UNIT-4

Information Management:

File concept, Access method, Directory structure, Protection File system structure, Allocation methods, Free space management, Directory implementation, Disk structure, Disk Scheduling, Disk management, Swap space management.

Introduction

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks.
- So that the computer system will be convenient to use, the operating system provides a uniform logical view of information storage.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file.
- Files are mapped by the operating system onto physical devices.
- These storage devices are usually non-volatile, so the contents are persistent through power failures and system reboots.

Introduction

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.

Introduction

- A file has a certain defined structure which depends on its type.
- A text file is a sequence of characters organized into lines (and possibly pages).
- A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An object file is a sequence of bytes organized into blocks understandable by the system's linker.
- An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- Name. The symbolic file name is the only information kept in human readable form.
- Identifier. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- Type. This information is needed for systems that support different types of files.
- Location. This information is a pointer to a device and to the location of the file on that device.
- Size. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- Protection. Access-control information determines who can do reading, writing, executing, and so on.
- Time, date, and user identification. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

Directory structure

- The information about all files is kept in the directory structure, which also resides on secondary storage.
- Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.
- It may take more than a kilobyte to record this information for each file.
- In a system with many files, the size of the directory itself may be megabytes.
- Because directories, like files, must be nonvolatile, they must be stored on the device and brought into memory piecemeal, as needed.

Structures of Directory in Operating System

- A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner. In other words, directories are like folders that help organize files on a computer.
- Just like you use folders to keep your papers and documents in order, the operating system uses directories to keep track of files and where they are stored.
- Different structures of directories can be used to organize these files, making it easier to find and manage them.
- Understanding these directory structures is important because it helps in efficiently organizing and accessing files on your computer.

Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries.

Operations that can be performed on a directory:

- **Search for a file**. We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship between files, we may want to be able to find all files whose names match a particular pattern.
- **Create a file.** New files need to be created and added to the directory.
- **Delete a file**. When a file is no longer needed, we want to be able to remove it from the directory.
- **List a directory.** We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
- **Rename a file.** Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
- **Traverse the file system**. We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals. Often, we do this by copying all files to magnetic tape. This technique provides a backup copy in case of system failure. In addition, if a file is no longer in use, the file can be copied to tape and the disk space of that file released for reuse by another file.

Different Types of Directory in OS

In an operating system, there are different types of directory structures that help organize and manage files efficiently.

Each type of directory has its own way of arranging files and directories, offering unique benefits and features.

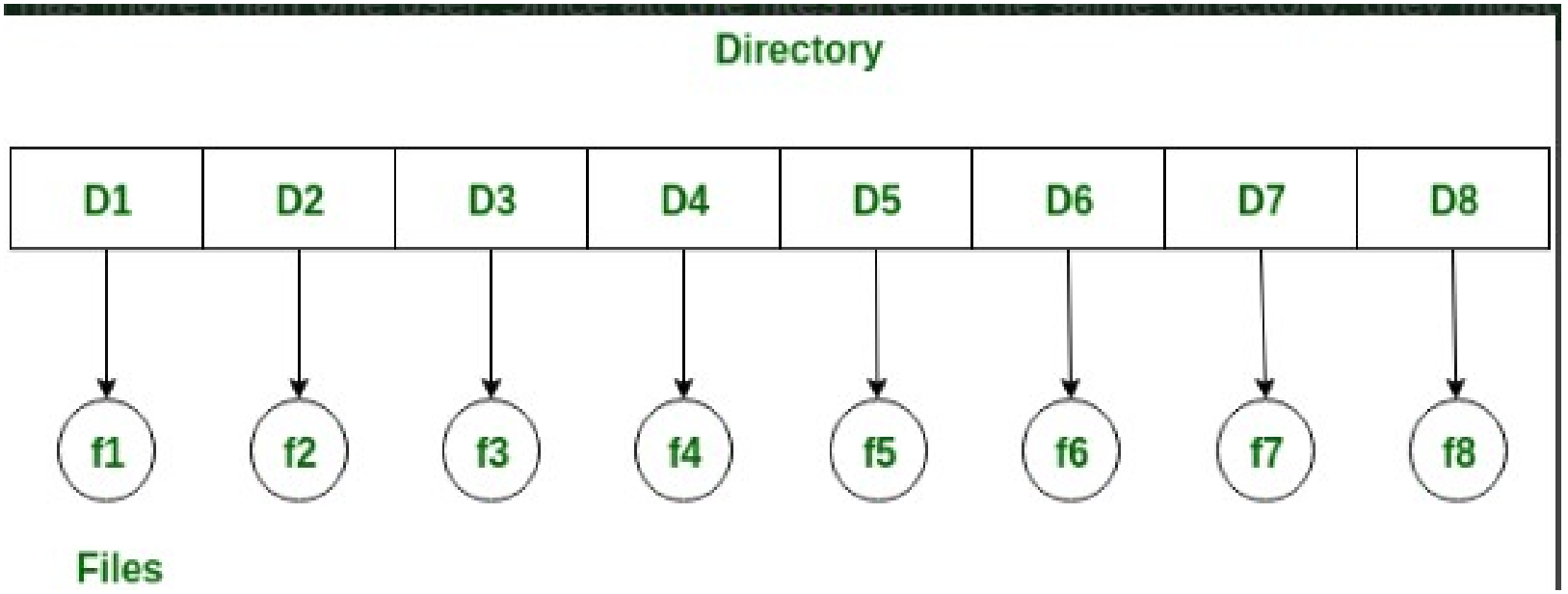
These are:

- Single-Level Directory
- Two-Level Directory
- Tree Structure/ Hierarchical Structure
- Acyclic Graph Structure
- General-Graph Directory Structure

1) Single-Level Directory

- The single-level directory is the **simplest directory structure**. In it, all files are contained in the same directory which makes it easy to support and understand.
- A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user.
- Since all the files are in the same directory, they must have a **unique name**. If two users call their dataset test, then the unique name rule violated.

1) Single-Level Directory



1) Single-Level Directory

Advantages

- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.
- **Logical Organization** : Directory structures help to logically organize files and directories in a hierarchical structure. This provides an easy way to navigate and manage files, making it easier for users to access the data they need.
- **Increased Efficiency:** Directory structures can increase the efficiency of the file system by reducing the time required to search for files. This is because directory structures are optimized for fast file access, allowing users to quickly locate the file they need.
- **Improved Security** : Directory structures can provide better security for files by allowing access to be restricted at the directory level. This helps to prevent unauthorized access to sensitive data and ensures that important files are protected.
- **Facilitates Backup and Recovery** : Directory structures make it easier to backup and recover files in the event of a system failure or data loss. By storing related files in the same directory, it is easier to locate and backup all the files that need to be protected.
- **Scalability:** Directory structures are scalable, making it easy to add new directories and files as needed. This helps to accommodate growth in the system and makes it easier to manage large amounts of data.

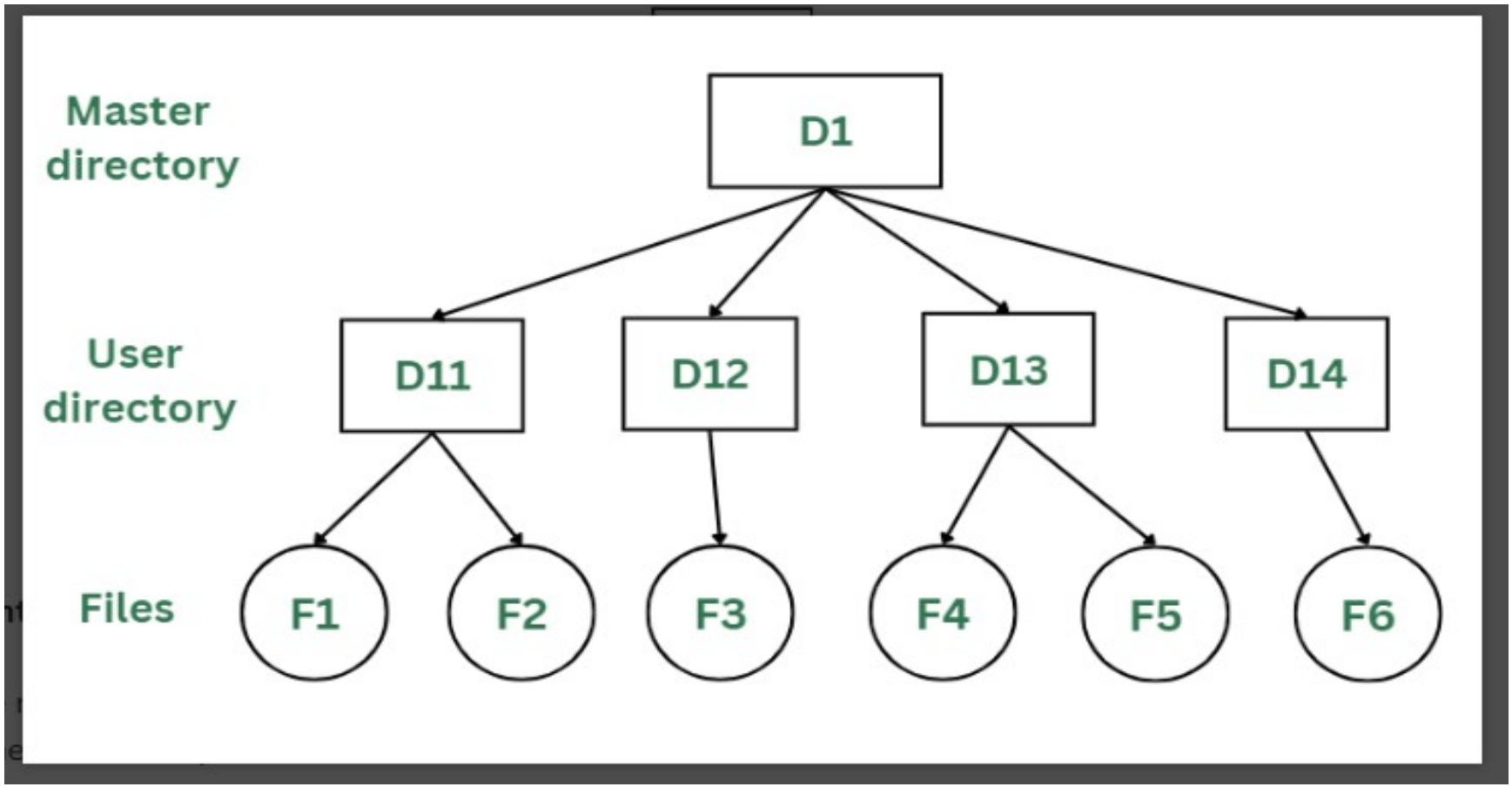
Disadvantages

- There may chance of name collision because two files can have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

2) Two-Level Directory

- As we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a **separate directory for each user**.
- In the two-level directory structure, each user has their own **user files directory (UFD).** The UFDs have similar structures, but each lists only the files of a single user. System's **master file directory (MFD*)*** is searched whenever a new user id is created.

2) Two-Level Directory



2) Two-Level Directory

Advantages

- The main advantage is there can be more than two files with same name, and would be very helpful if there are multiple users.
- A security would be there which would prevent user to access other user's files.
- Searching of the files becomes very easy in this directory structure.

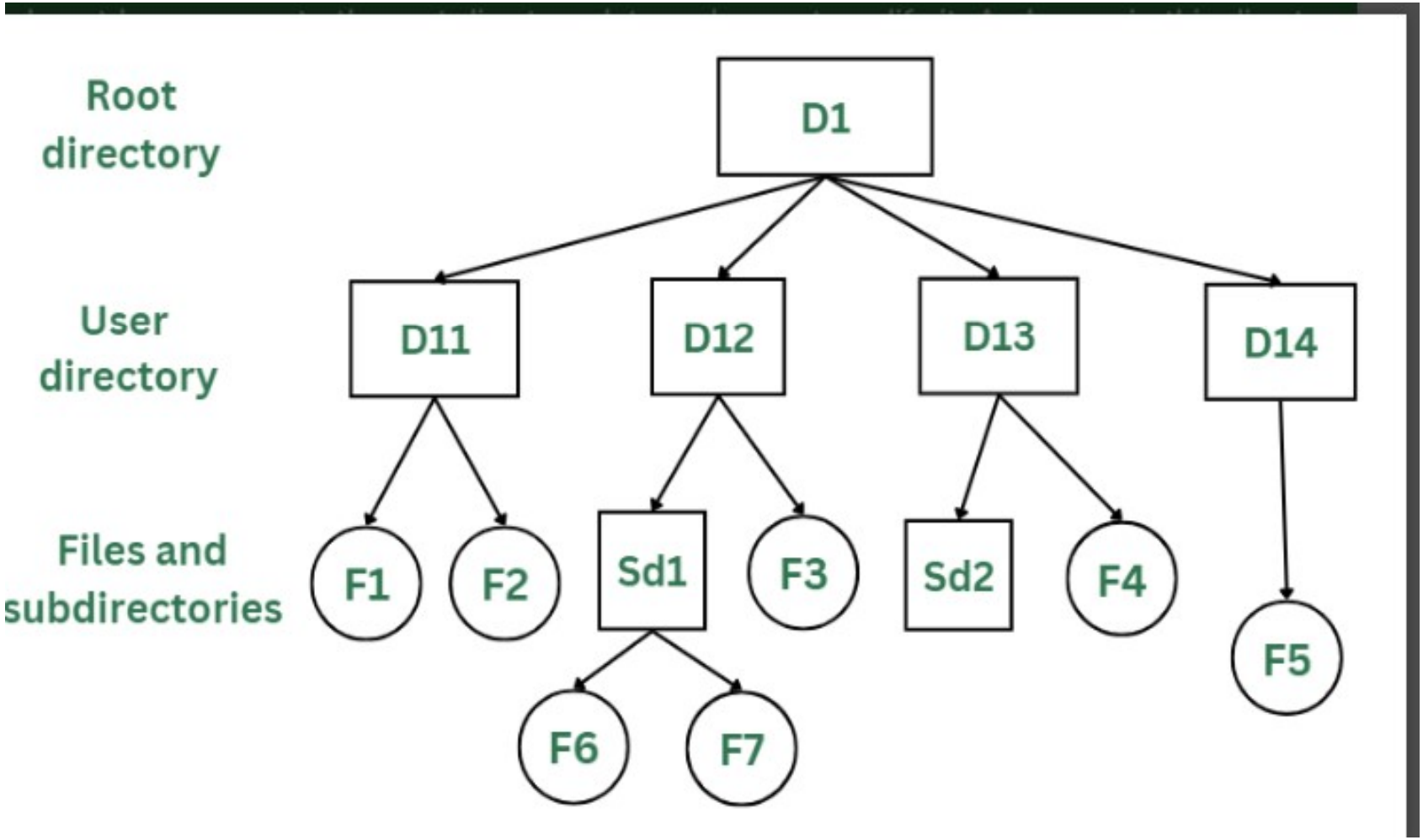
Disadvantages

- As there is advantage of security, there is also disadvantage that the user cannot share the file with the other users.
- Unlike the advantage users can create their own files, users don't have the ability to create subdirectories.
- Scalability is not possible because one user can't group the same types of files together.

3) Tree Structure/ Hierarchical Structure

- Tree directory structure of operating system is most commonly used in our **personal computers**. User can create files and subdirectories too, which was a disadvantage in the previous directory structures.
- This directory structure resembles a real tree upside down, where the **root directory** is at the peak. This root contains all the directories for each user. The users can create subdirectories and even store files in their directory.
- A user do not have access to the root directory data and cannot modify it. And, even in this directory the user do not have access to other user's directories. The structure of tree directory is given below which shows how there are files and subdirectories in each user's directory.

3) Tree Structure/ Hierarchical Structure



3) Tree Structure/ Hierarchical Structure

Advantages

- This directory structure allows subdirectories inside a directory.
- The searching is easier.
- File sorting of important and unimportant becomes easier.
- This directory is more scalable than the other two directory structures explained.

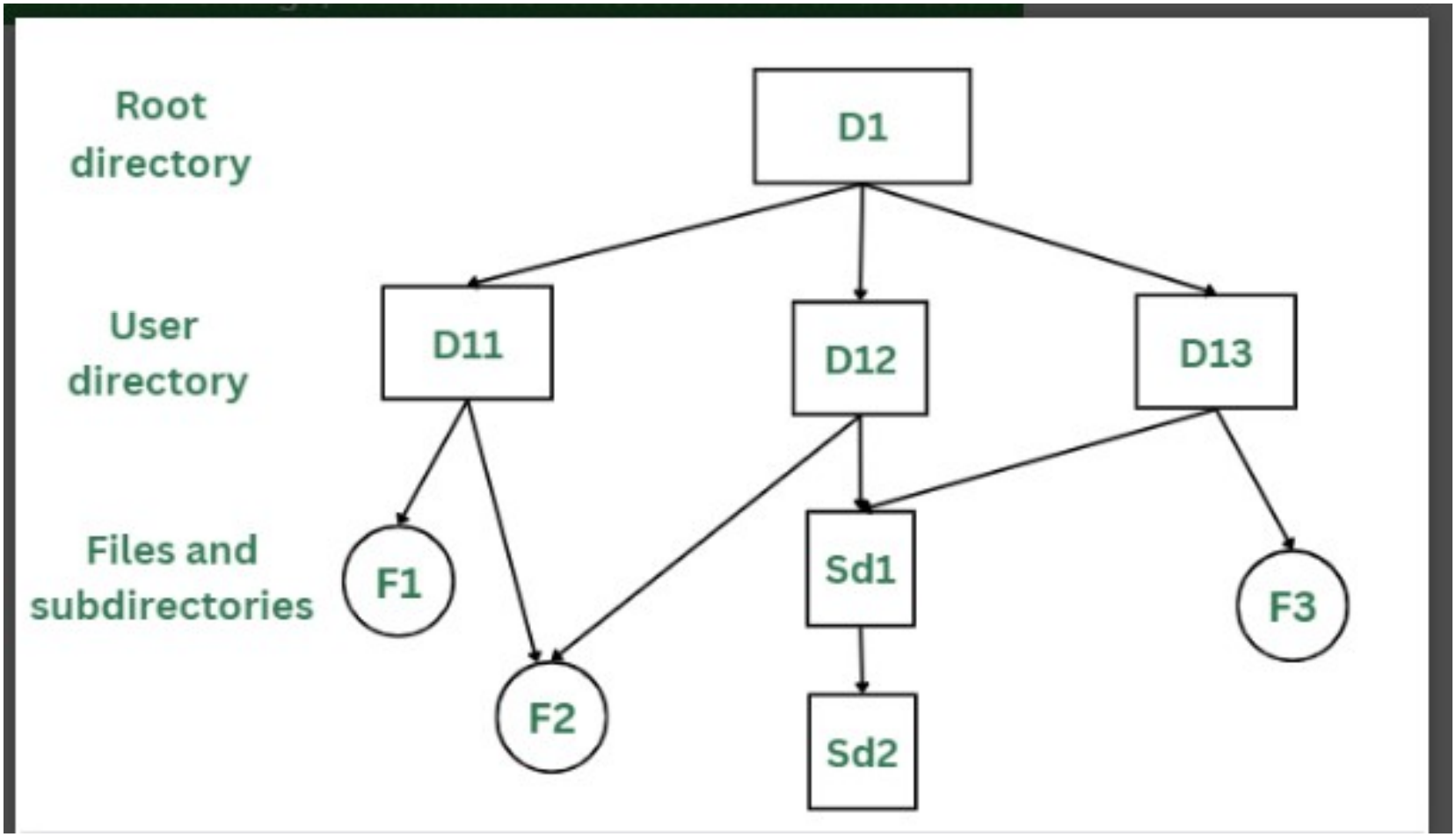
Disadvantages

- As the user isn't allowed to access other user's directory, this prevents the file sharing among users.
- As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
- Users cannot modify the root directory data.
- If files do not fit in one, they might have to be fit into other directories.

4) Acyclic Graph Structure

- As we have seen the above three directory structures, where none of them have the capability to access one file from multiple directories. The file or the subdirectory could be accessed through the directory it was present in, but not from the other directory.
- This problem is solved in acyclic graph directory structure, where a file in one directory can be accessed from multiple directories. In this way, the files could be shared in between the users. It is designed in a way that multiple directories point to a particular directory or file with the help of links.
- In the below figure, this explanation can be nicely observed, where a file is shared between multiple users. If any user makes a change, it would be reflected to both the users.

4) Acyclic Graph Structure



4) Acyclic Graph Structure

Advantages

- Sharing of files and directories is allowed between multiple users.
- Searching becomes too easy.
- Flexibility is increased as file sharing and editing access is there for multiple users.

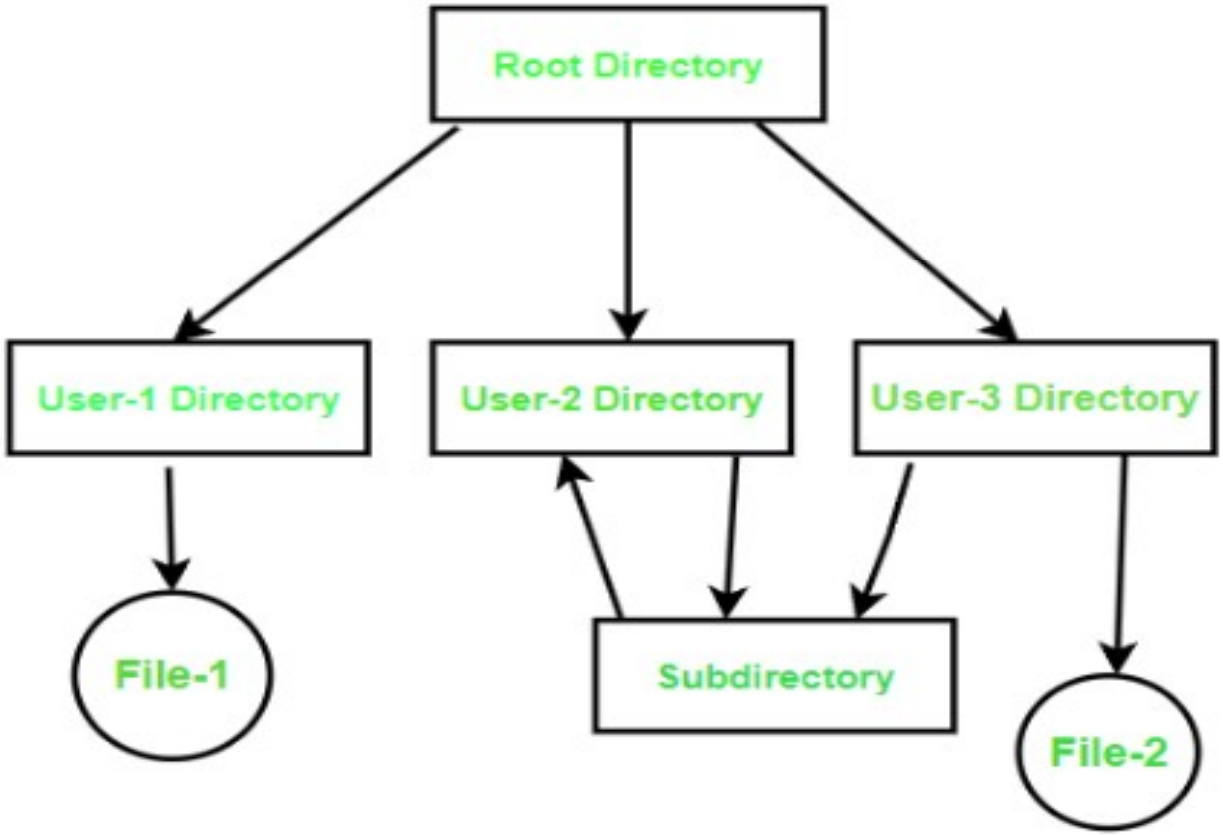
Disadvantages

- Because of the complex structure it has, it is difficult to implement this directory structure.
- The user must be very cautious to edit or even deletion of file as the file is accessed by multiple users.
- If we need to delete the file, then we need to delete all the references of the file in order to delete it permanently.

5) General-Graph Directory Structure

- Unlike the acyclic-graph directory, which avoids loops, the general-graph directory can have cycles, meaning a directory can contain paths that loop back to the starting point.
- This can make navigating and managing files more complex.

A cycle is formed in the User 2 directory. While this structure offers more flexibility, it is also more complicated to implement.



5) General-Graph Directory Structure

Advantages of General-Graph Directory

- More flexible than other directory structures.
- Allows cycles, meaning directories can loop back to each other.

Disadvantages of General-Graph Directory

- More expensive to implement compared to other solutions.
- Requires garbage collection to manage and clean up unused files and directories.

File Operations

- **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file**. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Reading a file.** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file position pointer . Both the read and write operations use this same pointer, saving space and reducing system complexity.
- **Repositioning within a file.** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/0. This file operation is also known as a file seek.
- **Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file.** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged -except for file length-but lets the file be reset to length zero and its file space released.

File Operations

- Other common operations include appending new information to the end of an existing file and renaming an existing file.
- These primitive operations can then be combined to perform other file operations. For instance, we can create a copy of a file, or copy the file to another I/O device, such as a printer or a display, by creating a new file and then reading from the old and writing to the new.
- We also want to have operations that allow a user to get and set the various attributes of a file. For example, we may want to have operations that allow a user to determine the status of a file, such as the file's length, and to set file attributes, such as the file's owner.

Note

- Most of the file operations mentioned involve searching the directory for the entry associated with the named file. To avoid this constant searching, many systems require that an open () system call be made before a file is first used actively.
- The operating system keeps a small table, called the open-file table containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table. create and delete are system calls that work with closed rather than open files.

Information associated with an open file

- File pointer. On systems that do not include a file offset as part of the read() and write() system calls, the system must track the last read write location as a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.

- File-open count. As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open counter tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.

Information associated with an open file

- Disk location of the file. Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

- Access rights. Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/0 requests.

FILE LOCKING

- Some operating systems provide facilities for locking an open file (or sections of a file).
- File locks allow one process to lock a file and prevent other processes from gaining access to it.
- File locks are useful for files that are shared by several processes-for example, a system log file that can be accessed and modified by a number of processes in the system.
- A shared lock is akin to a reader lock in that several processes can acquire the lock concurrently.
- An exclusive lock behaves like a writer lock; only one process at a time can acquire such a lock.
- Operating systems may provide either mandatory or advisory file-locking mechanisms. If a lock is mandatory, then once a process acquires an exclusive lock, the operating system will prevent any other process from accessing the locked file.

File Types

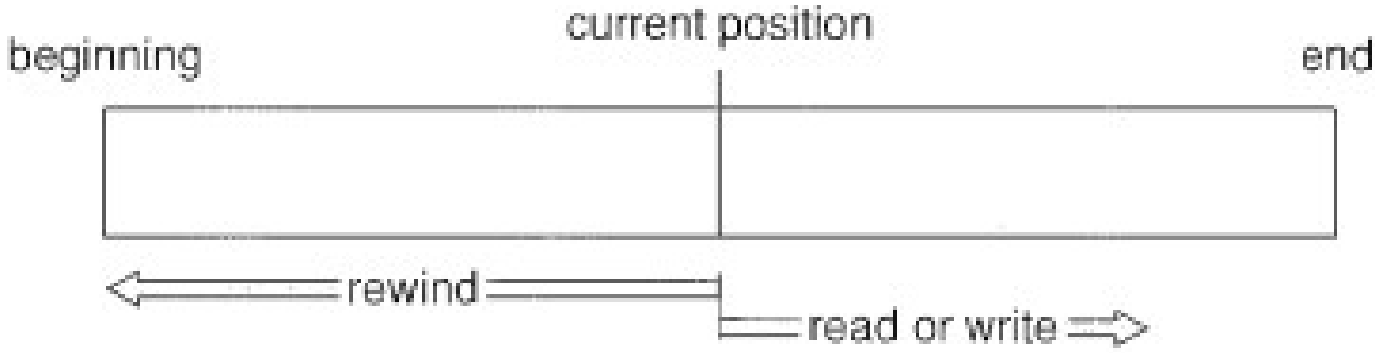
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access methods

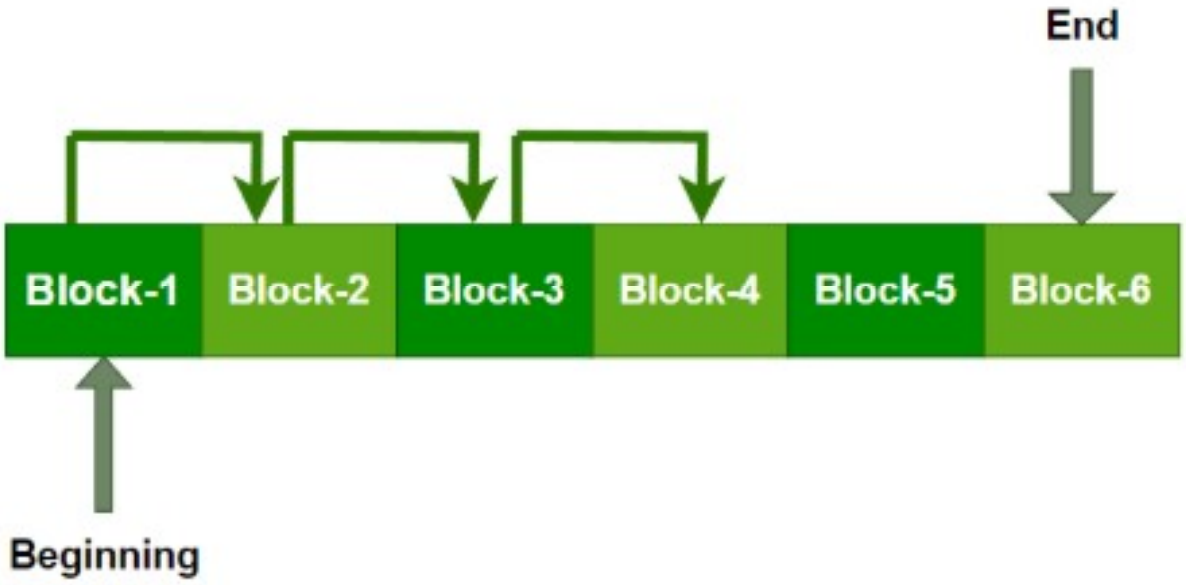
- Files store information. When it is used, this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways. Some systems provide only one access method for files.
- Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

Sequential Access

- The simplest access method is Sequential Access . Information in the file is processed in order, one record after the other.
- This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion. Reads and writes make up the bulk of the operations on a file. A read operation-read next-reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- Similarly, the write operation-write next-appends to the end of the file and advances to the end of the newly written material (the new end of file). Such a file can be reset to the beginning; and on some systems, a program may be able to skip forward or backward n records for some integer n-perhaps only for n = 1.



Sequential Access



Key Points related to Sequential Access

- Data is accessed from one record right after another record in an order.
- When we use the read command, it moves ahead pointer by one.
- When we use the write command, it will allocate memory and move the pointer to the end of the file.
- Such a method is reasonable for tape.

Sequential Access Method

Advantages of Sequential Access Method

- It is simple to implement this file access mechanism.
- It useslexicographic order to quickly access the next entry.
- It is suitable for applications that require access to all records in a file, in a specific order.
- It is less prone to data corruption as the data is written sequentially and not randomly.
- It is a more efficient method for reading large files, as it only reads the required data and does not waste time reading unnecessary data.
- It is a reliable method forbackup and restore operations, as the data is stored sequentially and can be easily restored if required.

Disadvantages of Sequential Access Method

- If the file record that needs to be accessed next is not present next to the current record, this type of file access method is slow.
- Moving a sizable chunk of the file may be necessary to insert a new record.
- It does not allow for quick access to specific records in the file. The entire file must be searched sequentially to find a specific record, which can be time-consuming.
- It is not well-suited for applications that require frequent updates or modifications to the file. Updating or inserting a record in the middle of a large file can be a slow and cumbersome process.
- Sequential access can also result in wasted storage space if records are of varying lengths. The space between records cannot be used by other records, which can result in inefficient use of storage.

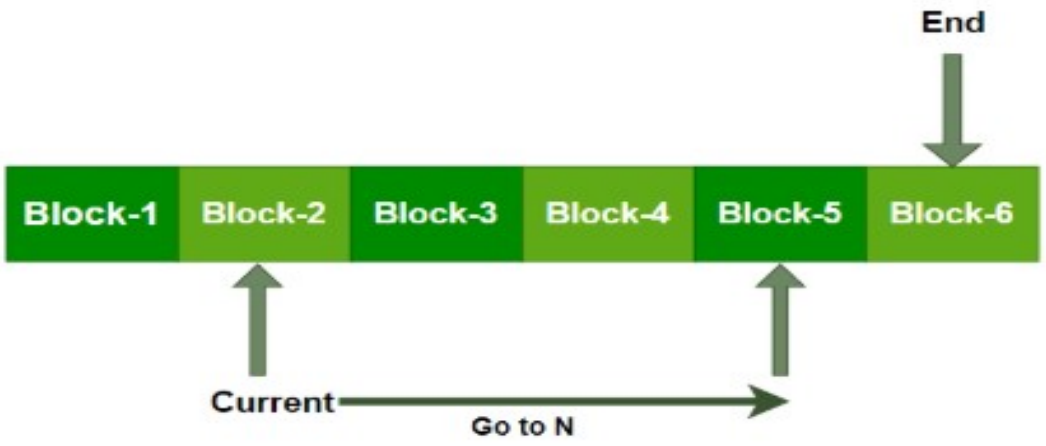
Direct Access/Relative Access

- The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7.
- There are no restrictions on the order of reading or writing for a direct-access file.

Direct Access/Relative Access

- Direct-access files are of great use for immediate access to large amounts of information.
- Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.
- As a simple example, on an airline-reservation system, we might store all the information about a particular flight (for example, flight 713) in the block identified by the flight number. Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store il1formation about a larger set such as people, we might compute a hash function on the people's names or search a small in-memory index to determine a block to read and search.

Direct Access/Relative Access



Direct Access Method

Direct Access Method

Advantages of Direct Access Method

- The files can be immediately accessed decreasing the average access time.
- In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

Disadvantages of Direct Access Method

- **Complex Implementation** : Implementing direct access can be complex, requiring sophisticated algorithms and data structures to manage and locate records efficiently.
- **Higher Storage Overhead** : Direct access methods often require additional storage for maintaining data location information (such as pointers or address tables), which can increase the overall storage requirements.

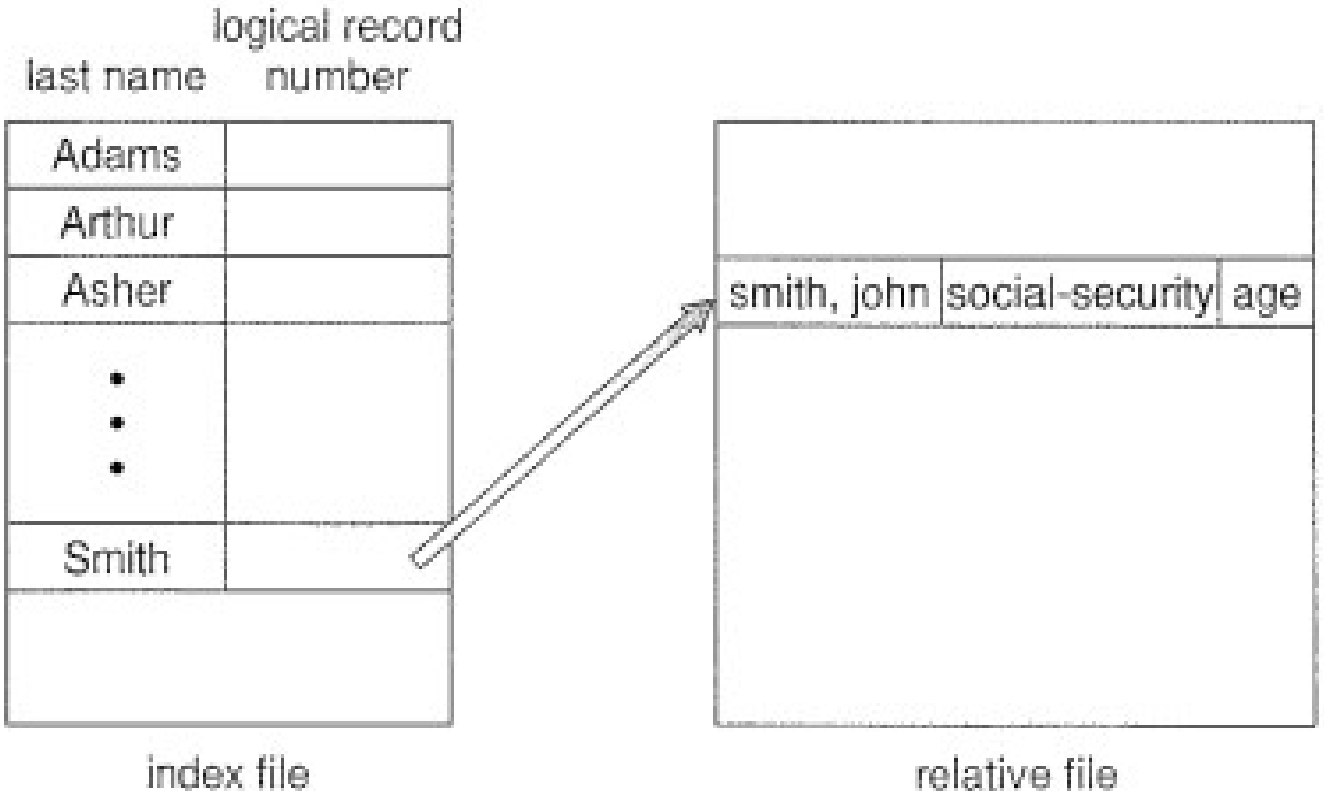
Note

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Other Access Methods

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- It is like an index in the back of a book that contains pointers to the various blocks.
- To find a record in the file, we first search the index and then use the to access the file directly and to find the desired record.

Example of index and relative files.



Index Sequential method

- It is the other method of accessing a file that is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks.
- To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.

Key Points Related to Index Sequential Method

- It is built on top of Sequential access.
- It control the pointer by using index.

Index Sequential Method

Advantages of Index Sequential Method

- **Efficient Searching** : Index sequential method allows for quick searches through the index.
- **Balanced Performance** : It combines the simplicity of sequential access with the speed of direct access, offering a balanced approach that can handle various types of data access needs efficiently.
- **Flexibility** : This method allows both sequential and random access to data, making it versatile for different types of applications, such as batch processing and real-time querying.
- **Improved Data Management** : Indexing helps in better organization and management of data. It makes data retrieval faster and more efficient, especially in large databases.
- **Reduced Access Time** : By using an index to directly locate data blocks, the time spent searching for data within large datasets is significantly reduced.

Disadvantages of Index Sequential Method

- **Complex Implementation** : The index sequential method is more complex to implement and maintain compared to simple sequential access methods.
- **Additional Storage** : Indexes require additional storage space, which can be significant for large datasets. This extra space can sometimes offset the benefits of faster access.
- **Update Overhead** : Updating the data can be more time-consuming because both the data and the indexes need to be updated. This can lead to increased processing time for insertions, deletions, and modifications.
- **Index Maintenance** : Keeping the index up to date requires regular maintenance, especially in dynamic environments where data changes frequently. This can add to the system's overhead.

Relative Record Access

- Relative record access is a file access method used in operating systems where records are accessed relative to the current position of the file pointer. In this method, records are located based on their position relative to the current record, rather than by a specific address or key value.
- **Key Points Related to Relative Record Access**
- Relative record access is a random access method that allows records to be accessed based on their position relative to the current record.
- This method is efficient for accessing individual records but may not be suitable for files that require frequent updates or random access to specific records.
- Relative record access requires fixed-length records and may not be flexible enough for some applications.
- This method is useful for processing records in a specific order or for files that are accessed sequentially.

Relative Record Access

Advantages of Relative Record Access

- **Random Access:** Relative record access allows random access to records in a file. The system can access any record at a specific offset from the current position of the file pointer.
- **Efficient Retrieval:** Since the system only needs to read the current record and any records that need to be skipped, relative record access is more efficient than sequential access for accessing individual records.
- **Useful for Sequential Processing:** Relative record access is useful for processing records in a specific order. For example, if the records are sorted in a specific order, the system can access the next or previous record relative to the current position of the file pointer.

Disadvantages of Relative Record Access

- **Fixed Record Length:** Relative record access requires fixed-length records. If the records are of varying length, it may be necessary to use padding to ensure that each record is the same length.
- **Limited Flexibility:** Relative record access is not very flexible. It is difficult to insert or delete records in the middle of a file without disrupting the relative positions of other records.
- **Limited Application:** Relative record access is best suited for files that are accessed sequentially or with some regularity, but it may not be appropriate for files that are frequently updated or require random access to specific records.

Content Addressable Access

- Content-addressable access (CAA) is a file access method used in operating systems that allows records or blocks to be accessed based on their content rather than their address. In this method, a hash function is used to calculate a unique key for each record or block, and the system can access any record or block by specifying its key.

Keys in Content-Addressable Access

- **Unique:** Each record or block has a unique key that is generated using a hash function.
- **Calculated based on content:** The key is calculated based on the content of the record or block, rather than its location or address.

Key Points Related to Content-Addressable Access

- Content-addressable access is a file access method that allows records or blocks to be accessed based on their content rather than their address.
- CAA uses a hash function to generate a unique key for each record or block.
- CAA is efficient for searching largedatabases or file systems and is more flexible than other access methods.
- CAA requires additional overhead for calculating the hash function and may have collisions or limited key space.

Content-Addressable Access

Advantages of Content-Addressable Access

- **Efficient Search:** CAA is ideal for searching large databases or file systems because it allows for efficient searching based on the content of the records or blocks.
- **Flexibility:** CAA is more flexible than other access methods because it allows for easy insertion and deletion of records or blocks.
- **Data Integrity** : CAA ensures data integrity because each record or block has a unique key that is generated based on its content.

Disadvantages of Content-Addressable Access

- **Overhead:** CAA requires additional overhead because the hash function must be calculated for each record or block.
- **Collision:** There is a possibility of collision where two records or blocks can have the same key. This can be minimized by using a good hash function, but it cannot be completely eliminated.
- **Limited Key Space:** The key space is limited by the size of the hash function used, which can lead to collisions and other issues.

Protection File system structure

- File protection in an operating system is the process of securing files from unauthorized access, alteration, or deletion.
- It is critical for data security and ensures that sensitive information remains confidential and secure.
- Operating systems provide various mechanisms and techniques such as file permissions, encryption, access control lists, auditing, and physical file security to protect files.
- Proper file protection involves user authentication, authorization, access control, encryption, and auditing. Ongoing updates and patches are also necessary to prevent security breaches.
- File protection in an operating system is essential to maintain data security and minimize the risk of data breaches and other security incidents.

Type of File protection

- **File Permissions** − File permissions are a basic form of file protection that controls access to files by setting permissions for users and groups. File permissions allow the system administrator to assign specific access rights to users and groups, which can include read, write, and execute privileges. These access rights can be assigned at the file or directory level, allowing users and groups to access specific files or directories as needed. File permissions can be modified by the system administrator at any time to adjust access privileges, which helps to prevent unauthorized access.
- **Encryption** − Encryption is the process of converting plain text into ciphertext to protect files from unauthorized access. Encrypted files can only be accessed by authorized users who have the correct encryption key to decrypt them. Encryption is widely used to secure sensitive data such as financial information, personal data, and other confidential information. In an operating system, encryption can be applied to individual files or entire directories, providing an extra layer of protection against unauthorized access.
- **Access Control Lists (ACLs)** − Access control lists (ACLs) are lists of permissions attached to files and directories that define which users or groups have access to them and what actions they can perform on them. ACLs can be more granular than file permissions, allowing the system administrator to specify exactly which users or groups can access specific files or directories. ACLs can also be used to grant or deny specific permissions, such as read, write, or execute privileges, to individual users or groups.
- **Auditing and Logging** − Auditing and logging are mechanisms used to track and monitor file access, changes, and deletions. It involves creating a record of all file access and changes, including who accessed the file, what actions were performed, and when they were performed. Auditing and logging can help to detect and prevent unauthorized access and can also provide an audit trail for compliance purposes.
- **Physical File Security** − Physical file security involves protecting files from physical damage or theft. It includes measures such as file storage and access control, backup and recovery, and physical security best practices. Physical file security is essential for ensuring the integrity and availability of critical data, as well as compliance with regulatory requirements.

Protection File system structure

- The files which have direct access of the any user have the need of protection.
- The files which are not accessible to other users doesn't require any kind of protection. The mechanism of the protection provide the facility of the controlled access by just limiting the types of access to the file.

- Access can be given or not given to any user depends on several factors, one of which is the type of access required. Several different types of operations can be controlled:

Read – Reading from a file.

Write – Writing or rewriting the file.

Execute – Loading the file and after loading the execution process starts.

Append – Writing the new information to the already existing file, editing must be end at the end of the existing file.

Delete – Deleting the file which is of no use and using its space for the another data.

List – List the name and attributes of the file.

- Operations like renaming, editing the existing file, copying; these can also be controlled. There are many protection mechanism. each of them mechanism have different advantages and disadvantages and must be appropriate for the intended application.

Access Control :

- There are different methods used by different users to access any file. The general way of protection is to associate *identity-dependent access* with all the files and directories an list called access-control list (ACL) which specify the names of the users and the types of access associate with each of the user. The main problem with the access list is their length. If we want to allow everyone to read a file, we must list all the users with the read access. This technique has two undesirable consequences:
- Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of the users in the system.
- Previously, the entry of the any directory is of the fixed size but now it changes to the variable size which results in the complicates space management. These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classification of users in connection with each file:
- **Owner –** Owner is the user who has created the file.
- **Group –** A group is a set of members who has similar needs and they are sharing the same file.
- **Universe –** In the system, all other users are under the category called universe.

File Permissions

To change access control protection on a file system in Linux, you can use the chmod command to modify permissions, chown to change ownership, and chgrp to change the group, or leverage Access Control Lists (ACLs) using setfacl for more granular control.

1. Understanding File Permissions:

chmod: This command modifies file and directory permissions, allowing you to control who can read, write, and execute them.

Symbolic Mode: Uses letters (u, g, o, a) for user, group, others, and all, and operators (+, -, =) to add, remove, or set permissions (r, w, x).

Example: chmod u+rwx file.txt (gives user read, write, and execute permissions).

Numeric Mode: Uses octal numbers to represent permissions (e.g., 755, 644).

Example: chmod 755 file.txt (owner: read, write, execute; group and others: read, execute).

chown: Changes the owner of a file or directory.

Example: chown user:group file.txt (changes owner to 'user' and group to 'group').

chgrp: Changes the group of a file or directory.

Example: chgrp group file.txt (changes group to 'group').

2. Access Control Lists (ACLs):

setfacl: Allows you to set ACLs for more granular control, enabling permissions for specific users or groups beyond the standard owner, group, and others.

Example: setfacl -m u:user:rwx file.txt (gives user 'user' read, write, and execute permissions).

getfacl: Displays the ACLs for a file or directory.

Example: getfacl file.txt.

File Permissions

3. File System Structure:

Linux uses a hierarchical file system, with everything residing under the root directory /.

Understanding the file system structure is important for managing permissions and ACLs effectively.

The File system Hierarchy Standard (FHS) provides conventions for organizing the file system.

4. Examples:

Make a file executable for the owner: chmod u+x script.sh.

Grant read access to a group: chmod g+r file.txt.

Change ownership of a file to user 'john' and group 'developers': chown john:developers file.txt.

Set an ACL to allow user 'alice' read access: setfacl -m u:alice:r file.txt.

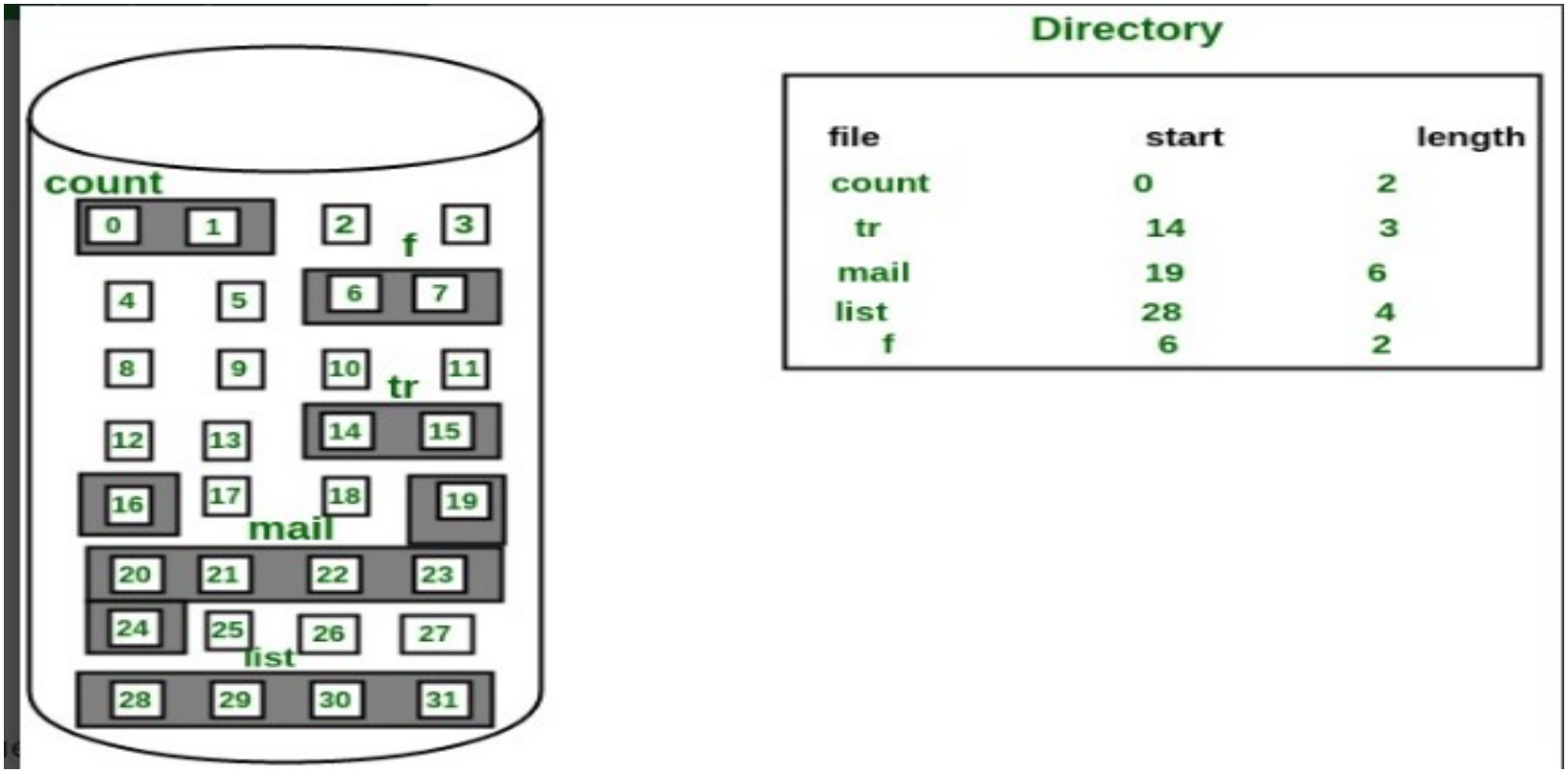
File Allocation Methods

- The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation
- The main idea behind these methods is to provide:
  - Efficient disk space utilization.
  - Fast access to the file blocks.

1. Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1*. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

  The directory entry for a file with contiguous allocation contains
- Address of starting block
- Length of the allocated portion.
- The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



1. Contiguous Allocation

Advantages:

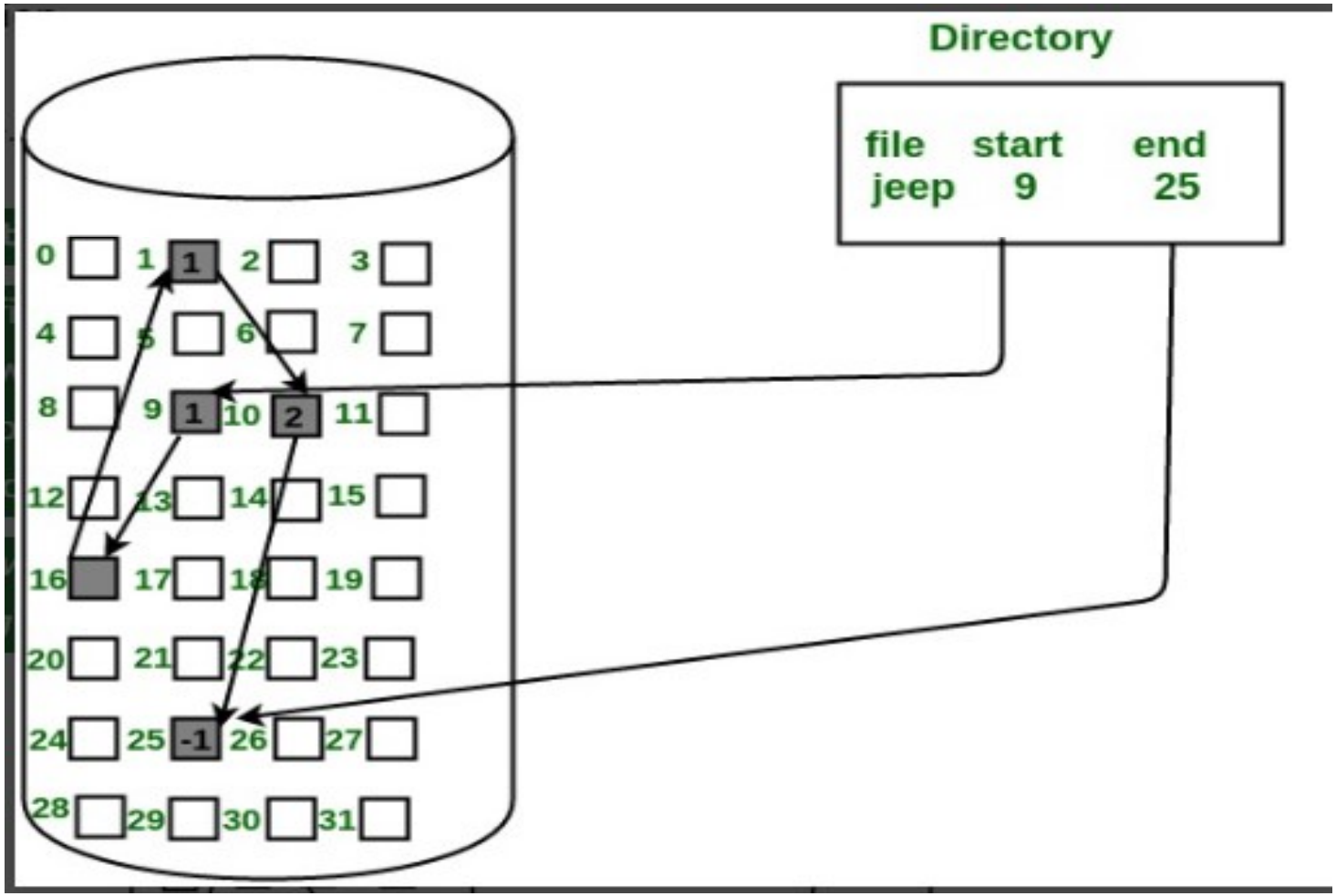
- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.
  The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- *The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.*



2. Linked List Allocation

Advantages:

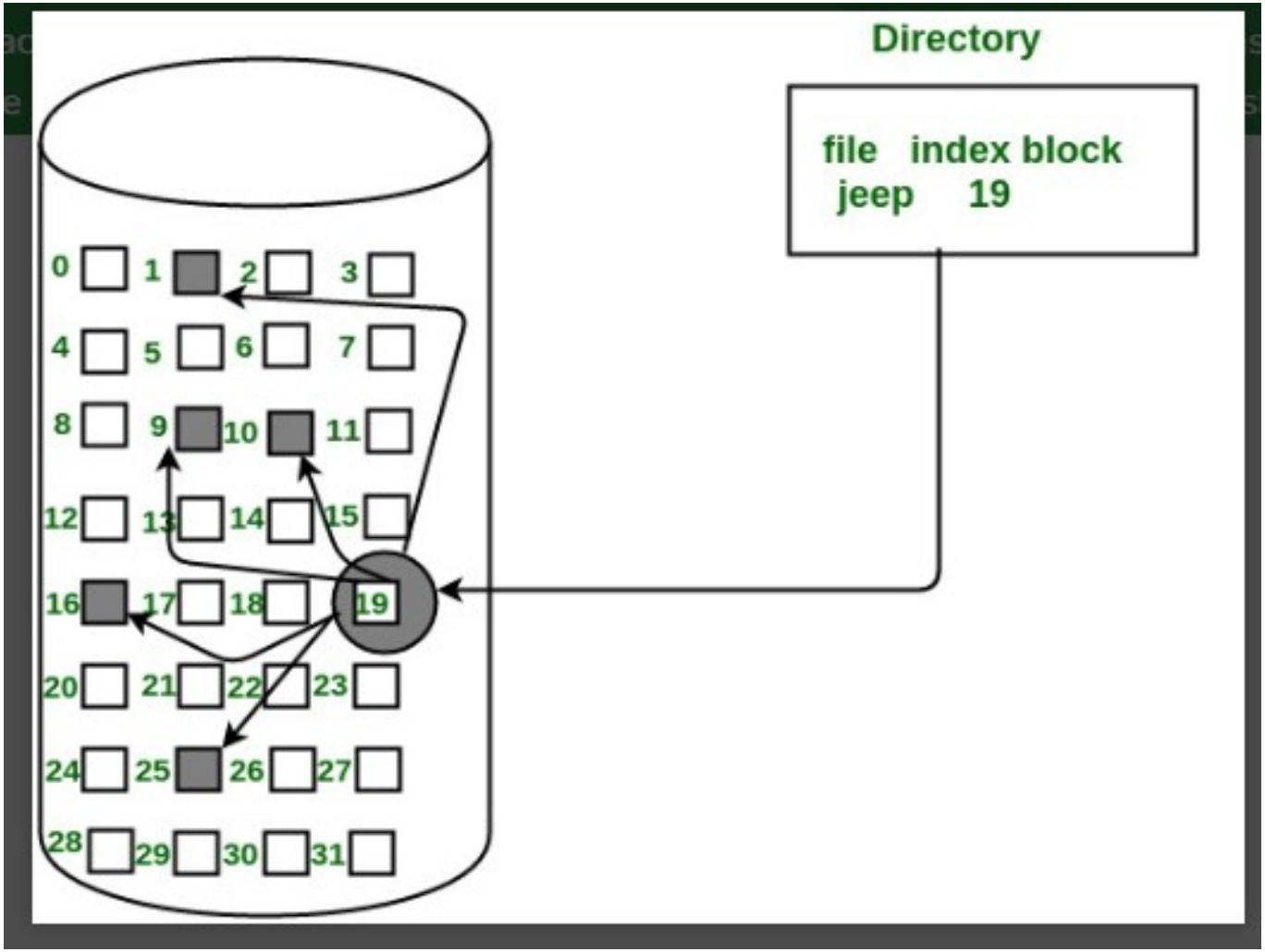
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



3. Indexed Allocation

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Free Space Management Techniques

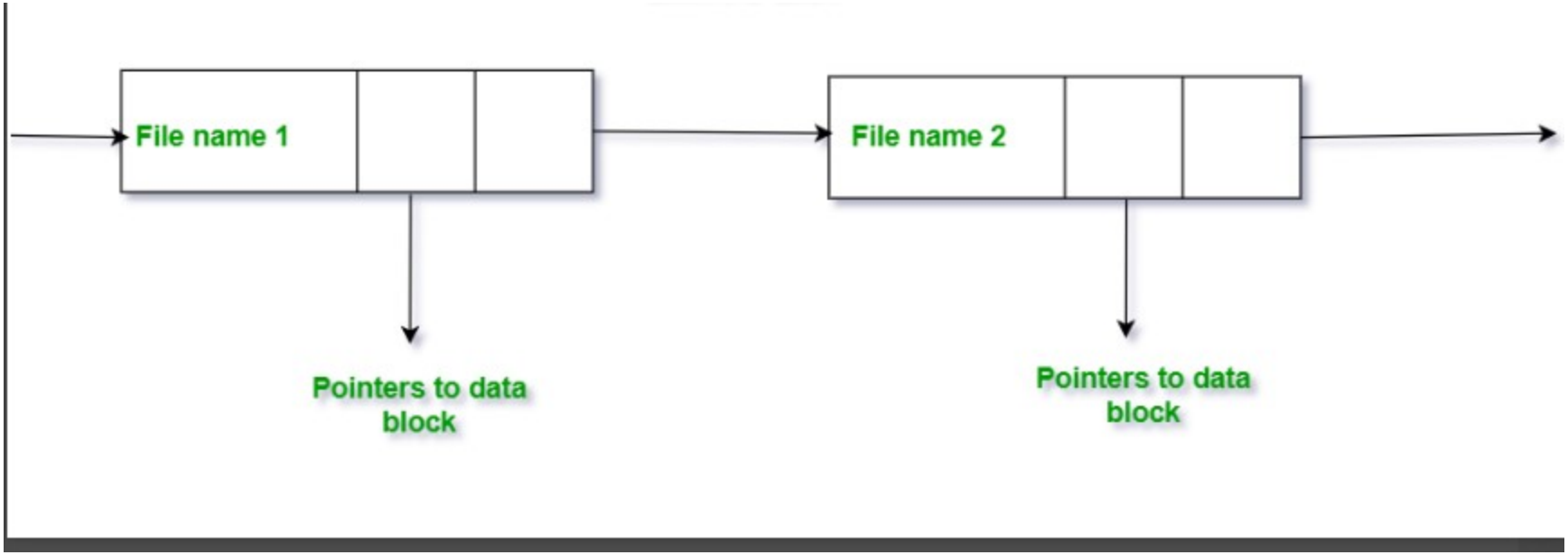
- **Linked Allocation:** In this technique, each file is represented by a linked list of disk blocks. When a file is created, the operating system finds enough free space on the disk and links the blocks of the file to form a chain. This method is simple to implement but can lead to fragmentation and waste of space.
- **Contiguous Allocation:** In this technique, each file is stored as a contiguous block of disk space. When a file is created, the operating system finds a contiguous block of free space and assigns it to the file. This method is efficient as it minimizes fragmentation but suffers from the problem of external fragmentation.
- **Indexed Allocation:** In this technique, a separate index block is used to store the addresses of all the disk blocks that make up a file. When a file is created, the operating system creates an index block and stores the addresses of all the blocks in the file. This method is efficient in terms of storage space and minimizes fragmentation.
- **File Allocation Table (FAT):** In this technique, the operating system uses a file allocation table to keep track of the location of each file on the disk. When a file is created, the operating system updates the file allocation table with the address of the disk blocks that make up the file. This method is widely used in Microsoft Windows operating systems.
- **Volume Shadow Copy:** This is a technology used in Microsoft Windows operating systems to create backup copies of files or entire volumes. When a file is modified, the operating system creates a shadow copy of the file and stores it in a separate location. This method is useful for data recovery and protection against accidental file deletion.

Directory Implementation in Operating System

- Directory implementation in the operating system can be done using Singly Linked List and Hash table.
- The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms.
- There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

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.
- Here we implement a directory by using a linear list of filenames with pointers to the data blocks.



Directory Implementation using Singly Linked List

- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.
- In order to delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it.
- To reuse the directory entry we can mark that entry as unused or we can append it to the list of free directories.
- To delete a file linked list is the best choice as it takes less time.

Disadvantage

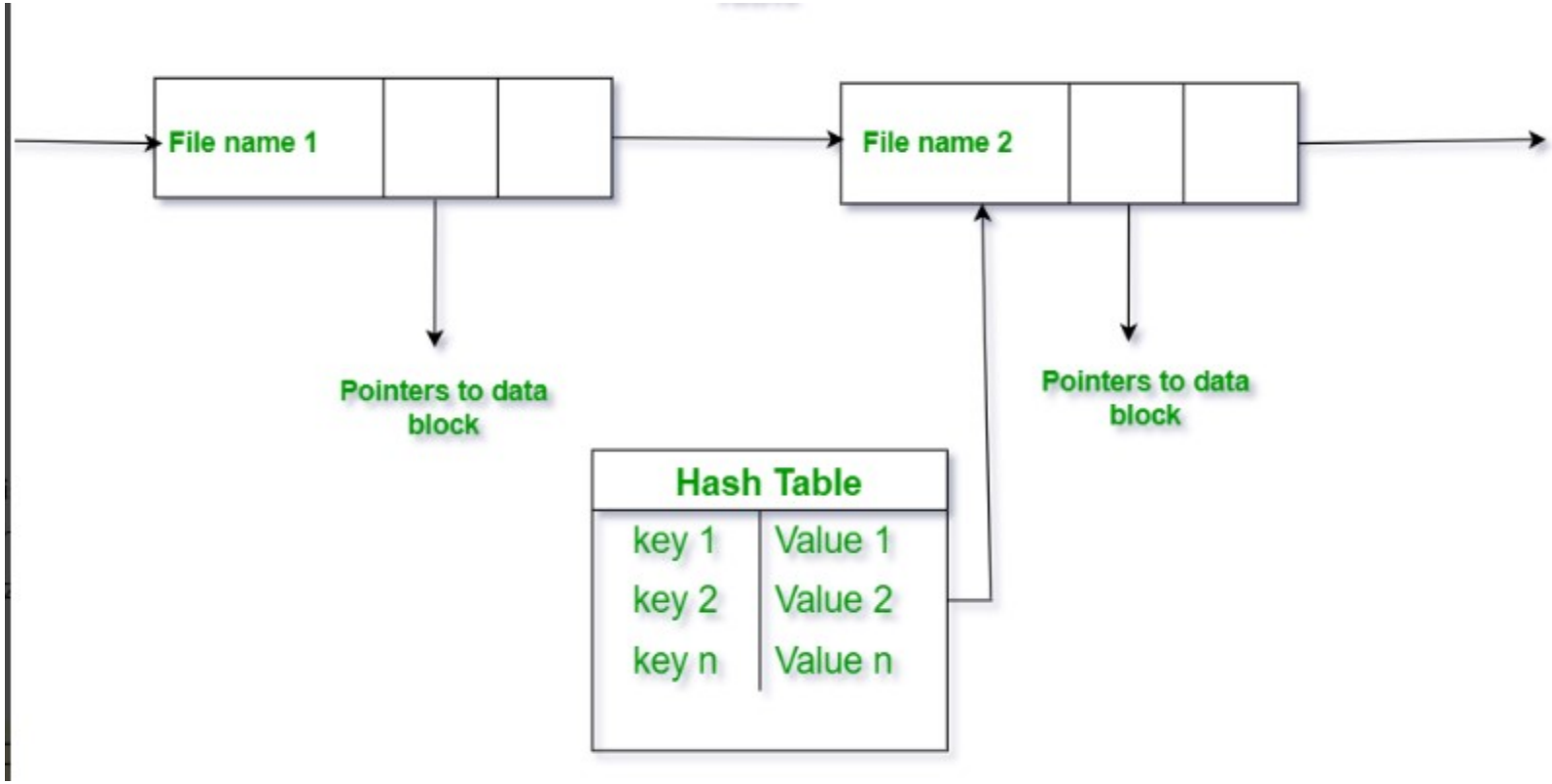
- The main disadvantage of using a linked list is that when the user needs to find a file the user has to do a linear search. In today's world directory information is used quite frequently and linked list implementation results in slow access to a file. So the operating system maintains a cache to store the most recently used directory information.

Directory Implementation using Hash Table

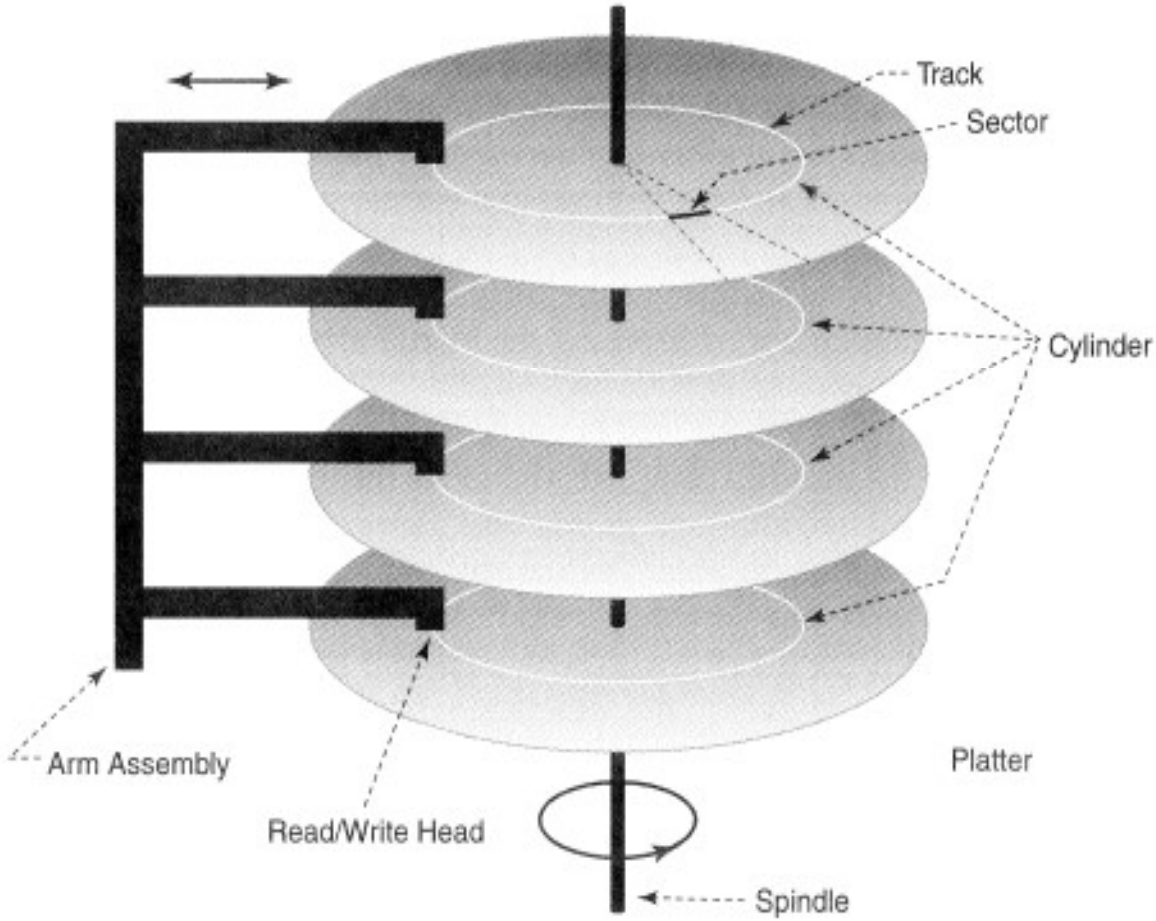
- An alternative data structure that can be used for directory implementation is a hash table. It overcomes the major drawbacks of directory implementation using a linked list. In this method, we use a hash table along with the linked list. Here the linked list stores the directory entries, but a hash data structure is used in combination with the linked list.
- In the hash table for each pair in the directory key-value pair is generated. The hash function on the file name determines the key and this key points to the corresponding file stored in the directory. This method efficiently decreases the directory search time as the entire list will not be searched on every operation. Using the keys the hash table entries are checked and when the file is found it is fetched.

Disadvantage:

- The major drawback of using the hash table is that generally, it has a fixed size and its dependency on size. But this method is usually faster than linear search through an entire directory using a linked list.



Disk structure



Disk structure

- In an operating system, disk structure refers to **the physical organization of data on a disk, typically involving platters, tracks, sectors, and cylinders, which are then logically organized into partitions and file systems for efficient storage and retrieval**.

Physical Organization:

- **Platters:** Hard disks consist of multiple circular platters coated with magnetic material.
- **Tracks:** Each platter has concentric tracks, which are circular paths where data is stored.
- **Sectors:** Tracks are further divided into sectors, the smallest units of storage on a disk, typically holding 512 bytes of data.
- **Cylinders:** A cylinder is a set of corresponding tracks on all platters, representing a logical unit of storage.
- **Read/Write Heads:** Each platter has a read/write head, which reads and writes data on the corresponding tracks.

Logical Organization:

- **Partitions:** The disk is logically divided into partitions, which are independent storage areas that can be formatted with different file systems.
- **File Systems:** The operating system uses a file system (e.g., NTFS, FAT32) to organize files and directories within a partition, allowing for efficient storage and retrieval of data.
- **Logical Blocks:** Even though the disk is physically arranged as sectors and tracks, the data is logically arranged and addressed as an array of blocks of fixed size.
- **Disk Scheduling:** Disk scheduling algorithms (e.g., FCFS, SSTF, SCAN) are used to optimize the order in which disk I/O requests are serviced, improving disk bandwidth and access time.

Disk scheduling

- **Disk scheduling algorithms** are crucial in managing how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed, significantly impacting the speed and efficiency of data access. Common disk scheduling methods include First-Come, First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, C-SCAN, LOOK, and C-LOOK. By understanding and implementing these algorithms, we can optimize system performance and ensure faster data retrieval.
- Disk scheduling is a technique operating systems use to manage the order in which disk I/O (input/output) requests are processed.
- Disk scheduling is also known as I/O Scheduling.
- The main goals of disk scheduling are to optimize the performance of disk operations, reduce the time it takes to access data and improve overall system efficiency.

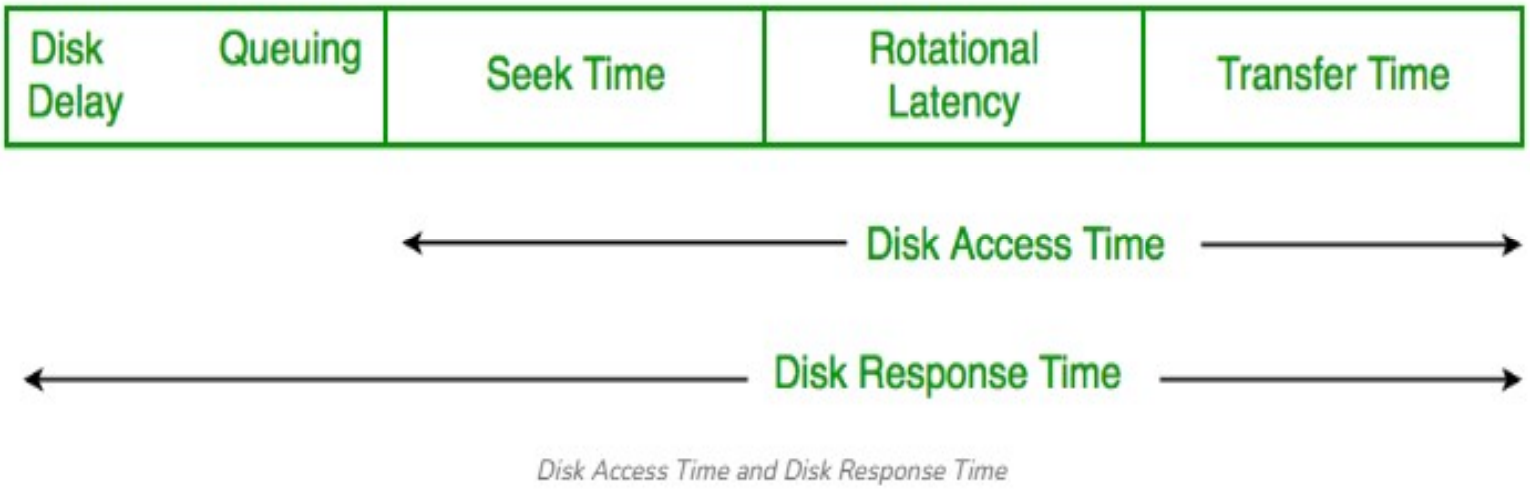
Importance of Disk Scheduling in Operating System

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more requests may be far from each other so this can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Key Terms Associated with Disk Scheduling

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written. So the disk scheduling algorithm that gives a minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of the disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and the number of bytes to be transferred.
- **Disk Access Time:**
- Disk Access Time = Seek Time + Rotational Latency + Transfer Time
- Total Seek Time = Total head Movement * Seek Time

Disk Response Time: Response Time is the average time spent by a request waiting to perform its I/O operation. The average *Response time* is the response time of all requests. *Variance Response Time* is the measure of how individual requests are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.



Disk Access Time and Disk Response Time

Disk Scheduling Algorithms

Goal of Disk Scheduling Algorithms

- Minimize Seek Time
- Maximize Throughput
- Minimize Latency
- Fairness
- Efficiency in Resource Utilization

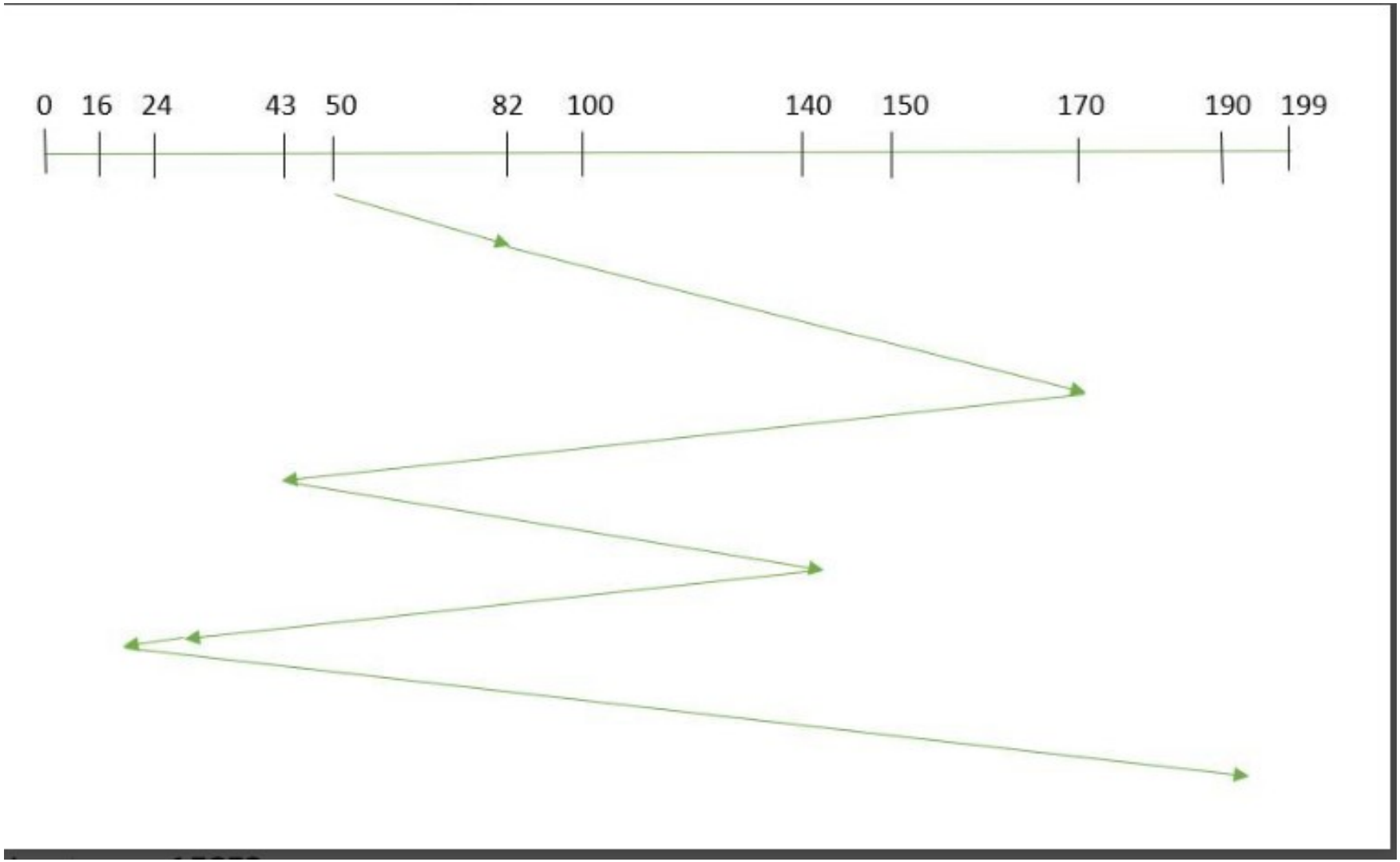
Disk Scheduling Algorithms

- There are several Disk Several Algorithms. We will discuss in detail each one of them.
- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN
- C-SCAN
- LOOK
- C-LOOK
- RSS (Random Scheduling)
- **LIFO (Last-In First-Out)**
- **N-STEP SCAN**
- F-SCAN

1. FCFS (First Come First Serve)

- FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50



Example:

- Suppose the order of request is- (82,170,43,140,24,16,190)
  And current position of Read/Write head is: 50
- So, total overhead movement (total distance covered by the disk arm) =
  (82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

Advantages of FCFS

- Here are some of the advantages of First Come First Serve.
- Every request gets a fair chance
- No indefinite postponement

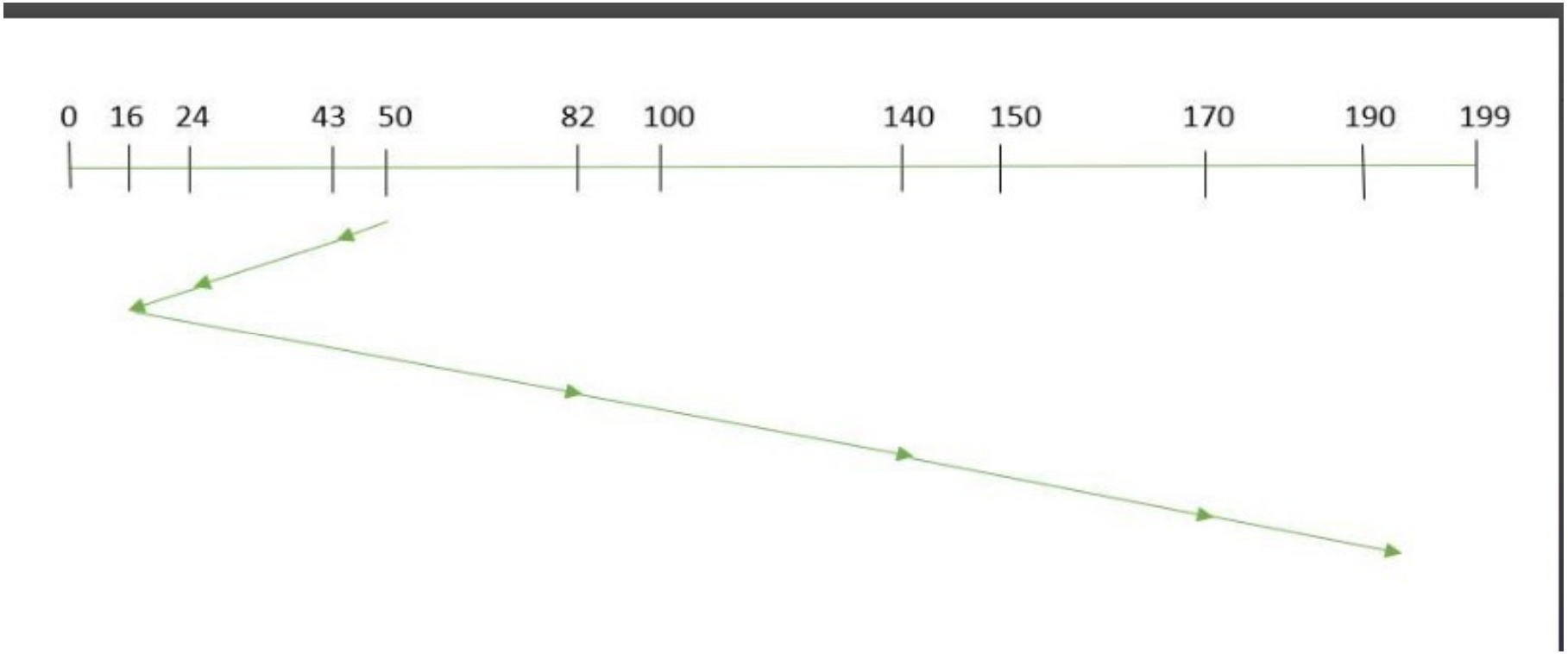
Disadvantages of FCFS

- Here are some of the disadvantages of First Come First Serve.
- Does not try to optimize seek time
- May not provide the best possible service

2. SSTF (Shortest Seek Time First)

- In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.
- As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system.

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50



Example

- Total overhead movement (total distance covered by the disk arm) = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

Advantages of Shortest Seek Time First

- Here are some of the advantages of Shortest Seek Time First.
- The average Response Time decreases
- Throughput increases

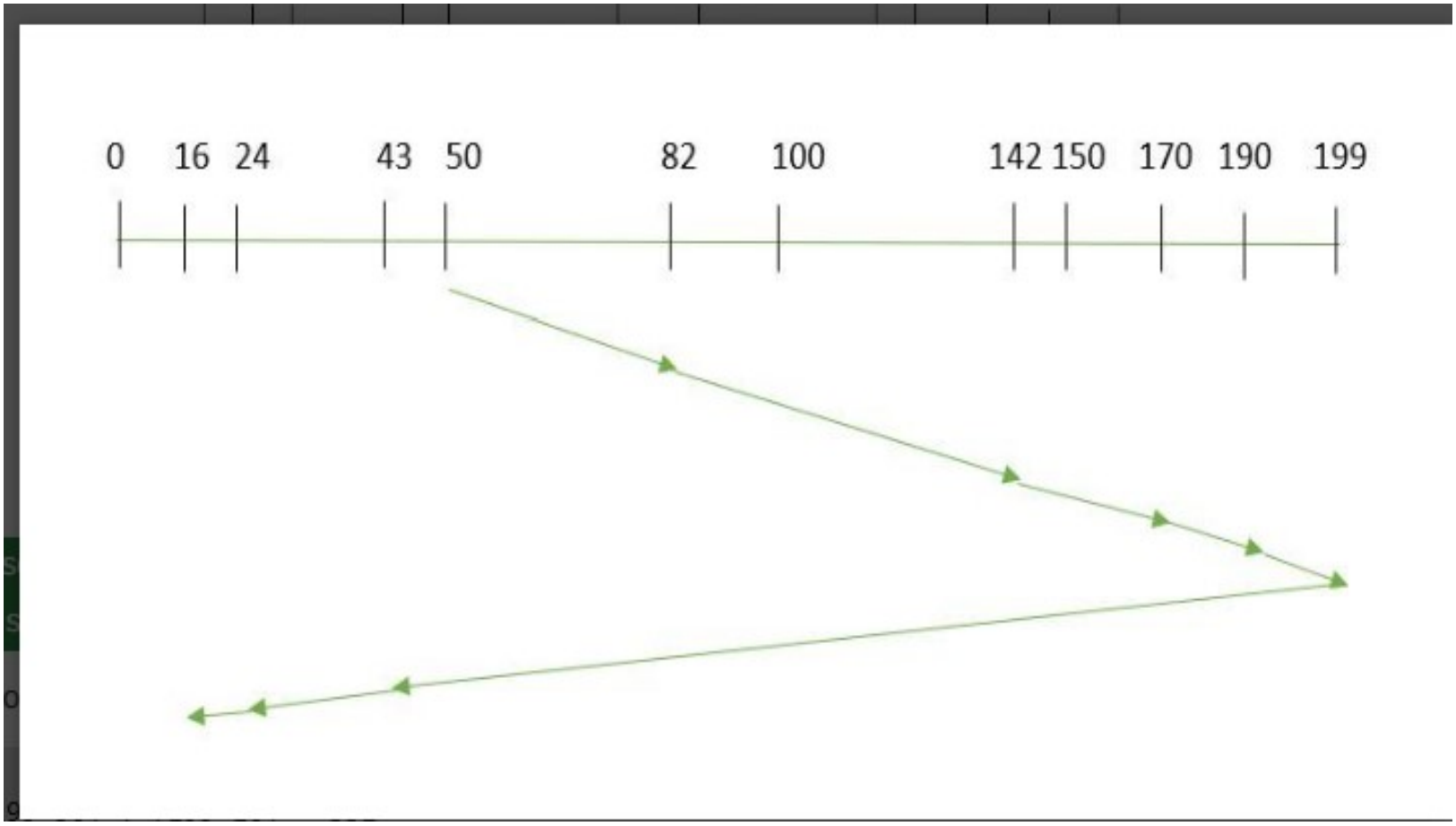
Disadvantages of Shortest Seek Time First

- Here are some of the disadvantages of Shortest Seek Time First.
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

3. SCAN

- In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path.
- So, this algorithm works as an elevator and is hence also known as an **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



SCAN Algorithm

- Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as
- = (199-50) + (199-16) = 332

Advantages of SCAN Algorithm

- Here are some of the advantages of the SCAN Algorithm.
- High throughput
- Low variance of response time
- Average response time

Disadvantages of SCAN Algorithm

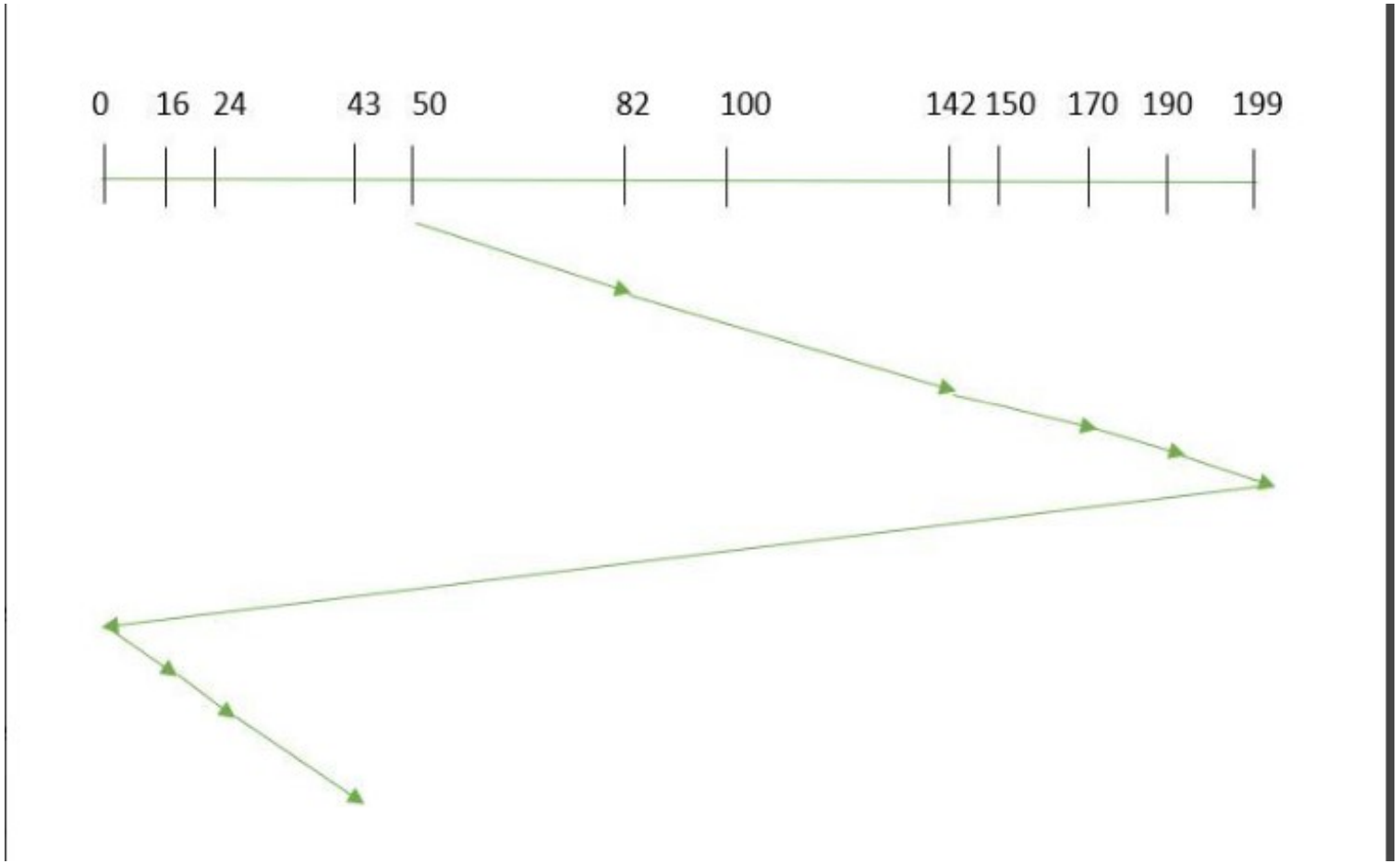
- Here are some of the disadvantages of the SCAN Algorithm.
- Long waiting time for requests for locations just visited by disk arm

4. C-SCAN

- In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

- These situations are avoided in the *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



C-SCAN Algorithm

- So, the total overhead movement (total distance covered by the disk arm) is calculated as:
- =(199-50) + (199-0) + (43-0) = 391

Advantages of C-SCAN Algorithm

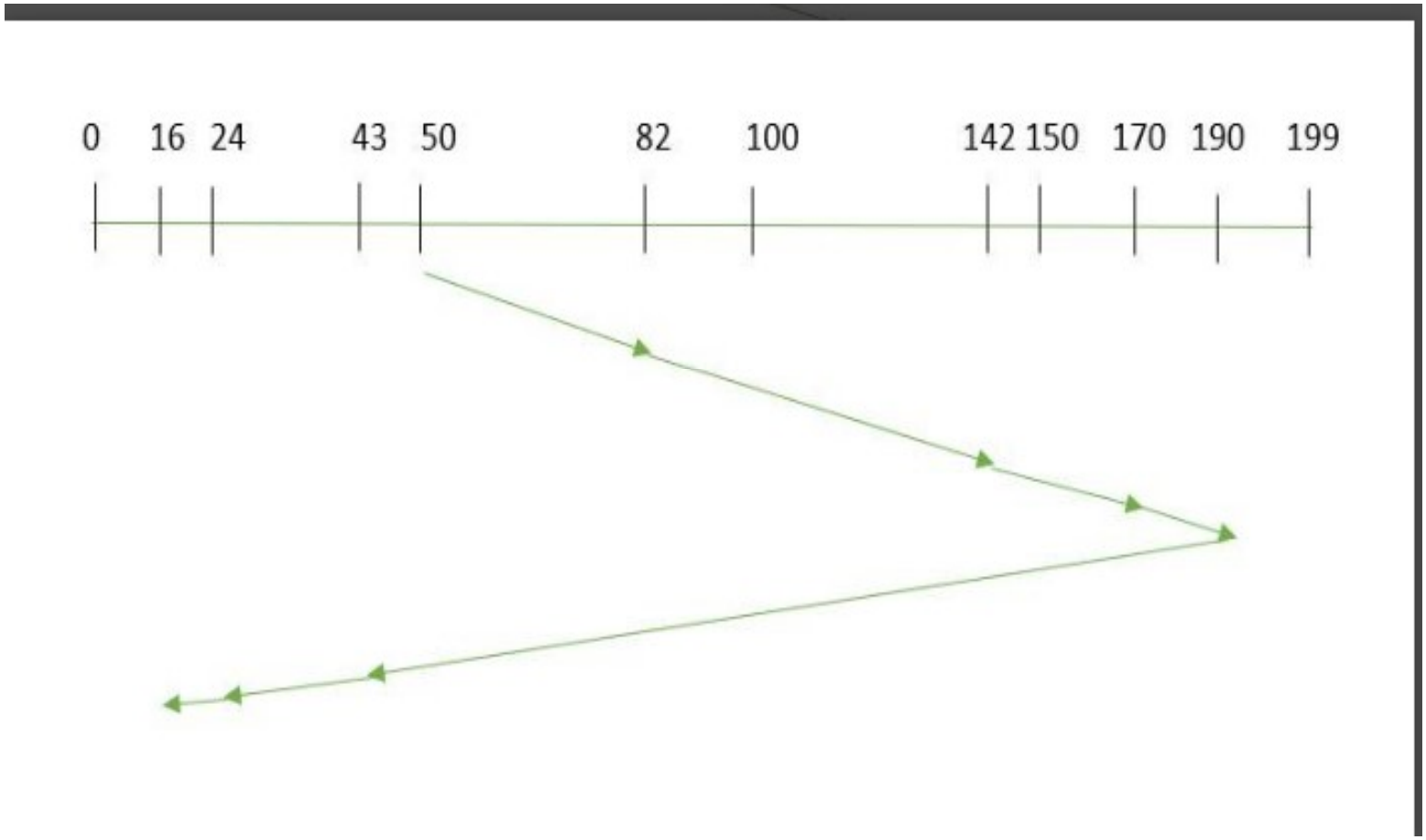
Here are some of the advantages of C-SCAN.

- Provides more uniform wait time compared to SCAN.

5. LOOK

- LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.
- Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



5. LOOK

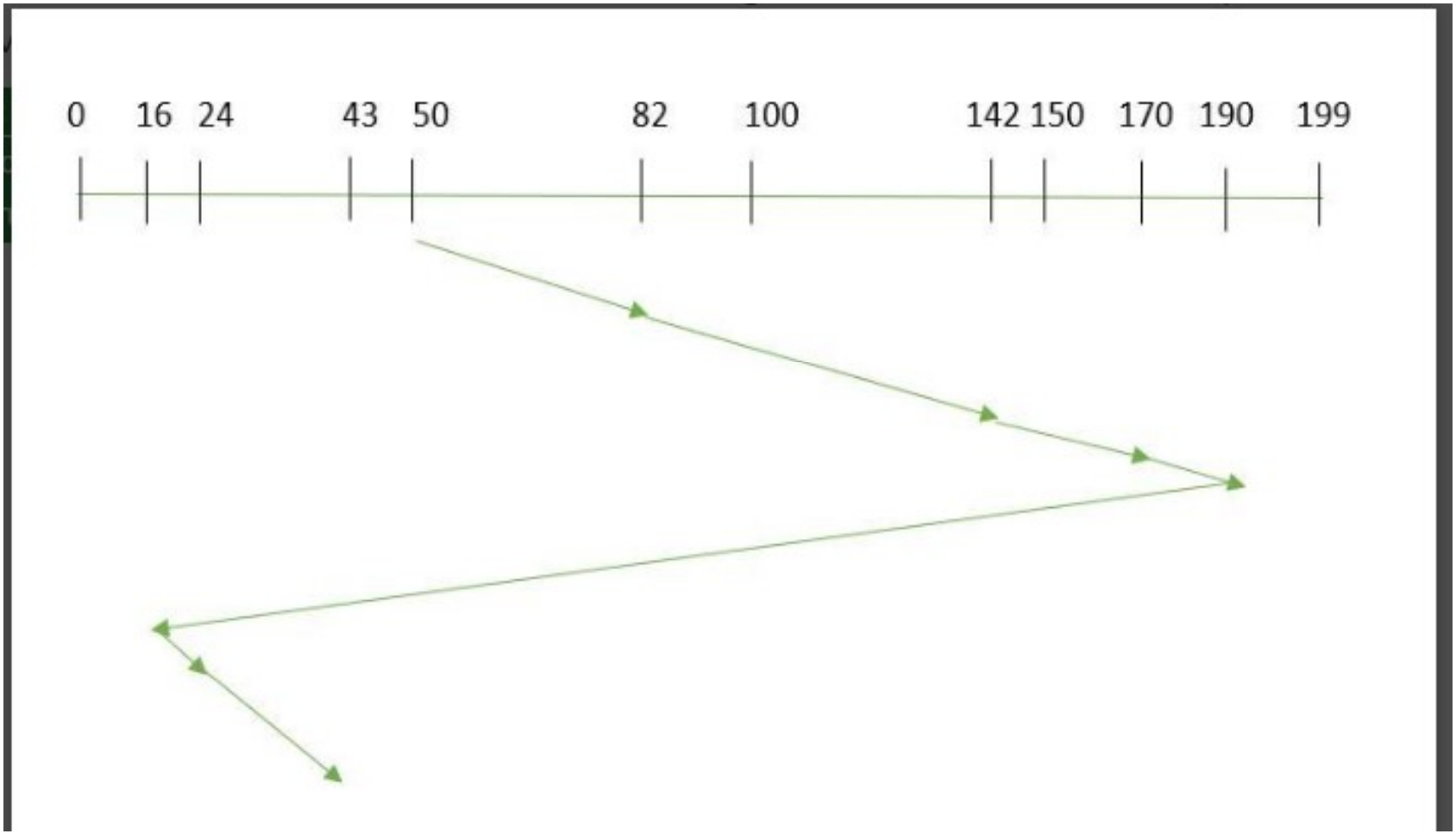
- So, the total overhead movement (total distance covered by the disk arm) is calculated as:= (190-50) + (190-16) = 314

6. C-LOOK

- As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm.
- In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request.
- Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”**



6. C-LOOK

- So, the total overhead movement (total distance covered by the disk arm) is calculated as = (190-50) + (190-16) + (43-16) = 341

Note

7. RSS (Random Scheduling)

It stands for Random Scheduling and just like its name it is natural. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm sits perfectly. Which is why it is usually used for analysis and simulation.

8. LIFO (Last-In First-Out)

In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is newest or last entered is serviced first, and then the rest in the same order.

Advantages of LIFO (Last-In First-Out)

Here are some of the advantages of the Last In First Out Algorithm.

Maximizes locality and resource utilization

Can seem a little unfair to other requests and if new requests keep coming in, it cause starvation to the old and existing ones.

Note

9. N-STEP SCAN

It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N request and this way all get requests to get a guaranteed service

Advantages of N-STEP SCAN

Here are some of the advantages of the N-Step Algorithm.

It eliminates the starvation of requests completely

10. F-SCAN

This algorithm uses two sub-queues. During the scan, all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

Advantages of F-SCAN

Here are some of the advantages of the F-SCAN Algorithm.

F-SCAN along with N-Step-SCAN prevents “arm stickiness” (phenomena in I/O scheduling where the scheduling algorithm continues to service requests at or near the current sector and thus prevents any seeking)

Each algorithm is unique in its own way. Overall Performance depends on the number and type of requests.

Note: Average Rotational latency is generally taken as 1/2(Rotational latency).

Disk Management

- Disk management is the process of organizing and maintaining the storage on a computer's hard disk.
- It involves dividing the hard disk into **partitions, formatting** these partitions to different file systems, and regularly maintaining and optimizing disk performance.
- The goal of disk management is to provide a convenient and organized storage system for users to store and access their data, as well as to ensure that the computer runs smoothly and efficiently.

Disk Management tasks

- Disk management tasks include **partitioning** the hard disk, **formatting partitions** to different file systems, and maintaining and optimizing disk performance through tasks such as **defragmentation** and **backup**.
- These tasks help ensure a convenient and organized storage system for users and improve the overall performance and stability of the computer.

Disk Management Tools

- Disk management tools are software programs that allow you to manage the storage space on your hard disk. Common disk management tools include **Windows Disk Management, Disk Utility** on Mac OS, and **GParted** on Linux. These tools allow you to easily manage the storage space on your hard disk without needing to use command-line tools.
- **Built-in disk management tools in Windows and macOS** –
- **Windows** has the built-in **Disk Management tool,** which allows you to perform tasks such as creating, resizing, and deleting partitions, formatting partitions, and defragmenting and optimizing disk performance. Similarly, **MacOS** has the **Disk Utility,** which provides similar functionality for managing storage on Mac systems.
- **Third-party disk management software** –
- Third-party disk management software refers to software programs developed by companies other than the operating system provider, that provide additional or advanced functionality for managing the storage space on your hard disk. Examples of third-party disk management software include **EaseUS Partition Master, AOMEI Partition Assistant, and Paragon Partition Manager.**

Swap space management

- **Swapping** is a memory management technique used in multi-programming to increase the number of processes sharing the CPU.
- It is a technique of removing a process from the main memory and storing it into secondary memory, and then bringing it back into the main memory for continued execution.
- This action of moving a process out from main memory to secondary memory is called **Swap Out** and the action of moving a process out from secondary memory to main memory is called **Swap In**.

Swap space management

- **Swap-Space :**
  The area on the disk where the swapped-out processes are stored is called swap space.
- **Swap-Space Management :**
  Swap-Space management is another low-level task of the operating system. Disk space is used as an extension of main memory by the virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decreases system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput.

Swap space management

- A computer has a sufficient amount of physical memory, but we need more, so we swap some memory on disk most of the time.
- ***Swap space*** is a space on a hard disk that is a substitute for physical memory. It is used as virtual memory, which contains process memory images. Whenever our computer runs short of physical memory, it uses its virtual memory and stores information in memory on a disk.
- This interchange of data between virtual memory and real memory is called ***swapping*** and space on disk as swap space. Swap space helps the computer's operating system pretend that it has more RAM than it actually has. It is also called a ***swap file***

What is Swap-Space Management?

- Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system.
- Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use. For example, systems that implement swapping may use swap space to hold an entire process image, including the code and data segments. ***Paging*** systems may simply store pages that have been pushed out of the main memory. The amount of swap space needed on a system can vary depending on the amount of physical memory, the amount of virtual memory it is backing, and how it is used. It can range from a few ***megabytes*** of disk space to ***gigabytes***.

Note

- Note that it may be safer to overestimate than to underestimate the amount of swap space required because if a system runs out of swap space, it may be forced to abort processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it does no other harm.
- Some systems recommend the amount to be set aside for swap space. Solaris, for example, suggests setting swap space equal to the amount by which virtual memory exceeds page-able physical memory. Previously, Linux suggested setting swap space to double the amount of physical memory, although most Linux systems now use considerably less swap space. There is currently much debate in the Linux community about whether to set aside swap space at all.
- Some operating systems, including Linux, allow multiple swap spaces. These swap spaces are usually put on separate disks so the load placed on the I/O system by paging and swapping can be spread over the system's I/O devices.

Uses of Swap Space

- The different operating system uses Swap-space in various ways. The systems that are implementing swapping may use swap space to hold the entire process, including image, code, and data segments.
- ***Swapping*** is a memory management technique used in multi-programming to increase the number of processes sharing the CPU. It is a technique of removing a process from the main memory, storing it into secondary memory, and then bringing it back into the main memory for continued execution. This action of moving a process out from main memory to secondary memory is called ***Swap Out***. The action of moving a process out from secondary memory to main memory is called ***Swap In***.
- ***Paging*** systems may simply store pages that have been pushed out of the main memory. The need for swap space on a system can vary from megabytes to gigabytes. Still, it also depends on the amount of physical memory, the virtual memory it is backing, and how it uses the virtual memory.
- It is safer to overestimate than to underestimate the amount of swap space required because if a system runs out of swap space, it may be forced to abort the processes or may crash entirely. Overestimation wastes disk space that could be used for files, but it does not harm others.