SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

Department of Computer Science and Engineering

23CST206–OPERATING SYSTEMS AND VIRTUALIZATION

B.E- CSE /IV SEMESTER

UNIT – III MEMORY MANAGEMENT

Topic 2: Segmentation

Segmentation

Memory management technique dividing programs into logical segments

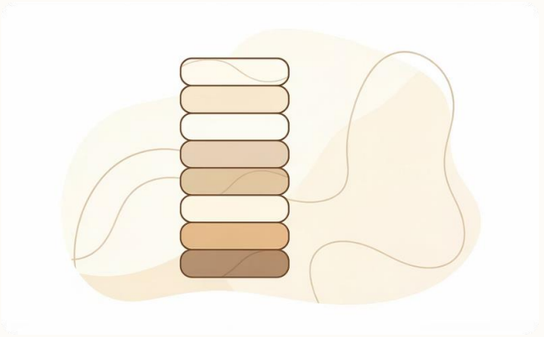
DEFINITION

What is Segmentation?

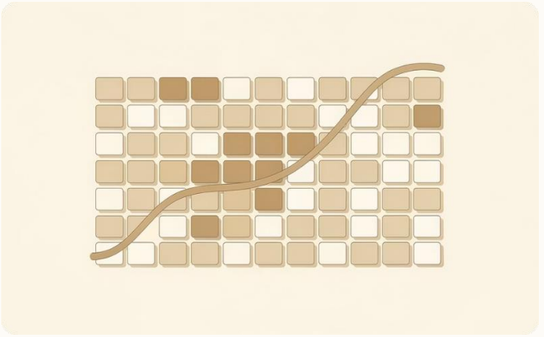
Memory management technique dividing programs into logical segments based on programmer's view.

Code segment

Data segment



Stack segment



Heap segment

Each segment can vary in size.

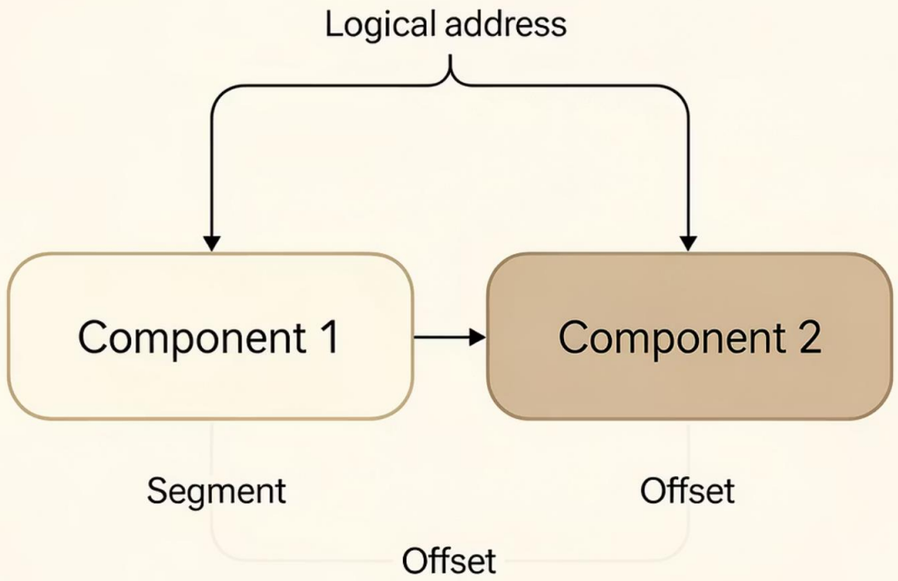
Logical Address Format

Segment Number

Identifies which segment

Offset

Displacement within segment



⟨ Segment Number , Offset ⟩

Segment Table

Structure

Each process has a segment table

Each entry contains:

Base Address

Starting physical address of segment

Limit

Length of the segment

How Segmentation Works

01

Generate Address

CPU generates logical address ⟨segment number, offset⟩

02

Index Table

Segment number indexes the segment table

03

Check Limit

Offset checked with limit. If offset > limit → segmentation fault

04

Calculate Address

Physical address = base + offset

Advantages of Segmentation

Logical View

Protection

No Internal Fragmentation

Flexibility

Disadvantages of Segmentation

External Fragmentation

Complex Memory Management

Increased Overhead

Protection Granularity

Segmentation with Paging

Hybrid technique combining advantages of both segmentation and paging

How it Works

1. **Divide into Segments**

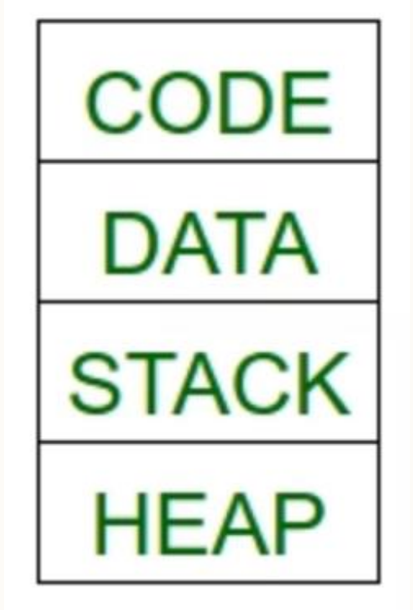
   Process divided into code, data, stack, heap

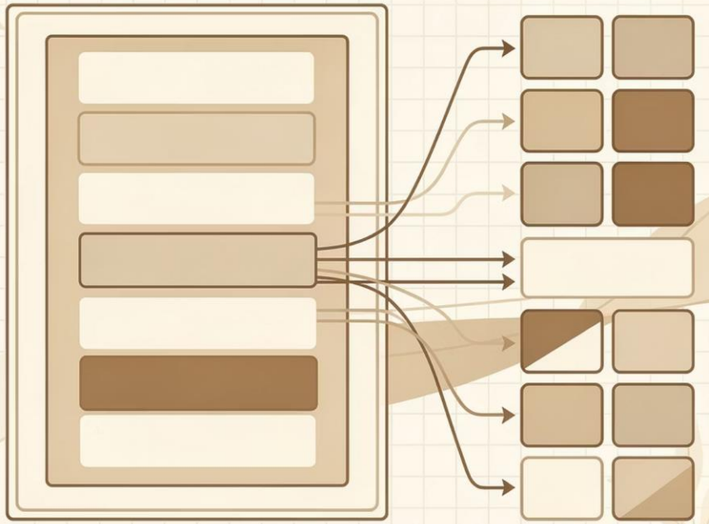
2. **Divide into Pages**

   Each segment divided into fixed-size pages

3. **Page Tables**

   Each segment has own page table



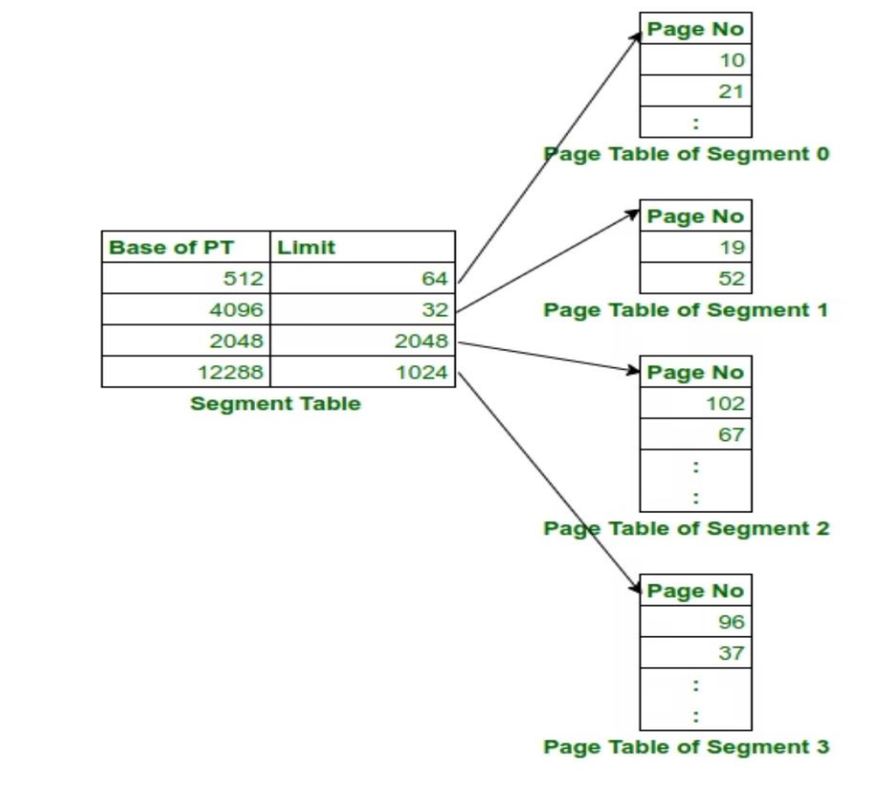


Address Translation Process

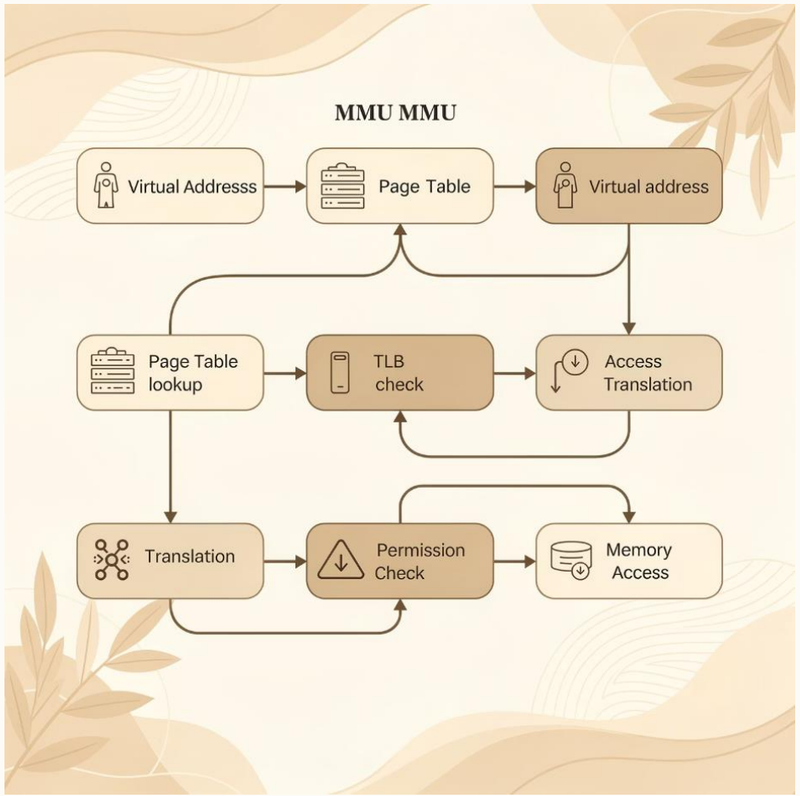
CPU Logical Address Structure

MMU checks segment table → finds page table base → uses page number to locate frame → combines with offset for physical address

Segmented Paging



Complete Workflow



Pros & Cons

Advantages

- Reduced page table size and memory requirements
- Programmer's view with paging benefits
- Reduces external fragmentation vs segmentation

Disadvantages

- Internal fragmentation still exists in pages
- Extra hardware required
- Sequential translation increases memory access time

WORKFLOW

Segmented Paging Workflow

Hybrid memory management: program divided into segments, each segment divided into pages

Steps 1-2: Program Division

1

Divide Program into Segments

Logical division:

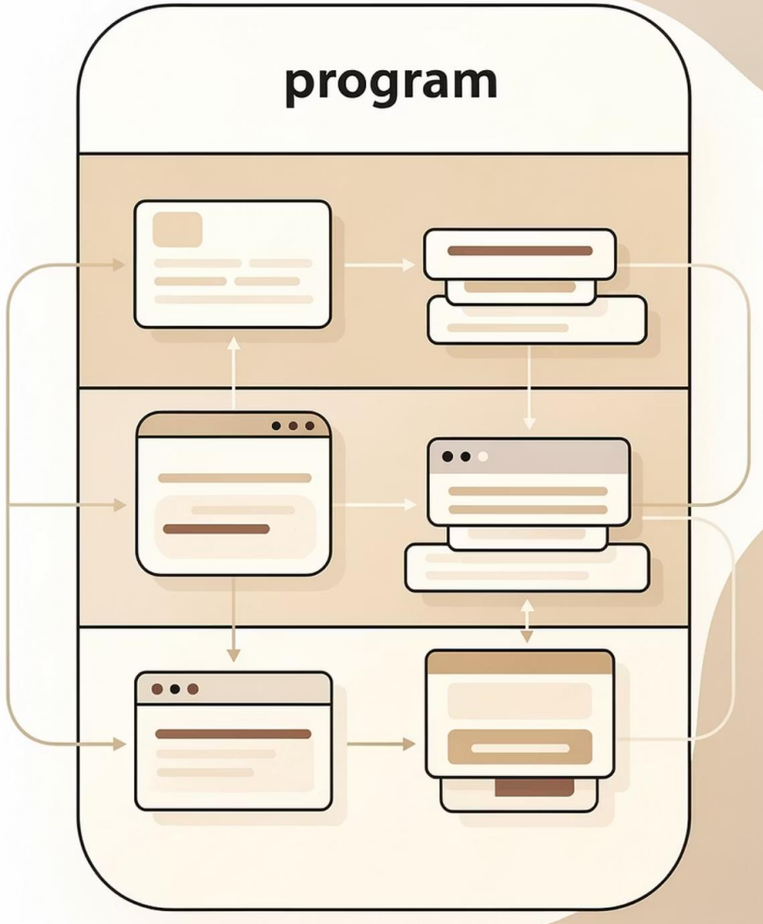
- Code segment
- Data segment
- Stack segment

Each segment is variable size

Divide Each Segment into Pages

Each segment divided into fixed-size pages

Physical memory divided into frames of same size



Steps 3-6: Table Creation & Lookup

1

Step 3: Create Segment Table

OS maintains segment table for each process. Entries contain base address of page table and segment length

2

Step 4: Create Page Tables

Each segment has own page table mapping pages to frames

3

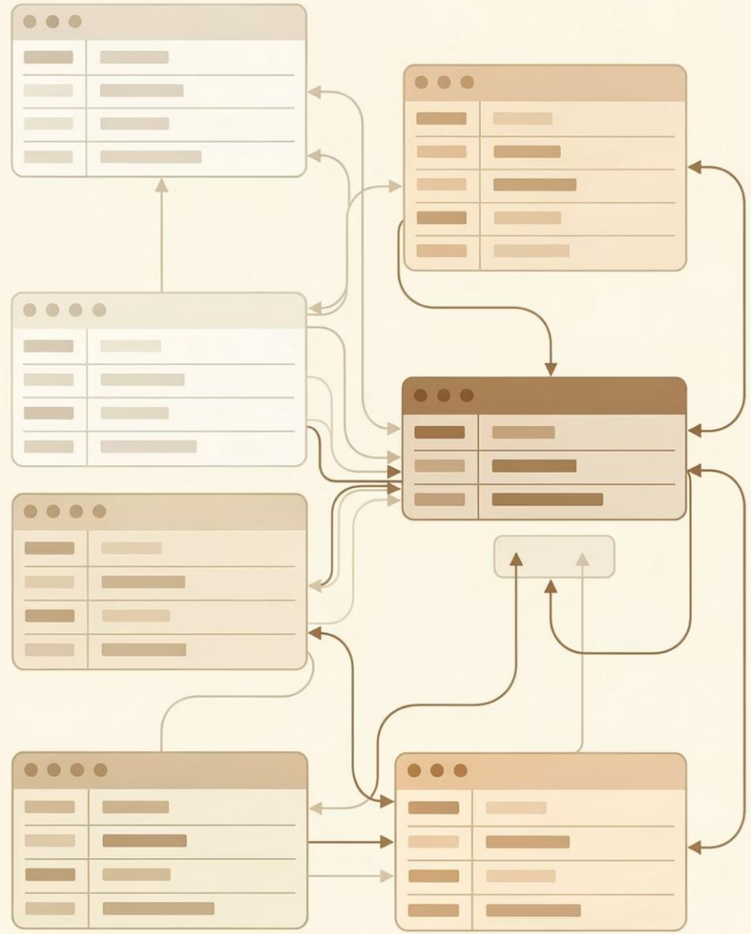
Step 5: Logical Address Generation

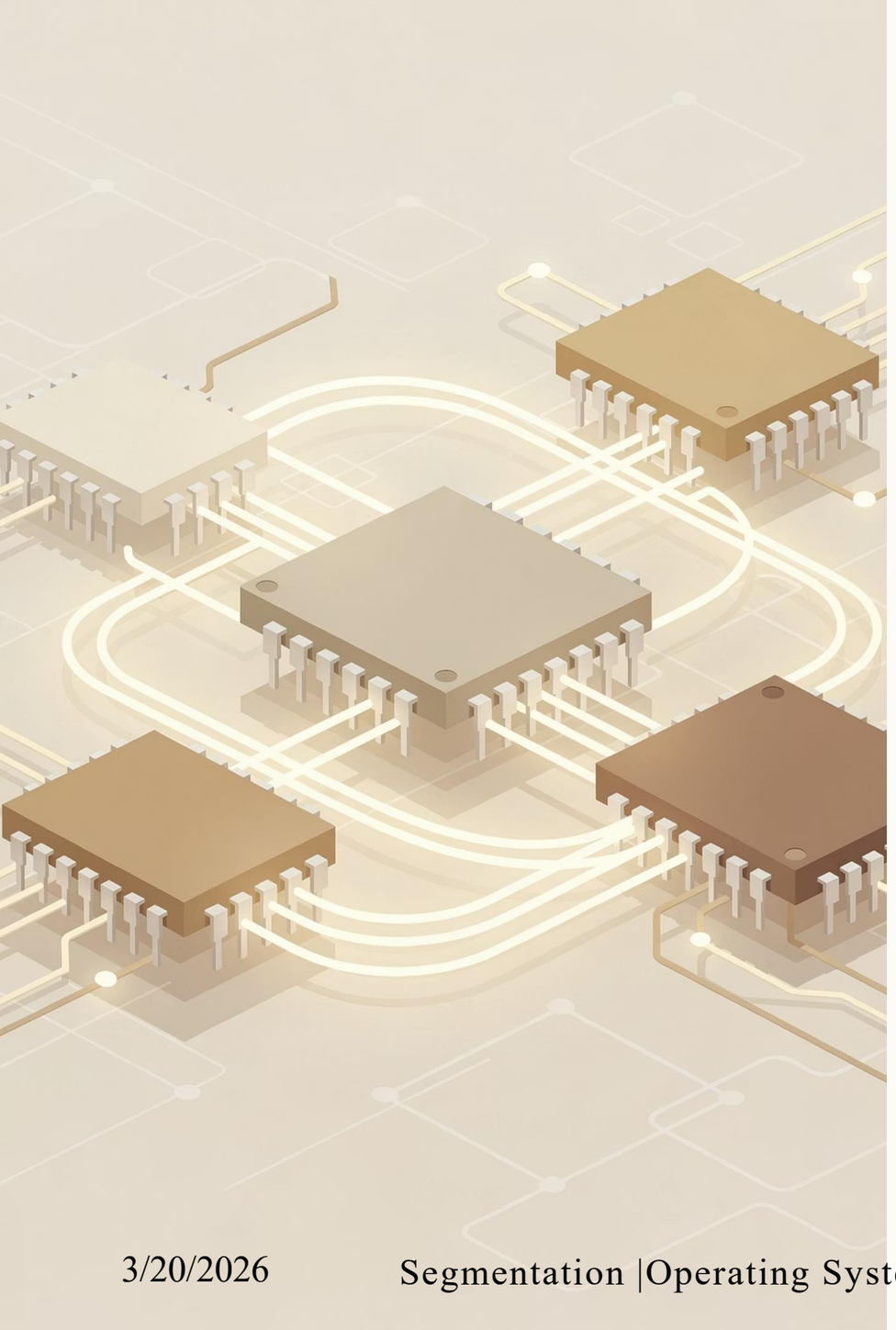
CPU generates: ⟨ Segment Number , Page Number , Offset ⟩

4

Step 6: Segment Table Lookup

Segment number indexes segment table. OS validates segment number





Steps 7-9: Address Formation

Step 7: Page Table Lookup

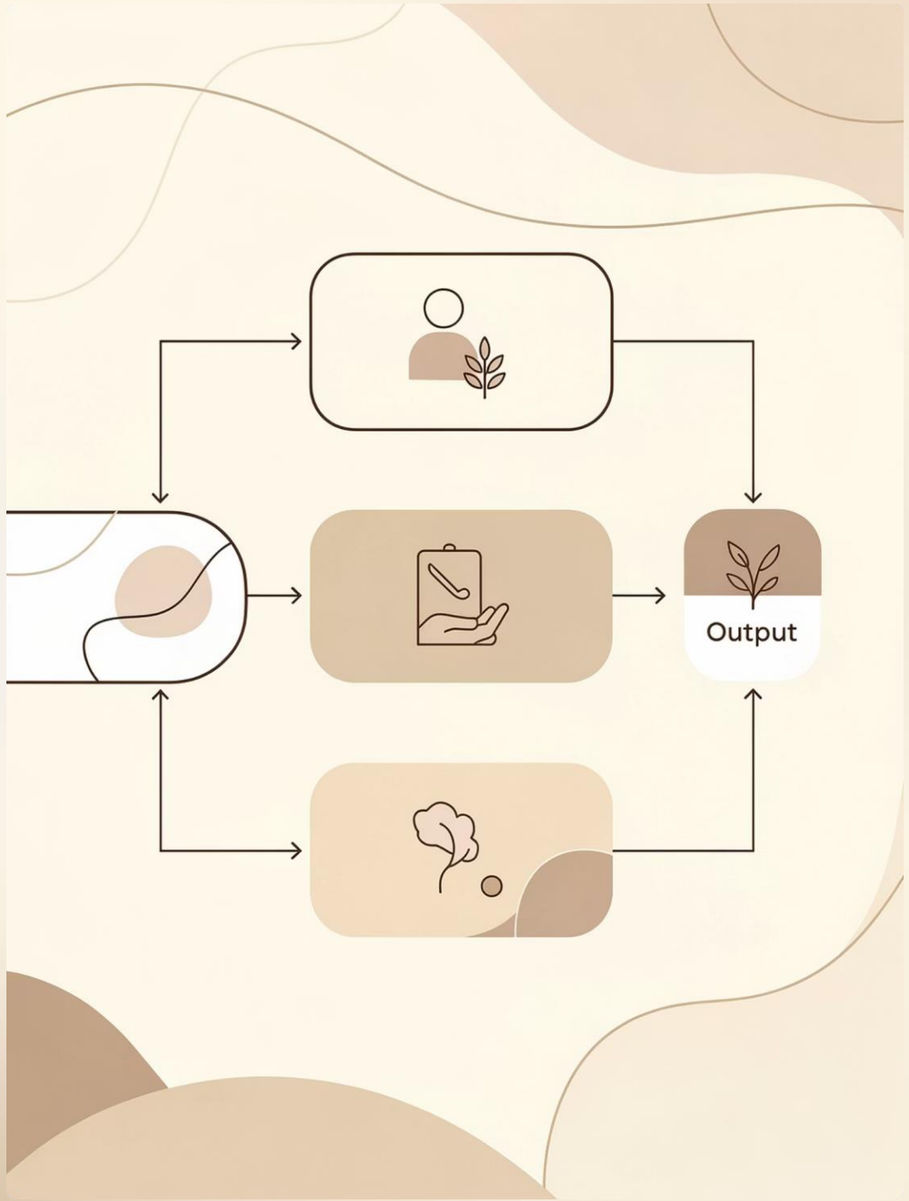
Page number accesses segment's page table to obtain frame number

Step 8: Physical Address Formation

Physical address = Frame Number + Offset

Step 9: Memory Access

CPU accesses data from calculated physical address



Step 10: Page Fault Handling

When Page Not in Memory

Page fault occurs → OS loads page from disk into free frame → Page table updated

Simple Flow Summary

1. Logical Address (S, P, D)
2. Segment Table
3. Page Table
4. Physical Address

DESIGN THINKING

Problem Statement

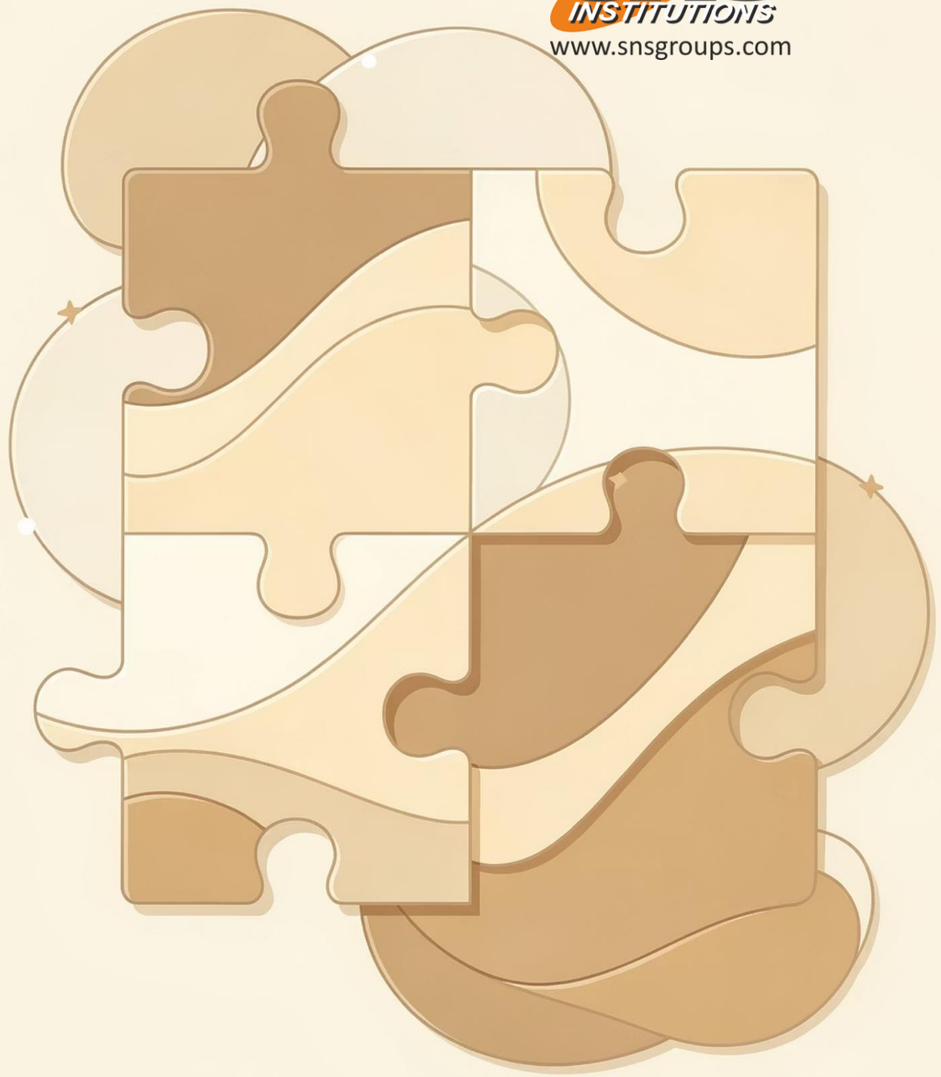
How can the OS manage memory so programs are logically organized and memory is used efficiently without wastage?

Challenge

Modern programs have logically different parts: code, data, stack, heap

Requirements

- Preserve logical structure
- Use memory efficiently
- Avoid fragmentation
- Support protection and sharing



Empathize

Programmers

Users

Operating System

Define

1

Logical Division

Supports logical division of programs

2

Dynamic Growth

Allows dynamic growth of program parts

3

Reduce Fragmentation

Minimizes memory fragmentation

4

Address Translation

Enables safe and efficient address translation

Solution 1

Segmentation – focuses on logical structure

Solution 2

Segmentation with Paging combines logic with efficient memory usage

Ideate

Idea 1: Segmentation

- Divide program into logical segments (Code, Data, Stack)
- Allow each segment to grow independently
- Use segment table for address translation

Idea 2: Segmentation with Paging

- Keep logical view of segmentation
- Divide each segment into fixed-size pages
- Use paging to eliminate external fragmentation

Prototype

Segmentation Prototype

Logical address format: ⟨ Segment Number , Offset ⟩

Segment table contains: Base address, Limit (segment size)

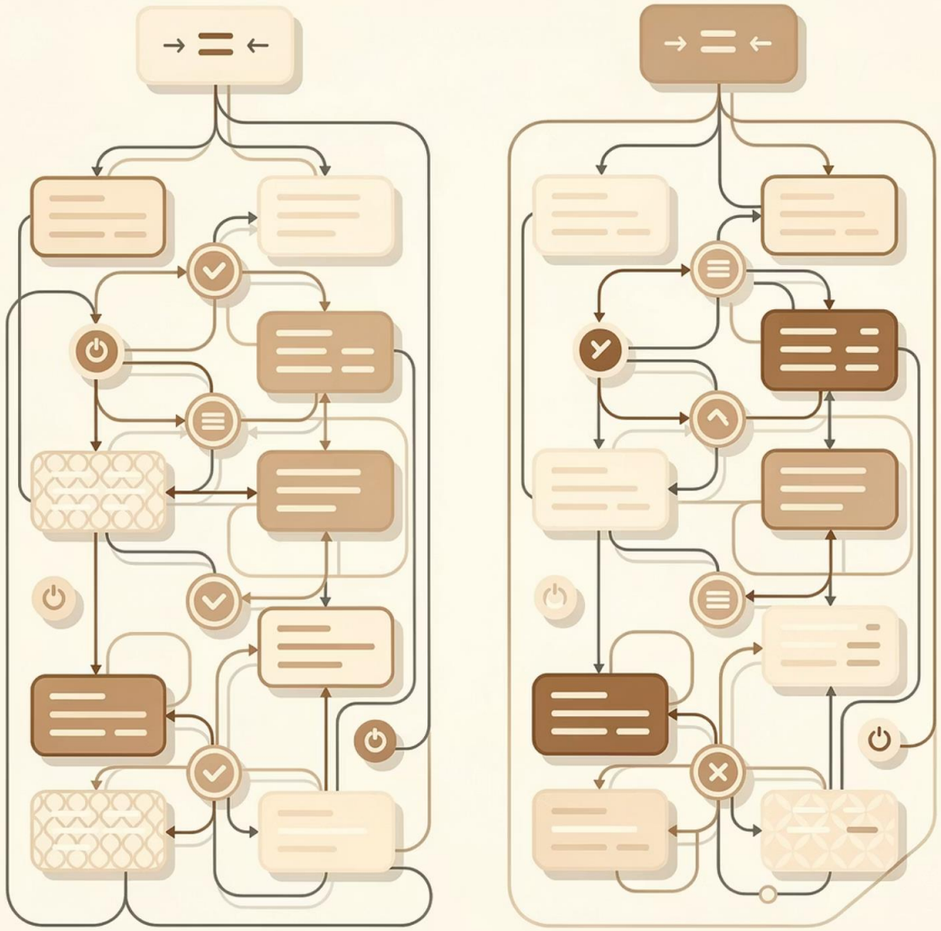
Address translation: Physical Address = Base + Offset

Segmentation with Paging Prototype

Logical address format: ⟨ Segment Number , Page Number , Offset ⟩

Uses: Segment table → finds page table, Page table → finds frame number

Address translation: Physical Address = Frame Number + Offset



Test

Testing Segmentation

Load programs with multiple segments

Check:

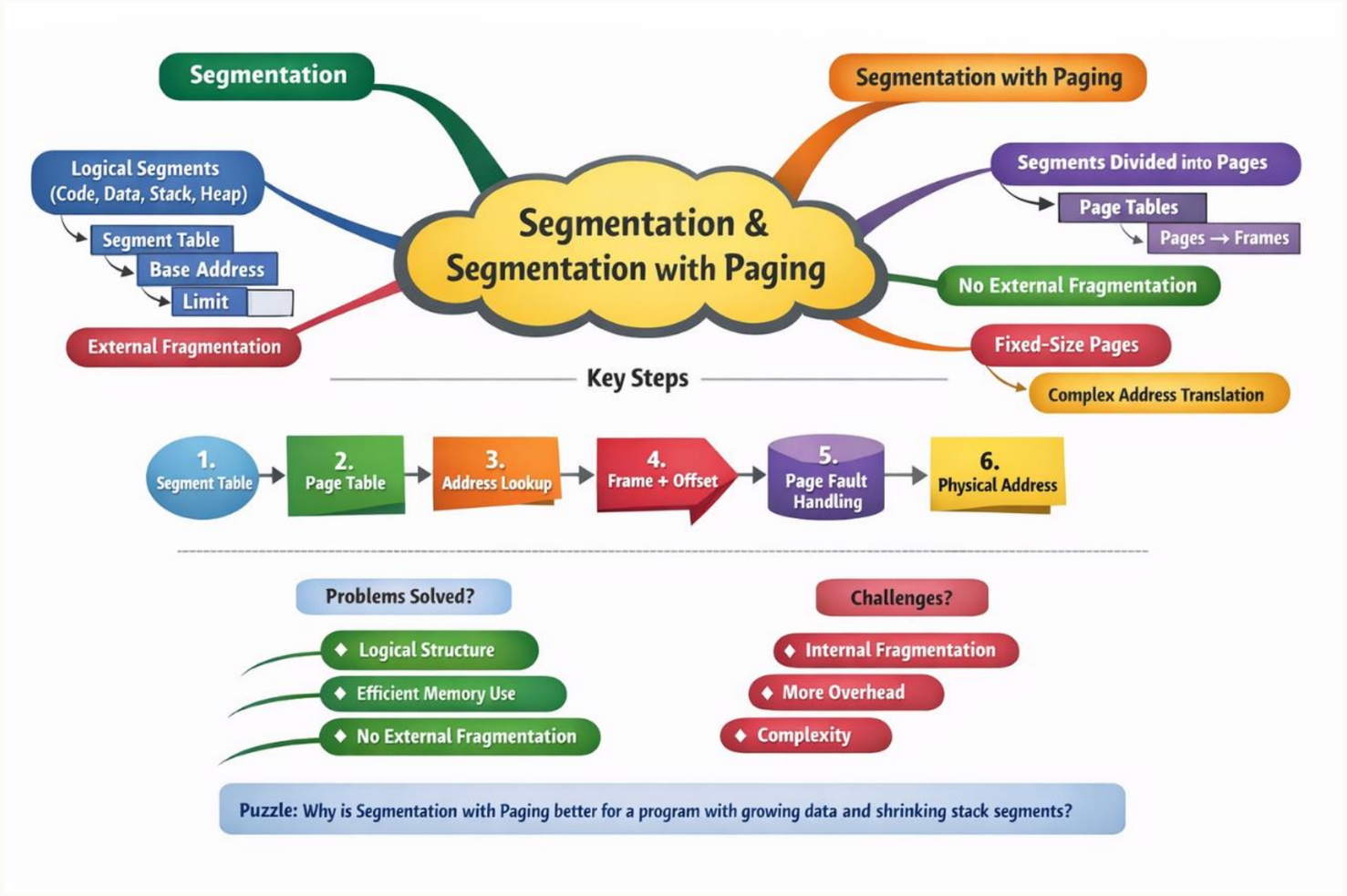
- Logical organization
- Protection between segments
- Presence of external fragmentation

Testing Segmentation with Paging

Load large and small segments

Observe:

- No external fragmentation
- Efficient memory usage
- Increased address translation steps


Segmentation
Logical Segments (Code, Data, Stack, Heap)
Segment Table
Base Address
Limit
External Fragmentation
Segmentation & Segmentation with Paging
Segmentation with Paging
Segments Divided into Pages
Page Tables
Pages → Frames
No External Fragmentation
Fixed-Size Pages
Complex Address Translation
Key Steps
1. Segment Table
2. Page Table
3. Address Lookup
4. Frame + Offset
5. Page Fault Handling
6. Physical Address
Problems Solved?
Logical Structure
Efficient Memory Use
No External Fragmentation
Challenges?
Internal Fragmentation
More Overhead
Complexity
Puzzle: Why is Segmentation with Paging better for a program with growing data and shrinking stack segments?