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 6:Free Space Management

Free Space Management in Operating System

Understanding how operating systems efficiently track and manage available disk space for optimal file system performance.

Space Tracking

OS monitors free blocks and metadata

Allocation Strategies

Choose best-fit, first-fit, or extent

Fragmentation Control

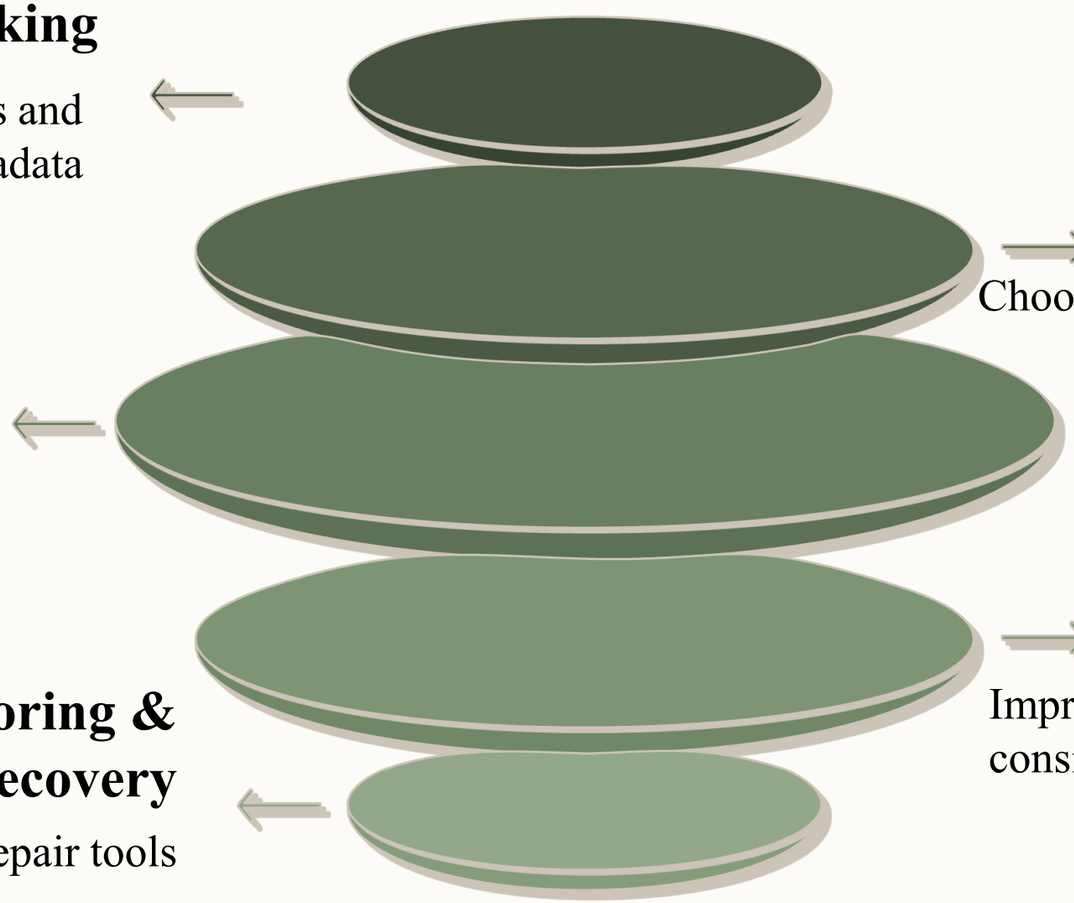
Defragmentation and allocation balancing

Caching & Journaling

Improve speed and ensure consistency

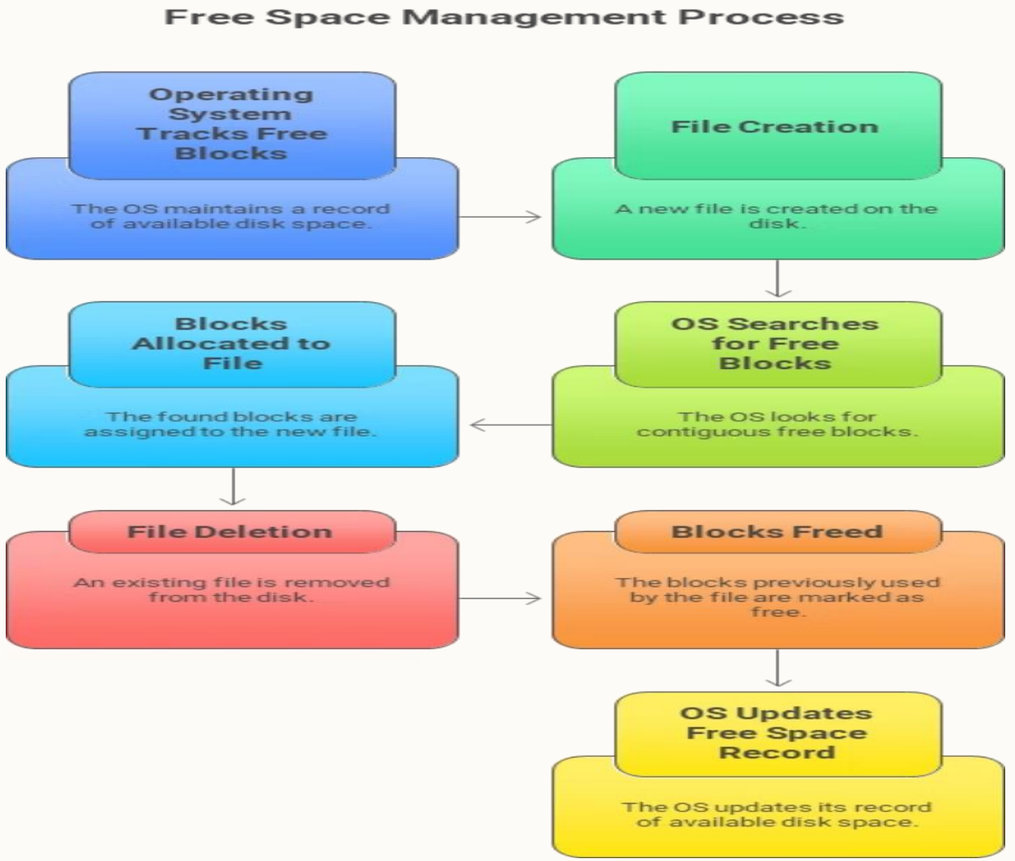
Monitoring & Recovery

Metrics, alerts, and repair tools

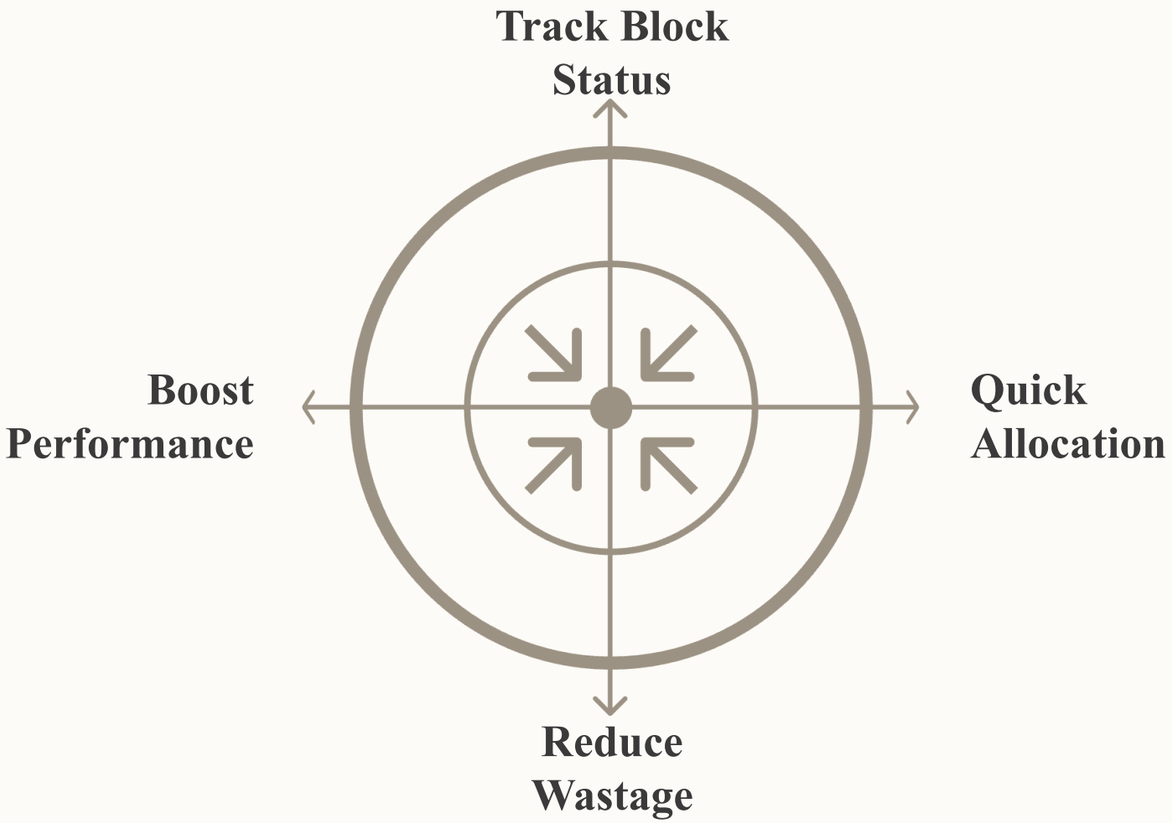


What is Free Space Management?

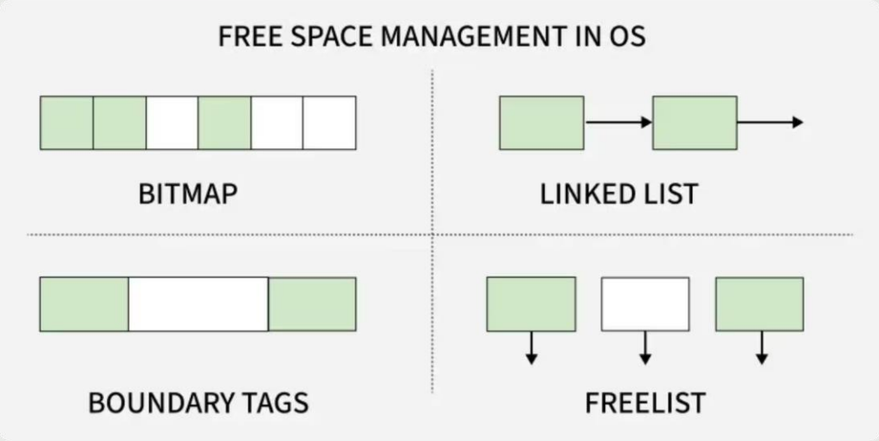
Free space management is a technique used by the operating system to keep track of free disk blocks so that space can be efficiently allocated and deallocated when files are created or deleted.



Need for Free Space Management

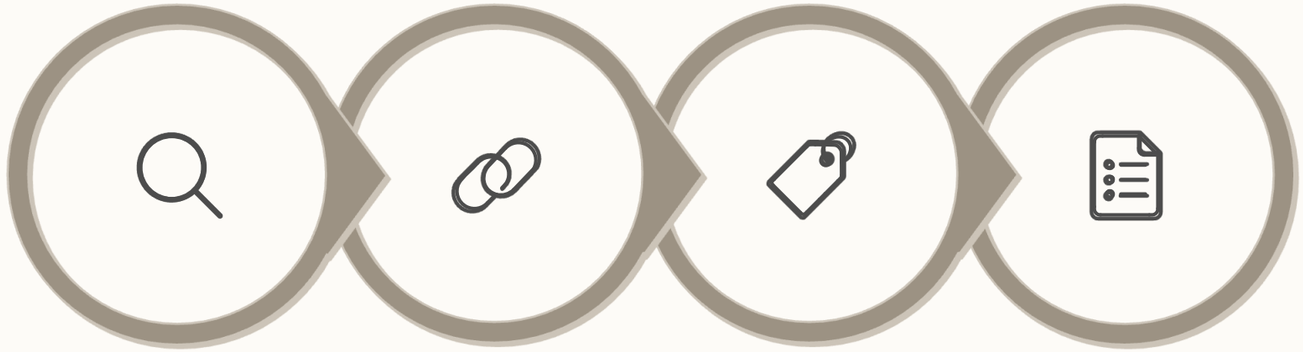


Methods of Free Space Management



Operating systems employ various techniques to manage free disk space, each with distinct advantages and trade-offs.

Understanding these methods helps in choosing the right approach for different storage scenarios.



Bit Vector **Linked List** **Boundary Tags** **Free List**

Bit Vector (Bitmap)

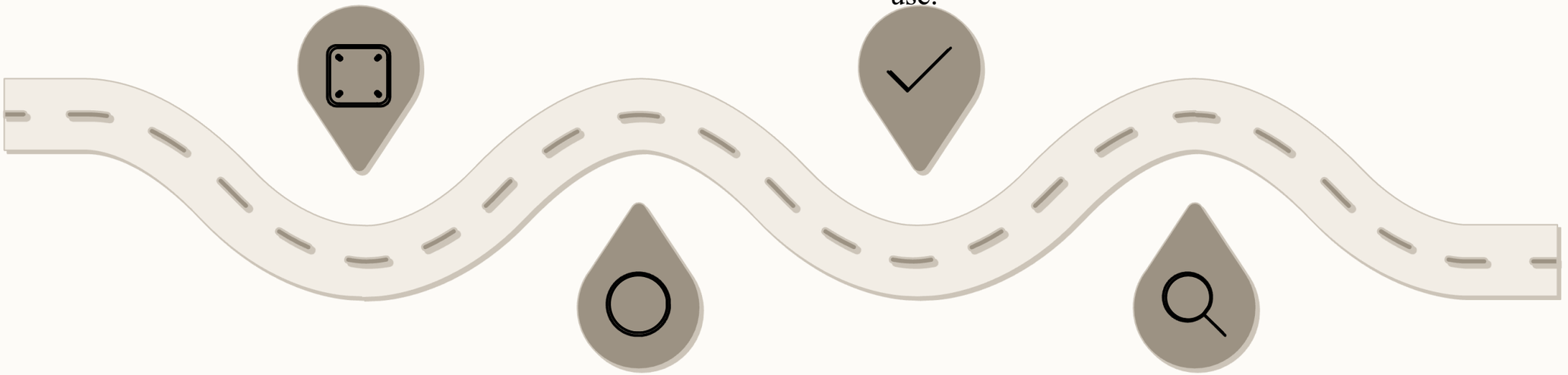
A Bitmap or Bit Vector is a series or collection of bits where each bit corresponds to a disk block. Each disk block is represented by a single bit in the vector.

Bit = Disk Block

Represents a single block on the disk.

1 = Allocated block

A bit value of one means the block is in use.

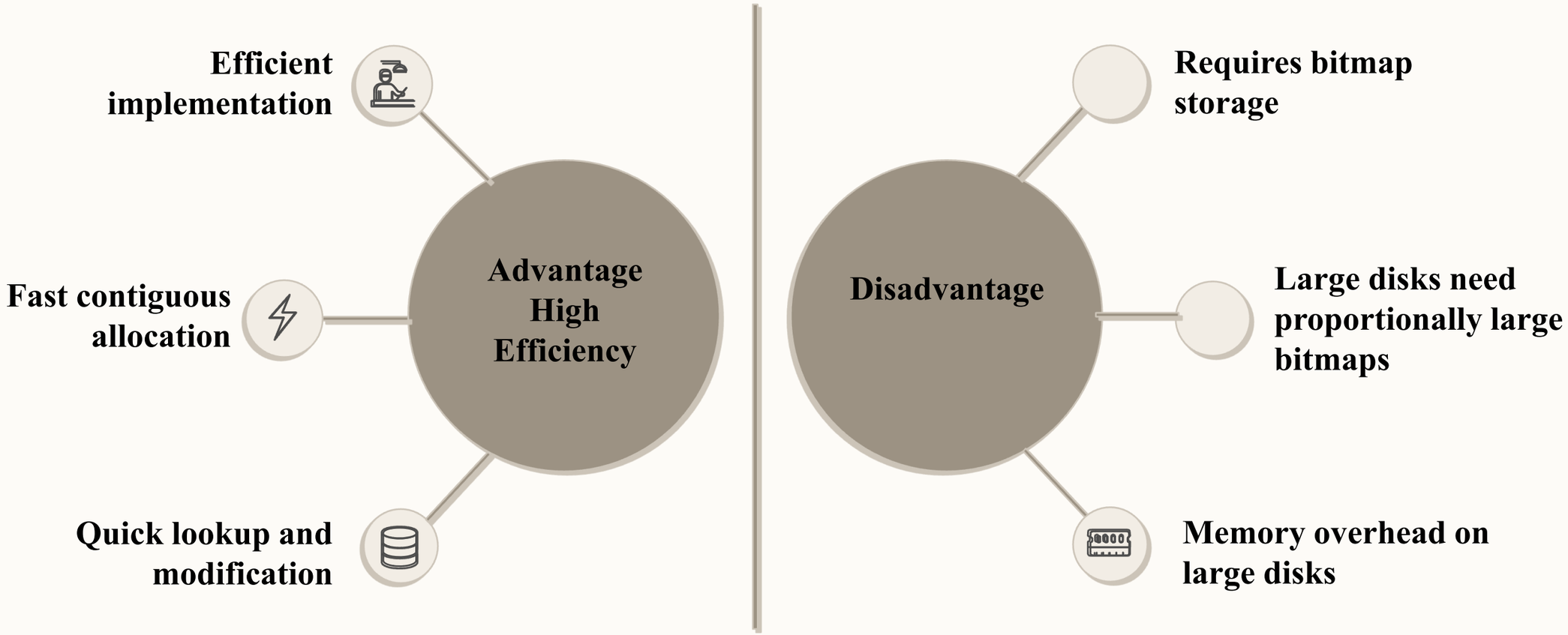


0 = Free block

A bit value of zero indicates the block is available.

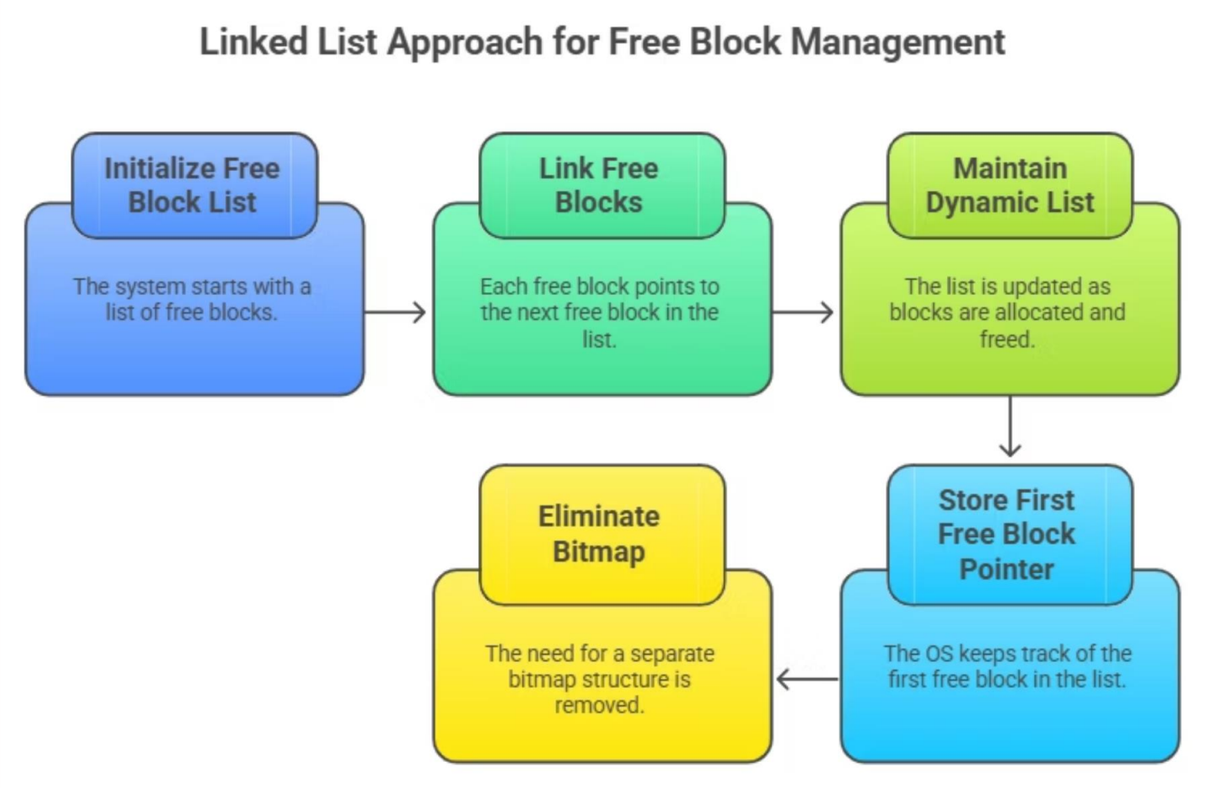
Find Contiguous Free

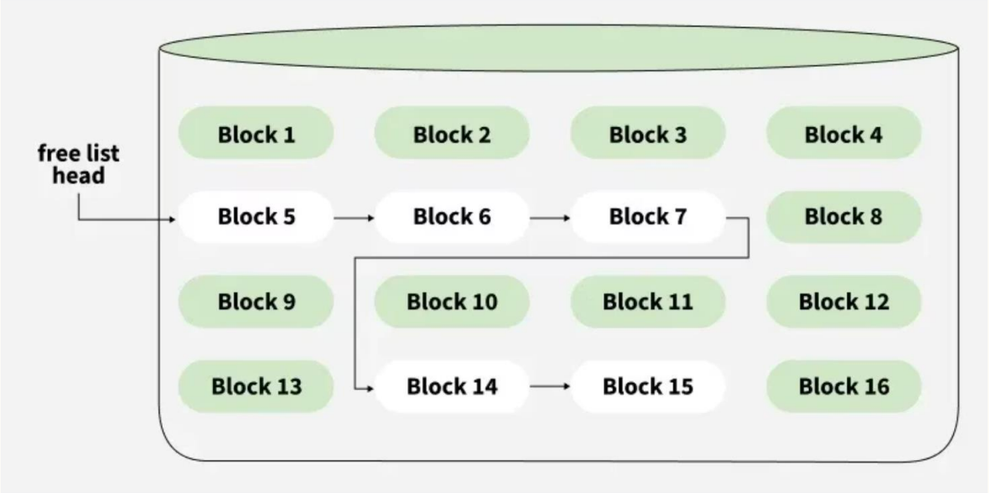
Search the bitmap for a sequence of free blocks.


Efficient implementation
Fast contiguous allocation
Quick lookup and modification
Advantage High Efficiency
Disadvantage
Requires bitmap storage
Large disks need proportionally large bitmaps
Memory overhead on large disks

METHOD 2

Linked List Approach

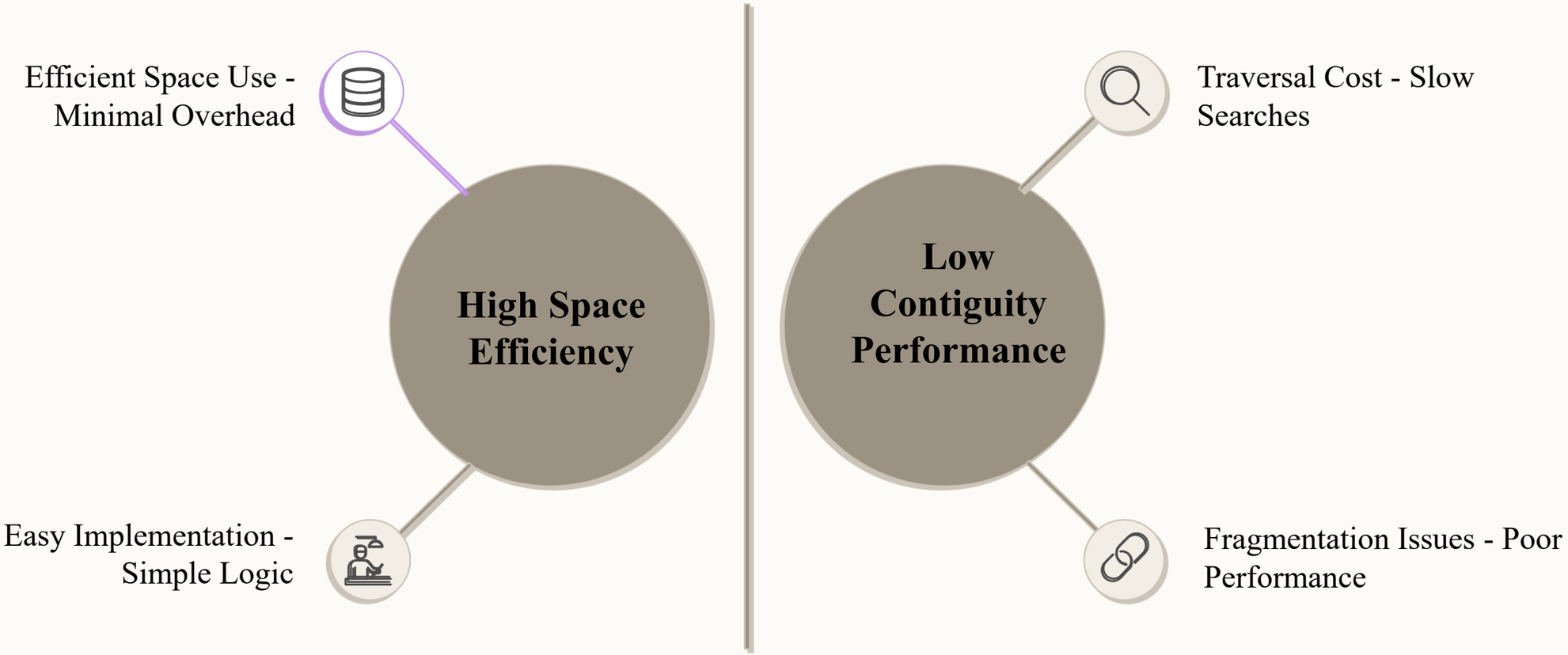




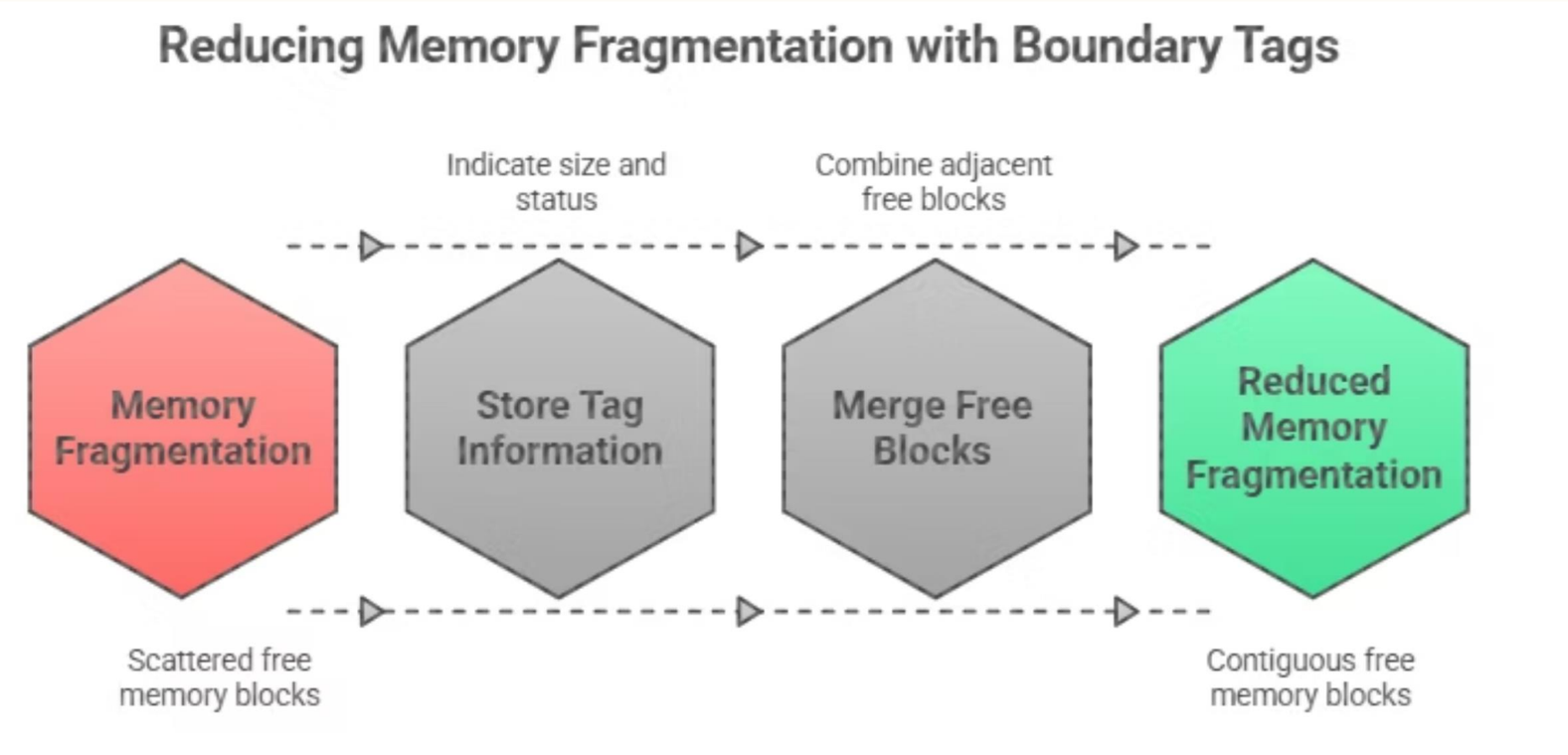
Linked List: Pros and Cons

Advantages

Disadvantages



Boundary Tags



METHOD 4

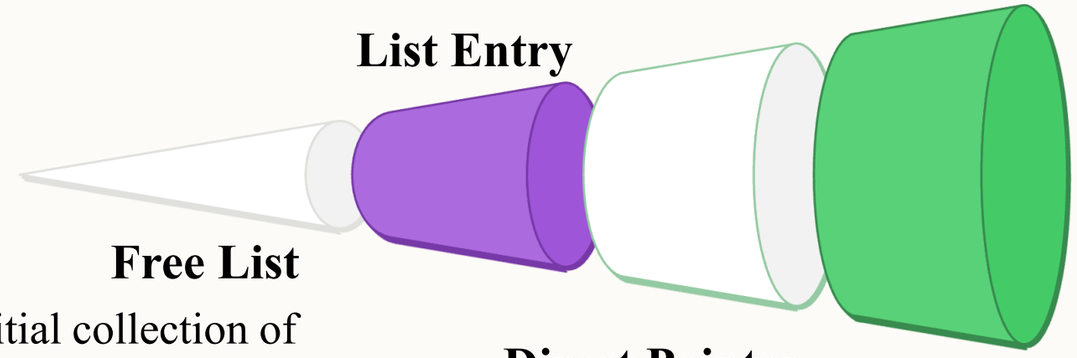
Free List

Reading or modifying the allocated memory block.

Access & Update

A specific block within the free list.

List Entry

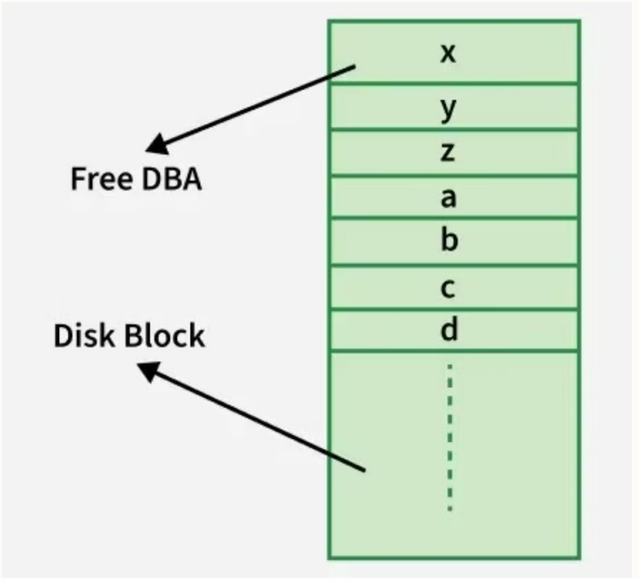


Free List

Initial collection of available memory blocks.

Direct Pointer

Reference used to locate the memory block.

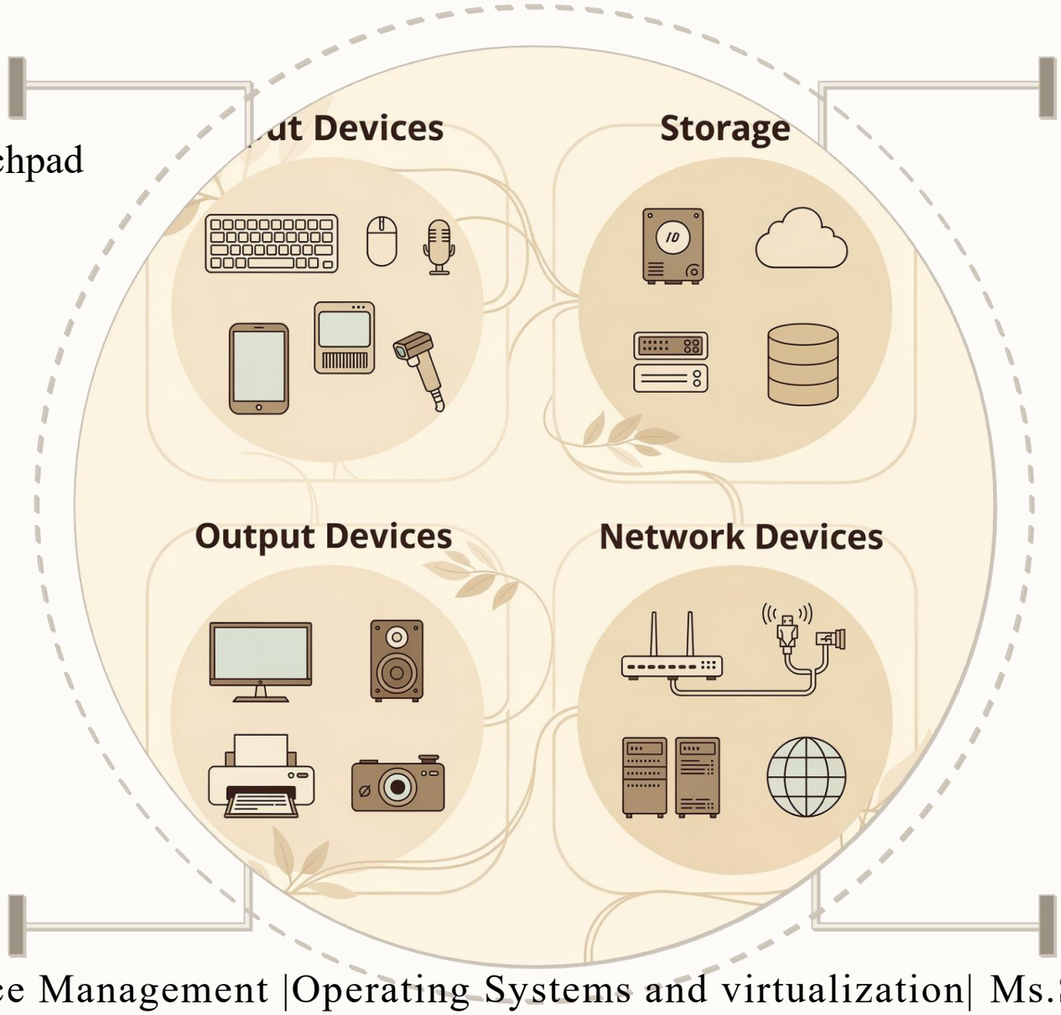


I/O Systems in Operating System

The I/O system is responsible for managing communication between the CPU, memory, and external devices. It serves as the critical bridge between software applications and hardware peripherals.

Input Devices

Mouse, keyboard, touchpad



Storage Devices

Disk drives, USB devices

Output Devices

Printers, LED screens, displays

Interfaces, audio I/O, ADC

Network Devices

Role and Objectives of I/O Systems

Main Role

Receive Request

An application request is formally submitted to I/O

Translate & Send

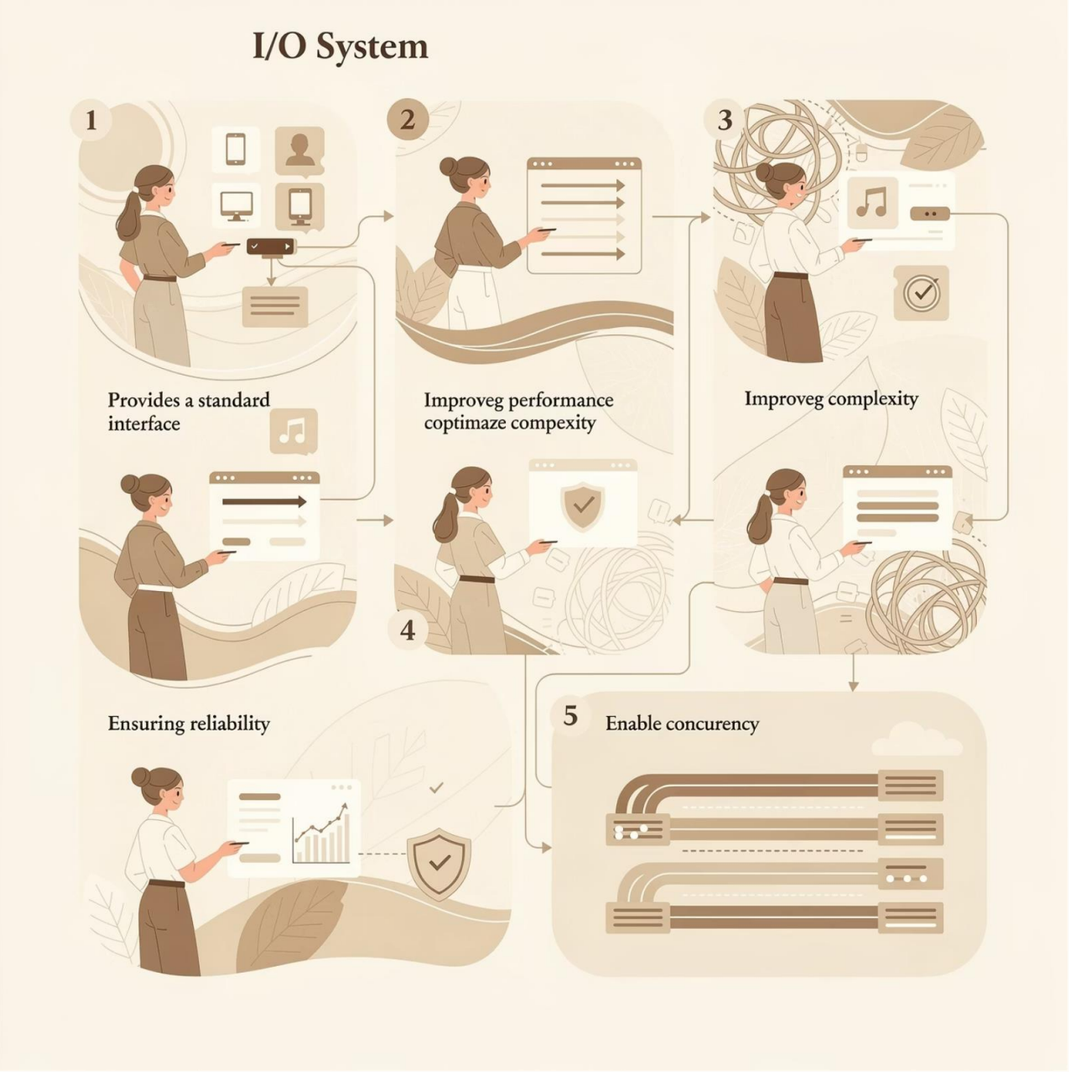
The request is processed by I/O and sent to the device.

Device Response

The hardware device sends back a response to I/O.

Relay to App

The response is relayed back to the application.



Categories of I/O Devices

Block Devices

Operate with blocks of data, transferring information in fixed-size chunks.

Examples: Hard disks, USB drives, Disk-On-Key

These devices are typically used for storage and allow random access to data blocks.

Character Devices

Operate with single characters or bytes, transferring data as a stream.

Examples: Serial ports, parallel ports, sound cards

These devices handle sequential data streams and are often used for communication.

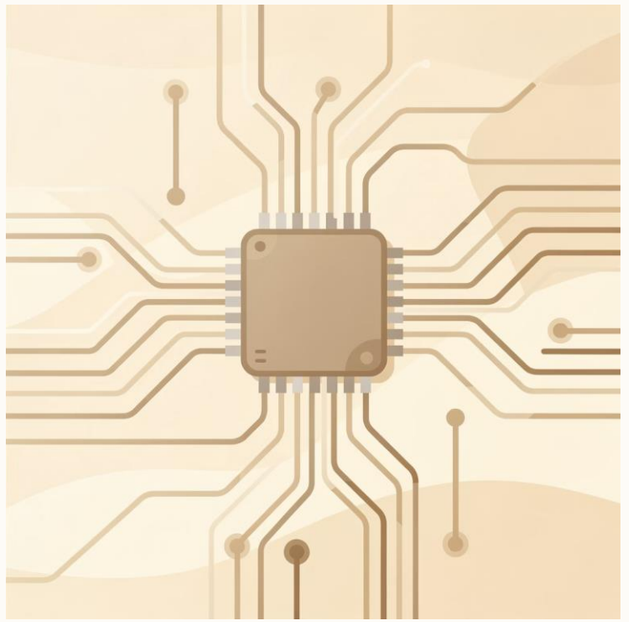
Device Controllers and Device Drivers

Device Driver

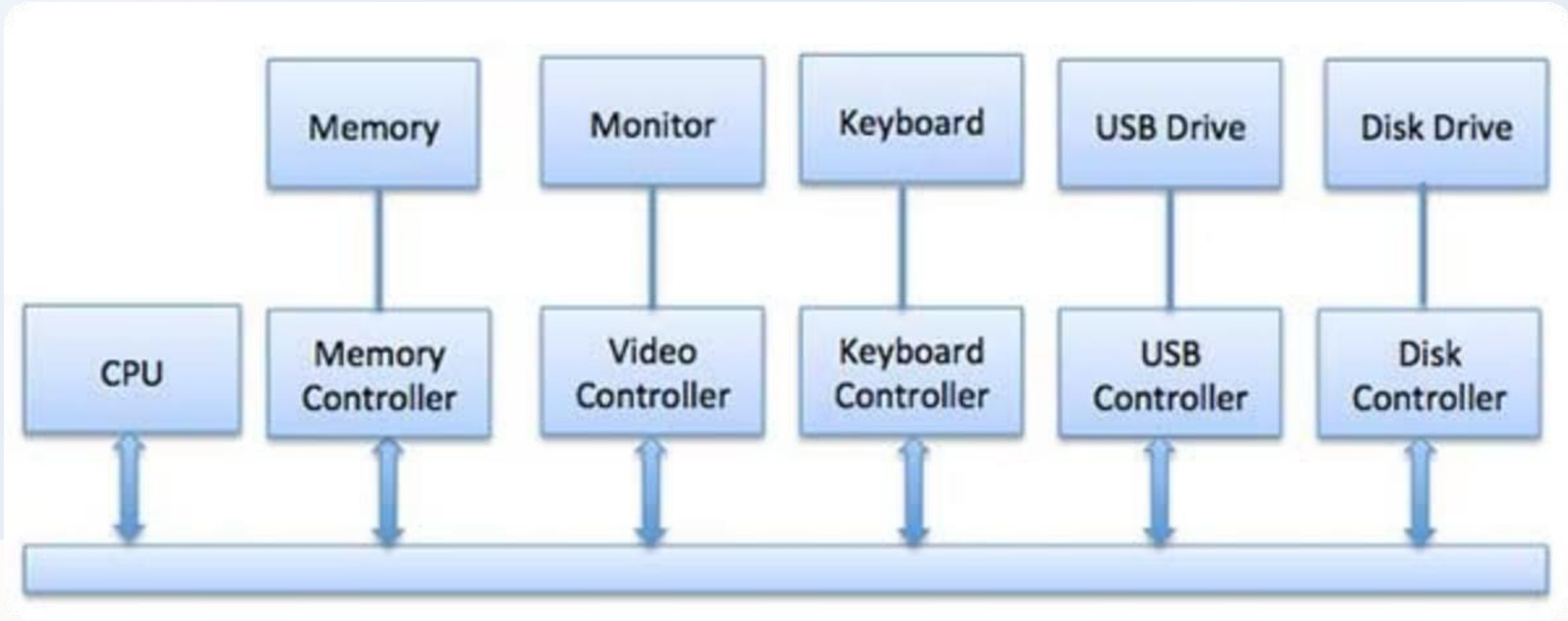
Software module that allows the OS to control a device. It acts as a translator between the operating system and the hardware.

Device Controller

Hardware component that interfaces between the device and driver. It manages the physical operation of the device.



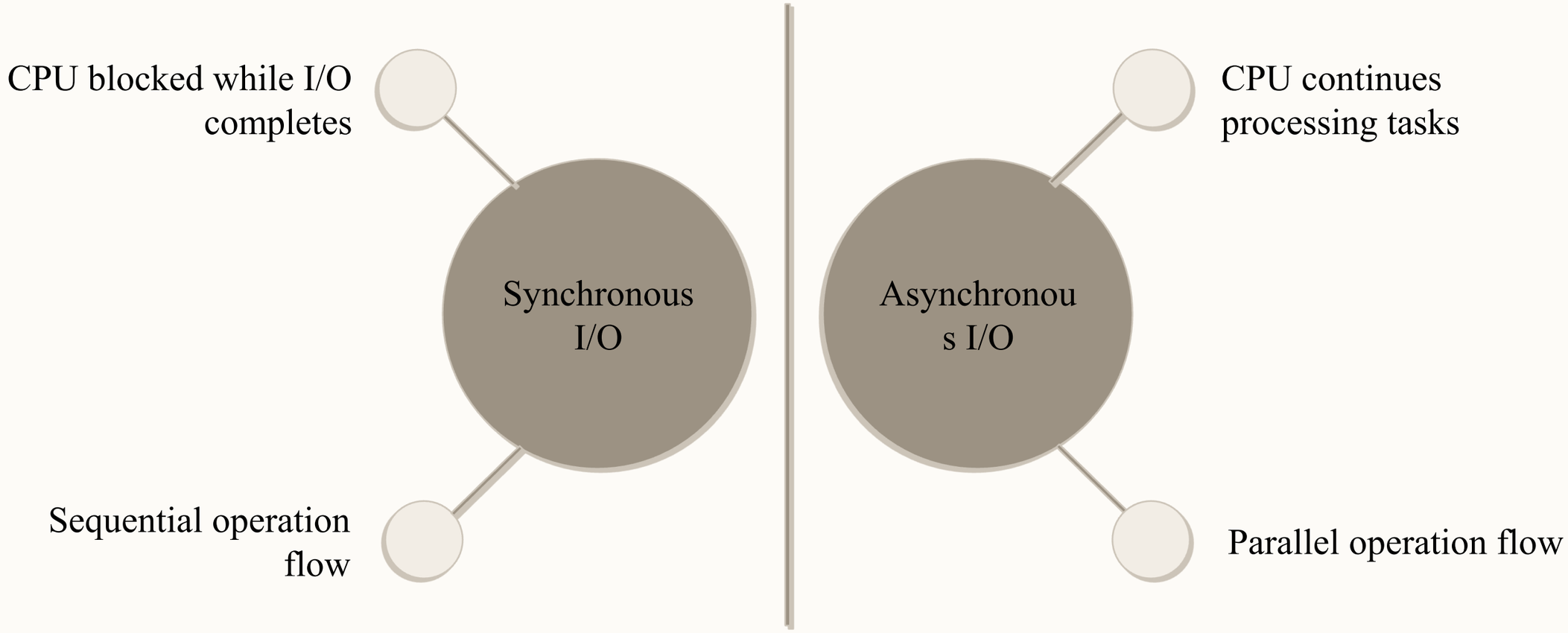
- Converts serial bit streams into blocks of bytes
- Performs error detection and correction
- May control multiple devices simultaneously



Device Controller Architecture

This diagram illustrates the relationship between device drivers, device controllers, and the operating system, showing how data flows from applications through the OS kernel to physical hardware devices.

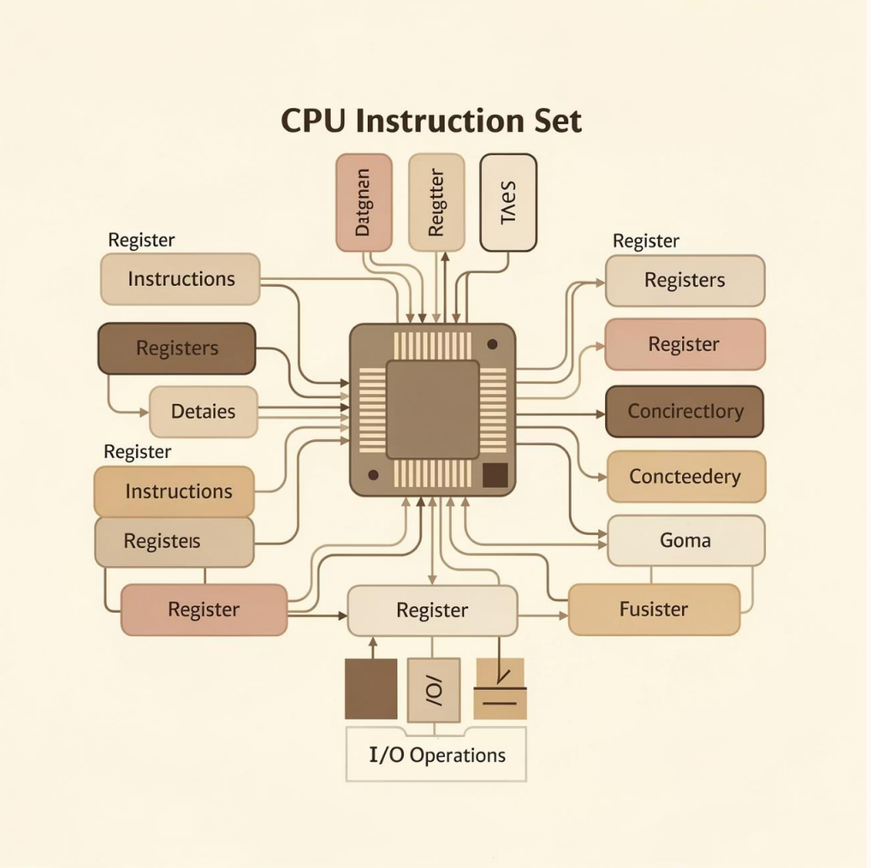
I/O Types: Synchronous vs Asynchronous



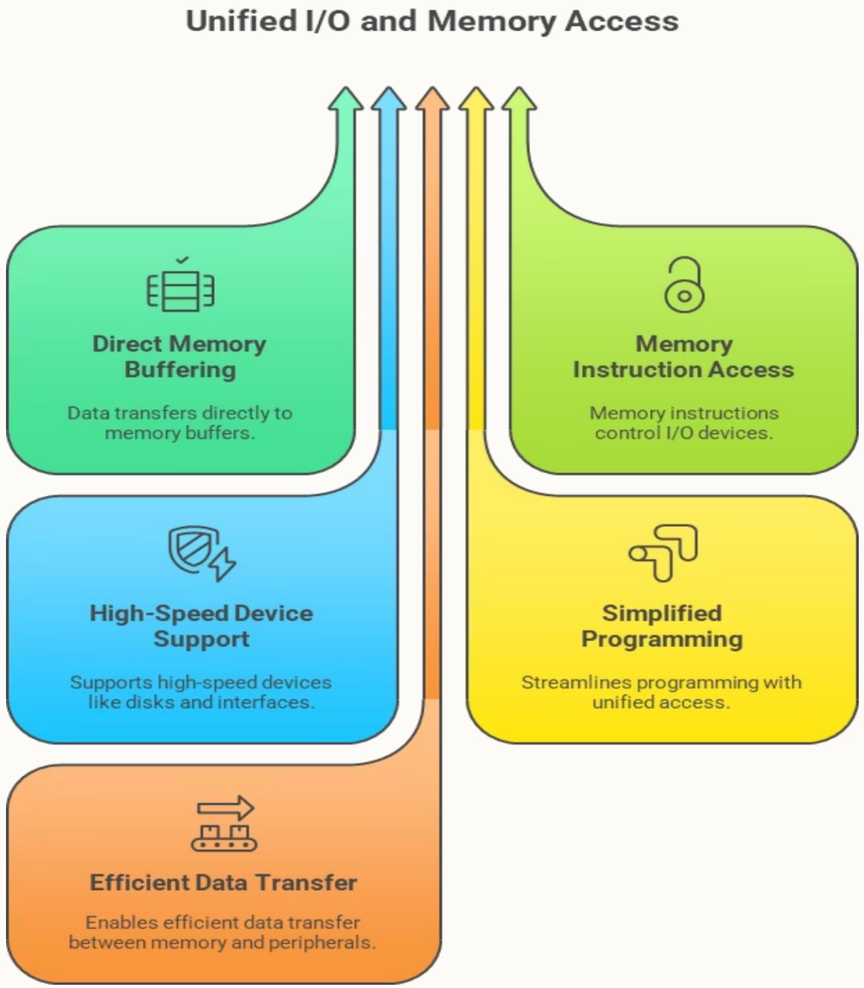
Special Instruction I/O

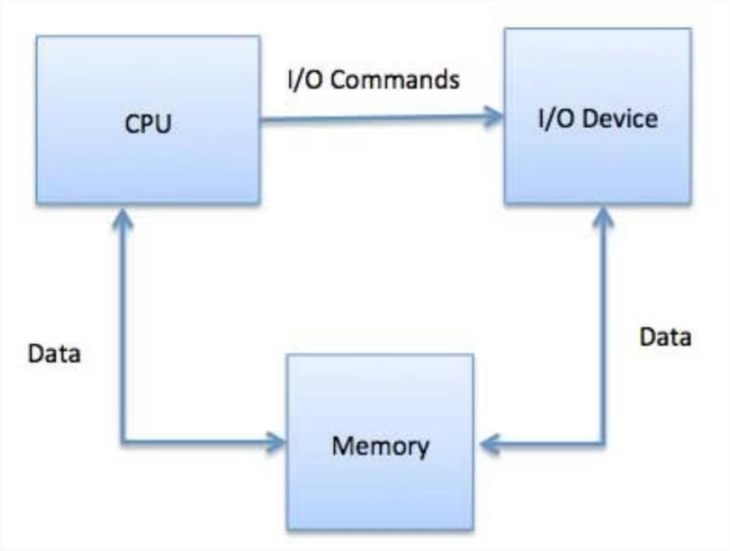
CPU uses dedicated instructions to read from or write to devices. This method employs special machine instructions specifically designed for I/O operations.

Example: Sending a command to a printer or reading a character from a keyboard.

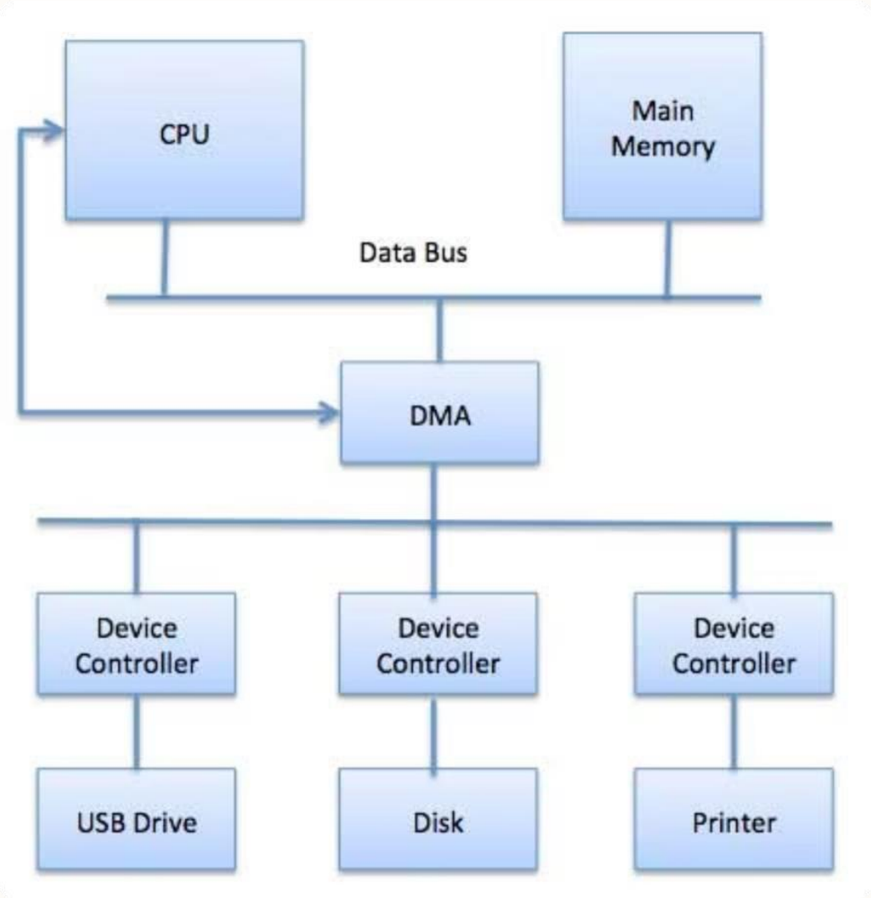


Memory-Mapped I/O





Direct Memory Access (DMA)



CPU delegates data transfer between memory and device to DMA controller. The CPU is involved only at the start and end of transfer, significantly reducing CPU overhead for fast devices like disks.

Step 1: Initiation

Device driver instructs transfer to buffer X

Step 2: Start Transfer

Disk controller starts DMA transfer

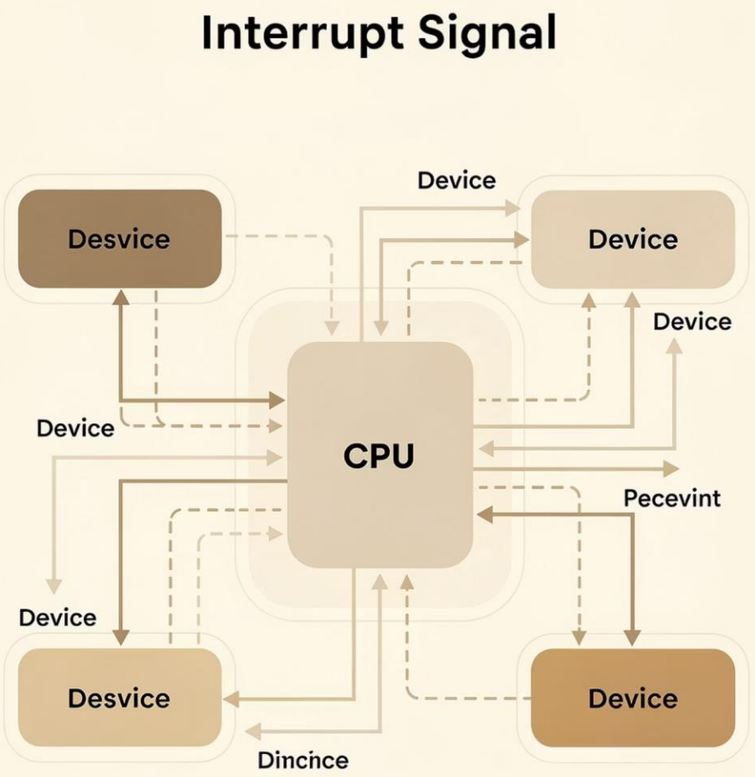
Step 3: Data Movement

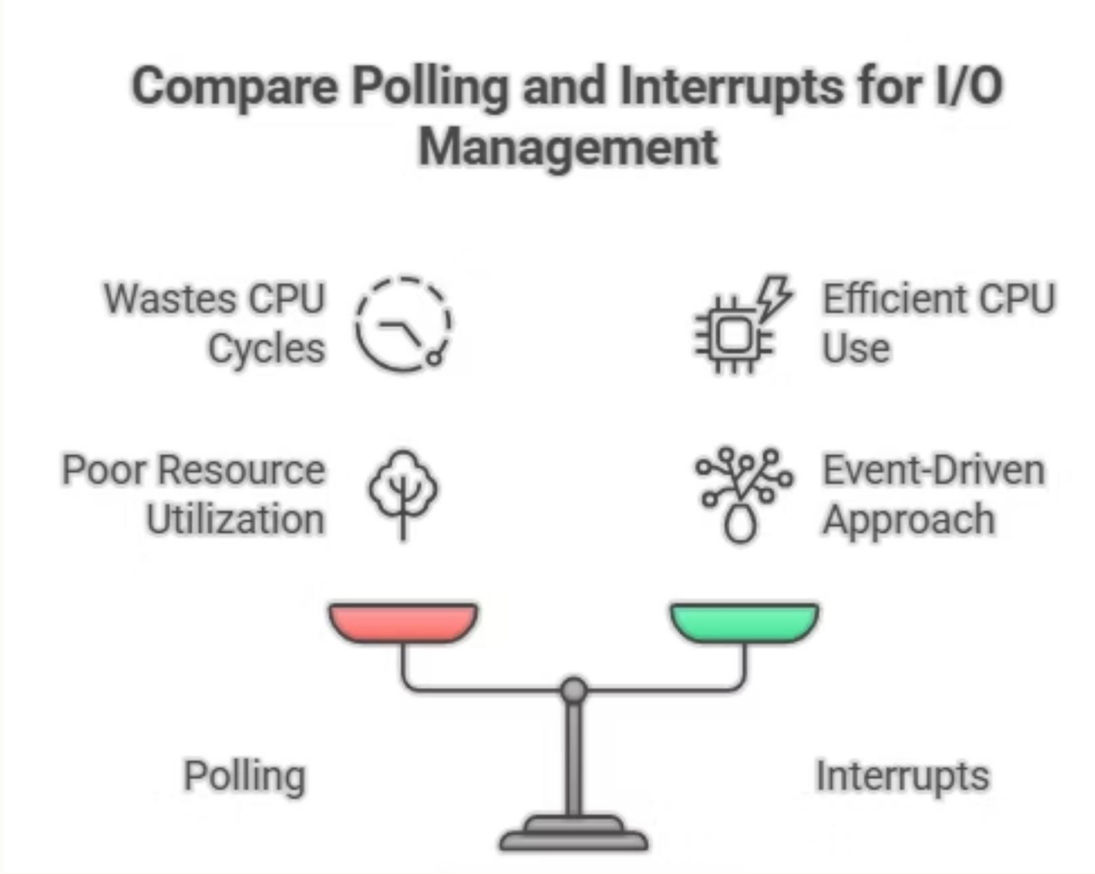
DMA controller moves bytes to memory buffer, updating counters and addresses

Step 4: Completion

After completion, DMA sends interrupt to CPU

Polling vs Interrupts





1. Problem Statement

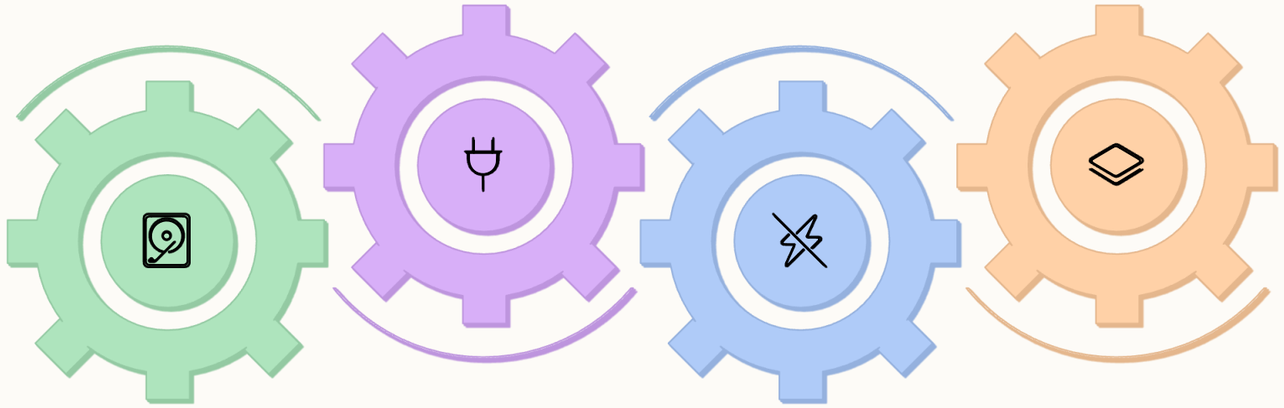
The operating system must **efficiently manage disk space** and **communicate with multiple I/O devices**.

Robust I/O Drivers

Ensuring reliable communication between hardware and software.

Fragmentation Control

Managing scattered data to maintain system performance.



Efficient Disk Allocation

Optimizing how data is stored on physical media.

DMA & Offloading

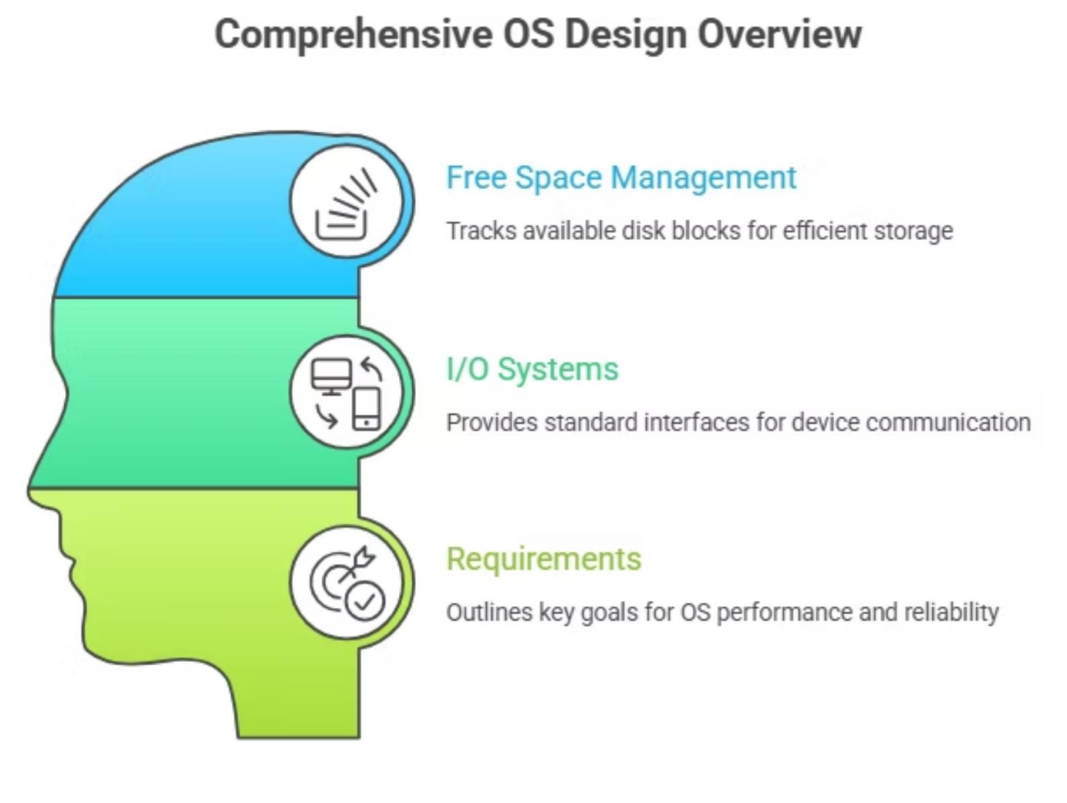
Freeing up CPU resources by delegating data transfers.


Operating System Challenges
The OS must efficiently manage disk space & handle multiple I/O devices.
Disk Space Management
Track Free Disk Blocks
Avoid Fragmentation & Waste
I/O Device Communication
Fast & Reliable I/O with
Keyboard
Disk
Printer
Network
Minimize CPU Usage
CPU
Use Buffering & Caching
Boost Performance
Efficient Disk & I/O Management for High Performance

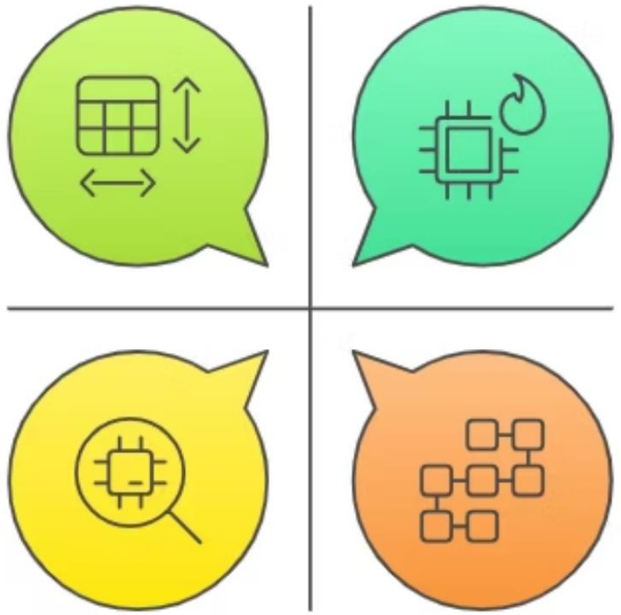
2. Define (Understanding the Problem)

Free Space Management:

- OS needs to know **which disk blocks are free or occupied**.
- Requires methods that support **efficient allocation, deallocation, and file growth**.



Ideation for OS Solutions



1

Bit Vector

Bit Vector provides efficient solutions with minimal complexity.

2

Interrupts

Interrupts offer high efficiency with moderate complexity.

3

Polling

Polling is a low-complexity but inefficient I/O method.

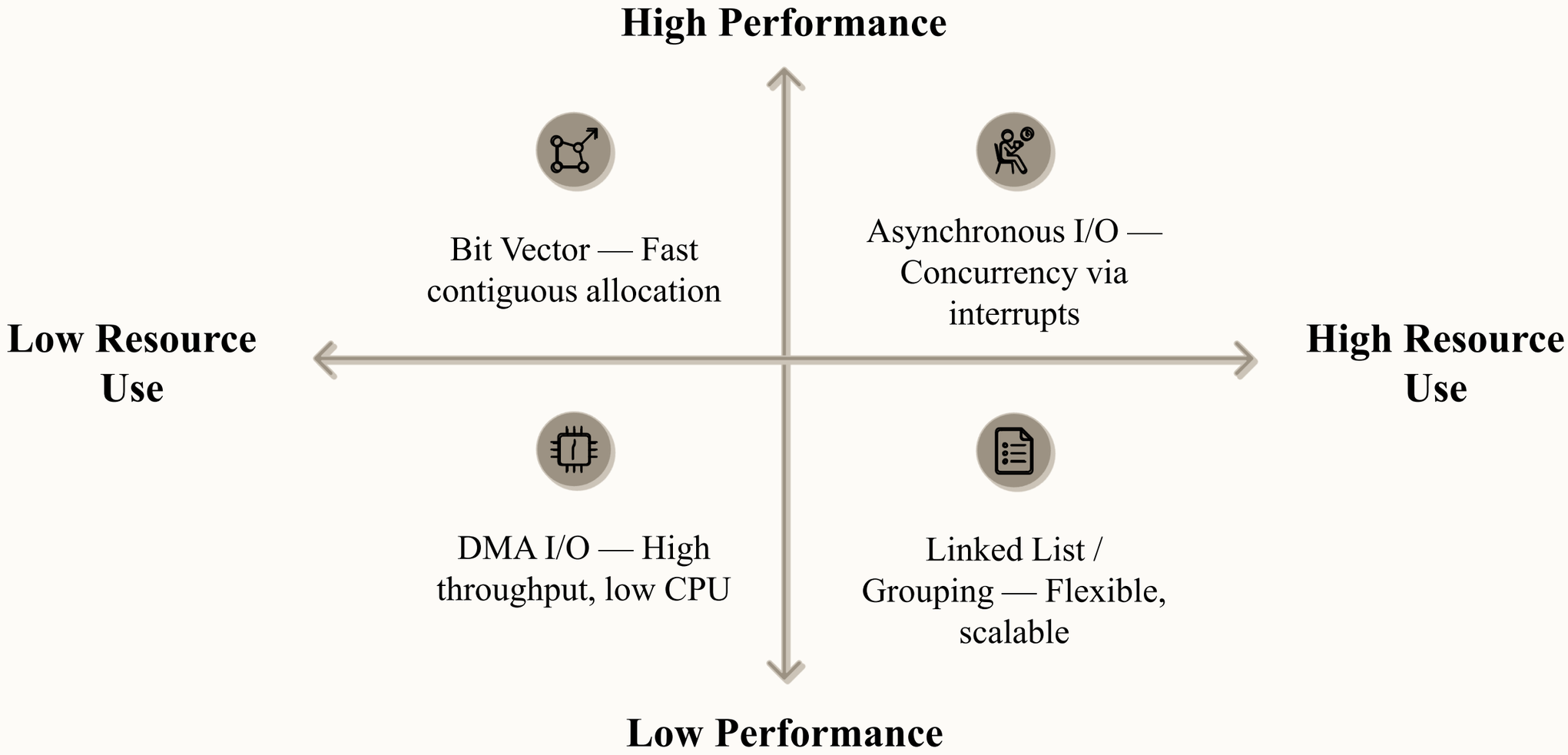
4

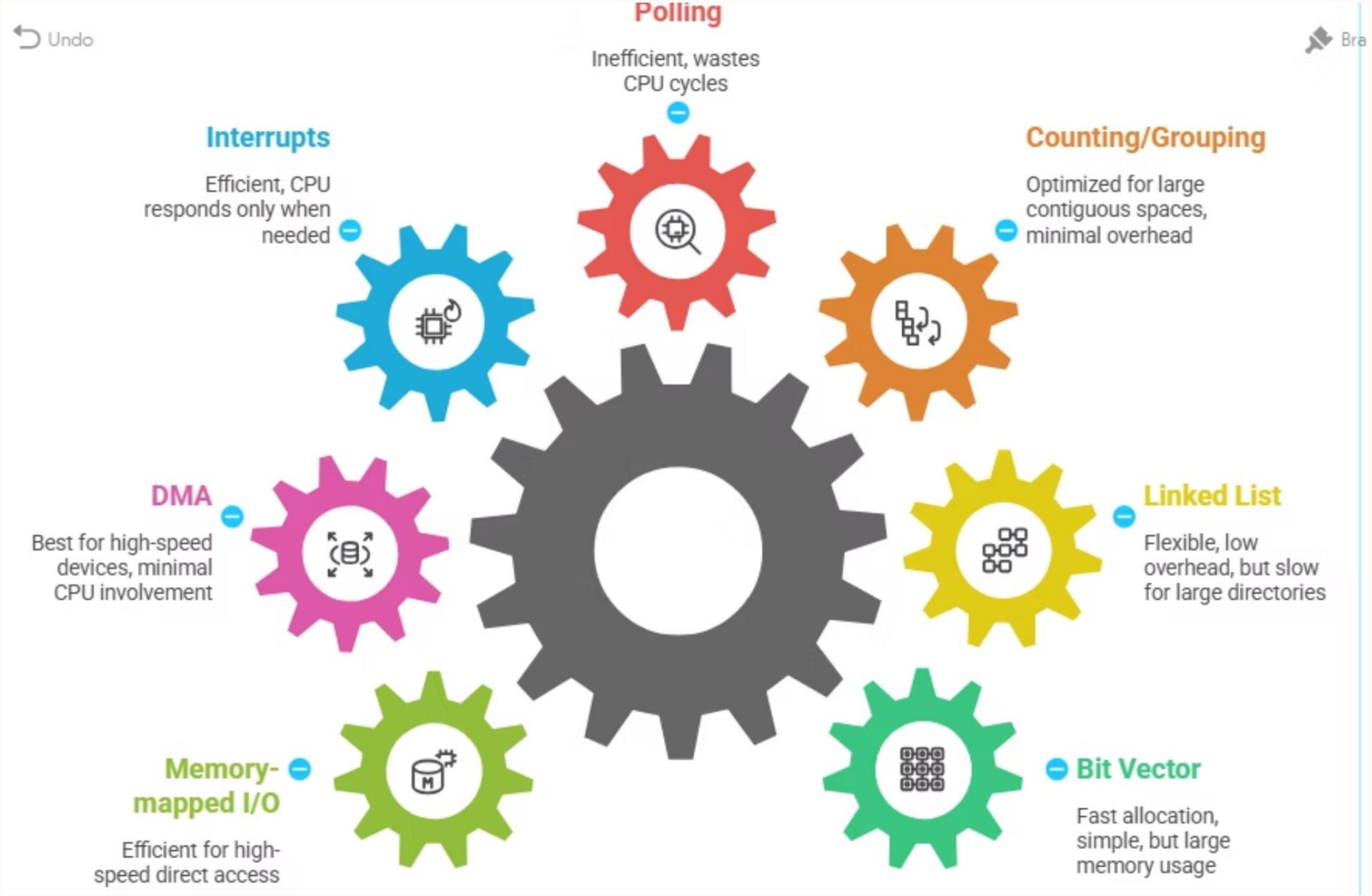
Grouping

Grouping is a complex but less efficient free space management technique.

4. Prototype (Implementation Concepts)

Exploring various approaches to implement efficient free space management and robust I/O systems, balancing performance with resource utilization.

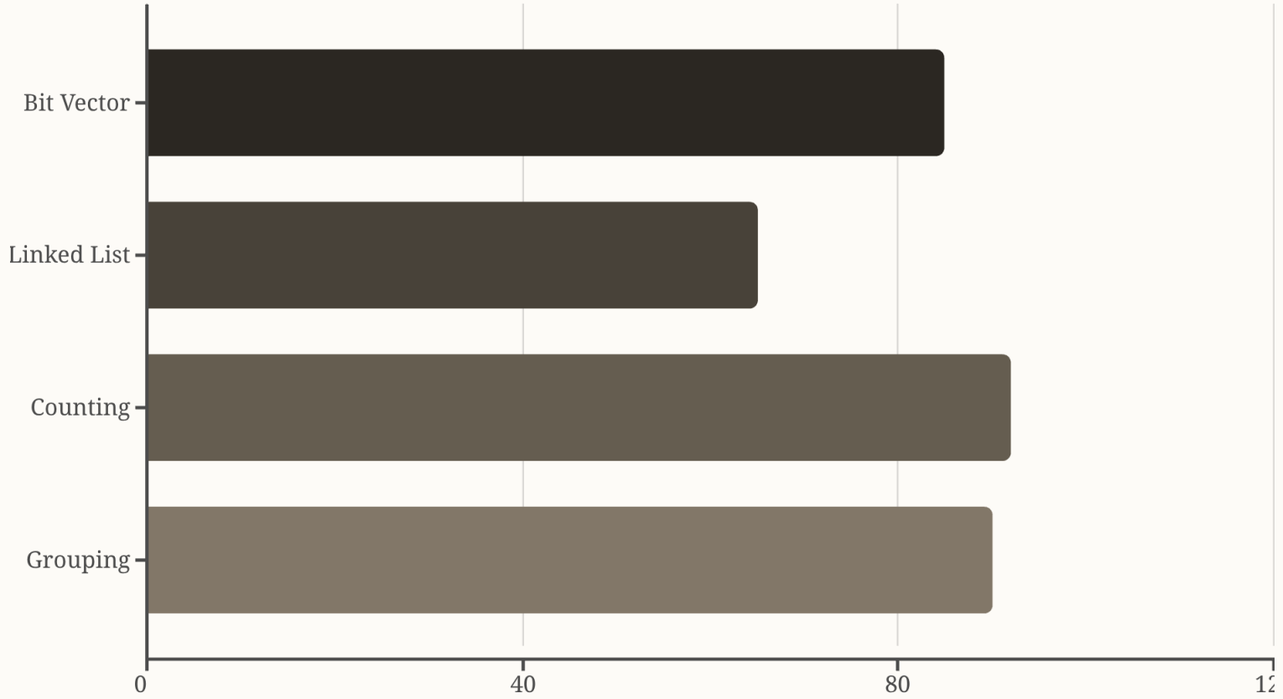


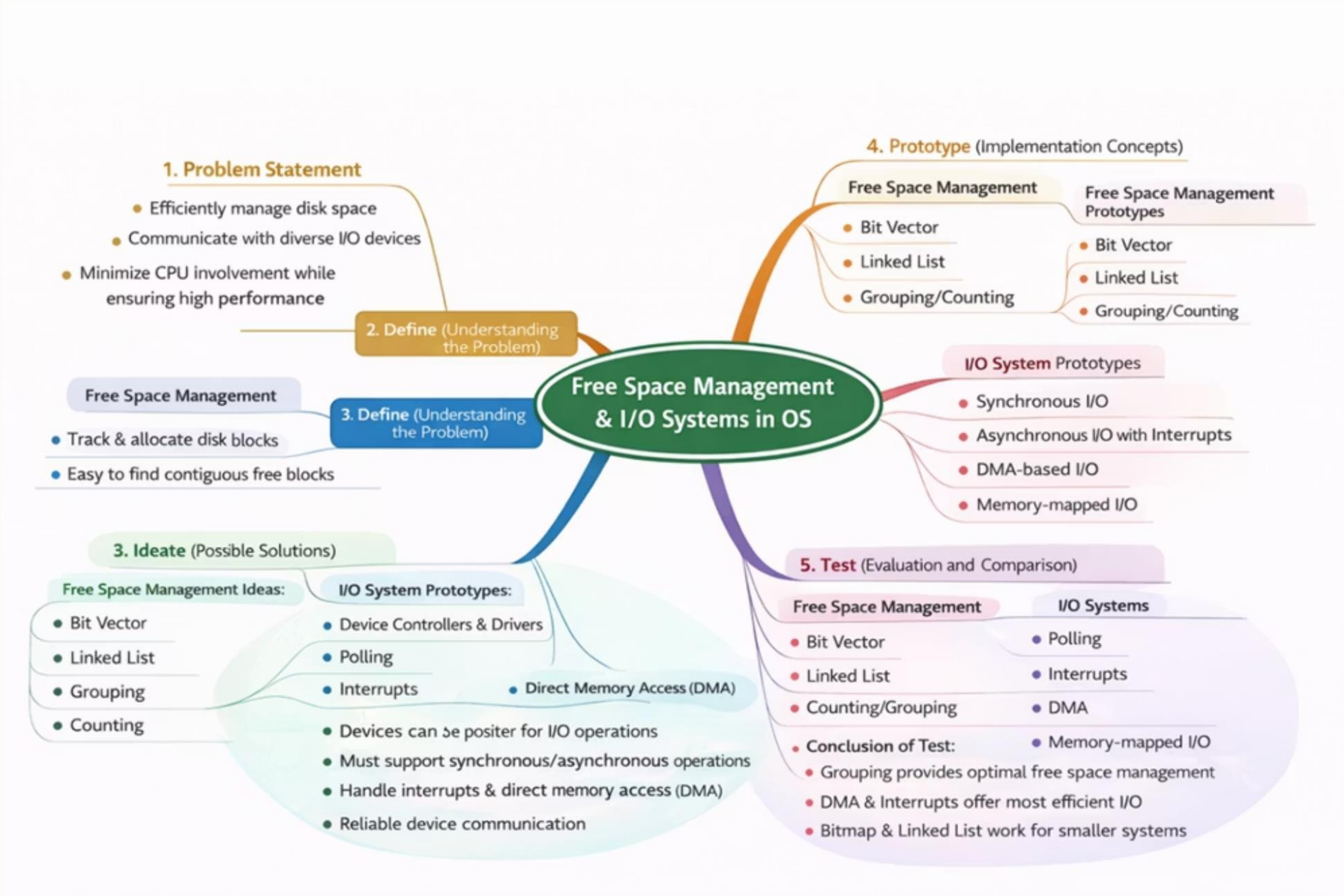

Undo
Bra
Polling
Inefficient, wastes CPU cycles
Interrupts
Efficient, CPU responds only when needed
Counting/Grouping
Optimized for large contiguous spaces, minimal overhead
DMA
Best for high-speed devices, minimal CPU involvement
Linked List
Flexible, low overhead, but slow for large directories
Memory-mapped I/O
Efficient for high-speed direct access
Bit Vector
Fast allocation, simple, but large memory usage

Evaluation and Comparison

After prototyping various approaches, we can now evaluate their performance characteristics and determine optimal solutions for different scenarios.

Free Space Management Analysis




Free Space Management & I/O Systems in OS
1. Problem Statement
Efficiently manage disk space
Communicate with diverse I/O devices
Minimize CPU involvement while ensuring high performance
2. Define (Understanding the Problem)
Free Space Management
Track & allocate disk blocks
Easy to find contiguous free blocks
3. Define (Understanding the Problem)
3. Ideate (Possible Solutions)
Free Space Management Ideas:
Bit Vector
Linked List
Grouping
Counting
I/O System Prototypes:
Device Controllers & Drivers
Polling
Interrupts
Direct Memory Access (DMA)
Devices can be positer for I/O operations
Must support synchronous/asynchronous operations
Handle interrupts & direct memory access (DMA)
Reliable device communication
4. Prototype (Implementation Concepts)
Free Space Management
Bit Vector
Linked List
Grouping/Counting
Free Space Management Prototypes
Bit Vector
Linked List
Grouping/Counting
I/O System Prototypes
Synchronous I/O
Asynchronous I/O with Interrupts
DMA-based I/O
Memory-mapped I/O
5. Test (Evaluation and Comparison)
Free Space Management
Bit Vector
Linked List
Counting/Grouping
I/O Systems
Polling
Interrupts
DMA
Memory-mapped I/O
Conclusion of Test:
Grouping provides optimal free space management
DMA & Interrupts offer most efficient I/O
Bitmap & Linked List work for smaller systems