# UNIT 4 - MEMORY MANAGEMENT.

Memory management ensures efficient use of computer's memory resources.

Importance ?

↳ Efficient Resource utilization.
↳ Process isolation & protection.
↳ Dynamic Allocation.
↳ Support for sharing.

Requirements ?

Relocation - As processes are swapped in & out of mem., physical location changes. So os must support relocation, to ensure consistency.

Protection - Its imp to protect each program's memory space so that one program can't accidentally / intentionally mess with another programs data.

Sharing - System should allow programs to share data when needed, but with the rules to keep everything secure.

Logical Organization - Memory should be organized such that it makes easy for programs to find what they need. This help in managing complex applications.

Physical Organization - System needs to handle both main & secondary memory storage efficiently; ensuring smooth operation for programs.

Main mem → fast access → volatile mem → high cost
secondary mem → slow access → non-volatile → low cost

volatile ? temporary storage & when off all data lost.

## Memory Partitioning

The way that computers manage their memory by dividing it into small sections / partitions.
This helps the computer allocate memory to different programs that are running at the same time.

| Fixed Partitioning | Dynamic Partitioning |
|---|---|
| Memory is divided into no. of fixed size section. | Memory is divided based on how much each program needs. |
| Each program gets a specific section / amount of memory. | Programs get exactly the space they need; no more, no less. ↳(memory) |
| Can waste space if program doesn't fill its section (called internal fragmentation). | Can create small gaps in memory over time (called external fragmentation). |
| Limited flexibility ie the no. of programs that can run is fixed by no. of sections. | More flexible ie can run more programs based on their memory needs. |
| Easy & simple to set up. | More complex to manage. |

A program (size) cannot be larger than the biggest section.

Can waste memory since not all space may be used ie. less efficient.

Low efforts to manage as sections are fixed

No strict size limit; programs can be any size if there is enough memory.

More efficient; uses memory better since it allocates only what is needed.

Higher efforts needed to keep track of different sized sections & gaps.

Internal fragmentation

Wasted spaces within an allocated memory block.

Occurs when memory blocks are larger than what a program needs.

Eg. If a program needs 3MB but get a 4MB block, 1 MB is wasted

This leads to inefficient use of memory within blocks.

Hard to eliminate; can be reduced by using smaller blocks / dynamic allocations.

How to Fix

Use contiguous allocation with sized that closely match needs

External fragmentation

Wasted space outside allocated blocks due to scattered free areas

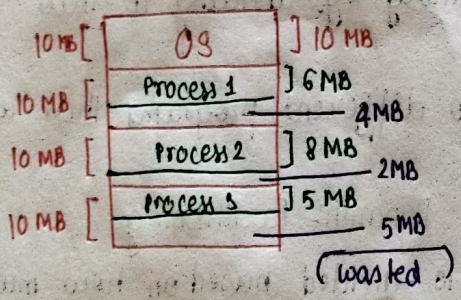
Occurs when programs are loaded & removed, leaving small gaps.

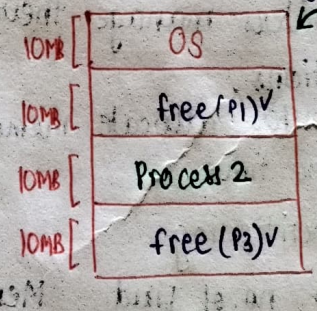
Eg. If there are free spaces of 4MB & 2MB, but new program needs 5MB, it can't fit.

Can stop new programs from starting even if there is enough total free memory.

Can be managed by moving things around (compaction) to create large free space.

Use paging or segmentation to manage memory more flexibly.




considering P1 & 3 completed their work

Now P4 of 11MB is in memory. Even if 20 MB is free, process 4 cannot be loaded into memory. Because we need contiguous block of 11 MB to load P4.

Compaction

↳ reduces external fragmentation.

↳ main reason for compaction is to create larger blocks of free memory so that new programs can fit into memory without problems.

↳ computer look at all memory being used & all free spaces → moves program that are currently running closer together, which creates one big free space → this moving process ensures that everything still works correctly after being rearranged.

↳ Adv - reduce wasted spaces in mem

Disadv - takes time; might cause delays & issues.

Eg:

| Book A | Free | Book B | Free | Book C | Free |
|---|---|---|---|---|---|

Before

After →

| Book A | Book B | Book C | Free |
|---|---|---|---|

→ very big space

## ** Buddy System

It is a memory allocation technique used in os to manage memory efficiently.

What is Buddy System?

↳ This system divides memory into block of sizes that are powers of 2 (eg. 16KB, 32KB, 64 KB)

↳ When a program request memory, the system searches for smallest available block that can fit the request.

↳ If a suitable block is found, it is allocated.
If not, a larger block is split into 2 smaller "buddies" until appropriate size is obtained.

↳ When memory is freed, the system checks if the buddy block is also free.
If it is, they are merged back into larger block, reducing fragmentation.

How it Work?

① Memory Division - dividing mem into $2^n$.

② Block Allocation - when process req ⇒ look for smallest block that can accomodate req.
⇒ If not match, split to appropriate size until match occur

③ Block Deallocation
Process release memory ⇒ check buddy block is free ⇒ yes, merge back into larger block

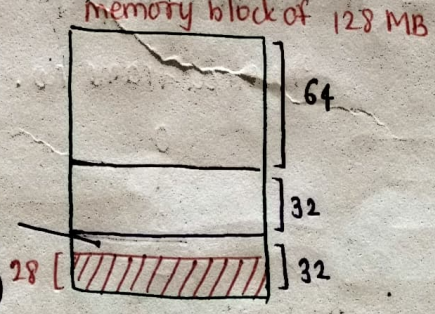
Advantages

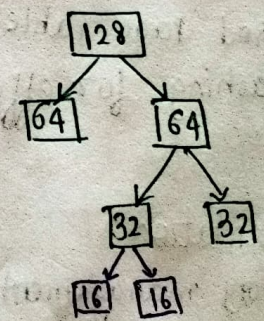
↳ Efficient memory use
↳ Fast Allocation/Deallocation
↳ Easy implementation
↳ Ease of coalescing - free blocks merge when adj mem. blocks are freed.

Disadvantages

↳ Internal fragmentation
↳ Power of 2 limitation
↳ Extra Book-keeping → requires to maintain info about free blocks & their sizes.

Eg. fit 28 MB




## Relocation

Refers to the ability to move processes within memory during execution. This helps in compaction & reduce fragmentation.

Eg. If P1 want A (add 1000)
If P1 is loaded at 2000, as will adjust all references of A from 1000 to 3000 (2000+1000)

## Paging

↳ Divides the computers memory into fixed size of blocks called pages in logical memory & frames in physical memory.
↳ This eliminates need of contiguous allocation of memory, as programs can be loaded into any available frame.

Advantages
↳ Efficient memory use ↳ support VM ↳ protection & sharing

Disadvantages
↳ overhead/overload ↳ external fragmentation ↳ Page faults.

### Page Table Structure

↳ crucial data structure used in OS to manage VM.
↳ It maps/translates virtual address to physical address in system using VM

Types of Page Tables

① Single-Level page table -
each process has a single page table that directly maps virtual pages to physical frames

| Virtual Page No. | frame No. |
|---|---|
| 0 | 5 |
| 1 | 2 |
| 2 | 0 |
| 3 | 3 |

② Multilevel (Hierarchical) Page Table -
↳ Breaks down page-table into multiple levels to save space.

Root Page Table (P1)
↓
Sub Page Table (P2)
↓
frame no

③ Inverted Page Table -
↳ instead of having 1 page table/process, this uses a single table that contains one entry for each physical fame, mapping it to virtual pages

| Physical frame no. | V page No & Process ID |
|---|---|
| 0 | VPN:2, PID:A |
| 1 | VPN:1, PID:B |
| 2 | VPN:0, PID:A |
| 3 | VPN:3, PID:C |

④ Hashed Page Table -
uses hashing for efficient lookups

## Segmentation

↳ memory management technique used, that divides memory into variable-sized segments.
↳ Each segment represents a logical part of program such as
Code segment - contains executable code
Data segment - holds global & static variables
Static segment - used for function calls & local variables

## How Address Translation is Performed in Segmentation System.

Address translation involves converting logical addresses into physical addresses.
This process uses a segment table, which contains information about each segment.

Steps - ① Logical Address Structure : this consist of 2 parts
Segment Number - Identify which segment is being accessed.
offset - specifies the exact location within that segment.

② Using the Segment Table : Each process has segment table that includes -
Base Address - starting physical address of segment in memory.
Limit (Size) - length of segment, which ensures that access stay within bounds.

③ Translation Process : When program generates logical address, os performs
(i) Extract segment no. & offset from logical address.
(ii) Look up the segment table to find base add$^r$. and limit.
(iii) Check if offset is within limit. If exceeds, it raises error (seg. fault)
(iv) Calculate the physical add$^r$ by adding base add$^r$ of offset.

Eg. We have : seg. no = 2
offset = 150

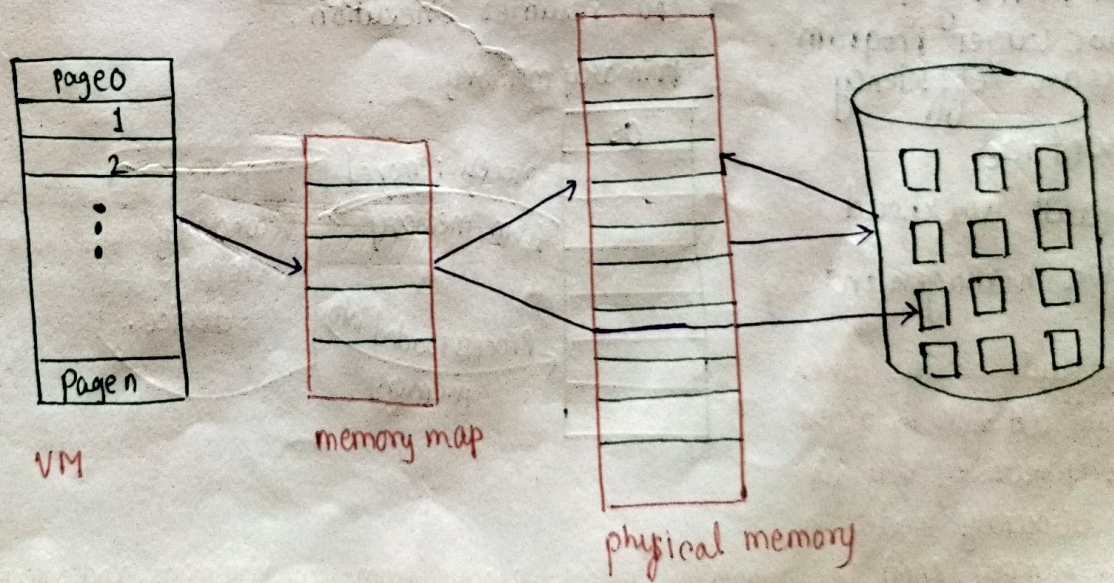
Sample seg. table

| seg. no. | Base add$^r$ | Limit |
|---|---|---|
| 0 | 1000 | 200 |
| 1 | 1200 | 150 |
| 2 | 1400 | 300 |

(i) look up seg 2
Base add$^r$ = 1400 & Limit = 300
(ii) Check if offset (150) is within Limit (300). It is ✓
(iii) Physical Add$^r$ = Base Add$^r$ + Offset
= 1400 + 150
= 1550.

## VIRTUAL MEMORY

VM is a memory management technique that allows a computer to use secondary storage as if it were part of main memory (RAM).
This creates illusion of a larger amount of RAM than is physically available.



# Demand Paging

↳ memory management process/technique used by os to optimize use of memory resources.

↳ Demand paging is used in VM, where the pages are brought in main memory only when required/demanded by CPU.

↳ Hence it is called "lazy swapper" because swapping of pages is done only when CPU requires it

↳ In demand paging, the pager ~~ob~~ brings only those necessary pages into the memory instead of swapping in a whole process.

## How Demand Paging works?

① Initial Load → when program starts, only some of its pages are loaded into RAM, rest all stay in HD (secondary mem)

② Page Table → this tells which pages are currently in mem (valid) & which are not (invalid)

③ Accessing Pages → when CPU tries to access page:
If page is valid (ie in mem), it can be used right away.
If page is invalid, a page fault occurs.

④ Handling page faults → OS takes control & checks if the required page exists on hard drive.
If it does, OS loads page into RAM & updates page table.
If it does not exist, the process may be terminated.

⑤ Resume Execution → After loading the needed page, the program continues running as if nothing happened.

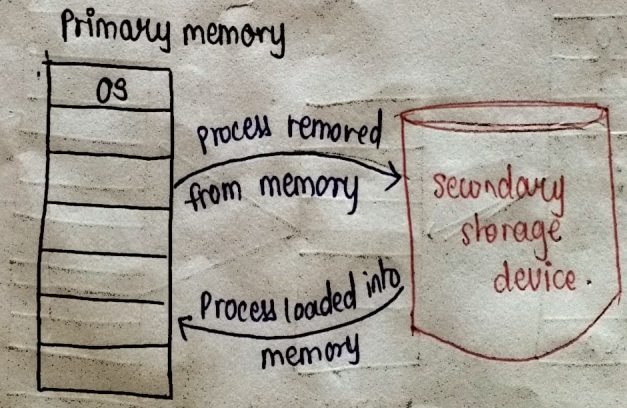
Eg. 4 pages - P0, P1, P2, P3. At first only P1 & P3 are loaded into RAM
If CPU wants to access P2: → It checks the page table
→ Since P2 is not in RAM, page fault occurs.
→ OS finds P2 on HD & loads into RAM.
→ Page table updates & now P2 is in memory
→ CPU resumes execution.

### Advantages

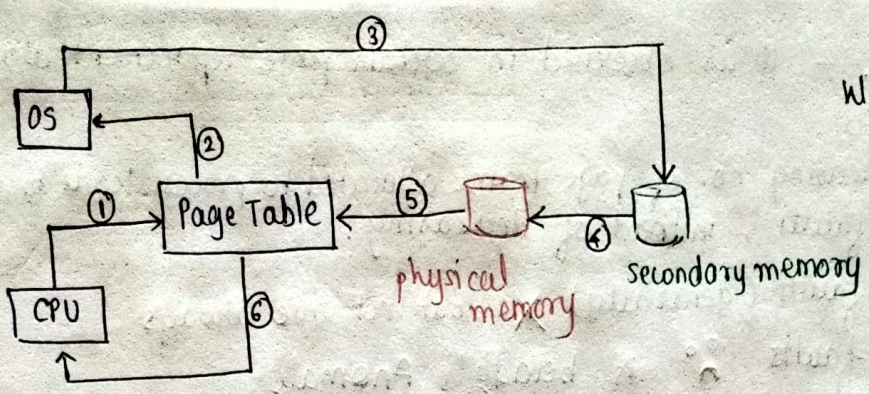
↳ Saves Memory.
↳ Allows Larger Program.
↳ Increase Efficiency.

### Disadvantages

↳ Page fault overhead.
↳ Complex management.




A page fault occurs when program attempts to access a page of memory that is not currently loaded in physical memory (RAM). This triggers OS to retrive page from secondary mem, and allow execution.




Working procedure

Page Replacement Algorithms: (Numericals only)

When physical memory is full & a new page must be loaded, the os must replace an existing page.

① FIFO
first loaded, first removed
~~(stack)~~ (queue)

② LRU (Least Recently used)
replace the page that has not been used for longest time.

③ optimal
replace the page that will not be used for longest period in future

Allocation of frames.

Process by which an os assigns fixed size blocks of physical memory (called frames) to different processes.

This ensures that each process has enough memory to operate effectively while optimizing the use of available RAM.

① fixed allocation
fixed no of frames

② Dynamic Allocation
frames are dynamically allocated

Thrashing

Occurs when system spends more time swapping pages in & out of memory than executing process.

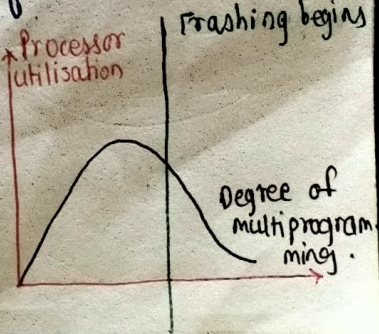
This happens when there is insufficient memory to hold the working sets of all active processes.

The system becomes overwhelmed by page fault, significantly reducing performance

Solution to trashing

① working set model - ensure sufficient frames are allocated or not.

② Page fault frequency - allocate frame acc. to page fault.
page fault ↑ frames ↑



Belady's Anomaly. - It is observed in certain page replacement algo, particularly in FIFO.
It occur when increasing no. of page frame allocated to process leads to increase in page faults, instead of decreasing.

ie ↑ frames = ↓ faults (generally) ✗ but not true always
↑ frames = ↑ faults ∴ is belady's Anomaly

Eg. Reference string : 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

Case 1: FIFO with 3 frames gives 9 page faults

Case 2: FIFO with 4 frames gives 10 page faults

Q) What are the distinctions among logical, relative & physical addresses?

| Logical | Physical | Relative |
|---|---|---|
| Generated by CPU (address) | Actual address in RAM where data is stored. | address that is based on, starting point. |
| visibility: seen by users & programs | Not seen by users; used by computer | depends on context; used in programming. |
| used by programs to access memory | used by computer to find data in memory. | used for jumps & offsets in code. |
| Needs to be converted to access physical addr. | accessed directly by hardware. | Converted to logical/ physical addr based on where it is used. |
| Memory space: Represents the programs virtual mem. space | Represents actual mem space in computer | Represents how far from a base addr something is, not an exact location |

Memory Allocation Strategies | Algorithms.

**First fit**
allocates first available block that fits the request

**Best fit**
Allocates smallest available block that fits the request

**Worst fit**
Allocates the largest available block for the request

**Next fit**
Similar to first fit but starts searching from the last allocated block.