Unit 3

MEMORY MANAGEMENT

3.1. Introduction To Storage Organization:

- The term memory can be defined as a collection of data in a specific format.
- It is used to store instructions and process data.
- The memory comprises a large array or group of words or bytes, each with its own location.
- The **main memory** is central to the operation of a Modern Computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.

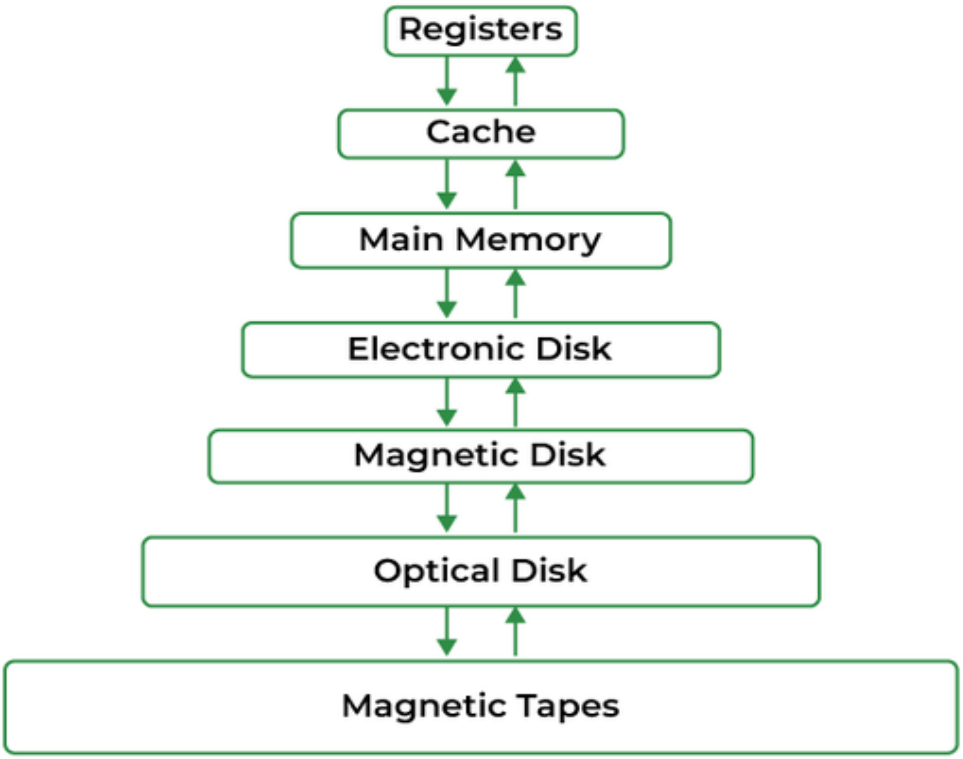


Fig: Main Memory

Memory Management:

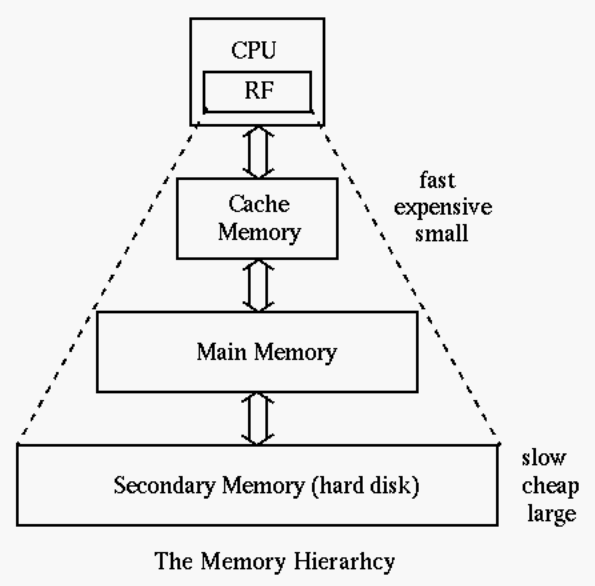
- The task of subdividing the memory among different processes is called Memory Management.
- Memory management is a method in the operating system to manage operations between main memory and disk during process execution.

Need of Memory Management:

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize Fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

3.2. Memory Hierarchy:

- Memory hierarchy is a ranking of computer memory devices, with devices having the fastest access time at the top of the hierarchy, and devices with slower access times but larger capacity and lower cost at lower levels.



The Memory Hierarhcy

The memories in the hierarchy are as follows:

• At the top level of the memory hierarchy are the CPU's **general purpose registers.** The registers provide the fastest access to data. The register file is also the smallest memory object in the memory hierarchy.

• The **Level One Cache (L1)** system is the next highest performance subsystem in the memory hierarchy. The size is usually quite small (typically between 4Kbytes and 32Kbytes), though much larger than the registers available on the CPU chip.

• The **Level Two Cache (L2)** is present on some CPUs. The Level Two Cache is generally much larger than the level one cache (e.g., 256 or 512KBytes versus 16 Kilobytes).

• Below the Level Two Cache system **main memory** subsystem is available. This is the general-purpose, relatively low-cost memory found in most computer systems. Main memory or internal memory is the only one directly accessible to the CPU.

• **Secondary memory** also known as external memory or auxiliary storage is different from primary memory in that it is not directly accessible by the CPU. The data stored in such memories are permanently stored and are not lost even the power cut off i.e. non-volatile in nature.

3.3.Logical and Physical Address Space:

1. **Logical Address Space:** An address generated by the CPU is known as a "Logical Address". It is also known as a virtual Address. Logical address space can be defined as the size of the process. A logical address can be changed.
2. **Physical Address Space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address". It is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. It is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

3.4.Static and Dynamic Loading:

Loading a process into the main memory is done by a loader.

There are two different types of loading :

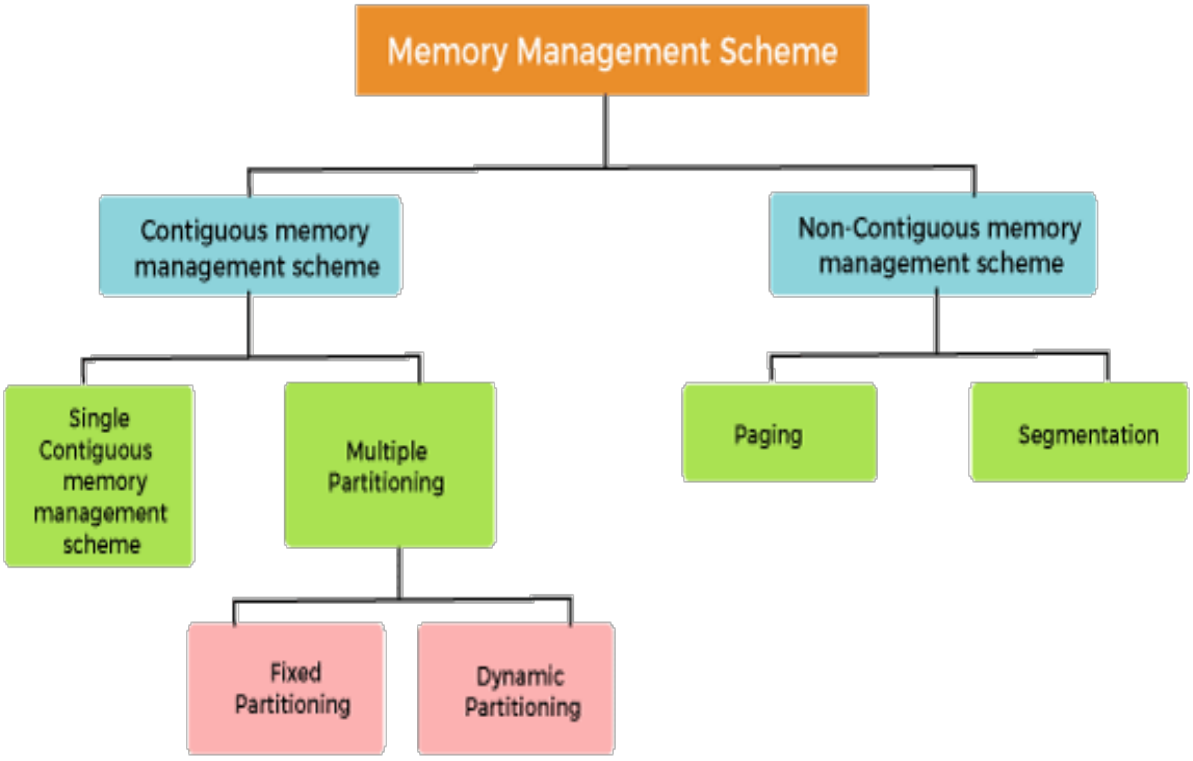
- **Static Loading:** Static Loading is basically loading the entire program into a fixed address. It requires more memory space.
- **Dynamic Loading:** The entire program and all data of a process must be in physical memory for the process to execute. In dynamic loading, a routine is not loaded until it is called.

3.5.Static and Dynamic Linking:

- To perform a linking task a linker is used.
- A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

1. **Static Linking:** In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
2. **Dynamic Linking:** The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, "Stub" is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available, then the program loads the routine into memory.

3.6. Memory Management Techniques:



Classification of memory management schemes

1. **Contiguous memory allocation**:

- Contiguous Memory Allocation is a type of memory allocation technique where processes are allotted a continuous block of space in memory.
- This block can be of fixed size for all the processes in a fixed-size partition scheme or can be of variable size depending on the requirements of the process in a variable-size partition scheme.

Contiguous Memory Management has two types:

1.1. **Single contiguous memory management**

- It is one of the simplest memory management techniques used in operating systems.
- In this technique, the entire program is loaded into one contiguous block of memory during execution.
- It assumes that there is only one program running at a time in memory, apart from the operating system, making it easy to implement but also highly limited in flexibility.

Uses:

- Early operating systems like MS-DOS used single contiguous memory management.
- Embedded systems or simple systems where multitasking is not required may still use this technique.

1.2. **Multiple Partitioning Memory Management Technique:**

- Multiple partitioning memory management techniques allow more efficient use of memory by dividing it into multiple sections or partitions to accommodate several processes simultaneously.
- These techniques are improvements over single contiguous allocation and are widely used in modern operating systems.

Types of multiple partitioning memory management techniques:

A. Fixed Partitioning (Static Partitioning)

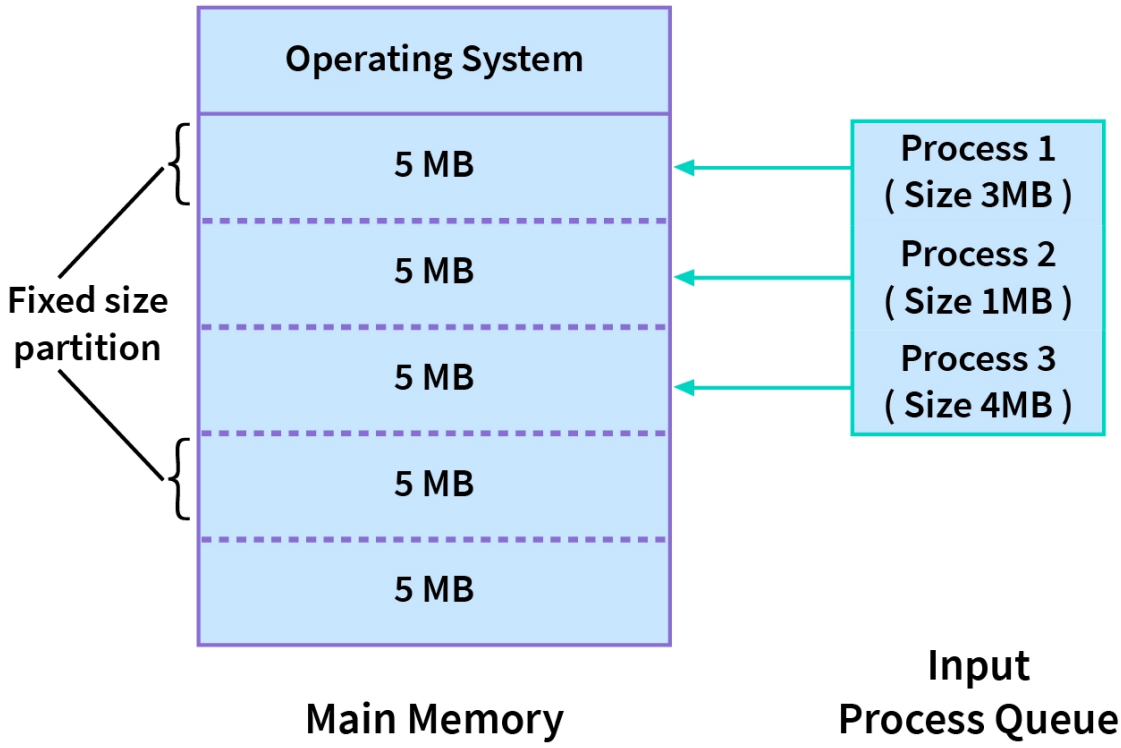
- Memory is divided into fixed-sized partitions at system startup, and each partition can hold one process.
- Partitions are created with fixed sizes (could be equal or unequal).
- Each process is loaded into a separate partition.

Advantages

- Simple to implement.
- Fast allocation since partitions are predefined.

Disadvantages

- **Internal Fragmentation:** Unused memory within a partition if a process is smaller than the partition size.
- **Wasted Memory:** Inefficient if process sizes do not match partition sizes well.
- Number of processes is limited by the number of partitions.



B. <u>Variable Partitioning (Dynamic Partitioning)</u>

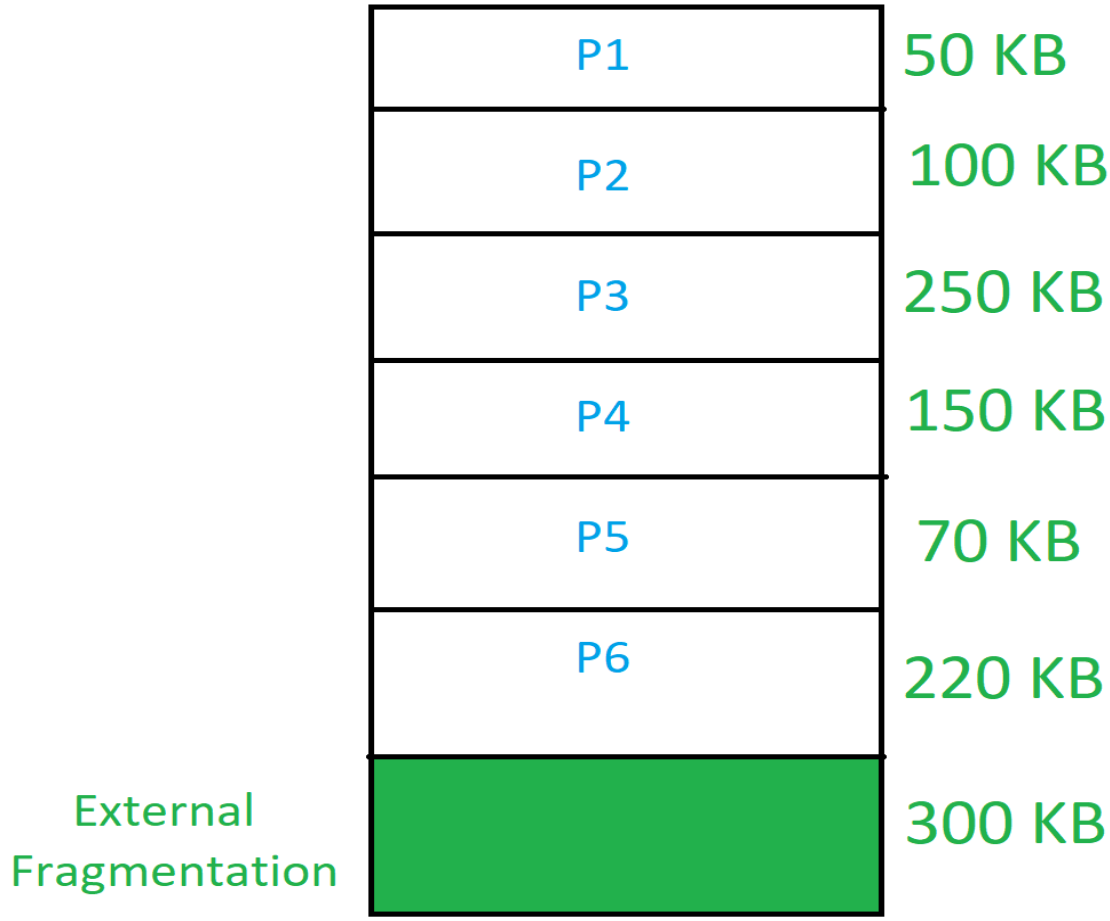
- Memory is allocated dynamically according to the exact size required by a process, without predefined partitions.
- The size of partitions varies based on the process.
- Allocates the exact amount of space needed for each process.

Advantages

- **No Internal Fragmentation:** Memory is allocated precisely as required.
- More flexible than fixed partitioning.

Disadvantages

- **External Fragmentation:** Free memory becomes scattered into small holes, making it difficult to allocate to new processes.
- Requires compaction to rearrange memory and consolidate free space.



Advantages of Contiguous Memory Allocation

- **Faster Execution:** Less time is consumed in the execution of processes due to the fact that memory is provided in a sequential manner.
- **Minimal Overhead:** There are fewer address translations, which all contribute to the efficiency of the system.
- **Easier to Control:** Organization of the memory in the operating system is sequential, which makes it possible to have efficient control of the processes.

Disadvantages of Contiguous Memory Allocation

- **Internal and External Fragmentation:** It was noted that one way that memory blocks can be implemented is that they may become fragmented, which results in inefficiency.
- **Limited Multiprogramming:** The disadvantage of fixed partitioning is that it hinders the ability to control and allocate several processes.
- **Wastage of Memory:** With fixed partitions, memory can be wasted, especially where its usage is not optimal as required.

3.7 Fragmentation:

- Fragmentation is an unwanted issue that occurs in an operating system in which a process is unloaded and loaded from memory, causing the free memory space to become fragmented.
- As a result, the process can not be assigned to the memory blocks due to their small size.

Cause of Fragmentation

- User processes are loaded and unloaded from the main memory, and processes are kept in memory blocks in the main memory.
- Many spaces remain after process loading and swapping that another process cannot load due to their size.
- Main memory is available, but its space is insufficient to load another process because of the dynamical allocation of main memory processes.

Types of Fragmentation

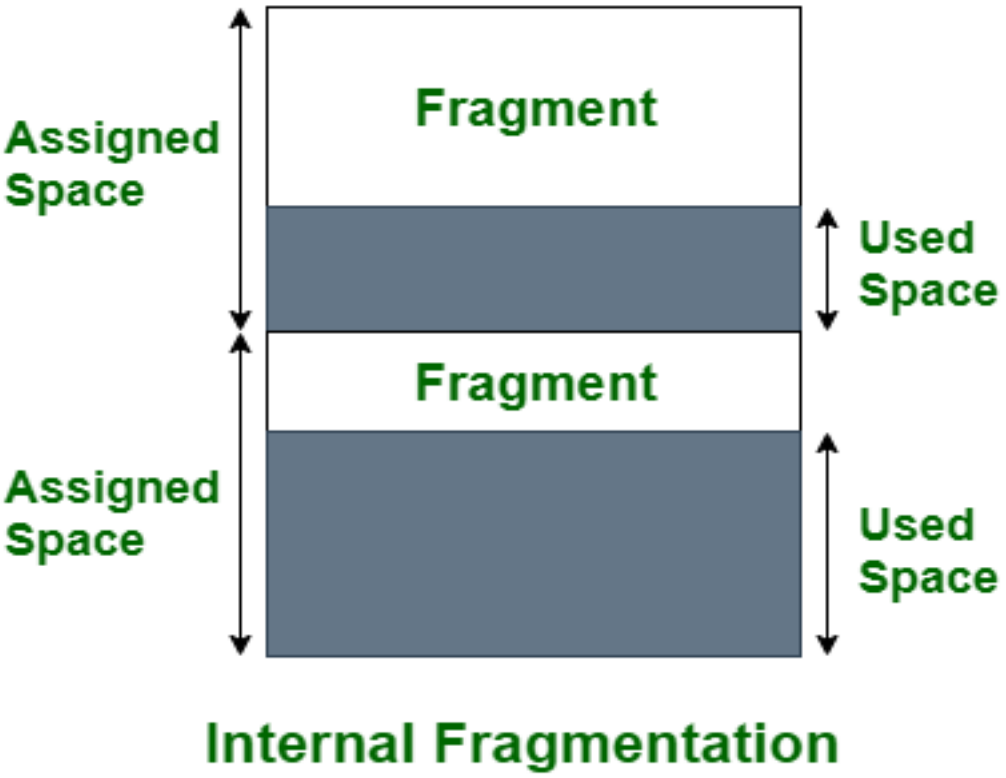
There are two main types of fragmentation:

- Internal Fragmentation
- External Fragmentation

1. Internal Fragmentation

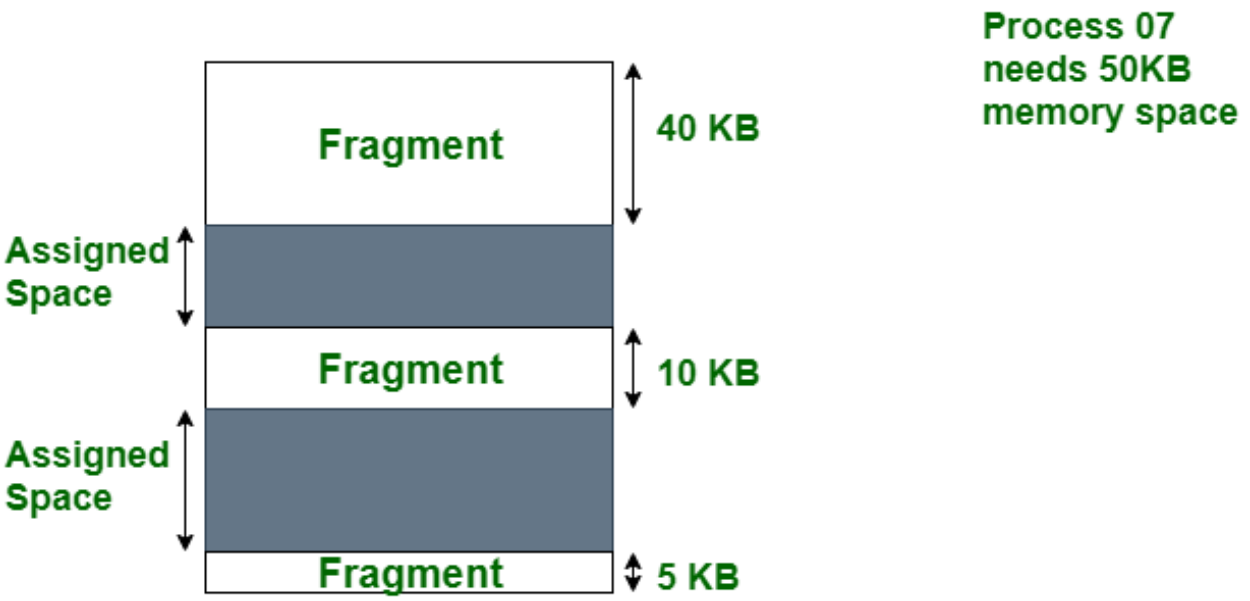
- Internal fragmentation occurs when there is unused space within a memory block.
- For example, if a system allocates a 64KB block of memory to store a file that is only 40KB in size, that block will contain 24KB of internal fragmentation.

- When the system employs a fixed-size block allocation method, such as a memory allocator with a fixed block size, this can occur.



2. <u>External Fragmentation</u>

- External fragmentation occurs when a storage medium, such as a hard disc or solid-state drive, has many small blocks of free space scattered throughout it.
- This can happen when a system creates and deletes files frequently, leaving many small blocks of free space on the medium.
- When a system needs to store a new file, it may be unable to find a single contiguous block of free space large enough to store the file and must instead store the file in multiple smaller blocks. This can cause external fragmentation and performance problems when accessing the file.



Effects of Fragmentation:

- **Slower Performance:** Fragmentation slows down the read and write speed of the disk, as the disk head has to move to different locations to access the fragments of a file. This, in turn, increases the access time and reduces the overall speed of the system, causing slowdowns and lag in applications.
- **Disk Space Wasting:** Fragmentation can also lead to disk space being wasted, as fragments may occupy more space than required. This can result in a shortage of disk space, causing the system to become unstable and vulnerable to crashes or errors.
- **Data Loss:** In severe cases, fragmentation can also cause the system to run out of disk space, leading to data loss. This can be particularly detrimental to users who rely on their systems to store critical information and data.
- **Increased Risk of System Crashes:** The more fragmented a disk becomes, the more prone it is to crashes and errors. This can cause the system to become unstable and vulnerable to data loss, crashes, and other issues.
- **Reduced Battery Life:** Fragmentation can also negatively impact the battery life of laptops and other mobile devices. This is because fragmentation requires the disk to work harder, which in turn puts additional strain on the battery.

Advantages of Fragmentation:

- **Efficient Memory Utilization:** Fragmentation allows for the efficient use of memory by allocating smaller memory blocks that match the exact size of data, reducing wasted space
- **Improved Allocation Speed:** Allocating smaller memory fragments can be faster than allocating large, contiguous blocks, especially in scenarios with varying memory demands
- **Flexibility:** Fragmentation allows for more flexible memory allocation, accommodating data structures with different sizes and lifetimes

Disadvantages of Fragmentation

- **Wasted Space:** Can result in unused memory within allocated blocks which is internal fragmentation
- **Complex Management:** Requires advanced algorithms for effective memory management
- **Overhead:** Adds overhead in terms of time and memory used for management
- **Risk of Memory Exhaustion:** Poor management can lead to "out of memory" errors despite available free memory

3.8 Memory Allocation Algorithm:

A. First Fit

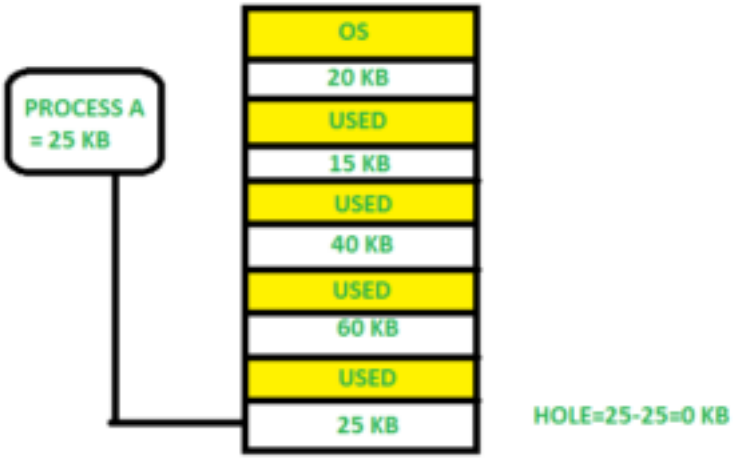
B. Best Fit

C. Worst Fit

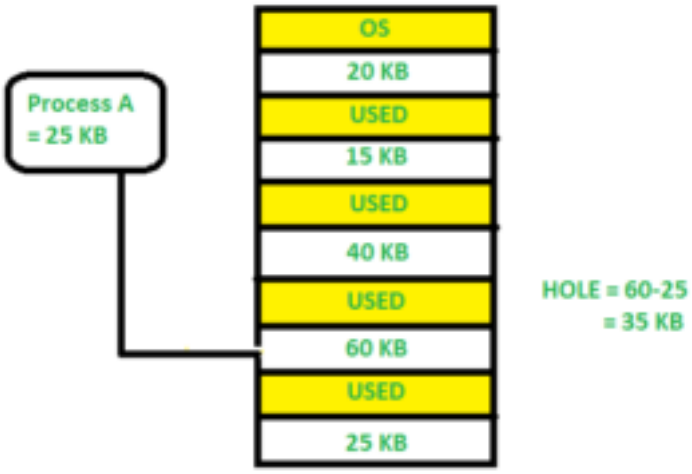
D. Next Fit

1. **First Fit:** In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.

2. **Best Fit :**Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



3. **Worst Fit**: Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



3. **Next Fit:** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

S.N.	Partition Allocation Method	Advantages	Disadvantages
1.	**Fixed Partition**	Simple, easy to use, no complex algorithms needed	Memory waste, inefficient use of memory resources
2.	**Dynamic Partition**	Flexible, more efficient, partitions allocated as required	Requires complex algorithms for memory allocation
3.	**Best-fit Allocation**	Minimizes memory waste, allocates smallest suitable partition	More computational overhead to find smallest split
4.	**Worst-fit Allocation**	Ensures larger processes have sufficient memory	May result in substantial memory waste
5.	**First-fit Allocation**	Quick, efficient, less computational work	Risk of memory fragmentation

3.9 Non-Contiguous Memory Management Technique:

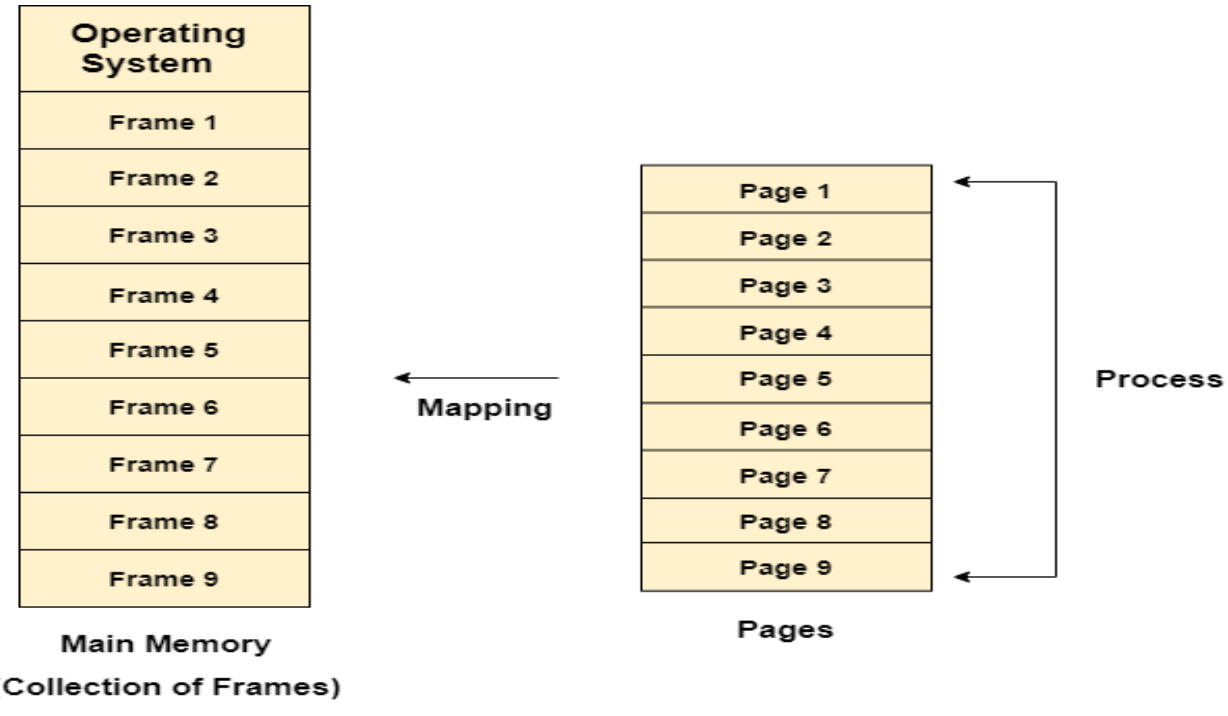
- Non-contiguous allocation, also known as dynamic or linked allocation, is a memory allocation technique used in operating systems to allocate memory to processes that do not require a contiguous block of memory.
- In this technique, each process is allocated a series of non-contiguous blocks of memory that can be located anywhere in the physical memory.

There are Two types:

- Paging
- Segmentation

1. Paging:

- Paging is a technique that divides memory into fixed-sized blocks.
- The main memory is divided into blocks known as Frames and the logical memory is divided into blocks known as Pages.
- This concept of Paging in OS includes dividing each process in the form of pages of equal size and also, the main memory is divided in the form of frames of fixed size.
- Now, each page of the process when retrieved into the main memory, is stored in one frame of the memory, and hence, it is also important to have the pages and frames of equal size for mapping and maximum utilization of the memory.



Working of Paging:

- Paging is a method used by operating systems to manage memory efficiently.
- It breaks physical memory into fixed-size blocks called "frames" and logical memory into blocks of the same size called "pages."
- When a program runs, its pages are loaded into any available frames in the physical memory.
- This approach prevents fragmentation issues by keeping memory allocation uniform.
- Each program has a page table, which the operating system uses to keep track of where each page is stored in physical memory.
- When a program accesses data, the system uses this table to convert the program's address into a physical memory address.
- Paging allows for better memory use and makes it easier to manage. It also supports virtual memory, letting parts of programs be stored on disk and loaded into memory only when needed.
- This way, even large programs can run without fitting entirely into main memory.
- If Logical Address = 31 bit, then Logical Address Space = 231 words = 2 G words (1 G = 230)
- If Logical Address Space = 128 M words = 27 * 220 words, then Logical Address = log2 227 = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 222 words = 4 M words (1 M = 220)
- If Physical Address Space = 16 M words = 24 * 220 words, then Physical Address = log2 224 = 24 bits
- The mapping from virtual to physical address is done by the MMU which is a hardware device and this mapping is known as the paging technique.
- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called frames.
- The Logical Address Space is also split into fixed-size blocks, called pages.
- Page Size = Frame Size

Example

Physical Address = 12 bits, then Physical Address Space = 4 K words

Logical Address = 13 bits, then Logical Address Space = 8 K words

Page size = frame size = 1 K words (assumption)

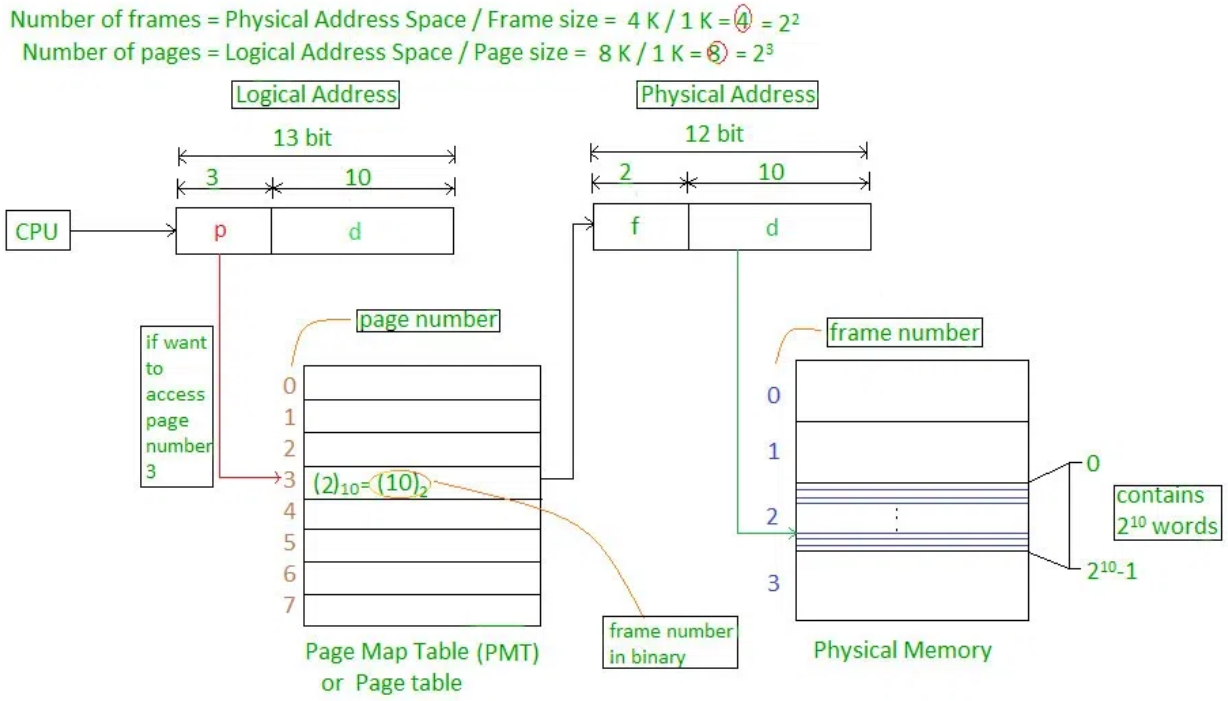


Fig: Working of Paging

The address generated by the CPU is divided into

Page number(p): Number of bits required to represent the pages in Logical Address Space or Page number

Page offset(d): Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

In a paging scheme, the physical cope with the area is divided into fixed-length frames, each of which contains some bytes or words. When a manner is running, its logical address space is split into constant-size pages, which might be mapped to corresponding frames within the physical address space.

Advantages of Paging

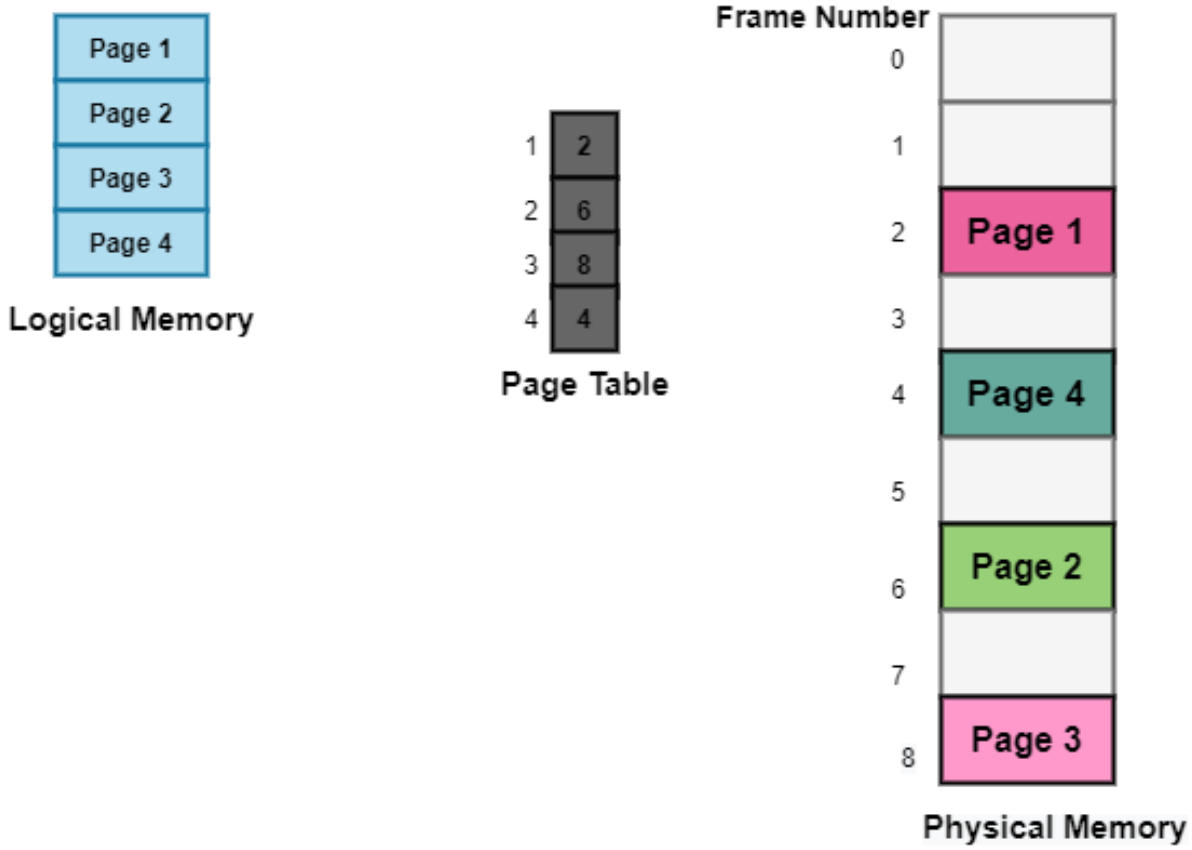
- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.

Disadvantages of Paging

- May cause Internal fragmentation
- Page tables consume additional memory.
- Multi-level paging may lead to memory reference overhead.

Page Table Structure:

- The data structure that is used by the virtual memory system in the operating system of a computer to store the mapping between physical and logical addresses is commonly known as Page Table.
- Thus, page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.



The above diagram shows the paging model of Physical and logical memory.

Characteristics of the Page Table

- It is stored in the main memory.
- Generally; the Number of entries in the page table = the Number of Pages in which the process is divided.
- PTBR means page table base register and it is basically used to hold the base address for the page table of the current process.
- Each process has its own independent page table.

Techniques used for Structuring the Page Table

Some of the common techniques that are used for structuring the Page table are as follows:

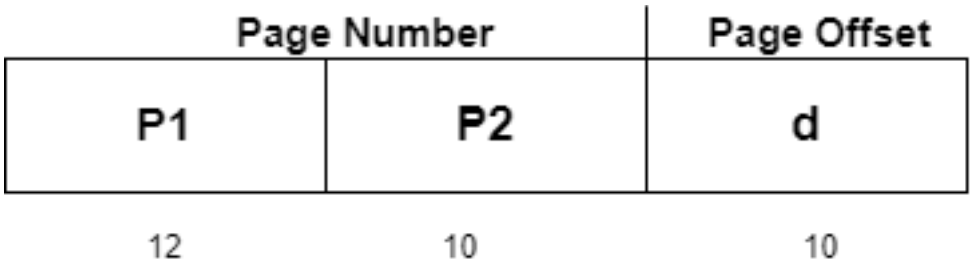
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

1. Hierarchical Paging

- Another name for Hierarchical Paging is multilevel paging.
- There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- In this type of Paging the logical address space is broke up into Multiple page tables.
- Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.

Two Level Page Table

- Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:
- Page Number consisting of 22 bits.
- Page Offset consisting of 10 bits.
- As we page the Page table, the page number is further divided into :
- Page Number consisting of 12 bits.
- Page Offset consisting of 10 bits.
- Thus the Logical address is as follows:

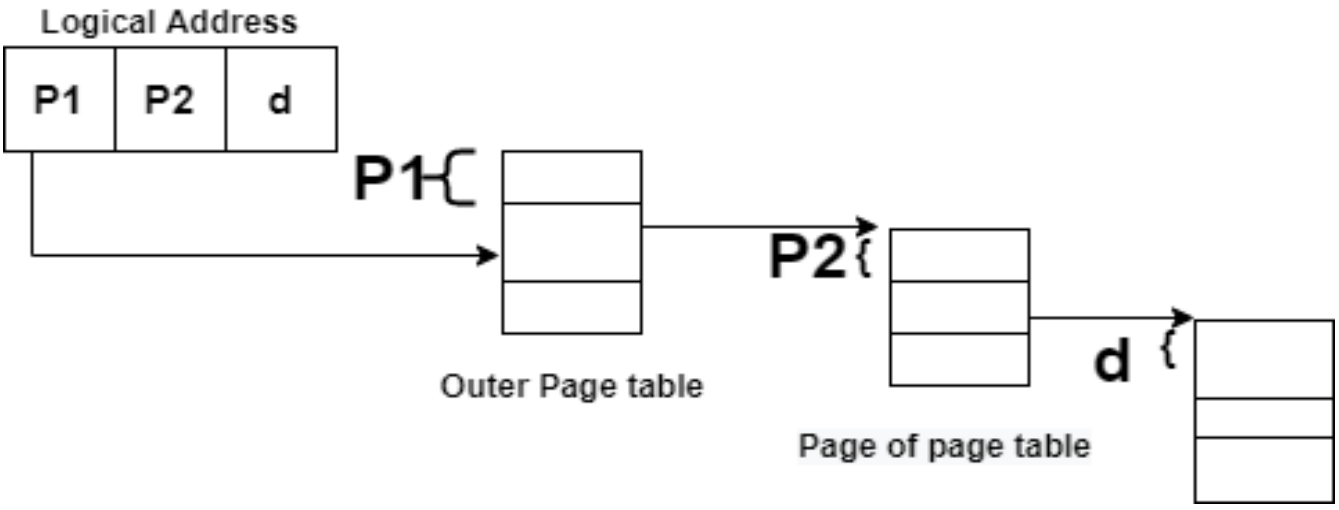


- In the above diagram,

P1 is an index into the Outer Page table.

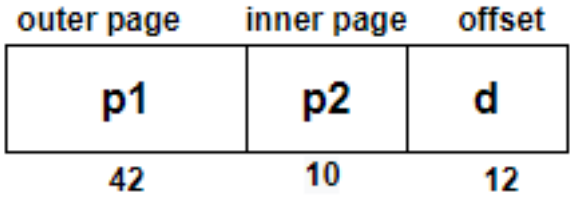
P2 indicates the displacement within the page of the Inner page Table.

- As address translation works from outer page table inward so is known as forward-mapped Page Table.
- Below given figure below shows the Address Translation scheme for a two-level page table

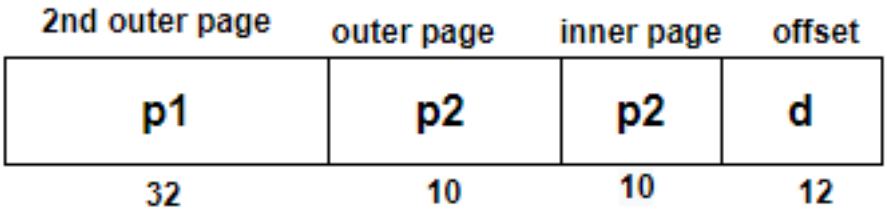


Three Level Page Table

- For a system with 64-bit logical address space, a two-level paging scheme is not appropriate.
- Let us suppose that the page size, in this case, is 4KB.If in this case, we will use the two-page level scheme then the addresses will look like this:

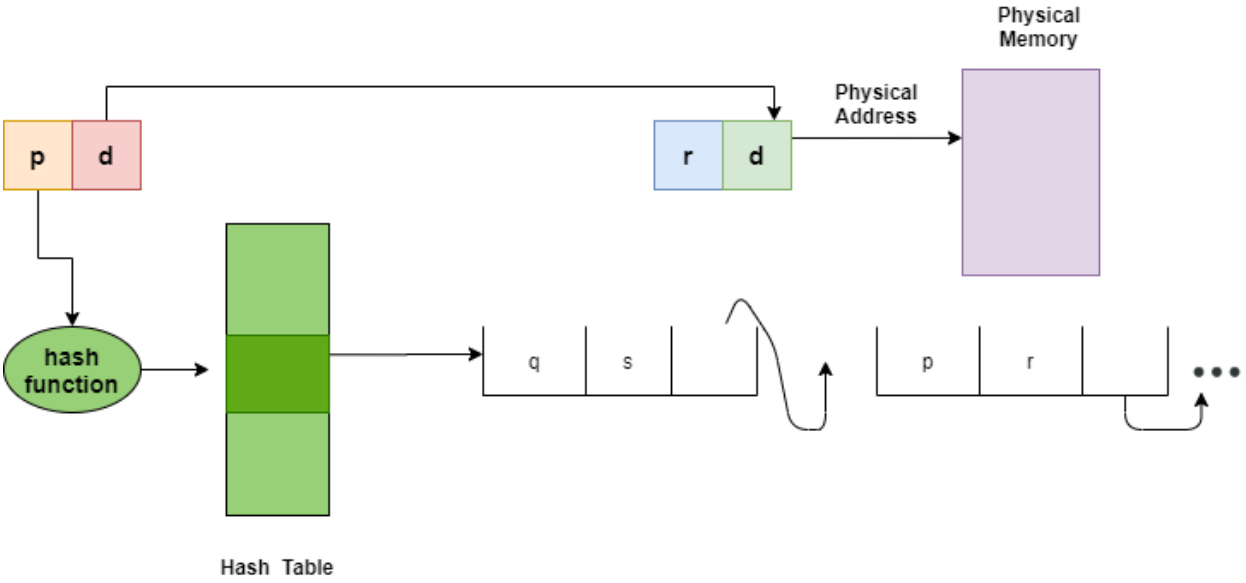


- Thus in order to avoid such a large table, there is a solution and that is to divide the outer page table, and then it will result in a Three-level page table:



2. <u>Hashed Page Tables</u>

- This approach is used to handle address spaces that are larger than 32 bits.
- In this virtual page, the number is hashed into a page table.
- This Page table mainly contains a chain of elements hashing to the same elements.
- Each element mainly consists of :
- The virtual page number
- The value of the mapped page frame.
- A pointer to the next element in the linked list.
- Given below figure shows the address translation scheme of the Hashed Page Table:



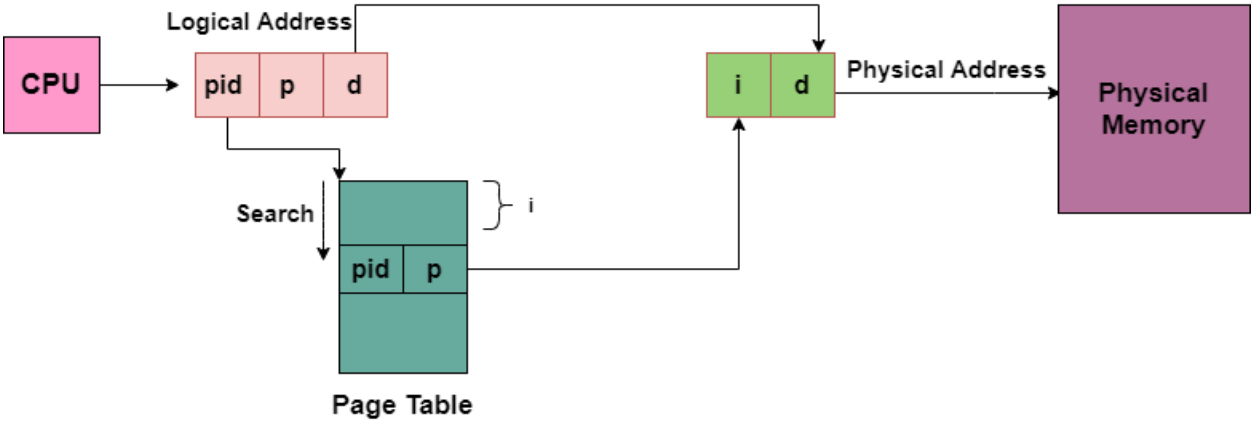
The above Figure shows Hashed Page Table

- The Virtual Page numbers are compared in this chain searching for a match; if the match is found then the corresponding physical frame is extracted.
- In this scheme, a variation for 64-bit address space commonly uses clustered page tables.

3. Inverted Page Tables

- The Inverted Page table basically combines A page table and A frame table into a single data structure.
- There is one entry for each virtual page number and a real page of memory
- And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
- Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

Given below figure shows the address translation scheme of the Inverted Page Table:



Memory Management Unit (MMU)

- MMU is a computer hardware component that responsible for all memory and caching operations which are associated with the CPU.

Functions of MMU, specifically in the context of paging:

- **Virtual Memory Translation:** The MMU translates virtual addresses generated by a process into physical addresses in RAM (Random Access Memory). This translation allows processes to run as if they have access to a large, continuous block of memory, even if the physical memory is fragmented or limited.
- **Paging:** Paging is a memory management scheme where physical memory is divided into fixed-size blocks called "frames," and logical memory is divided into equally sized blocks called "pages." The MMU maps pages to frames, enabling efficient allocation and management of memory.

- **Page Tables:** The MMU uses page tables to maintain the mapping between virtual pages and physical frames. Each process has its own page table, which keeps track of the virtual-to-physical address translations.
- **Page Faults:** When a process accesses a page that is not in physical memory, a page fault occurs. The MMU triggers an interrupt, and the operating system must load the required page from secondary storage (usually a hard disk) into a free frame in RAM.
- **Page Replacement:** If all frames in physical memory are in use, the operating system must choose a page to evict. This process, known as page replacement, is often managed using algorithms like the Least Recently Used (LRU) or the Second Chance algorithm.

2. **Segmentation:**

- A process is divided into Segments.
- The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments.
- Segmentation gives the user's view of the process which paging does not provide.
- Here the user's view is mapped to physical memory.

Types of Segmentation in Operating Systems

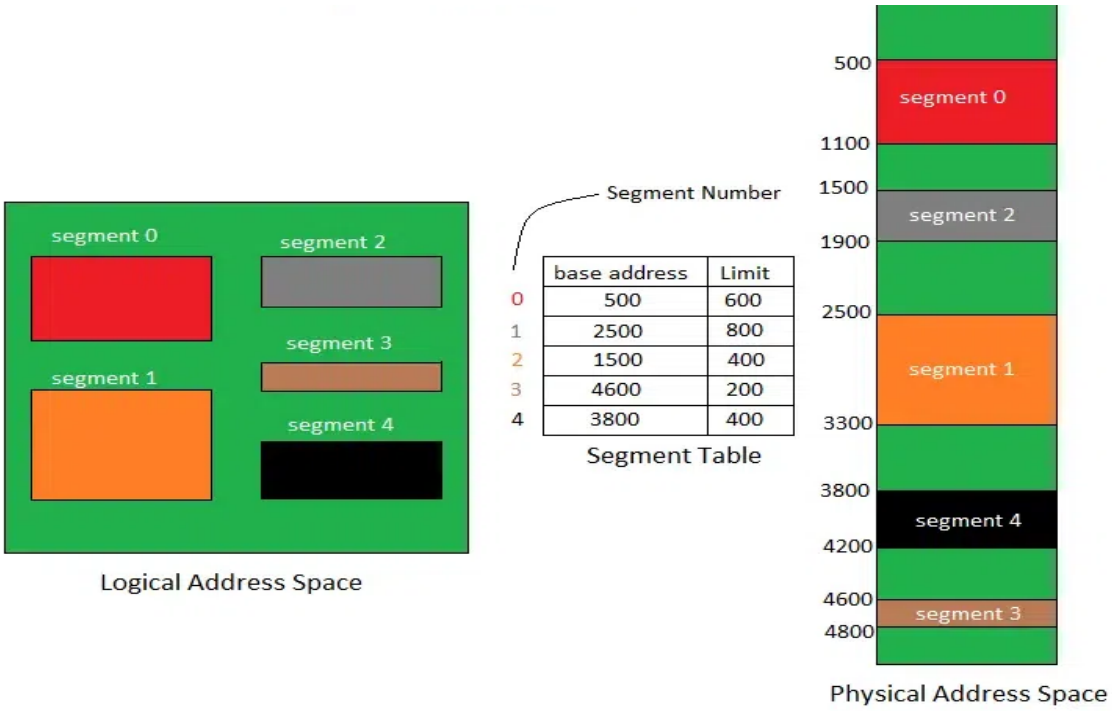
- **Virtual Memory Segmentation:** Each process is divided into several segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.
- **Simple Segmentation:** Each process is divided into several segments, all of which are loaded into memory at run time, though not necessarily contiguously.

Segment table:

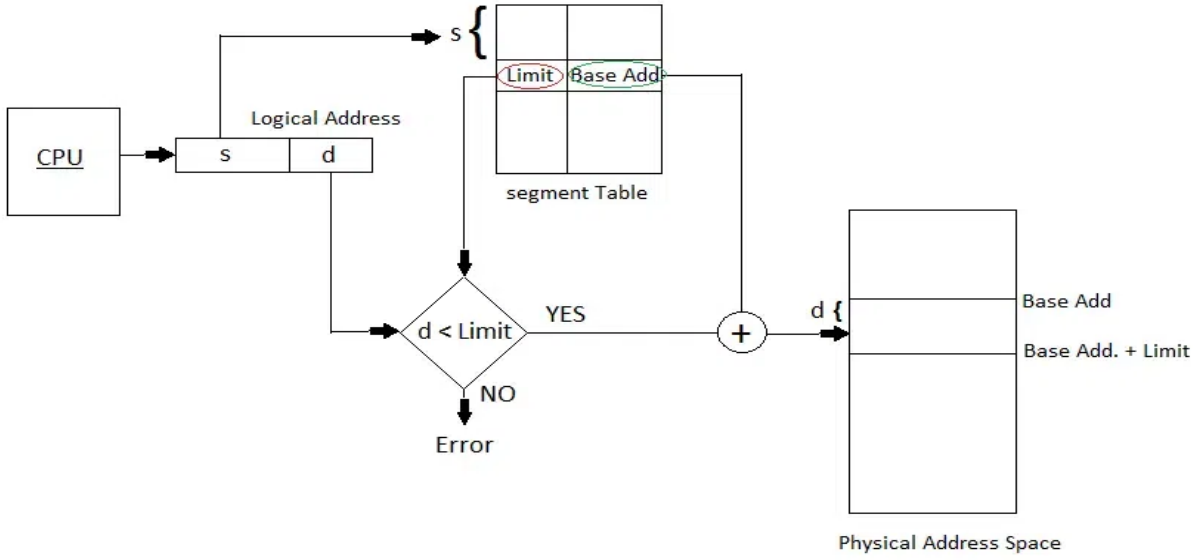
It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

Base Address: It contains the starting physical address where the segments reside in memory.

Segment Limit: Also known as segment offset. It specifies the length of the segment.



Translation of Two-dimensional Logical Address to Dimensional Physical Address.



Translation

The address generated by the CPU is divided into:

Segment number (s): Number of bits required to represent the segment.

Segment offset (d): Number of bits required to represent the position of data within a segment.

Advantages of Segmentation in Operating System

- **Reduced Internal Fragmentation:** Segmentation can reduce internal fragmentation compared to fixed-size paging, as segments can be sized according to the actual needs of a process.
- **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.
- **Sharing:** Segmentation allows for sharing of memory segments between processes. This can be useful for inter-process communication or for sharing code libraries.
- **Protection:** Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process's memory segment. This can help increase the security and stability of the system.

Disadvantages of Segmentation in Operating System

- **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.
- **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.
- **Overhead:** Using a segment table can increase overhead and reduce performance. Each segment table entry requires additional memory, and accessing the table to retrieve memory locations can increase the time needed for memory operations.
- **Complexity:** Segmentation can be more complex to implement and manage than paging. Managing multiple segments per process can be challenging, and the potential for segmentation faults can increase as a result.

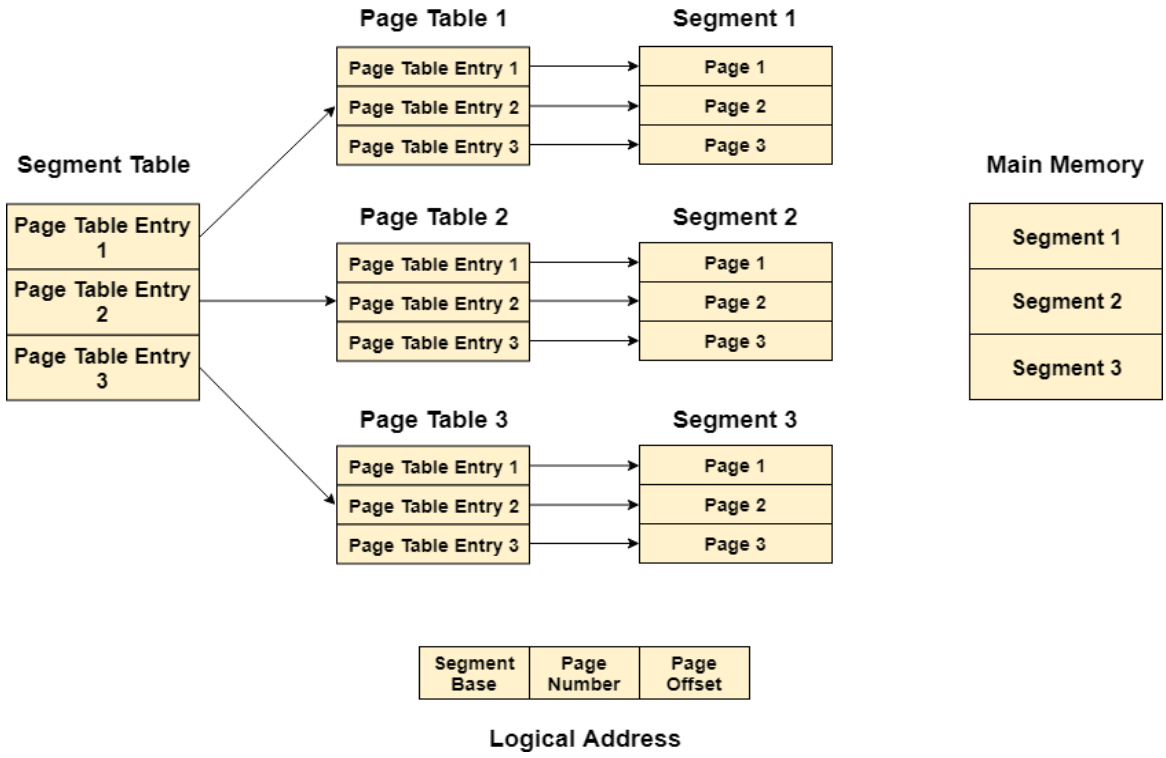
Paging	Segmentation
In Paging, the program is divided into fixed or mounted size pages.	In Segmentation, the program is divided into variable size segments.
For the paging operating system is accountable.	For segmentation compiler is accountable.
Page size is determined by hardware.	Here, the segment size is given by the user.
It is faster in comparison to segmentation.	Segmentation is slow.
Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
In paging, the logical address is split into a page number and page offset.	Here, the logical address is split into segment number and segment offset.
Paging comprises a page table that encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.
In paging, the operating system must maintain a free frame list.	In segmentation, the operating system maintains a list of holes in the main memory.
Paging is invisible to the user.	Segmentation is visible to the user.
It is hard to allow sharing of procedures between processes.	Facilitates sharing of procedures between the processes.
This protection is hard to apply.	Easy to apply for protection in segmentation.
Paging results in a less efficient system.	Segmentation results in a more efficient system.

Aspect	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
Method	Allocates memory in a contiguous block to a process	Allocates memory to a process in non-contiguous blocks
Block Size	Memory allocated in a single, continuous chunk	Memory allocated in non-contiguous blocks of varying sizes
Management	Easy to manage by the operating system	Requires additional overhead and can be more complicated to manage
Memory Usage	May result in memory wastage and external fragmentation	Efficient use of memory and reduces fragmentation within memory blocks
Suitable For	Systems with limited amounts of memory and fast access to memory is important	Larger memory sizes and systems that require more efficient use of memory
Advantages	Simple and efficient technique for memory management	More flexible and efficient technique for larger memory sizes and systems that require more efficient use of memory
Disadvantages	Can be inflexible and result in memory wastage and fragmentation	Requires additional overhead and can be more complicated to manage

3.10.Segmentation with Paging:

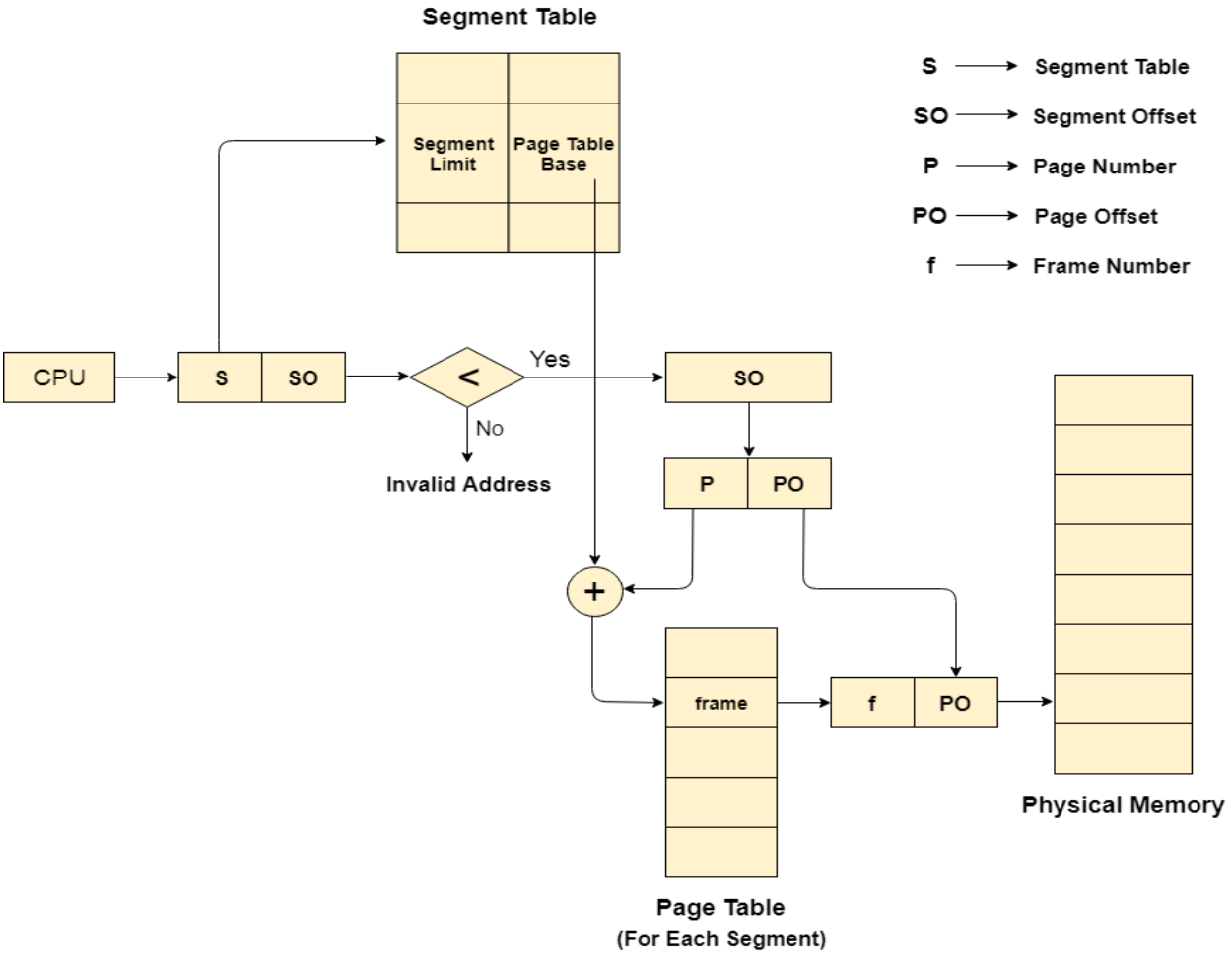
- Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.
- In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.
- Pages are smaller than segments.
- Each Segment has a page table which means every program has multiple page tables.

- The logical address is represented as Segment Number (base address), Page number and page offset.
- Segment Number → It points to the appropriate Segment Number.
- Page Number → It Points to the exact page within the segment
- Page Offset → Used as an offset within the page frame
- Each Page table contains the various information about every page of the segment.
- The Segment Table contains the information about every segment.
- Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



Translation of logical address to physical address

- The CPU generates a logical address which is divided into two parts:
- Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.
- The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



Advantages of Segmented Paging

- It reduces memory usage.
- Page table size is limited by the segment size.
- Segment table has only one entry corresponding to one actual segment.
- External Fragmentation is not there.
- It simplifies memory allocation.

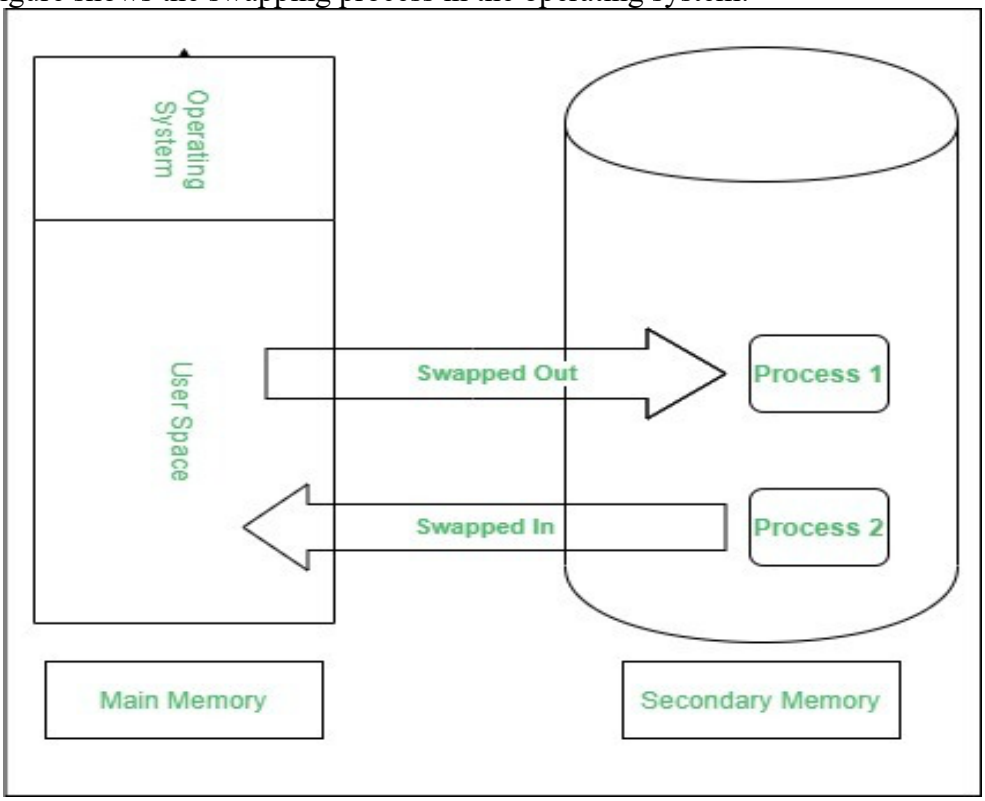
Disadvantages of Segmented Paging

- Internal Fragmentation will be there.
- The complexity level will be much higher as compared to paging.
- Page Tables need to be contiguously stored in the memory.

3.11.Swapping:

- Swapping in an operating system is a process that moves data or programs between the computer's main memory (RAM) and a secondary storage.
- This helps manage the limited space in RAM and allows the system to run more programs than it could otherwise handle simultaneously.

The below figure shows the swapping process in the operating system:



Swapping has been subdivided into two concepts: swap-in and swap-out.

- Swap-out is a technique for moving a process from RAM to the hard disc.
- Swap-in is a method of transferring a program from a hard disc to main memory, or RAM.

Process of Swapping

- When the RAM is full and a new program needs to run, the operating system selects a program or data that is currently in RAM but not actively being used.
- The selected data is moved to the secondary storage, making space in RAM for the new program.
- When the swapped-out program is needed again, it can be swapped back into RAM, replacing another inactive program or data if necessary.

Advantages of Swapping in OS

- Swapping processes improve the degree of multi-programming.
- It provides the advantages of Virtual Memory for the user.
- It helps in better memory management and efficient utilization of RAM.

Disadvantages of Swapping in OS

- The swapping algorithm must be perfect; otherwise, the number of Page Faults will increase, and performance will decrease.
- Inefficiency will occur when there are common/shared resources between many processes.
- The user will lose information if there is heavy swapping, and the computer loses its power.

S.No.	Swapping	Paging
1.	It is process where the entire process is copied to another location.	It is a memory allocation technique.
2.	This process occurs when the entire process has been transferred to the disk.	This process occurs when a part of a process is transferred to the disk.
3.	Here, the data is swapped temporarily from the main memory to a secondary memory.	The contiguous block of memory is made non-contiguous but it consists of fixed size called frames known as pages.
4.	It can be done without using any memory management methods.	It uses non-contiguous memory management technique.
5.	It can be done by processes that are inactive as well.	Only a process that is currently active can perform paging operation.
6.	It helps give a direction with respect to the solution.	There is no suggestion about the solution in this technique.

3.12. Memory Management with Swapping:

1. **Bit maps:**

- A **bitmap** is an array of bits, where each bit corresponds to a fixed-sized block of memory.
- A bit value of 0 typically indicates that the block is **free**, and a bit value of 1 indicates that the block is **allocated**.

Steps in Memory Management with Bitmaps

1. Initialization:

2. Allocation:

3. Deallocation:

Advantages of Using Bitmaps

1. Simple Implementation:
2. Efficient for Fixed-Sized Blocks:
3. Compact Representation:

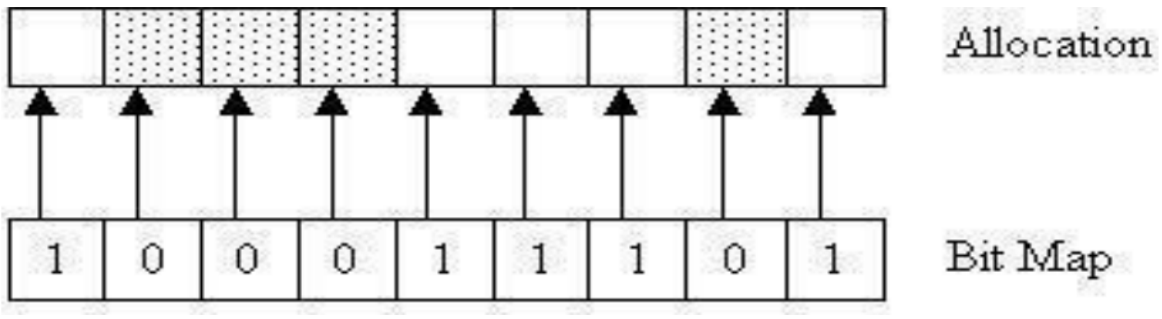
Disadvantages of Using Bitmaps

1. Internal Fragmentation:
2. Search Overhead:
3. Limited Flexibility:

Applications of Bitmap-Based Memory Management

1. Disk Space Management:
2. Paging in Virtual Memory:
3. Embedded Systems:

- The other problem with a bit map memory scheme is when we need to allocate memory to a process. Assume the allocation size is 4 bytes. If a process requests 256 bytes of memory, we must search the bit map for 64 consecutive zeroes.
- This is a slow operation and for this reason bit maps are not often used.



2. **Memory Usage with Linked Lists:**

- It is a technique where the operating system uses a linked data structure to track free and allocated memory blocks
- Free and allocated memory can be represented as a linked list. The memory shown above as a bit map can be represented as a linked list as follows.

Each entry in the list holds the following data

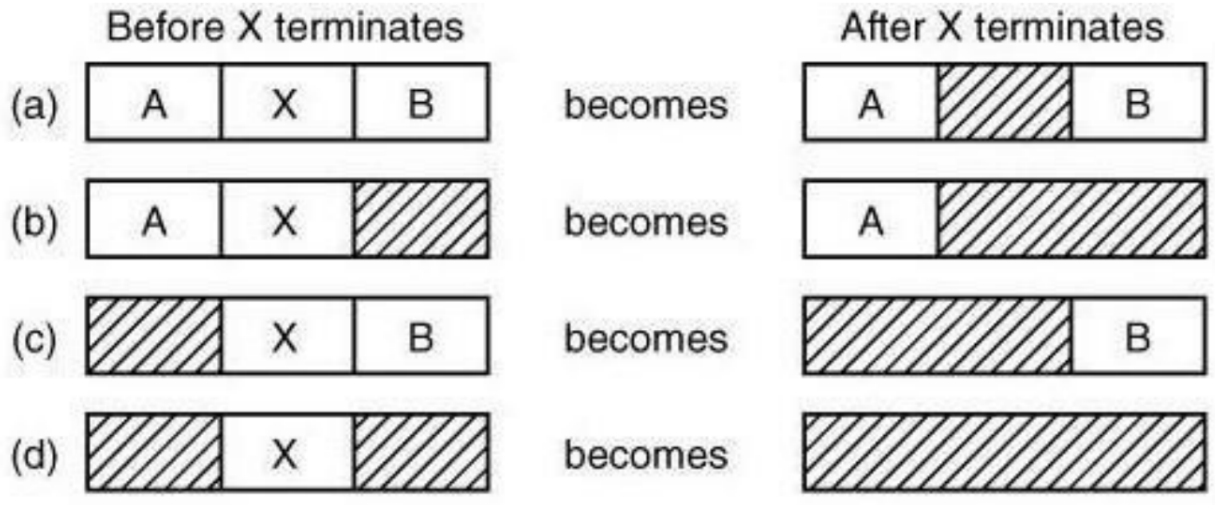
• P or H: for Process or Hole

• Starting segment address

• the length of the memory segment

• the next pointer is not shown but assumed to be present

- It is possible that two processes can be next to each other and we need to keep them as separate elements in the list so that if one process ends we only return the memory for that process.
- Consecutive holes, on the other hand, can always be merged into a single list entry.
- A terminating process can have four combinations of neighbours.
- If X is the terminating process the four combinations are;



- In this example, the segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward.

- A terminating process normally has two neighbours (except when it is at the very top or very bottom of memory).
- These may be either processes or holes, leading to the four combinations shown in fig above. When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process (or an existing process being swapped in from disk).
- We assume that the memory manager knows how much memory to allocate.

Steps in Memory Management Using Linked Lists

1. Initialization:
2. Allocation:
3. Deallocation:
4. Compaction (Optional):

Advantages of Using Linked Lists

1. Flexible Block Sizes:
2. Efficient Deallocation:
3. Dynamic Growth:

Disadvantages of Using Linked Lists

1. Traversal Overhead:
2. Fragmentation:
3. Memory Overhead:

3.12. Basic Memory Management (Without Swapping or Paging):

- Entire process remains in memory from start to finish and does not move.
- The sum of the memory requirements of all jobs in the system cannot exceed the size of physical memory.

I. Mono-programming without swapping or paging (Single User):

- The simplest memory management scheme is to run just one program at a time, sharing the memory between program and operating system.
- When the system is organized in this way, only one program at a time can be running.
- As soon as the user types a command the operating system copy the requested program from disk to memory and executes it. When the program is finished, the operating system displays a prompt character and waits for new command. When it receives the command, it loads the new program into memory.

Advantages:

- Simplicity
- It does not require expertise to understand or use such system.

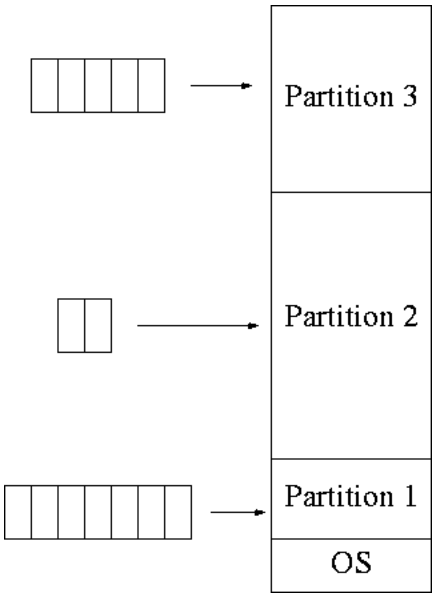
Disadvantages:

- **Underutilization of memory:** Since at a time only one program resides in memory, it may not occupy whole memory which results in wastage of memory space.
- **Idle CPU:** CPU sits idle during the period when a running program needs some input/output operation.

II. Multiprogramming with fixed partitions:

- Most modern system allows multiple processes to run at the same time. Multiple processes running at once mean that when one process is waiting for input/output operation, another process can use the CPU.
- Thus, multiprogramming increases the CPU utilization. The easiest way to achieve multiprogramming is to divide memory into several partitions. When a job arrives, it can be put into the input queue for the smallest partition enough to hold it.
- The partitions are fixed in this scheme. Fixed memory partition with separate input queue has an individual input queue for each partition and programs are assigned to each partition from that queue.

- Fixed memory partition with single input queue has only one input queue for all partitions and programs are assigned to the partition from the front of queue.



3.13.Relocation and Protection:

1. Relocation:

- Relocation is the process of dynamically adjusting the memory addresses used in a program so it can be loaded and executed at any memory location.
- Relocation ensures the operating system can efficiently manage memory and run multiple programs simultaneously.

Types of Relocation

1. **Static Relocation:**
    1. Performed during program compilation or linking.
    2. Absolute memory addresses are assigned before execution.
    3. Lack of flexibility and less efficient use of memory.
2. **Dynamic Relocation:**
    1. Handled by the operating system during program loading or execution.
    2. The program uses logical addresses (relative addresses).
    3. More flexible and allows better memory utilization.

Advantages of Relocation

- Efficient memory utilization.
- Support for multi-programming and multitasking.

2. Memory Protection:

- It ensures that one process cannot access or modify the memory allocated to another process or the OS itself.
- This mechanism is vital for maintaining system stability, security, and efficient multitasking

Memory Protection Techniques:

1. Base and Limit Registers

- Base Register: Holds the starting address of a process's memory.
- Limit Register: Specifies the size of the process's memory.

2. Segmentation

- Divides memory into segments, each corresponding to a logical unit such as code, data, or stack.

3. Paging

- Divides memory into fixed-size blocks called pages (logical memory) and frames (physical memory).

4. Virtual Memory

- Extends paging by using disk space as additional memory.
- Protects processes by maintaining separate virtual address spaces for each process.

5.Hardware Support for Memory Protection

Benefits of Memory Protection

- Prevents accidental or malicious corruption of memory.
- Supports multitasking by isolating processes.
- Enhances system security by protecting the OS kernel.
- Facilitates debugging by detecting invalid memory accesses (e.g., segmentation faults).

3.13. Virtual Memory:

- Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited.
- It allows the system to compensate for physical memory shortages, enabling larger applications to run on systems with less RAM.

3.13.1. Working of Virtual Memory:

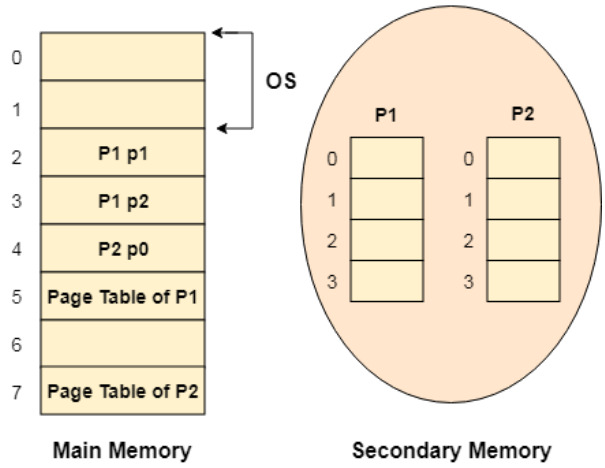
- Virtual Memory is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.
- All memory references within a process are logical addresses that are dynamically translated into physical address at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during execution.
- A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

Snapshot of a virtual memory management system

- Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each.
- The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.
- The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.
- The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.

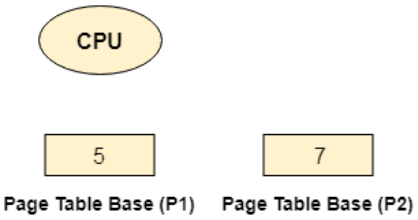
	Frame	Present / Absent	D Bit	Reference bit	Protection
0		0	0	0	
1	2	1	0	1	
2	3	1	1	0	
3		0	0	0	

Page Table of P1



	Frame	Present / Absent	D Bit	Reference bit	Protection
0	4	1	0	1	
1		0	0	0	
2		0	0	0	
3		0	0	0	

Page Table of P2



Advantages of Virtual Memory

- The degree of Multiprogramming will be increased.
- User can run large application with less real RAM.
- There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

- The system becomes slower since swapping takes time.
- It takes more time in switching between applications.
- The user will have the lesser hard disk space for its use.

Types of Virtual Memory

- In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU.
- The CPU generates virtual addresses that the MMU translates into physical addresses.

There are two main types of virtual memory:

- **Paging**
- **Segmentation**

Applications of Virtual memory:

1. **Increased Effective Memory:** One major practical application of virtual memory is, virtual memory enables a computer to have more memory than the physical memory using the disk space. This allows for the running of larger applications and numerous programs at one time while not necessarily needing an equivalent amount of DRAM.
2. **Memory Isolation:** Virtual memory allocates a unique address space to each process and that also plays a role in process segmentation. Such separation increases safety and reliability since one process cannot interact with and or modify another's memory space through a mistake, or even a deliberate act of vandalism.
3. **Efficient Memory Management:** Virtual memory also helps in better utilization of the physical memories through methods that include paging and segmentation
4. **Simplified Program Development:** For case of programmers, they don't have to consider physical memory available in a system in case of having virtual memory.

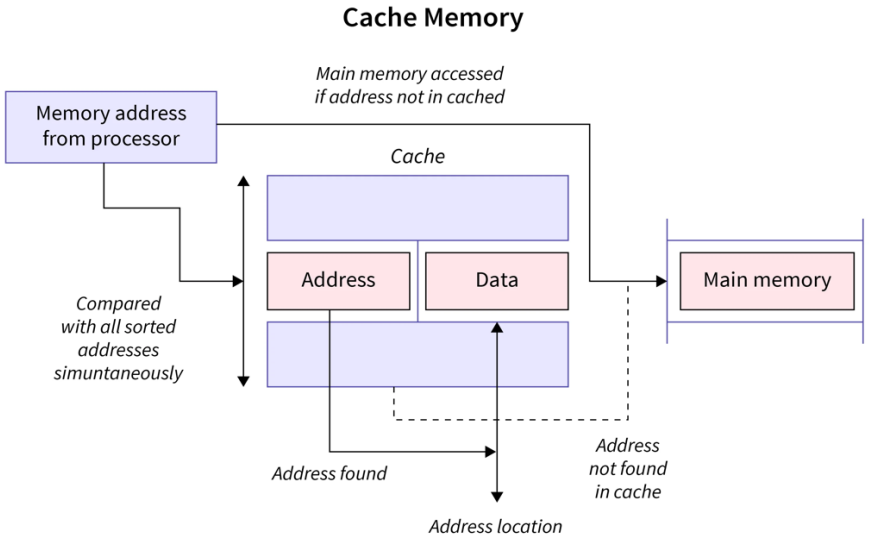
Feature	Virtual Memory	Physical Memory (RAM)
Definition	An abstraction that extends the available memory by using disk storage	The actual hardware (RAM) that stores data and instructions currently being used by the CPU
Location	On the hard drive or SSD	On the computer's motherboard
Speed	Slower (due to disk I/O operations)	Faster (accessed directly by the CPU)
Capacity	Larger, limited by disk space	Smaller, limited by the amount of RAM installed
Cost	Lower (cost of additional disk storage)	Higher (cost of RAM modules)
Data Access	Indirect (via paging and swapping)	Direct (CPU can access data directly)
Volatility	Non-volatile (data persists on disk)	Volatile (data is lost when power is off)

3.14. TLB in OS:

- The Translation Lookaside Buffer (TLB) in an operating system is a hardware component that serves as a cache for the virtual-to-physical address translation process.
- It stores a subset of recently used virtual memory page mappings, making it faster to translate virtual addresses generated by programs into corresponding physical addresses in main memory (RAM).
- When a program accesses memory, the TLB is consulted, and if it contains the necessary translation, this results in a TLB hit, which significantly speeds up memory access.

3.14.1. Need of TLB:

- The Translation Lookaside Buffer (TLB) is needed in computer systems, particularly in the context of virtual memory and memory management, for several crucial reasons:



1. **Faster Memory Access:**

- When a program accesses memory, a TLB hit can provide the required translation almost instantly, reducing the time it takes to fetch data from main memory (RAM).
- This leads to faster program execution and improved system performance.

2. **Efficient Use of Memory:**

- Virtual memory systems allow programs to access more memory than physically available.
- This reduces contention for memory resources and improves overall system efficiency.

3. **Reduced Power Consumption:**

- Accessing main memory consumes more power compared to accessing the TLB.

4. **Minimized Page Table Access:**

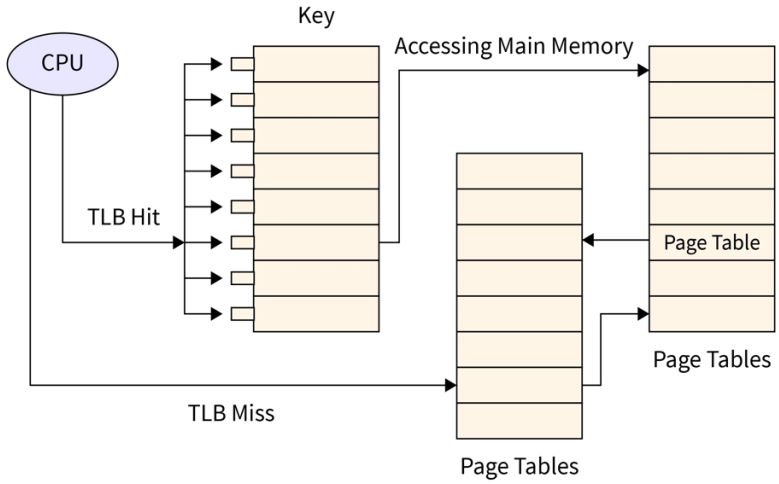
- Without TLB, every virtual-to-physical address translation would require accessing the page table in main memory, which can be time-consuming.

5. **Improved System Responsiveness:**

- TLB helps in maintaining the system's responsiveness.
- It ensures that memory access operations, which are fundamental to program execution, are as quick as possible, reducing perceived delays and ensuring a smoother user experience.

3.14.2. Steps in TLB Hit:

- A TLB hit occurs when the Translation Lookaside Buffer (TLB) contains the required virtual-to-physical address translation for a memory access operation.
- When a TLB hit happens, the steps involved are relatively straightforward and efficient:



- CPU generates a virtual memory address, typically as part of a load or store instruction.
- CPU checks for a match between the generated virtual address and the entries stored in the TLB.
- CPU retrieves the corresponding physical address from the TLB entry. This physical address is used to access the actual data stored in main memory (RAM).

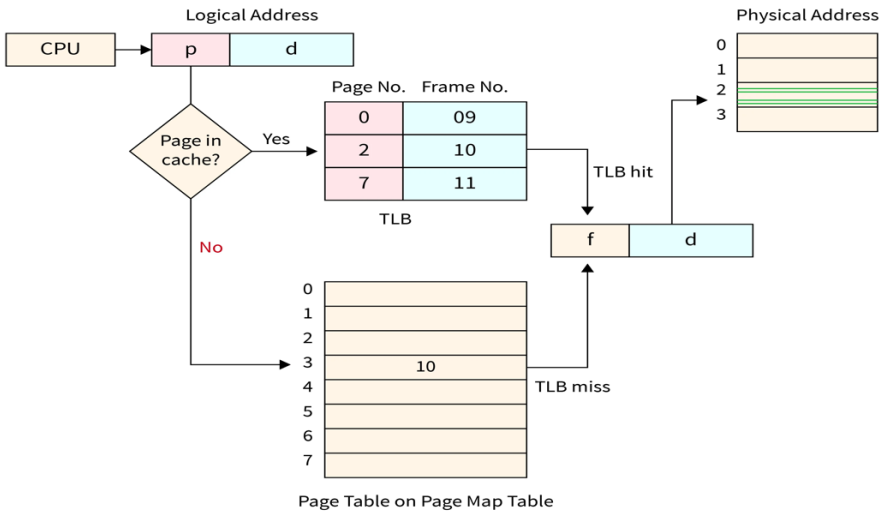
- With the physical address obtained from the TLB, the CPU can access the data in main memory.

3.14.3. Steps in TLB Miss:

- When a TLB miss occurs, it means that the Translation Lookaside Buffer (TLB) does not contain the required virtual-to-physical address translation for a memory access operation. In this case, the operating system and hardware must perform additional steps to retrieve the necessary translation.

Here are the steps involved in a TLB miss:

- A program or process initiates a memory access operation by generating a virtual memory address.
- CPU checks the TLB for a match between the generated virtual address and the entries stored in the TLB. In the case of a TLB miss, no matching entry is found in the TLB.
- Now the CPU must consult the page table, which is a data structure in main memory maintained by the operating system.
- The CPU accesses the page table to retrieve the page table entry corresponding to the virtual address.
- CPU updates the TLB with the new translation information.
- With the translation information from the page table entry, the CPU calculates the physical address corresponding to the original virtual address.
- The CPU can now use the calculated physical address to access the data stored in main memory (RAM).



3.14.4. EAT:

- In the context of a Translation Lookaside Buffer (TLB) in operating systems, "Effective Access Time" (EAT) refers to the total time it takes to complete a memory access operation, taking into account the possibility of both TLB hits and TLB misses.
- It is a critical metric used to evaluate the overall performance of a memory system with a TLB.

The Effective Access Time (EAT) can be calculated using the following formula:

EAT=(Hit Time∗Hit Rate)+(Miss Penalty∗Miss Rate)

EAT=(Hit Time∗Hit Rate)+(Miss Penalty∗Miss Rate)

Where:

Hit Time:

This is the time it takes to perform a TLB lookup and retrieve the physical address if there is a TLB hit. In other words, it represents the time it takes for a successful TLB access.

Hit Rate:

This is the probability that a memory access results in a TLB hit. It is calculated as the number of TLB hits divided by the total number of memory accesses.

Miss Penalty:

This is the time it takes to resolve a TLB miss. It includes the time required to access the page table in main memory and potentially update the TLB with the missing translation.

Miss Rate:

This is the probability that a memory access results in a TLB miss. It is calculated as the number of TLB misses divided by the total number of memory accesses.

3.14.5. Advantages of TLB:

- **Faster Memory Access:**

TLB accelerates the virtual-to-physical address translation process.

- **Reduced Latency:**

By caching frequently used address translations, TLB ensures that frequently accessed data can be retrieved more quickly, improving system responsiveness.

- **Efficient Use of Virtual Memory:**

TLB helps efficiently manage this virtual memory by caching frequently used address translations, reducing the need to access the page table in main memory for every memory reference.

- **Lower Power Consumption:**

By reducing the frequency of memory accesses, TLB can contribute to lower power consumption, which is particularly important in mobile devices and energy-efficient computing environments.

- **Minimized Page Table Access:**

Without TLB, every virtual-to-physical address translation would require accessing the page table in main memory, which can be time-consuming.

- **Improved System Responsiveness:**

TLB ensures that memory access operations, which are fundamental to program execution, are as quick as possible, reducing perceived delays and ensuring a smoother user experience.

3.14.6. Limitations of TLB

- **Limited Size:**

TLBs have a finite number of entries, and their size is limited by hardware constraints.

- **Management Overhead:**

Managing the TLB, including updating and evicting entries, adds some overhead to memory access operations.

- **Complex Hardware:**

TLBs require additional hardware components, which can increase the cost and complexity of the CPU or memory management unit.

- **Cache Consistency:**

In multiprocessor systems or systems with multiple caches, maintaining cache consistency, including the TLB, can be challenging.

- **TLB Miss Penalty:**

When a TLB miss occurs, and the required translation is not in the TLB, it results in a longer memory access time.

- **Inefficient Use of Space:**

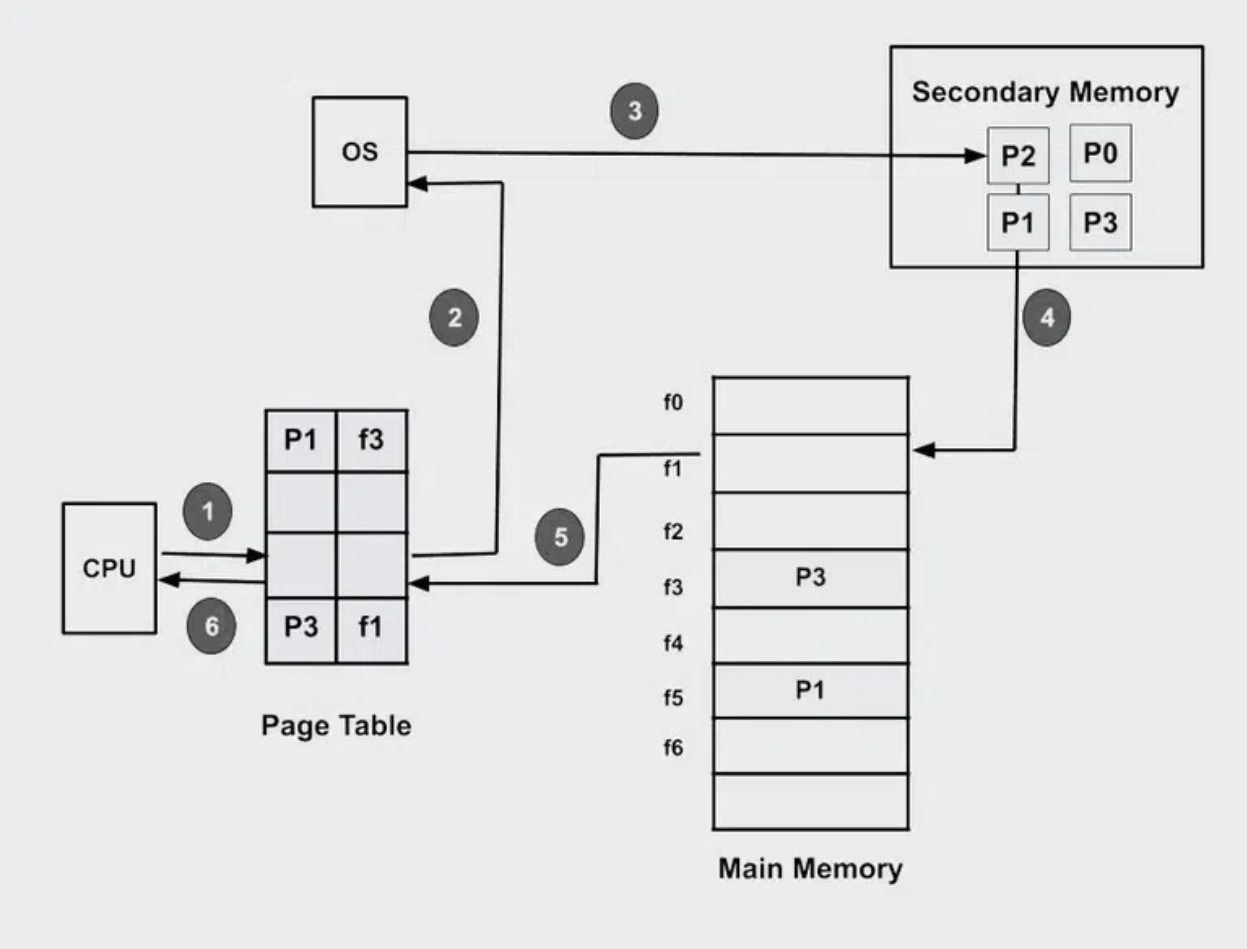
TLBs may not always be fully utilized, and some entries may become stale if the mapping changes frequently.

3.15. Demand Paging:

- Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU.
- In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start.
- A page fault occurred when the program needed to access a page that is not currently in memory.

3.15.1. Working Process of Demand Paging

Let us understand this with the help of an example. Suppose we want to run a process P which have four pages P0, P1, P2, and P3. Currently, in the page table, we have pages P1 and P3.



The OS s demand paging mechanism follows a few steps in its operation.

Program Execution: Upon launching a program, the operating system allocates a certain amount of memory to the program and establishes a process for it.

Creating Page Tables: To keep track of which program pages are currently in memory and which are on disk, the operating system makes Page Table for each process.

Handling Page Fault: When a program tries to access a page that isn't in memory at the moment, a page fault happens. In order to determine whether the necessary page is on disk, the operating system pauses the application and consults the page tables.

Page Fetch: The operating system loads the necessary page into memory by retrieving it from the disk if it is there.

The page's new location in memory is then reflected in the page table.

Resuming The Program: The operating system picks up where it left off when the necessary pages are loaded into memory.

Page Replacement: If there is not enough free memory to hold all the pages a program needs, the operating system may need to replace one or more pages currently in memory with pages currently in memory. on the disk. The page replacement algorithm used by the operating system determines which pages are selected for replacement.

Page Cleanup: When a process terminates, the operating system frees the memory allocated to the process and cleans up the corresponding entries in the page tables.

Advantages of Demand Paging

1. **Efficient use of physical memory:** Query paging allows for more efficient use because only the necessary pages are loaded into memory at any given time.
2. **Support for larger programs:** Programs can be larger than the physical memory available on the system because only the necessary pages will be loaded into memory.
3. **Faster program start:** Because only part of a program is initially loaded into memory, programs can start faster than if the entire program were loaded at once.
4. **Reduce memory usage:** Query paging can help reduce the amount of memory a program needs, which can improve system performance by reducing the amount of disk I/O required.

Disadvantages of Demand Paging

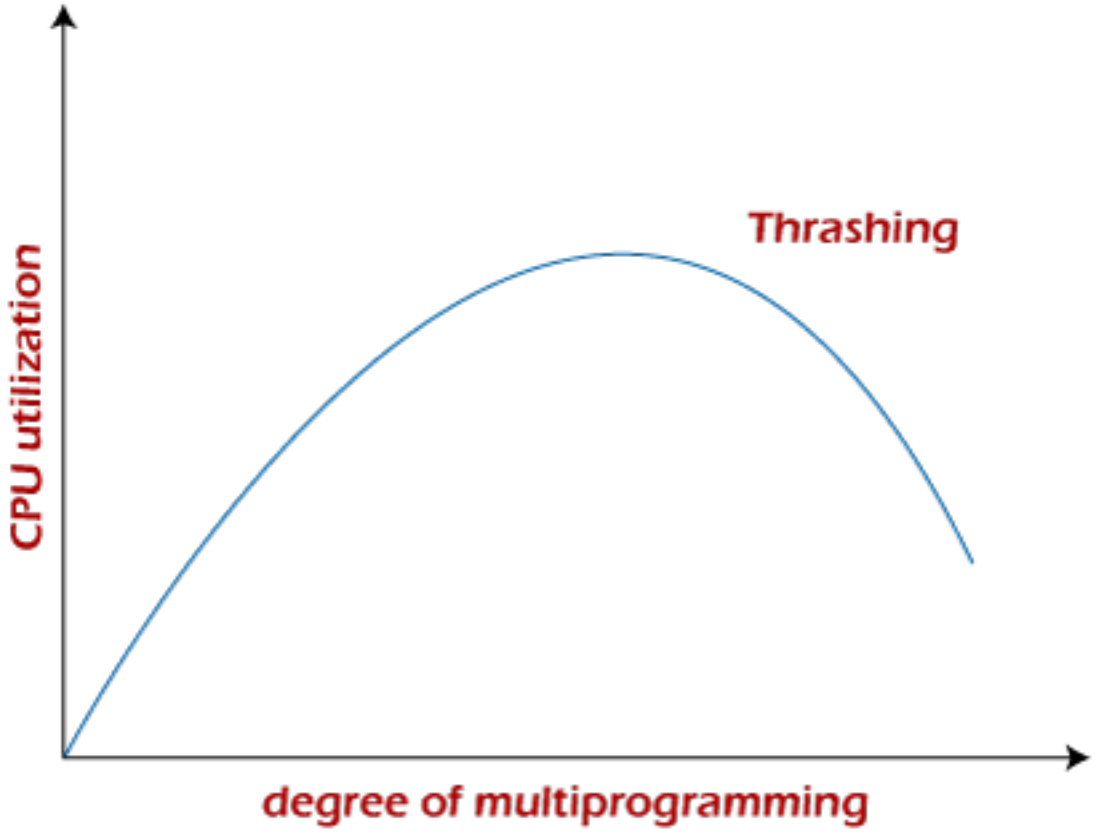
1. **Page Fault Overload:** The process of swapping pages between memory and disk can cause a performance overhead, especially if the program frequently accesses pages that are not currently in memory.
2. **Degraded Performance:** If a program frequently accesses pages that are not currently in memory, the system spends a lot of time swapping out pages, which degrades performance.
3. **Fragmentation:** Query paging can cause physical memory fragmentation,degrading system performance over time.
4. **Complexity:** Implementing query paging in an operating system can be complex, requiring complex algorithms and data structures to manage page tables and swap space.

3.15.2. Common Algorithms used for Demand Paging in OS

1. **First In First Out (FIFO):** This algorithm is used to manage resources in operating systems like CPU time, memory, and input/output operations. The first process that enters the queue gets executed first, and the rest wait till they are executed. Here, algorithms are used to manage the queue to make sure that the processes are added and removed correctly. When a new process arrives, it is added to the end of the queue using the enqueue operation
2. **Optimal Page Algorithm:** As the name suggests, the Optimal page replacement algorithm is used to replace pages. This works by replacing the page whose demand in the future is the least as compared to other pages from the frame. Belady's Anomaly does not occur in this algorithm as it uses a stack based algorith
3. **Least Recently Used (LRU) Algorithm:** Whenever a page fault occurs, the least recently used page will be replaced with a new page. Hence the page not utilised for the longest time in the memory gets replaced. The Least Recently Used (LRU) Algorithm gives fewer page faults than any other algorithm and is capable of complete analysis.
4. **Page Buffering Algorithm:** This algorithm is used to get a process to start quickly and keep a pool of free frames.
5. **Least Frequently Used (LFU) Algorithm:** In this algorithm, the least frequently used pages are replaced. The page with the least visits in a given period of time is removed.The page that comes first is replaced first, given that the frequency of the page remains constant.

3.16. Thrashing:

- **Thrash** is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory.
- **Thrashing** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing.
- It causes the performance of the computer to degrade or collapse.
- Thrashing is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages.
- This state in the operating system is known as thrashing.
- Because of thrashing, the CPU utilization is going to be reduced or negligible.



3.16.1. Causes of Thrashing in OS

- **Insufficient Physical Memory:** When the available physical RAM is not enough to hold the needed data and programs, leading to excessive swapping between RAM and disk.
- **Overallocation of Memory:** When an operating system allocates more memory to applications than what is available in physical memory, causing constant page swaps.

- **Poor Memory Management:** Inefficient use of memory by applications or the operating system, leading to frequent loading and unloading of data.
- **Excessive Multitasking:** Running too many applications or processes at the same time, which exceeds the system's memory capacity.
- **Highly Demanding Applications:** Applications that require a large amount of memory, leading to frequent swapping when they compete for resources.
- **Inefficient Swapping Strategy:** Poor management of the swap file or partition by the operating system, leading to inefficient use of disk space for memory swapping.

3.16.2. Techniques to Prevent Thrashing

1. Locality Model

- A locality is a set of pages that are actively used together.
- The locality model states that as a process executes, it moves from one locality to another. Thus, a program is generally composed of several different localities which may overlap.
- For example, when a function is called, it defines a new locality where memory references are made to the function call instructions, local and global variables, etc.
- Similarly, when the function is exited, the process leaves this locality.

2. Working-Set Model

- A process goes from locality to locality when it runs, while a locality is a set of pages that are actively being used together.
- The working set window is defined by the parameter Δ(theta) in this model.
- The working set is the collection of pages in the most current page references.
- The working set contains pages that are currently in use.
- If Δ is too small, the entire locality will not be covered.
- On the other hand, if Δ is too large, it may overlap numerous localities.
- The size of the working set is thus the essential feature.
- We calculate the working-set size, WSSi, for each process in the system if there are m processes.

  D = ΣWSSi, where D is the overall demand for frames.
- Each process actively makes use of the pages in its working set.

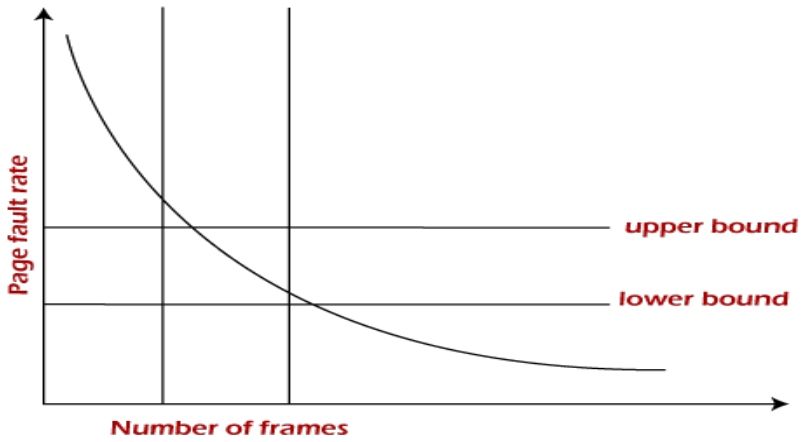
- As a result, WSSi frames are required by process i.
- If the overall demand exceeds the total amount of possible frames (D > m), thrashing will occur because certain processes will run out of frames.
- After choosing, the operating system monitors each process's working set and allocates enough frames to that working set to provide it with its working-set size.
- Another process can be started if there are enough spare frames.
- The OS selects a process to suspend if the sum of the working-set sizes exceeds the total number of available frames.
- The series of numbers in the following diagram represents the pages required by a process.
- If the working set size is set to nine, the following pages are displayed at times t1 and t2:

Computation of Working Sets

- Peter D. Denning introduced the working set parameter as T, which means that the working set is made up of all pages that have been referenced in the recent T seconds.
- The clock method may be extended to retain an idle time for each page.
- The working set contains pages with idle times shorter than T.

3. **Page Fault Frequency**

- A more direct approach to handle thrashing is the one that uses the Page-Fault Frequency.



- The problem associated with thrashing is the high page fault rate, and thus, the concept here is to control the page fault rate.
- If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

- Upper and lower limits can be established on the desired page fault rate, as shown in the diagram.
- If the page fault rate falls below the lower limit, frames can be removed from the process.
- Similarly, if the page faults rate exceeds the upper limit, more frames can be allocated to the process.
- In other words, the graphical state of the system should be kept limited to the rectangular region formed in the given diagram.
- If the page fault rate is high with no free frames, some of the processes can be suspended and allocated to them can be reallocated to other processes.
- The suspended processes can restart later.

Algorithms during Thrashing:

1. **Global Page Replacement**

- The Global Page replacement has access to bring any page, whenever thrashing found it tries to bring more pages.
- Due to this, no process can get enough frames and as a result, the thrashing will increase more and more.
- Thus the global page replacement algorithm is not suitable whenever thrashing happens.

2. **Local Page Replacement**

- Unlike the Global Page replacement, the local page replacement will select pages which only belongs to that process.
- Due to this, there is a chance of a reduction in the thrashing.
- As it is also proved that there are many disadvantages of Local Page replacement.
- Thus local page replacement is simply an alternative to Global Page replacement.

3.16.4. Effects on System Performance and User Experience

- **Slow application response times:** Thrashing can cause applications to take longer to load and respond to user input. This is because the operating system is spending a lot of time swapping pages of memory between physical memory and disk, rather than executing the application's code.
- **Increased system load:** Thrashing can cause the system to become more overloaded. This is because the CPU is spending a lot of time swapping pages of memory between physical

memory and disk, rather than executing code. This can lead to increased CPU utilization, memory usage, and disk activity.

- **System crashes:** In severe cases, thrashing can cause the system to crash. This is because the operating system may not be able to allocate enough memory to all of the running processes, or it may not be able to swap pages of memory between physical memory and disk quickly enough.

Thrashing	Swapping
Here, the CPU spends most of its time swapping pages in and out of the main memory. It results in a decrease in the system's performance.	It is a technique that includes moving an entire process or a part of it from the main memory to the secondary memory and vice versa.
It is caused by overcommitment of the physical memory where the system system tries to manage many processes with only limited RAM.	It is done by the OS to manage the memory efficiently and mainly to free up the space in RAM when not required by the process.
It degrades the system's performance.	It improves the overall performance of the system.
To resolve it, the system can reduce the number of running processes or can increase the physical memory(RAM).	It can be controlled by many memory management techniques to optimize the memory usage.
If not taken care of, it can lead to delays and crashes.	It prevents memory exhaustion leading to efficient memory usage.

3.17. Coalescing and Compaction:

- **Coalescing** is the act of merging two adjacent free blocks of memory.
- When an application frees memory, gaps can fall in the memory segment that the application uses.
- Among other techniques, coalescing is used to reduce external fragmentation, but is not totally effective.

- Coalescing can be done as soon as blocks are freed, or it can be deferred until sometime later (known as deferred coalescing), or it might not be done at all.
- Coalescence and related techniques like heap compaction can be used in garbage collection.
- **Compaction**: It is time consuming procedure, wasteful of processor time.
- It involves shifting a program in memory in such a way that the program does not notice the change.
- This consideration requires that logical addresses be relocated dynamically, at execution time. If addresses are relocated only at load time we cannot support compact storage.

3.18. Page Replacement Algorithms:

Common Page Replacement Techniques

- First In First Out (FIFO)
- Optimal Page replacement
- Least Recently Used (LRU)
- Most Recently Used (MRU)

1. **First In First Out (FIFO)** –
    - This algorithm is similar to the operations of the queue.
    - All the pages are stored in the queue in the order they are allocated frames in the main memory.
    - The one which is allocated first stays in the front of the queue.
    - The one which is allocated the memory first is replaced first. T
    - he one which is at the front of the queue is removed at the time of replacement.

Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1

Pages >>	6	7	8	9	6	7	1	6	7	8	9	1
Frame 3			8	8	8	7	7	7	7	7	9	9
Frame 2		7	7	7	6	6	6	6	6	8	8	8
Frame 1	6	6	6	9	9	9	1	1	1	1	1	1
	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

- As in the above figure shown, Let there are 3 frames in the memory.
- 6, 7, 8 are allocated to the vacant slots as they are not in memory.
- When 9 comes page fault occurs, it replaces 6 which is the oldest in memory or front element of the queue.
- Then 6 comes (Page Fault), it replaces 7 which is the oldest page in memory now.
- Similarly, 7 replaces 8, 1 replaces 9.
- Then 6 comes which is already in memory (Page Hit).
- Then 7 comes (Page Hit).
- Then 8 replaces 6, 9 replaces 7. Then 1 comes (Page Hit).
- Number of Page Faults = 9
- While using the First In First Out algorithm, the number of page faults increases by increasing the number of frames.
- This phenomenon is called Belady's Anomaly.

Let's take the same above order of pages with 4 frames.

Pages >>	6	7	8	9	6	7	1	6	7	8	9	1
Frame 4				9	9	9	9	9	9	8	8	8
Frame 3			8	8	8	8	8	8	7	7	7	7
Frame 2		7	7	7	7	7	7	6	6	6	6	1
Frame 1	6	6	6	6	6	6	1	1	1	1	9	9
	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

- In the above picture shown, it can be seen that the number of page faults is 10.
- There were 9 page faults with 3 frames and 10 page faults with 4frames.
- The number of page faults increased by increasing the number of frames.

2. **Optimal Page Replacement**
    - In this algorithm, the page which would be used after the longest interval is replaced.
    - In other words, the page which is farthest to come in the upcoming sequence is replaced.

Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1, 7, 9, 6

Pages >>	6	7	8	9	6	7	1	6	7	8	9	1	7	9	6
Frame 3			8	9	9	9	1	1	1	1	1	1	1	1	1
Frame 2		7	7	7	7	7	7	7	7	7	7	7	7	7	7
Frame 1	6	6	6	6	6	6	6	6	6	8	9	9	9	9	6
	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Hit	Hit	Miss	Miss	Hit	Hit	Hit	Miss

- First, all the frames are empty. 6, 7, 8 are allocated to the frames (Page Fault).
- Now, 9 comes and replaces 8 as it is the farthest in the upcoming sequence. 6 and 7 would come earlier than that so not replaced.
- Then, 6 comes which is already present (Page Hit).
- Then 7 comes (Page Hit).
- Then 1 replaces 9 similarly (Page Fault).
- Then 6 comes (Page Hit), 7 comes (Page Hit).
- Then 8 replaces 6 (Page Fault) and 9 replaces 8 (Page Fault).
- Then 1, 7, 9 come respectively which are already present in the memory.
- Then 6 replaces 9 (Page Fault), it can also replace 7 and 1 as no other page is present in the upcoming sequence.
- The number of Page Faults = 8

- This is the most optimal algorithm but is impractical because it is impossible to predict the upcoming page references.

3. **Least Recently Used**
    - This algorithm works on previous data.
    - The page which is used the earliest is replaced or which appears the earliest in the sequence is replaced.

Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1, 7, 9, 6

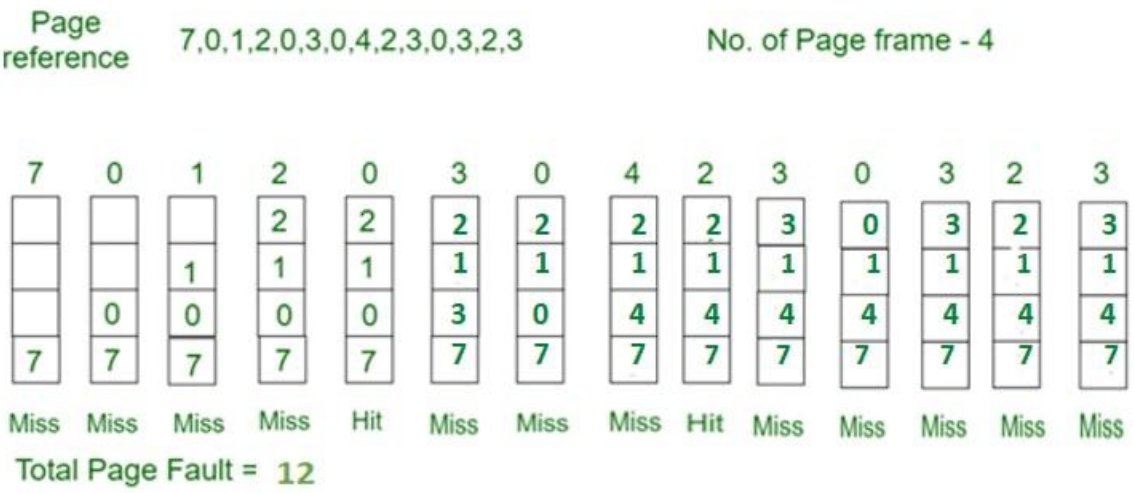
Pages>>	6	7	8	9	6	7	1	6	7	8	9	1	7	9	6
Frame 3			8	8	8	7	7	7	7	7	7	1	1	1	6
Frame 2		7	7	7	6	6	6	6	6	6	9	9	9	9	9
Frame 1	6	6	6	9	9	9	1	1	1	8	8	8	7	7	7
	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Hit	Miss

- First, all the frames are empty. 6, 7, 8 are allocated to the frames (Page Fault).
- Now, 9 comes and replaces 6 which is used the earliest (Page Fault).
- Then, 6 replaces 7, 7 replaces 8, 1 replaces 9 (Page Fault).
- Then 6 comes which is already present (Page Hit).
- Then 7 comes (Page Hit).
- Then 8 replaces 1, 9 replaces 6, 1 replaces 7, and 7 replaces 8 (Page Fault).
- Then 9 comes (Page Hit).
- Then 6 replaces 1 (Page Fault).
- The number of Page Faults = 12

4. **Most Recently Used (MRU)**

- In this algorithm, page will be replaced which has been used recently.
- Belady's anomaly can occur in this algorithm.

Example 4: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using MRU Page Replacement Algorithm.



- Most Recently Used – Page Replacement
- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults
- 0 is already their so–> 0 page fault
- when 3 comes it will take place of 0 because it is most recently used —> 1 Page fault
- when 0 comes it will take place of 3 —> 1 Page fault
- when 4 comes it will take place of 0 —> 1 Page fault
- 2 is already in memory so —> 0 Page fault
- when 3 comes it will take place of 2 —> 1 Page fault
- when 0 comes it will take place of 3 —> 1 Page fault
- when 3 comes it will take place of 0 —> 1 Page fault
- when 2 comes it will take place of 3 —> 1 Page fault
- when 3 comes it will take place of 2 —> 1 Page fault

3.19. Other Page Replacement Algorithms:

1. Second Chance Replacement Algorithm:

- In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will not be replaced.
- The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.
- It can be implemented by adding a "second chance" bit to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process).
- Thus, a page with the "second chance" bit set to 1 is never replaced during the first consideration and will only be replaced if all the other pages deserve a second chance too!

Example –

Let's say the reference string is 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4 and we have 3 frames.

- Let's see how the algorithm proceeds by tracking the second chance bit and the pointer.
- Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the second chance array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.
- Pass-4: Frame={0, 4, 1}, second_chance = {0, 1, 0} [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).
- Pass-5: Frame={2, 4, 1}, second_chance= {0, 1, 0} [0 replaced; it's second chance bit was 0, so it didn't get a second chance], pointer=1 (updated), pf=4
- Pass-6: Frame={2, 4, 1}, second_chance={0, 1, 0}, pointer=1, pf=4 (No change)
- Pass-7: Frame={2, 4, 3}, second_chance= {0, 0, 0} [4 survived but it's second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5
- Pass-8: Frame={2, 4, 3}, second_chance= {0, 1, 0} [4 referenced again], pointer=0, pf=5
- Pass-9: Frame={2, 4, 3}, second_chance= {1, 1, 0} [2 referenced again], pointer=0, pf=5
- Pass-10: Frame={2, 4, 3}, second_chance= {1, 1, 0}, pointer=0, pf=5 (no change)

- Pass-11: Frame={2, 4, 0}, second_chance= {0, 0, 0}, pointer=0, pf=6 (2 and 4 got second chances)
- Pass-12: Frame={2, 4, 0}, second_chance= {0, 1, 0}, pointer=0, pf=6 (4 will again get a second chance)
- Pass-13: Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (pointer updated, pf updated)
- Page-14: Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (No change)
- Page-15: Frame={1, 4, 2}, second_chance= {0, 0, 0}, pointer=0, pf=8 (4 survived again due to 2nd chance!)
- Page-16: Frame={1, 4, 2}, second_chance= {0, 1, 0}, pointer=0, pf=8 (2nd chance updated)
- Page-17: Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (pointer, pf updated)
- Page-18: Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (No change)

Examples –

Input: 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1

3

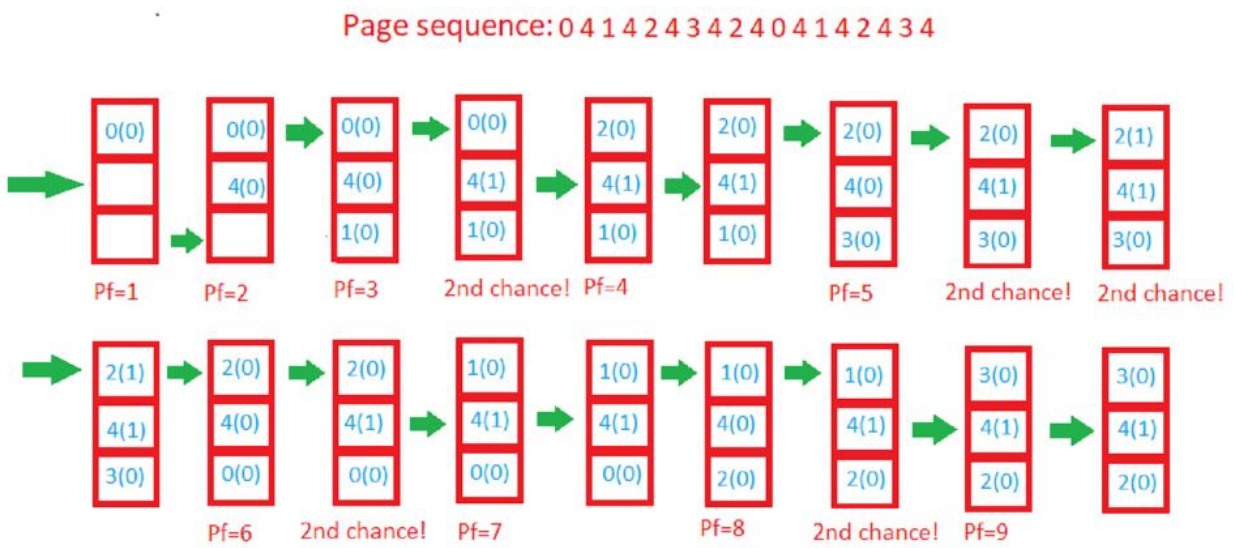
Output: 14

Input: 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1

4

Output: 11

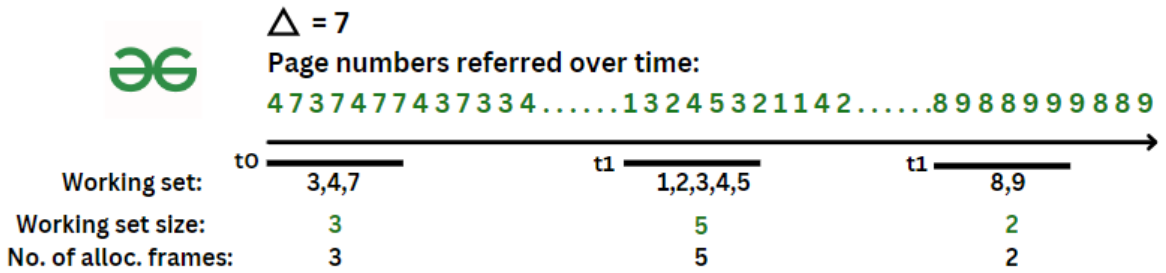
Finally, we report the page fault count.



2. **Working Set Page Replacements /WS Clock Page Replacement**

The Working Set Page Replacement Algorithm:

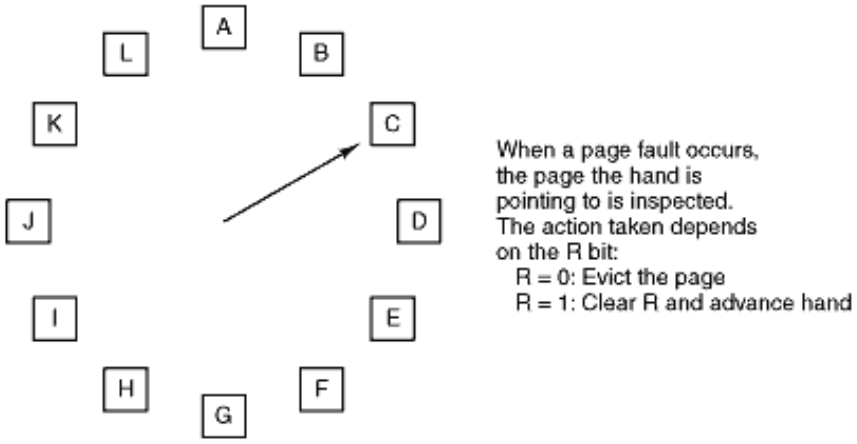
- The working set model finds out the number of unique pages in a locality and maintains those many frames for the process.
- Hence all the required pages can be in the main memory for a certain locality.
- This prevents thrashing and reduces page faults. As the locality changes, the number of frames allocated is also made to change.
- A working set window is maintained of some x no. of pages (x is generally denoted by delta: Δ).
- At every instance, this window examines the past Δ references made by the process and determines the working set.
- The working set is the set of unique pages from these past Δ references.
- The working set the size of a process 'i' is denoted as WSSi and is used to determine the number of frames to allocate to the processi.
- Summation of all WSSi is the total frames required by all processes, denoted by D.
- If the no. of total frames in the main memory is less than D, thrashing is inevitable for some process as it won't have adequate frames.



- Example of a working set model sampling at 3-time instances with delta = 7
- If for a period T, the locality of reference is just 3 pages, then the process can be allocated 3 frames for time T, and there will be no thrashing as all currently required pages will be in the main memory.
- If the locality then increases to 5 pages, 2 more frames can be given to the process.
- If the locality decreases to 2 pages, 3 frames can be released for other processes.
- (Refer to figure) If WSS is greater than available frames, that process is suspended till frames are available.

3. Clock Page Replacement Algorithm:

- The Second Chance Page Replacement algorithm has a drawback that it constantly moves the pages around on its list.
- To solve this, the OS keeps all the page frames on a circular list in the form of a clock.
- The hand of the clock (pointer) points to the oldest page.



- When a page fault takes place, the page being pointed is inspected. If its R bit is 0, the page is evicted and the new page is inserted into its place.
- The pointer moves one position ahead.
- On the other hand, if R is 1, then it's cleared and the hand is advanced to the next page until a page with a false R bit is found.
- Since this works the way a clock works, it is called Clock Page Replacement Algorithm.