INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

OPERATING SYSTEMS

Course Code: A50510

III B. Tech I semester (JNTUH-R15)

Prepared by:
Mrs. B.DHANALAXMI
Associate Professor
IARE10034

Operating System

TEXT BOOKS:

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, "Operating System Principles", 8e, Wiley Student Edition.

2. W. Stallings, "Operating Systems - Internals and Design Principles", 6e, Pearson.

REFERENCES:

1. P. C. P. Bhatt, "An Introduction to Operating Systems", PHI.

UNIT-I
Syllabus

Operating System Introduction: Operating Systems objectives and functions, Computer System Architecture, OS Structure, OS Operations, Evolution of Operating Systems - Simple Batch, Multi programmed, time-shared, Personal Computer, Parallel, Distributed Systems, Real-Time systems, Special-Purpose Systems, Operating System services, User OS interface, System Calls, Types of System Calls, System Programs, Operating System Design and Implementation, OS Structure, Virtual Machines.

What is an Operating System?

- ✓ A program that acts as an intermediary between a user of a computer and the computer hardware.

- ✓ Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.

- ✓ Use the computer hardware in an efficient manner.

Operating System Definitions:

- ✓ A program that acts as an intermediary between a user of a computer and the computer hardware.
- ✓ Operating system is a combination of different software's.
- ✓ Operating system is a key program where it manages the hardware and software.

- ✓ Operating system is also a manager which manages the resources of the system.

- ✓ An Operating system is a program that enables the computer hardware to communicate and operate with the computer hardware.

- ✓ A more common definition is that the operating system is the one program running at all times on the computer(usually called the **kernel**), with all else being system programs and application programs.

✓ Different types of Operating System:

1. Character User Interface(CUI)
2. Graphical User Interface(GUI)

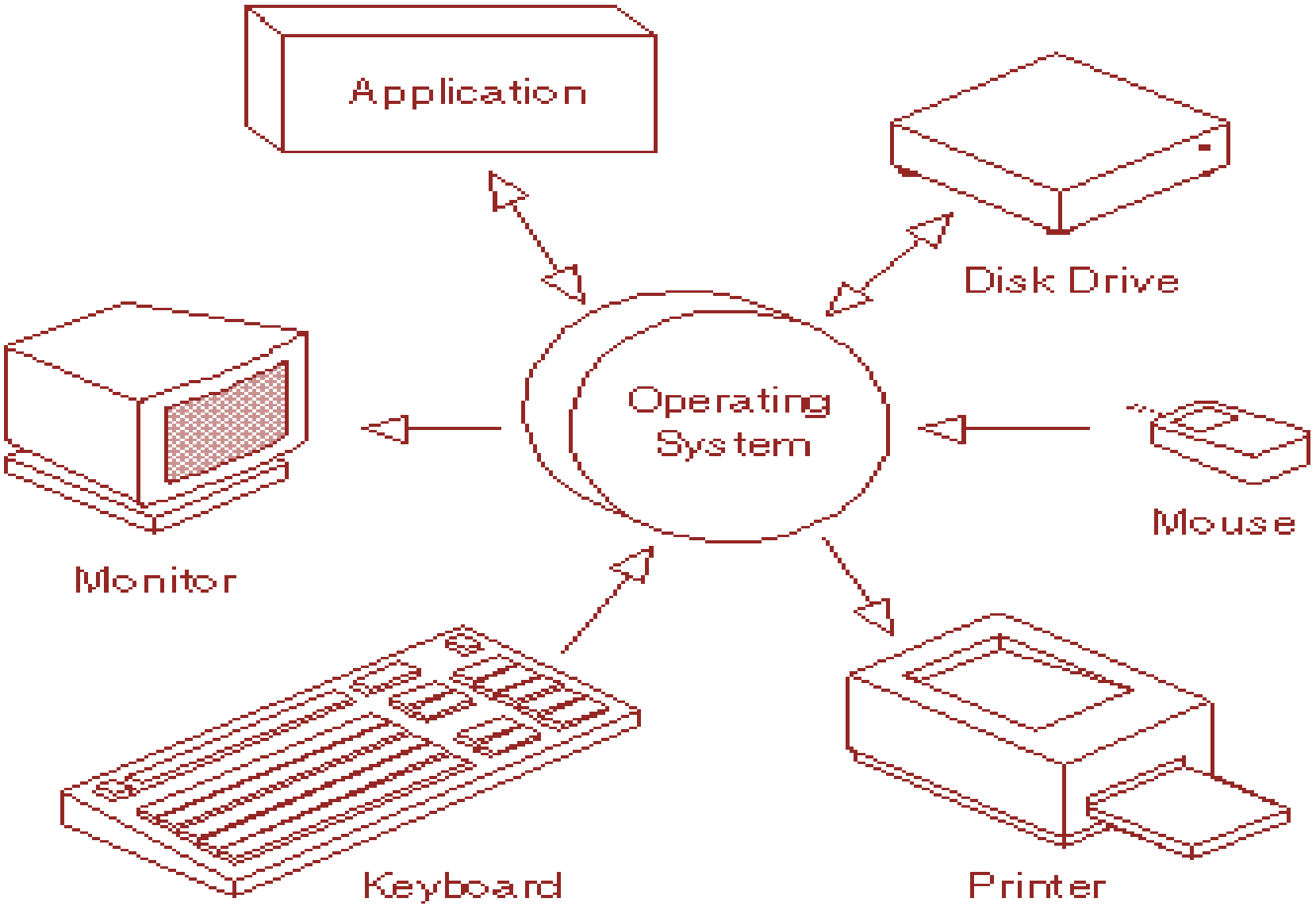
CUI	GUI
I. It is used in connection with computers.	It is also used in connection with computers.
II. In this, the user has to type on keyboard to proceed.	In this, the user use mouse instead of a keyboard.
III. It is not easy to navigate.	It is easy to navigate.
IV. There is only text in case of CUI. Ex: DOS	Graphics & other icons are there in GUI. Ex: Windows

What is an Operating System ?

A modern computer consists of:

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices.

Managing all these varied components requires a layer of software – the **Operating System (OS).**


Application
Disk Drive
Operating System
Monitor
Mouse
Keyboard
Printer

Operating System Goals

OS goals:

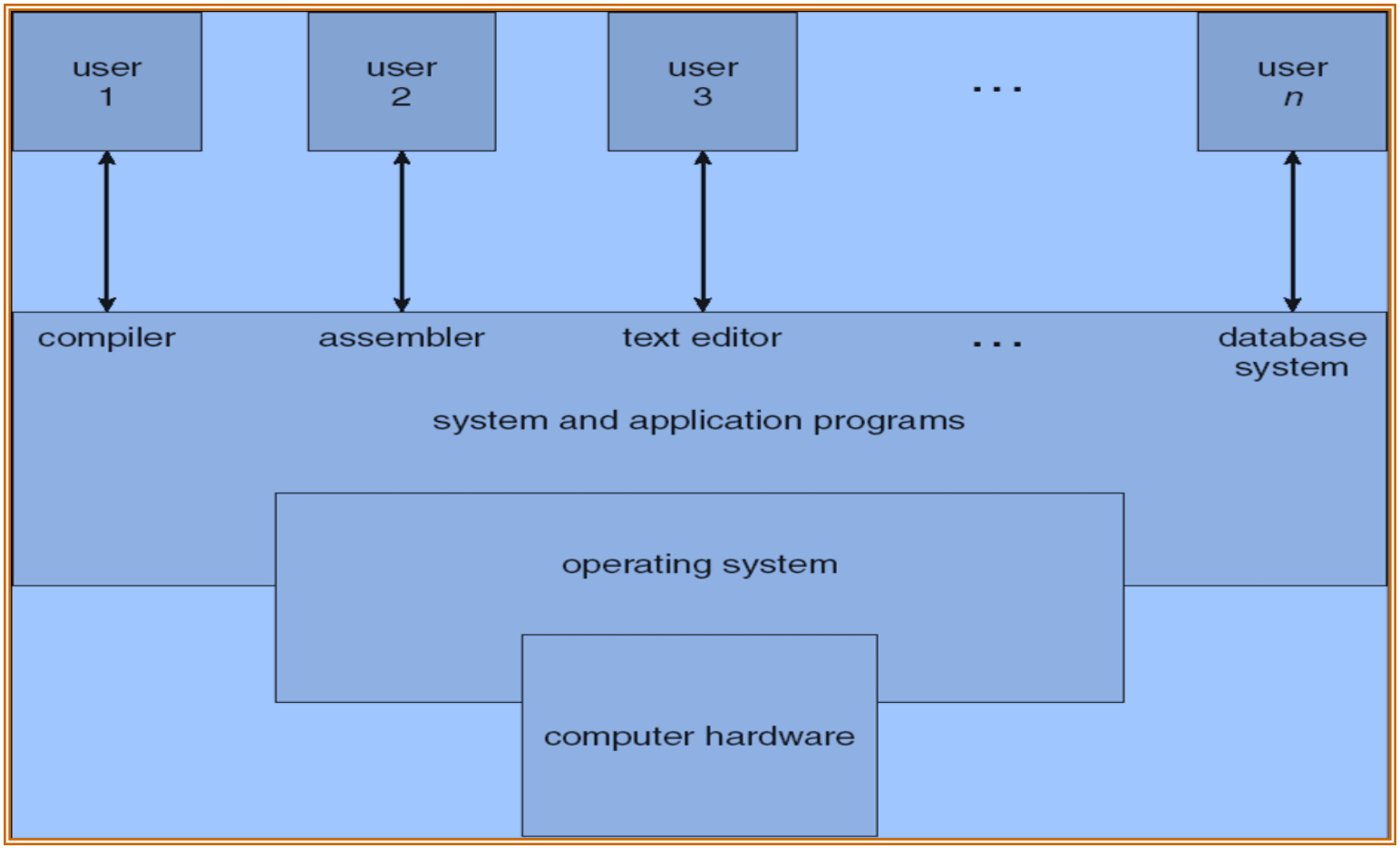
- Control/execute user/application programs.
- Make the computer system convenient to use.
- Ease the solving of user problems.
- Use the computer hardware in an efficient manner.

Computer System Overview

Computer system can be divided into four components

1. Hardware – provides basic computing resources (CPU, memory, I/O devices).
2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
3. Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. Users (people, machines, other computers).

Four Components of a Computer System



There are two types of views:

1. User view: Users want convenience, **ease of use** and don't care about **resource utilization**

2. System view: System view of the OS is mainly focused on resource allocation to meet the requirements of different application programs and end users.

The Boot/Startup Process

- BIOS Takes an inventory of software.
- CMOS provides instructions.
- Software interact directly with the CPU.

The Boot Process:

Step1: BIOS tests Hardware(POST).

Step2:BIOS searches for & loads OS.

Step3:The OS configures the system.

Step4:The user executes application.

STEP:1

- The ROM BIOS assigns resources.
- Begin by reading configuration information stored in jumper, and the CMOS chip & comparing that information to hardware.

STEP:2

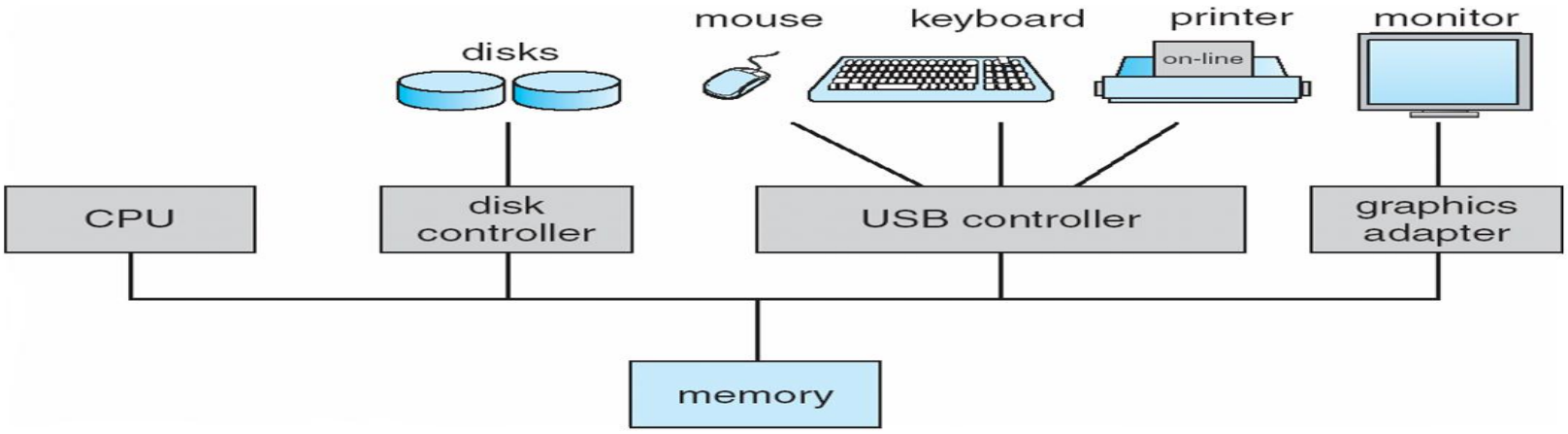
- BIOS finds & loads the OS
- Most often the OS is loaded from logical drive C on to the hard drive.
- Configuration information on the CMOS chip tells startup BIOS when to look for OS.
- BIOS turns on the device , reads the beginning files of the OS , copies them into memory , then turns control over to the OS.

STEP:3

- Os completes the booting process
- The OS checks some of the same things that startup BIOS checked (available memory & whether memory is reliable)
- The OS loads software to control the mouse , CD-ROM , scanner and other peripheral device drivers.

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Storage Structure

- Main memory/RAM/primary memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer

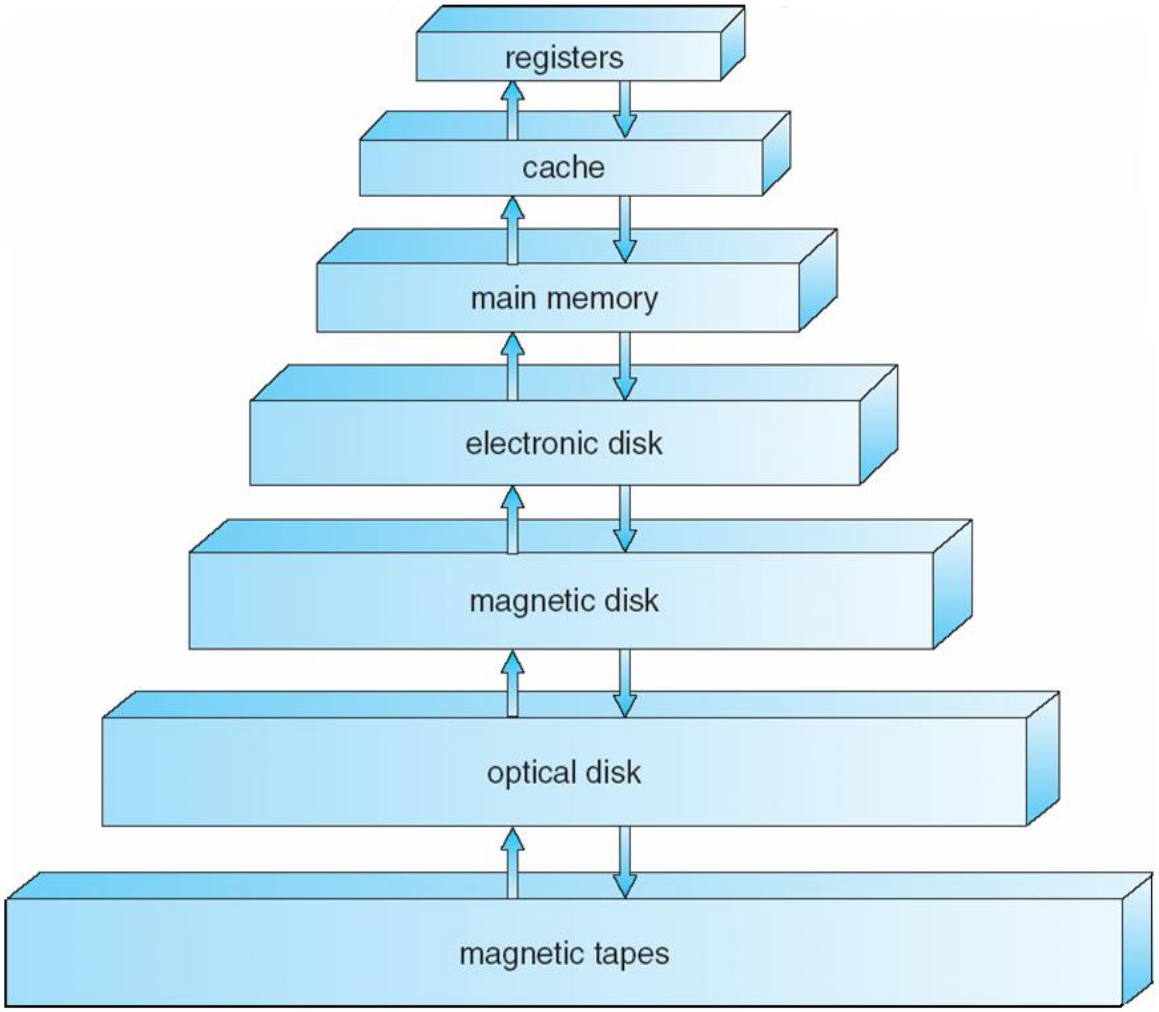
✓:

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed is faster in RAM
  - Cost for RAM is costlier
  - RAM size is smaller than harddisk
  - Volatility-RAM is volatile

- **Caching** – copying information into faster storage system; main memory can be viewed as a *cache* for secondary storage

Storage-Device Hierarchy



Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy
  - Cache is costlier but smaller in size.

I/O Structure

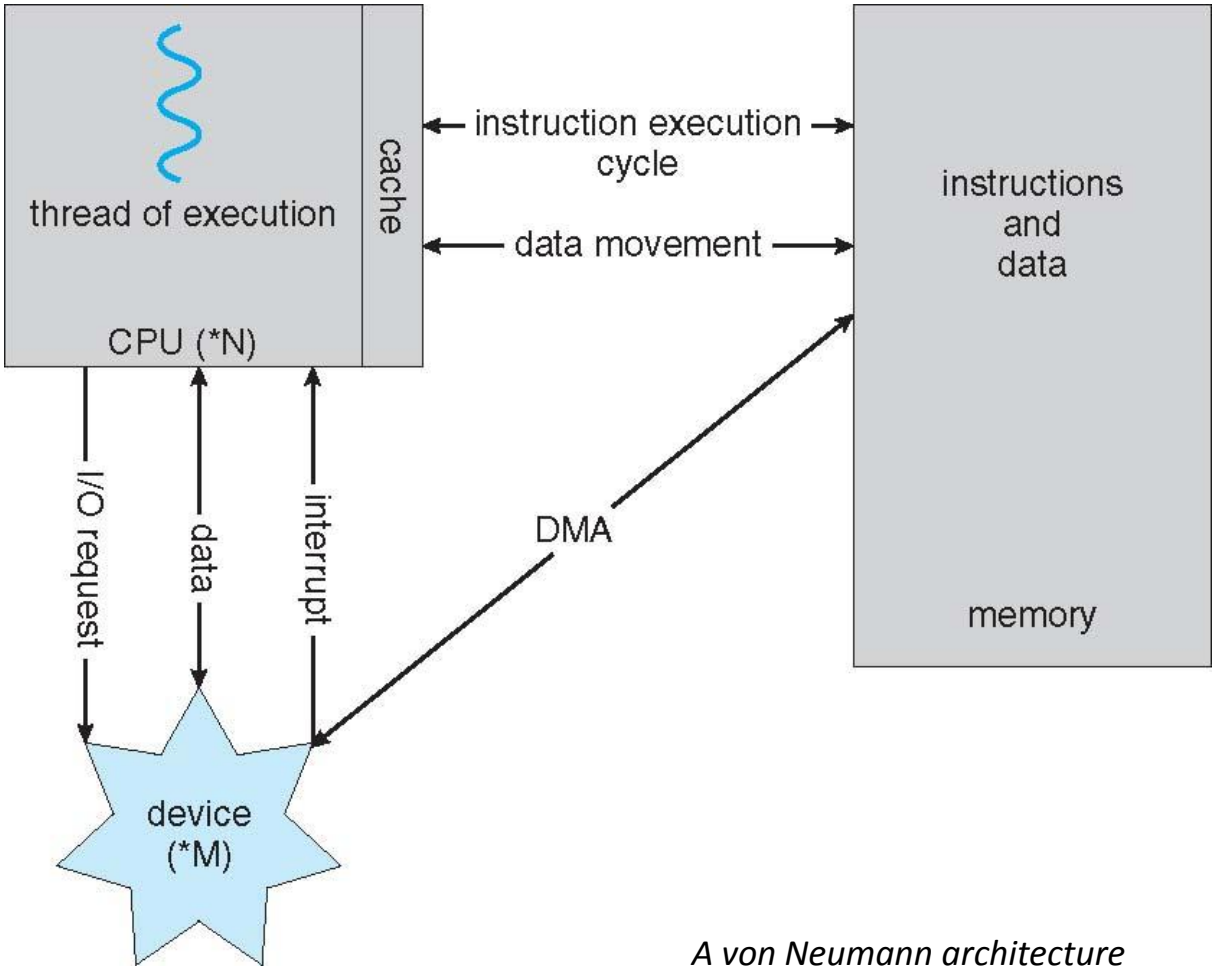
- ✓ **Device driver**: Operating system have a device driver for each device controller
- ✓ To start an I/O Operation, the device driver loads appropriate registers within the device controller. The device controller starts transfer of data from the device to its local buffer.
- ✓ Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
- ✓ This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem DMA is used.
- ✓ DMA is going to know the address of the particular process present in the hard disk.

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds

- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention

How a Modern Computer Works



A von Neumann architecture

Computer-System Architecture

A computer system may be organized in two different ways:

1.Single-Processor

2.MultiProcessor

Single-Processor Systems: Where we have a single CPU

in order to execute the program or application

Multiprocessor systems(parallel systems, tightly coupled systems):

When two or more processes or systems running simultaneously which are independent of each other sharing a particular memory and it can execute simultaneously i.e. called as multiprocessing.

Advantages of multiprocessor systems

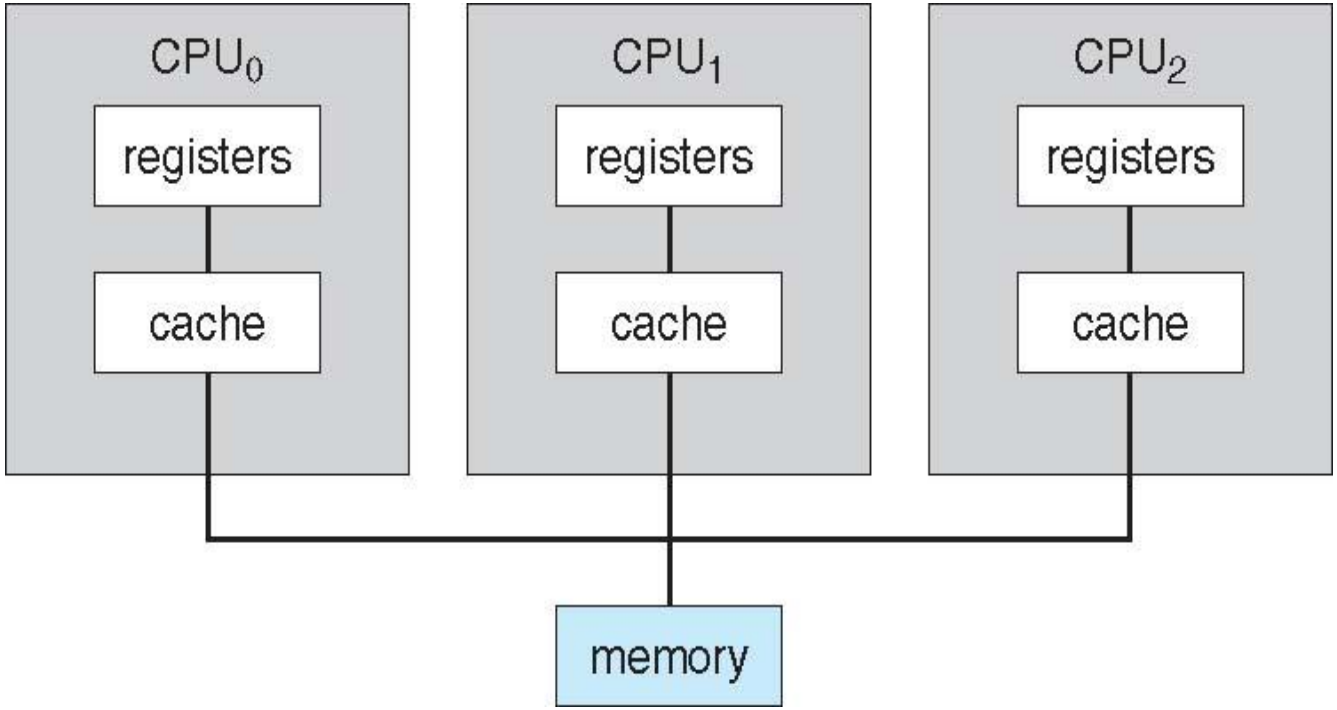
The three main advantages of a multiprocessor systems are:

1. Increased throughput
2. Economy of scale
3. Increased reliability

The two types of multiple processor systems are:

1. Asymmetric multiprocessing(Master-Slave relationship): Each processor is assigned with some task.
2. Symmetric multiprocessing(All processor are peers): Each processor performs all the tasks within the os, and all the processors may behave like cpu.

Symmetric Multiprocessing Architecture



Clustered Systems

- ✓**Clustered systems**: Clustered systems composed of two or more individual systems or nodes joined together.
- ✓Set of systems sharing a common memory/storage area.
- ✓The general accepted definition is that clustered computers share together and are closely linked via a Local-area network(LAN)
- ✓The load balancing is happening in clustered systems.

Operating System Structure

Types of Operating systems:

1. Batch Operating system
2. Multiprogramming Operating system
3. Multitasking Operating system
4. Multiprocessing Operating system
5. Real-time Operating system

Batch Operating system: Disadvantages of Batch OS:

1. Starvation problem is there
2. CPU will be idle
3. It is not really used for interactive applications

Multiprogramming Operating system: It is an extension of batch os.

Advantages:

1. CPU will be busy all the time
2. Efficiency has improved

Multitasking Operating system: It is an extension of multiprogramming.

Advantages:

1. Interactivity has been improved
2. There will be a preemption
3. CPU will be busy all the time

Multiprocessing Operating system:Advantages:

1. Throughput(the number of jobs which are going to finish per unit time) can be improved
2. It will be faster as all the devices are working parallely
3. Reliability also can be improved

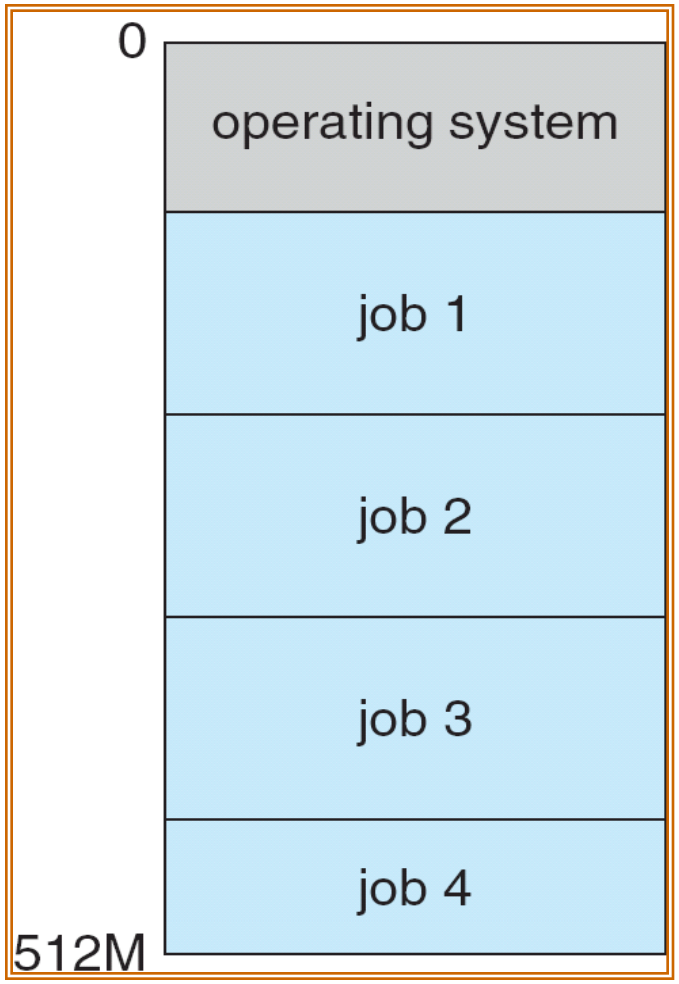
Real-time Operating system:

Giving some jobs and that jobs/processes will have some dead-line.

Multiprogramming:

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

Memory Layout for Multi programmed System



Timesharing (multitasking)

- ✓ Timesharing is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
- ✓ **Response time** should be < 1 second
- ✓ Each user has at least one program executing in memory ⇨ **process**
- ✓ If several jobs ready to run at the same time ⇨ **CPU scheduling**
- ✓ If processes don't fit in memory, **swapping** moves them in and out to run
- ✓ **Virtual memory** allows execution of processes not completely in memory

Operating-system Operations

- ✓ Interrupt Transfers Control to the Interrupt Service Routine generally , through the **interrupt vector,** which contains the addresses of all the service routines.
- ✓ Interrupt architecture must save the address of the interrupted instruction.
- ✓ Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt
- ✓ A trap is a Software generated interrupt caused by an error or a user request.
- ✓ An operating system is interrupt driven.

Dual-Mode Operation

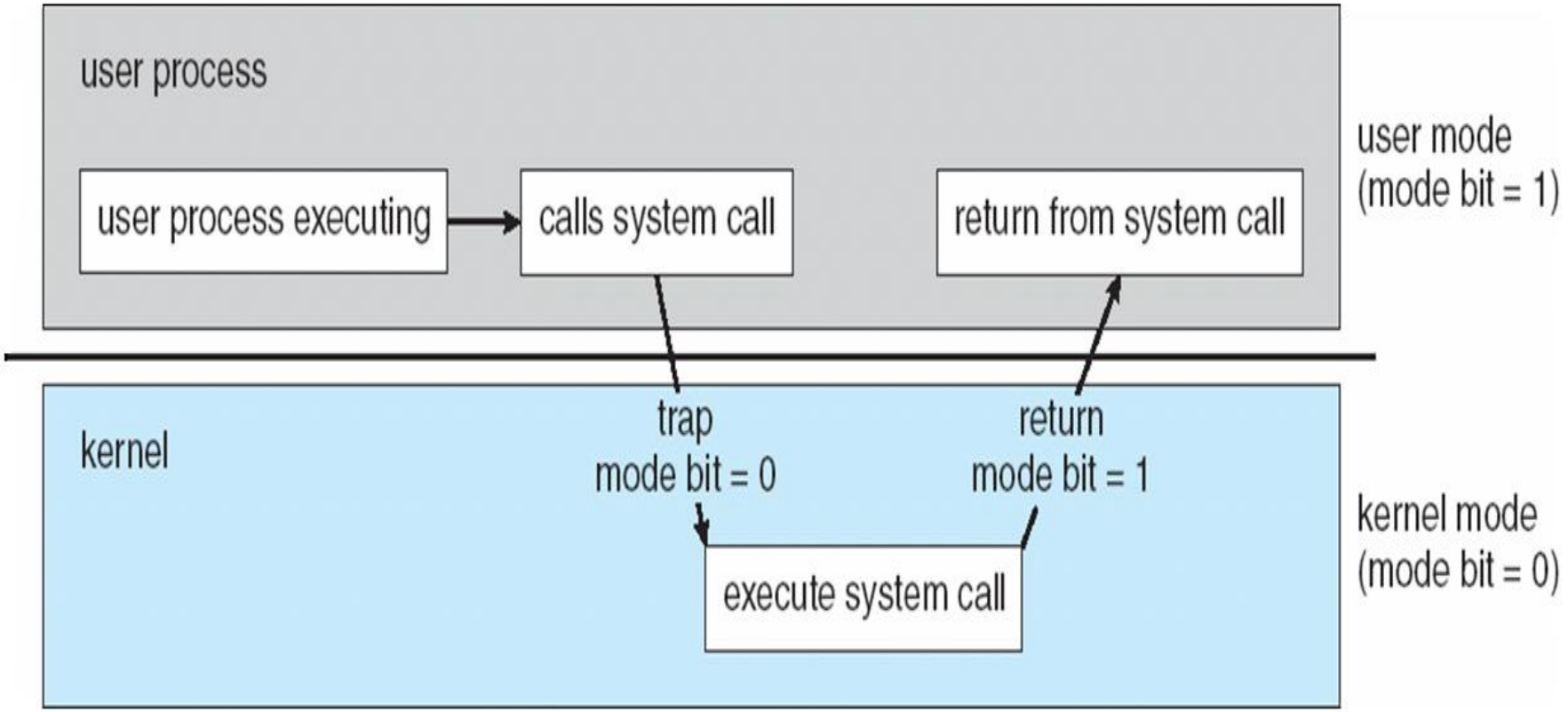
In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of the OS code and user defined code.

There are two separate modes of operation:

1. User mode
2. Kernel mode(Supervisor mode/System mode/Privileged mode)

- ✓ A bit, called the **mode bit,** is added to the hardware of the computer to indicate the current mode:
- ✓ kernel(0) or user(1).
- ✓ With , we the mode bit, we are able to distinguish between a task i.e., executed on behalf of the os and one i.e., executed on behalf of the user.
- ✓ When the computer system is executing on behalf of a user application, the system is in usermode
- ✓ And when a user application requests a service from the os, it must transition from user to kernel mode to fulfill the request.

Transition from user mode to kernel mode:



Time:

There are two situations which are being arrived while the execution of a program

i)The primary duty of operating system is to prevent user programs from getting stuck into an infinite loop.

ii)If that particular program the signal is not reaching the kernel

- ✓ The main task of a timer is to generate an interrupt to the CPU.
- ✓ There are two types of timers , Fixed Timer & Variable Timer.
- ✓ Every second, the timer interrupts and the counter is decremented by 1.As long as counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates the programs for exceeding the assigned time limit.

Process Management

- ✓ A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.

- ✓ Process needs resources to accomplish its task
  - ✓ CPU, memory, I/O, files
  - ✓ Initialization data
- ✓ Single-threaded process has one **program counter** specifying location of next instruction to execute
  - ✓ Process executes instructions sequentially, one at a time, until completion
- ✓ Typically system has many processes, some user, some operating system running concurrently on one or more CPUs

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization(Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data)
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- ✓ All data in memory before and after processing.
- ✓ All instructions in memory in order to execute
- ✓ Memory management determines what is in memory when
  Optimizing CPU utilization and computer response to users
- ✓ Memory management activities
  - ✓ Keeping track of which parts of memory are currently being used and by whom
  - ✓ Deciding which processes (or parts thereof) and data to move into and out of memory
  - ✓ Allocating and deallocating memory space as needed

Storage Management

Operating system is not only managing the main memory/primary memory/RAM, but it also manages the memory on secondary storage devices.

File-System management:

- File management is one of the most visible component of an OS.
- Computer can store the information in the form of files. Files represents data and programs.
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

Mass-Storage Management:

- ✓ Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when the power is lost, the computer system must provide secondary storage to back up main memory.
- ✓ The proper management of disk storage is of central importance to a computer system.

  The Operating system is responsible for the following activities in connection with disk management:
  - Free-space management
  - Disk scheduling
- ✓ Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may hinge on the speeds of the disk subsystem.

Caching:

- ✓ **Caching** is an important principle of computer systems, Information normally kept in some storage systemm such as main memory.
- ✓ The programmer(or compiler) implements the register-allocation and register-placement algorithms to decide which information to keep in registers and which to keep in main memory.
- ✓ Most systems have an instruction cache to hold the next instructions expected to be executed. Without the cache, the CPU would have to wait several cycles while an instruction was fetched from main memory.
- ✓ Cache memory is between RAM and registers.
- ✓ **Caching** – copying information into faster storage system; main memory can be viewed as a *cache* for secondary storage.

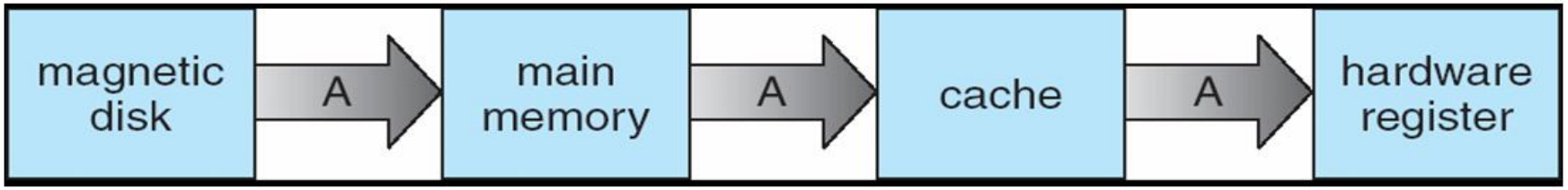
Performance of Various Levels of Storage

- ✓ Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Migration of Integer A from Disk to Register

- ✓ Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy.



Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache.

I/O Subsystem

✓One purpose of OS is to hide peculiarity of hardware devices from the user

✓ I/O subsystem responsible for

✓Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance)

✓General device-driver interface

✓ Drivers for specific hardware devices

Protection and Security

- ✓ **Protection** – It involves guarding a user's data and programs against interference by other unauthorized users of the system.

- ✓ **Security** – It involves guarding of a user's data and programs against interference by external entities.

  Ex: Unauthorized persons, Viruses etc.

Security is provided by :PWD settings, Data Encryption, Biometrics , Firewall Settings (S/W & H/W setup between an internal Computer & the system)

Facts to protection of information:

1. **Secrecy**: Only authorized user should access the information i.e., available in the document.
2. **Privacy:** It is used only for the purpose which is intended and shared. So, it should not be misused.

Special Purpose Systems

- The main objective of special purpose system is to deal with limited computation domains.

There are different types of special purpose systems , some of them are listed below:

1.Real-Time Embedded Systems

2.Multimedia Systems

3.Handheld Systems

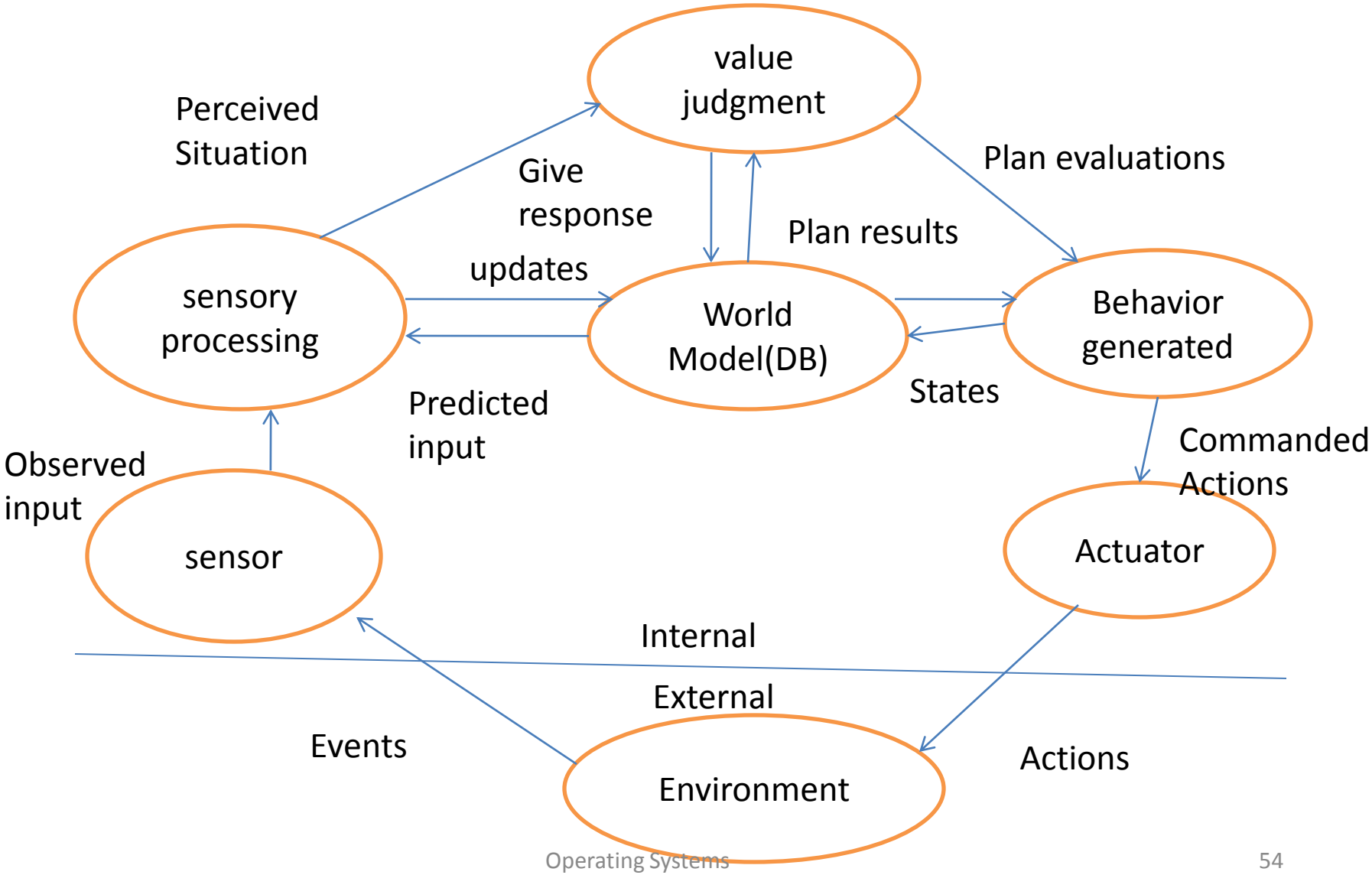
1.Real-Time Embedded Systems:

These devices are found everywhere , from car engines and manufacturing robots , microwave ovens.

Real-Time Embedded Systems

- An embedded system is a special purpose computer system designed t perform one or few dedicated functions , often with real time computing constrain.
- These are embedded as a part of complete device including hardware and mechanical parts.
- These devices are found everywhere , from car engines and manufacturing robots , microwave ovens.
- Basically general purpose computers , running standard operating systems such as UNIX the special purpose applications to implement the functionality.
- Others are hardware devices with application specific integrated circuits (ASCIs) that perform their task without OS.
- Embedded system always runs Real-Time operating systems.

Real-Time Control System(NIST)



Cont..

Process Steps for the real Time control system for taking decisions:

- Sensors will monitor the environment and update the World Model.
- Behavior Generation submits tentative plan to World Model.
- World Model generates expected results and send to Value Judgment (cost /benefit/uncertainty attributes).
- Actuator interacts with environment.

Rules for Real-time system:

- It has a well defined , fixed time constraints.
- Processes must be done within the defined constraints or the system will fail.
- Real-Time system functions correctly only if it returns correct results within its time constrain.

Multimedia Systems

- All operating systems designed only to handle conventional data like Text files ,programs , word-processing , spread sheets.
- In technology there is incorporation of Multimedia Data into computer system.
- Multimedia Data consists of audio , video files as well as conventional data.
- Multi media includes audio files such as DVD'S MP3 ,VIDEO CONFERENCING, SHORT VIDEO CLIPS , NEW STORIES DOWNLOADED FROM INTERNET, LIVE WEBCASTS.
- As technology increases they are being directed towards smaller devices , cellular telephones etc..

Handheld Systems

- Handheld systems includes Personal Digital Assistance(PDAs), such as PALM & POCLET-PCs, CELLULAR TELEPHONES , uses special purpose embedded systems.

LIMITATIONS OF HANDELD SYSTEMS:

1.Physical memory in a handheld depends on the device – 1MB and 1 GB.(memory must be efficiently used)

2.Speed of the processor used in the device-(Higher power the higher processing).

3.I/O-(Lack of physical space limits input methods to small keyboards , small monitors etc.)

Computing Environments

- The word computing is referred as a process of using a computer technology for completion of a task.
- There are 4 different types of Computing Environments they are:

1. Traditional Computing
2. Client-Server Computing
3. Peer-Peer Computing
4. Web-Based Computing

Traditional Computing

- In past they used to have a PCs connected to network with servers providing files and print services.
- No possibility of remote access and portability was achieved only with laptops.
- So , the companies have provided with a “portals” which provides web access to their internal servers.
- Through this service their was little remote access and portability options achieved.

Client-Server Computing

- This is a specialized distributed system , called a client-server system.
- The computer Server system provides an interface to which a client can send a request to perform an action in response the server executes the action and send back to the client.
- The File-server system provides a file-system interface where clients can create , update , read and delete files.

Peer-to-Peer Computing

- Another structure of Distributed system is Peer-to-Peer system model.
- Clients and servers are not distinguished from one another instead all nodes are considered as peers .
- Each may act as a server or client depending whether it is requesting a service or providing a service.
- To Participate in a peer-to-peer system , a node must first join the network of peers.
- To Determine what services are available is accomplish in two general ways:

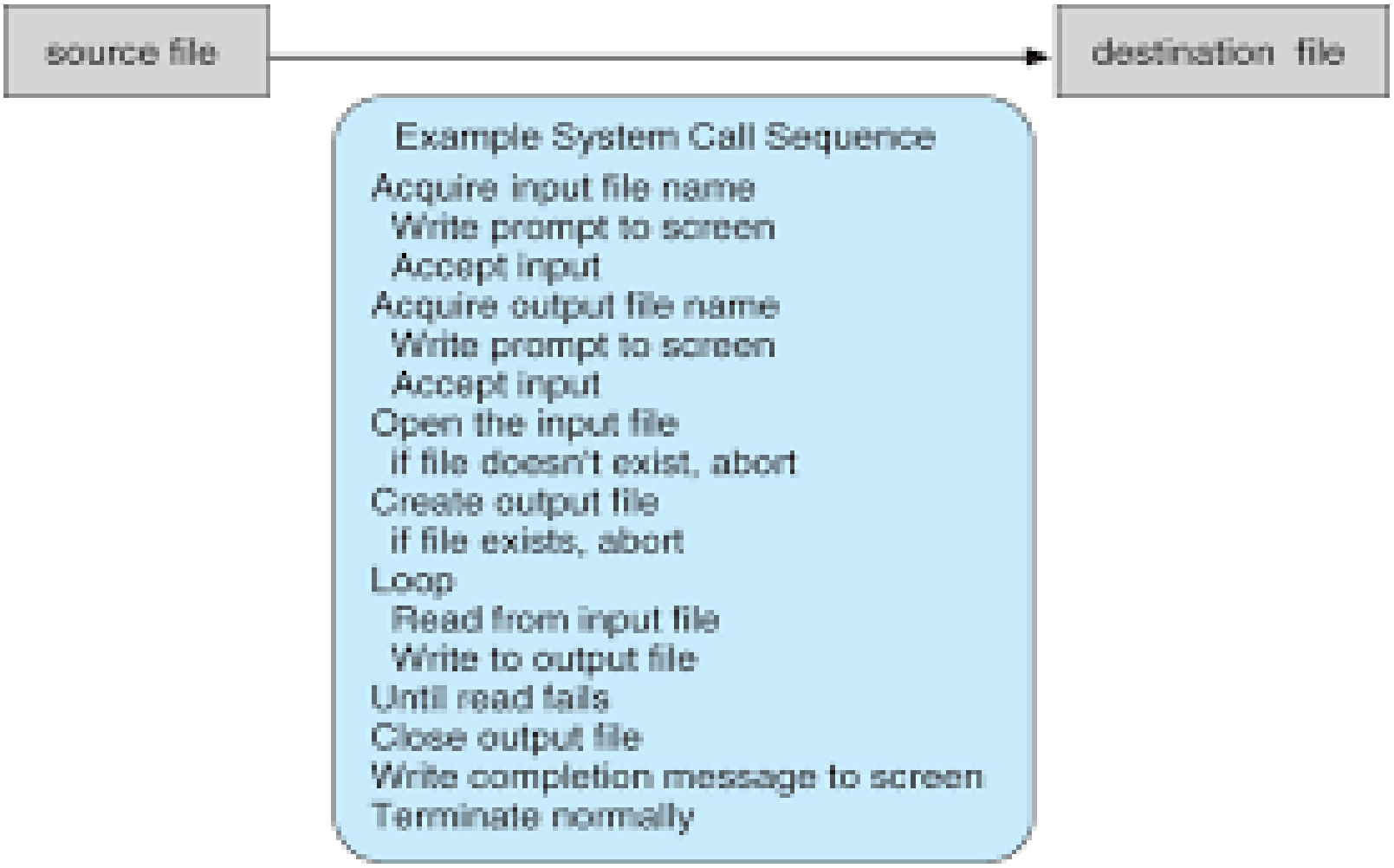
When a node joins a network it registers its service with a centralized lookup service on a network.

A discovery protocol must be provided by other peers in the network.

System Calls

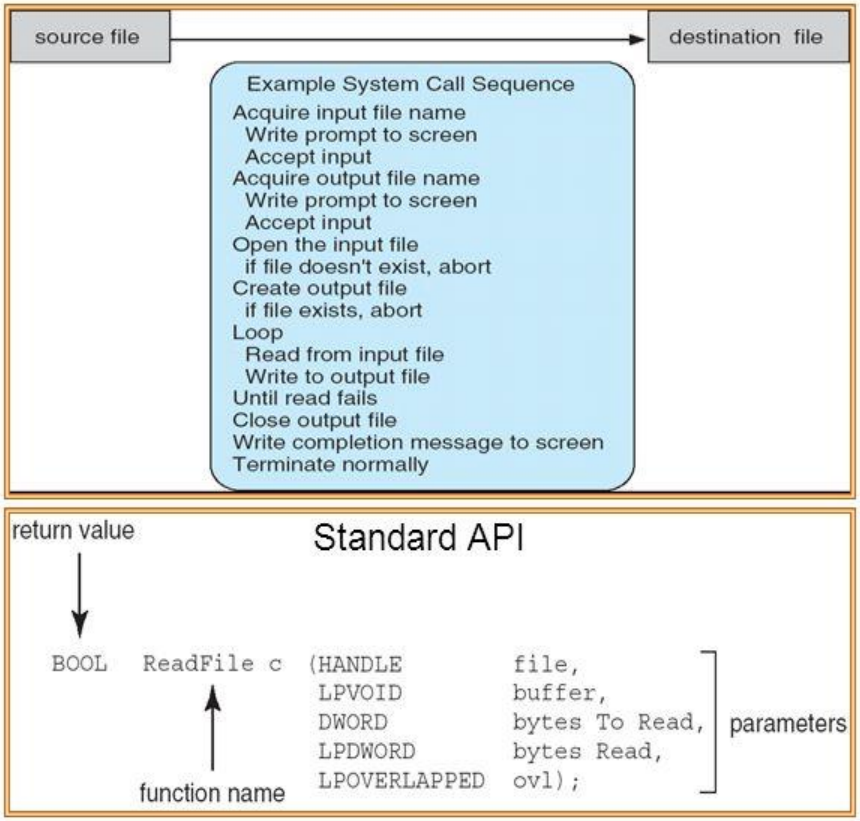
- The user want to interact with all the devices inorder to solve the problem.
- System calls basically manages hardware and instruct to do some action requested by the user.
- A system call is processed in kernel Mode.
  Generally available as assembly-language instructions.
  Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)
- Access to the hardware is via High Level Application Program Interface (API) rather than direct system call use.
- Three most common APIs are win32 for windows , POSIX for linux and Mac OS X, java API for java programs.

Example of System calls Execution



Why APIs?

System call sequence to copy contents of one file to another



Cont..

- Description of parameters passed for ReadFile():

1.HANDLE file:the file to be read.

2.LPVOID buffer : a buffer where the data will be read into and written from.

3.DWORD bytesToRead : The number of bytes to be read into buffer.

4.LPDWORD bytesRead- The number of bytes read during last read.

5.LPOVERLAPPED ovl-indicate if overlapped i/o is used.

System Calls Implementation

- Typing a number associated with each system call.
  - ✓ System call interface maintains a table indexed according to these numbers.
- The system call interface invokes intended system call in OS kernel and returns status to the system call and any return values.
- The caller need nothing to know how a system call is implemented.
  - ✓ Just need to obey API and understand what OS will do as a result file.
- Most details of OS are hidden from Programmer by API.
  - ✓ Managed by run-time support library (set of functions build into library along with compiler)

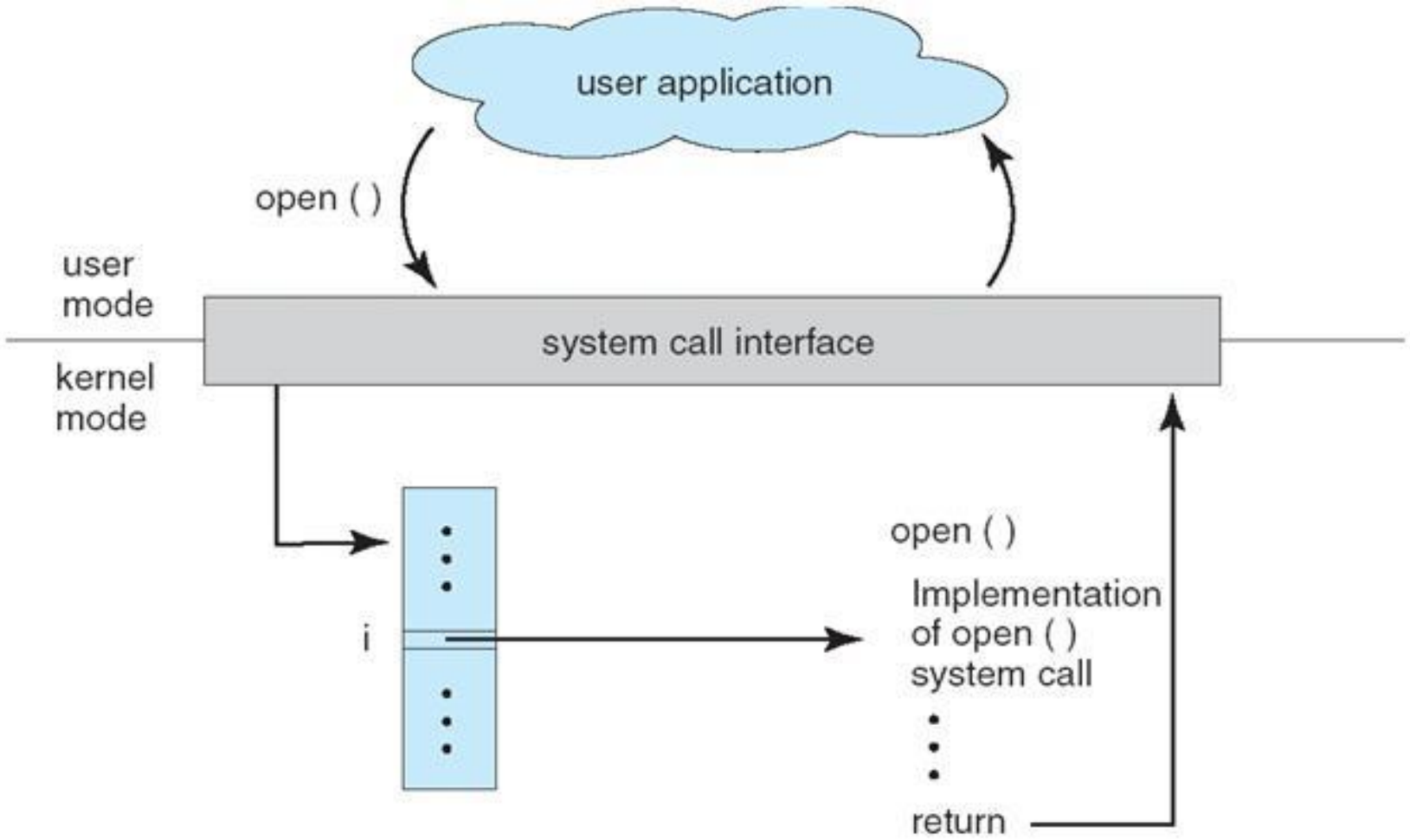


fig: Handing of user application invoking the open()system call

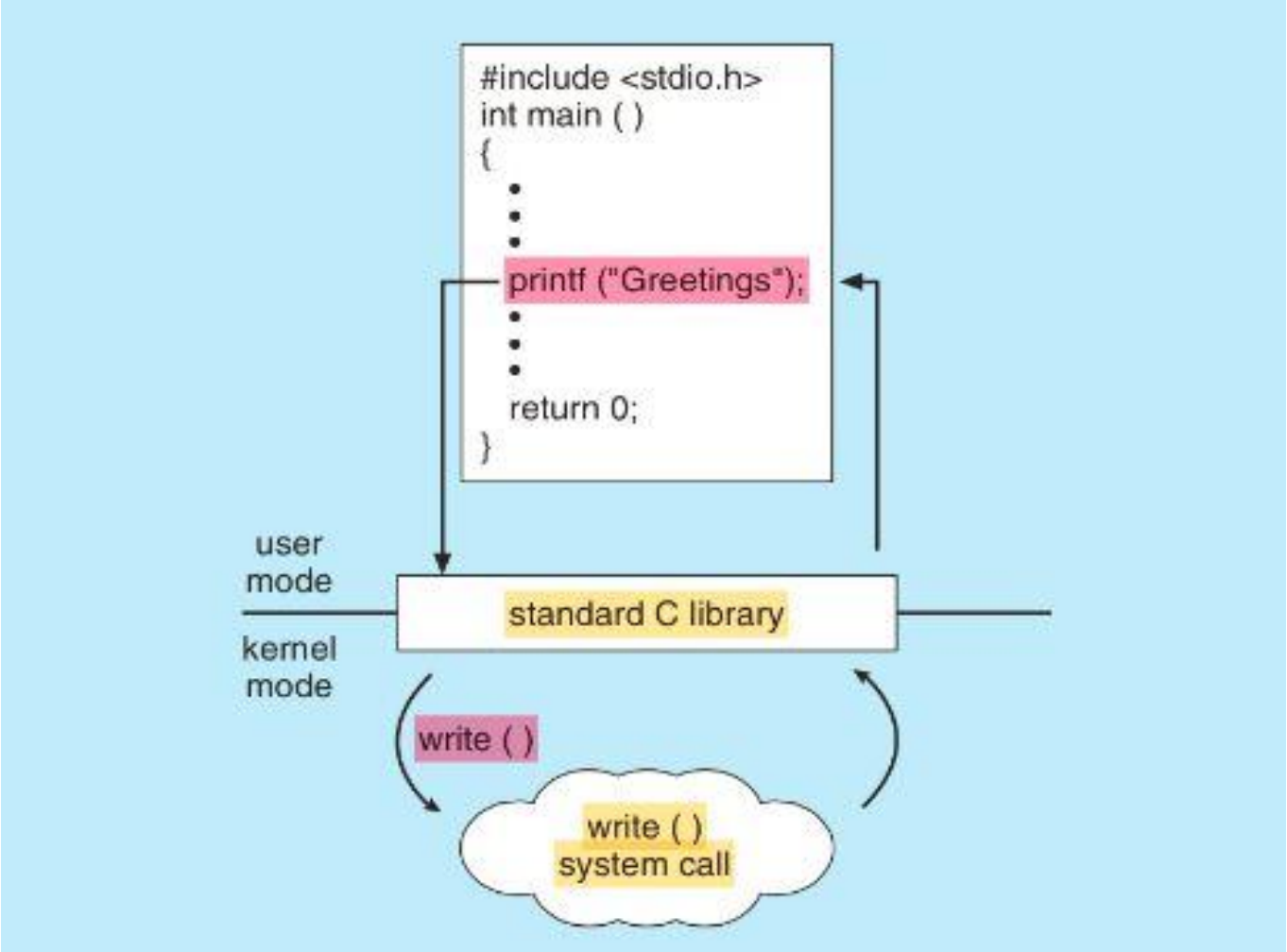


fig: Standard C library handling of write()

Cont..

Often more information is required than simply identity of desired system call.

- ✓Exact type and amount of information vary according to OS and call.

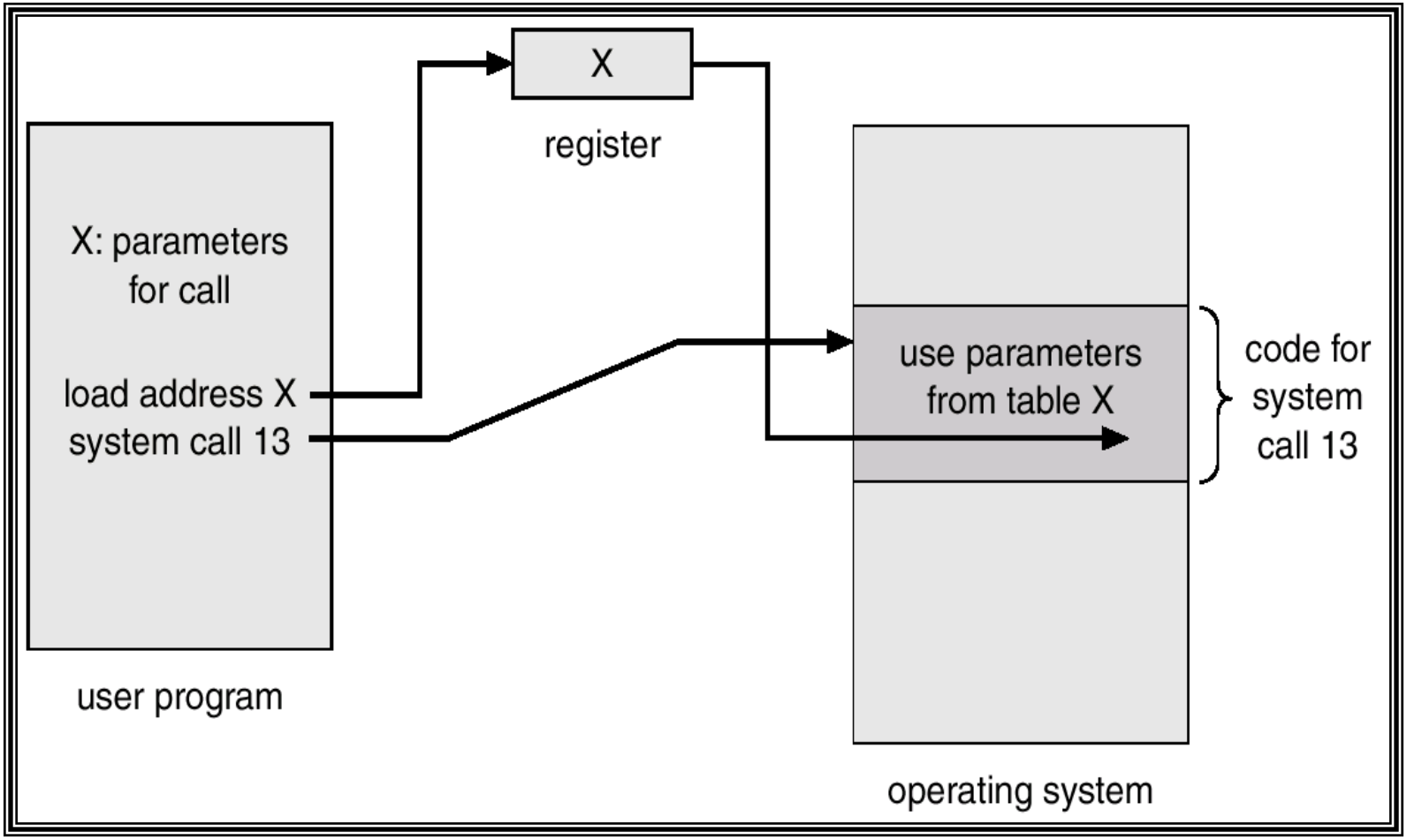
Three general methods are used to pass parameters between a running program and the operating system.

Simplest : pass the parameters in register.

- ✓In some cases may be more parameters than registers.

- Parameters stored in a block , or table , in memory , and address of block passed as a parameter in register.
  - ✓ This approach was taken by LINUX & SOLARIS.
- Parameters placed or pushed on to stack by the program and popped off the stack by the operating system.
- Block or Stack methods donot limit the number or length of parameters being passed

Passing as a Parameters



Types of System calls

System calls can be roughly divided into five major categories:

1. Process control:

✓ end, abort

✓load, execute

✓create process, terminate process

2. File management

✓Create file, delete file

✓ Open, close

Cont..

3.Device management:

- ✓ read, write
- ✓Request device, release device

4. Information ,maintenance

- ✓Get process, file, or device attributes
- ✓Get time or date, set time or date

5. Communications

- ✓Create, delete communication connection
- ✓Send, receive messages

System Generation (SYSGEN)

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
- *Booting* – starting a computer by loading the kernel.
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

The evolution of the Operating systems are as follows:

1. Serial Processing system
2. Simple batch Processing system
3. Multiprogramming Operating system
4. Time sharing system(Multitasking system)
5. Multiprocessor system

UNIT-II
Syllabus

Process and CPU Scheduling - Process Concepts-The Process, Process State, Process Control Block, Threads, Process Scheduling-Scheduling Queues, Schedulers, Context Switch, Preemptive Scheduling, Dispatcher, Scheduling Criteria, Scheduling algorithms, Multiple-Processor Scheduling, Real-Time Scheduling, Thread Scheduling, Case Studies: Linux, Windows.

Process Coordination-Process Synchronization, The Critical Section Problem, Peterson's solution,

Synchronization Hardware, Semaphores, and Classic Problems of Synchronization, Monitors, Case Studies: Linux, Windows.

Program/process

- A process invokes or initiates a program. It is an instance of a program that can be multiple and running the same application.
- Example:- Notepad is one program and can be opened twice.
- A process is a program in execution.
- It is an active entity whereas a program is an passive entity. It is a set of instructions.
- Program becomes process when executable file loaded into memory

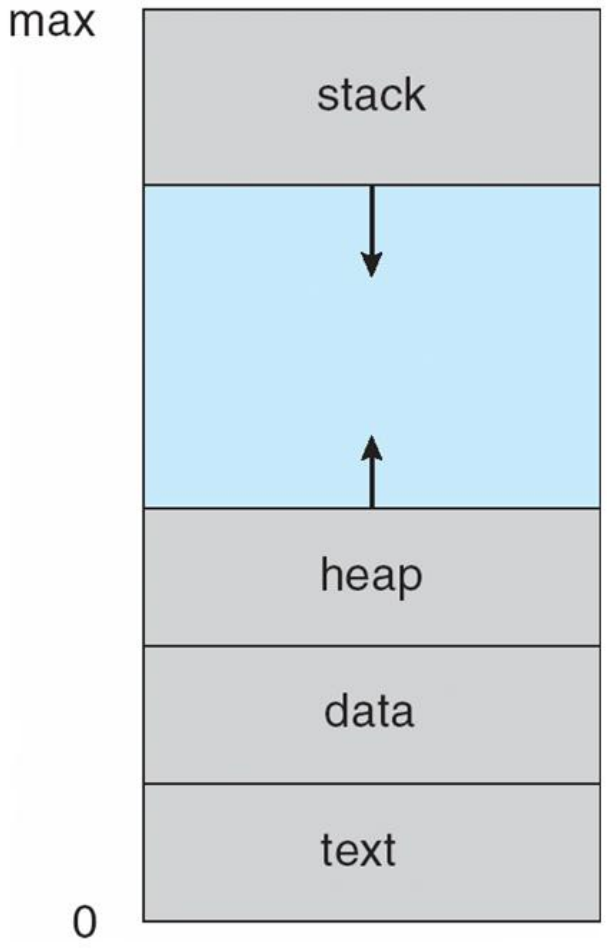
What is a process?

- A process can be thought of as a program in execution,
- A process will need certain resources—such as CPU time, memory, files, and I/O devices to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.

Process

- It contains
  - Text Section-contains text code
  - Data Section-global variables
  - HeapSection -dynamic memory allocation
  - Stack Section
    - Temporary data like function parameters, local data

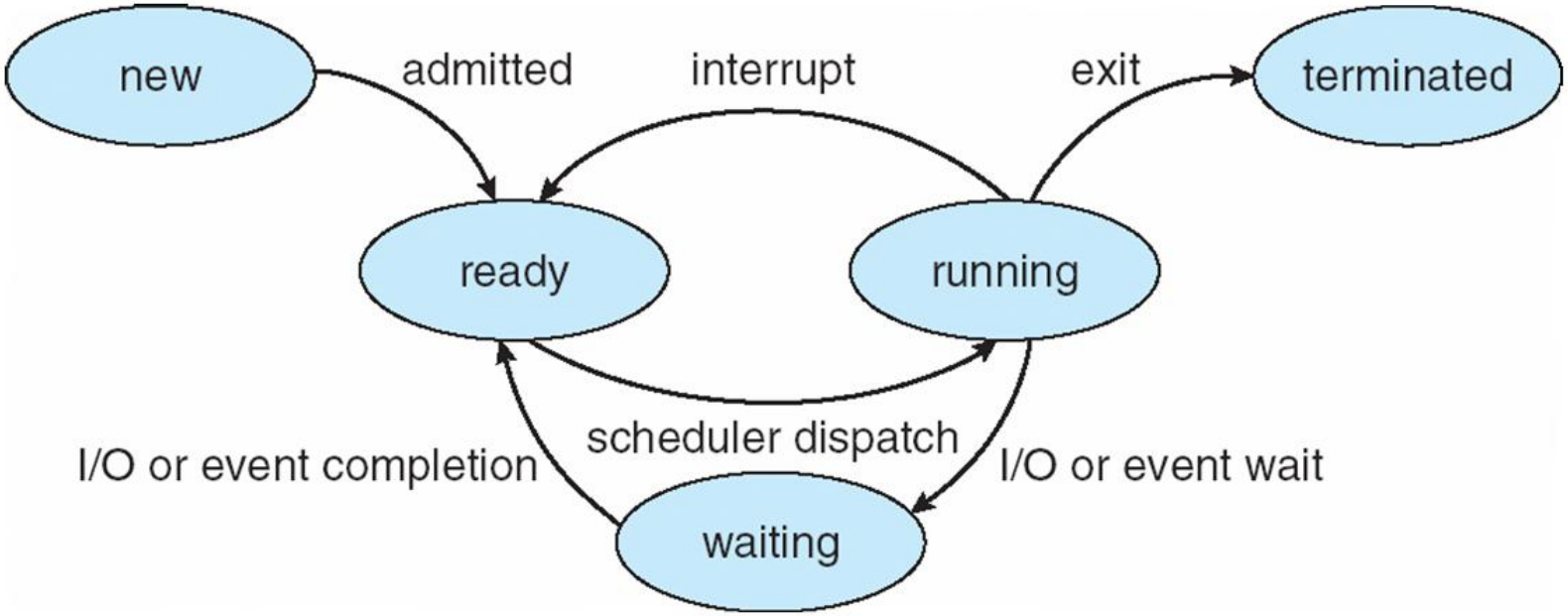
Process in Memory



Process State

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

Diagram of Process State



- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Schedulers

- Three types
  - Long Term Scheduler
  - Medium Term Scheduler
  - Short Term Scheduler

Schedulers

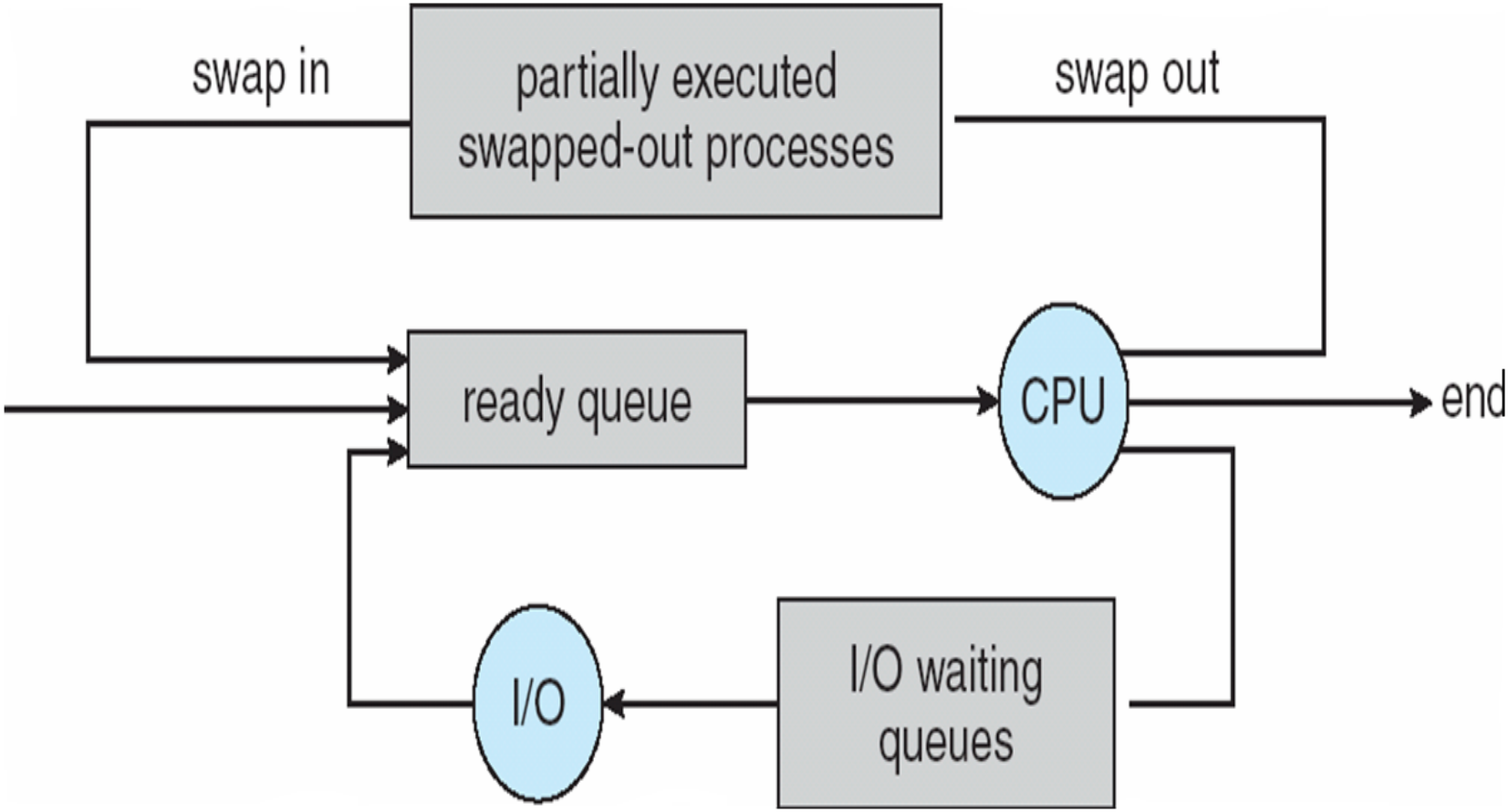
- **Long-term scheduler** (or job scheduler)
  - Selects which processes should be brought into the ready queue
  - It is invoked very in-frequently (seconds, minutes) ⇒ (may be slow)
  - Controls the degree of multiprogramming i.e the no. of processes in memory.
- **Short-term scheduler** (or CPU scheduler)
  - selects which process should be executed next and allocates CPU
  - It is invoked very frequently (milliseconds) for CPU ⇒ (must be fast)

Cont..

- Medium Term Scheduler(swapping in/out):

•Medium term scheduling is a part of the swapping.
•It move a blocked process temporarily to secondary storage.
• It reduces the degree of the multiprogramming .
• The medium term scheduler is in-charge of handling the swapped out-processes.

Addition of Medium Term Scheduling



Process Control Block (PCB)

Information associated with each process

- Process id
- Program counter
- Process state
- Priority
- CPU registers/General Purpose Registers
- CPU scheduling information
- Memory-management information
- Accounting Information
- I/O status information

Process Control Block (PCB)

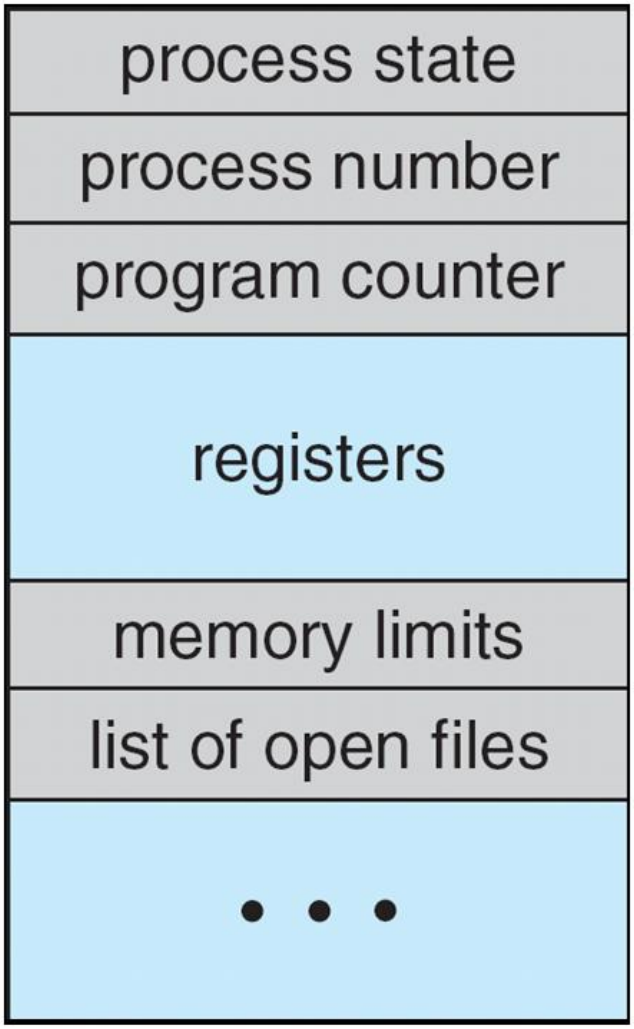
Information associated with each process

- Process Id: Every process have a unique id's
- Process state
  - The state may be new, ready, running, waiting, halted, and so on.
- Program counter
  - The counter indicates the address of the next instruction to be executed for this process
- CPU registers/General Purpose Registers
  - The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose register etc.

Process Control Block (PCB)

- CPU scheduling information
  - scheduling queues, and any other scheduling parameters.
- Memory-management information
  - the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system
- Accounting information
  - includes the amount of CPU and real time used, time limits, account numbers, job or process numbers,
- I/O status information
  - the list of I/O devices allocated to the process, a list of open files, and so on.

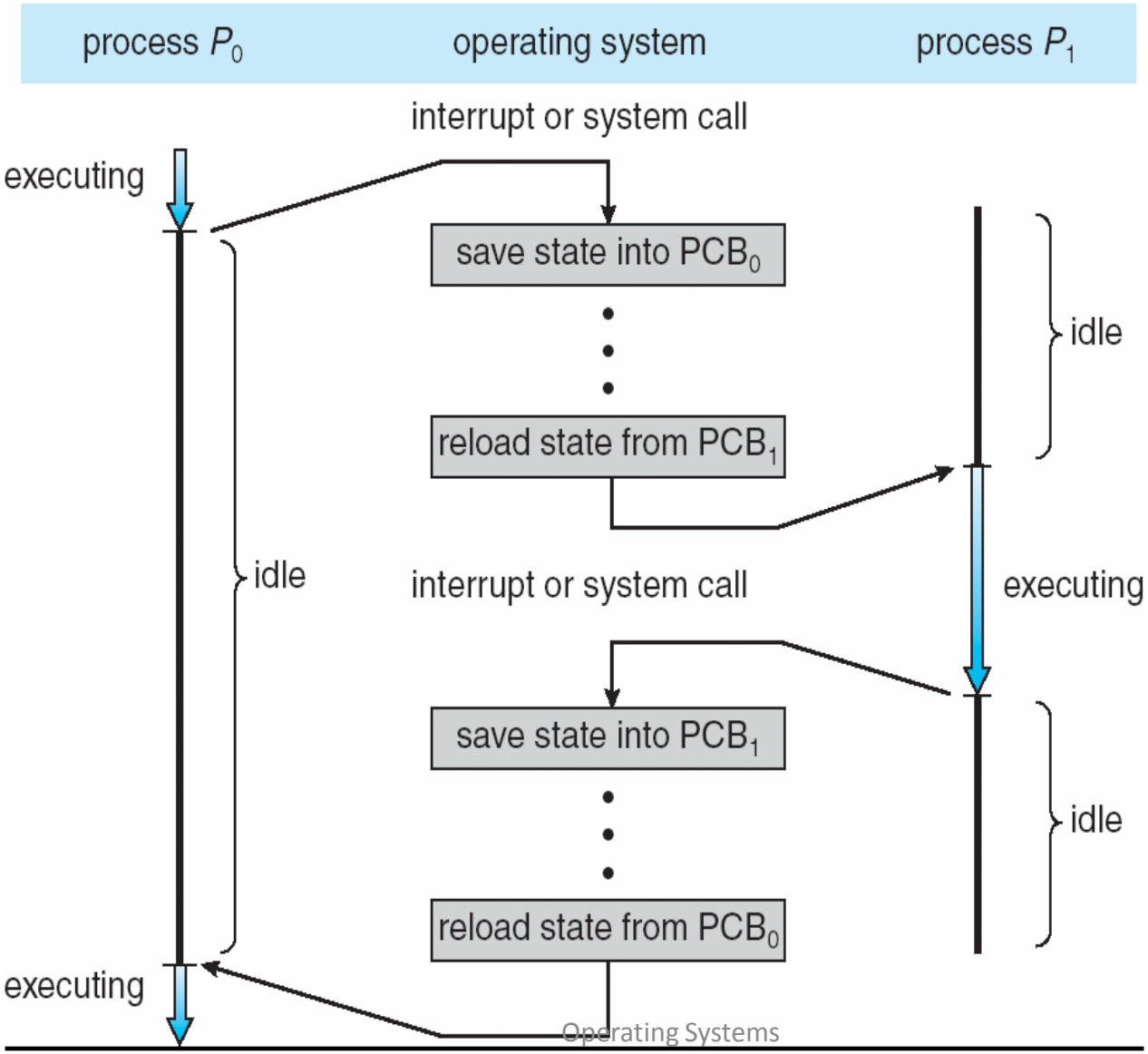
Process Control Block (PCB)



Context Switch

- Context Switch
  - When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
  - Context-switch time is overhead
  - The system does no useful work while switching
  - The more complex the OS and the PCB → longer the context switch
  - Some hardware provides multiple sets of registers per CPU →multiple contexts loaded at once

CPU Switch From Process to Process



Process Scheduling

- The main objective of multiprogramming is to have some process running at all times, to maximize CPU Utilization.
- The process scheduler is responsible to meet the objective.

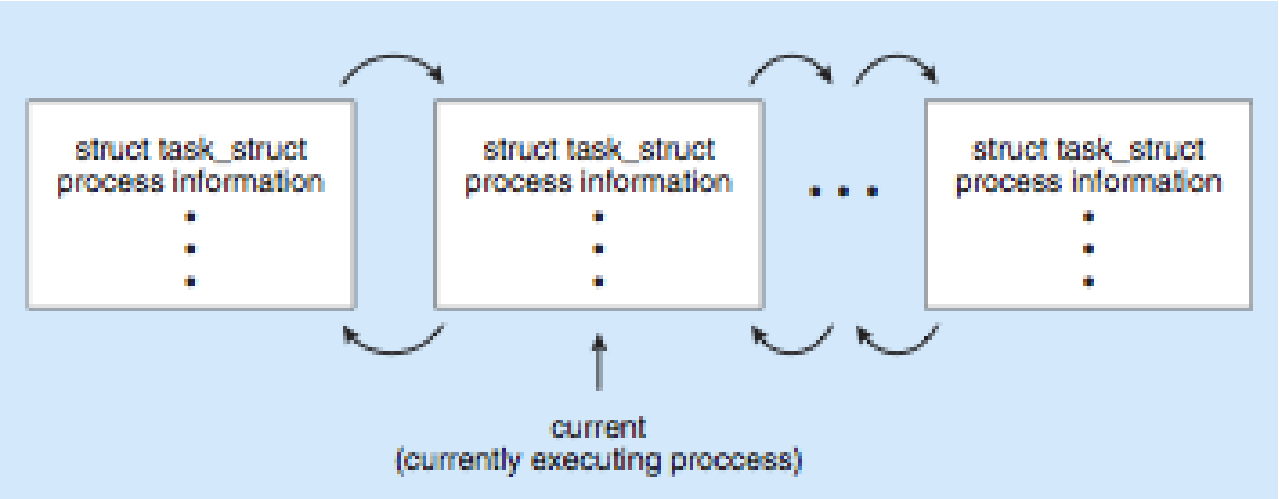
Scheduling Queues:

- **Job Queue**: All the processes are inside the system.
- **Ready Queue** :The processes which are residing in Mani Memory and are ready and waiting to be executed by the CPU.
- **Device queues** – set of processes waiting for an I/O devices.
- Processes migrate among the various queues

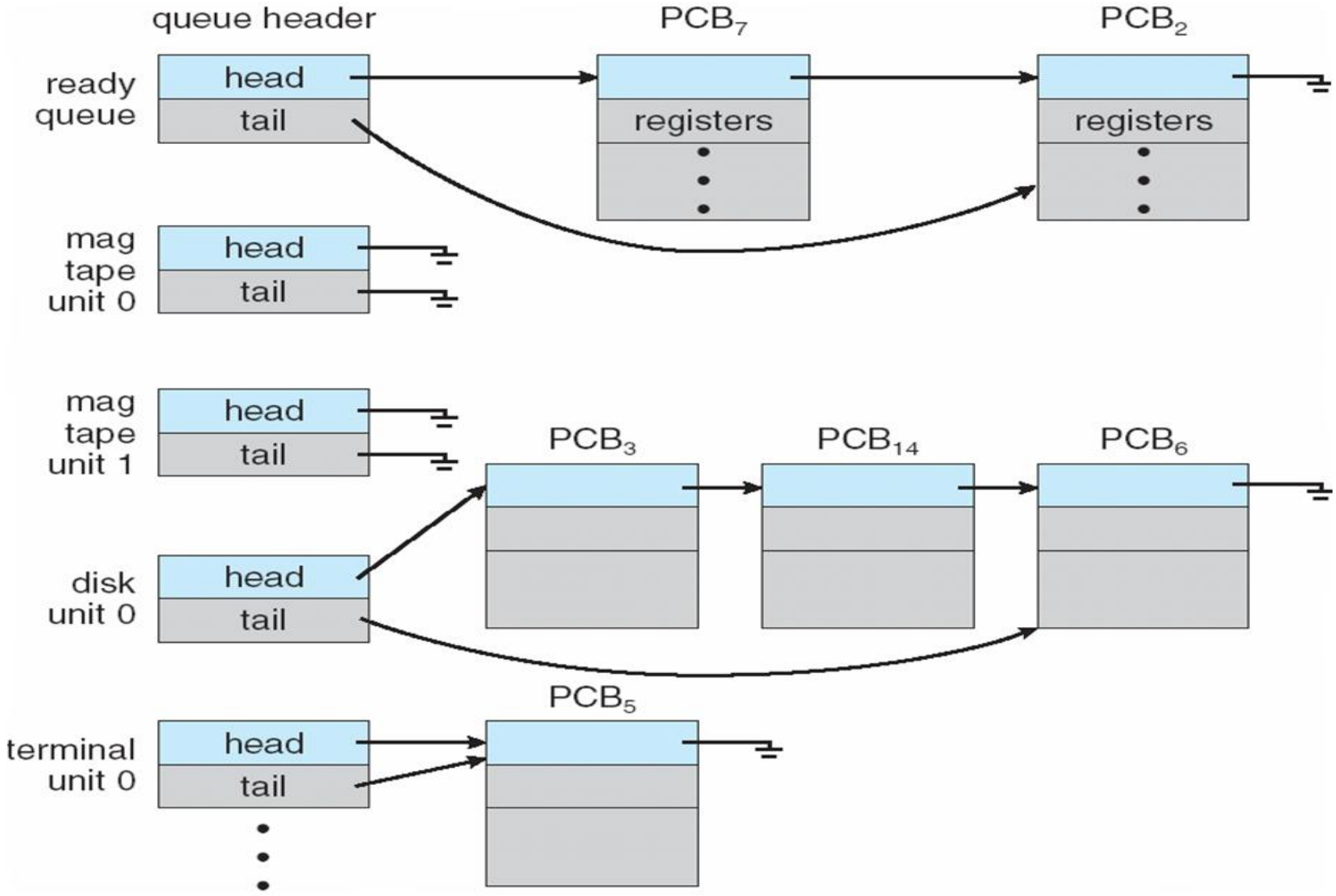
Process Representation in Linux

- Represented by the C structure `task_struct`

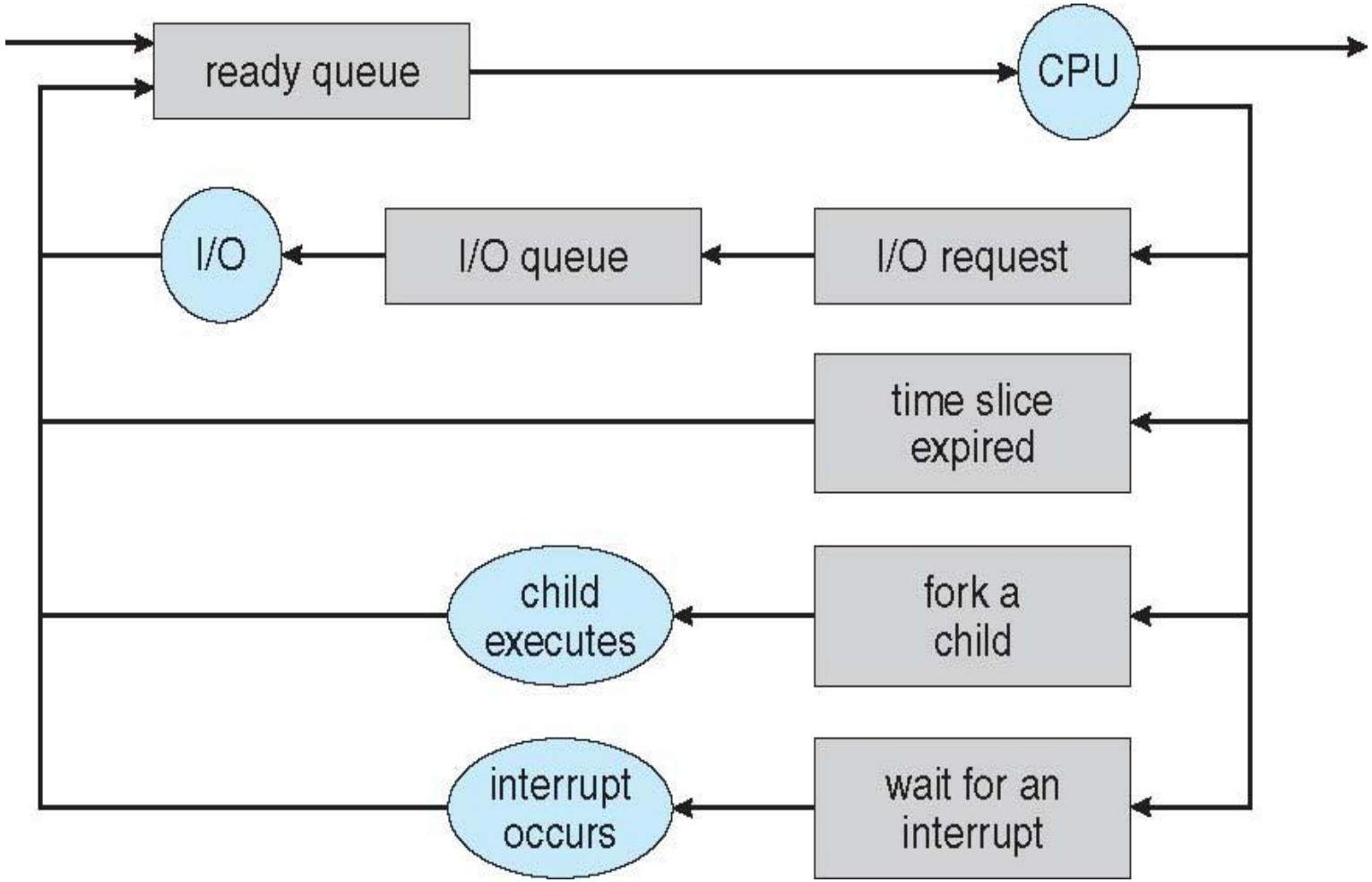
```
pid t pid; /* process identifier */
long state; /* state of the process */
unsigned int time slice /* scheduling information */ struct task struct
*parent; /* this process's parent */ struct list head children; /* this
process's children */ struct files struct *files; /* list of open files */
struct mm struct *mm; /* address space of this pro */
```



Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



Operation on Process

- Process Creation
- Process Scheduling
- Process Executing
- Process Termination

Operation on Process

- Process Creation
  - process will need certain resources (CPU time, memory, files,I/O devices) to accomplish its task.
  - These resources can be obtained from two sources
    - Operating Systems
    - Parent Process
  - The parent may have to partition/Share its resources among its children,
  - In addition to the various physical and logical resources that a process obtains when it is created, initialization data (input) may be passed along by the parent process to the child process.

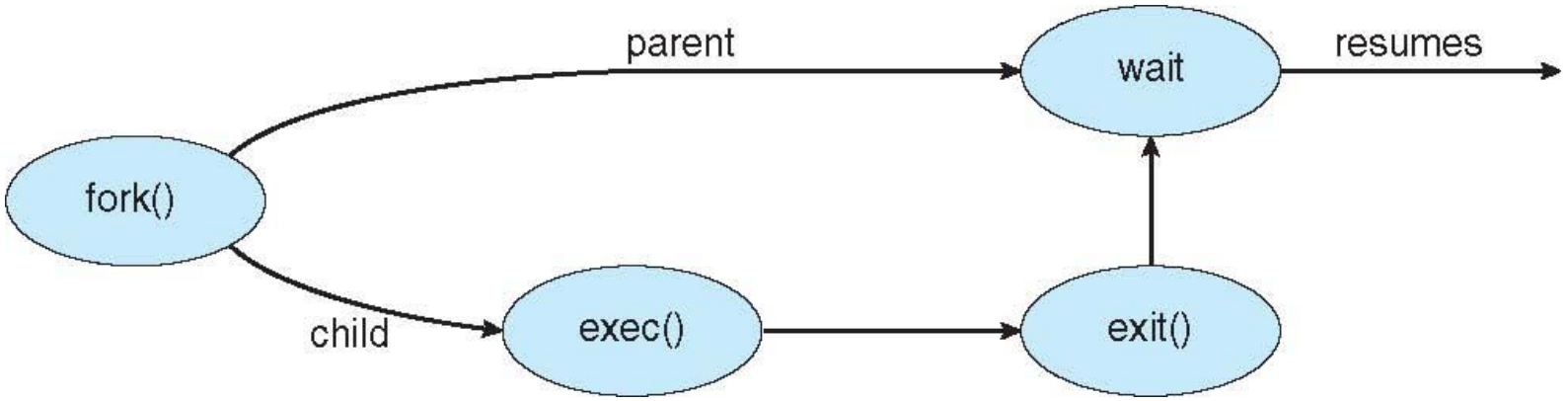
- When a process creates a new process, two possibilities exist in terms of execution

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

- These are two possibilities in terms of the address space of the new process.

1. The child process is a duplicate of the parent process.
2. The child process has a new program loaded into it.

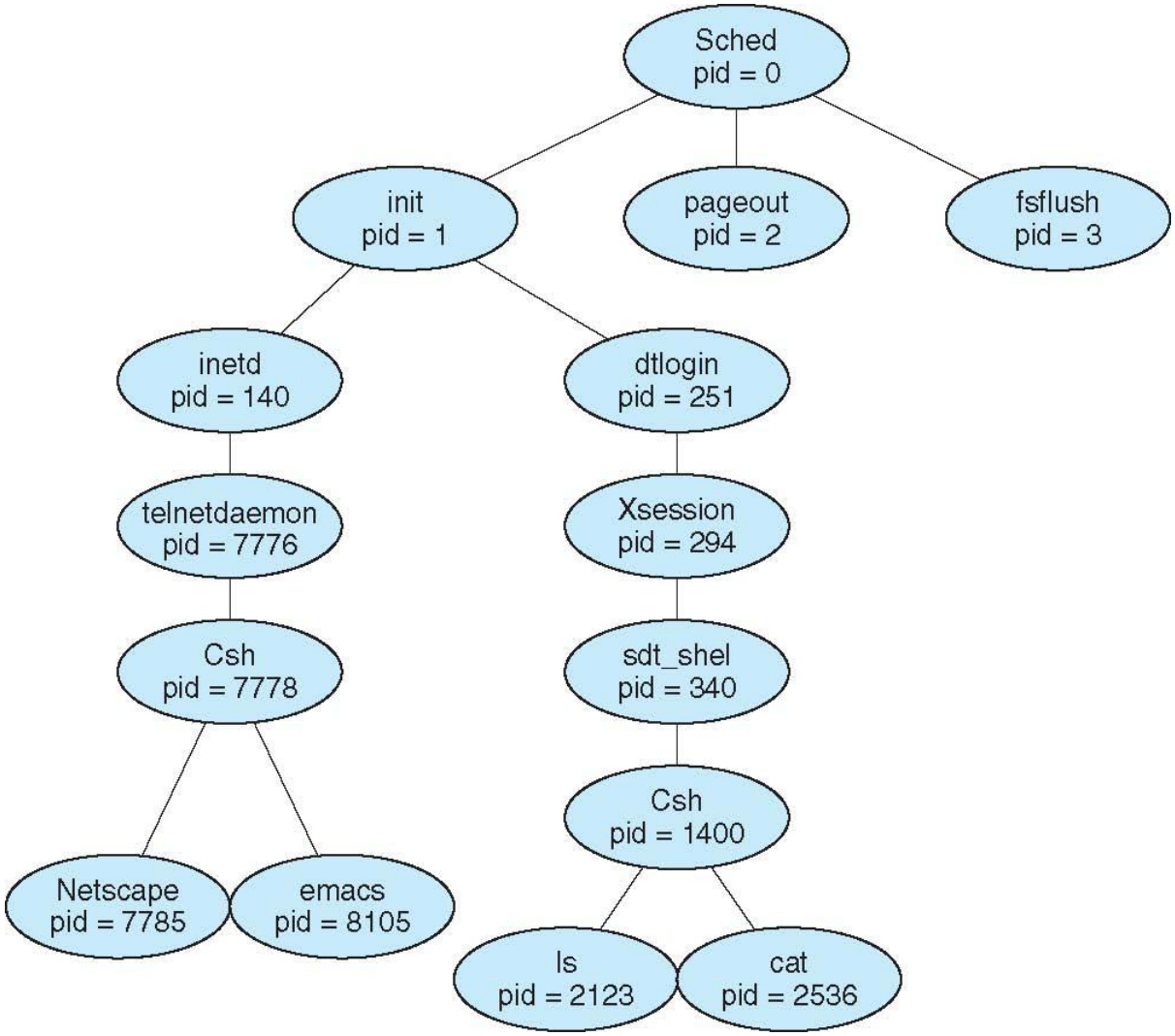
Process Creation



C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t  pid;
      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            return 1;
      }
      else if (pid == 0) { /* child process */
            execlp("/bin/ls", "ls", NULL);
      }
      else { /* parent process */
            /* parent will wait for the child */
            wait (NULL);
            printf ("Child Complete");
      }
      return 0;
}
```

A Tree of Processes on Solaris



Operation on Process

- Process Termination
  - A process terminates when it finishes executing its final statement and asks the operating system to delete it by using an appropriate system call eg. the exit() system call in Unix system
  - A parent may terminate the execution of one of its children for a variety of reasons, such as these:
    - The child has exceeded its usage of some of the resources that it has been allocated.
    - The task assigned to the child is no longer required.
    - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

Cooperating Processes

- **Independent** process: Any process that does not share the data with any other process

- **Cooperating/Dependent** process: Any process that share the data with any other process

- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

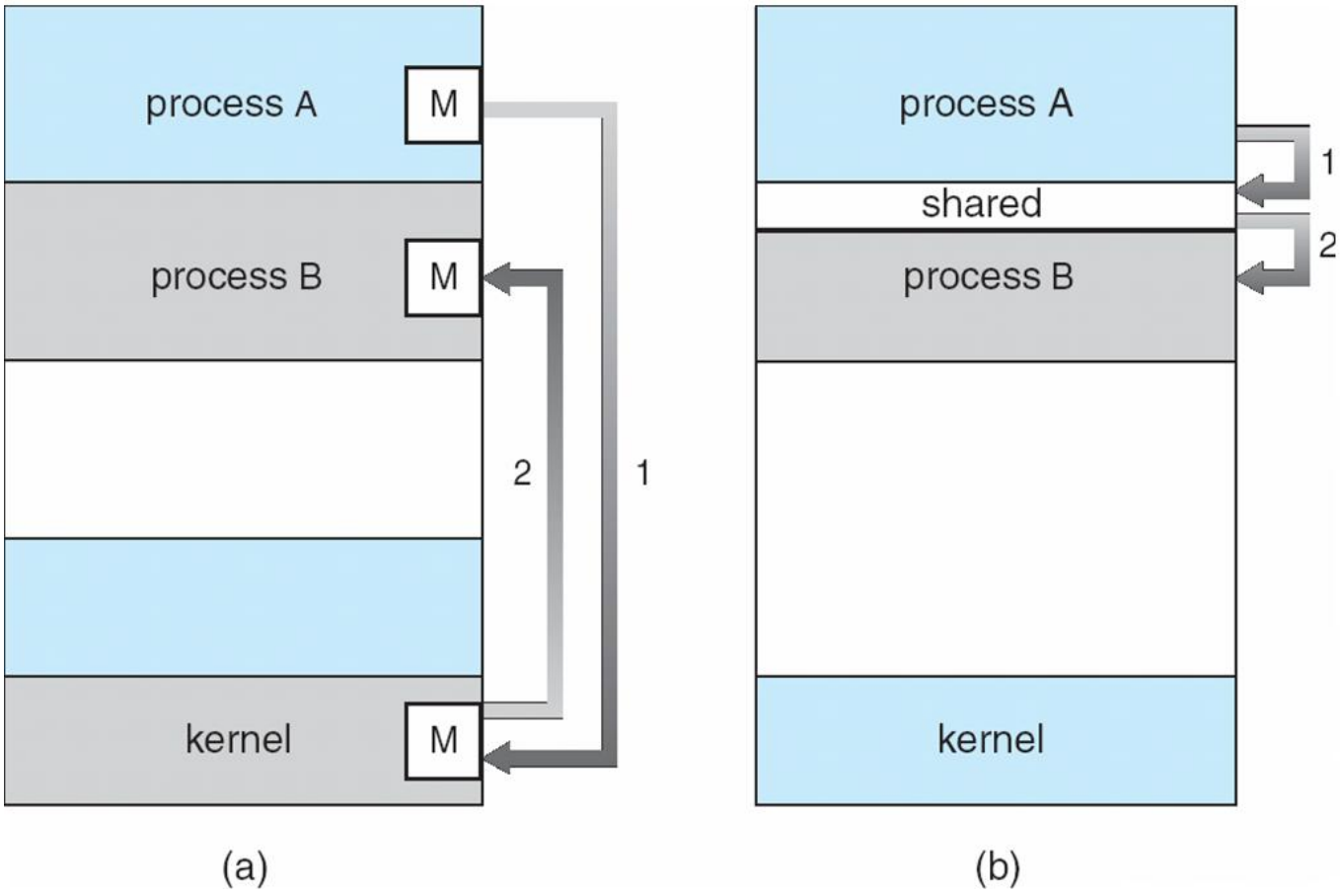
- Different ways to communicate with Processes.

1. Processes can communicate using files.
2. Os supports something called as pipes.
3. Processes could communicate through variables that are shared between them.
4. Processes could communicate by sending & receiving messages to each other.

Inter process Communication

- Cooperating processes need **inter process communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



Communications Models

- Shared Memory Model
  - In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region
- Message Passing Model
  - In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.

Message passing

- It refers to a mean of communication between

1. Different threads within a process.
2. different processes running on same node.
3. different processes running on different nodes.

- In this a sender or a source process sends a message to a known receiver or destination process.
- Message has a predefined structure and message passing uses two system calls send & receive.

send(name of destination, message);

receive(name of the source, message);

- Mode of communication can take place through two methods

Direct addressing

Indirect addressing.

- **Direct addressing**: In this type , the two process should need to know each other to communicate.

1. **symmetric mode**: Both sender and receiver should know their identity.
2. **Asymmetric mode**: The receiver will not know where the information is coming from

- **Indirect addressing**: In this message send & receive from a mailbox . A mailbox can be abstractly viewed as an object into which a message may be placed & from which message may be removed by process.
- The sender & receiver process should share that mailbox to communicate

Different communication links are:

1. one to one link
2. Many to one link
3. one to many link
4. many to many link

Communication Models

Message Passing systems	Shared Memory Systems
Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided.	Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.
Message passing is easier to implement than Shared Memory systems communication	Difficult to implement than Message Passing Systems
Message-passing systems are typically implemented using system calls and thus require the more time consuming task of kernel intervention.	Shared memory is faster than message passing, as in shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required

Shared Memory Systems

- A shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space
- The form of the data and the location are determined by these processes and are not under the operating system's control.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
- A producer process produces information that is consumed by a consumer process.
- Among this producer/consumer process the situations can happen like this:

-→ As the producer generates the item is same as the rate at which the consumer process consumes the items.

Producer-Consumer Problem

→If the producer process produces the items at faster rate, then the rate at which the consumer process can consume the items so naturally some of the items will lost.

→ Similarly, if the consumer process can consume at faster rate, then the rate at which the items produced by the producer will be less, i.e., the consumer process has to wait for the availability of items.

Producer-Consumer Problem

So, instead of sending the data directly from producer to consumer, in between the producer and consumer we have a buffer pool.

In this buffer pool, we have two types of situations:

- *unbounded-buffer* places no practical limit on the size of the buffer
- *bounded-buffer* assumes that there is a fixed limit on the buffer size

Bounded-Buffer – Shared-Memory Solution

- Shared data

type item= ;//type item as a variable type item

Var buffer=array[0…..n-1] of item;//buffer is an array having ‘n’ no. of locations

in,out=0…..n-1;//to point the items in the buffer

next p,next c: item;

in=0;

Out=0;

Bounded-Buffer – Producer

producer: request

begin

repeat

---

produce an item in the next p;

while(count=n) do skip;//buffer is full

while(in+1)mod n=out do skip;//buffer is full, producer process keeps on executing this while loop

buffer[in]=next p;//produces the items at a buffer location 'in'

in=[in+1]mod n;//its circular buffer in nature

count=count+1;//After placing the new items in the buffer it increments the count value by 1

until false;

end

Bounded-Buffer – Consumer

consumer: begin

repeat

while(count=0) do skip;//buffer is empty

while in=out do skip;//consumer process will wait in this while loop as buffer is empty

next c=buffer[out];//consumer process takes the items from the buffer into a variable next c

out=(out+1) modn;

---

consumes item in next c;

---

count=count-1;//once the consumer process consumes the items from the buffer, it will reduce the count by 1.

until false;

end

Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Message Passing

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Synchronization

Message passing may be either blocking or nonblocking

- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

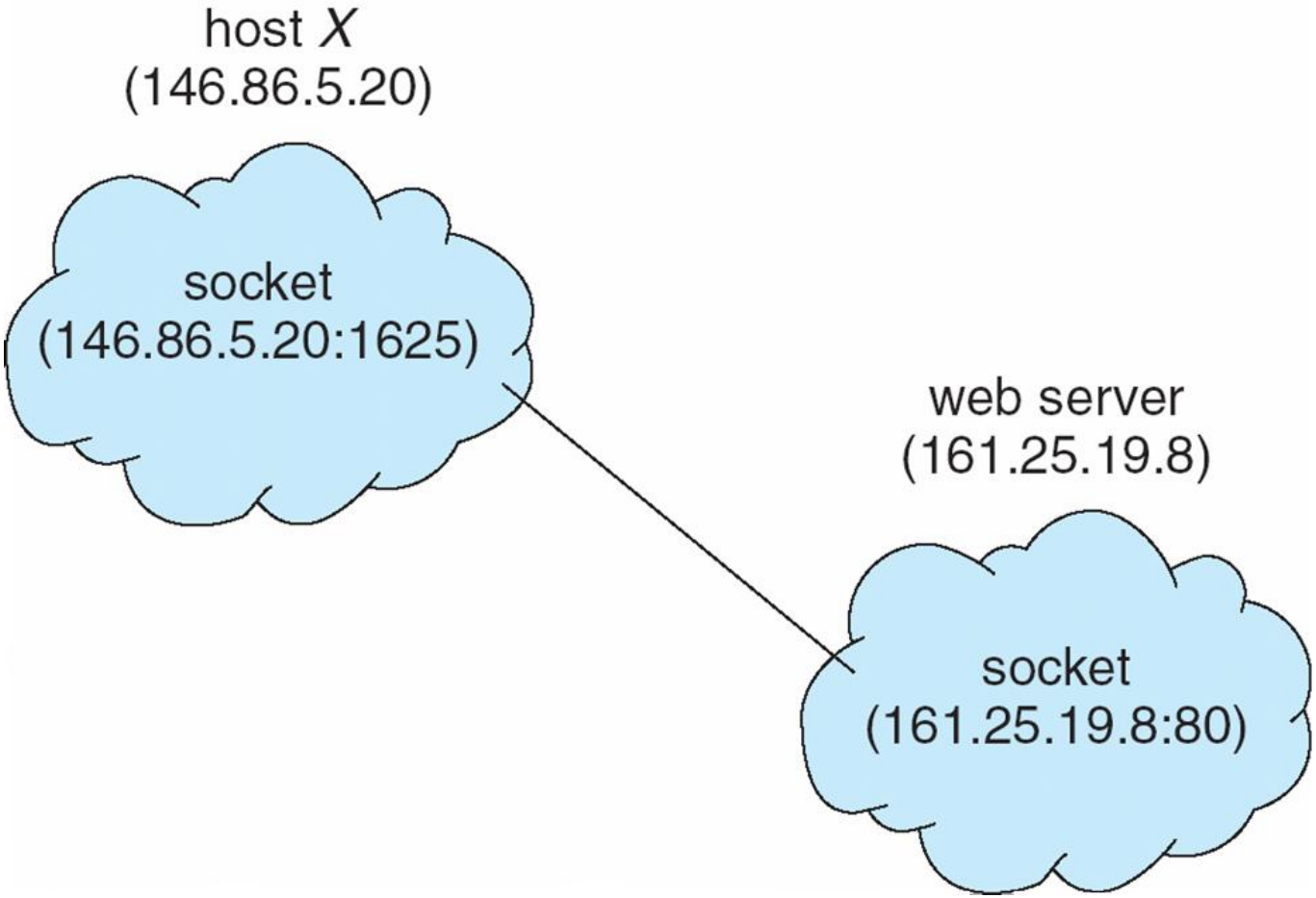
Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)

Sockets

- A socket is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
- Communication consists between a pair of sockets

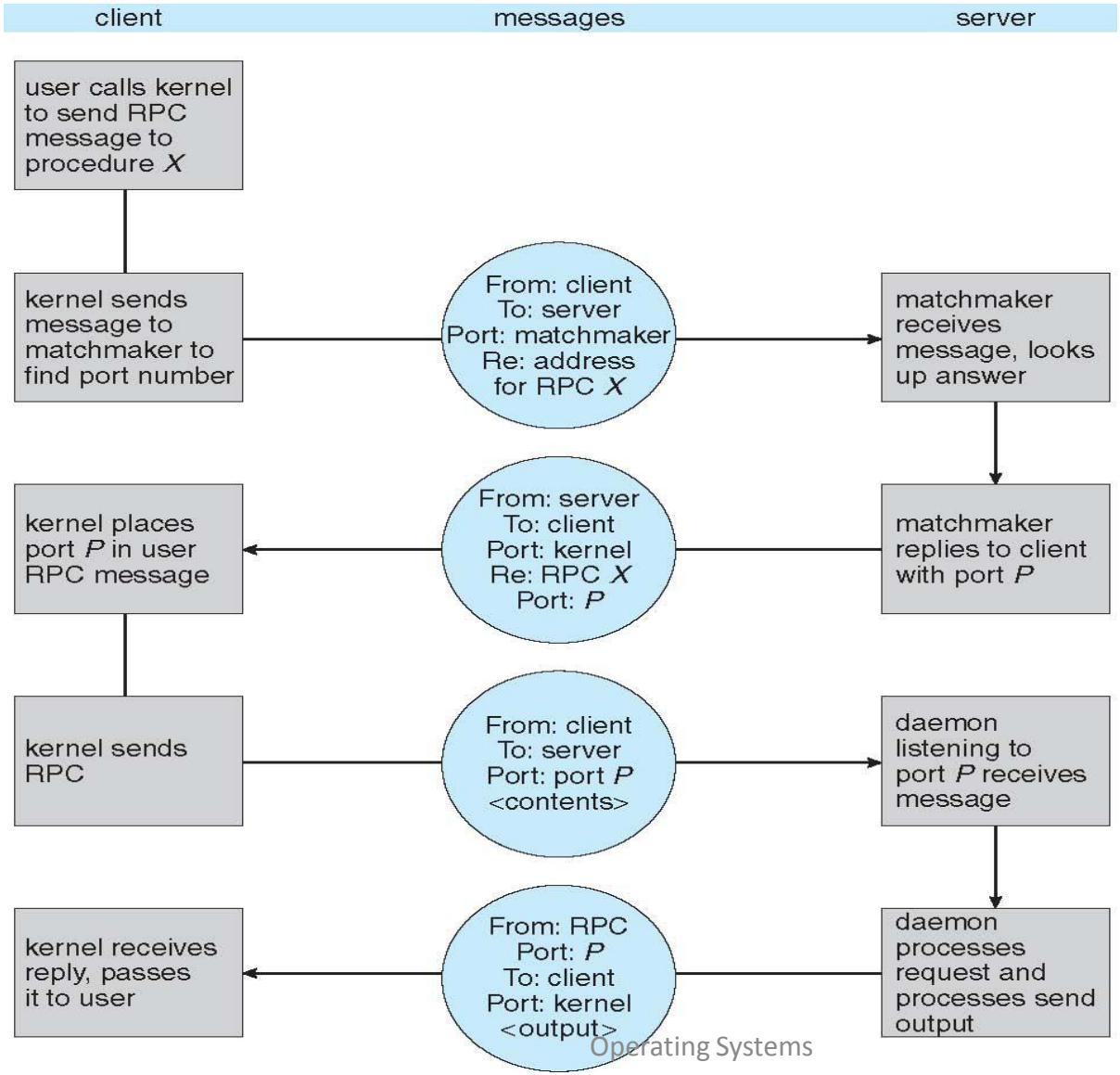
Socket Communication



Remote Procedure Calls

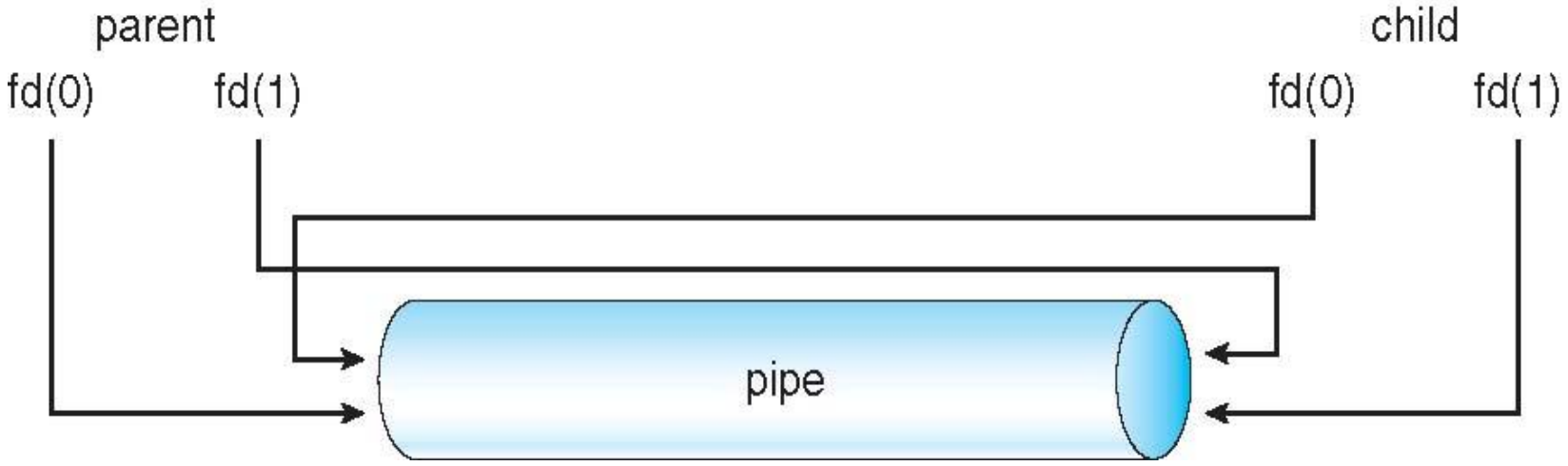
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and *marshalls* the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

Execution of RPC



Ordinary Pipes

- **Ordinary Pipes** allow communication in standard producer-consumer style
- Producer writes to one end (the *write-end* of the pipe)
- Consumer reads from the other end (the *read-end* of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



UNIT-III
Syllabus

Memory Management and Virtual Memory – Logical & Physical Address Space, Swapping, Contiguous Allocation, Paging, Structure of Page Table, Segmentation, Segmentation with Paging, Virtual Memory, Demand Paging, Performance of Demanding Paging, Page Replacement, Page Replacement Algorithms, Allocation of Frames, Thrashing.

Basic Hardware

- Main memory and the registers built into the processor itself are the only storage that the CPU can access directly.
- The Instructions in execution and any data being used by the Instructions, must be in one of these direct-access storage devices.
- Registers that are built into CPU generally accessible within one cycle of CPU clock.
- The same cannot be said as Main Memory, which is accessed via a transaction on the memory bus, memory access may take many CPU cycles.
- Remedy is to add fast memory between the CPU and Main Memory called a cache.

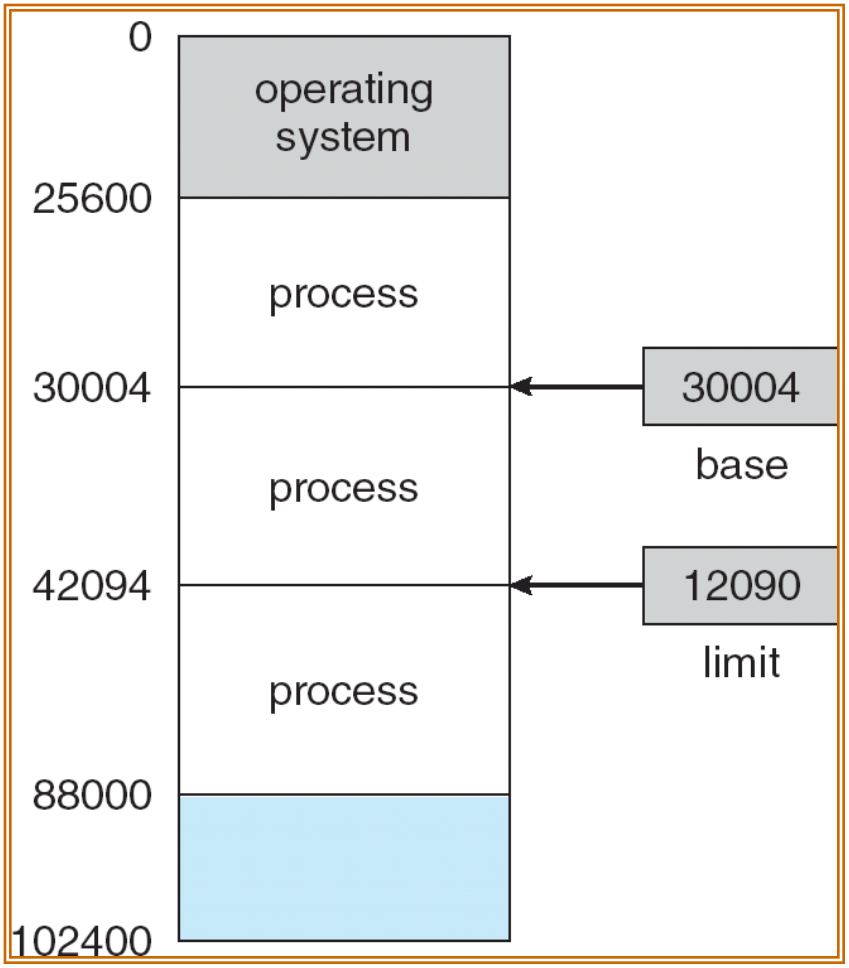
- Ensuring correct operation to protect the operating system from access by user processes and also to protect user processes from one another.
- The protection is provided by hardware.
- First of all we need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- We can provide this protection by using two registers

1.Base Register(Holds the smallest legal physical memory address)

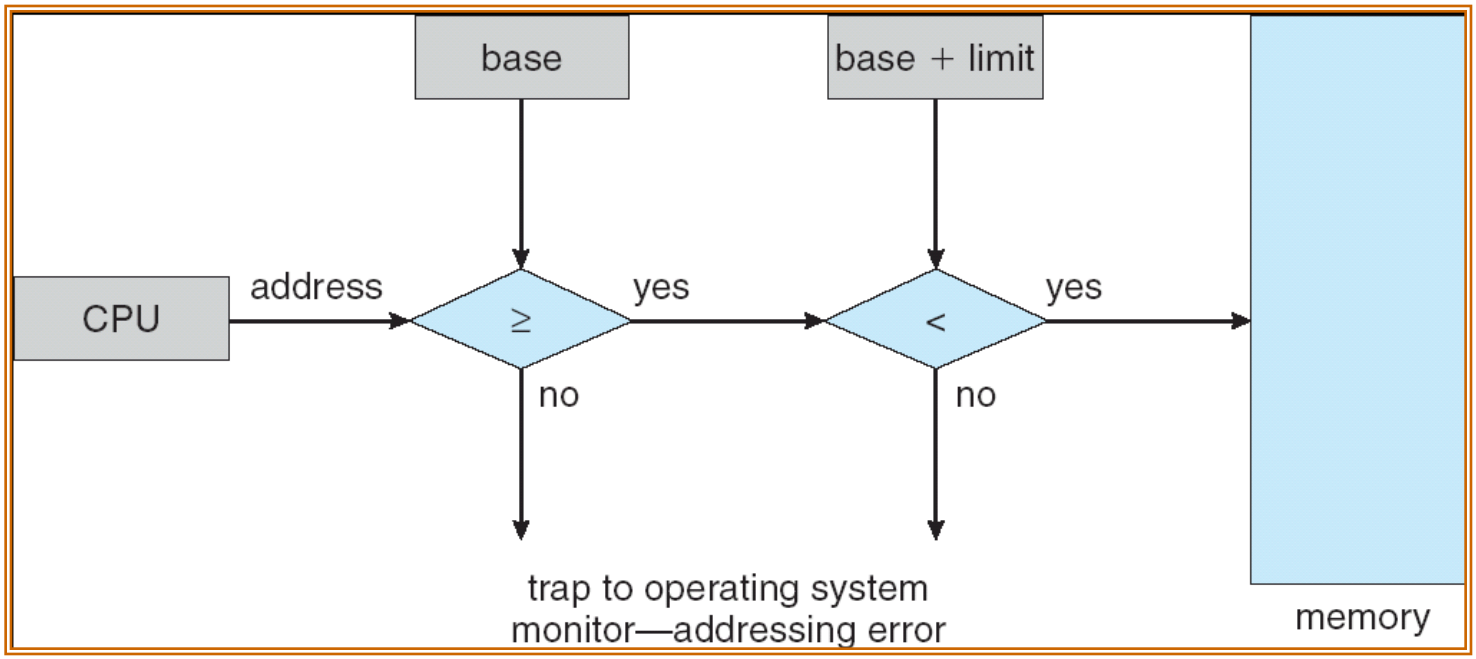
2.Limit Register(specifies the size of the range)

eg:If base register holds 300040 and the limit register is 120900 then the program can legally access all addresses from 300040 to 420939.

A base and a limit register define a logical address space



HW address protection with base and limit registers



Address Binding

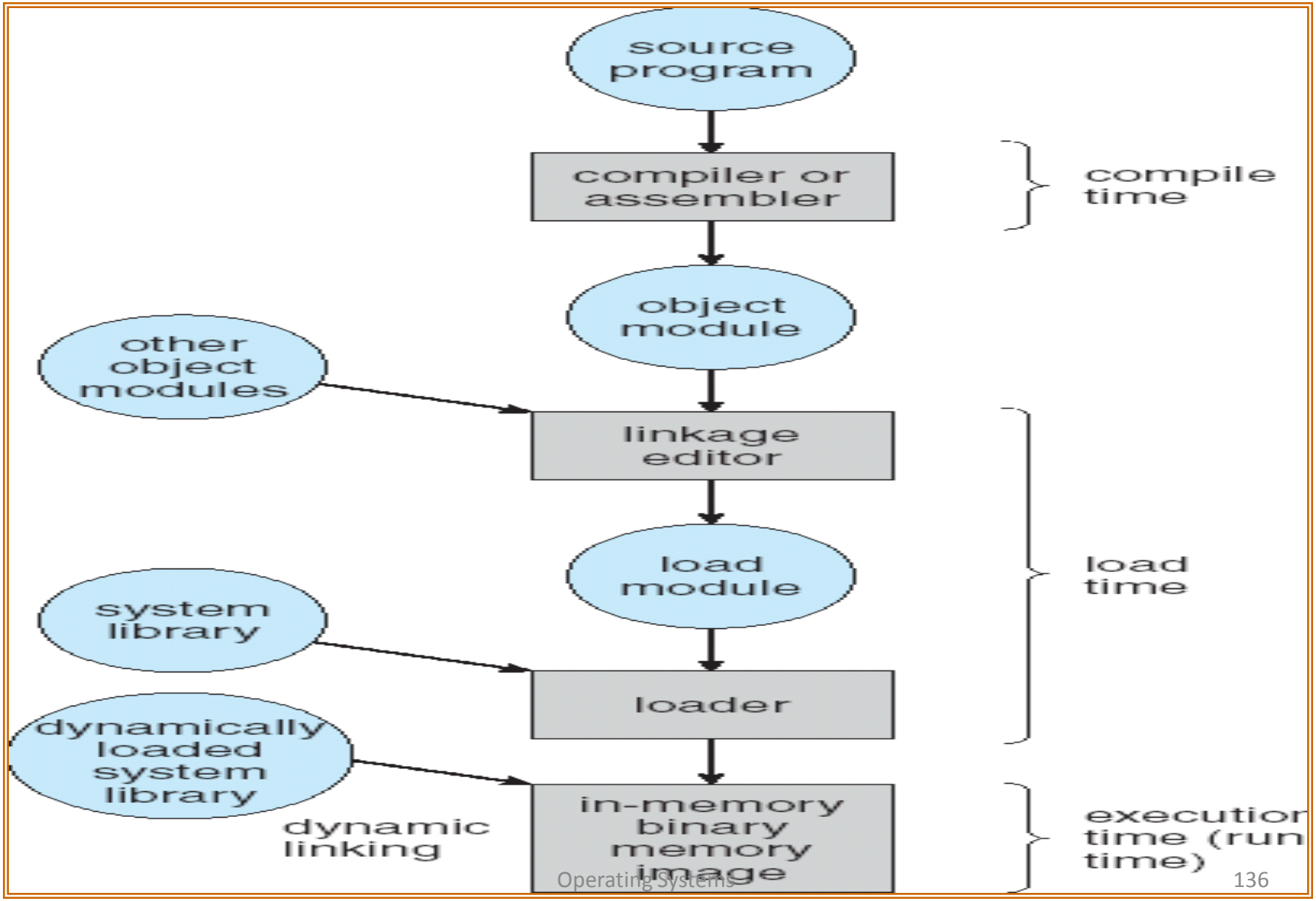
- Normally, a program resides on a disk as a binary executable file.
- To be executed, the program must be brought into memory and placed with in a process.
- Depending on the memory management in use, the program may be moved between disk and memory during its execution.
- The processes on the disk that are waiting to be brought into memory for execution form the input queue.
- As a process is executed, it accesses instructions and data from memory. Eventually, the process terminates, and its memory space is declared as available.

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, *absolute code* can be generated; must recompile code if starting location changes
- **Load time**: Must generate *relocatable code* if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).

Multistep Processing of a User Program



Dynamic Loading

1. To obtain better memory utilization , we can use dynamic loading.
2. The main program is loaded into memory and is executed .
3. All routines are placed in a relocatable load format, with dynamic loading a routine is not loaded until it is called.

Advantages:

1. Better Memory space utilization.
2. Performance of CPU Increases.

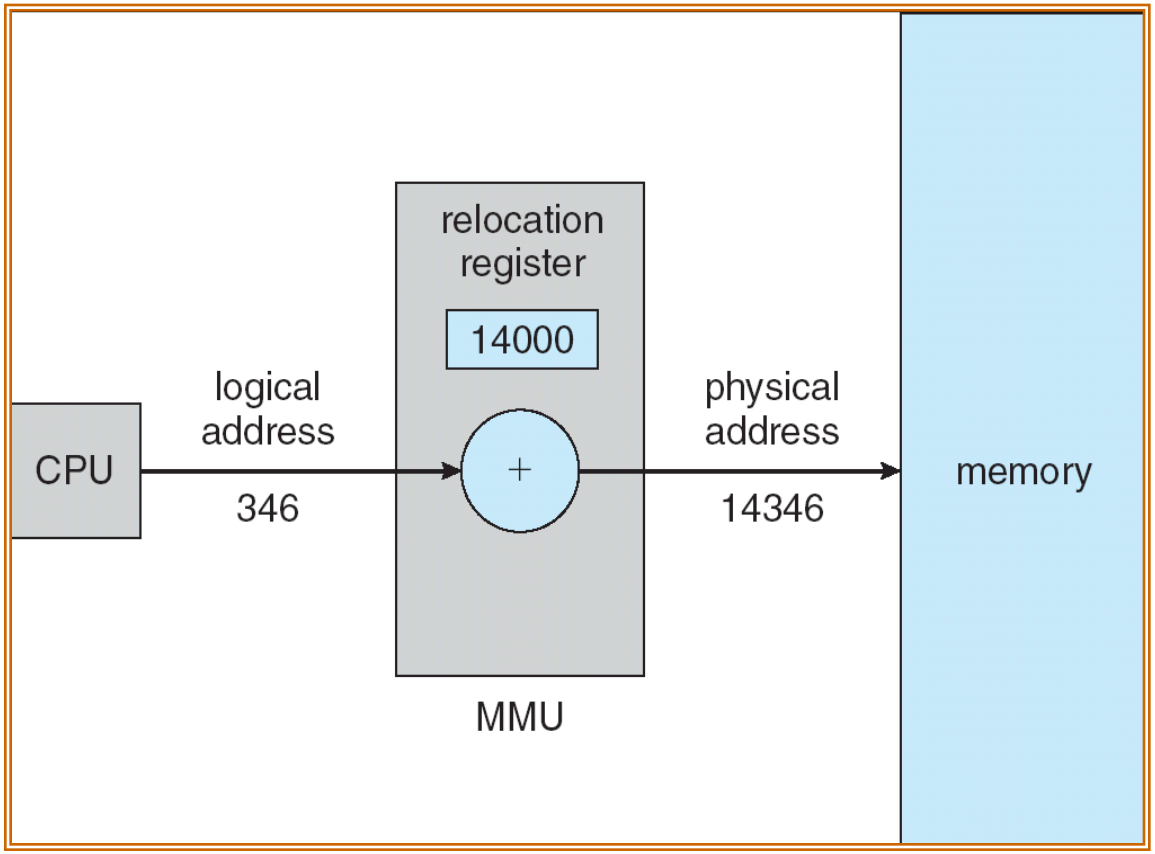
Dynamic Linking& Shared Libraries

- Some operating system supports Static Linking & some supports Dynamic Linking.
- Linking feature is usually used with system libraries , such as language subroutine libraries.
- The CPU links the dependent programs to main executing program when it is needed.

Logical vs. Physical Address Space

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

Dynamic relocation using a relocation register



Memory-Management Unit (MMU)

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

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

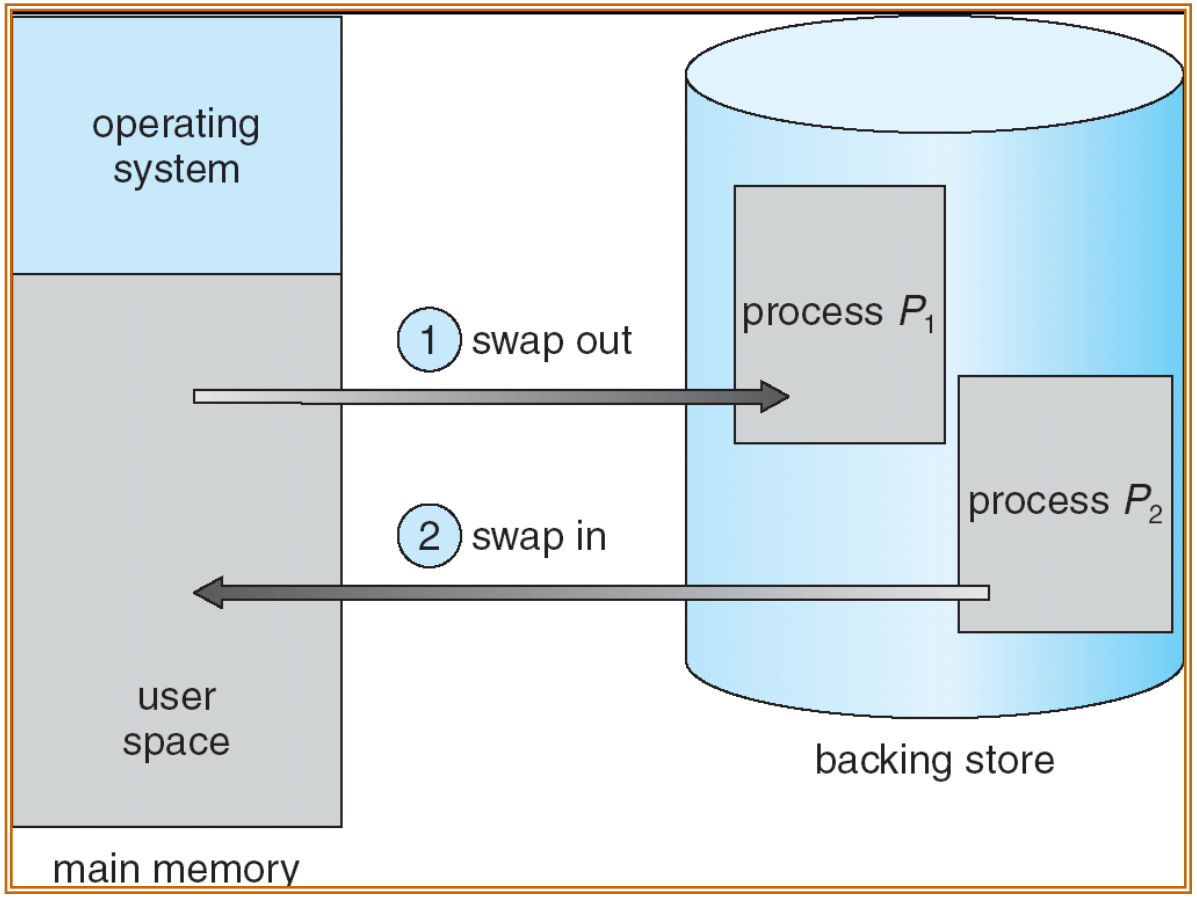
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images

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

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

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Schematic View of Swapping



Contiguous Memory Allocation

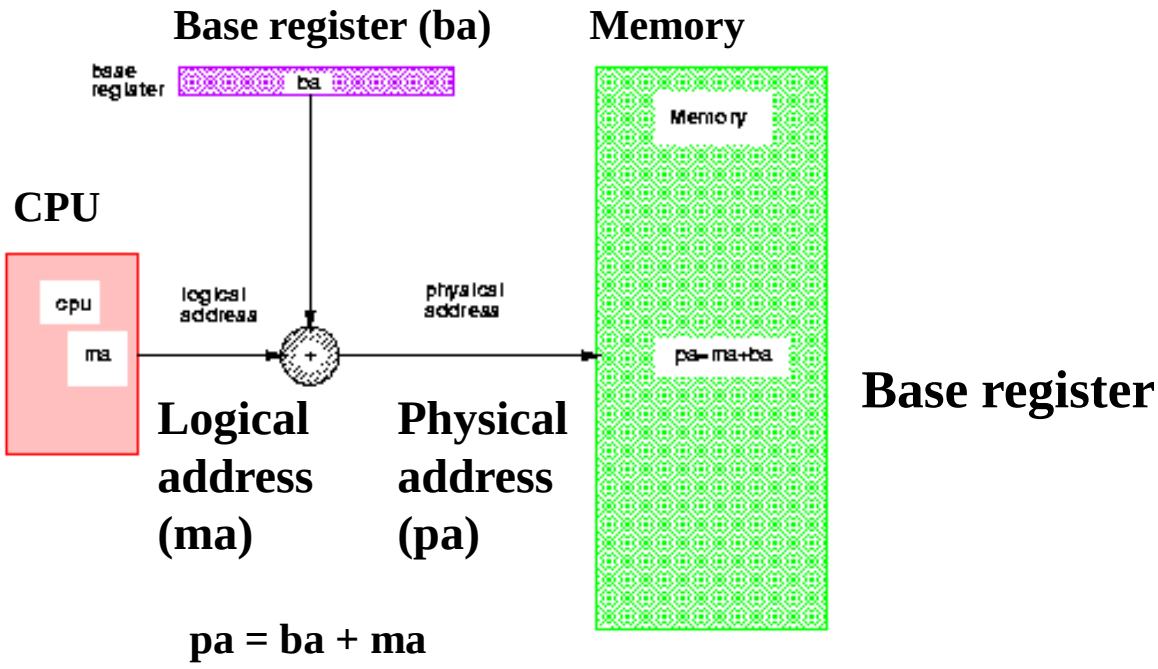
- Main memory usually into two partitions:
  - Resident **operating system,** usually held in **low memory** with interrupt vector
  - User **processes** then held in **high memory**
  - Each process contained in single contiguous sections of memory
- Relocation register used to protect user processes from each other, and from changing operating system code and data

Base register contains value of smallest physical address

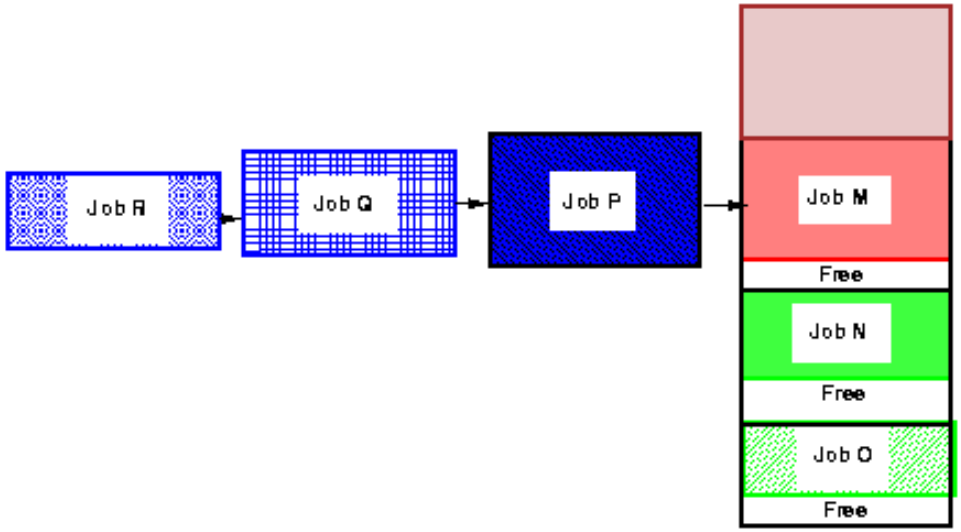
Limit register contains range of logical addresses-each logical address must be less than the limit resister.

MMU maps logical address dynamically

Hardware support for relocation and limit Register



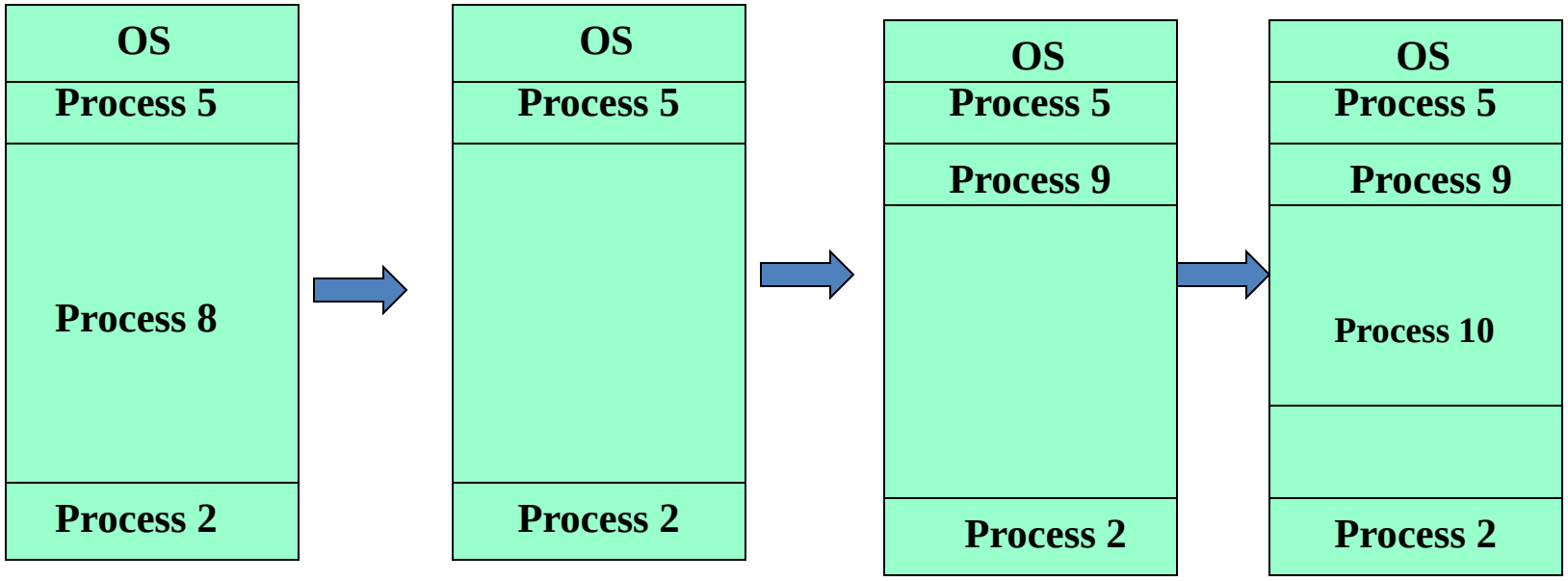
Fixed partitions



Contiguous Allocation (cont.)

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

Contiguous Allocation example



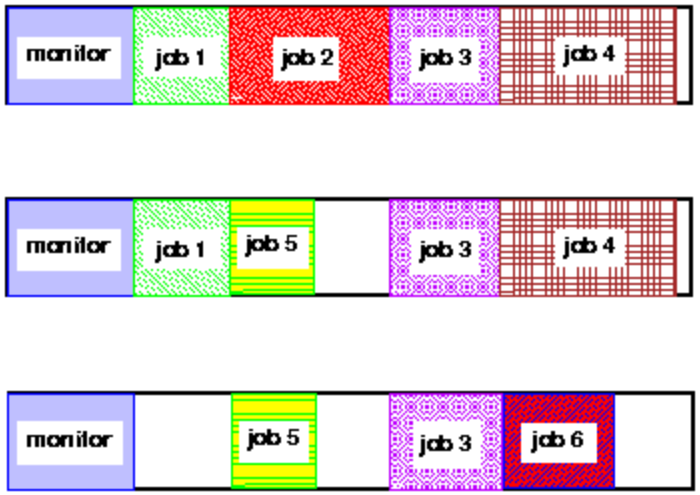
Dynamic Storage Allocation Problem

- How to satisfy a request of size n from a list of free holes.
  - First-fit
    - allocate the first hole that is big enough
  - Best-fit
    - Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
  - Worst-fit
    - Allocate the largest hole; must also search entire list. Produces the largest leftover hole.
- First-fit and best-fit are better than worst-fit in terms of speed and storage utilization.

Fragmentation

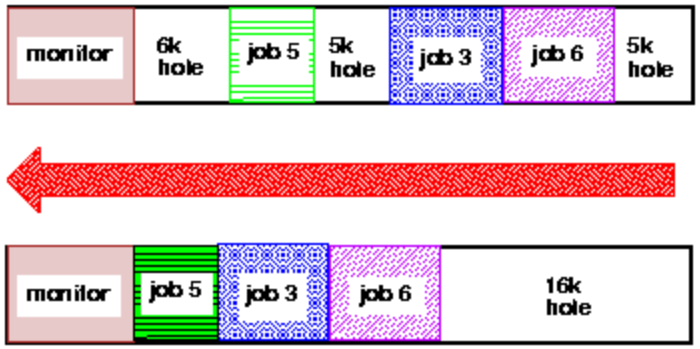
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Though Compaction is possible it is a costlier solution.
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Fragmentation example



Compaction

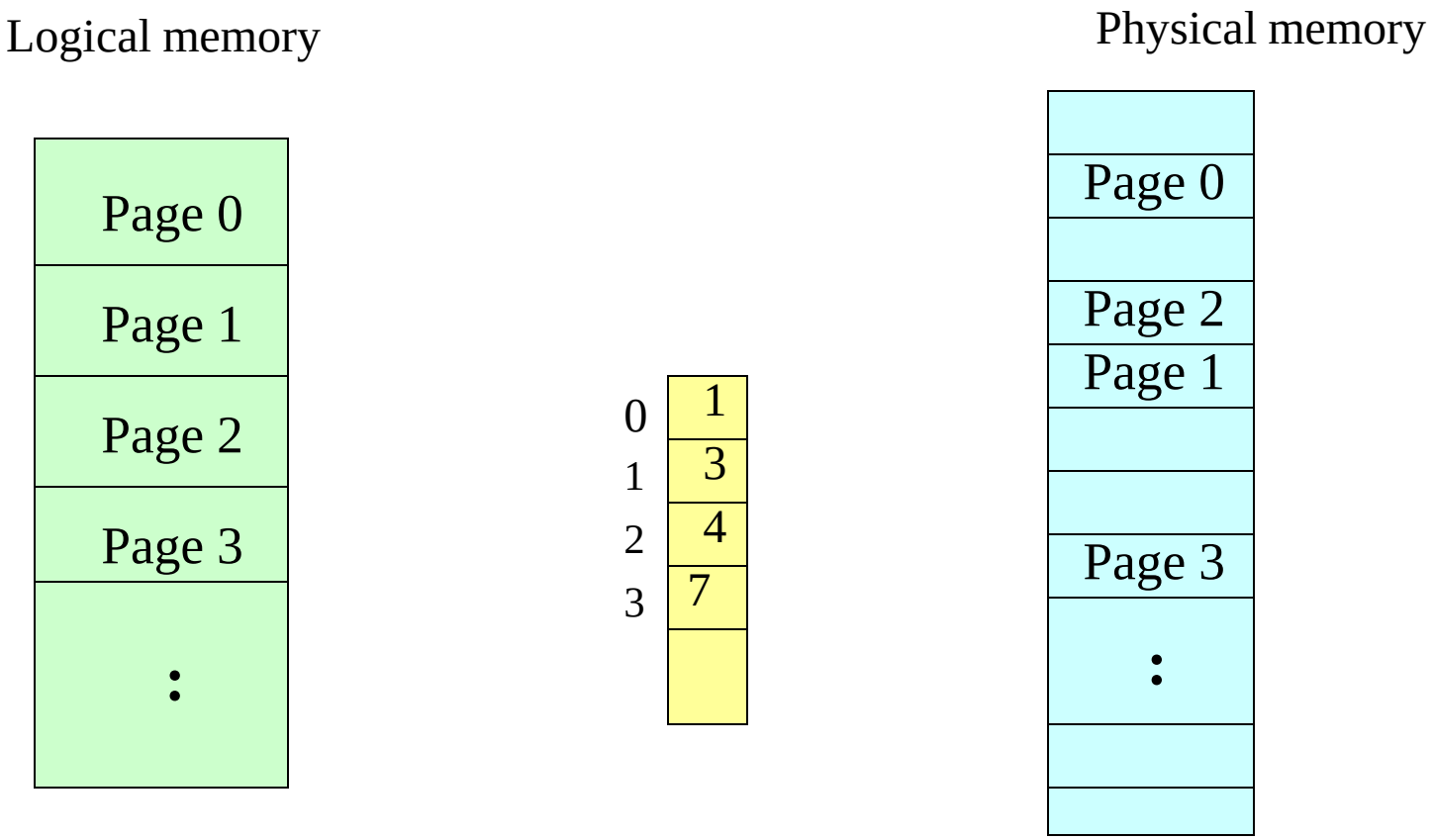
Memory **compaction** is the process of moving allocated objects together, and leaving empty space together. Consider a system with 3 pages and about 50% of their objects are allocated. By **compacting** all the living objects into the first two pages, leaving the third page completely empty.



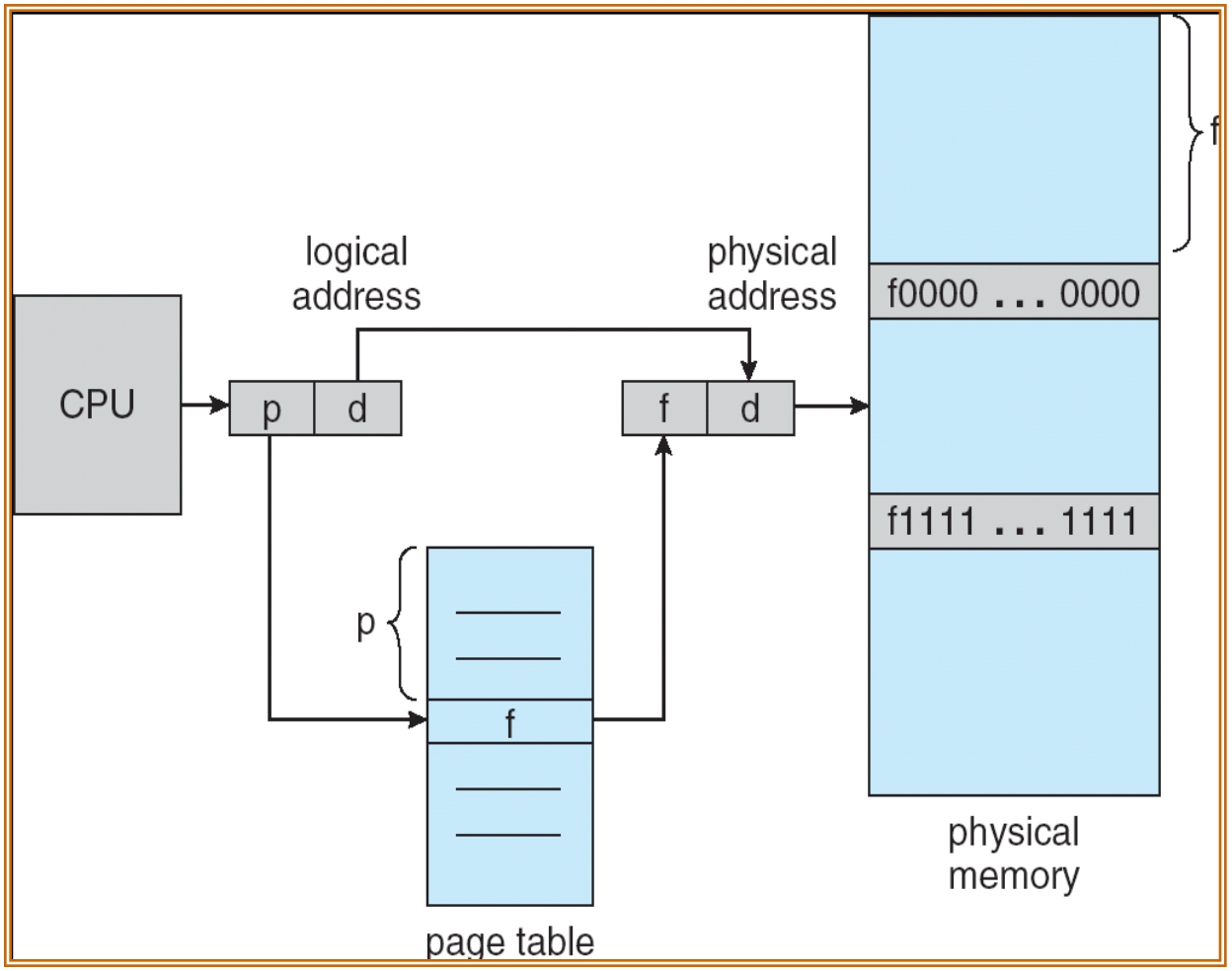
Paging

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

Example of Paging



Paging Hardware



Paging Hardware(Cont.)

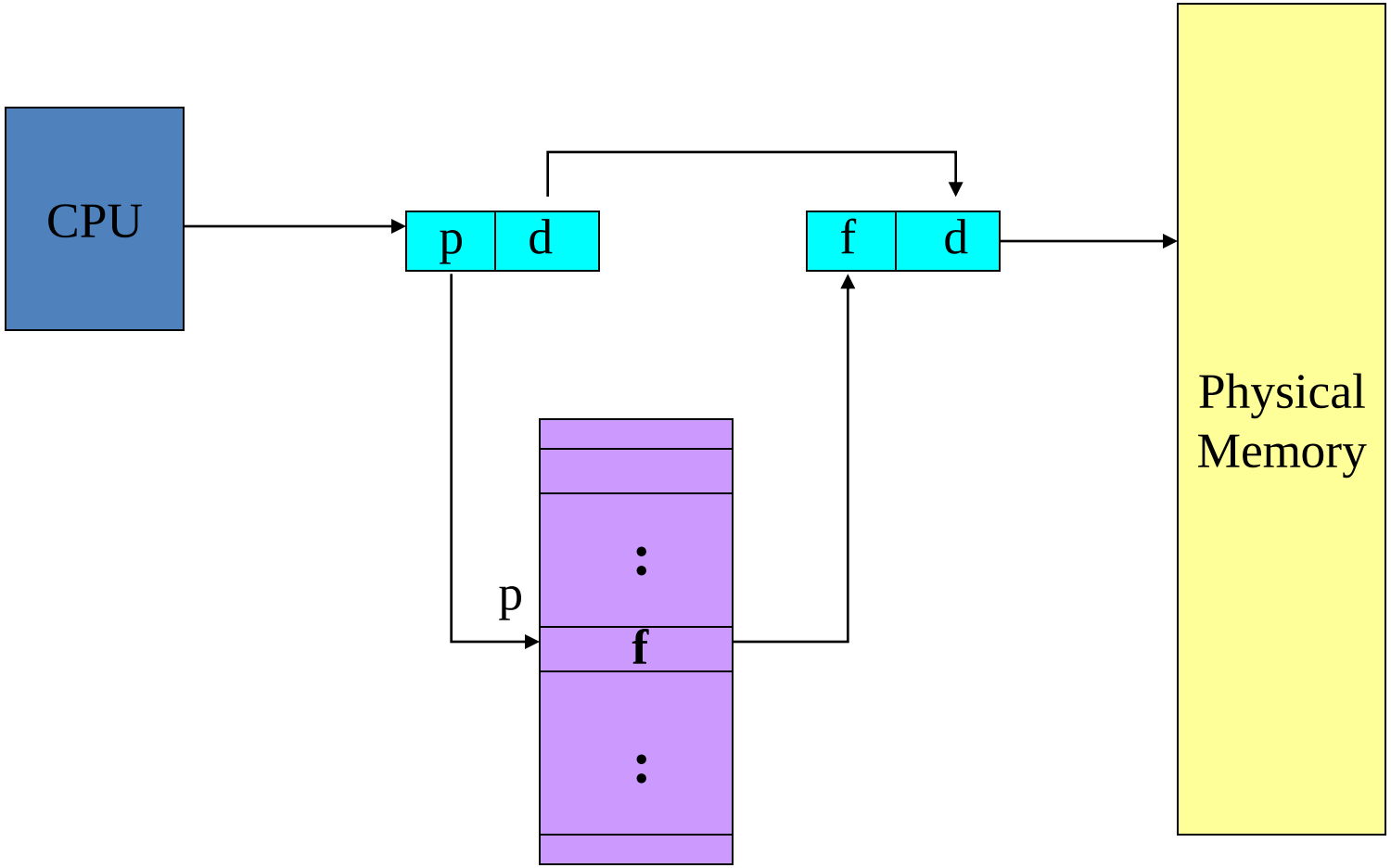
The hardware support for paging is illustrated in the above figure.

- Every address generated by the CPU is divided into two parts: **a page number(p) and a page offset(d)**.
- The page number is used as an index into a **page table**.
- The **page table** contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
- This paging model of memory is shown in the below figure:

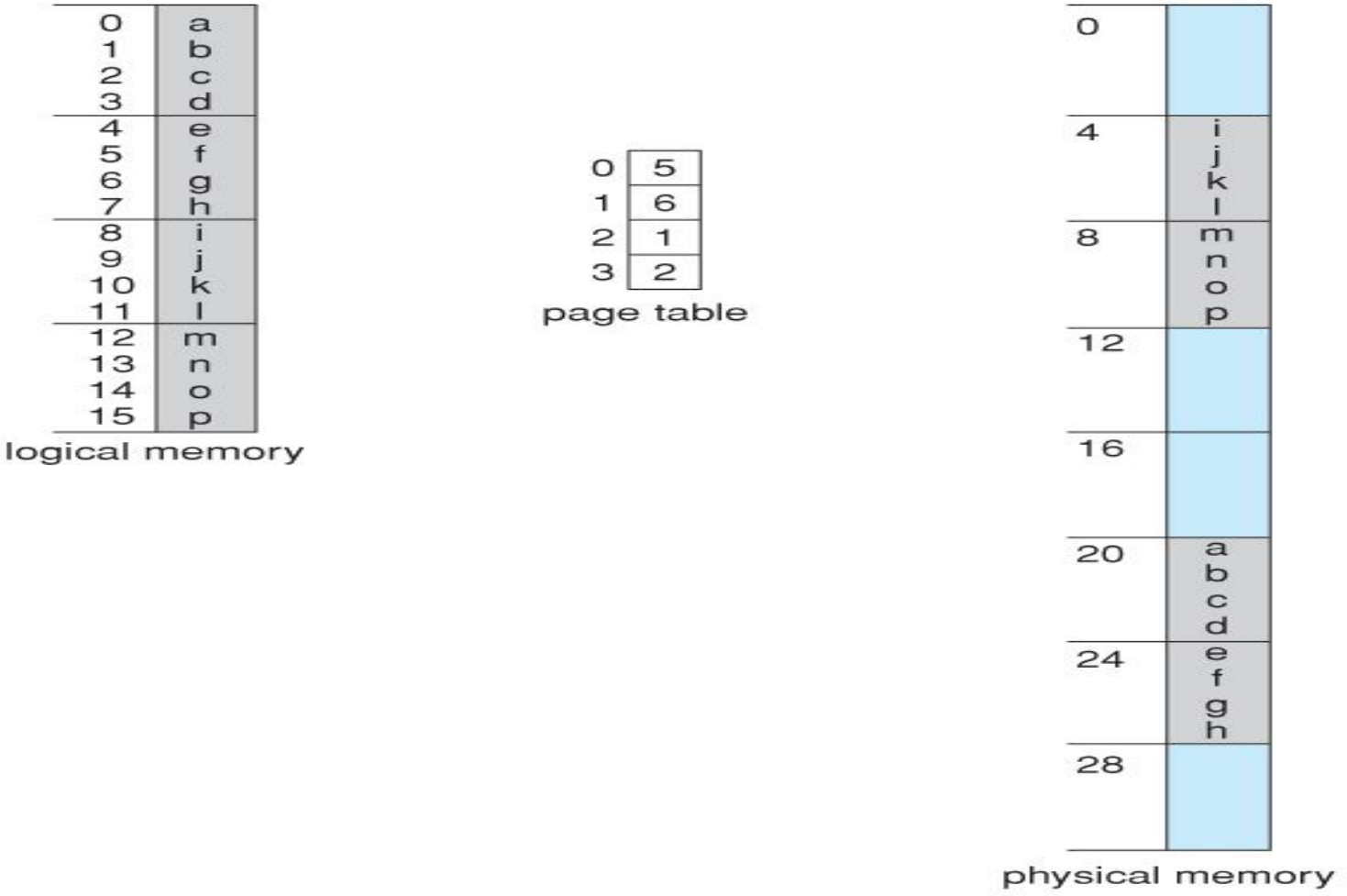
Address Translation Scheme

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

Address Translation Architecture



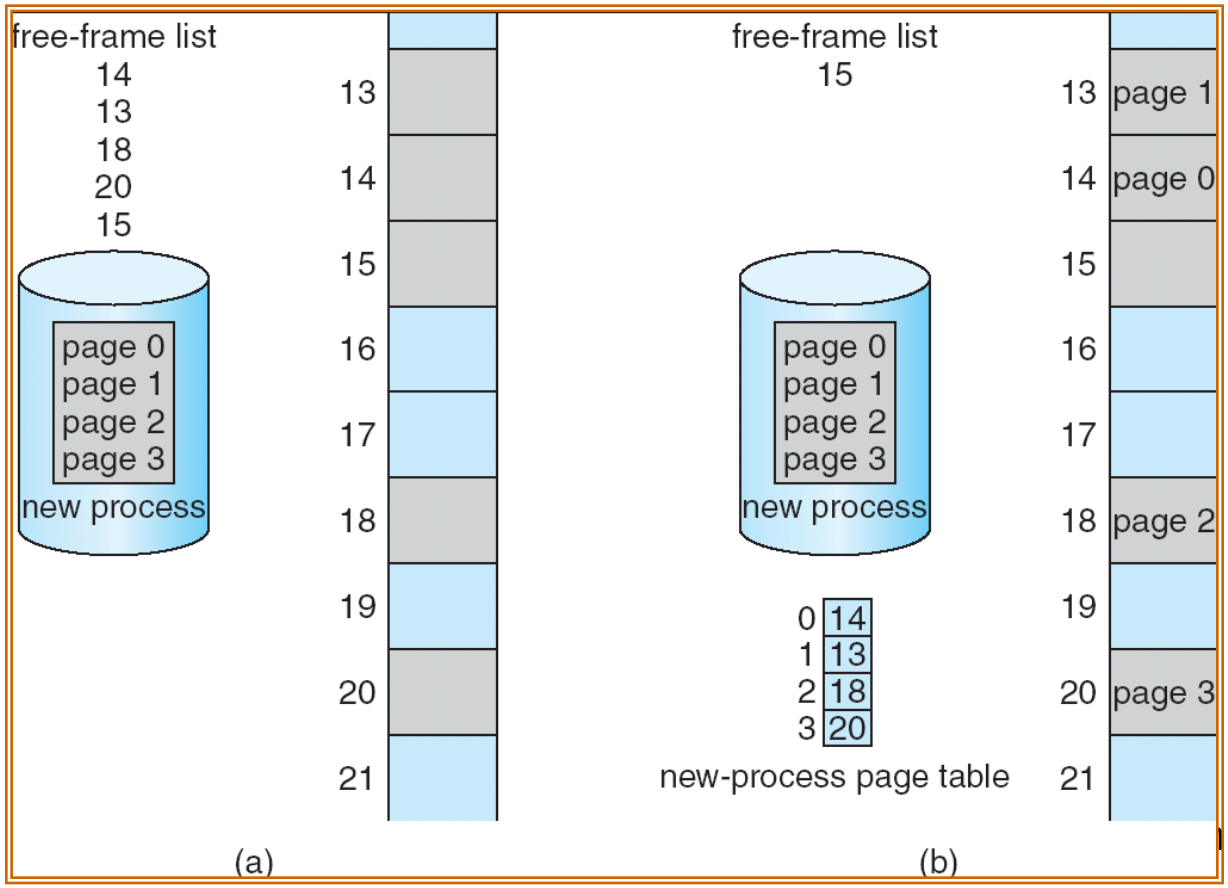
Paging Example for a 32-byte memory with 4-byte pages



Free Frames

- The operating system is managing physical memory, it must be aware of the allocation details of physical memory—which frames are allocated, which frames are available, how many total frames there are, so on.
- This information is generally kept in a data structure called a **frame table.**
- The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.
- The free frames before allocation and after allocation is shown in the below figure:

Free Frames


free-frame list
14
13
18
20
15
page 0
page 1
page 2
page 3
new process
13
14
15
16
17
18
19
20
21
(a)
free-frame list
15
page 0
page 1
page 2
page 3
new process
0 14
1 13
2 18
3 20
new-process page table
13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21
(b)

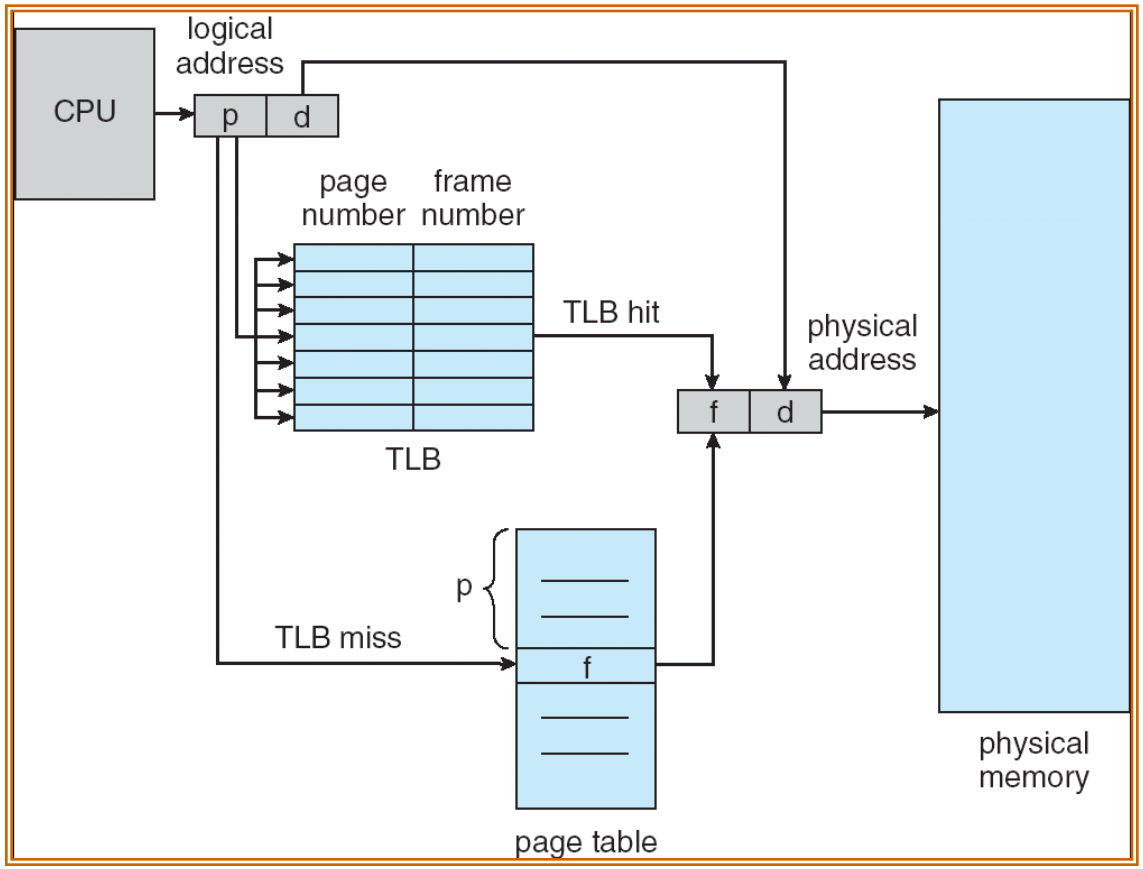
Implementation of Page Table

- Page table is kept in main memory
- *Page-table base register (*PTBR) points to the page table
- *Page-table length register* (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

- If a particular page is present in TLB Hit, then it will find the particular frame number and access the frame in the memory.
- If a particular page is not present in TLB Hit, then it will go to main memory and check the page table where it contains whole bunch of pages and frames and then it will access that particular frame in the memory.

- If a page is present in TLB Cache, then it is much faster similarly if a page is not present in TLB Cache then it becomes slower.

Paging Hardware With TLB

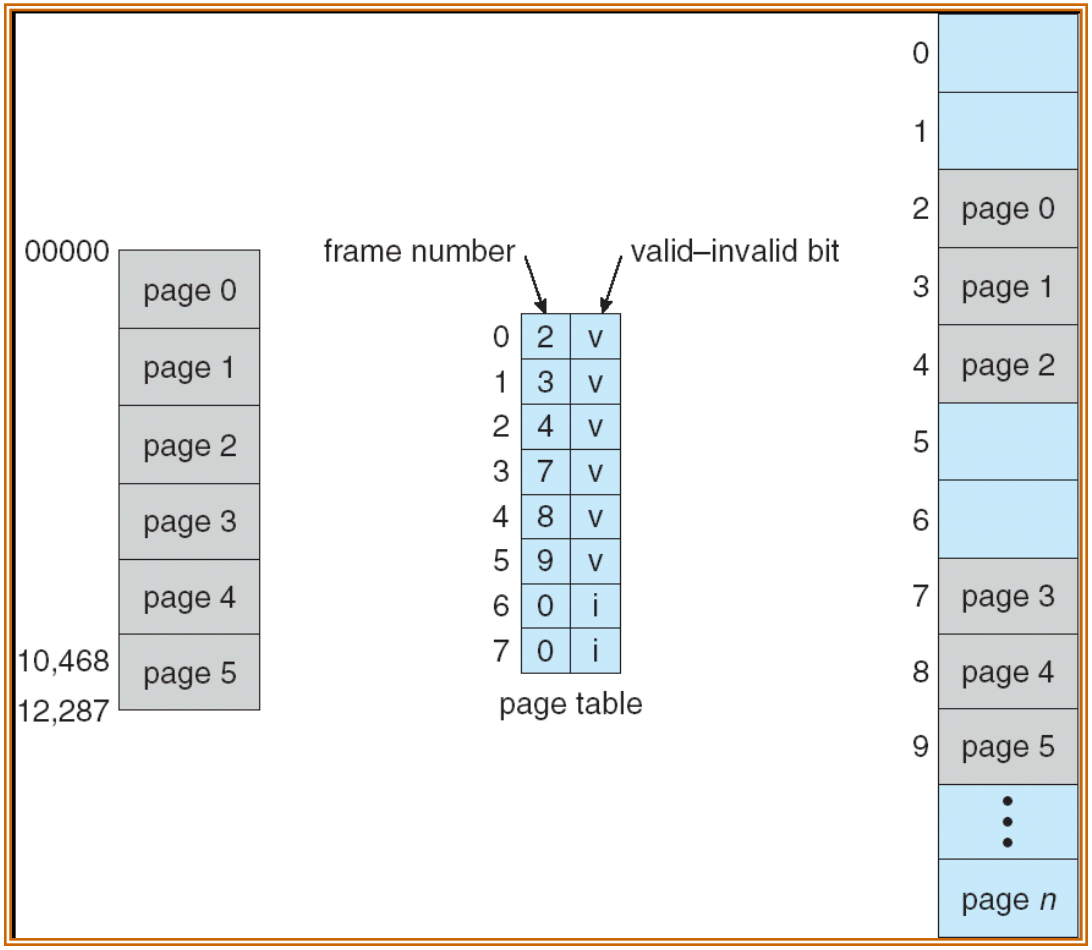


Memory Protection

- Memory protection implemented by associating protection bit with each frame

- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process's logical address space, and is thus a legal(valid) page
  - "invalid" indicates that the page is not in the process's logical address space

Valid (v) or Invalid (i) Bit In a Page Table

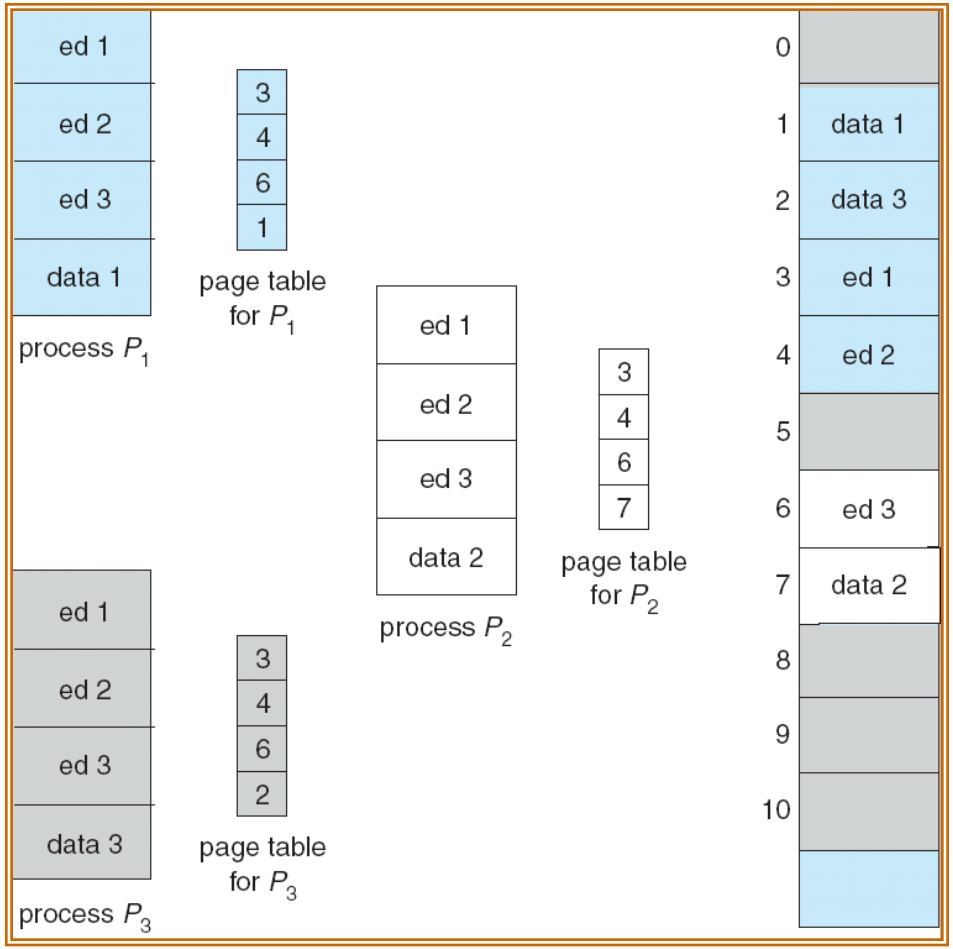


Shared Pages

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

- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



Page Table Structure

- Page Tables are the data structures used to store the mappings of logical address present in process memory to physical address present in RAM.

Different approaches for reducing page search in TLB are as follows:

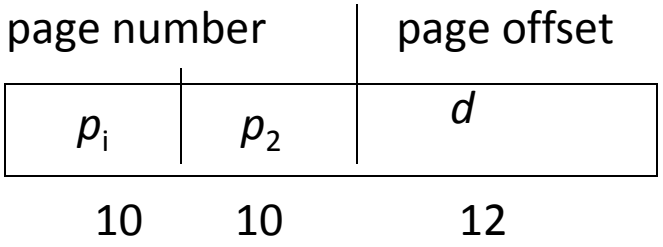
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables.
- Each logical address is divided into array page table , sets offset to various level of pages & final offset is specific to memory access.
- A simple technique is a two-level page table

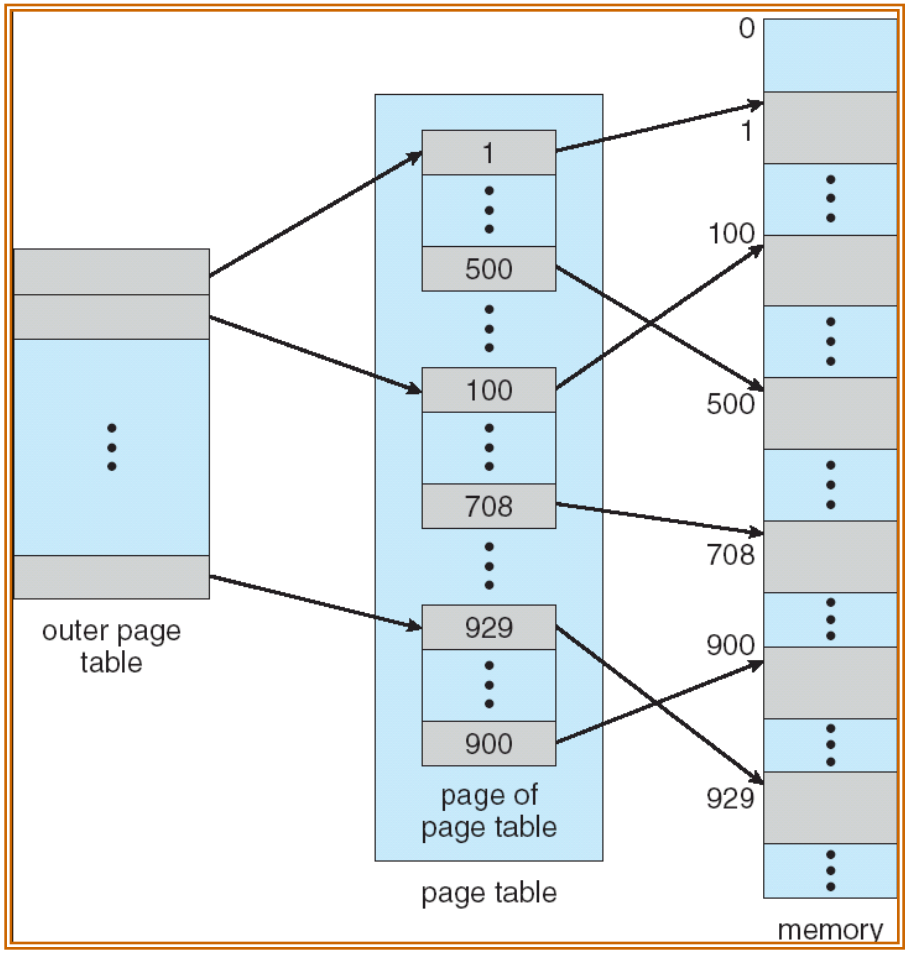
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:



where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Two-Level Page-Table Scheme



Hashed Page Tables

- A Common approach for address spaces larger than 32 bits, then it has to store in a hashed page table.

- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location i.e., virtual page number will be generated in the hash function.

- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

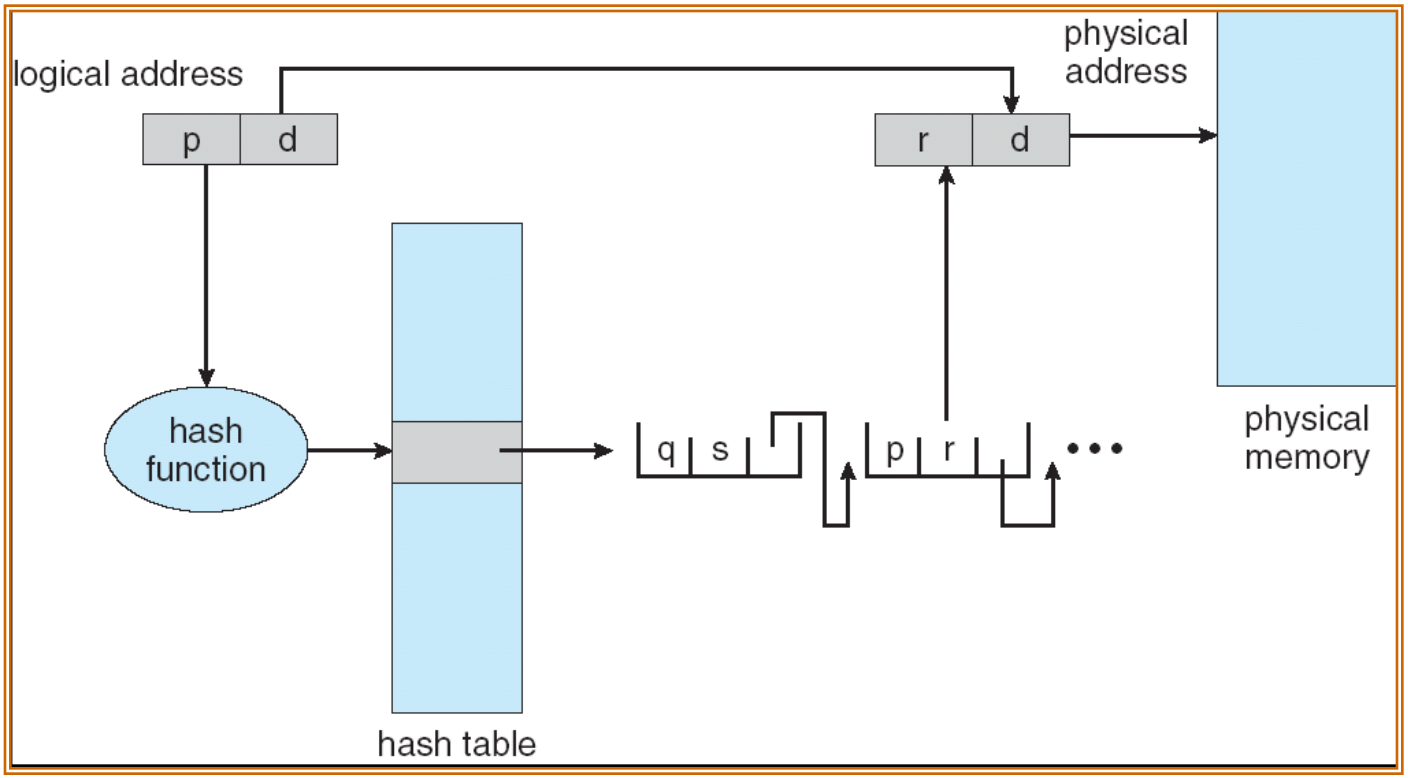
Hashed Page Tables(Cont)

The virtual page number will be ultimately pointing to the particular physical address on the physical memory.

Each element consists of three fields:

i. The Virtual Page number,
ii. The value of the mapped page frame,
iii. A pointer to the next element in the linked list.

Hashed Page Table

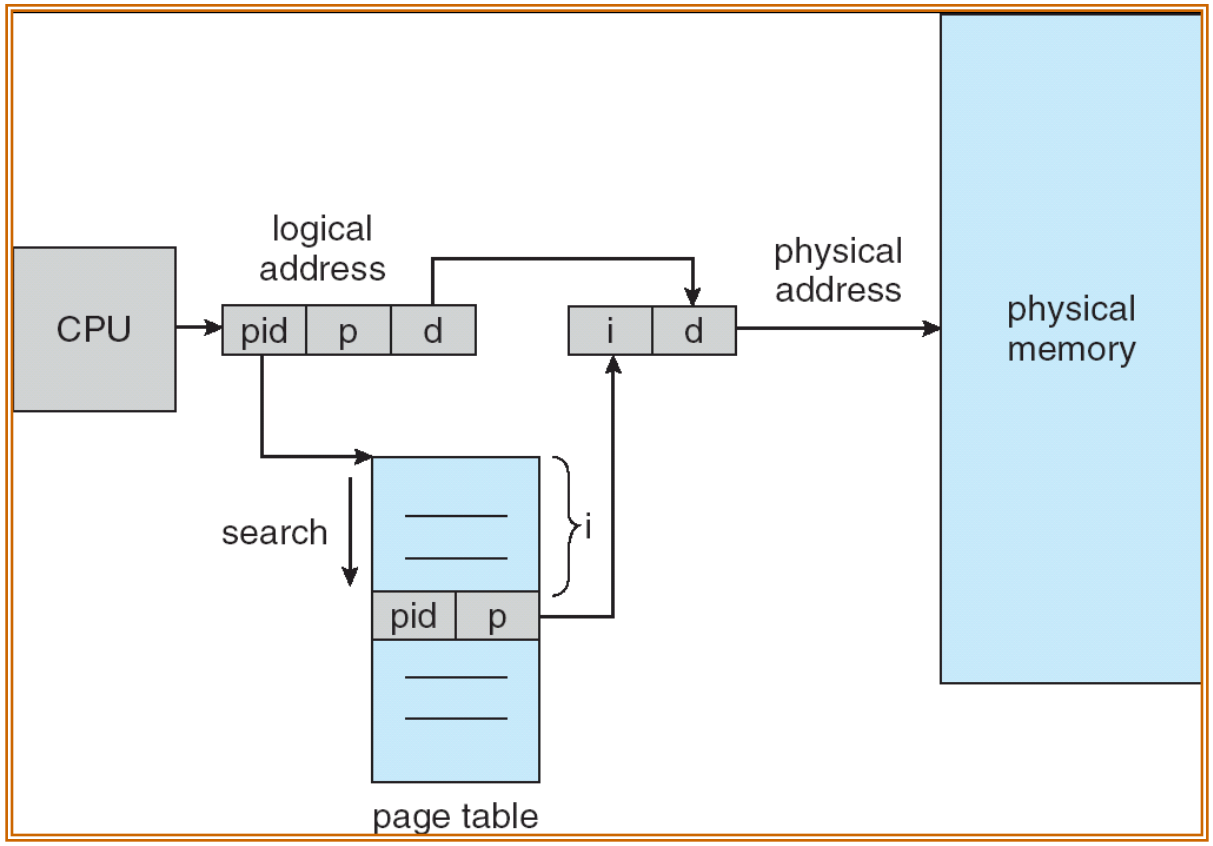


Inverted Page Tables

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
- Each inverted page table entry is a pair of

  <Process-id,page-number,offset>

Inverted Page Table Architecture



Segmentation

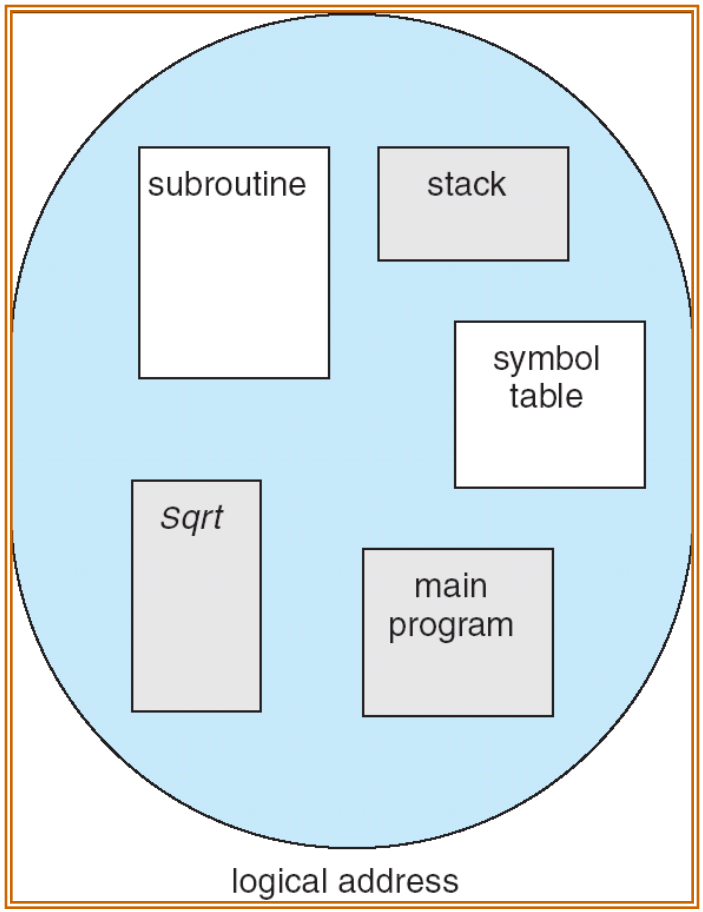
- A program is divided into parts known as segments.
- Paging is fixed in size where as segment is variable in size.
- Paging is physical entity where as segmentation is a logical entity.
- Like paging, logical address converts into two parts:

i. Segment-number

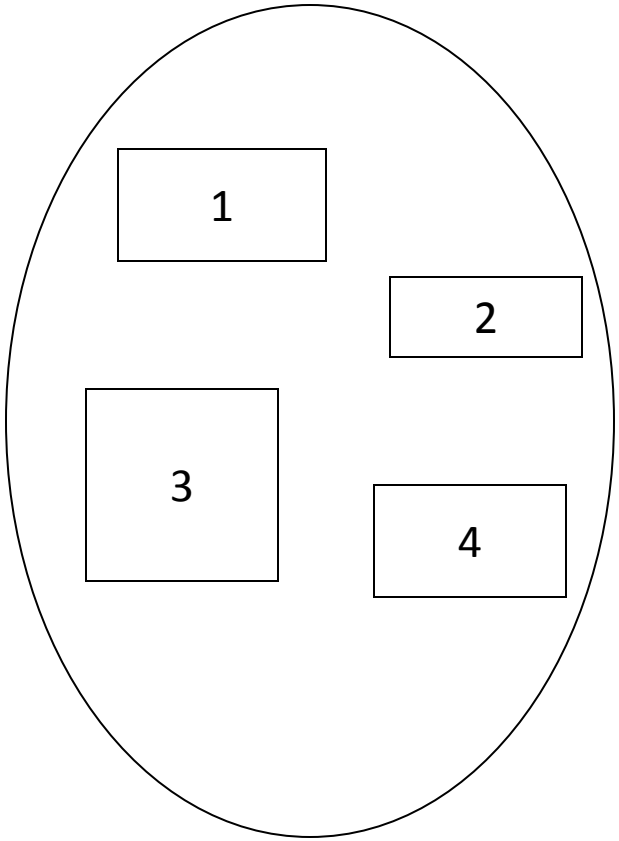
ii. Offset

- The logically stored segments should be physically stored in the main memory that the CPU may access.
- Paging is invisible but segmentation is visible to the user
- Minimum size of a segment is equal to page size.
- Maximum size of a segment will be virtual address space i.e., it is the limit of CPU addressability.(Each segment consist of many pages)

User's View of a Program



Logical View of Segmentation



user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

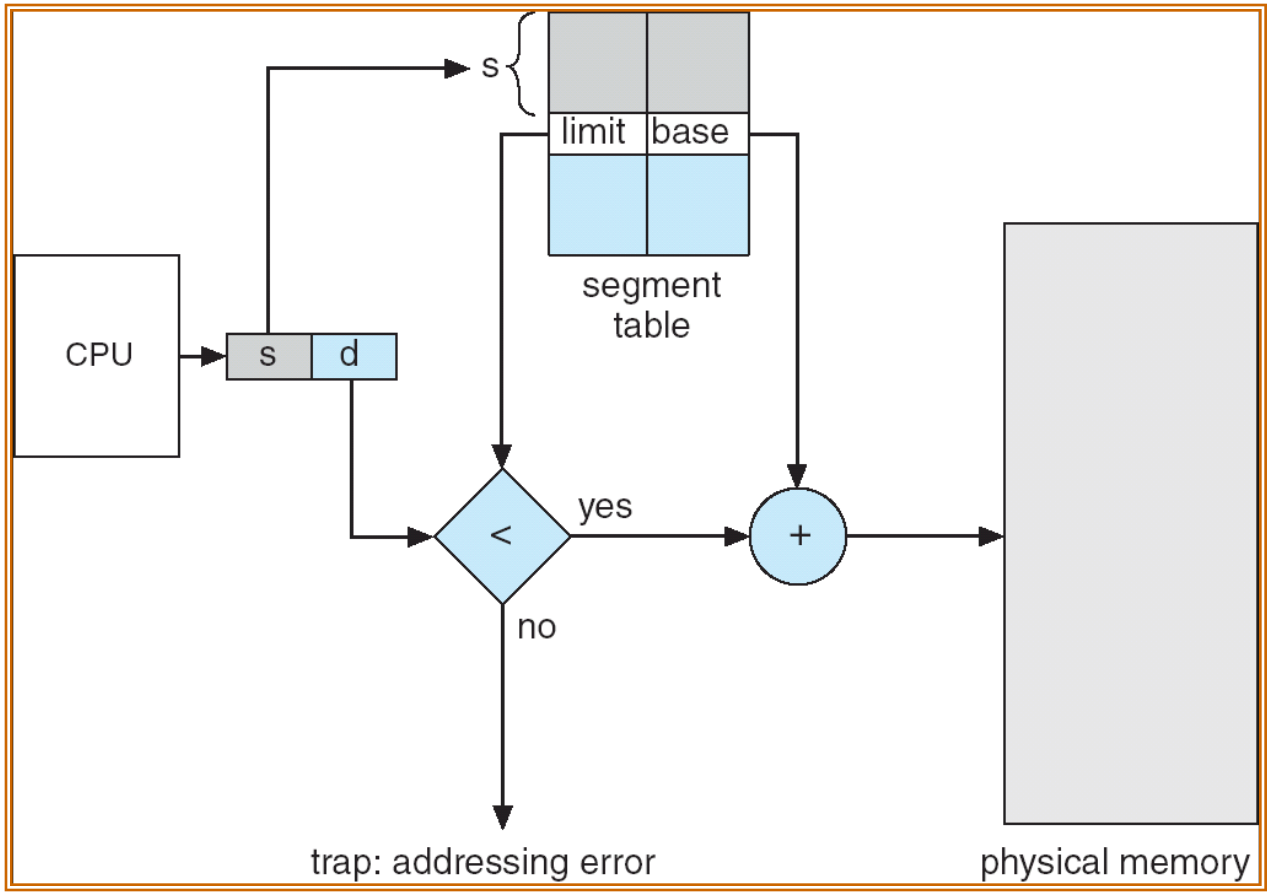
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - *limit* – specifies the length of the segment
- *Segment-table base register (STBR)* points to the segment table's location in memory
- *Segment-table length register (STLR)* indicates number of segments used by a program;

  segment number *s* is legal if *s* < STLR

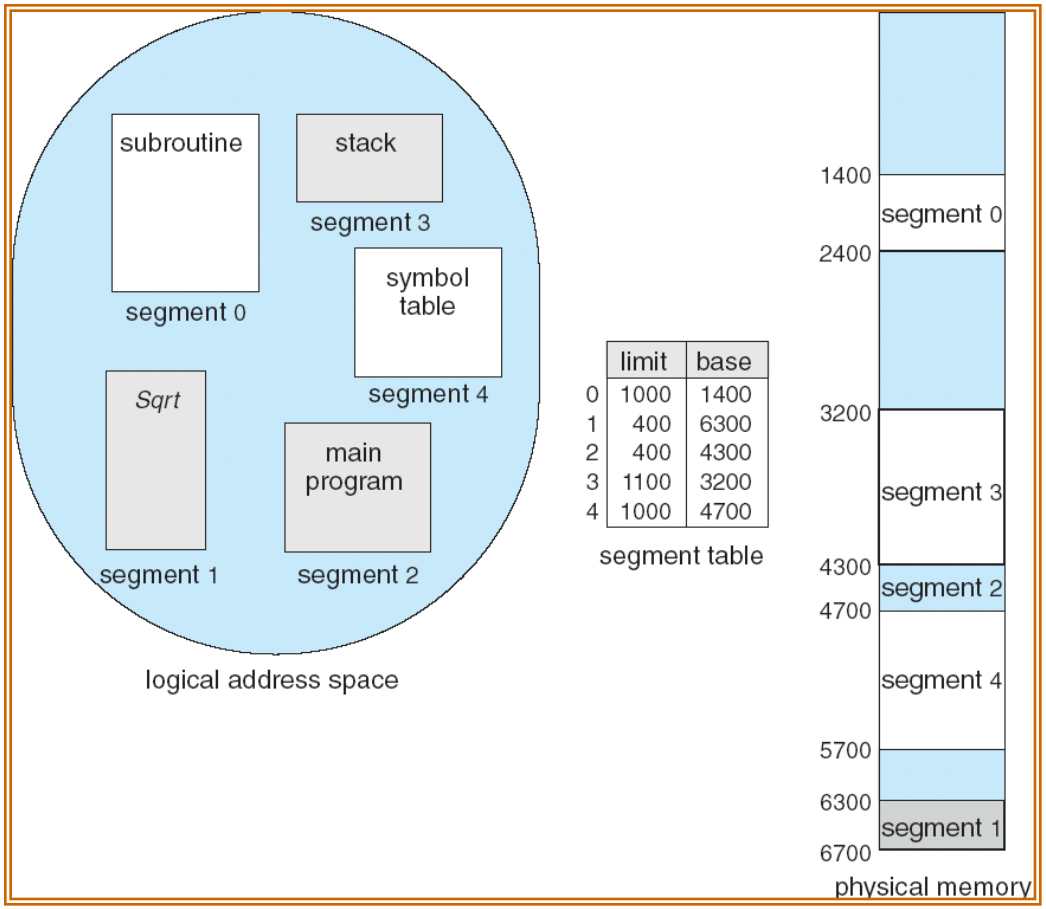
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - validation bit = 0 $\Rightarrow$ illegal segment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation hardware



Example of Segmentation



Segmentation(Cont)

- For example the CPU generates <segmentation no, offset>

The steps followed for segmentation :

Step 1:Base address is fetched

Step 2: It must check the offset and length of the segment

Step 3:If step 2 is satisfied it will take the base address and add to the offset of the given.

Step 4:Then it will go to the address and will fetch that address.

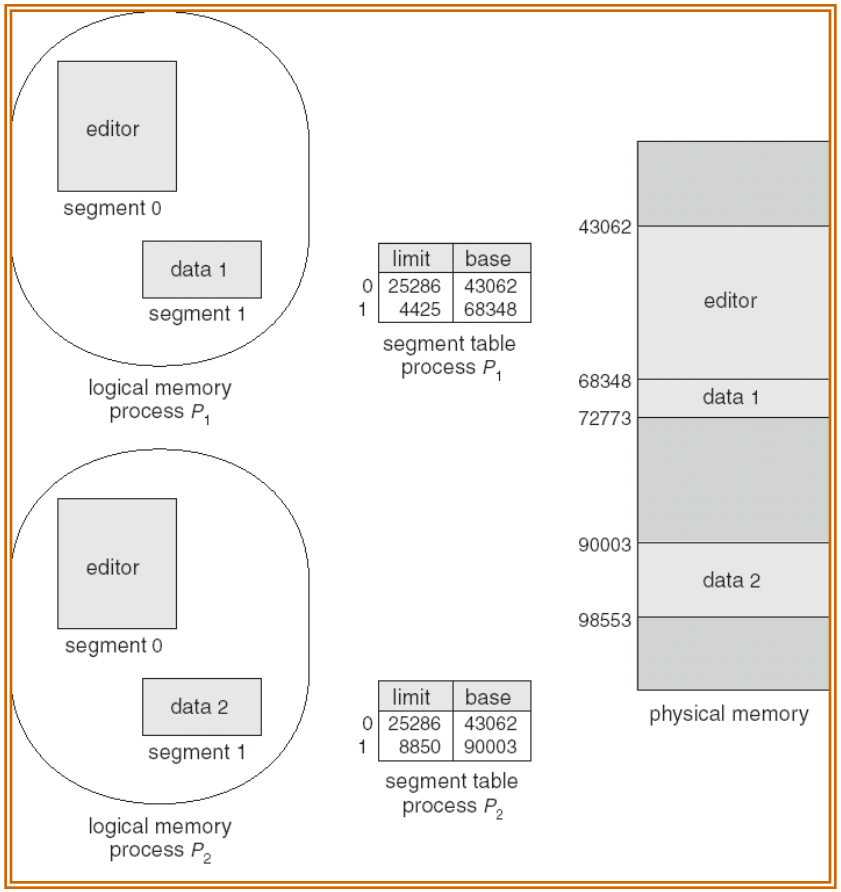
Segmentation(Cont)

- Segment in virtual address space/logical address space where as paging is in physical address space
- Holes: Portions of memory space which are not utilized
- Previously in paging, we saw how to translate logical address to physical address. Now, we will see how to translate virtual address to physical address.

Segmentation Architecture (Cont.)

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

Sharing of Segments



Virtual Memory

- Background
- Demand paging
  - Performance of demand paging
- Page Replacement
  - Page Replacement Algorithms
- Thrashing

Need for Virtual Memory

- Virtual Memory
  - Separation of user logical memory from physical memory.
  - Only *PART* of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Need to allow pages to be swapped in and out.
- Virtual Memory can be implemented via
  - Paging
  - Segmentation

Virtual Memory(Cont)

- There are many benefits to execute a program that is only partially in memory:

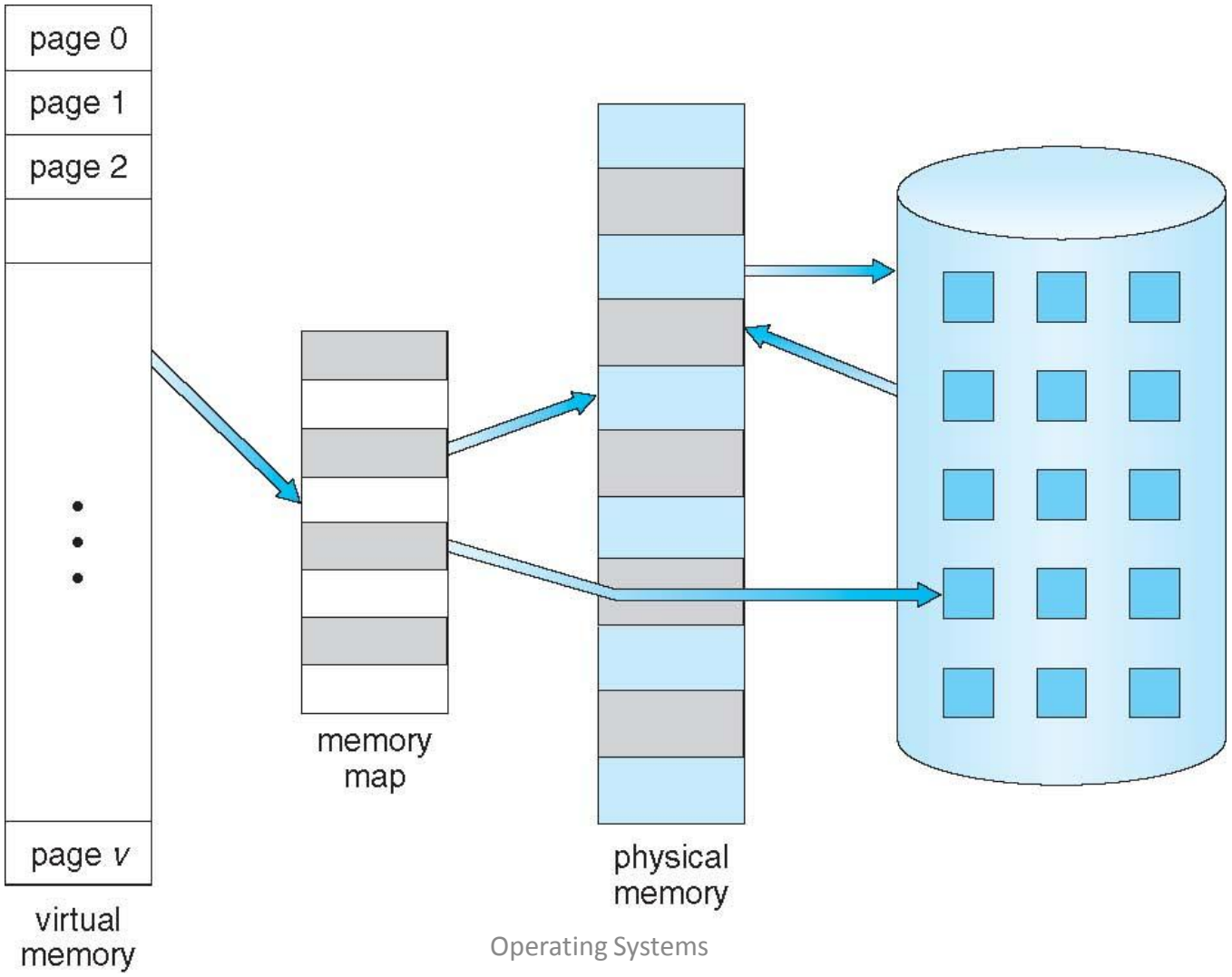
i) A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space simplifying the program task.

ii) Because each user program could take less physical memory so that more programs can run at the same time, So that CPU utilization and throughput will increase.

iii) Less I/O would be needed to load or swap user programs into memory so each user program would run fast.

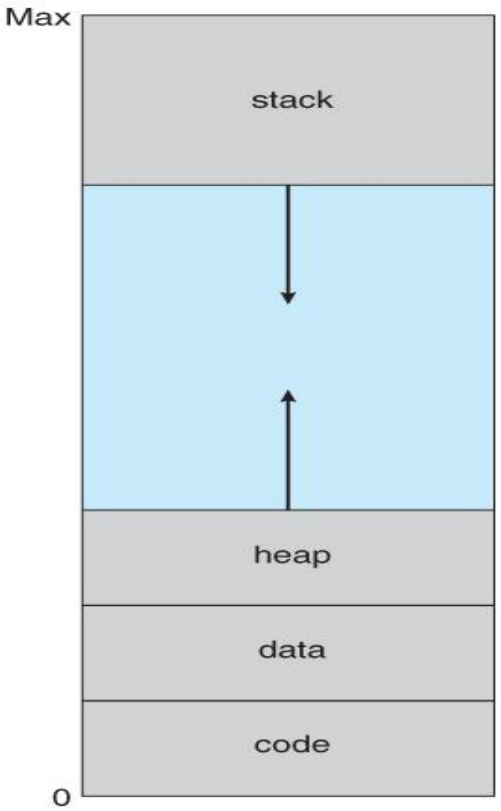
Thus, running a program that is not entirely in memory would benefit both user and the system.

Virtual Memory That is Larger Than Physical Memory

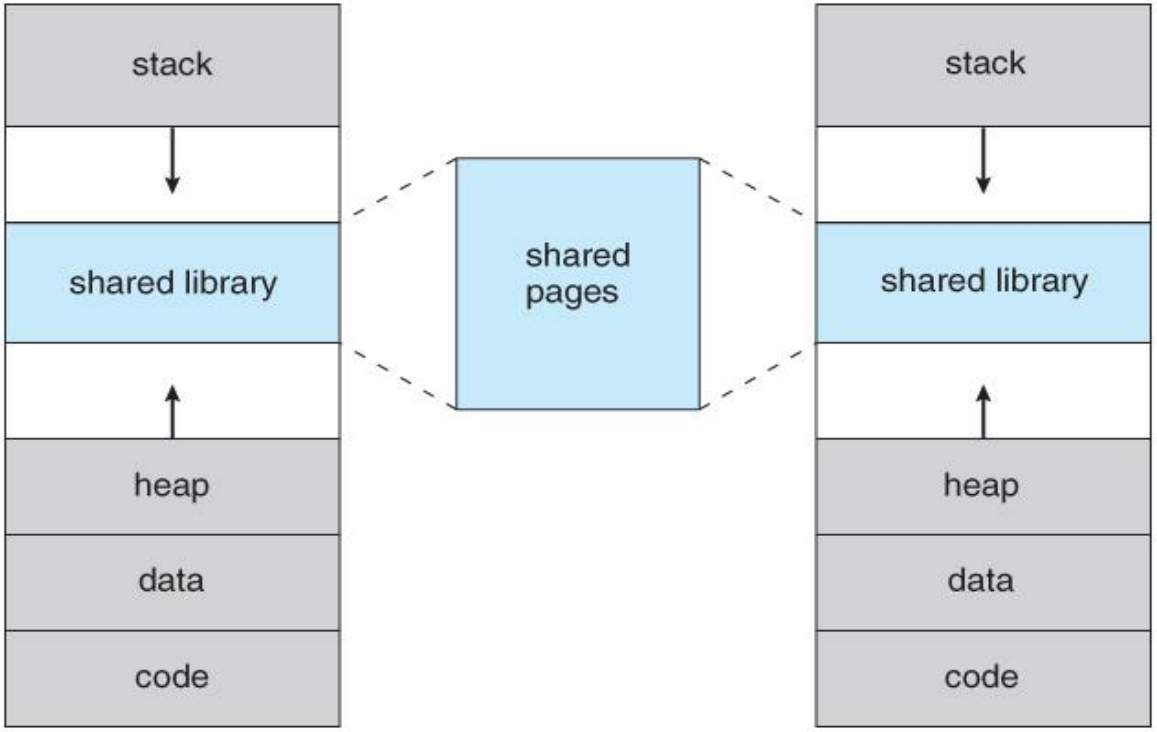


Virtual address space

- The virtual address space of a process refers to the logical(or virtual) view of how a process is stored in memory.



Shared Library using virtual memory

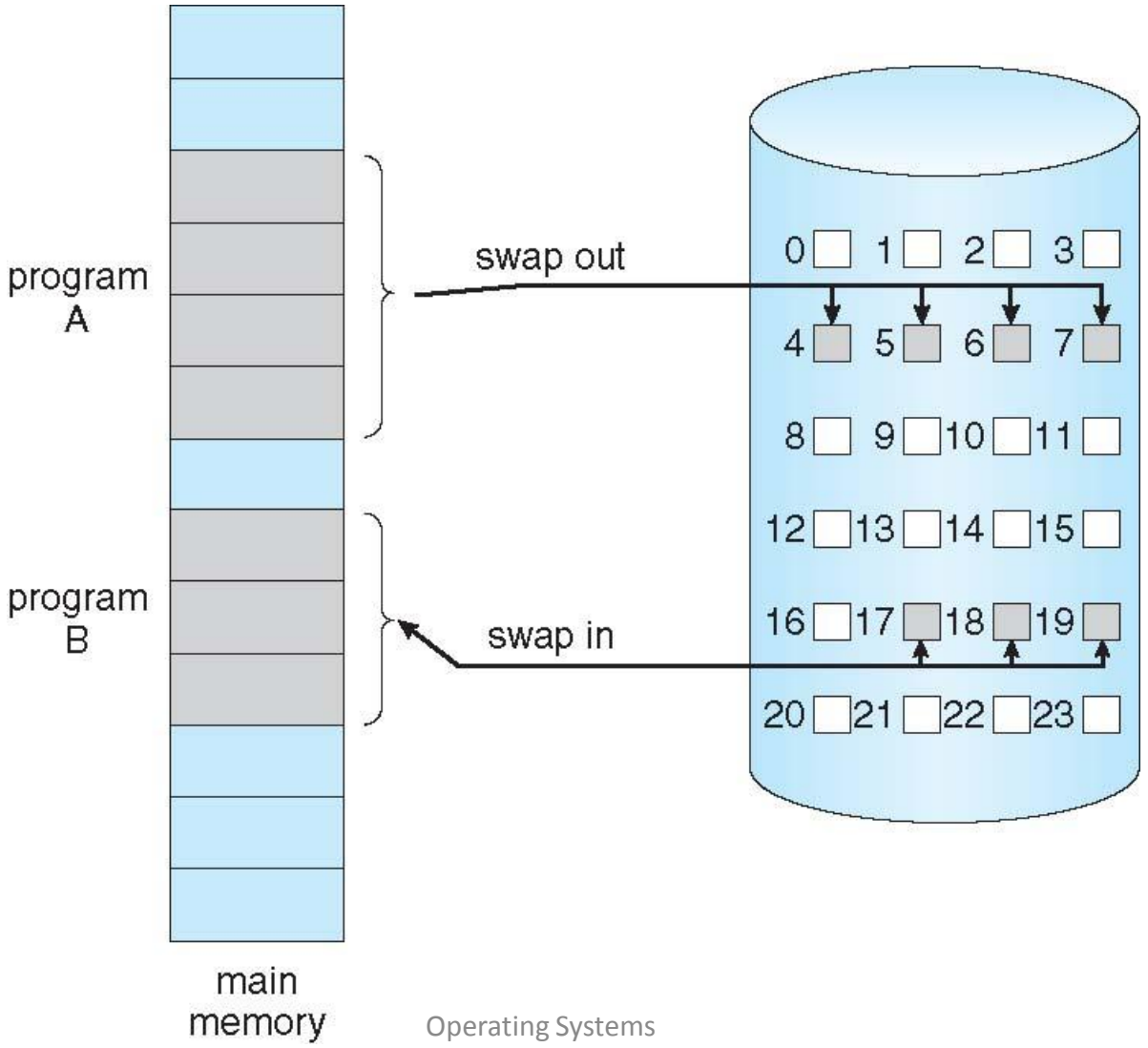


Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory.
  - The pages which are in need by the CPU for execution is known as

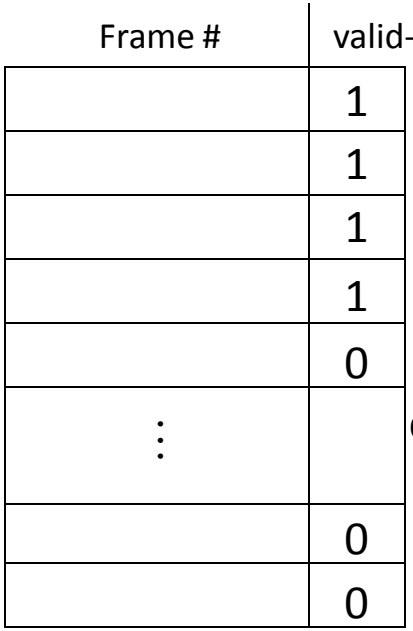
    Demand paging
  - The demand paging system is similar to a paging system with swapping where a process reside in secondary memory when we want to execute a process we swap it into memory.
  - **A lazy swapper** will never swap a page into memory unless that page will be needed.

Transfer of a Paged Memory to Contiguous Disk Space



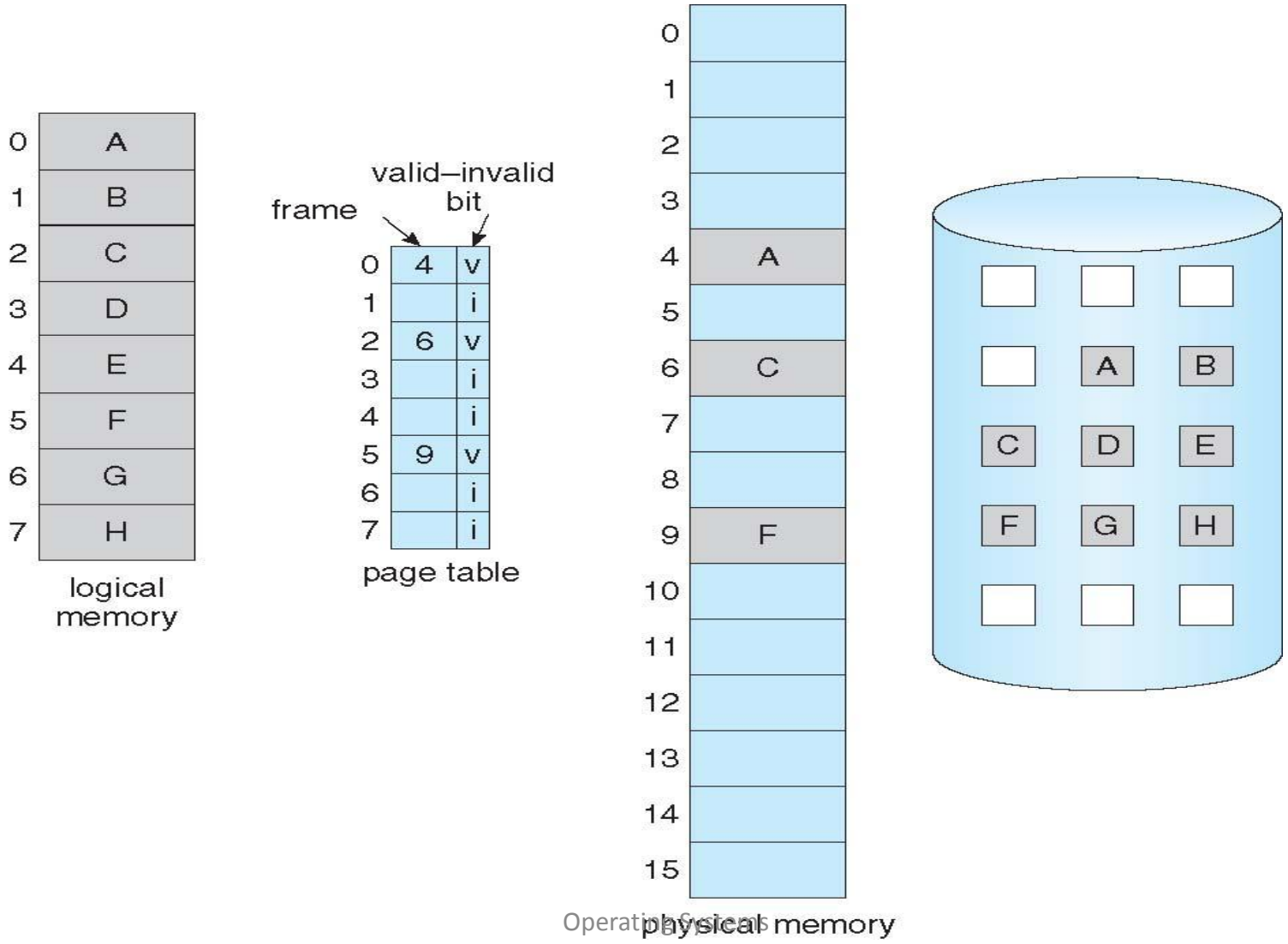
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated ($1 \Rightarrow$ in-memory, $0 \Rightarrow$ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.



- During address tra d bit in page table entry is $0 \Rightarrow$ page fault.

Page Table When Some Pages Are Not in Main Memory



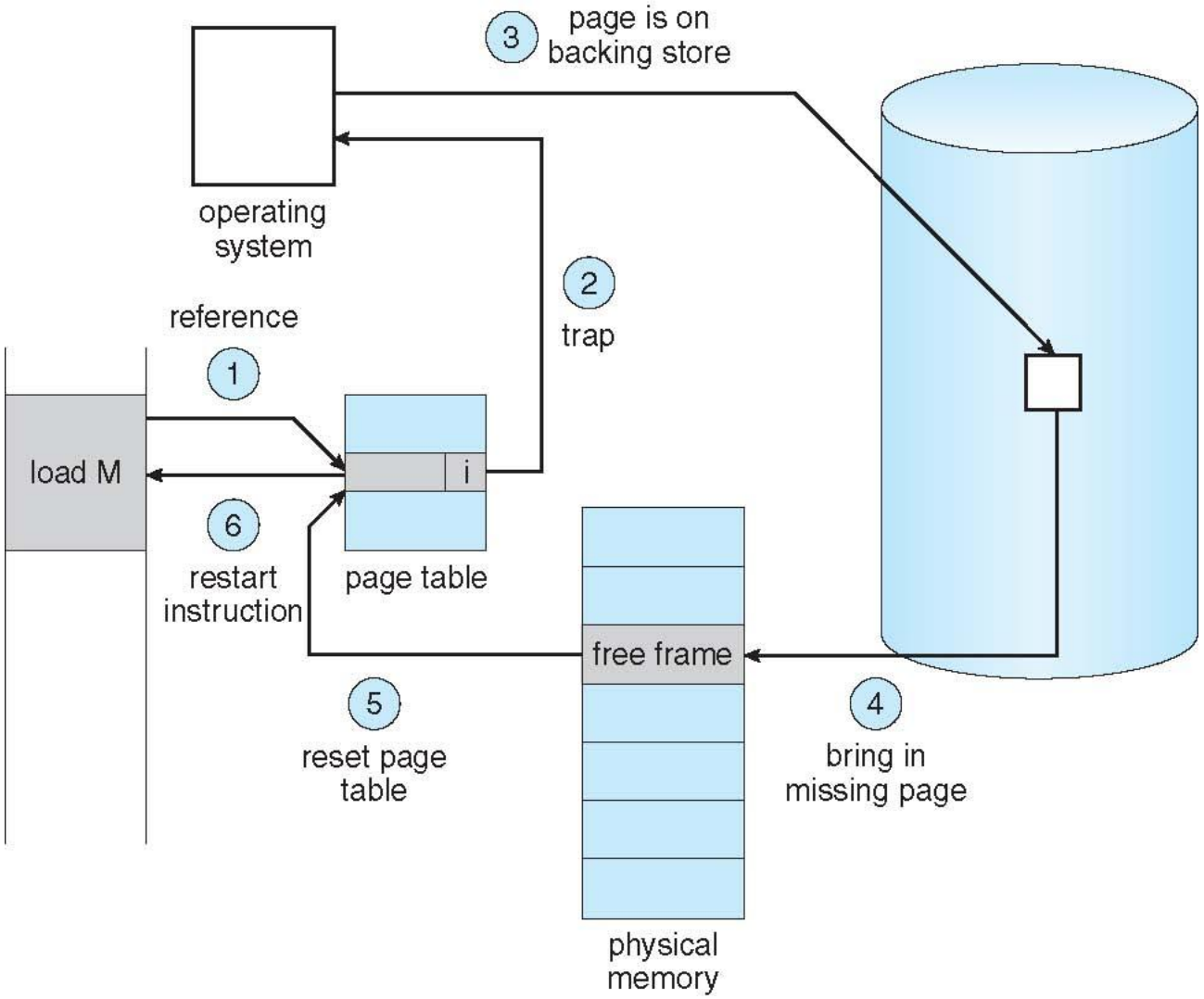
Page Fault

•If the process tries to access a page that was not bought into memory ,access to a page marked invalid causes a "Page Fault".
•The paging hardware , in translating the address through the page table, will notice that the invalid bit is set , causes the trap to the operating system .
•This trap is the result of the operating system's failure to bring the desired page into memory.

The procedure for handling this page fault is:

1.We check an internal table for this process to determine whether the reference was a valid or an invalid memory access.
2.If the reference was invalid , we terminate the process. If it was valid , but we have not yet brought in that page , we now page in it.
3.We find a free frame.(by taking one from the free frame list).
4.We schedule a disk operation to read the desired page into the newly allocated frame.
5. When disk read is completed , we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6.We restart the instruction that was interrupted by the trap . The process can now access the page as though it had always been in memory.

Steps in Handling a Page Fault



Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

  EAT = (1 – p) x memory access + p (page fault time)

To calculate the effective access time , we must know how much time is needed to service a page fault.

1. Trap the Operating System.
2. Save the user registers & process states.
3. Determine that interrupt was a page fault.
4. Check that the page reference was legal and determine the location of the page on the disk.
5. Issue a read from the disk to a free frame.

a) wait in a queue for the device until the read request is serviced.

b) wait for the device seek and /or latency time.

c) Begin the transfer of the page to free frame.

6. While waiting , allocate the CPU to some other user.
7. Receive an interrupt from the disk I/O subsystem.
8. Save the registers and process for the other users.(if step 6 is executed)
9. Determine the interrupt was from the disk.
10. Correct the page table and other tables to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user registers , process state , and new page table , and then resume the interrupted instruction.

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Swap Page Time = 10 msec = 10,000 msec

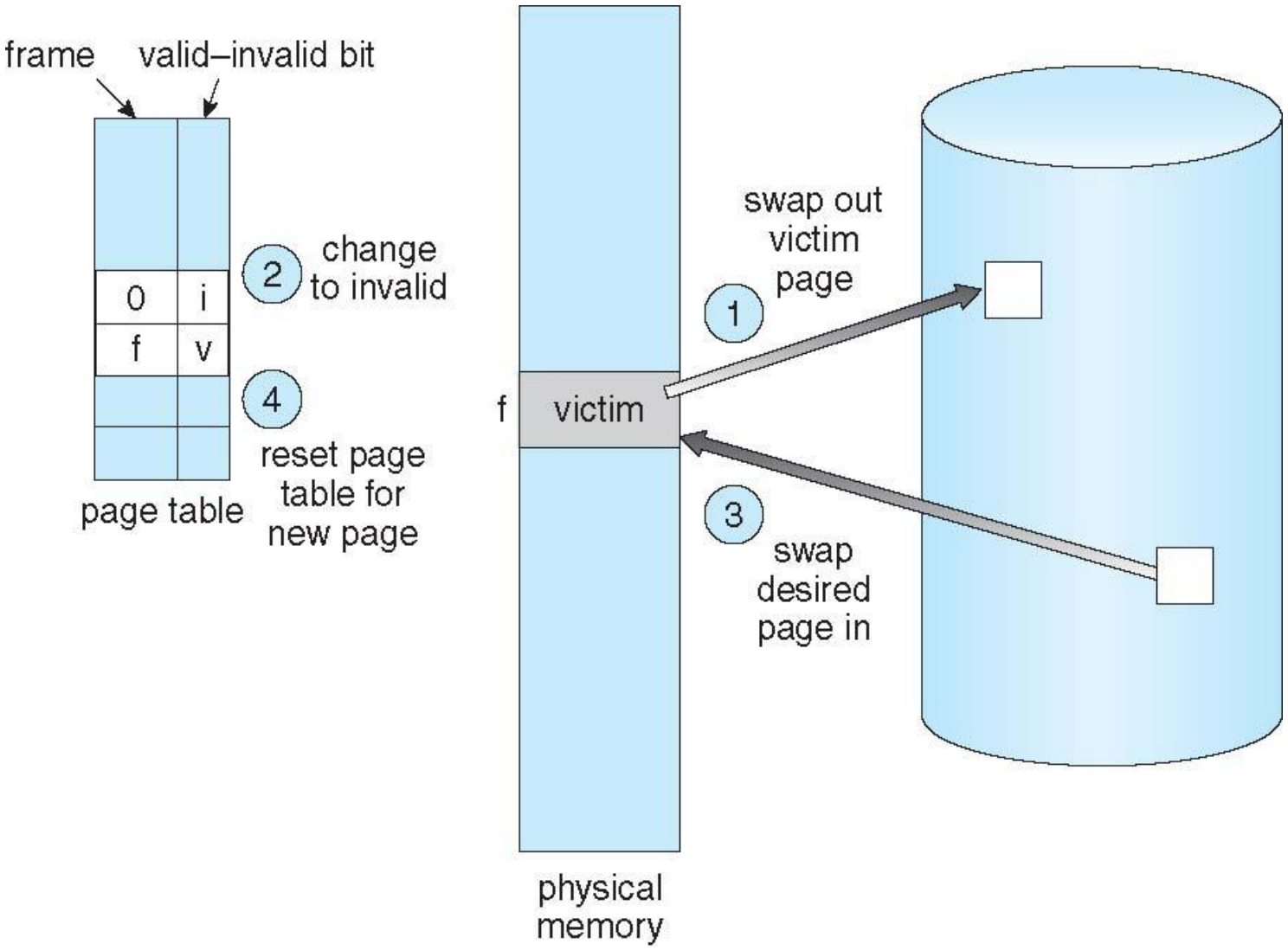
$$EAT = (1 - p) \times 1 + p\ (15000)$$

$$1 + 15000P \quad \text{(in msec)}$$

Basic Page Replacement

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

Page Replacement

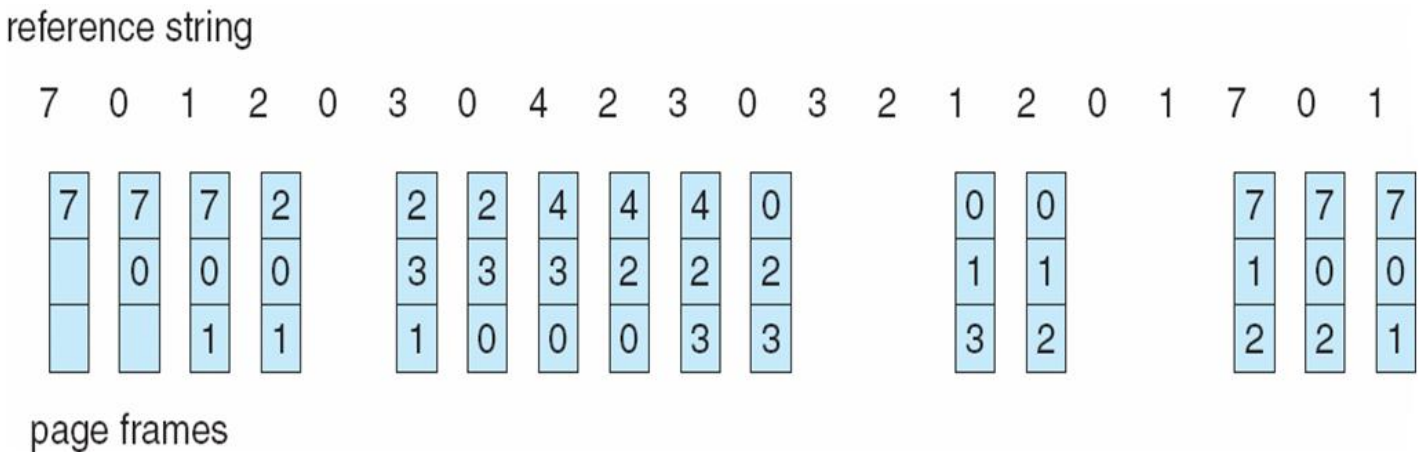


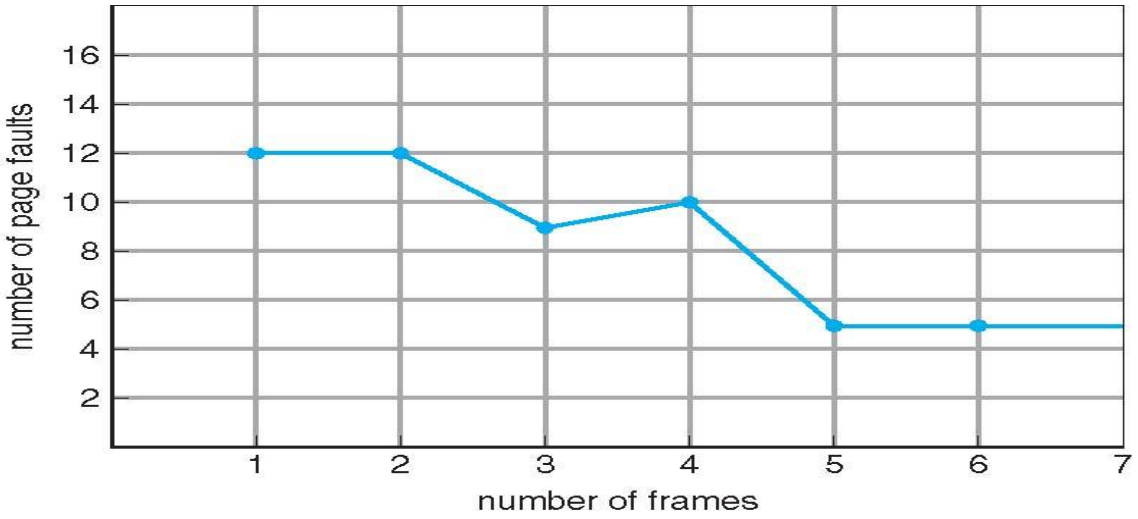
Page and Frame Replacement Algorithms

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

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

FIFO Page Replacement

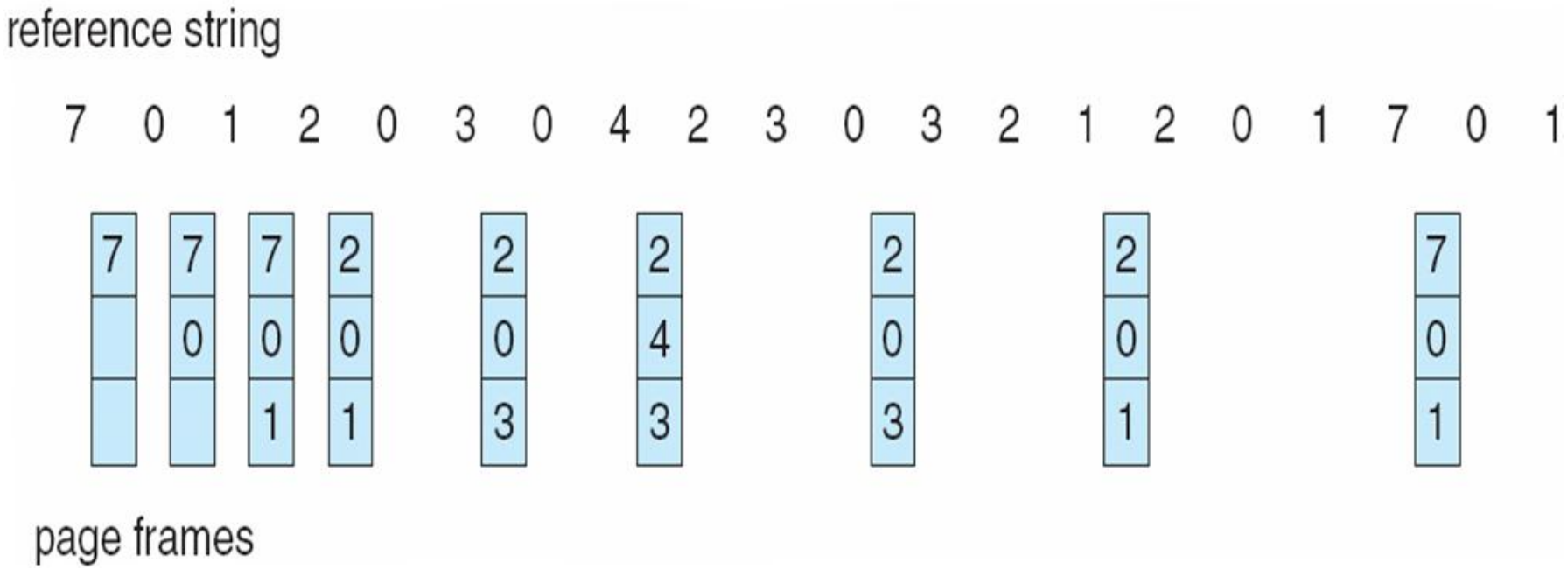




FIFO Illustrating Belady's Anomaly

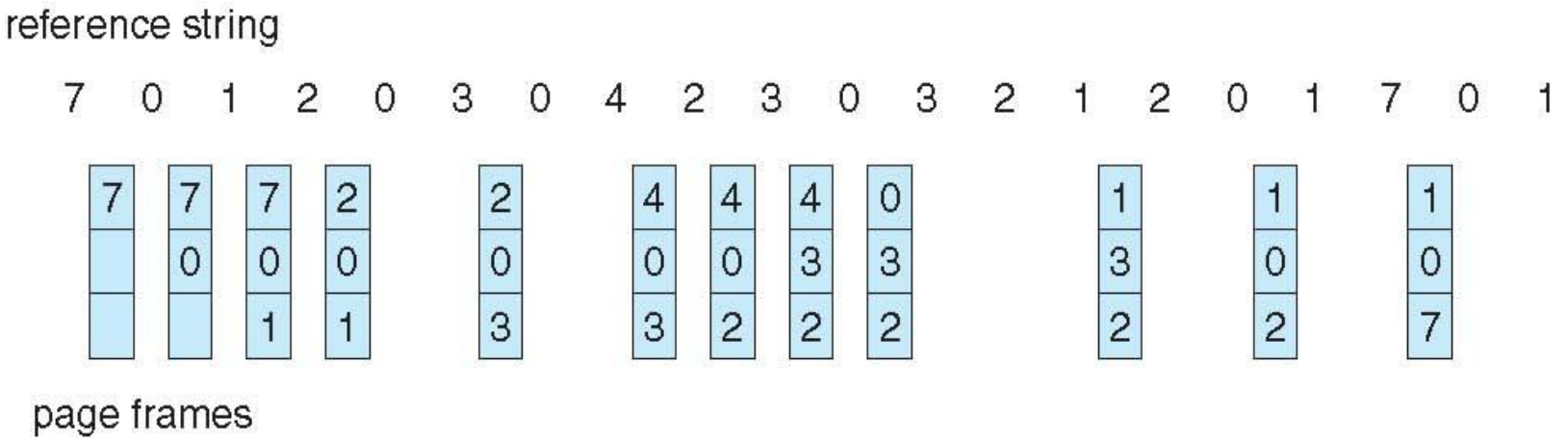
Optimal Algorithm

Replace page that will not be used for longest period of time
– optimal for the example on the next slide



Least Recently Used (LRU) Algorithm

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



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

LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially -= 0
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit.
  - Clock replacement.
  - If page to be replaced (in clock order) has reference bit = 1. then:
    - set reference bit 0.
    - leave page in memory.
    - replace next page (in clock order), subject to same rules.

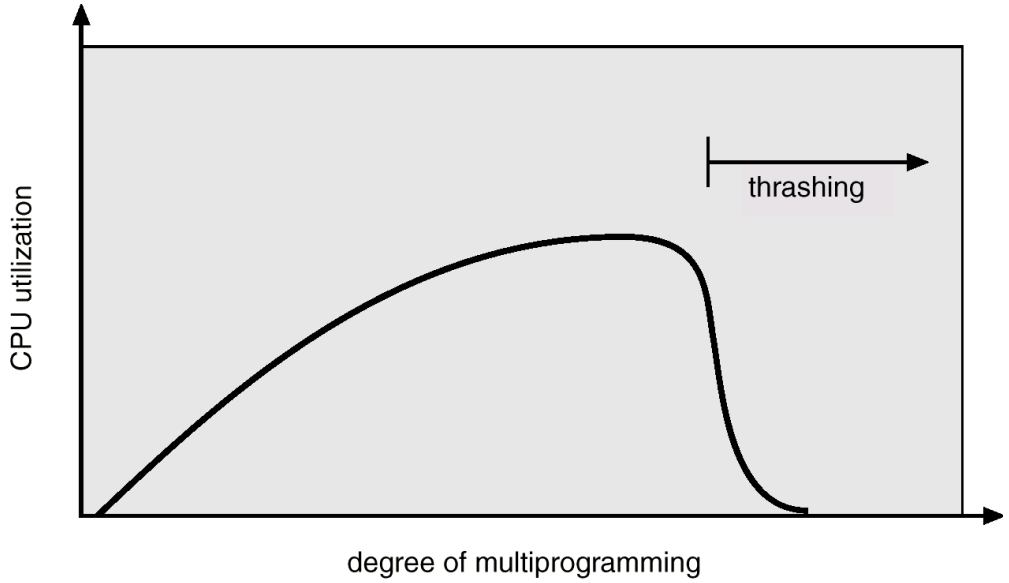
Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - operating system thinks that it needs to increase the degree of multiprogramming.
  - another process added to the system.
- Thrashing ≡ a process is busy swapping pages in and out.

Thrashing Diagram



- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

Deadlocks

Deadlocks

- The Deadlock Problem
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Chapter Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks

- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

Deadlocks

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters into a waiting state. Sometimes, a waiting process is never again able to change its state, because the resources it has requested are held by some other waiting processes. This situation is called as **Deadlock.**

System Model

- Resource types R_1, R_2, . . ., R_m

  CPU cycles, memory space, I/O devices.

- Each process utilizes a resource as follows:
  - request : The process requests the resource. If the request cannot be granted immediately, for e.g, if the resource is being used by another process, then the requesting process must wait until it can acquire the resource.
  - Use: The process can operate on the resource for e.g, if the resource is a printer, the process can print on the printer.
  - Release: The process release the resources.

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process in a system which cannot be shared simultaneously by more than one process.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** once a resource is allocated to a particular process that resource cannot be preempted, the process has to voluntarily should release the resource and complete the task.
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.

- If it overcome all the four conditions in a system, then we have overcome Deadlock
- If at least one condition is overcome, then we can say sometimes as Deadlock has been overcome.

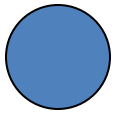
Resource-Allocation Graph

The Deadlock can easily be identified in the graphical format.

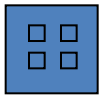
- A graph is represented as a pair i.e., G=(V,E)
- V is partitioned into two types of nodes:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the active processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- E is partitioned into two types:
- request edge – directed edge $P_1 \rightarrow R_j$
- allocation edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

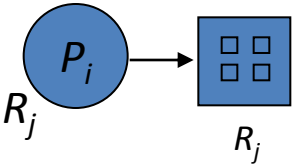
- Process



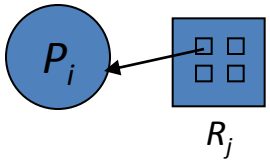
- Resource Type with 4 instances



- P_i requests instance of R_j



- P_i is holding an instance of R_j



Example of a Resource Allocation Graph

- The Resource-allocation graph shown in the graph depicts the following situation

The sets P,R,E.

P={P1,P2,P3}

R={R1,R2,R3,R4}

E={P1->R1 , P2->R3 , R1->P2 , R2->P2 , R2->P1 , R3->P3 }

Resources instances:

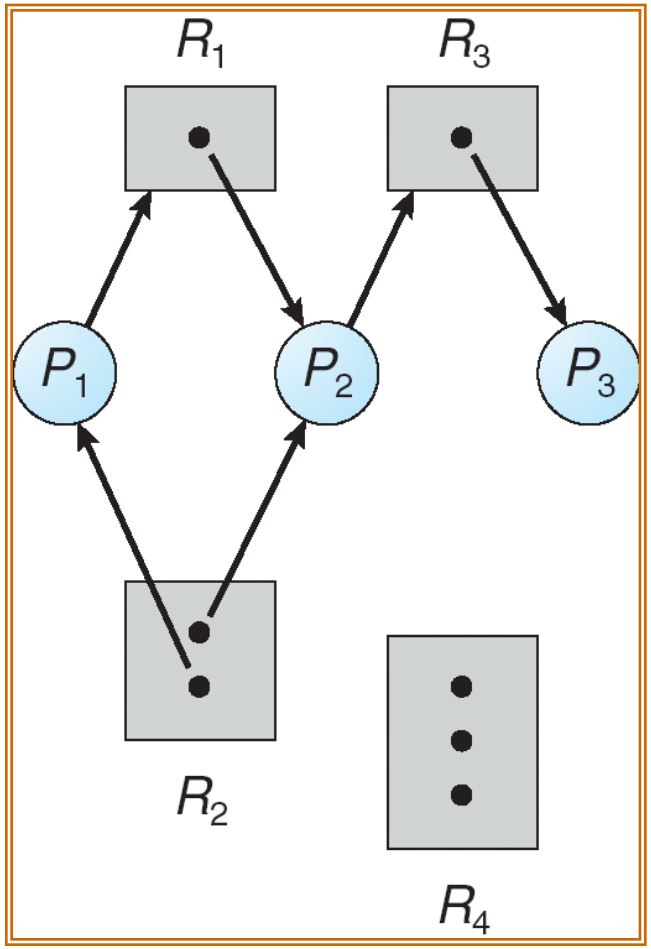
One instance of resource type R1

Two instances of resource type R2

One instance of resource type R3

Three instances of resource type R4.

Example of a Resource Allocation Graph



Example of a Resource Allocation Graph

- In the above RAG, in resource type R1, we have got single instance of resource.
- In resource type R2, we have got two instances of resources.
- In resource type R3, we have got single instance of a resource.
- In resource type R4, we have got three instance of a resources.

So, at a particular type of an instance, a resource type R3 is allocated to process P3 and P3is not waiting for any other resource.

Similarly, process P2 is holding an instance of resource type R1 and even it is holding an instance of a resource type R2, and it is waiting for an instance of resource type R3.

Similarly, the process P1 is holding an instance of resource type R2, and it is waiting for a resource type R1.

---------→So, at this particular situation, it doesnot lead to Deadlock.

Example of Resource Allocation Graph With A Deadlock

- The Resource-allocation graph shown in the graph depicts the following situation

The sets P,R,E.

P={P1,P2,P3}

R={R1,R2,R3,R4}

E={P1->R1 , P2->R3 , R1->P2 , R2->P2 , R2->P1 , R3->P3 , P3->R2}

Resources instances:

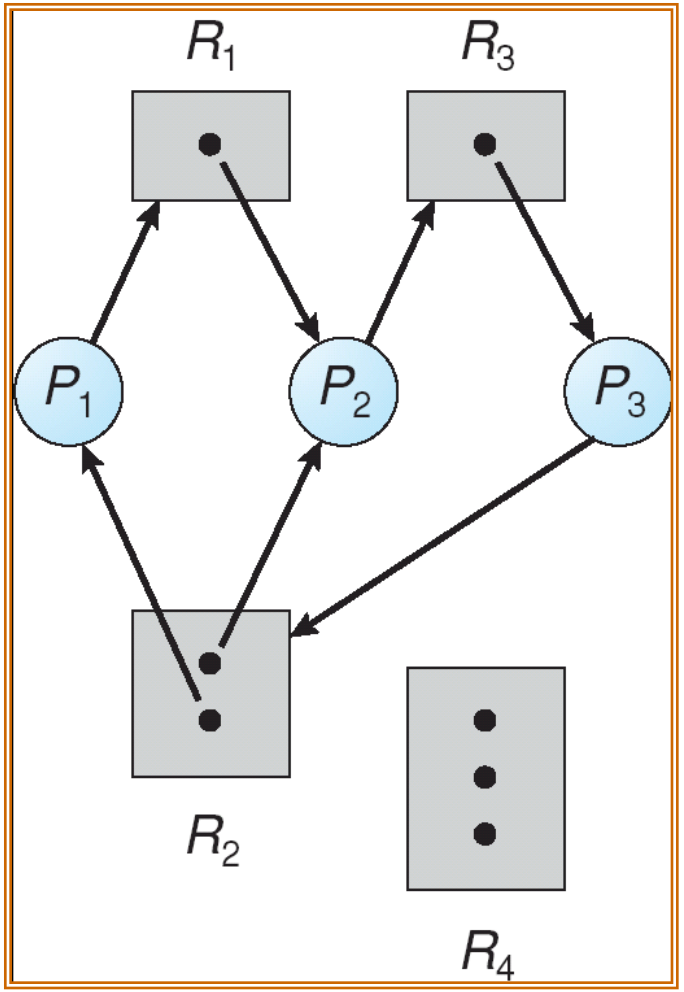
One instance of resource type R1

Two instances of resource type R2

One instance of resource type R3

Three instances of resource type R4.

Resource Allocation Graph With A Deadlock



Example of Resource Allocation Graph With A Cycle But No Deadlock

- The Resource-allocation graph shown in the graph depicts the following situation

The sets P,R,E.

P={P1,P2,P3,P4}

R={R1,R2}

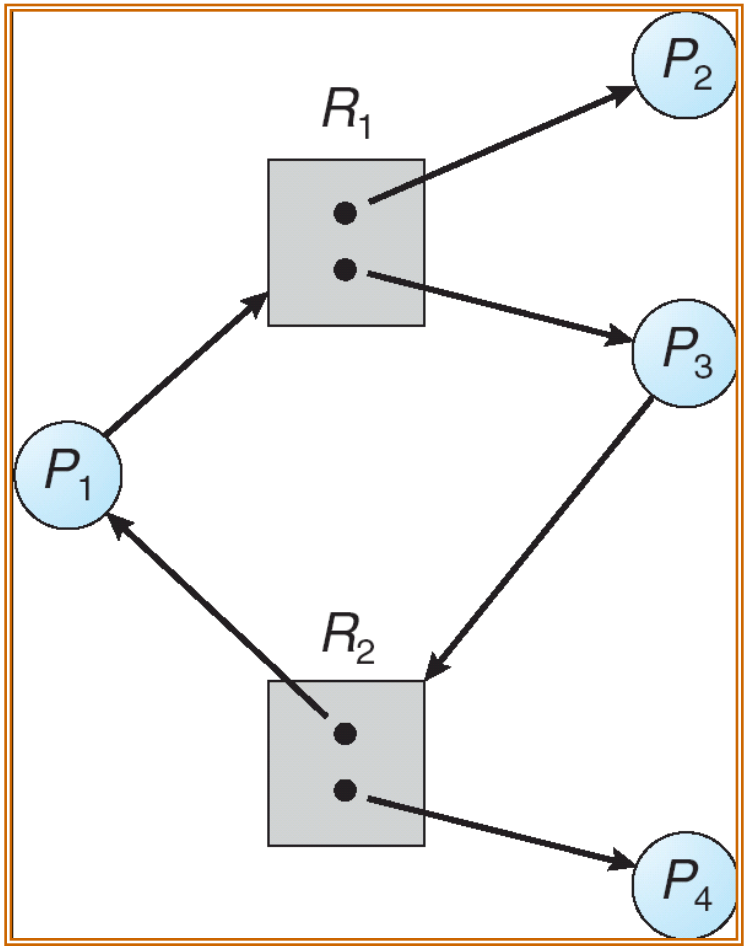
E={P1->R1 , R1->P2 , R1->P3 , R2->P1 , R2->P4 , P3->R2}

Resources instances:

Two instances of resource type R1

Two instances of resource type R2

Resource Allocation Graph With A Cycle But No Deadlock



- In the above RAG, the cycle from P1→R1→P3→R2→P1, but even in the presence of this cycle, it does not lead to a deadlock, because the process P2 and P4 though they r holding the resource types R1 and R2 but they are not waiting for any other resource, they will complete their operation and will release the resources, and whenever they release the resources that resources are allocated to the requesting processes.
- So, this is an example of RAG even when there is a cycle, then it does not lead to deadlock.
- However, if every resource type contains a single instance of the resource in that case existence of a cycle means there is always a deadlock.
- In the above RAG, if we remove a single instance from R1 and R2, then there is an existence of deadlock

Basic Facts

- If graph contains no cycles $\Rightarrow$ there can be a deadlock or may not be a deadlock.
- If graph contains a cycle $\Rightarrow$ there can be a deadlock or may not be a deadlock.
- If every resource type contains a multiple instance of a resource, then the existence of cycle means then there may be a deadlock or may not be a deadlock.
- If every resource type contains a single instance of a resource, then the existence of cycle means then there is a deadlock.

Methods for Handling Deadlocks

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

Different kinds of strategies

The three different kinds of strategies to overcome deadlock:

1. Deadlock Prevention: Preventing the deadlock from the occurrence.
2. Deadlock Avoidance: Avoiding the deadlock from the occurrence.
3. Deadlock detection and Recovery: In this, we allow the system to allocate the resources whenever it is available and periodically it check whether there has been a deadlock or not.

   If there is a deadlock, we will try to break down it.

   This will also lead insufficient utilization of resources.

Deadlock Prevention

There will be a restrictions on the processes, the way they request for the resources

- In this, we will try to break either,Mutual Exclusion or hold and wait, No Preemption, Circular wait.
- **Mutual Exclusion** – Cannot be broken.i.e., always there is some resource in a system which is always mutual exclusive.

- **Hold and Wait** – can be avoided i.e., it means whenever a process is requesting an additional resource it is holding some other resource with it.
  - For each process it must require all the resources before its execution.
  - A Process must request a resource if it doesn't have any.

  Disadvantages

  1. It uses poor utilization of resources but it solves the deadlock problem.
  2. Starvation is possible.

Deadlock Prevention (Cont.)

- **No Preemption** –can be broken
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - Preempted resources are added to the list of resources for which the process is waiting.

- **Circular Wait** – can be broken if we allow the processes to request a resource in a particular order.

We define a mapping function:

F:R→N i.e., for every resource type I have a corresponding integer number.

A process while holding a resource type Ri put a request for a resource type Rj only when F(Rj)>F(Ri).

Before putting a request for Rj, it should release the resource type Ri. So, if this condition cannot satisfy then the process cannot put the request for a resource type Rj.

Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

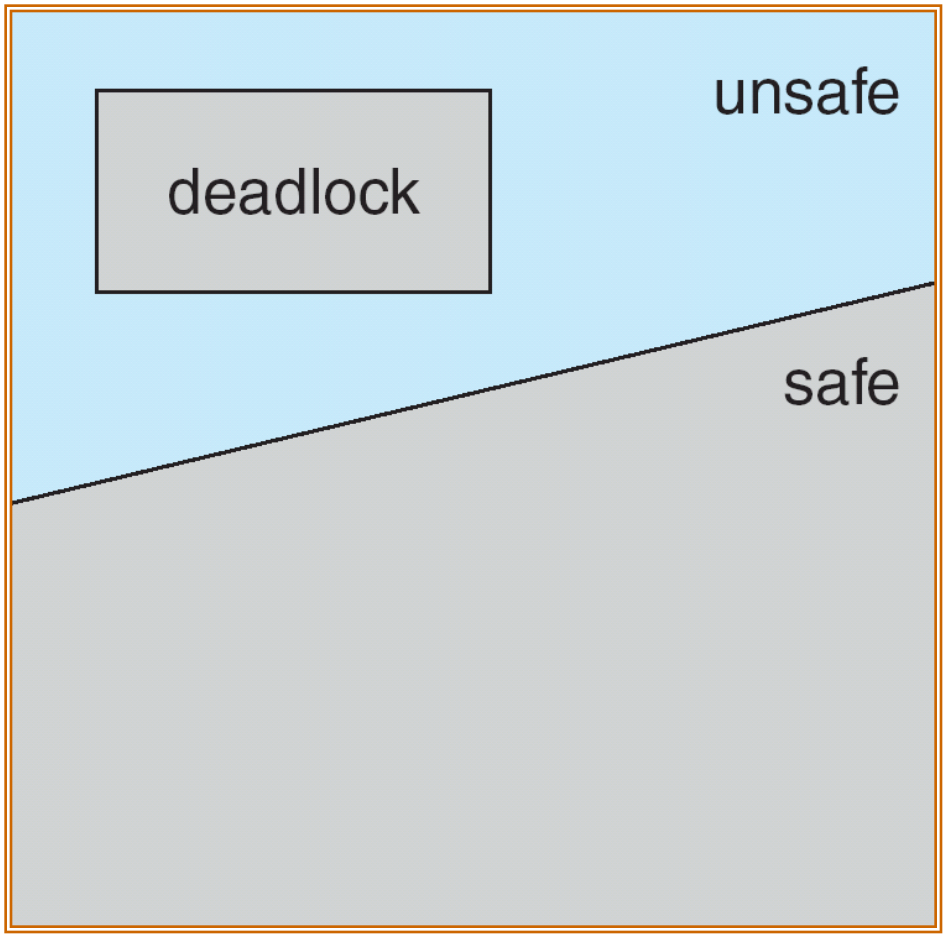
Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence <P_1, P_2, …, P_n> is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with j<I.
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Basic Facts

- If a system is in safe state $\Rightarrow$ no deadlocks.
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock.
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

Safe, Unsafe , Deadlock State



Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let *n* = number of processes, and *m* = number of resources types.

- *Available:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type R_j available.
- *Max: n x m* matrix. If *Max* [*i,j*] = *k*, then process P_i may request at most *k* instances of resource type R_j.
- *Allocation: n* x *m* matrix. If Allocation[*i,j*] = *k* then P_i is currently allocated *k* instances of R_j.
- *Need: n* x *m* matrix. If *Need*[*i,j*] = *k*, then P_i may need *k* more instances of R_j to complete its task.

Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*].

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:

   Work = *Available*

   Finish [*i*] = *false* for *i* - 1,2,3, …, *n.*

2. Find and *i* such that both:

   (a) *Finish* [*i*] = *false*

   (b) $Need_i \leq Work$

   If no such *i* exists, go to step 4.

3. *Work* = *Work* + $Allocation_i$
   Finish[*i*] = *true*
   go to step 2.

4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- *If safe $\Rightarrow$ the resources are allocated to Pi.*
- *If unsafe $\Rightarrow$ Pi must wait, and the old resource-allocation state is restored*

Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types *A* (10 instances),
  B (5instances, and *C* (7 instances).
- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Example (Cont.)

- The content of the matrix. Need is defined to be Max – Allocation.

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

Example P_1 Request (1,0,2) (Cont.)

- Check that Request $\leq$ Available (that is, (1,0,2) $\leq$ (3,3,2) $\Rightarrow$ true.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement.
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

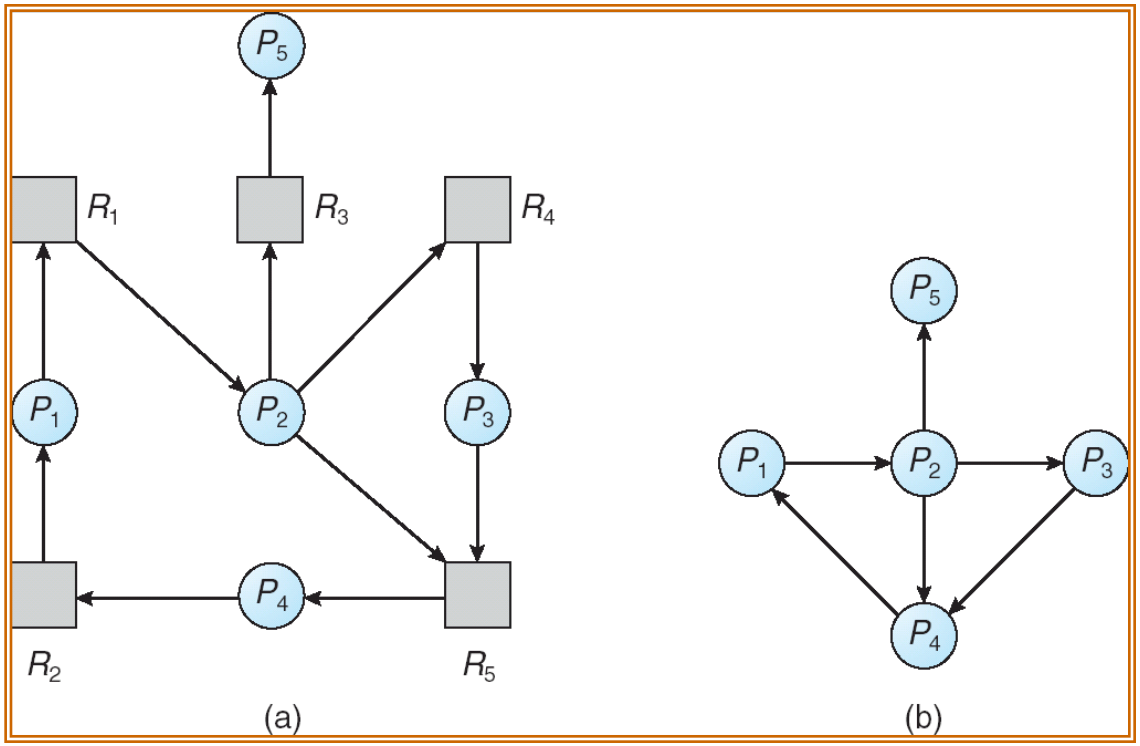
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain *wait-for* graph
  - Nodes are processes.
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

- *Available:* A vector of length *m* indicates the number of available resources of each type.
- *Allocation:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
- *Request:* An *n* x *m* matrix indicates the current request of each process. If *Request* $[i_j] = k$, then process P_i is requesting *k* more instances of resource type. R_j.

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:
   (a) *Work* = *Available*
   (b) For *i* = 1,2, …, *n*, if $Allocation_i \neq 0$, then
   Finish[i] = false;otherwise, *Finish*[i] = *true*.
2. Find an index *i* such that both:
   (a) *Finish*[*i*] == *false*
   (b) $Request_i \leq Work$

   If no such *i* exists, go to step 4.

Detection Algorithm (Cont.)

3. *Work* = *Work* + *Allocation*$_i$
   Finish[*i*] = *true*
   go to step 2.

4. If *Finish*[*i*] == false, for some *i*, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked.

Algorithm requires an order of O(m x $n^{2)}$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances).
- Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in *Finish*[*i*] = true for all *i*.

Example (Cont.)

- P_2 requests an additional instance of type *C*.

	Request
	A B C
P_0	0 0 0
P_1	2 0 1
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

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

Recovery from Deadlock: Resource Preemption

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

I/O Systems

I/O Systems

- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Streams
- Performance

Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - **Port** :It serves as an interface between the computer and other peripheral dvices.It has many uses to connect a monitor , webcam , speakers mouse etc.
  - Port typically consists of 4 things:
  1. Status : device busy , ready or error condition
  2. Control : Commands to perform
  3. Data-in : data sent from device to CPU
  4. Data-out : data sent from CPU to device
  - **Bus** (**daisy chain** or shared direct access)

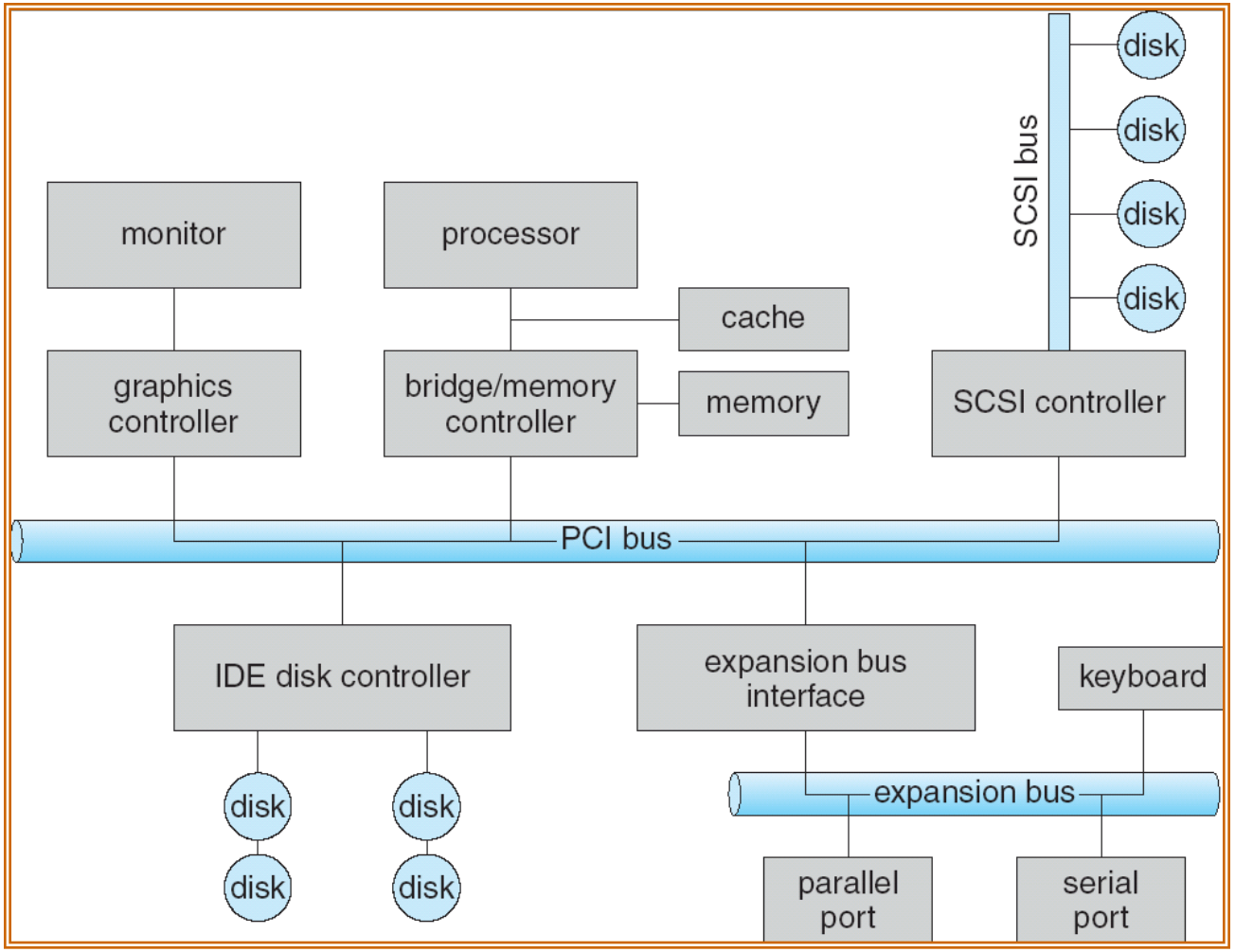
Cont..

•**Controller** (**host adapter**):Acts as an interface to the bus and actual devices . It receives the commands from the bus, translates them into device actions and reads/writes data on to the system bus.

•I/O instructions control devices
Traditional Devices : Disk drivers , printers , keyboards , modem , mouse display.
Non – Traditional Devices : Joystick , robot actuator etc.

- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped** I/O

A Typical PC Bus Structure



Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Polling

- The CPU will repeatedly ask for any new requests if any for the I/O devices.
- Determines state of device
  - command-ready
  - busy
  - Error
- **Steps for Polling:**

Step1: If bus is not busy makes bus request.

Step2:For every device there will be an device ID.

Step 3: If device gets bus grant , then it changes bus status as busy

- **Busy-wait** cycle to wait for I/O from device

Disadvantages:

1. Device inputs must be waiting for polling intervals.
2. Shortening polling intervals leads to inefficiency , since system must spend a lot of time checking idle devices.

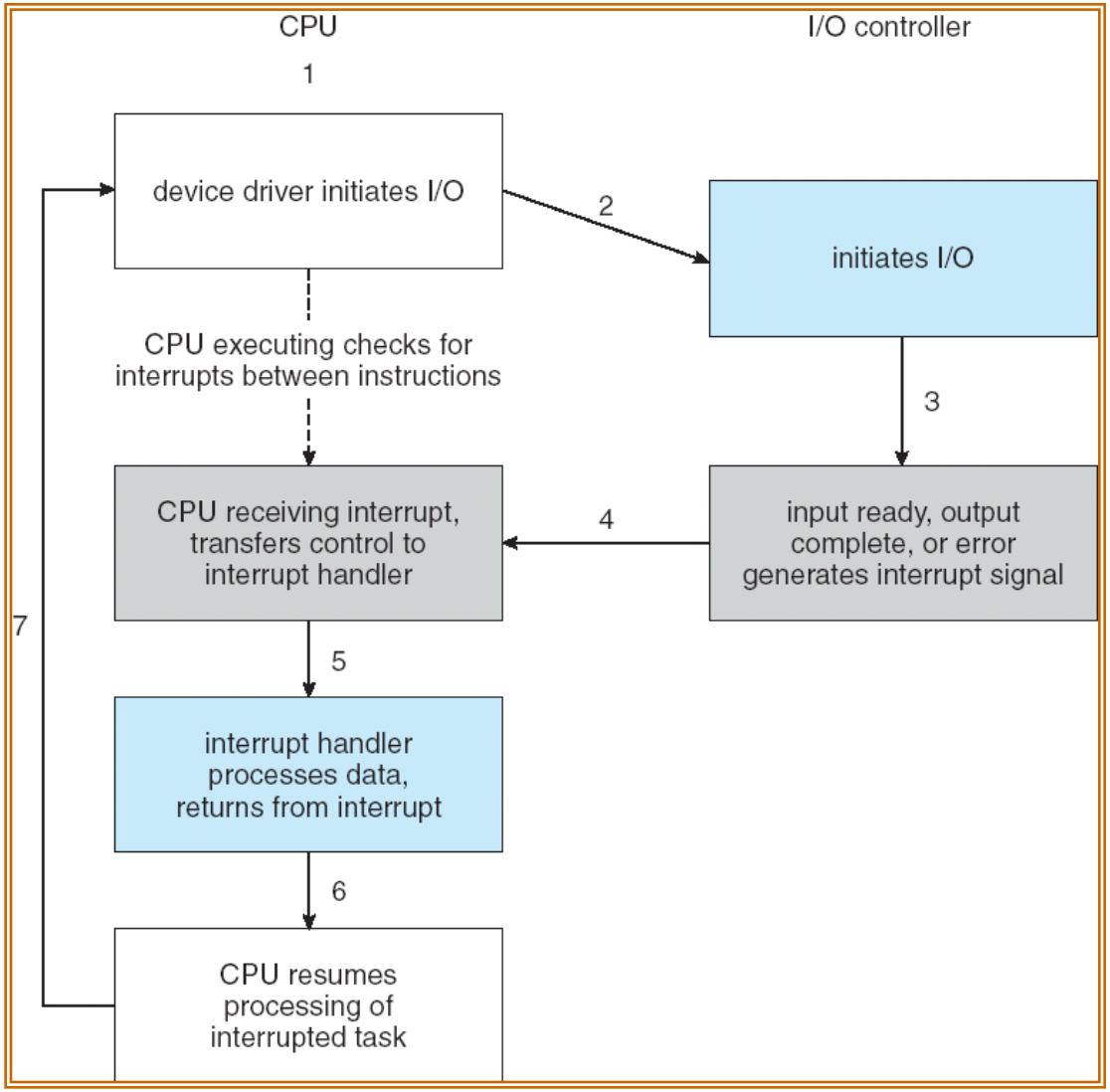
Interrupts:

- The hardware device only signals OS system with events occur.
- OS preempts any running process to handle the event.

Interrupts

- CPU **Interrupt-request line** triggered by I/O device
- **Interrupt handler** receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some **nonmaskable**
- Interrupt mechanism also used for exceptions

Interrupt-Driven I/O Cycle



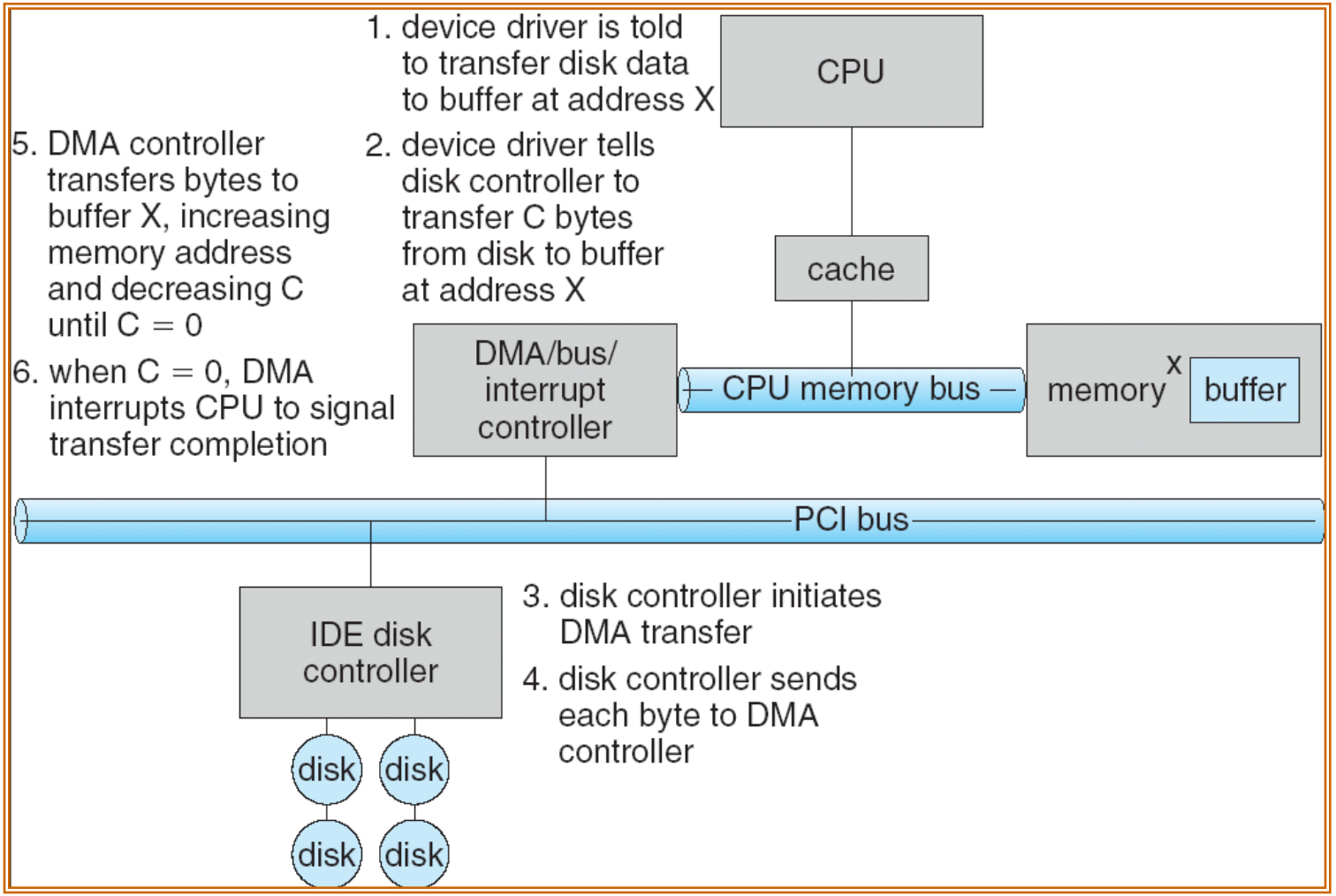
Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Direct Memory Access

- Used to avoid **programmed I/O** for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory

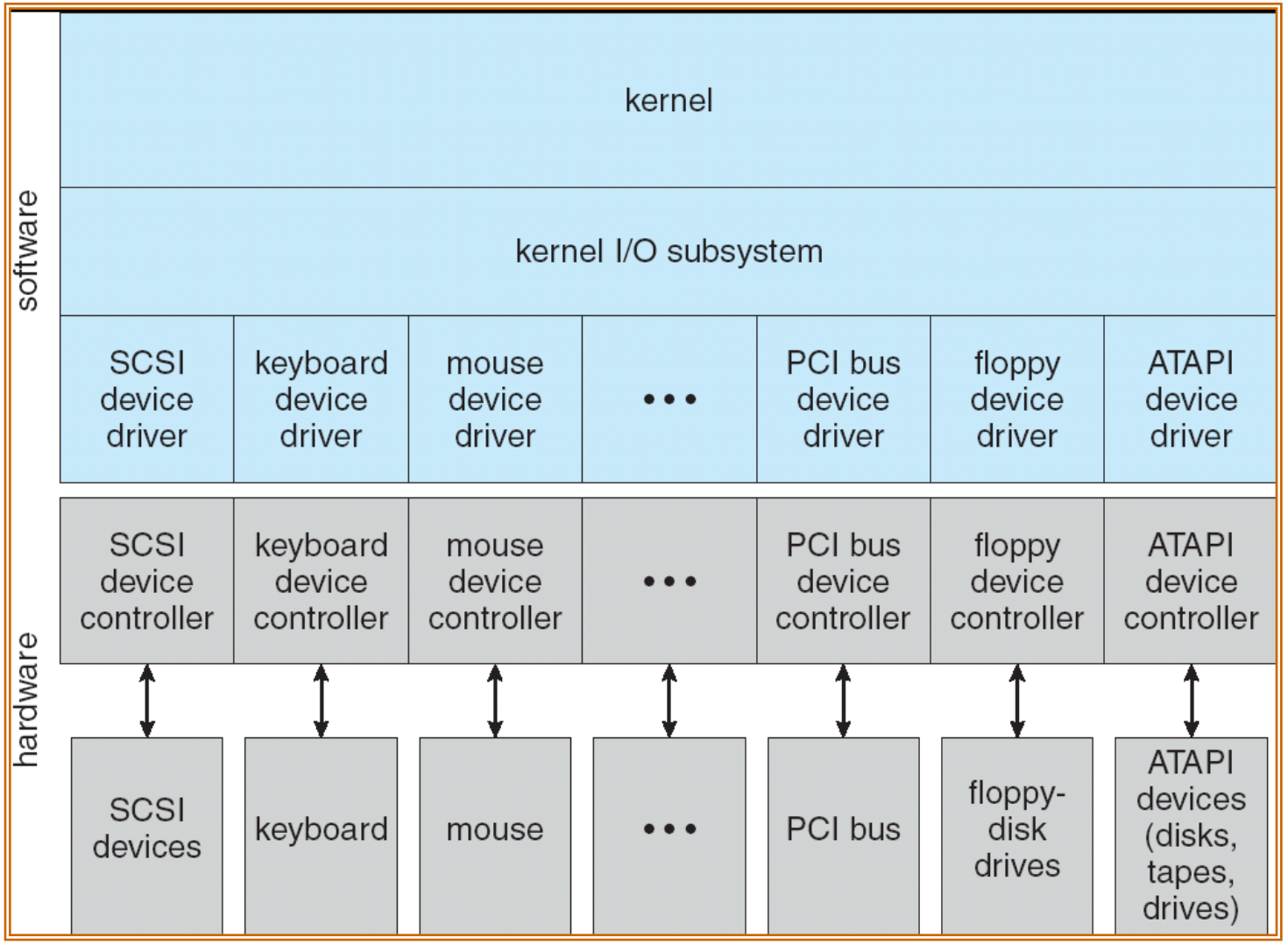
Six Step Process to Perform DMA Transfer



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions
  - **Character-stream** or **block**
  - **Sequential or random-access**
  - **Sharable or dedicated**
  - **Speed of operation**
  - **read-write, read only,** or **write only**

A Kernel I/O Structure



Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

- The major access conventions include block I/O , character stream I/O , memory mapped file access and network sockets.
- Most OS have an escape that transparently passes arbitrary commands from an application to device driver.
- The system call is ioctl()-for I/O Control.
- the ioctl() system call has 3 arguments

1. file descriptor - connects application to the driver by referring to the hardware device.
2. integer that select one of the command implemented in the driver.
3. pointer to an arbitrary data structure in memory that enables the application and the driver to communicate any necessary control information or data.

Block and Character Devices

- block device interface captures all the aspects necessary for accessing disk drivers and block –oriented devices.
- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible

- Character devices include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing

Network Devices

- one interface available in many operating systems , including UNIX and Windows NT is the network socket interface.
- The system call in the socket interface enable an application to create a socket.
- To connect a local socket to remote address which plugs this socket created by another application , to listen for any remote application to plug into the local socket , and to send receive packets over the connection.
- Socket interface also provides a function select() that manages a set of sockets.
- The select() returns the information about which socket is have a packet waiting to be received and which socket have a room to accept a packet to be sent.
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Clocks and Timers

- Provide current time, elapsed time, timer
- **Programmable interval timer** used for timings, periodic interrupts
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers

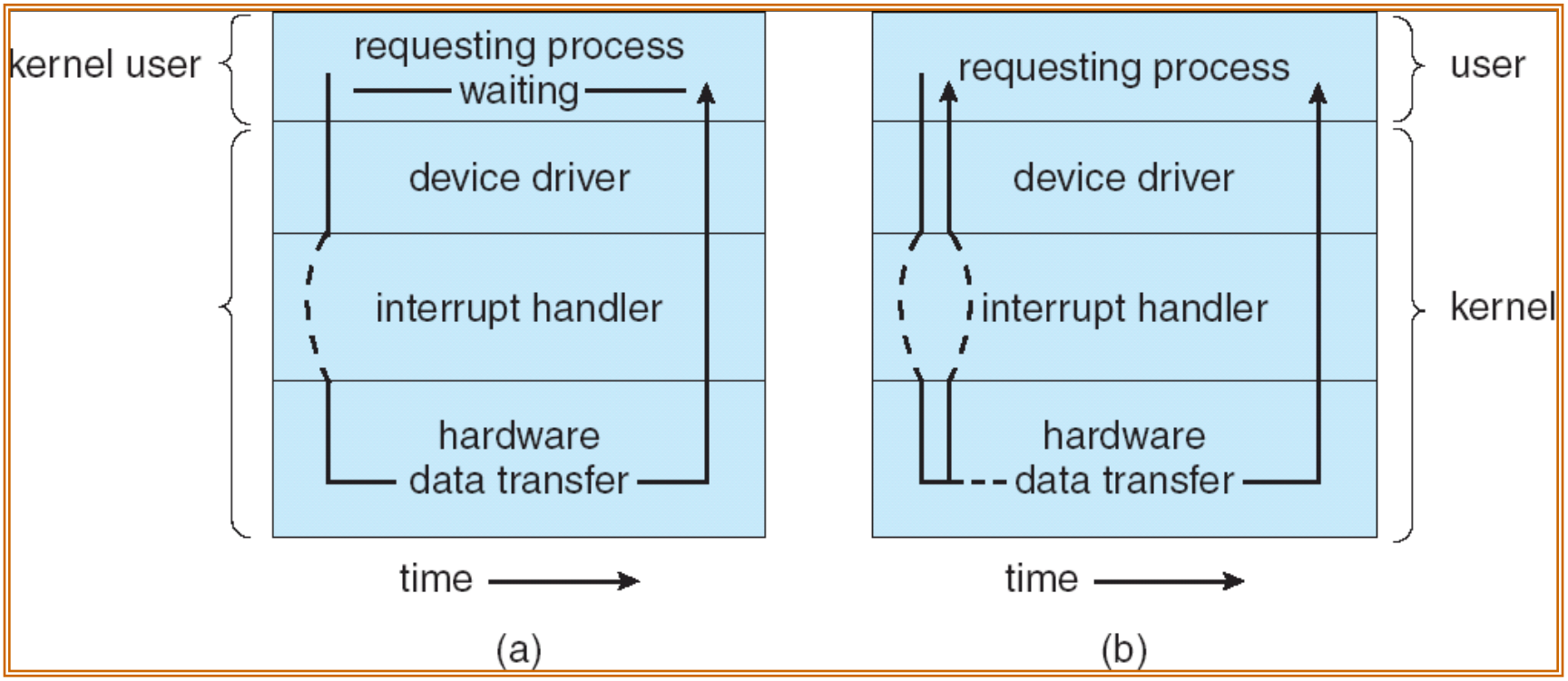
Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs

- **Nonblocking** - I/O call returns as much as available(keyborad, mouse , video conferencing)
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written

- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Two I/O Methods



Synchronous

Asynchronous

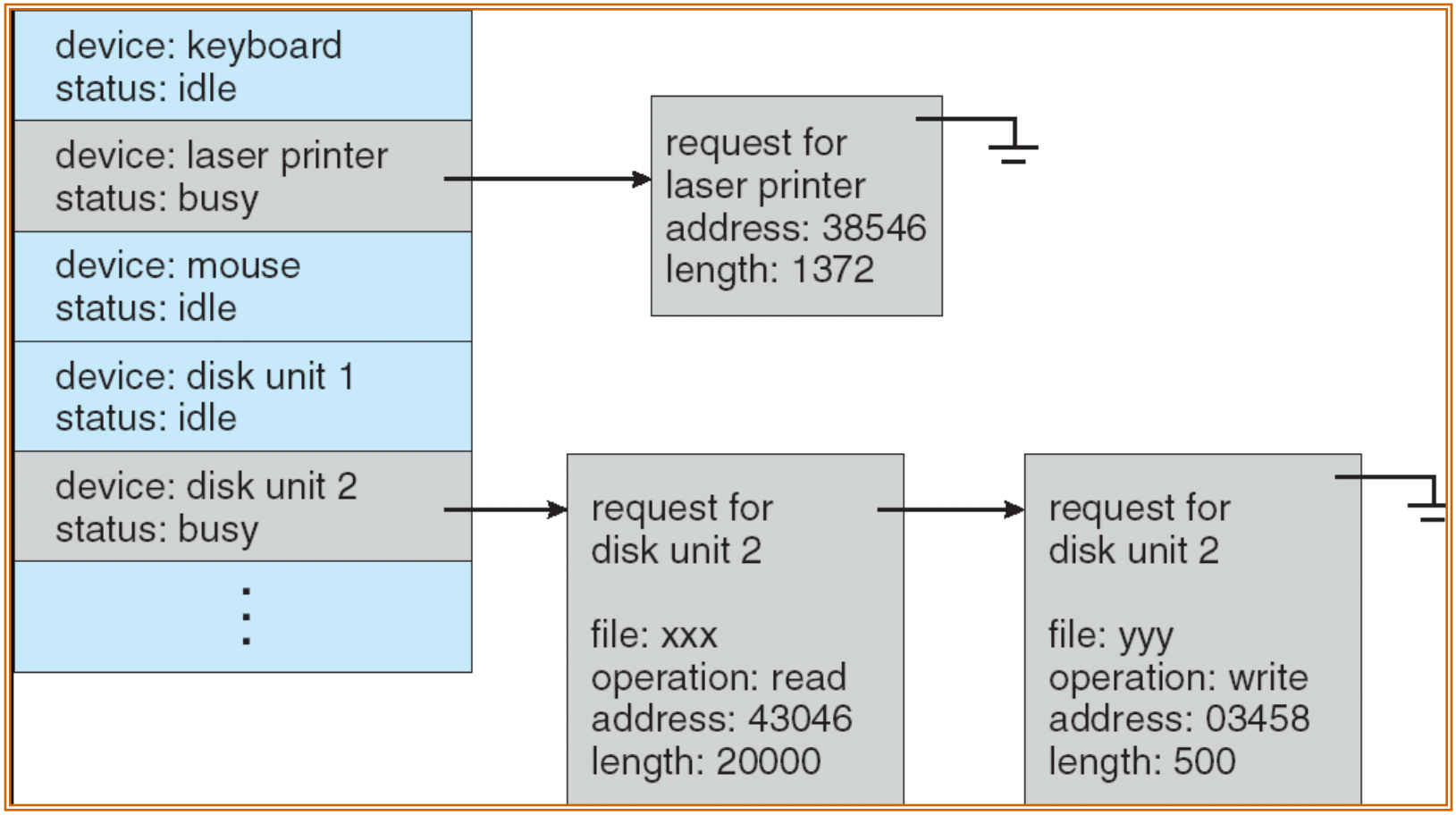
Kernel I/O Subsystem

- I/O Scheduling
  - Some I/O request ordering via per-device queue

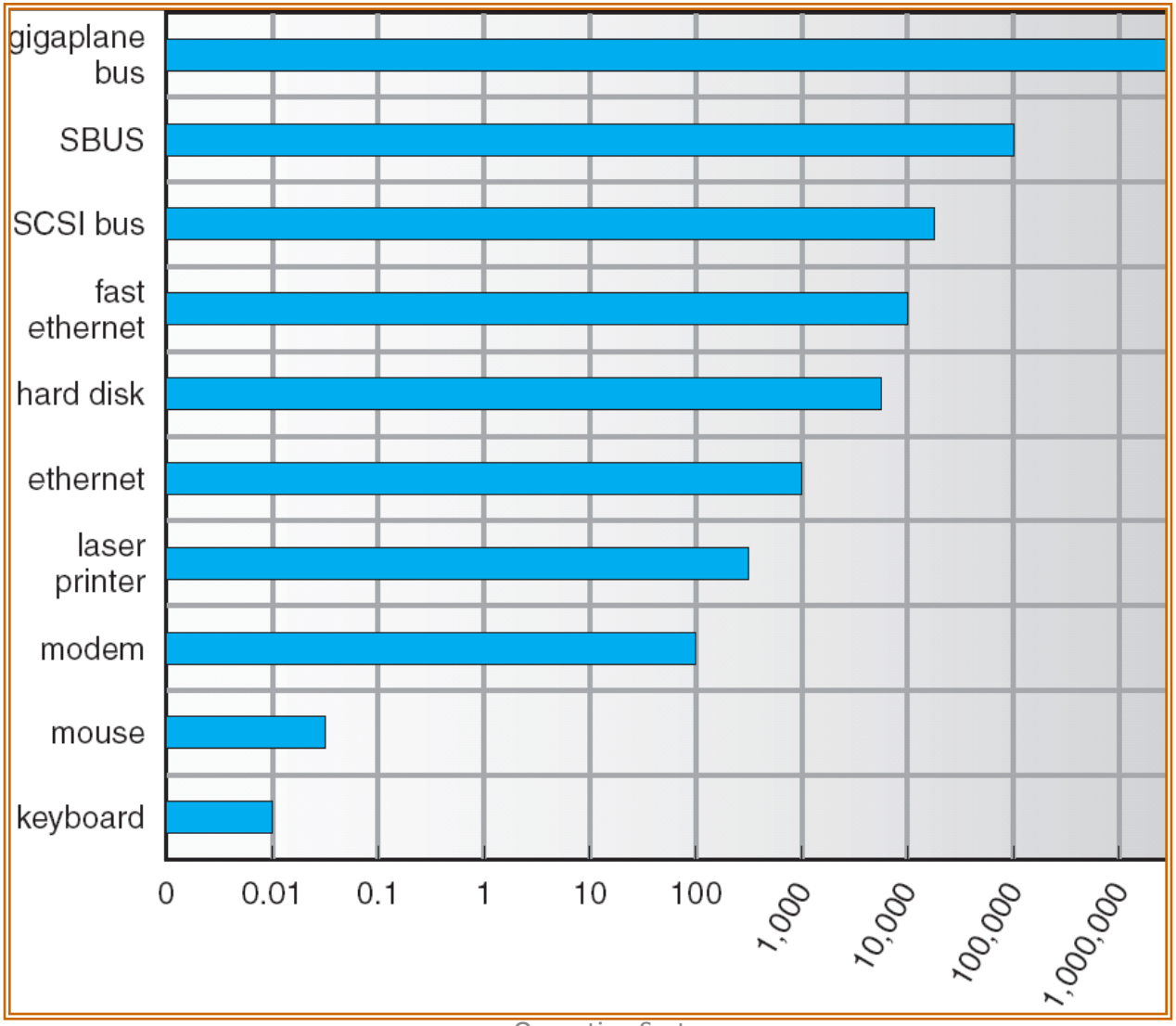
  Device –status table:contains entry for each I/O device indicating its type , address and state.

  OS indexes into I/O device table to determine device status and to modify table entry to include interrupt.
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch(web)
  - To maintain “copy semantics”(buffer of data modified)

Device-status Table



Sun Enterprise 6000 Device-Transfer Rates



Kernel I/O Subsystem

- **Caching** - fast memory holding copy of data
  - Always just a copy
  - Key to performance
- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing
- **Device reservation** - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

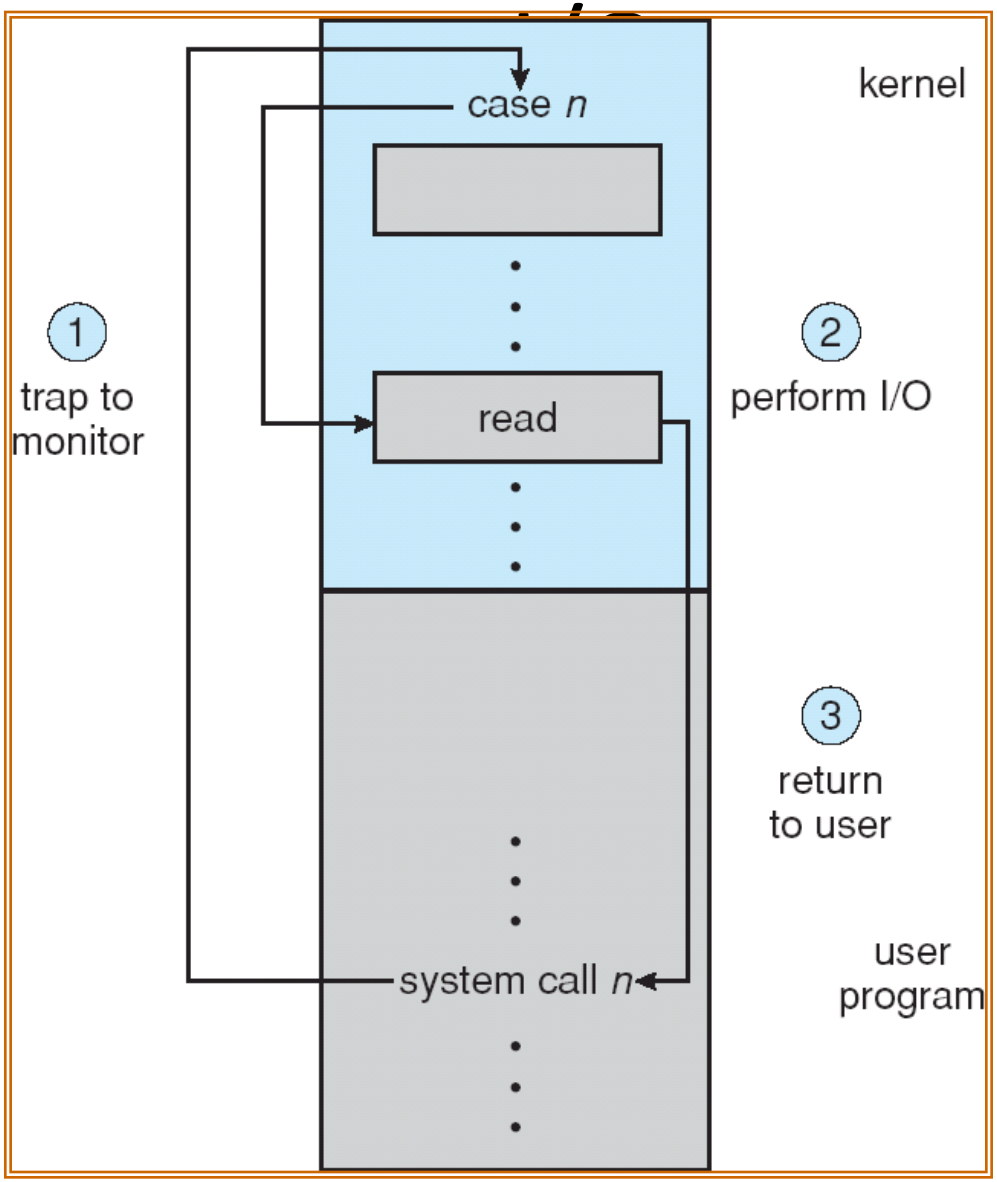
Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

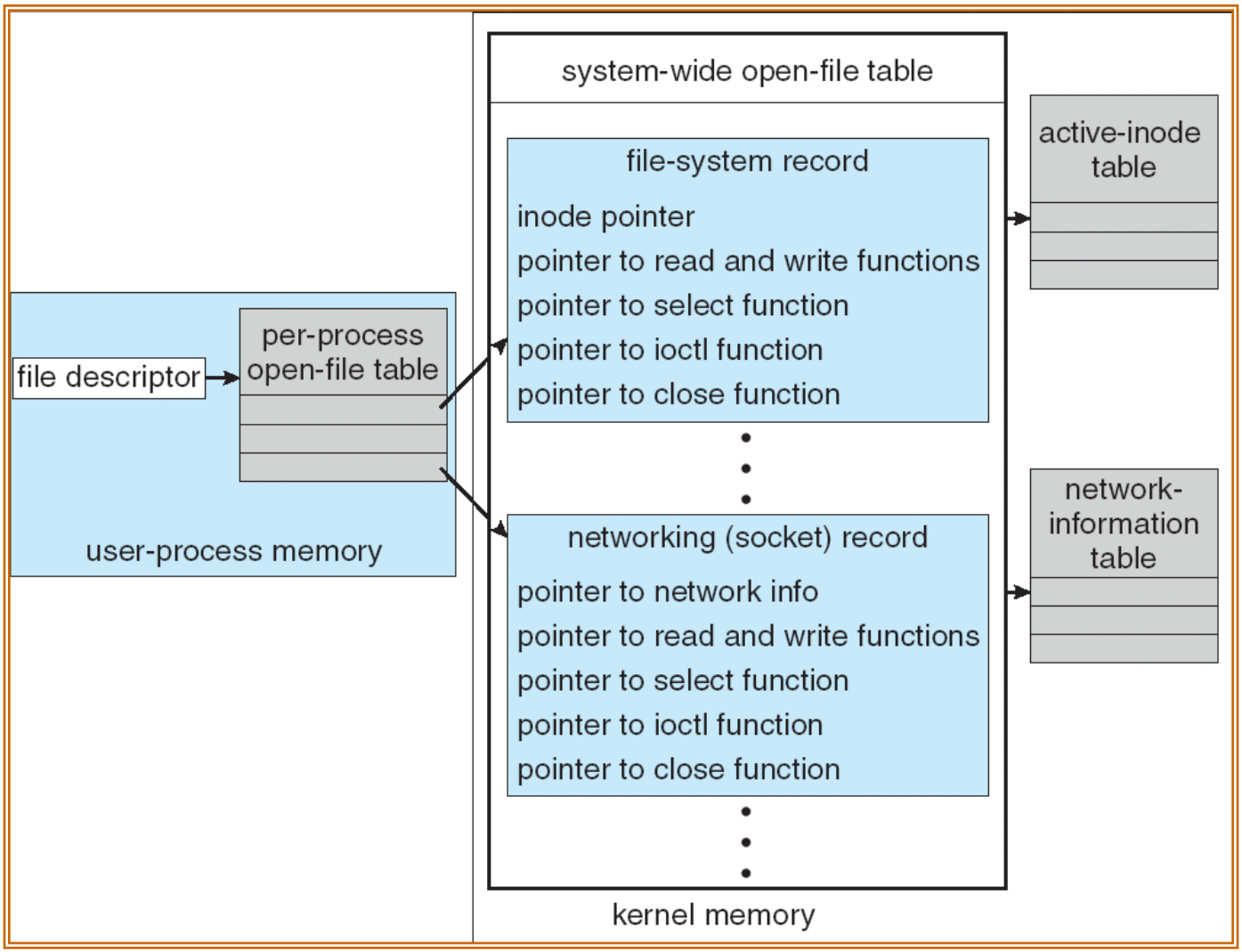
Use of a System Call to Perform



Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O

UNIX I/O Kernel Structure

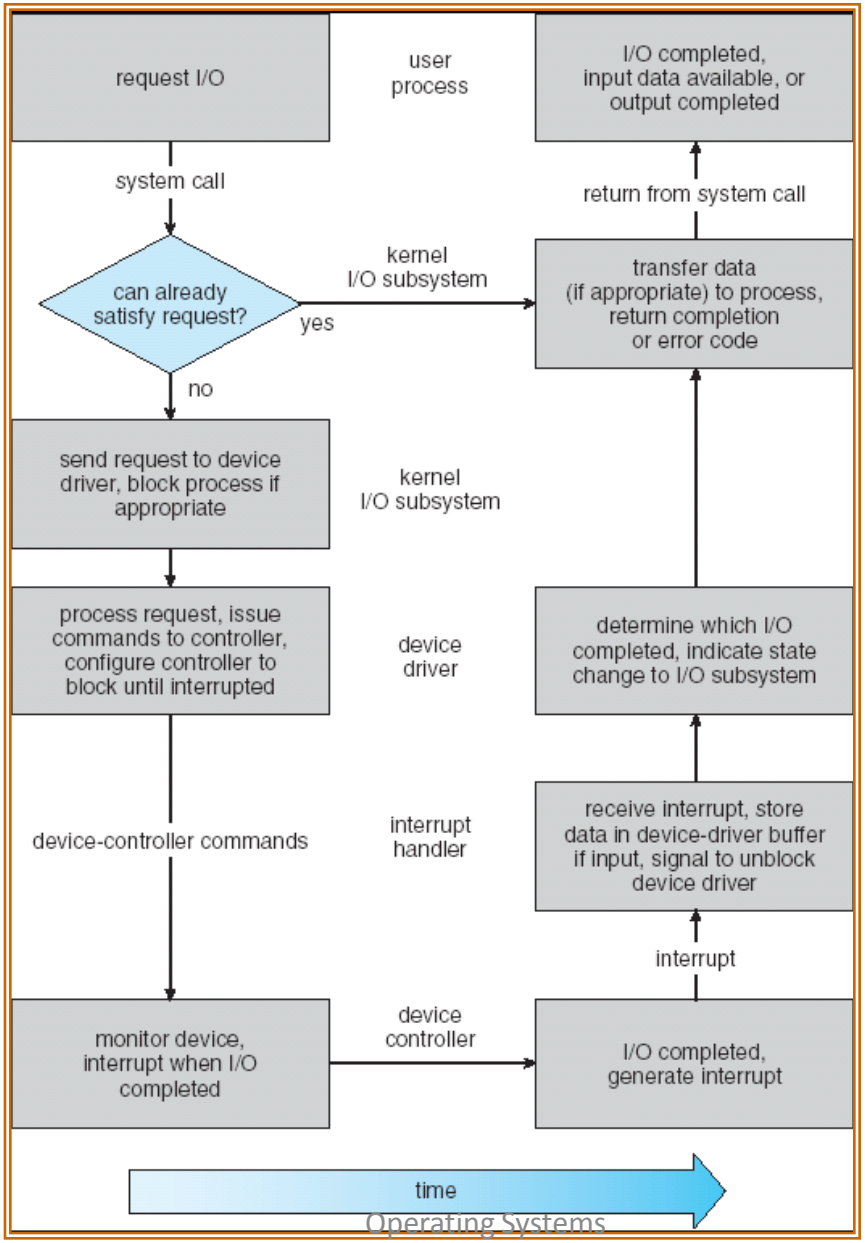


I/O Requests to Hardware Operations

- A process issues a block read() system call to a file descriptor of a file that has been opened previously.
- The system-call code in the kernel checks the parameters for correctness . In the case of input , if data already available in the buffer cache , the data are returned to the process and the I/O request completed.
- Otherwise a physical I/O must be performed. The process is moved from run queue to wait queue and the I/O request is scheduled.
- The device driver allocates kernel buffer space to receive the data and schedule the I/O . Device driver send command to the device controller by writing into device control register.
- The device controller operates the device hardware to perform the data transfer.

- The driver may poll for status an data or it may have set up for DMA transfer into kernel memory. So we assume that DMA is transferring and generates an interrupt when transfer completes
- The correct interrupt handler receives the interrupt via interrupt vector table, stores any necessary data , signals , the device driver and returns from the interrupt.
- The device driver receives the signal , determines which I/O request has completed and signals the kernel I/O subsystem that the request has been completed.
- The kernel transfers data to the address space of the requesting process and moves the process from the wait queue back to the ready queue.

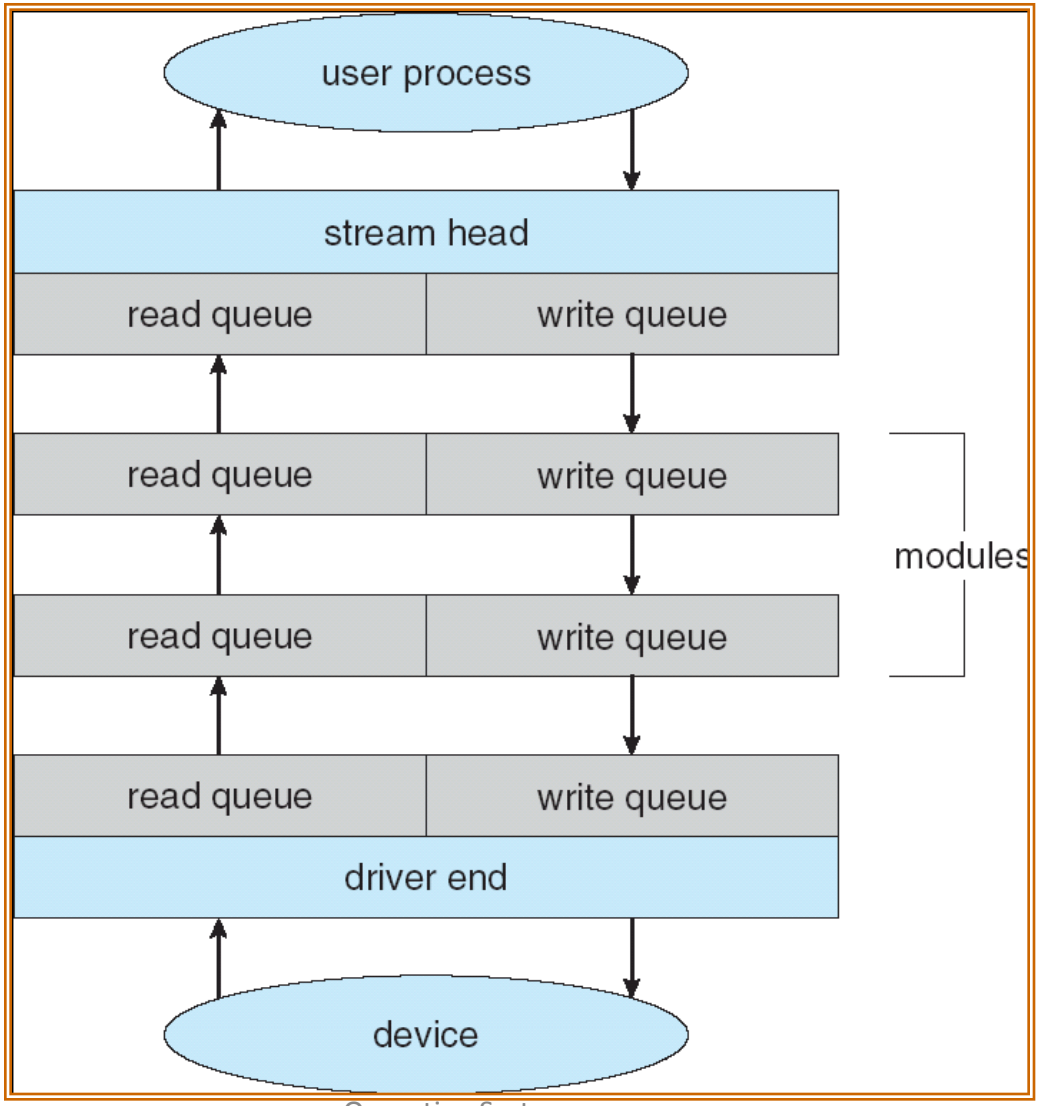
Life Cycle of An I/O Request



Streams

- A stream is a full duplex connection between a device driver and user level process.
- It consists of a **stream head** that interfaces with the user processes.
- A **driver end** that controls the devices and zero or more modules between the stream head and driver end.
- Each module contains a read queue and a write queue.
- A user writes the data into the device using **write() or putmsg().**
- A user reads data from stream heads using either **read() or getmsg()**

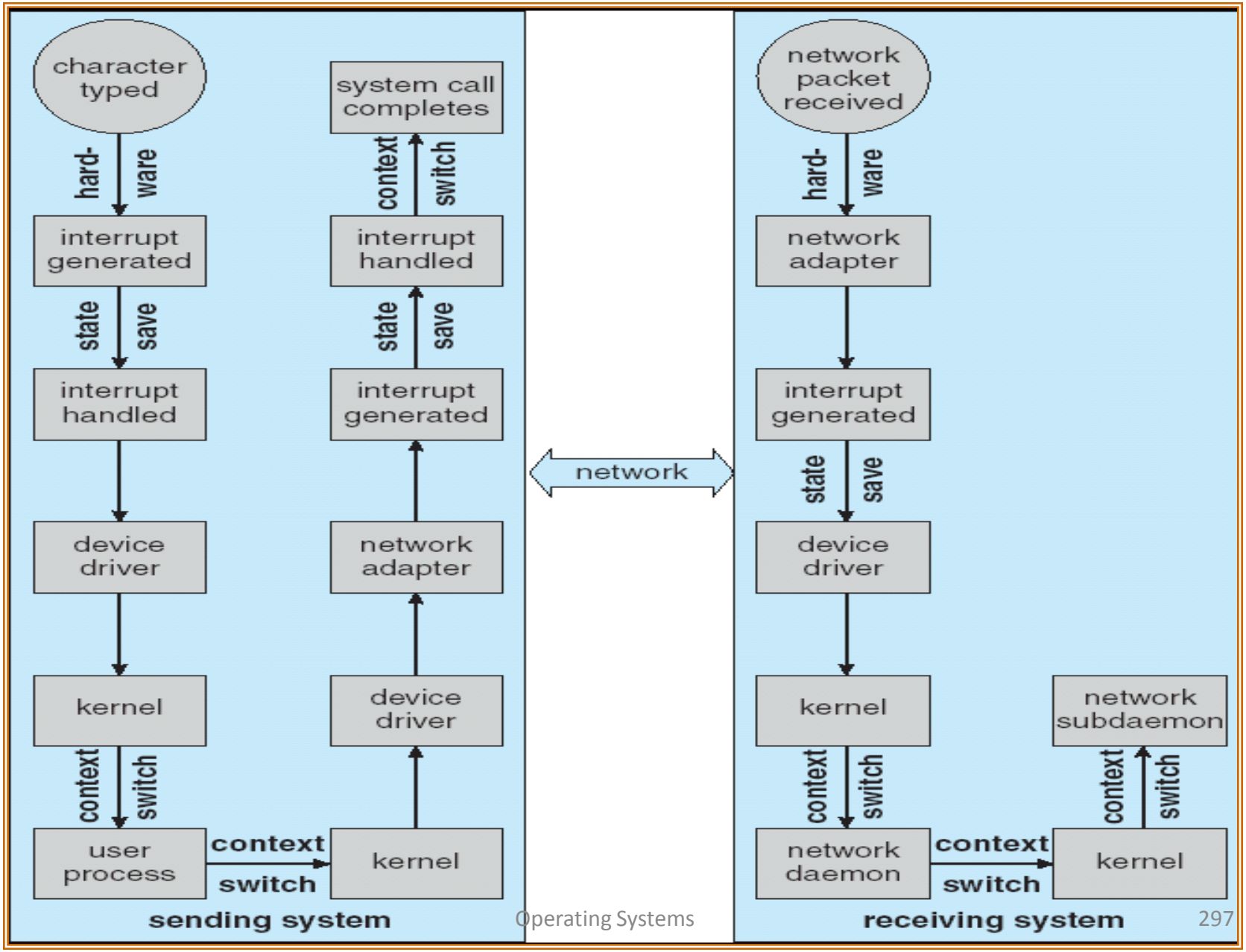
The STREAMS Structure



Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

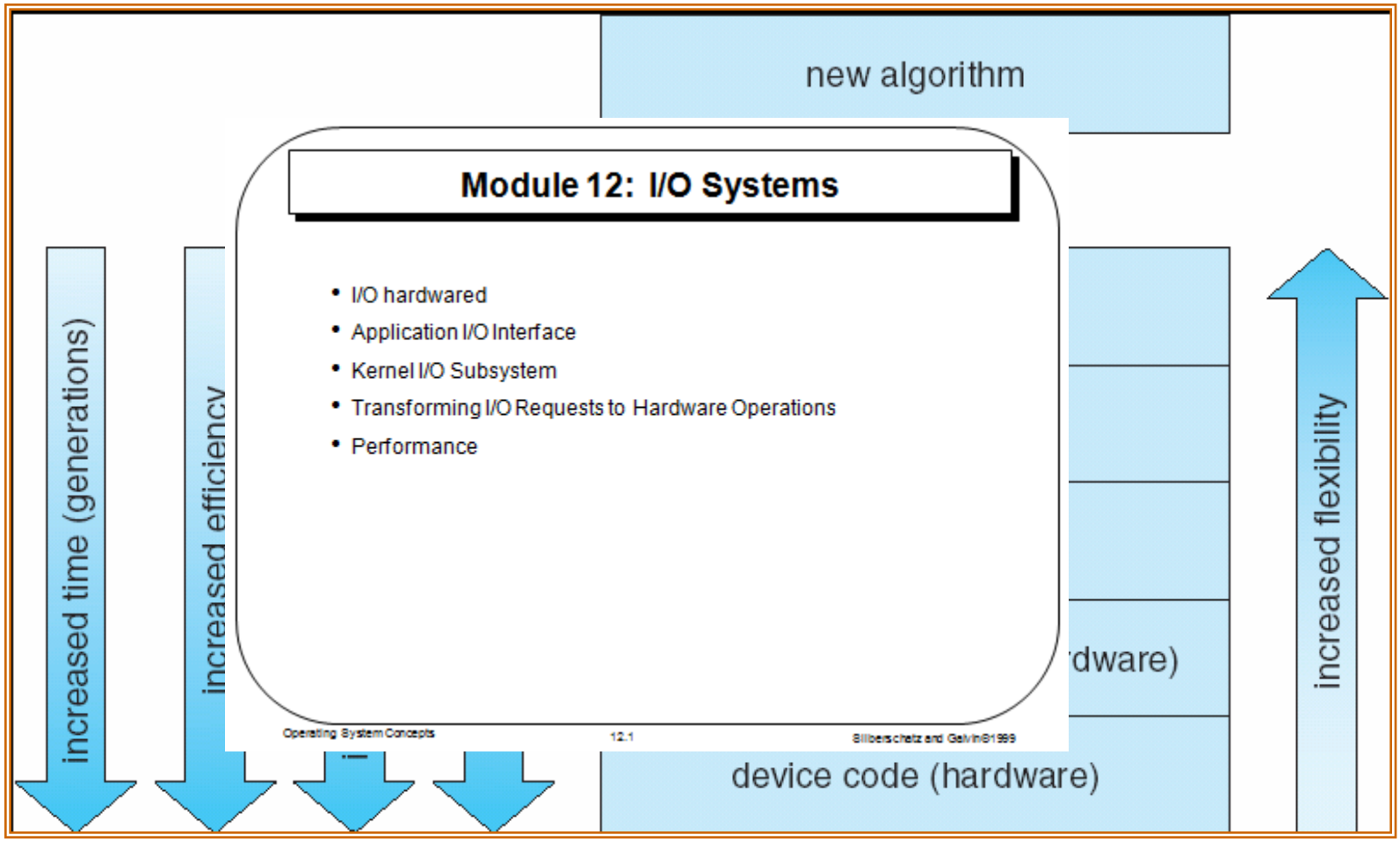
Inter computer Communications



Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression



UNIT-IV

Syllabus

File System Interface – The Concept of File, Access methods, Directory Structure, File System Mounting, File Sharing, Protection, File System Implementation File System Structure, File System Implementation, Allocation methods, Free-Space Management, Directory Implementation, Efficiency and Performance.

Mass Storage Structure – Overview of Mass Storage Structure, Disk Structure, Disk Attachment, Disk Scheduling, Disk Management, and Swap space Management.

- The File system consists of two distinct parts:

1. Collection of files
2. Directory Structure.

- Computer will store information in various storage media such as magnetic tapes , magnetic disks, optical disks.
- A file is a named collection of related information that is recorded in secondary storage.
- In users view a file is the smallest allotment of logical secondary storage i.e data can be written in secondary storage unless they are within a file.
- File consists of program and data .
- File is a sequence of bits , bytes , lines or records.
- File meaning is given by file creator & user.
- The file information may be of different types, source program , object program , executable program , numeric data , text , payroll record , graphic images etc.

File Attributes

- **Name** – only information kept in human-readable form.
- **Type** – needed for systems that support different types.
- **Location** – pointer to file location on device.
- **Size** – current file size.
- **Protection** – controls who can do reading, writing, executing.
- **Time**, **date**, **and user identification** – data for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk.

File Operations

- create : Space should be available and entry for new file must be made in directory.
- write : System call specifying both the name of the file and information written to the file.(write pointer)
- read : System call specifying both the name of the file and in memory the next block of the file should be put.(read pointer)
- reposition within file – The directory is searched for the appropriate entry , and the current –file –position pointer is repositioned to a given value. (file seek)
- delete : To delete a file , we search the directory for the named file.
- truncate: The user may want to erase the contents of the file

.

Cont…

- An open system call is first made before a file is opened .
- The operating system is going to maintain a small table called as open –file table.
- When a file operation is requested the file is specified via an index into this table, so no searching is required.

Different piece of information are associated with an open file are:

- File pointer: On system it doesn't include a file offset as part of the read() and write() system calls, the system must track the last read –write location as a current fil epointer.
- File-open count :It tracks the number of opens and closes and reaches zero on last close.
- Disk location of the file:appending the information on the file , the information needed to locate the file on disk for each operation.
- Access rights: Each process opens a file in an access mode.

File Types – name, extension

File Type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	complied, machine language, not linked
Source code	c, p, pas, 177, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data documents
Word processor	wp, tex, rrf, etc.	various word-processor formats
Library	lib, a	libraries of routines
Print or view	ps, dvi, gif	ASCII or binary file
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed.

Access Methods

- Sequential Access

  read next
  write next
  reset
  no *read after last write*
  (rewrite)

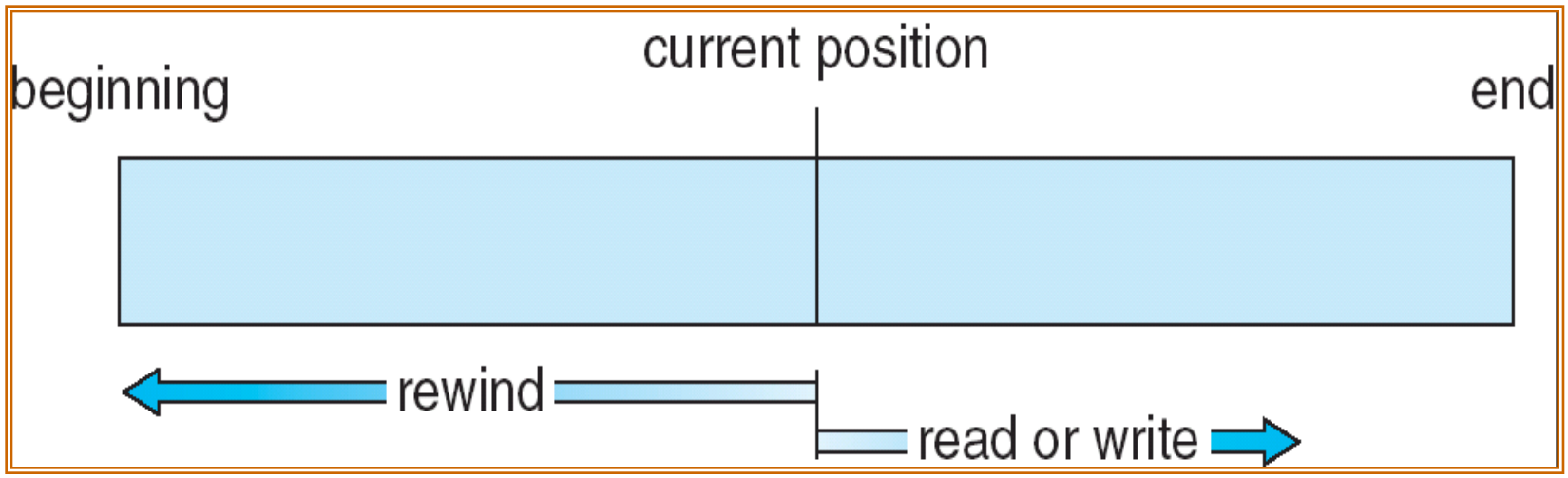
- Direct Access

  read n
  write n
  position to n
  read next
  write next
  rewrite n

  n = relative block number

Sequential-access files

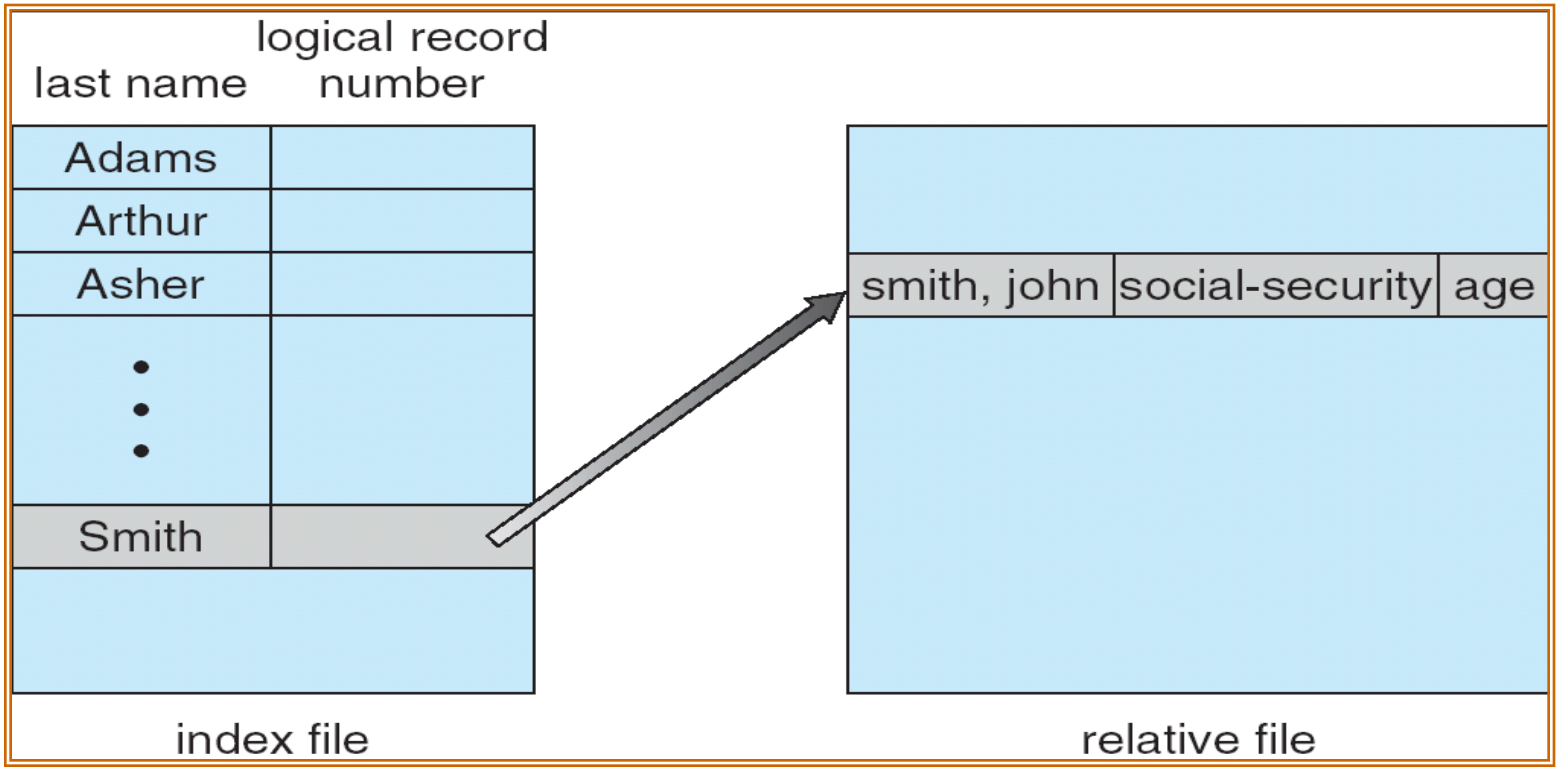
•Sequential access means that a group of element such as tapes , disk file , memory array is accessed in a predetermined ordered sequence.
•Sequence access is something the only way of accessing the data eg: tapes
•It may also be the access method of choice , eg: if all that is wanted is to process a sequence of data element in order.



Simulation of Sequential Access on a Direct-access File

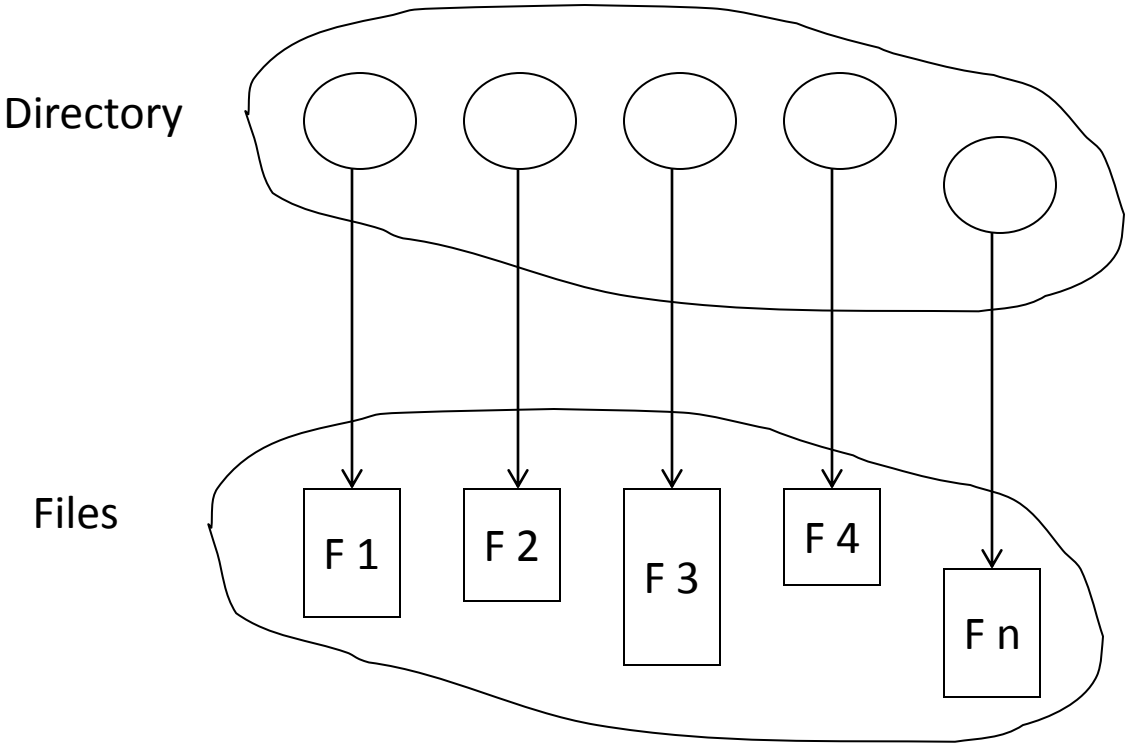
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Example of Index and Relative files



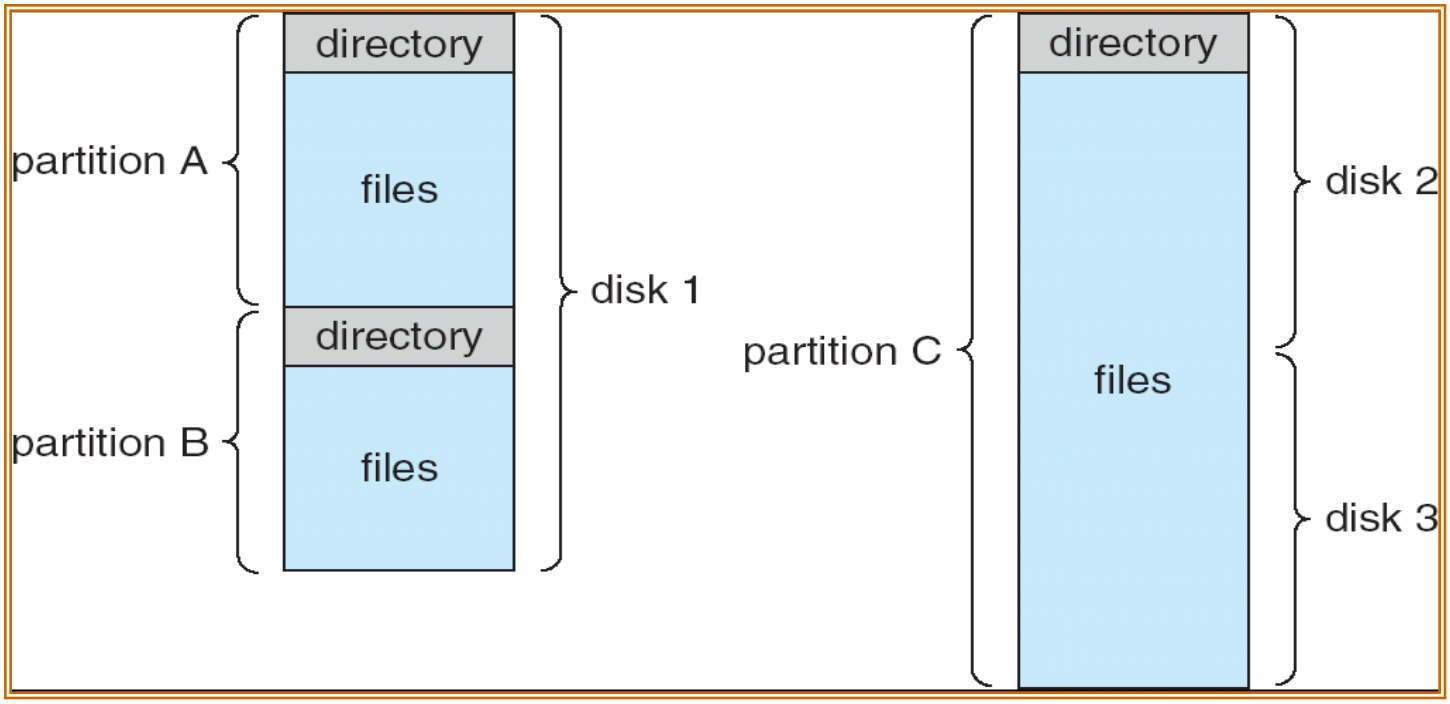
Directory Structure

- A collection of nodes containing information about all files.



- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.

A Typical file system organization



Information in a Device Directory

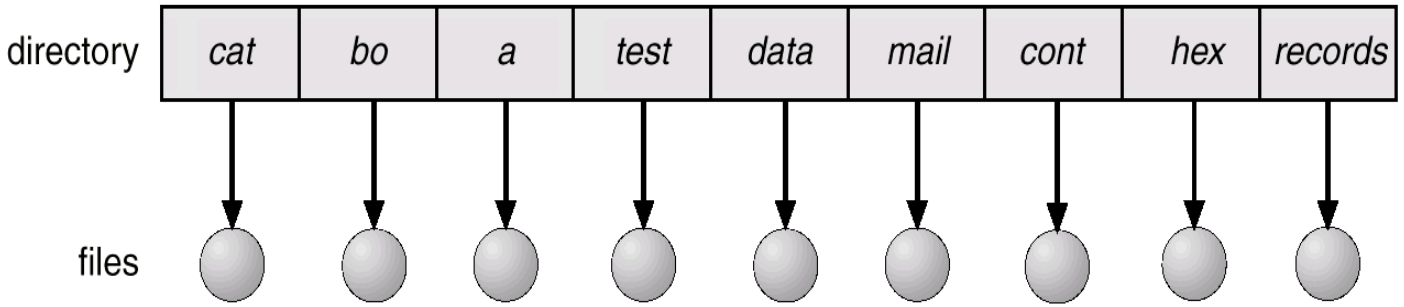
- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Single-Level Directory

- The single level directory structure is as follows:.
- All the folders are in the same directory which is easy to support and understand.

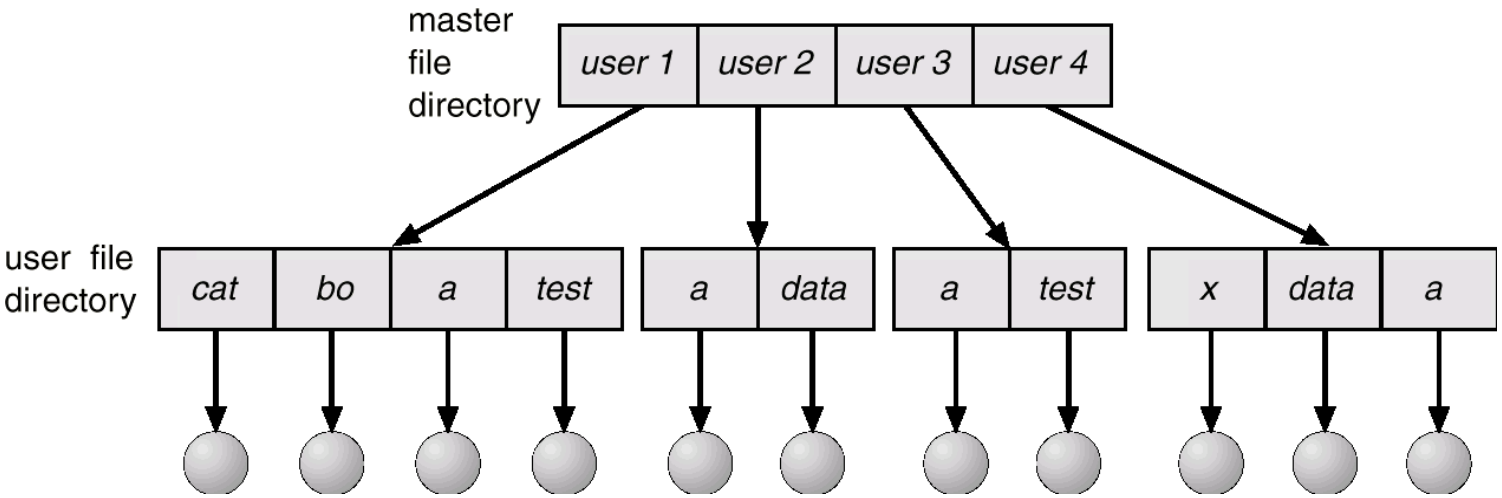


Limitations:

- Naming problem
- Grouping problem

Two-Level Directory

- Separate directory for each user.
- Each user has his own user file directory (UFD).



- Can hide the information from other users
- Can have the same file name for different user
- Efficient searching
- Providing the security with in the system level

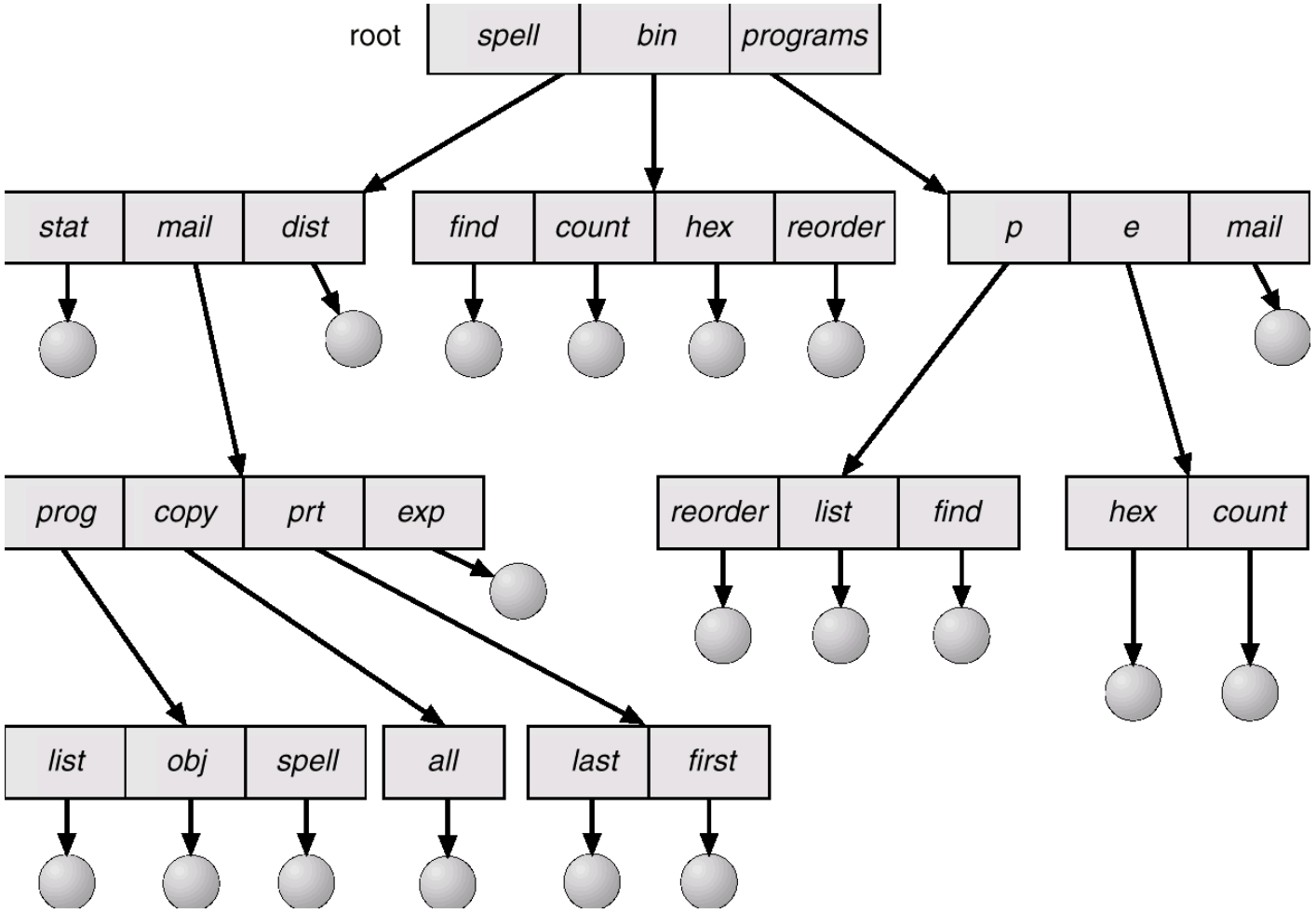
- Advantages :

1. No Name-collision
2. Isolates Users

- Limitations:

1. If the processes are dependent on each other.

Tree-Structured Directories



Tree-Structured Directories (Cont.)

- There are two types of path names:

1. Absolute pathname: It begins at the root and follows a path down to the specified file, giving the directory names on the path.
2. Relative pathname: It defines a path from the current directory.

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list

Tree-Structured Directories (Cont.)

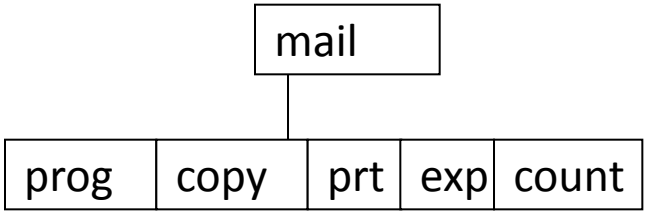
- Absolute or relative path name
- Creating a new file is done in current directory.
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory.

  mkdir <dir-name>

  Example: if in current directory /spell/mail

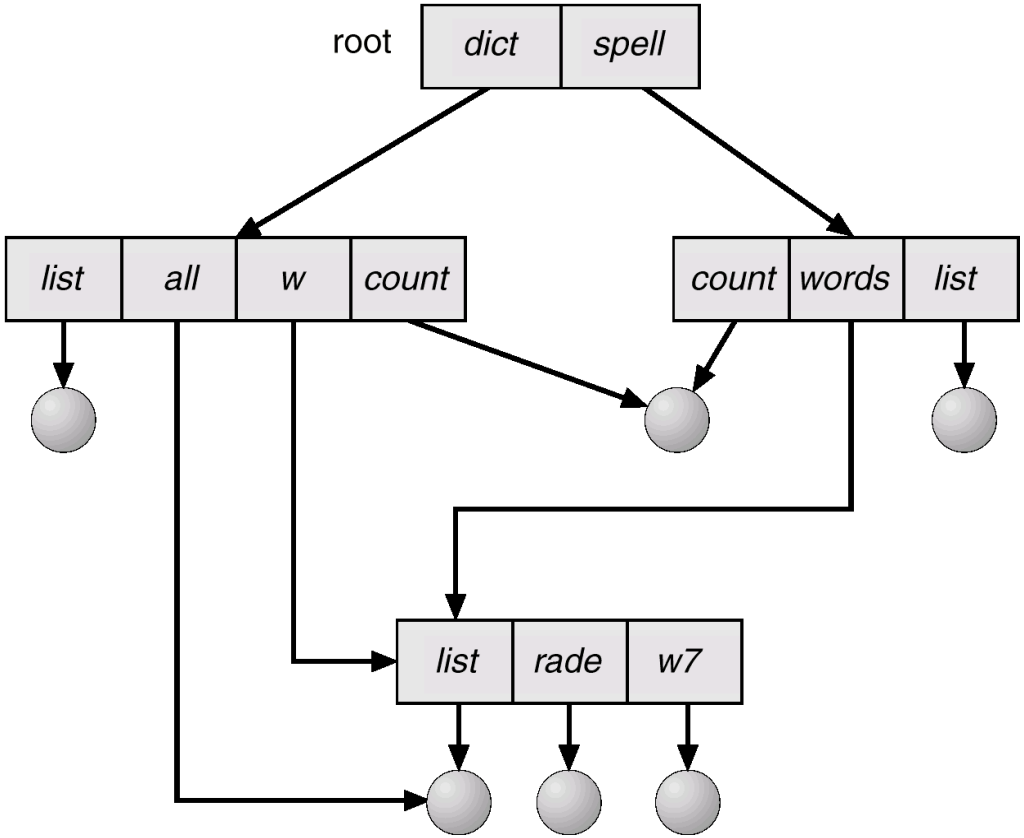
  mkdir count



- Deleting "mail" $\Rightarrow$ deleting the entire subtree rooted by "mail".

Acyclic-Graph Directories

- Have shared subdirectories and files.

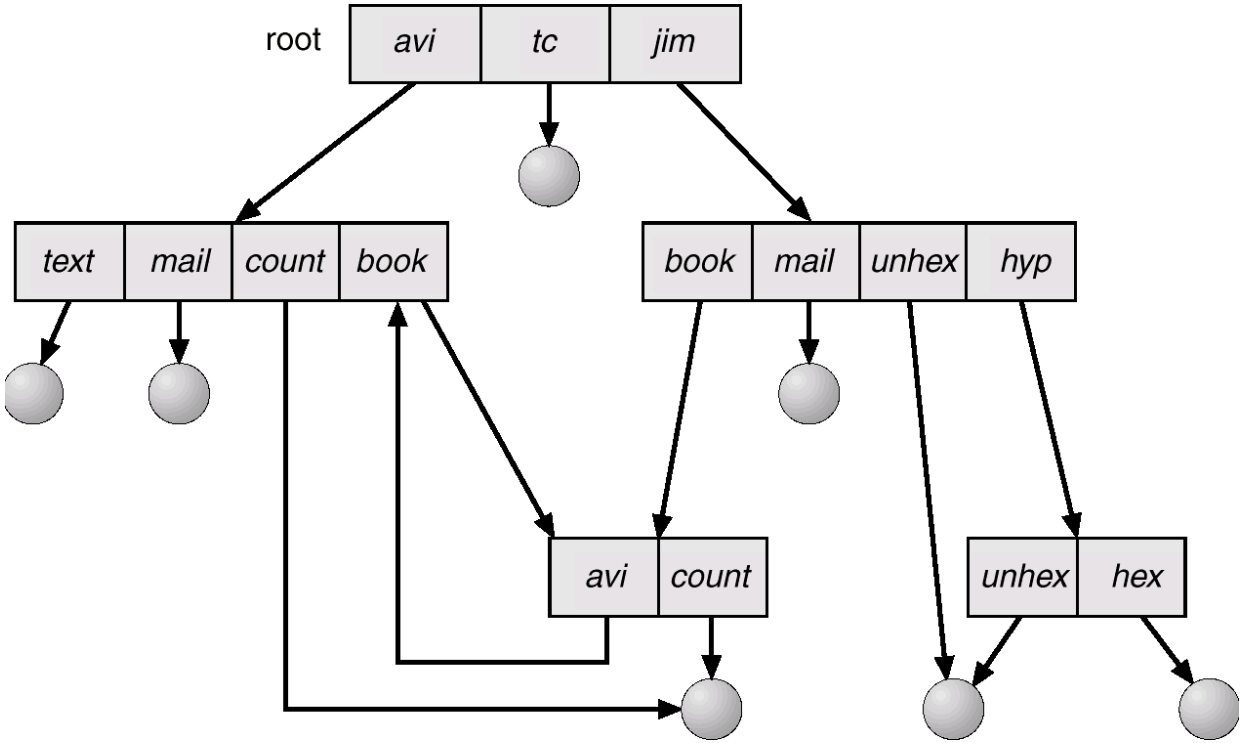


Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *list* $\Rightarrow$ dangling pointer.

  Solutions:
  - Backpointers, so we can delete all pointers.
    Variable size records a problem.
  - Backpointers using a daisy chain organization.
  - Entry-hold-count solution.

General Graph Directory



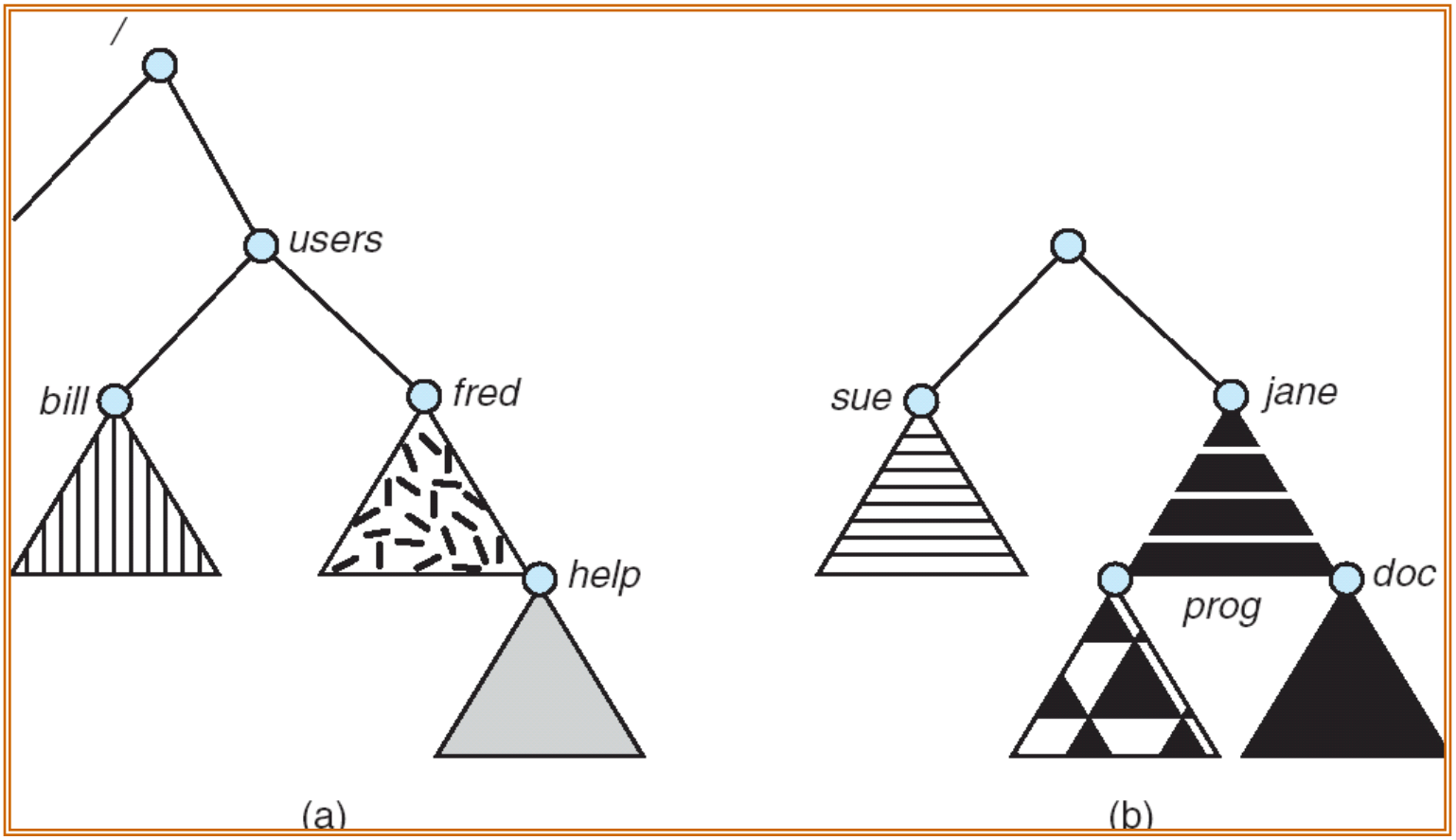
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories.
  - Garbage collection.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

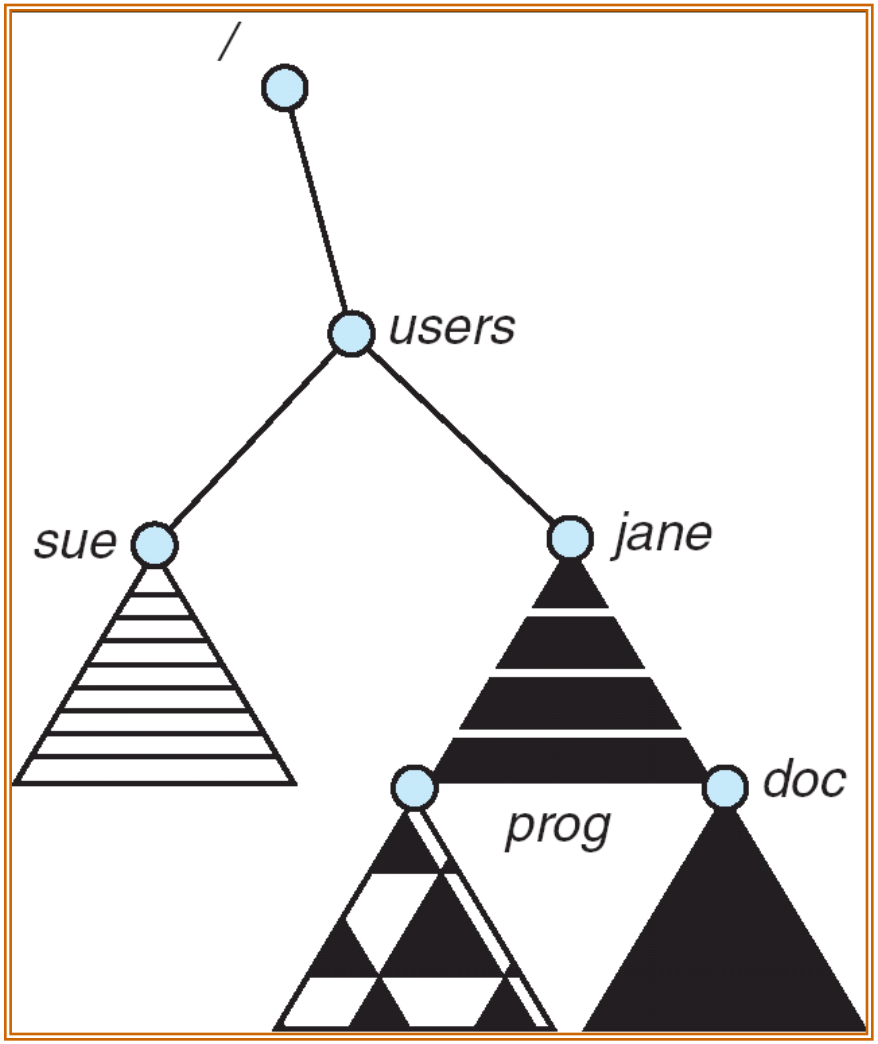
File system Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**

a) Existing b) Unmounted partitions



Mount Point



File Sharing

---Sharing of files on **multi-user systems, file naming and file protection becomes preeminent.(Owner/Group)**

---Sharing may be done through a **remote file system** scheme , through FTP protocol, DFS (Distributed File systems),WWW(World-wide-web).

----Sharing may be done through a **client-server systems, through IP addressing.**

---On **distributed information systems,** files may be shared by all the systems across a network.

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read: Read from the file
  - Write: Write or rewrite the file.
  - Execute: Load the file into memory and execute it.
  - Append: Write new information at the end of the file.
  - Delete: Delete the file and free its space for possible reuse.
  - List: List the name and attributes of the file.

Access Lists & Groups

- Only administrators can create users & groups
- 3 modes of access: **r**ead, **w**rite, e**x**ecute
- 3 classification of users in connection with each file:

a) **Owner**: The user who created the file is the owner.

b) **Group**: A set of users who are sharing the file and need similar access is a group, or work group.

c) **Universe:** All other users in the system constitute the universe.

File-System Implementation

- File-System Structure
- Allocation Methods
- Free-Space Management
- Directory Implementation
- Efficiency and Performance
- Recovery

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- A file system possess two quite different design problems:

------The first problem is defining how the file system should look to the user.

-------The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

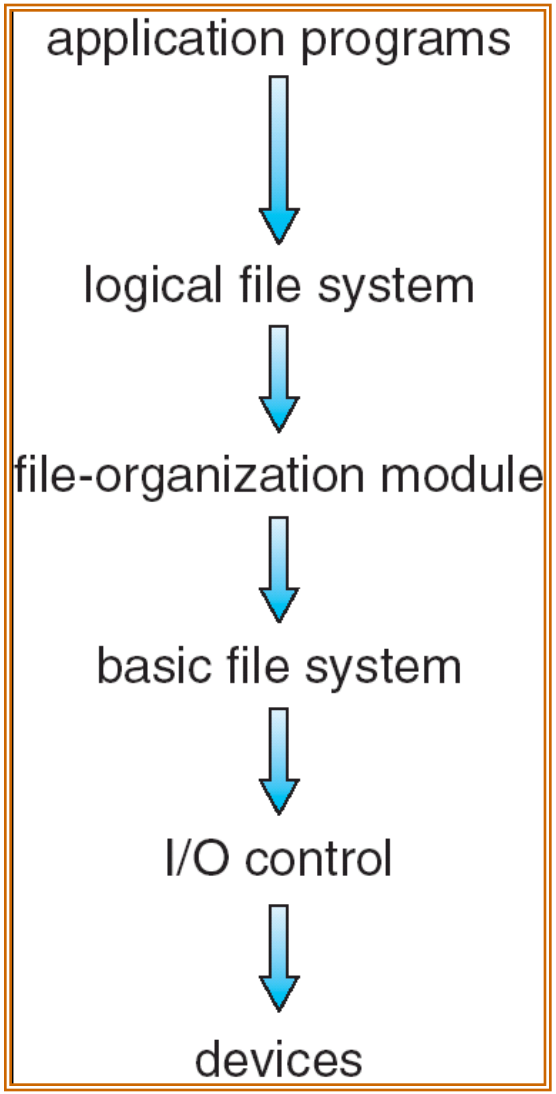
- File system organized into layers.
- ***File control block*** – storage structure consisting of information about a file.

Cont…

Definition:

- Set of system software's which are responsible to provide services to the users as per file accessibility . So , giving access to the users to access the file is the major responsibility of **File Management System.**
- **File Management system is concerned with various responsibilities:**
- Storage , Retrieval , Manipulation , validity etc.
- To manage all these responsibilities they have a **“File System Architecture”**

Layered File System



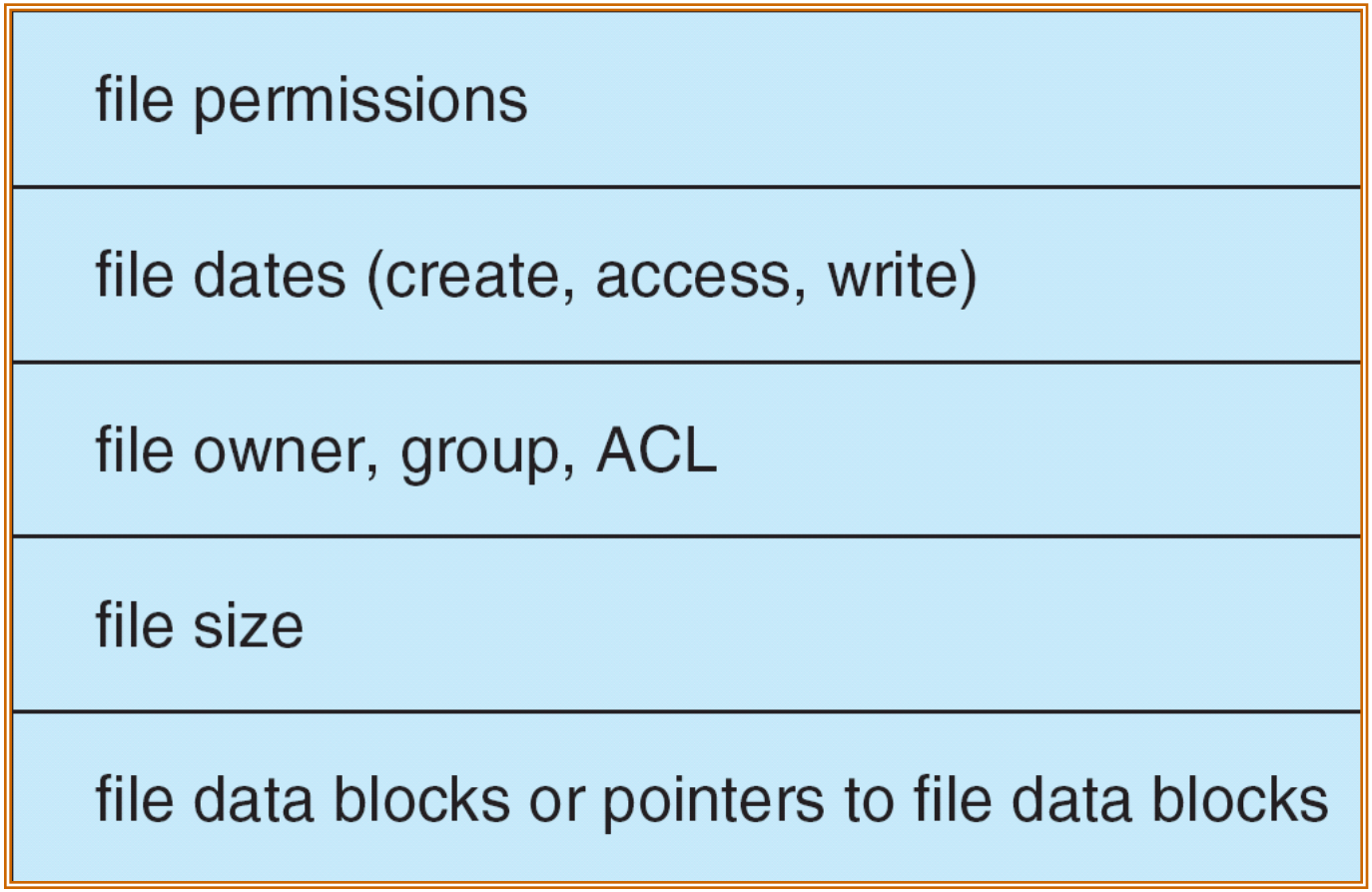
Cont..

- **The lowest layer is the devices , which consists of all hardware devices which are present inside the system.**
- **The second lowest level , the I/O controller consists of device drivers and the interrupt handlers to transfer information between the hard disk and the main memory . The device driver can be also thought of a translator.**
- **The third lowest level , the Basic File System will issue the generic commands to the appropriate device drivers to read and write physical bocks on the disck , it also manages the memory buffers and cache that holds various file systems, directories and data blocks.**

Cont..

- **The next layer is the File Organization Module knows about their logical blocks and physical blocks , by knowing the type of the file and location of the file the file organization module will translate the logical block into physical block .**
- **The next layer is the Logical File system will maintain the File Structure via File Control Block, which contains the complete information of a File , It is also responsible for Protection & security .**
- **The Highest level is the Application Program where the user is Interacting with the file.**

A Typical File Control Block



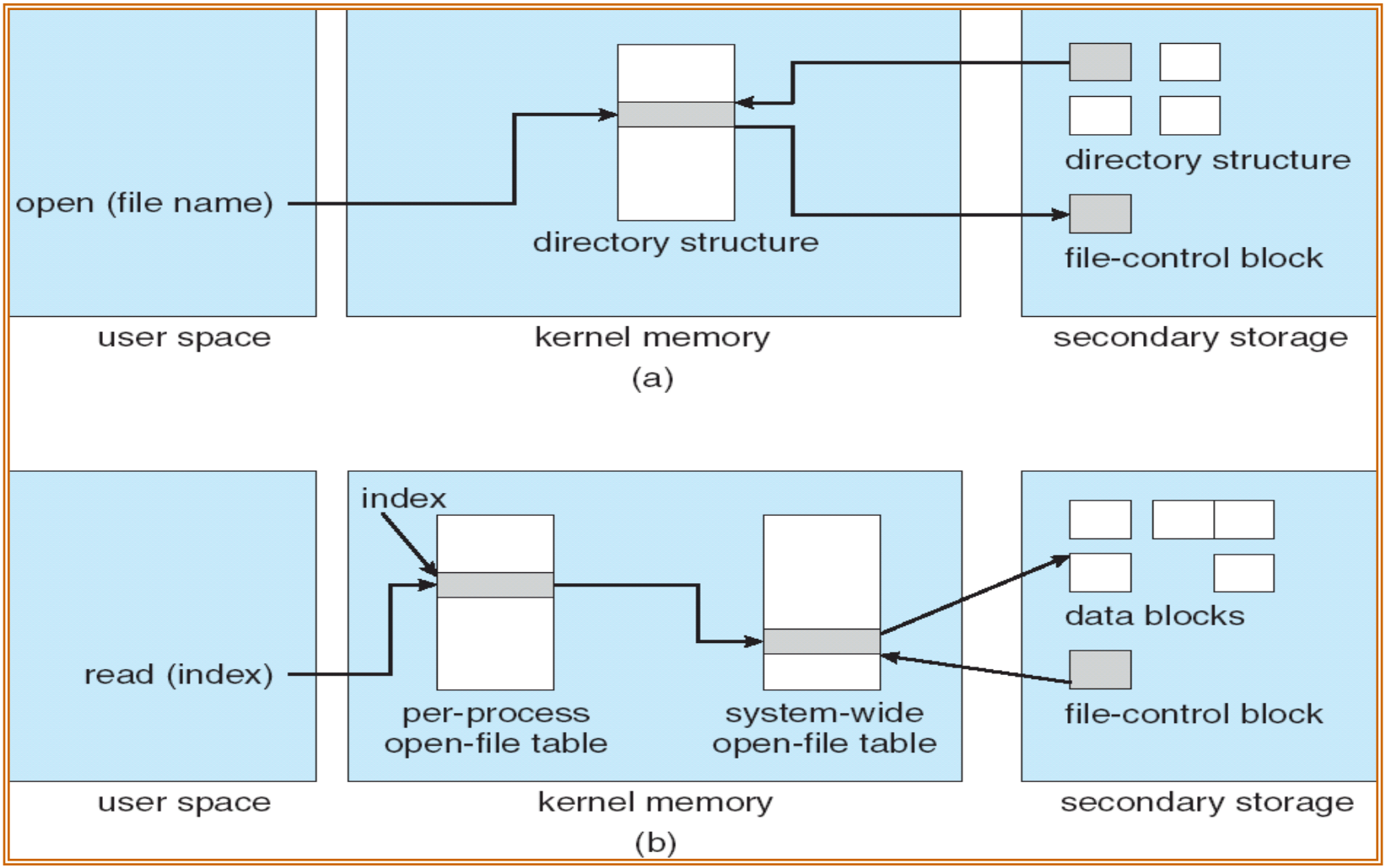
File System Implementation

- **On the Disk the File System may contain the Information of how to Boot an Operating system stored their , the total no of blocks , the directory structure and files.**
- **A Boot Control Block will contain the information needed by the system to boot an operating system from the volume.**
- **A Volume Control Block contains volume details such as the number of blocks in the partition ,the size of the blocks , free block count.**
- **Directory Structure is used to Organize a File.**
- **A Per-File FCB will contain the details about the File.**

In-Memory File System Structures

- In-Memory information is used for both File-System Management and Performance Improvement via Cache.
- An in-memory mount table contains information about each mounted volume.
- An in-memory directory –structure holds the directory information of recently accessed directory.
- System-wide open file table contains a copy of the FCB of each open file .
- The Pre-process open file-table contains a pointer to the appropriate entry in the system –wide open file table.
- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.

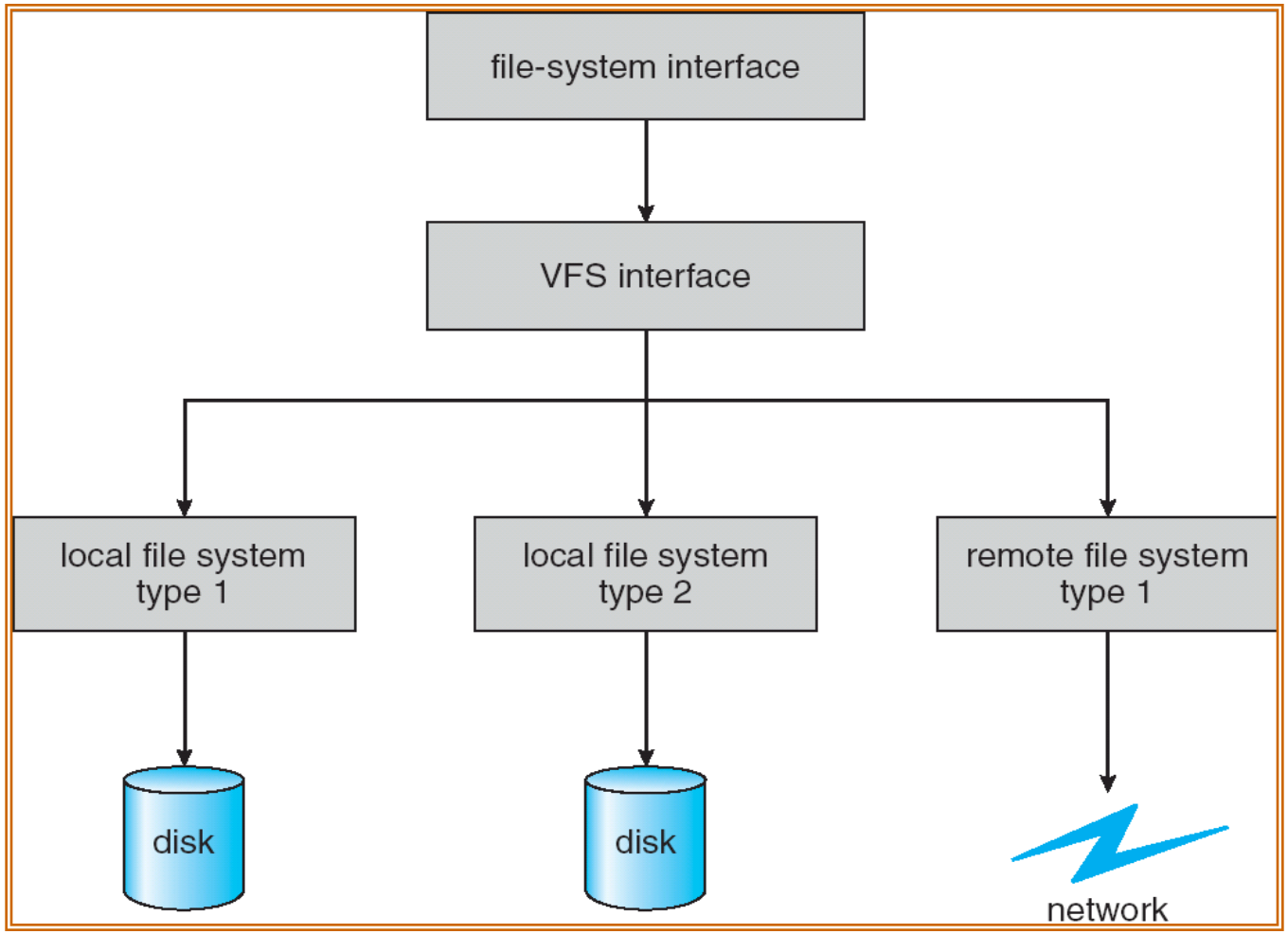
In-Memory File System Structures



Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of a Virtual File Systems



Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

Allocation Methods

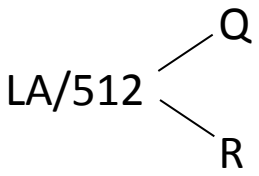
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- If these contiguous blocks you want to access then it follows sequential access methods , Direct access methods.

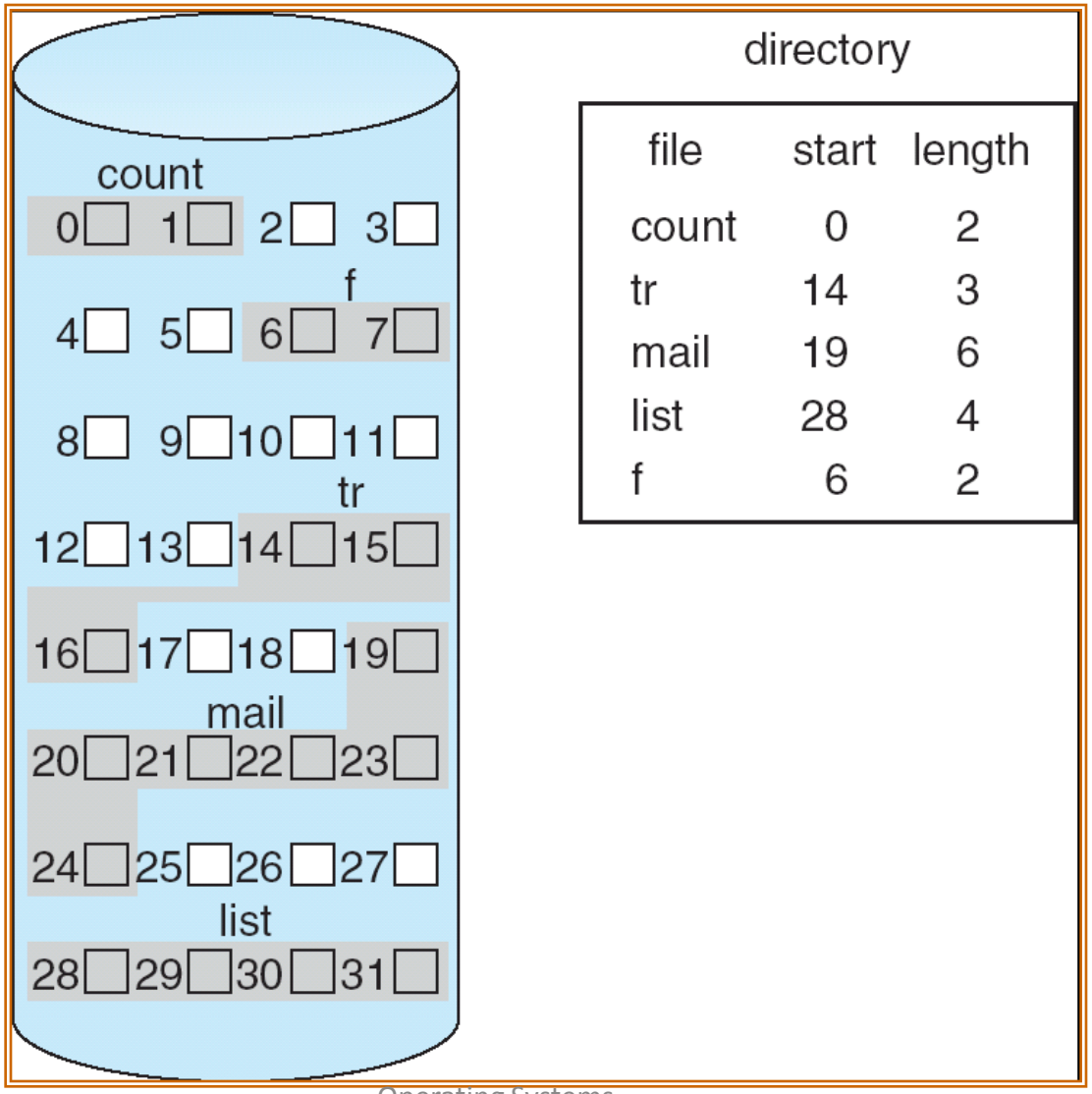
Disadvantages:

- dynamic storage-allocation problem.
- Finding space for a file , Knowing file size , External Fragmentation need for a compaction.
- Mapping from logical to physical.



  - Block to be accessed = ! + starting address
  - Displacement into block = R

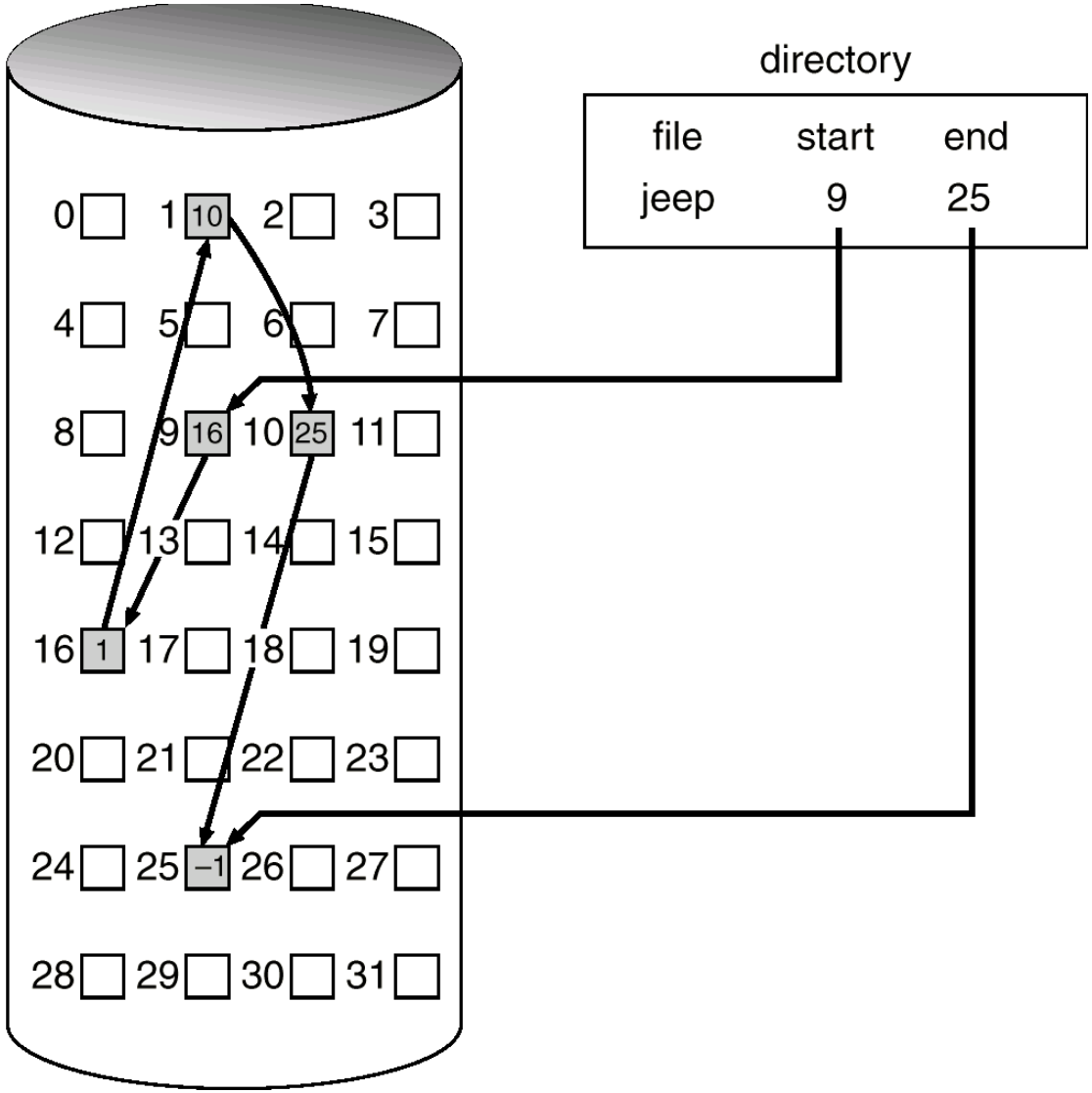
Contiguous Allocation of Disk Space



Linked Allocation

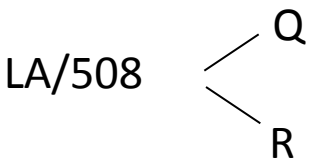
- Linked allocation solves all problems of contiguous allocation.
- With Linked allocation, each file is a linked list of disk blocks
- The disk blocks may be scattered anywhere on the disk
- The directory contains a pointer to the first and last blocks of the file
- Each block contains a pointer to the next block. These pointers are not made available to the user.

- Allocate as needed, link together; e.g., file starts at block 9



Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping : Let the LA be the Logical address & let the block size be 512 bytes.
- Each block uses its first 4 bytes to hold a pointer to next bock.



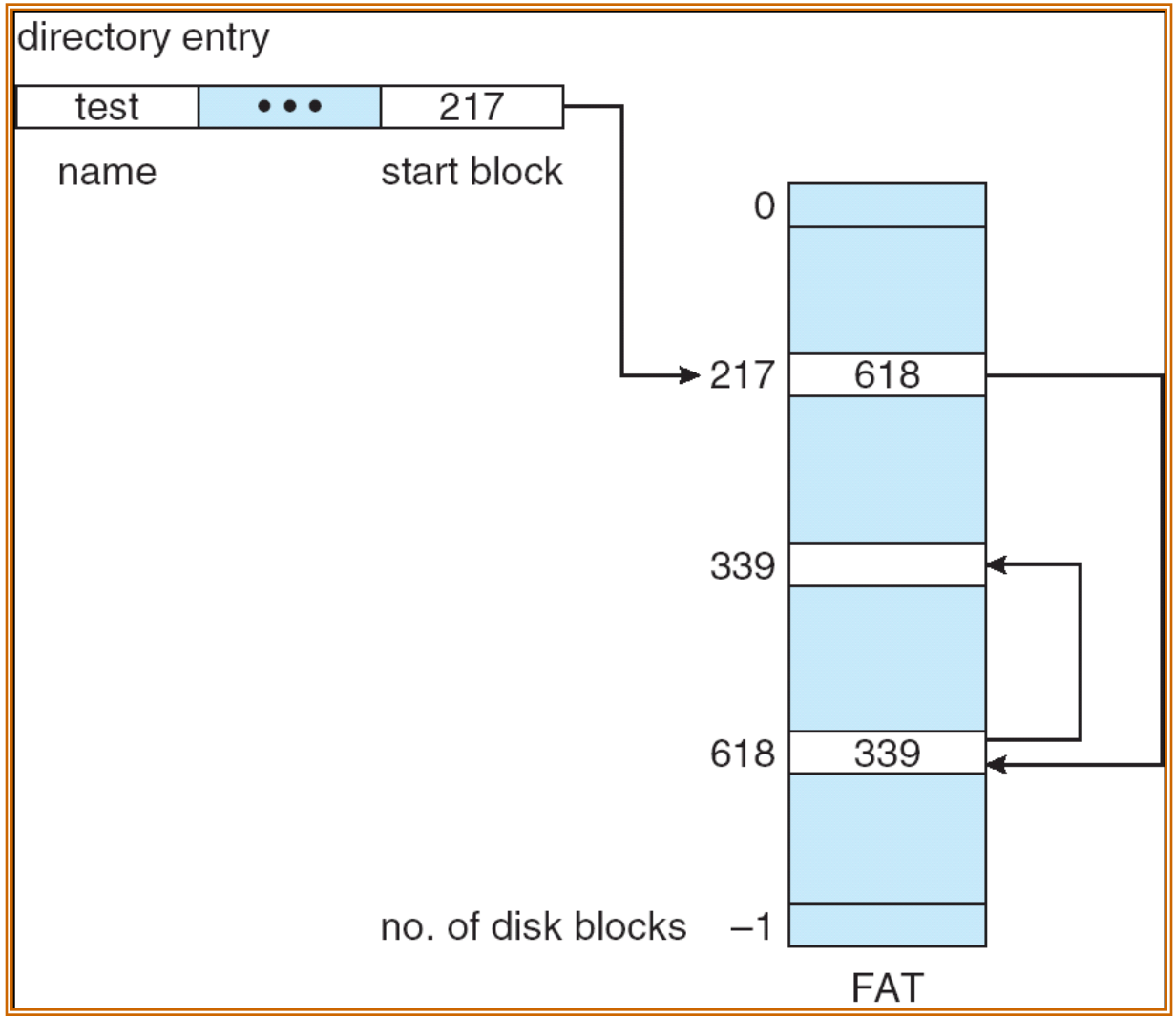
  - Block to be accessed is the Qth block in the linked chain of blocks representing the file.
  - Displacement into block = R + 1

- *File-allocation table (FAT)* – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation (Cont.)

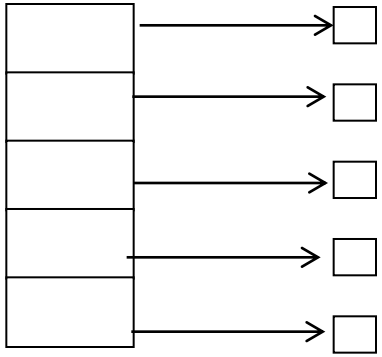
- An important variation on linked allocation is the use of a file-allocation table(FAT).
- This simple but efficient method of disk-space allocation is used by the MS-DOS and OS/2 operating system.
- A section of disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed by block number.
- The directory entry contains the block number of the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.

File-Allocation Table



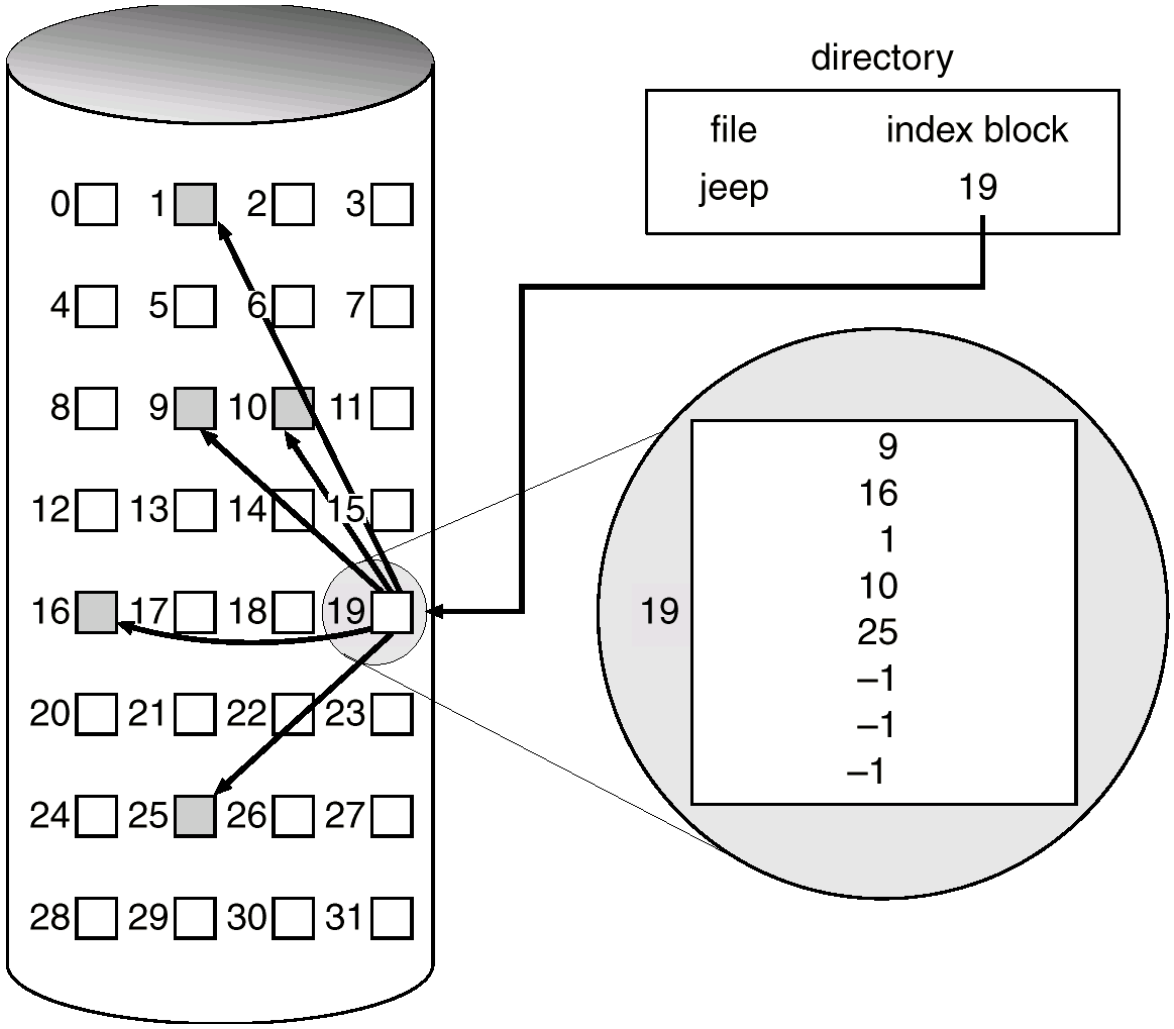
Indexed Allocation

- Brings all pointers together into the one location i.e, the *index block.*
- *E*ach file has its own index block, which is an array of disk-block addresses.
- Logical view.



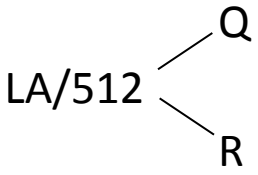
index table

Indexed Allocation of Disk Space



Indexed Allocation (Cont.)

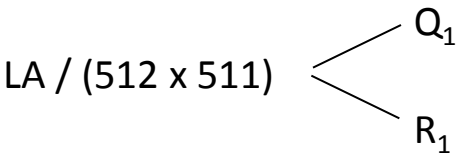
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.



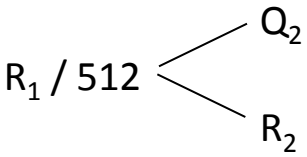
  - *Q* = displacement into index table
  - *R* = displacement into block

Indexed Allocation – Linked Mapping

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).



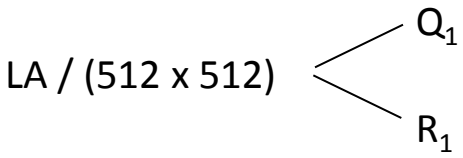
  - Q_1 = block of index table
  - R_1 is used as follows:



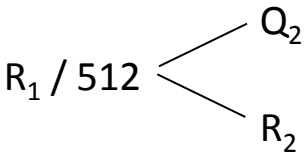
  - Q_2 = displacement into block of index table
  - R_2 displacement into block of file:

Indexed Allocation – Multilevel Indexed Mapping

- Two-level index (maximum file size is 512^3)

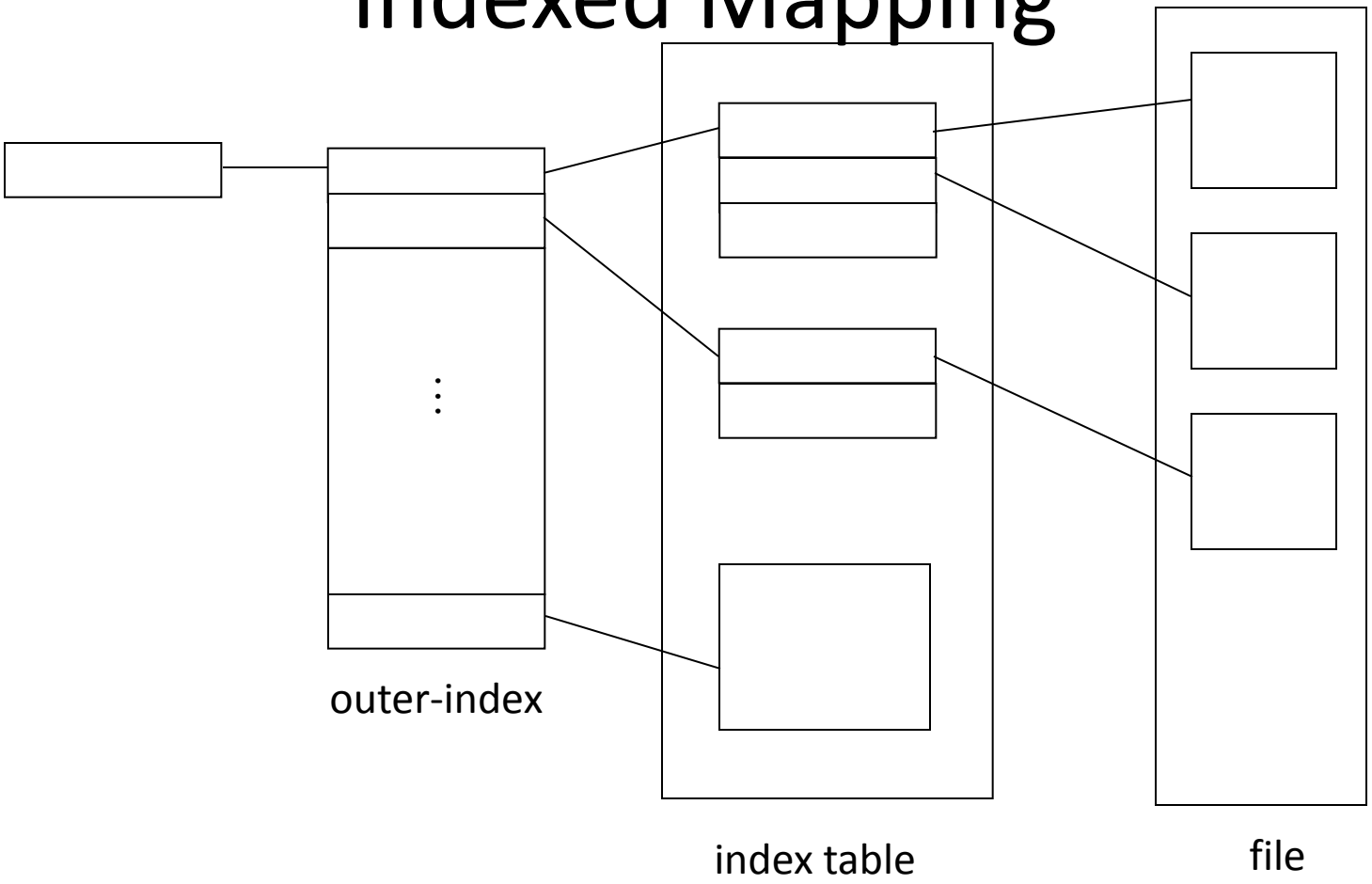


  - Q_1 = displacement into outer-index
  - R_1 is used as follows:

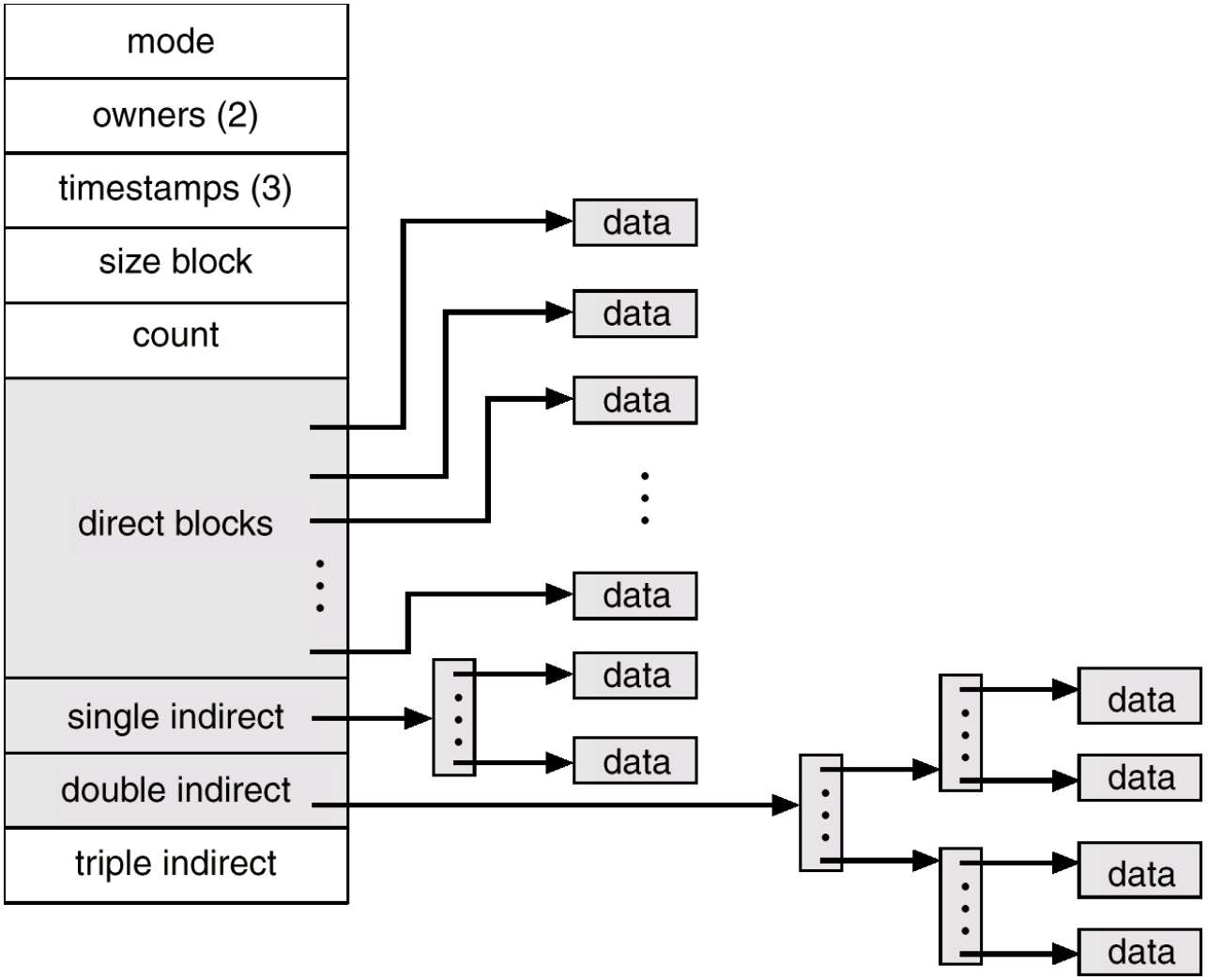


  - Q_2 = displacement into block of index table
  - R_2 displacement into block of file:

Indexed Allocation – Multilevel Indexed Mapping



Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- **Free-space list**: It is going to keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks—those are not allocated to some file or directory.
- To create a new file, we search the free space list for the required amount of space and allocate that space to the new file. Then this space is then removed from the free space list.
- When the file is deleted, its disk space is added to the free space list.
- The free-space list is implemented as a:

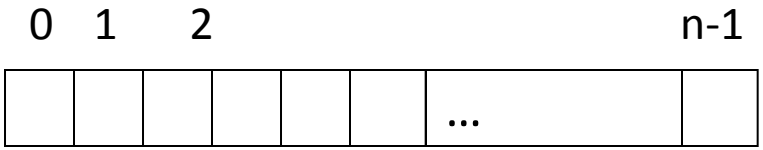
-----Bit Vector/Bit Map

------Linking

------Grouping

Free-Space Management

- Bit vector (*n* blocks)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

Free-Space Management (Cont.)

- Linked list (free list)
  - Cannot get contiguous space easily.
  - No need to traverse the entire list.(if # free blocks recorded)
  - No waste of space.
- Grouping

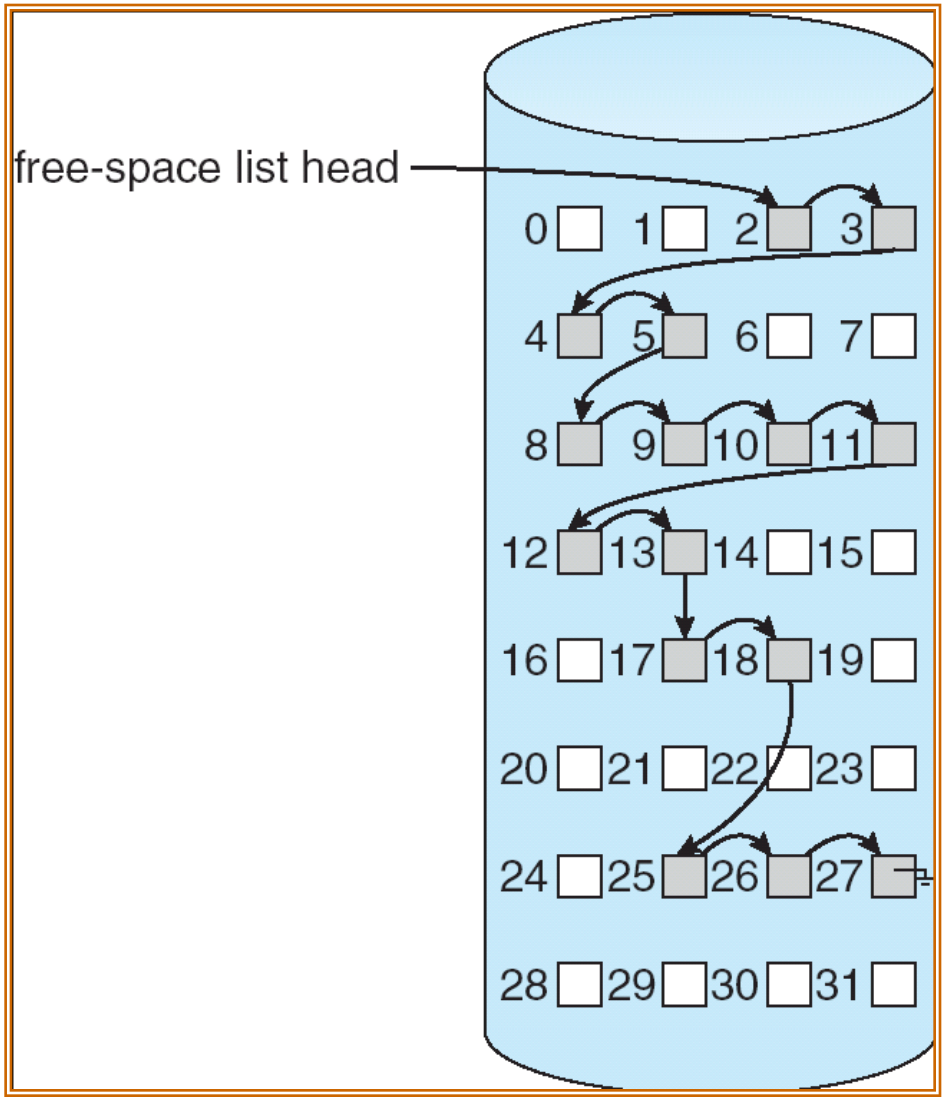
  Modifying linked list to store addresses of n free blocks in first free block. The first n-1 of these bocks are actually free.
- Counting

  Several contiguous blocks may be allocated or freed simultaneously , particularly when space is allocated with the contiguous-allocation algorithm or through clustering.

Directory Implementation

- Linear list of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location
  - fixed size

Linked Free-Space List on Disk



Secondary-Storage Structure

- Disk Structure
- Disk Scheduling
- Disk Management
- Swap-Space Management
- Disk Reliability
- Stable-Storage Implementation
- Tertiary Storage Devices
- Operating System Issues
- Performance Issues

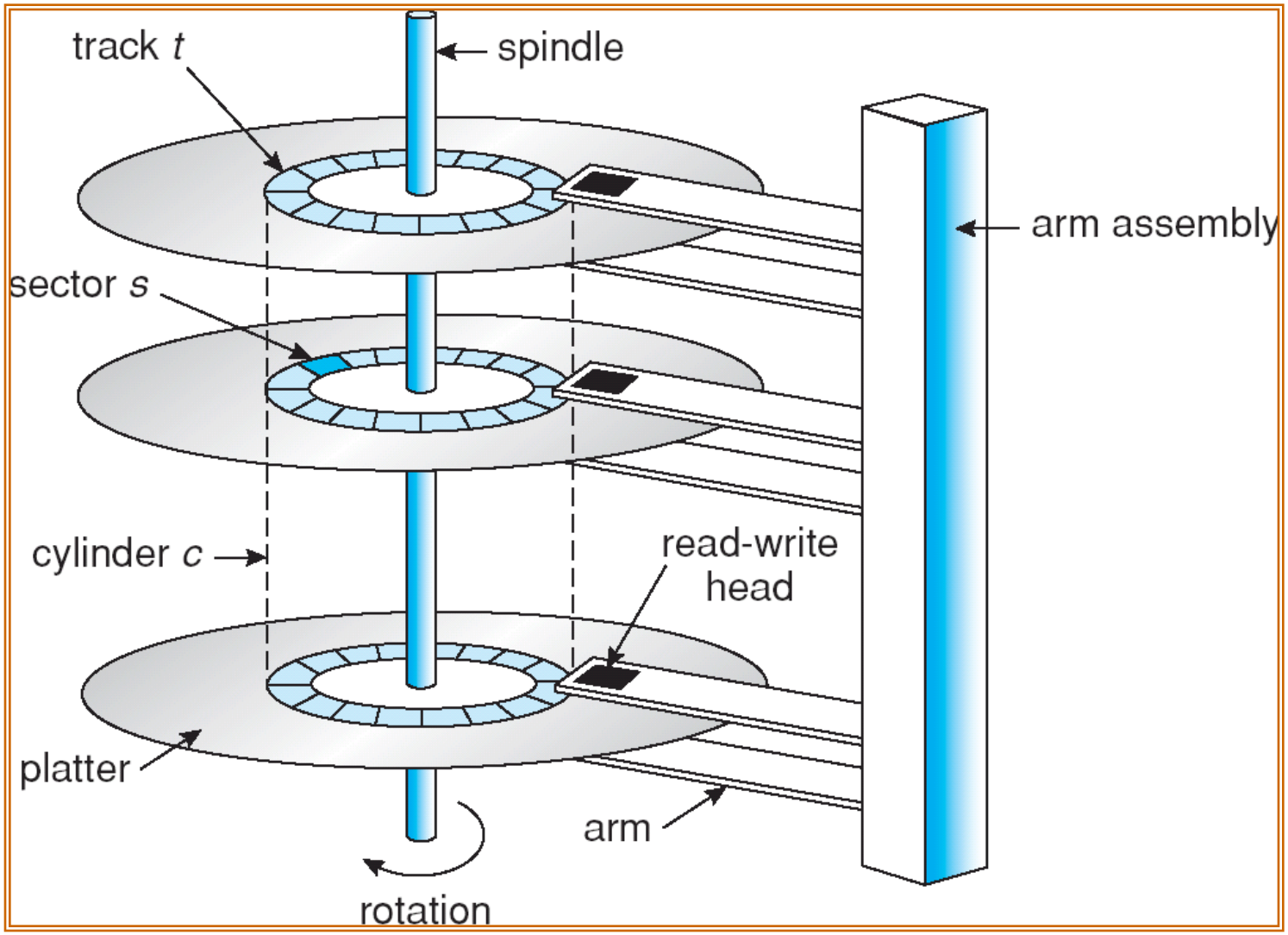
Overview of Mass-Storage Structure

- Magnetic disk stores the large amount of data permanently .
- Each Disk contains n no of Platters inside a Disk.
- Each Disk Platter has a Flat Circular Shape , Like a CD .
- Each disk platter surface is covered with magnetic flux .The data can be magnetically recorded on to the disk platter.
- The Read/Write head files are placed just above the platter .This heads are attached to the Disk Arm.
- Spindle is going to rotate the disk platter so the data present on the platter can be read by the Read/Write head files by changing the sector positions.
- The Platter is divided into Circular Tracks , again the track is divide in to sectors.
- Set of tracks that are at one arm position makes up a cylinder.
- In Hard Disk their might be thousand number of cylinders and each track may contain hundred no .of sectors.

Cont..

- whenever we are using the disk , the drive motor i.e Spindle is going to move at High Speed , which rotates at 60 to 200 times per second.
- Disk speed has two parts:
- Transfer Rate: The time it takes to move the Data from Hard Disk to Computer Memory (Main Memory).
- Random Access Time: Here to calculate the Random Access Time we need to consider the Seek Time & Rotational Latency.
- Seek Time: The Time taken For the Read/Write Head File to go the Desired Position of the sector to read the Data.
- Rotational Latency: The time taken to rotate the Spindle.
- Head Crash: If the head will make contact with the disk surface , the disk will sometimes get damages the magnetic surface , this is called as Head Crash.(Hard to recover the Head Crash).
- A Disk Drive is attached to a Computer by a set of wires called as an I/O bus.
- The Data transferring is happening with the help of Controllers called as Host Controller for the Computer , Disk Controller for the Drivers.

Moving-head Disk Mechanism



Overview of Mass Storage Structure (Cont.)

- Magnetic tape
  - Was early secondary-storage medium
  - Relatively permanent and holds large quantities of data
  - Access time slow
  - Random access ~1000 times slower than disk
  - Mainly used for backup, storage of infrequently-used data, transfer medium between systems
  - Kept in spool and wound or rewound past read-write head
  - Once data under head, transfer rates comparable to disk
  - 20-200GB typical storage
  - Common technologies are 4mm, 8mm, 19mm, LTO-2 and SDLT

Disk Structure

- The Logical Block , is the smallest unit of Transfer of data between Hard Disk & the Computer.
- The size of the logical Block will be usually 512 bytes or some may be 1024 bytes.
- The logical locks are going to be moved on to the sectors sequentially.
- Sector 0 is the First Sector of the First Track on the outermost Cylinders.

Mapping From Logical to Physical:

First it Travels to the Track then rest of the tracks with in the same cylinders and the cylinders will be travelling from outermost to the inner most .

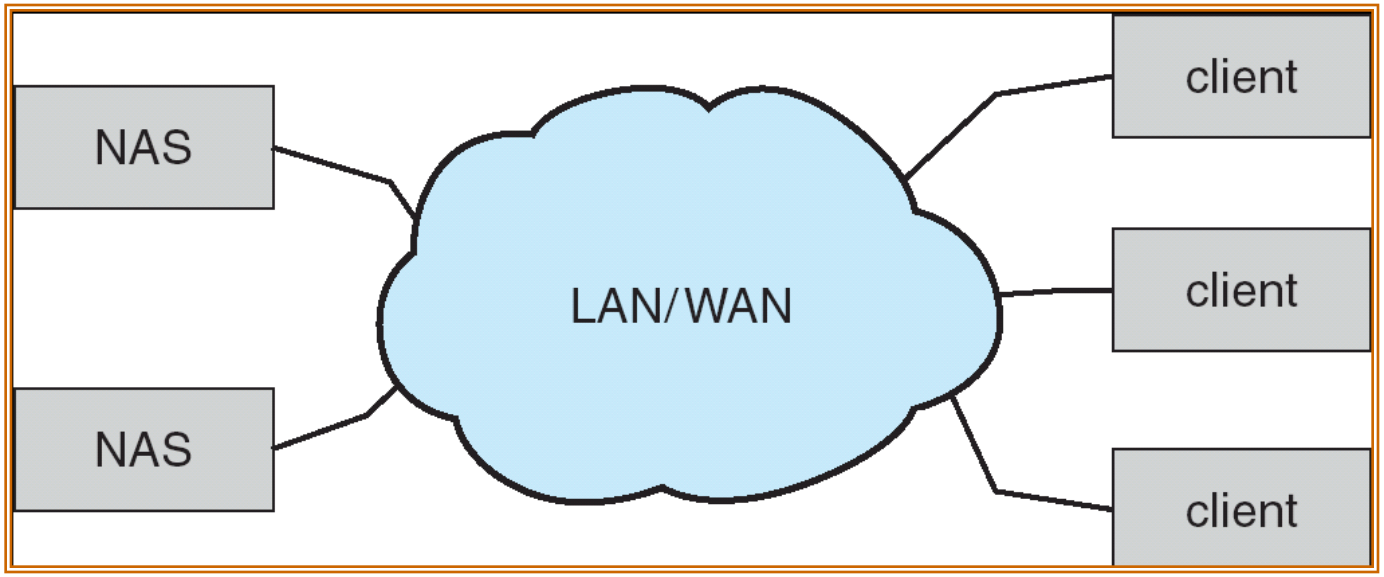
Whenever it want to read the data the logical block no into an Old style address. i.e The Disk address will be Cylinder no , Track no & the Sector no within that track.

Disk Attachment

- Host-attached storage accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
  - Each target can have up to 8 **logical units** (disks attached to device controller
- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of **storage area networks** (**SAN**s) in which many hosts attach to many storage units
  - Can be **arbitrated loop** (**FC-AL**) of 126 devices

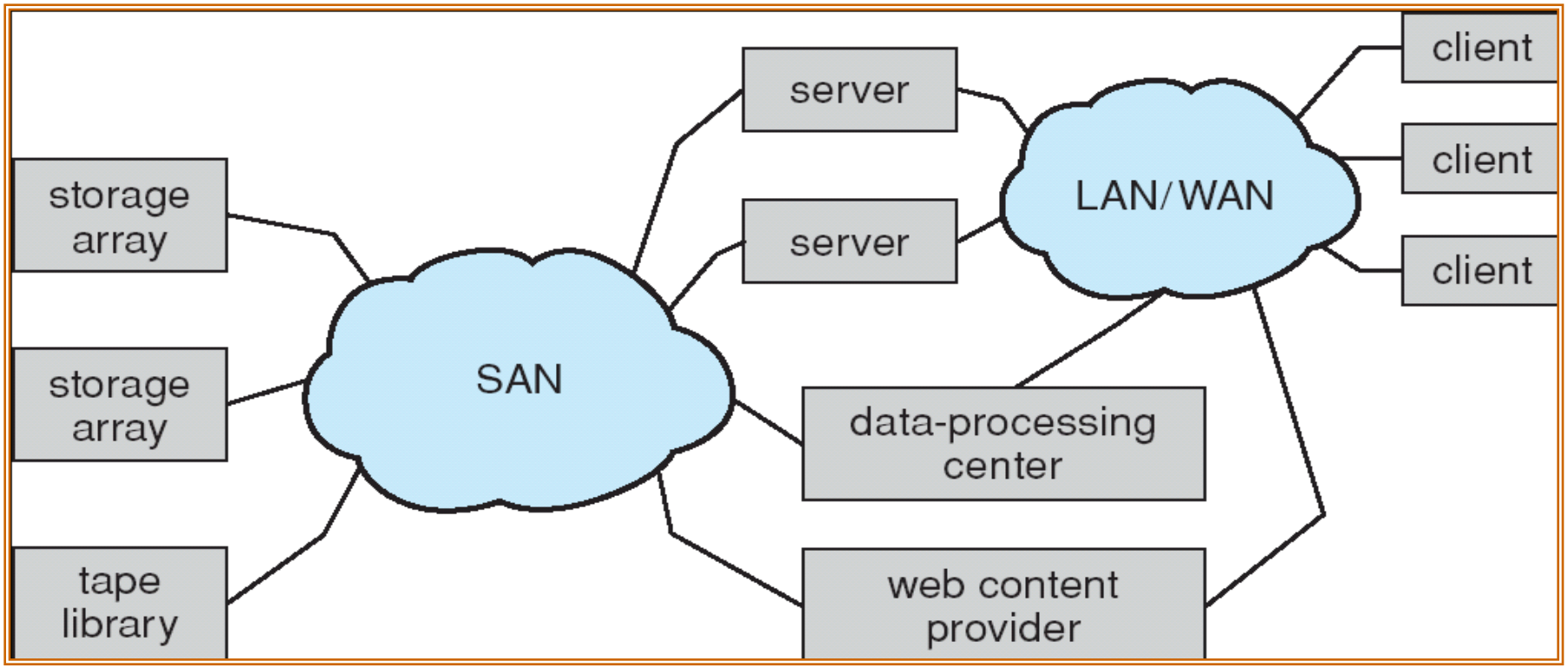
Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage
- New iSCSI protocol uses IP network to carry the SCSI protocol



Storage Area Network

- Common in large storage environments (and becoming more common)
- Multiple hosts attached to multiple storage arrays - flexible



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time $\approx$ seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling (Cont.)

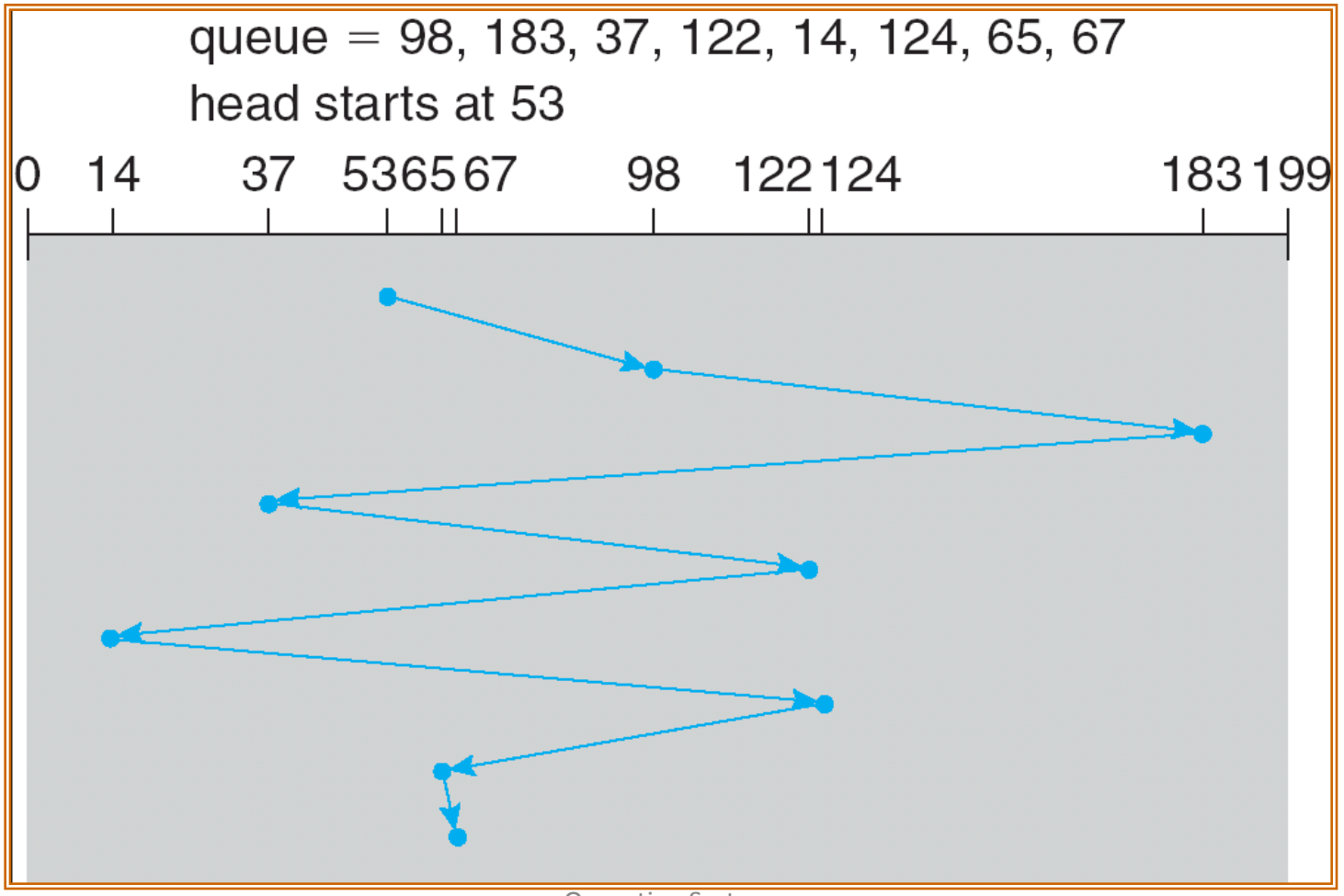
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

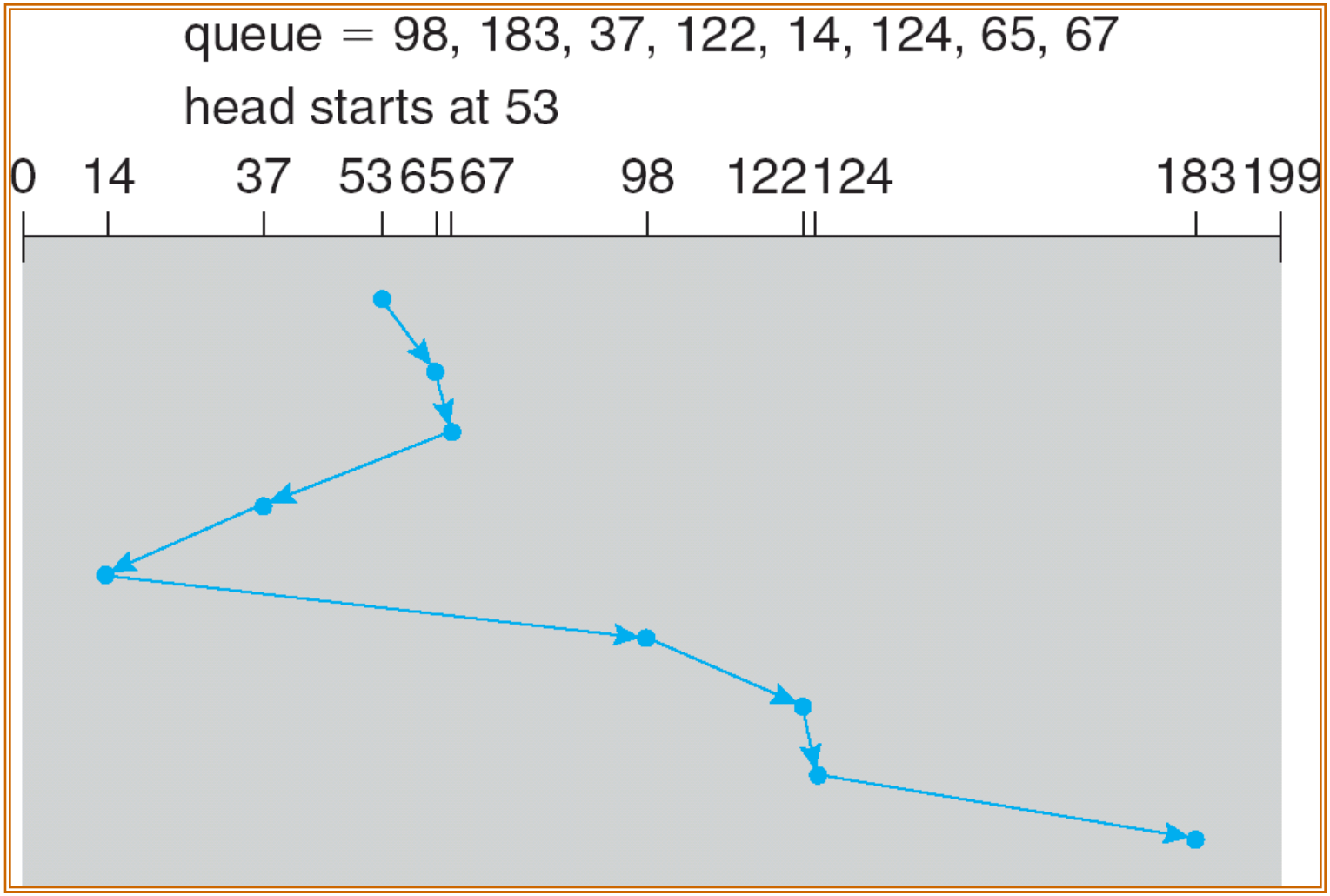
Illustration shows total head movement of 640 cylinders.



SSTF

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

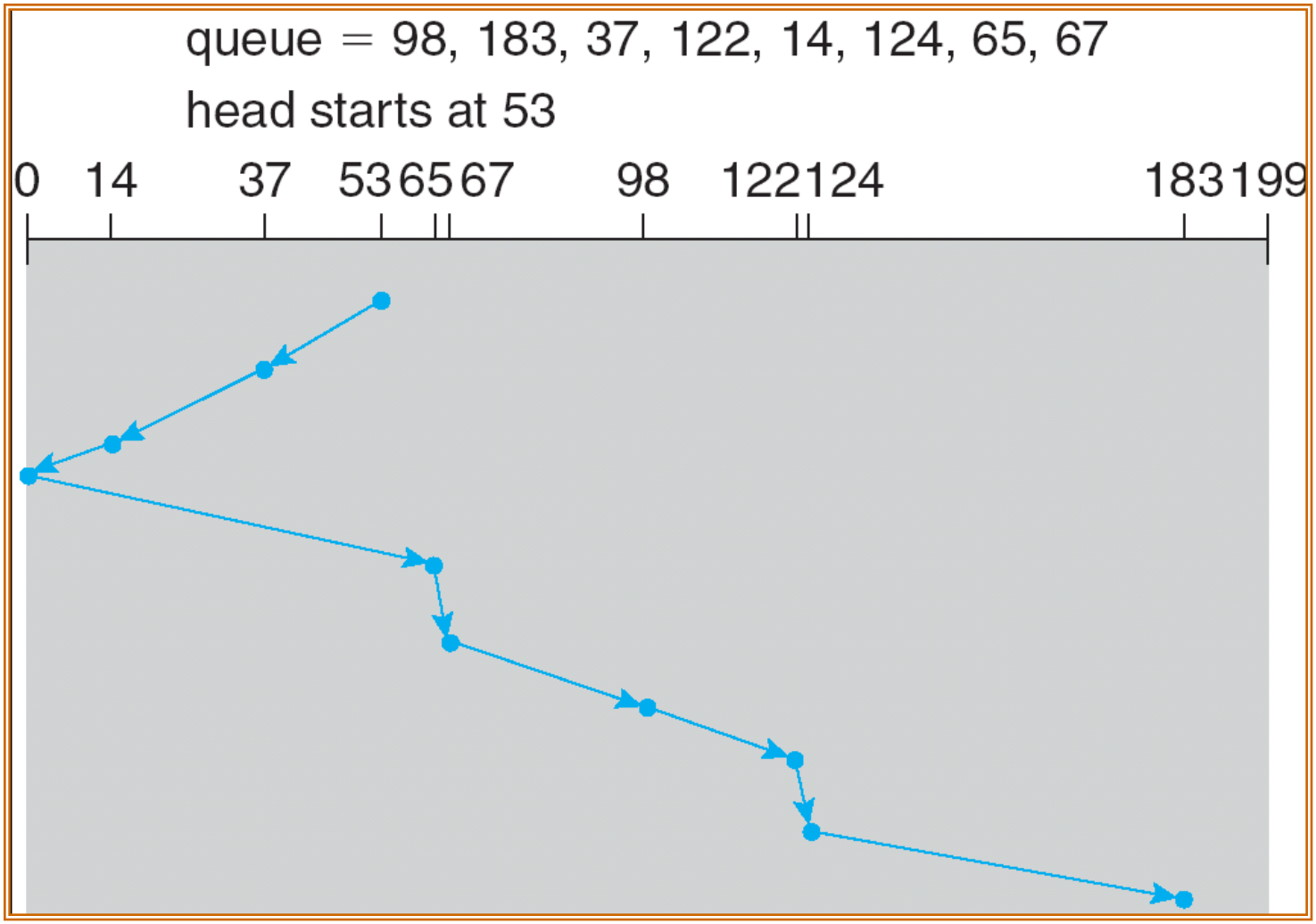
SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- Illustration shows total head movement of 208 cylinders.

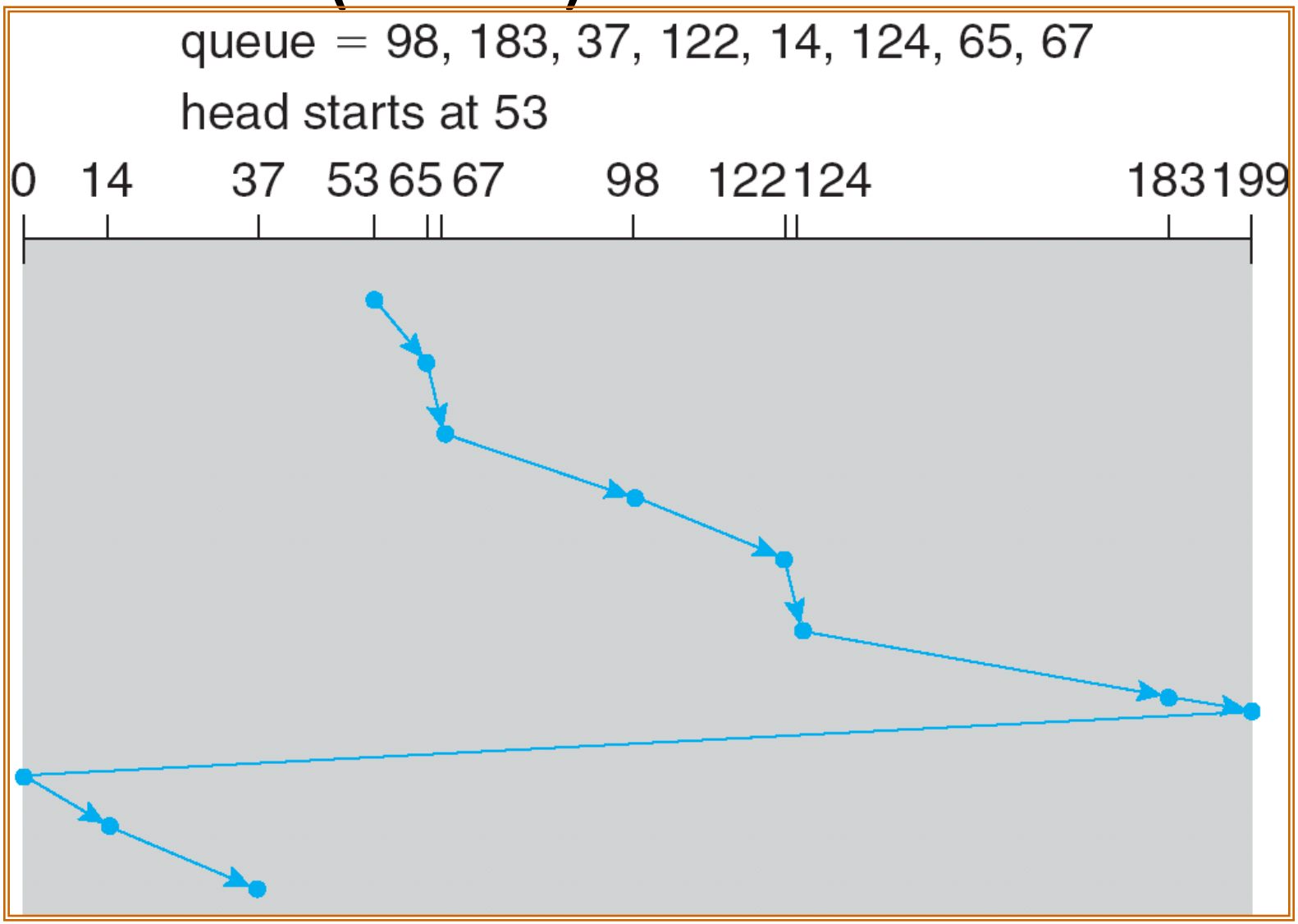
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

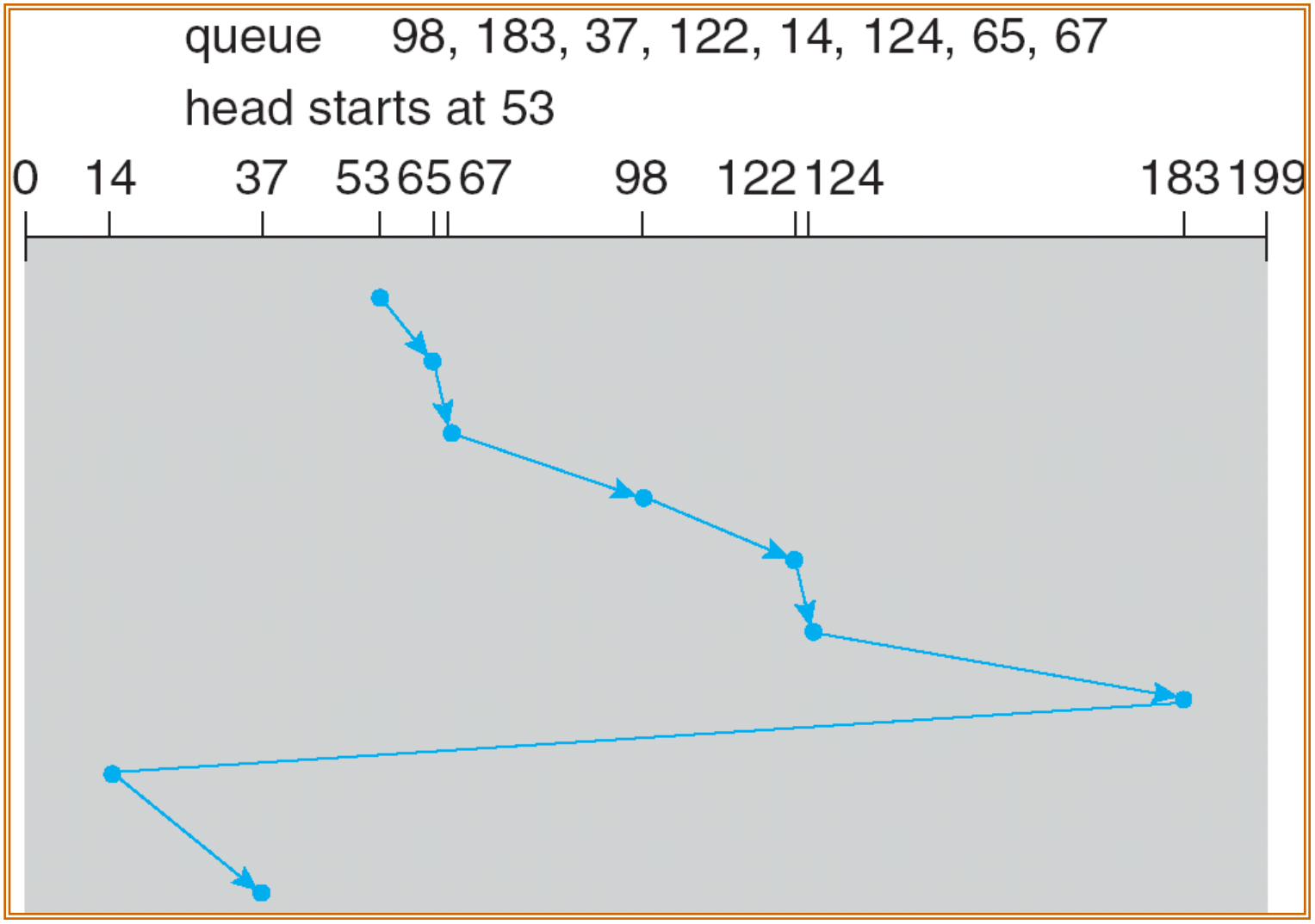
C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

C-LOOK (Cont.)


queue 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

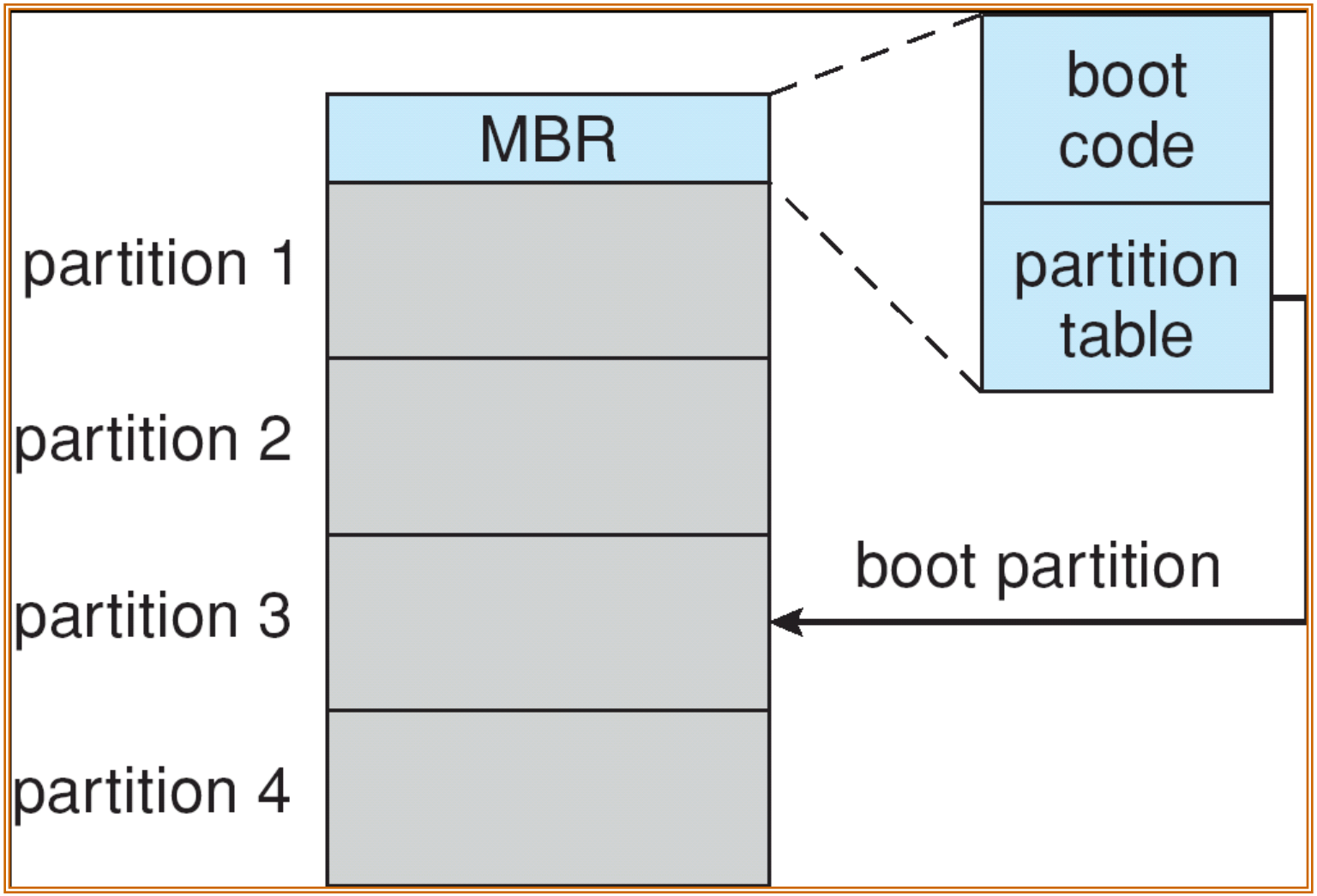
Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

Disk Management

- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - *Partition* the disk into one or more groups of cylinders.
  - *Logical formatting* or "making a file system".
- Boot block initializes system.
  - The bootstrap is stored in ROM.
  - *Bootstrap loader* program.
- Methods such as *sector sparing* used to handle bad blocks.

Booting from a Disk in Windows 2000



Swap-Space Management

- Moving entire process between disk and memory is called as swapping.
- Swapping occurs when the amount of physical memory is critically low and process are moved from memory to swap space to free available memory.
- Swap space management is another low level task of OS.
- This is designed to provide the best throughput for the virtual memory system.

➢ How swap is used

➢ swap space located on disk

➢ swap space managed

Swap Space Used:

- It is used in different ways based on OS , depending on the memory management algorithms.

1. System implementing swapping may use swap space to hold an entire process image including code and data segment.
2. System implementing paging may use swap space to hold few pages that are pushed out of memory.

- Swap space should be over estimated but not under estimated , if it runs out of space it may force to abort process or may crash entirely.
- Overestimation just waste memory where there is no much harm.
- Some have multiple swap spaces usually put on separate disks so that the load placed on the I/O separated by paging and swapping can be spread over the system's I/O drive.

Swap Space loaded on disk:

Swap space can be loaded in two places

1. **It can be carved out of the normal file system.**
2. **Separate disk portion.**

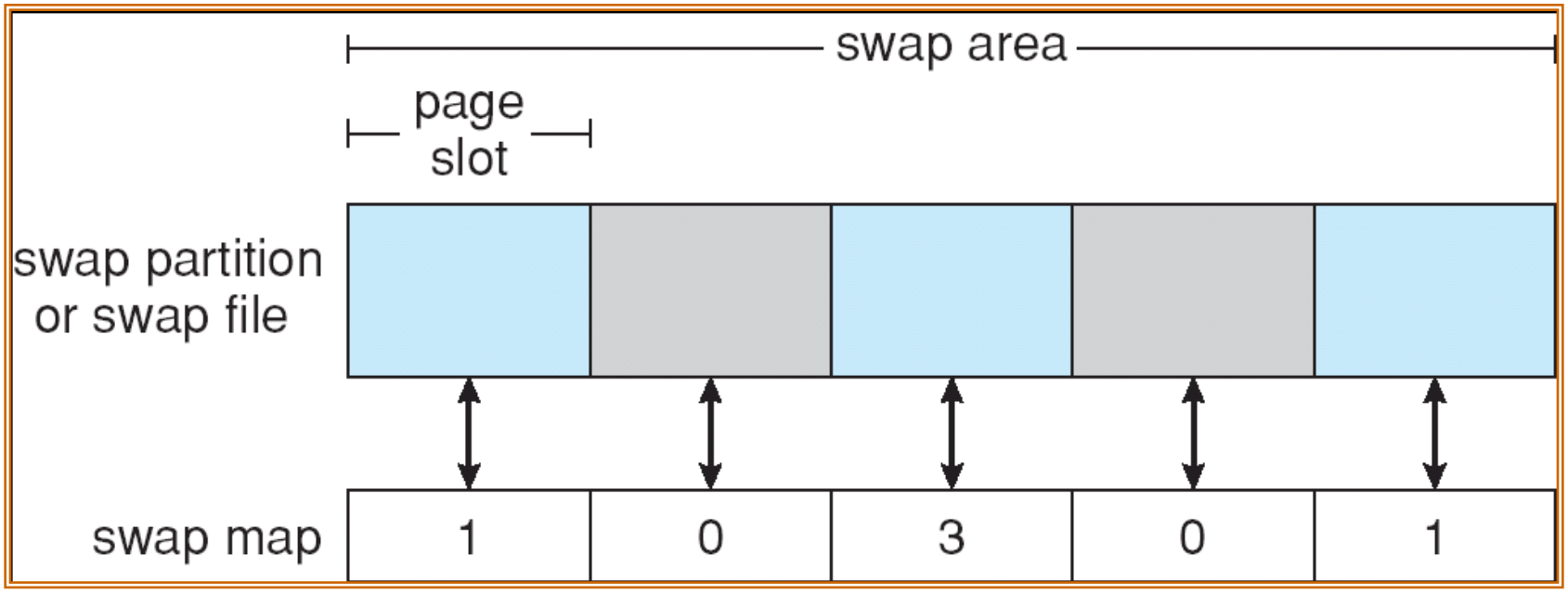
<u>It can be carved out of the normal file system:</u>

- If it is a normal simple file with in the file system , normal file system routine can be used to create it , name it & allocate the space.
- Easy to implement & inefficient.
- External fragmentation increases.

<u>Separate disk portion:</u>

- Create a separate raw partition separate swap space storage manager is used to allocate/ deallocate the blocks from the raw partition.
- Optimize the speed rather storage efficiency.
- Internal Fragmentation increases.

Data Structures for Swapping on Linux Systems



RAID Structure

- Suppose if there is any Failure in your Hard disk , then we need to retrieve the information.
- The retrival of the information is possible through RAID.
- Disk Drives have continued to get smaller and cheaper , so it is now economically feasible to attach many disks to a computer system.
- RAID – multiple disk drives provides reliability via redundancy.
- Raids are used to address the performance and reliability.
- RAID stands for Redundant Array of Independent Disks.
- RAID is arranged into six different levels.
- The following Figure shows RAID levels , in which P indicates error-correcting bits & C-indicates Second copy of the data.

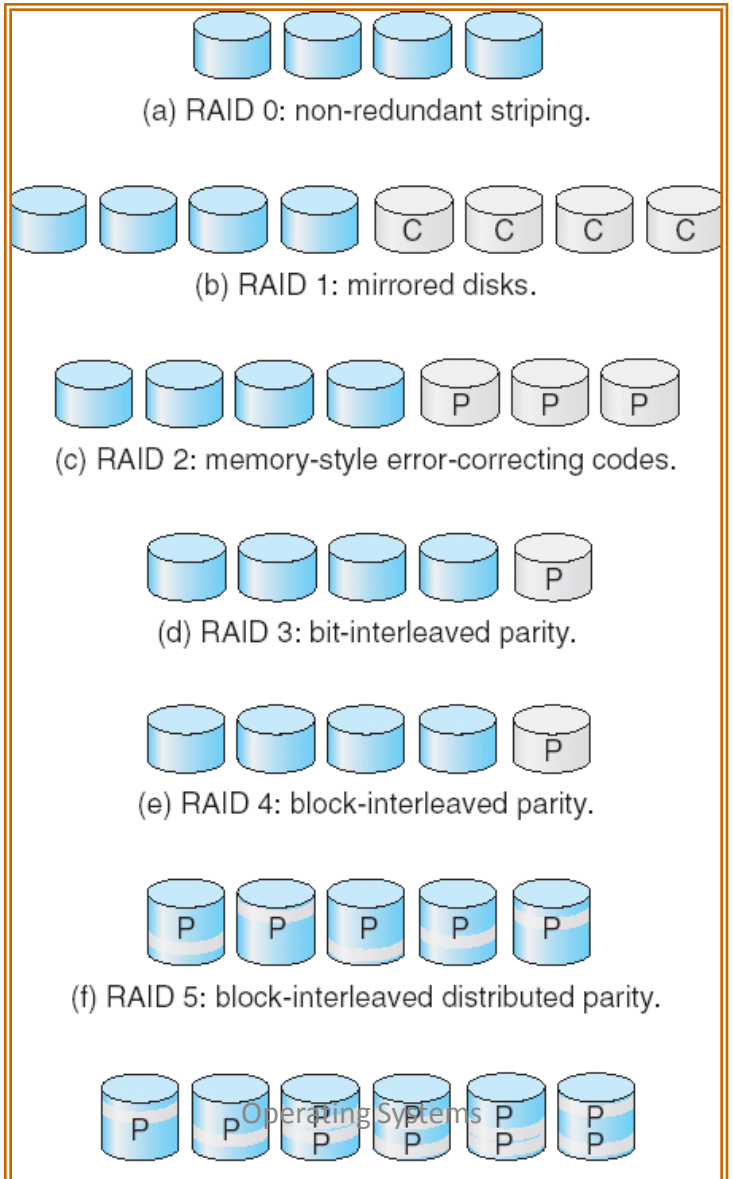
RAID (cont)

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively.
- RAID-0:It Refers to disk arrays with stripping at the level of blocks but without any redundancy .
- RAID-1: It refers to disk mirroring.
- RAID-2: It is also known as memory style-error correcting code. ,memory systems have long detected certain errors by using parity bits.
- RAID-3:It is known as bit-interleaved parity organization. improves on level-2 by taking into account unlike memory systems, disk controller can detect whether a sector has been read correctly , so a single parity bit can be used for both error correction and detection.
  Adv:1.Storage overhead has reduced.
  2. Requires fewer I/O per second.

RAID (cont)

- RAID-4:It is known as block-interleaved parity organization uses block level stripping .
- RAID-5: It is known as Block-Interleaved Distributed Parity differs from spreading data and parity among all N+1 disks , rather than storing data in N disks and parity in one Disk.
- Raid levels 0+1 and 1+0: Raid Levels 0+1 refers to the combination of RAID level 0 and RAID level 1 . With RAID level -0 it provides performance and with RAID level 1 it gives reliability.

RAID Levels



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

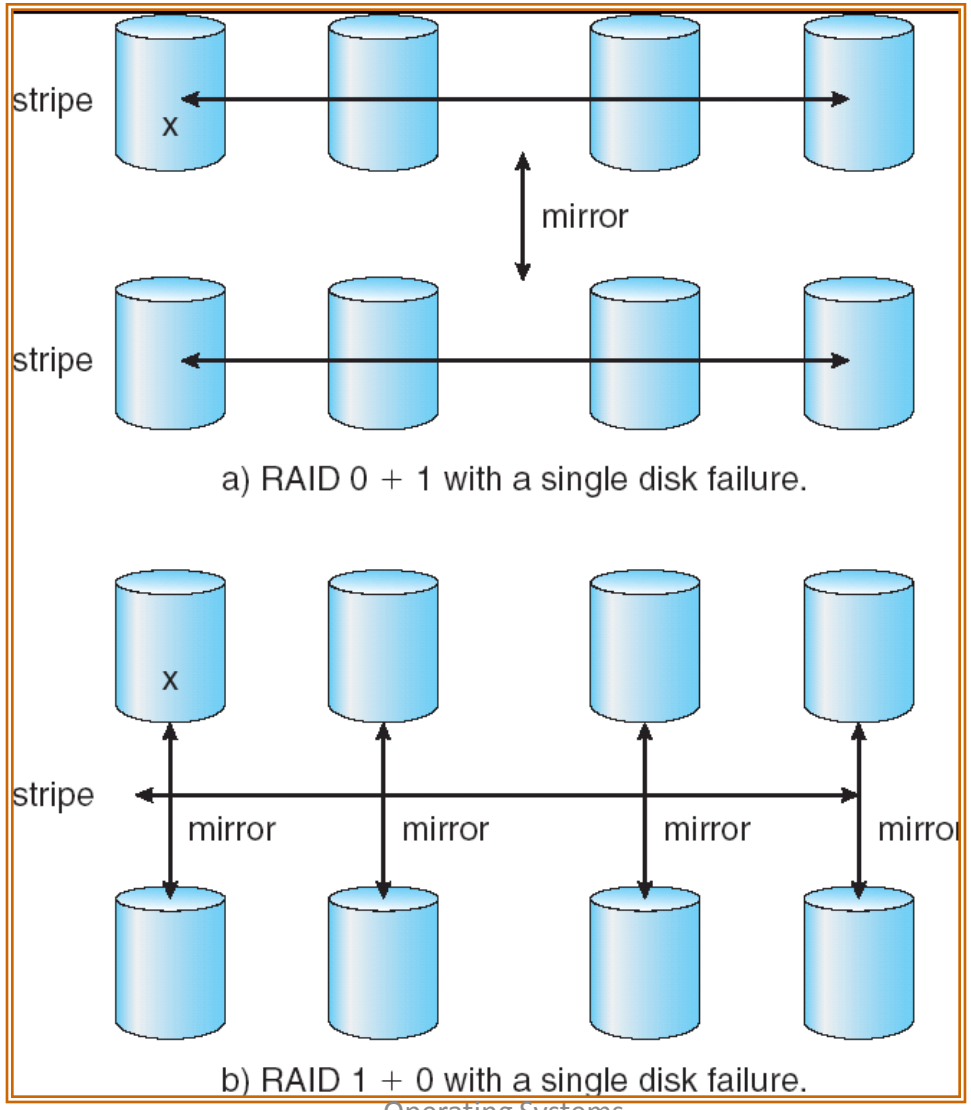
(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

RAID (0 + 1) and (1 + 0)



UNIT-V
Syllabus

Deadlocks – System Model, Deadlock Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection and Recovery from Deadlock.

Protection – System Protection, Goals of Protection, Principles of Protection, Domain of Protection, Access Matrix, Implementation of Access Matrix, Access Control, Revocation of Access Rights, Capability-Based Systems, Language-Based Protection.

Overview

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

- Operating system uses two sets of techniques to computer threats to information namely:

1. Protection
2. Security

Protection: It involves guarding a user’s data programs against interference by other authorized users of the system.

Security: It involves guarding a users' data programs against interference by external entities. E.g.: unauthorized persons

Protection

- It refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- Protection is required against the shared memory , logical space etc.
- The most obvious is the need to prevent mischievous, intentional violation of an access restriction by a user.

Goals of Protection

- To ensure that each shared resource is used only in accordance with the system policies, which may be set either by system designer or by system administrator.

- To ensure that errant programs cause the minimal amount of damage possible.

Goals of Protection

- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Goals of Protection

- The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use.
- These policies can be established in a variety of ways. Some are fixed in the design of the system, while others are formulated by the management of a system.
- Policies for resource use may vary by application, and they may change over time.

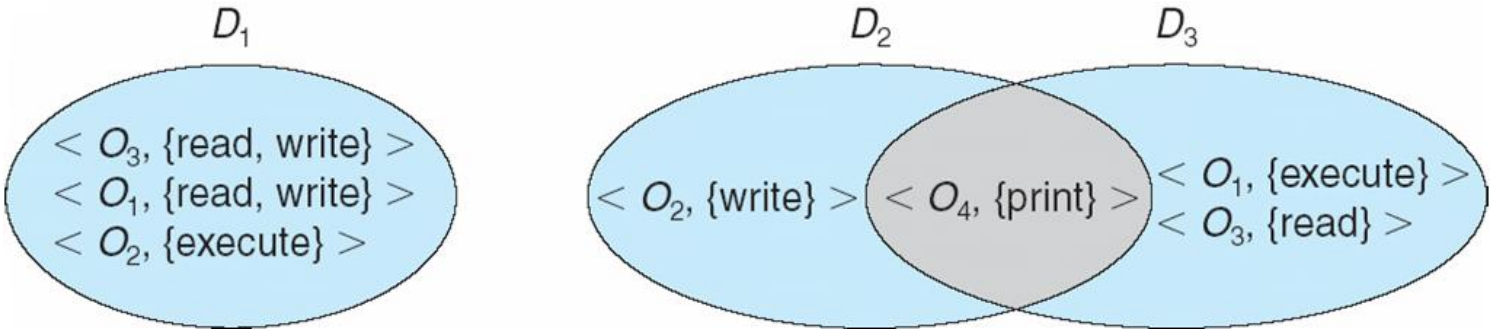
Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation(RBAC)**
- Must consider "grain" aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC
- Domain can be user, process, procedure

Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object

- Domain = set of access-rights



Domains

- A domain can be realized in a variety of ways:
- Each *user may be a domain.*
  - *The set of objects that can be* accessed depends on the identity of the user.
  - Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
- Each *process may be a domain.*
  - *The set of objects that can be* accessed depends on the identity of the process.
  - Domain switching occurs when one process sends a message to another process and then waits for a response.
- Each *procedure may be a domain.*
  - *The set of objects that can be* accessed corresponds to the local variables defined within the procedure.
  - Domain switching occurs when a procedure call is made.

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
    - Each file has associated with it a domain bit (setuid bit)
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed
    - When execution completes user-id is reset
- Domain switch accomplished via passwords
    - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
    - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- When a user creates a new object ,a column is added to the access matrix
- *Access(i, j)* is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

Access Matrix

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

- The access matrix provides an appropriate mechanism for defining and implementing strict control for both the static and dynamic association between processes and domains.
- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).
- Processes should be able to switch from one domain to another.
- A process executing in domain D2 can switch to domain D3 or to domain D4.

Access Matrix of Figure A with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Access Matrix with *Copy* Rights

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix with *Copy* Rights

- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right.
- The *copy right allows the copying of the access right only within the column* (that is, for the object) for which the right is defined.
- A process executing in domain *D2 can copy the read operation into any* entry associated with file F2.

Access Matrix with *Copy* Rights

- A right is copied from access(i, j) to access(k,j); it is then removed from access(i,j). This action is a *transfer of a right, rather than a copy.*
- Propagation of the *copy right may be limited. That is, when the right R* is copied from access(i,j) to access(k,j), only the right R (not R*)* is created.
- A process executing in domain D_k *cannot further copy the* right *R.*

Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix With *Owner* Rights

- If access(i,j) includes the *owner right, then a process executing in domain D_i, can add and remove* any right in any entry in column j.
- For example, in Figure , domain D_i is the owner of F_1, and thus can add and delete any valid right in column F_1.

Access Matrix With *Control* Rights

- The *copy and owner rights allow a process to change the entries in a column.*

- A mechanism is also needed to change the entries in a row.

- The *control right* is applicable only to domain objects. If access(i,j) includes the *control right*, then a process executing in domain *Di can remove any access right from* row j.
- For example, suppose that, we include the *control right in* access(D2, D4).
- Then, a process executing in domain D2 could modify domain D4.

Modified Access Matrix of Figure B

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- Global table
  - Store ordered triples < *domain*, *object*, *rights-set* > in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Additional I/O is required

Implementation of Access Matrix

- Access lists for objects(ACL)
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs< *domain*, *rights-set* > defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access

Implementation of Access Matrix

- Capability Lists for Domains
  - Instead of object-based, list is domain based
  - A capability list for a domain is a list of objects together with the operations allowed on those objects.

Implementation of Access Matrix

- Lock Key Mechanism
  - Compromise between access lists and capability lists
  - Each object has a list of unique bit patterns, called locks.
  - Similarly, each domain has a list of unique bit patterns, called keys.
  - A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.

ACL and Capability List

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

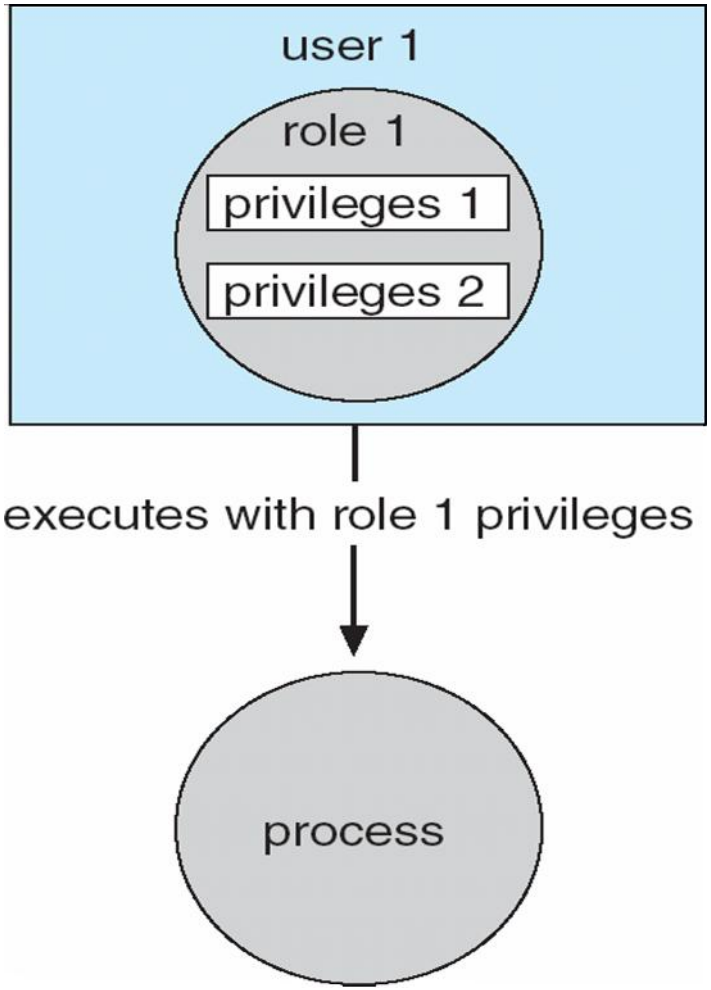
- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file

Access Control

- Protection can be applied to non-file resources

- Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - *Privilege* is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned *roles* granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix

Role-based Access Control in Solaris 10



Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent
- **Access List** – Delete access rights from access list
  - Simple – search access list and remove entry
  - Immediate, general or selective, total or partial, permanent or temporary
- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - Reacquisition – periodic delete, with require and denial if revoked
  - Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
  - Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - Keys – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

What is capability

- Suppose we design a computer system so that in order to access an object, a program must have a special token.
- This token **designates an object and gives the program the authority to perform a specific set of actions (such as reading or writing) on that object**.
- Such a token is known as a *capability*

What is capability

- Capabilities can be delegated.
- Capabilities can be copied.

Capability-Based Systems-Hydra

- Fixed set of access rights known to and interpreted by the system
  - i.e. read, write, or execute each memory segment
  - User can declare other **auxiliary rights** and register those with protection system
  - Accessing process must hold capability and know name of operation
  - **Rights amplification** allowed by trustworthy procedures for a specific type
    - Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights
    - Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities
    - Solves the *problem of mutually suspicious subsystems*
  - Includes library of prewritten security routines

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources

- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable

- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM
- The protection domain indicates what operations the class can (and cannot) perform
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

Overview

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Implementing Security Defenses
- Firewalling to Protect Systems and Networks
- Computer-Security Classifications
- An Example: Windows

The Security Problem

- System **secure** if resources used and accessed as intended under all circumstances.
  - Unachievable

  Normally security problem caused by the following:
- **Intruders** (crackers) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
  - Attack can be accidental or malicious
  - Easier to protect against accidental than malicious misuse

Security Violation Categories

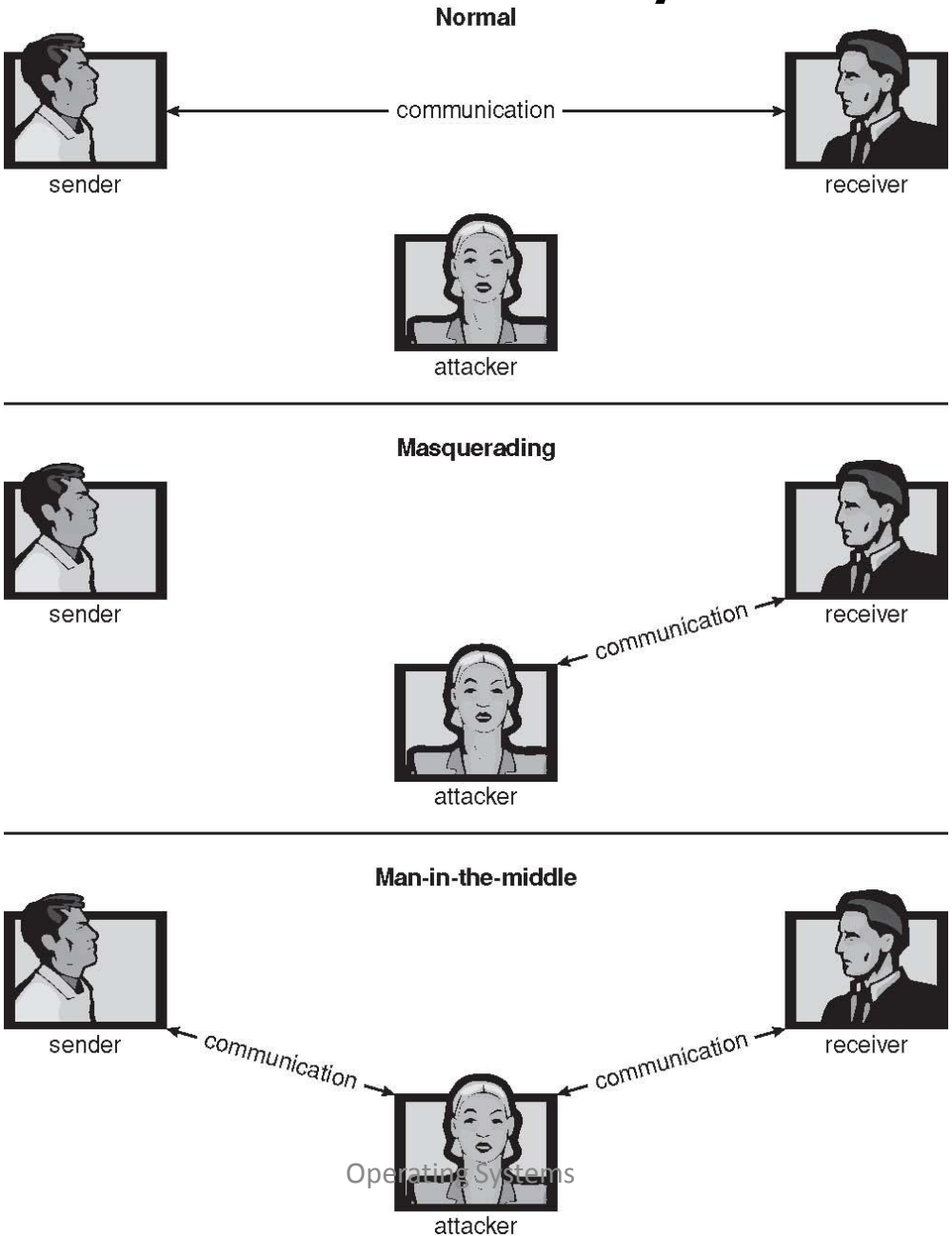
- **Breach of confidentiality**
  - Unauthorized reading of data eg: breaking up Passwords.
- **Breach of integrity**
  - Unauthorized modification of data eg: Checked by one of method called as Digital Signatures.
- **Breach of availability**
  - Unauthorized destruction of data eg: Web-site defacement
- **Theft of service**
  - Unauthorized use of resources eg:An intruder may install a daemon on a system that acts as a file server.
- **Denial of service (DOS)**
  - Prevention of legitimate use

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges

  eg:Gtalk
- **Replay attack**
  - As is or with message modification
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication

Standard Security Attacks



Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

Program Threats

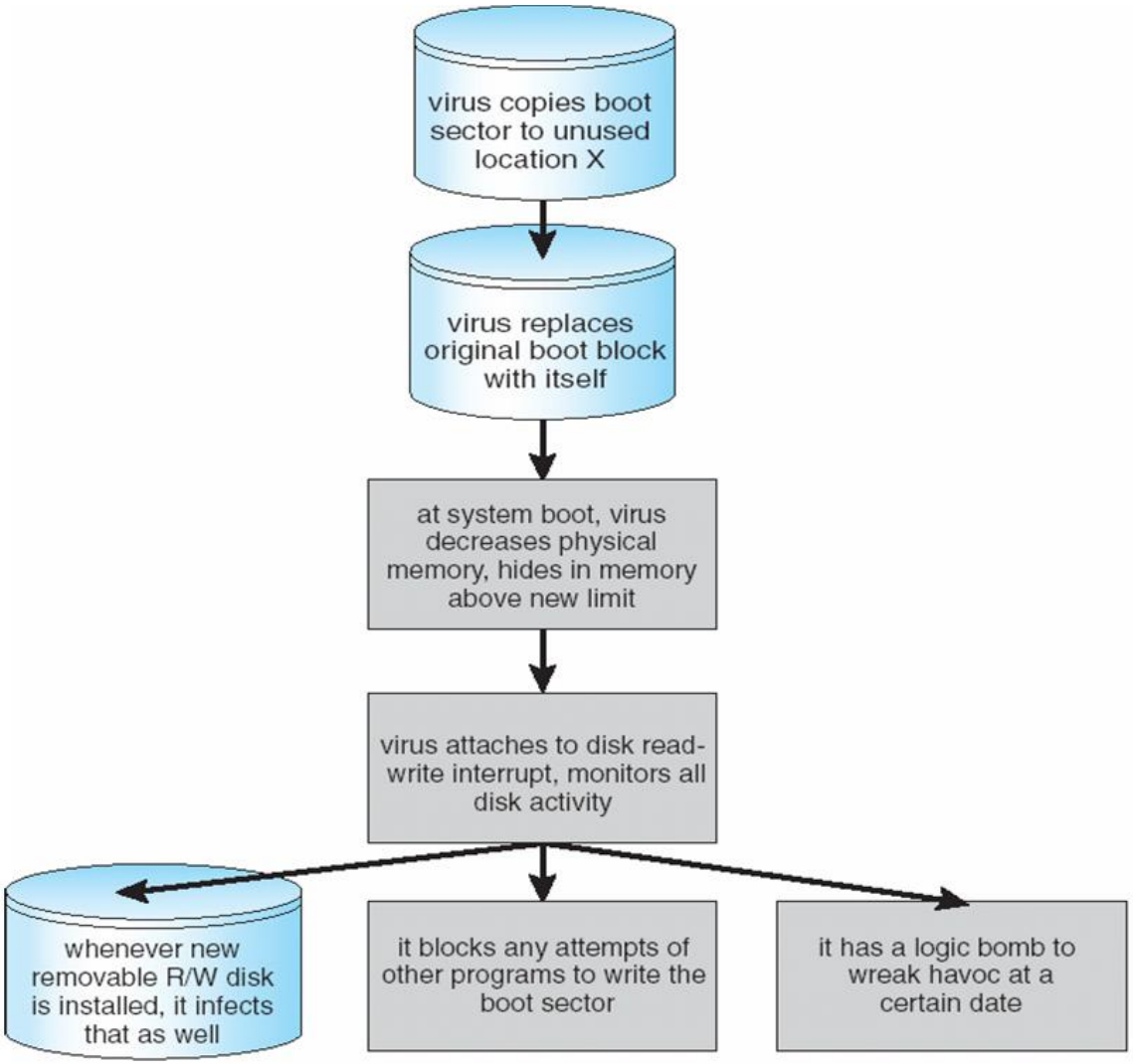
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Spyware**, **pop-up browser windows**, **covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

Program Threats (Cont.)

- **A code fragment embedded in a program is called Virus.**
- **This has a property of "Self Replication".**
- **Viruses are born via e-mail they can also be spread by downloading viral programs from internet file sharing services (or) exchange infected disks.**
- **Virus dropper** inserts virus onto the system

- Many categories of viruses, literally many thousands of viruses
  - File / parasitic eg: Pen drive (the file extensions will be changing)
  - Boot / memory eg: Boot sector code computer virus.
  - Macro eg: excel sheets
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Stealth – it gives false information
  - Tunneling –virus protected by a tunnel
  - Multipartite-same virus with multiple behaviours
  - Armored – hide the virus signature to be detected by the antivirus.

A Boot-sector Computer Virus



System and Network Threats

- Some systems “open” rather than secure by default
  - Reduce attack surface
  - But harder to use, more knowledge needed to administer

- Network threats harder to detect, prevent
  - Protection systems weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once system attached to internet
    - Or on network with system attached to internet
  - Even determining location of connecting system difficult
    - IP address is only knowledge

System and Network Threats (Cont.)

- **Worms** – use **spawn** mechanism; standalone program
- Internet worm
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
  - Exploited trust-relationship mechanism used by *rsh* to access friendly systems without use of password
  - **Grappling hook** program uploaded main worm program
    - 99 lines of C code
  - Hooked system then uploaded main code, tried to attack connected systems
  - Also tried to break into other users accounts on local system via password guessing
  - If target system already infected, abort, except for every 7^{th} time

System and Network Threats (Cont.)

- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Detection of answering service protocol
  - Detection of OS and version running on system
  - `nmap` scans all ports in a given IP range for a response
  - `nessus` has a database of protocols and bugs (and exploits) to apply against a system
  - Frequently launched from **zombie systems**
    - To decrease trace-ability

System and Network Threats (Cont.)

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed denial-of-service** (**DDOS**) come from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
    - How many started-connections can the OS handle?
  - Consider traffic to a web site
    - How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment

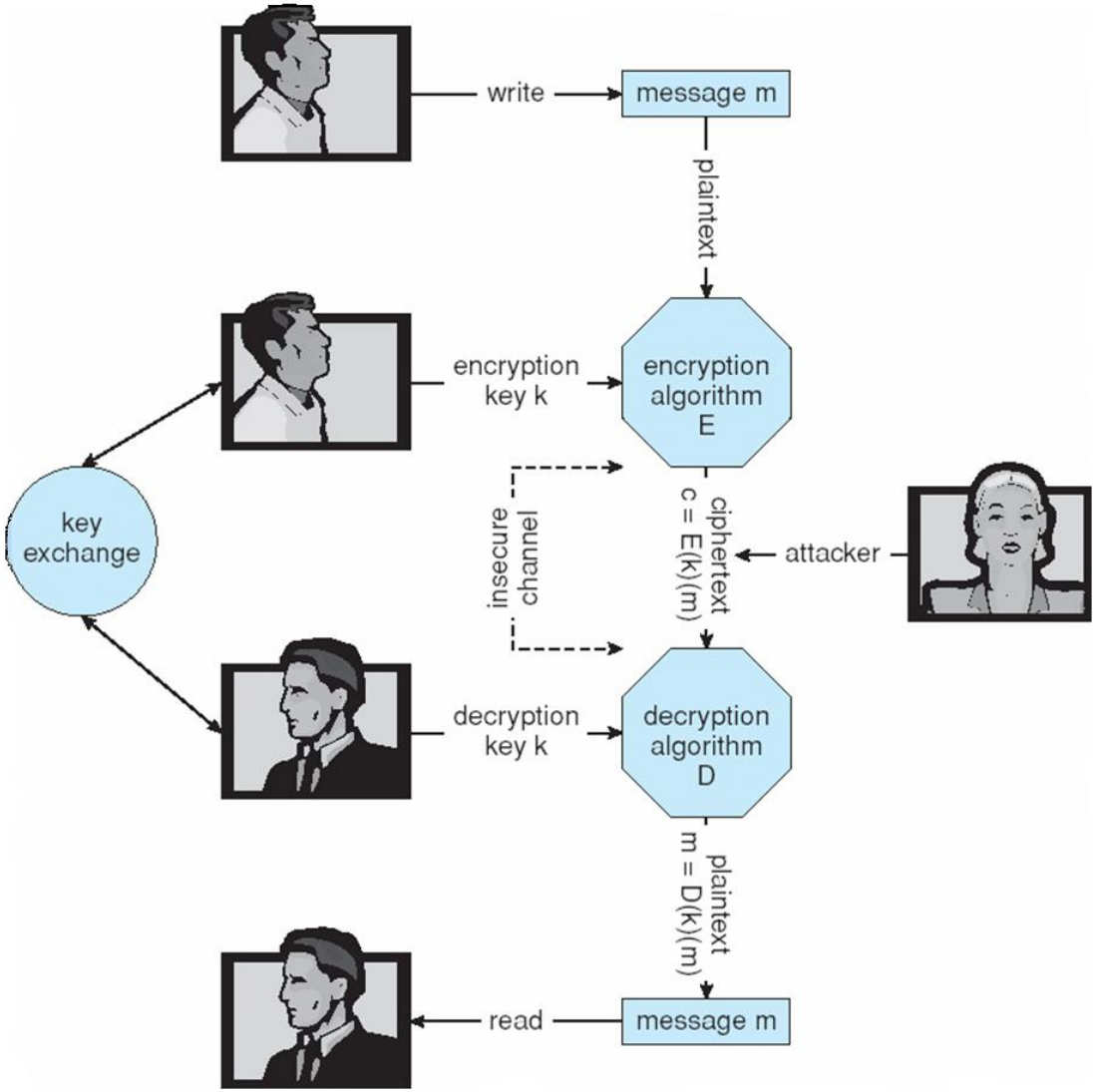
Cryptography as a Security Tool

- Broadest security tool available
  - Internal to a given computer, source and destination of messages can be known and protected
    - OS creates, manages, protects process IDs, communication ports
  - Source and destination of messages on network cannot be trusted without cryptography
    - Local network – IP address?
      - Consider unauthorized host added
    - WAN / Internet – how to establish authenticity
      - Not via IP address

Cryptography

- Means to constrain potential senders (*sources*) and / or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver

Secure Communication over Insecure Medium



Encryption

- **Encryption** algorithm consists of
  - Set *K* of keys
  - Set *M* of Messages
  - Set *C* of ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, $E(k)$ is a function for generating ciphertexts from messages
    - Both *E* and $E(k)$ for any *k* should be efficiently computable functions
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, $D(k)$ is a function for generating messages from ciphertexts
    - Both *D* and $D(k)$ for any *k* should be efficiently computable functions
- An encryption algorithm must provide this essential property: Given a ciphertext $c \in C$, a computer can compute m such that E(k)(m) = c only if it possesses D(k)
  - Thus, a computer holding $D(k)$ can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding $D(k)$ cannot decrypt ciphertexts
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive $D(k)$ from the ciphertexts

Symmetric Encryption

- Same key used to encrypt and decrypt
  - $E(k)$ can be derived from $D(k)$, and vice versa
- DES is most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
- Triple-DES considered more secure
- Advanced Encryption Standard (**AES**), **twofish** up and coming
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is a input to psuedo-random-bit generator
    - Generates an infinite **keystream**

Asymmetric Encryption

- Public-key encryption based on each user having two keys:
  - public key – published key used to encrypt data
  - private key – key known only to individual user used to decrypt data

- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is RSA block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number

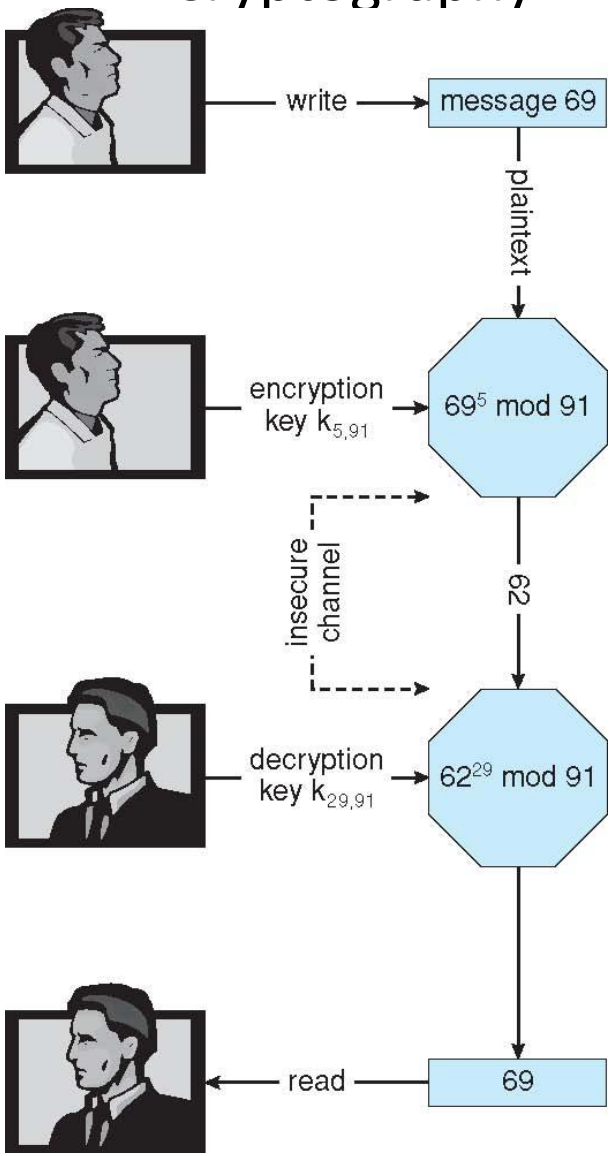
Asymmetric Encryption (Cont.)

- Formally, it is computationally infeasible to derive $D(k_d, N)$ from $E(k_e, N)$, and so $E(k_e, N)$ need not be kept secret and can be widely disseminated
  - $E(k_e, N)$ (or just k_e) is the **public key**
  - $D(k_d, N)$ (or just k_d) is the **private key**
  - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)
  - Encryption algorithm is $E(k_e, N)(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
  - The decryption algorithm is then $D(k_d, N)(c) = c^{k_d} \bmod N$

Asymmetric Encryption Example

- For example. make p = 7and q = 13
- We then calculate $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and< 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29
- We how have our keys
  - Public key, k_e, N = 5, 91
  - Private key, k_d , N = 29, 91
- Encrypting the message 69 with the public key results in the cypher text 62
- Cypher text can be decoded with the private key
  - Public key can be distributed in clear text to anyone who wants to communicate with holder of public key

Encryption and Decryption using RSA Asymmetric Cryptography



Authentication

- Constraining set of potential senders of a message
  - Complementary and sometimes redundant to encryption
  - Also can prove message unmodified

- Algorithm components
  - A set *K* of keys
  - A set *M* of messages
  - A set *A* of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, $S(k)$ is a function for generating authenticators from messages
    - Both *S* and $S(k)$ for any *k* should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$. That is, for each $k \in K$, $V(k)$ is a function for verifying authenticators on messages
    - Both *V* and $V(k)$ for any *k* should be efficiently computable functions

Authentication (Cont.)

- For a message m, a computer can generate an authenticator $a \in A$ such that $V(k)(m, a)$ = `true` only if it possesses $S(k)$
- Thus, computer holding $S(k)$ can generate authenticators on messages so that any other computer possessing $V(k)$ can verify them
- Computer not holding $S(k)$ cannot generate authenticators on messages that can be verified using $V(k)$
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive $S(k)$ from the authenticators

Authentication – Hash Functions

- Basis of authentication
- Creates small, fixed-size block of data (**message digest**, **hash value**) from *m*
- Hash Function *H* must be collision resistant on *m*
  - Must be infeasible to find an $m' \neq m$ such that $H(m) = H(m')$
- If $H(m) = H(m')$, then $m = m'$
  - The message has not been modified
- Common message-digest functions include **MD5**, which produces a 128-bit hash, and **SHA-1**, which outputs a 160-bit hash

Authentication - MAC

- Symmetric encryption used in **message-authentication code** (**MAC**) authentication algorithm

- Simple example:
  - MAC defines $S(k)(m) = f(k, H(m))$
    - Where f is a function that is one-way on its first argument
      - k cannot be derived from $f(k, H(m))$
    - Because of the collision resistance in the hash function, reasonably assured no other message could create the same MAC
    - A suitable verification algorithm is $V(k)(m, a) \equiv (f(k,m) = a)$
    - Note that k is needed to compute both $S(k)$ and $V(k)$, so anyone able to compute one can compute the other

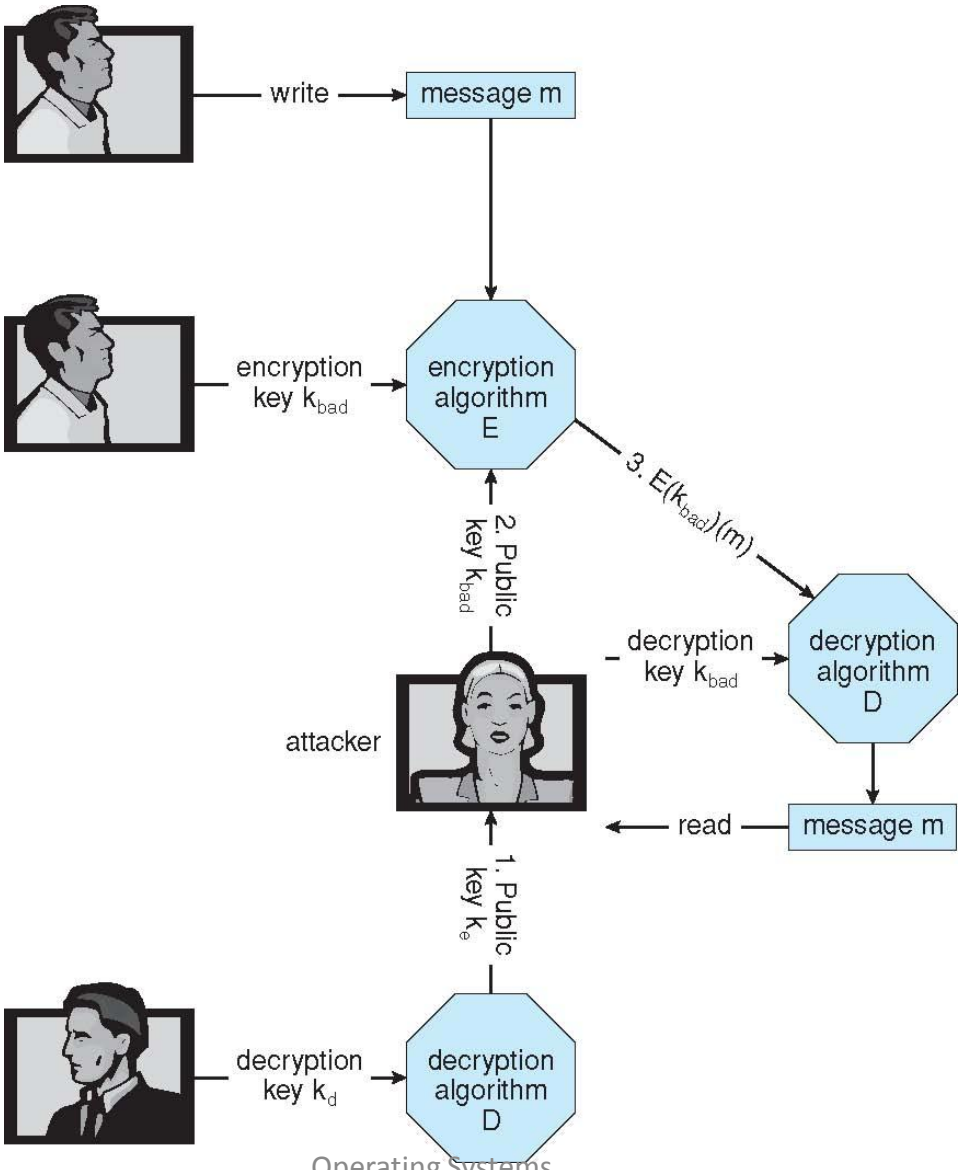
Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- In a digital-signature algorithm, computationally infeasible to derive $S(k_s)$ from $V(k_v)$
  - V is a one-way function
  - Thus, k_v is the public key and k_s is the private key
- Consider the RSA digital-signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message $S(k_s)(m) = H(m)^{k_s} \bmod N$
  - The key k_s again is a pair d, N, where N is the product of two large, randomly chosen prime numbers p and q
  - Verification algorithm is $V(k_v)(m, a) \equiv (a^{k_v} \bmod N = H(m))$
    - Where k_v satisfies $k_v k_s \bmod (p-1)(q-1) = 1$

Authentication (Cont.)

- Why authentication if a subset of encryption?
  - Fewer computations (except for RSA digital signatures)
  - Authenticator usually shorter than message
  - Sometimes want authentication but not confidentiality
    - Signed patches et al
  - Can be basis for **non-repudiation**

Man-in-the-middle Attack on Asymmetric Cryptography

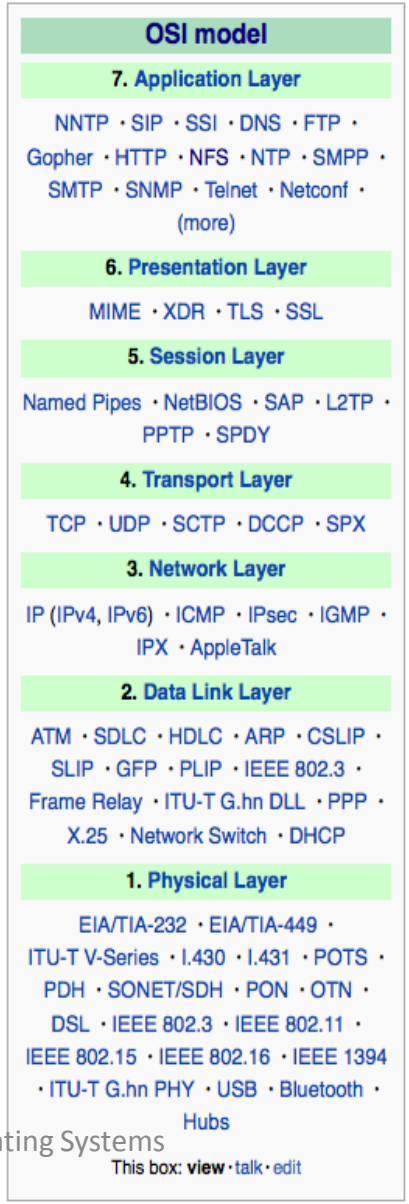


Digital Certificates

- Proof of who or what owns a public key
- Public key digitally signed a trusted party
- Trusted party receives proof of identification from entity and certifies that public key belongs to entity
- Certificate authority are trusted party – their public keys included with web browser distributions
  - They vouch for other authorities via digitally signing their keys, and so on

Implementation of Cryptography

- Can be done at various levels of ISO Reference Model
  - SSL at the Transport layer
  - Network layer is typically IPSec
    - IKE for key exchange
    - Basis of VPNs
- Why not just at lowest level?
  - Sometimes need more knowledge than available at low levels
    - i.e. User authentication
    - i.e. e-mail delivery



OSI Model			
	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication
	Segments	4. Transport	End-to-end connections and reliability, flow control
Media layers	Packet/Datagram	3. Network	Path determination and logical addressing
	Frame	2. Data Link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

Source: http://en.wikipedia.org/wiki/OSI_model

Encryption Example - SSL

- Insertion of cryptography at one layer of the ISO network model (the transport layer)
- SSL – Secure Socket Layer (also called TLS)
- Cryptographic protocol that limits two computers to only exchange messages with each other
  - Very complicated, with many variations
- Used between web servers and browsers for secure communication (credit card numbers)
- The server is verified with a **certificate** assuring client is talking to correct server
- Asymmetric cryptography used to establish a secure **session key** (symmetric encryption) for bulk of communication during session
- Communication between each computer then uses symmetric key cryptography

User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through *passwords*, can be considered a special case of either keys or capabilities
- Passwords must be kept secret
  - Frequent change of passwords
  - History to avoid repeats
  - Use of “non-guessable” passwords
  - Log all invalid access attempts (but not the passwords themselves)
  - Unauthorized transfer
- Passwords may also either be encrypted or allowed to be used only once
  - Does encrypting passwords solve the exposure problem?
    - Might solve **sniffing**
    - Consider **shoulder surfing**
    - Consider Trojan horse keystroke logger
    - How are passwords stored at authenticating site?

Passwords

- Encrypt to avoid having to keep secret
  - But keep secret anyway (i.e. Unix uses super user-only readably file `/etc/shadow`)
  - Use algorithm easy to compute but difficult to invert
  - Only encrypted password stored, never decrypted
  - Add “salt” to avoid the same password being encrypted to the same value
- One-time passwords
  - Use a function based on a seed to compute a password, both user and computer
  - Hardware device / calculator / key fob to generate the password
    - Changes very frequently
- Biometrics
  - Some physical attribute (fingerprint, hand scan)

- Multi-factor authentication
  - Need two or more factors for authentication
    - i.e. USB “dongle”, biometric measure, and password

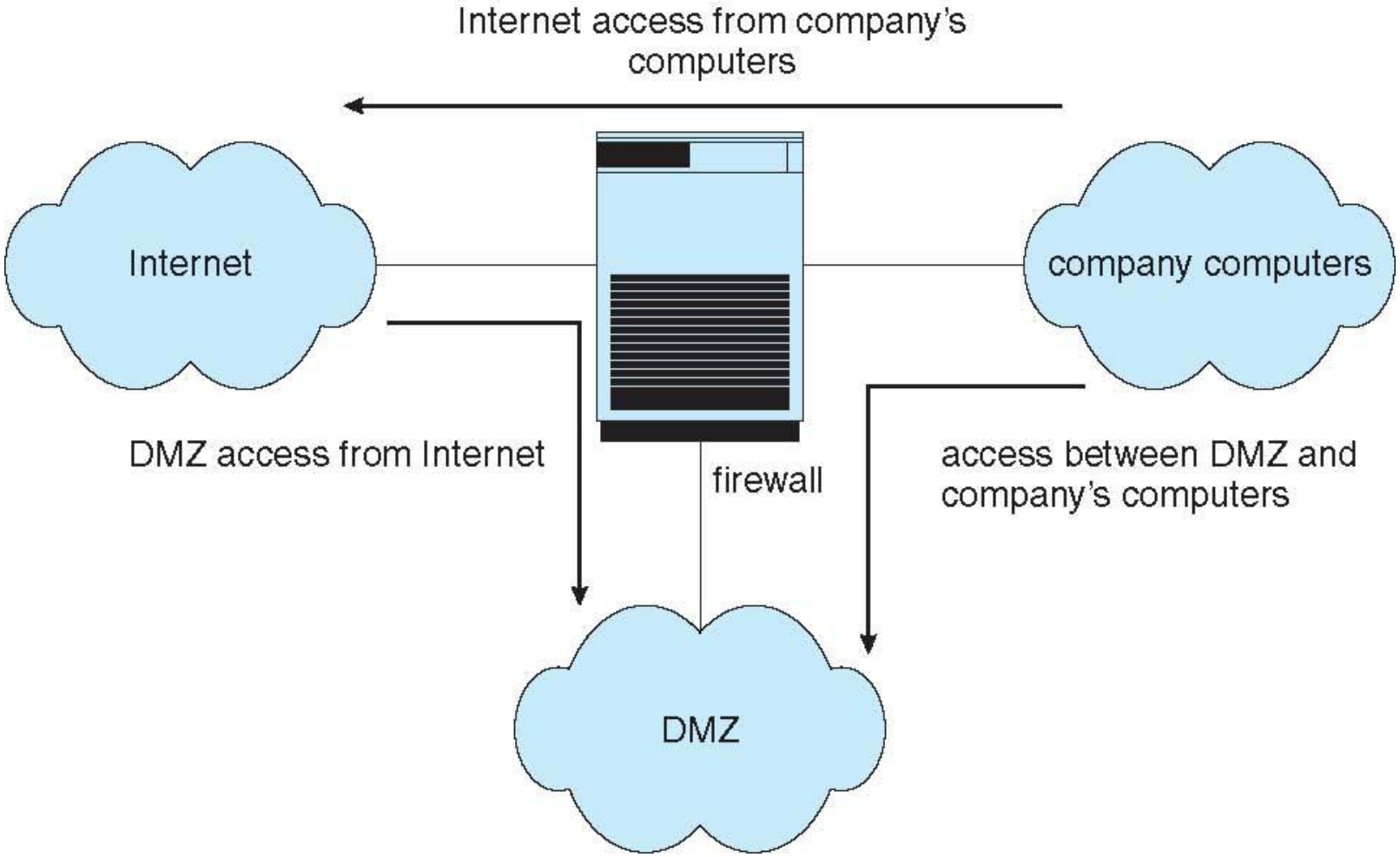
Implementing Security Defenses

- **Defense in depth** is most common security theory – multiple layers of security
- Security policy describes what is being secured
- Vulnerability assessment compares real state of system / network compared to security policy
- Intrusion detection endeavors to detect attempted or successful intrusions
  - **Signature-based** detection spots known bad patterns
  - **Anomaly detection** spots differences from normal behavior
    - Can detect **zero-day** attacks
  - **False-positives** and **false-negatives** a problem
- Virus protection
- Auditing, accounting, and logging of all or specific system or network activities

Firewalling to Protect Systems and Networks

- A network firewall is placed between trusted and untrusted hosts
  - The firewall limits network access between these two security domains
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address which can be spoofed
- **Personal firewall** is software layer on given host
  - Can monitor / limit traffic to and from the host
- **Application proxy firewall** understands application protocol and can control them (i.e., SMTP)
- **System-call firewall** monitors all important system calls and apply rules to them (i.e., this program can execute that system call)

Network Security Through Domain Separation Via Firewall



Computer Security Classifications

- U.S. Department of Defense outlines four divisions of computer security: **A**, **B**, **C**, and **D**
- **D** – Minimal security

- **C** – Provides discretionary protection through auditing
  - Divided into **C1** and **C2**
    - **C1** identifies cooperating users with the same level of protection
    - **C2** allows user-level access control

- **B** – All the properties of **C**, however each object may have unique sensitivity labels
  - Divided into **B1**, **B2**, and **B3**

- **A** – Uses formal design and verification techniques to ensure security

Example: Windows

- Security is based on user accounts
  - Each user has unique security ID
  - Login to ID creates **security access token**
    - Includes security ID for user, for user's groups, and special privileges
    - Every process gets copy of token
    - System checks token to determine if access allowed or denied

- Uses a subject model to ensure access security
  - A subject tracks and manages permissions for each program that a user runs

- Each object in Windows has a security attribute defined by a security descriptor
  - For example, a file has a security descriptor that indicates the access permissions for all users