MODULE –V
DEADLOCKS, PROTECTION (10)

System Model: Deadlock characterization,
methods of handling deadlocks,
deadlock prevention,
dead lock avoidance,
dead lock detection and recovery from deadlock
system protection,
goals of protection,
principles of protection,
domain of protection,
access matrix,
implementation of access matrix,
access control,
revocation of access rights,
capability-based systems,
language-based protection.

Deadlock

A **deadlock** is a situation in computer systems where a group of processes becomes stuck because each process is waiting for a resource that another process in the group is holding. None of the processes can proceed, resulting in a standstill.

Key Conditions for Deadlock

Deadlock occurs when the following four conditions are simultaneously present:

1. **Mutual Exclusion**: Resources involved are non-shareable (only one process can use a resource at a time).
2. **Hold and Wait**: A process holding one resource is waiting to acquire additional resources held by other processes.
3. **No Preemption**: Resources cannot be forcibly removed from a process; they must be released voluntarily.
4. **Circular Wait**: A set of processes forms a circular chain, where each process is waiting for a resource held by the next process in the chain.

Example

- **Processes**: P1 and P2
- **Resources**: R1 and R2
- Scenario:
    - P1 holds R1 and requests R2.
    - P2 holds R2 and requests R1.

This forms a circular wait, leading to a deadlock.

Deadlock

Deadlock Handling Strategies

There are three common approaches to handle deadlocks:

1. **Deadlock Prevention**:
    1. Ensure that at least one of the four conditions cannot occur. For example:
        1. Avoid circular waits by defining an ordering of resources.
        2. Implement preemption to forcibly release resources.
2. **Deadlock Avoidance**:
    1. Use algorithms like **Banker's Algorithm** to ensure the system is always in a safe state by carefully allocating resources.
3. **Deadlock Detection and Recovery**:
    1. Detect deadlocks using algorithms (e.g., graph-based methods).
    2. Recover by terminating one or more processes or forcibly preempting resources.
4. **Ignoring Deadlock** (Ostrich Algorithm):
    1. Assume that deadlocks are rare and do nothing about them (common in systems like Unix).

EXAMPLE

Examples

•Database Systems: Two transactions waiting for locks held by each other.

•Operating Systems: Multiple threads competing for system resources like memory or file handles.

•Networking: Two nodes waiting for each other to acknowledge packets.

Directory Structure

The directory structure is the organization of files within the file system:

- **Flat Directory**: All files are stored in a single directory with no hierarchy.
- **Hierarchical Directory**: Files are organized into a tree structure of directories and subdirectories.
- **Pathnames**: Describes the location of a file within the directory structure (absolute or relative paths).

Example: On your computer, you have folders like **Documents**, **Pictures**, **Videos**, and **Downloads**. These directories are part of a **hierarchical directory structure**, where each folder may contain subfolders and files, organized in a tree structure.

For example:

- C:\Users\John\Documents\Resume.docx
- C:\Users\John\Pictures\Vacation\photo1.jpg

File System Mounting

Mounting is the process of integrating a file system into the system’s overall file structure:

•**Mount Point**: A directory where the file system is attached.

•**Mounting Process**: When a file system is mounted, the operating system assigns it a location in the directory tree, making it accessible.

Example: If you're using a Linux machine and you plug in a USB drive, the system will "mount" the USB drive to a directory (like /mnt/usb or /media/usb) so you can access its files.

Similarly, in Windows, plugging in a USB drive assigns it a drive letter, such as E:\.

File Sharing

File sharing allows multiple users or processes to access and manipulate files concurrently:

- •**Access Control**: Mechanisms to prevent simultaneous conflicting operations (e.g., locks, semaphores).
- •**File Locking**: Ensures that a file is not accessed simultaneously by conflicting operations.
- •**Concurrency Control**: Deals with preventing data corruption during concurrent access.

Example: **Google Drive** or **Dropbox** allow users to **share files** with others. If multiple people have access to the same shared file (e.g., a Google Docs file), they can collaborate on it in real-time, with the system managing **file locking** to prevent data corruption (concurrent access).

Protection

File protection mechanisms ensure that only authorized users can access or modify files:

•**Permissions**: File access control (read, write, execute) for users and groups.

•**Access Control Lists (ACLs)**: Define who can access a file and what operations they can perform.

•**File Ownership**: Each file has an owner, typically the user who created it, and they control access.

Example: **File Permissions** on Linux or macOS, where you can set read, write, and execute permissions for different users. For instance:

- rw-r--r-- means the owner has read and write permissions, while others only have read access.
- In Windows, file properties let you set permissions such as who can read or modify a file.

File System Structure

The internal organization of the file system includes:

- **Superblock**: Contains metadata about the file system (size, block size, free space information).
- **Inodes**: Store information about files (e.g., size, type, location of data blocks).
- **Data Blocks**: Actual storage locations for the file's content.

Example: The **NTFS file system** in Windows includes the **Master File Table (MFT)** to store metadata about files (e.g., file size, location on disk, access time).

Inodes in **ext4 (Linux)** file systems contain metadata for each file, such as file type, permissions, and pointers to data blocks.

File System Implementation

The actual implementation of a file system involves translating system calls into actions on the storage medium:

•**File Allocation**: How data is stored and mapped onto disk blocks.

•**File Allocation Tables**: Structures that map files to disk blocks, e.g., FAT, inode tables.

Example: When you create a new file (e.g., example.txt) in your **ext4** file system on Linux:

•The system allocates space for the file, updates the **inode** with file metadata (like size, location of data blocks), and creates an **entry** in the directory structure pointing to that inode.

Allocation Methods

There are various methods for allocating disk space to files:

•**Contiguous Allocation**: Files are stored in contiguous blocks on the disk.

- **Advantages**: Simple, fast access.
- **Disadvantages**: Fragmentation, difficult to allocate large files if space is fragmented.

•**Linked Allocation**: Each block of the file contains a pointer to the next block.

- **Advantages**: No fragmentation, easy to grow files.
- **Disadvantages**: Slower access due to pointer traversal, inefficient space utilization.

•**Indexed Allocation**: An index block holds pointers to the file’s data blocks.

- **Advantages**: Direct access to data, efficient space utilization.
- **Disadvantages**: Additional overhead for storing index blocks.

EXAMPLE

•**Contiguous Allocation**: **Video files** often benefit from contiguous allocation because large files (e.g., a 1 GB movie) are stored in a single, contiguous block, allowing fast sequential reads.
•**Linked Allocation**: **Email Attachments**: Each part of a large email attachment may be stored in separate disk blocks that are linked together, allowing easy expansion of the file.
•**Indexed Allocation**: **Database Indexing**: A relational database like MySQL might use an index to efficiently locate records in a table, without having to read the entire table.

Free Space Management

Managing free space on disk is crucial for file system efficiency:
•Bitmaps: Use a bit array where each bit represents whether a block is free (0) or allocated (1).
•Free List: A linked list of free blocks, each pointing to the next available block.

Example: When saving a file (e.g., a photo or video), the system needs to find **free blocks** on the disk to store the data.
•**Bitmaps**: The system might use a bitmap to track free and used blocks. A **free list** might store pointers to available blocks on the disk. Tools like **Disk Cleanup** on Windows help free up disk space by identifying unused blocks.

Directory Implementation

The directory structure can be implemented in several ways:

•**Linear List**: A simple list of filenames.
•**Hash Table**: Provides faster lookups but with more complexity.
•**Tree Structure**: Allows for hierarchical organization, improving the efficiency of search and access.

Example: When you open a folder in **Windows Explorer** or **macOS Finder**, the system retrieves the list of files and directories within that folder. If there are thousands of files, the system might use an **index** or **hash table** to improve the speed of searching for a file.

Efficiency and Performance

Optimizing the performance of file systems involves reducing overhead and improving speed:

- **Caching**: Keeping frequently accessed data in memory to speed up access.
- **Pre-allocation**: Allocating space for files in advance to avoid fragmentation.
- **Defragmentation**: Rearranging fragmented files to reduce the time needed for data retrieval.

Example:

•Caching: When you open a document from Google Drive, the system might cache parts of the file locally on your device to speed up access.

•Defragmentation: If you're using a traditional HDD, you might run Defragment and Optimize Drives on Windows to rearrange fragmented files for faster access.

Mass Storage Structure: Disk Structure

Mass storage typically refers to devices like hard drives, SSDs, and other persistent storage devices:

- **Tracks and Sectors**: The physical layout of storage devices. Tracks are concentric circles on the disk platter, and sectors are smaller units on these tracks.
- **Disk Arm Movement**: The movement of the read/write head across the disk surface.

Example: On a hard disk drive (HDD), data is stored in sectors and tracks. For example, a 2TB HDD might have hundreds of tracks on each platter, and each track is divided into smaller sectors.

- SSD: For solid-state drives (SSD), the concept of tracks and sectors still exists but is abstracted. Data is stored in pages and blocks, and the SSD controller handles wear leveling.

Disk Attachment

Describes how disks are connected to the system:

•**IDE (Integrated Drive Electronics)**: Traditional interface for connecting hard drives.

•**SCSI (Small Computer System Interface)**: High-performance interface.

•**SATA (Serial Advanced Technology Attachment)**: A newer interface that replaced IDE.

Example: In a **data center**, **SCSI** or **SATA** disks are attached to servers to provide large-scale storage. A cloud service like **Amazon AWS** or **Google Cloud** may use similar disk attachment technologies for their storage services, although those disks are often virtualized.

Disk Scheduling

Disk scheduling algorithms are used to decide the order in which disk I/O operations should be performed:

•**FCFS (First-Come-First-Served)**: Processes requests in the order they arrive.

•**SSTF (Shortest Seek Time First)**: Selects the request that is closest to the current head position.

•**SCAN (Elevator)**: Moves the disk arm in one direction satisfying requests until it reaches the end, then reverses.

•**C-SCAN (Circular SCAN)**: Similar to SCAN but instead of reversing, the head moves to the beginning once it reaches the end.

Example: When you issue a command to read data from your hard drive (e.g., opening a file), the disk scheduler decides which request to process first based on algorithms like:

•FCFS: If requests come in one by one without optimization.

•SSTF: If the system prioritizes reading the sector closest to the disk head to minimize seek time.

•SCAN: The system may move the head back and forth across the disk to fulfill requests.

Disk Management

Disk management involves tasks like:

•**Partitioning**: Dividing the disk into separate areas for different file systems or purposes.

•**Formatting**: Preparing the disk for use by creating a file system.

•**Error Detection and Recovery**: Ensuring data integrity by identifying and recovering from disk errors.

Example: **Partitioning** a disk involves dividing it into smaller sections, such as creating a C: drive for Windows and a D: drive for personal data.

•**Formatting** the disk sets up a file system (e.g., **NTFS** for Windows, **ext4** for Linux).

Swap Space Management

Swap space is used when physical memory is full and additional memory is needed:

- **Swap Partition**: A dedicated disk space used for swapping.
- **Swap File**: A file on the disk used as virtual memory.

Example: When you run out of RAM while running multiple programs on your computer, the system starts using **swap space** on the hard drive (or **virtual memory**). For instance, Linux or macOS might use a swap file or a dedicated partition (e.g., swap0) to store data temporarily.

Dynamic Memory Allocation: Basic Concepts

Dynamic memory allocation refers to the runtime allocation and deallocation of memory for programs:

•**Malloc and Free**: Functions in C for allocating and freeing memory.

•**Heap**: The region of memory from which dynamic memory is allocated.

Example: In a **C program**, you might allocate memory dynamically for an array like this:

```
int *arr = (int*)malloc(10 * sizeof(int)); // Allocates memory for 10 integers
```

Later, you can **free** the memory when it's no longer needed:
free(arr).

Library Functions

Commonly used library functions for managing files and memory include:

•**File I/O Functions**: open(), read(), write(), close() in C.

•**Memory Management Functions**: malloc(), calloc(), realloc(), free().

Example: In a **C program**, you might use the following standard library functions for file operations:

•**Opening a file**: fopen("file.txt", "r")

•**Reading from a file**: fread(buffer, sizeof(char), length, file)

•**Writing to a file**: fprintf(file, "Hello, World!")

•**Closing a file**: fclose(file)

Contiguous memory allocation

Advantages of Variable-size Partition Scheme

1. **No Internal Fragmentation:** Space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of **internal fragmentation**. Also, there will be no unused space left in the partition.

2. **Degree of Multiprogramming is Dynamic:** There is no internal fragmentation in this partition scheme due to which there is no **unused space** in the memory. Thus more processes can be loaded into the memory at the same time.

3. **No Limitation on the Size of Process:** The partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the **process size.**

Contiguous memory allocation

Disadvantages of Variable-size Partition Scheme

1. **External Fragmentation: In the** diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in **External Fragmentation**.

2. **Difficult Implementation: T**he allocation of memory at run-time rather than during the system configuration. The OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

Paging

- In Operating System, memory management is the responsibility of dividing the memory among the various processes.
- The fundamental goal of memory management is to make optimal use of memory by minimizing internal and external fragmentation.
- One such algorithm for memory management techniques is **paging.**
- **Paging** is a storage mechanism that allows OS to retrieve processes from the secondary storage into the main memory in the form of pages.
- In the Paging, the main memory is divided into small fixed-size blocks of physical memory, which is called **frames.**
- The size of a frame should be kept the same as that of a page to have maximum utilization of the main memory and to avoid external fragmentation.
- Paging is used for faster access to data, and it is a logical concept.

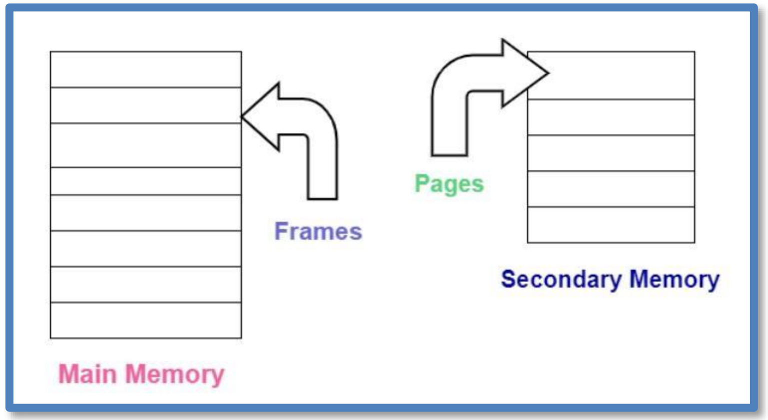
Paging

Why do we need Paging?

- Let there is a process P1 of size **4 MB** and there are two holes of size 2 MB each but are not contiguous.
- Despite having the total available space equal to 4 MB it is useless and can't be allocated to the process.
- Paging helps in allocating memory to a process at different locations in the main memory.
- It reduces memory wastage and removes **external fragmentation**.

Paging

- The Frame has the same size as that of a Page. A frame is basically a place where a (logical) page can be (physically) placed.

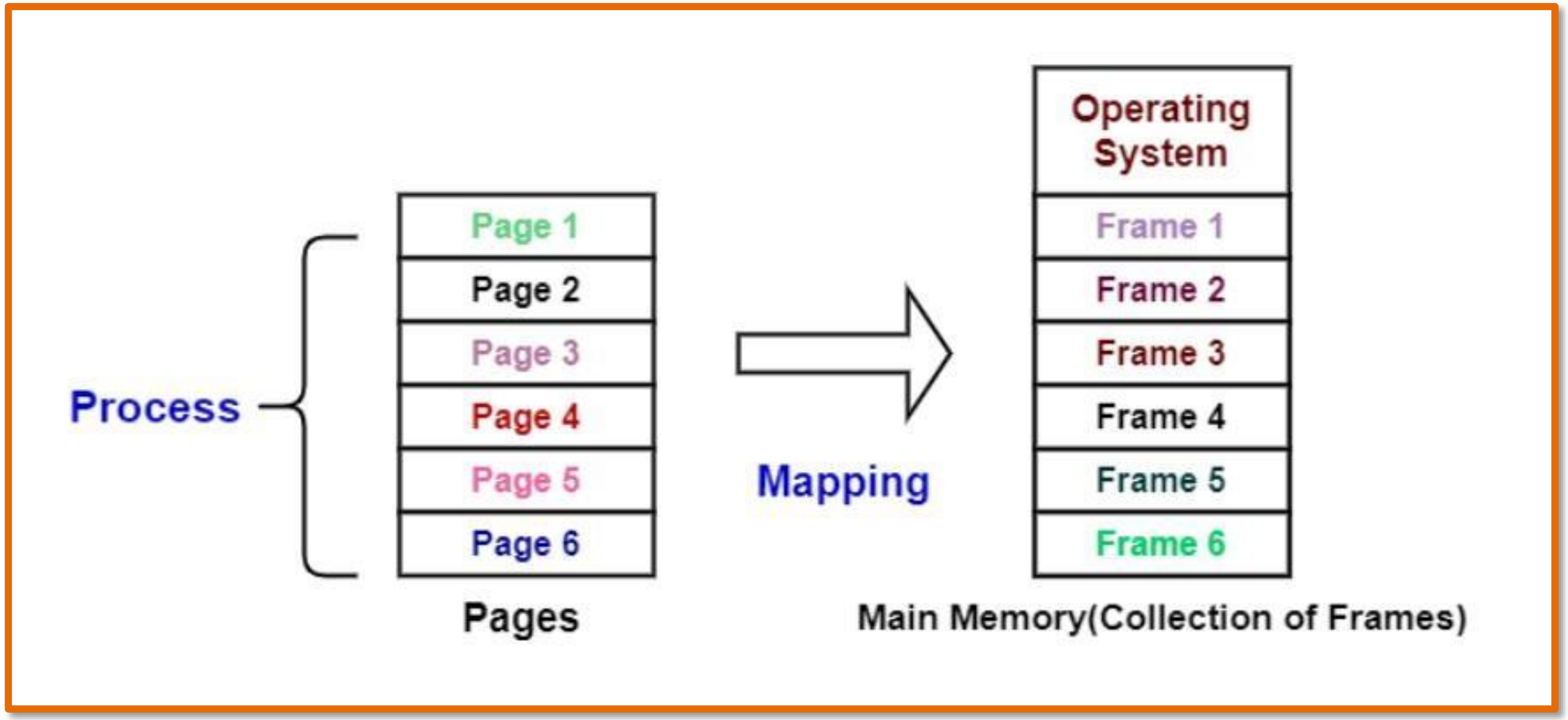


Each process is mainly divided into parts where the size of each part is the same as the page size.

- Pages of a process are brought into the main memory only when there is a requirement otherwise they reside in the secondary storage.

Paging

- One page of a process is mainly stored in one of the frames of the memory. Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames.

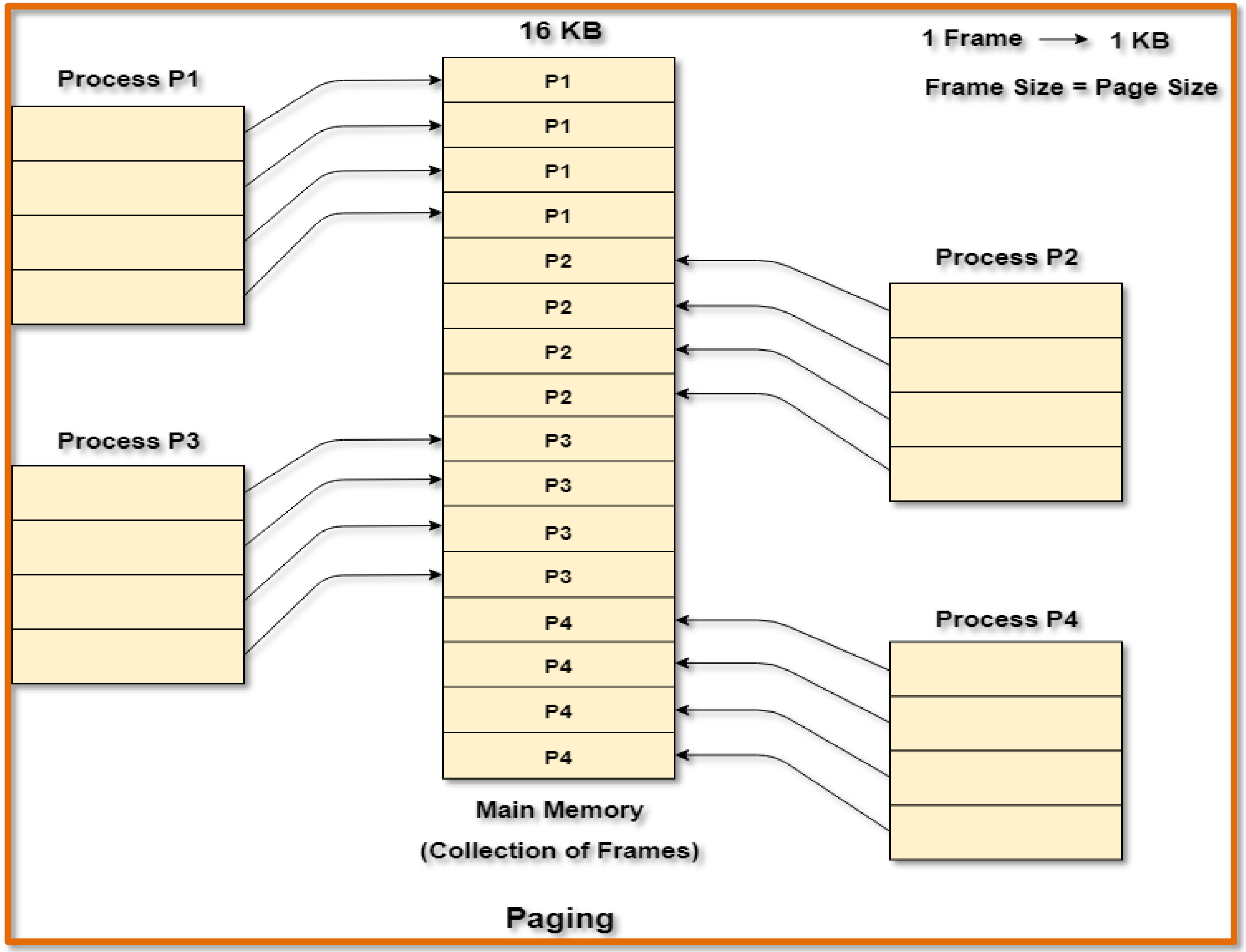


Paging

Example

- Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.
- There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.
- Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

Paging


16 KB
1 Frame → 1 KB
Frame Size = Page Size
Process P1
Process P2
Process P3
Process P4
P1
P1
P1
P1
P2
P2
P2
P2
P3
P3
P3
P3
P4
P4
P4
P4
Main Memory
(Collection of Frames)
Paging

Paging

Conversion of Logical Address into Physical Address

- The CPU always generates a Logical Address. But, the Physical Address is needed to access the main memory.

The Logical Address generated by the CPU has two parts:

- **Page Number(p)** - It is the number of bits required to represent the pages in the Logical Address Space. It is used as an index in a page table that contains the base address of a page in the physical memory.
- **Page Offset(d)** - It denotes the page size or the number of bits required to represent a word on a page. It is combined with the Page Number to get the Physical Address.

The **page number** determines which page of the process the CPU wishes to read data from. The **Page Offset** defines which word on the page the CPU wants to read.

Paging

Conversion of Logical Address into Physical Address

- The CPU always generates a Logical Address. But, the Physical Address is needed to access the main memory.

The Physical Address consists of two parts:

- **Frame Number(f)** - It is the number of bits required to represent a frame in the Physical Address Space. It is the location of the required page inside the Main Memory.

- **Frame Offset(d)** - It is the page size or the number of bits required to represent a word in a frame. It is equal to the Page Offset.

Paging

What is Paging Protection?

The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

Paging

Memory Management Unit

- The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address.
- The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.
- When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

- Page Number
- Offset

Memory management unit of OS needs to convert the page number to the frame number.

Structure of Page table

Page Table

- The Page Table contains the base address of each page inside the Physical Memory.
- It is then combined with Page Offset to get the actual address of the required data in the main memory.
- The Page Number is used as the index of the Page Table which contains the base address which is the Frame Number.
- Page offset is then used to retrieve the required data from the main memory.

Structure of Page table

Hardware implementation of Page Table

- Dedicated registers help in the hardware implementation of a page table. But this is satisfactory until the page table is small.
- For larger page tables we use a translation Look-aside buffer (TLB). It is a small and fast look up hardware cache.

Features of TLB

- It is associative and has high speed memory.
- Each of its entries is divided into two parts, namely, **tag and value.**
- When this memory is used, then an item is compared with all tags simultaneously.
- If the item is found, then corresponding value is returned.

Structure of Page table

The physical address consists of two parts:

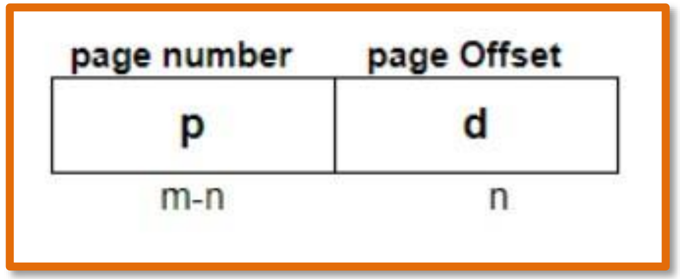
1. Page offset(d)
2. Frame Number(f)

where,

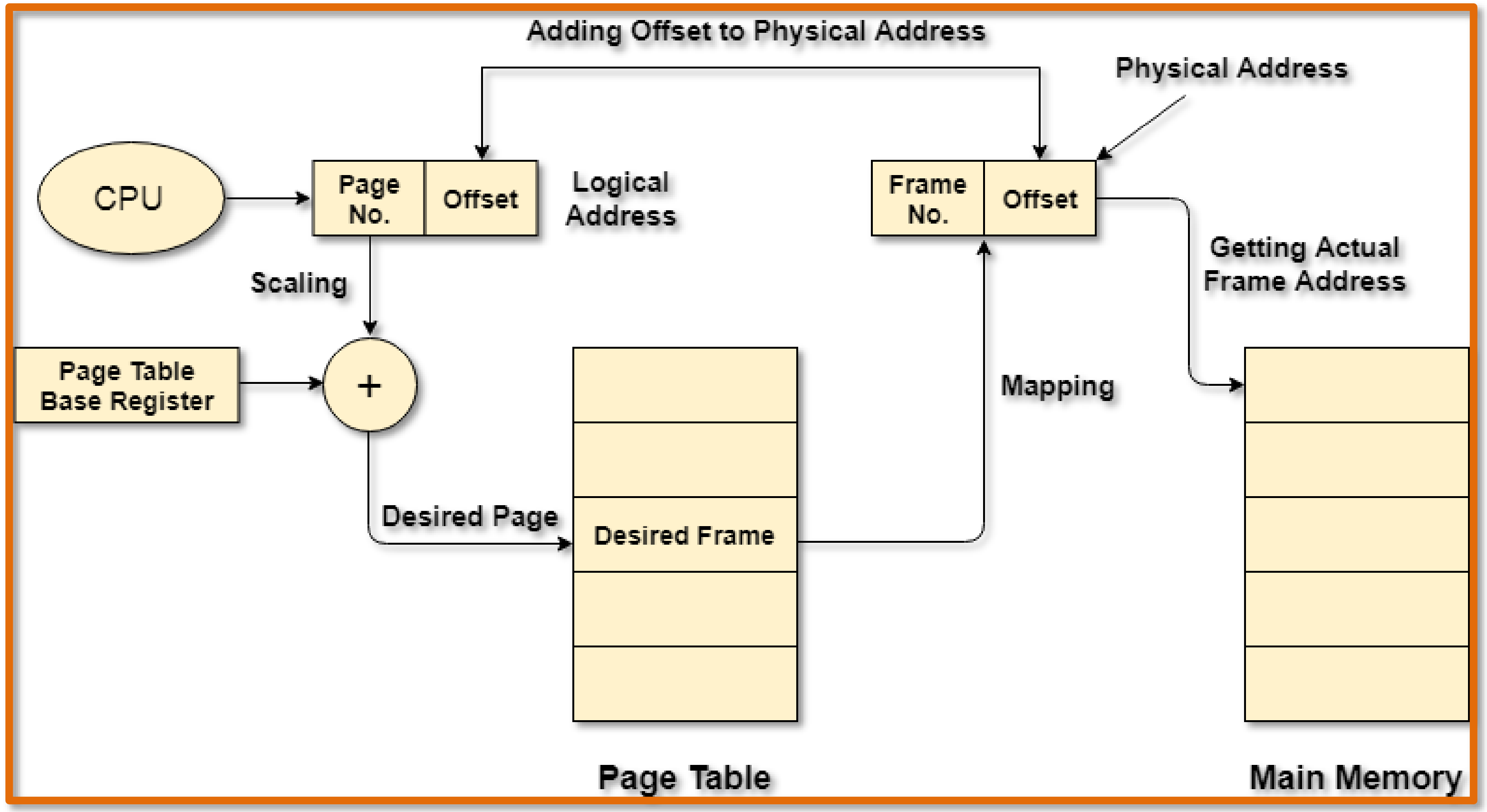
The **Frame number** is used to indicate the specific frame where the required page is stored. **Page Offset** indicates the specific word that has to be read from that page.

The logical address is as follows:

where **p** indicates the index into the page table, and **d** indicates the displacement within the page.



Structure of Page table



Paging

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.

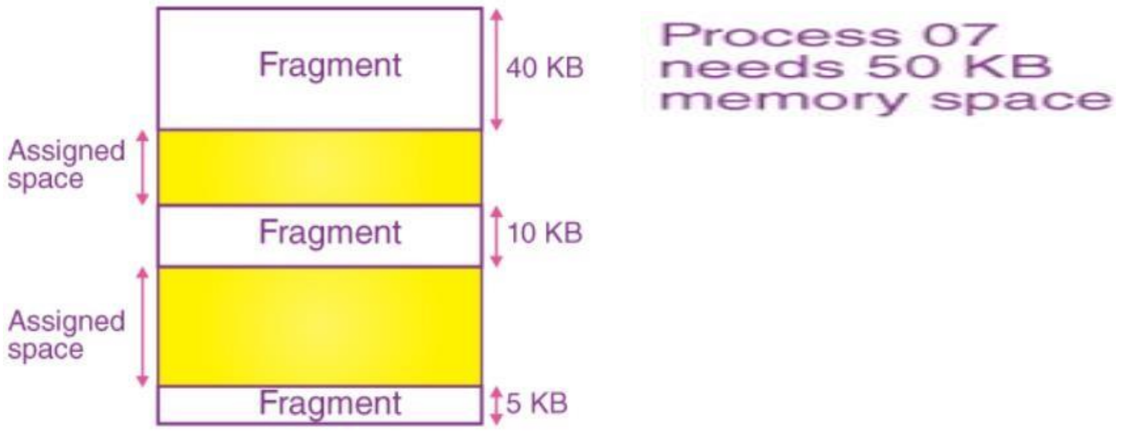
External Fragmentation

The total memory space is sufficient to satisfy a given request or reside any process in it. Yet it isn’t contiguous; thus, it can’t be used. It is what leads to external fragmentation, and it can be reduced using compaction or by shuffling the memory contents in order to place together all the free memory into a single large block. In order to make the compaction feasible, relocation must be dynamic

External fragmentation occurs whenever a method of dynamic memory allocation happens to allocate some memory and leave a small amount of unusable memory. The total quantity of the memory available is reduced substantially in case there’s too much external fragmentation. Now, there’s enough memory space in order to complete a request, and it is not contiguous. Thus, it is known as external fragmentation.

External Fragmentation

External fragmentation occurs whenever the used storage gets differentiated into certain smaller lots and punctuated using assigned memory space. It’s actually a weak point of various storage allocation methodologies when they can’t actually schedule memory utilised by systems. As a consequence, while the unused storage is available to us, it is inaccessible essentially since it’s split separately into fragments that might be too limited to meet the requirements of the software. The word “external” is derived here from the matter that the inaccessible space gets stored outside of the assigned regions.



External Fragmentation

Segmentation

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

There are types of segmentation:

Virtual memory segmentation –

Each process is divided into a number of segments, not all of which are resident at any one point in time.

Simple segmentation –

Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

Segmentation

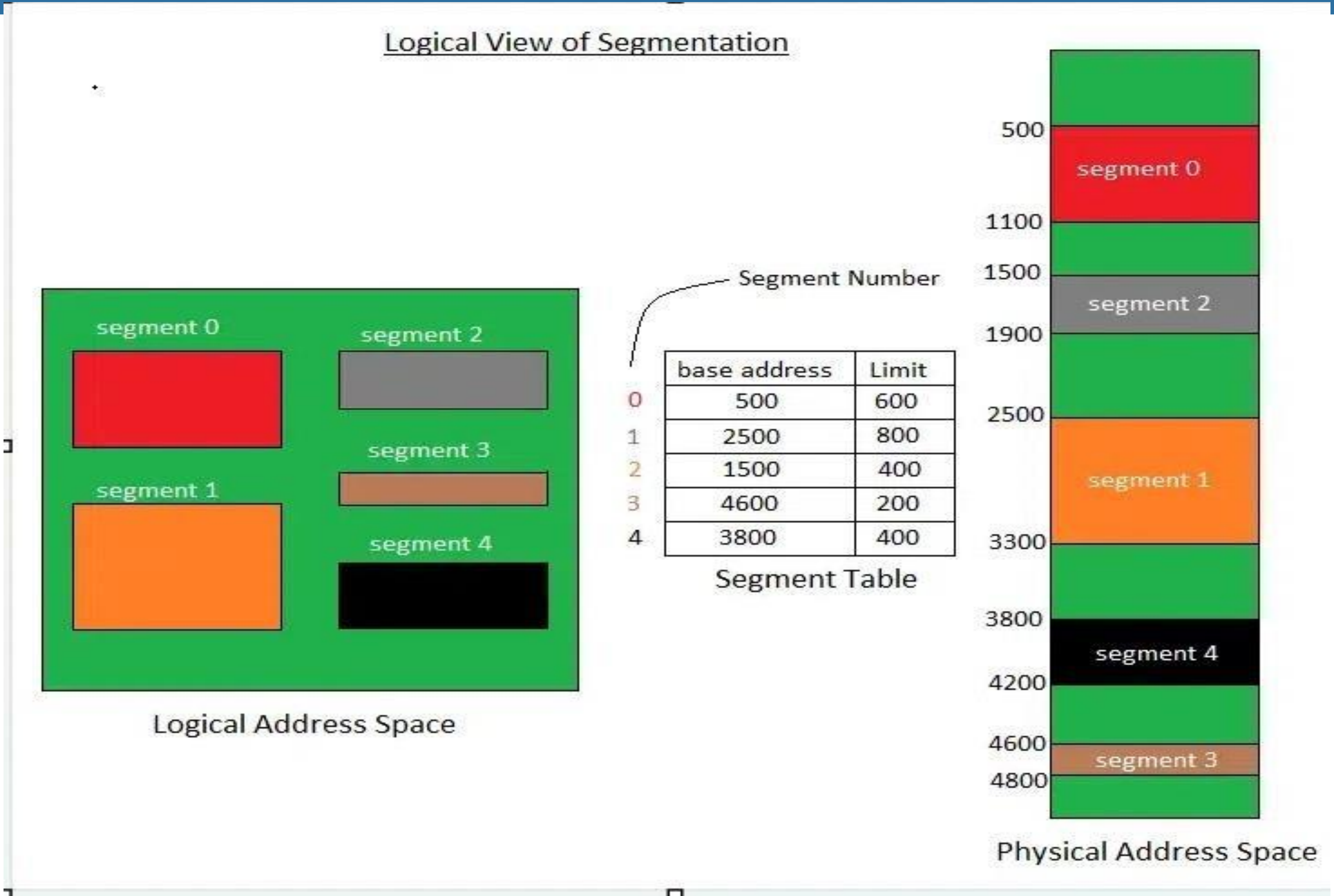
There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called **Segment Table.**

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

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

Limit: It specifies the length of the segment.

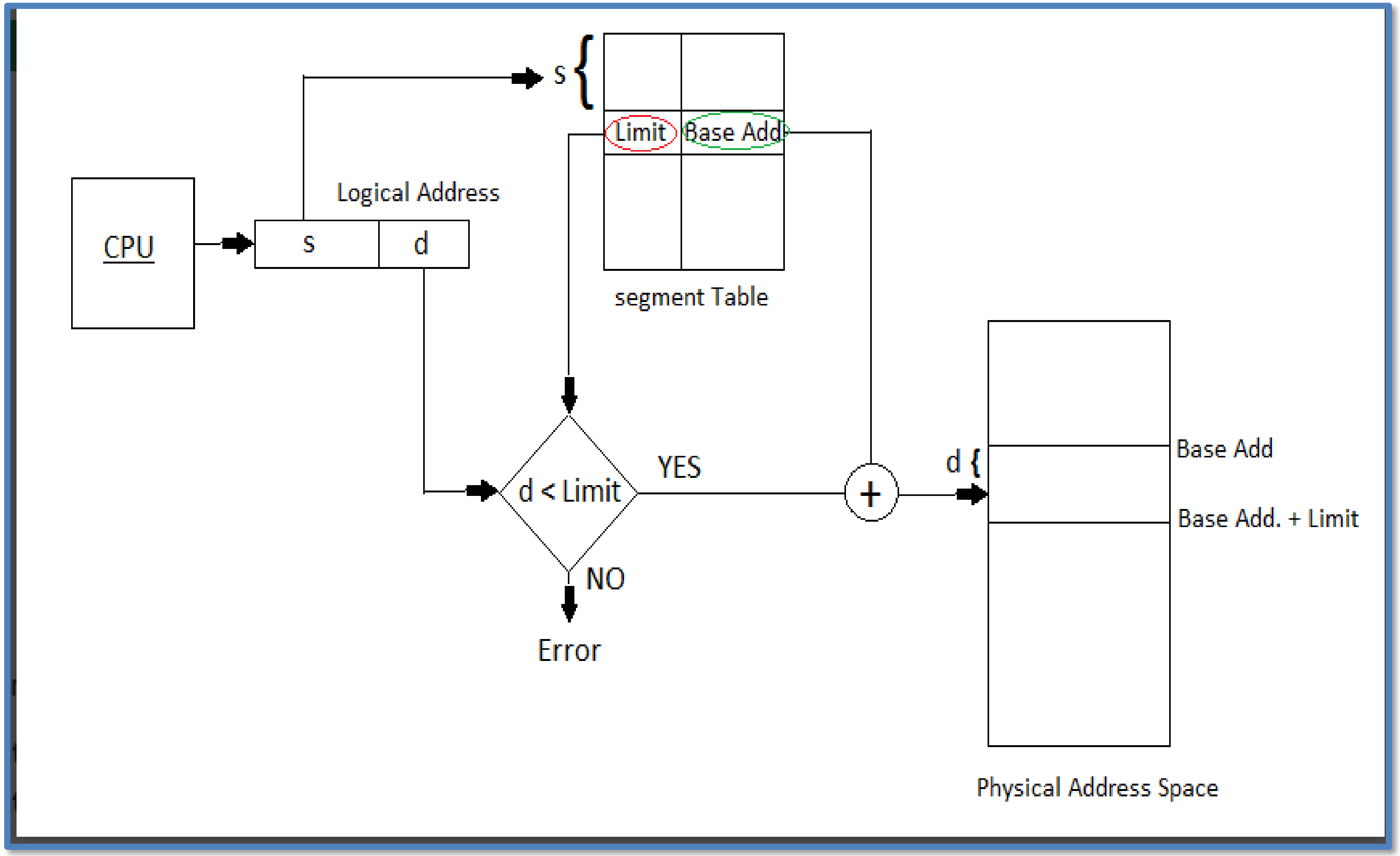
Segmentation



	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segmentation

Translation of Two dimensional Logical Address to one dimensional Physical Address.



Segmentation

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 size of the segment.

Advantages of Segmentation

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation

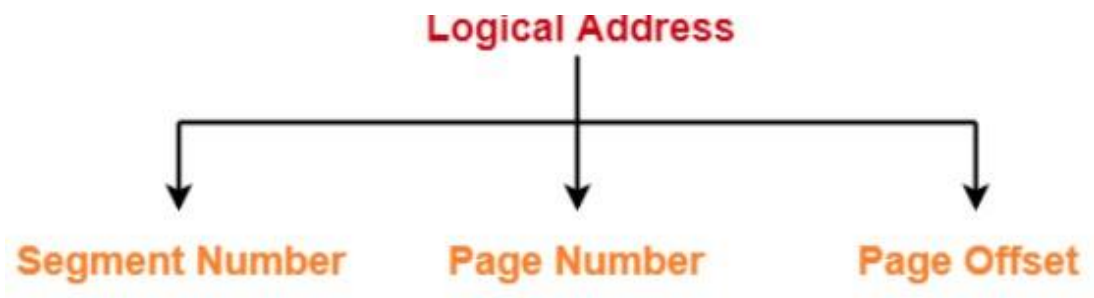
- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

Segmentation with paging

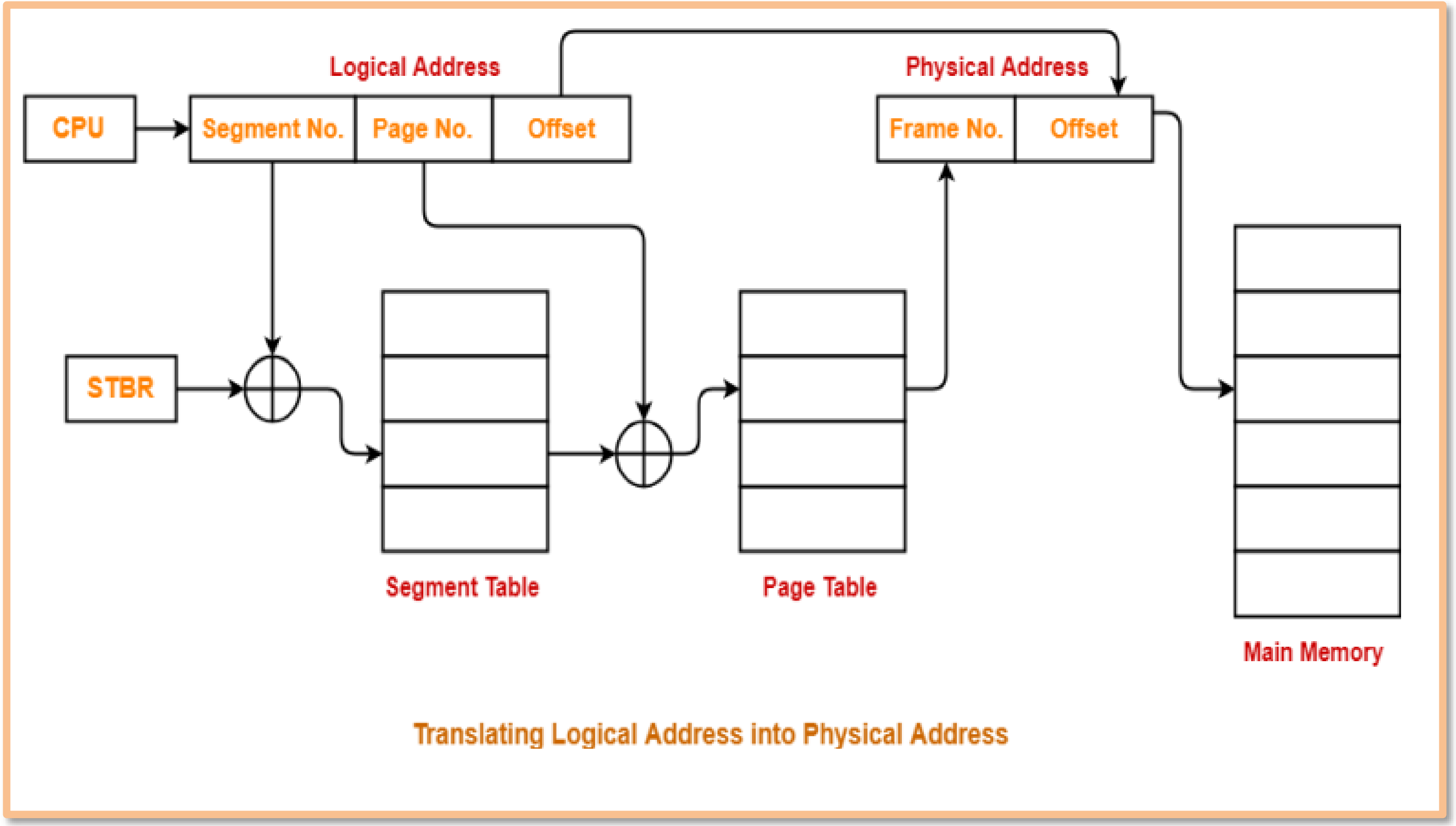
Pure segmentation is not being used in many of the operating systems

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.



Segmentation with paging



Translating Logical Address into Physical Address

Segmentation with paging

The **Segment number** is mapped to the segment table. The limit of the respective segment is compared with the offset.

If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid addresses, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.

Segmentation with paging

Advantages of Segmentation

- No internal fragmentation
- Average Segment Size is larger than the actual page size.
- Less overhead
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as compared to the page table in paging.

Disadvantages

- It can have external fragmentation.
- it is difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithms.

Differences between Paging and Segmentation

S. No	Key	Paging	Segmentation
1	Memory Size	In Paging, a process address space is broken into fixed sized blocks called pages.	In Segmentation, a process address space is broken in varying sized blocks called sections.
2	Accountability	Operating System divides the memory into pages.	Compiler is responsible to calculate the segment size, the virtual address and actual address.
3	Size	Page size is determined by available memory.	Section size is determined by the user.
4	Speed	Paging technique is faster in terms of memory access.	Segmentation is slower than paging.
5	Fragmentatio n	Paging can cause internal fragmentation as some pages may go underutilized.	Segmentation can cause external fragmentation as some memory block may not be used at all.
6	Logical Address	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.
7	Table	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.

Virtual Memory

- Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.
- In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.
- Instead of loading one big process in the main memory, the operating System loads the different parts of more than one process in the main memory.
- The degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.
- Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Virtual Memory

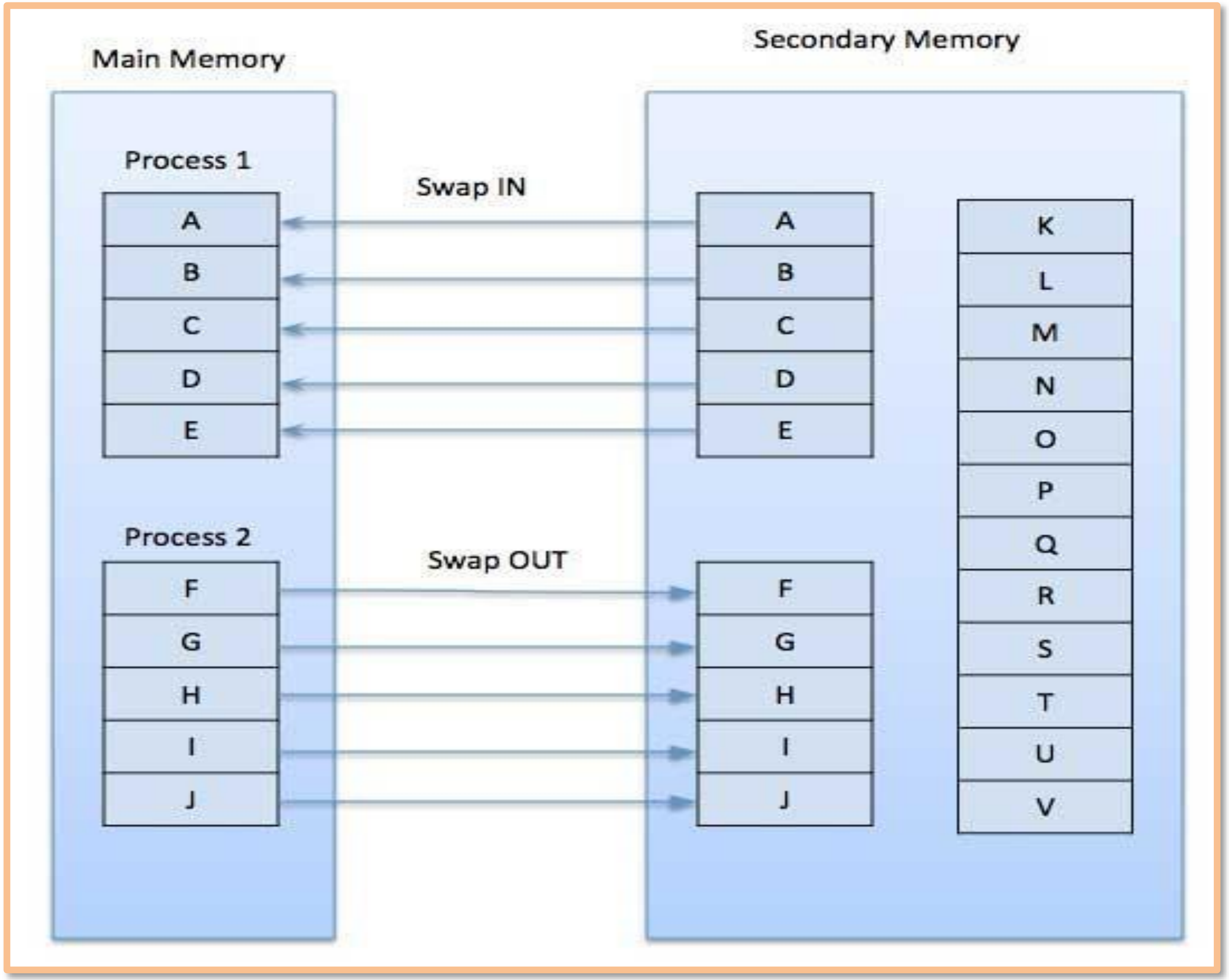
How Virtual Memory Works?

- Virtual memory has become quite common these days.
- Whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.
- Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

Demand Paging

- **Demand paging** is a process of swapping in the Virtual Memory system.
- In this process, all data is not moved from hard drive to main memory because while using this demand paging, when some programs are getting demand then data will be transferred.
- If required data is already existed into memory then not need to copy of data.
- The demand paging system is done with swapping from auxiliary storage to primary memory, so it is known as “Lazy Evaluation”.

Demand Paging



Demand Paging

How Does Demand Paging Working

- Demand paging system is totally depend on the page table implementation because page table helps to maps logical memory to physical memory.
- Bitwise operators are implemented in the page table to indication that pages are ok or not (valid or invalid).
- All valid pages are existed into primary memory, and other side invalid pages are existed into secondary memory.

Demand Paging

Advantages of Demand Paging

- Memory can be utilized with better efficiently.
- We have to right for scaling of virtual memory.
- If, any program is larger to physical memory then It helps to run this program.
- No need of compaction.
- Easy to share all pages
- Partition management is more simply.

Disadvantages of Demand Paging

- It has more probability of internal fragmentation.
- Its memory access time is longer.
- Page Table Length Register (PTLR) has limit for virtual memory
- Page map table is needed additional memory and registers.

Page fault

- If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault.
- The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

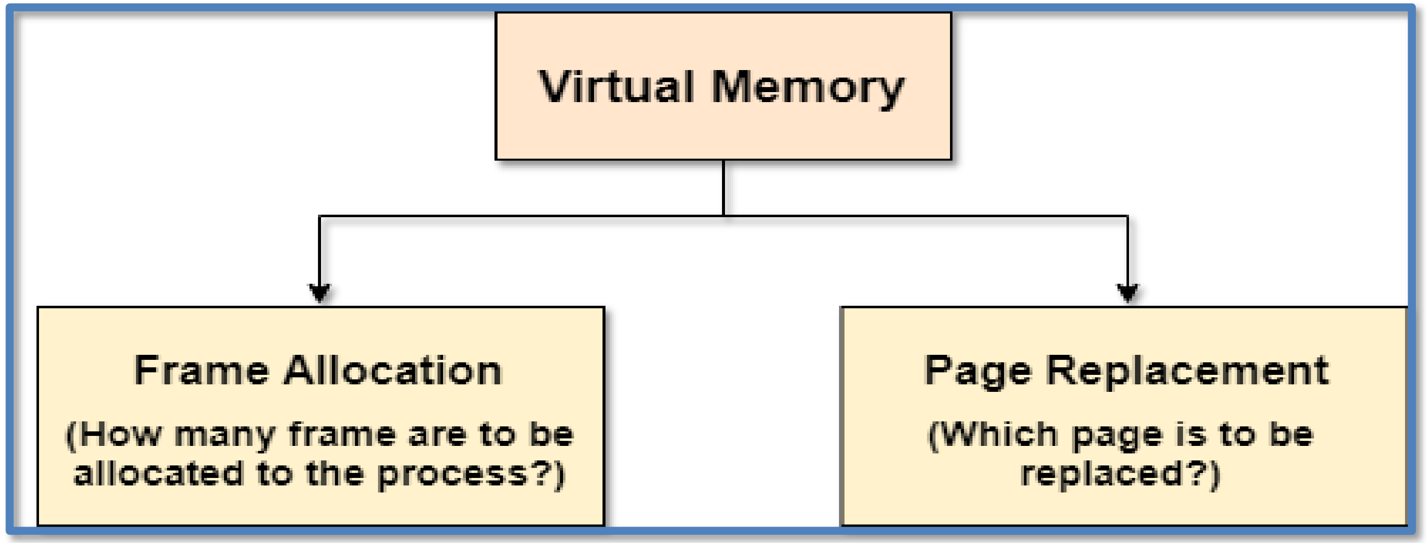
Page replacement

- The page replacement algorithm decides which memory page is to be replaced.
- The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

- The page replacement algorithm decides which memory page is to be replaced.
- The process of replacement is sometimes called swap out or write to disk.
- Page replacement is done when the requested page is not found in the main memory (page fault).

Page replacement

- The page replacement algorithm decides which memory page is to be replaced.
- The process of replacement is sometimes called swap out or write to disk.
- Page replacement is done when the requested page is not found in the main memory (page fault).


Virtual Memory
Frame Allocation
(How many frame are to be allocated to the process?)
Page Replacement
(Which page is to be replaced?)

Page replacement

Why Need Page Replacement Algorithms?

Page Fault: A Page Fault occurs when a program running in CPU tries to access a page that is in the address space of that program, but the requested page is currently not loaded into the main physical memory, the RAM of the system.

Page replacement algorithms help the Operating System in deciding which page to replace.

The primary objective of all the page replacement algorithms is to **minimize the number of page faults.**

Types of Page replacement

1. **Optimal Page Replacement algorithm: T**his algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. **Least recent used (LRU) page replacement algorithm: T**his algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO :** In this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

Page replacement

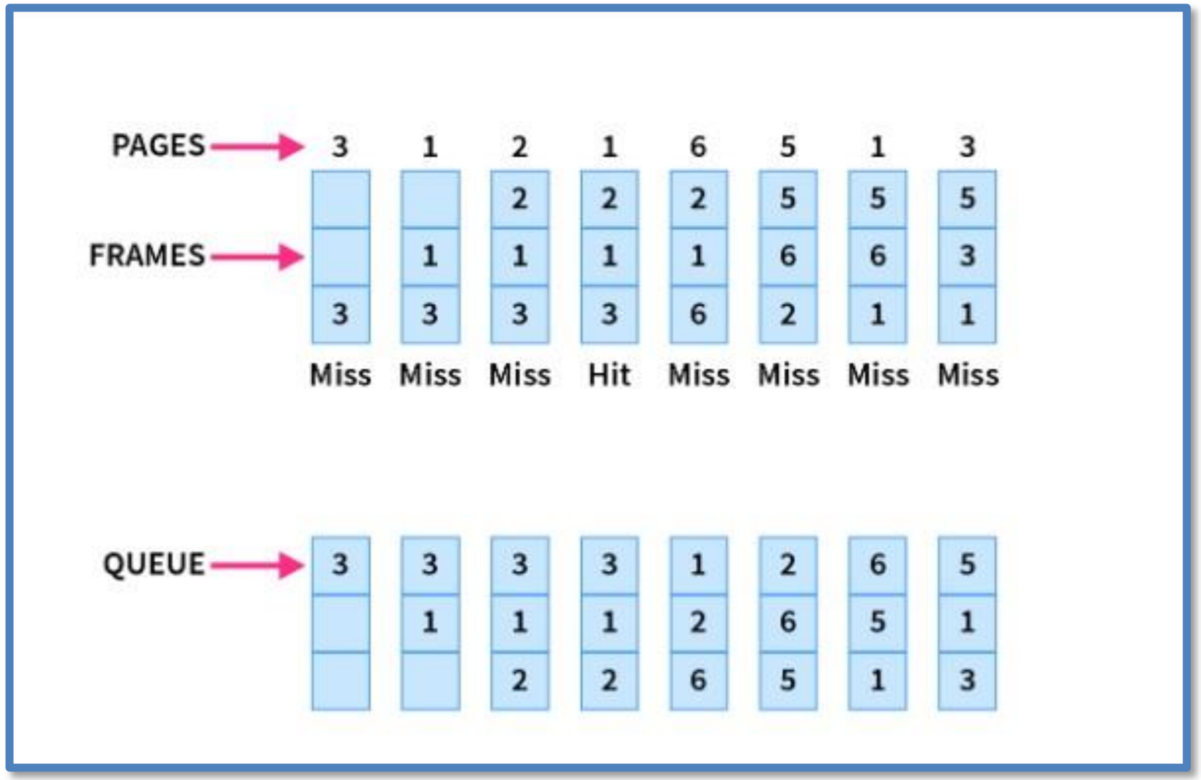
Page Replacement Algorithms in Operating Systems

1. First In First Out (FIFO)

- FIFO algorithm is the simplest of all the page replacement algorithms. In the FIFO algorithm maintain a queue of all the pages that are in the memory currently.
- The oldest page in the memory is at the front-end of the queue and the most recent page is at the back or rear-end of the queue.
- Whenever a page fault occurs, the operating system looks at the **front-end** of the queue to know the page to be replaced by the newly **requested page**.
- It also adds this newly requested page at the **rear-end** and removes the oldest page from the **front-end** of the queue.

Page replacement

- **Example:** Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames. Let's try to find the number of page faults:

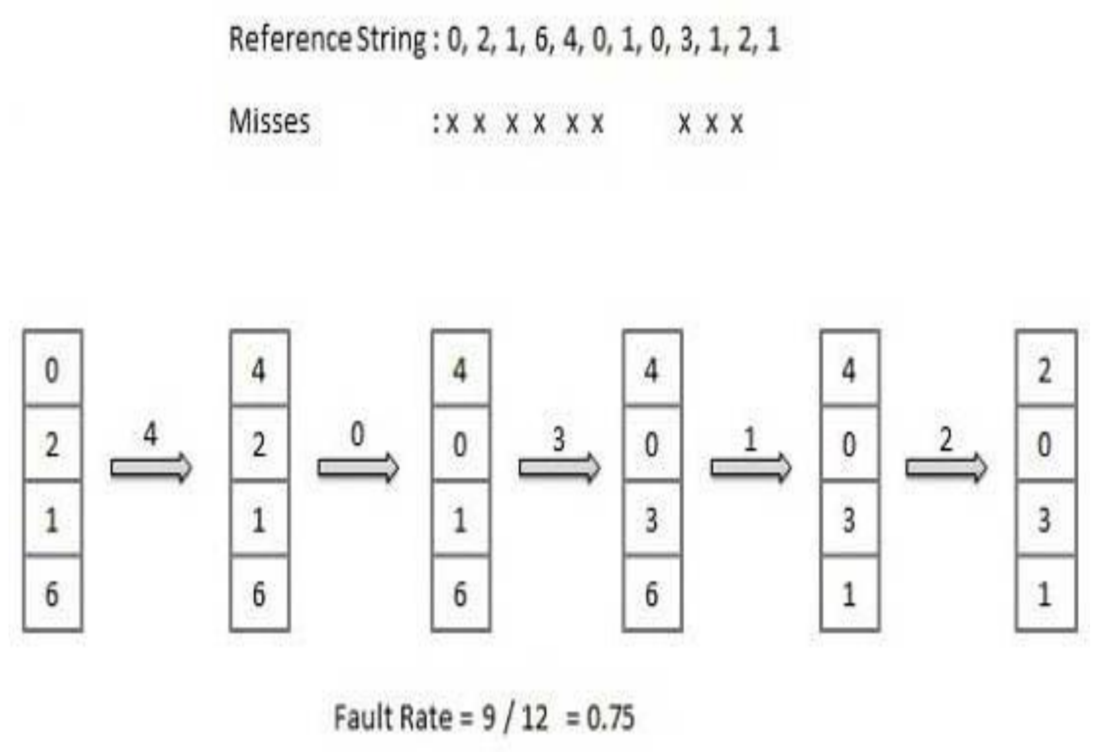


PAGES	3	1	2	1	6	5	1	3
FRAMES			2	2	2	5	5	5
		1	1	1	1	6	6	3
	3	3	3	3	6	2	1	1
	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss
QUEUE	3	3	3	3	1	2	6	5
		1	1	1	2	6	5	1
			2	2	6	5	1	3

Page replacement algorithms

First In First Out (FIFO) algorithm

Oldest page in main memory is the one which will be selected for replacement. Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

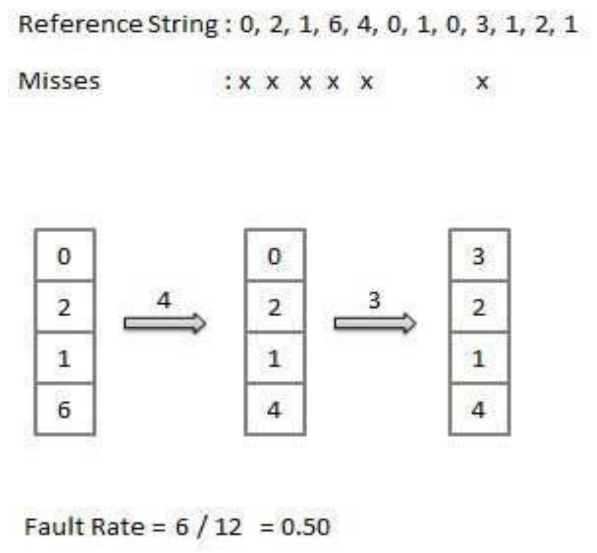


Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses :x x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 3 6
1
4 0 3 1
2
2 0 3 1
Fault Rate = 9 / 12 = 0.75

Page replacement Algorithms

Optimal Page algorithm

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN. Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



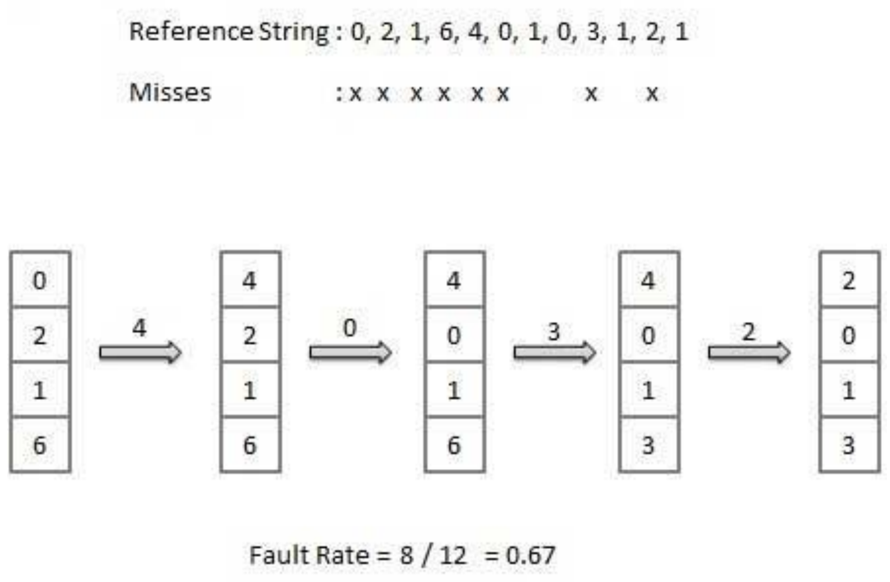
```
Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses           : x  x  x  x  x           x
```

Fault Rate = 6 / 12 = 0.50

Page replacement Algorithms

Least Recently Used (LRU) algorithm

Page which has not been used for the longest time in main memory is the one which will be selected for replacement. Easy to implement, keep a list, replace pages by looking back into time.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x

Fault Rate = 8 / 12 = 0.67

Page replacement Algorithms

Page Buffering algorithm:

To get a process start quickly, keep a pool of free frames. On page fault, select a page to be replaced. Write the new page in the frame of free pool, mark the page table and restart the process.

Least frequently Used(LFU) algorithm:

The page with the smallest count is the one which will be selected for replacement.

Most frequently Used(MFU) algorithm:

This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- The **main memory** of the operating system is divided into various frames. The process is stored in these frames, and once the process is saved as a frame, the CPU may run it.
- As a result, the operating system must set aside enough frames for each process. As a result, the operating system uses various algorithms in order to assign the frame.

There are mainly five ways of frame allocation algorithms in the OS. These are as follows:

1. **Equal Frame Allocation**
2. **Proportional Frame Allocation**
3. **Priority Frame Allocation**
4. **Global Replacement Allocation**
5. **Local Replacement Allocation**

Allocation of Frames

1. Equal Frame Allocation

In equal frame allocation, the processes are assigned equally among the processes in the OS. For example, if the system has 30 frames and 7 processes, each process will get 4 frames. The 2 frames that are not assigned to any system process may be used as a **free-frame** buffer pool in the **system.**

Disadvantage

In a system with processes of varying sizes, assigning equal frames to each process makes little sense. Many allotted empty frames will be wasted if many frames are assigned to a small task.

Allocation of Frames

2. Proportional Frame Allocation

The proportional frame allocation technique assigns frames based on the size needed for execution and the total number of frames in memory.

The allocated frames for a process **pi** of size **si** are **ai = (si/S)*m**, in which **S** represents the total of all process sizes, and **m** represents the number of frames in the system.

Disadvantage

The only drawback of this algorithm is that it doesn't allocate frames based on priority. Priority frame allocation solves this problem.

Allocation of Frames

3. Priority Frame Allocation

Priority frame allocation assigns frames based on the number of frame allocations and the processes. Suppose a process has a high priority and requires more frames that many frames will be allocated to it. Following that, lesser priority processes are allocated.

4. Global Replacement Allocation

When a process requires a page that isn't currently in memory, it may put it in and select a frame from the all frames sets, even if another process is already utilizing that frame. In other words, one process may take a frame from another.

Advantages

Process performance is not hampered, resulting in higher system throughput.

Disadvantages

The process itself may not solely control the page fault ratio of a process. The paging behavior of other processes also influences the number of pages in memory for a process.

Allocation of Frames

5. Local Replacement Allocation

When a process requires a page that isn't already in memory, it can bring it in and assign it a frame from its set of allocated frames.

Advantages

The paging behavior of a specific process has an effect on the pages in memory and the page fault ratio.

Disadvantages

A low priority process may obstruct a high priority process by refusing to share its frames.

Thrashing

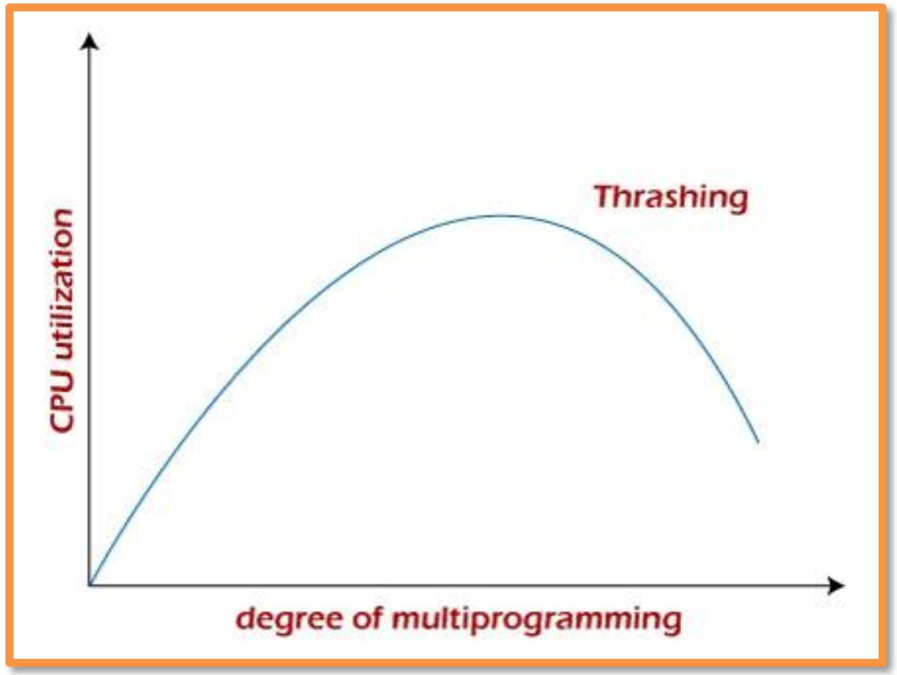
- If the number of page faults is equal to the number of referred pages or the number of page faults are so high so that the CPU remains busy in just reading the pages from the secondary memory then the effective access time will be the time taken by the CPU to read one word from the secondary memory and it will be so high. The concept is called thrashing.

- If the page fault rate is PF %, the time taken in getting a page from the secondary memory and again restarting is S (service time) and the memory access time is ma then the effective access time can be given as;

- $EAT = PF \times S + (1 - PF) \times (ma)$

Thrashing

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.



Thrashing

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.

Algorithms during Thrashing

Whenever thrashing starts, the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

1. Global Page Replacement

Global page replacement can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more. Therefore, the global page replacement algorithm is not suitable when thrashing happens.

Thrashing

2. Local Page Replacement

Local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. Therefore, local page replacement is just an alternative to global page replacement in a thrashing scenario.

Causes of Thrashing in OS

There can be three main causes of thrashing.

- High degree of Multiprogramming.
- Less number of frames compared to the processes requirement.
- The process scheduling scheme which swaps in more processes when CPU utilization is low.