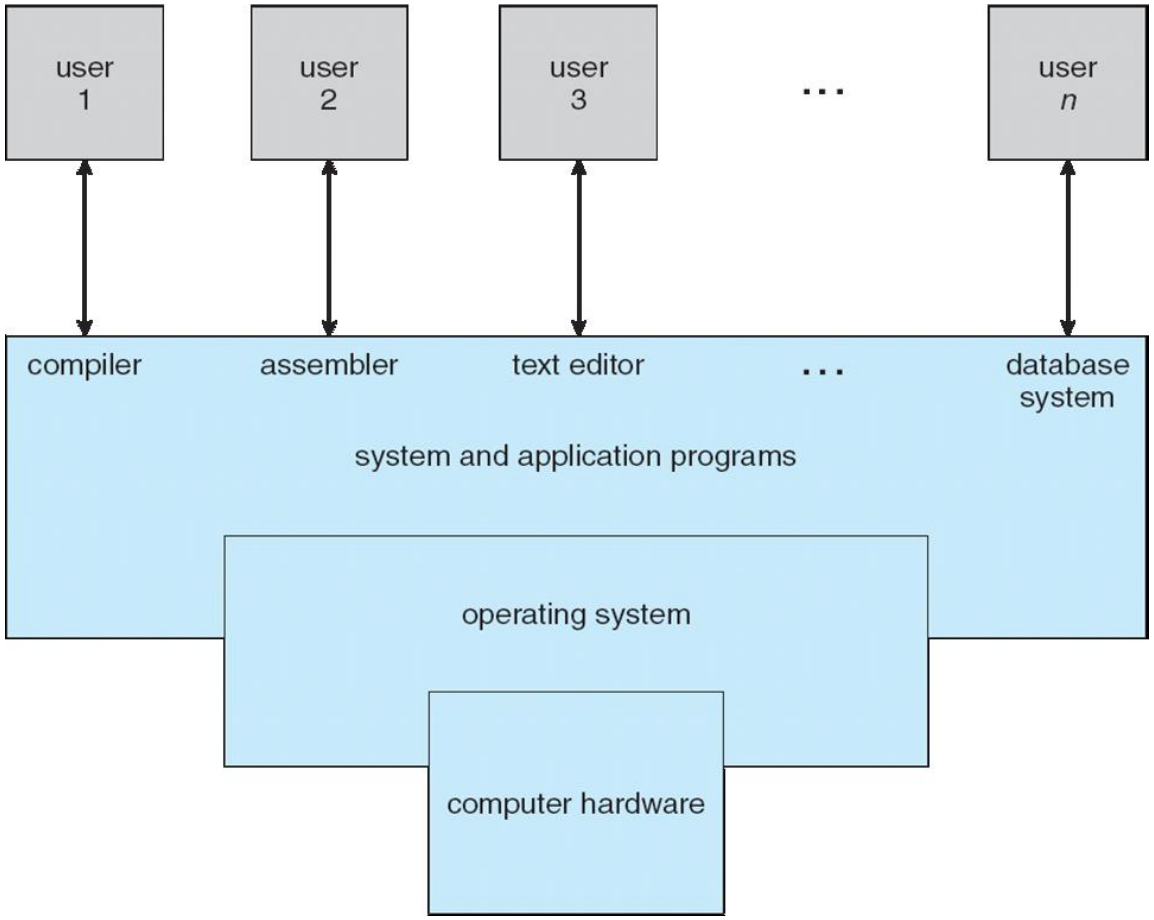
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Four Components of a Computer System



What Operating Systems Do

- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Computer Startup

- When you turn on the power to a computer, the first program that runs is usually a set of instructions kept in ROM.
- This code examines the system hardware to make sure everything is functioning properly.
- This **power-on self test** (POST) checks the CPU, memory, and basic input-output systems (BIOS) for errors and stores the result in a special memory location.
- Once the POST has successfully completed, the software loaded in ROM (sometimes called the BIOS or **firmware**) will begin to activate the computer's disk drives

Continue..

- In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: the **bootstrap loader**.

- The bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation.

Operating System Structure

- **Multiprogramming:** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

Continue..

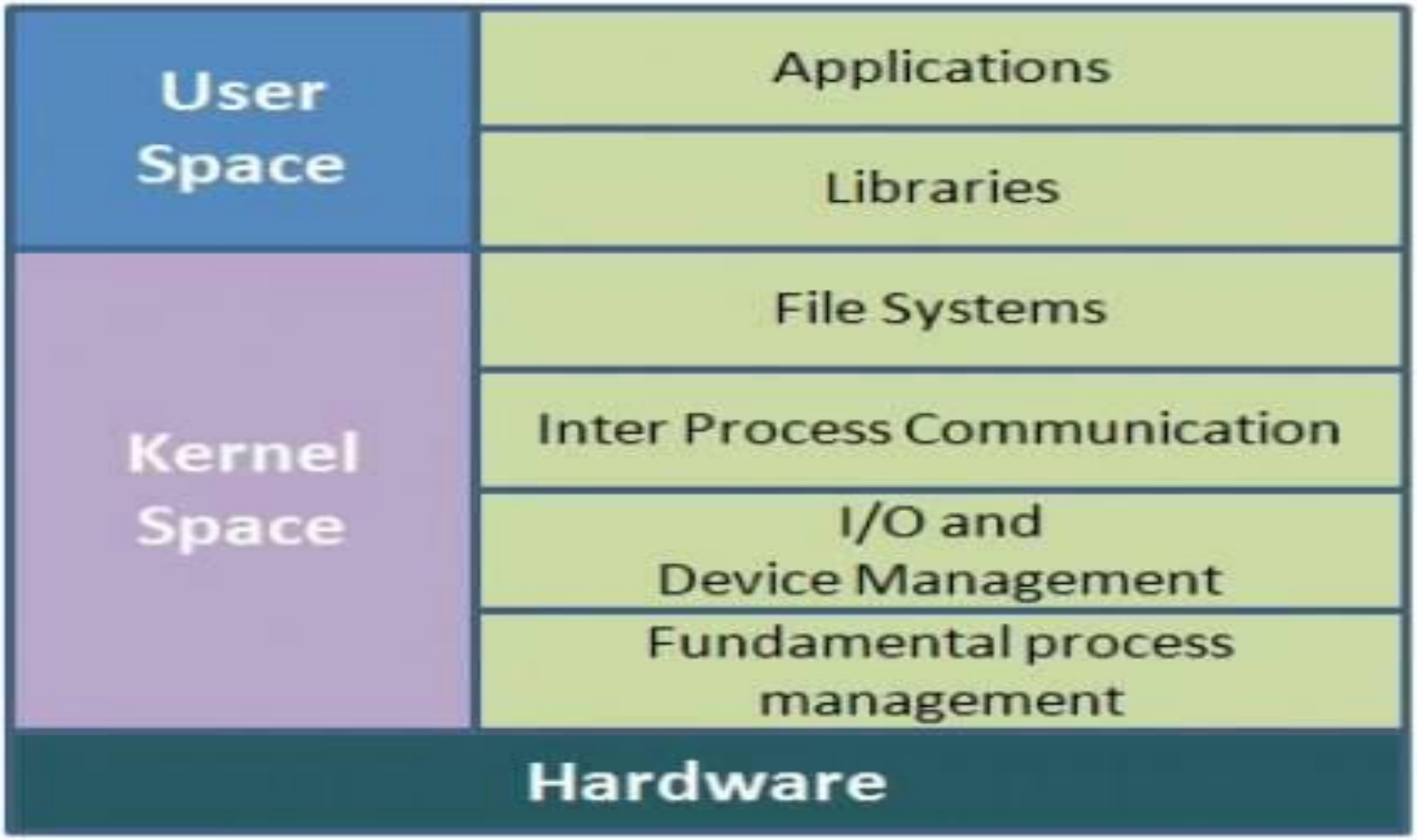
- Each user has at least one program executing in memory ⇨**process**
- If several jobs ready to run at the same time ⇨ **CPU scheduling**
- If processes don't fit in memory, **swapping** moves them in and out to run
- **Virtual memory** allows execution of processes not completely in memory.

Monolithic Kernel

- Kernel is a computer program that control over everything occurs in the system.
- **Kernel** is the core part of an OS which manages system resources. It also acts like a bridge between application and hardware of the computer.

- A **Monolithic kernel** is an OS architecture where the entire operating system (which includes the device drivers, file system, and the application IPC) is working in **kernel** space.
- All user process run in normal mode.
- All core OS function run in kernel mode.
- **Monolithic kernels** are able to dynamically load (and unload) executable modules at runtime.
- Data sharing between OS module is globally shared variable.

Monolithic Kernel



Continue..

Advantages of Monolithic Kernel –

- It provides CPU scheduling, memory management, file management and other operating system functions through system calls.
- It is a single large process running entirely in a single address space.

Disadvantages of Monolithic Kernel

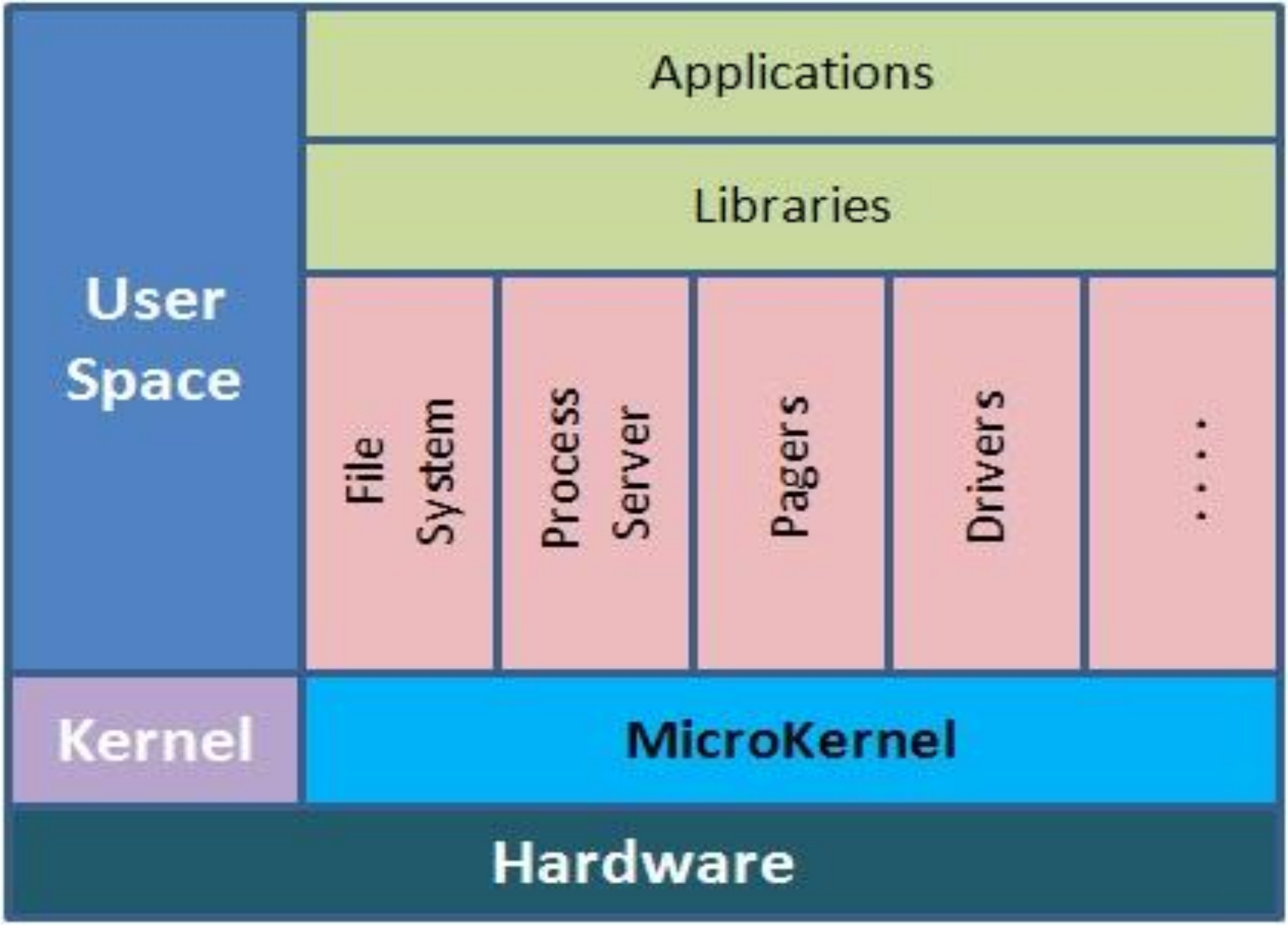
- if anyone service fails it leads to entire system failure.
- If user has to add any new service. User needs to modify entire operating system.
- Example of some Monolithic Kernel based OSs are: Unix, Linux

Microkernel

- The idea behind microkernel OS is reduce the kernel space to only basic process communication and I/O control and let the other services run in user space.
- In microkernel, the **user services** and **kernel services** are implemented in different address space.
- Communication in microkernel is done by massage passing.

Advantages:

- if any user service fails it does not affect kernel service.
- It is easily **extendable** i.e. if any new services are to be added they are added to user address space and hence requires no modification in kernel space.


Applications
Libraries
User Space
File System
Process Server
Pagers
Drivers
. . . .
Kernel
MicroKernel
Hardware

Microkernel

Disadvantages:

- Potential performance loss (more s/w interface due to message passing)
- Process management is complex.

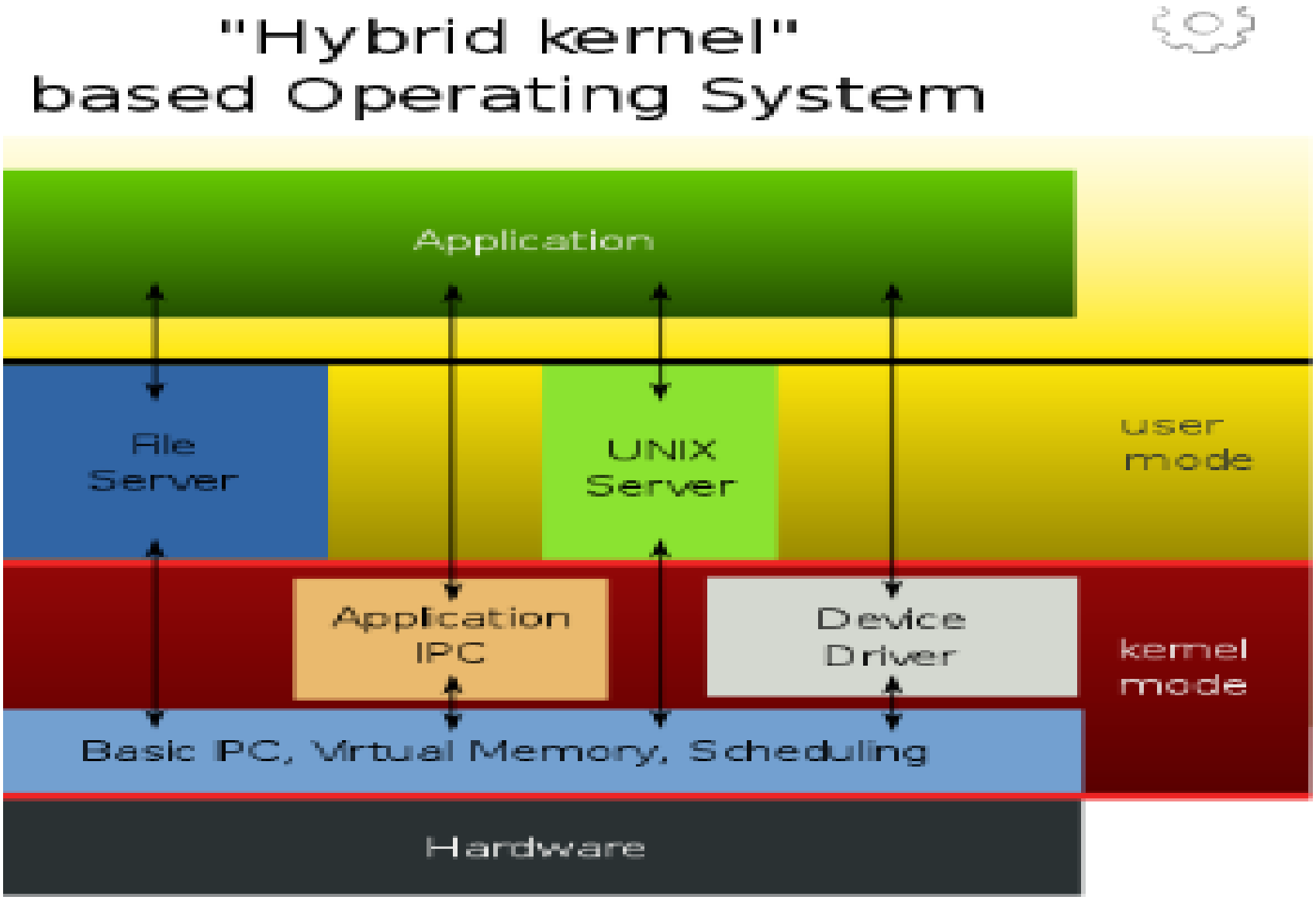
Example: Symbian, Mac OS, QNX etc

- **Monolithic** kernels are usually faster than **microkernels**.
- The first **microkernel** Mach was 50% slower than most **monolithic** kernels, while later ones like L4 were only 2% or 4% slower than the **monolithic** designs

Hybrid kernel

- Hybrid kernel approach is derived from the best of both micro and monolithic kernel.
- This kernel approach combines the speed and simpler design of monolithic kernel with the modularity and execution safety of microkernel.
- Instead of load whole things into memory, core module are loaded dynamically into memory on demand.
- **Example** of a **hybrid kernel** are Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008 and Windows 7.

Hybrid kernel



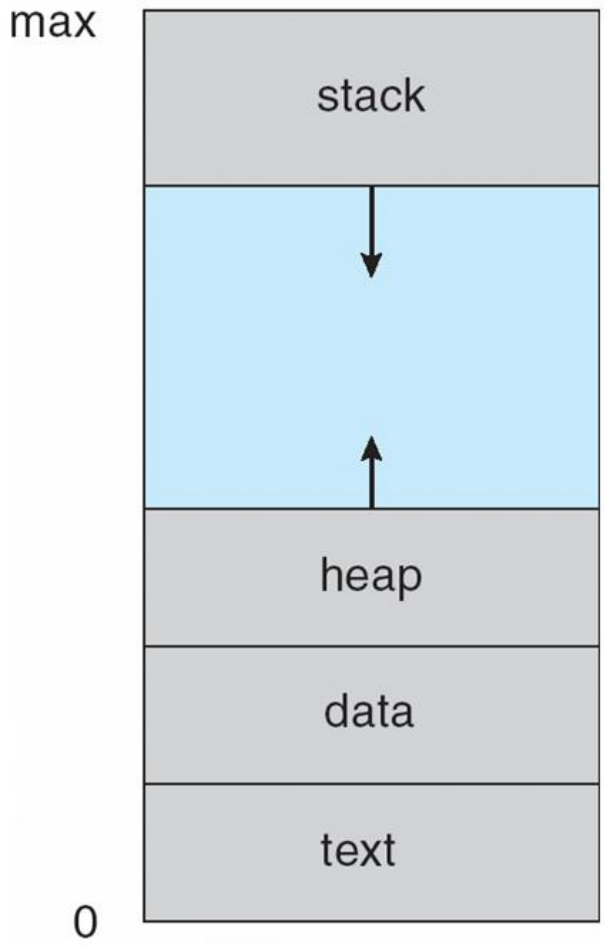
Process Management

Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution mode;

- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process in Memory



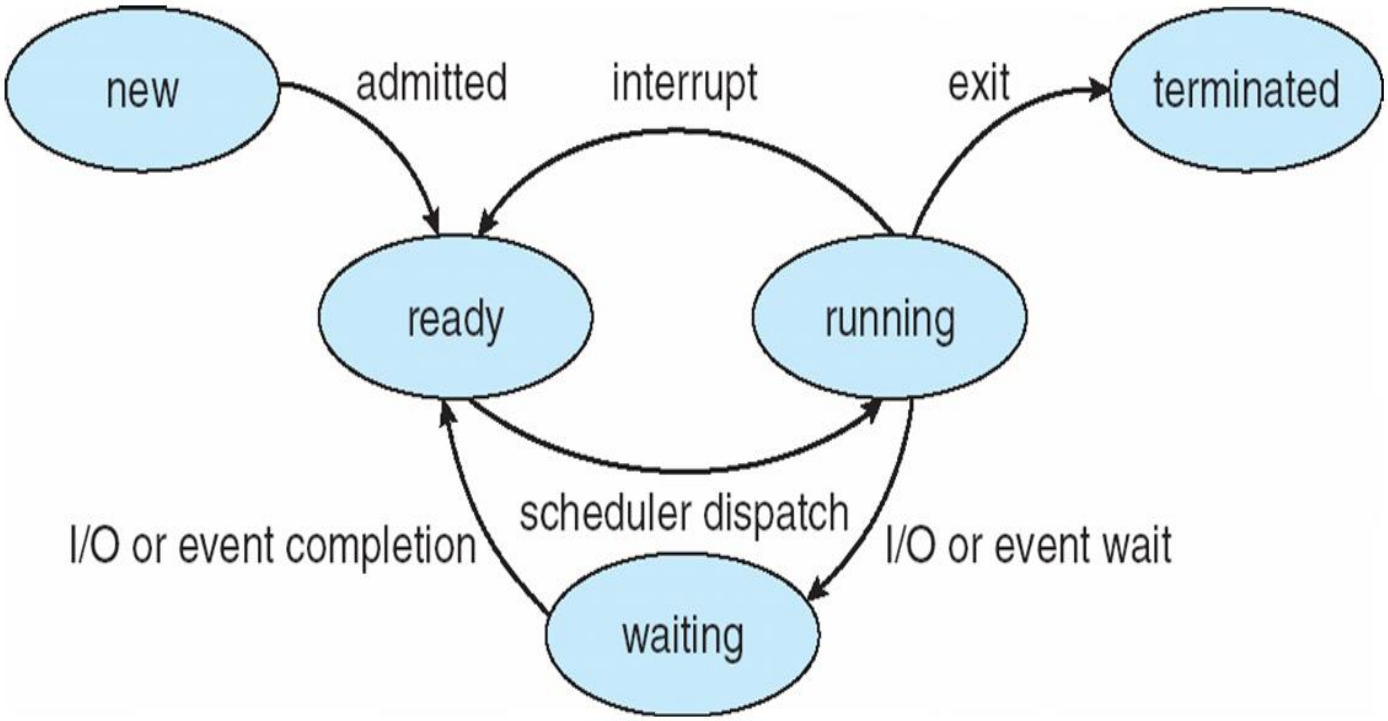
Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc

Process State

- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Diagram of Process State



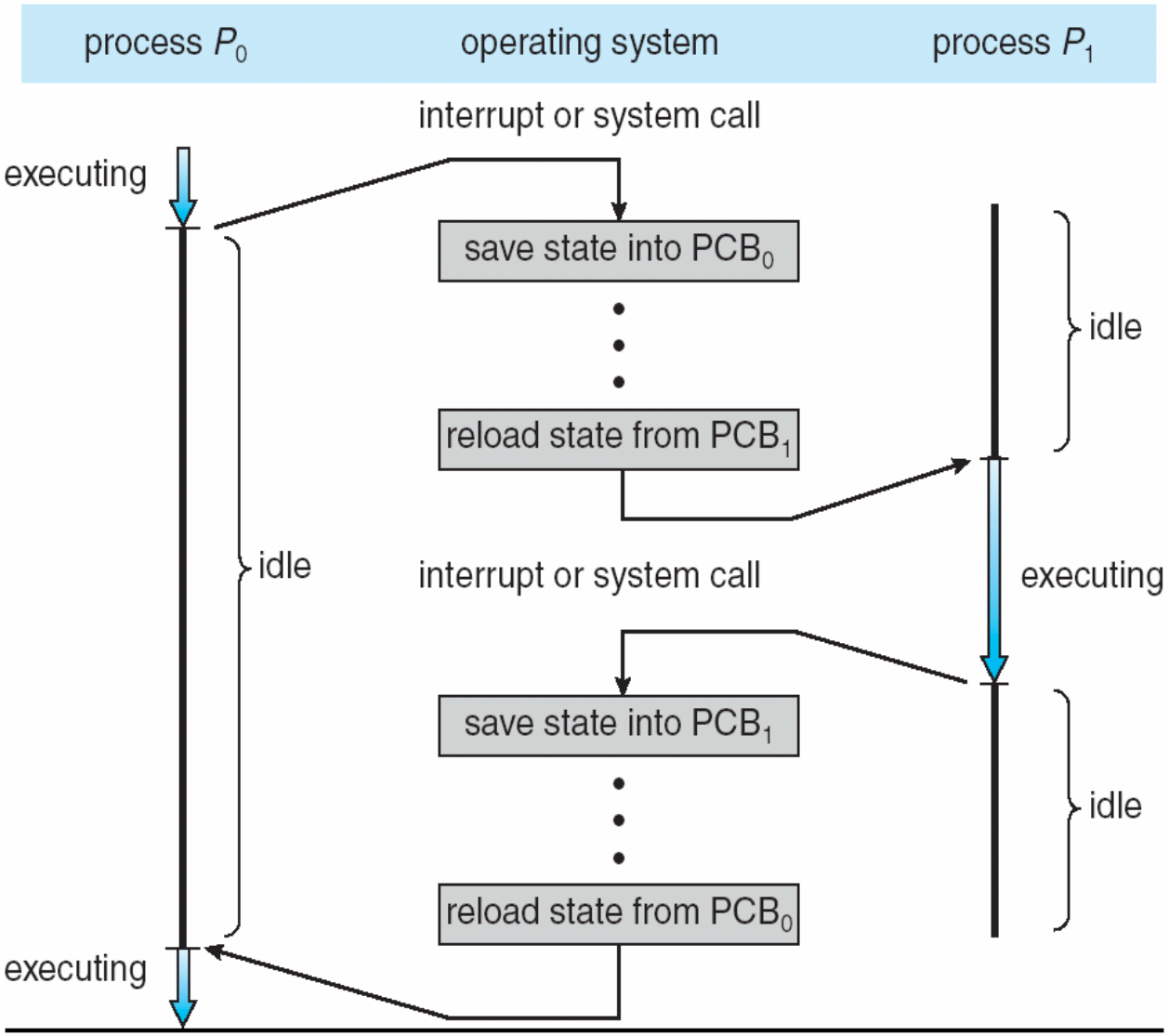
Process Control Block (PCB)

Information associated with each process (also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

Basic Questions

- What is the difference between ready queue and job queue ?

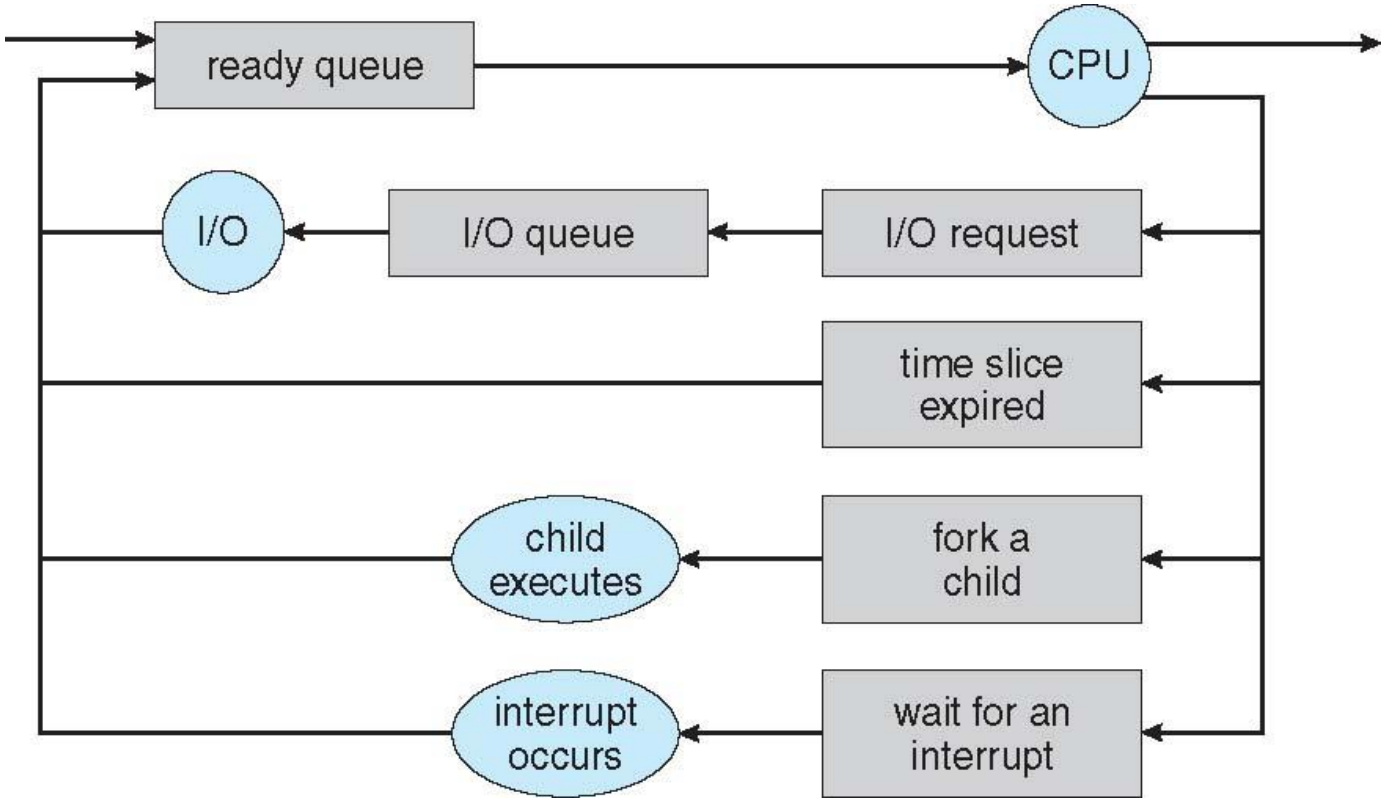

process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Representation of Process Scheduling

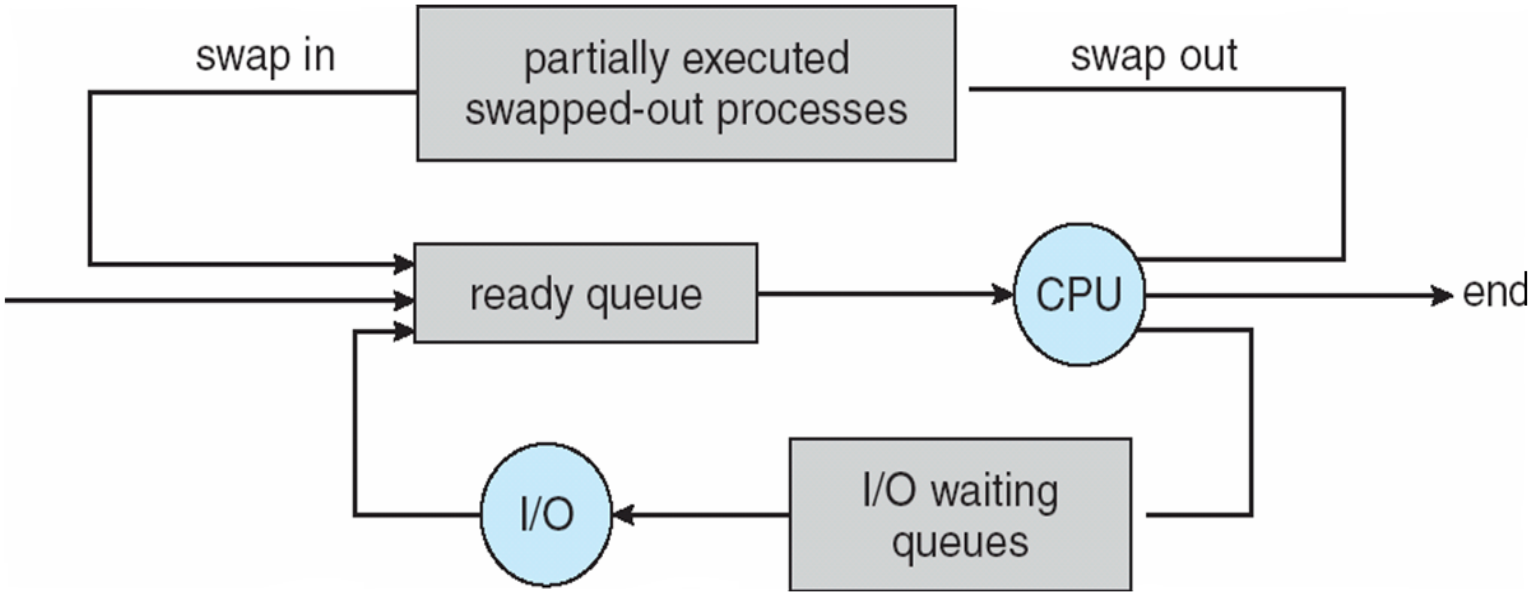
- **Queueing diagram** represents queues, resources, flows



Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



Context Switch

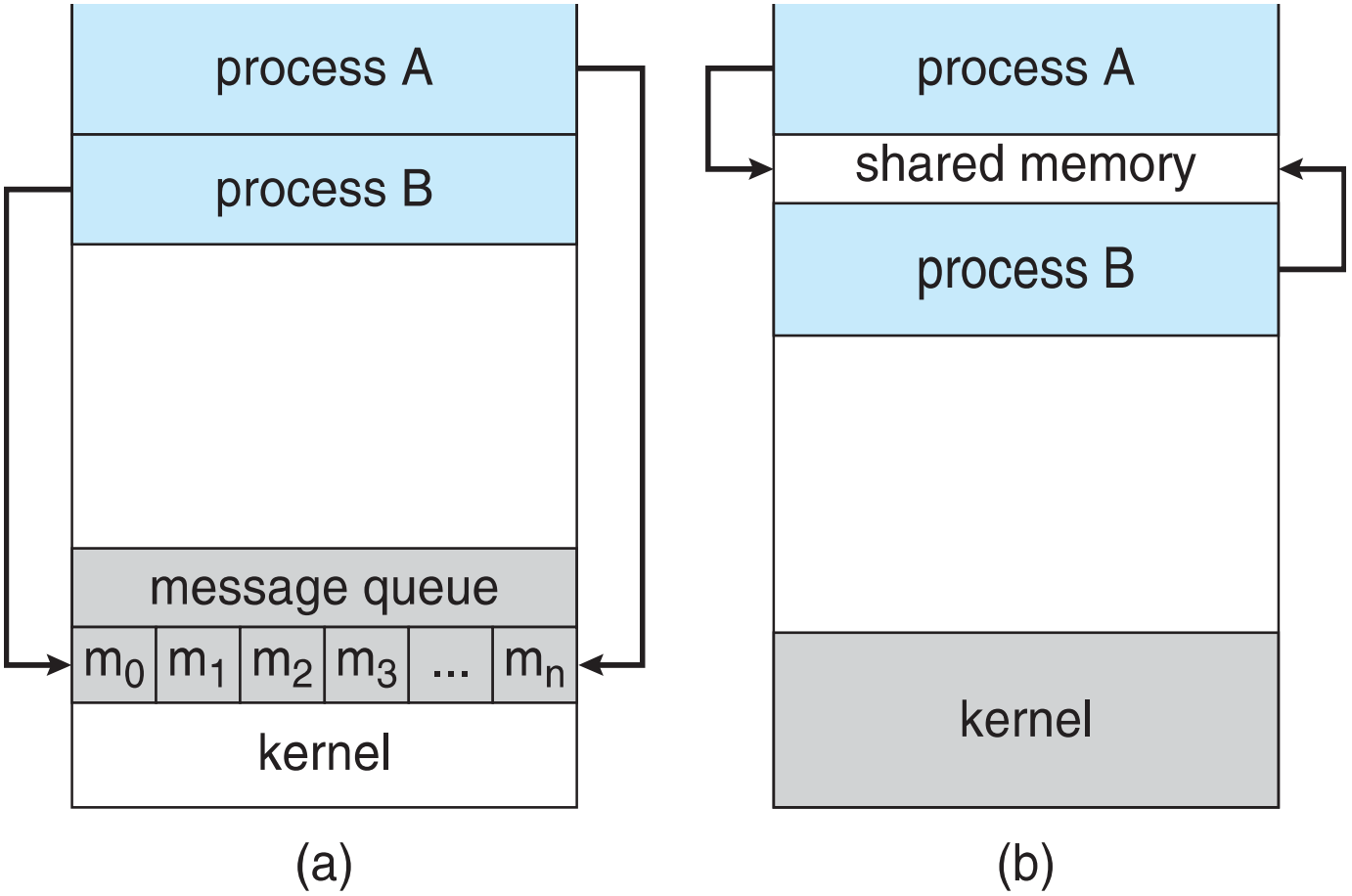
- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Context switch times are highly dependant on hardware support.

Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Communications Models

(a) Message passing. (b) shared memory.



CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3 The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

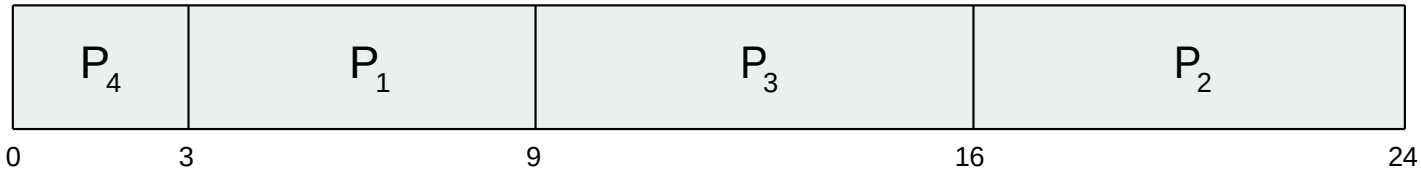
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

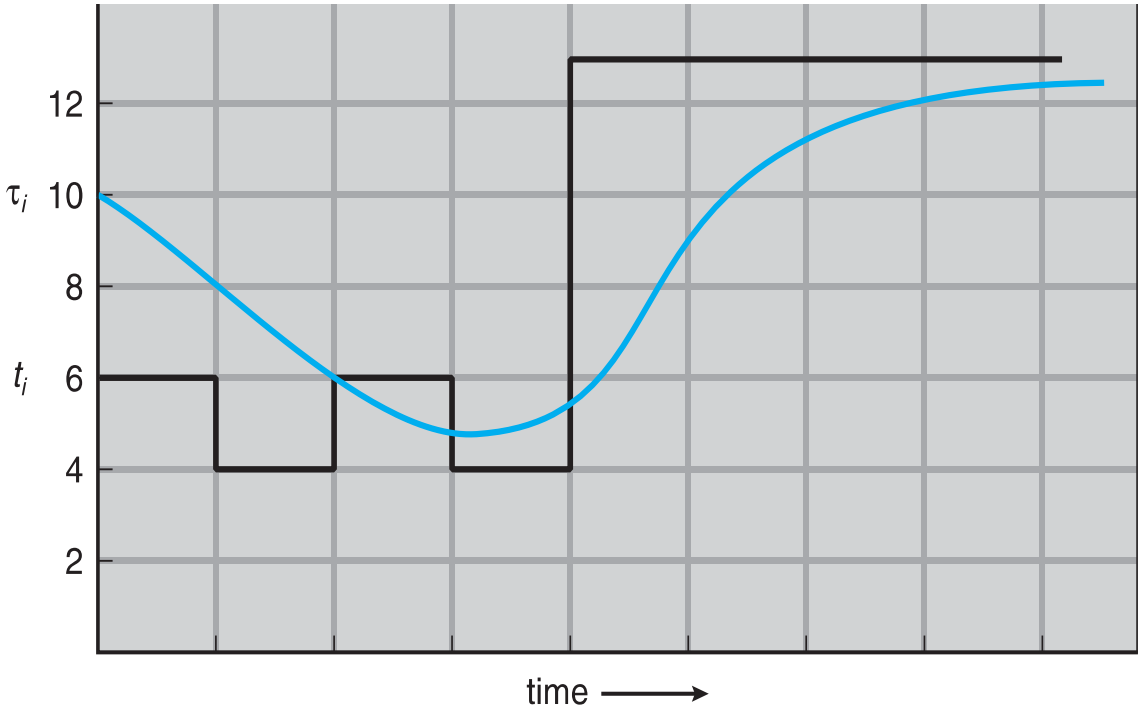
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

Prediction of the Length of the Next CPU Burst



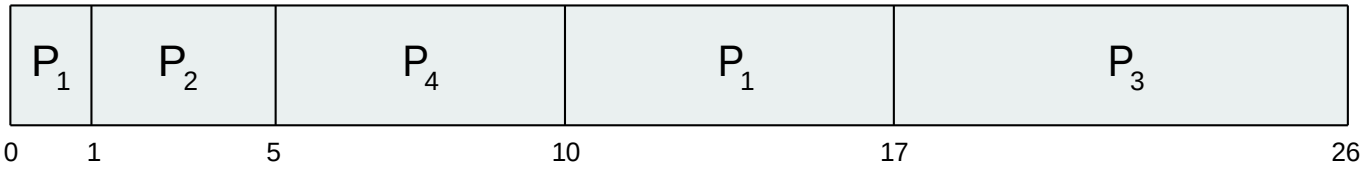
CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

ProcessAarri *Arrival* TimeT		Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

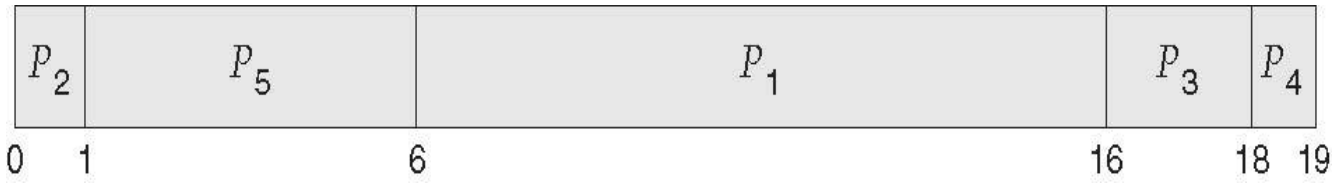
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF algorithm is a special case of the general priority scheduling algorithm.
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

ProcessAarri	Burst TimeT	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

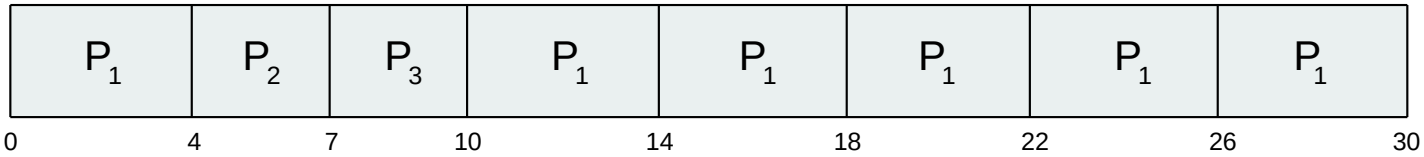
Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n\text{-}1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Continue..

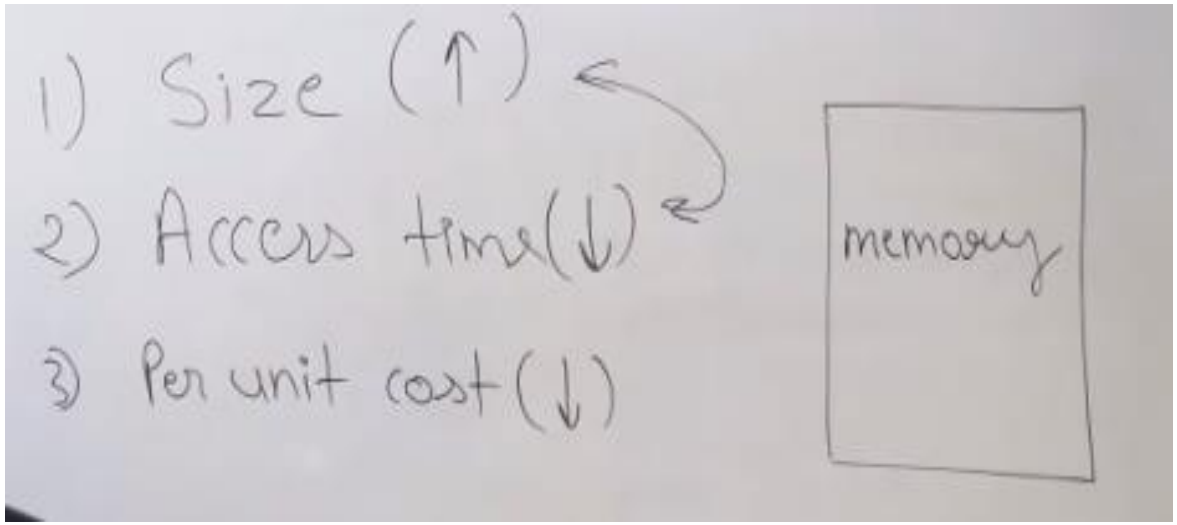
Process Id	Arrival time	Burst Time	Avg. Waiting Time	T.A.T
P0	0	5		
P1	1	3		
P2	2	1		
P3	3	2		
P4	4	3		

Avg. waiting time=5.8

Avg. turnaround time=8.6

Memory Management

Introduction about memory

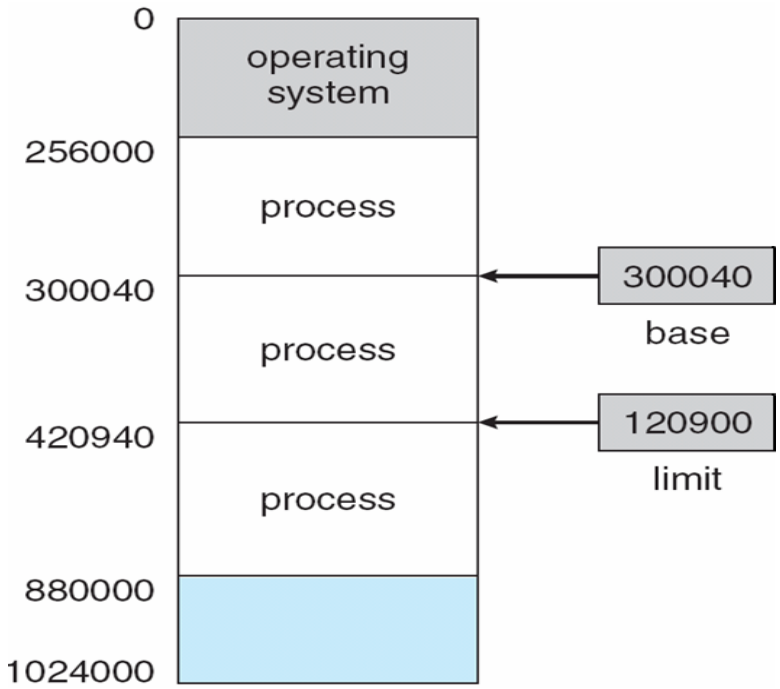


Introduction

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

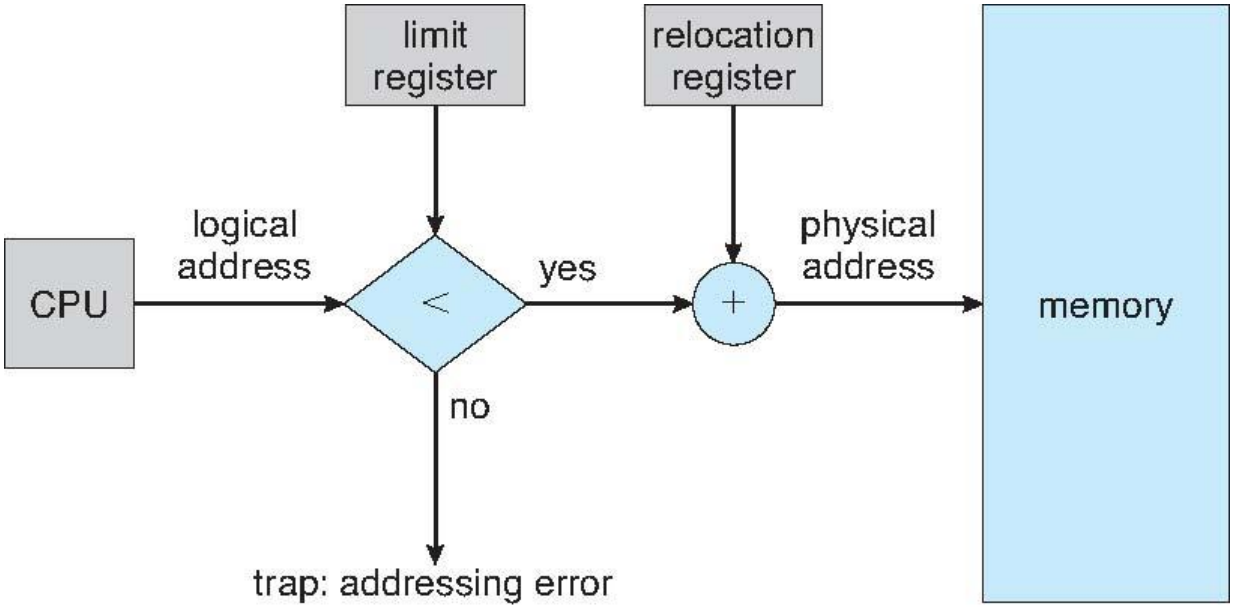
Questions ?

1. Why we need logical address ?
2. Can the logical address space be larger than the physical address space? Can it be smaller
3. What is the difference between process switch and context switch
4. What resources are used when a thread is created ? How do they are differ from those when a process is created

Memory-Management Unit (MMU)

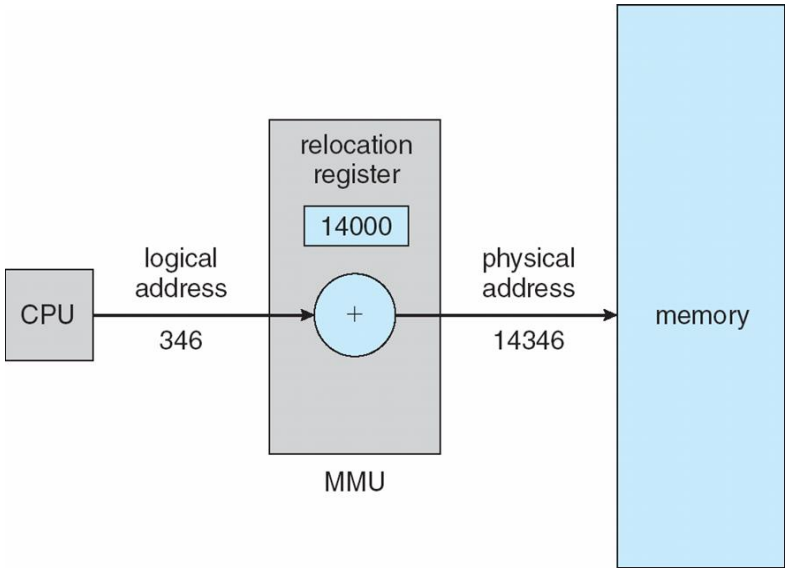
- Hardware device that at run time maps virtual to physical address
- Many methods possible
- Consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Hardware Support for Relocation and Limit Registers



Dynamic relocation

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- No special support from the operating system is required
  - Implemented through program design



Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory

- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed

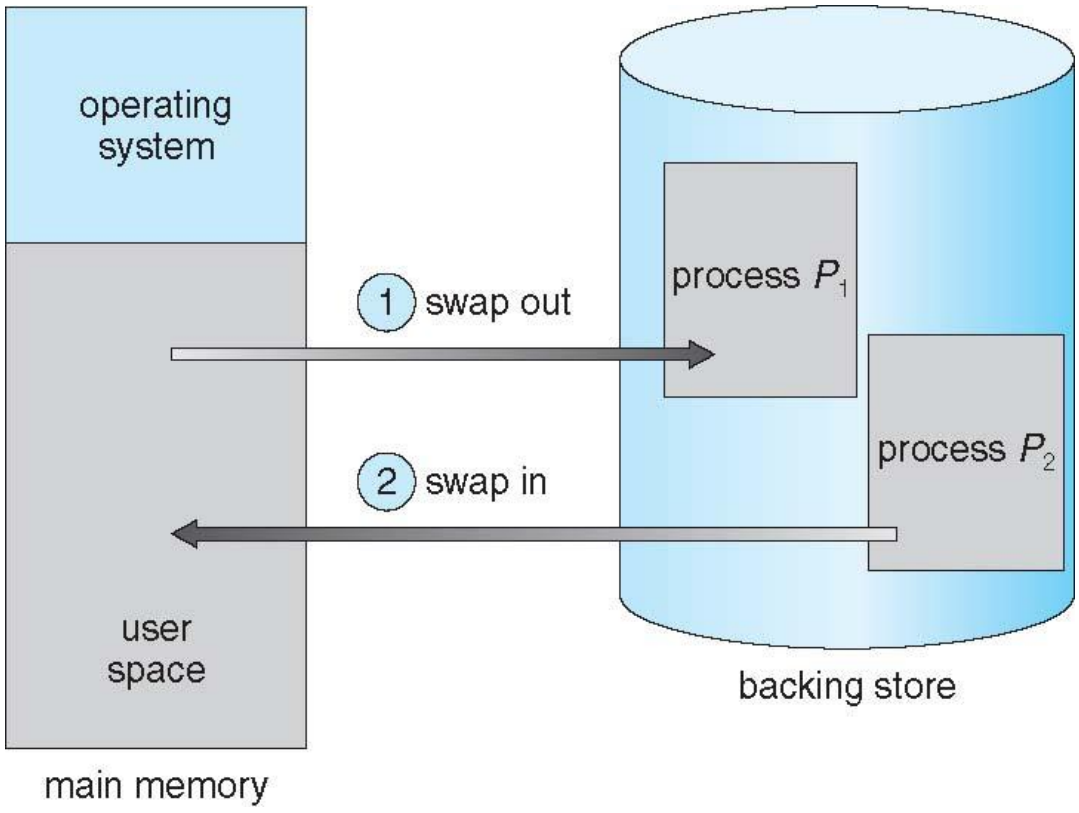
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped

- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Schematic View of Swapping



Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)

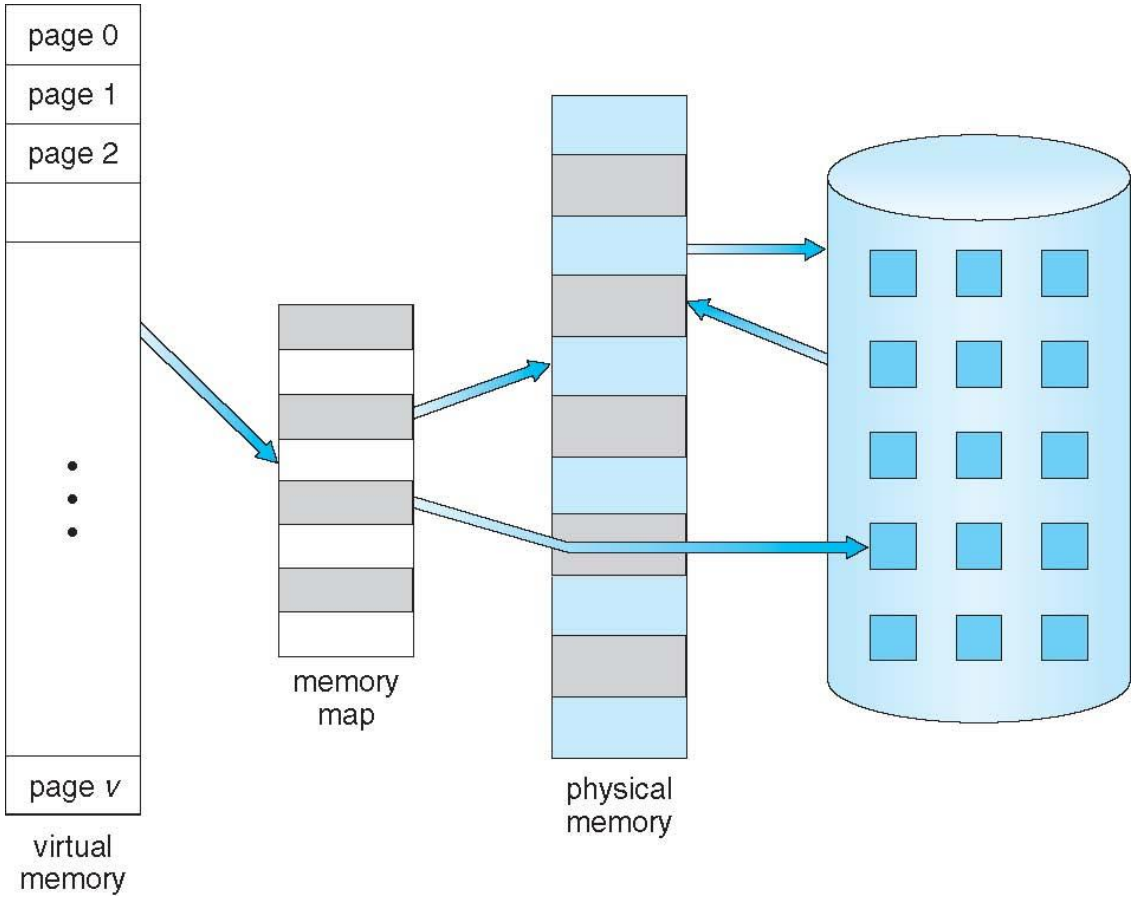
Virtual Memory

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - More programs running concurrently
  - Virtual memory is implemented by various operating systems such as Windows, Mac OS X, and Linux.

Background (Cont.)

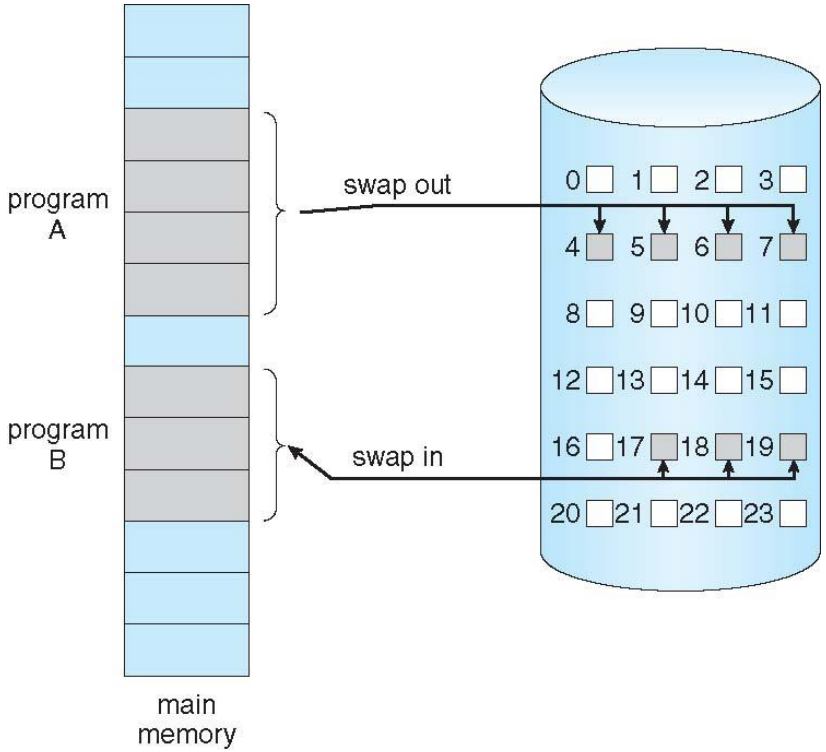
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed.



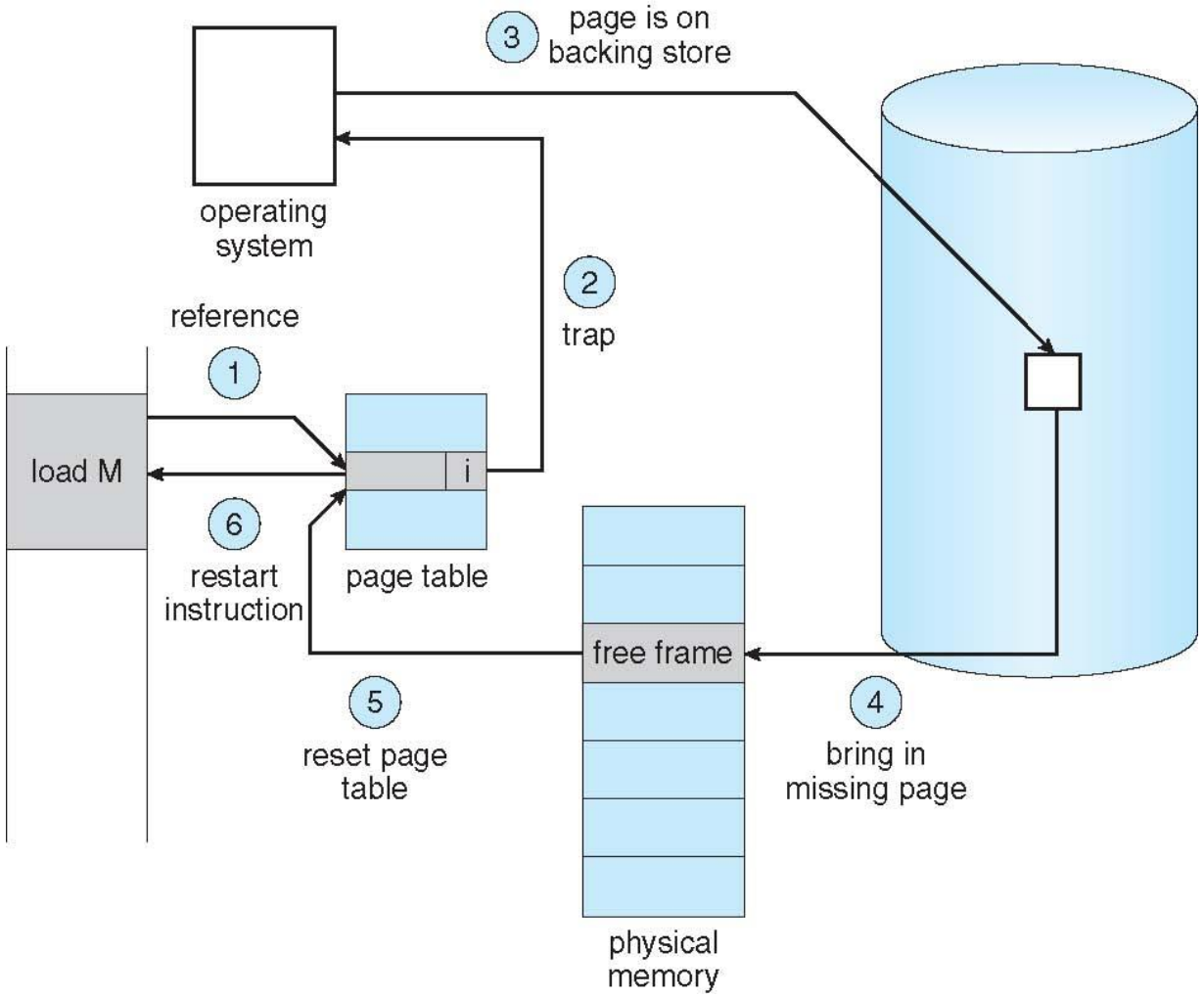
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault

Steps in Handling a Page Fault

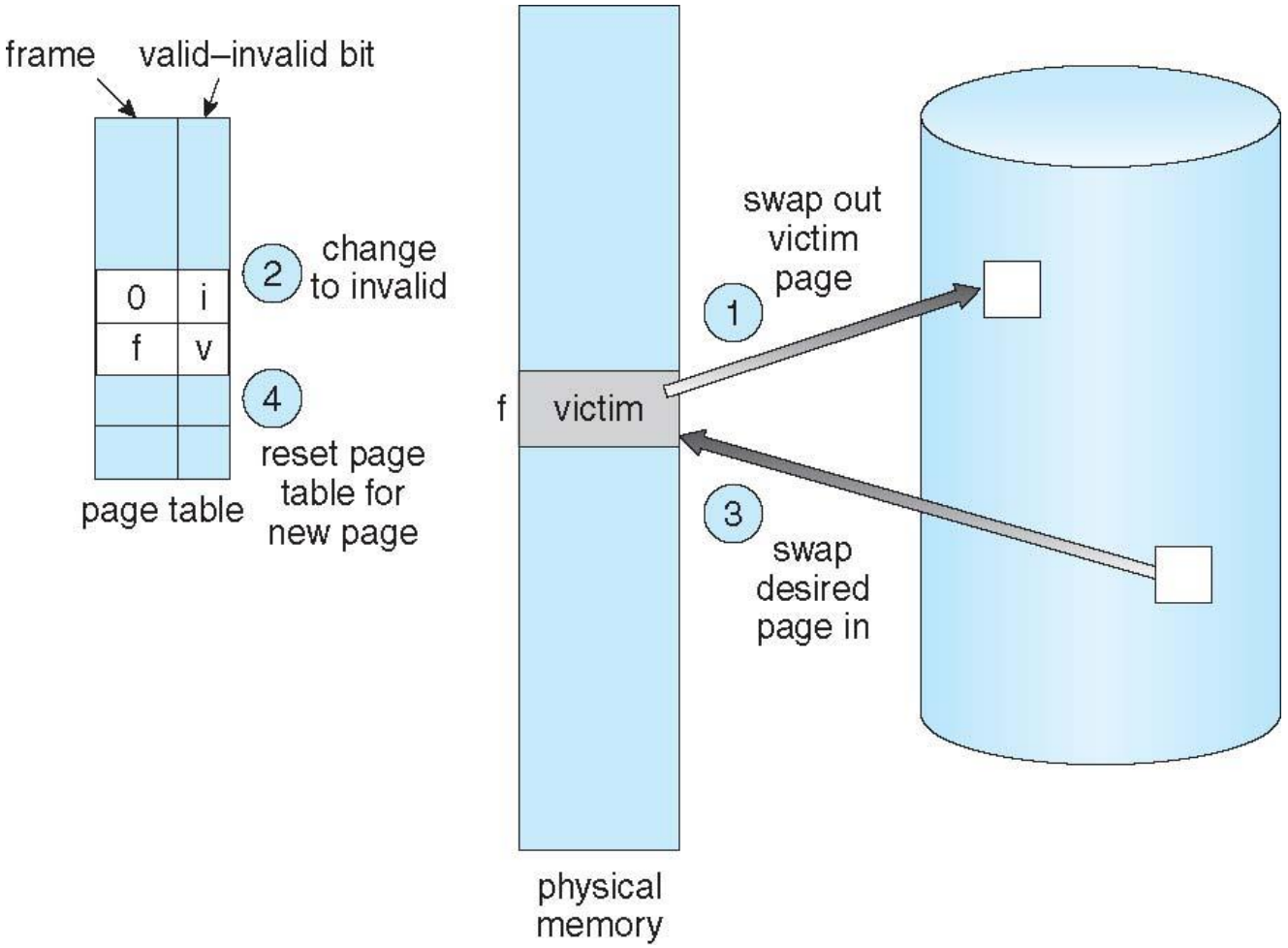


Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



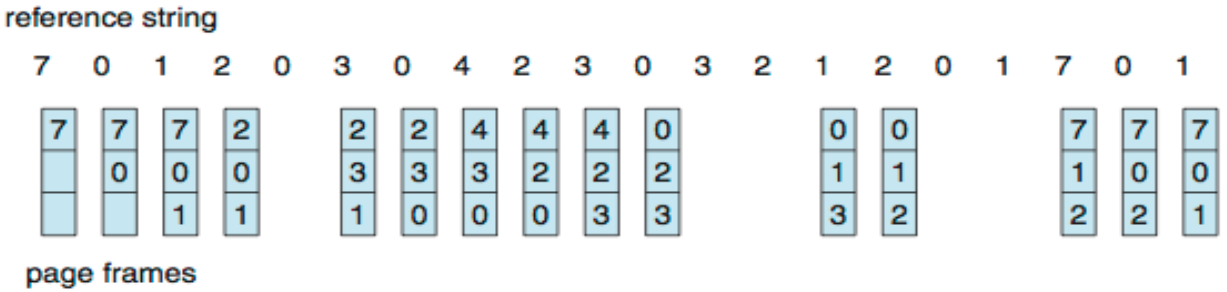
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

First-In-First-Out (FIFO) Algorithm

- Reference string:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

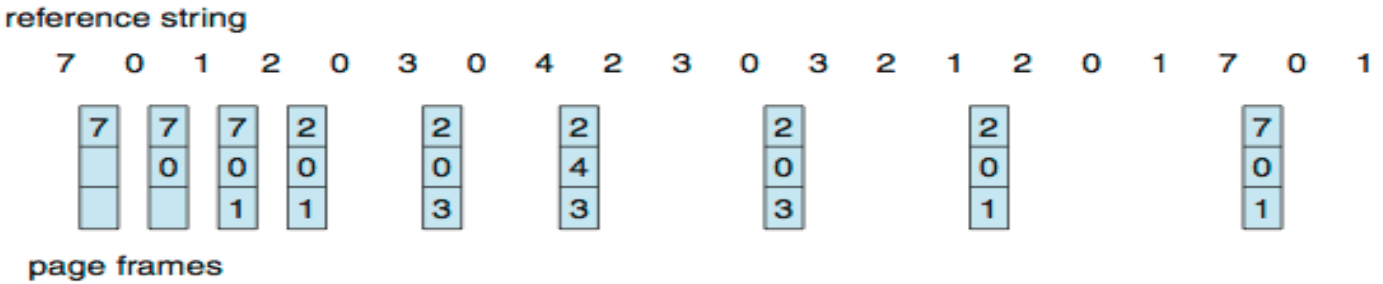


15 page faults

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**
- How to track ages of pages?
  - Just use a FIFO queue

Optimal Algorithm

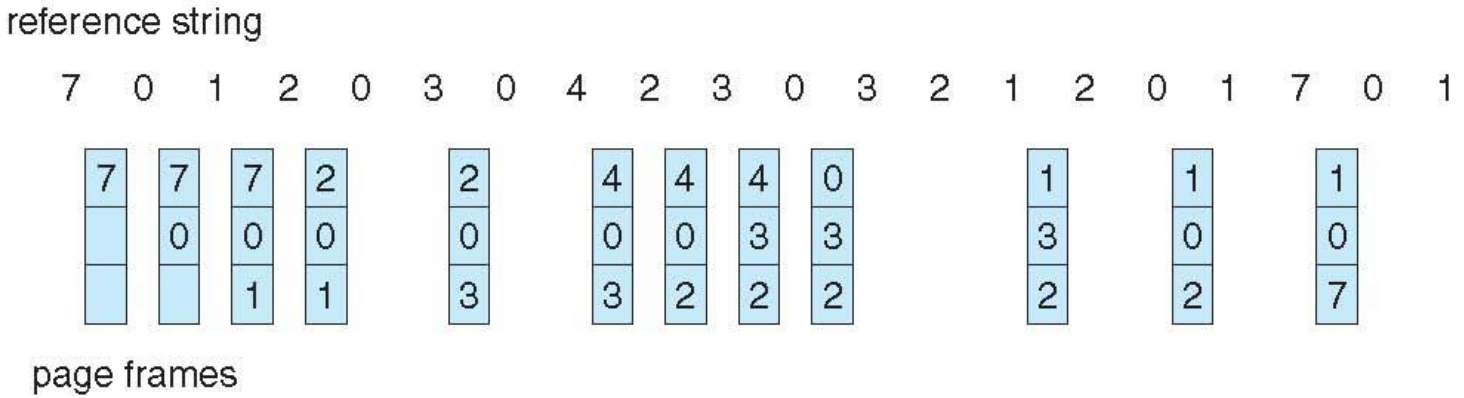
- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future



- Used for measuring how well your algorithm performs

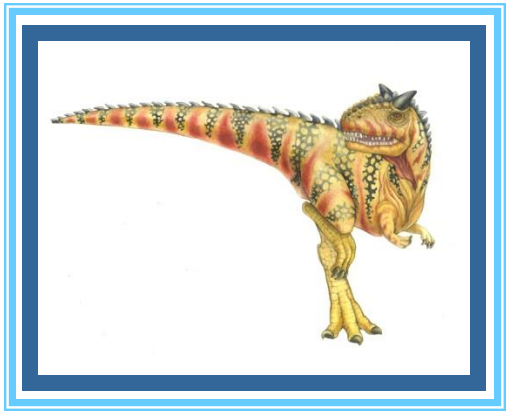
Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used

I/O Systems



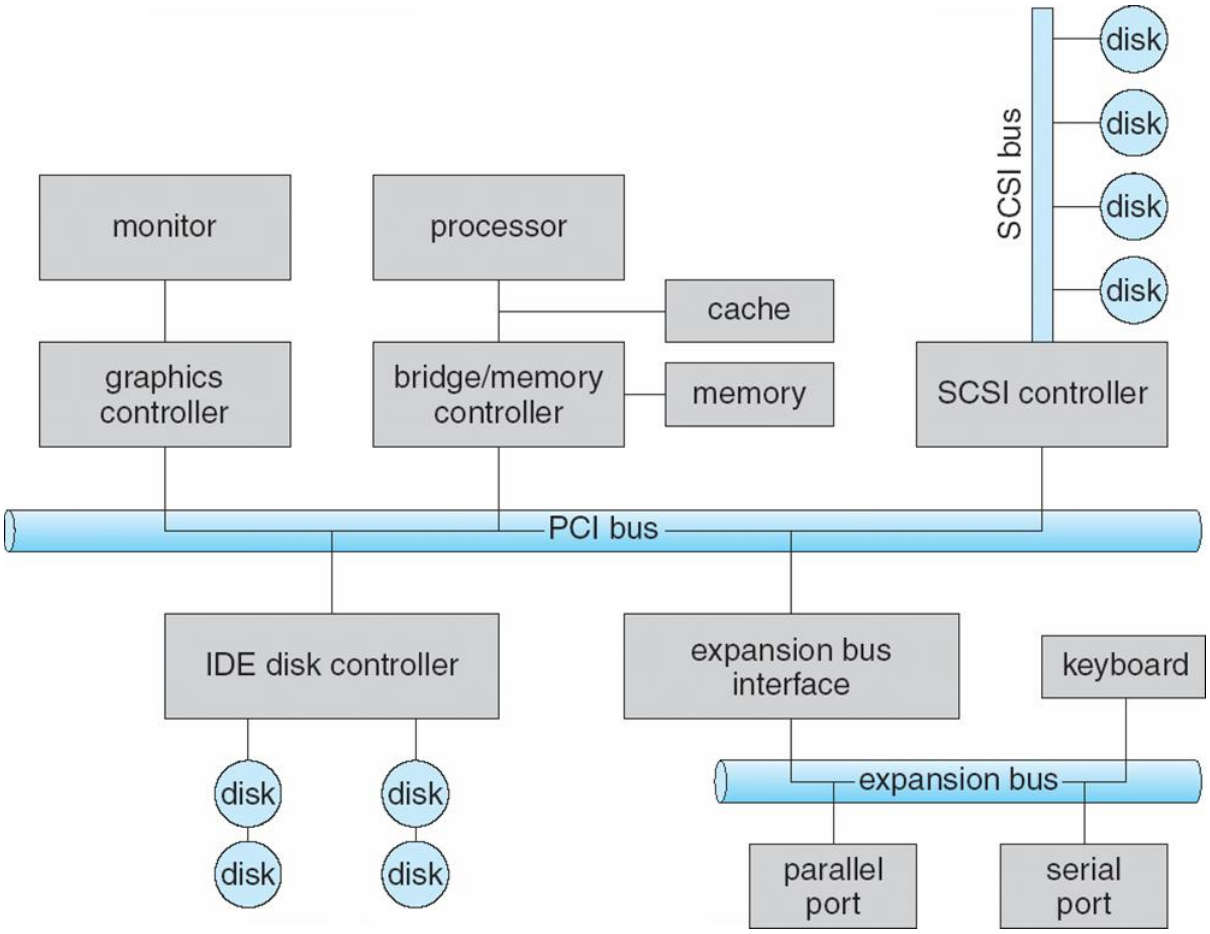
Overview

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - Various methods to control them
  - Performance management
- Ports, busses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

I/O Hardware

- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human-interface
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus – single bus and dual bus**
  - **Controller** (**host adapter**) – electronics that operate port, bus, device

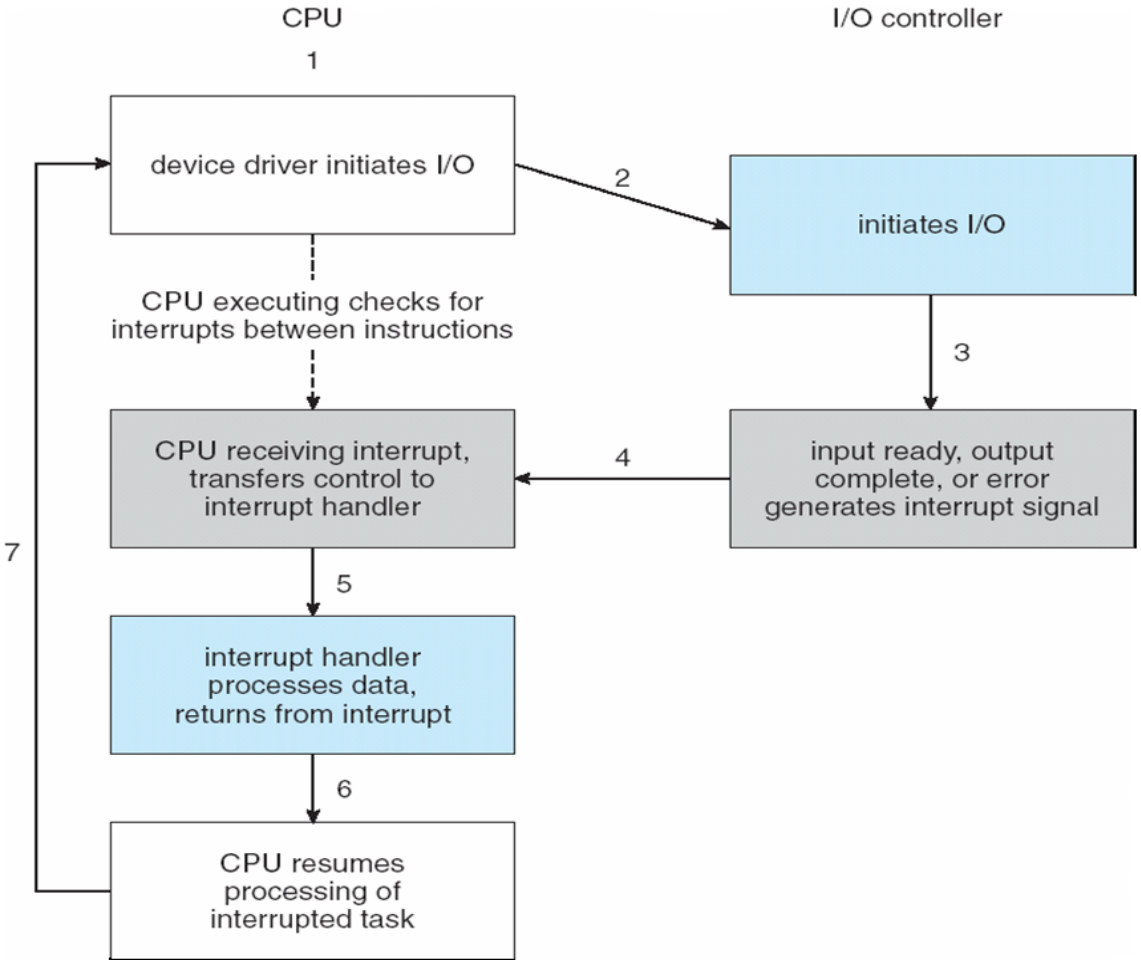
A Typical PC Bus Structure



Interrupts

- It is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- An interrupt is a special signal that cause the computer cpu to suspend what it is doing and transfer the control to special program called an interrupt handler.
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
  - **non-maskable interrupt** cannot be disabled or ignored by the CPU
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **non-maskable**

Interrupt-Driven I/O Cycle



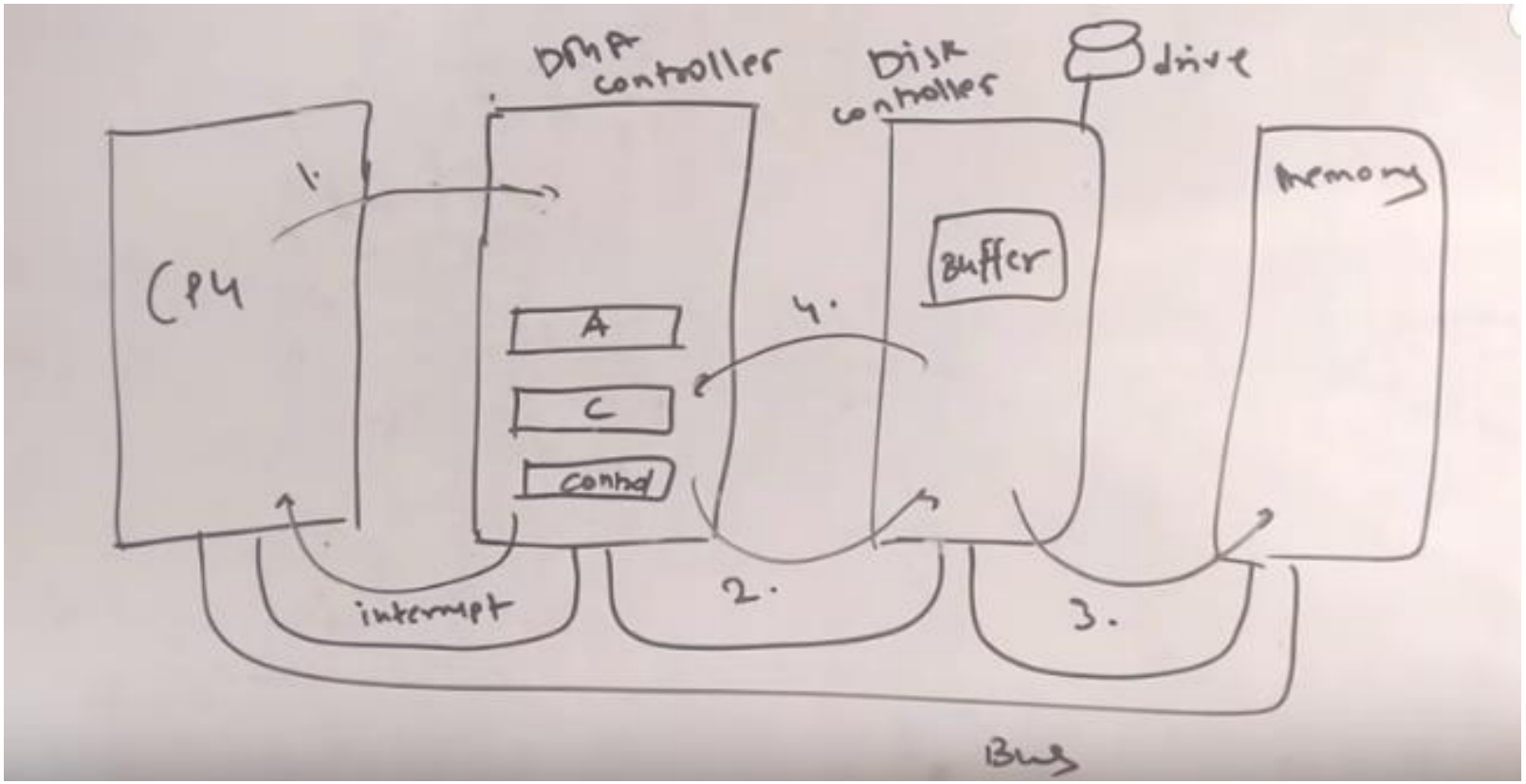
Interrupts (Cont.)

- Interrupt mechanism also used for **exceptions**
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

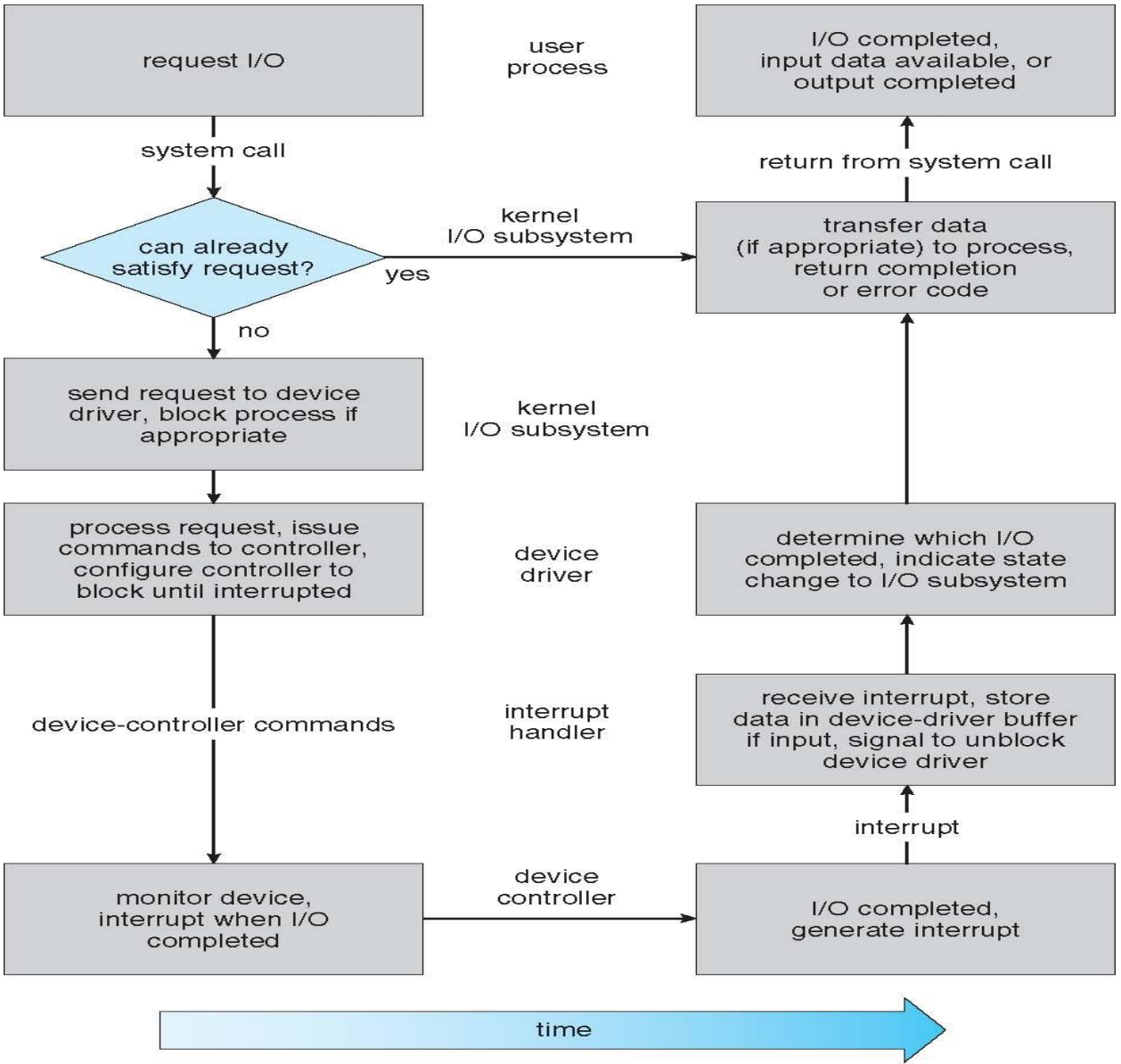
Direct Memory Access

- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
  - When done, interrupts to signal completion
- Version that is aware of virtual addresses can be even more efficient - **DVMA**

DMA


DMA controller
Disk controller
drive
Memory
CPU
1.
A
C
cmnd
4.
buffer
interrupt
2.
3.
Bus

Life Cycle of An I/O Request



DISTRIBUTED OPERATING SYSTEM

DISTRIBUTED SYSTEM

- A distributed system is a collection of **autonomous** computers that appear to the users of the system as a **single computer.**

Why Distributed Systems?

1. Resource sharing
2. Scalability
3. Reliability
4. Need for higher processing speed

DISTRIBUTED COMPUTING

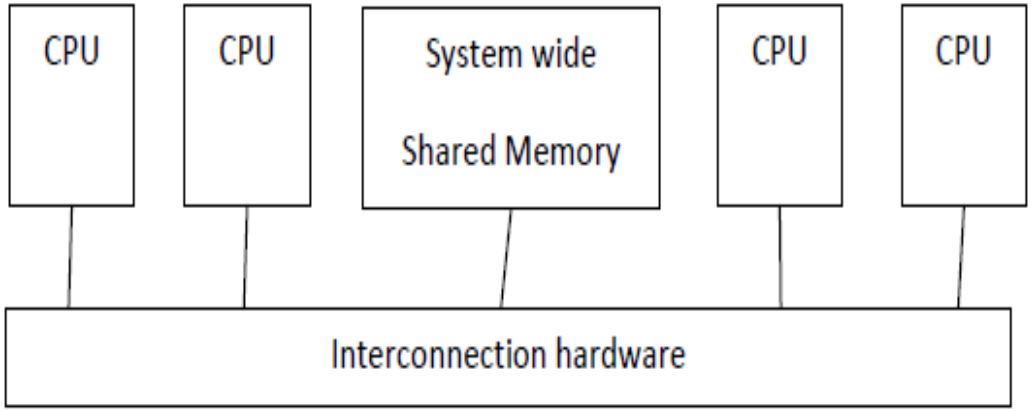
- Microelectronic technology have resulted in the availability of fast and inexpensive processors.

- Communication technology have resulted in cost-effective and highly efficient computer networks.

- Computer architectures consisting of interconnected, multiple processors are of two types:
  - Tightly coupled systems.
  - Loosely coupled systems.

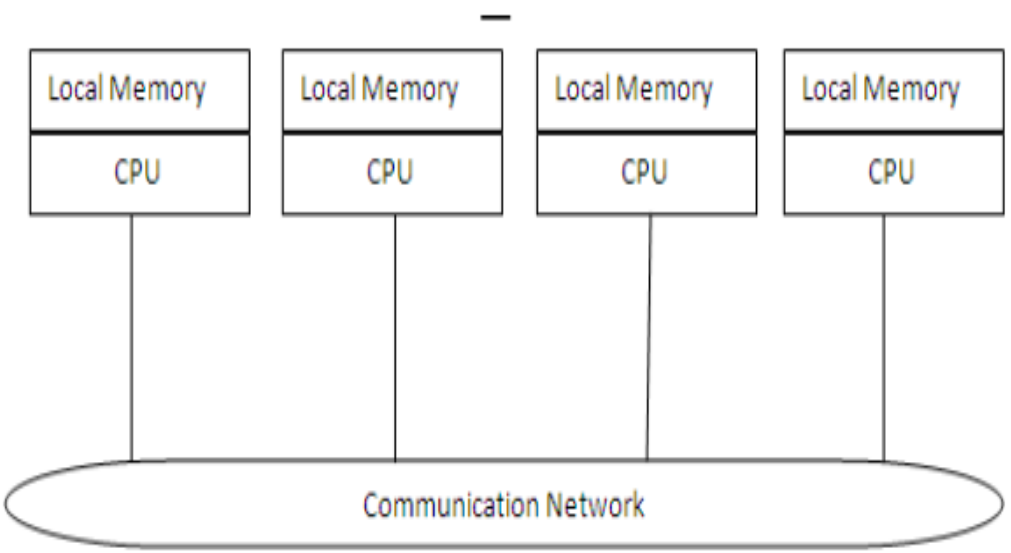
Tightly Coupled Systems (parallel processing systems)

- Single system wide primary memory is shared by all the processors.
- Communication between the processors takes place through the shared memory.



Loosely Coupled Systems (distributed computing systems)

- The processors do not share memory
- Each processor has its own local memory.
- Communication between the processors is done by message passing.



- Processors of loosely coupled systems can be located far from each other to cover a wider geographical area.

- In tightly coupled systems,

1. number of processors usefully deployed is small and
2. limited by the bandwidth of the shared memory.

- **This is not the case with distributed computing systems.**

1. Distributed computing systems are more freely expandable
2. Can have almost unlimited number of processors.
3. Its own resources are local, whereas the other processors and their resources are remote.

- Together, a processor and its resources are called **node/site/machine** of the distributed computing system.

TIME-SHARING SYSTEMS

- In 1970s that computers started to use the concept of **time sharing.**

- **The advent of time-sharing systems was the first step toward distributed computing systems.**

- It provided with two important concepts used in distributed computing systems
  - the sharing of computer resources simultaneously by many users,
  - accessing of computers from a place different from the main computer room.

Centralized Time-Sharing Systems

- Processing of a user's job was done at the user's own computer.
- Main computer was simultaneously shared by a larger number of users.
- Shared resources such as files, databases, and software libraries were placed on the main computer.
- **Limitation:** terminals could not be placed very far from the main computer room as ordinary cables were used to connect the terminals to the main computer.

Evolution of Distributed Computing Systems

- Between 1960s and early 1970s emerged two key networking technologies the LAN (local area network) and WAN (wide area network).

- In 1990's there was advancement in networking technology, the ATM (Asynchronous Transfer Mode) technology.

- ATM's can make very high speed networking possible, in both LAN and WAN environments.

- The availability of high-bandwidth networks can allow distributed computing systems to support multimedia applications.

- **The merging of computer and networking technologies gave birth to distributed computing systems in the late 1970s.**

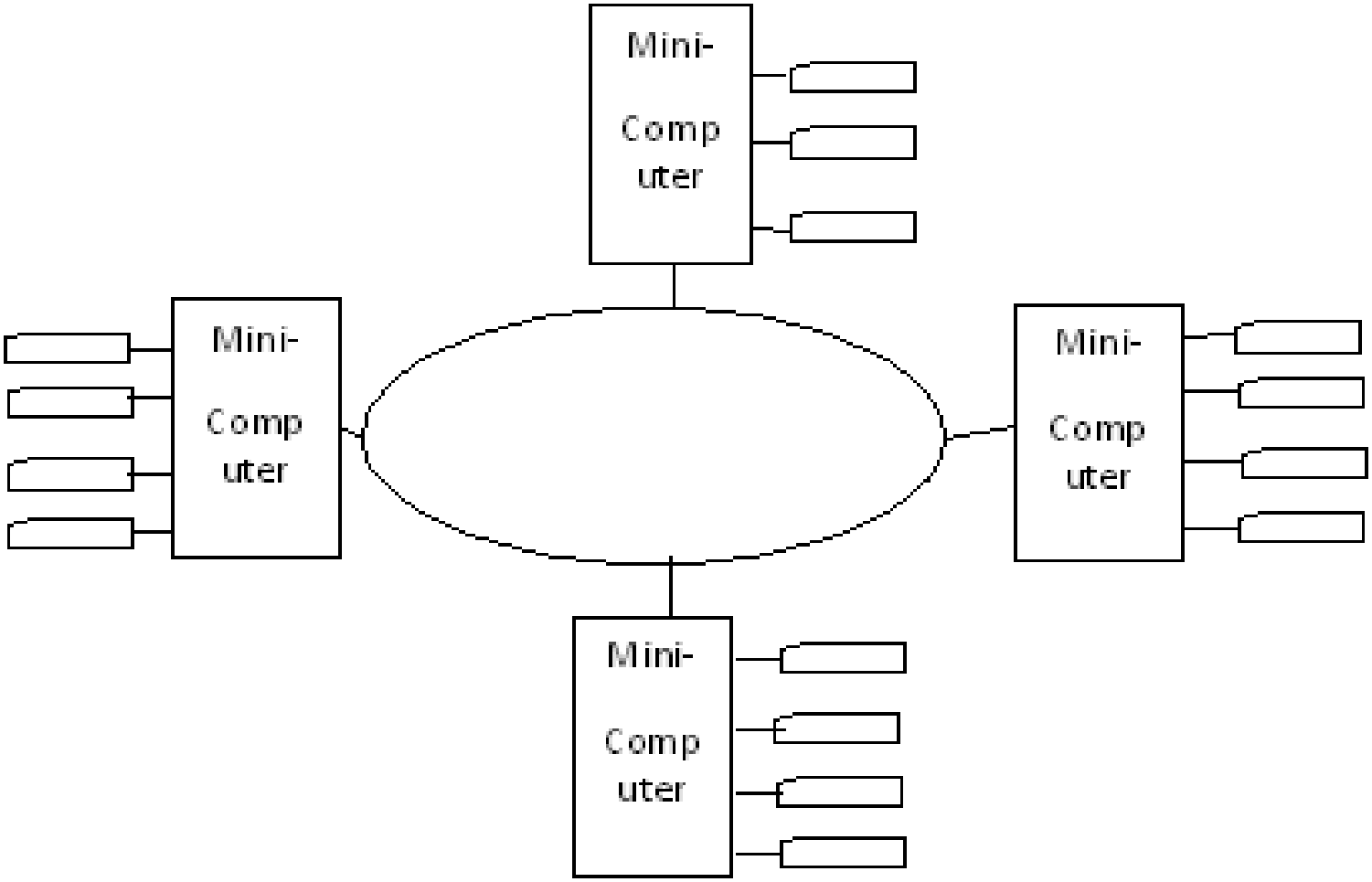
MODELS OF DISTRIBUTED COMPUTING SYSTEM

1. Minicomputer
2. Workstation
3. Workstation-server
4. Processor-pool
5. Hybrid

Minicomputer Model

- Is an extension of the centralized time-sharing system
- Consists of a few minicomputers interconnected by a communication network.
- Each minicomputer has multiple users simultaneously logged on to it.
- Each user is logged on to one specific minicomputer, with remote access to other minicomputers.
- The network allows a user to access remote resources that are available on some other machines.
- Used when resource sharing with remote users is desired.

Distributed computing system based on the minicomputer model



Workstation Model

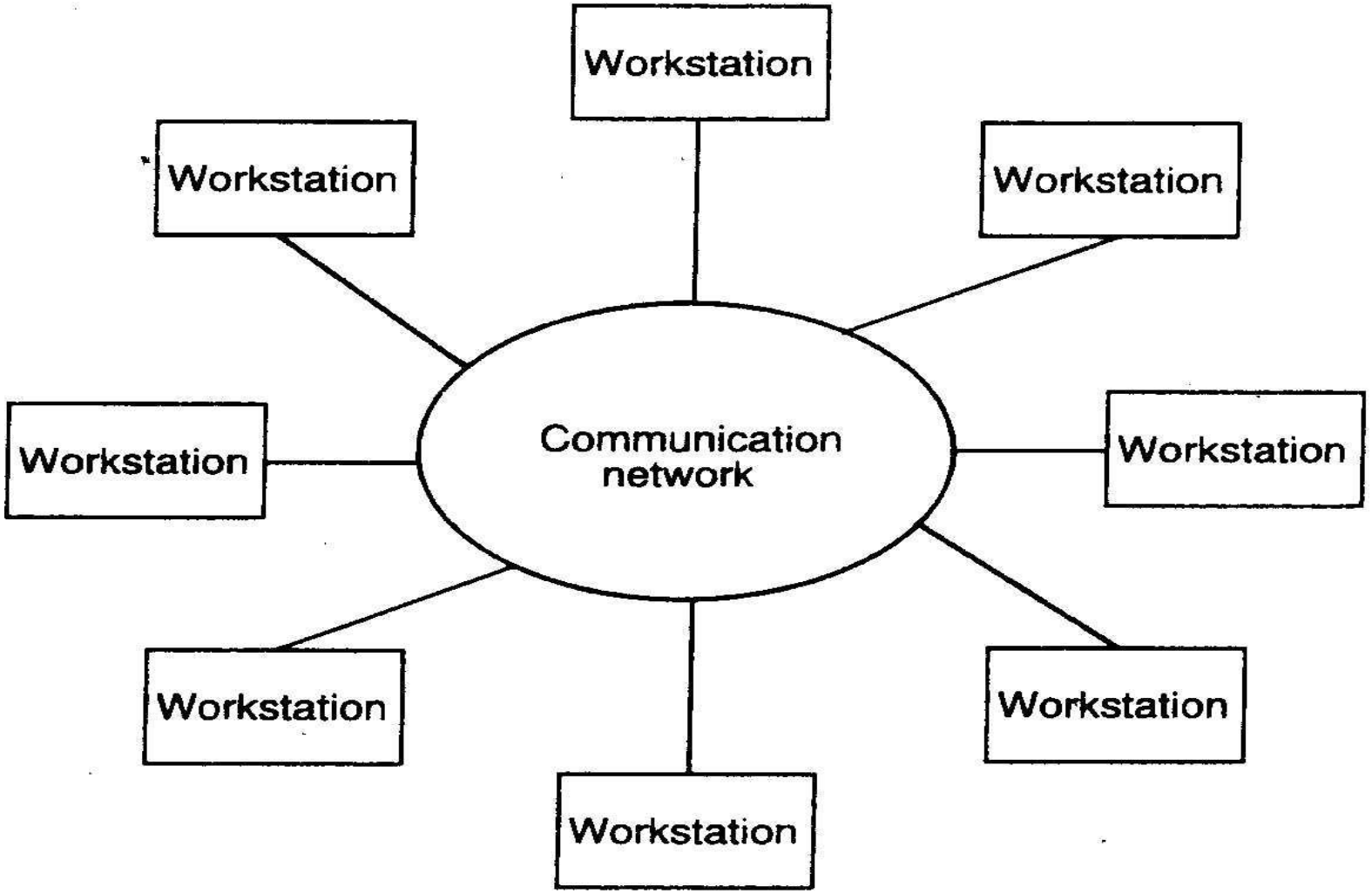
- It consists of several workstations interconnected by a communication network.
- Each workstation is equipped with its own disk and serve as a single-user computer.
- In this model:

1. a user logs onto one of the workstations called "home" workstation
2. submits jobs for execution.

- When the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently,

1. it transfers one or more of the processes from the user's workstation to some other workstation that is currently idle and gets the process executed there,
2. the result of execution is returned to the user's workstation.

Distributed computing system based on the workstation model



Limitations of Workstation Model

- This model is not so simple to implement due to issues like: -

1. How does the system find an idle workstation?

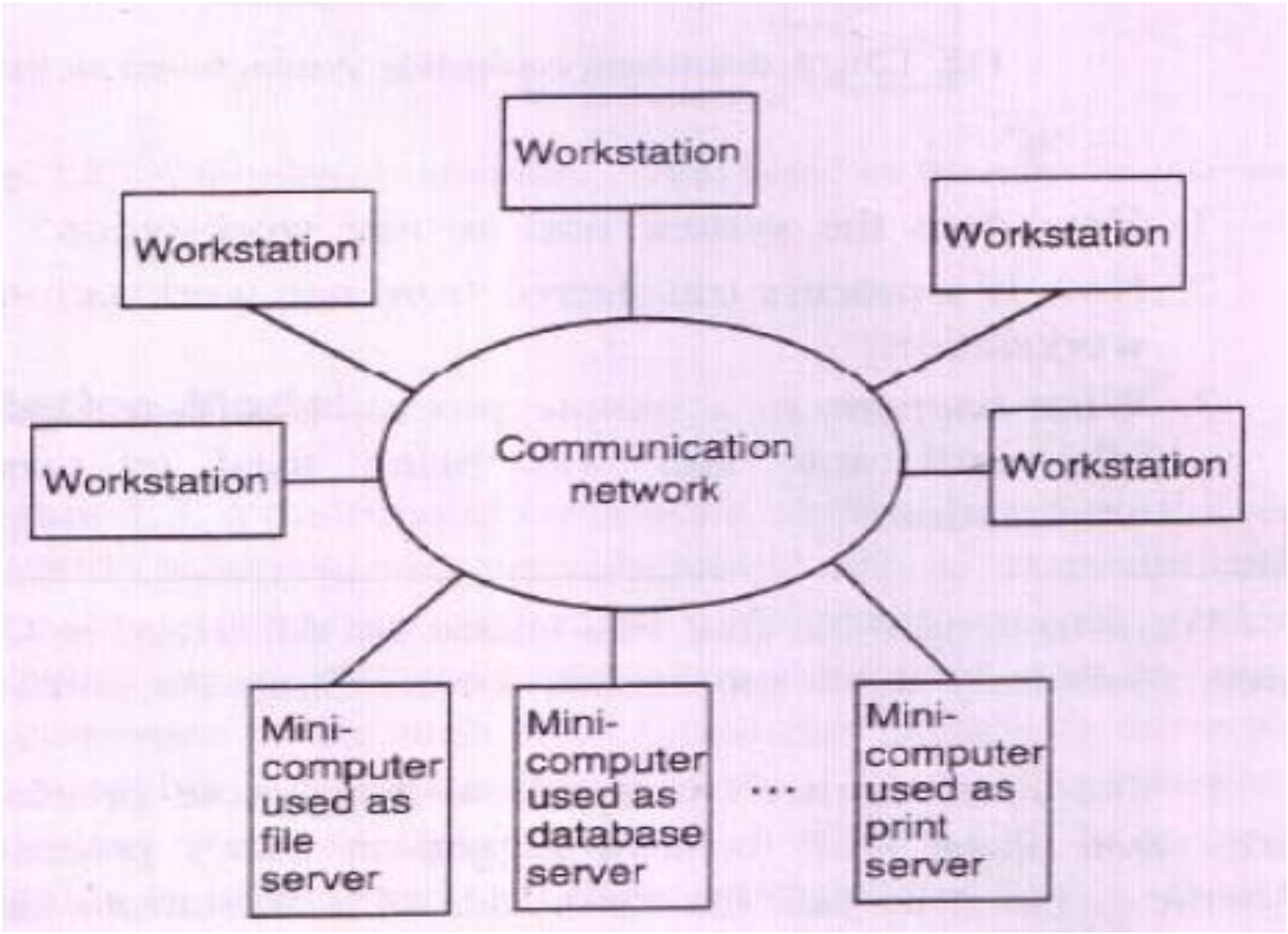
2. How is a process transferred from one workstation to get it executed on another workstation?

3. What happens to a remote process if a user logs onto a workstation that was idle until now and was being used to execute a process of another workstation?

Workstation-Server Model

- It is a network of personal workstations, each with its own disk and a local file system.
- A workstation with its own local disk is called a *diskful* workstation.
- A workstation without a local disk is called a *diskless* workstation.

Distributed computing system based on the workstation server model



- It consists of a few minicomputers and several workstations interconnected by a communication network.

- Minicomputers are used for implementing the file system and other for providing other types of services, such as database service and print service.

- There are specialized machines (or the specialized workstations) for running server processes for managing and providing access to shared resources.

- For higher reliability and better scalability, multiple servers are used for managing the resources of a particular type in a distributed computing system.

ADVANTAGES OF WORKSTATION-SERVER MODEL

- It is cheaper to use a few minicomputers along with large, fast disks that are accessed over the network than a large number of diskful workstations, with each workstation having a small, slow disk.

- Diskless workstations are also preferred to diskful workstations from a system maintenance point of view. (Backup and hardware maintenance are easier).

- All files are managed by the file servers, so users have the flexibility to use any workstation and access the files in the same manner irrespective of which workstation the user is currently logged on.

- A user has guaranteed response time because workstations are not used for executing remote processes.

- **LIMITATION:** the model does not utilize the processing capability of idle workstations.

Request-Response Protocol

- Also known as the **client-server model** of communication.

- A client process sends a request to a server process for getting some service such as reading a block of a file.
- The server executes the request and sends back a reply to the client that contains the result of request processing.

- Provides effective general-purpose approach to the sharing of information and resources in distributed computing systems.

- It is possible for both the client and server processes to be run on the same computer.

- Some processes are both client and server processes. (a server process may use the services of another server).

Processor-Pool Model

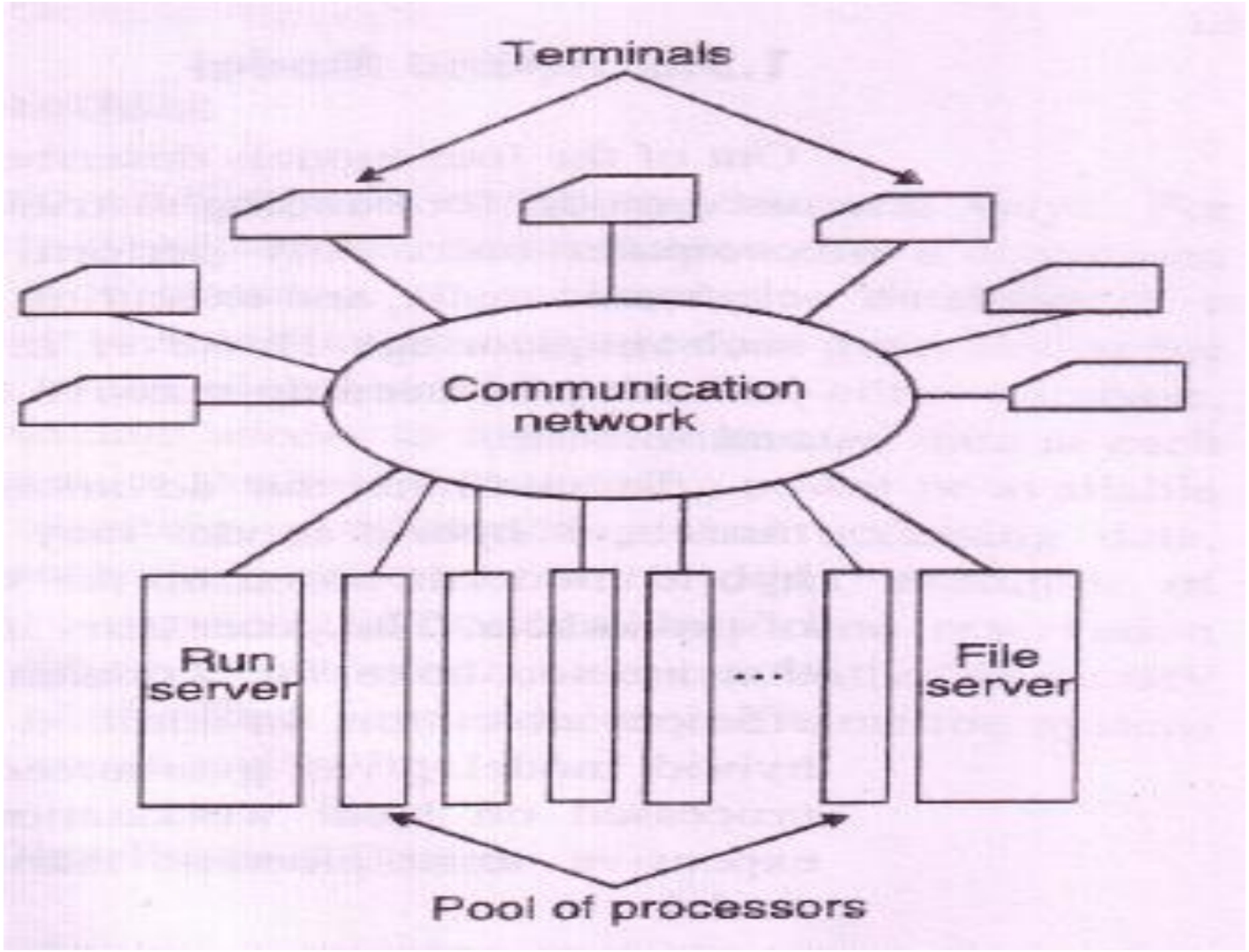
- It is based on the observation that most of the time a user does not need any computing power but once in a while user may need a very large amount of computing power for a short time

- In workstation-server model a processor is allocated to each user.
- In the processor pool model the processors are pooled together to be shared by the users as needed.

- The pool of processors consists of a large number of microcomputers and minicomputers attached to the network.

- Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system.

Distributed computing system based on processor-pool model



Hybrid Model

- **Combine the advantages of both the workstation-server and processor-pool models.**

- Is based on the workstation-server model with the addition of a pool of processors.

- The processors in the pool can be allocated dynamically for computations:

1. that are too large for workstations or
2. that require several computers concurrently.

- Gives guaranteed response to interactive jobs by allowing them to be processed on local workstations of the users.

- **Limitation:** More expensive to implement than the workstation-server model or the processor-pool model.

Limitations of Distributed Computing Systems

- Distributed computing systems are **complex** and **difficult to build** than traditional centralized systems.

- **Complexity** is mainly due to:
  - Using and managing a very large number of distributed resources,
  - Handling the communication and security problems

- The performance and reliability of a distributed computing system depends on the performance and reliability of the communication network.

- Special software is needed to handle loss of messages or to prevent overloading of the network, which degrades the performance and responsiveness to the users.

- Special software security measures are needed to protect the widely distributed shared resources and services against intentional or accidental violation of access control and privacy constraints.

Advantages of Distributed Computing Systems

- Despite the increased complexity and the difficulty of building distributed computing systems, **the installation and use of distributed computing systems is increased.**

- The technical needs, the economic pressures, and the major advantages that have led to the emergence and popularity of distributed computing systems

…….contd

Inherently Distributed Applications

- Several applications are **inherently distributed** in nature and require a distributed computing system for their realization.

- **Examples:** airline reservation system, banking system and a factory automation system controlling robots and machines all along an assembly line.

- These applications require that some processing power be available at many distributed locations for

- ❑ collecting,
- ❑ preprocessing and
- ❑ accessing data.

RESOURCE SHARING

- Sharing of software resources (such as software libraries, databases, hardware resources) can be done in a very effective way among all the computers and the users of a single distributed computing system.

Better Price-Performance Ratio

- With increasing power
- With the increasing speed of communication networks

distributed computing systems have a much better price-performance ratio than a single large centralized system.

- Small number of CPUs based on the processor-pool model can be used by a large number of users from inexpensive terminals, giving a high price-performance ratio as compared to either a centralized time-sharing system or a personal computer.

- They also facilitate resource sharing among multiple computers.

Shorter Response Times and Higher Throughput

- Multiple processors can be utilized properly for providing shorter response times and higher throughput than a single-processor centralized system.

- If a particular computation can be partitioned into a number of subcomputations that can run concurrently, in a distributed computing system all the subcomputations can be simultaneously run with each one on a different processor.

Higher Reliability

- **Reliability refers to the degree of tolerance against errors and component failures in a system.**
- Multiplicity of storage devices and processors allows the maintenance of multiple copies of critical information within the system.
- Execution of important computations redundantly protect them against failures.
- If one of the processors fails, the computation can be successfully completed at the other processor,
- If one of the storage devices fails, the information can be used from the other storage device.
- The geographical distribution of the processors and other resources in a limits the scope of failures caused by natural disasters.
- **Availability, refers to the fraction of time for which a system is available for use.** In comparison to a centralized system, a distributed computing system enjoys the advantage of increased availability.

Better Flexibility in Meeting Users' Needs

- Different types of computers are suitable for performing different types of computations.

- Example, computers with ordinary power are suitable for ordinary data processing jobs, whereas high-performance computers are more suitable for complex mathematical computations.

- A distributed computing system have a pool of different types of computers.
- Most appropriate one can be selected for processing a user's job depending on the nature of the job.

- Example: in a distributed computing system based on the hybrid model, interactive jobs can be processed at a user's own workstation and the processors in the pool may be used to process non-interactive, computation-intensive jobs.

INTRODUCTION

- The operating systems used for Distributed Computing Systems (DCS) is classified into two **Network Operating Systems (NOS)** and **Distributed Operating Systems (DOS).**

- Difference between **NOS and DOS:**

- In a NOS the users view the DCS as a collection of distinct machines connected by a communication subsystem. The users are aware that multiple computers are being used.

  A DOS hides the existence of multiple computers and provides a single-system image to its users. It makes a collection of networked machines act as a virtual uniprocessor.

- In NOS each computer of the DCS has its own local operating system and there is no coordination at all among the computers except for the rule that when two processes of different computers communicate with each other they must use a mutually agreed on communication protocol.

  While with a DOS there is a single system wide operating system and each computer of the DCS runs a part of this global operating system.

- Fault tolerance capability of a DOS is usually very high as compared to that of a NOS.

What is Distributed Operating Systems (DOS)?

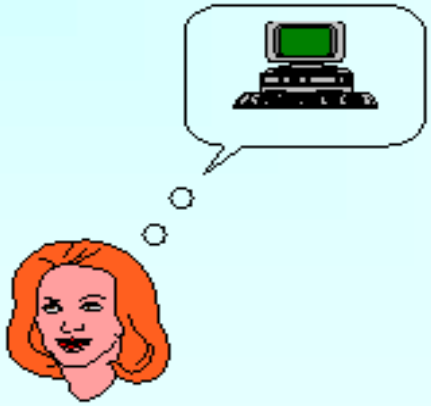
- A **distributed operating system** is one that looks to its users like an ordinary centralized operating system but runs on multiple, independent CPUs.

- The key concept is **transparency** (the use of multiple processors should be invisible (transparent) to the user).

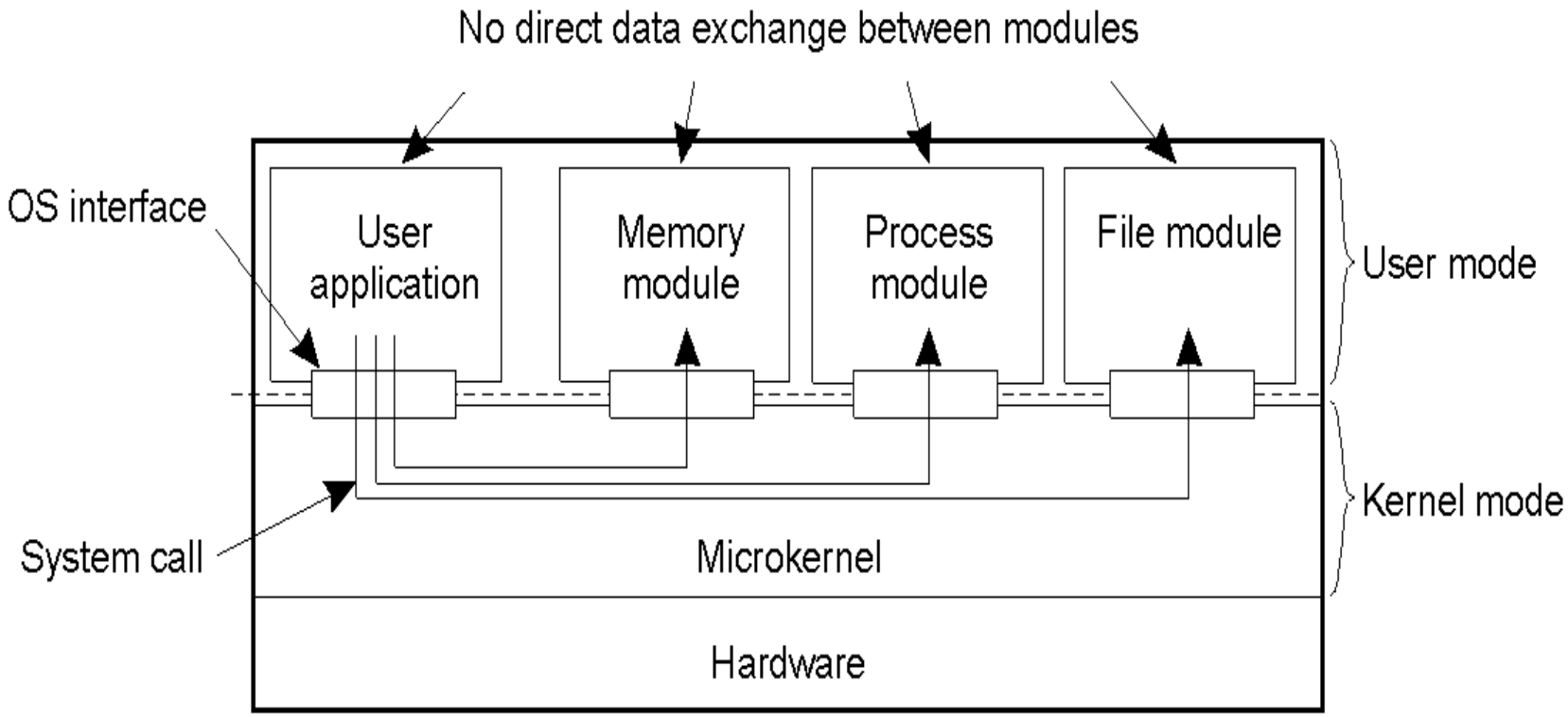
- A distributed computing system that uses a network operating system is referred as a **NETWORK SYSTEM**, and one that uses a distributed operating system is referred as a **TRUE DISTRIBUTED SYSTEM**.

A Distributed Operating System

Goal: An operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.



Uniprocessor Operating Systems



Multicomputer Operating Systems