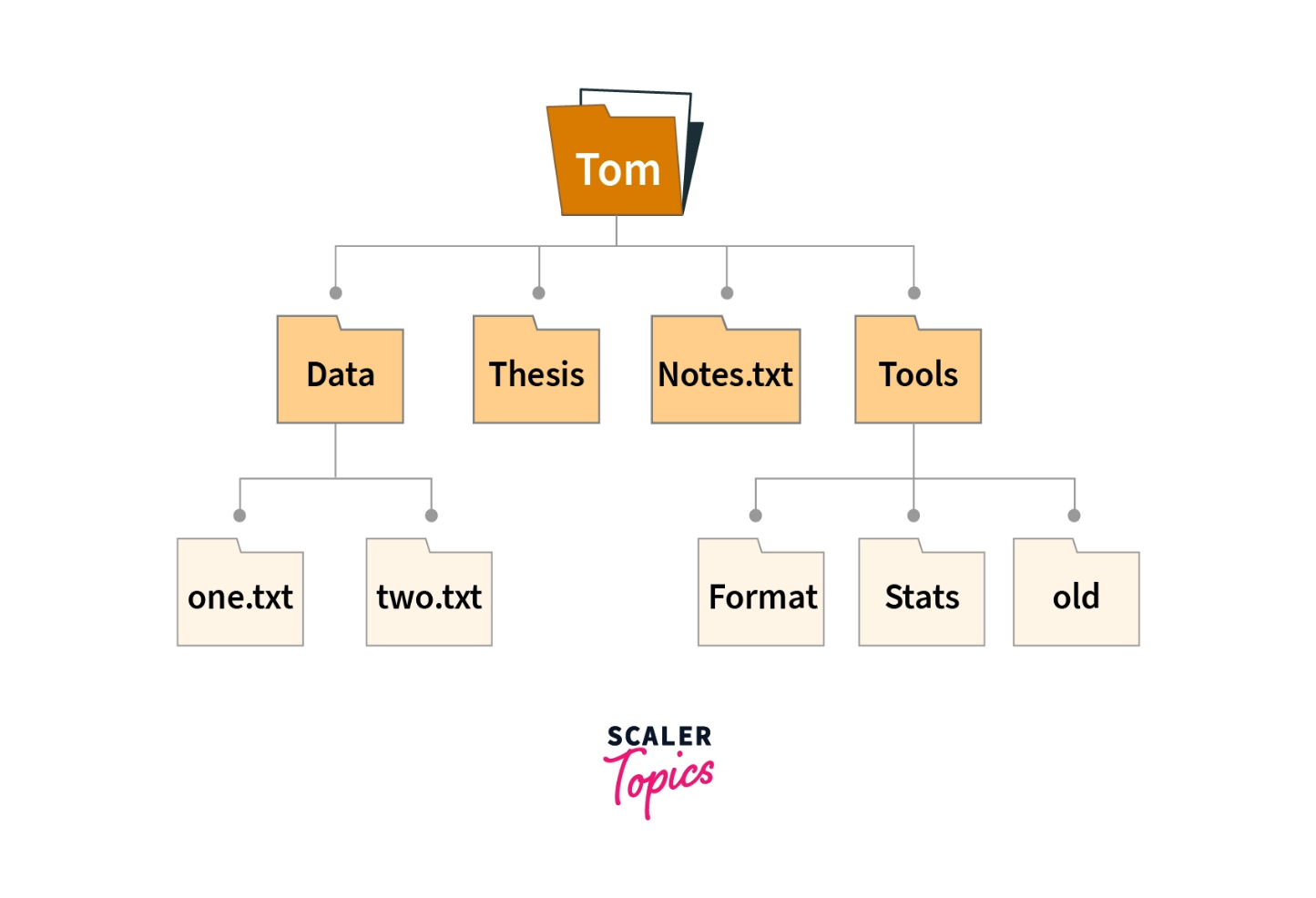
**Introduction to File Systems**

**1. What is a File System?**

- Definition: A file system is a method used by operating systems to control how data is stored and retrieved. It organizes data into files and directories and manages their storage on a storage medium like a hard drive or SSD.



- Role in Operating Systems:

- Provides a structure to store and retrieve files.

- Manages space allocation and file metadata (e.g., size, creation date).

- Handles access control, permissions, and security.

**Advantages of using a file system in OS**

1. **Organized Data Storage**: A file system provides a structured way to store and organize data on storage devices, making it easy to locate and access files.
2. **Efficient Data Retrieval**: It enables quick and efficient retrieval of files by providing a hierarchical directory structure and indexing mechanisms.
3. **Data Security and Permissions**: File systems offer security features like access control lists and file permissions, ensuring that only authorized users can access or modify specific files.
4. **Data Integrity and Reliability**: They include features like journaling and checksums to maintain data integrity and protect against data corruption or loss due to unexpected events.
5. **Space Management**: File systems handle allocation and deallocation of storage space, preventing issues like fragmentation and ensuring optimal use of available space.

**Disadvantages of using a file system in OS**

1. **Limited File Naming Conventions**: Some file systems impose restrictions on file names, such as maximum length or prohibited characters, which can be limiting for users.
2. **Potential for Fragmentation**: Over time, file systems can become fragmented, meaning that files are stored in non-contiguous blocks on the storage device, potentially leading to reduced performance.
3. **Security Vulnerabilities**: While file systems provide security features, they can still be susceptible to vulnerabilities and attacks if not properly configured or updated.
4. **Overhead for Small Files**: File systems may allocate a minimum block size for each file, which can result in wasted space for small files, leading to less efficient use of storage.
5. **Complex Maintenance**: Managing a file system, especially on large storage systems, can be complex and may require regular maintenance tasks such as defragmentation, disk cleanup, and periodic backups.

**2. Types of File Systems**

- Disk-Based File Systems:

- Examples: FAT32, NTFS (Windows), HFS+ (Mac), ext3/ext4 (Linux).

- Characteristics and usage scenarios for each.

* **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
* **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
* **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
* **HFS (Hierarchical File System):** A file system used by macOS.
* **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

- Network File Systems:

- Examples: NFS (Network File System), SMB/CIFS (Server Message Block).

- How they allow files to be shared over a network.

- Distributed File Systems:

- Examples: Hadoop Distributed File System (HDFS), Google File System (GFS).

- Designed for large-scale data storage and parallel processing.

3. Key Concepts in File Systems

- Files and Directories: How files are organized into hierarchical structures (directories or folders).

- File Metadata: Information stored about files, including size, timestamps, permissions, etc.

- Partitions and Volumes: Logical divisions of a disk managed by a file system.

- Mounting: How file systems are attached to the operating system so that they can be used.

**Files Types And Their Functions**

| **File type** | **Usual extension** | **Function** |
| --- | --- | --- |
| Executable | exe, com, bin | Read to run machine language program |
| Object | obj, o | Compiled, machine language not linked |
| Source Code | C, java, pas, asm, a | Source code in various languages |
| Batch | bat, sh | Commands to the command interpreter |
| Text | txt, doc | Textual data, documents |
| Word Processor | wp, tex, rrf, doc | Various word processor formats |
| Archive | arc, zip, tar | Related files grouped into one compressed file |
| Multimedia | mpeg, mov, rm | For containing audio/video information |
| Markup | xml, html, tex | It is the textual data and documents |
| Library | lib, a ,so, dll | It contains libraries of routines for programmers |
| Print or View | gif, pdf, jpg | It is a format for printing or viewing an ASCII or binary file. |

**File System Implementation**

**I) How File Systems Work**

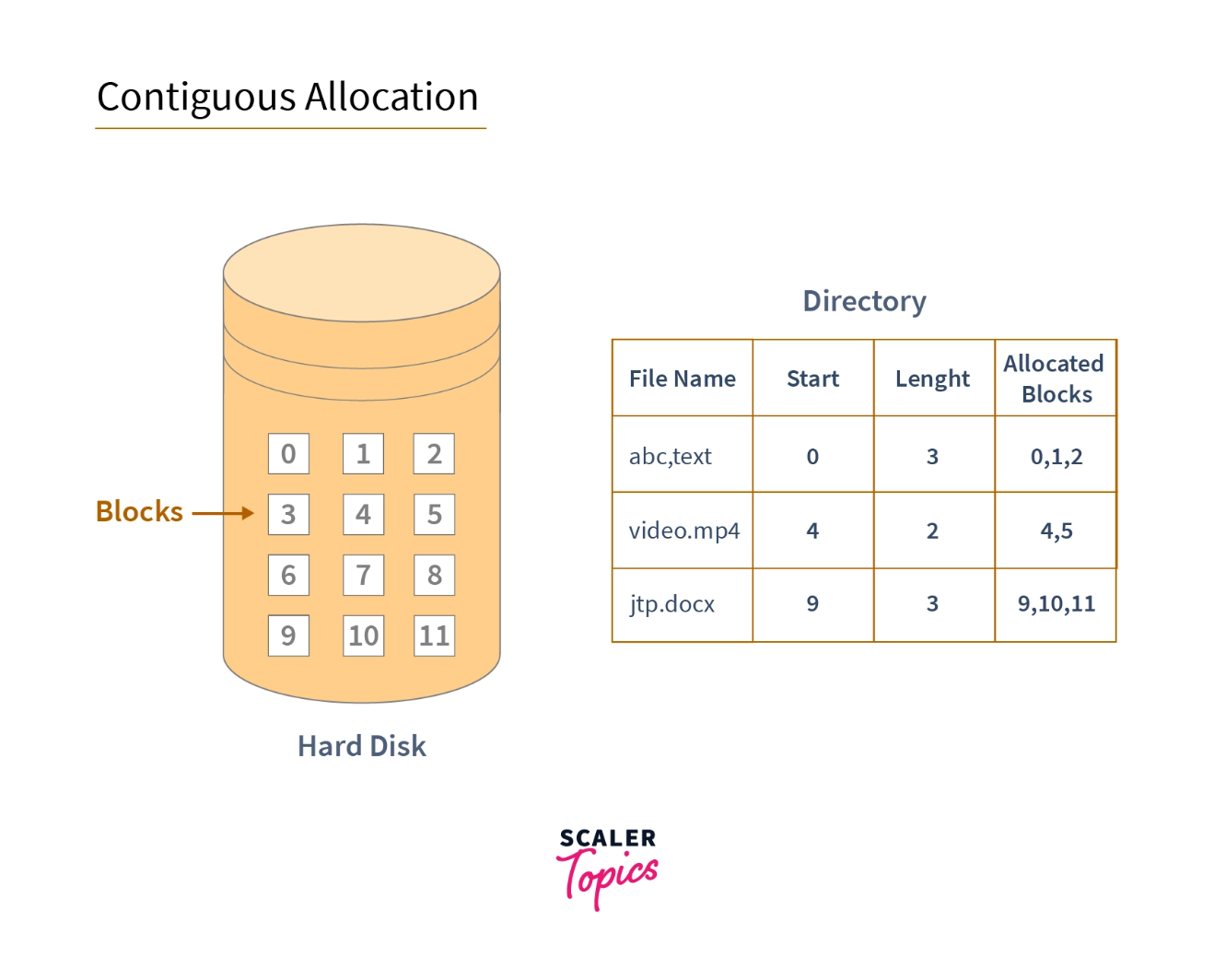
- Data Blocks and Inodes:

- Data Blocks: Units of data storage on the disk.

- Inodes: Metadata structure in Unix-like systems containing file information (but not the actual data).

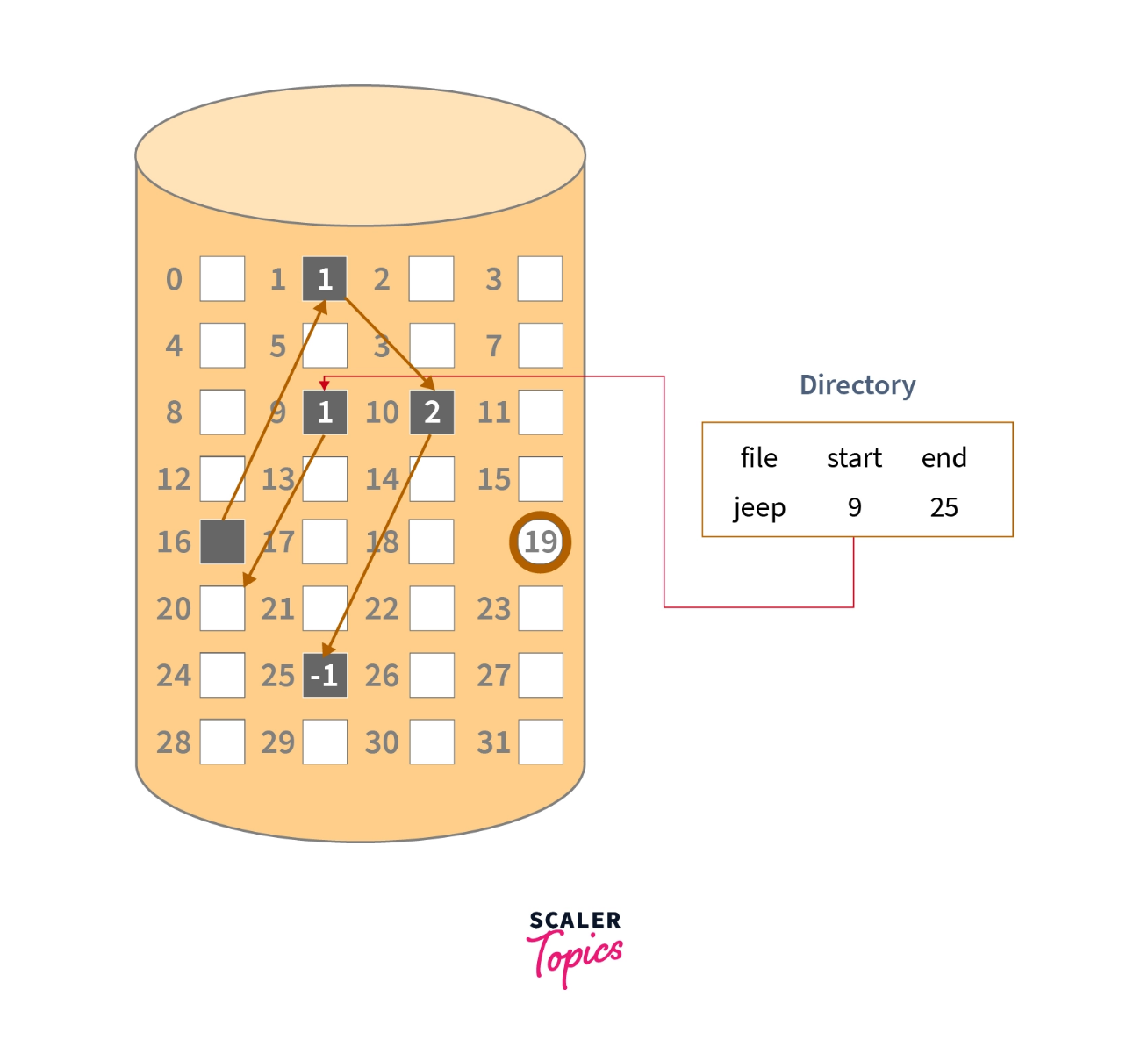
- Example: ext4 file system using inodes and blocks.

**File Allocation Methods:**

1. **Contiguous Allocation:** Stores files in contiguous blocks. Fast access but can lead to fragmentation. 

* **Advantages:**
  1. **Fast Access**: Files are stored in contiguous blocks, leading to faster sequential access.
  2. **Simple Implementation**: Easy to calculate the location of a file by its starting block and size.
  3. **Minimal Overhead**: Since no additional pointers are needed, memory overhead is minimal.
* **Disadvantages:**
  1. **External Fragmentation**: As files grow or shrink, fragmentation can occur, leading to inefficient use of space.
  2. **Difficult File Growth**: If a file grows beyond its allocated space, relocating the file becomes necessary, which is inefficient.
  3. **Limited Flexibility**: Hard to allocate contiguous space for files as storage fills up.

1. **Linked Allocation:** Each file is a linked list of blocks scattered across the disk. More flexible but slower.

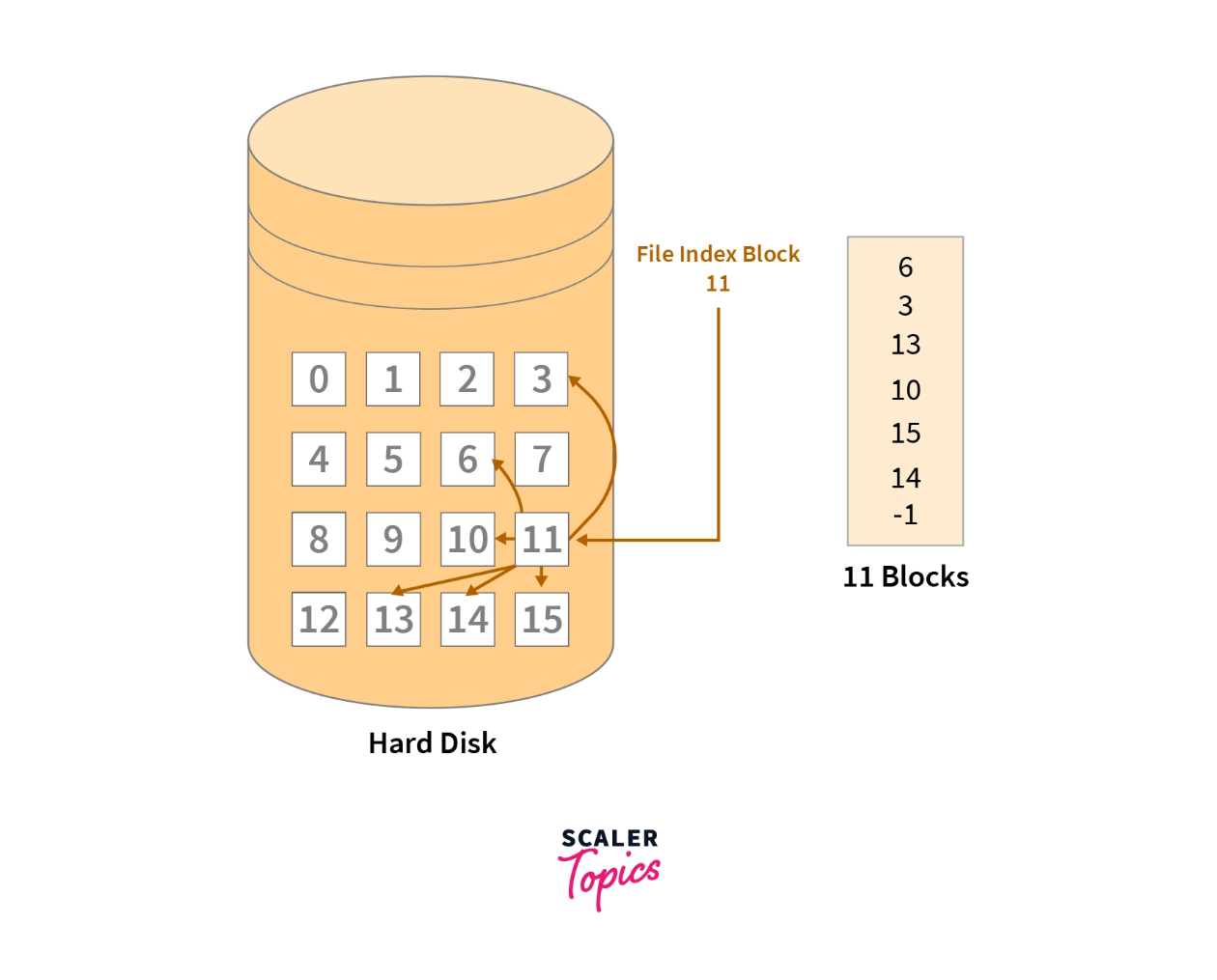


**Advantages:**

1. **No External Fragmentation**: Files can be stored in non-contiguous blocks, reducing fragmentation.
2. **Dynamic File Size**: File size can grow easily by adding new blocks, as blocks are linked.
3. **Efficient Space Utilization**: Any free block can be used, making better use of available disk space.

**Disadvantages:**

1. **Slow Access**: Access is slower due to the need to follow pointers between blocks.
2. **Higher Overhead**: Additional memory is required for the pointers, increasing overhead.
3. **Not Suitable for Random Access**: Random access is inefficient, as each block must be traversed from the start.
4. **Indexed Allocation:** Uses an index block to point to all file blocks. Used in Unix-based systems (e.g., ext4).



**Advantages:**

1. **Direct Access**: All block addresses are stored in an index, allowing direct access to any block.
2. **No Fragmentation Issues**: Since files don’t need to be stored in contiguous blocks, fragmentation is not a concern.
3. **Efficient for Large Files**: Suitable for large files, as all block pointers are stored in one place.

**Disadvantages:**

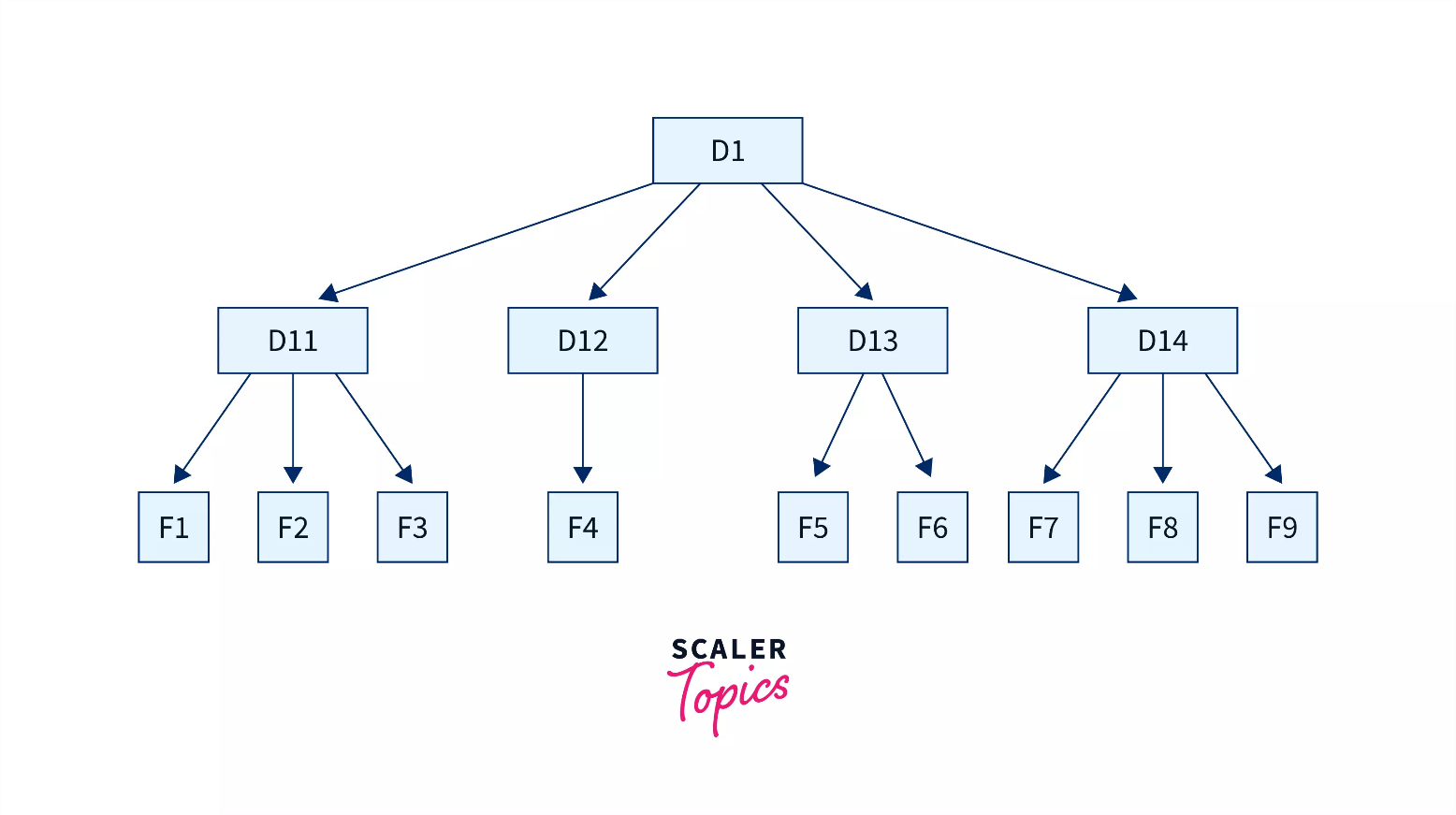
1. **Overhead for Small Files**: Index blocks add overhead, especially for small files.
2. **Index Block Limitations**: If the index block is full, additional methods (e.g., chaining) are needed to handle more blocks.
3. **Complexity**: Requires more complex management, particularly when handling very large files that exceed a single index block.

File Access Methods:

- Sequential Access: Files are accessed in a specific order (e.g., log files).

- Random Access: Files can be accessed in any order (e.g., databases).

**2. File Directory Structures**

****The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Below are information contained in a device directory.

* Name
* Type
* Address
* Current length
* Maximum length
* Date last accessed
* Date last updated
* Owner id
* Protection information

**The operation performed on the directory are:**

* Search for a file
* Create a file
* Delete a file
* List a directory
* Rename a file
* Traverse the file system

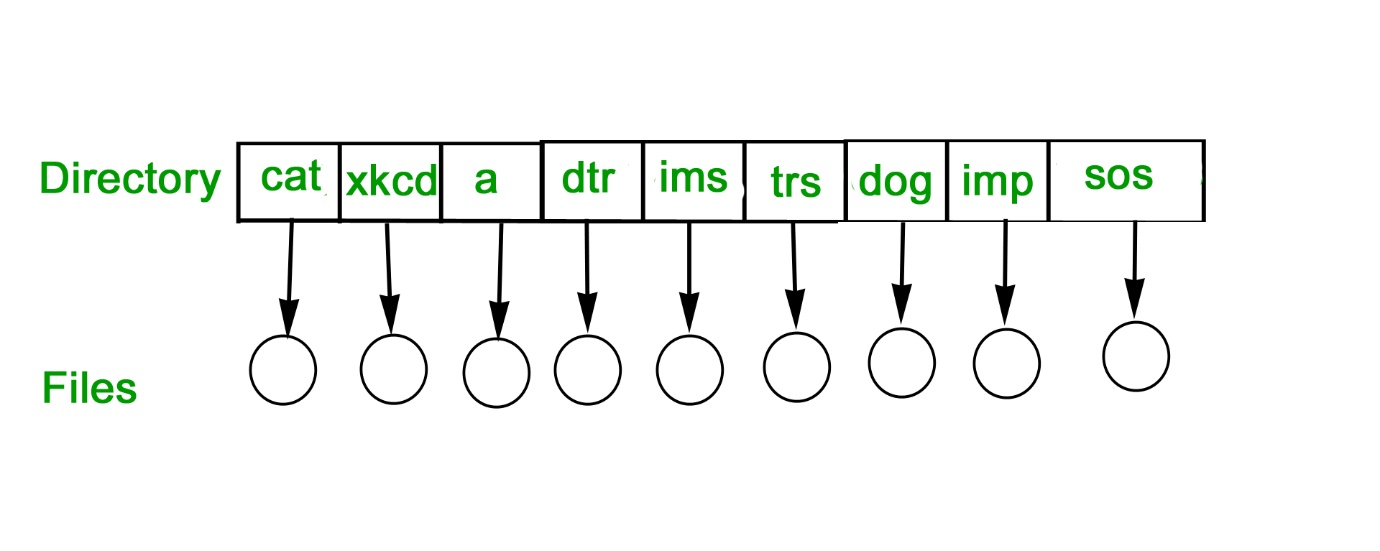
**Advantages of Maintaining Directories**

* **Efficiency:** A file can be located more quickly.
* **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
* **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

**Single-Level Directory**

All files stored in a single directory. Simple but impractical for large systems. In this, a single directory is maintained for all the users.

* **Naming Problem:** Users cannot have the same name for two files.
* **Grouping Problem:** Users cannot group files according to their needs.

****

**\*\*Advantages:\*\***

**1. Simple and easy implementation compared to other directory structures.**

**2. Easier file searching when file sizes are small.**

**3. Supports operations like searching, creation, deletion, and updating.**

**\*\*Disadvantages:\*\***

**1. Name collisions occur as multiple users may create files with the same name.**

**2. File replacement occurs when a new file with the same name is created.**

**3. File searching becomes time-consuming when files are large.**

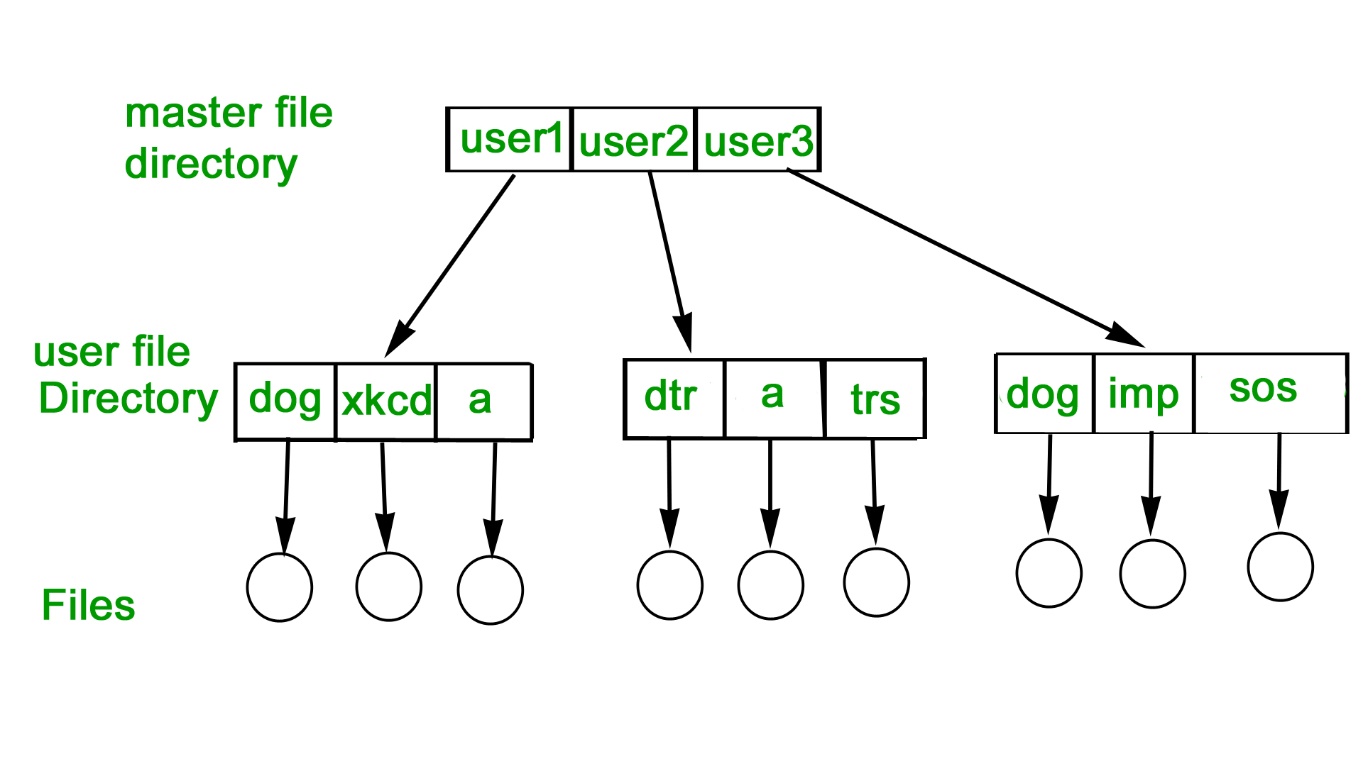
**4. No support for grouping similar file types together.**

**`**

**Two-Level Directory**

Separate directories for each user. Provides isolation but not flexible for shared files. In this separate directories for each user is maintained.

* **Path Name**: Due to two levels there is a path name for every file to locate that file.
* Now, we can have the same file name for different users.
* Searching is efficient in this method.

****

**### Advantages:**

**1. Different users can have the same directory and file names without conflict.**

**2. Searching for files becomes simpler.**

**3. User-defined directories provide privacy, preventing unauthorized access.**

**4. Users cannot access other users' directories without permission.**

**### Disadvantages:**

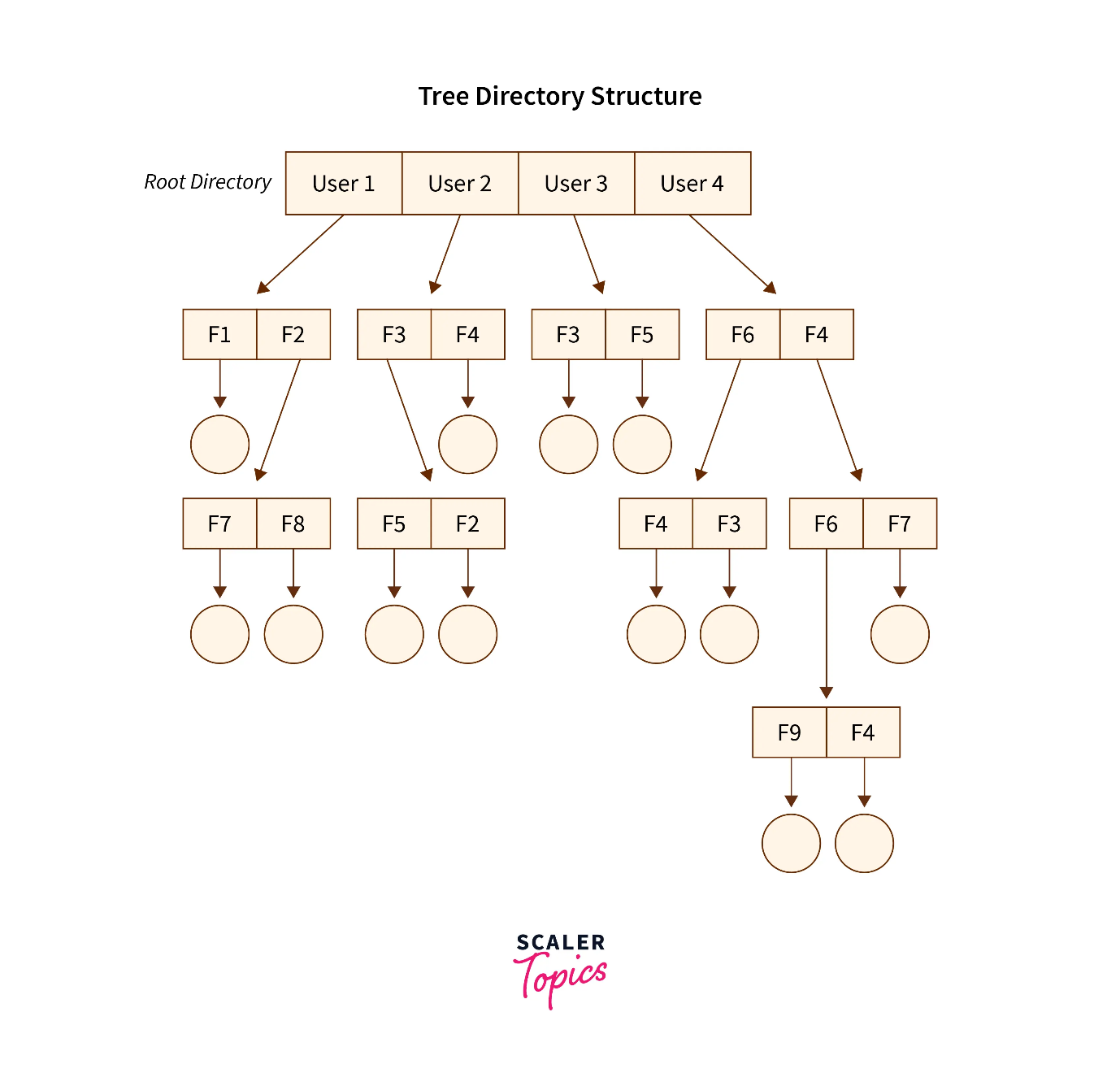
**1. Users cannot share files with other users.**

**2. No scalability, as files of the same type cannot be grouped together.**

**3. Users cannot create subdirectories, only one user file directory is allowed per master directory.**

**Tree-Structured Directory**

Hierarchical organization of directories and files, common in modern operating systems. The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.

****

**\*\*Advantages:\*\***

**1. Searching is effective using absolute or relative paths.**

**2. Similar files can be grouped into one directory.**

**3. Reduced chances of name/type collisions.**

**4. Scalable directory structure.**

**\*\*Disadvantages:\*\***

1. Files cannot be shared between users.

2. Users cannot modify/update other users' root directories.

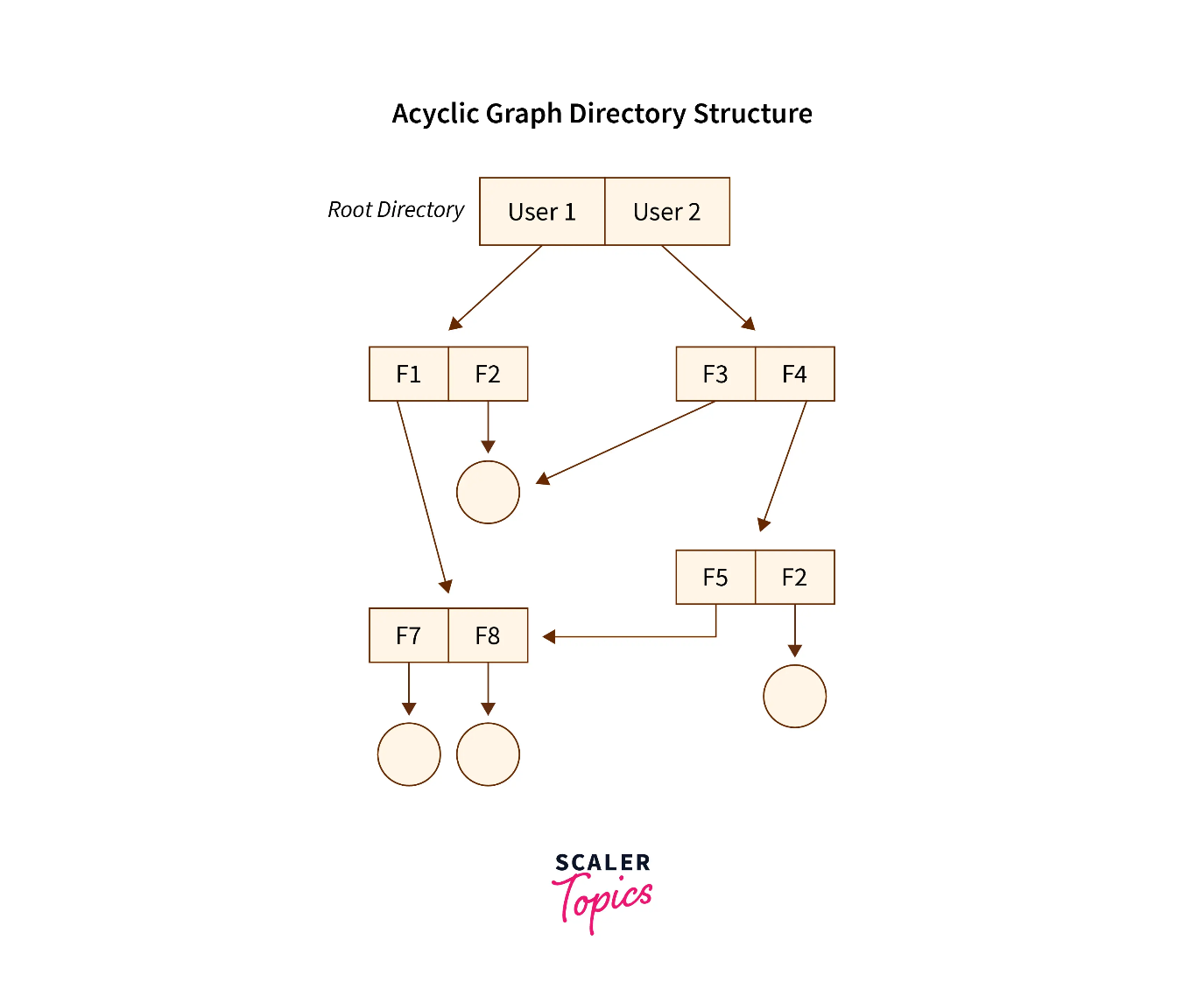
3. Accessing files requires navigating through multiple directories, making it inefficient.

4. Files that do not fit the hierarchical model need to be saved in multiple directories.

**Acyclic Graph Directory**

Allows directories and files to be shared between users, avoiding duplication.

- Example: Use Windows Explorer or Linux `tree` command to visualize a tree-structured directory.



\*\*Advantages:\*\*

1. Files can be shared between users.

2. Files are easier to search compared to the tree-structured directory, as multiple paths can lead to the same file.

\*\*Disadvantages:\*\*

1. Deleting a file may cause issues due to file sharing via linking.

a. Soft links result in dangling pointers if the file is deleted.

b. Hard links require all references to be removed, complicating future file references.