Operating Systems
Course Code: **71203002004**

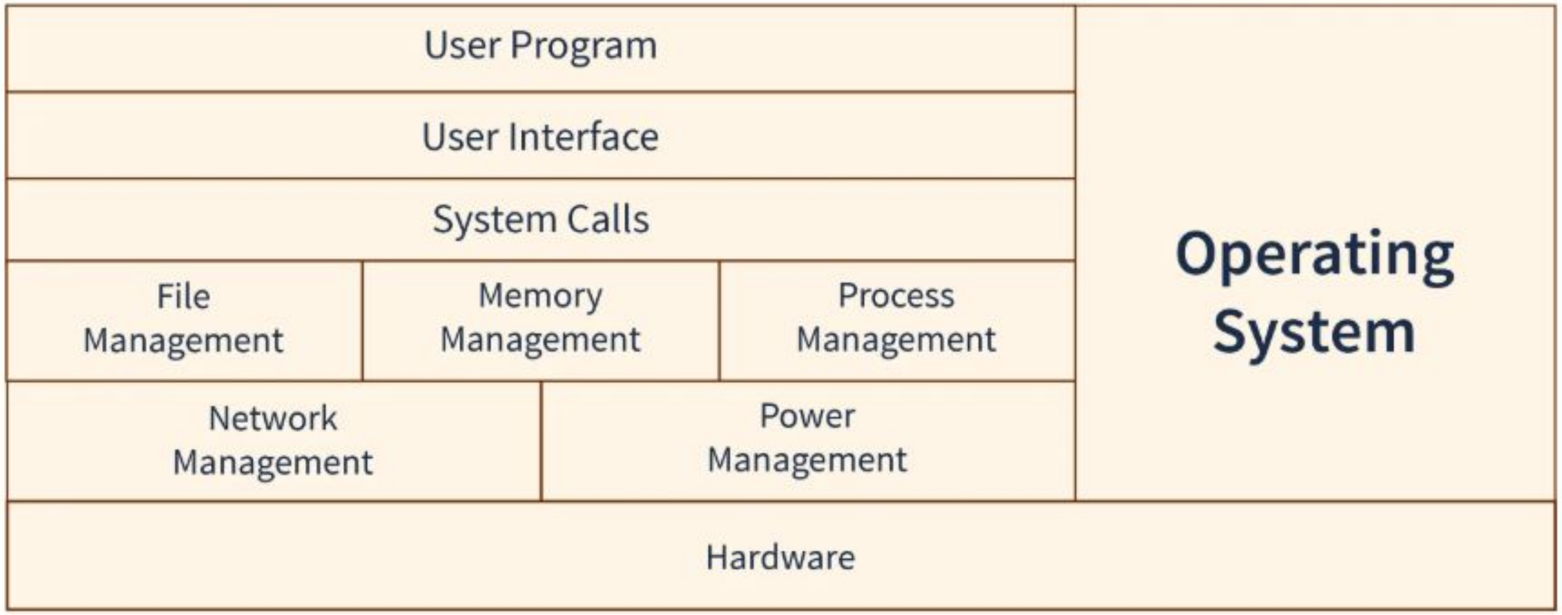
Memory Management Fundamentals

by -

Minal Rajwar

Memory Management Techniques in Operating System

Memory management is a core function of the operating system (OS). It ensures that multiple processes get the required memory efficiently, without wasting space or causing conflicts. It also helps in optimizing CPU utilization and overall system performance.



Memory Management in OS

1. Memory Allocation Schemes

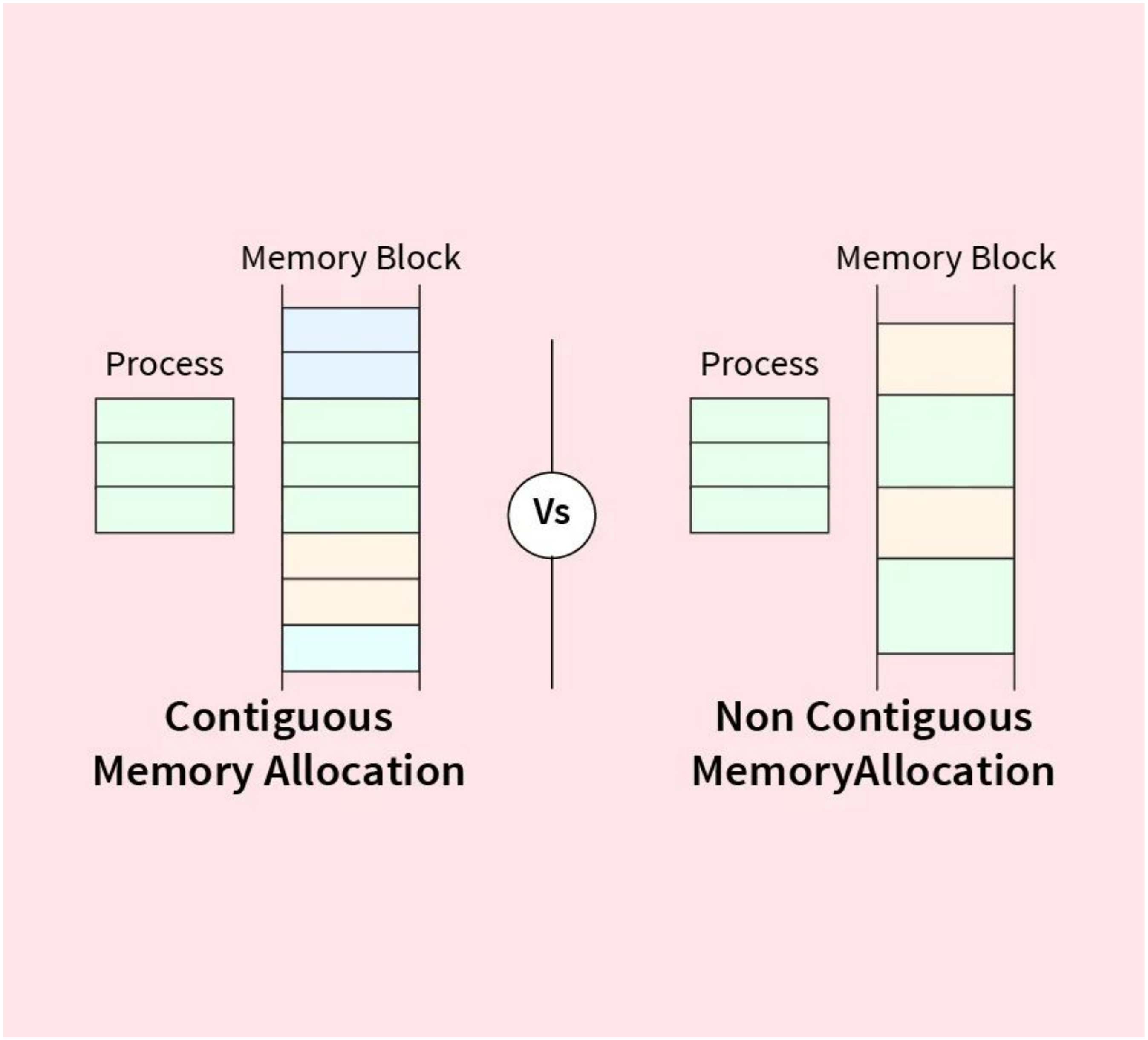
a) Contiguous Memory Allocation

- Process is stored in a single continuous block of memory.
- Example: Arrays.
- **Problem**: Leads to **fragmentation**.
- Used in simple systems but not efficient for modern multiprogramming OS.

1. Memory Allocation Schemes

b) Non-Contiguous Memory Allocation

- Program is divided into blocks (fixed or variable size).
- Blocks are stored in different parts of memory.
- Example: Paging, Segmentation.
- Provides better flexibility and efficient use of available memory.


Memory Block
Process
Contiguous
Memory Allocation
Vs
Memory Block
Process
Non Contiguous
MemoryAllocation

2. Static vs Dynamic Loading and Linking

1. Static Loading

- All program components (code + data) are loaded into **main memory before execution** starts.
- Executable code and data are placed at predetermined fixed memory addresses.
- Advantage: Provides direct access to all resources, with no runtime overhead of loading.
- Disadvantage: Wastes memory if many loaded components are never used.

- **Best suited for small programs or systems with limited dynamic capabilities.**

2. Static vs Dynamic Loading and Linking

2. Dynamic Loading

- Program components are loaded into memory **only when required during execution**.
- Reduces memory usage since only active components occupy memory.
- Advantage: Efficient memory utilization, especially for large programs with many optional modules.
- Disadvantage: Extra overhead at runtime when new modules need to be loaded.

- **Common in modern systems to improve flexibility and reduce startup memory usage.**

2. Static vs Dynamic Loading and Linking

3. Static Linking

- All required libraries are **copied into the executable** at compile time.
- The result is a **self-contained executable** that runs without external dependencies.
- Advantage: Portable and independent of external library versions.
- Disadvantage: Larger file size and redundancy if multiple programs use the same library.

- **Useful in embedded systems where updates and dependencies must be avoided.**

2. Static vs Dynamic Loading and Linking

4. Dynamic Linking

- Libraries are **not included in the executable**, but linked at runtime when needed.
- Multiple programs share the same copy of a library, saving memory and disk space.
- Advantage: Saves storage, easier to update libraries without recompiling programs.
- Disadvantage: Dependency on correct library versions being present at runtime.

- **Widely used in modern operating systems to support shared libraries (DLLs in Windows, .so in Linux).**

2. Static vs Dynamic Loading and Linking

5. Example

- Multiple programs use a **math library**:
    - **Static Linking** → Each program has its own copy, increasing executable size.
    - **Dynamic Linking** → All programs share one copy, reducing redundancy and saving memory.

Comparison Table

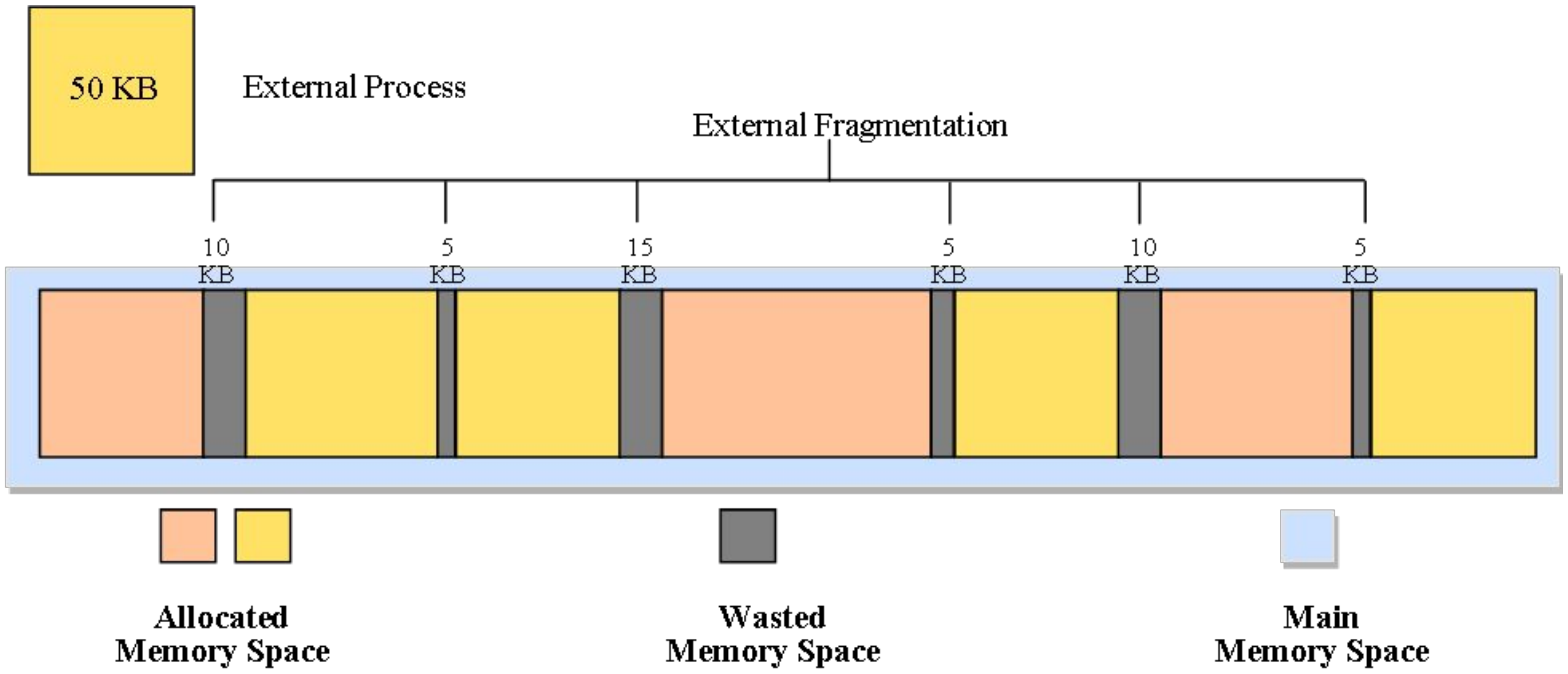
Aspect	Static Loading	Dynamic Loading	Static Linking	Dynamic Linking
When?	Before execution begins	During execution, when needed	At compile time	At runtime
Memory Use	Higher (all components loaded)	Lower (only needed parts loaded)	Higher (each executable has a copy)	Lower (shared libraries)
Performance	Faster execution (everything ready)	Slower initially (runtime loading overhead)	Faster startup (no external dependency)	May be slower due to runtime linking
File Size	Fixed size, may include unused parts	Smaller (only essential components loaded)	Large executable	Smaller executable
Flexibility	Low	High (loads modules on demand)	Low (harder to update libraries)	High (easy updates, shared libraries)
Dependency	None during execution	Requires loader to fetch modules	Independent of system libraries	Dependent on system libraries availability

3. Fragmentation

Fragmentation occurs when free memory is not efficiently used. It reduces overall system efficiency as processes cannot be allocated space even when total free memory is sufficient.

a) External Fragmentation

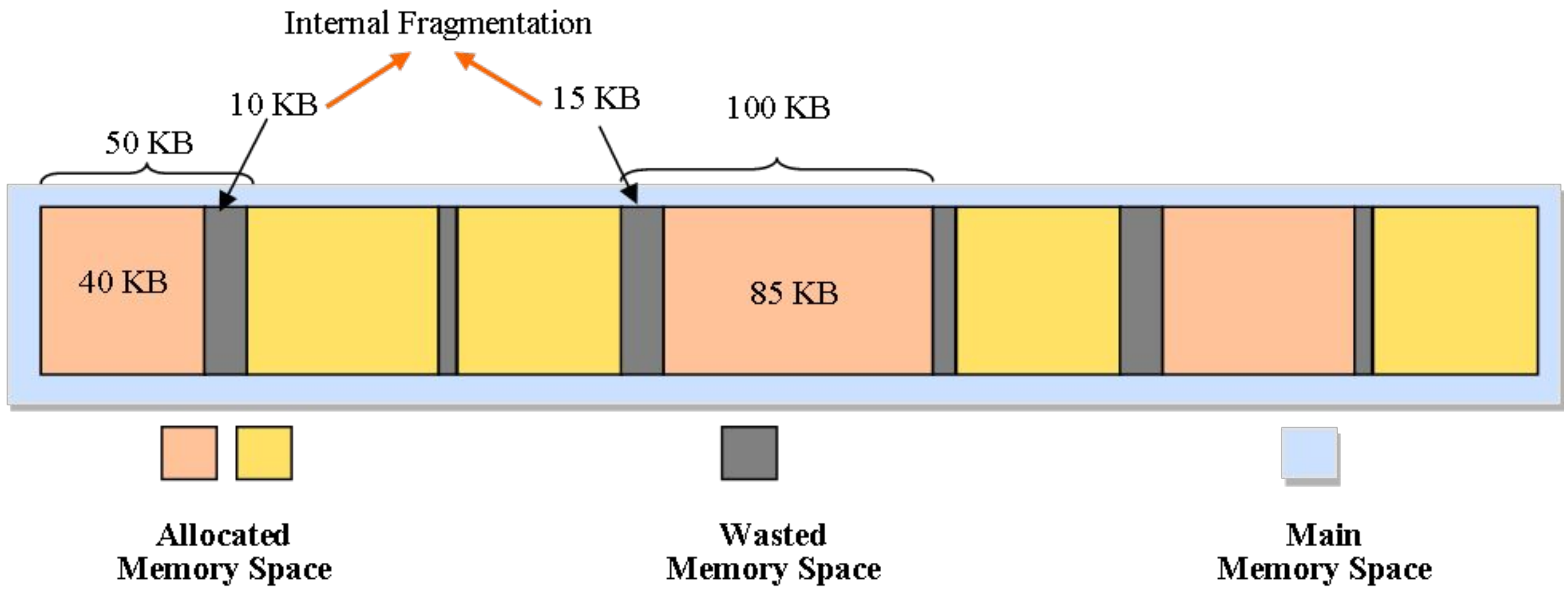
- Free memory is available but scattered (non-contiguous).
- Example: Need 5 KB, but memory has 2 KB + 2 KB + 1 KB.
- **Solution**: Compaction (rearranging memory).
- Reduces effective memory utilization despite having enough space.

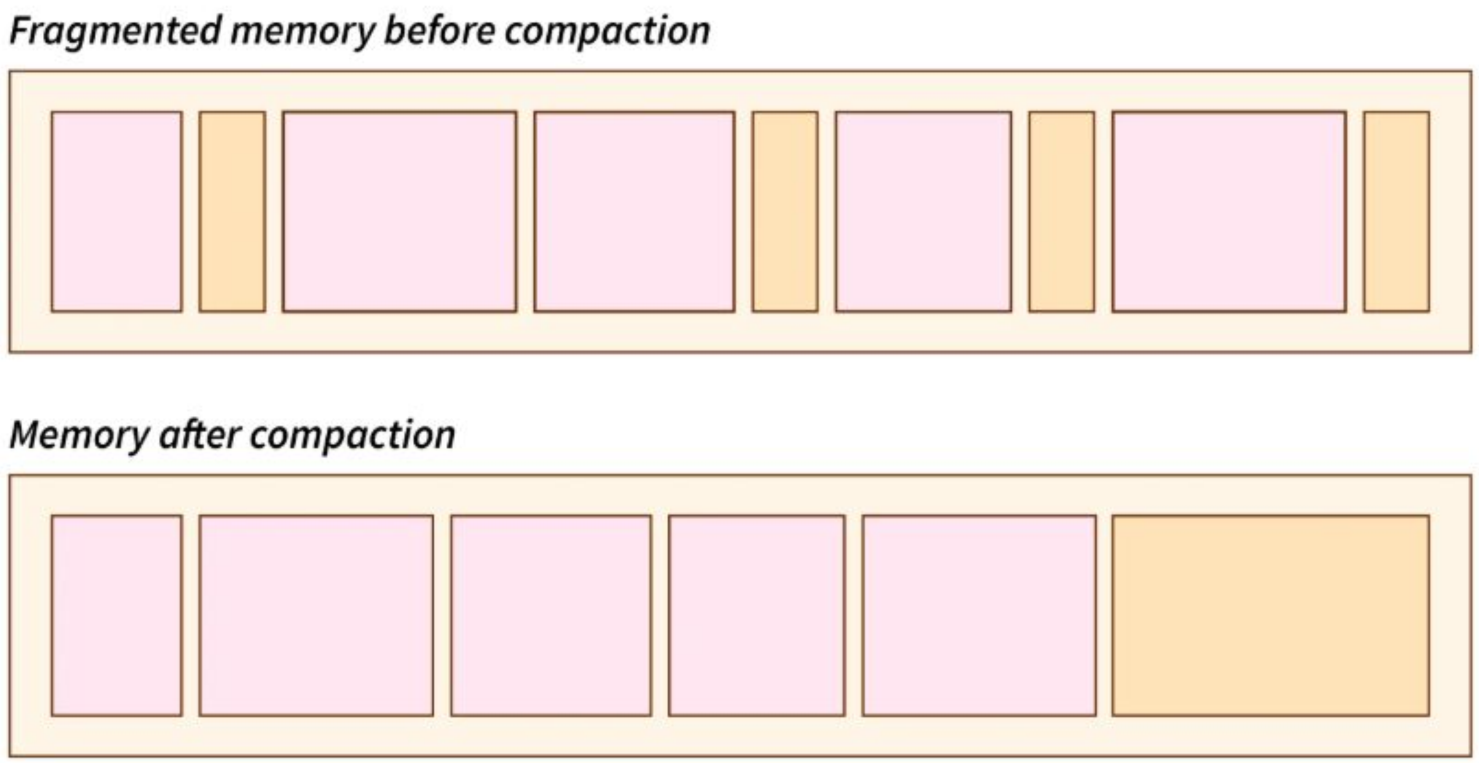

50 KB
External Process
External Fragmentation
10 KB
5 KB
15 KB
5 KB
10 KB
5 KB
Allocated Memory Space
Wasted Memory Space
Main Memory Space

3. Fragmentation

b) Internal Fragmentation

- More memory is allocated than needed.
- Example: Need 2 KB but allocated 6 KB → 4 KB wasted.
- **Solution**: Assign smallest partition that fits.
- Common in fixed partition memory allocation methods.

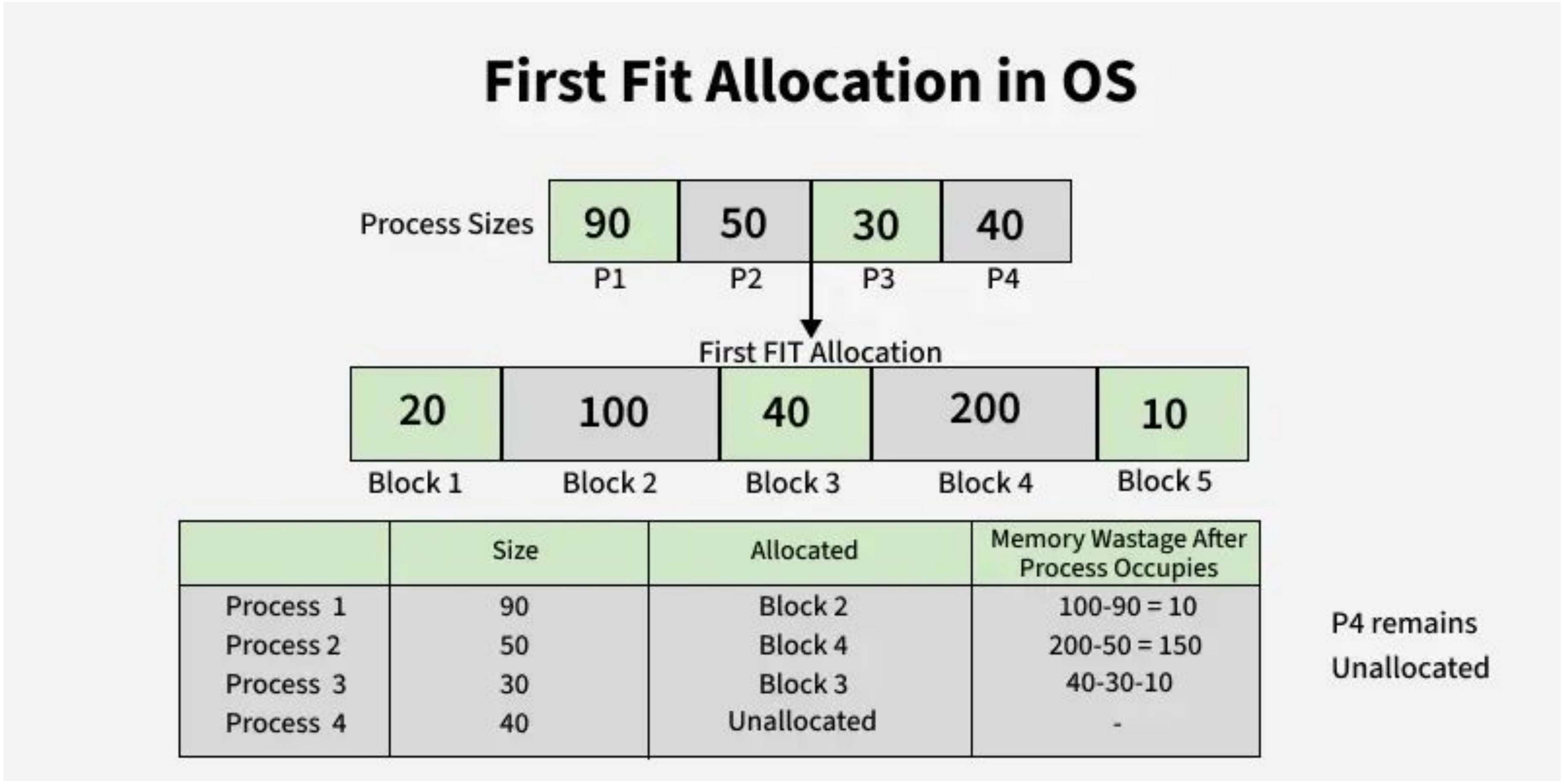

Internal Fragmentation
10 KB
15 KB
50 KB
100 KB
40 KB
85 KB
Allocated Memory Space
Wasted Memory Space
Main Memory Space


Fragmented memory before compaction
Memory after compaction

4. Memory Allocation Strategies

When a process requests memory, the OS must decide how to allocate it efficiently.

1. **First Fit**
    - Assign the first block of memory that is big enough.
    - Simple, fast.
    - May cause fragmentation.
    - Works well when memory requests are frequent and small.



First Fit Allocation in OS

	Size	Allocated	Memory Wastage After Process Occupies
Process 1	90	Block 2	100-90 = 10
Process 2	50	Block 4	200-50 = 150
Process 3	30	Block 3	40-30-10
Process 4	40	Unallocated	-

P4 remains Unallocated

4. Memory Allocation Strategies

2. Best Fit

- Search all blocks and assign the **smallest suitable block**.
- Less wastage.
- Slow, more fragmentation.
- Tries to minimize leftover space but increases search time.

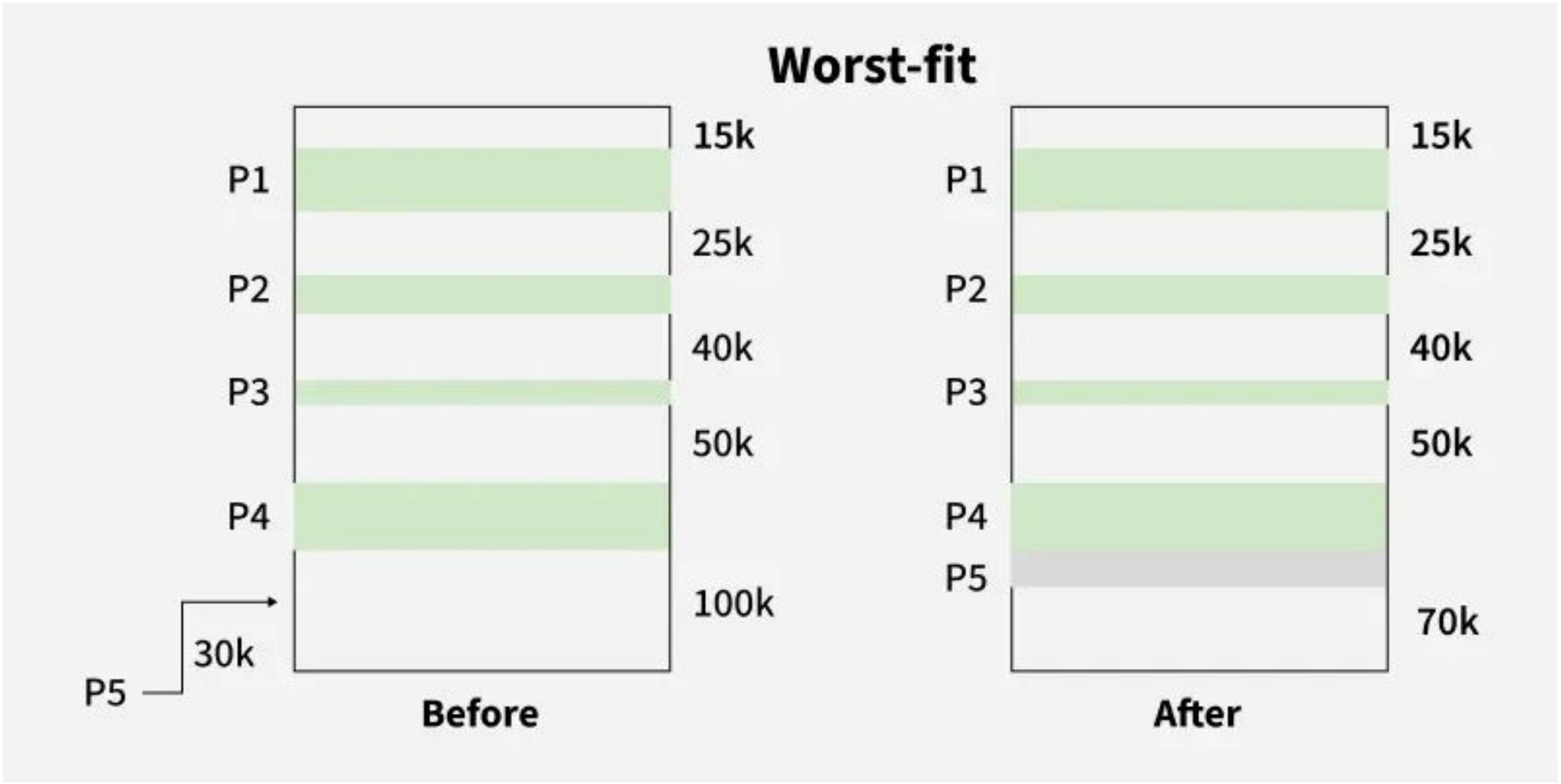
Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J4	50 K	Busy	None
300875	200 K	J2	200 K	Busy	None
809567	700 K	J3	500 K	Busy	200 K
Total available :	980 K	Total used :	770 K		210 K

4. Memory Allocation Strategies

3. Worst Fit

- Assign the **largest available block**.
- Leaves larger free blocks for future.
- Higher fragmentation.
- Useful when processes with large memory requirements are expected.


Worst-fit
P1
P2
P3
P4
15k
25k
40k
50k
100k
30k
P5
Before
P1
P2
P3
P4
P5
15k
25k
40k
50k
70k
After

5. Common Memory Management Techniques

Swapping – Moving processes between main memory and disk to free space. Helps in running more processes than the available physical memory.

Paging – Divide memory into fixed-size pages and frames. Eliminates external fragmentation but may cause internal fragmentation.

5. Common Memory Management Techniques

Segmentation – Divide program into variable-size segments based on logical units. Provides better program structure and protection.

Compaction – Rearrange memory to remove external fragmentation. Improves utilization but requires CPU overhead during rearrangement.

DISCUSSION & REVISION

1. Which memory allocation type divides memory into continuous blocks?
2. Which fragmentation occurs due to scattered free blocks in memory?
3. Which fragmentation occurs when a process is allocated more memory than needed?
4. Which memory allocation strategy chooses the first suitable free block?
5. Which memory management technique divides memory into fixed-size pages?
6. Which linking method copies library code into the executable file?
7. Which loading method brings program components into memory only when required?
8. Which memory allocation type allows program blocks to be stored at different locations?

REFERENCES

1. https://er.yuvayana.org/memory-management-strategies/
2. https://www.shiksha.com/online-courses/articles/memory-management-techniques-in-operating-system/
3. https://www.scaler.com/topics/memory-management-in-operating-system/
4. https://www.geeksforgeeks.org/operating-systems/first-fit-allocation-in-operating-systems/
5. https://www.geeksforgeeks.org/operating-systems/best-fit-allocation-in-operating-system/
6. https://www.geeksforgeeks.org/operating-systems/worst-fit-allocation-in-operating-systems/