

# Basics of Operating system

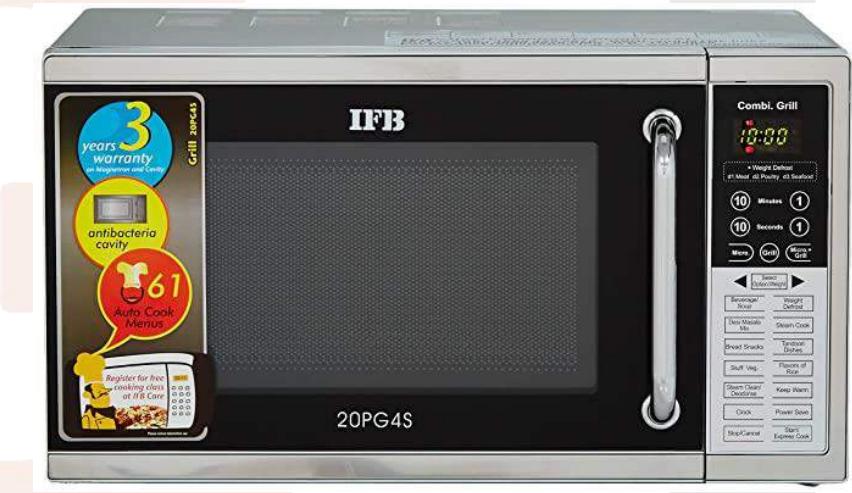
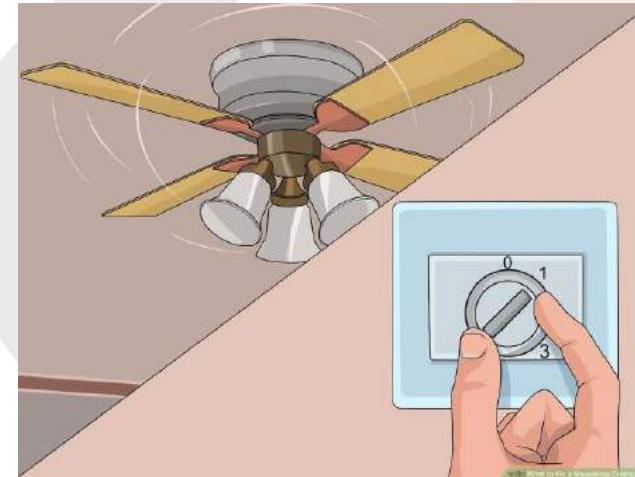
1. What is Operating System
2. Abstract view of Operating System
3. Goals and functions of Operating System
4. Evolution of Operating System
5. Multiprogramming Operating System
6. Multitasking/Time Sharing Operating System
7. Multiprocessing Processing Operating System
8. Real Time Operating System
9. Distributed Operating System
10. CLI Vs GUI
11. Structure of Operating System
12. Micro-Kernel approach
13. System call



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# What is Operating System

- Whatever used as an interface between the user and the core machine is OS.
  - E.g. steering of car, switch of the fan etc., Buttons over electronic devices.



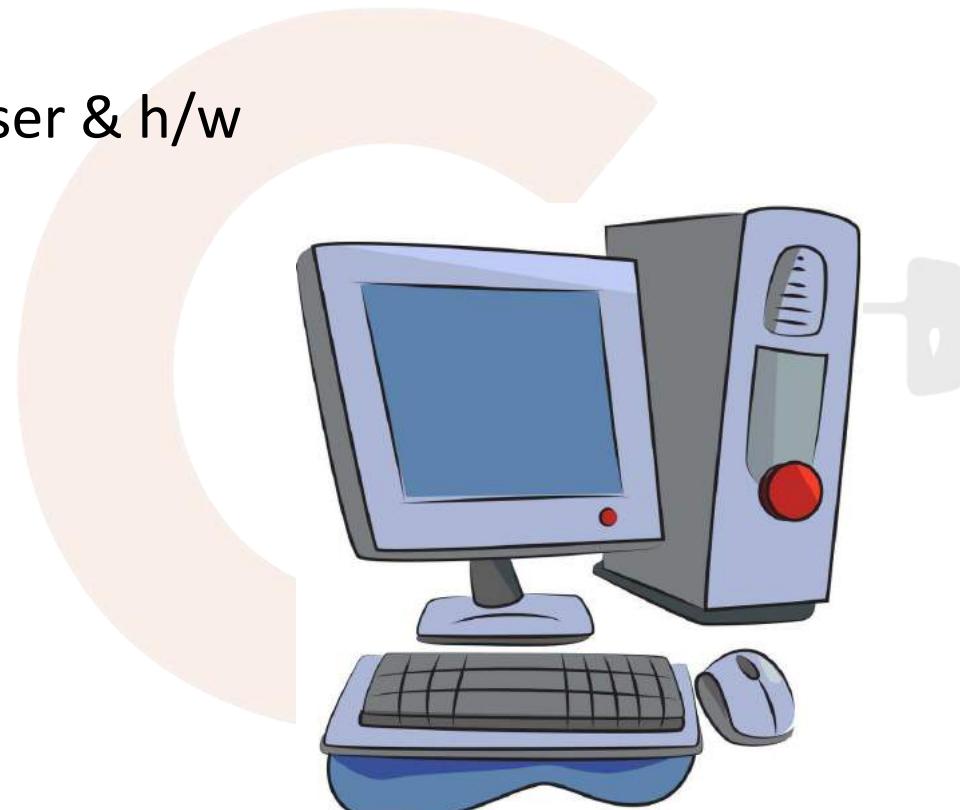
- Question comes why we need an operating system?
  - To enable everybody to use h/w In a convenient and efficient manner

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Definition of Operating System

- There is no exact or precise definition for OS but we can say, “A program or System software”
  - Which Acts as an **intermediary** between user & h/w



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

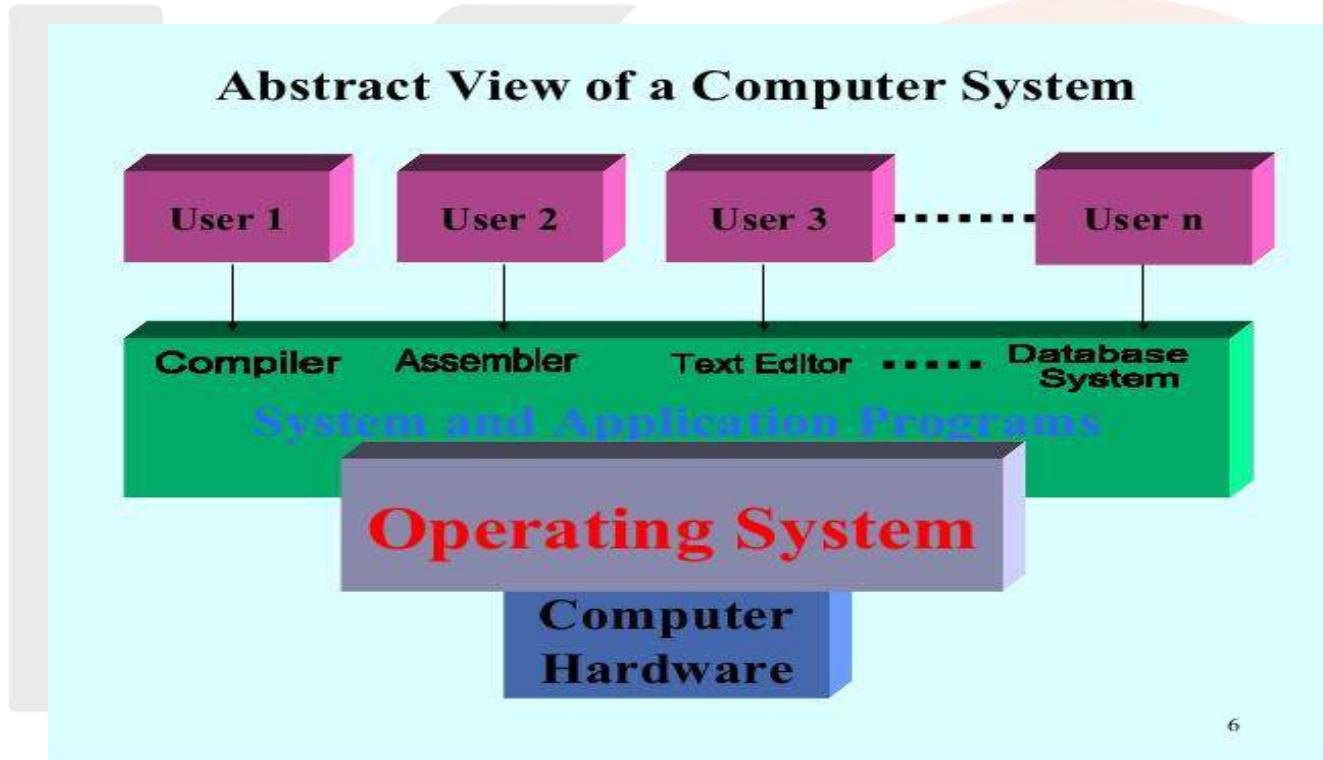
- **Resource Manager/Allocator** - Manage system resources in an unbiased fashion both h/w (mainly CPU time, memory, system buses) & s/w (access, authorization, semaphores) and provide functionality to application programs.
- OS controls and coordinates the use of resources among various application programs.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- OS provides platform on which other application programs can be installed, provides the environment within which programs are executed.



6



ios



# Operating Systems

Ubuntu

Mac

Windows

Android

ios

Linux

Tizen

Debian

Chrome

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- For personal computers - Microsoft windows (82.7%), Mac (13.23), Linux (1.57)
- For mobile phones - Android (87.5), IOS (12.1) etc.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q An operating system is: (NET-DEC-2006)

- (A) Collection of hardware components
- (B) Collection of input-output devices
- (C) Collection of software routines
- (D) All the above



To access all paid content get **KG Prime** at ₹25/day

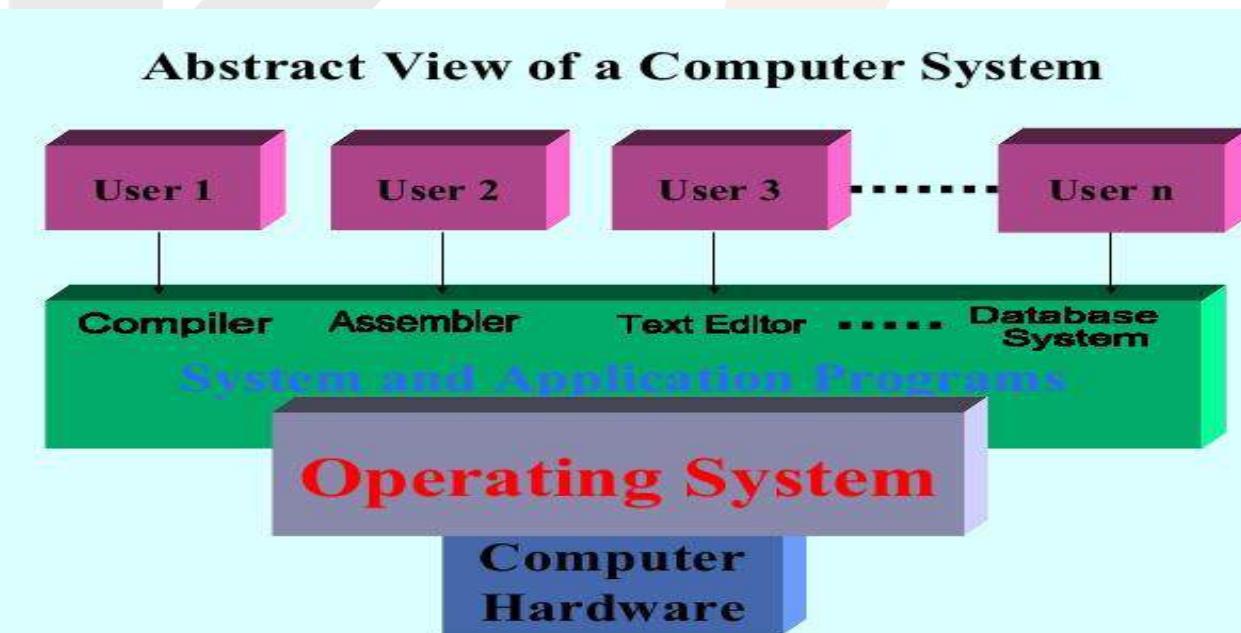
**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Computer hardware** – CPU, memory units, i/o devices, system bus, registers etc. provides the basic computing resources.
- **OS** - Control and coordinates the use of the hardware among the various applications programs.
- **System and Applications programs** - Defines the way in which these resources are used to solve the computing problems of the user.
- **User**





# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Goals and Functions of operating system

- Goals are the ultimate destination, but we follow functions to implement goals.
- Goals
  - Primary goals (Convenience / user friendly)
  - Secondary goals (Efficiency (Using resources in efficient manner) / Reliability / maintainability)





सत्यमेव जयते

**Government Of India**

सबका साथ सबका विकास

Ministries	Departments
58	93

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Functions of operating system

- Process management
- Memory management
- I\O device management
- File management
- Network management
- Security & Protection



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



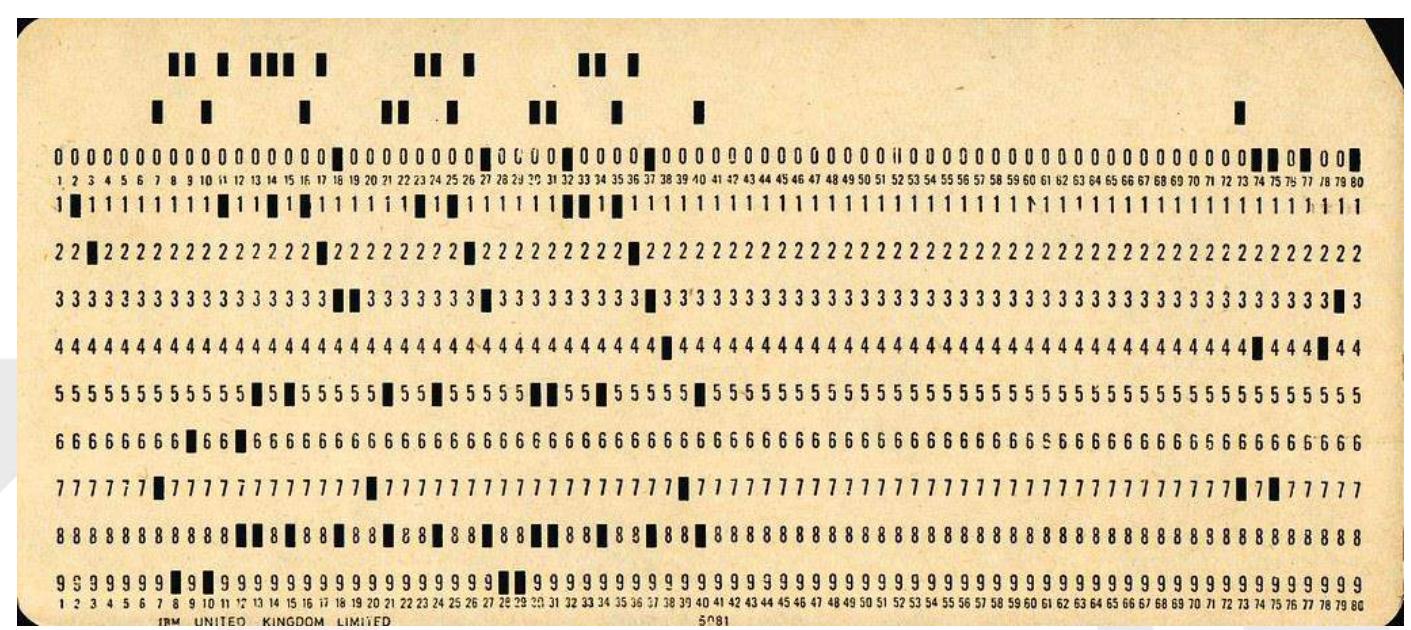
# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Evolution of Operating System

- Early computers were not interactive device, there user use to prepare a job which consist three parts
  - Program
  - Control information
  - Input data
- Only one job is given input at a time as there was no memory, computer will take the input then process it and then generate output.
- Common input/output device were punch card or tape drives.
- So these devices were very slow, and processor remain idle most of the time.

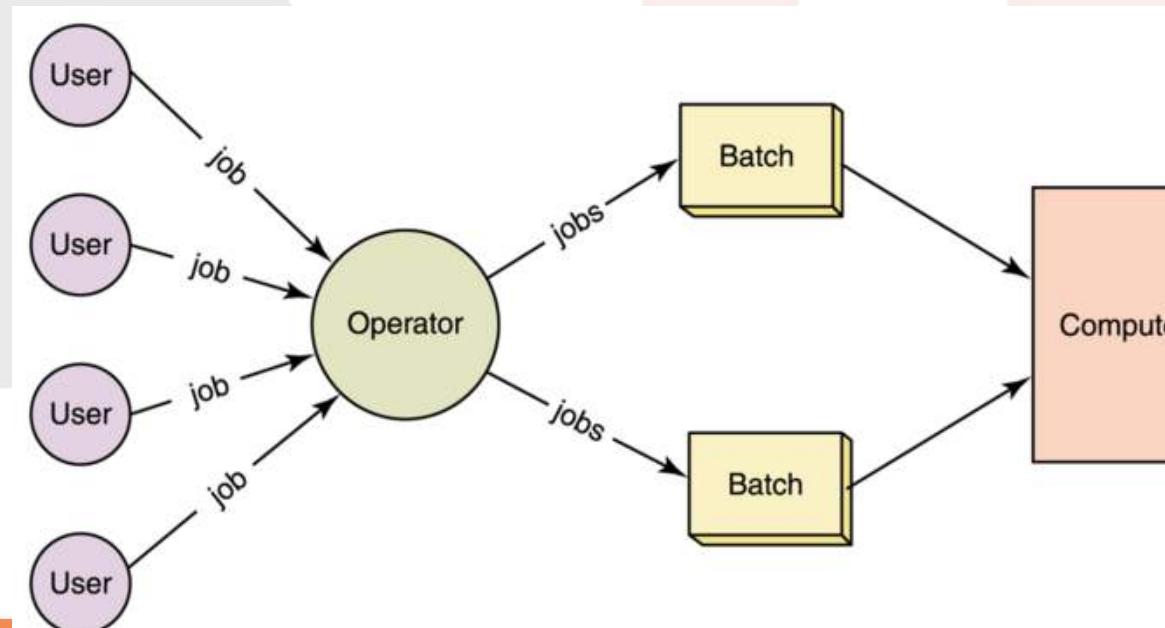
## Punch Card in Punch Card Machine



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Batch Operating System

- To speed up the processing job with similar types (programming language) were batched together and were run through the processor as a group (batch).
- In some system grouping is done by the operator while in some systems it is performed by the 'Batch Monitor' resided in the low end of main memory)
- Then jobs (as a deck of punched cards) are bundled into batches with similar requirement.
- Then the submitted jobs were 'grouped as FORTRAN jobs, COBOL jobs etc.



To access all paid content get **KG Prime** at ₹25/day

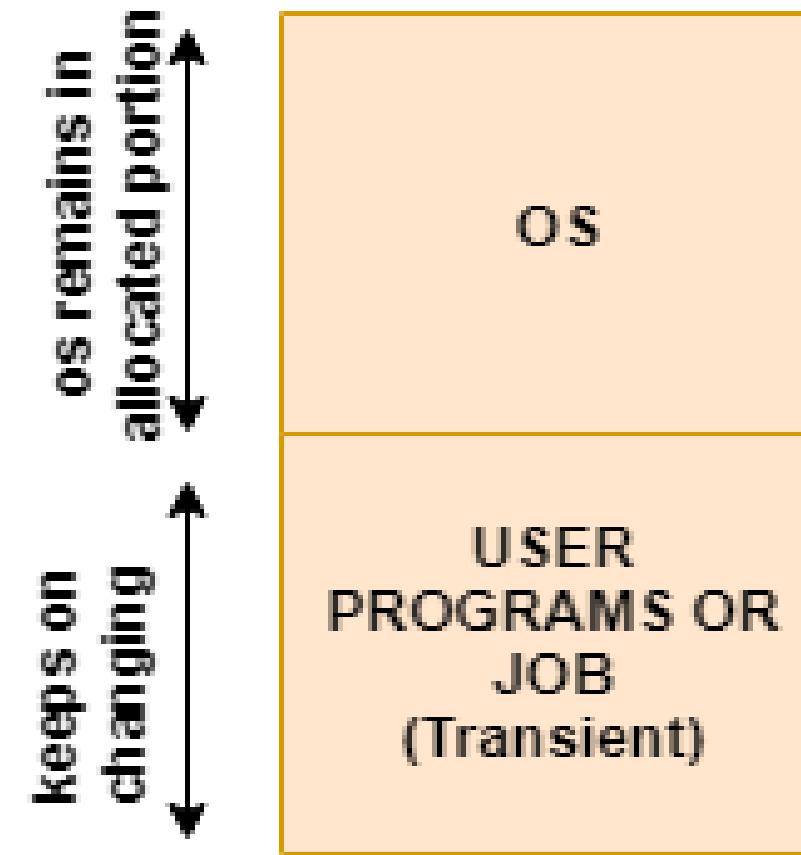
[CLICK HERE ↗](#)

## Advantage

- The batched jobs were executed automatically one after another saving its time by performing the activities (like loading of compiler) only for once. It resulted in improved system utilization due to reduced turnaround time.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention

## Disadvantage

- Memory limitation – memory was very limited, because of which interactive process or multiprogramming was not possible



To access all paid content get **KG Prime** at ₹25/day

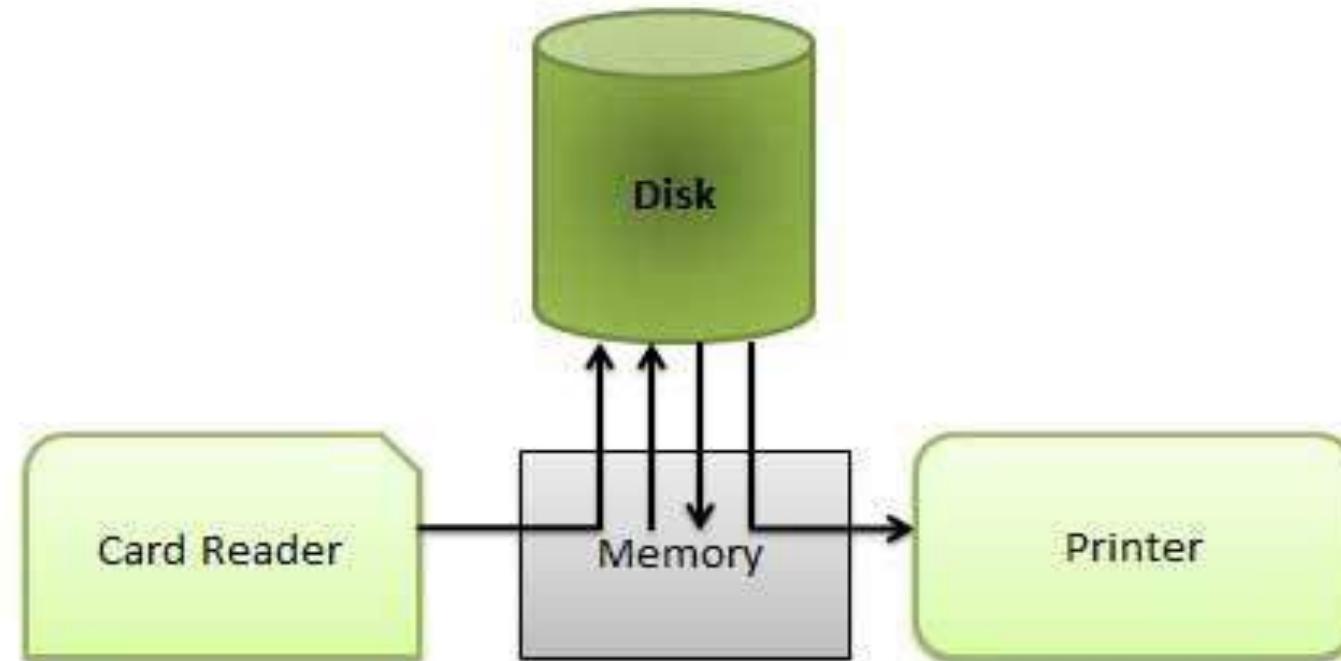
[CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Spooling

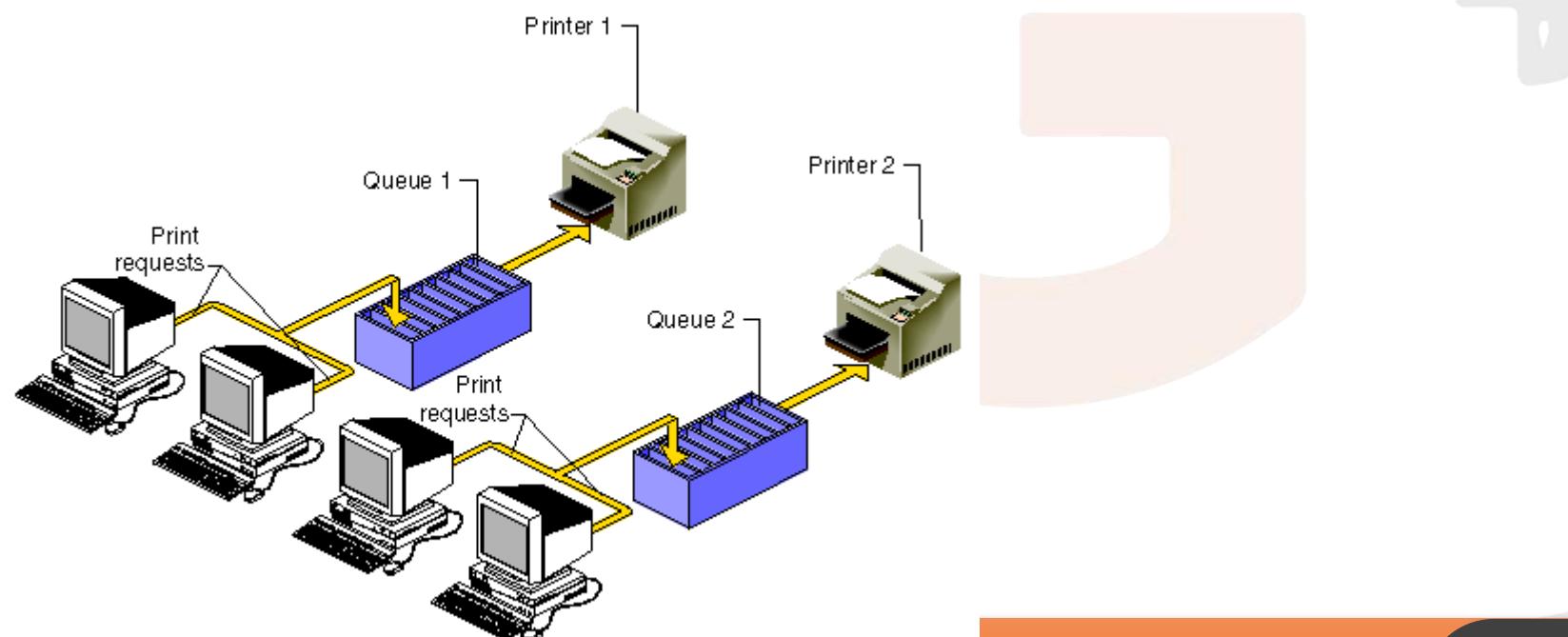
- ("Spool" is technically an acronym for simultaneous peripheral operations online.)
- In a computer system peripheral equipment, such as printers and punch card readers etc, are very slow relative to the performance of the rest of the system. Spooling is useful because devices access data at different rates.
- Spooling is a process in which data is temporarily held to be used and executed by a device, program or the system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.
- Generally, the spool is maintained on the computer's physical memory, buffers or the I/O device-specific interrupts.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- The most common implementation of spooling can be found in typical input/output devices such as the keyboard, mouse and printer. For example, in printer spooling, the documents/files that are sent to the printer are first stored in the memory. Once the printer is ready, it fetches the data and prints it.
- A spooler works by intercepting the information going to the printer, parking it temporarily on disk or in memory. The computer can send the document information to the spooler at full speed, then immediately return control of the screen to you.
- The spooler, meanwhile, hangs onto the information and feeds it to the printer at the slow speed the printer needs to get it. So if your computer can **spool**, you can work while a document is being printed.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Even experienced a situation when suddenly for some seconds your mouse or keyboard stops working? Meanwhile, we usually click again and again here and there on the screen to check if its working or not. When it actually starts working, what and wherever we pressed during its hang state gets executed very fast because all the instructions got stored in the respective device's spool.
- Spooling is capable of overlapping I/O operation for one job with processor operations for another job. i.e. multiple processes can write documents to a print queue without waiting and resume with their work.

**Q Which of the following is an example of a spooled device? (GATE-1996) (1 Marks)**

- (A) a line printer used to print the output of a number of jobs**
- (B) a terminal used to enter input data to a running program**
- (C) a secondary storage device in a virtual memory system**
- (D) a graphic display device**

**Q which of the following is an example of a spooled device? (GATE-1998) (1 Marks)**

- (a) The terminal used to enter the input data for the C program being executed**
- (b) An output device used to print the output of a number of jobs**
- (c) The secondary memory device in a virtual storage system**
- (d) The swapping area on a disk used by the swapper**

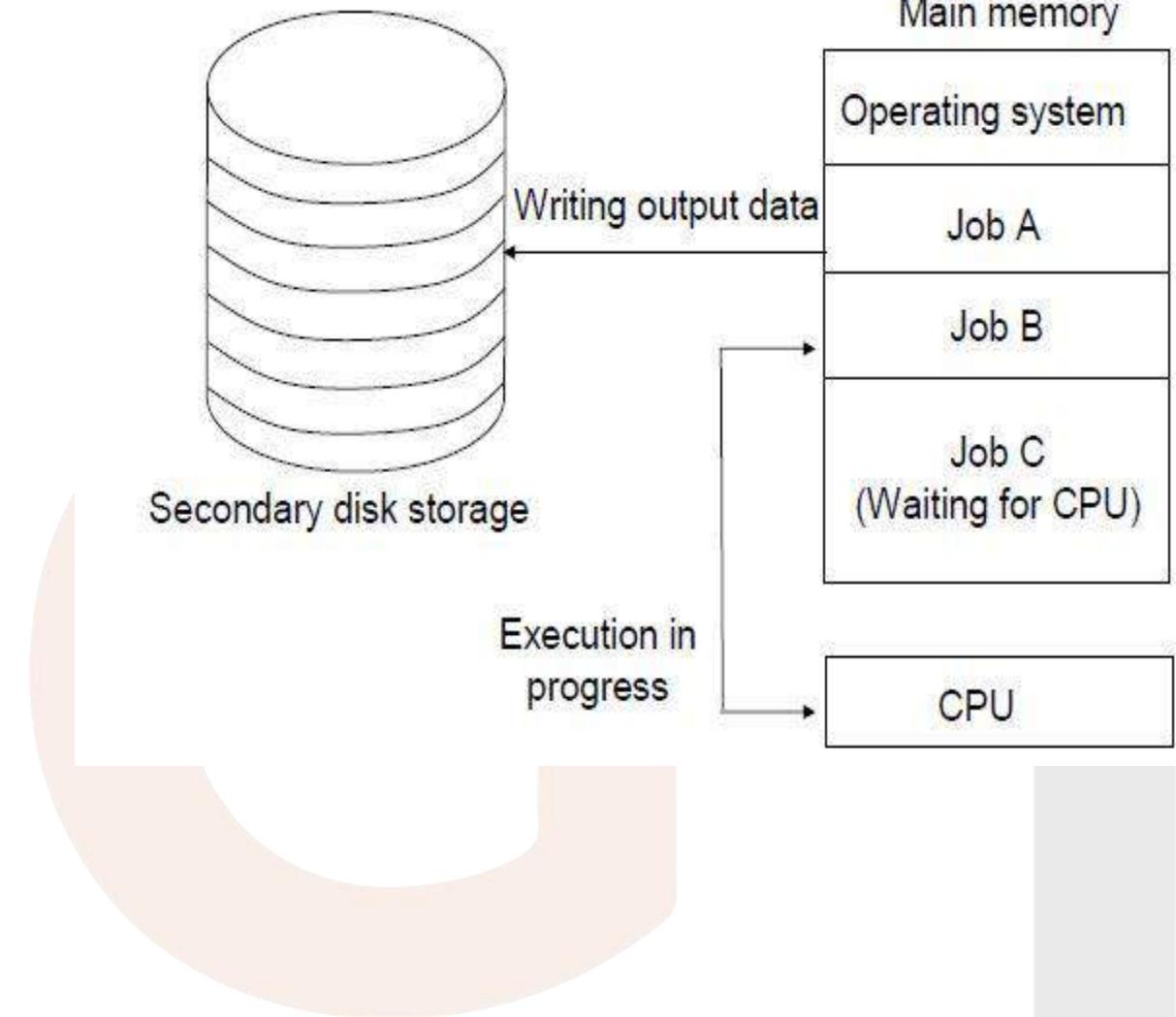


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Multiprogramming Operating System

- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. The basic idea of multiprogramming operating system is it keeps several jobs in main memory simultaneously.
- The jobs are kept initially on the disk in the **job pool**. This pool consists of all processes residing on disk awaiting allocation of main memory.
- The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete.

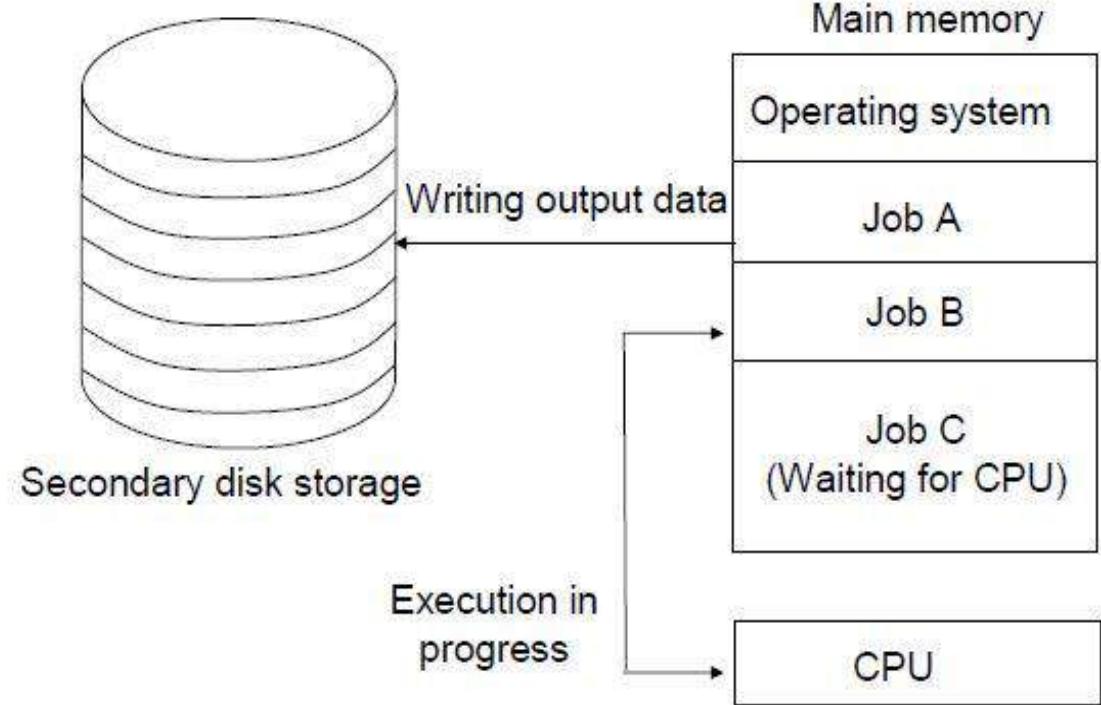


*Show must go on*

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- In a non-multi-programmed system, the CPU would sit idle and wait but in a multi-programmed system, the operating system simply switches to, and executes, another job. When **that** job needs to wait the CPU switches to **another** job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. So, conclusion is as long as at least one job needs to execute, the CPU is never idle.



Processor किसी के लिए wait नहीं करेगा

## Advantage

- High and efficient CPU utilization.
- Less response time or waiting time or turnaround time
- In most of the applications multiple tasks are running and multiprogramming systems better handle these type of applications
- Several processes share CPU time

## Disadvantage

- It is difficult to program a system because of complicated schedule handling.
- To accommodate many jobs in main memory, complex memory management is required.

**Q Which of the following features will characterize an OS as multi-programmed OS? (NET-DEC-2012) (Gate-2002) (1 Marks)**

- (a) More than one program may be loaded into main memory at the same time.
  - (b) If a program waits for certain event another program is immediately scheduled.
  - (c) If the execution of a program terminates, another program is immediately scheduled.
- (A) (a) only
- (B) (a) and (b) only
- (C) (a) and (c) only
- (D) (a), (b) and (c) only

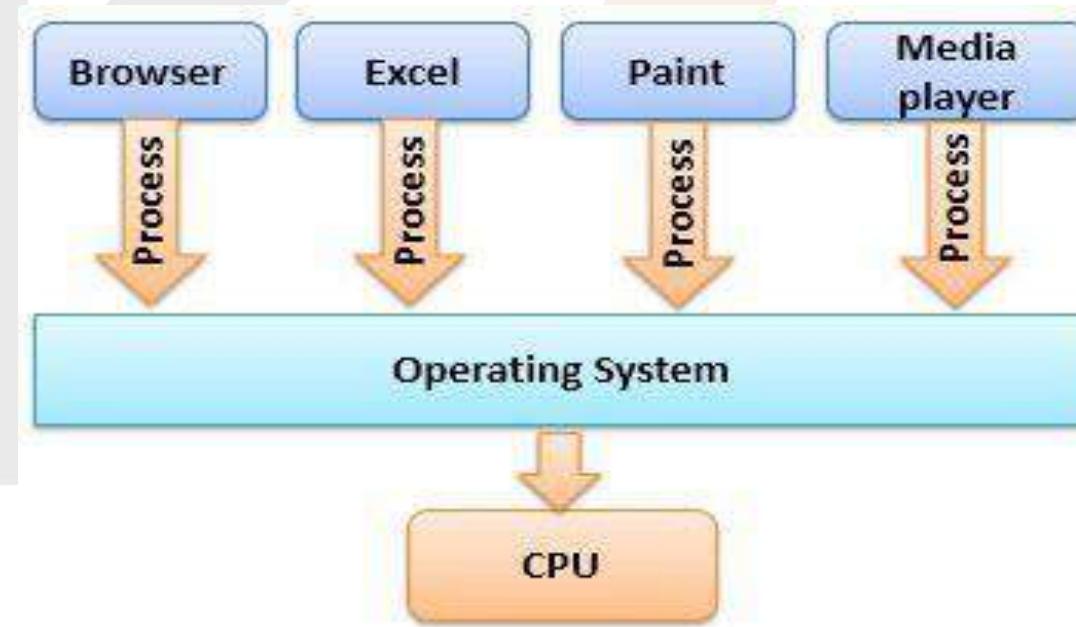


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Multitasking Operating system/time sharing/Multiprogramming with Round Robin/ Fair Share

- In the modern operating systems, we are able to play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all running at the same time. (by context switching, the illusion of parallelism is achieved)
- For multitasking to take place, firstly there should be multiprogramming i.e. presence of multiple programs ready for execution. And secondly the concept of time sharing.



Multitasking

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Time sharing (or multitasking) is a logical extension of multiprogramming, it allows many users to share the computer simultaneously. the CPU executes multiple jobs (May belong to different user) by switching among them, but the switches occur so frequently that, each user is given the impression that the entire computer system is dedicated to his/her use, even though it is being shared among many users.

**Q** If you want to execute more than one program at a time, the systems software that are used must be capable of: **(NET-DEC-2005)**

**(A)** word processing

**(C)** compiling

**(B)** virtual memory

**(D)** multitasking



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

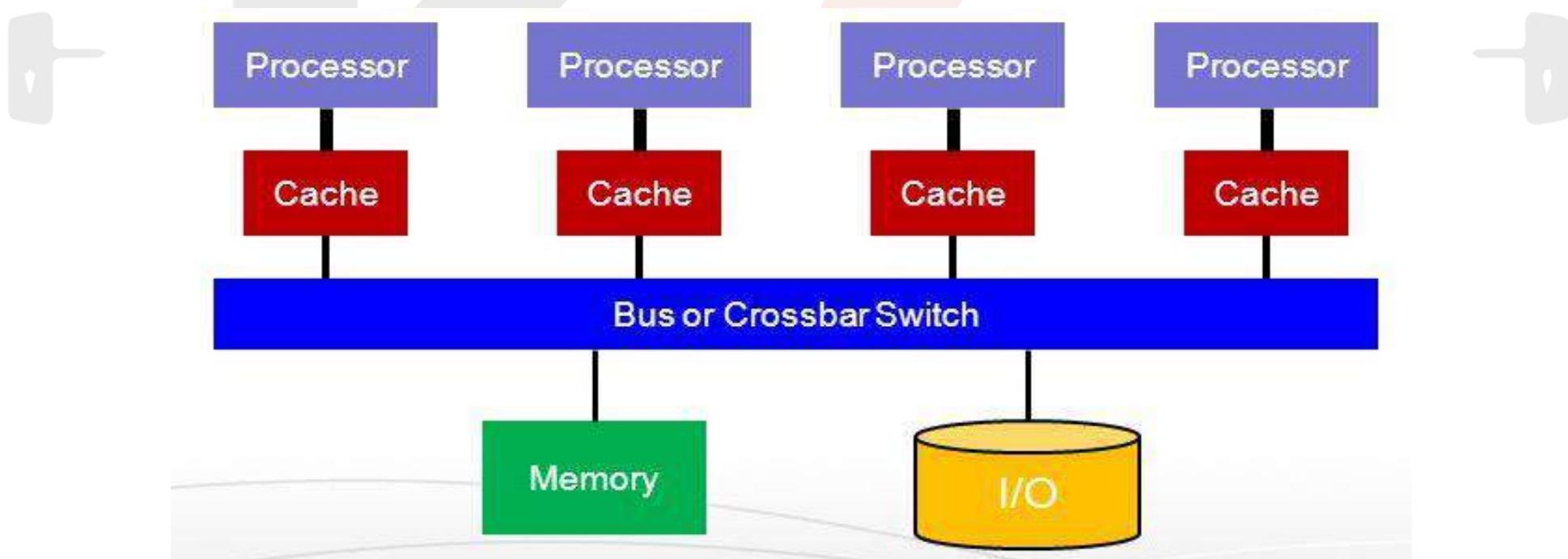


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Multiprocessing Operating System/ tightly coupled system

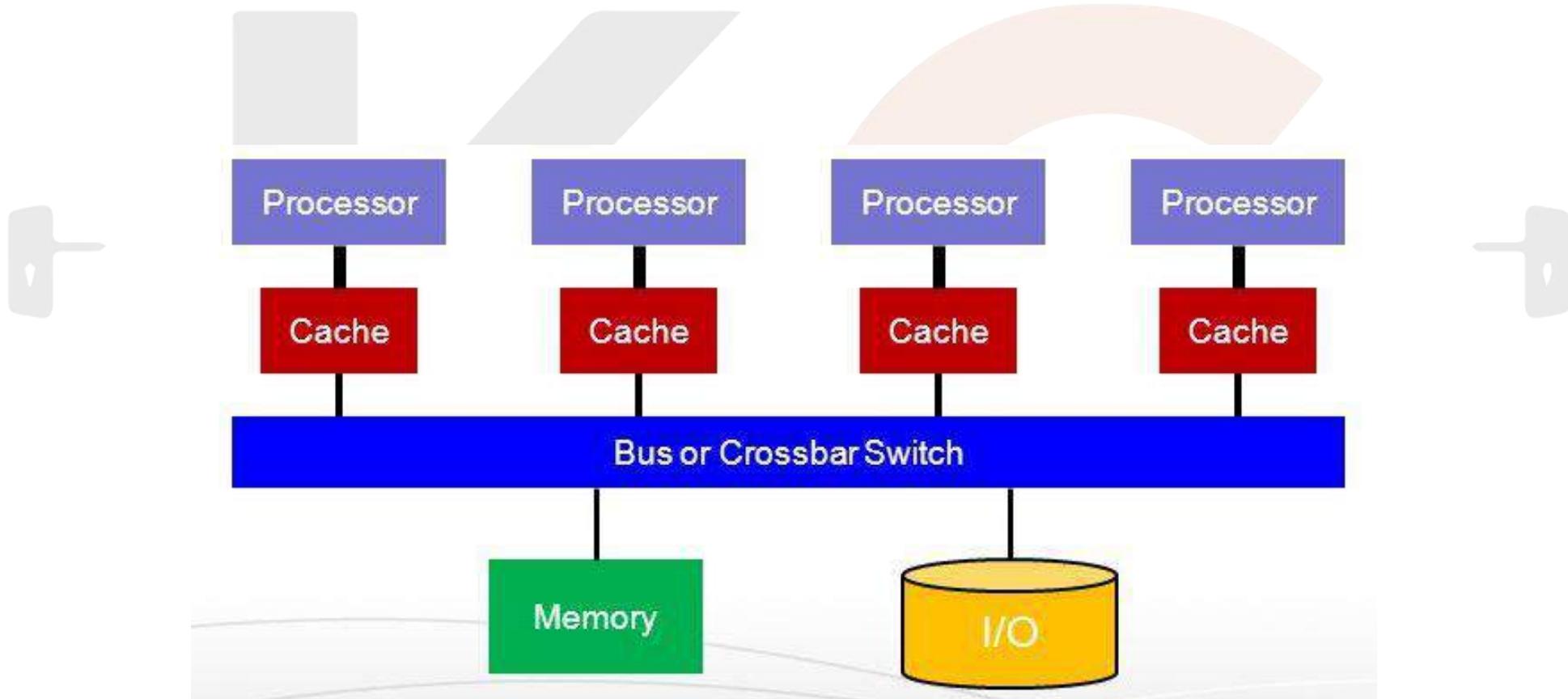
- Multiprocessor Operating System refers to the use of two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral devices. These systems are referred as tightly coupled systems
- So in multiprocessing, multiple concurrent processes each can run on a separate CPU, here we achieve a true parallel execution of processes. Parallel processing is the ability of the CPU to simultaneously process incoming jobs.



To access all paid content get **KG Prime** at ₹25/day

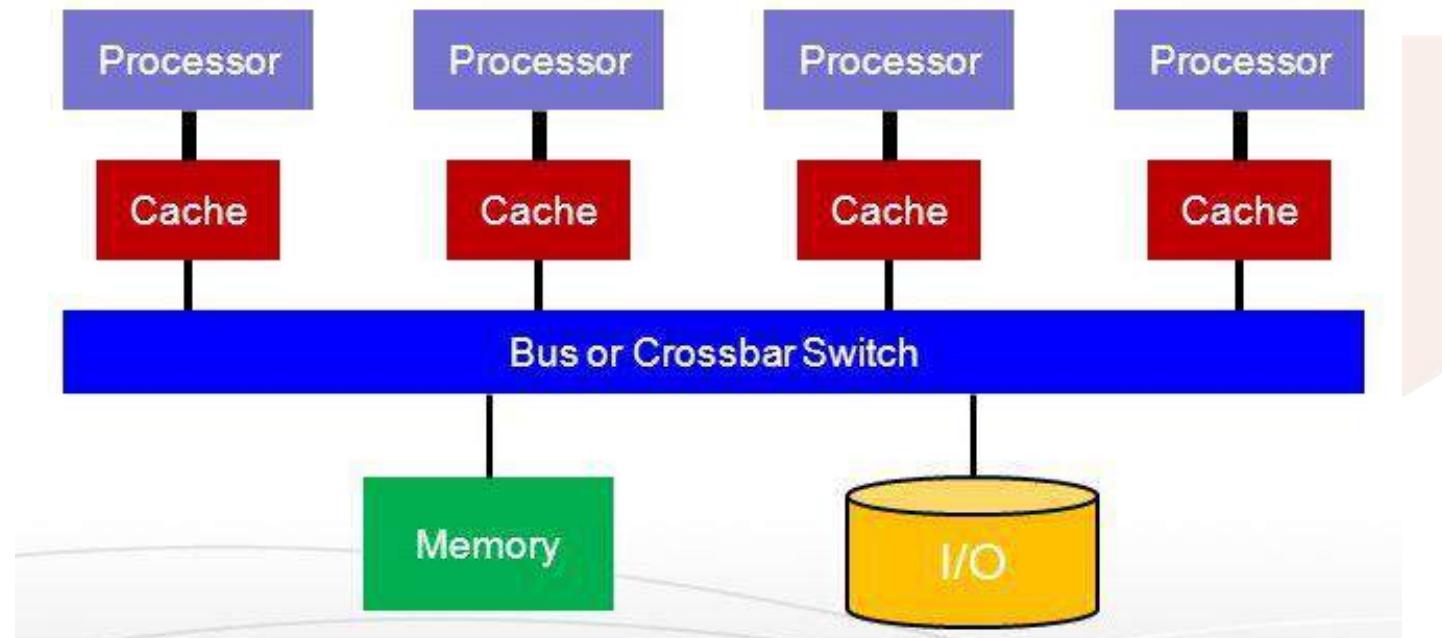
**CLICK HERE ↗**

- Multiprocessing becomes most important in computer system, where the complexity of the job is more, and CPU divides and conquers the jobs. Generally, the parallel processing is used in the fields like artificial intelligence and expert system, image processing, weather forecasting etc.



## Multiprocessing can be of two types

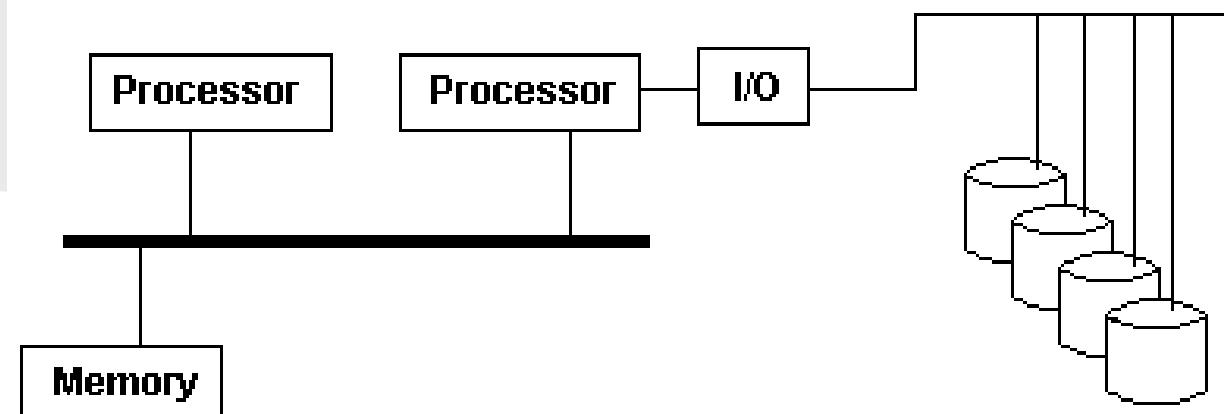
- **Symmetric multiprocessing** - In a symmetric multi-processing, a single OS instance controls two or more identical processors connected to a single shared main memory.
- Most of the multi-processing PC motherboards utilize symmetric multiprocessing. Here each processor runs an identical copy of operating system and these copies communicate with each other.
- SMP means that all processors are peers; no boss–worker relationship exists between processors. Windows, Mac OSX and Linux.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- **Asymmetric** - This scheme defines a master-slave relationship, where one processor behaves as a master and control other processor which behaves as slaves.
- For e.g. It may require that only one particular CPU respond to all hardware interrupts, whereas all other work in the system may be distributed equally among CPUs;
- Or execution of kernel-mode code may be restricted to only one particular CPU, whereas user-mode code may be executed in any combination of processors.
- Multiprocessing systems are often easier to design if such restrictions are imposed, but they tend to be less efficient than systems in which all CPUs are utilized.
- A boss processor controls the system, other processors either look to the boss for instruction or have predefined tasks. This scheme defines a boss-worker relationship. The boss processor schedules and allocates work to the worker processors.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

# Advantage of multiprocessing

- **Increased Throughput** - By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with  $N$  processors is not  $N$ , however; rather, it is less than  $N$ . When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. These types of systems are used when very high speed is required to process a large volume of data.
- **Economy of Scale** - Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.
- **Increased Reliability (fault tolerance)** - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fail, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether
- **Less Battery consumption and heat generation**

## Disadvantages

- It's more complex than simple operating systems.
- It requires context switching which may impacts performance.
- If one processor fails then it will affect in the speed
- large main memory required



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Real time Operating system

- A real time system is a time bound system which has well defined fixed time constraints. Processing must be done within the defined constraints or the system will fail. Very fast and quick respondent systems.
- RTOS intended to serve real-time applications that process data as it comes in, typically without buffer delays. Are used in an environment where a large number of events (generally external) must be accepted and processed in a short time.
- For example, a measurement from a petroleum refinery indicating that temperature is getting too high and might demand for immediate attention to avoid an explosion. Airlines reservation system, Air traffic control system, Systems that provide immediate updating, Systems that provide up to the minute information on stock prices, Defense application systems like as RADAR.



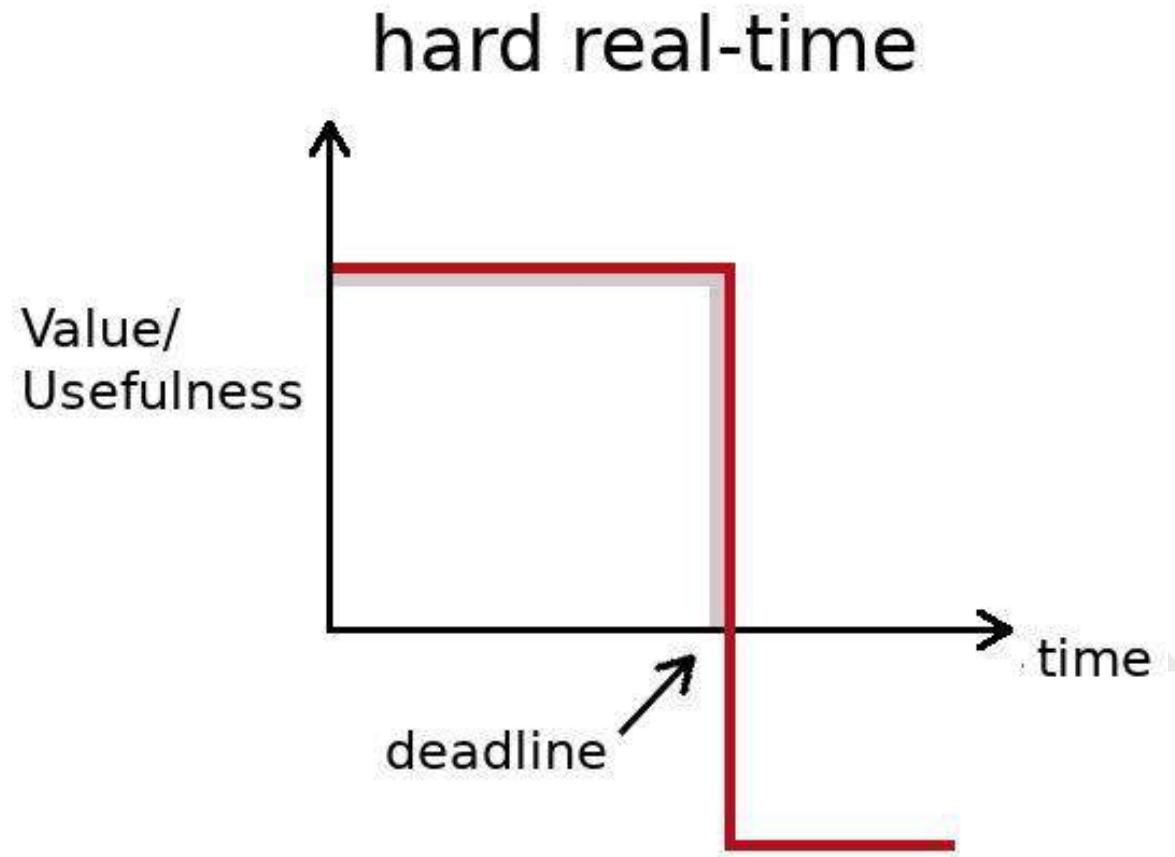
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.
- Little swapping of programs between primary and secondary memory. Most of the time, processes remain in primary memory in order to provide quick response, therefore, memory management in real time system is less demanding compared to other systems.



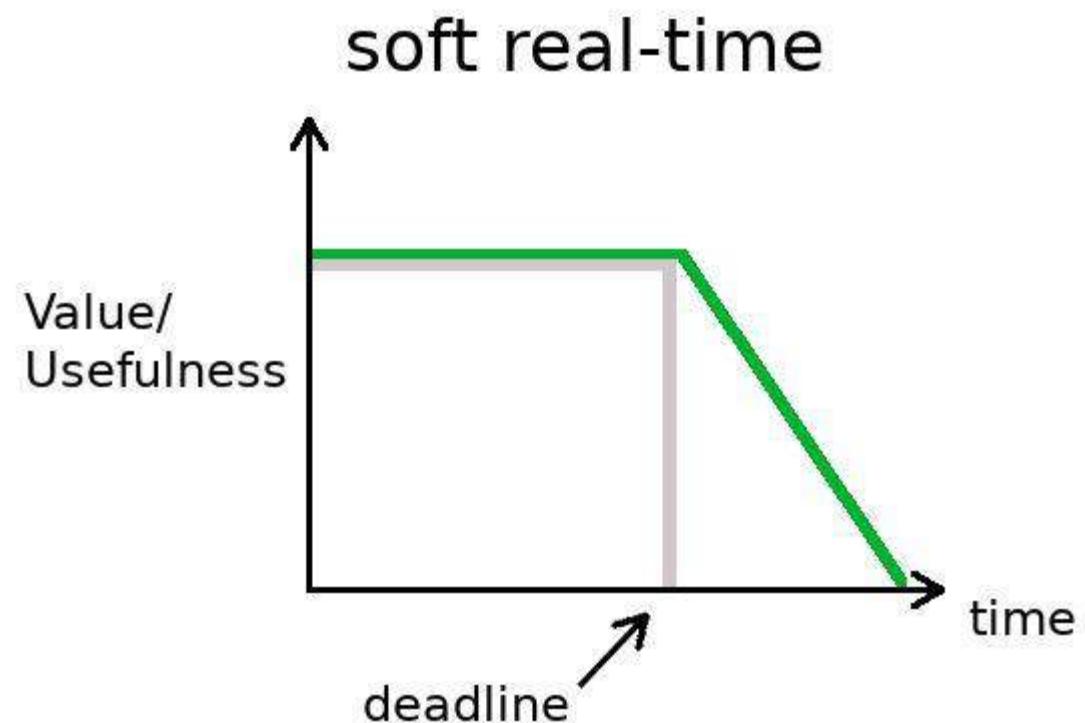
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Hard real-time operating system** - This is also a type of OS and it is predicted by a deadline. The predicted deadlines will react at a time  $t = 0$ . Some examples of this operating system are air bag control in cars, anti-lock brake, and engine control system etc.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- There are three different types of RTOS which are following
  - **Soft real-time operating system** - The soft real-time operating system has certain deadlines, may be missed and they will take the action at a time  $t=0+$ . The soft real-time operating system is a type of OS and it does not contain constrained to extreme rules. The critical time of this operating system is delayed to some extent. The examples of this operating system are the digital camera, mobile phones and online data etc.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

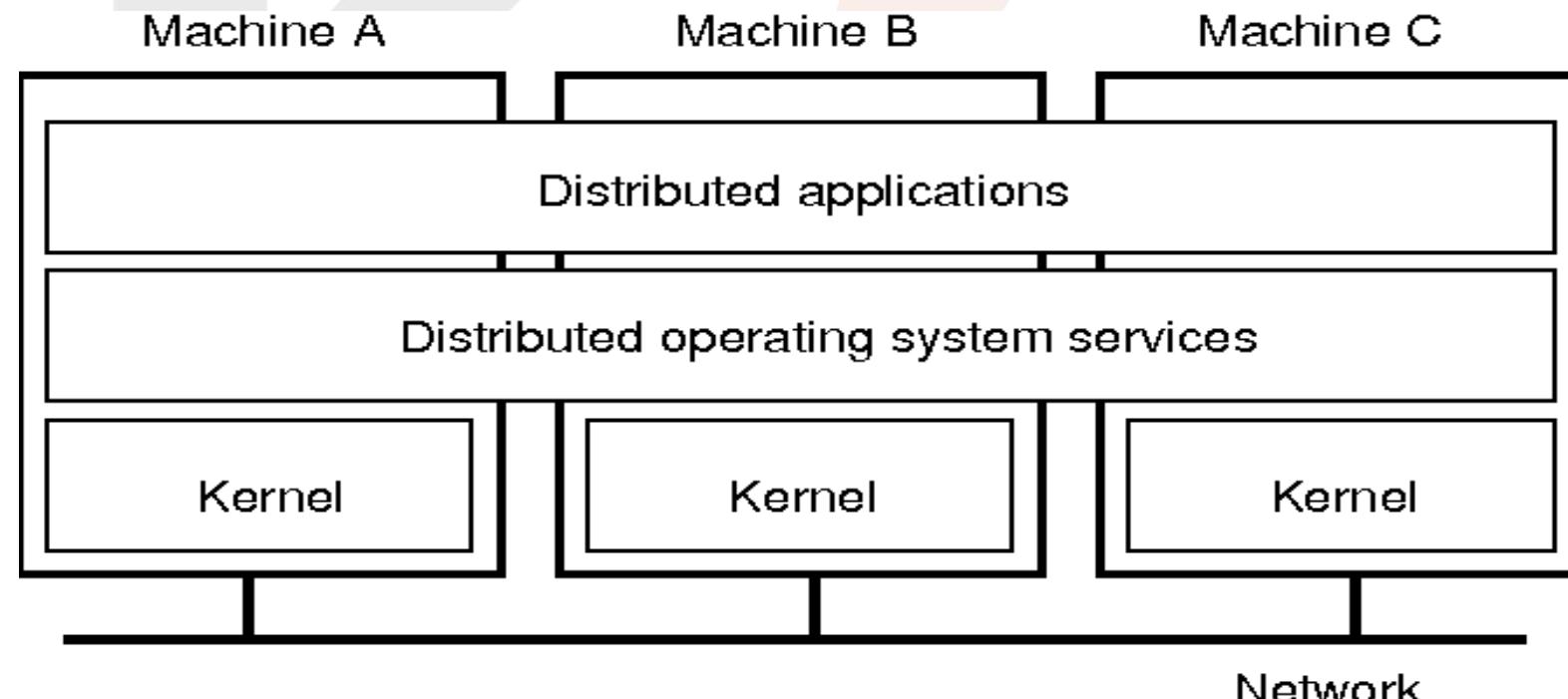


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Distributed OS

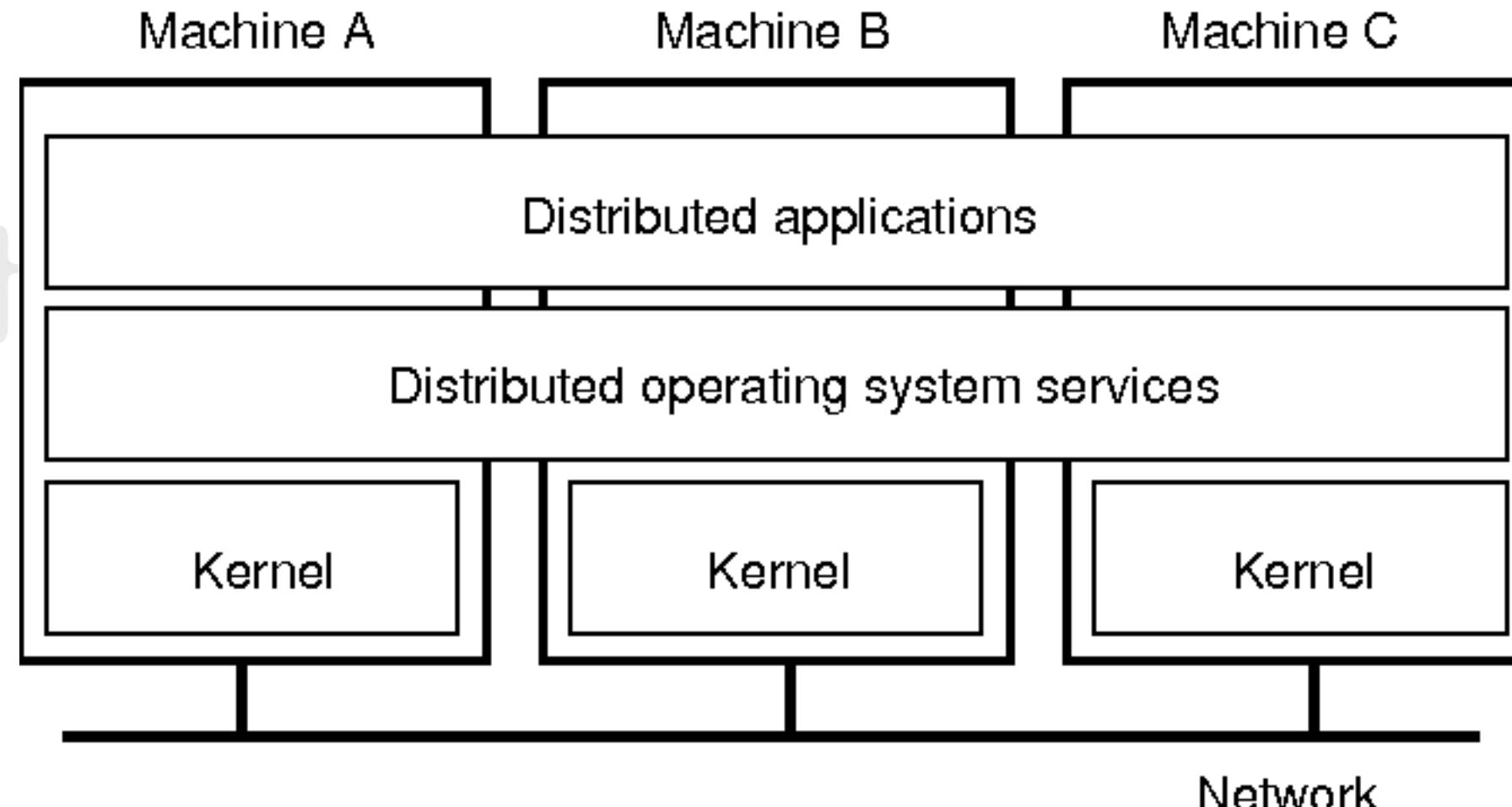
- A **distributed operating system** is a software over a collection of independent, networked, communicating, loosely coupled nodes and physically separate computational nodes.
- These nodes do not share memory or a clock. Instead, each node has its own local memory. The nodes communicate with one another through various networks, such as high-speed buses and the Internet.
- They handle jobs which are serviced by multiple CPUs. Each individual node holds a specific software subset of the global aggregate operating system.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

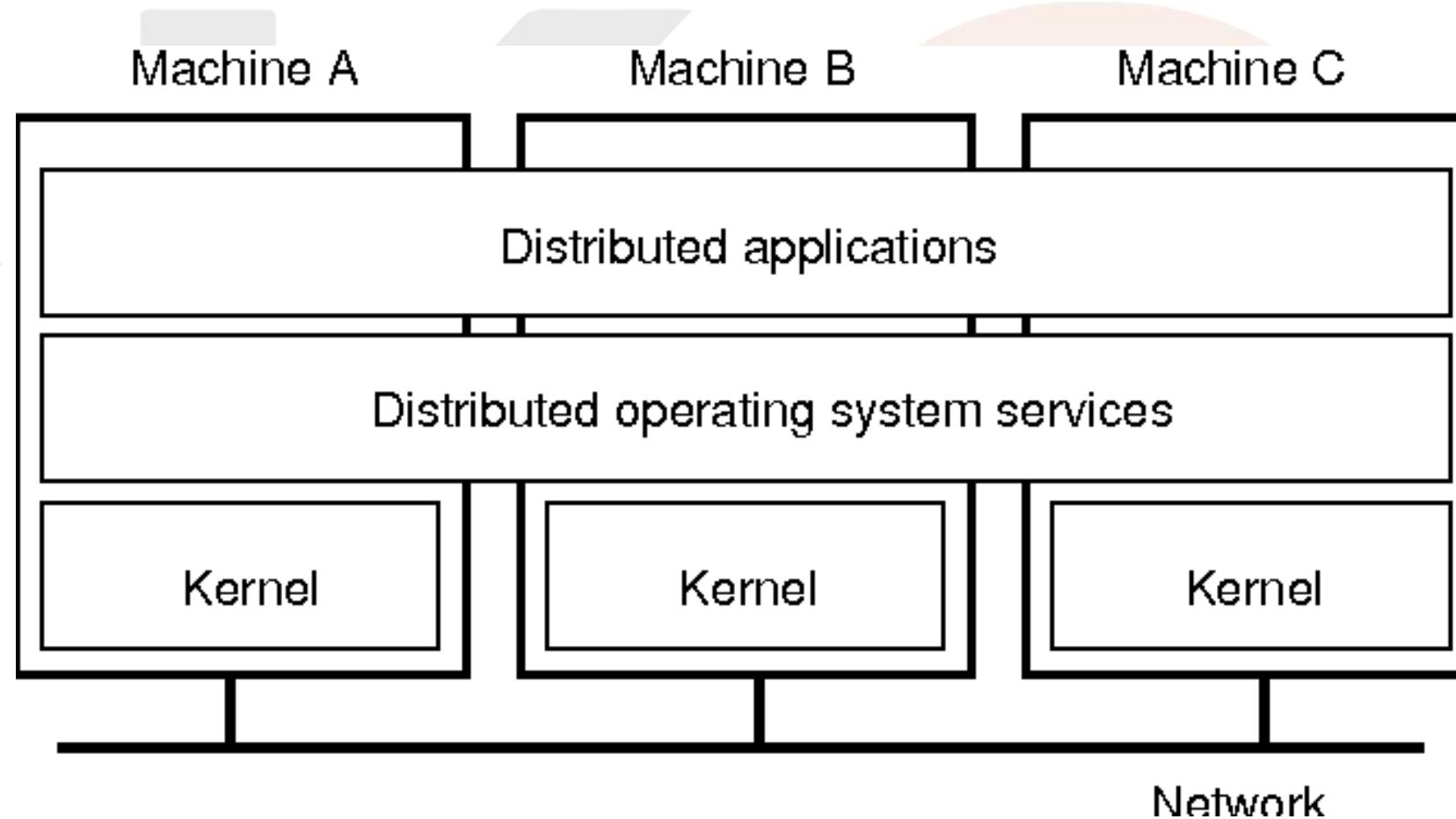
- Each subset is a composite of two distinct service provisioners. The first is a ubiquitous minimal **kernel**, or **microkernel**, that directly controls that node's hardware.
- Second is a higher-level collection of *system management components* that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

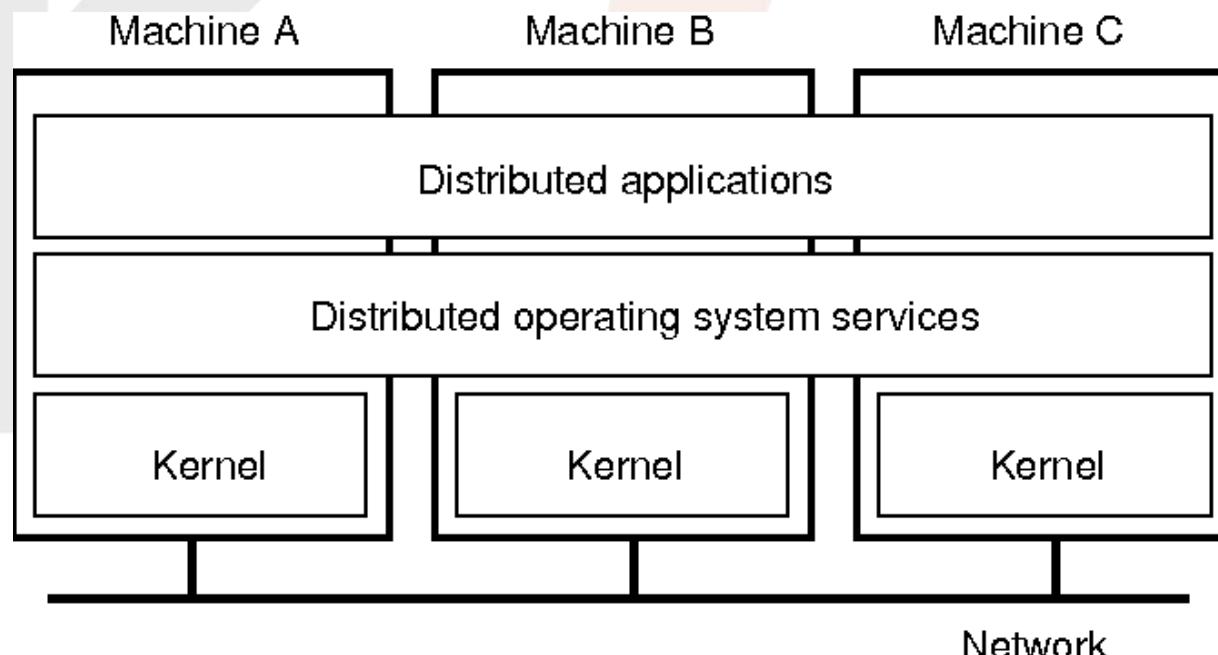
- The microkernel and the management components collection work together. They support the system's goal of integrating multiple resources and processing functionality into an efficient and stable system.
- This seamless integration of individual nodes into a global system is referred to as *transparency*, or *single system image*; describing the illusion provided to users of the global system's appearance as a single computational entity.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- To a user, a distributed OS works in a manner similar to a single-node, monolithic operating system. That is, although it consists of multiple nodes, it appears to users and applications as a single-node.
- There are four major reasons for building distributed systems: resource sharing, computation speedup, reliability, and communication.
- E.g. Plan 9 from Bell Labs, **Inferno**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

List - I	List - II
(1) Spooling	(i) Allows several jobs in memory to improve CPU utilization
(2) Multiprogramming	(ii) Access to shared resources among geographically dispersed computers in a transparent way
(3) Time sharing	(iii) Overlapping I/O and computations
(4) Distributed computing	(iv) Allows many users to share a computer simultaneously by switching processor frequently

	(1)	(2)	(3)	(4)
a)	iii	i	ii	iv
b)	iii	i	iv	ii
c)	iv	iii	ii	i
d)	ii	iii	iv	i

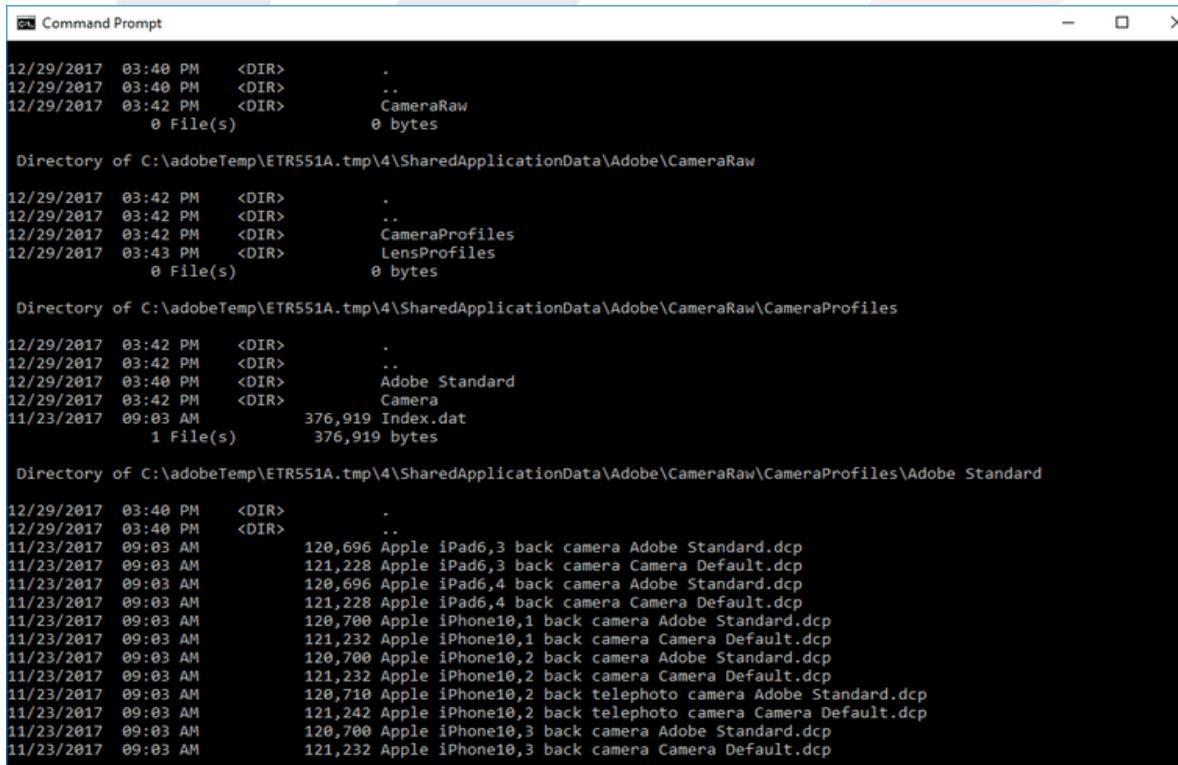
# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# User and Operating-System Interface

- There are several ways for users to interface with the operating system. Here, we discuss two fundamental approaches.
- One provides a command-line interface, or command interpreter, that allows users to directly enter commands to be performed by the operating system.
- The other allows users to interface with the operating system via a graphical user interface, or GUI.

- **Command Interpreters** - Some operating systems include the command interpreter in the kernel.
- Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (on interactive systems).
- On systems with multiple command interpreters to choose from, the interpreters are known as shells. For example, on UNIX and Linux systems, a user may choose among several different shells, including the *Bourne shell*, *C shell*, *Bourne-Again shell*, *Korn shell*, and others.



```
Command Prompt

12/29/2017 03:40 PM <DIR> .
12/29/2017 03:40 PM <DIR> ..
12/29/2017 03:42 PM <DIR> CameraRaw
    0 File(s)    0 bytes

Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw

12/29/2017 03:42 PM <DIR> .
12/29/2017 03:42 PM <DIR> ..
12/29/2017 03:42 PM <DIR> CameraProfiles
12/29/2017 03:43 PM <DIR> LensProfiles
    0 File(s)    0 bytes

Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw\CameraProfiles

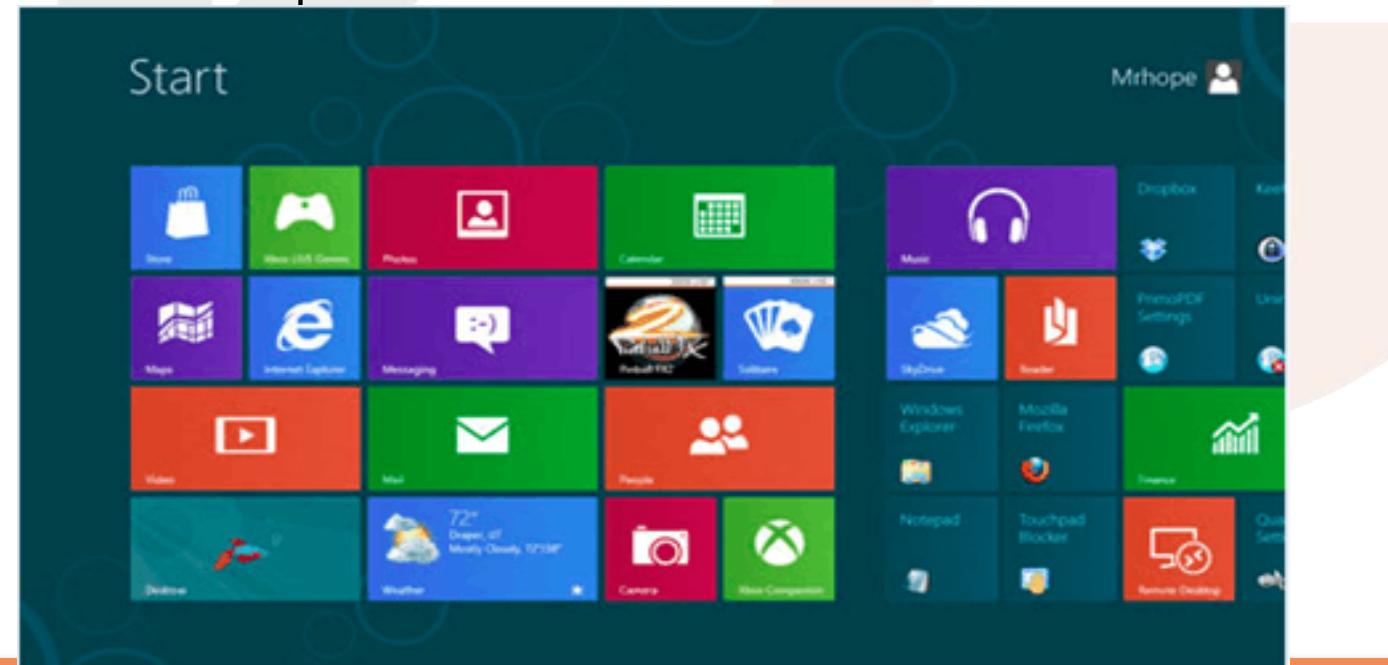
12/29/2017 03:42 PM <DIR> .
12/29/2017 03:42 PM <DIR> ..
12/29/2017 03:40 PM <DIR> Adobe Standard
12/29/2017 03:42 PM <DIR> Camera
11/23/2017 09:03 AM 376,919 Index.dat
    1 File(s)    376,919 bytes

Directory of C:\adobeTemp\ETR551A.tmp\4\SharedApplicationData\Adobe\CameraRaw\CameraProfiles\Adobe Standard

12/29/2017 03:40 PM <DIR> .
12/29/2017 03:40 PM <DIR> ..
11/23/2017 09:03 AM 120,696 Apple iPad6,3 back camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,228 Apple iPad6,3 back camera Camera Default.dcp
11/23/2017 09:03 AM 120,696 Apple iPad6,4 back camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,228 Apple iPad6,4 back camera Camera Default.dcp
11/23/2017 09:03 AM 120,700 Apple iPhone10,1 back camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,232 Apple iPhone10,1 back camera Camera Default.dcp
11/23/2017 09:03 AM 120,700 Apple iPhone10,2 back camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,232 Apple iPhone10,2 back camera Camera Default.dcp
11/23/2017 09:03 AM 120,710 Apple iPhone10,2 back telephoto camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,242 Apple iPhone10,2 back telephoto camera Camera Default.dcp
11/23/2017 09:03 AM 120,700 Apple iPhone10,3 back camera Adobe Standard.dcp
11/23/2017 09:03 AM 121,232 Apple iPhone10,3 back camera Camera Default.dcp
```

To access all paid content get KG Prime at ₹25/day [CLICK HERE ↗](#)

- **Graphical User Interfaces** - A second strategy for interfacing with the operating system is through a user-friendly graphical user interface, or GUI.
- Here, users employ a mouse-based window- and-menu system characterized by a desktop.
- The user moves the mouse to position its pointer on images, or icons, on the screen (the desktop) that represent programs, files, directories, and system functions.
- Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a folder—or pull down a menu that contains commands.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- Because a mouse is impractical for most mobile systems, smartphones and handheld tablet computers typically use a touchscreen interface.
- Here, users interact by making gestures on the touchscreen—for example, pressing and swiping fingers across the screen.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- The choice of whether to use a command-line or GUI interface is mostly one of personal preference.
- System administrators who manage computers and power users who have deep knowledge of a system frequently use the command-line interface. For them, it is more efficient, giving them faster access to the activities they need to perform.
- Indeed, on some systems, only a subset of system functions is available via the GUI, leaving the less common tasks to those who are command-line knowledgeable.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

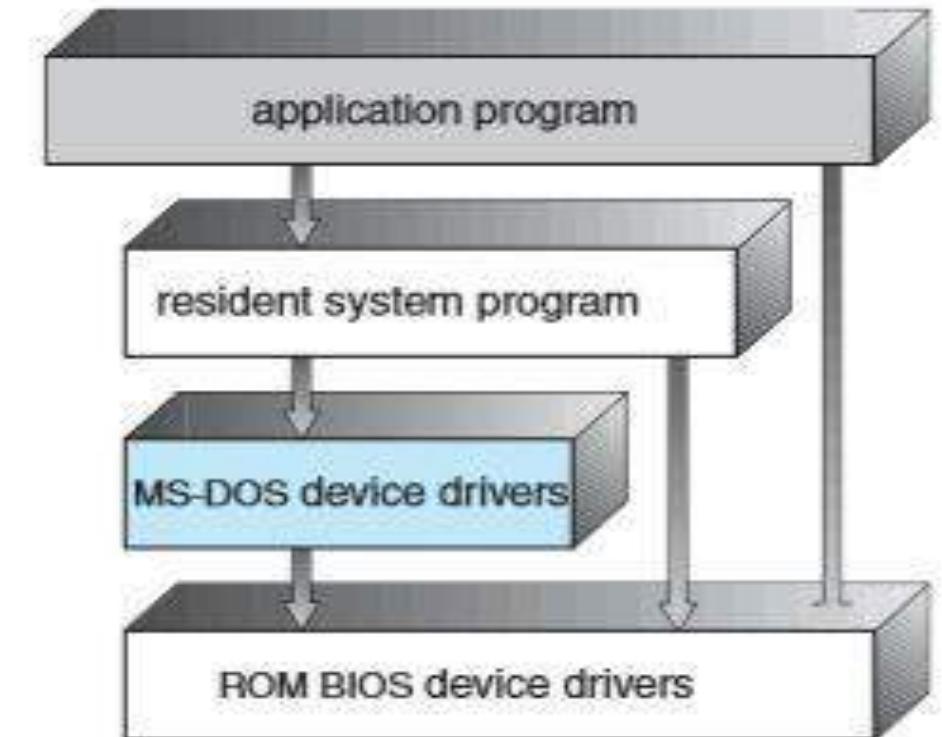


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

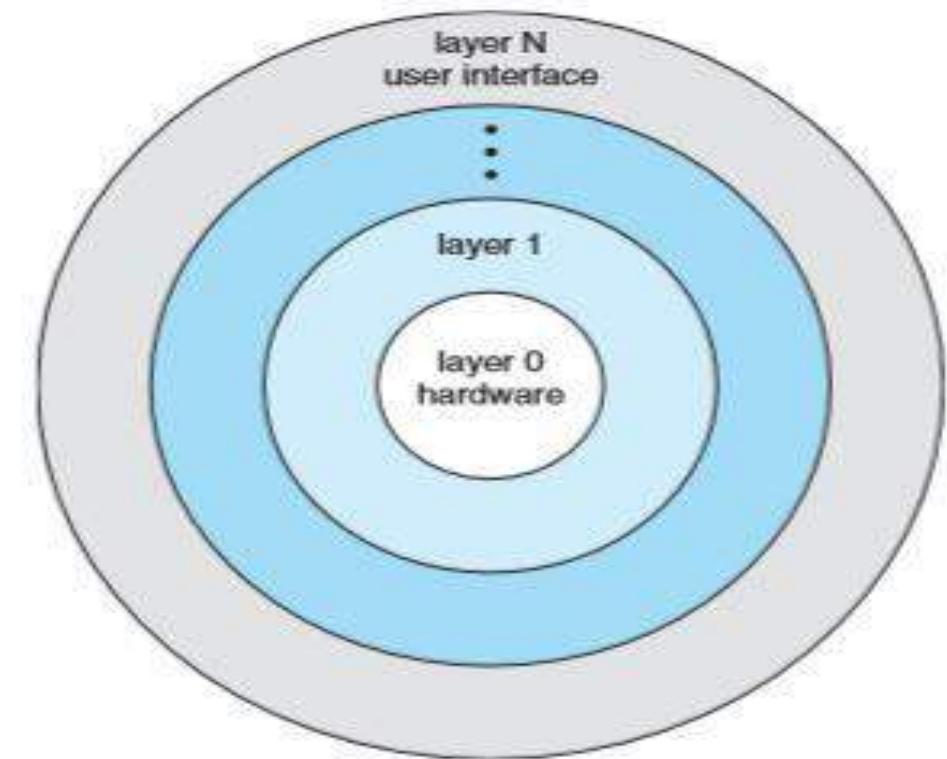
## Structure of Operating System

- A common approach is to partition the task into small components, or modules, rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.
- **Simple Structure** - Many operating systems do not have well-defined structures. Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system.



MS-DOS layer structure.

- **Layered Approach** - With proper hardware support, operating systems can be broken into pieces. The operating system can then retain much greater control over the computer and over the applications that make use of that computer.
- Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems.
- Under a top-down approach, the overall functionality and features are determined and are separated into components.
- Information hiding is also important, because it leaves programmers free to implement the low-level routines as they see fit.
- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer  $N$ ) is the user interface.



A layered operating system.

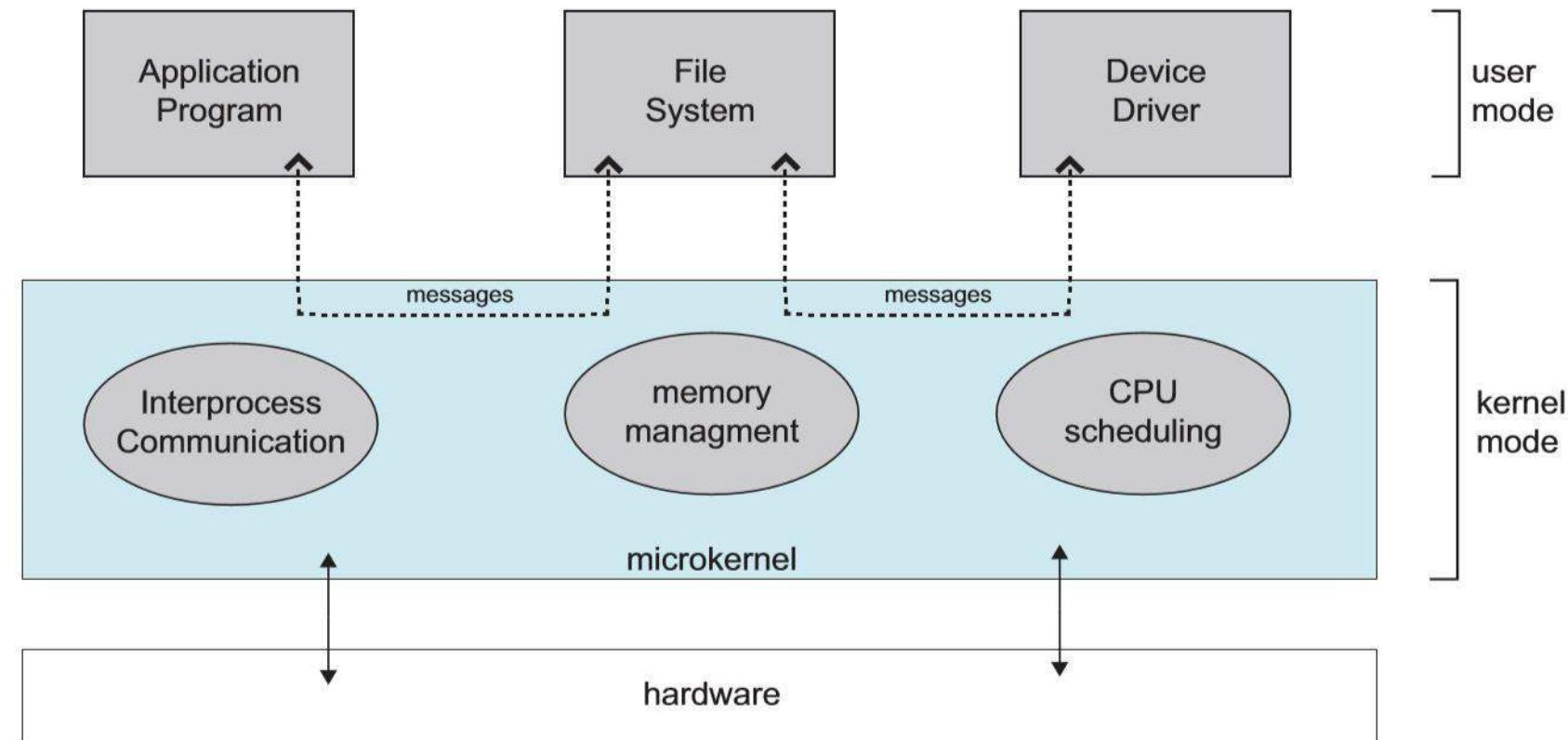


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Micro-Kernel approach

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel.
- When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.
- The MINIX 3 microkernel, for example, has only approximately 12,000 lines of code.
- Developer Andrew S. Tanenbaum

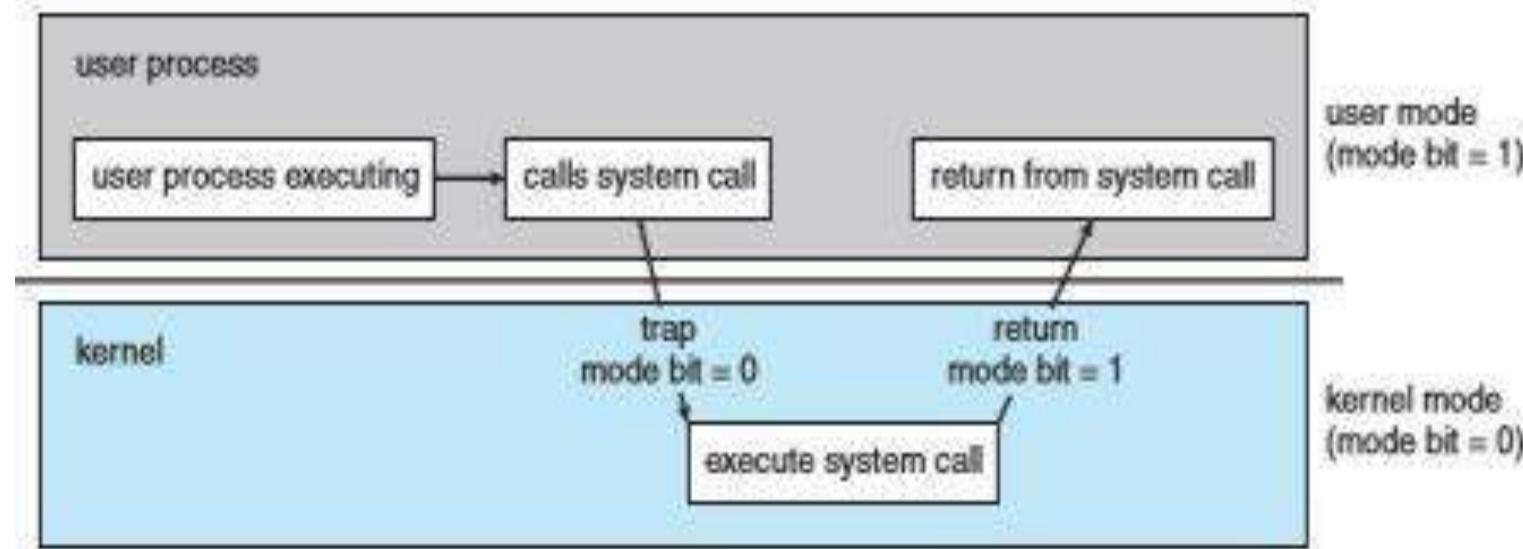


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Mode bit

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user- defined code.
- At the very least, we need two separate *modes* of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).
- When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)



## User Mode



## Kernel Mode

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0).
- The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.
- The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management, and interrupt management.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Farmer



Instruction set programmer



To access all paid content get **KG Prime** at ₹25/day

[CHOOSE PLAN](#)



Industry(Gov / Private)



OS designer(API)

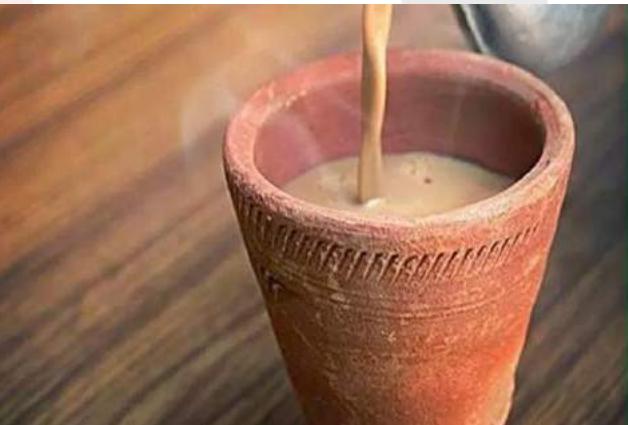
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



Chef

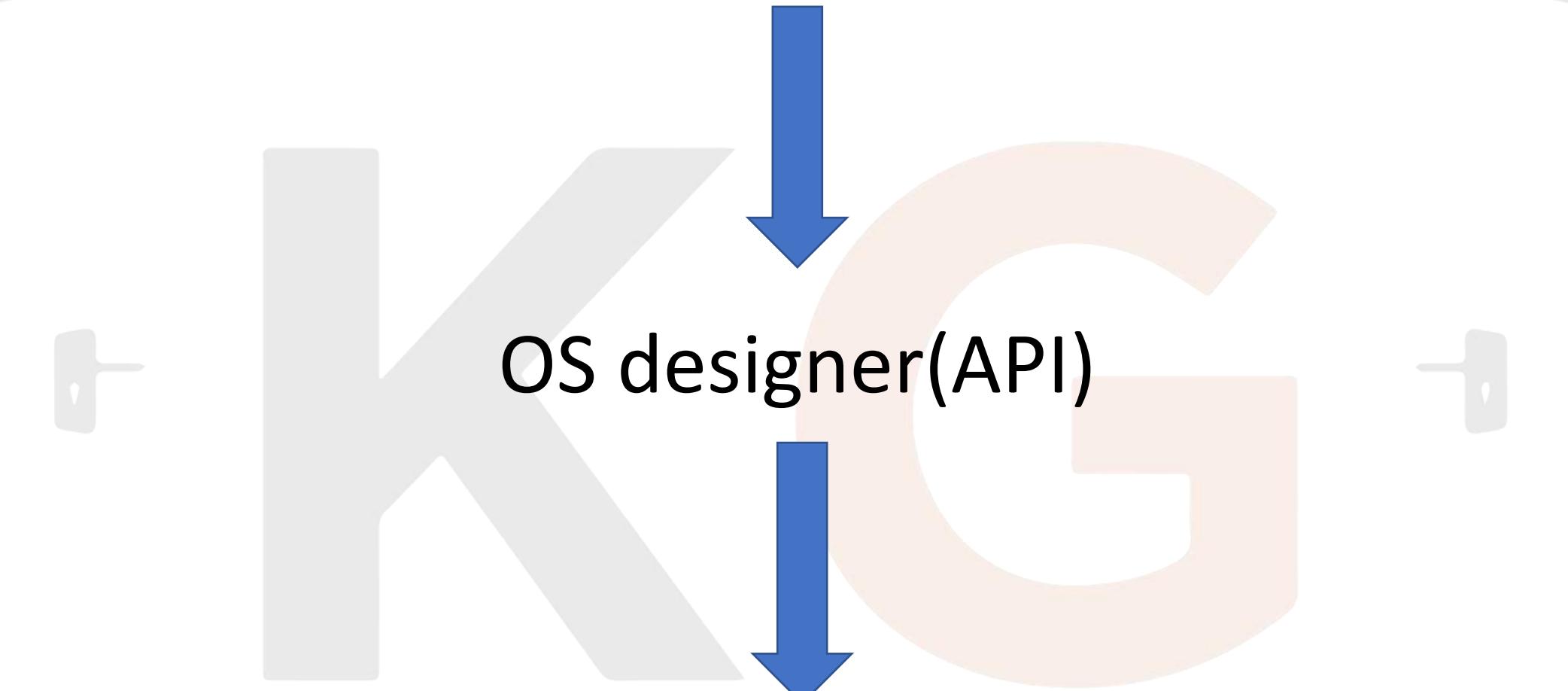


Programmer



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Instruction set programmer



Programmer

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

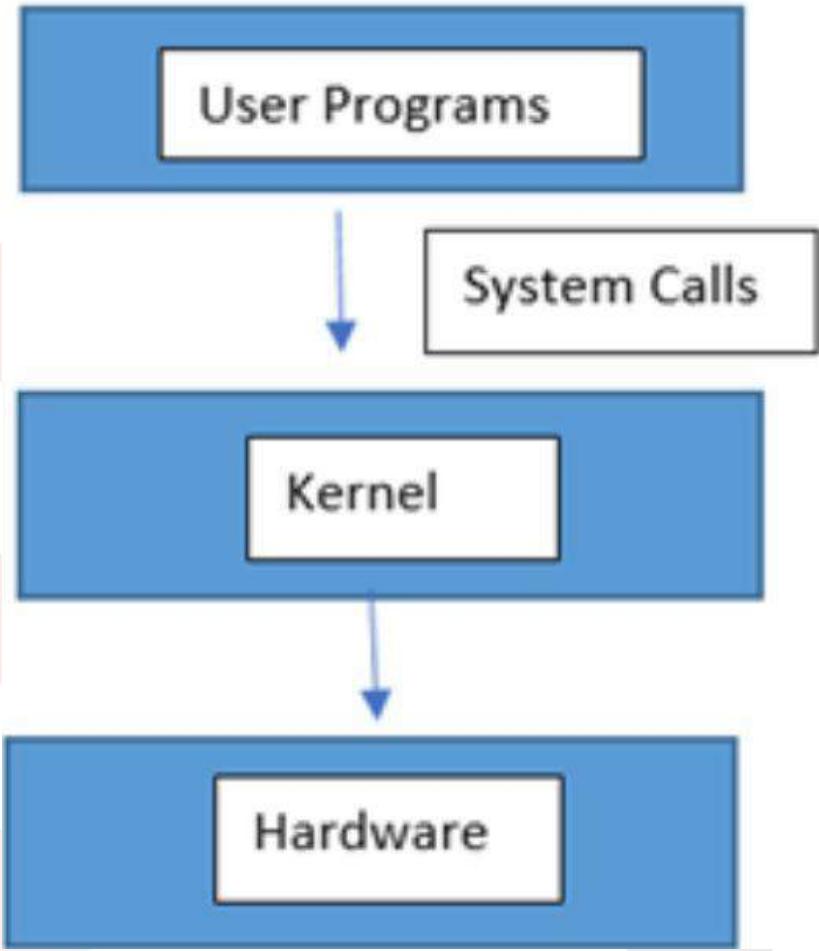


# break

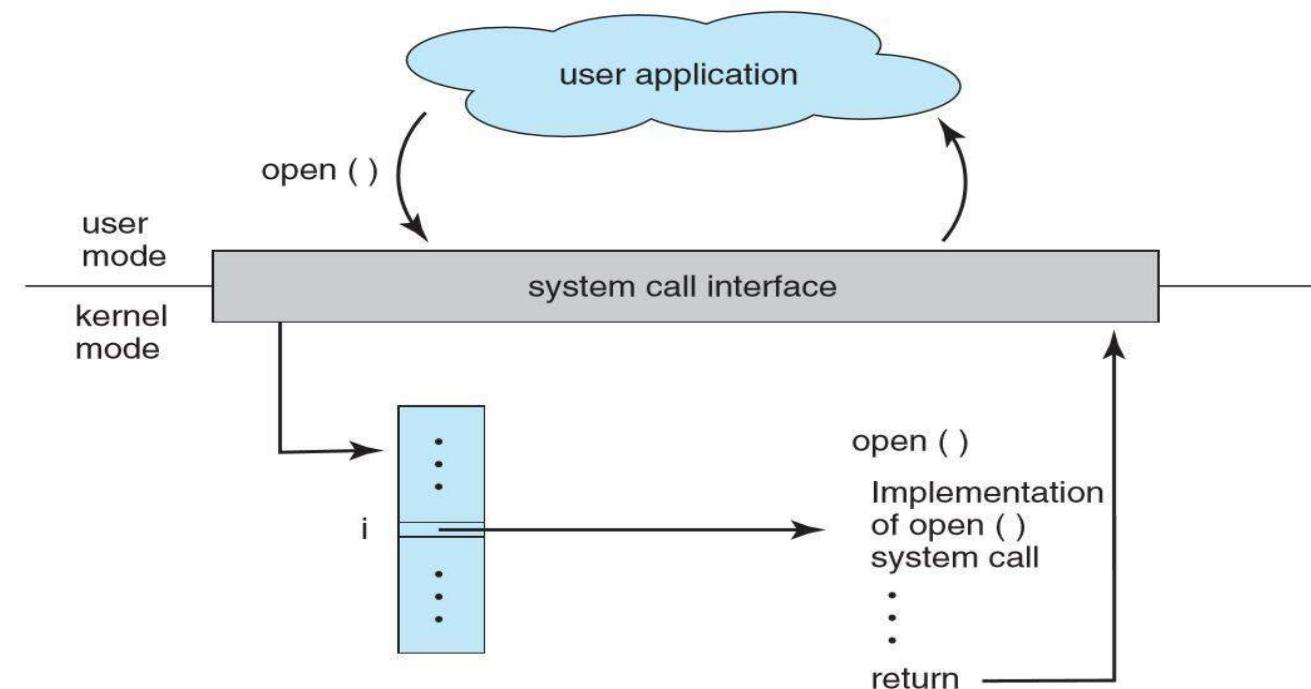
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# System call

- System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.
- System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++.
- The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.



- Three of the most common APIs available to application programmers are the
  - Windows API for Windows systems
  - POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and Mac OS X)
  - The Java API for programs that run on the Java virtual machine.
- The caller need know nothing about how the system call is implemented or what it does during execution. Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.

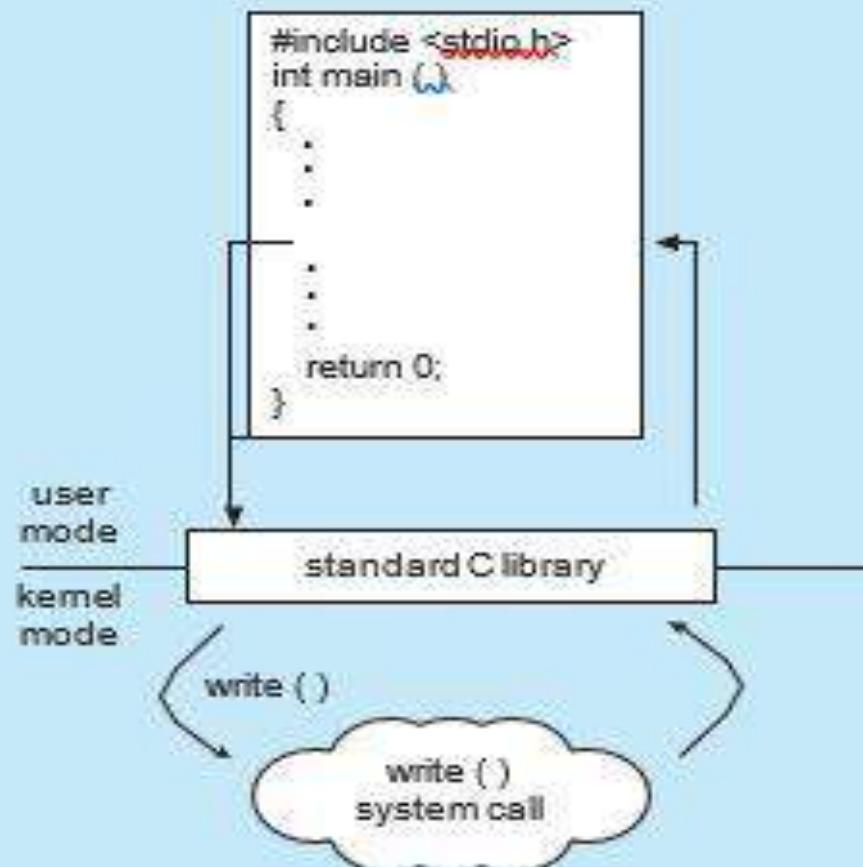


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below:



- **Types of System Calls** - System calls can be grouped roughly into six major categories: [process control](#), [file manipulation](#), [device manipulation](#), [information maintenance](#), [communications](#), and [protection](#).
- **Process control**
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory

## • File management

- create file, delete file

- open, close

- read, write, reposition

- get file attributes, set file attributes

- Device management

- request device, release device

- read, write, reposition

- get device attributes, set device attributes

- logically attach or detach devices

- **Information maintenance**

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

## • Communications

- create, delete communication connection

- send, receive messages transfer status information

# Fork

- **Requirement of Fork command**

- In number of applications specially in those where work is of repetitive nature, like web server i.e. with every client we have to run similar type of code. Have to create a separate process every time for serving a new request.
- So it must be a better solution that instead to creating a new process every time from scratch we must have a short command using which we can do this logic.

- Idea of fork command

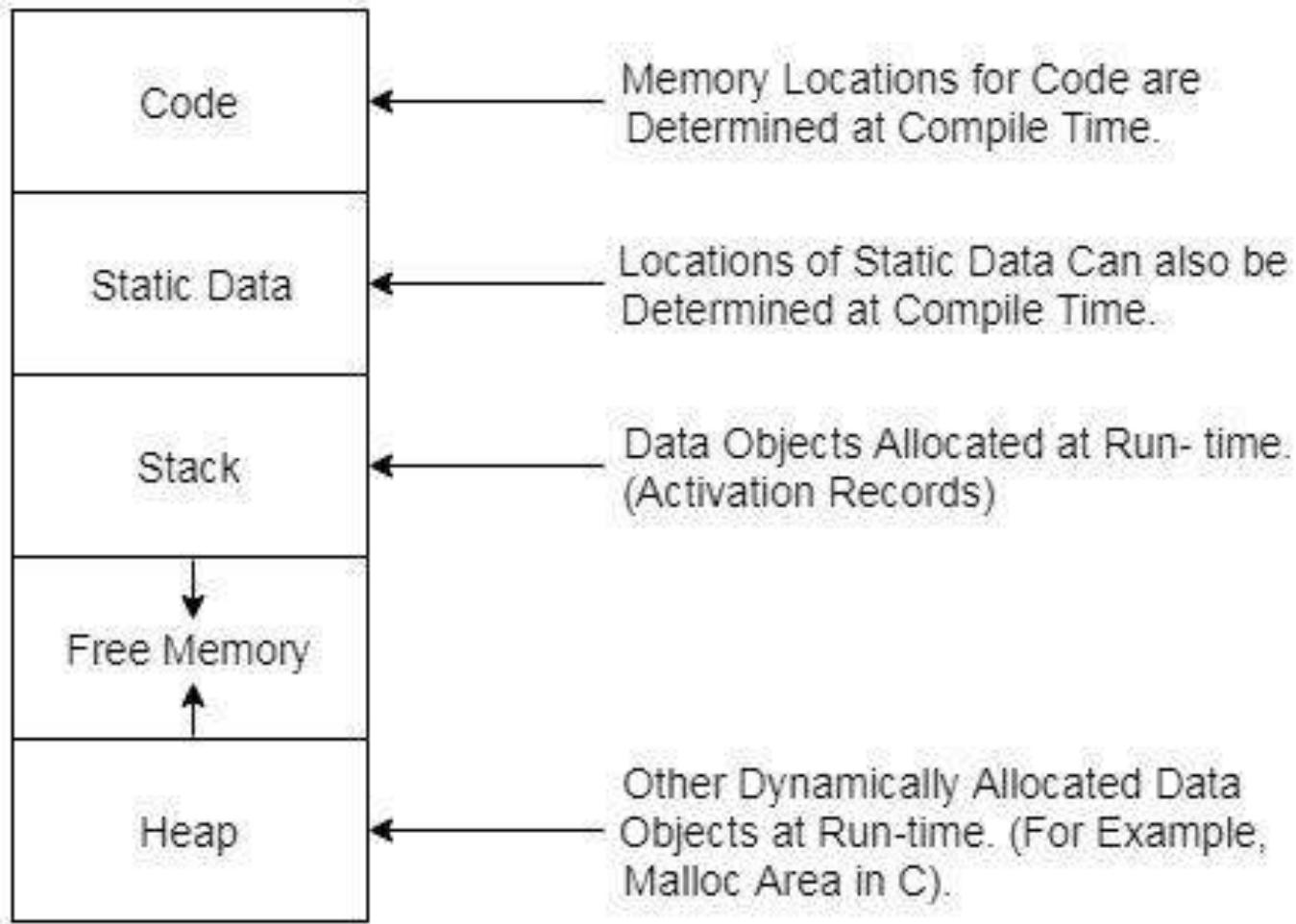
- Here fork command is a system command using which the entire image of the process can be copied and we create a new process, this idea help us to complete the creation of the new process with speed.
- After creating a process, we must have a mechanism to identify weather in newly created process which one is child and which is parent.

- **Implementation of fork command**

- In general, if fork return 0 then it is child and if fork return 1 then it is parent, and then using a programmer level code we can change the code of child process to behave as new process.



- **Advantages of using fork commands**
  - Now it is relatively easy to create and manage similar types of process of repetitive nature with the help of fork command.
- **Disadvantage**
  - To create a new process by fork command we have to do system call as, fork is system function
    - Which is slow and time taking
    - Increase the burden over Operating System
  - Different image of the similar type of task have same code part which means we have the multiple copy of the same data waiting the main memory



# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Fork is (ISRO 2008)

a) the creation of a new job

b) the dispatching of a task

c) increasing the priority of a task

d) the creation of a new process



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** The following C program is executed on a Unix/Linux system:

```
#include <unistd.h>
int main ()
{
    int i ;
    for (i=0; i<10; i++)
        if (i%2 == 0)
            fork ( ) ;
    return 0 ;
}
```

The total number of child processes created is \_\_\_\_ (GATE-2019) (1 Marks)

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A process executes the code

```
fork();
```

```
fork();
```

```
fork();
```

the total number of child processes created is (GATE - 2012) (1 Marks)

a) 3

b) 4

c) 7

d) 8

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

Q A process executes the following code (GATE - 2008) (2 Marks)

```
for (i=0; i<n; i++)  
fork();
```

a) n

b)  $(2^n) - 1$

c)  $2^n$

d)  $(2^{n+1}) - 1$

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

```
if (fork() == 0)
{
    a = a + 5;
    printf ("%d, %d /n", a, &a);
}
else
{
    a = a -5;
    printf("%d, %d /n", a, &a);
}
```

Let  $u$ ,  $v$  be the values printed by the parent process, and  $x$ ,  $y$  be the values printed by the child process. Which one of the following is TRUE? (GATE-2005) (2 Marks)

- a)  $u = x + 10$  and  $v = y$
- b)  $u = x + 10$  and  $v \neq y$
- c)  $u + 10 = x$  and  $v = y$
- d)  $u + 10 = x$  and  $v \neq y$

Q a process executes the following segment of code:

```
fork(i=1;i<=n;i++)
```

```
fork();
```

the number of new processes created is (GATE-2004) (1 Marks)

a) n

b)  $(n(n+1))/2$

c)  $2^n - 1$

d)  $3^n - 1$

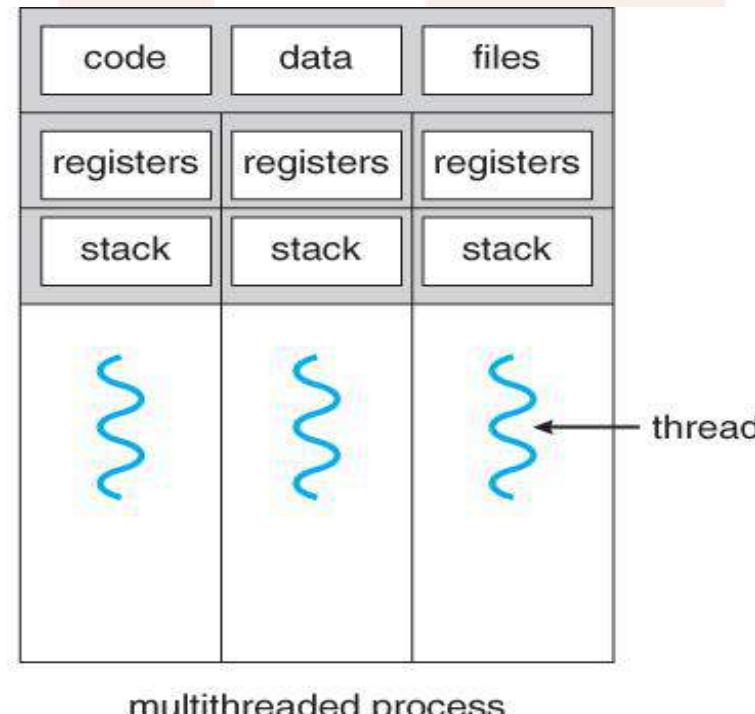
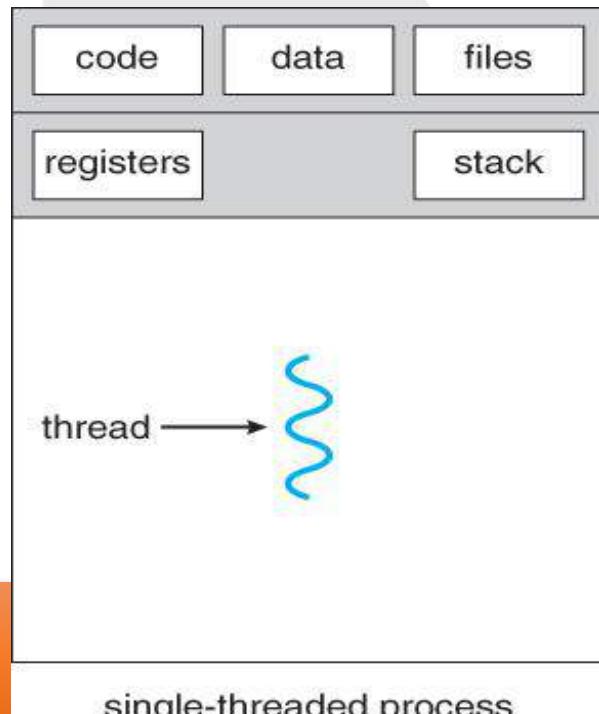
To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, ( and a thread ID. )
- Traditional (heavyweight) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.
- multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.



To access all

ay CLICK HERE ↗

## Motivation

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.
- This is particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking.
- For example in a word processor, a background thread may check spelling and grammar while a foreground thread processes user input ( keystrokes ), while yet a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.
- Another example is a web server - Multiple threads allow for multiple requests to be satisfied simultaneously.

There are four major categories of benefits to multi-threading:

- **Responsiveness** - One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
- **Resource sharing** - By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
- **Economy** - Creating and managing threads ( and context switches between them ) is much faster than performing the same tasks for processes.
- **Scalability**, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors. ( Note that single threaded processes can still benefit from multi-processor architectures when there are multiple processes contending for the CPU, i.e. when the load average is above some certain threshold. )

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Which of the following is/are shared by all the threads in a process? (GATE - 2017) (1 Marks)

I. Program Counter

(A) I and II only

II. Stack

(B) III only

III. Address space

(C) IV only

IV. Registers

(D) III and IV only

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Threads of a process share **(GATE - 2017) (1 Marks)**

- (A) global variables but not heap
- (B) heap but not global variables
- (C) neither global variables nor heap
- (D) both heap and global variables

Q A thread is a light weight process. In the above statement, weight refers to (NET-DEC-2012)

(A) time

(B) number of resources

(C) speed

(D) All the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A thread is usually defined as a “light weight process” because an operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is TRUE? **(GATE - 2011) (1 Marks)**

- (A)** On per-thread basis, the OS maintains only CPU register state
- (B)** The OS does not maintain a separate stack for each thread
- (C)** On per-thread basis, the OS does not maintain virtual memory state
- (D)** On per-thread basis, the OS maintains only scheduling and accounting information

# Break

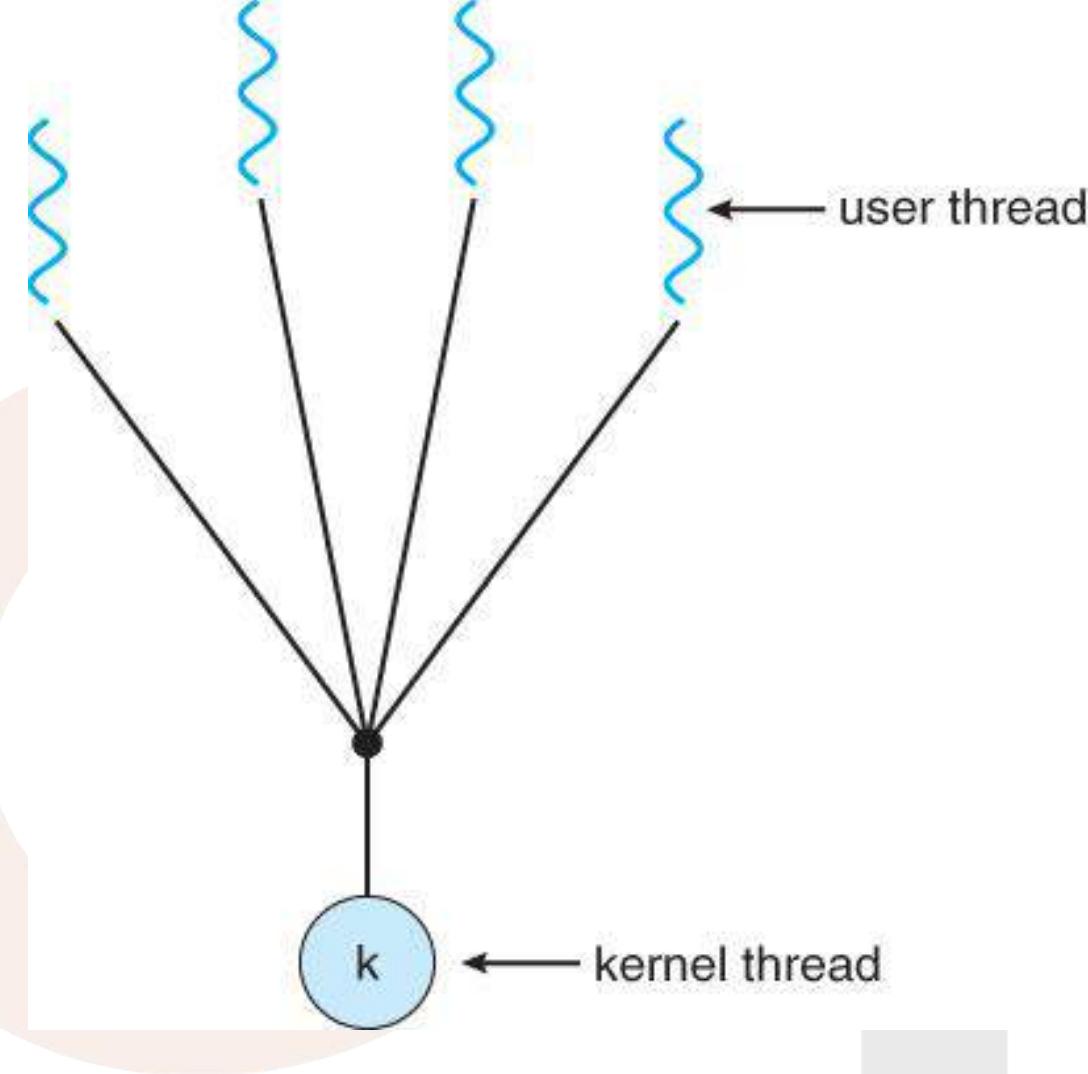
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Multithreading Models

- There are two types of threads to be managed in a modern system: User threads and kernel threads.
- User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
- Kernel threads are supported within the kernel of the OS itself. All modern OS support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously

## Many-To-One Model

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- However, if a blocking system call is made, then the entire process blocks, even if the other user threads would otherwise be able to continue.
- Because a single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.
- Green threads for Solaris implement the many-to-one model in the past, but few systems continue to do so today.

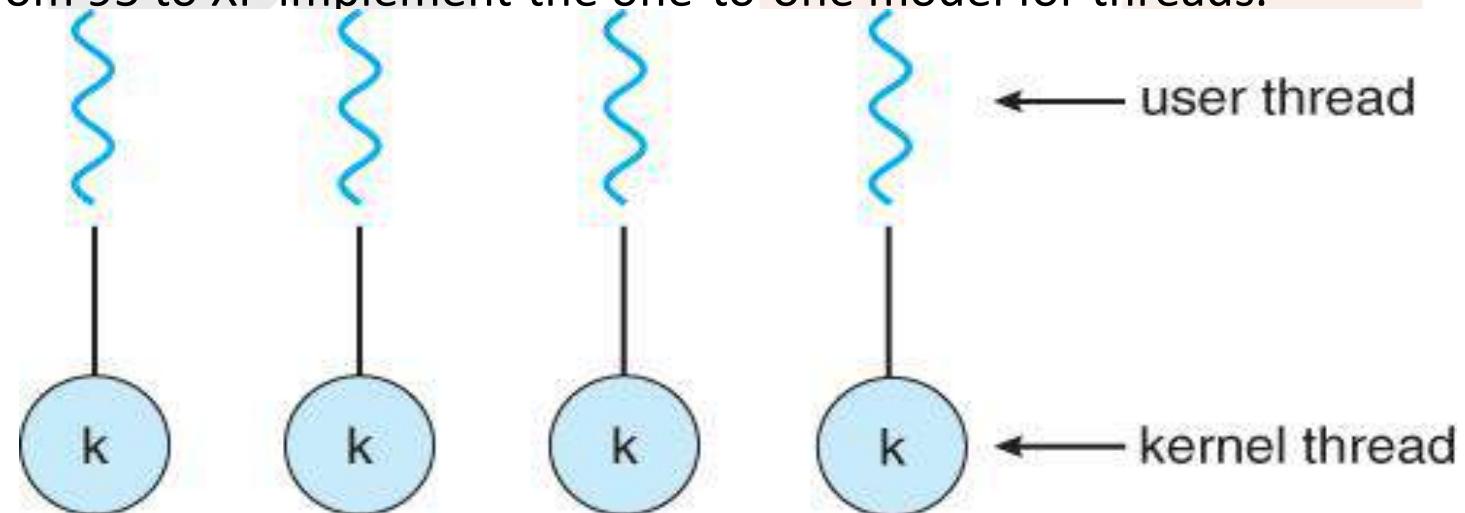


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## One-To-One Model

- The one-to-one model creates a separate kernel thread to handle each user thread.
- One-to-one model overcomes the problems listed above involving blocking system calls and the splitting of processes across multiple CPUs.
- However, the overhead of managing the one-to-one model is more significant, involving more overhead and slowing down the system.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.

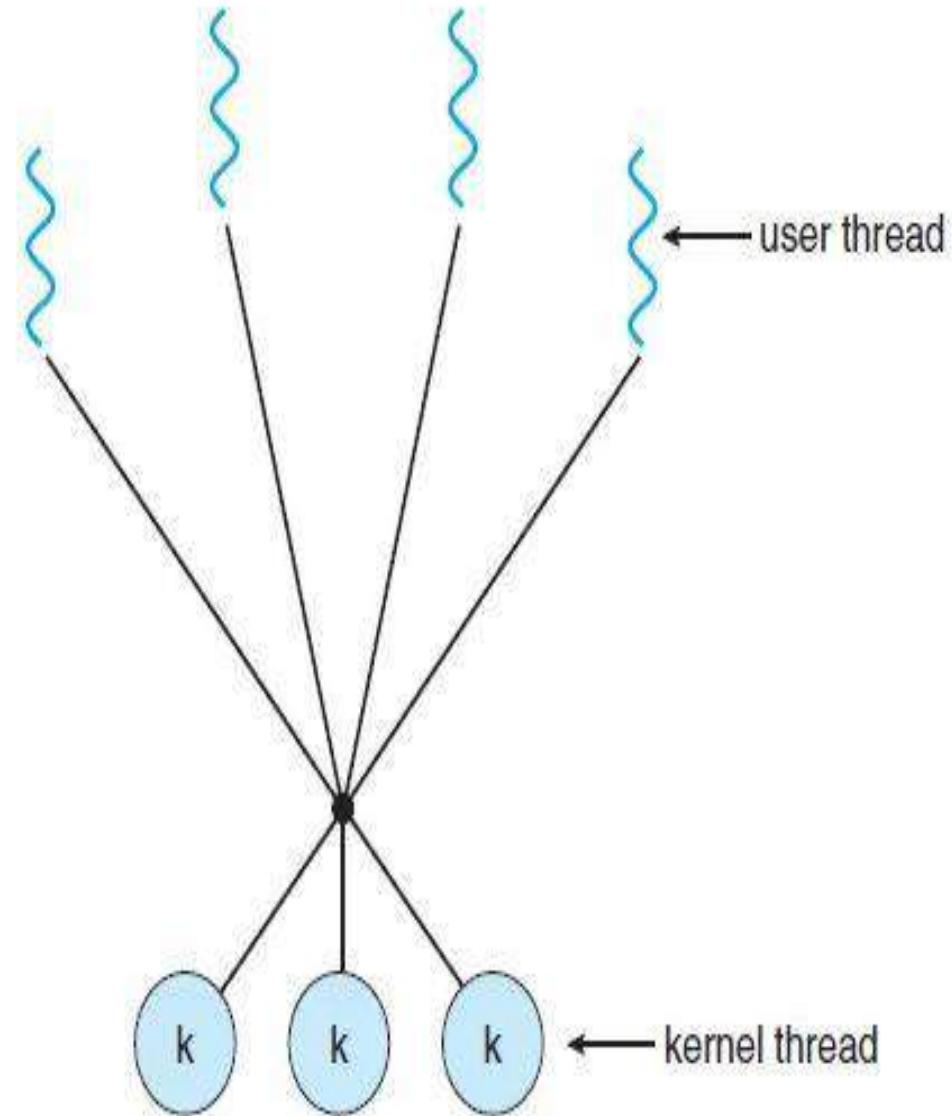


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors.



# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** User level threads are threads that are visible to the programmer and are unknown to the kernel. The operating system kernel supports and manages kernel level threads. Three different types of models relate user and kernel level threads. Which of the following statements is/are true? **(NET-NOV-2017)**

- (a) (i)** The Many - to - one model maps many user threads to one kernel thread
  - (ii)** The one - to - one model maps one user thread to one kernel thread
  - (iii)** The many - to - many model maps many user threads to smaller or equal kernel threads
- 
- (b) (i)** Many - to - one model maps many kernel threads to one user thread
  - (ii)** One - to - one model maps one kernel thread to one user thread
  - (iii)** Many - to - many model maps many kernel threads to smaller or equal user threads
- 
- A) (a) is true; (b) is false**
- B) (a) is false; (b) is true**
- C) Both (a) and (b) are true**
- D) Both (a) and (b) are false**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q One of the disadvantages of user level threads compared to Kernel level threads is (NET-JAN-2017)**

- a)** If a user level thread of a process executes a system call, all threads in that process are blocked.
- b)** Scheduling is application dependent.
- c)** Thread switching doesn't require kernel mode privileges.
- d)** The library procedures invoked for thread management in user level threads are local procedures.

**Q Which one of the following is FALSE? (GATE - 2014) (1 Marks)**

- (A) User level threads are not scheduled by the kernel.**
- (B) When a user level thread is blocked, all other threads of its process are blocked.**
- (C) Context switching between user level threads is faster than context switching between kernel level threads.**
- (D) Kernel level threads cannot share the code segment**

**Q** Let the time taken to switch between user mode and kernel mode of execution be  $T_1$  while time taken to switch between two user processes be  $T_2$ . Which of the following is correct? (NET-JUNE-2013)

- a)  $T_1 < T_2$
- b)  $T_1 > T_2$
- c)  $T_1 = T_2$
- d) Nothing can be said about the relation between  $T_1$  and  $T_2$ .

**Q** Let the time taken to switch between user and kernel modes of execution be  $t_1$  while the time taken to switch between two processes be  $t_2$ . Which of the following is TRUE? (GATE-2011) (1 Marks)

(A)  $t_1 > t_2$

(B)  $t_1 = t_2$

(C)  $t_1 < t_2$

(D) nothing can be said about the relation between  $t_1$  and  $t_2$

**Q** Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE? **(GATE - 2007) (1 Marks)**

- A)** Context switch time is longer for kernel level threads than for user level threads.
- B)** User level threads do not need any hardware support.
- C)** Related kernel level threads can be scheduled on different processors in a multi-processor system
- D)** Blocking one kernel level thread blocks all related threads

**Q** Consider the following statements with respect to user-level threads and kernel supported threads

- i. context switch is faster with kernel-supported threads
- ii. for user-level threads, a system call can block the entire process
- iii. Kernel supported threads can be scheduled independently
- iv. User level threads are transparent to the kernel

Which of the above statements are true? **(GATE-2004) (1 Marks)**

**(A)** (ii), (iii) and (iv) only

**(B)** (ii) and (iii) only

**(C)** (i) and (iii) only

**(D)** (i) and (ii) only

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Process

- In general, a process is a program in execution.
- A Program is not a Process by default. A program is a passive entity, i.e. a file containing a list of instructions stored on disk (secondary memory) (often called an executable file).
- A program becomes a Process when an executable file is loaded into main memory and when its PCB is created.
- A process on the other hand is an Active Entity, which require resources like main memory, CPU time, registers, system buses etc.



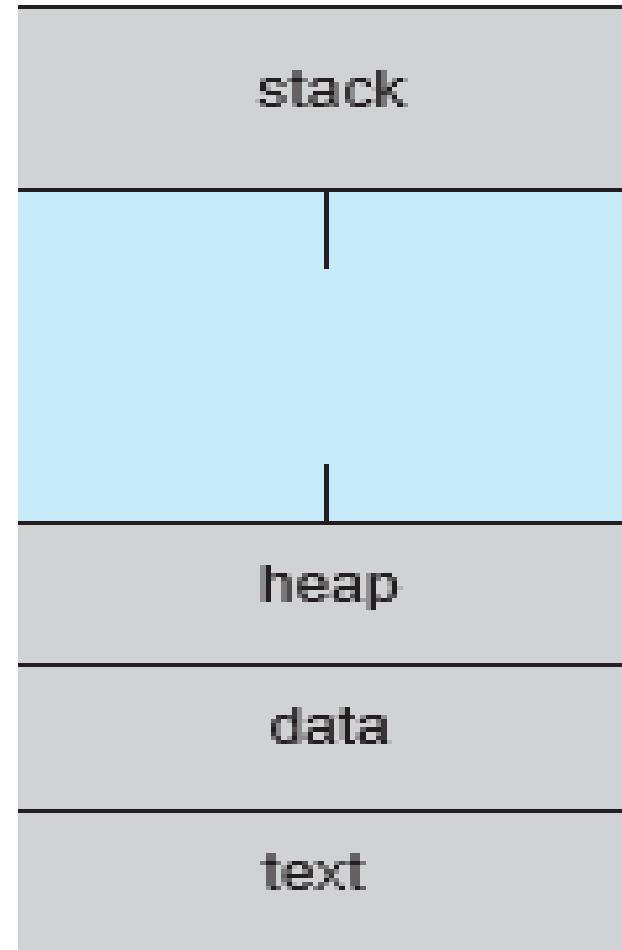
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Even if two processes may be associated with same program, they will be considered as two separate execution sequences and are totally different process.
- For instance, if a user has invoked many copies of web browser program, each copy will be treated as separate process. even though the text section is same but the data, heap and stack sections can vary.

- A Process consists of following sections:

- **Text section:** also known as Program Code.
- **Stack:** which contains the temporary data (Function Parameters, return addresses and local variables).
- **Data Section:** containing global variables.
- **Heap:** which is memory dynamically allocated during process runtime.

max



Q Process is a: (ISRO 2009)

- a) A program in high level language kept on disk
- b) Contents of main memory
- c) A program in execution
- d) A job in secondary memory



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Process Control Block (PCB)

- Each process is represented in the operating system by a process control block (PCB) — also called a task control block.
- PCB simply serves as the repository for any information that may vary from process to process. It contains many pieces of information associated with a specific process, including these:
  - **Process state:** The state may be new, ready, running, waiting, halted, and so on.
  - **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
  - **CPU registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

process state
process number
program counter
registers
memory limits
list of open files
• • •

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

process state
process number
program counter
registers
memory limits
list of open files
• •



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# The Human Life Cycle

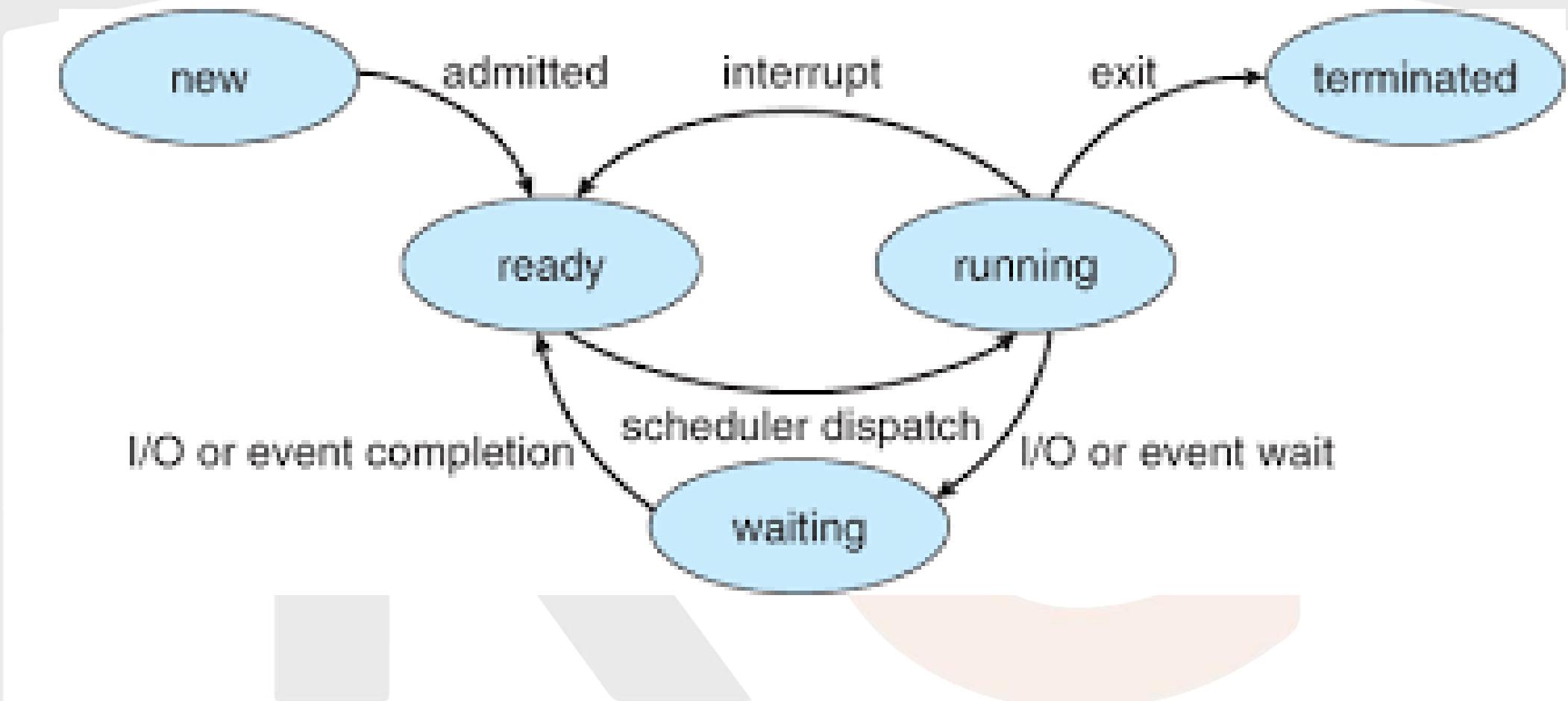


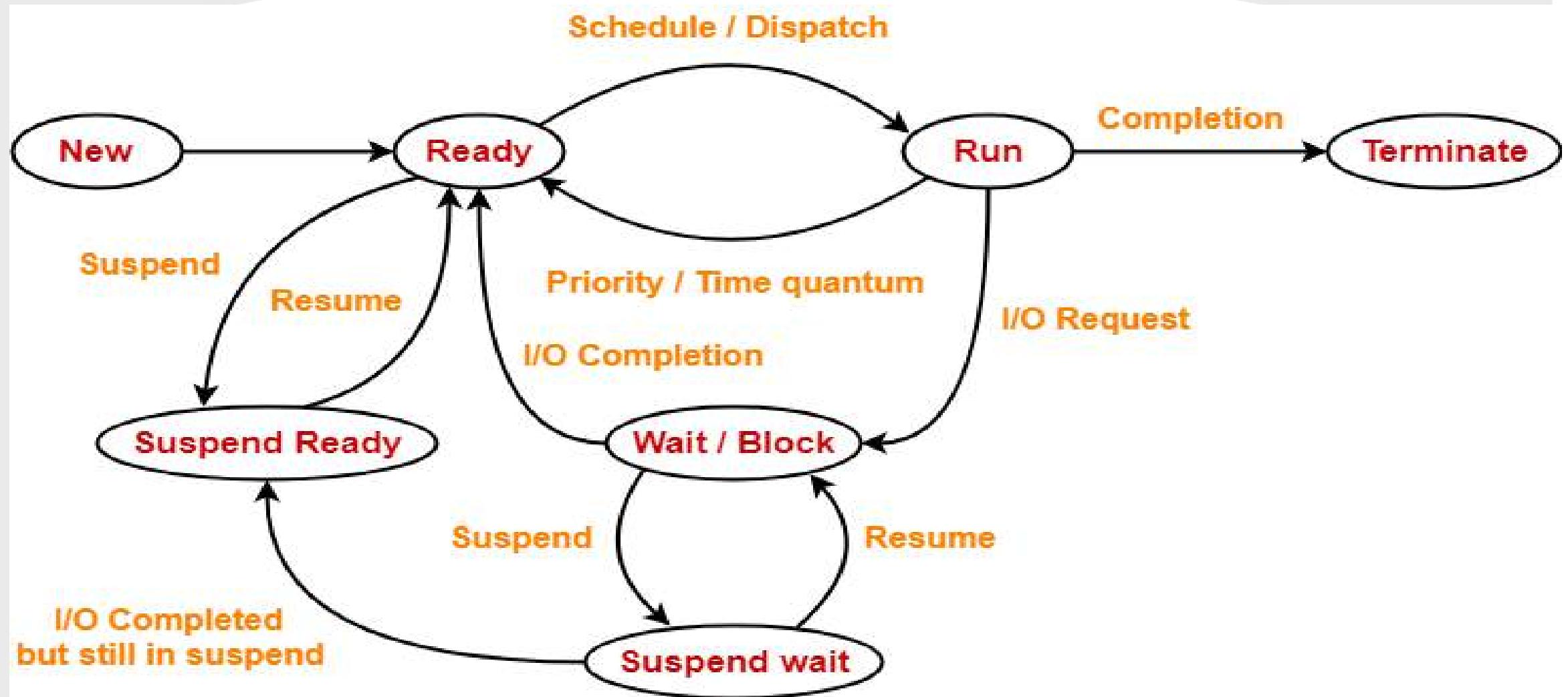
To ac

CK HERE ↗

## Process States

- A Process changes states as it executes. The state of a process is defined in parts by the current activity of that process. A process may be in one of the following states:
  - New: The process is being created.
  - Running: Instructions are being executed.
  - Waiting (Blocked): The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - Ready: The process is waiting to be assigned to a processor.
  - Terminated: The process has finished execution.





Process State Diagram

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q How many states can a process be in? (NET-DEC-2007)**

- a) 2**
- b) 3**
- c) 4**

**d) 5**

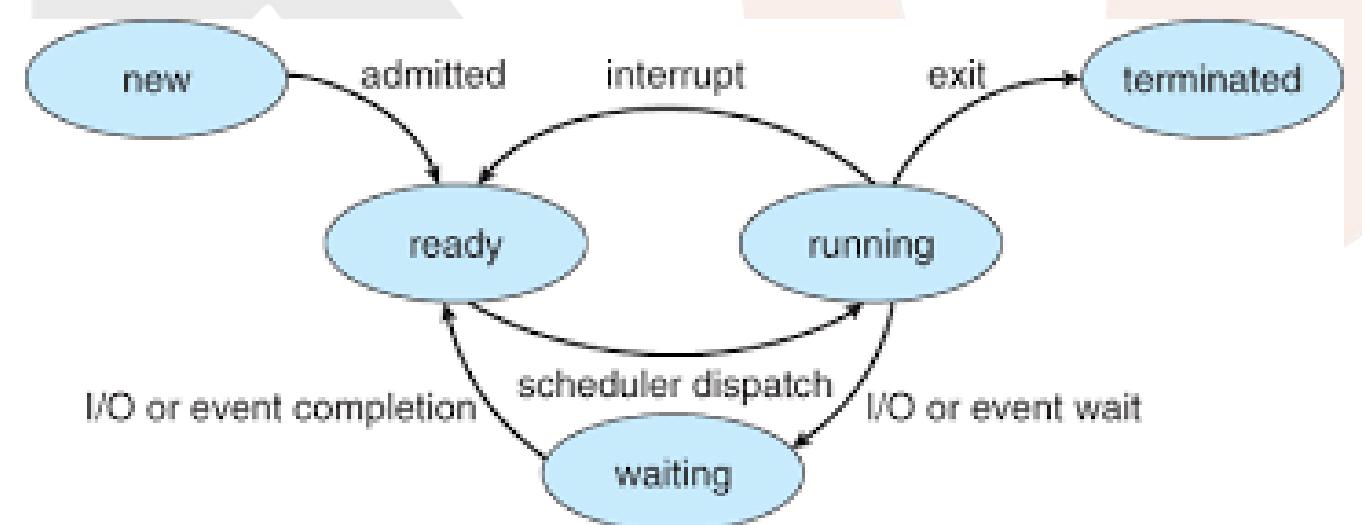


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q What is the minimum and maximum number of processes that can be in the ready, run, and blocked states, if total number of process is n?

	Min	Max
Ready		
Run		
Block		



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

**Q** On a system with  $P$  CPUs and  $n$  processes, what is the minimum and maximum number of processes that can be in the ready, run, and blocked states?

	<b>Min</b>	<b>Max</b>
<b>Ready</b>		
<b>Run</b>		
<b>Block</b>		

Q The state of a process after it encounters an I/O instruction is (ISRO 2013)

- a) ready
- b) blocked
- c) idle
- d) running



Q There are three processes in the ready queue. When the currently running process requests for I/O how many process switches take place? (ISRO 2011)

a) 1

b) 2

c) 3

d) 4



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q What is the ready state of a process?

- a) when process is scheduled to run after some execution
- b) when process is using the CPU
- c) when process is unable to run until some task has been completed
- d) none of the mentioned

## Q A task in a blocked state

- a) is executable
- b) is running
- c) must still be placed in the run queues
- d) is waiting for some temporarily unavailable resources



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Which combination of the following features will suffice to characterize an OS as a multi-programmed OS? **(GATE-2002) (2 Marks)**

**(a)** More than one program may be loaded into main memory at the same time for execution.

**(b)** If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

**(c)** If the execution of program terminates, another program is immediately scheduled for execution.

**(A)** a

**(B)** a and b

**(C)** a and c

**(D)** a, b and c

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

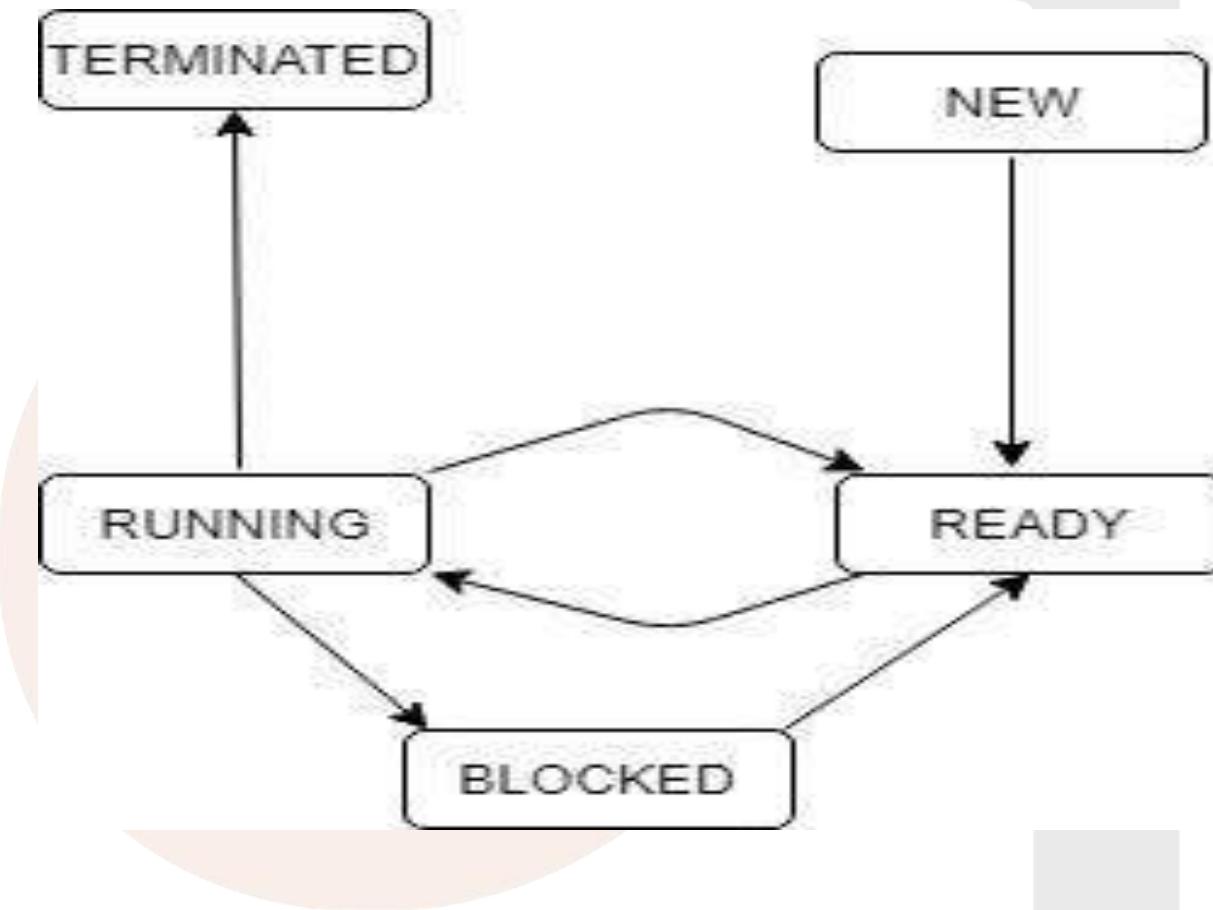
Q The process state transition diagram in below figure is representative of Untitled Diagram (GATE-1996) (1 Marks)

a) a batch operating system

b) an operating system with a preemptive scheduler

c) an operating system with a non-preemptive scheduler

d) a uni-programmed operating system

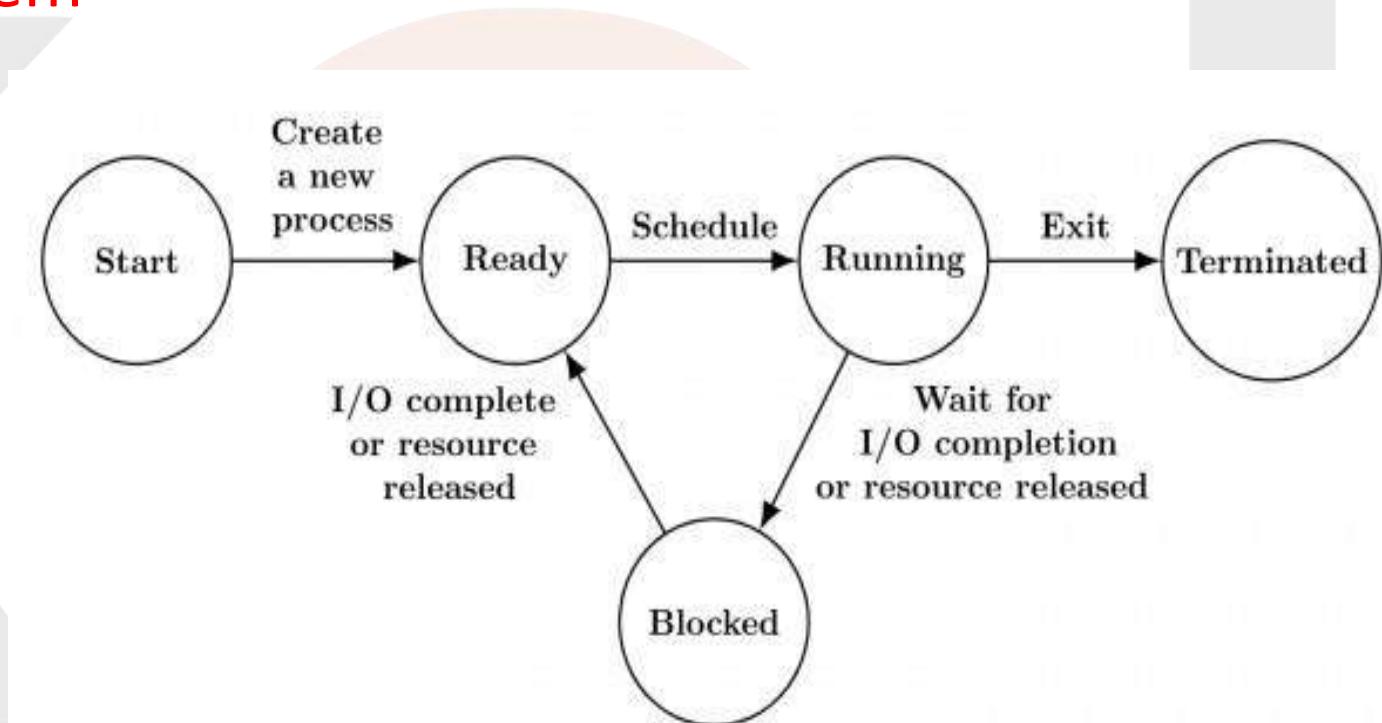


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** The process state transition diagram of an operating system is as given below. Which of the following must be FALSE about the above operating system? (GATE-2006) (1 Marks)

- a) It is a multiprogram operating system
- b) It uses preemptive scheduling
- c) It uses non-preemptive scheduling
- d) It is a multi-user operating system



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

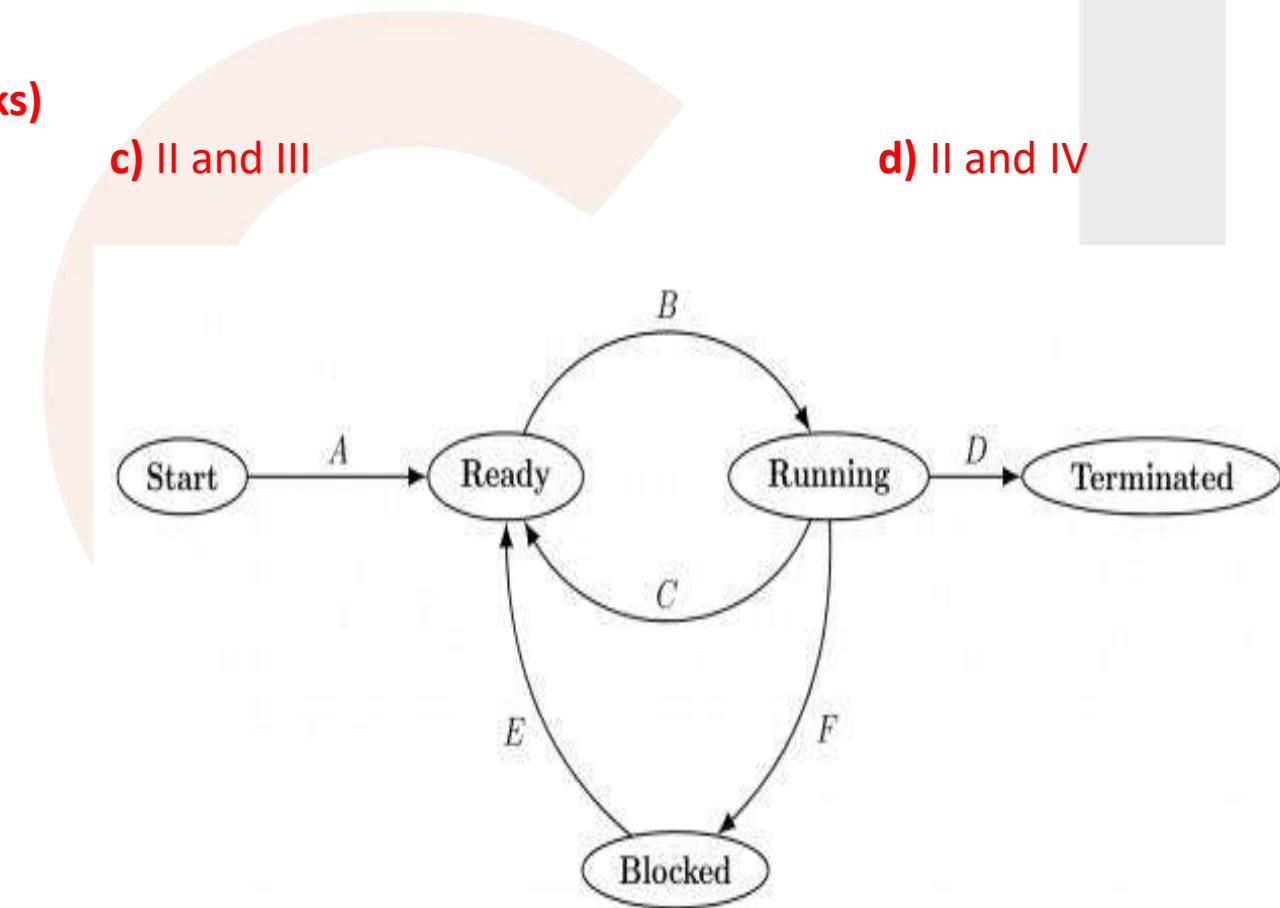
Q In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state:

Now consider the following statements:

- I) If a process makes a transition D, it would result in another process making transition A immediately.
- II) A process P2 in blocked state can make transition E while another process P1 is in running state.
- III) The OS uses preemptive scheduling.
- IV) The OS uses non-preemptive scheduling.

Which of the above statements are TRUE? (GATE-2009) (2 Marks)

- a) I and II
- b) I and III
- c) II and III
- d) II and IV



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A computer handles several interrupt sources of which the following are relevant for this question.

- Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
- Interrupt from Mouse (raises interrupt if the mouse is moved or a button is pressed)
- Interrupt from Keyboard (raises interrupt when a key is pressed or released)
- Interrupt from Hard Disk (raises interrupt when a disk read is completed)

Which one of these will be handled at the HIGHEST priority? **(GATE - 2011) (1 Marks)**

**(A)** Interrupt from Hard Disk

**(B)** Interrupt from Mouse

**(C)** Interrupt from Keyboard

**(D)** Interrupt from CPU temperature sensor

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Q Match the following: (NET-SEP-2013)

List – I	List - II
Process state	Reason for transition
a) Ready → Running	i. Request made by the process is satisfied or an event for which it was waiting occurs.
b) Blocked → Ready	ii. Process wishes to wait for some action by another process.
c) Running → Blocked	iii. The process is dispatched.
d) Running → Ready	iv. The process is pre-empted.

	a	b	c	d
a)	iii	i	ii	iv
b)	iv	i	iii	ii
c)	iv	iii	i	ii
d)	iii	iii	ii	i

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

Q Which is the correct definition of a valid process transition in an operating system? (NET-JUNE-2005)

a) Wake up : Ready → Running

b) Dispatch : Ready → Running

c) Block: Ready → Running

d) Timer run out: Ready → Blocked

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

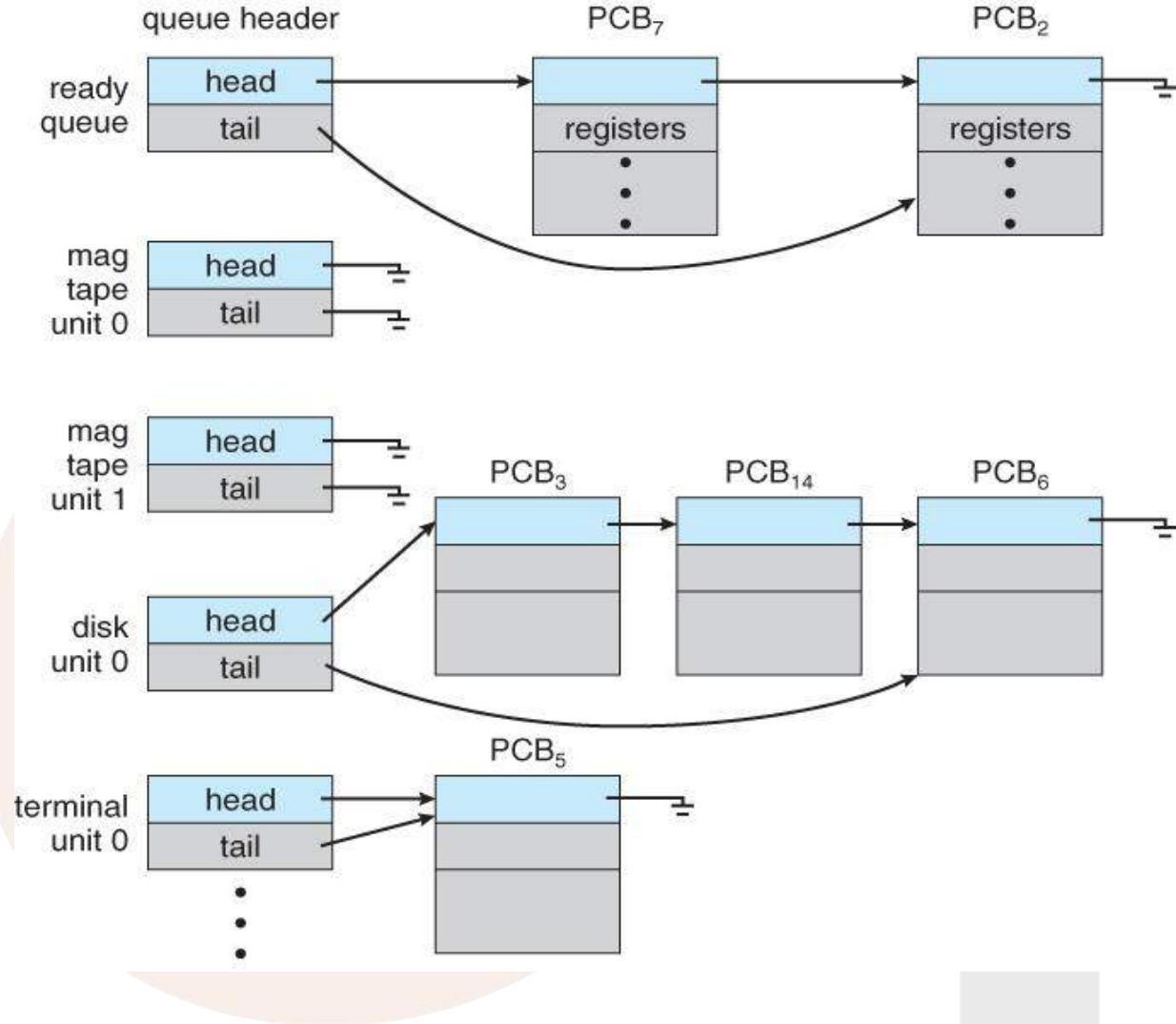


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for execution on the CPU. Note: For a single-processor system, there will never be more than one running process.

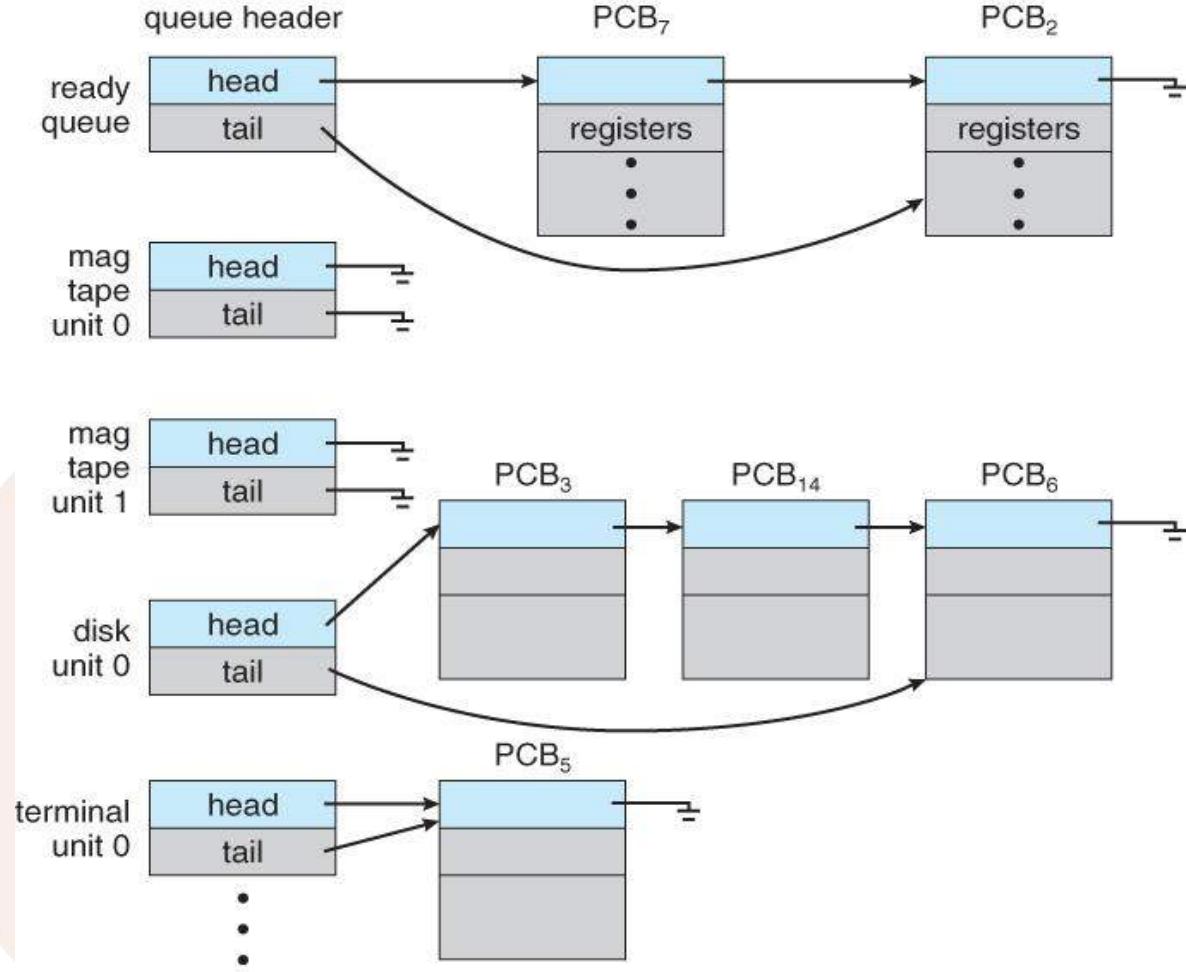
- Scheduling Queues - As processes enter the system, they are put into a job queue, which consists of all processes in the system.
- The processes that are residing in main memory and are ready to execute are kept on a list called the ready queue. This queue is generally stored as a linked list.
- A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

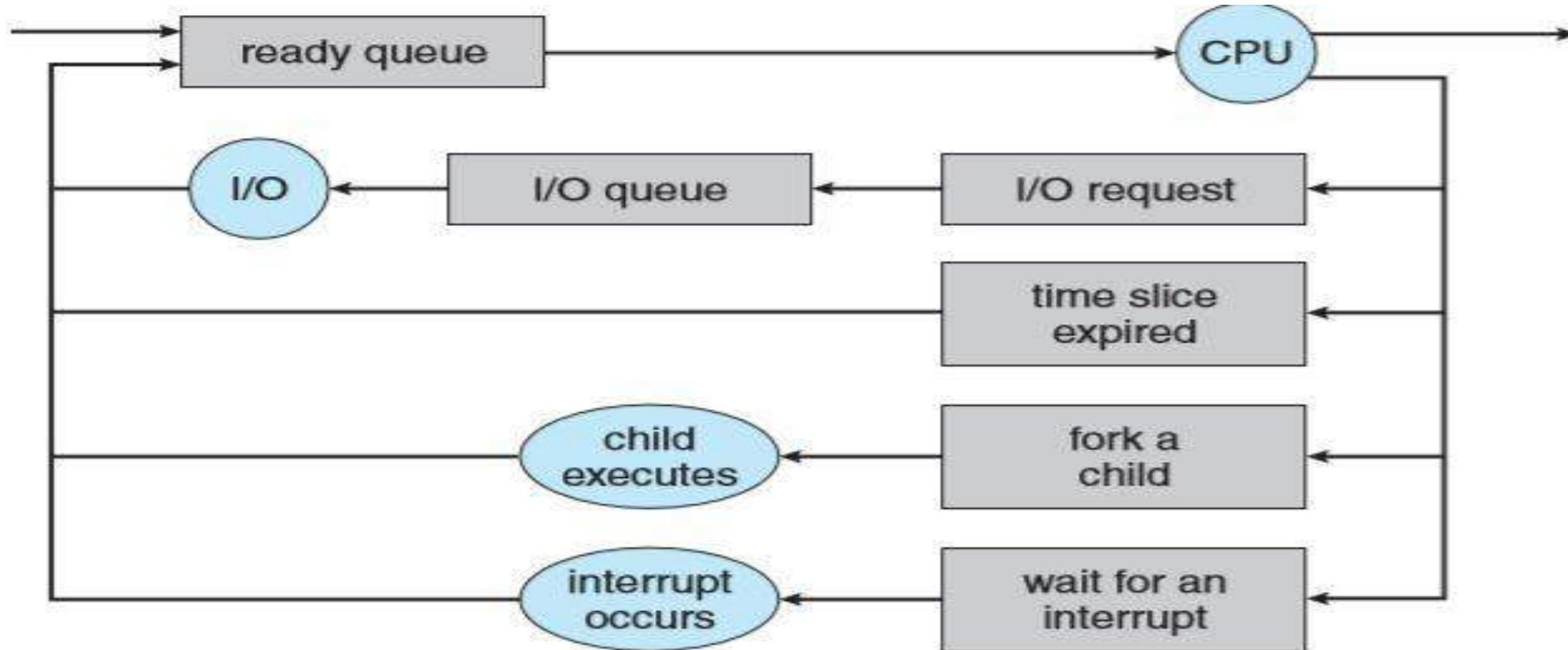
- The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.
- Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.



To access all paid content get **KG Prime** at ₹25/day

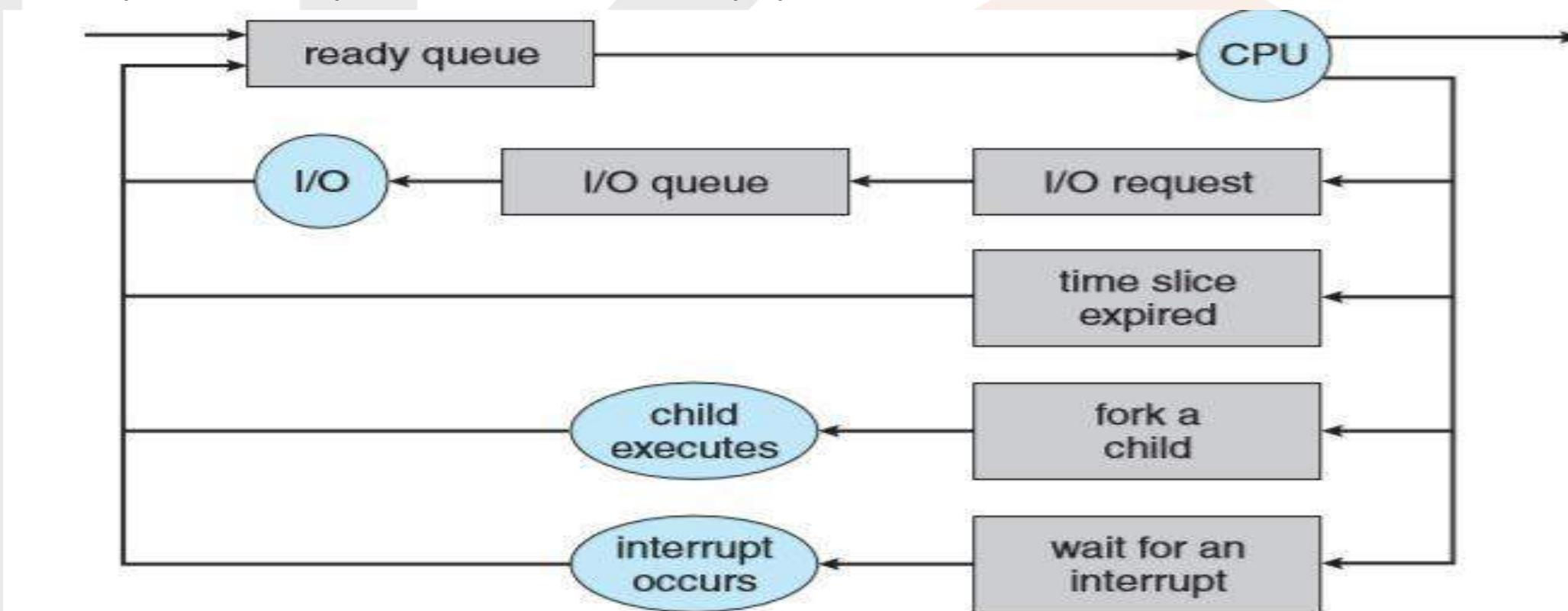
**CLICK HERE** ↗

- Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues. A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched.



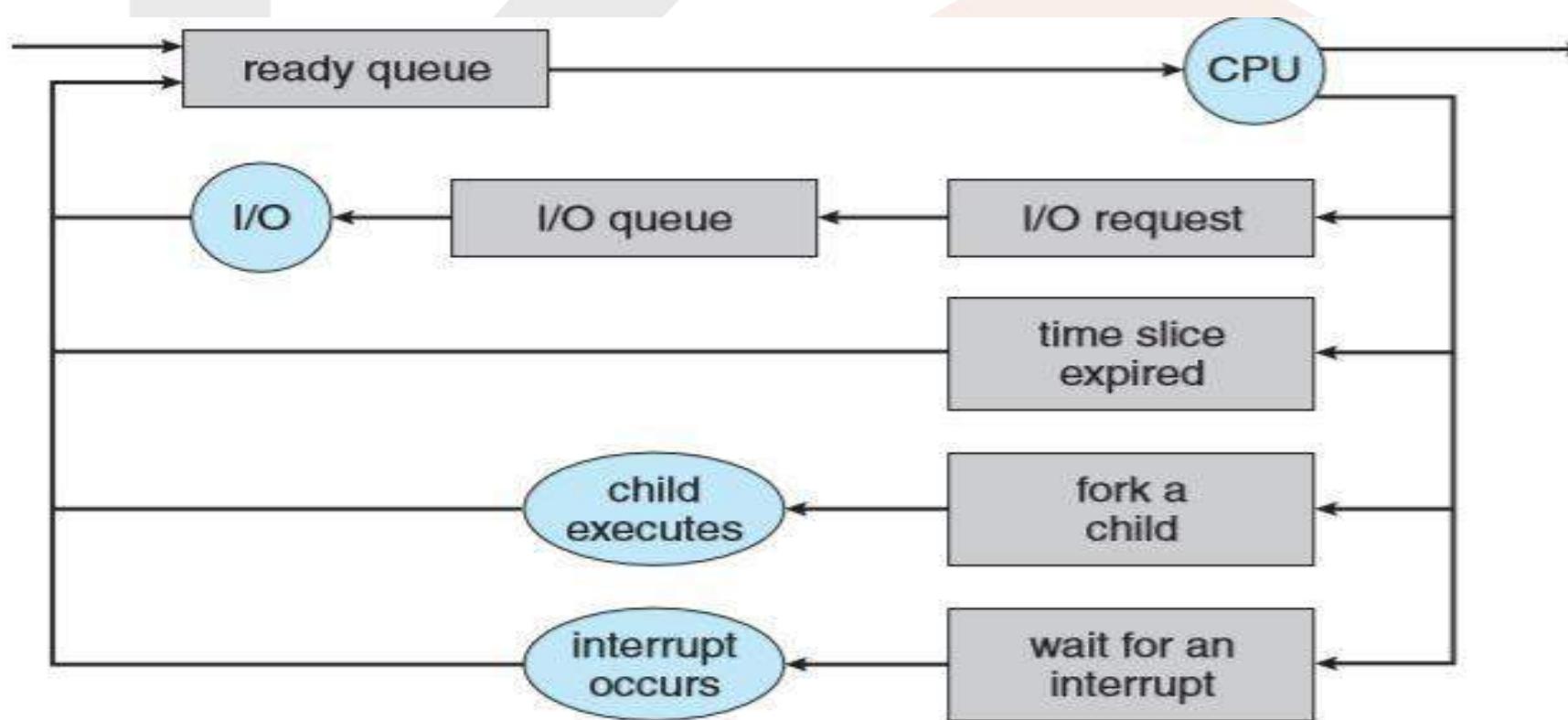
Queueing-diagram representation of process scheduling.

- Once the process is allocated the CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new child process and wait for the child's termination.
  - Time slice expired, the process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



Queueing-diagram representation of process scheduling.

- In the first two cases the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.
- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



Queueing-diagram representation of process scheduling.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

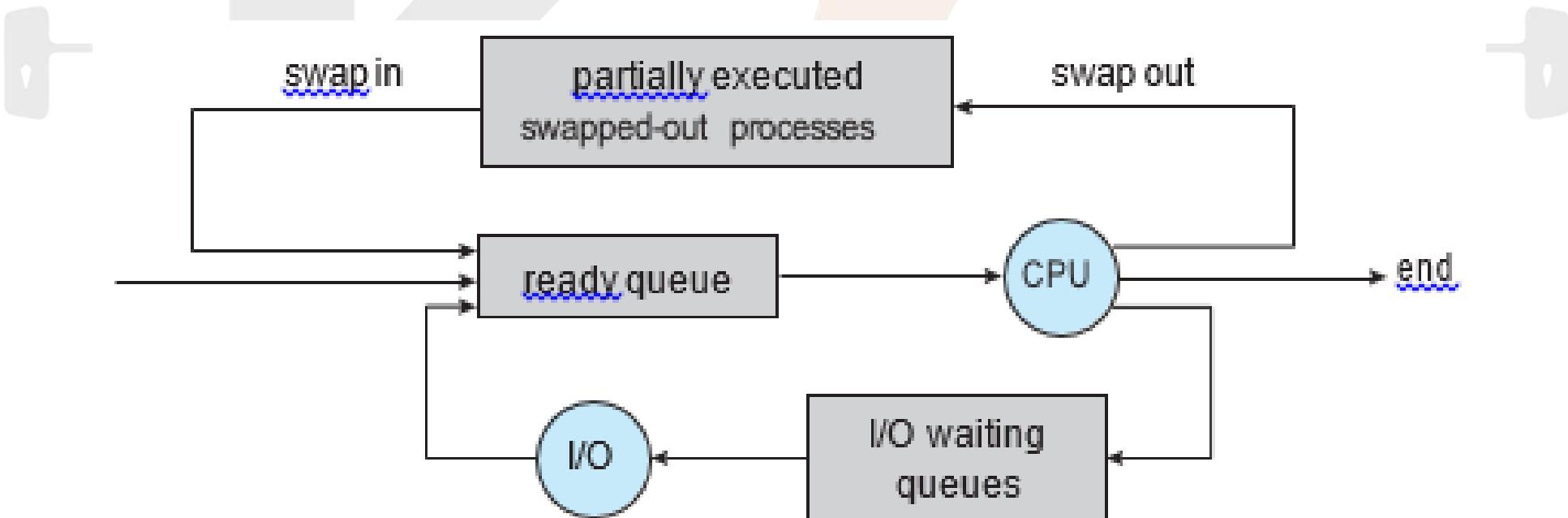
# Schedulers

- **Schedulers**: A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.
- **Types of Schedulers**
  - **Long Term Schedulers (LTS)/Spooler**: In multiprogramming os, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.
  - **Short Term Scheduler (STS)**: The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.

- **Difference between LTS and STS** - The primary distinction between these two schedulers lies in frequency of execution.
- The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Often, the short-term scheduler executes at least once every 100 milliseconds.
- Because of the short time between executions, the short-term scheduler must be fast. The long-term scheduler on the other hand executes much less frequently; minutes may separate the creation of one new process and the next.

## Degree of Multiprogramming - The number of processes in memory is known as Degree of Multiprogramming.

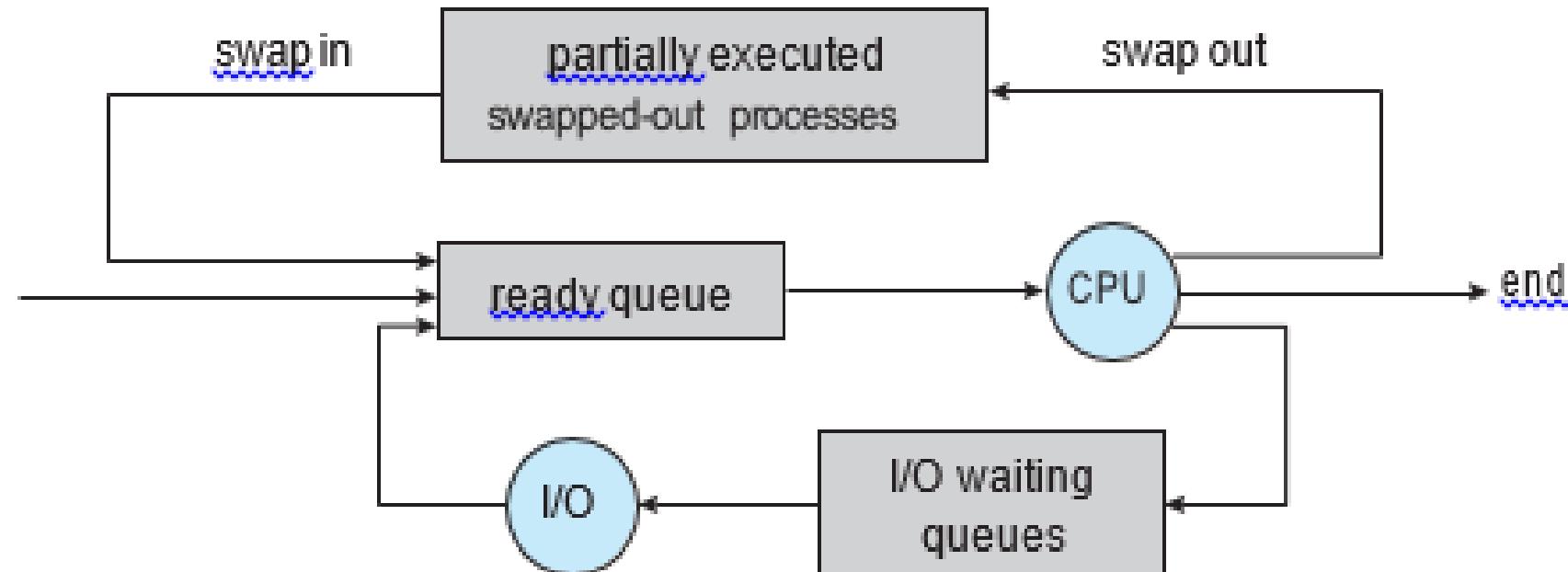
- The long-term scheduler controls the degree of multiprogramming as it is responsible for bringing in the processes to main memory.
- If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. So, this means the long-term scheduler may need to be invoked only when a process leaves the system. Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.



**Medium-term scheduler:** The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.

Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called swapping.

The process is swapped out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.



- **Dispatcher** - The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.
- This function involves the following: Switching context, switching to user mode, jumping to the proper location in the user program to restart that program.
- The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Q A software to create a Job Queue is called..... (NET-DEC-2006)

a) Linkage editor

b) Interpreter

c) Driver

d) Spooler

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A special software that is used to create a job queue is called (NET-JUNE-2011)

(A) Drive

(B) Spooler



(C) Interpreter



(D) Linkage editor



Q Which module gives control of the CPU to the process selected by the short - term scheduler? (NET-NOV-2017)

a) Dispatcher



c) Scheduler



b) Interrupt



d) Threading



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## CPU Bound and I/O Bound Processes

- A process execution consists of a cycle of CPU execution or wait and i/o execution or wait.
- Normally a process alternates between two states.
- Process execution begin with the CPU burst that may be followed by a i/o burst, then another CPU and i/o burst and so on.
- Eventually in the last will end up on CPU burst. So, process keep switching between the CPU and i/o during execution.

- **I/O Bound Processes**: An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
- **CPU Bound Processes**: A CPU-bound process, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.
- If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.
- Similarly, if all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
- So, to have the best system performance LTS needs to select a good combination of I/O and CPU Bound processes.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Context Switch

- When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. Context-switch time is pure overhead, because the system does no useful work while switching.
- Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). A typical speed is a few milliseconds.

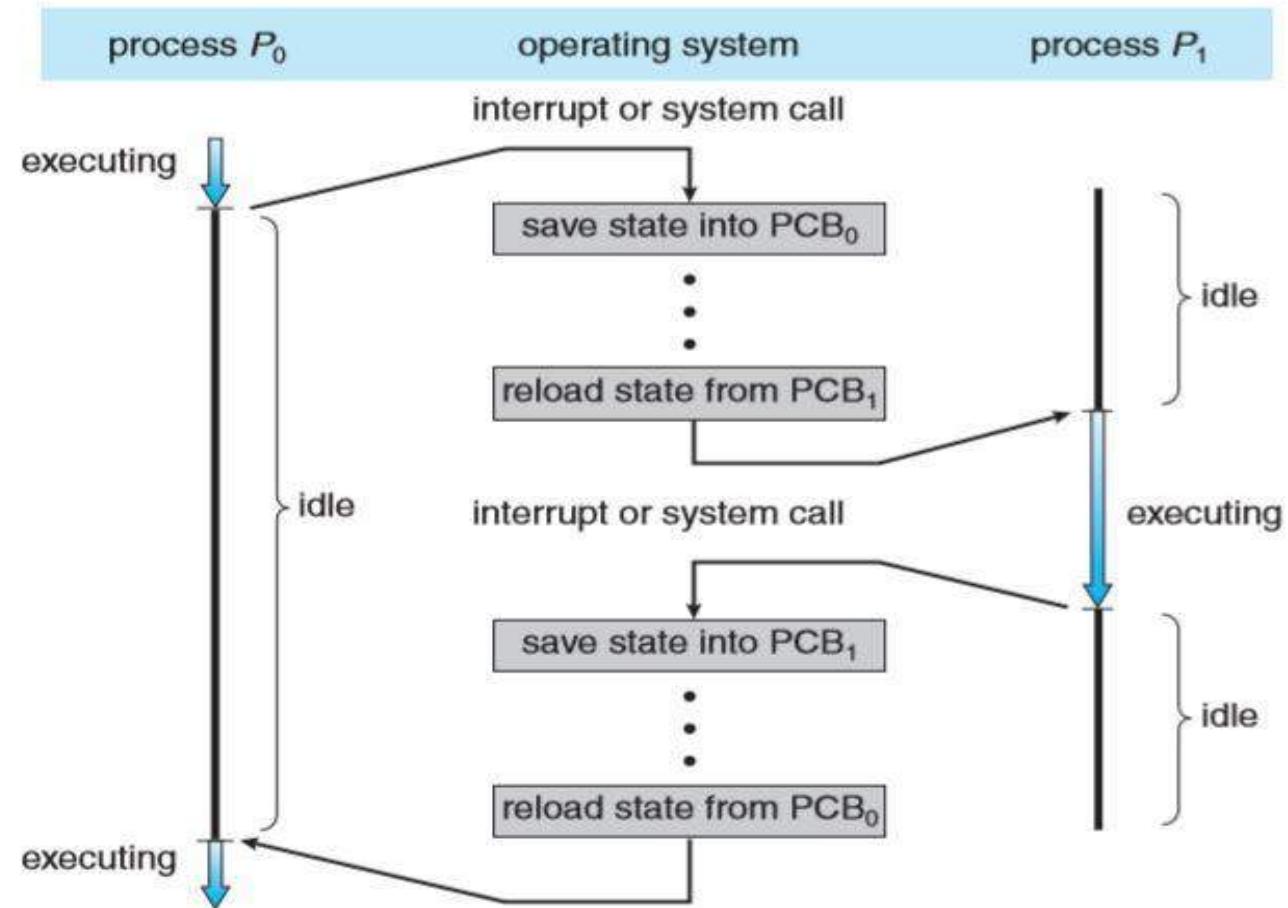


Diagram showing CPU switch from process to process.

**Q Which of the following does not interrupt a running process? (GATE-2001) (2 Marks)**

**(A) A device**

**(B) Timer**

**(C) Scheduler process**

**(D) Power failure**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Which is the correct definition of a valid process transition in an operating system?

a) Wake up: ready → running

b) Dispatch: ready → running

c) Block: ready → running

d) Timer runout: ready → running

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A process stack does not contain

a) function parameters

b) local variables



c) return addresses



d) PID of child process

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Loading operating system from secondary memory to primary memory is called..... (NET-DEC-2006)

(A) Compiling

(B) Booting

(C) Refreshing

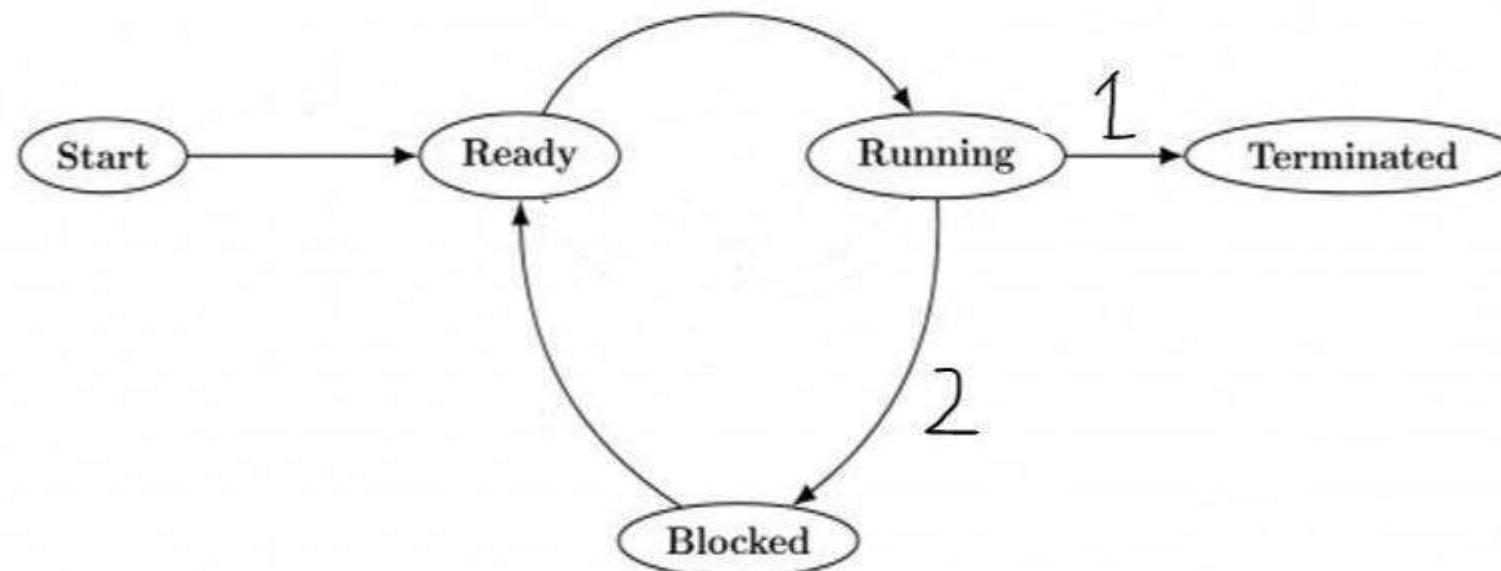
(D) Reassembling

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Type of scheduling

- **Non-Pre-emptive:** Under Non-Pre-emptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU willingly.
- A process will leave the CPU only
  1. When a process completes its execution (Termination state)
  2. When a process wants to perform some i/o operations(Blocked state)

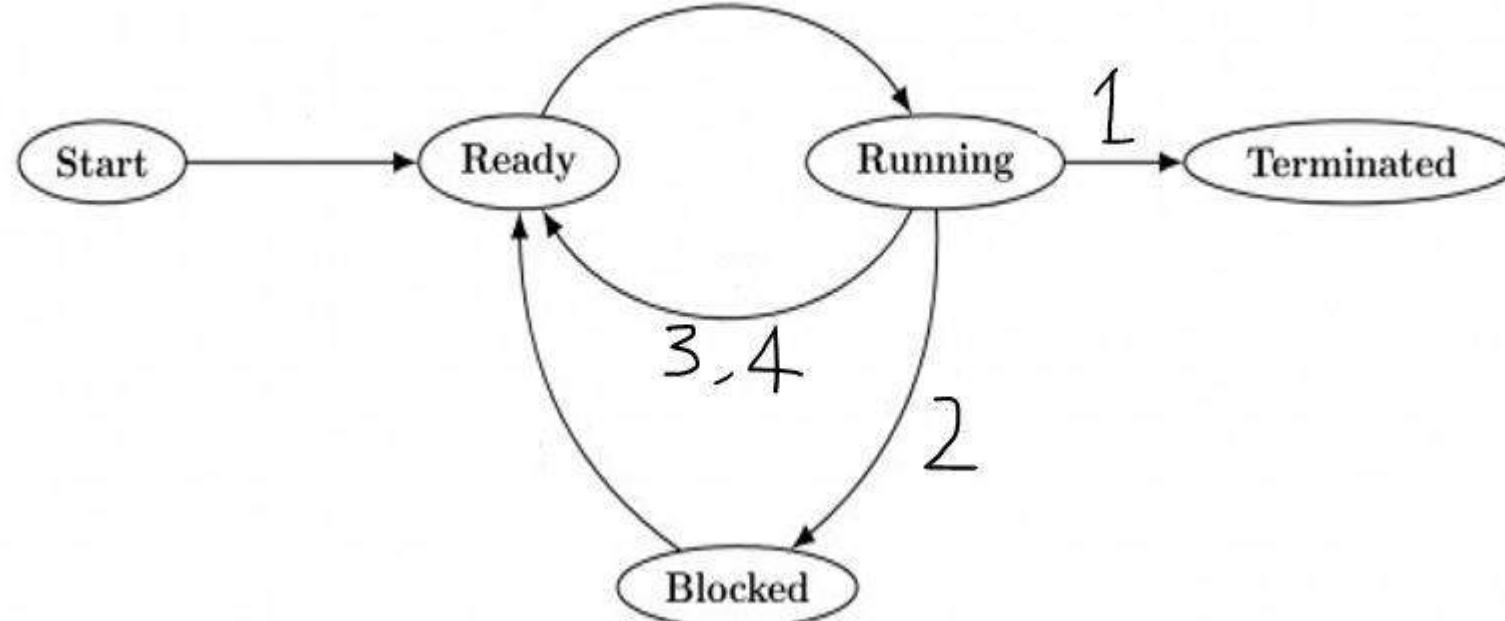


To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

# Pre-emptive

- Under Pre-emptive scheduling, once the CPU has been allocated to a process, A process will leave the CPU willingly or it can be forced out. So it will leave the CPU
  - When a process completes its execution
  - When a process leaves CPU voluntarily to perform some i/o operations
  - If a new process enters in the ready states (new, waiting), in case of high priority
  - When process switches from running to ready state because of time quantum expire.



To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Scheduling criteria** - Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another. So, in order to efficiently select the scheduling algorithms following criteria should be taken into consideration:

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **CPU utilization:** Keeping the CPU as busy as possible.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- **Response Time**: Is the time it takes to start responding, not the time it takes to output the response.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Note: The CPU-scheduling algorithm does not affect the amount of time during which a process executes I/O; it affects only the amount of time that a process spends waiting in the ready queue.
- It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Terminology

- **Arrival Time (AT):** Time at which process enters a ready state.
- **Burst Time (BT):** Amount of CPU time required by the process to finish its execution.
- **Completion Time (CT):** Time at which process finishes its execution.
- **Turn Around Time (TAT):** Completion Time (CT) – Arrival Time (AT), Waiting Time + Burst Time (BT)
- **Waiting Time:** Turn Around Time (TAT) – Burst Time (BT)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## FCFS (FISRT COME FIRST SERVE)

- FCFS is the simplest scheduling algorithm, as the name suggest, the process that requests the CPU first is allocated the CPU first.
- Implementation is managed by FIFO Queue.
- It is always non pre-emptive in nature.



Get KG Prime at ₹25/day

CLICK HERE ↗

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	2	4			
P <sub>1</sub>	1	2			
P <sub>2</sub>	0	3			
P <sub>3</sub>	4	2			
P <sub>4</sub>	3	1			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	0	3			
P <sub>1</sub>	2	2			
P <sub>2</sub>	6	4			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	6	4			
P <sub>1</sub>	2	5			
P <sub>2</sub>	3	3			
P <sub>3</sub>	1	1			
P <sub>4</sub>	4	2			
P <sub>5</sub>	5	6			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Consider the following table of arrival time and burst time for three processes  $P_0$ ,  $P_1$ ,  $P_2$ ,  $P_3$ ,  $P_4$  What is the average Waiting Time and Turnaround Time for the five processes?

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
$P_0$	0	14			
$P_1$	1	3			
$P_2$	2	1			
$P_3$	3	2			
$P_4$	4	5			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Advantage

- Easy to understand, and can easily be implemented using Queue data structure.
- Can be used for Background processes where execution is not urgent.

P. No	AT	BT	TAT=CT-AT	WT=TAT -BT
$P_0$	0	100		
$P_1$	1	2		
Average				

P. No	AT	BT	TAT=CT-AT	WT=TAT -BT
$P_0$	1	100		
$P_1$	0	2		
Average				

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Convoy Effect

- If the smaller process have to wait more for the CPU because of Larger process then this effect is called Convoy Effect, it result into more average waiting time.
- Solution, smaller process have to be executed before longer process, to achieve less average waiting time.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Disadvantage

- FCFS suffers from convoy which means smaller process have to wait larger process, which result into large average waiting time.
- The FCFS algorithm is thus particularly troublesome for time-sharing systems (due to its non-pre-emptive nature), where it is important that each user get a share of the CPU at regular intervals.
- Higher average waiting time and TAT compared to other algorithms.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Shortest Job First (SJF)(non-pre-emptive)

## Shortest Remaining Time First (SRTF)/ (Shortest Next CPU Burst) (Pre-emptive)

- Whenever we make a decision of selecting the next process for CPU execution, out of all available process, CPU is assigned to the process having smallest burst time requirement. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If there is a tie, FCFS is used to break tie.

- It supports both version non-pre-emptive and pre-emptive (purely greedy approach)
- In Shortest Job First (SJF)(non-pre-emptive) once a decision is made and among the available process, the process with the smallest CPU burst is scheduled on the CPU, it cannot be pre-empted even if a new process with the smaller CPU burst requirement then the remaining CPU burst of the running process enter in the system.

- In Shortest Remaining Time First (SRTF) (Pre-emptive) whenever a process enters in ready state, again we make a scheduling decision weather, this new process with the smaller CPU burst requirement then the remaining CPU burst of the running process and if it is the case then the running process is pre-empted and new process is scheduled on the CPU.
- This version (SRTF) is also called optimal is it guarantee minimal average waiting time.

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	1	7			
P <sub>1</sub>	2	5			
P <sub>2</sub>	3	1			
P <sub>3</sub>	4	2			
P <sub>4</sub>	5	8			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	0	6			
P <sub>1</sub>	1	4			
P <sub>2</sub>	2	3			
P <sub>3</sub>	3	1			
P <sub>4</sub>	4	2			
P <sub>5</sub>	5	1			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	3	4			
P <sub>1</sub>	4	2			
P <sub>2</sub>	5	1			
P <sub>3</sub>	2	6			
P <sub>4</sub>	1	8			
P <sub>5</sub>	2	4			
Average					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider an arbitrary set of CPU-bound processes with unequal CPU burst lengths submitted at the same time to a computer system. Which one of the following process scheduling algorithms would minimize the average waiting time in the ready queue? **(GATE - 2016) (1 Marks)**

- (a) Shortest remaining time first
- (b) Round-robin with time quantum less than the shortest CPU burst
- (c) Uniform random
- (d) Highest priority first with priority proportional to CPU burst length

Q For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time? (GATE-2015) (2 Marks)

Process	Arrival Time	Processing Time
A	0	3
B	1	6
C	4	4
D	6	2

- (A) First Come First Serve
- (B) Non-pre-emptive Shortest Job First
- (C) Shortest Remaining Time
- (D) Round Robin with Quantum value two

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below: (GATE - 2019) (2 Marks)

Process	Arrival Time	CPU Time	CT	TAT	WT
$P_1$	0	3			
$P_2$	1	1			
$P_3$	3	3			
$P_4$	4	$Z$			

These processes are run on a single processor using pre-emptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of  $Z$  is \_\_\_\_.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Advantage

- Pre-emptive version guarantees minimal average waiting time so some time also referred as optimal algorithm.
- Provide a standard for other algo in terms of average waiting time
- Provide better average response time compare to FCFS

## Disadvantage

- This algo cannot be implemented as there is no way to know the length of the next CPU burst.
- Here process with the longer CPU burst requirement goes into starvation.
- No idea of priority, longer process has poor response time.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Consider the following three processes with the arrival time and CPU burst time given in milliseconds: (NET-July-2018)

Process	Arrival Time	Burst Time
$P_1$	0	7
$P_2$	1	4
$P_3$	2	8

The Gantt Chart for pre-emptive SJF scheduling algorithm is \_\_\_\_\_.

a)  $P_1(0-7) | P_2(7-13) | P_3(13-21)$

b)  $P_1(0-1) | P_2(1-5) | P_1(5-11) | P_3(11-19)$

c)  $P_1(0-7) | P_2(7-11) | P_3(11-19)$

d)  $P_2(0-4) | P_3(4-12) | P_1(12-19)$

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Consider the following four processes with the arrival time and length of CPU burst given in milliseconds: (NET-Nov-2018)

Process	Arrival Time	Burst Time	CT	TAT	WT
$P_1$	0	8			
$P_2$	1	4			
$P_3$	2	9			
$P_4$	3	5			

The average waiting time for pre-emptive SJF scheduling algorithm is \_\_\_\_\_.

a) 6.5      b) 7.5      c) 6.75      d) 7.75

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Consider the following set of processes, with the arrival times and the CPU-burst times given in milliseconds (GATE - 2017) (2 Marks)

Process	Arrival Time	CPU Time	CT	TAT	WT
$P_1$	0	5			
$P_2$	1	3			
$P_3$	2	3			
$P_4$	4	1			

What is the average turnaround time for these processes with the pre-emptive shortest remaining processing time first (SRPT) algorithm?

- (A) 5.50      (B) 5.75      (C) 6.00      (D) 6.25

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the following CPU processes with arrival times (in milliseconds) and length of CPU bursts (in milliseconds) as given below: (GATE - 2017) (1 Marks)

Process	Arrival Time	CPU Time	CT	TAT	WT
P <sub>1</sub>	0	7			
P <sub>2</sub>	3	3			
P <sub>3</sub>	5	5			
P <sub>4</sub>	6	2			

If the pre-emptive shortest remaining time first scheduling algorithm is used to schedule the processes, then the average waiting time across all processes is \_\_\_\_\_ milliseconds.

**Q** Consider three CPU intensive processes  $P_1$ ,  $P_2$ ,  $P_3$  which require 20, 10 and 30 units of time, arrive at times 1, 3 and 7 respectively. Suppose operating system is implementing Shortest Remaining Time first (pre-emptive scheduling) algorithm, then \_\_\_\_\_ context switches are required (suppose context switch at the beginning of Ready queue and at the end of Ready queue are not counted). (NET-Aug-2016)

Process	Arrival Time	Burst Time
$P_1$	1	20
$P_2$	3	10
$P_3$	7	30

- a) 3
- b) 2
- c) 4
- d) 5

Q Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is pre-emptive shortest remaining-time first.

Process	Arrival Time	CPU Time	CT	TAT	WT
P <sub>1</sub>	0	10			
P <sub>2</sub>	3	6			
P <sub>3</sub>	7	1			
P <sub>4</sub>	8	3			

The average turnaround time of these processes is \_\_\_\_\_ milliseconds. (GATE - 2016) (2 Marks)

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Consider the following set of processes that need to be scheduled on a single CPU. All the times are given in milliseconds? (GATE-2014) (2 Marks)

Process Name	Arrival Time	Execution Time	CT	TAT	WT
A	0	6			
B	3	2			
C	5	4			
D	7	6			
E	10	3			

Using the *shortest remaining time first* scheduling algorithm, the average process turnaround time (in msec) is \_\_\_\_\_.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** An operating system uses shortest remaining time first scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds) (GATE-2014) (2 Marks)

Process	Arrival Time	Burst Time	CT	TAT	WT
$P_1$	0	12			
$P_2$	2	4			
$P_3$	3	6			
$P_4$	8	5			

The average waiting time (in milliseconds) of the processes is \_\_\_\_\_.

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the following table of arrival time and burst time for three processes  $P_0$ ,  $P_1$  and  $P_2$ .

Process	Arrival Time	Burst Time	CT	TAT	WT
$P_0$	0	9			
$P_1$	1	4			
$P_2$	2	9			

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?

**(GATE-2011) (2 Marks)**

- (A) 5.0 ms      (B) 4.33 ms      (C) 6.33      (D) 7.33

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** An operating system uses Shortest Remaining Time first (SRT) process scheduling algorithm. Consider the arrival times and execution times for the following processes:

Process	Arrival Time	CPU Time	CT	TAT	WT
P <sub>1</sub>	0	20			
P <sub>2</sub>	15	25			
P <sub>3</sub>	30	10			
P <sub>4</sub>	45	15			

What is the total waiting time for process P<sub>3</sub>? (GATE - 2007) (2 Marks)



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

**Q** Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end? **(GATE-2006) (1 Marks)**

Process	Arrival Time	CPU Time
P <sub>1</sub>	0	10
P <sub>2</sub>	2	20
P <sub>3</sub>	6	30



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- As SJF is not implementable, we can use the one technique where we try to predict the CPU burst of the next coming process.
- The method is used as exponential averaging technique, where we consider the previous value and previous prediction.

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

$$\Tau_{(n+1)} = \alpha t_n + (1 - \alpha) \Tau_n.$$



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Process	t	tau
P <sub>1</sub>	10	20
P <sub>2</sub>	12	
P <sub>3</sub>	14	
P <sub>4</sub>		

$$\text{Tau}_{(n+1)} = \alpha t_n + (1 - \alpha) \text{tau}_n$$

- This idea is also more of theoretical importance as most of the time the burst requirement of the coming process may vary by a large extent and if the burst time requirement of all the process is approximately same then there is no advantage of using this scheme.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Priority scheduling

- Here a priority is associated with each process. At any instance of time out of all available process, CPU is allocated to the process which possess highest priority (may be higher or lower number).
- Tie is broken using FCFS order. No importance to senior or burst time. It supports both non-pre-emptive and pre-emptive versions.



- In Priority (non-pre-emptive) once a decision is made and among the available process, the process with the highest priority is scheduled on the CPU, it cannot be pre-empted even if a new process with higher priority more than the priority of the running process enter in the system.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- In Priority (pre-emptive) once a decision is made and among the available process, the process with the highest priority is scheduled on the CPU.
- if it a new process with priority more than the priority of the running process enter in the system, then we do a context switch and the processor is provided to the new process with higher priority.

- Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095. There is no general agreement on whether 0 is the highest or lowest priority, it can vary from systems to systems.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	AT	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
$P_0$	1	4	4			
$P_1$	2	2	5			
$P_2$	2	3	7			
$P_3$	3	5	8(H)			
$P_4$	3	1	5			
$P_5$	4	2	6			
Average						

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	AT	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
P <sub>0</sub>	0	50	4			
P <sub>1</sub>	20	20	1(h)			
P <sub>2</sub>	40	100	3			
P <sub>3</sub>	60	40	2			
Average						

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	AT	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
P <sub>0</sub>	1	4	5			
P <sub>1</sub>	2	5	2			
P <sub>2</sub>	3	6	6			
P <sub>3</sub>	0	1	4			
P <sub>4</sub>	4	2	1			
P <sub>5</sub>	5	3	8(H)			
Average						

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Advantage**

- Gives a facility specially to system process.
- Allow us to run important process even if it is a user process.

- **Disadvantage**

- Here process with the smaller priority may starve for the CPU
- No idea of response time or waiting time.

- Note: - Specially use to support system process or important user process
- Ageing: - a technique of gradually increasing the priority of processes that wait in the system for long time. E.g. priority will increase after every 10 mins

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival	Burst Time	Priority	CT	TAT	WT
	Time					
$P_1$	0	11	2			
$P_2$	5	28	0			
$P_3$	12	2	3			
$P_4$	2	10	1			
$P_5$	9	16	4			

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is \_\_\_\_\_. (GATE-2017) (2 Marks)

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Some of the criteria for calculation of priority of a process are:

- a. Processor utilization by an individual process.
- b. Weight assigned to a user or group of users.
- c. Processor utilization by a user or group of processes

In fair share scheduler, priority is calculated based on: **(NET-Jan-2017)**

- a) only (a) and (b)
- b) only (a) and (c)
- c) (a), (b) and (c)
- d) only (b) and (c)

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider a uniprocessor system executing three tasks  $T_1$ ,  $T_2$  and  $T_3$ , each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of 3, 7 and 20 milliseconds, respectively. The priority of each task is the inverse of its period and the available tasks are scheduled in order of priority, with the highest priority task scheduled first. Each instance of  $T_1$ ,  $T_2$  and  $T_3$  requires an execution time of 1, 2 and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1st milliseconds and task preemptions are allowed, the first instance of  $T_3$  completes its execution at the end of \_\_\_\_\_ milliseconds. (GATE-2015) (2 Marks)

Process	Arrival Time	Burst Time

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** The problem of indefinite blockage of low-priority jobs in general priority scheduling algorithm can be solved using: **(NET-Dec-2012)**

- a) Parity bit**
- b) Aging**
- c) Compaction**
- d) Timer**



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** We wish to schedule three processes  $P_1$ ,  $P_2$  and  $P_3$  on a uniprocessor system. The priorities, CPU time requirements and arrival times of the processes are as shown below.

Process	Priority	CPU time	Arrival time
$P_1$	10(highest)	20 sec	00:00:05
$P_2$	9	10 sec	00:00:03
$P_3$	8 (lowest)	15 sec	00:00:00

We have a choice of pre-emptive or non-pre-emptive scheduling. In pre-emptive scheduling, a late-arriving higher priority process can pre-empt a currently running process with lower priority. In non-pre-emptive scheduling, a late-arriving higher priority process must wait for the currently executing process to complete before it can be scheduled on the processor. What are the turnaround times (time from arrival till completion) of  $P_2$  using pre-emptive and non-pre-emptive scheduling respectively. (GATE-2005) (2 Marks)

- (A) 30 sec, 30 sec      (B) 30 sec, 10 sec      (C) 42 sec, 42 sec      (D) 30 sec, 42 sec

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Round robin



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Round robin

- This algo is designed for time sharing systems, where it is not, necessary to complete one process and then start another, but to be responsive and divide time of CPU among the process in the ready state. Here ready queue is treated as a circular queue (FIFO).
- It is similar to FCFS scheduling, but pre-emption is added to enable the system to switch between processes.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval equivalent 1 Time quantum (where value of TQ can be anything).

- We fix a time quantum, up to which a process can hold the CPU in one go, with in which either a process terminates or process must release the CPU and enter the ready queue and wait for the next chance.
- The process may have a CPU burst of less than given time quantum. In this case, the process itself will release the CPU voluntarily.
- CPU Scheduler will select the next process for execution. OR The CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system.
- A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

- If there are  $n$  processes in the ready queue and the time quantum is  $q$ , then each process gets  $1/n$  of the CPU time in chunks of at most  $q$  time units.
- Each process must wait no longer than  $(n - 1) \times q$  time units until its next time quantum.



P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	0	4			
P <sub>1</sub>	1	5			
P <sub>2</sub>	2	2			
P <sub>3</sub>	3	1			
P <sub>4</sub>	4	6			
P <sub>5</sub>	6	3			
<b>Average</b>					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P <sub>0</sub>	5	5			
P <sub>1</sub>	4	6			
P <sub>2</sub>	3	7			
P <sub>3</sub>	1	9			
P <sub>4</sub>	2	2			
P <sub>5</sub>	6	3			
<b>Average</b>					

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- If the time quantum is extremely large, the RR policy is the same as the FCFS policy.
- If the time quantum is extremely small (say, 1 millisecond), the RR approach is called processor sharing and (in theory) creates the appearance that each of  $n$  processes has its own processor running at  $1/n$  the speed of the real processor. We also need also to consider the effect of context switching on the performance of RR scheduling.

- **Advantage**

- Perform best in terms of average response time
- Works well in case of time-sharing systems, client server architecture and interactive system
- kind of SJF implementation

- **Disadvantage**

- Longer process may starve
- Performance depends heavily on time quantum - If value of the time quantum is very less, then it will give lesser average response time (good but total no of context switches will be more, so CPU utilization will be less), If time quantum is very large then average response time will be more bad, but no of context switches will be less, so CPU utilization will be good.
- No idea of priority



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q consider the following set of processes and the length of CPU burst time given in milliseconds:

Process	CPU Burst Time	CT	TAT	WT
$P_1$	5			
$P_2$	7			
$P_3$	6			
$P_4$	4			

Assume that processes being scheduled with round robin scheduling algorithm with time quantum 4ms. Then the waiting for  $P_4$  is \_\_\_\_\_. (NET-DEC-2018)

a) 0

b) 4

c) 6

d) 12

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the following set of processes with the length of CPU burst time in milliseconds (ms):

Process	Burst Time	Priority	CT	TAT	WT
A	6	3			
B	1	1			
C	2	3			
D	1	4			
E	5	2			

Assume that processes are stored in ready queue in following order: A – B – C – D – E  
Using round robin scheduling with time slice of 1 ms, the average turnaround time is (NET-Spet-2013)

(A) 8.4 ms

(B) 12.4 ms

(C) 9.2 ms

(D) 9.4 ms

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero? **(GATE-2013)**  
**(1 Marks)**

- (A) This algorithm is equivalent to the first-come-first-serve algorithm
- (B) This algorithm is equivalent to the round-robin algorithm.
- (C) This algorithm is equivalent to the shortest-job-first algorithm..
- (D) This algorithm is equivalent to the shortest-remaining-time-first algorithm



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q In round robin CPU scheduling as time quantum is increased the average turnaround time (NET-June-2012)**

**(A) Increases**

**(B) Decreases**

**(C) remains constant**

**(D) varies irregularly**

**Q** Consider  $n$  processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes  $s$  seconds, what must be the quantum size  $q$  such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every  $t$  seconds? (GATE-1998) (1 Marks) (NET-Dec-2012)

a)  $q \leq (t-ns)/(n-1)$

b)  $q \geq (t-ns)/(n-1)$

c)  $q \leq (t-ns)/(n+1)$

d)  $q \geq (t-ns)/(n+1)$

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** If the time-slice used in the round-robin scheduling policy is more than the maximum time required to execute any process, then the policy will **(GATE-2008)**  
**(1 Marks)**

- (A) degenerate to shortest job first
- (B) degenerate to priority scheduling
- (C) degenerate to first come first serve
- (D) none of the above



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** In processor management, round robin method essentially uses the pre-emptive version of  
..... (NET-JUNE-2006)

- (A) FILO
- (B) FIFO
- (C) SJF
- (D) Longest time first



**Q** Four jobs to be executed on a single processor system arrive at time  $0^+$  in the order A, B, C, D. their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one-time unit is. **(GATE-1996) (2 Marks) (ISRO-2008)**

- (a) 10      (b) 4      (c) 8      (d) 9

**Q Which scheduling policy is most suitable for a time-shared operating system?  
(GATE-1995) (1 Marks)**

- (a) Shortest Job First**
  
- (b) Round Robin**
  
- (c) First Come First Serve**
  
- (d) Elevator**

**Q** Assume that the following jobs are to be executed on a single processor system. The jobs are assumed to have arrived at time  $0^+$  and in the order p, q, r, s, t. calculate the departure time (completion time) for job p if scheduling is round robin with time slice 1. (GATE-1993) (2 Marks)

Job Id	CPU Burst Time
P	4
Q	1
R	8
S	1
T	2



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q In which of the following scheduling criteria, context switching will never take place? (NET-July-2018)

a) Round Robin

b) Pre-emptive SJF

c) Non-Pre-emptive SJF

d) Preemptive priority

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Which of the following scheduling algorithms may cause starvation? (NET-Jan-2017)

- a. First-come-first-served
  - b. Round Robin
  - c. Priority
  - d. Shortest process next
  - e. Shortest remaining time first
- 
- a) a, c and e
  - b) c, d and e
  - c) b, d and e
  - d) b, c and d

Q A scheduling Algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (lowest priority). The scheduler reevaluates the process priority for every 'T' time units and decides next process to be scheduled. If the process has no I/O operations and all arrive at time zero, then the scheduler implements \_\_\_\_\_ criteria. (NET-June-2016)

a) Priority scheduling

b) Round Robin Scheduling

c) Shortest Job First

d) FCFS

Q Three processes A, B and C each execute a loop of 100 iterations. In each iteration of the loop, a process performs a single computation that requires  $t_c$  CPU milliseconds and then initiates a single I/O operation that lasts for  $t_{i/o}$  milliseconds. It is assumed that the computer where the processes execute has sufficient number of I/O devices and the OS of the computer assigns different I/O devices to each process. Also, the scheduling overhead of the OS is negligible. The processes have the following characteristics:

Process Id	$T_c$	$T_{i/o}$
A	100	500
B	350	500
C	200	500



The processes A, B, and C are started at times 0, 5 and 10 milliseconds respectively, in a pure time-sharing system (round robin scheduling) that uses a time slice of 50 milliseconds. The time in milliseconds at which process C would complete its first I/O operation is \_\_\_\_\_. (GATE-2014) (2 Mark)

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the following processes with time slice of 4 milliseconds (I/O requests are ignored):  
The average turnaround time of these processes will be (NET-June-2013)

Process	Arrival Time	Processing Time	CT	TAT	WT
A	0	8			
B	1	4			
C	2	9			
D	3	5			

- (A) 19.25 milliseconds  
(C) 19.5 milliseconds

- (B) 18.25 milliseconds  
(D) 18.5 milliseconds

**Q** Pre-emptive scheduling is the strategy of temporarily suspending a running process **(NET-June-2012)**

**a)** before the CPU time slice expires

**b)** to allow starving processes to run

**c)** when it requires I/O

**d)** to avoid collision



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q Consider the 3 processes,  $P_1$ ,  $P_2$  and  $P_3$  shown in the table. (GATE-2012) (2 Marks)**

Process	Arrival Time	Time Unit Required
$P_1$	0	5
$P_2$	1	7
$P_3$	3	4

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2-time units) are

- (A) FCFS:  $P_1, P_2, P_3$       RR:  $P_1, P_2, P_3$   
(C) FCFS:  $P_1, P_2, P_3$       RR:  $P_1, P_3, P_2$

- (B) FCFS:  $P_1, P_3, P_2$       RR:  $P_1, P_3, P_2$   
(D) FCFS:  $P_1, P_3, P_2$       RR:  $P_1, P_2, P_3$

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Which of the following statements are true? (GATE-2010) (2 Marks)

1) Shortest remaining time first scheduling may cause starvation

2) Pre-emptive scheduling may cause starvation

3) Round robin is better than FCFS in terms of response time

(A) 1 only

(B) 1 and 3 only

(C) 2 and 3 only

(D) 1, 2 and 3

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q An example of a non-preemptive CPU scheduling algorithm is: (NET-June-2008)

(A) Shortest job first scheduling.

(B) Round robin scheduling.

(C) Priority scheduling.

(D) Fair share scheduling.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle? **(GATE-2006) (2 Marks)**

(A) 0%

(B) 10.6%

(C) 30.0%

(D) 89.4%

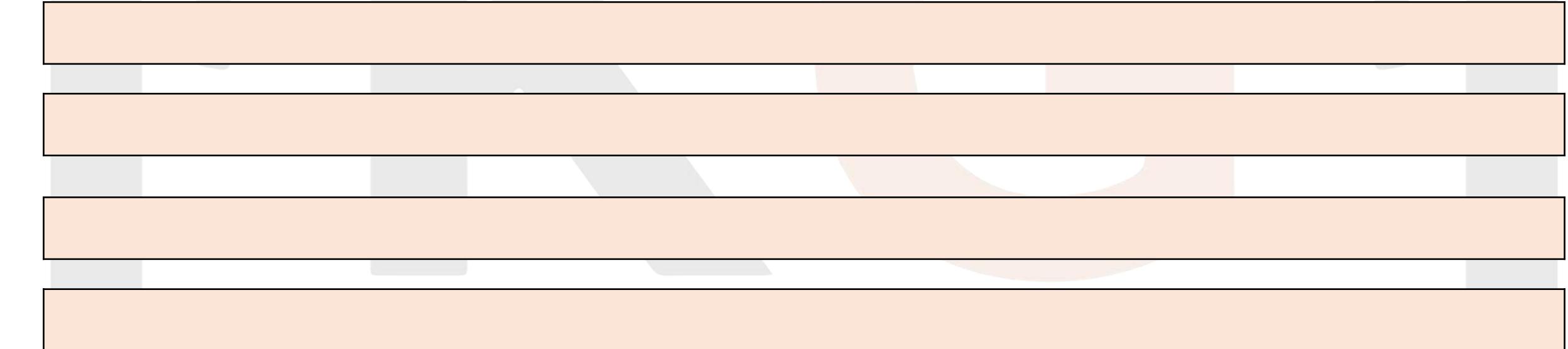


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q The arrival time, priority, and duration of the CPU and I/O bursts for each of three processes  $P_1$ ,  $P_2$  and  $P_3$  are given in the table below. Each process has a CPU burst followed by an I/O burst followed by another CPU burst. Assume that each process has its own I/O resource. (GATE-2006) (2 Marks)

Process	Arrival Time	Priority	Burst duration, CPU, I/O CPU
$P_1$	0	2	1,5,3
$P_2$	2	3(L)	3,3,1
$P_3$	3	1(H)	2,3,1



The programmed operating system uses preemptive priority scheduling. What are the finish times of the processes  $P_1$ ,  $P_2$  and  $P_3$ ?

- (A) 11, 15, 9      (B) 10, 15, 9      (C) 11, 16, 10      (D) 12, 17, 11

CLICK HERE ↗

Q .....is one of pre-emptive scheduling algorithm. (NET-Dec-2006)

(A) Shortest-Job-first

(B) Round-robin

(C) Priority based

(D) Shortest-Job-next

**Q** A uniprocessor computer system only has two processes, both of which alternate 10 ms CPU bursts with 90 ms I/O bursts. Both the processes were created at nearly the same time. The I/O of both processes can proceed in parallel. Which of the following scheduling strategies will result in the least CPU utilization (over a long period of time) for this system? **(GATE-2003) (2 Marks)**

- (A) First come first served scheduling**
- (B) Shortest remaining time first scheduling**
- (C) Static priority scheduling with different priorities for the two processes**
- (D) Round robin scheduling with a time quantum of 5 ms**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Longest Job First

- Process having longest Burst Time will get scheduled first.
- It can be both pre-emptive and non-pre-emptive in nature.
- The pre-emptive version is referred to as Longest Remaining Time First (LRTF) Scheduling Algorithm.

P. No	AT	BT
P <sub>1</sub>	0	3
P <sub>2</sub>	1	2
P <sub>3</sub>	2	4
P <sub>4</sub>	3	5
P <sub>5</sub>	4	6



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Longest Remaining Time First (LRTF)

- It is pre-emptive in nature.
- Longest Burst time process will get scheduled first.

P. No	AT	BT
P <sub>1</sub>	0	2
P <sub>2</sub>	0	4
P <sub>3</sub>	0	8



**Q** Consider three processes (process id 0, 1, 2 respectively) with compute time bursts 2, 4 and 8 time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with the lowest process id. The average turnaround time is? **(GATE-2006) (2 Marks)**

P. No	AT	BT	CT	TAT
P <sub>0</sub>	0	2		
P <sub>1</sub>	0	4		
P <sub>2</sub>	0	8		

(A) 13 units

(B) 14 units

(C) 15 units

(D) 16 units

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Highest response ratio next (HRRN)

- Scheduling is a non-pre-emptive discipline, similar to shortest job next (SJN), in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting.
- Jobs gain higher priority the longer they wait, which prevents indefinite waiting or in other words what we say starvation. In fact, the jobs that have spent a long time waiting compete against those estimated to have short run times.
- Response Ratio =  $(W + S)/S$
- Here, **W** is the waiting time of the process so far and **S** is the Burst time of the process.
- So, the conclusion is it gives priority to those processes which have less burst time (or execution time) but also takes care of the waiting time of longer processes, thus preventing starvation.

**Q** Consider a set of  $n$  tasks with known runtimes  $r_1, r_2, \dots, r_n$  to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput? **(GATE-2001) (1 Marks)**

- (A)** Round-Robin
- (B)** Shortest-Job-First
- (C)** Highest-Response-Ratio-Next
- (D)** First-Come-First-Served

Q highest response ratio next scheduling policy favours ----- jobs, but it also limits the waiting time of ----- jobs. (GATE-1990) (1 Marks)

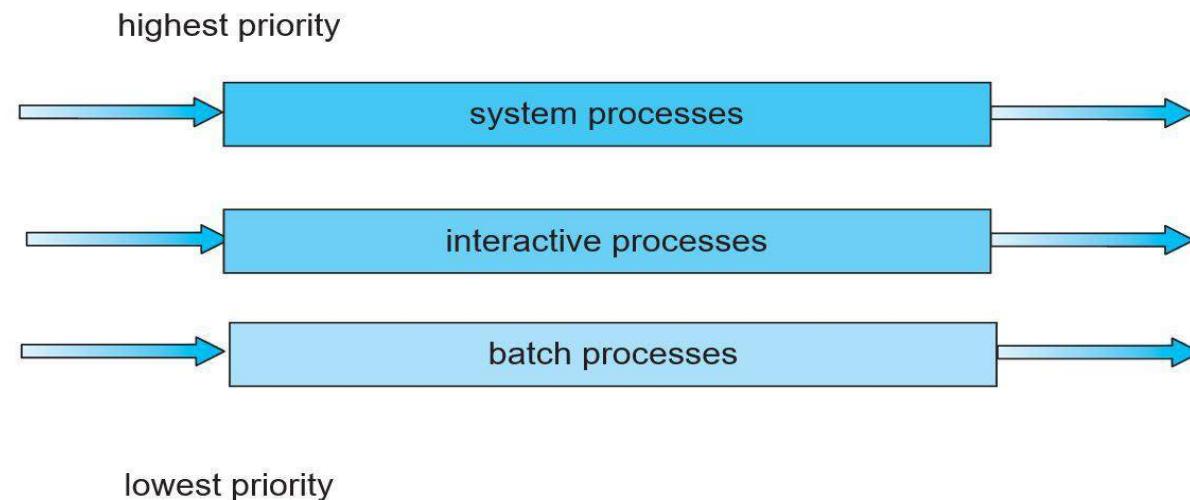


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Multi Level-Queue Scheduling

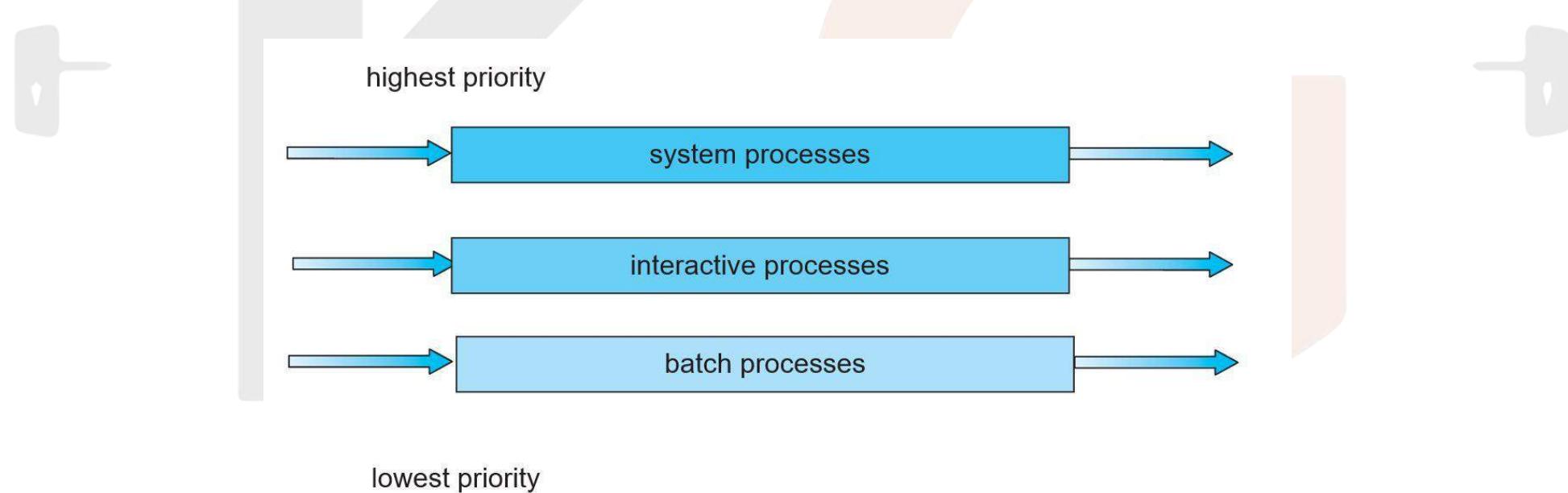
- Another class of scheduling algorithm has been created for situations in which processes are easily classified into different groups. For example, a common division is made between **foreground** (interactive) processes and **background** (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs.
- A **multilevel queue** scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.



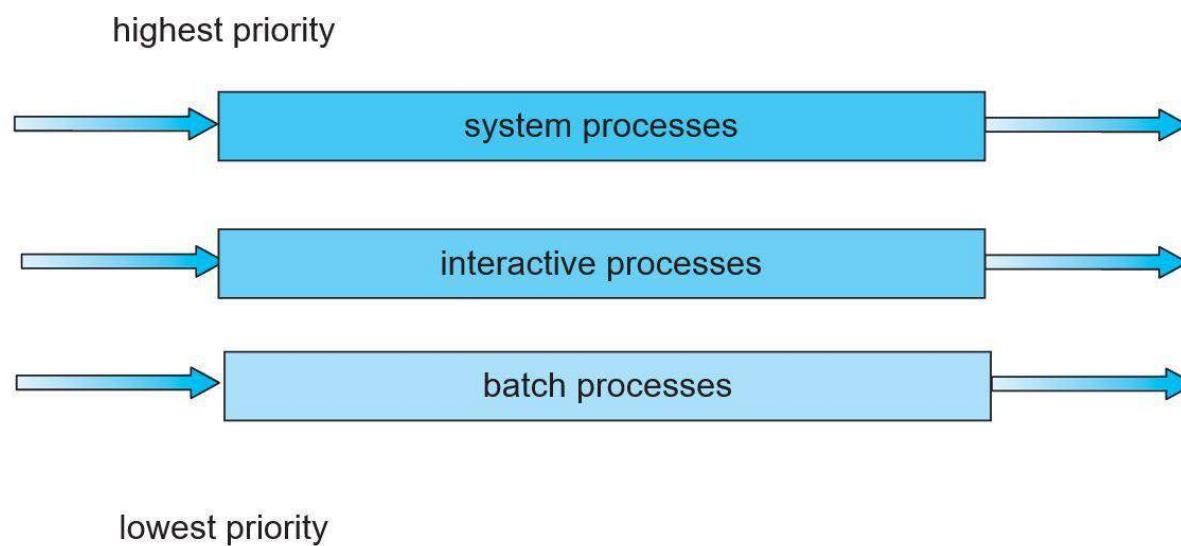
To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- Each queue has its own scheduling algorithm. For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. For example, the foreground queue may have absolute priority over the background queue.



- Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.
- Another possibility is to time-slice among the queues. Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes. For instance, in the foreground– background queue example, the foreground queue can be given 80 percent of the CPU time for RR scheduling among its processes, while the background queue receives 20 percent of the CPU to give to its processes on an FCFS basis.



**Q** Consider the following justifications for commonly using the two-level CPU scheduling:

**I.** It is used when memory is too small to hold all the ready processes.

**II.** Because its performance is same as that of the FIFO.

**III.** Because it facilitates putting some set of processes into memory and a choice is made from that.

**IV.** Because it does not allow to adjust the set of in-core processes.

Which of the following is true? **(NET-Dec-2014)**

**(A)** I, III and IV

**(B)** I and II

**(C)** III and IV

**(D)** I and III

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



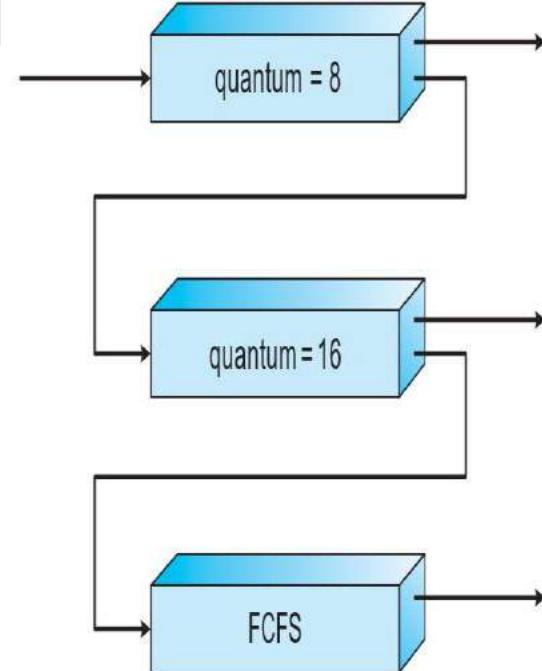
# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Multi-level Feedback Queue Scheduling

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature. This setup has the advantage of low scheduling overhead, but it is inflexible.
- The **multilevel feedback queue** scheduling algorithm, in contrast, allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- This scheduling algorithm gives highest priority to any process with a CPU burst of 8 milliseconds or less. Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/O burst.
- Processes that need more than 8 but less than 24 milliseconds are also served quickly, although with lower priority than shorter processes. Long processes automatically sink to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.

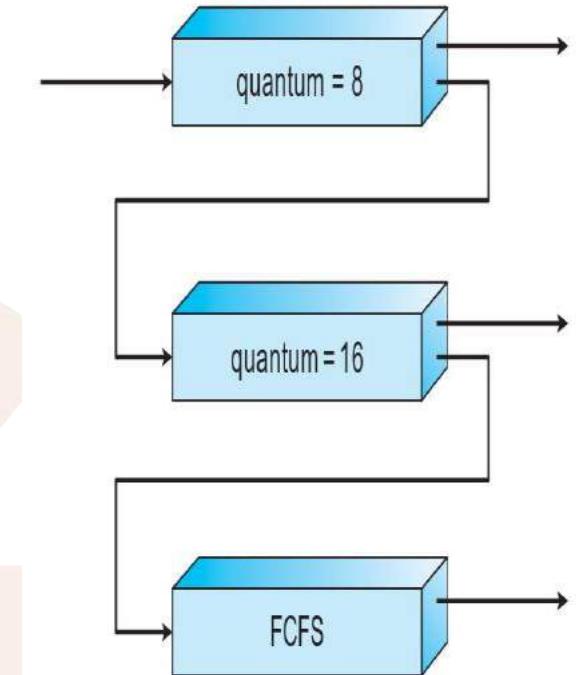


To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher- priority queue
- The method used to determine when to demote a process to a lower- priority queue
- The method used to determine which queue a process will enter when that process needs service
- The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it is also the most complex algorithm,
- since defining the best scheduler requires some means by which to select values for all the parameters.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Which of the following statements is not true for Multi-Level Feedback Queue processor scheduling algorithm? **(NET-June-2015)**

- a)** Queues have different priorities
- b)** Each queue may have different scheduling algorithm
- c)** Processes are permanently assigned to a queue
- d)** This algorithm can be configured to match a specific system under design

## Q Match the following:

List – I	List – II
a. Multilevel feedback queue	i. Time-slicing
b. FCFS	ii. Criteria to move processes between queues
c. Shortest process next	iii. Batch processing
d. Round robin scheduling	iv. Exponential smoothening

Codes: (NET-June-2014)

	a	b	c	d
(A)	i	iii	ii	iv
(B)	iv	iii	ii	i
(C)	iii	i	iv	i
(D)	ii	iii	iv	i

Q An example of a non-pre-emptive scheduling algorithm is: (NET-Dec-2008)

(A) Round Robin

(B) Priority Scheduling

(C) Shortest job first

(D) 2 level scheduling



Q Which of the following scheduling algorithms is non-preemptive? (GATE-2002) (1 Marks)

(A) Round Robin

(B) First-In First-Out

(C) Multilevel Queue Scheduling

(D) Multilevel Queue Scheduling with Feedback

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** An Operating System (OS) crashes on the average once in 30 days, that is, the Mean Time Between Failures (MTBF) = 30 days. When this happens, it takes 10 minutes to recover the OS, that is, the Mean Time To Repair (MTTR) = 10 minutes. The availability of the OS with these reliability figures is approximately: **(NET-AUG-2016)**

- a) 96.97%
- b) 97.97%
- c) 99.009%
- d) 99.97%

**Q** Consider a uniprocessor system where new processes arrive at an average of five processes per minute and each process needs an average of 6 seconds of service time. What will be the CPU utilization? **(NET-JUNE-2014)**

- a) 80 %
- b) 50 %**
- c) 60 %
- d) 30 %

Q.4

Three processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the length of the CPU bursts, the minimum achievable average waiting time for these three processes in a non-preemptive scheduler (round to nearest integer) is \_\_\_\_\_ milliseconds.

**(Gate-2021) (1 Marks)**

Q.19

Which of the following statement(s) is/are correct in the context of CPU scheduling?

(GATE-2021)

- (a) The goal is to only maximize CPU utilization and minimize throughput.
- (b) Turnaround time includes waiting time.
- (c) Round-robin policy can be used even when the CPU time required by each of the processes is not known apriori.
- (d) Implementing preemptive scheduling needs hardware support.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Real Time Speed



## Process Synchronization

- As we understand in a multiprogramming environment a good number of processes compete for limited number of resources.
- Concurrent access to shared data at same time may result in data inconsistency for e.g.

```
P ()  
{  
    read ( i );  
    i = i + 1;  
    write( i );  
}
```

# Race Condition

- The condition in which the output of a process depends on the execution sequence of process. i.e. if we change the order of execution of different process with respect to other process the output may change.
- That is why we need some kind of synchronization to eliminate the possibility of data inconsistency.

Q The following two functions  $P_1$  and  $P_2$  that share a variable B with an initial value of 2 execute concurrently.

$P_1()$	$P_2()$
{	{
$C = B - 1;$	$D = 2 * B;$
$B = 2 * C;$	$B = D - 1;$
}	}

The number of distinct values that B can possibly take after the execution is  
**(GATE-2015) (1 Mark)**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

$P_1()$	$P_2()$
{	{
$(I_{11}) C = B - 1;$	$(I_{21}) D = 2 * B;$
$(I_{12}) B = 2 * C;$	$(I_{22}) B = D - 1;$
}	}

Case1	Case2	Case3	Case4	Case5	Case6
$I_{11}$	$I_{21}$	$I_{11}$	$I_{21}$	$I_{11}$	$I_{21}$
$I_{12}$	$I_{22}$	$I_{21}$	$I_{11}$	$I_{21}$	$I_{11}$
$I_{21}$	$I_{11}$	$I_{22}$	$I_{12}$	$I_{12}$	$I_{22}$
$I_{22}$	$I_{12}$	$I_{12}$	$I_{22}$	$I_{22}$	$I_{12}$

$P_1()$	$P_2()$
{	{
$(I_{11}) C = B - 1;$	$(I_{21}) D = 2 * B;$
$(I_{12}) B = 2 * C;$	$(I_{22}) B = D - 1;$
}	}

	Case1	Case2	Case3	Case4	Case5	Case6
$I_{11}C=1$	$I_{21}D=4$	$I_{11}C=1$	$I_{21}D=4$	$I_{11}C=1$	$I_{21}D=4$	$I_{11}C=1$
$I_{12}B=2$	$I_{22}B=3$	$I_{21}D=4$	$I_{11}C=1$	$I_{21}D=4$	$I_{11}C=1$	$I_{21}D=4$
$I_{21}D=4$	$I_{11}C=2$	$I_{22}B=3$	$I_{12}B=2$	$I_{12}B=2$	$I_{22}B=3$	$I_{11}C=1$
$I_{22}B=3$	$I_{12}B=4$	$I_{12}B=2$	$I_{22}B=3$	$I_{22}B=3$	$I_{12}B=2$	$I_{12}B=2$

**Q** Consider three concurrent processes  $P_1$  and  $P_2$  and  $P_3$  as shown below, which access a shared variable D that has been initialized to 100. (GATE-2019) (2 Marks)

The processes are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then the value of  $Y-X$  is \_\_\_\_\_.

$P_1$	$P_2$	$P_3$
.	.	.
.	.	.
$D = D + 20$	$D = D - 50$	$D = D + 10$
.	.	.
.	.	.

**Q** When the result of a computation depends on the order of the processes execution, there is said to be **(GATE-1998) (1 Marks)**

- a) cycle stealing**
- b) race condition**
- c) a time lock**
- d) a deadlock**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# General Structure of a process

- **Initial Section:** Where process is accessing private resources.
- **Entry Section:** Entry Section is that part of code where, each process request for permission to enter its critical section.
- **Critical Section:** Where process is access shared resources.
- **Exit Section:** It is the section where a process will exit from its critical section.
- **Remainder Section:** Remaining Code.

```
P()
{
    While(T)
    {
        Initial Section
        Entry Section
        Critical Section
        Exit Section
        Remainder Section
    }
}
```

# Criterion to Solve Critical Section Problem

- **Mutual Exclusion**: No two processes should be present inside the critical section at the same time, i.e. only one process is allowed in the critical section at an instant of time.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next(means other process will participate which actually wish to enter).
- there should be no deadlock.

- **Bounded Waiting**: There exists a bound or a limit on the number of times a process is allowed to enter its critical section and no process should wait indefinitely to enter the CS.



## Some Points to Remember

- Mutual Exclusion and Progress are mandatory requirements that needs to be followed in order to write a valid solution for critical section problem.
- Bounded waiting is optional criteria, if not satisfied then it may lead to starvation.

**Q** Suppose P, Q and R are co-operating processes satisfying Mutual Exclusion condition. Then, if the process Q is executing in its critical section then (NET-DEC-2018)

a) Both 'P' and 'R' execute in critical section

b) Neither 'P' nor 'R' executes in their critical section

c) 'P' executes in critical section

d) 'R' executes in critical section

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Which of the following conditions does not hold good for a solution to a critical section problem? (NET-DEC-2014)

- a)** No assumptions may be made about speeds or the number of CPUs.
- b)** No two processes may be simultaneously inside their critical sections.
- c)** Processes running outside its critical section may block other processes.
- d)** Processes do not wait forever to enter its critical section.

Q Part of a program where the shared memory is accessed and which should be executed indivisibly, is called: (NET-JUNE-2007)

a) Semaphores

b) Directory

c) Critical Section

d) Mutual exclusion

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Solutions to Critical Section Problem

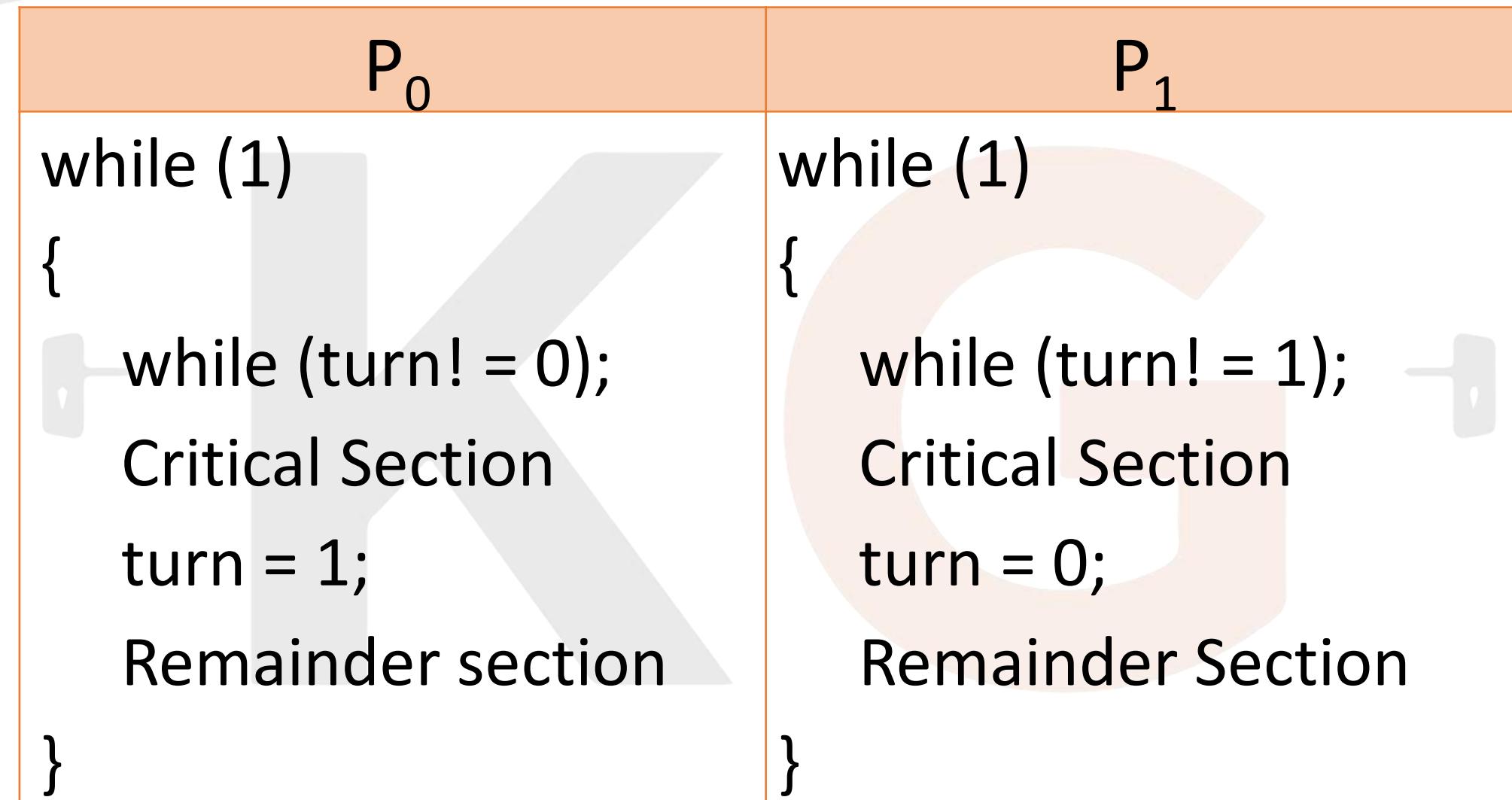
We generally have the following solutions to a Critical Section Problems:

1. Two Process Solution
  1. Using Boolean variable turn
  2. Using Boolean array flag
  3. Peterson's Solution
2. Operating System Solution
  1. Counting Semaphore
  2. Binary Semaphore
3. Hardware Solution
  1. Test and Set Lock
  2. Disable interrupt
4. Computer and Programming Support Type
  1. Monitors

## Two Process Solution

- In general it will be difficult to write a valid solution in the first go to solve critical section problem among multiple processes, so it will be better to first attempt two process solution and then generalize it to N-Process solution.
- There are 3 Different idea to achieve valid solution, in which some are invalid while some are valid.
- **1- Using Boolean variable turn**
- **2- Using Boolean array flag**
- **3- Peterson's Solution**

- Here we will use a Boolean variable turn, which is initialize randomly(0/1).



- The solution follows Mutual Exclusion as the two processes cannot enter the CS at the same time.
- The solution does not follow the Progress, as it is suffering from the strict alternation. Because we never asked the process whether it wants to enter the CS or not?

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

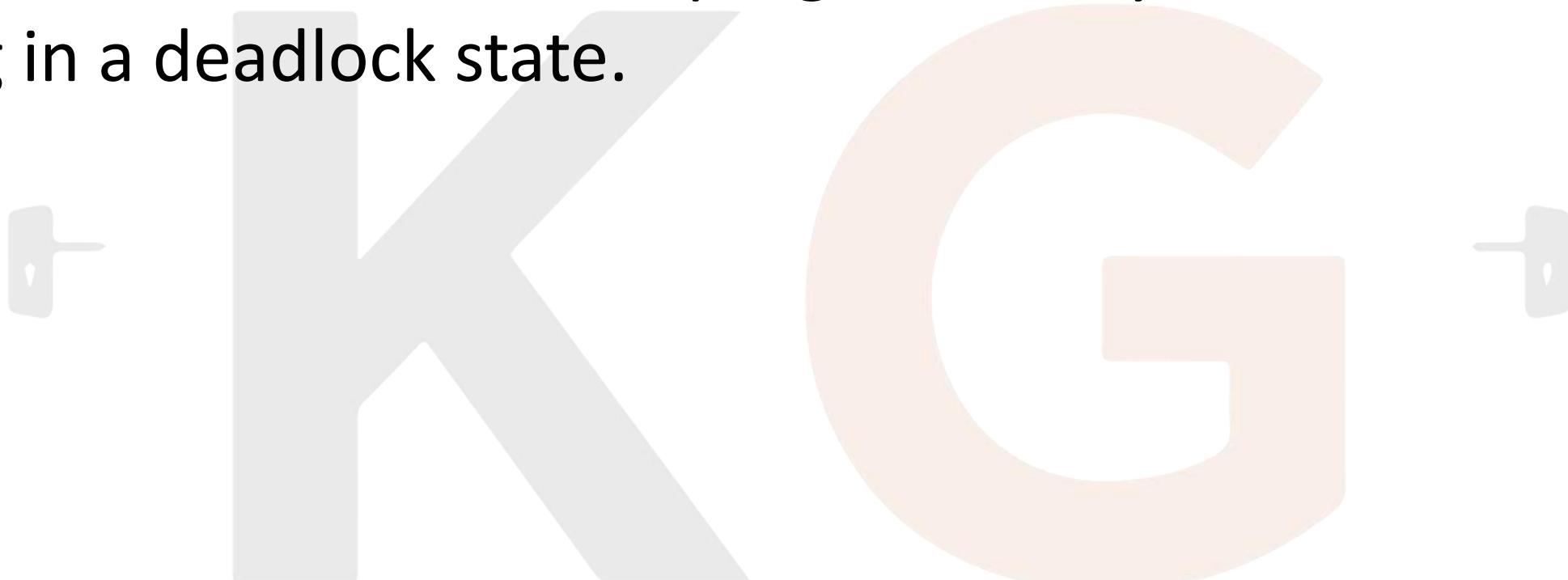
- Here we will use a Boolean array flag with two cells, where each cell is initialized to 0

$P_0$	$P_1$
<pre>while (1) {     flag [0] = T;     while (flag [1]);     Critical Section     flag [0] = F;     Remainder section }</pre>	<pre>while (1) {     flag [1] = T;     while (flag [0]);     Critical Section     flag [1] = F;     Remainder Section }</pre>

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

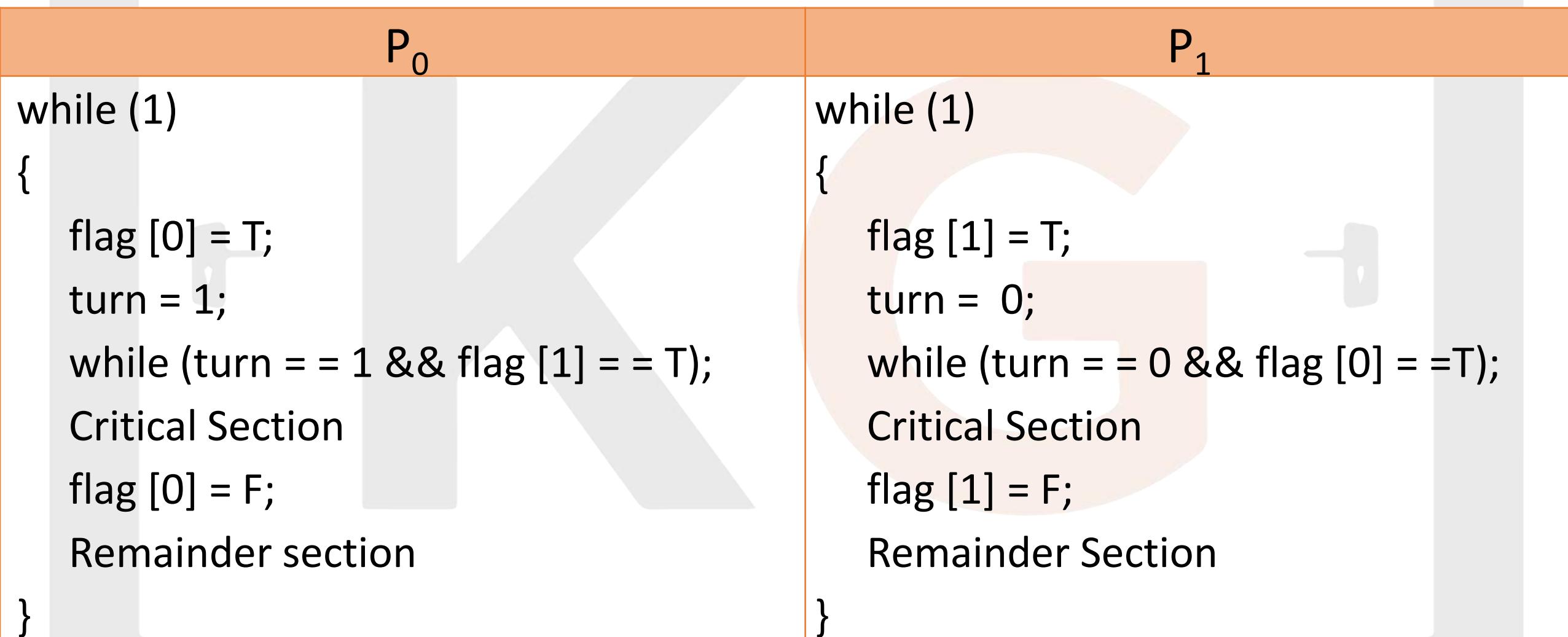
- This solution follows the Mutual Exclusion Criteria.
- But in order to achieve the progress the system ended up being in a deadlock state.



# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Peterson's solution is a classic Software-based solution to the critical-section problem for two processes.
- We will be using both: **turn** and **Boolean flag**.



- This solution ensures **Mutual Exclusion, Progress and Bounded Wait.**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the methods used by processes  $P_1$  and  $P_2$  for accessing their critical sections whenever needed, as given below. The initial values of shared Boolean variables  $S_1$  and  $S_2$  are randomly assigned. (GATE-2010) (1 Marks) (NET-JUNE-2012)

$P_1()$	$P_2()$
$\text{While}(S_1 == S_2);$	$\text{While}(S_1 != S_2);$
Critical section	Critical section
$S_1 = S_2;$	$S_1 = \text{not}(S_2);$

Which one of the following statements describes the properties achieved?

- (A) Mutual exclusion but not progress
- (B) Progress but not mutual exclusion
- (C) Neither mutual exclusion nor progress
- (D) Both mutual exclusion and progress

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the following two-process synchronization solution. (GATE-2016) (2 Marks)

The shared variable turn is initialized to zero.

Which one of the following is TRUE?

(a) This is a correct two-process synchronization solution.

(b) This solution violates mutual exclusion requirement

(c) This solution violates progress requirement.

(d) This solution violates bounded wait requirement

Process 0	Process 1
<b>Entry: loop while (turn == 1);</b>	Entry: loop while (turn == 0);
<b>(critical section)</b>	(critical section)
<b>Exit: turn = 1;</b>	Exit: turn = 0;

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes. Here, varP and varQ are shared variables and both are initialized to false. Which one of the following statements is true? (GATE-2015) (1 Mark)

- (A) The proposed solution prevents deadlock but fails to guarantee mutual exclusion
- (B) The proposed solution guarantees mutual exclusion but fails to prevent deadlock
- (C) The proposed solution guarantees mutual exclusion and prevents deadlock
- (D) The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion

Process X	Process Y
While(t)	While(t)
{	{
varP = T;	varQ = T;
While (varQ == T)	While (varP == T)
{	{
Critical section	Critical section
varP = F;	varQ = F;
}	}
}	}

**Q** Two processes,  $P_1$  and  $P_2$ , need to access a critical section of code. Consider the following synchronization construct used by the processes: Here,  $wants_1$  and  $wants_2$  are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct? (GATE-2007) (2 Mark)

- (A) It does not ensure mutual exclusion.
- (B) It does not ensure bounded waiting.
- (C) It requires that processes enter the critical section in strict alternation.
- (D) It does not prevent deadlocks, but ensures mutual exclusion.

$P_1()$	$P_2()$
While( $t$ )	While( $t$ )
{	{
$wants_1 = T$	$wants_2 = T$
While ( $wants_2 == T$ );	While ( $wants_1 == T$ );
Critical section	Critical section
$wants_1 = F$	$wants_2 = F$
Remainder section	Remainder section
}	}

**Q Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below. (GATE-2001) (2 Mark)**

Repeat

flag[i] = T;

turn = j;

while(P) do no-op;

Enter critical section, perform actions, then critical section

flag[i] = f;

Perform other non-critical section actions

Until false;

For the program to guarantee mutual exclusion, the predicate P in the while loop should be.

- (A) flag[j] = true and turn = i
- (C) flag[i] = true and turn = j

- (B) flag[j] = true and turn = j
- (D) flag[i] = true and turn = i

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Operation System Solution

- Semaphores are synchronization tools using which we will attempt n-process solution.
- A semaphore S is a simple integer variable that, but apart from initialization it can be accessed only through two standard atomic operations: wait(S) and signal(S).
- The wait(S) operation was originally termed as P(S) and signal(S) was originally called V(S).
- The definition of wait () is as follows:

<b>Wait(S)</b>
{
while(s<=0);
s--;
}

<b>Signal(S)</b>
{
s++;
}

- Peterson's Solution was confined to just two processes, and since in a general system can have n processes, Semaphores provides n-processes solution.
- While solving Critical Section Problem only we initialize semaphore  $S = 1$ .

$P_i()$
{
While( $T$ )
{
Initial Section
wait( $s$ )
Critical Section
signal( $s$ )
Remainder Section
}
}

- Semaphores are going to ensure Mutual Exclusion and Progress but does not ensures bounded waiting.

- It is critical that semaphore operations be executed atomically.
- We must guarantee that no two processes can execute wait() and signal() operations on the same semaphore at the same time. This is a critical-section problem; and in a single-processor environment, we can solve it by simply inhibiting interrupts during the time the wait() and signal() operations are executing.
- This scheme works in a single-processor environment because, once interrupts are inhibited, instructions from different processes cannot be interleaved. Only the currently running process executes until interrupts are reenabled and the scheduler can regain control.

- In a multiprocessor environment, interrupts must be disabled on every processor. Otherwise, instructions from different processes (running on different processors) may be interleaved in some arbitrary way.
- Disabling interrupts on every processor can be a difficult task and furthermore can seriously diminish performance. Therefore, SMP systems must provide alternative locking techniques —such as compare and swap() or spinlocks —to ensure that `wait()` and `signal()` are performed atomically

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Most Basic Questions

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Two atomic operations permissible on Semaphores are \_\_\_\_\_ and \_\_\_\_\_. (NET-NOV-2017)

- a) wait, stop
- b) wait, hold
- c) hold, signal
- d) wait, signal



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A relationship between processes such that each has some part (critical section) which must not be executed while the critical section of another is being executed, is known as (NET-JUNE-2011)

- a) Semaphore
- b) Mutual exclusion
- c) Multiprogramming
- d) Message passing



**Q** In order to allow only one process to enter its critical section, binary semaphore are initialized to **(NET-JUNE-2010)**

a) 0

b) 1

c) 2

d) 3



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q..... synchronize critical resources to prevent deadlock. (NET-JUNE-2006)

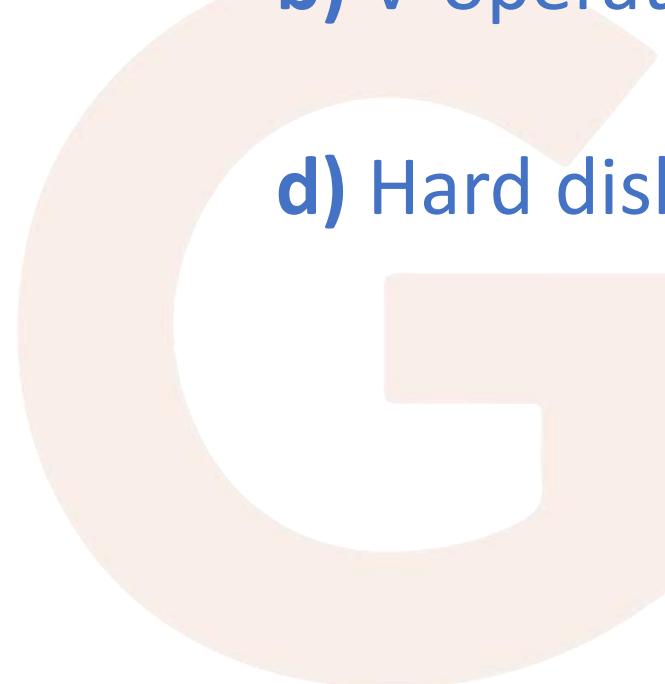
a) P-operator



c) Semaphores



b) V-operator



d) Hard disk



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q Semaphores are used to: (NET-DEC-2004)**

- a) Synchronise critical resources to prevent deadlock**
- b) Synchronise critical resources to prevent contention**
- c) Do I/o**
- d) Facilitate memory management**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A critical section is a program segment? (GATE-1996) (1 Mark)

- (a) which should run in a certain specified amount of time
- (b) which avoids deadlocks
- (c) where shared resources are accessed
- (d) which must be enclosed by a pair of semaphore operations, P and V

Q A critical region is (GATE-1987) (1 Marks)

- a) One which is enclosed by a pair of P and V operations on semaphores.
- b) A program segment that has not been proved bug-free.
- c) A program segment that often causes unexpected system crashes.
- d) A program segment where shared resources are accessed.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Ordering

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



Q Using Semaphores ensure that the order of execution of these concurrent process  $p_1$ ,  $p_2$  and  $p_3$  must be  $p_2 \rightarrow p_3 \rightarrow p_1$ ?

$P_1()$	$P_2()$	$P_3()$
code	code	code

Q A certain computation generates two arrays a and b such that  $a[i]=f(i)$  for  $0 \leq i < n$  and  $b[i]=g(a[i])$  for  $0 \leq i < n$ . Suppose this computation is decomposed into two concurrent processes X and Y such that X computes the array a and Y computes the array b. The processes employ two binary semaphores R and S, both initialized to zero. The array a is shared by the two processes. The structures of the processes are shown below. (GATE-2013) (2 Marks)

Process X:	Process Y:
private i; for (i=0; i < n; i++) { a[i] = f(i); ExitX(R, S); }	private i; for (i=0; i < n; i++) { EntryY(R, S); b[i]=g(a[i]); }

- 
- a) ExitX(R, S) {  
    P(R);  
    V(S);  
}  
EntryY(R, S) {  
    P(S);  
    V(R);  
}
- b) ExitX(R, S) {  
    V(R);  
    V(S);  
}  
EntryY(R, S) {  
    P(R);  
    P(S);  
}
- c) ExitX(R, S) {  
    P(S);  
    V(R);  
}  
EntryY(R, S) {  
    V(S);  
    P(R);  
}
- d) ExitX(R, S) {  
    V(R);  
    P(S);  
}  
EntryY(R, S) {  
    V(S);  
    P(R);  
}

**Q** Two concurrent processes  $P_1$  and  $P_2$  use four shared resources  $R_1$ ,  $R_2$ ,  $R_3$  and  $R_4$ , as shown below.

Both processes are started at the same time, and each resource can be accessed by only one process at a time. The following scheduling constraints exist between the access of resources by the processes:

- i)  $P_2$  must complete use of  $R_1$  before  $P_1$  gets access to  $R_1$
- ii)  $P_1$  must complete use of  $R_2$  before  $P_2$  gets access to  $R_2$ .
- iii)  $P_2$  must complete use of  $R_3$  before  $P_1$  gets access to  $R_3$ .
- iv)  $P_1$  must complete use of  $R_4$  before  $P_2$  gets access to  $R_4$ .

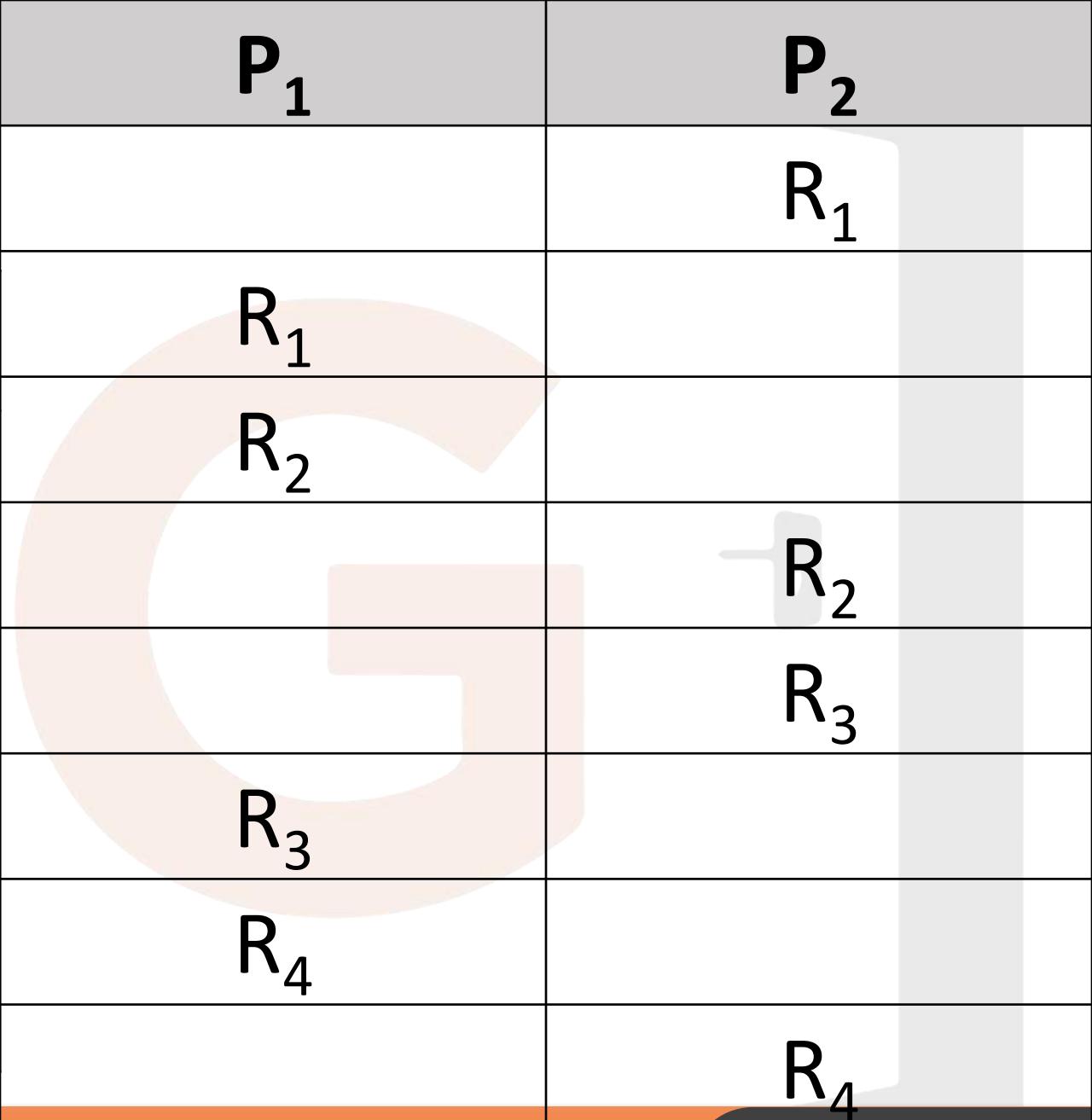
There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed? (GATE-2005) (2 Marks)

- (A) 1      (B) 2      (C) 3      (D) 4

$P_1$	$P_2$
Compute:	Compute;
Use $R_1$ ;	Use $R_1$ ;
Use $R_2$ ;	Use $R_2$ ;
Use $R_3$ ;	Use $R_3$ ;
Use $R_4$ ;	Use $R_4$ ;

$P_2$  must complete use of  $R_1$  before  $P_1$  gets access to  $R_1$   
 $P_1$  must complete use of  $R_2$  before  $P_2$  gets access to  $R_2$ .  
 $P_2$  must complete use of  $R_3$  before  $P_1$  gets access to  $R_3$ .  
 $P_1$  must complete use of  $R_4$  before  $P_2$  gets access to  $R_4$ .

$P_1$	$P_2$
Compute:	Compute;
Use $R_1$ ;	Use $R_1$ ;
Use $R_2$ ;	Use $R_2$ ;
Use $R_3$ ;	Use $R_3$ ;
Use $R_4$ ;	Use $R_4$ ;



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

$P_1$	$P_2$
$P(X)$	$R_1$
$R_1$	$V(X)$
$R_2$	$P(Y)$
$V(Y)$	$R_2$
$P(X)$	$R_3$
$R_3$	$V(X)$
$R_4$	$P(Y)$
$V(Y)$	$R_4$

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q Semaphores S and T. The code for the processes P and Q is shown below. Synchronization statements can be inserted only at points W, X, Y and Z.

Process P	Process Q
While(t)	While(t)
{	{
W:	Y:
print '0';	print '1';
print '0';	print '1';
X:	Z:
}	}

**Which of the following will always lead to an output starting with '001100110011' ? (GATE-2004) (2 Marks)**

- (A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
- (B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0
- (C) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
- (D) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S initially 1, and T initially 0

Q Semaphores S and T. The code for the processes P and Q is shown below. Synchronization statements can be inserted only at points W, X, Y and Z.

Process P	Process Q
While(t)	While(t)
{	{
W:	Y:
print '0';	print '1';
print '0';	print '1';
X:	Z:
}	}

Which of the following will ensure that the output string never contains a substring of the form  $01^n0$  or  $10^n1$  where n is odd? (GATE-2004) (2 Marks)

- (A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
- (B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
- (C) P(S) at W, V(S) at X, P(S) at Y, V(S) at Z, S initially 1
- (D) V(S) at W, V(T) at X, P(S) at Y, P(T) at Z, S and T initially 1

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# DeadLock

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q which of the following will ensure deadlock free execution, if both  $s_1$  and  $s_2$  both are initialized to 1?

Process P	Process Q
While( $t$ )	While( $t$ )
{	{
wait( $s_1$ )	wait( $s_2$ )
wait( $s_2$ )	wait( $s_1$ )
CS	CS
}	}

Process P	Process Q
While( $t$ )	While( $t$ )
{	{
wait( $s_1$ )	wait( $s_1$ )
wait( $s_2$ )	wait( $s_2$ )
CS	CS
}	}

**Conclusion:** if we want to avoid deadlock the order to apply wait operation on different semaphores among different must be same.



Q Suppose S and Q are two semaphores initialized to 1.  $P_1$  and  $P_2$  are two processes which are sharing resources. (NET-JUNE-2013)

Their execution may sometimes lead to an undesirable situation called

(A) Starvation

(B) Race condition

(C) Multithreading

(D) Deadlock

$P_1$	$P_2$
wait(S);	wait(Q);
wait(Q);	wait(S);
Critical section 1;	critical-section 2;
signal(S);	signal(Q);
signal(Q);	signal(S);

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d, and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the V operation (i.e., signal) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes? (GATE-2013) (1 Marks)

- (A) X: P(a)P(b)P(c) Y: P(b)P(c)P(d) Z: P(c)P(d)P(a)
- (B) X: P(b)P(a)P(c) Y: P(b)P(c)P(d) Z: P(a)P(c)P(d)
- (C) X: P(b)P(a)P(c) Y: P(c)P(b)P(d) Z: P(a)P(c)P(d)
- (D) X: P(a)P(b)P(c) Y: P(c)P(b)P(d) Z: P(c)P(d)P(a)

X	Y	Z
CS	CS	CS

**Q** Consider two processes  $P_1$  and  $P_2$  accessing the shared variables  $X$  and  $Y$  protected by two binary semaphores  $S_X$  and  $S_Y$  respectively, both initialized to 1.  $P$  and  $V$  denote the usual semaphore operators, where  $P$  decrements the semaphore value, and  $V$  increments the semaphore value. The pseudo-code of  $P_1$  and  $P_2$  is as follows: **(GATE-2004) (2 Marks)**

In order to avoid deadlock, the correct operators at  $L_1$ ,  $L_2$ ,  $L_3$  and  $L_4$  are respectively

- (A)  $P(S_Y)$ ,  $P(S_X)$ ;  $P(S_X)$ ,  $P(S_Y)$
- (B)  $P(S_X)$ ,  $P(S_Y)$ ;  $P(S_Y)$ ,  $P(S_X)$
- (C)  $P(S_X)$ ,  $P(S_X)$ ;  $P(S_Y)$ ,  $P(S_Y)$
- (D)  $P(S_X)$ ,  $P(S_Y)$ ;  $P(S_X)$ ,  $P(S_Y)$

$P_1$	$P_2$
While true do {	While true do {
$L_1$ : .....	$L_3$ : .....
$L_2$ : .....	$L_4$ : .....
$X = X + 1;$	$Y = Y + 1;$
$Y = Y - 1;$	$X = Y - 1;$
$V(S_X);$	$V(S_X);$
$V(S_Y);$	$V(S_Y);$
}	}

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** A shared variable  $x$ , initialized to zero, is operated on by four concurrent processes  $W, X, Y, Z$  as follows. Each of the processes  $W$  and  $X$  reads  $x$  from memory, increments by one, stores it to memory, and then terminates. Each of the processes  $Y$  and  $Z$  reads  $x$  from memory, decrements by two, stores it to memory, and then terminates. Each process before reading  $x$  invokes the  $P$  operation (i.e., wait) on a counting semaphore  $S$  and invokes the  $V$  operation (i.e., signal) on the semaphore  $S$  after storing  $x$  to memory. Semaphore  $S$  is initialized to two. What is the maximum possible value of  $x$  after all processes complete execution? (GATE-2013) (2 Marks)

(A) -2

(B) -1

(C) 1

(D) 2

W	X	Y	Z
Wait( $S$ )	Wait( $S$ )	Wait( $S$ )	Wait( $S$ )
$R(x)$	$R(x)$	$R(x)$	$R(x)$
$X = x + 1$	$X = x + 1$	$X = x - 2$	$X = x - 2$
$W(x)$	$W(x)$	$W(x)$	$W(x)$
Signal( $S$ )	Signal( $S$ )	Signal( $S$ )	Signal( $S$ )

Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on left.

The variables process\_arrived and process\_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

**Q** The above implementation of barrier is incorrect. Which one of the following is true? **(GATE-2006) (2 Marks)**

- a)** The barrier implementation is wrong due to the use of binary semaphore S.
- b)** The barrier implementation may lead to a deadlock if two barrier invocations are used in immediate succession.
- c)** Lines 6 to 10 need not be inside a critical section.
- d)** The barrier implementation is correct if there are only two processes instead of three.

```
void barrier (void) {  
1: P(S);  
2: process_arrived++;  
3: V(S);  
4: while (process_arrived !=3);  
5: P(S);  
6: process_left++;  
7: if (process_left==3) {  
8: process_arrived = 0;  
9: process_left = 0;  
10: }  
11: V(S);  
}
```

Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on left.

The variables process\_arrived and process\_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

**Q** Which one of the following rectifies the problem in the implementation? **(GATE-2006) (2 Marks)**

- (A) The barrier implementation is wrong due to the use of binary semaphore S
- (B) The barrier implementation may lead to a deadlock if two barrier invocations are used in immediate succession.
- (C) Lines 6 to 10 need not be inside a critical section
- (D) The barrier implementation is correct if there are only two processes instead of three.

```
void barrier (void) {  
1: P(S);  
2: process_arrived++;  
3: V(S);  
4: while (process_arrived !=3);  
5: P(S);  
6: process_left++;  
7: if (process_left==3) {  
8: process_arrived = 0;  
9: process_left = 0;  
10: }  
11: V(S);  
}
```

**Q** Given below is a program which when executed spawns two concurrent processes:

```
semaphore X := 0 ;  
/* Process now forks into concurrent processes P1 & P2 */
```

Consider the following statements about processes P<sub>1</sub> and P<sub>2</sub>:

- i) It is possible for process P<sub>1</sub> to starve.
- ii) It is possible for process P<sub>2</sub> to starve.

Which of the following holds? (GATE-2005) (2 Marks)

- (A) Both I and II are true
- (B) I is true but II is false
- (C) II is true but I is false
- (D) Both I and II are false

P <sub>1</sub>	P <sub>2</sub>
repeat forever V (X) ; Compute ; P(X) ;	repeat forever P(X) ; Compute ; V(X) ;

# Break

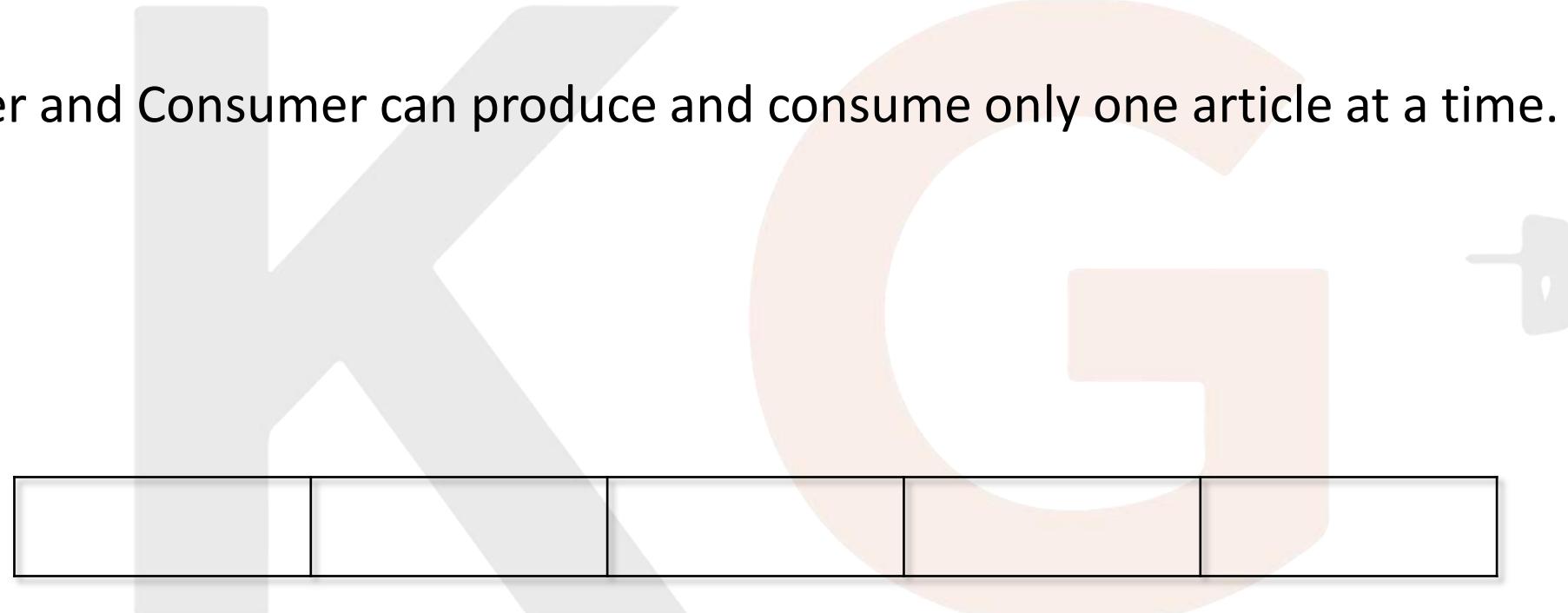
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Classical Problems on Synchronization

- There are number of actual industrial problem we try to solve in order to improve our understand of Semaphores and their power of solving problems.
- Here in this section we will discuss a number of problems like
  - Producer consumer problem/ Bounder Buffer Problem
  - Reader-Writer problem
  - Dining Philosopher problem

# Producer-Consumer Problem

- **Problem Definition** – There are two process Producer and Consumers, producer produces information and put it into a buffer which have  $n$  cell, that is consumed by a consumer.
- Both Producer and Consumer can produce and consume only one article at a time.



Real Time Speed



# Producer-Consumer Problem needs to sort out three major issues

- A producer needs to check whether the buffer is overflowed or not after producing an item before accessing the buffer.
- Similarly, a consumer needs to check for an underflow before accessing the buffer and then consume an item.
- Also, *the producer and consumer must be synchronized, so that once a producer and consumer is accessing the buffer the other must wait.*



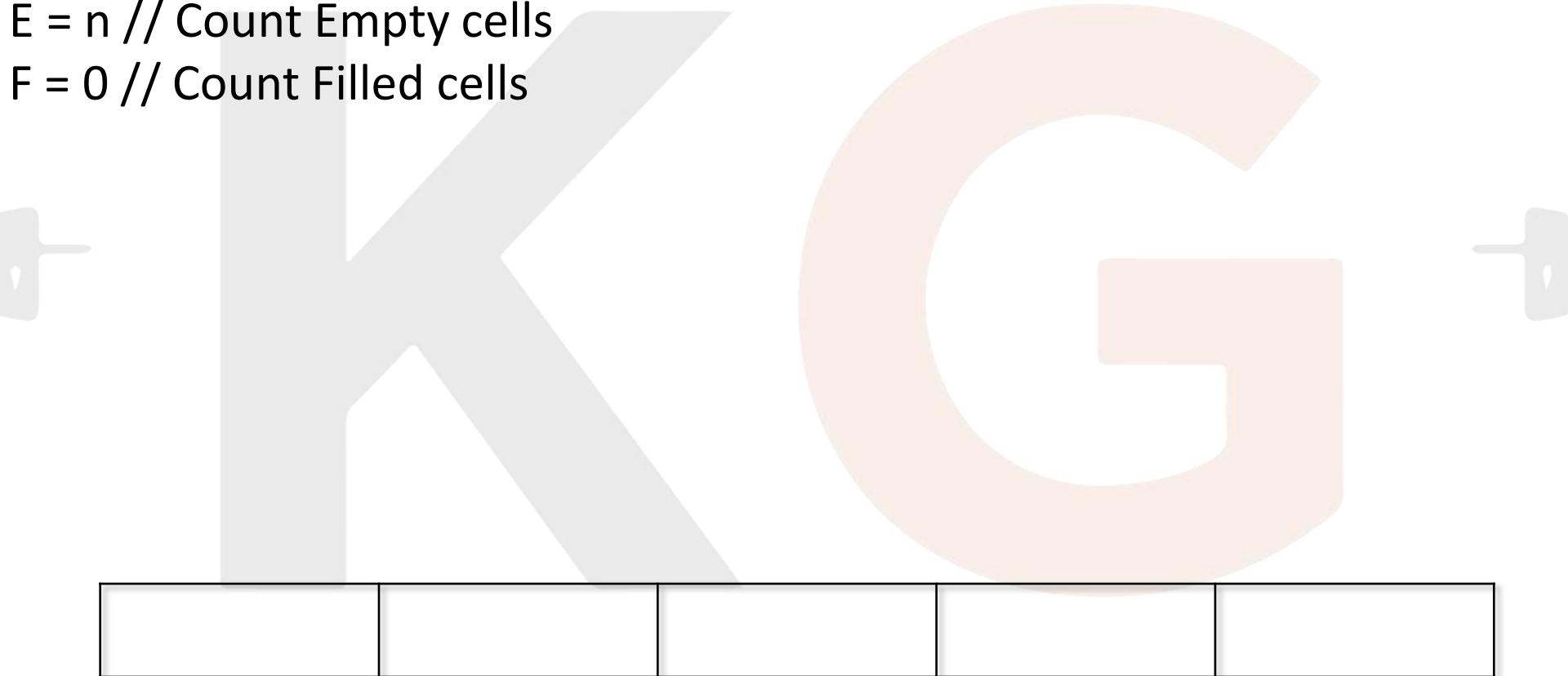
## Solution Using Semaphores

Now to solve the problem we will be using three semaphores:

Semaphore S = 1 // CS

Semaphore E = n // Count Empty cells

Semaphore F = 0 // Count Filled cells



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



## Semaphore S =



## Semaphore E =

## Semaphore F =

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

Semaphore S =



Semaphore E =

Semaphore F =

	Producer()	Consumer()
Producer()	Producer()	Consumer()
{	{	{
while(T)		while(T)
{		{
// Produce an item		
		// Pick item from buffer
	// Add item to buffer	
		// Consume item
}		}

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Semaphore S =



Semaphore E =

Semaphore F =

Producer()	Consumer()
Producer()	Consumer()
{	{
while(T)	while(T)
{	{
// Produce an item	
	wait(S)
wait(S)	// Pick item from buffer
// Add item to buffer	signal(S)
signal(S)	
	// Consume item
}	}

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Total three resources  
are used

- semaphore E take count of empty cells and over flow
- semaphore F take count of filled cells and under flow
- Semaphore S take care of buffer

Producer()	Consumer()
{	{
while(T)	while(T)
{	{
// Produce an item	wait(F)//UnderFlow
wait(E)//OverFlow	wait(S)
wait(S)	// Pick item from buffer
// Add item to buffer	signal(S)
signal(S)	wait(E)
wait(F)	Consume item
}	}
}	}

**Q** Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is  $N$ . Three semaphores *empty*, *full* and *mutex* are defined with respective initial values of 0,  $N$  and 1. Semaphore *empty* denotes the number of available slots in the buffer, for the consumer to read from. Semaphore *full* denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by  $P$ ,  $Q$ ,  $R$  and  $S$ , in the code below can be assigned either *empty* or *full*. The valid semaphore operations are: *wait()* and *signal()*. (GATE-2018) (2 Marks)

Which one of the following assignments to  $P$ ,  $Q$ ,  $R$  and  $S$  will yield the correct solution?

- (A)  $P: full$ ,  $Q: full$ ,  $R: empty$ ,  $S: empty$
- (B)  $P: empty$ ,  $Q: empty$ ,  $R: full$ ,  $S: full$
- (C)  $P: full$ ,  $Q: empty$ ,  $R: empty$ ,  $S: full$
- (D)  $P: empty$ ,  $Q: full$ ,  $R: full$ ,  $S: empty$

Producer:	Consumer:
Do{	Do{
Wait(P);	Wait(R);
Wait(mutex);	Wait(mutex);
//Add item to buffer	//Consume item from buffer
Signal(mutex);	Signal(mutex);
Signal(Q);	Signal(S);
} while(1);	} while(1);

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q The Bounded buffer problem is also known as \_\_\_\_\_. (NET-NOV-2017)

- a) Producer - consumer problem
- b) Reader - writer problem
- c) Dining Philosophers problem
- d) Both (2) and (3)

**Q** Consider the procedure below for the Producer-Consumer problem which uses semaphores:

Semaphore n = 0;

Semaphore s = 1;

Which one of the following is TRUE? (GATE-2014) (2 Marks)

- (A) The producer will be able to add an item to the buffer, but the consumer can never consume it.
- (B) The consumer will remove no more than one item from the buffer.
- (C) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.
- (D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.

Void Producer ()	Void Consumer ()
{	{
<b>While(true)</b>	While(true)
{	{
<b>Produce ();</b>	semWait(s);
<b>SemWait(s);</b>	semWait(n);
<b>addToBuffer();</b>	RemovefromBuffer();
<b>semSignal(s);</b>	semSignal(s);
<b>SemSignal(n);</b>	consume();
}	}
}	}

Q Producer consumer problem can be solved using: (NET-DEC-2005)

a) semaphores



c) monitors

b) event counters



d) all the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Process  $P_1$  repeatedly adds one item at a time to a buffer of size  $n$ , and process  $P_2$  repeatedly removes one item at a time from the same buffer using the programs given below. In the programs, K, L, M and N are unspecified statements. (GATE-2004) (2 Marks)

The statements K, L, M and N are respectively

- (A) P(full), V(empty), P(full), V(empty)
- (B) P(full), V(empty), P(empty), V(full)
- (C) P(empty), V(full), P(empty), V(full)
- (D) P(empty), V(full), P(full), V(empty)

P1()	p2()
while(T) {	while(T) {
K;	M;
P(mutex);	P(mutex);
Add an item to the buffer;	Remove an item to the buffer;
V(mutex);	V(mutex);
L;	N;
}	}

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Reader-Writer Problem

- Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database (readers), whereas others may want to update (that is, to read and write) the database(writers). The former are referred to as readers and to the latter as writers.
  - If two readers access the shared data simultaneously, no adverse effects will result. But, if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.

title	release_year	length	replacement_cost
Meet the Fockers	2004	159	29.99
Wingin' It	2004	179	29.99
Minority Report	2002	172	29.99
Tracer	2004	142	29.99
Song of the Sea	2004	103	29.99
Blackout	2004	179	29.99
Savvy Fucker	2004	134	29.99
River's Edge	2004	149	29.99
Right: Crime	2004	133	29.99
Quest: Mission	2004	177	29.99
Position: Forward	2004	159	29.99
Assisting: Legally	2004	148	29.99
Lawless: Victim	2004	181	29.99
Simple: Superbitch	2004	124	29.99
Serious: Hell	2004	172	29.99
Japanese: Pot	2004	135	29.99
Ultimate: Driver	2004	163	29.99
Fluents: Karate	2004	149	29.99
Fantastic: Park	2004	131	29.99
Extraordinary: Computer	2004	127	29.99
Everyone: Craft	2004	163	29.99
Thirty: Ace	2004	147	29.99
Clyde: Theory	2004	139	29.99
Clockwork: Paradox	2004	143	29.99
Billboard: Superstardom	2004	172	29.99
(2004)			

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

- To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database.
- **Points that needs to be taken care for generating a Solutions:**
  - The solution may allow more than one reader at a time, but should not allow any writer.
  - The solution should strictly not allow any reader or writer, while a writer is performing a write operation.

- **Solution using Semaphores**

- The reader processes share the following data structures:
- semaphore mutex = 1, wrt =1; // Two semaphores
- int readcount = 0; // Variable

## Mutex =

**W<sub>rt</sub>** =

**Readcount =**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

**Mutex =**

**Wrt =**

**Readcount =**



<b>Writer()</b>	<b>Reader()</b>
CS //Write	CS //Read

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Three resources are used

- Semaphore Wrt is used for synchronization between WW, WR, RW
- Semaphore reader is used to synchronize between RR
- Readcount is simple int variable which keep counts of number of readers

Writer()	Reader()
<b>Wait(wrt)</b>	
CS //Write	CS //Read
<b>Signal(wrt)</b>	

Three resources are used

- Semaphore Wrt is used for synchronization between WW, WR, RW
- Semaphore reader is used to synchronize between RR
- Readcount is simple int variable which keep counts of number of readers

Writer()	Reader()
	<b>Readcount++</b>
Wait(wrt)	
CS //Write	CS //Read
Signal(wrt)	
	<b>Readcount--</b>

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Three resources are used

- Semaphore Wrt is used for synchronization between WW, WR, RW
- Semaphore reader is used to synchronize between RR
- Readcount is simple int variable which keep counts of number of readers

Writer()	Reader()
	<b>Wait(mutex)</b>
	Readcount++
Wait(wrt)	<b>signal(mutex)</b>
CS //Write	CS //Read
Signal(wrt)	<b>Wait(mutex)</b>
	Readcount--
	<b>signal(mutex)</b>

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Three resources are used

- Semaphore Wrt is used for synchronization between WW, WR, RW
- Semaphore reader is used to synchronize between RR
- Readcount is simple int variable which keep counts of number of readers

Writer()	Reader()
	Wait(mutex)
	Readcount++
	<b>If(readcount ==1)</b>
	<b>wait(wrt) // first</b>
Wait(wrt)	signal(mutex)
CS //Write	CS //Read
Signal(wrt)	Wait(mutex)
	Readcount--
	<b>If(readcount ==0)</b>
	<b>signal(wrt) // last</b>
	signal(mutex)

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q To overcome difficulties in Readers-Writers problem, which of the following statement/s is/are true? (NET-DEC-2018)

- 1) Writers are given exclusive access to shared objects
- 2) Readers are given exclusive access to shared objects
- 3) Both readers and writers are given exclusive access to shared objects.

Choose the correct answer from the code given below:

- a) 1
- b) 2
- c) 3
- d) 2 and 3 only

Q Let  $P_i$  and  $P_j$  two processes,  $R$  be the set of variables read from memory, and  $W$  be the set of variables written to memory. For the concurrent execution of two processes  $P_i$  and  $P_j$  which of the conditions are not true? (NET-JUNE-2015)

a)  $R(P_i) \cap W(P_j) = \emptyset$



b)  $W(P_i) \cap R(P_j) = \emptyset$



c)  $R(P_i) \cap R(P_j) = \emptyset$

d)  $W(P_i) \cap W(P_j) = \emptyset$

**Q** Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization **(GATE-2007) (2 Marks)**

Writer()	Reader()
	Wait(mutex)
	Readcount++
	If(readcount ==1)
	$S_1$
Wait(wrt)	$S_2$
Writing is performed	CS //Read
Signal(wrt)	$S_3$
	Readcount--
	If(readcount ==0)
	$S_4$
	signal(mutex)

The values of  $S_1, S_2, S_3, S_4$ , (in that order) are

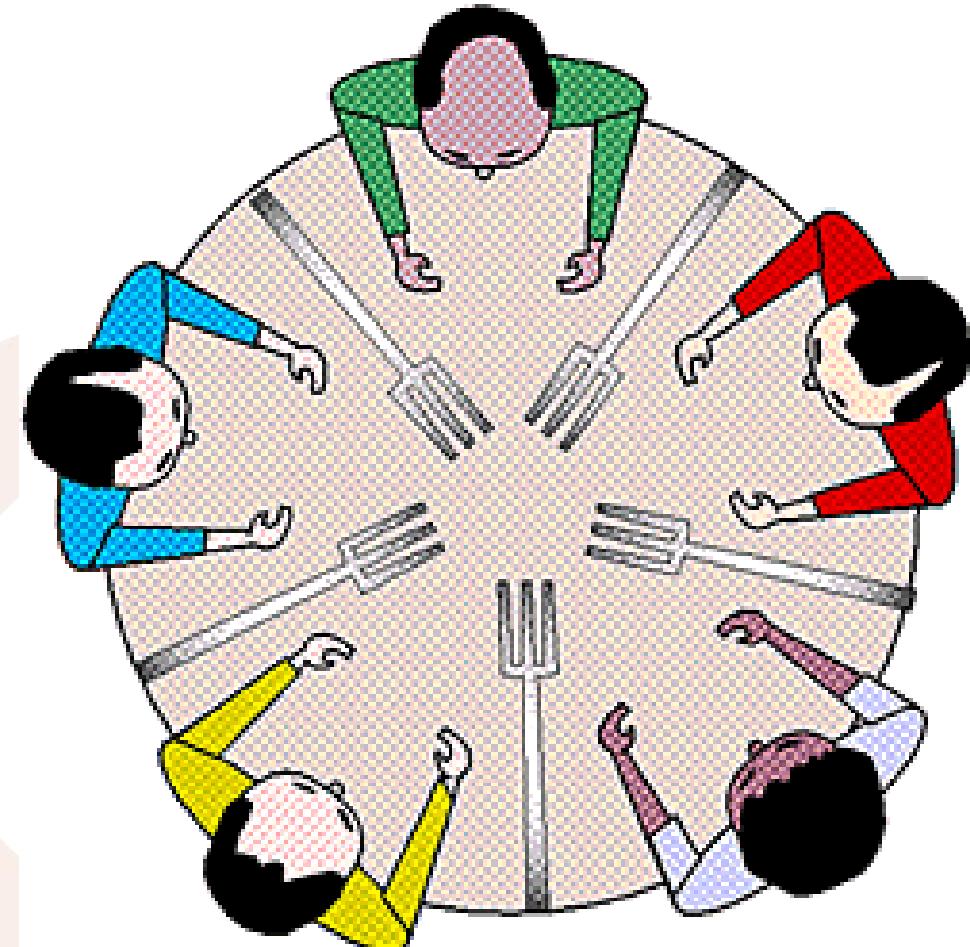
- (A) signal (mutex), wait (wrt), signal (wrt), wait (mutex)
- (B) signal (wrt), signal (mutex), wait (mutex), wait (wrt)
- (C) wait (wrt), signal (mutex), wait (mutex), signal (wrt)
- (D) signal (mutex), wait (mutex), signal (mutex), wait (mutex)

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Dining Philosopher Problem

- Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher.
- In the center of the table is a bowl of rice, and the table is laid with five single chopsticks.
- When a philosopher thinks, she does not interact with her colleagues.

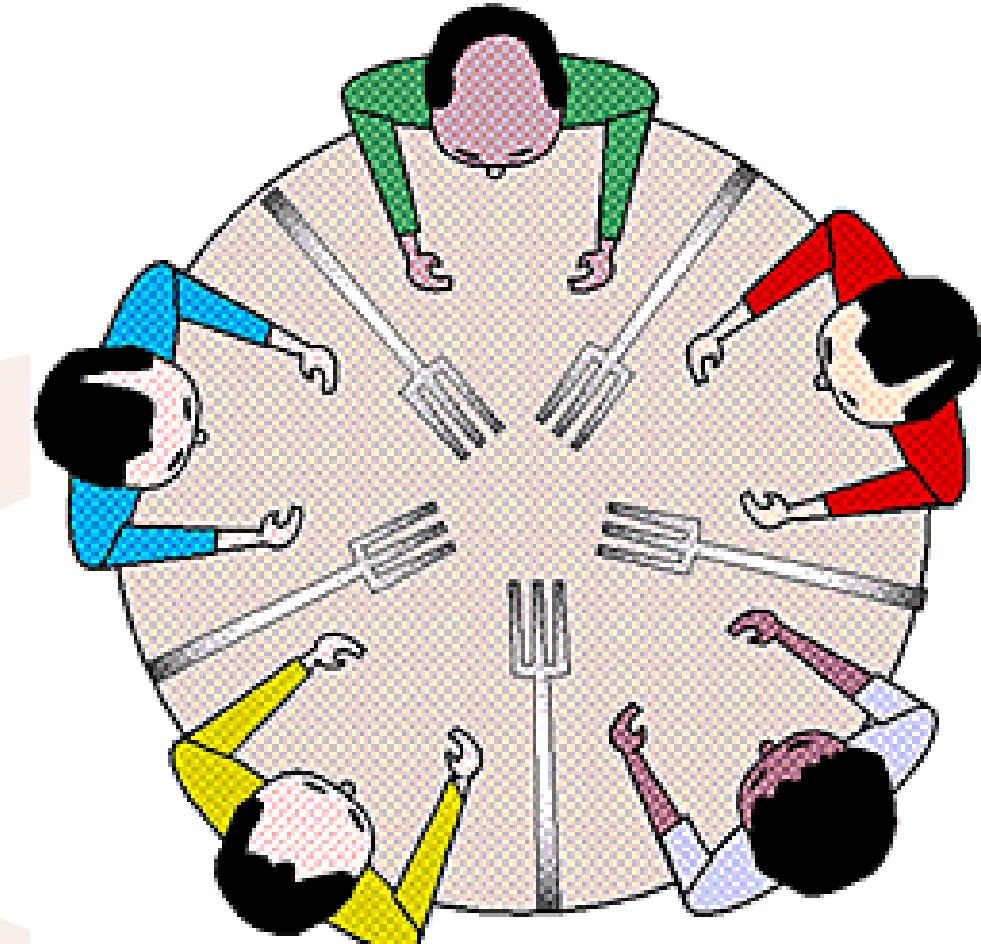


To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).

- A philosopher may pick up only one chopstick at a time. Obviously, she can't pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# Indian Chopsticks



To access all paid content get **KG Prime** at ₹

LICK HERE ↗

## Solution for Dining Philosophers

Void Philosopher (void)

{

while ( T )

{

**Thinking ( );**

wait(chopstick [i]);

wait(chopstick([(i+1)%5]));

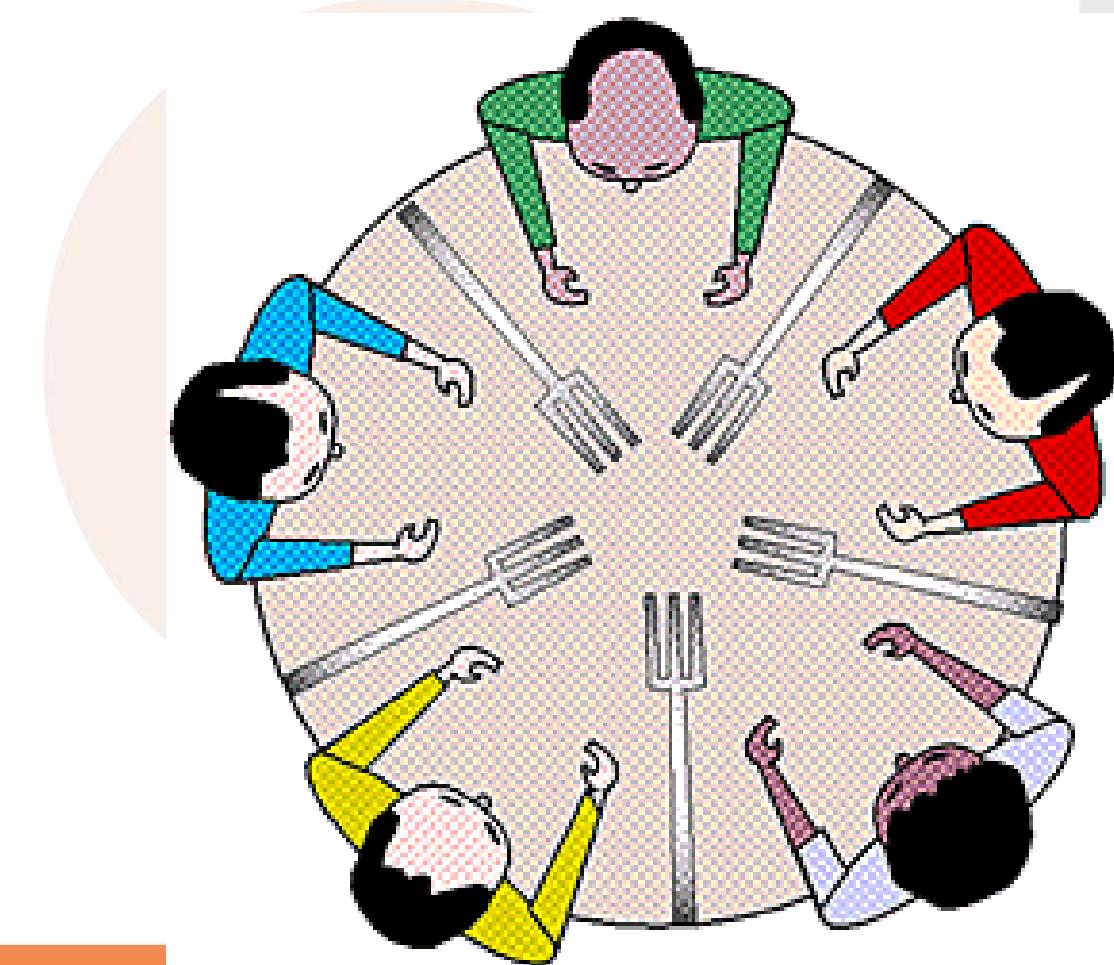
**Eat( );**

signal(chopstick [i]);

signal(chopstick([(i+1)%5]));

}

}



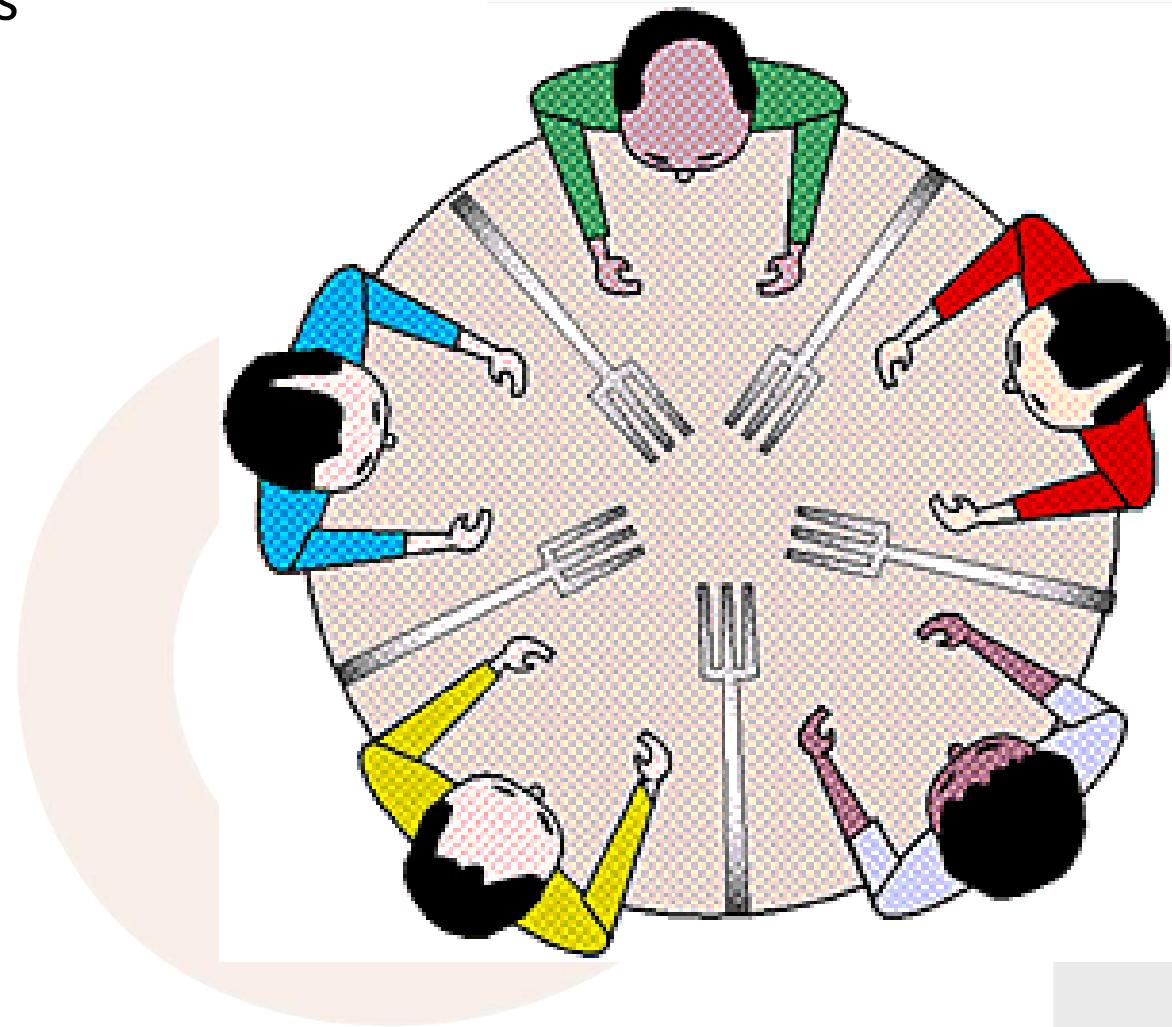
- Here we have used an array of semaphores called chopstick[]
- Solution is not valid because there is a possibility of deadlock.



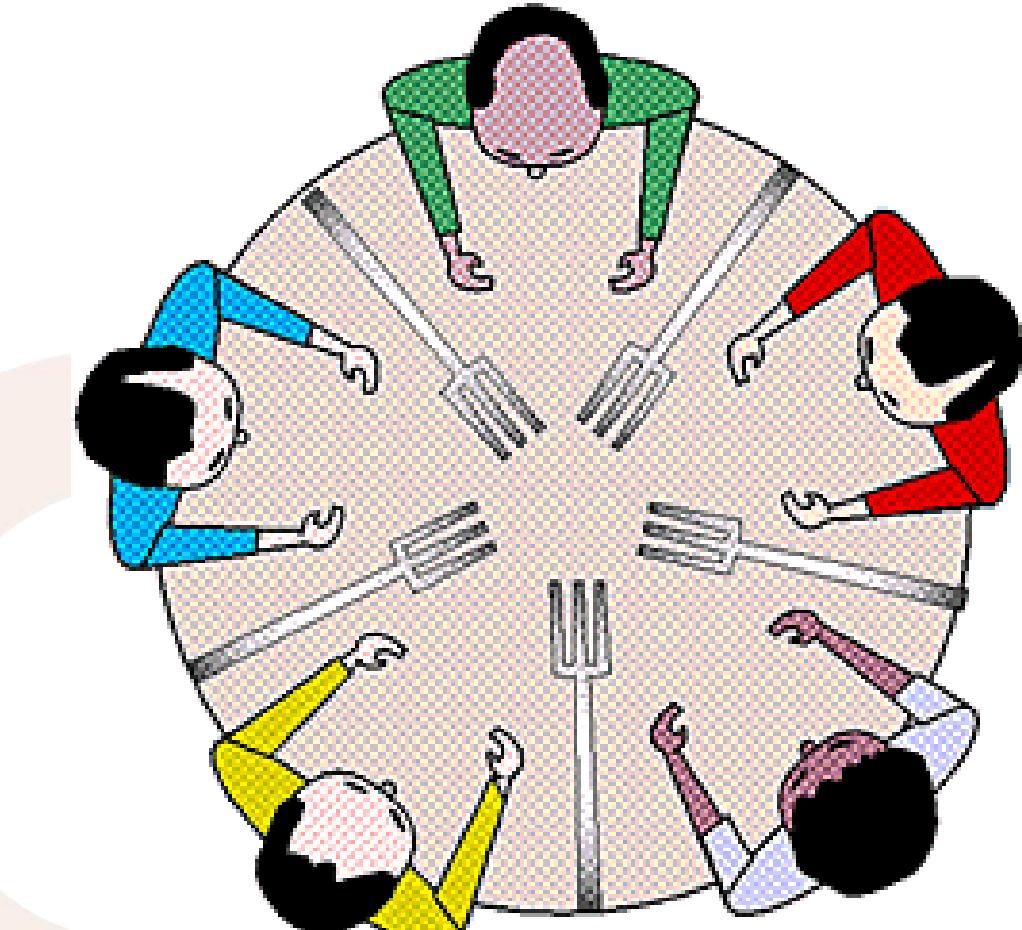
To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- The proposed solution for deadlock problem is
  - Allow at most four philosophers to be sitting simultaneously at the table.
  - Allow six chopstick to be used simultaneously at the table.
  - Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in a critical section).

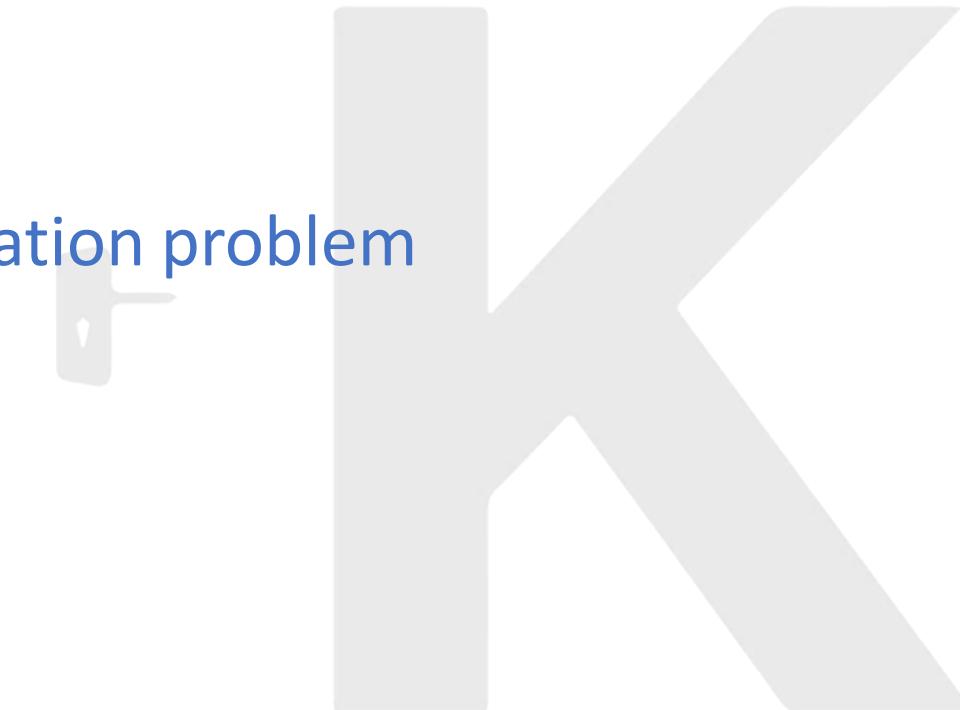


- One philosopher picks up her right chopstick first and then left chop stick, i.e. reverse the sequence of any philosopher.
- Odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.



Q Dining Philosopher's problem is a: (NET-JUNE-2015)

a) Producer - consumer problem



c) Starvation problem

b) Classical IPC problem



d) Synchronization primitive

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

**Q** Let  $m[0] \dots m[4]$  be mutexes (binary semaphores) and  $P[0] \dots P[4]$  be processes. Suppose each process  $P[i]$  executes the following:

`wait (m[i]); wait(m[(i+1) mode 4]);`

-----

`release (m[i]); release (m[(i+1)mod 4]);`

This could cause: **(GATE-2000) (1 Marks)**

(A) Thrashing

(C) Starvation, but not deadlock

(B) Deadlock

(D) None of the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A solution to the Dining Philosophers Problem which avoids deadlock is: (GATE-1996) (2 Marks)

- (A) ensure that all philosophers pick up the left fork before the right fork
- (B) ensure that all philosophers pick up the right fork before the left fork
- (C) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork
- (D) None of the above

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Types of Semaphore

- **Binary Semaphores:** The value of a binary semaphore can range only between 0 and 1.
- **Counting Semaphores:** can range over an unrestricted domain Counting semaphore can range over an unrestricted domain. i.e.  $-\infty$  to  $+\infty$ .
- Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the number of resources available.
- Each process that wishes to use a resource performs a `wait()` operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a `signal()` operation (incrementing the count).
- When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

## Implementation of semaphore

- This simple implementation of semaphore with this wait(S) and signal(S) function suffer from busy waiting.



- **Modified Wait**

- To overcome the need for busy waiting, we can modify the definition of the wait() and signal() operations as follows: When a process executes the wait() operation and finds that the semaphore value is not  $>0$ , it must wait. However, rather than engaging in busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.

```
wait(semaphore *S)
{
    S->value--;
    if (S->value < 0)
    {
        add this process to S->list;
        block();
    }
}
```

- **Modified Signal**

- A process that is blocked, waiting on a semaphore  $S$ , should be restarted when some other process executes a `signal()` operation. The process is restarted by a `wakeup()` operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue. (The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU-scheduling algorithm.)

```
Signal(semaphore *S)
{
    S->value++;
    if (S->value <= 0)
    {
        remove a process P from S->list;
        wakeup(P);
    }
}
```



To implement semaphores under this definition, we define a semaphore as follows:

```
typedef struct
{
    int value;
    struct process *list;
} semaphore;
```

Each semaphore has an integer value and a list of processes list.

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- The block() operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls.
- Note that in this implementation, semaphore values may be negative, whereas semaphore values are never negative under the classical definition of semaphores with busy waiting. If a semaphore value is negative, its magnitude is the number of processes waiting on that semaphore. This fact results from switching the order of the decrement and the test in the implementation of the wait() operation.

- The list of waiting processes can be easily implemented by a link field in each process control block (PCB). Each semaphore contains an integer value and a pointer to a list of PCBs. One way to add and remove processes from the list so as to ensure bounded waiting is to use a FIFO queue, where the semaphore contains both head and tail pointers to the queue.



Q At a particular time of computation, the value of a counting semaphore is 10. Then 12 P operations and “x” V operations were performed on this semaphore. If the final value of semaphore is 7, x will be: (NET-JULY-2018)

- a) 8
- b) 9
- c) 10
- d) 11



**Q** There are three processes  $P_1$ ,  $P_2$  and  $P_3$  sharing a semaphore for synchronizing a variable. Initial value of semaphore is one. Assume that negative value of semaphore tells us how many processes are waiting in queue. Processes access the semaphore in following order : **(NET-JAN-2017)**

- (1)  $P_2$  needs to access
- (2)  $P_1$  needs to access
- (3)  $P_3$  needs to access
- (4)  $P_2$  exits critical section
- (5)  $P_1$  exits critical section

The final value of semaphore will be:

- a) 0
- b) 1
- c) -1
- d) -2

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 20 P(S) operations and 12 V(S) operations are issued in some order. The largest initial value of S for which at least one P(S) operation will remain blocked is \_\_\_\_\_. (GATE-2016) (2 Marks)



Q A semaphore count of negative n means ( $s=-n$ ) that the queue contains ..... waiting processes. (NET-DEC-2009)

a)  $n + 1$

b)  $n$

c)  $n - 1$

d) 0



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** The P and V operations on counting semaphores, where  $s$  is a counting semaphore, are defined as follows:

**P(s)** :  $s = s - 1;$

if ( $s < 0$ ) then wait;

**V(s)** :  $s = s + 1;$

if ( $s \leq 0$ ) then wakeup a process waiting on  $s$ ;

Assume that  $P_b$  and  $V_b$  the wait and signal operations on binary semaphores are provided. Two binary semaphores  $X_b$  and  $Y_b$  are used to implement the semaphore operations  $P(s)$  and  $V(s)$  as follows:

The initial values of  $X_b$  and  $Y_b$  are respectively (GATE-2008)

**(2 Marks)**

(A) 0 and 0

(B) 0 and 1

(C) 1 and 0

(D) 1 and 1

<b>P(s)</b>	$P_b(X_b);$ $s = s - 1;$ if ( $s < 0$ ) { $V_b(X_b);$ $P_b(Y_b);$ } else $V_b(X_b);$
<b>V(s)</b>	$P_b(X_b);$ $s = s + 1;$ if ( $s \leq 0$ ) $V_b(Y_b);$ $V_b(X_b);$

Q A counting semaphore was initialized to 10. Then 6P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is **(GATE-1998) (1 Marks)**

a) 0

b) 8

c) 10

d) 12

**Q** At a particular time of computation the value of a counting semaphore is 7. Then 20 P operations and 15V operations were completed on this semaphore. If the new value of semaphore is will be **(GATE-1992) (1 Marks)**

(A) 42

(B) 2

(c) 7

(D) 12

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Disable Interrupt

- This could be a hardware solution where process have a privilege instruction, i.e. before entering into critical section, process will disable all the interrupts and at the time of exit, it again enable interrupts.
- This solution is only used by OS, as if some user process enter into critical section, then can block the entire system.

```
Pi()
{
    While(T)
    {
        Initial Section
        Entry Section//Disable interrupt
        Critical Section
        Exit Section//Enable interrupt
        Remainder Section
    }
}
```

- Unfortunately, this solution is not as feasible in a multiprocessor environment. Disabling interrupts on a multiprocessor can be time consuming, since the message is passed to all the processors. This message passing delays entry into each critical section, and system efficiency decreases.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

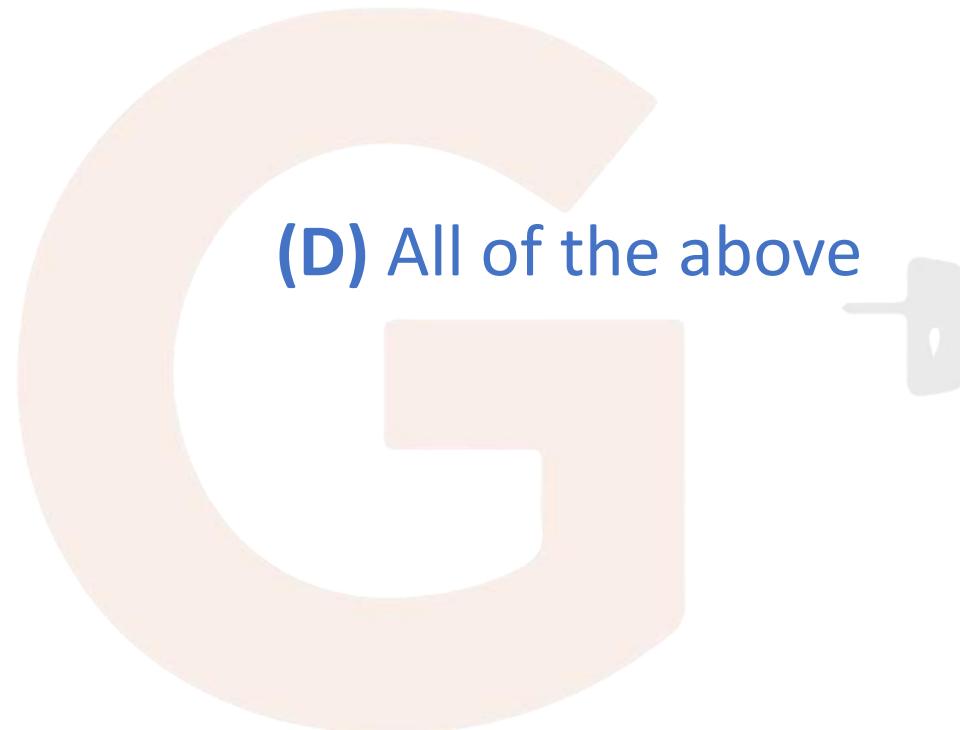
Q In an operating system, indivisibility of operation means: (NET-DEC-2015)

(A) Operation is interruptible



(C) Processor cannot be pre-empted

(B) Race – condition may occur



(D) All of the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Hardware Type Solution **Test** and **Set**

- Software-based solutions such as Peterson's are not guaranteed to work on modern computer architectures. In the following discussions, we explore several more solutions to the critical-section problem using techniques ranging from hardware to software, all these solutions are based on the premise of locking —that is, protecting critical regions through the use of locks.
- The critical-section problem could be solved simply in a single-processor environment if we could prevent interrupts from occurring while a shared variable was being modified.

Boolean test and set (Boolean \*target)

```
{  
    Boolean rv = *target;  
    *target = true;  
    return rv;  
}
```

While(1)

```
{  
    while (test and set(&lock));  
    /* critical section */  
    lock = false;  
    /* remainder section */  
}
```

- Many modern computer systems therefore provide special hardware instructions that allow us either to test and modify the content of a word atomically —that is, as one uninterruptible unit. We can use these special instructions to solve the critical-section problem in a relatively simple manner.
- The important characteristic of this instruction is that it is executed atomically. Thus, if two test and set() instructions are executed simultaneously (each on a different CPU), they will be executed sequentially in some arbitrary order.

Q Fetch\_And\_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock(L)
{
    while (Fetch_And_Add(L,1))
        L = 1;
}
```

```
ReleaseLock (L)
{
    L = 0;
}
```

This implementation (GATE-2012) (2 Marks)

- (A) fails as L can overflow
- (B) fails as L can take on a non-zero value when the lock is actually available
- (C) works correctly but may starve some processes
- (D) works correctly without starvation

**Q** The enter\_CS() and leave\_CS() functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
    while test-and-set(X) ;
}
```

```
void leave_CS(X)
{
    X = 0;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements: **(GATE-2009) (2 Marks)**

- I. The above solution to CS problem is deadlock-free
- II. The solution is starvation free.
- III. The processes enter CS in FIFO order.
- IV. More than one process can enter CS at the same time.

Which of the above statements is TRUE?

- (A) I only
- (B) I and II
- (C) II and III
- (D) IV only

**Q** The atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. consider the following implementation of P and V functions on a binary semaphore S. Which one of the following is true? **(GATE-2006) (2 Marks)**

- (A) The implementation may not work if context switching is disabled in P
- (B) Instead of using fetch-and –set, a pair of normal load/store can be used
- (C) The implementation of V is wrong
- (D) The code does not implement a binary semaphore

```
void P (binary_semaphore *s)
{
    unsigned y;
    unsigned *x = &(s->value);
    do
    {
        fetch-and-set x, y;
    }
    while (y);
}
```

```
void V (binary_semaphore *s)
{
    S->value = 0;
}
```

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as  $S_0 = 1$ ,  $S_1 = 0$ ,  $S_2 = 0$ . **(GATE-2010) (2 Marks)**

How many times will process  $P_0$  print '0'?

- (A) At least twice
- (B) Exactly twice
- (C) Exactly thrice
- (D) Exactly once

Process $P_0$	Process $P_1$	Process $P_2$
While(true){	Wait( $S_1$ );	Wait( $S_2$ );
Wait( $S_0$ );	Release( $S_0$ );	Release( $S_0$ );
Print'0'		
Release( $S_1$ );		
Release( $S_2$ );		
}		

**Q Suppose a processor does not have any stack pointer register. Which of the following statements is true? (GATE-2001) (1 Marks)**

- (A) It cannot have subroutine call instruction**
- (B) It can have subroutine call instruction, but no nested subroutine calls**
- (C) Nested subroutine calls are possible, but interrupts are not**
- (D) All sequences of subroutine calls and also interrupts are possible**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Each Process  $P_i$ ,  $i = 1, \dots, 9$  is coded as follows (GATE-1997) (1 Marks)

repeat

P(mutex)

## {Critical section}

V(mutex)

forever

The code for  $P_{10}$  is identical except it uses  $V(mutex)$  in place of  $P(mutex)$ . What is the largest number of processes that can be inside the critical section at any moment?

- (A) 1      (B) 2      (C) 3      (D) None of above

Q Consider the following proposed solution for the critical section problem. There are  $n$  processes:  $P_0 \dots P_{(n-1)}$ . In the code, function pmax returns an integer not smaller than any of its arguments. For all  $i$ ,  $t[i]$  is initialized to zero.

Code for  $P_i$ :

```
do {  
    c[i]=1; t[i] = pmax(t[0],...,t[n-1])+1; c[i]=0;  
  
    for every j  $\neq i$  in {0,...,n-1} {  
        while (c[j]);  
        while (t[j] != 0 && t[j]<=t[i]);  
    }  
    Critical Section;  
    t[i]=0;  
    Remainder Section;  
} while (true);
```

Which one of the following is TRUE about the above solution? (GATE-2016) (2 Marks)

- (a) At most one process can be in the critical section at any time
- (b) The bounded wait condition is satisfied
- (c) The progress condition is satisfied
- (d) It cannot cause a deadlock

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q.49

Consider the following pseudocode, where S is a semaphore initialized to 5 in line#2 and counter is a shared variable initialized to 0 in line#1. Assume that the increment operation in line#7 is not atomic.

```
1. int counter = 0;  
2. Semaphore S = init(5);  
3. void parop(void)  
4. {  
5.     wait(S);  
6.     wait(S);  
7.     counter++;  
8.     signal(S);  
9.     signal(S);  
10. }
```

If five threads execute the function parop concurrently, which of the following program behavior(s) is/are possible?

- (a) The value of counter is 1 after all the threads successfully complete the execution of parop.
- (b) The value of counter is 0 after all the threads successfully complete the execution of parop.
- (c) There is a deadlock involving all the threads.
- (d) The value of counter is 5 after all the threads successfully complete the execution of parop.

Ans. (a, c, d)



CLICK HERE ↗

## Basics of Dead-Lock

- In a multiprogramming environment, several processes may compete for a finite number of resources.
- In these scenario there is a possibility of Deadlock



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**





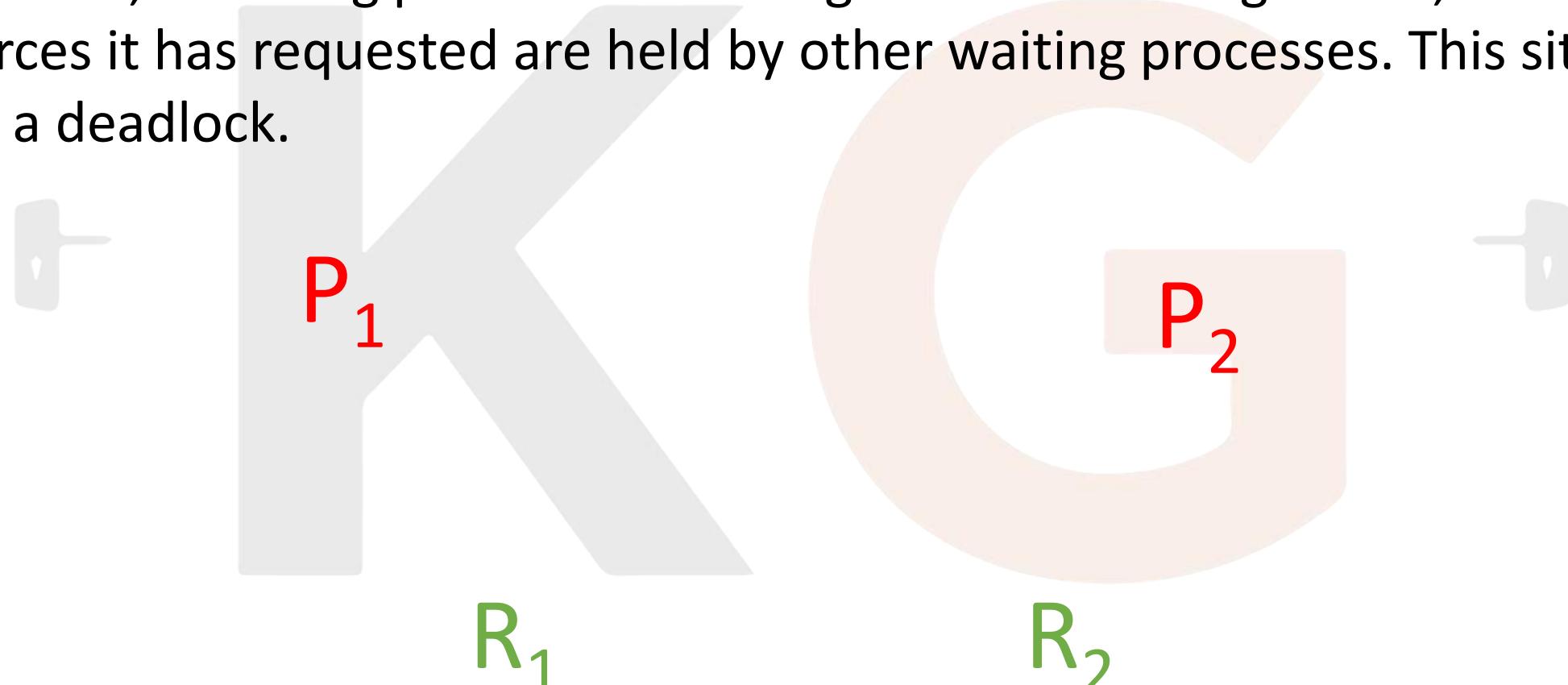
# Tax

# Services



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- A process requests resources; if the resources are not available at that time, the process enters a waiting state.
- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.



- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.
- Starvation is long waiting but deadlock is infinite waiting

Pehele aap



Pehele aap

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## System model

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request**. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- **Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release**. The process releases the resource.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Necessary conditions for deadlock

A deadlock can occur if all these 4 conditions occur in the system simultaneously.

- Mutual exclusion
- Hold and wait
- No pre-emption
- Circular wait



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- **Mutual exclusion**:- At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource.
- If another process requests that resource, the requesting process must be delayed until the resource has been released. And the resource Must be desired by more than one process.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Hold and wait:** - A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes. E.g. Plate and spoon



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **No pre-emption:** - Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



- **Circular wait**: - A set  $P_0, P_1, \dots, P_n$  of waiting processes must exist such that  $P_0$  is waiting for a resource held by  $P_1$ ,  $P_1$  is waiting for a resource held by  $P_2$ , ...,  $P_{n-1}$  is waiting for a resource held by  $P_n$ , and  $P_n$  is waiting for a resource held by  $P_0$ .







# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Deadlock Handling methods

- Prevention: - Design such protocols that there is no possibility of deadlock.
- Avoidance: - Try to avoid deadlock in run time so ensuring that the system will never enter a deadlocked state.
- Detection: - We can allow the system to enter a deadlocked state, then detect it, and recover.
- Ignorance: - We can ignore the problem altogether and pretend that deadlocks never occur in the system.

- Last solution is the one used by most operating systems, including Linux and Windows. It is then up to the application developer to write programs that handle deadlocks.
- Some researchers have argued that none of the basic approaches alone is appropriate for the entire spectrum of resource-allocation problems in operating systems.
- The basic approaches can be combined, however, allowing us to select an optimal approach for each class of resources in a system.

Q A Dead-lock in an Operating System is (NET-DEC-2010)

(A) Desirable process

(B) Undesirable process

(C) Definite waiting process

(D) All of the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Prevention

- It means designing such systems where there is no possibility of existence of deadlock. For that we have to remove one of the four necessary condition of deadlock.

Polio vaccine



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Mutual exclusion**: -In prevention approach, there is no solution for mutual exclusion as resource can't be made sharable as it is a hardware property and process also can't be convinced to do some other task.
- In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Hold & wait

- In conservative approach, process is allowed to run if & only if it has acquired all the resources.



## Hold & wait

- An alternative protocol allows a process to request resources only when it has none.
- A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

## Hold & wait

- Wait time outs we place a max time outs up to which a process can wait. After which process must release all the holding resources & exit.





# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## No pre-emption

- if a process requests some resources, we first check whether they are available. If they are, we allocate them.
- If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we pre-empt the desired resources from the waiting process and allocate them to the requesting process.
- If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be pre-empted, but only if another process requests them.
- A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were pre-empted while it was waiting.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Circular wait

- We can eliminate circular wait problem by giving a natural number mapping to every resource and then any process can request only in the increasing order and if a process wants a lower number, then process must first release all the resource larger than that number and then give a fresh request.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

- I)** Process should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.
- II)** The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers
- III)** The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers
- IV)** The resources are numbered uniquely. A process is allowed to request for resources only for a resource with resource number larger than its currently held resources

Which of the above policies can be used for preventing deadlock? **(GATE-2015) (2 Marks)**

- (A)** Any one of I and III but not II or IV
- (B)** Any one of I, III and IV but not II
- (C)** Any one of II and III but not I or IV
- (D)** Any one of I, II, III and IV

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Resources are allocated to the process on non-sharable basis is (NET-JUNE-2012)

(A) mutual exclusion  
(C) no pre-emption

(B) hold and wait  
(D) circular wait

Q Which of the following is scheme to deal with deadlock? (NET-JUNE-2012)

- (A) Time out
- (B) Time in
- (C) Both (A) & (B)
- (D) None of the above

**Q** An operating system implements a policy that requires a process to release all resources before making a request for another resource. Select the TRUE statement from the following: **(GATE-2008) (2 Marks)**

- (A) Both starvation and deadlock can occur
- (B) Starvation can occur but deadlock cannot occur
- (C) Starvation cannot occur but deadlock can occur
- (D) Neither starvation nor deadlock can occur

**Q Which of the following is NOT a valid deadlock prevention scheme?  
(GATE-2000) (2 Marks)**

- (A) Release all resources before requesting a new resource**
- (B) Number the resources uniquely and never request a lower numbered resource than the last one requested.**
- (C) Never request a resource after releasing any resource**
- (D) Request and all required resources be allocated before execution.**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a maximum of  $K$  instances. Resource instances can be requested and released only one at a time. The largest value of  $K$  that will always avoid deadlock is \_\_\_\_\_.  
**(GATE-2018) (1 Marks)**



Q Consider a system having 'm' resources of the same type. These resources are shared by three processes  $P_1$ ,  $P_2$  and  $P_3$  which have peak demands of 2, 5 and 7 resources respectively. For what value of 'm' deadlock will not occur? (NET-AUG-2016)

- a) 70
- b) 14
- c) 13
- d) 7

Q A system has 6 identical resources and N processes competing for them. Each process can request at most 2 resources. Which one of the following values of N could lead to a deadlock? (GATE-2015) (1 Marks)

- a) 1
- b) 2
- c) 3
- d) 6

Q A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is (GATE-2014) (2 Marks)

- (A) 6
- (B) 7
- (C) 8
- (D) 9

Q Consider a system having  $m$  resources of the same type. These resources are shared by 3 processes A, B and C which have peak demands of 3, 4 and 6 respectively. For what value of  $m$  deadlock will not occur? (NET-DEC-2012)

- (A) 7
- (B) 9
- (C) 10
- (D) 13

Q A computer has 6 tape drives with 'n' processes competing for them. Each process may need two drives. For which values of 'n' is the system deadlock free? (NET-JUNE-2008)

(A) 1

(B) 2

(C) 3

(D) 6

**Q** Suppose  $n$  processes,  $P_1, \dots, P_n$  share  $m$  identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process  $P_i$  is  $S_i$ , where  $S_i > 0$ . Which one of the following is a sufficient condition for ensuring that deadlock does not occur? **(GATE-2005) (2 Marks)**

- a)  $\forall_i, S_i < m$
- b)  $\forall_i, S_i < n$
- c)  $\sum_{i=1}^n S_i < (m + n)$
- d)  $\sum_{i=1}^n S_i < (m * n)$

Q A computer has six tape drives, with  $n$  processes competing for them. Each process may need two drives. What is the maximum value of  $n$  for the system to be deadlock free? (GATE-1998) (2 Marks)

- (A) 6
- (B) 5
- (C) 4
- (D) 3

Q An operating system contains 3 user processes each requiring 2 units of resource  $R$ . The minimum number of units of  $R$  such that no deadlocks will ever arise is (GATE-1997) (2 Marks)

- (A) 3
- (B) 5
- (C) 4
- (D) 6

**Q** Consider a system having  $m$  resources of the same type. These resources are shared by 3 processes A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of  $m$  deadlock will not occur? (GATE-1993) (2 Marks)

- (a) 7
- (b) 9
- (c) 10
- (d) 13
- (e) 15

Q A computer system has 6 tape drives, with  $n$  process completing for them. Each process may need 3 tape drives. The maximum value of  $n$  for which the system is guaranteed to be deadlock free is (GATE-1992) (2 Marks)

- (a) 2
- (b) 3
- (c) 4
- (d) 1



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Problem with Prevention:** - Different deadlock Prevention approach put different type of restrictions or conditions on the processes and resources Because of which system becomes slow and resource utilization and reduced system throughput.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Prevention



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Avoidance



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- So, in order to avoid deadlock in run time, System try to maintain some books like a banker, whenever someone ask for a loan(resource), it is granted only when the books allow.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Avoidance

- To avoid deadlocks we require additional information about how resources are to be requested. which resources a process will request and use during its lifetime i.e. maximum number of resources of each type that it may need.
- With this additional knowledge, the operating system can decide for each request whether process should wait or not.

	Max Need		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

	Allocation		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

	Current Need		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

System Max		
E	F	G

Available		
E	F	G

	Max Need		
	E	F	G
P <sub>0</sub>	4	3	1
P <sub>1</sub>	2	1	4
P <sub>2</sub>	1	3	3
P <sub>3</sub>	5	4	1

	Allocation		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

	Current Need		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

System Max		
E	F	G
8	4	6

Available		
E	F	G

	Max Need		
	E	F	G
P <sub>0</sub>	4	3	1
P <sub>1</sub>	2	1	4
P <sub>2</sub>	1	3	3
P <sub>3</sub>	5	4	1

	Allocation		
	E	F	G
P <sub>0</sub>	1	0	1
P <sub>1</sub>	1	1	2
P <sub>2</sub>	1	0	3
P <sub>3</sub>	2	0	0

	Current Need		
	E	F	G
P <sub>0</sub>			
P <sub>1</sub>			
P <sub>2</sub>			
P <sub>3</sub>			

System Max		
E	F	G
8	4	6

Available		
E	F	G

	Max Need		
	E	F	G
P <sub>0</sub>	4	3	1
P <sub>1</sub>	2	1	4
P <sub>2</sub>	1	3	3
P <sub>3</sub>	5	4	1

	Allocation		
	E	F	G
P <sub>0</sub>	1	0	1
P <sub>1</sub>	1	1	2
P <sub>2</sub>	1	0	3
P <sub>3</sub>	2	0	0

	Current Need		
	E	F	G
P <sub>0</sub>	3	3	0
P <sub>1</sub>	1	0	2
P <sub>2</sub>	0	3	0
P <sub>3</sub>	3	4	1

System Max		
E	F	G
8	4	6

Available		
E	F	G
3	3	0



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- A deadlock-avoidance algorithm dynamically examines the resource-allocation state.
- The resource- allocation state is defined by the number of available and allocated resources and the maximum demands of the processes before allowing that request first

	Max Need		
	E	F	G
P <sub>0</sub>	4	3	1
P <sub>1</sub>	2	1	4
P <sub>2</sub>	1	3	3
P <sub>3</sub>	5	4	1

	Allocation		
	E	F	G
P <sub>0</sub>	1	0	1
P <sub>1</sub>	1	1	2
P <sub>2</sub>	1	0	3
P <sub>3</sub>	2	0	0

	Current Need		
	E	F	G
P <sub>0</sub>	3	3	0
P <sub>1</sub>	1	0	2
P <sub>2</sub>	0	3	0
P <sub>3</sub>	3	4	1

System Max		
E	F	G
8	4	6

Available		
E	F	G
3	3	0

- We check, if there exist “some sequence in which we can satisfy demand of every process without going into deadlock, if yes, this sequence is called safe sequence” and request can be allowed. Otherwise there is a possibility of going into deadlock.

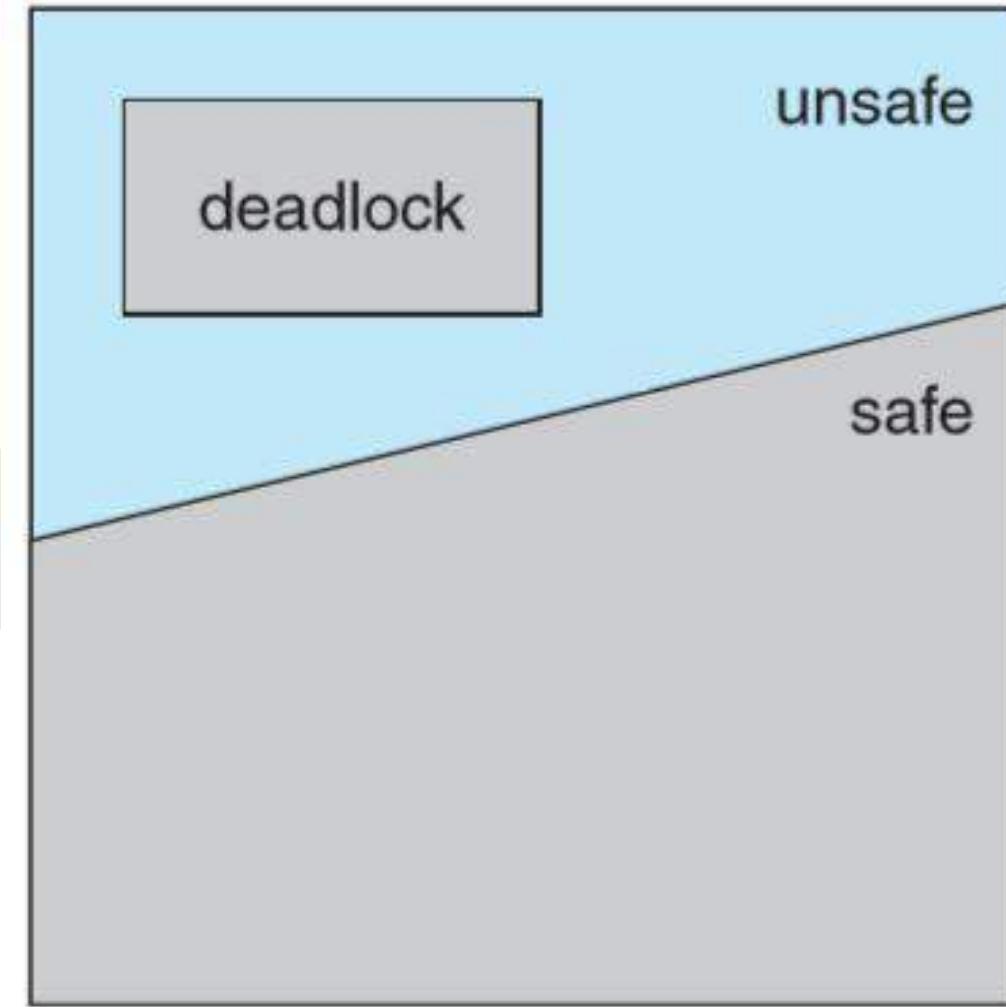
	Max Need		
	E	F	G
P <sub>0</sub>	4	3	1
P <sub>1</sub>	2	1	4
P <sub>2</sub>	1	3	3
P <sub>3</sub>	5	4	1

	Allocation		
	E	F	G
P <sub>0</sub>	1	0	1
P <sub>1</sub>	1	1	2
P <sub>2</sub>	1	0	3
P <sub>3</sub>	2	0	0

	Current Need		
	E	F	G
P <sub>0</sub>	3	3	0
P <sub>1</sub>	1	0	2
P <sub>2</sub>	0	3	0
P <sub>3</sub>	3	4	1

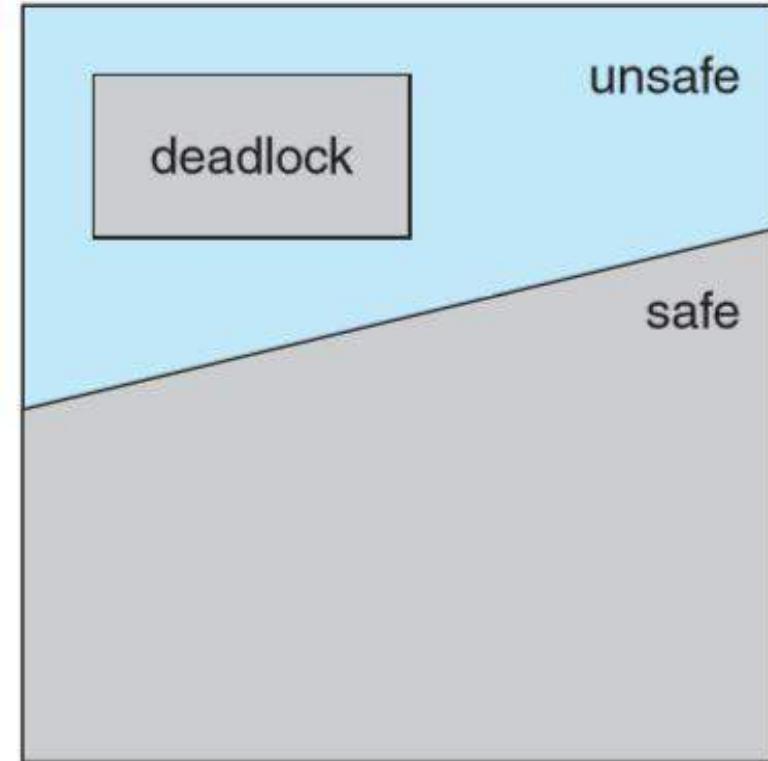
System Max		
E	F	G
8	4	6

Available		
E	F	G
3	3	0



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Given the concept of a safe state, we can define avoidance algorithms that ensure that the system will never deadlock.
- The idea is simply to ensure that the system will always remain in a safe state.
- Initially, the system is in a safe state. Whenever a process requests a resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait.
- The request is granted only if the allocation leaves the system in a safe state.





# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** A single processor system has three resource types X, Y and Z, which are shared by three processes. There are 5 units of each resource type. Consider the following scenario, where the column alloc denotes the number of units of each resource type allocated to each process, and the column request denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST? (GATE-2007) (2 Marks)

	Allocation			request		
	X	Y	Z	X	Y	Z
$P_0$	1	2	1	1	0	3
$P_1$	2	0	1	0	1	2
$P_2$	2	2	1	1	2	0

- (A)  $P_0$
- (B)  $P_1$
- (C)  $P_2$
- (D) None of the above, since the system is in a deadlock

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Banker's Algorithm

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

**Available**: A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type  $R_j$  are available.

Available		
E	F	G
3	3	0

**Max**: An  $n*m$  matrix defines the maximum demand of each process. If Max[i][j] equals k, then process  $P_i$  may request at most k instances of resource type  $R_j$ .

	Max Need		
	E	F	G
$P_0$	4	3	1
$P_1$	2	1	4
$P_2$	1	3	3
$P_3$	5	4	1

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Allocation:** An  $n*m$  matrix defines the number of resources of each type currently allocated to each process. If  $\text{Allocation}[i][j]$  equals  $k$ , then process  $P_i$  is currently allocated  $k$  instances of resource type  $R_j$ .

	Allocation		
	E	F	G
$P_0$	1	0	1
$P_1$	1	1	2
$P_2$	1	0	3
$P_3$	2	0	0

**Need/Demand/Requirement:** An  $n*m$  matrix indicates the remaining resource need of each process. If  $\text{Need}[i][j]$  equals  $k$ , then process  $P_i$  may need  $k$  more instances of resource type  $R_j$  to complete its task. Note that  $\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$ . These data structures vary over time in both size and value.

	Current Need		
	E	F	G
$P_0$	3	3	0
$P_1$	1	0	2
$P_2$	0	3	0
$P_3$	3	4	1

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

- 1- Let **Work** and **Finish** be vectors of length  $m$  and  $n$ , respectively. Initialize **Work** = **Available** and **Finish**[ $i$ ] = false for  $i = 0, 1, \dots, n - 1$ .
- 2- Find an index  $i$  such that both  
    **Finish**[ $i$ ] == false  
    **Need** <sub>$i$</sub>   $\leq$  **Work**  
    If no such  $i$  exists, go to step 4.
- 3- **Work** = **Work** + **Allocation** <sub>$i$</sub>   
    **Finish**[ $i$ ] = true  
    Go to step 2.
- 4- If **Finish**[ $i$ ] == true for all  $i$ , then the system is in a safe state.

This algorithm may require an order of  $m * n^2$  operations to determine whether a state is safe.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Resource-Request Algorithm

Next, we describe the algorithm for determining whether requests can be safely granted. Let  $\text{Request}_i$  be the request vector for process  $P_i$ . If  $\text{Request}_i[j] = k$ , then process  $P_i$  wants  $k$  instances of resource type  $R_j$ . When a request for resources is made by process  $P_i$ , the following actions are taken:

1- If  $\text{Request}_i \leq \text{Need}_i$ , go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2- If  $\text{Request}_i \leq \text{Available}$ , go to step 3. Otherwise,  $P_i$  must wait, since the resources are not available.

3- Have the system pretend to have allocated the requested resources to process  $P_i$  by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

If the resulting resource-allocation state is safe, the transaction is completed, and process  $P_i$  is allocated its resources. However, if the new state is unsafe, then  $P_i$  must wait for  $\text{Request}_i$ , and the old resource-allocation state is restored.

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the following snapshot of a system running  $n$  concurrent processes. Process  $i$  is holding  $X_i$  instances of a resource  $R$ ,  $1 \leq i \leq n$ . Assume that all instances of  $R$  are currently in use. Further, for all  $i$ , process  $i$  can place a request for at most  $Y_i$  additional instances of  $R$  while holding the  $X_i$  instances it already has. Of the  $n$  processes, there are exactly two processes  $p$  and  $q$  such that  $Y_p = Y_q = 0$ . Which one of the following conditions guarantees that no other process apart from  $p$  and  $q$  can complete execution? (GATE-2019) (2 Marks)

- a)  $X_p + X_q < \text{Min}\{Y_k \mid 1 \leq k \leq n, k \neq p, k \neq q\}$
- b)  $X_p + X_q < \text{Max}\{Y_k \mid 1 \leq k \leq n, k \neq p, k \neq q\}$
- c)  $\text{Min}(X_p, X_q) \geq \text{Min}\{Y_k \mid 1 \leq k \leq n, k \neq p, k \neq q\}$
- d)  $\text{Min}(X_p, X_q) \leq \text{Max}\{Y_k \mid 1 \leq k \leq n, k \neq p, k \neq q\}$

**Q** In a system, there are three types of resources: E, F and G. Four processes  $P_0$ ,  $P_1$ ,  $P_2$  and  $P_3$  execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example,  $Max[P_2, F]$  is the maximum number of instances of F that  $P_2$  would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

	Allocation			Max		
	E	F	G	E	F	G
$P_0$	1	0	1	4	3	1
$P_1$	1	1	2	2	1	4
$P_2$	1	0	3	1	3	3
$P_3$	2	0	0	5	4	1

Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of E and 3 instances of F are the only resources available.

From the perspective of deadlock avoidance, which one of the following is true? (GATE-2018) (2 Marks)

- (A) The system is in *safe* state
- (B) The system is not in *safe* state, but would be *safe* if one more instance of E were available
- (C) The system is not in *safe* state, but would be *safe* if one more instance of F were available
- (D) The system is not in *safe* state, but would be *safe* if one more instance of G were available

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider a system with five processes  $P_0$  through  $P_4$  and three resource types A, B and C. Resource type A has seven instances, resource type B has two instances and resource type C has six instances suppose at time  $T = 0$  we have the following allocation.

If we implement Deadlock detection algorithm, we claim that system is \_\_\_\_\_.

**(NET-NOV-2017)**

**(a) Semaphore**

**(b) Deadlock state**

**(c) Circular wait**

**(d) Not in deadlock state**

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



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Suppose there are four processes in execution with 12 instances of a Resource R in a system. The maximum need of each process and current allocation are given below:

Process	Max Need	Allocation	Current Need
$P_1$	8	3	
$P_2$	9	4	
$P_3$	5	2	
$P_4$	3	1	

With reference to current allocation, is system safe? If so, what is the safe sequence? **(NET-JULY-2016)**

- a) No
- b) Yes,  $P_1P_2P_3P_4$
- c) Yes,  $P_4P_3P_1P_2$
- d) Yes,  $P_2P_1P_3P_4$

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** ↗

Q Consider a system with twelve magnetic tape drives and three processes  $P_1$ ,  $P_2$  and  $P_3$ . Process  $P_1$  requires maximum ten tape drives, process  $P_2$  may need as many as four tape drives and  $P_3$  may need up to nine tape drives. Suppose that at time  $t_1$ , process  $P_1$  is holding five tape drives, process  $P_2$  is holding two tape drives and process  $P_3$  is holding three tape drives. At time  $t_1$ , system is in: (NET-DEC-2015)

- A) safe state
- b) unsafe state
- c) deadlocked state
- d) starvation state

Process	Max Need	Allocation	Current Need
$P_1$	10	5	
$P_2$	4	2	
$P_3$	9	3	

**Q** A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for 3 processes are shown below:  
Which of the following best describes the current state of the system? **(GATE-2014)**  
**(2 Marks)**

Process	Current Allocation	Maximum Requirement	Current Requirement	Current Available
$P_1$	3	7		
$P_2$	1	6		
$P_3$	3	5		

- (A) Safe, Deadlocked  
(C) Not Safe, Deadlocked
- (B) Safe, Not Deadlocked  
(D) Not Safe, Not Deadlocked

**Q** An operating system has 13 tape drives. There are three processes  $P_1$ ,  $P_2$  &  $P_3$ . Maximum requirement of  $P_1$  is 11 tape drives,  $P_2$  is 5 tape drives and  $P_3$  is 8 tape drives. Currently,  $P_1$  is allocated 6 tape drives,  $P_2$  is allocated 3 tape drives and  $P_3$  is allocated 2 tape drives. Which of the following sequences represent a safe state? (NET-DEC-2014)

(A)  $P_2 P_1 P_3$

(B)  $P_2 P_3 P_1$

(C)  $P_1 P_2 P_3$

(D)  $P_1 P_3 P_2$

Process	Max Need	Allocation	Current Need
$P_1$	11	6	
$P_2$	5	3	
$P_3$	8	2	



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes  $P_0$ ,  $P_1$ , and  $P_2$ . The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution. There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the current state:

	Allocation			Max		
	X	Y	Z	X	Y	Z
$P_0$	0	0	1	8	4	3
$P_1$	3	2	0	6	2	0
$P_2$	2	1	1	3	3	3

**REQ<sub>1</sub>**:  $P_0$  requests 0 units of X, 0 units of Y and 2 units of Z

**REQ<sub>2</sub>**:  $P_1$  requests 2 units of X, 0 units of Y and 0 units of Z

Which one of the following is TRUE? (GATE-2014) (2 Marks)

- (A) Only **REQ<sub>1</sub>** can be permitted.      (B) Only **REQ<sub>2</sub>** can be permitted.  
 (C) Both **REQ<sub>1</sub>** and **REQ<sub>2</sub>** can be permitted.      (D) Neither **REQ<sub>1</sub>** nor **REQ<sub>2</sub>** can be permitted

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** ↗

**Q** Consider a system with five processes  $P_0$  through  $P_4$  and three resource types  $R_1$ ,  $R_2$  and  $R_3$ . Resource type  $R_1$  has 10 instances,  $R_2$  has 5 instances and  $R_3$  has 7 instances. Suppose that at time  $T_0$ , the following snapshot of the system has been taken: (NET-JUNE-2014)

Assume that now the process  $P_1$  requests one additional instance of type  $R_1$  and two instances of resource type  $R_3$ . The state resulting after this allocation will be

- (A) Ready state      (B) Safe state      (C) Blocked state      (D) Unsafe state

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

**Q** An operating system using banker's algorithm for deadlock avoidance has ten dedicated devices (of same type) and has three processes  $P_1$ ,  $P_2$  and  $P_3$  with maximum resource requirements of 4, 5 and 8 respectively. There are two states of allocation of devices as follows: (NET-JUNE-2013)

Which of the following is correct ?

State 1	Processes	$P_1$	$P_2$	$P_3$
	Device allocated	2	3	4
State 2	Processes	$P_1$	$P_2$	$P_3$
	Device allocated	0	2	4

- (A) State 1 is unsafe and state 2 is safe.
- (B) State 1 is safe and state 2 is unsafe.
- (C) Both, state 1 and state 2 are safe.
- (D) Both, state 1 and state 2 are unsafe.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes? **(GATE-2008) (2 Marks)**

- (A) In deadlock prevention, the request for resources is always granted if the resulting state is safe
- (B) In deadlock avoidance, the request for resources is always granted if the result state is safe
- (C) Deadlock avoidance is less restrictive than deadlock prevention
- (D) Deadlock avoidance requires knowledge of resource requirements a priori

**Q** Consider the following snapshot of a system running  $n$  processes. Process  $P_i$  is holding  $X_i$  instances of a resource  $R$ ,  $1 \leq i \leq n$ . currently, all instances of  $R$  are occupied. Further, for all  $i$ , process  $i$  has placed a request for an additional  $Y_i$  instances while holding the  $X_i$  instances it already has. There are exactly two processes  $p$  and  $q$  such that  $Y_p = Y_q = 0$ . Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock? **(GATE-2006) (2 Mark)**

**(A)**  $\min(X_p, X_q) < \max(Y_k)$  where  $k \neq p$  and  $k \neq q$       **(B)**  $X_p + X_q \geq \min(Y_k)$  where  $k \neq p$  and  $k \neq q$

**(C)**  $\max(X_p, X_q) > 1$

**(D)**  $\min(X_p, X_q) > 1$

Q Banker's algorithm is used for ..... purpose. (NET-DEC-2005)

(A) Deadlock avoidance

(B) Deadlock removal

(C) Deadlock prevention

(D) Deadlock continuations

Q Bankers algorithm is for. (NET-JUNE-2005)

(A) Dead lock Prevention

(B) Dead lock Avoidance



(C) Dead lock Detection



(D) Dead lock creation

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

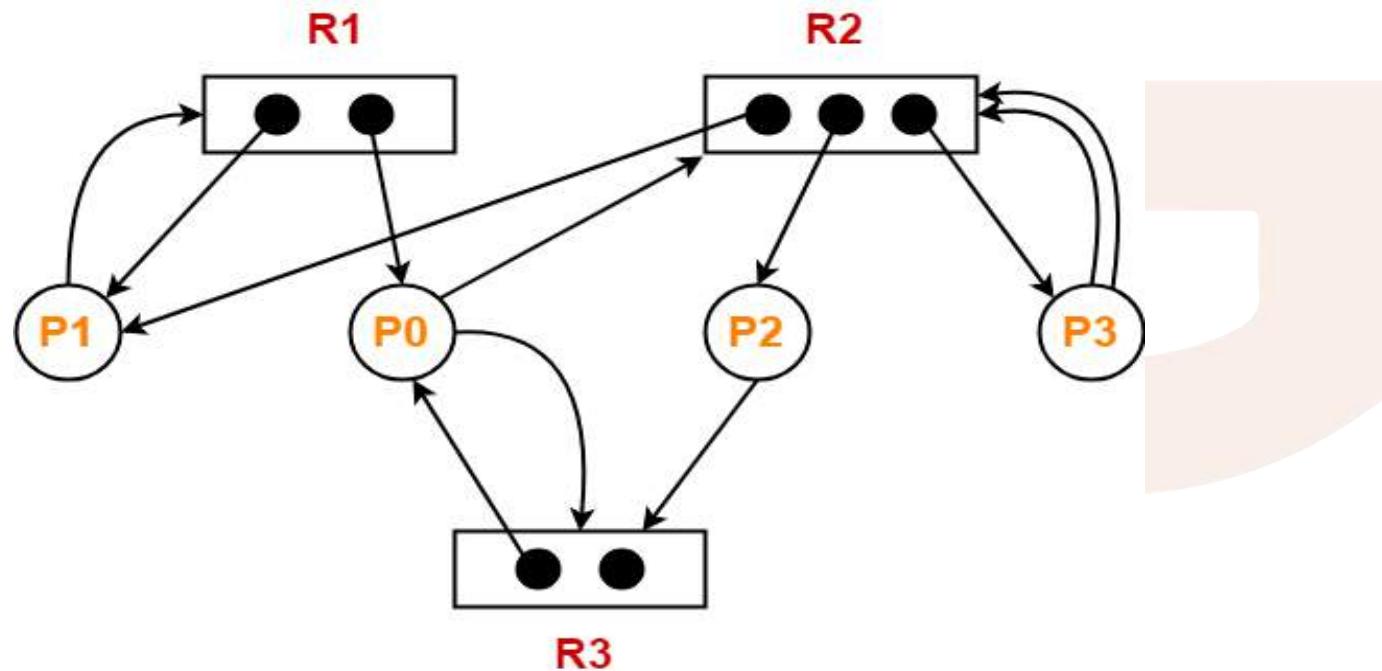


# break

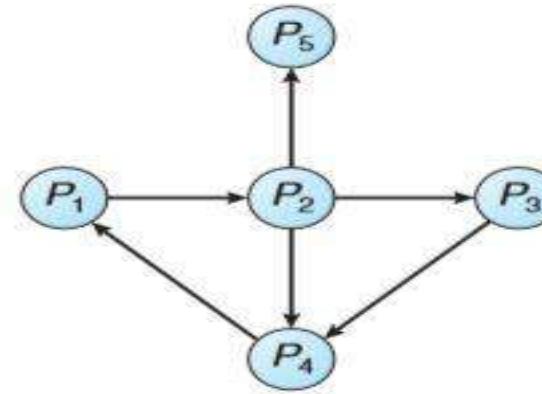
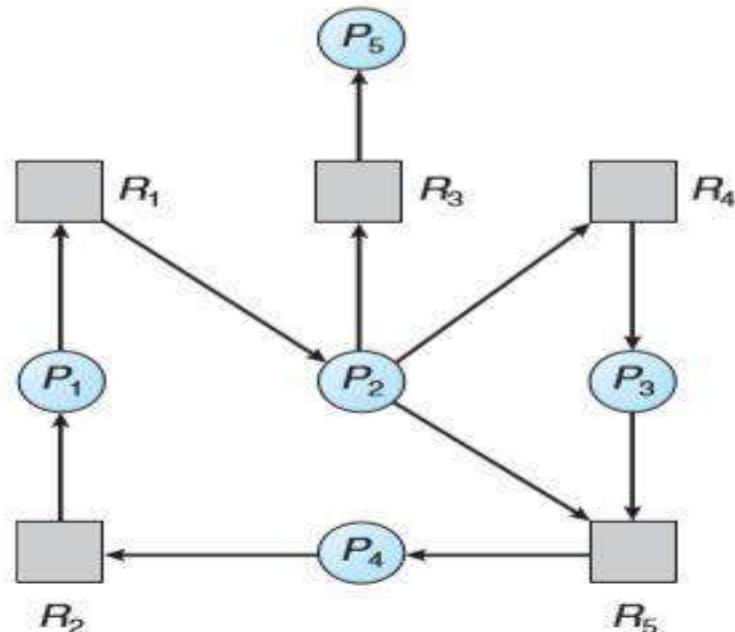
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Resource Allocation Graph

- Deadlock can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices  $V$  and a set of edges  $E$ .
- The set of vertices  $V$  is partitioned into two different types of nodes:  $P = \{P_1, P_2, \dots, P_n\}$ , the set consisting of all the active processes in the system, and  $R = \{R_1, R_2, \dots, R_m\}$ , the set consisting of all resource types in the system.



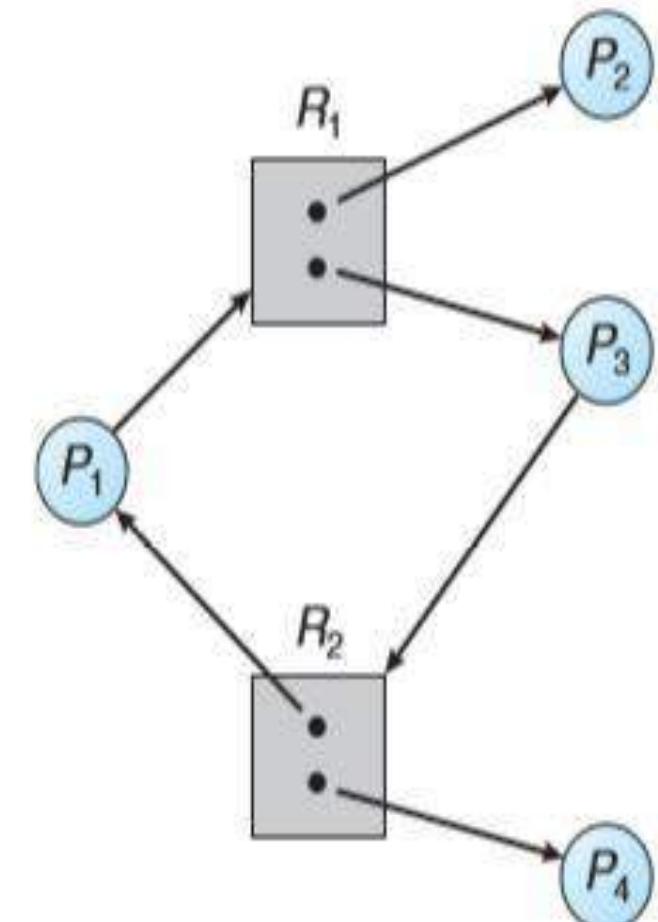
- A directed edge from process  $P_i$  to resource type  $R_j$  is denoted by  $P_i \rightarrow R_j$ ; it signifies that process  $P_i$  has requested an instance of resource type  $R_j$  and is currently waiting for that resource.
- A directed edge from resource type  $R_j$  to process  $P_i$  is denoted by  $R_j \rightarrow P_i$ ; it signifies that an instance of resource type  $R_j$  has been allocated to process  $P_i$ .
- A directed edge  $P_i \rightarrow R_j$  is called a request edge; a directed edge  $R_j \rightarrow P_i$  is called an assignment edge.



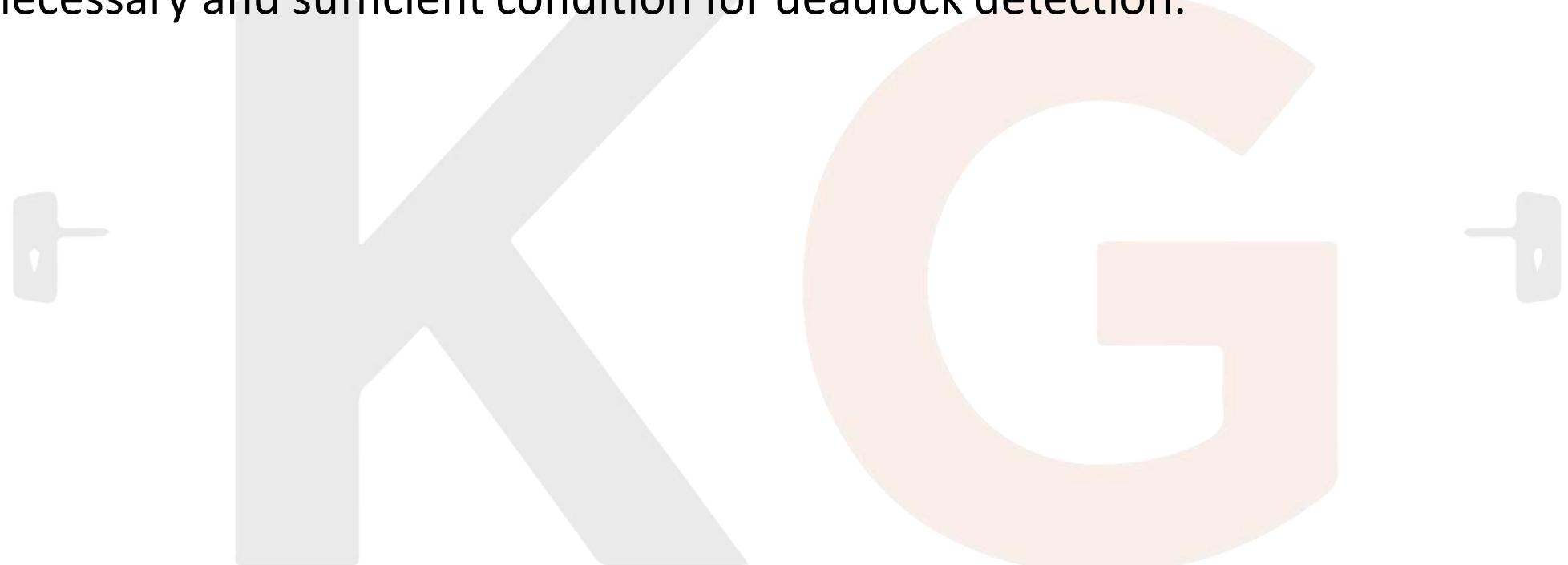
To access all paid content get **KG Prime** at ₹25/day

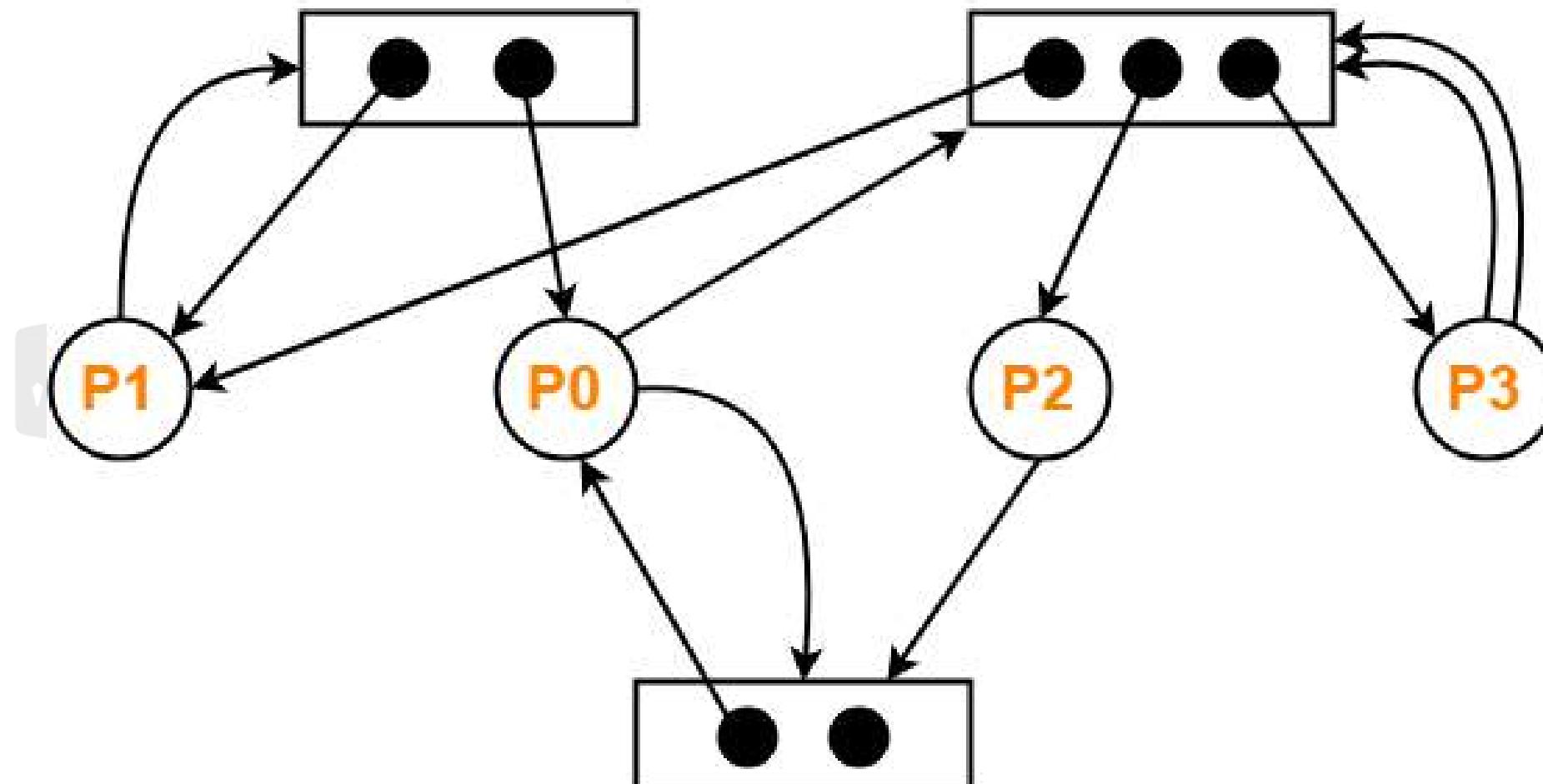
[CLICK HERE ↗](#)

- Pictorially, we represent each process  $P_i$  as a circle and each resource type  $R_j$  as a rectangle. Since resource type  $R_j$  may have more than one instance, we represent each such instance as a dot within the rectangle. Note that a request edge points to only the rectangle  $R_j$ , whereas an assignment edge must also designate one of the dots in the rectangle.
- When process  $P_i$  requests an instance of resource type  $R_j$ , a request edge is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource. As a result, the assignment edge is deleted.



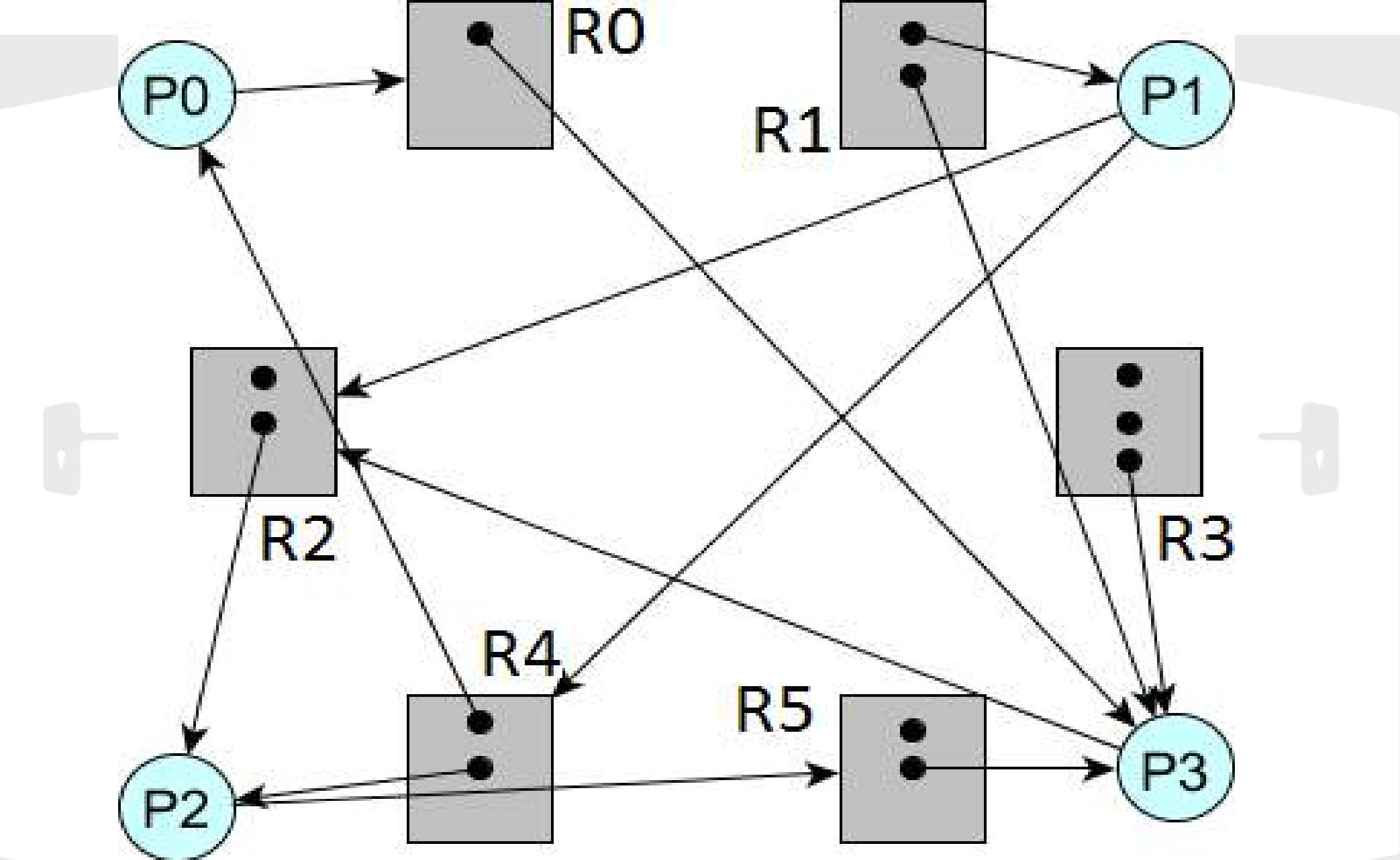
- Cycle in resource allocation graph is necessary but not sufficient condition for detection of deadlock.
- If every resource have only one resource in the resource allocation graph than detection of cycle is necessary and sufficient condition for deadlock detection.



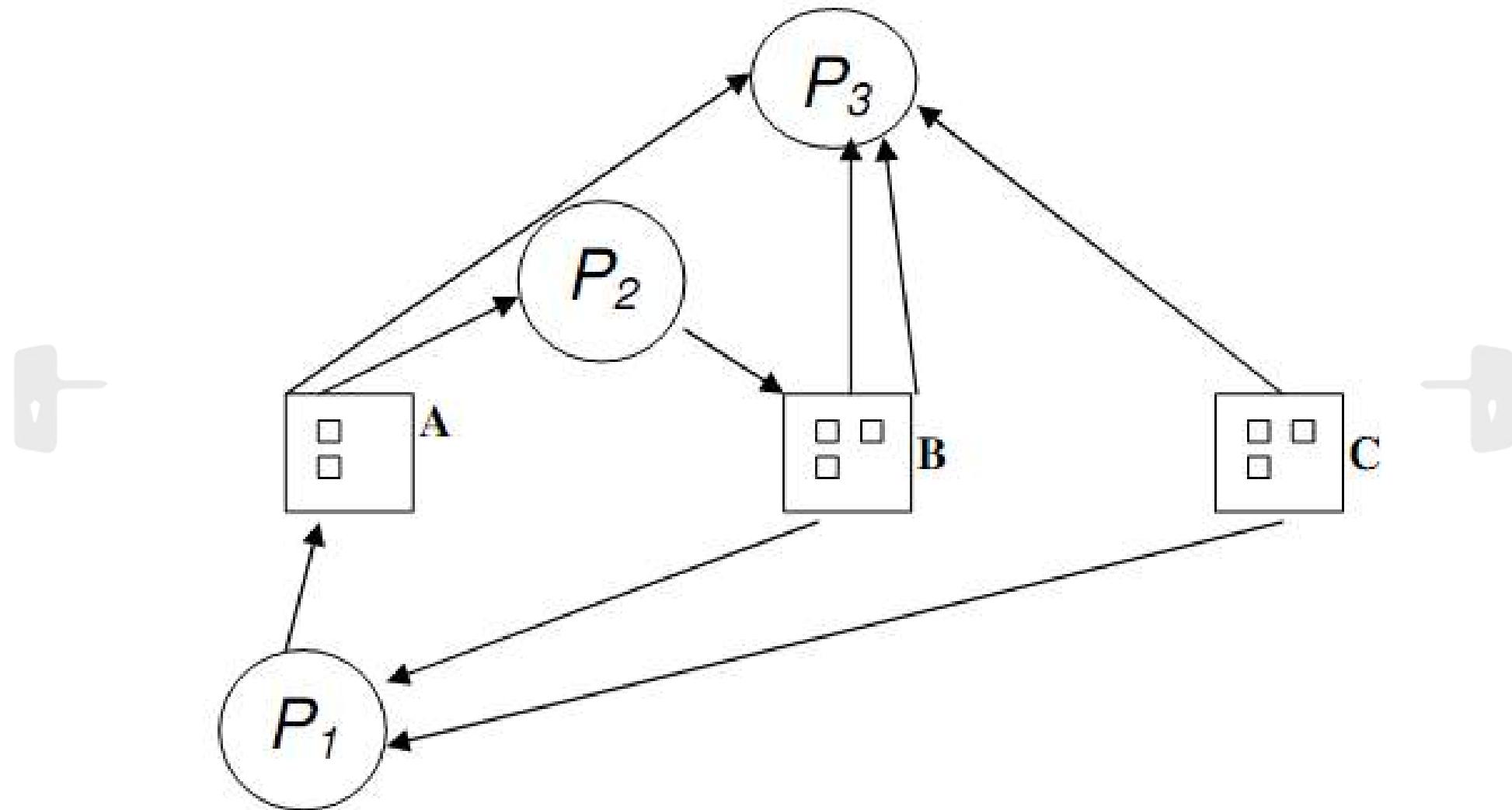


**R3**

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Deadlock detection and recovery

- Here we do not check safety and where any process request for some resources then these resources are allocated immediately, if available.
- Here there is a possibility of deadlock, which must be detected using different approaches.
  - Active Approach
  - Lazy Approach

## Resource-Request Algorithm

Next, we describe the algorithm for determining whether requests can be safely granted. Let  $\text{Request}_i$  be the request vector for process  $P_i$ . If  $\text{Request}_i[j] = k$ , then process  $P_i$  wants  $k$  instances of resource type  $R_j$ . When a request for resources is made by process  $P_i$ , the following actions are taken:

1- If  $\text{Request}_i \leq \text{Need}_i$ , go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2- If  $\text{Request}_i \leq \text{Available}$ , go to step 3. Otherwise,  $P_i$  must wait, since the resources are not available.

3- Have the system pretend to have allocated the requested resources to process  $P_i$  by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

If the resulting resource-allocation state is safe, the transaction is completed, and process  $P_i$  is allocated its resources. However, if the new state is unsafe, then  $P_i$  must wait for  $\text{Request}_i$ , and the old resource-allocation state is restored.

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

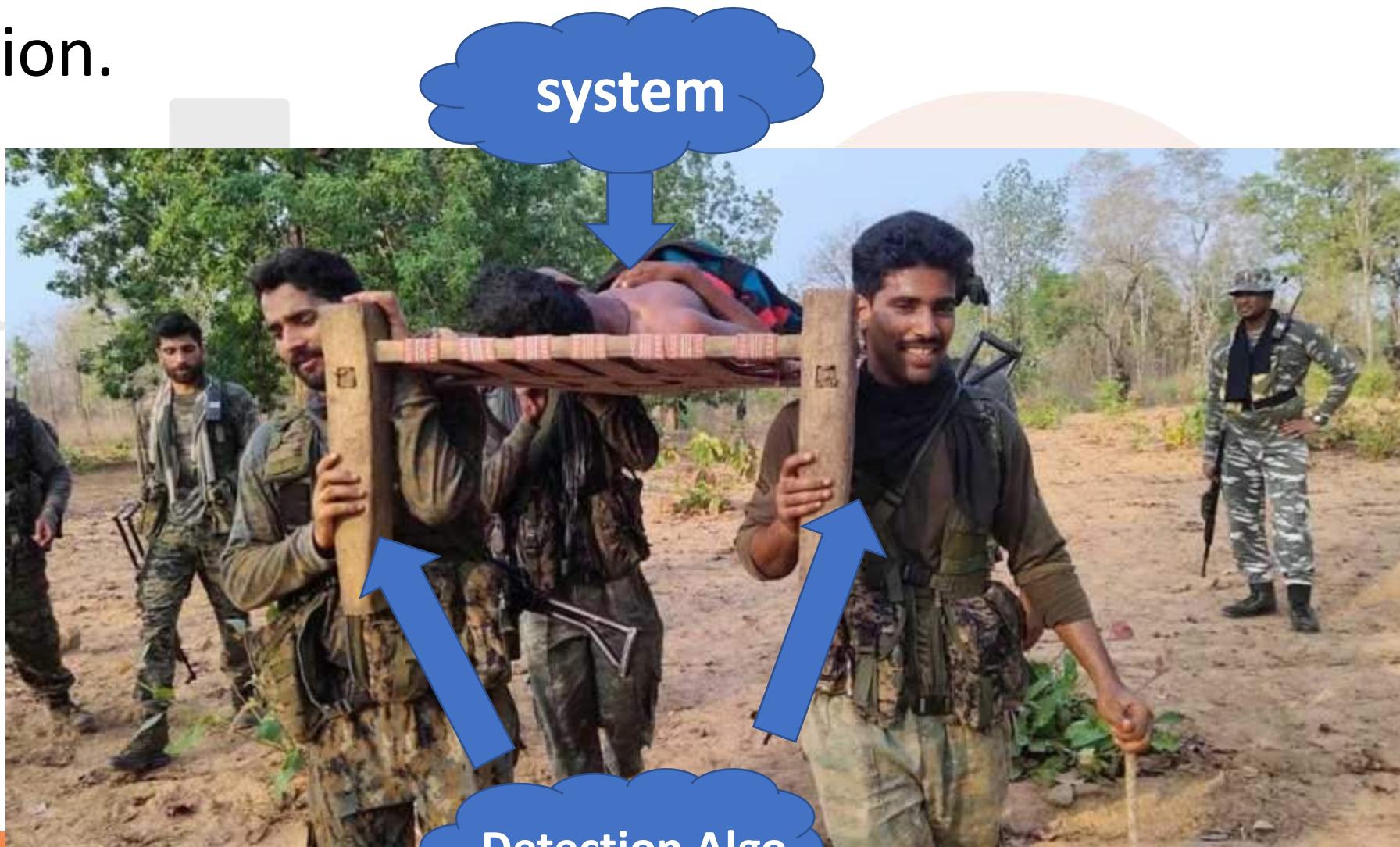
## Deadlock detection

- **Active approach**: Here we simply invoke the algorithm at defined intervals—for example, once per hour. or whenever CPU utilization drops below 40 percent.

Health Check-up



- **Lazy approach**: whenever CPU utilization drops below 40 percent or some unusual performance is there, we go for Detection.



To access all paid content

Amazon Prime at ₹25/day

CLICK HERE ↗



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Recovery from Deadlock

- When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually.
- Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock.

- **Process Termination**: One is simply to abort one or more processes to break the circular wait.
- **Pre-empt Resource**: The other is to pre-empt some resources from one or more of the deadlocked processes.

## Process Termination

- **Abort all deadlocked processes**: This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.

- **Abort one process at a time**: until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.



- Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state.
- Similarly, if the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## How to choose a victim

- If the partial termination method is used, then we must determine which deadlocked process (or processes) should be terminated. This determination is a policy decision, similar to CPU-scheduling decisions. The question is basically an economic one; we should abort those processes whose termination will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may affect which process is chosen, including:

- What is the priority of the process
- How long the process has computed and how much longer the process will compute before completing its designated task
- How many and what types of resources the process has used (for example, whether the resources are simple to pre-empt)
- How many more resources the process needs in order to complete
- How many processes will need to be terminated?
- Whether the process is interactive or batch

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Resource Pre-emption

- To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed:

- **Selecting a victim** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost.
- Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed.

- **Rollback**. If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource.
- We must roll back the process to some safe state and restart it from that state.
- Since, in general, it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it.
- Although it is more effective to roll back the process only as far as necessary to break the deadlock, this method requires the system to keep more information about the state of all running processes.

- **Starvation**. How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?
- In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim.
- As a result, this process never completes its designated task, a starvation situation any practical system must address.
- Clearly, we must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-pre-emptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the sources as follows if executed independently. **(GATE-2009) (2 Marks)**

Process P1:	Process P2:	Process P3:
t=0: requests 2 units of R2	t=0: requests 2 units of R3	t=0: requests 1 unit of R4
t=1: requests 1 unit of R3	t=2: requests 1 unit of R4	t=2: requests 2 units of R1
t=3: requests 2 units of R1	t=4: requests 1 unit of R1	t=5: releases 2 units of R1
t=5: releases 1 unit of R2 and 1 unit of R1.	t=6: releases 1 unit of R3	t=7: requests 1 unit of R2
t=7: releases 1 unit of R3	t=8: Finishes	t=8: requests 1 unit of R3
t=8: requests 2 units of R4		t=9: Finishes
t=10: Finishes		

	R1	R2	R3	R4
P1				
P2				
P3				
Available				
	R1	R2	R3	R4

Which one of the following statements is TRUE if all three processes run concurrently starting at time t=0?

- (A) All processes will finish without any deadlock
- (B) Only P1 and P2 will be in deadlock.
- (C) Only P1 and P3 will be in a deadlock.
- (D) All three processes will be in deadlock

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q Simplest way of deadlock recovery is (NET-DEC-2013)**

- a) Roll back**
- c) Lock one of the processes**

- b) Pre-empt resource**
- d) Kill one of the processes**



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Ignorance

- In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened.
- Operating System behaves like there is no concept of deadlock.

**Ostrich Algorithm**



- In this case, the undetected deadlock will cause the system's performance to deteriorate, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state.
- Eventually, the system will stop functioning and will need to be restarted manually.

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Although this method may not seem to be a viable approach to the deadlock problem, it is nevertheless used in most operating systems.
- Expense is one important consideration. Ignoring the possibility of deadlocks is cheaper than the other approaches. Since in many systems, deadlocks occur infrequently (say, once per year), the extra expense of the other methods may not seem worthwhile.

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- In addition, methods used to recover from other conditions may be put to use to recover from deadlock. In some circumstances, a system is in a frozen state but not in a deadlocked state.
- We see this situation, for example, with a real-time process running at the highest priority (or any process running on a nonredemptive scheduler) and never returning control to the operating system.
- The system must have manual recovery methods for such conditions and may simply use those techniques for deadlock recovery.

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider the solution to the bounded buffer producer/consumer problem by using general semaphores S,F and E. The semaphore S is the mutual exclusion semaphore initialized to 1. The semaphore F corresponds to the number of free slots in the buffer and is initialized to N. The semaphore E corresponds to the number of elements in the buffer and is initialized to 0. **(GATE-2006) (2 Mark)**

Which of the following interchange operations may result in a deadlock?

- I) Interchanging Wait (F) and Wait (S) in the Producer process
  - II) Interchanging Signal (S) and Signal (F) in the Consumer process
- a) (I) only
  - b) (II) only
  - c) Neither (I) nor (II)
  - d) Both (I) and (II)

Producer()	Consumer()
{	{
while(T)	while(T)
{	{
Produce()	wait(E)//UnderFlow
wait(F)//OverFlow	wait(S)
Wait(S)	pick()
append()	signal(S)
signal(S)	signal(F)
signal(E)	consume()
}	}
}	}

**Q** In a certain operating system, deadlock prevention is attempted using the following scheme. Each process is assigned a unique timestamp, and is restarted with the same timestamp if killed. Let  $P_h$  be the process holding a resource  $R$ ,  $P_r$  be a process requesting for the same resource  $R$ , and  $T(P_h)$  and  $T(P_r)$  be their timestamps respectively. The decision to wait or preempt one of the processes is based on the following algorithm.

if  $T(P_r) < T(P_h)$

    then kill  $P_r$

else wait

Which one of the following is TRUE? **(GATE-2004) (2 Marks)**

- (A) The scheme is deadlock-free, but not starvation-free
- (B) The scheme is not deadlock-free, but starvation-free
- (C) The scheme is neither deadlock-free nor starvation-free
- (D) The scheme is both deadlock-free and starvation-free

Q The following is a code with two threads, producer and consumer, that can run in parallel. Further, S and Q are binary semaphores equipped with the standard P and V operations.

semaphore S = 1, Q = 0;	
integer x;	
producer:	consumer:
while (true) do	while (true) do
P(S);	P(Q);
x = produce ();	consume (x);
V(Q);	V(S);
done	done

Which of the following is TRUE about the program above? (GATE-2008) (2 Marks)

- (A) The process can deadlock
- (B) One of the threads can starve
- (C) Some of the items produced by the producer may be lost
- (D) Values generated and stored in 'x' by the producer will always be consumed before the producer can generate a new value

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider a system with seven processes A through G and six resources R through W. Resource ownership is as follows : process A holds R and wants T process B holds nothing but wants T process C holds nothing but wants S process D holds U and wants S & T process E holds T and wants V process F holds W and wants S process G holds V and wants U Is the system deadlocked ? If yes, \_\_\_\_\_ processes are deadlocked. (NET-AUG-2016)

- a)** No
- b)** Yes, A, B, C
- c)** Yes, D, E, G
- d)** Yes, A, B, F

Q Consider a system with five processes P0 through P4 and three resource types R1 R and R3. Resource type R1 has 10 instances, R2 has 5 instances and R3 has 7 instances. Suppose that at time T0, the following snapshot of the system has been taken: (NET-JUNE-2014)

Assume that now the process P1 requests one additional instance of type R1 and two instances of resource type R3. The state resulting after this allocation will be

(A) Ready state

(B) Safe state

(C) Blocked state

(D) Unsafe state

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	3	9	0	2			
P3	2	1	1	2	2	2			
P4	0	2	2	4	3	3			



# Extra

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** A system has four processes and five resources. The current allocation and maximum needs are as follows: **(NET-DEC-2015)**

The smallest value of  $x$  for which the above system in safe state is \_\_\_\_\_.

- a) 1
  - b) 3
  - c) 2
  - d) Not safe for any value of  $x$

	Allocation					Request					Available				
	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
P1	1	0	2	1	1	1	1	2	1	3	0	0	X	1	1
P2	2	0	1	1	0	2	2	2	1	0					
P3	1	1	0	1	0	2	1	3	1	0					
P4	1	1	1	1	0	1	1	2	2	1					

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** 

**Q** Consider a system which have 'n' number of processes and 'm' number of resource types. The time complexity of the safety algorithm, which checks whether a system is in safe state or not, is of the order of: **(NET-AUG-2016)**

- a)  $O(mn)$
- b)  $O(m^2n^2)$
- c)  $O(m^2n)$
- d)  $O(mn^2)$

# Deadlock detection and recovery

- When should we invoke the detection algorithm? It depends on two factors:
  - How often is a deadlock likely to occur?
  - How many processes will be affected by deadlock when it happens?

- **Avoidance:** If deadlocks occur frequently, then the detection algorithm should be invoked frequently. Of course, invoking the deadlock-detection algorithm for every resource request will incur considerable overhead in computation time.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q.3. Consider a computer system with multiple shared resource types, with one instance per resource type. Each instance can be owned by only one process at a time. Owning and freeing of resources are done by holding a global lock (L). The following scheme is used to own a resource instance:

```
function OwnResource (Resource R)
    Acquire lock L // a global lock
    if R is available then
        Acquire R
        Release lock L
    else
        if R is owned by another process P then
            Terminate P, after releasing all resources owned by P
            Acquire R
            Restart P
            Release lock L
        end if
    end if
end function
```

Which of the following choice(s) about the above scheme is/are correct?

- (a) The scheme violates the mutual exclusion property.
- (b) The scheme ensures that deadlocks will not occur.
- (c) The scheme may lead to live-lock.
- (d) The scheme may lead to starvation.

Ans. (b, c, d)

day **CLICK HERE** ↗

- A **Livelock** is a situation where a request for an exclusive lock is denied repeatedly, as many overlapping shared locks keep on interfering each other.
- The processes keep on changing their status, which further prevents them from completing the task. This further prevents them from completing the task.

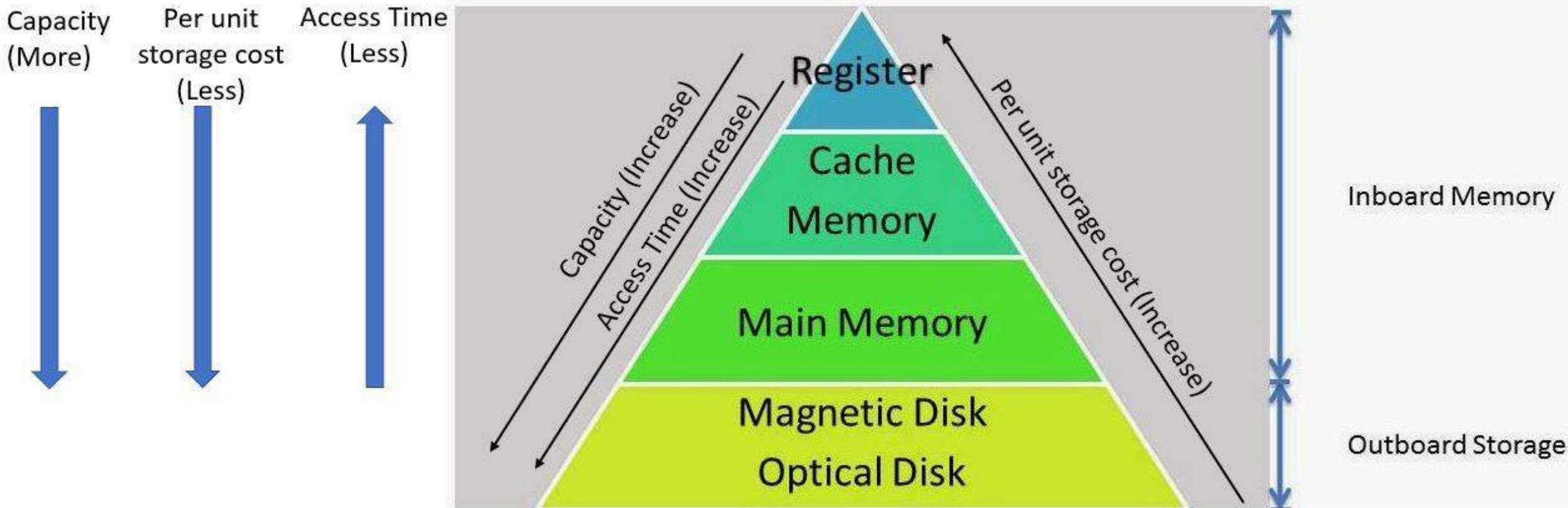
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Memory Hierarchy

- Let first understand what we need from a memory
  - Large capacity
  - Less per unit cost
  - Less access time(fast access)

# Memory Hierarchy

- The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



**Cycle**



**Car**



**Airbus**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



**Lathi**

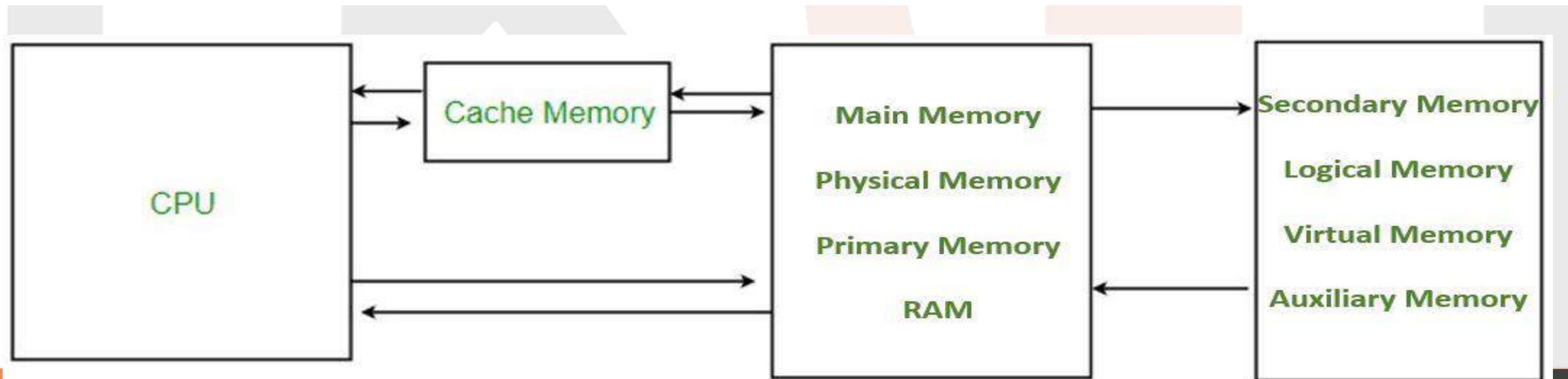
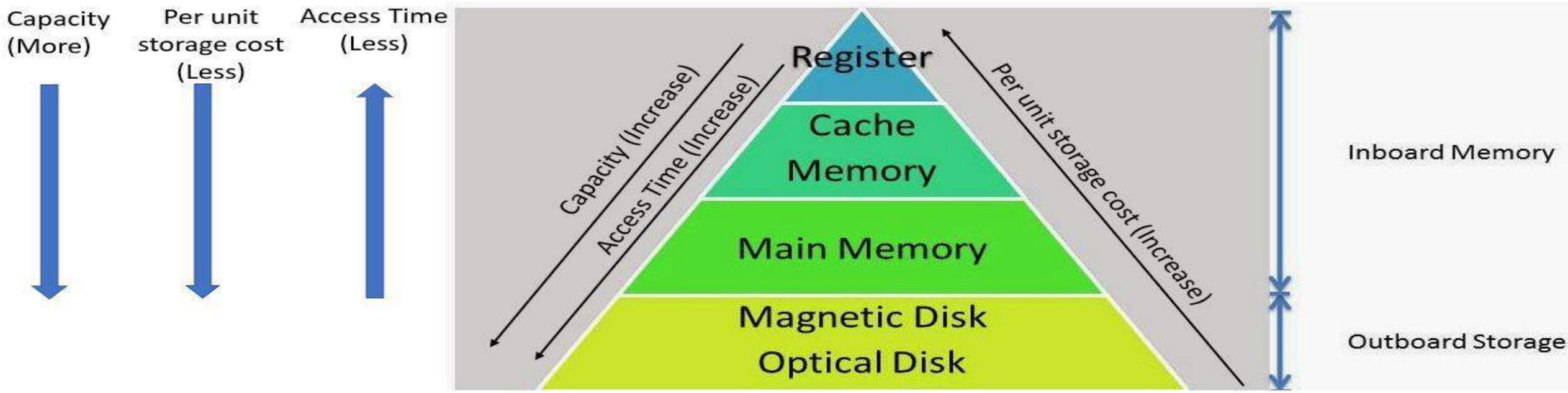


**303**



**AK-47**

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



**Showroom**



**Go down**

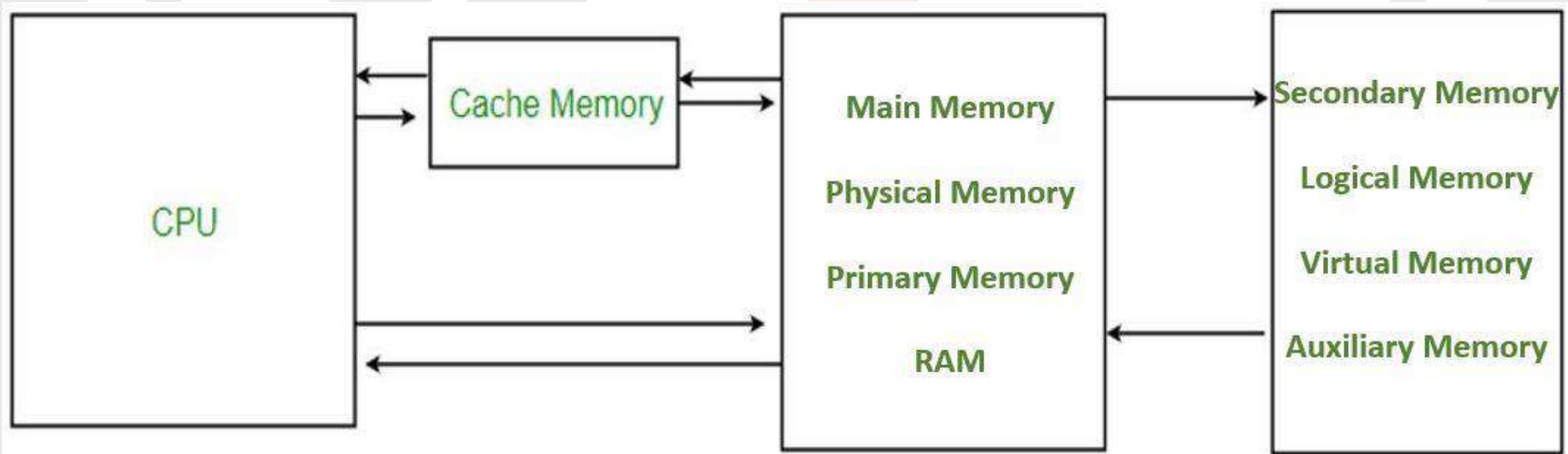


**Factory**

To access all paid content get **KG Prime** at ₹25/day

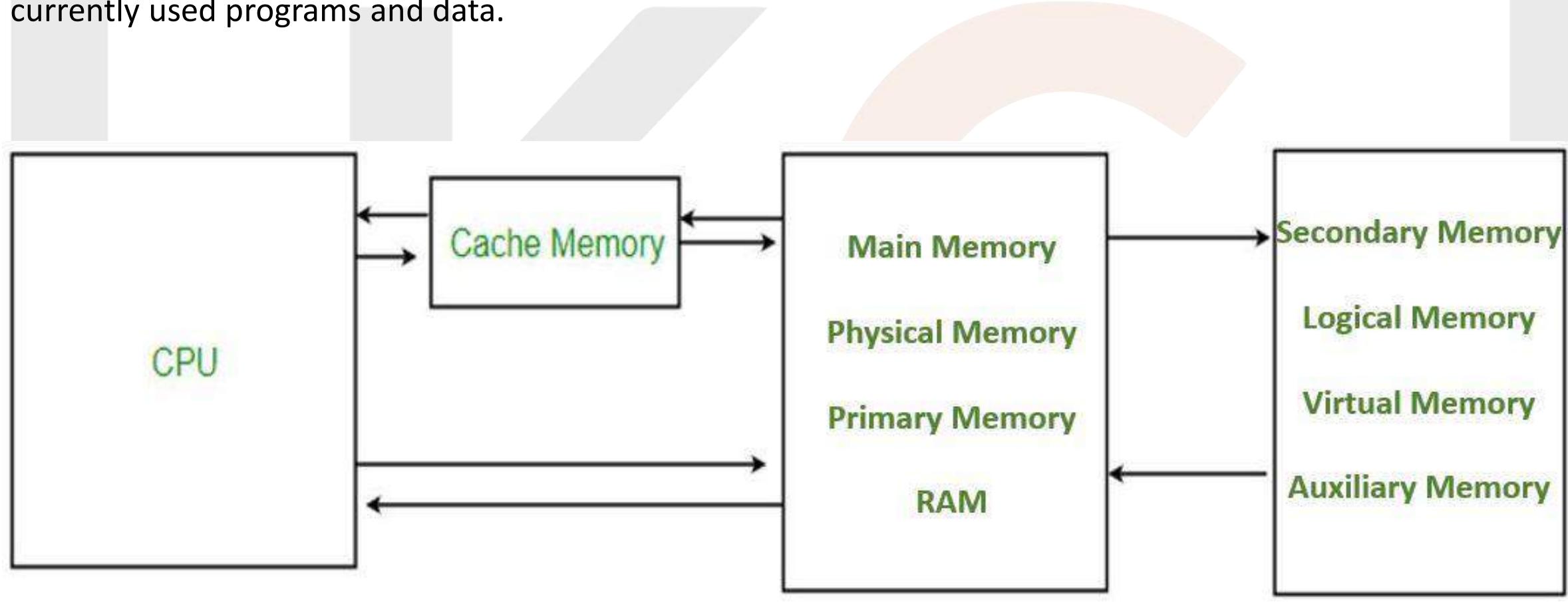
**CLICK HERE ↗**

- If we want this hierarchy to work fine, then most of the time when CPU needs a data it must be present in Cache, if not possible main memory, worst case Secondary memory.
- But this is a difficult task to do as my computer has, 8 TB of Secondary Memory, 32 GB of Main Memory, but only 768KB, 4MB, 16 MB of  $L_1$ ,  $L_2$ ,  $L_3$  cache respectively.
- If somehow we can estimate what data CPU will require if future we can prefetch in Cache and Main Memory.
- Locality of reference Helps we to perform this estimation.



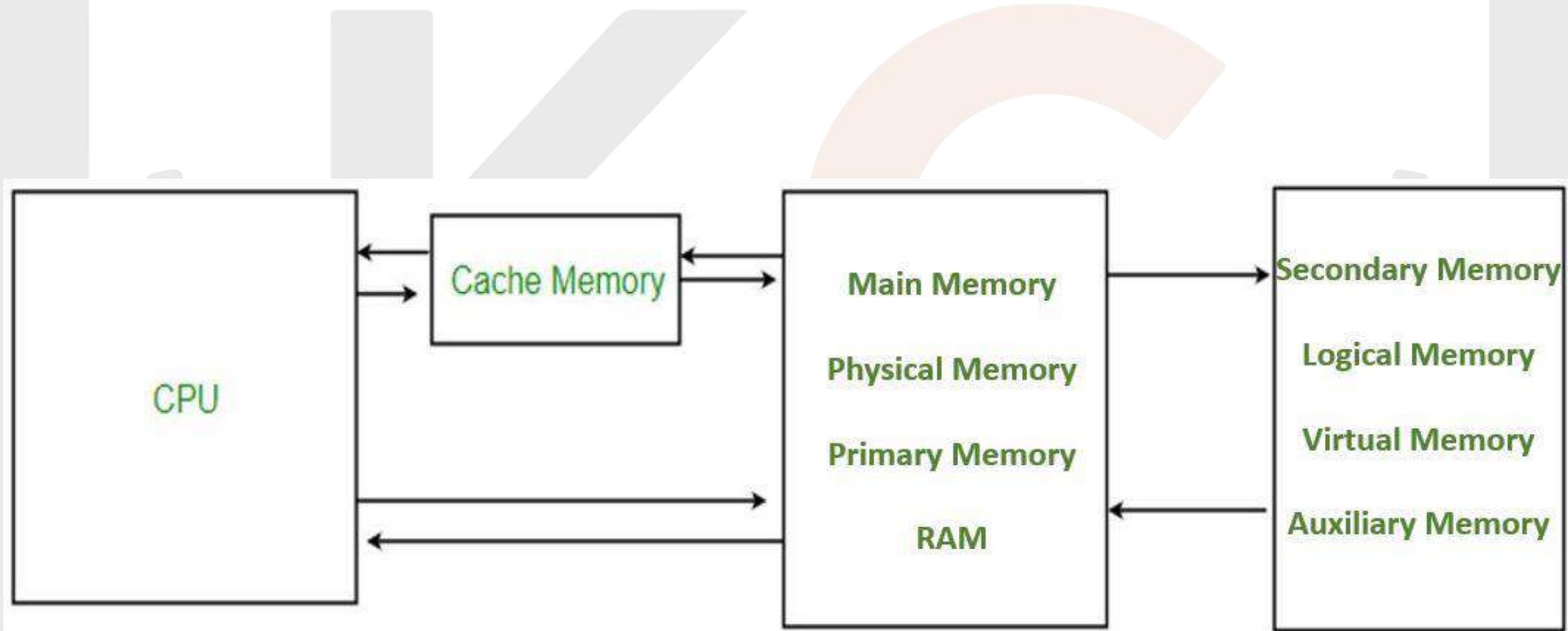
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- The main memory occupies a central position by being able to communicate directly with the CPU.
- When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory.
- Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

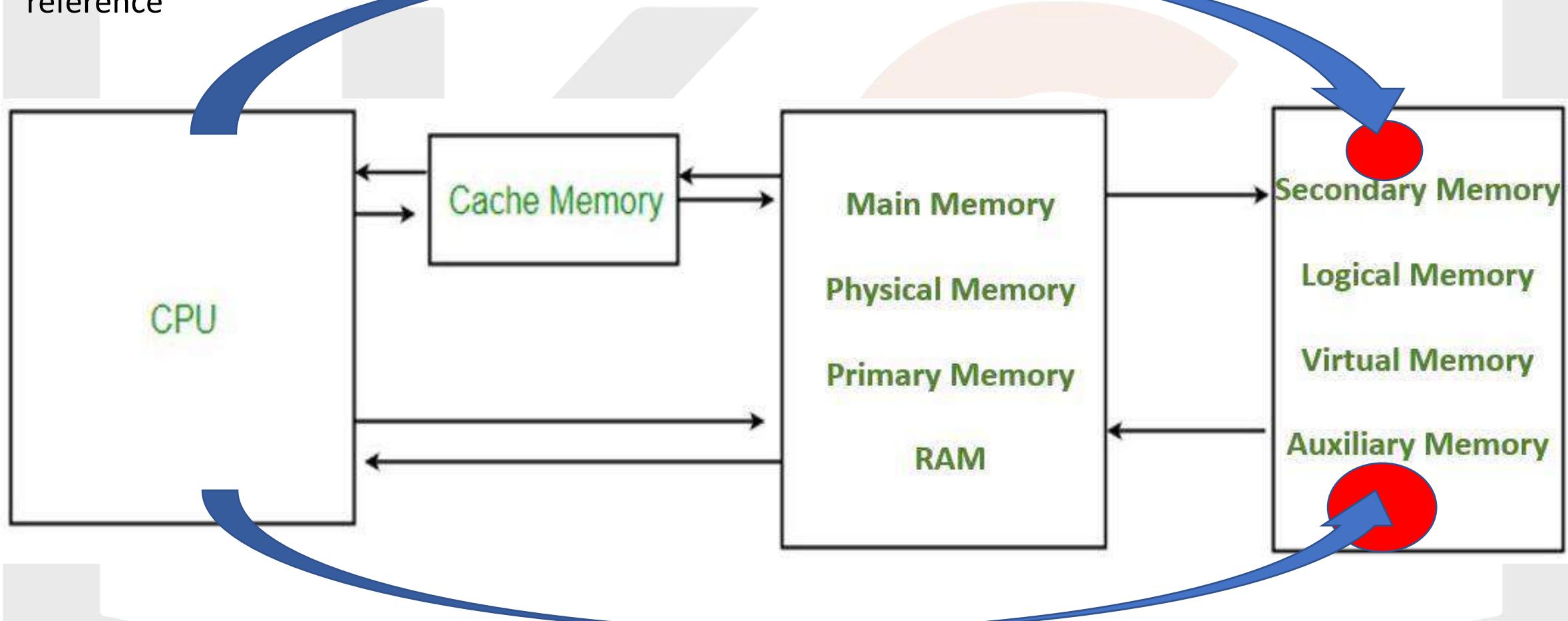
- A special very-high speed memory called a Cache is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

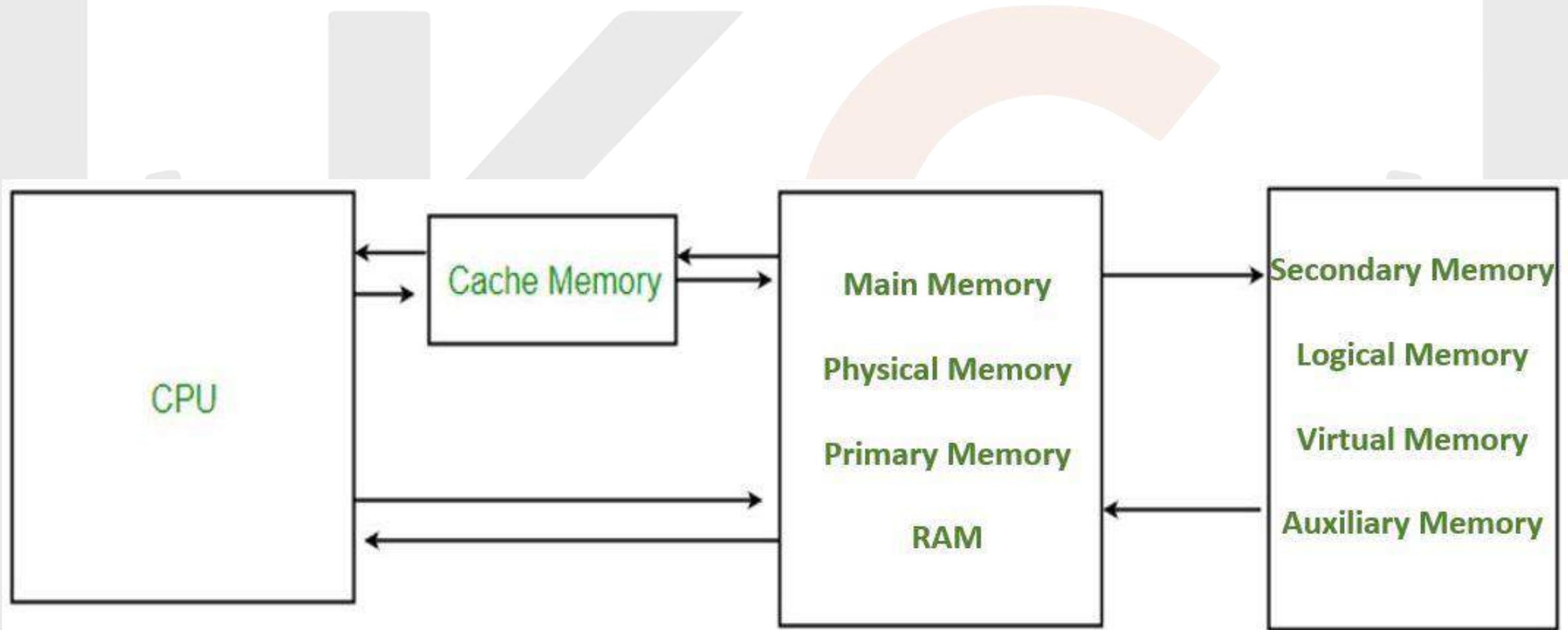
# Locality of Reference

- The references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. There are two types of locality of reference



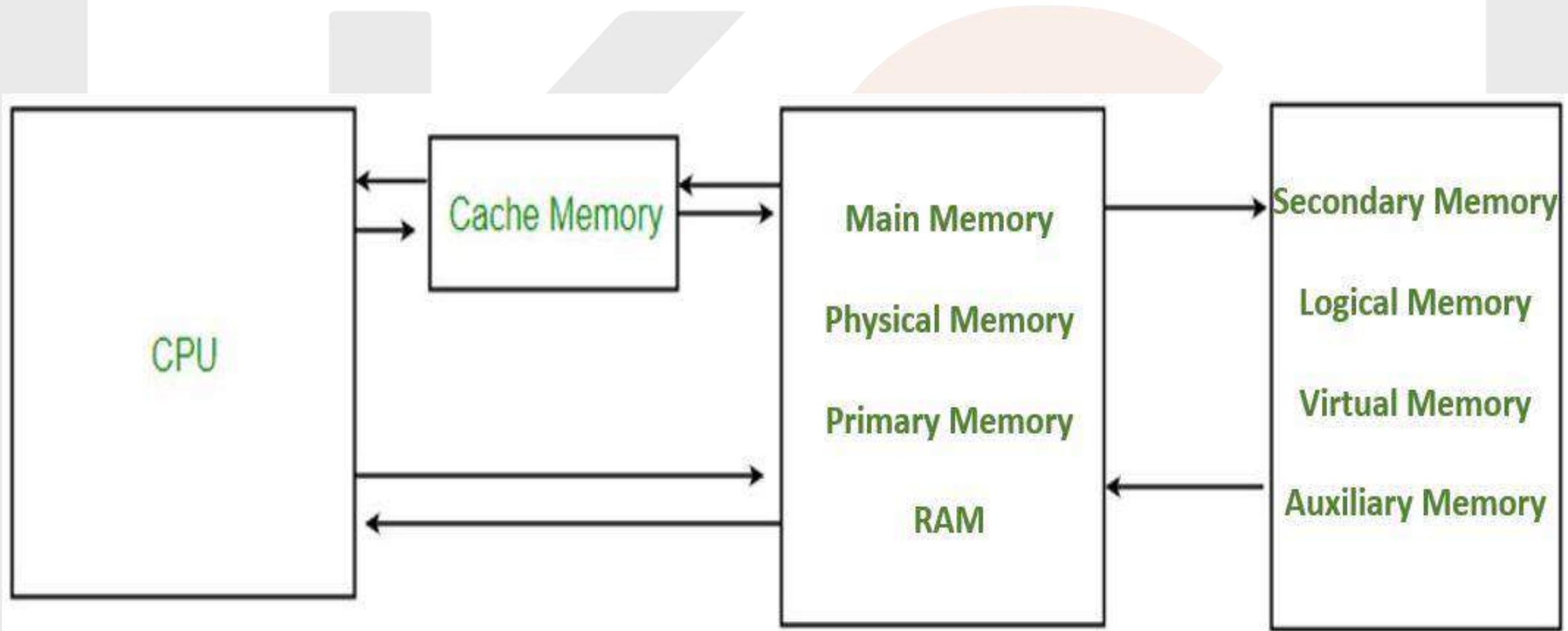
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Spatial Locality:** Spatial locality refers to the use of data elements in the nearby locations.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **Temporal Locality:** Temporal locality refers to the reuse of specific data, and/or resources, within a relatively small-time duration. Or, the most frequently used items will be needed soon. (LRU is used for temporal locality)



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



**Showroom**



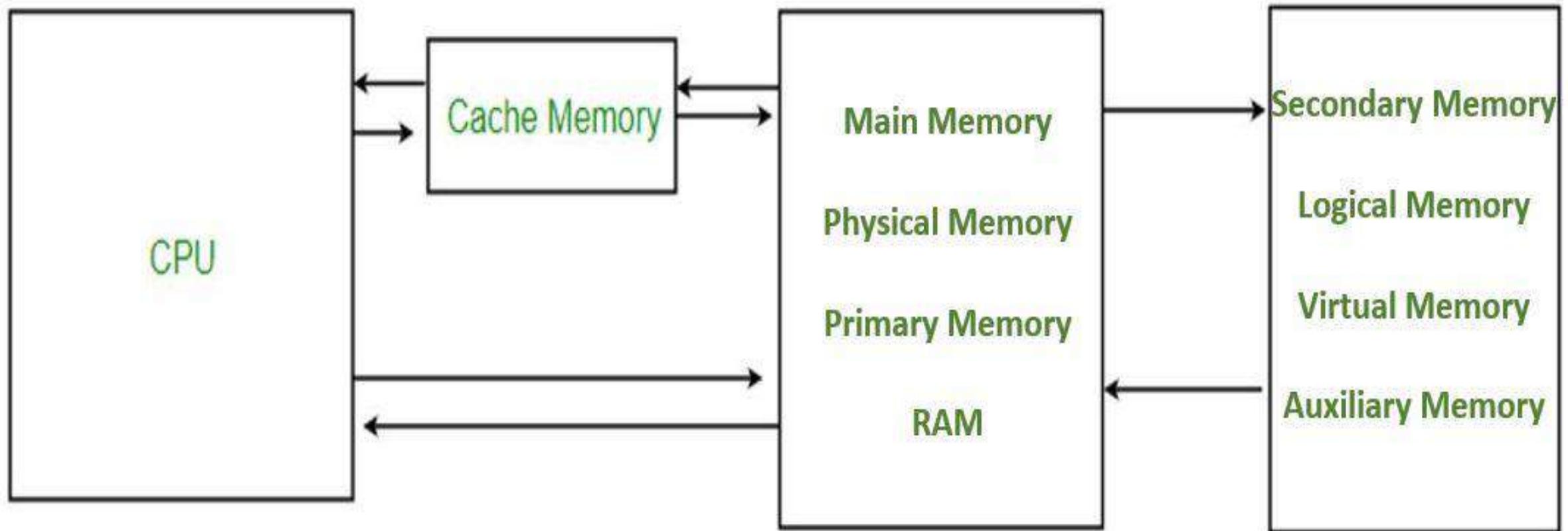
**Go down**



**Factory**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- Reason to have memory hierarchy instead of a single memory is
  - Average access time (Should be less)
  - Capacity (Should be high)
  - Cost Per unit (Should be less)

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Duty of Operating System

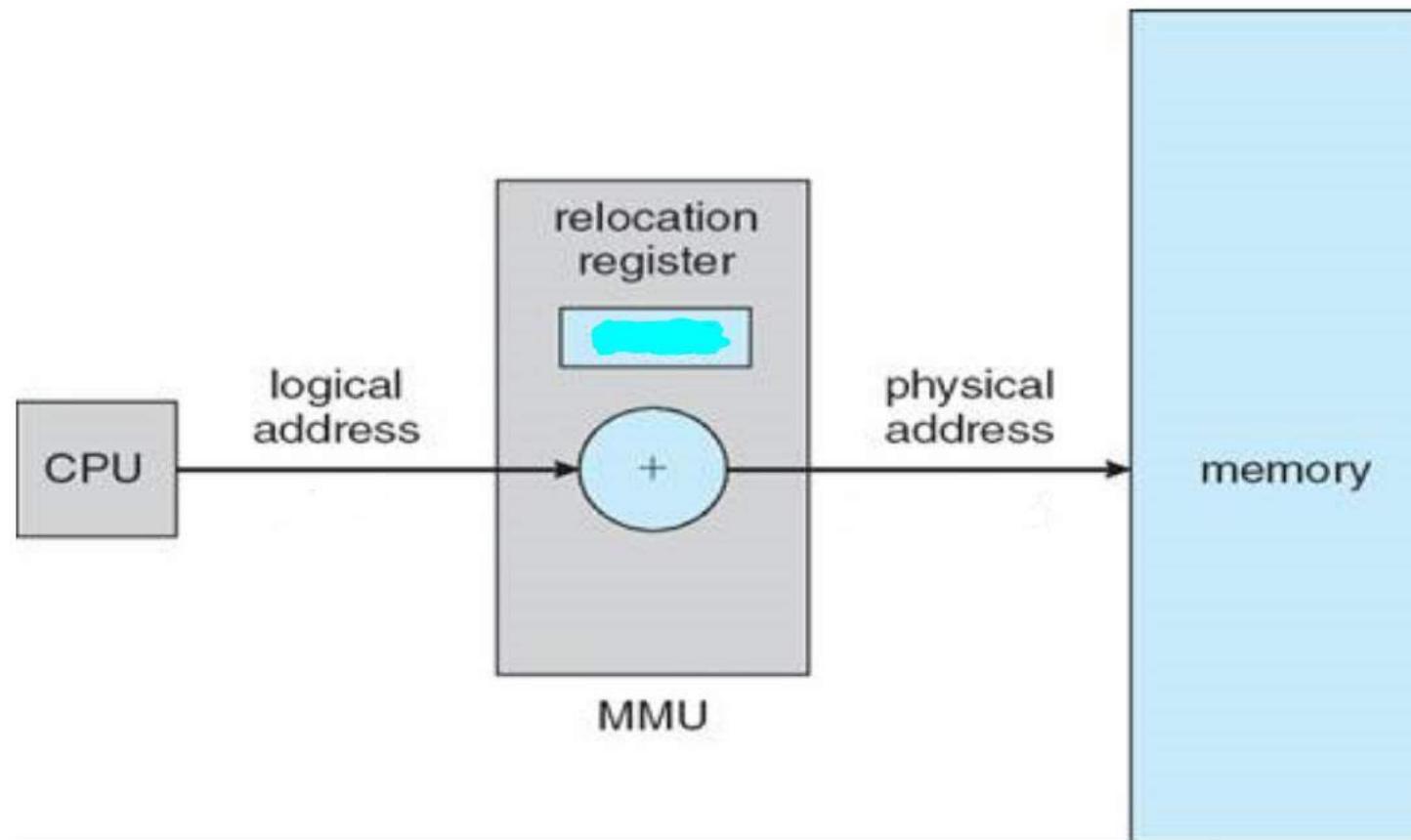
- Operating system is responsible for the following activities in connection with memory management:
  - Address translation from Logical address to Physical address.
  - Deciding which processes (or parts of processes) and data to move into and out of memory. Allocating and deallocating memory space as needed.
  - Keeping track of which parts of memory are currently being used and who is using them.

- The first duty of OS is to translate this logical address into physical address. There can be two approaches for storing a process in main memory.
  - Contiguous allocation policy
  - Non-contiguous allocation policy

## Contiguous allocation policy

- We know that when a process is required to be executed it must be loaded to main memory, by policy has two implications.
  - It must be loaded to main memory completely for execution.
  - Must be stored in main memory in contiguous fashion.

- Here we use a memory management unit(OS) whose duty is to take logical address and translate it into physical address.
- MMU contains a relocation register, which contains the base address of the process in the main memory and it is added in the logical address every time

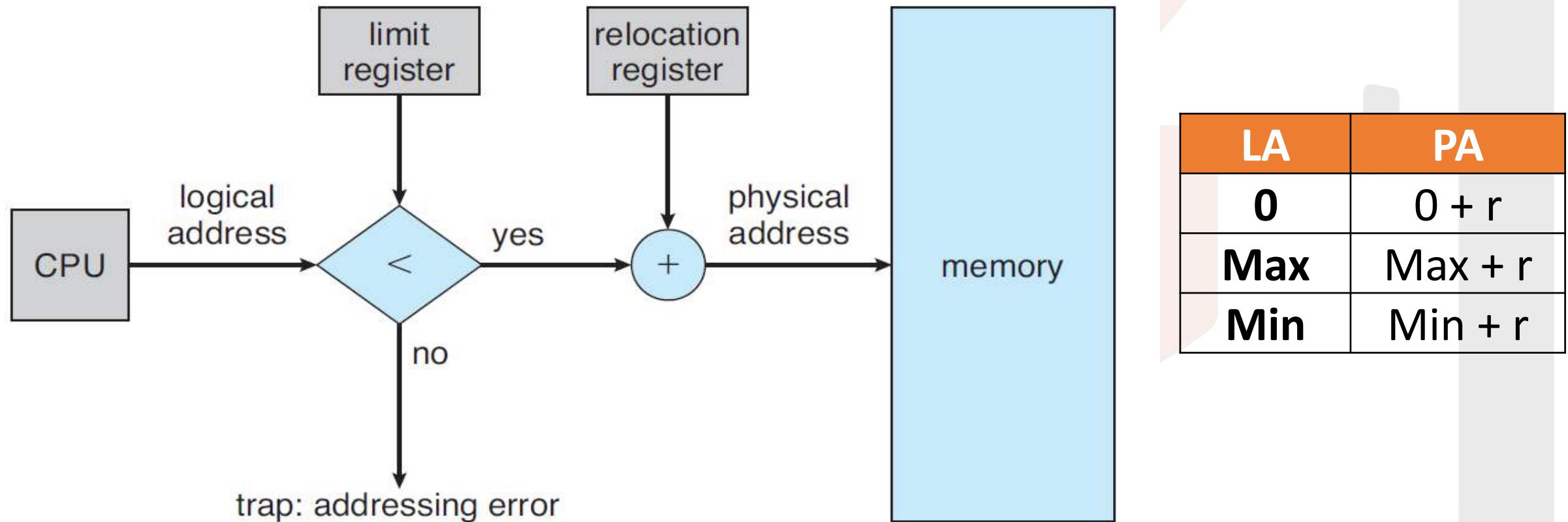


To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)

- **Advantage**: - easy strategy, simple to use, simple to understand.
- **Disadvantage**: - if a process generates logical address greater than its maximum limit, then it can easily access the data of some other process.

- In order to check whether address generated to CPU is valid(with in range) or invalid, we compare it with the value of limit register, which contains the max no of instructions in the process.
- So, if the value of logical address is less than limit, then it means it's a valid request and we can continue with translation otherwise, it is a illegal request which is immediately trapped by OS.



**Q** Consider the following tables of relocation and limit registers and find the illegal attempts and if valid find the physical address?

	Limit Register	Relocation Register
P <sub>0</sub>	500	1200
P <sub>1</sub>	275	550
P <sub>2</sub>	212	880
P <sub>3</sub>	420	1400
P <sub>4</sub>	118	200

Following are the request by process

P<sub>0</sub>—450

P<sub>1</sub>—300

P<sub>2</sub>—210

P<sub>3</sub>—450

P<sub>4</sub>---80

To access all paid content get **KG Prime** at ₹25/day

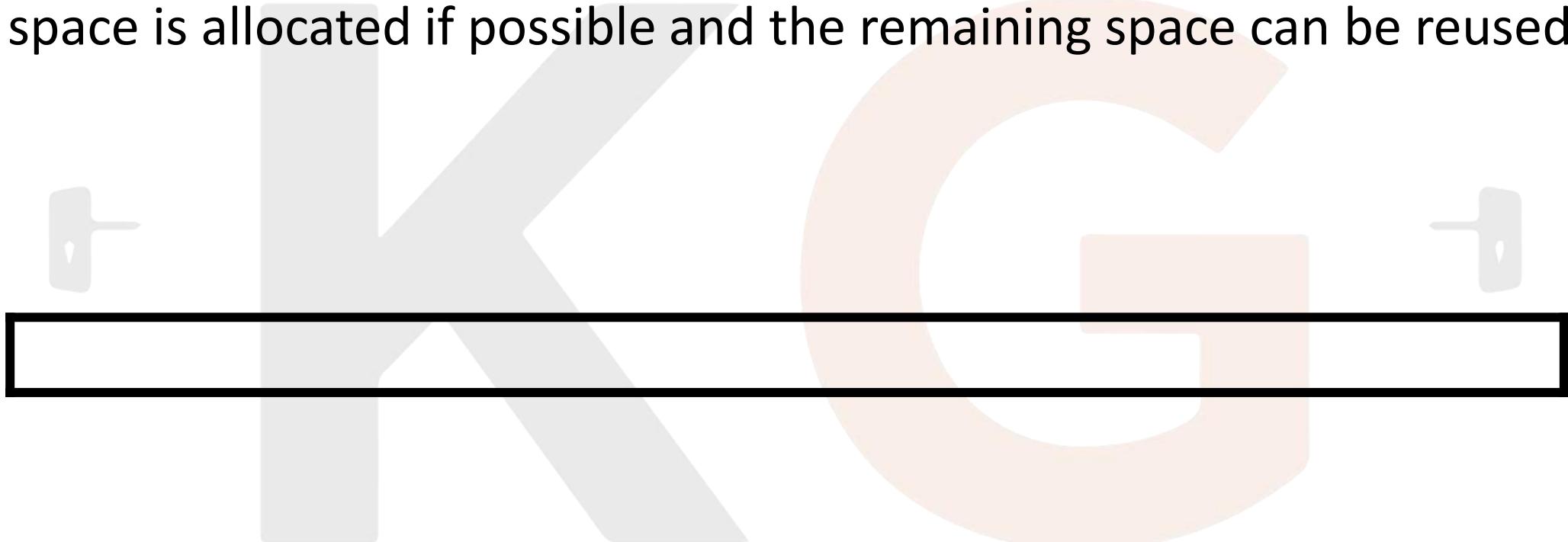
**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

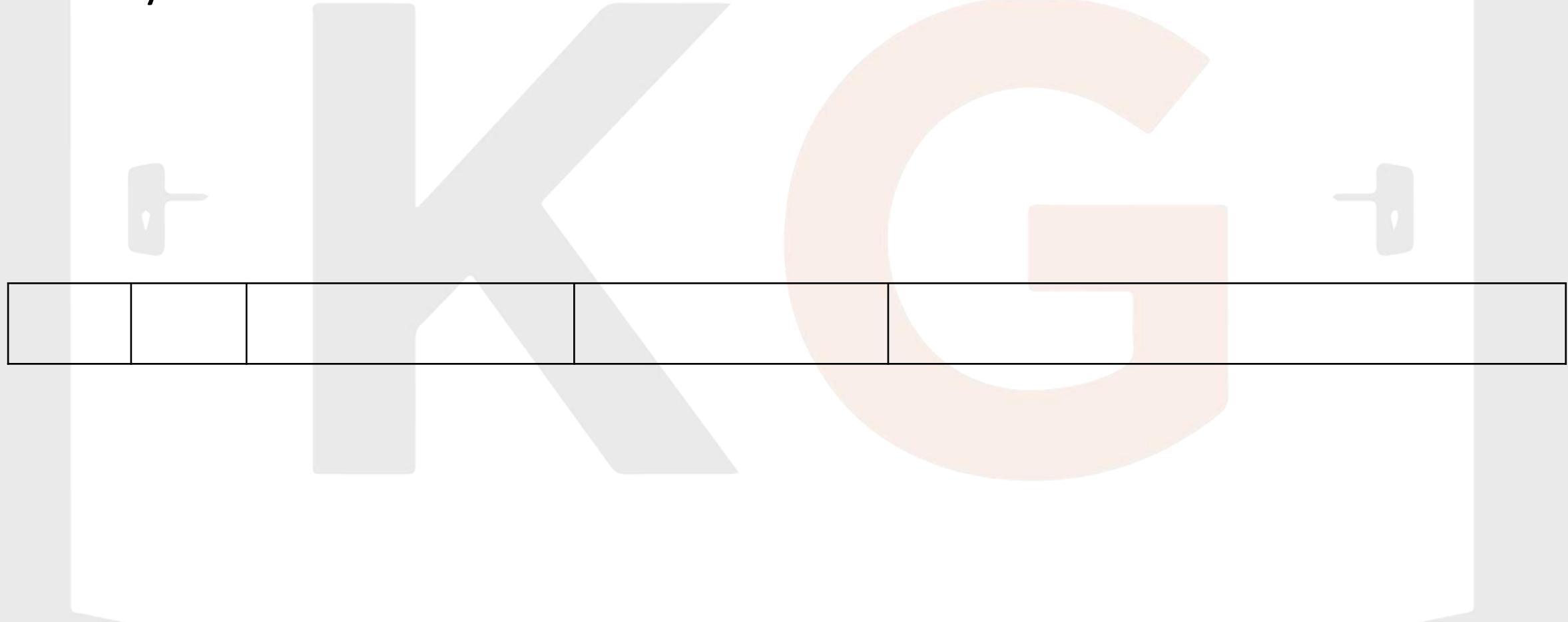
## Space Allocation Method in Contiguous Allocation

- **Variable size partitioning:** -In this policy, in starting, we treat the memory as a whole or a single chunk & whenever a process request for some space, exactly same space is allocated if possible and the remaining space can be reused again.

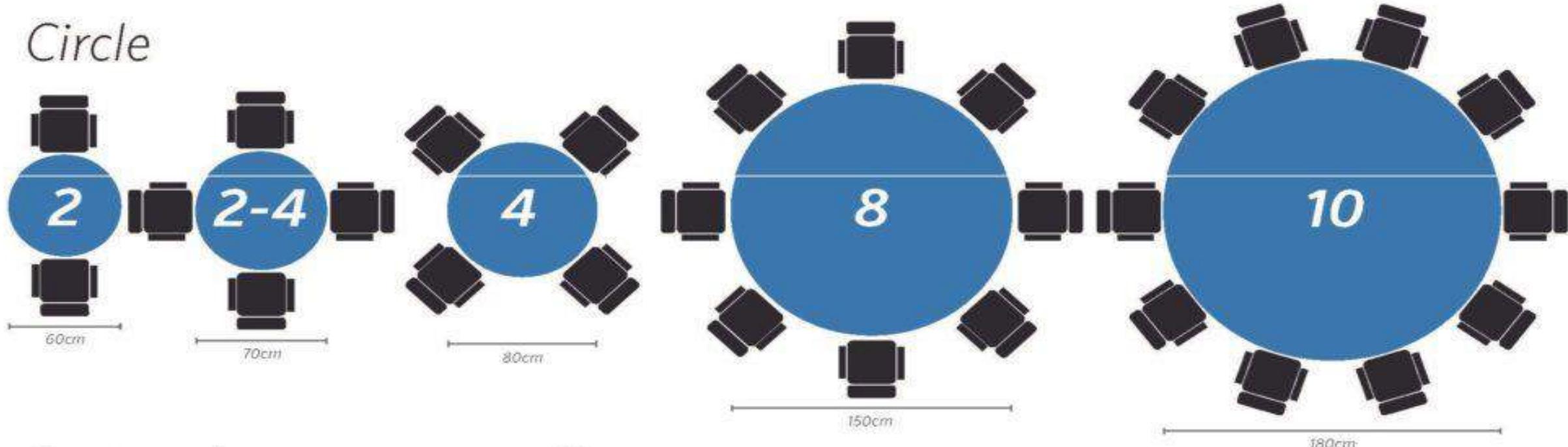




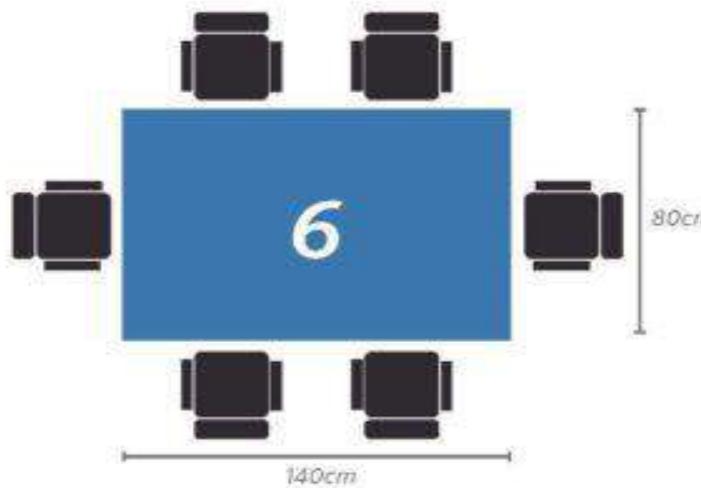
- **Fixed size partitioning**:- here, we divide memory into fixed size partitions, which may be of different sizes, but here if a process request for some space, then a partition is allocated entirely if possible, and the remaining space will be wasted internally.



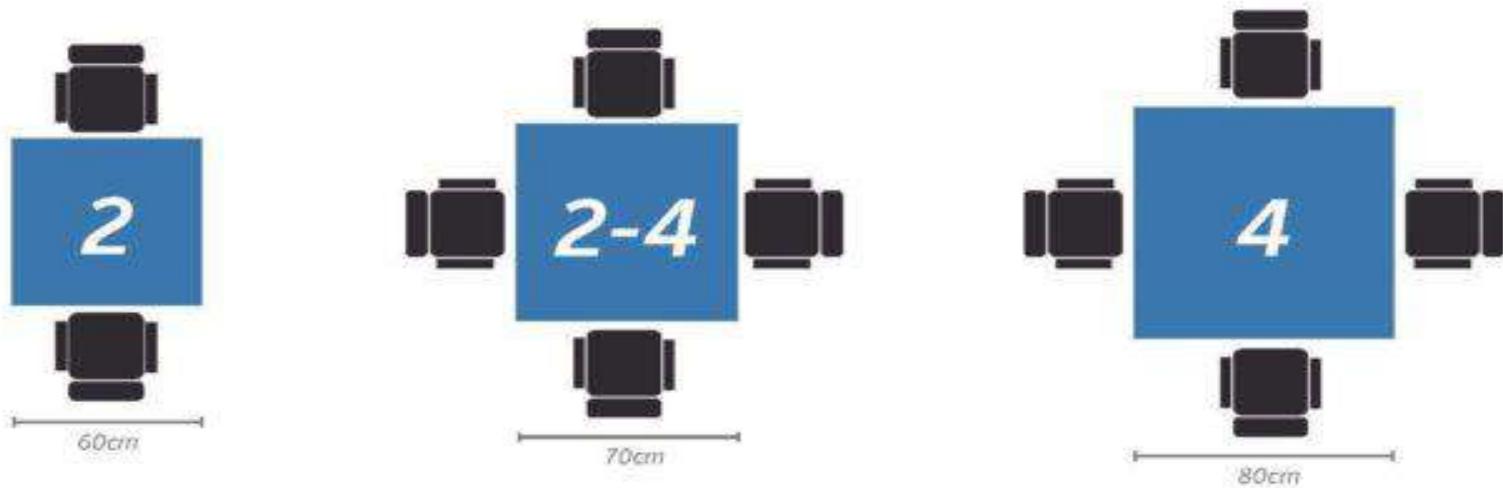
## Circle



## Rectangle



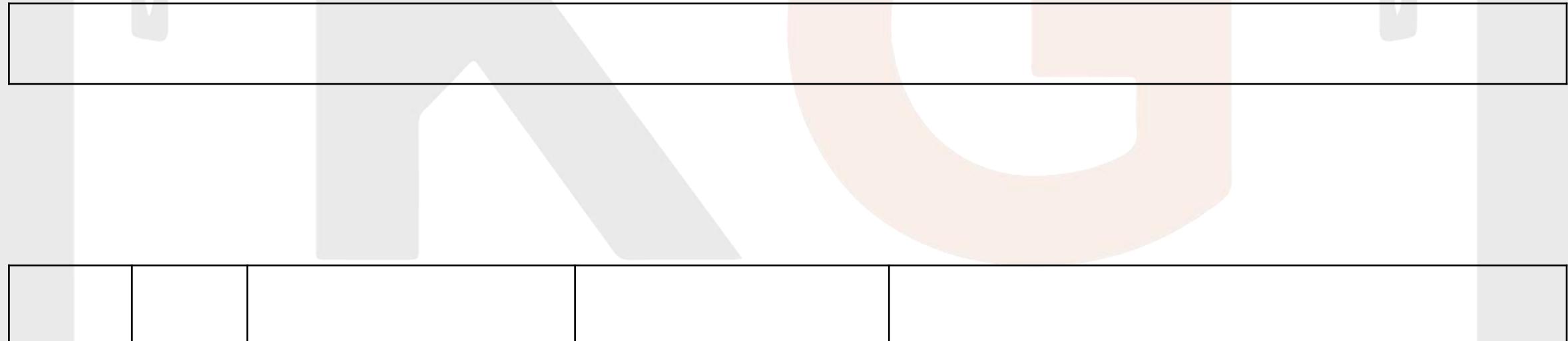
## Square



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **First fit policy:** - as the name implies, it states searching the memory from the base and will allocate first partition which is capable enough.

- **Advantage:** - simple, easy to use, easy to understand
- **Disadvantage:** -poor performance, both in terms of time and space



- **Best fit policy**: - Here, we search the entire memory and will allocate the smallest partition which is capable enough.
  - **Advantage**: - perform best in fix size partitioning scheme.
  - **Disadvantage**: - difficult to implement, perform worst in variable size partitioning as the remaining spaces which are of very small size.



- **Worst fit policy**: - it also searches the entire memory and allocate the largest partition possible.
  - **Advantage**: - perform best in variable size partitioning
  - **Disadvantage**: - perform worst in fix size partitioning, resulting into large internal fragmentation.



- **Next fit policy:** - next fit is the modification in the best fit where, after satisfying a request, we start satisfying next request from the current position.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

- **External fragmentation:** - External fragmentation is a function of contiguous allocation policy. The space requested by the process is available in memory but, as it is not being contiguous, cannot be allocated this wastage is called external fragmentation.





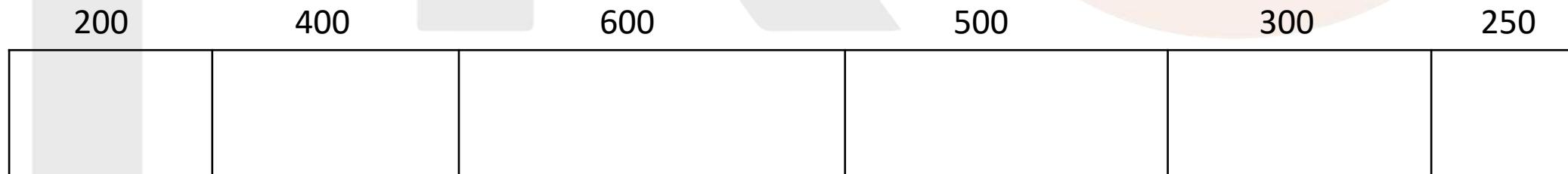
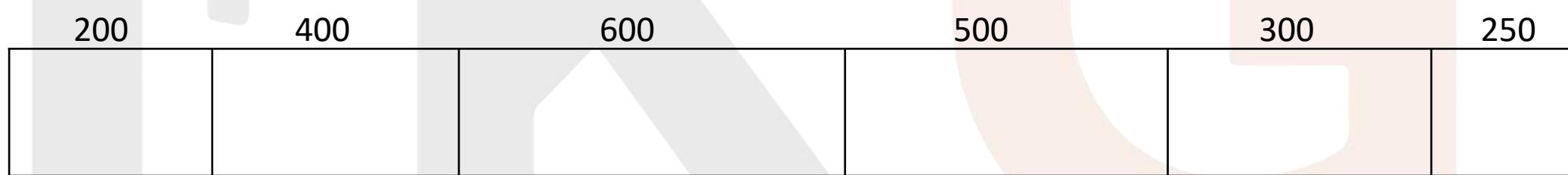
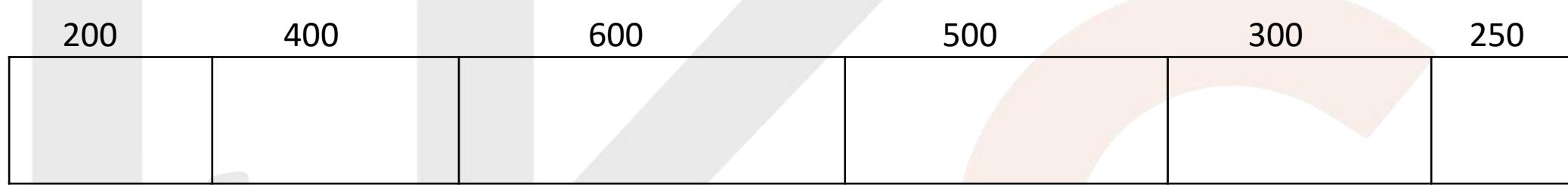
**The Big Cason Family Are Expecting Their 17th Child**

- **Internal fragmentation:** - Internal fragmentation is a function of fixed size partition which means, when a partition is allocated to a process. Which is either the same size or larger than the request then, the unused space by the process in the partition Is called as internal fragmentation



**Q** Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process? (GATE-2015) (2 Marks)

- (A) 200 KB and 300 KB      (B) 200 KB and 250 KB      (C) 250 KB and 300 KB      (D) 300 KB and 400 KB



To access all paid content get **KG Prime** at ₹25/day

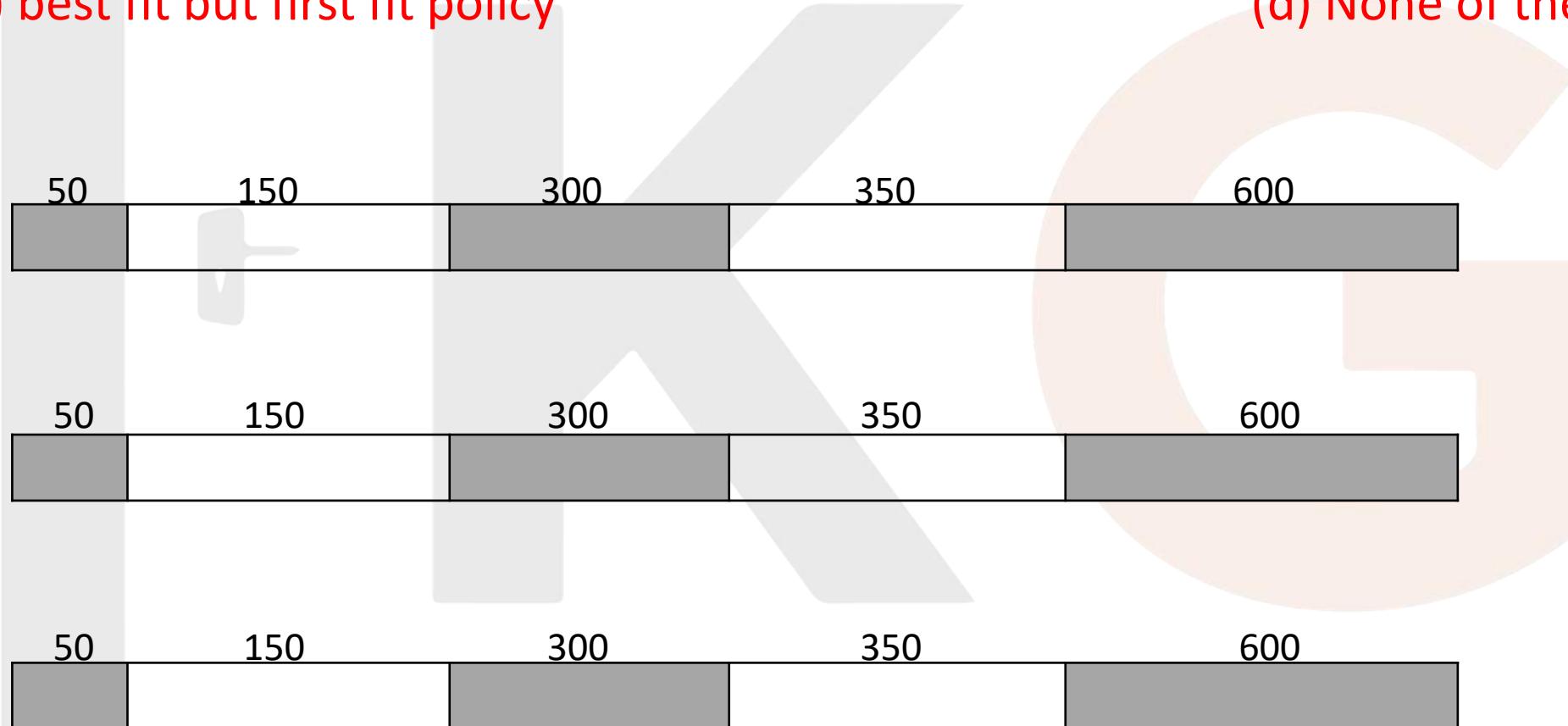
**CLICK HERE ↗**

Q Consider the following heap (figure) in which blank regions are not in use and hatched region are in use. (GATE-1994) (2 Marks)

The sequence of requests for blocks of size 300, 25, 125, 50 can be satisfied if we use

- (a) either first fit or best fit policy (any one)
- (b) first fit but not best fit policy
- (c) best fit but first fit policy

- (d) None of the above



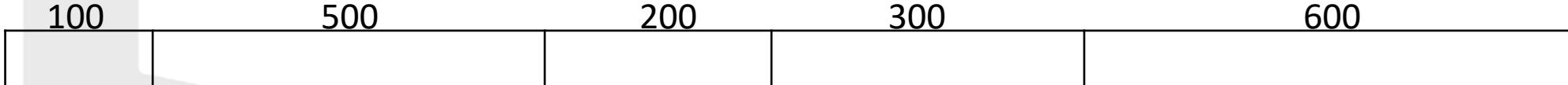
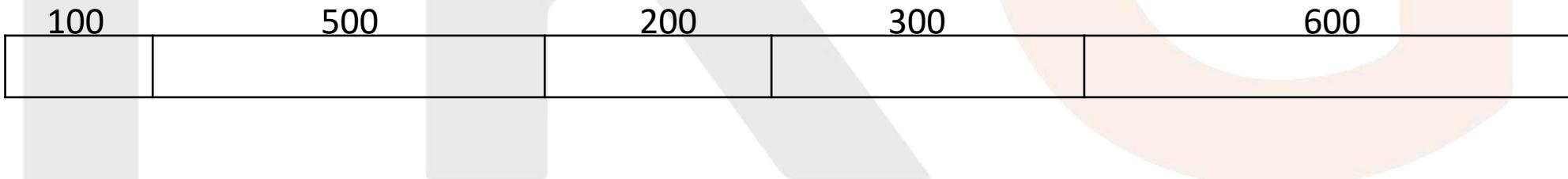
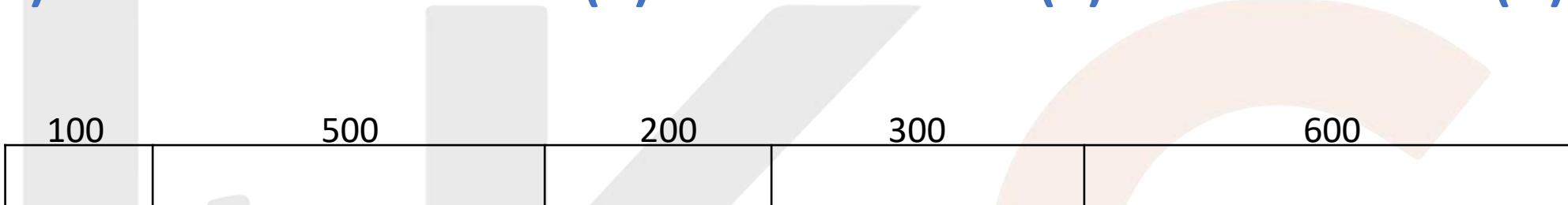
Q Given memory partitions of 100 K, 500 K, 200 K, 300 K and 600 K (in order) and processes of 212 K, 417 K, 112 K, and 426 K (in order), using the first-fit algorithm, in which partition would the process requiring 426 K be placed ? **(NET-DEC-2012)**

(A) 500 K

(B) 200 K

(C) 300 K

(D) 600 K



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A program is located in the smallest available hole in the memory is .....  
**(NET-DEC-2009)**

**(A)** best – fit

**(B)** first – fit

**(C)** worst – fit

**(D)** buddy



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A 1000 Kbyte memory is managed using variable partitions but No compaction. It currently has two partitions of sizes 200 Kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for? **(GATE-1996) (1 Marks)**

(a) 151

(b) 181

(c) 231

(d) 541

- How can we solve external fragmentation?

- We can also swap processes in the main memory after fixed intervals of time & they can be swapped in one part of the memory and the other part become empty(Compaction, defragmentation). This solution is very costly in respect to time as it will take a lot of time to swap process when system is in running state.
- Either we should go for non-contiguous allocation, which means process can be divided into parts and different parts can be allocated in different areas.

**Q** Variable partition memory management technique with compaction results in:  
**(NET-JUNE-2009)**

**(A)** Reduction of fragmentation

**(B)** Minimal wastage

**(C)** Segment sharing

**(D)** None of the above



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q Which of the following statements are true ? (NET-JULY-2018)**

- (a)** External Fragmentation exists when there is enough total memory space to satisfy a request but the available space is contiguous.
- (b)** Memory Fragmentation can be internal as well as external.
- (c)** One solution to external Fragmentation is compaction.

Code :

**(a)** (a) and (b) only

**(b)** (a) and (c) only

**(c)** (b) and (c) only

**(d)** (a), (b) and (c)

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q In which of the following storage replacement strategies, is a program placed in the largest available hole in the memory? (NET-DEC-2004)

(A) Best fit

(B) First fit

(C) Worst fit

(D) Buddy

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

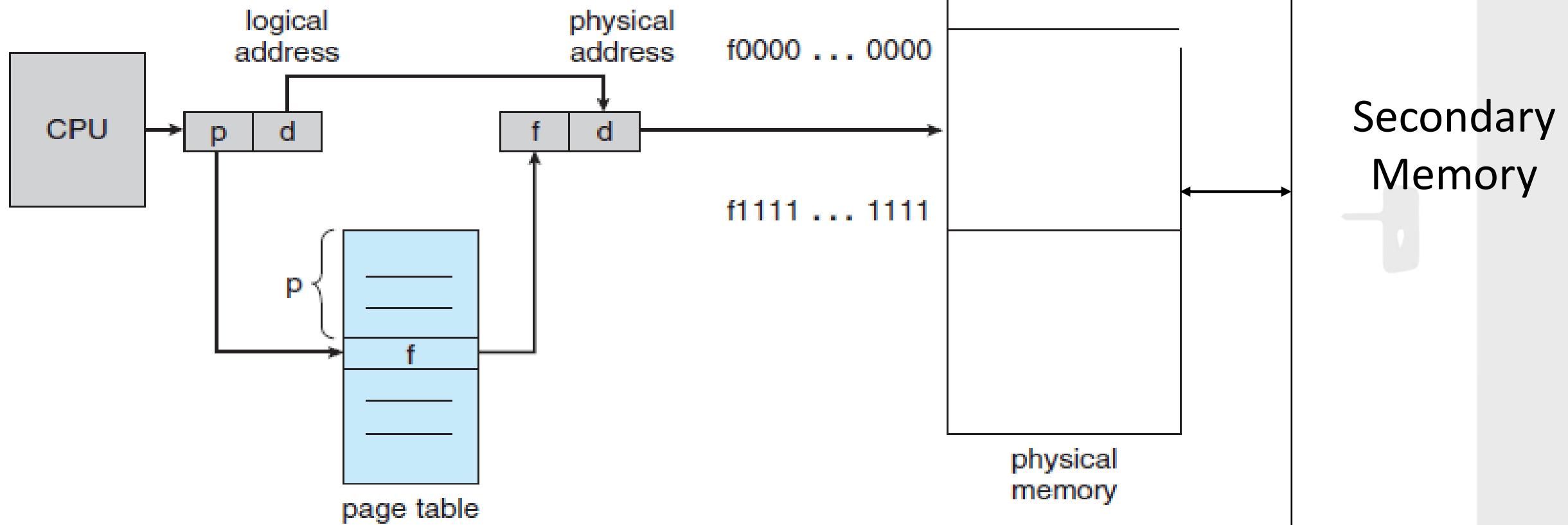
# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

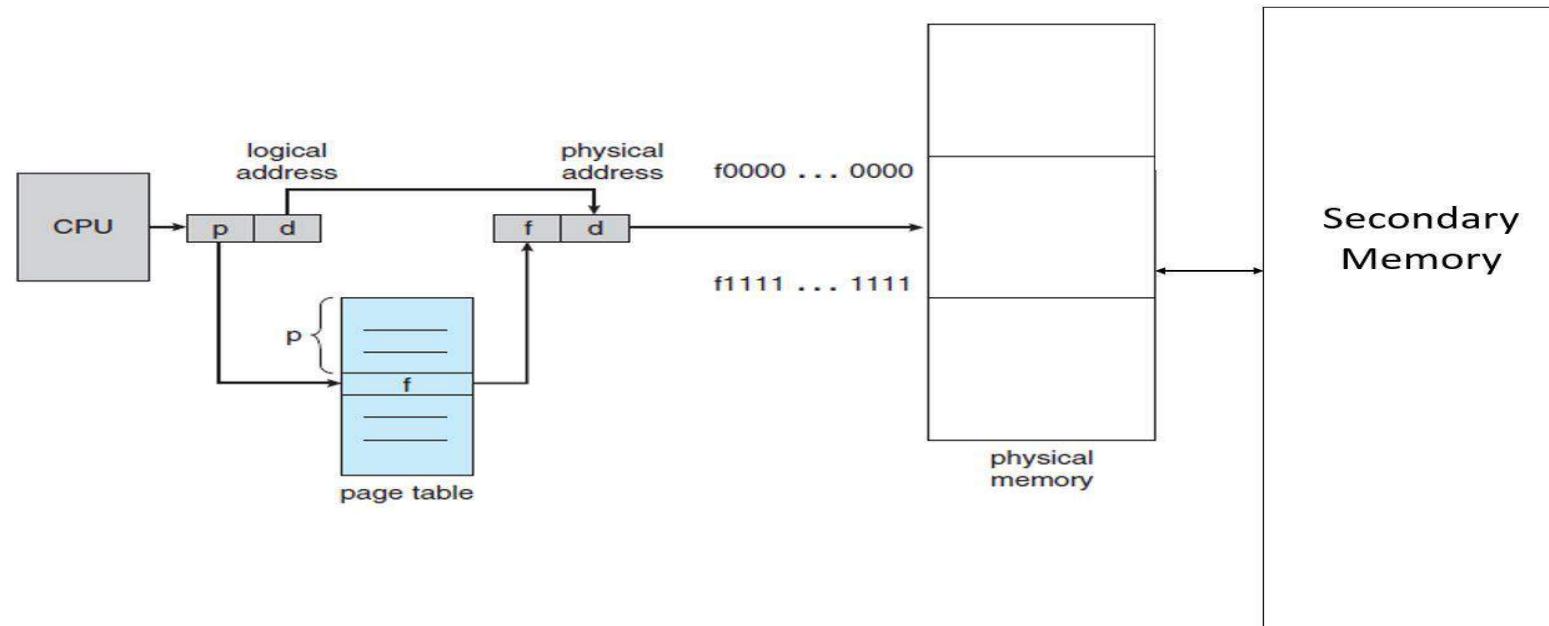
## Non-Contiguous Memory allocation(Paging)

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentation





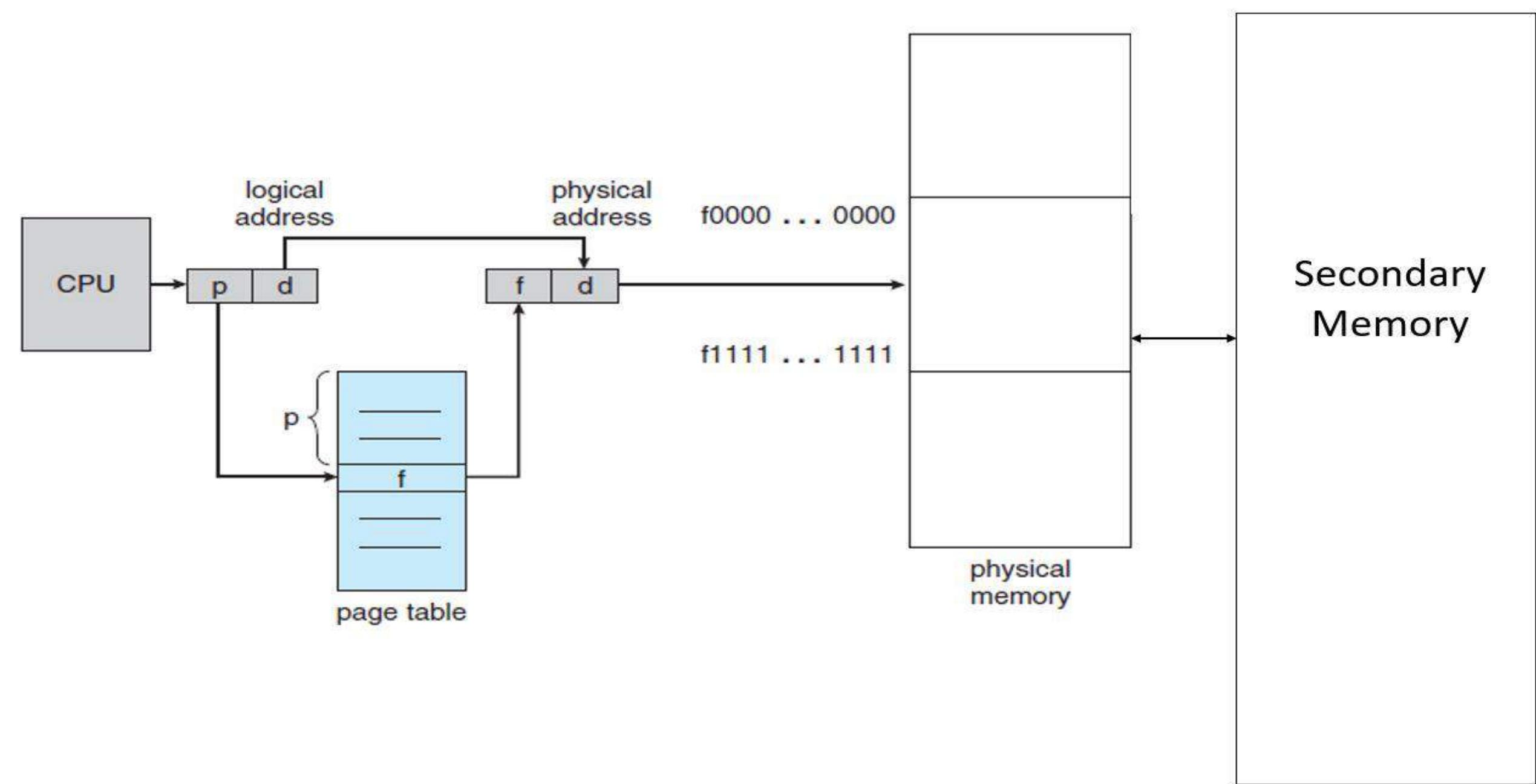
- Secondary memory is divided into fixed size partition (because management is easy) all of them of same size called pages (easy swapping and no external fragmentation).
- Main memory is divided into fix size partitions (because management is easy), each of them having same size called frames (easy swapping and no external fragmentation).
- Size of frame = size of page
- In general number of pages are much more than number of frames (approx. 128 time)



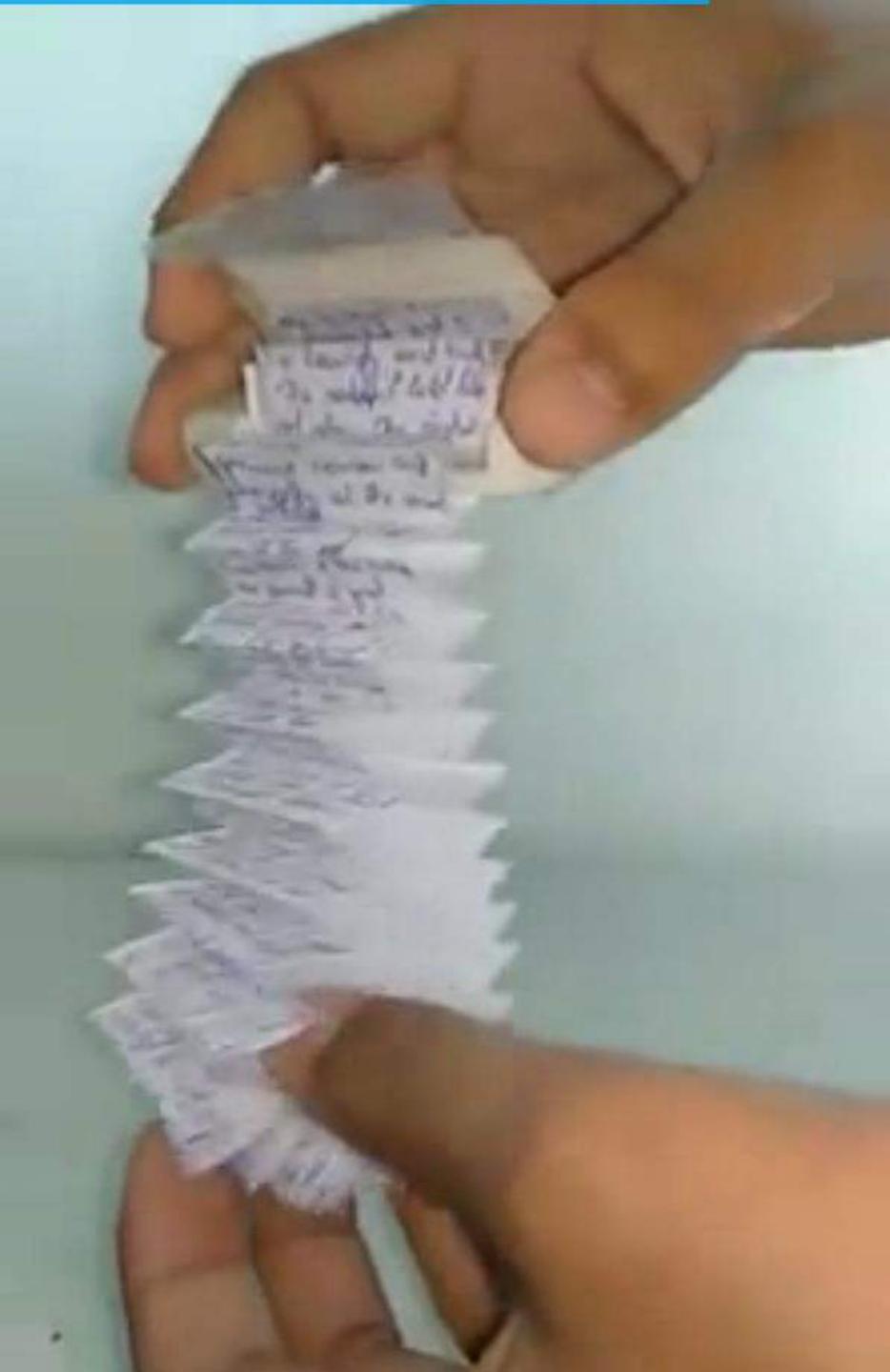
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



To access all paid content get KG Prime at ₹25/day [CLICK HERE](#) 



ent get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## Translation process

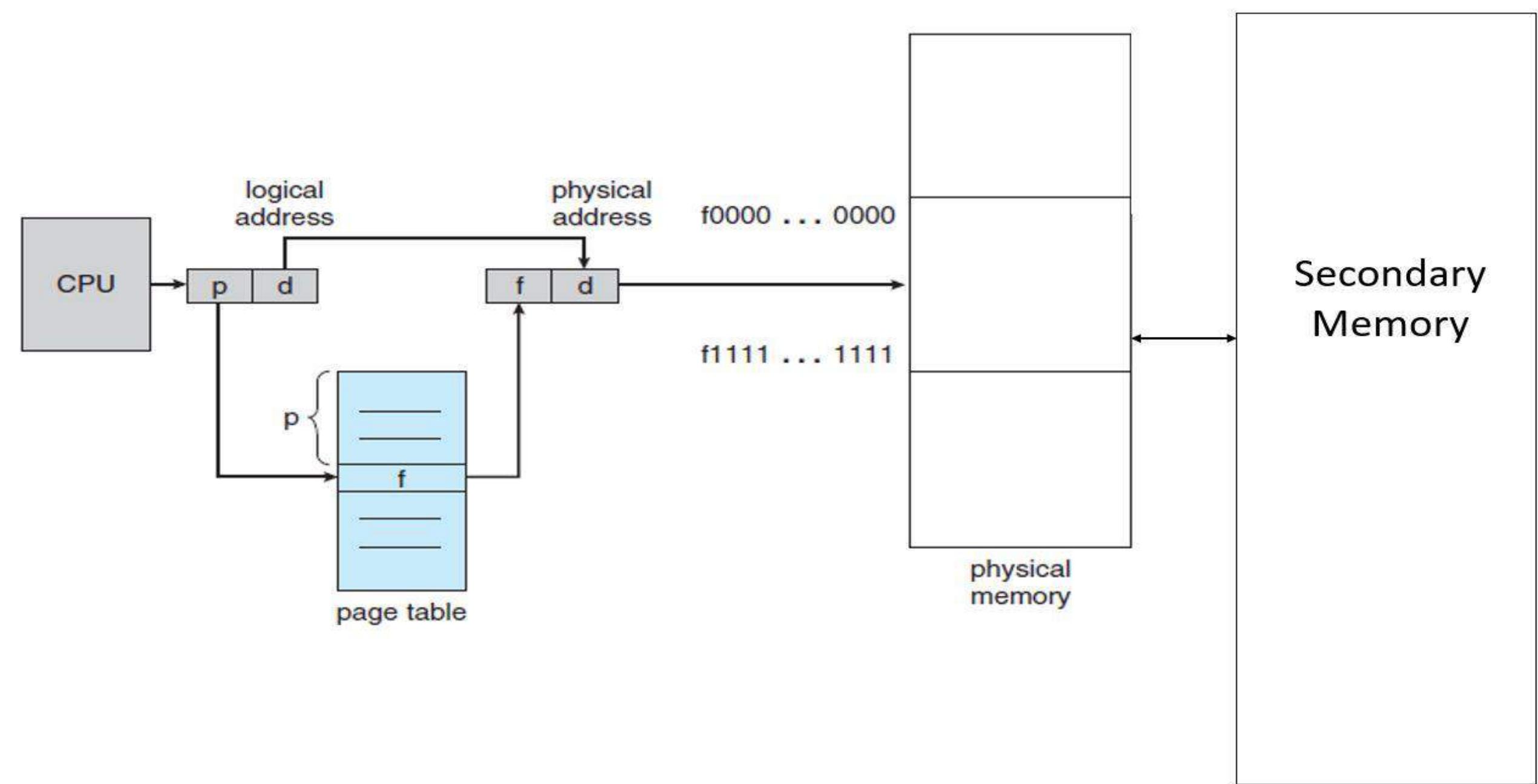
1. CPU generate a logical address is divided into two parts - **p** and **d**
  1. where p stands for page no and d stands for instruction offset.
2. The **page number(p)** is used as an index into a **Page table**
3. **Page table base register(PTBR)** provides the base of the page table and then the corresponding page no is accessed using p.
4. Here we will finds the corresponding frame no (the base address of that frame in main memory in which the page is stored)
5. Combine corresponding frame no with the instruction offset and get the physical address. Which is used to access main memory.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Page Table

1. Page table is a data structure not hardware.
2. Every process have a separate page table.
3. Number of entries a process have in the page table is the number of pages a process have in the secondary memory.
4. Size of each entry in the page table is same it is corresponding frame number.
5. Page table is a data structure which is it self stored in main memory.



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Advantage

- Removal of External Fragmentation



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Disadvantage

- Translation process is slow as Main Memory is accessed two times(one for page table and other for actual access).
- A considerable amount of space is wasted in storing page table(meta data).
- System suffers from internal fragmentation(as paging is an example of fixed size partition).
- Translation process is difficult and complex to understand and implement.

**Q** Page information in memory is also called as Page Table. The essential contents in each entry of a page table is/are \_\_\_\_\_. (NET-JULY-2018)

**(a)** Page Access information

**(b)** Virtual Page number

**(c)** Page Frame number

**(d)** Both virtual page number and Page Frame Number

**Q The essential content(s) in each entry of a page table is / are? (GATE-2009) (1 Marks)**

**(A) Virtual page number**

**(B) Page frame number**

**(C) Both virtual page number and page frame number**

**(D) Access right information**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

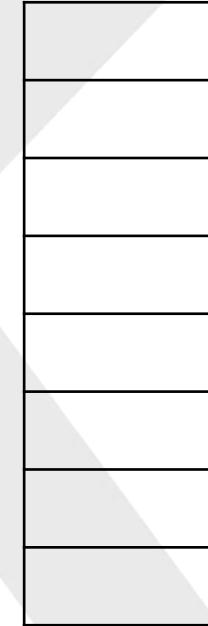
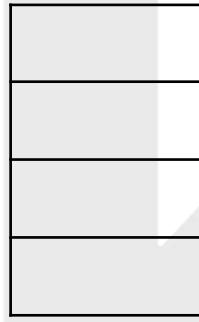
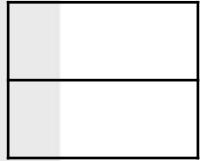
$10^3$	1 Thousand
$10^6$	1 Million
$10^9$	1 Billion
$10^{12}$	1 Trillion

$10^3$	1 kilo
$10^6$	1 Mega
$10^9$	1 Giga
$10^{12}$	1 Tera
$10^{15}$	1 Peta
$10^{18}$	1 Exa
$10^{21}$	1 Zetta
$10^{24}$	1 Yotta

$2^{10}$	1 kilo
$2^{20}$	1 Mega
$2^{30}$	1 Giga
$2^{40}$	1 Tera
$2^{50}$	1 Peta
$2^{60}$	1 Exa
$2^{70}$	1 Zetta
$2^{80}$	1 Yotta

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

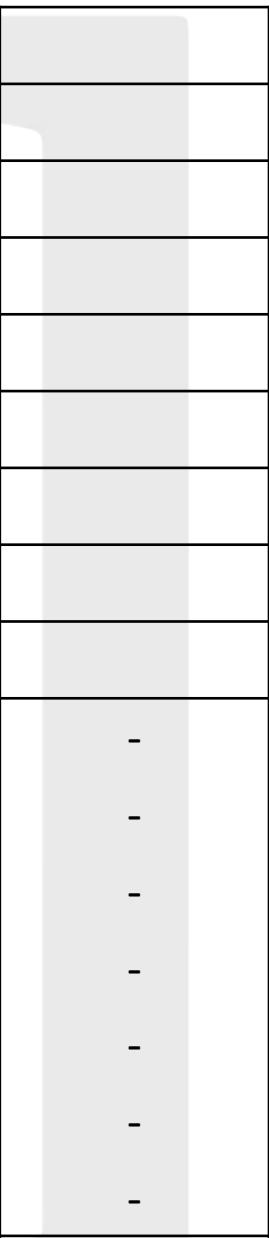
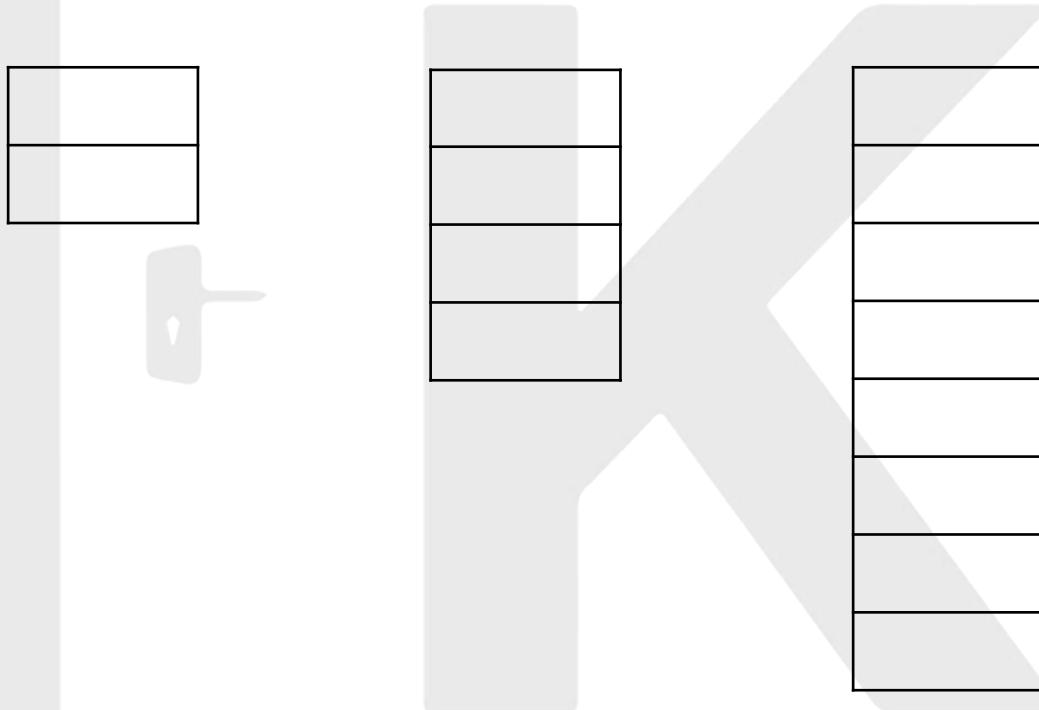
Address Length in bits	$n$
No of Locations	$2^n$



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Address Length in bits	$n$
No of Locations	$2^n$

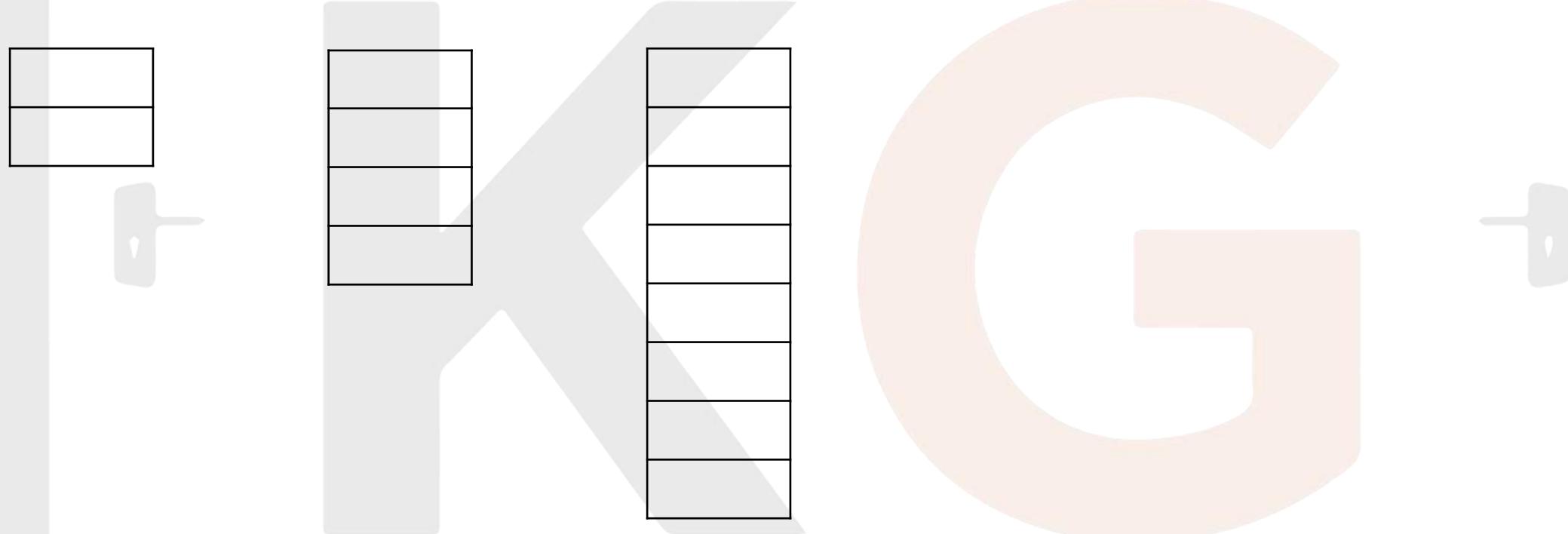


Memory Size = Number of Location \* Size of each Location

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE** ↗

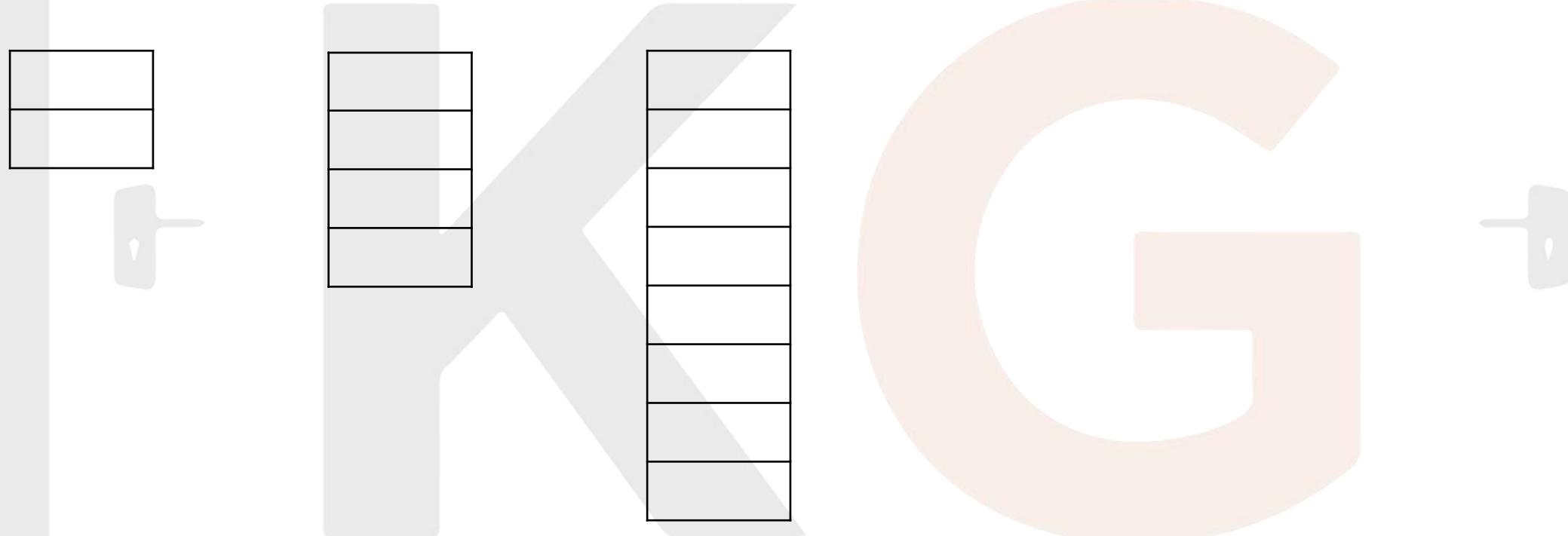
No of Locations	n
Address Length in bits	Upper Bound( $\log_2 n$ )



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE**

No of Locations	n
Address Length in bits	Upper Bound( $\log_2 n$ )

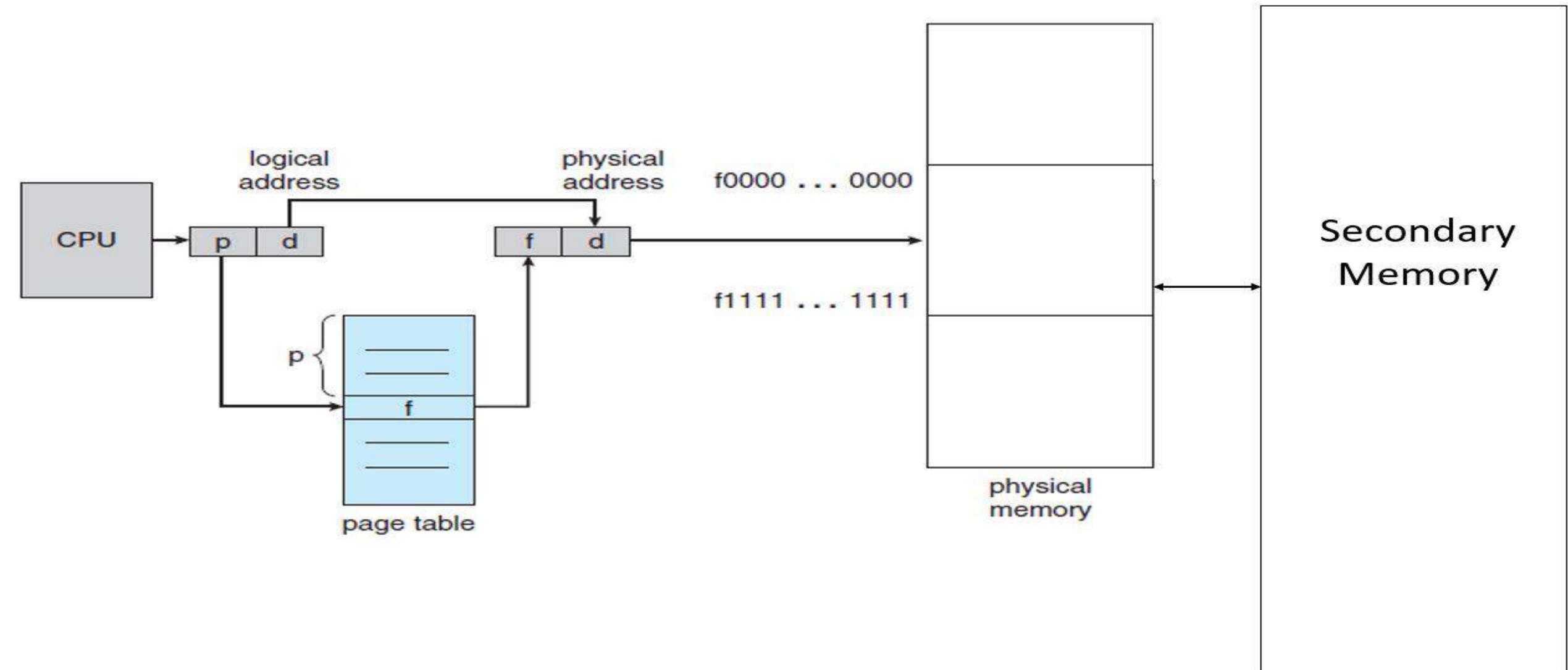


Memory Size/ Size of each Location = Number of Location

To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE](#) ↗

- Page Table Size = No of entries in Page table \* Size of each entry(f)
- Process Size = No of Pages \* Size of each page



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

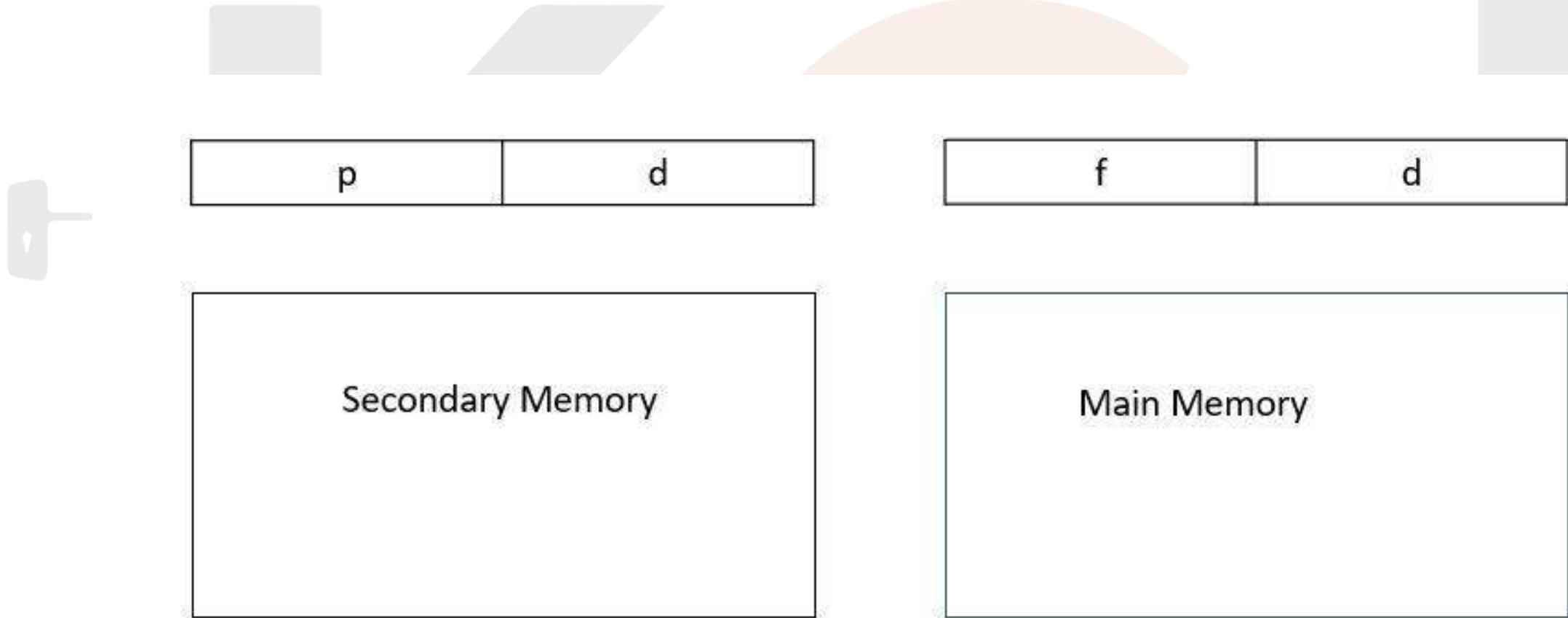
Logical Address in bits

Secondary Memory

Physical Address in bits

Main Memory

p  
f  
d



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

S No	SM	LA	MM	PA	p	f	d	addressable	Page Size
1	32 GB		128 MB					1B	1KB
2		42		33			11	1B	
3	512GB			31				1B	512B
4	128GB		32GB		30			1B	
5					28	14			4096B

**p**

**d**

**f**

**d**

**Secondary Memory**

**Main Memory**

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q A memory management system has 64 pages with 512 bytes page size. Physical memory consists of 32 page frames. Number of bits required in logical and physical address are respectively (NET-DEC-2016)

a) 14 and 15



b) 14 and 29



c) 15 and 14

d) 16 and 32

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Consider a system with byte-addressable memory, 32-bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is \_\_\_\_\_ (GATE-2015) (2 Marks)

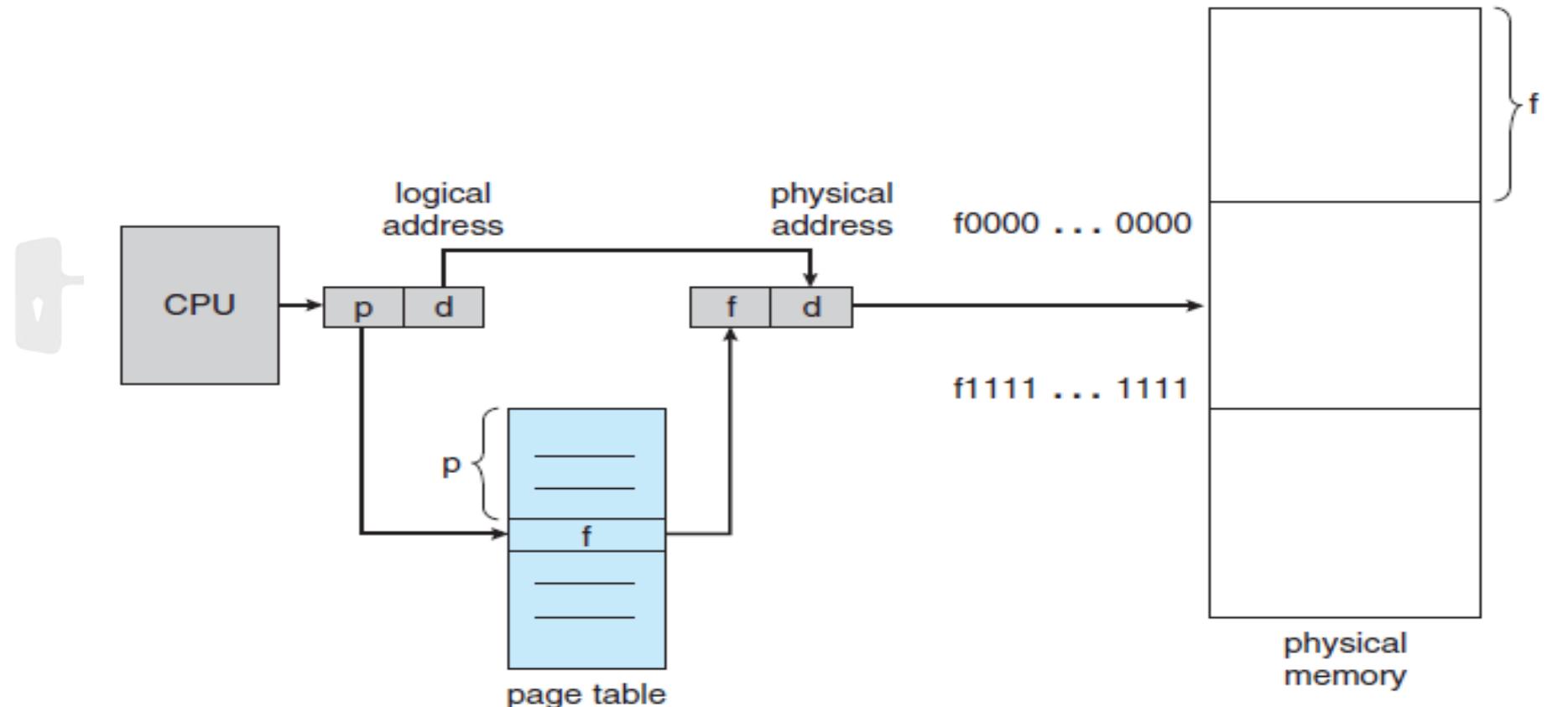


Figure 8.10 Paging hardware.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A Computer system implements 8 kilobyte pages and a 32-bit physical address space. Each page table entry contains a valid bit, a dirty bit three permission bits, and the translation. If the maximum size of the page table of a process is 24 megabytes, the length of the virtual address supported by the system is \_\_\_\_\_ bits? (GATE-2015) (2 Marks)

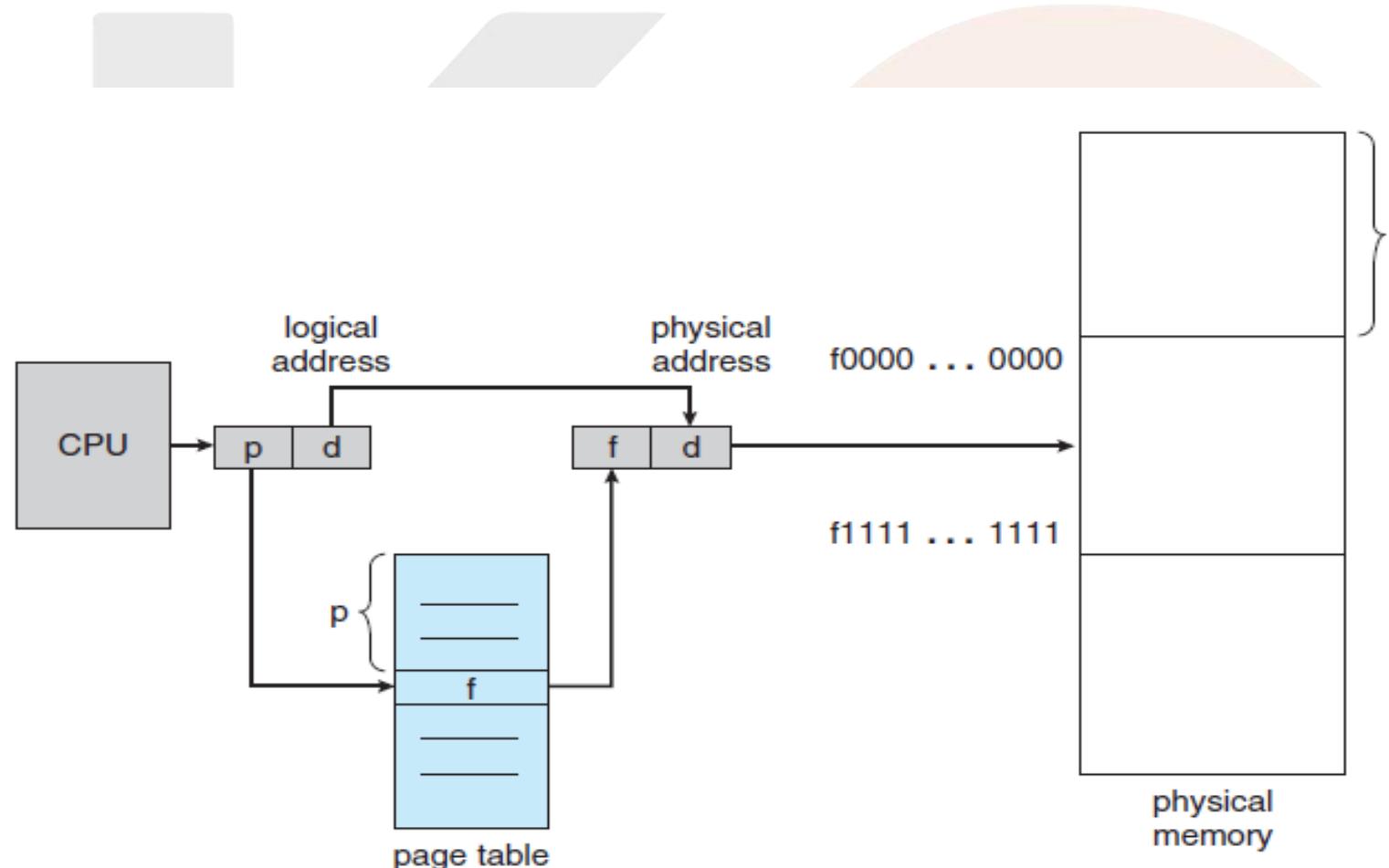


Figure 8.10 Paging hardware.

To access a

CLICK HERE ↗

**Q** A computer system supports 32-bit virtual address as well as 32-bit physical addresses. Since the virtual address space is of same size as that of physical address space, if we want to get rid of virtual memory, which one of the following is true? **(NET-JUNE-2012)**

- (A)** Efficient implementation of multiuser support is no longer possible.
- (B)** The processor cache can be made more efficient.
- (C)** Hardware support for memory management is not needed.
- (D)** CPU scheduling can be made more efficient.

**Q** Consider a logical address space of 8 pages of 1024 words mapped with memory of 32 frames. How many bits are there in the physical address? **(NET-DEC-2011)**

- (A)** 9 bits      **(B)** 11 bits      **(C)** 13 bits      **(D)** 15 bits



**Q** Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size is 4KB, what is the approximate size of the page table? **(GATE-2001) (2 Mark)**

- (a) 16 MB      (b) 8 MB      (c) 2 MB      (d) 24 MB

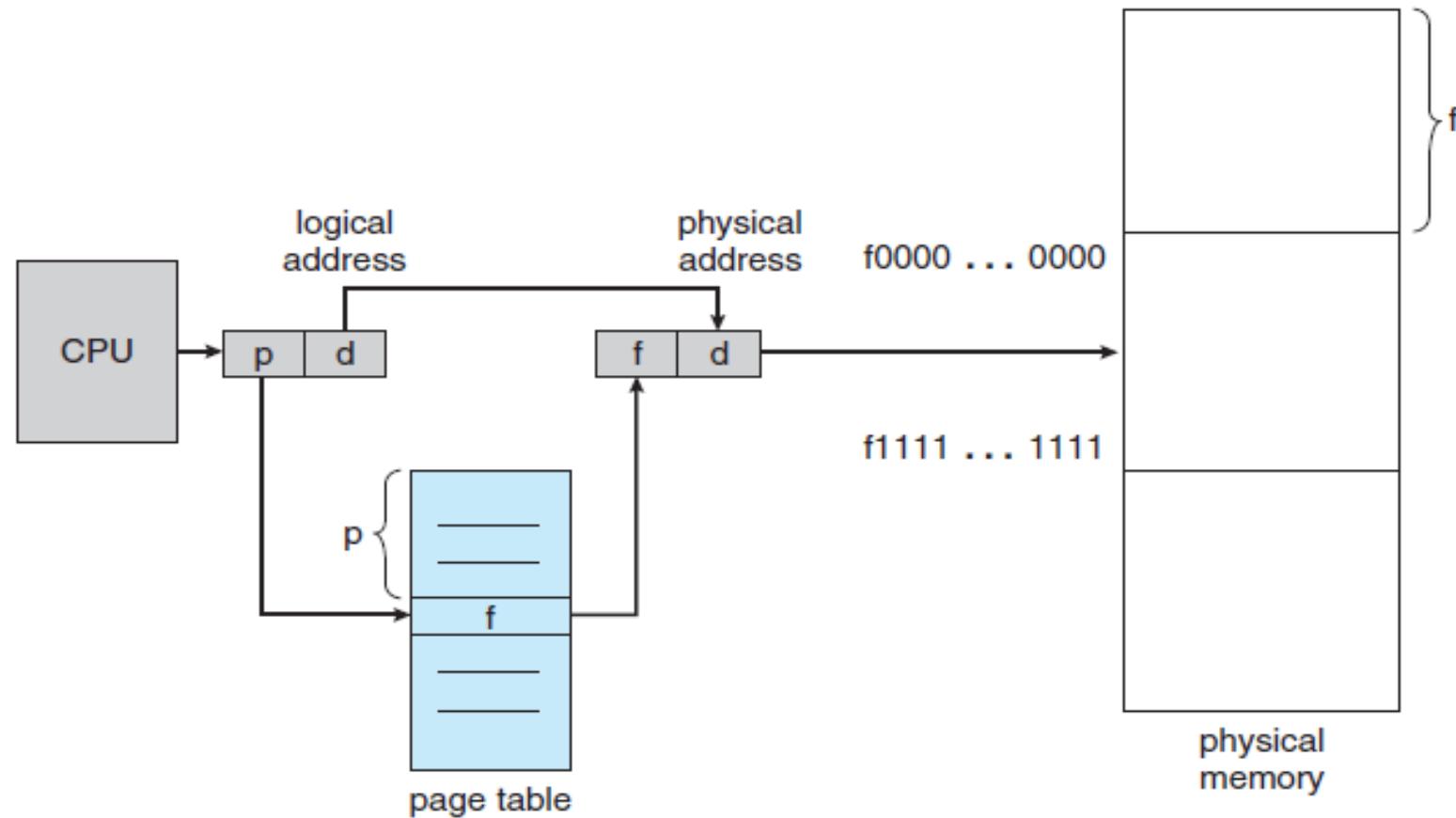


Figure 8.10 Paging hardware.

To access

CLICK HERE ↗

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

S No	SM	LA	MM	PA	p	f	d	addressable	Process Size	PT Size
1	16 GB			26			10	1B		2MB
2		44		36			12	1B	1GB	
3		39		28			8	1B	512MB	

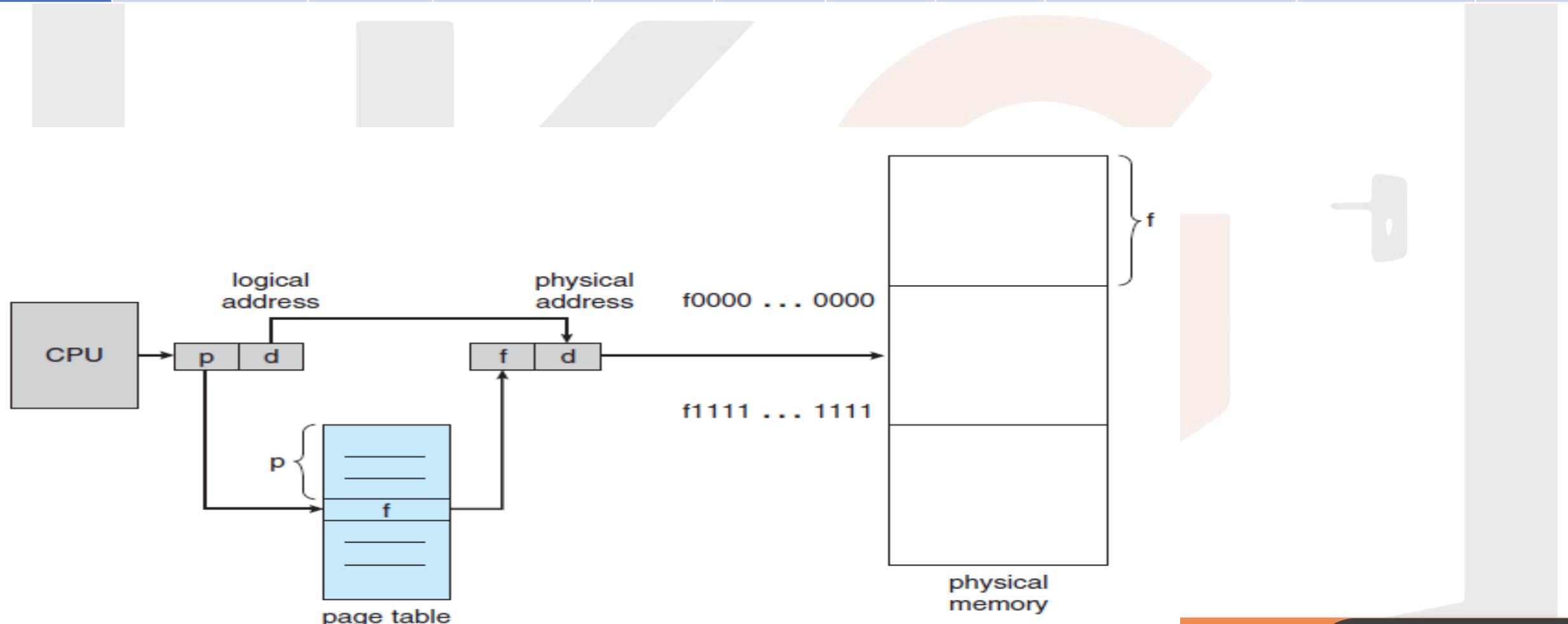


Figure 8.10 Paging hardware.

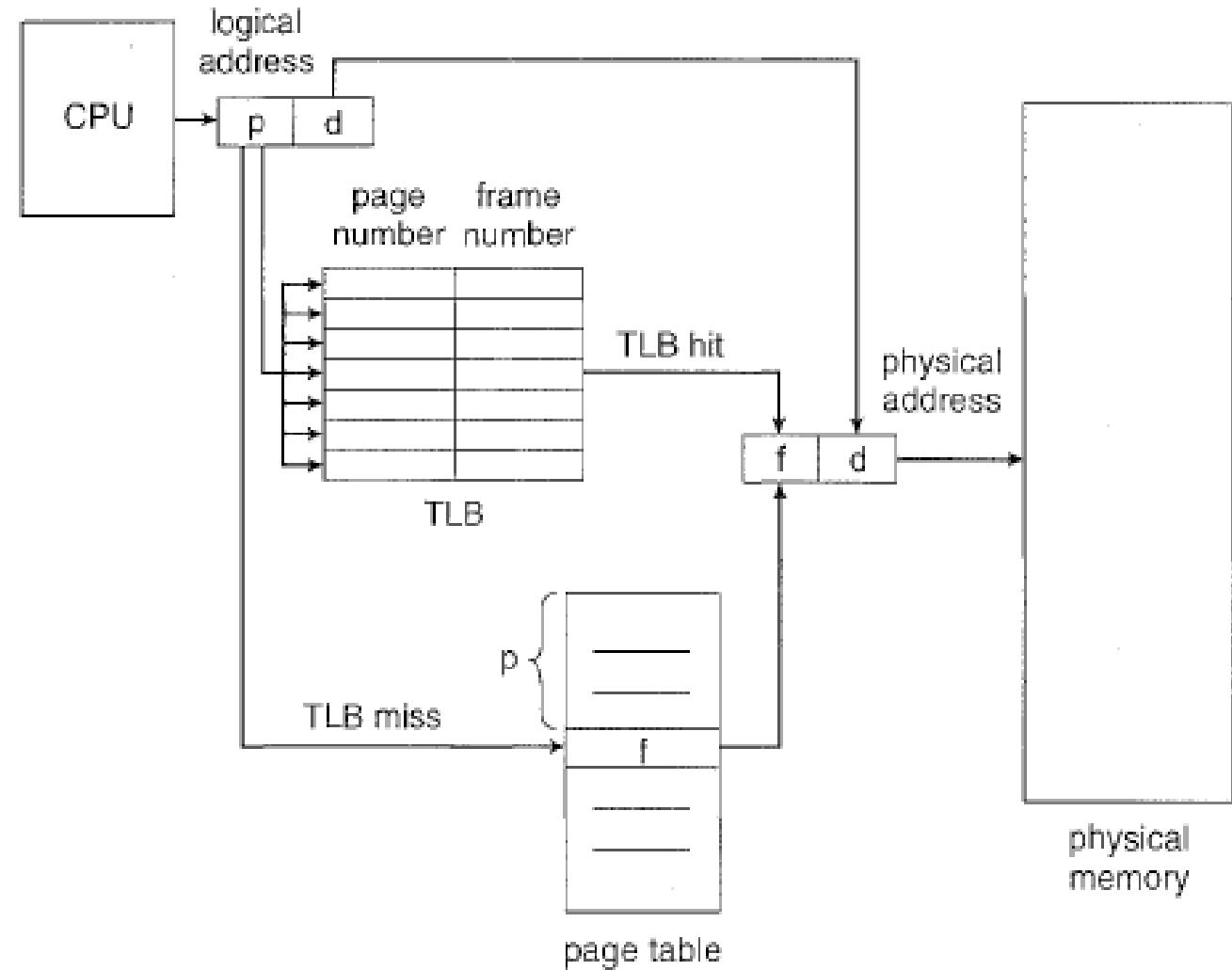
5/day **CLICK HERE** ↗

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

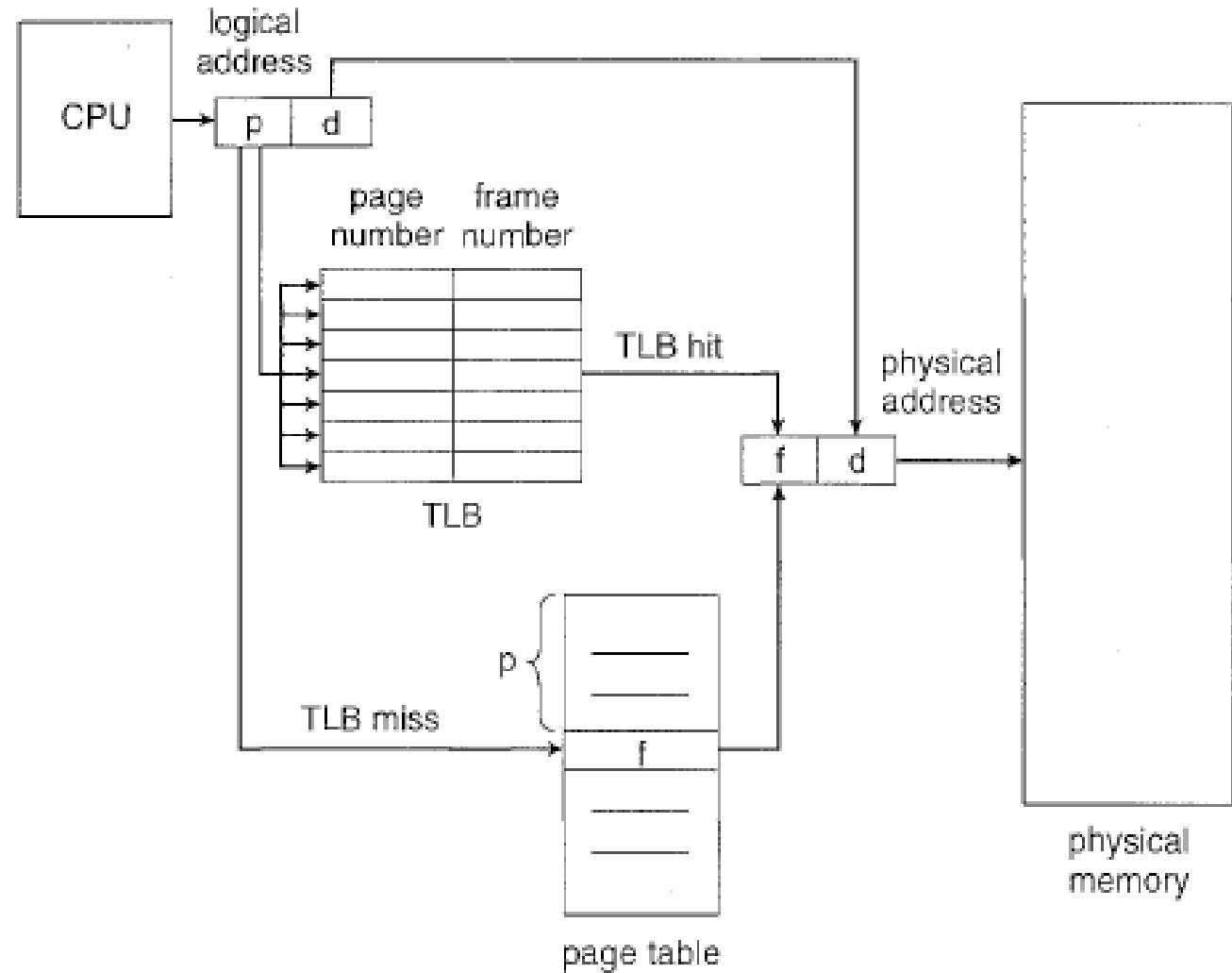
- A serious problem with page is, translation process is slow as page table is accessed two times (one for page table and other for actual access).

- To solve the problems in paging we take the help of TLB. The TLB is associative, high-speed memory.

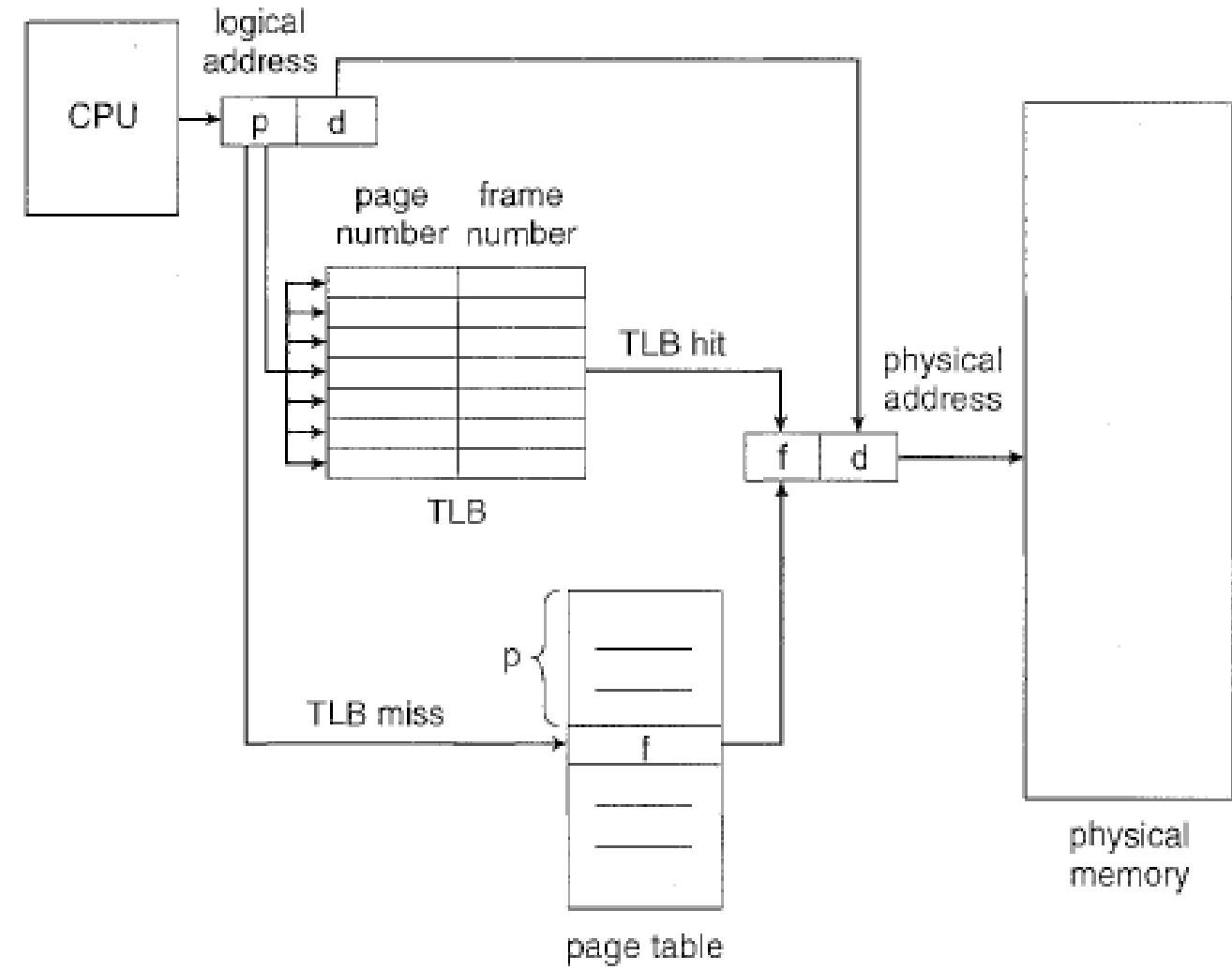


- Each entry in the TLB consists of two parts: a key (Page no) and a value (frame no). When the associative memory is search for page no, the page no is compared with all page no simultaneously. If the item is found, the corresponding frame no field is returned.

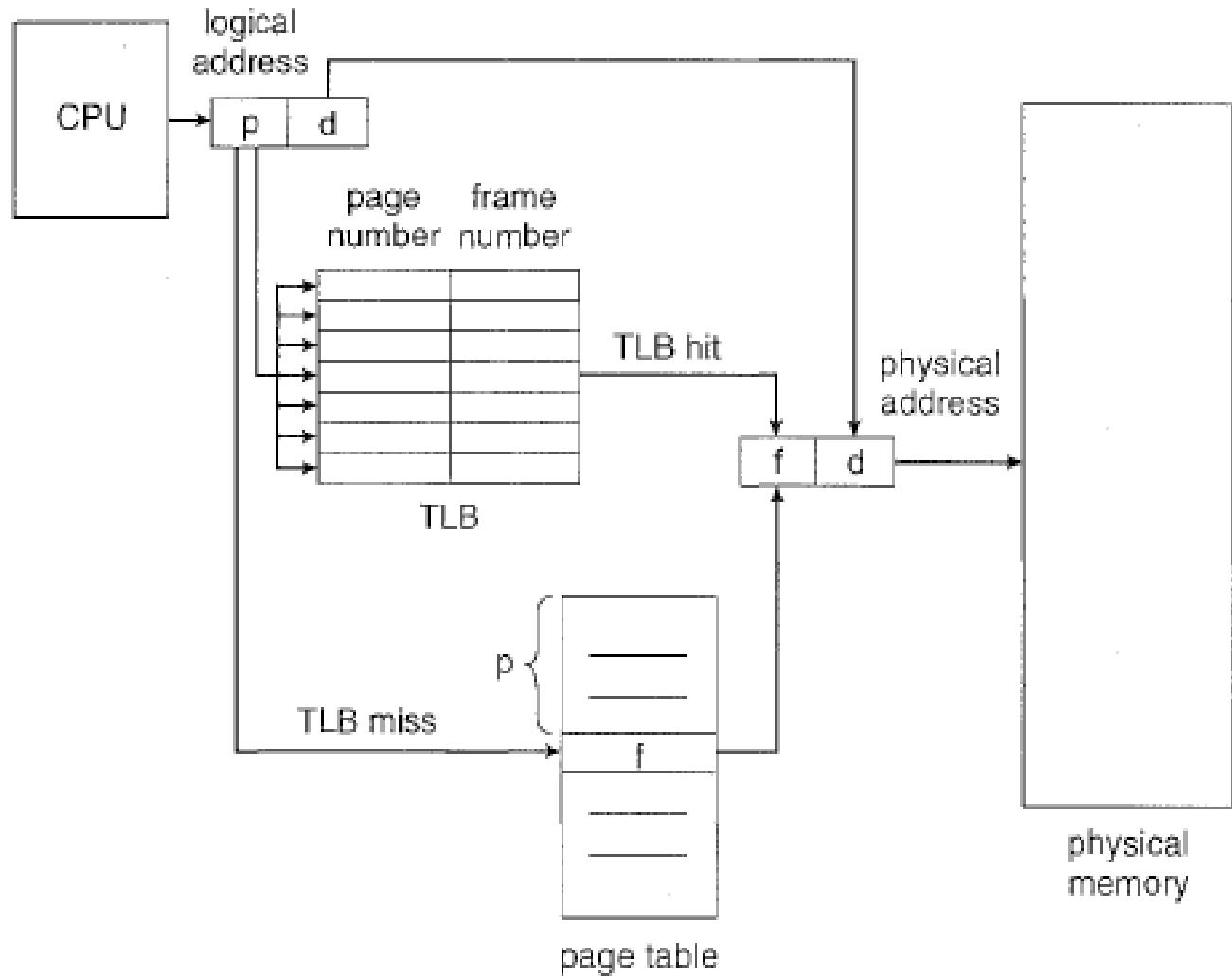
- The search is fast; **the hardware, however, is expensive**, TLB Contains the frequently referred page numbers and corresponding frame number.



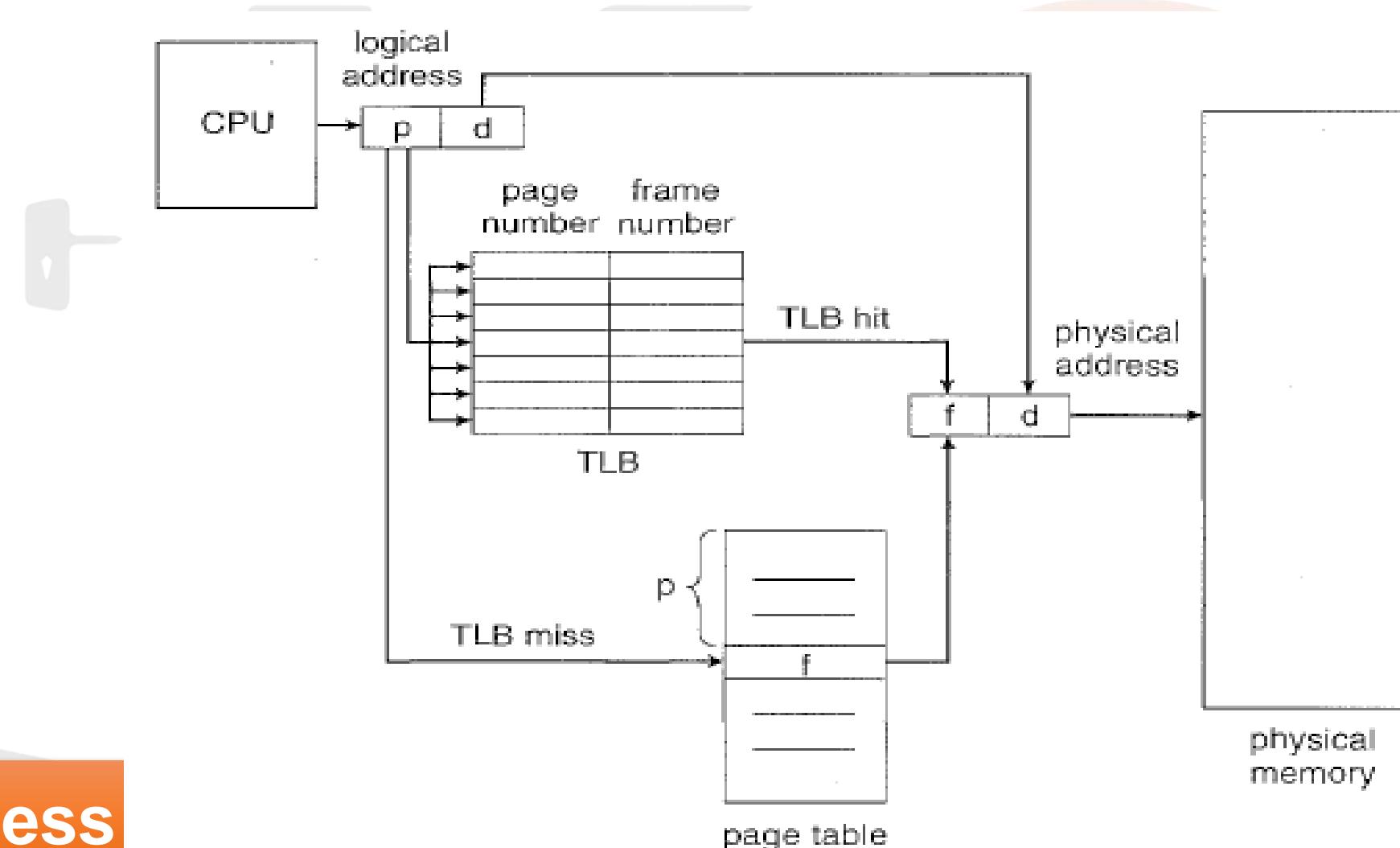
- The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory.
- If the page number is not in the TLB (known as a **TLB Miss**), then a memory reference to the page table must be made.



- Also we add the page number and frame number to the TLB, so that they will be found quickly on the next reference.
- If the TLB is already full of entries, the operating system must select one for replacement i.e. Page replacement policies.
- The percentage of times that a particular page number is found in the TLB is called the **Hit Ratio**.



- **Effective Memory Access Time:**
  - Hit [TLB + Main Memory] + 1-Hit [TLB + 2 Main Memory]
- **TLB removes the problem of slow access.**



To access

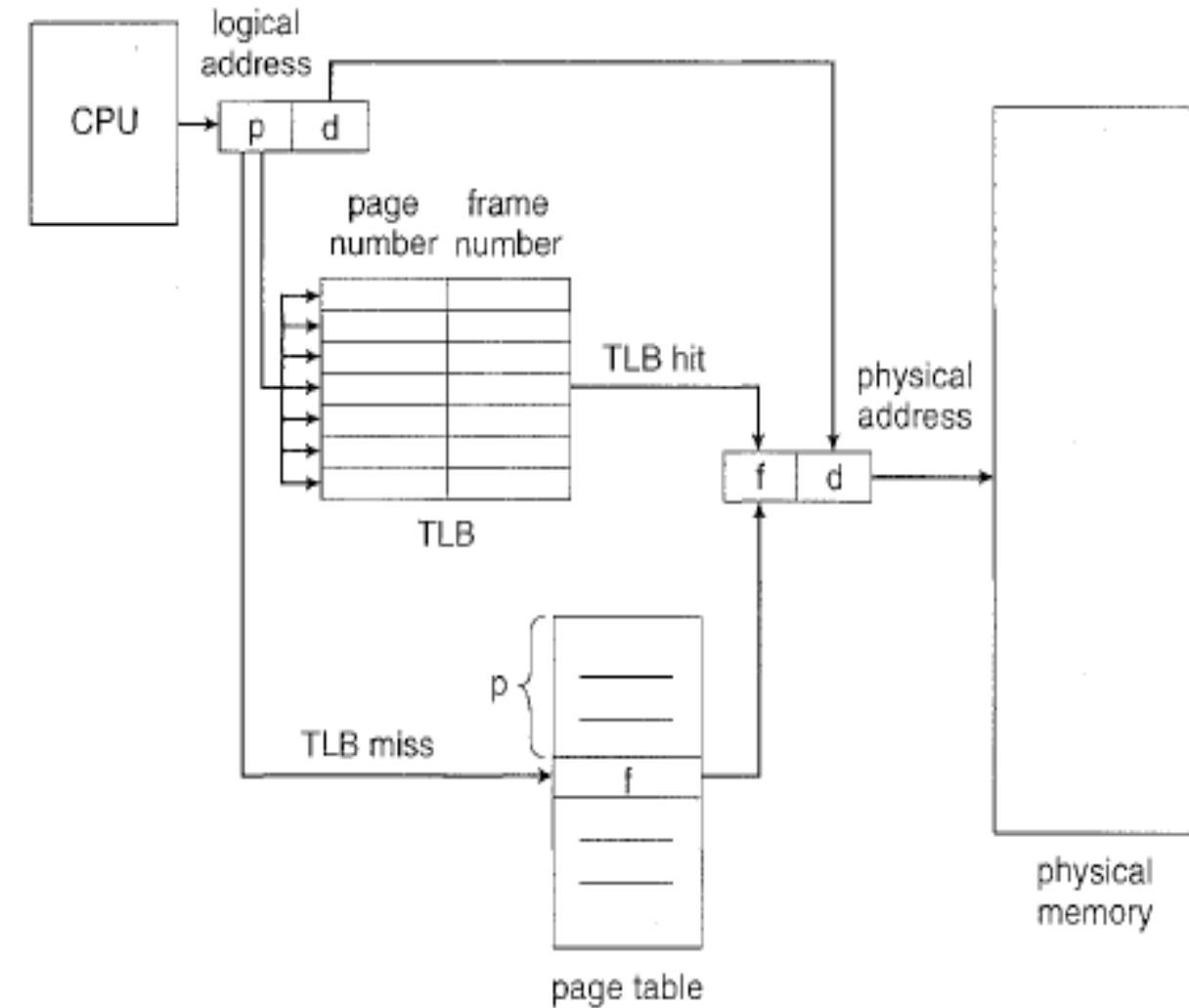
CLICK HERE ↗

- **Disadvantage of TLB:**

- TLB can hold the data of one process at a time and in case of multiple context switches TLB will be required to flush frequently.

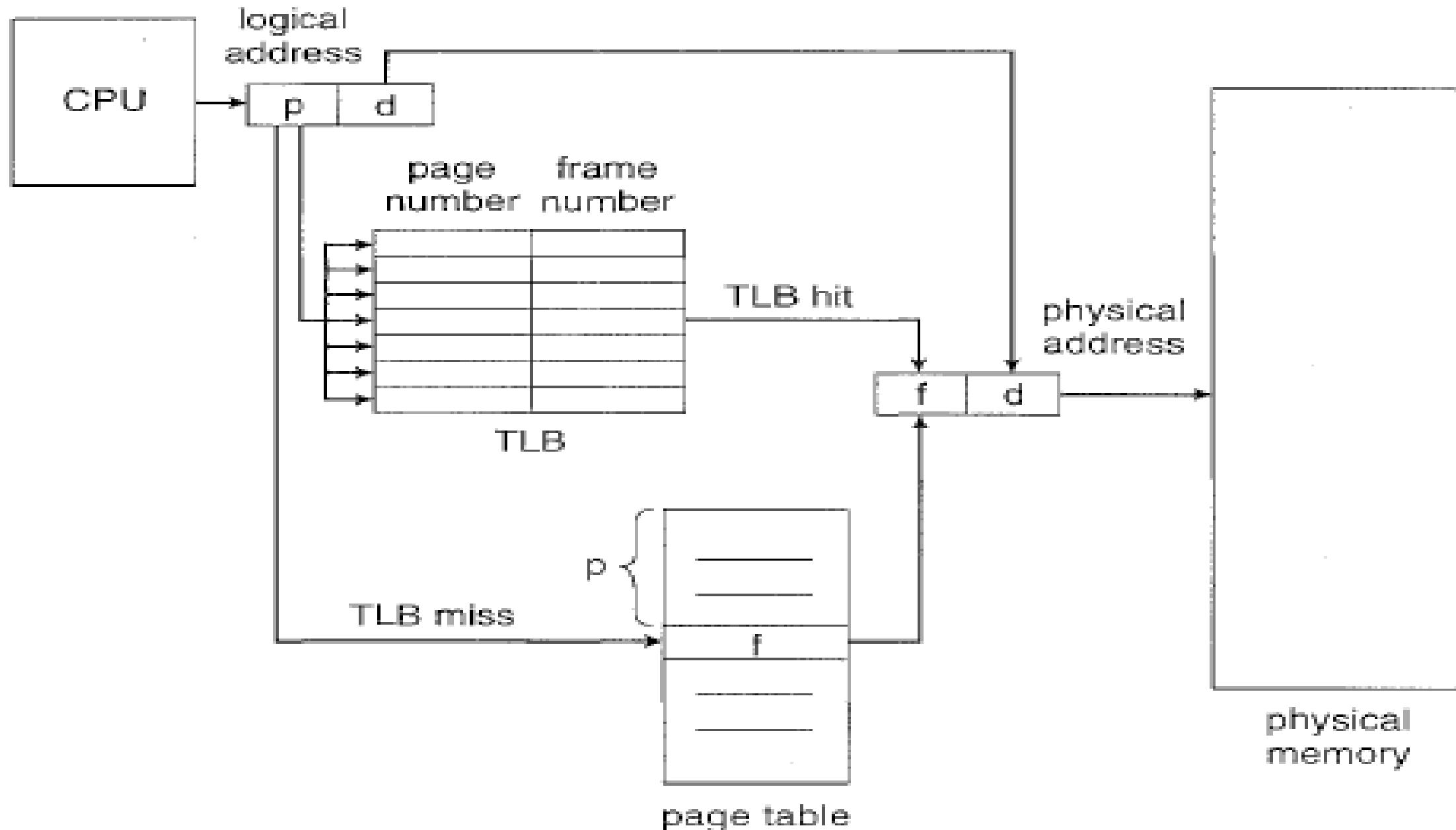
- **Solution:**

- Use multiple TLB's but it will be costly.
- Some TLBs allow certain entries to be **wired down**, meaning that they cannot be removed from the TLB. Typically, TLB entries for kernel code are wired down.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Assume that in a certain computer, the virtual addresses are 64 bits long and the physical addresses are 48 bits long. The memory is word addressable. The page size is 8kB and the word size is 4 bytes. The Translation Look-aside Buffer (TLB) in the address translation path has 128 valid entries. At most how many distinct virtual addresses can be translated without any TLB miss? **(GATE-2019) (2 Marks)**

(A)  $16 \times 2^{10}$

(B)  $8 \times 2^{20}$

(C)  $4 \times 2^{20}$

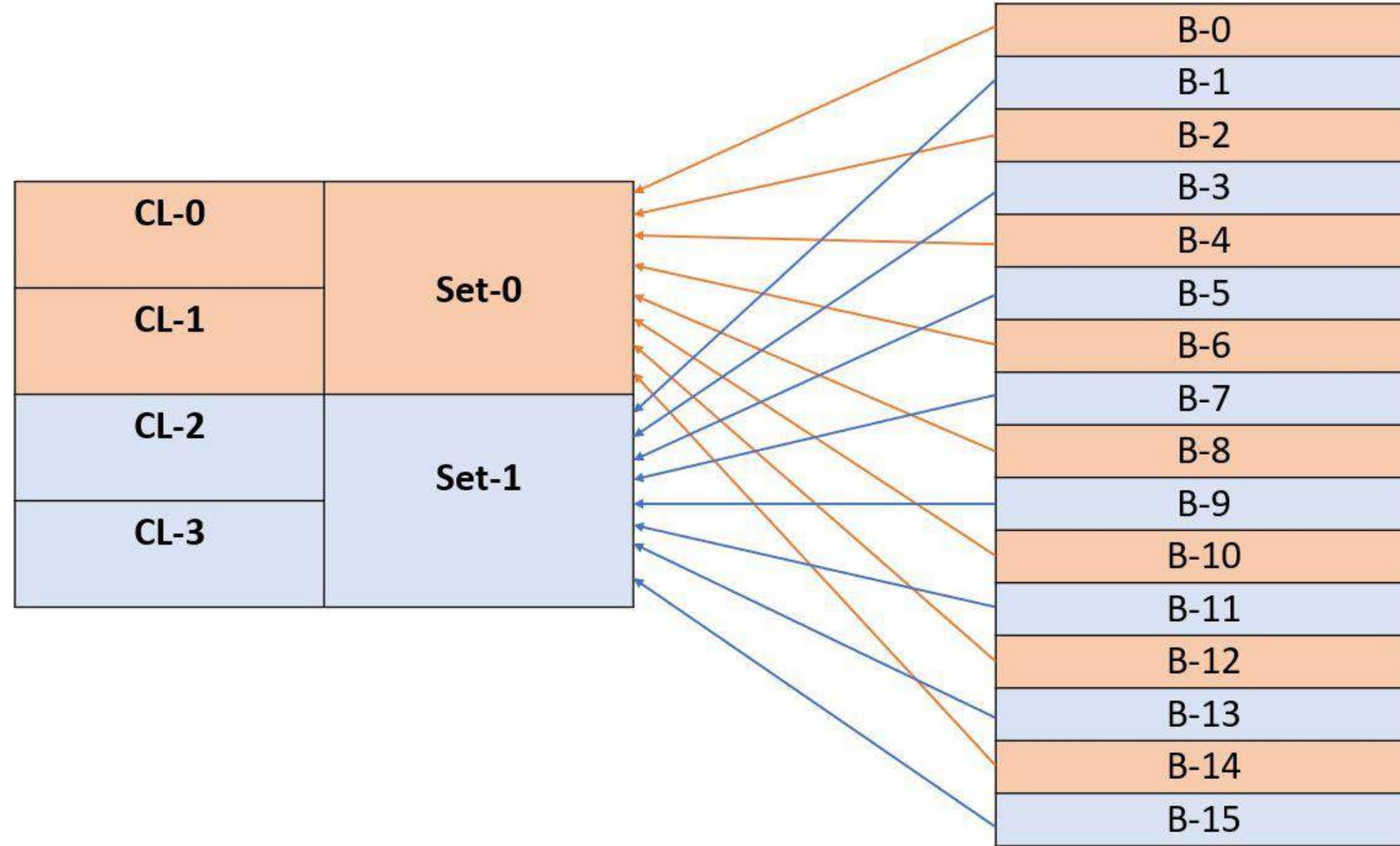
(D)  $256 \times 2^{10}$

Q In a paging system, it takes 30 ns to search translation Look-a-side Buffer (TLB) and 90 ns to access the main memory. If the TLB hit ratio is 70%, the effective memory access time is : (NET-JAN-2017)

- a) 48ns
- b) 147ns
- c) 120ns
- d) 84ns

Q A computer system implements a 40-bit virtual address, page size of 8 kilobytes, and a 128-entry translation look-aside buffer (TLB) organized into 32 sets each having four ways. Assume that the TLB tag does not store any process id. The minimum length of the TLB tag in bits is \_\_\_\_\_ (GATE-2015) (2 Marks)

- (A) 20
- (B) 10
- (C) 11
- (D) 22



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is \_\_\_\_\_. (CS-2014) (2 Marks)

**Q** Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is \_\_\_\_\_. (NET-JUNE-2014)

- (A) 120      (B) 122      (C) 124      (D) 118

**Q** The hit ratio of a Translation Look Aside Buffer (TLAB) is 80%. It takes 20 nanoseconds (ns) to search TLAB and 100 ns to access main memory. The effective memory access time is \_\_\_\_\_. **(NET-SEP-2013)**

- (A) 36 ns
- (B) 140 ns
- (C) 122 ns
- (D) 40 ns

**Q Translation Look-aside Buffer (TLB) is (NET-DEC-2013)**

- (A)** a cache-memory in which item to be searched is compared one-by-one with the keys.
- (B)** a cache-memory in which item to be searched is compared with all the keys simultaneously.
- (C)** an associative memory in which item to be searched is compared one-by-one with the keys.
- (D)** an associative memory in which item to be searched is compared with all the keys simultaneously.

**Q** The virtual address generated by a CPU is 32 bits. The Translation Look-aside Buffer (TLB) can hold total 64-page table entries and a 4-way set associative (i.e. with 4- cache lines in the set). The page size is 4 KB. The minimum size of TLB tag is (NET-DEC-2013)

- (A) 12 bits
- (B) 15 bits**
- (C) 16 bits
- (D) 20 bits

Q A paging system tlb takes 10ns main memory takes 50ns what is effective memory access time if tlb hit ratio is 90%? **(GATE-2008) (1 Marks)**

a) 54

b) 60

c) 65

d) 75

**Q** A CPU generates 32-bit virtual addresses. The page size is 4 KB. The processor has a translation look-aside buffer (TLB) which can hold a total of 128-page table entries and is 4-way set associative. The minimum size of the TLB tag is? **(GATE-2006) (2 Marks)**

- (A) 11 bits
- (B) 13 bits
- (C) 15 bits
- (D) 20 bits

**Q Which of the following is not a form of memory? (GATE-2002) (1 Marks)**

**(A) instruction cache**

**(B) instruction register**

**(C) instruction opcode**

**(D) translation lookaside buffer**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Size of Page

- If we increase the size of page table then internal fragmentation increase but size of page table decreases.
- If we decrease the size of page then internal fragmentation decrease but size of page table increases.
- So we have to find what should be the size of the page, where both cost are minimal.

**Q** For the implementation of a paging scheme, suppose the average process size be  $x$  bytes, the page size be  $y$  bytes, and each page entry requires  $z$  bytes. The optimum page size that minimizes the total overhead due to the page table and the internal fragmentation loss is given by (NET-DEC-2014)

A)  $x/2$

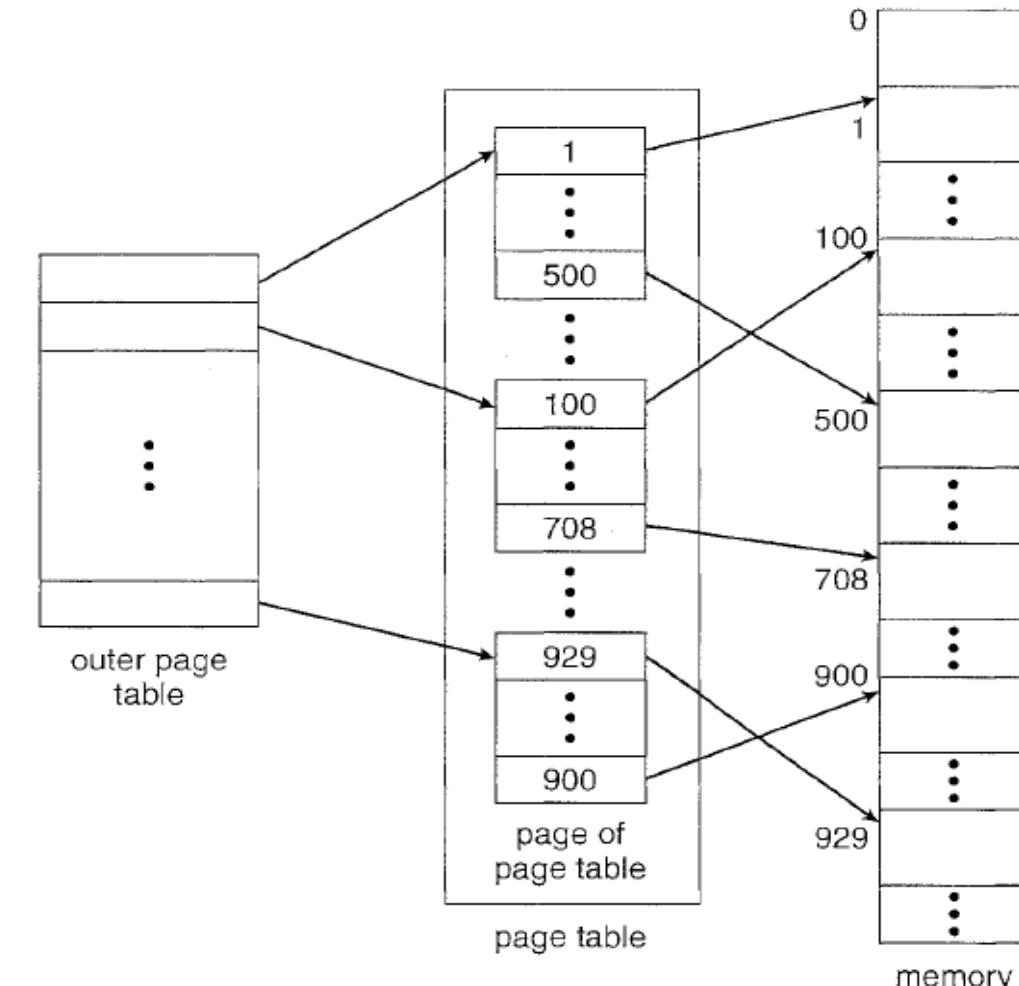
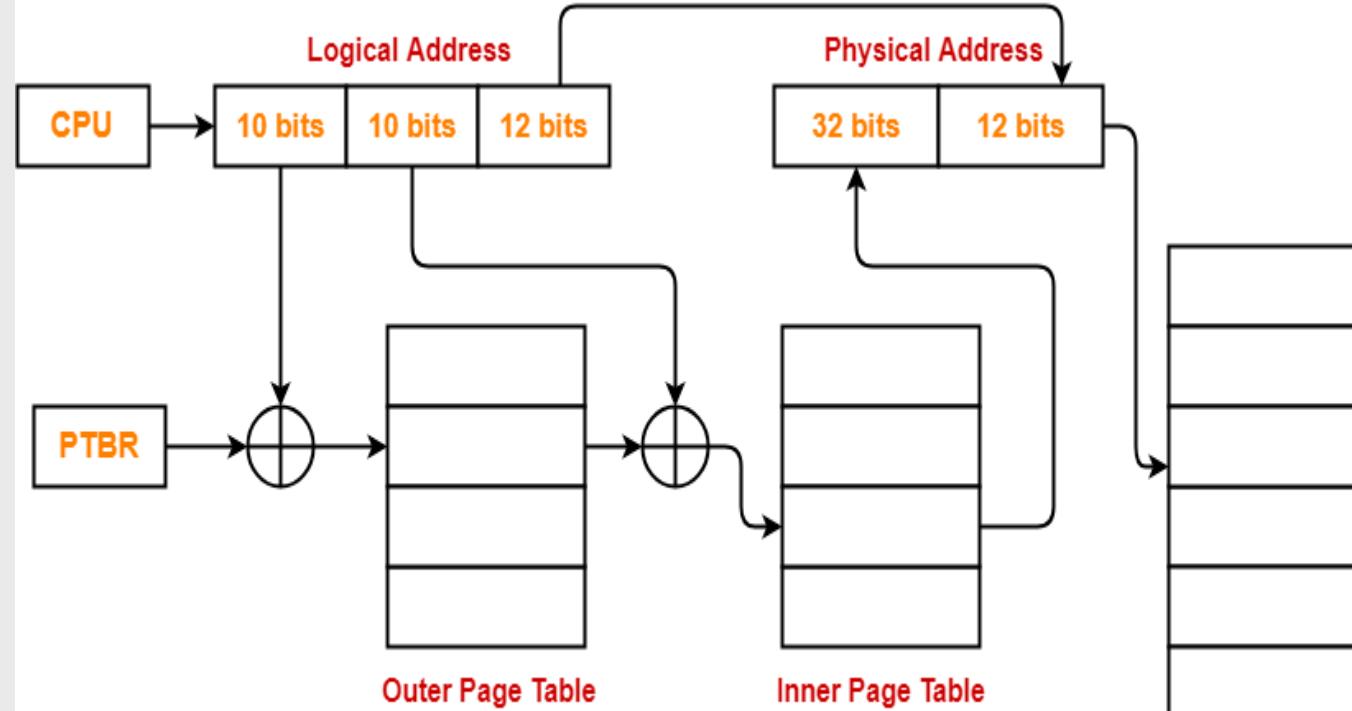
b)  $xz/2$

c)  $\text{root}(2xz)$

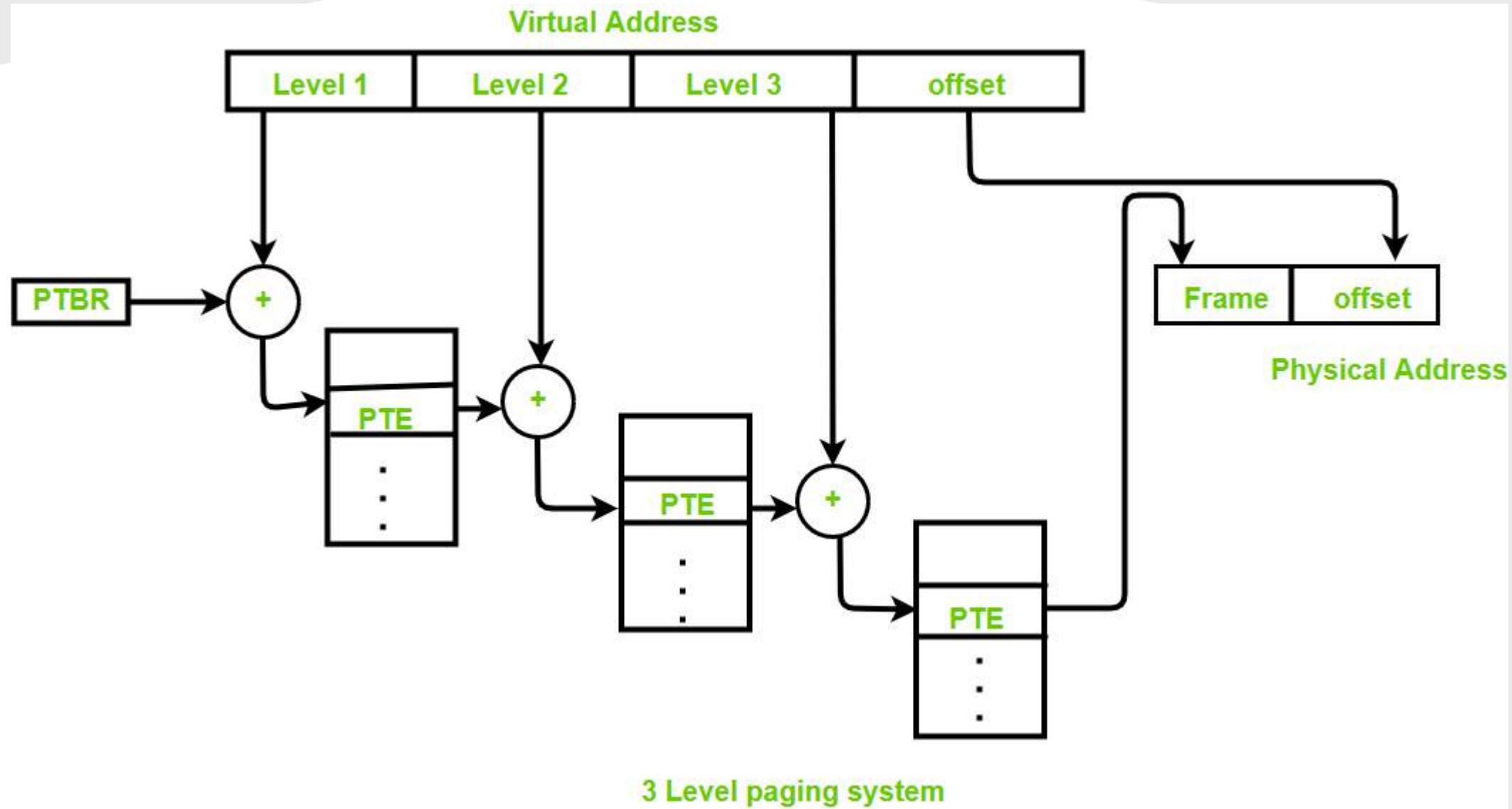
d)  $\text{root}(xz)/2$

## Multilevel Paging / Hierarchical Paging

- Modern systems support a large logical address space ( $2^{32}$  to  $2^{64}$ ).
  - In such cases, the page table itself becomes excessively large and can contain millions of entries and can take a lot of space in memory, so cannot be accommodated into a single frame.
  - A simple solution to this is to divide page table into smaller pieces.
  - One way is to use a **two-level paging algorithm**, in which the page table itself is also paged.



To access all paid content get **KG Premium** Main Memory



**Q** A multilevel page table is preferred in comparison to a single level page table for translating virtual address to physical address because **(GATE-2009) (1 Marks)**

- (A)** It reduces the memory access time to read or write a memory location.
- (B)** It helps to reduce the size of page table needed to implement the virtual address space of a process.
- (C)** It is required by the translation lookaside buffer.
- (D)** It helps to reduce the number of page faults in page replacement algorithms.

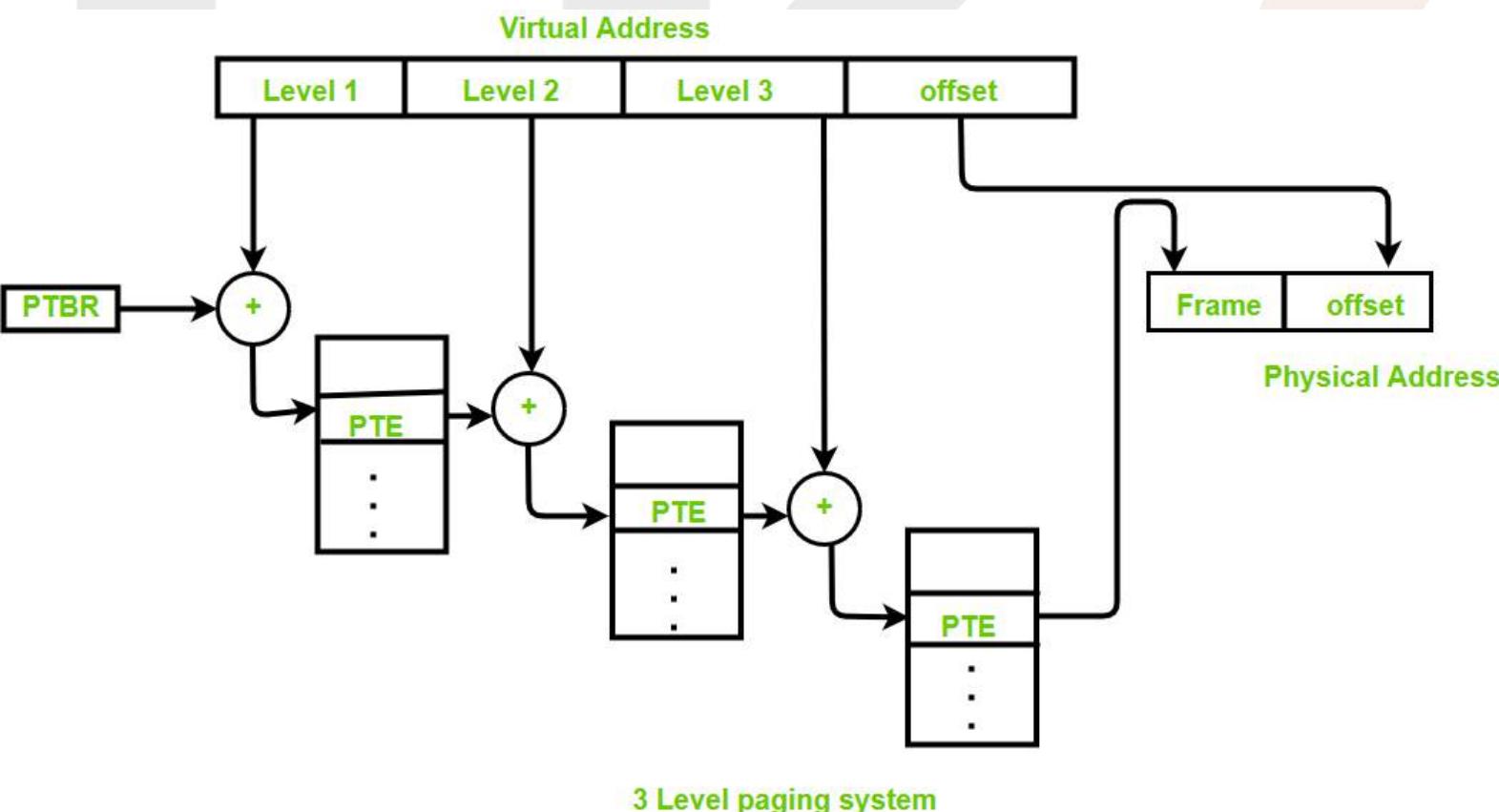
Q A processor uses 36-bit physical addresses and 32-bit virtual addresses, with a page frame size of 4 Kbytes. Each page table entry is of size 4 bytes. A three level page table is used for virtual to physical address translation, where the virtual address is used as follows • Bits 30-31 are used to index into the first level page table • Bits 21-29 are used to index into the second level page table • Bits 12-20 are used to index into the third level page table, and • Bits 0-11 are used as offset within the page. The number of bits required for addressing the next level page table (or page frame) in the page table entry of the first, second and third level page tables are respectively (GATE-2008) (2 Marks)

(A) 20, 20 and 20

(B) 24, 24 and 24

(C) 24, 24 and 20

(D) 25, 25 and 24



Le at ₹25/day **CLICK HERE** ↗

**Q** In a system with 32 bit virtual addresses and 1 KB page size, use of one-level page tables for virtual to physical address translation is not practical because of (GATE-2003) (1 Marks)

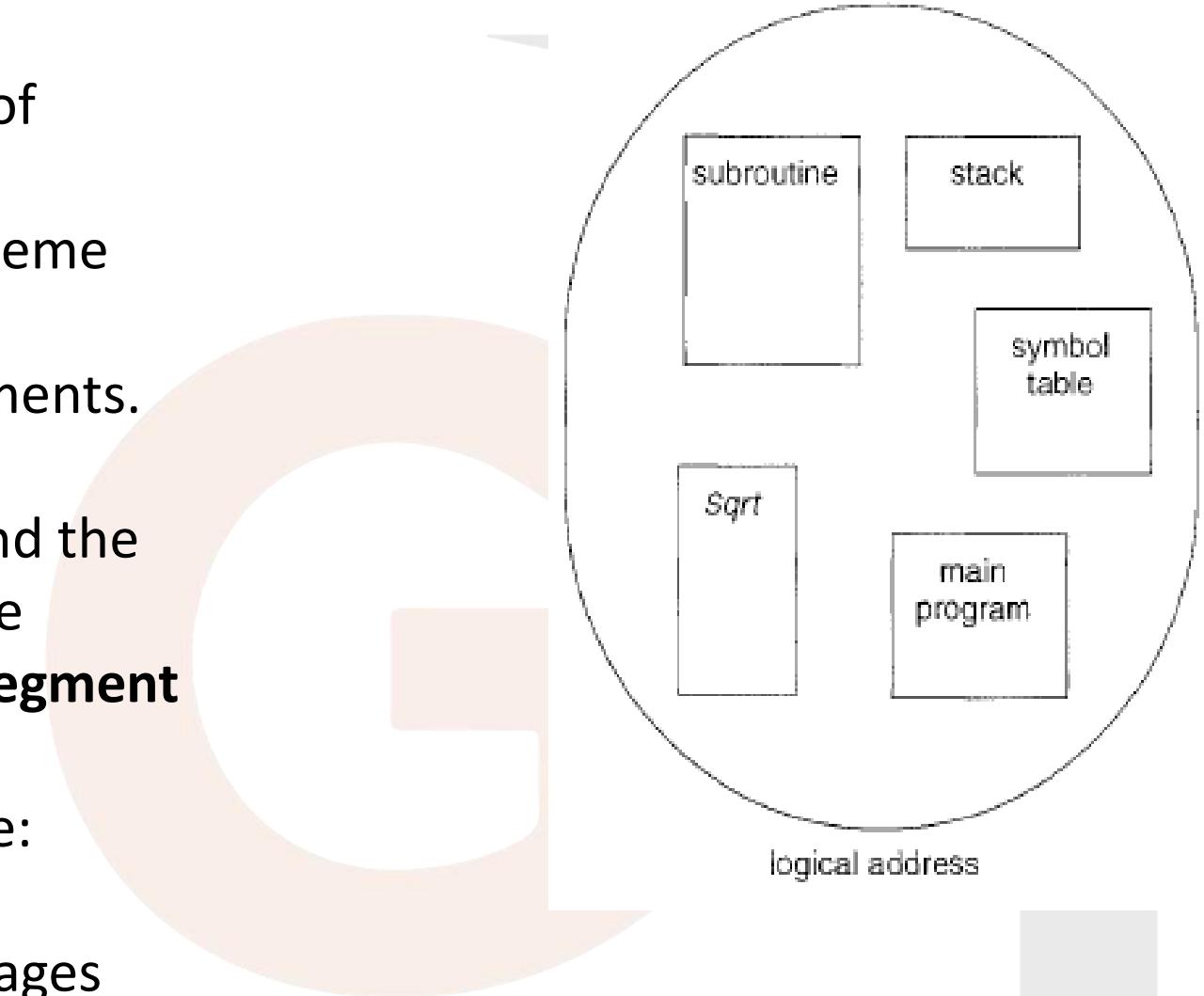
- (A) the large amount of internal fragmentation
- (B) the large amount of external fragmentation
- (C) the large memory overhead in maintaining page tables
- (D) the large computation overhead in the translation process

# Break

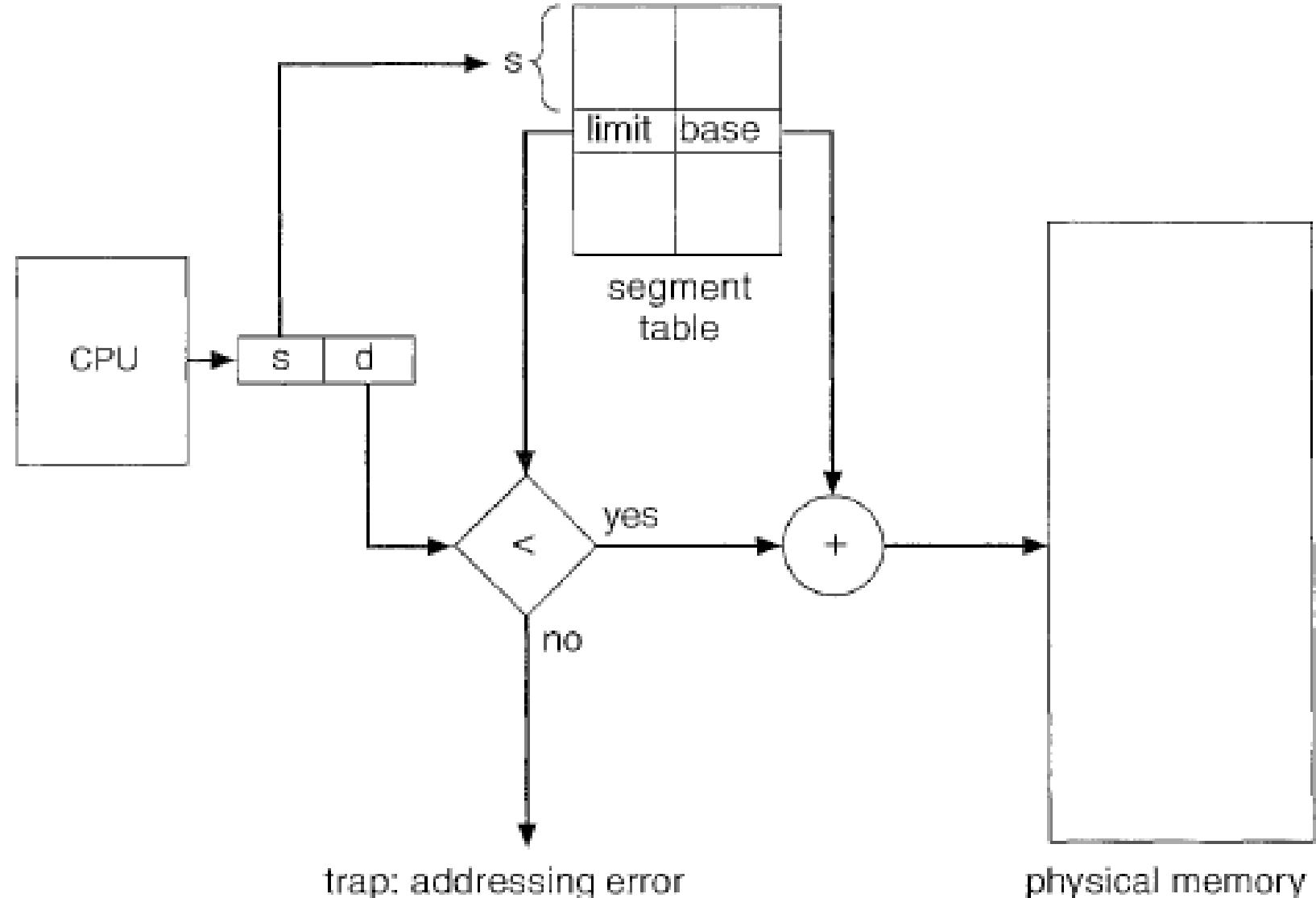
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Segmentation

- Paging is unable to separate the user's view of memory from the actual physical memory.
- Segmentation is a memory-management scheme that supports this user view of memory.
- A logical address space is a collection of segments.
- Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: **a segment name and an offset**.
- Thus, a logical address consists of a two tuple:
- **<segment-number, offset>**.
- Segments can be of variable lengths unlike pages and are stored in main memory.

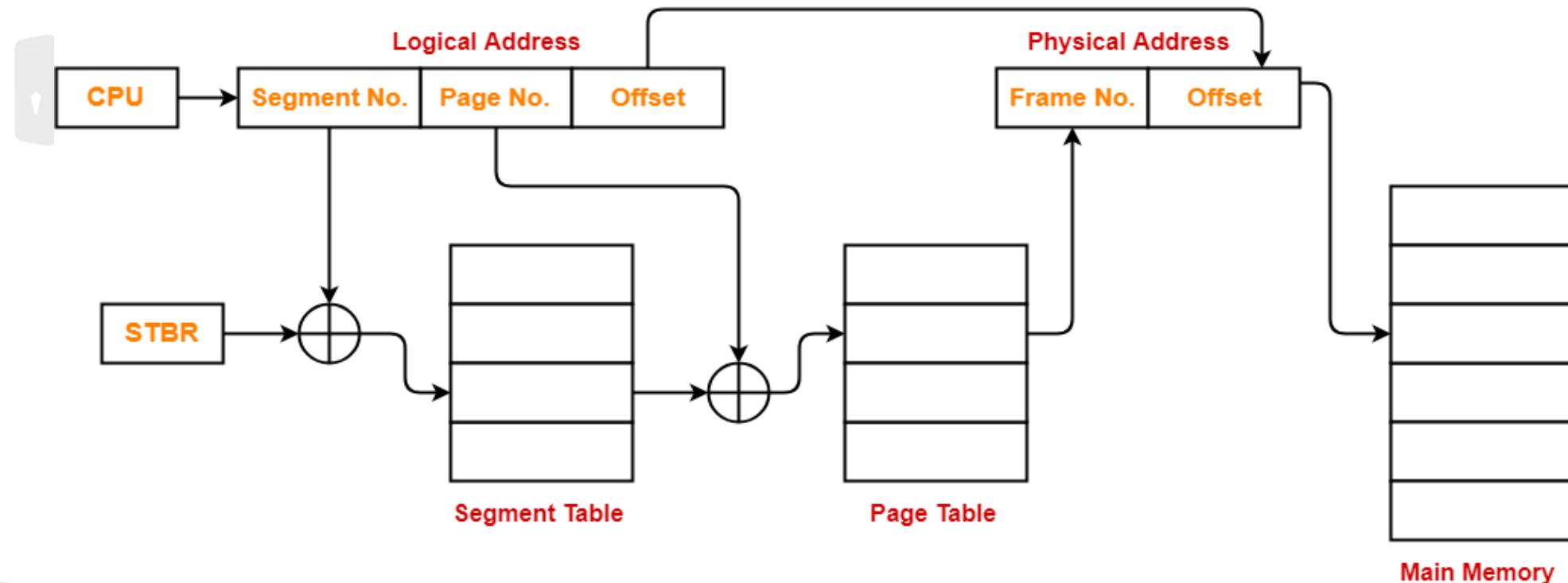


- **Segment Table:** Each entry in the segment table has a **segment base** and a **segment limit**. The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.
- The segment number is used as an index to the segment table.
- The offset  $d$  of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system.
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.
- Segmentation suffers from **External Fragmentation**.



## Segmentation with Paging

- Since segmentation also suffers from external fragmentation, **it is better to divide the segments into pages as the segments size increases.**
- In segmentation with paging scheme a process is divided into segments and further the segments are divided into pages.
- One can argue it is segmentation with paging is quite similar to multilevel paging, but actually it is better, because here when page table is divided the size of partition can be different (as actually the size of different chapters can be different).



To access an page content get **RO** time at **Zero day** [CLICK HERE](#) ↗

Translating Logical Address into Physical Address

- All properties of segmentation with paging is same as multilevel paging



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A computer uses 46-bit virtual address, 32-bit physical address, and a three-level paged page table organization. The page table base register stores the base address of the first-level table (T1), which occupies exactly one page. Each entry of T1 stores the base address of a page of the second-level table (T2). Each entry of T2 stores the base address of a page of the third-level table (T3). Each entry of T3 stores a page table entry (PTE). The PTE is 32 bits in size. The processor used in the computer has a 1 MB 16 way set associative virtually indexed physically tagged cache. The cache block size is 64 bytes.

What is the size of a page in KB in this computer? **(GATE-2013) (2 Marks)**

**(A) 2**

**(B) 4**

**(C) 8**

**(D) 16**

**Q.21** In the context of operating systems, which of the following statements is/are correct with **(Gate-2021) (1 Marks)**

respect to paging?

- (a) Paging incurs memory overheads.
- (b) Multi-level paging is necessary to support pages of different sizes.
- (c) Page size has no impact on internal fragmentation.
- (d) Paging helps solve the issue of external fragmentation.



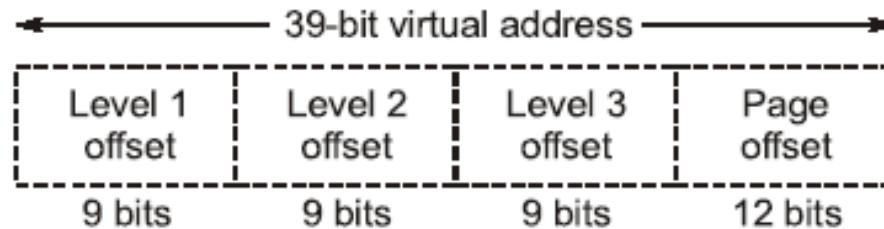
To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q.52

Consider a three-level page table to translate a 39-bit virtual address to a physical address as shown below:

**(GATE- 2021)**

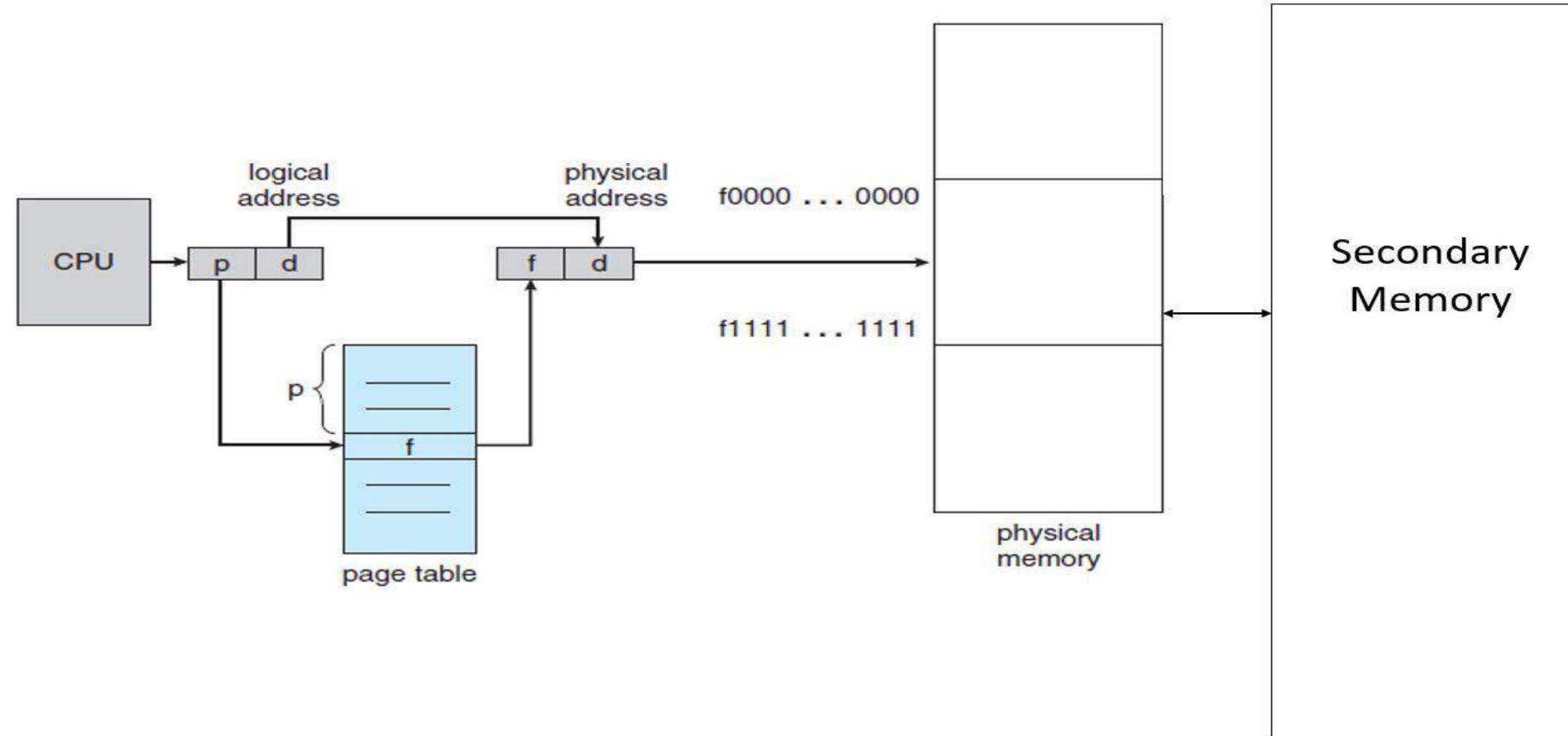


The page size is 4 KB ( $1 \text{ KB} = 2^{10} \text{ bytes}$ ) and page table entry size at every level is 8 bytes. A process  $P$  is currently using 2 GB ( $1 \text{ GB} = 2^{30} \text{ bytes}$ ) virtual memory which is mapped to 2 GB of physical memory. The minimum amount of memory required for the page table of  $P$  across all levels is \_\_\_\_\_ KB.

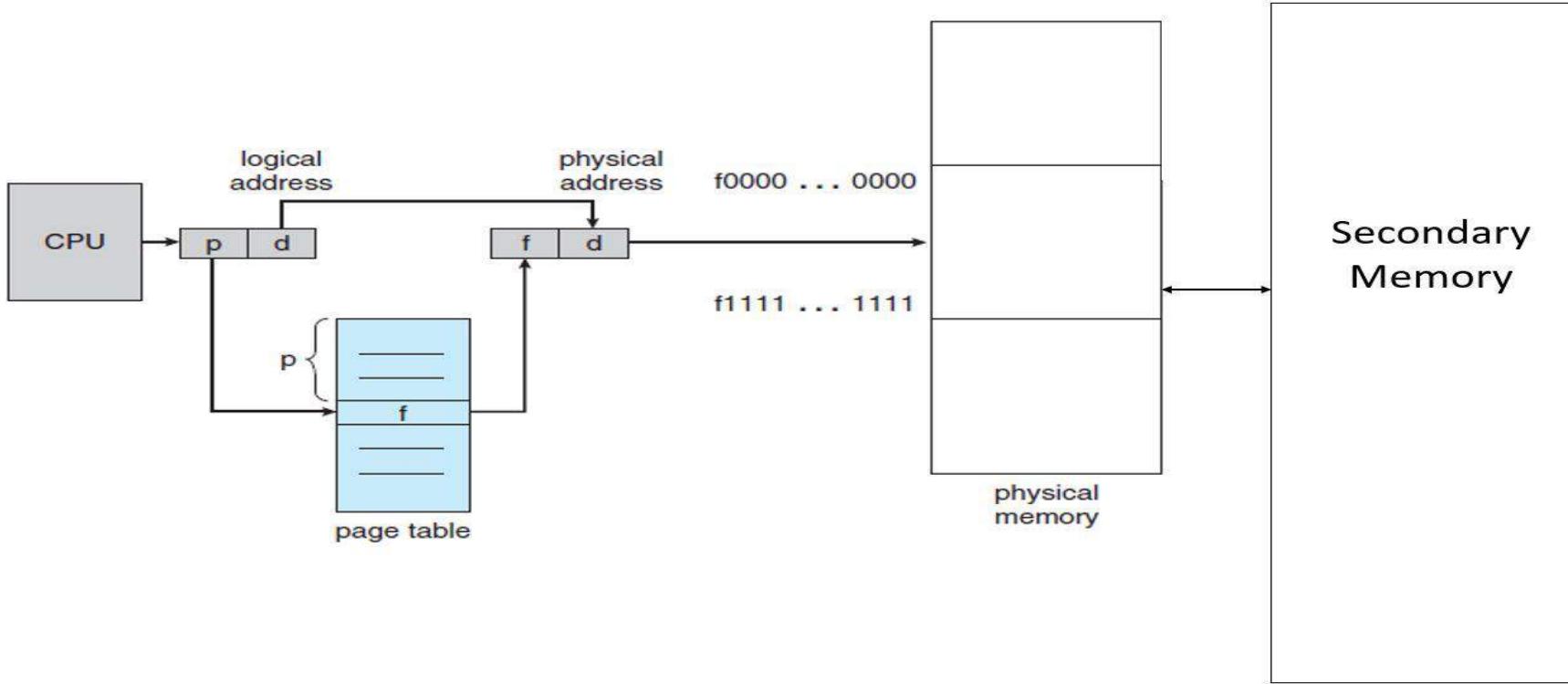


## Virtual Memory

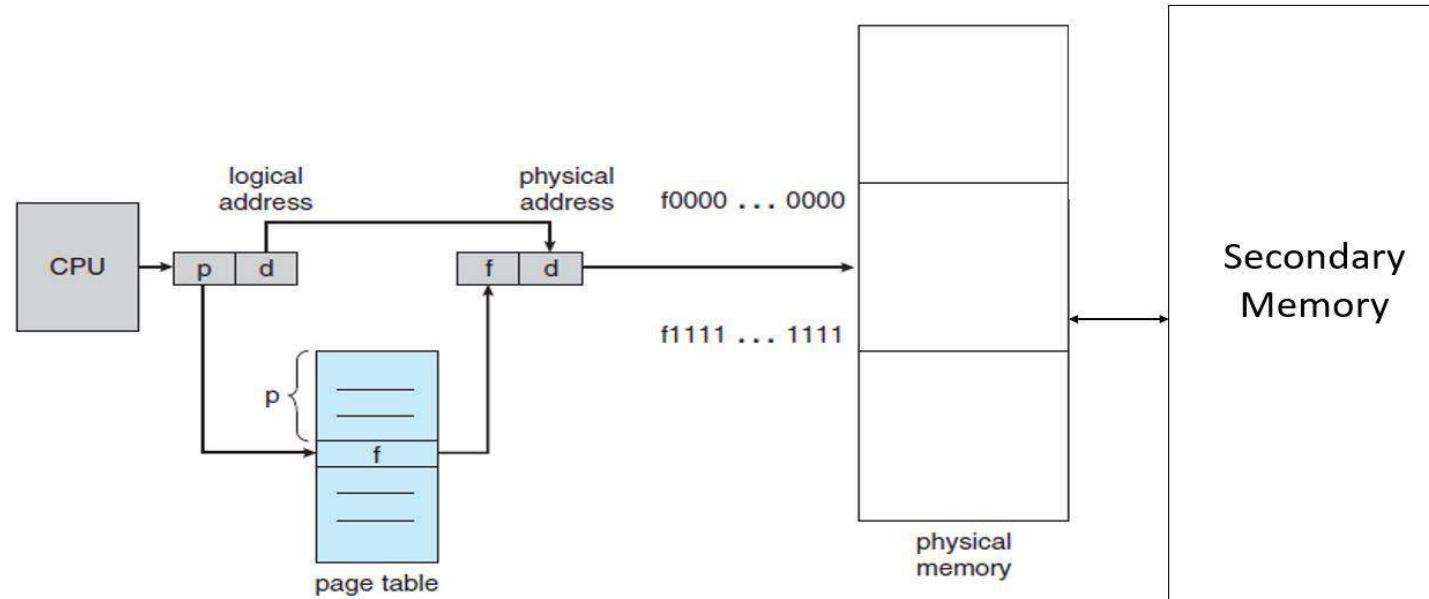
- One important goal in now-a-days computing environment is to keep many processes in memory simultaneously to follow multiprogramming, and use resources efficiently especially main memory.



- **Pure Demand Paging/Demand Paging:** We can start executing a process with no pages in memory. When the operating system sets the instruction pointer to the first instruction of the process, which is on a non-memory-resident page, the process immediately faults for the page.
- After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that it can execute with no more faults. This scheme is **Pure Demand Paging:** never bring a page into memory until it is required.



- The **Locality of Reference** helps Demand Paging in performing reasonably.
- A demand-paging system is extension to a paging system with swapping.
- But rather than swapping the entire process into memory, we use a **Lazy Swapper**. A lazy swapper never swaps a page into memory unless that page will be needed.
- We use the term **Pager** instead of **swapper** in demand paging.



To access a

ay

CLICK HERE ↗

**Q Which of the following is incorrect for virtual memory? (NET-JAN-2017)**

- a) Large programs can be written**
- b) More I/O is required**
- c) More addressable memory available**
- d) Faster and easy swapping of process**



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Page making process from main memory to disk is called (NET-DEC-2010)

(A) Interruption



(C) Swapping



(B) Termination



(D) None of the above

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q If the executing program size is greater than the existing RAM of a computer, it is still possible to execute the program if the OS supports:

**(NET-DEC-2008)**

**(A) multitasking**

**(B) virtual memory**



**(C) paging system**



**(D) none of the above**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Moving Process from main memory to disk is called: (NET-JUNE-2005)

(A) Caching

(B) Termination

(C) Swapping

(D) Interruption

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Advantage

- A program would no longer be constrained by the amount of physical memory that is available, Allows the execution of processes that are not completely in main memory, i.e. process can be larger than main memory.
- More programs could be run at the same time as use of main memory is less.
- Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.
- Virtual memory also allows processes to share files easily and to implement shared memory.

## Disadvantages

- Virtual memory is not easy to implement.
- It may substantially decrease performance if it is used carelessly (Thrashing)

Q Which of the following memory allocation scheme suffers from external fragmentation? (NET-DEC-2012)

(A) Segmentation

(B) Pure demand paging

(C) Swapping

(D) Paging

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q Which of the following statements is false? (GATE-2001) (1 Marks)**

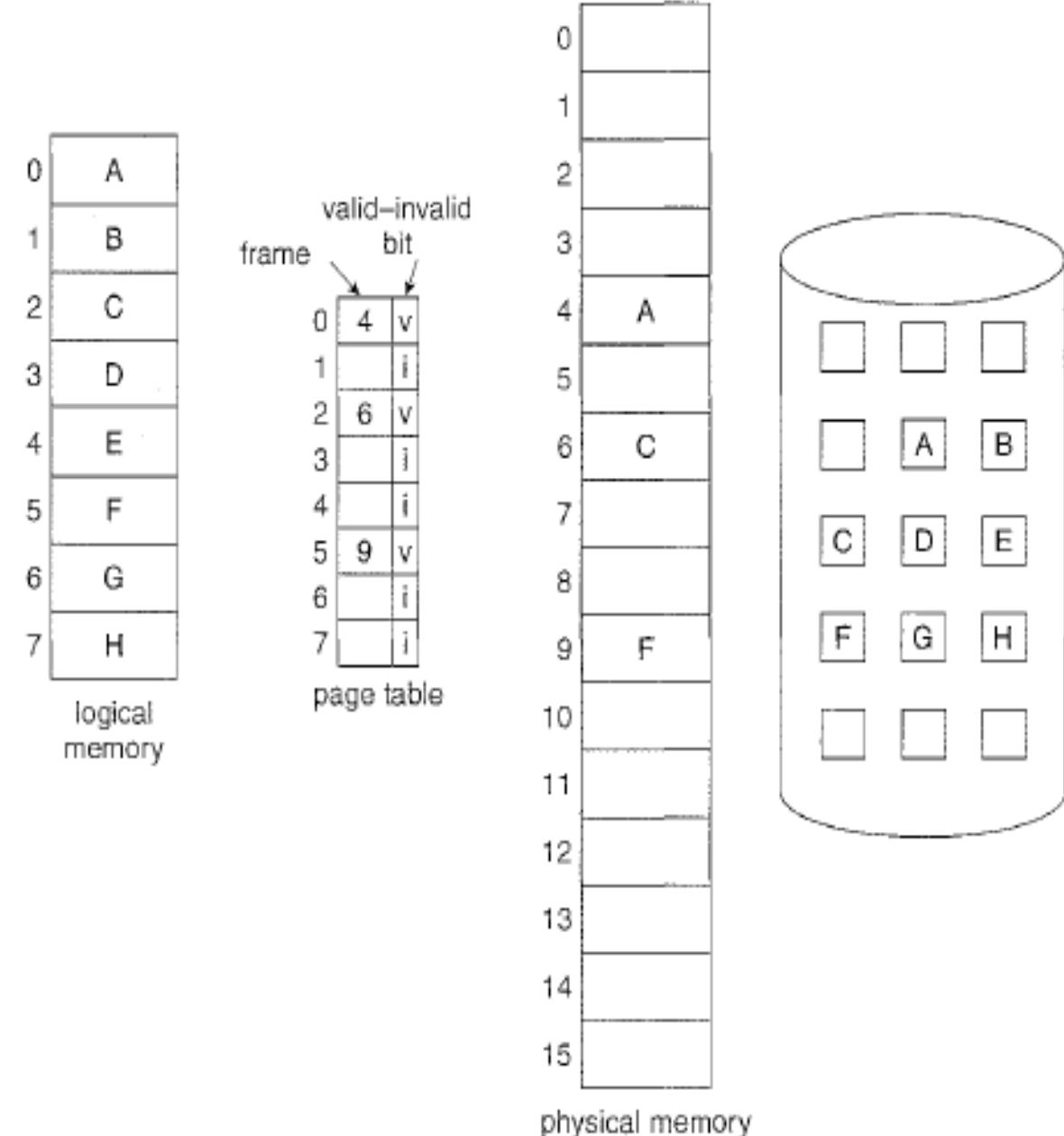
- (A) Virtual memory implements the translation of a program's address space into physical memory address space**
- (B) Virtual memory allows each program to exceed the size of the primary memory**
- (C) Virtual memory increases the degree of multiprogramming**
- (D) Virtual memory reduces the context switching overhead**

# Break

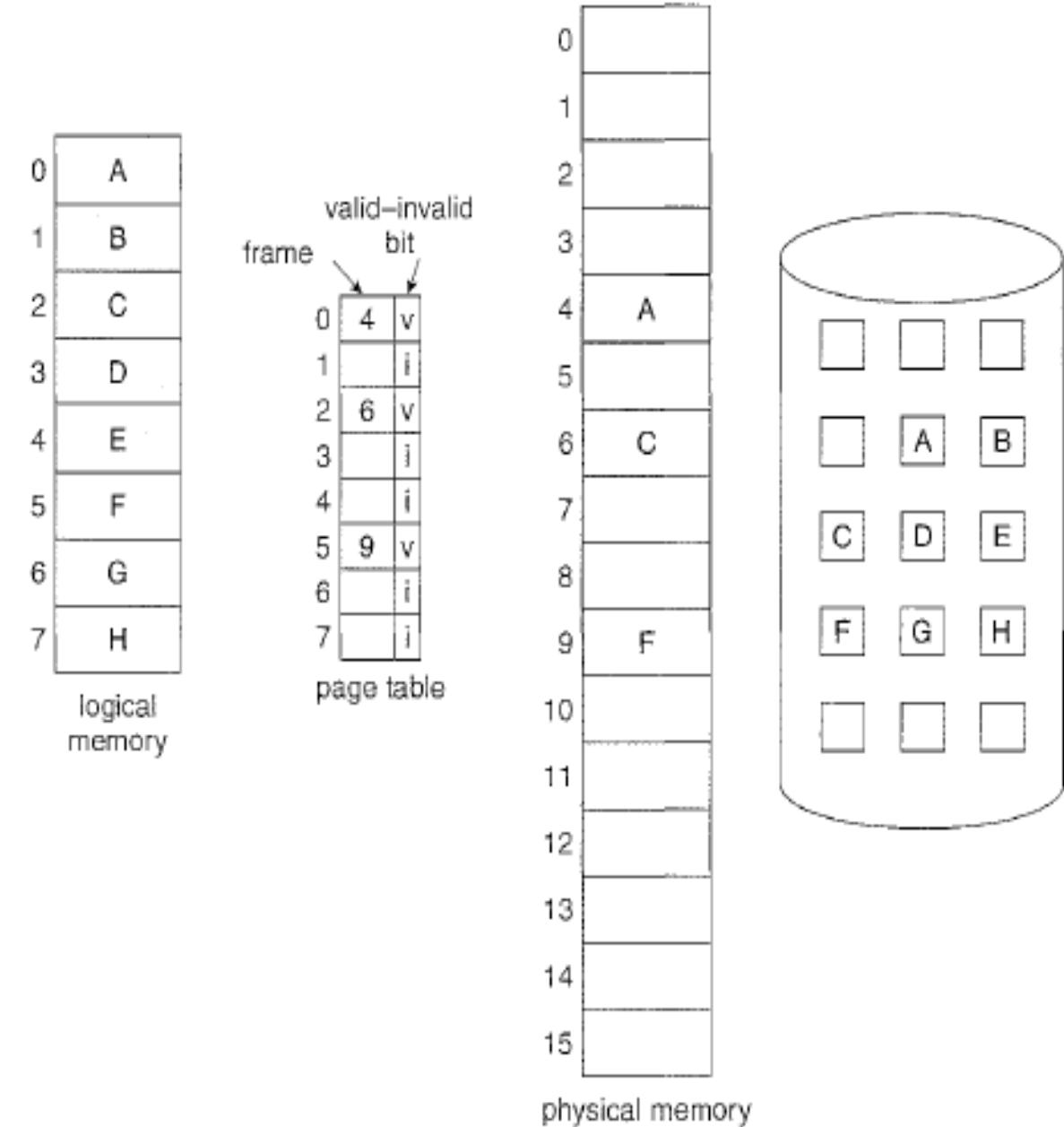
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Basic Concepts

- When a process is started execution, no page is loaded into main memory before demand so, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.
- Now we need some hardware to distinguish between which pages are in memory and which are not, so **valid -invalid bit scheme** can be used for it.



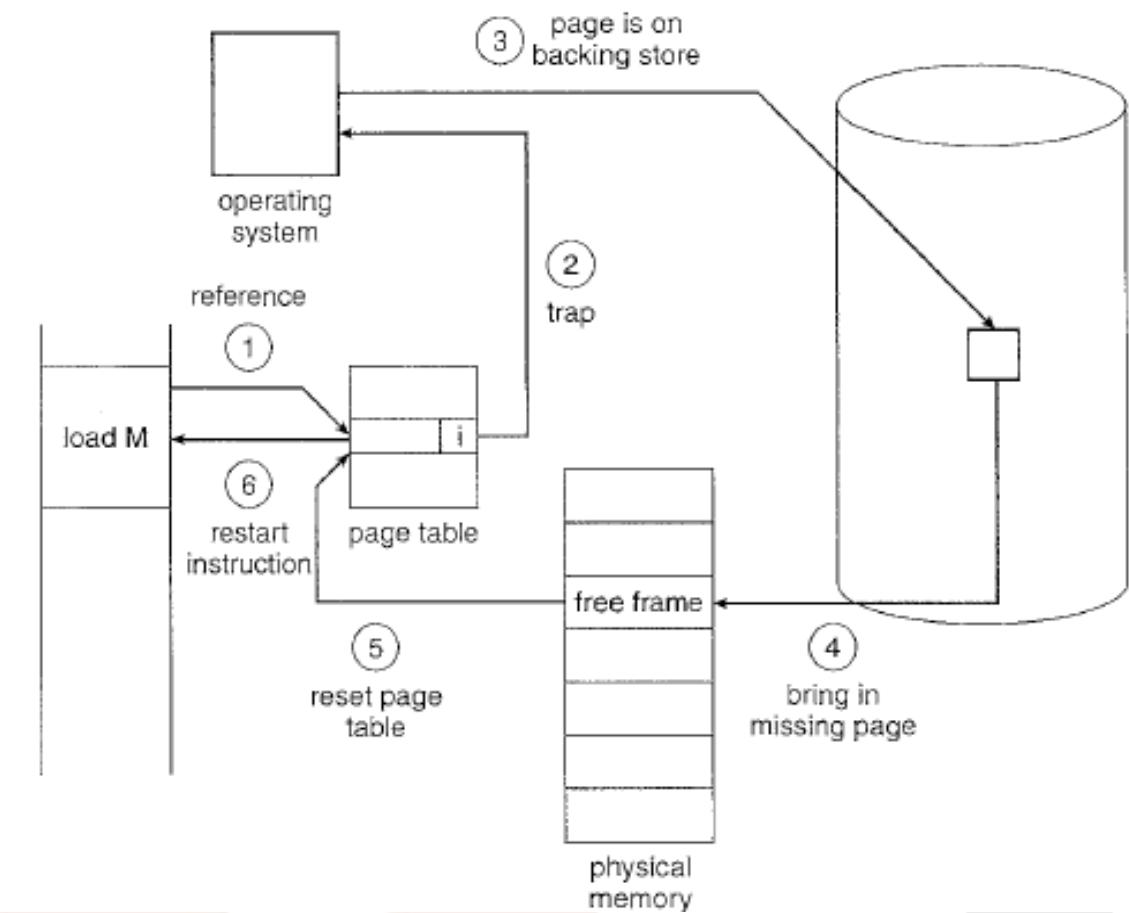
- This time, however, when this bit is set to "**valid**" the associated page is both legal and in memory. If the bit is set to "**invalid**" the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.
- The page-table entry for a page that is brought into memory is set as usual but the page-table entry for a page that is not currently in memory is either simply marked invalid.



**Page Fault:** - When a process tries to access a page that is not in main memory then a **Page Fault Occurs.**

### Steps to handle Page Fault

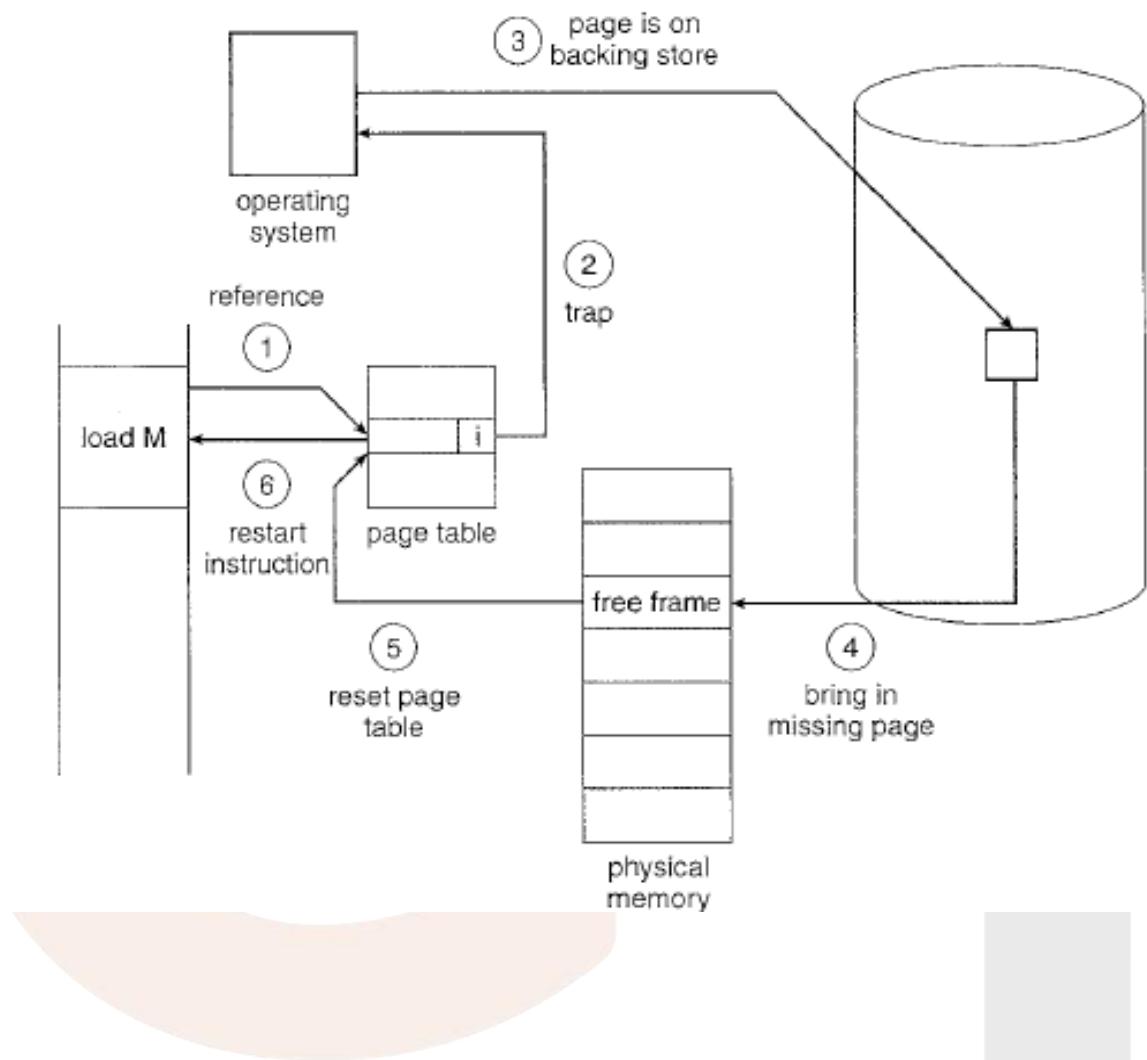
- If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
- We find a free frame (by taking one from the free-frame list, for example).
- We schedule a disk operation to read the desired page into the newly allocated frame.



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.
- A crucial requirement for demand paging is the ability to restart any instruction after a page fault. Because we save the state (registers, condition code, instruction counter) of the interrupted process when a page fault occurs, we must be able to restart the process in exactly the same place and state.



Q A page fault (NET-DEC-2009)

(A) is an error specific page.

(B) is an access to the page not currently in memory.

(C) occur when a page program occur in a page memory.

(D) page used in the previous page reference.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q A page fault ..... (NET-JUNE-2006)

(A) is an error in specific page

(B) is an access to the page not currently in main memory

(C) occurs when a page program accesses a page of memory

(D) is reference to the page which belongs to another program

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Performance of Demand Paging

- **Effective Access time for Demand Paging:**
  - $(1 - p) \times ma + p \times \text{page fault service time.}$
- Here,  $p$ : Page fault rate or probability of a page fault.
- $ma$  is memory access time.

**Q** In a paged memory, the page hit ratio is 0.40. The time required to access a page in secondary memory is equal to 120 ns. The time required to access a page in primary memory is 15 ns. The average time required to access a page is \_\_\_\_\_.

**(NET-JULY-2018)**

- (a) 105
- (b) 68
- (c) 75
- (d) 78

**Q** Consider a process executing on an operating system that uses demand paging. The average time for a memory access in the system is  $M$  units if the corresponding memory page is available in memory, and  $D$  units if the memory access causes a page fault. It has been experimental measured that the average time taken for a memory access in the process is  $X$  units. Which one of the following is the correct expression for the page fault rate experienced by the process? **(GATE-2018) (2 Marks)**

- (A)  $(D - M) / (X - M)$
- (B)  $(X - M) / (D - M)$
- (C)  $(D - X) / (D - M)$
- (D)  $(X - M) / (D - X)$

**Q** Suppose that the number of instructions executed between page fault is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Further, consider that a normal instruction takes one microsecond, but if a page fault occurs, it takes 2001 microseconds. If a program takes 60 sec to run, during which time it gets 15,000 page faults, how long would it take to run if twice as much memory were available? **(NET-DEC-2015)**

- A) 60 sec**
- b) 30 sec**
- c) 45 sec**
- d) 10 sec**

**Q** In a demand paging memory system, page table is held in registers. The time taken to service a page fault is 8 msec. if an empty frame is available or if the replaced page is not modified, and it takes 20 msec., if the replaced page is modified. What is the average access time to service a page fault assuming that the page to be replaced is modified 70% of the time? **(NET-DEC-2014)**

- (A)** 11.6 msec.
- (B)** 16.4 msec.
- (C)** 28 msec.
- (D)** 14 msec.

Q Virtual memory is (NET-JUNE-2011)

(A) related to virtual reality

(B) a form of ROM

(C) a form of RAM

(D) None of the above



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one-page fault is generated for every  $10^6$  memory accesses, what is the effective access time for the memory? **(GATE-2011) (1 Marks)**

- (A) 21ns      (B) 30ns      (C) 23ns      (D) 35ns

Q The memory allocation scheme subjected to “external” fragmentation is: (NET-JUNE-2006)

(A) Segmentation

(B) Swapping

(C) Demand paging

(D) Multiple contiguous fixed partitions

**Q** Suppose the time to service a page fault is on the average 10 milliseconds, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in average memory access time of? **(GATE-2000) (1 Marks).**

(A)1.9999 milliseconds

(B)1 millisecond

(C)9.999 microseconds

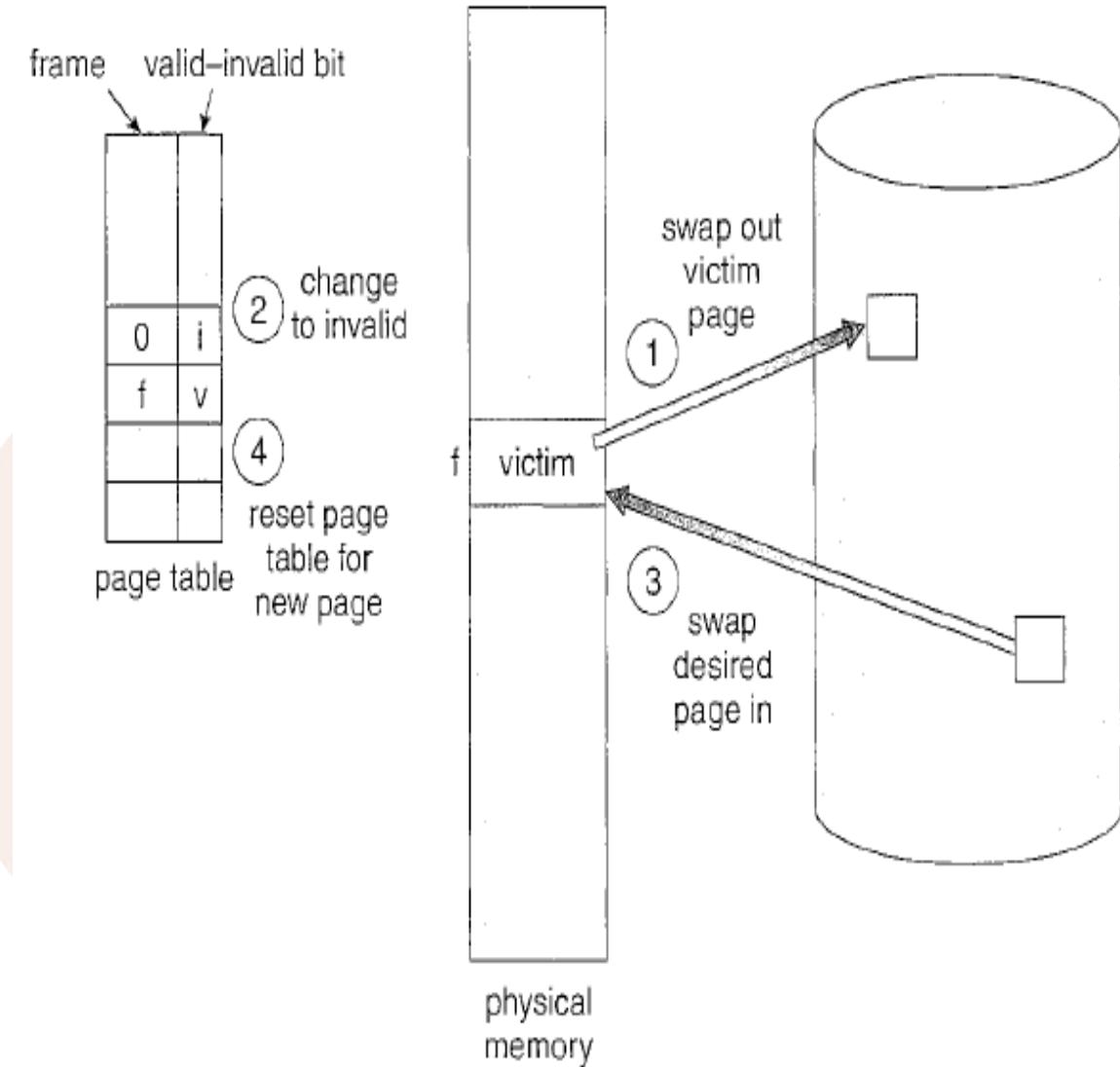
(D)1.9999 microseconds

# Break

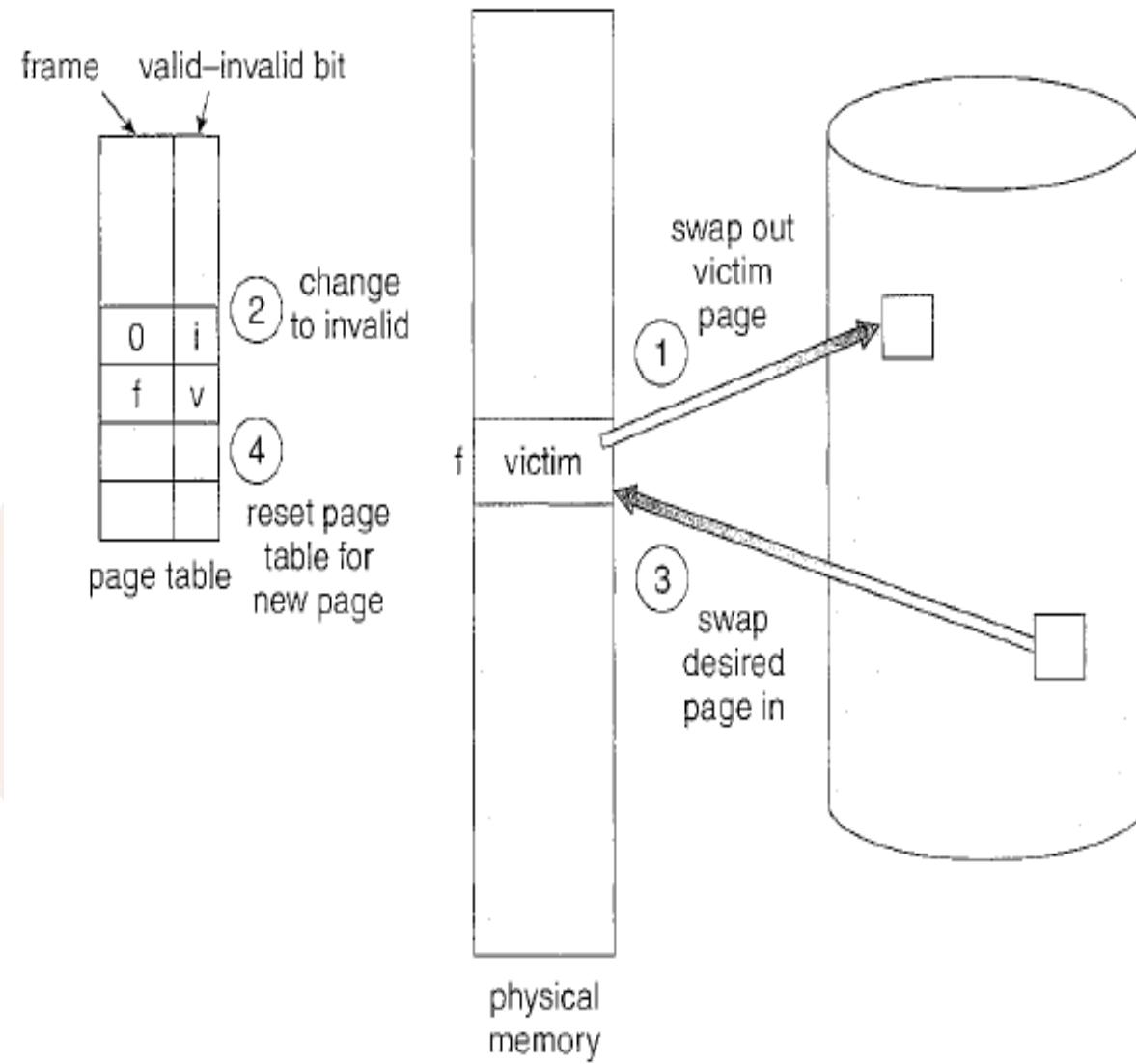
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Page Replacement

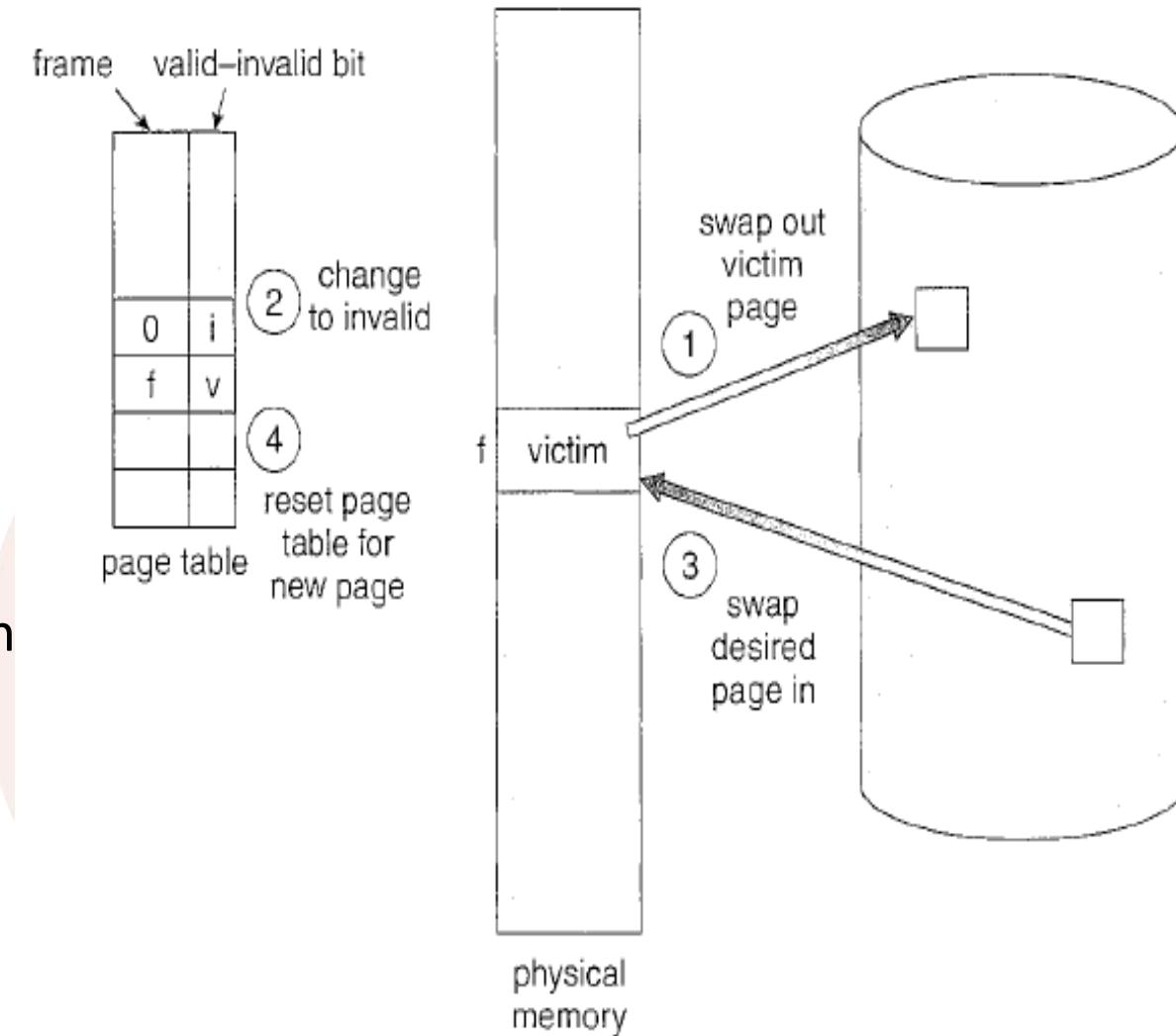
- With the increase in multiprogramming each process will get less amount of space in main memory so, the rate of page faults may rise.
- thus, to reduce the degree of multiprogramming the operating system swaps out processes from the memory freeing the frames and thus the process that requires to execute can now execute.



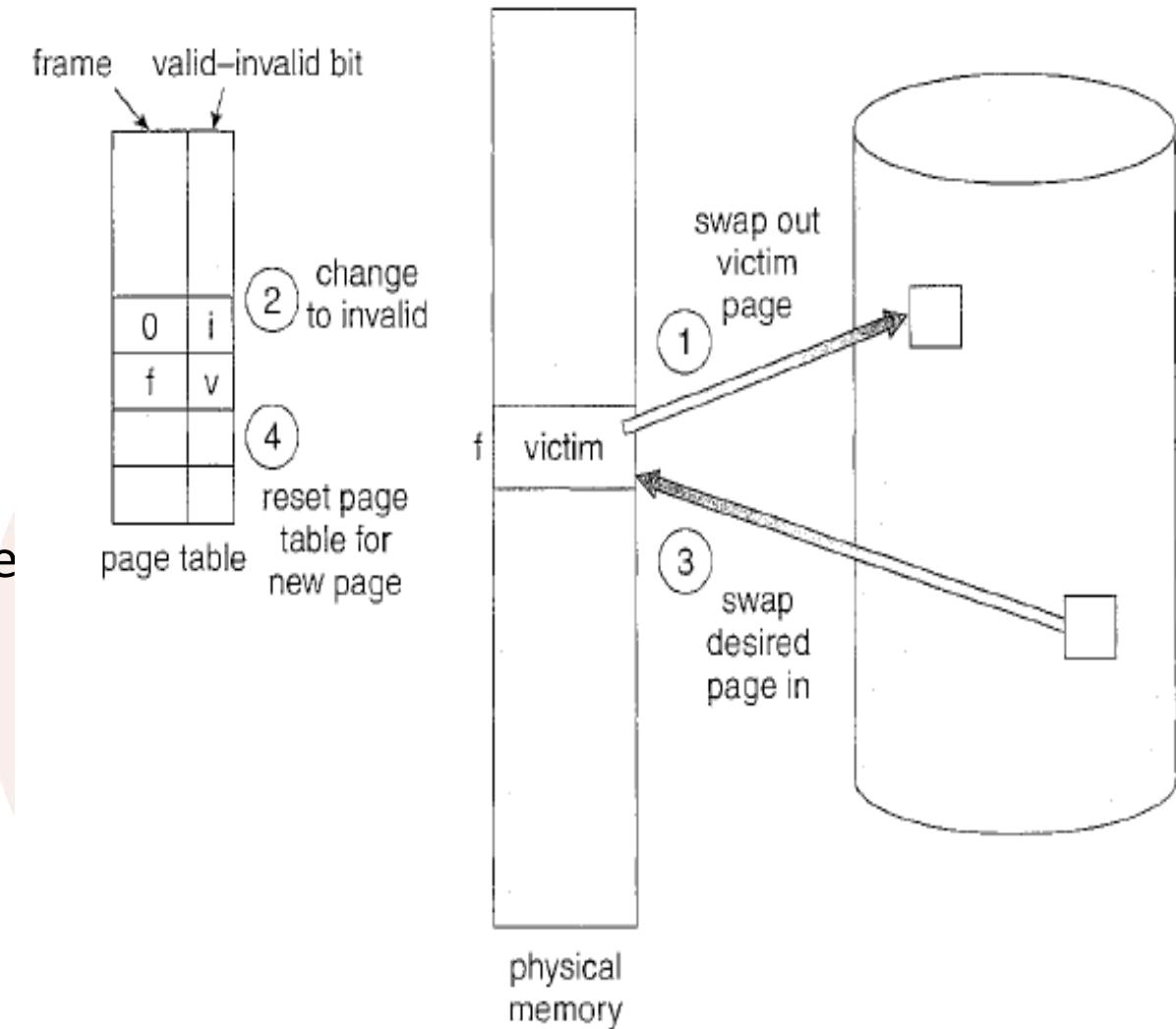
- If no frames are free, two-page transfers (one out and one in) are required. This situation effectively doubles the page-fault service time and increases the effective access time accordingly.
- We can reduce this overhead by using a **Modify bit or Dirty Bit**. When this scheme is used, each page or frame has a modify bit associated with it in the hardware.



- The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.
- **If the bit is set:** the page has been modified since it was read in from the disk. In this case, we must write the page to the disk.
- **If the modify bit is not set:** however, the page has not been modified since it was read into memory. In this case, we need not write the memory page to the disk: it is already there.



- Now, we must solve two major problems to implement demand paging: We must be able to develop a **frame allocation algorithm** and a **page replacement algorithm**.
- Frame Allocation will decide how much frames to allocate to a process.
- Page Replacement will decide which page to replace next.



Q Dirty bit is used to show the (NET-DEC-2018)

- a) Wrong Page
- b) Page with corrupted data
- c) Page with low frequency occurrence
- d) Page that is modified after being loaded into cache memory

**Q** A virtual memory has a page size of 1K words. There are eight pages and four blocks. The associative memory page table contains the following entries:

Which of the following list of virtual addresses (in decimal) will not cause any page fault if referenced by the CPU ? **(NET-DEC-2015)**

a) 1024, 3072, 4096, 6144

b) 1234, 4012, 5000, 6200

c) 1020, 3012, 6120, 8100

d) 2021, 4050, 5112, 7100

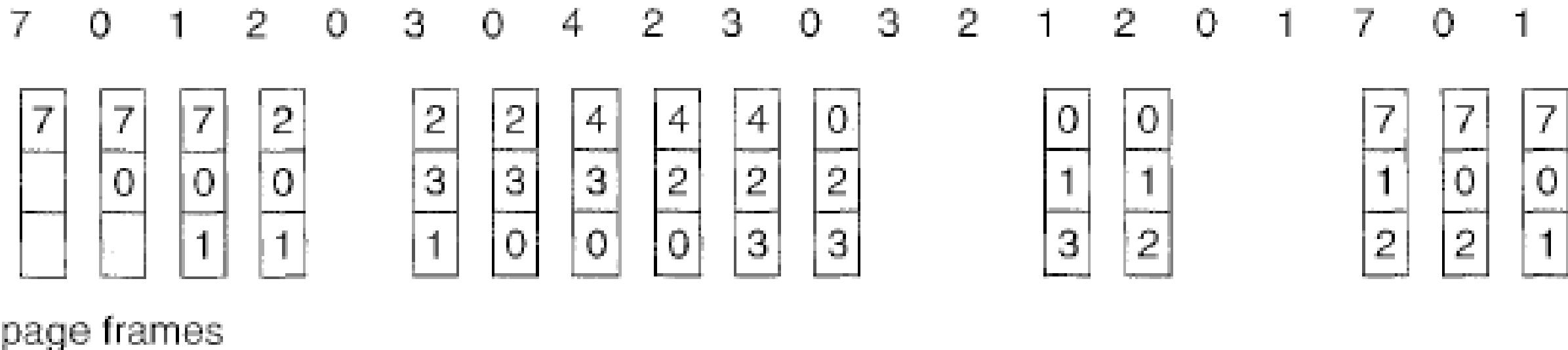
Page	Block
0	3
2	1
5	2
7	0

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Page Replacement Algorithms

- **First In First Out Page Replacement Algorithm:** - A FIFO replacement algorithm associates with each page, the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. i.e. the first page that came into the memory will be replaced first.



- In the above example the number of page fault is 15.

- The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good.
- **Belady's Anomaly:** for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.



**Q** Suppose for a process P, references to pages occur in order are 1, 2, 4, 5, 2, 1, 2, 4. Assume that the main memory can accommodate 3 pages and the main memory already has the pages 1 and 2 in the order 1-first, 2-second. At this moment, assume fifo page replacement algorithm is used then the number of page faults that occur to complete the execution of process P is **(NET-DEC-2018)**

**(A) 4**

**(B) 3**

**(C) 5**

**(D) 6**

**Q. In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases?**

**(GATE-2016) (1 Marks)**

**(a) LRU (Least Recently Used)**

**(b) OPT (Optimal Page Replacement)**

**(c) MRU (Most Recently Used)**

**(d) FIFO (First In First Out)**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Consider the reference string 0 1 2 3 0 1 4 0 1 2 3 4. If FIFO page replacement algorithm is used, then the number of page faults with three-page frames and four-page frames are \_\_\_\_\_ and \_\_\_\_\_ respectively. (NET-JUNE-2016)

- a) 10, 9
- b) 9, 9
- c) 10, 10
- d) 9,10

Q Consider the following page trace: 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5. Percentage of page fault that would occur if FIFO page replacement algorithm is used with number of frames for the JOB m = 4 will be **(NET-JUNE-2012)**

(A) 8

(B) 9

(C) 10

(D) 12



Q A system uses FIFO policy for page replacement. It has 4-page frames with no pages loaded to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur? (GATE-2010) (1 Marks)

- (A) 196
- (B) 192
- (C) 197
- (D) 195

Q In which one of the following page replacement policies, Belady's anomaly may occur? (GATE-2009) (1 Marks)

- (A) FIFO
- (B) Optimal
- (C) LRU
- (D) MRU

**Q** A virtual memory system uses First In First Out (FIFO) page replacement policy and allocates a fixed number of frames to a process. Consider the following statements: **(GATE-2007) (2 Marks)**

**P:** Increasing the number of page frames allocated to a process sometimes increases the page fault rate.

**Q:** Some programs do not exhibit locality of reference.

Which one of the following is TRUE?

- (A) Both P and Q are true, and Q is the reason for P
- (B) Both P and Q are true, but Q is not the reason for P.
- (C) P is false, but Q is true
- (D) Both P and Q are false.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q Which page replacement policy suffers from Belady's anomaly? (NET-JUNE-2007)

(A) LRU

(B) LFU

(C) FIFO

(D) OPTIMAL



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A program has five virtual pages, numbered from 0 to 4. If the pages are referenced in the order 012301401234, with three-page frames, the total number of page faults with FIFO will be equal to: (NET-DEC-2007)

(A) 0

(B) 4

(c) 6

(D) 9

**Q** Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will **(GATE-2001) (1 Marks)**

- (A)** always decrease the number of page faults
- (B)** always increase the number of page faults
- (C)** sometimes increase the number of page faults
- (D)** never affect the number of page faults

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Optimal Page Replacement Algorithm

- Replace the page that will not be used for the longest period of time.
- It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

reference string

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



page frames

- The number of page faults occurred: 9

- Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.
- It is mainly used for comparison studies.



Q Assume that there are 3 page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is \_\_\_\_\_. (GATE-2014) (2 Marks)



**Q** A process has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, 1. If optimal page replacement policy is used, how many page faults occur for the above reference string?

**(GATE-2007) (2 Marks)**

**(A) 7**

**(B) 8**

**(C) 9**

**(D) 10**

**Q** The optimal page replacement algorithm will select the page that **(GATE-2002) (1 Marks)**

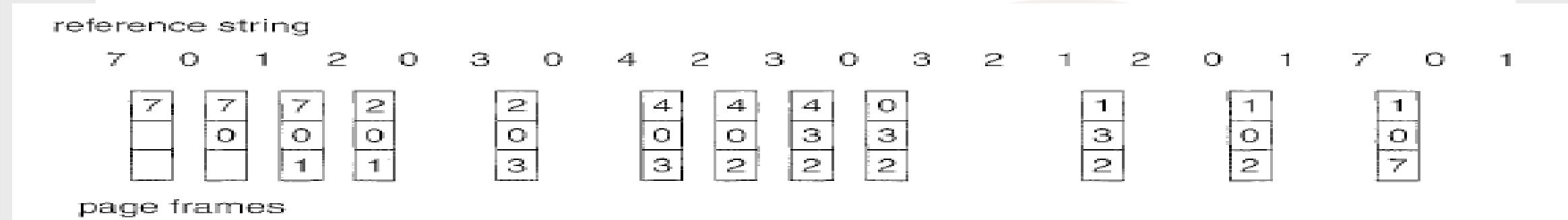
- (A)** Has not been used for the longest time in the past.
- (B)** Will not be used for the longest time in the future.
- (C)** Has been used least number of times.
- (D)** Has been used most number of times.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Least Recently Used (LRU) Page Replacement Algorithm

- Replace the page that has not been used for the longest period of time.
- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.



- LRU gives us 12-page faults.
- LRU is much better than FIFO replacement.
- The LRU policy is often used as a page-replacement algorithm and is considered to be good.
- LRU also does not suffer from Belady's Anomaly.

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

- The major problem is how to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance.
- LRU and OPT belong to a class of page-replacement algorithms, called **stack algorithms** and can never exhibit Belady's anomaly.
- A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for  $n$  frames is always a subset of the set of pages that would be in memory with  $n + 1$  frames.

Q Consider a virtual page reference string 1, 2, 3, 2, 4, 2, 5, 2, 3, 4. Suppose LRU page replacement algorithm is implemented with 3 page frames in main memory. Then the number of page faults are \_\_\_\_\_.

(NET-JULY-2018)

(a) 5

(b) 7

(c) 9

(d) 10

Q Consider a virtual page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Suppose a demand paged virtual memory system running on a computer system such that the main memory has 3 page frames. Then \_\_\_\_\_ page replacement algorithm has minimum number of page faults. (NET-NOV-2017)

- A) FIFO
- b) LIFO
- c) LRU
- d) Optimal

**Q** Recall that Belady's anomaly is that the page-fault rate may increase as the number of allocated frames increases. Now, consider the following statements:

**S<sub>1</sub>:** Random page replacement algorithm (where a page chosen at random is replaced) suffers from Belady's anomaly

**S<sub>2</sub>:** LRU page replacement algorithm suffers from Belady's anomaly

Which of the following is CORRECT? (GATE-2017) (1 Marks)

(a) S<sub>1</sub> is true, S<sub>2</sub> is true

(b) S<sub>1</sub> is true, S<sub>2</sub> is false

(c) S<sub>1</sub> is false, S<sub>2</sub> is true

(d) S<sub>1</sub> is false, S<sub>2</sub> is false

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the following page reference string : 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6 Which of the following options, gives the correct number of page faults related to LRU, FIFO, and optimal page replacement algorithms respectively, assuming 05 page frames and all frames are initially empty ? **(NET-AUG-2016)**

- a) 10, 14, 8
- b) 8, 10, 7
- c) 7, 10, 8
- d) 7, 10, 7

**Q** Suppose that the virtual Address space has eight pages and physical memory with four page frames. If LRU page replacement algorithm is used, \_\_\_\_\_ number of page faults occur with the reference string.

0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1 **(NET-AUG-2016)**

**(A)** 13

**(B)** 12

**(C)** 11

**(D)** 10

**Q** Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. Which one of the following is true with respect to page replacement policies First-In-First Out (FIFO) and Least Recently Used (LRU)? **(GATE-2015) (2 Marks)**

- (A) Both incur the same number of page faults
- (B) FIFO incurs 2 more-page faults than LRU
- (C) LRU incurs 2 more-page faults than FIFO
- (D) FIFO incurs 1 more page faults than LRU



Q A LRU page replacement is used with four page frames and eight pages. How many page faults will occur with the reference string 0172327103 if the four frames are initially empty? **(NET-JUNE-2015)**

- a) 6
- b) 7
- c) 8
- d) 5

**Q** A system uses 3-page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below? 4, 7, 6, 1, 7, 6, 1, 2, 7, 2 **(GATE-2014) (2 Marks)**

**Q** Consider a program that consists of 8 pages (from 0 to 7) and we have 4 page frames in the physical memory for the pages. The page reference string is : 1 2 3 2 5 6 3 4 6 3 7 3 1 5 3 6 3 4 2 4 3 4 5 1. The number of page faults in LRU and optimal page replacement algorithms are respectively (without including initial page faults to fill available page frames with pages) : **(NET-JUNE-2014)**

(A) 9 and 6

(B) 10 and 7

(c) 9 and 7

(D) 10 and 6

**Q** A job has four pages A, B, C, D and the main memory has two page frames only. The job needs to process its pages in following order : ABACABDBACD. Assuming that a page interrupt occurs when a new page is brought in the main memory, irrespective of whether the page is swapped out or not. The number of page interrupts in FIFO and LRU page replacement algorithms are

**(NET-JUNE-2013)**

**(A) 9 and 7**

**(B) 7 and 6**

**(C) 9 and 8**

**(D) 8 and 6**

**Q** Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then **(GATE-2012) (2 Marks)**

(A) OPTIMAL < LRU < FIFO  
(C) OPTIMAL = LRU

(B) OPTIMAL < FIFO < LRU  
(D) OPTIMAL = FIFO

Q A process, has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1,2,1,3,7,4,5,6,3,1 Least Recently Used (LRU) page replacement policy is a practical approximation to optimal page replacement. For the above reference string, how many more page faults occur with LRU than with the optimal page replacement policy? (GATE-2007) (2 Marks)

(A) 0

(B) 1

(C) 2

(D) 3

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Solution: Two implementations are feasible:

- **Counters.** In the simplest case, we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.
- The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock registers are copied to the time-of-use field in the page-table entry for that page.
- In this way, we always have the "time" of the last reference to each page. We replace the page with the smallest time value.

- **Stack:** Another approach to implementing LRU replacement is to keep a stack of page numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top.
- In this way, the most recently used page is always at the top of the stack and the least recently used page is always at the bottom.

## Counting-Based Page Replacement

- We keep a counter of the number of references that have been made to each page and develop the following two schemes.
- The **least frequently used (LFU)** page-replacement algorithm requires that the page with the smallest count be replaced.
- The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

- One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.
- **The most frequently used (MFU)** page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- Neither MFU nor LFU replacement is common. The implementation of these algorithms is expensive, and they do not approximate OPT replacement well.

**Q** Consider a computer system with ten physical page frames. The system is provided with an access sequence  $(a_1, a_2, \dots, a_{20}, a_1, a_2, \dots, a_{20})$ , where each  $a_i$  is a distinct virtual page number. The difference in the number of page faults between the last-in-first-out page replacement policy and the optimal page replacement policy is \_\_\_\_\_. (GATE-2016) (1 Marks)

**Q** Consider a main memory with 3 page frames for the following page reference string : 5, 4, 3, 2, 1, 4, 3, 5, 4, 3, 4, 1, 4. Assuming that the execution of process is initiated after loading page 5 in memory, the number of page faults in FIFO and second chance replacement respectively are  
**(NET-SEP-2013)**

**(A)** 8 and 9

**(B)** 10 and 11

**(C)** 7 and 9

**(D)** 9 and 8

**Q** A computer has twenty physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, ..., 100 in that order, and repeats the access sequence **THRIC**E. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacement policy for this program? **(GATE-2014) (2 Marks)**

(A) Least-recently-used  
(C) Last-in-first-out

(B) First-in-first-out  
(D) Most-recently-used

**Q Increasing the RAM of a computer typically improves performance because (GATE-2005) (1 Marks) (ISRO-2015)**

**(A) Virtual memory increases**

**(B) Larger RAMs are faster**

**(C) Fewer page faults occur**

**(D) Fewer segmentation faults occur**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Frame Allocation Algorithms

- Minimum Number of frames to be allocated to each process depends on Instruction Set Architecture, and the maximum number of allocation of frames depends on the size of the process.
- Equal Allocation: Frames will be equally allocated to each process. Ex: if we have 30 frames and 3 processes, each will get 10 regardless of their size.

- **Weighted / Proportional Allocation:** Frames will be allocated according to the weights of the process. The allocation of frames per process:  $a(i) = s(i)/S \times m$ .
- Where,  $a(i)$  is the number of allocated frames,  $s(i)$  is the process and  $S$  is the size of the virtual memory in general,  $S = \sum s(i)$ , and  $m$  is the total number of frames available.
- Example: If we have 3 processes of size 20 k, 30k and 50 k then 30 frames will be allocated as:
- P1 gets:  $20/100 \times 30 = 6$  frames, similarly P2 and P3 will get 9 and 15 frames respectively.

## Some points to remember

- If the multiprogramming level is increased, each process will lose some frames to provide the memory needed for the new process.
- Conversely, if the multiprogramming level decreases, the frames that were allocated to the departed process can be spread over the remaining processes.

## Global vs Local Frame Allocation

With multiple processes competing for frames, we can classify page-replacement algorithms into two broad categories:

- **Global Replacement**
- **Local Replacement**



**Global Replacement:** Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another.

- With global replacement, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it.
- One problem with a global replacement algorithm is that a process cannot control its own page-fault rate. As The set of pages in memory for a process depends not only on the paging behaviour of that process but also on the paging behaviour of other processes.
- Global replacement generally results in greater system throughput and is therefore the more common method.

**Local Replacement:** Local replacement requires that each process select from only its own set of allocated frames.

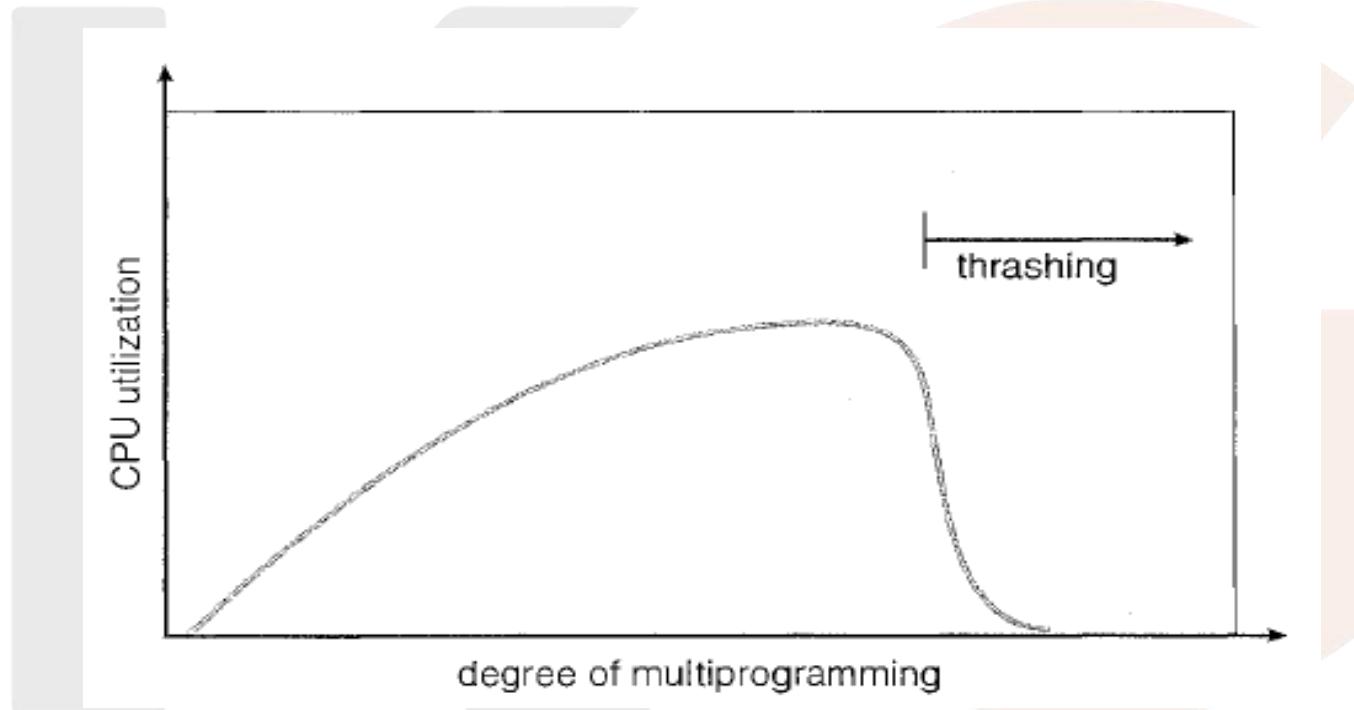
- With a local replacement strategy, the number of frames allocated to a process does not change.
- Under local replacement, the set of pages in memory for a process is affected by the paging behaviour of only that process.
- Local replacement might hinder a process, however, by not making available to it other, less used pages of memory.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Thrashing

- A process is thrashing if it is spending more time paging than executing. High paging activity is called Thrashing.



## Causes of Thrashing

- **High Degree of Multiprogramming:** If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system.
- A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong. Now if that a process enters a new phase in its execution and needs more frames. It will start faulting and taking frames away from other processes.
- These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

- The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result.
- The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more.
- 
- Thrashing has occurred, and system throughput plunges. The page fault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging.

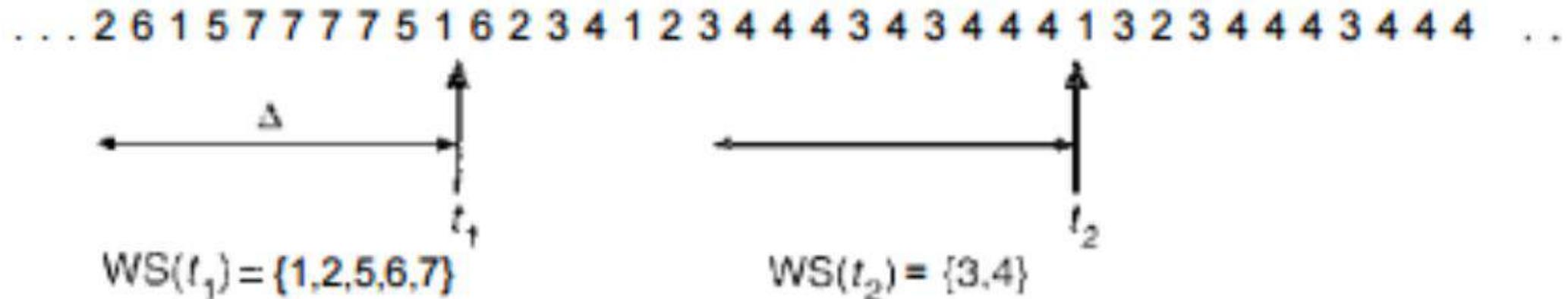
## Solution

- With local replacement, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well.
- To prevent thrashing, we must provide a process with as many frames as it needs.
- We decide how to allocate the number of frames to a process by **the working-set strategy**.

## The Working Set Strategy

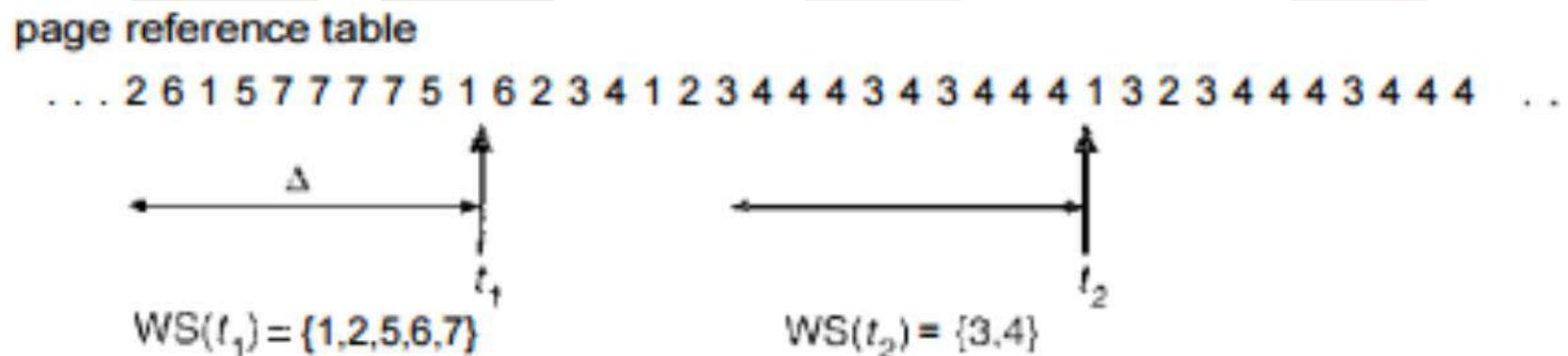
- This approach defines the **locality model** of process execution.
- The locality model states that, as a process executes, it moves from locality to locality. A locality is a set of pages that are actively used together
- A program is generally composed of several different localities, which may overlap.

page reference table



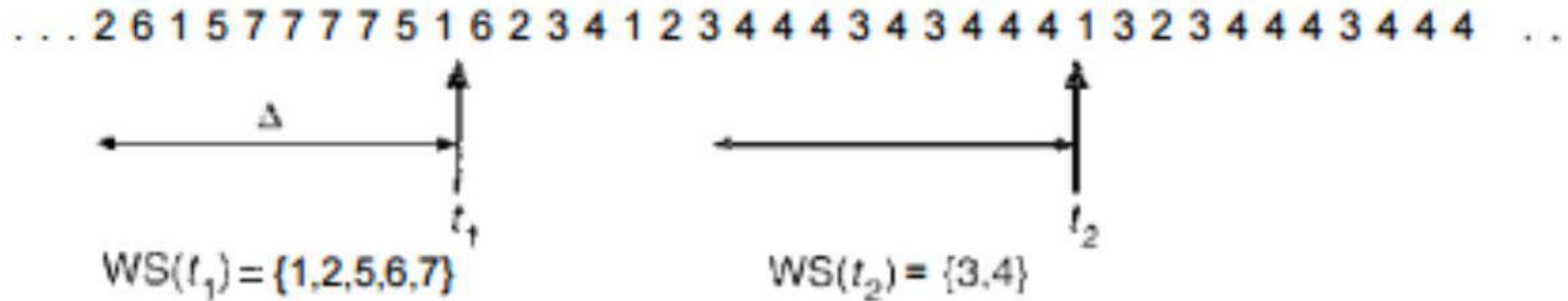
## Working Set Model

- This model uses a parameter  $\Delta$ , to define the **working set window**.
- The set of pages in the most recent  $\Delta$  page references is the working set.
- If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set  $\Delta$  time units after its last reference.
- The working set is an approximation of the program's locality.



- The accuracy of the working set depends on the selection of  $\Delta$ . If  $\Delta$  is too small, it will not encompass the entire locality; if  $\Delta$  is too large, it may overlap several localities.
- If we can calculate the Working-Set Size (WSS) for each process then:  $D = \sum WSS_i$ , where  $D$  is the total demand for frames.
- If the total demand is greater than the total number of available frames ( $D > m$ ), thrashing will occur, because some processes will not have enough frames.

**page reference table**



# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Page-Fault Frequency

- It is a much more direct strategy to control **Page-Fault Frequency**.
- Thrashing has a high page-fault rate and thus we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.

- We establish upper and lower bounds on the desired page-fault rate.
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame; if the page-fault rate falls below the lower limit, we remove a frame from the process. **With the working-set strategy, we may have to suspend a process. If the page-fault rate increases and no free frames are available, we must select some process and suspend it.**

Q Working set model is used in memory management to implement the concept of  
**(NET-JUNE-2013)**

**(A) Swapping**

**(B) Principal of Locality**



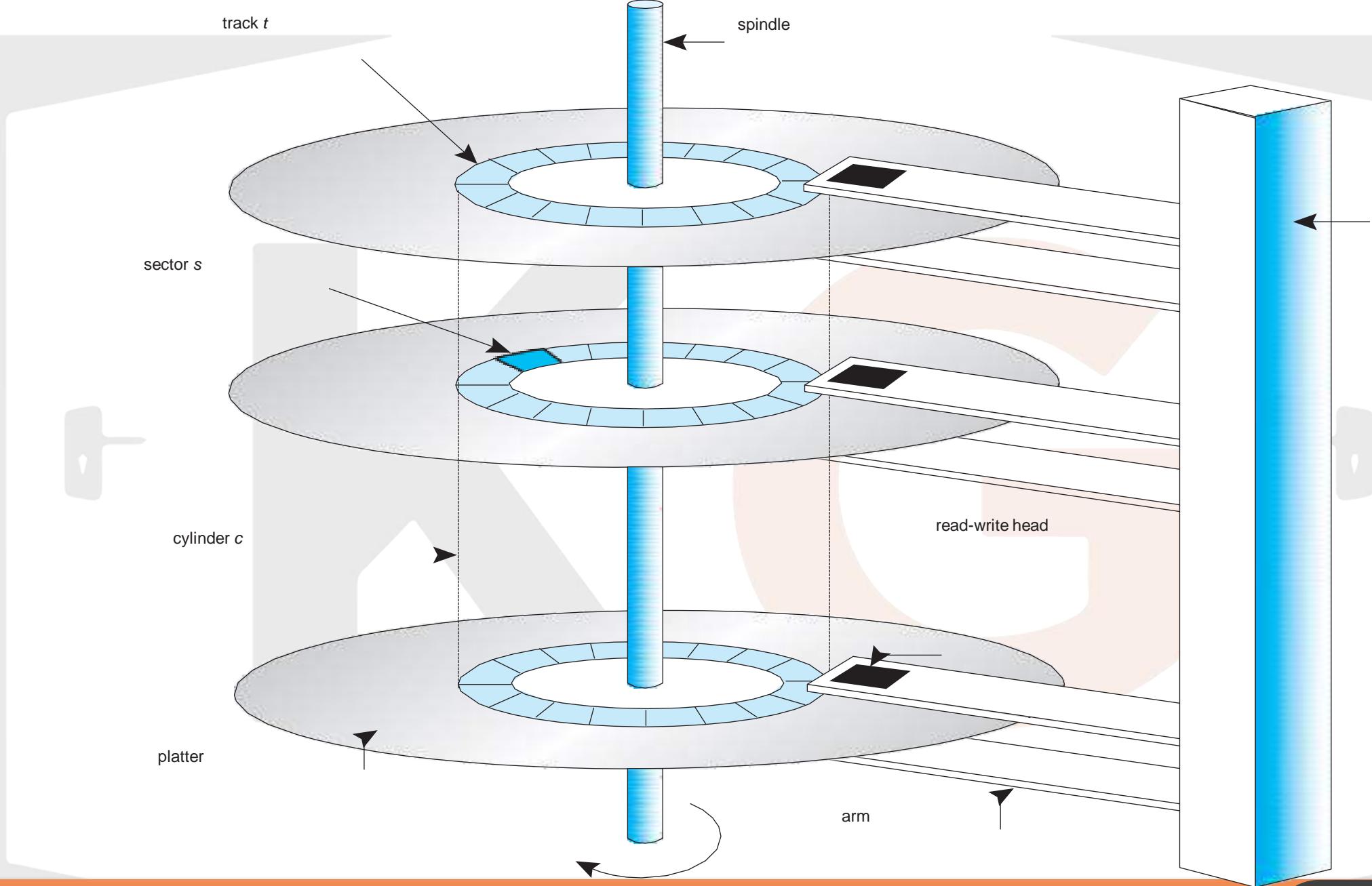
**(C) Segmentation**



**(D) Thrashing**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)



## Disk scheduling

- One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, efficiency means having fast access time and large disk bandwidth.
- The disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:

- Whether this operation is input or output
  - What is the disk address for the transfer
  - What is the memory address for the transfer
  - What is the number of sectors to be transferred
- 
- If the desired disk drive and controller are available, the request can be serviced immediately.
  - If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.
  - Thus, when one request is completed, the operating system chooses which pending request to service next. How does the operating system make this choice? Any one of several disk-scheduling algorithms can be used.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## **FCFS (First Come First Serve)**

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. In FCFS, the requests are addressed in the order they arrive in the disk queue. This algorithm is intrinsically fair, but it generally does not provide the fastest service.

### **Advantages:**

- Easy to understand easy to use
- Every request gets a fair chance
- no starvation (may suffer from convoy effect)

### **Disadvantages:**

- Does not try to optimize seek time (extra seek movements)
- May not provide the best possible service

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

Q If the disk head is located initially at 32, find the number of disk moves required with FCFS if the disk queue of I/O blocks requests are 98, 37, 14, 124, 65, 67. (NET-DEC-2012)

- (A) 239      (B) 310      (C) 321      (D) 325

Q Assuming that the disk head is located initially at 32, find the number of disk moves required with FCFS if the disk queue of I/O block requests are 98, 37, 14, 124, 65, 67: **(NET-JUNE-2013)**

- (A) 310      (B) 324      (C) 320      (D) 321

**Q If the Disk head is located initially at track 32, find the number of disk moves required with FCFS scheduling criteria if the disk queue of I/O blocks requests are: 98, 37, 14, 124, 65, 67 (NET-JULY-2016)**

a) 320

b) 322

c) 321

d) 319

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## SSTF Scheduling

- It seems reasonable to service all the requests close to the current head position before moving the head far to service other REQUESTS. This assumption is the basis for the SSTF algorithm.
- In SSTF (Shortest Seek Time First), the request nearest to the disk arm will get executed first i.e. requests having shortest seek time are executed first.
- Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal. In the example, we can do better by moving the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122, 124, and 183. This strategy reduces the total head movement to 208 cylinders.

## Advantages:

- Seek movements decreases
- Average Response Time decreases
- Throughput increases

## Disadvantages:

- Overhead to calculate the closest request.
- Can cause Starvation for a request which is far from the current location of the header
- High variance of response time as SSTF favours only some requests
- SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause starvation of some requests.

**Q** Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence 4, 34, 10, 7, 19, 73, 2, 15, 6, 20. Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1ms to move from one cylinder to adjacent one and shortest seek time first policy is used? **(GATE-2009) (1 Marks)**

- (A) 95 ms      (B) 119 ms      (C) 233 ms      (D) 276 ms

**Q** Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time, the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing \_\_\_\_\_ number of requests. (GATE-2014) (1 Marks)



**Q** Consider an imaginary disk with 40 cylinders. A request come to read a block on cylinder 11. While the seek to cylinder 11 is in progress, new requests come in for cylinders 1, 36, 16, 34, 9 and 12 in that order. The number of arm motions using shortest seek first algorithm is (NET-DEC-2014)

(A) 111

(B) 112

(C) 60

(D) 61



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Suppose SSTF disk scheduling algorithm implemented to meet the requests then the total number of head movements are \_\_\_\_\_ if the disk head is initially at 53. **(NET-NOV-2017)**

- a) 224
- b) 248
- c) 236
- d) 240



Q A disk drive has 100 cylinders, numbered 0 to 99. Disk requests come to the disk driver for cylinders 12, 26, 24, 4, 42, 8 and 50 in that order. The driver is currently serving a request at cylinder 24. A seek takes 6 msec per cylinder moved. How much seek time is needed for shortest seek time first (SSTF) algorithm? **(NET-JUNE-2015)**

a) 0.984 sec

b) 0.396 sec

c) 0.738 sec

d) 0.42 sec

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## SCAN

The disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each track, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the Elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

- If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.
- Assuming a uniform distribution of requests for cylinders, consider the density of requests when the head reaches one end and reverses direction. At this point, relatively few requests are immediately in front of the head since these cylinders have recently been serviced. The heaviest density of requests is at the other end of the disk. These requests have also waited the longest.

## Advantages:

- Simple easy to understand and use
- No starvation but more wait for some random process
- Low variance and Average response time

## Disadvantages:

- Long waiting time for requests for locations just visited by disk arm.
- Unnecessary move to the end of the disk, even if there is no request.

**Q** On a disk with 1000 cylinders (0 to 999) find the number of tracks, the disk arm must move to satisfy all the requests in the disk queue. Assume the last request service was at track 345 and the head is moving toward track 0. The queue in FIFO order contains requests for the following tracks: 123, 874, 692, 475, 105, 376 (Assume SCAN algorithm) (**NET-DEC-2012**)

- (A) 2013      (B) 1219      (C) 1967      (D) 1507

**Q** Consider a disk queue with I/O requests on the following cylinders in their arriving order: 6, 10, 12, 54, 97, 73, 128, 15, 44, 110, 34, 45. The disk head is assumed to be at cylinder 23 and moving in the direction of decreasing number of cylinders. Total number of cylinders in the disk is 150. The disk head movement using SCAN-scheduling algorithm is: (NET-JAN-2017)

a) 172

b) 173

c) 227

d) 228



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is \_\_\_\_\_ tracks (GATE-2015) (2 Marks)

**Q** Consider a disk queue with request for input/output to block on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. Assume that disk head is initially positioned at cylinder 53 and moving towards cylinder number 0. The total number of head movements using Shortest Seek Time First (SSTF) and SCAN algorithms are respectively (NET-DEC-2013)

(A) 236 and 252 cylinders  
(C) 235 and 640 cylinders

(B) 640 and 236 cylinders  
(D) 235 and 252 cylinders

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## C-SCAN Scheduling

Circular-scan is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

### **Advantages:**

- Provides more uniform wait time compared to SCAN
- Better response time compared to scan

### **Disadvantage:**

- More seeks movements in order to reach starting position

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the situation in which the disk read/write head is currently located at track 45 (of tracks 0-255) and moving in the positive direction. Assume that the following track requests have been made in this order: 40, 67, 11, 240, 87. What is the order in which C-SCAN would service these requests and what is the total seek distance?

- (A) 600      (B) 810      (C) 505      (D) 550

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## LOOK Scheduling

It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

### **Advantage: -**

- Better performance compared to SCAN
- Should be used in case to less load

### **Disadvantage: -**

- Overhead to find the last request
- Should not be used in case of more load.

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## C LOOK

As LOOK is similar to SCAN algorithm, in similar way, C-LOOK is similar to C-SCAN disk scheduling algorithm. In C-LOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

### **Advantage: -**

- Provides more uniform wait time compared to LOOK
- Better response time compared to LOOK

### **Disadvantage: -**

- Overhead to find the last request and go to initial position is more
- Should not be used in case of more load.

**Q** Consider the situation in which the disk read/write head is currently located at track 45 (of tracks 0-255) and moving in the positive direction. Assume that the following track requests have been made in this order: 40, 67, 11, 240, 87. What is the order in which optimized C-SCAN would service these requests and what is the total seek distance?**(GATE-2015) (2 Marks)**

- (A) 600
- (B) 810
- (C) 505
- (D) 550

**Q Which of the following statements is not true about disk-arm scheduling algorithms? (NET-JUNE-2014)**

- (A) SSTF (shortest seek time first) algorithm increases performance of FCFS.
- (B) The number of requests for disk service are not influenced by file allocation method.
- (C) Caching the directories and index blocks in main memory can also help in reducing disk arm movements.
- (D) SCAN and C-SCAN algorithms are less likely to have a starvation problem.

**Q** In \_\_\_\_\_ disk scheduling algorithm, the disk head moves from one end to other end of the disk, serving the requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk without serving any requests on the return trip.

**(NET-NOV-2017)**

- a) LOOK
- b) SCAN
- c) C-LOOK
- d) C-SCAN



**Q** Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is \_\_\_\_\_ tracks (GATE-2015) (2 Marks)



**Q** Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is \_\_\_\_\_. (GATE-2018) (2 Marks)



# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Conclusion

## **Selection of a Disk-Scheduling Algorithm**

- Given so many disk-scheduling algorithms, how do we choose the best one? SSTF is common and has a natural appeal because it increases performance over FCFS. SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem.
- With any scheduling algorithm, however, performance depends heavily on the number and types of requests. For instance, suppose that the queue usually has just one outstanding request. Then, all scheduling algorithms behave the same, because they have only one choice of where to move the disk head: they all behave like FCFS scheduling.

**Q** Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs? **(GATE-2004) (1 Marks)**

(A) 50%

(B) 40%

(C) 25%

(D) 0%

**Q** Which of the following disk scheduling strategies is likely to give the best throughput? **(GATE-1999) (1 Marks)**

- (a)** Farthest cylinder next
- (c)** First come first served
- (b)** Nearest cylinder next
- (d)** Elevator algorithm

## Q Match the following (NET-DEC-2004)

(a) Disk scheduling	(1) Round robin
(b) Batch processing	(2) Scan
(c) Time sharing	(3) LIFO
(d) Interrupt processing	(4) FIFO

Codes:

(A)	a-3	b-4	c-2	d-1
(B)	a-4	b-3	c-2	d-1
(C)	a-2	b-4	c-1	d-3
(D)	a-3	b-4	c-1	d-2

**Q** The maximum amount of information that is available in one portion of the disk access arm for a removal disk pack (without further movement of the arm with multiple heads) **(NET-DEC-2011)**

(A) a plate of data  
(C) a track of data

(B) a cylinder of data  
(D) a block of data



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider the input/output (I/O) requests made at different instants of time directed at a hypothetical disk having 200 tracks as given in the following table

Serial No	1	2	3	4	5
Track No	12	85	40	100	75
Time of arrival	65	80	110	100	175

Assume that :

Current head position is at track no. 65 Direction of last movement is towards higher numbered tracks Current clock time is 160 milliseconds Head movement time per track is 1 millisecond. “look” is a variant of “SCAN” disk-arm scheduling algorithm. In this algorithm, if no more I/O requests are left in current direction, the disk head reverses its direction. The seek times in Shortest Seek First (SSF) and “look” disk-arm scheduling algorithms respectively are:

**(NET-SEP-2013)**

- (A) 144 and 123 milliseconds
- (C) 149 and 124 milliseconds

- (B) 143 and 123 milliseconds
- (D) 256 and 186 milliseconds

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, ..., 199), and 256 sectors per track (numbered as 0, 1, ... 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 16, 1]

Currently head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts. Power dissipation associated with rotational latency and switching of head between different platters is negligible.

The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithm is \_\_\_\_\_. (GATE-2018) (2 Marks)

**Q** For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to **(GATE-2008) (1 Marks)**

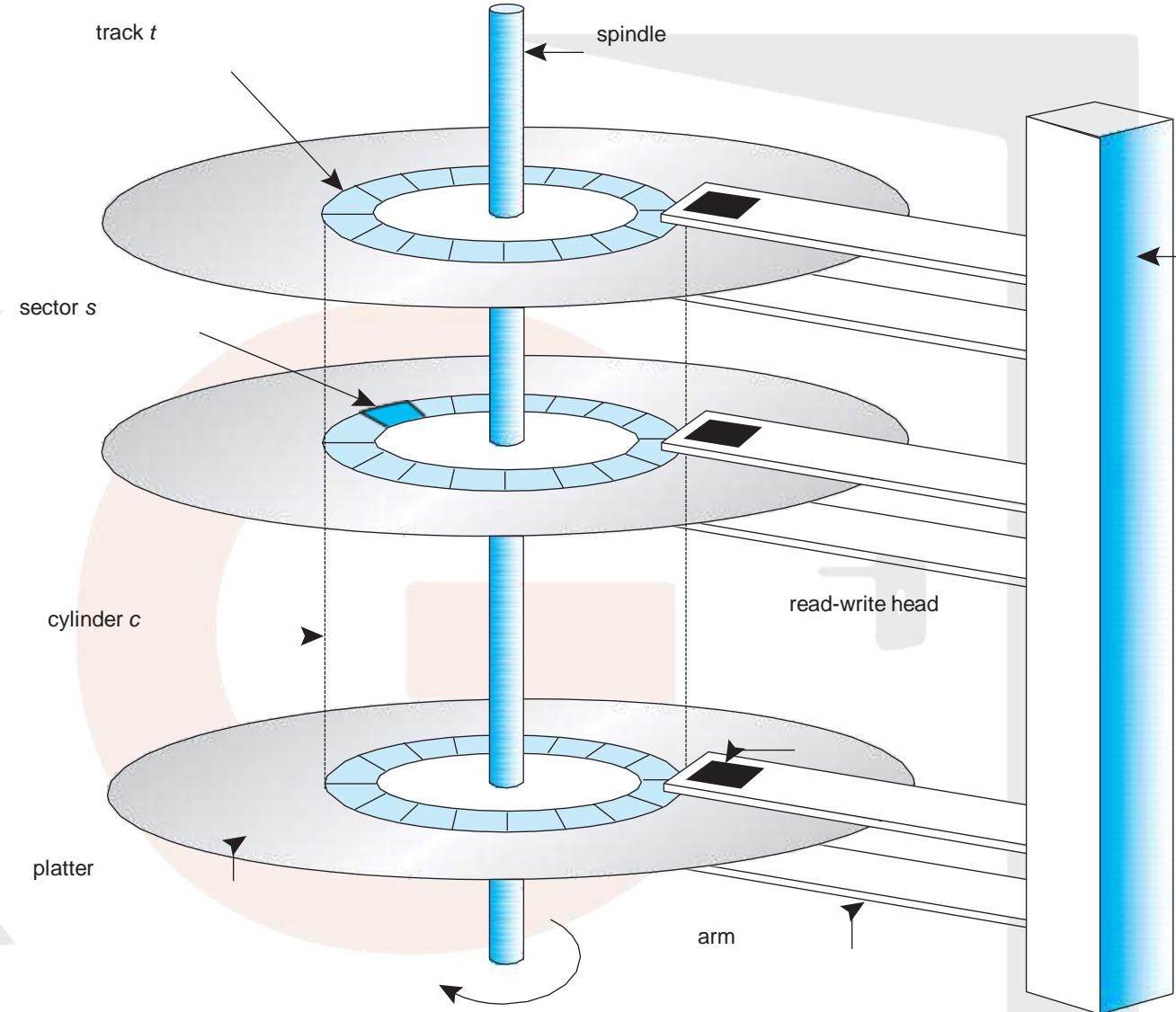
- a) non-uniform distribution of requests
- b) arm starting and stopping inertia
- c) higher capacity of tracks on the periphery of the platter
- d) use of unfair arm scheduling policies

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Transfer time

- Magnetic disks provide the bulk of secondary storage for modern computer systems. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.
- A read – write head “flies” just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.

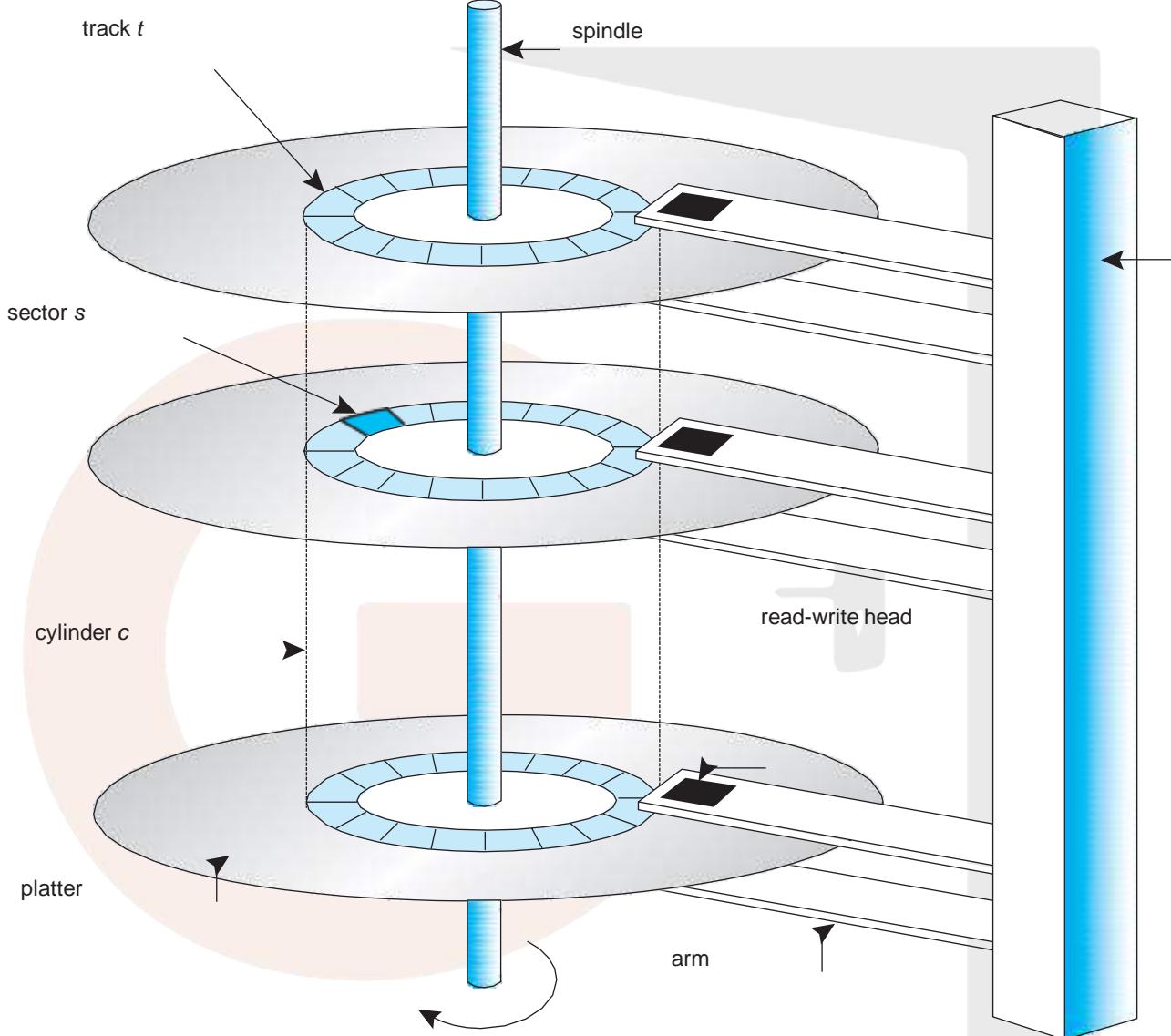


To access all paid content get **KG Prime** at ₹25/day

[CLICK HERE ↗](#)



- When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute (**RPM**). Common drives spin at 5,400, 7,200, 10,000, and 15,000 RPM.
- Total transfer time has two parts. The **positioning time**, or **random-access time**, consists of two parts: the time necessary to move the disk arm to the desired cylinder, called the **seek time**, and the time necessary for the desired sector to rotate to the disk head, called the **rotational latency**.



- Total Transfer Time = Seek Time + Rotational Latency + Transfer Time
- **Seek Time**: - It is a time taken by Read/Write header to reach the correct track. (Always given in question)
- **Rotational Latency**: - It is the time taken by read/Write header during the wait for the correct sector. In general, it's a random value, so far average analysis, we consider the time taken by disk to complete half rotation.
- **Transfer Time**: - it is the time taken by read/write header either to read or write on a disk. In general, we assume that in 1 complete rotation, header can read/write the either track, so
- total time will be = (File Size/Track Size) \*time taken to complete one revolution.

**Q** For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to **(GATE-2008) (2 Marks)**

- (A)** non-uniform distribution of requests
- (B)** arm starting and stopping inertia
- (C)** higher capacity of tracks on the periphery of the platter
- (D)** use of unfair arm scheduling policies

Q Consider a disk where there are 512 tracks, each track is capable of holding 128 sectors and each sector holds 256 bytes, find the capacity of the track and disk and number of bits required to reach correct track, sector and disk.



**Q** Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively **(GATE-2006) (2 Marks)**

(A) 256 Mbyte, 19 bits

(B) 256 Mbyte, 28 bits

(C) 512 Mbyte, 20 bits

(D) 64 Gbyte, 28 bit

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

# Break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

Q consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?



**Q Consider a disk with 16384 bytes per track having a rotation time of 16 msec and average seek time of 40 msec. What is the time in msec to read a block of 1024 bytes from this disk? (NET-DEC-2015)**

- a) 57 sec
- b) 49 sec
- c) 48 sec
- d) 17 sec

**Q** Consider a system with 8 sector per track and 512 bytes per sector. Assume that disk rotates at 3000 rpm and average seek time is 15ms standard. Find total time required to transfer a file which requires 8 sectors to be stored.

- a)** Assume contiguous allocation
  
- b)** Assume Non- contiguous allocation

**Q** Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is \_\_\_\_\_. (GATE-2015) (2 Marks)



**Q** Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of  $50 \times 10^6$  bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is \_\_\_\_\_ (GATE-2015) (2 Marks)

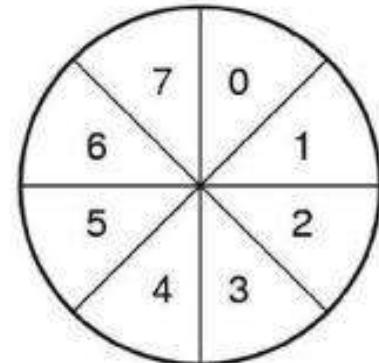


**Q** An application loads 100 libraries at start-up. Loading each library requires exactly one disk access. The seek time of the disk to a random location is given as 10 milli second. Rotational speed of disk is 6000 rpm. If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries? (The time to transfer data from the disk block once the head has been positioned at the start of the block may be neglected) (GATE-2011) (2 Marks)

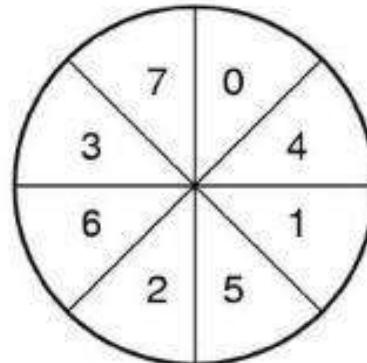
- (A) 0.50 s      (B) 1.50 s      (C) 1.25 s      (D) 1.00 s

- **Single interleaving:** - in 2 rotation we read 1 track

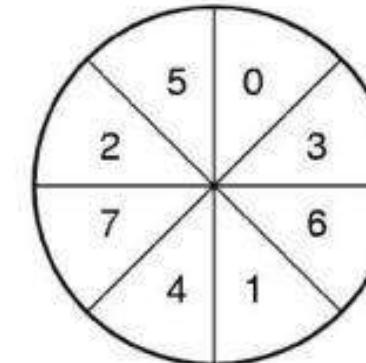
- **Double interleaving:** - in 2.75 rotation we read 1 track



(a)



(b)



(c)

- No interleaving
- Single interleaving
- Double interleaving

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

## File allocation methods

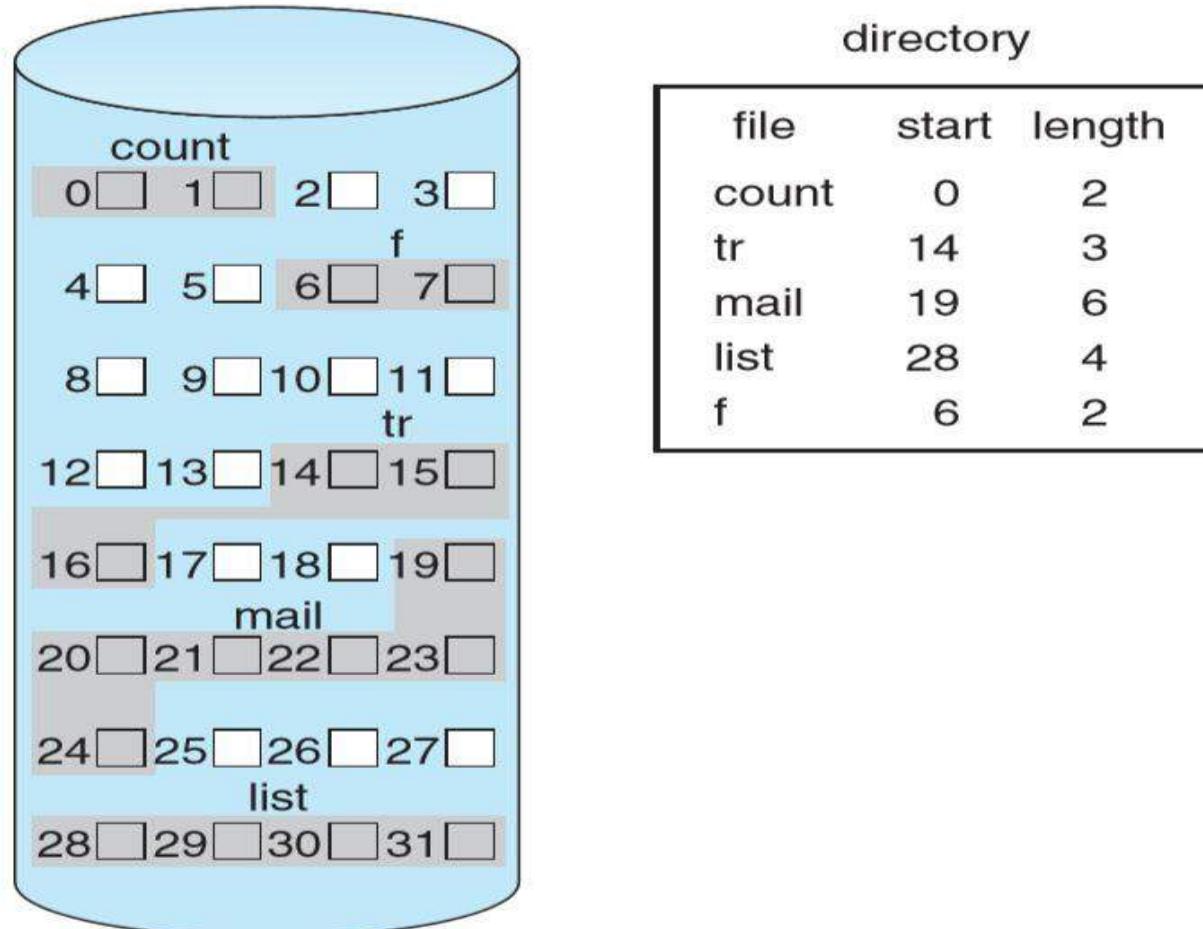
The main aim of file allocation problem is how disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use:

- **Contiguous**
- **Linked**
- **Indexed**

Each method has advantages and disadvantages. Although some systems support all three, it is more common for a system to use one method for all files.

## Contiguous Allocation

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block  $b + 1$  after block  $b$  normally requires no head movement.



- **Advantage**

- Accessing a file that has been allocated contiguously is easy. Thus, both sequential and direct access can be supported by contiguous allocation.

- **Disadvantage**

- Suffer from the problem of external fragmentation.
- Suffer from huge amount of external fragmentation.
- Another problem with contiguous allocation file modification

**Q** Suppose there are six files  $F_1, F_2, F_3, F_4, F_5, F_6$  with corresponding sizes 150 KB, 225 KB, 75 KB, 60 KB, 275 KB and 65 KB respectively. The files are to be stored on a sequential device in such a way that optimizes access time. In what order should the files be stored? **(NET-NOV-2017)**

- A)**  $F_5, F_2, F_1, F_3, F_6, F_4$
- c)**  $F_1, F_2, F_3, F_4, F_5, F_6$

- b)**  $F_4, F_6, F_3, F_1, F_2, F_5$
- d)**  $F_6, F_5, F_4, F_3, F_2, F_1$

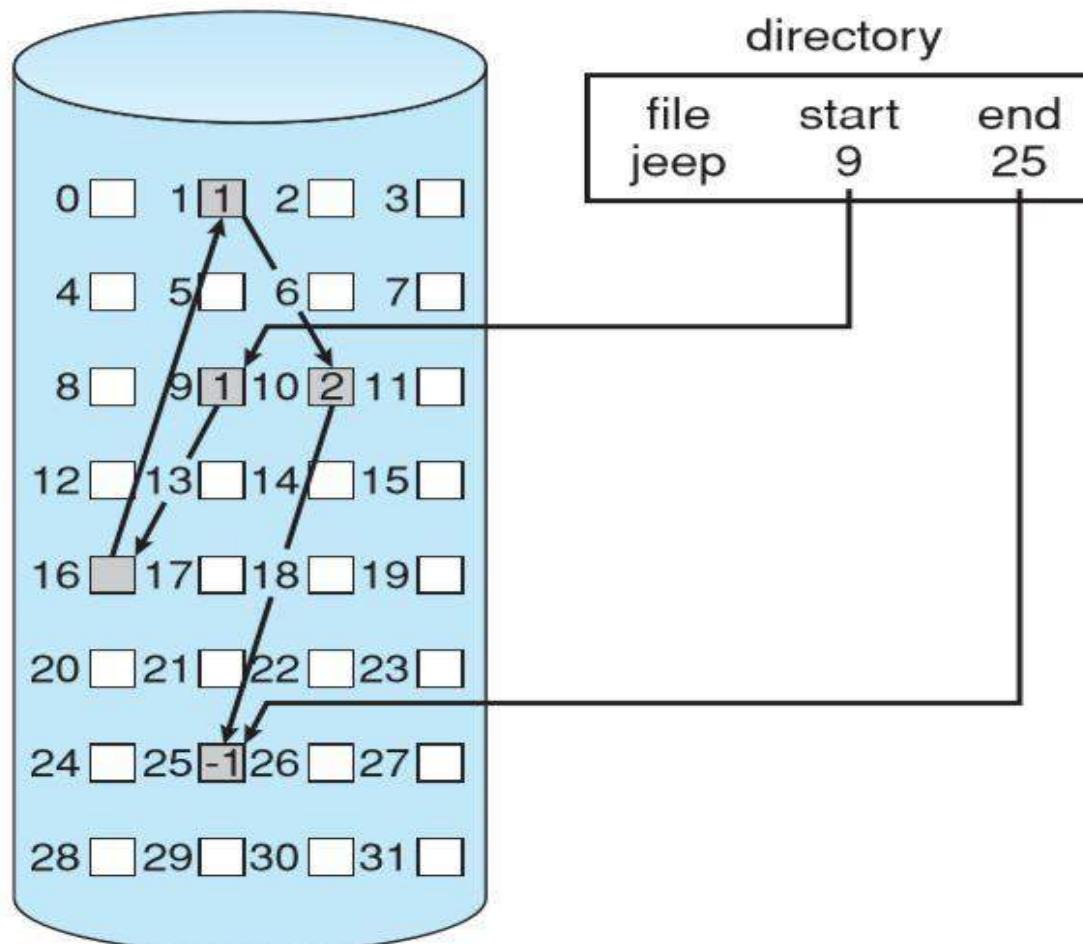


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

# Linked Allocation

- Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.



- **Advantage: -**

- To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. To read a file, we simply read blocks by following the pointers from block to block. The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available.
- There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.

- **Disadvantage:** -
  - To find the  $i^{\text{th}}$  block of a file, we must start at the beginning of that file and follow the pointers until we get to the  $i^{\text{th}}$  block. Each access to a pointer requires a disk read.
  - Another disadvantage is the space required for the pointers, so each file requires slightly more space than it would otherwise.
  - Yet another problem is reliability. Recall that the files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged. One partial solution is to use doubly linked lists, and another is to store the file name and relative block number in each block. However, these schemes require even more overhead for each file.

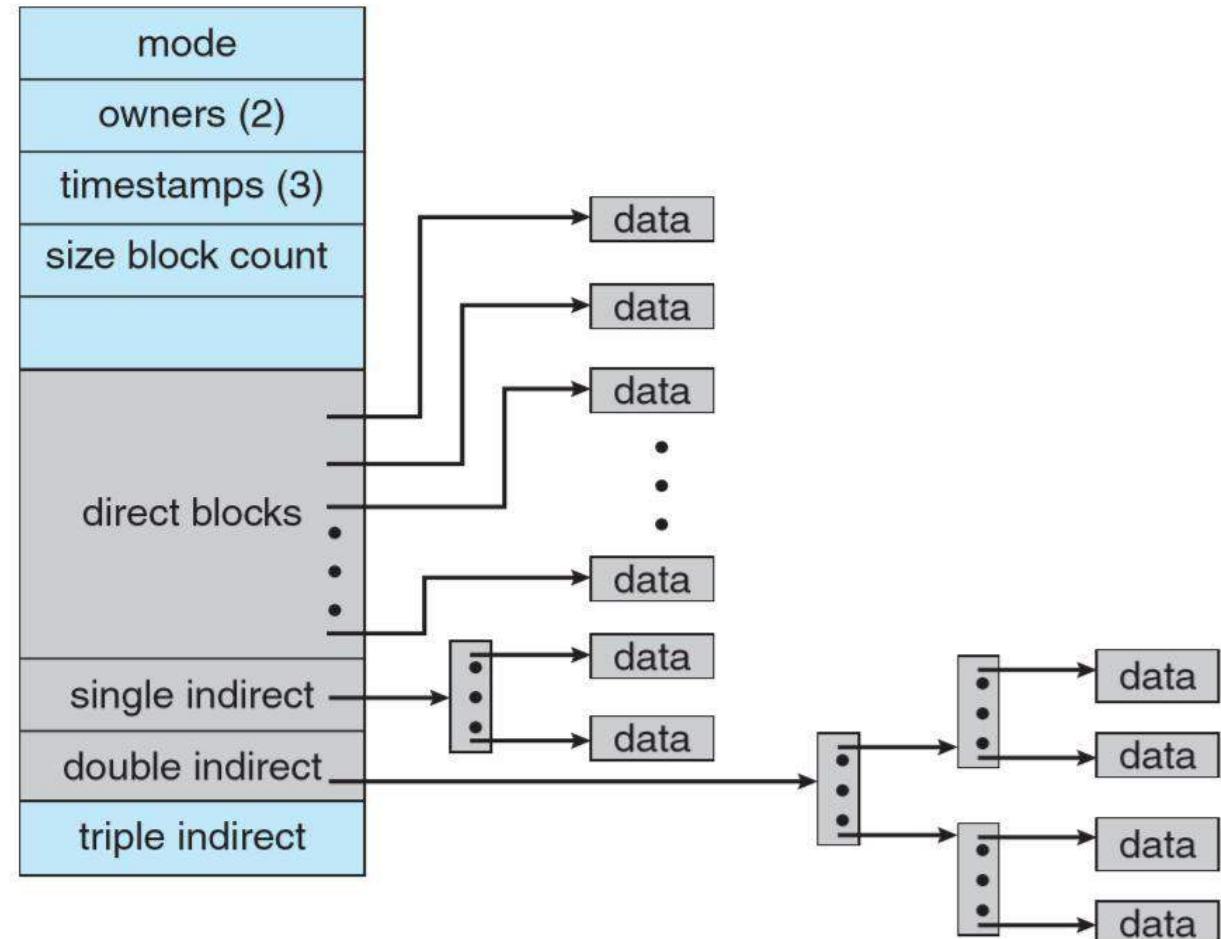
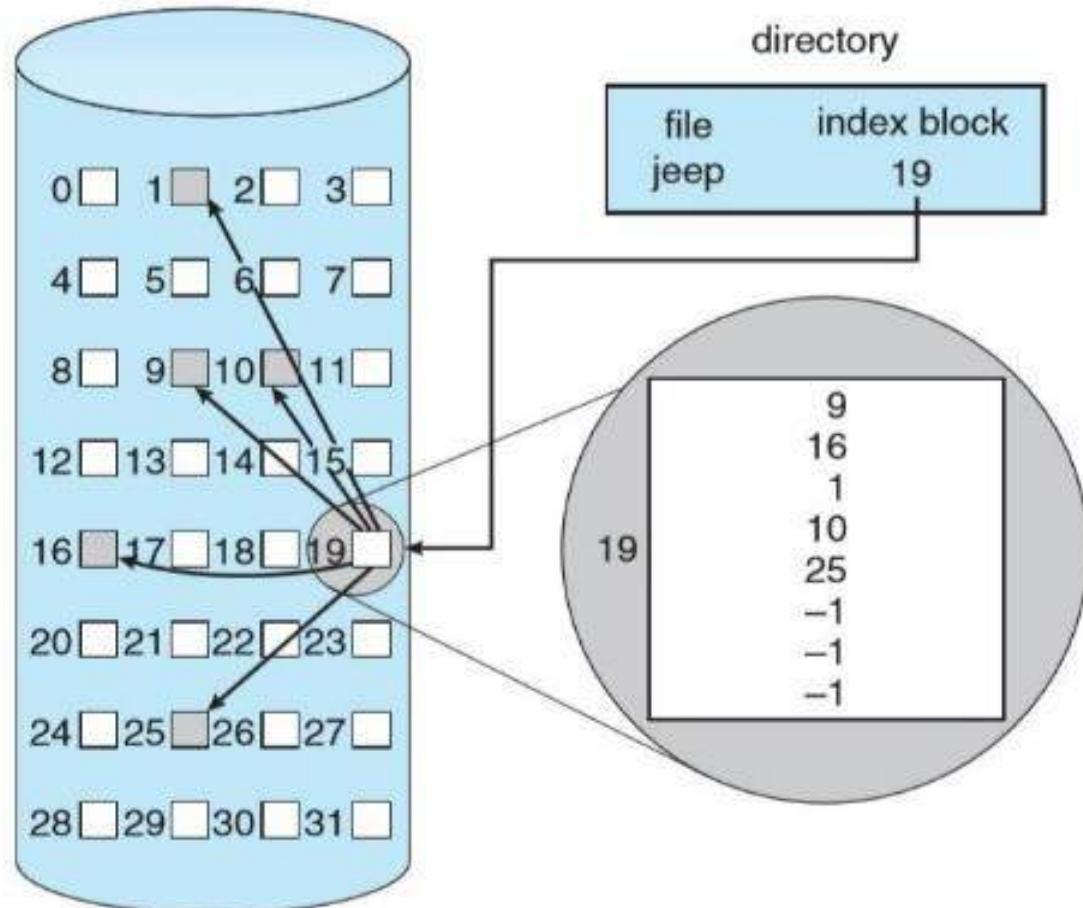


# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Indexed Allocation

- Indexed allocation solves problems of contiguous and linked allocation, by bringing all the pointers together into one location: the index block.



- Each file has its own index block, which is an array of disk-block addresses. The  $i^{th}$  entry in the index block points to the  $i^{th}$  block of the file. The directory entry contains the address of the index block. To find and read the  $i^{th}$  block, we use the pointer in the  $i^{th}$  index-block entry.
- When the file is created, all pointers in the index block are set to null. When the  $i^{th}$  block is first written, a block is obtained from the free-space manager, and its address is put in the  $i^{th}$  index-block entry.
- This point raises the question of how large the index block should be. Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file.

- **Linked scheme:** To allow for large files, we can link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is null (for a small file) or is a pointer to another index block (for a large file).
- **Multilevel index.** A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size.

- **Combined scheme.** Another alternative, used in UNIX-based file systems, is to keep the first, say, 15 pointers of the index block in the file's inode.
- The first 12 of these pointers point to direct blocks; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files do not need a separate index block.
- The next three pointers point to indirect blocks. The first points to a single indirect block, which is an index block containing not data but the addresses of blocks that do contain data.
- The second points to a double indirect block, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The last pointer contains the address of a triple indirect block.

- **Advantage**

- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.

- **Disadvantage**

- Indexed allocation does suffer from wasted space, however. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** In a file allocation system, which of the following allocation scheme(s) can be used if no external fragmentation is allowed? **(GATE-2017) (1 Marks)**

- I. Contiguous
- II. Linked
- III. Indexed

**(a)** I and III only

**(b)** II only

**(c)** III only

**(d)** II and III only

**Q** Consider a file currently consisting of 50 blocks. Assume that the file control block and the index block is already in memory. If a block is added at the end (and the block information to be added is stored in memory), then how many disk I/O operations are required for indexed (single-level) allocation strategy? **(NET-NOV-2016)**

A) 1

b) 101

c) 27

d) 0

## Q Match the following: (NET-JUNE-2014)

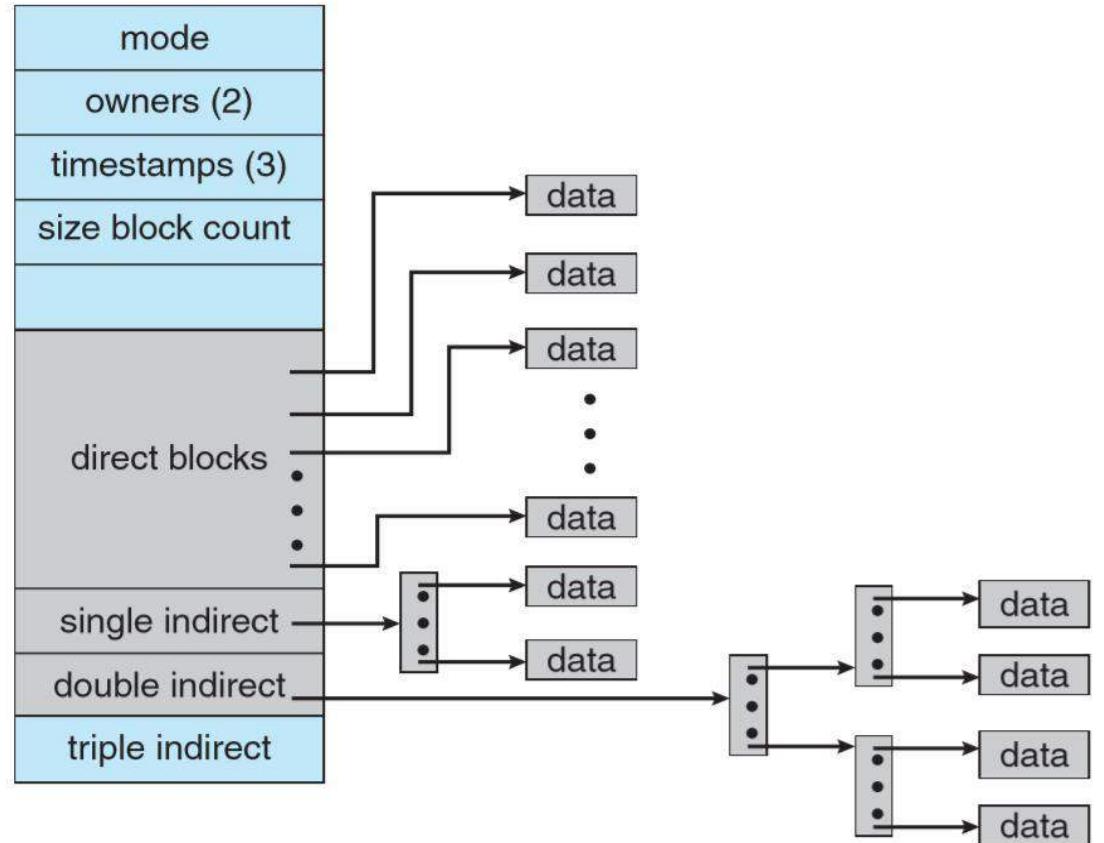
List-I	List-II
a) Contiguous allocation	i) This scheme supports very large file sizes.
b) Linked allocation	ii) This allocation technique supports only sequential files.
c) Indexed allocation	iii) Number of disks required to access file is minimal.
d) Multi-level indexed	iv) This technique suffers from maximum wastage of space in storing pointers.

	(a)	(b)	(c)	(d)
(a)	(iii)	(iv)	(ii)	(i)
(b)	(iii)	(ii)	(iv)	(i)
(c)	(i)	(ii)	(iv)	(iii)
(d)	(i)	(iv)	(ii)	(iii)

To access all paid content get **KG Prime** at ₹25/day

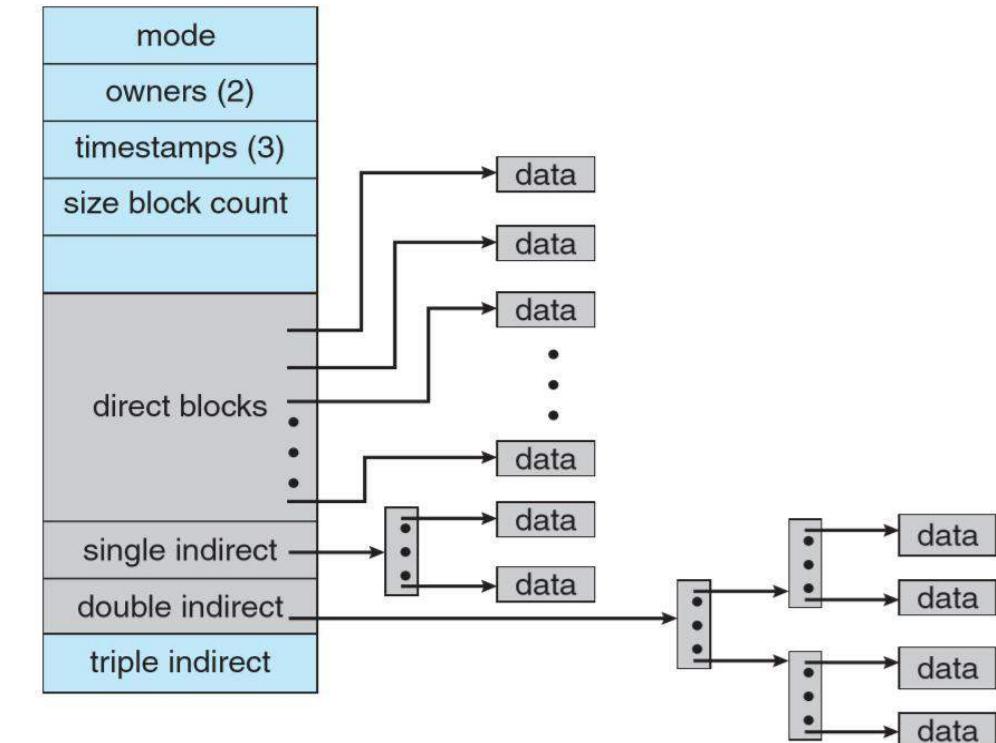
**CLICK HERE ↗**

Q The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4 kB, and the disk block address is 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) \_\_\_\_\_ GB. (GATE-2014) (2 Marks)



Q A file system with 300 Gbyte disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 Bytes and the size of each disk block address is 8 Bytes. The maximum possible file size in this file system is **(GATE-2012) (2 Marks)**

- (A) 3 Kbytes
- (B) 35 Kbytes
- (C) 280 Bytes
- (D) Dependent on the size of the disk



**Q The data blocks of a very large file in the Unix file system are allocated using (GATE-2008) (1 Marks)**

**(A) contiguous allocation**

**(B) linked allocation**

**(C) indexed allocation**

**(D) an extension of indexed allocation**

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

**Q** A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointer. Disk block size is 1 Kbyte, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size? **(GATE-2004) (2 Marks)**

- (A)  $2^{24}$  bytes
- (B)  $2^{32}$  bytes
- (C)  $2^{34}$  bytes
- (D)  $2^{48}$  bytes

**Q** In the index allocation scheme of blocks to a file, the maximum possible size of the file depends on **(GATE-2002) (1 Marks)**

- (a)** the size of the blocks, and the size of the address of the blocks.
- (b)** the number of blocks used for the index, and the size of the blocks.
- (c)** the size of the blocks, the number of blocks used for the index, and the size of the address of the blocks.
- (d)** None of the above



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Free-Space Management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks—those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
- This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

**Q** How many disk blocks are required to keep list of free disk blocks in a 16 GB hard disk with 1 kB block size using linked list of free disk blocks? Assume that the disk block number is stored in 32 bits. **(NET-DEC-2014)**

**(A)** 1024 blocks

**(B)** 16794 blocks

**(C)** 20000 blocks

**(D)** 1048576 blocks

To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**



# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Bit Vector

- Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be 001111001111110001100000011100000 ...
- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.

- Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory. Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.
- A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks.
- A 1-TB disk with 4-KB blocks requires 256 MB to store its bit map. Given that disk size constantly increases, the problem with bit vectors will continue to escalate as well.

Q How much space will be required to store the bit map of a 1.3 GB disk with 512 bytes block size? (NET-DEC-2013)

(A) 332.8 KB

(B) 83.6 KB

(C) 266.2 KB

(D) 256.6 KB

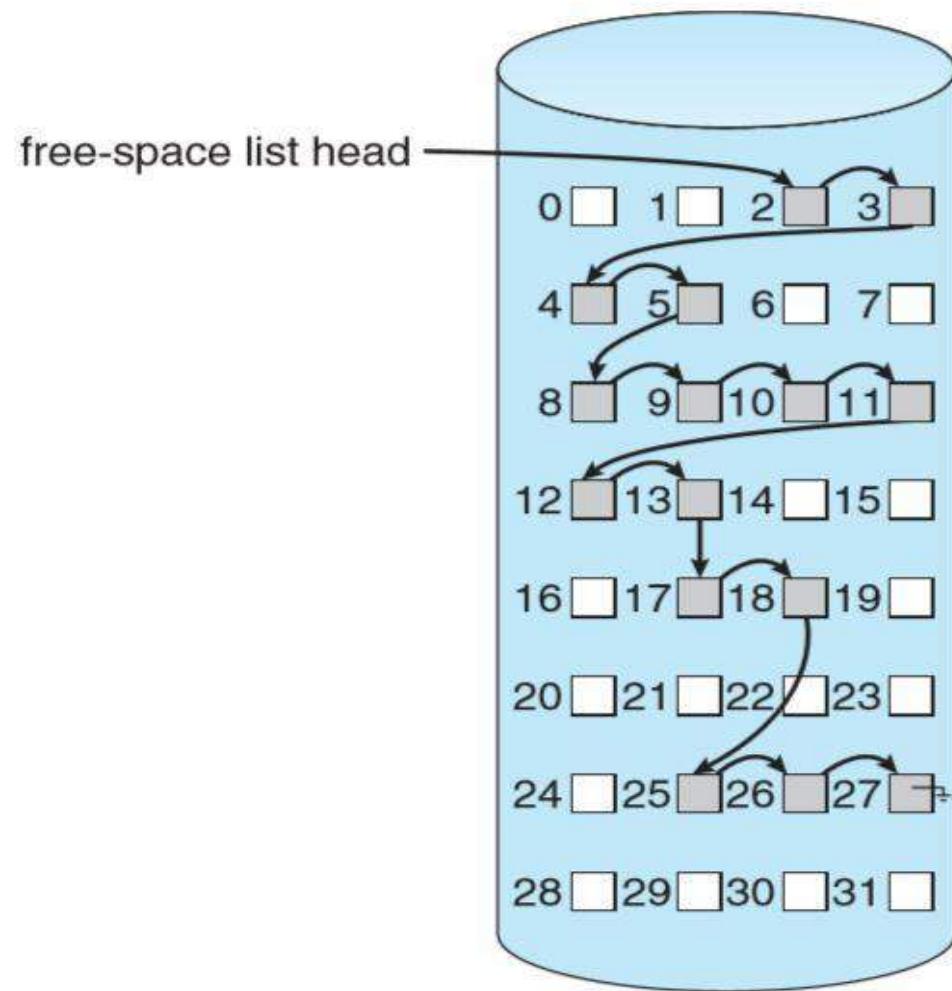


# break

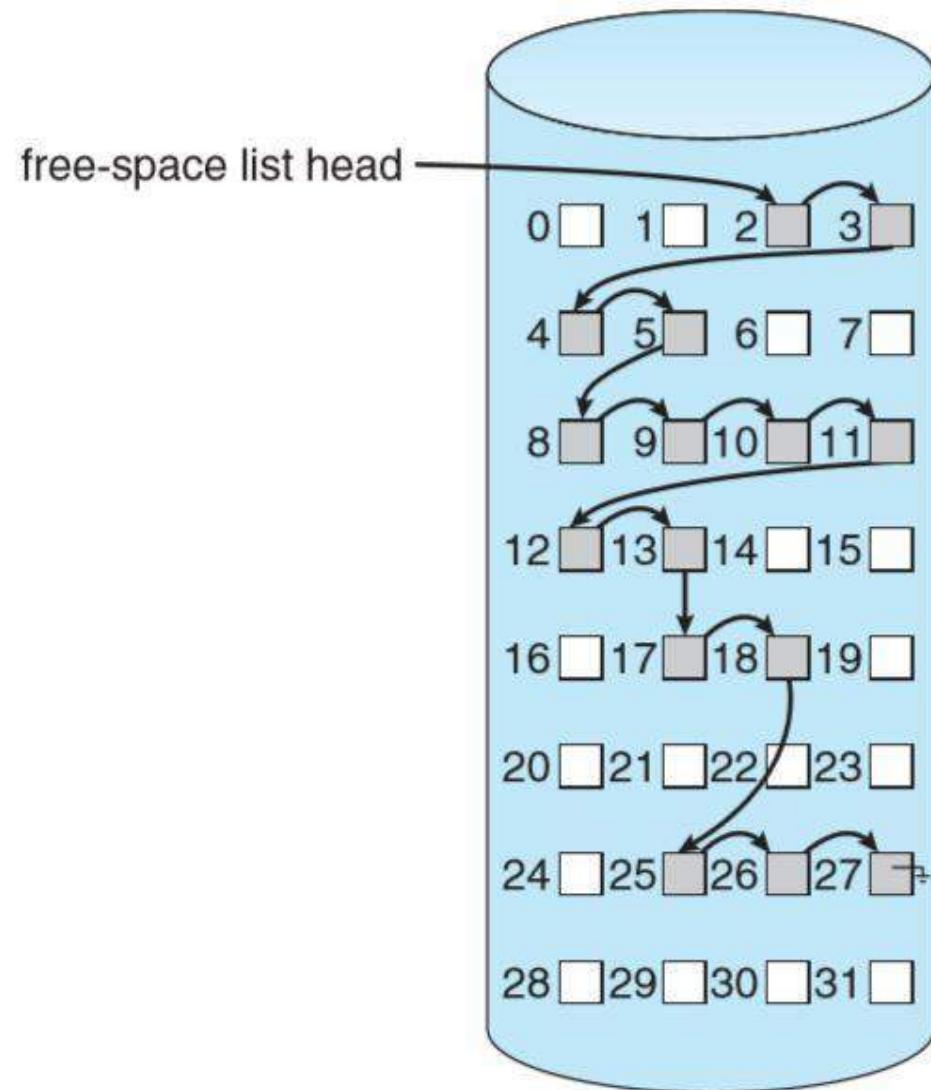
To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

## Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on. Recall our earlier example, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.



- In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.
- Fortunately, however, traversing the free list is not a frequent action. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.



Q A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a  $100 \times 10^6$  bytes disk on which the file system is stored and data block size is  $10^3$  bytes, the maximum size of a file that can be stored on this disk in units of  $10^6$  bytes is \_\_\_\_\_. (GATE-2014) (2 Marks)

# break

To access all paid content get **KG Prime** at ₹25/day [CLICK HERE ↗](#)

**Q** An experimental file server is up 75% of the time and down for 25% of the time due to bugs. How many times does this file server have to be replicated to give an availability of at least 99%? (NET-NOV-2016)

a) 2

b) 4

c) 8

d) 16



Q.24

Consider a linear list based directory implementation in a file system. Each directory is a list of nodes, where each node contains the file name along with the file metadata, such as the list of pointers to the data blocks. Consider a given directory foo.

**(Gate-2021) (1 Marks)**

Which of the following operations will necessarily require a full scan of foo for successful completion?

- (a) Opening of an existing file in foo
- (b) Renaming of an existing file in foo
- (c) Creation of a new file in foo
- (d) Deletion of an existing file from foo

**Ans. (a, b, d)**

[Note: But the official answer key given by IIT-Bombay is option (b, c)]



To access all paid content get **KG Prime** at ₹25/day

**CLICK HERE ↗**

*Bedtime Stories on*  
**Operating Systems**



*Bedtime Stories on*  
**Operating Systems**

Bedtime Stories on Operating Systems  
© Nobody, 2015.

The author will not like to take (dis)credit for writing this book.  
The authenticity and the originality of the contents of this book are debatable.  
This book is updated regularly, make sure that you are reading the latest version.  
Depictions of dinosaurs in popular media are grossly flawed.

- This book has been printed on environment-friendly paper •

## Contents

1. An Introduction to Operating Systems .....	1
2. Process and Process Scheduling .....	5
3. Process Synchronization .....	15
4. Deadlocks .....	21
5. Memory Management .....	27
6. Virtual Memory Management .....	33
7. Storage Management .....	39
8. (a) I/O Management and (b) System Protection and Security .....	43



## UNIT 1

### AN INTRODUCTION TO OPERATING SYSTEMS

A **computer** is a general purpose device that can execute sequences of instructions presented in a formal format to perform numerical calculations and other tasks.

Revise: block diagram of a computer, concept of memory hierarchy.

**Computer science** is the study of computer systems and computing processes.

Read: Newell, A., Perlis, A. J. and Simon, H. I. 1967. Is 'computer science' science or engineering? *Science*, **167**(3795): 1373-1374.

**Computer hardware** is the collection of all physical elements of the computer system.

**Computer software** is the collection of all programs stored in and executed by a computer system.

**Application software** performs specific task for the user.

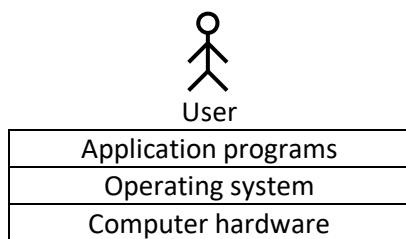
**System software** operates and controls the computer system, and provides a platform to run application software.

An **operating system** is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner.

Operating systems exist because they offer a reasonable way to solve the problem of creating a usable computer system.

An operating system –

- manages the computer hardware
- facilitates execution of application programs
- acts as an intermediary between the user and the computer hardware
- designed to be convenient and efficient



The operating system provides the means for proper use of the resources in the operation of the computer system.

Design goals –

- |   |                              |
|---|------------------------------|
| - convenience (ease of use)               | - personal computers         |
| - efficiency (proper resource allocation) | - high performance computers |
| - energy conservation                     | - handheld devices           |
| - minimal user interference               | - embedded devices           |

An operating system acts as an –

- **Resource allocator**
- **Control program**

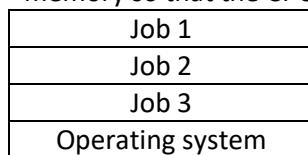
First operating system –

- ATLAS (Manchester Univ., late 1950s – early 1960s)
- evolved from control programs

Types of operating systems –

- Single process operating system [MS DOS, 1981]
- Batch-processing operating system [ATLAS, Manchester Univ., late 1950s – early 1960s]
- Multiprogramming operating system [THE, Dijkstra, early 1960s]
- Multitasking operating system [CTSS, MIT, early 1960s]

**Multiprogramming** increases CPU utilization by keeping multiple jobs (code and data) in the memory so that the CPU always has one to execute.



**Multitasking** is a logical extension of multiprogramming.

CPU executes multiple tasks by switching among them.

The switching is very fast.

Requires an interactive (hands-on) computer where the user can directly interact with the computer.

Response time should be minimal.

A **kernel** is that part of the operating system which interacts directly with the hardware and performs the most crucial tasks.

A **microkernel** is much smaller in size than a conventional kernel and supports only the core operating system functionalities.

A **shell**, also known as a command interpreter, is that part of the operating system that receives commands from the users and gets them executed.

A **system call** is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.

User programs typically do not have permission to perform operations like accessing I/O devices and communicating other programs.

A user program invokes system calls when it requires such services.

System calls provide an interface between a program and the operating system.

System calls are of different types.

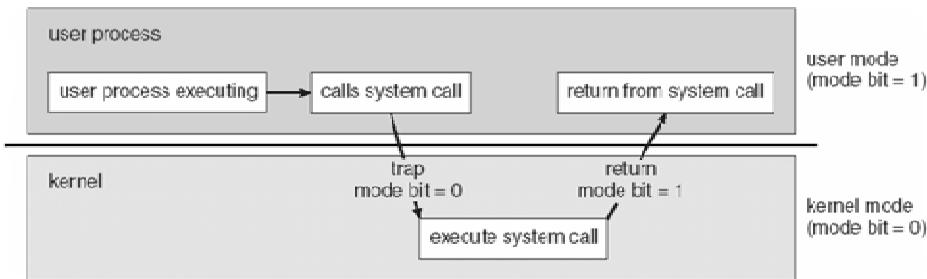
E.g. – fork, exec, getpid, getppid, wait, exit.

**Dual-mode operation –**

- **User mode**
- **Kernel mode / supervisor mode / system mode / privileged mode**

Mode bit– 0: kernel, 1: user

Request using a system call



Duties of an operating system –

- Process management
  - creating and deleting user and system processes
  - suspending and resuming processes
  - interprocess communication
  - process synchronization
  - deadlock handling
- Memory management
  - Keeping track of which part of memory is being used by which job
  - Allocating and deallocating memory space
- Storage management
  - file system management
    - creating, deleting and manipulating files and directories
  - mass storage management
    - free space management
    - storage allocation
    - disk scheduling
- Caching
- Input-output management

Operating system services –

- Helpful to the user
  - user interface (CUI/shell and GUI)
  - program execution
  - I/O operation
  - file system manipulation
  - communication
  - error detection
- Helpful to the system
  - resource allocation
  - accounting
  - protection and security

Operating system structures –

- Monolithic [MS DOS, Unix, Linux]
- Layered [THE]
- Microkernel [Mach, MINIX]

A **real-time operating system (RTOS)** has well-defined and fixed time constraints which have to be met or the system will fail.

An RTOS is used when rigid time constraints have been placed on the operation of processes or flow of data.

An RTOS is often used in the control device in a dedicated application.

Hard- and soft- RTOS.

Applications: embedded systems, robotics, scientific utilities, etc.

Operating systems for smart phones –

- CPUs of smart phones are made to be much slower to conserve energy

**Booting** is the process of starting the computer and loading the kernel.

When a computer is turned on, the power-on self-tests (POST) are performed.

Then the bootstrap loader, which resides in the ROM, is executed.

The bootstrap loader loads the kernel or a more sophisticated loader.

Is a **device driver** a part of the operating system?

Case studies: CP/M, MS DOS, Unix, Linux, Windows, etc.

<http://amturing.acm.org> –

- Edsger Wybe Dijkstra
- Kenneth Lane Thompson
- Frederick Phillips Brooks, Jr.
- Barbara Liskov

Chapters: 1 and 2.

Sections: 1.1, 1.4 – 1.9, 1.11, 2.1 – 2.5, 2.7 and 2.10.

## UNIT 2

### PROCESS AND PROCESS SCHEDULING

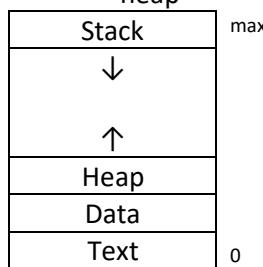
A **process** is a program in execution.

A process is a unit of work in a computer system.

The terms process and job are used interchangeably.

A process comprises of –

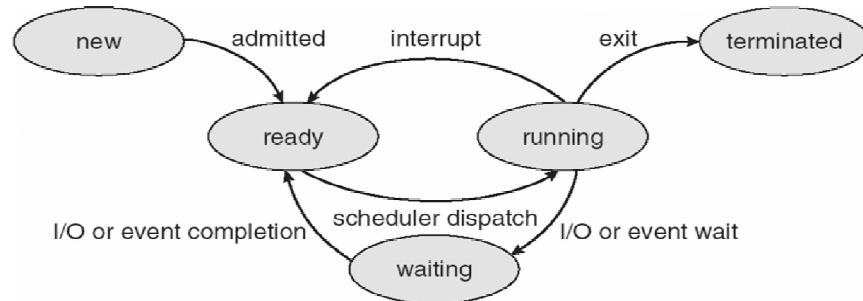
- text section containing the program code
- current activity represented by the values of the program counter and other registers
- program stack
- data section containing global variables
- heap



A program is a passive entity while a process is an active entity.

**Process state** is defined by the current activity of the process.

As a process executes, its state changes.



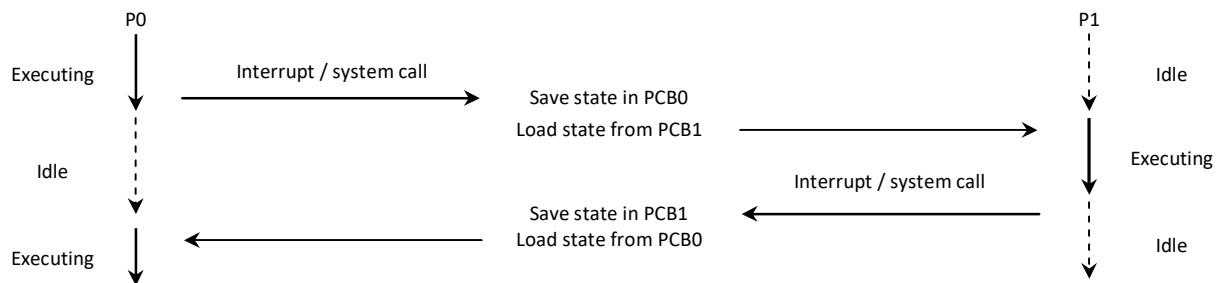
Only one process can be in the running state at any instant.

Many processes can be ready or waiting.

Each process is internally represented by the operating system by a **process control block (PCB)** also called task control block.

PCB contains all information associated with the process –

- Process state
- Values of program counter and other registers
- CPU scheduling information
- Accounting information
- I/O status information
- priority, pointer to scheduling queue, etc.
- process id, CPU- and real-time used, time limits, etc.
- list of i/o devices allocated, list of open files, etc.



**Process scheduling** is selecting one process for execution out of all the ready processes.

The objective of multiprogramming is to have some process running at all times so as to maximize CPU utilization.

The objective of multitasking is to switch the CPU among the processes so frequently that the user can interact with each process while it is running.

To meet these objectives the process scheduler selects one of the available processes for execution.

**Scheduling queues** are used to perform process scheduling.

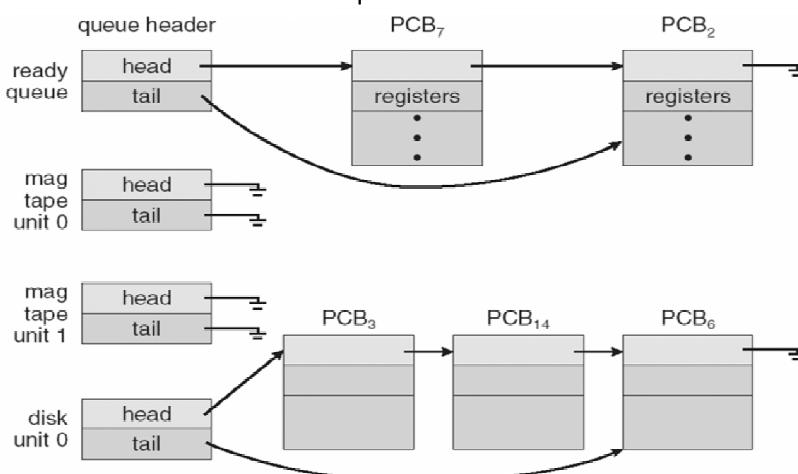
As a process enters the system, it is put in a **job queue** that contains all the processes in the system.

The processes that are residing in the memory and are ready for execution are kept in the **ready queue**.

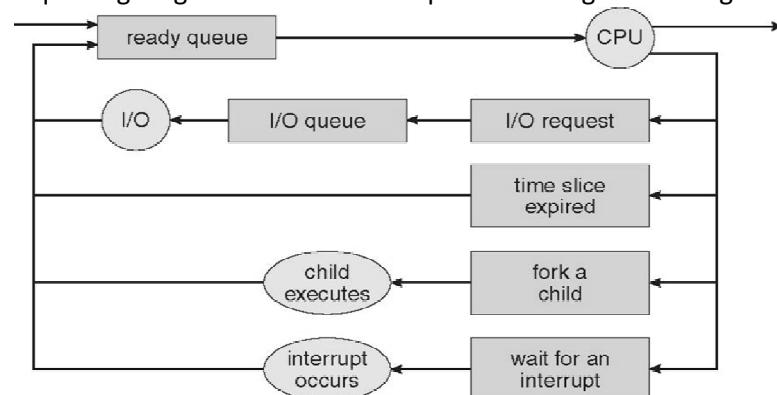
The ready queue is implemented as a linked list of PCBs with a header containing pointers to the first and the last PCBs.

The list of the processes waiting for a particular i/o device is called a **device queue**.

Each device has its own device queue.



A queuing diagram shows how the processes migrate among the various scheduling queues.



There are three types of **schedulers**.

A process migrates among the various scheduling queues throughout its lifetime.

The operating system has to select the processes from the queues according to some criteria.

The selection is done by the appropriate scheduler.

A **long-term scheduler (job scheduler)** selects processes from those submitted by the user and loads them into the memory.

The long-term scheduler controls the degree of multiprogramming which is represented by the number of processes in the memory.

It is invoked less frequently.

A **short-term scheduler (CPU scheduler)** selects one of the processes in the memory and allocates the CPU to it.

The short-term scheduler is invoked frequently and should be very fast.

The long-term scheduler should select a proper mix of CPU-bound processes and i/o-bound processes.

A CPU-bound process spends most of its time doing computations.

An i/o-bound process spends most of its time doing i/o.

Some multitasking operating systems, like Unix and Windows, do not use long-term schedulers.

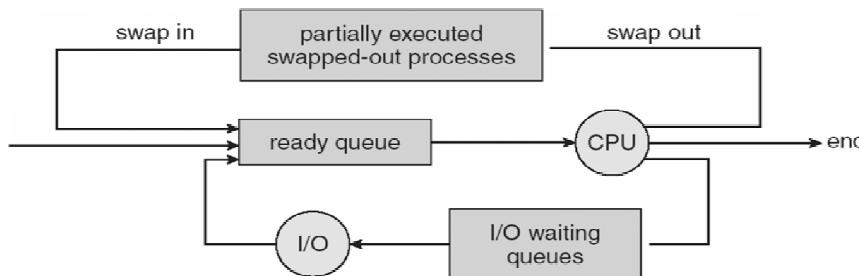
All new processes are put in the memory for the perusal of the short-term scheduler.

A **medium-term scheduler** removes processes from the memory and from the competition for the CPU, thus reducing the degree of multiprogramming.

The processes can be later reintroduced in the memory and their executions can be resumed.

This scheme is called **swapping**.

Swapping may be used to improve process mix and to free up some memory in uncontrollable circumstances.



**Context switching** is done to switch between processes.

Switching the CPU to another process requires saving the state of the current process and reloading the state of another process.

States are saved into and reloaded from PCBs.

Context-switch time is a pure overhead as the system does not do any useful work during a control switch.

Context-switch time depends highly on the hardware.

Context switching is faster on RISC processors with overlapped register windows.

**Process creation** is one process creating another process.

The processes are called parent process and child process, respectively.

Each process has a unique id.

A process may obtain resources either from its parent or from the operating system directly.

A parent process may continue executing with its children processes or may wait for them to complete.

A process may be a duplicate of its parent process (same code and data) or may have a new program loaded into it.

**Process termination** marks the deletion of the PCB of the process.

A parent process may terminate a child process –

- if it has exceeded its resource usage
- if its result is no more needed
- if the parent process is terminating and the operating system does not allow an **orphan process** (this may lead to cascading process terminations)

Typically, the kernel is the first process to be created, is the ancestor of all other processes and is at the root of the process tree.

A **zombie process** is a process that has terminated but its PCB still exists because its parent has not yet accepted its return value.

**Interprocess communication** –

- Reasons –
  - information sharing
  - computational speedup
  - modularity
  - convenience
- Models –
  - shared memory
  - message passing [send(P,message) and receive(id,message)]

A **thread** is the smallest sequence of instructions that can be managed independently by a scheduler.

A thread is a component of a process.

Multiple threads can exist within the same process, executing concurrently and share resources such as memory.

The threads of a process share its instructions (executable code) and its context (the values of its variables at any given moment).

**Difference between process and thread** –

- processes are typically independent while threads exist as parts of a process
- processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources
- processes have separate address spaces, whereas threads share their address space
- processes interact only through system-provided inter-process communication mechanisms
- context switching between threads in the same process is typically faster than context switching between processes

**Advantages of multi-threaded programming** –

- responsiveness
- faster execution
- better resource utilization
- easy communication
- parallelization

The execution of a process consists of alternate **CPU bursts** and **i/o bursts**, starting and ending with CPU bursts.

The CPU scheduler is invoked when a process –

- switches from running state to waiting state (condition 1)
- switches from running state to ready state (condition 2)
- switches from waiting state to ready state (condition 3)
- terminates (condition 4)

In **non-preemptive scheduling** or cooperating scheduling, a process keeps the CPU until it terminates or switches to the waiting state.

Some machines support non-preemptive scheduling only.

E.g. – Window 3.1x.

In **preemptive scheduling**, a process can be forced to leave the CPU and switch to the ready queue.

E.g. – Unix, Linux, Windows 95 and higher.

CPU scheduling is optional for conditions 2 and 3, but necessary in the other two conditions.

MS DOS does not support multiprogramming, hence no CPU scheduling.

A **dispatcher** is the module of the operating system that gives control of the CPU to the process selected by the CPU scheduler.

Steps –

- switching context
- switching to user mode
- jumping to the proper location in the user program

**Dispatch latency** is the time taken to stop a process and start another.

Dispatch latency is a pure overhead.

Scheduling criteria –

- $\uparrow$  **CPU utilization** - percentage
- $\uparrow$  **Throughput** - number of processes completed per unit time
- $\downarrow$  **Turnaround time** - time from submission to completion (time spent in different queues + time spent in CPU + time spent in different i/o devices)
- $\downarrow$  **Waiting time** - time spent in ready queue (only)
- $\downarrow$  **Response time** - time from submission to first response

Variance in response time should be minimal.

Gantt chart (Henry Gantt)

**Scheduling algorithms** are used to select a process for execution.

There are several well known scheduling algorithms.

**First-come first-served (FCFS) scheduling** –

- non-preemptive
- high average waiting time
- **convoy effect** - several small processes may need to wait if a large process is given the CPU

**Exercise 1.**

Process	Arrival time (ms)	CPU burst time (ms)
P1	0	24
P2	1	3
P3	2	3

Calculate throughput, average turnaround time and average waiting time.

**Shortest-job-first (SJF) scheduling –**

- process with the smallest next CPU burst is selected
- FCFS to break ties
- more appropriate term: shortest-next-CPU-burst-first
- optimal, but cannot be implemented

We may predict the length of the next CPU burst.

We use exponential average.

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, \quad \text{where } 0 \leq \alpha \leq 1.$$

If  $\alpha = 0$ , predicted length is constant.

If  $\alpha = 1$ , predicted length is equal to that of the last (actual) CPU burst.

Typically,  $\alpha = 0.5$ .

$\tau_0$  = constant or system average.

SJF can be either preemptive or non-preemptive.

Preemptive SJF = shortest-remaining-time-first.

**Exercise 2.**

Process	Arrival time (ms)	CPU burst time (ms)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Calculate throughput, average turnaround time and average waiting time for (a) non-preemptive and (b) preemptive scheduling.

**Priority scheduling –**

- CPU is allocated to the process with the highest priority
- priority range be 0 to 7 (say), with 0 representing the highest or the lowest priority
- priority may depend on internal factors (time limit, memory requirement, number of open files, etc.) and external factors (user, department, etc.)
- may be preemptive or non-preemptive
- SJF is a special case of priority scheduling, with priority inversely proportional to predicted next CPU burst length
- may cause starvation, *i.e.* indefinite blocking of processes
- **aging**: gradually increase the priority of a process waiting for a long time
- **priority inversion**: a low-priority process gets the priority of a high-priority process waiting for it

**Exercise 3.**

Process	Arrival time (ms)	CPU burst time (ms)	Priority
P1	0	8	7
P2	2	4	5
P3	4	6	0 (highest)
P4	6	4	1

*Dispatch latency = 1 ms.*

*Calculate CPU utilization, throughput, average turnaround time and average waiting time for (a) non-preemptive and (b) preemptive scheduling.*

**Round robin (RR) scheduling –**

- a small **time quantum** or time slice is defined
- the ready queue is treated as a circular queue
- each process is allocated the CPU for one time quantum
- preemptive
- if time quantum is too large, then RR behaves like FCFS
- if time quantum is too small, then RR behaves like processor sharing
- rule of thumb: 80% CPU bursts should be shorter than the time quantum

**Exercise 4.**

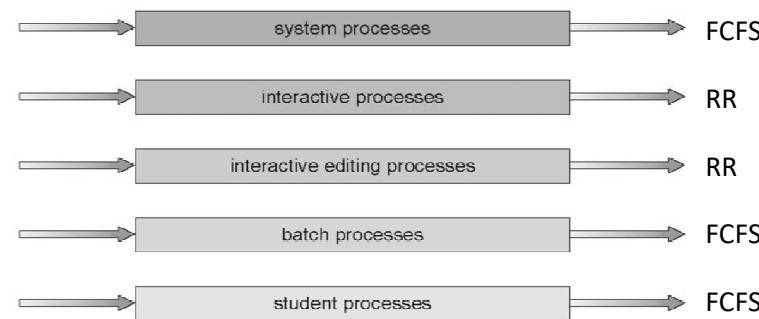
Process	Arrival time (ms)	CPU burst time (ms)
P1	0	18
P2	2	4
P3	4	6
P4	6	12

*Calculate throughput, average turnaround time and average waiting time for (a) time quantum = 8 ms and (b) time quantum = 2 ms.*

**Multilevel queue scheduling –**

- the ready queue is partitioned into several queues
- a process is permanently assigned to one queue
- each queue has its own scheduling algorithm
- inter-queue scheduling: preemptive priority scheduling or RR (80% time for foreground processes and 20% time for background processes)

highest priority

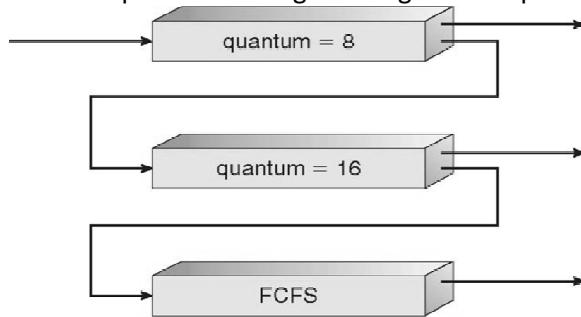


**Exercise 5.**

Process	Arrival time (ms)	CPU burst time (ms)	Queue
P1	0	18	Q2
P2	2	4	Q2
P3	4	6	Q1 (highest-priority)
P4	6	12	Q2

*Q1: FCFS**Q2: RR, time quantum = 2 ms**Inter-queue: preemptive priority scheduling**Calculate throughput, average turnaround time and average waiting time.***Multilevel feedback queue scheduling –**

- allows processes to move between queues
- inter-queue scheduling: preemptive priority scheduling
- a process waiting too long in a low-priority queue may be moved to a high-priority queue

**Exercise 6.**

Process	Arrival time (ms)	CPU burst time (ms)
P1	0	32
P2	2	10
P3	4	16
P4	6	30

*Calculate throughput, average turnaround time and average waiting time.***Comparison –**

	😊	😢
First come first served scheduling	Simple	Inefficient (high turnaround time, waiting time and response time)
Shortest job first scheduling	Most efficient	Impossible to implement
Priority scheduling	Low waiting time for high priority processes	Indefinite blocking of low priority processes
Round robin scheduling	Efficient and no indefinite blocking of processes	Too much context switching
Multilevel queue scheduling	Low waiting time for high priority processes	Complex
Multilevel feedback queue scheduling	Fast turnaround for short processes	Complex

**Multiple processor scheduling –**

- multiprocessor systems: homogeneous or heterogeneous
- CPU scheduling approaches
  - asymmetric multiprocessing      - one processor performs scheduling for all
  - symmetric multiprocessing      - each processor is self-scheduling
  - e.g. – Linux supports symmetric multiprocessing
- processor affinity –
  - because of high cost of invalidating and repopulating caches, migration of processes from one processor to another is avoided
  - attempt is made to keep a process at a given processor
  - hard- and soft- affinity
- load balancing –
  - attempt to keep all processors equally loaded
  - pull migration, push migration, combination
  - only for symmetric systems
  - counteracts the benefits of processor affinity

Chapters: 3 – 5.

Sections: 3.1 – 3.4, 4.1, 4.2 and 5.1 – 5.4.



## UNIT 3

### PROCESS SYNCHRONIZATION

A cooperating process is one that can affect or be affected by other processes executing in the system.

P0	P1
...	...
i++;	i--;
...	...
R1 $\leftarrow$ i	R2 $\leftarrow$ i
R1 $\leftarrow$ R1 + 1	R2 $\leftarrow$ R2 - 1
i $\leftarrow$ R1	i $\leftarrow$ R2

Initially, i = 10

P0: R1 = 10

P1: R2 = 10

P0: R1 = 11

P1: R2 = 9

P0: i = 11

P1: i = 9

**Race condition** is a situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the accesses take place.

To avoid such situations, it must be ensured that only one process can manipulate the data at a given time.

This can be done by **process synchronization**.

A **critical section problem** is defined as follows –

- there are n processes, viz. P<sub>0</sub>, P<sub>1</sub>, P<sub>2</sub> ... P<sub>n-1</sub>
- each process has a section of code, called the **critical section**, in which the process changes common variables and files
- the problem is to ensure that when one process is executing in its critical section then no other process can execute its own critical section
- the critical section is preceded by an **entry section** in which a process seeks permission from other processes
- the critical section is followed by an **exit section**
- the remaining code is called the **remainder section**

```
do { entry section
      critical section
      exit section
      remainder section
  } while (true);
```

A solution to the critical section problem must satisfy the following properties –

- **Mutual exclusion**
- **Progress** - If no process is in its critical section and some processes want to enter their critical sections, then the processes which are in their entry sections or exit sections decide which process will enter its critical section next. This selection cannot be postponed indefinitely.
- **Bounded waiting** - There is a limit on the number of times other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**Peterson's solution** (2 processes only) –

Shared variables:

```
int turn;
Boolean flag[2];
```

Code for  $P_i$ :

```
do { flag[i] = true;
      turn = j;
      while (flag[j] && turn == j)
      ;
      critical section
      flag[i] = false;
      remainder section
} while (true);
```

Prove: Peterson's solution satisfy the three properties.

The critical section problem can be also solved by dedicated hardware.

A **semaphore** is an integer variable that, apart from initialization, is accessed only through two atomic operations called `wait()` and `signal()`.

<b>wait (S)</b>	<b>signal (S)</b>
{while (S <= 0)	{S++;
; // busy waiting	}
S--;	
}	

Only one process can access a semaphore at a time.

**Binary semaphore** –

- also called mutex lock
- used to implement solution of critical section problem with multiple processes
- initialized to 1

```
do { wait (mutex);
      critical section
      signal (mutex);
      remainder section
} while (true);
```

**Counting semaphore** –

- used to control access to a resource that has multiple instances
- initialized to n

Semaphores may lead to indefinite wait (starvation) and deadlocks.

P0	P1
wait (S);	wait (Q);
wait (Q);	wait (S);
...	...
signal (S);	signal (Q);
signal (Q);	signal (S);

### Producers – consumers problem (bounded buffer problem)

Number of buffers = n

Semaphores –

```
mutex = 1; //access to buffers
empty = n;
full = 0;
```

Producers

```
do { // produce an item
    wait (empty);
    wait (mutex);
    // add the item to the buffer
    signal (mutex);
    signal (full);
} while (true);
```

Consumers

```
do { wait (full);
    wait (mutex);
    // remove an item from buffer
    signal (mutex);
    signal (empty);
    // consume the item
} while (true);
```

### Readers – writers problem

Shared data –

```
int readcount = 0;
```

Semaphores –

```
mutex = 1;
wrt = 1;
```

Writers

```
do { wait (wrt);
    // write
    signal (wrt);
} while (true);
```

Readers

```
do { wait (mutex);
    readcount++;
    if (readcount == 1)
        wait (wrt);
    signal (mutex);
    // read
    wait (mutex);
    readcount--;
    if (readcount == 0)
        signal (wrt);
    signal (mutex);
} while (true);
```

### Dining philosophers problem



Semaphores –

```
chopsticks[5];
all initialized to 1
```

Pi

```
do { wait (chopstick[i]);
     wait (chopstick[(i+1)%5]);
     // eat
     signal (chopstick[i]);
     signal (chopstick[(i+1)%5]);
     // think
 } while (true);
```

If all philosophers get hungry simultaneously then there will be a deadlock.

A **monitor** is a high-level process synchronization construct.

Only one process can be active within the monitor at a time.

A monitor type presents a set of programmer defined operations that are provided mutual exclusion within the monitor.

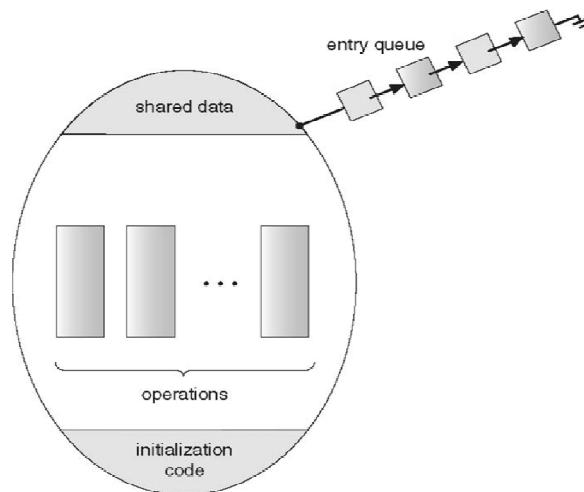
A monitor can have variables of the condition type that can be accessed by wait() and signal() operations only.

The operation x.wait() means that the process making this operation is suspended until another process invokes x.signal().

The operation x.signal() resumes exactly one suspended process and has no effect if there is none.

Syntax:

```
monitor monitor_name
{
    // shared variable declarations
    initialization_code (...) { ... }
    procedure P1 (...) { ... }
    procedure Pn (...) { ... }
}
```



Solution of the dining philosophers problem using a monitor:

```
monitor dp
{
    enum {THINKING, HUNGRY, EATING} state[5];
    condition self [5];
```

```
initialization_code()
{for (int i=0; i < 5; i++)
 state[i] = THINKING;
}

void test (int i)
{if ((state[(i+4)%5]!=EATING)&&
 (state[i]==HUNGRY)&&
 (state[(i+1)%5]!=EATING)]
    {state[i] = EATING ;
     self[i].signal();
    }
}

void pickup (int i)
{state[i] = HUNGRY;
 test(i);
 if (state[i]!=EATING)
    self[i].wait;
}

void putdown (int i)
{state[i] = THINKING;
 test((i+4)%5); // test left neighbor
 test((i+1)%5); // test right neighbor
}

}
```

```
Pi
...
dp.pickup(i);
...
// eat
...
dp.putdown(i);
...
```

Chapter: 6.  
Sections: 6.1 – 6.7.



## UNIT 4

### DEADLOCKS

In a multiprogramming system, several processes may compete for a finite number of resources. A process requests for resources, and if the resources are not available at the time then the process enters the waiting state.

Sometimes, a process will wait indefinitely because the resources it has requested for are being held by other similar waiting processes.

**Deadlock** is a situation in which two or more processes are waiting indefinitely because the resources they have requested for are being held by one another.

The resources are partitioned into several types, each of which has several identical instances.

E.g. – memory spaces, registers, i/o devices, etc.

If a process requests for an instance of the resource type, then the allocation of any one instance of the resource type will satisfy the process.

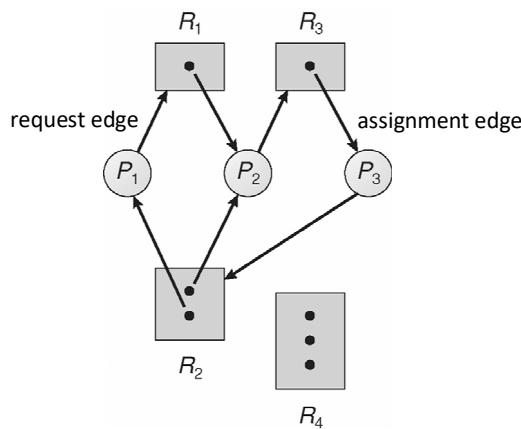
A process utilizes a resource in the following sequence: request → use → release.

#### Necessary conditions for deadlocks –

- **Mutual exclusion** - one or more non-sharable resources
- **Hold and wait** - a process is holding some resources and waiting for other resources
- **No preemption** - resources cannot be preempted
- **Circular wait** - a set  $\{P_0, P_1, P_2 \dots P_n\}$  exist such that  $P_0$  is waiting for resources held by  $P_1$ ,  $P_1$  is waiting for resources held by  $P_2$ , and so on, and  $P_n$  is waiting for resources held by  $P_0$

Circular wait implies hold and wait.

#### Resource allocation graph –



If there is a cycle, then there may be a deadlock (necessary but not sufficient condition).

If there is no cycle, then there is not deadlock.

If each resource type has one instance, then a cycle implies that there is a deadlock (necessary and sufficient condition).

If each resource type in a cycle has one instance, then there is a deadlock (necessary and sufficient condition).

#### Methods for handling deadlocks –

- **Deadlock prevention** - Ensure that at least one of the necessary conditions for occurrence of deadlocks cannot hold.

- **Deadlock avoidance**

- Each process informs the operating system about the resources it will require in its lifetime. The operating system allocates the resources to the processes in a sequence that will not lead to a deadlock.

- **Deadlock detection and recovery**

No deadlock handling mechanism –

- deadlocks are infrequent, may be only once in a year
- deadlock handling is expensive
- e.g. – Windows, Linux, etc.

Deadlock prevention –

1. Mutual exclusion

- Mutual exclusion is necessary for non-sharable resources, like printer and speaker.
- Mutual exclusion can be prevented from sharable resources, like read-only files.

2. Hold and wait

- Ensure that when a process requests for resources it is not holding some other resources.
- Protocol 1: Request and get all the resources in the beginning.
- Protocol 2: Release current resources before requesting other resources.
- Disadvantages – low resource utilization, starvation of processes requiring several resources.

3. No preemption

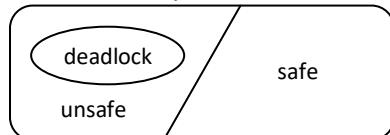
- If a process requests for some resources and they cannot be allocated right now, then the resources that the process is holding are preempted.
- Resources that can be saved and later restored – registers and files.
- Resources that cannot be preempted – printer.

4. Circular wait

- Arrange the resource types as R1, R2, R3 ... Rm.
- Protocol 1: Request resources in increasing order.
- Protocol 2: If a process requests for R<sub>i</sub>, then it must release R<sub>j</sub> for all  $j \geq i$ .
- All instances of a resource type should be allocated together.
- Prove: The protocols can prevent deadlock.
- Disadvantages – low resource utilization, reduced throughput.

Deadlock avoidance –

- A **safe state** is one in which the system can allocate resources to each process up to the maximum in some order and still avoid deadlock.
- A system is in a **safe state** if there is a safe sequence.
- A sequence of processes  $\langle P_1, P_2, P_3 \dots P_n \rangle$  is safe if for each  $P_i$  the resources that  $P_i$  may still request can be satisfied by the currently available resources plus all resources held by all  $P_j$  with  $j < i$ .
- $P_i$  may need to wait for one or more  $P_j$ s.



The **banker's algorithm** is used to implement deadlock avoidance.

This algorithm can be used in a bank to ensure that the bank never allocates the available money in a way that it could no longer satisfy the needs of all its clients.

A new process must declare the maximum number of instances of each resource type that it may need.

This number should not exceed the total number of instances of that resource type in the system.

When a process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state.

If yes, then the resources may be allocated to the process.

If no, then the process must wait till other processes release enough resources.

n processes

m resource types

Data structures –

- Available[m] - currently available instances
- Max[nxm] - maximum demand
- Allocation[nxm] - currently allocated
- Need[nxm] = Max - Allocation

These data structures may vary in size with time.

For Pi: Max[i], Allocation[i], Need[i].

**Safety algorithm** checks if the system is in a safe state.

1. Let Work[m] and Finish[n] be vectors  
Work = Available  
Finish[i] = false, for i = 0, 1, 2 ... n-1
2. Find an i such that (Finish[i] == false && Need[i] <= Work)  
If no such i exists then go to step 4
3. Work = Work + Allocation[i]  
Finish[i] = true  
Go to step 2
4. If (Finish[i] == true) for all i = 0, 1, 2 ... n-1, then it is a safe state else it is an unsafe state

Time complexity =  $O(mxn^2)$ .

**Resource-request algorithm** checks if a request can be safely granted.

Pi is requesting for more resources and Request[m] be the request.

1. If Request > Need[i], then error
2. If Request > Available, then wait
3. Pretend to allocate the request  
Available = Available - Request  
Allocation[i] = Allocation[i] + Request  
Need[i] = Need[i] - Request

If the resultant state is safe then the resources are actually allocated, else values of Available, Allocation[i] and Need[i] are restored to their previous values.

Time complexity =  $O(m)$ .

*Exercise 1.*

*A system has four processes, viz. P1 to P4, and three resource types. The system is in a safe state with*

*Available = [1 1 1], Max =  $\begin{bmatrix} 0 & 1 & 1 \\ 6 & 2 & 4 \\ 4 & 3 & 5 \\ 1 & 2 & 1 \end{bmatrix}$  and Allocation =  $\begin{bmatrix} 0 & 0 & 1 \\ 4 & 2 & 3 \\ 3 & 3 & 3 \\ 0 & 2 & 1 \end{bmatrix}$ . Which of the following requests should be granted?*

*(a) Request = [1 0 1] from P3.*

*(b) Request = [1 0 1] from P2.*

*Solution.*

(a) Resource-request algorithm -

$$\text{Request} = [1 \ 0 \ 1], \text{ Available} = [1 \ 1 \ 1] \text{ and Need} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

Since  $\text{Request} < \text{Need}[3]$  and  $\text{Request} < \text{Available}$ , pretend to allocate resources

$$\text{Available} = [0 \ 1 \ 0], \text{ Allocation} = \begin{bmatrix} 0 & 0 & 1 \\ 4 & 2 & 3 \\ 4 & 3 & 4 \\ 0 & 2 & 1 \end{bmatrix}, \text{ and Need} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Safety algorithm -

Initially - Work: [0 1 0] Finish: [FFF F]

P1 selected - Work: [0 1 1] Finish: [TFFF F]

P3 selected - Work: [4 4 5] Finish: [TFTT F]

P4 selected - Work: [4 6 6] Finish: [TFTT T]

P2 selected - Work: [8 8 9] Finish: [TTTT T]

Safe sequence: <P1, P3, P4, P2>

Safe state: Request should be granted.

(b) Resource-request algorithm -

$$\text{Request} = [1 \ 0 \ 1], \text{ Available} = [1 \ 1 \ 1] \text{ and Need} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

Since  $\text{Request} < \text{Need}[2]$  and  $\text{Request} < \text{Available}$ , pretend to allocate resources

$$\text{Available} = [0 \ 1 \ 0], \text{ Allocation} = \begin{bmatrix} 0 & 0 & 1 \\ 5 & 2 & 4 \\ 3 & 3 & 3 \\ 0 & 2 & 1 \end{bmatrix}, \text{ and Need} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

Safety algorithm -

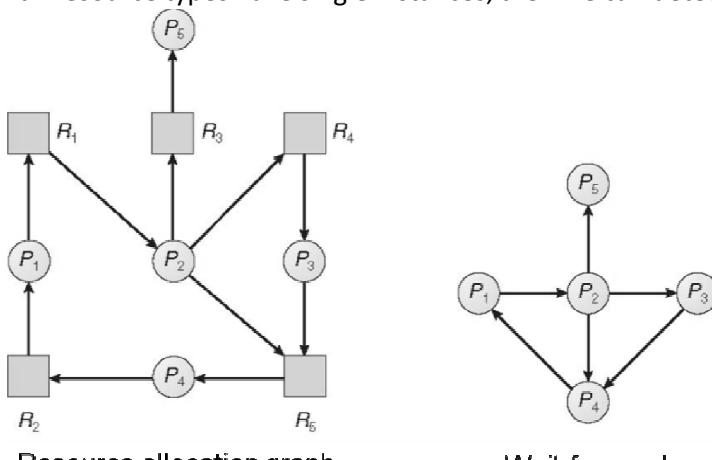
Initially - Work: [0 1 0] Finish: [FFF F]

P1 selected - Work: [0 1 1] Finish: [TFFF F]

Unsafe state: Request should not be granted.

Deadlock detection -

If all resource types have single instances, then we can detect deadlocks using a wait-for graph.



A cycle in the wait-for graph denotes that there is a deadlock and all the processes in the cycle are deadlocked.

Time complexity of cycle detection algorithm =  $O(n^2)$ .

A modification of the banker's algorithm can be used to detect deadlock in a system that has resource types with multiple instances.

Request[nxm] represents the requests.

1. Let Work[m] and Finish[n] be vectors

Work = Available

For  $i = 0, 1, 2 \dots n-1$ , if ( $\text{Allocation}[i] \neq 0$ ) then  $\text{Finish}[i] = \text{false}$  else  $\text{Finish}[i] = \text{true}$

2. Find an  $i$  such that ( $\text{Finish}[i] == \text{false} \ \&\& \ \text{Request}[i] \leq \text{Work}$ )

If no such  $i$  exists, then go to step 4

3.  $\text{Work} = \text{Work} + \text{Allocation}[i]$

$\text{Finish}[i] = \text{true}$

Go to step 2

4. If  $\text{Finish}[i] == \text{false}$  for some  $i$ , then there is a deadlock and  $P_i$  is deadlocked

Time complexity =  $O(mxn^2)$ .

How often to invoke deadlock detection algorithm?

- after fixed time interval
- when CPU utilization falls

Recovery from a deadlock –

- Process termination
  - abort all deadlocked processes
  - abort one process at a time until the deadlock is broken
- Resources preemption
  - selecting a victim
    - rollback
    - starvation

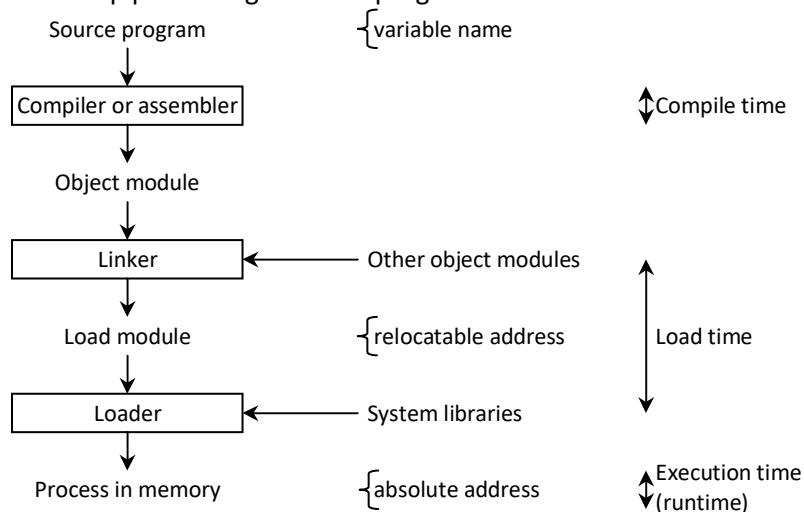
Chapter: 7.  
Sections: 7.1 – 7.7.



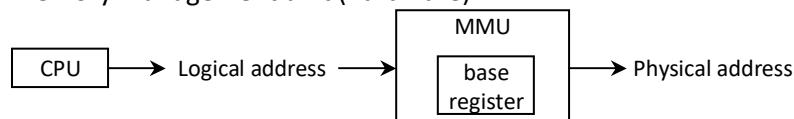
## UNIT 5

### MEMORY MANAGEMENT

Multistep processing of a user program –



Memory management unit (hardware) –



Swapping –

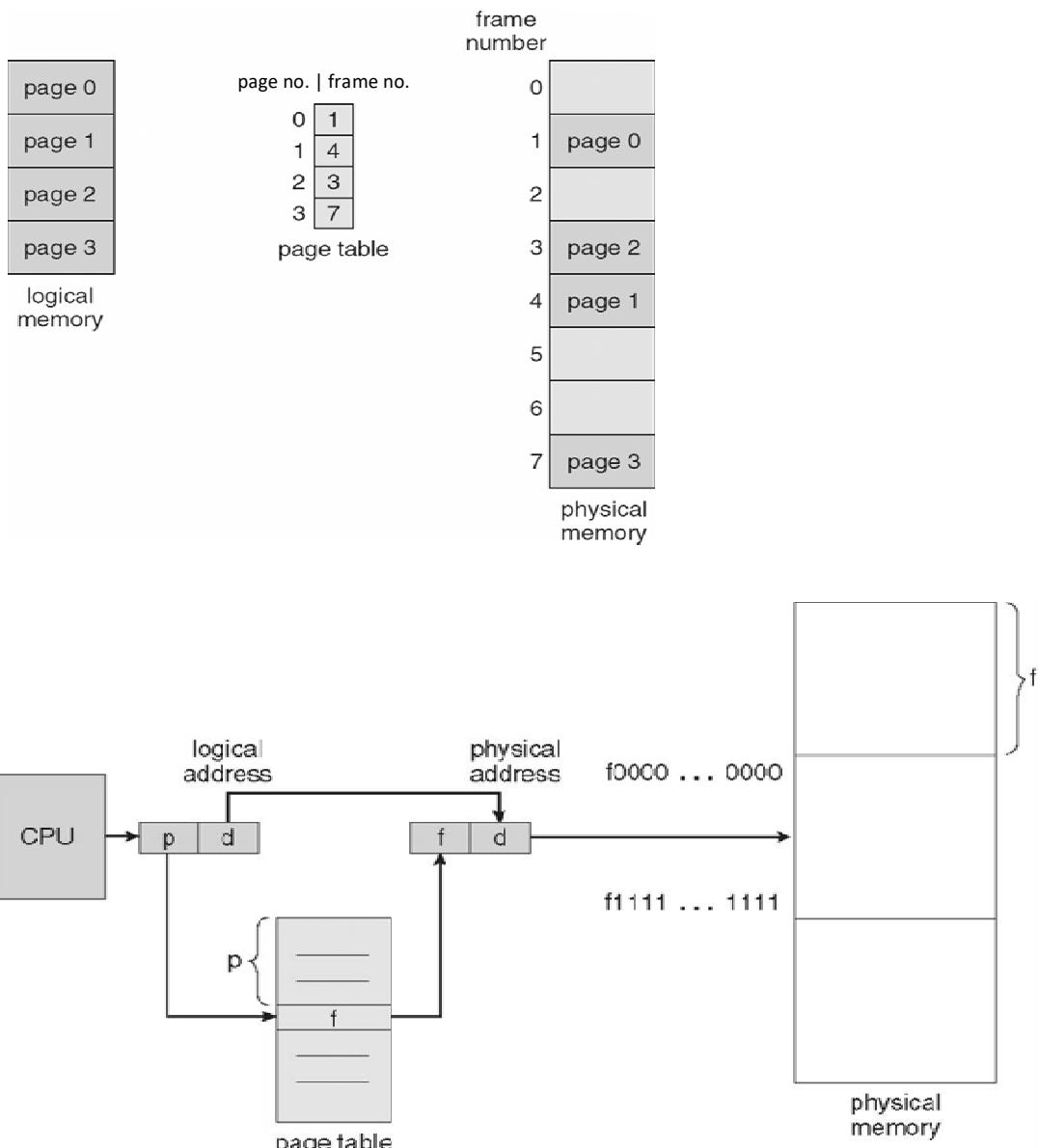
- swap-out, swap-in
- priority scheduling: roll-out, roll-in
- relocation is helpful

Contiguous memory allocation –

- Static: divide the memory into fixed-sized blocks
  - allocate one block to each process
  - number of partitions controls the degree of multiprogramming
- Dynamic: allocate available space to a process
  - Keep track of holes
  - strategies: **first-fit**, **best-fit**, **worst-fit**
  - the worst-fit strategy results in large leftover holes (reusable)
- Contiguous memory allocation leads to **external fragmentation**
  - there may be enough memory left to accommodate a process but it is scattered
  - solution: **compaction** (costly)

**Paging** is a non-contiguous memory allocation scheme

Page size = frame size



Advantages –

- size of process independent of page size
- no external fragmentation

Disadvantage –

- **internal fragmentation**

Research topic –

- dynamic page size

Each process is allocated a page table.

The page table is stored in the memory.

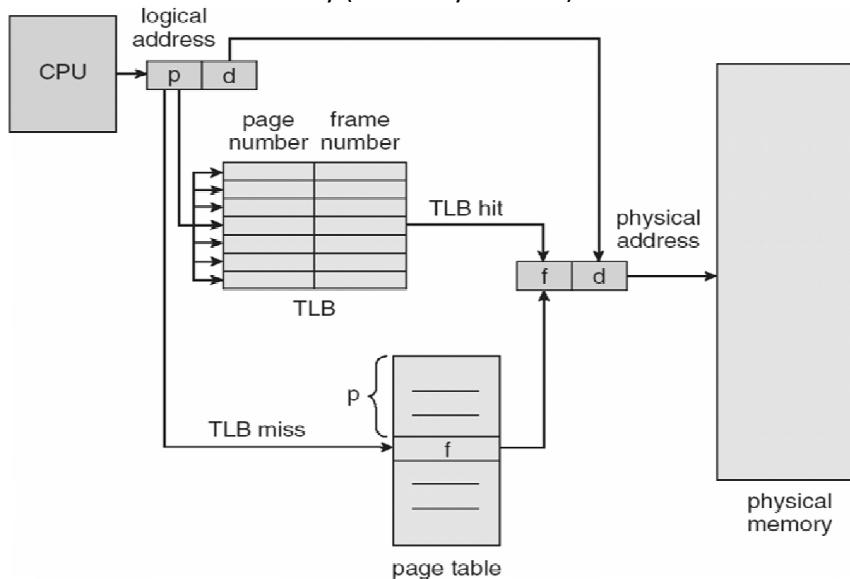
A **page table base register** (PTBR) points to the page table.

Changing the page table requires just changing the value of the register, thus reducing the context switch time.

One read/write operation requires two memory accesses.

**Transition look-aside buffer (TLB) –**

- dedicated and fast (hardware)
- associative memory (search by content)



$$\begin{aligned} \text{Effective access time} = & \text{hit ratio} \times (\text{TLB access time} + \text{memory access time}) \\ & + (1 - \text{hit ratio}) \times (\text{TLB access time} + 2 \times \text{memory access time}) \end{aligned}$$

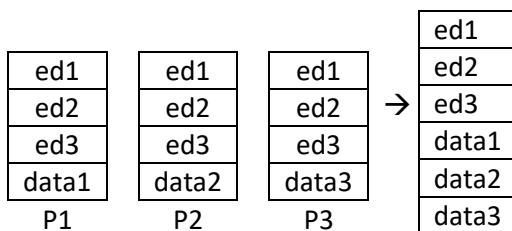
Each page table entry may have –

- a read-only bit
- a valid-invalid bit (invalid denotes that the page belongs to another process)

Shared pages

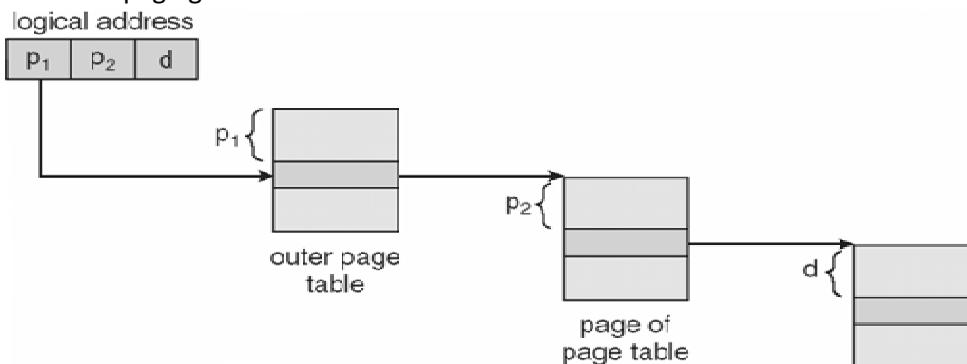
Reentrant code = pure code = non-self-modifying code = not changed during execution.

Pages common to multiple processes can be shared if they contain reentrant code.



Hierarchical page tables

Two-level paging:



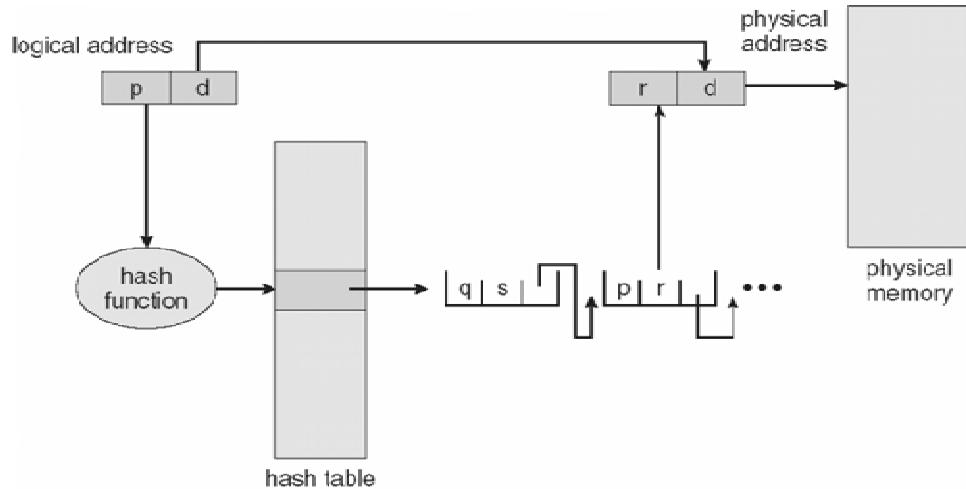
Three-level paging: <p<sub>1</sub>, p<sub>2</sub>, p<sub>3</sub>, d> - second outer page table, first outer page table, inner page table.  
Too many memory accesses.

### Hashed page table

A hashed value is used as a virtual page number.

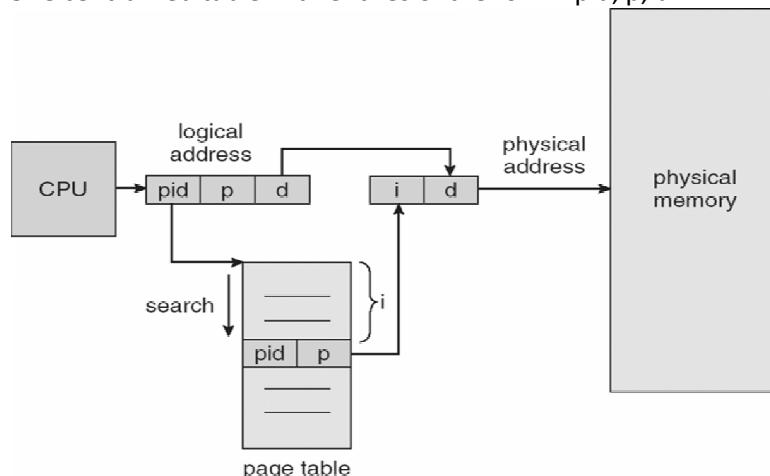
Each entry in the hash table is a linked list of elements that hash to the same location (to avoid collisions).

Each element has three fields: virtual page number, frame number, pointer to the next element in the list.



### Inverted page table

One centralized table with entries of the form <pid, p, d>



Difficult to implement shared pages.

**Segmentation** is a memory management scheme where the linear memory is logically partitioned into segments each having unique purpose.

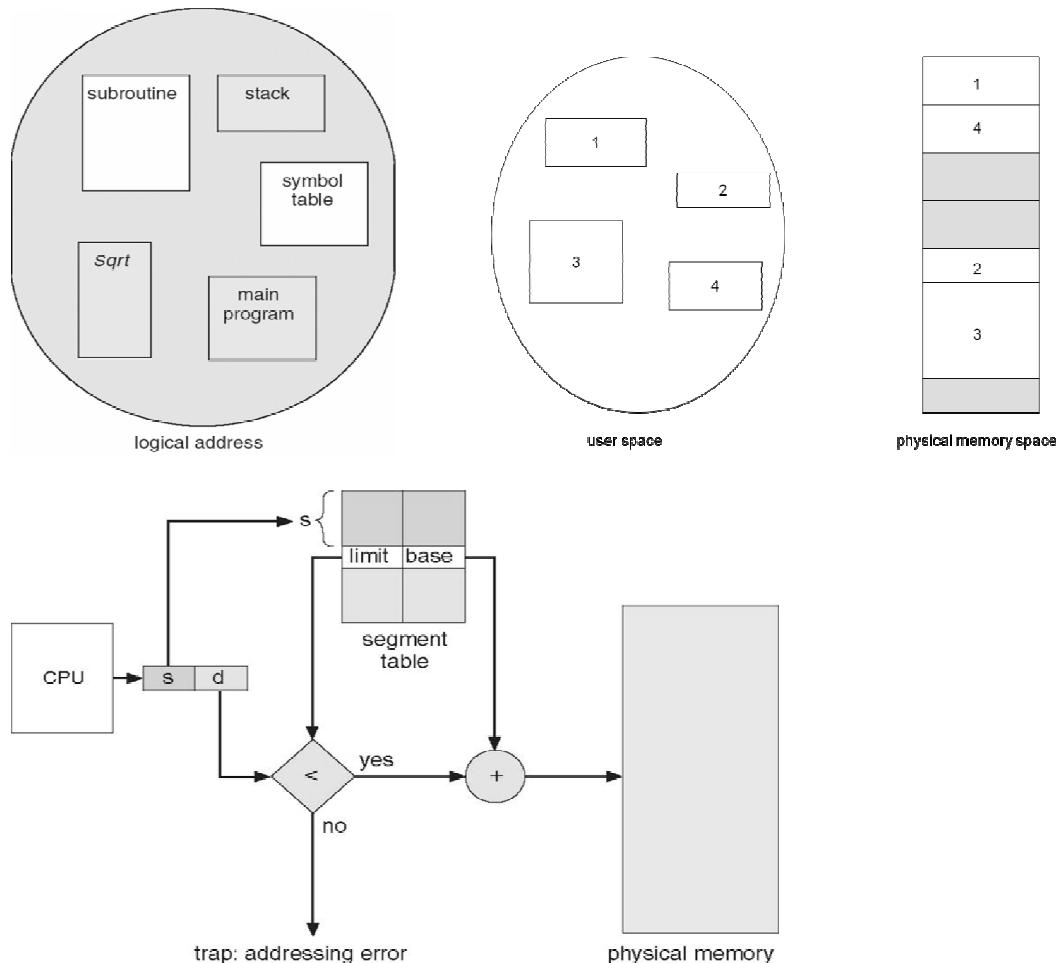
Address: <segment\_number, offset>.

A process may have segments each storing a part of code or data.

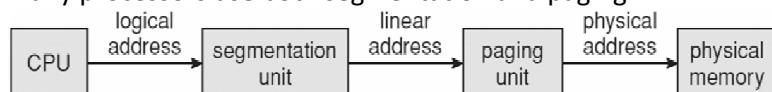
A typical program has the following segments –

- code (CS)
- data (DS)
- stack (SS)
- extra (ES)

There is typically hardware support to use a limited number of segments at a given time.



Many processors use both segmentation and paging.



E.g. – Intel Pentium

- uses segmentation with paging
- up to 16 K segments per process
- a process can access up to 6 segments at a time
- a segment can be of up to 4 GB
- pages size can be between 4 KB and 4 MB

Chapter: 8.  
Sections: 8.1 – 8.6.



## UNIT 6

### VIRTUAL MEMORY MANAGEMENT

**Virtual memory** is a technique that allows the execution of processes that are not completely in memory.

This allows execution of programs larger than the memory.

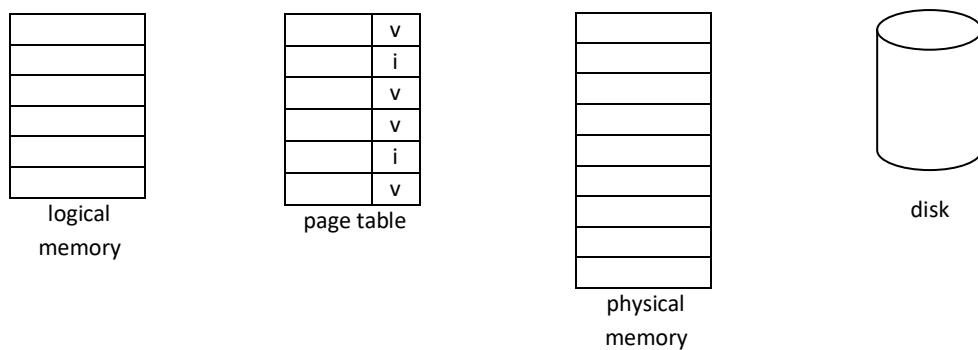
This allows higher degree of multiprogramming.

**Demand paging** is an approach to implement virtual memory.

Pages are loaded into the memory only when the CPU wants to access them.

Pages that the CPU does not want to access are not loaded.

We use a program called lazy swapper; also known as pager.



v (valid): page is in memory

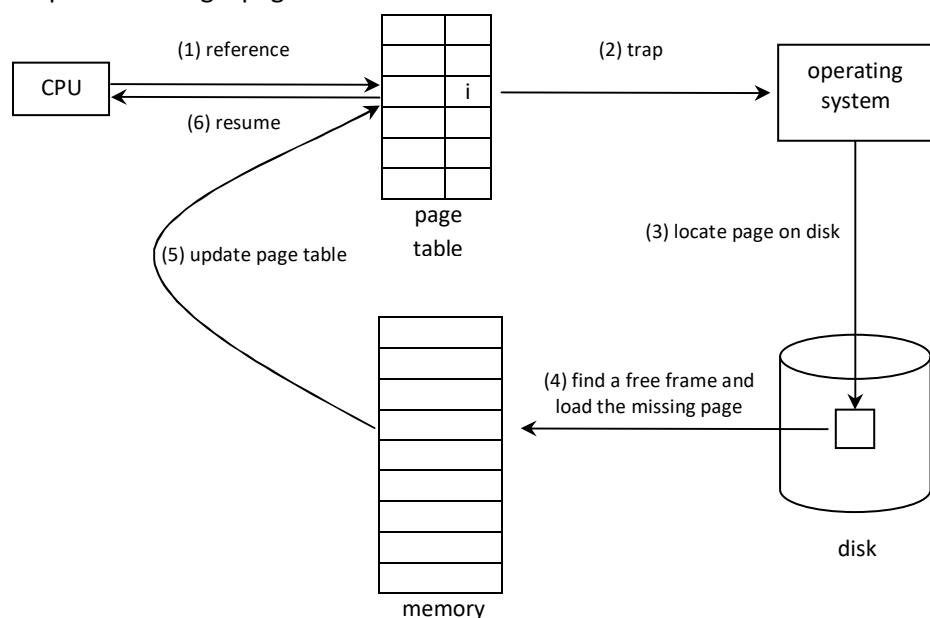
i (invalid): page is not in memory

If the CPU wants to access a memory-resident page, then the execution continues normally.

If the CPU wants to access a page that is not in memory, then a **page fault** trap occurs.

(trap: highest priority non-maskable external-hardware interrupt)

Steps in handling a page fault –



**Pure demand paging** is an extreme case of demand paging where the execution of a process is started with none of its pages in memory.

Starting is fast and response time is low.

Takes advantage of locality of references for both code and data.

Hardware support for virtual memory (already present) –

- page table
- disk

Let, page fault ratio =  $p$ .

Typically,  $0 < p < 1$  with  $p \rightarrow 0$  actually.

Effective access time =  $(1-p) \times \text{memory access time}$   
 $+ p \times \text{page fault service time}$

**Page replacement** is the process of replacing one page by another in the memory when there is no free frame.

How to load a page in memory when there is no free frame?

1. Save the content of the page currently in memory on disk.
2. Load new page.
3. Update page table.

Avoid saving unmodified pages –

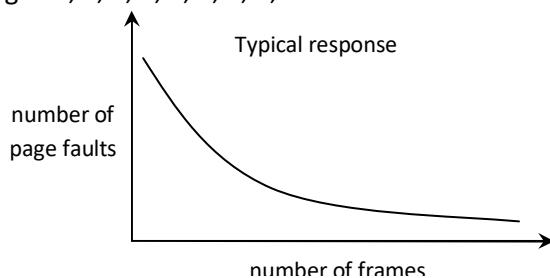
- use modify bit / dirty bit

We require **page replacement algorithms** and **frame allocation algorithms**.

We evaluate an algorithm by running it on a particular string of memory references and calculating the number of page faults.

This string of memory reference is called a **reference string** and is artificially generated by a random number generator.

E.g. – 0, 1, 0, 3, 5, 6, 1, 0, ...



Page replacement algorithms –

### 1. FIFO page replacement algorithm

- no need to note time, only a queue is required
- simple, but performance not good
- shows Belady's anomaly
- e.g. –
  - reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.
  - frames = 3.
  - page faults = 15.

### Belady's anomaly

Number of page faults increases with number of frames allocated.

- e.g. -
  - reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
  - frames = 3, page faults = 9.
  - frames = 4, page faults = 10.

### 2. Optimal page replacement algorithm

- replace the page that will not be used for the longest period of time
- lowest page fault rate, no Belady's anomaly
- needs future knowledge of reference string, hence, impossible to implement
- used for comparison
- e.g. (same) – 9 page faults.

### 3. Least-recently used (LRU) page replacement algorithm

- replace the least-recently used page
- no Belady's anomaly
- implemented using counters or a stack
- e.g. (same) – 12 page faults.

Implementation of LRU page replacement algorithm using counters –

- the page table has a 'time of use' field
- there is a logical clock which is incremented after every memory reference
- when a reference is made to a page, its 'time of use' is set to the current reading of the logical clock
- the LRU page has the minimum 'time of use' value, search and replace it
- logical clock must not overflow

Implementation of LRU page replacement algorithm using a stack –

- maintain a stack of page numbers referenced
- when a page is referenced, remove it from the stack and put it on the top
- the LRU page is at the bottom, no need to search
- suitable for software/microcode implementation

LRU page replacement algorithm requires substantial hardware support.

A **stack algorithm** does not suffer from Belady's anomaly.

A page replacement algorithm is a stack algorithm if we can show that the pages in memory for  $n$  frames is always a subset of pages in memory that will be there in memory for  $(n+1)$  frames.

For LRU page replacement algorithm –

- for  $n$  frames:  $n$  most recently used pages are in memory
- for  $(n+1)$  frames:  $(n+1)$  most recently used pages are in memory
- hence, it is a stack algorithm

### 4. LRU approximation page replacement algorithms

- most systems do not provide support necessary for LRU page replacement algorithm
- however, in most systems there is a **reference bit** for each page table entry
- initially, the reference bit is cleared to 0
- if the page is referenced, then the reference bit is set to 1

#### 4a. Additional-reference-bits page replacement algorithm

- we have a 8-bit shift register for each page table entry
- after fixed time intervals, the shift registers are right-shifted with the reference bit filling the msb
- the LRU page will have the minimum shift register value
- shift register value may not be unique – swap out all tied pages or use FIFO to break the tie

#### 4b. Second chance page replacement algorithm

- it is like FIFO
- however, if reference bit == 1, do not replace page and clear the reference bit
- frequently used pages will never be replaced
- e.g. (same) – 11 page faults.

#### 4c. Enhanced second chance page replacement algorithm

- principle: modified pages are not good candidate for replacement
- use (reference bit, modify bit) as an ordered pair
- classes: (0,0), (0,1), (1,0) and (1,1)
- replace a page of the lowest non-empty class by FIFO
- may require multiple iterations of the queue

### 5. Counting based page replacement algorithms

- a counter for each page
- **least frequently used (LFU) page replacement algorithm**
- **most frequently used (MFU) page replacement algorithm**
- expensive but performance not good

Allocation of frames –

Minimum number of frames – otherwise too many page faults

Frame allocation algorithms –

- **equal allocation**
- **proportional allocation** (proportional to the virtual memory of the process)

**Local page replacement** – a process can control its page fault rate

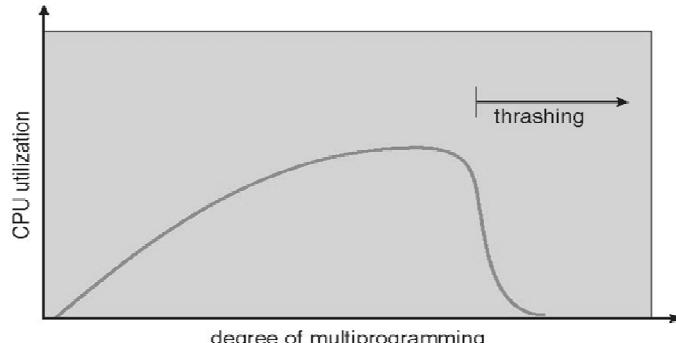
**Global page replacement** – better throughput, commonly used

**Thrashing** is a situation when a system is spending more time in servicing page faults than executing. If a process does not have the number of frames it needs to store the pages in active use, then it will page fault frequently.

Since all the pages in memory are in active use, the page that will be replaced will be needed again. This will bring down the CPU utilization and the system will load more processes in the memory in order to increase the degree of multiprogramming.

This will trigger a chain reaction of page faults.

This situation when the CPU utilization diminishes due to high paging activity is called thrashing.

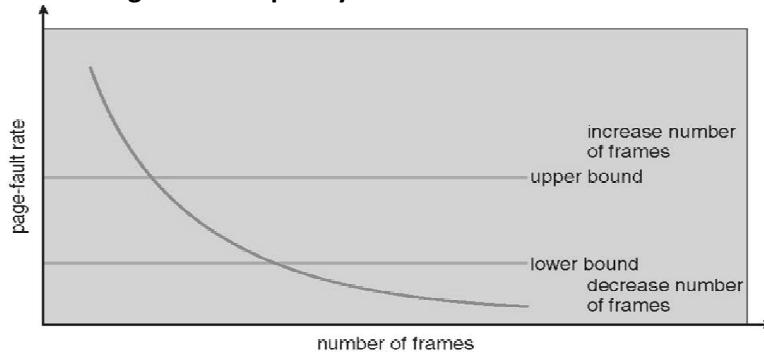


Avoiding thrashing –

- **Working-set model**

- based on locality of references
- working set of a process = pages referenced in  $\Delta$  most recent references
- e.g. –
  - $\Delta = 10$ .
  - reference string: 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 3 4 3 3 4 4 4 3 3.
- $WSS_i$  = working-set size of  $P_i$
- total demand of frames,  $D = \sum WSS_i$
- let,  $m$  = number of frames
- if  $D > m$ , then there will be thrashing
- we swap out one or more processes
- this keeps degree of multiprogramming as high as possible and optimizes CPU utilization
- keeping track of working-sets is difficult

- **Page fault frequency**



Chapter: 9.  
Sections: 9.1, 9.2 and 9.4 – 9.6.



## UNIT 7

### STORAGE MANAGEMENT

Hard disks are the most common on-line storage medium.

The operating system abstracts from the physical properties of its storage devices to define a logical storage, the **file**.

A file is a named collection of related information that is recorded on secondary storage.

A file is the smallest allotment of logical secondary storage.

Data cannot be written to secondary storage unless they are within a file.

Files store both code and data.

File formats – magic number.

To provide efficient and convenient access to the disk, the operating system imposes a **file system** to allow data to be stored, located and retrieved easily.

Design issues –

- how the file system should look to the user?
- how to map logical files onto the physical disk?

The file system is typically implemented as a multi-leveled system.

Application programs



Logical file system [Manages metadata]



logical block address

File organization module [Allocation management, free space management]



physical block address

Basic file system



high-level generic command

I/o control

[Device driver, interrupt]



low-level device specific command

Disk

Example of disk address: drive 1, track 2, sector 1.

A **directory** is a logical construct representing a set of files and subdirectories.

Implementing a file system –

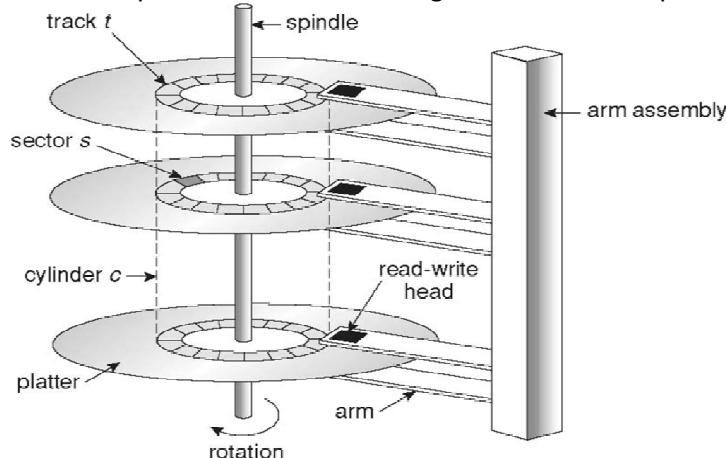
- **boot control block** (volume) - information needed to boot an operating system from that volume, if there is one
- **volume control block** (volume) - information about the volume
- **master file table** (volume) – information about the directory structure
- **file control block** (file) - information about the file

Allocation methods –

- contiguous
- linked
- indexed

Free space management – maintain a free-space-list

Hard disks provide the bulk of storage for modern computers.



Each disk platter has a flat circular shape.

Bits are stored magnetically on the platters.

A read-write head flies over a platter.

The arm moves the heads together.

The surface of the platter is divided into circular **tracks**.

Tracks are subdivided into **sectors**.

The set of tracks at one arm position consist a **cylinder**.

There are thousands of cylinders in a disk and hundreds of sectors in a track.

The outermost cylinder is denoted cylinder 0.

A motor rotates the disk at a high speed.

**Transfer rate:** rate at which data is transferred between the bus and the disk.

**Seek time:** time taken to move the arm to the cylinder.

**Rotational latency:** time taken to rotate the disk so that the head comes over the sector.

**Positioning time** (random access time) = seek time + rotational latency.

Performance of a disk depends on transfer rate and positioning time.

Disks can be addressed as a large one-dimensional array of **logical blocks**.

A block is the smallest unit of transfer.

The one-dimensional array of logical blocks has to be mapped onto the physical sectors.

Sequence –

- cylinder (outermost → innermost)
- track (top → bottom)
- sector

Storage –

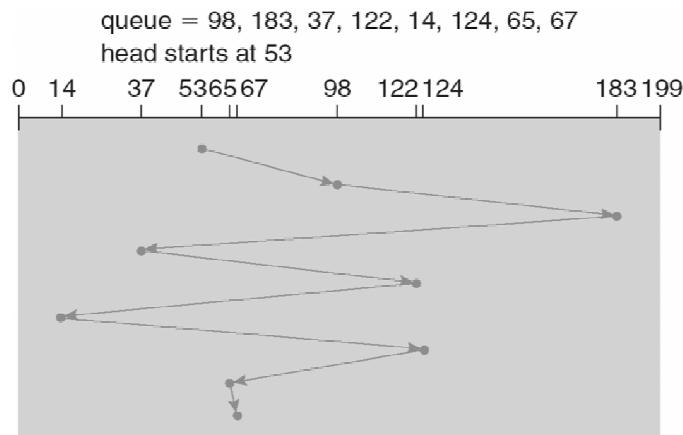
- uniform bit density - more sectors on outer tracks
- non-uniform bit density - same number of sectors on all tracks

Bandwidth of a disk = number of bytes transferred / time elapsed between first request and last service

**Disk scheduling** – try to improve access time and bandwidth

**Disk scheduling algorithms** –

- **First come first served (FCFS) scheduling**
- **Shortest-seek-time-first (SSTF) scheduling**
- **SCAN scheduling (elevator algorithm)**
- **C-SCAN scheduling**
- **LOOK scheduling**
- **C-LOOK scheduling**



Disk formatting –

- physical formatting or low-level formatting
  - fills the disk with a special data structure for each sector
  - the data structure has header, data area and trailer
  - contains sector number, error correcting code, etc.
- logical formatting
  - creates a file system

To increase efficiency, some file systems group blocks to form chunks.

Some operating systems allow some programs to perform raw I/O, e.g. – large databases.

Bad blocks are formed when some sectors become defective.

Logical formatting informs the file system to ignore them.

Chapters: 10 to 12.

Sections: 10.1, 10.3, 11.1, 11.2, 11.4, 11.5, 12.1, 12.2, 12.4 and 12.5.



## UNIT 8

### (A) I/O MANAGEMENT AND (B) SYSTEM PROTECTION AND SECURITY

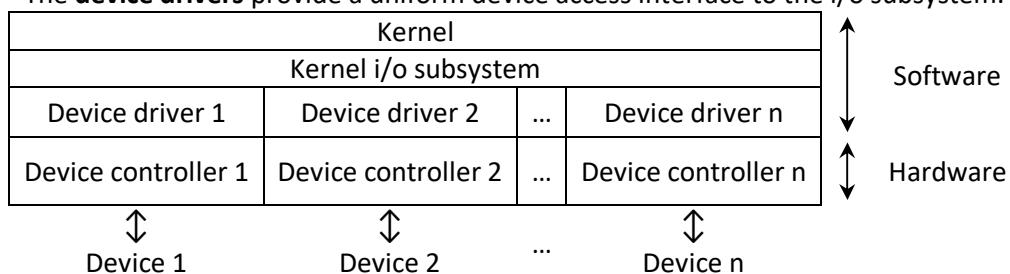
Two main responsibilities of a computer: processing and i/o.

Some processes perform more i/o and minimal processing.

I/o devices vary widely in function and speed.

The **i/o subsystem** of the kernel separates the rest of the kernel from the complexities of managing the i/o devices.

The **device drivers** provide a uniform device access interface to the i/o subsystem.



Types of devices –

- storage devices (disk, USB drives)
- transmission devices (modem)
- human-interface devices (keyboard, mouse, display)
- specialized devices (microscope, telescope, case of flight control computer)

Hardware –

- **port** - a connection point where a device is connected
- **bus** - set of wires to transfer data and control information
- **controller** - controls a port or a device

An i/o port consists of –

- status register
- control register
- data-in register
- data-out register

Modes of performing i/o –

- polling (busy-waiting)
- interrupts
- DMA
  - burst mode
  - cycle stealing mode
  - direct virtual memory access (DVMA)

Complementary devices –

- teletype = keyboard + display
- photocopier = scanner + printer

Virtual devices –

- PDF printer

Properties of devices –

- character-stream versus block
- sequential access versus random access
- synchronous versus asynchronous
- sharable versus dedicated
- speed
- read-only, write-only, read-write

Services provided by the i/o subsystem –

- i/o scheduling
- buffering
- caching
- spooling (a spool is a buffer that holds the output of a device, like a printer, that cannot accept interleaved data streams)
- error handling
- i/o protection

Protection - from internal and genuine errors

Security - from external and intentional threats

Operating systems implement protection and provide support for security.

Object - hardware or software objects

Domain - specification of resources a process can access

#### Access matrix

Object Domain \	O1	O2	O3	O4
D1	R			R
D2		RW	R	RW
D3	R		E	

Switch - switch domain

Copy (\*)

- transfer
- copy limited
- copy unlimited

Object Domain \	O1	O2	O3	O4	D1	D2	D3
D1	R			R			
D2		RW*	R	RW	S		S
D3	R		E*		S	S	

Implementation of access matrix –

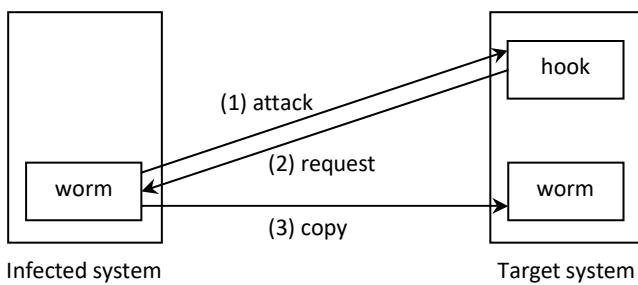
(Access matrix is sparse)

- Global table
  - ordered triples <domain, object, rights>
- **Access lists of objects**
  - a column of access matrix
  - blank entities may be discarded

- **Capability lists of domains**
  - a row of access matrix
  - a process should not be able to modify its capability list

Threats –

- Trojan horse
  - attractive and harmless cover program
  - harmful hidden program
  - used as virus carrier
- Trap door
  - hidden door
- Logic bomb
  - initiate attack under specific situation
- Virus
  - self-replicating and infectious
  - modify and destroy files
  - Linux is not susceptible to virus because it does not allow modifying executable programs
- Worm
  - infection programs spread through networks



Types of computer viruses –

- file/parasitic
  - appends itself to a file
- boot/memory
  - infects the boot sector
- macro
  - written in high-level language like VB and affects MS Office files
- source code
  - searches and modifies source codes
- polymorphic
  - changes on copying each time
- encrypted
  - encrypted virus + decrypting code
- stealth
  - avoids detection by modifying parts of system that can be used to detect it, like the read system call
- tunneling
  - installs itself in the interrupt service routines and device drivers
- multipartite
  - infects multiple parts of the system

Chapters: 13, 17 and 18.  
Sections: 13.1 – 13.4, 17.4, 17.5, 18.2 and 18.3.





