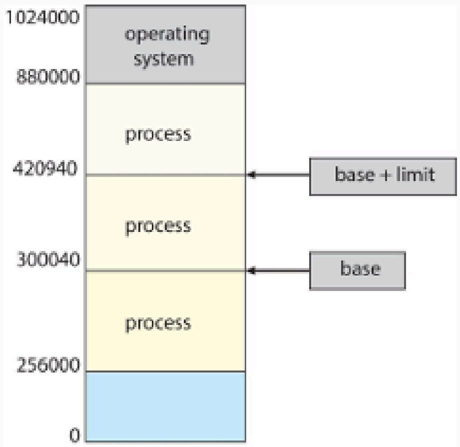
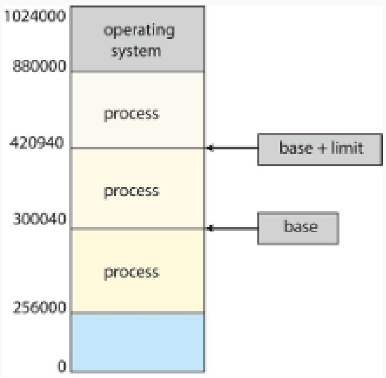
Memory Protection: Base and Limit Registers

- Base Register- Holds the smallest legal physical memory address
- Limit Register- specifies the size of the range



Memory Protection: Base and Limit Registers

- Example:
- if the base register holds 300040
- and the limit register is 120900,
- then the program can legally access all addresses from 300040 through 420939 (inclusive).



Memory Management – Notes (Slides 1-25)

Slide 1-3: Introduction to Memory Management

What is Memory Management?

- Memory Management is responsible for handling **memory allocation** and **deallocation** in an operating system.
- It ensures efficient execution of programs by organizing memory.
- **Objectives:**
    1. Organize **memory hardware** efficiently.
    2. Implement various **memory management techniques**.

Slide 4-5: Memory Basics

What is Memory?

- Memory is **a large array of bytes/words**, each with a unique address.
- It stores:
    - **Program instructions** to be executed.
    - **Data** required for execution.

How does the CPU interact with Memory?

- **Fetch-Decode-Execute Cycle**:
    1. CPU **fetches** the instruction from memory.
    2. **Decodes** the instruction and fetches operands from memory (if required).
    3. **Executes** the instruction and may store results back in memory.
- **Program Counter (PC)**: Holds the address of the next instruction to execute.

Slide 6-7: CPU-Memory Interaction

How does the CPU fetch and execute instructions?

- CPU fetches instructions from memory using the **Program Counter (PC)**.
- These instructions may include:
    - **Reading data from memory.**
    - **Writing data to memory.**

Instruction Execution Cycle

1. **Fetch instruction** from memory.
2. **Decode the instruction.**
3. **Fetch operands** (if required).
4. **Execute** the instruction.
5. **Store results** (if needed).

Memory Requests

- Memory unit receives:
    - **Read requests:** Address + Read instruction.
    - **Write requests:** Address + Data + Write instruction.

Slide 8-10: Issues in Memory Management

What are the Key Challenges in Memory Management?

Issue	Solution
Access Speed	Use **cache memory** between CPU and RAM.
Data Protection	Implement **base and limit registers**.
Limited Memory Size	Use **Swapping & Virtual Memory** to extend memory.

Why is Main Memory Important?

- **CPU can only directly access main memory & registers.**
- **Disk addresses cannot be accessed directly.**
- Data & instructions must be in **RAM for execution**.

Slide 11-12: Basic Memory Hardware

How does the CPU access memory?

1. **Registers**: Fastest memory (one CPU cycle access).
2. **Cache Memory**: Small but fast memory between CPU and RAM.
3. **Main Memory (RAM)**: Slower, takes multiple CPU cycles.
4. **Disk (Secondary Storage)**: Cannot be accessed directly.

What is the Role of Cache Memory?

- Cache sits **between CPU and RAM**.
- **Reduces CPU stalls** caused by slow memory access.

Slide 13-14: Memory Protection

Why is Memory Protection Needed?

- Prevents **unauthorized memory access** between processes.
- Protects **operating system from user programs**.

How is Memory Protection Implemented?

- **Base Register:** Stores the smallest physical memory address a process can access.
- **Limit Register:** Defines the maximum address a process can access.
- **Hardware Protection Mechanism:**
    - Every memory access is checked against **Base and Limit registers**.
    - **Illegal access → Trap to OS** (error).

Slide 15-17: Memory Access Protection

How does the OS Prevent Unauthorized Access?

- CPU must **check every memory access** in **user mode**.
- **Instructions to modify Base/Limit Registers are privileged**.
- Only the **OS can load Base & Limit Registers**.

What happens when a process violates memory protection?

- If a user process tries to access memory **outside its range**:
    - **Trap occurs** → OS handles the error.
    - Process may be **terminated or restricted**.

Slide 18-20: Address Binding

What is Address Binding?

- The process of mapping **logical addresses** (generated by the CPU) to **physical addresses** (actual locations in RAM).

When does Address Binding Occur?

1. **Compile-Time Binding:**
    - If memory location is **known at compile time**, absolute addresses are assigned.
    - **Disadvantage:** If program location changes, recompilation is needed.
2. **Load-Time Binding:**
    - If memory location is **not known at compile time**, relocatable addresses are used.
    - Final binding happens at **load time**.
3. **Execution-Time Binding:**
    - If a process **can move during execution**, addresses must be mapped dynamically.
    - Requires **Memory Management Unit (MMU)**.

Slide 21-22: Logical vs. Physical Address Space

What is a Logical Address?

- The address **generated by the CPU**.
- Also known as **Virtual Address**.

What is a Physical Address?

- The actual **location in RAM** where instructions/data are stored.

When are Logical and Physical Addresses the Same?

- **Compile-Time & Load-Time Binding** → Logical and Physical Addresses are **identical**.

When do Logical and Physical Addresses Differ?

- **Execution-Time Binding** → Logical and Physical Addresses are **different**.
- MMU dynamically translates **Logical** → **Physical** address.

Logical vs. Physical Address Space

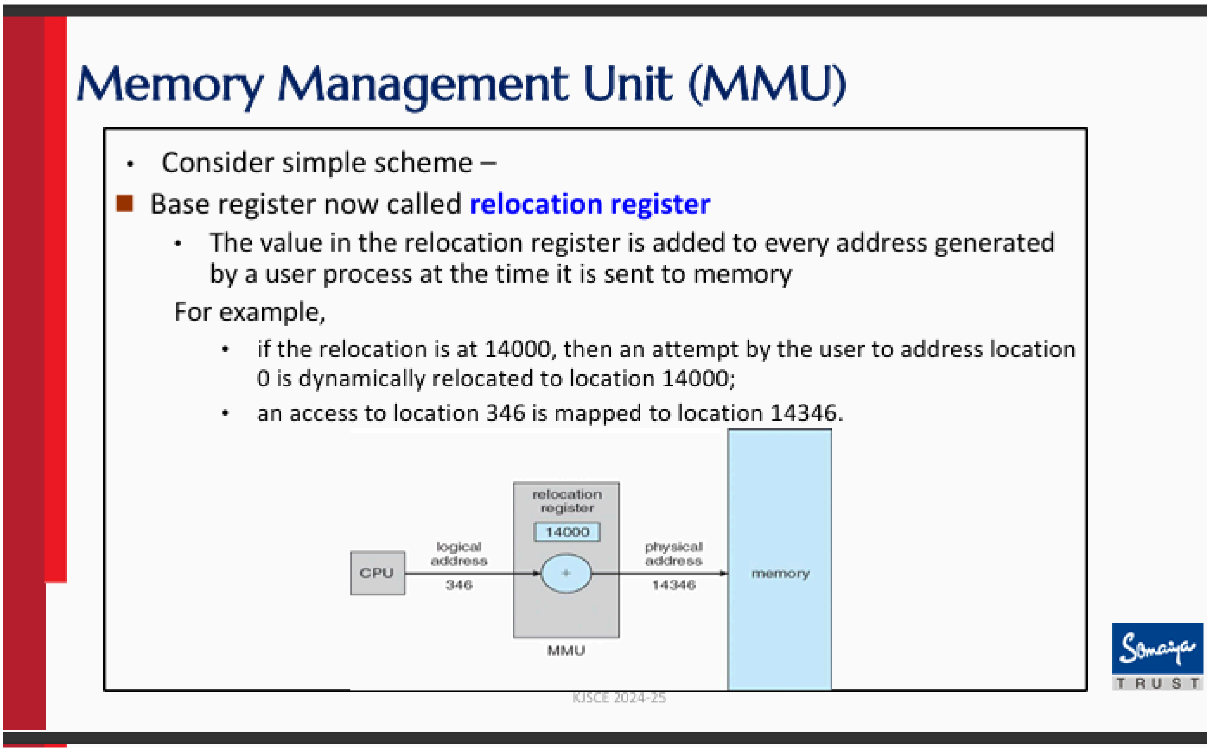
Concept	Logical Address	Physical Address
Definition	Address generated by CPU	Actual address in RAM
Also Called	Virtual Address	Real Memory Address
Used By	User Programs	Memory Hardware
Visibility	Hidden from the process	Visible to Memory Unit

Slide 23-25: Memory Management Unit (MMU)

What is the Memory Management Unit (MMU)?

- **Hardware device** that **translates Logical Addresses into Physical Addresses**.

How does the MMU Work?

- Uses a **relocation register** (Base Register) to add an **offset** to logical addresses.
- 

- Example:
  - **Logical Address:** 100
  - **Relocation Register Value:** 14000
  - **Physical Address:** 14100

Why is MMU Important?

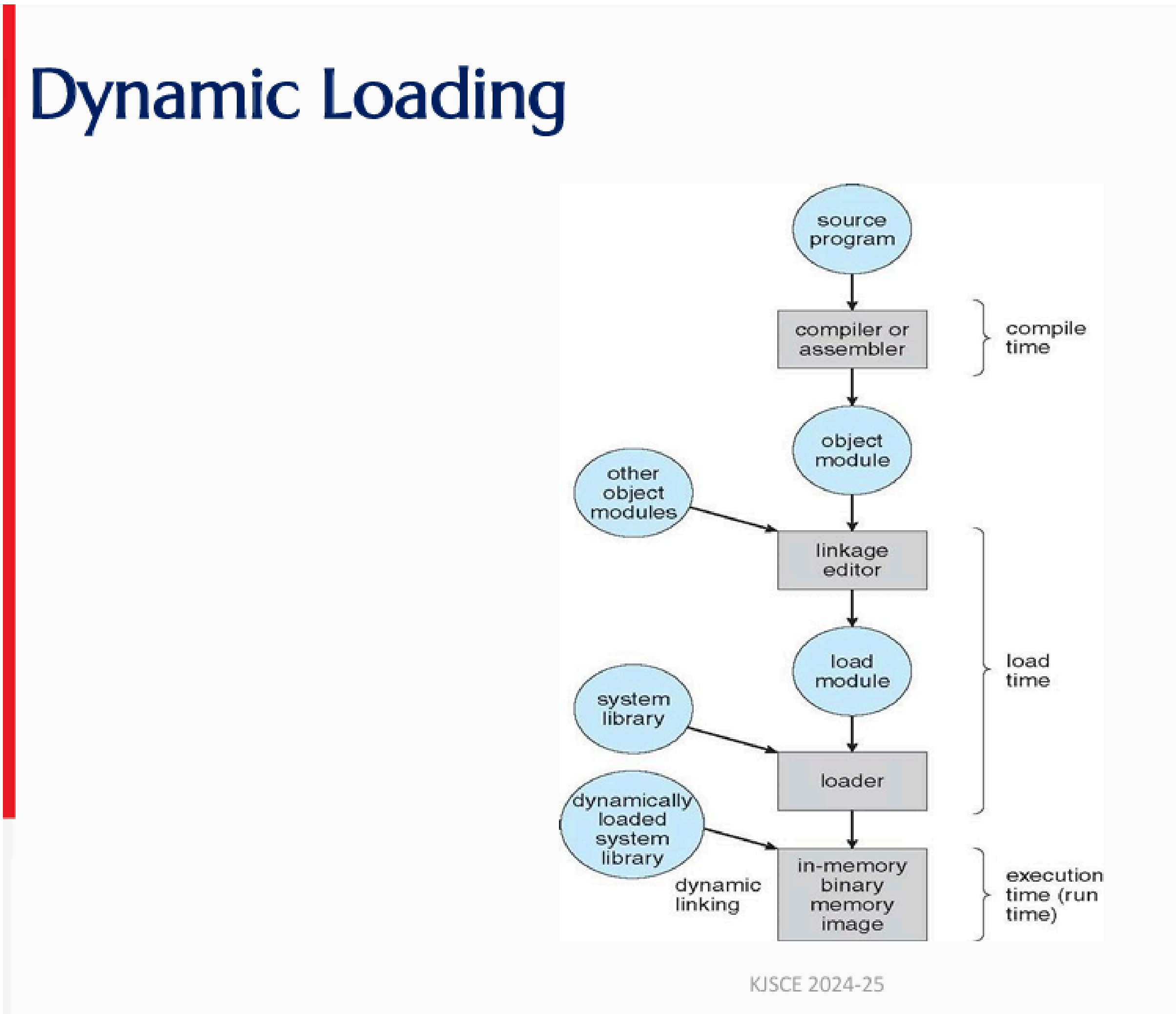
✔ Allows **dynamic relocation** of processes.
✔ Ensures **memory protection**.
✔ Supports **virtual memory mechanisms**.

Memory Management Unit (MMU)

- The user program never sees the 'real' physical address.
- User program deals with logical addresses
- The memory mapping hardware converts logical addresses to physical addresses
- Logical addresses: Range is 0 to max
- Physical addresses: R+0 to R+max where R is base value

Summary of Slides 1-25

Topic	Key Takeaways
Memory Basics	Memory contains program & data.
CPU-Memory Interaction	Instructions are fetched from memory for execution.
Memory Protection	Base & Limit Registers prevent illegal memory access.
Address Binding	Converts logical to physical addresses.
Logical vs. Physical Addresses	Logical (CPU) differs from Physical (RAM) in execution-time binding.
Memory Management Unit (MMU)	Maps logical addresses to physical addresses dynamically.



Slide 26-28: Dynamic Loading

What is Dynamic Loading?

- Instead of **loading the entire program into memory**, only required routines/functions are loaded **when needed**.
- **Why?** To **save memory** and allow execution of **large programs** that may not fit entirely in RAM.

How Does Dynamic Loading Work?

1. The **main program is loaded** into memory.
2. When a routine (function) **is called**, the program **checks if it's already loaded**.
3. If **not found**, the routine is **loaded from disk to memory**.
4. The program’s **address table updates** with the new routine’s location.
5. Execution resumes with the newly loaded function.

Advantages of Dynamic Loading

✔ **Saves memory** by **loading only necessary parts** of a program.
✔ **Reduces initial loading time** (faster program startup).
✔ **Can handle large programs** efficiently.
✔ **No OS-level support needed** (can be implemented at the program level).

Slide 29-31: Dynamic Linking

What is Dynamic Linking?

- Instead of including system libraries **inside the program**, dynamic linking **loads libraries at runtime**.
- **Why?** To **reduce memory usage and enable easy updates** of shared libraries.

How Does Dynamic Linking Work?

1. A **stub (placeholder)** is included in the program for each required function.
2. When a function is called:
   - **Stub checks** if the function is already in memory.
   - If **not**, the shared library is **loaded from disk**.
   - Stub is **replaced with the function's actual address**.
3. The function is now available, and next time it runs **directly from memory**.

Advantages of Dynamic Linking

✔ **Reduces executable size** (no need to store entire libraries in each program).
✔ **Faster loading of programs**.
✔ **Allows automatic updates** (a library update applies to all programs using it).

Slide 32-34: Swapping

What is Swapping?

- **Swapping temporarily moves processes from RAM to disk** to free memory for active processes.
- The swapped-out process is **brought back into RAM** when needed.

How Does Swapping Work?

1. A **process is swapped out** (moved from RAM to disk) when inactive.
2. Another process is **swapped in** (moved from disk to RAM) to execute.
3. When the swapped-out process is needed again, it is **loaded back** into RAM.

Example: Swapping Calculation

- Process Size = **10 MB**
- Disk Transfer Speed = **40 MB/s**
- Time to swap **in or out** = **10MB / 40MB/s = 0.25 sec (250 ms)**
- **Total swap time (in + out) = 500 ms**

Problems with Swapping

❌ **Slow speed** (disk is much slower than RAM).
❌ **High CPU overhead** (frequent swaps affect performance).
❌ **Not suitable for real-time systems** (delays execution).

Slide 35-37: Dispatcher and Swapper

What is the Dispatcher?

- **The Dispatcher is responsible for context switching** and handing control to a process selected by the CPU scheduler.
- **It is the final step in CPU scheduling.**

Dispatcher Functions

1. **Switches context** (saves the state of the old process and loads the new process).
2. **Loads the process's registers** from PCB (Process Control Block).
3. **Transfers control to the selected process**.

Dispatcher Performance - Dispatch Latency

- **Dispatch latency**: The time taken to stop one process and start another.
- A good dispatcher has **low dispatch latency** to minimize delays.

What is the Swapper?

- **The Swapper is responsible for moving processes in and out of memory (swapping).**
- It **works with the CPU scheduler and Dispatcher** to manage memory efficiently.

How Does the Swapper Work?

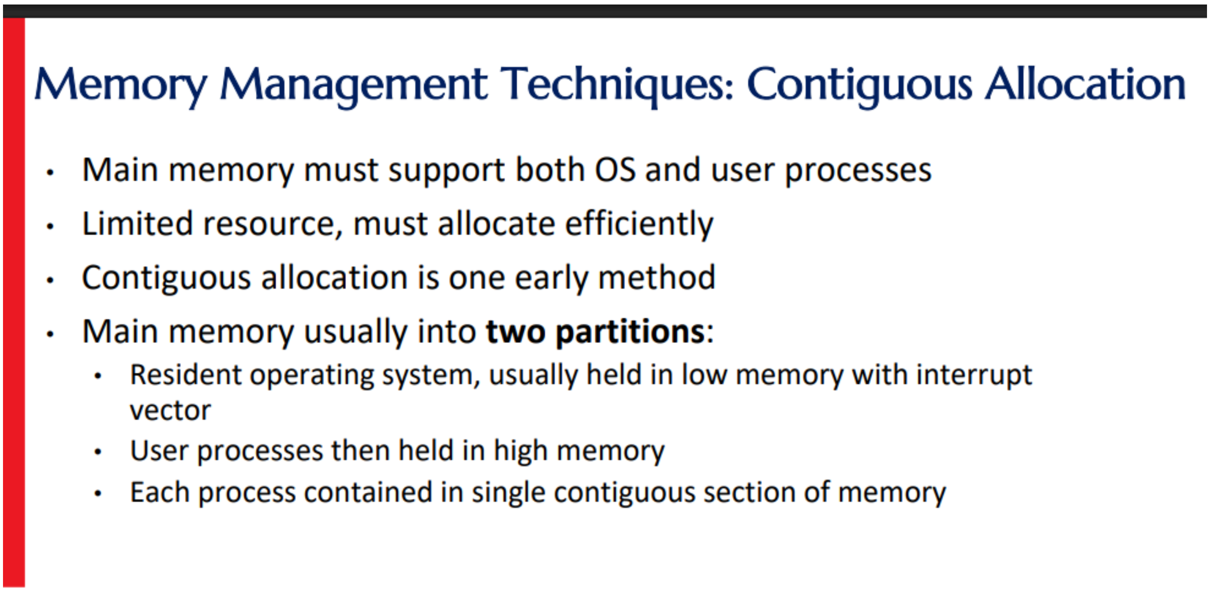
1. **If no free memory is available, the Swapper moves a process to disk.**
2. **Loads a new process into the freed-up memory.**

3. **When a swapped-out process is needed again, the Swapper loads it back into memory.**

Swapping Process with Dispatcher and Swapper

Component	Function
Swapper	Moves processes **between RAM and disk**.
Dispatcher	Switches between processes **already in RAM**.
CPU Scheduler	Chooses which process to execute next.

Slide 38-40: Contiguous Memory Allocation



What is Contiguous Memory Allocation?

- **Each process is stored in a single continuous block of memory**.
- Memory is allocated **in fixed or variable-sized partitions**.

Types of Contiguous Allocation:

1. **Fixed Partitioning (Static Allocation)**
   - Memory is divided into **predefined fixed-sized partitions**.
   - **Each partition holds one process.**
   - **Problem: Internal Fragmentation** (wasted space inside partitions).
2. **Variable Partitioning (Dynamic Allocation)**

- Memory is **divided dynamically** based on process needs.
- **Problem: External Fragmentation** (free memory is scattered).

Advantages & Disadvantages

Method	Pros	Cons
Fixed Partitioning	Simple, fast allocation	Wastes memory (internal fragmentation)
Variable Partitioning	Efficient, flexible	Causes fragmentation (external)

Summary of Slides 26-40

Topic	Key Takeaways
Dynamic Loading	Loads only necessary routines to save memory.
Dynamic Linking	Links system libraries at runtime, saving disk space.
Swapping	Temporarily moves processes between RAM & disk.
Dispatcher	Switches processes already in RAM for execution.
Swapper	Moves processes between RAM and disk.
Contiguous Allocation	Assigns memory in a single block (fixed or dynamic).

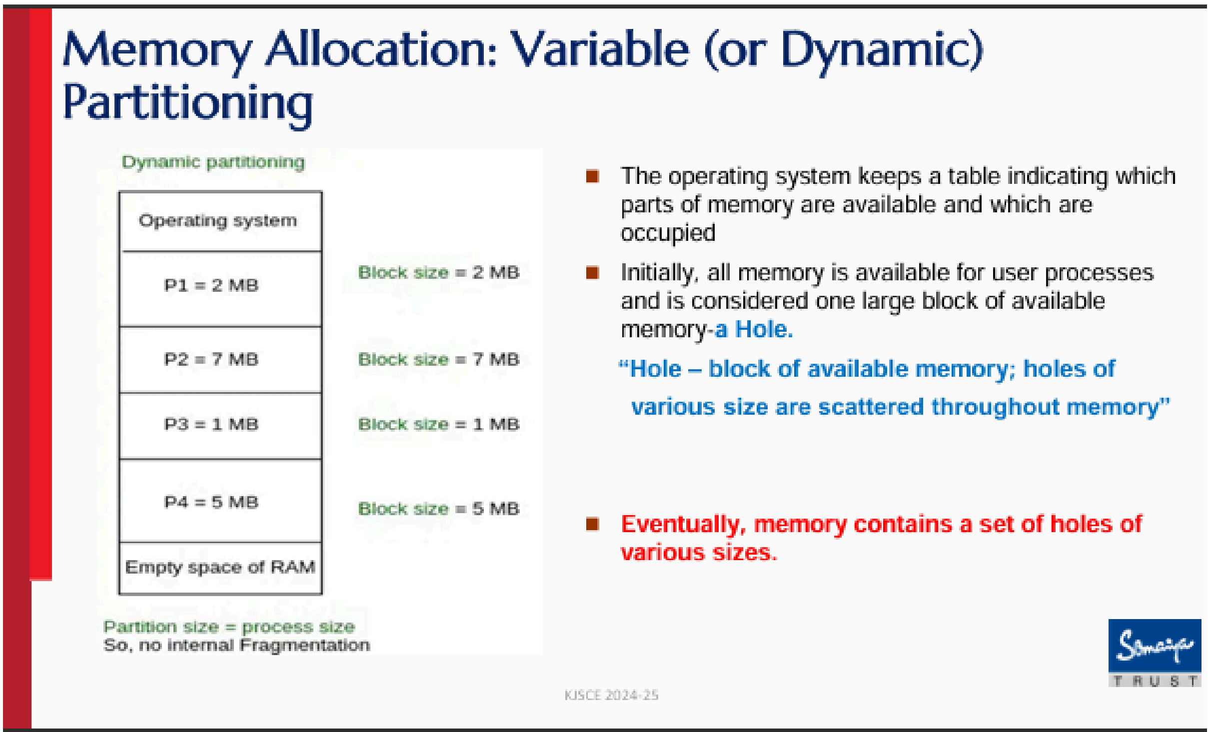
Memory Allocation: Fixed Partitioning

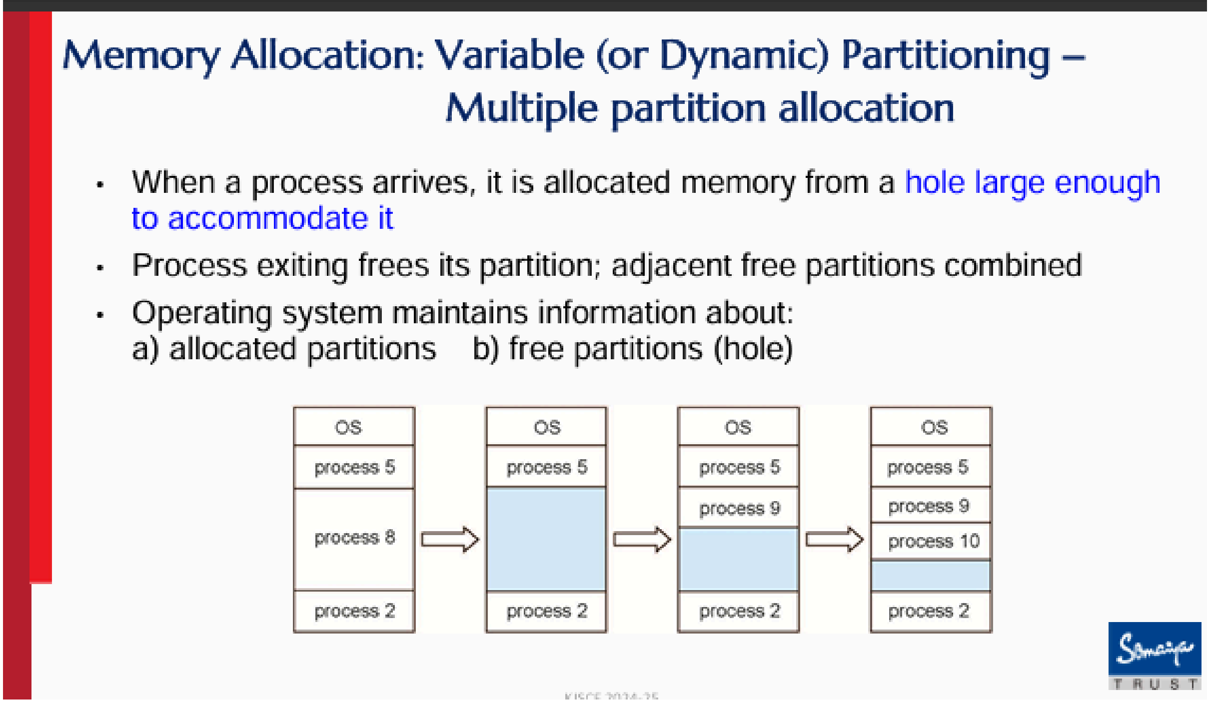
- As processes enter the system, they are put into an input queue
- At any given time, we have a list of available block sizes and an input queue
- The operating system can order the input queue according to a scheduling algorithm.
  - Process selected from input queue is allocated memory from a hole large enough to accommodate it
  - The operating system can wait until a large enough block is available,
  - or it can skip down the input queue to see whether the smaller memory requirements of some other process can be met

  - **Hole** – block of available memory

Memory Allocation: Variable (or Dynamic) Partitioning

- In contrast with fixed partitioning, partitions are not made before the execution or during system configuration
- Initially, memory is empty and partitions are made during the run-time according to the process's need instead of partitioning during system configuration
- Size of the partition will be equal to the incoming process
  - Variable-partition sizes for efficiency (sized to a given process' needs)
- The partition size varies according to the need of the process so that internal fragmentation can be avoided to ensure efficient utilization of RAM
- No of partitions are not fixed; depends on the number of incoming processes and the Main Memory's size





Memory Management – Notes (Slides 40-55)

Slide 40-42: Contiguous Memory Allocation

What is Contiguous Memory Allocation?

- A **process occupies a single continuous block** of memory.
- The entire process **must be loaded into RAM before execution**.

Types of Contiguous Memory Allocation:

1. **Fixed Partitioning (Static Allocation)**
   - Memory is divided into **fixed-size partitions** before execution.
   - Each partition holds **only one process**.
2. **Variable Partitioning (Dynamic Allocation)**
   - Partitions are created **dynamically at runtime** based on process size.
   - This reduces **internal fragmentation** but can cause **external fragmentation**.

Pros & Cons of Contiguous Allocation

Method	Advantages	Disadvantages
Fixed Partitioning	Simple & fast allocation	Internal fragmentation (wasted memory inside partitions)
Variable Partitioning	More flexible & efficient	External fragmentation (scattered free memory)

Slide 43-45: Memory Allocation Strategies

How is Memory Allocated to Processes?

Three major strategies are used:

Strategy	How it Works	Pros	Cons
First-Fit	Allocates the first available block that fits the process.	Fast & simple	May leave small holes (external fragmentation).
Best-Fit	Allocates the smallest available block that fits the process.	Reduces wasted space (less fragmentation).	**Slower**, requires searching all available blocks.
Worst-Fit	Allocates the largest available block.	Leaves large leftover space for future allocation.	**Wastes memory**, leads to **more fragmentation**.

Which Allocation Strategy is the Best?

- **First-Fit & Best-Fit are generally better than Worst-Fit.**
- **First-Fit** is **fastest** but may cause **fragmentation over time**.
- **Best-Fit** minimizes wasted space but is **slower** due to searching.

Slide 46-48: Fragmentation

What is Fragmentation?

- **Fragmentation occurs when memory is wasted due to inefficient allocation.**
- It reduces the efficiency of memory usage.

Types of Fragmentation

1. **Internal Fragmentation**
    - When a process **is allocated more memory than it needs**.
    - **Example:** A process needs **14 KB**, but is placed in a **16 KB partition → 2 KB is wasted**.
    - **Solution:** Use **dynamic allocation techniques** that allocate memory exactly as needed.
2. **External Fragmentation**
    - When **free memory exists, but it is scattered in small blocks**.
    - **Example:** There is **enough total free memory** for a process, but it is **not contiguous**.
    - **Solution: Compaction (shuffling memory to combine free blocks).**

Slide 49-50: 50% Rule & Compaction

What is the 50% Rule?

- **In First-Fit allocation,** about **1/3 of allocated memory is lost to fragmentation**.
- If **N memory blocks are allocated**, around **0.5N blocks remain fragmented and unusable**.

How Can We Reduce External Fragmentation?

1. **Compaction**
    - Moves processes **towards one end of memory**.
    - Gathers all **free memory into a single block**.
    - **Problem:** Requires CPU time & interrupts execution.
2. **Non-Contiguous Allocation**

    - Instead of requiring a single block, a process **is split into multiple smaller blocks across memory**.
    - Implemented using **Paging & Segmentation**.

Slide 51-52: Non-Contiguous Memory Allocation

What is Non-Contiguous Memory Allocation?

- Instead of requiring a **single continuous block**, a process **is allocated memory in multiple smaller blocks scattered throughout RAM**.
- **Used in Paging & Segmentation.**

Advantages of Non-Contiguous Allocation

✔ Eliminates **external fragmentation**.
✔ Allows **efficient use of memory**.
✔ Supports **dynamic memory allocation** (process size can change).

Disadvantages

❌ **Slower execution** (extra address translation step).
❌ **Increases CPU overhead** due to complex memory mapping.

Summary of Slides 40-55

Topic	Key Takeaways
Contiguous Allocation	Fixed or variable memory blocks.
Memory Allocation Strategies	First-Fit (fastest), Best-Fit (least waste), Worst-Fit (largest free block).
Fragmentation	Internal (wasted space inside blocks), External (scattered free space).
Compaction & Non-Contiguous Allocation	Combines free memory or allocates in multiple blocks.

https://drive.google.com/file/d/15mAbsuqp1w8pfRzBhYXgGEOay-Qzq1Wy/view?usp=drive_link

https://docs.google.com/presentation/d/1Mscy4a7TpoA_YiT_xf5kldIlq--c_gsa/edit?usp=sharing&ouid=114865698636231339398&rtpof=true&sd=true

https://drive.google.c om/drive/folders/19YtvLnayx0y5rbVhJ3p-i957xdHlenrJ?usp=sharing