




Concepts of Operating Systems

- Vineela

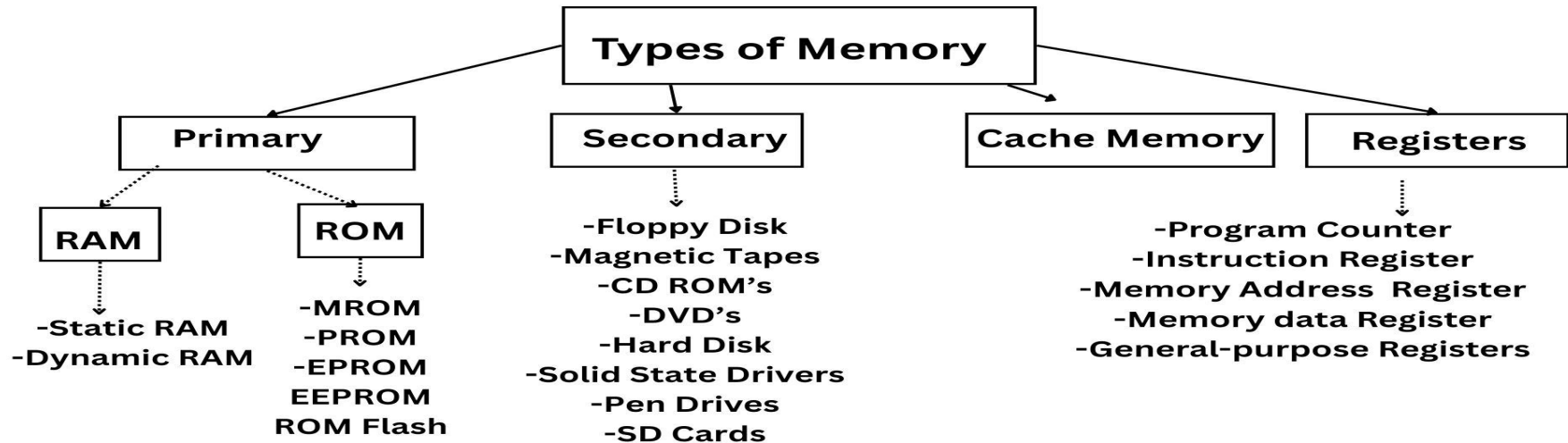
Sessions 6 & 7

Memory Management

- What are different types of memories; What is the need of Memory management
 - Continuous and Dynamic allocation
 - First Fit, Best Fit, worst Fit
 - Compaction
 - Internal and external fragmentation
 - Segmentation – What is segmentation; Hardware requirement for segmentation; segmentation table and its interpretation
 - Paging – What is paging; hardware required for paging; paging table; Translation look aside buffer
 - Concept of dirty bit
 - Shared pages and reentrant code
 - Throttling
 - IO management
- 

Memory Management

- A physical device that stores data or information temporarily or permanently in it is called **Memory** where data is stored and processed.
- The **task of subdividing the memory among different processes** is called Memory Management.
- The main aim is **to achieve efficient utilization of memory**.



RAM (Random Access Memory)

- RAM typically contains 8-bits wherein each memory location, typically are stored.
- It can be possible to read and write to and from a RAM location respectively
- The drawback of RAM is that it is volatile.

From the memory, data can be **accessed in two different ways** – Sequential Access and Random Access

- **Sequential Access** – In this, it is **mandatory to access information strictly in order**.


Ex - If there are 4000 memory locations, it have to be accessed in the order of 1, 2, 3,...,4000.

Thus, it takes minimum time to access information from location 0 and at most time to access information from location 4000.

Ex - Magnetic tape is an example that employs sequential access.

- **Random Access** – In a random access technique, it can be possible to access a memory location in any order.

Ex - One can read from the 4000 locations in the order of 1500, 1210, 3060, 1640, 1352, and so on.



Read Only Memory (ROM)


- It is non-volatile in nature.
- Only reading operation is possible from a ROM location. Thus, in a computer, ROM is used for storing information which is not lost when power is switched off.

The different versions of ROM are:

1- Mask-Programmed ROM

- It derives this name because the information is written to this type of ROM at the time of manufacture by applying a suitable mask
- Once written, the information cannot be changed, even by the manufacturer
- It is **used in equipment produced in large quantities**

2 - Programmable Read Only Memory (PROM)


- The user writes information to this type of ROM using a PROM programmer.
 - Once written, the information cannot be changed.
 - Like Mask-Programmed ROM, the information is permanent.
 - It is more expensive than mask ROM but **allows purchasing in smaller quantities.**
- 

Read Only Memory (ROM)

3- Erasable Programmable Read Only Memory (EPROM)

- As its content is **erasable and rewritable**, the user can modify it multiple times.
- Data is **erased using strong ultraviolet (UV) light** on the quartz window of the EPROM chip, which removes all content.
- Users can purchase a single piece of EPROM and rewrite its content multiple times.

4 - Electrically Erasable Programmable Read Only Memory (EEPROM)

- Unlike EPROM, **EEPROM data is erased using electrical signals** rather than UV light.
 - EEPROM allows selective data erasure and is more expensive than other ROM types.
 - It is gaining popularity due to its flexibility.
- 

Secondary Memory

- Computer secondary memory stores data and programs permanently, even when the computer is off.
- The secondary memory is **also known as external memory or auxiliary memory**.

Classification of Secondary Memory

- Magnetic Storage

- Hard Disk Drive (HDD)
- Floppy Disk (Obsolete)
- Magnetic Tape

- Optical Storage

- CD (Compact Disc)
 - CD-ROM (Read-Only Memory)
 - CD-R (Recordable)
 - CD-RW (Rewritable)
- DVD (Digital Versatile Disc)
 - DVD-ROM
 - DVD-R/DVD+R
 - DVD-RW/DVD+RW
- Blu-ray Disc (BD)

-Flash Storage (Solid-State Storage)

- Solid-State Drive (SSD)
- USB Flash Drive (Pen Drive)
- Memory Cards (SD Card, microSD, etc.)

-Cloud Storage

- Google Drive
- Dropbox
- OneDrive
- Amazon S3

-Hybrid Storage

- Hybrid Drive (HDD + SSD Combination)
- SSHD (Solid-State Hybrid Drive)




Cache Memory

- Cache memory is smaller and faster than RAM. It is placed closer to the CPU than the RAM.

Register Memory

- Register memory, which is also called processor registers or "registers," is the smallest and fastest type of computer memory that is directly integrated into the CPU.

Important functions of Registers:

- **Instruction Execution** – Registers hold the instructions that the CPU is currently running. This includes the operation code (opcode) and associated operands with it.
 - **Data Storage** – Registers store CPU-processed data. This can provide memory addresses, intermediate values during arithmetic or logical operations, and other data needed by the instructions being executed.
 - **Addressing** – Memory addresses are used to store or retrieve data from memory locations in RAM or other parts of the computer's memory hierarchy.
- 

Types of Registers

- Program Counter (PC) – Stores the memory address of the next instruction to be fetched and executed.
- Instruction Register (IR) – Holds the current instruction being executed by the CPU.
- Memory Address Register (MAR) – Stores the memory address of data being read from or written to memory.
- Memory Data Register (MDR) – Contains the actual data being read from or written to memory.
- General-Purpose Registers (GPRs) – Used for general data storage and manipulation during program execution.

Why to manage Memory?

- To improve both utilization of the CPU and the speed of the computer's response to its users



What is Memory Allocation?

- **When a program or process is to be executed, it needs some space in the memory**
- **For this reason, some part of the memory has to be allotted to a process according to its requirements**
- **This process is called Memory Allocation**

Continuous and Dynamic 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 or can be of variable size depending on the requirements of the process

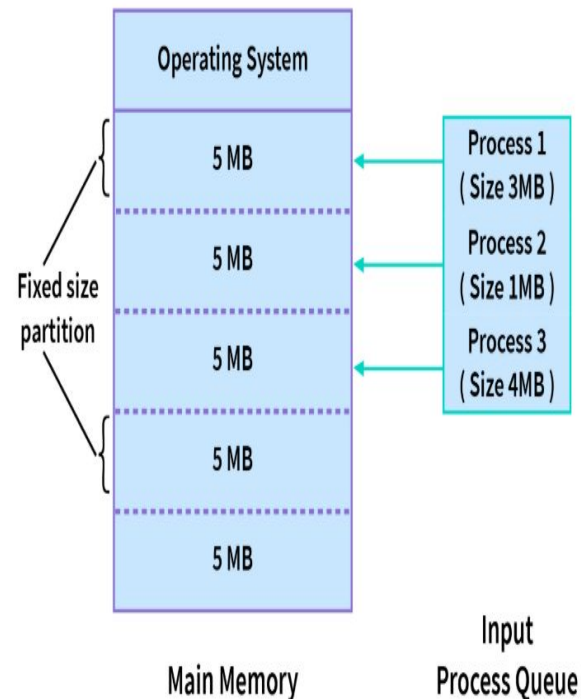
What is Contiguous Memory Allocation in OS?

- As the name implies, we allocate contiguous blocks of memory to each process from the totally empty space based on its size.
- This allocation can be done in two ways:
 1. Fixed-size Partition Scheme
 2. Variable-size Partition Scheme



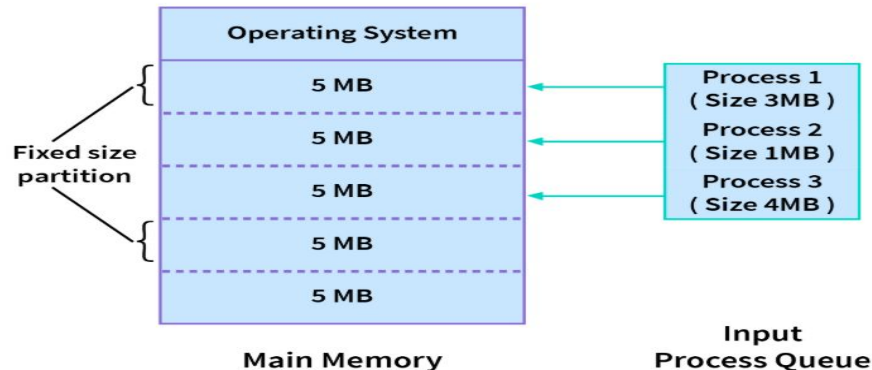
Fixed-size Partition Scheme (Static Partitioning)

- In the diagram above, we have 3 processes in the input queue that have to be allotted space in the memory.
- As we are following the fixed-size partition technique, the memory has fixed-sized blocks.
- First process, which is of size 3MB is also allotted a 5MB block
- Second process, which is of size 1MB, is also allotted a 5MB block, and the 4MB process is also allotted a 5MB block.
- So, the process size doesn't matter. Each is allotted the same fixed-size memory block.
- It is clear that in this scheme, the number of continuous blocks into which the memory will be divided will be decided by the amount of space each block covers, and this, in turn, will dictate how many processes can stay in the main memory at once.



Variable-size Partition Scheme (Dynamic Partitioning)

- In this no fixed blocks or partitions are made in the memory.
- Instead, each process is allotted a variable-sized block depending upon its requirements.
- That means, that whenever a new process wants some space in the memory, if available, this amount of space is allotted to it.
- Hence, the size of each block depends on the size and requirements of the process which occupies it.
- In the diagram below, there are no fixed-size partitions.
- Instead, the first process needs 3MB memory space and hence is allotted that much only. Similarly, the other 3 processes are allotted only that much space that is required by them.
- As the blocks are variable-sized, which is decided as processes arrive.



- So far, we've seen the two types of schemes for contiguous memory allocation.
- But what happens when a new process comes in and has to be allotted a space in the main memory?
- How is it decided which block or segment it will get?

Processes that have been assigned continuous blocks of memory will fill the main memory at any given time. However, when a process completes, it leaves behind **an empty block known as a hole**. This space could also be used for a new process. Hence, the main memory consists of processes and holes, and any one of these holes can be allotted to a new incoming process. We have **three strategies to allot a hole to an incoming process**:

- **First-Fit**
- **Best-Fit**
- **Worst-Fit**

Strategies Used for Contiguous Memory Allocation Input Queues

First-Fit : Allot the process to the first hole, which is big enough.

- This is a **very basic strategy** in which we start from the beginning and **allot the first hole, which is big enough as per the requirements of the process.**
- The first-fit strategy can also be implemented in a way ,where we can **start our search for the first-fit hole from the place we left off last time.**

Best-Fit : Allot the smallest hole that satisfies the requirements of the process.

- This is a **greedy strategy** that aims to **reduce any memory wasted because of internal fragmentation** in the case of static partitioning, and hence we allot that hole to the process, which is the **smallest hole that fits the requirements of the process.**
- Hence, we need to first **sort the holes according to their sizes and pick the best fit** for the process without wasting memory.

Worst-Fit : Allot the largest size hole among all to the incoming process.

- This strategy is the **opposite of the Best-Fit strategy.** We **sort the holes according to their sizes and choose the largest hole to be allotted to the incoming process.**
- The idea behind this allocation is that as the process is allotted a large hole, it will have a lot of space left behind as internal fragmentation.
- Hence, this **will create a hole that will be large enough to accommodate a few other processes.**

Compaction

- Compaction is a technique **to collect all the free memory present in the form of fragments into one large chunk of free memory**, which can be used to run other processes.

Why to go for Compaction?

- While allocating memory to process, the operating system often faces a problem when there's a sufficient amount of free space within the memory to satisfy the memory demand of a process.
- however the process's memory request can't be fulfilled because the free memory available is in a non-contiguous manner, this problem is referred to as **external fragmentation**.
- To solve such kinds of problems compaction technique is used.

How it does?

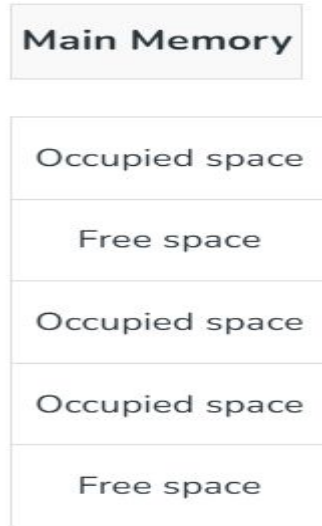
- By moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous.

When to use Compaction?

- It is not always easy to do compaction.
- Compaction can be done **only when the relocation is dynamic and done at execution time**.
- Compaction **can not be done when relocation is static and is performed at load time or assembly time**.

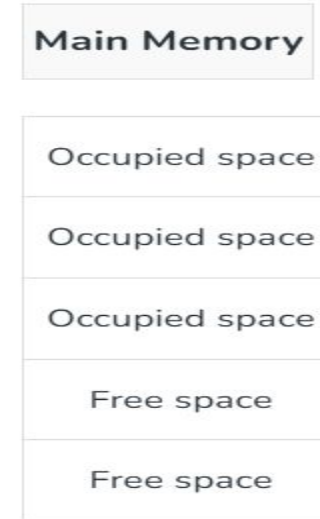
Before Compaction

- Before compaction, the main memory has some free space between occupied space. This condition is known as **external fragmentation**.
- Due to less free space between occupied spaces, large processes cannot be loaded into them.



After Compaction

- After compaction, all the occupied space has been moved up and the free space at the bottom.
- This makes the space contiguous and **removes external fragmentation**. Processes with large memory requirements can be now loaded into the main memory.



Advantages and Disadvantages of Compaction

Advantages

- Reduces external fragmentation.
- Make memory usage efficient.
- Since memory becomes contiguous more processes can be loaded to memory, thereby increasing scalability of OS.
- Fragmentation of file system can be temporarily removed by compaction.
- Improves memory utilization as there is less gap between memory blocks.

Disadvantages

- System efficiency reduces and latency is increased.
- A huge amount of time is wasted in performing compaction.
- CPU sits idle for a long time.
- Not always easy to perform compaction.
- It may cause deadlocks since it disturbs the memory allocation process.

Internal and External Fragmentation

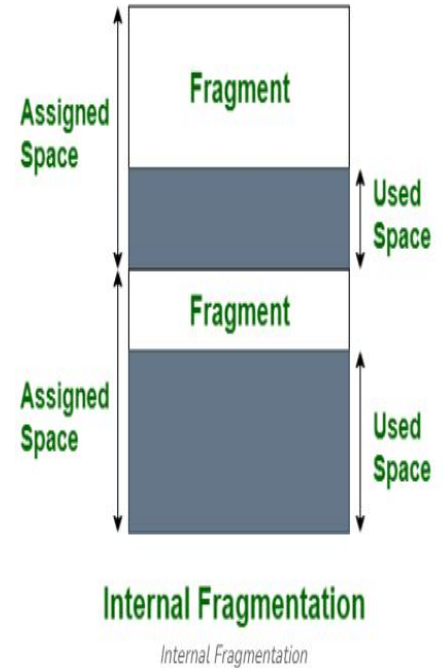
- Memory fragmentation is a prevalent problem in operating systems that can **result in the inefficient use of memory resources**.
- There are two types of fragmentation: internal and external

What is Internal Fragmentation?

- Whenever a method is requested for the memory, the mounted-sized block is allotted to the method.
- In the case, where the memory allocated to the method is somewhat larger than the memory requested, then **the difference between allotted and requested memory is called Internal Fragmentation**.

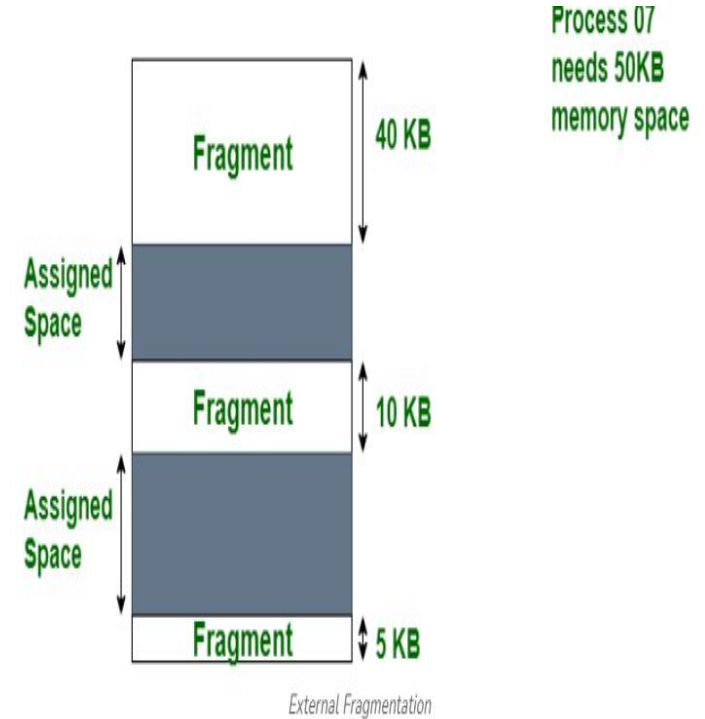
How to solve Internal Fragmentation?

- We fixed the sizes of the memory blocks, which has caused this issue. If we use dynamic partitioning to allot space to the process, this issue can be solved.



What is External Fragmentation?

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



Internal fragmentation	External fragmentation
This happens when the method or process is smaller than the memory	This happens when the method or process is removed
The solution is the best-fit block	The solution is compaction and paging
This occurs when memory is divided into fixed-sized partitions.	This occurs when memory is divided into variable size partitions based on the size of processes.
The difference between memory allocated and required space or memory is called Internal fragmentation	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation
It occurs with paging and fixed partitioning	It occurs with segmentation and dynamic partitioning
It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
It occurs in worst fit memory allocation method	It occurs in best fit and first fit memory allocation method

Segmentation

- The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments
- Each segment has a name and a length
- It is a memory - management scheme that supports the programmer view of memory
- A logical address space is a collection of segments which varies in length


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.

Ex - Programmer (stack, math library, the main program)when writing a simple Sqrt() function is not concerned with

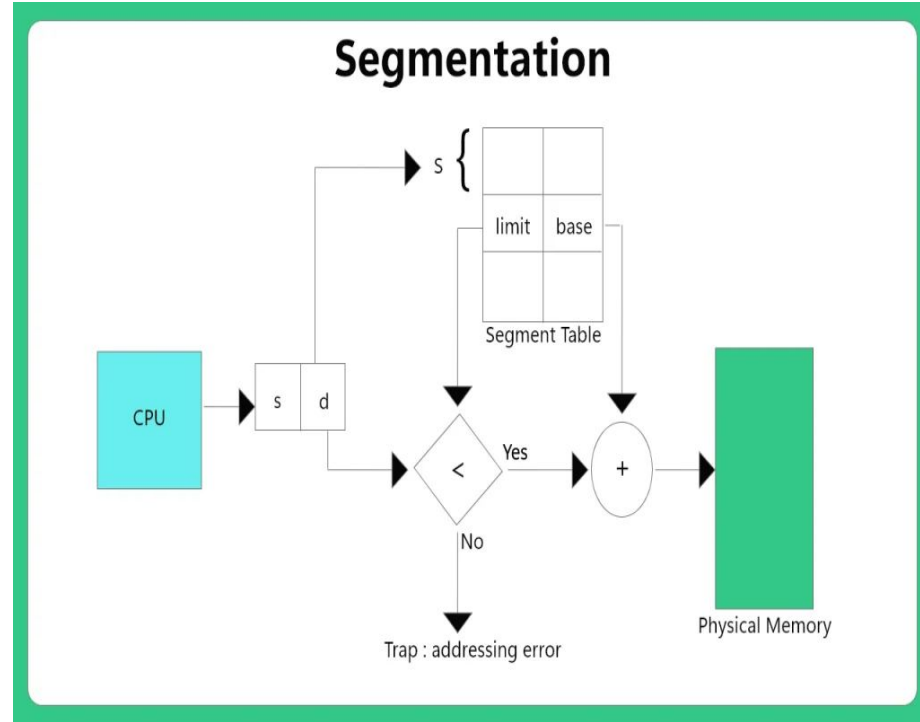
- whether the stack is stored before or after Sqrt()
- not cares what addresses in memory these elements occupy.

Compiler create separate segments for the following:

- 1) The Code
 - 2) Global Variables
 - 3) The heap, from which memory is allocated
 - 4) The stacks used by each thread
 - 5) The standard C library
- 

Segmentation Table and its interpretation

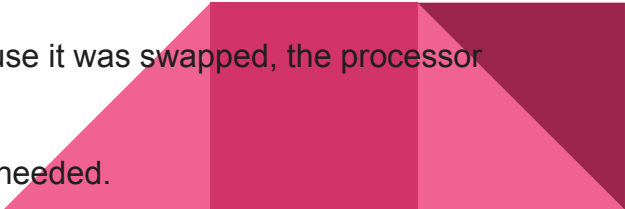
- Although the programmer can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, a one-dimensional sequence of bytes.
- Thus, we define an implementation to map two-dimensional programmer-defined addresses into one-dimensional physical address.
- This mapping is effected by a segment table.
- Each entry in segment table has a segment base (contains starting physical address) and segment limit (specifies the length of the segment).
- A logical address consists of two parts: a segment number s and an offset in to that segment d .
- segment number s - index to the segment table
- Offset d - logical address (between 0 and base limit) - if its not, we trap to OS.
- When an offset is legal, it is added to segment base to produce address in physical memory of the desired byte.



Paging

- Virtual memory is often implemented using **paging**, which breaks memory into fixed-size blocks called **pages** (in the virtual address space) and **page frames** (in the physical memory)
- The OS uses a **page table** to map virtual addresses to physical addresses, allowing data to be stored in non-contiguous areas of physical memory
- When a process accesses a page that is not currently in physical memory, a **page fault** occurs, and the OS loads the required page from secondary storage (usually a hard drive or SSD) into RAM.

Demand paging

- It is a process that **keeps pages of a process that are infrequently used in secondary memory, and pulls them only when required to satisfy the demand.**
 - As a result, when a context switch happens, the OS begins executing the new program after loading the first page and only retrieves the application's referenced pages.
 - If the software addresses a page that is not available in the main memory because it was swapped, the processor deems it as an invalid memory reference.
 - **Ex -** Wishlisting clothes in e-commerce site and decide to buy them only when needed.
- 

How Demand Paging Works:

Step 1 - Initial Setup:

- The program is initially stored on disk.
- Only essential parts (like the initial instructions) are loaded into RAM.

Page Table:

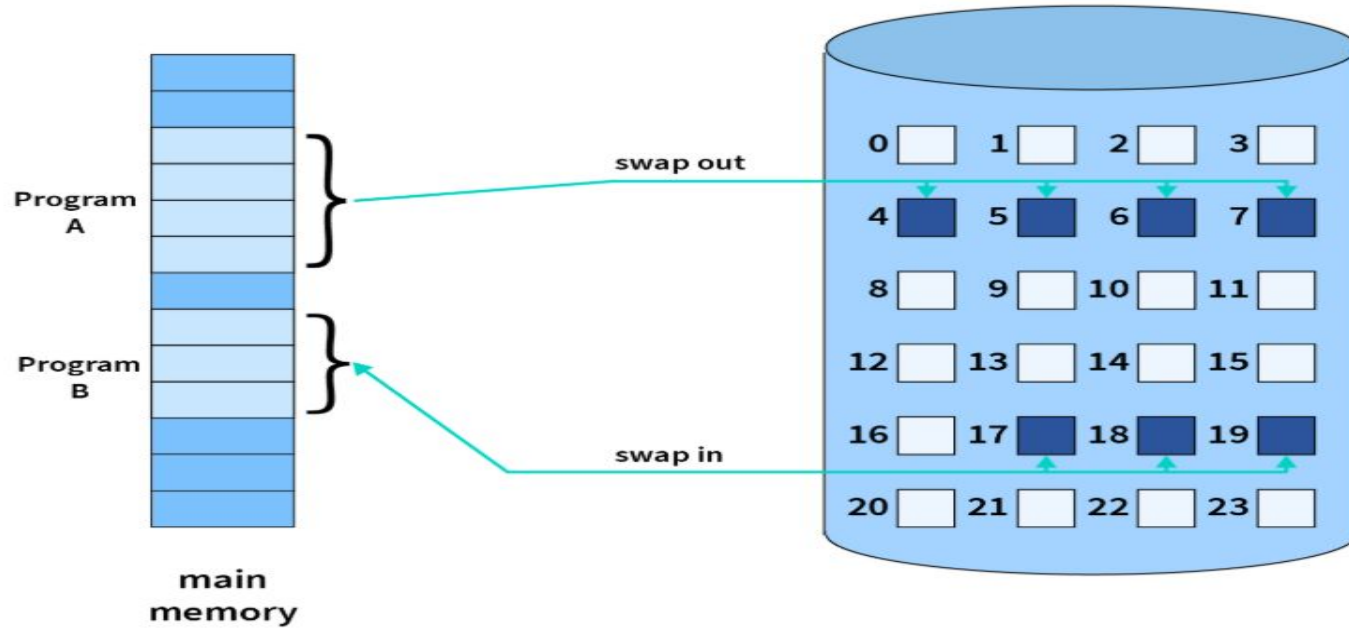
- Each process has a **page table** that keeps track of pages in memory.
- Pages not currently in RAM are marked as **invalid**.

Step 2 - Page Fault:

- If the CPU tries to access a page not in RAM, a **page fault** occurs.
- The OS then: → Pauses the program
→ Loads the required page from disk into memory.
→ Updates the page table Resumes execution.



Example of Demand Paging



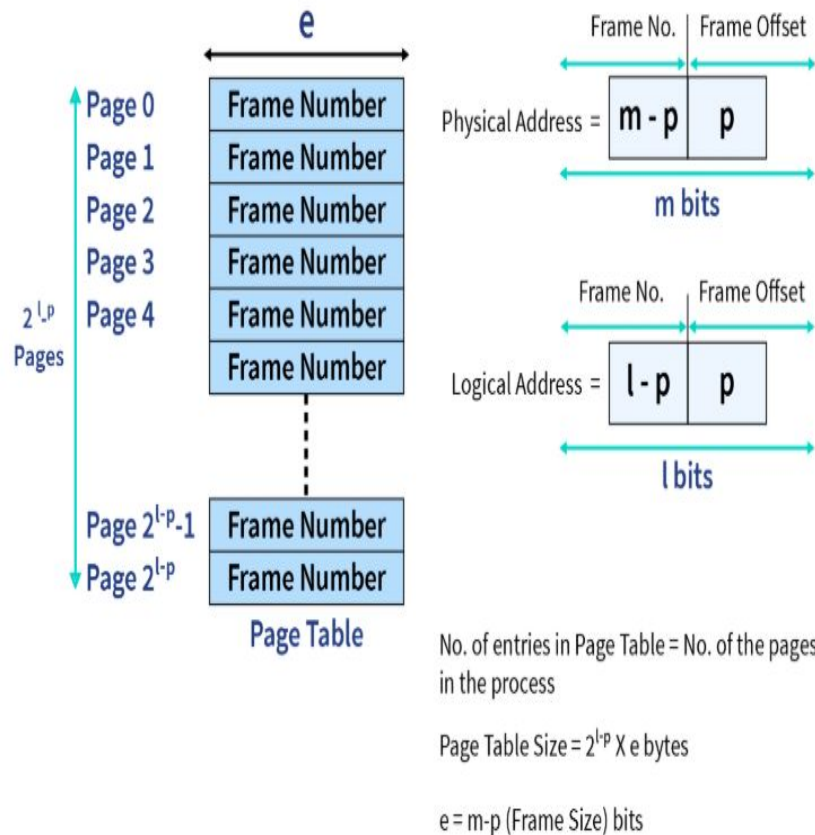
- In the above image, when Program A finishes executing, it swaps out the memory that was in use.
- Program B then swaps in the memory that was required by it to fulfill the timely demand.

Page Frames

- A page frame is used to structure physical memory.
- A page frame size is a power of two bytes and varies between platforms.
- The CPU accesses the processes through their logical addresses while the main memory recognizes the physical address only.
- The Memory Management Unit eases this process by converting the page number (logical address) to the frame number (physical address). The offset is the same in both.

Page Table

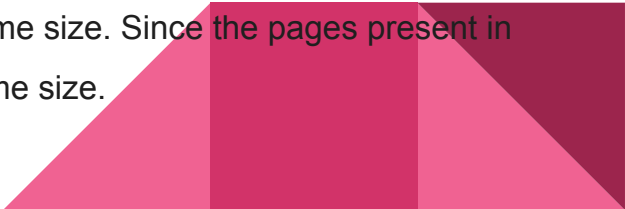
- The page table maps the page number to its frame number.
- A page table is a logical data structure that is used by a virtual memory to record the mapping between virtual and physical addresses.
- The programs performed by the accessing process use virtual addresses, whereas the hardware, notably the random-access memory (RAM) subsystem, uses physical addresses.
- The page table is an important part of virtual address translation, which is required to access data in memory.



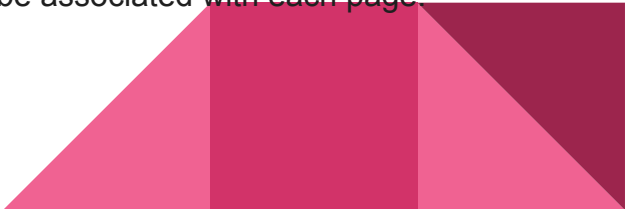
Paging

- The **process of retrieving processes in the form of pages from the secondary storage into the main memory** is known as paging.
- Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory
- The basic purpose of paging is to separate each procedure into pages.
- Paging is a function of memory management where a computer will store and retrieve data from a device's secondary storage to the primary storage.
- The primary concept behind paging is to break each process into individual pages resulting the separation of the primary memory into frames.

Mechanism of Paging

- One page of the process must be saved in one of the given memory frames. These pages can be stored in various memory locations, but finding contiguous frames/holes is always the main goal. Process pages are usually only brought into the main memory when they are needed; else, they are stored in the secondary storage.
 - The frame sizes may vary depending on the OS. Each frame must be of the same size. Since the pages present in paging are mapped on to the frames, the page size should be similar to the frame size.
- 

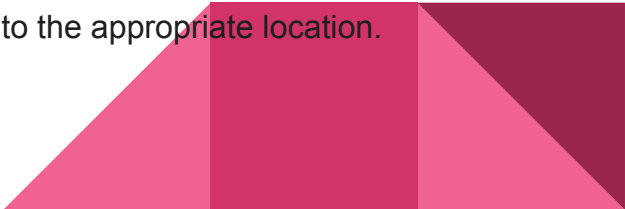
Dirty bit

- A dirty bit, also known as a modified bit or write bit, is a flag that is used in computer systems **to indicate whether a particular memory address or disk block has been modified since it was last written to.**
 - Each page or frame has a modify bit associated with it
 - Modify bit is set whenever page is written into - This indicates page has been modified
 - When page selected for replacement, modify bit is examined.
 - **If the page is dirty** (meaning it has been modified in memory), the OS will write the page back to the disk (or swap space) to ensure that the changes are saved.
 - **If the page is not dirty** (meaning it has not been modified), the OS can safely discard the page without writing it back to the disk because the contents on disk are still valid.
 - In order to reduce the page fault service time, a special bit called the dirty bit can be associated with each page.
- 

How does the dirty bit work?

- When a process modifies a memory address or writes data to a disk block, the dirty bit for that address or block indicates it has been changed.
- This allows the system to keep track of which portions of memory or disk need to be saved or written back to secondary storage when resources become scarce or when a shutdown occurs.

Why is the dirty bit important in caching?

- Caching is a technique used to improve performance by storing frequently accessed data closer to the processor or in a faster storage medium.
 - When data is read from the cache, it is typically marked as clean because it matches the corresponding data in the main memory or disk.
 - However, when the cached data is modified, the dirty bit is set to indicate that the data in the cache has been changed and needs to be written back to the main memory or disk at some point.
 - This ensures that the changes made to the data are not lost and are propagated to the appropriate location.
- 

Shared Pages

- **Shared pages** refer to **memory pages** that are **shared between multiple processes**.
- This is done to **reduce memory usage** and **improve efficiency**, especially when processes use the same code or data.
- Paging system also has a problem with sharing as one of its design issues.
- On a large computer system, that is capable of running multiple programs at once, it is common for multiple users to be occupied with the same program at the same time.
- Now, simply **share the pages in order to prevent having two distinct copies of the same page stored in your memory** at the same time.
- Pages that can only be read are generally shareable, such as the text of a program, however, data pages are not.

What results to Shared Pages?

- When two or more than two processes (referred to collectively as multiple processes) share some code, it can result in a problem with shared pages




Shared Pages

How It Works

1. The OS loads a **shared library** (e.g., libc.so) into memory.
2. Multiple processes **map** the same physical pages into their **virtual address space**.
3. **Page tables** of each process point to the **same physical memory page**.
4. If a process tries to **write** to a read-only shared page:
 - A **page fault** occurs
 - The OS makes a **private copy** of the page (COW)

Example

Suppose two processes, A and B, both use the printf() function:

- printf() comes from libc, loaded once into memory
 - A and B's page tables point to the same physical page for printf()
 - They **share** the code → memory saved
- 

Shared Pages

Example of Shared Pages

- Let's say that process X and process Y are both running the editor and sharing its pages. What would happen?
- If the scheduler makes the decision to remove process X from memory, evicting all of its pages and filling the empty page frames with the other program will cause process Y to generate a large number of page faults in order to restore them.
- If the scheduler makes the decision to remove process Y from memory, evicting all of its pages and filling the empty page frames with the other program.
- In a similar fashion, whenever the process X comes to an end, it is essential to be able to discover that the pages are still in use. This ensures that the disc space associated with those pages is not accidentally freed.



Reentrant code (Pure Code)

- Reentrant code is code that can be safely interrupted in the middle of execution and called again ("re-entered") before the previous executions are finished **without affecting the outcome**.
- Reentrant functions or routines are designed in such a way that they maintain their integrity and can be called concurrently without causing problems like data corruption or unexpected behavior.

Key Characteristics of Reentrant Code:

1. **No static or global state:** Reentrant code does not rely on shared or static variables, as they can be overwritten when re-entered. Any state information should be local to the function.
2. **No side effects:** A reentrant function should not modify any external state or depend on it.
3. **Atomic operations:** The function must execute in a way that ensures it can be interrupted and resumed without interfering with other invocations.
4. **Thread safety:** Reentrant code is often thread-safe, though thread safety and reentrancy are not exactly the same. Thread safety ensures that multiple threads can use the code concurrently without conflict, while reentrancy ensures that the same function can be re-entered at any point.

Finally Reentrant code is Independent, Interrupt-safe , Shareable and Essential for system-level programming



Example of Reentrant code

Non-Reentrant Code (has global variable):

```
int counter = 0;

void increment() {
    counter++;
}
```

This is **not reentrant** because:

- Uses global variable counter
- If interrupted, another execution may modify counter, leading to incorrect results

Reentrant Code (uses local variable):

```
int increment(int x) {
    return x + 1;
}
```

This is **reentrant** because:

- No shared/global data
- Can be safely called by multiple threads or during interrupts



Throttling

- It is a technique **used to manage system resources**, such as CPU time, memory and network bandwidth, to ensure that processes and applications do not overwhelm the system.
- By controlling the rate at which certain tasks or processes are executed, the operating system can maintain performance, fairness, and responsiveness, especially in multi-tasking environments, where many processes are running concurrently.

- **Types of Throttling in an OS:**

CPU Throttling (Process Scheduling):

- This involves limiting the CPU resources allocated to a process to prevent one process from consuming all the available CPU time.
- This is often done by **process scheduling algorithms** and is especially important in systems with multiple processes or threads.
- **Example:** In an OS with multiple processes, the kernel uses scheduling algorithms like Round-Robin or Priority Scheduling to allocate CPU time to each process. If a process consumes too much CPU time, the OS might throttle it by reducing its priority or placing it in a waiting queue.

Throttling

I/O Throttling:

- Throttling can be applied to disk and network I/O operations to prevent a single process from monopolizing disk access or network bandwidth.

Example: The OS might limit the read/write rate for a specific process or user to ensure fair disk access and prevent the system from slowing down due to excessive disk activity.

Network Throttling:


- Network throttling involves limiting the data transfer rate over a network interface to ensure that no single process or user consumes all the bandwidth.

Example: A file download process might be throttled to prevent it from using all available network bandwidth, which could slow down other applications, such as web browsing or online communication.

Memory Throttling (Swap Throttling):

- In systems with limited physical memory (RAM), when the system runs out of available memory, the operating system may "throttle" processes by swapping memory pages in and out of disk storage (swap space) to free up space in RAM.

Example: If the system is running out of memory, the OS might use techniques like **Out-of-Memory (OOM) Killer** in Linux to throttle or terminate processes that are consuming too much memory.



IO Management


- **I/O Management** refers to the processes and mechanisms that an operating system (OS) uses to manage input and output (I/O) operations, such as communication between the system and external devices like keyboards, mice, displays, disk drives, printers, and network interfaces.
- I/O management ensures that the system can handle these operations efficiently, providing fair access to I/O devices, managing data transfer and maintaining the stability and performance of the system.

Key Components of I/O Management

I/O Devices:

- I/O devices can be categorized into **input devices** (keyboard, mouse) and **output devices** (monitor, printer).
- **Storage devices** like hard drives, SSDs, and network devices are also integral parts of I/O management.

Device Drivers:

- A **device driver** is a software component that acts as a bridge between the OS and hardware devices.
 - It translates the generic I/O commands from the OS into device-specific commands.
 - Drivers abstract the hardware complexities and allow the OS to communicate with various devices uniformly.
- 

IO Management

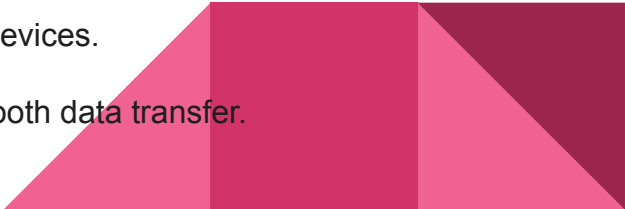
I/O Control:

- I/O control refers to the software and mechanisms that govern how input and output operations are initiated, managed, and completed. This includes interactions between the OS, device drivers, and hardware.
- I/O control ensures that the OS can access and control multiple I/O devices simultaneously without interference

I/O Scheduling:

- I/O scheduling is the process of determining the order in which I/O requests are serviced by the OS. Efficient I/O scheduling is crucial for optimizing system performance and reducing latency.
- Algorithms like **First-Come, First-Served (FCFS)**, **Shortest Seek Time First (SSTF)**, and **Look/Seek Algorithms** are used to manage I/O requests to storage devices

Buffering:

- Buffering is used to store data temporarily while it is being transferred between devices or between devices and memory.
 - Buffers help reduce the time difference between fast processors and slower I/O devices.
 - **Double buffering** and **circular buffering** are common techniques to ensure smooth data transfer.
- 

Session 8:

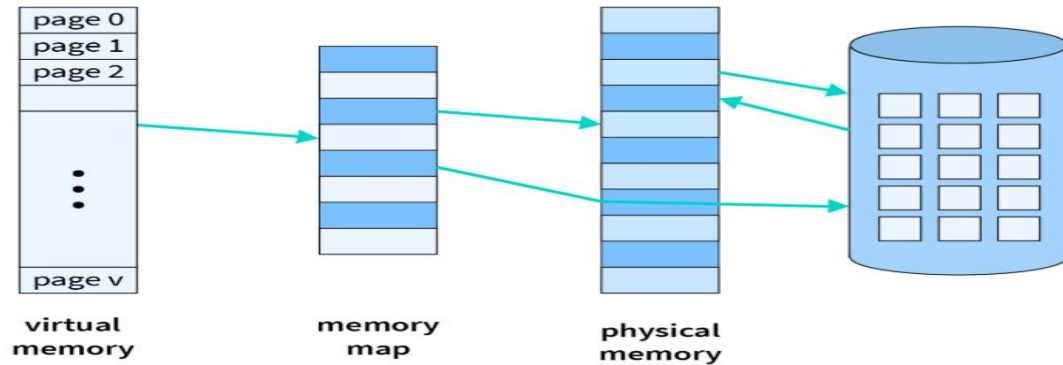
Virtual Memory

- What is virtual memory
- Demand paging
- Page faults
- Page replacement algorithms




What is virtual memory

- It is a part of the secondary storage that gives the user the **illusion that it is a part of the main memory**
- It helps in running multiple applications with low main memory and increases the degree of multiprogramming in systems without exhausting the RAM (Random Access Memory)
- It is commonly **implemented using demand paging**.



How Virtual Memory Works in a Modern OS?

Consider a system with 8 GB of RAM and a process that needs 16GB of memory:

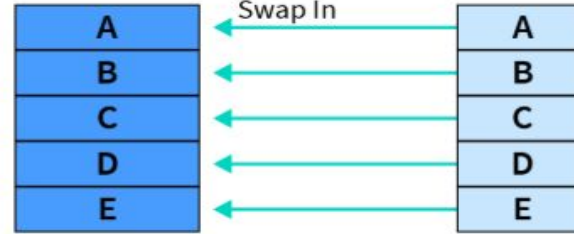
- The OS allocates **virtual memory** addresses for the process, which it can use as if it has 16GB of contiguous memory.
 - The OS creates a **page table** to map these virtual addresses to physical addresses in RAM.
 - Since the system only has 8 GB of RAM, only part of the process's memory will reside in RAM at any given time.
 - When the process tries to access memory that isn't in RAM, the **page table** indicates that the page is not present, and the OS triggers a **page fault**.
 - The OS then swaps out another part of memory from RAM (perhaps an unused page from another process) and swaps in the required page from the swap space on disk.
 - This process continues as needed, allowing the process to access the memory it needs while not exceeding the physical RAM.
- 

Swap In and Swap Out

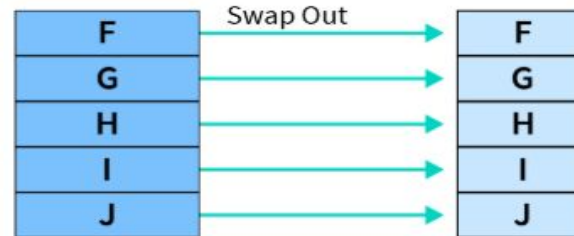
- When the primary memory (RAM) is insufficient to store data required by several applications, we use a method known as swap out to transfer certain programs from RAM to the hard drive.
- Similarly, when RAM becomes available, we swap in the applications from hard disk to RAM.
- We may manage many processes inside the same RAM by using swaps. Swapping aids in the creation of virtual memory and is cost-effective.

Main memory

Process 1



Process 2



Secondary Memory

K
L
M
N
O
P
Q
R
S
T
U
V

Key Terms:

- **Page Table:** A data structure used by the OS to map virtual addresses to physical addresses.
- **Page Fault:** A fault that occurs when a process accesses a virtual memory page that is not currently in physical memory.
- **Swap Space:** A portion of the disk used by the OS to store pages that are not currently in physical memory.
- **Thrashing:** A condition where the system spends too much time swapping pages in and out of memory, resulting in poor performance.



Page Fault

- This occurs when a program tries to access a part of memory (**a page**) that is **not currently in RAM**.
- The operating system must step in to retrieve the page from **secondary storage** (like a hard drive or SSD) and load it into **main memory** (RAM).

Types of Page Faults:

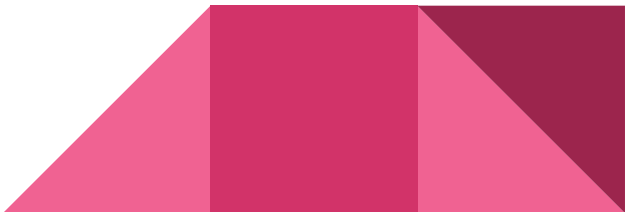
1. Minor Page Fault (Soft Page Fault)

- The page is not in RAM but is **in memory cache** (e.g., in a different part of memory).
- Fast to resolve

2. Major Page Fault (Hard Page Fault)

- The page is not in RAM or cache and must be loaded **from disk**.
- Slower and more costly.

3. Invalid Page Fault

- The memory address is invalid or the process doesn't have access.
 - Leads to **segmentation fault** or **process termination**.
- 

Page Fault Handling Process:

- **CPU tries to access a page.**
- **Page table lookup** shows page is not in memory.
- **Page fault interrupt** is triggered.
- OS suspends the process and:
 - Finds a free memory frame (or uses page replacement).
 - Loads the required page from disk.
 - Updates the page table.
- **Process resumes** execution as if nothing happened.

Example of Page Fault:

A program accesses address 0x0040, which is in **page 2**.

- Page 2 is not in RAM → **Page fault** occurs.
- OS fetches page 2 from disk.
- Loads it into an available frame.
- Updates page table → maps page 2 to new frame.
- Process continues.

Page Replacement Algorithms

- Page replacement algorithms are crucial in operating systems for memory management, specifically when dealing with **virtual memory**.
- These algorithms decide which memory pages to swap out when a new page needs to be loaded, but memory is full.

1. FIFO (First-In, First-Out) - Removes the oldest page in memory (the one loaded first).

- **Pros:** Simple to implement.
- **Cons:** May remove frequently used pages → **Belady's anomaly** (more frames may increase page faults).

2. LRU (Least Recently Used) - Replaces the page that hasn't been used for the longest time.

- **Pros:** Good performance, mimics real-world usage
- **Cons:** Requires tracking usage → more complex and expensive to implement.

3. Optimal (OPT) - Replaces the page that won't be used for the longest time in the future.

- **Pros:** Best possible performance.
- **Cons:** **Not practical** — future knowledge is required; used mainly for benchmarking.



4. Clock (Second Chance) - Circular buffer with a reference bit for each page. If the bit is 0, replace it; if 1, set it to 0 and move on.

- **Pros:** Efficient approximation of LRU, low overhead.
- **Cons:** Slightly more complex than FIFO.

5. NRU (Not Recently Used) - Uses reference and modify bits; pages are classified into 4 categories, and one from the lowest category is replaced.

- **Pros:** Prioritizes unmodified and unreferenced pages.
- **Cons:** Needs periodic bit resetting.

6. LFU (Least Frequently Used) - Replaces the page with the fewest accesses.

- **Pros:** Good if past frequency predicts future use.
- **Cons:** May hold onto pages that were heavily used in the past but are no longer needed.

7. MFU (Most Frequently Used) - Opposite of LFU; removes the most frequently used pages.

- **Pros:** Based on the idea that pages used often may no longer be needed.
- **Cons:** Less commonly used; generally performs worse.