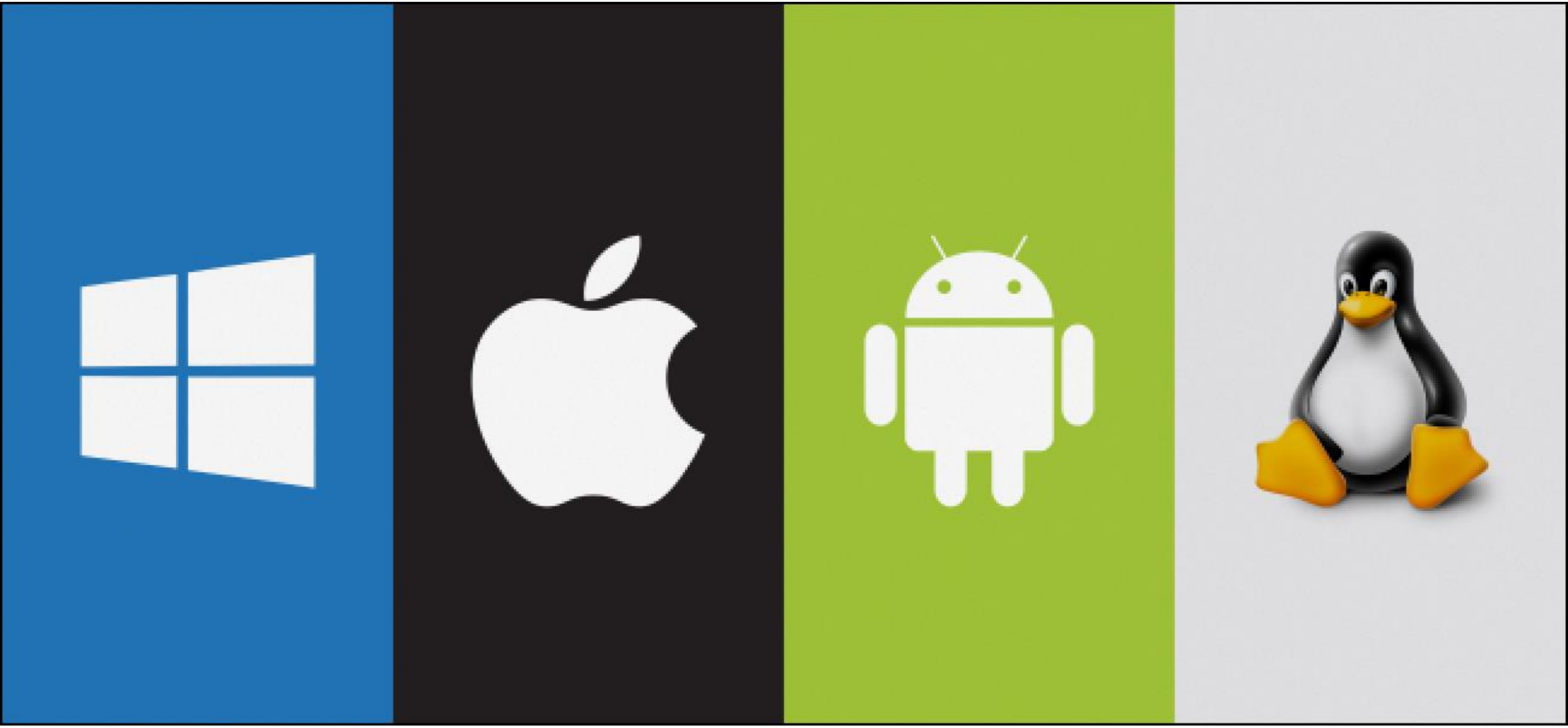
CSE 4305
Computer Organization and Architecture

Operating System Support

Course Teacher: Md. Hamjajul Ashmafee

Lecturer, CSE, IUT

Email: ashmafee@iut-dhaka.edu



Operating System Program

- A **program** that controls the execution of other programs and acts as an interface between applications and computer hardware
- Main **functionalities**:
  - Manage computer resources (**specially** memory management)
  - Provide services to the programmer (Interface)
  - Schedule the execution of other programs (Scheduling)
- Main **objectives**:
  - **Convenience:** make computer more convenient to **use**
  - **Efficiency: manage** all computer resources in efficient manner

OS as User Interface

- **User** uses an application as **end user** – not concerned about underlying computer architecture
  - User views a computer in terms of application (from where user gets services)
- **OS** provides hardware and software support for any application
- Application programmer **develops** an application program
  - Application program is a **set of processor instructions** that control the computer resources to provide services to a user
- To control the computer resources easily, a set of **system programs** are provided – **utilities**
    - Most important utility program – **OS**
    - **OS masks** the details of the hardware from the programmer, providing an **interface** to use the system

System Program (OS) Services

- **Program Creation**
  - Provide ***editor*** and ***debugger*** to assist the programmer
    - These services are **not the part** of OS, but they are accessible **through** OS
- **Program Execution**
  - A number of **steps** are required to execute a program, like:
    - **Load** instruction and data from main memory
    - **Initialize** I/O devices and file systems
    - **Prepare** other resources
  - A complete ***instruction cycle*** is involved to execute an entire instruction controlled by the OS

System Program (OS) Services…

- **Controlled Access to Files**
  - OS understand the **format** and **details** of the storage medium (register, cache, main memory, secondary storages and so on)
  - For ***multiuser system***, OS provides **protection** to access the files for authorized users (ensures *integrity* and *consistency*)
- **Access to I/O Devices**
  - I/O devices have their **own specific set of instructions** or control signals for operation
  - OS **hides** the details from the programmer
    - Rather programmer might **simply** think of **mere** read and write operation to contact with I/O devices

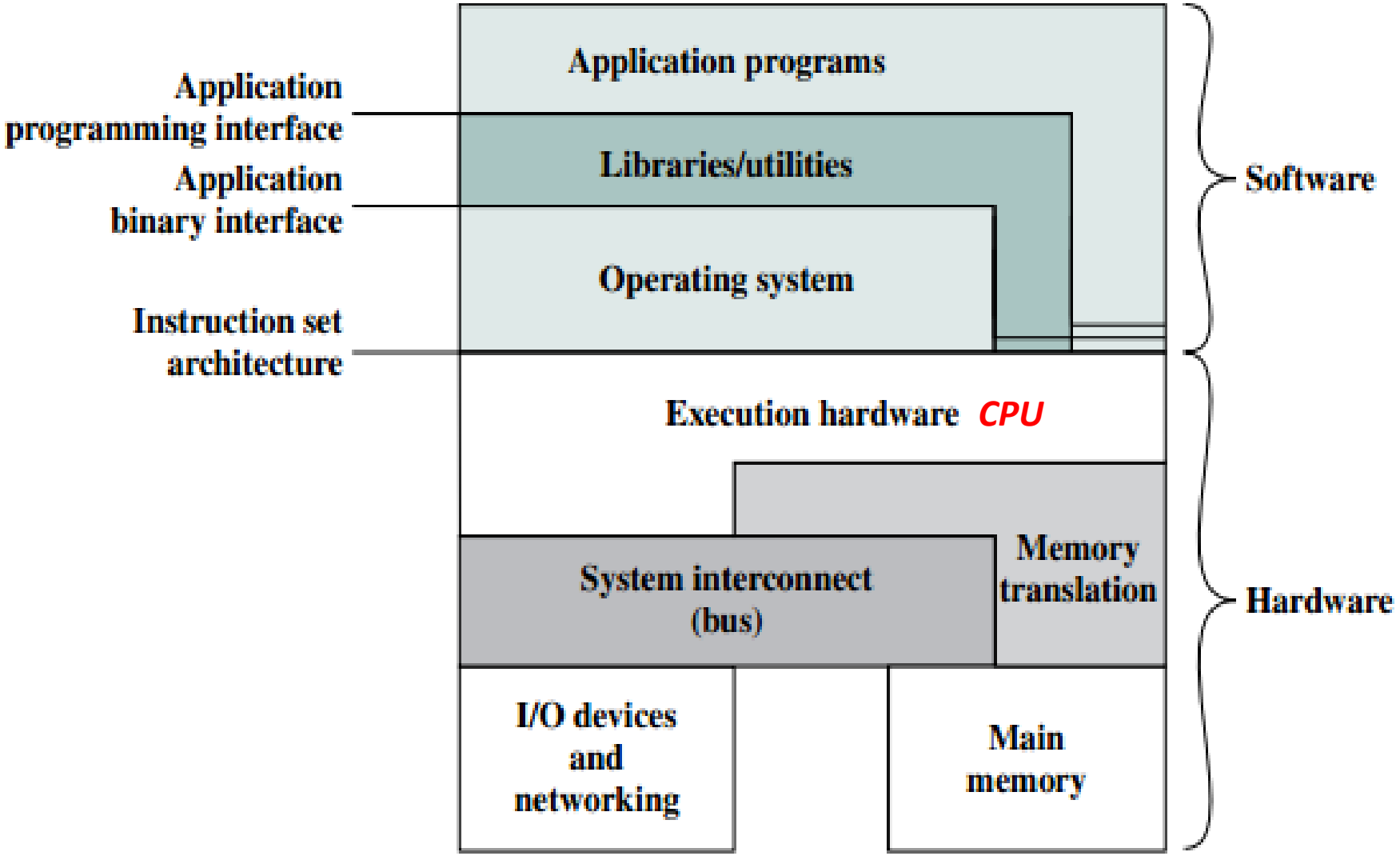
System Program (OS) Services…

- **System Access**
  - For **shared** or **public systems**, OS control access to the system and limited resources
  - Provide protection against unauthorized users
  - Resolve conflicts for resource contention/dispute
- **Accounting**
  - OS collects usage data of various resources monitoring their performances
    - Good to **anticipate** future enhancement and **tune** the performance
    - For multiuser system, to **prepare** billings

System Program (OS) Services…

- **Error Detection and Response**
  - **Kinds of errors:**
    - **Internal or external hardware**: memory error, device failure or malfunction
    - **Software:** arithmetic overflow, access to forbidden memory location, inability to grant request
  - **Responses from OS:**
    - Abort the current program
    - Retry the operation
    - Report the error to the application

Hardware and Software Structure



Key Interfaces in Computer System

- **Instruction Set Architecture (ISA) Interface**
  - **Collect** machine language instructions and **translate** them into corresponding control signals
  - **Boundary** between hardware and software
  - **Accessed** by application programs (***user ISA***) and utility programs (***system ISA***)
- **Application Binary Interface (ABI)**
  - **Defines** different system call interfaces to OS to take proper actions (by OS)
  - **Presents** different H/W resources and services available in the system to the User through user ISA
  - **Enables** binary portability across (any) programs or applications
    - Programs from any platform can get access to the system for such same ABI portability
- **Application Programming Interface (API)**
  - **Gives** a program **access** to the H/W resources and services using supplementary high level language (HLL) library functions through user ISA
  - **Enables** an application easily **portable (across any OSs)** having same API

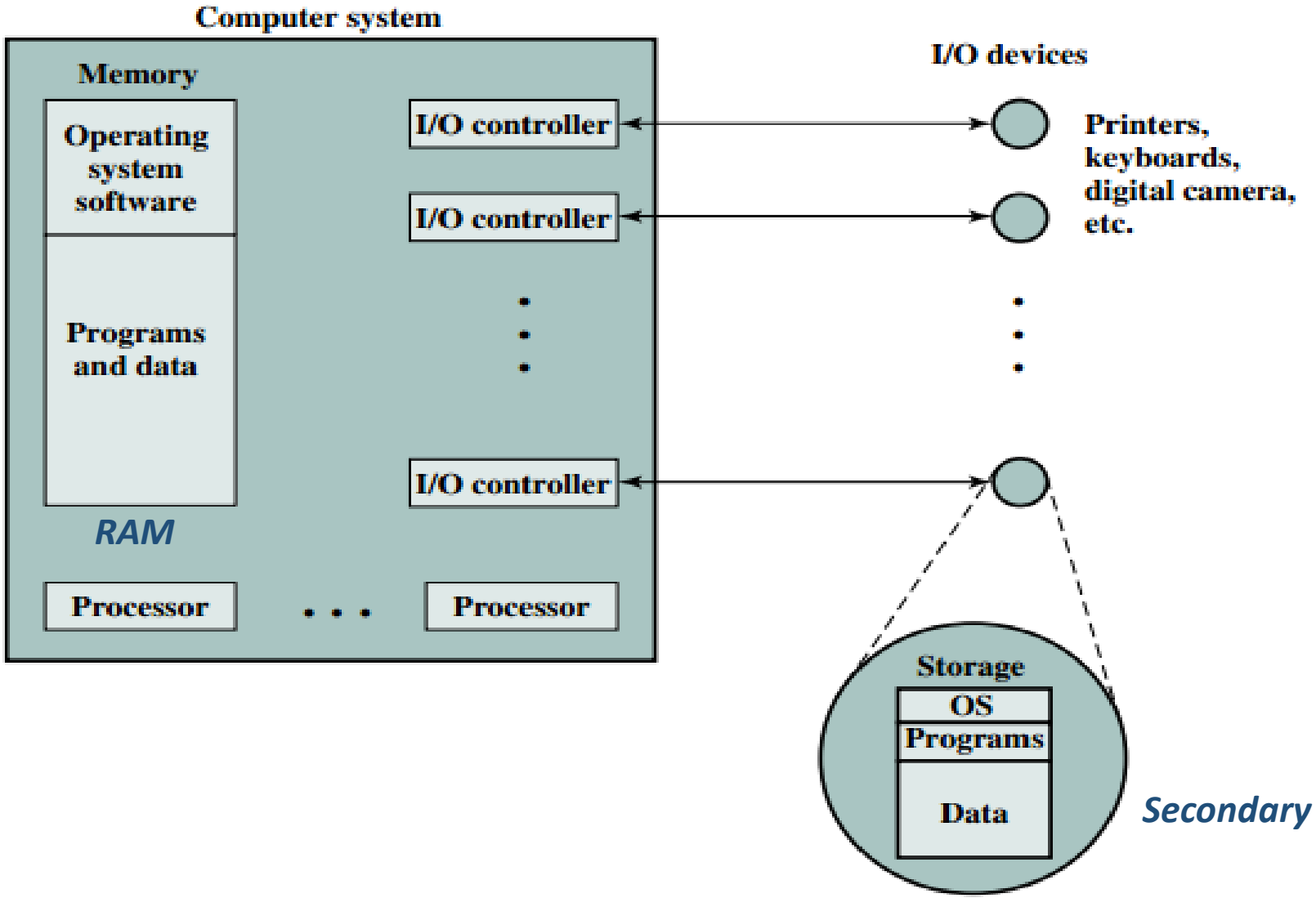
OS as Resource Manager

- Computer is **a set of resources** for data movement, storage and processing along with some others for controlling them
  - **OS is responsible to control these resources**
- Normally controller of a system is something that is a separate part from that system which is under controlled
  - That is not the case for OS
- OS is an **unusual control mechanism** because:
  - It is **executed** in the same way as any other ordinary computer programs
    - That OS itself is executed by the processor
  - It frequently **releases** its control and depends on processor to regain it.

OS as A Usual Computer Program

- OS as a program, provides instructions for execution to the processor
- **Moreover,** OS directs processor to utilize system resources and to execute other programs in time (*a special program*)
- To execute other programs, processor requires to stop executing OS
- While operating the computer, a certain portion of the OS should be in main memory
  - That portion of OS contains the most frequent functions – **Kernel or Nucleus**
- **Memory allocation** as a resource is controlled by **OS** and **processor** to store a **kernel** (OS) and share with other programs and data
  - Processor itself is a resource – shared by OS to execute its instruction

OS as Other Computer Program



Types of OS

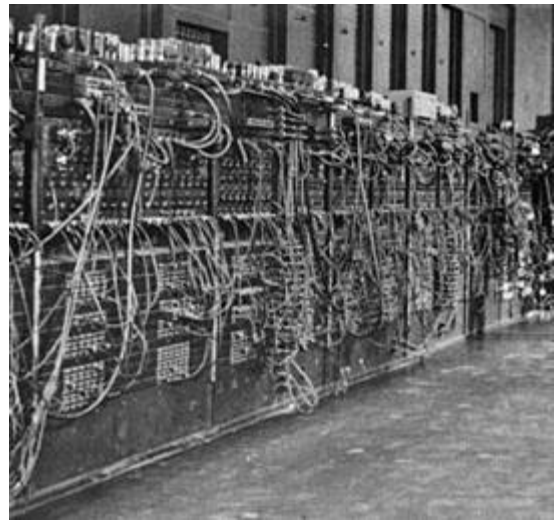
- Based on interaction with computer:
  - **Interactive system**
    - User or programmer **interacts directly** with the computer executing several requests
    - Through keyboard, display terminal
    - Also possible to communicate with computer during execution of a job !!!
  - **Batch system**
    - User program **batched** (grouped) together (also from other users) and **submitted** to the computer by a computer operator
    - After the program is completed, results are printed out for user as a final feedback regarding the compilation of the user program

Types of OS…

- Based on number of programs under execution:
  - **Multiprogramming**
    - To keep the processor as busy as possible having more than one program at a time
    - **Several programs** are loaded into main memory and processor **switches** rapidly among them
  - **Uniprogramming**
    - That works only one program at a time

Early Computer Systems [1940-1950]

- Programmer interact with the hardware directly – No OS
  - **Hardwired Programming**
- The **hardware-console** of the processor was consisted of:
  - Display lights – to indicate any error condition
  - Toggle switch
  - Input device – to load program (card reader)
  - Printer – to produce output
- Mode of operation – **serial processing**
  - Users have access to the computer in series



Early Computer Systems: Problems

- **Scheduling**
  - A sign-up sheet to reserve processor time that should be strictly maintained
    - A program can't be started earlier than its scheduled time even if its previous program has already left the processor
- **Setup time**
  - To start executing a program, it needs to load compiler, linker and others files which implies mounting and dismounting tapes or setting up card decks
    - Takes a lot of time to be started
    - Highly erroneous task that if there is any fault the program needs to be started from very beginning

Simple Batch OS Systems

- Early processors were very expensive
  - It was important to maximize the processor utilization
  - **Solution** – **Simple Batch System** (also known as **Monitor**) – keep jobs ready to use
- User no longer has the direct access to the processor
  - **User** submits the job on card or tape to a computer operator who will group (batch) them sequentially
  - **Computer operator** will place the entire batch on an input device to be used by the **monitor**
- **Monitor** – Controls the sequence of events
    - A portion of monitor must be in main memory for execution – **Resident Monitor**
    - The rest of monitor consisting of utilities and common functions are loaded when required by user program
    - Handle scheduling problems of the jobs in waiting

Monitor: How It Works?

- Monitor reads in a job at a time sequentially from the input device (e.g. card reader or magnetic tape)
  - When the current job is placed in the user program area of main memory, the control is passed to this job for execution
  - When the current job is completed, it will return the control to the monitor
  - Monitor will then read in next job in the user program area of main memory

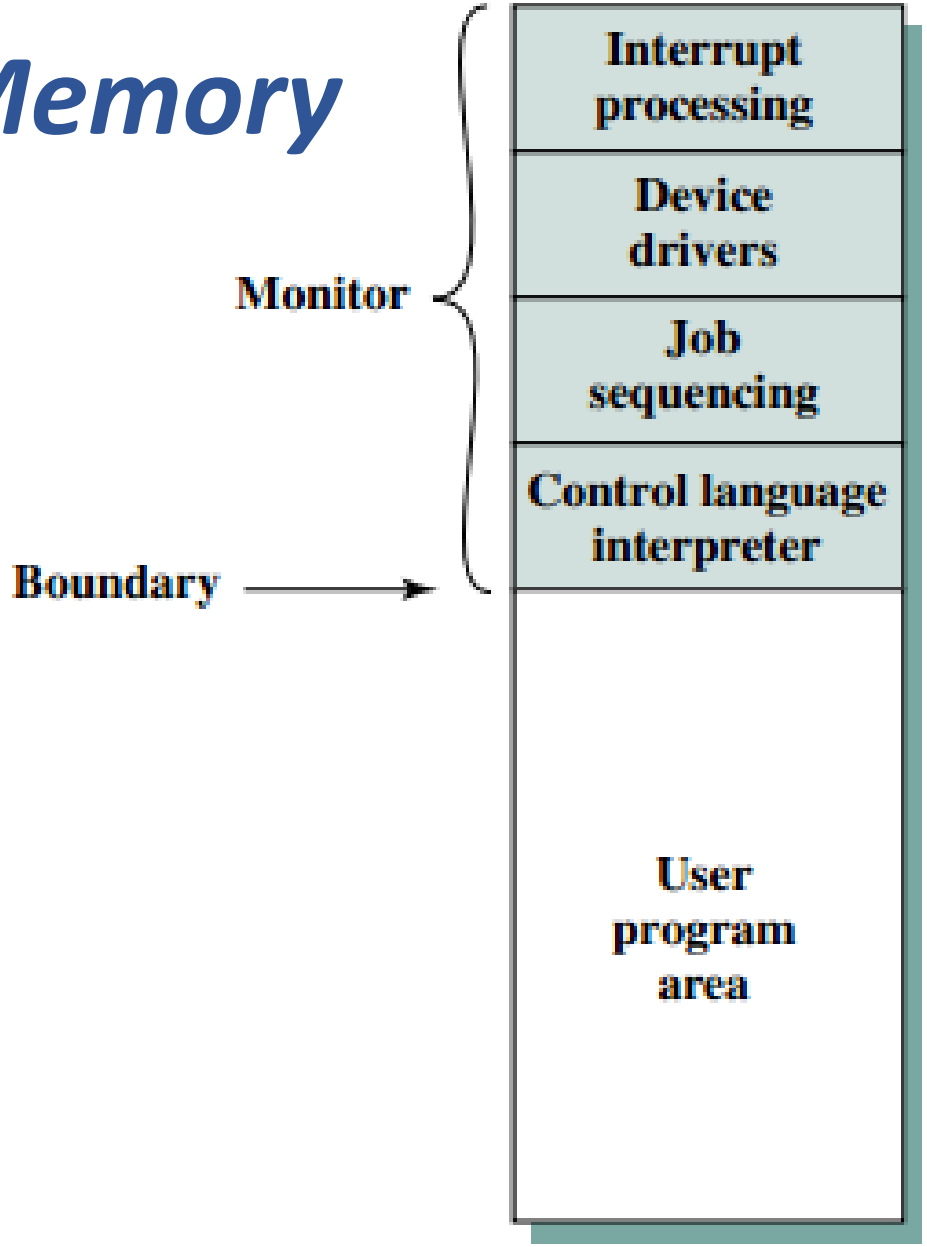
[Note: From the point of view of the **monitor** itself.**]**

Monitor: How It Works?...

- At the beginning, the processor will start executing instructions from the monitor (resident) in main memory
  - Execution will cause to read in next ready job to the main memory (in user program area), and processor will next branch to the job brought in main memory ***(control passed to the job)***
  - The processor will fetch and execute the instructions from the user program (current job) until ending of the program or any other error condition
  - Then processor will fetch and execute next instruction from the monitor ***(control returned to the monitor)***

[Note: From the point of view of the **processor]**

Monitor in Memory



Required Hardware Features for Batch OS

- **Memory protection**
  - To protect certain memory boundary whether to be violated or not
  - If it happens, job will be aborted and control returns to the monitor
- **Timer**
  - Prevent a job to monopolize the system
  - A timer is set at the beginning of each program to supervise its expiration
    - Cause an interrupt to return control to the monitor if the timer expires
- **Privileged instructions**
  - Some instructions are regarded as privileged that only executed by the monitor
  - If processor encounters any of these instructions executed under any user program, cause an error to return the control to the monitor
- **Interrupts**
  - Nowadays OS has this feature of flexibility to relinquish and regain control from user programs through interrupts

Sacrifices Due to Batch OS

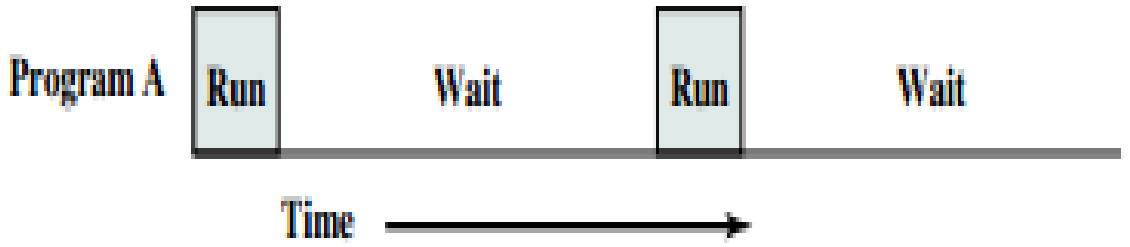
- Main memory space is **sacrificed** for monitor
- Processor time is **sacrificed** for monitor

Problems with Simple Batch System *[Uniprogramming]*

- Simple batch OS system (i.e. monitor) make the sequence of the jobs (Scheduling)
  - Still the processor sits idle – because I/O devices are slower than processor
  - After finishing executing instructions, the processor must wait until that I/O operation concludes to abort the current program and switch to the next
  - Also other resources are underutilized – like disk, printer, terminal and others are not being used simultaneously

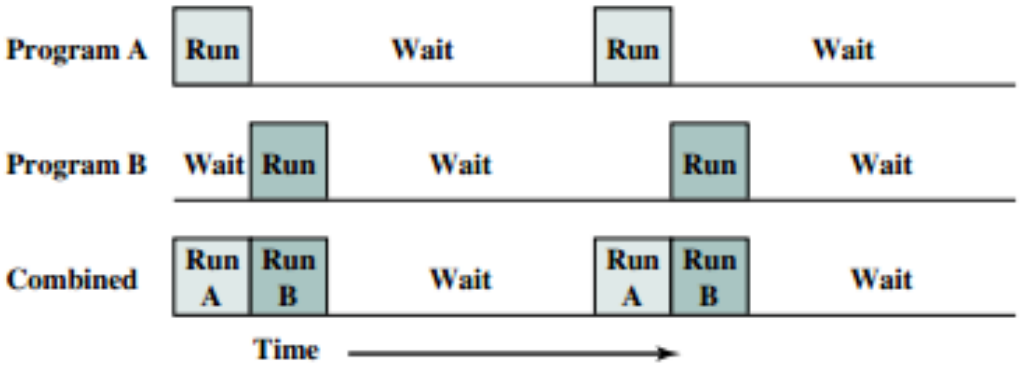
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

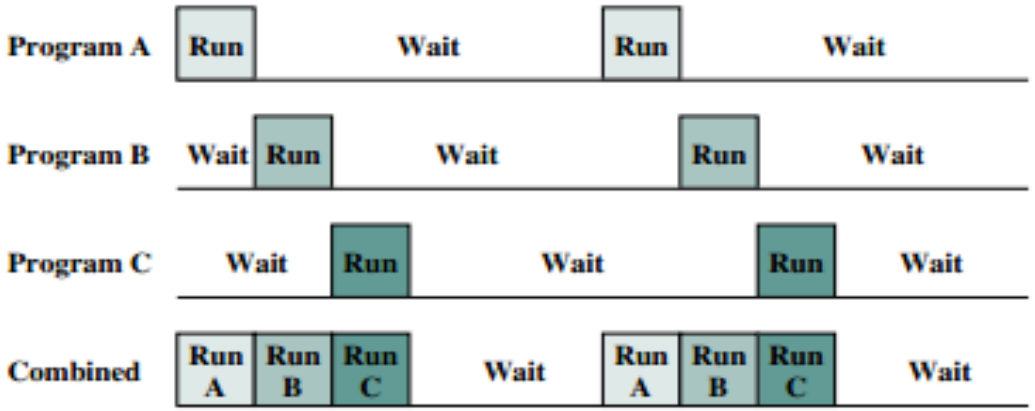


Multi-Programmed Batch OS System

- To overcome the problem of simple batch OS, now more than one user program will be placed in memory
  - When the processor needs to wait for I/O operation in one job, it will switch to the other job
    - Likely no waiting time for the processor
  - To get more efficiency, we might flourish memory to hold more programs and switch among them
    - All the jobs will run concurrently utilizing all the computer resources
  - **Central theme of modern OS**



(b) Multiprogramming with two programs



(c) Multiprogramming with three programs

More illustration in example 8.1 of chapter 8 – W. Stallings, Page 285

Multi-Programmed Batch OS: Hardware Features

- Allow certain hardware features – **I/O Interrupt** and **DMA**
  - Through **Interrupt driven I/O** and **DMA**, processor issues an I/O command on the behalf of one job and proceed to another one
    - Meanwhile I/O operations are carried out by the I/O device controller
  - When I/O operations are completed, the processor is interrupted by the device controller
    - In response, processor switches to an ISR to handle that interrupt.
  - After finishing ISR, the processor control will be passed to another job in sequence

More About Multi-Programming Batch OS

- **More sophisticated** than simple batch OS:
  - Needs several jobs ready to run in the main memory
    - **Memory management** is required (Coming Soon…)
  - Also the processor needs to decide which one will run next
    - Some sort of **scheduling algorithms** are demanded

Interactive OS System

- In different cases, it is desired to have a **mode** in which a **user** can interact with the computer directly
  - E.g. Transaction processing
  - **Past history:** for each user, it was required to have a dedicated computer, which was not available in 1960s
    - Rather than that, a **time sharing system** was developed with a single computer
- As **multiprogramming OS system** can handle multiple batch jobs, it can also handle **multiple interactive jobs** in **time sharing** manner **[Improvement]**
  - Now processor-time can be shared among **multiple users** through computer terminals
- OS **interleaves** the execution of each user program for a short time
  - If ***n*** users request for service at a time, each program will get on an average ***1/n*** of effective time from total service
  - No need to have dedicated computers as it is well enough with slower human reaction time

Batch Multiprogramming VS Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

i.e. Not allow user Interaction *i.e. Allow user Interaction*

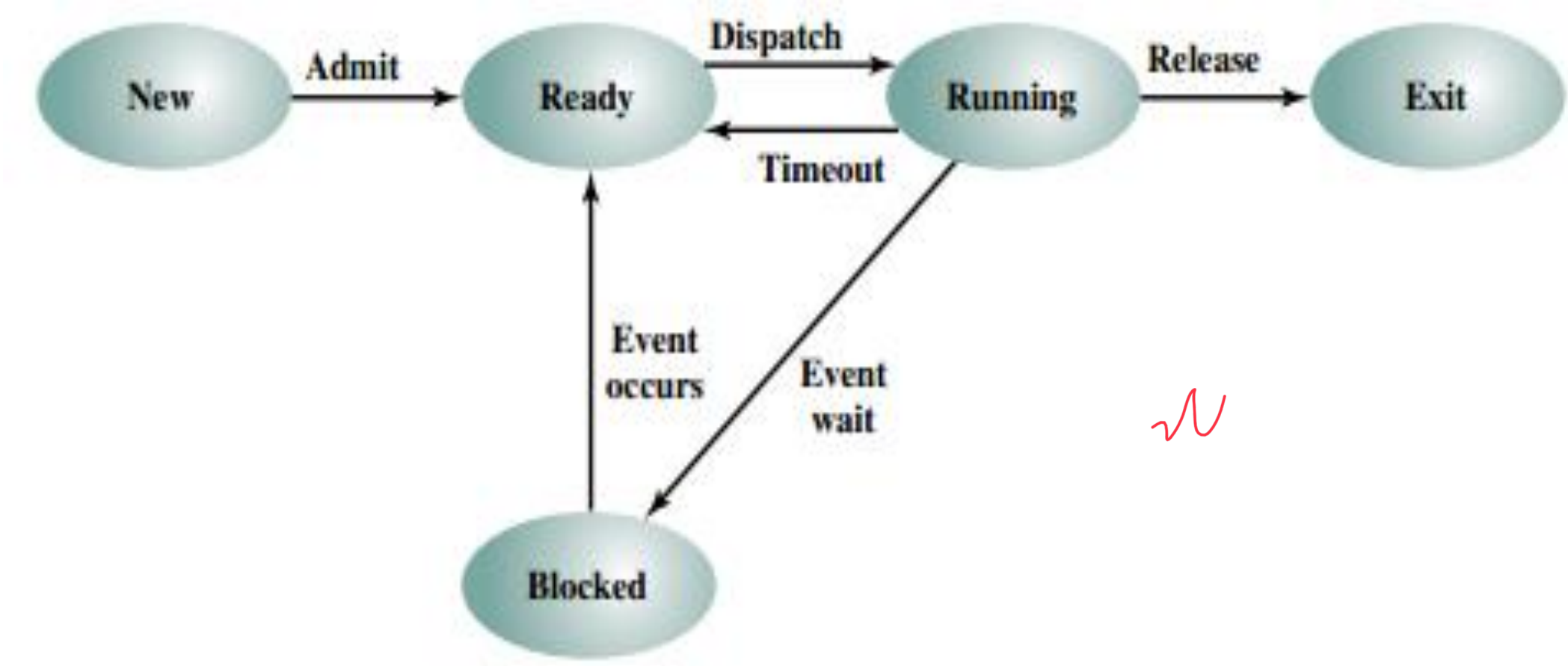
Concept of "Process"

- "***Process***" – this term was first used by Multics OS in 1960s
- As somewhat more generalized term than "**job**" - also defined as:
  - **A program** in execution
  - The "**animated spirit**" of a program (*active soul*)
  - As an **entity** to which a processor is **assigned**

Process States

- During the lifetime of a process, its status will be changed very frequently
  - Status at any point of time is referred as a **state** with some certain information that defines that status of the process
  - To learn the operation of scheduling, we have to know about process state
  - There are five states for a process:
    1. **New**
    2. **Ready**
    3. **Running**
    4. **Waiting/Blocked/Halted**
    5. **Exit/Terminated**

Five State Process Model



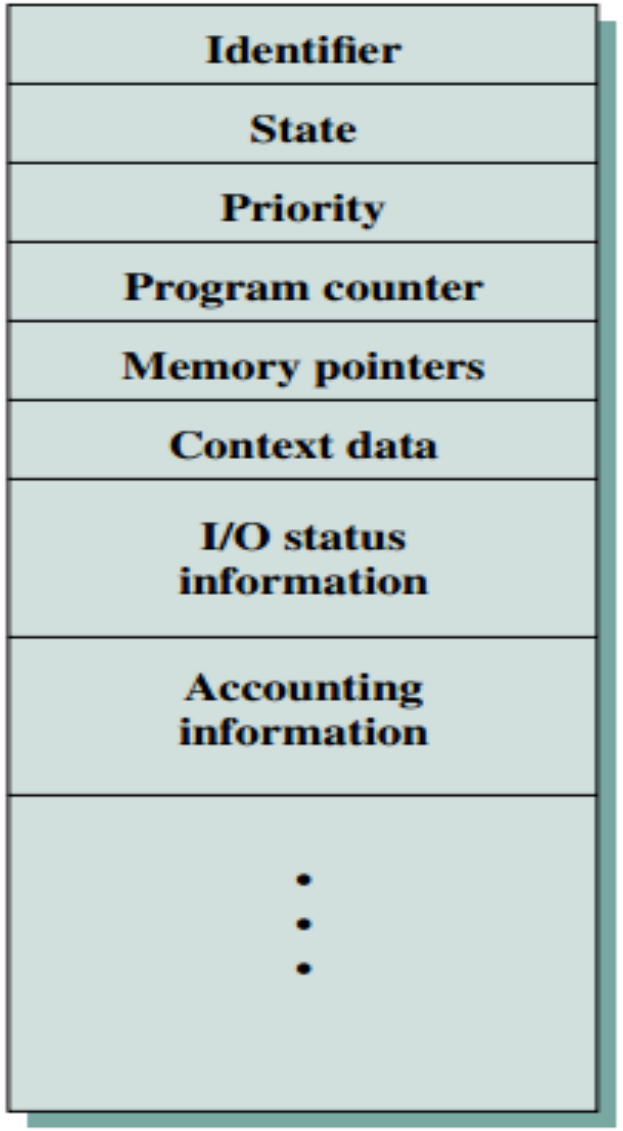
States of a Process

- **New**
  - When a program is admitted by the scheduler (long-term/high-level) but not yet ready to execute
  - OS will initialize this process and move it from ***new*** to ***ready*** state
- **Ready**
  - The process is ready to execute and is awaiting access to the processor
- **Running**
  - The process is being executed by the processor
- **Waiting/Blocked/Halted**
  - The process is suspended from execution into waiting for some system resources such as I/O devices
- **Exit/Terminated**
  - The process is terminated and will be destroyed by the OS

Process Control Block (PCB)

- For each of the process in the system, the OS must maintain its **information**
  - It indicates the **state** of the process and other necessary information necessary for **process execution**
- In this purpose, each process is represented before OS by a **P**rocess **C**ontrol **B**lock (PCB), which contains:
  - **Identifier:** Unique address of the current process
  - **State:** Current state of the process (e.g. new, ready, running, waiting, exit)
  - **Priority:** Relative priority level
  - **Program Counter:** The address of the next instruction in the process to be executed next
  - **Memory Pointers:** Starting and end address of the process in memory
  - **Context Data:** System register data during the process is executing (saved during context switching). It is required when the processor resumes the execution of the process
  - **I/O Status Information:** It includes I/O requests, I/O devices assigned to it, files assigned to it, and so on
  - **Accounting Information:** It includes amount of used processor time and clock time, time limits, account numbers and so on

PCB Block Diagram

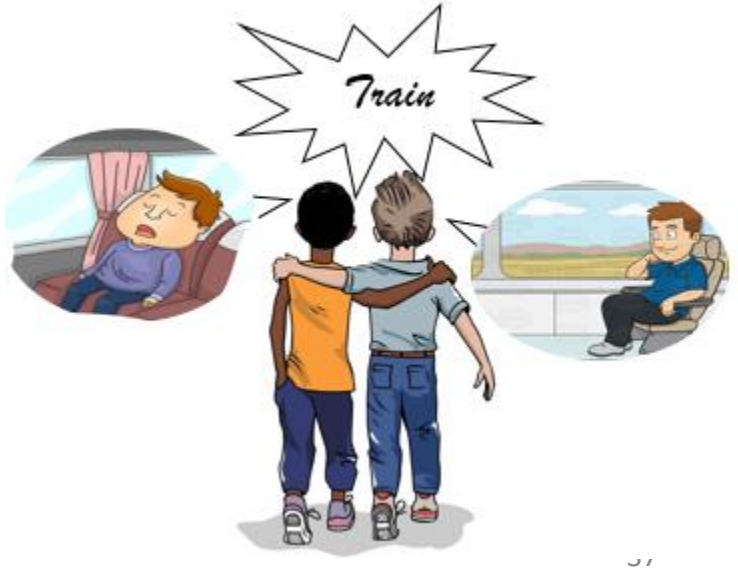


Scheduling and Its Types

- **Key** to the **multiprogramming** – ***Scheduling***
- 4 types of scheduling:

Long-term scheduling	The decision to add to the pool of processes to be executed.
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory.
Short-term scheduling	The decision as to which available process will be executed by the processor.
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device.

Analogy of Scheduling…



Long Term Scheduling

- This type of scheduling will determine among all the requests **which programs are submitted to the system for processing**
  - Controls the degree of multiprogramming (number of processes in the memory)
- Once a program/job is submitted to the system becomes a "**process**"
  - ***i.e.*** added to any one of the system queue of different schedulers
    - ***e.g.*** Based on systems, this process will be admitted to either short time scheduling or medium term scheduling
- **Long term queue** is the list of jobs waiting to use the system
    - Waiting to be a ***process***
    - If the conditions are met, these jobs will be allocated memory and enlisted to any of a queue

LTS: When a New Job Comes, What OS Will Do

- In **multiprogramming or simple batch** system:
  - Newly submitted jobs are held in a **batch queue** – Long term scheduler creates **process** from them based on following decisions:
    1. Whether OS can take one or more additional process
    2. Which job or jobs to accept and turns them into process
       - Decision may include **priority**, **expected execution time**, **I/O requirements** and others
- In **interactive** (time sharing) system:
  - A process request is generated when a **user attempts to connect** to the system
    - A requesting user can't be simply queued up and kept waiting for a long time (***expectation***)
    - Rather the OS will accept all the requests until the system is saturated (***preference***)
    - If the system is saturated under some circumstances, it will notify the user that the system is full and request to try again later (***exceptional***)

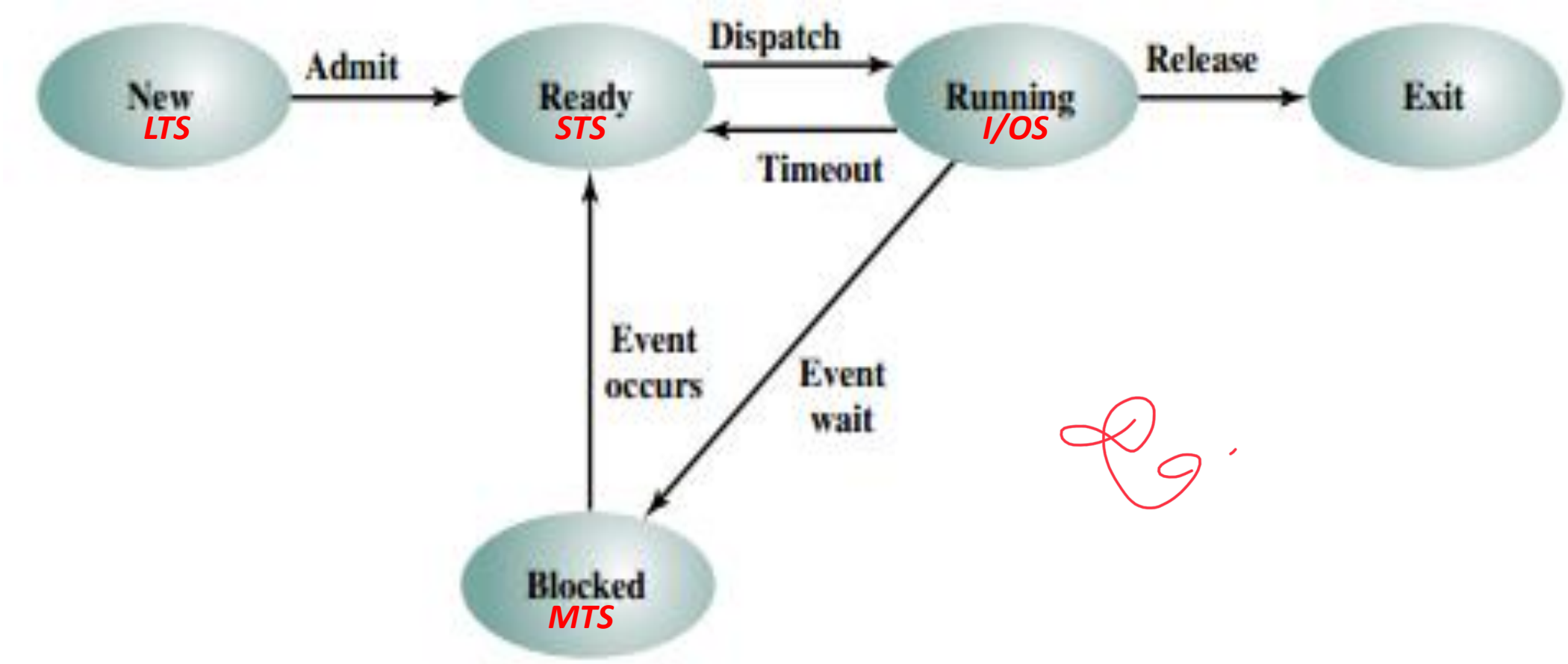
Medium Term Scheduling

- It is a part of **swapping function** (***i.e.*** swap in – swap out)
  - Relinquishing own space of a process from the main memory temporarily for another process (**swap out**)
  - Swapping in decision is made based on the ***degree of multiprogramming*** for the swapped out processes (**swap in**)
  - This scheduling is used when the processes are in the ***"waiting"*** state
    - These processes are enlisted in *medium term queue*

Short Term Scheduling

- Long term scheduler is used very infrequent
  - Whereas ***short term scheduler*** executes very frequently
- Also known as "***Dispatcher***"
- It takes very fine tuned decision to select from a **short term queue** which job will be processed next form the "***ready***" state
  - All this next process will be processed by processor
  - Normally "**round-robin**" algorithm is used to select next process so that each process will get its time slot in turn
    - Sometimes **priority** algorithm may also be used

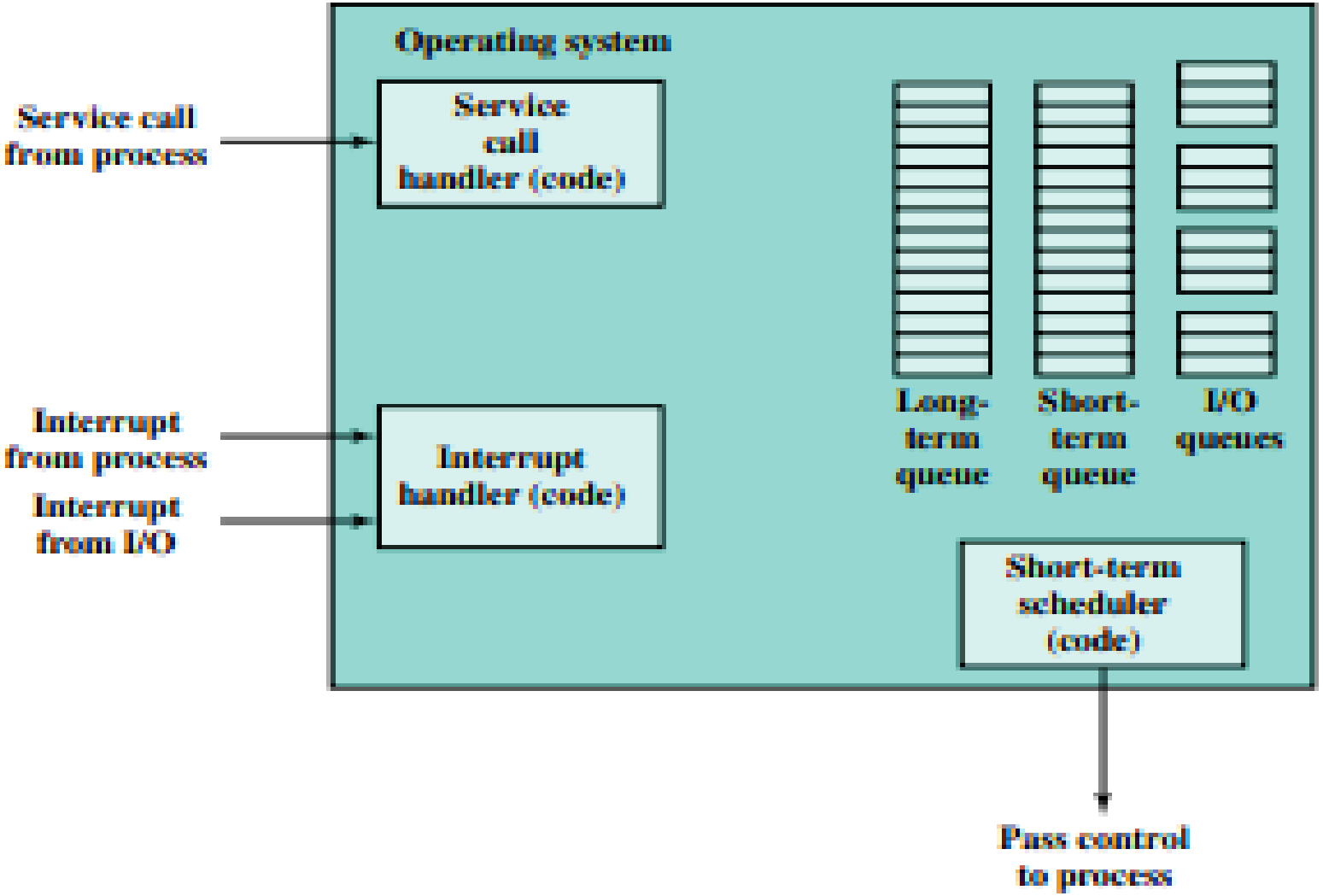
Schedulers in Five State Process Model



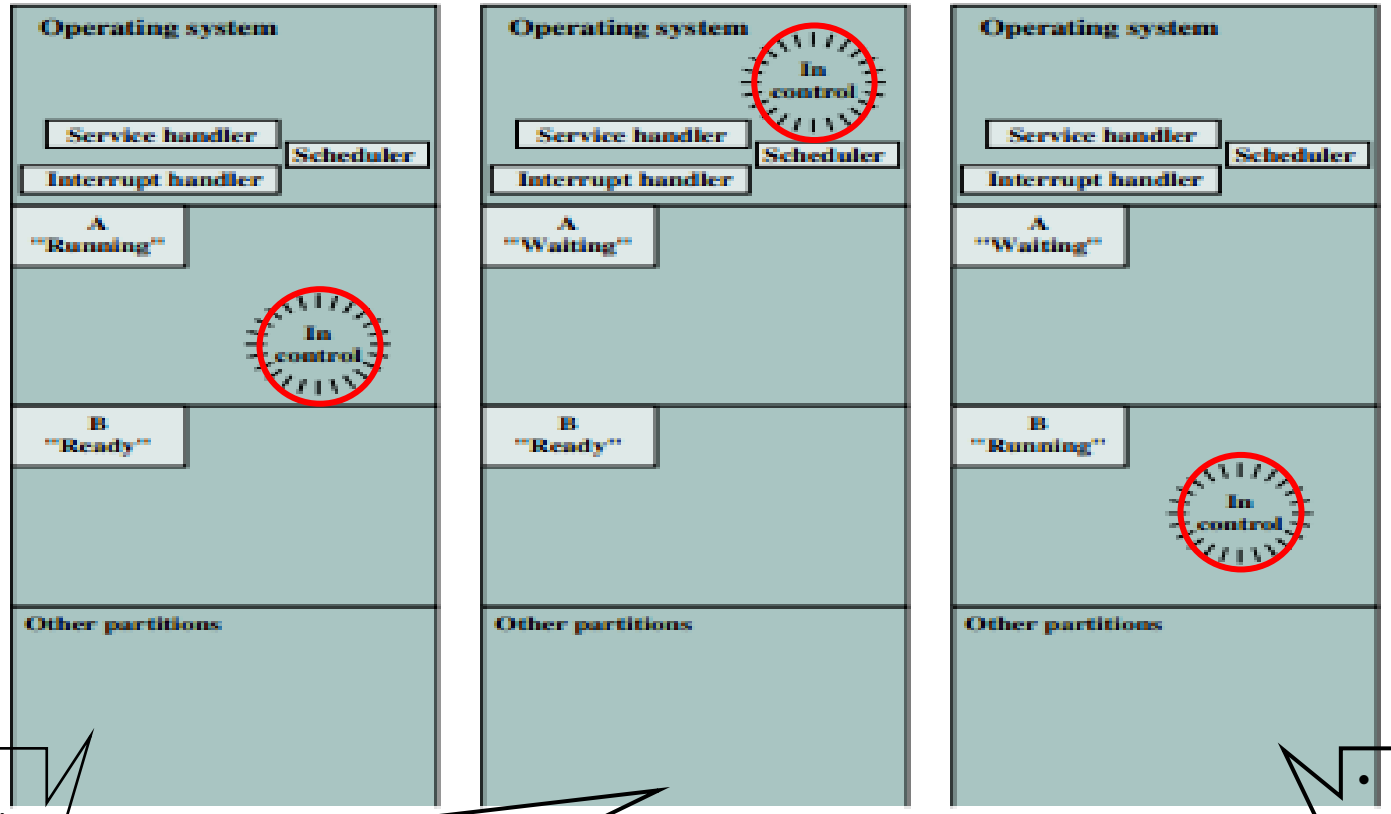
Scheduling Technique

- At a given point of time, **main memory is partitioned** into a number of **active processes** along with the **kernel** of the OS
- At any point of time, any active process or the OS itself can be being **executed in the processor**
    - All of them must be present in the main memory
    - Processor will take instructions from the program under those processes and execute them
- An **OS** may be executed **at any time** to provide services. For example:
    - Execute a **service call** (*e.g.* an I/O request) generated by any process
    - Response an **interrupt** raised by any process
    - Some **events unrelated** (*i.e.* outside) **to the current process** to which the OS must response
    - OS also maintains a number of **queues**
        - Each queue is simply a waiting list of processes waiting for some resources

Key Elements of a Kernel of OS



Scheduling Technique Example



- At the end of the process ***A***, current context data and PC will be stored in the PCB of ***A***
- Process ***A*** will be sent to **waiting** state
- Then the control transfers to the OS

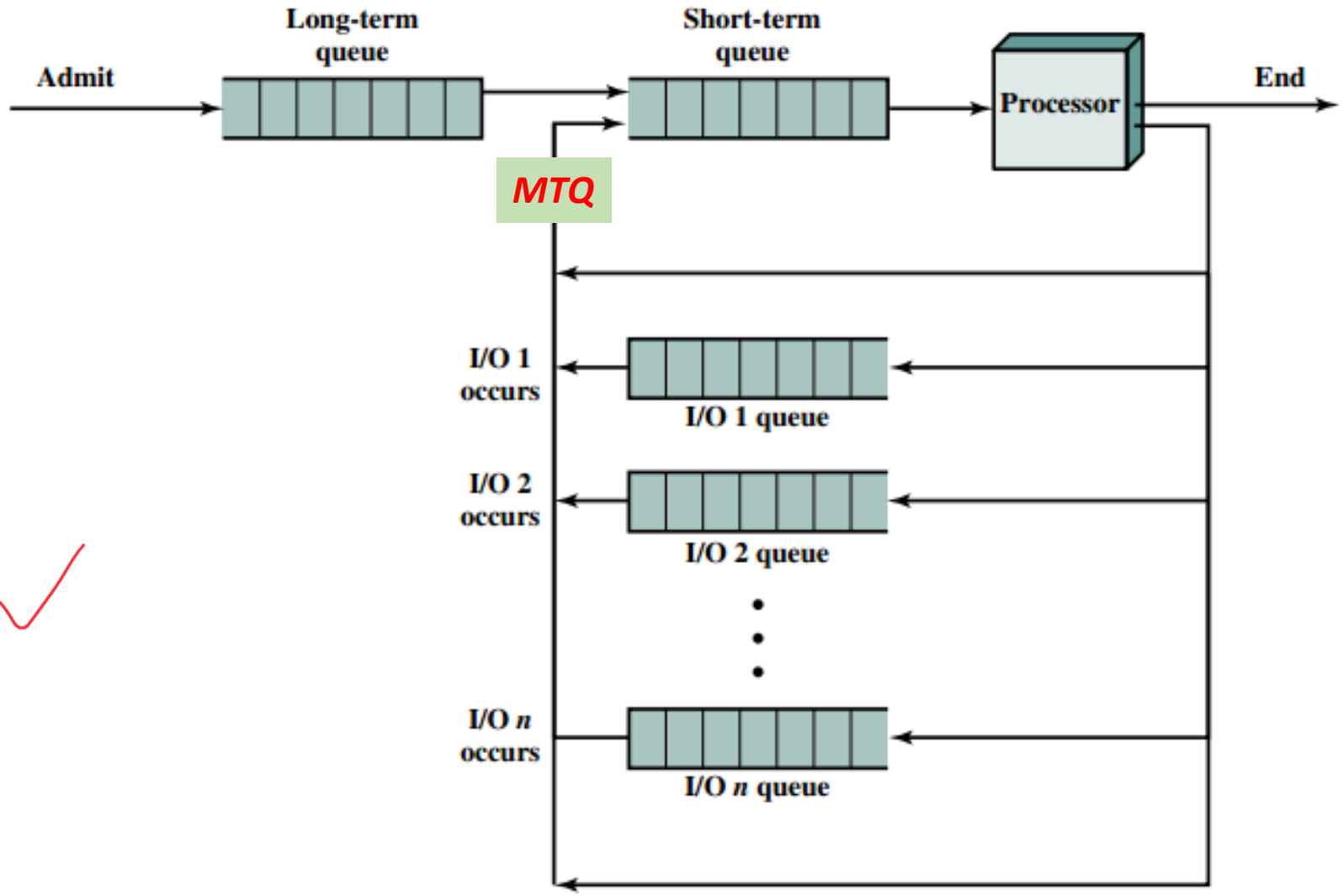
- OS performs some work – e.g. initiates I/O command for process ***A***
- Short term scheduler decides which process will be executed next from **ready** state
- OS instructs the processor to restore ***B'***s context data and PC

- Process ***B*** gets the control back along with its context and PC.
- Processor will proceed to execute process ***B*** through its instructions
- Process ***B*** is now in **running** state

I/O Scheduling

- It is used for **each** of **the I/O device**
  - Because more than one device will request for same device
  - For each of I/O device, OS will maintain an single I/O queue for them
- When an I/O operation is completed of a certain I/O device, the OS will remove the satisfied process from that I/O queue and place it in the short term queue (or medium term queue)
  - The OS will then select another waiting process for I/O service and give necessary I/O commands to that I/O device

All Queuing Diagrams of Processor Scheduling



Memory Management & Its Necessity

- For **Uniprogramming system**, memory is divided into **two portions**:
    1. OS (Resident-monitor/Kernel)
    2. User Program currently being executed
- For **Multiprogramming system**, "user" portion of the memory is subdivided to accommodate multiple processes
- Subdivision of main memory is carried out **dynamically** by the OS
    - Known as **memory management**
- If a few processes are placed in memory, then they will be **stalled** in "**waiting**" queue for I/O devices within very short time and processor will be **idle** again
    - Thus memory needs to be allocated efficiently to accommodate as many processes as possible

How to Improve Multiprogramming

- Normally there are three kinds of queues within a processor to keep it as busy as possible:
  - **Long term queue** – for new processes submitted to the processor
  - **Short term queue** – for ready processes to be executed by the processor
  - **I/O queue** – just left the processor for I/O operations
- But as processor is so much fast, all processes will complete quickly their processing tasks and wait for same I/O (**likelihood**)
  - So, even with multiprogramming , a processor could be idle most of the time
  - Main memory can be expanded to accommodate more processes (**alternative**) – **But:**
    - Even today main memory (RAM) is expensive (8~16 GB - limited number of processes)
    - Processes are growing larger (limited space for a process)
- Many techniques for OS to **expand** the main memory
  - Swapping, partitioning, paging, virtual memory, segmentation and so on.

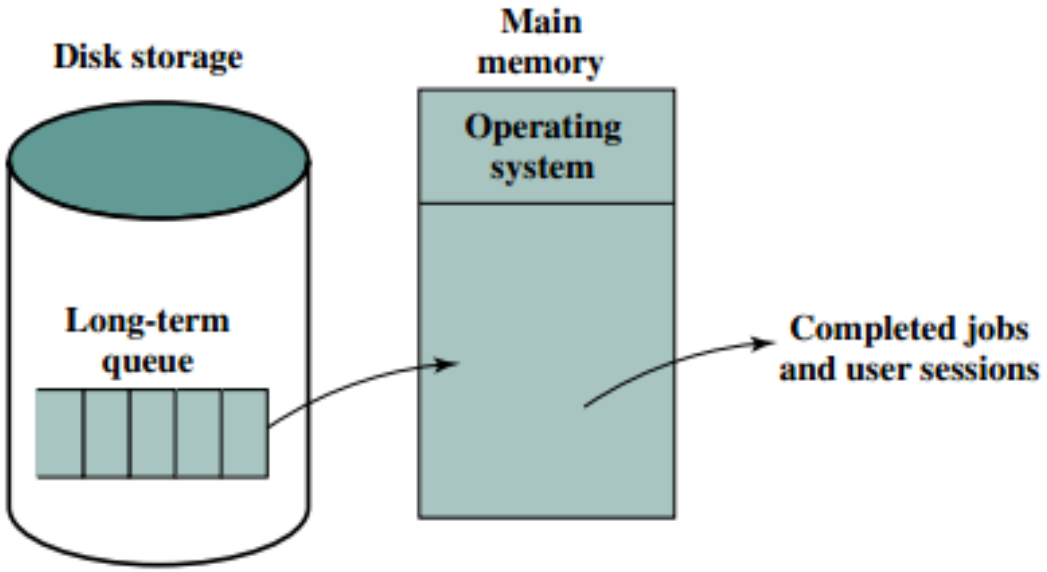
Swapping - Why

- We have already a **long term queue** of processes stored on the **secondary storage**
  - Processes are **waiting** outside the main memory to get access into the processor and main memory
  - Brought one at a time when there will be **space available** in main memory
    - **How? -** If existing processes are completed, they will be kicked out of memory leaving space for next
  - But as discussed earlier, processor is very fast nowadays and all the processes in main memory are in "**waiting**" state for I/O operations (**reality**)
    - Processor is still idle – we should think better way to get around this problem !!!
    - Figure for clarification

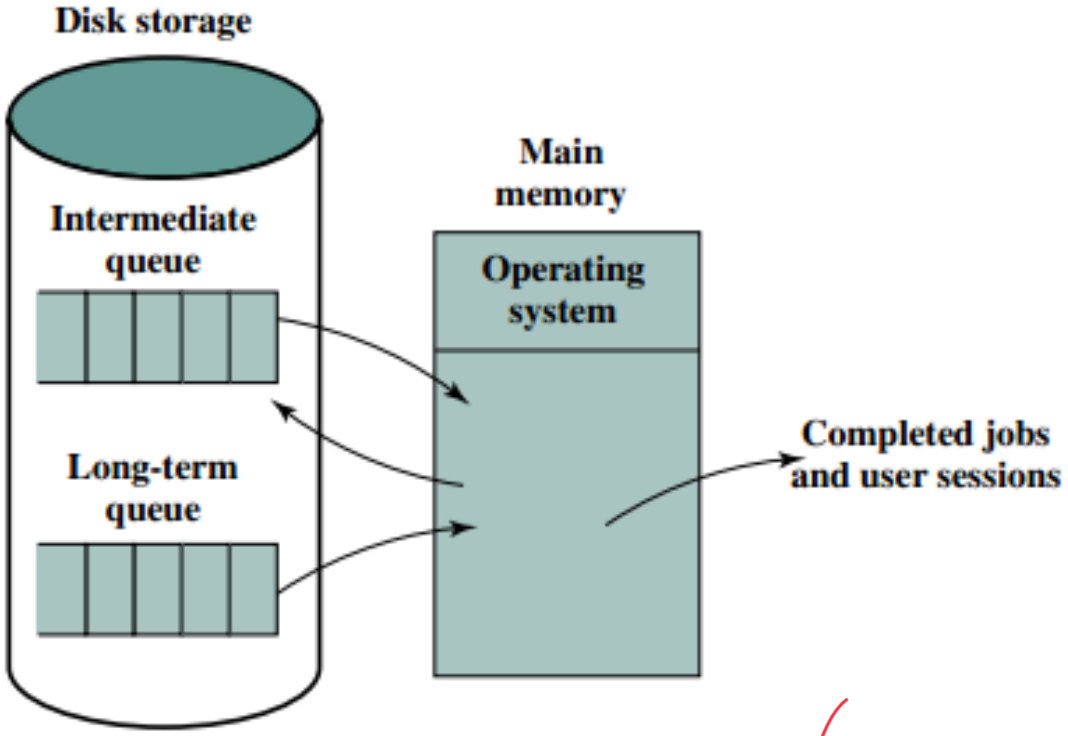
Swapping - Strategy

- Rather the processor swaps one of these **"waiting"** processes out of the main memory into **an intermediate queue** (or medium term queue) in disk
  - A **queue** of current processes which are temporarily blocked – **Swapping out**
- OS then **brings** into memory another process from the intermediate queue **or** honors a new process request from the long term queue – **Swapping in**
  - However, swapping is an **I/O operation** – very potential to make the problem worse to take longer time to access this secondary storage
    - But **disk drive I/O** is the fastest I/O on this system, *"swapping"* will enhance the performance
    - ***Virtual memory concept*** will also improves the performance over simple swapping

Depiction of Swapping



(a) Simple job scheduling

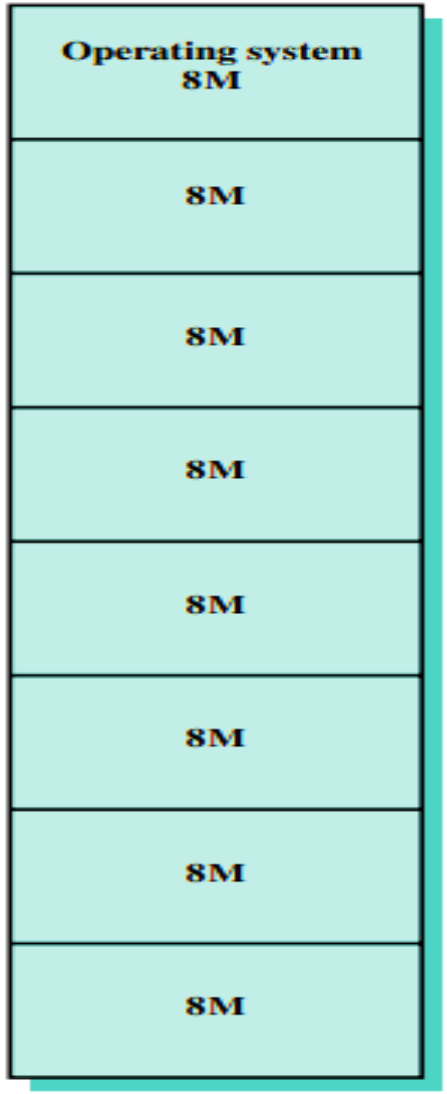


(b) Swapping

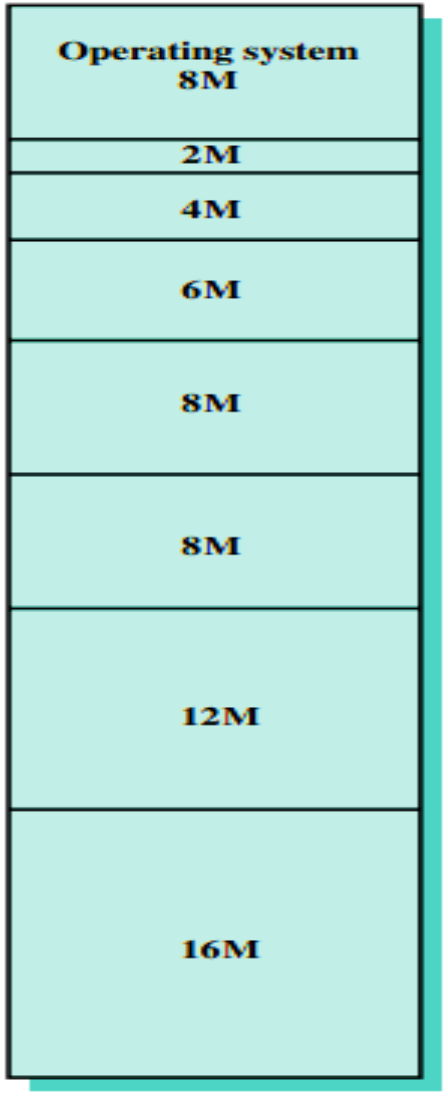
Partitioning

- In **multiprogramming**, to accommodate more than one user program, we need to **partition** the main memory
- Simplest way of partitioning – **Equal sized (fixed) partition**
- **Alternative way: Fixed unequal sized partition**
  - When a process is brought into memory, it will be placed in the smallest available partition to fit it
  - But it is also an **inefficient** partitioning – there will be still some **wasted** memory
    - Mostly, a process will not require exactly as much memory as provided by the partition

Partitioning – Fixed Partition



(a) Equal-size partitions

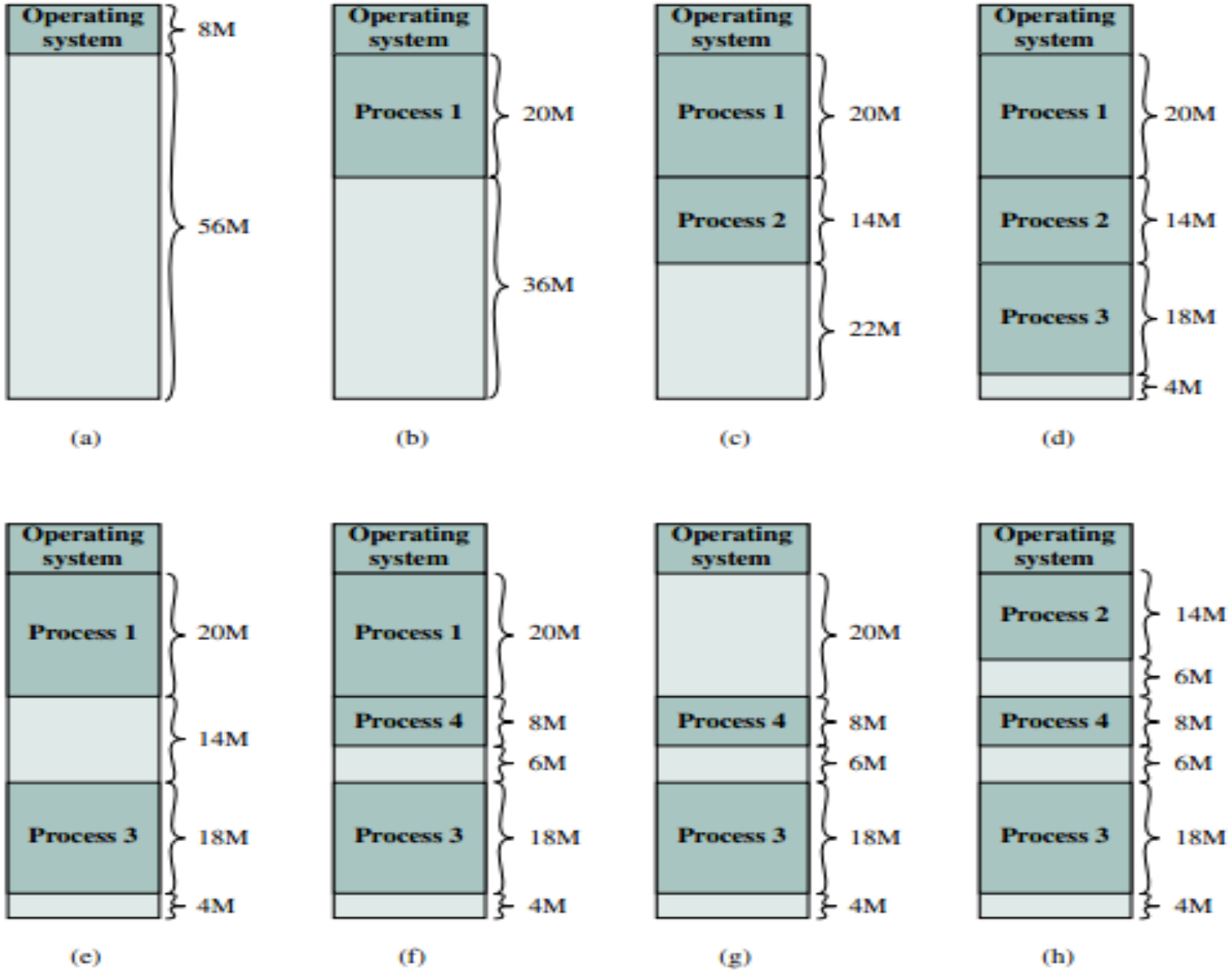


(b) Unequal-size partitions

More Efficient Partition

- A more efficient approach – **variable sized partition** or **dynamic partition**
  - When a process is brought to memory, it is allocated **exactly** as much memory as it requires; not more than that
- But though it starts well, eventually it leads to a situation for a memory containing **a lot of small holes**
  - As time goes on, memory is appeared as **more and more fragmented** to a new job
    - Free memory spaces are so much **scattered**
    - Utilization of memory management **degrades**
  - **Solution**: **COMPACTION**
    - From time to time, OS will shift all the processes to place them all together as well as all free memory space together in one block (**defragmentation**)
      - But it is very **time consuming -** wasteful of processor time

Dynamic or Variable Sized Partition



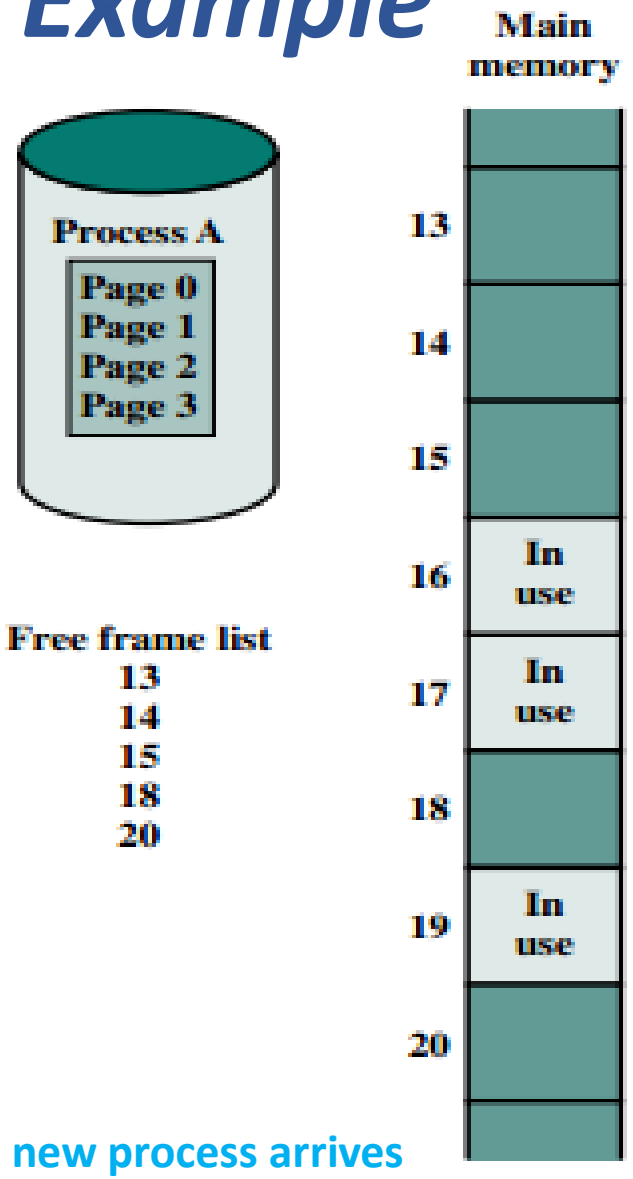
Logical to Physical Address Conversion

- It is clear from prior discussion that process is not likely to be loaded at the very same place of memory each time it is swapped in.
  - Moreover, following compaction, a process may still be shifted while it is already in main memory
  - So address of a process is always changing (not fixed) – addresses of its data and instructions too
    - **Which type of address is changing?** – logical or physical
  - So we will make a **distinction** between logical and physical address
    - A **logical address** is expressed as a location **relative** to the beginning of the program
      - So inside a program, all the data and instructions will hold logical address
    - A **physical address** is an **actual** location in main memory
      - When a processor executes a process, its logical address must be converted to physical one
        - A **hardware feature** is there monitored by OS
        - **Conversion: How? -** Logical address of each instruction (its relative address or offset) will be added with the physically starting address of the process (base address) to find out its original location in the main memory

Paging

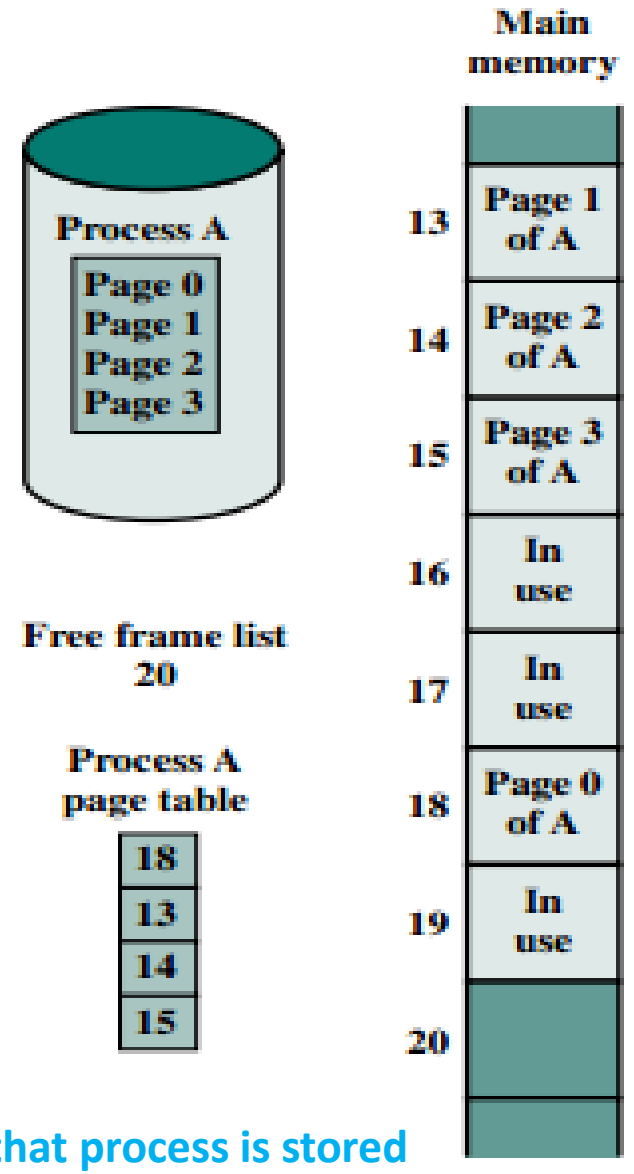
- As we have explored all types of partitions, we notice their drawbacks
- **Alternative:**
  - Memory is partitioned into equal and relatively small fixed sized chunks being same for all - Known as **pages**
    - These pages of a program could be assigned to available chunks of memory – known as **frames** or **page-frames** in main memory
      - **Frame size = Page size**
      - **Maximum** wasted space in memory for any process occurs in a fraction only at the last page
      - At a certain time, some of the frames in main memory are in use and some are free
        - **List** of free frames are maintained by OS
        - Pages of a certain program is loaded to these free frames by OS
        - Smaller processes requires fewer pages and larger processes requires more.

Paging – Example



When a new process arrives

(a) Before



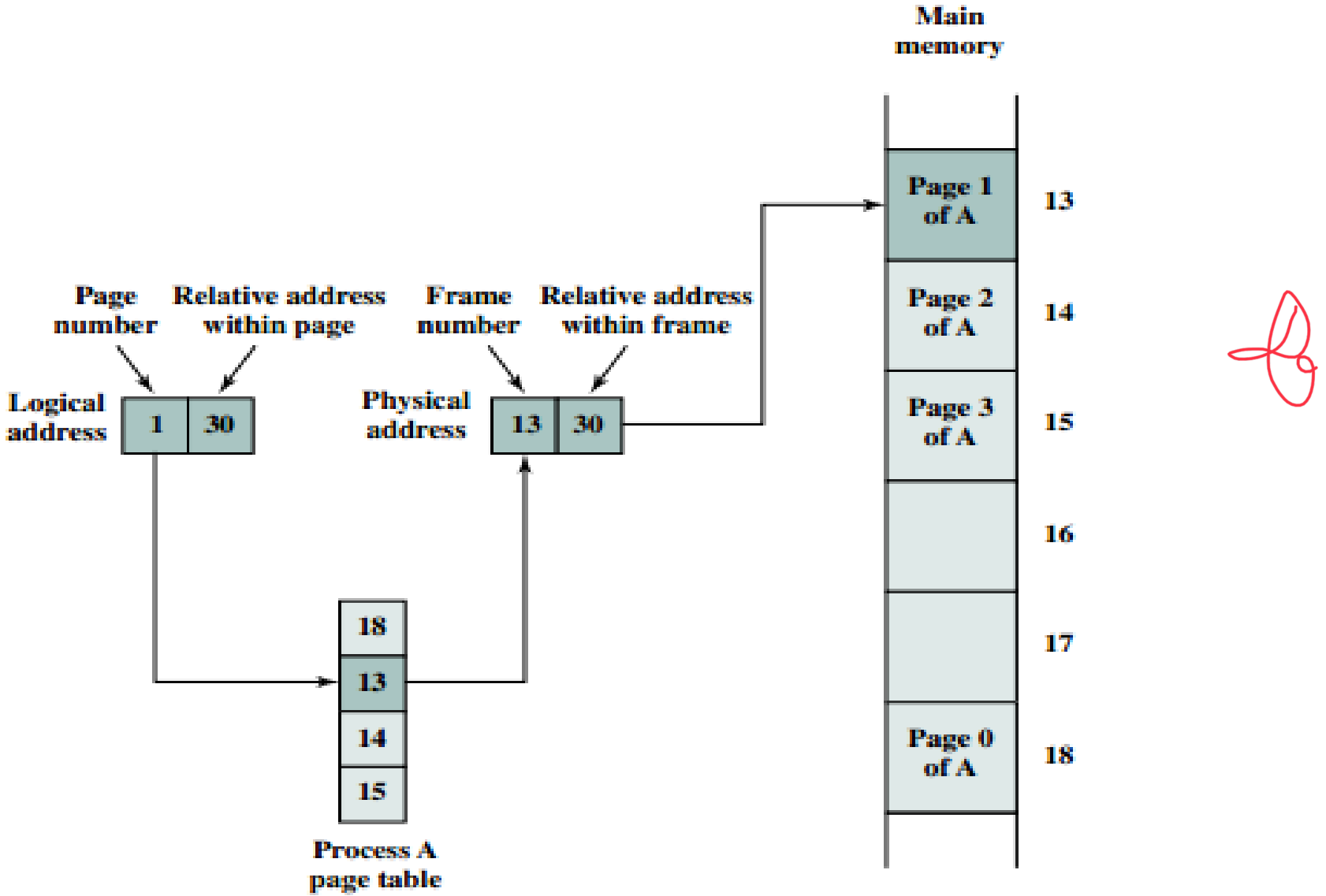
When that process is stored

(b) After

Page Table

- If there is not sufficient unused frames to hold a complete process, can we not load and execute that process?
- Again we have to consider the concept of **logical address** of a process
  - But here we will not use a single base (starting) address of a process (***i.e.*** logical address of that process)
    - Rather OS maintains a **page table** for each current process
      - This **page table** points the exact **frame locations** of each pages of the process in main memory
      - Within a program, each **logical address** of data or instruction consists of **page number** and a relative address within that page (**offset**)
      - Still this logical (page + offset) to physical (frame + offset) address conversion is done by processor hardware
        - Processor must know how to access the **page table** of the current process

Logical to Physical Address Translation



Virtual Memory & Demand Paging

- To achieve effective multiprogramming system, paging is used
  - Another important concept can be developed through breaking a process into pages – **virtual memory**
- **Virtual memory** is developed based on **demand paging**
  - Each page of a process is brought into the main memory only when is needed – on demand
  - **Example:** Suppose a large process consists of a long program (instructions) and a number of arrays of data. Usually at a time of execution, processor requires a small section of the program (*e.g.* subroutine) and one or two arrays of data based on the principle of locality. If we load the complete process in the memory, a bigger part of that process will be unused before it is suspended. For better use of memory, we will load a very few pages. Now if it requires another page that is not in memory, a **page fault** will be occurred. It results in:
    1. This event will ask the OS to bring the **desired** page
    2. But OS must be **clever** to bring a new page in; sometimes **replacing** an **old** page from memory
       - Known as **page replacement** – *e.g.* LRU algorithm
       - But it should avoid **thrashing** too

Virtual Memory – Its Consequence

- With demand paging, it is not necessary to load entire process into memory
  - It is possible for a process to be **larger** than main memory
  - Without demand paging, a programmer should be concern about the size of main memory
    - If it is larger than the main memory, the programmer should structure the process into pieces so that one gets access at a time
- With demand paging, **partitioning** a process is the job of OS and processor hardware
    - Now this entire process will be stored in disk storage – no restriction regarding **program size**
    - A process is executed when it is stored in the main memory (real memory but smaller)
      - But programmer could sense **even larger space** utilizing disk storage – this concept referred to virtual memory

Note: Very effective for multiprogramming

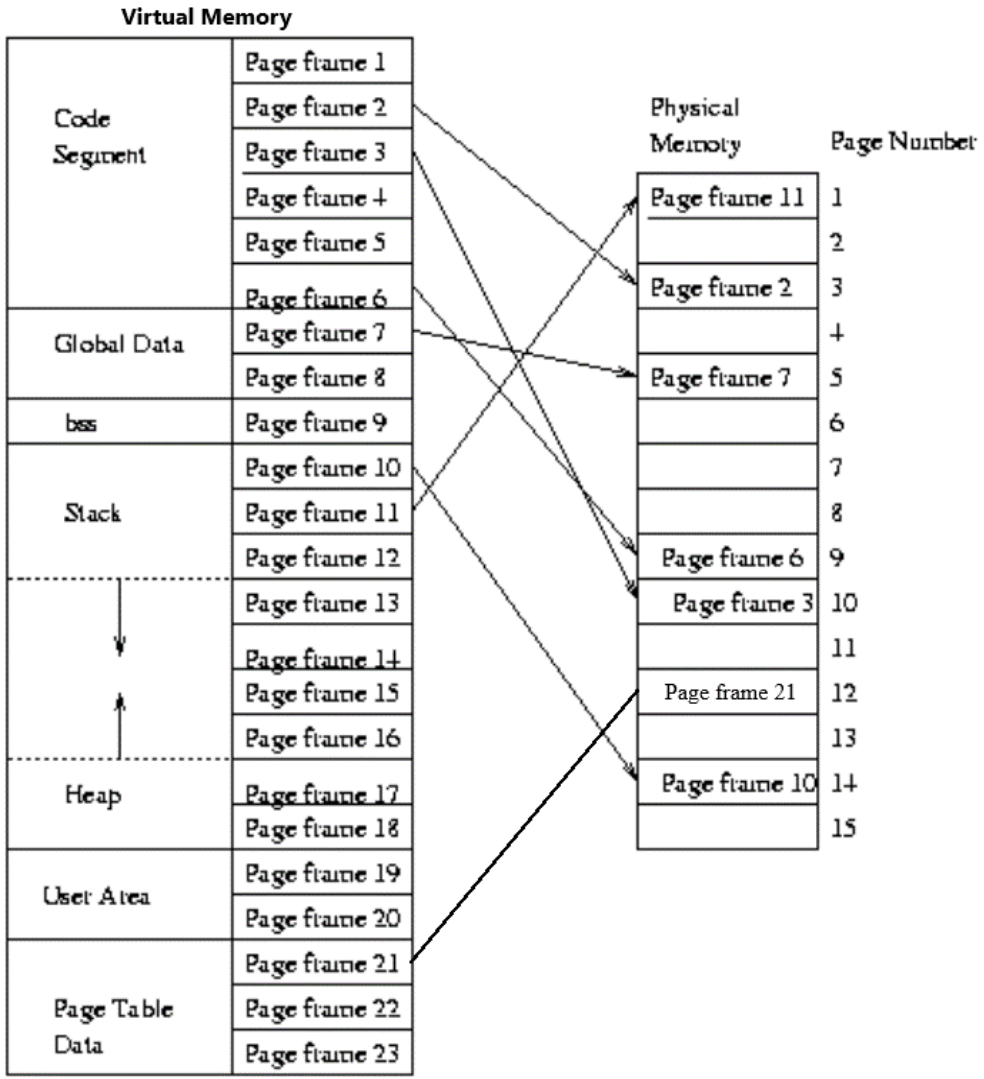
Page Table Structure

- To read a word from main memory, it requires to translate its virtual or logical address to physical one
  - Actually a page number and associated offset will be translated to corresponding frame number and associated offset with the help of page table
  - As the processes are of variable lengths, page table is also has different length
    - So page table can't be well suited in register. As a result:
      - Instead, a page table must be stored in main memory to be accessed
      - But starting of the page table can be saved in a register
  - Each process has one page table, but sometimes entries of this page table may be very large in number
    - Example: For a 2 GB process (2^{31} Bytes) in a virtual memory, using 512 Bytes (2^9 Bytes) page will require 2^{22} page table entries alone.

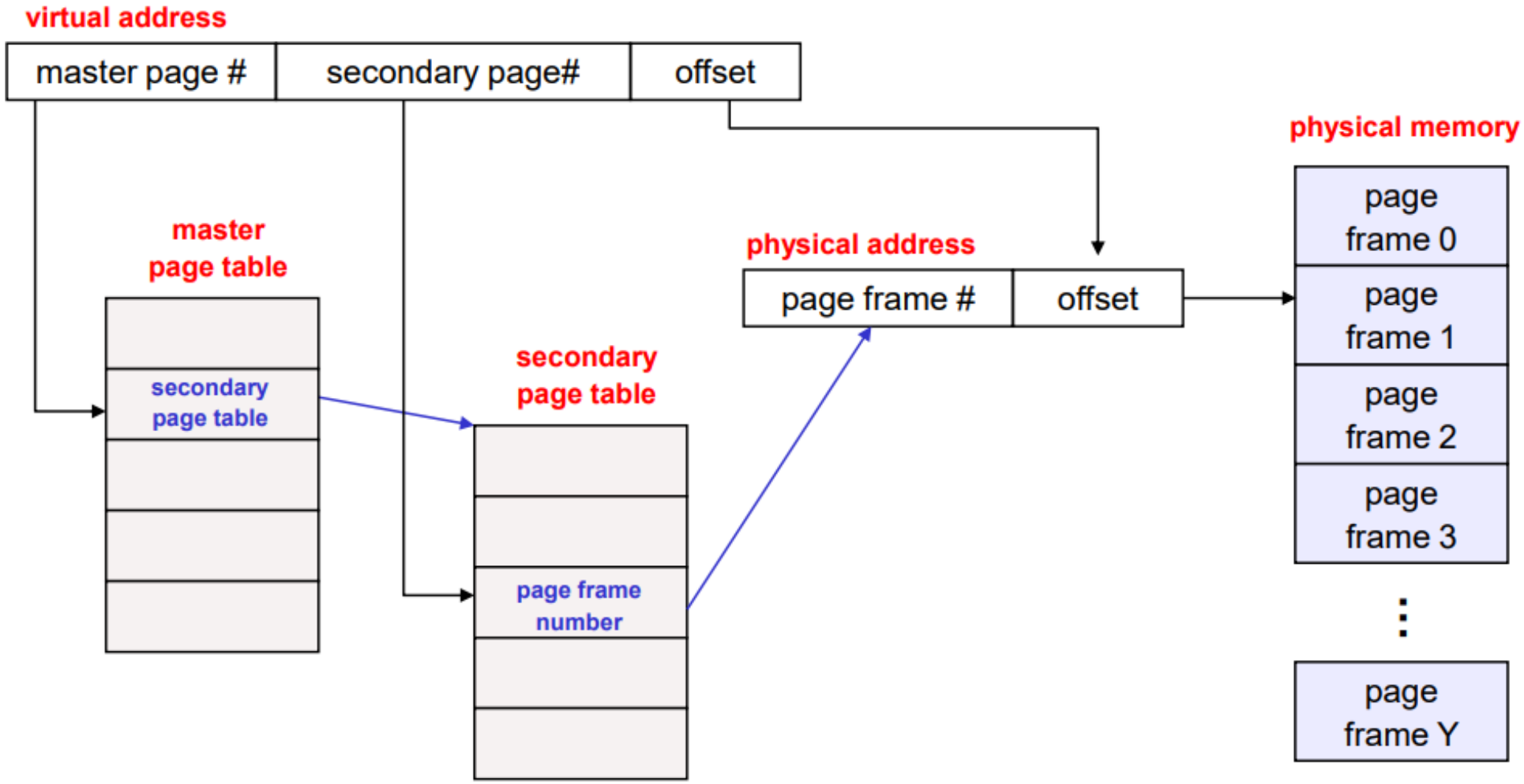
Page Table Structure Schemes

- **Scheme 1:**
  - Most of the cases, this page table is stored in **virtual memory** rather than main memory
    - Thus, page table itself is subject to **paging** like other processes
    - When the process is running, at least a part of the its page table (one or more pages of entire page table) must be present in main memory (**including currently executing page entry**)
- **Scheme 2:**
  - Sometimes **two-level** scheme is used to organize large page table
    - Here first level addressing is regarded as a **page directory** where each of its entries points to a **page table** (Second level)
      - This second level page table size is restricted to a page size
    - **Example:** If the length of page directory is ***X*** and each page table (second level) holds ***Y*** page entries, then maximum length of a process might be ***X * Y***

Scheme 1: Page Table Itself in Virtual Memory



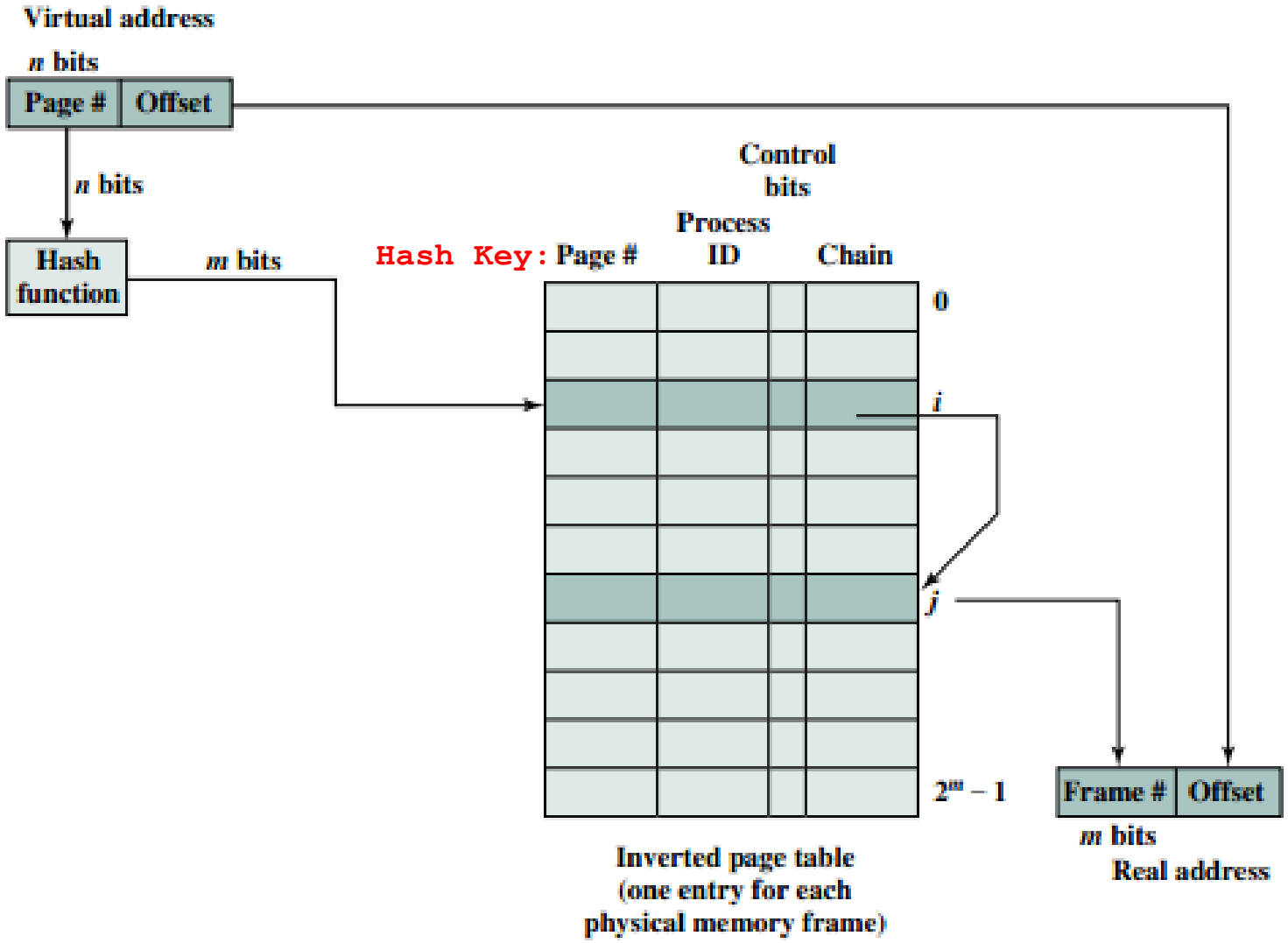
Scheme 2: Two-Level Page Table



Page Table Structure Schemes…

- **Scheme 3:**
  - Use of one- or two-level page tables through **inverted page table** structure
    - Here a **hash function** is used to map a **page entry** from its virtual address to its corresponding **hash value**
    - This **hash value** will point to an **entry** from **inverted page table**
      - Each entry of this table represents a main memory page frame rather than a virtual page – so named as **inverted page table**
      - As number of frames in a main memory is fixed, the size of inverted page table is also fixed
    - **But** more than one virtual page address may be mapped into same hash table entry (**complexity**)
      - A **chaining technique** is used to manage the **overflow**
      - **Note:** As this hash technique results in chains of one or two entries (very short), it will not degrade the performance

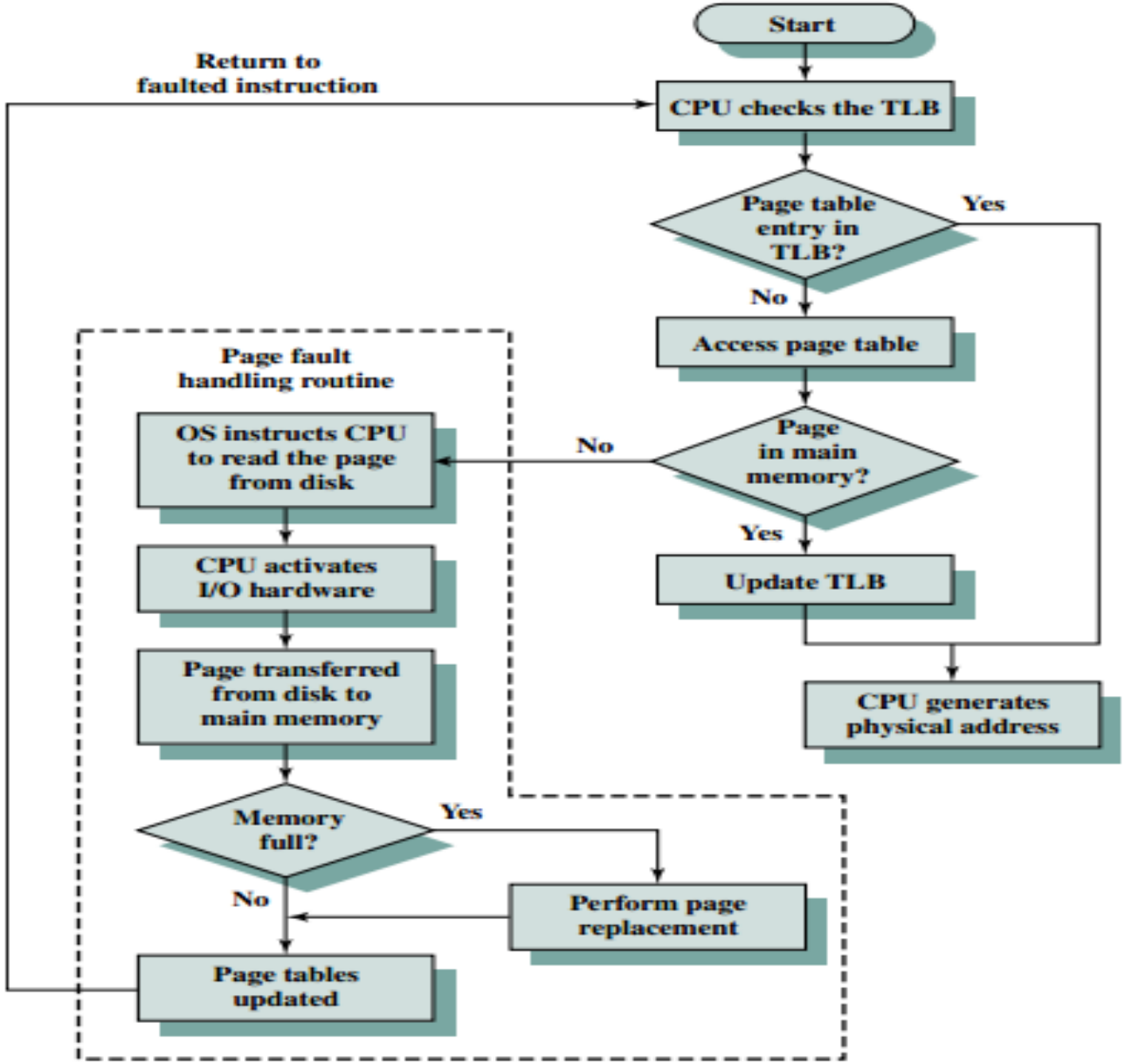
Scheme 3: Inverted Page Table



Translation Lookaside Buffer

- Each virtual memory reference two physical memory accesses (***limitation***):
    1. One to fetch the appropriate **page table entry**
    2. One to fetch the **desired data**
        - In fact **doubling** the memory access time
- To overcome this problem, most schemes use a **special cache** for *page table entries* – named as **Translation Lookaside Buffer**
    - Used same way as a memory cache
    - It contains **very recent** **page table entries**
        - Based on principle of locality, the most probable next virtual memory references will be at the location of recently used pages
        - Thus performance will be significantly improved

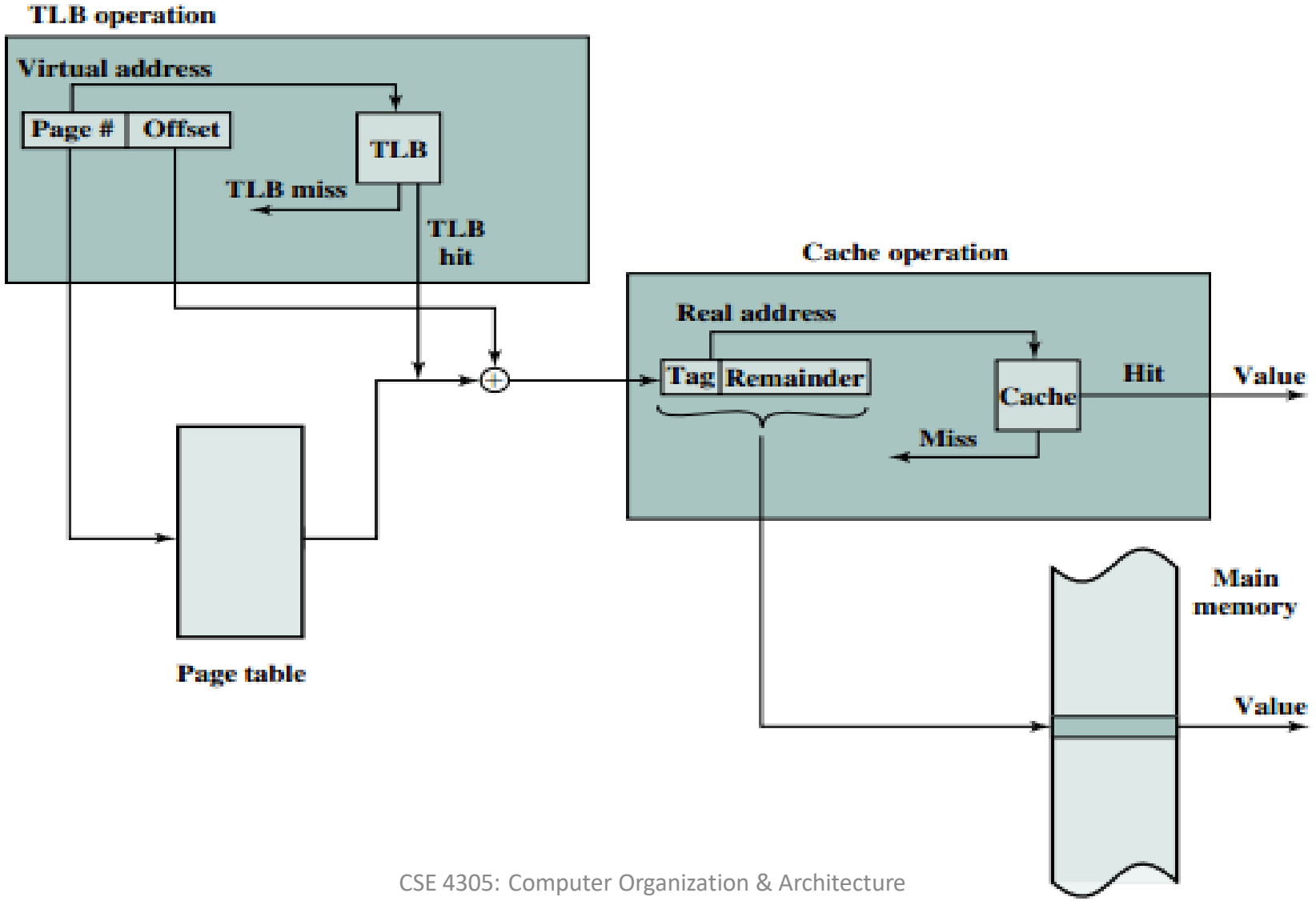
Operation of Paging with Help of TLB



Virtual Memory and Cache Memory

- Virtual memory must interact with **main memory cache** directly to retrieve the desired word for the processor faster
- **Steps:**
    1. The memory system consults with the TLB to if matched page table entry is present or not
        - If it is, then the real/physical address is generated combining corresponding frame number with offset
        - If it is not, this entry is acquired from the page table in main memory (if it is not in main memory, it should be loaded from the disk storage)
    2. If the physical address is generated, the cache is consulted to check if that desired word is present or not
        - If it is, then it is returned to the processor
        - If it is not, the word is retrieved from the main memory

Virtual Memory and Cache Memory...



Segmentation

- Another way to subdivide the main memory – **Segmentation**
  - **Paging**
    - **Invisible** to the programmer
    - **Advantage:** provide larger address space
  - **Segmentation**
    - **Visible** to programmer
    - **Advantage:** provide a mean so that programmer can easily organize a program and data with their associating privilege and protection attributes to use (**specialty**)

Segmentation of Memory

- Segmentation allows the programmer to view memory **as multiple address spaces** or **segments**
  - These segments are **variable in size** based on demand
  - Now the OS or programmer can **distribute** these program/instructions and data in different segments
  - One or more number of **program and data segments** can be under a certain program
  - Each segment has its own access **authorization** and **usage principle**
  - So now on, a memory references (address) is made of **segment number** and **offset** in it

Advantage of Segmentation

- This organization has a number of **advantages** to the **programmer** over non-segmented address space (e.g. paging):
  - It simplifies to manage the **growth** of data structure
    - Programmer need not pay any attention to fix the data structure size beforehand, rather OS can easily expand or shrink its size based on demand
  - Modification or recompilation of any segment **independently** is possible
    - It need not to relink or reload entire set of programs to make any change
  - These segments can be **shared** among other processes
    - It only need place a utility program or useful data in a segment so that they can be easily accessed by others
  - It can provide **protection** to the segments to access it in a convenient fashion.