



I am here to support you to understand and to be knowledgeable in the “Operating Systems”  
Through the ICS 431 course.





# Operating Systems 431

Weeks 1- 2

**Intensive Introduction of Operating Systems**  
**[Chapters 1, 2, and Parts of 3, 13]**

**Dr. Tarek Helmy El-Basuny**

# Outline

- In the last class:
- We presented the course information and some coordination issues.
- We agreed on the grading policies including the lab component.
- We discussed the topics to be covered during this course.
- We discussed the course objective and learning outcomes.
- From today's class,
- We will start the intensive introduction of the operating systems.

# During the Intensive OS Introduction

- We will introduce the computer System's Component,
- We will introduce the basic computer system organization and operation,
- We will introduce different definitions and views of the OS,
- We will introduce the major Operating Systems (OS) types,
- We will introduce the services an OS provides to users, processes, and other systems.
- We will introduce the common OS components,
- We will introduce the storage hierarchy structure in the computer systems,
- We will introduce caching features & problems and how the OS deals with?
- We will introduce the computing models and the OS support,
- We will introduce multiprocessor and multiprocessing systems,
- We will introduce the OS support to multi-processing systems,
- We will introduce real time and embedded OSs,
- We will introduce the interrupts, exceptions and how does the OS handle?
- We will introduce different I/O structures and techniques,
- We will introduce the security and protection issue of system's resources by the OS,
- We will introduce the system calls, the system programs how does the OS handle?
- We will introduce various ways of structuring an operating system,
- We will introduce how operating systems are installed, customized and how they boot?

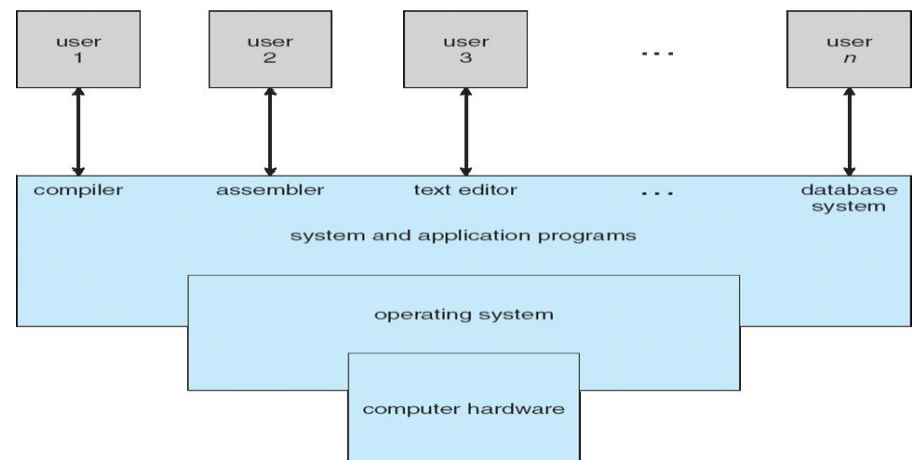
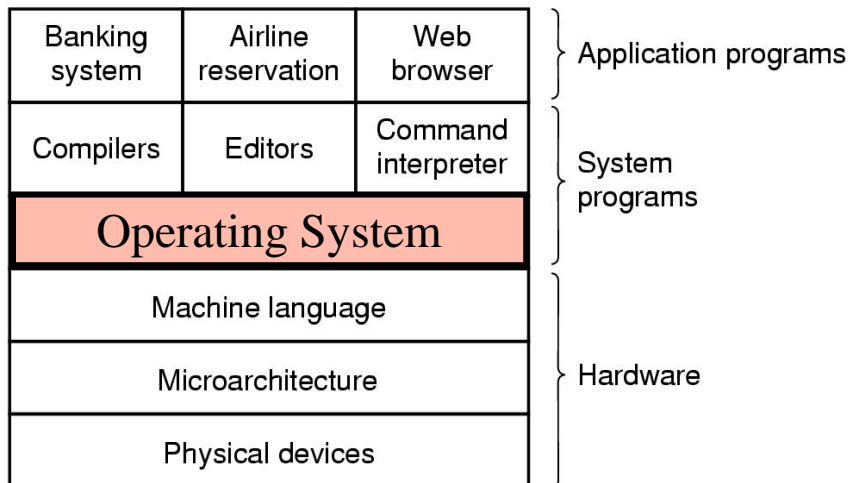
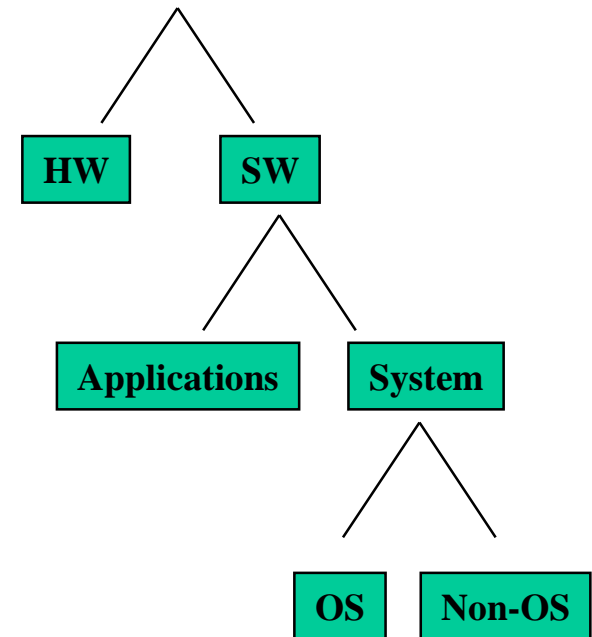
Covered by ch. 1, ch. 2  
& Parts of ch. 3 & ch. 13  
in the text book.

# Outline of Today's Class

- We will discuss:
  - Computer-System's Components & Organization,
  - Computer-System Operation,
  - Different Definitions of the OSs,
  - Benefits of the OS to the users and application programs,
  - Different types of the OSs,
  - The Major OS Issues,
  - Operating System Services,
  - Different views of the OS,

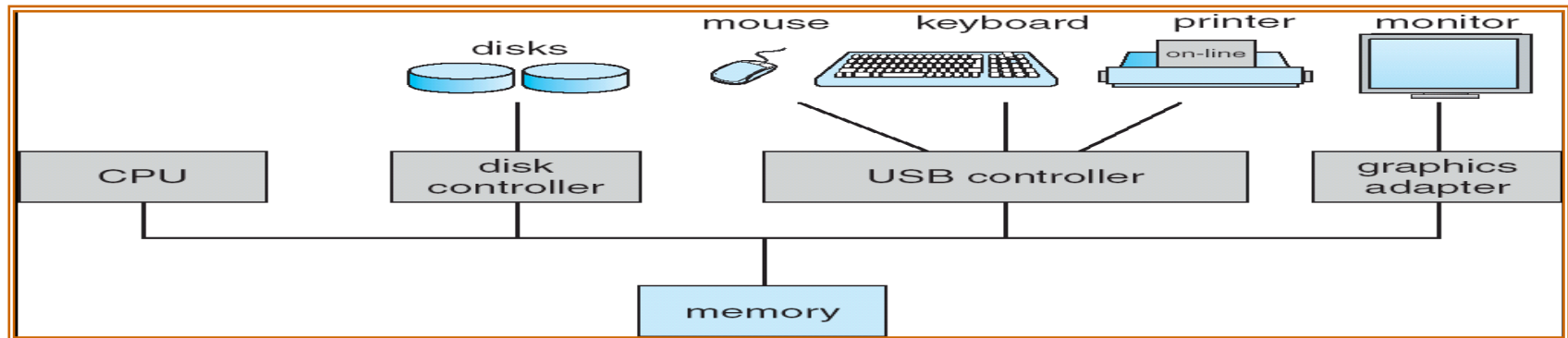
# Computer System's Component

- A computer system consists of
  - Hardware modules:** Provides basic computing resources (CPU, memory, I/O devices, etc.).
  - Applications programs:** Define the ways in which the system resources are used to solve the computing problems of the users (provide services) (compilers, database systems, video games, business programs).
  - Users** (people, other computers) run the programs.
  - Operating system:** Controls and coordinates the use of the hardware resources among various application programs for various users.



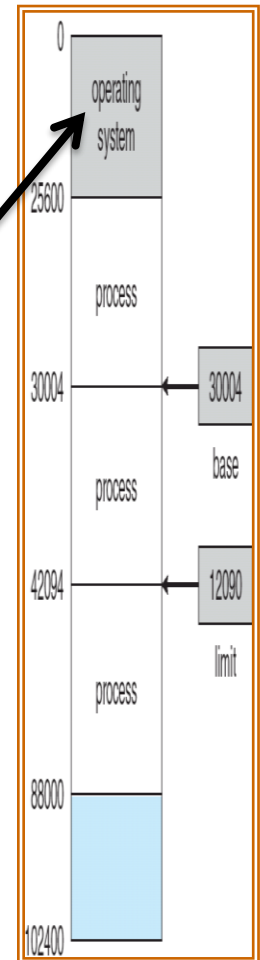
# Computer System Organization

- A computer system consists of a CPU, a memory and multiple I/O device controllers that are connected through a common bus.
- Device controllers interface with the I/O devices under the OS control.
  - One controller may handle several devices of the same type
  - Controllers have local buffers and a set of specific purpose registers.
  - The size of the local buffers varies from one device to another.
- Each device controller is responsible for moving the data between the device it controls and its local buffer under the OS control.
- I/O is from the device to local buffer of controller.
- The CPU can access information directly only if it is kept in main memory.
- I/O devices and the CPU can execute concurrently, competing for the memory access.
- To ensure orderly access to the memory, the OS with memory controller should synchronize access to the shared memory.



# Computer-System Operation

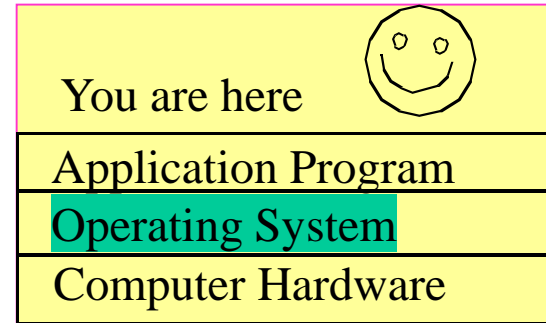
- After turning on a computer system, its main memory is empty: **how can the execution of programs be started?**
- A small part of the OS (**BIOS**) is built in the board (**ROM/EPROM**).
- **First**, the CPU executes the BIOS to validate the system resources.
- **Second**, the essential parts of the OS are loaded from the hard disk to the main memory and then executed.
- **Third**, the execution of a user's programs (**processes**) begin.
- Instructions are executed one after the other as specified in the running **program**.
- If an I/O operation is required, the program under execution is **temporarily suspended**, and the CPU switches to execute another **service program**.
- **Interrupts** are used to signal important events; they disrupt the flow of execution temporarily.
- The purpose of an **interrupt** is to transfer control from the current task to the OS.





# What is an Operating System?

- There is no a universally accepted definition for the OS.
- It's a program that mediates between the application programs and the hardware.



- The OS is a resource manager, where it supports:
  - Each program to get time and space with the resource,
- The OS allows application programs run peacefully
  - Enforces security policies,
  - Enforces safety measures,
  - Error detection & recovery,
  - Accounting,
  - Protection,

# Operating System Definitions

- The OS controls the execution of user's programs and operations of I/O devices.
- **The OS provides abstractions to simplify building applications: i.e.**
  - Files instead of “bytes on a disk”
  - Contiguous memory regions (**segments/pages**) instead of “bits in a RAM chip”.
- **The OS is an extended machine that:**
  - Presents user with a virtual machine, easier to use.
  - Hides the messy details which must be performed.
- An OS can be viewed **as an event-driven system that:**
  - Reacts to events as they occur by the user, I/O recourses, or the running processes.
- Write your own definition here, .....

# Why do we bother ourselves with OS?

- Software engineer's benefits
  - **Simplicity** of programming the applications.
    - Abstractions are **reusable** across many APIs.
    - See high-level abstractions (files) instead of low-level hardware details (device registers).
- User's benefits
  - **Safety**
    - Program “sees” own virtual machine, thinks it owns computer
    - OS **protects** processes from each other
    - OS **fairly multiplexes** resources across running processes
  - **Efficiency** (cost and speed)
    - **Share** one computer across many users
    - **Concurrent** execution of multiple programs on the same recourse.
    - Improve **responsiveness and throughput** (the amount of work that a computer can do in a given time period) **due to the concurrency**.

# The Operating System Types

- **Single Processor operating systems:** Designed to support concurrent executing of processes on a single processor machine.
- **Multiprocessor operating systems:** Designed to support large collection of concurrent executing of processes on multiprocessor machines (Multi-core).
- **Single Processing Operating Systems:** the OS allows a single process to run at a time.
- **Multi-Processing or Multi-Task Operating Systems:** the OSs that allow the execution of multiple processes at the same time/concurrently.
- **Multi-processing** can be of two types namely, **pre-emptive** or **co-operative**.
  - In **preemptive multiprocessing**, the operating system slices the CPU time and dedicates one slot to each of the processes. Unix-like operating systems such as **Solaris and Linux** support preemptive multiprocessing.
  - **Cooperative multiprocessing** is achieved by relying on each process to give time to the other processes in a defined manner.

# The Operating System Types

- **Server Operating System:** it provides runtime support for specialized, high-performance server applications, e-mail, HTTP, FTP, printer servers.
- **Multi-User/Time-Sharing Operating Systems:** The operating system allows multiple users to access a computer system concurrently through the sharing of time. Unix is an example of multi-user operating systems.
- **Single-User Operating Systems:** as opposed to a multi-user operating system, are usable by a single user at a time. Being able to have multiple accounts on a Windows operating system does not make it a multi-user system. Rather, only the network administrator is the real user. But for a Unix-like operating system, it is possible for more than one user to login at a time.
- **Single-User, Single-Task:** is designed to manage the computer so that one user can effectively do one thing at a time. The Palm OS for Palm handheld computers is a good example of a modern single-user, single-task operating system.

# The Operating System Types

- **Single-User, Multi-Tasking:** Allows a single user to run several programs at the same time. Windows and Mac platforms are both examples of that operating systems.
- **Multithreading Operating Systems:** that allow different threads of a running process to run concurrently.
- **Smart card/Embedded Operating System:** It comes pre-installed on a chip of the device (i.e. clock, microwave, ..) to provide highly stable functionalities. (i.e. Java Card OS, Aptura Smart Card OS). It is able to operate on:
  - Slow processors
  - Limited recourses due to the size

# The Operating System Types

- **Real-Time Operating Systems:** are those in which the **correctness** of the system depends **not only** on the **logical result** of computation, **but also** on the **time** at which the results are produced.
- Often used in dedicated applications such as **controlling scientific experiments**, medical imaging systems, industrial control systems, and some display systems. **There are two types:**
- **Hard real-time system**
  - Guarantees that critical tasks **be done on time**,
  - **All delays in the system must be bounded**,
  - **Secondary storage is usually limited or missing, why?**
    - Data stored in short term memory, or read-only memory (ROM),
    - **Useful for industrial applications.**
- **Soft real-time system**
  - **Deadline is important but not critical**,
  - Limited utility in industrial control or robotics,
  - Useful in real-time applications such as **multimedia, networking, advanced scientific projects, etc..**

# The Operating System Types

- **Cyber Shield Operating System:** designed to deliver verifiable security for conducting confidential online transactions, such as online banking, or accessing medical records. <https://www.youtube.com/watch?v=XuQdk1HQjNo>
- It is used to boot the PC with Cyber Shield-OS from a USB or CD, then conduct one's online banking or other sensitive Internet transactions, then when finished shut down Cyber Shield-OS and return to one's normal PC environment.
  - Cyber Shield-OS is not intended as a substitute for a general purpose operating system.
  - Cyber Shield-OS is not designed to support the PC's internal Wi-Fi network adapter. It only allows cable-based internet connection.
  - Cyber Shield-OS runs in PC RAM only. It is not designed to install on the PC's hard drive.
  - Cyber Shield-OS does not allow any changes to the original code, any attempts to install a plug-in will be discarded.
  - Cyber Shield-OS includes a built-in firewall that blocks LAN access during its operation.
  - The objective is to isolate the PC running Cyber Shield-OS from other PC's on the same LAN as the security status of the other PC's may be unknown.



# Outline of Today's Class

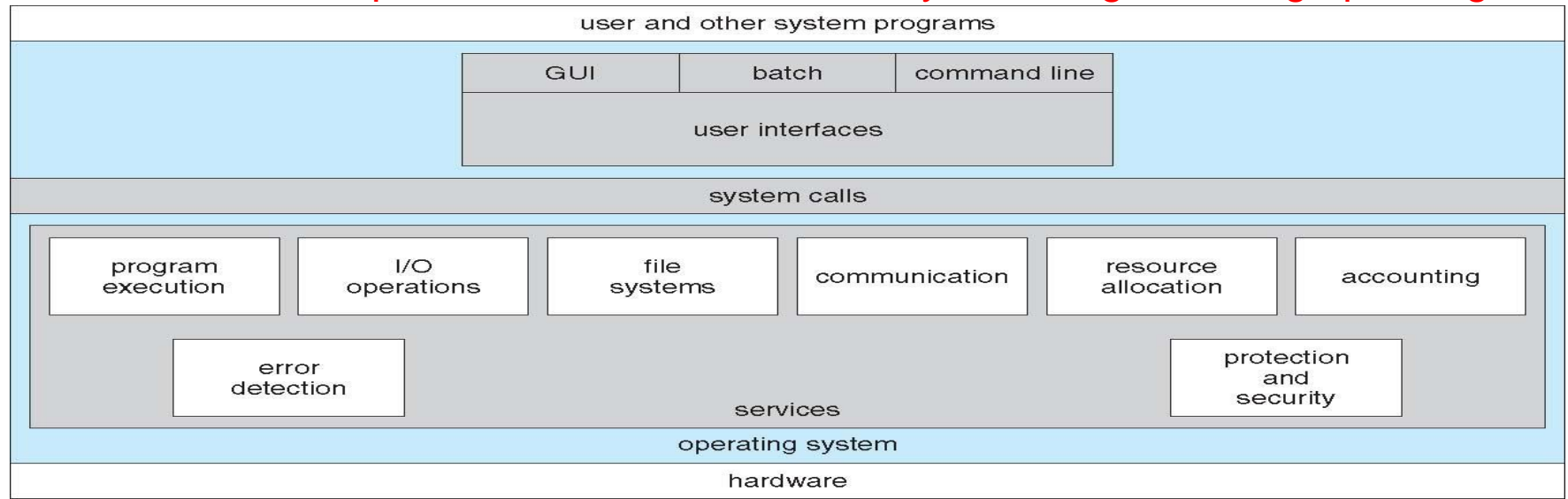
- We have discussed before:
  - Computer-System Components & Organization,
  - Computer-System Operation,
  - Different definitions of the OSs,
  - Benefits of the OS to users and application programs,
  - Different types of OSs,
  - The Major OS Issues,
- We will discuss today:
  - Operating System Services,
  - Different views of the OS,
  - Main Goals of the OS,
  - Why do we need to support Multi-processing and multiprocessors by OS?
  - Requirement of Multi-processing by OS.
  - Types of Multiprocessor systems.
  - Distributed, Network and Clustered systems from the OS point of view.
  - Computing Models and the OS support.
  - Main Components of the OS.
  - CPU and Main Memory Protection by the OS.
  - Storage Hierarchy in the computer system and why?

## The Major OS Issues

- As we are going to study the OS, we should think about the followings:
- **Structure:** How are the OS components organized?
- **Sharing:** How are resources shared across users/processes?
- **Concurrency:** How are concurrent processes (**computing and I/O**) created and controlled?
- **Naming:** How are resources named?
- **Communication:** How do processes exchange information, including across network?
- **Security:** How is the integrity of the OS and its resources ensured?
- **Accounting:** How do we keep track of a resource usage, and perhaps charge for it?
- **Performance:** How do we make it all go fast?
- **Reliability:** What happens if something goes wrong (**either with hardware or with a program**)?
- **Extensibility:** Can we add new features to the OS?
- **Scalability:** What happens as demands or resources increase?
- **Distribution:** How do multiple computers interact with each other?

# Operating System Services

- **Program execution:** the OS allows the system to load a program into memory (where, when, and how long?) and to execute it.
- **I/O operations:** Since user programs cannot execute I/O operations directly, the OS must provide some means to perform I/O.
- **File-system manipulation:** the OS allows programs to read and write files and directories, to create and delete them, to search for them, to list file Information, etc.
- **Communications:** the OS allows processes executing either on the same computer or on different systems tied together by a network to exchange information. Implemented via shared memory or through message passing.



# Operating System Services

- **Error detection and recovery:** OS needs to be continuously aware of possible errors.
  - Errors may occur in the CPU, memory, I/O devices, or in user's 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.
- **Resource allocation:** Allocating the resources fairly to concurrently running processes.
- **Accounting:**
  - Keep track of resources utilization, which processes use?
  - How much and what kinds of resources for account billing or for accumulating usage statistics?
- **Protection and Security:** Ensure that all access to system resources is controlled and defense of the system against internal and external attacks.

## There are 5 Views of the OS

Your view of an OS depends on who you are and your interest:

- The hardware engineer view
- The operating system designer's view
- The application programmer's (Software Engineer) view
- The end-user's view
- The system administrator's view

## The Hardware Engineer View

- The HW engineer interest revolves around:
- How to make the booting process more faster by writing efficient **Bootstrap program**.
  - A bootstrap program is the first code that is executed when the computer system is started .
  - Typically stored in ROM or EPROM, generally known as **Firmware/BIOS**.
  - Initializes all aspects of system.
  - Loads operating system kernel and starts execution of processes.
- How does the OS verify the devices and efficiently use them.
- How to make the interactions between the HW and the OS more efficient.

## The OS Designer's View

- The OS designer's interest revolves mainly about the OS itself, its internal **structure**, **efficiency**, **performance**, etc..
- How can we make the OS more friendly?
- How can we add more functionality to upgrade the OS?
- How do we debug the Os to make it more reliable, scalable, etc..

## The Application Programmer's View

Since the OS is like a library with a well defined set of API's. The application programmer's interest revolves mainly around:

- What abstractions are available from the OS?
- How well is the API structured?
  - Not too low-level, or high-level.
- How portable is the interface?
  - Don't want to keep rewriting the same program for each new OS release.



## The End-User's View

The end user's interest revolves mainly around:

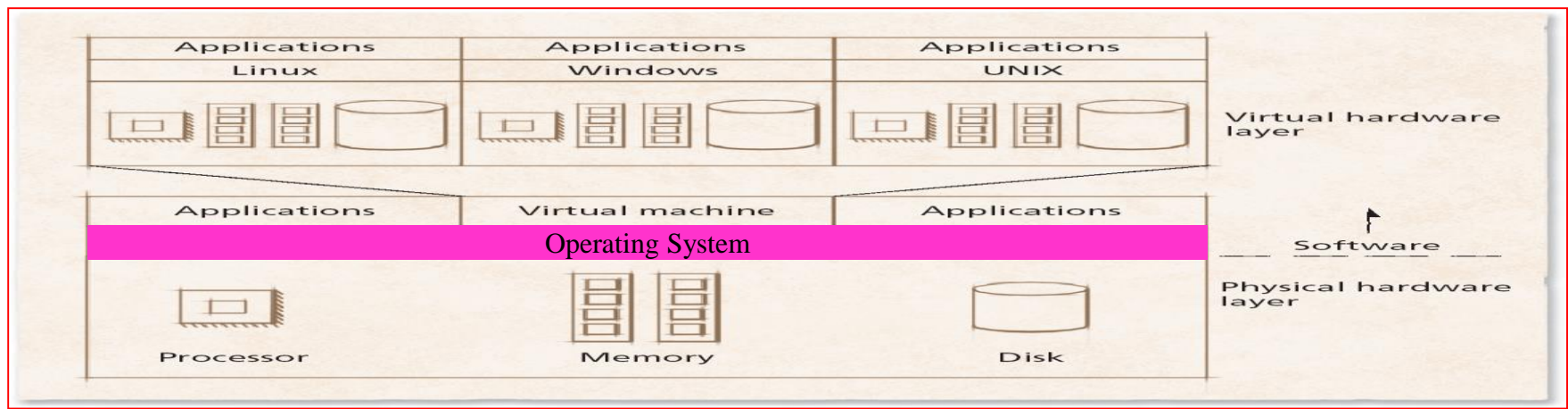
- End users care about their applications, not the OS.
- The OS should be transparent and friendly.
- The OS must not crash easily.
- The OS must protect their applications and privacy.

## The System Administrator View

- Since, the OS is a program that allows the efficient and fair usage of resources. The system administrator's interest revolves mainly around:
  - How easy is it to install a new software on that OS?
  - How does OS track usage of resources for accounting?
  - How does OS support the security aspects?
  - How does OS support the fairness among running processes and users?

# Main Goals of the OS

1. **Maximize Resources Utilization:** CPU cycles, Memory, Disk, etc. must be managed efficiently **to maximize overall system performance.**
2. **Resource Abstraction:** OS transforms the devices into more abstract and easily used devices. **In this way, it is building an extended machine.**
3. **Fairness:** the OS fairly multiplexes resources across programs.
4. **Maximize the throughput:** The amount of work that a computer can do in a given time period.
5. **Virtualization:** The operating system creates virtual copies of the processor and the memory . Supports different OS to communicate and exchange information. Gives each user the appearance of an unshared resource.



## Advantages & Disadvantages of Virtualization

### Advantage:

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
- Allows you to install different platforms on the same machine.

### Disadvantage:

- The isolation, however, permits no direct sharing of resources.
- That means the resources utilization will be low and throughput will be low as well.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine.

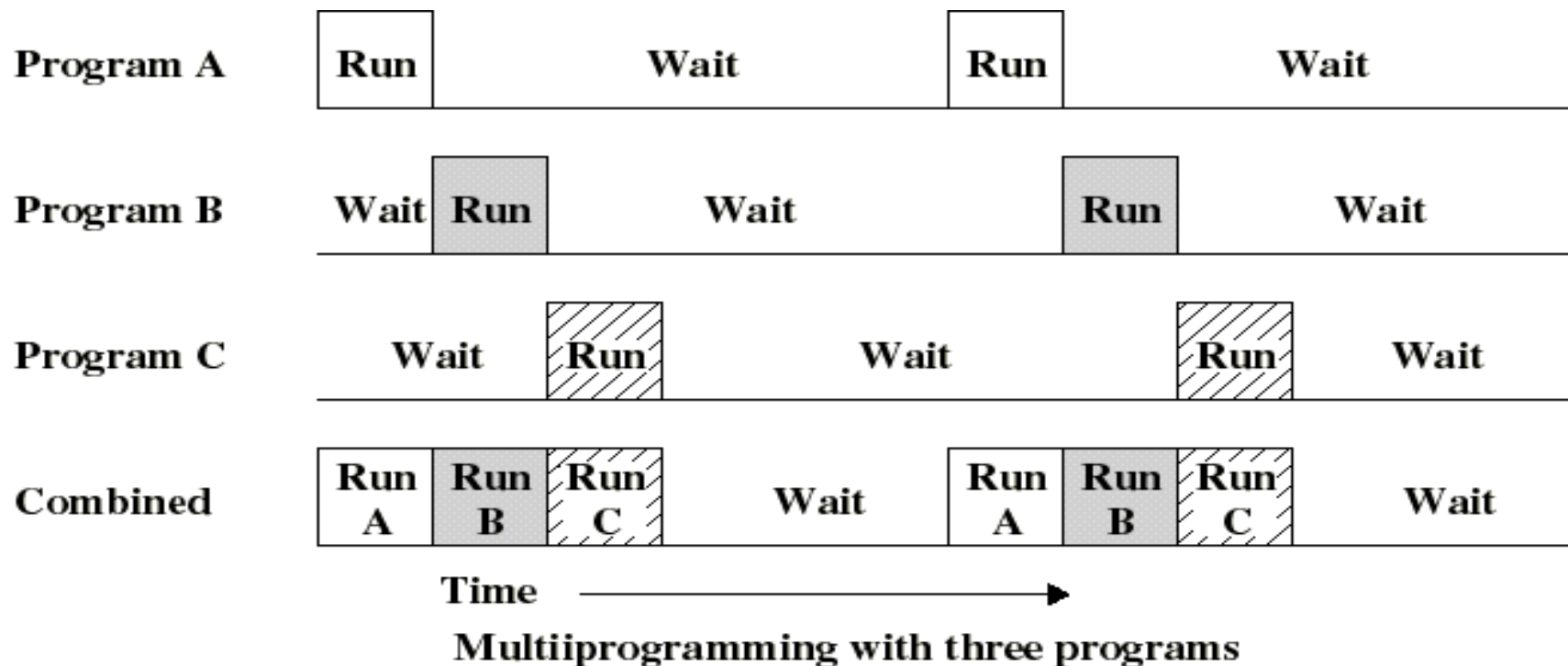
## Supporting of Multi-processing by OS: **why?**

- **Multiprocessing means** allowing more than one active process to be executed simultaneously through the interleaving between I/O and CPU bound instructions:
  - An **I/O-bound** process will have many short CPU bursts.
  - A **CPU-bound** process will have many long CPU bursts.
- **Why:** to maximize the throughput and increase system's resources utilization.
- **Since I/O bound Instructions are very slow in execution** compared to CPU bound instructions, **for example:**
  - If there is a CPU with a speed of 400 MHz (**400 million cycles/second**)
  - If each CPU-bound instruction takes 10 cycles/instruction
  - That means the CPU can execute about 40 million CPU-bound instructions/**second**
  - Or the CPU can do  $(40 \times 10^6) / 10^3 = 40,000$  CPU-bound instructions/**millisecond**.
  - If one I/O-bound instruction like (**Reading 1 disk block**) takes 20 ms
  - In a time to read one disk block, CPU can do  $20 * 40,000 = 800,000$  **CPU-bound instructions !!**. **The result:**
    - Poor CPU utilization when only one process is executing at a time
    - Low throughput.
    - Low utilization of the system resources in general



## Multiprocessing OS

- If several processes can execute concurrently, then the CPU can switch to another one whenever one is waiting for the I/O to complete and that maximizes the CPU utilization which will increase the throughput and resources utilization as well.



## Requirements for Multi-processing OS

- **Multiprocessing** OS runs multiple processes at once by interleaving I/O-bound and CPU-Bound instructions of the running processes.
  - But that needs **asynchronous** I/O devices [**once the I/O starts, the control returns to the OS without waiting for I/O completion**] and also needs to know when the I/O devices are done:
    - Interrupts, or Polling or DMA
- It needs HW support for memory management.
  - Keeping track of memory usage,
  - Process swapping,
  - Dynamic memory allocation,
  - **Base and Limit registers** to protect processes from each other.
- It needs SW support to manage resources conflict, **i.e.**
  - CPU scheduling (which process is to be run next),
  - Deadlock handling,
  - Memory allocation,
  - Management of I/O resources,
- The goal is to maximize the system's throughput and to increase system's resources utilization.
  - Perhaps at the cost of response time.

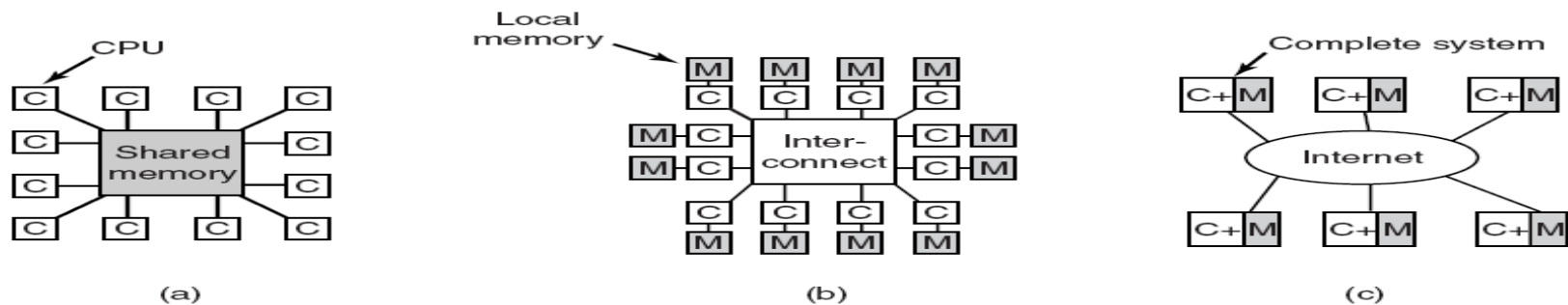
## Multiprocessing Modes by the OS

- CPU is shared among a number of processes based on a certain policy (i.e. SJF, FCFS, ..) while one process is waiting for the I/O, another one can use the CPU.
- CPU is shared among processes based on a pre-defined time interval.
  - Instead of waiting until the process gives up voluntarily the CPU in multi-processing environment, take it away at regular intervals (time-slices).
  - Divide CPU's time equally or non equally among the processes. If a process is truly interactive (e.g. editor), then it can be given more time.
- Advantages of both strategies:
  - CPU is kept busy and resources utilization is maximized .
- Disadvantages of both strategies:
  - Hardware and OS became significantly more complex for handling CPU scheduling, handling deadlock, protection, memory management, virtual memory, etc.



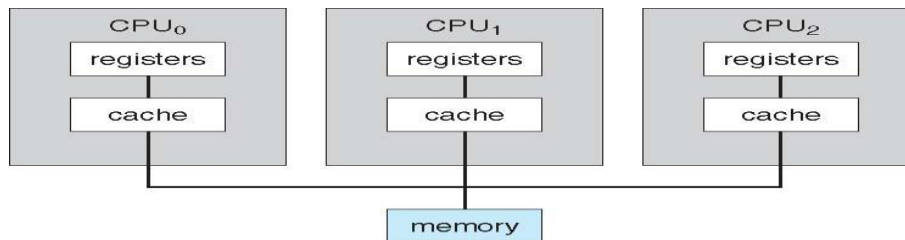
# Supporting of Multiprocessor Systems by the OS, **Why?**

- Three main advantages of supporting Multiprocessor system by the OS:
  - Increased throughput (more CPUs = more work in less time),
  - Economy of scale (saves money, CPUs share I/O resources),
  - Increased reliability (provides redundancy and fault tolerance),
- Types of Multiprocessor systems:
- **Tightly coupled system** (Multi-core system, more than one CPU in close communication).
  - Processors connected at the bus level and share both memory and clock,
  - Communication usually takes place through the shared memory,
- **Loosely coupled system**
  - Processors do not share memory and clock (**Networked machines**)
  - Processors interconnected via a high speed communication system,
  - Communication usually takes place through communicating messages,

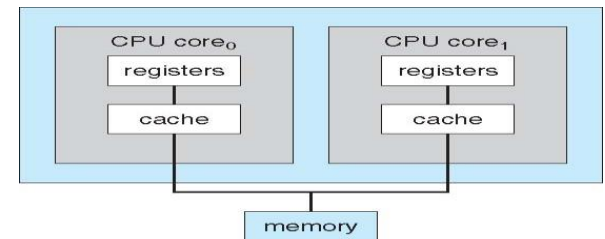


# Types of Multiprocessor Systems

- Symmetric Multi-Processing (SMP)
  - Each processor runs the same copy of the operating system
    - All processors are peers
  - It dynamically partitions tasks across the processors, manages the ordering of task completion, and controls the sharing of all resources among the cores.
  - Many processes can run simultaneously without performance deterioration
    - Data sharing should be carefully coordinated for efficiency
  - Most modern OSs (Windows, Unix, Linux) support SMP.
- Asymmetric Multi-Processing (AMP)
  - Master processor runs the OS, schedules and allocates work to slave processors.
  - Each slave is assigned a specific task by the master processor.
  - More common in extremely large systems (Distributed or Clustered systems).



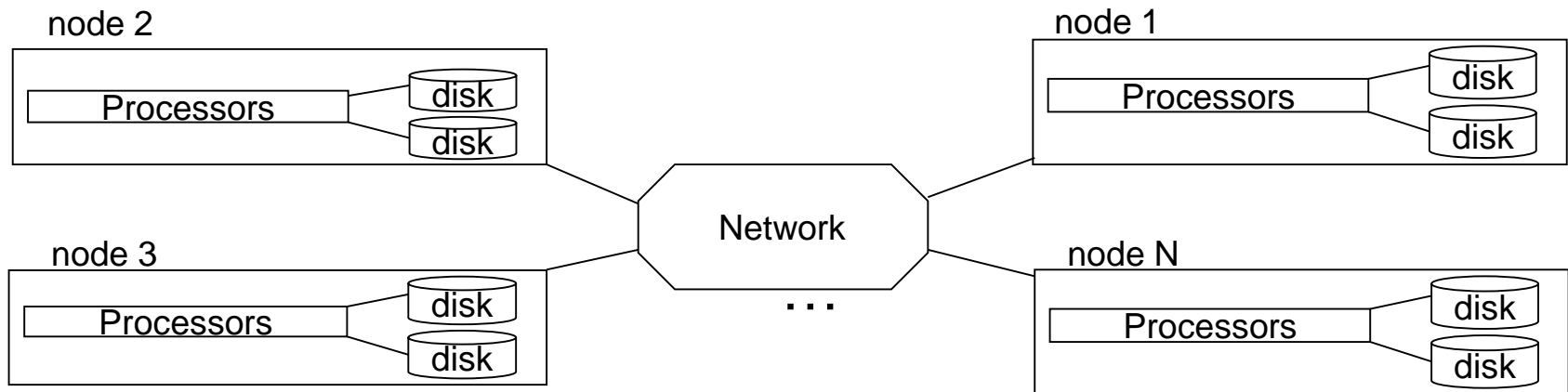
Symmetric Multiprocessing Architecture



A Dual-Core Design

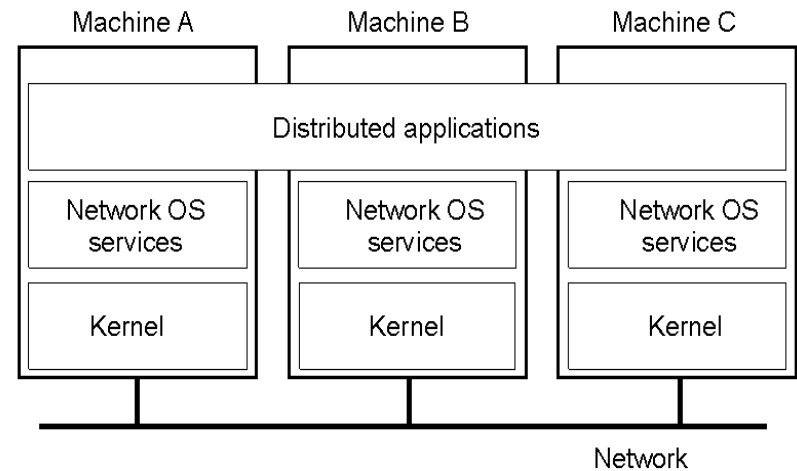
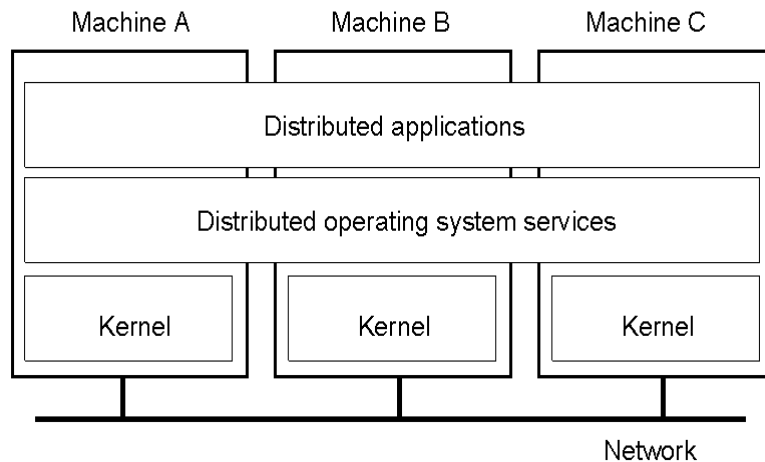
# Multiprocessor Systems: **Distributed Systems**

- A **distributed system** is: A collection of independent computers that geographically distributed and appear to its users as a single coherent system.
- Runs on a cluster of machines **with no shared memory**.
- **Distributed OS:**
  - A network of computers run a shared OS.
  - It provides the user with transparent access to the resources (including the **hot resources, i.e. CPU, main Memory**) of multiple machines.
  - Gives the impression, there is a single operating system controlling the network.
  - Distributed OS is a kind of **Cooperating OSs** (each OS has its own tasks but helping each other to achieve their tasks by sharing their hot resources)



- Advantages of distributed systems:
  - Resource sharing
    - Sharing and printing files at remote sites
    - Processing information in a distributed database
  - Computation speedup: Load sharing
  - Reliability: Detect and recover from site failure, function transfer, reintegrate failed site.
  - Distributed OS Supports communications between jobs:
    - Inter-Process Communication (IPC)
    - Message passing, shared memory
- Requires
  - A single global IPC mechanism,
  - A global protection mechanism,
  - Identical process management and system calls at all nodes,
  - Common file system at all nodes.

## Multiprocessor Systems: **Distributed** vs. **Network Systems**

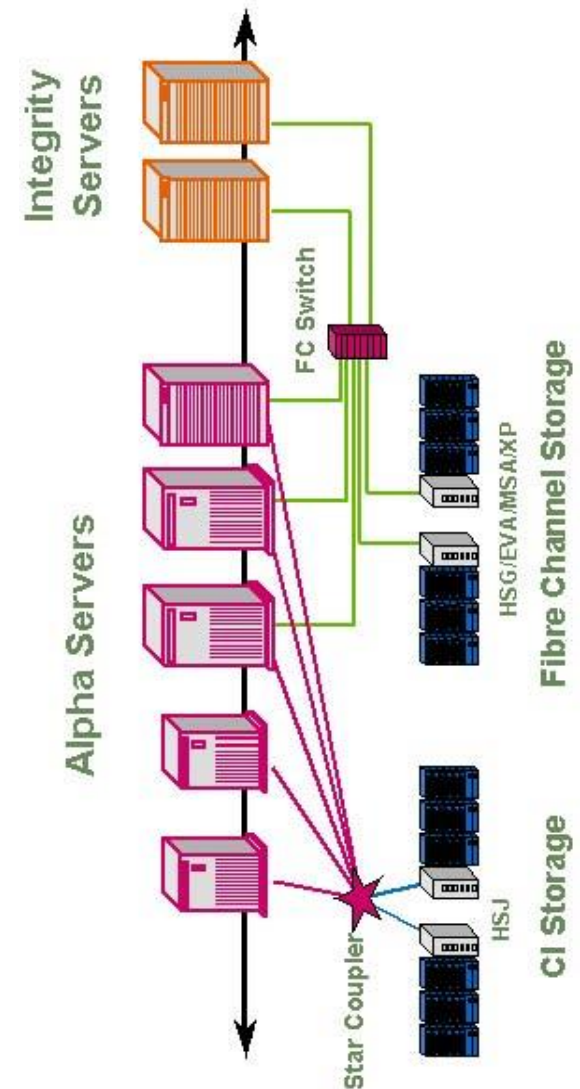


- The user is not aware of the multiple CPUs.
- Each machine runs a part of the DOS.
- Gives the impression there is a single OS controlling the network.
- Network is mostly transparent – it's a powerful virtual machine.
- Applications interact with single service layer.

- The user is aware of the existence of multiple CPUs.
- Each machine has its own private Operating System and runs independently from other computers on the network.
- Applications need to know about location of different services.
- Provides mainly file sharing.

## Multiprocessor Systems: **Clustered Systems**

- A computer's cluster is a group of networked/distributed computers working together closely to achieve a computational task.
- Clustered OS is a kind of **Collaborating OSs** (all OSs cooperate to achieve the same task by sharing their hot resources)
- Clustered systems share storage and closely linked through a local area network (LAN).
- Clustered systems support high availability, each node can monitor one or more nodes in the LAN.



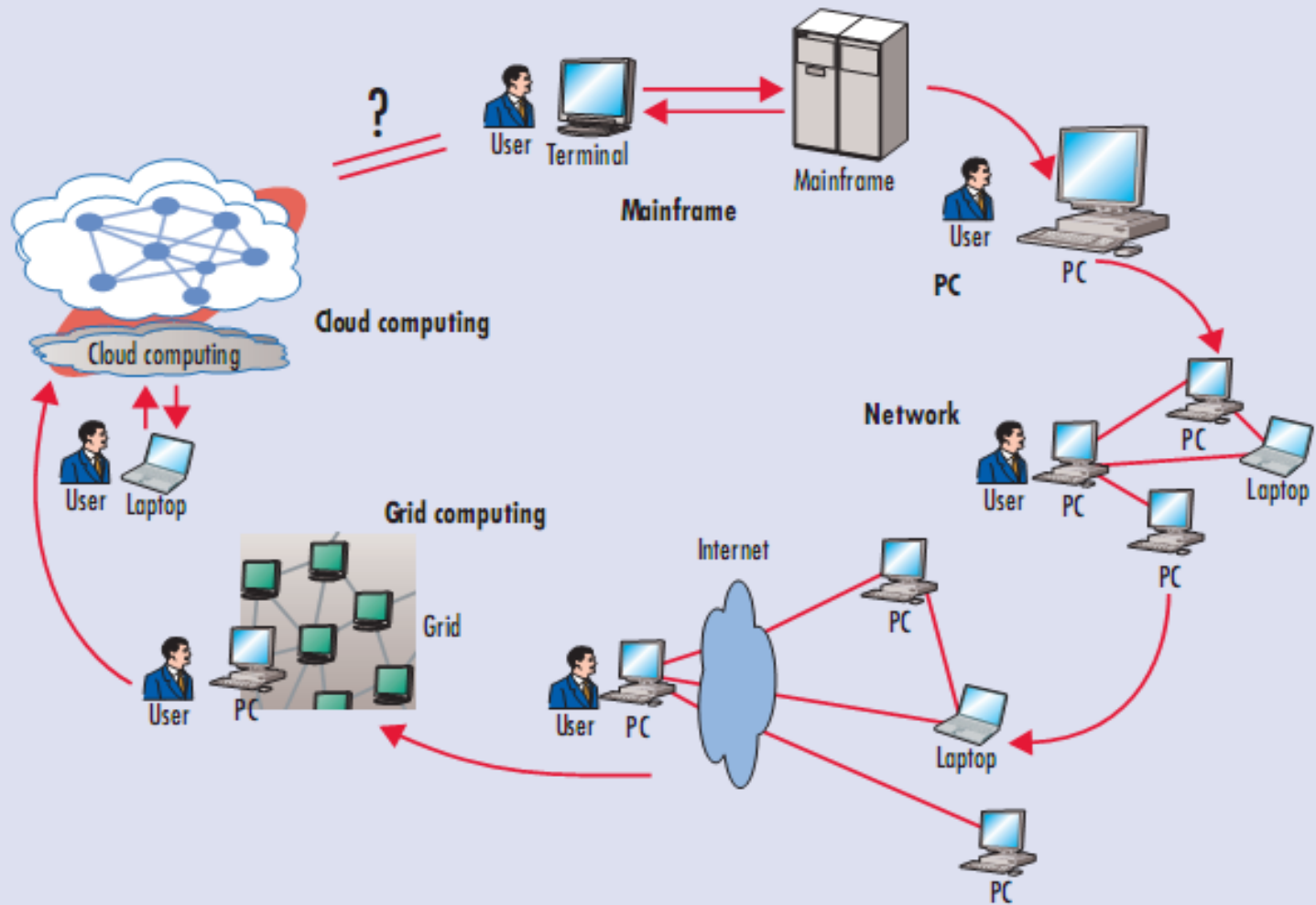
- **Possible clustering schemes:**

- **Symmetric mode** (two or more nodes running applications and monitoring each other).
- **Asymmetric clustering** (one is in hot standby mode while another is running applications; switches to backup if the active one fails).

- **Cluster systems categories:**

- **High-Availability clusters** are implemented primarily for the purpose of improving the availability of services that the cluster provides. They operate by having redundant nodes, which are then used to provide service when system components fail.
- **Load-Balancing clusters** is when multiple computers are linked together to share computational workload or function as a single virtual computer. Logically, from the user side, they are multiple machines, but function as a single virtual machine.
- **Computing/Grid clusters** are used primarily for computational purposes, rather than handling IO-oriented operations such as web service or databases.

# Computing Models Shift

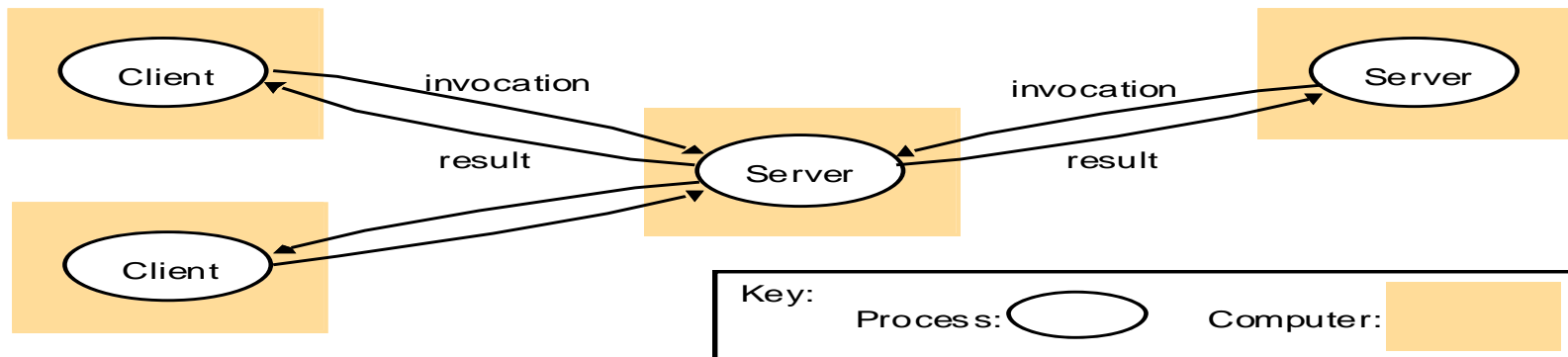
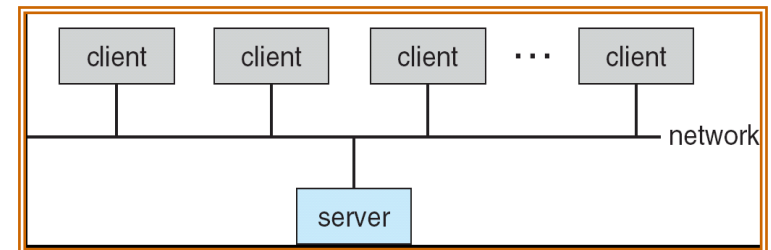
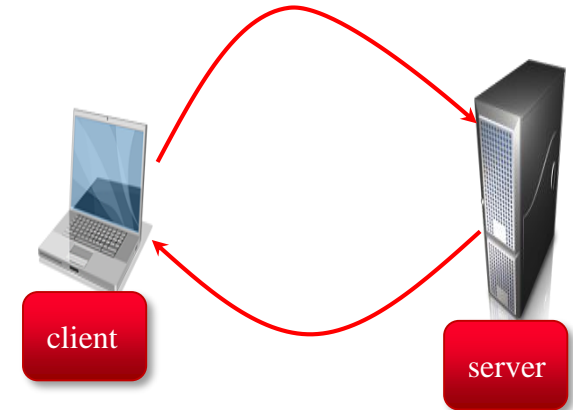


- The main objective here is to discuss how the operating support of each computing model will be different.



# Client-Server Computing Model

- **In the Client-Server model:** The client sends a request to a server, and the server responds with the information requested.
  - Popular models, e.g. **Telnet, FTP, etc.**
- **Client-Server Computing**
  - **Client** terminals are PCs
  - **Servers**, responding to requests sent by **clients**
    - **Server computer** provides an interface to the clients to request services (**i.e. database**).
    - **File-server** provides interface for clients to store and retrieve files.



# Client-Server Computing Model

- Advantages:

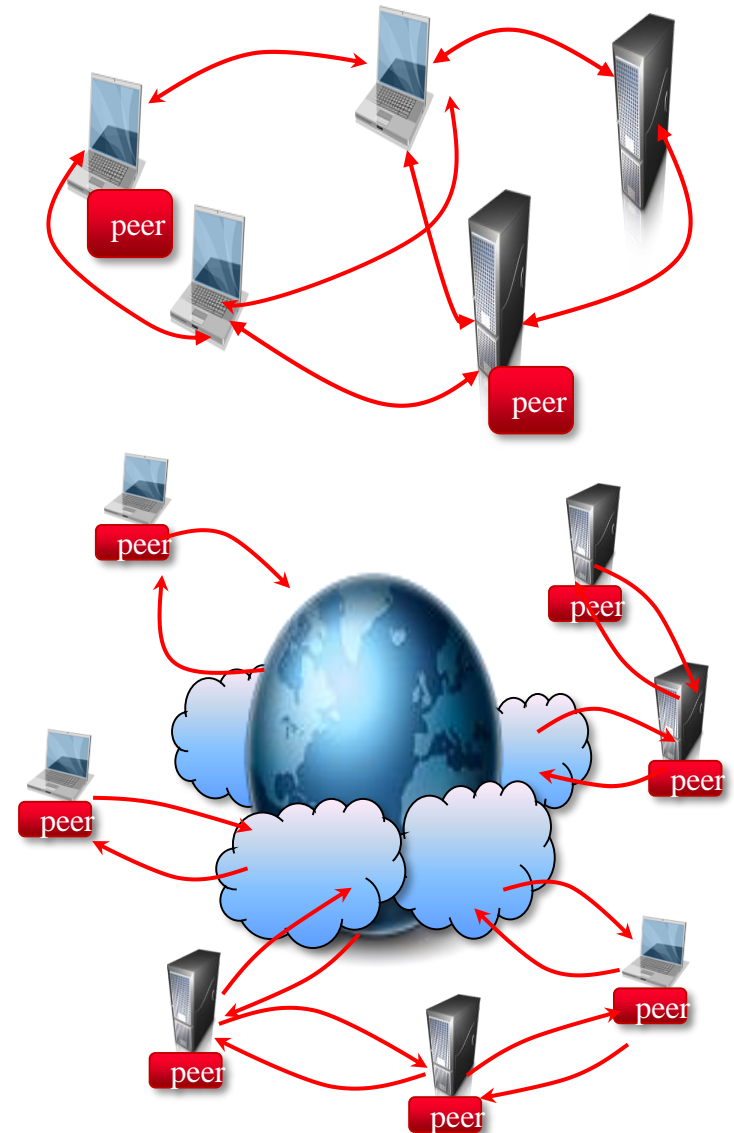
- **Centralization** - access, resources, and data security are controlled through the server.
- **Scalability** - any SW can be upgraded when needed.
- **Flexibility** - new technology can be easily integrated into the system.
- **Interoperability** - all components (clients, network, servers) work together.
- **Accessibility** - server can be accessed remotely and across multiple platforms.
- **Lower total costs**

- Disadvantages:

- As the number of simultaneous client requests to a given server increases, the server can become **overloaded**.
- The client–server paradigm lacks the **robustness**, if the server fails, clients' requests cannot be fulfilled.

# Peer-to-Peer Computing Model

- P2P is a model of distributed system:
  - All nodes are considered peers (**identical and there is no distinguish between clients and servers**)
  - May each act as client, server or both.
  - Peers make a portion of their resources, such as CPU time, Memory space, disk storage etc., directly available to other Peers without the need for central coordination by servers.
  - **A node joins a P2P network must:**
    - Registers its service with central lookup service on the network, **or**
    - Broadcasts a request for a needed service and responds to requests for service via **discovery protocol**.
  - Examples include Napster and Gnutella as file sharing systems.



- **Advantages**

- No central point of failure

- All peers in P2P network are the same.
    - Data and computation is decentralized/not centralized.
    - Peers are **autonomous**, they have full control.

- Scalability

- Since every peer is alike, it is possible to add more peers to the system and scale it to larger networks.

- **Disadvantages**

- Decentralized coordination

- How to keep the global state consistent?
    - Require distributed coherency protocols.

- All node's load may not be equal, **Load unbalance**.

- Computing power, bandwidth have an impact on overall performance.

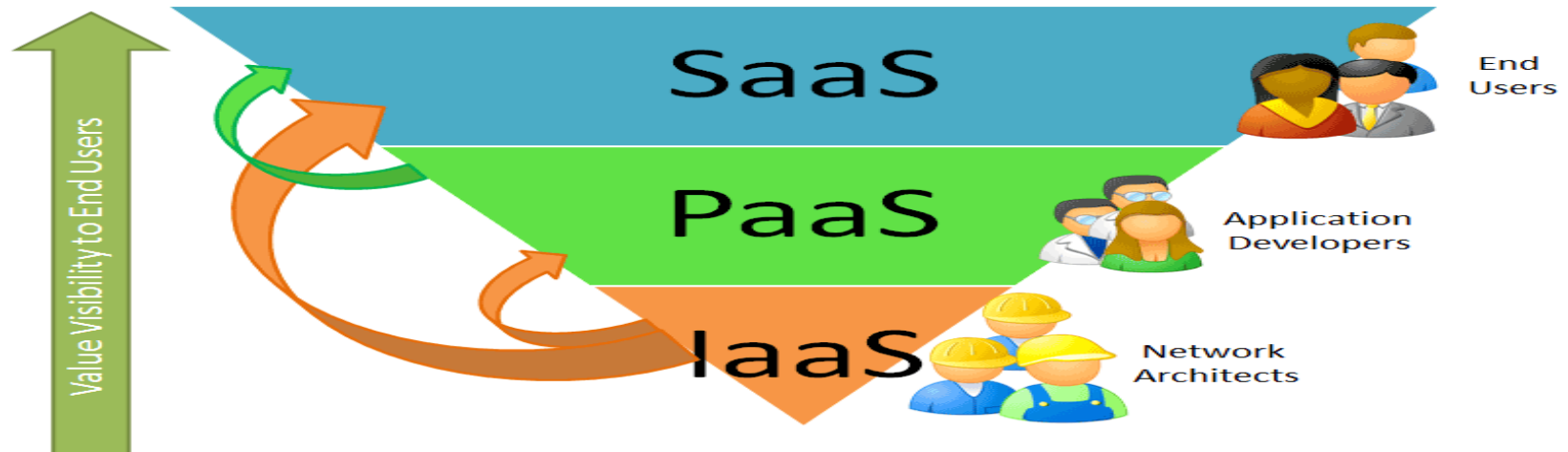
# Web/Cloud Computing Model

- Web has become everywhere and available all the time (Ubiquity)
- Computing-intensive applications need more computing and better performance.
- **Cloud computing** is an Internet-based computing, where shared resources, software and information are provided to computers and other devices on-demand, like the electricity grid.
- **Cloud computing** is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly released with minimal management effort.
- **Cloud computing** is a way of managing large numbers of highly virtualized resources such that, from a management perspective, they resemble a single large resource. This can then be used to deliver services with flexible scaling.
- **Cloud Providers** – Amazon, Google, e.g. Google owns more than 20,000 servers/data center.



# Cloud Computing Frameworks

- So we will just buy everything from the cloud.



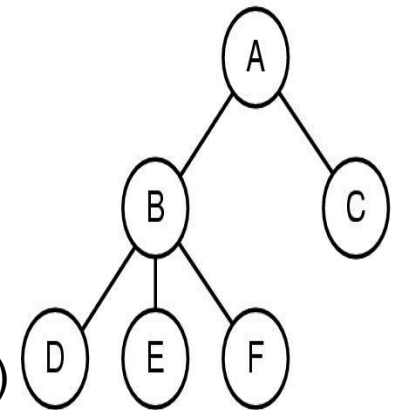
## Common OS Components

- Process Management Component (PMC)
- Memory Management Component (MMMC)
- File Management Component (FMC)
- Mass Storage Management Component (MSMC)
- I/O Management Component (IOMC)
- Protection and Security Component (PSC)
- Command-Interpreter Component (CIC) (GUI's shell in Windows OS and CLI's shell in Unix/Linux OSs).



## Process Management Component (PMC)

- A **process** is a program in execution phase.
- A process needs certain resources (i.e. CPU time, Memory Space, Files access, and I/O devices) to complete its task.
- The OS (PMC) provides the following operations to support processes:
  - Creating a process,
  - Deleting/killing a process,
  - Suspending a process,
  - Resuming a suspended process,
  - Cloning a process,
  - Supporting Inter-process communication,
  - Creating/killing, ...etc. of **a child process** (sub-process)



- In most systems, processes form a tree, with the root (parent) being the first process to be created.
- **An example of a process tree:**
  - A created two children processes, B and C
    - B created three children processes, D, E, and F
- We will get into the details of the process management in Ch. 3 soon.



## Main-Memory Management Component (MMMC)

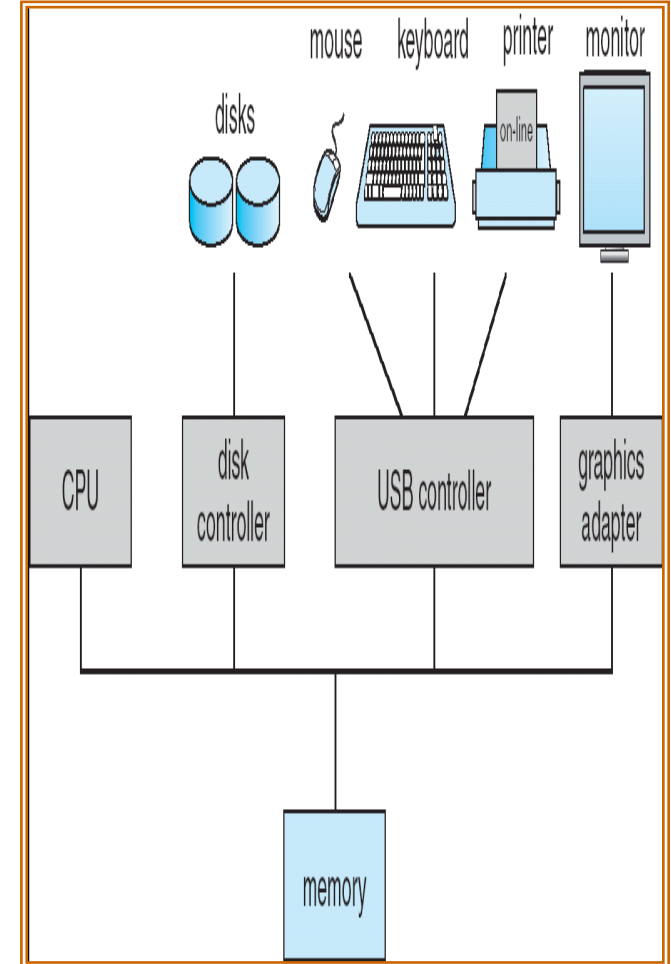
- The operating system (MMMC) is responsible for the following activities in connections with memory management:
  - The **MMMC** keeps track of which parts of the main memory are currently used and which parts are free.
  - The **MMMC** decides which processes to load when memory space becomes available.
    - A policy is needed
  - The **MMMC** decides how many block of memory to allocate to each process.
    - A policy is needed
  - The **MMMC** maintains the mappings of processes from physical to virtual memory and vise versa.
    - Through page tables
  - The **MMMC** decides when to remove/eject out a process from memory if memory space it required by a VIP process.
    - A policy is needed
- We will get into the details of the main memory management in Ch. 8 soon.

# File Management Component (**FMC**)

- File Management
  - A file in windows OS is a collection of related information defined by its creator. Files represent programs (both data, source, object and executable forms).
  - In Unix/Linux files are either ordinary files (doc., exe., pdf., media, etc.), directory files (folders) or device file (printers, CD, ..)
  - The **FMC** is responsible for the following activities in connections with the file management:
    - File creation and deletion.
    - Directory creation and deletion.
    - Support of primitives for manipulating files and directories.
    - Mapping files onto secondary storage.
    - File backup on stable (nonvolatile) storage media.
- We will get into the details of the file management in CHs. 10 &11 soon.

# I/O Management Component (**IOMC**)

- The **IOMC** is responsible for the following activities in connections with I/O management:
- Controls the processes access to I/O resources via drivers.
- Allows sharing and synchronizing of the I/O resource.
- Provides a standard interface between process (user's one or system's one) and I/O devices.
  - File system (disk), sockets (network).
- Provides buffers for caching services.
- Manages the device's driver code for specific I/O devices.



- We will present the details of I/O management in the intensive introduction & you need to read Ch.13.

## Mass-Storage Management Component (MSMC)

- Since main memory (primary storage) is **volatile** and **not enough** to accommodate all **data of concurrently running processes**.
- So, the computer system uses a **mass-storage** memory to extend (**create VM**) and to back up the main memory content.
- The OS uses (**VM**) as the principle on-line storage medium, for both processes and data.
- We will present the details of Virtual Memory Management in Ch.9.
- The MSMC is responsible for the following activities in connection with disk management:
  - Manages the free space available on the secondary-storage device.
  - Allocates of storage space when new files have to be written.
  - Schedules the read and write requests for mass-storage access.
- We will present the details of Mass-Storage Management in Ch.12.

## Command-Interpreter Component (CIC)

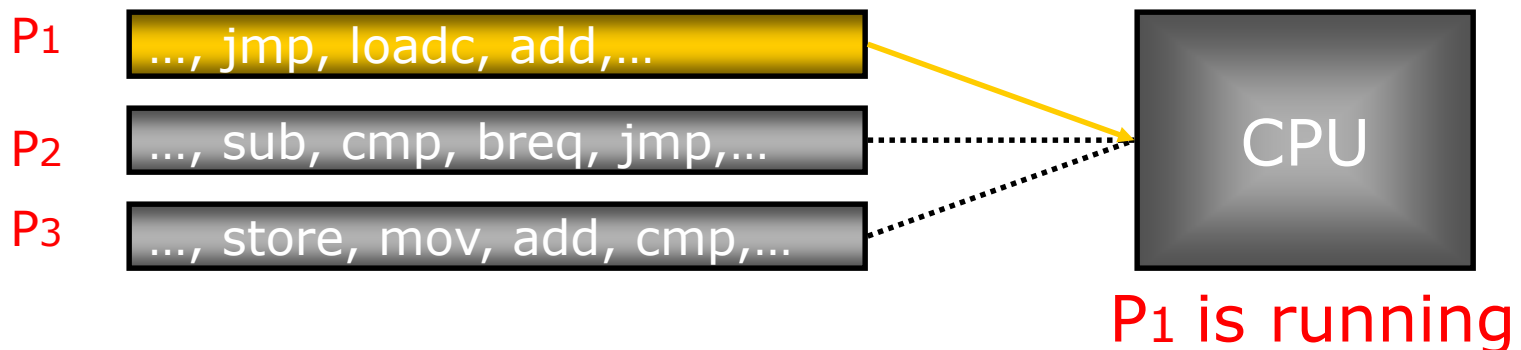
- Many commands are given to the OS either from keyboard (command-line interface, CLI), or script files (i.e. Unix/Linux), or from GUIs through the mouse (i.e. in Windows or Mac).
- These commands may deal with:
  - Process creation and management
  - I/O handling
  - Mass-storage management
  - Main-Memory management
  - Editing and File-system access
  - Networking (FTP, Telnet, etc.)
- The OS's module that reads, interprets and dispatches these commands is called variously:
  - Shell (any program that users use to interact with the OS)
  - Control-Card Interpreter
  - Command-Line Interpreter
- Its function is to get the command, interprets and dispatches it to the corresponding module in the OS to execute it.
- We will not present a specific one but you need to practice them on different OSs by yourself in the Lab part .

## Protection & Security Component (PSC)

- Since the OS supports multi-users with concurrent execution of multi-processes, then these processes or users must be protected from one another's.
  - Protection refers to mechanisms for controlling the access of processes, or users to the resources defined by a computer system (**internal**).
  - Security refers to defense of the system against **external** attacks.
- The **PSC** is responsible of:
  - Protecting processes' assigned resources (files, memory, etc.) from being accessed by other processes
  - Detecting any trial for intrusion
  - Graceful recovery from errors detected by the hardware
    - e.g. **illegal instructions, divide-by-zero**...
  - Preventing malicious destruction or resources.
  - Protects all system resources, i.e.
    - **CPU**, see in the next slide how does the OS protect the CPU.
    - **Memory**, see in the next slide how does the OS protect the memory.
    - .....
- We will present the details of Protection & Security aspects in **CHs.14&15**.

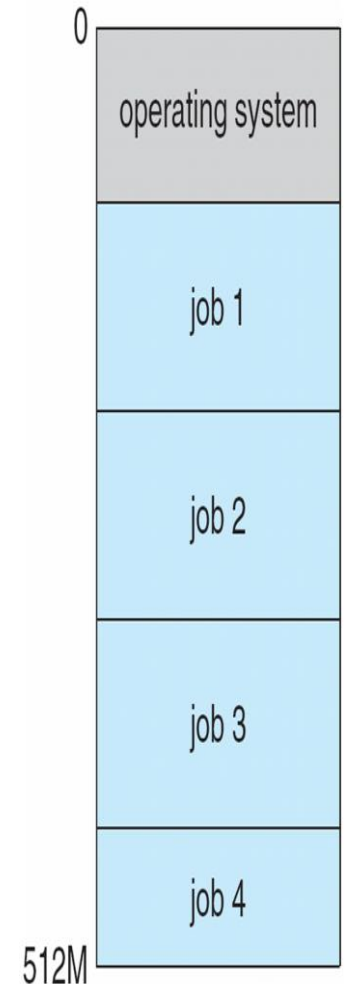
## Example: CPU Protection

- Since the CPU is one of the hot resources, the OS needs to prevent processes from hogging the CPU for a long time. **i.e.**
  - Infinite loops
  - Waiting for non-existent resources
- OS uses a **timer** to control process execution so that when the timer expires, the control switches back to the OS.
  - Context switch
- This is the idea of **CPU time-slicing**
  - Each process runs for a few msec.



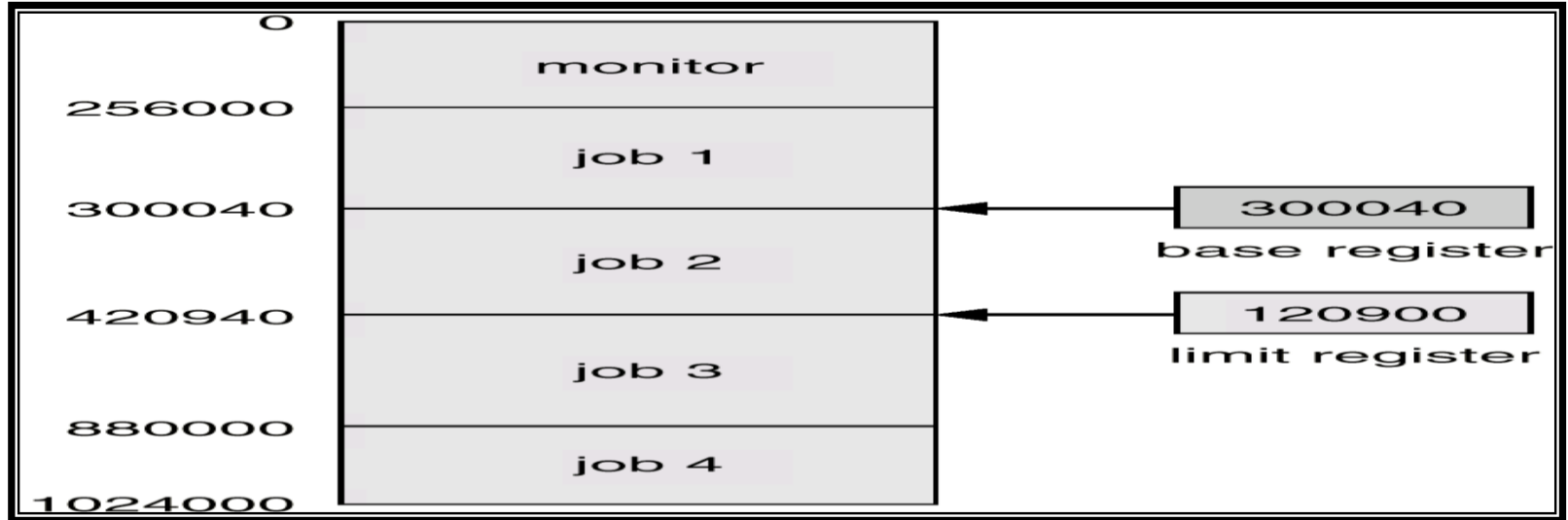
## Example: Memory Protection

- The OS needs to ensure that no process can interfere with any other processes, i.e.
  - Could overwrite other processes in memory
  - Needs to protect the OS memory in particular
- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- The OS uses two registers to determine the range of legal addresses of each process.
  - **Base register**: Holds the smallest legal physical memory address for the process.
  - **Limit register**: Contains the size of the process.

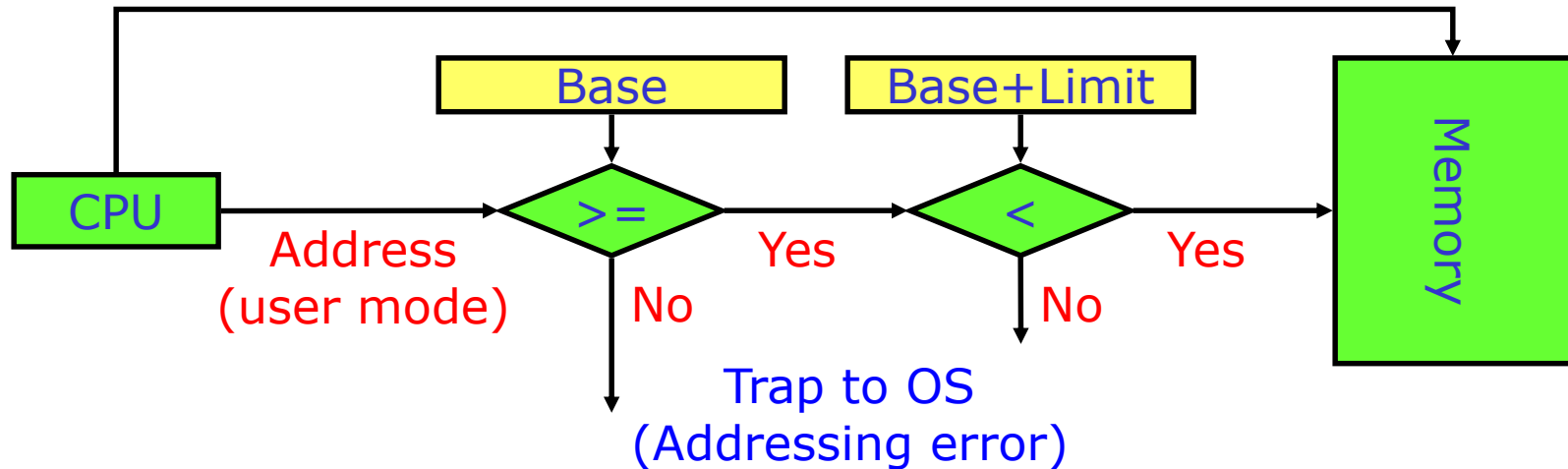




## Base & Limit Registers to protect a process

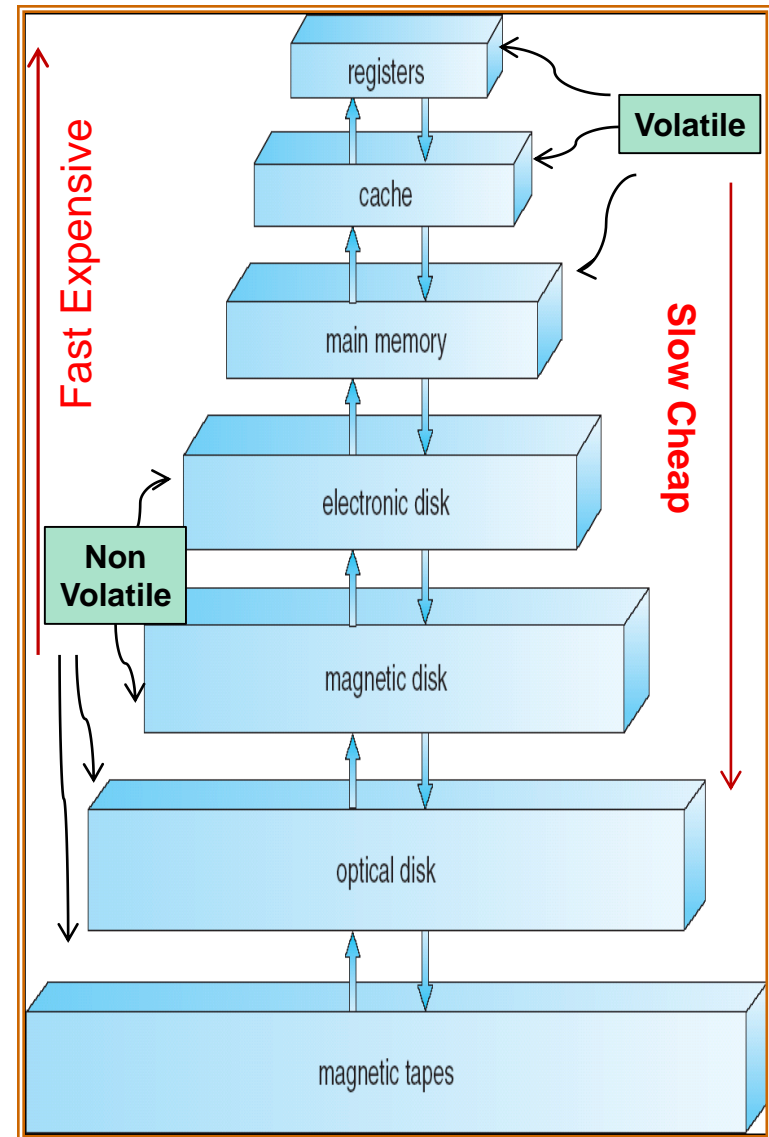


Address (supervisor mode)



# Storage Hierarchy in the Computer System

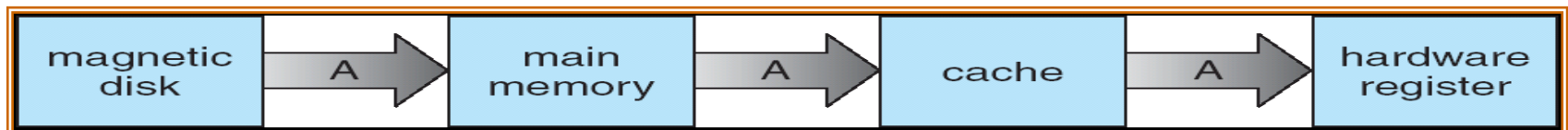
- In any computer system, the storage systems are organized in hierarchy **based on**:
  - Speed
  - Cost
  - Volatility
  - Size
- CPU registers** (index registers, ACC, etc.) provides a high speed cache for CPU.
- Cache memory**: Copying temporary information into faster storage devices with access time close to the CPU time.
- Main memory**: Only the storage media that the CPU can access directly.
- Main memory can be viewed as a last cache for secondary storage.
- Secondary storage**: Extension of main memory that provides large nonvolatile storage capacity.



# Caching and Consistency

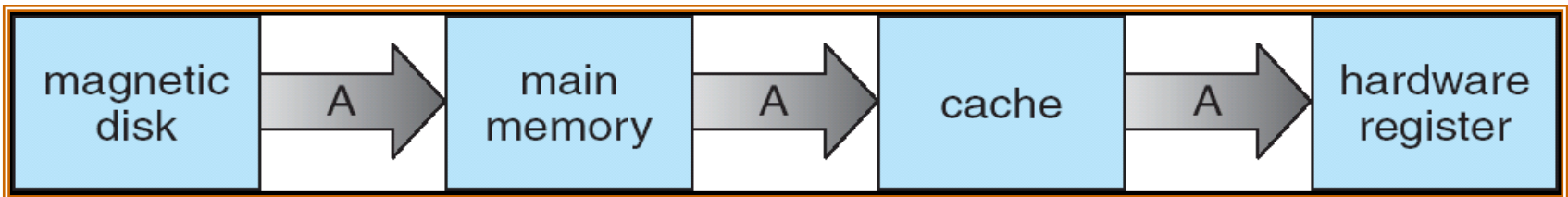
- Caching

- Use of high-speed memory to hold recently-accessed data.
- Requires a cache management policy (cache size, replacement policy).
- Data that resides on the disk are copied into the main memory, then into the cache and then into the CPU registers for modification such as a variable **incrementing A**.
- **Caching introduces another level in storage hierarchy**. This requires data that is simultaneously stored in more than one level to be **consistent**.
- **Consistency Problem**: due to replication of data, this may lead to one copy being different from the others. The OS must insure that all copies of the data will be the same.
- This is not a big problem, in computing environment where only one process executes at a time. The modification done on the CPU registers can be **copied back** to the disk for a consistency purpose.
- In multiprocessing environment, extreme care must be taken by the OS so that if several processes want to access A, then each should get the latest value of A.

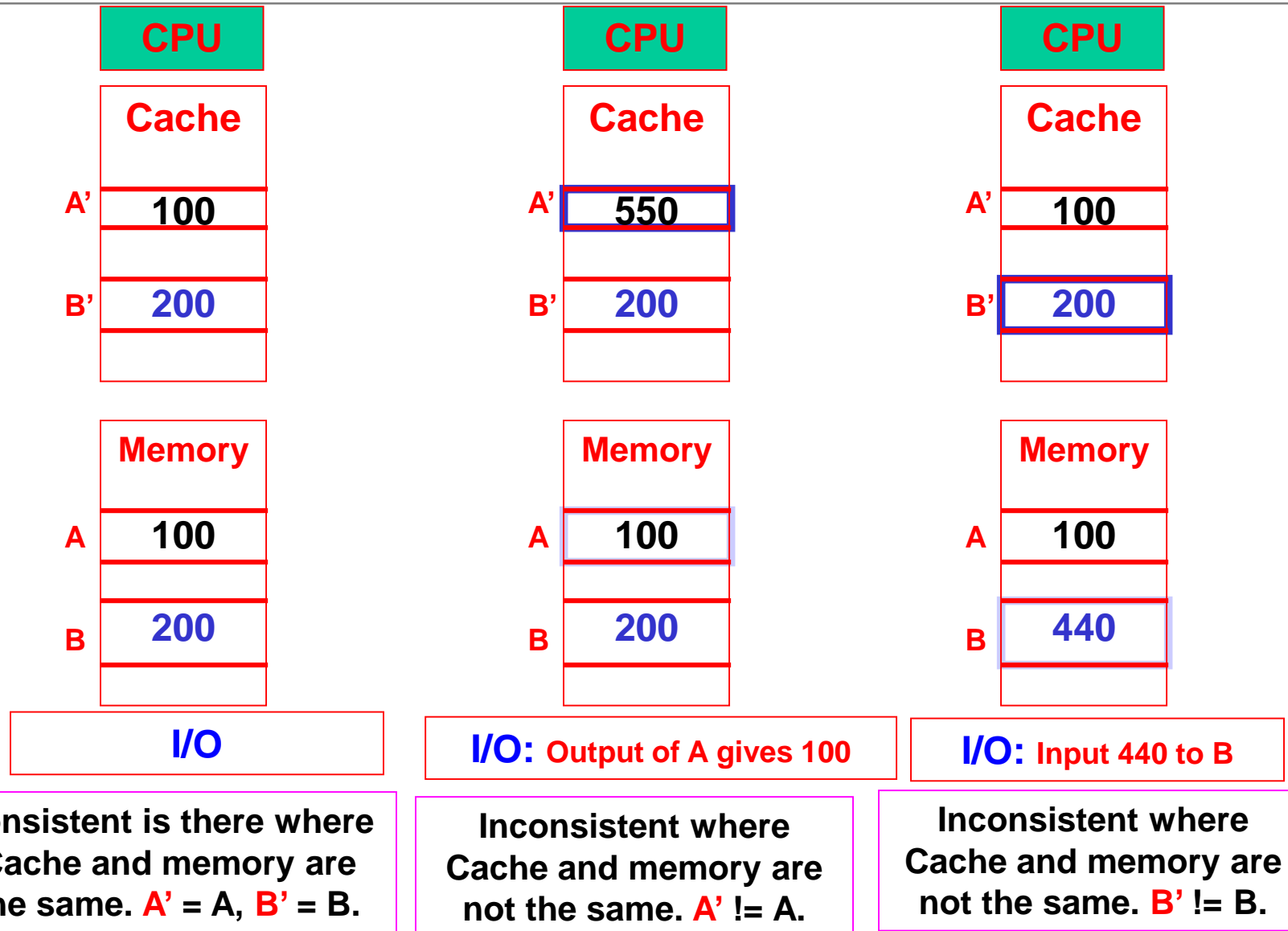


## Caching and Coherency

- **Coherency:** The OS needs to synchronize the data access in multiple caches such that reading a memory location via any cache will return the most recent data written to that location.
- To ensure that each cache has a copy of the newest version when it needs it.
- OS must ensure that an update of A's value will be immediately reflected in all other caches where A resides.
- Consistency leads to coherency but not the other way.

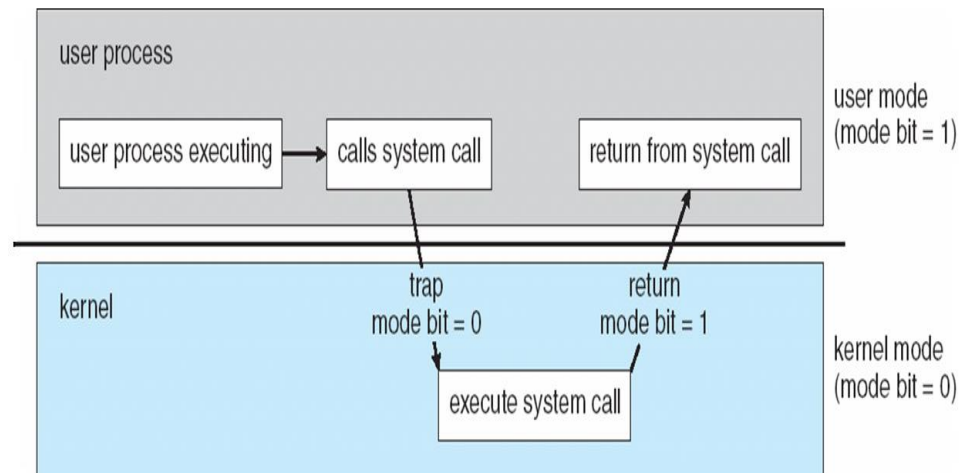
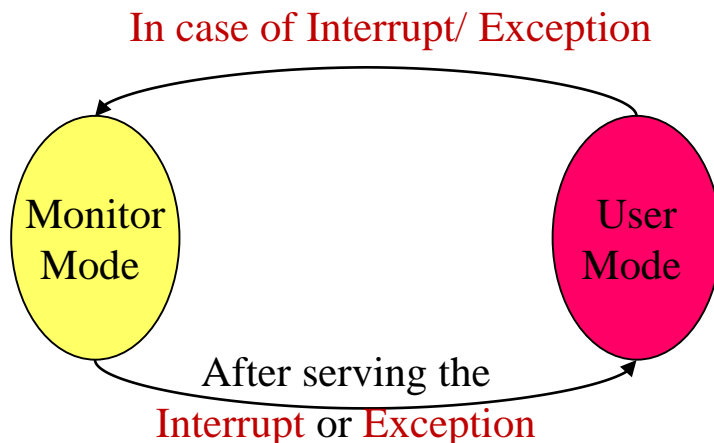


# The Problem of Cache Consistency



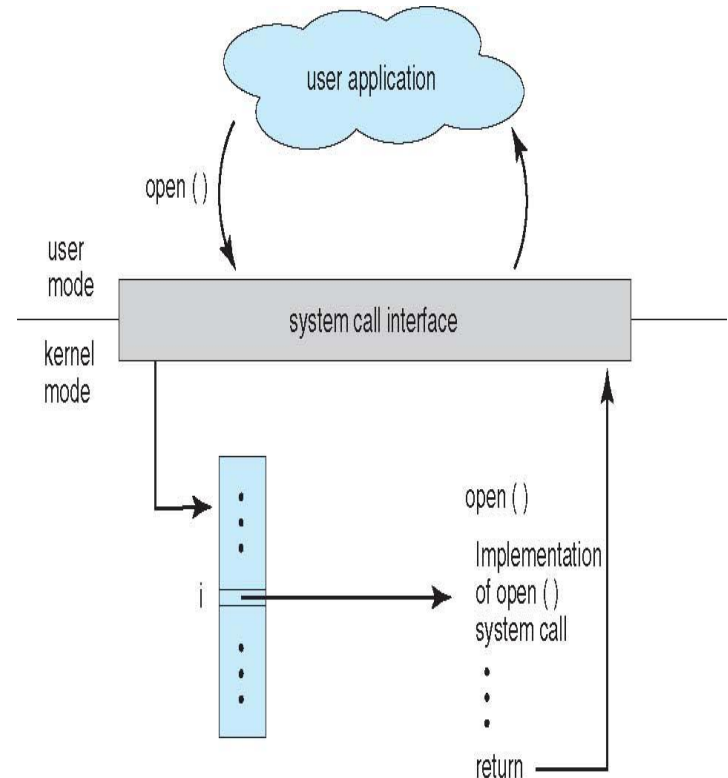
# Dual-Mode Operation

- Two modes of operation are there in any operating system.
  - User mode**: the control of execution will be done by a user's program.
  - Monitor mode** (supervisor, or kernel mode): the control of execution will be done by the OS.
- A **Mode bit** (a hardware bit) is used to indicate the current mode: monitor's mode where the mode bit is reset=0 and **user's mode where mode bit=1**.
- When an interrupt or a trap occurs, the control switches to monitor mode.
- All I/O instructions (**privileged instructions**) can be executed only in monitor mode.
- The OS must ensure that a user program could never gain control of the system in monitor mode.



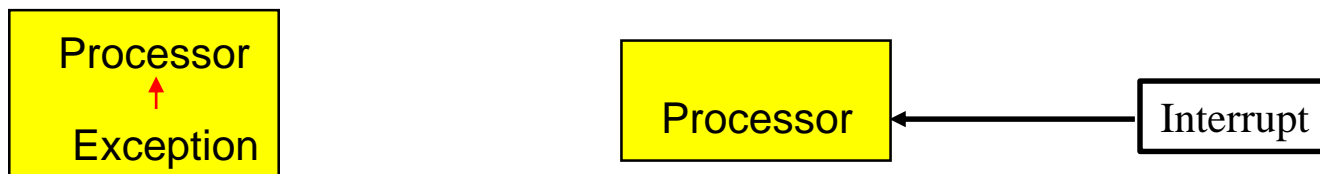
# Privileged Instructions

- Since user's program may issue **illegal** I/O instructions: i.e.
  - Write to a non-existence device
  - Read more data than a disk holds
- Some instructions are restricted to be executed under the OS control, **in monitor or supervisor mode**.
  - They are known as **privileged instructions**
- User programs can only perform I/O by **requesting it through the OS**.
- OS retains control over user's programs to:
  - Directly access I/O devices (**disks, network cards, etc.**)
  - Manipulate memory state management
    - Page table pointers
  - Manipulate special '**mode bits**'
    - Interrupt priority level
  - Halt instruction



# Interrupts and Exceptions

- Modern OSs are interrupts or exceptions driven programs.
- Two main types of events: **Interrupts** and **Traps/Exceptions**:
  - **Interrupts** are used to handle events **external** to the processor, caused by hardware devices and **are not visible to the user's programs**, e.g.:
    - A device finishes I/O operation (keyboards, mouse, etc. )
    - Timer fires
  - **Exceptions or traps** are used to handle events **internal** to the processor (detected by the processor while executing the process), means caused by software and **it is visible to the user's programs**, e.g.:
    - An exception e.g., “**div. by zero**”
    - A page fault, “**write to a read-only page**”
    - Asking to make arithmetic operation on non-numbers.
- The Interrupts transfer the control to the interrupt service routine, through the interrupt vectors which contain the addresses of all the service routines.





## Reasons of Interrupts/**Exception**

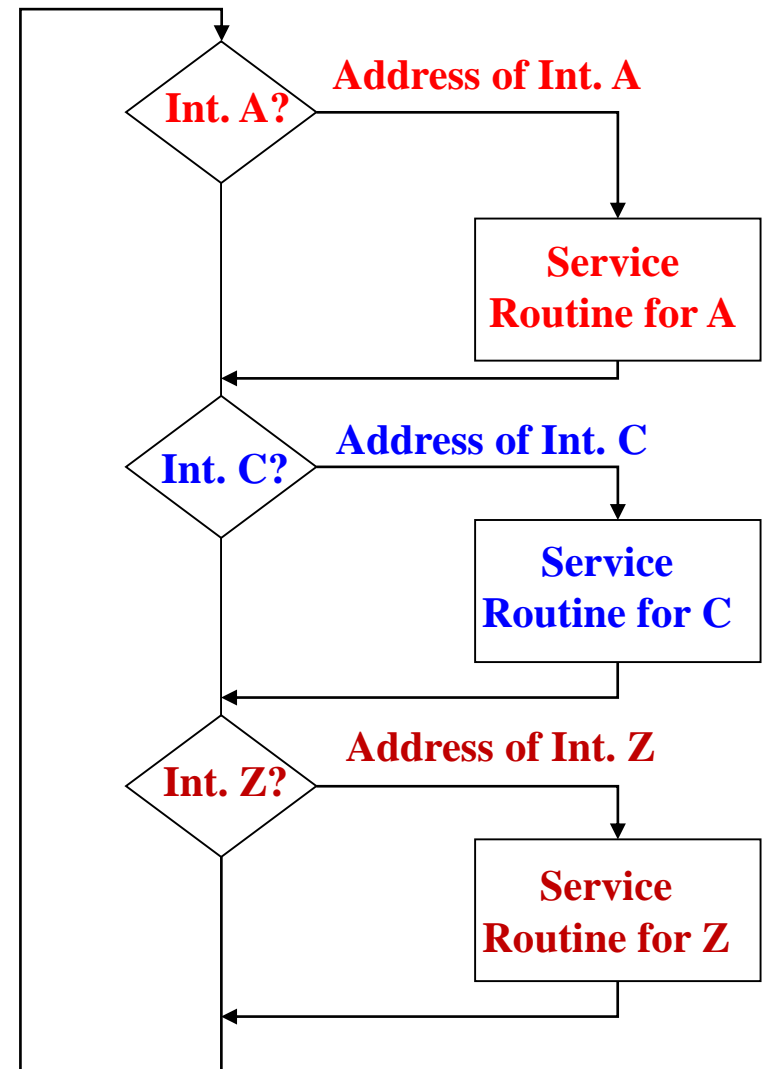
- Reasons for interrupts (or exceptions/traps) are:
  - Control of asynchronous I/O devices
  - Context switch between processes based on a CPU scheduling policy
  - Exceptional conditions (e.g., **illegal instruction**) occurred during execution.
  - System call: a user's process requests for OS services.
  - etc..
- **The OS must save the address of the interrupted instruction.**
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost of interrupt.

# Interrupt Handling

- Serving an interrupt is known as “**Interrupt Handling**”.
- Different types of interrupts are handled by different interrupt service routines.
- The OS maintains a table, known as **Interrupt Vector**, that contains the starting addresses of the service routines.
- **An integer is associated with each type of interrupt**. When an interrupt occurs, the corresponding integer is supplied to the OS usually by the hardware (**in a register**).
- The OS determines which service routine to be executed by mapping the **Interrupt Vector**.
- The OS preserves the state of the executing process by storing
  - CPU Registers (PC, Process Status Word, SP, Data Registers)
  - Additional information about the current process and its state
- Execute the interrupt service routine.
- Upload the status of the process and resume its execution.

# On Interrupts

- The hardware device calls the OS at a pre-specified location/register,
- The OS saves the state of the current process, (contents of, registers: PC, SP, general-purpose registers)
- The OS identifies the device and the cause of interrupt,
- Responds to the interrupt by executing the service routine,
- Execute a return from interrupt (RTI) to return to the interrupted process,
- OS restores the state of the interrupted process,
- Key Fact: None of these actions are visible to the user program.



## On Exceptions/Traps

- The running processes calls the OS at a pre-specified location,
- The OS identifies the cause of the exception (e.g. divide by 0),
- If the user's process has an exception handling routine, then the OS adjusts the user's process state so that it calls its handler routine,
  - If the user's process does not have a specified handler, then the OS suspends it and runs other available service routines.
- Execute a RTI instruction to return to the user's process.
- Key Fact: Effects of exceptions are visible to user's processes and causes abnormal execution flow.

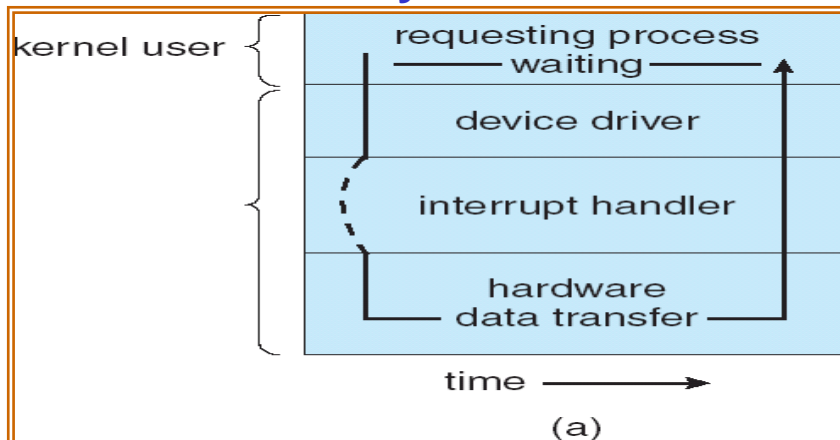
# Outline of OS Intensive Introduction Classes

- We have presented till now:
  - Definitions of the OSs, Benefits of the OS to users and application programs,
  - Computer-System Organization and Operation, Different types of OSs, the Major OS Issues, Operating System Services, Different views of the OS,
  - Main Goals of OS, Supporting of Multi-processing by OS: why?
  - Requirements of Multi-processing (HW and SW support), Multiprocessor Systems (Tightly and loosely coupled, symmetric and asymmetric mode of coupling),
  - Distributed vs. Network OSs, Clustered systems, and why do we support clustering?
  - Computing models: Client-server, P2P, Grid-computing, Cloud-Computing,
  - Fundamental components of the OS; i.e. Process Management, Memory Management, File Management, I/O Management, Mass-Storage Management, Command-Interpreter,
  - Protection & Security components of the OS, (CPU and Memory Protection as hot resources).
  - Storage Hierarchy, Consistency and Coherency Definitions and Support by the OS.
  - Dual-Mode Operation of the OS, Privileged Instructions,
  - Interrupts and Exceptions Definitions.
  - Reasons of Interrupts/Exception, Interrupts and Exceptions Handling.
- We are going to present Today:
  - I/O Handling Methods (Synchronous/blocking and Asynchronous/non-blocking)
  - I/O Data Transfer Techniques (Polling, Interrupt, Direct Memory Access)
  - System Calls, System Programs, and how to handle a system call?,
  - Passing System Call Parameters to the OS, and Processes Communication Methods,
  - Operating system Design Issues, Operating system Design goals,
  - Different ways of structuring the operating systems, Operating System Implementation, System Generation (SYSGEN). Process Management (Ch. 3)

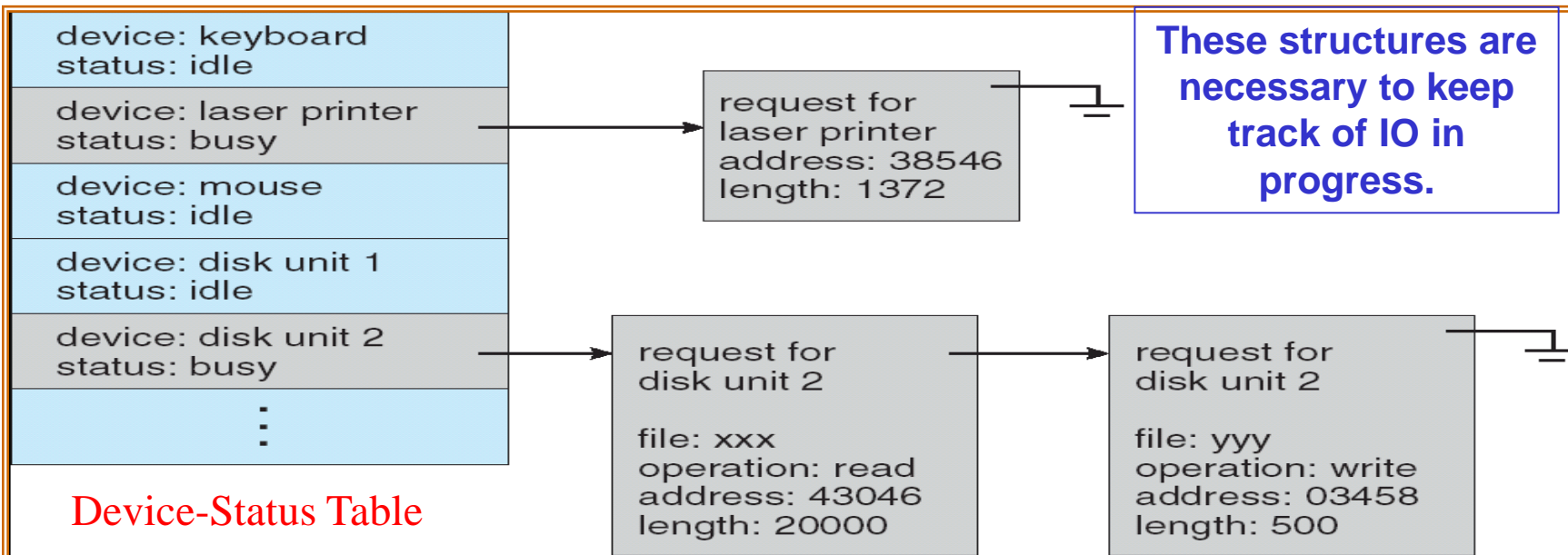
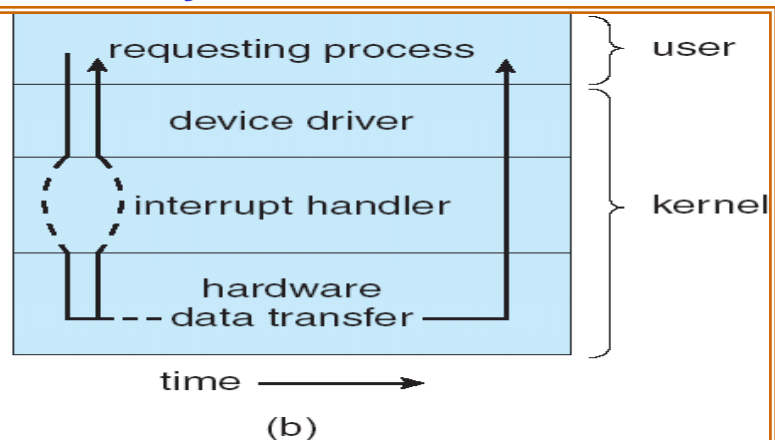
- **Synchronous/blocking**: blocking the progress of a process while the I/O operation is in progress, leaving system resources idle. This means that the CPU can spend almost all of its time idle waiting for I/O operations to complete. **How this is done?**
  - Request is made to the I/O device,
  - The CPU waits for the I/O device,
    - Wait instruction idles (**Looping**) the CPU until the request is completed.
  - At most, one I/O request is running at a time, no simultaneous I/O processing,
- **Asynchronous/non-blocking**: after requesting for an I/O operation, the control returns to user's process without waiting for the I/O request to be completed,
  - Request is made to the I/O device,
  - CPU records the request,
    - **Device-status table**: contains entry for each I/O device indicating its **type**, **address**, and **state**,
  - The CPU continues the process execution, **it can be a different one**
  - After completing of the I/O operation, the device interrupts the OS,
  - The processor records that the request is done,
  - It can resume the process execution.

# I/O Handling Methods: Ch 13

## Synchronous



## Asynchronous



- There are several ways of managing data transfer between I/O devices and the Main Memory.
- Programmed I/O (Polling)
  - Processor does all the work.
  - CPU checks by polling, reads and writes into device buffers.
- Interrupt Driven I/O
  - CPU asks devices to let it know when they are ready to transfer data.
  - Device notifies CPU when I/O operation complete.
- Direct Memory Access (DMA, a memory controller)
  - DMA is programmed by the OS to exchange data at high rate between main memory and the I/O device.
  - CPU asks DMA to perform the I/O directly to or from memory.
  - DMA controller performs “I/O”, not CPU.
  - CPU notified with DMA complete.
- The OS is responsible for choosing & managing the right technique for each specific I/O in order to deliver the best overall performance.

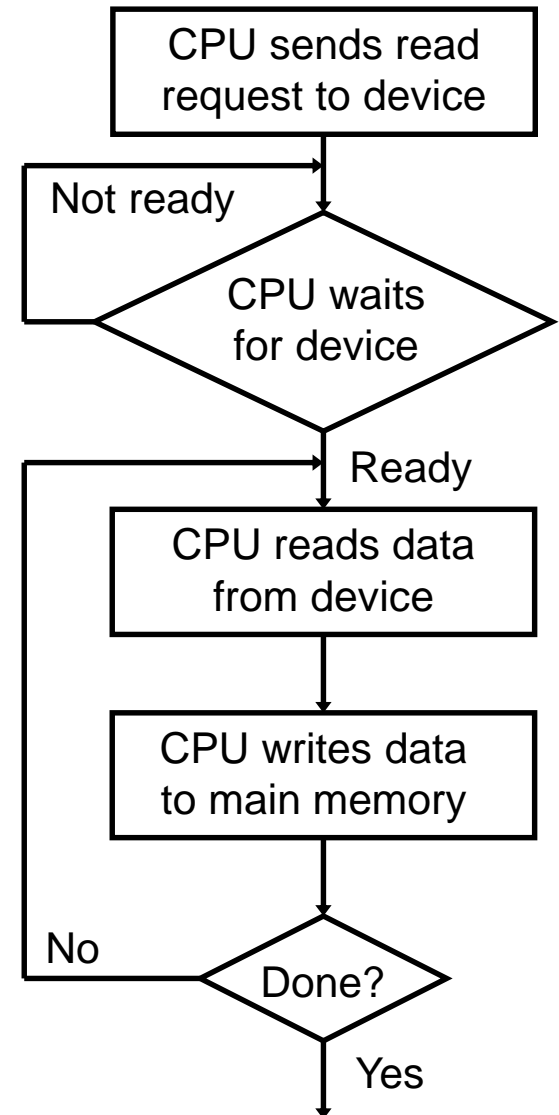


## Programmed I/O: Polling

### Programmed I/O (Polling): CPU Polls I/O Device's Status

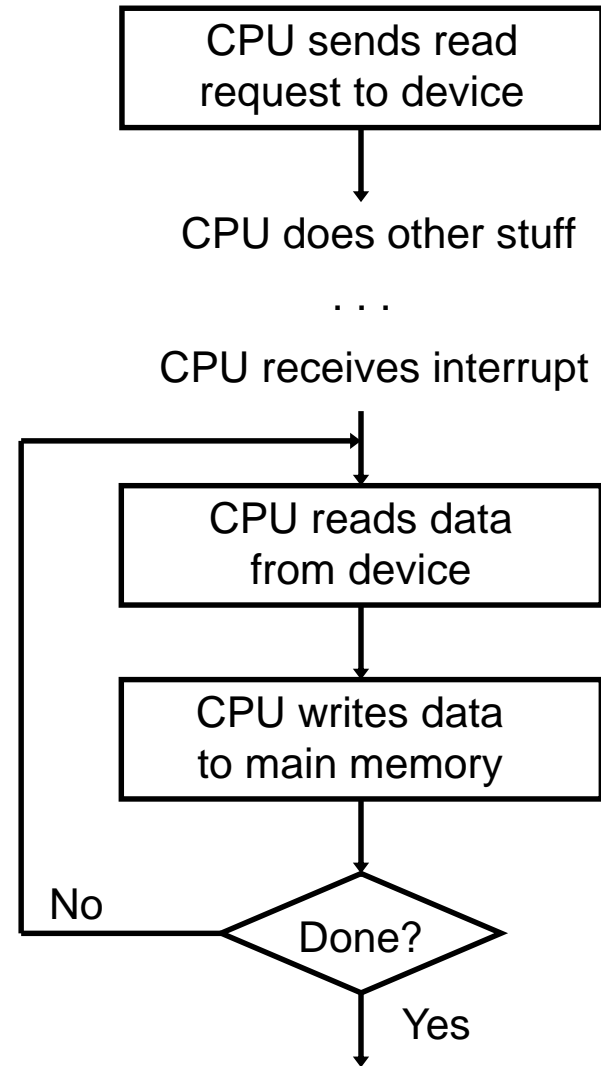
#### Registers

- CPU regularly checks or **polls in turn** each I/O channel or port to determine if it has information for input or is ready to accept data for output.
- I/O modules share the bus with the CPU.
- CPU has direct control over the I/O
  - Sensing the status through flag register
  - Read/write commands
  - Transferring data
- **Polling is time consuming:**
- CPU must swap between executing processes and polls of each port.
- **Wastes CPU time:** Programmed I/O asks for too much attention of the CPU if the device is fast.
- If the device is slow the CPU might have to wait a long time (**most devices are slow compared to modern CPUs**).
- The CPU is also involved as a middleman for the actual data transfer.



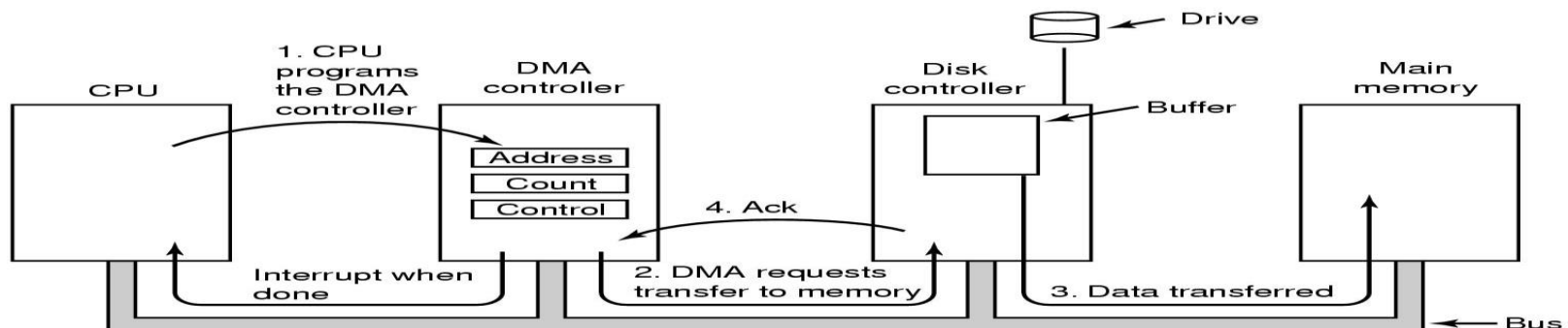
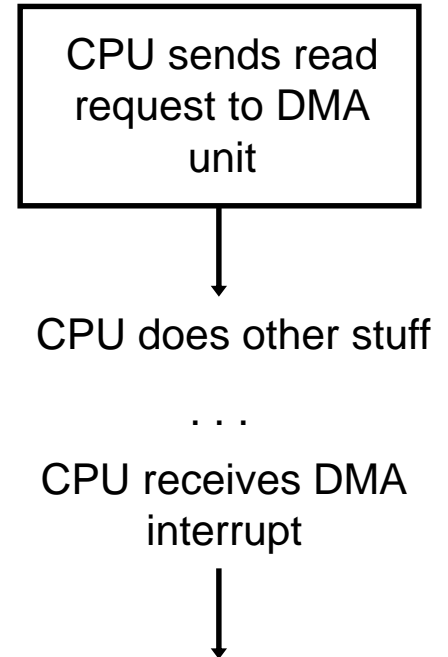
## Interrupt-Driven I/O

- The CPU sets up I/O operation and continues its work,
- The device performs the I/O (long time),
- If the device completes, it interrupts the CPU,
- Then the CPU responds to the interrupt and transfers the data,
- Continue once I/O is complete,
- Interrupts save overhead of polling the I/O resources by the CPU.

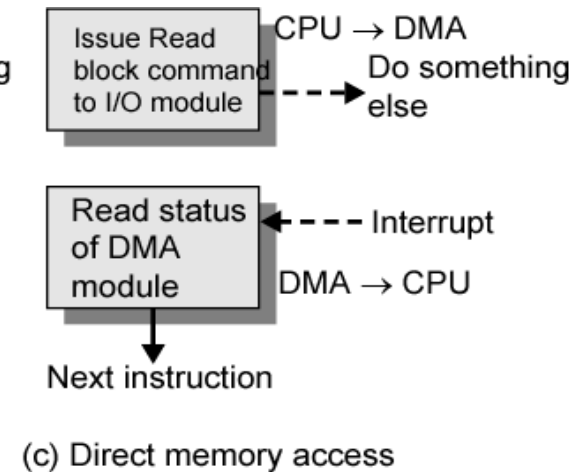
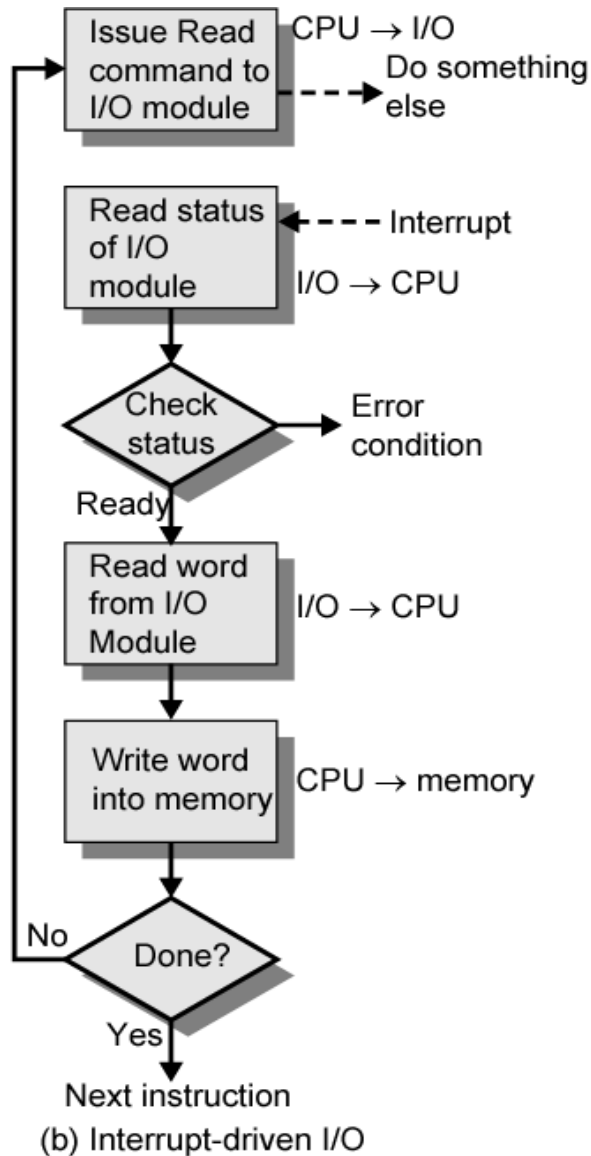
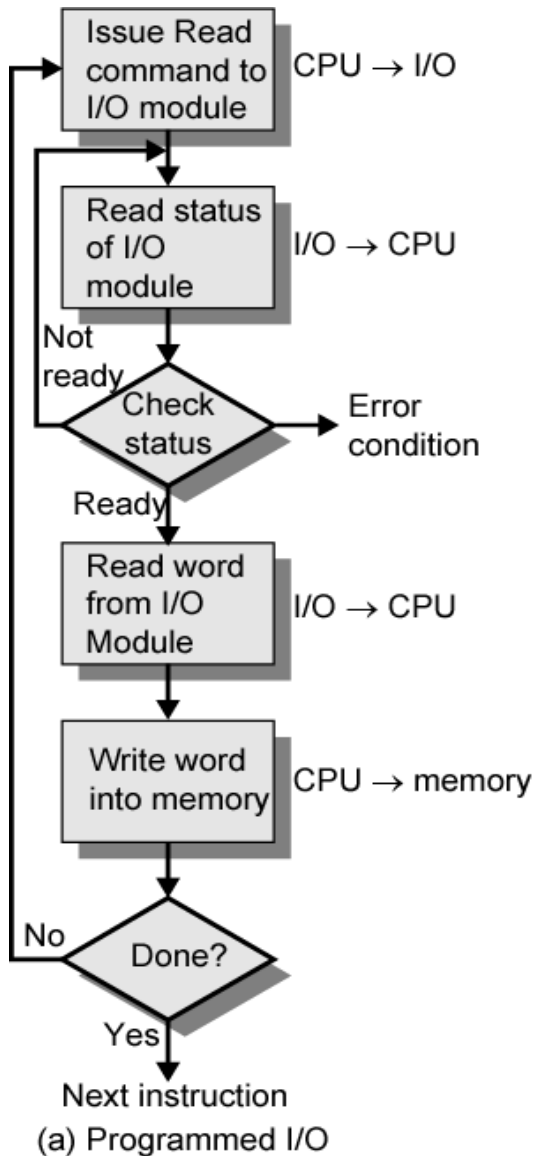


# Direct Memory Access

- Fast I/O devices (graphics cards, network cards and sound cards, etc.) use Direct Memory Access (DMA), why?
- DMA permits the I/O device to transfer data directly to or from main memory without having each byte handled by the CPU.
  - DMA controller moves data between device and memory then sends interrupt to CPU only when transfer is complete.
    1. CPU only initiates operation
    2. DMA controller transfers data directly to/from main memory
    3. Interrupt when transfer completed
- DMA enables more efficient use of interrupts, increases data throughput, and potentially reduces hardware costs by eliminating the need for specific FIFO buffers for every resource.

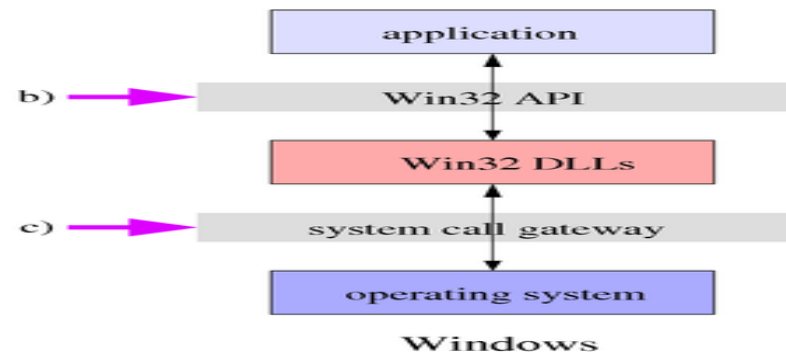
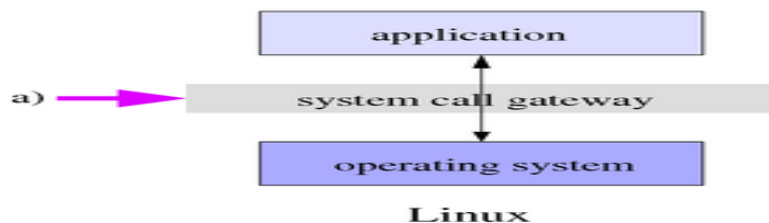


# Three Techniques for I/O



# System Calls

- System calls allow user's processes to request some services from the operating system **which the process itself is not allowed to do**.
- **A System call:** Provides a "direct access" to OS services (e.g., file system, I/O routines, memory allocate & free routines) by user's processes.
- System calls in fact, **are treated as a special case of interrupts**.
- System calls execute instructions that control the resources of the computer system, e.g., I/O instructions for devices.
- Programs that make system calls called "**system programs**".
- **System programs** were traditionally coded in assembly language but currently written by C, C++, and Java Programming languages.
- Mostly accessed by programs via a high-level **Application Program 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).



# System Programs

- The **system programs** are a set of utility programs help the process to issue a system call requesting services, such as:
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
  - **Status information**
    - Asks the system for info. i.e. **date, time, amount of available memory, disk space, number of users**
  - Others provide detailed performance, logging, and debugging information
  - **File modification**
    - Text editors to create and modify files
    - Special commands to search contents of files
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters.
- **Program loading and execution**- Absolute 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 (**http, ftp, telnet, ...**)
  - 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

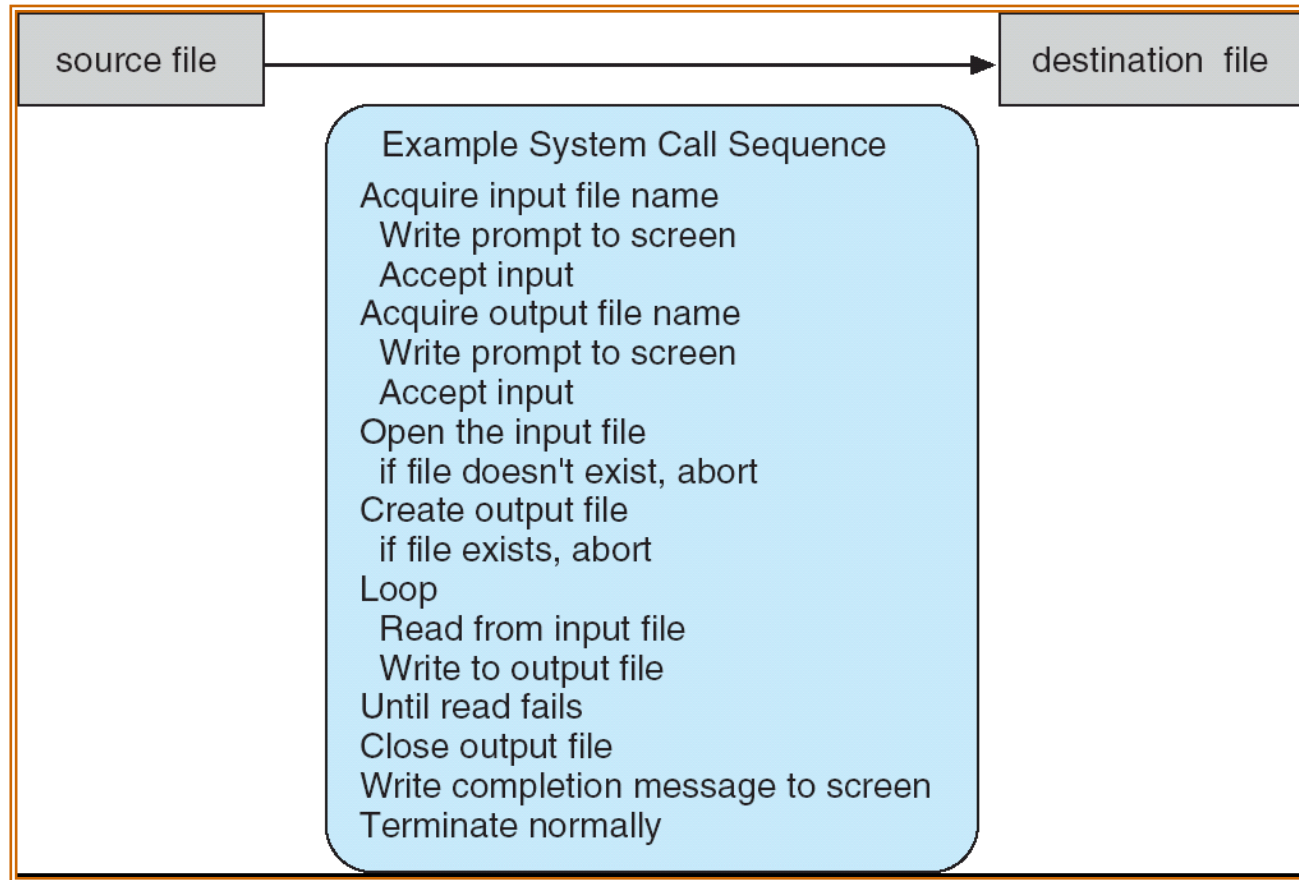
## Types of System Calls: Windows & Unix

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



## Example of System Calls

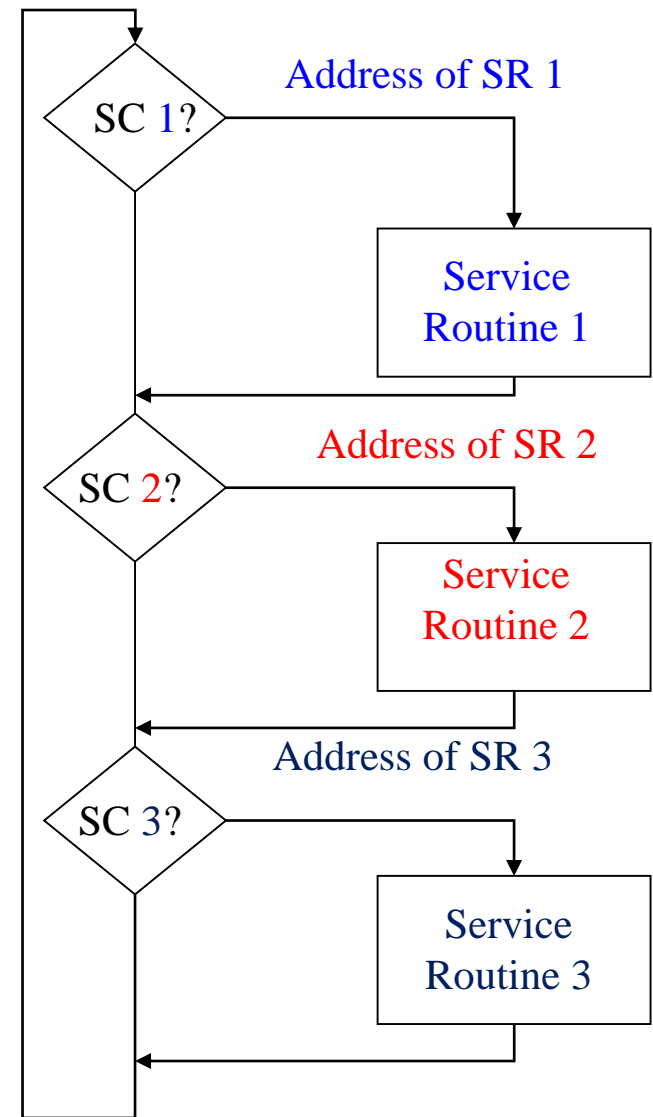
- System calls sequence to copy the contents of one file to another file.





## Handling a System Call

- User's process makes a system call **by executing a system call privileged instruction**.
- Hardware resets the mode-bit to **0** and switches to **kernel mode**.
- OS saves the state of the user's process.
- OS identifies the system call (**branch to case statement in system code**).
- Switch to a **service routine** based **on the associated system call number**.
- OS executes the **service routine**.
- OS restores the state of the user's process by loading its parameters from [**PCB**].
- Switches to the **user mode** by setting the mode bit.
- Resume executing the user's process.

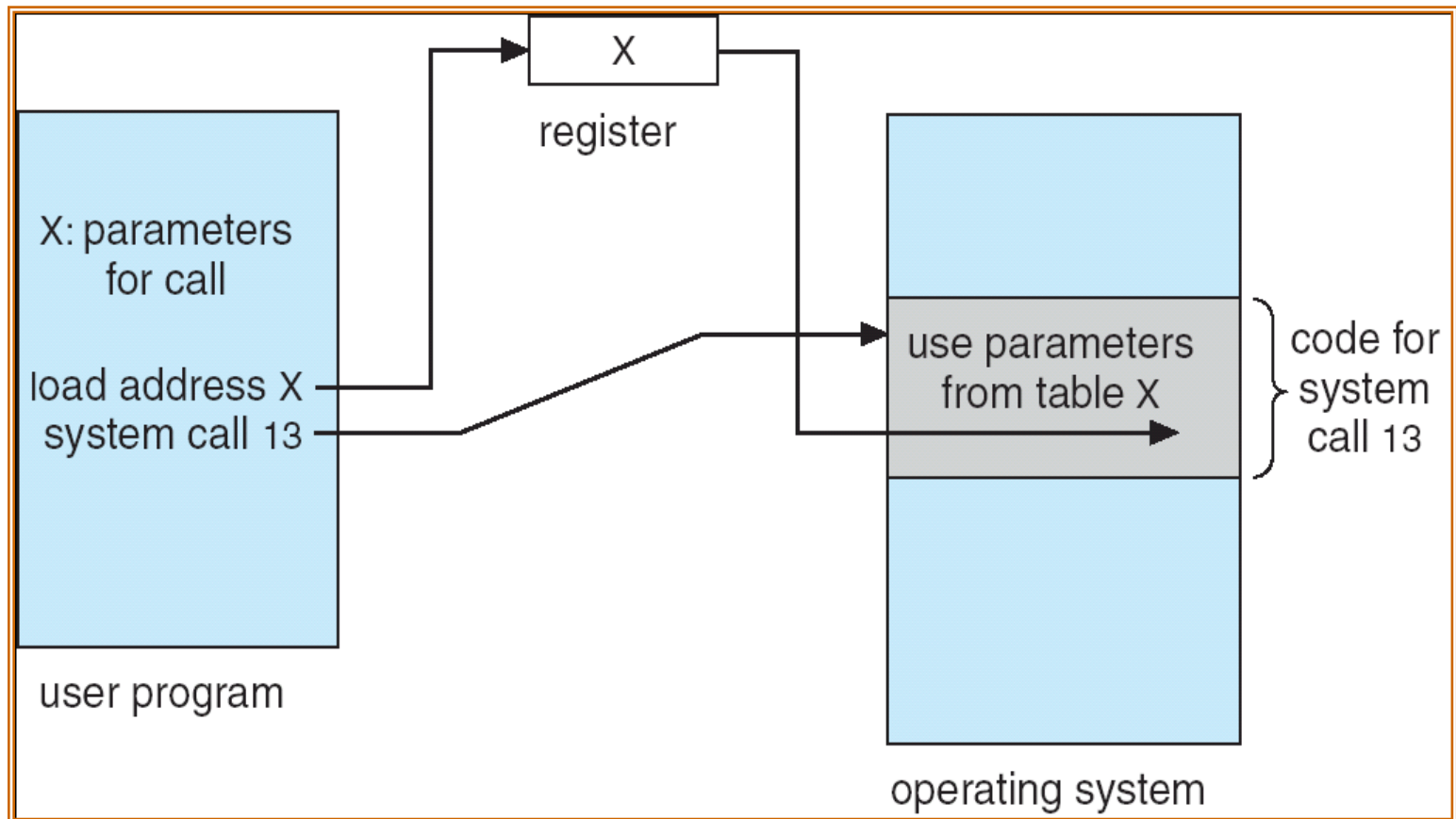


## Passing System Call Parameters to the OS

- Three methods are used to pass system call parameters to the OS:-
  1. System call parameters stored in registers
    - In some cases, may be more parameters than registers
  2. System call parameters stored in a block, or table, in memory, and address of the block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  3. System call parameters placed, or pushed, onto the stack by the process and popped off by the OS.

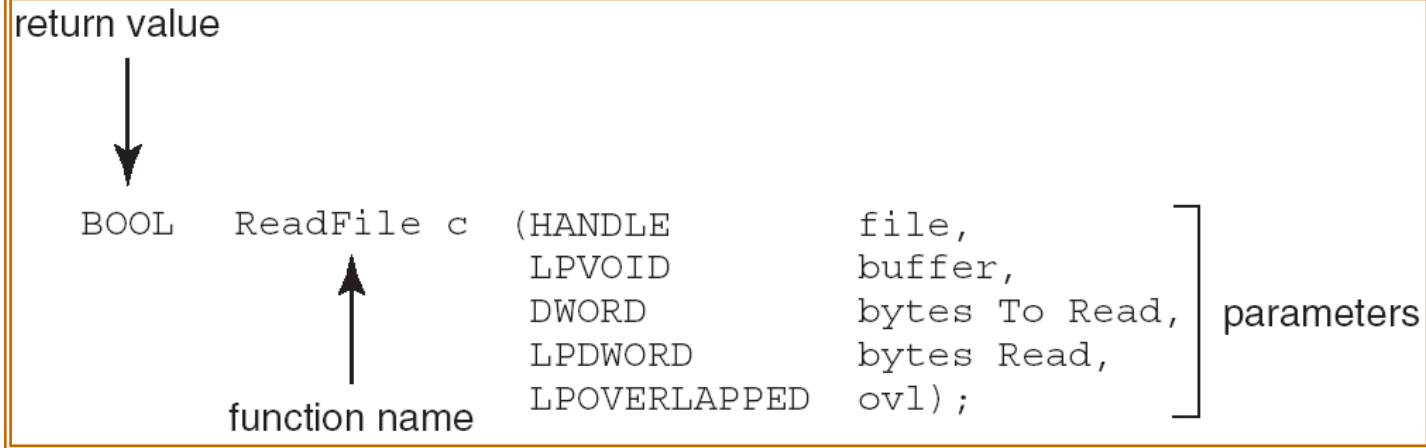
## Parameter Passing via Table

The methods to pass parameters between a running program and the OS.



## Example of Standard API

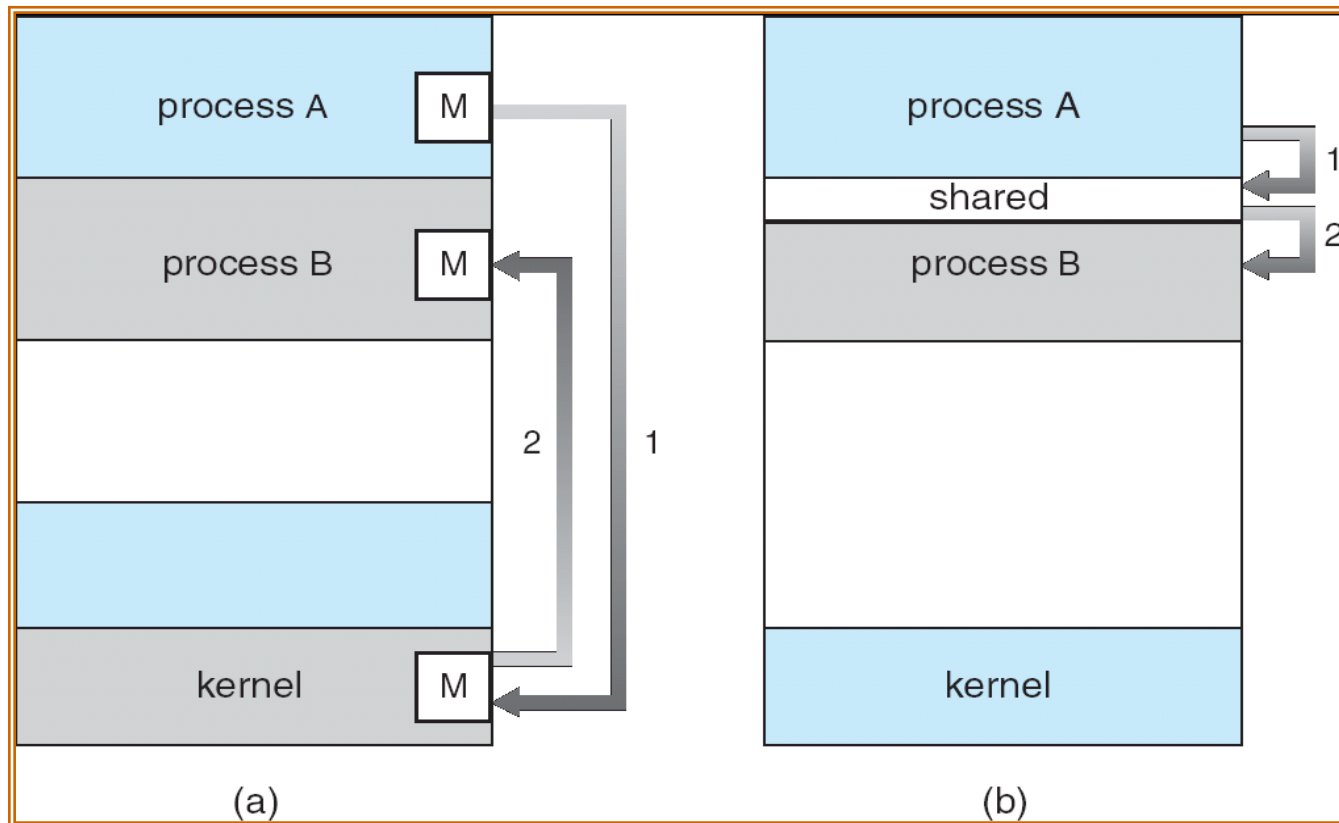
- Consider the **ReadFile()** system call in the Win32 API for reading from a file



- A description of the parameters passed to `ReadFile()`
  - **HANDLE** file—the file to be read
  - **LPVOID** buffer—a buffer where the data will be read into and written from
  - **DWORD** bytesToRead—the number of bytes to be read into the buffer
  - **LPDWORD** bytesRead—the number of bytes read during the last read
  - **LPOVERLAPPED** ovl—indicates if overlapped I/O is being used
- What is the reasonable way of passing these parameters to the OS?

## Processes Communication Methods

- Processes communicate with each other using direct message passing by **sending/receiving** messages through the OS kernel, or through shared memory .



Message Passing

Shared Memory

# Outline of OS Intensive Introduction Classes

- We have presented till now:
  - Definitions of the OSs, Benefits of the OS to users and application programs,
  - Computer-System Organization and Operation, Different types of OSs, the Major OS Issues, Operating System Services, Different views of the OS,
  - Main Goals of OS, Supporting of Multi-processing by OS: why?
  - Requirements of Multi-processing (HW and SW support), Multiprocessor Systems (Tightly and loosely coupled, symmetric and asymmetric mode of coupling),
  - Distributed vs. Network OSs, Clustered systems, and why do we support clustering?
  - Computing models: Client-server, P2P, Grid-computing, Cloud-Computing,
  - Fundamental components of the OS; i.e. Process Management, Memory Management, File Management, I/O Management, Mass-Storage Management, Command-Interpreter,
  - Protection & Security components of the OS, (CPU and Memory Protection as hot resources).
  - Storage Hierarchy, Consistency and Coherency Definitions and Support by the OS.
  - Dual-Mode Operation of the OS, Privileged Instructions,
  - Interrupts and Exceptions Definitions.
  - Reasons of Interrupts/Exception, Interrupts and Exceptions Handling.
  - I/O Handling Methods (Synchronous/blocking and Asynchronous/non-blocking)
  - I/O Data Transfer Techniques (Polling, Interrupt, Direct Memory Access)
  - System Calls, System Programs, and how to handle a system call?,
  - Passing System Call Parameters to the OS, and Processes Communication Methods,
- We are going to present Today:
  - Operating system Design Issues, Operating system Design goals,
  - Different ways of structuring the operating systems, Operating System Implementation, System Generation (**SYSGEN**).

## Operating System Design Issues

- As we have seen, the operating system consists of many components, so the designers need to think about:
  - How do we organize all of them?
  - Where do they exist?
  - How do they cooperate together?
- Answering these questions is a massive software engineering and design issue.
- What are the important software characteristics of an OS the designer should care?
  - Correctness, simplicity, and completeness,
  - Efficient performance,
  - Scalability and portability,
  - Suitability for distributed and parallel systems,
  - Compatibility with existing systems,
  - Security and fault tolerance,
  - .. etc.

## More Design Issues

- **Flexibility**

- It should be easy to add and remove functional modules from the operating system. This is the idea of *microkernel* operating system. In a microkernel operating system, it provides minimal services:
  - Inter-Process communications (IPC)
  - Some memory management
  - A limited amount of low-level process management and scheduling.
  - Low-level input/output
- These basic services are provided in kernel just because it is too expensive to provide them anywhere else.

- **Reliability**

- With distributed system, it is possible to provide higher reliability because some nodes can perform functions for the failed nodes.

- **Performance**

- With distributed systems, one expects better performance.

- **Scalability**

- The system design should be able to expand to large number of processors, not just a few.

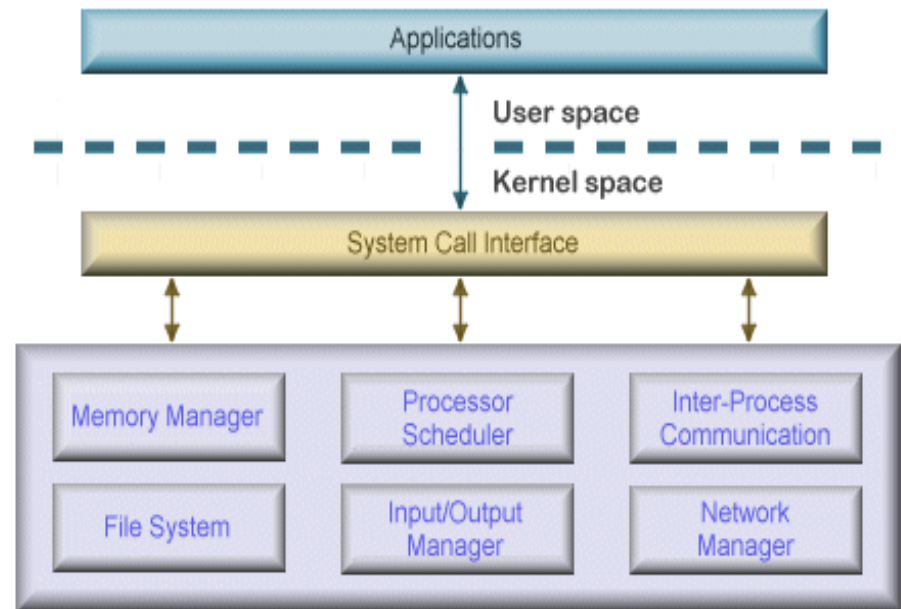
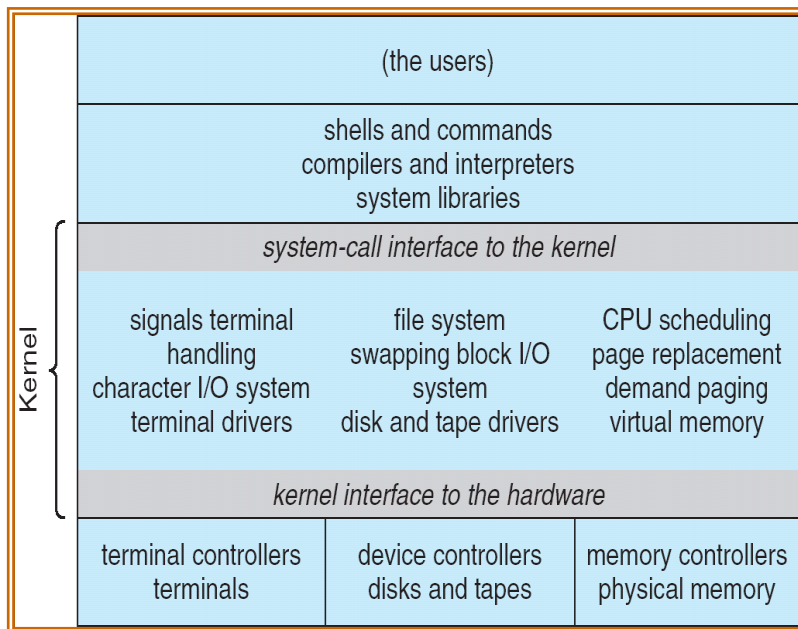


# OS Design Goals

- ❑ The OS is a kind of SW package and has to pass the Software Development Life Cycle, i.e. Requirements, Design, Implementation, Integration, Deployment and Maintenance phases.
- The OS design will be affected by the choice of the HW and the type of the system (Batch OS, Time shared OS, Single user OS, Multi-user OS, Real time OS, Distributed OS, or what?).
- The first problem in designing the system is to define the goals and specification of the system.
- **User's goals:** OS should be convenient to use, easy to learn, reliable, safe, multi-purpose, and fast.
- **System's goals:** OS should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.
- Next, we will present different ways of structuring the operating systems.

# Monolithic-Based of OS Structure: i.e. **Unix**

- The monolithic approach defines a high-level virtual interface over the hardware, with a set of primitives or system calls to implement OS services such as process management, concurrency, and memory management in several modules that run in supervisor mode.
- Kernel:** Everything below the system-call interface and above the physical hardware.
  - Provides file system, CPU scheduling, memory management, and other OS functions through system calls.
- Traditional UNIX OS versions were built as a monolithic kernel.

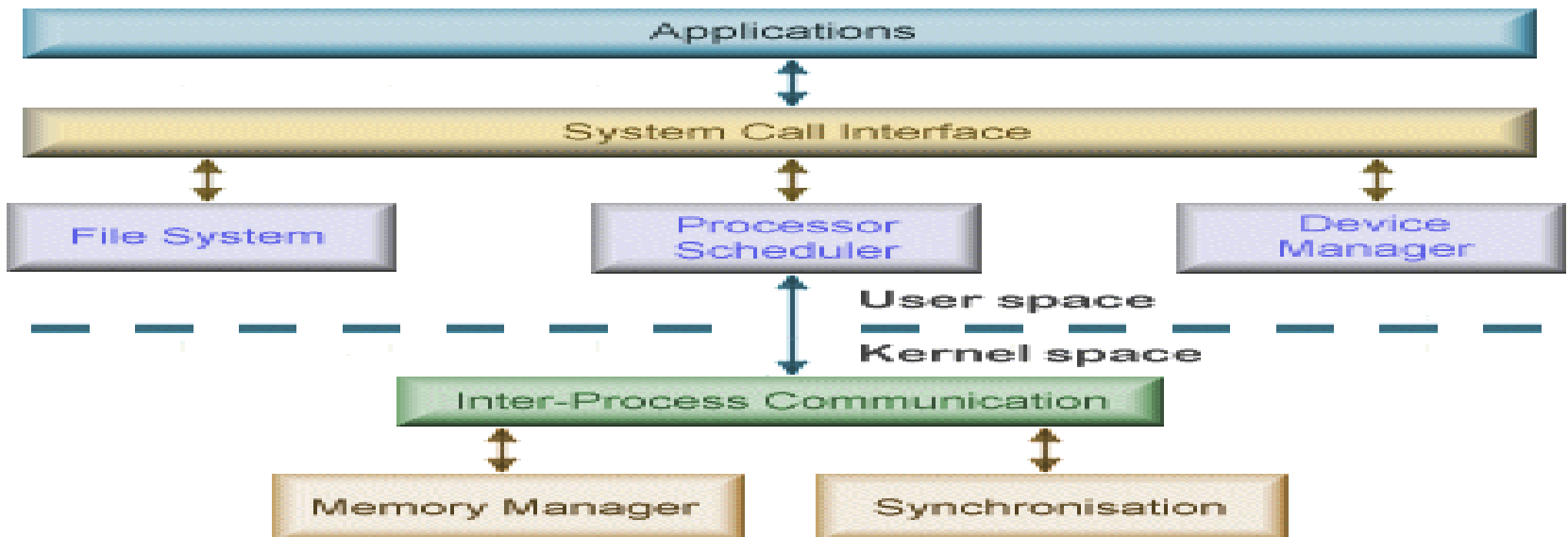


# Monolithic-Based OS Kernels

- Major advantage:
  - Cost of module interactions is low (**procedure/function/method call**)
- Disadvantages:
  - Hard to understand,
  - Hard to modify or to maintain,
  - Unreliable (**no isolation between system modules**)
- What is the alternative?
  - Can we moves some components from the **kernel** into “**user**” space to simplify its design and implementation?

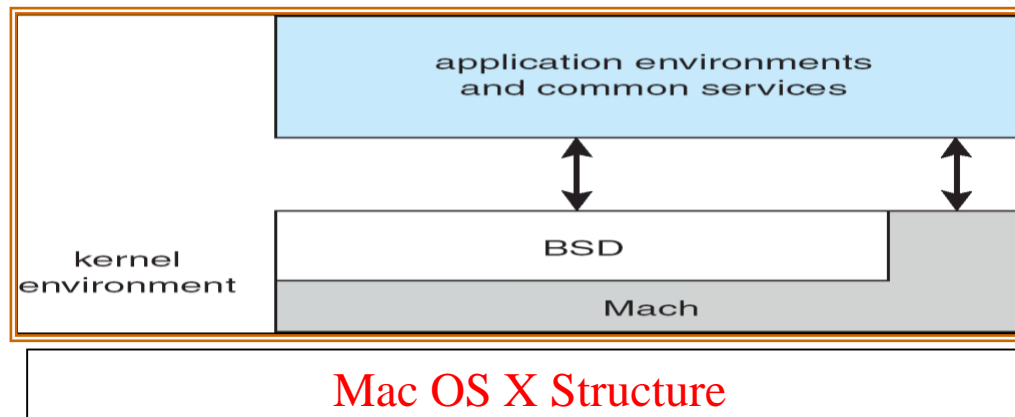
# Microkernel-Based OS Structure

- Moves as much as possible from the kernel into “user” space
- Communication takes place between user’s modules using message passing
  - Provides only small number of services
    - Attempt to keep kernel small and scalable
  - High degree of modularity
    - Extensible, portable and scalable
  - Increased level of inter-module communication
    - Can degrade system performance
- Mac and BSD (Berkeley Standard Distribution) OSs are examples.



# Microkernel-Based OS Structure

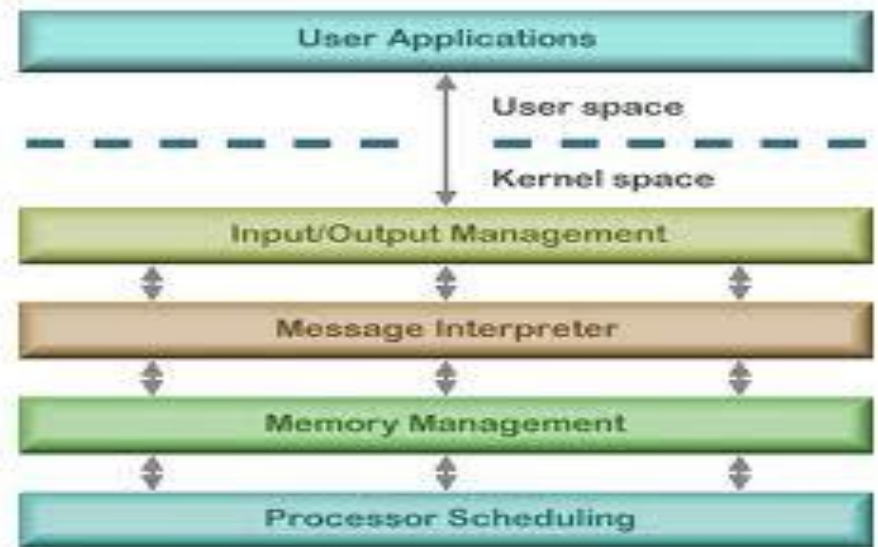
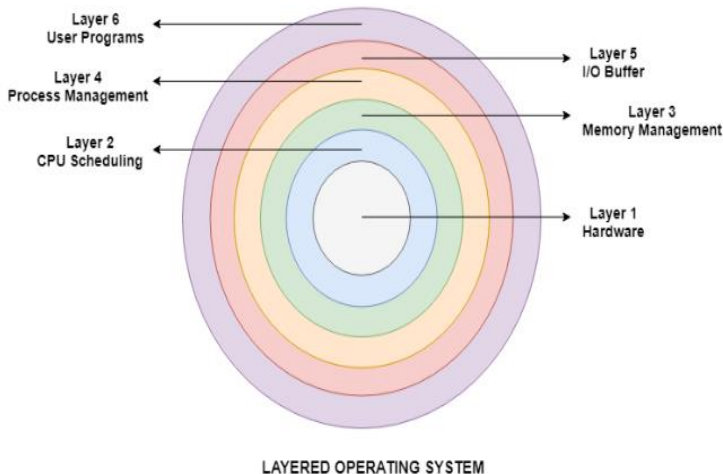
- Benefits:
  - Easier to extend a microkernel,
  - Easier to port the operating system to new architectures,
  - More reliable (**less code is running in kernel mode**),
- Disadvantages:
  - Performance overhead of user space to kernel space communication.



- Can we organize the OS into layers in order to simplify its design and implementation?

# Layering-Based OS Structure

- The layering approach means:
  - Implement the OS components as a set of layers,
  - All the layers can be defined separately and interact with each other as required,
  - Each layer acts as a 'virtual machine' to the layer above.
- Layered Systems
  - **Layer 1:** Responsible for the multiprocessing aspect of the operating system. It decides which process to be allocated to the CPU (Scheduling). It deals with interrupts and performs the context switches when a process change is required.



## Layering-Based OS Structure

- **Layer 2:** Concerns with allocating memory to processes.
- **Layer 3:** Deals with inter-process communication and communication between the OS and the console.
- **Layer 4:** Manages all I/O between the devices attached to the computer. This includes buffering information from the various devices.
- **Layer 5:** Where the user programs store.
- **Layer 6:** Where the overall control of the system (called the system operator).

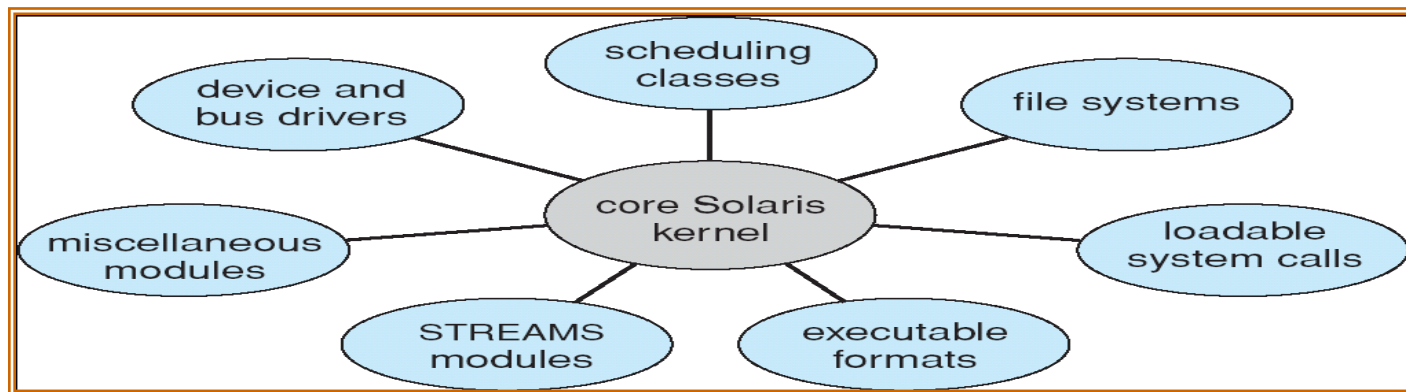
# Layering Advantages and Disadvantages

- Advantages:
  - Each layer can be tested and verified independently,
  - Layering eases maintenance, developing, and updating of system,
  - Explicit structure allows identification, relationship of complex system's pieces.
- Disadvantages:
- Disjunction between modularity and reality
  - Systems modeled as layers, but not really built that way.
- Strict layering isn't flexible enough
  - A layer can communicate only with the lower layer.
- Poor performance
  - Each layer crossing has **overhead** associated with it.



## Modules-Based OS Structure

- A **modular operating system** is built with its various functions broken up into distinct processes, each with its own interface.
- Most modern operating systems implement the kernel as modules
  - Uses object-oriented approach,
  - kernel does not have to implement message passing since **modules/Functions are free to contact each other directly**.
  - 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 flexibility**



Solaris Modular Approach

# OS Design to support Virtual Machines

- Creating software copies of the processor (**the capability to execute instructions**) and the memory (**the capability to store information**), each constitutes a Virtual Computer (VC).
- The resources of the physical machine are shared. **Virtual devices are sliced out of the physical ones. Virtual disks are subsets of physical ones.**
- Useful for running different OS on the same machine.
- To provide an interface or resource that differs from that of the lower levels.
- Primarily used for cross OS compatibility (**portability of code**), i.e. Java VM.
- Protection is excellent, but no sharing possible.

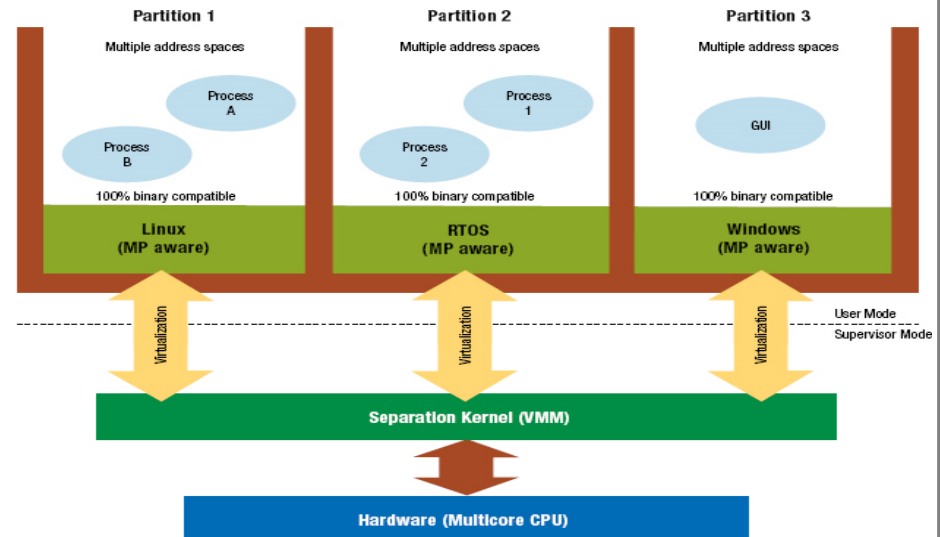
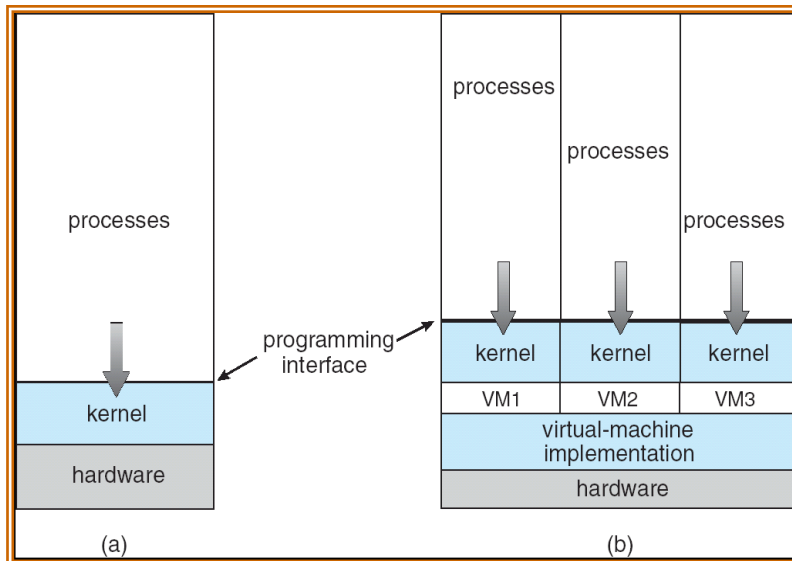
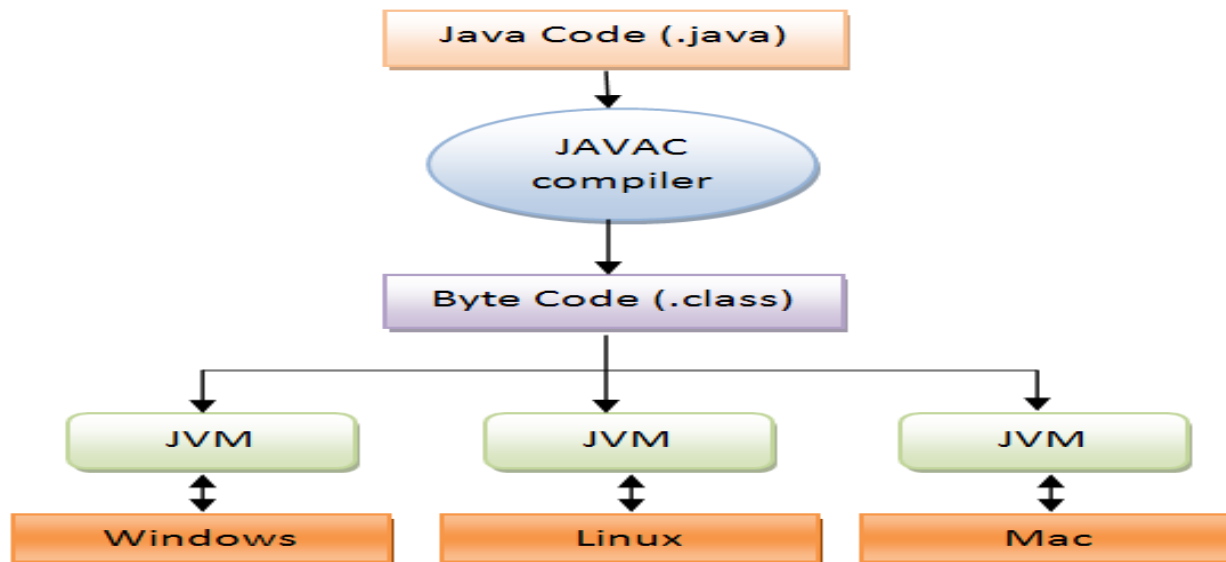


Figure 1 Virtualized OS architecture on a Multicore processor.

# Java Virtual Machine: Example

## Java Virtual Machine:

- Mediates between the application and the underlying platform:
  - Converts the application's byte code into machine-level code.
  - Handles tasks such as managing the system memory, providing security against malicious code, and managing multiple threads of program execution.
- **Thus:** Compiled Java programs are platform-independent, byte-codes executed by a Java Virtual Machine (JVM).



- The Java Virtual Machine allows Java code to be portable between various hardware and OS platforms.

### Advantages:

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.

### Disadvantage:

- The isolation, however, permits no direct sharing of resources which is against sharing of the resources to maximize their utilization.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

## System Implementation

- Traditionally OSs have been written in assembly language and that means it is available only for that processor's family.
- Modern OS are often written in Higher-Level Languages such as C, C++, Java.
- i.e. Unix is written in C and it is available on a different CPUs.
- The advantages of using HLL in writing the OSs are:
  - Can be written faster.
  - Is more compact.
  - Is easier to understand and debug.
  - An OS is far easier to port (move to some other hardware) if it is written in a high-level language.
- The major disadvantages are:
  - Reduced speed and increased storage requirements.
- In many OSs the critical modules (CPU scheduling, memory management) can be written in assembly to improve the performance of the OS.

## System Generation (**SYSGEN**) Module

- **OS** is designed to run on any machine; the OS must be configured for each computer, this process is known as **system generation**.
- The OS is often distributed on CDs and to generate a system, we use:
- **SYSGEN Module** obtains information concerning the specific configuration of the hardware system, it may ask the system operator to define the HW or probes the HW directly. **SYSGEN** must determine:
  - What type of the CPU? **Type of instruction set, floating point arithmetic, ..**
  - How much memory is available? **Some OS will detect the amount of memory by referencing the memory locations sequentially until getting into illegal address.**
  - What devices are available? The OS needs to know how to address each device, the characteristic of the device, **etc.**
  - What OS options are desired? Like how many buffer and of which size should be set? The type of CPU scheduling algorithm, the maximum no. of process to be supported? **etc.**
  - Plug and play feature.
- After the OS is generated, it must be available for the use by the HW through:
- **Bootstrap program/BIOS**: Code stored in ROM that is able to locate the kernel, to load it into memory, and to start its execution.
- **Booting**: The procedure of starting a computer by loading the fundamental parts (**kernel**) of the OS.

- We've completed our intensive introduction of the OS.
- This was a satellite picture about the concepts and principles of the OS.
- From now, we'll be at the ground level, looking at each component of the OS in more details.
- We will conduct a quiz soon about the introductory classes of the OS.  
Please prepare your self for that.
- Your references for the introductory sessions are:
  - Lectures' slides
  - Text book chapters 1, 2 and parts of chapters 3 and 13.



**The End!!**

**Thank you**

**Any Questions?**