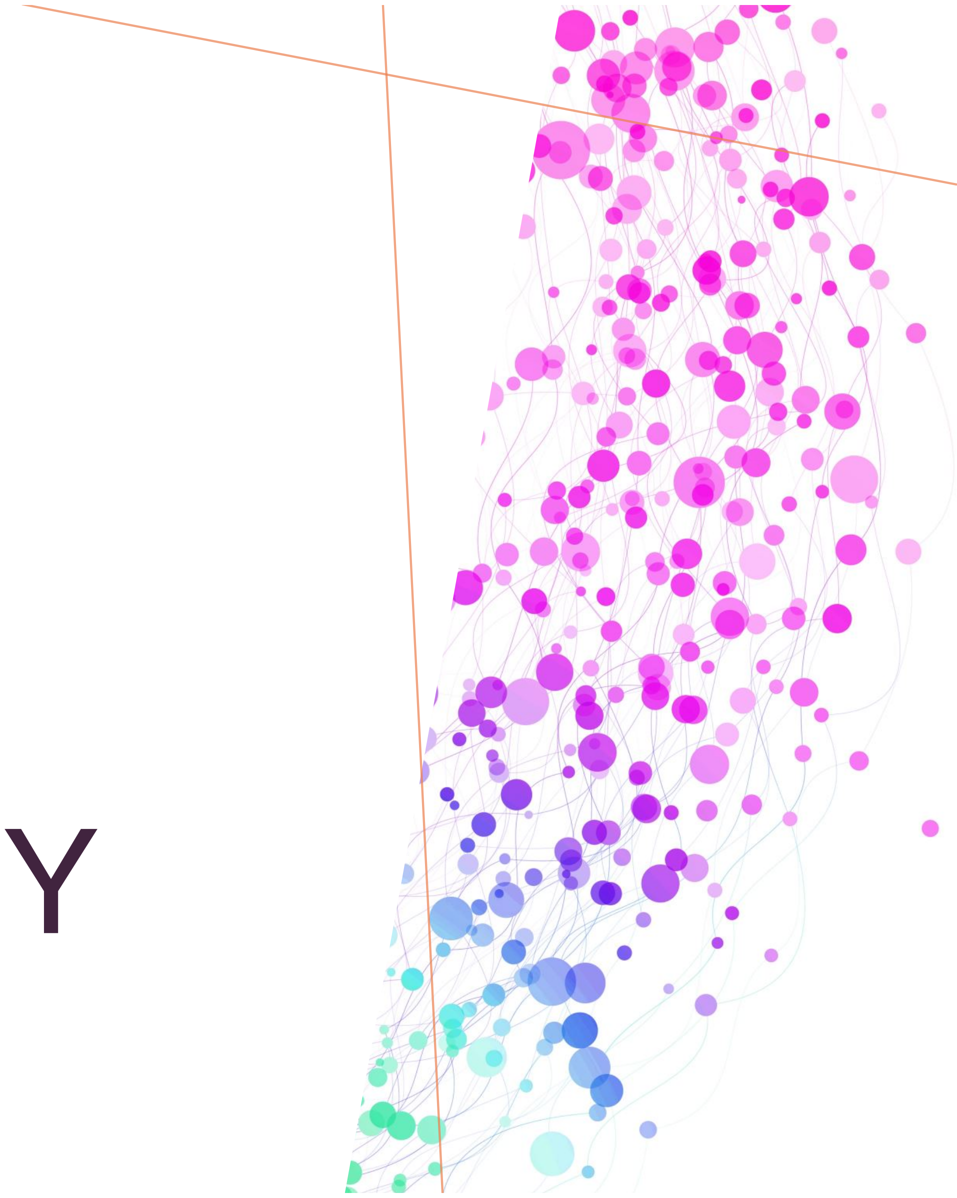
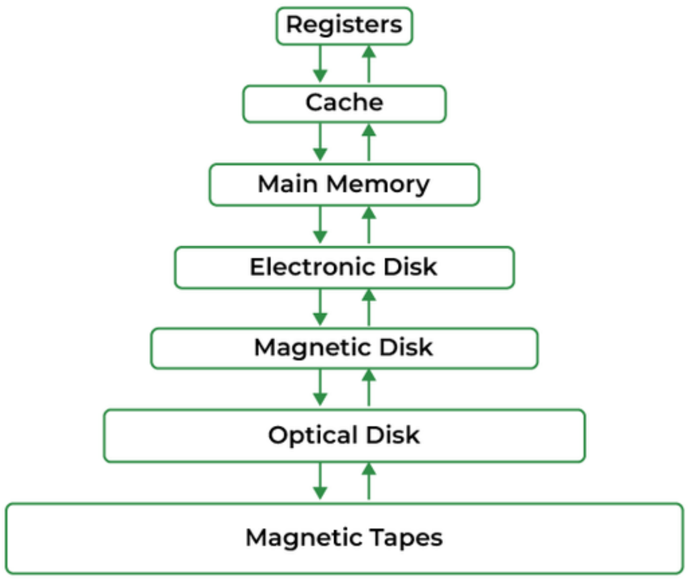
MAIN MEMORY



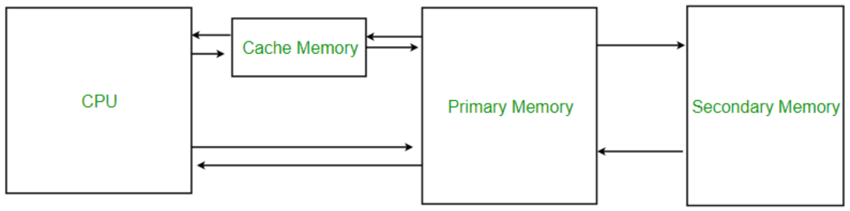
WHAT IS MAIN MEMORY?

- **Central Role**: Key to computer operation.
- **Structure**: Large array of words/bytes.
- **Function**: Rapidly accessible info for CPU and I/O devices.
- **Usage**: Stores active programs and data.
- **Speed**: Fast data transfer with the processor.
- **Also Known As**: RAM (Random Access Memory).
- **Volatility**: Loses data on power loss.



CACHE

- **High-Speed Memory**: Special, very fast memory.
- **Smaller & Faster**: Stores copies of frequently used main memory data.
- **Multiple Caches**: Independent caches in a CPU for instructions and data.
- **Purpose**: Reduces average time to access data from main memory.

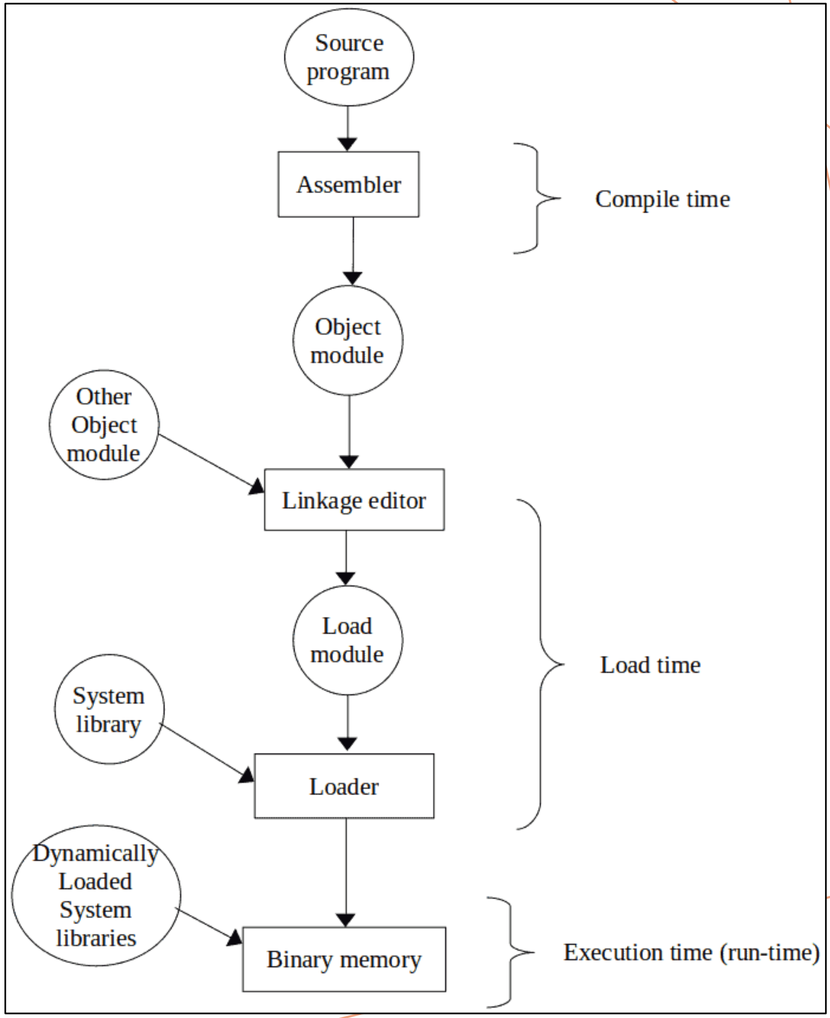


LOGICAL AND PHYSICAL ADDRESS SPACE

- **Logical Address Space**
  - Generated by the CPU
  - Also called a Virtual address
  - Defines the size of the process
  - Can be changed
- **Physical Address Space**
  - Seen by the memory unit
  - Also called a Real address
  - Computed by the Memory Management Unit (MMU)
  - Remains constant
  - Maps from logical addresses

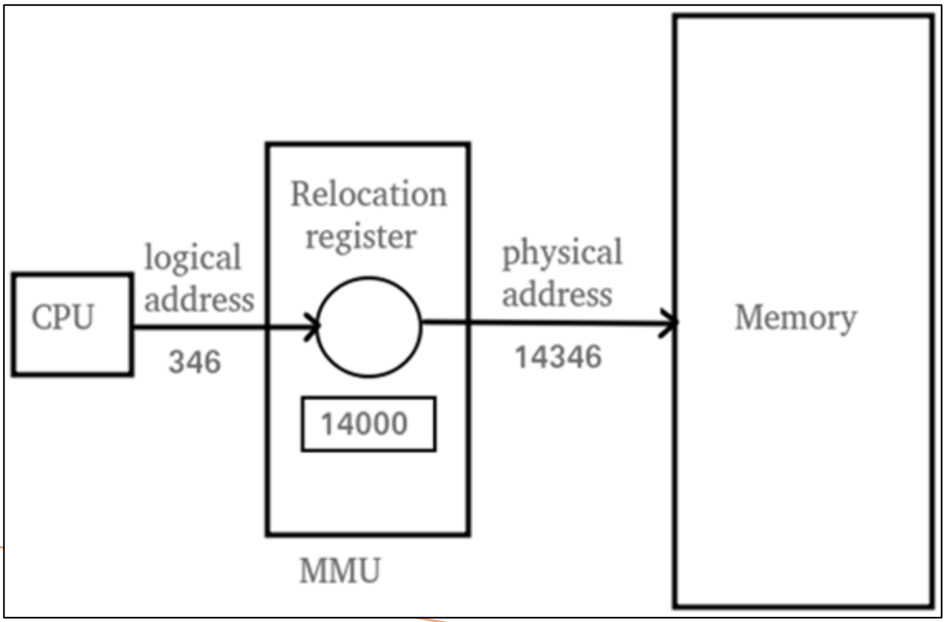
ADDRESS BINDING

- Compiled code addresses bind to relocatable addresses
- Can happen at three stages: Compile time, Load time, Execution time



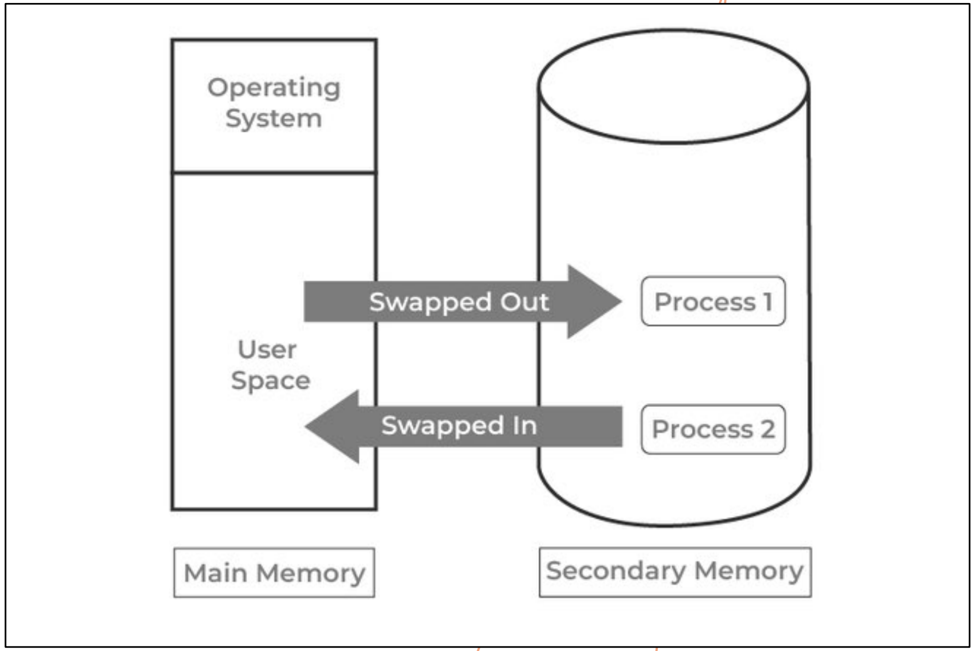
MEMORY-MANAGEMENT UNIT (MMU)

- Maps virtual to physical address
- Simple scheme: relocation register adds base value to address



SWAPPING

- Allows total physical memory space of processes to exceed physical memory
- Process swapped out temporarily to backing store, then brought back for execution



BACKING STORE AND SWAPPING VARIANTS

Backing Store

- **Fast disk** for storing copies of all memory images.
- **Function**: Holds overflow data from RAM.

Roll Out, Roll In

- **Swapping variant** for priority-based scheduling.
- **Roll Out**: Moves low-priority processes to backing store.
- **Roll In**: Loads high-priority processes into main memory.

DYNAMIC STORAGE-ALLOCATION PROBLEM

First-Fit

- **Method**: First available block.
- **Pro**: Fast.
- **Con**: Fragmentation.

Best-Fit

- **Method**: Smallest suitable block.
- **Pro**: Minimizes wasted space.
- **Con**: Slow, small fragments.

Worst-Fit

- **Method**: Largest available block.
- **Pro**: Reduces small fragments.
- **Con**: Inefficient use of memory.

Example

Consider a swapping system in which memory consists of the following whole sizes in memory order: 10K, 4k, 20k, 18k, 7k, 9k, 12k, and 15k. Which hole is taken for successive segment request of i)12k, ii)10k, iii)9k for first fit? Now repeat the question for best fit and worst fit.

First Fit

12k	→	20k
10k	→	10k
9k	→	18k

Best Fit

12k	→	12k
10k	→	10k
9k	→	9k

Worst Fit

12k	→	20k
10k	→	18k
9k	→	15k

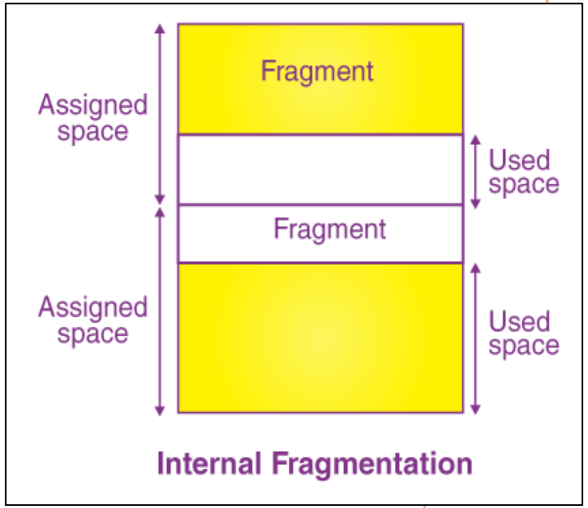
FRAGMENTATION

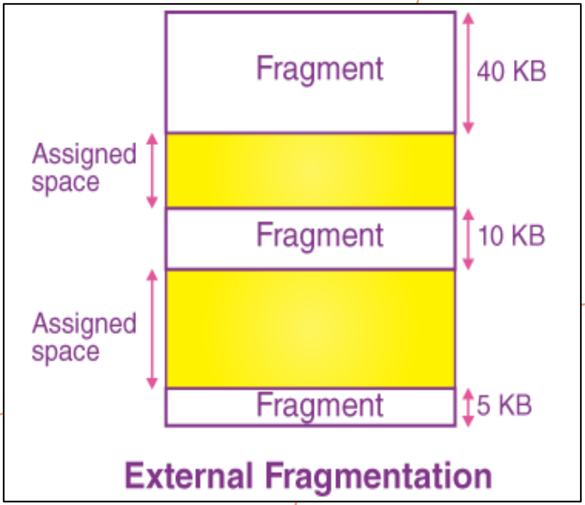
External Fragmentation

- **Definition**: Non-contiguous free memory blocks.
- **Cause**: Allocation and deallocation of memory over time.
- **Problem**: Insufficient continuous space for new allocations despite total free space being adequate.

Internal Fragmentation

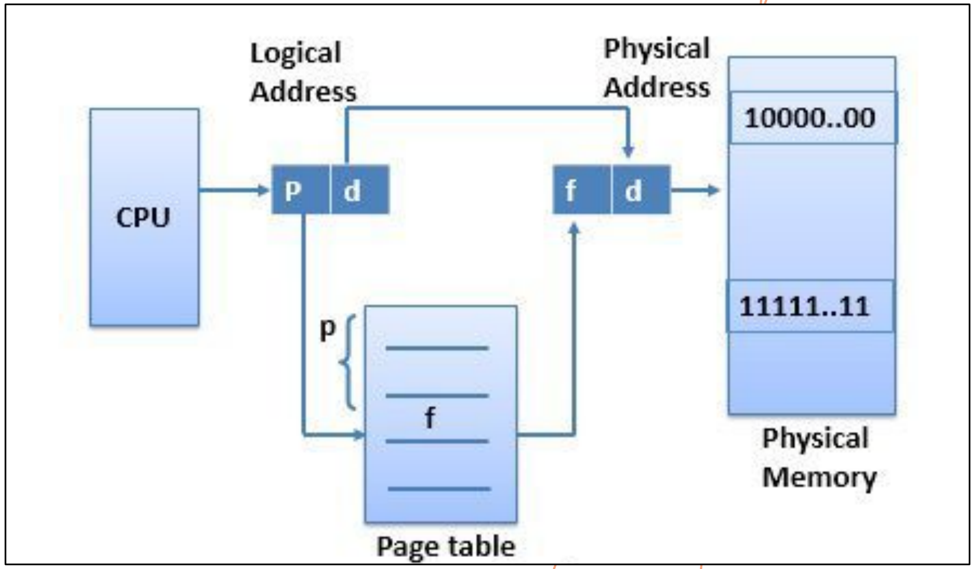
- **Definition**: Allocated memory slightly larger than requested.
- **Cause**: Fixed-sized memory blocks.
- **Problem**: Wasted space within allocated regions.





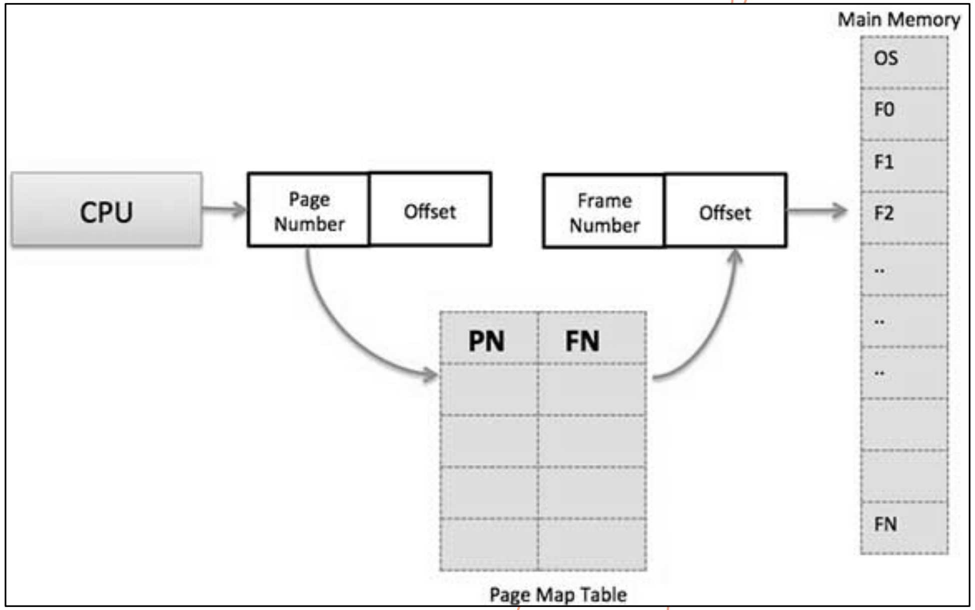
PAGING

- Physical memory divided into fixed-sized frames
- Logical memory divided into pages
- Page table translates logical to physical addresses



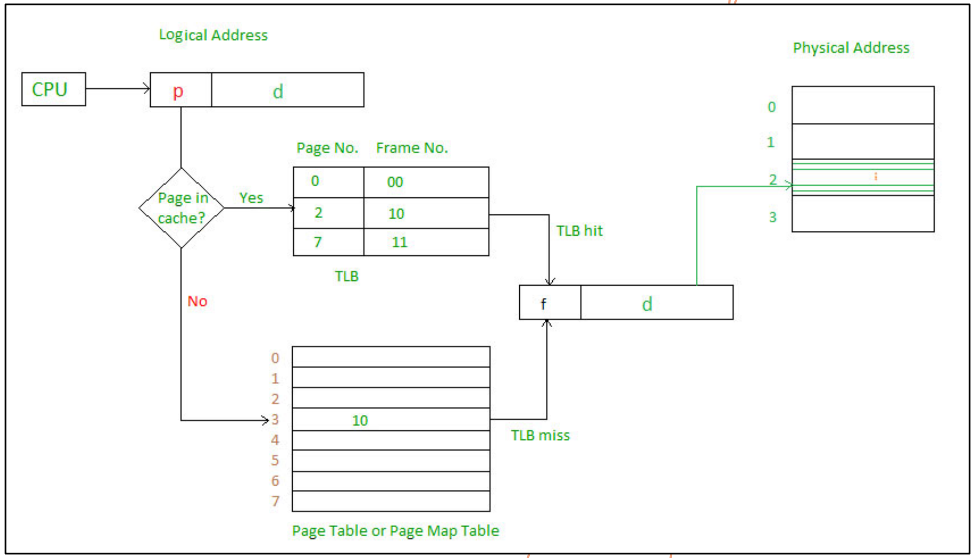
ADDRESS TRANSLATION

- **Page Number (p)**:
  - Identifies the page in memory.
- **Page Offset (d)**:
  - Specifies the position within the page.
- **Address Translation**:
  - Uses page number and offset to locate data in memory.



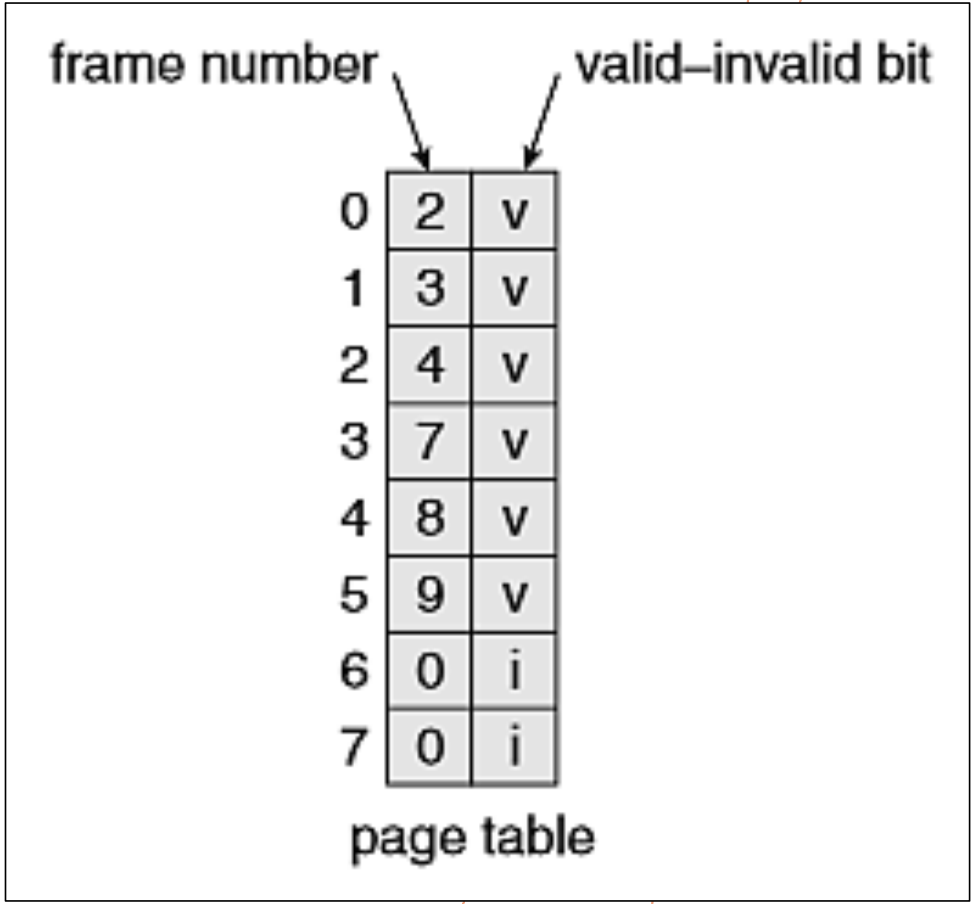
TRANSITION LOOK-ASIDE BUFFER (TLB)

- **CPU cache** for fast virtual address translation.
- **Features**:
  - Small size
  - Associative mapping
  - Loads data on TLB miss.



PAGE TABLE PROTECTION

- **Components**:
  - **Valid bit**: Indicates if the mapping is valid.
  - **Invalid bit**: Marks the mapping as invalid.
- **Purpose**:
  - Protects memory regions from unauthorized access.
  - Ensures data integrity by controlling access permissions.



MULTILEVEL AND HASHED PAGE TABLES

- **Multilevel Page Tables**:
  - Hierarchical organization for efficient memory management.
  - Divides large address spaces into smaller, manageable units.
- **Hashed Page Tables**:
  - Virtual page number hashed into page table for faster access.
  - Offers a more direct mapping approach, reducing lookup time.

SEGMENT TABLE

- **Function**:
  - Maps two-dimensional physical addresses efficiently.
  - Organizes memory in a structured manner for easy access.
- **Protected Entries**:
  - Valid bits and access privileges (read/write/execute) safeguard entries.
  - Ensures data integrity and security by controlling access permissions.

MEMORY ADDRESS DECOMPOSITION

Page Number (PN):

- Definition: Indicates the page where the data is stored.

Formula:

$$\text{Page number} = \left\lfloor \frac{\text{Memory Address}}{\text{Page Size}} \right\rfloor$$

Offset (O):

- Definition: Specifies the position of the data within the page.

Formula:

$$\text{Offset} = \text{Memory Address} \mod \text{Page Size}$$

EXAMPLE

1. Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a) 3085
b) 42095
c) 215201

ADDRESS: 3085

Step 1: Calculate Page Number

- Page number=3085 / 1024=3

Step 2: Calculate Offset

- Offset=3085 mod 1024=3085−(1024×⌊3085 / 1024⌋)=13

ADDRESS: 42095

Step 1: Calculate Page Number

- Page number = 42095 / 1024 = 41

Step 2: Calculate Offset

- Offset = 42095 mod 1024 = $42095 - (1024 \times \lfloor 42095/1024 \rfloor)$ = 111

ADDRESS: 215201

Step 1: Calculate Page Number

- Page number = 215201 / 1024 = 210

Step 2: Calculate Offset

- Offset = 215201 mod 1024=215201−(1024×⌊215201/1024⌋) = 161

EXAMPLE TRANSLATING LOGICAL ADDRESSES

•To solve the memory management exercise, start by determining the page size and identifying the page number and offset for each address. Then, refer to the provided page/frame table to translate logical addresses into physical addresses, using the frame's base address and offset. If an entry for the page number doesn't exist in the table, mark it as a page fault. Repeat this process for all given addresses, ensuring accuracy in calculations and entries. (**<u>Page size:</u>** 0.75 KB (768 bytes)).

•Addresses are:

1) 0x3A7
2) 0x1FFC
3) 0x14D3

- Note that the page table has two columns, the page number in decimal, and the frame's base address in hexadecimal.

PAGE TABLE

Page # (dec)	Frame’s base address
1	0x100
3	0x600
15	0x900
20	0xAA00
32	0x3E500

SOLUTION

Address (Hex)	Address (Dec)	Page # (Dec)	Frame's Base (Hex)	Offset (Hex)	Physical Address (Hex)	Page Fault
0x3A7	935	935 / 768 = 1	0x100	935 % 768 = 167 (0xA7)	0x1A7	No
0x1FFC	8188	10	-	0xFC	-	Yes
0x14D3	5331	6	-	0xD3	-	Yes

ACTIVITY TRANSLATING LOGICAL ADDRESSES

•To solve the memory management exercise, start by determining the page size and identifying the page number and offset for each address. Then, refer to the provided page/frame table to translate logical addresses into physical addresses, using the frame's base address and offset. If an entry for the page number doesn't exist in the table, mark it as a page fault. Repeat this process for all given addresses, ensuring accuracy in calculations and entries. (**<u>Page size:</u>** 0.75 KB (768 bytes)).

•Addresses are:

1) 0x6B3
2) 0x12FF
3) 0x60B4

- Note that the page table has two columns, the page number in decimal, and the frame's base address in hexadecimal.

PAGE TABLE

Page # (dec)	Frame’s base address
1	0x100
3	0x600
15	0x900
20	0xAA00
32	0x3E500

SOLUTION

Address (Hex)	Address (Dec)	Page # (Dec)	Frame's Base (Hex)	Offset (Hex)	Physical Address (Hex)	Page Fault
0x6B3	1715	2	-	0xB3	-	Yes
0x12FF	4863	6	-	0xFF	-	Yes
0x60B4	24756	32	0x3E500	0xB4	0X3E5B4	No