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 5:Storange Management

Storage Management in OS

Storage management is a critical function in modern operating systems, forming the foundation for organizing, accessing, and protecting data across diverse storage devices.

Organizes Data

Arranges information across disks and SSDs

Manages Access

Controls how users and apps access data

Protects Data

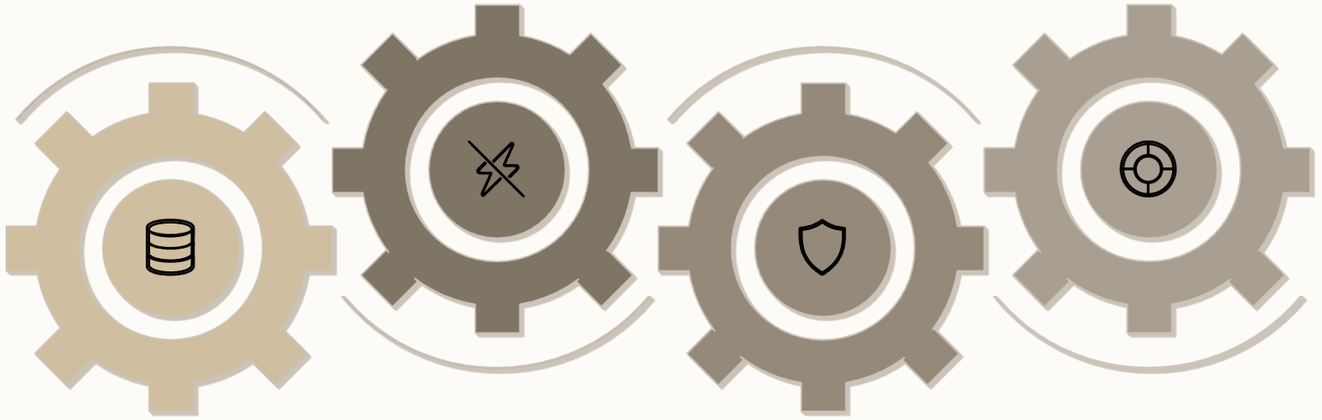
Ensures integrity and security of storage

Understanding Storage Management

Storage management is the process of organizing, storing, and managing data on secondary storage devices like hard disks and SSDs. The file system serves as the critical component of the OS that manages how data is stored, accessed, and organized on these disks.

Fast Access
Enable quick file retrieval and low latency

Reliability
Ensure recovery and maintain data integrity

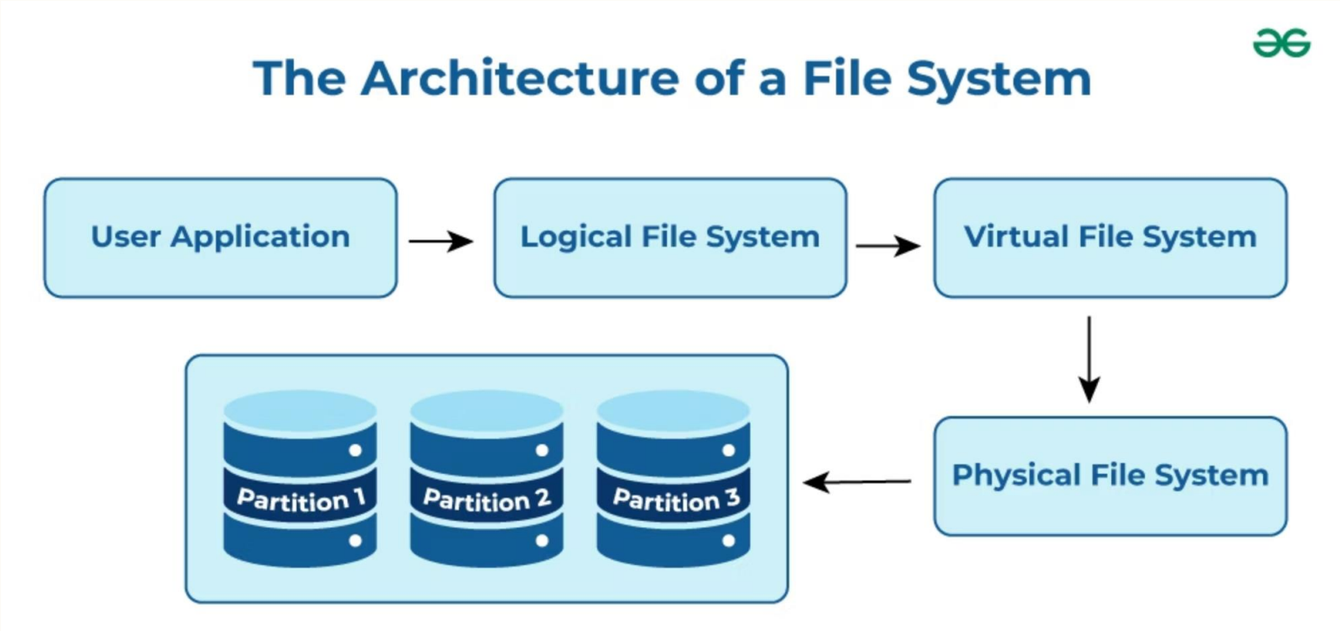


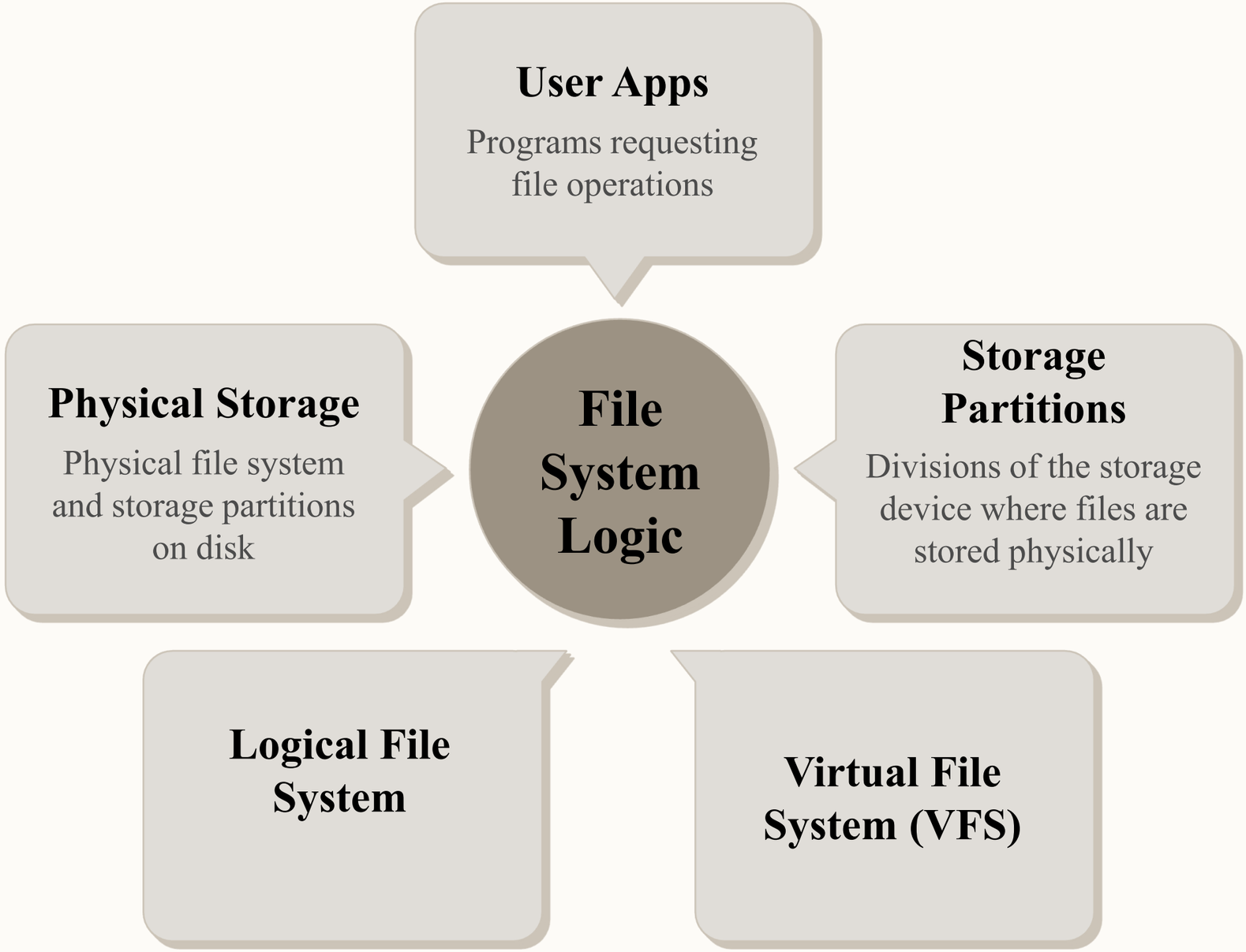
Space Efficiency
Maximize storage capacity with minimal waste

Data Protection
Prevent unauthorized access and breaches

File System Architecture

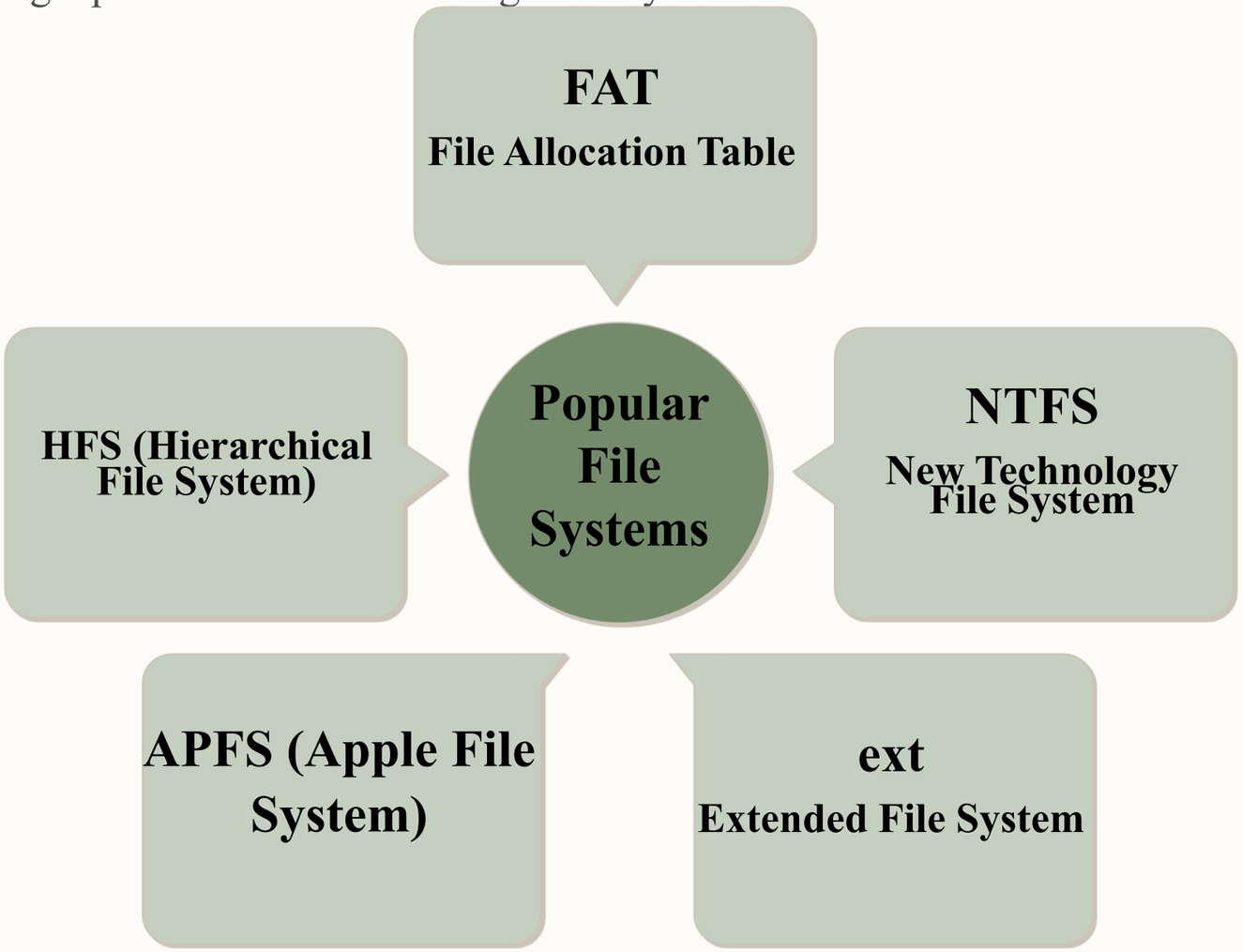
File systems are a crucial part of any operating system, providing a structured way to store, organize and manage data on storage devices such as hard drives, SSDs and USB drives. The architecture consists of multiple layers working together seamlessly.



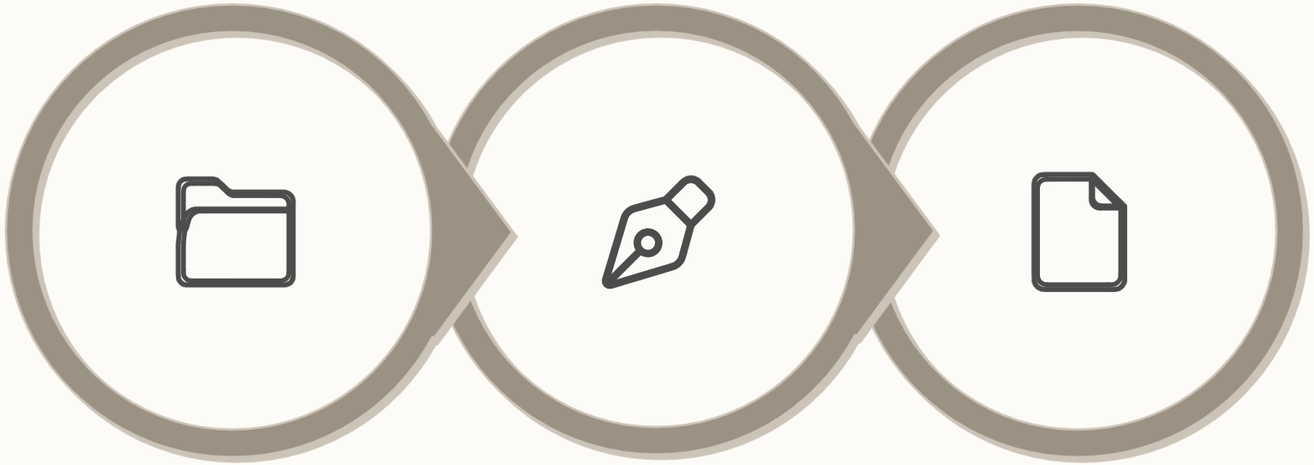

User Apps
Programs requesting file operations
Physical Storage
Physical file system and storage partitions on disk
File System Logic
Storage Partitions
Divisions of the storage device where files are stored physically
Logical File System
Virtual File System (VFS)

Popular File Systems

Different operating systems employ various file system architectures, each designed with specific features and optimizations for their target platforms. Understanding these systems is essential for effective storage management.



Issues Handled By File System



Free Space Management

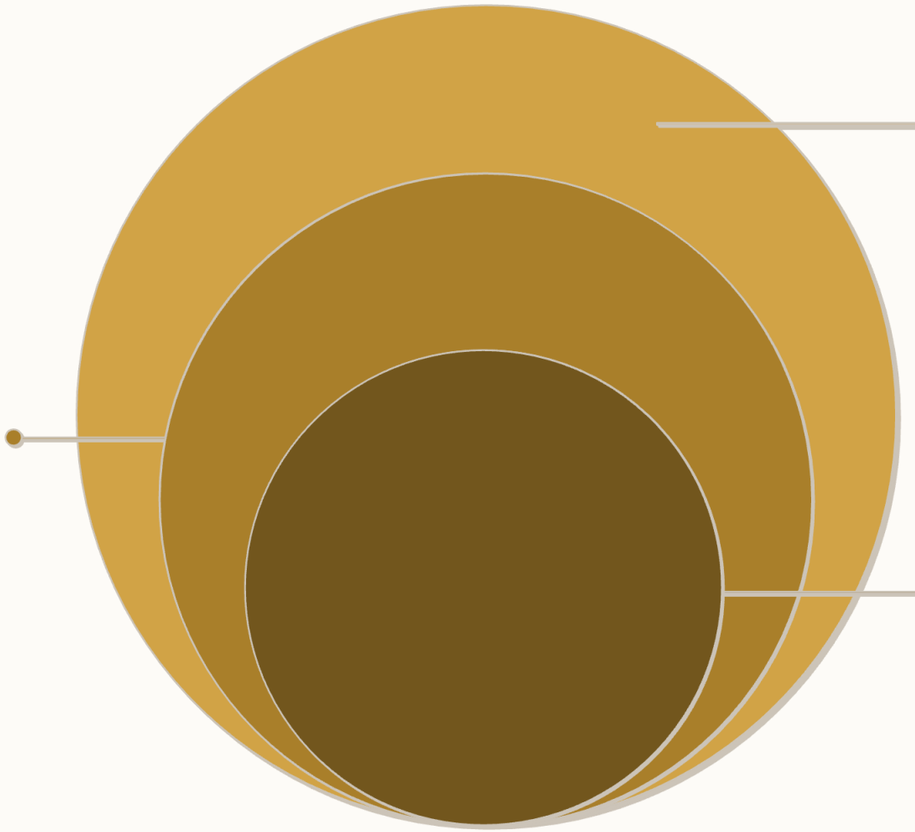
Space Reallocation

File Placement Strategy

File Directories

The collection of files is a file directory and contains information about the files, including attributes, location, and other metadata.

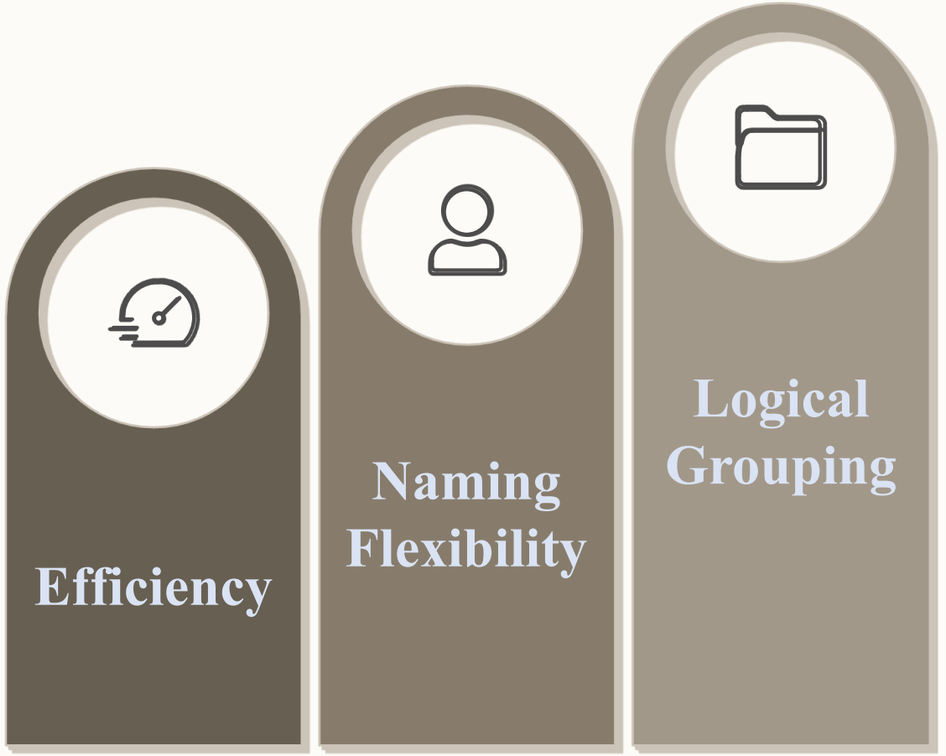
Metadata & Attributes
Names, sizes, permissions, timestamps



OS Management
Location mapping and access optimization

File Directory
Central repository of files and metadata

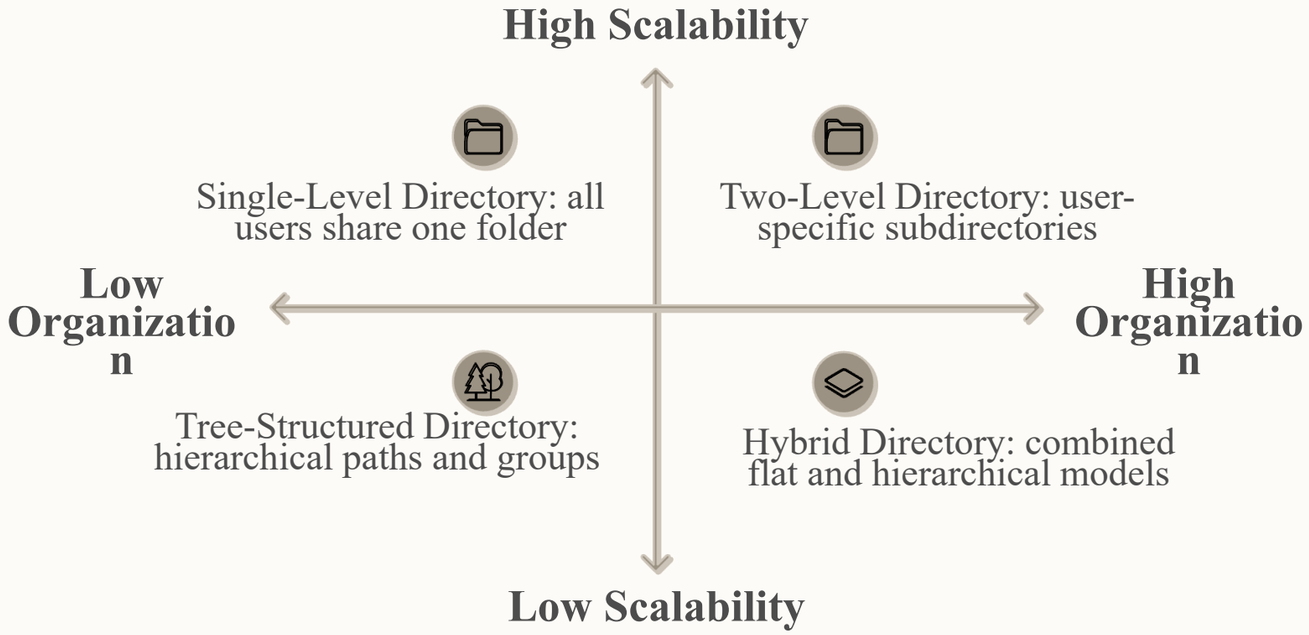
Advantages of Maintaining Directories

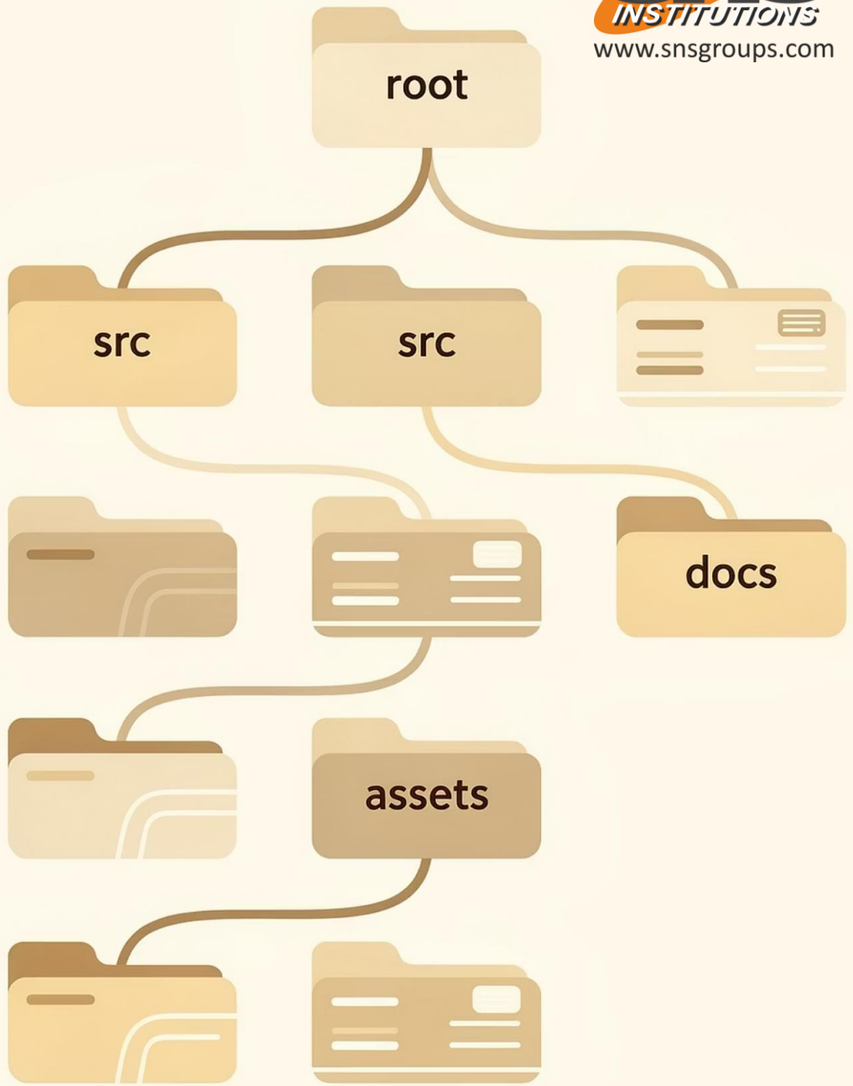


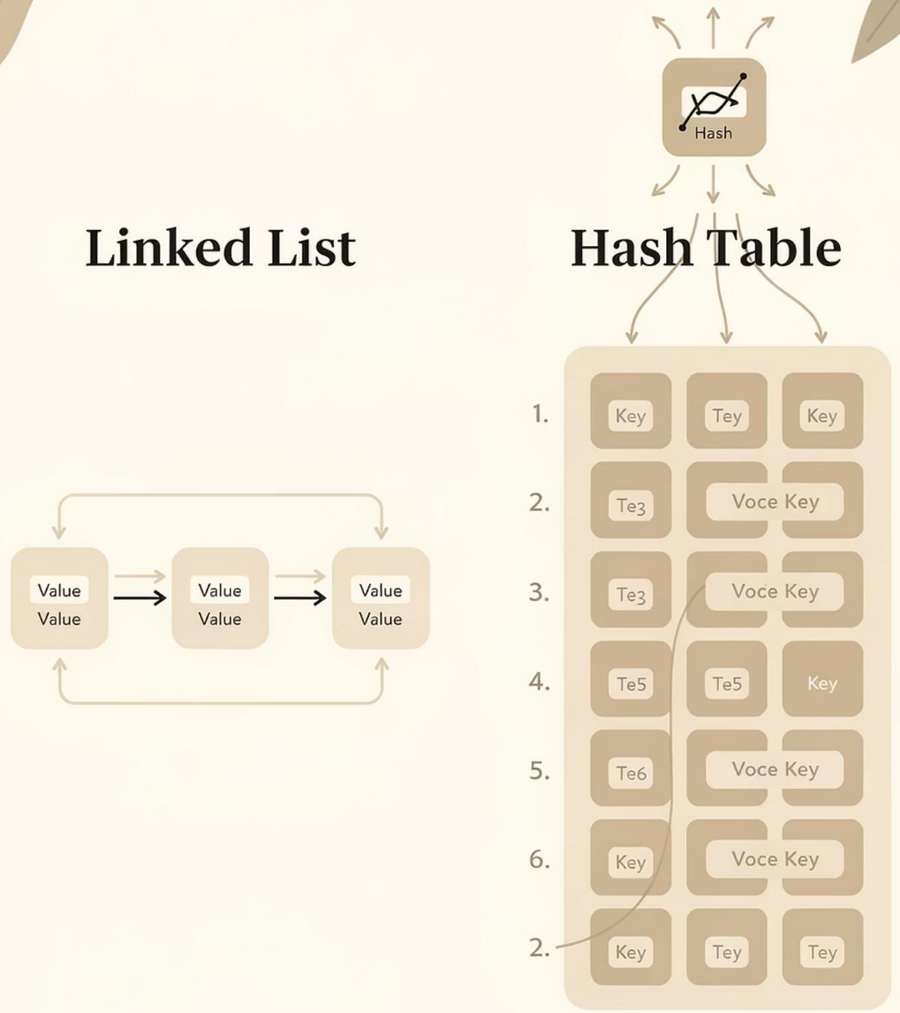
DIRECTORY STRUCTURES

Structures of Directory

Operating systems implement various directory structures to organize files efficiently. Each structure offers different advantages in terms of simplicity, flexibility, and search efficiency.







Directory Implementation in Operating System

Directory implementation defines how directory entries (file names and their metadata) are stored and managed in an operating system.

Method 1

Directory Implementation using Singly Linked List

Method 2

Directory Implementation using Hash Table

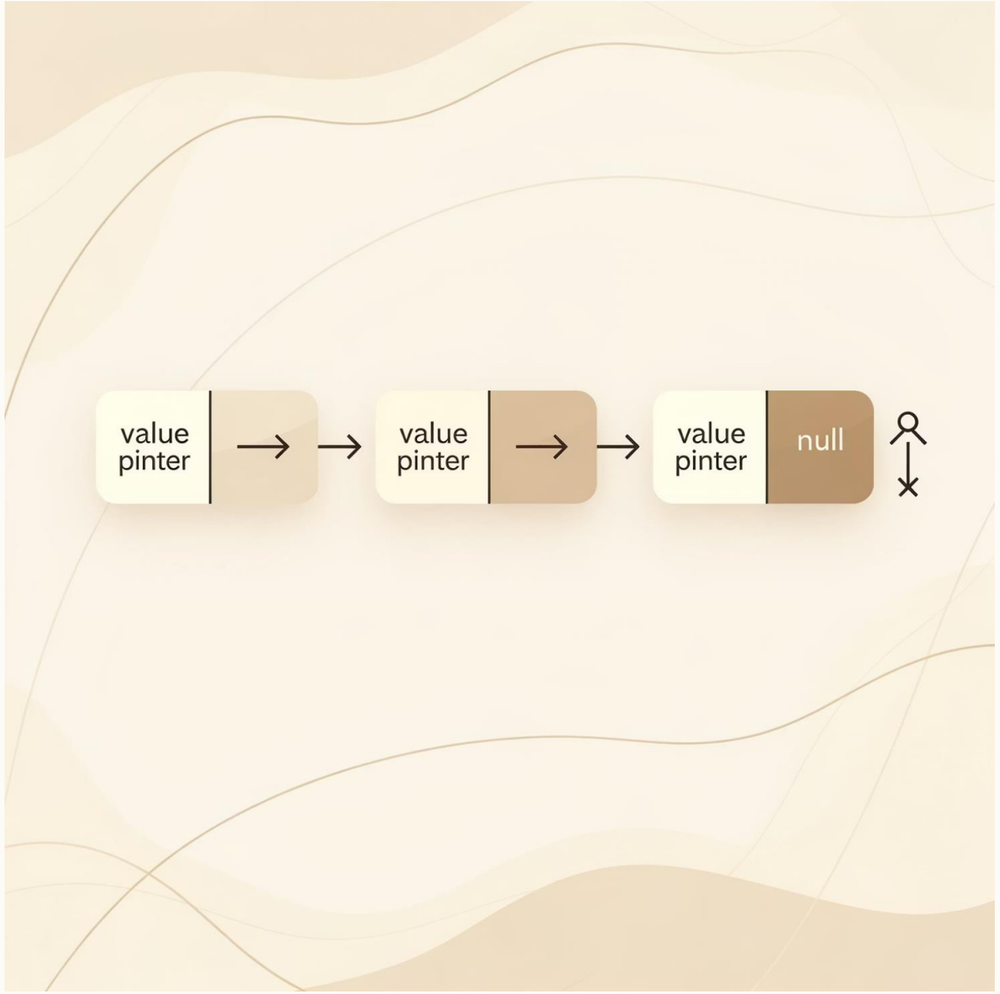
Directory Implementation using Singly Linked List

The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. We implement a directory by using a linear list of filenames with pointers to the data blocks.

Directory Entry Components

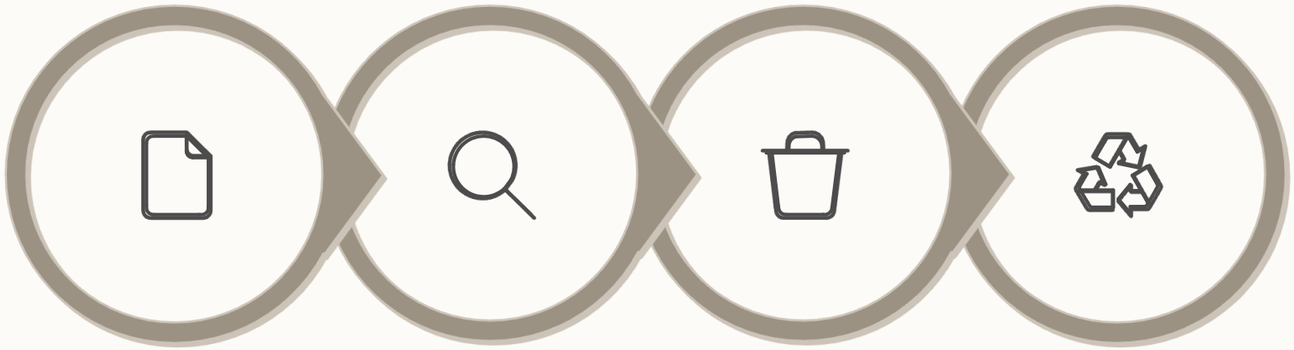
Each directory entry contains:

- File name for identification
- Pointer to file's metadata or data blocks

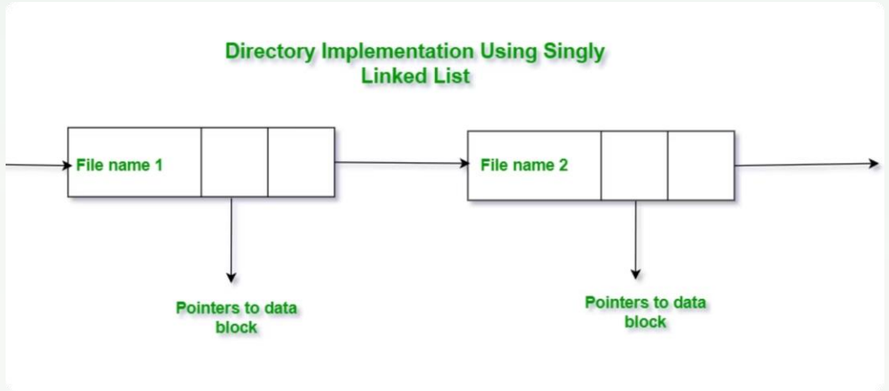


Working Principle

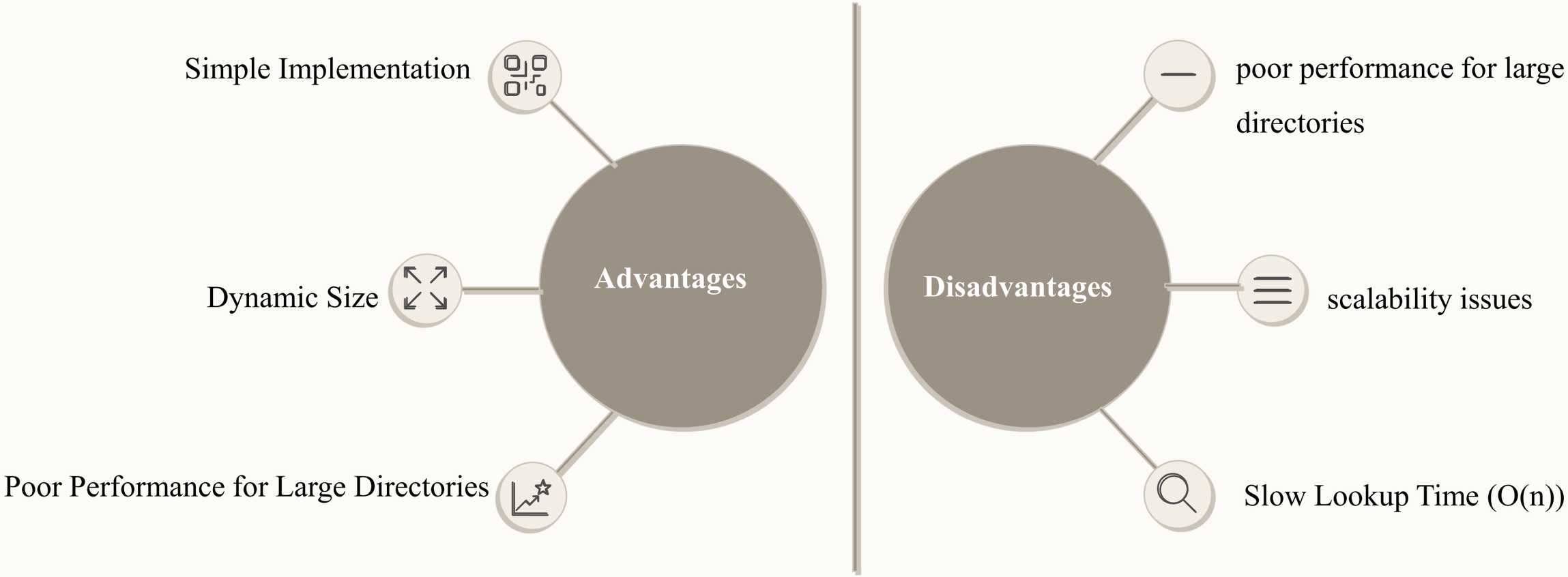
All directory entries are stored sequentially and connected using pointers, forming a chain-like structure that can be traversed from beginning to end.



File Creation **File Lookup** **File Deletion** **Reusing Entries**



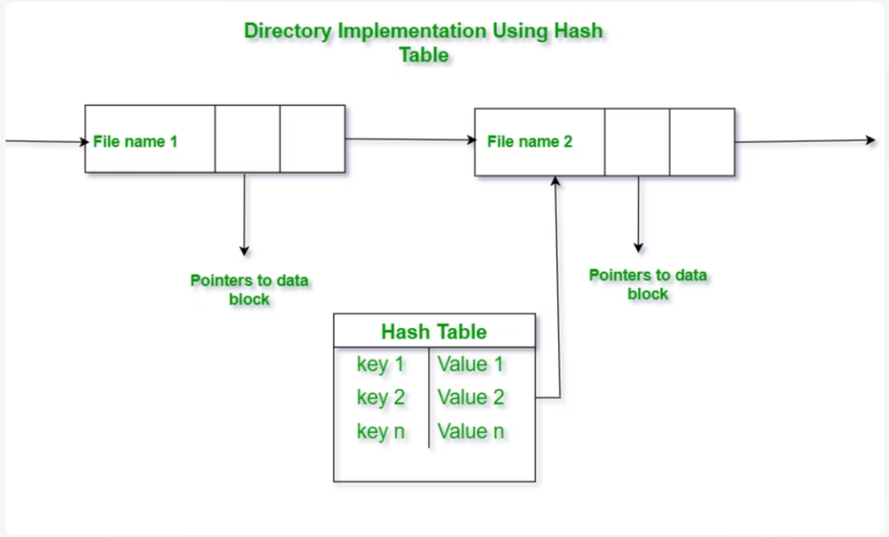
Linked List: Pros and Cons



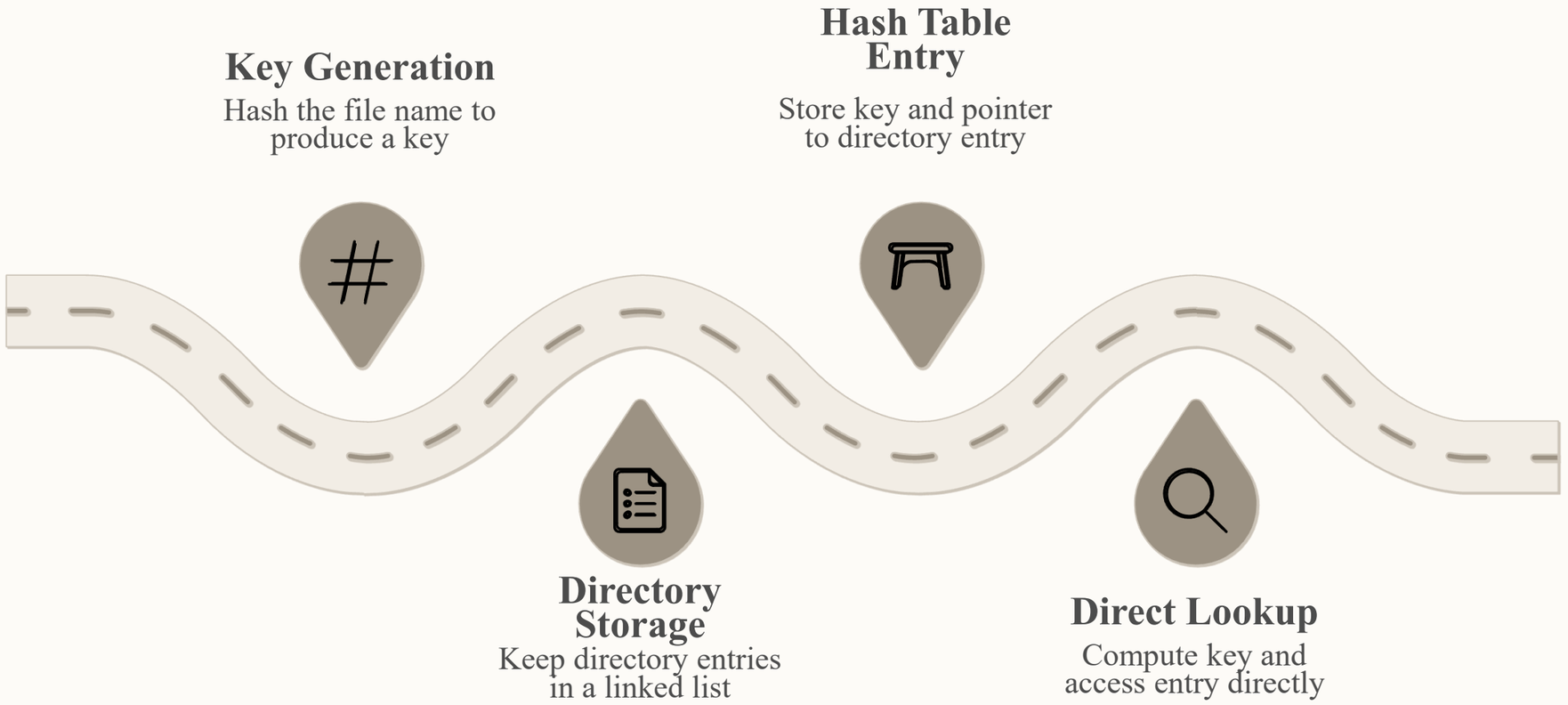
Directory Implementation using Hash Table

Working Principle

1. A hash function is applied to file names to generate keys
2. The hash table stores key–pointer pairs
3. The pointer refers to the directory entry stored in a linked list

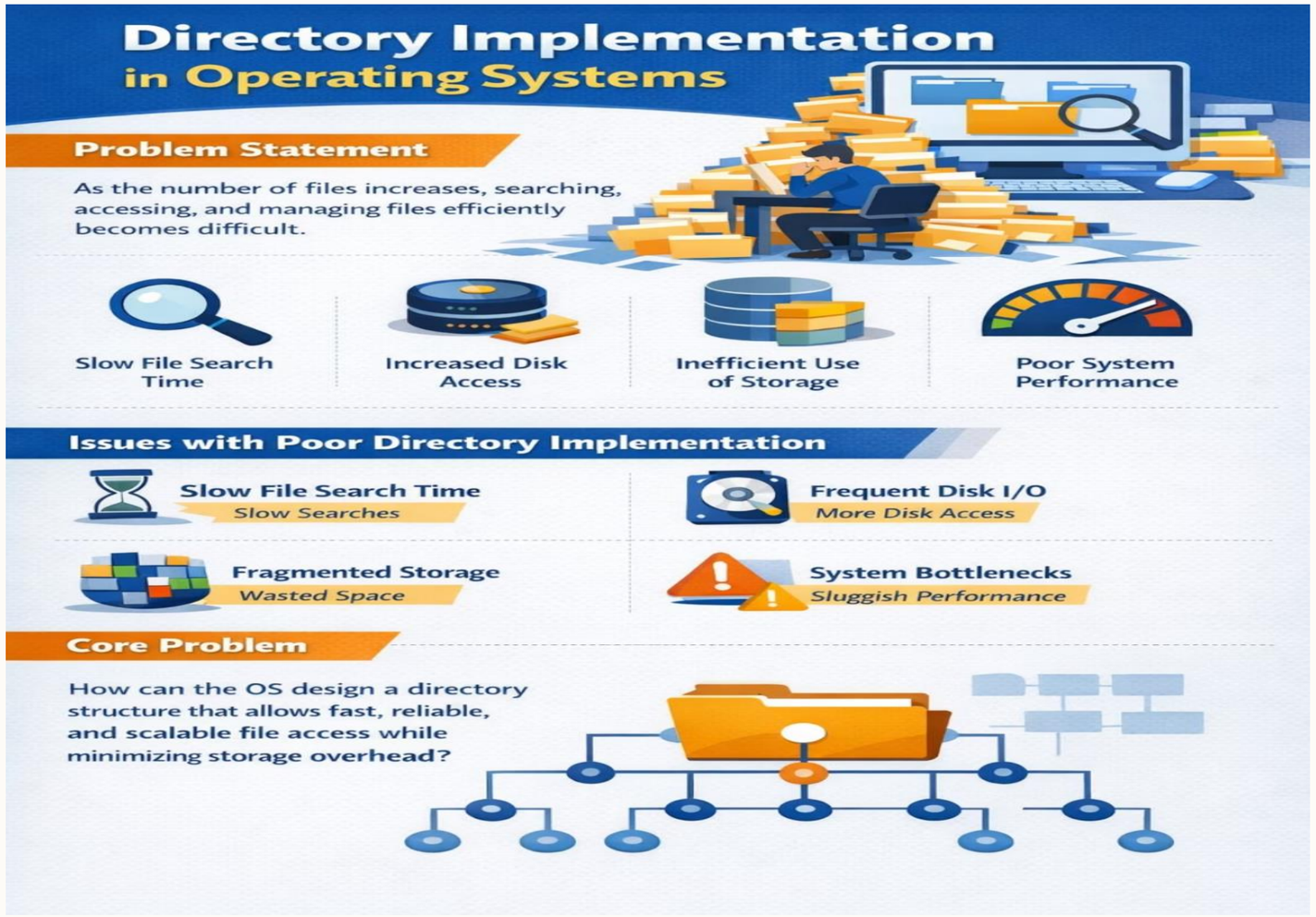


Hash Table Implementation Steps



Design Thinking Notes

1. Problem Statement



2. Empathize (Understanding the Nee

Users

The individuals interacting with the system.

Performance

The speed and efficiency of operations.

System Analysis

Operating System

The software that manages computer hardware.

Search Cost

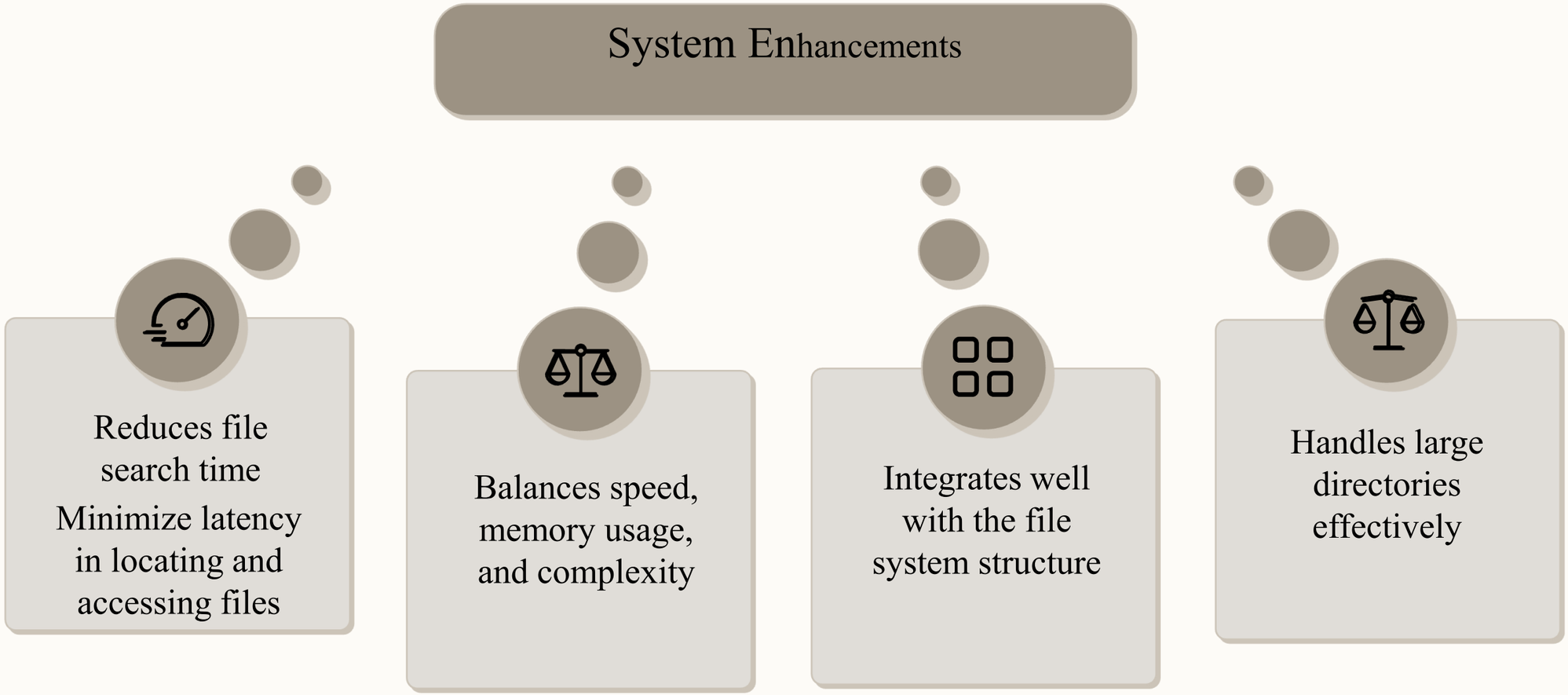
The resources required to find information.

Storage Devices

Hardware for data storage and retrieval.

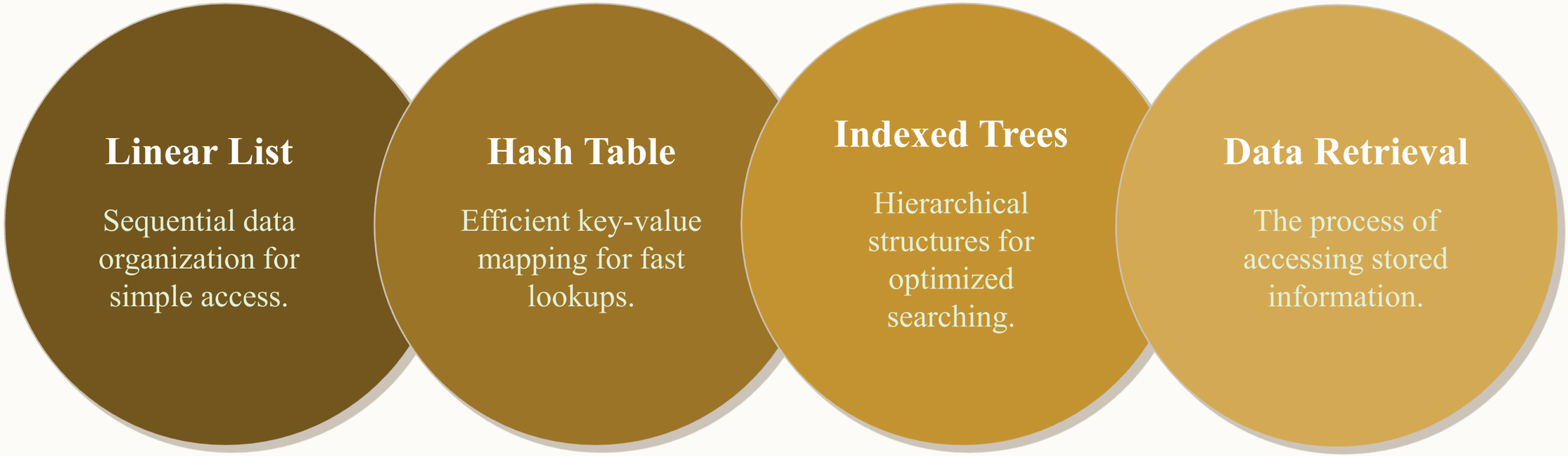
3. Define (Clear Problem Definition)

Design an efficient directory implementation technique that meets the following critical requirements:



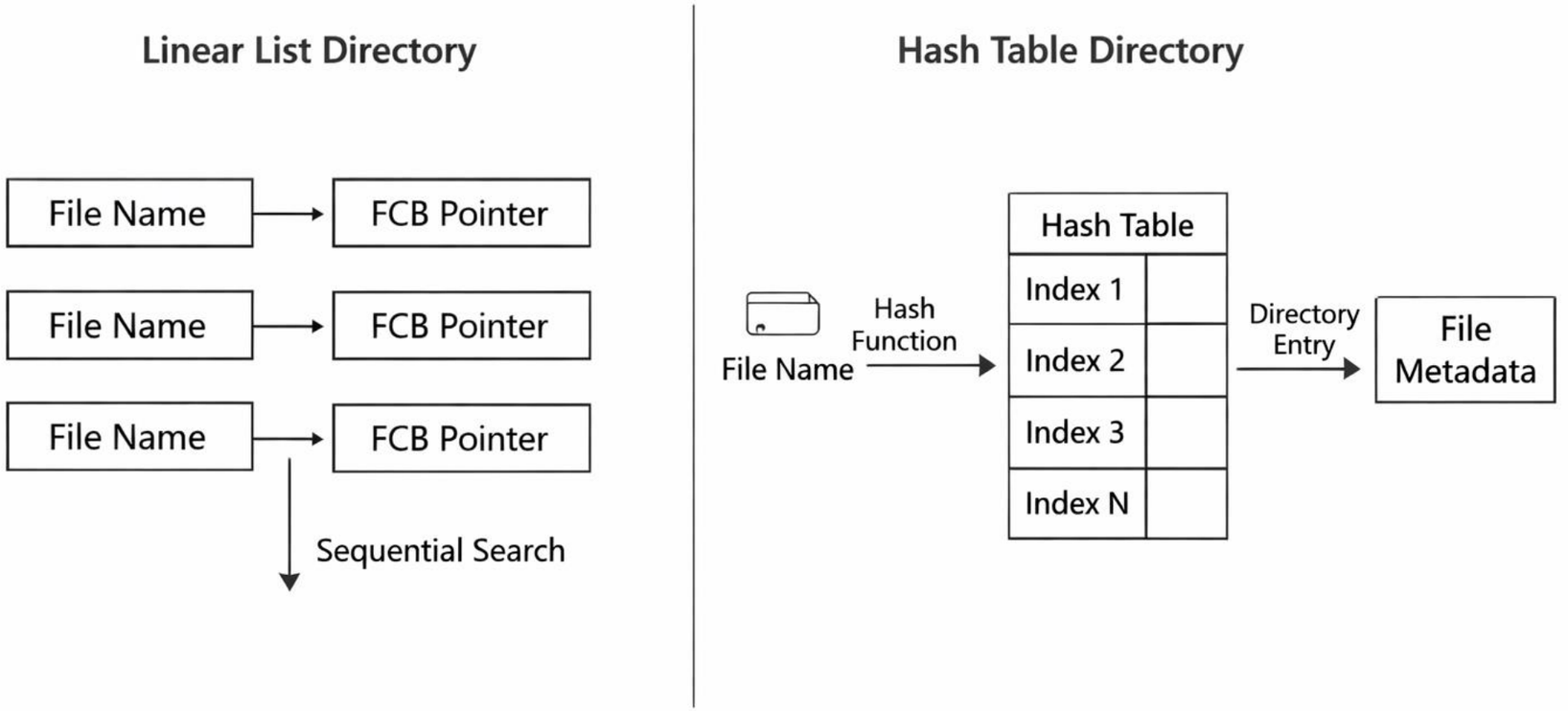
4. Ideate (Possible Solutions)

Different approaches to implement directories were considered, each offering unique advantages and tradeoffs:



Each solution represents a different balance point in the design space, trading off simplicity, speed, memory usage, and scalability. The choice depends on specific system requirements and usage patterns.

5. Prototype


Linear List Directory
File Name
FCB Pointer
File Name
FCB Pointer
File Name
FCB Pointer
Sequential Search
Hash Table Directory
File Name
Hash Function
Hash Table
Index 1
Index 2
Index 3
Index N
Directory Entry
File Metadata

Evaluation of Solutions

Criteria	Linear List	Hash Table
Search Speed	Slow (O(n))	Fast (O(1) avg)
Implementation	Simple	Complex
Memory Usage	Low	Higher
Scalability	Poor	Good

Result

- Linear List is suitable for small directories
- Hash Table is better for modern OS with large file systems

Final Solution

Modern operating systems adopt hash-based or tree-based directory implementations to ensure:

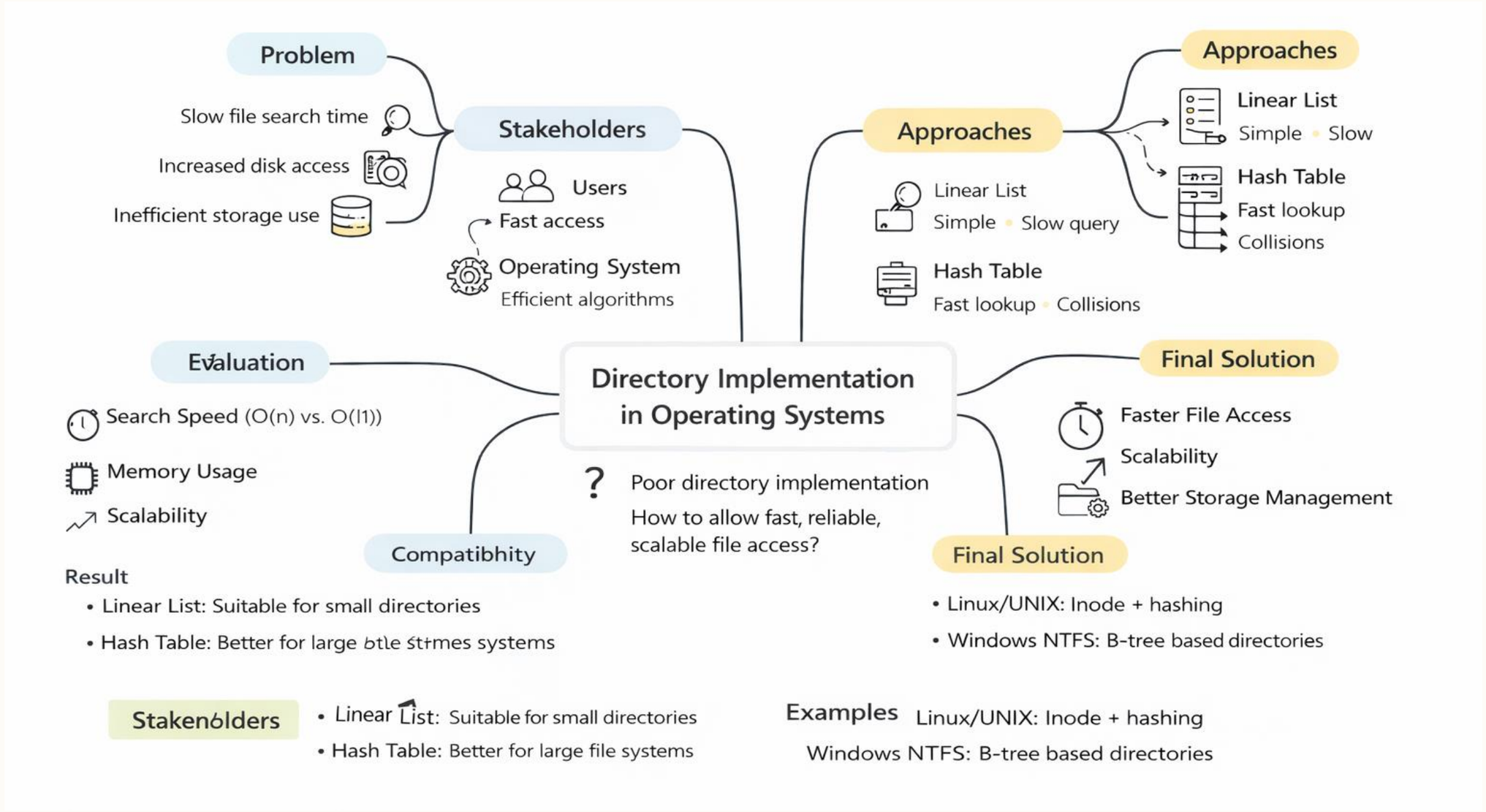
Faster File Access

Scalability

Better Storage Management

Examples

- Linux/UNIX: Inode + hashing
- Windows NTFS: B-tree based directories


Problem
Slow file search time
Increased disk access
Inefficient storage use
Stakeholders
Users
Fast access
Operating System
Efficient algorithms
Approaches
Linear List
Simple • Slow query
Hash Table
Fast lookup • Collisions
Approaches
Linear List
Simple • Slow
Hash Table
Fast lookup
Collisions
Evaluation
Search Speed (O(n) vs. O(1))
Memory Usage
Scalability
Result
• Linear List: Suitable for small directories
• Hash Table: Better for large file systems
Directory Implementation in Operating Systems
? Poor directory implementation
How to allow fast, reliable, scalable file access?
Final Solution
Faster File Access
Scalability
Better Storage Management
Compatibility
Final Solution
• Linux/UNIX: Inode + hashing
• Windows NTFS: B-tree based directories
Stakeholders
• Linear List: Suitable for small directories
• Hash Table: Better for large file systems
Examples
Linux/UNIX: Inode + hashing
Windows NTFS: B-tree based directories