UNIT-5– Notes

Memory Organization

◆ 1. Definition

Memory organization refers to the way computer memory is structured, arranged, and accessed by the CPU to store and retrieve instructions and data efficiently.
It determines:

- How memory is divided (hierarchically and physically)
- How data is addressed and accessed
- How the operating system and hardware coordinate to manage memory use

◆ 2. Types of Memory in a Computer System

Type	Description	Example
Primary Memory	Directly accessible by CPU; temporary storage	RAM, Cache, Registers
Secondary Memory	Non-volatile; used for long-term storage	Hard Disk, SSD
Tertiary Memory	Used for backup or archival	Magnetic tapes, Optical disks

◆ 3. Memory Hierarchy

Memory is organized in a **hierarchy** based on speed, cost, and capacity:

Level	Memory Type	Speed	Cost	Capacity	Managed By
1	**Registers**	Fastest	Very High	Very Low	CPU
2	**Cache Memory**	Very Fast	High	Small	Hardware
3	**Main Memory (RAM)**	Medium	Moderate	Medium	OS
4	**Secondary Memory**	Slow	Low	Large	OS / User
5	**Tertiary Storage**	Slowest	Lowest	Very Large	User

Concept:
Data moves between these levels depending on frequency of access (Principle of Locality).

◆ 4. Structure of Main Memory

Main memory (RAM) is divided into:

1. **Operating System Area** – reserved for system software.
2. **User Area** – used for user processes and programs.

Example Layout:

```
+----------------------------+
| Operating System (OS)      |
+----------------------------+
| User Program 1             |
+----------------------------+
| User Program 2             |
+----------------------------+
| Free Space                 |
+----------------------------+
```

◆ 5. Memory Addressing

There are two types of addresses:

- **Logical Address:** Generated by the CPU during program execution.
- **Physical Address:** Actual location in main memory.

Address Translation:
The **Memory Management Unit (MMU)** converts logical addresses into physical addresses.

◆ 6. Types of Memory Organization

(a) Sequential Organization

- Data is stored and accessed sequentially.
- Example: Magnetic tapes.

(b) Direct Organization

- Data can be accessed directly using address.
- Example: Hard disks, RAM.

(c) Associative (Content Addressable) Organization

- Data is accessed based on content rather than address.
- Used in cache memory.

◆ 7. Components of Memory Organization

1. **Registers** – Small, high-speed storage in CPU.
2. **Cache Memory** – Stores frequently accessed data.
3. **Main Memory (RAM)** – Holds programs and data currently in use.
4. **Secondary Memory** – Stores data permanently.
5. **Address Bus & Data Bus** – Used to transfer addresses and data between CPU and memory.

◆ 8. Memory Access Methods

Method	Description	Example
Sequential Access	Access data in a sequence	Tape drives
Direct Access	Access data directly via address	Hard disk
Random Access	Access any location instantly	RAM
Associative Access	Access by content	Cache

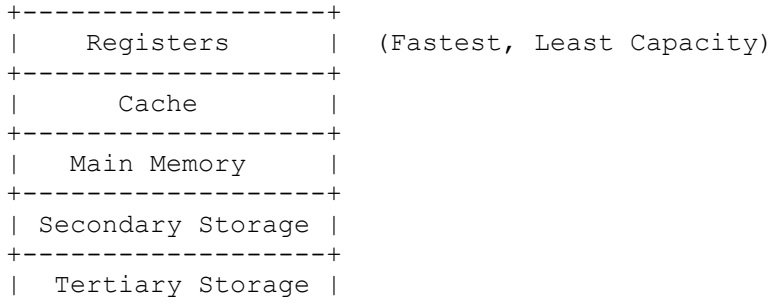
◆ 9. Memory Operations

- **Read Operation:** Data is fetched from memory to CPU.
- **Write Operation:** Data is written from CPU to memory.
- **Refresh Operation:** Required for DRAM to maintain data.

◆ 10. Importance of Memory Organization

- Efficient CPU performance
- Faster data access
- Better utilization of memory space
- Reduced delay in program execution
- Supports multiprogramming and multitasking

◆ 11. Example – Memory Hierarchy Diagram

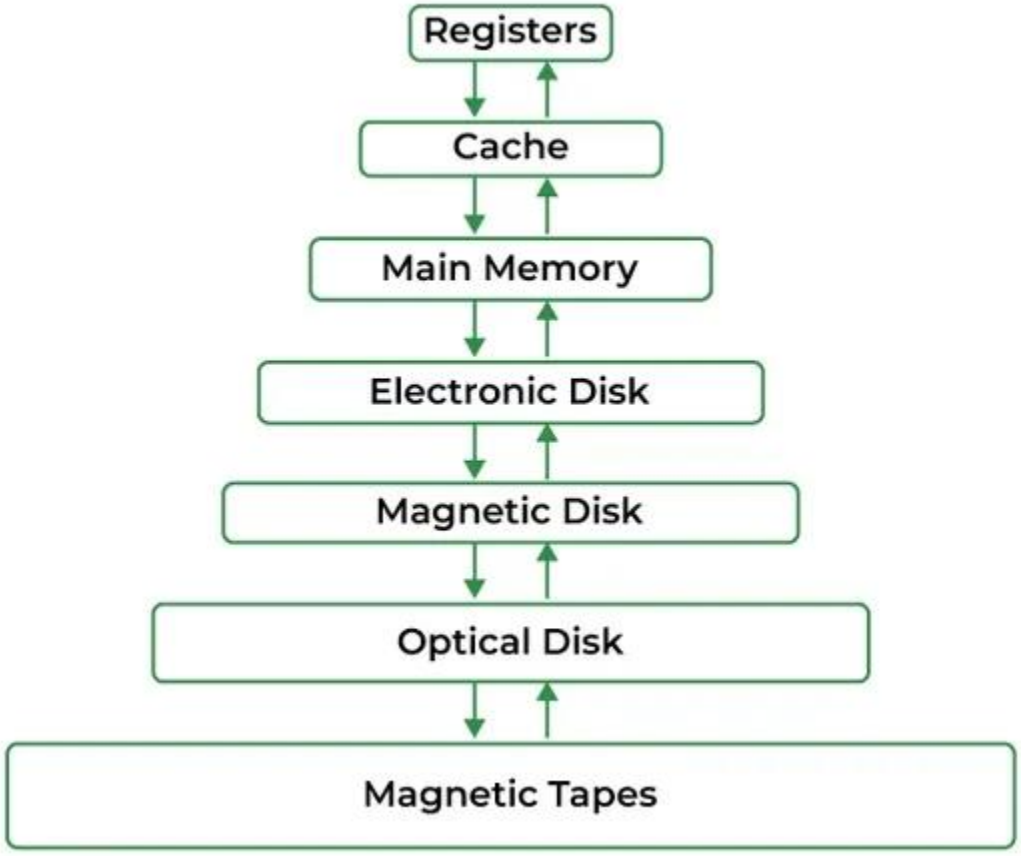


+------------------+

Memory Management

Memory management is a critical aspect of operating systems that ensures efficient use of the computer's memory resources. It controls how memory is allocated and deallocated to processes, which is key to both performance and stability. Below is a detailed overview of the various components and techniques involved in memory management.

Memory Management:



Why Memory Management is Required?

- 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.

Read more about Requirements of Memory Management System here.

Logical and Physical Address Space

- **Logical Address Space:** The logical address space is the set of all addresses that a process can generate using its CPU. It defines the range of memory locations available to the process from its perspective.

- **Physical Address Space:** The physical address space is the set of all actual memory addresses in the main memory (RAM). It represents the real locations where data and instructions are stored.

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:** Dynamic loading loads program routines into memory only when they are needed. This saves memory by not loading unused routines. The routines remain on disk in relocatable (can be loaded at any memory location) format until called. It allows better memory utilization, especially for large programs.

Static and Dynamic Linking

A **linker** combines object files into a single executable.

- **Static Linking:** All required modules are combined into one executable. No runtime dependency; some OSes support only this method.
- **Dynamic Linking:** Uses a **stub** (small code) for library calls. At runtime, the stub checks if the routine is in memory; if not, it loads it.

Memory Hierarchy

◆ 1. Definition

Memory hierarchy is the organization of different types of memory in a computer system based on **speed, cost, and capacity**.
It ensures that the CPU can access data efficiently and quickly while keeping overall cost low.

◆ 2. Purpose of Memory Hierarchy

- To overcome the **speed gap** between the CPU and main memory.
- To make frequently used data available quickly.
- To provide **large storage capacity** at a **low cost**.
- To achieve **efficient memory utilization**.

◆ 3. Principle of Locality

Memory hierarchy works based on the **principle of locality**, which means:

1. **Temporal Locality:**
   Recently accessed data is likely to be accessed again soon.
   → Example: Loop variables.
2. **Spatial Locality:**
   Data located close to recently accessed data is likely to be accessed soon.
   → Example: Sequential instruction execution.

◆ 4. Structure of Memory Hierarchy

Memory is organized into **multiple levels** — from fastest and smallest to slowest and largest:

Level	Memory Type	Speed	Cost per Bit	Capacity	Managed By
L1	**Registers**	Fastest	Very High	Very Low	CPU
L2	**Cache Memory**	Very Fast	High	Small	Hardware
L3	**Main Memory (RAM)**	Moderate	Medium	Medium	OS
L4	**Secondary Memory**	Slow	Low	Large	OS/User

Level	Memory Type	Speed	Cost per Bit	Capacity	Managed By
L5	**Tertiary Storage**	Slowest	Lowest	Very Large	User

◆ 5. Memory Hierarchy Diagram

```
+-------------------+
| CPU Registers     | ← Fastest, Smallest, Most Expensive
+-------------------+
| Cache Memory      |
+-------------------+
| Main Memory (RAM) |
+-------------------+
| Secondary Memory  |
+-------------------+
| Tertiary Storage  | ← Slowest, Largest, Cheapest
+-------------------+
```

◆ 6. Description of Each Level

1 Registers

- Located inside the CPU.
- Store temporary data and instructions.
- Fastest access time (in nanoseconds).
- Very limited capacity (few bytes to kilobytes).

2 Cache Memory

- High-speed memory between CPU and main memory.
- Stores **frequently used instructions/data**.
- Reduces CPU waiting time.
- Small in size (few MBs).
- Types: **L1, L2, L3 caches**.

3 Main Memory (RAM)

- Holds data and instructions currently being used.
- Volatile (data lost when power off).
- Access time in nanoseconds to microseconds.
- Managed by the operating system.

4 Secondary Memory

- Non-volatile permanent storage (e.g., HDD, SSD).
- Used when data does not fit into main memory.
- Access time in milliseconds.
- Much cheaper and larger than RAM.

5 Tertiary Storage

- Used for **backup and archival**.
- Examples: Magnetic tapes, optical discs, cloud storage.
- Slowest and lowest cost per bit.

◆ 7. Characteristics Comparison

Feature	Registers	Cache	Main Memory	Secondary	Tertiary
Speed	Very Fast	Fast	Medium	Slow	Very Slow
Cost/Bit	Very High	High	Medium	Low	Very Low
Capacity	Very Low	Low	Medium	High	Very High

Feature	Registers	Cache	Main Memory	Secondary	Tertiary
Volatility	Volatile	Volatile	Volatile	Non-volatile	Non-volatile
Location	CPU	Between CPU & RAM	Motherboard	External	External

◆ 8. Working of Memory Hierarchy

1. **CPU first checks Registers.**
   - If data is found → fastest access.
2. If not, **check Cache Memory** (Cache Hit/Miss).
3. If cache miss → fetch from **Main Memory**.
4. If not in main memory → load from **Secondary Storage** (Virtual Memory concept).
5. **Tertiary storage** is used for backup when data is not needed immediately.

◆ 9. Advantages of Memory Hierarchy

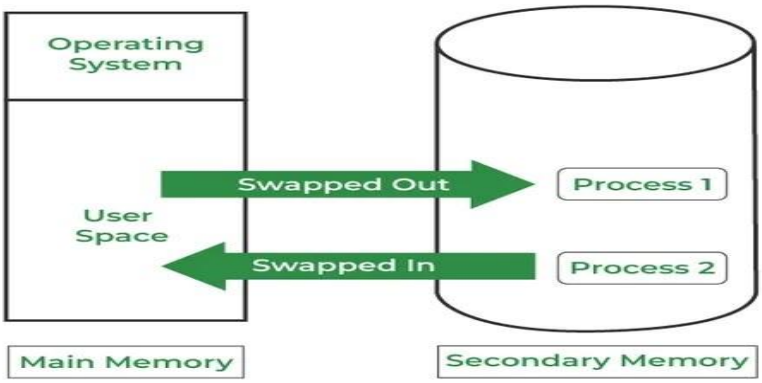
✓ Increased processing speed
✓ Efficient utilization of memory
✓ Lower overall cost
✓ Large apparent memory to user
✓ Reduces CPU idle time

◆ 10. Example of Access Time

Memory Type	Typical Access Time
Register	1 ns
Cache	2–10 ns
Main Memory	50–100 ns
SSD	100 µs
Hard Disk	5–10 ms
Tape	Seconds

Swapping

Swapping moves processes between main memory and secondary memory to manage limited memory space. It allows multiple processes to run by temporarily swapping out lower priority processes for higher priority ones. The swapped-out process resumes once it's loaded back. Transfer time depends on the amount of data swapped.



Swapping

Memory Management Strategies

◆ 1. Definition

Memory Management is a function of the operating system that **handles the allocation and deallocation of memory** to different programs and processes to ensure:

- Efficient utilization of main memory
- Fast execution of programs
- Protection and isolation among processes

◆ 2. Objectives of Memory Management

- To keep track of each memory location (used or free)
- To allocate memory to processes when required
- To deallocate memory when no longer needed
- To maximize CPU utilization and throughput
- To prevent memory fragmentation

◆ 3. Memory Management Strategies

Memory management strategies decide **how processes are placed in main memory**.
They are broadly divided into:

1. **Contiguous Allocation**
2. **Non-Contiguous Allocation**

◆ I. Contiguous Memory Allocation

Definition:

Each process is stored in **a single continuous block** of physical memory.

Characteristics:

- Simple to implement
- Fast access (because addresses are sequential)
- Causes **fragmentation** problems

1. Single User Contiguous Allocation

- Memory divided into **two parts**:
    1. **Operating System area**
    2. **User process area**
- Only **one process** runs at a time.
- Used in **simple systems** (e.g., early PCs, embedded systems)

Layout:

```
+-----------------------+
| Operating System      |
+-----------------------+
| User Process          |
+-----------------------+
```

2. Fixed Partition Multiprogramming

- Main memory divided into **fixed-size partitions**.
- Each partition holds **exactly one process**.
- When a partition is free, another process can occupy it.

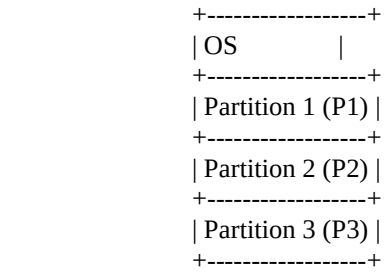
Advantages:

- Simple implementation
- Supports limited multiprogramming

Disadvantages:

- **Internal fragmentation:** unused space within a partition
- Fixed number of processes → limited flexibility

Example Layout:



3. Variable Partition Multiprogramming

- Memory divided into **variable-sized partitions dynamically** depending on process size.
- When a process ends, its space is freed for reuse.

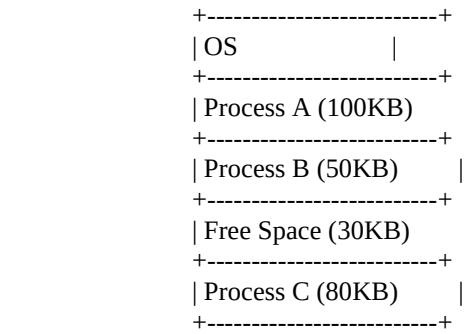
Advantages:

- Reduces internal fragmentation
- Flexible allocation according to process size

Disadvantages:

- Causes **external fragmentation** (small gaps left between partitions)
- Compaction may be required to combine free spaces

Layout Example:



4. Memory Swapping

- A process can be **swapped out** from main memory to disk and **swapped in** later.
- Used when memory is full to accommodate new processes.

Advantages:

- Increases degree of multiprogramming
- Allows execution of large programs

Disadvantages:

- Swapping time overhead
- Disk I/O is slow

◈ II. Non-Contiguous Memory Allocation

Definition:

A process can occupy **several non-adjacent blocks** of memory.
Used in **modern operating systems** (paging, segmentation).

Advantages:

- Reduces fragmentation

- Efficient use of memory
- Allows processes larger than physical memory

1. Paging

- Divides memory into **fixed-size blocks**:
  - Logical memory → **Pages**
  - Physical memory → **Frames**
- Each page of a process is loaded into any available frame.
- Uses a **page table** to map pages to frames.

Advantages:

- Eliminates external fragmentation
- Easy to allocate/free memory

Disadvantages:

- **Internal fragmentation** within last page
- Page table overhead

2. Segmentation

- Divides process memory into **variable-size logical segments** like:
  - Code
  - Data
  - Stack
- Each segment has a base and limit stored in the **segment table**.

Advantages:

- Reflects logical structure of program
- Supports sharing and protection

Disadvantages:

- Causes **external fragmentation**

3. Paging + Segmentation (Combined System)

- Each segment is divided into pages.
- Combines benefits:
  - Logical view of segmentation
  - Efficient allocation of paging

Used in **modern systems (like Intel processors)**.

◆ 4. Fragmentation in Memory Management

Type	Cause	Example	Solution
Internal Fragmentation	Fixed partitions larger than process	100KB partition, process needs 70KB	Variable partitioning
External Fragmentation	Free memory scattered in small blocks	3 free spaces: 10KB + 15KB + 5KB	Compaction or paging

---

◆ 5. Allocation Algorithms

When a process requests memory, the OS chooses where to place it:

Algorithm	Description
First Fit	Allocate the first available block large enough
Best Fit	Allocate the smallest block that fits the process
Worst Fit	Allocate the largest available block
Next Fit	Similar to first fit but starts searching from the last allocated position

---

◆ 6. Compaction

- Process of **rearranging processes in memory** to place all free memory together.
- Reduces external fragmentation.
- Time-consuming, so done occasionally.

Virtual Memory Organization – Notes

◆ 1. Definition

Virtual Memory is a memory management technique that allows the execution of processes that **may not be completely in main memory (RAM)**.
It gives the **illusion of a large, continuous main memory** by using a portion of the **secondary storage (disk)** as an extension of main memory.

◆ 2. Purpose of Virtual Memory

- To run programs **larger than physical memory**.
- To allow **multiprogramming** (multiple processes in memory).
- To provide **process isolation and protection**.
- To increase **CPU utilization** by reducing idle time.
- To make memory management **automatic and flexible**.

◆ 3. Concept of Virtual Memory

Each process has its **own logical (virtual) address space**.
This address space is mapped to physical memory by the **Memory Management Unit (MMU)**.
Conceptual View:

```
+-------------------+     +-------------------+
| Virtual Address   | ---> | Physical Address  |
| (Logical View)    |     | (Actual RAM Frame) |
+-------------------+     +-------------------+
```

The **unused part of a process** is stored temporarily on disk (called **swap space**).

◆ 4. Advantages of Virtual Memory

✓ Programs larger than physical memory can execute.
✓ Provides isolation and protection between processes.
✓ Enables efficient use of main memory.
✓ Reduces programmer's burden of memory management.
✓ Allows more processes to be loaded (higher degree of multiprogramming).

◆ 5. Virtual Memory Structure

Virtual memory divides:

- **Logical Memory** → into **pages**
- **Physical Memory (RAM)** → into **frames**

Only **active pages** are kept in RAM, and the rest are stored on **disk**.

◆ 6. Multilevel Storage Organization

Virtual memory relies on a **multilevel storage hierarchy**:

Level	Memory Type	Role
1	**Main Memory (RAM)**	Stores currently active pages
2	**Secondary Storage (Disk)**	Stores inactive pages (swap area)
3	**Cache / Registers**	Stores recently accessed data

Working Principle:
Frequently used data stays closer to the CPU → improves performance.

◆ 7. Block Mapping

In virtual memory, **block mapping** means mapping between **virtual blocks (pages)** and **physical blocks (frames)**.

Mapping types (used in cache and paging systems):

1. **Direct Mapping** – Each block maps to a specific frame.
2. **Associative Mapping** – Any block can go into any frame.
3. **Set-Associative Mapping** – A compromise between the two.

Mapping Table:

Maintained by the OS to translate virtual addresses into physical ones.

◆ 8. Paging – Basic Concepts

- Divides **logical memory into fixed-size pages** (e.g., 4 KB each).
- Divides **physical memory into frames** of the same size.
- OS maintains a **Page Table** that maps each page to a frame.

Example:

Page No.	Frame No.
0	5
1	2
2	7

When a process accesses data:

- CPU generates a **virtual address (page number + offset)**
- MMU translates it using the page table.

If the page is **not in RAM**, a **page fault** occurs → OS loads the page from disk.

Advantages of Paging

✓ Eliminates external fragmentation.

✓ Allows non-contiguous memory allocation.

✓ Simplifies memory allocation and protection.

Disadvantages

✗ Internal fragmentation (unused space within last page).

✗ Page table overhead.

✗ Page fault handling causes delays.

◆ 9. Segmentation – Basic Concepts

- Divides memory into **logical segments** such as:
  - Code Segment
  - Data Segment
  - Stack Segment
- Each segment has **variable length**.

Each segment is identified by a **segment number** and has a **base** (starting address) and **limit** (size).

Addressing:

Logical Address = <Segment Number, Offset>

MMU adds the base address to the offset → physical address.

Advantages:

- Reflects logical structure of programs.
- Enables sharing and protection.

Disadvantages:

- Causes **external fragmentation**.
- More complex than paging.

◆ 10. Paging + Segmentation Systems (Combined Approach)

Many modern systems (like Intel x86) combine both techniques:

- Each **segment** is divided into **pages**.
- Paging handles physical memory allocation.
- Segmentation manages logical program structure.

Benefits:

✓ Logical division of memory

✓ Efficient physical memory use

✓ Reduced fragmentation

◆ 11. Virtual Memory Address Translation

Virtual Address → Physical Address

1. CPU generates a **virtual address**.
2. MMU checks **page table** to find corresponding **frame**.
3. If page not in memory → **page fault** → load page from disk.
4. After loading, page table is updated.

Virtual Memory Management

◆ 1. Definition

Virtual Memory Management is the process by which the **Operating System (OS) manages virtual memory**, deciding:

- Which parts of a process are kept in **main memory (RAM)**
- Which parts are **stored on disk**
- When and how to **swap pages in and out**

It uses techniques like **demand paging** and **page replacement algorithms** to make efficient use of limited physical memory.

◆ 2. Objectives of Virtual Memory Management

- Execute **large programs** with limited physical memory
- **Increase CPU utilization** and system throughput
- Provide **isolation** between processes
- Reduce memory fragmentation
- **Efficiently handle page faults**

◆ 3. Conceptual Overview

Virtual memory divides:

- **Logical (virtual) address space** → into **pages**
- **Physical memory (RAM)** → into **frames**

Only the required pages are kept in RAM; the rest stay in **secondary storage (swap area)**.

Address translation:

Performed by the **Memory Management Unit (MMU)** using a **page table**.

◆ 4. Components of Virtual Memory Management

Component	Description
Page Table	Maps virtual pages to physical frames

Component	Description
Frame Table	Keeps track of which frames are occupied
Swap Space	Area on disk used to store pages not currently in memory
MMU (Memory Management Unit)	Hardware that translates virtual → physical addresses
TLB (Translation Lookaside Buffer)	Cache that stores recent page translations for faster access

◆ 5. Demand Paging

Definition:

In **demand paging**, pages are **loaded into main memory only when they are needed** during execution — not all at once.

Steps in Demand Paging:

1. CPU generates a **virtual address**.
2. MMU checks if the page is in memory.
    - If **yes** → normal execution.
    - If **no** → **page fault** occurs.
3. OS selects a **free frame** (or replaces an existing one).
4. The **required page** is fetched from disk into memory.
5. Page table is updated and process resumes.

Advantages:

✅ Efficient use of memory (only required pages are loaded).

✅ Faster program start-up.

✅ Enables execution of large programs.

Disadvantages:

❌ Page fault overhead (disk access is slow).

❌ Requires additional hardware support (MMU, TLB).

❌ May lead to **thrashing** if too many page faults occur.

◆ 6. Page Fault

- Occurs when a page required by a process **is not in main memory**.
- The OS must handle it by:
    1. Suspending the process.
    2. Finding the required page on disk.
    3. Loading it into a free or replaced frame.
    4. Updating the page table.
    5. Restarting the process.

Page Fault Steps Diagrammatically:

Page Request → Check Page Table → Page Not Found → Page Fault → Load from Disk → Update Page Table → Resume Execution

◆ 7. Page Replacement Strategies

When a **page fault** occurs and **no free frame** is available, OS must choose **which page to replace**.

Common Page Replacement Algorithms

Algorithm	Description	Pros	Cons
FIFO (First-In First-Out)	Replaces the oldest loaded page.	Simple	Poor performance; may replace frequently used pages.
LRU (Least Recently Used)	Replaces the page that hasn't been used for the longest time.	Good performance	Requires time tracking; hardware support needed.
Optimal (OPT)	Replaces the page that will not be used for the longest time in the future.	Theoretically best	Not practical (requires future knowledge).
Clock (Second Chance)	Modified FIFO; uses a reference bit to give pages a "second chance."	Efficient and simple	Slightly more complex than FIFO.
LFU (Least Frequently Used)	Replaces the page with the least number of accesses.	Based on actual usage	May not reflect current locality.

◆ 8. Page Replacement Example (FIFO)

Given:
Memory with 3 frames and reference string:
7, 0, 1, 2, 0, 3, 0, 4
Steps:

- Load 7, 0, 1 → full
- Next: Replace 7 (oldest) → 2
- Continue with replacement based on arrival order.

Page Faults: occur when required page not found.

◆ 9. Working Set Model

- The **Working Set** of a process = set of pages used recently.
- OS tries to keep this set in memory to reduce page faults.
- Helps detect **thrashing** (excessive paging activity).

◆ 10. Thrashing

Definition:

When the CPU spends more time **swapping pages in and out of memory** than executing instructions.

Causes:

- Too many processes in main memory.
- Insufficient frames allocated per process.
- Poor page replacement policy.

Solutions:

✓ Reduce degree of multiprogramming.

✓ Use working set or page fault frequency control.

✓ Increase physical memory.

◆ 11. Effective Access Time (EAT)

To measure performance:
EAT=(1−p)×tm+p×(tp+tm)EAT = (1 - p) \times t_m + p \times (t_p + t_m)EAT=(1−p)×tm+p×(tp+tm)
Where:

- p = page fault rate
- t_m = memory access time
- t_p = page fault service time (includes disk I/O)

Lower p → better performance.

◆ 12. Segmentation and Paging in Virtual Memory Management

Feature	Paging	Segmentation
Division	Fixed-size pages	Variable-size segments
Purpose	Physical memory management	Logical memory management
Fragmentation	Internal	External
Address	Page number + offset	Segment number + offset

Modern OSs often **combine both** for efficient virtual memory handling.

◆ 13. Advantages of Virtual Memory Management

✅ Efficient RAM utilization
✅ Supports large address spaces
✅ Provides memory protection & isolation
✅ Increases degree of multiprogramming
✅ Simplifies program loading

◆ 14. Disadvantages

❌ Page fault overhead
❌ Thrashing under heavy load
❌ Complex hardware and software support
❌ Disk access latency

Page Replacement Strategies (with Examples & Solutions)

◆ 1. Introduction

When a **page fault** occurs and there is **no free frame** available in main memory, the **Operating System** must decide **which page to remove** — this decision is made by the **page replacement algorithm**.

Goal:

☞ Minimize the number of **page faults** (i.e., disk accesses).
☞ Maintain **good CPU utilization** and **system performance**.

◆ 2. Common Page Replacement Algorithms

Algorithm	Description	Type
FIFO (First-In First-Out)	Replace the oldest page loaded into memory	Simple
OPT (Optimal)	Replace the page that will not be used for the longest time in future	Theoretical Best
LRU (Least Recently Used)	Replace the page that has not been used for the longest time	Practical & efficient
Clock / Second Chance	Modified FIFO with reference bits	Balanced
LFU (Least Frequently Used)	Replace the page with least usage count	Frequency-based

◆ 3. FIFO (First-In, First-Out) Algorithm

Concept:

- The page that entered first will be replaced first — like a **queue**.

Example 1: FIFO

Reference string:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3
Frames available: 3

Step	Page	Frame 1	Frame 2	Frame 3	Page Fault?
1	7	7	-	-	✓
2	0	7	0	-	✓
3	1	7	0	1	✓
4	2	2	0	1	✓ (7 replaced)
5	0	2	0	1	✗
6	3	2	3	1	✓ (0 replaced)
7	0	2	3	0	✓ (1 replaced)
8	4	4	3	0	✓ (2 replaced)
9	2	4	2	0	✓ (3 replaced)
10	3	4	2	3	✓ (0 replaced)

✓ **Total Page Faults = 9**

◆ 4. Optimal (OPT) Algorithm

Concept:
Replace the page that **will not be used for the longest period in the future**.
Best theoretical algorithm — used for comparison.

Example 2: OPTIMAL

Reference string:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3
Frames available: 3

Step	Page	Frame 1	Frame 2	Frame 3	Page Fault?	Replacement
1	7	7	-	-	✓	-
2	0	7	0	-	✓	-
3	1	7	0	1	✓	-
4	2	2	0	1	✓	7 (used farthest later)
5	0	2	0	1	✗	-
6	3	2	0	3	✓	1 replaced
7	0	2	0	3	✗	-
8	4	4	0	3	✓	2 replaced

Step	Page	Frame 1	Frame 2	Frame 3	Page Fault?	Replacement
9	2	4	0	2	✓	3 replaced
10	3	3	0	2	✓	4 replaced

✓ **Total Page Faults = 7**
→ **Optimal gives the minimum possible page faults.**

◆ 5. LRU (Least Recently Used) Algorithm

Concept:
Replace the page that **has not been used for the longest time** (based on recent history).
Implementation:

- Use **stack** or **timestamps**.

Example 3: LRU

Reference string:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3
Frames: 3

Step	Page	Frame 1	Frame 2	Frame 3	Page Fault?	Replacement
1	7	7	-	-	✓	-
2	0	7	0	-	✓	-
3	1	7	0	1	✓	-
4	2	2	0	1	✓	7 replaced
5	0	2	0	1	✗	-
6	3	2	0	3	✓	1 replaced
7	0	2	0	3	✗	-
8	4	4	0	3	✓	2 replaced
9	2	4	2	3	✓	0 replaced
10	3	4	2	3	✗	-

✓ **Total Page Faults = 7**
→ **LRU often performs close to the Optimal algorithm.**