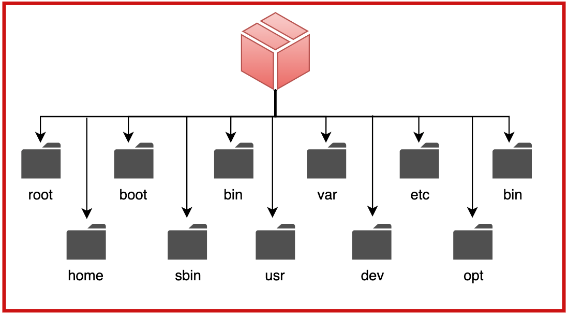
1. File system, and what are its components

A file system is a method and data structure that an operating system uses to control how data is stored and retrieved on a storage device.It organises data into files and directories, maintains **metadata**, and manages the allocation of space on the storage medium.

A file system is a crucial component of any operating system, serving as a framework for organising and managing the storage and retrieval of data on storage devices such as hard drives, SSDs, and USB drives. It provides a way to store data in an organised manner using files and directories, making it easy to locate and manipulate.

The file system abstracts the physical aspects of the storage medium, allowing users and applications to interact with data using logical structures. This abstraction simplifies data management and access, ensuring that data can be stored efficiently and retrieved quickly when needed. Additionally, file systems manage the allocation of space on the storage device, optimising the use of available storage and preventing conflicts.

**Hierarchical Structure**



The hierarchical structure of a file system is akin to a tree, with a root directory at the top that branches out into subdirectories and files. This structure facilitates an organised and intuitive way to store and access data. Each directory can contain multiple files and subdirectories, creating a multi-level organisation that can represent complex data relationships.

For example, a user's home directory might contain subdirectories for documents, pictures, and music, each of which can have further subdivisions. This hierarchical approach not only helps in logical organisation but also enhances security by allowing permissions to be set at different levels. The structure ensures that users can navigate and manage files effectively, maintaining an organised system that is easy to understand and use.

**Metadata**

**Metadata** in a file system provides essential information about each file and directory, enabling the system and users to understand its properties and status. Common metadata elements include:

* File name
* Size
* Type
* Creation and modification timestamps
* Ownership
* Access permissions

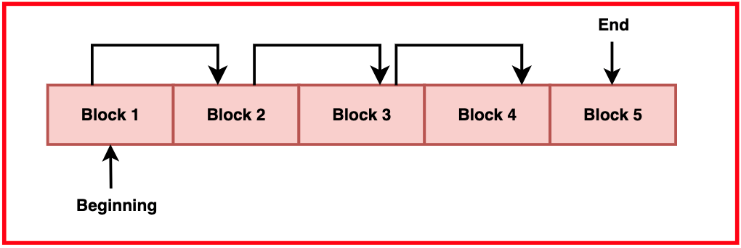
This information is critical for various file system operations.

For example, timestamps help in identifying the latest versions of files, while permissions ensure that only authorised users can access or modify certain files. Metadata also plays a role in system performance and security, as it allows the file system to efficiently manage access and maintain integrity. Advanced file systems may include additional metadata such as extended attributes, which can store custom information relevant to specific applications.

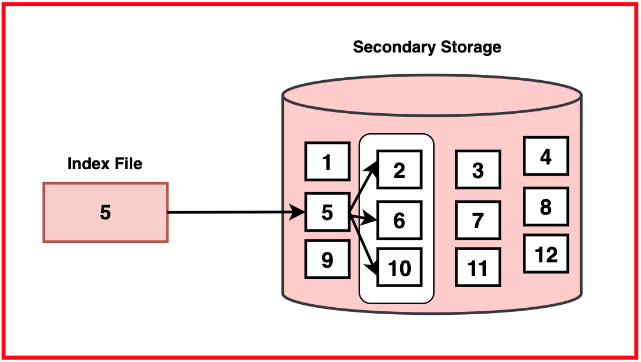
**Access Methods**

Access methods define how data within files can be accessed and manipulated. Different methods are suitable for different use cases:

* **Sequential Access:** This method reads or writes data in a linear sequence, suitable for tasks where data is processed in order, such as reading a log file.



* **Direct Access:** Allows data to be read or written at any location within the file without sequentially reading preceding data. This is ideal for databases and applications requiring random data access.
* **Indexed Access:** Utilises an index to quickly locate the data blocks within a file, improving access speed for large files with complex structures.



* **Random Access:** Similar to direct access but often used for smaller data chunks within a file, enabling quick retrieval or modification of specific sections of the file.

Each method has its own advantages and is chosen based on the specific needs of the application or system.

**File System Operations**

File system operations encompass a range of actions that can be performed on files and directories, enabling users and applications to manage data effectively. These operations include:

* Creating Files and Directories: Establishing new storage locations for data.
* Reading and Writing Data: Accessing and modifying the contents of files.
* Renaming and Moving: Changing the names or locations of files and directories.
* Deleting: Removing files or directories from the storage medium.
* Managing Permissions: Setting access controls to ensure only authorised users can perform certain actions.

These operations are fundamental to the functioning of a file system, allowing users to organise, protect, and interact with their data.

**File System Integrity**

Maintaining **file system integrity** is crucial to ensuring data consistency and reliability. Various techniques are employed to protect against data corruption, loss, and unauthorised access. These techniques include:

* **Journaling:** Keeping a log of changes to the file system, which helps in recovering from crashes or unexpected failures by replaying the logged changes.
* **Checksums:** Used to verify data integrity by comparing stored and computed values, ensuring that data has not been altered or corrupted.
* **File System Consistency Checks (FSCC):** Utilities that scan the file system for inconsistencies and repair any detected issues.
* **Redundancy Methods:** Such as mirroring or using parity bits to protect data against hardware failures by storing multiple copies or using error-correcting codes.

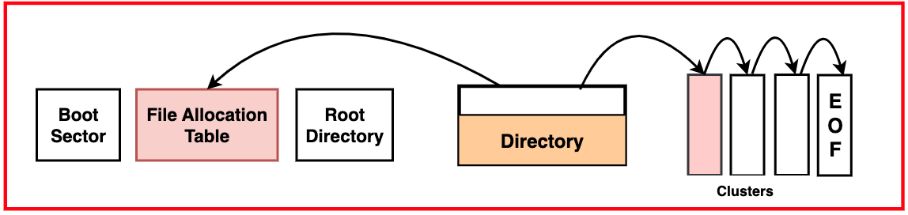
These techniques work together to ensure the file system remains reliable and resilient, safeguarding the data stored within it

1. Types of file system

**Disk-based File Systems**

Disk-based file systems are the traditional file systems used to manage data on physical storage devices like hard drives, SSDs, and optical discs. These file systems are responsible for organising data into files and directories, managing space on the disk, and ensuring data integrity and security.

**FAT (File Allocation Table)**





FAT is one of the oldest file systems, known for its simplicity and wide compatibility. It's commonly used in removable media like USB drives and memory cards. FAT has several variants, including FAT12, FAT16, and FAT32, each offering different limits on file and partition sizes. While FAT is simple and supported by virtually all operating systems, it lacks advanced features and scalability, making it unsuitable for modern large-scale storage needs.

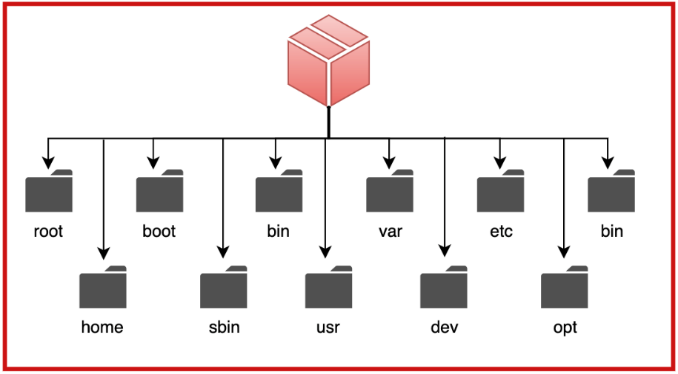
**NTFS (New Technology File System)**

NTFS is a robust and advanced file system used primarily in Windows environments. It offers features like file compression, encryption, access control lists (ACLs) for security, and journaling to maintain file system integrity in case of crashes. NTFS supports large volumes and file sizes, making it suitable for both personal computers and enterprise servers.

**ext4 (Fourth Extended File System)**

ext4 is a widely-used file system in Linux environments. It improves on its predecessors (ext2 and ext3) by offering larger volume and file size support, extents for better performance, and faster file system checks. ext4 is known for its reliability and efficiency, making it a popular choice for Linux-based systems.

**HFS+ (Hierarchical File System Plus)**



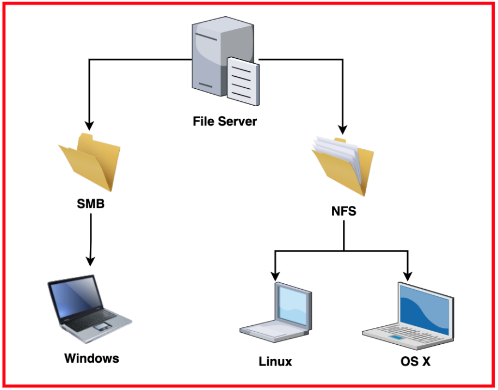
HFS+ is used in macOS systems. It supports features like journaling, large file sizes, and efficient storage management. HFS+ has been largely replaced by the more advanced APFS (Apple File System) in newer macOS versions, but it remains in use on older systems and some storage devices.

**Network File Systems**

Network file systems enable the sharing of files across a network, allowing multiple users and systems to access and manage data stored on a central server. These systems are essential for collaborative environments and distributed computing.

**NFS (Network File System)**

NFS is a distributed file system protocol used primarily in Unix and Linux environments. It allows a computer to access files over a network as if they were on its local disks. NFS supports features like file locking and access control, enabling efficient file sharing and collaboration.



**CIFS/SMB (Common Internet File System/Server Message Block)**

CIFS, also known as SMB, is used primarily in Windows networks for file and printer sharing. It provides robust features like file locking, authentication, and encrypted communications. SMB is widely used in both small office and enterprise environments.

**AFS (Andrew File System)**

AFS is a distributed file system that offers scalable file sharing across large networks. It provides features like strong security through Kerberos authentication, volume management, and efficient caching. AFS is used in academic and research institutions for large-scale distributed file sharing.

**DFS (Distributed File System)**

DFS is a set of client and server services in Windows Server that enables the creation of a single namespace for multiple file servers and shares. It simplifies data access and management in large networked environments by providing redundancy and load balancing.

**Distributed File Systems**

Distributed file systems manage data across multiple servers or storage devices, providing high availability, fault tolerance, and scalability. They are essential for handling large-scale data storage and processing in cloud and big data environments.

**HDFS (Hadoop Distributed File System)**

HDFS is designed for the storage and processing of large datasets in a distributed computing environment. It provides high throughput access to application data and is fault-tolerant, ensuring data replication and reliability across a cluster of machines. HDFS is a cornerstone of the Hadoop ecosystem used in big data analytics.

**GFS (Google File System)**

GFS, now succeeded by Colossus, is designed to support large-scale data processing at Google. It provides fault tolerance through data replication, efficient data streaming, and scalability to handle massive datasets across thousands of machines. GFS is optimized for large, sequential data reads and writes.

**Ceph**

Ceph is a distributed storage system that provides object, block, and file storage in a unified system. It is designed for high performance, scalability, and reliability, using intelligent data placement and replication to ensure fault tolerance and data integrity. Ceph is widely used in cloud storage solutions and enterprise environments.

**GlusterFS**

GlusterFS is a scalable network file system suitable for data-intensive tasks such as cloud storage and media streaming. It aggregates storage resources from multiple servers into a single namespace, providing high availability and performance. GlusterFS supports features like data replication, striping, and volume management.

**Amazon S3 (Simple Storage Service)**

S3 is an object storage service provided by Amazon Web Services (AWS). It offers scalable, high-durability storage for any type of data, accessible over the web via APIs. S3 is used for backup, archival, big data analytics, and content distribution.

**Specialized File Systems**

**ZFS (Zettabyte File System)**

ZFS is designed for data integrity, scalability, and advanced storage management. It includes features like data deduplication, compression, snapshots, and copy-on-write. ZFS ensures data integrity through end-to-end checksumming and self-healing capabilities, making it ideal for enterprise storage and data management solutions.

**Btrfs (B-tree File System)**

Btrfs is a modern Linux file system designed for fault tolerance, repair, and easy administration. It supports features like snapshots, subvolumes, compression, and integrated multi-device spanning. Btrfs is optimised for high performance and efficient storage management, making it suitable for servers and advanced storage systems.

**F2FS (Flash-Friendly File System)**

F2FS is optimised for NAND flash-based storage devices such as SSDs and eMMC. It aims to address the specific characteristics of flash memory, providing high performance and longevity. F2FS includes features like garbage collection, wear levelling, and efficient data allocation to optimise flash memory usage.

**ReFS (Resilient File System)**

ReFS is a Microsoft file system designed for data integrity, scalability, and resilience to data corruption. It supports features like integrity streams, large volume and file size support, and data scrubbing. ReFS is used in scenarios requiring high reliability and data protection, such as enterprise storage solutions.

**Virtual File Systems**

Virtual file systems provide an abstraction layer that allows different file systems to be accessed in a uniform way, presenting a consistent interface to users and applications regardless of the underlying storage medium.

**procfs (Process File System)**

procfs is a virtual file system in Unix-like systems that presents process information as files. It allows users and applications to interact with kernel data structures, such as process status and system information, through a standardized file interface. procfs simplifies system monitoring and management by providing a consistent and accessible view of system internals.

**tmpfs (Temporary File System)**

tmpfs is a virtual file system that stores files in volatile memory (RAM), providing fast access to temporary files that do not need to persist across reboots. tmpfs is used for tasks requiring high-speed storage, such as temporary data for running applications and caching. It combines the speed of RAM with the flexibility of a file system interface.

**FUSE (Filesystem in Userspace)**

FUSE allows users to create custom file systems without modifying the kernel. It provides a framework for developing file systems that run in user space, enabling the integration of diverse storage solutions and specialized file systems. FUSE is widely used for implementing file systems with unique features or specific application requirements.

**OverlayFS**

OverlayFS is a virtual file system that allows multiple file systems to be overlaid, presenting a single unified view. It is commonly used in container environments to provide writable layers on top of read-only base images. OverlayFS simplifies file system management in scenarios where changes need to be isolated and layered.

**UnionFS**

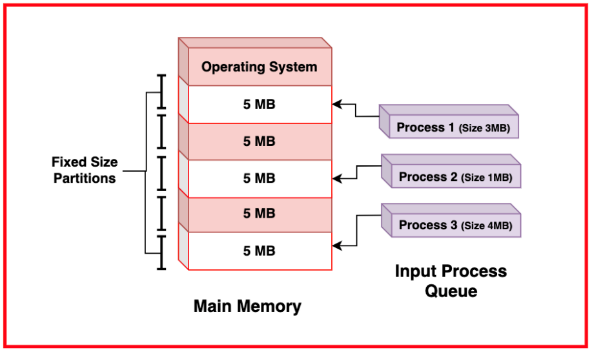
UnionFS is a virtual file system that merges multiple file systems into a single coherent file system. It allows files and directories from

1. File allocation and deallocation

**Contiguous Allocation**

Contiguous allocation is a file allocation method where each file is stored in a single contiguous block of disk space. This means that all the blocks occupied by a file are next to each other on the disk.

The primary advantage of contiguous allocation is its simplicity and efficiency in accessing files. Since the entire file is stored in a single continuous block, the system can quickly locate and read the file sequentially, leading to high performance for sequential file access. Additionally, calculating the location of any specific part of the file is straightforward.



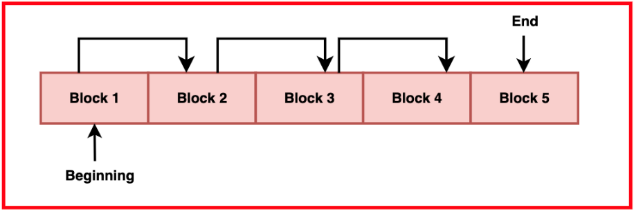
The major drawback of contiguous allocation is the problem of external fragmentation, where free space is scattered in small, non-contiguous blocks. This makes it difficult to find large contiguous spaces for new files, leading to insufficient disk space utilisation. Additionally, resizing files can be problematic, as there may not be enough contiguous space available to accommodate the new size.

Contiguous allocation is best suited for situations where files are relatively static in size and the storage system does not frequently add or remove files, such as read-only media like CD-ROMs.

**Linked Allocation**

Linked allocation addresses the fragmentation issue of contiguous allocation by allowing a file to be stored in non-contiguous blocks that are linked together.

Linked allocation eliminates external fragmentation, as each file block can be placed anywhere on the disk. It also simplifies file resizing, as additional blocks can be easily linked to the existing file chain.





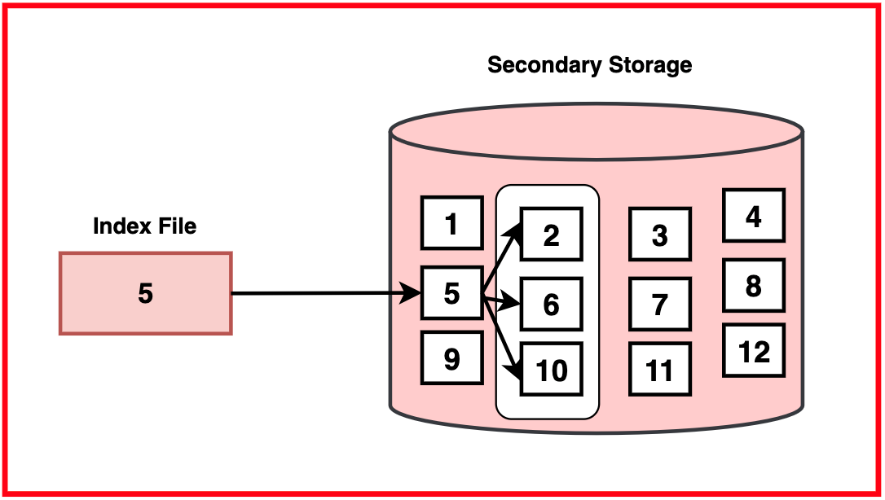
The major disadvantage is the added overhead of maintaining pointers in each block, which reduces the effective storage capacity. Furthermore, accessing a file sequentially can be slower because the system must read the pointer from each block to locate the next block. Random access is particularly inefficient, as it requires traversing the entire chain of pointers.

Linked allocation is suitable for systems where files are frequently created, deleted, or resized, such as certain database systems and file storage with many small files.

**Indexed Allocation**

Indexed allocation combines the benefits of contiguous and linked allocation by using an index block to store pointers to all the blocks of a file.

Indexed allocation supports efficient random access to file blocks, as the index provides direct pointers to each block. This method reduces external fragmentation and allows files to grow without needing contiguous blocks. Additionally, it simplifies the management of file blocks.



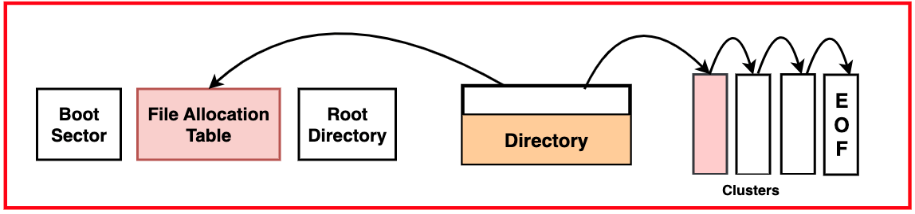
The main disadvantage is the overhead of storing the index block, which consumes additional disk space. For very large files, multiple index blocks might be needed, increasing complexity. Indexed allocation also requires a robust mechanism to handle the index blocks efficiently.

Indexed allocation is ideal for large files and systems requiring efficient random access, such as operating systems and applications handling large databases or multimedia files.

**File Allocation Tables (FAT)**

The File Allocation Table (FAT) is a data structure used by the FAT file system to manage disk space and keep track of the allocation status of each block.

FAT is simple and widely supported across different operating systems, making it highly compatible for use in removable media like USB drives and memory cards. The table structure allows easy navigation and management of file blocks.



FAT can suffer from fragmentation, both internal and external, leading to insufficient disk space usage. It also becomes less efficient as the disk size increases, because the table must be read frequently, which can slow down access times. Additionally, the simplicity of FAT means it lacks advanced features found in modern file systems, such as security and journaling.

FAT is commonly used for removable storage devices and systems requiring wide compatibility, such as cameras, MP3 players, and other consumer electronics.

**Allocation Methods Comparison**

| **Aspect** | **Contiguous Allocation** | **Linked Allocation** | **Indexed Allocation** | **FAT** |
| --- | --- | --- | --- | --- |
| Ease of Implementation | Simple | Moderate | Complex | Simple |
| Sequential Access Performance | High | Moderate | High | Moderate |
| Random Access Performance | Low | Low | High | Moderate |
| Fragmentation | High | Low | Low | Moderate to High |
| Space Utilisation | Low (due to Fragmentation) | Moderate | High | Moderate |
| Flexibility in File Resizing | Low | High | High | Moderate |

**File Deallocation Strategies**

File deallocation involves reclaiming space that was previously allocated to files that are no longer needed.

**Immediate Deallocation**

This strategy immediately releases the space occupied by a file back to the pool of free space as soon as the file is deleted. It ensures that space is available for new files without delay but can lead to fragmentation if not managed carefully.

**Delayed Deallocation**

In this approach, space is not immediately reclaimed upon file deletion. Instead, the system marks the file for deletion and reclaims the space during a scheduled maintenance period or when space is critically low. This can reduce fragmentation and improve performance but requires careful management to avoid running out of space.

**Garbage Collection**

Some file systems, particularly those used in SSDs and flash memory, employ garbage collection to manage deallocation. This involves periodically scanning the storage to identify and reclaim unused space, optimising storage usage and performance.

**Reference Counting**

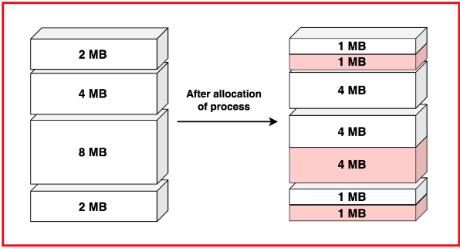
This method keeps track of how many references point to a file or block of data. Space is only deallocated when the reference count drops to zero, ensuring that shared data is not prematurely deleted. This is useful in systems with shared files or data blocks, such as versioned file systems.

**Deferred Deletion with Secure Wipe**

To enhance security, some systems use deferred deletion combined with secure wiping, where the space occupied by a deleted file is overwritten with random data before being released. This prevents recovery of sensitive data from deleted files but can add overhead to the deallocation process

1. Fragmentation

Fragmentation refers to the condition of a storage device where files are not stored in contiguous blocks, resulting in parts of files being scattered across different areas of the disk. This scattered arrangement can lead to inefficient use of storage space and reduced system performance.

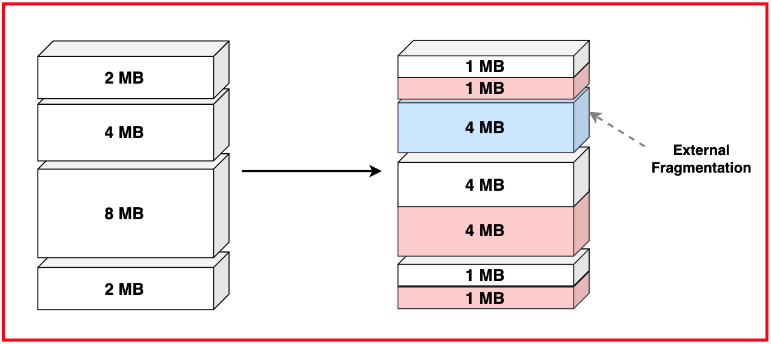


Fragmentation is caused by frequent creation, deletion, and resizing of files. When files are deleted, they leave gaps that may not be large enough to accommodate new files, leading to fragmentation. File resizing can also contribute, as the new size might require additional non-contiguous blocks.

**Types of Fragmentation**

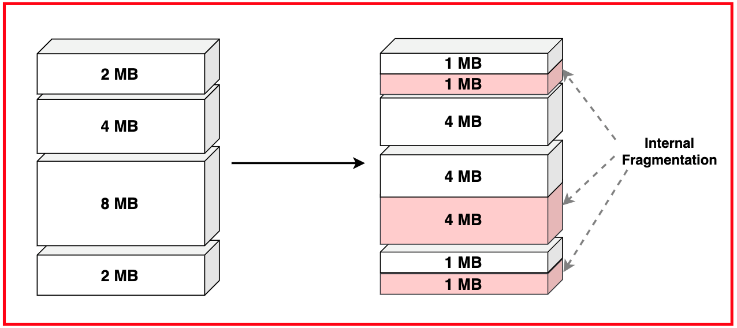
* 1. **External Fragmentation**

External Fragmentation occurs when free space is divided into small, non-contiguous blocks scattered throughout the disk. Even though there is enough total free space to store a new file, the lack of contiguous free space blocks prevents the file from being stored as a single unit.



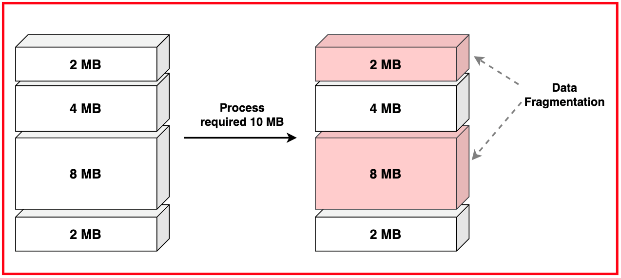
* 1. **Internal Fragmentation**

Internal Fragmentation occurs within allocated space when the allocated blocks are larger than the actual data stored, leaving unused space within the blocks. Fixed block sizes can exacerbate this type of fragmentation.



* 1. **Data Fragmentation**

Data Fragmentation involves files being broken into pieces that are stored in non-contiguous blocks. Linked and indexed allocation methods can lead to data fragmentation, where accessing files requires multiple seek operations.



* 1. **Free Space Fragmentation**

Free Space Fragmentation is the state where free space is available in small, non-contiguous segments. This fragmentation affects the allocation of new files or resizing of existing files, leading to insufficient disk space utilisation.

* 1. **File System Fragmentation**

File System Fragmentation encompasses all types of fragmentation within a file system, including external, internal, data, and free space fragmentation. It reflects the overall inefficiency in file storage and management.

**Impact on Performance**

Fragmentation can significantly degrade system performance in several ways:

* **Slower Read/Write Speeds:** Fragmentation requires the disk head to move more frequently to access different parts of a file, resulting in increased seek times and slower data transfer rates.
* **Increased Disk Wear:** Frequent seeking due to fragmented files causes more mechanical wear on hard drives, potentially shortening their lifespan.
* **Higher CPU Usage:** Fragmentation increases the workload on the CPU as it has to manage and locate fragmented file pieces, leading to higher system resource consumption.
* **Inefficient Storage Utilisation:** Fragmentation leads to inefficient use of disk space, as it becomes difficult to find contiguous free space for new files, leading to wasted storage capacity.
* **System Slowdowns:** Overall system performance can suffer due to the cumulative effects of fragmentation, affecting everything from boot times to application load times and general responsiveness.

**Fragmentation Prevention**

Preventing fragmentation involves strategies to minimise its occurrence:

* **Efficient Allocation Methods:** Using allocation methods that minimise fragmentation, such as indexed allocation, can help. These methods allow files to grow without needing contiguous space.
* **Regular Maintenance:** Scheduling regular defragmentation tasks can help keep fragmentation under control. This process reorganises files and free space to optimise storage layout.
* **Dynamic Block Sizes:** Using file systems that support dynamic block sizes can reduce internal fragmentation by better matching block sizes to file sizes.
* **File System Design:** Choosing file systems that are designed to handle fragmentation efficiently, such as ext4 or NTFS, can help mitigate the impact.
* **Avoiding Frequent Resizing:** Minimising the frequent resizing of files or using file systems that handle resizing efficiently can prevent fragmentation from excessive file growth and shrinking.

**Fragmentation Management**

Managing fragmentation involves both reactive and proactive strategies to maintain optimal performance:

* **Defragmentation Tools:** Using defragmentation tools that rearrange fragmented files and consolidate free space. Most operating systems provide built-in tools for this purpose.
* **Garbage Collection:** Implementing garbage collection in file systems, particularly those used in SSDs, to regularly clean up fragmented and unused space.
* **Monitoring and Alerts:** Using monitoring tools to track fragmentation levels and set alerts for when fragmentation reaches a threshold requiring action.
* **File System Upgrades:** Upgrading to file systems with better fragmentation handling can significantly improve performance and reduce maintenance overhead.
* **Disk Management Policies:** Implementing disk management policies that include regular maintenance schedules, usage patterns that minimise fragmentation, and training users on best practices for file management