AKTU

B. Tech II-Year

Gateway Classes

Semester -IV

CS IT & Allied Branches

BCS401-Operating system

UNIT-4 : Memory Management

Gateway Series for Engineering

- Topic Wise Entire Syllabus
- Long - Short Questions Covered
- AKTU PYQs Covered
- DPP
- Result Oriented Content

Download App

For Full Courses including Video Lectures

AKTU

B. Tech II-Year

Gateway Classes

BCS401-Operating system

Unit-4

Introduction to Memory Management

Syallbus

Memory Management: Basic bare machine, Resident monitor, Multiprogramming with fixed partitions, Multiprogramming with variable partitions, Protection schemes, Paging, Segmentation, Paged segmentation, Virtual memory concepts, Demand paging, Performance of demand paging, Page replacement algorithms, Thrashing, Cache memory organization, Locality of reference.

Download App

For Full Courses including Video Lectures

Operating System

Unit – 4 Lecture -1

CS/IT/CS Allied/MCA

By- Dr. Kapil Kumar
M.Tech.(CSE), Ph.D.(CSE)

OPERATING SYSTEM

Syllabus

Unit – 4

Memory Management: Basic bare machine, Resident monitor, Multiprogramming with fixed partitions, Multiprogramming with variable partitions, Protection schemes, Paging, Segmentation, Paged segmentation, Virtual memory concepts, Demand paging, Performance of demand paging, Page replacement algorithms, Thrashing, Cache memory organization, Locality of reference.

OPERATING SYSTEM

Today's Target

- ❑ Main Memory and Models
- ❑ Memory Management Unit and its functions
- ❑ Logical and Physical Address Spaces
- ❑ Swapping
- ❑ AKTU PYQs

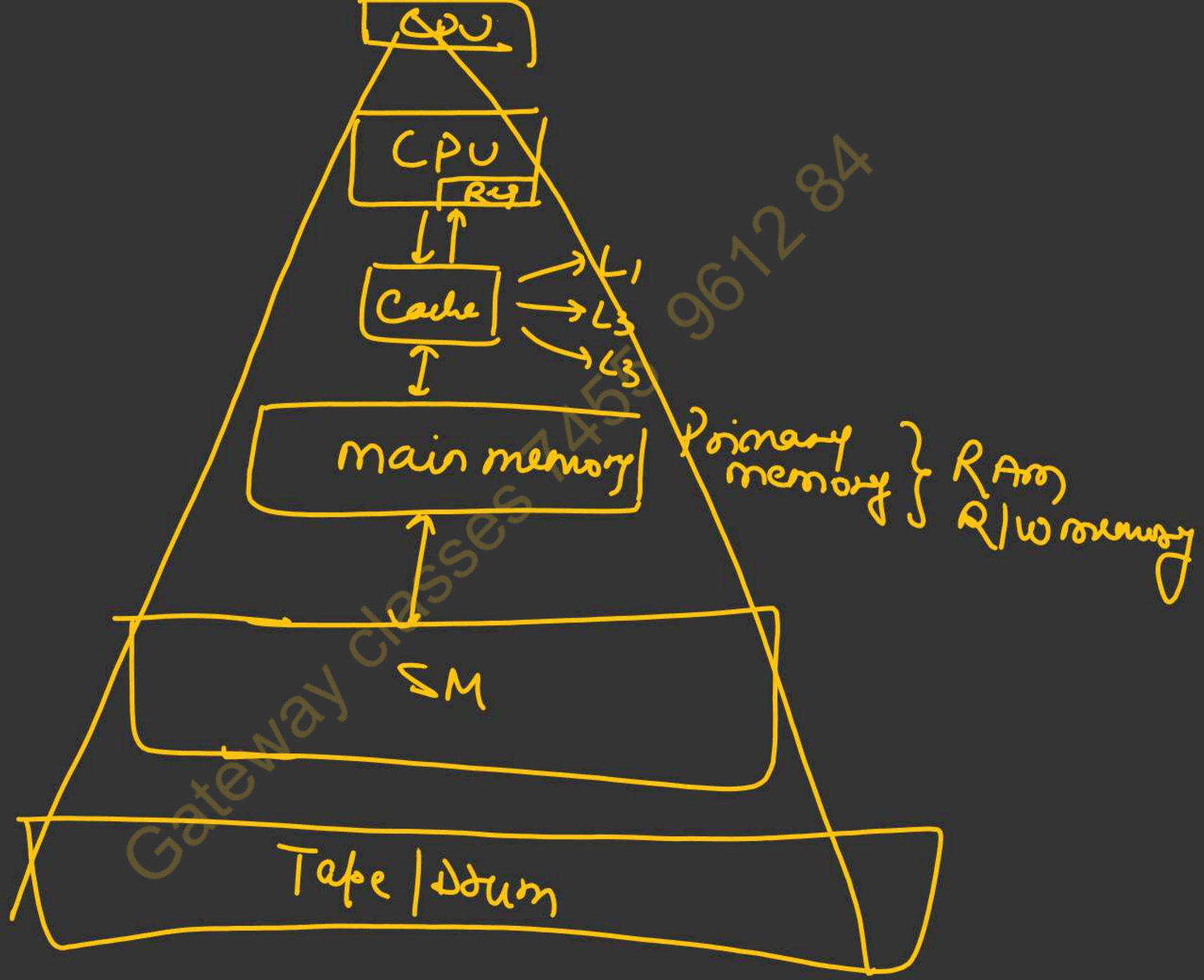
Main Memory and Models

- Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from the main memory.
- Some key main memory models are as follows -

1. **Bare Machine Model:** The bare machine model of main memory in an operating system refers to a simplistic way of managing memory where the hardware is used directly without any sophisticated operating system-level memory management.

Some Key characteristics of the bare machine model:

- **No Abstraction:** In the bare machine model, there is no abstraction of memory i.e. programs and processes access the physical memory addresses directly.
- **No Protection:** There is no protection mechanism in place to prevent one program from accessing or modifying the memory allocated to another program.



MAIN MEMORY

* No preloaded OS
* Task specific
* Simplest form of memory mgmt

❑ Single Address Space:

- All processes and the operating system share a single address space.
- This means that there is no concept of virtual memory, and each process is aware of the physical memory layout.

❑ Direct Access:

- Programs have direct access to hardware and memory.

❑ Suitable for Simple Systems:

- The bare machine model is typically suitable for simple, single-tasking systems where the overhead of sophisticated memory management is unnecessary.

❑ Lack of Features:

- Advanced features like paging, segmentation, memory protection, and virtual memory are not present in the bare machine model.

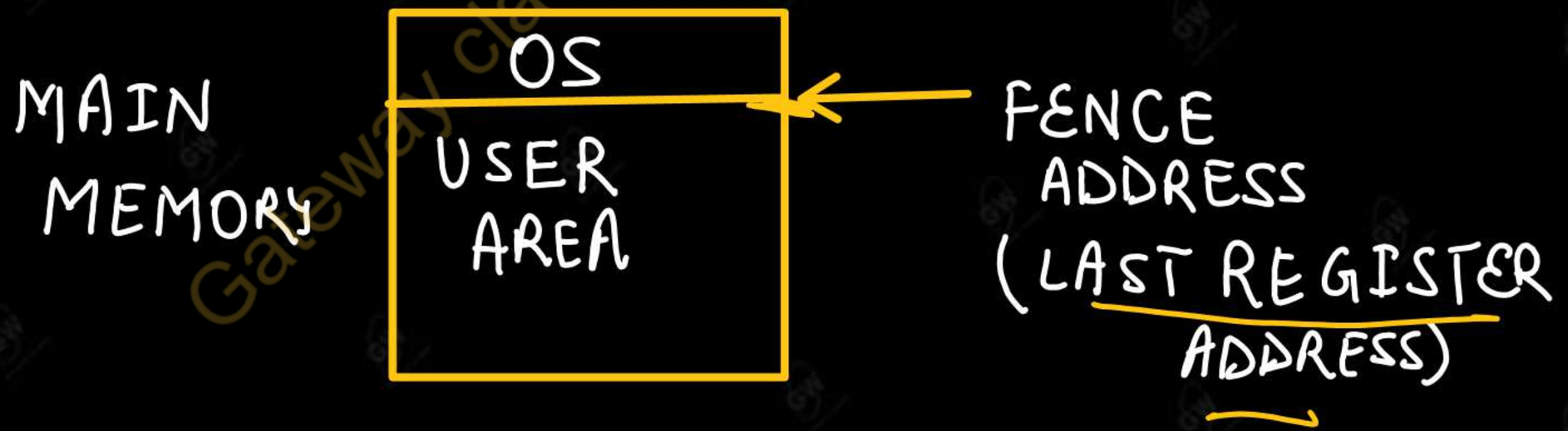
2. Resident Monitor Model:

- The resident monitor is a small, permanently resident program that remains in memory to control the execution of jobs.

Key Characteristics:

- **Permanent Residency:**

- The monitor stays permanently in the main memory, meaning it is always available to manage the execution of jobs and handle system control.
- Fence registers are used for security between operating system and user area.



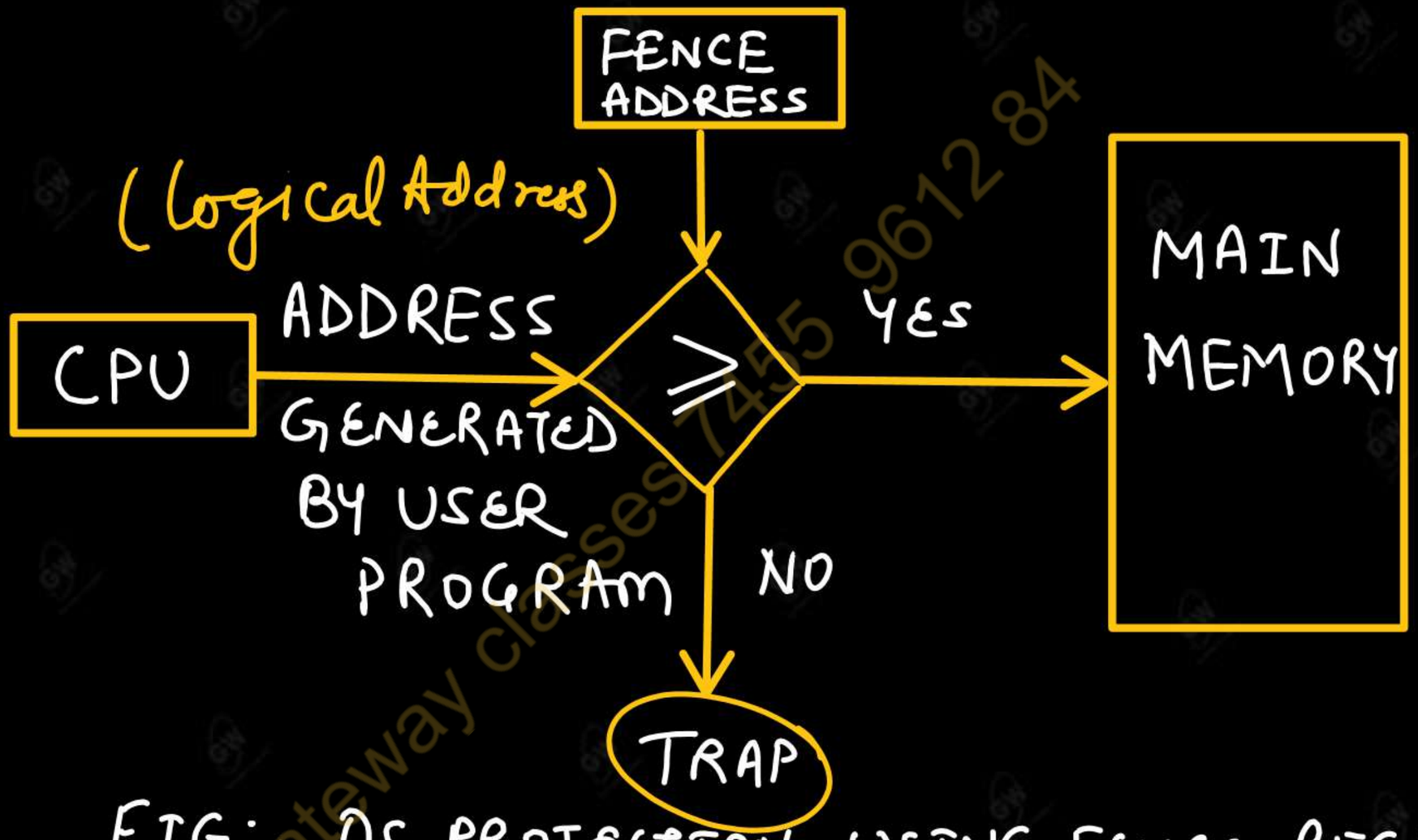


FIG:- OS PROTECTION USING FENCE REGISTER

❑ Batch Processing:

- The resident monitor is closely associated with batch processing systems.

❑ Job Sequencing:

- It automates the sequencing of jobs, loading the next job into memory after the current job completes. This reduces idle time and increases system throughput.

❑ Memory Layout:

- **Monitor Area:** A fixed part of the main memory is allocated to the resident monitor. This area is not used by user programs.
- **User Area:** The rest of the memory is available for loading and executing user jobs. Jobs are loaded into this area sequentially.

❑ Basic Memory Management:

- The resident monitor provides simple memory management.
- It loads jobs into available memory spaces but typically does not support advanced features like dynamic allocation or virtual memory.

Limitations:

- ❑ **No Multitasking:** The system typically handles one job at a time, lacking the ability to run multiple jobs concurrently.
- ❑ **Limited Memory Management:** Basic memory management without advanced features like virtual memory or dynamic allocation.
- ❑ **No Interactive Use:** Designed for batch processing rather than interactive use.

3. Modern Main Memory Model:

- Modern main memory models are highly sophisticated, supporting complex features like virtual memory, memory protection, multitasking, and efficient resource allocation.

Key Components and Features:

- **Virtual Memory:** Virtual memory allows the operating system to use disk storage as an extension of RAM, providing the illusion of a larger main memory space.
- **Paging:** Memory is divided into fixed-size blocks called pages. Virtual pages are mapped to physical pages in RAM, and the system can swap pages in and out of disk storage as needed.
- **Segmentation:** Memory is divided into segments based on logical divisions such as functions or data structures. Each segment can be of variable size, providing more flexible memory management.

- **Memory Protection:**
  - Ensures that each process has its own private memory space, preventing one process from accessing or modifying the memory of another process.
- **Dynamic Memory Allocation:**
  - Provides dynamic allocation and deallocation of memory during program execution.
- **Garbage Collection:** Automatically reclaims memory that is no longer in use by the program, used in languages like Java and Python.
- **Memory Caching:**
  - Small, fast cache memory located close to the CPU to store frequently accessed data and instructions, reducing access times.
  - Multi-level caches (L1, L2, L3) to balance speed and size.

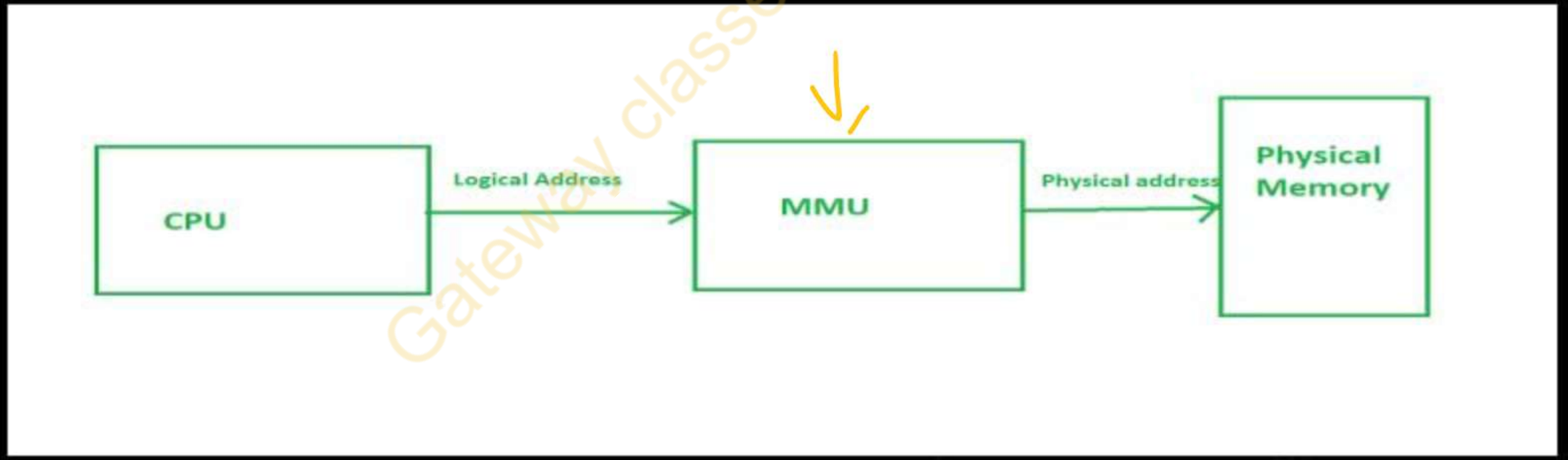
❑ Memory Over commitment:

- Allows the operating system to allocate more virtual memory than the physical memory available.
- **Swap Space:** Uses disk space to hold parts of the memory that are not currently in use, freeing up RAM for active processes.

Memory Management Unit

mmu

- The main purpose of an MMU is to serve as a link or bridge between the physical memory of the computer and the central processing unit (CPU).
- Another name for it is a paged memory management unit (PMMU).
- Every computer system has a memory management unit , it is a hardware component whose main purpose is to convert virtual addresses created by the CPU into physical addresses in the computer's memory.



Functions of Memory Management Unit(MMU)

❑ Address Translation:

- Converting virtual addresses (used by programs) into physical addresses (used by the hardware).
- **Physical Address:** A physical address refers to a specific location in the computer's physical memory (RAM). It is the actual location in the hardware where data is stored. The CPU uses physical addresses to fetch or store data in memory.
- **Virtual Address:** This is an address used by a program to access memory.
- MMU's primary job is to converting virtual addresses into physical addresses.
- This translation is essential for the CPU to access the correct locations in RAM and interact with the necessary data.

❑ Memory Protection:

- Ensuring that one process cannot access the memory allocated to another process without permission, thereby providing security and stability.
- MMUs play a crucial role in implementing memory protection mechanisms.
- By implementing access control rules and regulations, they stop illegal usage of particular memory locations.
- By doing this, the operating system's security and data integrity are ensured.

❑ Virtual Memory Management:

- MMUs serves a part in the implementation of this method, which allows heavier programs to be executed than what can fit in the physical RAM.
- The system can use virtual memory to extend RAM by using a portion of the storage space on the disc and dynamically switch data between RAM and the disc as needed.

GW
GATEWAY CLASSES

❑ Memory Segmentation:

- Memory segmentation is a feature found in certain MMUs.
- It splits the computer's memory into sections that have multiple authorizations and features.
- This segmentation provides a more granular control over memory access and aids in optimizing memory utilization.

❑ Caching:

- Often working with the CPU cache to manage and optimize access to memory, enhancing performance.

Logical Versus Physical Address space

- An address generated by the CPU is commonly referred to as a "logical address" where as an address seen by the memory unit – that is, the one loaded into "memory – address register" of the memory is commonly referred to as a "Physical address".
- The set of all logical addresses generated by a program is a "logical address space".
- The set of all physical addresses corresponding to these logical addresses is a "physical address space"
- The run time mapping from virtual to physical addresses is done by a hardware device called the memory – management unit (MMU).
- In this mapping schemes the base register is now called a relocation register.
- The value in the relocation register is added to every address generated by a user process at the time the address in sent to memory.

PC ⇒ Add of next instruction

IR : a = b + c

MAR

MBR

Exe

PC = PC + 1

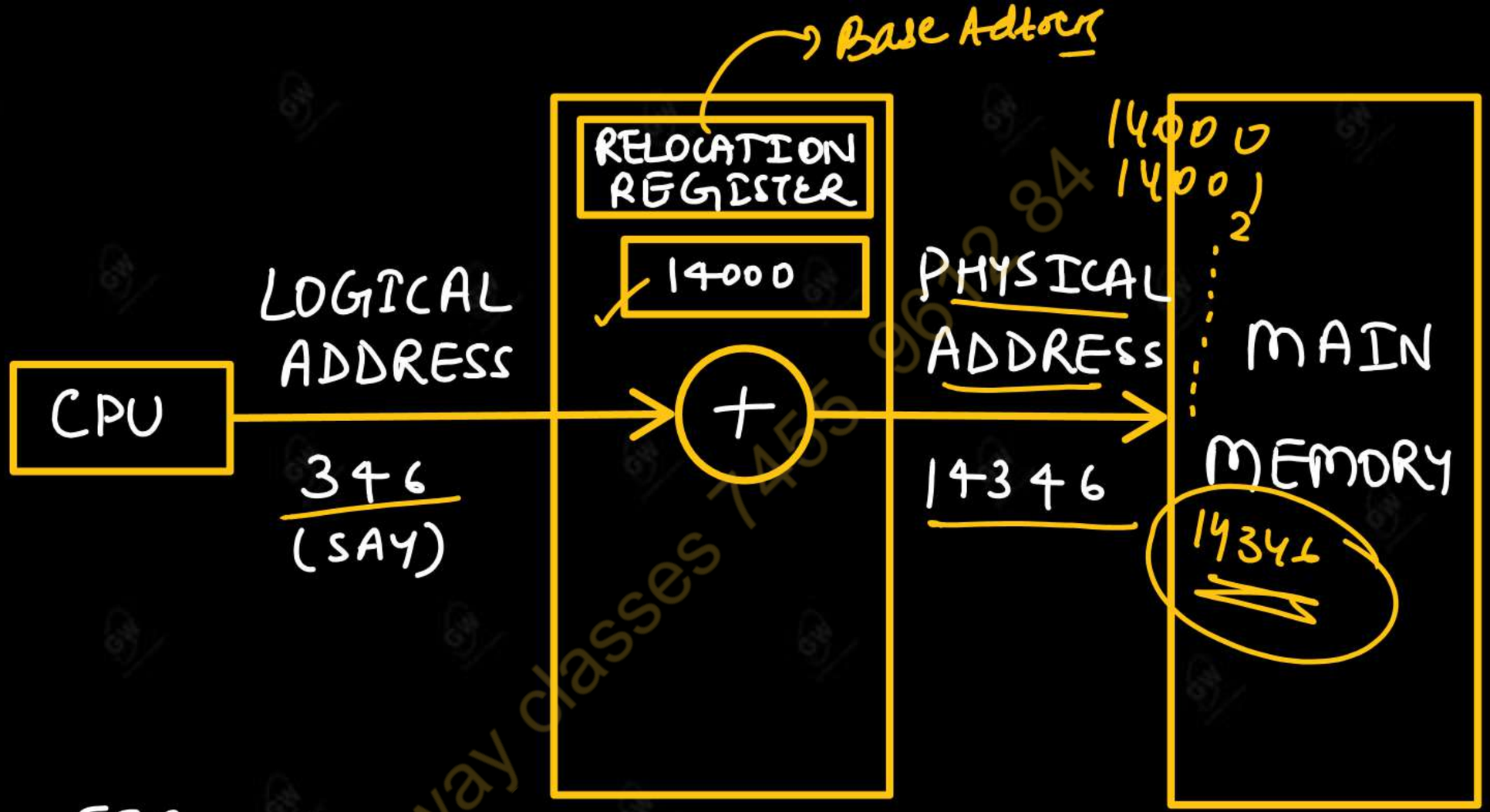


FIG: DYNAMIC RELOCATION USING RELOCATION REGISTER

SWAPPING

- A process must be in memory to be executed.
- A process can be swapped temporarily out of memory to a backing store and then brough back into memory for continued execution.
- For example, assume a multiprogramming environment with a round – robin CPU – scheduling algorithm.
- When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into memory space that has been freed.
- In the meantime, the CPU schedular will allocate a time slice to some other process in memory.
- When each process finishes its quantum, it will be swapped with another process.
- In case of priority scheduling , the swapping can be called roll out, roll in.

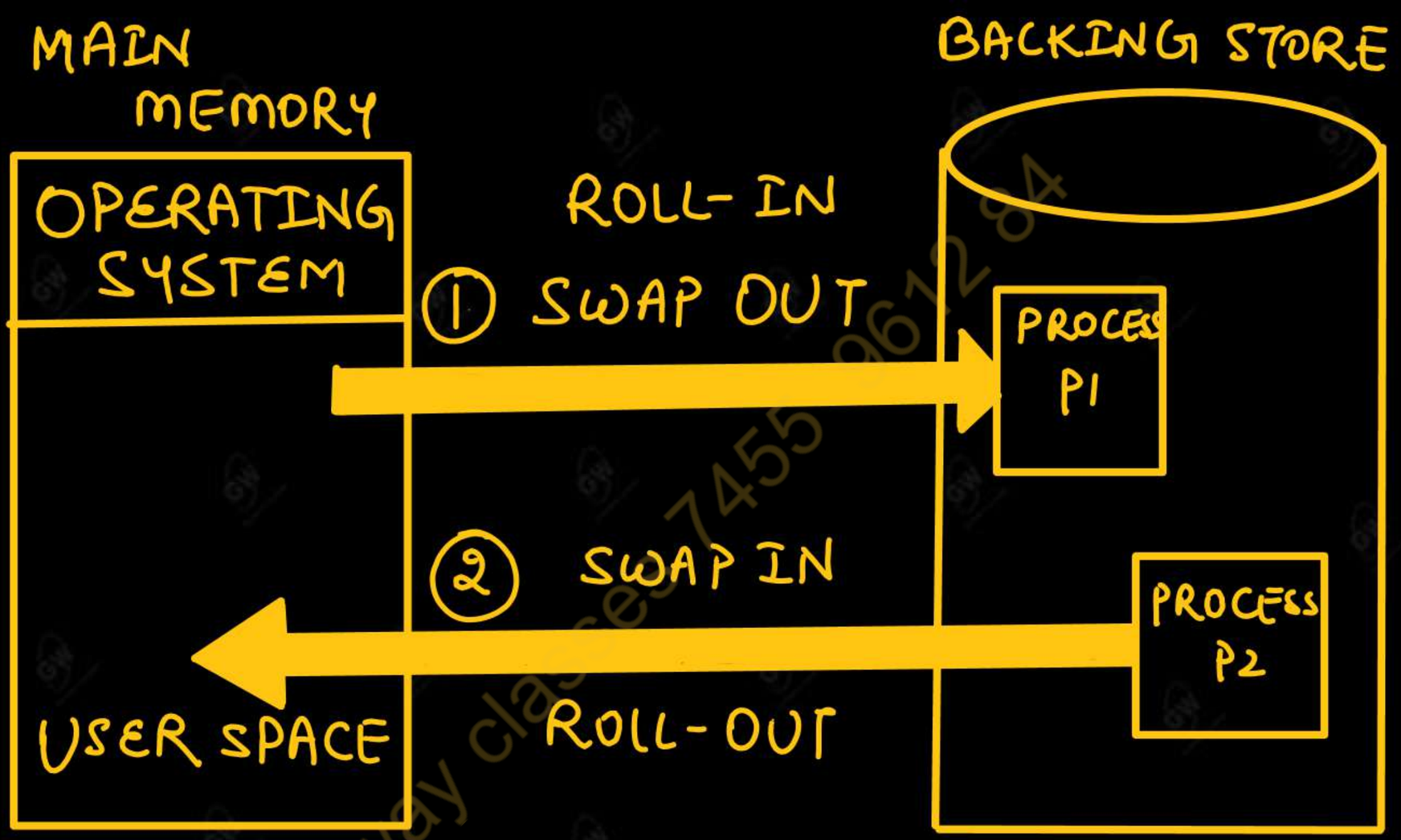


FIG: SWAPPING OF TWO PROCESSES USING A DISK AS A BACKING STORE

AKTY PYQs

Q.1. What is the main function of memory management unit? MMU — 2017-18, 2 Marks

Q.2. what do you mean by memory management? — 2018-19, 2 Marks

Q.3. Explain the logical address space and physical address space diagrammatically. — 2021-22, 2 Marks

Q.4. Distinguish between physical and logical address space of a process. — 2022-23, 2 Marks

OPERATING SYSTEM

Today's Target

- Memory Management Technique
- Contiguous Memory Allocation: Multiprogramming with Fixed Partitioning and Multiprogramming with Variable Partitioning
- Internal and External Fragmentation
- AKTU PYQs

Memory Management

- In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes.
- The task of subdividing the memory among different processes is called Memory Management.
- Memory management is a method in the operating system to manage operations between main memory and disk during process execution.
- The main aim of memory management is to achieve efficient utilization of memory.

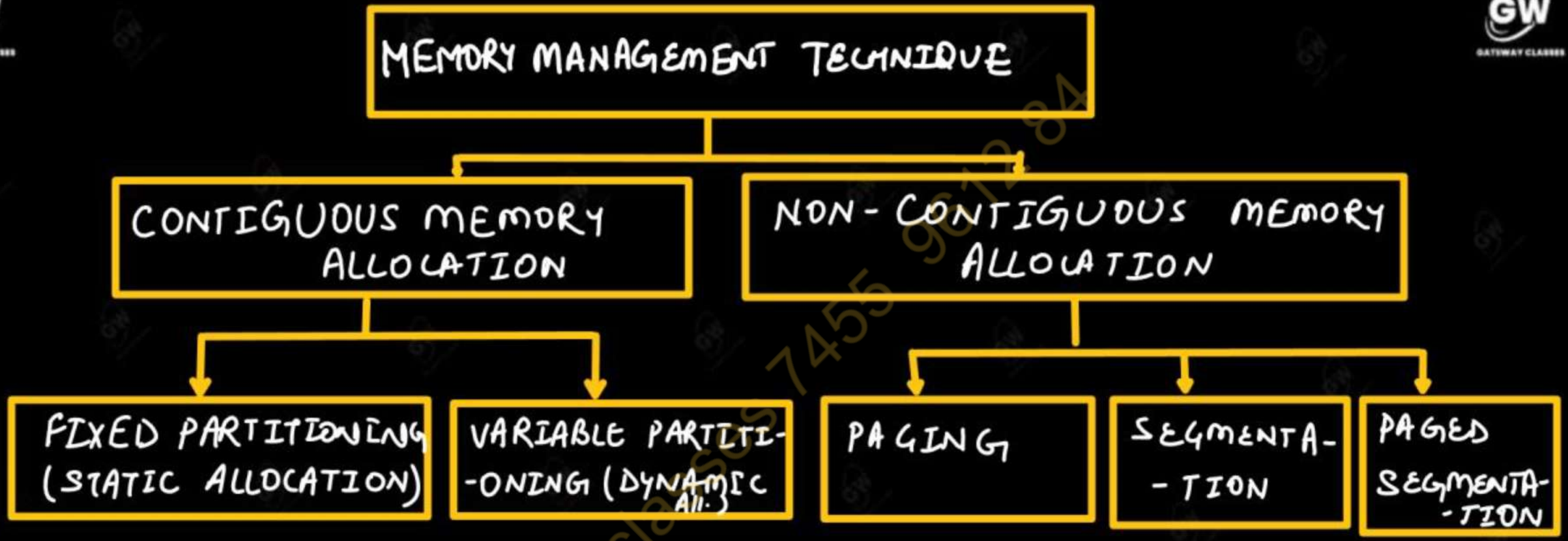


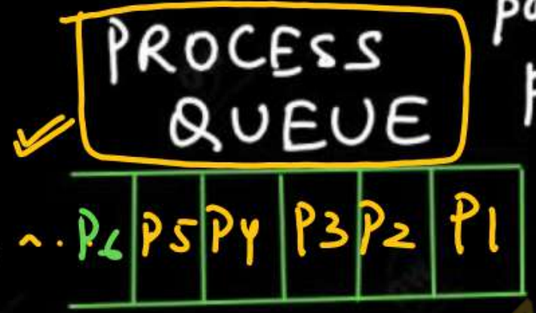
FIG :- VARIOUS MEMORY MANAGEMENT TECHNIQUES

Contiguous Memory Allocation

Multiprogramming with Fixed Partitioning (Static Allocation)

- It is one of the most oldest memory management technique which is easy to implement.
- Multi-programming with fixed partitioning is a contiguous memory management technique in which the main memory is divided into fixed sized partitions which can be of equal or unequal size. But their size once fixed cannot be changed i.e. starting and end address of the partition is fixed in advance. (static allocation)
- In case of equal size fixed partition, there is a single process queue in which all the processes waiting to come in main memory for execution are maintained.
- If there is partition available, the process is loaded into that partition.
- Since all partitions are of same size it does not matter which partition is used.
- Consider the next figure -

New process that enter the system is stored in this queue



Processes are stored in any available partition as all partitions are of equal size

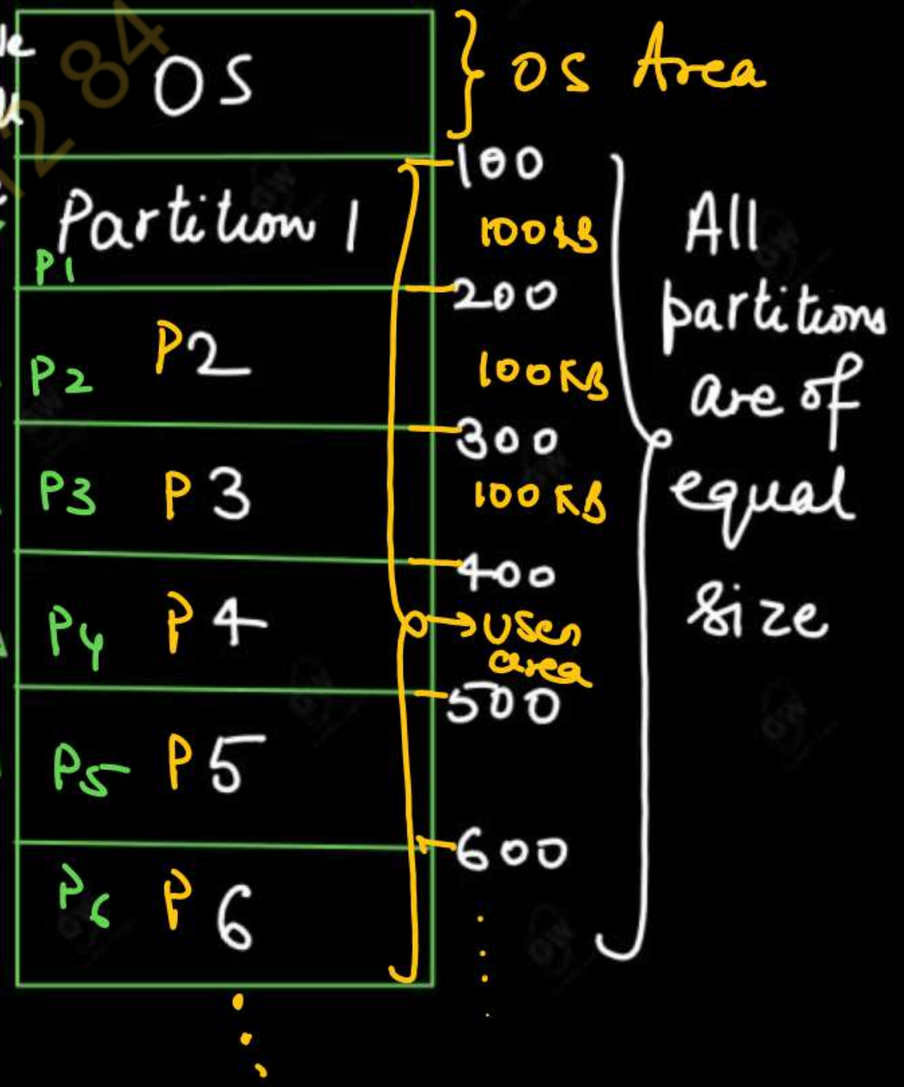


FIG:- FIXED PARTITIONING WITH EQUAL SIZE PARTITIONS

- In case of unequal size fixed partition, the size of each partition is also fixed in advance, it can not be changed. but sizes of partitions may be different.
- Whenever we have to allocate a process in memory then a free partition that is big enough to hold the process is found. Then the memory is allocated to the process.
- If there is no free space available then the process waits in the queue to be allocated memory.

Techniques

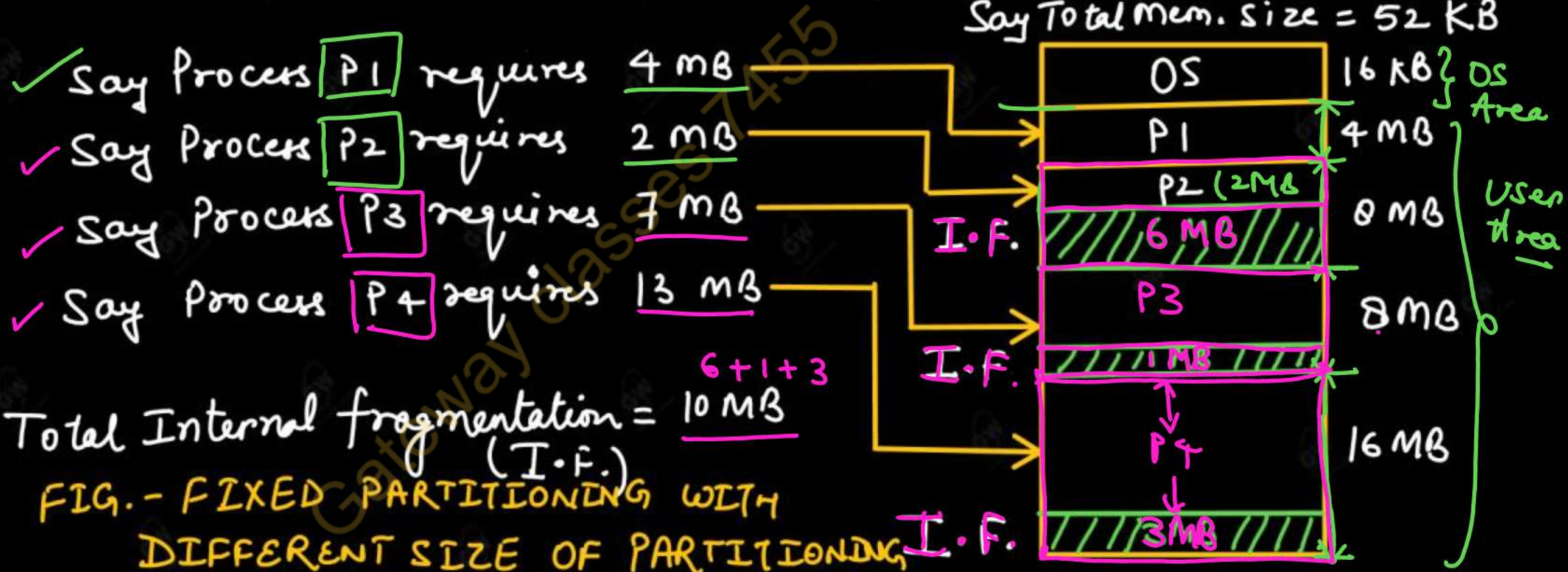


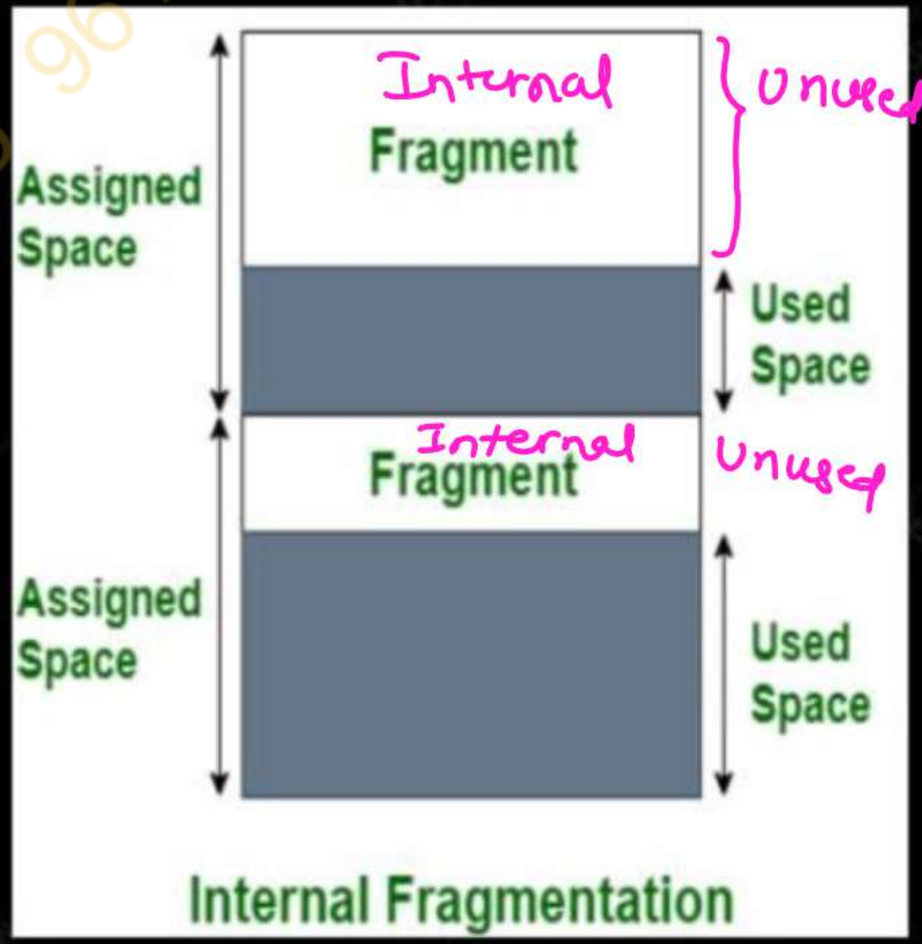
FIG. - FIXED PARTITIONING WITH DIFFERENT SIZE OF PARTITIONING

Limitations of Multiprogramming with Fixed Partitioning

(1) Internal Fragmentation:

- Internal fragmentation happens when the memory is split into mounted-sized blocks.
- Whenever a method is requested for the memory, the mounted-sized block is allotted to the method.
- In the case where the memory allotted to the method is somewhat larger than the memory requested, then the difference between allotted and requested memory is called internal fragmentation.
- We fixed the sizes of the memory blocks, which has caused this issue.

- If we use dynamic partitioning to allot space to the process, this issue can be solved.

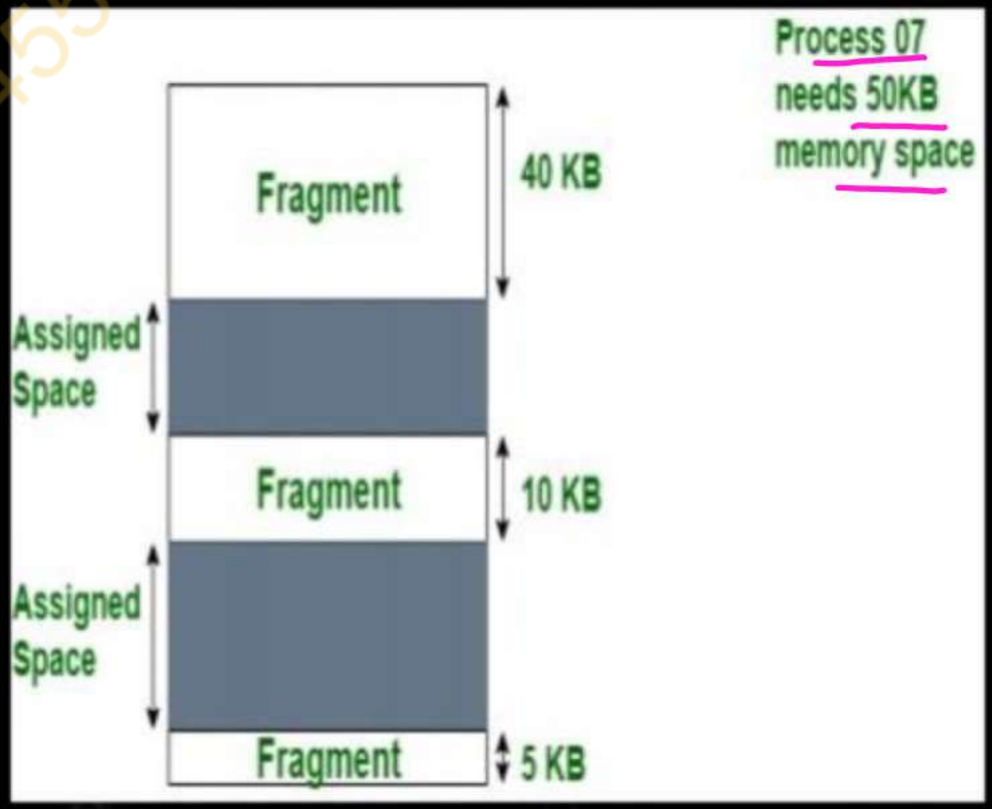


(2) Limitation on size of a process say the maximum size of a partition is 16 MB then a process bigger then 16 MB cannot be allocated to memory.

(3) Suppose memory is divided into 4 partitions then only maximum 4 processes can be accommodated into such memory i.e. limitation on degree of multiprogramming.

(4) External Fragmentation:

- External fragmentation happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method.
- However, the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner.
- Whether you apply a first-fit or best-fit memory allocation strategy it will cause external fragmentation.
- In this diagram, we can see that, there is enough space (55 KB) to run a process-07 (required 50 KB) but the memory (fragment) is not contiguous. Here, we use compaction, paging, or segmentation to use the free space to run a process.



Multiprogramming with Variable Partitioning

- Multi-programming with variable partitioning is a contiguous memory management technique in which the main memory is not divided into partitions that are variable (not fixed) and the process is allocated a chunk of free memory that is big enough for it to fit.
- Each partition is capable of holding one process. Partition can change size as the need arises.
- The space which is left is considered as the free space which can be further used by other processes. It also provides the concept of compaction.
- In compaction the spaces that are free and the spaces which not allocated to the process are combined and single large memory space is made.
- In variable partitioning whenever a process is coming to the RAM only then we are allocating the space to that process means keep the RAM empty for now, when process come in the RAM then at run time the capacity they need, the space they need according to that space, processes will be allocated in memory i.e. partition of main memory at run time.
- Consider the next diagram-

- When a process is dispatched from process queue to store in memory, Operating System first searches for the hole large enough to accommodate that process.
- If none of the available holes are large enough to accommodate that process then available adjacent free partitions can be merged together to form big hole to accommodate that process.
- This process of merging two adjacent free partitions to form big hole is called coalescing.

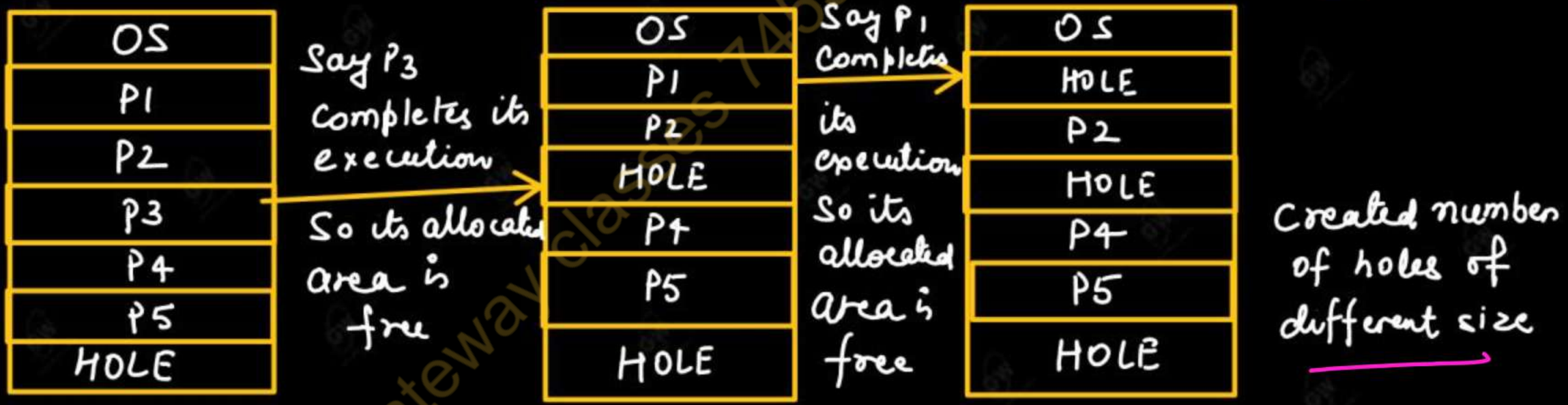


FIG:- HOLES IN VARIABLE PARTITIONING MULTIPROGRAMMING SYSTEM

❑ Limitation:-

- Suppose one process of 4 MB and another process of 4 MB is deallocated from memory then it will provide 2 holes of 4 MB each and say other process of size 8 MB comes then this process can not be allocated to any whole due to contiguous memory allocation because in contiguous memory allocation the process can not be spanned at different location.
- That is, although the memory is available but that is not in contiguous manner, processes can not be allocated to such memory i.e. "External fragmentation".
- Solution of External Fragmentation - Compaction Technique i.e. move all processes at one side, in this way all blanks will be collected at one place.

- The memory blocks available comprise a set of holes of various sizes scattered throughout the memory. When a process arrives and needs memory, the system searches the for a hole that is large enough for this process.
- This procedure is a particular instance of the general "dynamic storage – allocation problem", which concerns how to satisfy a request of sixe n from a list of fee holes.
- There are many solutions to this problem.
- The first –fit, best-fit, and worst-fit strategies are the ones onset commonly used to select a free hole from the set of available holes.

Difference Between Fixed and Variable Partitioning

S.NO.	Fixed partitioning	Variable partitioning
1.	In multi-programming with fixed partitioning the main memory is divided into fixed sized partitions. (Static) Equal - unequal	In multi-programming with variable partitioning the main memory is not divided into fixed sized partitions.
2.	Only one process can be placed in a partition.	In variable partitioning, the process is allocated a chunk of free memory.
3.	It does not utilize the main memory effectively.	It utilizes the main memory effectively.
4.	There is presence of internal fragmentation and external fragmentation.	There is external fragmentation.
5.	Degree of multi-programming is less.	Degree of multi-programming is higher.

Difference Between Fixed and Variable Partitioning

S. NO.	Fixed partitioning	Variable partitioning
6.	It is more easier to implement.	It is less easier to implement.
7.	There is limitation on size of process.	There is no limitation on size of process.

Difference Between Internal and External Fragmentation

S.NO	Internal fragmentation	External fragmentation
1.	4KB 2KB If size of allocated memory > required memory, so difference of memory is wasted i.e. the difference between memory allocated and required space or memory is called Internal fragmentation.	Total memory space exists to satisfy a request, but it is contiguous, contiguous says one process = one partition and can not be divided. The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
2.	Internal fragmentation happens when the process is smaller than the memory.	External fragmentation happens when the process is removed.

S.NO	Internal fragmentation	External fragmentation
3.	The solution of internal fragmentation is the best-fit block. It can be stopped by dynamic partitioning.	The solution to external fragmentation is compaction, paging and segmentation etc. (non contiguous memory allocation). Paged Segment Fatu
4.	As it is internal memory of allocated memory, so named internal fragmentation.	As it is unallocated memory part so named external fragmentation.

AKTU PYQs

Q.1. *Explain the principal advantages of multiprogramming.* **2022–23**, 2 Marks

Q.2. *Explain internal and external fragmentation.* **2022–23**, 2 Marks

Q.3. *Explain in brief about the Multiprogramming with fixed partitions.* **2021–22**, 2 Marks

Q.4. *Explain the difference between External fragmentation and Internal fragmentation. How to solve the fragmentation problem using paging.* **2018**–19, 10 Marks

Q.5. *Write the difference between internal and external fragmentation.*

2017–18, 2 *Marks*, 2016-17, 2 Marks

Q.6. *What are the different techniques to remove fragmentation in case of multiprogramming with fixed partitions and variables partitions?*

2015-16, 10 Marks

OPERATING SYSTEM

Today's Target -

- Dynamic Memory Allocation Techniques (First – Fit, Best – Fit and Worst – Fit)
- AKTU PYQs

DYNAMIC MEMORY ALLOCATION TECHNIQUES

- **First - Fit**
  - Allocate the first – hole that is big enough.
  - Searching can start either at the beginning of the set of holes or at the location where the previous first – fit search ended.
  - We can stop searching as soon as we find a free hole that is large enough.
- **Advantages:**
  - Simple and fast because it stops searching once a suitable block is found.
  - Tends to be faster than best fit and worst fit in practice.
- **Disadvantages:**
  - Can lead to fragmentation over time because it may leave many small, unusable memory blocks.
  - Does not always use memory as efficiently as other methods.

Best - Fit

- Allocate the smallest hole that is big enough.
- We must search the entire list, unless the list is ordered by size.
- This strategy produces the smallest leftover hole.

Advantages:

- Tends to use memory more efficiently by minimizing wasted space within allocated blocks.
- Can reduce fragmentation in some cases.

Disadvantages:

- Slower than first fit because it searches the entire list.

Worst - Fit

- Allocate the largest hole.
- Again we must search the entire list, unless it is sorted by size.
- This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best fit approach.

Advantages:

- Helps to avoid creating too many small, unusable blocks because it leaves large free blocks.
- Can be beneficial in situations where large memory blocks are frequently requested.

Disadvantages:

- Slower than first fit because it searches the entire list.
- May lead to inefficient use of memory over time, as large blocks are broken down and may not be reused efficiently.

Example 1: Given memory partitions of -

100 K, 500 K, 200 K, 300 K and 600 K (in order).

How would each of the first – fit, best - fit and worst – fit algorithms place processes of -

212 K, 417 K, 112 K and 426 K (in order)?

P1 P2 P3 P4

Which algorithms makes the most efficient use of memory. Find the Internal Fragmentation.

2012-13, 5 Marks, 2018-19, 10 Marks

Solution:-

Process ID	Memory Requested
P1	212 K
P2	417 K
P3	112 K
P4	426 K

First Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
100 K	–	–	FREE	–
500 K	P1	212 K	Allocated	500 – 212 = 288 K
200 K	P3	112 K	Allocated	200 – 112 = 88 K
300 K	–	–	FREE	–
600 K	P2	417 K	Allocated	600 – 417 = 183 K
Total- 1700K		741 K		Total IF 559 K

- Total Available Memory : 1700 K, Total Memory Used : 741 K, Total Internal Fragmentation: 559 K
- Process P4 has no memory partition allocated to it.
- Two memory partition of size 100 K and 300 K are free which can not be allocated to P4 because it has size more then these memory partitions.

Best Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
100 K	-	-	FREE	
500 K	P2	417 K	Allocated	500 - 417 = 83 K
200 K	P3	112 K	Allocated	200 - 112 = 88 K
300 K	P1	212 K	Allocated	300 - 212 = 88 K
600 K	P4	426 K	Allocated	600 - 426 = 174 K
1700 K		1167 K		Total IF = 433 K

- Total Available Memory: 1700 K, Total Memory Used: 1167 K, Total Internal Fragmentation: 433 K
- Only one memory partition of size 100 K is free but all the processes are allocated to the memory partitions.

Worst Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
100 K	-	-	FREE	—
500 K	P2	417 K	Allocated	500 - 417 = 83 K
200 K	-	-	FREE	-
300 K	P3	112 K	Allocated	300 - 112K = 188 K
600 K	P1	212 K	Allocated	600 - 212 K = 388 K
Total = 1700K		741 K		Int. Frg = 659 K

- Total Available Memory:1700 K, Total Memory Used:741 K, Total Internal Fragmentation:659 K
- Process P4 has no memory partition allocated to it.
- Two memory partition of size 100 K and 200 K are free which can not be allocated to P4 because it has size more then these memory partitions.
- Best – fit algorithm makes most efficient use of memory, because in this, all the processes are allocated memory partition with minimum internal fragmentation. (433K)

Example 2:- Given memory partitions of -

20 K, 100 K, 40 K, 200 K, 10 K (in order).

How would each of the first – fit, best fit and worst – fit algorithms place processes of -

90 K, 50 K, 30 K and 40 K (in order)?

P1 P2 P3 P4

Which algorithm makes the most efficient use of memory. Find the Internal Fragmentation.

Solution:-

Process ID	Memory Requested
P1	90 K
P2	50 K
P3	30 K
P4	40 K

First Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
20 K	–		FREE	–
100 K	P1	90 K	Allocated	100 – 90 = 10 K
40 K	P3	30 K	"	40 – 30 = 10 K
200 K	P2	50 K	"	200 – 50 = 150 K
10 K	–	–	FREE	–
370 K		170 K		Total I.F. = 170 K

- Total Available Memory : 370 K, Total Memory Used : 170 K, Total Internal Fragmentation:170 K
- Process P4 has no memory partition allocated to it.
- Two memory partition of size 20 K and 10 K are free which can not be allocated to P4 because it has size more then these memory partitions.

Best Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
20 K	—	—	FREE	—
100 K	P1	90K	Allocated	100−90 = 10 K
40 K	P3	30K	,,	40−30 = 10 K
200 K	P2	50K	,,	200−50 = 150 K
10 K	—	—	FREE	—
370K		170K		Total I.F = 170K

- Total Available Memory:370 K,
- Total Memory Used:170 K,
- Total Internal Fragmentation:170 K

Worst Fit:-

Memory Partition	Process ID	Memory Requested	Status (FREE/BUSY)	Internal Fragmentation
20 K	—	—	FREE	
100 K	P2	50 K	Allocated	100 − 50 = 50 K
40 K	P3	30 K	"	40 − 30 = 10 K
200 K	P1	90 K	"	200 − 90 = 110 K
10 K	—	—	FREE	—
370K		170K		Total I F = 270K

- Total Available Memory : 370 K
- Total Use : 170 K
- Total Fragmentation : 270 K

AKTU PYQs

Q.1. Given memory partitions of - 100 k, 500 k, 200 k, 300 k and 600 k (in order).

How would each of the first – fit, best fit and worst – fit algorithms place processes of

212 k, 417 k, 112 k and 426 k (in order)?

Which algorithms makes the most efficient use of memory. Find the internal fragmentation.

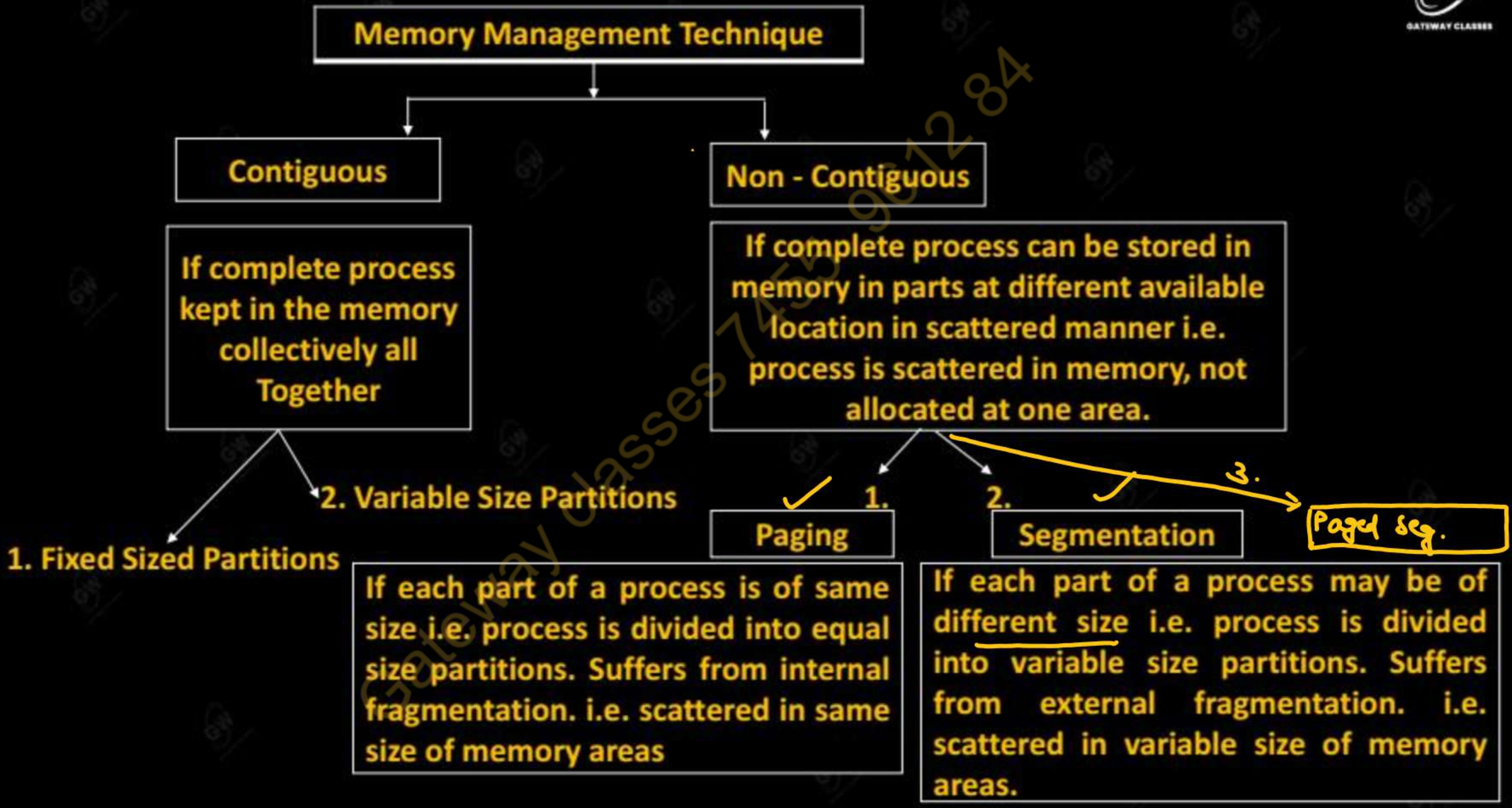
2012-13, 5 Marks, 2018-19, 10 Marks, 2019-20, 10 Marks

OPERATING SYSTEM

Today's Target -

- ❑ What is non – contiguous memory allocation?
- ❑ Paging
- ❑ AKTU PYQs

❑ We known that : -



Non-contiguous Memory Allocation

❖ **What is non contiguous memory allocation?**

- Contiguous means collective together and non contiguous means not collective altogether.
- Non contiguous memory allocation assigns the separate memory blocks at the different location in the physical or main memory. or primary memory
- If a process comes and demands for main memory then that process will be placed in memory at different locations in a non-contiguous manner.
- Where the space will be available, the process will be placed at different location in the main memory. In case of contiguous memory allocation, it was necessary to place the process at one location.
- In non-contiguous memory allocation, there will be different free memory partitions in scattered form (non-contiguous manner). Where the space will be available the process will be placed.

- In contiguous memory allocation, the problem of internal and external fragmentation was observed.
- Suppose a process of 2 KB comes and the size of partition is 10 KB, then 8KB will be the internal fragmentation, and if portions are remain free then the external fragmentation will also be observed.
- These problems can be overcome by using the concept of non contiguous memory allocation in which the process is divided into parts so that different parts may be placed at different memory spaces available in non-contiguous memory.

Paging

- Paging is a memory management technique used by the operating systems to manage how a computer's physical memory (RAM) is utilized.
- It involves dividing the computer's memory into fixed-size blocks called pages and mapping these pages to physical memory frames.
- Paging says when a process is stored in memory, that process will not be stored completely at one place in memory.
- The process can be in memory in different parts and each part of a process will be of same size i.e. process is divided in equal size of pages.
- These pages will be stored in main memory then the main memory must also be divided logically in different partitions.
- The size of main memory partition is known as frame and the size of frame will also be equal the size of page as these pages will be stored in main memory in frames i.e. physical memory or main memory is divided in same size of frames or page frames.

What are pages?

CPU

- Pages are fixed-size blocks of logical memory.
- The size of a page is typically a power of two, such as 4 KB or 8 KB.

What are frames?

- Frames are fixed-size blocks of physical memory that are the same size as pages.
- The physical memory is divided into these frames.

In paging process is divided in equal size of pages. Suppose a process P is divided into four equal size pages as follows :

Page-0 , Page–1, Page–2 and Page-3

- Now these pages need to be stored in main memory. This is the responsibility of memory management.

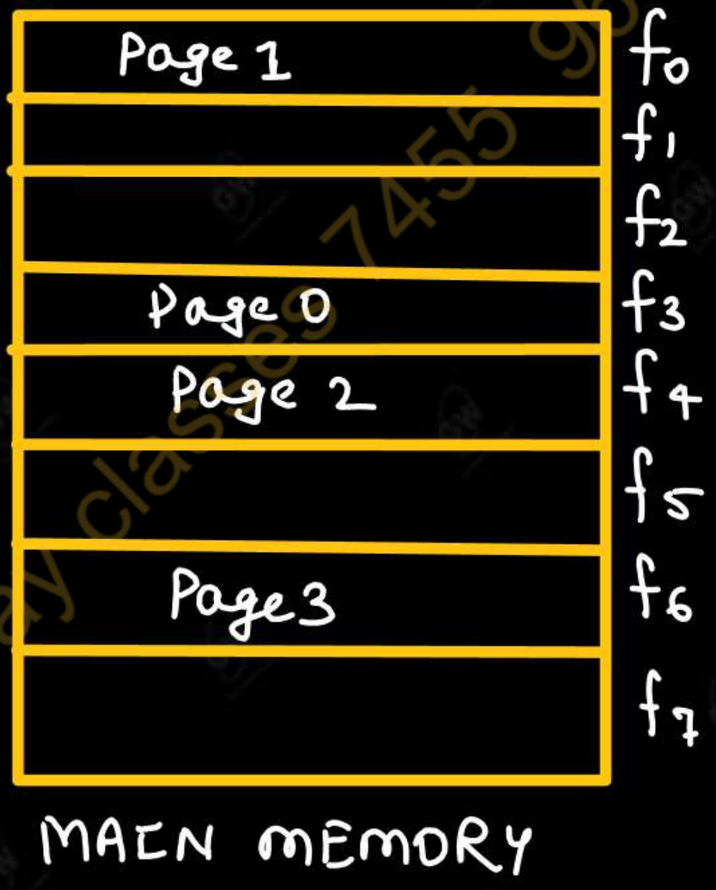
Physical

- Say main memory has eight frames of equal sizes as follows:

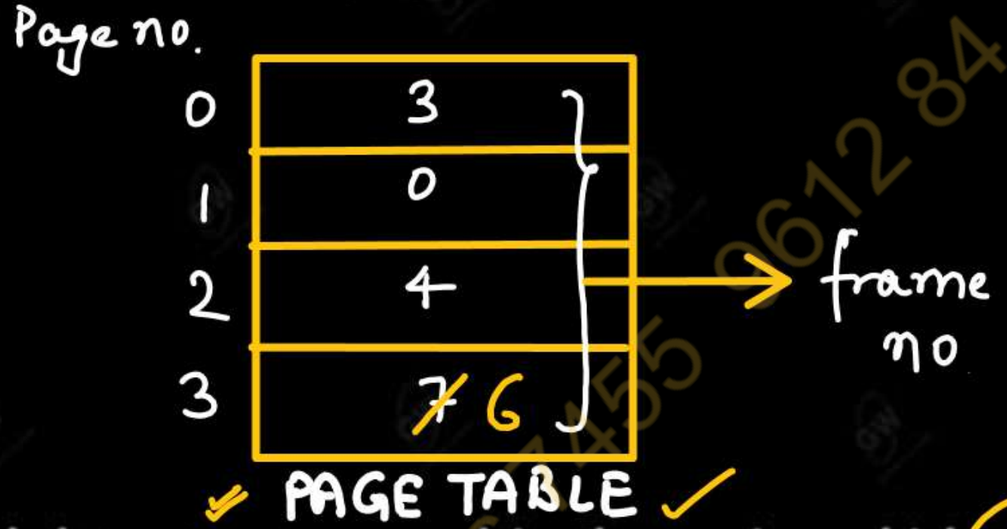
Size of each frame will be equal to the size of page

MAIN MEMORY OR PHYSICAL MEMORY

- According to paging, any page of a process can reside in any frame of main memory.
- Suppose a memory management technique store pages of process P in frames of main memory in this manner:

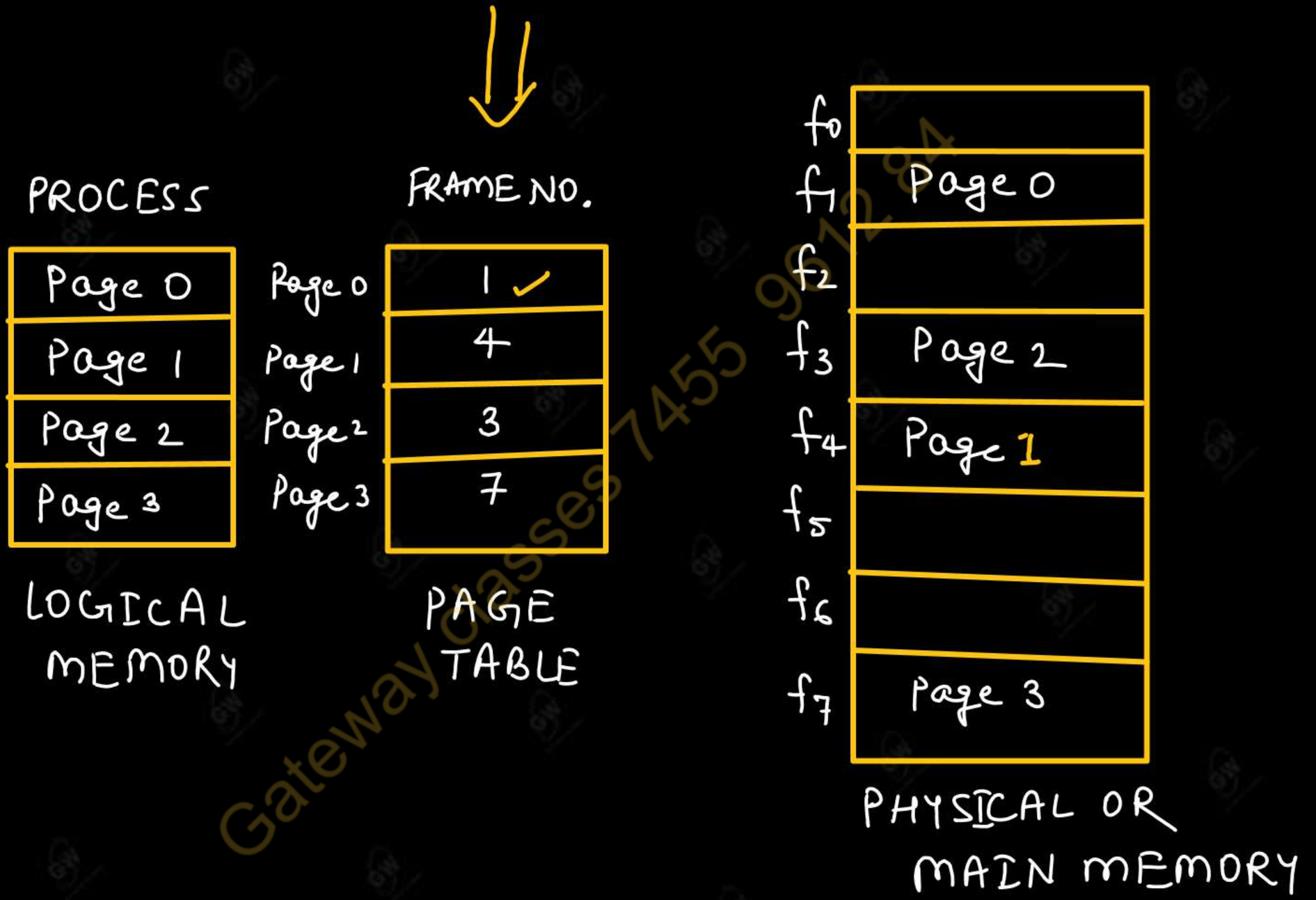


- Which page is stored in which frame of physical or main memory, that information is provided by the **Page Table**-

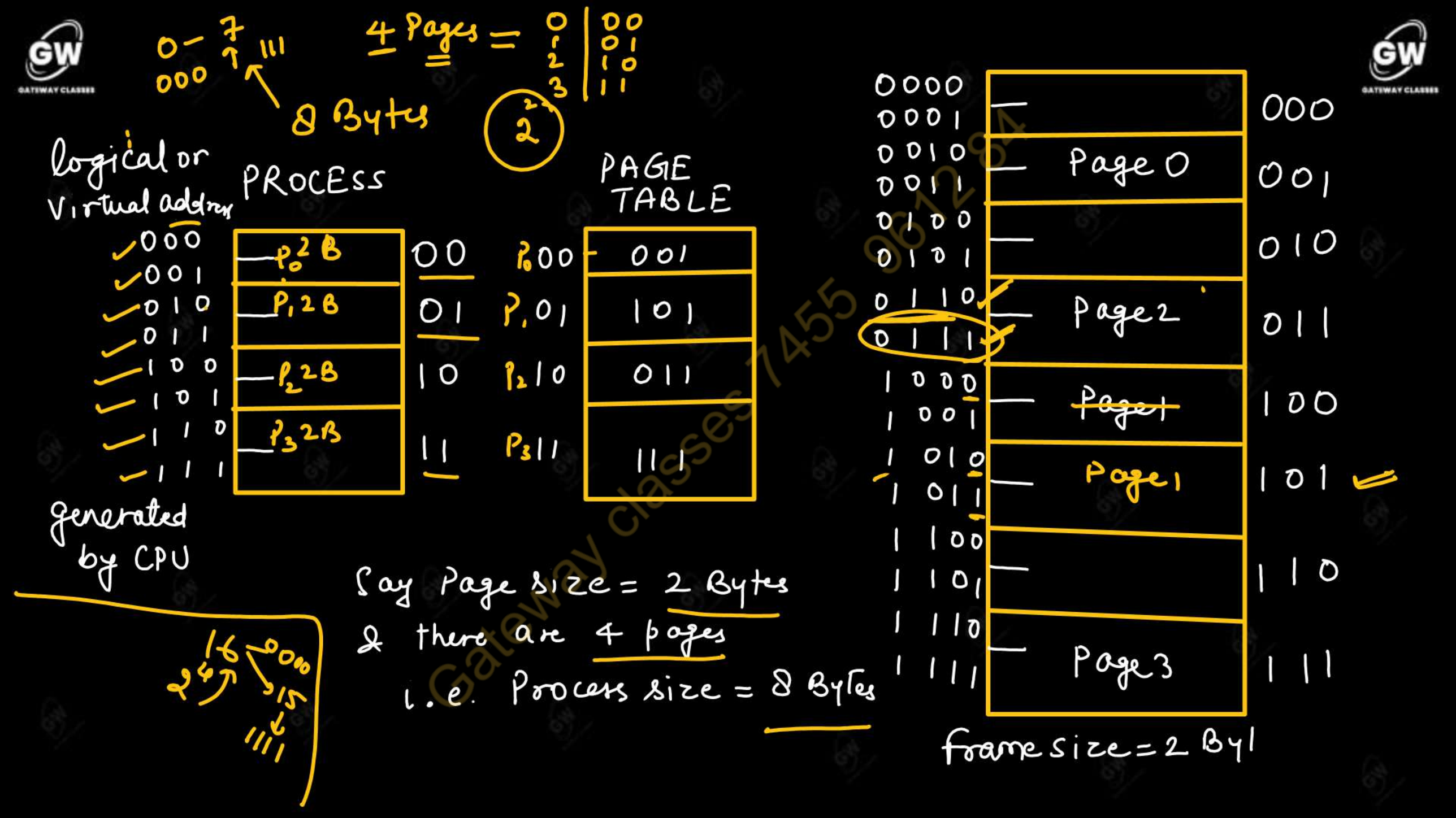


- A **page table** is a crucial data structure used in the paging which maps the logical (virtual) addresses used by a process to the physical addresses in the computer's RAM.
- The number of entries in the page table will be equal to the number of pages in the process.
- Each entry of the page table contain frame number .
- Each process running on the system has its own page table.
- Page table will contain the frame number i.e. which page of a process is stored in which frame number of physical memory.

- Let suppose there is an instruction in page 3, CPU will generate the reference of this instruction, that this instruction is required.
- The memory management technique will see that the required instruction is reside in which page. Say page number is 3 then memory management technique will refer the page table that in which frame number, page 3 resides.
- In the page table the page number three resides in frame number 7 and then move to the frame number 7 in the physical memory.
- Then the required byte where the required instruction is stored in frame number 7 will be accessed.
- In this way memory management technique provides the required instruction from the physical memory from the page table.
- Let us see example in next figure:-



- Where the page table is stored?
- Page table is also stored in main memory.
- If the size of page table is too small then that page table can also be stored in CPU registers itself.
- CPU cannot generate the physical address of main memory, CPU don't know where the process reside in main memory then how CPU will reach to the process?
- Lets take an example –



- The high-order bits of a virtual address (or logical address generated by the CPU) represent the page number.

- This page number is used to index into the page table to find the corresponding frame number in physical memory.
- Each entry in the page table contains the frame number of the corresponding physical memory frame where the page is stored.
- This frame number, combined with the offset, forms the physical address.
- The low-order bits of a virtual address represent the offset or page offset (i.e. the size of the page) within the page.

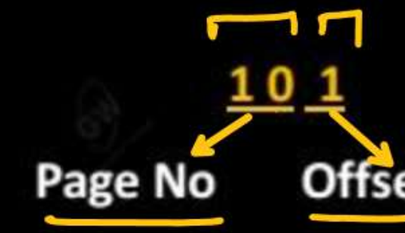
- The offset is added to the starting address of the frame to get the exact physical address.
- Say page size is of 4 bytes is each page address will be of 2 bits – 0 0, 0 1, 1 0 and 1 1
- Say page size is 256 bytes then number of bits in offset = 2 raised to the power 8 = 8 bits

Byte number
line number
displacement

- Say CPU demands logical address 1 0 1 then this logical address will be divided into two parts:

  (i) Page number &

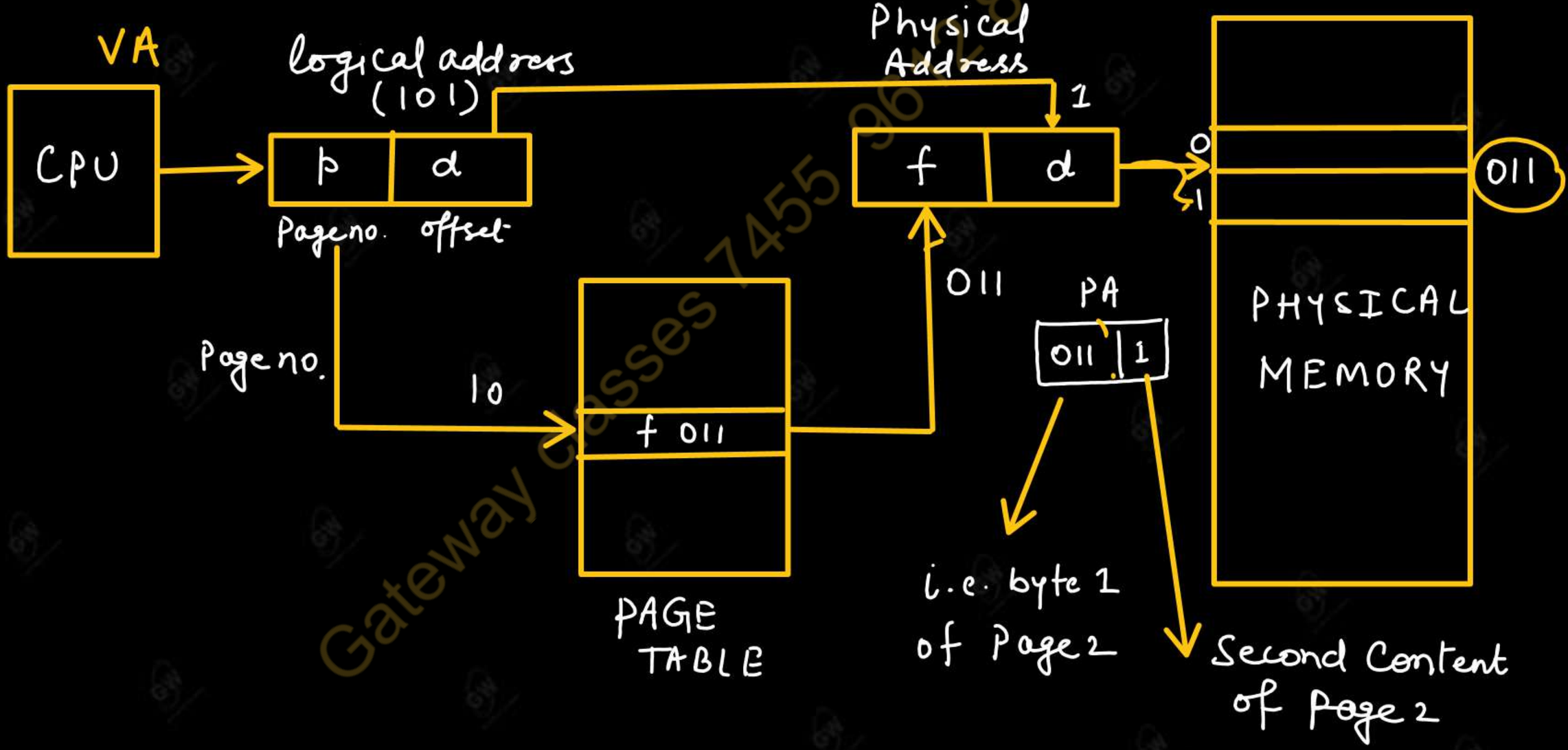
  (ii) Offset



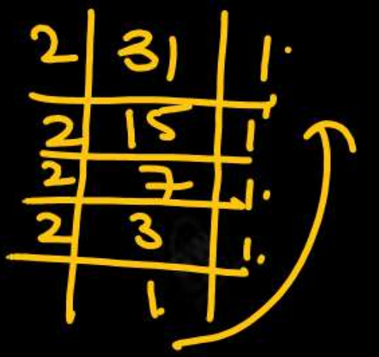
i.e. at page 1 0 the byte number 1

- Now memory management will see the page table for page 1 0. Through page table page 1 0 is at frame 0 1 1.
- Then will visit physical memory – frame 0 1 1 & select the byte 1 i.e. 1 0 1
- Let us see the next diagram -

FIG:- CONVERSION OF LOGICAL ADDRESS TO PHYSICAL ADDRESS THROUGH MMU



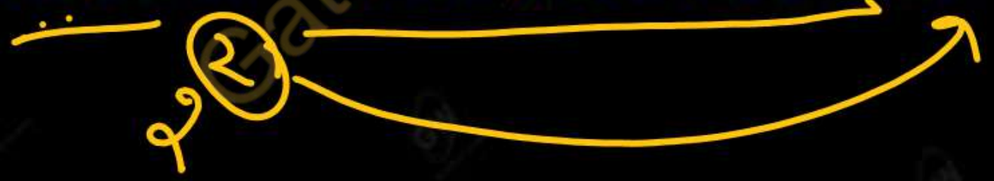
- When page table in main memory how CPU will reads to page table i.e. How CPU will know that where the page table is in memory?
- CPU will have a register which contain the base address of page table that register is known as PTBR (Page table base register).
- For each process it has its own page table, which helps in protection.
- A process can access only its page table through PTBR.



- If there are 16 Pages in a process then how many bits are required for numbering a page. – 4
- If a process size is 128 bytes and each page is of 4 Byte. Then how many pages?

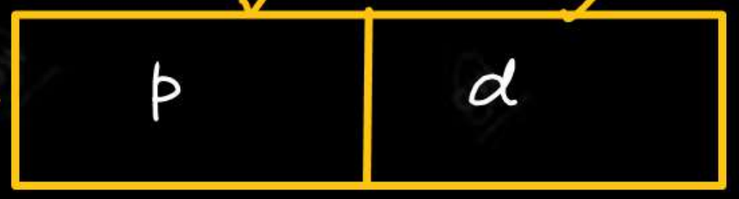
  128 / 4 = 32 pages and 5 bits are required for numbering these 32 pages.

- If page size = 4 bytes then how many bits in offset d? = 2.



- **Logical Address:**

Virtual CPU



no. of bits = $\log_2$ (no. of Pages in process)

no. of bits = $\log_2$ (size of page)

- **Physical Address:**

Main memory add or Actual Address

no. of bits = $\log_2$ (no. of frames in main memory)

- **Number of frames in main memory:** —

$$\frac{\text{main memory size}}{\text{Page size or frame size}}$$

- Number of pages in process: $\frac{\text{Process size}}{\text{Page size}}$

- Page table size = number of entries in page table * 1 entry size
  = number of pages in process * 1 entry size
  = number of pages in process * (frame number + extra bits) {if entry size is not given}

- What is logical address space?
- Collection of logical address i.e. process size.
- What is physical address space?
- Collection of physical addresses in main memory.

Paging

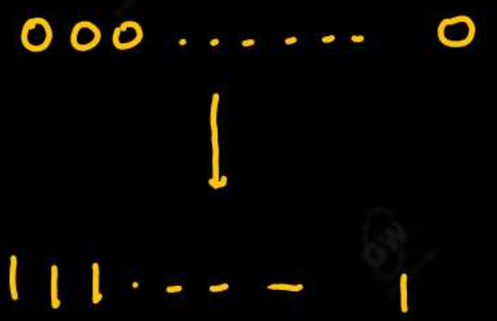
Example 1: Consider a page memory system with logical address of 26 bits and physical address of 32 bits. The page size is 2KB. Further consider that one page table entry size is 4 bytes. Find out the following:-

1. No. of bits in page offset = $2KB = 2^1 \times 2^{10} = 2^{11}$ (11)
2. No. of pages in process = 2^{15}
3. No. of bits in page number = (15)
4. No. of frames in physical memory = 2^{21}
5. No. of bits for frame number = 21 bits
6. Page table size

Solution:

- If logical address is of 26 bits then logical address space or process size will be 2^{26} bytes.
- If physical address is of 32 bits then physical address space will be of 2^{32} bytes size.
- Page size = 2KB = 2^{11} bytes i.e. offset will be of 11 bits i.e. number of bits in page number = 26 - 11 = 15.
- i.e. bits required for frame = 32 – 11 = 21 bits

1. Number of bits in page offset = 11
2. Number of pages in process = 2^{26} / 2^{11} = 2^{15}
3. Number of bits in page number = 15
4. Number of frames in Physical memory = 2^{21} frames
5. Number of bits for frame number = 21



6. Page table size –

Number of entries in page table = number of pages in process

page table size = number of pages in process * 1 Page table entry size

= number of pages in process * (frame number bits + Extra bits)

$= 2^{15} * 4$ bytes

$= 2^{15} * 2^2$

$= 2^{17}$ $= 2^7 * 2^{10} = 2^7$ KB

= 128 KB = 128 KB

Example 2: A system has 64 - bit virtual address and 43 - bits physical address. If the pages are 8 KB in size, then find out the number of bits required for VPN and PPN. (VPN = Virtual page no. or logical page no., PPN = Physical frame no. or Physical page no.

Solution:-

VIRTUAL ADDRESS / LOGICAL ADDRESS

PHYSICAL ADDRESS

Page size = 8 KB
$= 2^3 \times 2^{10}$
$= 2^{13}$
(13 Bits) offset

Page Size = 8 KB = 2^{13}

i.e. Virtual Page no = 51 bits

& PPN = 30 bits

Example 3: Consider a logical address space of 8 pages of 1024 words, each mapped onto a physical memory of 32 frames then:

Process size

No of Pages = 8

Size of each page = 1024 words = 2^{10}

(a) How many bits are in logical address?

(b) How many bits are in physical address?

Logical Add (13)

p 3	d 10

Solution:-

- The logical address is divided into two parts – (i) page address or page number (ii) offset
- The page address must be able to reference 8 (or 2^3) distinct pages. This requires 3 address bits.
- The offset must be able to reference 1024 (2^{10}) distinct words on each page. This requires 10 address bits. Therefore, 3 + 10 = 13 bits are required for logical address.
- The physical address is also divided into two parts:- Frame address (frame number) and Offset. The number of frame is 32 (or 2^5), so it will require 5 bits.
- The size of the frame is same as of the page, therefore this will require 10 bits, therefore 5 + 10 = 15 bits are required for physical address.

Physical Add

f 32 = 2^5	d	10

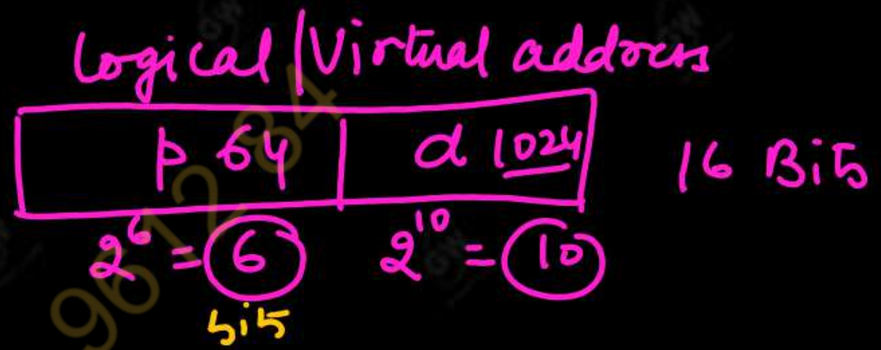
5 10

= 15 bits Ans

Example 4: Consider a logical address space of 64 pages of 1024 words each, mapped onto a physical memory of 32 frames.

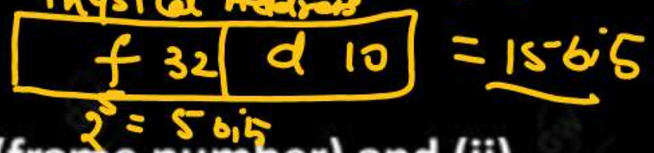
(a) How many bits are there in the logical address?

(b) How many bits are there in the physical address?



Solution:-

- The logical address is divided into two parts – (i) the page address or page number (ii) the offset
- The page address must be able to reference 64 (or 2^6) distinct pages. This requires 6 address bits.
- The offset must be able to reference 1024 (2^{10}) distinct words on each page. This requires 10 address bits. Therefore, 6 + 10 = 16 bits are required for logical address.

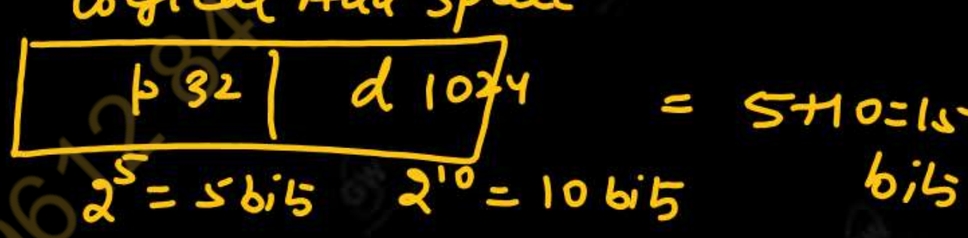


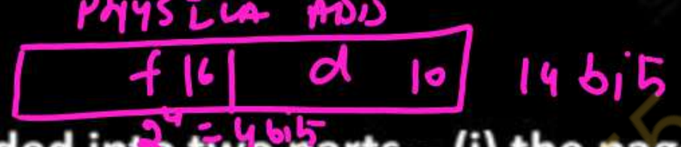
- The physical address is also divided into two parts:- (i) the frame address (frame number) and (ii) the Offset. The number of frame is 32 (or 2^5), so it will require 5 bits.
- The size of the frame is same as of the page, therefore this will require 10 bits, therefore 5 + 10 = 15 bits are required for physical address.

Example 5: Consider a logical address space of 32 pages with 1024 words per pages, mapped onto a physical memory of 16 frames.

(a) How many bits are required in the logical address?

(b) How many bits are required in the physical address?





Solution:-

- The logical address is divided into two parts – (i) the page address or page number (ii) the offset
- The page address must be able to reference 32 (or 2^5) distinct pages. This requires 5 address bits.
- The offset must be able to reference 1024 (2^{10}) distinct words on each page. This requires 10 address bits. Therefore, 5 + 10 = 15 bits are required for logical address.
- The physical address is also divided into two parts:- (i) the frame address (frame number) and (ii) the Offset. The number of frame is 16 (or 2^4), so it will require 4 bits.
- The size of the frame is same as of the page, therefore this will require 10 bits, therefore 4 + 10 = 14 bits are required for physical address.

Example 6: In paging, why page size is kept in power of 2?

Answer :

- Page sizes in memory management are typically kept as powers of two (e.g., 4 KB, 8 KB, 16 KB) due to several key reasons related to computational efficiency and system design:

1. Alignment and Addressing Efficiency:

- **Binary Arithmetic:** Computers use binary arithmetic. Using powers of two makes it easier to calculate page offsets and addresses. For example, with a 4 KB page (2^12 bytes), the page number and offset can be directly extracted from an address by splitting it into its higher and lower bits, which is computationally efficient.
- **Alignment:** Pages aligned to power-of-two boundaries ensure that calculations for page boundaries are simplified. For instance, determining if an address is at the start of a page is a matter of checking if the lower bits are zero.

2. Simplified Hardware Design:

- **TLB and Cache Optimizations:** Translation Lookaside Buffers (TLBs) and cache designs benefit from power-of-two page sizes, simplifying indexing and lookups. This reduces complexity in the hardware design and increases the speed of memory operations.
- **Modular Arithmetic:** Hardware can use bit masking instead of more complex arithmetic operations to determine page boundaries, which is faster and more efficient.

3. Memory Management Efficiency:

- **Fragmentation Reduction:** Using pages that are powers of two can help in reducing internal fragmentation within pages. Although it doesn't eliminate fragmentation, it standardizes page sizes and helps the operating system manage memory more predictably.
- **Simplicity in Page Table Management:** The operating system can more easily manage page tables when all pages are of a uniform size, and the use of power-of-two sizes aligns with this uniformity.

❑ Benefits of Paging -

1. Paging allows the operating system to allocate memory more flexibly and efficiently.

2. It reduces the problem of external fragmentation, where free memory is scattered in small blocks.

3. It supports for Virtual Memory. Paging is fundamental to implementing virtual memory, where the system uses disk space to simulate additional RAM. This allows programs to use more memory than is physically available, improving multitasking and overall system performance.

AKTU PYQs

Q.1. Explain the concept of paging. Also, explain paging hardware support using TLB with suitable diagram. 2022 - 23, 10 Marks

Q.2. Explain paging with example. Differentiate paging and segmentation. 2018 - 19, 10 Marks

Q.3. Explain implementation of paging technique with respect to address mapping page table, page size and hardware support for paging. 2017 - 18, 10 Marks

Q.4. Why page size is kept in power of two. 2016 - 17, 5 Marks

OPERATING SYSTEM

Today's Target -

- Paging : Hardware Support with TLB
- Time Required in Paging
- AKTU PYQs

Paging : Hardware Support

- The operating system allocates a page table for each process.
- A pointer to the page table is stored with the other register values (like the instruction counter) in the PCB.
- When the dispatcher is told to start a process, it must reload the user registers and define the correct hardware page table.
- The hardware implementation of a page table can be done in several ways.
- The page table is implemented as a set of dedicated registers in the simplest case.
- These registers should be built with high-speed logic to make the paging address translation efficient.
- Every memory access must go through the paging map, so efficiency is a major consideration. The CPU dispatcher reloads these registers, just as it reloads other registers.
- The use of registers for the page table is satisfactory if the page table is reasonably small (with fewer entries, 256 entries).

- Most computers, however, allow the page table to be very large (for example 1 million eateries in a page table).
- For these machines, the use of fast registers to implement the page table is not feasible rather, the page table is kept in main memory, and a page–table base register (PTBR) points to the page table. The problem with this approach is the time required to access a user's memory location.
- If we want to access location i, we must first index to the page table, using the value in the PTBR offset by the page number for i.
- This is combined with the page offset to produce the actual address.
- We can then access the desired place in memory.
- With this scheme, two memory accesses are needed to access a byte (one for the page table entry, one for the byte).
- Thus, memory access is allowed by a factor of 2.
- This delay would be intolerable under most circumstances.

- The standard solution to this problem is to use a special, small, fast-lookup hardware cache, called a translation look-aside buffer (TLB).
- The TLB is associative high-speed memory.
- Each entry in the TLB consists of two parts: a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- If the item is found, the corresponding value field is returned.
- The search is fast; the hardware, however, is expensive.

- The TLB is used with page tables in the following way -
  - The TLB contains only a few of the page table entry (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 the memory.
  - If the page number is not in found in TLB (known as a TLB miss) a memory reference to the page table, must be made.
  - When a frame number is obtained, we can use it to access the memory.

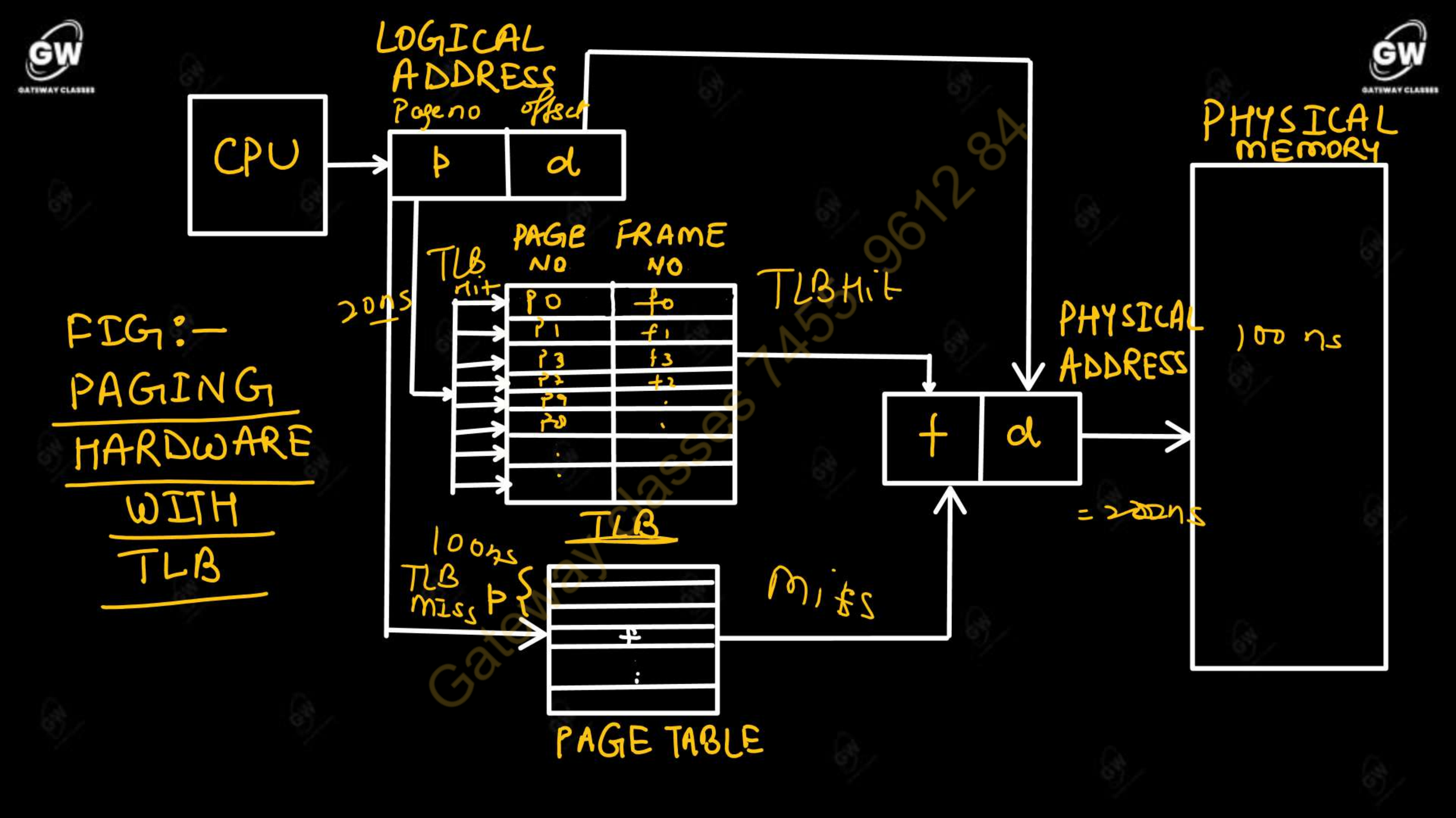


FIG:- PAGING HARDWARE WITH TLB

- In addition, 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. (any replacement algorithm)
- The percentage of times that a particular page number is found in the TLB is called the hit ratio. An 80% hit ratio, for example, means that we find the desired page number in the TLB 80 percent of the time.
- If it takes 20 ns to search the TLB and 100 ns to access memory, then a mapped–memory access takes 120 ns when the page number is in TLB.
- If we fail to find the page number in TLB (20 ns) then we first must access the memory for the page table and frame number (100 ns) and then access the desired byte in memory (100 ns) for a total of 220 ns. (20 + 100 + 100).

X TLB = 20 ns. X

mm = 100 ns.

100 + 100 + 20

TLB (20) + mm (PT) 100 ns + mm (100 ns)

=220 ns.

TLB (20ns) + mm (100ns)

=120 ns

Time Required in Paging

- In the paging scheme, if the CPU wishes to access any content then how much time it will take?
- CPU will generate a logical address and then will see the page table in main memory i.e. one memory access to access the page table, from the page table will get the frame number which will be added to the offset to get the physical address, the access that particular address of main memory to get that particular content at memory location.
- The page table is generally stored in the main memory.
- In this way, 2 memory accesses will be required to access the content of the main memory through a paging scheme.

Effective memory access time = $2 \times t_{mm}$

{ One for page table and one for content }

- If page table is very small and stored in register, then -

Effective Memory Access Time (EMAT) = One access of register for table + one access of main memory for content

{ But register access is very small in comparison to main memory and can be negligible.}

EMAT = t_{mm} { Register access time as compared to t_{mm} is negligible}

- But always page table will not be small. A process may have several pages which will increase the size of the page table. Then how the performance will be improved?
- If a process size is big and has a large number of pages then it may be observed that all pages will not be required at the same time, some pages may be required very frequently and entry of those frequently asked pages may be done in very high-speed memory TLB (Translation look aside buffer) (new hardware).
- This is used to improve the performance.
- TLB is a small and fast memory hardware that is used to store frequently used Page Table entries.
- Now, the CPU generates the logical address and the required page will be searched in TLB if that page entry is found in TLB i.e. in minimum access time of TLB, the physical address of main memory is found, then no need to access the main memory to see the page number.
- In this way, the main memory will be referred to only once to see the desired content in the main memory.

- In this way, to access the desired content in memory one access of TLB and one access of main memory will take place where TLB access time is very less and improve the overall performance.
- If the page entry is found in TLB that is TLB hit and if the page table entry is not found in TLB i.e. TLB misses i.e. two memory accesses will take place to access a particular content from the main memory.

TLB Hit

TLB Miss

$$\text{Effective Memory Access Time (EMAT)} = H \times (t_{TLB} + t_{mm}) + (1 - H)\ (t_{TLB} + 2\ t_{mm})$$

- **What happens when a process tries to access a page that is not present in the main memory or the TLB?**
  - When a process generates a virtual address that does not correspond to a page in the main memory or in the TLB, a page fault occurs. This means that the operating system needs to retrieve the missing page from secondary storage (e.g., hard disk) and bring it into the main memory. Once the page is in the main memory, the TLB is updated to include its corresponding PTE.
- **What are some standard replacement policies used by TLBs?**
  - TLBs typically use either a Least Recently Used (LRU) or a First-In-First-Out (FIFO) replacement policy. LRU means that the TLB entry that has not been used for the longest time is replaced, while FIFO means that the TLB entry that was first inserted is replaced. Other possible policies include Most Recently Used (MRU), Most Frequently Used (MFU), and Random.

FIFO | OPT | LRU

Example 1:- Main memory access time is 300 ns. TLB access time is 20 ns and TLB hit ratio is 80%.

Calculate effective memory access time (EMAT). (Ans = 380 ns)

Solution : -

Main memory access time = 300 ns

TLB access time = 20 ns

TLB hit ratio = 80%

this line is for next example.

$0.75 \times (0 + 200) + (1 - .75)(0 + 2 \times 200)$

Effective Memory Access Time (EMAT) $= H \times (t_{TLB} + t_{mm}) + (1 - H)\ (t_{TLB} + 2\, t_{mm})$

Effective memory access time = 0.8 * (20 + 300) + (1 – 0.8) (20 + 2 * 300)

= 380 ns

Example 2:- Consider a paging system with the page table stored in memory.

(a) If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

(b) If we add TLBs and 75% of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes zero time if the entry is there.)

Solution:-

(a) A paged memory reference requires 2 memory accesses: one to get the frame number for the page number from memory and accessing the data. So if a memory reference takes 200 ns then a paged memory reference will take 400 ns.

(b) If associative registers (TLBs) are added and the hit ratio is 75% then effective memory access time is given as –

(2 x 200)

Effective Memory Access Time (EMAT) = 0.75 * 200 + 0.25 * 400 = 250 ns

AKTU PYQs

Q.1. Explain the concept of paging. Also, explain paging hardware support using TLB with suitable diagram. **2022 - 23, 10 Marks**

Q.2. *Explain the terms hit ratio and miss ratio. On a simple paged system, associative registers hold the most active page entries and the full – page table is stored in main memory. If references satisfied by associative registers take* 100*ns, and references through main memory page table takes* 180 *ns, what must the hit ratio be to achieve an effective access time of* 125 *ns*? **2021 - 22, 10 Marks**

Q.3. *Explain implementation of paging technique with respect to address mapping page table, page size and hardware support for paging.* **2017 - 18, 10 Marks**

Q.4. *What hit ratio is required to reduce the effective memory access time from* 300 *nanoseconds without using TLB to* 250 *nanoseconds with TLB. The TLB access time is* 60 *nanoseconds.* **2016 - 17, 10 Marks**

OPERATING SYSTEM

Today's Target

- Page Replacement Algorithms (LRU , OPTIMAL , FIFO)
- AKTU PYQs

Page Replacement Algorithms

- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
- Page replacement becomes necessary when a page fault occurs and no free page frames are in memory.
- However, another page fault would arise if the replaced page is referenced again. Hence it is important to replace a page that is not likely to be referenced in the immediate future.
- Page replacement algorithms are techniques used in operating systems to manage memory efficiently when the virtual memory is full.
- When a new page needs to be loaded into physical memory, and there is no free space, these algorithms determine which existing page to replace.

Page Replacement Techniques

❑ Least Recently Used (LRU)

- This policy replaces the page which has the longest backward distance.
- In this algorithm, page will be replaced which is least recently used.

❑ Optimal Page Replacement (OPT)

- This policy replaces the page with the longest forward distance.
- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

❑ First In First Out (FIFO)

- This policy replaces the page which has been is the memory for the longest time.
- This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.
- When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider the following reference string: 0, 1, 2, 4, 2, 3, 7, 2, 1, 3, 1

How many page faults will occur for : (a) LRU (b) OPT (c) FIFO

Assuming 3 frames (Pages) in each case and initially frames are empty.

Solution: Reference String ---------->

	PAGE FRAMES	0	1	2	4	2	3	7	2	1	3	1	HIT RATIO	NO. OF PAGE FAULTS
1. LRU (longest backward distance)	A	0	0	0	4	4	4	7	7	7	3	3	3/11	8
	B		1	1	1	1	3	3	3	1	1	1		
	C			2	2	2	2	2	2	2	2	2		
	Page Fault	*	*	*	*	H	*	*	H	*	*	H		

	PAGE FRAMES	0	1	2	4	2	3	7	2	1	3	1	HIT RATIO	NO. OF PAGE FAULTS
OPT(longest forward distance)	A	0	0	0	4	4	3	7	7	7	3	3	4/11	7
	B		1	1	1	1	1	1	1	1	1	1		
	C			2	2	2	2	2	2	2	2	2		
	Faults	*	*	*	*	H	*	*	H	H	*	H		

	PAGE FRAMES	0	1	2	4	2	3	7	2	1	3	1	HIT RATIO	NO. OF PAGE FAULTS
FIFO (in the memory for the longest time)	A	0	0	0	4	4	4	4	2	2	2	2	2/11	9
	B		1	1	1	1	3	3	3	1	1	1		
	C			2	2	2	2	7	7	7	3	3		
	Faults	*	*	*	*	H	*	*	*	*	*	H		

Example 2: Consider the following reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

How many page faults will ocean for FIFO, OPT and LRU page releasement algorithm, when the page frames are 3.

Solution: Reference String ---------->

	PAGE FRAMES	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	HIT RATIO	NO. OF PAGE FAULTS
FIFO (page in memory for longest time)	A	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	1	7	7	7	5/20	15
	B		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0		
	C			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1		
	Page Fault	*	*	*	*	H	*	*	*	*	*	*	H	H	*	*	H	H	*	*	*		

The FIFO replacement algorithm is easy to understand and program. However its performance is not always good.

- The optimal page replacement algorithm is difficult to implement because future knowledge of the reference string. As a result the optimal algorithm is used mainly for the comparison studies.

Reference String---------->

	PAGE FRAMES	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	HIT RATIO	NO. OF PAGE FAULTS
OPT (longest forward distance i.e. replace the page that will not be used for the longest period of time)	A	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7	11/20	9
	B		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0		
	C			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1		
	Page Faults	*	*	*	*	H	*	H	*	H	H	*	H	H	*	H	H	H	*	H	H		

Reference String ---------->

	PAGE FRAMES	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	HIT RATIO	NO. OF PAGE FAULTS
LRU (the page that has not been used for longest period of time)	A	7	7	1	2	2	2	0	4	4	4	0	0	0	1	1	1	1	1	1	1	8/20	12
	B		0	0	0	0	0	0	0	0	3	3	3	3	3	2	0	0	0	0	0		
	C	-	-	1	1	0	3	3	2	2	2	2	2	2	2	2	2	0	7	7	7		
	Page Faults	*	*	*	*	H	*	H	*	*	*	*	H	H	*	H	*	H	*	H	H		

Belady's Anomaly

- A page fault occurs when a page is not found in the memory and needs to be loaded from the disk.
- If a page fault occurs and all memory frames have already been allocated, the replacement of a page in memory is required on the request of a new page. This is referred to as demand paging.
- The choice of which to replace is specified by page replacement algorithms.
- The commonly used page replacements algorithms are FIFO, LRU optional page replacement algorithms etc.
- Generally, on increasing of number of frames to a process's virtual memory, its execution becomes faster as fewer page faults occur.
- Sometimes the reverse happens i.e. more page faults occurs when more frames are allocated to a process. This most unexpected result is termed Belady's Anomaly.

- Be lady's Anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.
- This phenomenon is commonly experienced in FIFO, second choice page replacement algorithm and random page replacement algorithms.
- The other commonly used page replacement algorithms are optional and LRU, but Belady's anomaly can never occur in these algorithms for any reference string as they belong to a class of stack – based page replacement algorithms.
- FIFO does not follow a stack page replacement policy and therefore suffers Belady's Anomaly.

Example 3:- Find the page fault for following reference string for 3 frame and 4 frame using FIFO, LRU and OPT page replacement algorithm: 1, 2, 3, 4, 1, 2, 5, 6, 3, 1, 3, 2, 4, 5

Discuss whether Belady's anomaly exists here or not.

Solution:- For 3 Frames:-

Reference String ---------->

FIFO (page in memory for longest time)	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
	A															1/14	13
	B																
	C																
	PAGE FAULTS																

LRU (the page that has not been used for longest period of time)	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
	A															1/14	13
	B																
	C																
	PAGE FAULT																

OPT(longest forward distance i.e. replace the page that will not be used for the longest period of time)	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
	A															5/14	9
	B																
	C																
	PAGE FAULTS																

For 4 Frames:-

	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
FIFO(page in memory for longest time)	A																11
	B																
	C																
	D																
	PAGE FAULT																

For 4 Frames:-

LRU(the page that has not been used for longest period of time)	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
	A																11
	B																
	C																
	D																
	PAGE FAULTS																

For 4 Frames:-

	PAGE FRAMES	1	2	3	4	1	2	5	6	3	1	3	2	4	5	HIT RATIO	NO. OF RAGE FAULTS
OPT(longest forward distance i.e. replace the page that will not be used for the longest period of time)	A																
	B																
	C																
	D																
	PAGE FAULTS																

No. of Frames	Page replacement algorithm	No. of Page Faults	Belady's Anomaly is observed (yes/no)
3	FIFO	13	No
4	FIFO	11	

Example 3:- Consider the following reference - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Discuss whether Belady's anomaly exists here or not. (frame = 3, frame = 4)

Solution:- For 3 Frames:-

Reference String ---------->

	PAGE FRAMES	1	2	3	4	1	2	5	1	2	3	4	5	HIT RATIO	NO. OF RAGE FAULTS
FIFO(page in memory for longest time)	A	1	1	1	4	4	4	5	5	5	5	5	5	3/12	9
	B		2	2	2	1	1	1	1	1	3	3	3		
	C			3	3	3	2	2	2	2	2	4	4		
	PAGE FAULTS	*	*	*	*	*	*	*	H	H	*	*	H		

For 4 Frames:-

Reference String ---------->

FIFO(page in memory for longest time)	PAGE FRAMES	1	2	3	4	1	2	5	1	2	3	4	5	HIT RATIO	NO. OF RAGE FAULTS
	A	1	1	1	1	1	1	5	5	5	3	4	4	2/12	10
	B		2	2	2	2	2	2	1	1	1	1	5		
	C			3	3	3	3	3	2	2	2	2	2		
	D				4	4	4	4	4	4	3	3	3		
	PAGE FAULTS	*	*	*	*	H	H	*	*	*	*	*	*		

No. of Frames	Page Replacement Algorithm	No. of Page Faults	Belady's Anomaly is observed (yes/no)
3	FIFO	9	Yes
4		10	

How can Belady's Anomaly be rectified?

- Belady's anomaly refers to the counterintuitive situation in certain page replacement algorithms, specifically FIFO (First-In-First-Out), where increasing the number of page frames increases the number of page faults.
- This anomaly can be rectified using more advanced page replacement algorithms that do not suffer from this issue. Here are some strategies and algorithms that help avoid Belady's anomaly:

1. Least Recently Used (LRU):

- LRU replaces the page that has not been used for the longest period of time. It is based on the assumption that pages used recently will likely be used again soon.

2. Optimal Page Replacement (OPT):

- This theoretical algorithm replaces the page that will not be used for the longest time in the future. While it's not feasible in practice because it requires future knowledge, it serves as a benchmark for other algorithms.

3. Least Frequently Used (LFU):

- LFU replaces the page that has the smallest count of accesses over a time window.
- This algorithm assumes that pages with lower access frequency are less likely to be needed in the future.

4. Clock (Second Chance) Algorithm:

- The clock algorithm is a more efficient approximation of LRU.
- It uses a circular list (like a clock) and gives each page a second chance before replacing it, ensuring better performance than FIFO.

5. Adaptive Replacement Cache (ARC):

- ARC dynamically balances between LRU and LFU, adapting to changing access patterns.
- It maintains two lists to track recently used and frequently used pages, allowing it to adjust its behavior based on workload.

6. Enhanced Second Chance (ESC):

- Also known as the "Nth Chance" algorithm, ESC extends the clock algorithm by giving pages more than two chances before replacement.
- It adds more complexity but can improve performance by reducing page faults.

7. Random Replacement:

- Although not always optimal, random replacement can sometimes avoid pathological cases that lead to Belady's anomaly.
- It randomly selects a page for replacement, avoiding systematic issues inherent in FIFO.

- By using these more sophisticated algorithms, systems can effectively reduce the occurrence of Belady's anomaly and improve overall page replacement efficiency.

AKTU PYQs

Q.1. *Explain the concept of demand paging. Consider the given references to the following pages by a program:* 0, 9, 0, 1, 8, 1, 8, 7, 8, 7, 1, 2, 8, 2, 7, 8, 2, 3, 8, 3

How many pages faults will occur if the program has three – page frames available to it and uses:

(i) FIFO replacement (ii) LRU replacement (iii) Optimal replacemen 2022-23, 10 Marks

Q.2. *Illustrate the following page – replacement algorithms.*

(i) FIFO (ii) Optimal Page Replacement

Use the reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 *for a memory with three frames.*

2021-22, 10 Marks , 2018-19, 10 Marks

Q.3. *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

How many page faults would occur for the optimal page replacement algorithm, assuming three frames and all frames are initially empty. 2020-21, 10 Marks

Q.4. *What do you mean by Belady's anomaly? Which algorithm suffers from Belady's anomaly and how can it be rectified?* 2018-19, 10 Marks

Q.5. *How many page faults would occur for the following reference string for four frames using LRU page replacement algorithm* 1, 2, 3, 4, 5, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

2017-18, 10 Marks

Q.6. *Consider the following reference string* 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
How many page faults will occur for: (i) *FIFO* (ii) *LRU page replacement algorithm? Assuming three and four frames in each case and frames are initially empty.*

2015-16, 10 Marks

Q.7. *How many page faults would occur for the following reference string for four page frames using LRU and FIFO algorithms*: 1, 2, 3, 4, 5, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2

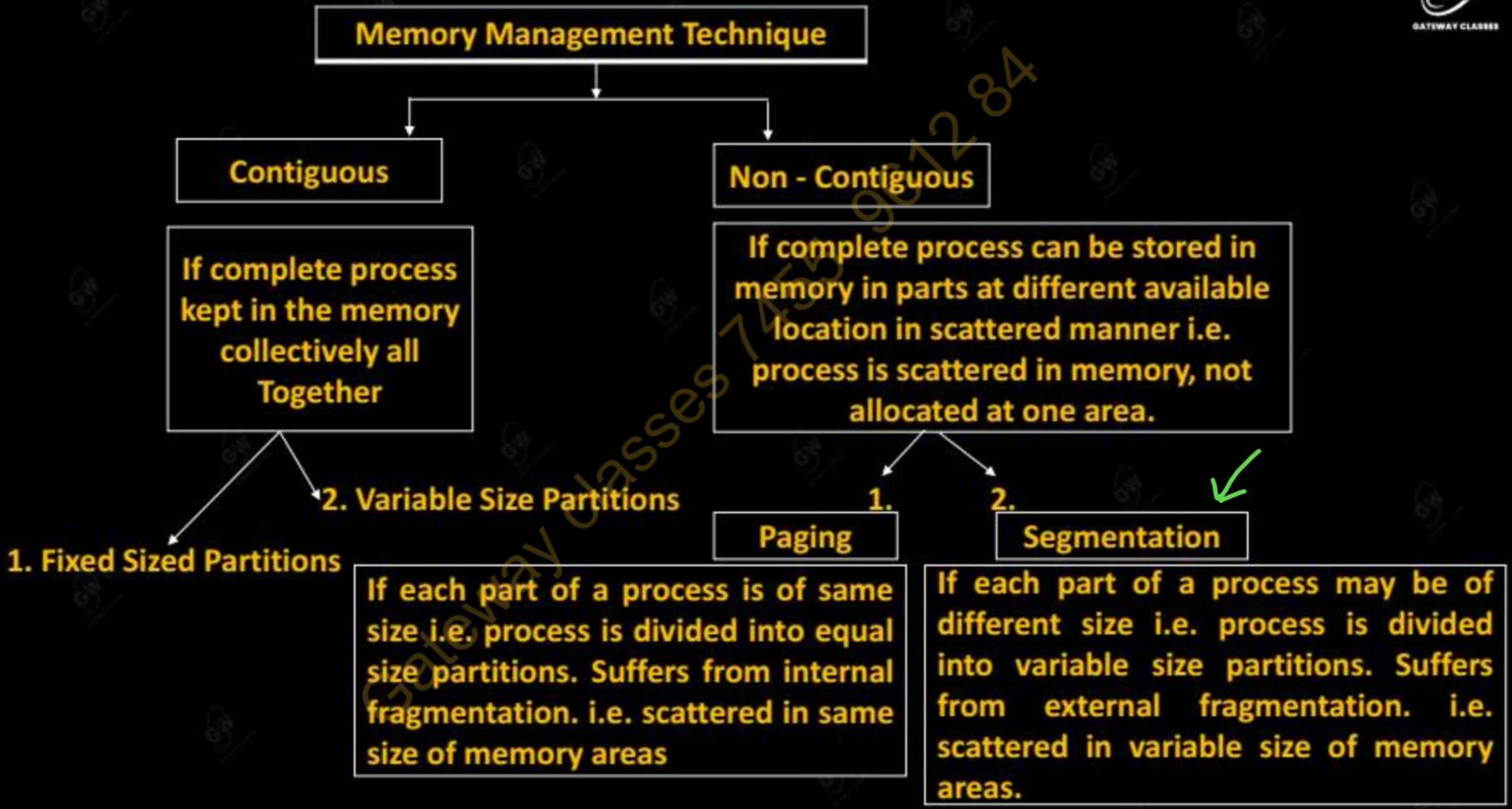
2014-15, 10 Marks

OPERATING SYSTEM

Today's Target -

- ❑ Segmentation
- ❑ Paged Segmentation
- ❑ AKTU PYQs

❑ We known that : -



SEGMENTATION

- In paging, we divide a process into fixed-size pages, and main memory is also divided into fixed-size frames as these fixed-size pages will be stored in fixed-size frames in physical memory in a scattered manner.
- Paging will have fixed size partitions and segmentation will have variable size partitions.
- An important aspect of memory management that became unavoidable with paging is the separation of the user's view of memory from the actual physical memory.
- The user's view of memory is not the same as the actual physical memory.
- The user's view is mapped onto physical memory.
- This mapping allows differentiation between logical memory and physical memory.
- On the other hand, users prefer to view memory as a collection of variable-sized segments, with no necessary ordering among segments.

- Consider how a user or a programmer thinks of a program when writing it.
- The programmer thinks of it as a main program with a set of methods, procedures, or functions. or subroutines or modules
- It may also include various data structures objects, arrays, stacks, variables, and so on.
- Each of these modules or data elements is referred to by name.
- Users or programmers talk about "the stack", "the math library", and "the main program", without caring about what addresses in memory these elements occupy.
- The user or programmer is not concerned with whether the stack is stored before or after the main function.

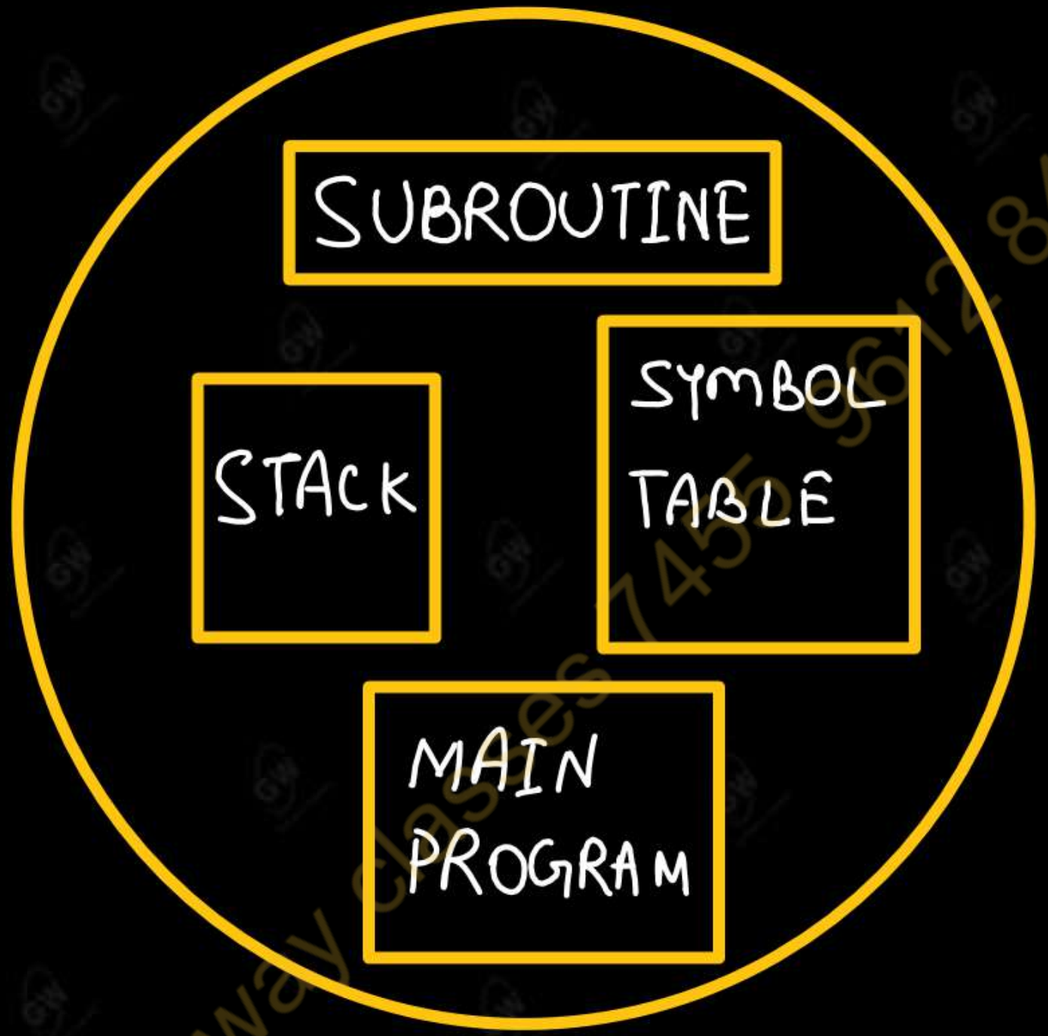


FIG: USER'S VIEW OF A PROGRAM

- Segmentation is a memory management technique that divides a program's memory into different segments based on logical divisions such as functions, data, and stack. Each segment is a contiguous block of addresses that is managed independently.
- Segmentation is dividing the process into logically related partitions. These partitions are known as segments.
- Each of these segments is of variable length; the length is intrinsically defined by the purpose of the segment in the program and segments are scattered in the physical memory.
- Segmentation supports the user's view of memory.
- A logical address space is a collection of segments.
- For simplicity of implementation, segments are numbered and are referred to by a segment number. Thus, a logical address consists of a two tuple.

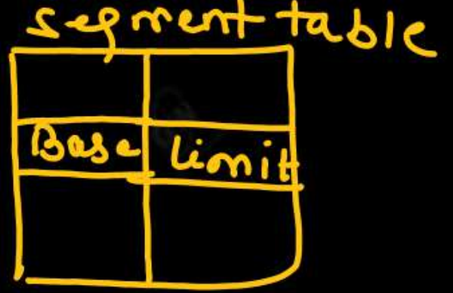
LA: <Segment-number, offset>

Paging : | p | d |

Segmentation : | s | d |

SEGMENTATION HARDWARE

- The segments are not of the same size then memory will also not be divided as divided in paging. In paging page size is equal to the frame size of main memory.
- As segments will be stored in memory wherever the space is available in memory, how OS will track which segment is stored where in the main memory?
- This mapping is affected by the segment table.
- Each entry in the segment table has a segment base and 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.



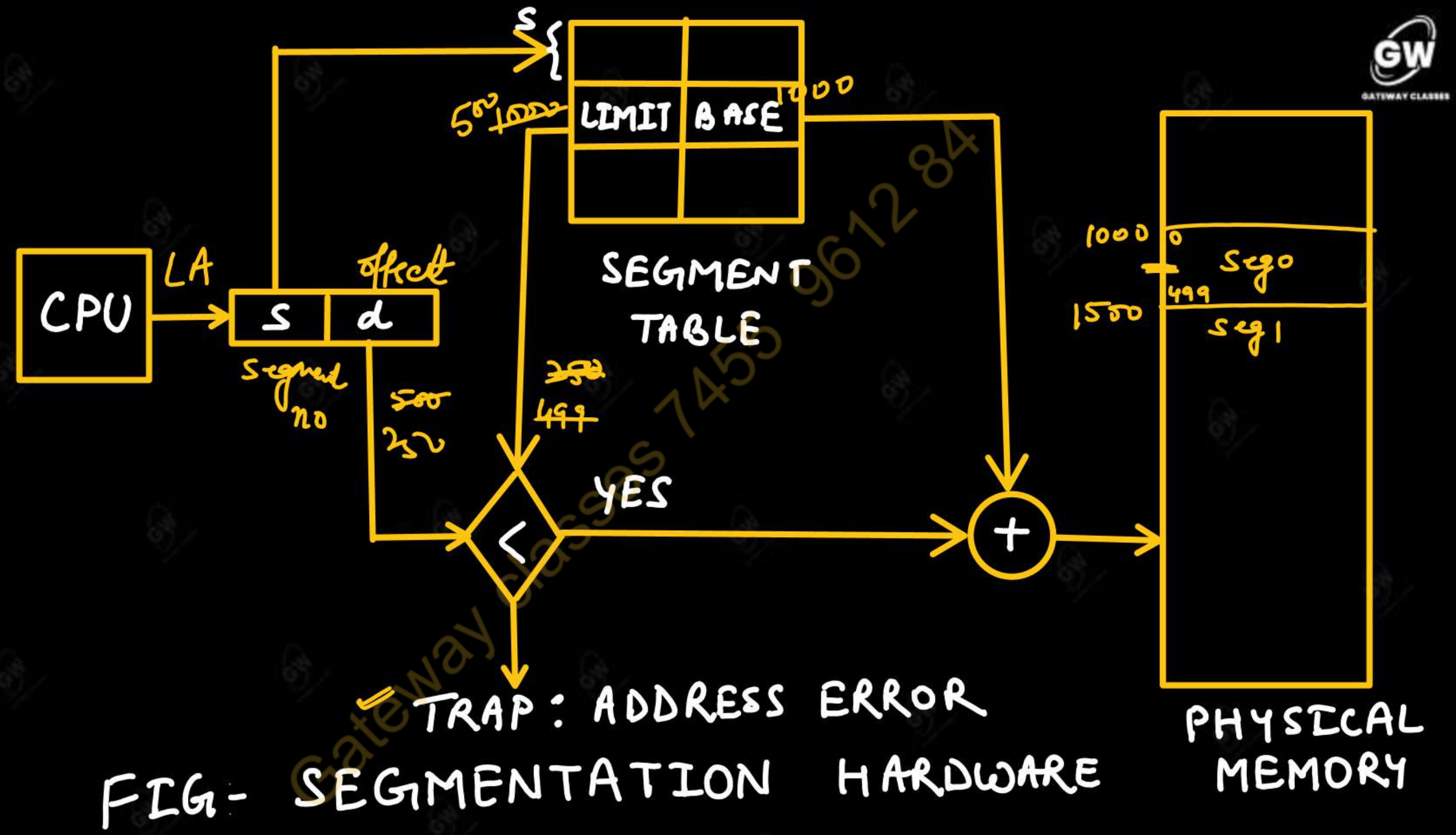
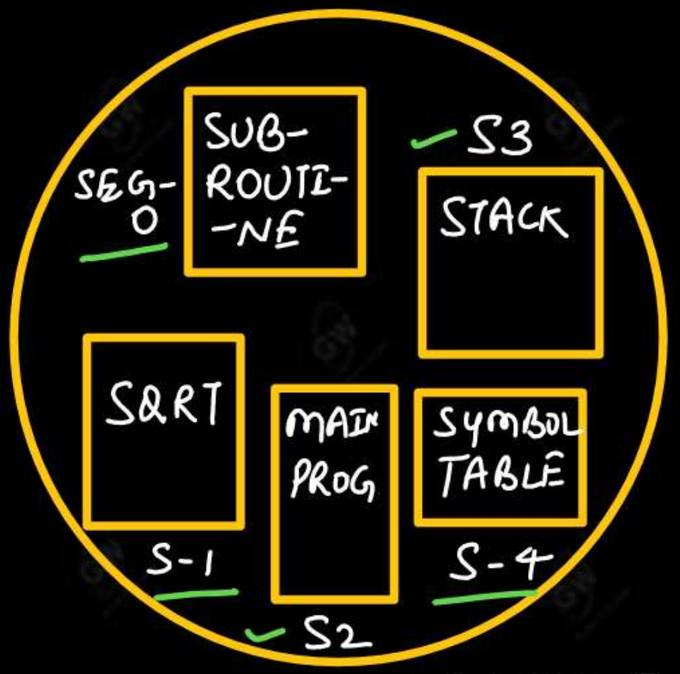


FIG- SEGMENTATION HARDWARE

- A logical address consists of two parts:

  (i) a segment number, s and

  (ii) an offset into the segment, d
- 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 (logical addressing attempt beyond end of segment).
- When an offset is legal it is added to the segment base to produce the address in the physical memory of the desired type.
- The segment table is thus essentially an array of base-limit register pairs.

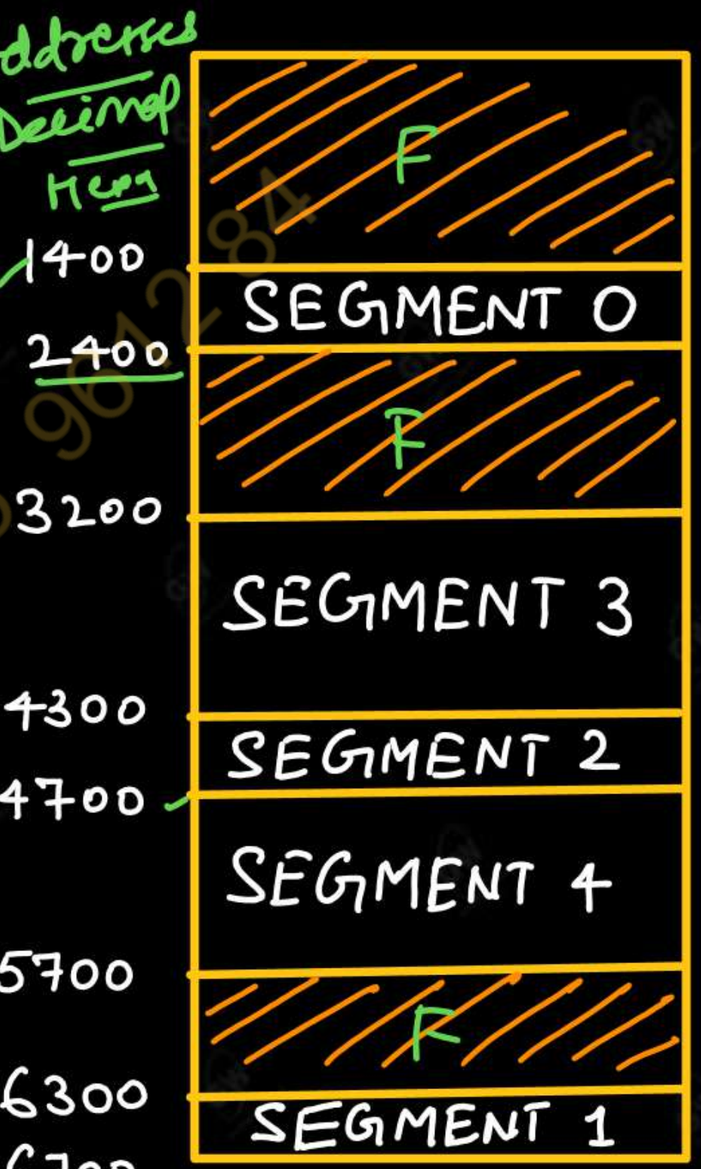


LOGICAL ADDRESS SPACE

SEGMENT TABLE

	LIMIT	BASE
S0	1000	1400
S1	400	6300
S2	400	4300
S3	1100	3200
S4	1000	4700

FIG:- SEGMENTATION PROCESS



PHYSICAL MEMORY

- In this example, we have 5 segments from 0 through 4.
- The segments are stored in physical memory as shown in the figure.
- The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit).
- For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53 = 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in the trap to OS, as this segment is only 1000 bytes long.

Advantages of Segmentation

- **Internal Fragmentation:** It occurs only when unused space arises, because the memory can be divided into fixed size parts. But a process can't use entire divided part.
- The **user's perception** of physical memory is quite **similar to segmentation**. Users can divide user programs into modules via segmentation. These modules are nothing more than separate processes' codes.
- The **user specifies the segment size**, whereas, in paging, the hardware determines the page size.
- Segmentation is a method that can be used to segregate data from **security** operations.
- **Flexibility:** Segmentation provides a higher degree of flexibility than paging. Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.
- **Protection:** Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process's memory segment.

Disadvantages of Segmentation

- **External Fragmentation:** As processes are loaded and removed from memory, the free memory space is broken into little pieces, causing external fragmentation. This is a notable difference from paging, where external fragmentation is significantly lesser.
- Due to the need for **two memory accesses**, one for the segment table and the other for main memory, access time to retrieve the instruction increases.
- **Fragmentation:** As mentioned, segmentation can lead to external fragmentation as memory becomes divided into smaller segments. This can lead to wasted memory and decreased performance.

Example 1: - Find the physical address for the following request with respect to the given segment table. Offsets - 430, 962, 331, 441

d (displacement)

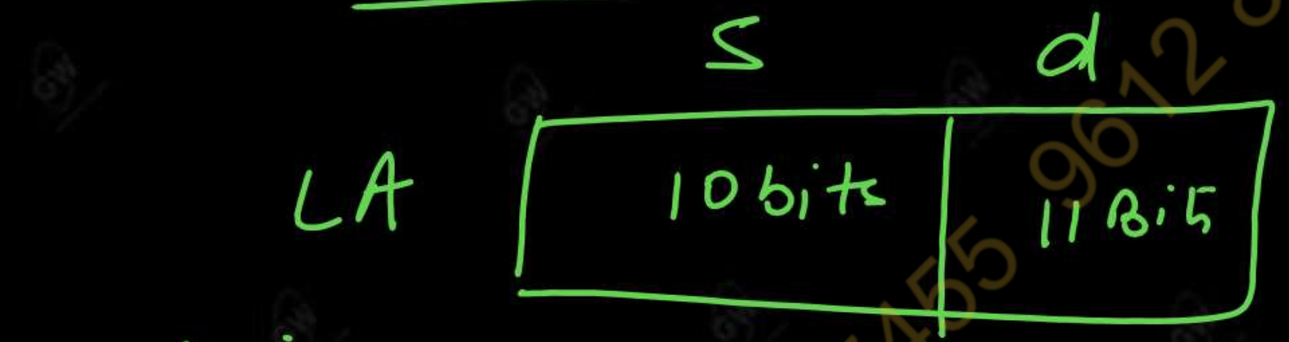
Segment No.	Base	Limit
0	6000	600
1	1500	1000
2	300	300
3	5000	450

Solution:-

s	d	Physical Address (Base + displacement)
0	430	6000 + 430 = 6430
1	962	1500 + 962 = 2462
2	321	Exceed the limit : trap , addressing error 321 > 300
3	441	5000 + 441 = 5441

Example 2: - Find the number of bits in the logical address if maximum segment size is 2KB, and number of segments in the process are 2^{10}.

Solution:-



Segment size = 2KB

$= 2^1 \times 2^{10}$

$= 2^{11}$

= 11 Bits

Segment no. $= 2^{10}$

= 10 bits

Total = 21 Bits

Ans

PAGED SEGMENTATION

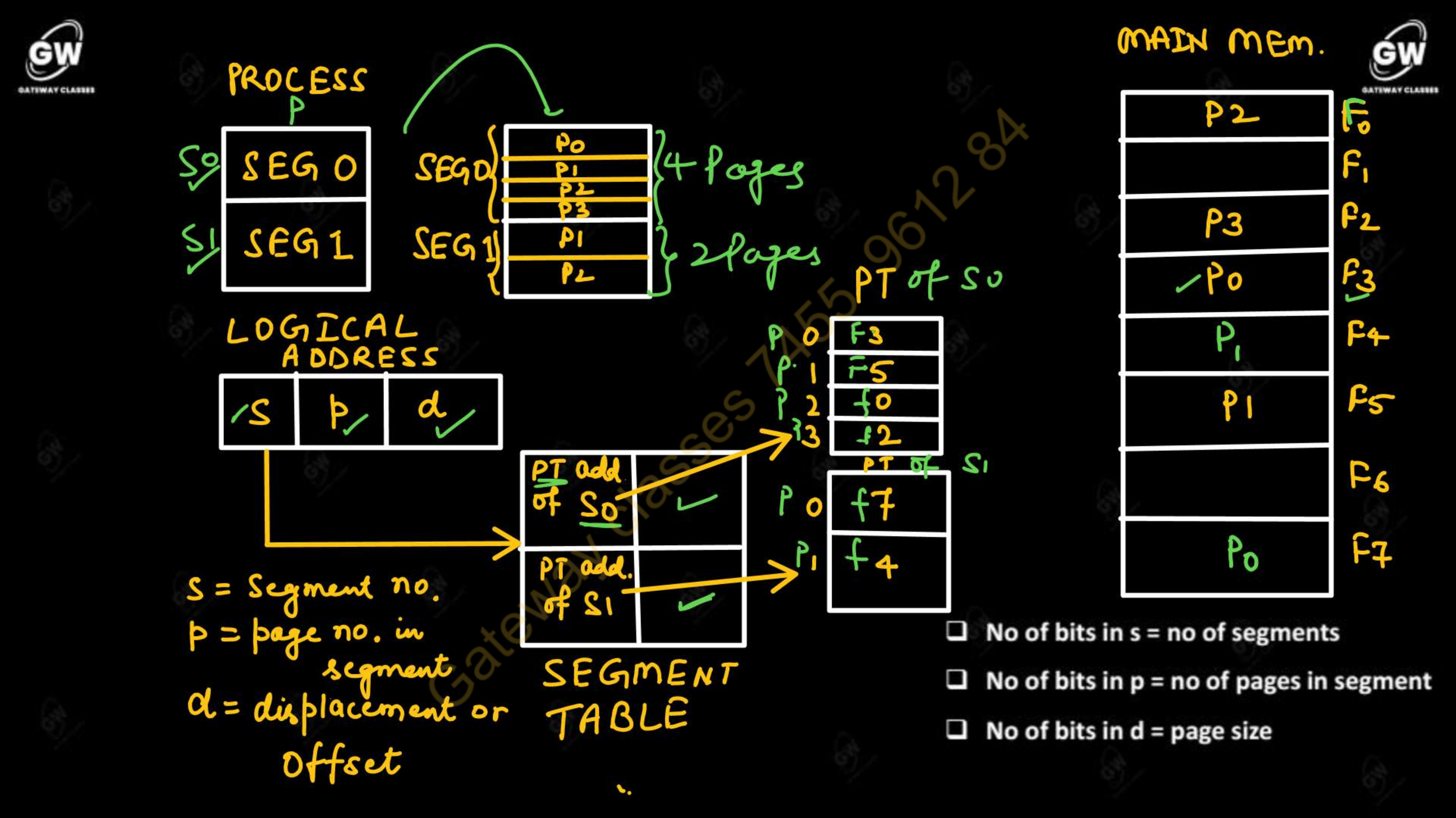
- Paging says to divide the process into fixed-sized (equal size) pages & those pages can be stored in different frames in main memory.
- Segmentation says to divide the process into different variable-size segments & these segments can be stored in different locations of physical memory.
- In segmentation, if the segment size will be very big then there may be some limitations.
- In pure paging, we will have one one-page table for each process i.e. for one big process one big page table.
- In paged segmentation, a process is divided into segments & now for each segment will maintain a particular page table.

PAGED SEGMENTATION OR SEGMENTED PAGING

- Pure segmentation is not very popular and not being used in many of the Operating Systems.
- Segmentation can be combined with paging to get the best features out of both the techniques.
- Divide each segment into multiple pages.
- Keep one page table for each segment.
- Segment table entry points to the page table of the segment.
- Memory has frames of size equal to page size and each page can be kept on any frame.

✓ PROCESS

GW GATEWAY CLASSES


PROCESS
P
S0
SEG 0
S1
SEG 1
SEG 0
P0
P1
P2
P3
4 Pages
SEG 1
P1
P2
2 Pages
LOGICAL ADDRESS
s
p
d
PT add of S0
PT add. of S1
SEGMENT TABLE
PT of S0
P 0
F3
P 1
F5
P 2
f0
P 3
f2
PT of S1
P 0
f7
P 1
f4
MAIN MEM.
P2
F0
F1
P3
F2
P0
F3
P1
F4
P1
F5
F6
P0
F7
s = Segment no.
p = page no. in segment
d = displacement or Offset
No of bits in s = no of segments
No of bits in p = no of pages in segment
No of bits in d = page size

Advantages of Segmented Paging

- The page table size is reduced as pages are present only for data of segments, hence reducing the memory requirements i.e. which segment is currently required that particular segment page table will be required in the main memory.
- Gives a programmer's view along with the advantages of paging.

Disadvantage of Segmented Paging

- Internal fragmentation still exists in pages.
- Extra hardware is required.
- Translation becomes more sequential increasing the memory access time. First of all, using the segment number go to the segment table the through page table entry go to the page table then through the page go to the main memory after getting the physical address.

AKTU PYQs

Q.1. *Explain Paging with example. Differentiate Paging and Segmentation.*

2018-19, 10 Marks

Q.2. *Consider the following segment table*

Segment	*Base*	*Length* limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What *are the physical addresses of the following logical addresses*?

(*a*) 0,430 (*b*) 1, 12 (*c*) 2,500 (*d*) 3,400 (*e*) 4,110

2017-18, 10 Marks

OPERATING SYSTEM

Today's Target -

- ❑ Virtual Memory and Demand Paging
- ❑ Thrashing
- ❑ Locality of Reference
- ❑ AKTU PYQs

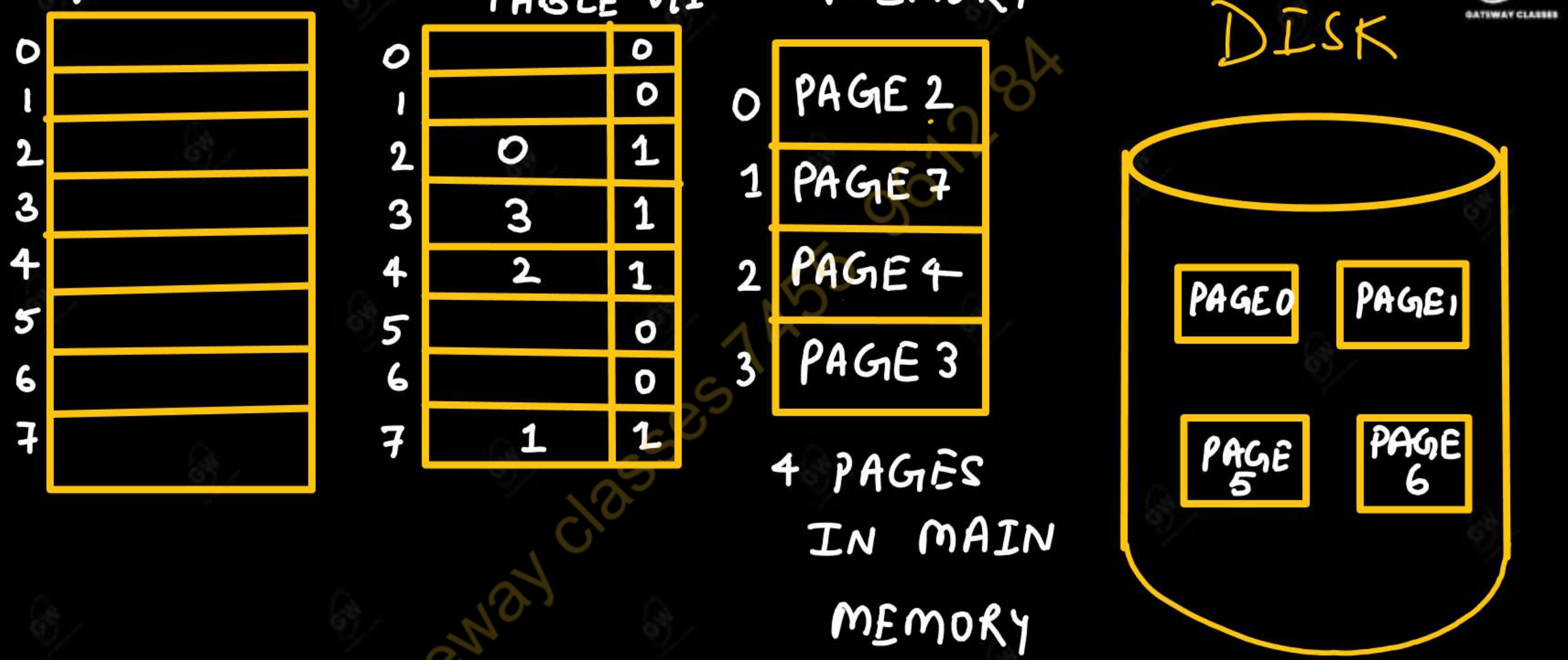
Virtual Memory

- Virtual memory is not hardware, it is a feature of the operating system.
- Virtual memory is a technique that allows the execution of processes that are not completely in memory i.e. enables to run larger process with smaller available memory.
- One major advantage of this scheme is that programs can be larger than physical memory.
- This technique frees programmers from the concerns of memory-storage limitations.
- An examination of real programs shows that, in many cases, the entire program is not needed.
- The basic idea is that the frequent or required pages of a big process will be stored in the main memory and others in secondary memory and the pages stored in secondary memory will be part of the main memory when required.
- Virtual memory involves the separation of logical memory as perceived by users from physical memory.
- This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

Primary | main

Virtual memory makes the task of programming much easier because the programmer no longer needs to worry about the amount of physical memory available; the programmer can concentrate instead on the problem to be programmed.

- If CPU demands any page of a process which is not in main memory that it is page fault i.e. when demanded page is not available in main memory. (or page miss)
- If the V/I bit is 0 in page table that means that page is not in main memory.
- Then, this page must be brought to main memory i.e. existing page of main memory will be swapped out using any page replacement algorithm and now the page is available in main memory so the page table will be updated and the V/I bit will be set with 1 to show that page is available in main memory.
- In this way the required page is swapped in main memory on demand, this is known as demand paging.
- Demand paging says to bring pages into main memory when CPU demands for that particular page.
- Operating System performs page fault service i.e. faulty pages will swap-in in main memory.



Implementing Virtual Memory Through Demand Paging

- Consider how an executable program might be loaded from disk into memory.
- One option is to load the entire program in physical memory at program execution time.
- However, a problem with this approach is that we may not initially need the entire program in memory.
- Suppose a program starts with a list of available options from which the user is to select.
- Loading the entire program into memory results in loading the executable code for all options, regardless of whether an option is selected by the user or not.
- An alternative strategy is to load pages only as they are needed.
- This technique is known as demand paging and is commonly used in virtual memory systems.

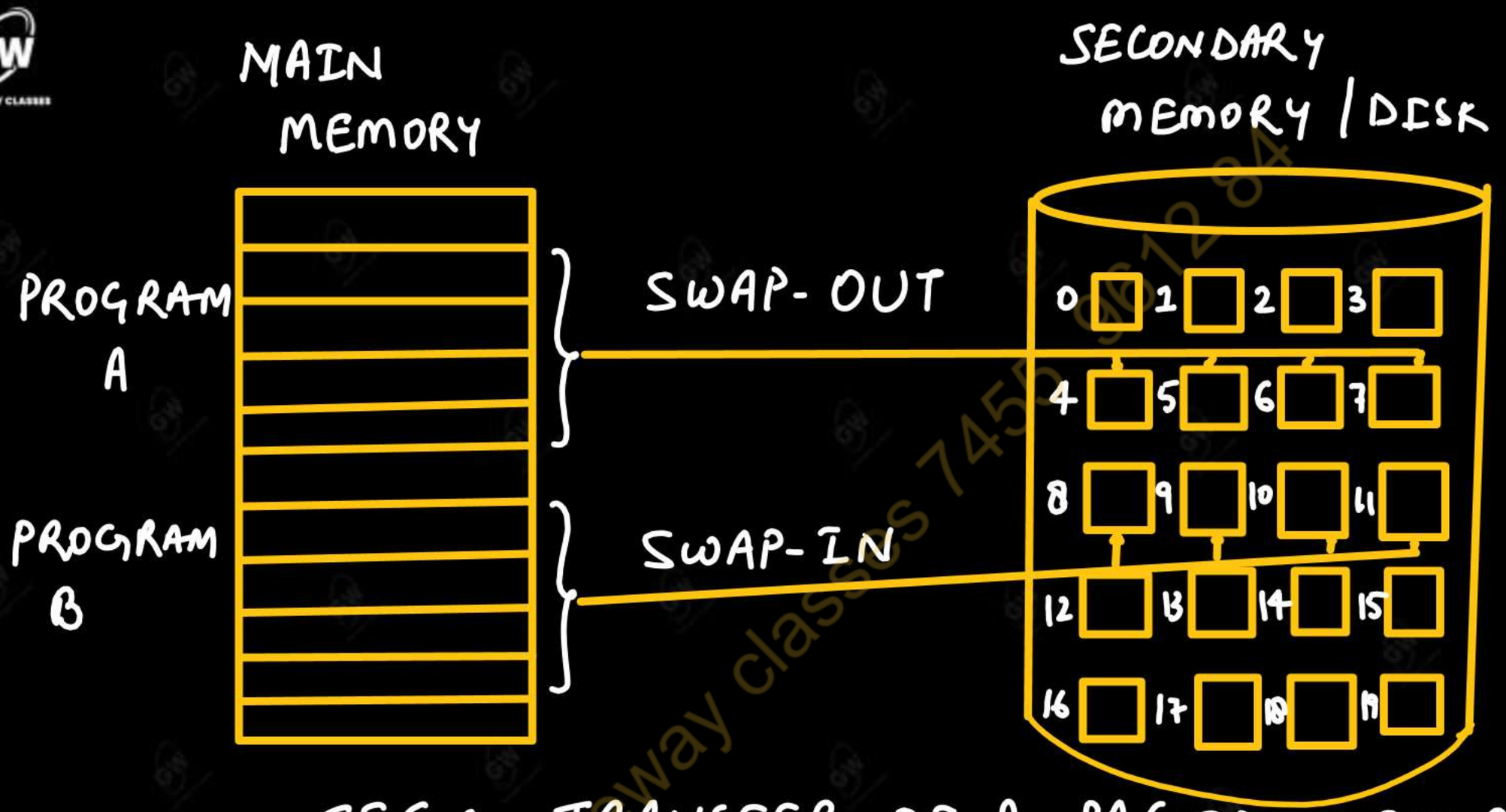


FIG:- TRANSFER OF A PAGED MEMORY TO CONTIGUOUS DISK SPACE

- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.
- A demand–paging system is similar to a paging system with swapping where the process resides in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory.
- Rather than swapping the entire process into memory, however, we use a lazy swapper.
- A lazy swapper swaps a page into memory unless that page will be needed.

BASIC CONCEPTS

- Since we are now viewing a process as a sequence of pages, rather than as one large contiguous address space, use of the term swapper is technically incorrect.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.
- We thus use pager rather than swapper in connection with demand paging.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those pages into memory.
- Thus, it avoids reading into memory pages that will not be used any way decreasing the swap time and the amount of physical memory needed.
- With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.

- The valid-invalid bit scheme can be used for this purpose.
- 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 is either 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 or contains the address of the page on disk.
- Hence, if we guess right and page in all and only those pages that are currently needed, the process will run exactly as though we had brought in all pages.

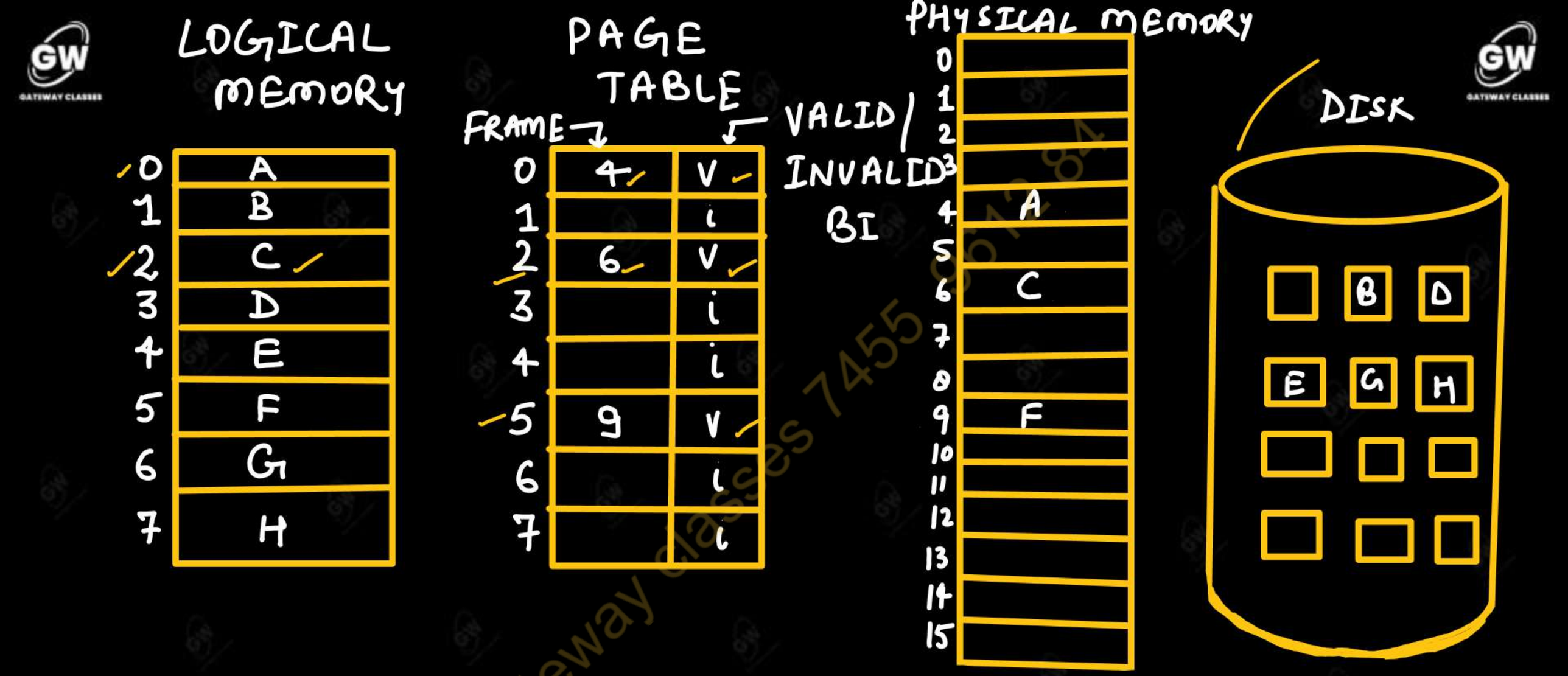


FIG:- PAGE TABLE WHEN SOME PAGES ARE NOT IN MAIN MEMORY

- While the process executes and accesses pages that are memory resident, execution proceeds normally.
- But what happens if the process tries to access a page that was not brought into memory? Access to a page marked invalid causes a page fault.
- The paging hardware in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.
- This trap is the result of the operating system's failure to bring the desired page into memory.
- The procedure for handling this page fault is straightforward-

1. We check an internal table (usually kept with the PCB) for this purpose to determine whether the reference was a valid or and invalid memory access.
2. If the reference is invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one from the free–frame list, for example)

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. 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.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

- In the extreme case, 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 brought into memory, the process continuous to execute, faulting as necessary until every page that it needs is in memory.
- At that point, it can execute with no more faults. This scheme is **Pure Demand Paging** never being a page into memory until it is required.

- The hardware to support demand paging is the same as the hardware for paging and segmentation:

➢ **PAGE TABLE:**

- This table has the ability to mark an entry invalid through a valid–invalid bit or a special value of protection bits.

➢ **SECONDARY MEMORY:**

- This memory holds these pages that are not present in the main memory.
- The secondary memory is known as a swap device, and the section of disk used for this purpose is known as swap space.
- A crucial requirement for demand paging is the ability to restart and instruction after a page fault.
- Because we save the state of the interrupted process when the page fault occurs, we must be able to restart the process in the same place and state, except that the desired page is now in memory and is accessible.

Time Required for Page Fault Service

i.e. Page fault service time

- Assume Page Fault rate = P
- EMAT = (1 – P) × time without page fault + p × time with page fault service

When page fault is not there

If page fault occur

i.e.

Effective Memory Access Time (EMAT) = (1 – P) × (2 × t_{mm}) + p * page fault service time.

Page fault service time is very high,

in that case 1 MAT to see the page table can be ignored

Example 1: Consider a process executing on 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 experimentally 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)

(a) (D – M) / (X – M)

(b) (X – M) / (D – M)

(c) (D – X) / (D – M)

(d) (X – M) / (D – X)

Mem = M
Page fault = D
EMAT = X Units

Solution:-

Time without page fault (i.e. page is in memory) = M units

If page fault occurs i.e. page is not in memory = D units

Average memory access time = X units

Page Fault Fate = ?

Assume page fault rate = p

$$X = (1 - p) * (2 * t_{mm}) + p * \text{page fault service time}$$

$$X = (1 - p) * M + p * D$$

$$X = M - M p + D p$$

$$X - M = p (D - M)$$

$$p = X - M / D - M$$

i.e. option b is the correct answer.

Thrashing

- Thrashing in operating systems occurs when the system spends a significant amount of time swapping data between main memory (RAM) and disk, rather than executing useful tasks.
- This happens when the system is overloaded with too many processes competing for limited resources, leading to excessive paging or swapping activity.
- Thrashing severely degrades system performance, causing a slowdown in overall operations.
- Consider any process that does not have enough frames.
- If the process does not have the number of frames it needs to support pages in active use, it will quickly page fault. At this point it must replace some pages.
- However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, replacing pages that it must bring back in immediately.
- This high-level paging activity is called thrashing.

CAUSE OF THRASHING

- Consider the following scenario, which is based on the actual behavior of early paging systems.
- The Operating System monitors the CPU utilization.
- If CPU unitization 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 suppose that a process enters a new phase in its execution and needs more frames.
- It starts faulting and taking frames away from processes that need those pages, however, and so they are fault, taking frames from other processes.
- These faulting processes most use the paging device to swap pages in and out.
- 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 paging devices.
- As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more.
- As a result, the effective memory-access times increase.
- No work is getting done, because the processes are spending all their time paging.
- Consider the next figure, in which CPU utilization is plotted against the degree of multiprogramming.

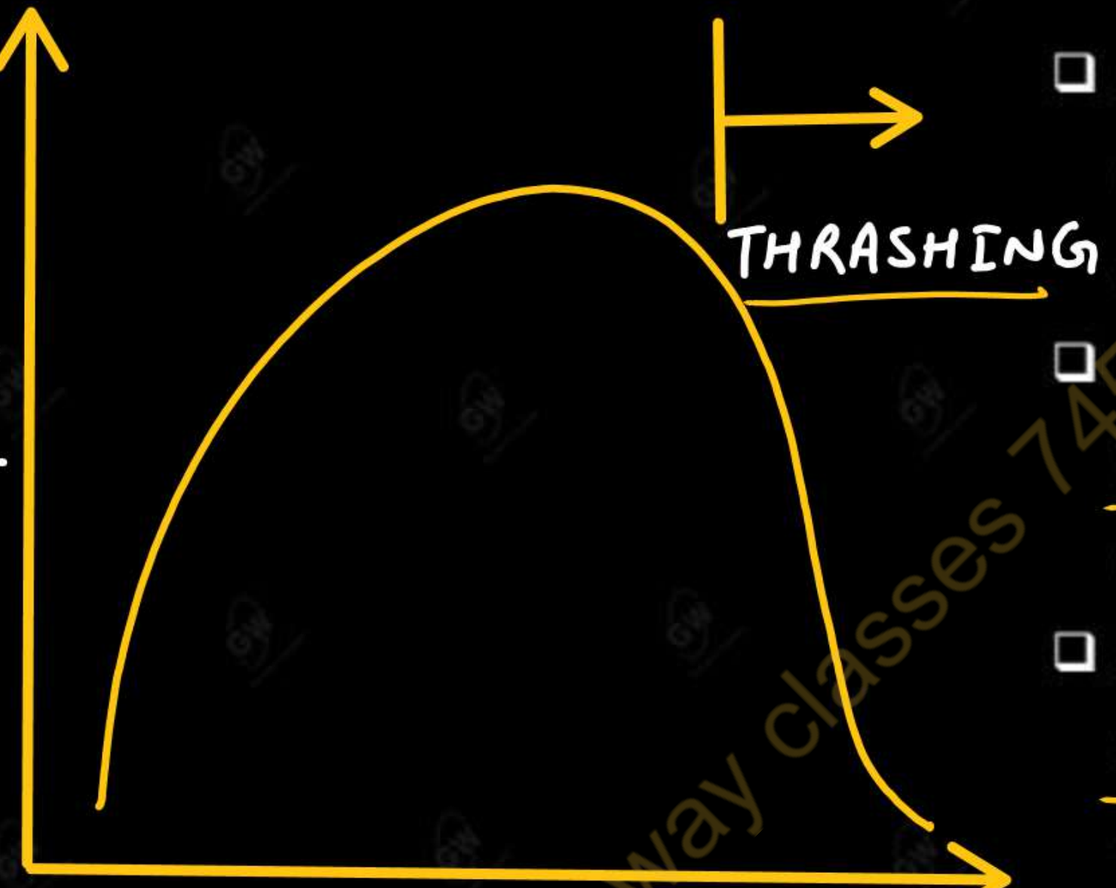


FIG:- THRASHING

- As the degree of multiprogramming increases, CPU utilization also increases, although more slowly until a maximum is reached.
- If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply.
- At this point, to increase CPU utilization and stop thrashing we must decrease the degree of multiprogramming.

Strategies to mitigate or solve Thrashing

1. **Increase Physical Memory (RAM):** Adding more physical memory can reduce the frequency of page faults, thereby alleviating thrashing.
2. **Adjusting the Degree of Multiprogramming:** Reducing the number of processes in the system can decrease the total demand for memory.
3. **Page Replacement Algorithms:** Implementing more efficient page replacement algorithms can help.
4. **Priority Paging:** Assigning priorities to pages so that more critical pages are less likely to be replaced. This helps ensure that essential pages remain in memory.

- ❑ By combining these strategies, an operating system can effectively manage memory and reduce the impact of thrashing, thus maintaining system performance and stability.

Locality of Reference

- It refers to the tendency of programs to access a relatively small portion of their address space at any given time.
- If the CPU has requested one address for memory access, then that particular address or nearby addresses will be allured soon.
- To access the same memory block again and again (by the processor).
- Some blocks of main memory are kept in cache memory. (data from main memory to cache memory).
- Locality of reference is implemented by two ways -

1. Spatial locality (based on space) - This principle suggests that if a program accesses a certain memory location, it is likely to access nearby memory locations soon. This is common in programs that access array elements or data structures sequentially. If CPU accesses the nearby addresses of previously accessed references. If a word is accessed now then the word adjacent to it (close proximity) will be accessed next.

2. Temporal locality (based on time) - This principle states that if a program accesses a certain memory location, it is likely to access the same location again in the near future. For example, loops and frequently called functions exhibit temporal locality because the same instructions or data are accessed repeatedly within a short period. If the CPU accesses the same address again after some time.

- A memory block contains a number of words.
- Therefore, the complete word of the main memory will be placed in cache memory, assuming that if any word to that particular block is accessed now, the next adjacent word of the same block will be accessed next time. In this way, it will increase the number of hits. (spatial locality).
- Keeping more words in a block affects spatial locality (block size) i.e. more words in block results in more hit ratio.
- If a word is referenced now then the same word will referenced will referenced again in near future. (Temporal locality).
- For example if any block is accused and placed in cache memory (from main memory) and if and word is accused from that block, temporal locality says that the same word may be accused soon again. (Time)
- LRU replacement algorithm is used in the temporal locality which works on the same principle.

AKTU PYQs

Q.1. *Explain the concept of demand paging. Consider the given references to the following pages by a program:* 0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3

How many pages faults will occur if the program has three – page frames available to it and uses:

(i) FIFO replacement (ii) LRU replacement (iii) Optimal replacement 2022-23, 10 Marks

Q.2. *Describe Swap – in and Swap – out in Demand paging.* 2017-18, 2 Marks

Q.3. *What is Demand paging?* 2016-17, 2 Marks

Q.4. *Explain Concept of Virtual Memory.* 2016-17, 2 Marks

Q.5. *Explain thrashing and locality of reference.* 2022-23, 2 Marks

Q.6. *What is Thrashing? What is the cause of Thrashing? How does the system detect Thrashing? What can the system do to eliminate this problem?*

2017-18, 10 Marks, 2016-17, 10 Marks,2015-16, 10 Marks

Q.7. *Explain Thrashing and its solution.* 2017-18, 10 Marks

Download Gateway Classes Application
From Google Play store
Link in Description

Next Lecture:- Unit – 5, I/O Management and Disk Management

Thank You

AKTU

B. Tech II-Year

Gateway Classes

Full Courses Available in App

AKTU B.Tech I- Year : All Branches

AKTU B.Tech II- Year

Branches : 1. CS IT & Allied

2. EC & Allied

3. ME & Allied

4. EE & Allied

Downlod App Now

Download App

Full Courses

- V. Lectures
- Pdf Notes
- AKTU PYQs
- DPP