

Introduction

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 Definition

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

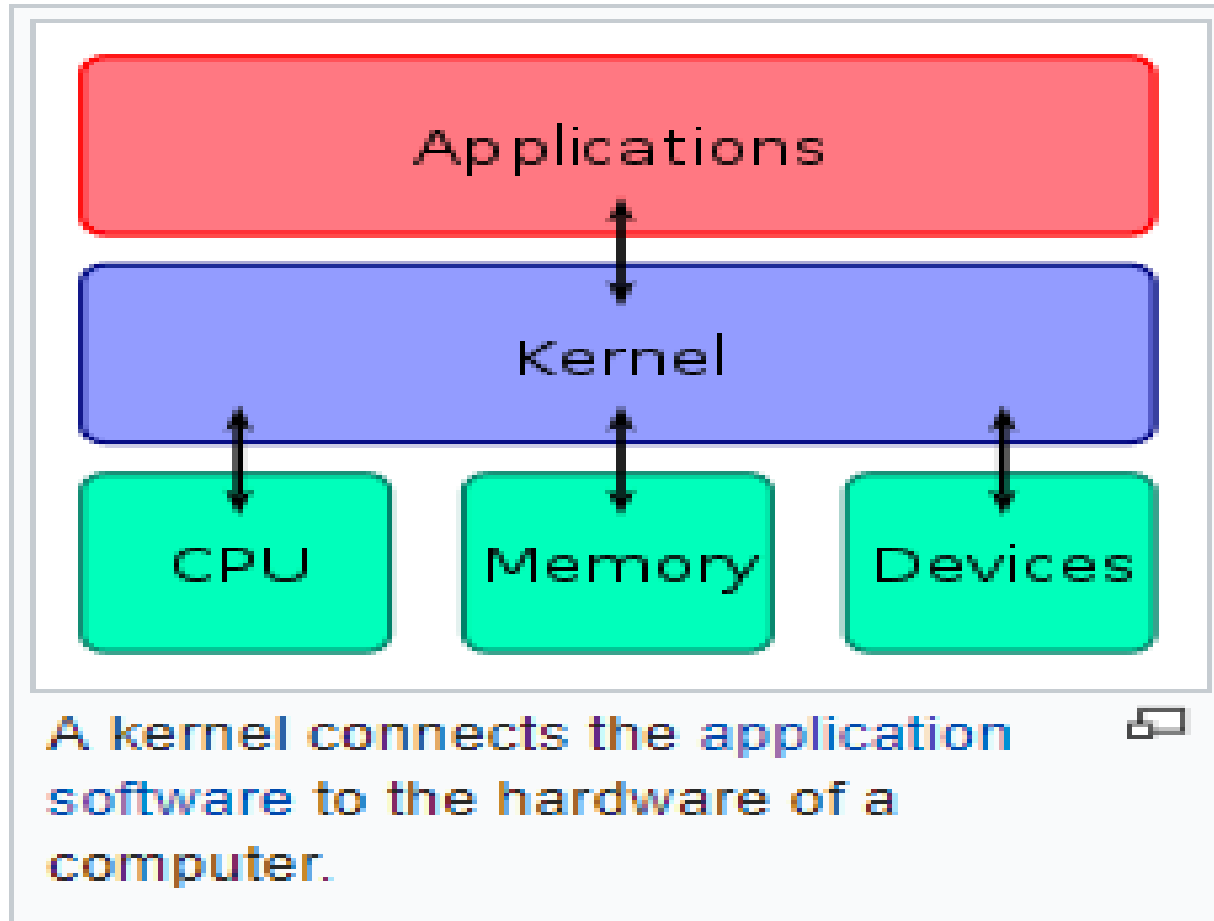
Operating System Definition (Cont.)

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
 - But varies wildly
- “The one program running at all times on the computer” is the **kernel**.
- Everything else is either
 - a system program (ships with the operating system) , or
 - an application program.

Computer Startup

- **bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM(erasable programmable read-only memory) , generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

Kernel



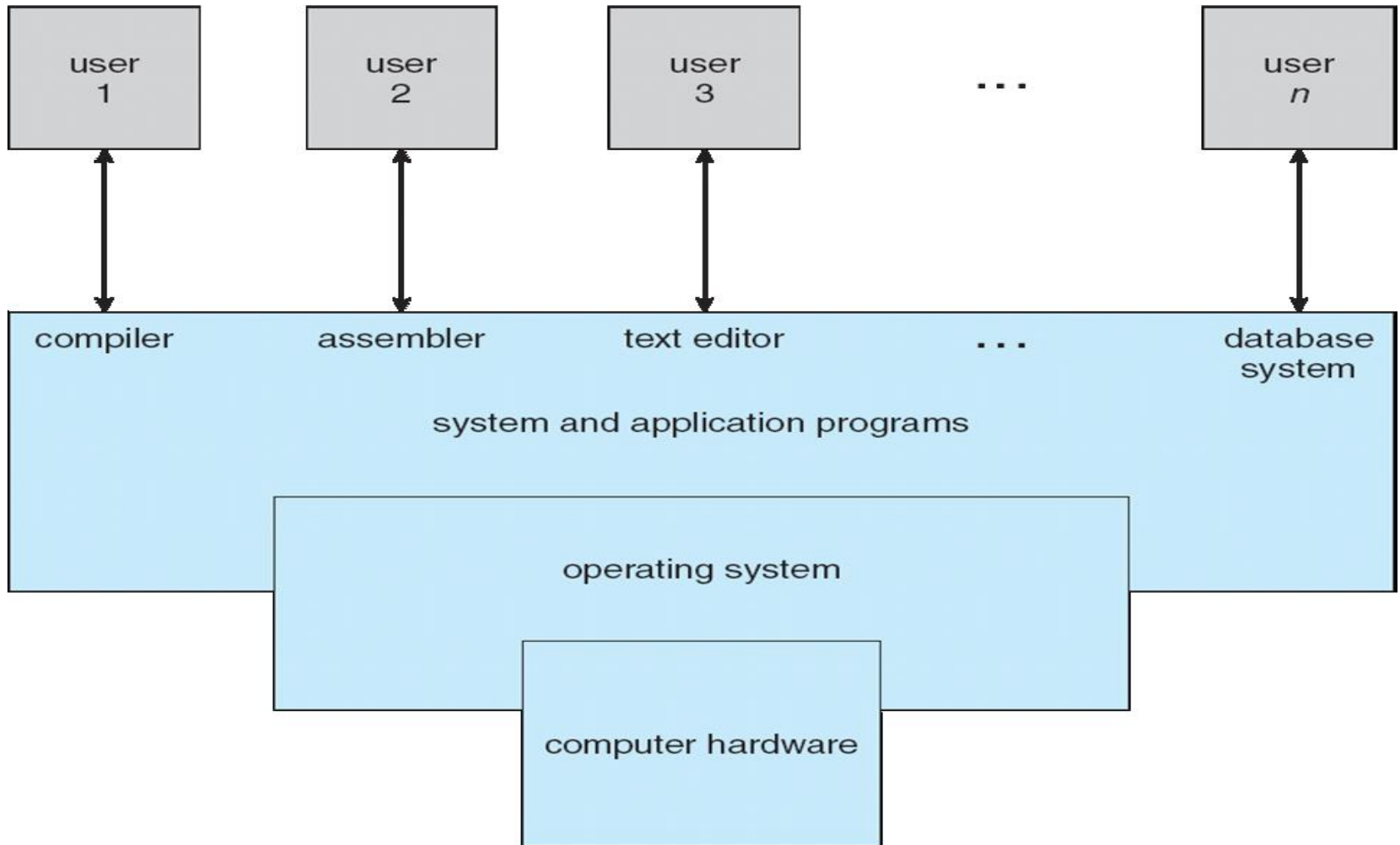
Computer System Structure

- Computer system can be divided into four components:
 - Hardware – provides basic computing resources
 - CPU, memory, I/O devices
 - Operating system
 - Controls and coordinates use of hardware among various applications and users

Computer System Structure

- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
 - Word processors, compilers, web browsers, database systems, video games
- Users
 - People, machines, other computers

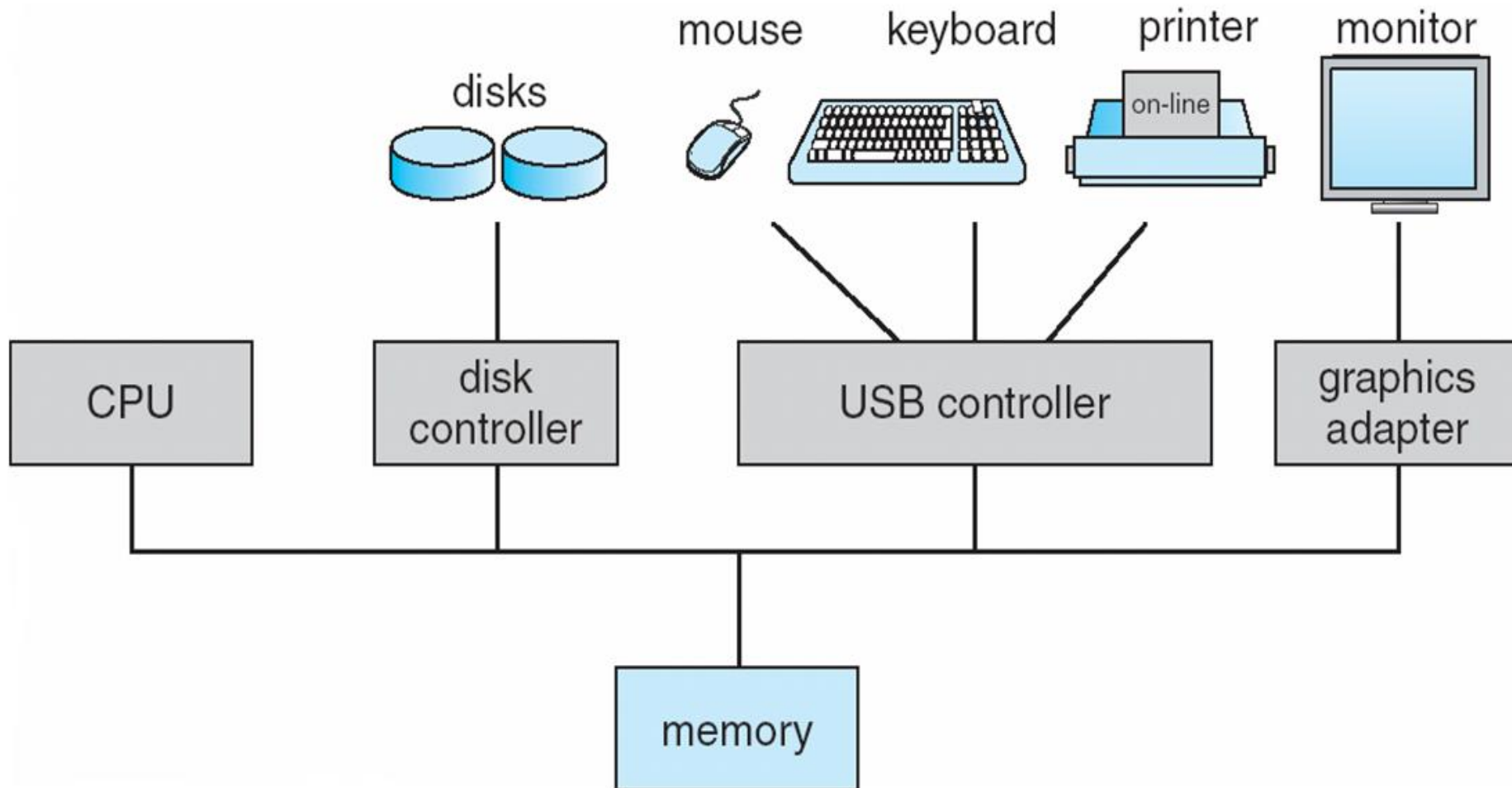
Four Components of a Computer System



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 Organization



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**

Common Functions of Interrupts

- 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
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Separate segments of code determine what action should be taken for each type of interrupt

Types of an Operating System

Batch operating system

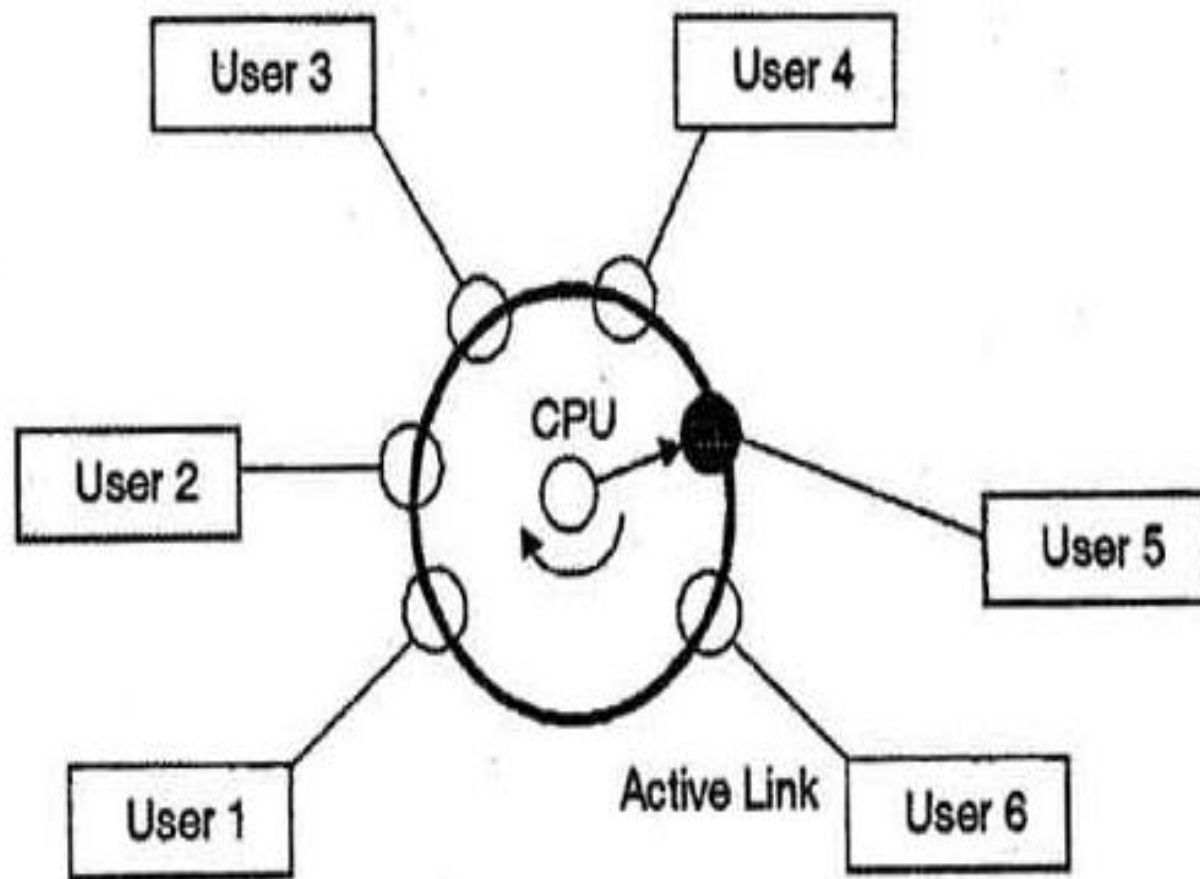
- The users of a batch operating system do not interact with the computer directly.
- Each user prepares his job on an off-line device like punch cards and submits it to the computer operator.
- To speed up processing, jobs with similar needs are batched together and run as a group.
- The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.
- Example : Payroll System , Bank Statements

Time-sharing operating systems

- Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time.
- Time-sharing or multitasking is a logical extension of multiprogramming.
- Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- Example : Multics , Unix

Time-sharing operating systems

- The time sharing system provides the direct access to a large number of users where CPU time is divided among all the users on scheduled basis.
- The OS allocates a set of time to each user. When this time is expired, it passes control to the next user on the system.
- The time allowed is extremely small and the users are given the impression that they each have their own CPU and they are the sole owner of the CPU.
- This short period of time during that a user gets attention of the CPU; is known as a time slice or a quantum. The concept of time sharing system is shown in figure.

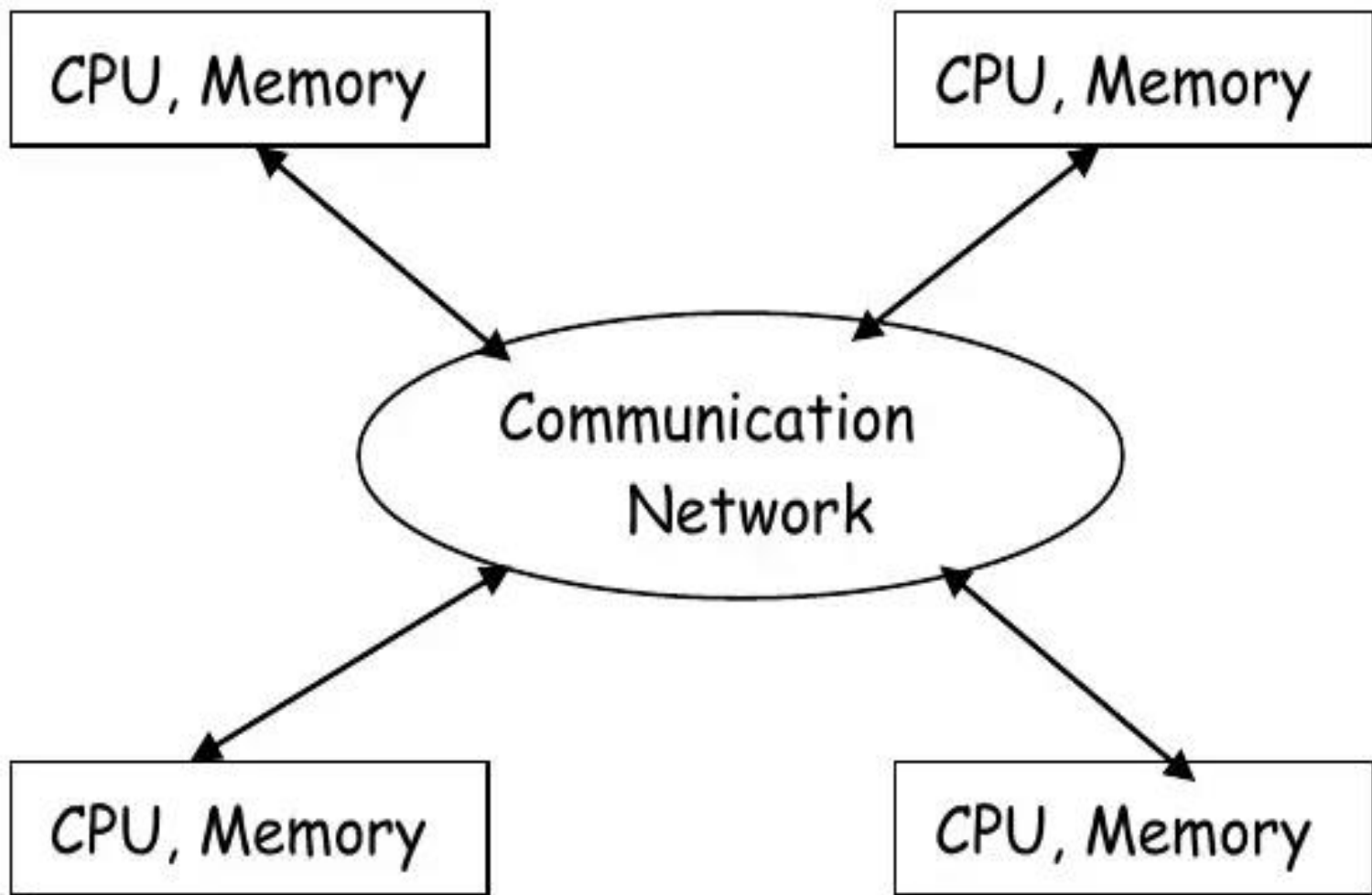


Distributed operating System

- Distributed systems use multiple central processors to serve multiple real-time applications and multiple users.
- Data processing jobs are distributed among the processors accordingly.
- The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems.
- Processors in a distributed system may vary in size and function

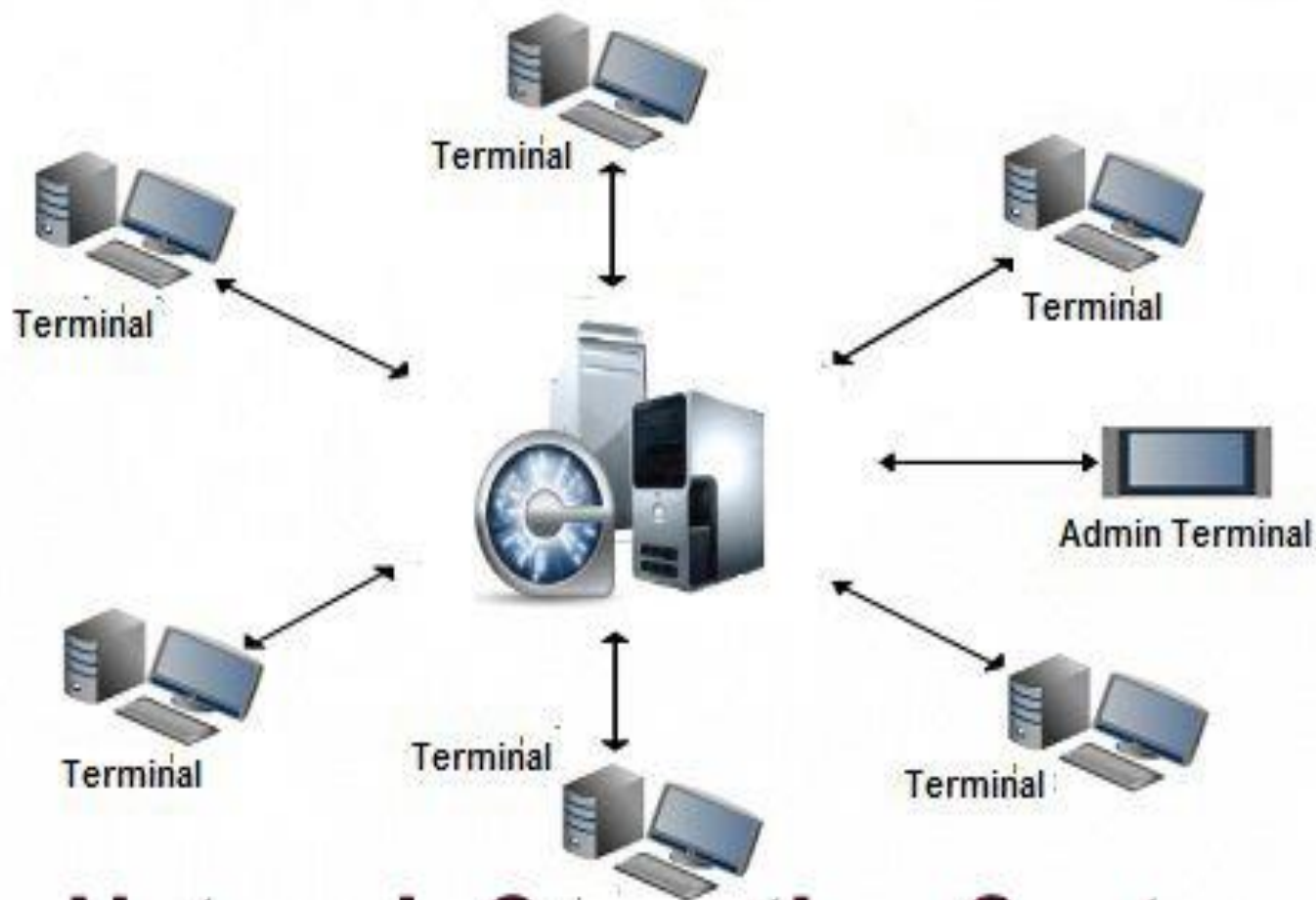
Distributed operating System

- These systems are referred as loosely coupled systems where each processor has its own local memory and processors communicate with one another through various communication lines, such as high speed buses or telephone lines.
- Example : Locus



Network operating System

- **Network Operating System** is an [operating system](#) that includes special functions for connecting computers and devices into a local-area network (LAN) or Inter-network.
- A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions.
- The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.



Network Operating System

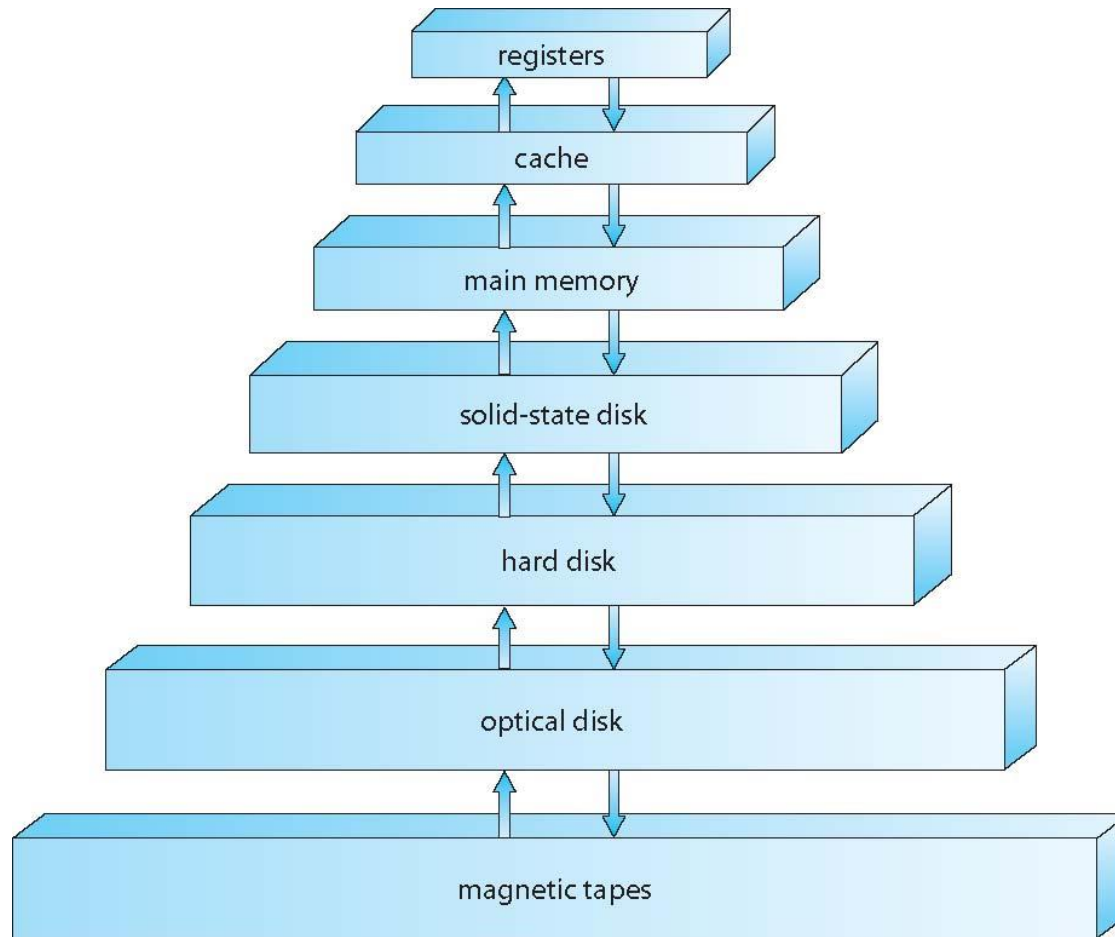
Storage Structure

- Main memory – only large storage media that the CPU can access directly(RAM)
 - Random access
 - Typically, **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard 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
- **Solid-state disks** – faster than hard disks, nonvolatile
 - Various technologies
 - Becoming more popular

Storage Hierarchy

- Storage systems organized in hierarchy
 - Speed
 - Cost
 - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
 - Provides uniform interface between controller and kernel

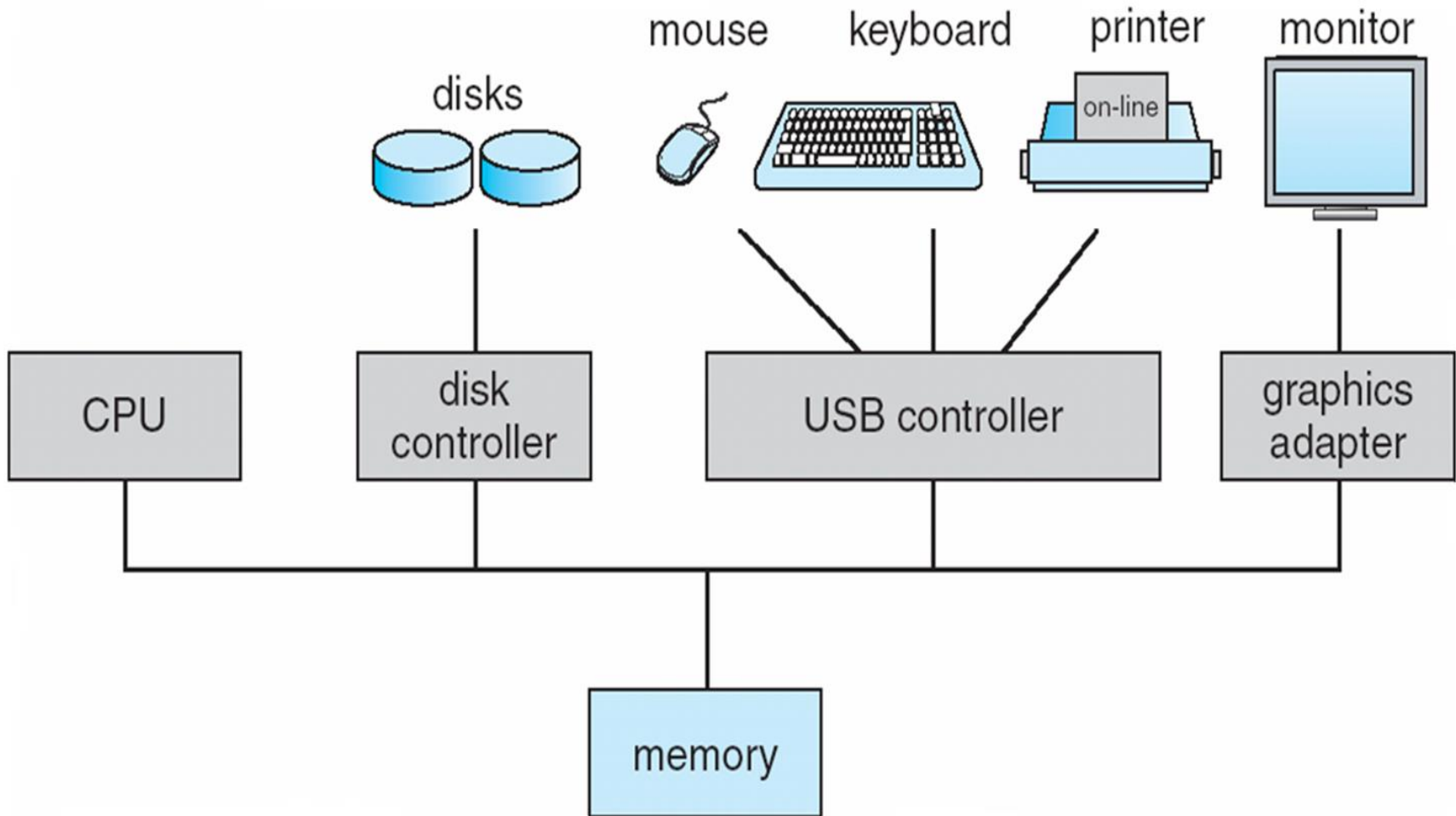
Storage-Device Hierarchy



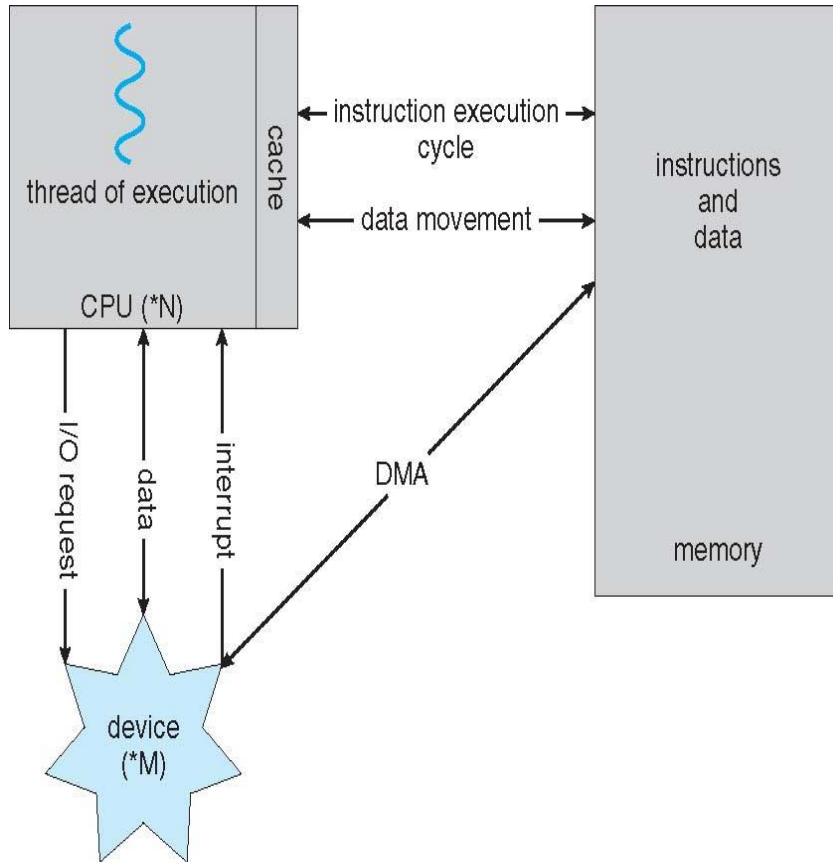
Caching

- 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

I/O STRUCTURE



How a Modern Computer Works



- > To start an I/O operation, the device driver loads the appropriate registers within the device controller
- > The device controller, in turn, examines the contents of these registers to determine what action to take
- > The controller starts the 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
- > The device driver then returns control to the operating system

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

A von Neumann architecture

Direct Memory Access Structure

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

Computer-System Architecture

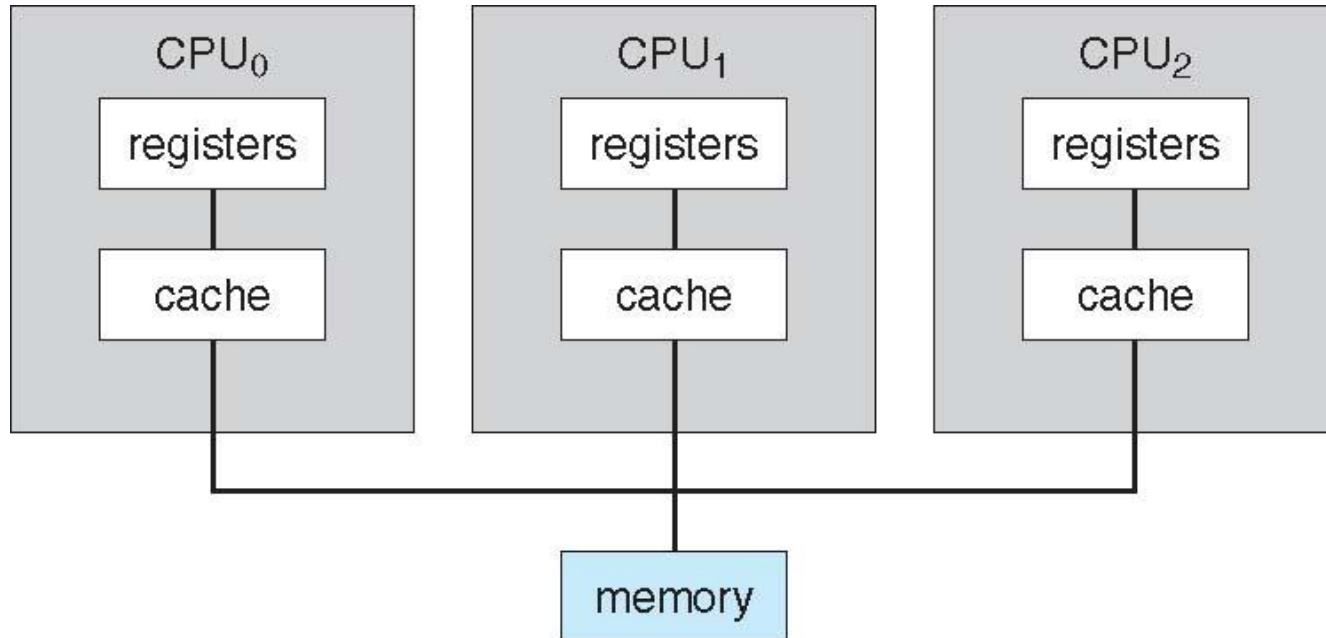
- Types of Computer systems based on number of General-purpose processors.
 1. Single Processor System
 2. Multi Processor System
 3. Clustered Systems

Computer-System Architecture

- Most systems use a single general-purpose processor
 - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
 - Also known as **parallel systems**, **tightly-coupled systems**
 - Advantages include:
 1. **Increased throughput**
 2. **Economy of scale**
 3. **Increased reliability** – graceful degradation or fault tolerance
 - Two types:
 1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
 2. **Symmetric Multiprocessing** – each processor performs all tasks

BASIS FOR COMPARISON	LOOSELY COUPLED MULTIPROCESSOR SYSTEM	TIGHTLY COUPLED MULTIPROCESSOR SYSTEM
Basic	Each processor has its own memory module.	Processors have shared memory modules.
Efficient	Efficient when tasks running on different processors, has minimal interaction.	Efficient for high-speed or real-time processing.
Memory conflict	It generally, do not encounter memory conflict.	It experiences more memory conflicts.
Interconnections	Message transfer system (MTS).	Interconnection networks PMIN, IOPIN, ISIN.
Data rate	Low.	High.
Expensive	Less expensive.	More expensive.

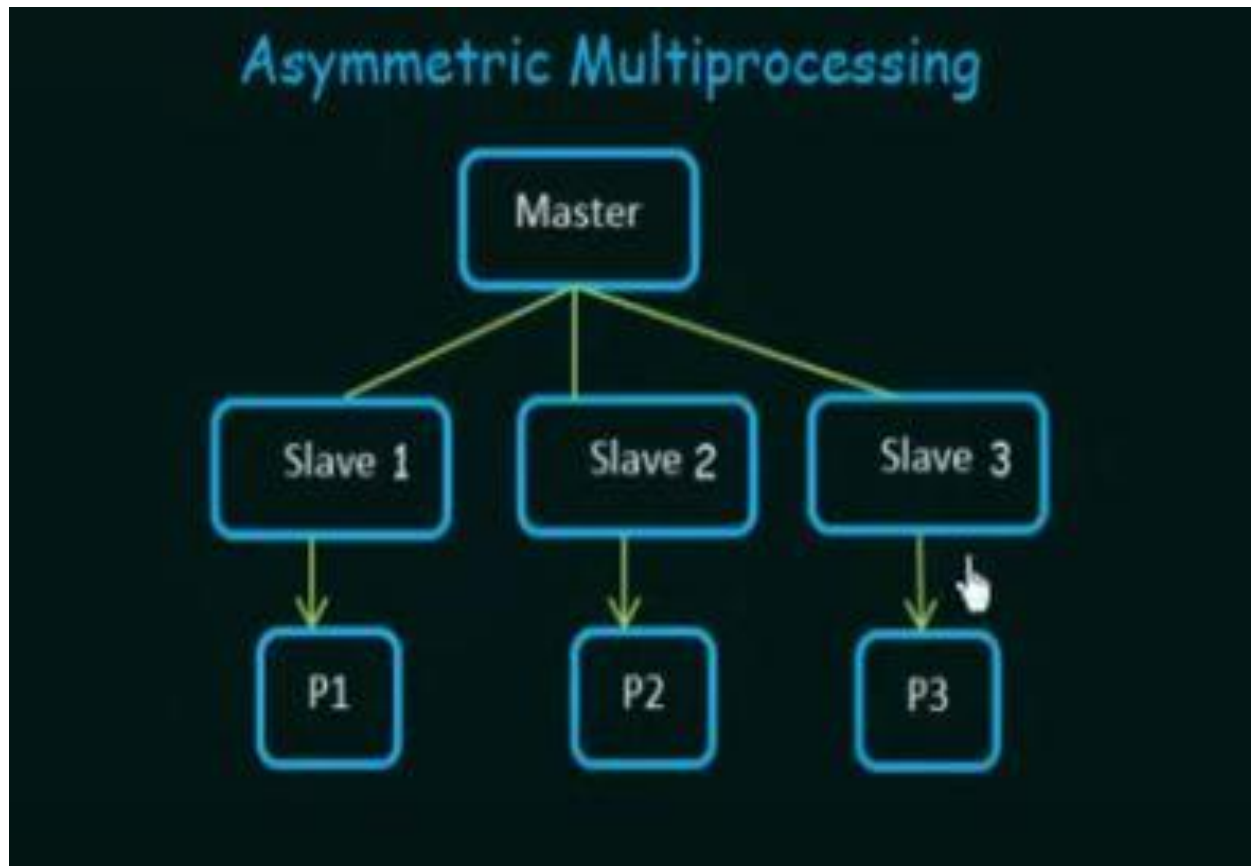
Symmetric Multiprocessing Architecture



Symmetric Multiprocessing Architecture

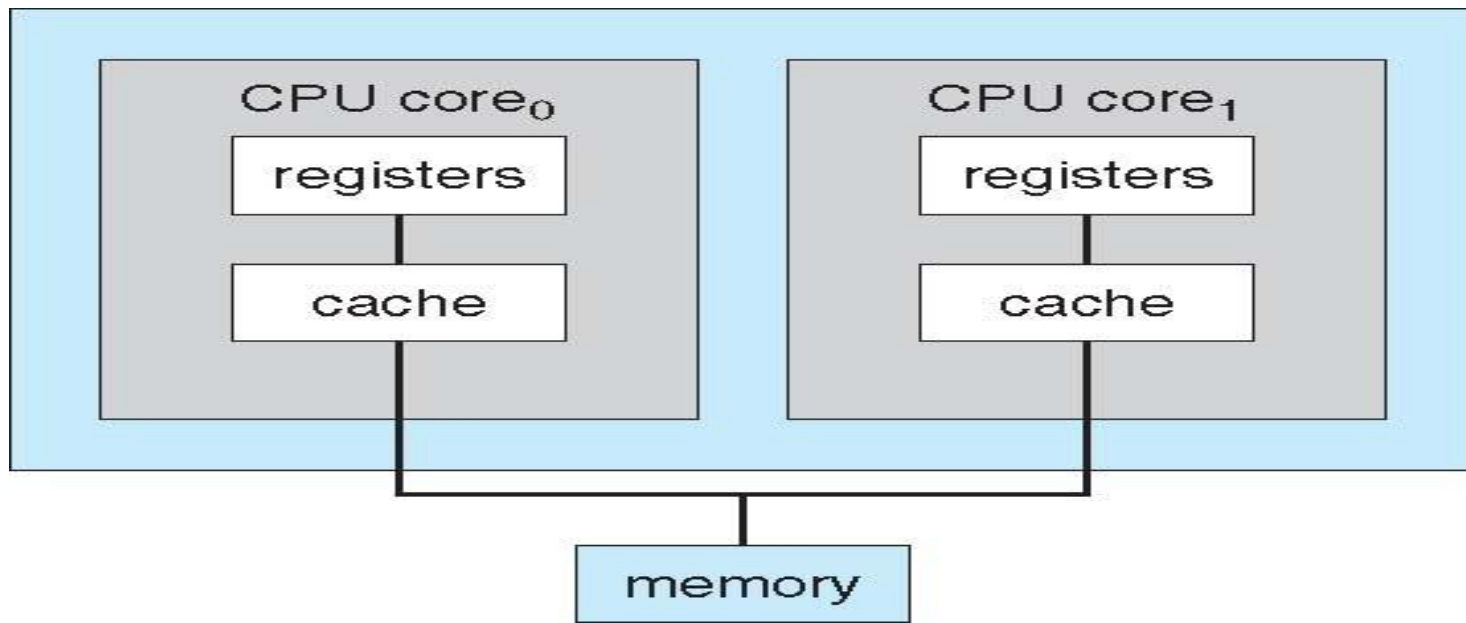
- Symmetric multiprocessing is the use of two or more self-scheduling processors sharing a common memory space.
- Each processor has access to I/O and memory devices. SMP applies multiple CPUs to a task to complete in parallel and faster fashion.

Asymmetric Multiprocessing



A Dual-Core Design

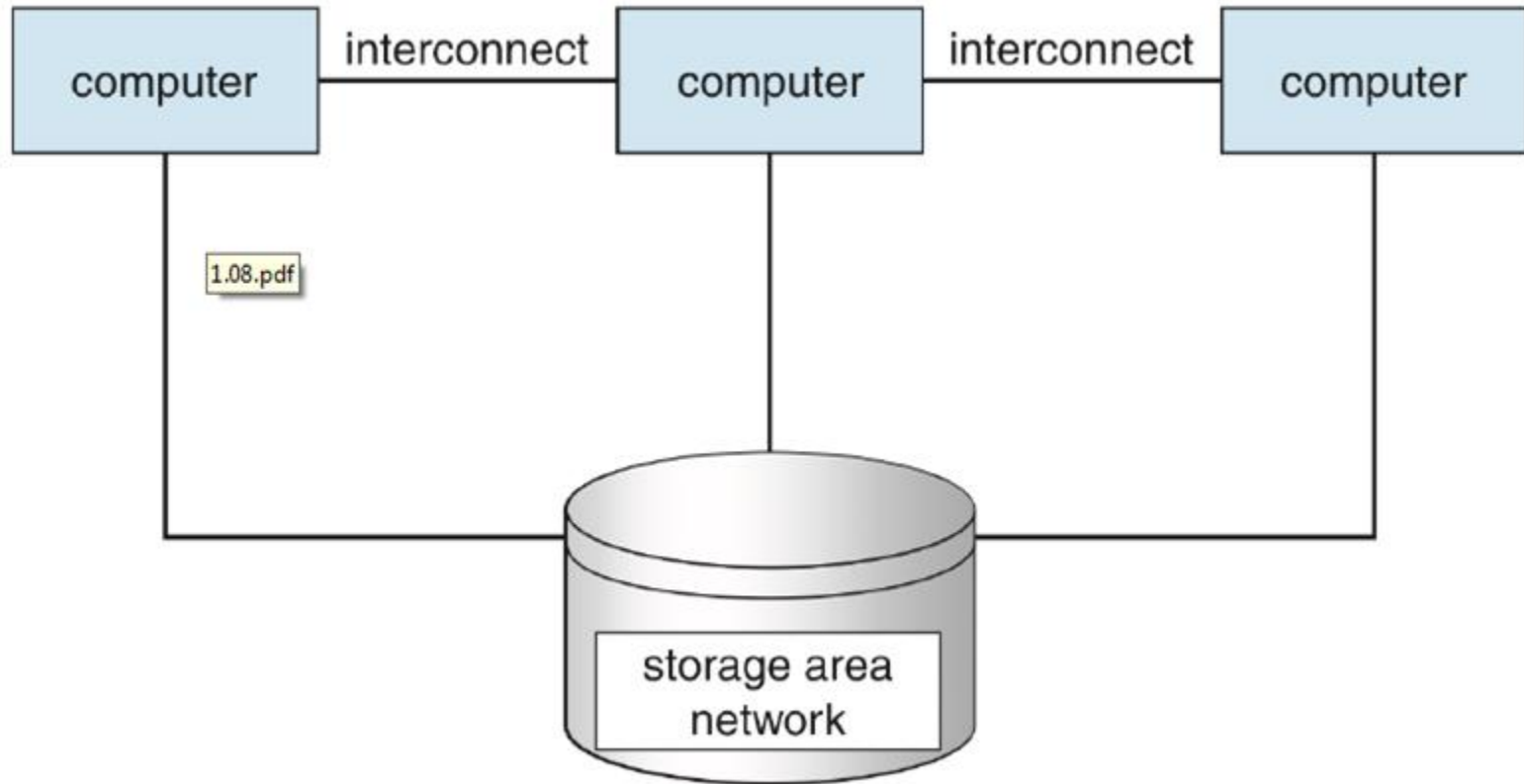
- Multi-chip and **multicore**
- Systems containing all chips
 - A dual-core processor is a **CPU** with two processors or "execution cores" in the same **integrated circuit**. Each processor has its own **cache** and controller, which enables it to function as efficiently as a single processor. However, because the two processors are linked together, they can perform operations up to twice as fast as a single processor can.



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
 - Usually sharing storage via a **storage-area network (SAN)**
 - Provides a **high-availability** service which survives failures
 - **Asymmetric clustering** has one machine in hot-standby mode
 - **Symmetric clustering** has multiple nodes running applications, monitoring each other
 - Some clusters are for **high-performance computing (HPC)**
 - Applications must be written to use **parallelization**
 - Some have **distributed lock manager (DLM)** to avoid conflicting operations

Clustered Systems



Operating System Services

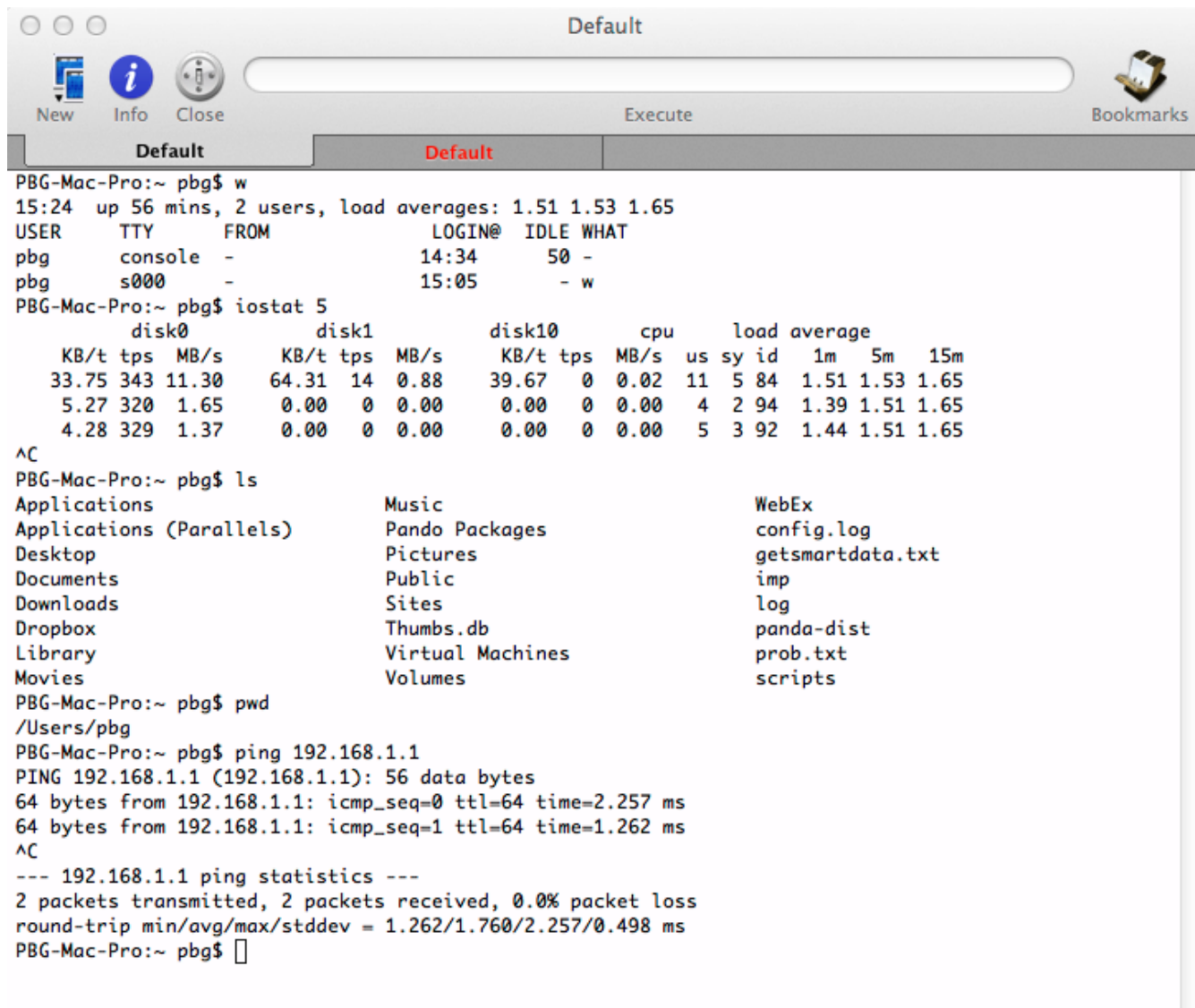
- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:
 - **User interface** - Almost all operating systems have a user interface (UI).
 - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device

User Operating System Interface - CLI

CLI or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter



```
Default
New Info Close Execute Bookmarks

PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM            LOGIN@   IDLE   WHAT
pbg       console  -               14:34    50    -
pbg       s000    -               15:05    -    w
PBG-Mac-Pro:~ pbg$ iostat 5

            disk0      disk1      disk10      cpu      load average
      KB/t tps MB/s    KB/t tps MB/s    KB/t tps MB/s  us sy id  1m  5m  15m
      33.75 343 11.30    64.31 14  0.88    39.67  0  0.02  11  5 84  1.51 1.53 1.65
      5.27 320  1.65     0.00  0  0.00     0.00  0  0.00   4  2 94  1.39 1.51 1.65
      4.28 329  1.37     0.00  0  0.00     0.00  0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications          Music                  WebEx
Applications (Parallels)  Pando Packages       config.log
Desktop               Pictures              getsmartdata.txt
Documents             Public                imp
Downloads             Sites                 log
Dropbox               Thumbs.db             panda-dist
Library              Virtual Machines     prob.txt
Movies               Volumes               scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

User Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
 - Usually mouse, keyboard, and monitor
 - **Icons** represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))
 - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen Interfaces

n Touchscreen devices require new interfaces

- Mouse not possible or not desired
- Actions and selection based on gestures
- Virtual keyboard for text entry
- Voice commands.



Operating System Services (Cont.)

One set of operating-system services provides functions that are helpful to the user (Cont.):

File-system manipulation - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file information, permission management.

Communications – Processes may exchange information, on the same computer or between computers over a network

Communications may be via shared memory or through message passing (packets moved by the OS)

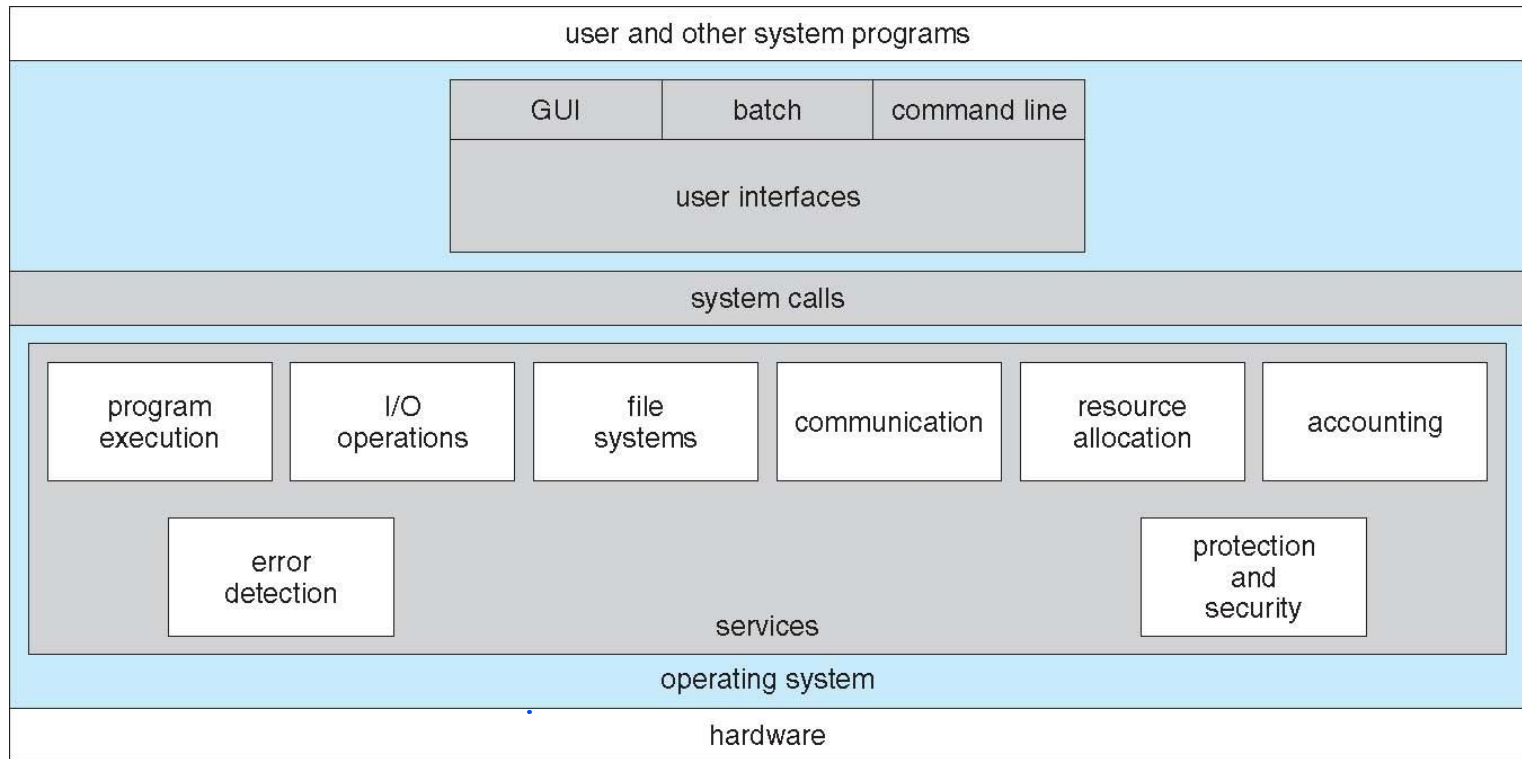
Error detection – OS needs to be constantly aware of possible errors

- May occur in the CPU and memory hardware, in I/O devices, in user program
- For each type of error, OS should take the appropriate action to ensure correct and consistent computing
- Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
 - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
 - **Accounting** - To keep track of which users use how much and what kinds of computer resources
 - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - **Protection** involves ensuring that all access to system resources is controlled
 - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



Operating-System Operations

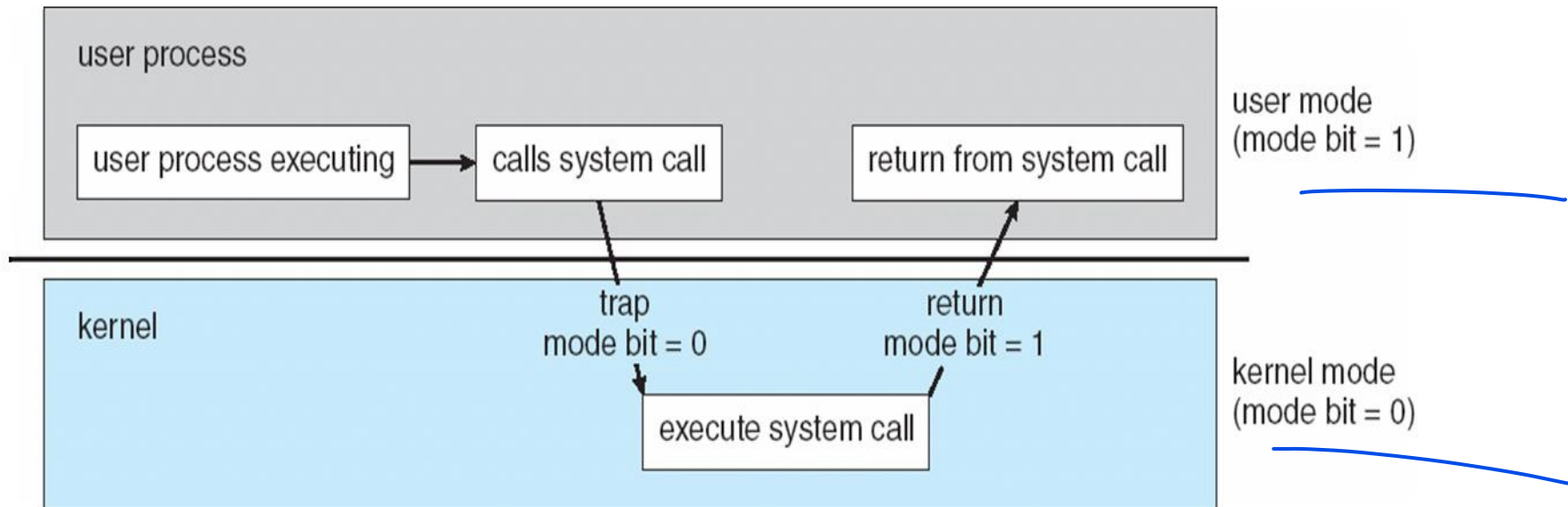
- **Interrupt driven** (hardware and software)
 - Hardware interrupt by one of the devices
 - Software interrupt (**exception** or **trap**):
 - Software error (e.g., division by zero)
 - Request for operating system service
 - Other process problems include infinite loop, processes modifying each other or the operating system

Operating-System Operations (cont.)

- **Dual-mode** operation allows OS to protect itself and other system components
 - **User mode** and **kernel mode**
 - **Mode bit** provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code
 - Some instructions designated as **privileged**, only executable in kernel mode
 - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
 - i.e. **virtual machine manager (VMM)** mode for guest **VMs**

Transition from User to Kernel Mode

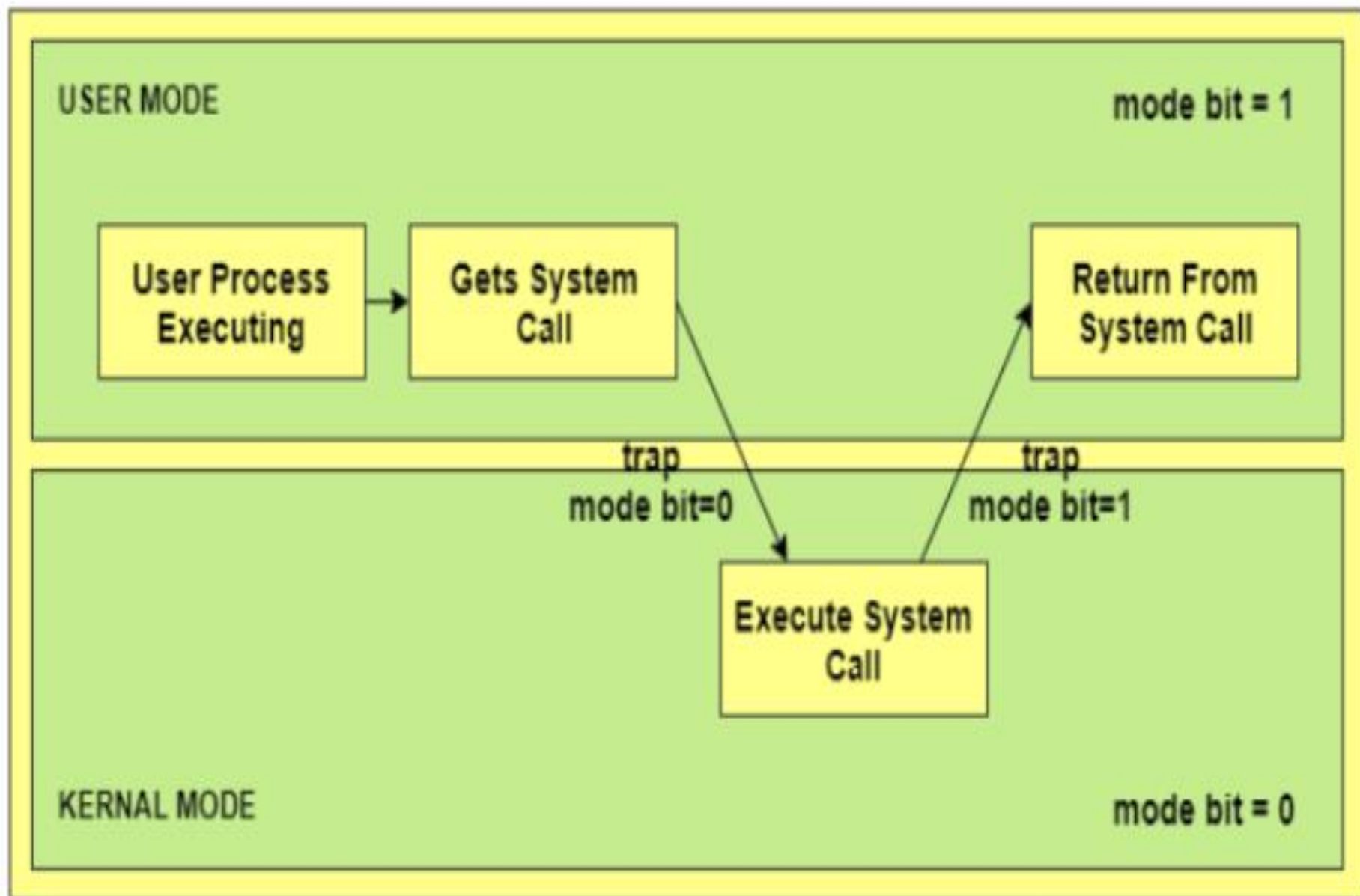
- Timer to prevent infinite loop / process hogging resources
 - Timer is set to interrupt the computer after some time period
 - Keep a counter that is decremented by the physical clock.
 - Operating system set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time



User Mode vs Kernel Mode

- **User Mode**
- The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs.
- The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

- **Kernel Mode**
- The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode.
- These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.
- The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mod



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
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads

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
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and 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

- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - 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

System calls

- The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions.
- They are also included in the manuals used by the assembly level programmers.
- System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

System calls

- In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel.
- System call **provides** the services of the operating system to the user programs via Application Program Interface(API).
- It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

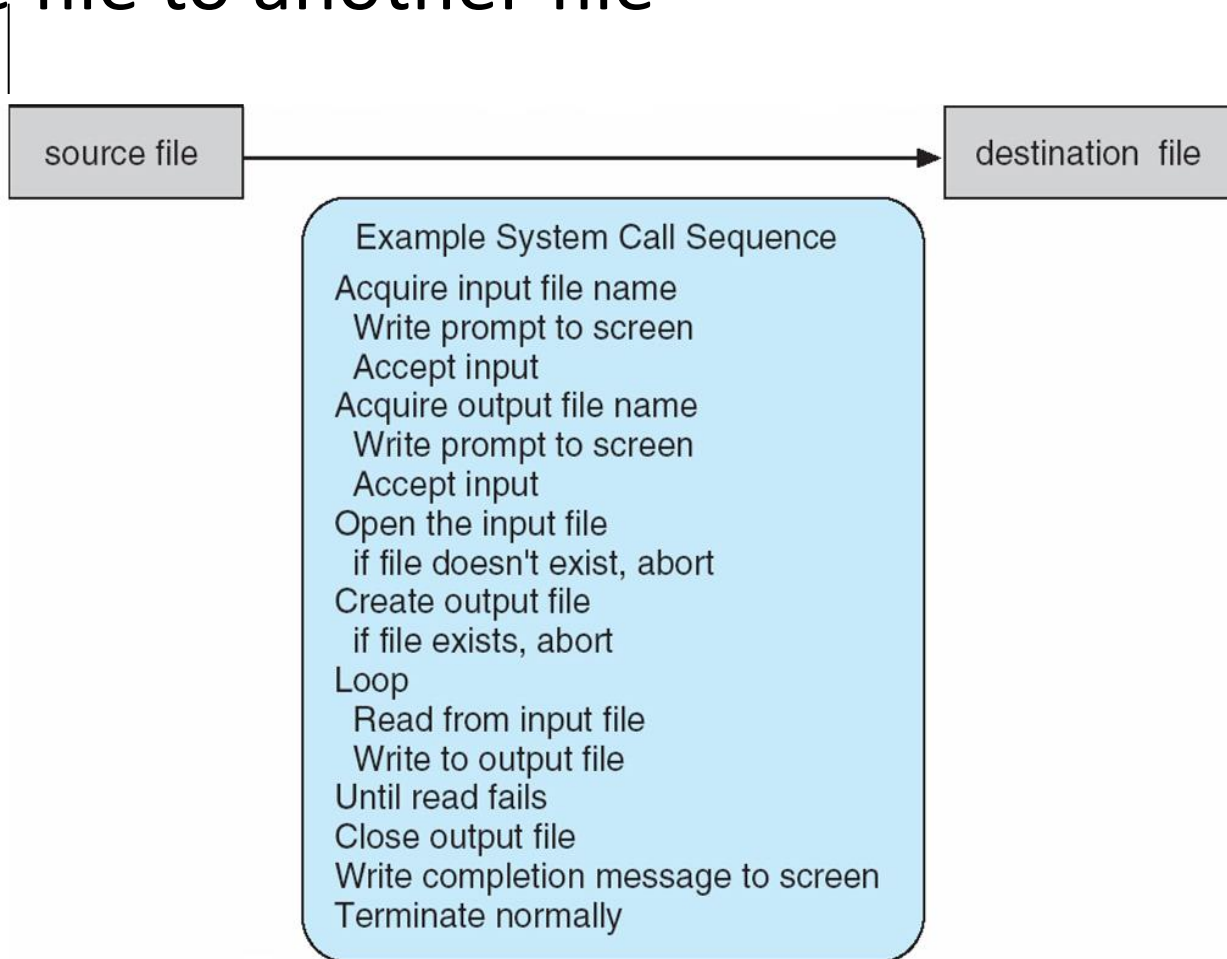
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout this text are generic

Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t  read(int fd, void *buf, size_t count)
```

return value	function name	parameters
-----------------	------------------	------------

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

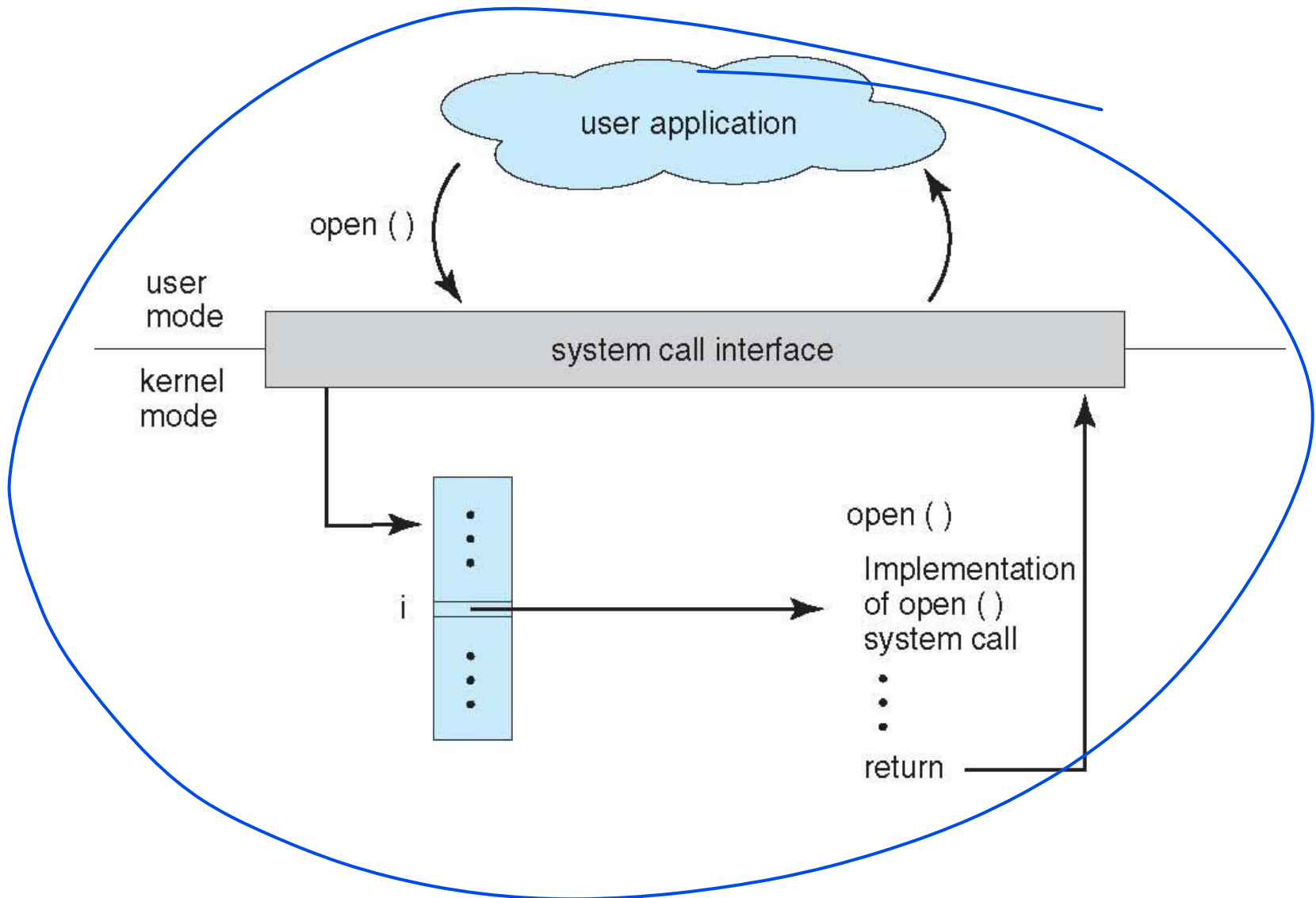
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

System Call Implementation

- Typically, a number associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most details of OS interface hidden from programmer by API
 - Managed by run-time support library (set of functions built into libraries included with compiler)

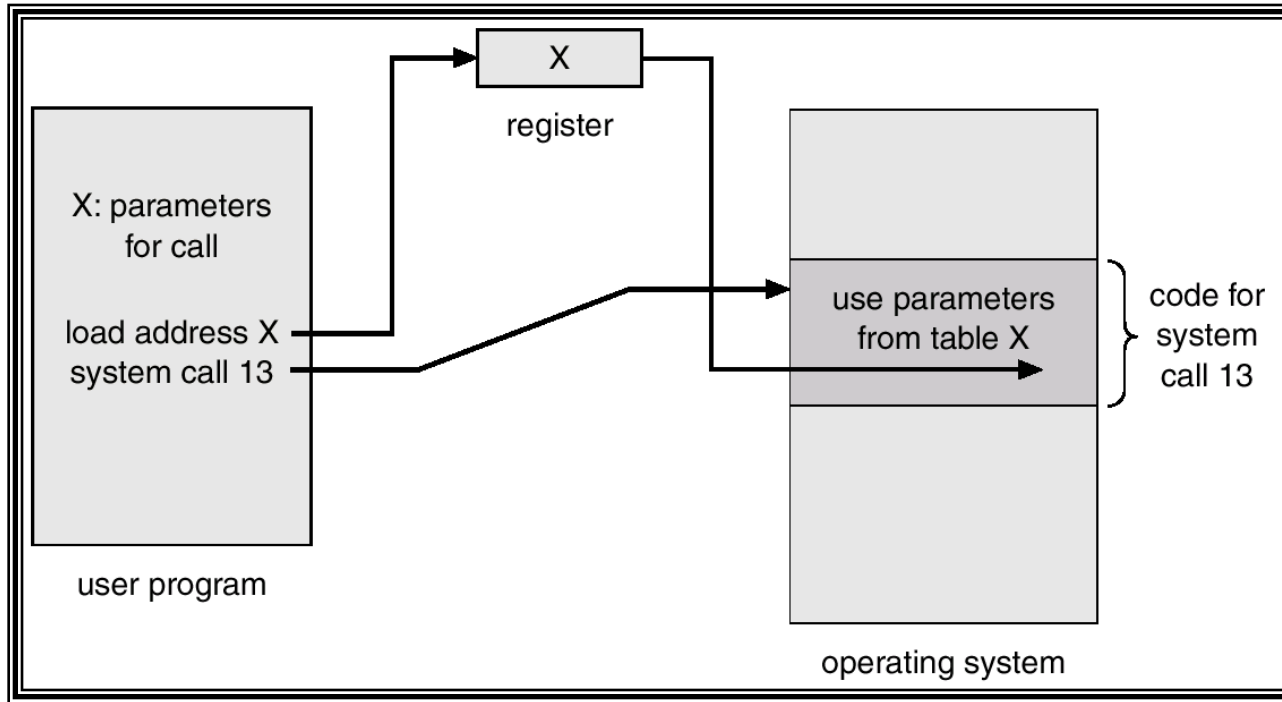
API – System Call – OS Relationship



System Call Parameter Passing

- Three general methods used to pass parameters to the OS
 - Simplest: pass the parameters in registers
 - 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 a register
 - This approach taken by Linux and Solaris
 - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
 - Block and stack methods do not limit the number or length of parameters being passed

Passing of Parameters As A Table



Types of System Calls

- **Process control**
 - End, abort, load, create, terminate, wait event, signal event, allocate and free memory
- **File management**
 - Create file, delete, open, close, read, write, get file attributes, set file attributes
- **Device management**
 - Request device, release device
- **Information maintenance**
 - Get time or date, set time, get process attributes, Set process attributes , get and set system data
- **Communications**
 - Send , receive message, transfer status information, create and delete communication connection

Types of System Calls

- Process control
 - create process, terminate process
 - end, abort
 - load, execute
 - get process attributes, set process attributes
 - wait for time
 - wait event, signal event
 - allocate and free memory
 - Dump memory if error
 - **Debugger** for determining **bugs, single step** execution
 - **Locks** for managing access to shared data between processes

Types of System Calls

- File management
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes
- Device management
 - request device, release device
 - read, write, reposition
 - get device attributes, set device attributes
 - logically attach or detach devices

Types of System Calls (Cont.)

- Information maintenance
 - get time or date, set time or date
 - get system data, set system data
 - get and set process, file, or device attributes
- Communications
 - create, delete communication connection
 - send, receive messages if **message passing model** to **host name** or **process name**
 - From **client** to **server**
 - **Shared-memory model** create and gain access to memory regions
 - transfer status information
 - attach and detach remote devices

Types of System Calls (Cont.)

- Protection
 - Control access to resources
 - Get and set permissions
 - Allow and deny user access

Services Provided by System Calls :

- Process creation and management
- Main memory management
- File Access, Directory and File system management
- Device handling(I/O)
- Protection
- Networking, etc

Examples of Windows and Unix System Calls

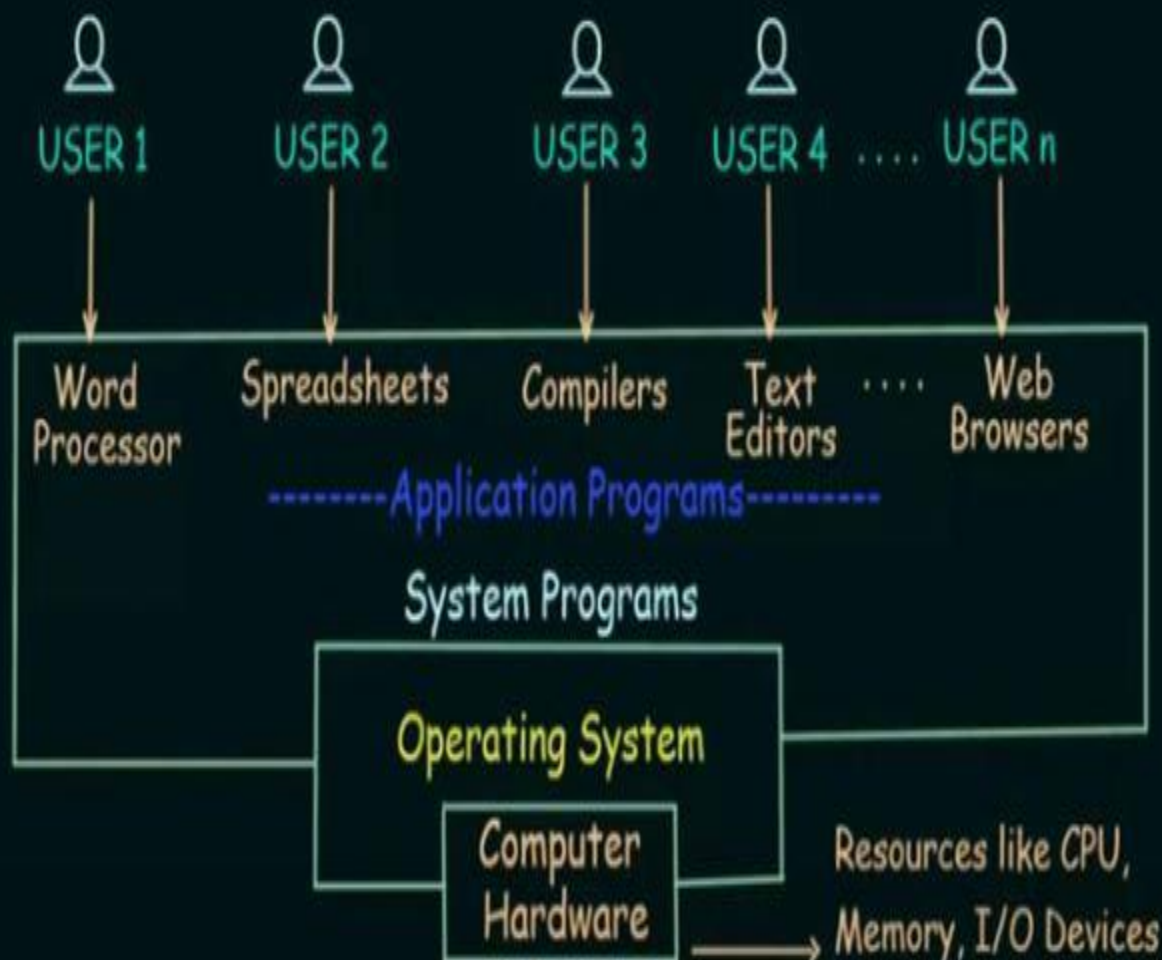
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Programs in Operating System

- **System Programming** can be defined as act of building Systems Software using System Programming Languages.
- According to Computer Hierarchy, one which comes at last is Hardware. Then it is Operating System, System Programs, and finally Application Programs.
- Program Development and Execution can be done conveniently in System Programs. Some of System Programs are simply user interfaces, others are complex.
- It traditionally lies between user interface and system calls.

System Programs

An important aspect of a modern system is the collection of system programs.



System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation
 - Status information sometimes stored in a File modification
 - Programming language support
 - Program loading and execution
 - Communications
 - Background services
 - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

- Provide a convenient environment for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
 - Some ask the system for info - date, time, amount of available memory, disk space, number of users
 - Others provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a **registry** - used to store and retrieve configuration information

System Programs (Cont.)

- **File modification**
 - Text editors to create and modify files
 - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs (Cont.)

- Some examples of system programs in O.S. are –
- Windows 10
- Mac OS X
- Ubuntu
- Linux
- Unix
- Android
- Anti-virus
- Disk formatting
- Computer language translators

Application Programs

- Word processors
- Game programs
- Spread sheets
- Data base systems
- Graphics programs
- Web browsers

Systems Programs

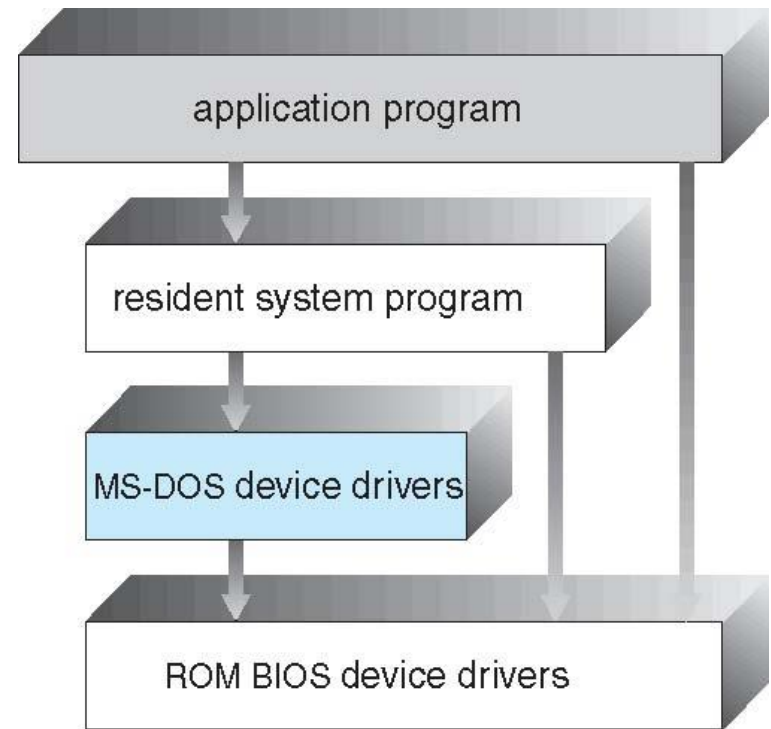
- Operating system
- Networking system
- Programming language software
- Web site server
- Data backup

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure – MS-DOS
 - More complex -- UNIX
 - Layered – an abstraction
 - Microkernel -Mach

Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
 - Not divided into modules
 - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



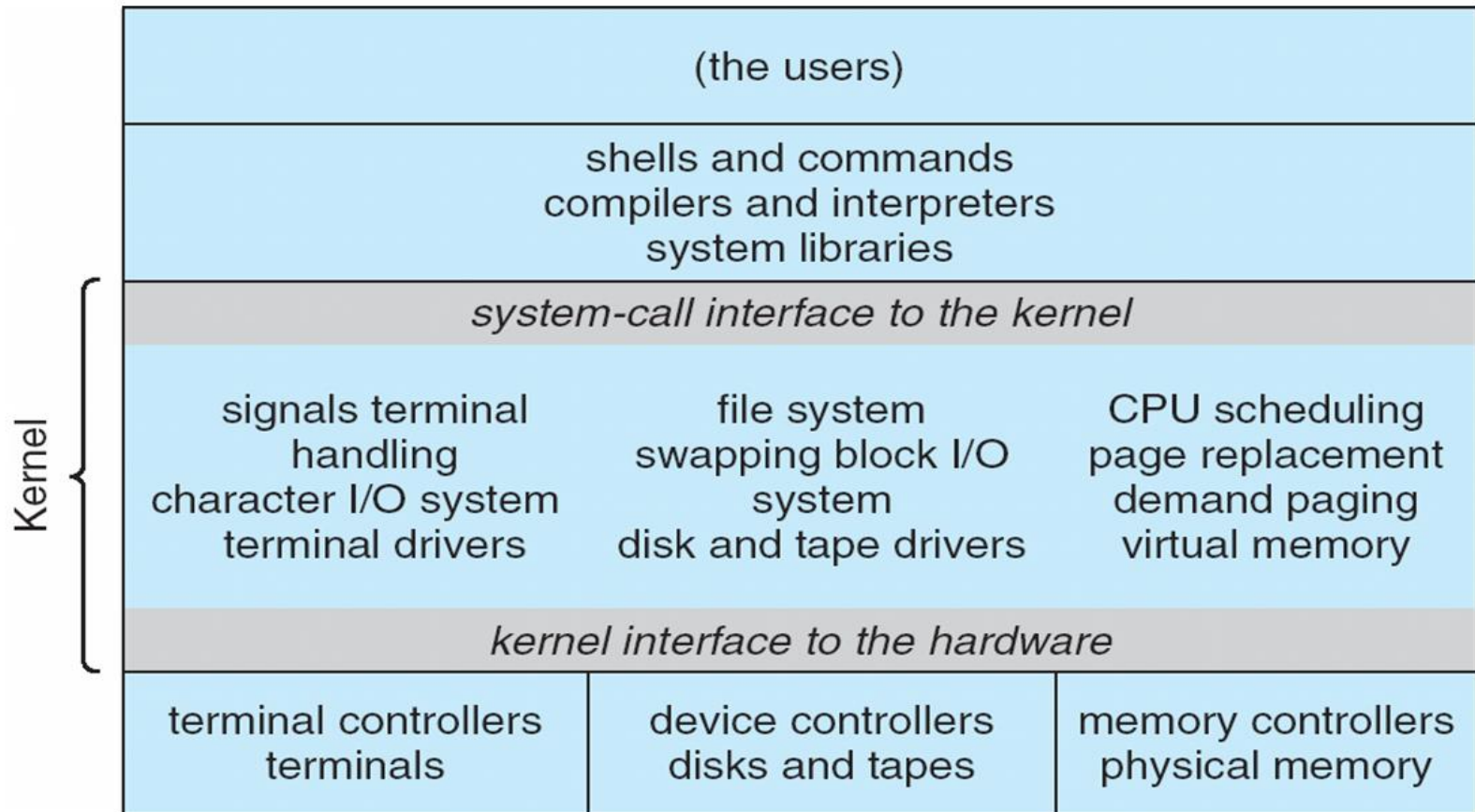
Non Simple Structure -- UNIX

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- Systems programs
- The kernel
 - Consists of everything below the system-call interface and above the physical hardware
 - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

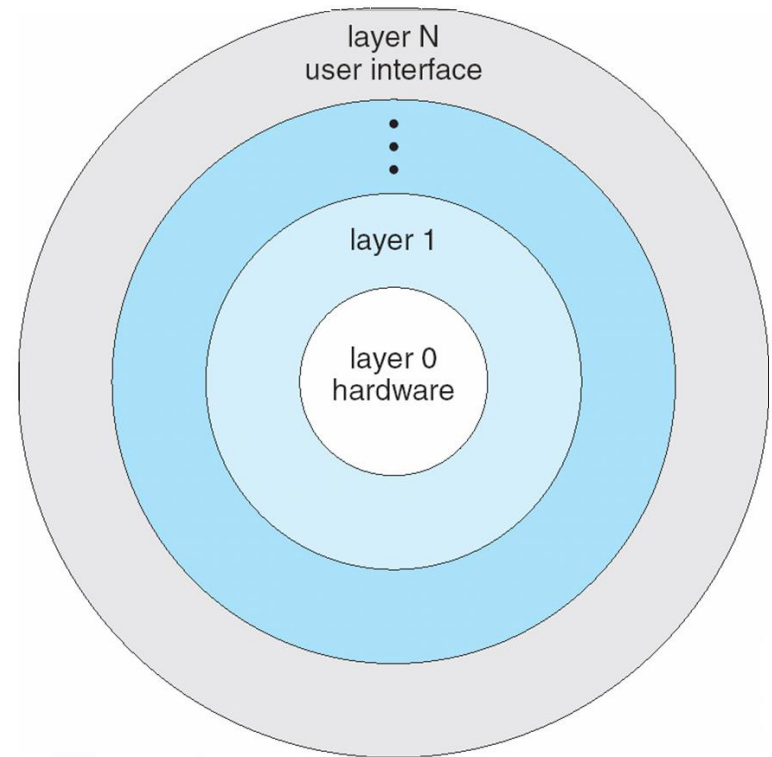
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach

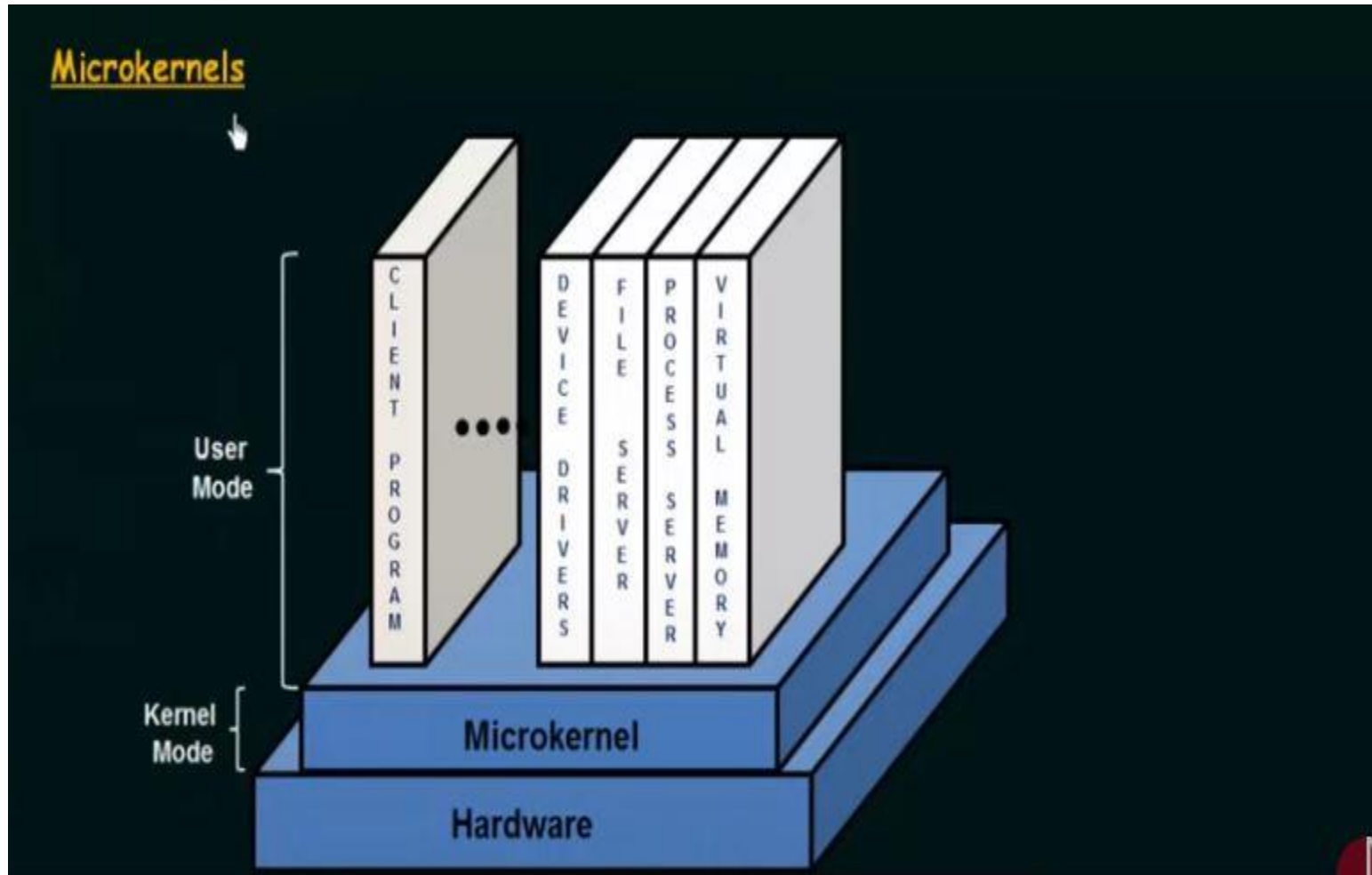
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



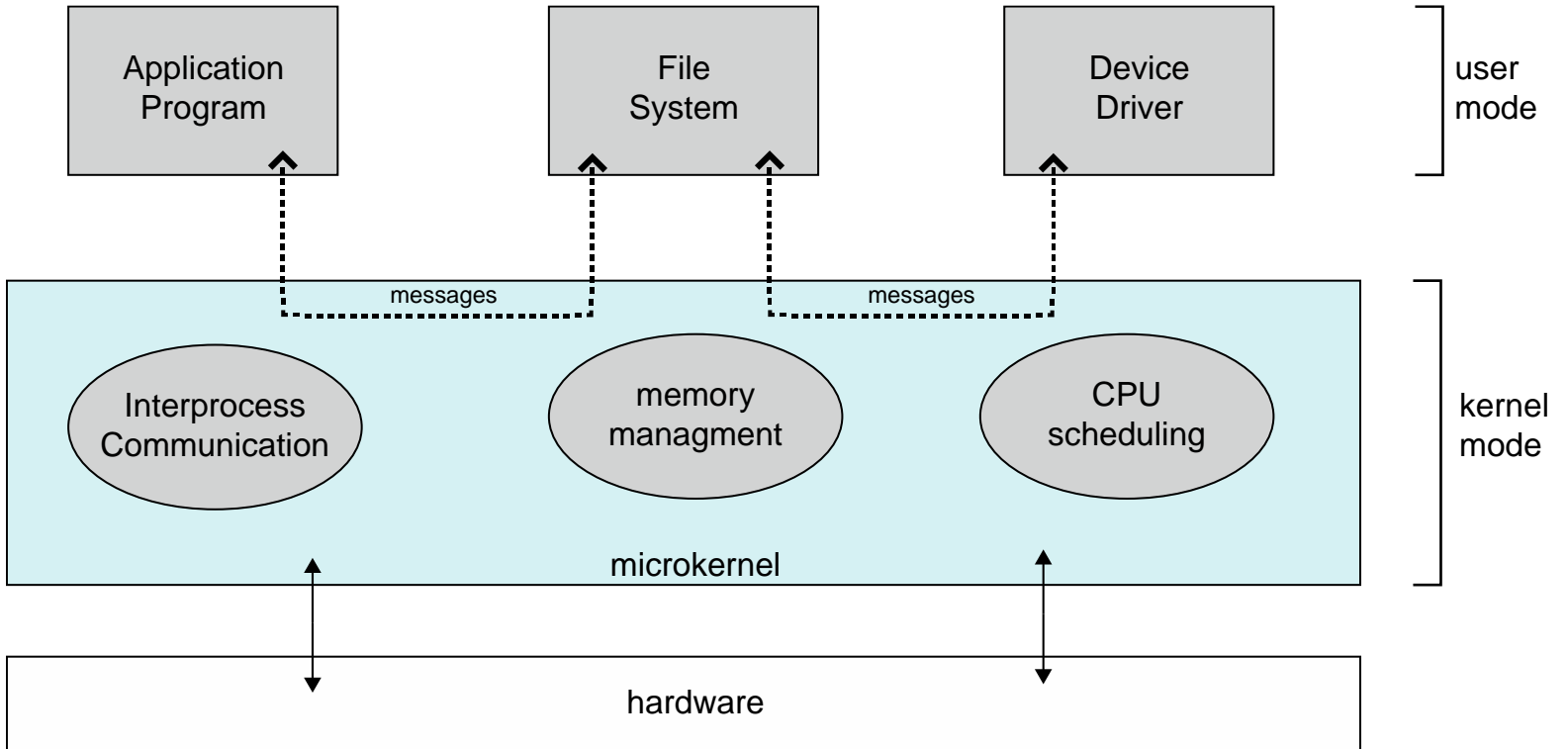
Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
 - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments:
 - Performance overhead of user space to kernel space communication

Microkernel System Structure



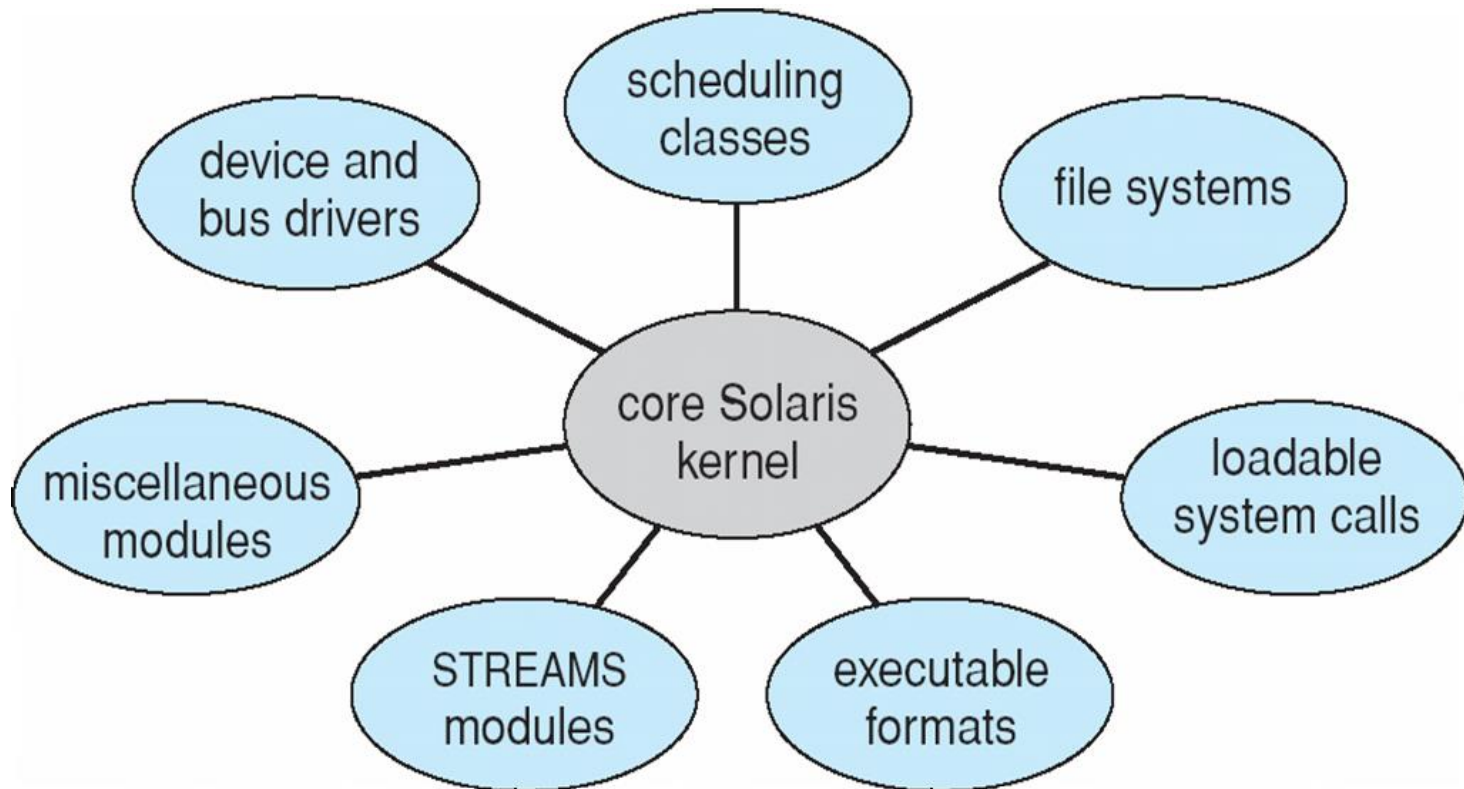
Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules**
 - Uses object-oriented approach
 - Each core component is separate
 - Each talks to the others over known interfaces
 - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
 - Linux, Solaris, etc

Solaris Modular Approach



Operating System Generation

- 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
 - Used to build system-specific compiled kernel or system-tuned
 - Can generate more efficient code than one general kernel

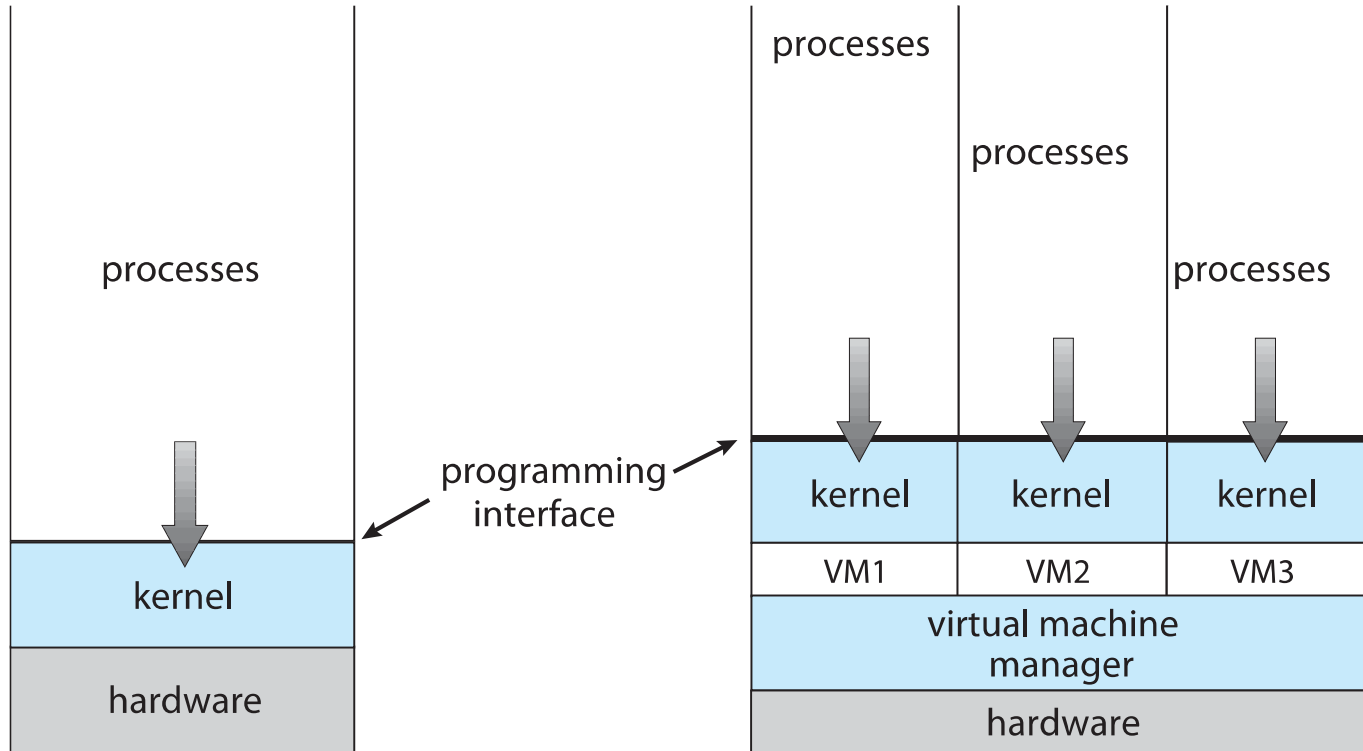
System Boot

- When power initialized on system, execution starts at a fixed memory location
 - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Virtual Machines

- Fundamental idea – abstract hardware of a single computer into several different execution environments
 - Similar to layered approach
 - But layer creates virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run
- Several components
 - **Host** – underlying hardware system
 - **Virtual machine manager (VMM)** or **hypervisor** – creates and runs virtual machines by providing interface that is *identical* to the host
 - (Except in the case of paravirtualization)
 - **Guest** – process provided with virtual copy of the host
 - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

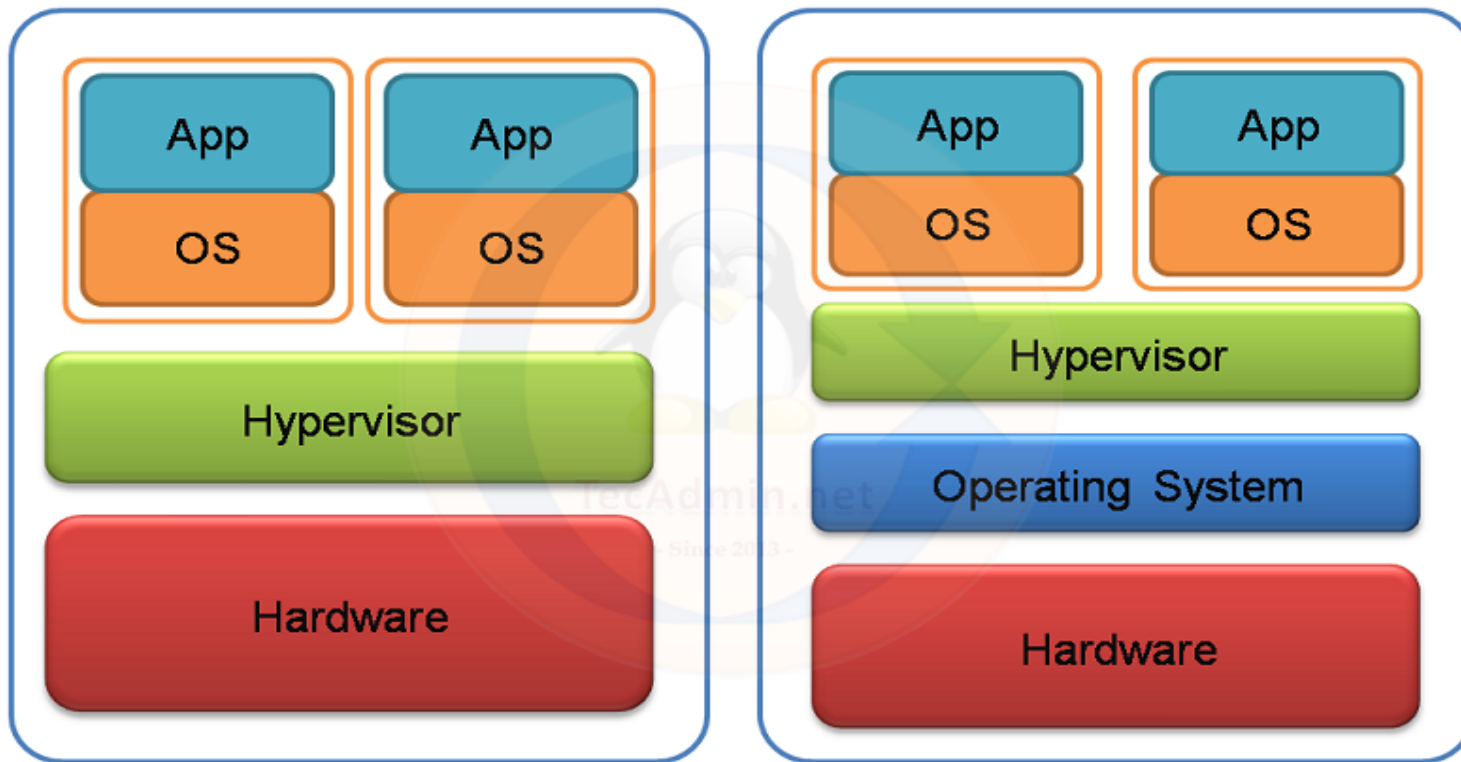
System Models



Non-virtual machine

Virtual machine

Virtualization



Type 1 (Bare Metal)
Virtualization

Type 2 (Hosted)
Virtualization

Type 1 Virtualization (Bare Metal)

- Type 1 hypervisor runs directly on the host's hardware to control the hardware and manage guest operating systems. Since it has direct access to physical resources, it's often referred to as a "bare-metal hypervisor."
- **Key Features:**
- **Performance:** Direct access to physical hardware results in better performance.
- **Isolation:** Each VM runs entirely isolated, ensuring that one VM's malfunction doesn't affect others.
- **Security:** The smaller footprint means a reduced attack surface compared to traditional OS.
- **Examples:** VMware vSphere/ESXi, Microsoft Hyper-V, and Xen.

Type 2 Virtualization (Hosted)

- **Type 2 Virtualization (Hosted)**
- **Definition:** A Type 2 hypervisor runs on a conventional operating system (the host OS) just like any other software application. It's often termed a "hosted hypervisor."
- **Key Features:**
- **Flexibility:** Easier to install and manage since it runs atop a standard OS.
- **Resource Overhead:** Consumes resources through the host OS.
- **Examples:** VMware Workstation, Oracle VirtualBox, and Parallels Desktop.

Paravirtualization

- Paravirtualization enables several different operating systems to run on one set of hardware by effectively using resources such as processors and memory.
- In paravirtualization, the operating system is modified to work with a virtual machine. The intention behind the modification of the operating system is to minimize the execution time required in performing the operations that are otherwise difficult to run in a virtual environment.
- Paravirtualization has many significant performance advantages and its efficiencies offer better scaling. As a result, it is used in various areas of technology such as:
 - Partitioning development environments from test systems
 - Disaster recovery
 - Migrating data from one system to another
 - Capacity management