

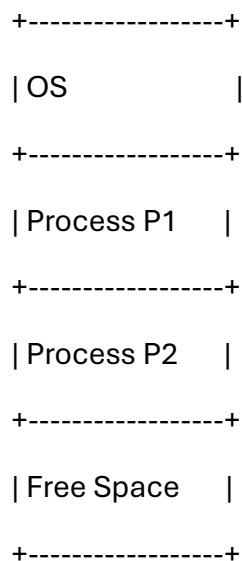
Contiguous Memory Allocation in Operating System

Contiguous memory allocation is a memory management technique in which **each process is allocated a single continuous block of main memory**. That means all the memory required by a process is placed **next to each other** in RAM.

Basic Idea

- Main memory is divided into **partitions**.
- Each process is loaded into **one partition only**.
- A process **cannot be split** across multiple memory locations.

Main Memory



Types of Contiguous Memory Allocation

1. Fixed Partitioning

- Memory is divided into **fixed-size partitions**.
- Each partition can hold **only one process**.
- Partition size is decided **before execution**.

Advantages

- ✓ Simple to implement
- ✓ Fast memory allocation

Disadvantages

X Internal Fragmentation (wasted space inside a partition)

X Process size limited to partition size

Partition Size = 100 MB

Process Size = 60 MB

→ 40 MB wasted (internal fragmentation)

2. Variable (Dynamic) Partitioning

- Memory is divided into partitions of **variable size**.
- Partition size is decided **when the process arrives**.
- Process gets **exactly the required memory**.

Advantages

✓ Less internal fragmentation

✓ Better memory utilization

Disadvantages

X External Fragmentation

X Needs memory compaction

Free blocks scattered in memory

Total free = enough, but not contiguous

Memory Allocation Strategies

(Used in Variable Partitioning)

1. First Fit

- Allocates the **first free block** that is large enough.
- Fast but may cause fragmentation.

2. Best Fit

- Allocates the **smallest free block** that fits the process.
- Reduces waste but slow (searches entire memory).

3. Worst Fit

- Allocates the **largest free block**.
 - Leaves large remaining holes.
-

Fragmentation in Contiguous Allocation

1. Internal Fragmentation

- Occurs in **fixed partitioning**.
- Unused memory **inside the allocated partition**.

2. External Fragmentation

- Occurs in **variable partitioning**.
 - Free memory exists but is **not contiguous**.
-

Compaction

- Technique to **reduce external fragmentation**.
- OS shifts processes to make all free space **continuous**.

Before Compaction: Free + Process + Free + Process

After Compaction: Process + Process + Free

⚠ Costly operation (requires moving processes).

Advantages of Contiguous Memory Allocation

- ✓ Simple design
 - ✓ Easy to manage
 - ✓ Faster access (single base address)
-

Disadvantages

- ✗ Memory wastage due to fragmentation
 - ✗ Difficult to support process growth
 - ✗ Not suitable for modern multiprogramming systems
-

Virtual Memory (Operating System)

Definition

Virtual Memory is a memory management technique that allows a system to execute processes larger than the size of physical RAM by using secondary storage (disk) as an extension of main memory.

👉 It creates an illusion of large main memory for programs.

Why Virtual Memory is Needed

- Programs are getting larger
 - RAM is limited and expensive
 - Better multiprogramming is required
 - Efficient memory utilization
-

Basic Concept

- Each process is given a virtual address space
- Only the required parts (pages) of a process are loaded into RAM
- Remaining parts stay on disk (swap space)

Virtual Address → MMU → Physical Address

↓

Page Table

How Virtual Memory Works

1. CPU generates a virtual address
2. MMU (Memory Management Unit) checks the page table
3. If page is in RAM → normal execution
4. If page is not in RAM → Page Fault
5. OS loads required page from disk into RAM
6. Page table is updated and execution resumes

Key Components of Virtual Memory

1. Page

- **Fixed-size block of virtual memory**
- **Example: 4 KB**

2. Frame

- **Fixed-size block of physical memory**
- **Same size as page**

3. Page Table

- **Maps pages → frames**
 - **Maintained by OS**
-

Page Fault

- **Occurs when a required page is not present in main memory**
- **OS must:**
 - **Pause the process**
 - **Fetch the page from disk**
 - **Replace a page if RAM is full**

⚠ Page faults are slow (disk access)

Page Replacement Algorithms

(Used when RAM is full)

1. FIFO

- **Removes the oldest loaded page**

2. LRU

- **Removes the least recently used page**

3. Optimal

- **Removes the page that will not be used for the longest time**

- Theoretical (used for comparison)
-

Types of Virtual Memory Implementation

1. Paging

- Memory divided into fixed-size pages
- No external fragmentation
- May cause internal fragmentation

2. Segmentation

- Memory divided into logical segments
 - Based on program structure
 - May cause external fragmentation
-

Thrashing

- When system spends more time handling page faults than executing processes
 - Caused by:
 - Too many processes
 - Insufficient RAM
-

Advantages of Virtual Memory

- ✓ Programs larger than RAM can run
 - ✓ Better memory utilization
 - ✓ More processes can reside in memory
 - ✓ Less I/O overhead
-

Disadvantages

- ✗ Slower due to disk access
- ✗ Complex implementation
- ✗ Page faults reduce performance

Difference: Physical vs Virtual Memory

Physical Memory Virtual Memory

Actual RAM	Logical memory
------------	----------------

Limited size	Appears large
--------------	---------------

Fast access	Slower (uses disk)
-------------	--------------------

Paging in Operating System

Definition

Paging is a memory management technique in which physical memory and logical (virtual) memory are divided into fixed-size blocks.

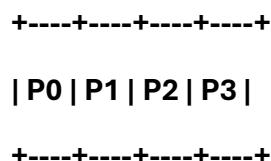
- Logical memory blocks → Pages
- Physical memory blocks → Frames

Paging is mainly used to implement virtual memory.

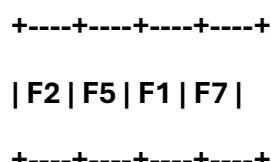
Basic Idea

- A process is divided into pages.
- Pages are loaded into any available frame in RAM.
- Pages need not be contiguous in physical memory.

Virtual Memory (Process)



Physical Memory (RAM)



Address Translation in Paging

A logical address generated by CPU has two parts:

Logical Address = (Page Number, Page Offset)

- **Page Number (p) → Index of page table**
- **Page Offset (d) → Position inside the page**

Physical Address = (Frame Number, Offset)

Page Table

- **Stores mapping between page number → frame number**
- **Each process has its own page table**
- **Maintained by the Operating System**

Page No Frame No

0	5
1	2
2	8
3	1

Paging with Diagram

CPU



| **Logical Address (p, d)**



Page Table



| **Frame Number**



Physical Memory

Page Fault

- Occurs when the required page is not present in RAM
 - OS actions:
 1. Suspend process
 2. Fetch page from disk
 3. Replace a page (if needed)
 4. Update page table
 5. Resume execution
-

Page Replacement Algorithms

(Used during page fault)

1. FIFO – First loaded page is replaced
 2. LRU – Least recently used page is replaced
 3. Optimal – Page not used for longest future time
-

Fragmentation in Paging

Internal Fragmentation

- Possible
- Occurs when process size is not multiple of page size

External Fragmentation

- Not present (major advantage)
-

Advantages of Paging

- ✓ No external fragmentation
- ✓ Efficient memory utilization
- ✓ Supports virtual memory
- ✓ Allows non-contiguous allocation

Disadvantages of Paging

- X Page table overhead**
 - X Address translation time**
 - X Internal fragmentation**
-

Paging vs Contiguous Allocation

Paging	Contiguous Allocation
Non-contiguous	Contiguous
No external fragmentation	External fragmentation
Uses page table	No page table
Used in modern OS	Used in old OS

Page Table Structure (Operating System)

What is a Page Table?

A page table is a data structure used by the Operating System to store the mapping between:

- **Virtual page numbers (VPN)**
- **Physical frame numbers (PFN)**

It helps the MMU (Memory Management Unit) convert a virtual address into a physical address.

Basic Page Table Structure

Each process has its own page table.

Simple Page Table Entry (PTE)

Field	Description
Frame Number	Physical frame where page is stored
Valid / Invalid Bit	1 → page in memory, 0 → page not in memory
Protection Bits	Read / Write / Execute permissions
Dirty Bit	1 → page modified
Reference Bit	Used for replacement algorithms

Logical Address Format

Logical Address

```
+-----+-----+
```

| Page Number | Page Offset |

```
+-----+-----+
```

- Page Number → Index into page table
 - Offset → Used directly in physical memory
-

Physical Address Format

Physical Address

```
+-----+-----+
```

| Frame Number | Page Offset |

```
+-----+-----+
```

Single-Level Page Table

Structure

- One page table per process
- Indexed directly by page number

Page Table

```
+---+---+
```

P#	Frame#
0	5
1	2
2	8
3	1

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

| 0 | 5 |

| 1 | 2 |

| 2 | 8 |

| 3 | 1 |

Problem

✗ Large memory overhead for big address spaces

Multi-Level Page Table

Idea

- Page table is divided into levels
- Only required portions are kept in memory

Two-Level Paging Example

Virtual Address

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

| P1 | P2 | Offset |

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

- P1 → Outer page table index
- P2 → Inner page table index

✓ Saves memory

Inverted Page Table

Structure

- One page table for the entire system
- Entry for each physical frame
- Stores:
 - Process ID

- Virtual page number

Frame | PID | Page#

Advantages

✓ Very small size

Disadvantages

✗ Slower lookup (needs hashing)

Hashed Page Table

- Used for large address spaces (64-bit systems)
 - Virtual page number is hashed
 - Linked list handles collisions
-

Translation Lookaside Buffer (TLB)

What is TLB?

- Small, fast cache inside MMU
- Stores recent page table entries

CPU → TLB → Page Table → Memory

✓ Reduces memory access time

Comparison of Page Table Structures

Type	Memory Use	Speed
Single-Level	High	Fast
Multi-Level	Medium	Medium
Inverted	Low	Slow
Hashed	Low	Medium

Demand Paging (Operating System)

Definition

Demand Paging is a virtual memory technique in which pages are loaded into main memory only when they are actually needed (on demand), not in advance.

👉 A page is brought into RAM only when a page fault occurs.

Basic Idea

- Initially, no pages (or very few) of a process are loaded into memory
 - When a process accesses a page:
 - If page is in RAM → execution continues
 - If page is not in RAM → page fault occurs
-

Working of Demand Paging (Step-by-Step)

1. CPU generates a virtual address
 2. MMU checks the page table
 3. If valid bit = 1 → page is in memory
 4. If valid bit = 0 → page fault
 5. OS:
 - Finds the page on disk
 - Allocates a free frame (or replaces a page)
 - Loads page into memory
 - Updates page table
 6. Process resumes execution
-

Demand Paging Diagram

CPU



| Virtual Address



Page Table (Valid/Invalid Bit)



└─ Valid → Physical Memory



└─ Invalid → Page Fault



Disk

Valid-Invalid Bit

- 1 (Valid) → page is present in memory
 - 0 (Invalid) → page is not present / illegal access
-

Page Fault Handling

Actions by OS

1. Save process state
2. Check if reference is valid
3. Find free frame
4. Read page from disk
5. Update page table
6. Restart instruction

⚠ Page fault is very expensive (disk I/O)

Page Replacement (When Memory is Full)

Common algorithms:

- FIFO
- LRU
- Optimal

Advantages of Demand Paging

- ✓ Reduced memory usage
 - ✓ Faster program start-up
 - ✓ More processes in memory
 - ✓ Efficient RAM utilization
-

Disadvantages

- ✗ Page fault overhead
 - ✗ Thrashing if too many faults
 - ✗ Complex implementation
-

Thrashing

- System spends most time handling page faults
 - Caused by:
 - High degree of multiprogramming
 - Insufficient memory
-

Demand Paging vs Pre-Paging

Demand Paging	Pre-Paging
---------------	------------

Pages loaded when needed	Pages loaded in advance
--------------------------	-------------------------

Fewer pages in memory	More pages in memory
-----------------------	----------------------

More page faults initially	Fewer initial faults
----------------------------	----------------------

Better memory utilization	More memory use
---------------------------	-----------------

Page Replacement Policies (Operating System)

What is Page Replacement?

When a page fault occurs and no free frame is available in main memory, the OS must remove (replace) one existing page to load the required page.

The rule used to decide which page to remove is called a page replacement policy.

Goals of Page Replacement

- **Minimize page faults**
 - **Improve CPU utilization**
 - **Reduce disk I/O**
 - **Avoid thrashing**
-

Common Page Replacement Policies

1. FIFO (First-In First-Out)

Idea

- **Replace the oldest page in memory (the one that came first)**

How it works

- **Maintain a queue**
- **Remove page at the front**

Example

Reference string: 7 0 1 2 0 3 0 4

Frames = 3

Page Faults = High

Advantages

- ✓ **Simple**
- ✓ **Easy to implement**

Disadvantages

- X May replace frequently used pages
 - X Belady's Anomaly occurs
-

2. Optimal Page Replacement

Idea

- Replace the page that will not be used for the longest time in the future

Advantages

- ✓ Minimum page faults
- ✓ Used as a benchmark

Disadvantages

- X Future references unknown
 - X Not implementable in real systems
-

3. LRU (Least Recently Used)

Idea

- Replace the page that was not used for the longest time in the past

Implementation

- Counter or stack-based

Advantages

- ✓ Good performance
- ✓ No Belady's anomaly

Disadvantages

- X Hardware support required
 - X Higher overhead
-

4. Second Chance (Clock Algorithm)

Idea

- FIFO + Reference bit
- Gives pages a second chance

Working

- If reference bit = 1 → reset to 0 and skip
- If reference bit = 0 → replace

Advantages

- ✓ Better than FIFO
 - ✓ Efficient
-

5. LFU (Least Frequently Used)

Idea

- Replace the page with lowest usage count

Advantages

- ✓ Considers frequency

Disadvantages

- ✗ Old pages may stay forever
 - ✗ Complex to implement
-

6. MFU (Most Frequently Used)

Idea

- Replace page with highest usage count

Logic

- Page used a lot → probably finished its job

Disadvantages

- ✗ Rarely used in practice
-

Comparison Table

Policy	Performance		
	Belady's Anomaly	Implementation	
FIFO	Poor	Yes	Easy

Policy	Performance Belady's Anomaly Implementation		
Optimal	Best	No	Not possible
LRU	Good	No	Complex
Second Chance	Medium	No	Easy
LFU	Medium	No	Hard

Belady's Anomaly

- Increasing number of frames increases page faults
 - Occurs in FIFO only
-

1. Thrashing (Operating System)

Definition

Thrashing is a condition in which the system spends most of its time handling page faults instead of executing actual processes.

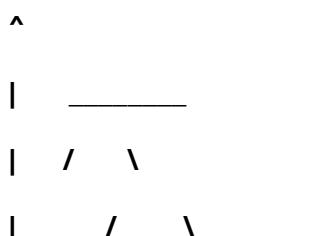
👉 CPU utilization becomes very low because the OS is busy swapping pages in and out of memory.

Why Thrashing Occurs

- Very high degree of multiprogramming
 - Insufficient physical memory
 - Poor page replacement algorithm
 - Processes do not have enough frames to maintain locality of reference
-

Thrashing Diagram (Conceptual)

CPU Utilization



Degree of Multiprogramming

Effects of Thrashing

- Very high page fault rate
 - Low CPU utilization
 - System slowdown
 - Poor overall performance
-

Methods to Control Thrashing

1. Working Set Model

- Keeps track of pages actively used
- Ensures process has enough frames

2. Page Fault Frequency (PFF)

- If fault rate too high → allocate more frames
- If fault rate too low → remove frames

3. Reduce Degree of Multiprogramming

- Suspend some processes

4. Use Better Page Replacement

- LRU, Working Set based
-

Exam Points

- Thrashing occurs in virtual memory systems
 - High page fault rate is the main indicator
 - Controlled by working set & PFF
-
-

2. Segmentation (Operating System)

Definition

Segmentation is a memory management technique where a program is divided into logical segments based on its structure.

Each segment represents:

- Code
 - Data
 - Stack
 - Heap
 - Functions
-

Segment Structure

- Variable size
- Logical grouping

Program

+-----+

| Code |

+-----+

| Data |

+-----+

| Stack |

+-----+

| Heap |

+-----+

Logical Address Format

Logical Address = (Segment Number, Offset)

Segment Table

Each process has a segment table.

Segment No Base Address Limit

0	4000	1200
1	5200	800
2	6000	600

- Base → starting physical address
- Limit → segment size

Address Translation in Segmentation

- 1. Segment number selects table entry**
 - 2. Offset is checked against limit**
 - 3. Physical address = Base + Offset**
-

Advantages of Segmentation

- ✓ Matches programmer's view**
 - ✓ Supports protection & sharing**
 - ✓ No internal fragmentation**
-

Disadvantages

- X External fragmentation**
 - X Compaction required**
 - X Complex memory management**
-

Segmentation vs Paging

Segmentation	Paging
Logical division	Physical division
Variable size	Fixed size
External fragmentation	No external fragmentation
Easy protection	Easier allocation