Information Technology Institute

Operating System Fundamentals

Table of Content

1. Overview
    1. Introduction (Lect 1)
    2. Computer System Structure(Lect 2)
    3. Operating System Structure (Lect 2)
2. Process Management
    1. Processes (Lect 3)
    2. CPU Scheduling (Lect 4)
    3. Deadlocks (Lect 5)
3. Storage Management
    1. Memory Management (Lect 5)
    2. Virtual Memory (Self Study)
    3. File Management (Self Study)
4. Introduction to Cloud Computing (Lect 6)

Self Study Topics

- Virtual Memory
  - Background.
  - Demand Paging.
  - Page Replacement.
  - Allocation of frames.
- File-System Interface
  - File Concept.
  - Access Methods.
  - Directory Structure.
  - Protection.

Self Study Topics (cont'd)

- File-System Implementation
  - File System Structure.
  - Allocation Methods.
  - Free-Space Management.
  - Directory Implementation.
  - Recovery.

Reference

- **Computer Operating System Concepts**
  - Author: Silberschatz
  - Publisher: Wiley
  - ISBN: 0471250600
- **Handbook of Cloud Computing**
  - Author: Borko Furht, Armando Escalante
  - Publisher: Springer
  - ISBN: 978-1-4419-6523-3

Chapter One

Introduction

Table of Content

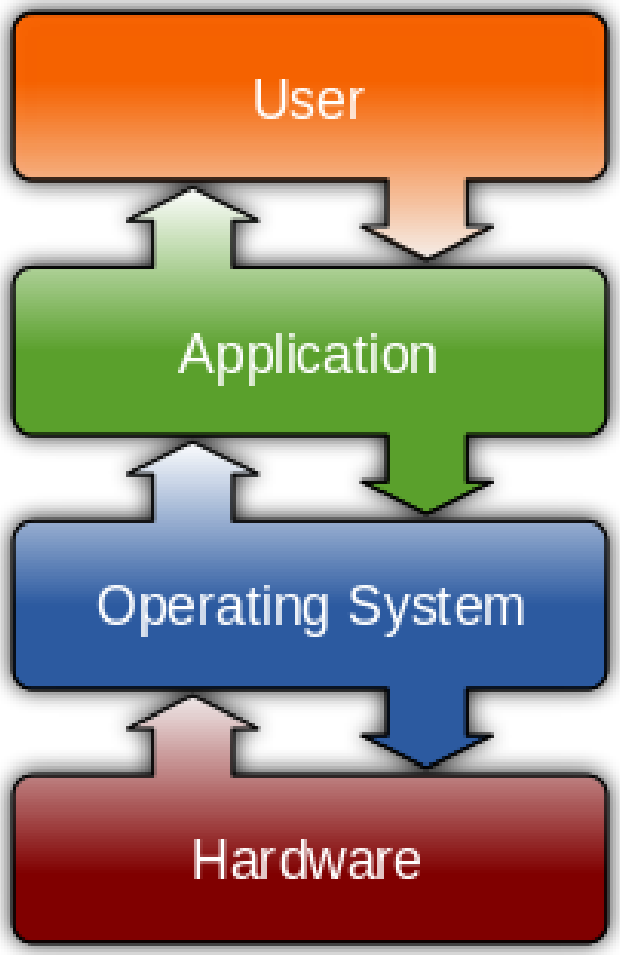
- Operating System
- Mainframe Systems
- Desktop Systems
- Multiprocessor Systems
- Distributed Systems
- Clustered System
- Real -Time Systems
- Handheld Systems
- Computing Environments

OPERATING SYSTEM

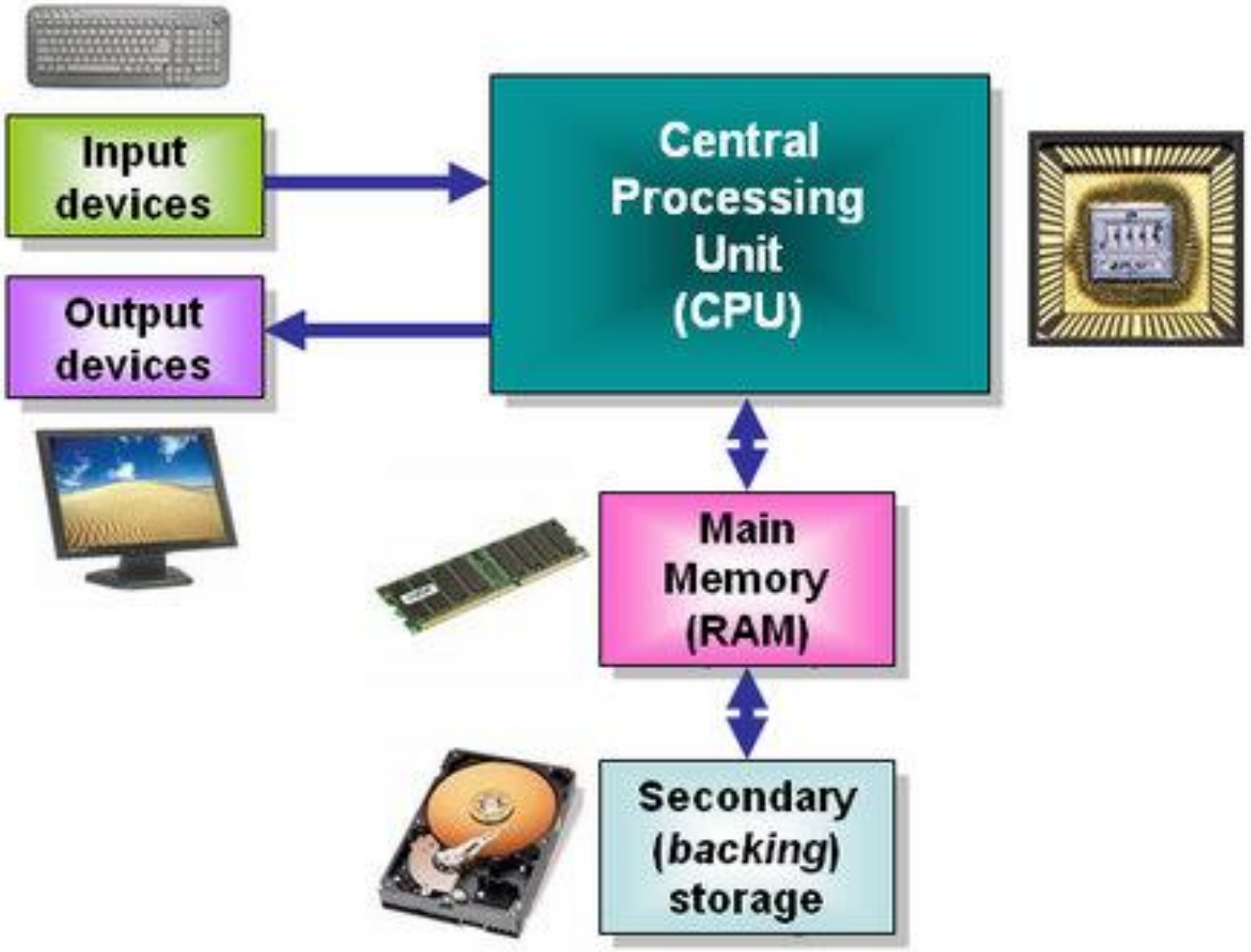
Operating System

- What is an Operating System?
  - It acts as an intermediary between a user and his hardware
- Operating system objective
  - Executes users programs
  - Solves its problems
  - Uses HW in an efficient manner
  - Makes user life easier ;)

Computer System Components



1. Computer Hardware



2. Operating System

- It controls and coordinates the use of the HW among the various application programs for the various users
  - It manages and allocates resources
  - It controls the execution of user programs and operations of I/O devices
- Kernel – the one program running at all times

3. Application Programs

- Compilers
- Web browsers
- Spread sheets
- Word processors
- …

4. Users

- People
- Machines
- Other Computers

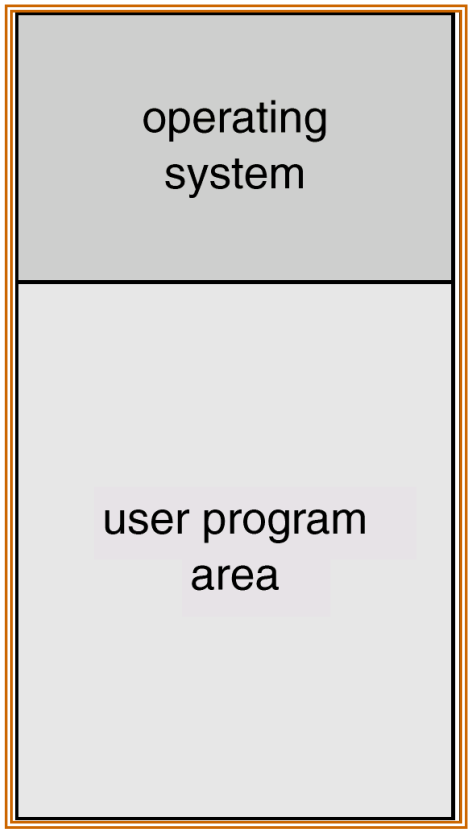
MAINFRAME SYSTEMS

Mainframe Systems

- Reduce setup time by batching similar jobs
- Automatic job sequencing
  - Automatically transfers control from one job to another.
  - First rudimentary operating system

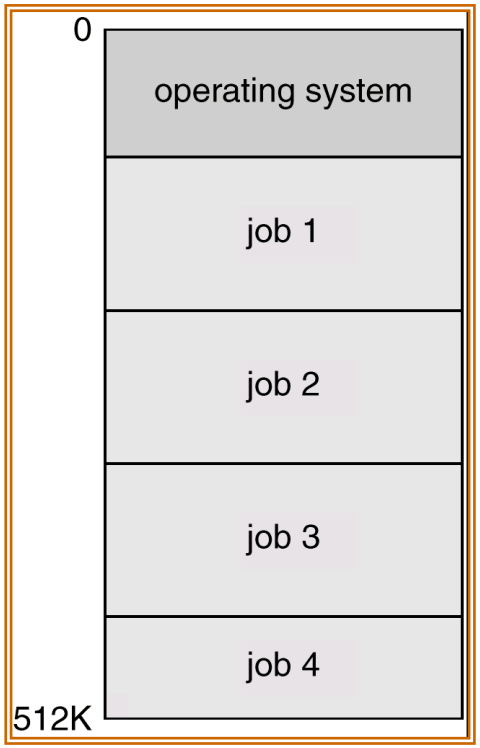
Mainframe Systems Cont'd

- Memory Layout for a Simple Batch System



Mainframe Systems Cont'd

- Multi-programmed Batch Systems
  - Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them



Mainframe Systems Cont'd

- Time-Sharing Systems (Interactive Computing)
  - The CPU is multiplexed among several jobs that are kept in memory and on disk
  - The CPU is allocated to a job only if the job is in memory
  - A job swapped in and out of memory to the disk
  - On-line communication between the user and the system is provided
    - When the operating system finishes the execution of one command, it seeks the next "control statement" from the user's keyboard
  - On-line system must be available for users to access data and code

DESKTOP SYSTEMS

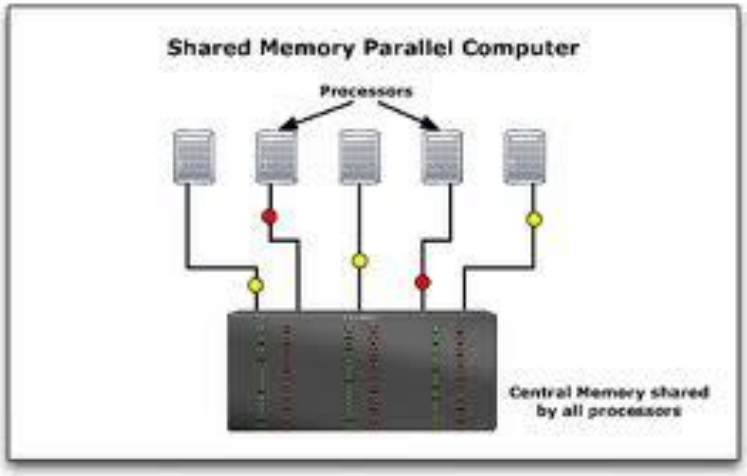
Desktop Systems

- *Personal computers*
  - Computer system dedicated to a single user
- I/O devices
  - Keyboards
  - Mice
  - Display screens
  - Small printers
- User convenience and responsiveness
- Can adopt technology developed for larger operating system
  - Often individuals have sole use of computer and do not need advanced CPU utilization of protection features
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

MULTIPROCESSOR SYSTEMS

Parallel Systems

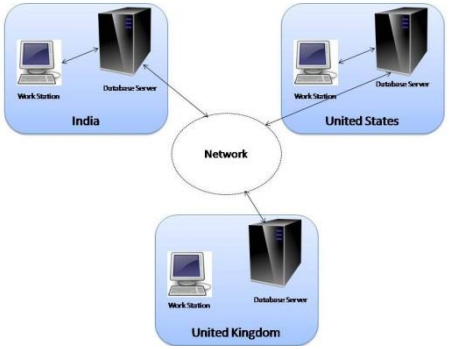
- Systems with more than one CPU in close communication
  - Also known as *multiprocessor systems*
- *Tightly coupled system*
  - processors share memory and a clock; communication usually takes place through the shared memory
- Advantages of parallel system:
  - Increased *throughput*
  - Economical
  - Increased reliability
    - graceful degradation
    - fail-soft systems



DISTRIBUTED SYSTEMS

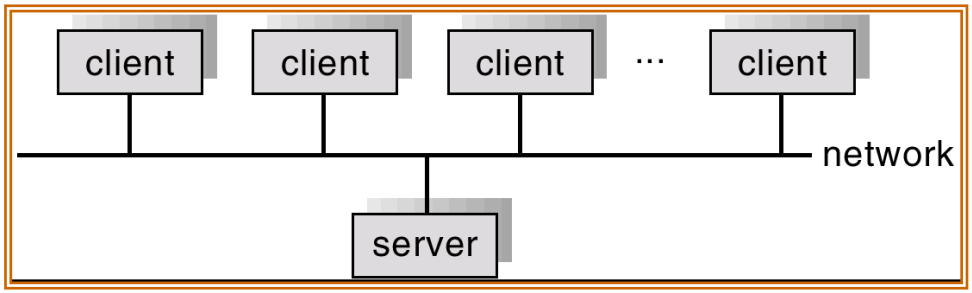
Distributed Systems

- Distribute the computation among several physical processors
- *Loosely coupled system*
  - Each processor has its own local memory
  - processors communicate with one another through various communications lines, such as high-speed buses or telephone lines
- Advantages of distributed systems
  - Resources Sharing
  - Computation speed up
    - load sharing
  - Reliability



Distributed Systems Cont'd

- Requires networking infrastructure
- Local area networks (*LAN*) or Wide area networks (*WAN*)
- May be either *client-server* or *peer-to-peer* systems



CLUSTERED SYSTEMS

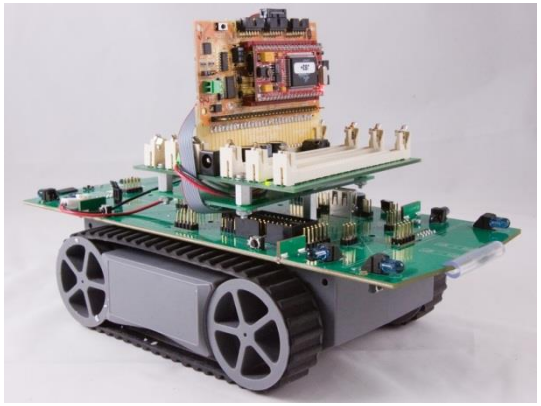
Clustered Systems

- Clustering allows two or more systems to share storage
- Provides high reliability
- *Asymmetric clustering*: one server runs the application or applications while other servers standby
- *Symmetric clustering*: all N hosts are running the application or applications

REAL-TIME SYSTEMS

Real-Time Systems

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
- Well-defined fixed-time constraints
- Real-Time systems may be either *hard* or *soft* real-time



Real-Time Systems Cont'd

- Hard real-time:
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems
- Soft real-time
  - Limited utility in industrial control of robotics
  - Integrate-able with time-share systems
  - Useful in applications (multimedia, virtual reality) requiring tight response times

HANDHELD SYSTEMS

Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular Phone & Smart Phone
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens

Computing Environments

- Traditional computing
  - PCs, Servers, limited remote access
- Web-Based Computing
  - Client-server and web services, convenient remote access, location-less servers
- Embedded Computing
  - Very limited operating system features
  - Little or no user interface, remote access

Information Technology Institute

Operating System Fundamentals

Chapter Two

COMPUTER SYSTEM STRUCTURE

Table of Content

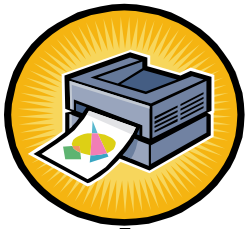
- Computer System Operation
- I/O Structure
- Storage Structure
- Hardware Protection

COMPUTER SYSTEM OPERATION

Computer System Operation

- CPU and device controllers are connected through common busses (data, address, and control).
- CPU and device controllers execute concurrently.
- Memory controller synchronizes access to memory.

Computer System Operation Cont'd



CPU

Hard Disk Controller

Printer Controller

Tape Drive Controller

Memory Controller

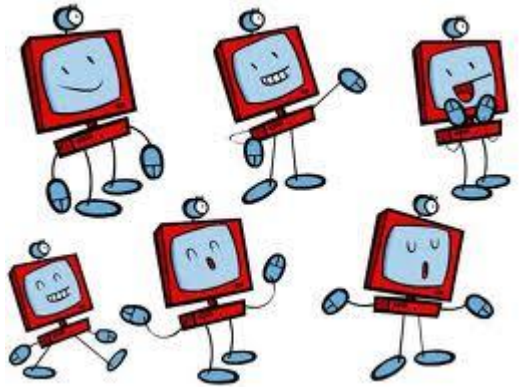
M e m o r y

Computer System Operation Cont'd

- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.

Computer System Startup

1. Power up
2. Initial program: bootstrap
   - Stored in ROM
   - Initialize:
     1. CPU registers
     2. Device controllers
     3. Memory contents
     4. Load the operating system (kernel)
3. Kernel starts the first process, init
4. Init waits for an event (interrupt) to occur



Interrupts

- A signal sent to the CPU
- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupts:
  - Hardware Interrupts
  - Software Interrupts: system calls

Interrupts Cont'd

- A trap is a software generated interrupt caused by:
  - Error: division by zero or invalid memory access
  - Request: from a user program to O/S
- An operating system is interrupt driven.

The 6502's interrupt latency is one of the 8-bit world's fastest.

Interrupt Handling

1. CPU is interrupted
2. CPU stops current process
3. CPU transfers execution to a fixed location
4. CPU executes interrupt service routine
5. CPU resumes process

- Notes:
  - Interrupts must be handled quickly
  - Interrupted process information must be stored

I/O STRUCTURE

I/O Structure

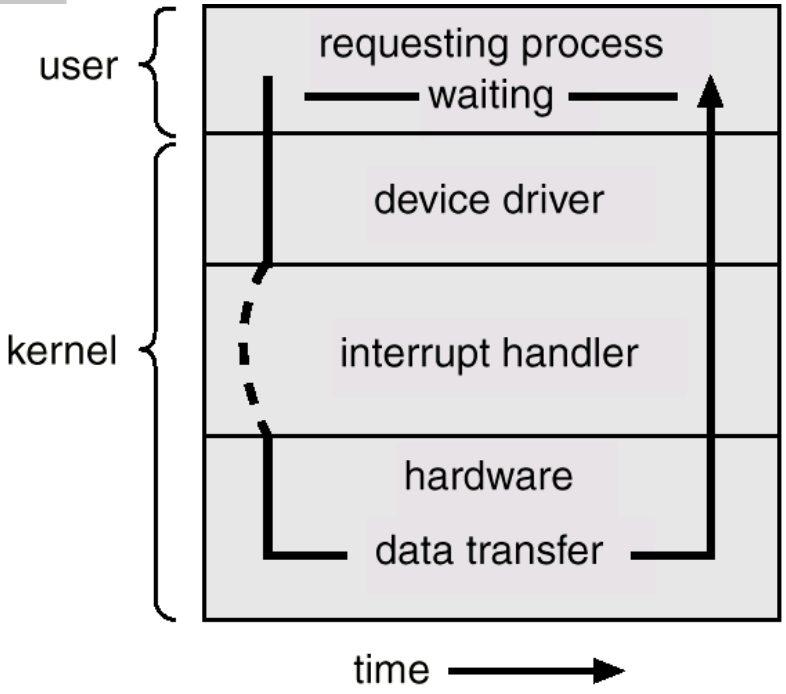
- Controllers:
  - Is in charge of a specific type of device
  - Moves data between device and local buffer
  - A controller may have more than one device
  - Buffer size varies

Two I/O Methods

- Synchronous I/O:
  - Process request I/O operation
  - I/O operation is started
  - I/O Operation is complete
  - Control is returned to the user process
- Asynchronous I/O:
  - Process Request I/O operation
  - I/O operation is started
  - Control is returned immediately to the user process
  - I/O continues while system operations occur

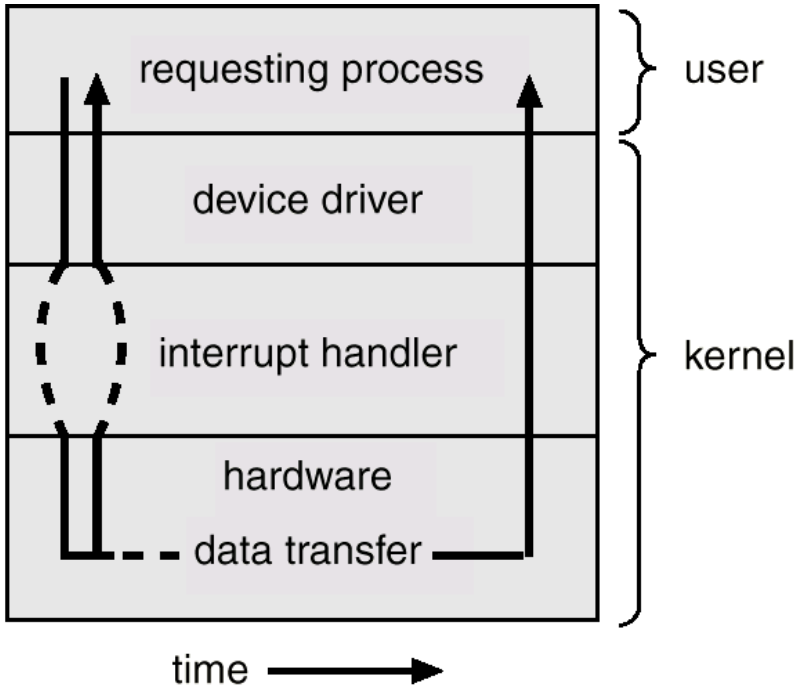
Two I/O Methods Cont'd

Synchronous



(a)

Asynchronous



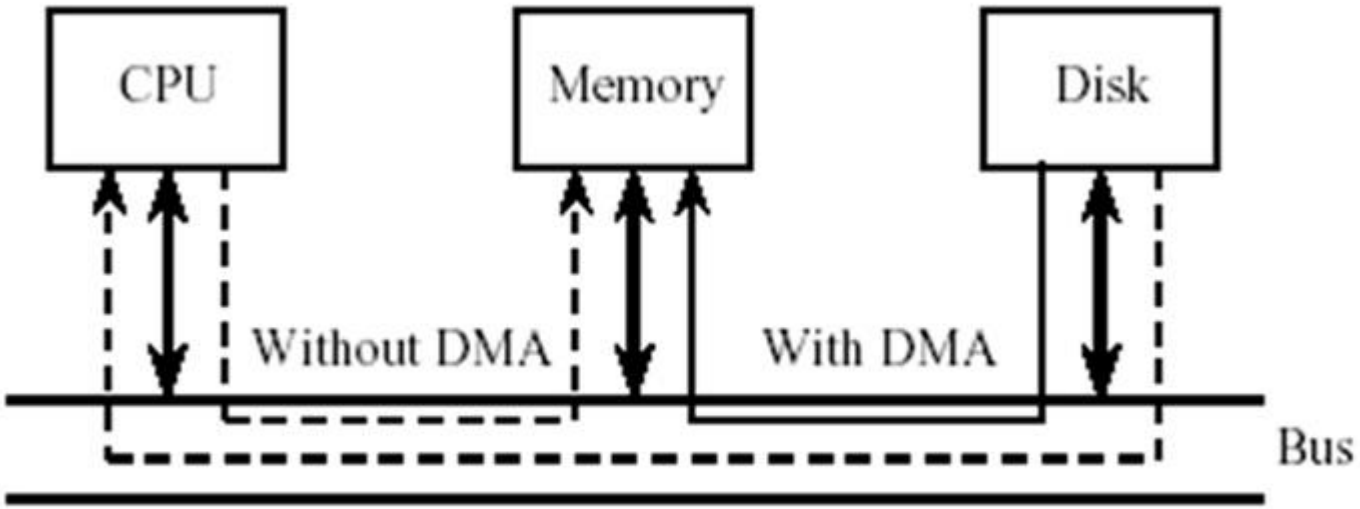
(b)

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.

Direct Memory Access Structure

Cont'd



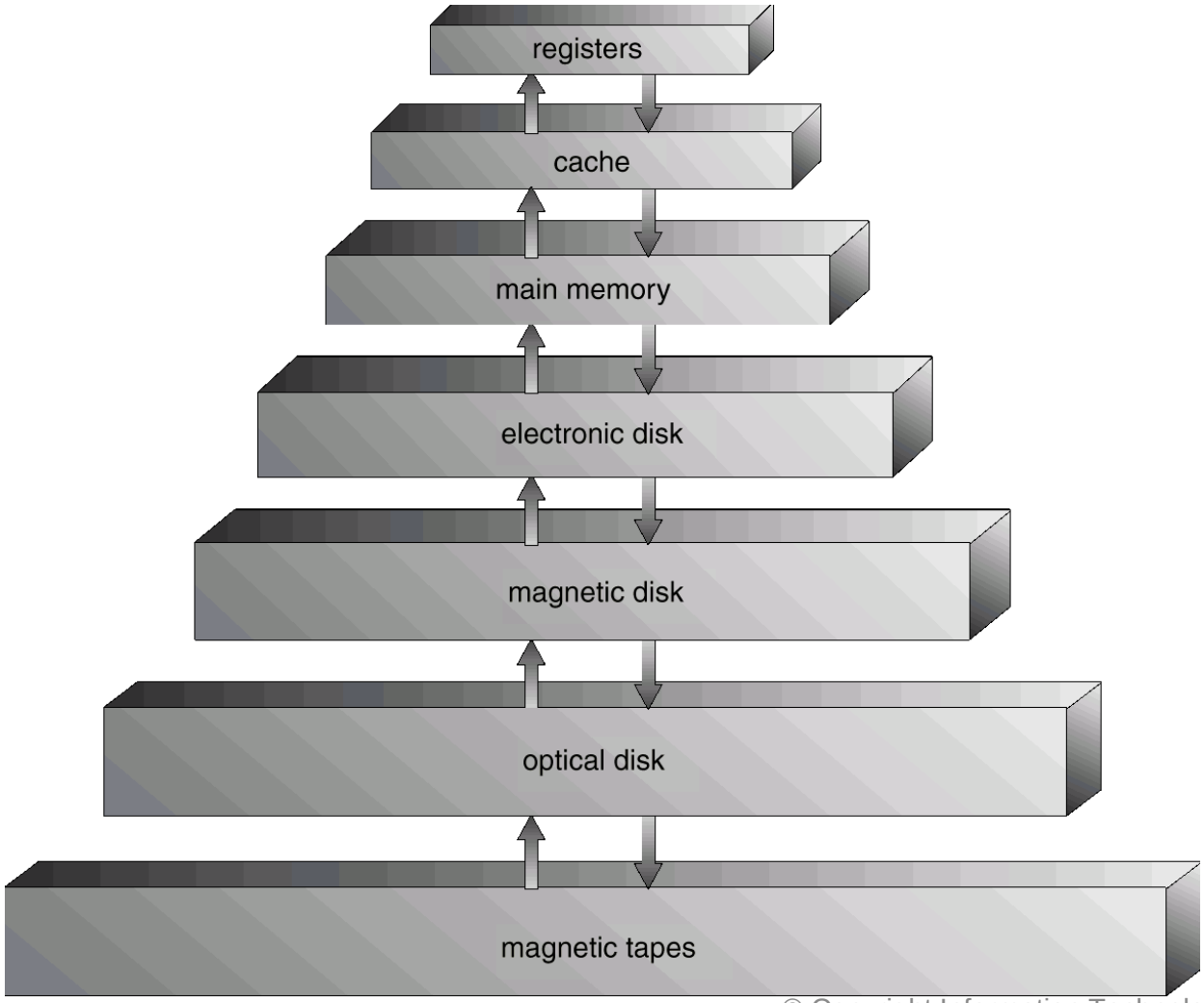
DMA transfer from disk to memory bypasses the CPU.

STORAGE STRUCTURE

Storage Structure

- Main memory
  - Only large storage media that the CPU can access directly.
- Secondary storage
  - Extension of main memory that provides large nonvolatile storage capacity.

Storage-Device Hierarchy



Difference of Storage Devices

- Speed
- Cost
- Capacity
- Volatility
- Reliability
- Portability

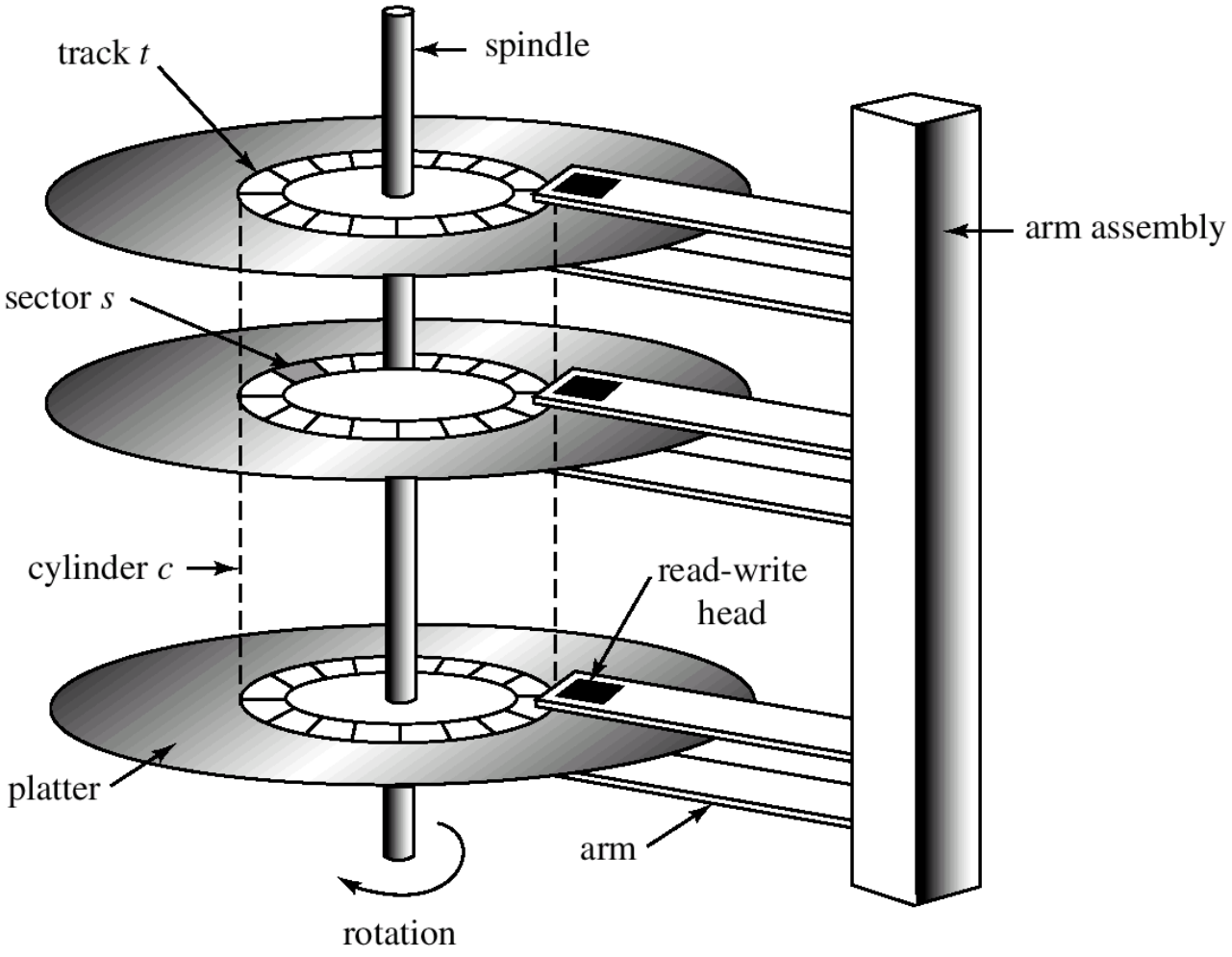
RAM

- Array of memory words
- Each byte has an address
- Memory Address:
  - Physical
  - Logical
- CPU Instructions:
  - Load: moves a word from main memory to CPU register
  - Store: move a word from CPU register to main memory

Magnetic Disk

- Rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors.
  - The disk controller determines the logical interaction between the device and the computer.

Magnetic Disk Cont'd



Magnetic Tapes

- Early secondary storage
- Slow access time
- Usage
  - Backup
  - Storage of infrequently used information

HARDWARE PROTECTION

Hardware Protection

- Early OS:
  - Single user
  - Programmer had full control of hardware
  - Programmer was responsible of I/O
- Error in a program
  - Single task
    - Only one program affected
  - Multi task
    - Could cause problems to other programs
- Desktop OS allow a program to access any part of memory or affect other programs instructions or data

Error Handling

- Errors
  - Illegal instruction
  - Infinite loop
  - Access of other memory addresses

- Handling Errors

1. Errors are detected by hardware
2. Hardware trap the error to the O/S
3. Errors are handled by the O/S
4. O/S terminates process
5. O/S dumps the process to disk (if needed)

Ensuring OS Proper Operation

1. I/O Protection: illegal instructions
   - Dual Mode (System Mode & User Mode)
2. Memory protection: illegal memory access
   - Base & limit registers
3. CPU Protection: infinite loops
   - Timers

Information Technology Institute

Operating System Fundamentals

Chapter Three

OPERATING SYSTEM STRUCTURES

Table of Content

- System Components
- Operating System Services
- System Calls
- System Structure

SYSTEM COMPONENTS

System Components

- Process management
- Main memory management
- File system management
- I/O system management
- Secondary storage management
- Networking
- Protection
- Command interpreter

Process Management

- A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - process communication

Main-Memory Management

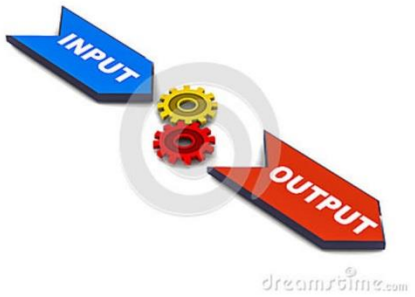
- Memory is a large array of bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and de-allocate memory space as needed

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs and data.
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (non-volatile) storage media.

I/O System Management

- OS hide particularities of I/O devices
  - Device drivers
    - Input: retrieve block of data
    - Output: hardware instructions for controller
- I/O subsystem
  - Memory management: spooling
  - Drivers for specific hardware



Secondary-Storage Management

- Since main memory (primary storage) is volatile and too small to accommodate all data and programs permanently, the computer system must provide secondary storage to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

Networking (Distributed Systems)

- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol*.
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Protection System

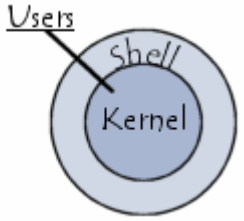
- *Protection* refers to a mechanism for controlling access by programs, or users to system resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.

Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - Protection
  - networking

Command-Interpreter System Cont'd

- The program that reads and interprets control statements is called variously:
  - command-line interpreter
  - shell (in UNIX)
- Its function is to get and execute the next command statement.



OPERATING SYSTEM SERVICES

Operating System Services

- Program execution
  - System capability to load a program into memory and to run it.
- I/O operations
  - Since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation
  - Program capability to read, write, create, and delete files.

Operating System Services Cont'd

- Communications
  - Exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.
- Error detection
  - Ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

SYSTEM CALLS

System Calls

- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

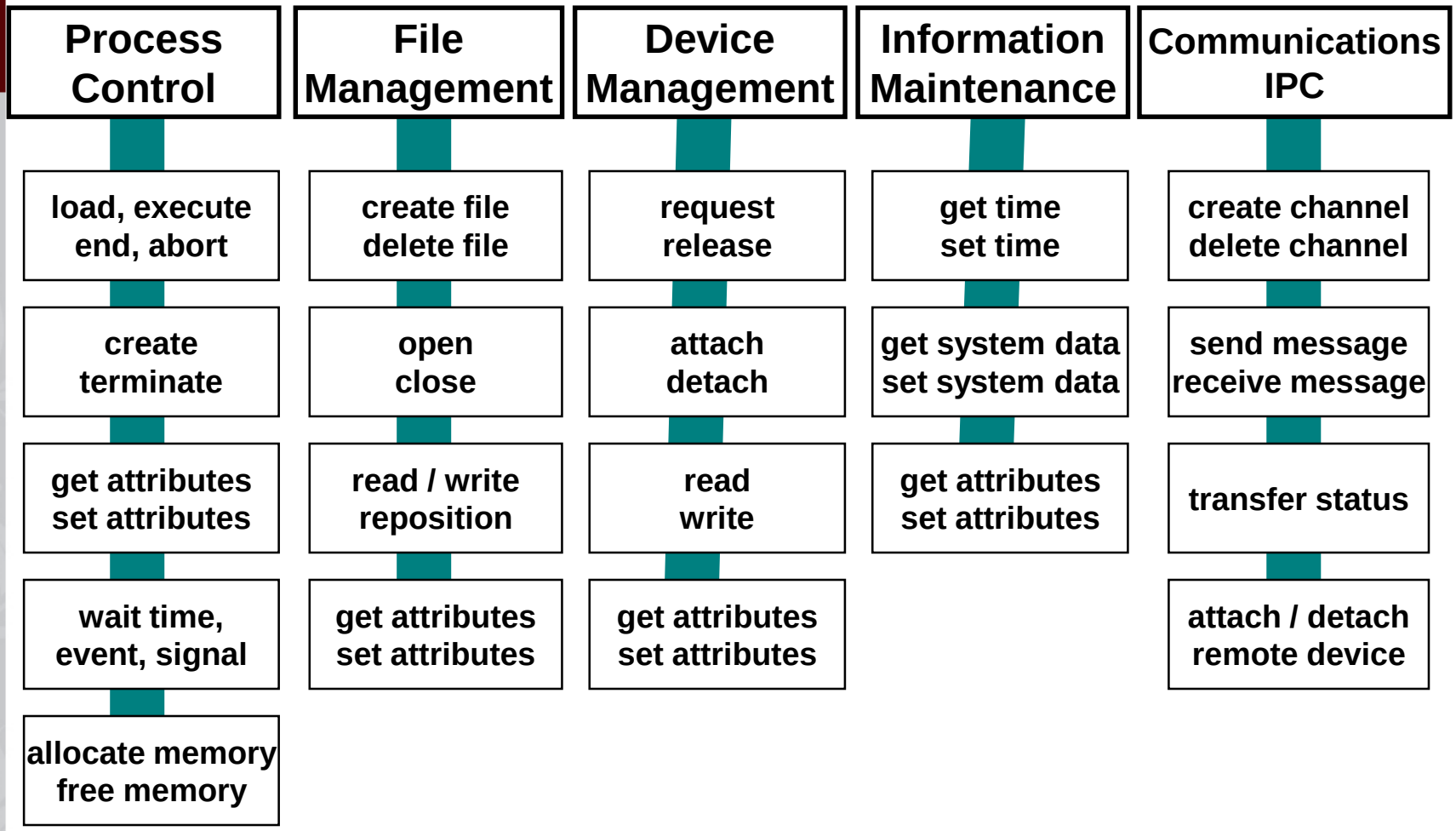
Copy Program Example

- Variable initialization
- open (file1)
- create (file2)
- read (file1)
- write (file2)
- close (file1)
- close (file2)

Types of System Calls

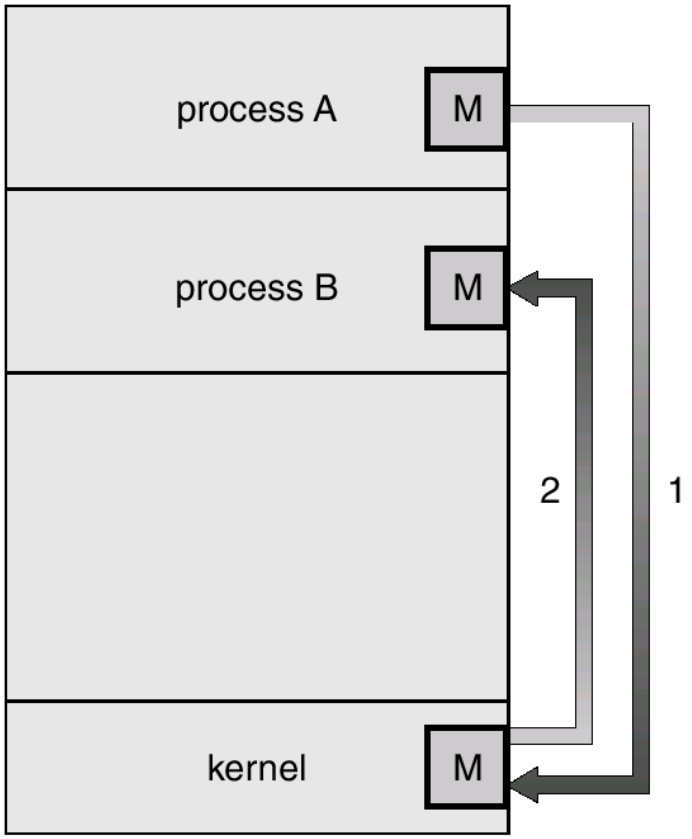
- Process control
- File management
- Device management
- Information maintenance
- Communications

System Calls Types

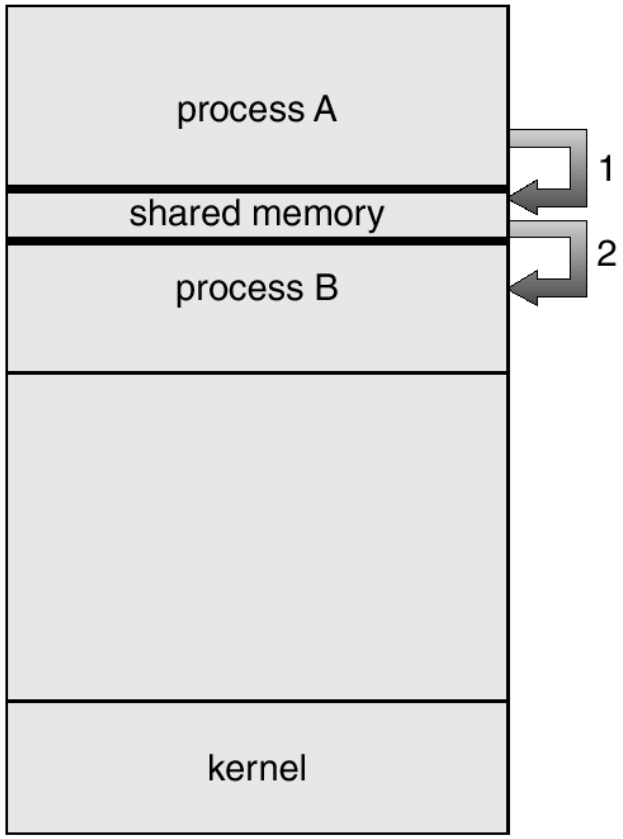


Communication Models

Message Passing



Shared Memory

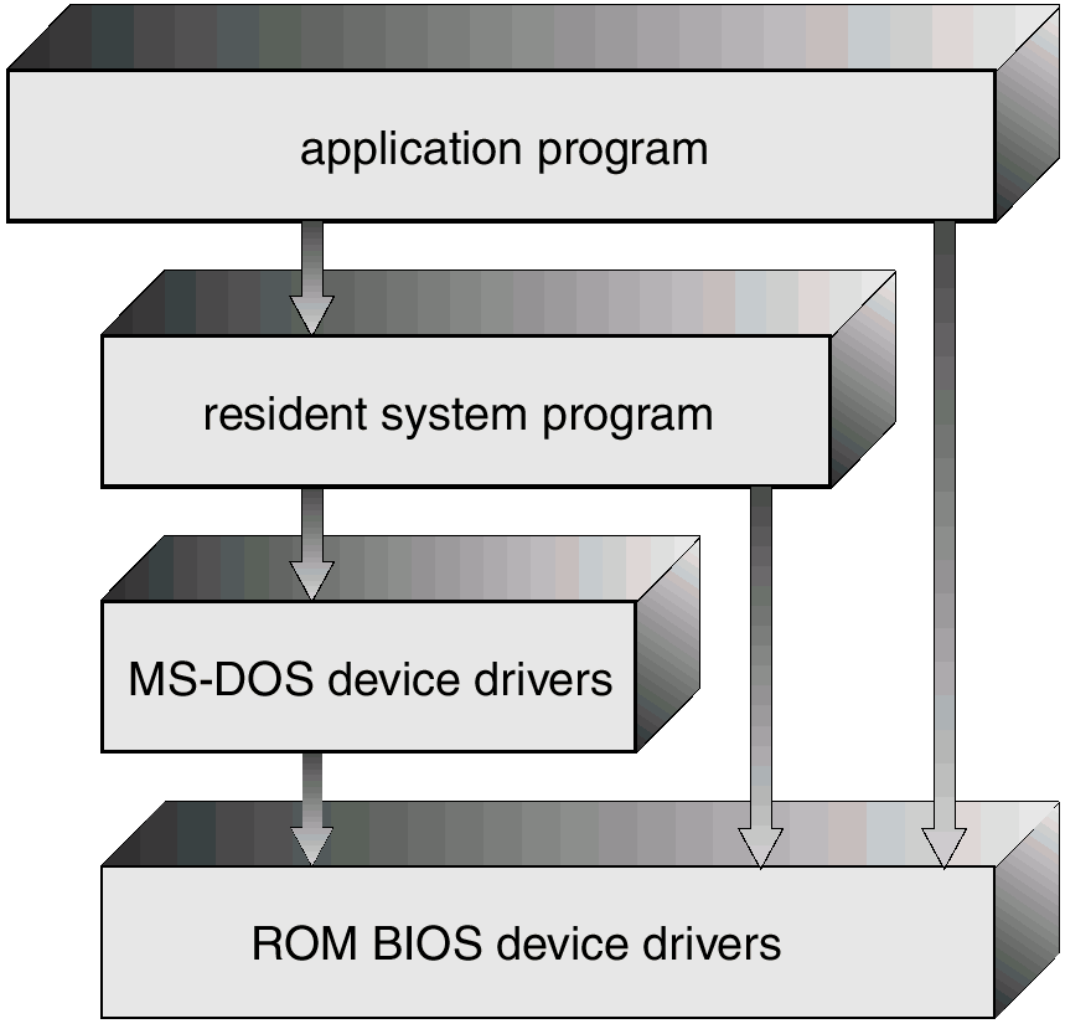


SYSTEM STRUCTURE

OS System Design Structure

- Simple structure
- Layered approach

MS-DOS Structure



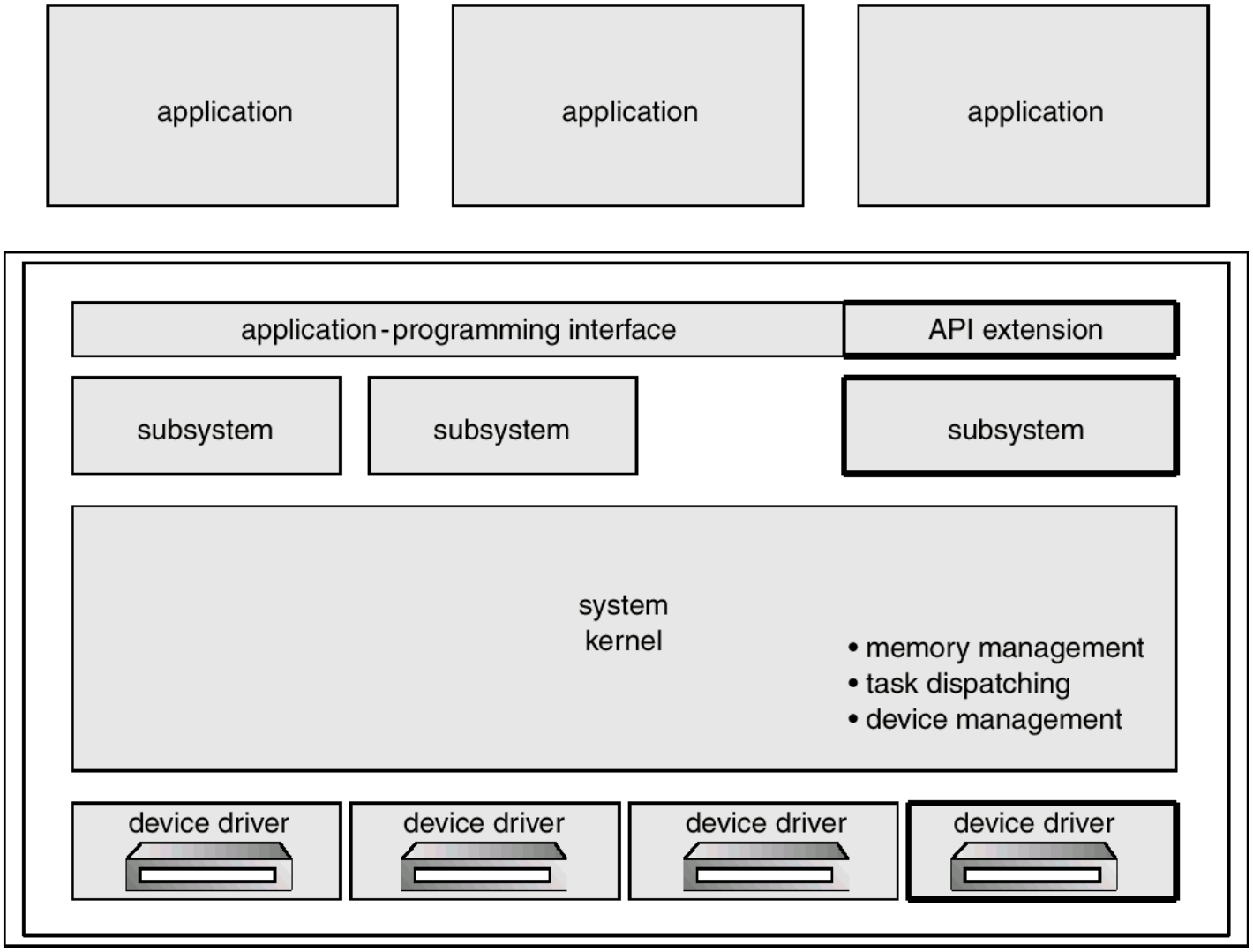
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

Layered Approach Cont'd

- Advantage:
  - Modularity
  - Debugging
  - Modification
- Disadvantage:
  - Layering overhead to the system call

OS/2 Layer Structure



Information Technology Institute

Operating System Fundamentals

Chapter Five

CPU SCHEDULING

Table of Content

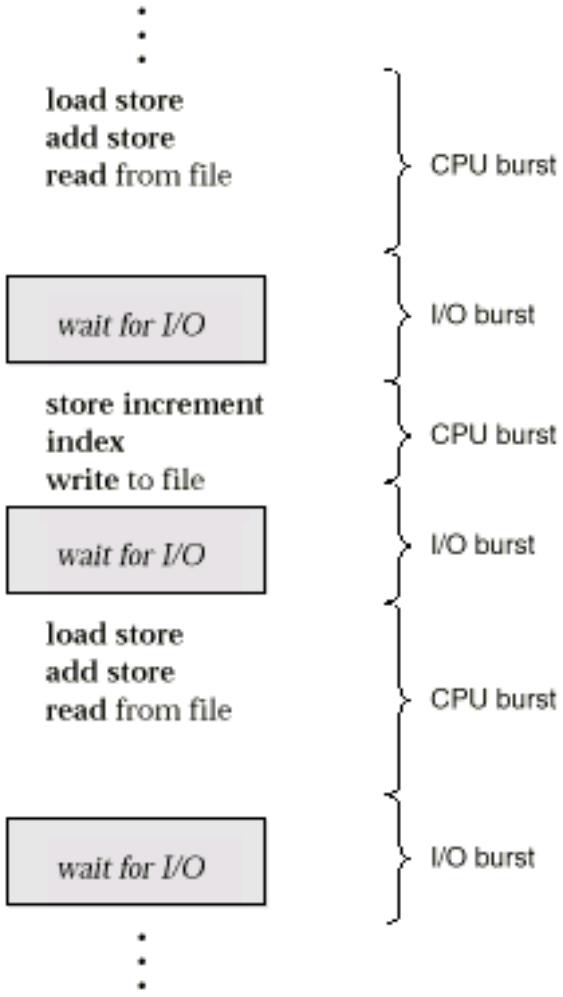
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

BASIC CONCEPTS

Basic Concepts

- Maximum CPU utilization obtained with multitasking
- CPU–I/O Burst Cycle
  - Process execution consists of a cycle of CPU execution and I/O wait.

Alternating Sequence of CPU And I/O Bursts



CPU Scheduler

- Selects from among the processes in memory that are ready to run, and allocates the CPU to one of them.
- 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.

CPU Scheduler Cont'd

- Preemptive
  - Process release the CPU before it finish execution
  - Example: Modern OS: Unix, Linux, Windows7
- Non-preemptive
  - Process release CPU when:
    - Running → Waiting
    - Running → Terminated
  - Example: MS Windows 3.1

Dispatcher

- Gives control of the CPU to the process selected by the short-term scheduler:
  - switching context
  - switching to suitable mode (User or Monitor)
  - jumping to the proper location in the user program to restart that program
- Dispatch latency
  - time taken by dispatcher to stop one process and start another running.

SCHEDULING CRITERIA

Scheduling Criteria

- CPU utilization
  - Keep the CPU as busy as possible
- Throughput
  - Number 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)

Optimization Criteria

- Maximize
  - CPU Utilization
  - Throughput
- Minimize
  - Turnaround time
  - Waiting time
  - Response time

- Considerations
  - Minimize maximum response time
  - Minimize the variance of response times

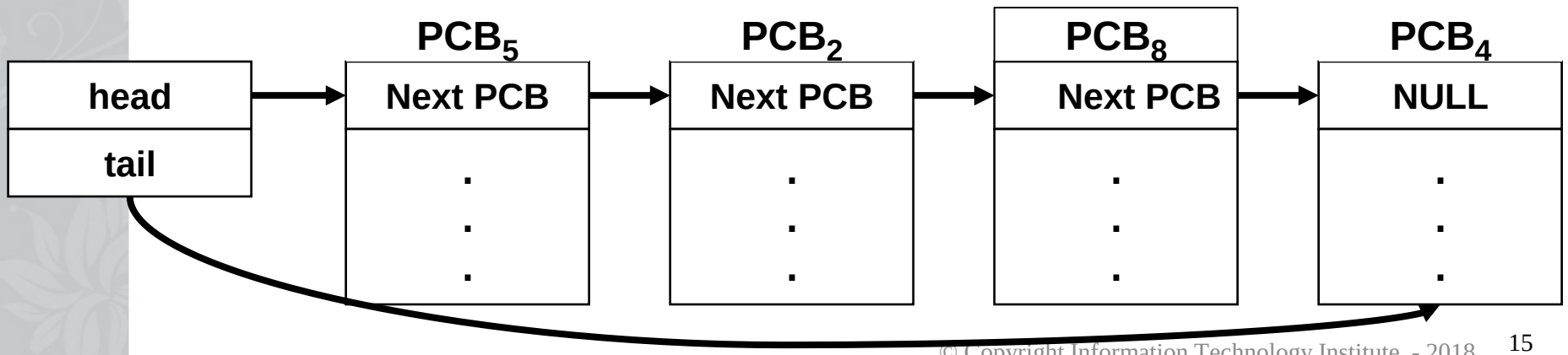
SCHEDULING ALGORITHMS

Scheduling Algorithms

- First-Come First Served
- Shortest-Job First
- Priority
- Round-Robin

First-Come, First-Served (FCFS) Scheduling

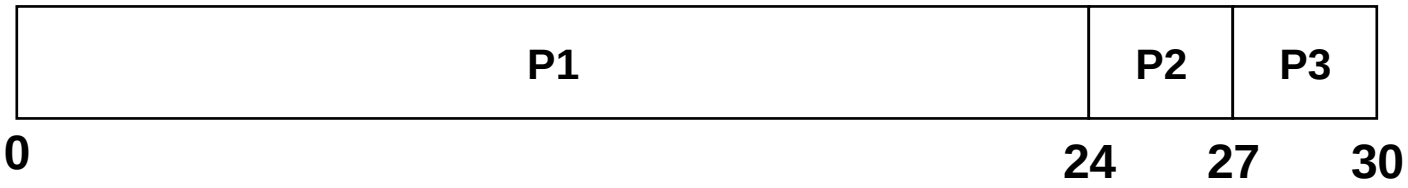
- Easily implemented
- Ready queue is FIFO
- P_n ready → P_n PCB is linked to tail of queue
- Process at head of ready queue → CPU
- Average waiting time is long!



Example 1

Process	Burst Time
P1	24
P2	3
P3	3

- Suppose that the processes arrive in the order: *P1* , *P2* , *P3* The Gantt Chart for the schedule is:

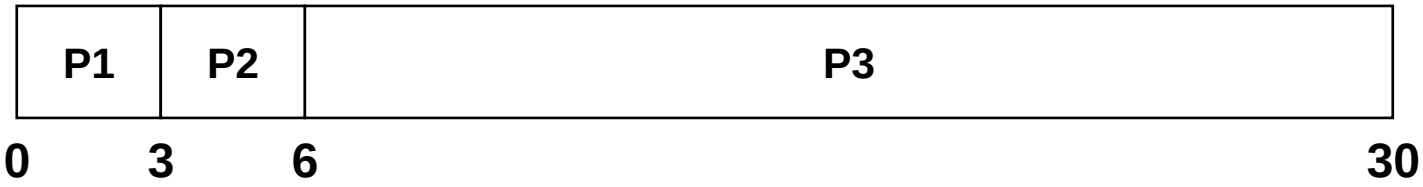


- Waiting time for *P1* = 0; *P2* = 24; *P3* = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Example 2

Process	Burst Time
P1	3
P2	3
P3	24

- Suppose that the processes arrive in the order: *P1* , *P2* , *P3* The Gantt Chart for the schedule is:



- Waiting time for *P1* = 0; *P2* =3; *P3* = 6
- Average waiting time: (0 + 3 + 6)/3 = 3

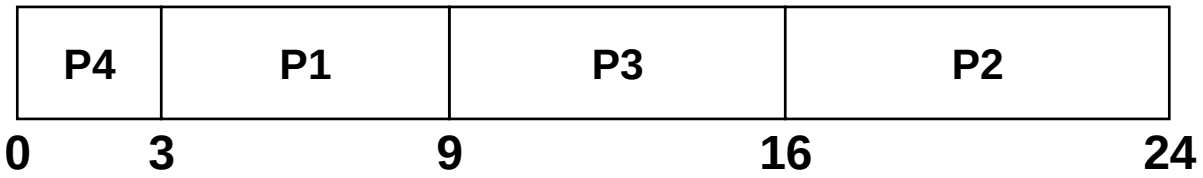
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal
  - Gives minimum average waiting time for a given set of processes.

Example 1

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

- Suppose that all processes arrive at the same time: The Gantt Chart for the schedule is:

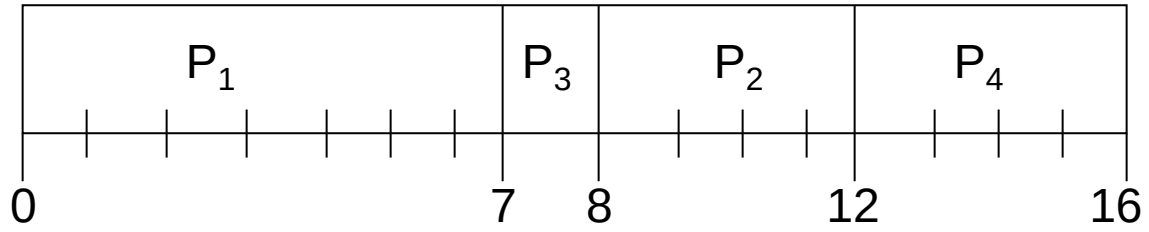


- Waiting time for *P1* = 3; *P2* =16; *P3* =9; P4=0
- Average waiting time: (3 + 16 + 9 + 0)/4 = 7

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (non-preemptive)

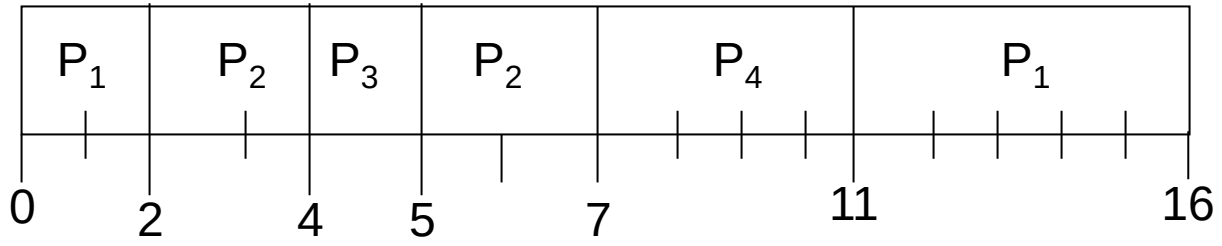


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

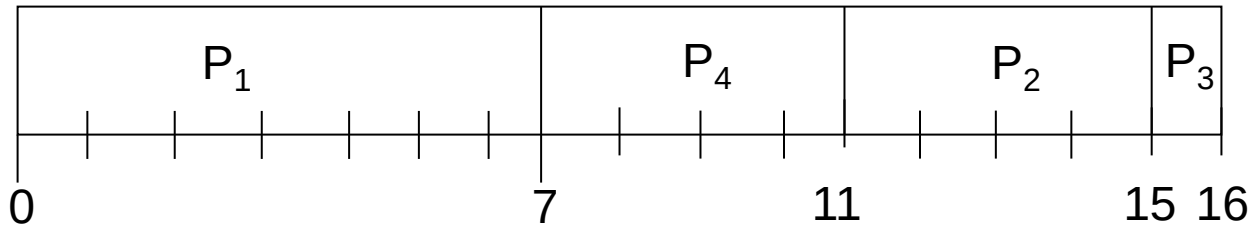
Priority Scheduling

- Priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
  - Preemptive
  - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
  - Problem ≡ Starvation – low priority processes may never execute.
  - Solution ≡ Aging – as time progresses increase the priority of the process.

Example of Non-Preemptive Priority

Process	Arrival Time	Burst Time	Priority
P1	0.0	7	3
P2	2.0	4	2
P3	4.0	1	4
P4	5.0	4	1

- Priority (Non-Preemptive)

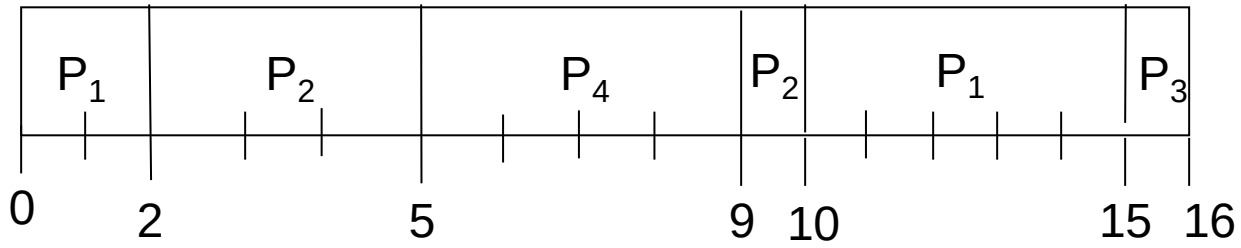


- Average waiting time = (0 + 9 + 11 +2)/4 = 22/4

Example of Preemptive Priority

Process	Arrival Time	Burst Time	Priority
P1	0.0	7	3
P2	2.0	4	2
P3	4.0	1	4
P4	5.0	4	1

- Priority (Preemptive)



- Average waiting time = (8 + 4 + 11 +0)/4 = 23/4

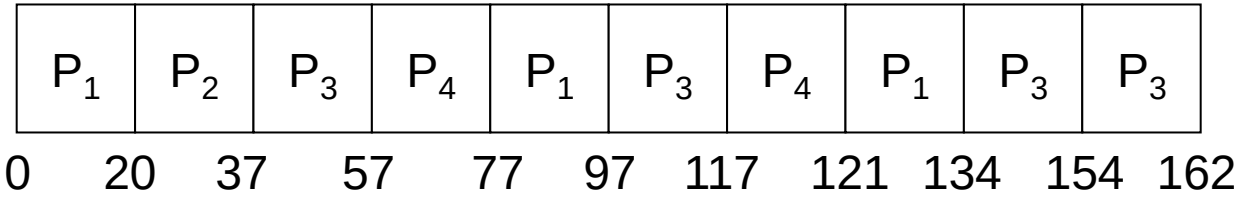
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), 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*-1)*q* time units.
- Performance
  - *q* large FIFO
  - *q* small *q* must be large with respect to context switch, otherwise overhead is too high.

Example of RR, Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

- The Gantt chart is:



*Note: higher average turnaround than SJF, but better response.

Information Technology Institute

Operating System Fundamentals

Chapter Four

PROCESSES

Table of Content

- Process Concept
- Process Scheduling
- Operations on Processes
- Threads
- Cooperating Processes
- Inter-process Communication

PROCESS CONCEPT

Process Concept

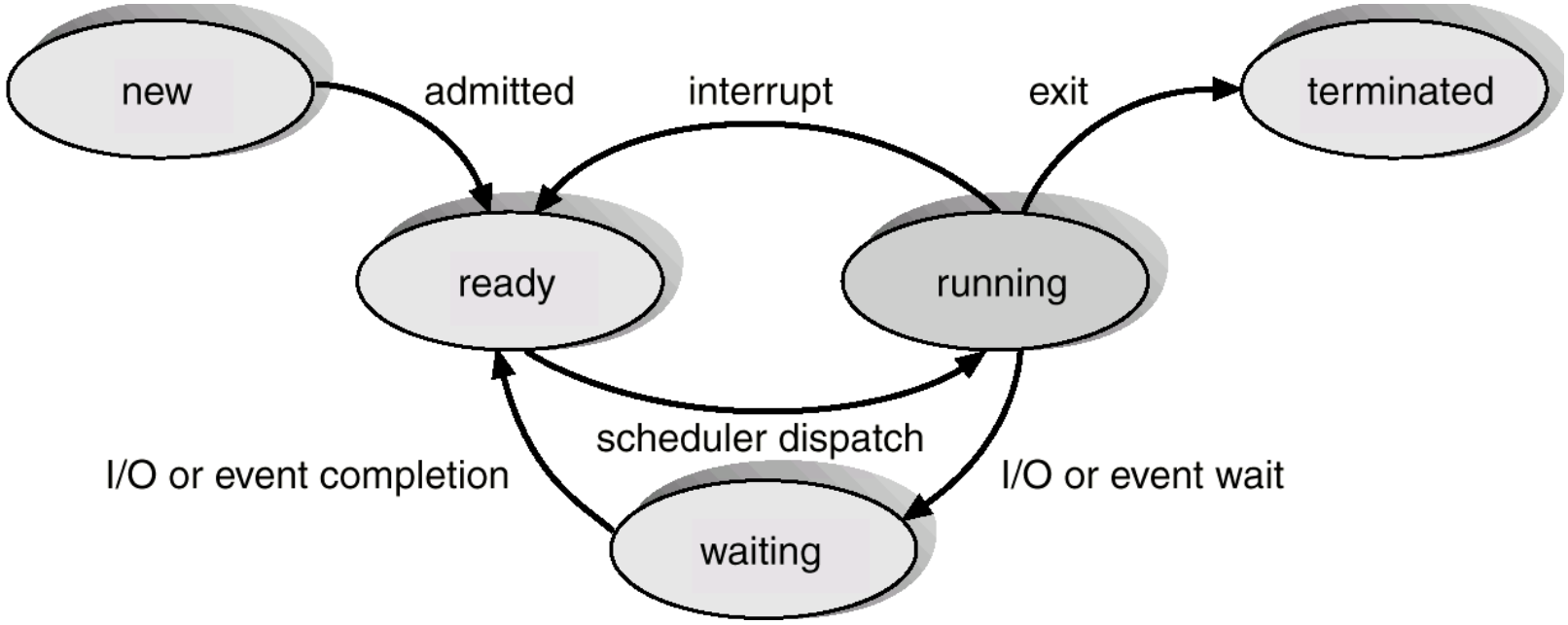
- Process – a program in execution; process execution must progress in sequential fashion.
- An operating system executes a variety of programs:
  - Batch system
  - Time-shared systems

* Textbook uses the terms job and process almost interchangeably

Process Contents

- Text section
  - Program instructions
- Program counter
  - Next instruction
- Stack
  - Local variables
  - Return addresses
  - Method parameters
- Data section
  - Global variables

Process State



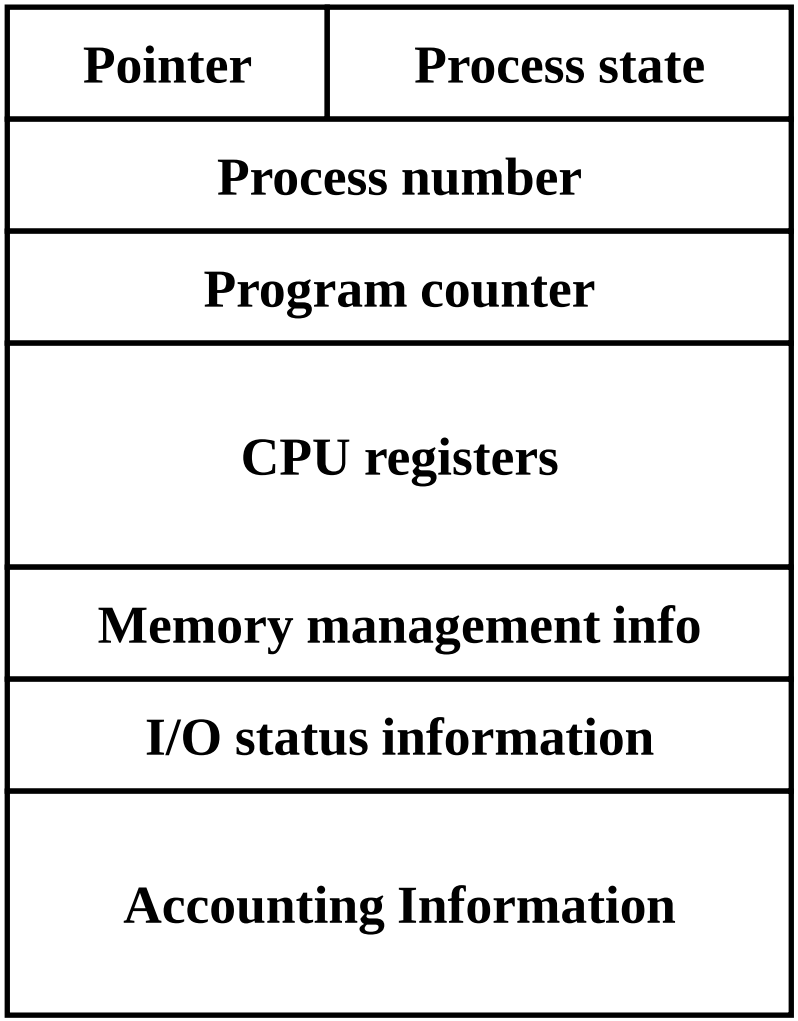
Process State Cont'd

- 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.

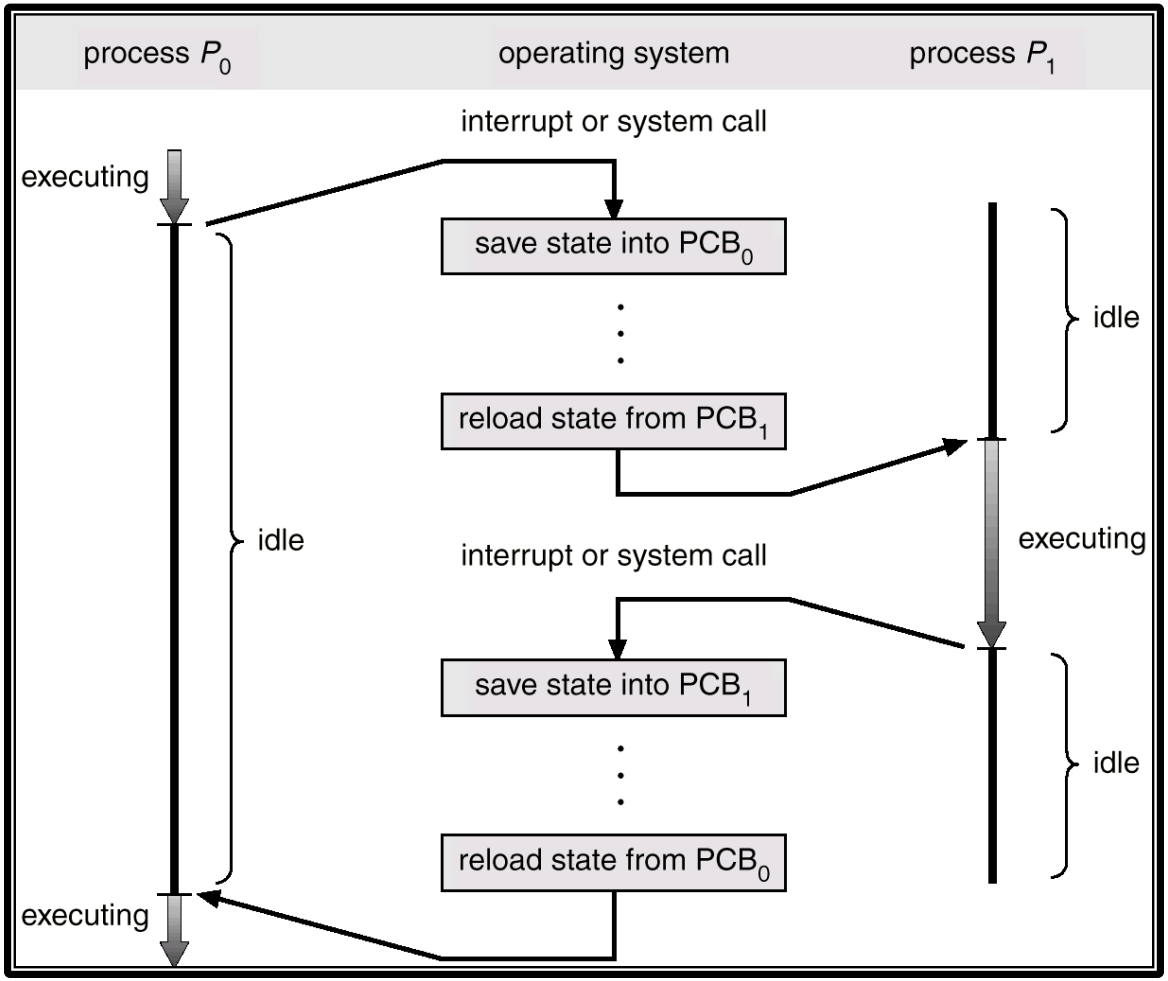
Process Control Block (PCB)

- Information associated with each process.
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

Process Control Block (PCB) Cont'd



CPU Switch From Process to Process



PROCESS SCHEDULING

Process Scheduling

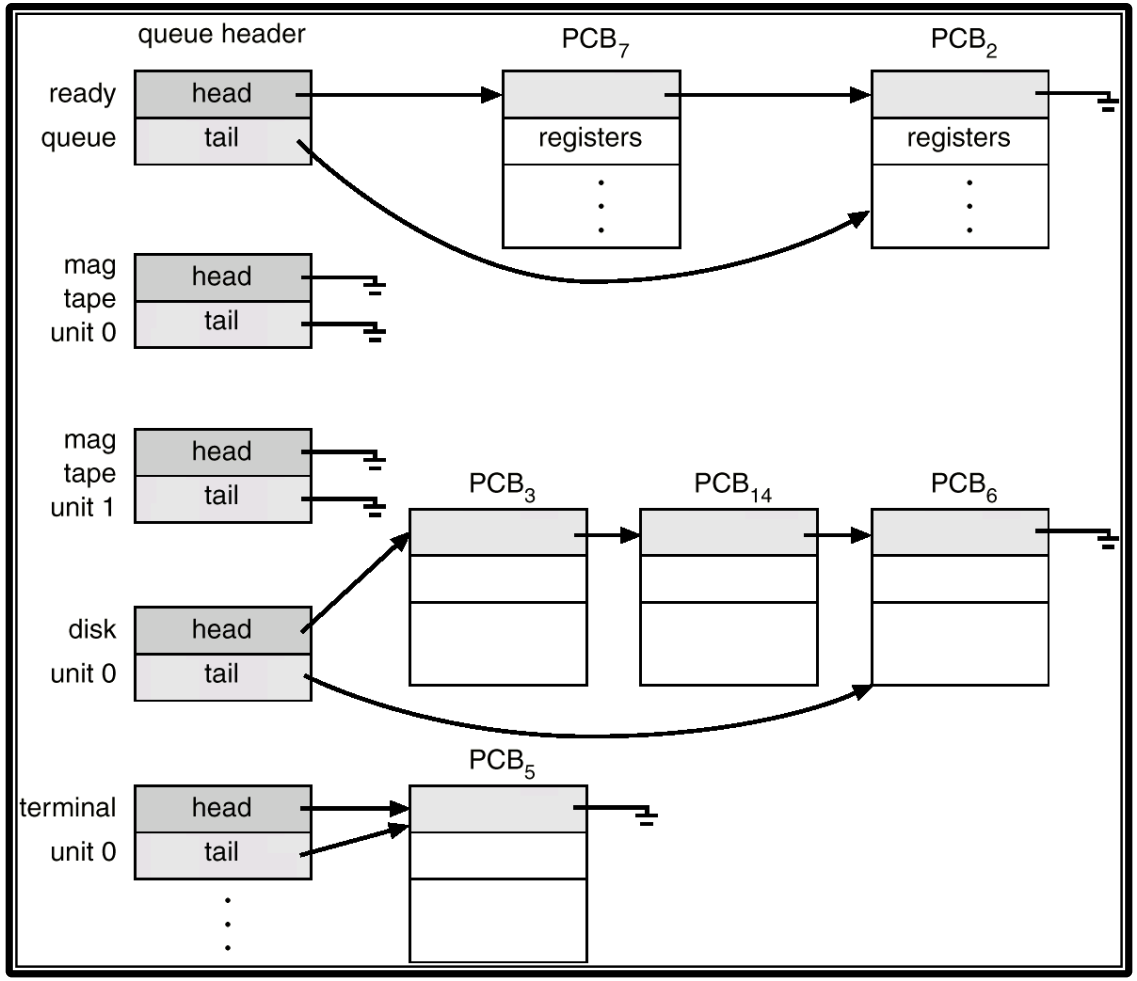
- Multi-Programming systems
  - Some processes executing at all times
  - Maximize CPU utilization
- Time-Sharing systems
  - Switch the CPU among processes frequently
  - Users can interact with a program while it’s executing
  - Virtually run processes at the same time

Process Scheduling Queues

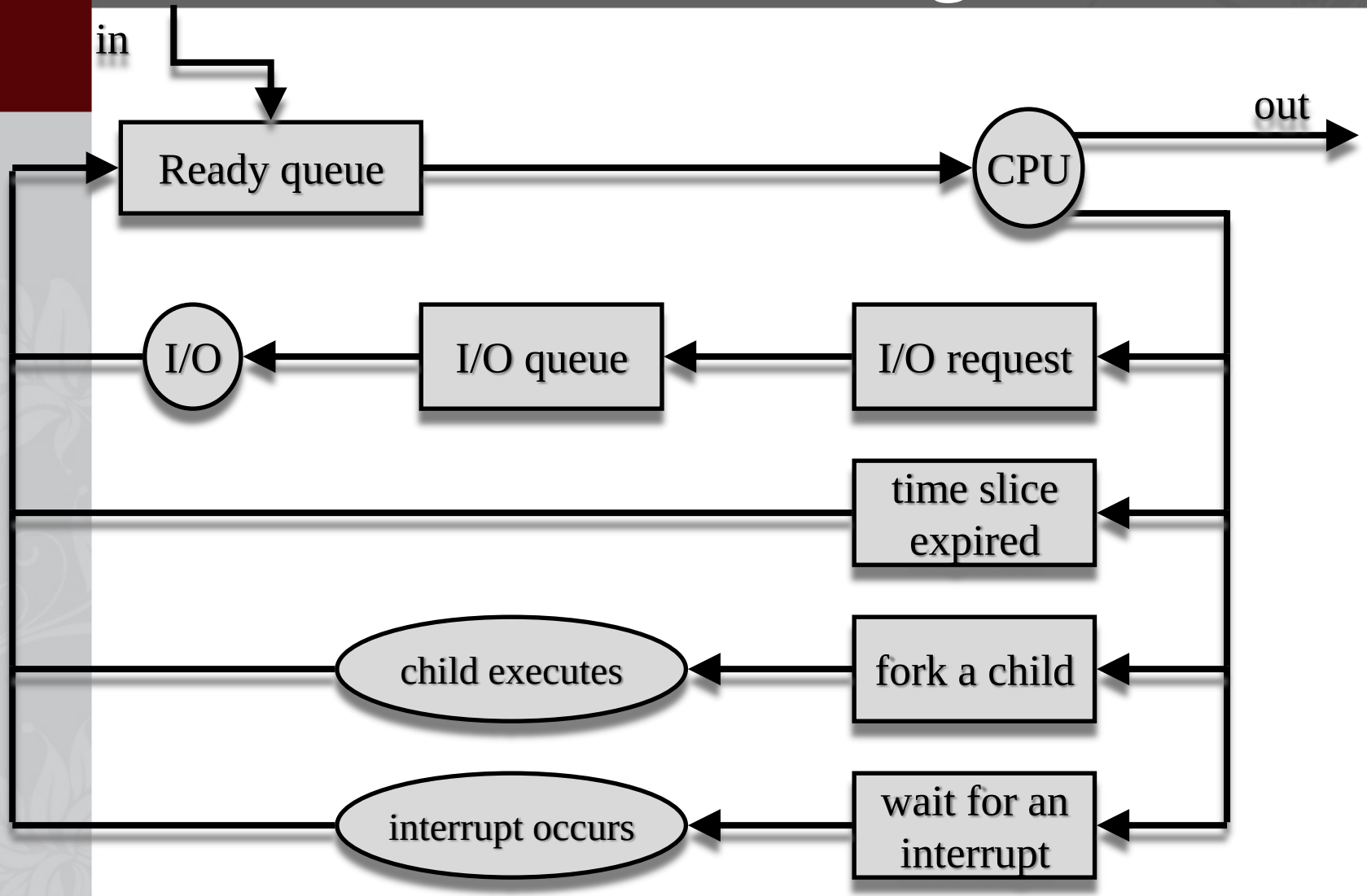
- Job queue
  - set of all processes in the system.
- Ready queue
  - set of all processes residing in main memory which is ready and waiting to execute.
- Device queues
  - set of processes waiting for an I/O device.

* Process migration between the various queues

Ready Queue And Various I/O Device Queues



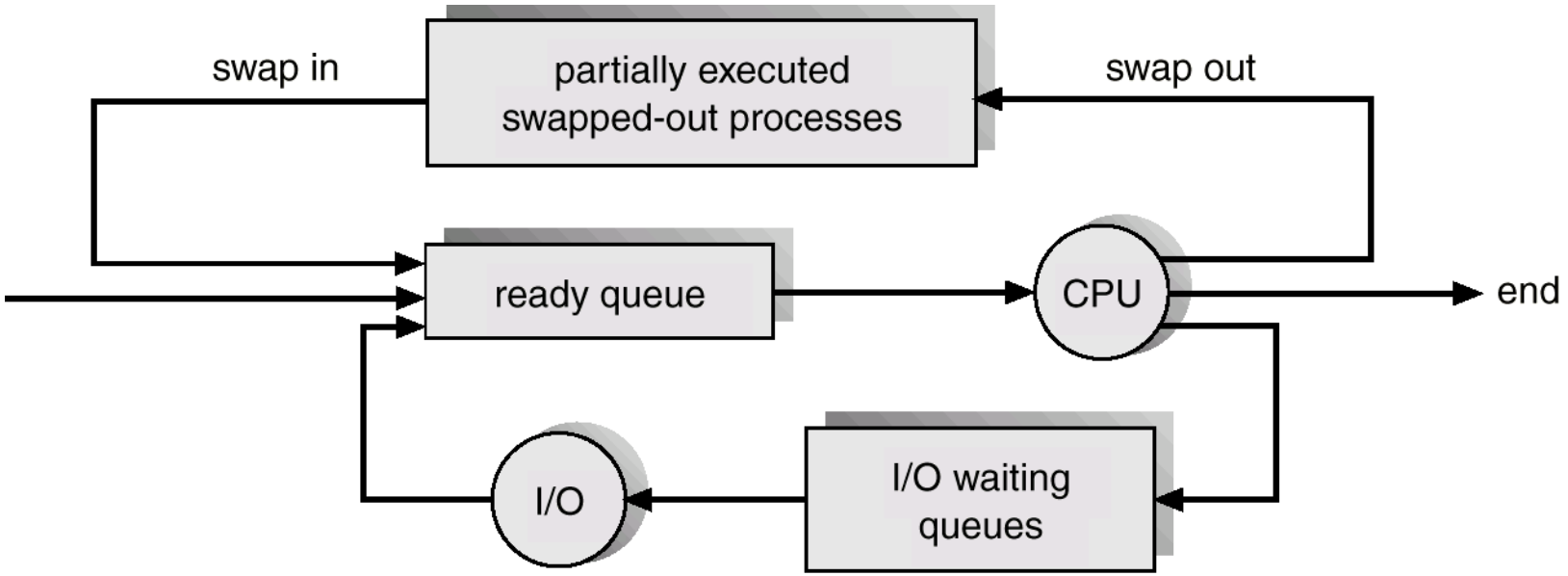
Representation of Process Scheduling



Schedulers

- Long-term scheduler (or job scheduler)
  - Selects which processes should be brought into the ready queue.
  - Executes infrequently
  - May be absent in some O/S
- Short-term scheduler (or CPU scheduler)
  - Selects which process should be executed next and allocates CPU.
  - Executes frequently

Addition of Medium Term Scheduling



Schedulers Cont'd

- 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.
- Proper System performance
  - Mix of CPU & I/O bound processes
- Improper system performance
  - All processes are I/O bound
  - All processes are CPU bound

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.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

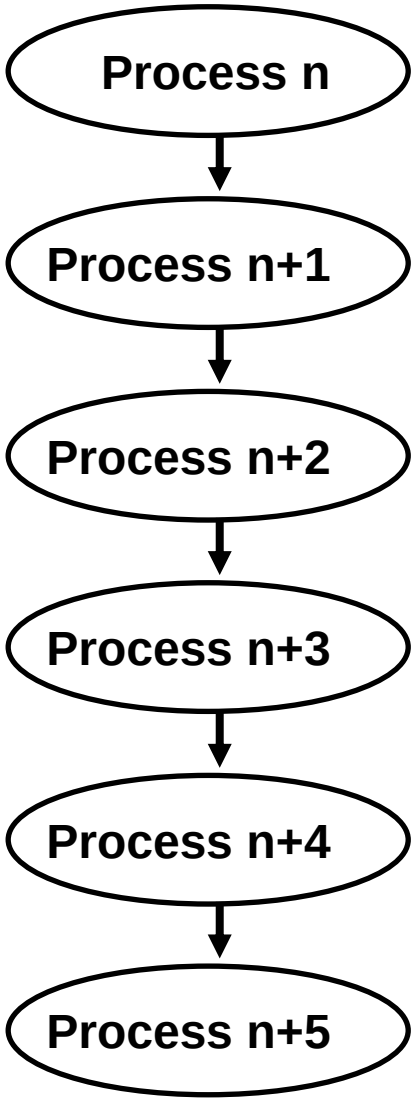
OPERATIONS ON PROCESSES

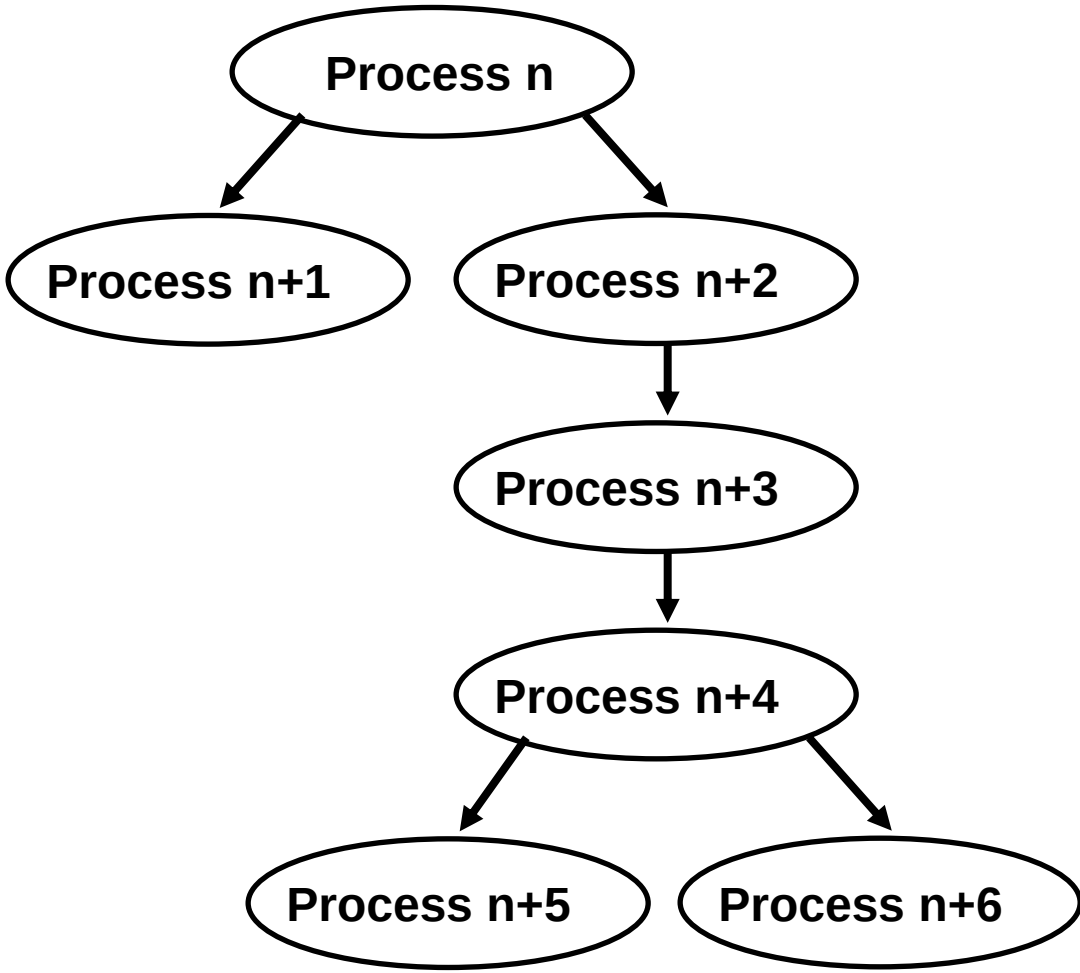
Operations on Processes

- Process Creation
- Process Termination

* Process resources
  - CPU time
  - Memory
  - Files
  - I/O devices

Process Creation





Process Creation Cont'd

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation Cont'd

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program.

Process Termination

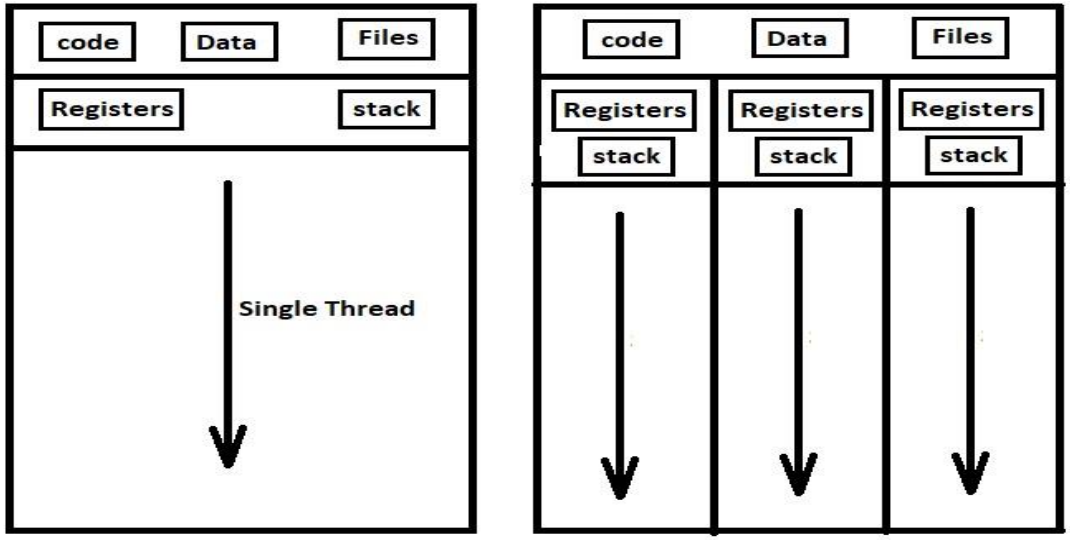
- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent.
  - Process' resources are de-allocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Threads

- Many modern operating systems now provide features for a process to contain multiple threads of control instead of a single Thread
- A thread, sometimes called a lightweight process (LWP)

Threads

- It shares with other threads belonging to the same process its **code section**, **data section**, and other operating-system resources, such as **open files**



Threads

- Benefits:

1. Responsiveness

2. Resource sharing

3. Economy

4. Utilization of multiprocessor architectures

Multi Threading Models

- Many-to-One Model:
  - Maps many user-level threads to one kernel thread. Thread
  - Management is done in user space, so it is efficient
  - The entire process will block if a thread makes a blocking system call
  - Multiple threads are unable to run in parallel on multiprocessors

Multi Threading Models

- One-to-one Model:
  - Maps each user thread to a kernel thread.
  - It provides more concurrency than the many-to-one model.
  -
  - Another thread can run when a thread makes a blocking system call.
  -
  - It also allows multiple threads to run in parallel on multiprocessors.
  - The only drawback is the overhead of creating kernel threads can burden the performance of an application.

Multi Threading Models

- Many-to-Many Model:
  - Multiplexes many user-level threads to a smaller or equal number of kernel threads.
  - Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
  - Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Threading Issues

- The fork and exec System Calls
  - Does the new process duplicate all threads or is the new process single-threaded?
- Cancellation
  - Allocating resources to a cancelled thread or cancel a thread while updating data that is shared by another thread
- Signal Handling
  - Handling signals in single-threaded programs is straightforward; signals are always delivered to a process. However, delivering signals is more complicated in multithreaded programs, as a process may have several threads. Where then should a signal be delivered?
- Thread Pools
  - Unlimited threads could exhaust system resources, such as CPU time or memory. One solution to this issue is to use thread pools
- Thread-Specific Data
  - Each thread might need its own copy of certain data in some circumstances.

Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

INTER-PROCESS COMMUNICATION

Inter-process Communication

- Message Passing
- Shared Memory

* Synchronization

Message Passing

- Messages
  - Fixed size
  - Variable size
- Communication
  - Direct:
    - send(P2, message), receive(P1, message)
  - Indirect:
    - send(ID, message), receive(ID, messages)

Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking.

Information Technology Institute

Operating System Fundamentals

Chapter Six

DEADLOCKS

Table of Content

- Introduction
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Recovery from Deadlock

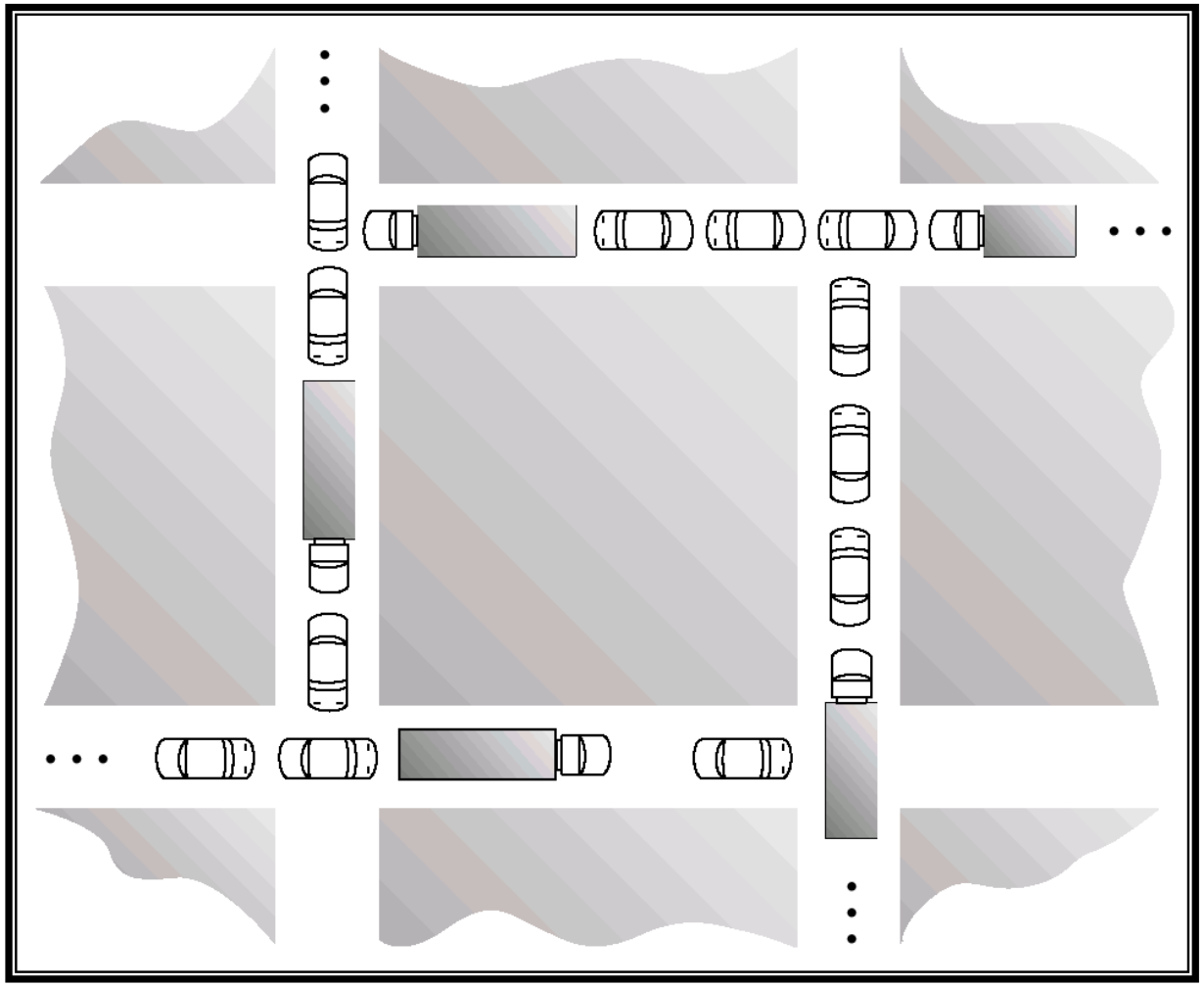
INTRODUCTION

The Deadlock Problem

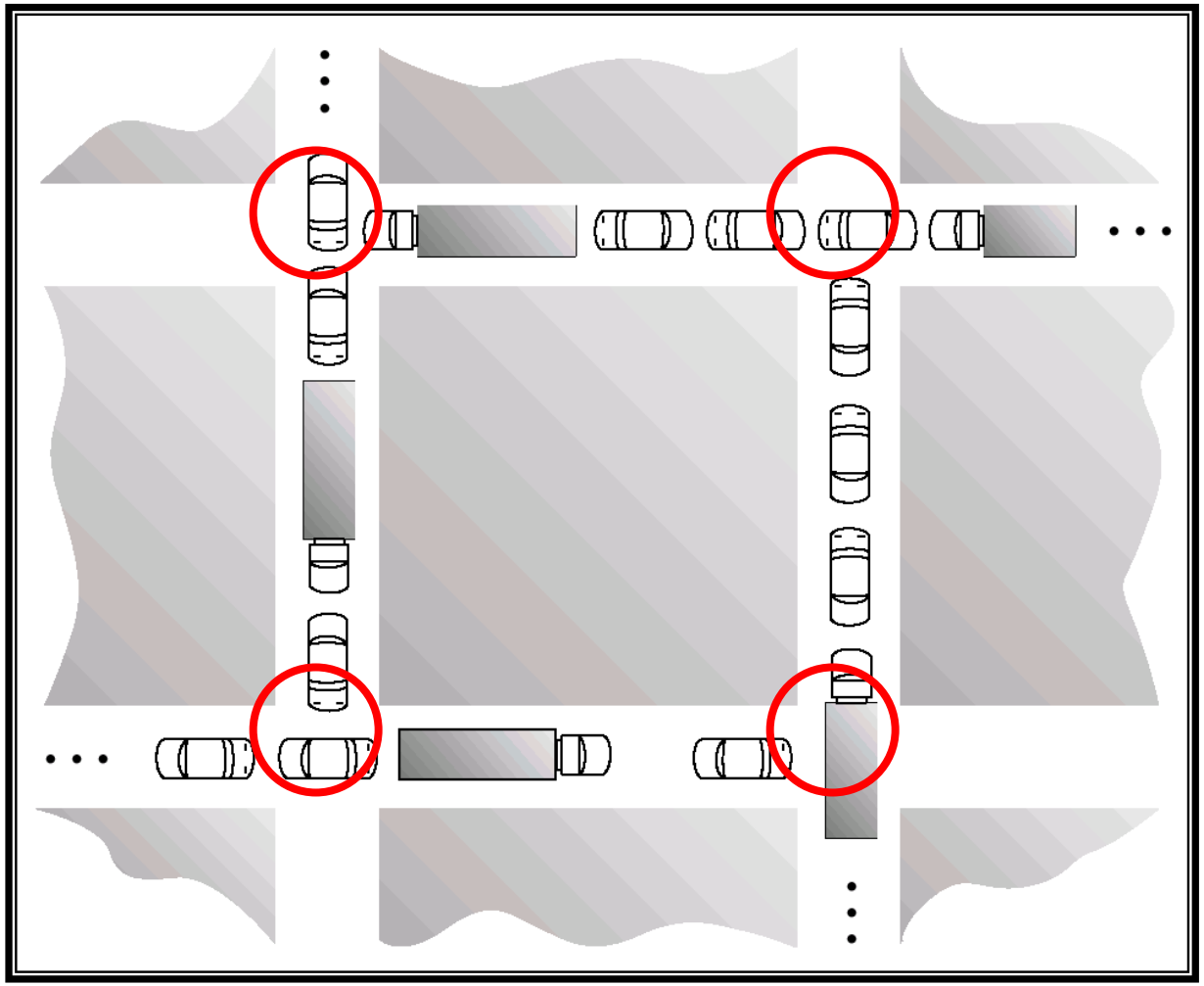
- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  - System has 2 tape drives.
  - P1 and P2 each hold one tape drive and each needs another one.
  - semaphores A and B, initialized to 1

P1	P2
wait (A);	wait(B)
wait (B);	wait(A)

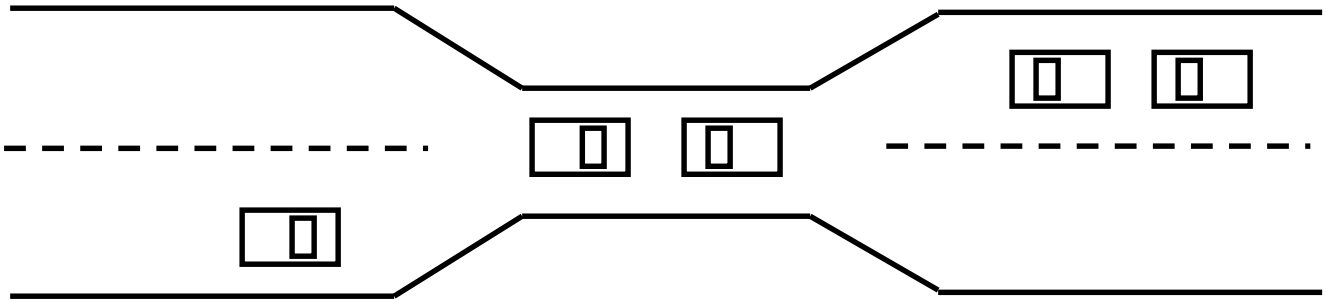
Is this Deadlock?



Yes, How to prevent it?



Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible

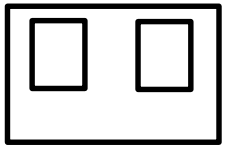
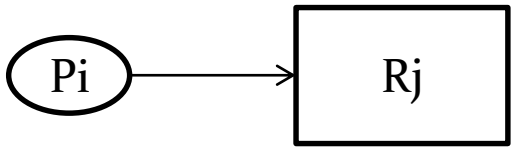
DEADLOCK CHARACTERIZATION

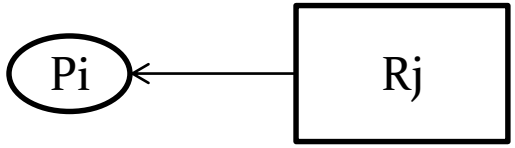
Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously.

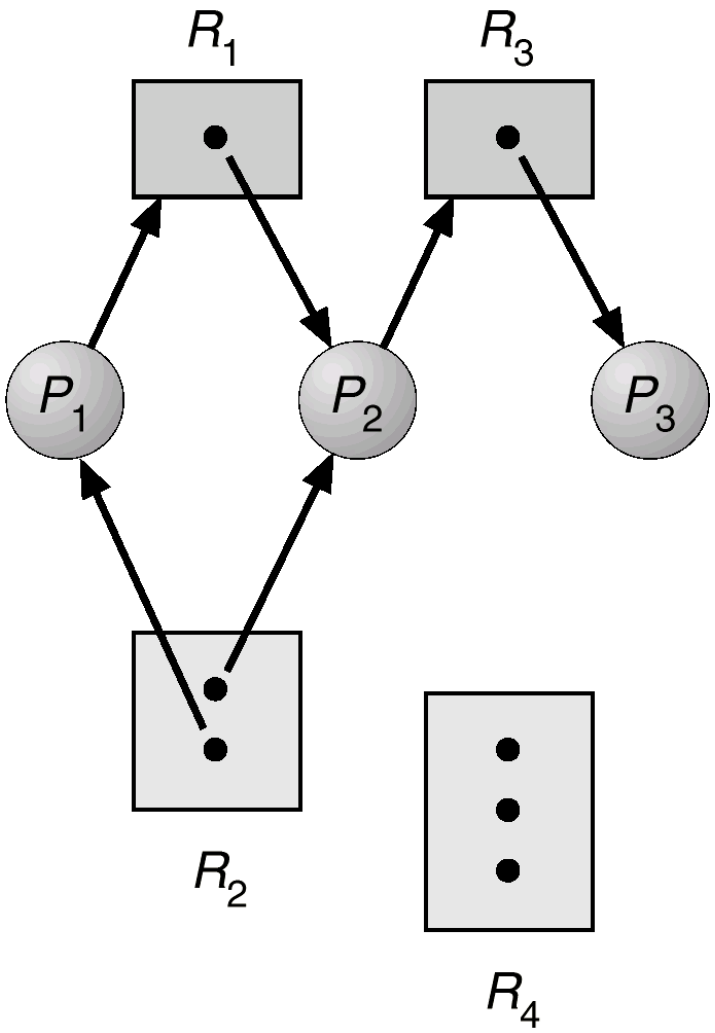
1. Mutual exclusion: only one process at a time can use a resource.
2. Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes.
3. No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. Circular wait: there exists a set {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0

Resource-Allocation Graph

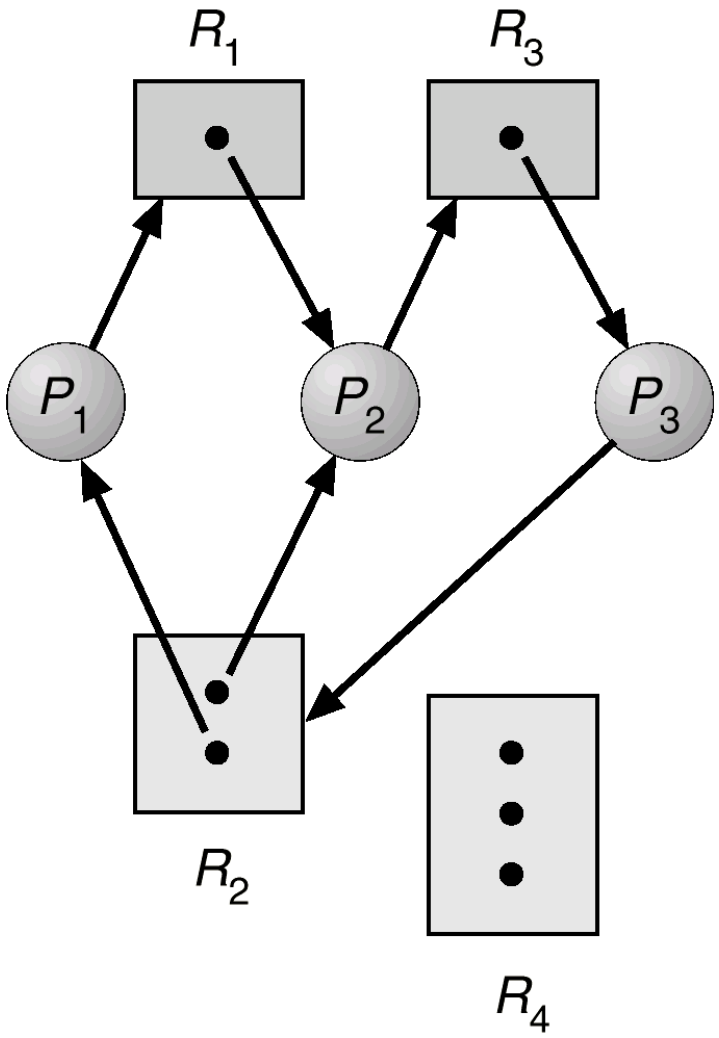
- Process 
- Resource Type with 2 instances 
- *Pi* requests instance of *Rj* 

- *Pi* is holding an instance of *Rj* 

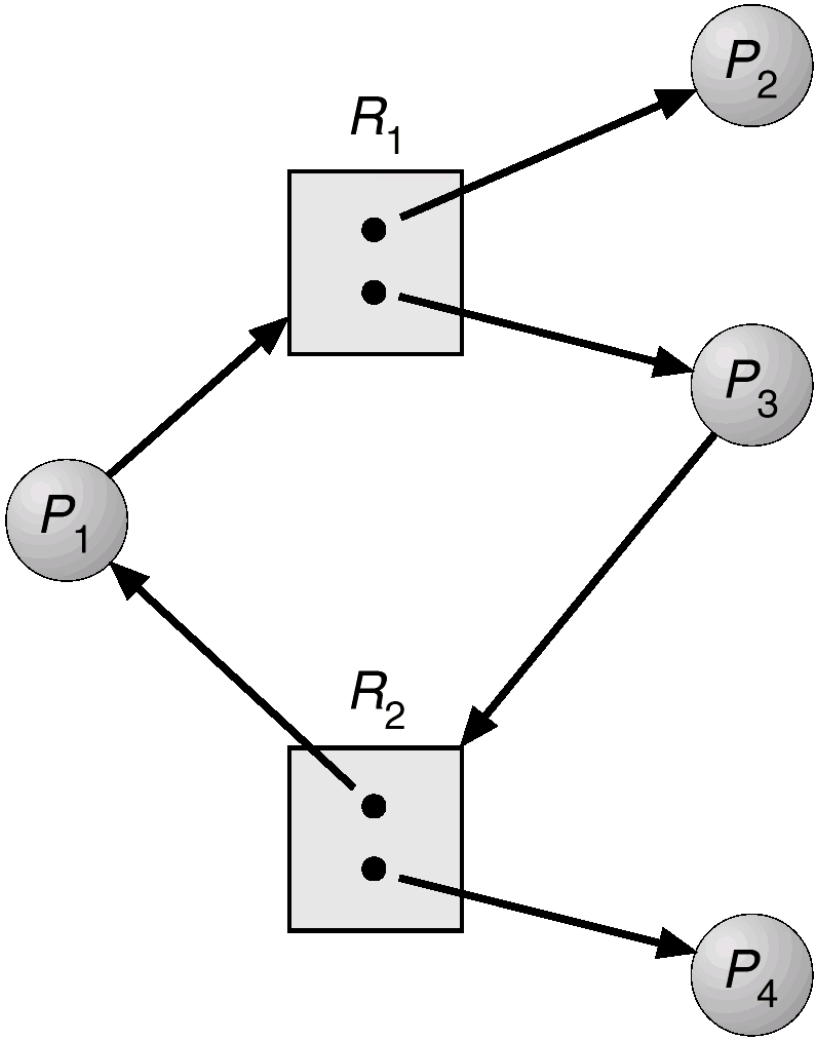
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles no deadlock.
- If graph contains a cycle
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

METHODS FOR HANDLING DEADLOCKS

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems.

DEADLOCK PREVENTION

Deadlock Prevention

- Mutual Exclusion
  - Not required for sharable resources; must hold for non-sharable resources.
- Hold and Wait
  - must guarantee that whenever a process requests a resource, it does not hold any other resources.
    - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
    - Low resource utilization; starvation possible.

Deadlock Prevention Cont'd

- No Preemption
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - Preempted resources are added to the list of resources for which the process is waiting.
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Circular Wait
  - Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

RECOVERY FROM DEADLOCK

Recovery from Deadlock

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim
  - minimize cost.
- Rollback
  - return to some safe state, restart process for that state.
- Starvation
  - same process may always be picked as victim, include number of rollback in cost factor.

Information Technology Institute

Operating System Fundamentals

Chapter Seven

Memory Management

Table of Content

- Logical versus Physical Address Space.
- Swapping.
- Contiguous Allocation.
- Paging.
- Segmentation.
- Segmentation with Paging.

LOGICAL VS PHYSICAL ADDRESS SPACE

Logical vs. Physical Address Space

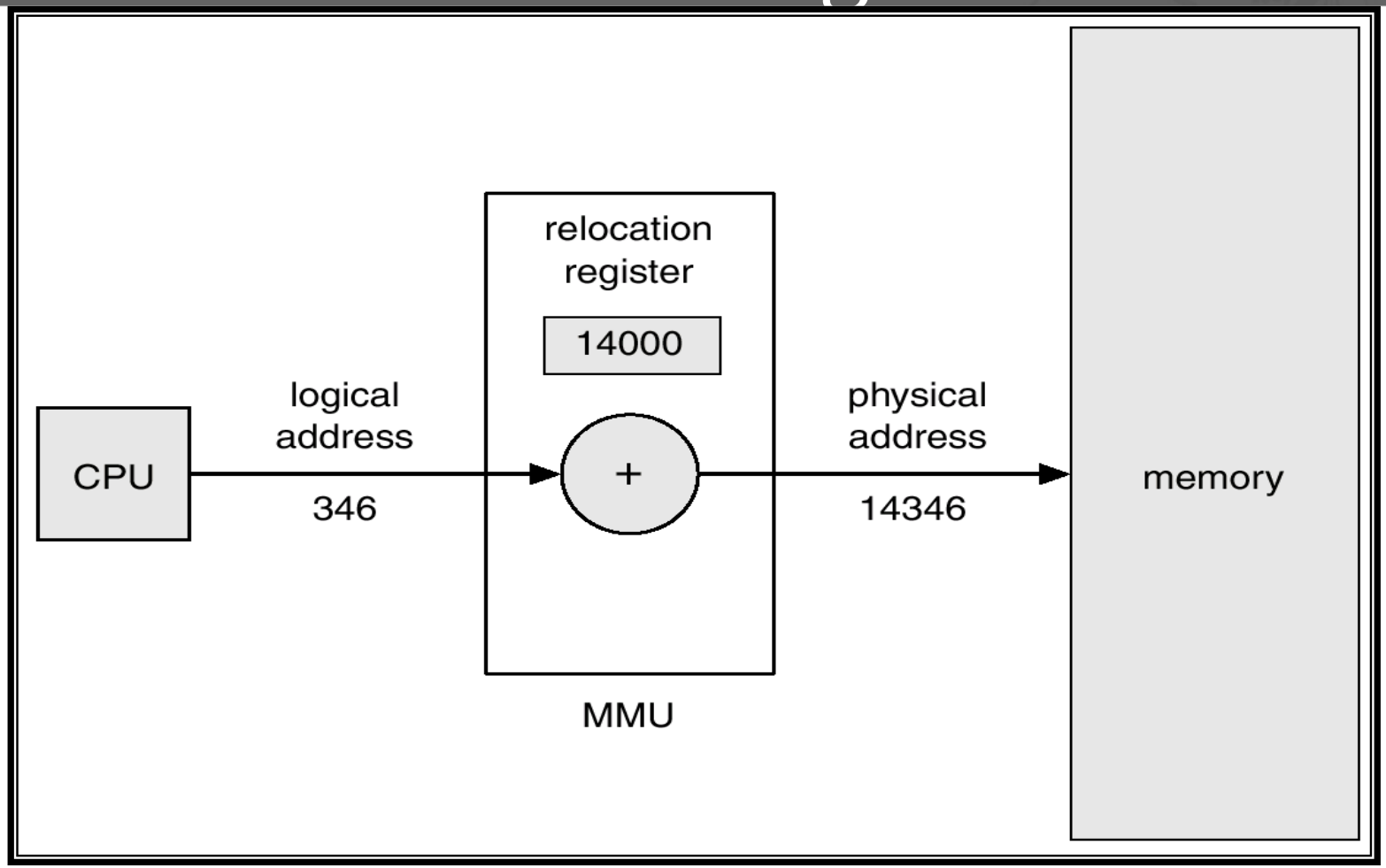
- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.
  - *Logical address* – generated by the CPU; also referred to as *virtual address*.
  - *Physical address* – address seen by the memory unit.

- 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.

Memory-Management Unit (MMU)

- Hardware device that maps logical (virtual) to physical address.
- In MMU scheme, the value in the relocation register (base register) is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Dynamic relocation using a relocation register

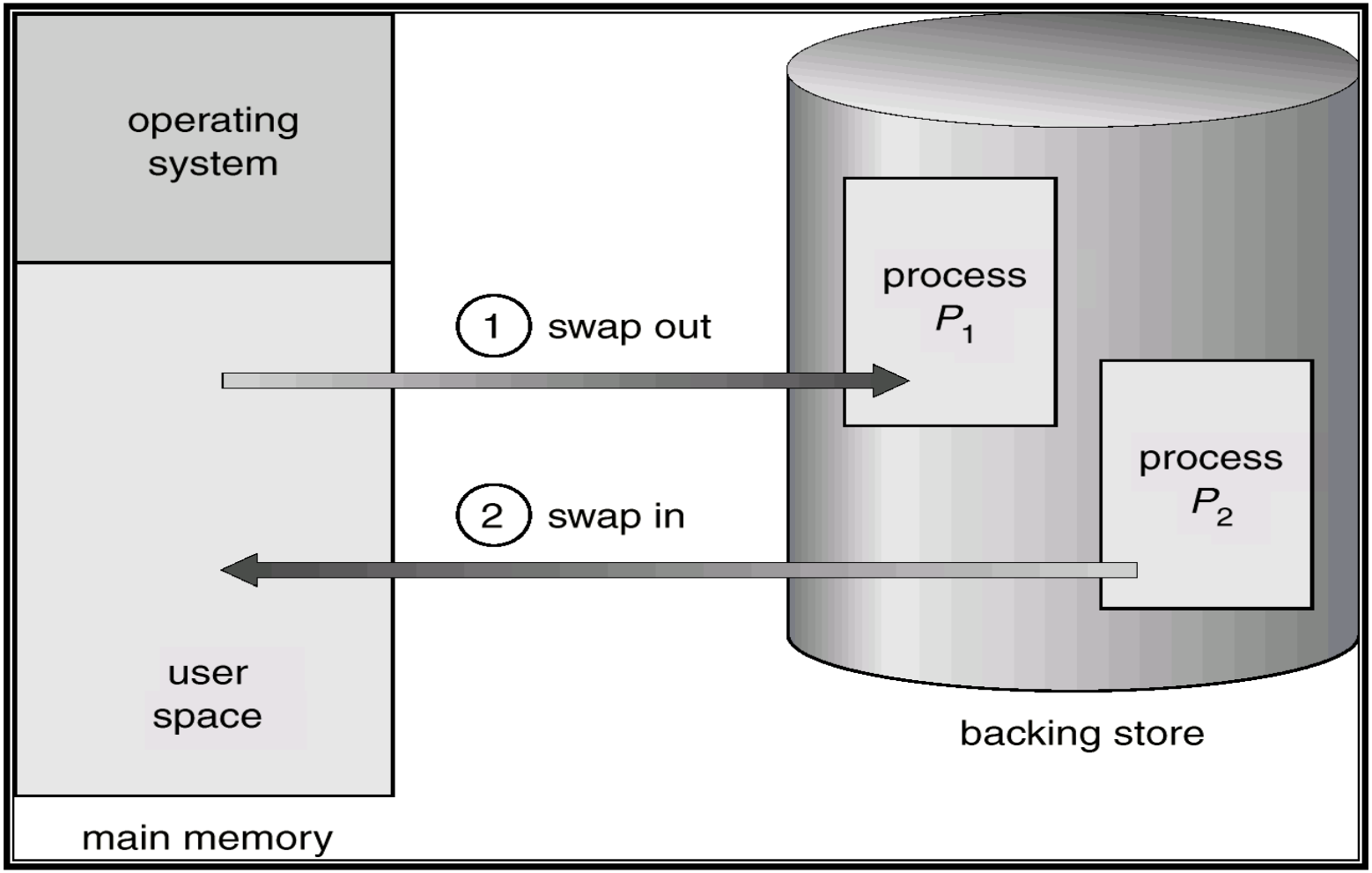


SWAPPING

Swapping

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- *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.
- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.

Schematic View of Swapping

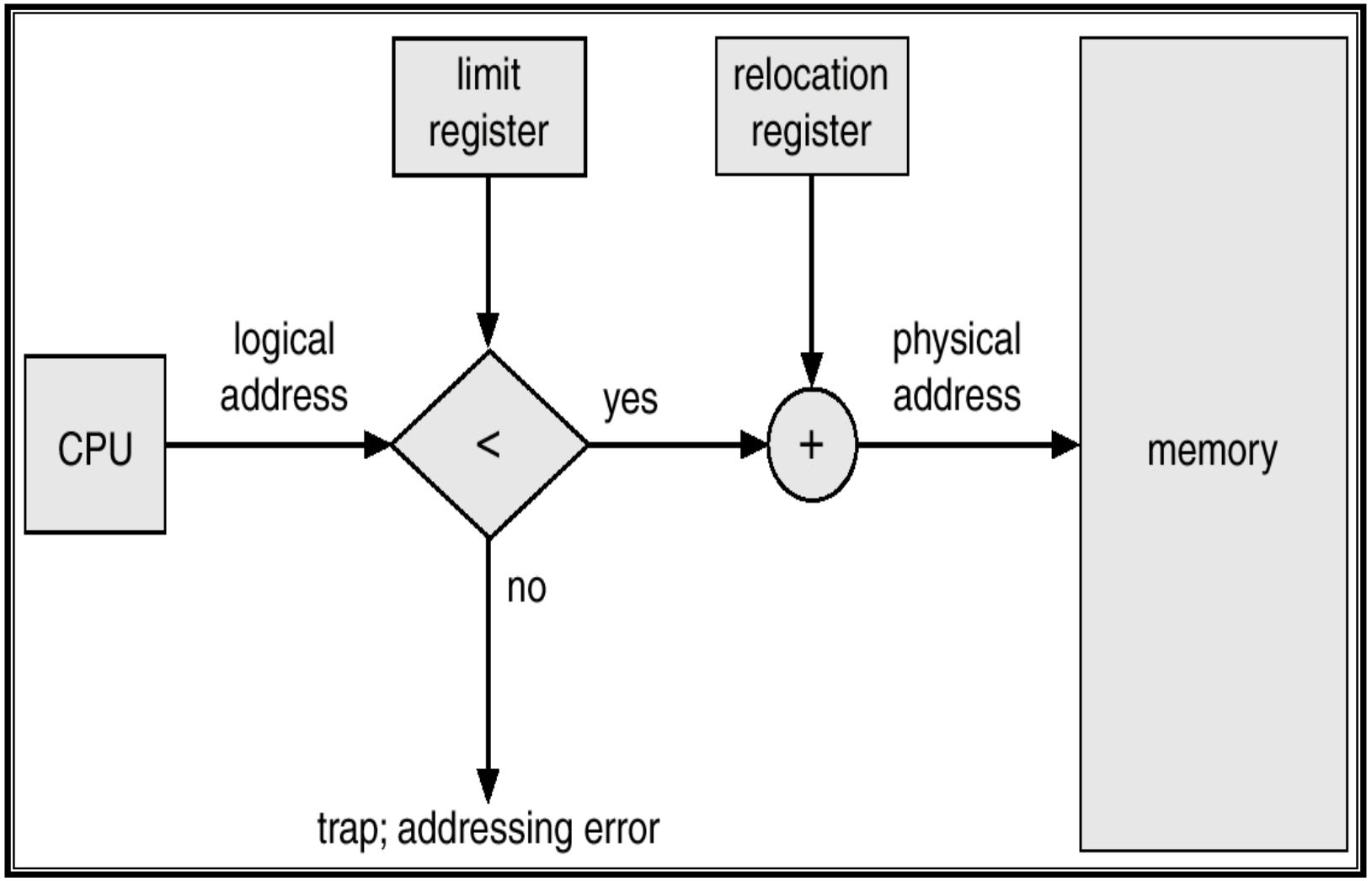


CONTIGUOUS ALLOCATION

Contiguous Allocation

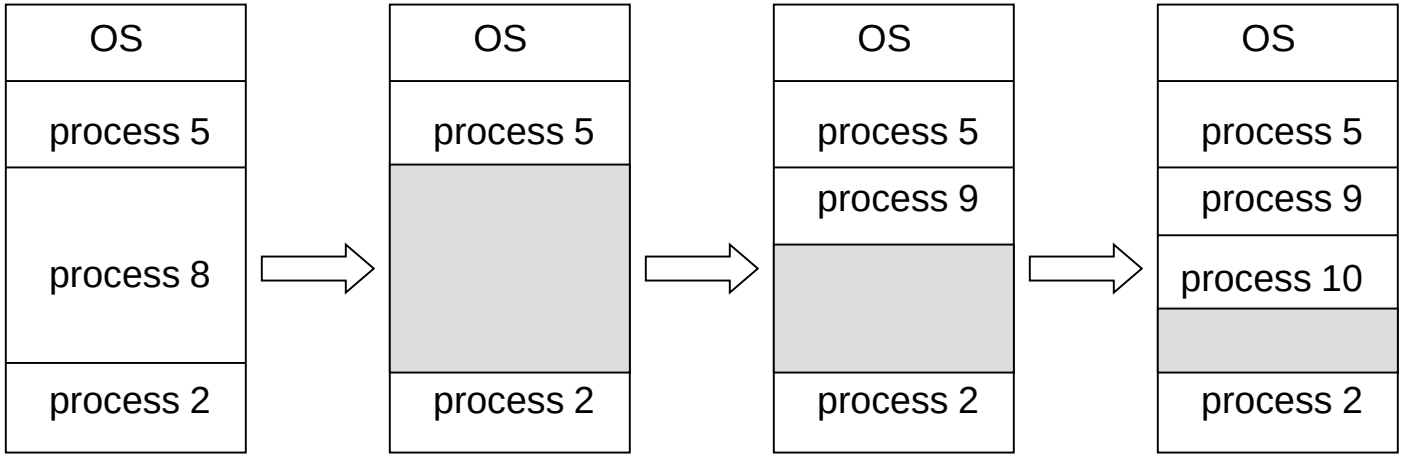
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses.

Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - *Hole* – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about:
    a) ***allocated partitions*** b) ***free partitions*** (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size n from a list of free holes.

- **First-fit**: Allocate the *first* hole that is big enough.
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Reduce external fragmentation by compaction

PAGING

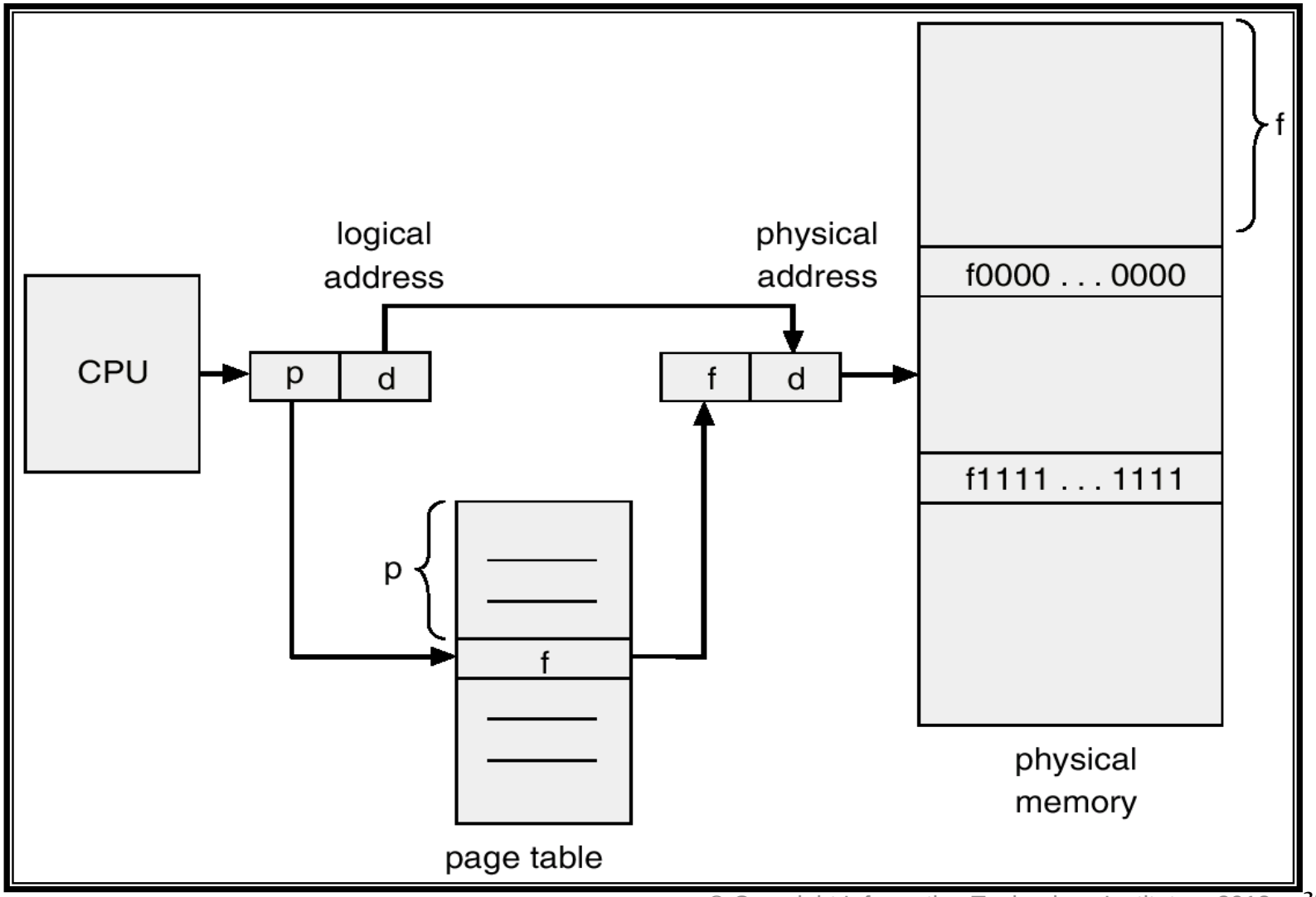
Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames.
- To run a program of size *n* pages, need to find *n* free frames and load program.
- Set up a page table to translate logical to physical addresses.
- *Internal fragmentation.*

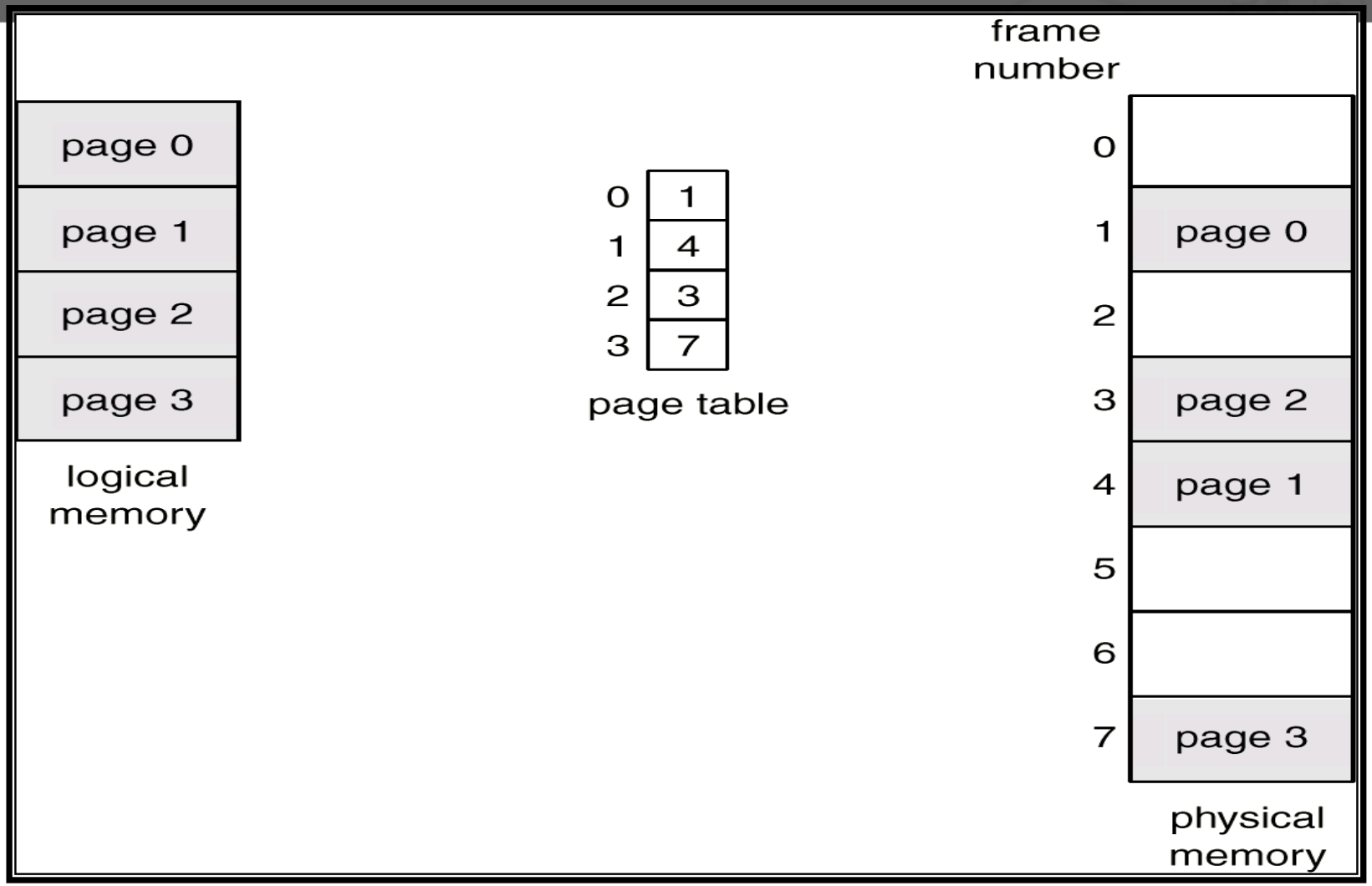
Address Translation Scheme

- Address generated by CPU is divided into:
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.

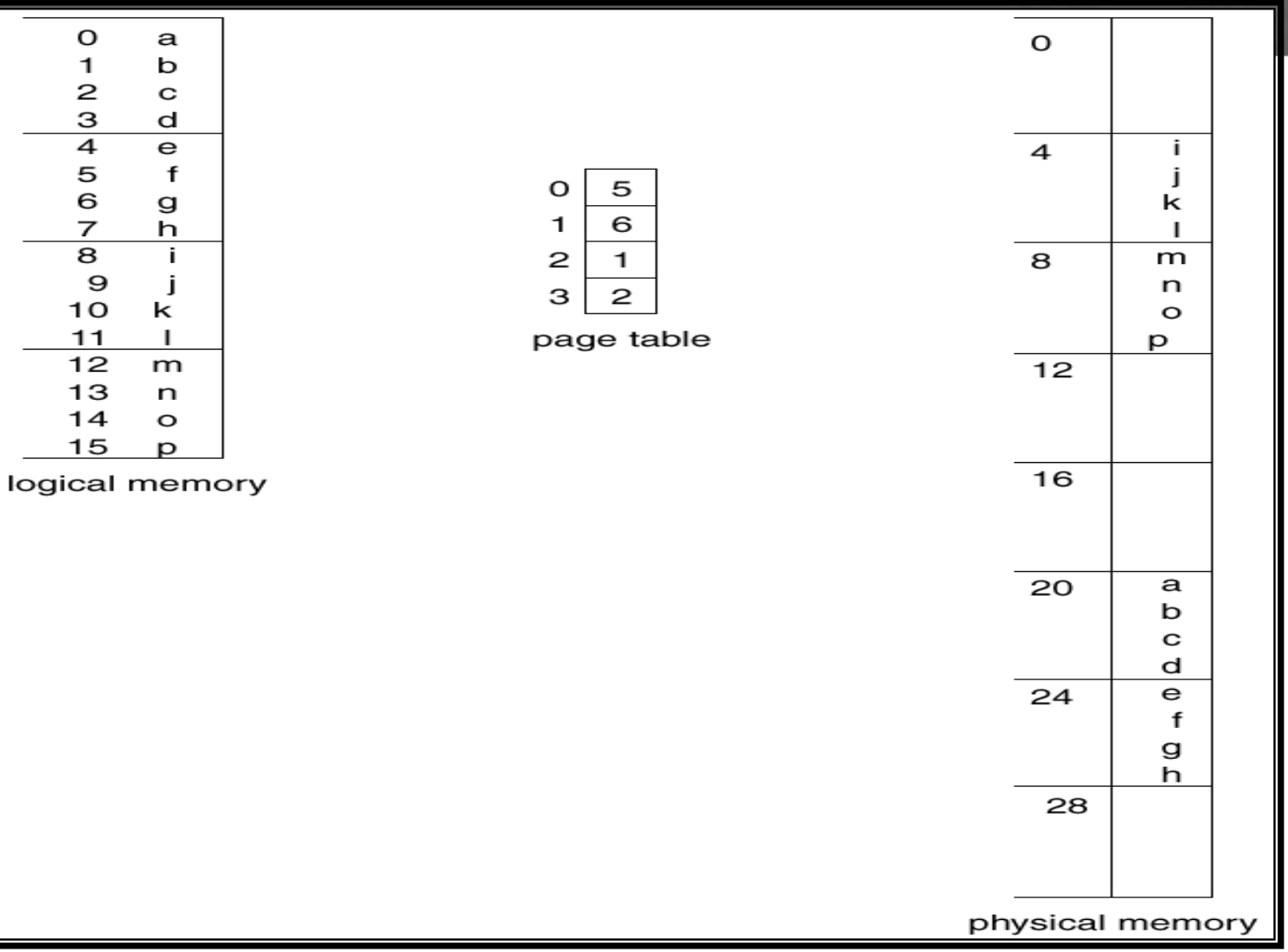
Address Translation Architecture



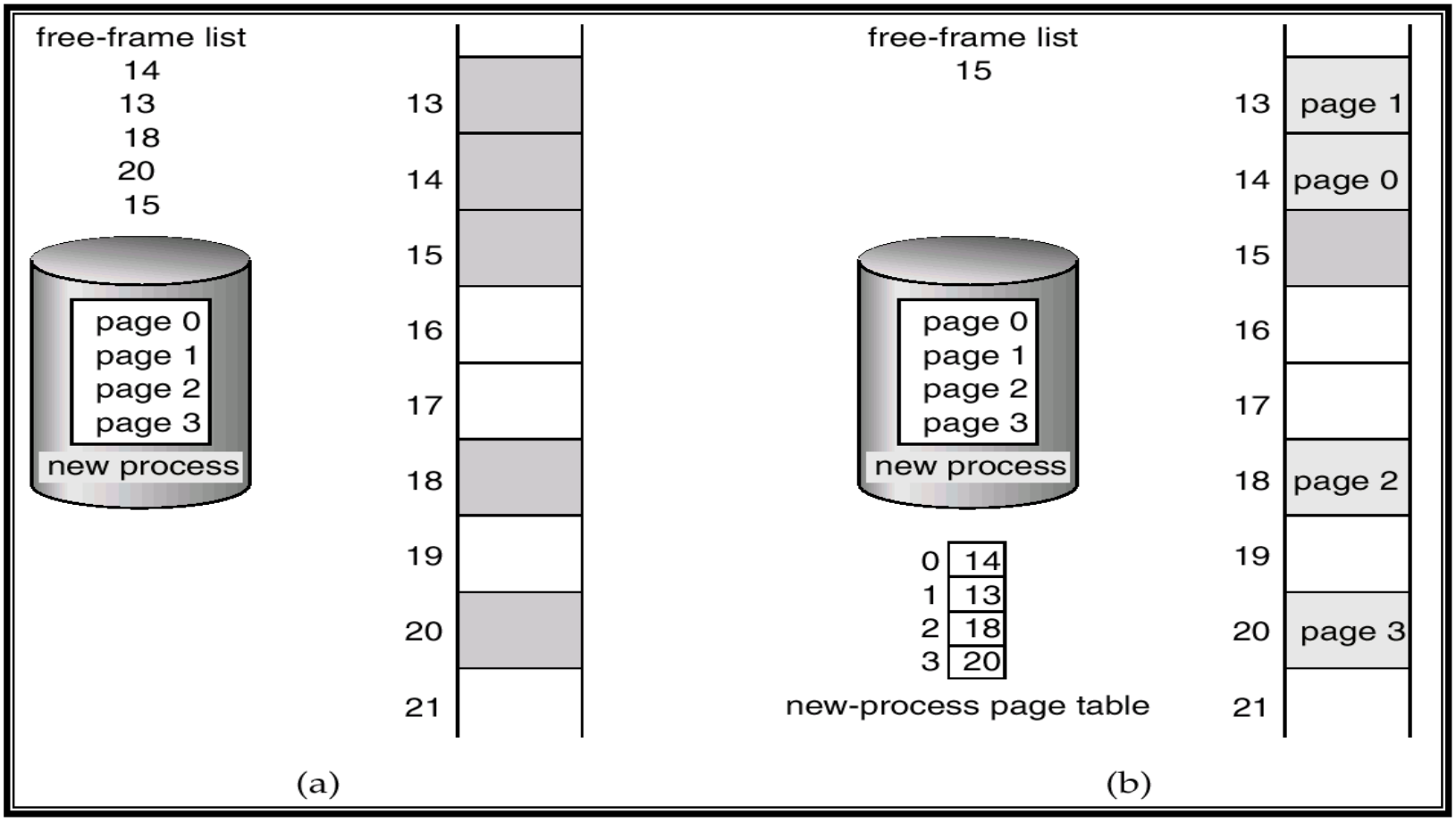
Paging Example



Paging Example



Free Frames



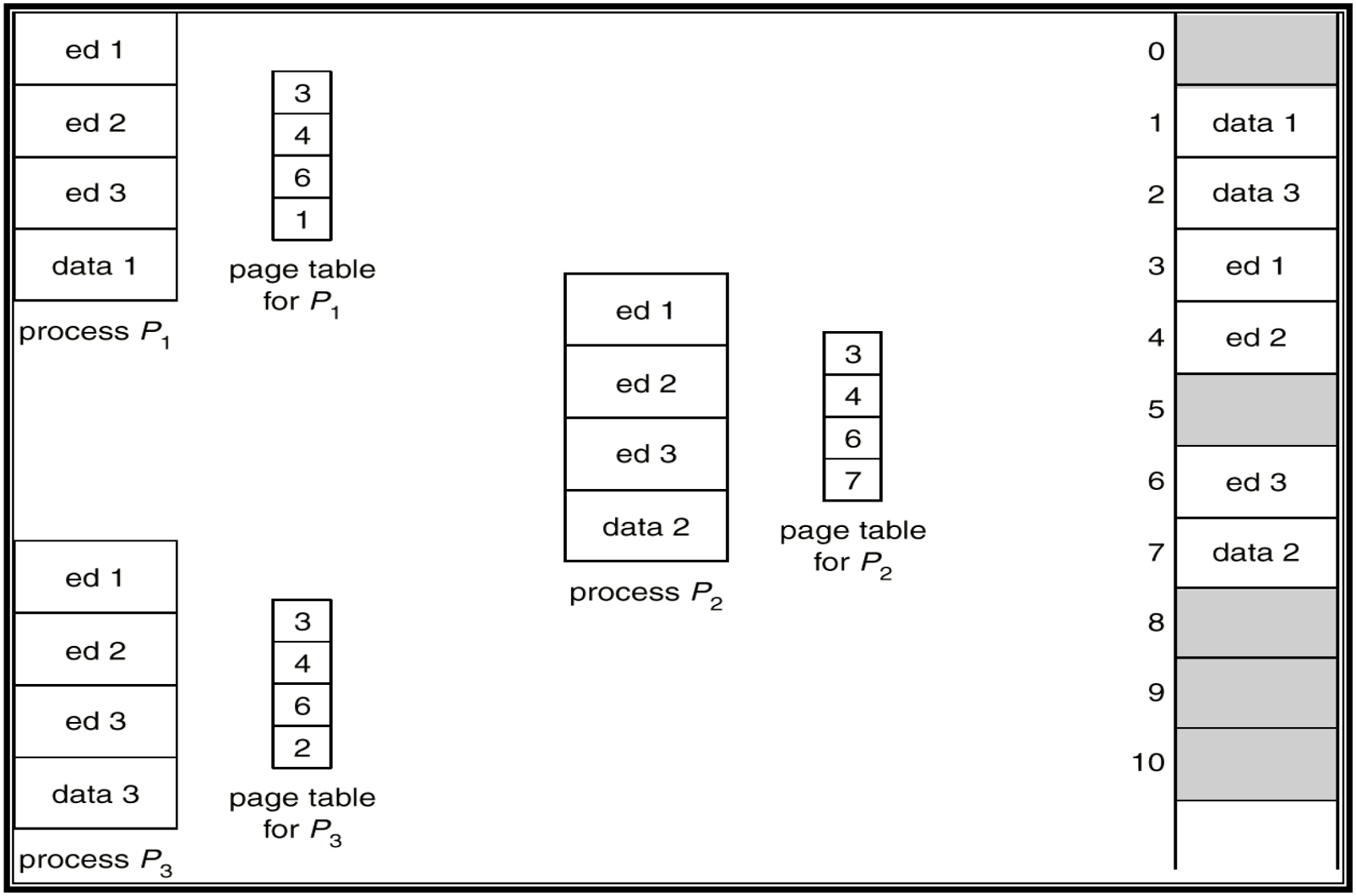
Before allocation

After allocation

Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes.

Shared Pages Example



SEGMENTATION

Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:

  main program,

  procedure,

  function,

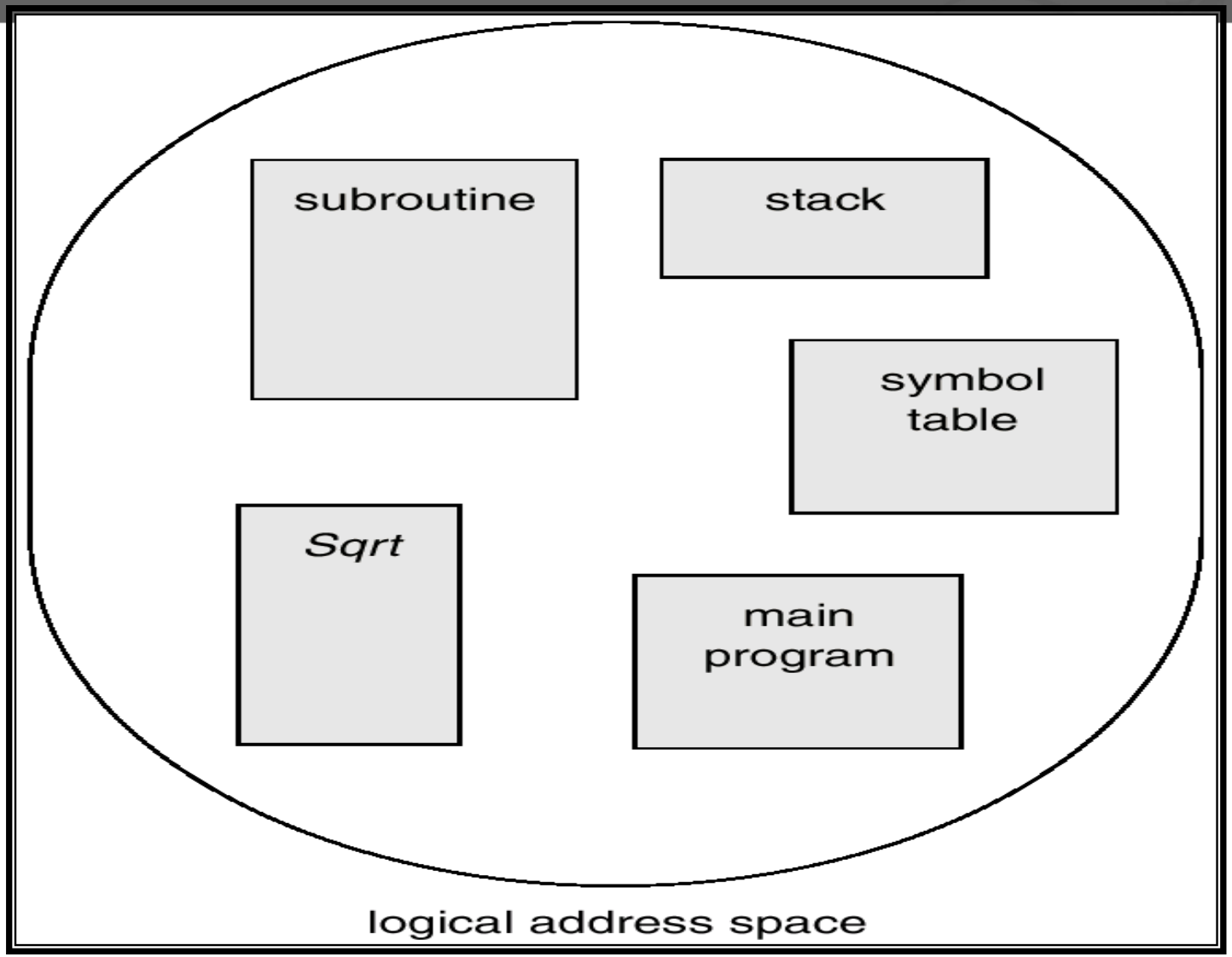
  method,

  common block,

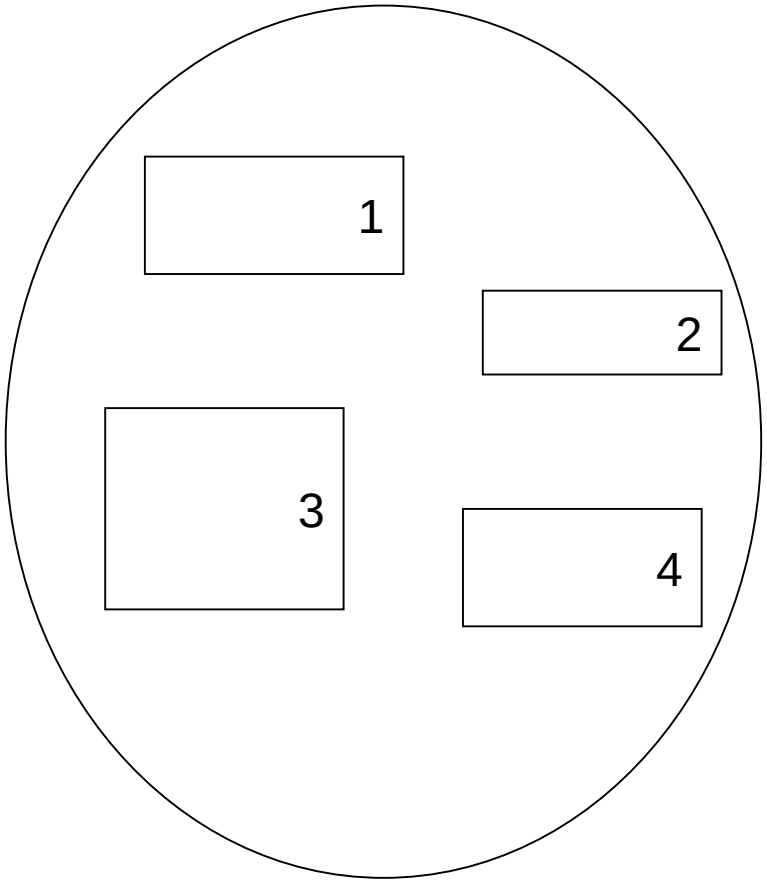
  stack,

  symbol table, arrays

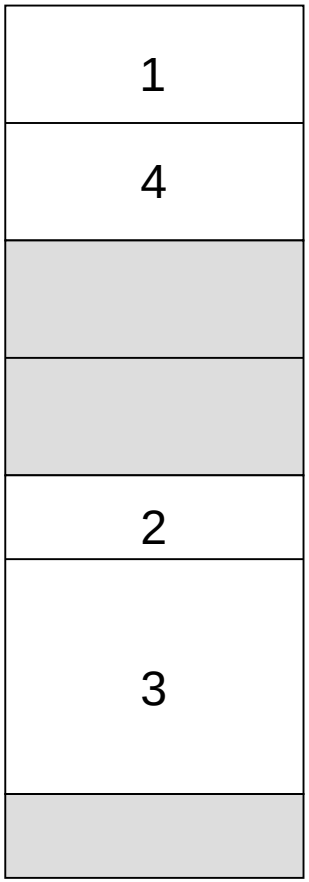
User's View of a Program



Logical View of Segmentation



user space



physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

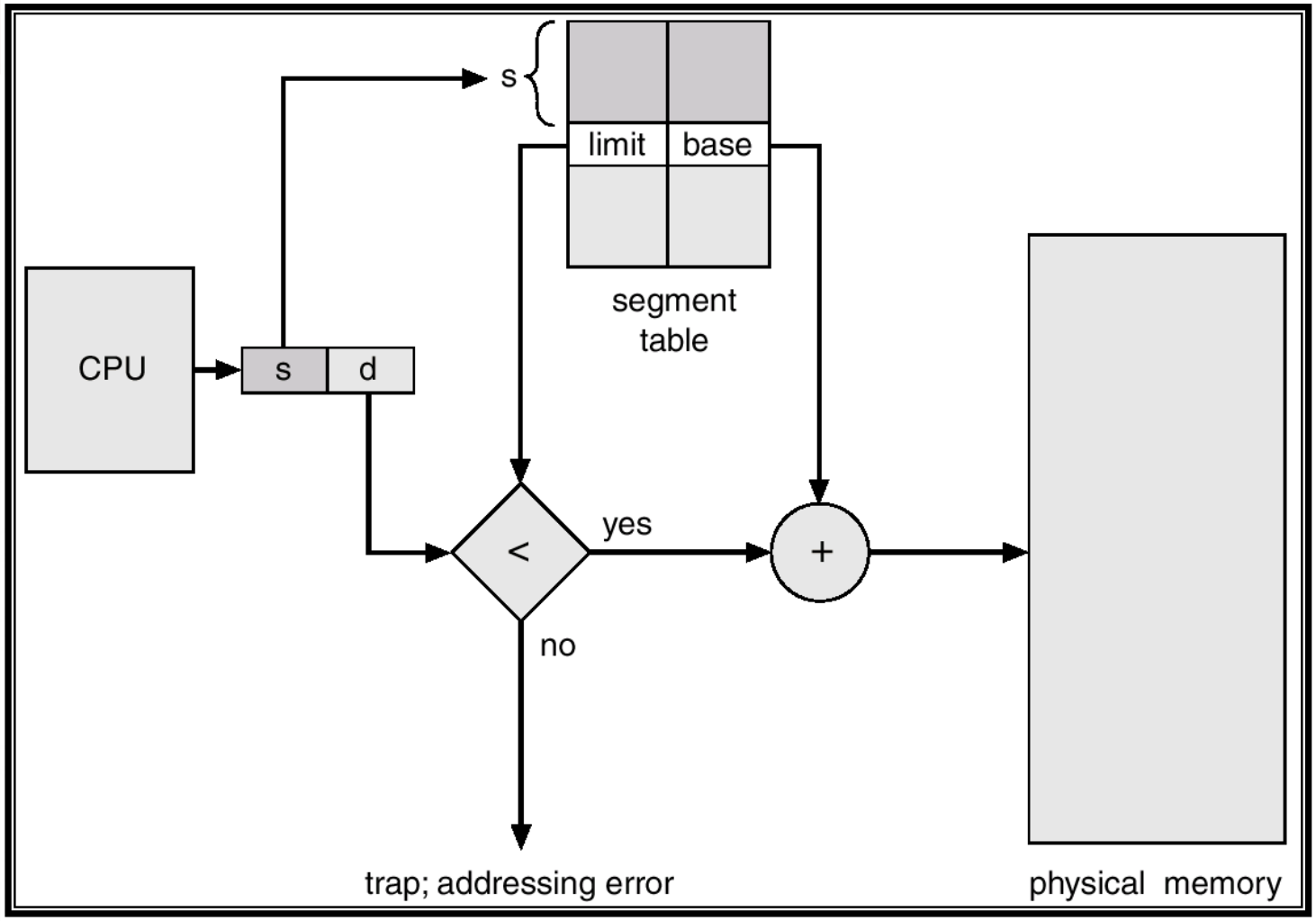
  <segment-number, offset>,

- *Segment table* – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - *limit* – specifies the length of the segment.

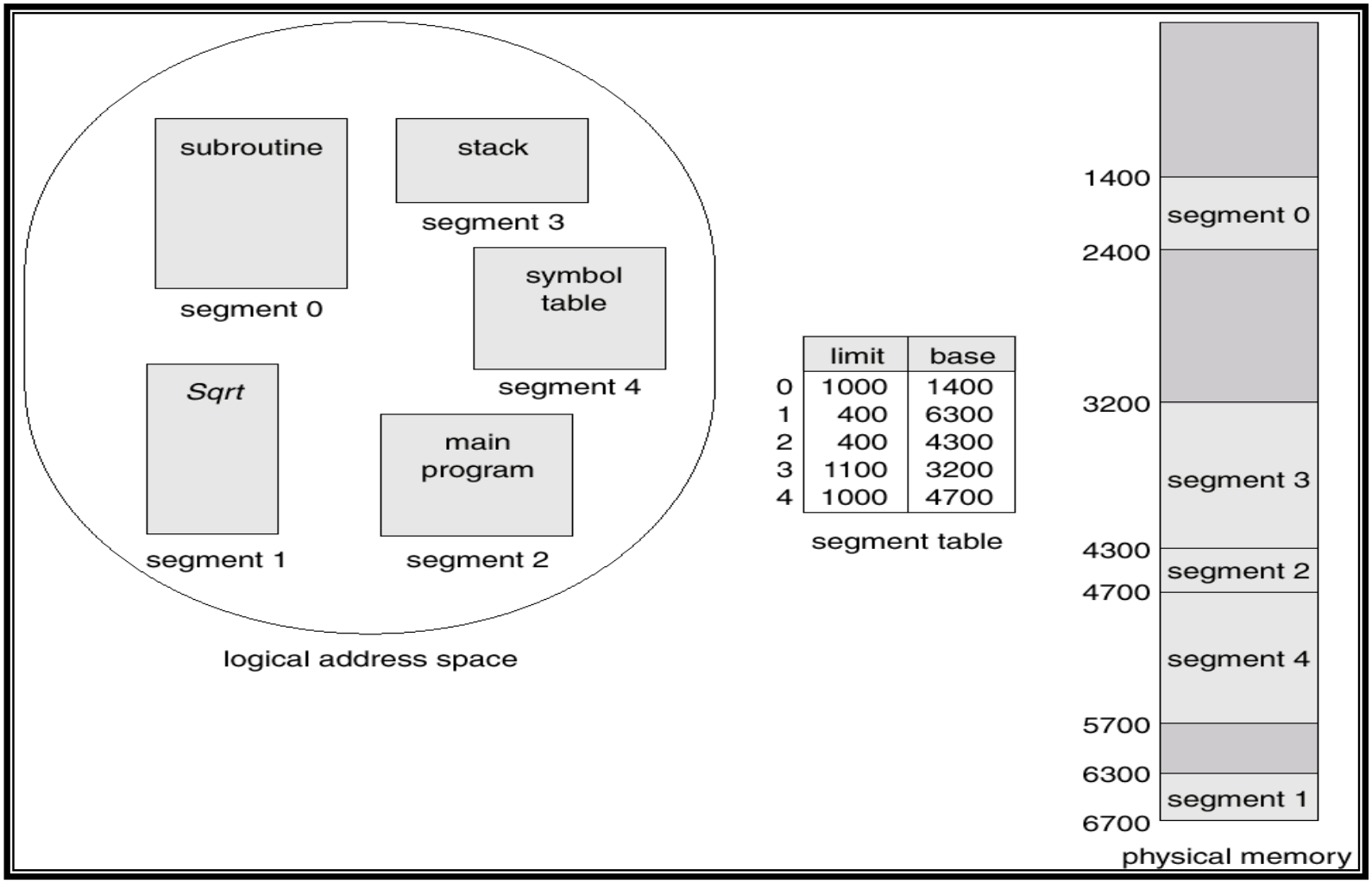
Segmentation Architecture (Cont.)

- Relocation.
  - dynamic
  - by segment table
- Sharing.
  - shared segments
  - same segment number
- Allocation.
  - first fit/best fit
  - external fragmentation

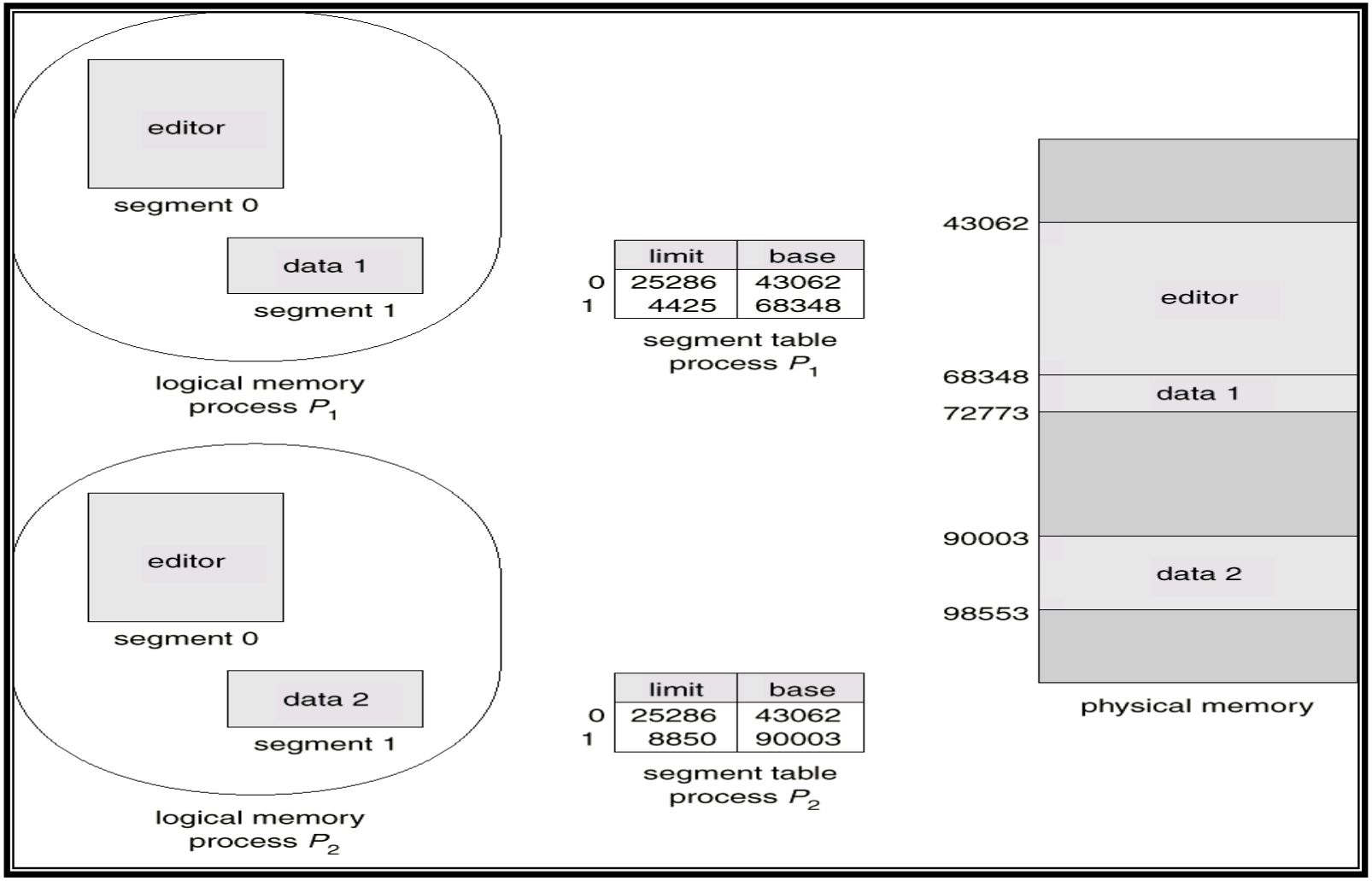
Segmentation Hardware



Example of Segmentation



Sharing of Segments

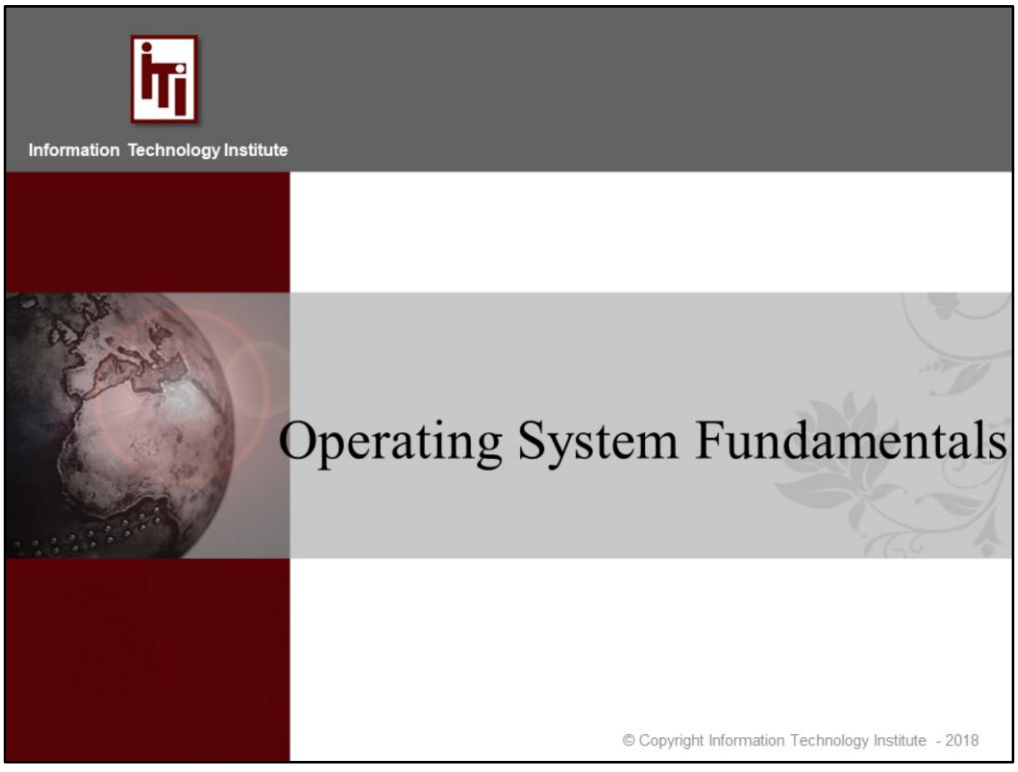


SEGMENTATION WITH PAGING

Segmentation with Paging – MULTICS

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.

- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.

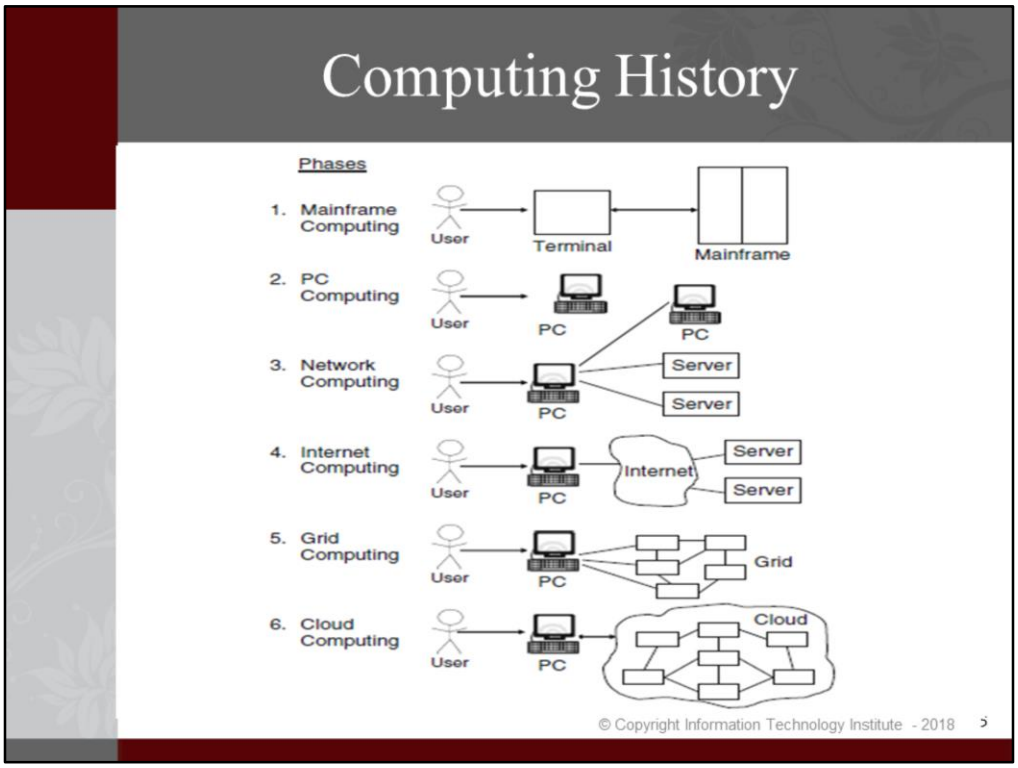

Information Technology Institute
Operating System Fundamentals
© Copyright Information Technology Institute - 2018

CLOUD COMPUTING OVERVIEW

Table of Content

- Introduction
- What is Cloud?
- What is Cloud Computing
- Cloud Computing Architecture
- Deployment Models
- Service Models
- Virtualization
- Cloud Computing Features and Challenges

INTRODUCTION



In phase 1, many users shared powerful mainframes using dummy terminals.
In phase 2, stand-alone PCs became powerful enough to meet the majority of users' needs.
In phase 3, PCs, laptops, and servers were connected together through local networks to share resources and increase performance.
In phase 4, local networks were connected to other local networks forming a global network such as the Internet to utilize remote applications and resources.
In phase 5, grid computing provided shared computing power and storage through a distributed computing system.
At its most basic level, **grid computing** is a **computer** network in which each **computer's** resources are shared with every other **computer** in the system. Processing power, memory and data storage are all community resources that authorized users can tap into and leverage for specific tasks.
In other words: **Grid computing** is a **distributed** architecture of large numbers of **computers** connected to solve a complex problem. In the **grid computing** model, servers or personal **computers** run independent tasks and are loosely linked by the Internet or low-speed networks. **Computers** may connect directly or via scheduling systems.

In phase 6, cloud computing further provides shared resources on the Internet in a scalable and simple way.

Comparing these six computing paradigms, it looks like that cloud computing is a return to the original mainframe computing paradigm. However, these two paradigms have several important differences. Mainframe computing offers finite computing power, while cloud computing provides almost infinite power and capacity. In addition, in mainframe computing dummy terminals acted as user interface devices, while in cloud computing powerful PCs can provide local computing power and cashing support.

Cloud Computing History

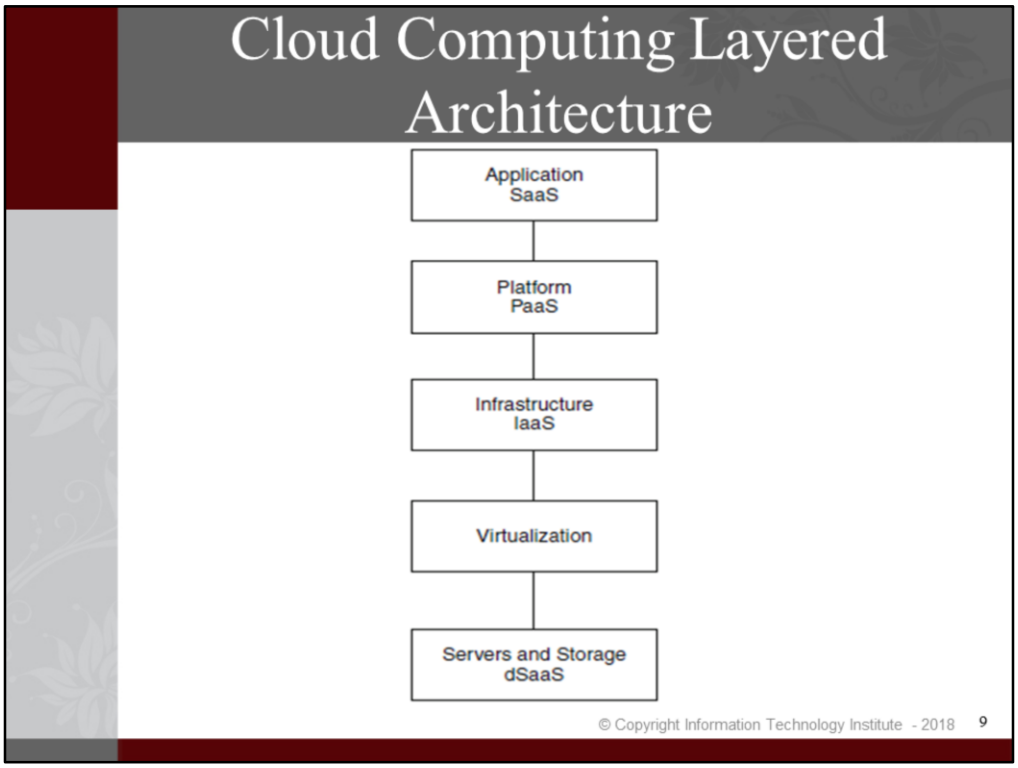
- Concept evaluated in 1950(IBM) called RJE (Remote Job Entry Process).
- In 2006 Amazon provided first public cloud AWS (Amazon Web Service).

What is Cloud?

- The term cloud refers to Network or Internet. Something that is present in at remote location.
- Cloud can provide
  - Services over network (on Public networks or on Private networks).
  - Applications such as email, web conferencing, customer relationship management (CRM)

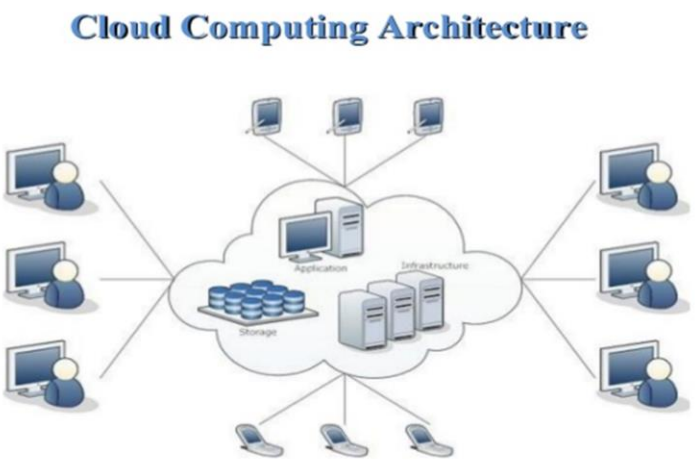
What is Cloud Computing?

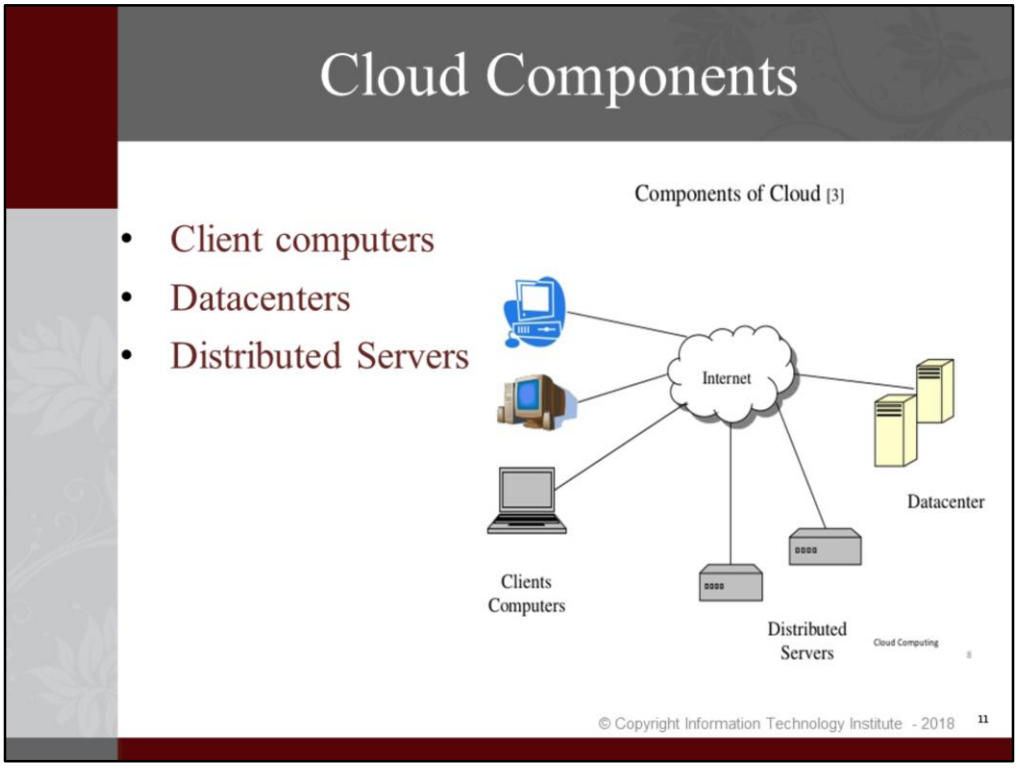
- Cloud computing can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as a services over the Internet.
- Cloud computing refers to manipulating, configuring, and accessing the application online. It offers online data storage, infrastructure and application.
- Cloud computing is both combination of software and hardware based computing resources delivered as a network service.



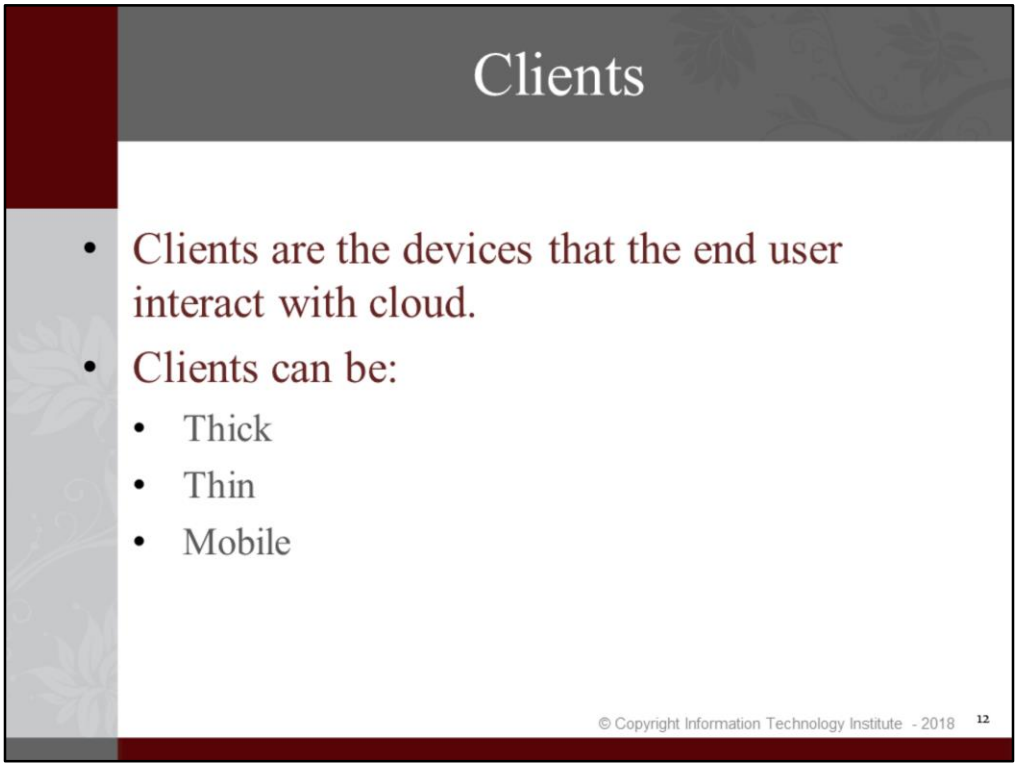
SaaS (Software-as-a-Service), which is shown on top of the stack. SaaS allows users to run applications remotely from the cloud.
Infrastructure-as-a-service (IaaS) refers to computing resources as a service. This includes virtualized computers with guaranteed processing power and reserved bandwidth for storage and Internet access.
Platform-as-a-Service (PaaS) is similar to IaaS, but also includes operating systems and required services for a particular application. In other words, PaaS is IaaS with a custom software stack for the given application.
The data-Storage-as-a-Service (dSaaS) provides storage that the consumer is used including bandwidth requirements for the storage.

Cloud Computing Architecture





The leading vendors of cloud computing components:
Computer hardware: Dell, HP, IBM, Sun
Storage: Sun, EMC, IBM
Infrastructure: Cisco, Juniper Networks, Brocade Communication
Computer software: 3tera. Eucalyptus. G-Eclipse. Hadoop
Operating systems: Solaris, AIX, Linux (Red Hat, Ubuntu)
Platform virtualization: Citrix, VMWare, IBM, Xen, Linux KVM, Microsoft, Sun xVM



A **fat client** (also called heavy, rich or **thick client**) is a computer (**client**) in **client**–server architecture or networks that typically provides rich functionality independent of the central server.

A **thin client** is a lightweight computer that is purpose-built for remoting into a server (typically cloud or desktop virtualization environments). It depends heavily on another computer (its server) to fulfill its computational roles.

Datacenter & Distributed Servers

- Datacenter is a collection of servers where applications is placed and is accessed via internet.
- Distributed Servers are in different places geographically, but they are working as if they are next to each other.

Central Server

- It administrates the system such as monitoring traffic, clients requests to ensure everything runs smoothly.
- It uses a special type of software called middleware.
- Middleware software allows computers to communicate with each other.

Models for Cloud Computing

- Deployment models
- Service models

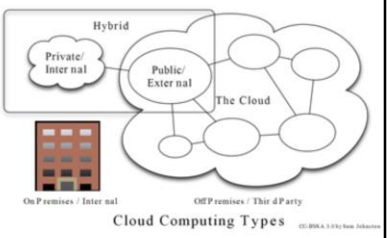
Deployment Models

- Deployment models define the type of access to the cloud.
- There are four types of access
  - Public
  - Private
  - Hybrid
  - Community

Deployment Models cont'd

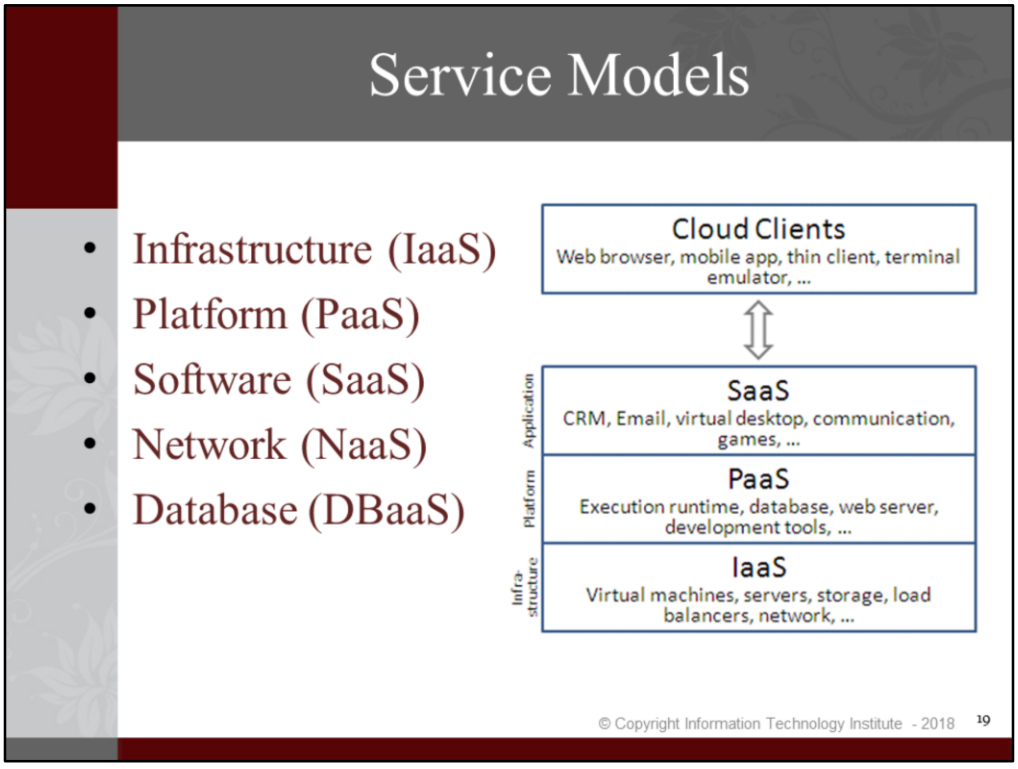
- Public cloud
  - It allows systems and services to be accessible to the general public.
  - It may be less secure due to its openness (email).
- Private cloud
  - It allows systems and services to be accessible within an organization
  - It is more secure due to its private nature

Deployment Models cont'd



Cloud Computing Types

- Hybrid cloud
  - It is a mixture of public and private cloud.
  - The critical activities are performed using private cloud while non critical activities are performed using public cloud.
- Community cloud
  - It allows systems and services to be accessible by a group of organizations



While many in the industry can debate the components, there are 11 major categories or patterns of cloud computing technology:
Storage-as-a-service
Database-as-a-service
Information-as-a-service
Process-as-a-service
Application-as-a-service
Platform-as-a-service
Integration-as-a-service
Security-as-a-service
Management/governance-as-a-service
Testing-as-a-service
Infrastructure-as-a-service

Storage-as-a-service (also known as disk space on demand), as you may expect, is the ability to leverage storage that physically exists at a remote site but is logically a local storage resource to any application that requires storage. This is the most primitive component of cloud computing and is a component or pattern that is leveraged by most of the other cloud computing components.
Database-as-a-service (DaaS) provides the ability to leverage the services of a remotely hosted database, sharing it with other users and having it logically

function as if the database were local. Different models are offered by different providers, but the power is to leverage database technology that would typically cost thousands of dollars in hardware and software licenses.

Information-as-a-service is the ability to consume any type of information, remotely hosted, through a well-defined interface such as an API. Examples include stock price information, address validation, and credit reporting.

Process-as-a-service is remote resource that can bind many resources together, such as services and data, either hosted within the same cloud computing resource or remotely, to create business processes. You can think of a business process as a meta-application that spans systems, leveraging key services and information that are combined into a sequence to form a process. These processes are typically easier to change than are applications and thus provide agility to those who leverage these process engines that are delivered on demand.

Application-as-a-service (AaaS), also known as software-as-a-service (SaaS), is any application that is delivered over the platform of the Web to an end user, typically leveraging the application through a browser. While many people associate application-as-a-service with enterprise applications such as Salesforce SFA, office automation applications are indeed applications-as-a-service as well, including Google Docs, Gmail, and Google Calendar.

Platform-as-a-service (PaaS) is a complete platform, including application development, interface development, database development, storage, testing, and so on, delivered through a remotely hosted platform to subscribers. Based on the traditional time-sharing model, modern platform-as-a-service providers provide the ability to create enterprise-class applications for use locally or on demand for a small subscription price or for free.

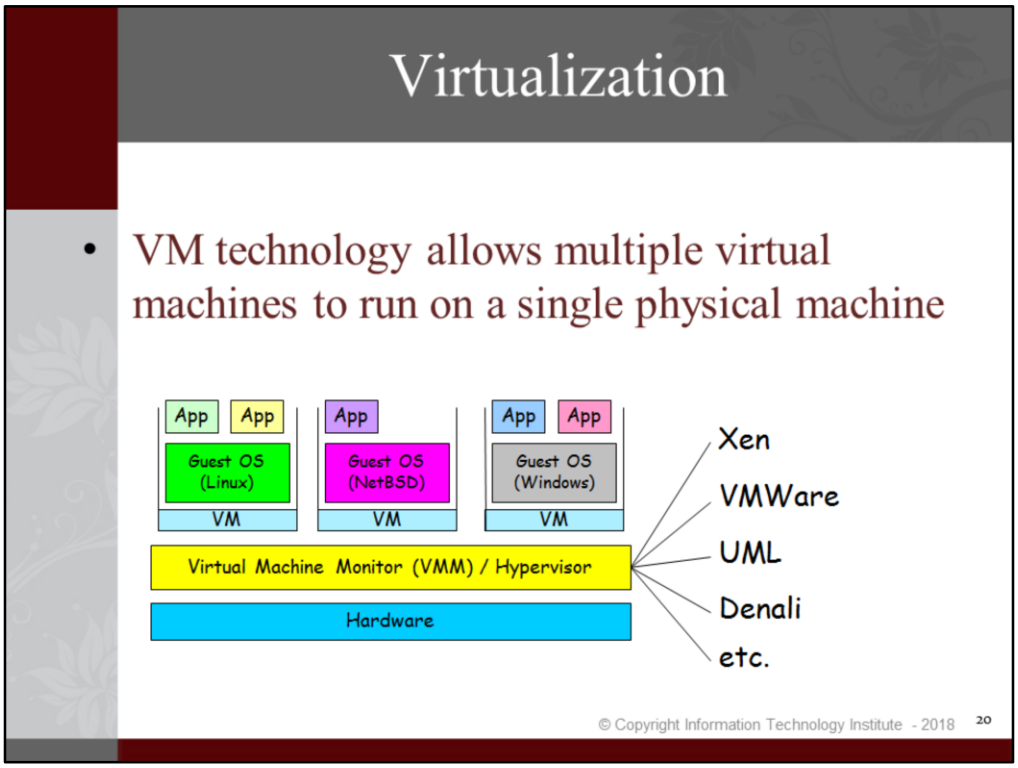
Integration-as-a-service is the ability to deliver a complete integration stack from the cloud, including interfacing with applications, semantic mediation, flow control, integration design, and so on. In essence, integration-as-a-service includes most of the features and functions found within traditional enterprise application integration (EAI) technology but delivered as a service.

Security-as-a-service, as you may have guessed, is the ability to deliver core security services remotely over the Internet. While the typical security services provided are rudimentary, more sophisticated services such as identity management are becoming available.

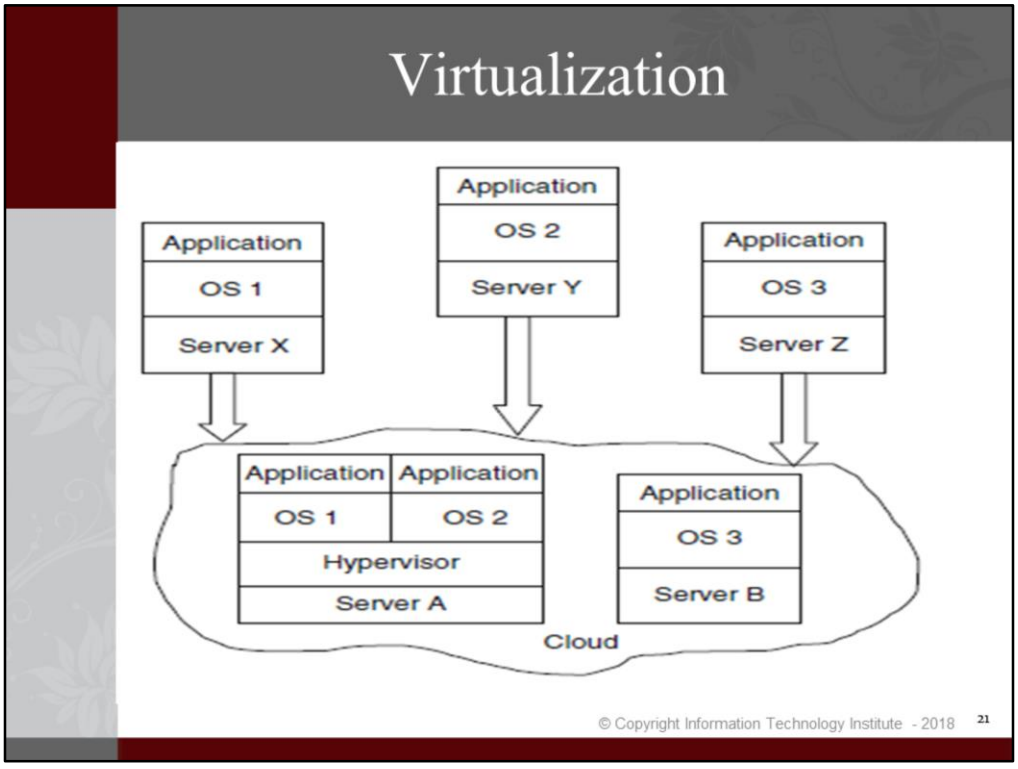
Management/governance-as-a-service (MaaS and GaaS) is any on-demand service that provides the ability to manage one or more cloud services. These are typically simple things such topology, resource utilization, virtualization, and uptime management. Governance systems are becoming available as well, offering, for instance, the ability to enforce defined policies on data and services.

Testing-as-a-service (TaaS) is the ability to test local or cloud-delivered systems using testing software and services that are remotely hosted. It should be noted that while

a cloud service requires testing unto itself, testing-as-a-service systems have the ability to test other cloud applications, Web sites, and internal enterprise systems, and they do not require a hardware or software footprint within the enterprise. ***Infrastructure-as-a-service (IaaS)*** is actually data center-as-a-service, or the ability to remotely access computing resources. In essence, you lease a physical server that is yours to do with as you will and, for all practical purposes, is your data center, or at least part of a data center. The difference with this approach versus more mainstream cloud computing is that instead of using an interface and a metered service, you have access to the entire machine and the software on that machine. In short, it is less packaged.

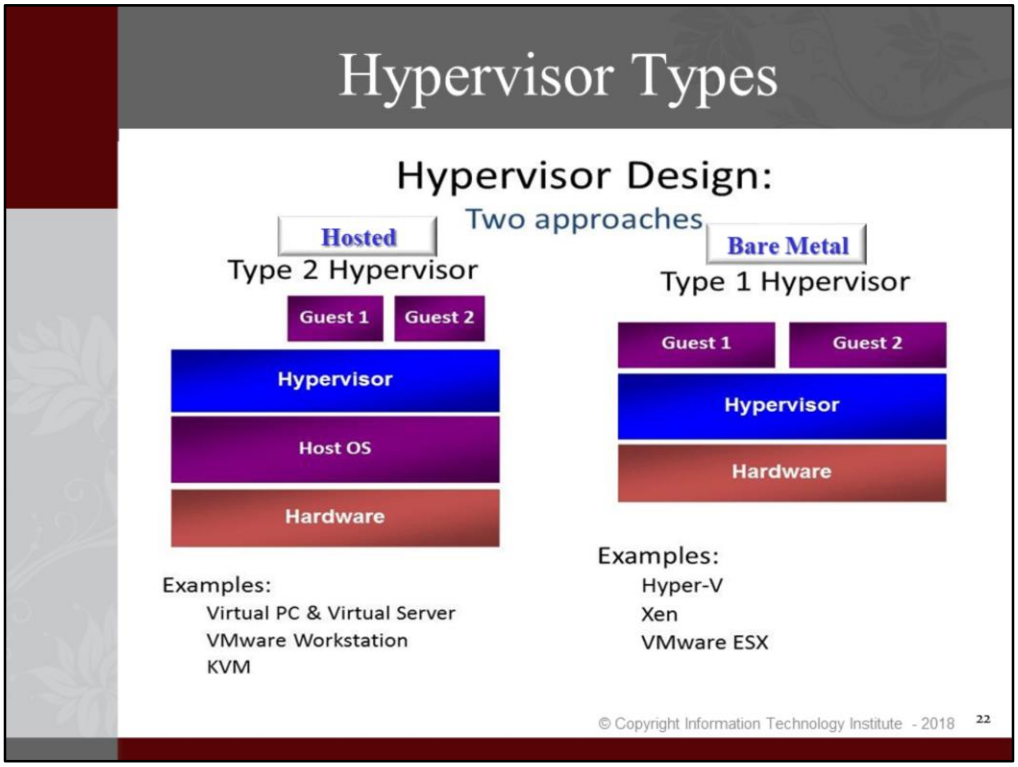


A **hypervisor** or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines. A computer on which a **hypervisor** runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine.



Virtual Machine is a completely separate individual operating system installation on your usual operating system. It is implemented by software emulation and hardware virtualization.

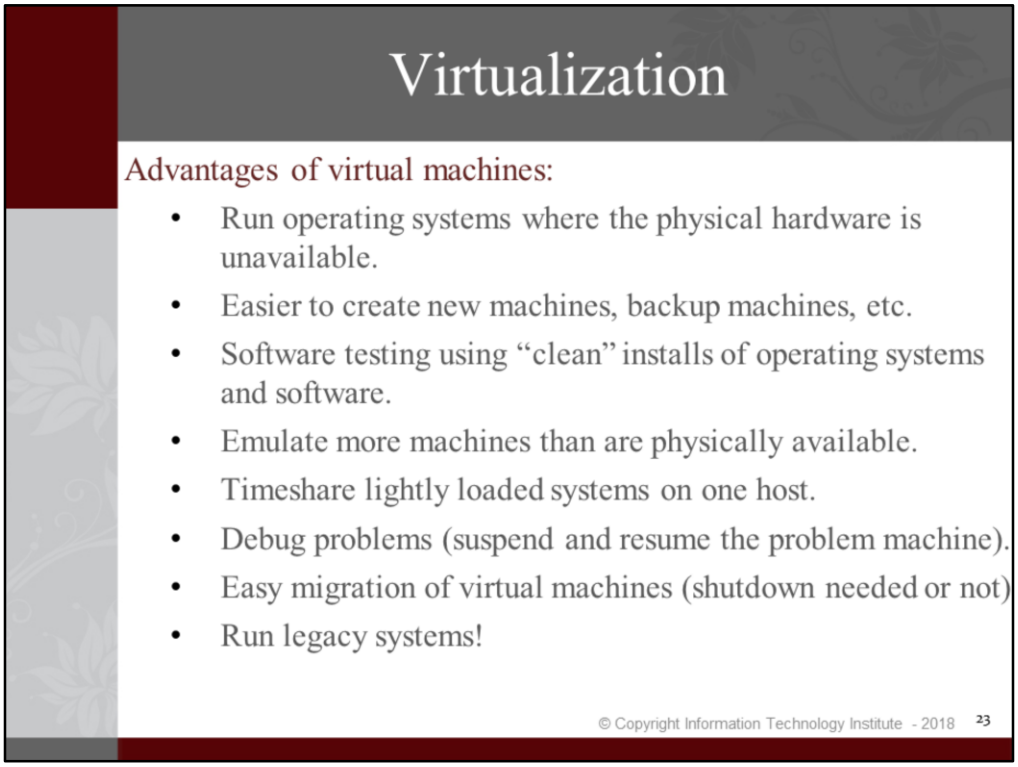
Virtual machine is a software implementation of a physical machine - computer - that works and executes analogically to it. Virtual machines are divided in two categories based on their use and correspondence to real machine: system virtual machines and process virtual machines. First category provides a complete system platform that executes complete operating system, second one will run a single program.



In a virtualization hypervisor comparison, the place to start is to understand the two types of hypervisors on the market, which are:

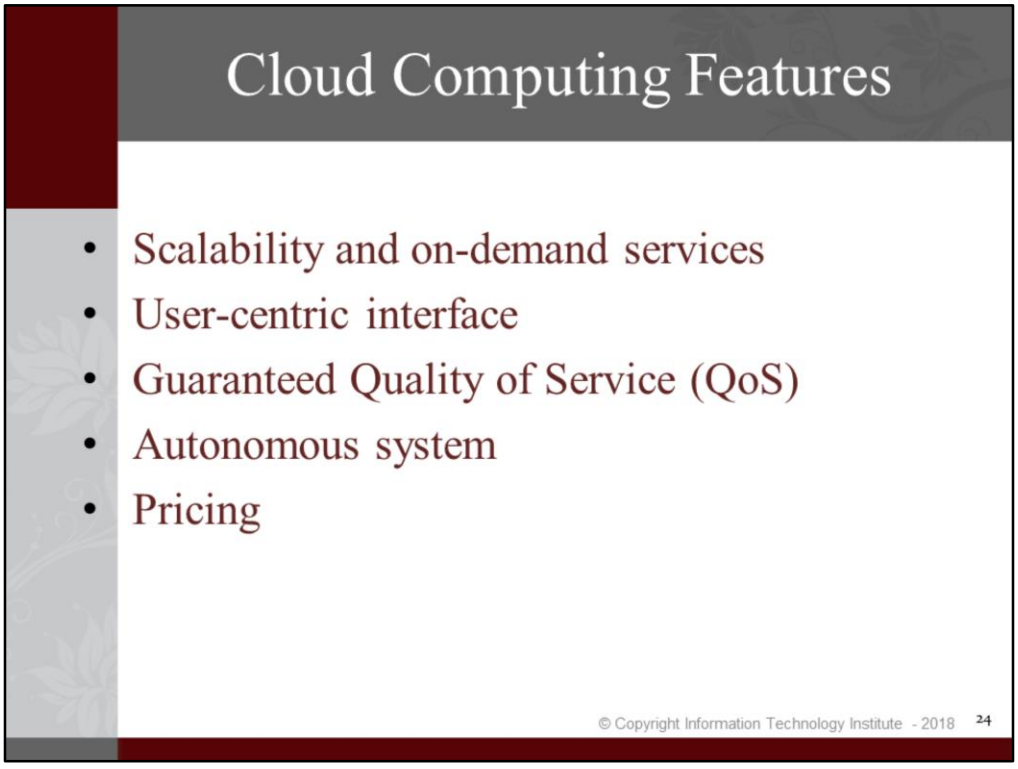
Type 1, which is considered a bare-metal hypervisor and runs directly on top of hardware.

Type 2, which operates as an application on top of an existing operating system.



Primary advantages of server virtualization
Reduce number of servers.
Reduce TCO.
Improve availability and business continuity.
Increase efficiency for development and test environments.
Improve availability of your virtual environment.
Assume a mixed virtual environment.

The main advantages of virtual machines:
Multiple OS environments can exist simultaneously on the same machine, isolated from each other;
Virtual machine can offer an instruction set architecture that differs from real computer's;
Easy maintenance, application provisioning, availability and convenient recovery.



• Scalability and on-demand services

Cloud computing provides resources and services for users on demand. The resources are scalable over several data centers.

• User-centric interface

Cloud interfaces are location independent and can be accesses by well established interfaces such as Web services and Internet browsers.

• Guaranteed Quality of Service (QoS)

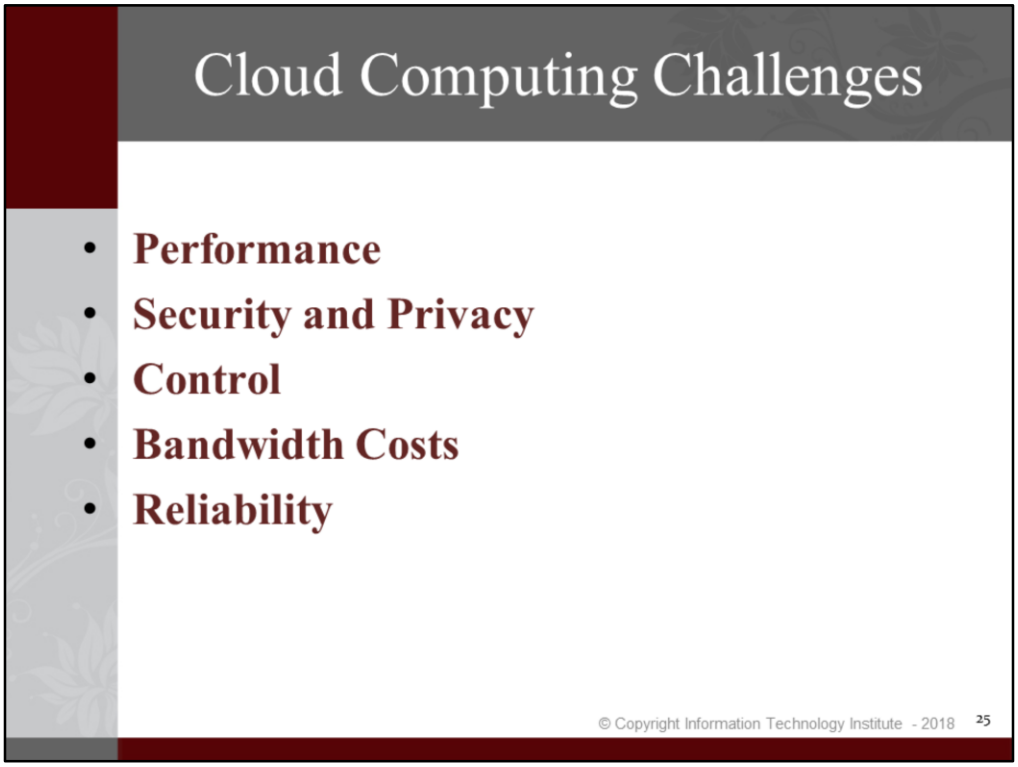
Cloud computed can guarantee QoS for users in terms of hardware/CPU performance, bandwidth, and memory capacity.

• Autonomous system

The cloud computing systems are autonomous systems managed transparently to users. However, software and data inside clouds can be automatically reconfigured and consolidated to a simple platform depending on user's needs.

• Pricing

Cloud computing does not require up-from investment. No capital expenditure is required. Users pay for services and capacity as they need them.



Performance:

The major issue in performance can be for some intensive transaction-oriented and other data-intensive applications, in which cloud computing may lack adequate performance. Also, users who are at a long distance from cloud providers may experience high latency and delays.

Security and Privacy:

Companies are still concerned about security when using cloud computing. Customers are worried about the vulnerability to attacks, when information and critical IT resources are outside the firewall. The solution for security assumes that that cloud computing providers follow standard security practices

Control:

Some IT departments are concerned because cloud computing providers have a full control of the platforms. Cloud computing providers typically do not design platforms for specific companies and their business practices.

Bandwidth Costs:

With cloud computing, companies can save money on hardware and software; however they could incur higher network bandwidth charges. Bandwidth cost may be low for smaller Internet-based applications, which are not data intensive, but could significantly grow for data-intensive applications.

Reliability:

There were cases where cloud computing services suffered a few-hours outages. In

the future, we can expect more cloud computing providers, richer services, established standards, and best practices.

26