



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

# Introduction to operating Systems

## CHAPTER 1

**CSE122/BAI1208/BCS1211/BIT1211/DIT1121**

A sequence of instructions written in a language understood by a computer is called a computer program.

Software refers to a set of programs, procedures, and associated documents (flow charts, manuals, etc.) describing the programs, and how they are to be used.

A software package is a group of programs that solve a specific problem or perform a specific type of job. e.g. a word processing package may contain programs for text editing, text formatting, drawing graphics, spell checking...

# Relationship between Hardware & Software

- Both hardware and software are necessary for a computer to do a useful job. Both are complementary to each other.
- Same hardware can be loaded with different software to make a computer perform different types of jobs.
- Except for upgrades (like increasing main memory and hard disk capacities, or adding speakers, modems, etc.) hardware is normally a one time expense, whereas software is a continuous expense.



# Types of Software

- **System Software:** Is a set of one or more programs designed to control the operation and extend the processing capability of a computer system.

System software performs one or more of the following:

- Supports development of other application software.
- Supports the execution of other application software.
- Monitors the effective use of various hardware resources such as the CPU, memory, peripherals, etc.
- Communicates with and controls operation of peripheral devices such as printer, disk, tape, etc.



# Types of Software continued...

Programs included in a system software package are called system programs. The programmers who prepare system software are referred to as system programmers.

Commonly known types of System Software;

1. **Operating systems:** These take care of efficient effective utilization of all hardware and software components of a computer system.
2. **Programming language translators:** These transform the instructions prepared by programmers in a programming language into a form that can be interpreted and executed by a computer system.
3. **Communications softwares:** These enable transfer of data and programs from one computer system to another in a network environment.
4. **Utility programs:** These help users in system maintenance tasks, and in performing of tasks of routine nature. e.g. formatting hard disks, taking backup of data, sorting of data



# Types of Software continued...

- **Application Software:** Is a set of one or more programs designed to solve a specific problem, or do a specific task.

Programs included in an application software package are called application programs.

The programmers who prepare application software are referred to as application programmers.

Examples include;

- **Word processing software:** enables us to use computers for creating, editing, viewing, formatting, storing, retrieving, and printing documents.
- **Database Software:** enables us to create a database, maintain it, organize its data, and selectively retrieve useful information from it.
- **Graphics software:**
- **Personal Assistance software:**
- **Education software and Entertainment software e.t.c.**

- Is a sequence of instructions (software) substituted for hardware and stored in a Read Only Memory (ROM) chip of a computer.
- Firmware is frequently a cost-effective alternative to wired electronic circuits and its use in compute design is gradually increasing.
- Increased use has today made it possible to produce smart machines of all types. These machines have microprocessor chips with embedded software.

# Middleware



- Is a set of tools and data that helps applications use networked resources and services.
- It is a separate software layer that acts as “glue” between the client and server parts of an application and provides a programming abstraction as well as masks the heterogeneity of underlying networks, hardware, and operating system from application programmers.
- Examples include;
  - Web servers
  - Application servers
  - Telecommunication softwares
  - Transaction monitors
  - Messaging and queueing software

# Computer Programming Languages



A language acceptable to a computer system is called a *Computer language* or *programming language*, and the process of off writing instructions in such a language is called *programming* or *coding*.

The words and symbols of a computer language must be used as per the set rules known as *syntax rules*.

Computer programming languages are classified into two;

- Low Level Languages
- High Level languages

# Low Level Languages (LLL)



- **Machine Language:** Is the only language the computer understands without using a translation program.
- Is normally written as a string of 1s and 0s but doesn't necessarily have to be. It can also be written using decimal digits if the computer circuitry permits this.
  
- The circuitry of a computer is wired in a way that it recognizes the machine language instructions immediately, and converts them into electrical signals needed to execute them.

# Advantages & Disadvantages of Machine Language

## ❖ Advantages:

- Can be executed very fast because of no need for translation.
- They occupy less memory.
- Processing speed is high because it is one to one language.

## ❖ Disadvantage:

- *Machine Dependent*: Internal design of every computer is different from the other, so also the language differs.
- *Difficult to program*: it needs the programmer to memorize dozens of operation code numbers, keep track of storage locations of data and instructions, must know hardware structure.
- *Error prone*: makes it difficult for the programmer to concentrate fully on the logic resulting in errors.
- *Difficult to modify*: Checking machine instruction s to locate errors is very difficult and time consuming.

# Advantages & Disadvantages of Assembly Language

## ❖ Advantages:

- Easier to understand and use
- Easier to locate and correct errors: Assemblers automatically detect and indicate errors.
- Easier to modify
- No worry about addresses.
- Easily relocatable.
- Efficiency of machine language.

## ❖ Disadvantages:

- Machine dependent
- Knowledge of hardware required
- Machine level coding

# Low Level Languages (LLL)



- **Assembly or Symbolic Language:** This allows instructions and storage locations to be represented by letters and symbols instead of numbers.
- An assembly language must be converted (translated) into its equivalent machine language program before it can be executed on the computer.
- This translation is done by the help of a translator program called an **Assembler**.
- Assembler is system software supplied by the computer manufacturers.
- It is called assembler because in addition to translating, it also assembles the machine language program in main memory of the computer and makes it ready for execution.
- During the process of translation, the source program is NOT under execution. It is only converted into a form that can be executed by the computer.

# Introduction

A computer system can be divided roughly into four components:

- **Hardware** - The central processing unit (CPU), the memory, and the input / output (I/O) devices - provide the basic computing resources.
- **Application Programs** - define the ways in which these resources are used to solve the computing problems of the users.
- **Operating System** - controls and coordinates the use of the hardware among the various application programs for the various users.

## Users

## Definition:



- An **Operating System** (often referred to as OS) is an integrated set of programs that controls the resources (CPU, memory, I/O devices, etc.) of a computer system and provides its users with an interface or *virtual machine* that is easier to use than the bare machine.

# Primary Objectives of the OS

## **Make a computer system easier to use.**

It hides details of hardware resources from programmers and other users & provides them with a convenient interface for using a computer system. It acts as an intermediary between the hardware and its users, providing high-level interface to low-level hardware.

## **Manage the resources of a computer system**

It involves such tasks as keeping track of who is using what resources, granting resource requests, accounting for resource usage, and mediating conflicting requests from different programs and users. Efficient and fair sharing of system resources among users and/ or programs is a key goal of all operating systems.

# Main Functions Of An Operating System...

- **Process Management:** taking care of creation and deletion of processes, scheduling of different resources to different processes and providing mechanisms for synchronization and communication among processes.
- **Memory Management:** taking care of allocation & de-allocation of memory space to programs.
- **File Management:** file related activities such as organization, storage, retrieval, naming, sharing and protection of files.
- **Security:** protecting the resources and information of a computer system against destruction and unauthorized access.
- **Command interpretation:** interpreting user commands, directing resources to process commands



# Measuring System Performance

Efficiency of an Operating System and overall performance of a computer system are measured usually in terms of the following parameters.

- **Throughput:** Is the amount of work that the system is able to do per unit time.
- **Turnaround Time:** How long it takes a system to complete a job submitted by a user. It is the interval between the time of submission of a job to the system for processing to the time of job completion
- **Response Time:** it is the interval between the time of submission of a job to the system for processing to the time of the system producing the first response.



# Functions of the Operating System

Operating systems vary in the way they accomplish their tasks.

- **Mainframe operating systems** are designed primarily to optimize utilization of hardware.
- **Personal computer (PC) operating systems** support complex games, business applications, and everything in between.
- **Handheld computer operating systems** are designed to provide an environment in which a user can easily interface with the computer to execute programs.
- Thus, some operating systems are designed to be convenient, others to be efficient, and some others, a combination of the two.

# Types of Operating system

## □ Simple Batch OS

- The users create programs, data and control information
- It is submitted to operator
- Its major task was to transfer control from one job to another
- Jobs with same needs were batched together
- **SPOOLING**
  - Using of buffer

**OS**

**User area**

## Multi-programmed

### Batched OS

- Job pool
- Start with one job
- When I/O is required, it stop the job
- It goes for another job
- When the job finishes I/O it joins the Job pool



- Time sharing OS
  - Interactive use of the system
  - Time slots
  - Virtual memory

## □ Personal Computer OS

- Single user
- One user so one program at a time
- Consist of 2 parts
  - BIOS- Basic Input Output System (ROM)
  - DOS
- When power is turned on BIOS takes control
  - Power on self test
    - Checking memory is OK and all other relevant units functions
    - Reads small portion of OS - Boot Program -Booting the system

## □ Multi- processor OS (parallel system)

- More than one processor
- Called tightly coupled system because the processors can share the memory or a clock
- More work done in a shorter time
- Increases reliability
- Two models
  - Symmetric multiprogramming model
    - Same job
  - Asymmetric multiprogramming model - master slave
    - Different job

- **Distributed Systems**
  - Distributed among several processors
  - Loosely coupled
  - Resource sharing
  - Computation speed up - load sharing
  - Reliability - sharing of work when one fails
  - Communication

## □ Real Time Systems

- | Designed to response to events that happen in real time
- | Time constrains
- | Airline reservation system
- | Response time is short
- | Two models
  - Hard real time system- complete at a specified time
  - Soft real time system- priority is set

# Operating System Structure

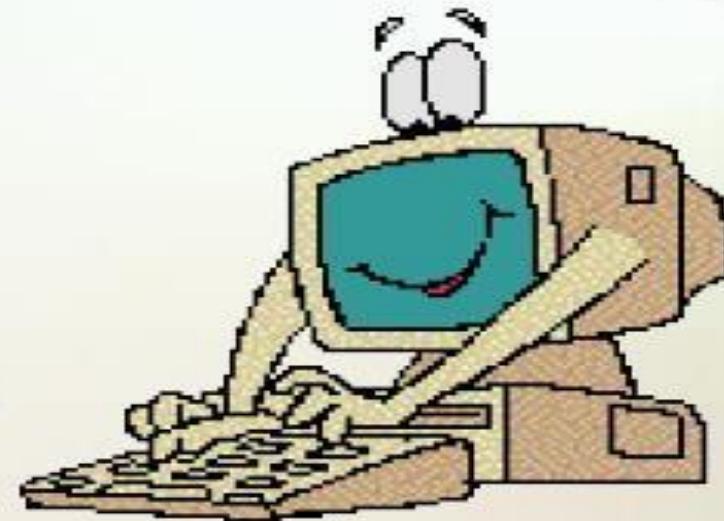


- Layered Approach
- The Kernel Based Approach
- The Virtual machine Approach

**More Clarity**

# What is an Operating System?

- The **most important** program that runs on your computer. It **manages** all other programs on the machine.
- Every PC **has to have one** to run other applications or programs. It's the first thing “**loaded**”.



# Operating System

- It performs basic tasks, such as:
  - Recognizing input from the keyboard or mouse,
  - Sending output to the monitor,



(c) 2002 Michael P. Stryk

# Operating System

- Keeping track of files and directories on the disk, and
- Controlling peripheral devices such as disk drives and printers.



# Is There More Than One Type of OS?

- Generally, there are four types, based on the type of computer they control and the sort of applications they support.

## 1. Single-user, single task

This type manages the computer so that one user can effectively do one thing at a time.

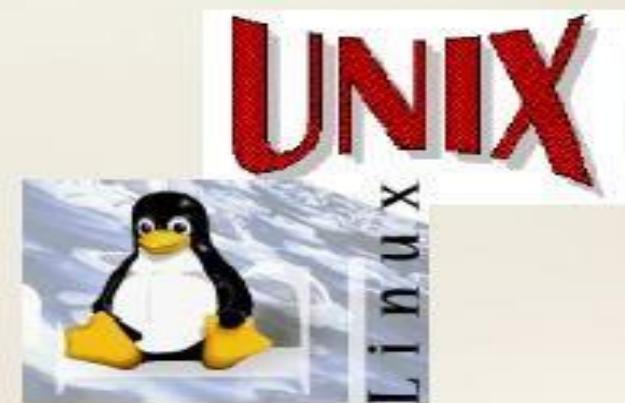


# **Types of Operating Systems**

## **2. Multi-user, multi-task**

**Allows two or more users to run programs at the same time. Some operating systems permit hundreds or even thousands of concurrent users.**

**Mainframe**



# Types of Operating Systems

## 3. Real Time Operating Systems

**RTOS** are used to control machinery, scientific instruments, and industrial systems.

There is typically very little user-interface capability.

Resources are managed so that a *particular operation executes precisely the same every time*.

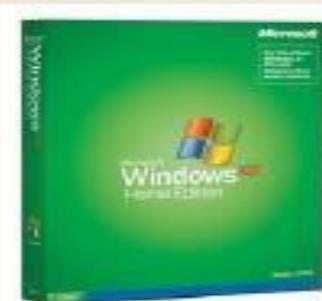


# Types of Operating Systems

## 4. Single-user, Multi-tasking

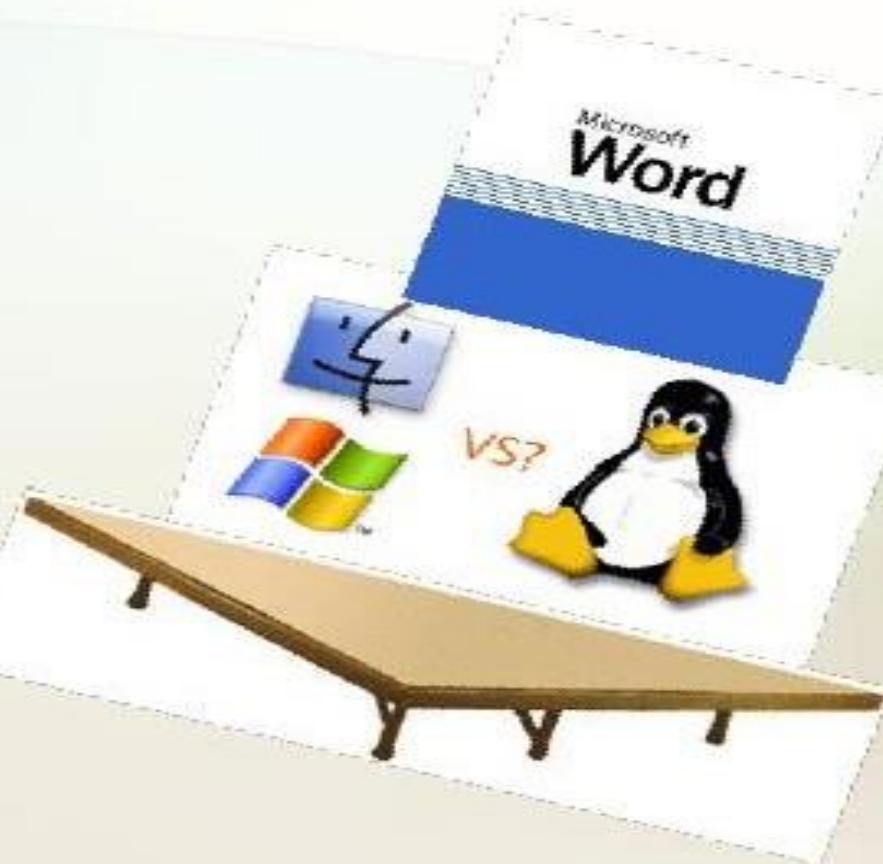
*This is the type of operating system **most desktops and laptops use today.***

**Microsoft's Windows and Apple's MacOS** are both examples of operating systems that will let a **single user have several programs in operation at the same time.**



# OS's Manage Applications

- Operating systems provide a software platform on top of which other “application” programs can run.
  - The application programs must be written to run on a particular operating system.
  - So, your choice of operating system determines what application software you can run.



# Operating System Functions

- Besides managing hardware and software resources on the system, the OS must manage resources and memory.
- There are two broad tasks to be accomplished.

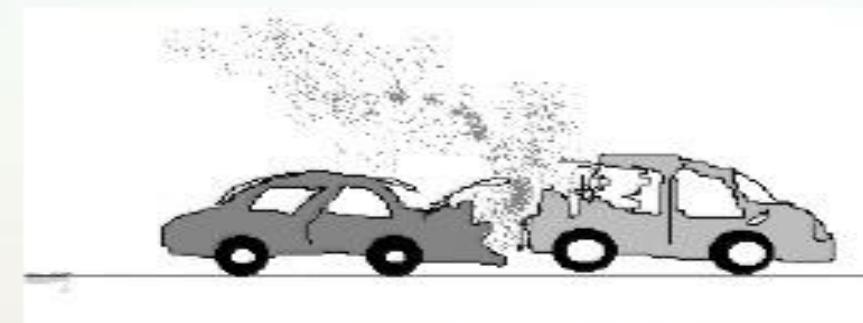


# OS - Memory Storage and Management

1. Each process must have enough memory in which to execute, and

It can neither run into the memory space of another process,

Nor be run into by another process.



# OS - Memory Storage and Management

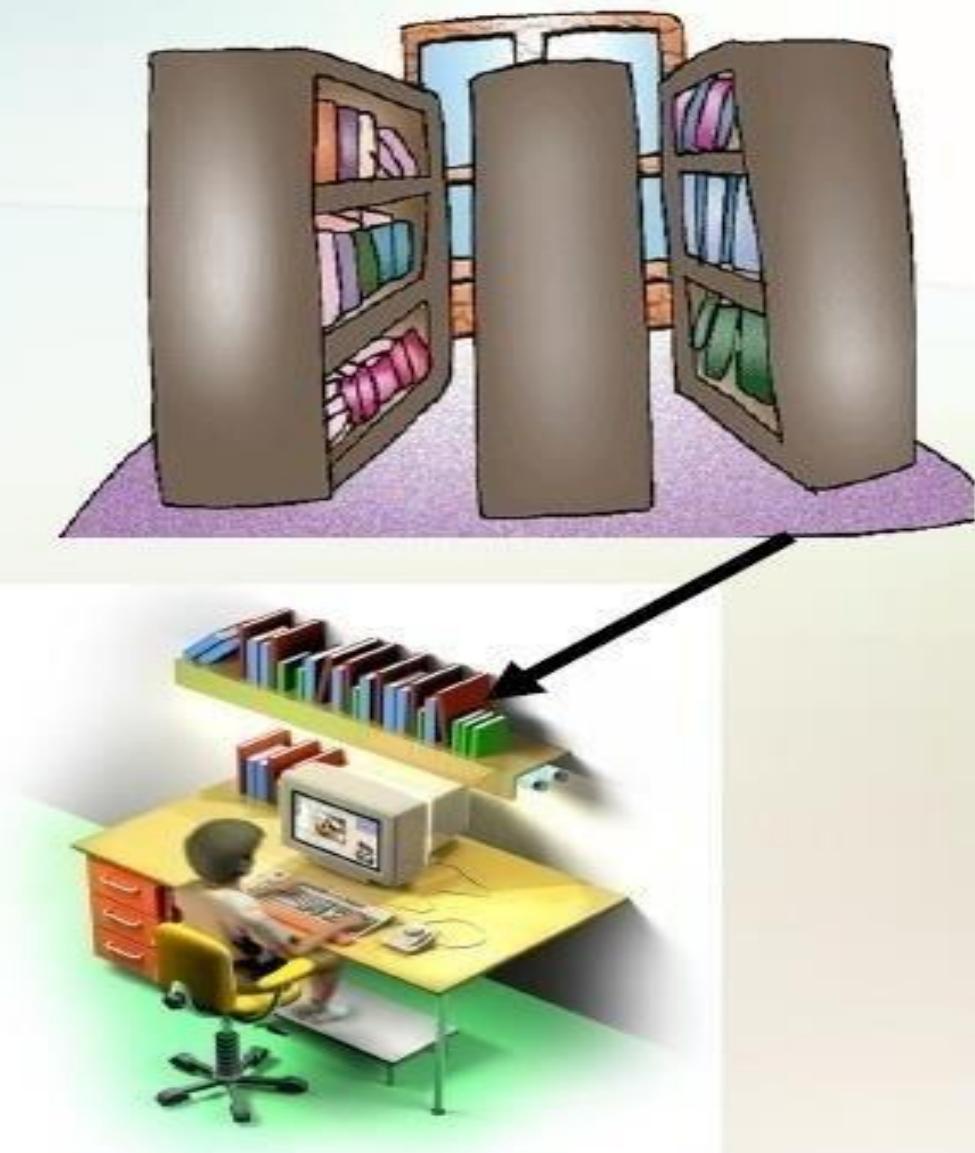
1. The different types of memory in the system **must be used properly** so that each process can run most effectively.

## Memory Management



# Cache Memory

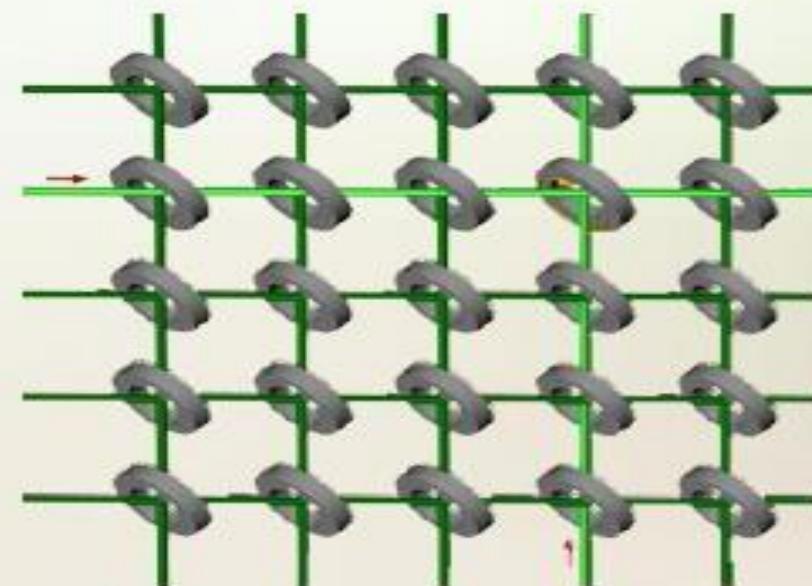
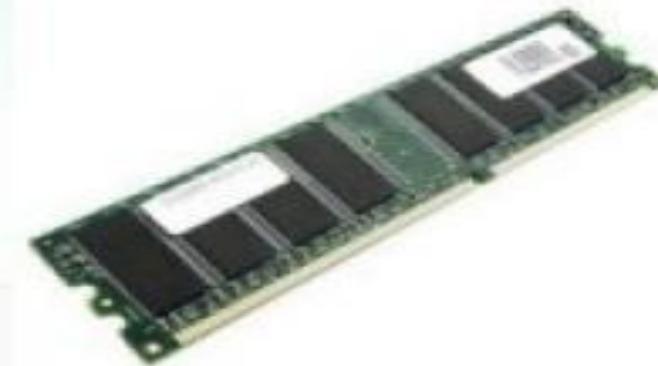
- **Cache** - A section of a computer's memory which temporarily retains recently accessed data in order to speed up repeated access to the same data.
- It provides rapid access without having to wait for systems to load.



# RAM Memory

- **Random access memory (RAM) is the best known form of computer memory.**

- RAM is considered "random access" because you can **access any memory cell directly** if you know the row and column that intersect at that cell.



# **RAM Memory**

- The **more RAM** your computer has, the **faster** programs can function. The two main types are called DRAM and SRAM. SRAM is faster than DRAM, but, more expensive.

*Remember, that if the power is turned off, then all data left in RAM, that has not been saved to the hard drive, is lost.*

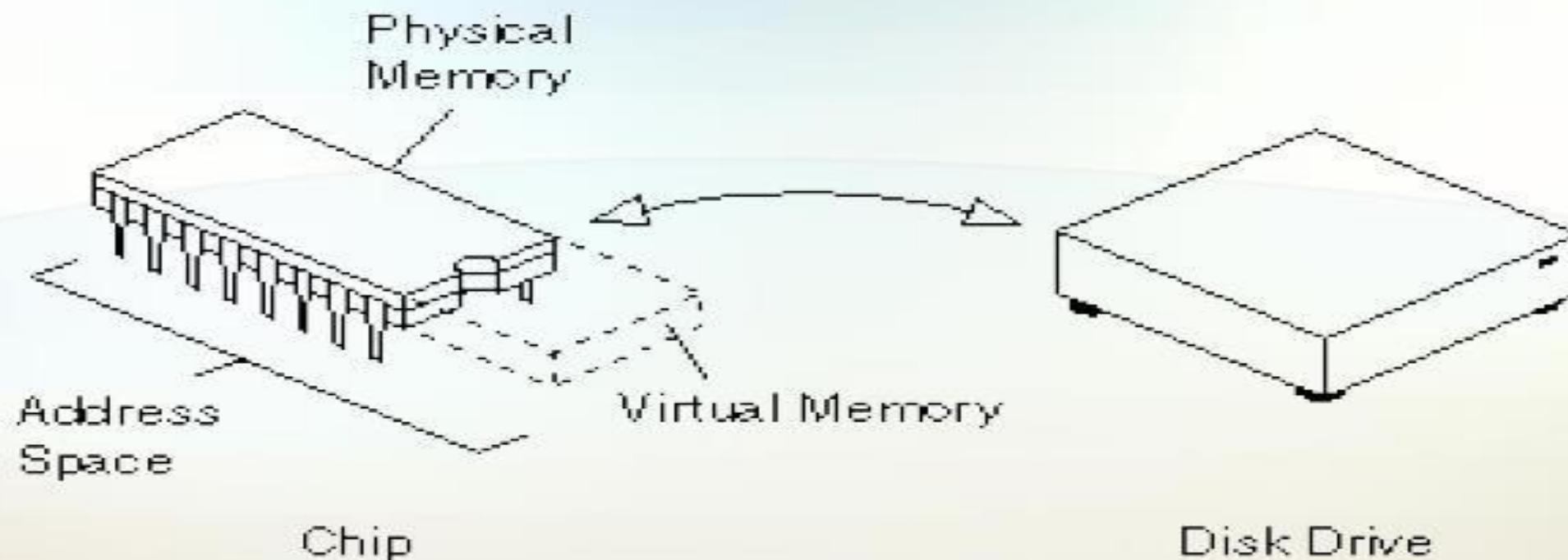
# Virtual Memory

- **Virtual Memory** – a method of using **hard disk space** to provide **extra** memory. It simulates additional RAM.

- In Windows, the amount of virtual memory available, equals the amount of free **RAM** plus the amount of **disk space** allocated to the **swap** file.



# Virtual Memory – Swap File



A swap file is an area of your hard disk that is set aside for virtual memory. Swap files can be either temporary or permanent.

# Okay – So Now What?





## OS - Wake up call

- When you turn on the power to a PC, the first program that runs is a set of instructions kept in the computer's read-only memory (ROM).



# OS - Wake up Call

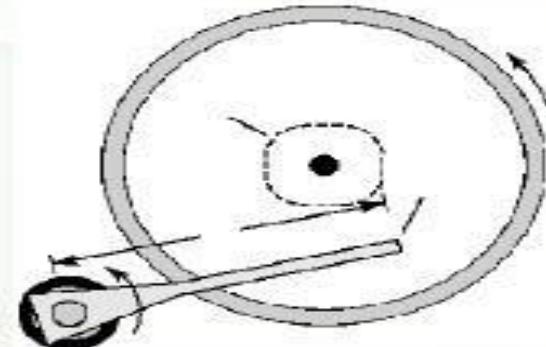


- It **checks** to make sure everything is **functioning properly**.
- It **checks** the CPU, memory, and basic input-output systems (BIOS) **for errors**.



# OS – Wake up Call

- Once successful, the software will begin to activate the computer's disk drives.



- It then finds the first piece of the operating system: the **bootstrap loader**.



# OS - Booting the PC

- 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.

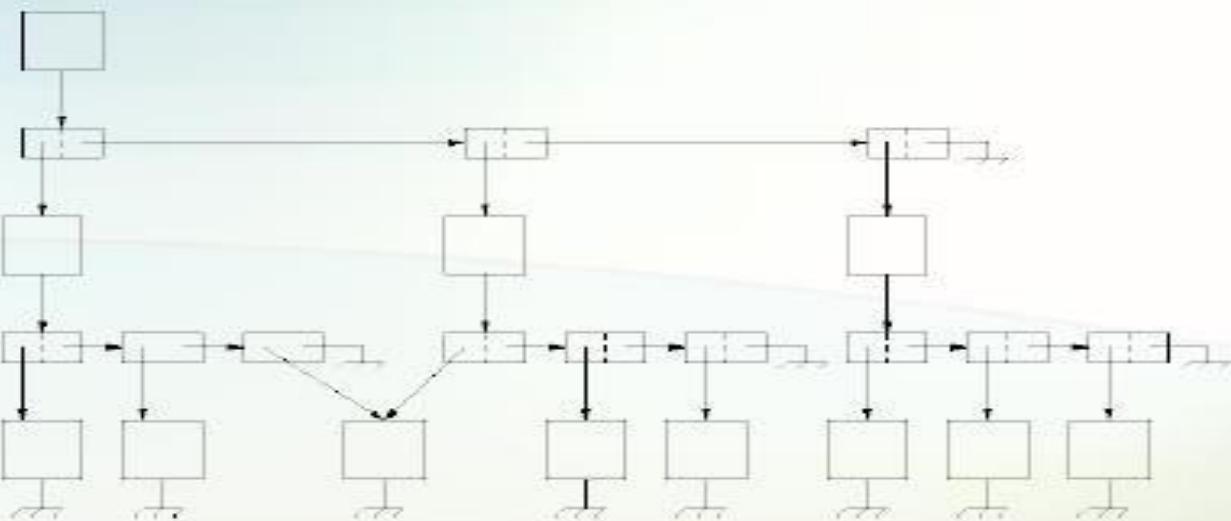
# OS - Booting the PC

- The bootstrap loader **sets up** the small **driver programs** that interface with and control the various hardware.
- It **sets up** the divisions of
  - memory
  - user information, and
  - applications.



# OS - Booting the PC

- It establishes the **data structures** needed to **communicate** within and between the **subsystems** and **applications** of the computer.
- Then it **turns control** of the computer over to the **operating system**.



# How Do I Tell The OS What I Want To Do?

- You must continue to give the operating system commands that are accepted and executed.
- *The first command was pushing the “ON” button which started the “boot” process.*



# Enter Commands

- **Commands can be entered several ways:**
  - Through a **keyboard**.
  - Pointing or clicking on an **object** with a mouse.  
(Graphical User Interface or GUI)
  - Sending a command from **another program**.





# *Windows and Mac are GUI's*



- ***Microsoft Windows and Apple Macintosh operating systems are “graphical user interfaces” or GUI's.***

***GUI is defined as: A picture used in place of a word or words to issue commands.***

# **GUI – Standards**

- **GUI interfaces have standards that are usually the same or similar in all systems and applications.**
- **Standards apply to:**
  - **Pointers and pointing devices**
  - **Icons, desktops, windows and menus**

# **Windows - GUI Pointers**

- **GUI** uses *pictures, symbols, or icons* rather than words to **represent** some object or function. For example:
- A **pointer** or **mouse pointer** is a small arrow or other symbol that moves on the screen as you move a mouse.



# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

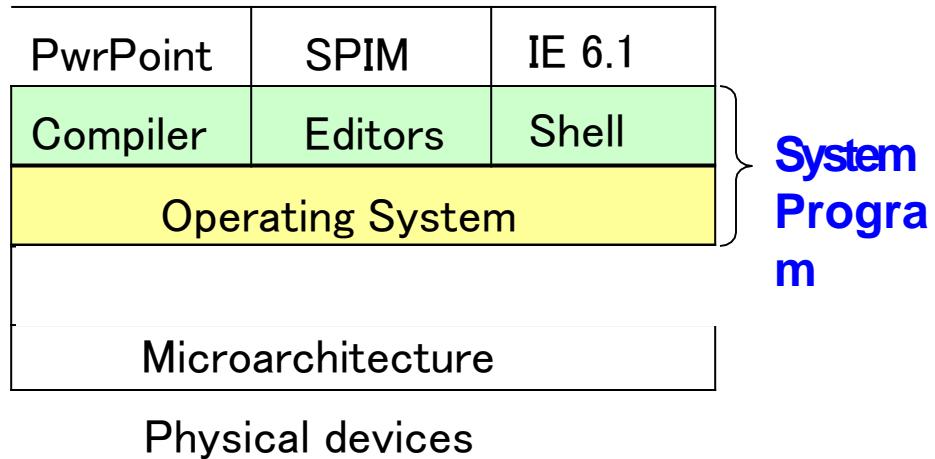
# CHAPTER 2

## Process Management

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

# What is an Operating System?

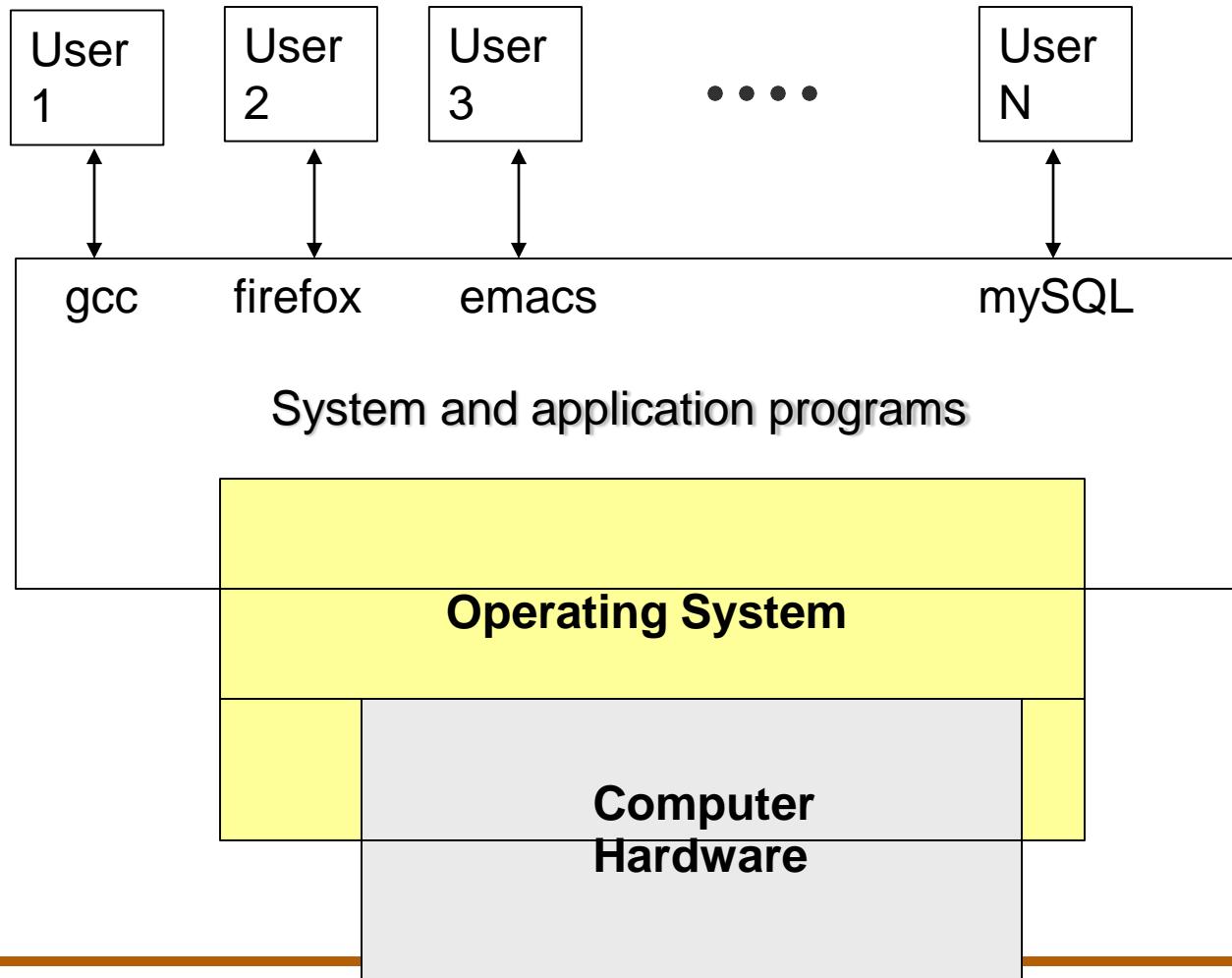
- An intermediate program between a user of a computer and the computer hardware (to hide messy details)
- Goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient and efficient to use



# Computer System Components

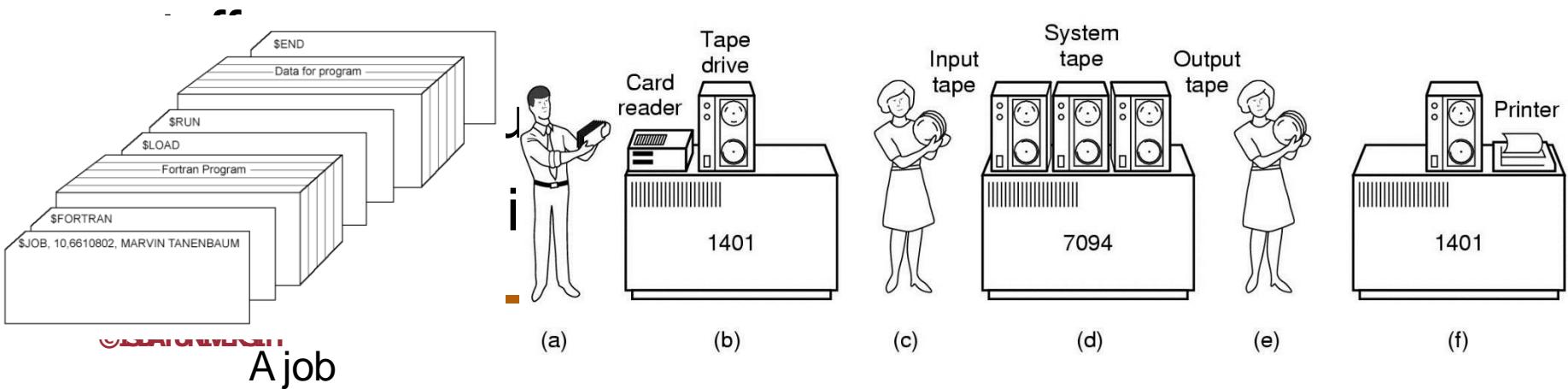
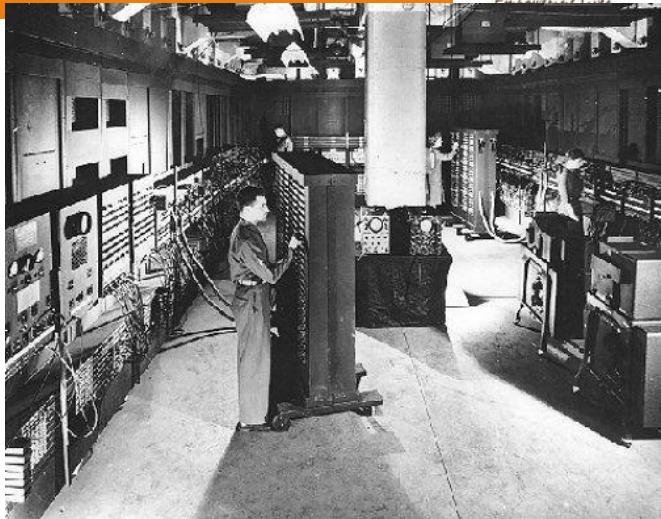
- Hardware
  - Provides basic computing resources (CPU, memory, I/O)
- Operating System
  - Controls and coordinates the use of the hardware among various application programs for various users
- Application Programs
  - Define the ways in which the system resources are used to solve the computing problems of users (e.g. database systems, 3D games, business applications)
- Users
  - People, machines, other computers

# Abstract View of System Components



# History of Operating Systems

- Vacuum Tubes and Plug boards (1945 - 55)
  - Programmers sign up a time on the signup sheep on the wall
- Transistors and batch system (1955 - 65)
  - Mainframes, operated by professional



# 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 OS 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

# Operating System Concepts

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary Management
- Networking
- Protection System
- Command-Interpreter System

# Process Management

- A *process* is a program in execution
- A process contains
  - Address space (e.g. read-only code, global data, heap, stack, etc)
  - PC, \$sp
  - Opened file handles
- A process needs certain resources, including CPU time, memory, files, and I/O devices
- The OS is responsible for the following activities for process management
  - Process creation and deletion
  - Process suspension and resumption
  - Provision of mechanisms for:
    - process synchronization
    - process communication

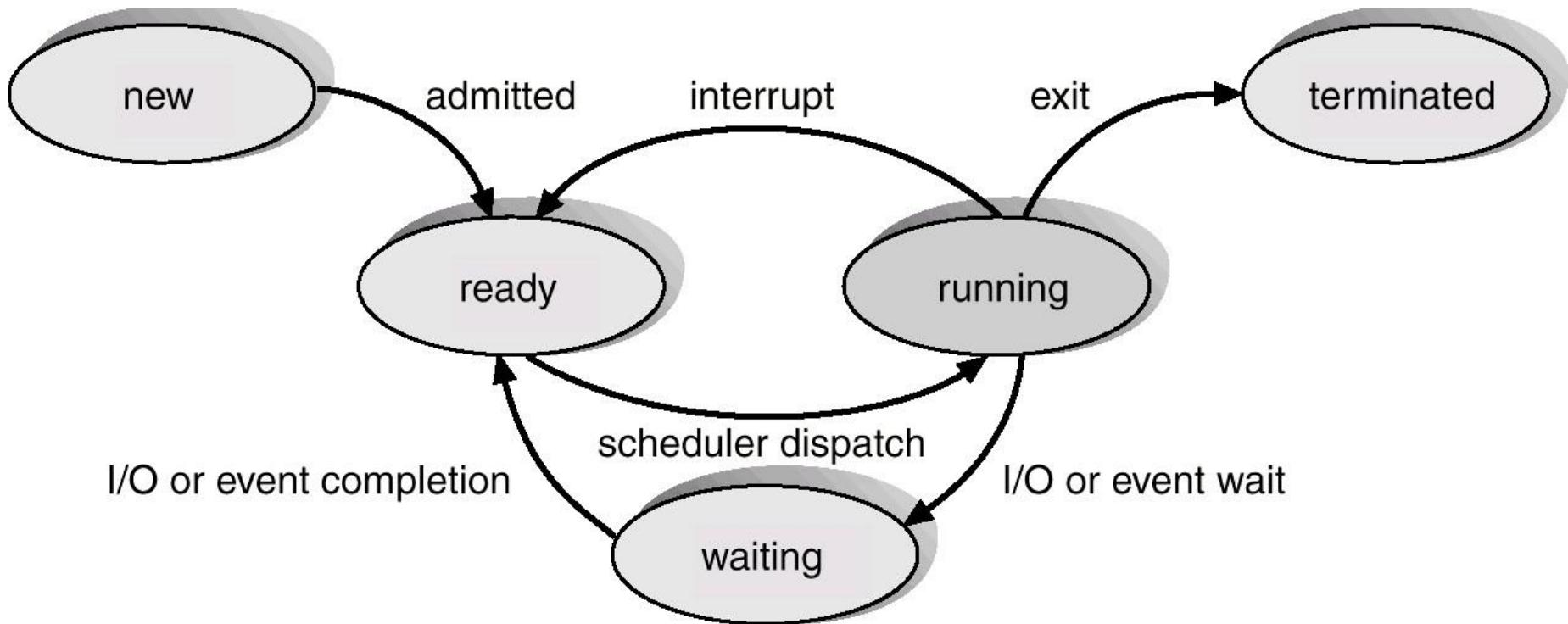
# Process States

- A process is typically in one of the three states
  - 1 **Running**: has the CPU
  - 2 **Blocked**: waiting for I/O or another thread
  - 3 **Ready to run**: on the ready list, waiting for the CPU

# Process State

- As a process executes, it changes state
  - new: The process is being created
  - ready: The process is waiting to be assigned to a process
  - running: Instructions are being executed
  - waiting: The process is waiting for some event (e.g. I/O) to occur
  - terminated: The process has finished execution

# Diagram of Process State

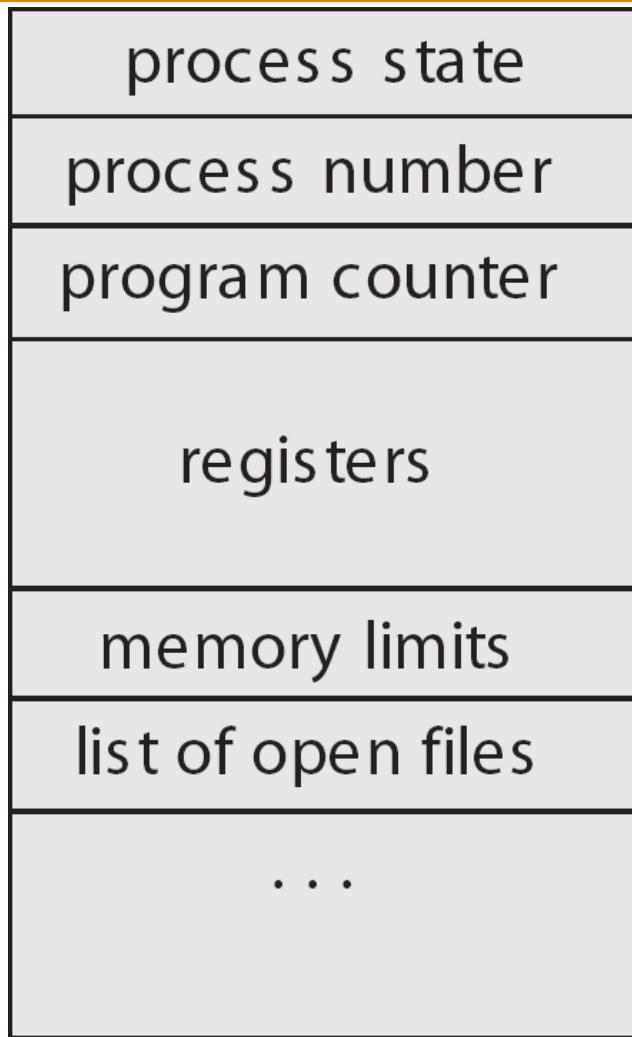


# Process Control Block (PCB)

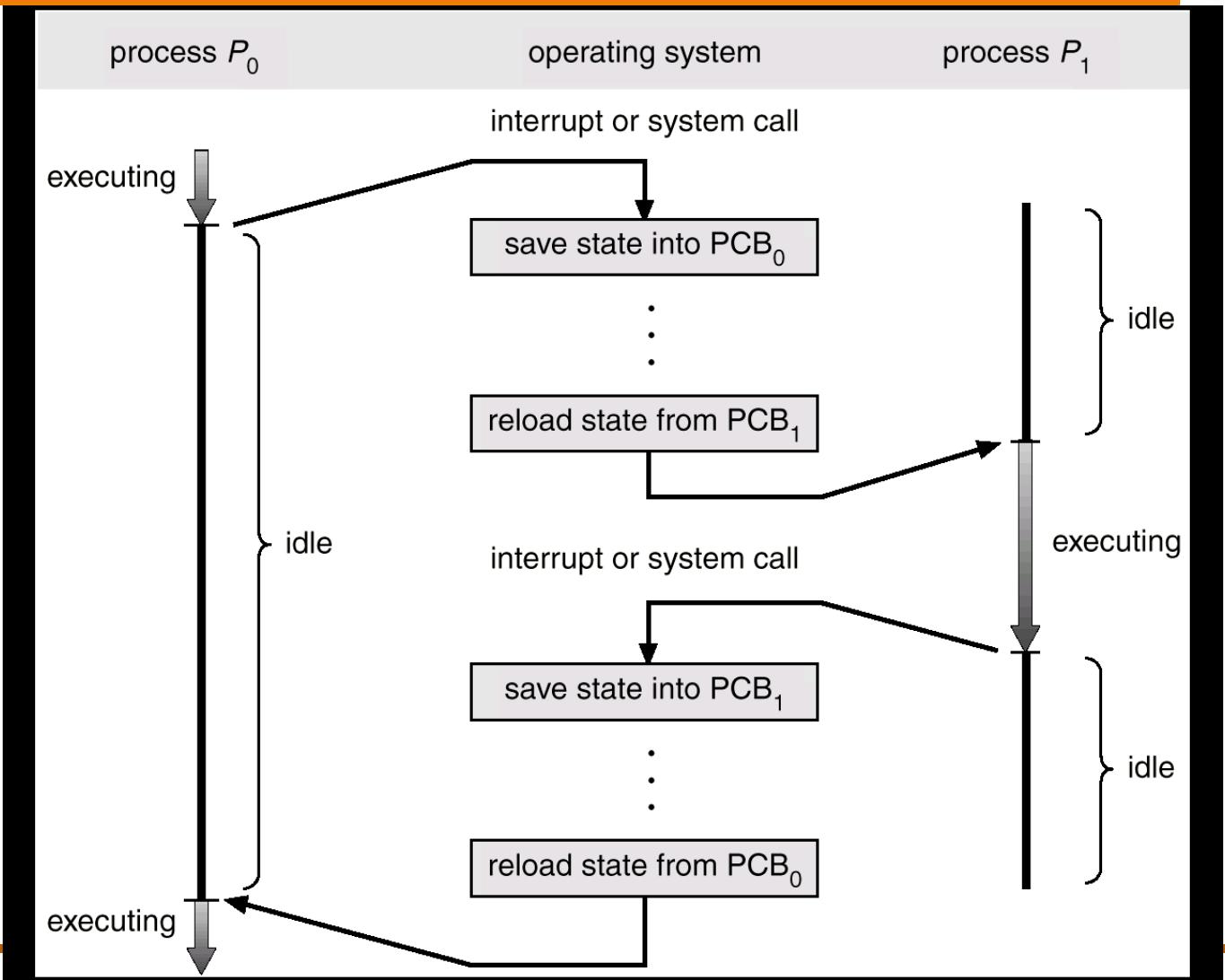
## Information associated with each process

- Process state
- Program counter
- CPU registers (for context switch)
- CPU scheduling information (e.g. priority)
- Memory-management information (e.g. page table, segment table)
- Accounting information (PID, user time, constraint)
- I/O status information (list of I/O devices allocated, list of open files etc.)

# Process Control Block

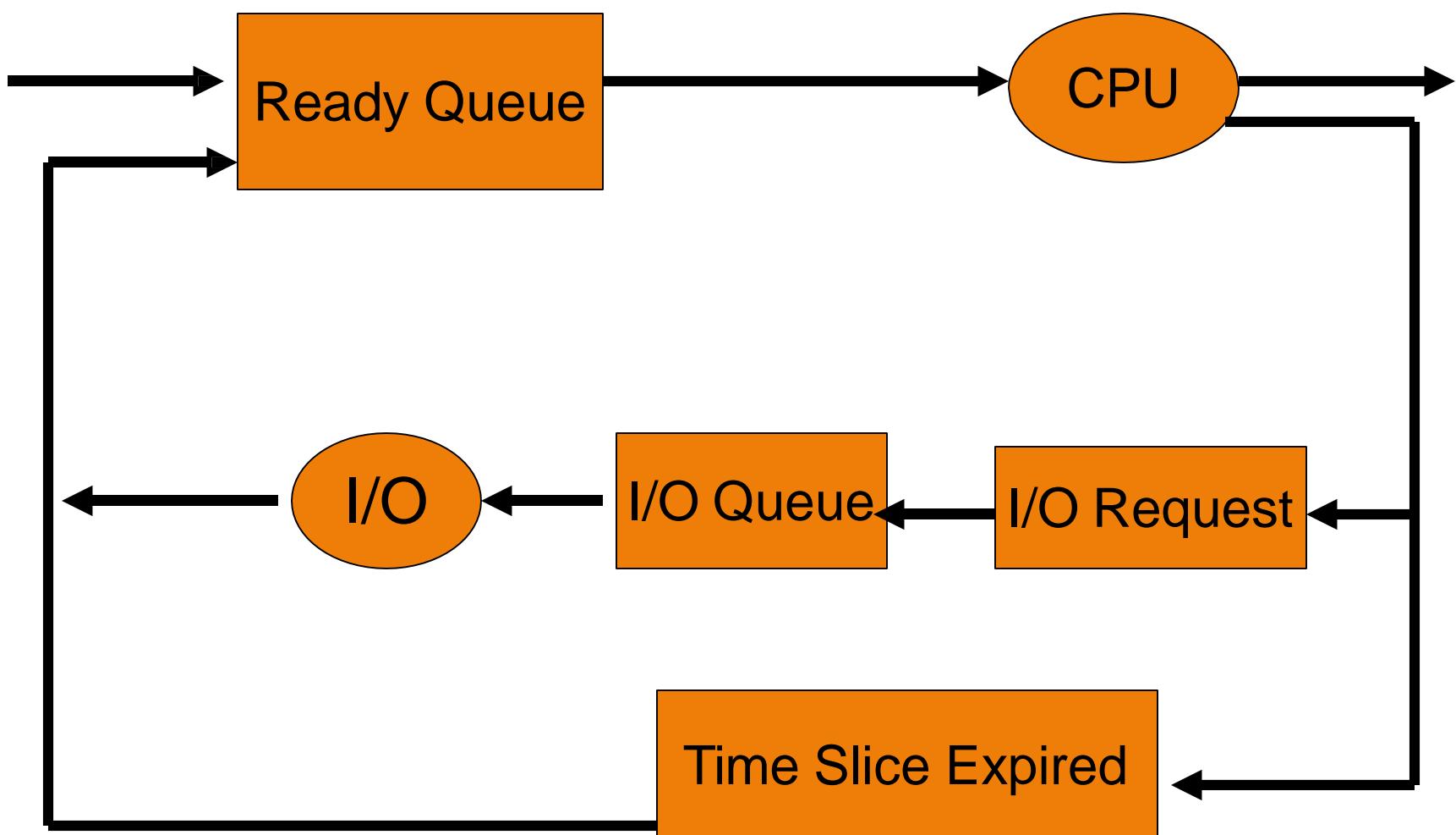


# CPU Switch From Process to Process



# Process Scheduling

- As a process enters the system, it joins a job queue
- Ready state and waiting for CPU for execution
- Ready queue is a list of PCB's implemented as a linked list of PCB pointing to the NXT
- Process get executed until an I/O occurs or time slice expire
- Once a process terminates, entries for this process in all queues are deleted. The PCB and resources allocated are released



# Schedulers

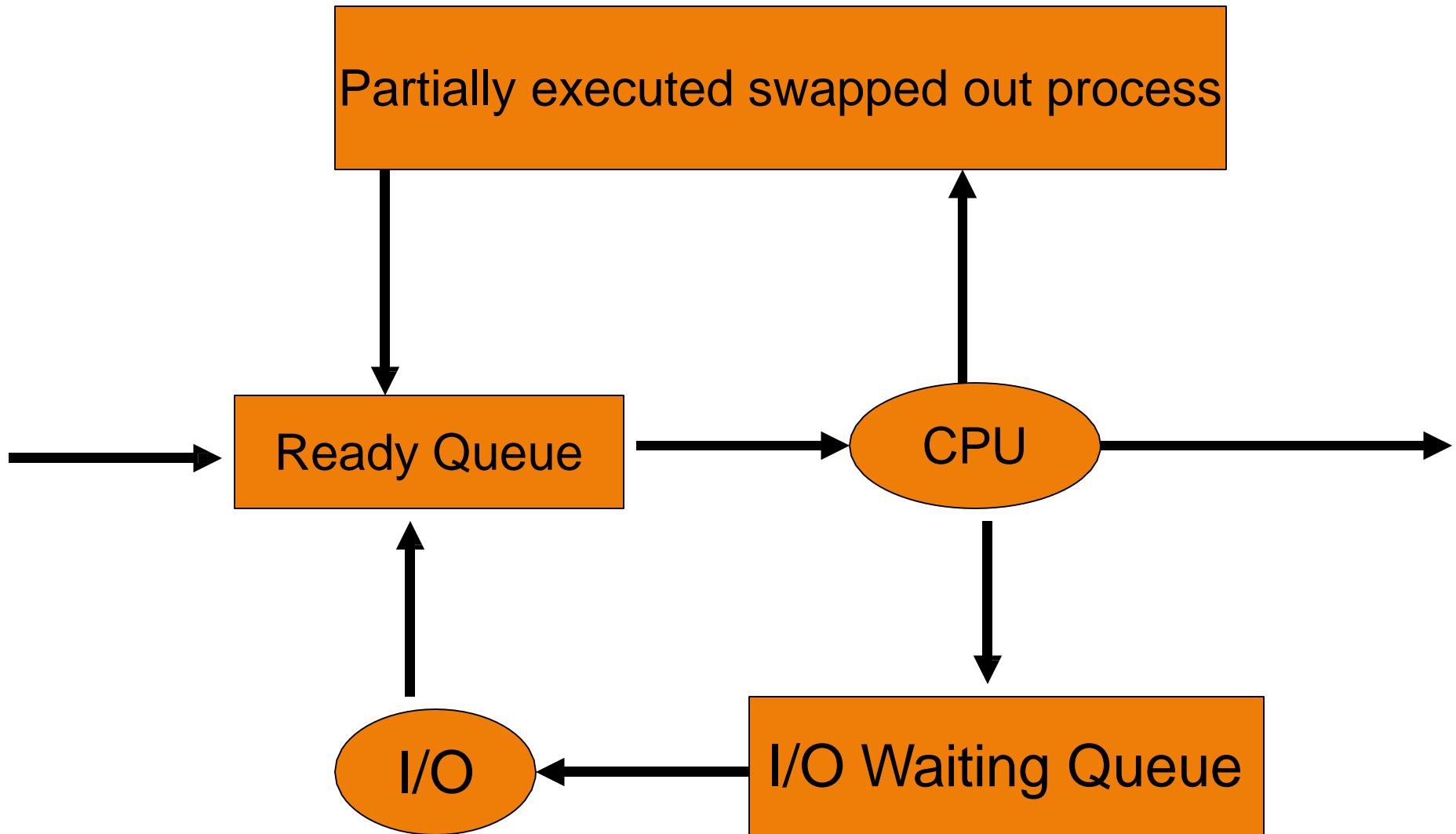
- OS scheduler schedules processes from ready queue for execution by the CPU
- It selects one of the many processes from the ready queue based on certain criteria
- Schedulers could be any one of the following
  - Long term, Short term, Medium term

- Many jobs could be ready for execution at the same time
- Out of these more than one could be spooled onto disk
- The long term scheduler or job scheduler picks and loads processes into memory from among the set of ready jobs
- The short term scheduler or CPU scheduler selects a process from among the ready processes to execute on the CPU

- Both differ in frequency of their execution
- Short terms to select a new process for CPU execution quite often, so it must be very fast
- A long term executes less frequently since new processes are not created at the same phase at which processes need to be executed
- The number of processes present in the ready queue determines the degree of multi programming

- Most of the processes can be classified as either I/O bound (more time on I/O) or CPU bound (More time on CPU)
- A good selection is a good mix of both, in this case I/O and CPU will be busy
- Sometimes processes keep switching between ready, running and waiting states with termination taking a long time
- One reason is increased degree of multi programming meaning more number of ready processes than the system can handle

- Medium term scheduler handles such situation
- When the system throughput falls below a threshold, some of the ready processes are swapped out of memory to reduce the degree of multi programming
- Later they are reintroduced into memory to join the ready queue



## □ Context Switching

- Switching from one process to another requires saving the state of the current process and loading the latest state of the next process

# Dispatching Loop

- The hardware view of the system execution:  
**dispatching loop**

- **LOOP**

- Run process
    - Save process states
    - Choose a new process to run
    - Load states for the chosen process

*Context  
Switch:*

*Dispatcher  
code*

*Scheduling*

# Simple? Not Quite...

- How does the dispatcher (OS) regain control after a process starts running?
- What states should a process save?
- How does the dispatcher choose the next thread?

# How Does the Dispatcher Regain Control?

- Two ways:
  1. Internal events
    - A process is waiting for I/O
    - A process is waiting for some other process
    - Yield— a process gives up CPU voluntarily
  2. External events
    - Interrupts—a complete disk request
    - Timer—it's like an alarm clock

# What States Should a process save?

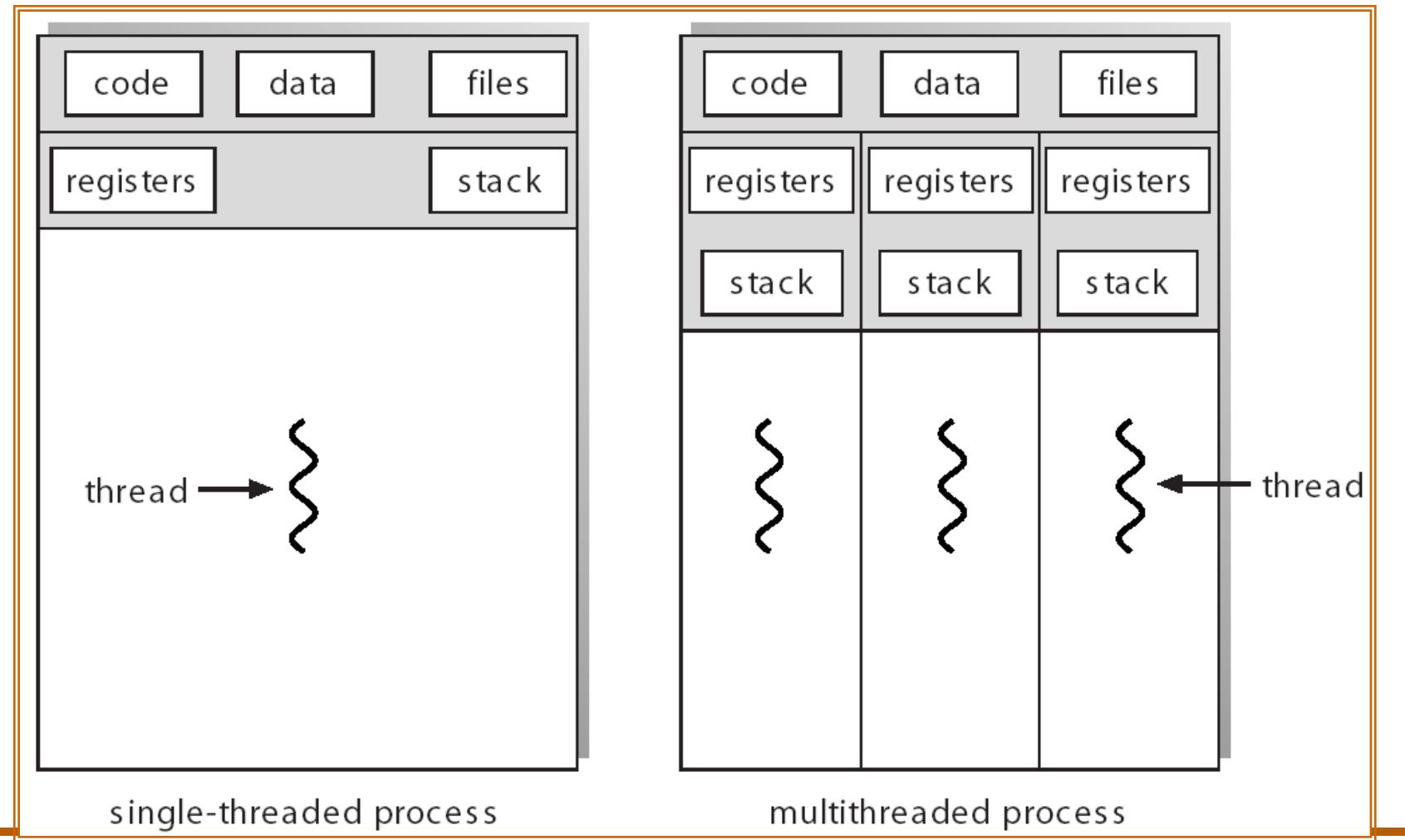
- Anything that the next process may trash
  - Program counter
  - Registers
  - Etc.

# How Does the Dispatcher Choose the Next process?



- The dispatcher keeps a list of processes that are ready to run
- If no processes are ready
  - Dispatcher just loops
- Otherwise, the dispatcher uses a scheduling algorithm to find the next process.

# Single and Multithreaded Processes



# Examples of Threads

- A web browser
  - One thread displays images
  - One thread retrieves data from network
- A word processor
  - One thread displays graphics
  - One thread reads keystrokes
  - One thread performs spell checking in the background
- A web server
  - One thread accepts requests
  - When a request comes in, separate thread is created to service
  - Many threads to support thousands of client requests
- RPC or RMI (Java)
  - One thread receives message
  - Message service uses another thread

# Threads vs. Processes

## Thread

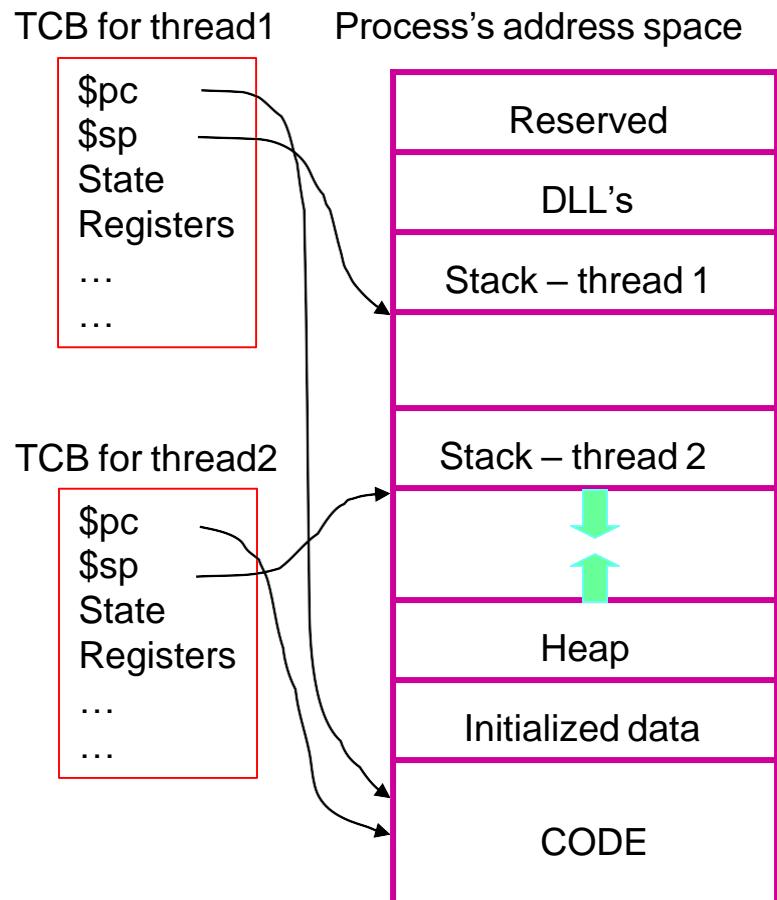
- A thread has no data segment or heap
- A thread cannot live on its own, it must live within a process
- There can be more than one thread in a process, the first thread calls main and has the process's stack
- Inexpensive creation
- Inexpensive context switching
- If a thread dies, its stack is reclaimed by the process

## Processes

- A process has code/data/ heap and other segments
- There must be at least one thread in a process
- Threads within a process share code/data/heap, share I/O, but each has its own stack and registers
- Expensive creation
- Expensive context switching
- If a process dies, its resources are reclaimed and all threads die

# Thread Implementation

- Process defines address space
- Threads share address space
  
  
  
  
  
  
- Process Control Block (PCB) contains process-specific info
  - PID, owner, heap pointer, active threads and pointers to thread info
  
  
  
  
  
  
- Thread Control Block (TCB) contains thread-specific info
  - Stack pointer, PC, thread state, register ...



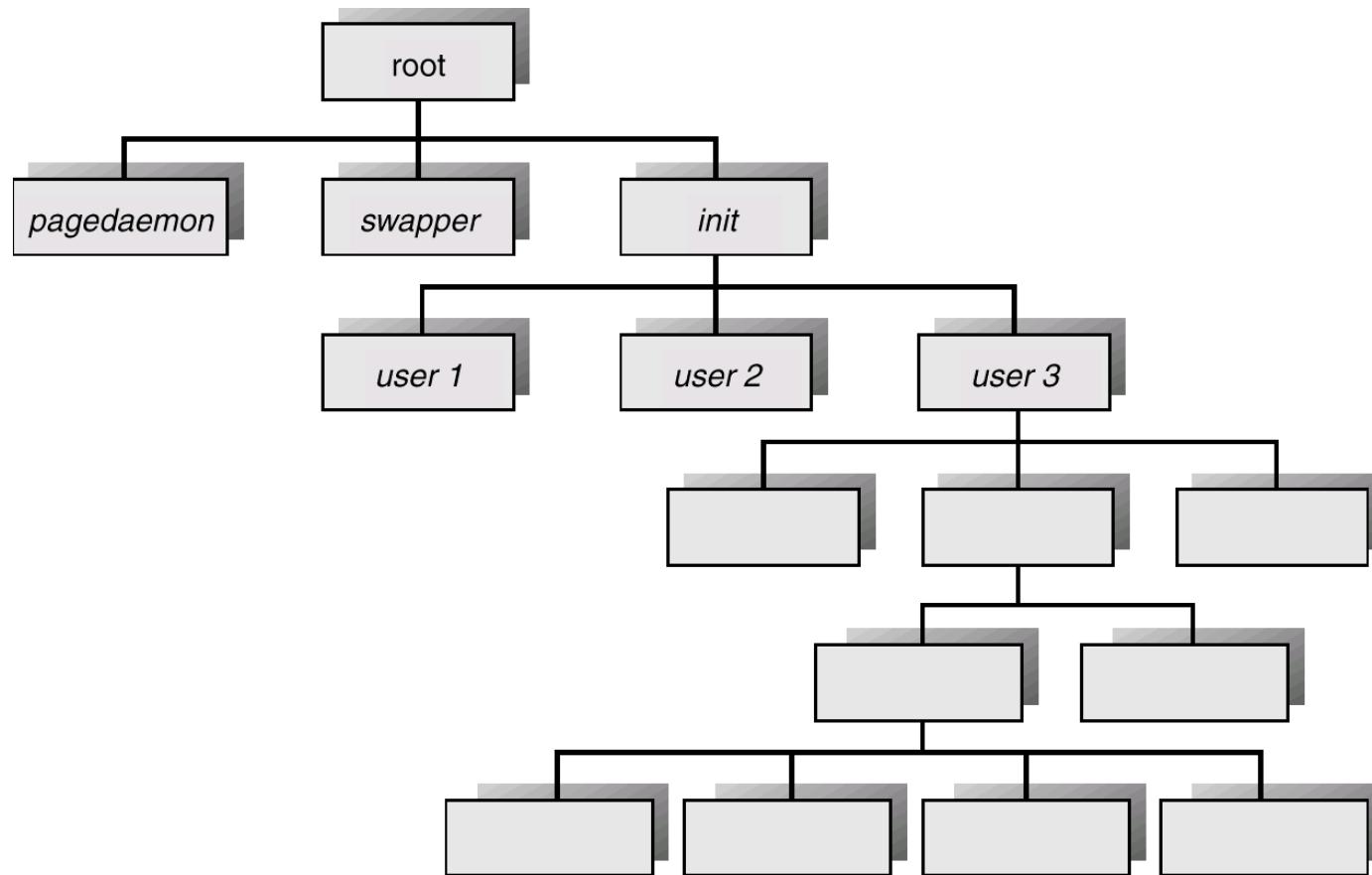
# Benefits

- Responsiveness
  - When one thread is blocked, your browser still responds
  - E.g. download images while allowing your interaction
- Resource Sharing
  - Share the same address space
  - Reduce overhead (e.g. memory)
- Economy
  - Creating a new process costs memory and resources
  - E.g. in Solaris, 30 times slower in creating process than thread
- Utilization of MP Architectures
  - Threads can be executed in parallel on multiple processors
  - Increase concurrency and throughput

# Process Creation

- 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
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

# Processes Tree on a UNIX System



# Process Creation (Cont.)

- 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

# C Program Forking Separate Process

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
    int pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child
process */
        execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
        /* parent will wait for the
child to complete */
        wait(NULL);
        printf("Child Complete");
        exit(0);
    }
}
```

# Process Termination

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

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

## CHAPTER 3

# Operating system in Distributed Processing

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

# Characteristics of Distributed Systems



- Processing may be distributed by location
- Processing is divided among different processors
- Processes can be executed on dissimilar processors
- OS running on each processor may be different

# Characteristics of Parallel processing



- All processors are tightly coupled
- Use shared memory for communication
- Any processor can execute any job
- All processors are similar
- A common operating system

# Distributed Processing



- Distributed - network
- Distributed Applications
  - Different programs in different computers
  - Horizontal Distribution
    - All computers are at the same level
    - All the computers are capable of handling any functionality
  - Vertical / Hierarchical Distribution
    - Functionality is distributed among various computers
    - Computers at different levels performs specialized functions

## □ Distributed Environment

### □ Data for applications could be maintained as

- Centralized data
- Replicated data
- Partitioned data

### □ Centralized data

- Resides only at one central computer
- Master database
- OS that implements the functions of Information management in Central system

### □ Replicated data

- Duplicated
- In case if it is frequently used by the clients

## ■ Partitioned way

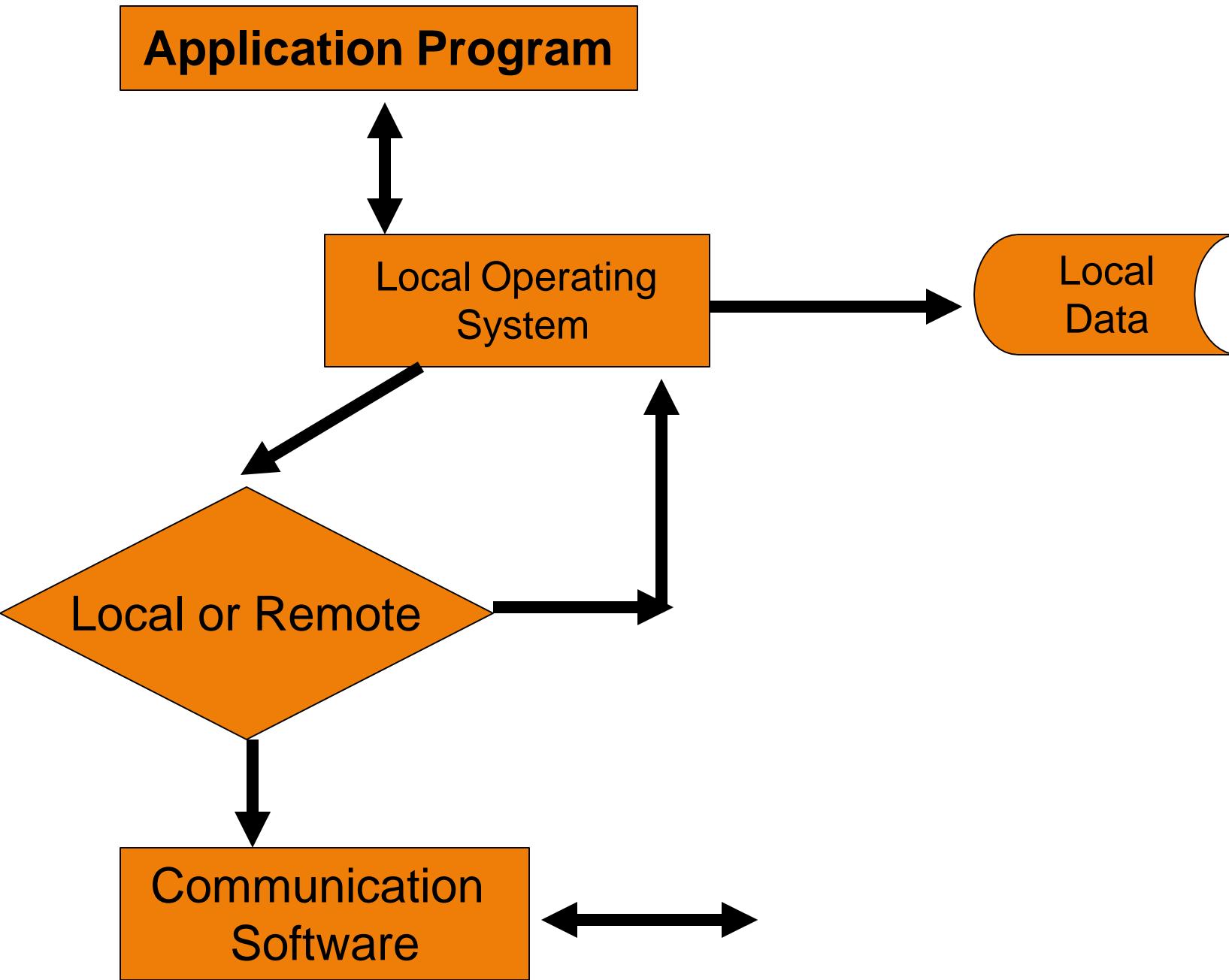
- The entire database is sliced into many parts
- Each part resides on a computer
- Processing depends upon the kind of data distribution

## □ Distribution of control

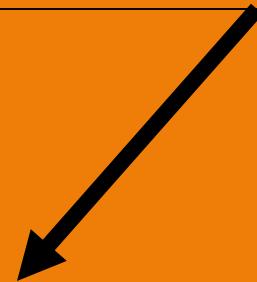
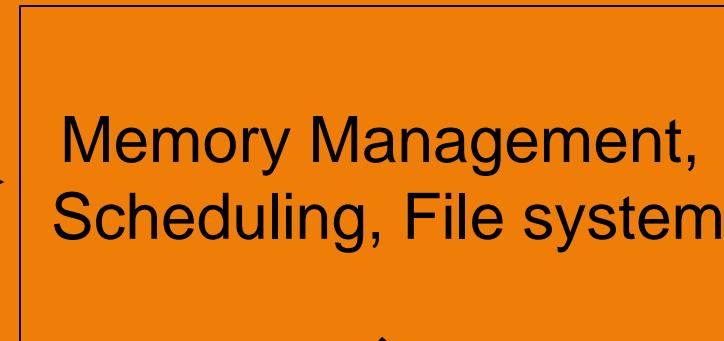
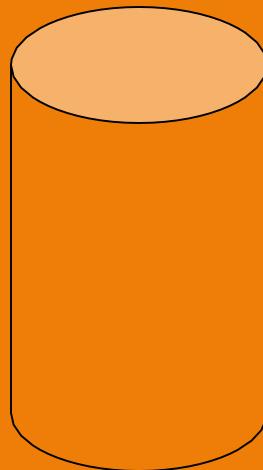
- Deciding which program should be scheduled to run next, at which computer
- What is its data requirement
- any requirement of data transfer

- Shared data resides on the server and client side
- Software called redirection software exists in the client
- System call generated in client will be handled by local operating system
- Remote Procedure call (RPC)
  - | Client to server
- Communication management software
  - | server

# Client Workstation



## Server Side



# Functions of NOS

- Redirection
  - Resides in client and server
  - Interrupts
  - If it is local processing continues
  - If remote it has to generate a request to server
  - Conversion is necessary (different OS)

- Communication management
  - Software manages communication between client and server
  - Concerned with error free transmission of message
- File/Printer Services
  - Runs on server
- Network management
  - Monitoring network and its components
  - Maintain a list of hardware and locations

# Global Operating System (GOS)



- NOS is responsible for memory and process management
- NOS converts request into task and execute it
- Memory and processing power in all other computers in the network is not tapped to the maximum by a NOS
- This is what exactly GOS attempts to do
- Memory is managed at a global level

## ■ Various functions

- User interface
- Information management
- Process/ object management
- Memory management
- Communication management
- Network management
- Part of the kernel of a GOS is duplicated at all sites
- The kernel contains software to control hardware

## □ Migration

|| Are necessary for optimal use of available resources

|| It includes

### ■ Data migration

- Send the full file
- Send only the required portion
- Replicate data

### ■ Computation migration

- A process on one node can request for execution of another process at a remote site through RPC

- Avoid data file transfer

# ■ Process migration

- Process does not complete due to the non availability of requirements
- So the process is migrated to another node for execution
- Load balancing
- Special facilities
- Reducing network load

## □ Resource Allocation / Deallocation

- Maintains a global list for all resources and allocates them to processes

- A procedure is present on the server to locate and retrieve data presents on a shared device attached to it
- It's a part of OS
- When the Clients requests for some data on the server this procedure is called remotely from the client
- Hence it is called Remote procedure call

## □ Message Passing Schemes

### ■ Uses SEND and RECEIVE

## □ Types of Services

### ■ Reliable service

- Telephone call
- Ensures that the receiver receives correctly
- Increased load on network

### ■ Unreliable service

- Postal service
- Guarantees a high probability

### ■ Message passing schemes

- Blocking - work is blocked in client until it receives data
- Non Blocking - continues without waiting

## □ Calling procedure

### General format

- CALL P(A, B)
- P is the called procedure
- A is the passed parameters
- B returned parameters
- Can be passed by value or reference
- Reference is very difficult, to share a common address space by different processors

## □ Parameter Representation

- Standard format
- conversion

## ■ Ports

- If a server provides multiple services then normally a port number is associated with each service
- Simplifies communication

# **Security and Protection**

- Major threats to security can be categorized as
  - || Passive
    - Tapping
    - Disclosure
  - || Active
    - Amendment
    - Fabrication
    - Denial
- OSI defines the elements of security in the following terms
  - || Confidentiality
  - || Integrity
  - || Availability

## □ Attacks on security

### Authentication

- Guess or steal password
- Trial and error
- System left with logged on

### I Browsing

- Going through system files (OS)

### I Invalid parameters

- Failure to validate

### I Line tapping

### I Improper access control

### I Rouge software

- Using softwares

## □ Computer worms

- Full program by itself
- Spreads to other computers over a network
- Consumes network resources
- A computer worm does not harm any other program or data
- Consumes large resources
- Operates on network
- Sends copies to all address and spread
- Safeguards
  - Prevent its creation
  - Prevent its spreading

## □ Computer Virus

- Inflicting other programs
- It cannot operate independently
- Direct harm by corrupting codes as well as data

### □ Types of viruses

- Boot sector infectors
- Memory resident infectors
- File specific infectors
- Command processor infectors
- General purpose infectors

### □ Infection methods

- Append, Replace, insert, delete, redirect

## Mode of operations

- Through mails and advertisements

## Virus Detection

- Programs check for the integrity of binary files
- A mismatch indicates a tampering

## Virus removal

- Anti virus programs
- Scan the disk for such patterns of the known virus

## Virus Prevention

- Use legal copies of software
- running of monitoring programs

## □ Security Design Principles

### □ Public design

- Penetrator will know about it

### □ Least privileges

- To processes

### □ Explicit demand

- Access rights

### □ Continuous verification

### □ Simple design

### □ User acceptance

### □ Multiple conditions

- Two passwords or two keys



## □ Authentication

- || Verifying whether a person is a legal user or not
- || Password is most common scheme
- || Stored as encrypted
- || Length of passwords play an important role
- || Salting is a technique to make it difficult to break a password
  - || It appends a random number n to the password before encryption
- || Change password at regular intervals
- || Few guesses for a login

## □ Protection Mechanism

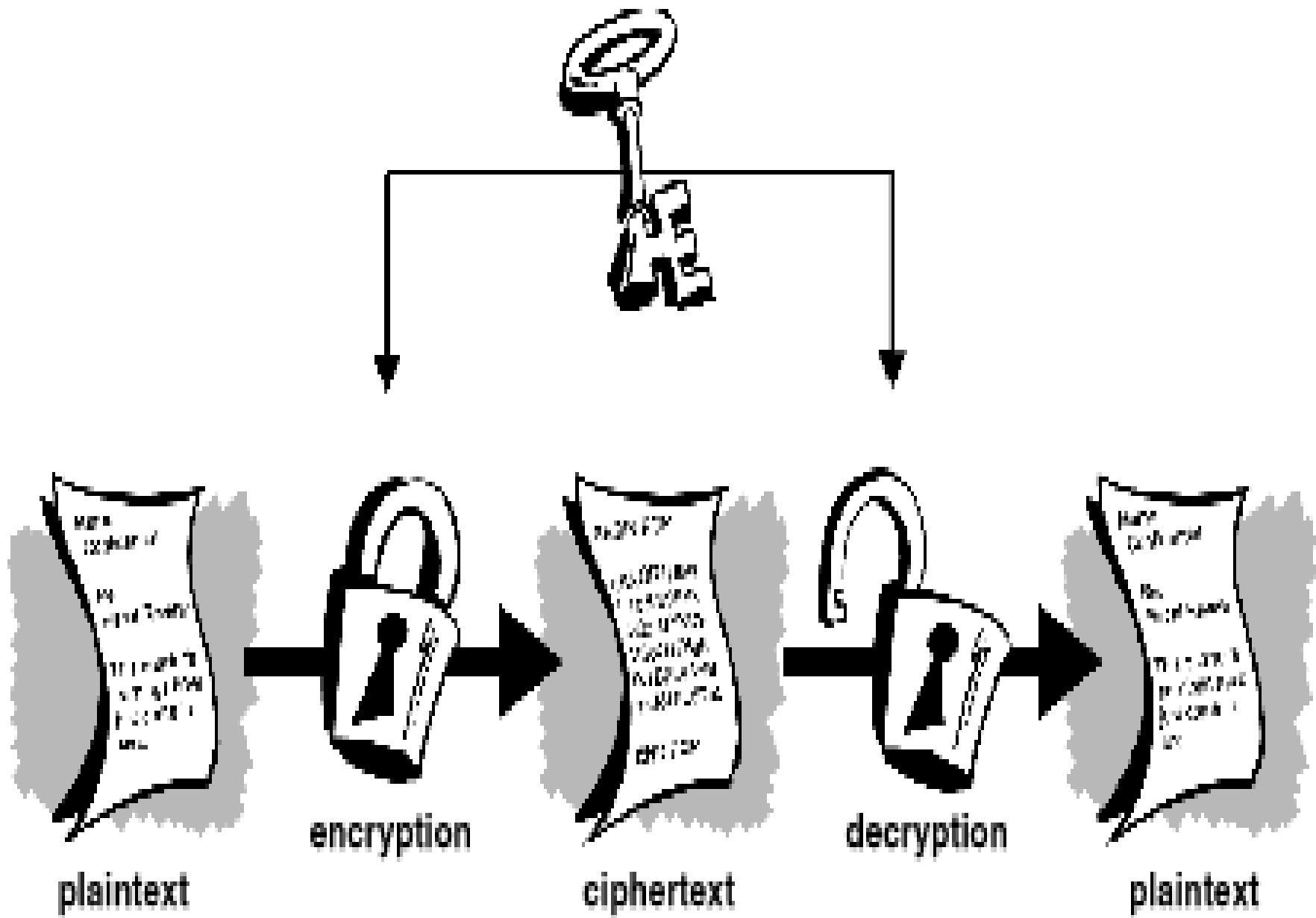
- Access rights
- Different domains (n)
- 0 domain has maximum access rights
- Domain n-1 has the least

# Encryption

- **Encryption** is the process of transforming information (referred to as plaintext) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is **encrypted** information (in cryptography, referred to as ciphertext).

# Decryption

- The process of decoding data that has been *encrypted* into a secret format. Decryption requires a secret key



# Terms in Cryptography

- Secret Writing
- Plain text or clear text
  - | Original message
- Cipher text or cryptogram
  - | Transformed message
- Explain the figure in the book
- Key
  - | 2 digits - 100, 3 - 1000 , 6 - million

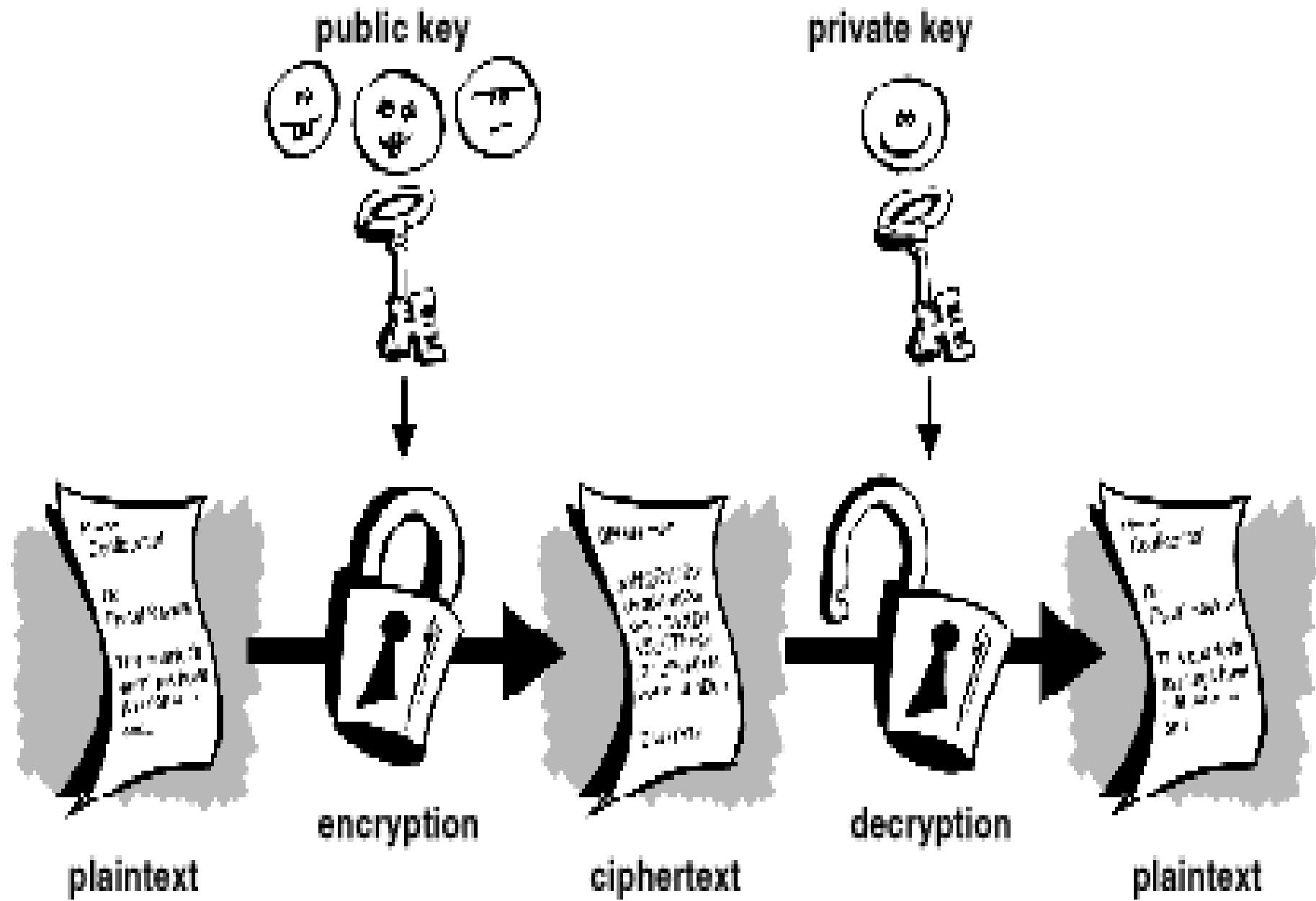
# Substitution Cipher

- Each letter or group of letters is replaced by another letter or group of letters to disguise
- Caesar cipher
  - A becomes D, B becomes E... Z becomes C
  - Example attack, dwwdfn
  - A slight generalization of the caesar cipher allows the cipher text alphabet to be shifted by K letters, instead of always 3
- Map the 26 letters to some other letters
- Mono alphabetic Substitution
  - Symbol for symbol



# Public Key Signature

- in which a user has a pair of cryptographic keys- a **public key** and a **private key**. The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message encrypted with the public key can be decrypted only with the corresponding private key.



# Multiprocessor Systems

## □ **Advantages of Multiprocessors**

- Performance and Computing Power
- Fault Tolerance
- Flexibility
- Modular growth
- Functional Specialization
- Cost/Performance

## □ **Classification of computers (Flynn)**

- SISD - single instruction single data stream
- SIMD - Single instruction multiple data stream
- MISD - Multiple instruction Single Data stream
- MIMD - Multiple instruction multiple Data stream

- Based on the relationship between processes and memory , multi processor system can be classified as
  - I Tightly Coupled
    - Share global shared memory
    - Inter process communication through shared memory
  - I Loosely Coupled
    - Access their own private/local memory
    - Message passing
    - Distributed systems fit into this class
  - I Hybrid systems have both local and global memories

- Based on memory access delays, multi processor systems are classified as



- Uniform Memory Access (UMA)

- Multiple processors can access all the available memory with the same speed

- Non Uniform Memory Access (NUMA)

- Different access times
  - Based on nearness of the complexity in switching logic

- No remote memory access (NORMA)

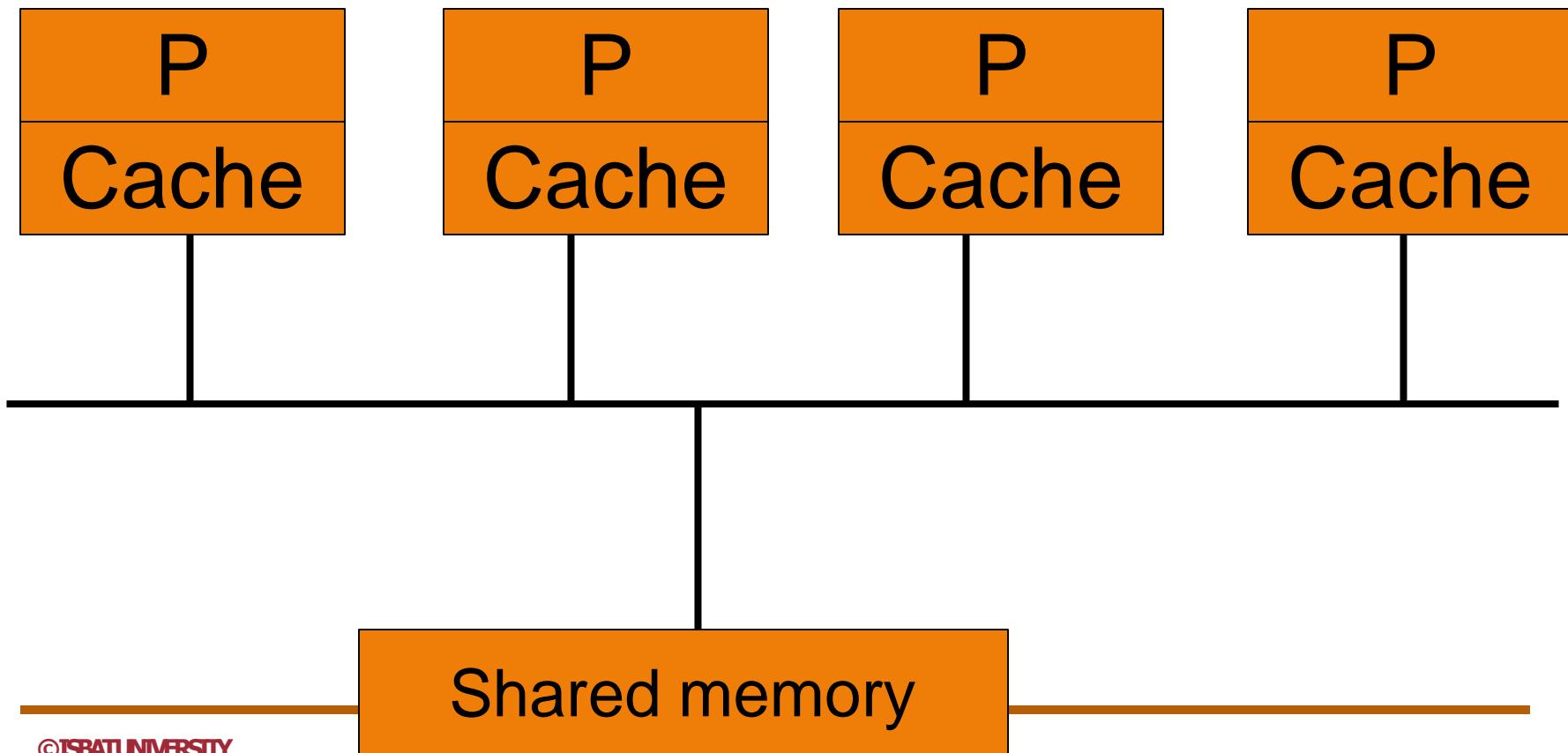
- Systems have no shared memory

## □ Multiprocessor Interconnections

- Bus Oriented Systems
- Crossbar connected systems
- Hyper Cubes
- Multistage switch based systems

# ❑ Bus Oriented Systems

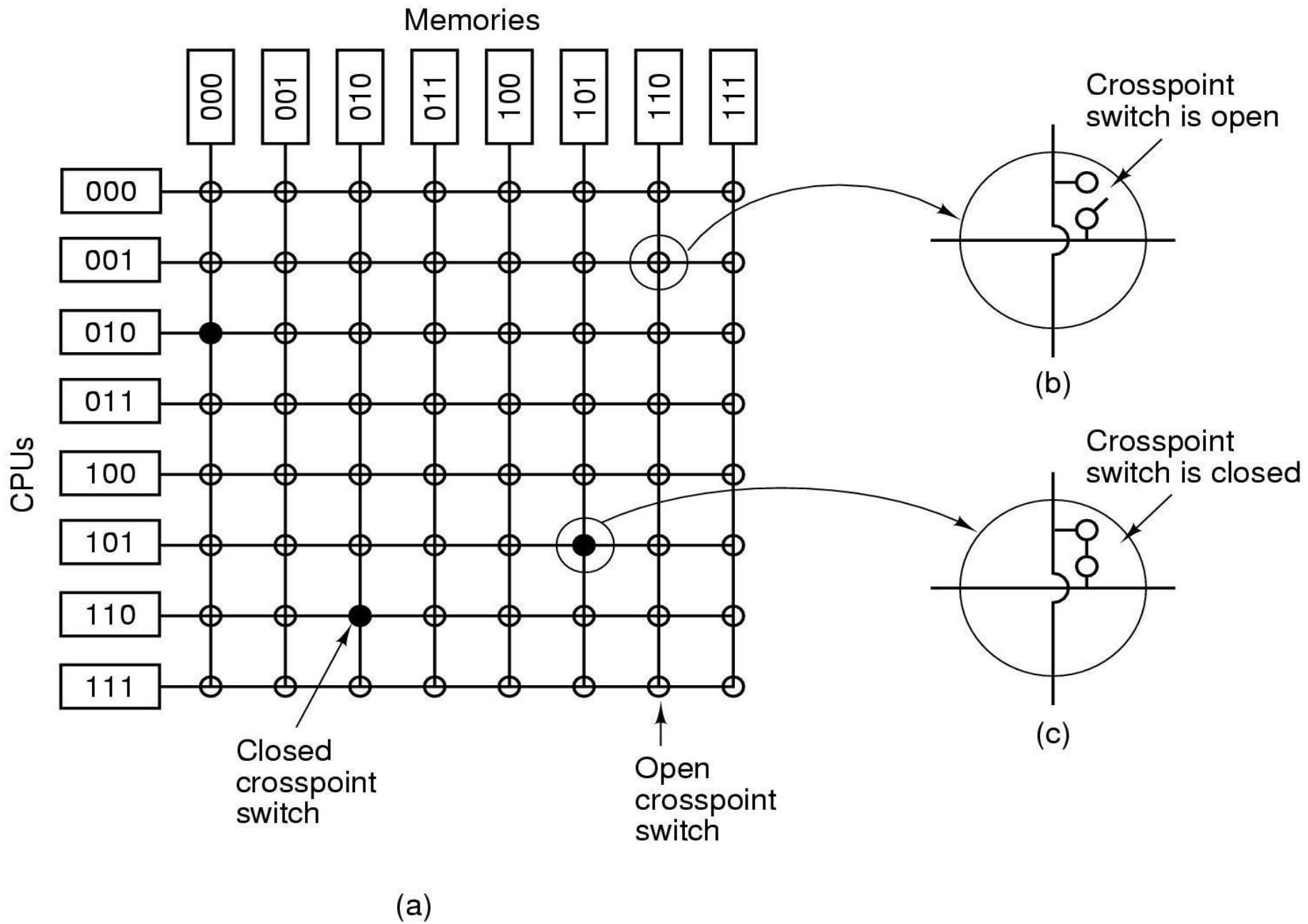
- I Shared bus connects processors and memory



- Possibility of contention in shared resource
- Better performance because of cache
- Cache coherence
  - Multiple physical copies of the same data must be consistent
  - Maintaining cache coherence increases bus traffic

## □ Crossbar connected systems

- || Simultaneous access of n processors and n memories is possible if each of the processors accesses a different memory
- || Contention can occur
  - Careful distribution of data among the different memory location can reduce or eliminate contention
- ||  $N^2$  cross points exist



## □ Hyper cubes

- || In a 3 dimensional hypercube N=8
- || Cube topology has one processor at each node/ vertex
- || Each processor at a node has a direct link to log N nodes

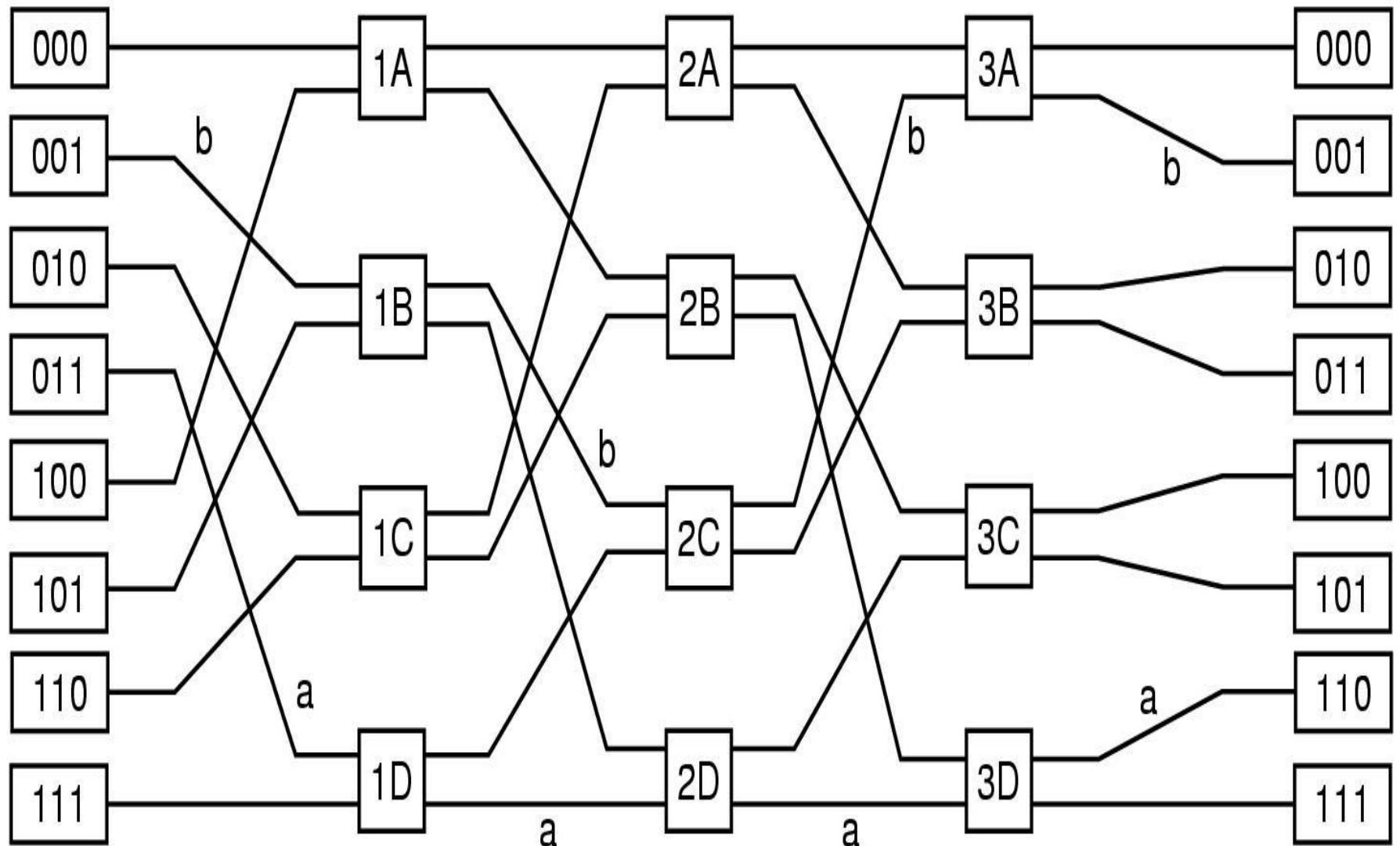
## □ Multistage switch based systems

- Each switch is a  $2 \times 2$  crossbar that can do anyone of the following
  - Copy input to output
  - Swap input and output
  - Copy input to both output
- Routing is fixed and is based on the destination address and the source address

### 3 Stages

CPUs

Memories



# Types of Multiprocessor OS

## □ Separate Supervisors

- Each node having a separate OS with memory and I/O resources
- Addition of a few services and data structure will help to support aspects of multiprocessors

## □ Master/Slave

- Master is dedicated to execute the Operating system
- Master schedule and control slaves
- Master fails entire system fails

## □ **Symmetric**

- All are functionally identical
- Memory and I/O available to all
- Operating system is also symmetric, any processor can execute it
- Floating master - the one who is executing the OS

# Multiprocessor OS functions and requirement



- Resource management
  - Processors management
  - Memory management
  - I/O devices management
- Processor scheduling
- Inter process and inter processor synchronization
- Creation and management of threads and processes

# OS Design and Implementation Issues



- Processor management and scheduling
  - Support for multiprocessing
  - Allocation of processing resources
  - Scheduling
- Memory management
  - Virtual memory
  - TLB

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

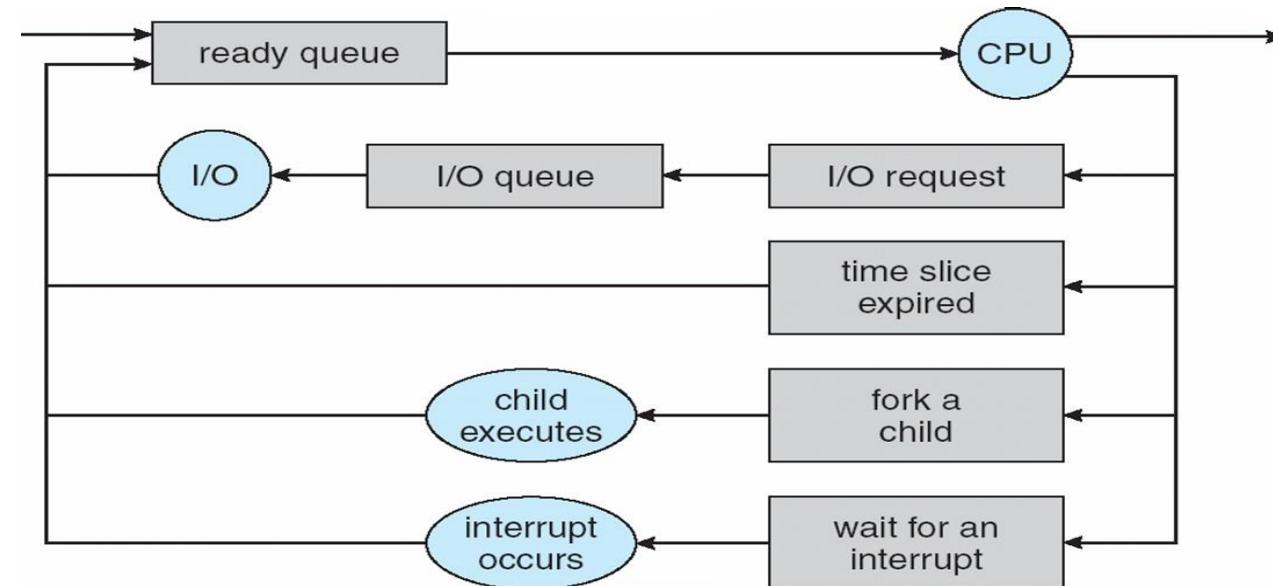
# CHAPTER 4

## CPU Scheduling

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

# Process Scheduling



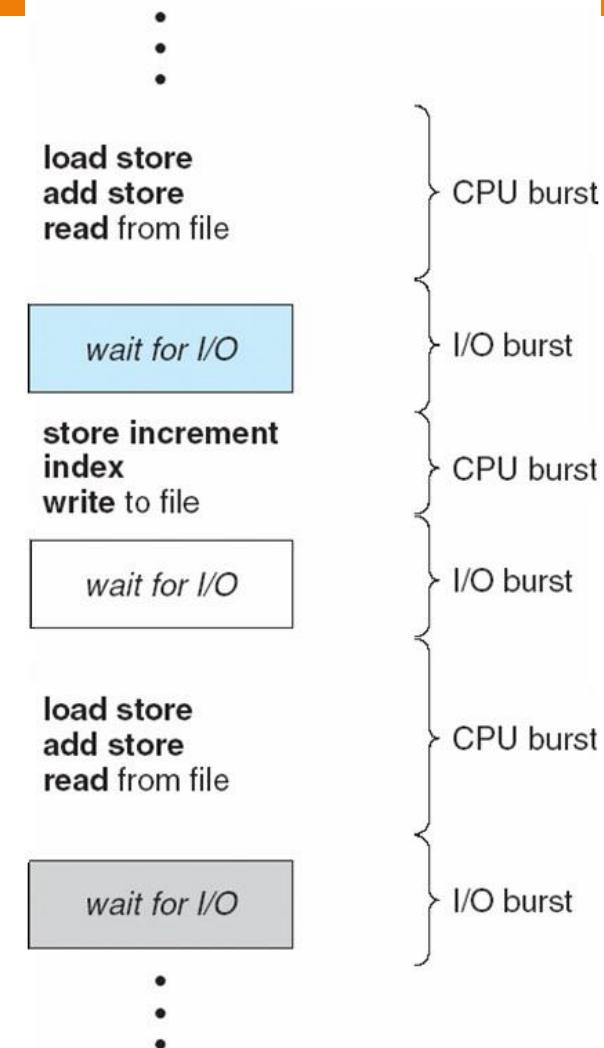
- Recall the life-cycle of a thread or process
  - A thread in its life time moves among queues.
- Question: How OS decide which threads to add to or remove from a queue?
  - CPU scheduling concerns with the Ready Queue
  - Other queues need to be scheduled as well, scheduling objectives and mechanisms are different

- Maximum CPU utilization obtained with multiprogramming
  - Under a single process system, when the process is waiting for I/O, the CPU is idle
- CPU-I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst** distribution
  - From extensive measurement, this is either exponential or hyper-exponential, with a large number short CPU bursts and a small number of long CPU bursts

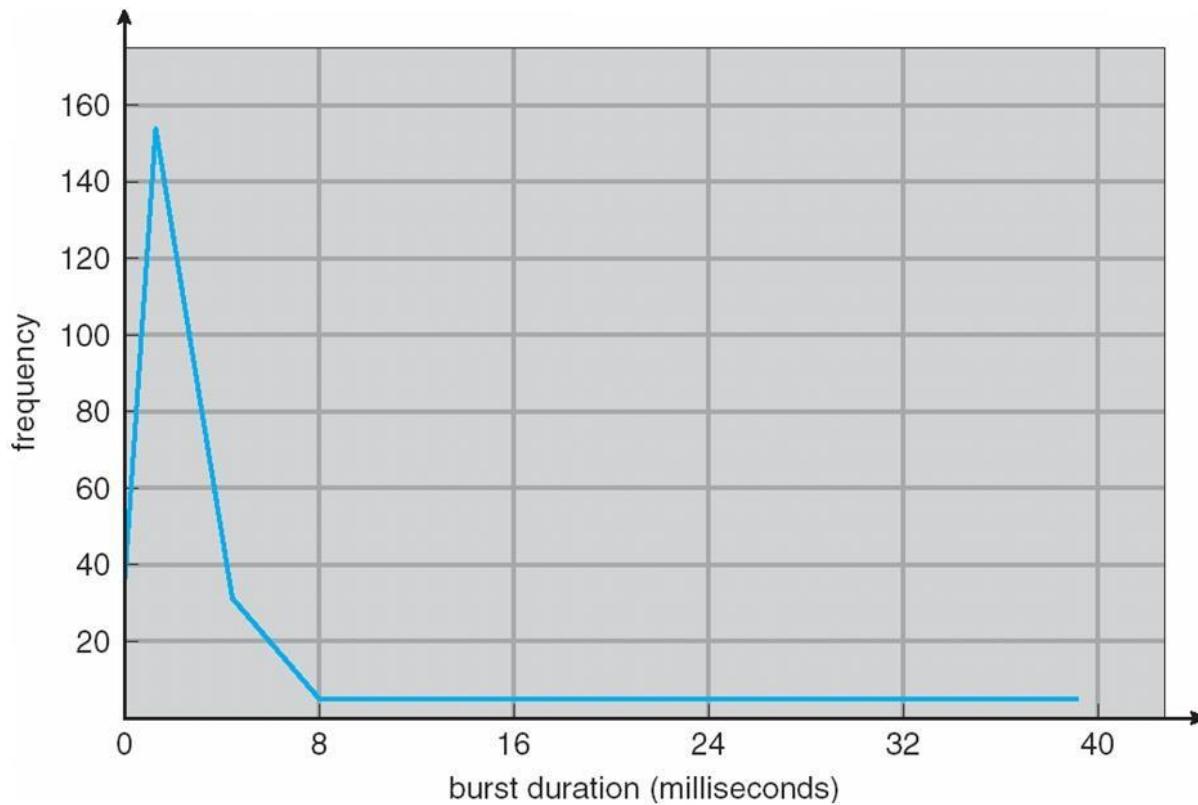
# Alternating Sequence of CPU And I/O Bursts

- Execution model: programs alternate between bursts of CPU and I/O

- CPU-I/O Burst Cycle: a program typically uses the CPU for some time, then does I/O, then uses CPU again
- Each scheduling decision is about which job to give to the CPU for use by its next CPU burst
- With time-slicing, thread may be forced to give up CPU before finishing current CPU burst



# Histogram of CPU-burst Times



# CPU Scheduler and Dispatcher

- Selects from among the processes in memory that are ready to execute, 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
- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency (overhead) – time it takes for the dispatcher to stop one process and start another running

# Factors Related to CPU Scheduling

- There could be factors affecting the design of CPUscheduling, some are conflicting
  - CPUutilization - keep the CPU as busy as possible
  - Throughput- # of processes or jobs that complete their execution per time unit
  - Turnaround time - the total amount of time to execute a particular process
  - Waiting time - the cumulative 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, and interactive job)

# Scheduling Criteria

- Maximize CPU utilization
  - CPU is not idle unless no jobs on the ready queue
- Minimize Response Time
  - Minimize elapsed time to do an operation (or job)
  - Response time is what the user sees:
    - Time to echo a keystroke in editor
    - Time to compile a program
    - Real-time tasks: Must meet deadlines imposed
- Maximize Throughput
  - Maximize operations (or jobs) per second
  - Throughput related to response time, but **not identical**:
    - Minimizing response time could lead to more context switching
  - Two parts to maximizing throughput
    - Minimize overhead (for example, context-switching)
    - Efficient use of resources (CPU, disk, memory, etc)
- Fairness
  - Share CPU among users in some “fair” way, no single job hogging



# Basic Concepts of scheduling

- CPU -I/O Burst Cycled
  - Process execution consists of alternate CPU execution and I/O wait
  - A cycle of these two events repeats till the process complete execution
  - Execution begins with CPU burst followed by an I/O burst and so on
  - A CPU burst terminate the execution
  - An I/O bound job will be having a short CPU burst and CPU bound will have long CPU burst

## □ CPU Scheduling

- | Done by short term scheduler
- | Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- | Ready queue may be implemented as a FIFO queue, priority queue, a tree or simply an unordered linked list
- | The records in the queue are generally PCB's of the processes

## □ Preemptive/ Non preemptive scheduling

- CPU scheduling decisions may take place under the following four circumstances
  - 1. Switches from run to wait
  - 2. Switches from run to ready state
  - 3. Switches from wait to ready state
  - 4. Terminates

- I scheduling under condition 1 & 4 is said to be non preemptive
  - CPU executes until the CPU is released
- I Scheduling under condition 2 & 3 is said to be preemptive
  - Stops and go for ready queue to make the CPU available for another process

## □ Dispatcher

- | Component involved in the CPU Scheduling function
- | Module that gives control of the CPU to the process selected by the short term scheduler
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart
- | It has to be fast
- | The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency

## □ Scheduling Criteria

### □ CPU Utilization

### □ Throughput

- Number of process completed per time unit

### □ Turnaround time

- The interval of time between submission and completion ( execution time + waiting time)

### □ Waiting time

- Sum of all the time spent at different instances, waiting in the ready queue

### □ Response Time

- Difference between time of submission and the time the first response occurs

# Optimization Criteria



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

# Scheduling Algorithms

# First-Come, First-Served (FCFS) Scheduling

<u>Process</u>	<u>Burst Time</u>
$P_1$	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:  $(0 + 24 + 27)/3 = 17$

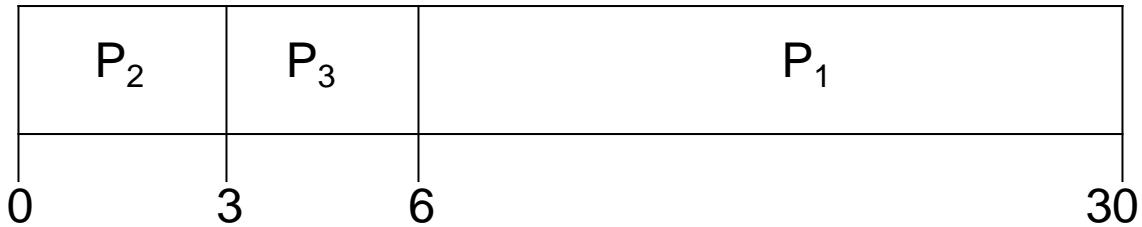
## Process    Burst Time

$P_1$	24
$P_2$	3
$P_3$	3

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 (one big CPU-bound process may force other processes to wait for completion of its CPUburst if this big process arrives first)
- Non - Preemptive
- Hence, execution of short processes prior to a long process is desirable

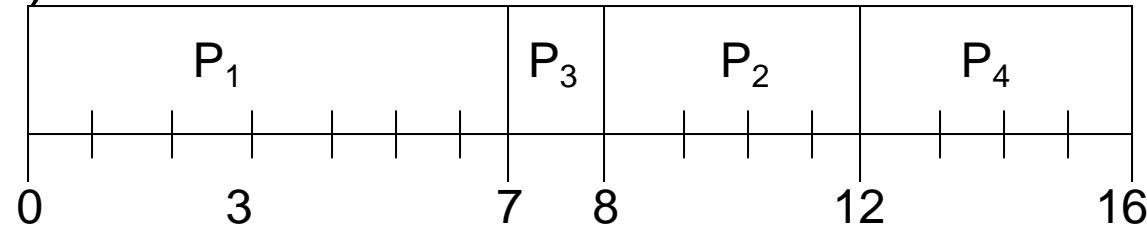
# Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time (the smallest next CPU burst).
- **Two schemes:**
  - nonpreemptive - 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, it can preempt the current process. This scheme is known as the **Shortest-Remaining-Time-First (SRTF)**
- **SJF is optimal** if it gives minimum average waiting time for a given set of processes

# Example of Non-Preemptive SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

- SJF (non-preemptive)



- $p1 \quad p2 \quad p3 \quad p4$
- Average waiting time =  $(0 + (8-2) + (7-4) + (12-5))/4$

# ProcessArrival TimeBurst Time

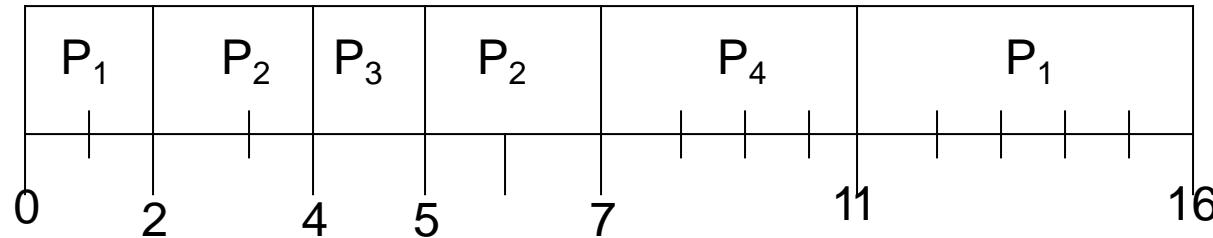
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

# Example of Preemptive SJF

## Shortest-Remaining-Time-First (SRTF)

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

□ SJF (preemptive)



- p1    p2    p3    p4
- Average waiting time =  $(9 + 1 + 0 + 2)/4 = 3$

# 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 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

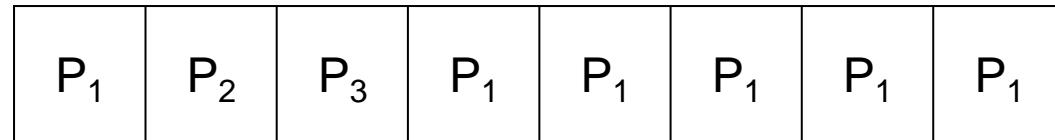
## Round Robin (RR)

- 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.

## Example of RR with Time Quantum = 4

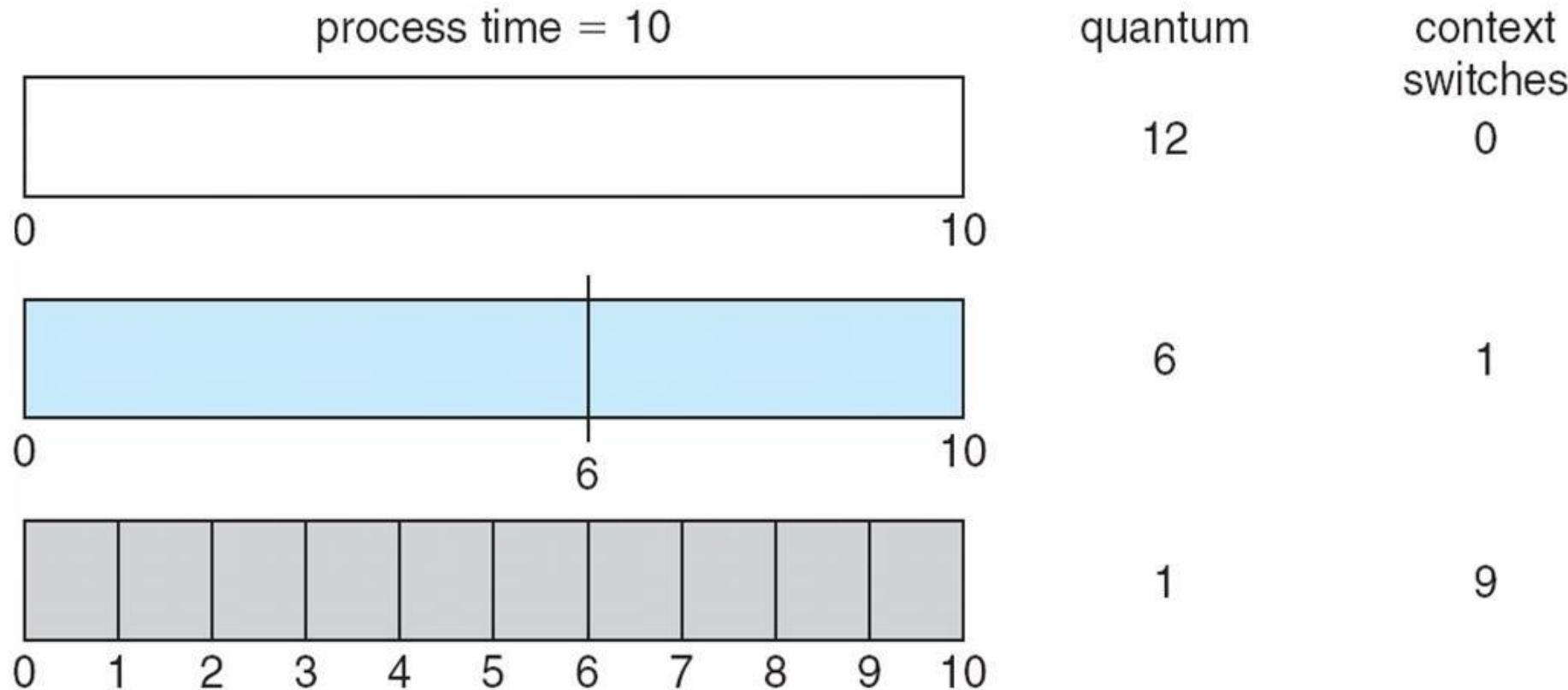
<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

The Gantt chart is:



Typically, higher average turnaround than SJF, but better response

# Time Quantum and Context Switch Time



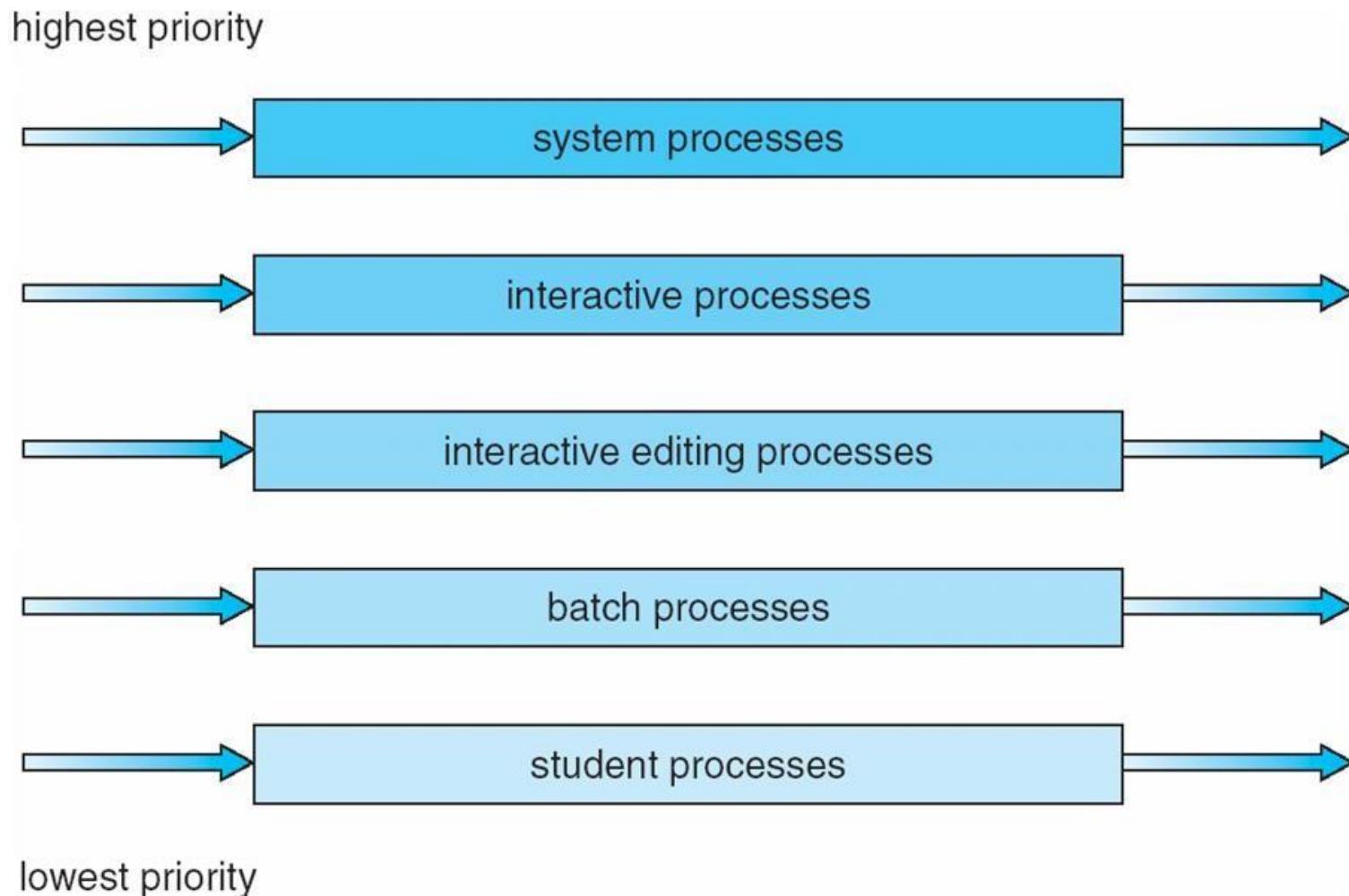
The time slice should be large with respect to the context switch time, else if RR scheduling is used the CPU will spend more time in context switching

# Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm
  - foreground - RR
  - background - FCFS

- Scheduling must be done between the queues
  - | Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - | Time slice - each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - | 20% to background in FCFS

# Multilevel Queue Scheduling



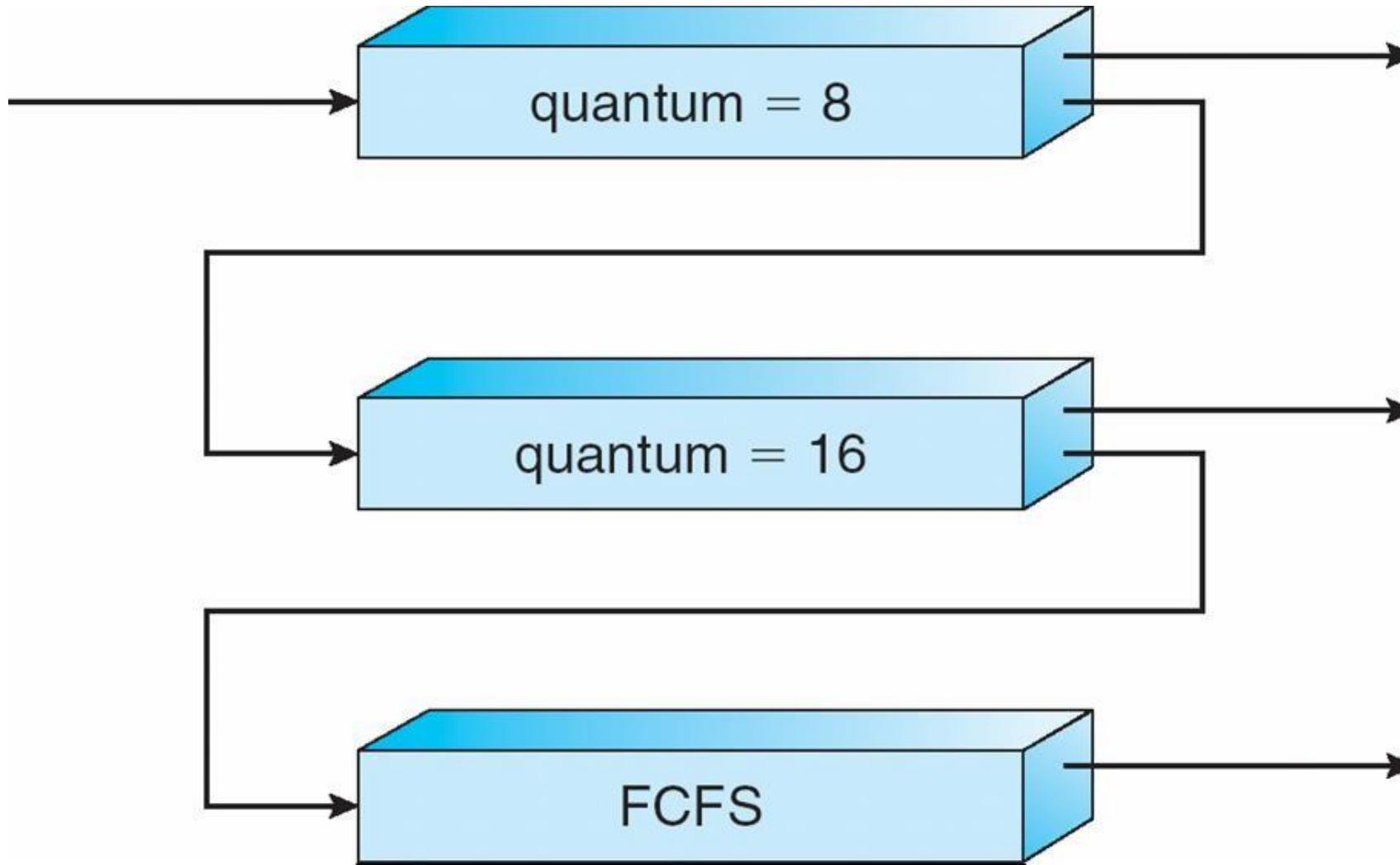
## Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

# Multilevel Feedback Queue

- Three queues:
  - $Q_0$  - RR with time quantum 8 milliseconds
  - $Q_1$  - RR time quantum 16 milliseconds
  - $Q_2$  - FCFS
- Scheduling
  - A new job enters queue  $Q_0$ , which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue  $Q_1$ .
  - At  $Q_1$  job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue  $Q_2$ .

# Multilevel Feedback Queues



# Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Heterogeneous** - different kind of CPU's
- **Homogeneous** - same kind of CPU
  - Separate queues for CPU's
  - Processor can be ideal
  - Use a common queue
  - Each processor is self scheduling
  - Ensure that two processor do not go for the same process

- One processor will be appointed for scheduling the process to other processors
- Master slave relationship
- **Asymmetric multiprocessing** - only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** - each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes

## □ Real Time Scheduling

### I Hard Real Time

- Complete within a guaranteed amount of time
- Process submitted with time
- Scheduler can accept or reject the process
- It's called as resource reservation

### I Soft Real Time

- Less restrictive
- Priority scheduling
- Real time process will be having the highest priority
- Priority will not degrade
- Dispatch latency must be small

## □ Evaluation of CPU Scheduling Algorithms

- Maximizes CPU Utilization under the constraint that the maximum response is 1 second
- Maximize throughput such that turnaround time is linearly proportional to total execution time
- Deterministic Modeling
  - If the specific workload is known then the exact values for major criteria can be fairly easily calculated, and the best is determined
- Queuing Model
  - Rate at which new process arrives, the ratio of CPU burst times to I/O times, the distribution of CPU Burst time and I/O burst times

- Calculate certain performance characteristics of waiting queue
- Little's formula
- $N = \text{Lambda} * W$ 
  - N : average queue length
  - W : average waiting time
  - Lambda : average arrival of new jobs

## Simulations

- Trace steps
  - Performance of a real time system under typical expected workload

## I Implementation

- The only way to determine how a proposed scheduling algorithm is going to operator is to implement it on a real system

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

## CHAPTER 5

### Threads & Concurrency

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

- Overview
- Multi-core Programming
- Multi-threading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

- A **thread in computer science** is short for a **thread** of execution. **Threads** are a way for a program to divide (termed "split") itself into two or more simultaneously (or pseudo-simultaneously) running tasks.
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

# Objectives

4



- Identify the basic components of a thread, and contrast threads and processes
- Describe the benefits and challenges of designing multi-threaded applications
- Illustrate different approaches to implicit threading, including thread pools and fork-join
- Describe how the Linux operating system represents threads
- Explore multi-threaded applications using the Pthreads, Java, and Windows threading APIs

# Motivation

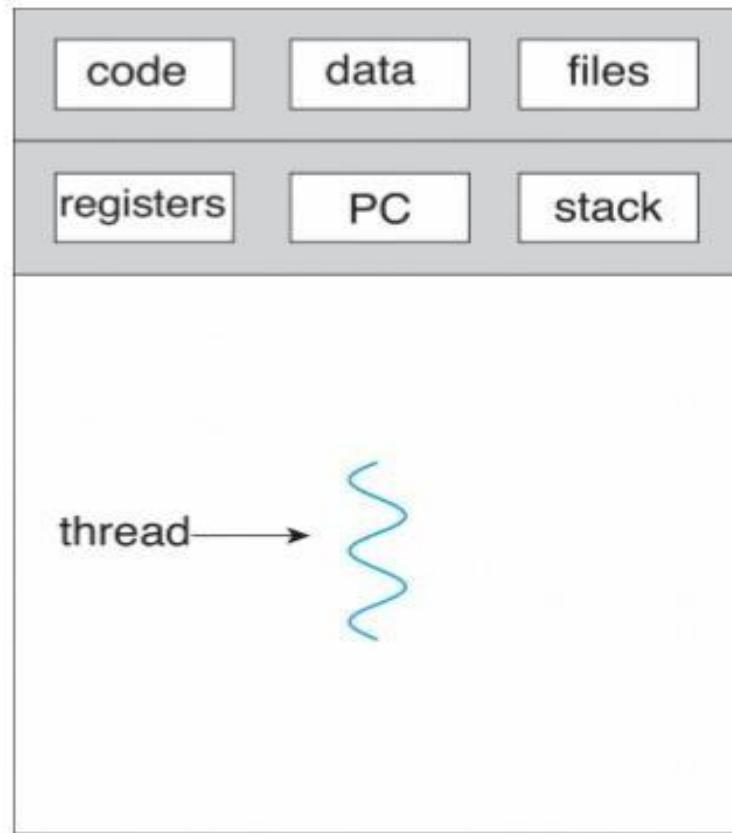
5



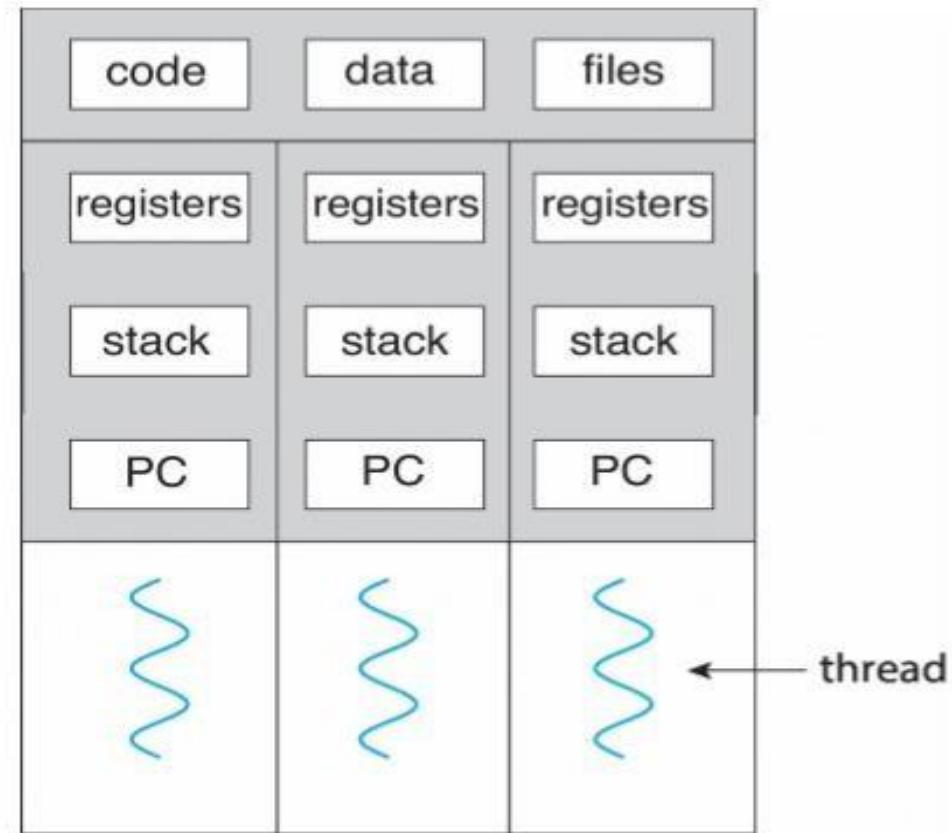
- Kernels are generally multi-threaded
- Most modern applications are multi-threaded
- Whereas cooperating processes are independent, cooperating threads run within the same process (think application)
- Multiple functions or tasks within an application can be implemented by separate threads. Example decomposition:
  - A thread to update display
  - A thread to fetch data from a database
  - A thread to run a tool such as a spell-checker
  - A thread to respond to a network request
- Process creation is costly and slow, whereas thread creation is light-weight
- Proper threading can simplify code, increase efficiency

Activate \\\  
Go to Settings

# Single versus Multi-threaded Processes



single-threaded process



multithreaded process

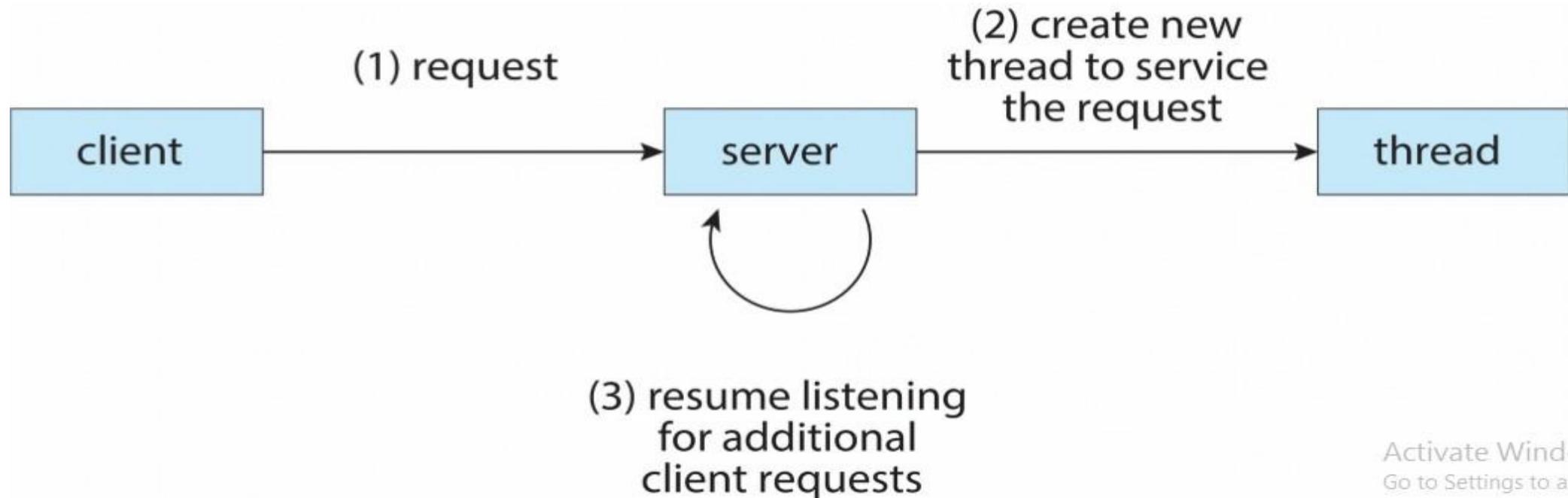
***Each thread has its own register set, stack, and PC.***

# Multi-threaded Server Architecture

7



- 1) client sends request to server;
- 2) server creates a thread to process the request, and
- 3) immediately returns to listening for the next request from a client in the same main thread.



Activate Windc  
Go to Settings to ac

# Benefits of Threads

8



- **Responsiveness** – may allow continued execution if part of process is blocked, or some slow operation in a different thread - especially important for user interfaces
- **Resource Sharing** – threads share same address space in single process, easier than processes using shared memory or message passing
- **Cost** – cheaper than process creation, thread switching lower overhead than context switching
- **Scalability** – process can take advantage of multi-core architectures

Activate Window

# Concurrency vs. Parallelism

9

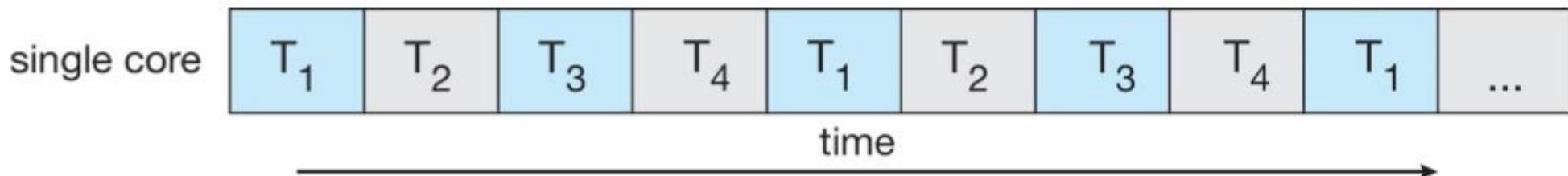
- Two or more sequences of instructions are said to be **concurrent** if no matter what order they are executed in relation to each other, the final result of their combined computation is the same.
- This means that they can be executed simultaneously on different processors, or interleaved on a single processor in any order, and whatever outputs they produce will be the same.
- A system with two or more concurrent processes is called a **concurrent program** or a **concurrent system**.
- Two processes or threads execute **in parallel** if they execute at the same time on different processors.
- **Parallel programs** are those containing instruction sequences that can be executed in parallel. A parallel program is always a concurrent program, but a system can have concurrency even though it is not a parallel program.

Activate Window

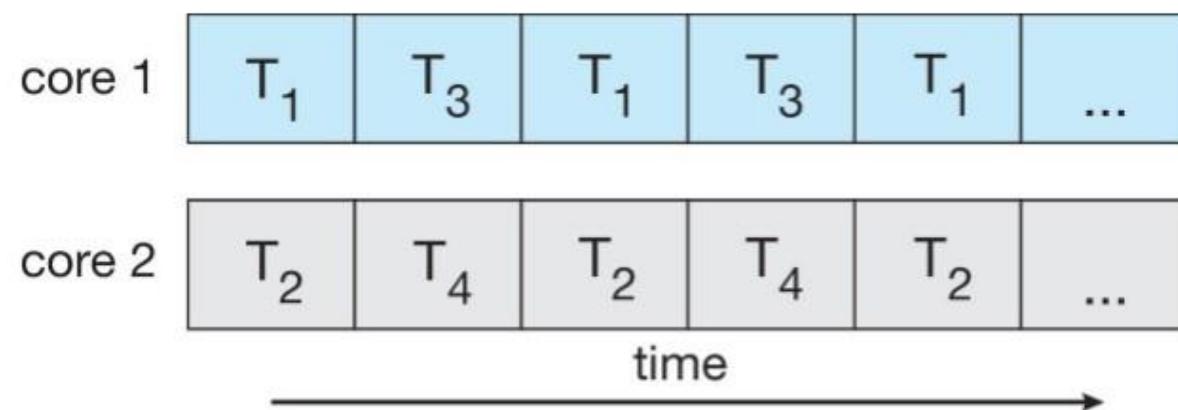
# Concurrency vs. Parallelism

10

## ■ Concurrent execution on single-core system:



## ■ Parallelism on a multi-core system:



Activate Windows  
Go to Settings to activa

# Multi-core Programming

11

- **Multi-core** or **multi-processor** systems challenge programmers to take advantage of hardware, but it is not easy:
  - How to decompose a single task into many independent parallel tasks
  - How to load-balance the tasks
  - How to split data onto separate cores/processors
  - How to identify data dependency and handle synchronization
  - How to test and debug parallel programs

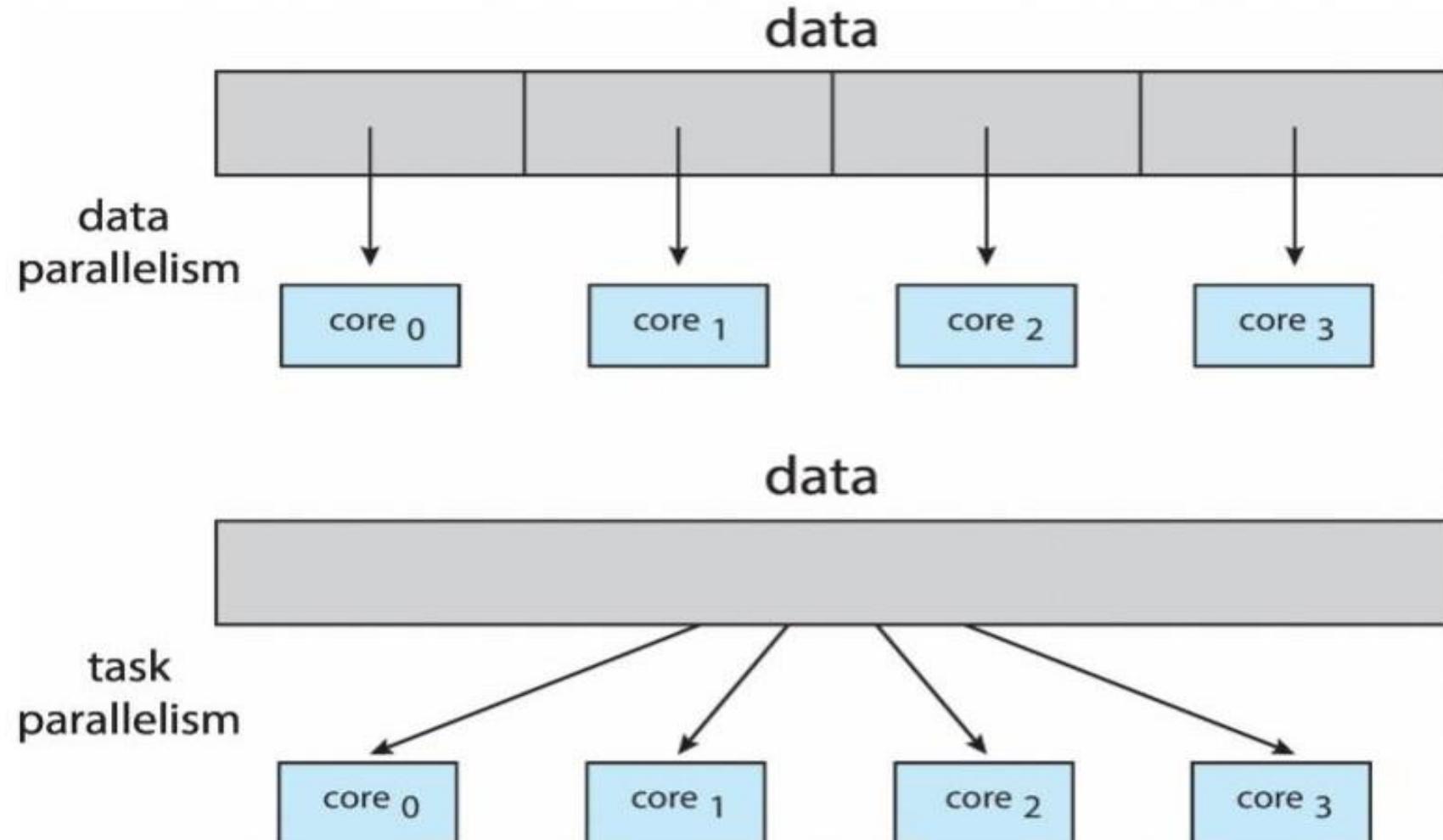
# Multi-core Programming (cont.)

12

- Types of inherent parallelism:
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
    - ▶ an image on which the same operation is applied to all pixels
    - ▶ a payroll with taxes to be calculated for all individuals
    - ▶ a set of points to be rotated through same angle in space
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
    - ▶ same data set evaluated by multiple algorithms for some property (census data analyzed for demographics, financials, geographic, etc)

# Data and Task Parallelism

13



# Amdahl's Law

14

- In 1967, Gene Amdahl argued that there was an inherent limitation to the amount of speedup that could be obtained by performing a computation using more processors. His argument is known as "Amdahl's Law". If
  - $S$ ,  $0 \leq S \leq 1$ , is the fraction of operations that must be executed serially (in sequence), and
  - $N$  is the number of processing cores, then the speed-up is bounded above:

$$\text{speedup} \leq \frac{1}{S + \frac{(1-S)}{N}}$$

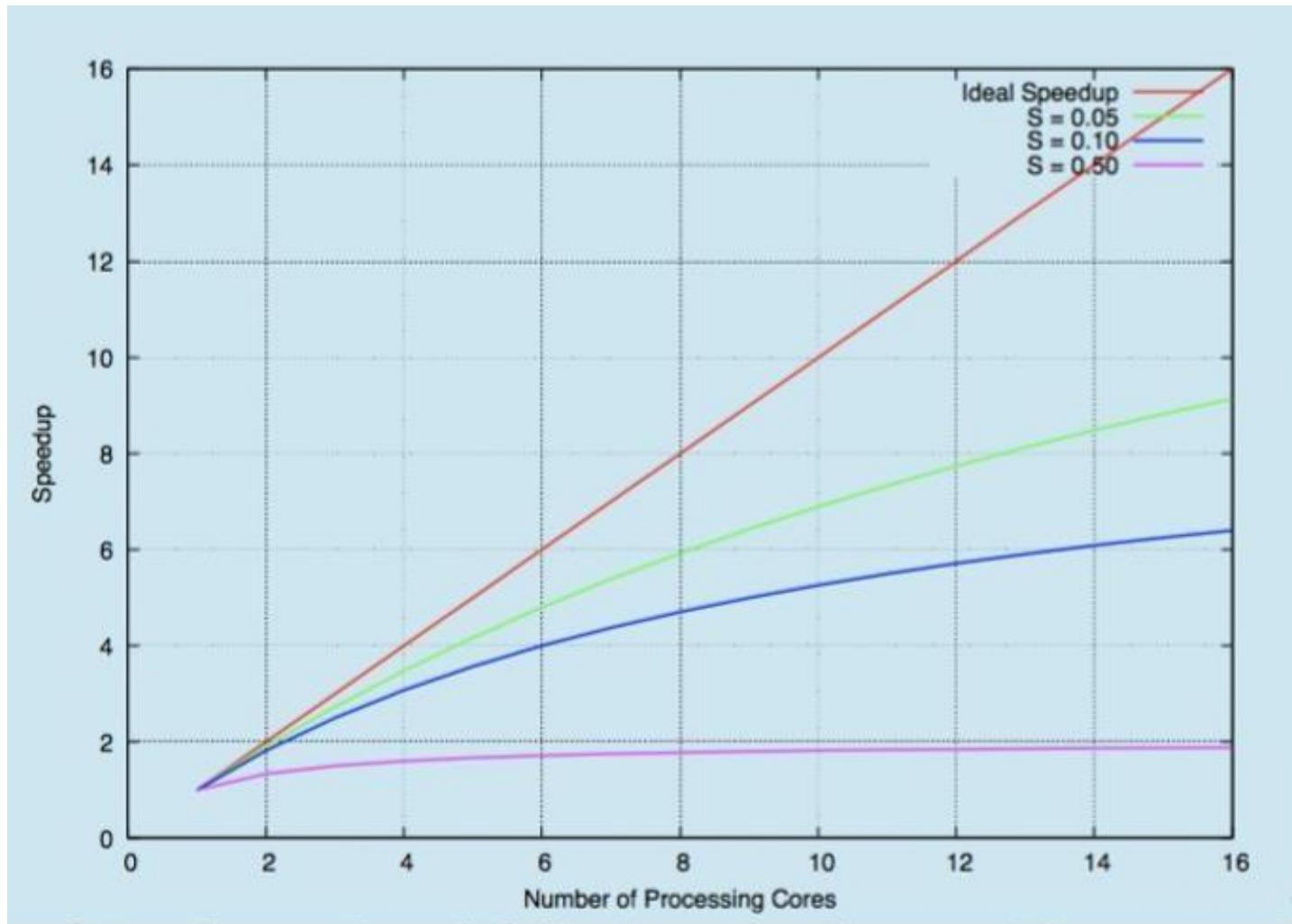
- Example: if program is 75% parallel / 25% serial, ( $S=0.25$ ) moving from 1 to 2 cores ( $N=2$ ) results in speedup of  $1/((1/4) + (3/4)/2) = 1.6$
- As  $N$  approaches infinity, speedup approaches  $1 / S$

**Serial portion of an application limits maximum performance gained by adding additional cores**

Activate Window  
Go to Settings to act

# Amdahl's Law Graphically

15



# User Threads and Kernel Threads

16

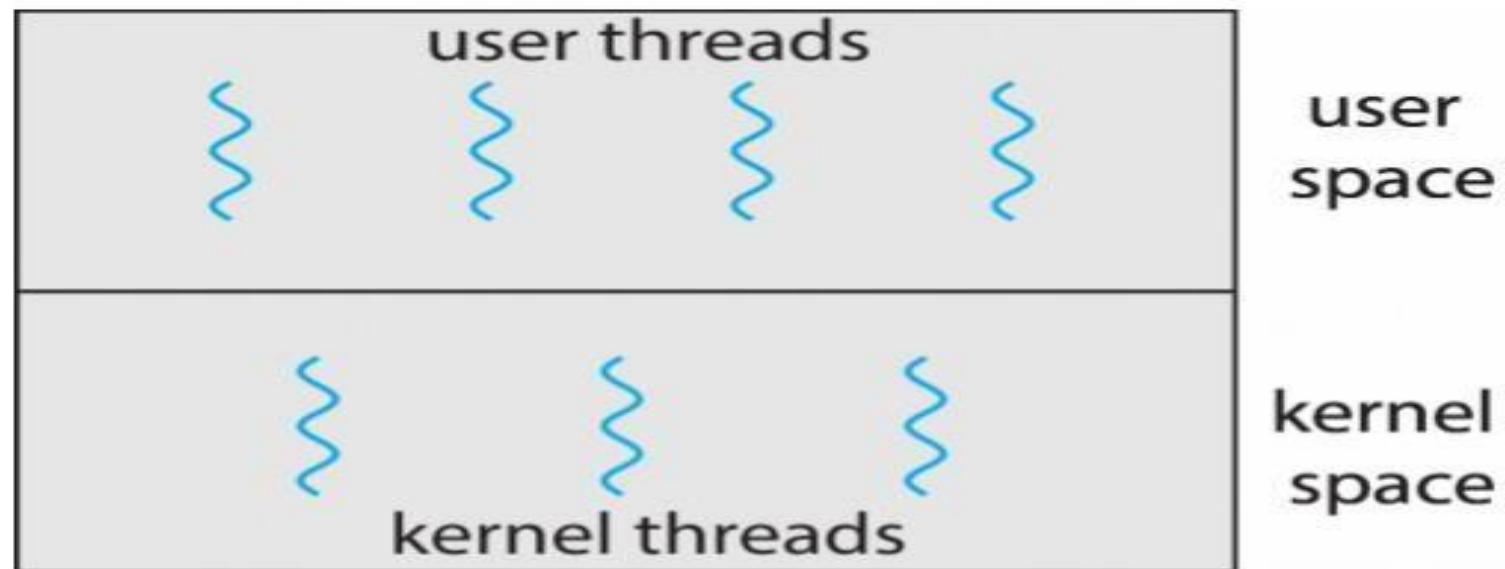
- **User threads** are supported by user-level libraries
- Three primary user thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** are supported directly by the kernel
  - Examples – virtually all modern operating systems, including:
    - ▶ Windows
    - ▶ Linux
    - ▶ Mac OS X
    - ▶ iOS
    - ▶ Android

Act  
Go to

# User and Kernel Threads

17

- When threads are provided as user threads, they still must be mapped onto kernel threads.
- There is not necessarily an equal number of user and kernel threads.



# Multi-threading Models

18

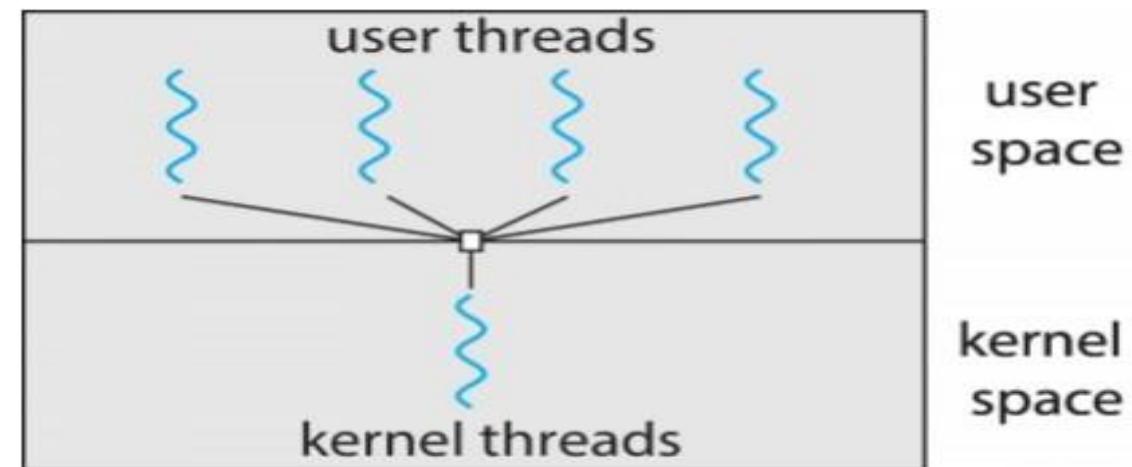


- How to map user threads to kernel threads?
- Three different models:
  - Many-to-One: many user-level threads map to single kernel thread
  - One-to-One: one user-level thread maps to one kernel thread
  - Many-to-Many: many user-level threads map to many kernel threads

# Many-to-One

19

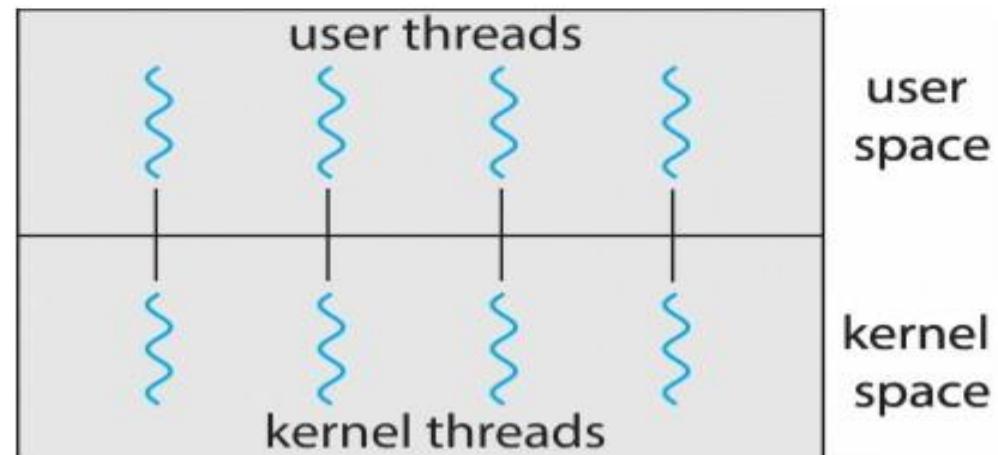
- Many user-level threads mapped to single kernel thread.
- Weaknesses:
  - One thread blocking causes all to block
  - Multiple threads may not run in parallel on multi-core system because only one may be in kernel at a time
- Few systems currently use this model because modern systems have many cores which are not utilized well.
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



# One-to-One

20

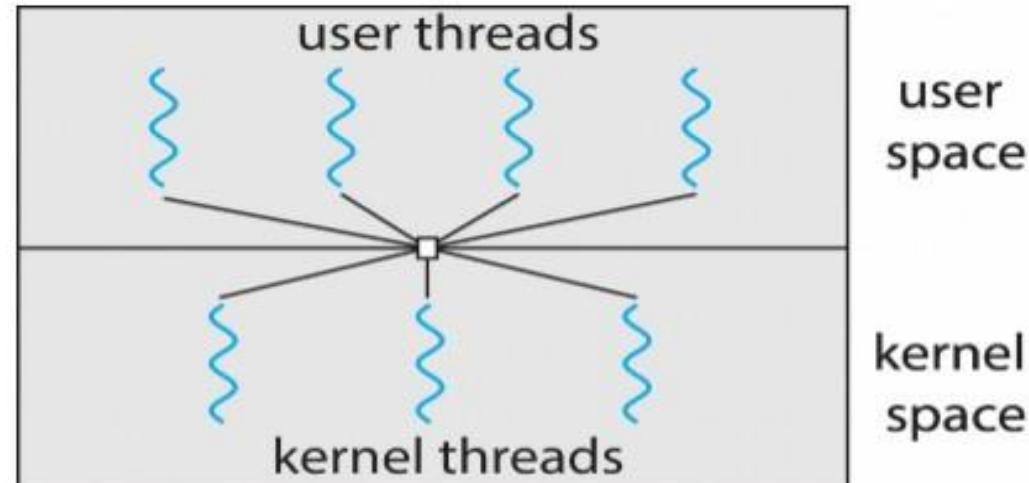
- Each user-level thread maps to one kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead:
  - Creating a user thread requires creating a kernel thread, and too many kernel threads can degrade performance of system.
- Examples
  - Windows
  - Linux



# Many-to-Many Model

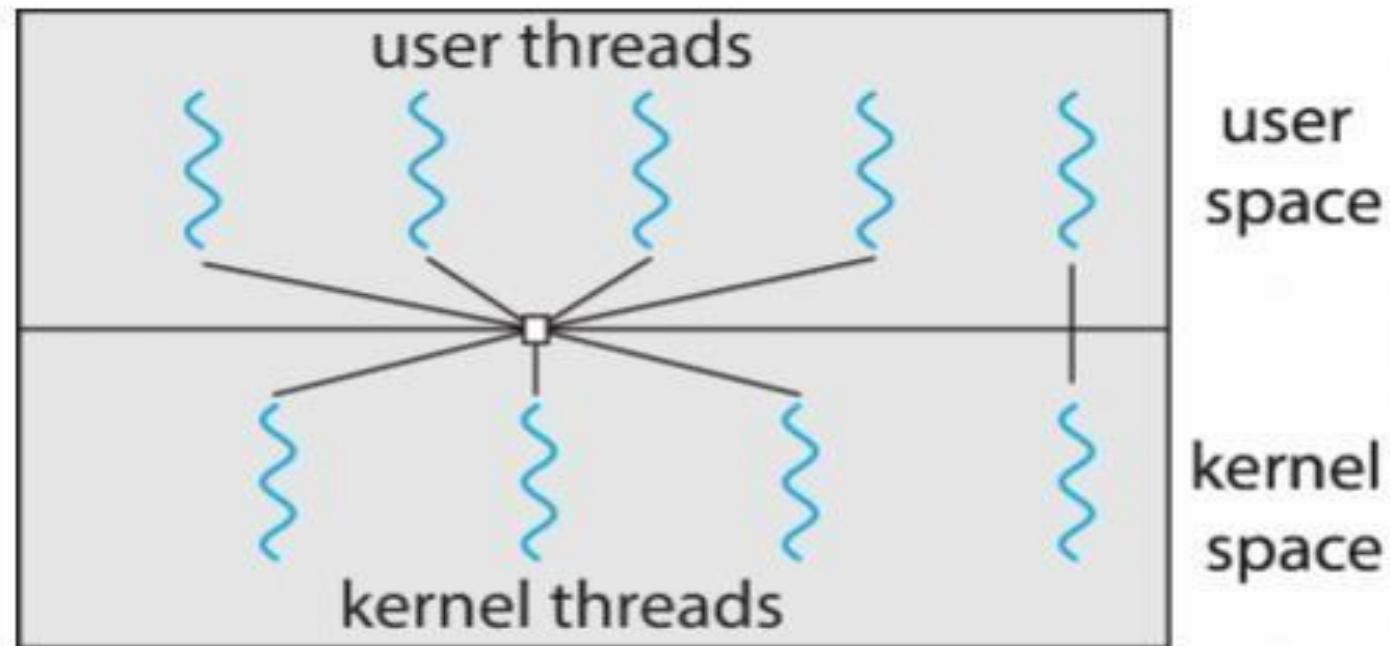
21

- Allows many user level threads to be multiplexed onto an equal or smaller number of kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Program can have as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor. If thread blocks, kernel can schedule a different thread.
- Windows with the ThreadFiber package
- Otherwise not very common



# Two-level Model

- Similar to the many-to-many, except that it allows a user thread to be **bound** to a kernel thread.



# Thread Libraries

23

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS
- Three prevalent libraries: POSIX threads (Pthreads), Windows, and Java threads.

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

# Pthreads Example 1

25

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void* hello_world( void* unused)
{
    printf("The child says, \"Hello world!\"\n");
    pthread_exit(NULL) ;
}

int main( int argc, char *argv[])
{
    pthread_t child_thread;

    /* Create the thread and launch it. */
    if ( 0 != pthread_create(&child_thread, NULL,
                            hello_world, NULL ) ){
        printf("pthread_create failed.\n");
        exit(1);
    }
    printf("This is the parent thread.\n");
    /* Wait for the child thread to terminate. */
    pthread_join(child_thread, NULL);
    return 0;
}
```

# Implicit Threading

26

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Five methods explored
  - Thread Pools
  - Fork-Join
  - OpenMP
  - Grand Central Dispatch
  - Intel Threading Building Blocks

# Thread Pools

27

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - ▶ i.e. Tasks could be scheduled to run periodically
- Windows API supports thread pools:

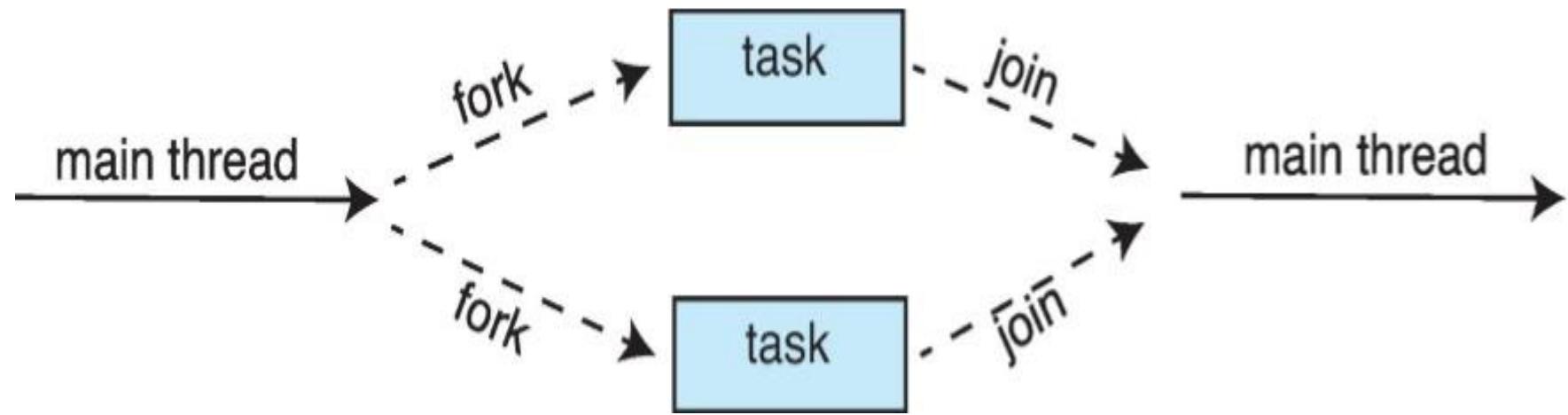
```
DWORD WINAPI PoolFunction(VOID Param) {  
    /*  
     * this function runs as a separate thread.  
    */  
}
```

# Fork-Join Parallelism

28



- Multiple threads (tasks) are **forked**, and then **joined**.



# Fork-Join Parallelism

29

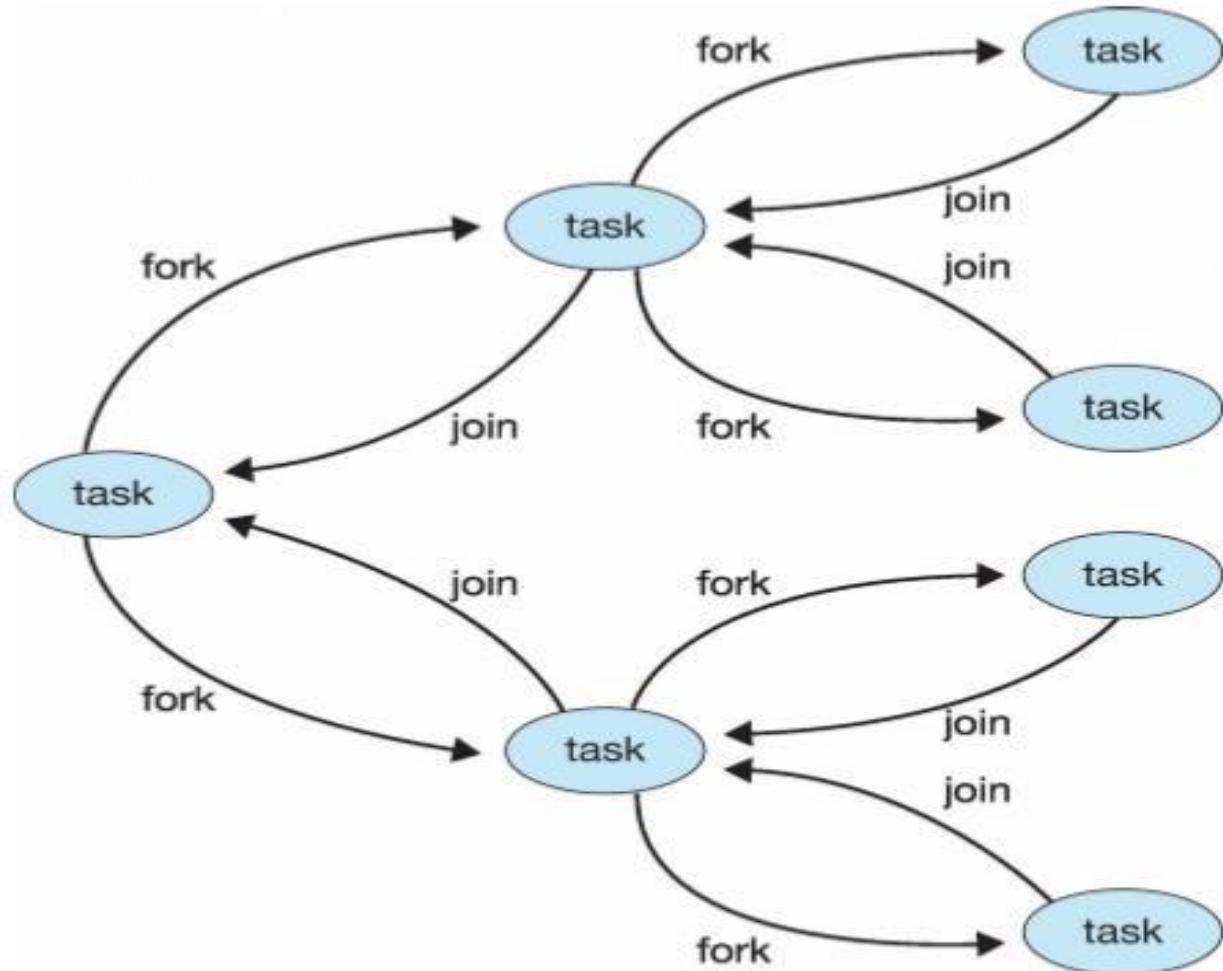
■ General algorithm for fork-join strategy:

```
Task(problem)
    if problem is small enough
        solve the problem directly
    else
        subtask1 = fork(new Task(subset of problem))
        subtask2 = fork(new Task(subset of problem))

        result1 = join(subtask1)
        result2 = join(subtask2)

    return combined results
```

# Fork-Join Parallelism



# Semantics of fork() and exec()

31

- Does **fork( )** duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- **exec( )** usually works as normal – replace the running process including all threads

# Signal Handling

32

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled by one of two signal handlers:
    1. default
    2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
  - **User-defined signal handler** can override default
  - For single-threaded, signal delivered to process

# Signal Handling (Cont.)

33

- Where should a signal be delivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

# Thread Cancellation

34

- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- Pthread code to create and cancel a thread:

```
pthread_t tid;  
  
/* create the thread */  
pthread_create(&tid, 0, worker, NULL);  
  
.  
. . .  
  
/* cancel the thread */  
pthread_cancel(tid);  
  
/* wait for the thread to terminate */  
pthread_join(tid, NULL);
```

# Thread Cancellation (Cont.)

35

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	—
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
  - Cancellation only occurs when thread reaches **cancellation point**
    - I.e. **pthread\_testcancel()**
    - Then **cleanup handler** is invoked
- On Linux systems, thread cancellation is handled through signals

# Thread-Local Storage

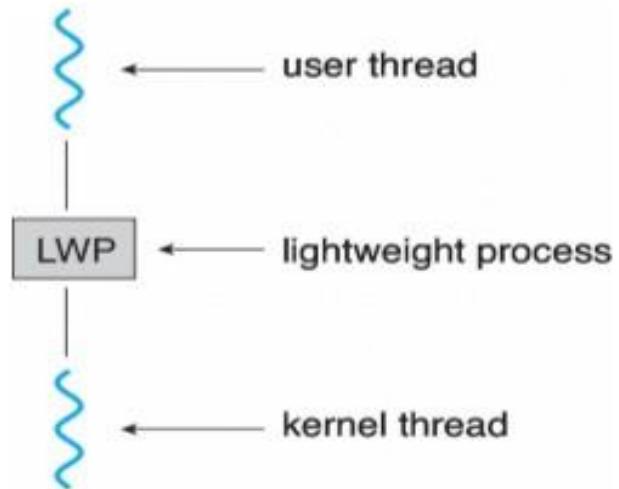
36

- **Thread-local storage (TLS)** allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- Different from local variables
  - Local variables visible only during single function invocation
  - TLS visible across function invocations
- Similar to **static** data
  - TLS is unique to each thread

# Scheduler Activations

37

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between user and kernel threads – **lightweight process (LWP)**
  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread
  - How many LWPs to create?
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads



## ■ Windows Threads

## ■ Linux Threads

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

**BAI1208**

**CHAPTER 6**  
**Deadlocks**

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

# What is Deadlock?

2

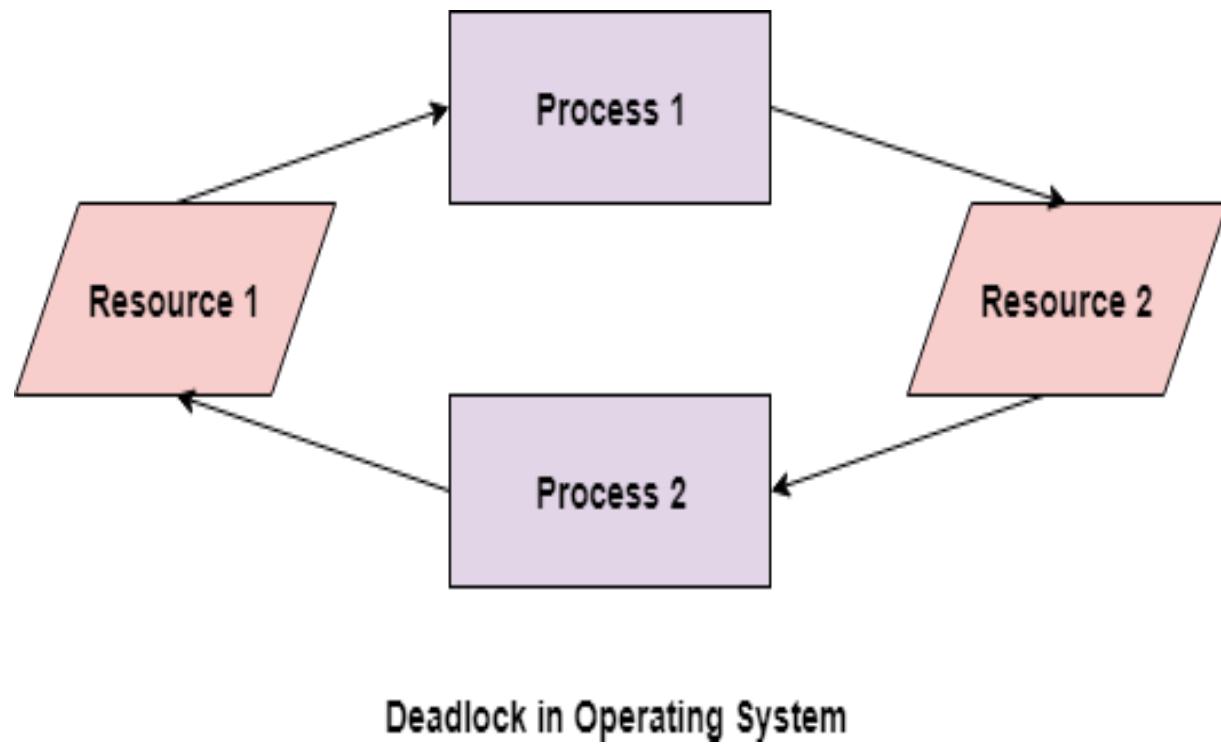
**Deadlock** is a situation that occurs in OS when any process enters a waiting state because another waiting process is holding the demanded resource. Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software.

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.

# Deadlock Example

3



In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

## Deadlock Detection

4

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods:

All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.

Resources can be preempted from some processes and given to others till the deadlock is resolved.

## Deadlock Prevention

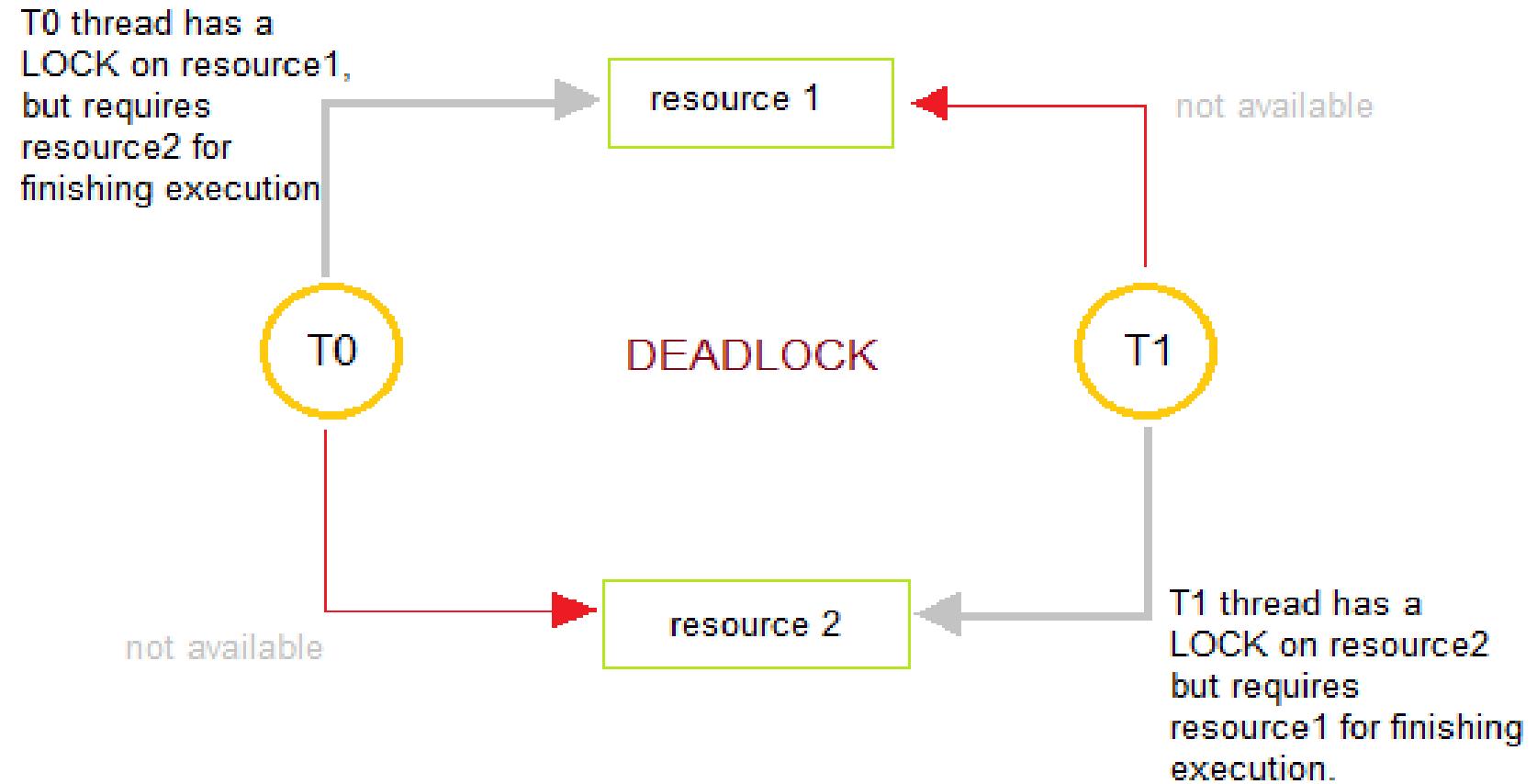
It is very important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight chance that a transaction may lead to deadlock in the future, it is never allowed to execute.

## Deadlock Avoidance

It is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases.

# Deadlock Example

5



# DEADLOCKS

## EXAMPLES:

- "It takes money to make money".
- You can't get a job without experience; you can't get experience without a job.

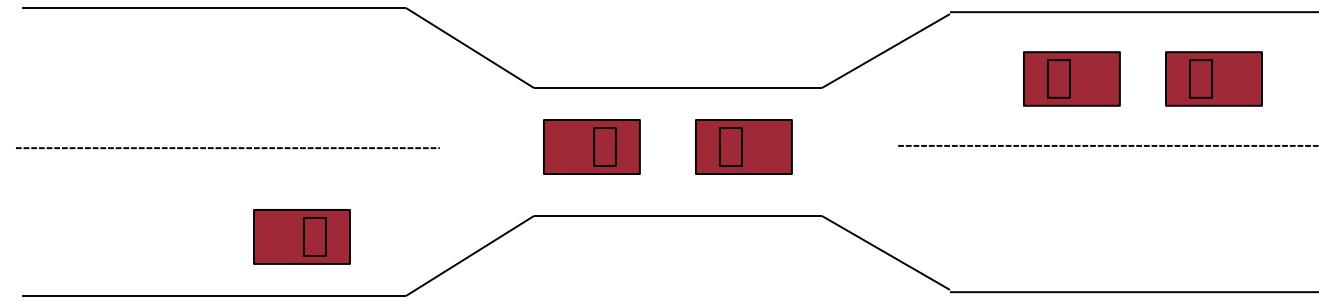
## BACKGROUND:

- The cause of deadlocks: Each process needing what another process has. This results from sharing resources such as memory, devices, links.

Under normal operation, a resource allocations proceed like this:

1. Request a resource (suspend until available if necessary ).
2. Use the resource.
3. Release the resource.

# 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.

## NECESSARY CONDITIONS

**ALL** of these four **must** happen simultaneously for a deadlock to occur:

### Mutual exclusion

One or more than one resource must be held by a process in a non-sharable (exclusive) mode.

### Hold and Wait

A process holds a resource while waiting for another resource.

### No Preemption

There is only voluntary release of a resource - nobody else can make a process give up a resource.

### Circular Wait

Process A waits for Process B waits for Process C .... waits for Process A.

# RESOURCE ALLOCATION GRAPH



A visual ( mathematical ) way to determine if a deadlock has, or may occur.

**G = ( V, E )**    The graph contains nodes and edges.

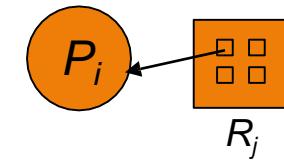
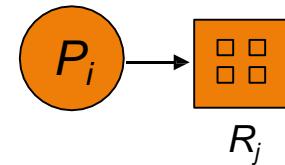
**V**                Nodes consist of processes = { P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, ... } and resource types

{ R<sub>1</sub>, R<sub>2</sub>, ... }

**E**                Edges are ( P<sub>i</sub>, R<sub>j</sub> ) or ( R<sub>i</sub>, P<sub>j</sub> )

An arrow from the **process** to **resource** indicates the process is **requesting** the resource. An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

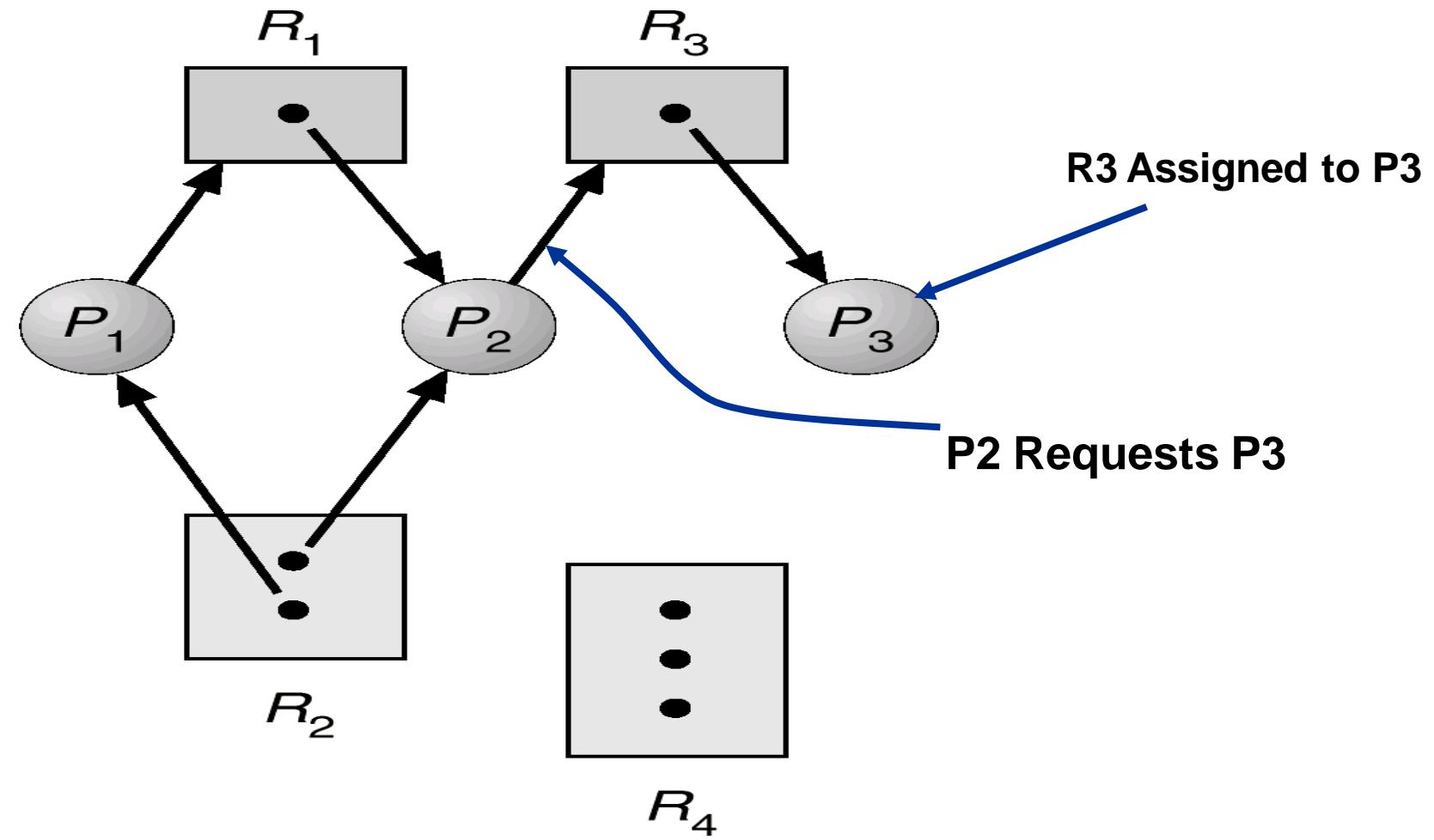
Process is a circle, resource type is square; dots represent number of instances of resource in type. Request points to square, assignment comes from dot.



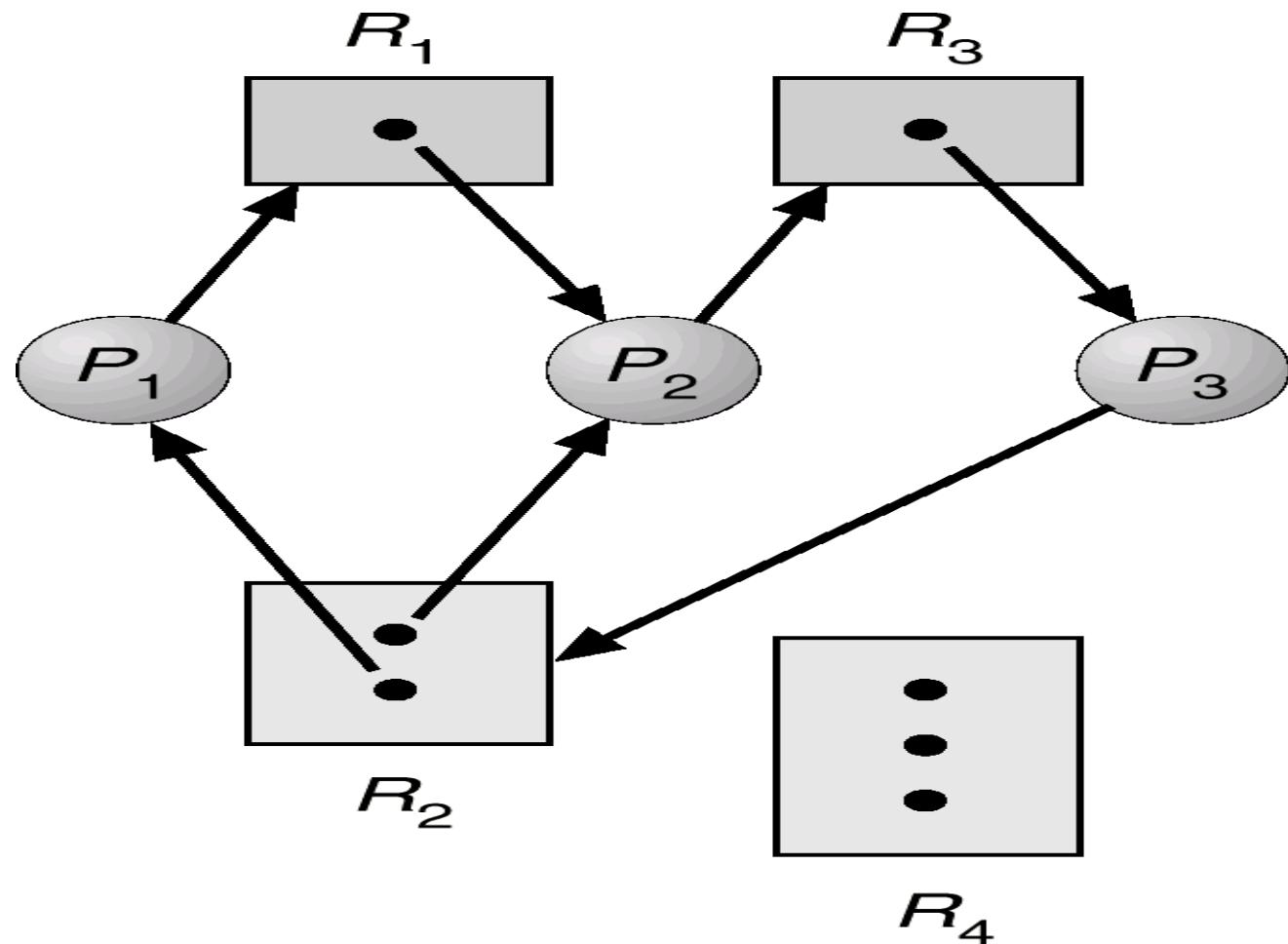
# RESOURCE ALLOCATION GRAPH



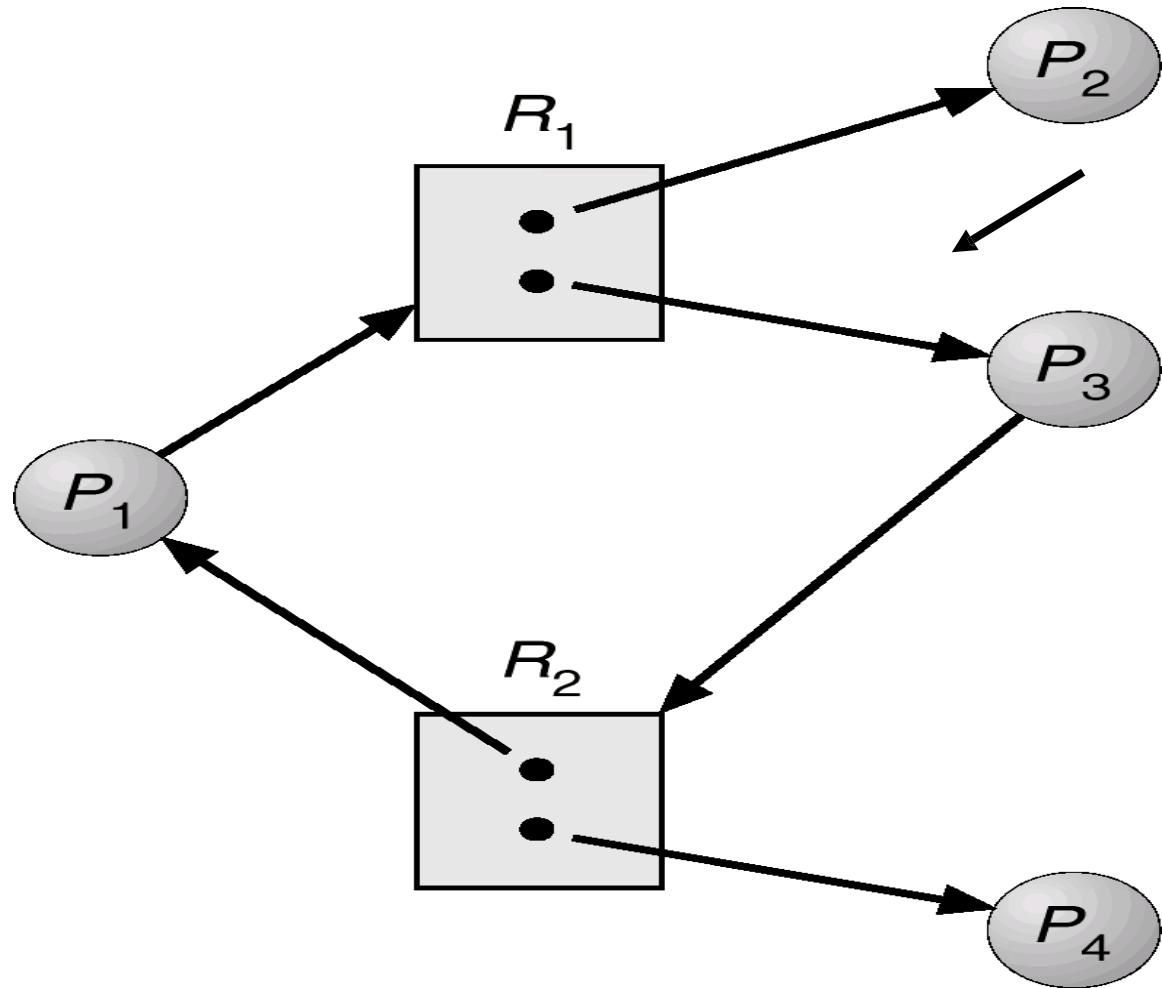
- If the graph contains no cycles, then no process is deadlocked.
- If there is a cycle, then:
  - a) If resource types have multiple instances, then deadlock MAY exist.
  - b) If each resource type has 1 instance, then deadlock has occurred.



## Resource allocation graph with a deadlock.



## Resource allocation graph with a cycle but no deadlock.



# HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES



- ❑ There are three methods:
  - ❑ Most Operating systems do this!!
- ❑ Ignore Deadlocks:
- ❑ Ensure deadlock **never** occurs using either
  - Prevention** Prevent any one of the 4 conditions from happening.
  - Avoidance** Allow all deadlock conditions, but calculate cycles about to happen and stop dangerous operations..
- ❑ **Allow** deadlock to happen. This requires using both:
  - Detection** Know a deadlock has occurred.
  - Recovery** Regain the resources.

## □ Deadlock Prevention

- Do not allow one of the four conditions to occur

- **Mutual exclusion:**

- Automatically holds for printers and other non-sharables.
- Shared entities (read only files) don't need mutual exclusion (and aren't susceptible to deadlock.)
- Prevention not possible, since some devices are intrinsically non-sharable.

- **Hold and wait:**

- Collect all resources before execution.
- A particular resource can only be requested when no others are being held. A sequence of resources is always collected beginning with the same one.
- Utilization is low, starvation possible.

## I No preemption:

- Release any resource already being held if the process can't get an additional resource.
- Allow preemption - if a needed resource is held by another process, which is also waiting on some resource, steal it. Otherwise wait.

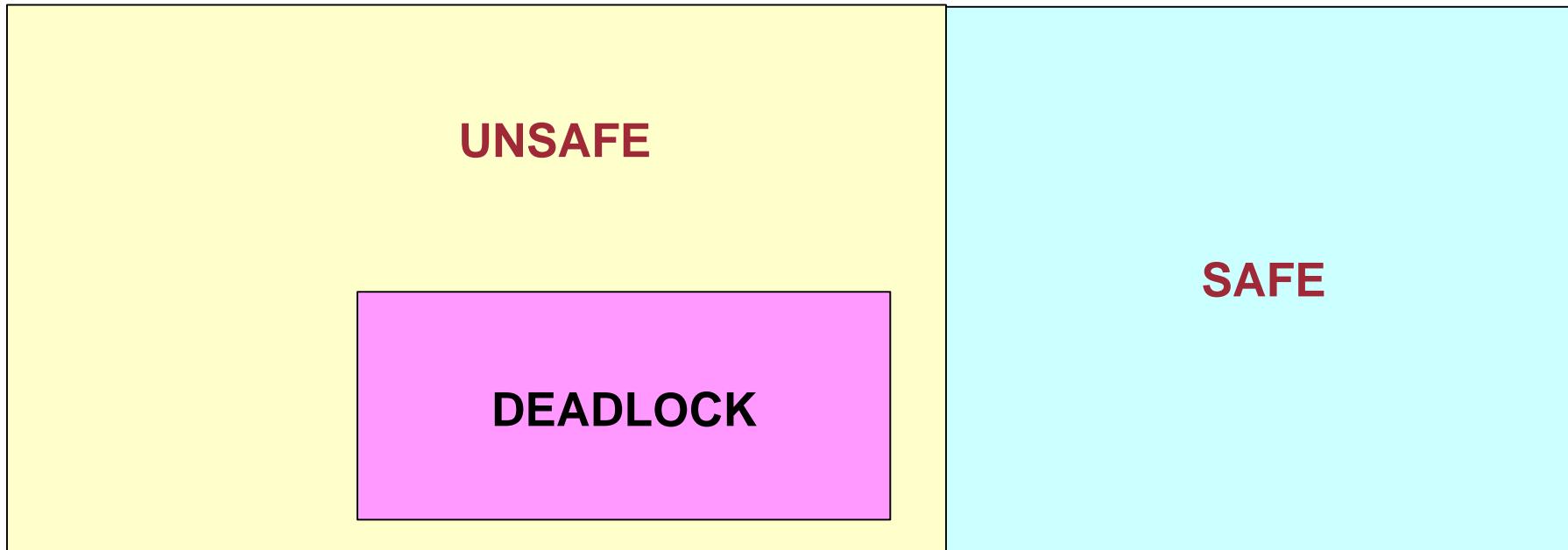
## I Circular wait:

- Number resources and only request in ascending order.
- EACH of these prevention techniques may cause a decrease in utilization and/or resources. For this reason, prevention isn't necessarily the best technique.
- Prevention is generally the easiest to implement.

# Deadlock avoidance

- Prior information about the usage of resources by processes is required
- This helps in deciding whether a process should wait for a resource or not
- It ensures that circular wait never happens
- Safe state
  - I If it can allocate resources upto the maximum available and is not in a state of deadlock
- Unsafe state

**NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.**



Only with luck will processes avoid deadlock.

O.S. can avoid deadlock.

Let's assume a very simple model: each process declares its maximum needs. In this case, algorithms exist that will ensure that no unsafe state is reached.

## **EXAMPLE:**

There exists a total of 12 tape drives. The current state looks like this:

## Example

- In this example,  $\langle p_1, p_0, p_2 \rangle$  is a workable sequence.
- Suppose  $p_2$  requests and is given one more tape drive. What happens then?

Process	Max Needs	Allocated	Current Needs
P0	10	5	5
P1	4	2	2
P2	9	3	6

## □ Resource allocation algorithm

- Let be another edge called a claim edge
- A directed edge from ~~process~~ to resource shows that the process may request the resource some time later (  $P_i \rightarrow R_j$  )
- Dashed lines represent a claim edge
- The claim edge becomes request edge when an actual request is made
- When the resource is released the assignment edge is converted to claim edge
- The process must be associated with all its claim edge before it starts executing
- the request is granted only in the case it will not create a cycle

## □ Bankers algorithm

- ▀ Resource allocation graph algorithm is not applicable where resources have multiple instances
- ▀ Banker algorithm can be used in such cases
- ▀ A new process entering the system must make known a priori the maximum instance of each resource that it needs
- ▀ If the allocation ensures safe state then it is allocated else processes must waitdc

## □ Data structures maintained

- N : number of processes in the system
- M : the number of recourse type
- Available : Available[ j ] = k , it means k instance of the Resource j is available at a particular instance
- Max : maximum need for the resource by a process.  
 $\text{Max}[i][j]=k$ , says the ith process can request for atmost k instance of jth resource
- Allocation : Allocation [i][j] =k , currently holding
- Need: Need[i][j]=k , means its needs k more instances

**Need[I]** - the remaining resource needs of each process.

**Work** - Temporary variable - how many of the resource are currently available.

**Finish[I]** - flag for each process showing we've analyzed that process or not.

## □ Safety Algorithm

1. **Initialize work = available**  
**Initialize finish[i] = false, for i = 1,2,3,..n**
2. **Find an i such that:**  
**finish[i] == false and need[i] <= work**  
**If no such i exists, go to step 4.**
3. **work = work + allocation[i]**  
**finish[i] = true**  
**goto step 2**
4. **if finish[i] == true for all i, then the system is in a safe state.**

## □ Resource Request Algorithm

- $\text{Request}_i[j]=k$ , process I want  $k$  instance of resource  $j$
- Step 1 If  $\text{Request}_i \leq \text{need}_i$  go to step 2
- Else error
- Step 2 if  $\text{request}_i \leq \text{Available}_i$  goto step 3
- Else must wait
- Step 3 allocation is made
- $\text{Available} = \text{Available} - \text{Request}_i$
- $\text{Allocation}_i = \text{allocation}_i + \text{request}_i$
- $\text{Need}_i = \text{need}_i - \text{request}_i$

## Do these examples:

Consider a system with: five processes,  $P_0 \rightarrow P_4$ , three resource types, A, B, C.

Type A has 10 instances, B has 5 instances, C has 7 instances.

At time  $T_0$  the following snapshot of the system is taken.



Is the system  
in a safe  
state?

Max Needs = allocated + can-be-requested

	←	Alloc	→		←	Req	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
$P_0$	0	1	0		7	4	3		3	3	2
$P_1$	2	0	0		1	2	2				
$P_2$	3	0	2		6	0	0				
$P_3$	2	1	1		0	1	1				
$P_4$	0	0	2		4	3	1				

## Do these examples:

Consider a system with: five processes,  $P_0 \rightarrow P_4$ , three resource types, A, B, C.

Type A has 10 instances, B has 5 instances, C has 7 instances.

At time  $T_0$  the following snapshot of the system is taken.

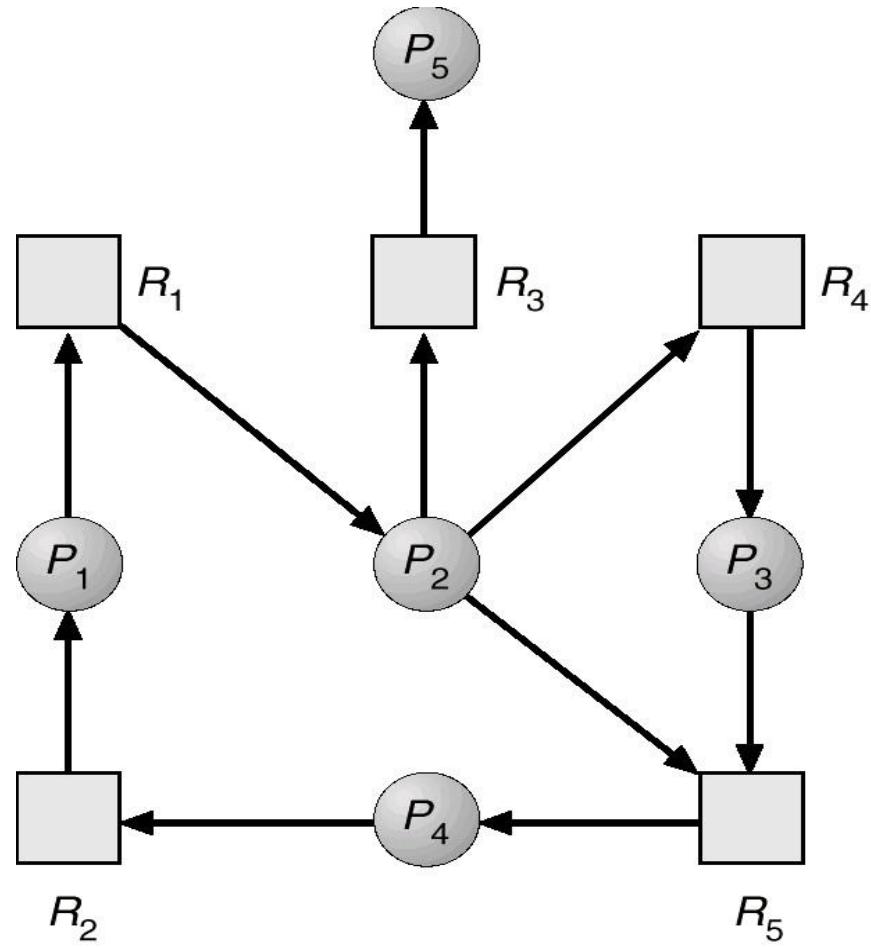
Max Needs = allocated + can-be-requested

	←	Alloc	→		←	Req	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
P0	0	1	0		7	4	3		3	3	2
P1	2	0	0		1	2	2				
P2	3	0	2		6	0	0				
P3	2	1	1		0	1	1				
P4	0	0	2		4	3	1				

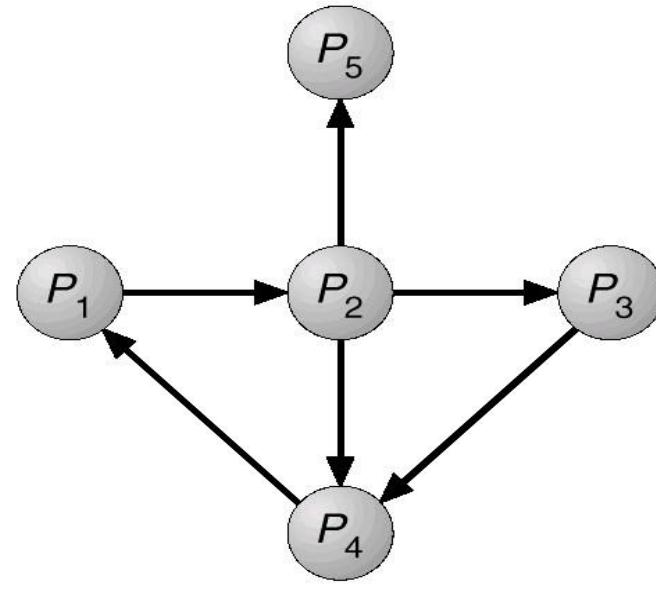
Step	Work	Finish	Safe Sequence
0	3 3 2	FFFFF <>	
Example from the book			

# Deadlock detection

- Detect the deadlock
- Recover from the deadlock
- Single Instance of a Resource
  - Wait for graph
    - remove the resources from the usual graph and collapse edges.
    - An edge from  $p(j)$  to  $p(i)$  implies that  $p(j)$  is waiting for  $p(i)$  to release.



(a)



(b)

## □ Multiple instance of a resource

- Data structure
- **available** - records how many resources of each type are available.
- **allocation** - number of resources of type m allocated to process n.
- **request** - number of resources of type m requested by process n.
- Let **work** and **finish** be vectors of length **m** and **n** respectively.

- 1. Initialize                   $\text{work[]} = \text{available}[]$**   
**For     $i = 1, 2, \dots, n$ , if  $\text{allocation}[i] \neq 0$  then**  
     **$\text{finish}[i] = \text{false}$ ; otherwise,  $\text{finish}[i] = \text{true}$ ;**
- 2. Find an  $i$  such that:**  
     **$\text{finish}[i] == \text{false}$  and  $\text{request}[i] \leq \text{work}$**   
    **If no such  $i$  exists, go to step 4.**
- 3.  $\text{work} = \text{work} + \text{allocation}[i]$**   
     **$\text{finish}[i] = \text{true}$**   
    **goto step 2**
- 4. if  $\text{finish}[i] == \text{false}$  for some  $i$ , then the system is in deadlock state.**  
    **IF  $\text{finish}[i] == \text{false}$ , then process  $p[i]$  is deadlocked.**

## **EXAMPLE**

We have three resources, A, B, and C. A has 7 instances, B has 2 instances, and C has 6 instances. At this time, the allocation, etc. looks like this:

		←	Alloc	→		←	Req	→		←	Avail	→
	A	B	C		A	B	C		A	B	C	
P0	0	1	0		0	0	0		0	0	0	
P1	2	0	0		2	0	2					
P2	3	0	3		0	0	0					
P3	2	1	1		1	0	0					
P4	0	0	2		0	0	2					

- **PROCESS TERMINATION:**
- Could delete all the processes in the deadlock -- this is expensive.
  - Delete one at a time until deadlock is broken ( time consuming ).
  - Select who to terminate based on priority, time executed, time to completion, needs for completion, or depth of rollback
  - In general, it's easier to preempt the resource, than to terminate the process.
-

## □ **RESOURCE PREEMPTION:**

- Select a victim - which process and which resource to preempt.
- Rollback to previously defined "safe" state.
- Prevent one process from always being the one preempted ( starvation ).

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

## CHAPTER 7

# Memory Management

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121



# Introduction to Memory Management

2

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhances the performance.

## Cont...

3

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as **Dynamic Linking**.

**Memory management** is the functionality of an **operating system** which handles or manages primary **memory** and moves processes back and forth between main **memory** and disk during execution. **Memory management** keeps track of each and every **memory** location, regardless of either it is allocated to some process or it is free.

# Swapping

4

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

## Contiguous Memory Allocation

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

# Memory Protection

5

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

# Memory Allocation

6

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

**First Fit:** The first hole that is big enough is allocated to program.

**Best Fit:** The smallest hole that is big enough is allocated to program.

**Worst Fit:** The largest hole that is big enough is allocated to program.

# Fragmentation

7

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation.**

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but of no use.

# Paging

8

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contiguous. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

## Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses*.

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

## Segmentation

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

# Segmentation with Paging

9

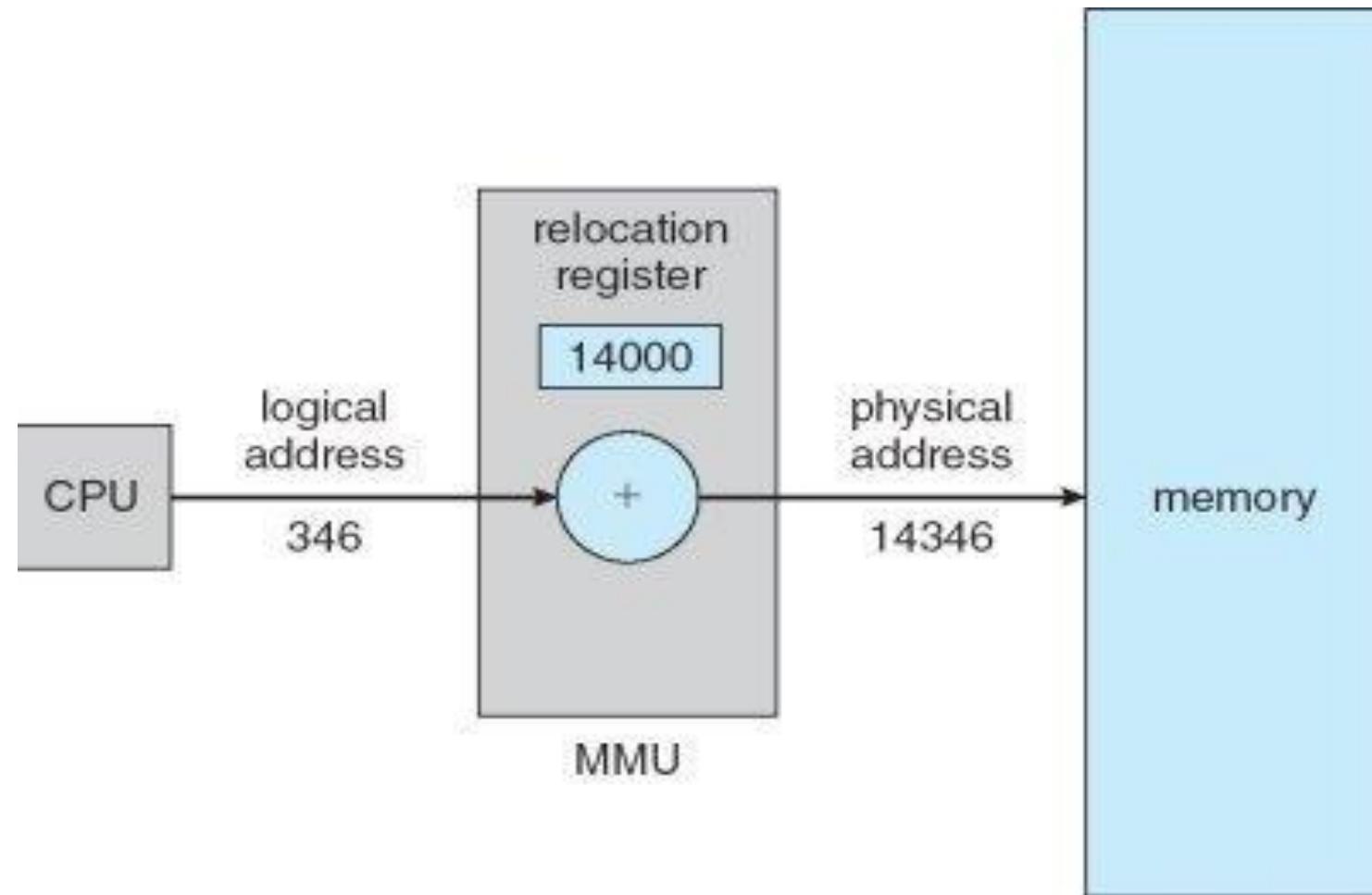
Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

- Segment numbers(S)
- Page number (P)
- The displacement or offset number (D)



# Logical versus Physical Address Space

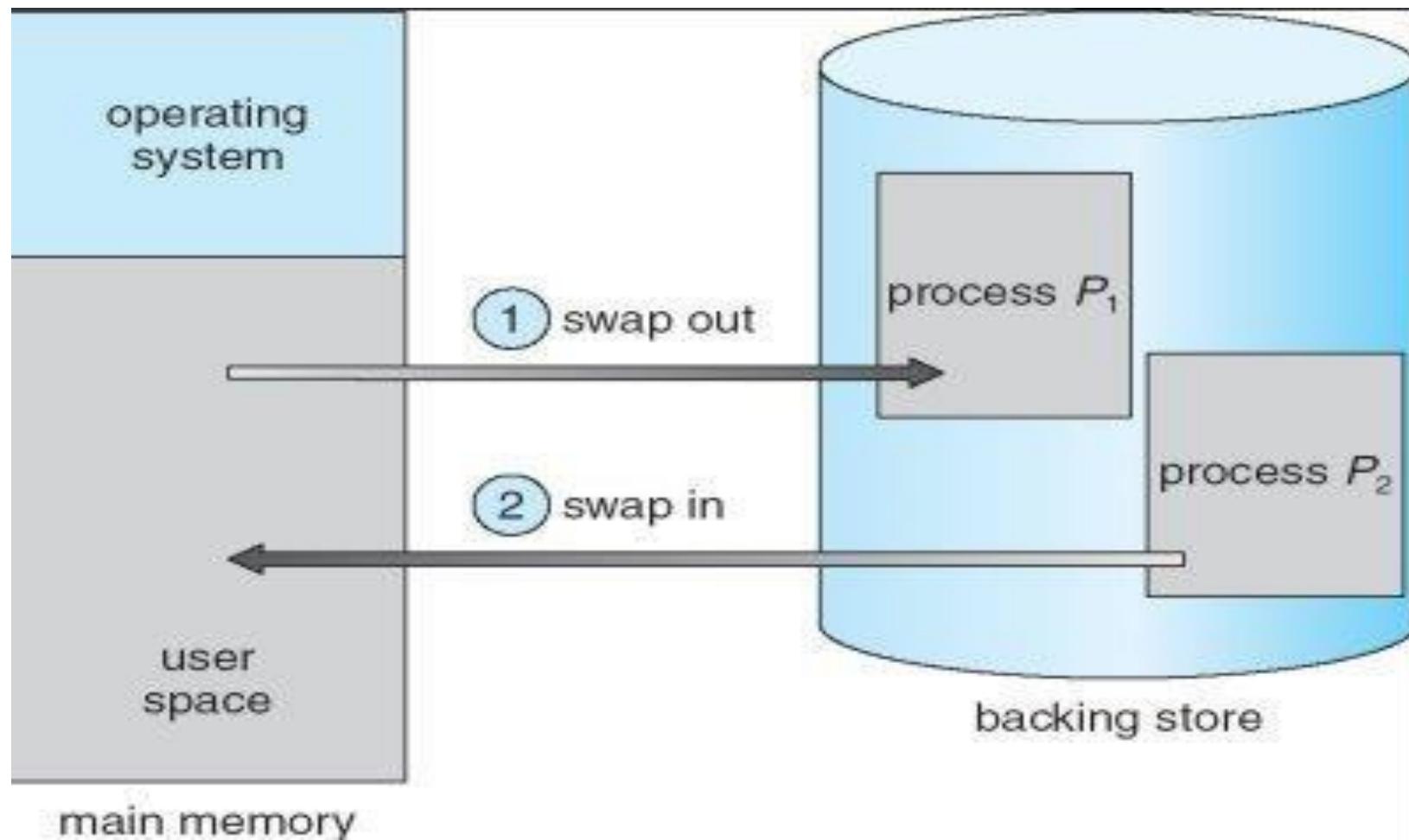
- Address generated by CPU is referred to as logical address
- logical address is also sometimes referred to as a virtual address
- The set of logical addresses generated by CPU for a program is called the logical address space
- Address loaded into the memory address register (MAR) to fetch or store is physical address
- At run time the virtual addresses are mapped to physical addresses by the memory management unit (MMU)
- Relocation register contains a value to be added to every address generated by the CPU



# Swapping

- Processes are swapped between the main memory and the backing store when priority based CPU scheduling algorithm is used
- Some time called as rollout/roll in
- Swapped out process can be swapped in to the same or different memory location
- If binding is done at load time swap in to same location
- If binding is done at execution time swap in into different memory space
- A process to be swapped out must be idle

- Problems in swapping
  - P1 waiting for I/O
  - P1 is swapped out and p2 is swapped in to that memory
  - I/O uses that memory which belongs to P2
- Solutions
  - Never swap out a process waiting for I/O
  - I/O takes place in OS buffer and not in user area



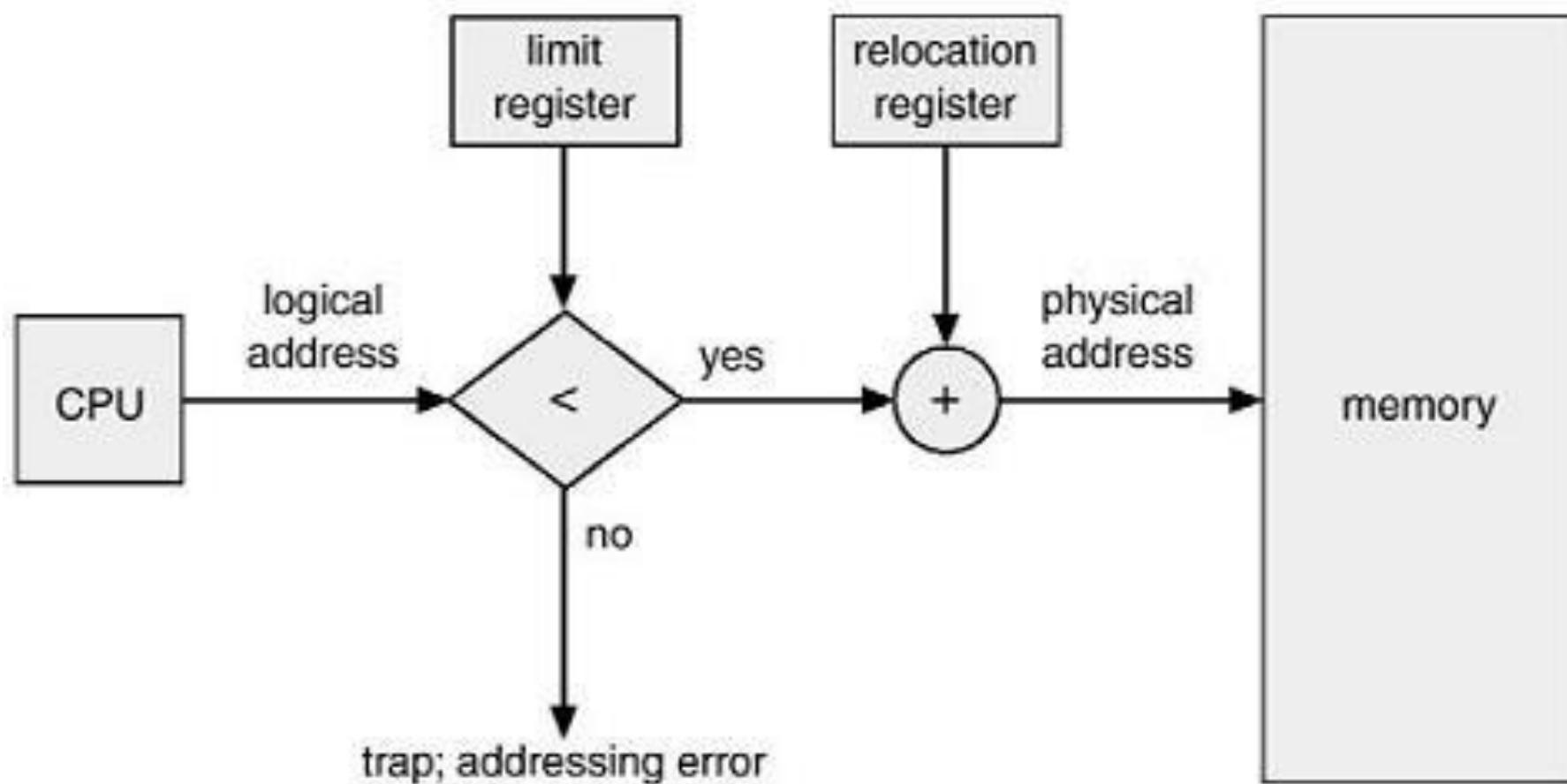


# Contiguous Allocation

- Main memory is usually divided into two partitions
  - Resident Operating system loaded into it
  - Loading user programs

## □ Single Partition Allocation

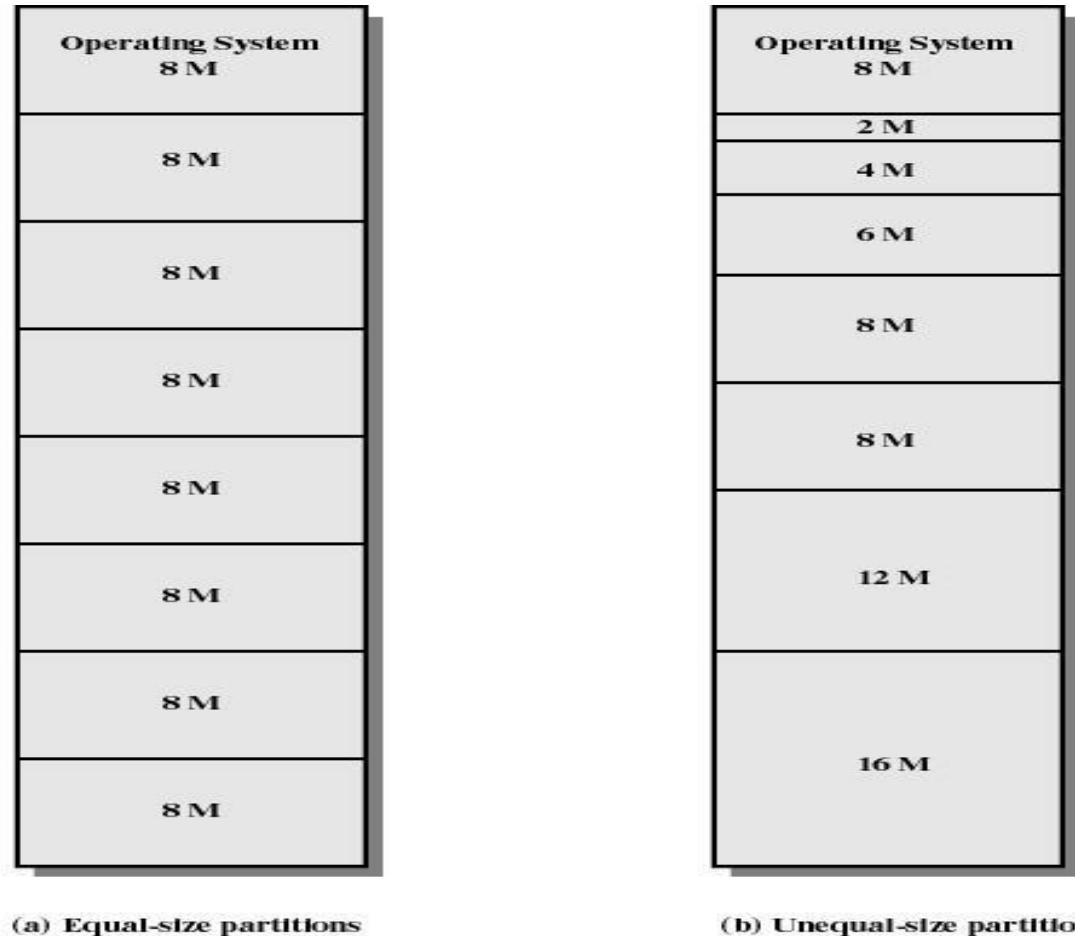
- OS resides in lower memory
- User processes execute in higher memory
- User programs can access Lower memory intentionally or accidentally
- Protection can be provided by the use of limit register and relocation register
- relocation register contain the smallest physical address that can be accessed
- The limit register contains the range of logical addresses



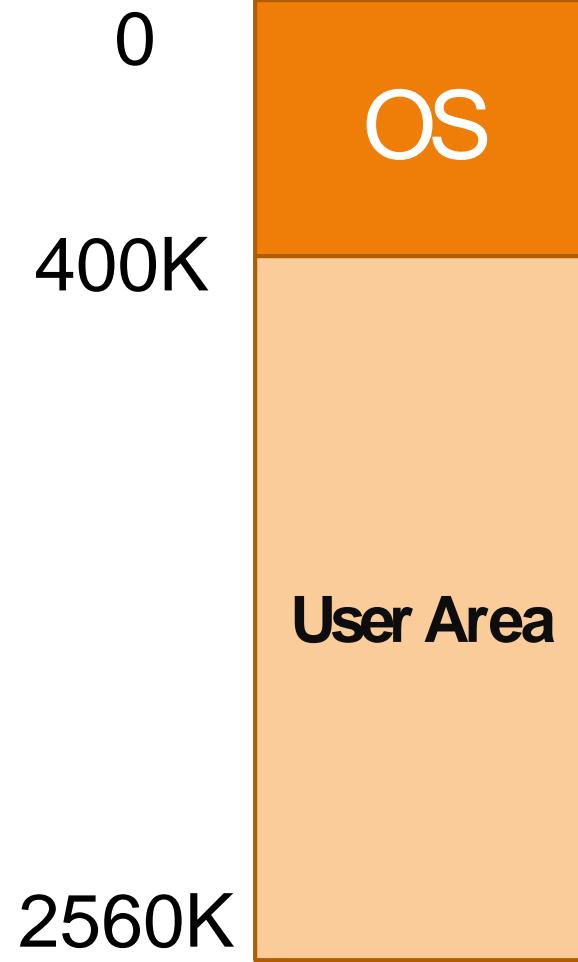
## □ Multiple Partition Allocation

Many processes resides in memory so that CPU can switch between processes

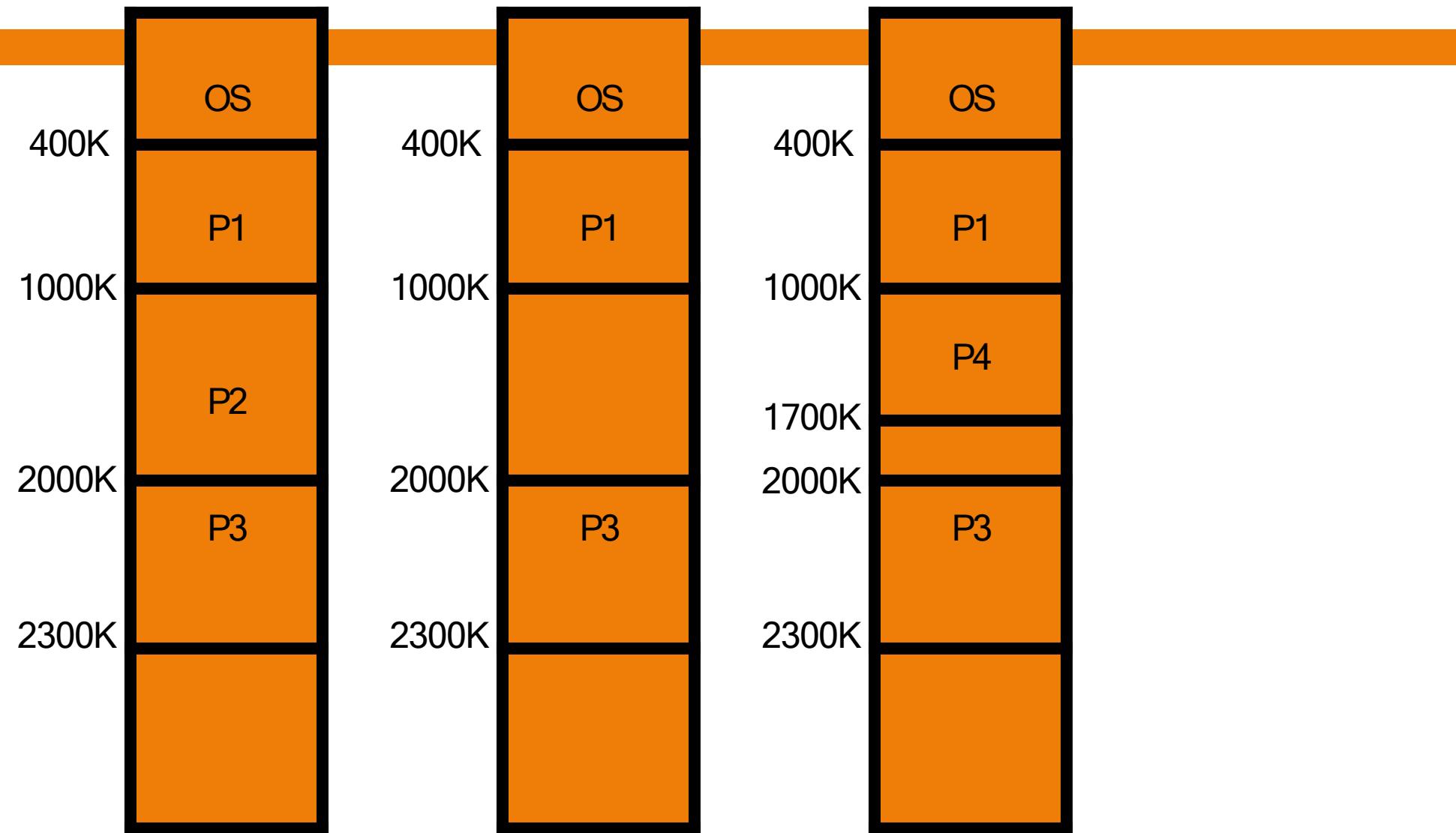
- User area of the memory has to be divided into partitions
- Fixed number of partitions
- The degree of multiprogramming is equal to no of partitions
- Size of partition size is an issue
- Small size big program will not fit
- If it is too large waste of memory
- Variable size can be used (not fixed)

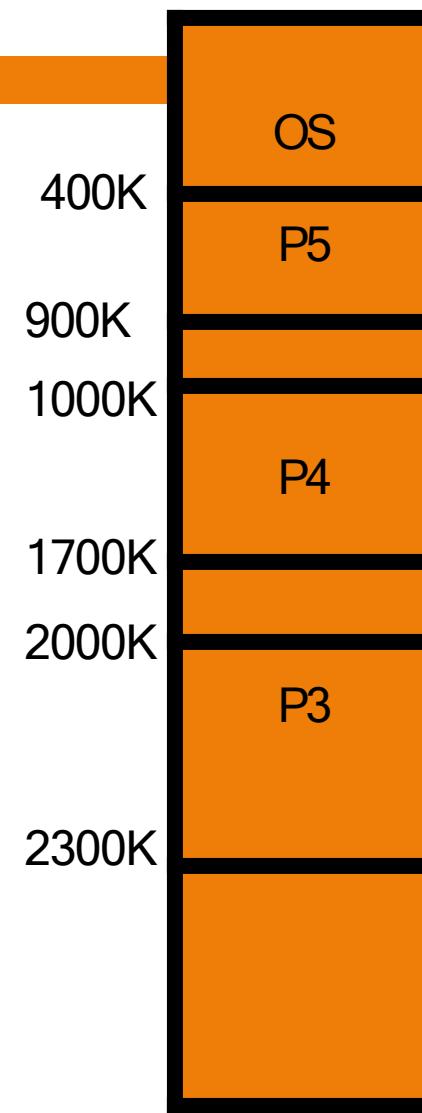
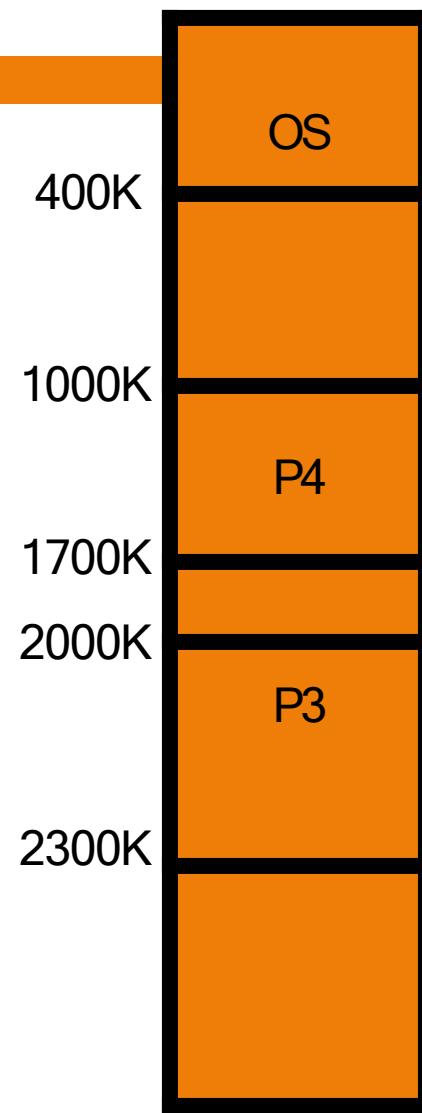


**Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory**



Process	Memory	Time
P1	600k	10
P2	1000k	5
P3	300K	20
P4	700K	8
P5	500K	15





- A table keeps track of that memory which is in use and which is free
- New holes created that are adjacent to existing holes merge to form big holes
- First fit
  - ▣ Allocate the first hole that is big enough
  - ▣ Search can start from beginning or from where last terminated

- Best Fit
  - ▣ Allocate the smallest hole that is big enough to hold the process
  - **Entire list has to be searched or ordered list of holes**
  
- Worst fit
  - ▣ **Allocate the large hole available**
  - ▣ **Entire list has to be searched or ordered list of holes**

# Fragmentation



- The large holes will be divided into set of smaller holes that are scattered in between the processes
- External Fragmentation
  - ▣ There may be a situation where the total size of these scattered holes is large enough to hold another process for execution but the process cannot be loaded
- Internal Fragmentation
  - ▣ The situation where only few bytes say 1 or 2 would be free
  - ▣ Cost of keeping track of this small hole will be high
  - ▣ Extra bit of hole is also allocated to requesting process

## □ Compaction

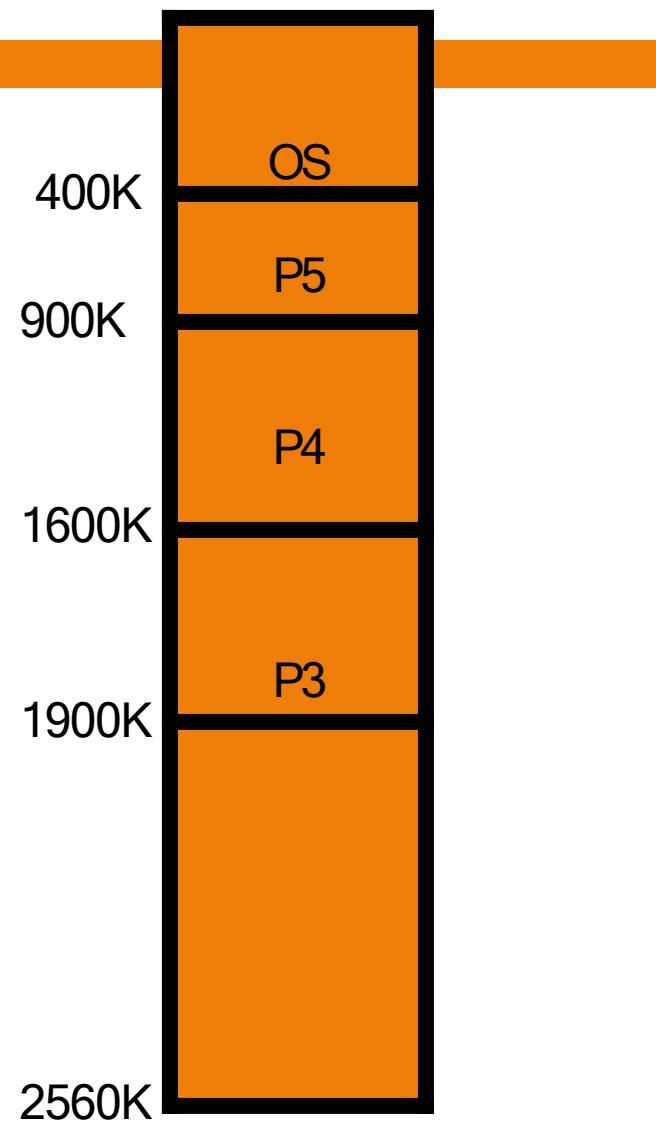
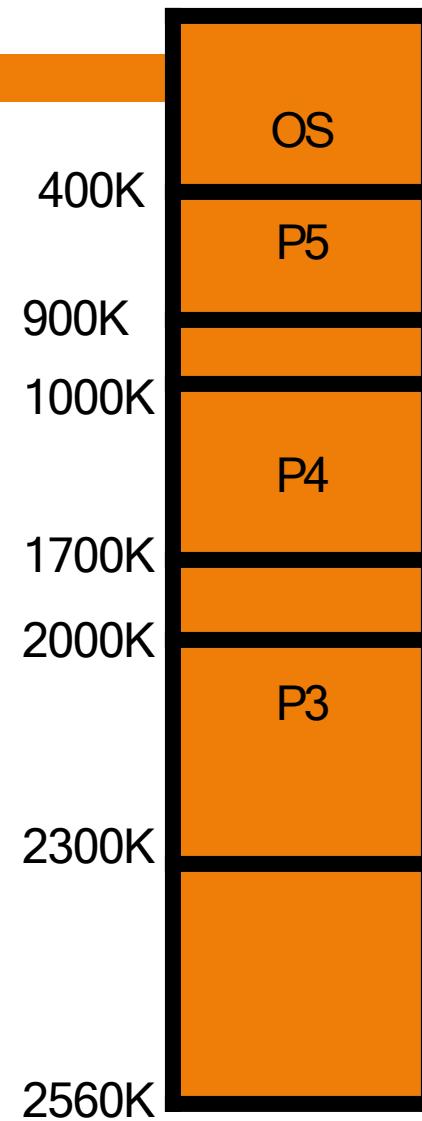
■ Relocate processes in memory so those fragmented holes create one contiguous hole in the memory

■ Issues in compaction

■ Relocation

■ Where to create the hole

■ Process to be roll out and roll in



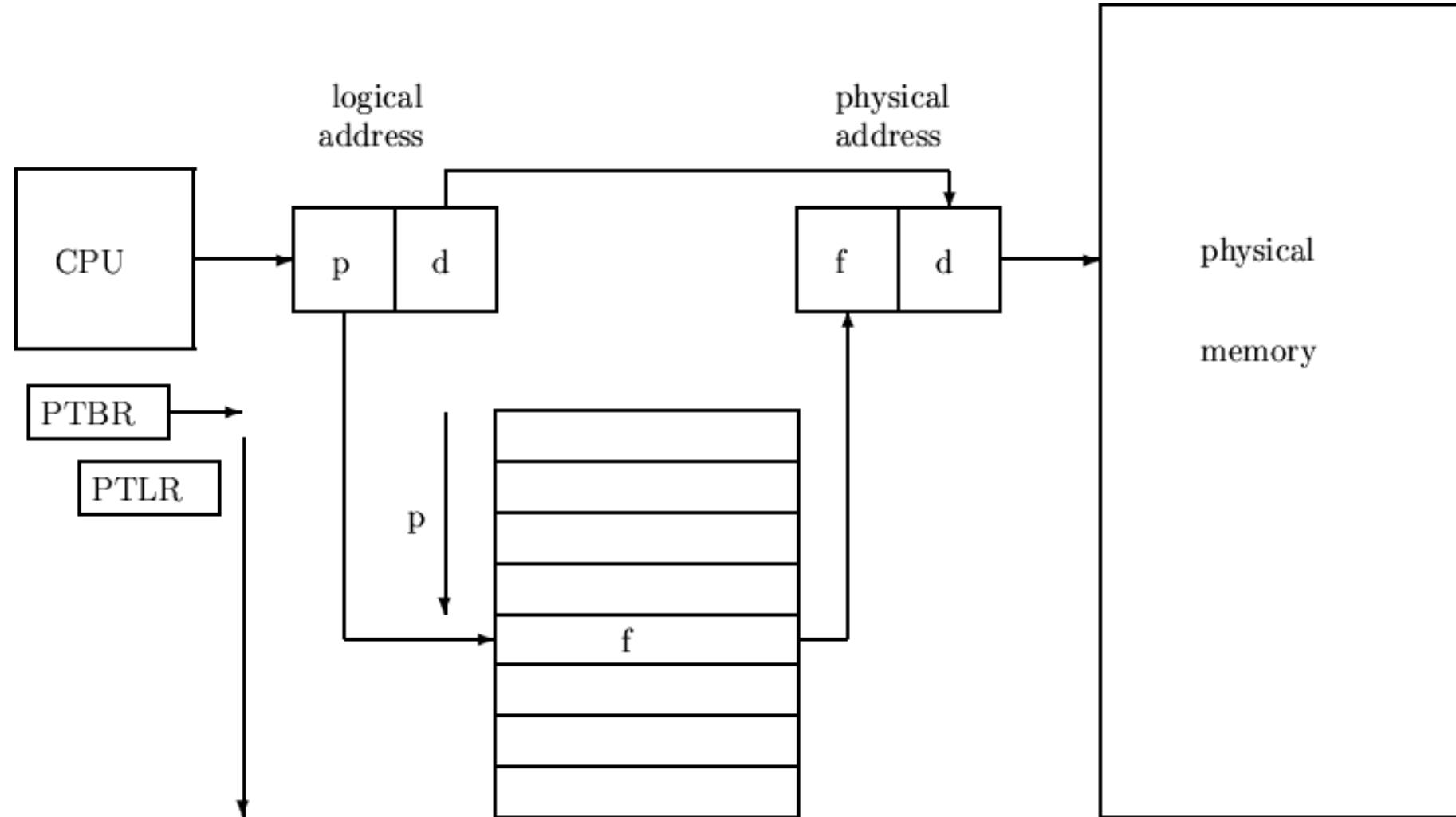


# Paging

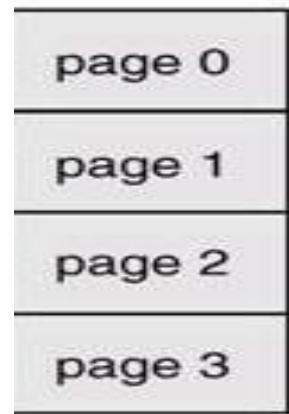
- Contiguous allocation scheme requires that a process can be loaded into memory for execution if and only if contiguous memory large enough to hold the process is available
  - ▣ Problems
    - External fragmentation
    - Compaction is one solution
- Non contiguous allocation
  - ▣ Implemented using paging scheme

## □ Concept of Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called pages and chunk of memory are called frames
- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page



- Logical Address generated by CPU consist of two parts
  - ▣ Page Number (p)
    - Index to a page table
    - Page table contain the base address of each frame in physical memory
  - ▣ Page Offset (d)
- The base address is added with page offset to generate the physical address to access the memory unit
- The size of the page is usually a power 2

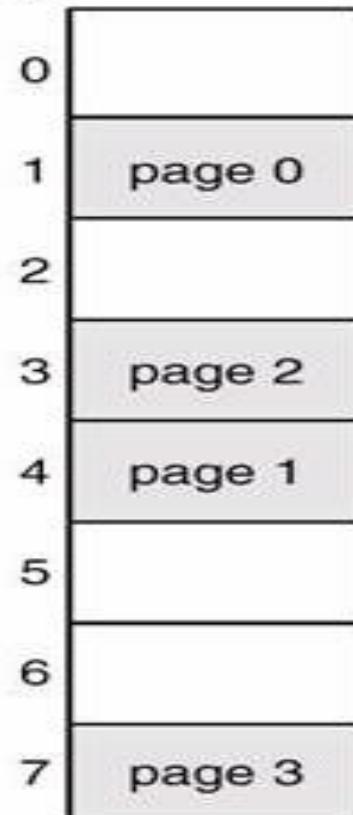


logical  
memory

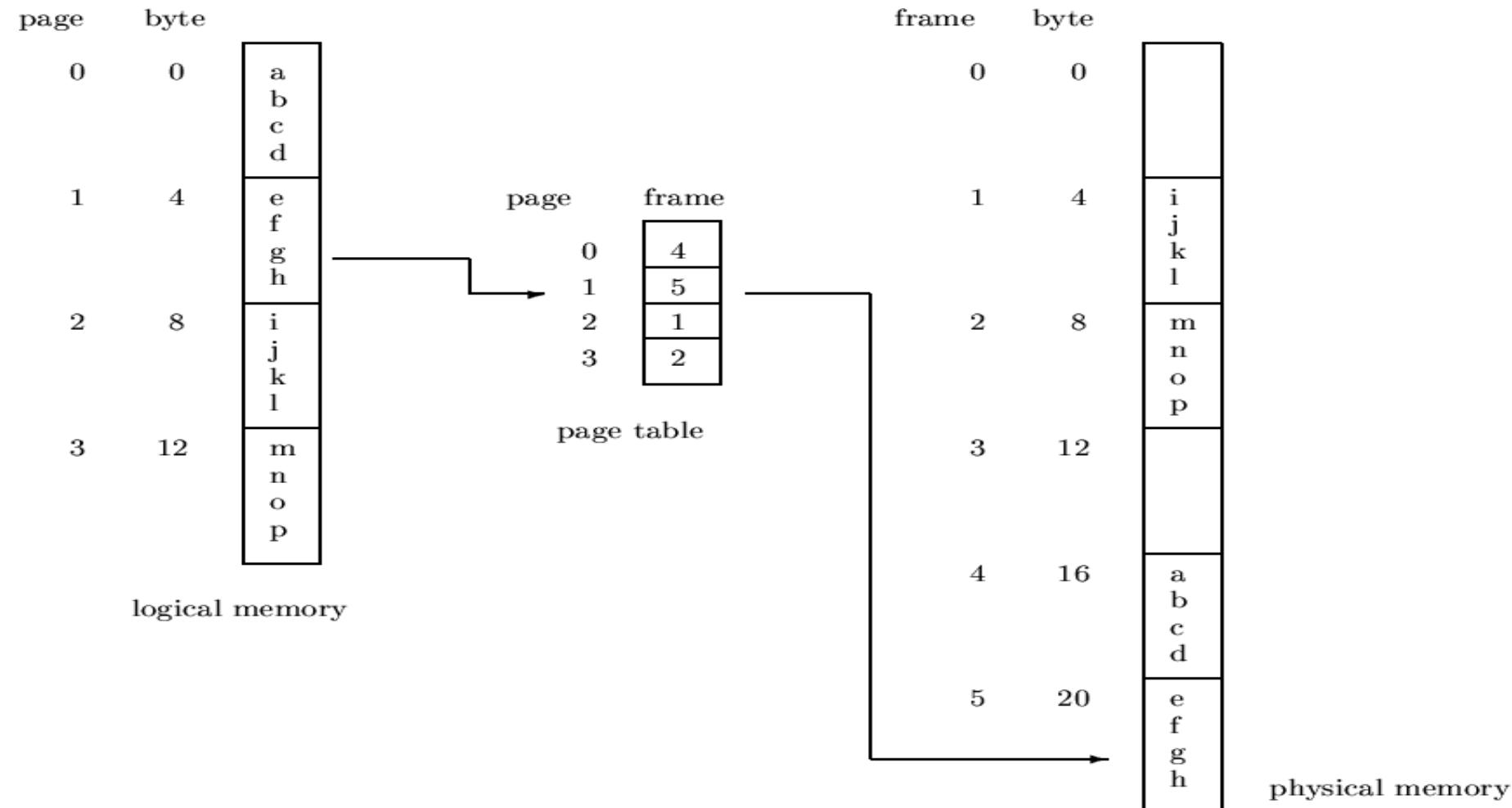
0	1
1	4
2	3
3	7

page table

frame  
number



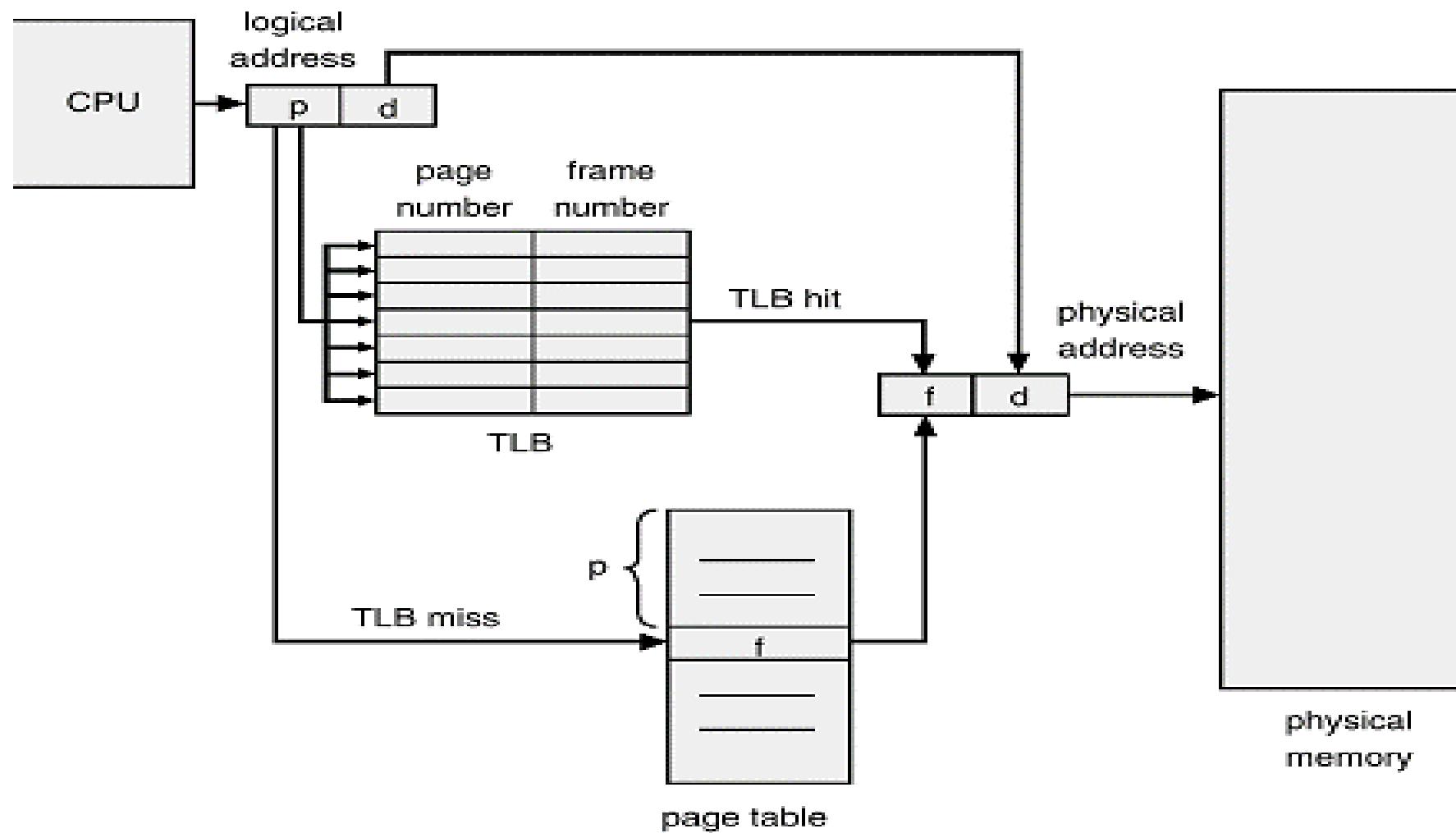
physical  
memory



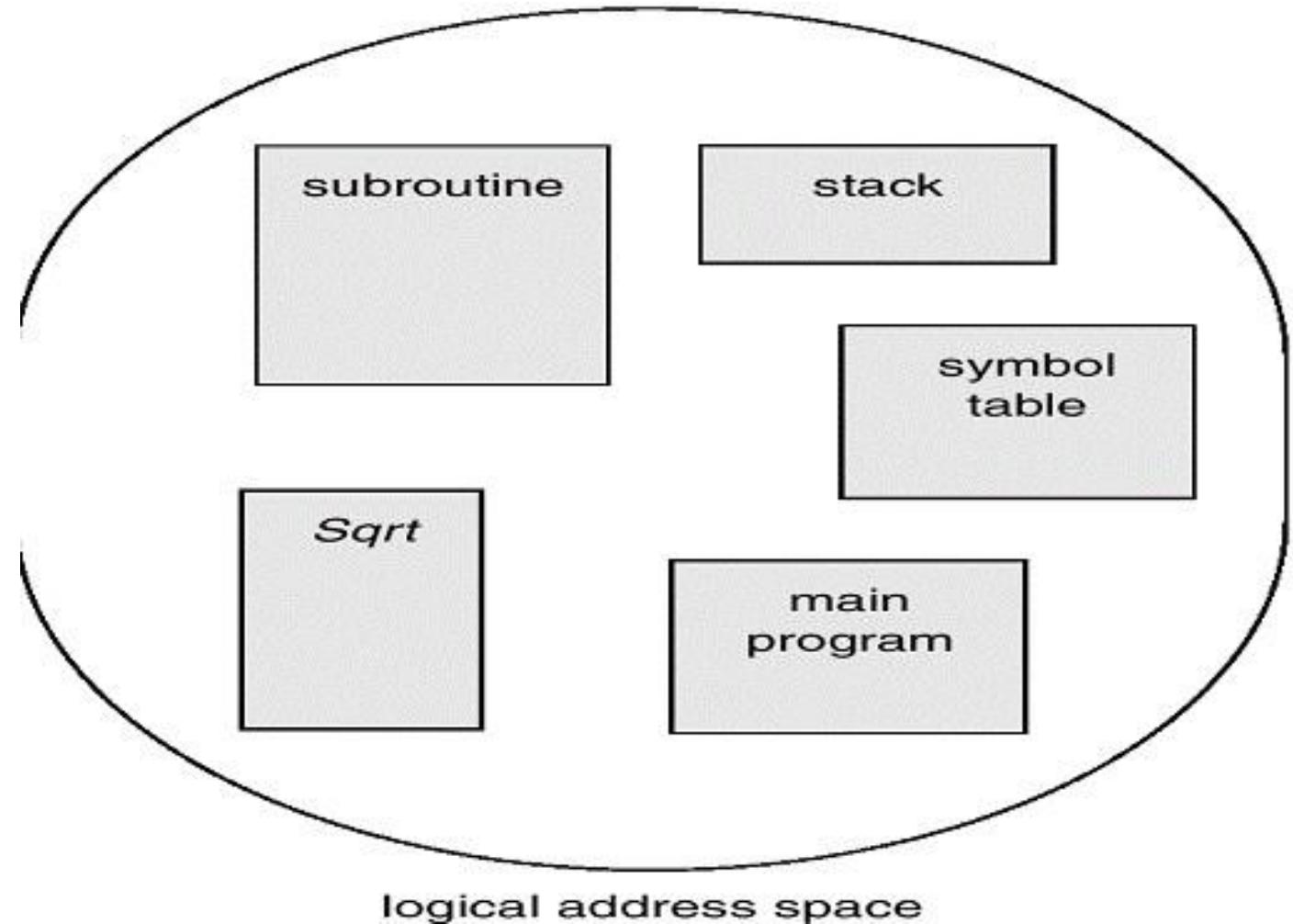
- Paging does not suffer from External fragmentation
- Internal fragmentation
  - ▣ The last page will not full
- A list of free frames are maintained
- Allocation details are maintained in the frame table
  - ▣ One entry for each frame
  - ▣ Showing whether it is free
  - ▣ Allocated to which page of which process

## □ Page Table Implementation

- Page table for process is kept in the memory
  - PTBR (page table base register)
  - Two memory access is required
- To overcome the above problem we use cache TLBs (Translation look-aside buffer)

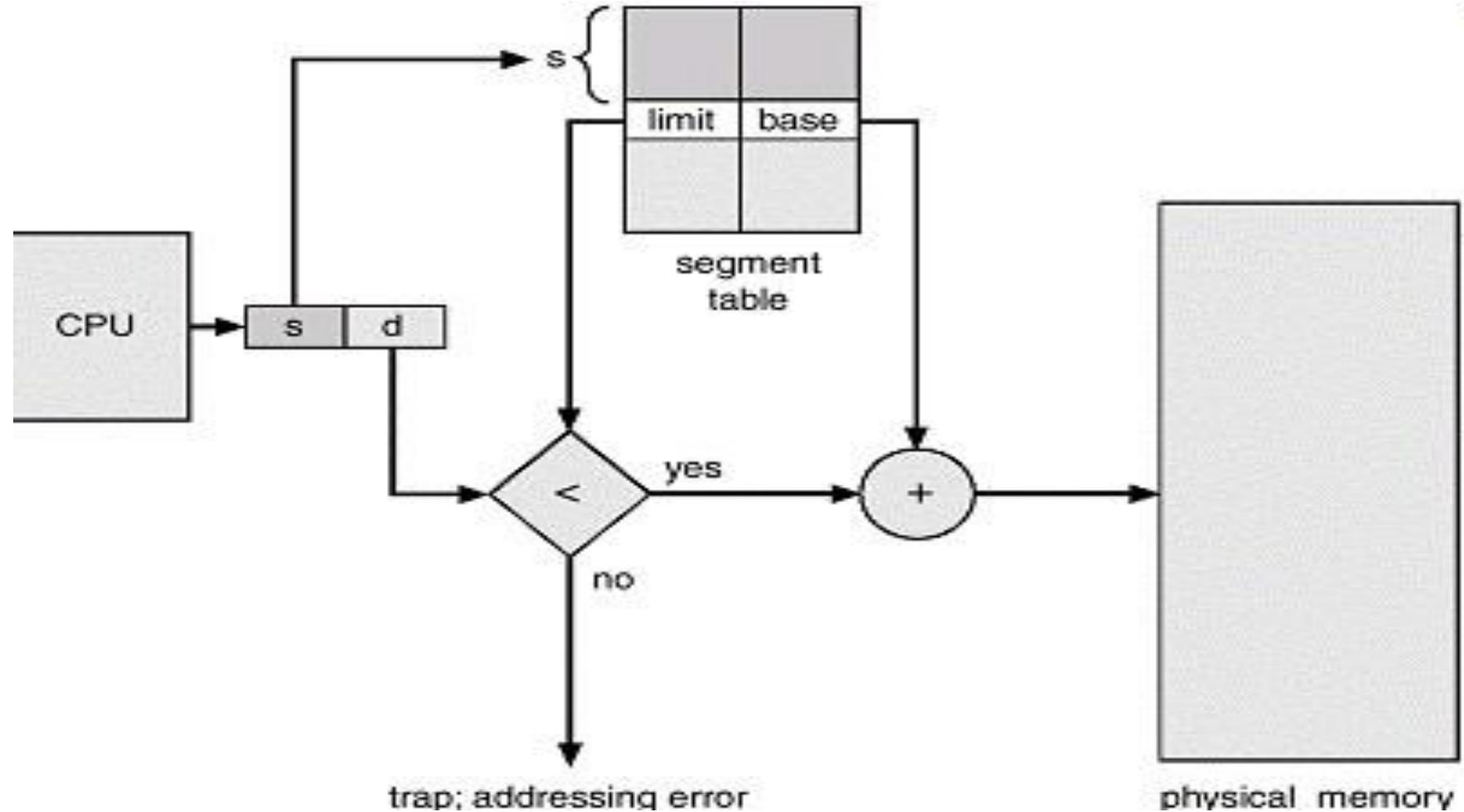


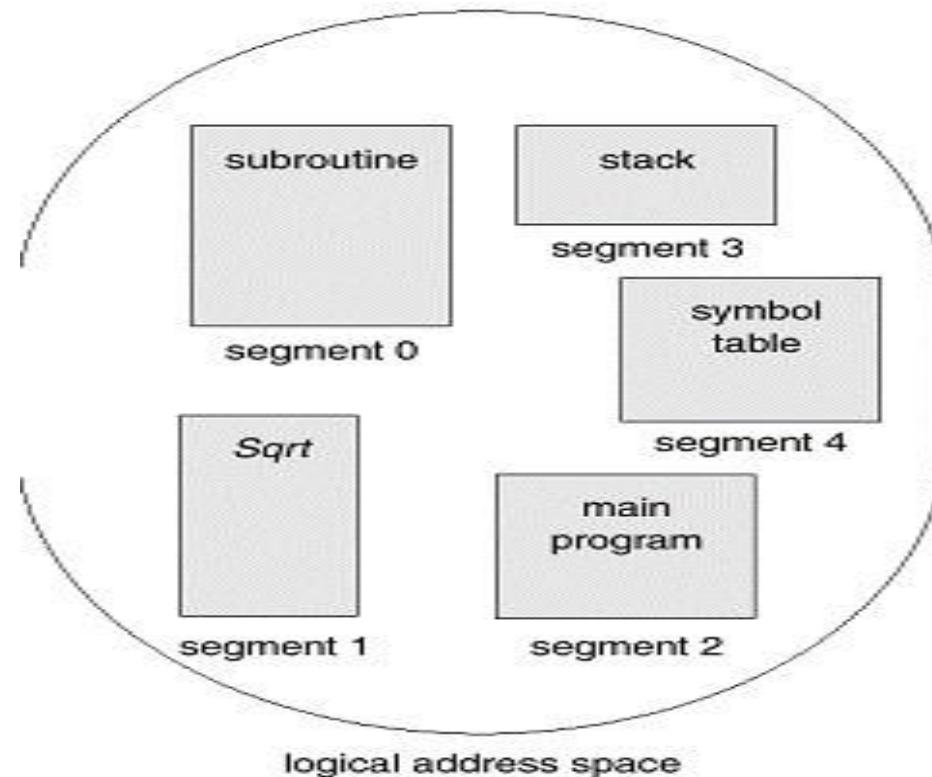
- Concept of Segmentation
  - ▣ Supports user view of main memory
  - ▣ Logical address is a collection of segments, each having a name and a length
  - ▣ A segment is a logical unit such as:
    - main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays



## □ Segment Hardware

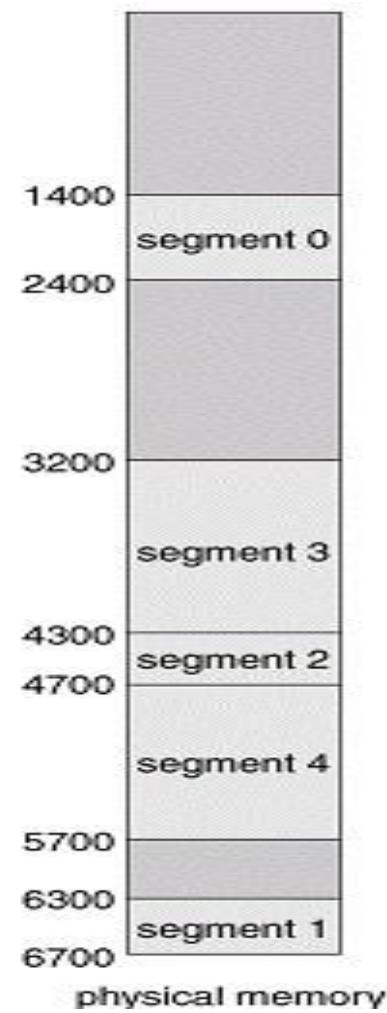
- Mapping of segments to physical address
- Mapping is present in a segment table
  - An entry in a segment table consist of a base and a limit
  - Base corresponds to starting physical address
  - Limit is the length of the segment
- The logical address generated by the CPU consist of a segment number and an offset within the segment
- Segment number is used as an index to segment table
- Offset lie between 0 to limit specified in the table





	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



- External fragmentation
  - Memory allocation Done using bob scheduler
  - Similar to paging but variable sized pages
  - Free segment is small
  - Memory compaction

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

# CHAPTER 8

## Virtual Memory

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

# Virtual Memory

2

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

# Benefits of having Virtual Memory

3

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

- Paging and segmentation helps to implement the concept of multi-programming
- They have a few disadvantages
  - ▣ Require the entire process to be in the main memory before execution can begin
  - ▣ Limitation on the size of the process
- Virtual memory is a technique that allows execution of the processes that may not be entirely in memory
- Mapping of a large virtual space onto a smaller physical memory

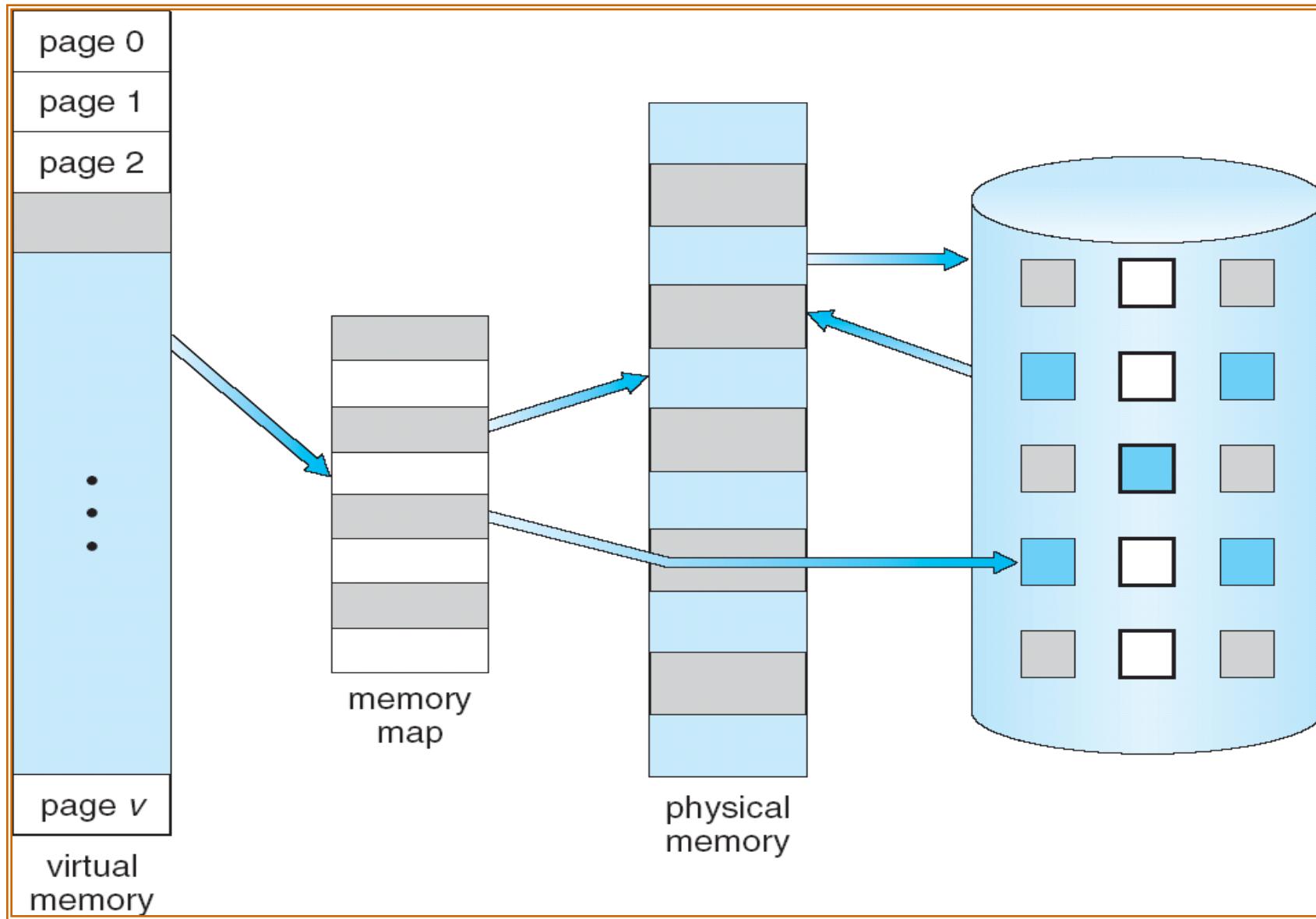


# Need for Virtual Memory Technique

- Many a time the entire process need not be in memory during execution
  - ▣ Code used to handle errors and exceptional cases is executed only in case it occurs
  - ▣ Static declaration for arrays with upper bound very high and 10 % is only used
  - ▣ Features and options provided in the program as a future enhancement
  - ▣ All its parts may not be needed at the same time

# The benefits

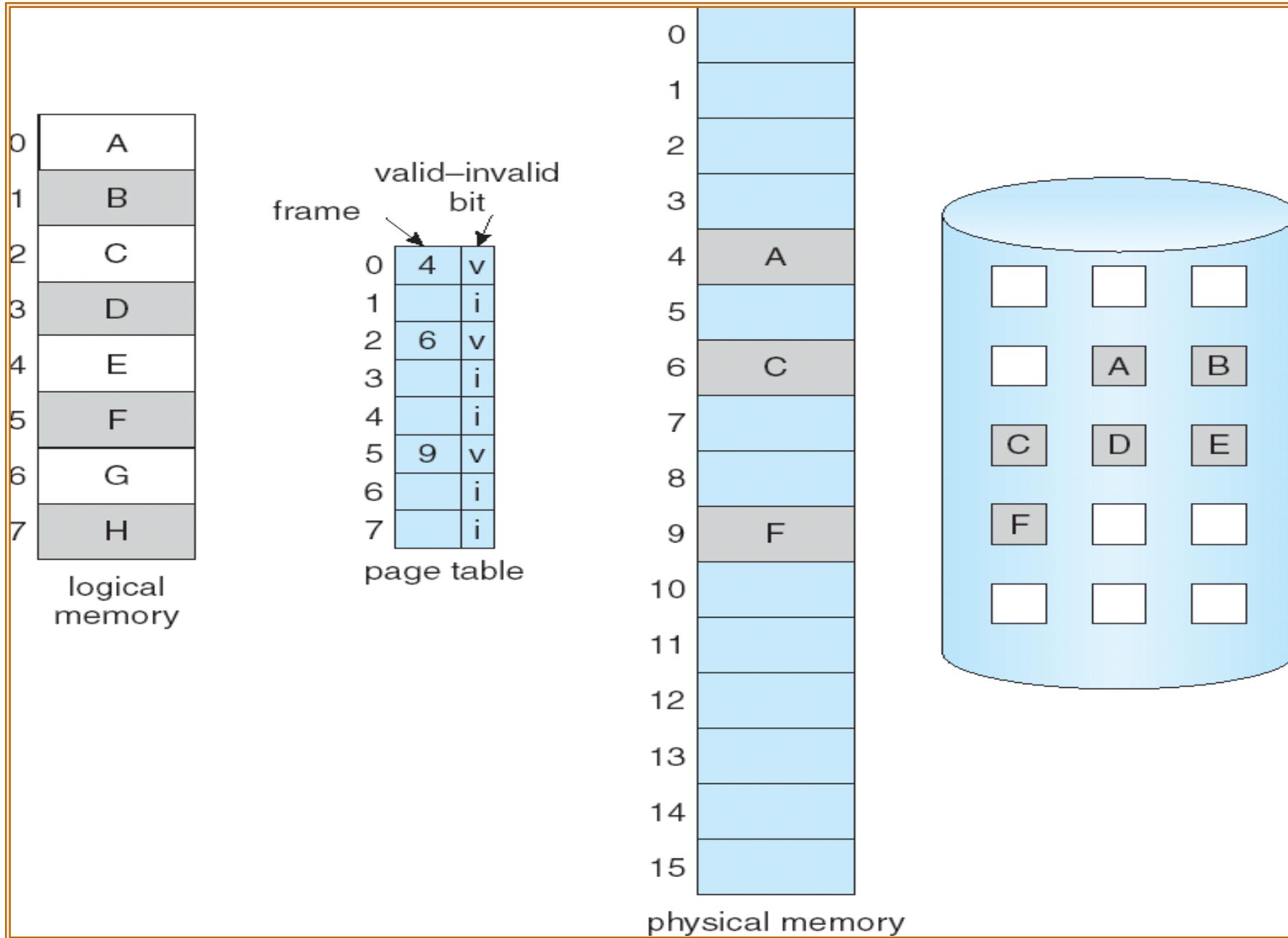
- physical memory is not a constrain for programs
- Memory required for program is less will increase multiprogramming
- I/O time needed for load/swap is less
- Virtual memory implemented using demand paging
- Demand segmentation can be also used
- A combined approach using paged segmentation scheme is also available





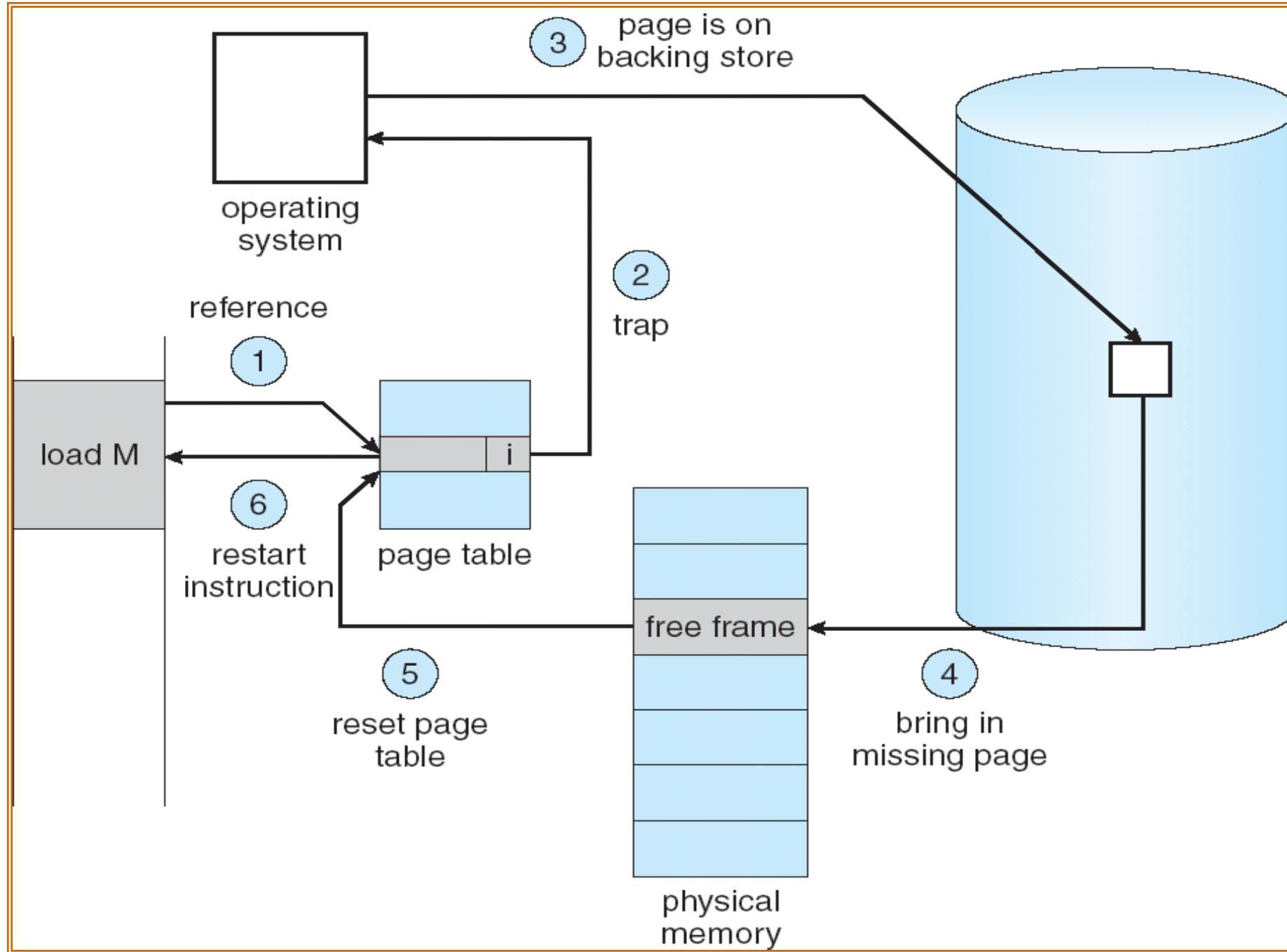
# Demand Paging

- Similar to paging with swapping
- Necessary pages of the process are swapped into memory thereby decreasing swap time physical memory requirement
- The protection valid -invalid bit is used in page table
- If the bit is set, then the corresponding page is valid and also in physical memory
- If the bit is not set, then any of the following can occur



- Process is accessing a page not belonging to it- illegal access
- Legal page but currently not in the memory
- In both cases page fault error occurs
- Error is valid in the first case
- Second case it is a fault by the OS
- Page fault can be handled as follows
  - 1. Check the valid-invalid bit for validity
  - 2. if valid the page is in the memory and the physical address is generated
  - 3. If not an addressing fault occurs
  - 4. Check the page is it in Backing store- if so valid page reference

- 5. search for a free frame
- 6. bring the page into the free frame
- 7. Updated the page table
- 8. restart the execution stopped by addressing fault
- process starts executing with no page in memory
- The very first instruction generates a page fault and brought to memory
- Pure demand paging
  - Never bring in a page into memory until it is required
- A page fault at any point in the fetch -execute cycle of an instruction causes the cycle to be repeated



- Execution starts with none of its pages in memory
- Each of its pages page fault at least once when it is first referenced
- The pages which are not referenced will never be brought into memory
- This saves memory space and load time
- Degree of multiprogramming can be increased
- It will create a situation where no frame is available to load a page from secondary

## □ Options are

- Terminate the process, not a good option
- Swap a process to free all its frame, reduce the degree of multi programming
- Page replacement seems to be the best option

## □ Page replacement

- Find the required page in the backing store
- Find for a free frame
  - If there exists one use it
  - If not find a victim using a page replacement algorithm

- Write the victim into the backing store
- **Modify the page table to reflect a free frame**

- Bring the required page into the free frame
- Update the page table to reflect the change
- Restart the process
- Page fault creates a swap out and swap in
- Swap out is not required every time, overwritten
- Each frame in memory associated with a dirty bit that is reset when the page is brought
- The bit is set whenever the frame is modified
- Look for frame for which the bit is not set



# Page Replacement Algorithm

- Generates as low a number of page faults as possible
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults and page replacements on that string.
- Address Sequence : 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105
- Page size : 100 bytes

- Need to known the number of frames available to the process
- As the number of frames increases the page fault decreases
  - if the frame available is 3 then there would be only 3 page faults
  - One for each page reference
  - If there is only 1 frame then 11 page faults
  - One for every page reference

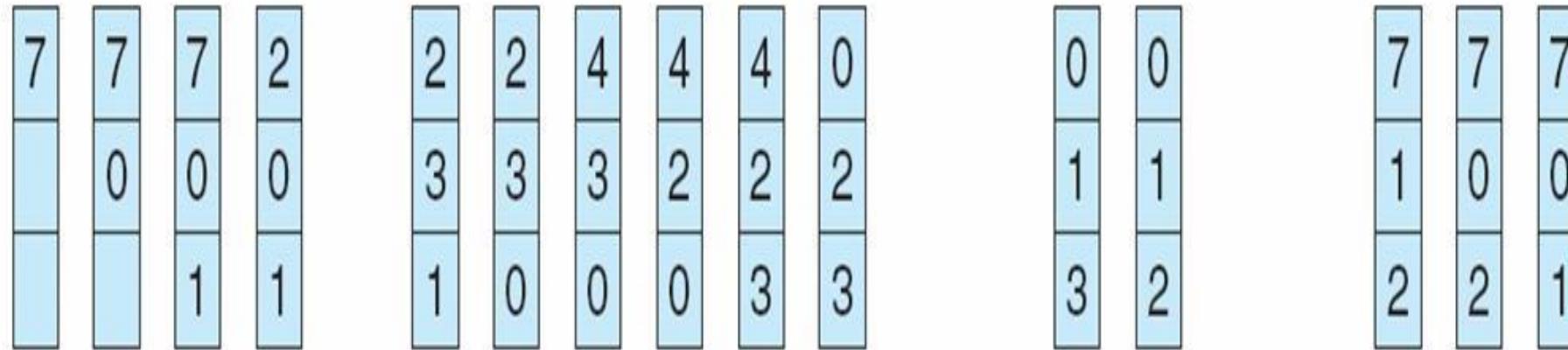
# FIFO Page Replacement Algorithm



## □ Simplest

- The oldest page in memory is the victim reference string

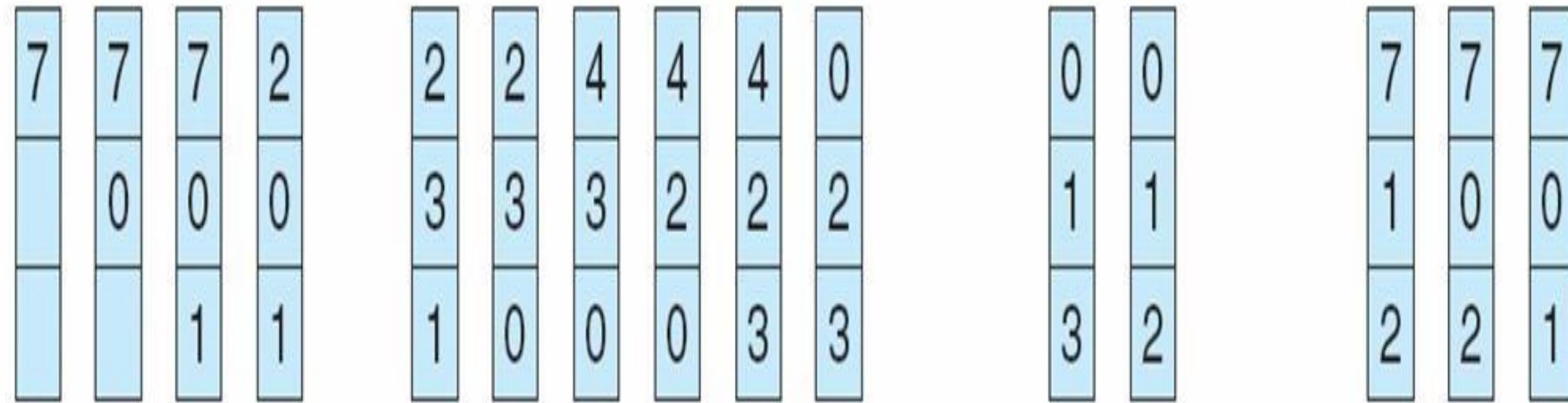
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

reference string

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



page frames

- The performance is not always good
- The page may have a heavily used variable in constant use, such a page swapped out will cause a page fault almost immediately to be brought in
- The number of page faults increases and result in slower process execution
- Implemented using a queue

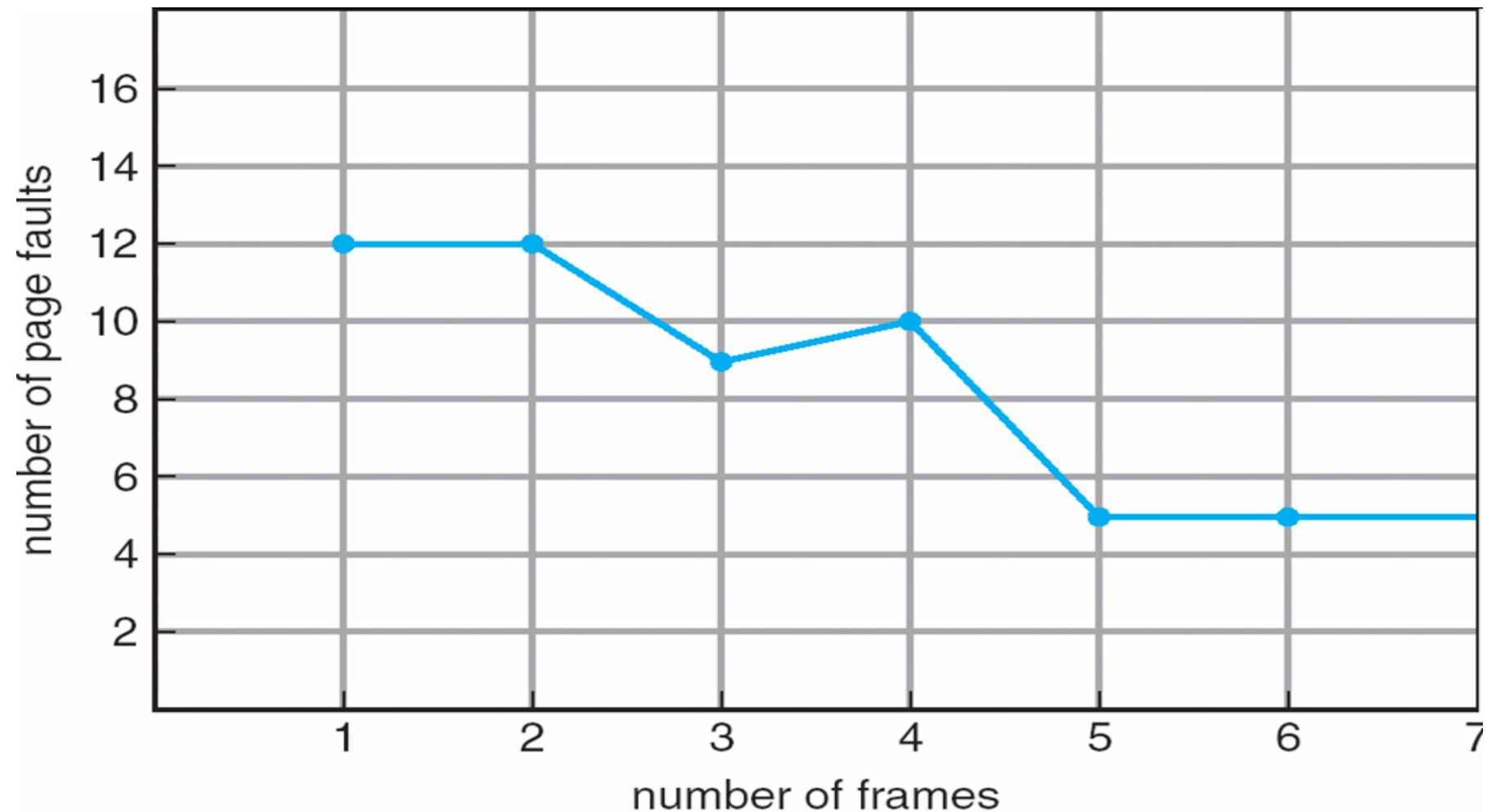
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frames (3 pages can be in memory at a time per process):

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames:
- FIFO Replacement manifests Belady's Anomaly:
- more frames  $\Rightarrow$  more page faults

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

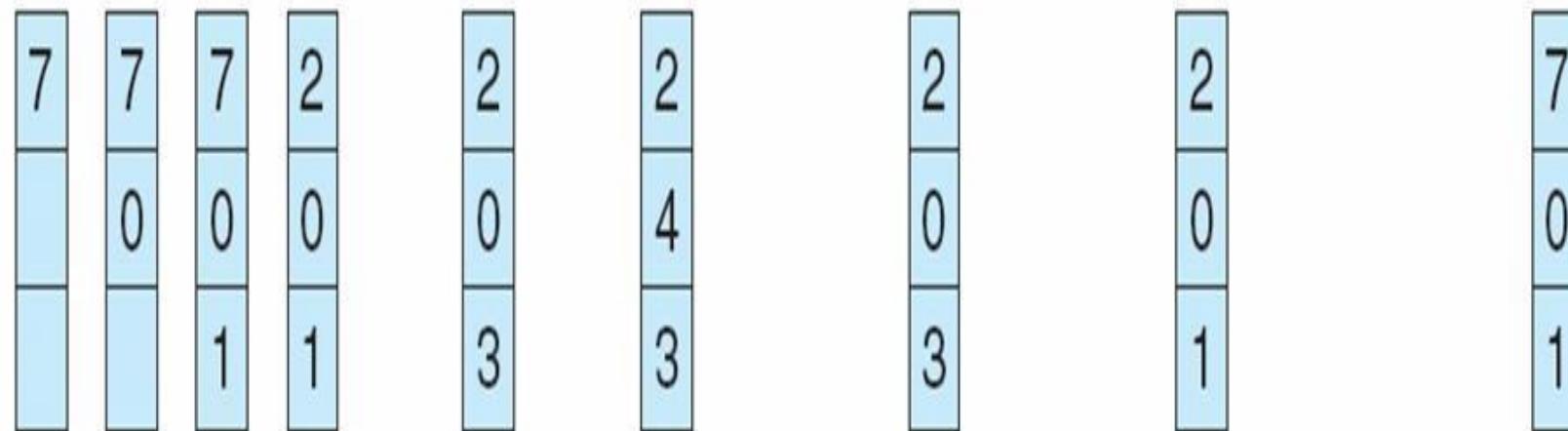


# Optimal Algorithm

- Produces the lowest page fault rate
- Replace the page that will not be used for the longest period of time to come
- Does not suffer from Belady's anomaly
- Impossible to implement (need to know the future) but serves as a Benchmark to compare with the other algorithms

reference string

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



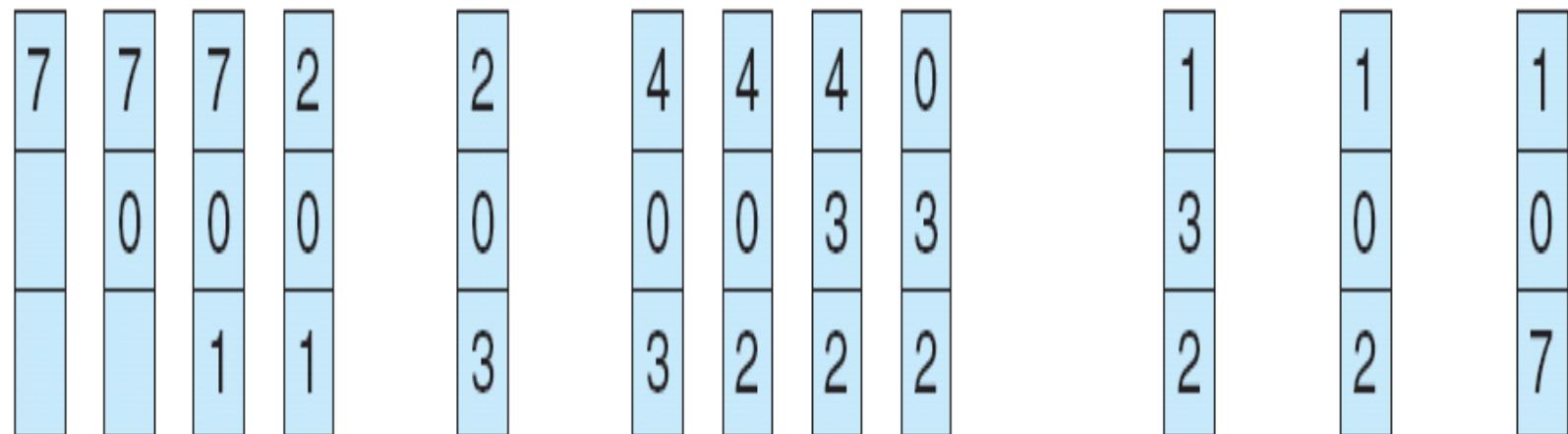
page frames

# LRU Page Replacement Algorithm

- FIFO Looks back
- Optimal Looks ahead
- Here replace the page that has not been used for the longest period of time
- Least Recently used algorithm (LRU)
- An order for the frames by time of last use is required
  - ▣ Use of counters – storing the time when it is used
    - Every page will be having the time
  - ▣ Use of stack
    - Recently used will be on the top of the stack
    - The bottom will be the LRU page

reference string

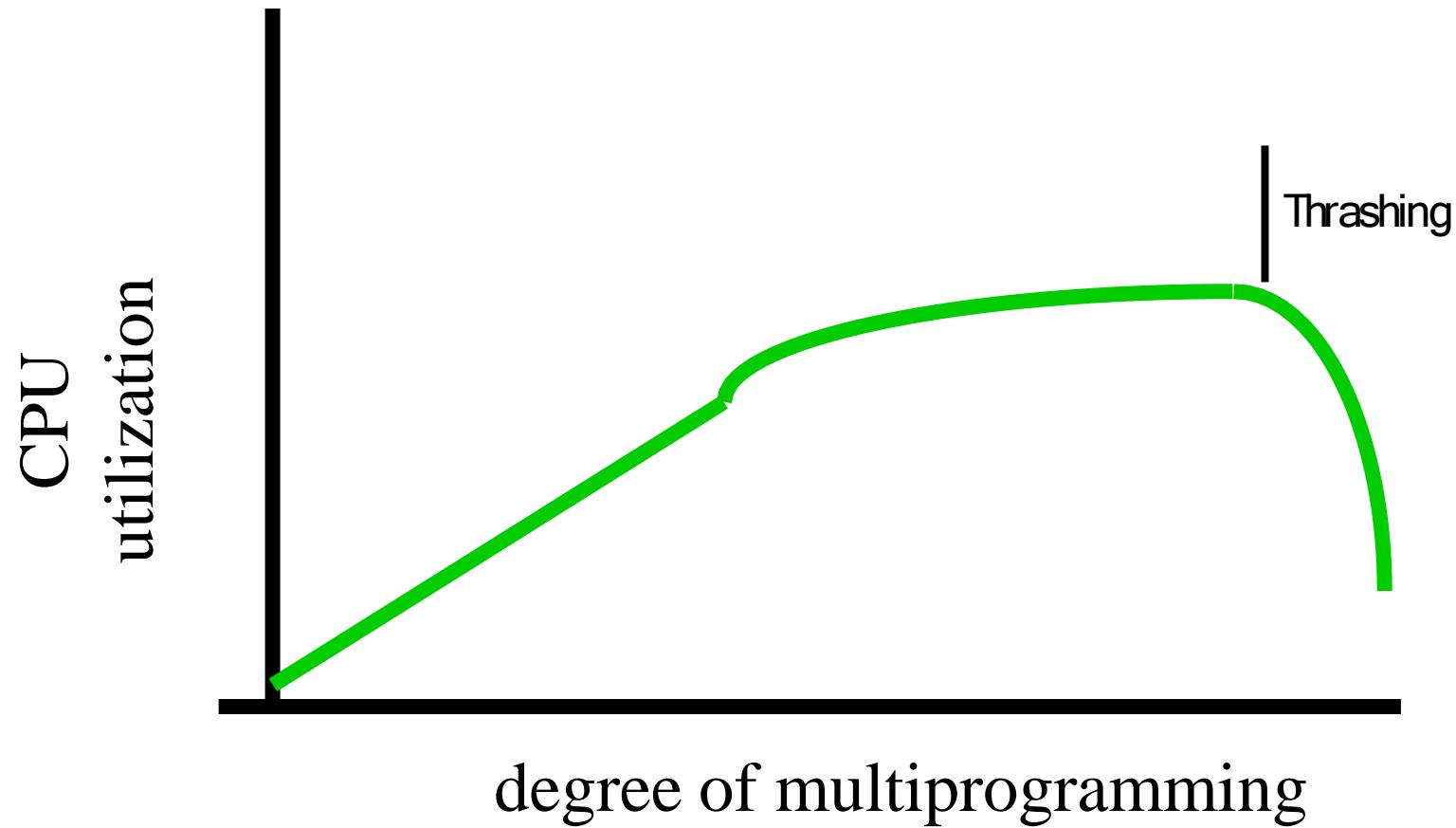
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

# Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - ▣ low CPU utilization
  - ▣ OS thinks it needs increased multiprogramming
  - ▣ adds another process to system
- *Thrashing* is when a process is busy swapping pages in and out
- The high paging activity is called as Thrashing



## □ How do we fix thrashing?

- *Working Set Model*
- *Page Fault Frequency*
- **Working Set Model**
  - Based on locality model
  - Working set window is defined
  - It is a parameter that maintains the most recent page reference
  - The set of most recent page reference is called the working set
  - An active page always find itself in the working set
  - A page not used will drop off the working set



# Page Fault Frequency

- Upper bound & lower bound
- Exceeds the upper bound one more frame is allocated
- Falls below lower bound then a frame already allocated can be removed
- Thus monitoring the page fault rate helps prevent thrashing

# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

# CHAPTER 9

## File Management

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121

# File management

2

File management is organizing and keeping track of files and folders. It helps you stay organised so information is easy to locate and use. Using the file management tools, you can save files in folders with appropriate names so these files can be found or identified easily, create new folders quickly for recognition of information, delete unnecessary files and folders, search for files and folders, create shortcuts of files for quick access and compress files and folders to save space

# File Management Objectives

3



- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Minimize the potential for lost or destroyed data
- Provide a standardized set of I/O interface routines to user processes
- Provide I/O support for multiple users in the case of multipleuser systems

# Minimal User Requirements

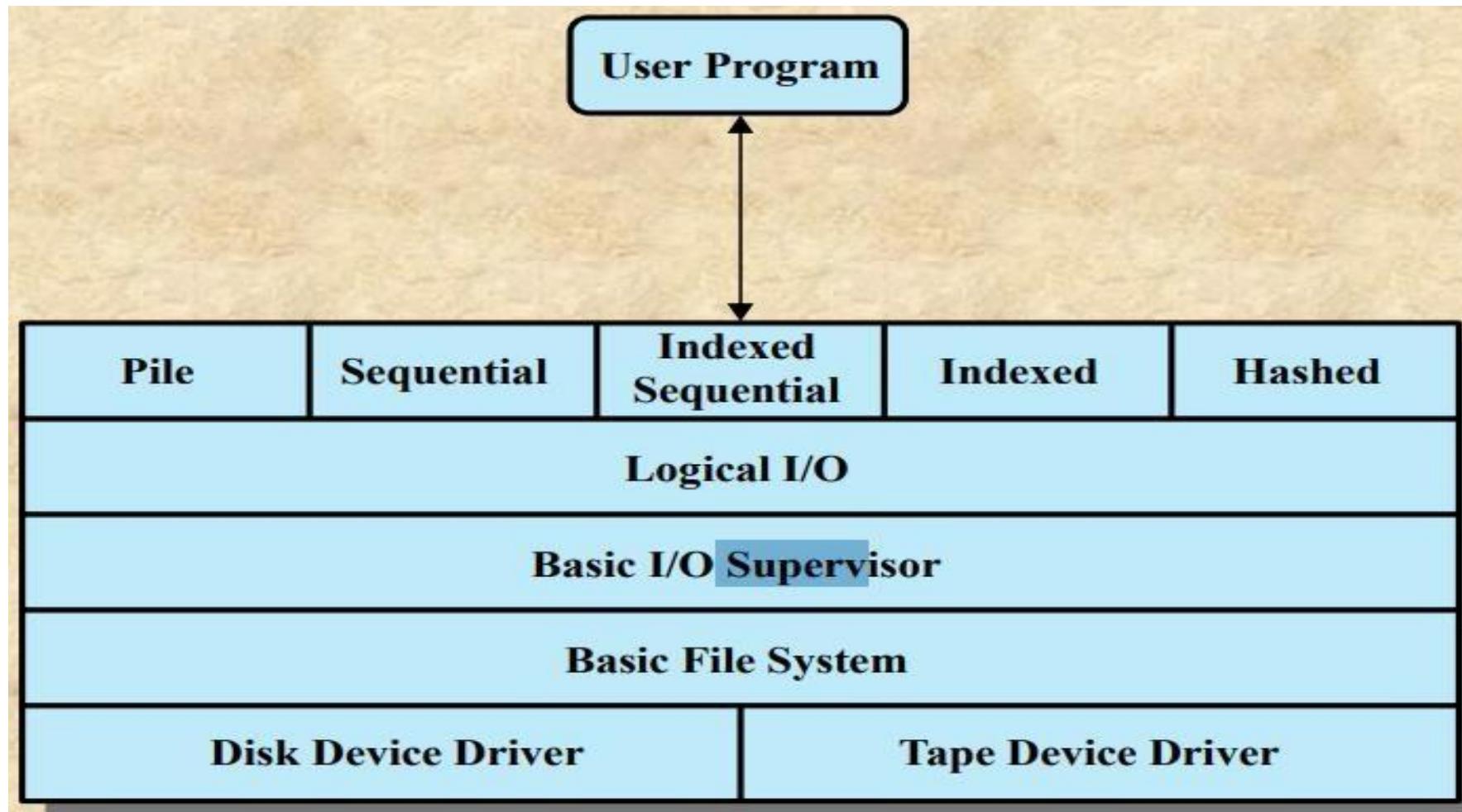
4

Each user

1. should be able to create, delete, read, write and modify files
2. may have controlled access to other users' files
3. may control what type of accesses are allowed to the files
4. should be able to restructure the files in a form appropriate to the problem
5. should be able to move data between files
6. should be able to back up and recover files in case of damage
7. should be able to access his or her files by name rather than by numeric identifier

## Figure 9.1 File System Software Architecture

5



# Device Drivers

6

- ❑ Lowest level
- ❑ Communicates directly with peripheral devices
- ❑ Responsible for starting I/O operations on a device
- ❑ Processes the completion of an I/O request
- ❑ Considered to be part of the operating system

# Basic File System

7

- Also referred to as the physical I/O level
- Primary interface with the environment outside the computer system
- Deals with blocks of data that are exchanged with disk or tape systems
- Concerned with the placement of blocks on the secondary storage device
- Concerned with buffering blocks in main memory
- Considered part of the operating system

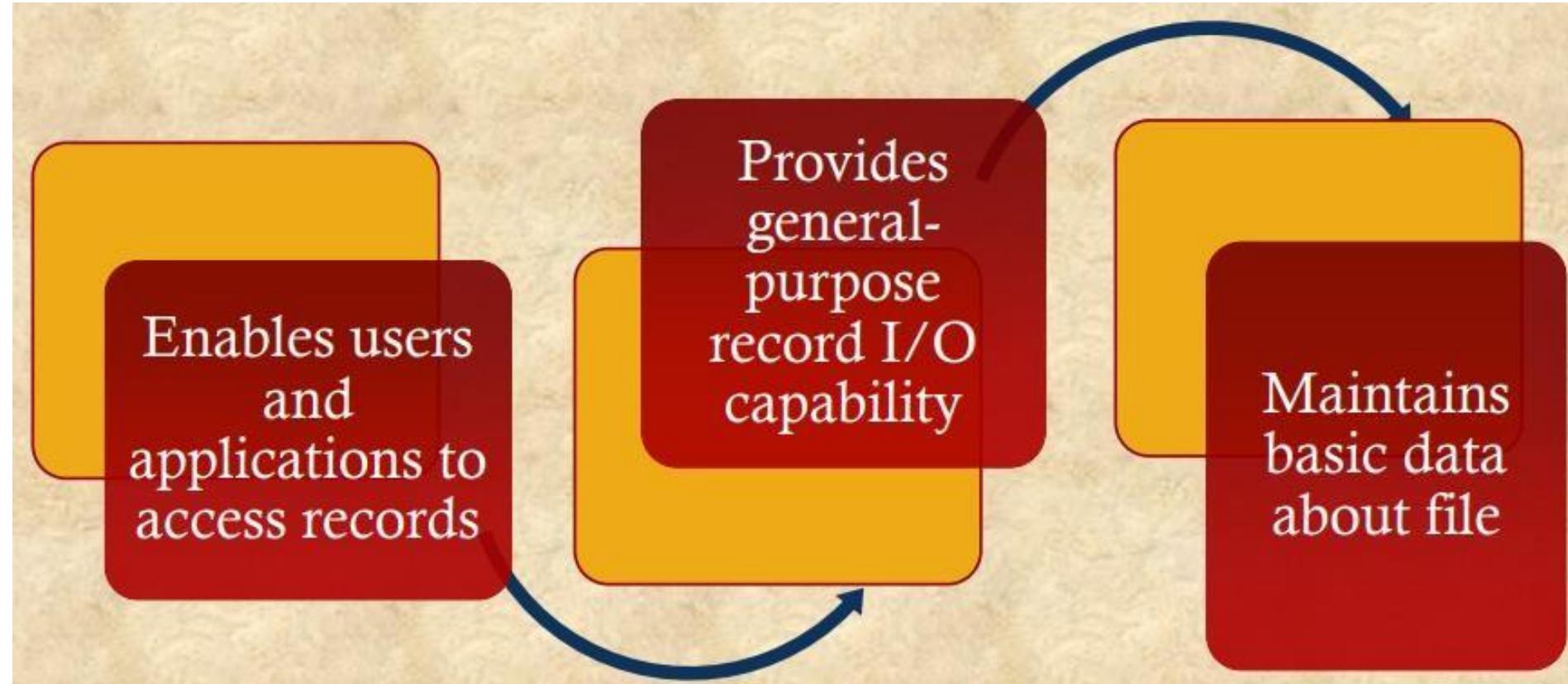
# Basic I/O Supervisor

8

- Responsible for all file I/O initiation and termination
- Control structures that deal with device I/O, scheduling, and file status are maintained
- Selects the device on which I/O is to be performed
- Concerned with scheduling disk and tape accesses to optimize performance
- I/O buffers are assigned and secondary memory is allocated at this level
- Part of the operating system

# Logical I/O

9



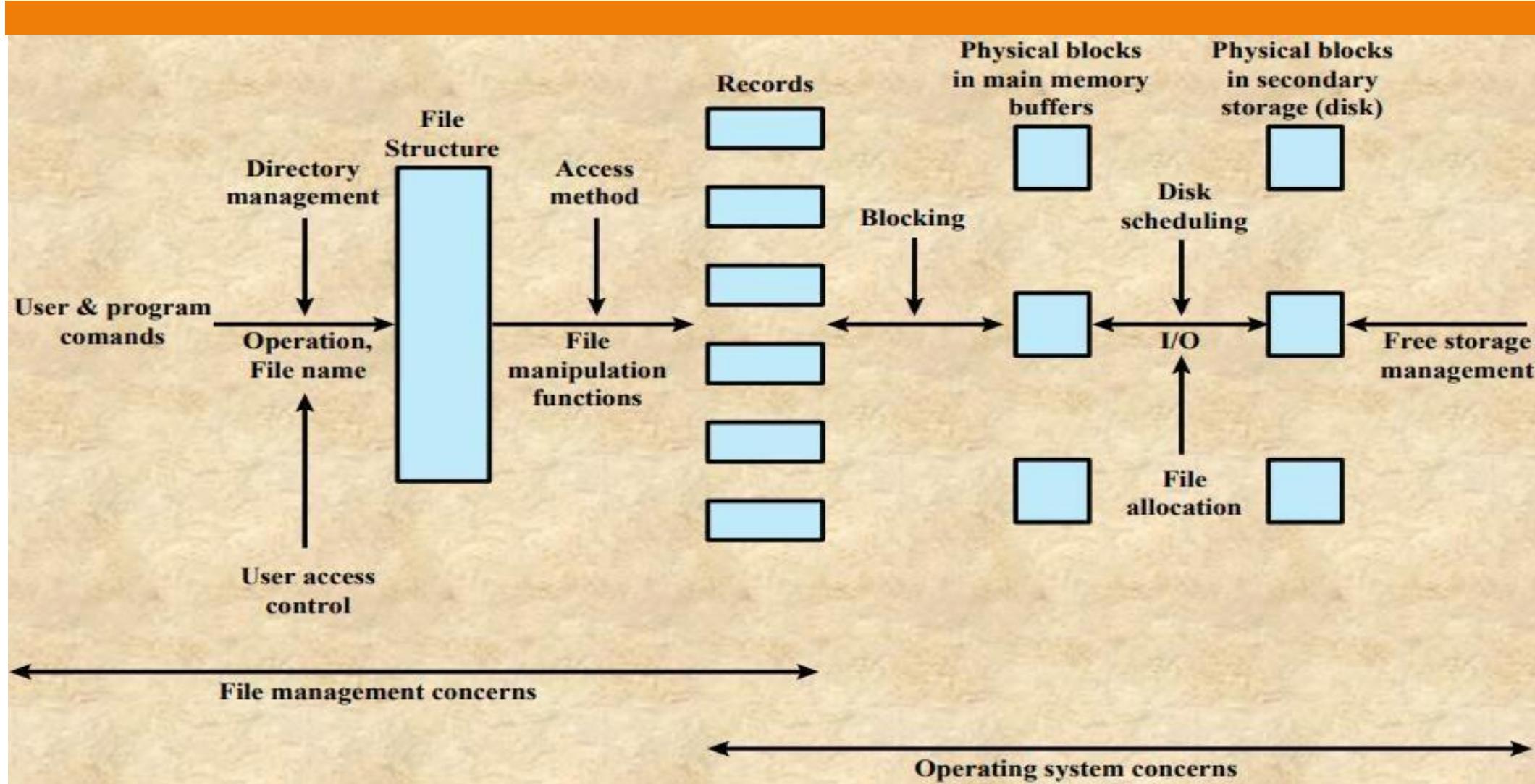
# Access Method

10

- ❑ Level of the file system closest to the user
- ❑ Provides a standard interface between applications and the file systems and devices that hold the data
- ❑ Different access methods reflect different file structures and different ways of accessing and processing the data

## Figure 9.2 Elements of File Management

11



# File Organization and Access

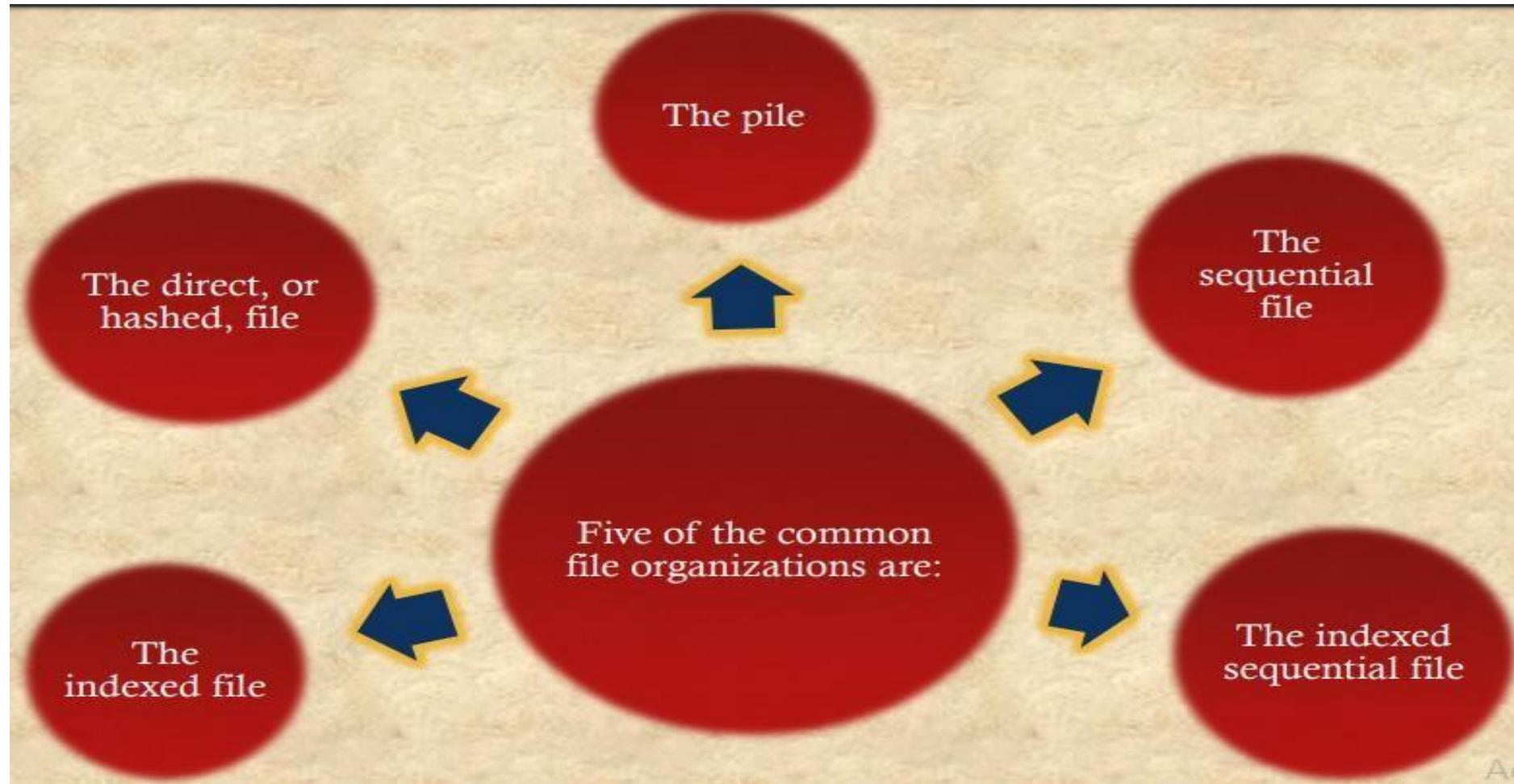
12

- ❑ File organization is the logical structuring of the records as determined by the way in which they are accessed
- ❑ In choosing a file organization, several criteria are important:
  - short access time
  - ease of update
  - economy of storage
  - simple maintenance
  - reliability
- ❑ Priority of criteria depends on the application that will use the file



# File Organization Types

13



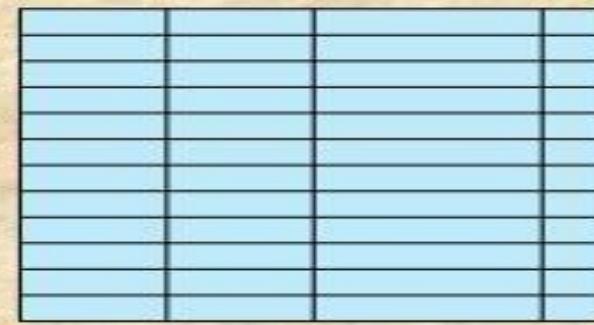
# Figure 9.3 Common File Organizations

14



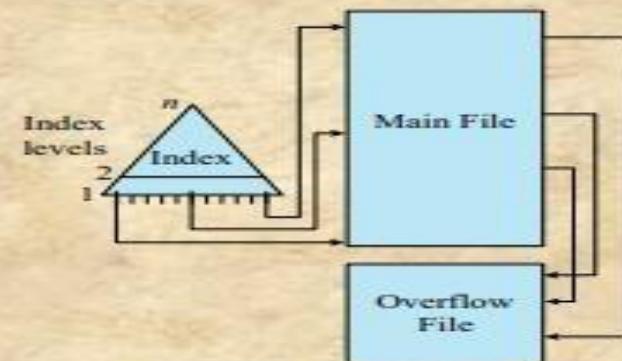
Variable-length records  
Variable set of fields  
Chronological order

**(a) Pile File**

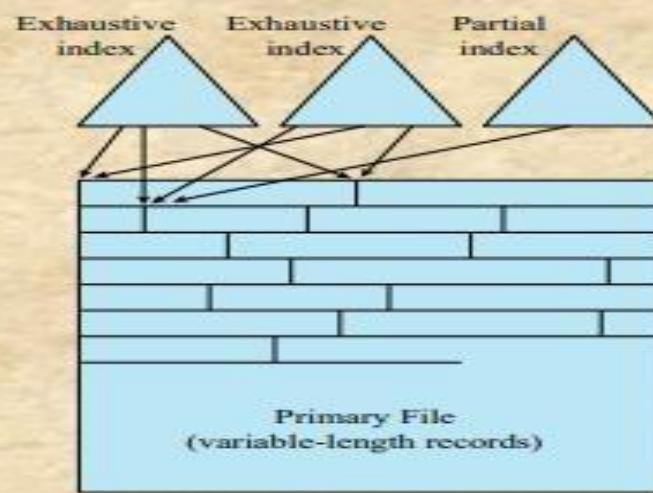


Fixed-length records  
Fixed set of fields in fixed order  
Sequential order based on key field

**(b) Sequential File**



**(c) Indexed Sequential File**



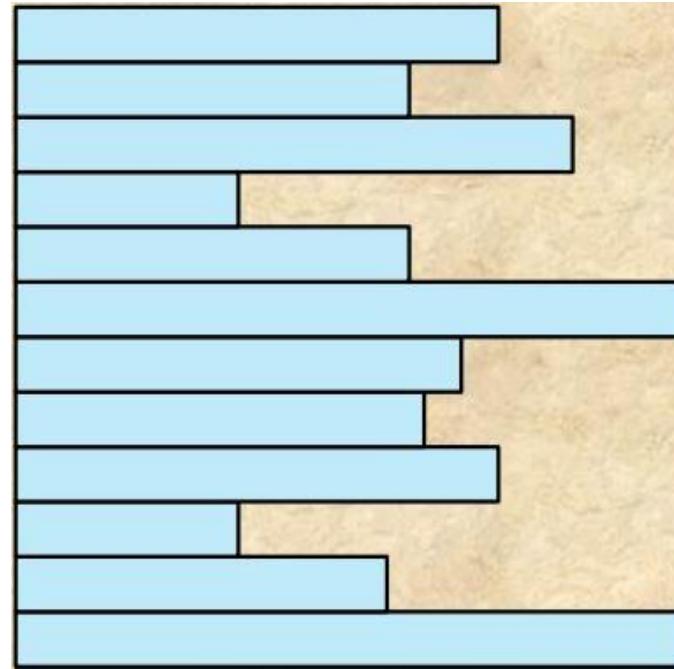
**(d) Indexed File**

# (a) Pile File

15

## The Pile

- Least complicated form of file organization
- Data are collected in the order they arrive
- Each record consists of one burst of data
- Purpose is simply to accumulate the mass of data and save it
- Record access is by exhaustive search



Variable-length records  
Variable set of fields  
Chronological order

## **(b) Sequential File**

# The Sequential File

- Most common form of file structure
  - A fixed format is used for records
  - Key field uniquely identifies the record
  - Typically used in batch applications
  - Only organization that is easily stored on tape as well as disk

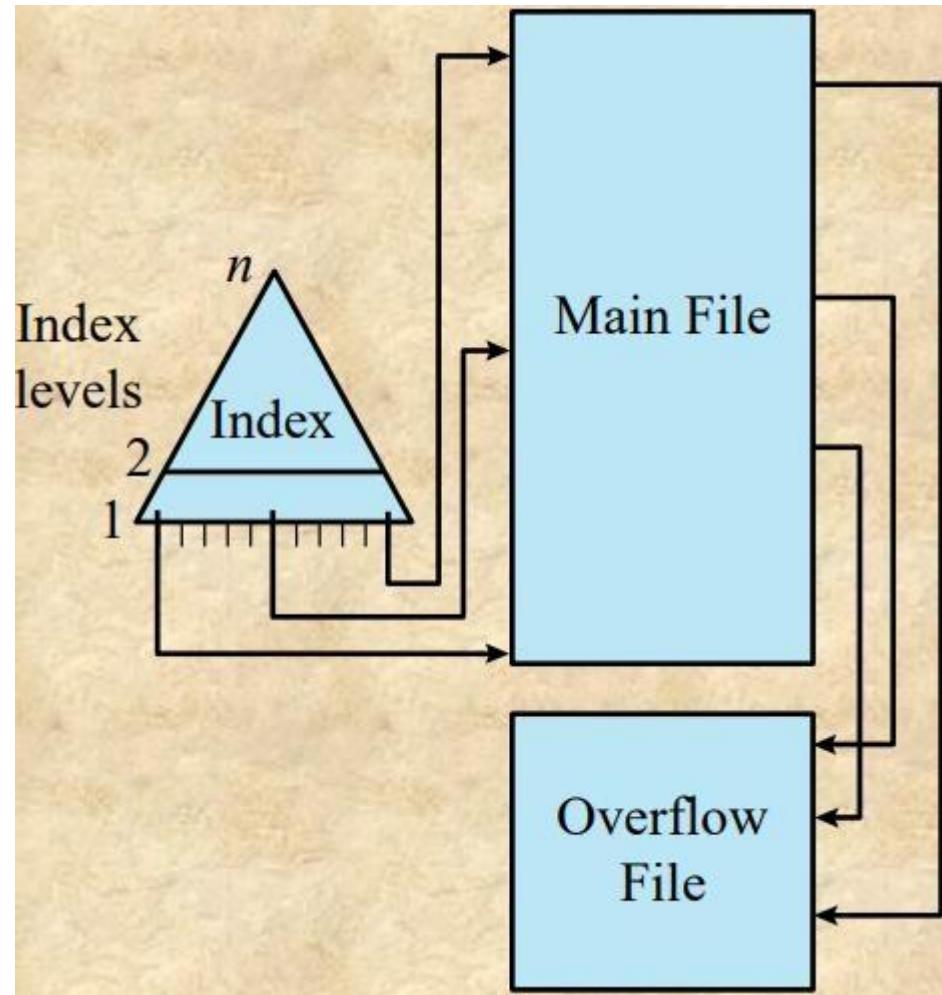
Fixed-length records Fixed set  
of fields in fixed order  
Sequential order based on  
key field

## (c) Indexed Sequential File

17

### Indexed Sequential File

- Adds an index to the file to support random access
- Adds an overflow file
- Greatly reduces the time required to access single record
- Multiple levels of indexing can be used to provide greater efficiency in access

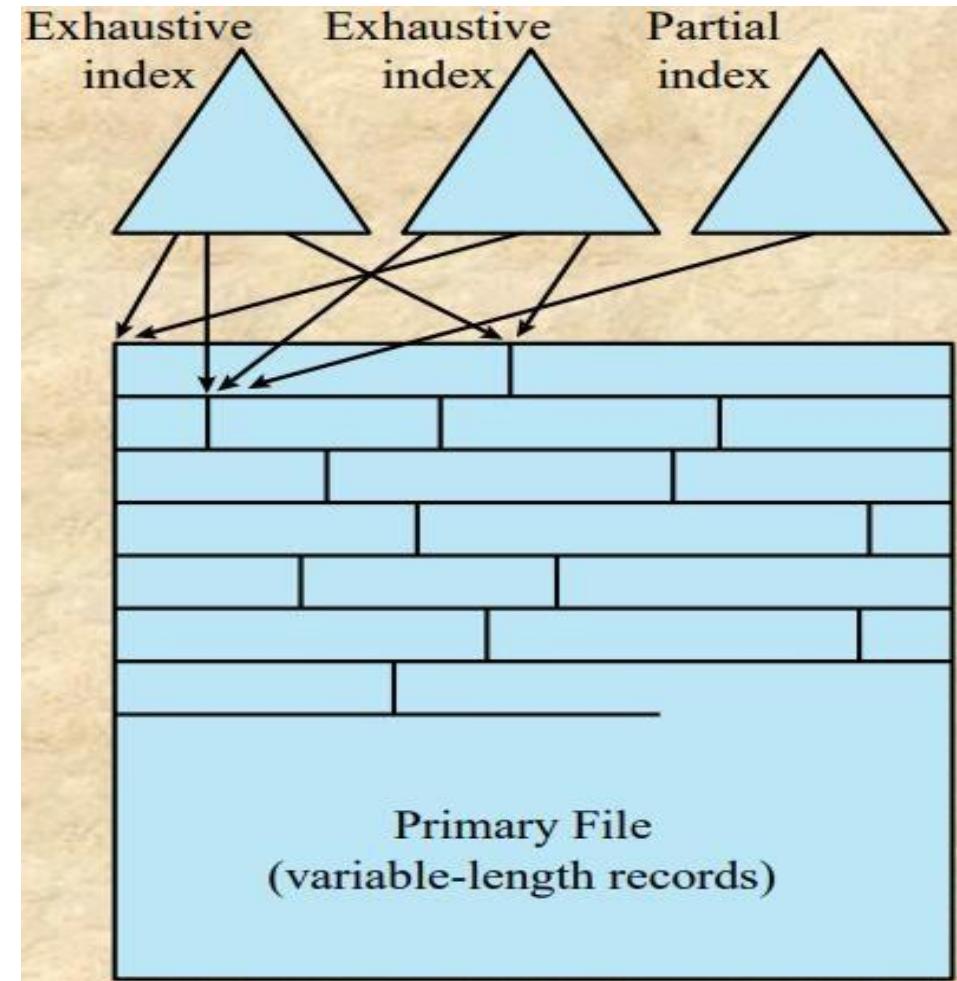


## (d) Indexed File

18

### Indexed File

- Records are accessed only through their indexes
- Variable-length records can be employed
- Exhaustive index contains one entry for every record in the main file
- Partial index contains entries to records where the field of interest exists
- Used mostly in applications where timeliness of information is critical
- Examples would be airline reservation systems and inventory control systems



# Thank you



**ISBAT**  
**UNIVERSITY**  
BLENDED LEARNING PLATFORM

# CHAPTER 10

## Implementing File System

CSE122\_BAI1208\_BCS1211\_BIT1211\_DIT1121



# Concept of a File

- OS provides a uniform logical view of information stored in different media ( magnetic disks, tapes, optical disk etc)
- The OS abstracts from the physical properties of its storage devices to define a logical storage unit called a file
- These files mapped on to physical devices by the OS during usage
- Storage devices are non volatile

- A file is a collection of related information recorded on a secondary storage
- Collection of records
- Depending on the contents of a file, each file has a predefined structure
- Attributes of a File
  - Type
  - Location : pointer to the information
  - Size
  - Protection : user access
  - Time, date and user id

## □ Operations on Files

### □ Creating a File

- Space allocation
- Entry to directory

### □ Writing a file

- Name
- Content

### □ Reading a file

### □ Repositioning within a file : Also known as file seek

### □ Deleting a file

- Resources released
- Content the directory erased

### □ Truncating a file : Attributes remains the same

## □ **Types of files**

Include type of file as a part of the file name

- Name : name + extension
- Executable files have a .com, .exe, .bat

## □ **Structure of file**

- File types are an indication of the internal structure of a file
- For execution the operating system need to known the structure
- In Unix it treats each file as a sequence of bytes. It is up to application program to interpret
- DISK I/O is always in terms of blocks

- A block is physical unit of storage
- All blocks are of same size , 512 bytes
- No of logical records are packed into one physical block
- File access is always in terms of blocks
- Mapping is usually
- Total file size is not always an exact multiple of the block size, some part of the last block is wasted
- Internal fragmentation
- Larger the physical block size the greater is the internal fragmentation

# File Access Methods

- Sequential Access
  - Accessed one record after the other sequentially
  - Based on tape model
  - Best suited when most of the records in the file are to be processed

## Direct Access

- May not require to process every record in a file
- Can only access if we are having a key value to a record
- Device has to be direct access, allows random access of any file block
- Database is of this type
- File has to be defined as sequential or direct at the time of creation
- If direct, sequential access is possible

## □ Indexed sequential Access

- Slight modification of direct access
- Combination of both
- Access a file direct first then sequentially from that points onwards
- Involves maintaining an index
- Index is a pointer to a block
- Direct access to the index will provide information related to the record
- That information is used for accessing the file

# Directory Structure



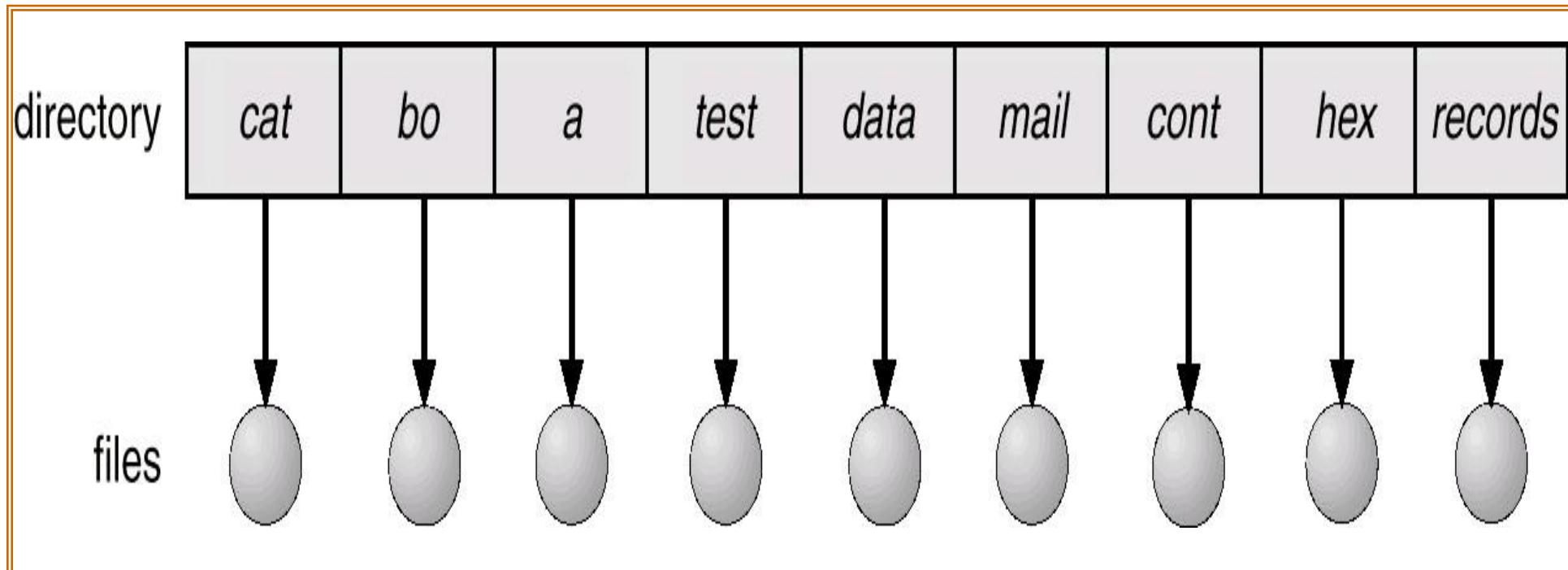
- Files systems are very large
- Usually a two level organization is done
  - I File system is divided into partitions (virtual disks) each partition is considered as a separate storage device
  - I Each partition has information about the file in it, it's a table of contents known as directory

- Directory maintains information
  - Name of file
  - Location
  - Size
  - type

## □ Single Level Directory

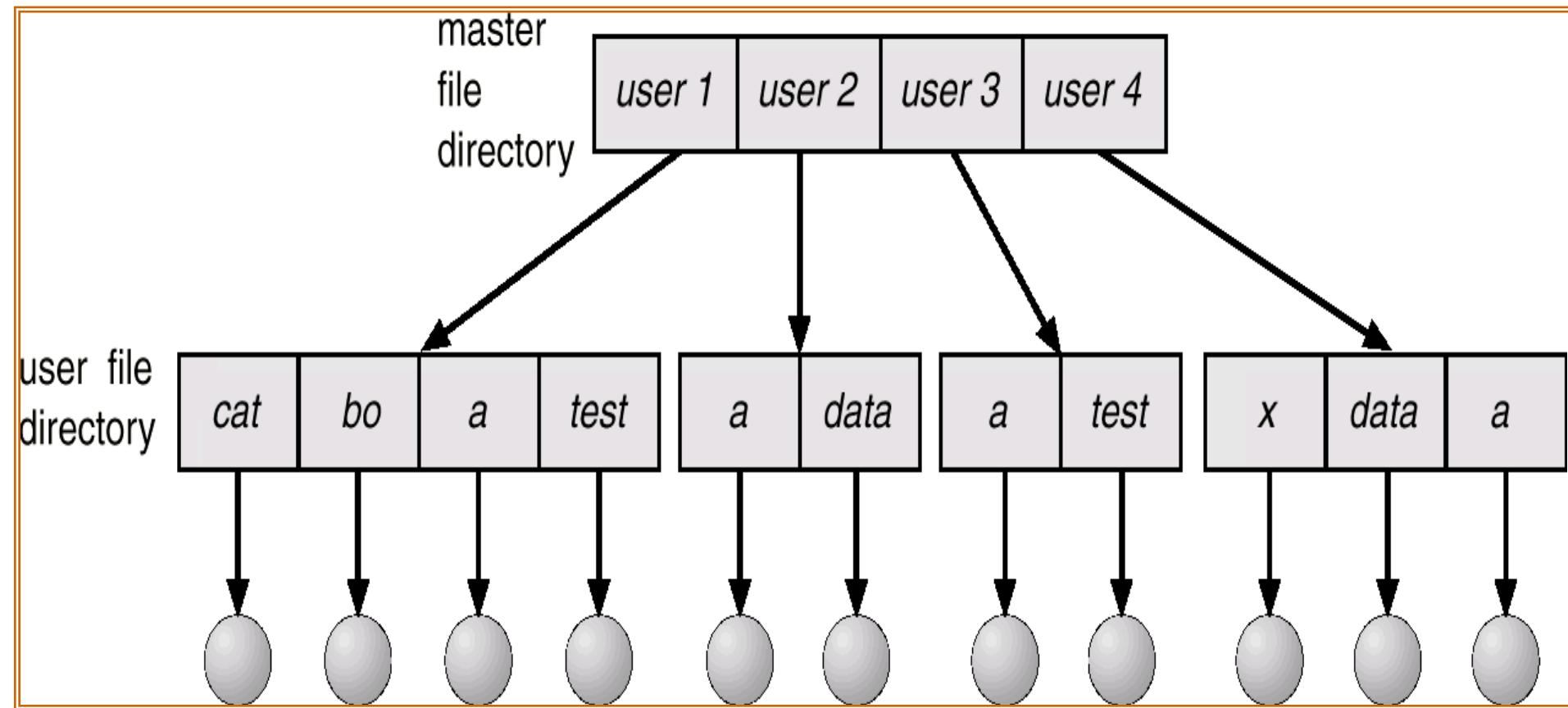
Simple directory structure that is very easy to support

- All files resides in one and the same directory
- Only one directory to list all the files
- No two files can have the same name



## □ Two Level Directory

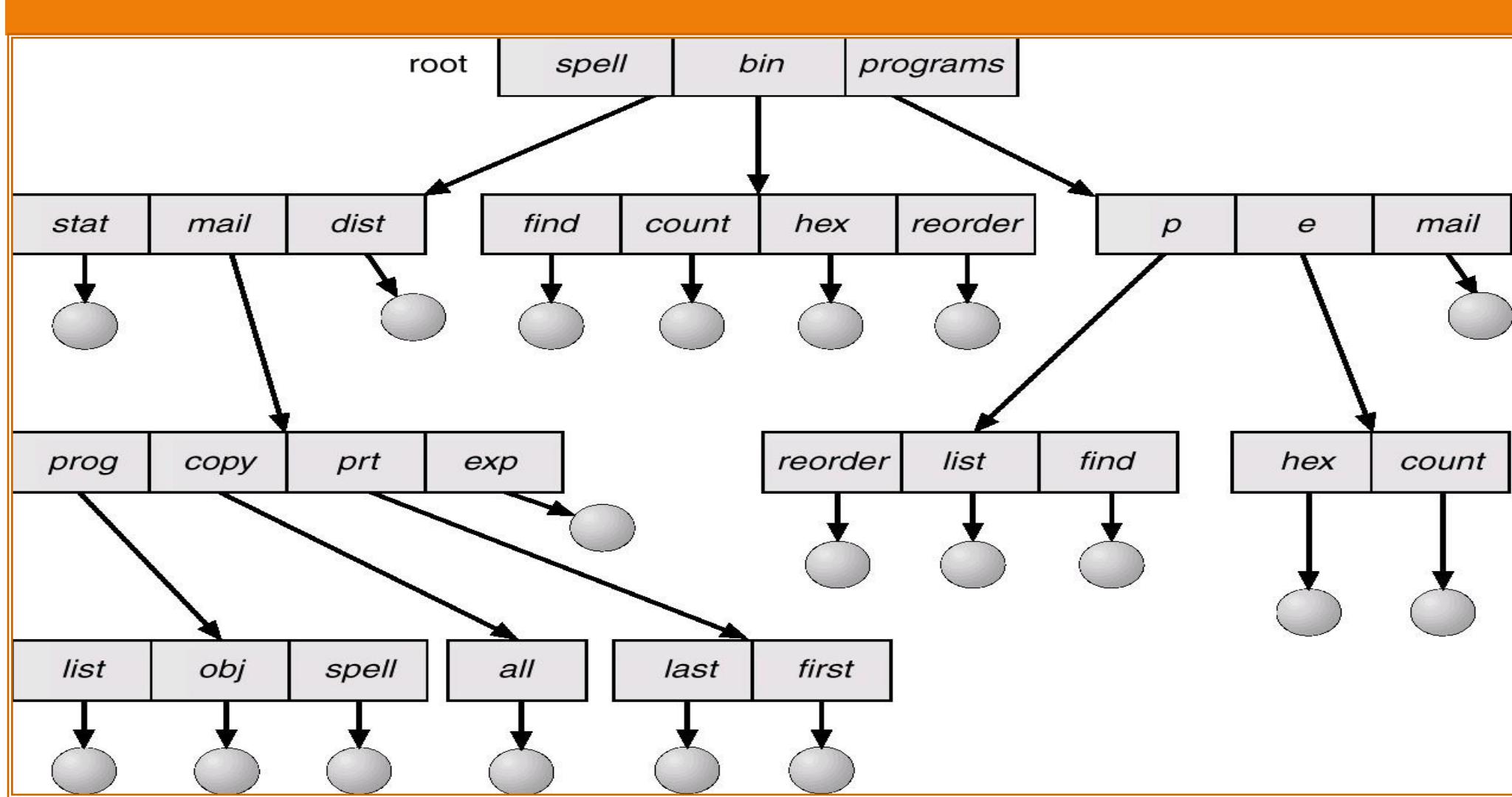
- | Separate directories for each user
- | Directory structure of each user is similar in structure and maintains file information about files present in that directory
- | One master directory for partition
  - It has entries for each user
- | Users are isolated from one another
- | File maintenance is easy
- | Access to a file is through user name and file name (path)
- | Path uniquely defines a file



# Tree Structure



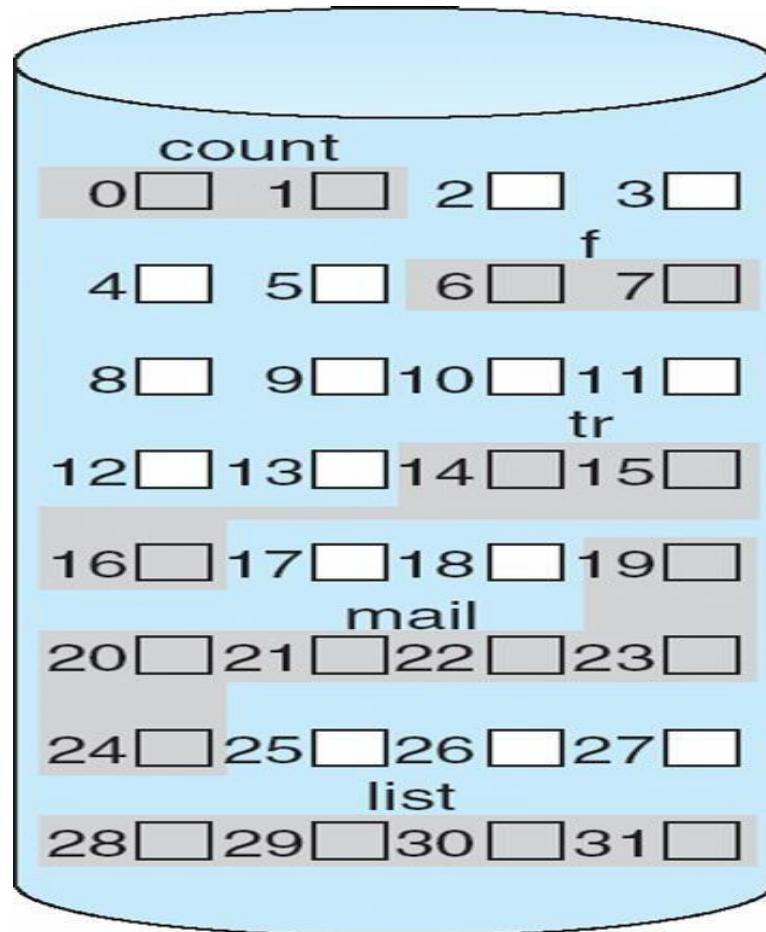
- A two level directory is a tree of height two
- The generalization allows users to organizes files with user directories into subdirectories
- Every file has a unique path
- Absolute path :- path from root down to the file
- Relative path :- begins with current directory
- Directory is treated as a yet another file in the directory
- Deleting a directory (two options)
  - Delete all file then delete the directory
  - Delete all entries when deleting directory
- May be a recursive process



- Allocation of disk space to files
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

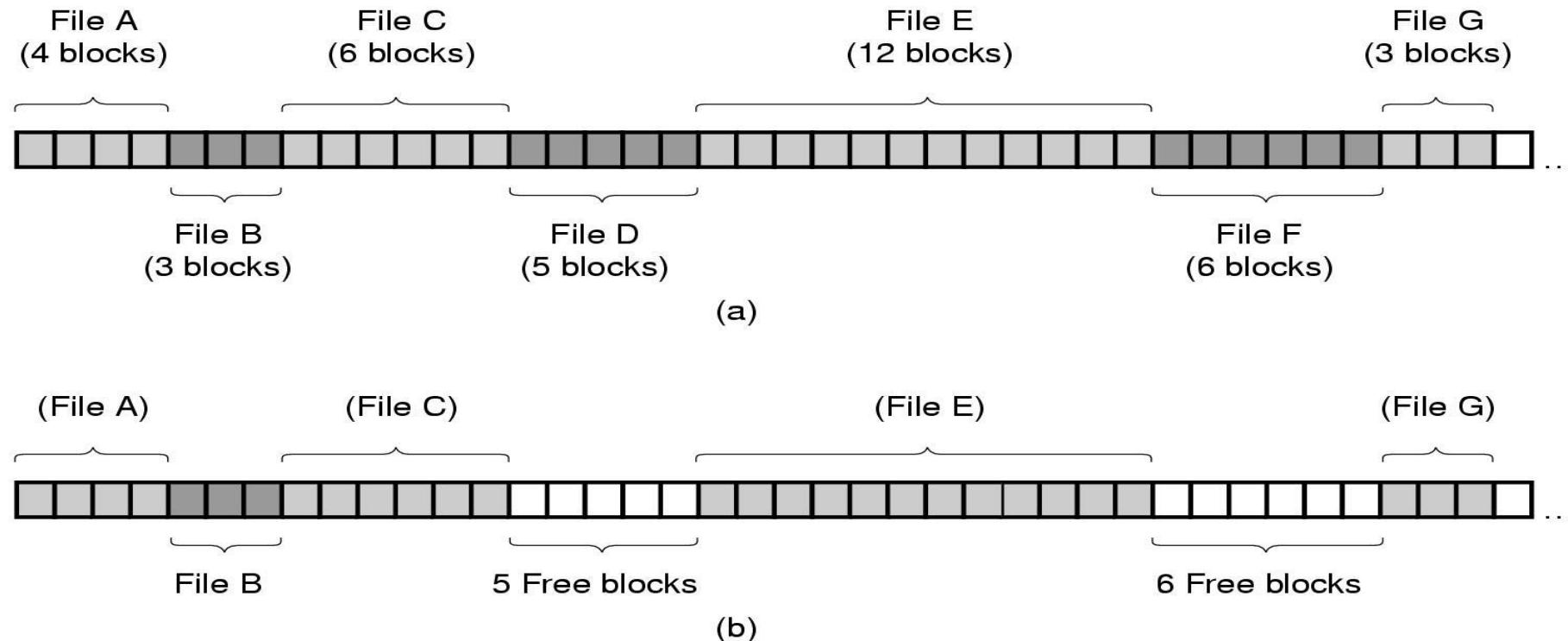
## □ Contiguous allocation

- Requires files to occupy contiguous blocks in the disk
- Disk access time is reduced
- Disk head movement is restricted to only one track
- Simple: only starting location (block #) and length (number of blocks) required.
- Enables random access.
- Wasteful of space (dynamic storage-allocation problem).



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

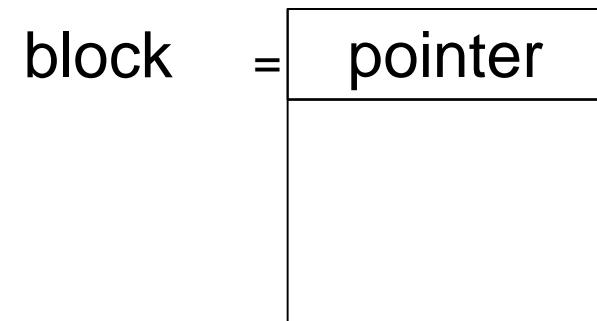


- Contiguous allocation of disk space for 7 files.
- (b) The state of the disk after files D and F have been removed.

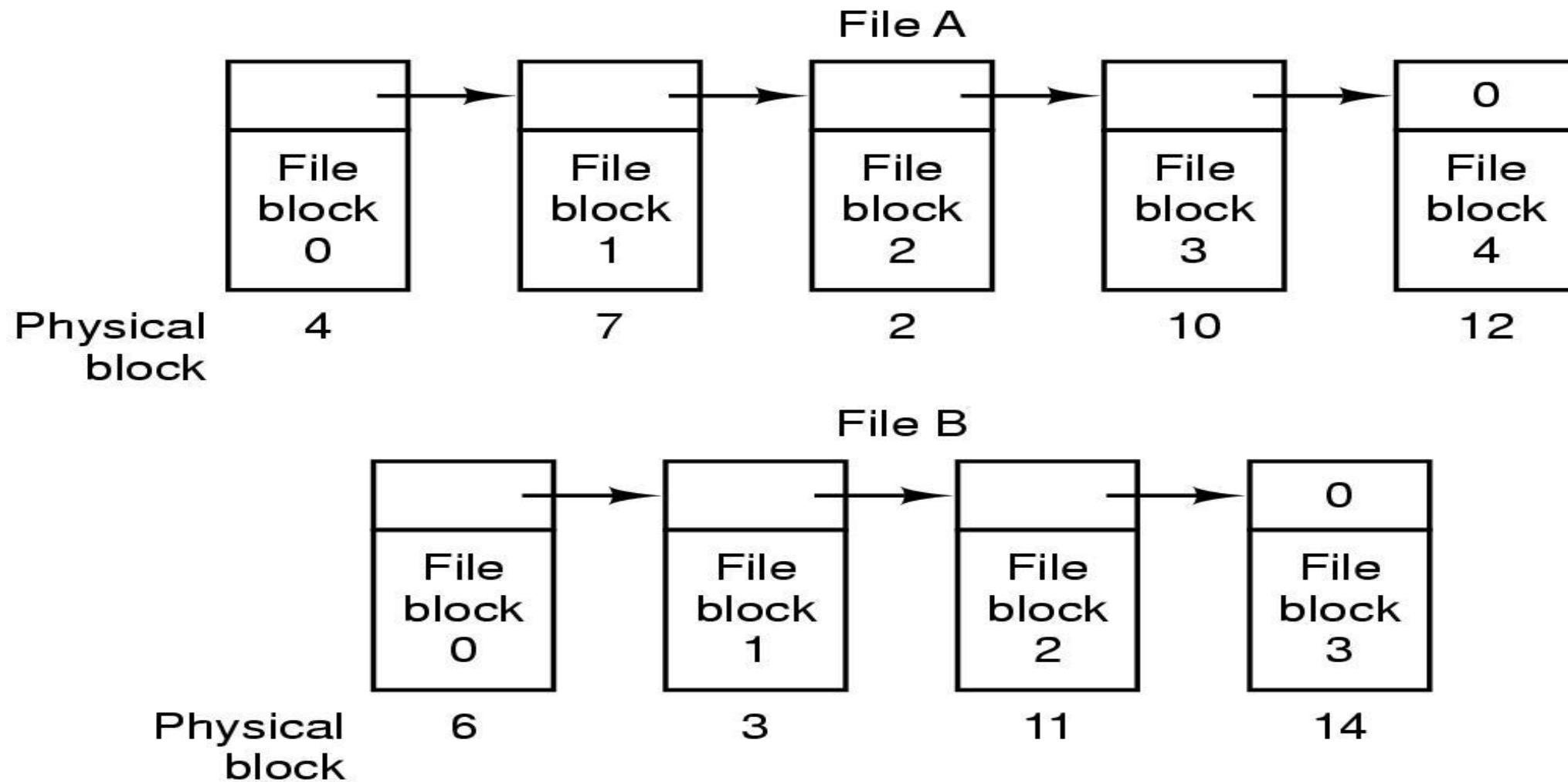
- External Fragmentation
- Compaction is the solution
  - It has a very large overhead
- Files cannot grow.
  - Reallocate the file into a bigger space and release the existing space
  - File size in advance
    - Internal fragmentations

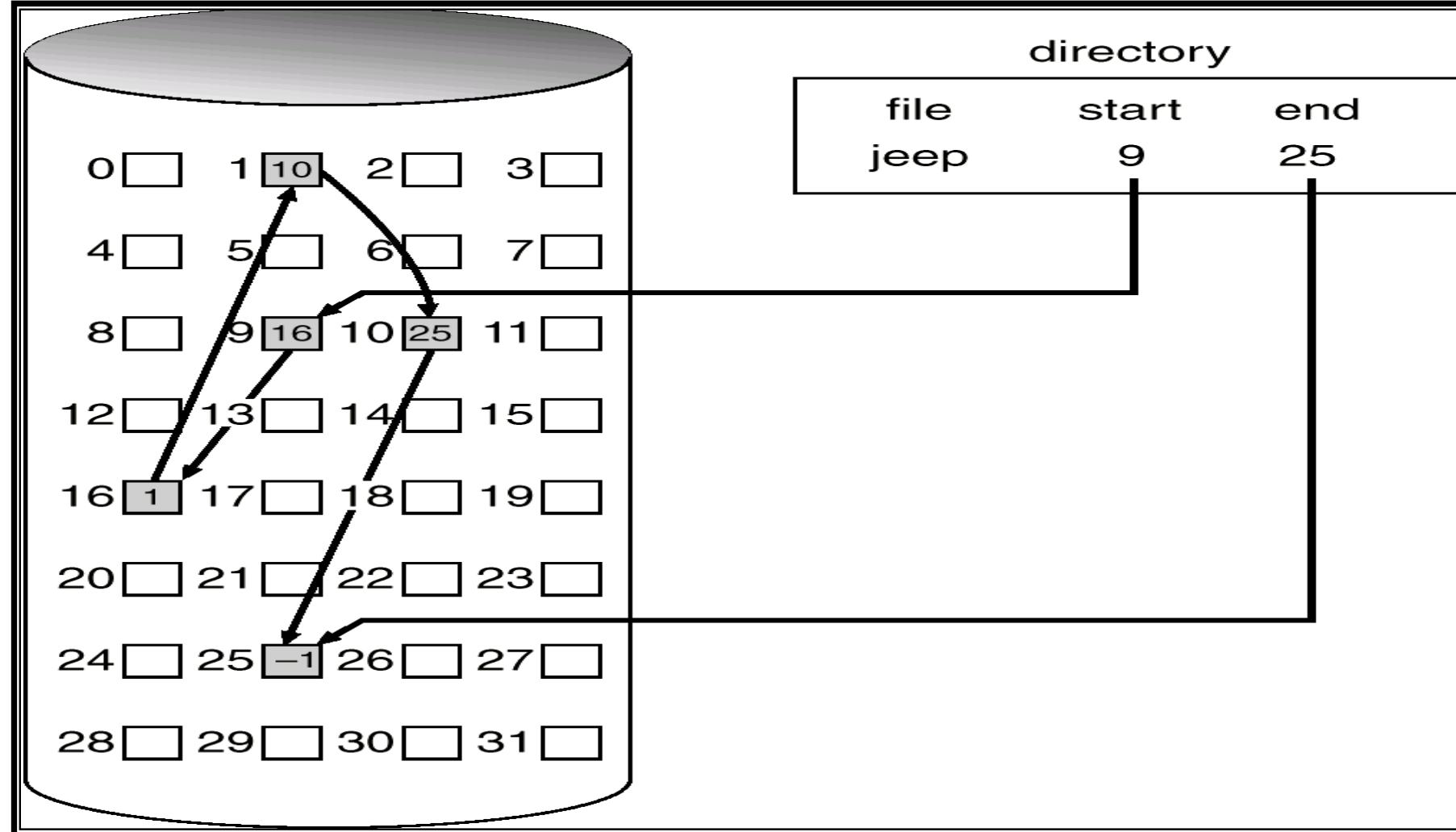
## ❑ Linked allocation

- ❑ Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



- ❑ Simple: need only starting address.
- ❑ Free-space management: no waste of space.
- ❑ No random access.





- Linked list allocation using a table in memory:
  - I File-Allocation Table (FAT) -  
disk-space allocation used by MS-DOS and OS/2.
  - I Random access of files is possible
    - FAT can be scanned for a direct block address

**Physical  
block**

0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

File A starts here →

File B starts here →

Unused block →

## directory entry

test	• • •	217
------	-------	-----

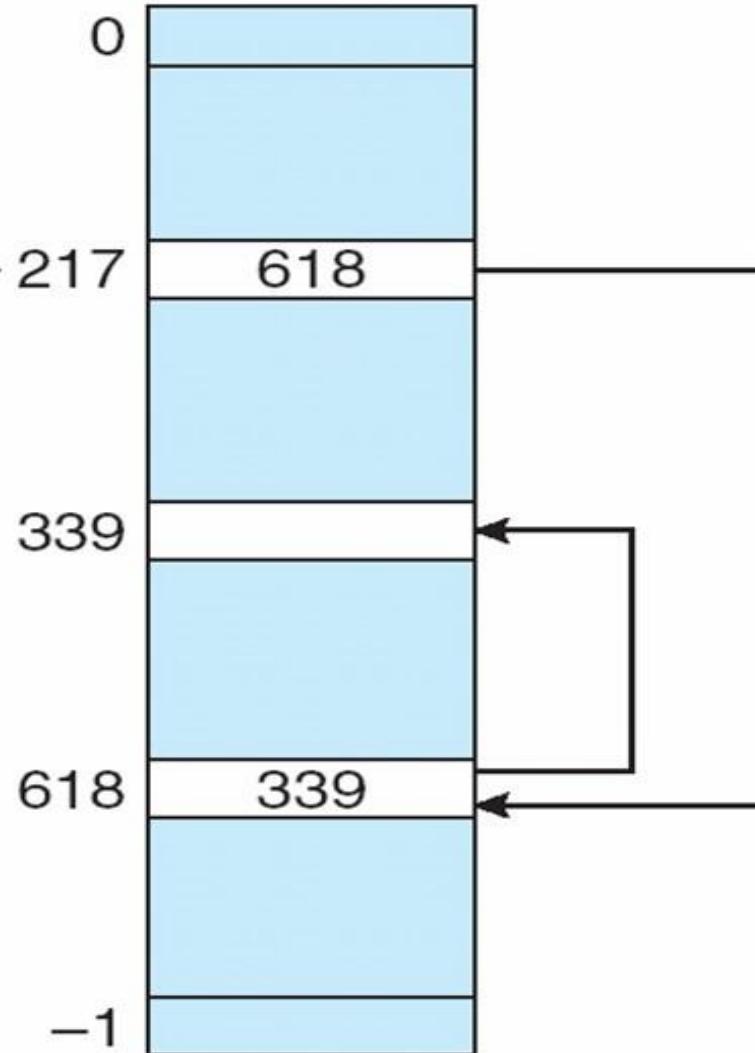
name

start block

no. of disk blocks

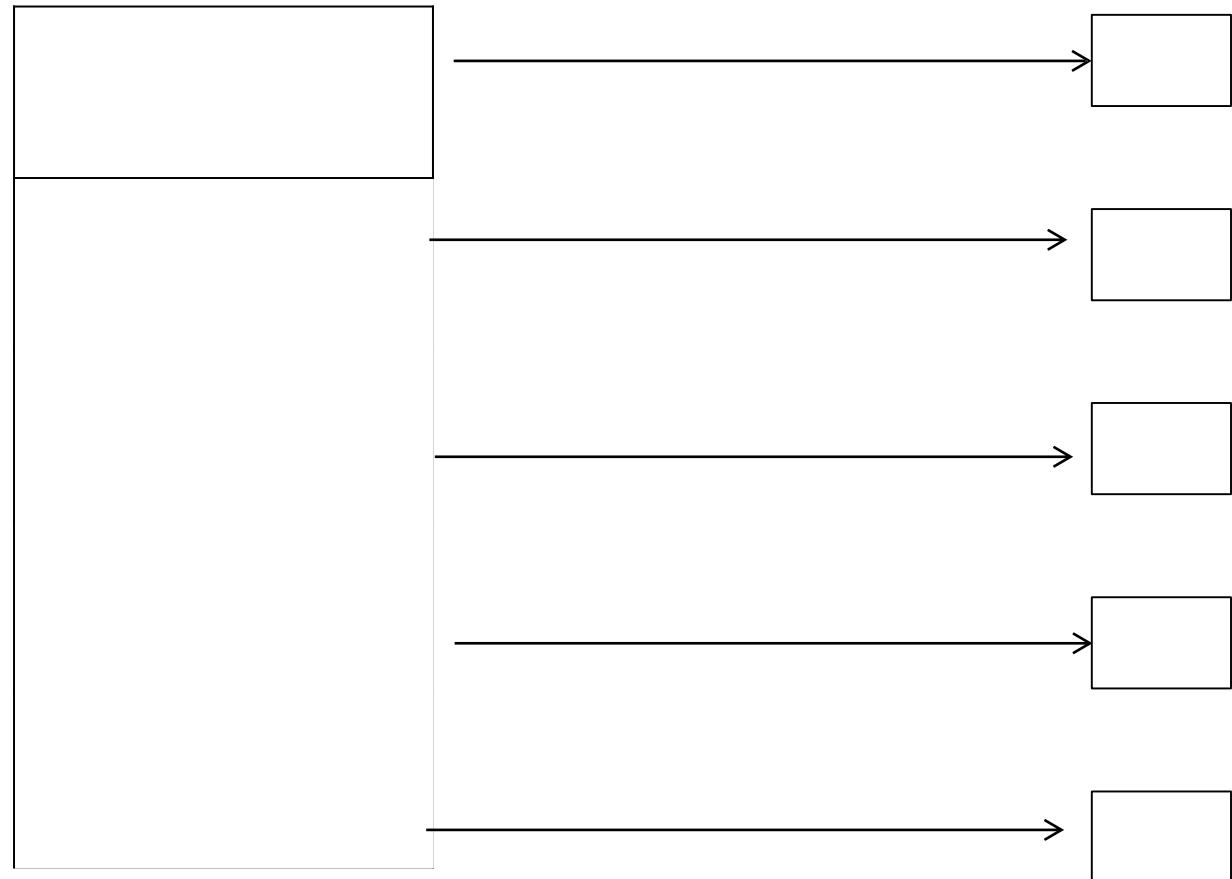
-1

FAT

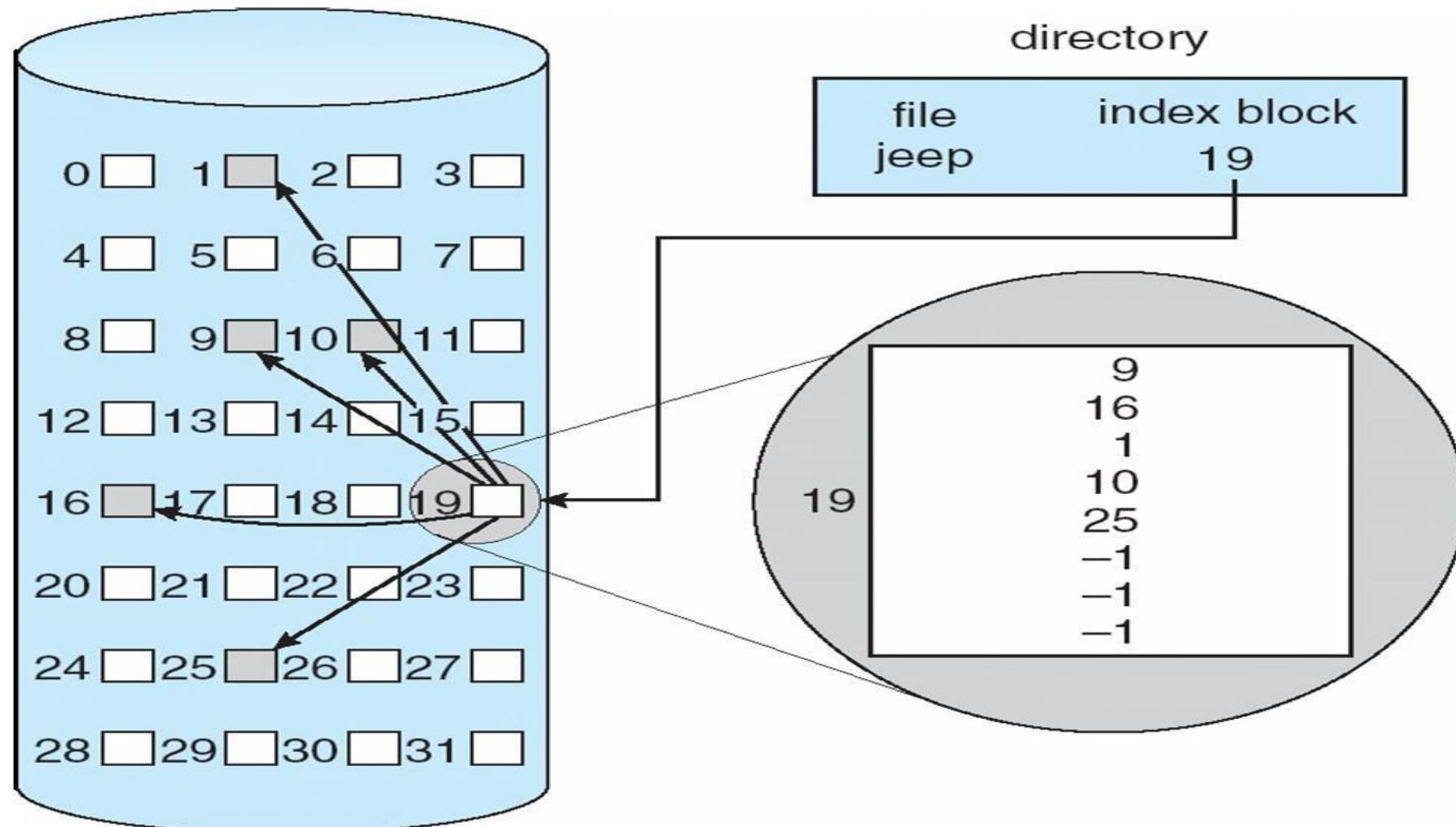


## Indexed allocation

- Problems of external fragmentation and size deceleration present in contiguous allocation are overcome in linked allocation
- But the absence of FAT, linked allocation does not support random access
- Indexed allocation will solve the problem
- All the pointers together into a index block
- Each file is having a index block
- Address of the index block finds an entry in the directory



index table



# Free Space Management

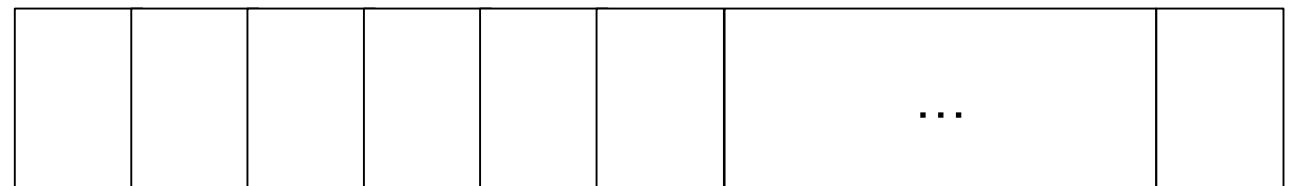


- Maintained by the OS
- Free space list
- It can be implemented in many ways
  - Bit vector
  - Linked list
  - Grouping
  - Counting

## □ Bit Vector

- Very common
- N is total no of blocks

0      1      2      ...      n-1



bit[ $i$ ] =   
 1  $\Rightarrow$  block[ $i$ ] free  
 2  $\Rightarrow$  block[ $i$ ] occupied

## □ **Linked List**

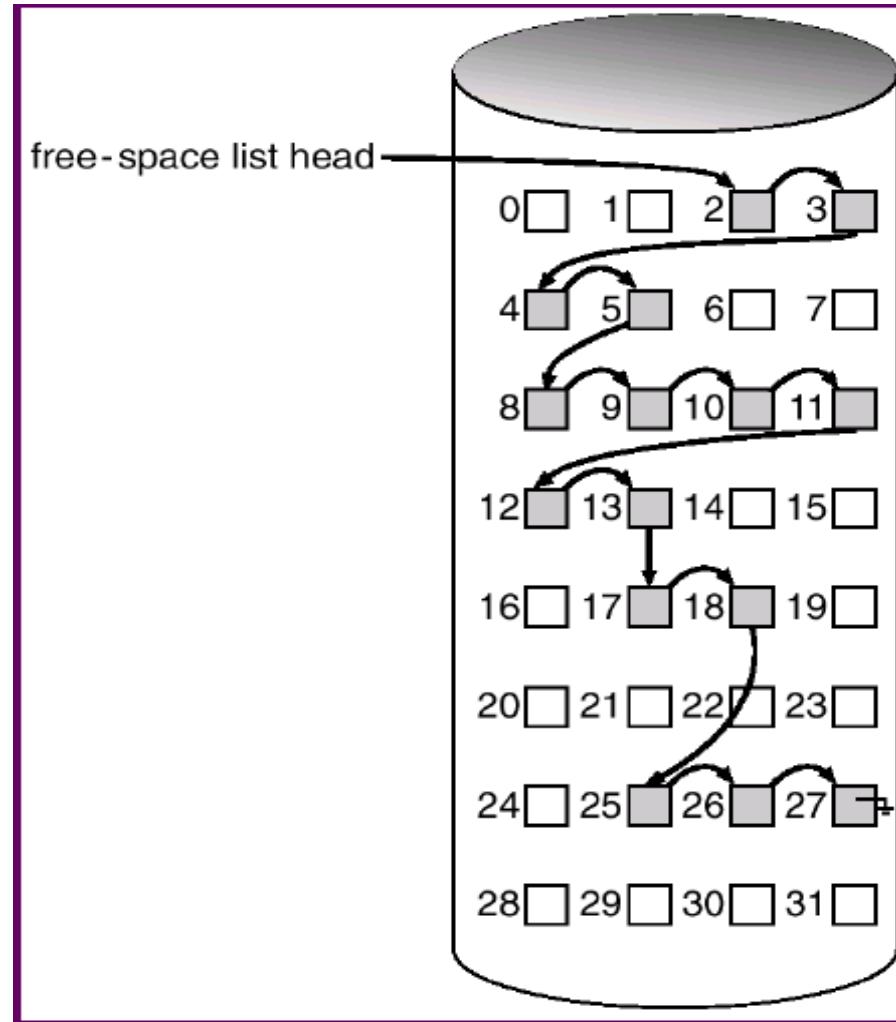
- All free blocks are linked together
- List head contains the address of the first free block
- Works well for linked allocation
- Cannot get contiguous space easily
- No waste of space

## □ **Grouping**

- store the address of n free blocks in the first free block.
- The first n-1 are actually free.
- The final block contains the addresses of another n free blocks...

## □ Counting

- | Used in contiguous memory allocation
- | When memory is free
- | Starting address of the block + count is stored
  - Count is how many free blocks from that block



➤ **Bit Vector**

**1100001100000011100111110001111**

➤ **Grouping**

**Block 2 → 3, 4, 5**

**Block 5 → 8, 9, 10**

**Block 10 → 11, 12, 13**

**Block 13 → 17, 28, 25**

**Block 25 → 26, 27**

➤ **Counting**

**2 4**

**8 6**

**17 2**

**25 3**

## □ **Linear List**

- ▀ Linear List of file names with pointers to the data blocks is one way to implement a directory
- ▀ Searching is required, it will consume a lot of time

## □ **Harsh Table**

- ▀ Linear list is used for directory entries
- ▀ A harsh table takes a value computed from the file name and returns a pointer to the file name in the linear list

# Thank you