Module 4

FILE SYSTEM INTERFACE, MASS-STORAGE STRUCTURE

What is File System?

- A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes.
- It is a method of data collection that is used as a medium for giving input and receiving output from that program.
- In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user.
- Every File has a logical location where they are located for storage and retrieval.

File attributes

- A file has a name and data. Moreover, it also stores meta information like file creation date and time, current size, last modified date, etc. All this information is called the attributes of a file system.
- 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.
- Timestamps 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

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 a directory.
- Opening a file: Rather than have all file operations specify a file name, causing the operating system to evaluate the name, check access permissions, and so on, all operations except create and delete require a file open() first. If successful, the open call returns a file handle that is used as an argument in the other calls.
- Writing a file: To write a file, we make a system call specifying both the open file handle and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place if it is sequential. 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 file handle and where (in memory) the next block of the file should be put. Again, the system needs to keep a read pointer to the location in the file where the next read is to take place, if sequential. 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-positio pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity.

- Repositioning within a file: The current-file-position pointer of the open file is repositioned to a given value. Repositioning within a file need not involve any actual I/O. 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 or mark as free the directory entry. Note that some systems allow hard links—multiple names (directory entries) for the same file. In this case the actual file contents is not deleted until the last link is deleted.
- 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. The file can then be reset to length zero, and its file space can be released.

Commonly used terms in File systems

- **Field:**

This element stores a single value, which can be static or variable length.

- **DATABASE:**

Collection of related data is called a database. Relationships among elements of data are explicit.

- **FILES:**

Files is the collection of similar record which is treated as a single entity.

- **RECORD:**

A Record type is a complex data type that allows the programmer to create a new data type with the desired column structure. Its groups one or more columns to form a new data type. These columns will have their own names and data type.

File Type

- It refers to the ability of the operating system to differentiate various types of files like text files, binary, and source files. However, Operating systems like MS_DOS and UNIX has the following type of files:

Character Special File

- It is a hardware file that reads or writes data character by character, like mouse, printer, and more.

Ordinary files

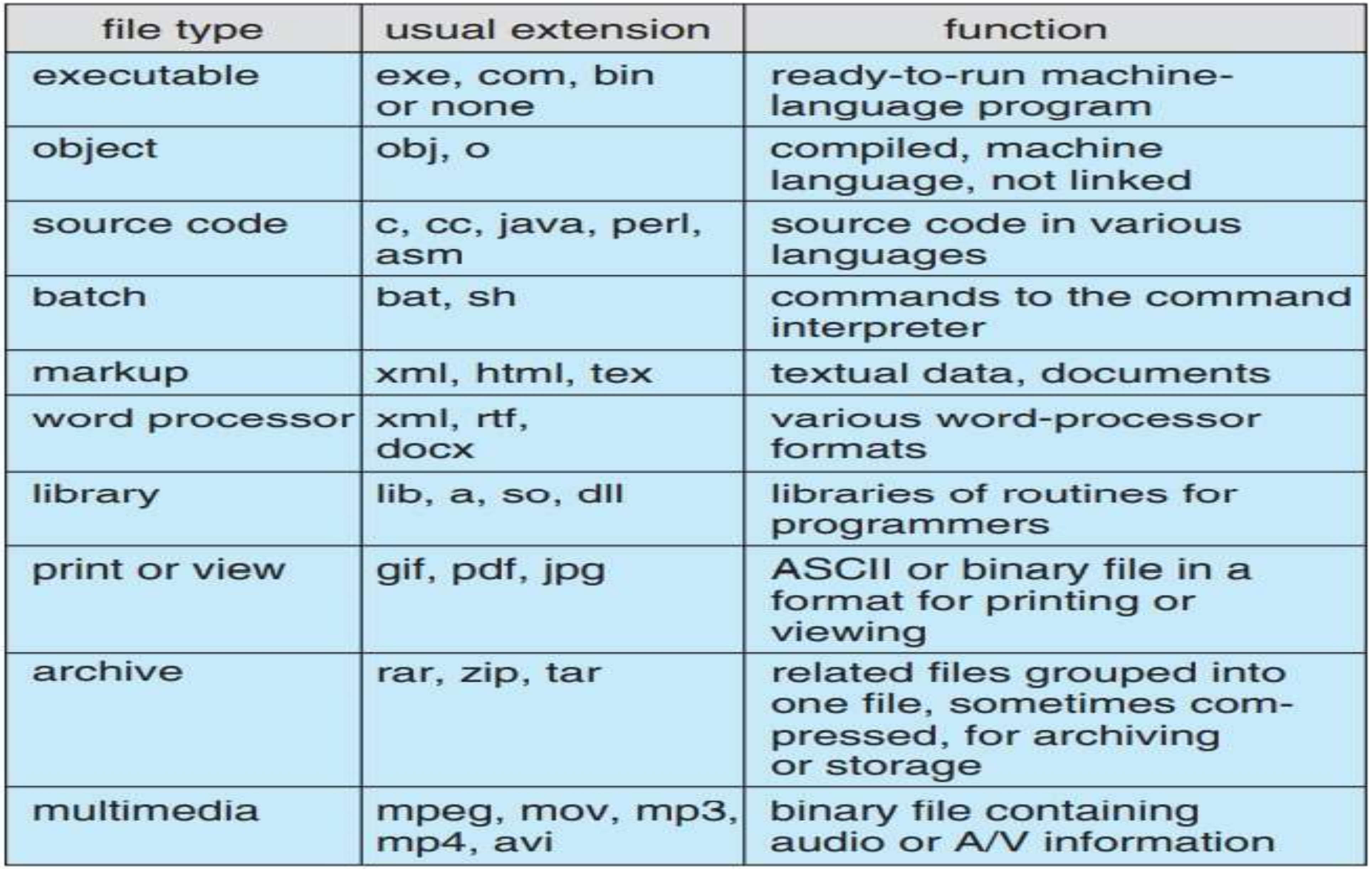
- These types of files stores user information.
- It may be text, executable programs, and databases.
- It allows the user to perform operations like add, delete, and modify.

Directory Files

- Directory contains files and other related information about those files. Its basically a folder to hold and organize multiple files.

Special Files

- These files are also called device files. It represents physical devices like printers, disks, networks, flash drive, etc.



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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Table: Common File Type

Access Methods

- File access is a process that determines the way that files are accessed and read into memory. Generally, a single access method is always supported by operating systems. Though there are some operating system which also supports multiple access methods.
- **Three file access methods are:**
- Sequential access
- Direct random access
- Indexed sequential access

Sequential Access Method

- A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other.
- This access method is the most primitive one. Example: 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, which is depicted in figure is based on a tape model of a file and works as well on sequential-access devices as it does on random-access ones.

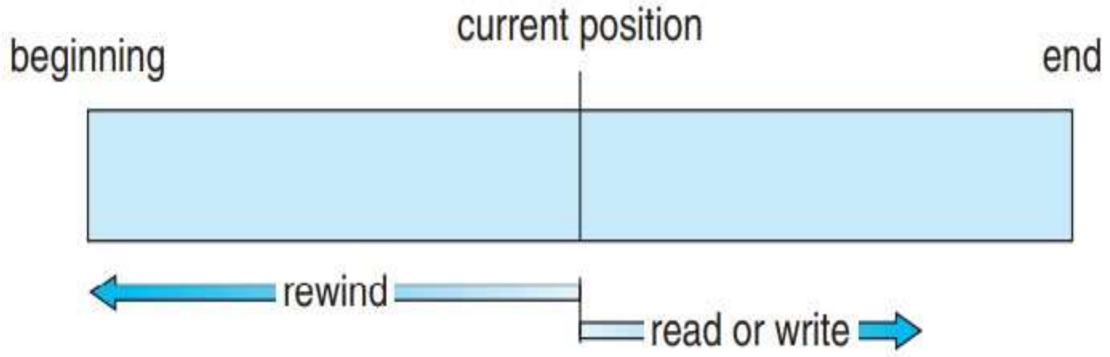


Fig: Sequential access file

Random Access

- The random access method is also called direct random access.
- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

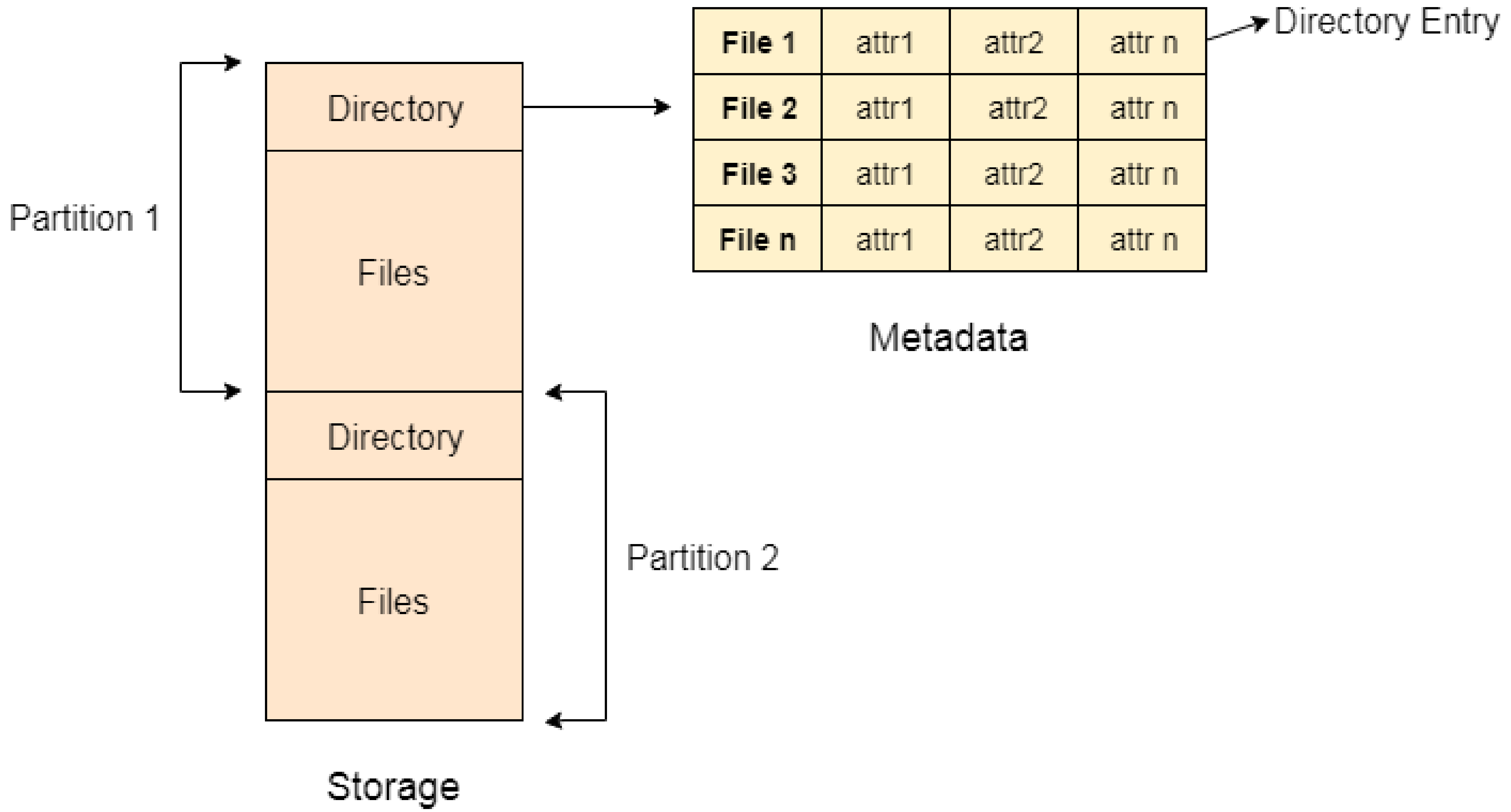
1. Read n: reading record n
2. Write n: writing record n
3. Jump to record n
4. Query current record: used to return back to this record later

Indexed sequential access

- This type of accessing method is based on simple sequential access.
- In this access method, an index is built for every file, with a direct pointer to different memory blocks.
- This method, the Index is searched sequentially, and its pointer can access the file directly.
- Multiple levels of indexing can be used to offer greater efficiency in access. It also reduces the time needed to access a single record.

Structure of directory

- Directory can be defined as the listing of the related files on the disk.
- The directory may store some or the entire file attributes.
- To get the benefit of different file systems on the different operating systems.
- A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.
- Each partition must have at least one directory in which, all the files of the partition can be listed.
- A directory entry is maintained for each file in the directory which stores all the information related to that file.


Directory Entry
File 1
attr1
attr2
attr n
File 2
File 3
File n
Metadata
Partition 1
Directory
Files
Directory
Files
Partition 2
Storage

Every Directory supports a number of common operations on the file:

1.File Creation

2.Search for the file

3.File deletion

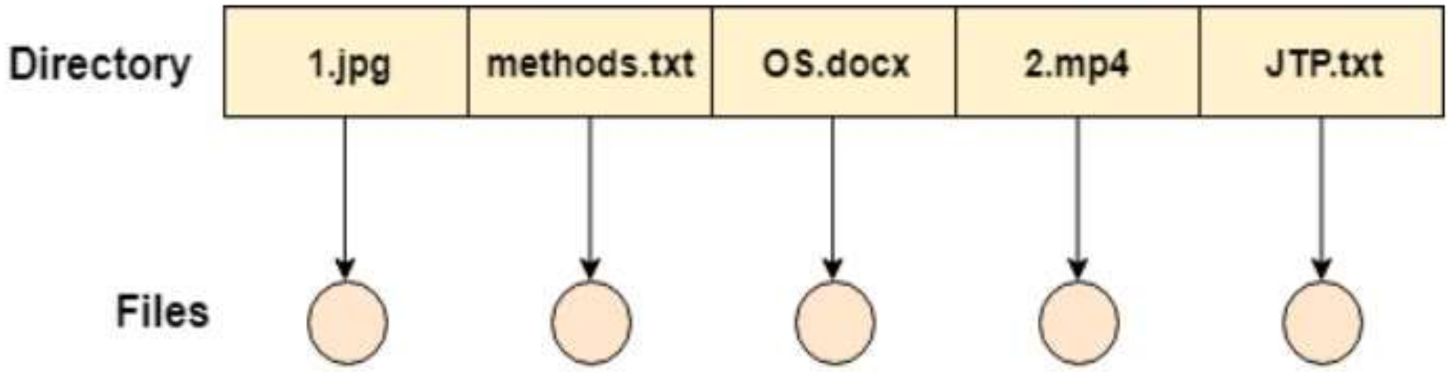
4.Renaming the file

5.Traversing Files

6.Listing of files

Single level Directory

- The simplest method is to have one big list of all the files on the disk.
- The entire system will contain only one directory which is supposed to mention all the files present in the file system.
- The directory contains one entry per each file present on the file system.
- This type of directories can be used for a simple system.



Single Level Directory

Advantages

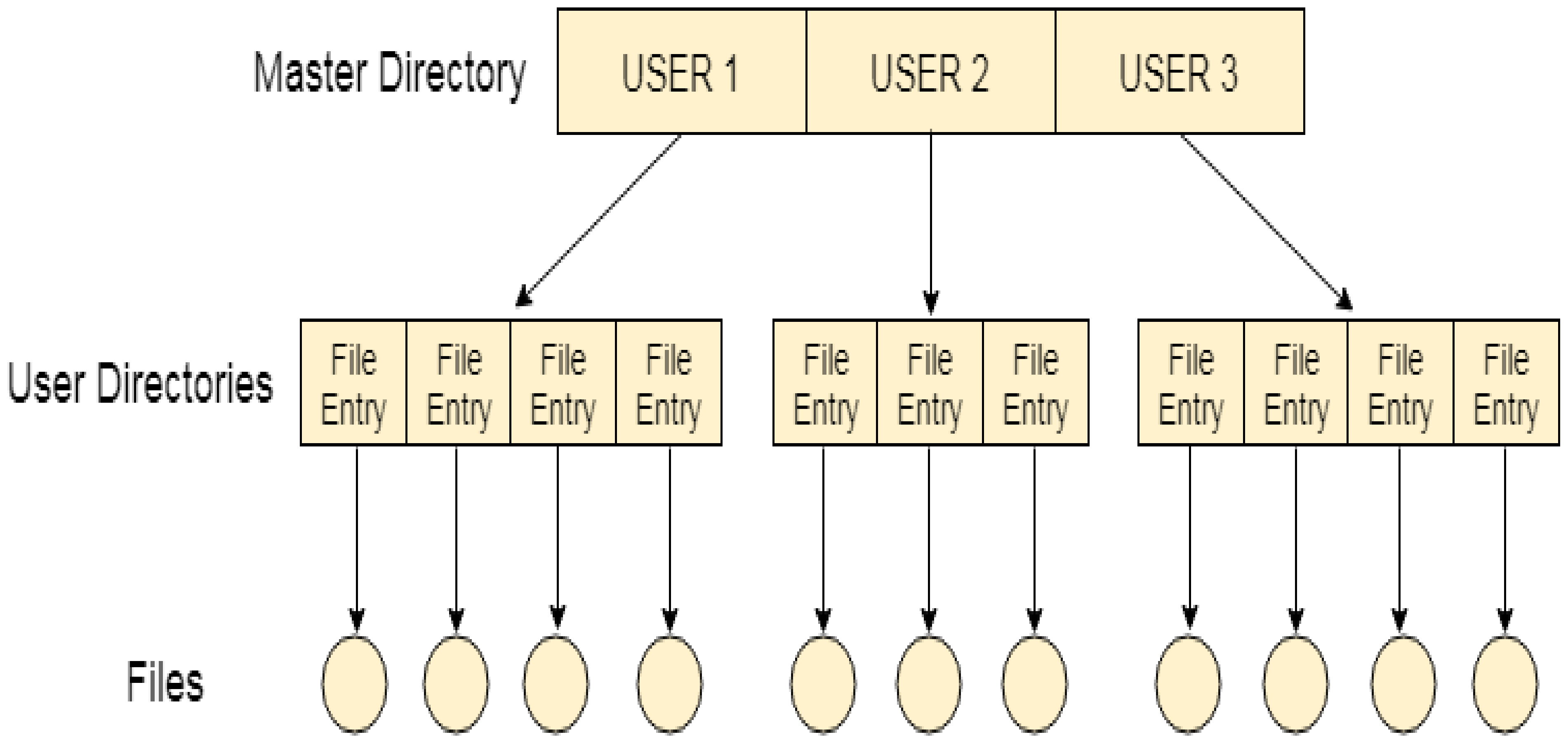
1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.
5. Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

Two Level Directory

- In two level directory systems, we can create a separate directory for each user.
- There is one master directory which contains separate directories dedicated to each user.
- For each user, there is a different directory present at the second level, containing group of user's file.
- The system doesn't let a user to enter in the other user's directory without permission.



Two Level Directory

Characteristics of two level directory system

1.Each files has a path name as ***/User-name/directory-name/***

2.Different users can have the same file name.

3.Searching becomes more efficient as only one user's list needs to be traversed.

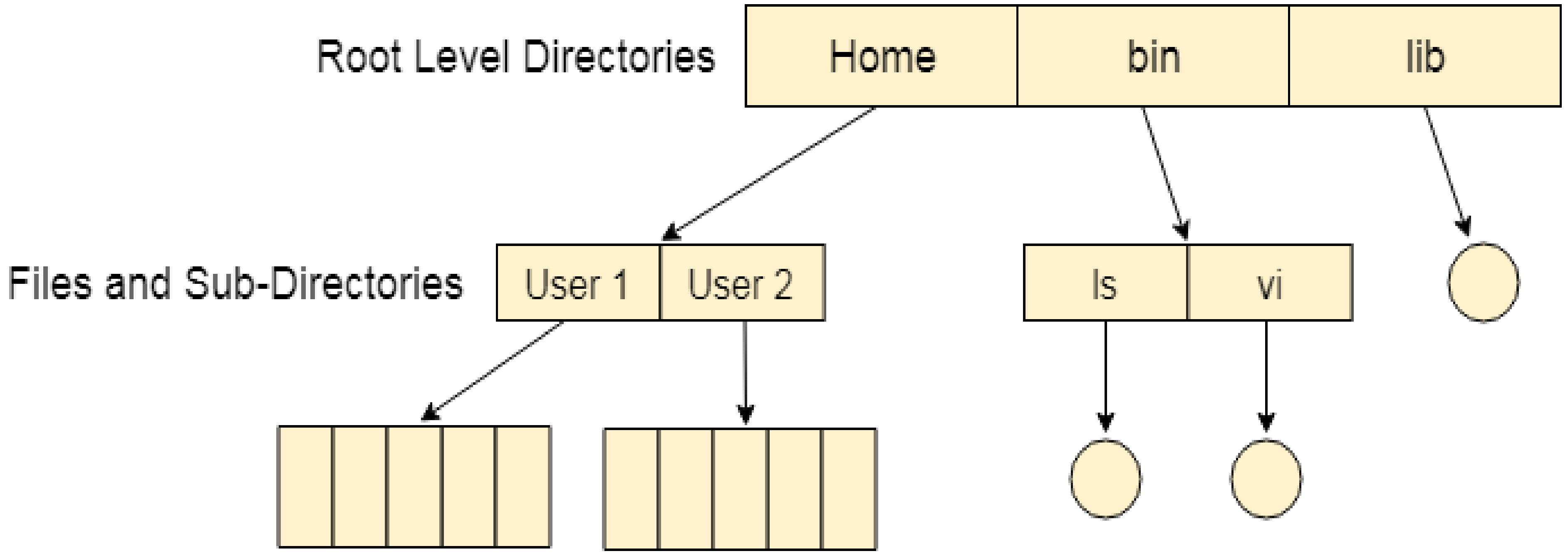
4.The same kind of files cannot be grouped into a single directory for a particular user.

- Every Operating System maintains a variable as **PWD** which contains the present directory name (present user name) so that the searching can be done appropriately.

Tree Structured Directory

- In Tree structured directory system, any directory entry can either be a file or sub directory.
- Tree structured directory system overcomes the drawbacks of two level directory system.
- The similar kind of files can now be grouped in one directory.
- Each user has its own directory and it cannot enter in the other user's directory. However, the user has the permission to read the root's data but he cannot write or modify this.
- Only administrator of the system has the complete access of root directory.
- Searching is more efficient in this directory structure. The concept of current working directory is used.

- A file can be accessed by two types of path, either relative or absolute.
- Absolute path is the path of the file with respect to the root directory of the system while relative path is the path with respect to the current working directory of the system.
- In tree structured directory systems, the user is given the privilege to create the files as well as directories.



The Structured Directory System

Permissions on the file and directory

- A tree structured directory system may consist of various levels therefore there is a set of permissions assigned to each file and directory.
- The permissions are **R W X** which are regarding reading, writing and the execution of the files or directory.
- The permissions are assigned to three types of users: owner, group and others.
- There is a identification bit which differentiate between directory and file. For a directory, it is **d** and for a file, it is dot **(.)**

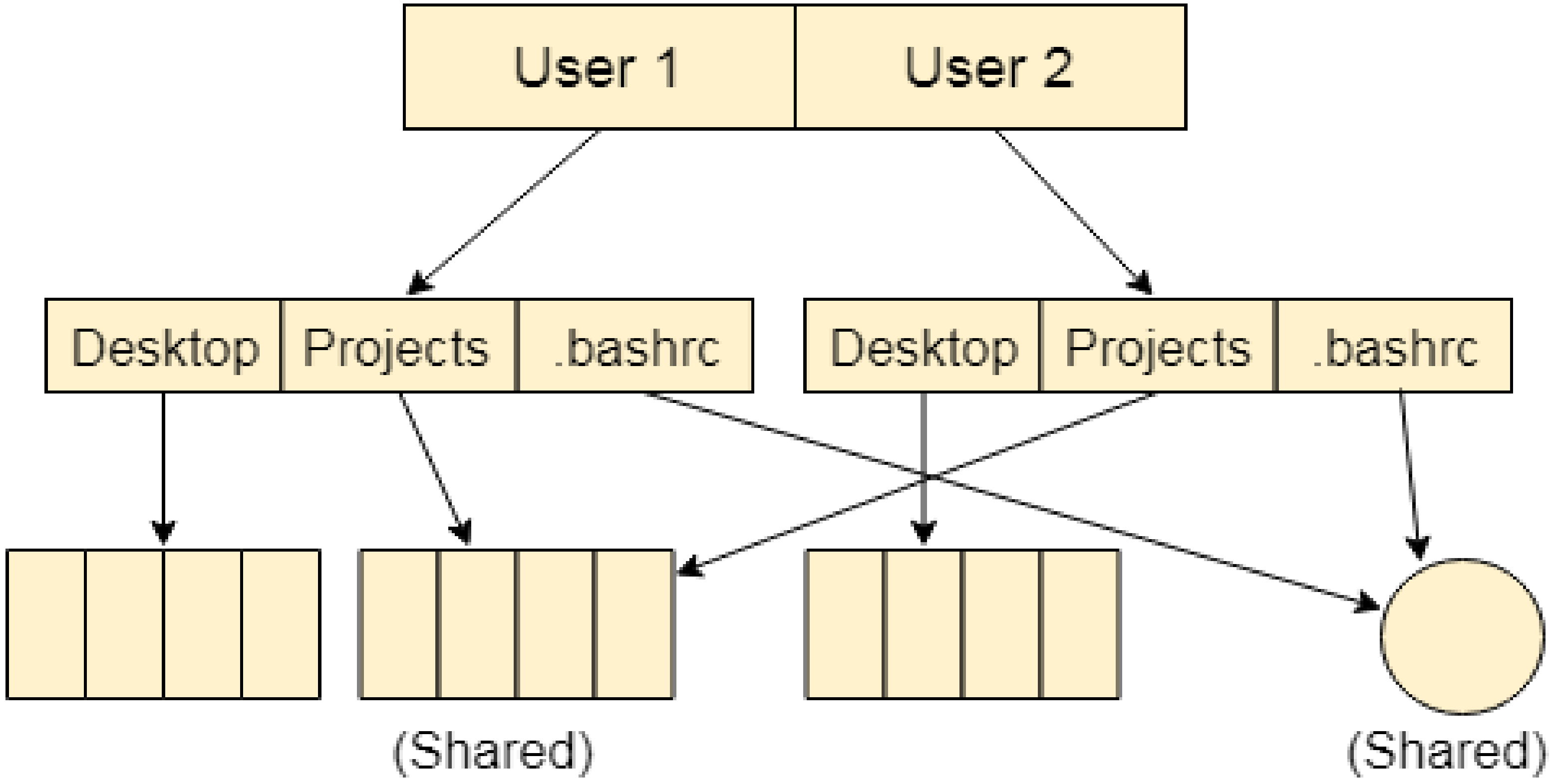
Acyclic-Graph Structured Directories

- The tree structured directory system doesn't allow the same file to exist in multiple directories therefore sharing is major concern in tree structured directory system.
- We can provide sharing by making the directory an acyclic graph.
- In this system, two or more directory entry can point to the same file or sub directory. That file or sub directory is shared between the two directory entries.
- These kinds of directory graphs can be made using links or aliases.
- We can have multiple paths for a same file. Links can either be symbolic (logical) or hard link (physical).

- If a file gets deleted in acyclic graph structured directory system, then

1. In the case of soft link, the file just gets deleted and we are left with a dangling pointer.

2. In the case of hard link, the actual file will be deleted only if all the references to it gets deleted.



Acyclic-Graph Structured Directory System

General Graph Directory

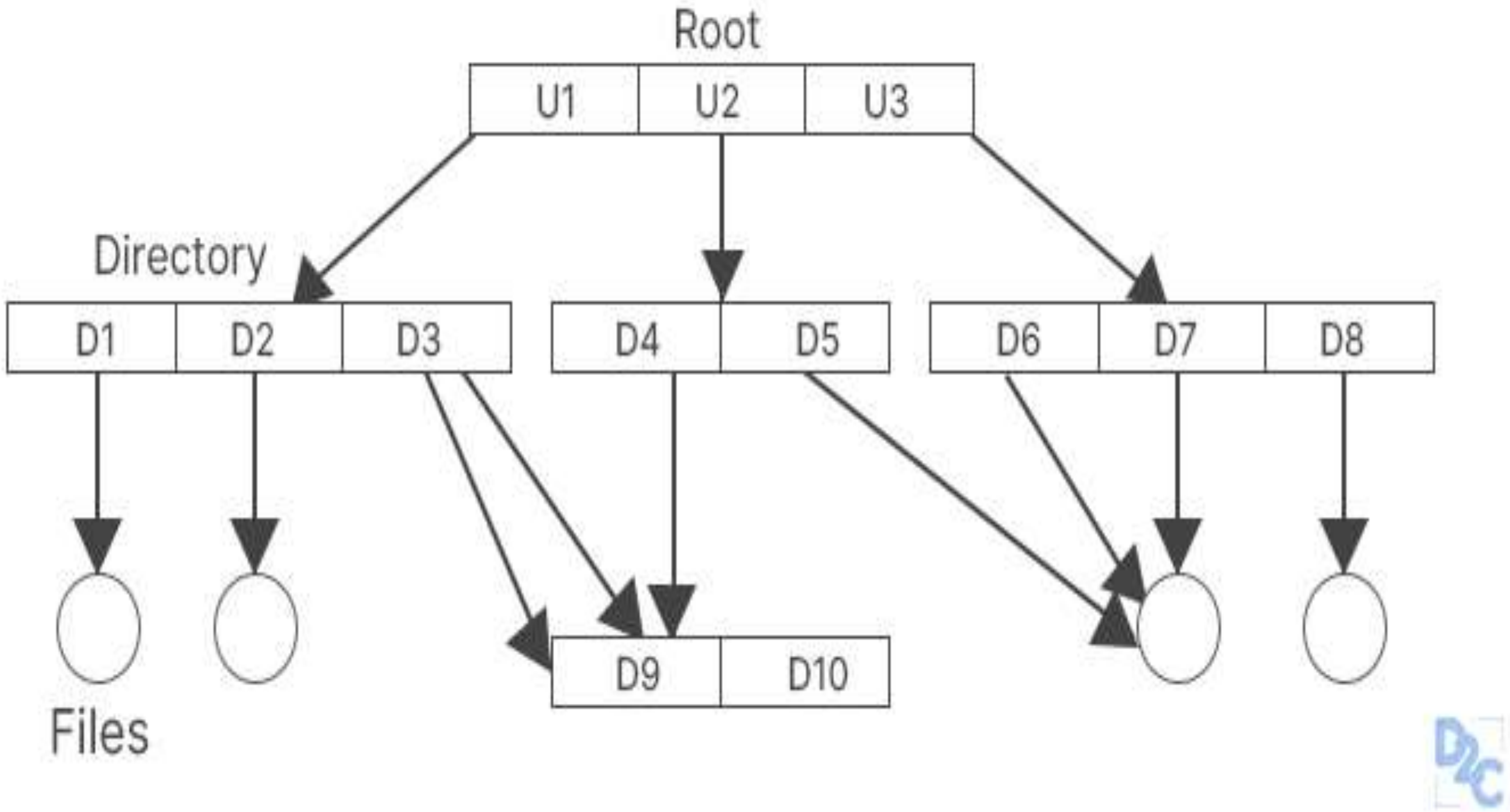
- Cycles are allowed inside a directory structure where numerous directories can be derived from more than one parent directory in a general graph directory structure.
- When general graph directories are allowed, commands like search a directory and its subdirectories for something must be used with caution.
- If cycles are allowed, the search is infinite.
- The biggest issue with this type of directory layout is figuring out how much space the files and folders have used up.

Advantages of General Graph Directory

- The General-Graph directory structure is more adaptable than the others.
- In the general-graph directory, cycles are permitted.

Disadvantages of General Graph Directory

- It is more expensive than other options.
- Garbage collection is required.

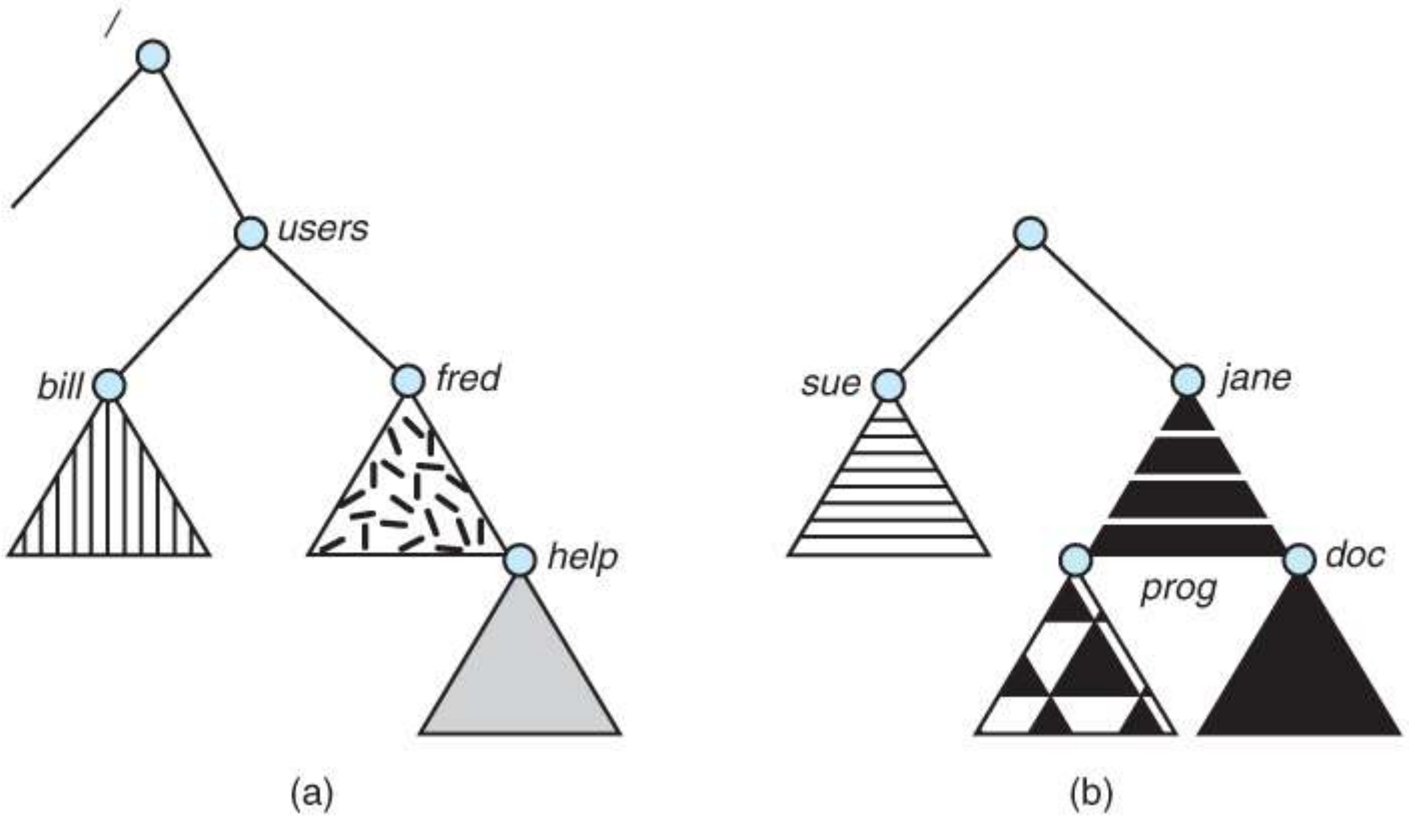

Root
U1
U2
U3
Directory
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
Files

File System Mounting

- Mounting refers to the grouping of files in a file system structure accessible to the user of the group of users.
- It can be local or remote, in the local mounting, it connects disk drivers as one machine, while in the remote mounting it uses Network File System (NFS) to connect to directories on other machines so that they can be used as if they are the part of the user's file system.
- The directory structure can be built out of multiple volumes which are supposed to be mounted to make them available within the file-system namespace.
- The procedure for mounting is simple, the OS is given the name of the device and the location within the file structure where the file system is attached.

- For example, in a UNIX system, there is a single directory tree, and all the accessible storage must have a location in the single directory tree.
- Mounting is used to make the storage accessible.
- A file system containing the user’s home directories might be mounted as /home, and they can be accessed by using directory names with time like /home/janc.
- Similarly, if the file system is mounted as /user, then we will use /user/janc to access it.
- Then the operating system verifies if the device contains a valid file system by asking the device driver to read the directory and verify that the directory has the expected format.
- Then the operating system finally notes down the directory structure that the file system is mounted at the specified mount point.

- This method helps the operating system traverse through the directory structure and switch among file systems as appropriate.
- A system may either allow the same file system to be mounted repeatedly on different mount points or it may allow one mount per file system.
- For example, the Macintosh operating system. In this whenever the system encounters a disk for the first time, it searches for the file system in the disk, and if it finds one it automatically mounts the system at the root level and adds a folder icon on the screen labelled as the name of the file system. The Microsoft OS maintains a two-level directory structure.


/
users
bill
fred
help
(a)
sue
jane
prog
doc
(b)

File sharing

1. MULTIPLE USERS:

When an operating system accommodates multiple users, the issues of file sharing, file naming and file protection become preeminent.

ü The system either can allow user to access the file of other users by default, or it may require that a user specifically grant access to the files.

ü These are the issues of **access control and protection.**

ü To implementing sharing and protection, the system must maintain more file and directory attributes than a on a single-user system.

ü **The owner** is the user who may change attributes, grand access, and has the most control over the file or directory.

- **The group attribute** of a file is used to define a subset of users who may share access to the file.
- Most systems implement **owner attributes** by managing a list of **user names and associated user identifiers** (user Ids).
- When a user logs in to the system, the authentication stage determines the appropriate user ID for user. That user ID is associated with all of user’s processes and threads.
- When they need to be user readable, they are translated, back to the user name via the user name list.
- Likewise, group functionality can be implemented as a system wide list of **group names and group identifiers**.
- Every user can be in one or more groups, depending upon operating system design The user’s **group Ids** is also included in every associated process and thread.

Remote File System:

- Networks allowed communications between **remote computers**.
- Networking allows the **sharing or resource spread** within a campus or even around the world.
- User manually transfer files between machines via programs like ftp.
- A distributed file system (DFS) in which remote directories is visible from the local machine.
- The World Wide Web: A browser is needed to gain access to the remote file and separate operations (essentially a wrapper for ftp) are used to transfer files.

a. The client-server Model:

- Remote file systems allow a computer to a mount one or more file systems from one or more remote machines.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client –server facility.
- Client identification is more difficult.
- Clients can be specified by their network name or other identifier, such as IP address, but these can be spoofed (or imitate).
- An unauthorized client can spoof the server into deciding that it is authorized, and the unauthorized client could be allowed access.

b) Distributed Information systems:

- Distributed information systems, also known as distributed naming service, have been devised to provide a unified access to the information needed for remote computing.
- Domain name system (DNS) provides **host-name-to-network address** translations for their entire Internet (including the World Wide Web).
- Before DNS was invented and became widespread, files containing the same information were sent via e-mail of ftp between all networked hosts.

c) Failure Modes:

- **Redundant arrays of inexpensive disks (RAID)** can prevent the loss of a disk from resulting in the loss of data.
- RAID is a way of storing the same data in different places on multiple hard disks or solid-state drives (SSDs) to protect data in the case of a drive failure.
- Remote file system has more failure modes.
- Remote file systems add new failure modes, due to network failure , server failure.
- Recovery from failure can involve state information about **status of each remote request.**
- Stateless protocols such as NFS v3 include all information in each request, **allowing easy recovery but less security.**

d) Consistency Semantics:

- It is characterization of the system that specifies the semantics of multiple users accessing a shared file simultaneously.
- These semantics should specify **when modifications of data by one user are observable by other users**.
- The semantics are typically implemented as code with the file system.
- A series of file accesses (that is reads and writes) attempted by a user to the same file is always enclosed between the open and close operations.
- The series of access between the open and close operations is a file session.

(i) UNIX Semantics:

The UNIX file system uses the following consistency semantics:

1. Writes to an open file by a user are visible immediately to other users that have this file open at the same time.
2. One mode of sharing allows users to share the pointer of current location into the file. Thus, the advancing of the pointer by one user affects all sharing users.

(ii) Session Semantics:

The Andrew file system (AFS) uses the following consistency semantics:

1. Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously.
2. Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect this change.

(iii) Immutable –shared File Semantics:

- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has two key properties:
- Its name may **not be reused** and its contents may **not be altered**.

Protection in File System

- Users want to protect the information stored in the file system from **improper access** and **physical damage**.
- To protect our information, one can make duplicate copies of the files, some systems automatically copy the files to protect the user from losing important information if the original files are accidentally destroyed.
- There can be situations of hardware errors like extreme temperature, vandalism, head crashes, failures, etc. that may cause damage to the files.
- To understand protection better we will need to understand the types of access.

Types of access:

- **File sharing** is the method of providing partial or full access to the users of the file system, as multiple users have access to the same data, there is a need for protection.
- One way to protect file systems can be to prohibit access, but it is an extreme scenario and is not of practical use.
- What is needed is **controlled access**. It may depend on various factors.
- These are the following operations that can be controlled:

1. Delete
2. Append
3. Execute
4. Read
5. Write
6. List

- However, protection is provided only at a lower level. For example, copying a file may be implemented by simply providing a sequence of **read requests**. In this scenario, a user with read access can copy the file and print it, and so on.

Access Control

- There are numerous ways to access any file, one of the prominent ones is to associate identity-dependent access with all files and directories.
- A list is created called the access-control list which enlists the names of users and the type of access granted to them.
- However, it is very long as all the users need to be listed down. This process can often be tedious and unrewarding, especially if one does not have the list of users before the task.
- To resolve this situation and condense the length of access-control list, the following classifications are used:
- **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.

Other ways of protection:

- Another approach is to use passwords to enable access to the file systems.
- However, this method has certain disadvantages:

1. If one password is used for all the files, then in a situation where the password happens to be known by the other users, all the files will be accessible.
2. It can be difficult to remember a lengthy and large number of passwords.

File System Structure

- The file system provides the users efficient access to the disk by allowing convenient storage, location, and retrieval of data.
- The Operating Systems use the layering method for every task including file systems, and every layer is responsible for some activities.
- The layers are as follows:

1. Application program
2. Logical File System
3. File Organization Module
4. Basic File System
5. I\O Control
6. Devices

Application of each layer:

1. **The application program** requests for a file, the request is sent to the logical file system.

2. **The logical file system** stores the meta data of the file and the directory structure. If the application program does not have the required permissions, then the logical file system shows an error. It is also responsible for verifying the path to the file.

3. Files are stored in the hard disks from where it is to be retrieved. The files are divided into logical blocks. To store and retrieve data from the file, the logical blocks need to be mapped to the physical blocks. This mapping is done by **the file organization module**. It also looks over space management. The file organization module decided which physical block is to be allocated to the applications.

4. Once the decision is made, it passes the information to **the basic file system**. The basic file system issues a command to the I\O Control to fetch the blocks.

5. **The I\O control** handles any interrupts and contains the device drivers to access the hard disk.

File System Implementation

- We can use the following data structures to implement file systems:

1. On-disk structures

2. In-memory structures

On-disk structures

- They store information about the total number of disk blocks, their location, free disk blocks, etc. Different on-disk structures are as follows:

Directory structure

- Directory structure stores the file names and associated inode numbers.

Volume Control Block

- Volume Control Block stores information about a particular partition. Examples: block pointers, free block count, etc.

Boot Control Block

- Boot Control Block is generally the first block of volume and it stores the information needed to boot an operating system. It is also known as partition boot sector in NTFS and boot block in UNIX.

- **Per-File FCB**
- Per-File FCB stores details about files and has a unique identifier number to associate with the directory entry.

In-memory structures

The in-memory structure is maintained in the main memory and is helpful for file management for caching. Different in-memory structures are:

Mount Table

- Mount Table stores information about the mounted volume.

Per-process open-file table

- Per-process open-file table stores information opened by the process and it maps with the system open-file.

Directory-structure cache

- Directory-structure cache stores information about the recently accessed directories.

System-wide open-file table

- The system-wide open-file table stores the copy of the FCB of each open file.

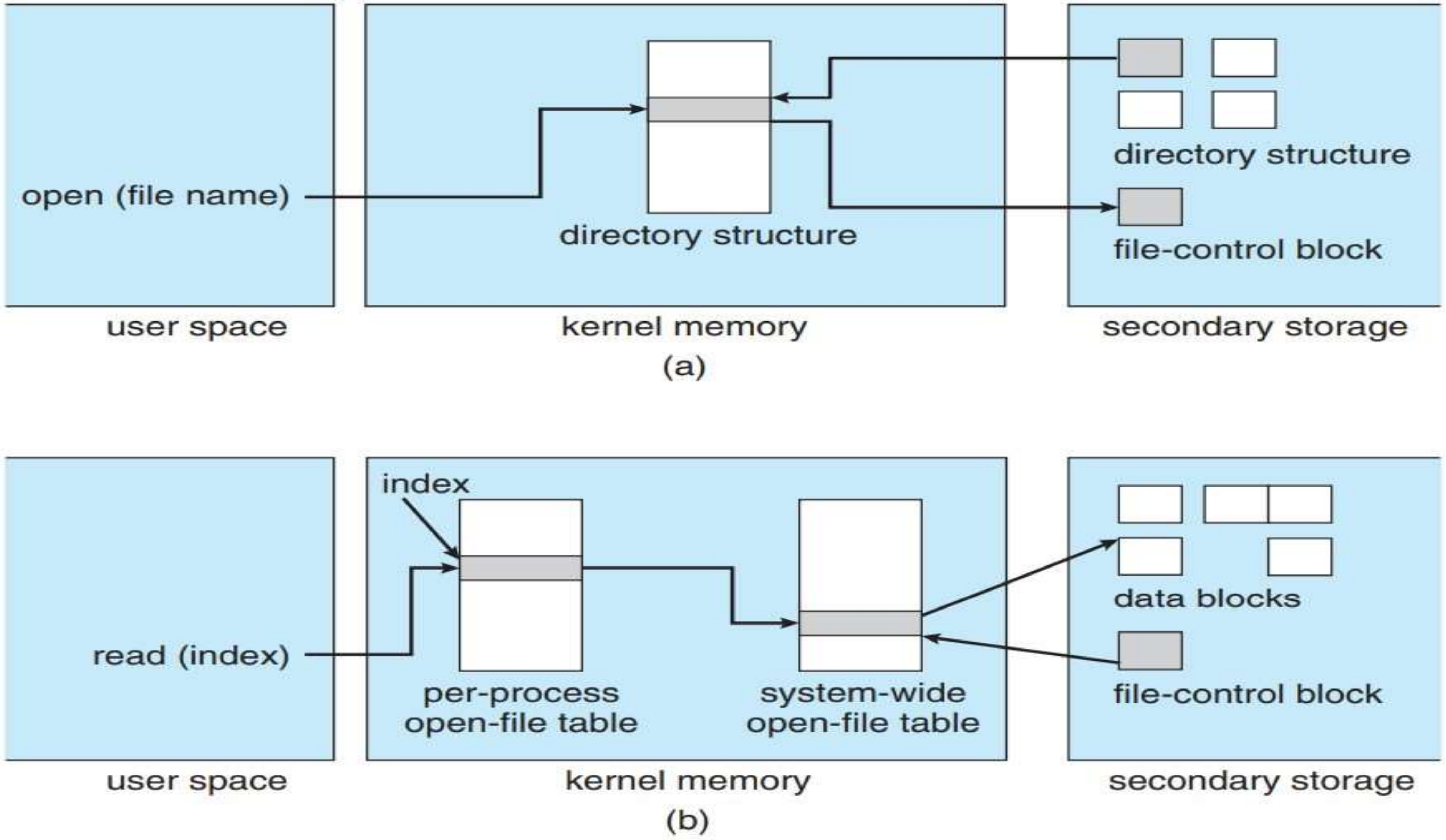


Figure 14.3 In-memory file-system structures. (a) File open. (b) File read.

Directory implementation

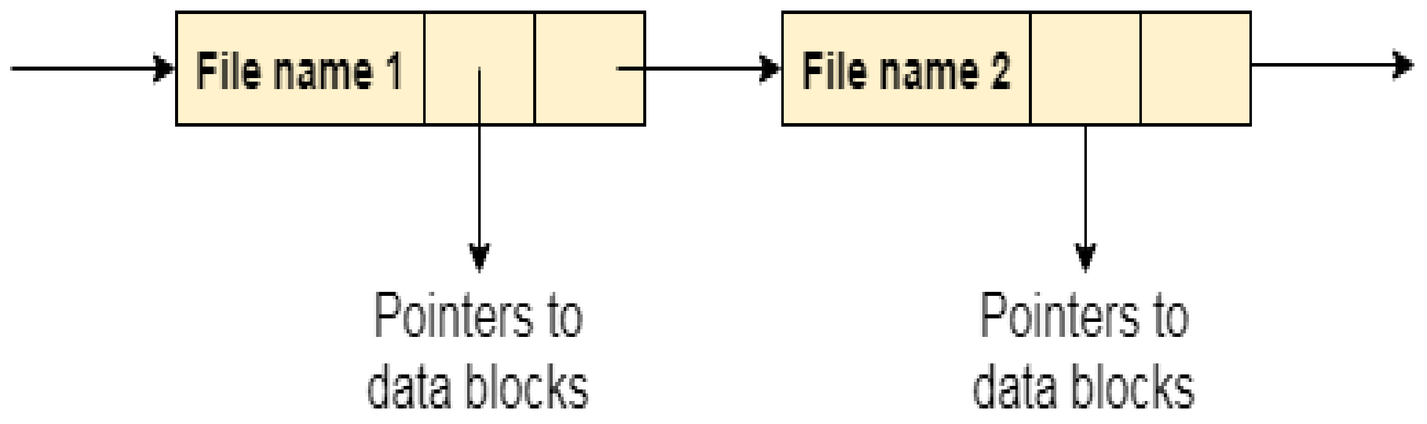
- There is the number of algorithms by using which, the directories can be implemented.
- However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.
- The directory implementation algorithms are classified according to the data structure they are using.
- There are mainly two algorithms which are used in these days.

Linear List

- In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics:

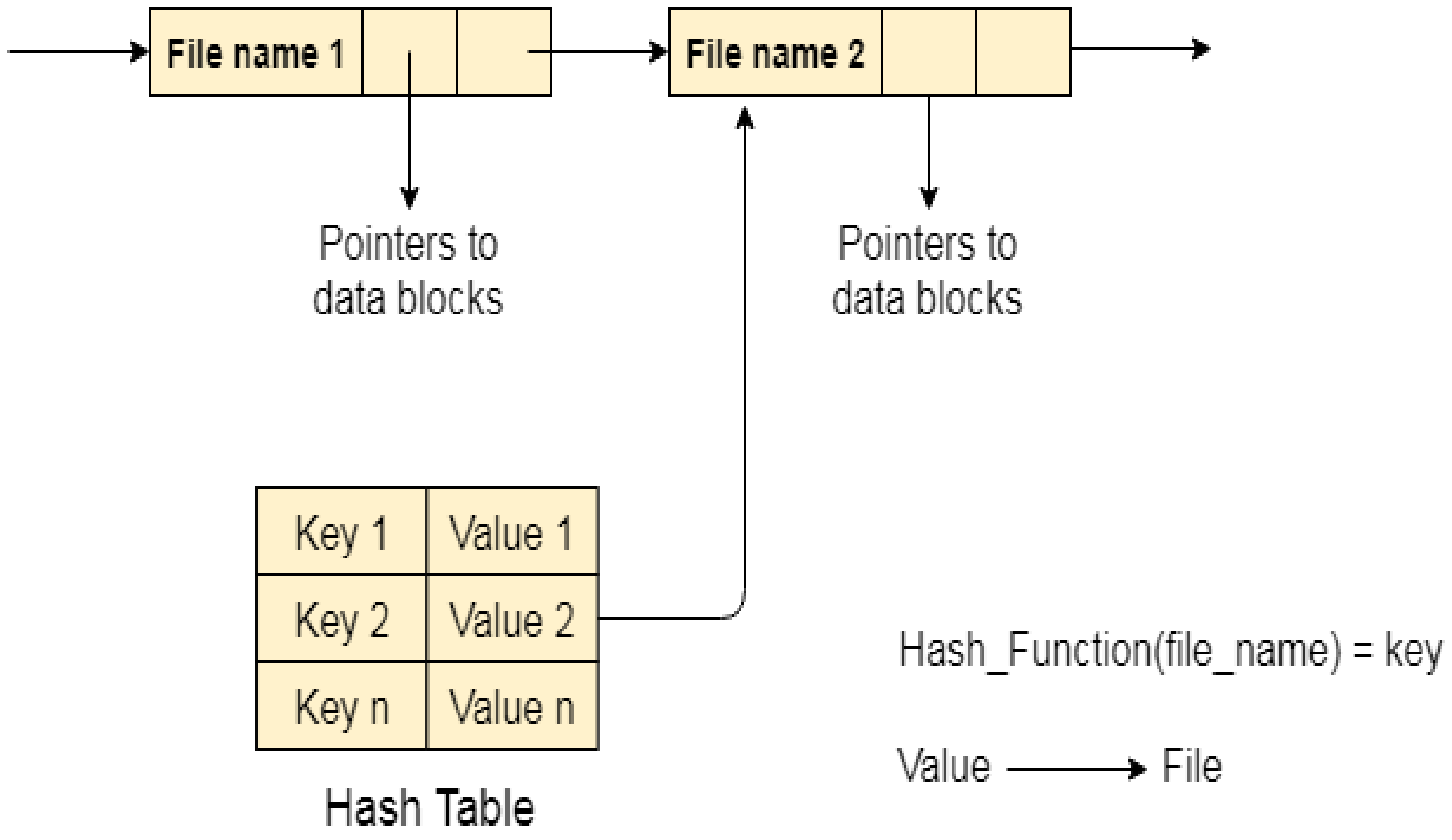
1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.


File name 1
File name 2
Pointers to data blocks
Pointers to data blocks

Linear List

Hash Table

- To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.
- A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.
- Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.


File name 1
File name 2
Pointers to data blocks
Pointers to data blocks
Key 1
Value 1
Key 2
Value 2
Key n
Value n
Hash Table
Hash_Function(file_name) = key
Value
File

File Allocation Methods

- The direct-access nature of secondary storage gives us flexibility in the implementation of files.
- In almost every case, many files are stored on the same device.
- The main problem is how to allocate space to these files so that storage space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating secondary storage space are in wide use:

➢Contiguous Allocation

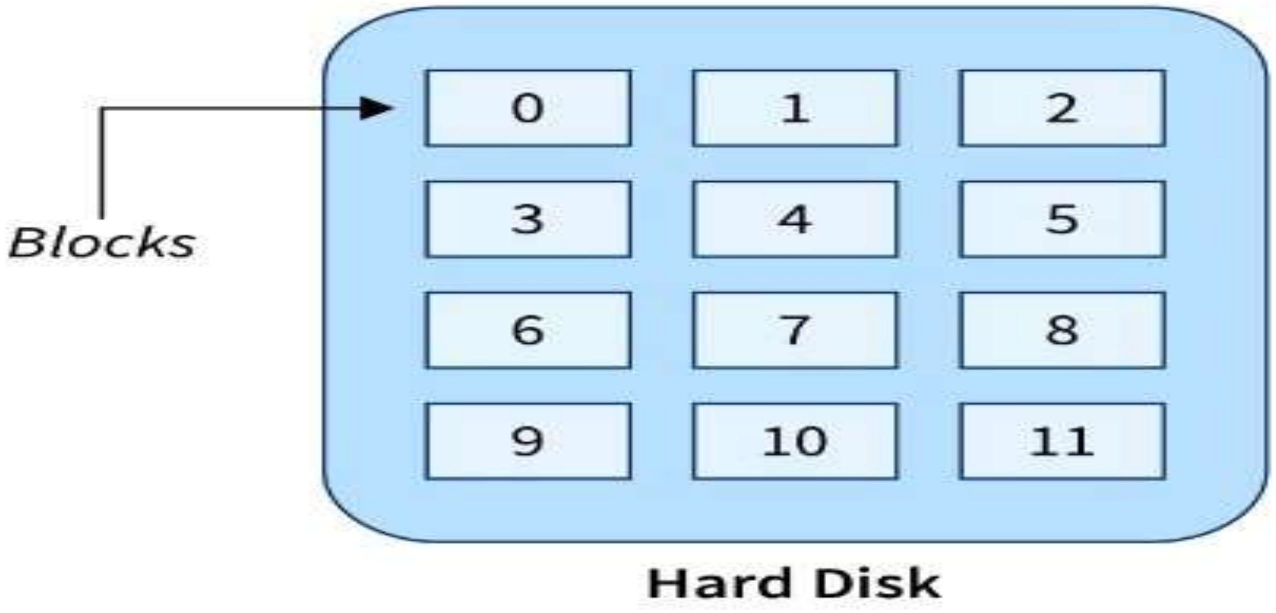
➢Linked Allocation

➢Indexed Allocation

Contiguous Allocation

- In contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are adjacent.
- Assuming a file needs 'n' number of blocks in the disk and the file begins with a block at position 'x', the next blocks to be assigned to it will be x+1,x+2,x+3,...,x+n-1 so that they are in a contiguous manner.
- Let's understand this diagrammatically.
- **Example**
- We have three different types of files that are stored in a contiguous manner on the hard disk.

Contiguous Allocation



File Name	Start	Length	Allocated Blocks
file1.txt	0	4	0,1,2,3
sun.jpg	5	3	5,6,7
mov.mp4	9	3	9,10,11

Directory

- In the above image on the left side, we have a memory diagram where we can see the blocks of memory.
- At first, we have a text file named file1.txt which is allocated using contiguous memory allocation, it starts with the memory block 0 and has a length of 4 so it takes the 4 contiguous blocks 0,1,2,3.
- Similarly, we have an image file and video file named sun.jpg and mov.mp4 respectively, which you can see in the directory that they are stored in the contiguous blocks. 5,6,7 and 9,10,11 respectively.
- Here the directory has the entry of each file where it stores the address of the starting block and the required space in terms of the block of memory.

Advantages

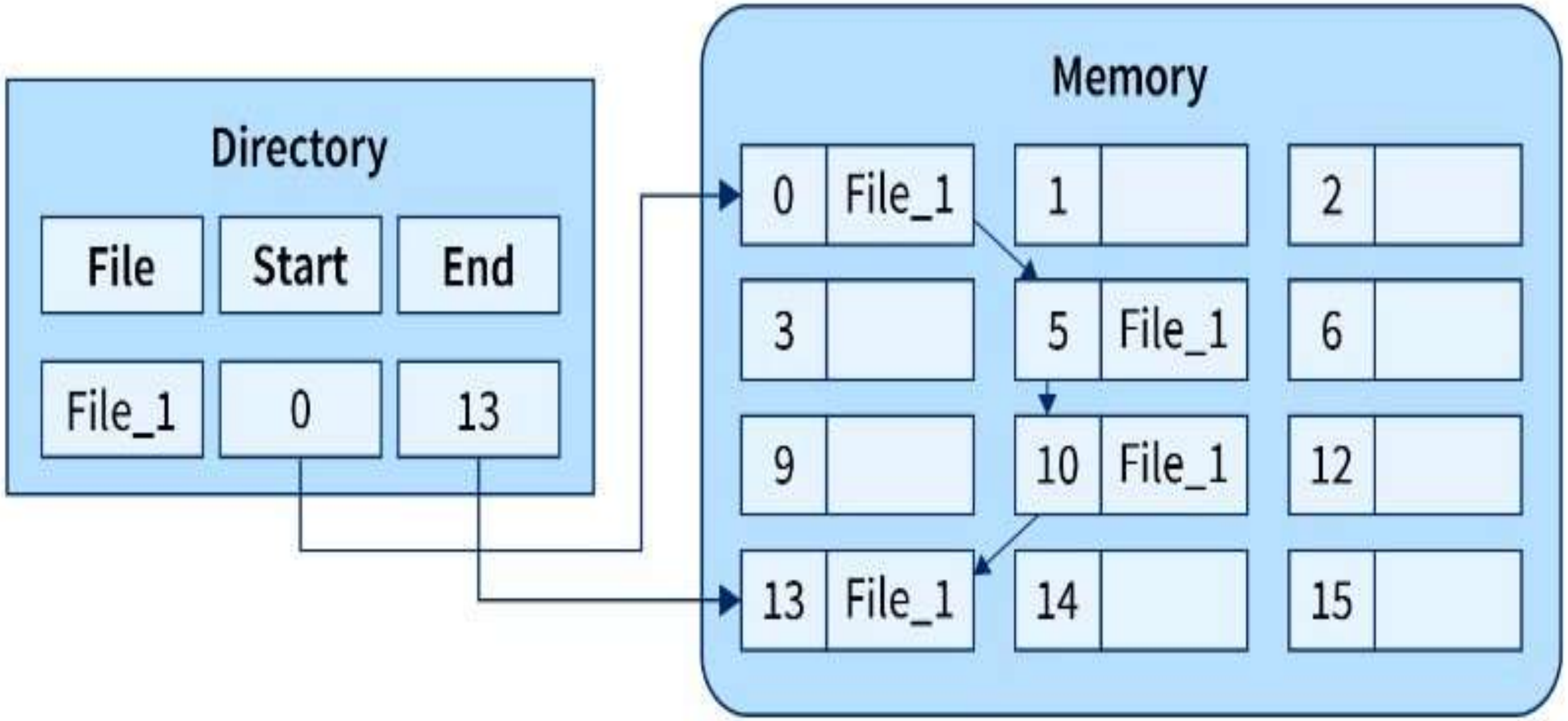
- It is very easy to implement.
- There is a minimum amount of seek time.
- The disk head movement is minimum.
- Memory access is faster.
- It supports sequential as well as direct access.

Disadvantages

- At the time of creation, the file size must be initialized.
- As it is pre-initialized, the size cannot increase. As
- Due to its constrained allocation, it is possible that the disk would fragment internally or externally.

Linked File Allocation

- The Linked file allocation overcomes the drawback of contiguous file allocation.
- Here the file which we store on the hard disk is stored in a scattered manner according to the space available on the hard disk.
- Now, you must be thinking about how the OS remembers that all the scattered blocks belong to the same file.
- So as the name linked File Allocation suggests, the pointers are used to point to the next block of the same file, therefore along with the entry of each file each block also stores the pointer to the next block.
- Let's understand this better diagrammatically by taking an example.
- **Example**
- Here we have one file which is stored using Linked File Allocation.



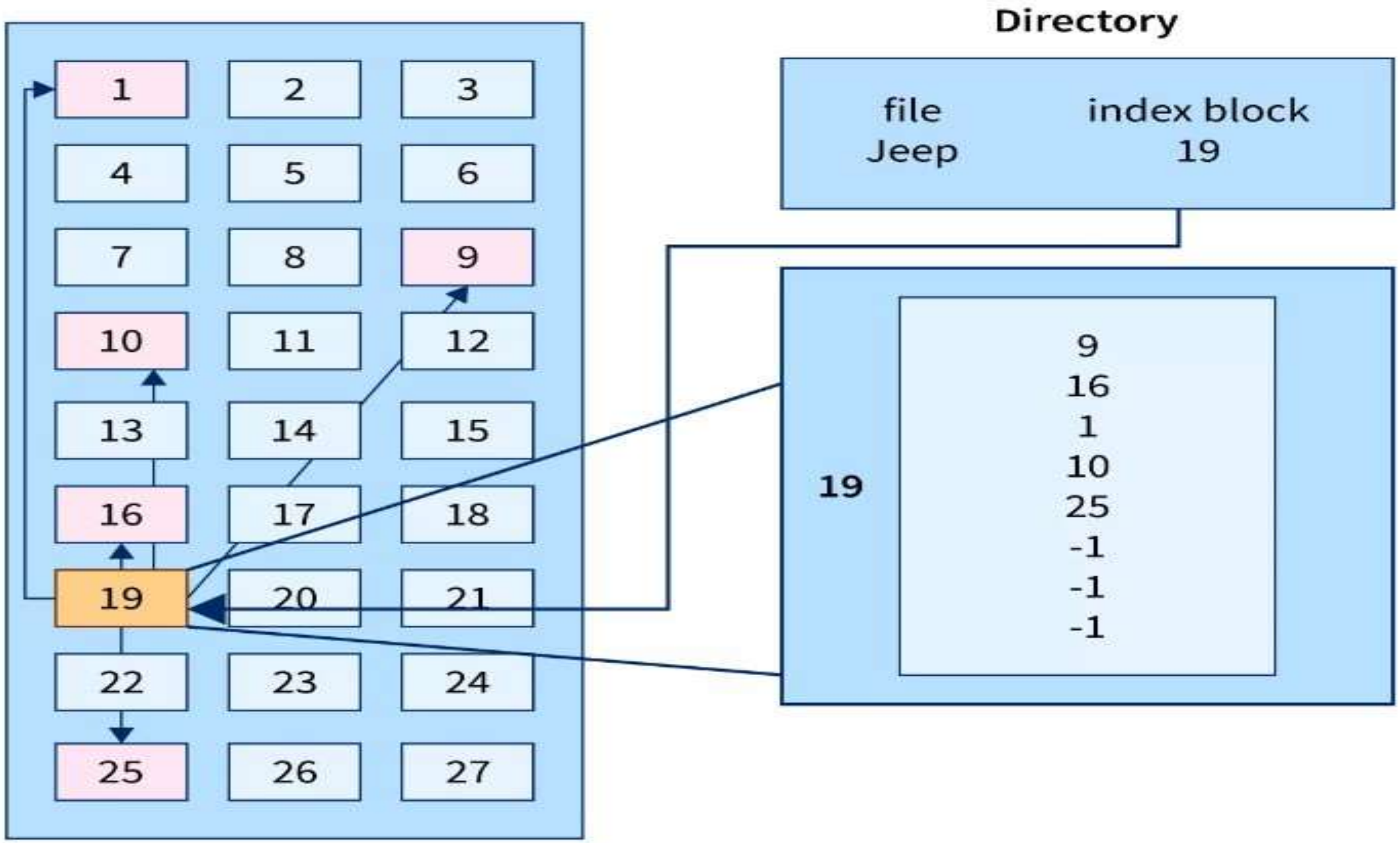
Directory
File
Start
End
File_1
0
13
Memory
0
File_1
1
2
3
5
File_1
6
9
10
File_1
12
13
File_1
14
15

- In the above image on the right, we have a memory diagram where we can see memory blocks.
- On the left side, we have a directory where we have the information like the address of the first memory block and the last memory block.
- In this allocation, the starting block given is 0 and the ending block is 15, therefore the OS searches the empty blocks between 0 and 15 and stores the files in available blocks, but along with that it also stores the pointer to the next block in the present block.
- Hence it requires some extra space to store that link.

- **Advantages**
- There is no external fragmentation.
- The directory entry just needs the address of starting block.
- The memory is not needed in contiguous form, it is more flexible than contiguous file allocation.
- **Disadvantages**
- It does not support random access or direct access.
- If pointers are affected so the disk blocks are also affected.
- Extra space is required for pointers in the block.

Indexed File Allocation.

- The indexed file allocation is somewhat similar to linked file allocation as indexed file allocation also uses pointers but the difference is here all the pointers are put together into one location which is called **index block**.
- That means we will get all the locations of blocks in one index file.
- The blocks and pointers were spread over the memory in the Linked Allocation method, where retrieval was accomplished by visiting each block sequentially.
- But here in indexed allocation, it becomes easier with the index block to retrieve.
- Let's take an example to explain this better.
- **Example**
- As shown in the diagram below block 19 is the index block which contains all the addresses of the file named **text1**. In order, the first storage block is 9, followed by 16, 1, then 10, and 25. The negative number -1 here denotes the empty index block list as the file text1 is still too small to fill more blocks.



Directory
file
index block
Jeep
19
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
19
9
16
1
10
25
-1
-1
-1

Advantages

- It reduces the possibilities of external fragmentation.
- Rather than accessing sequentially it has direct access to the block.

Disadvantages

- Here more pointer overhead is there.
- If we lose the index block we cannot access the complete file.
- It becomes heavy for the small files.
- It is possible that a single index block cannot keep all the pointers for some large files.

- To resolve this issue, we can use the following approaches:

1.Linked scheme

2.Multilevel Index

3.Combined Scheme

1. Linked Scheme

If the file is big then more blocks are required so one index block is insufficient to store all the pointers, therefore to store the pointers two or more index blocks are used where these index boxes are connected using linked file allocation that is each index block stores the pointer to the next index block.

2. Multilevel Index

- In this method, the multiple indexes blocks along with the levels of these blocks.
- Here, the level 1 block is used for pointing to the level 2 block which points to the blocks occupied by the file.
- These index blocks can be extended to three or more levels according to the size of the file.

3. Combined Scheme

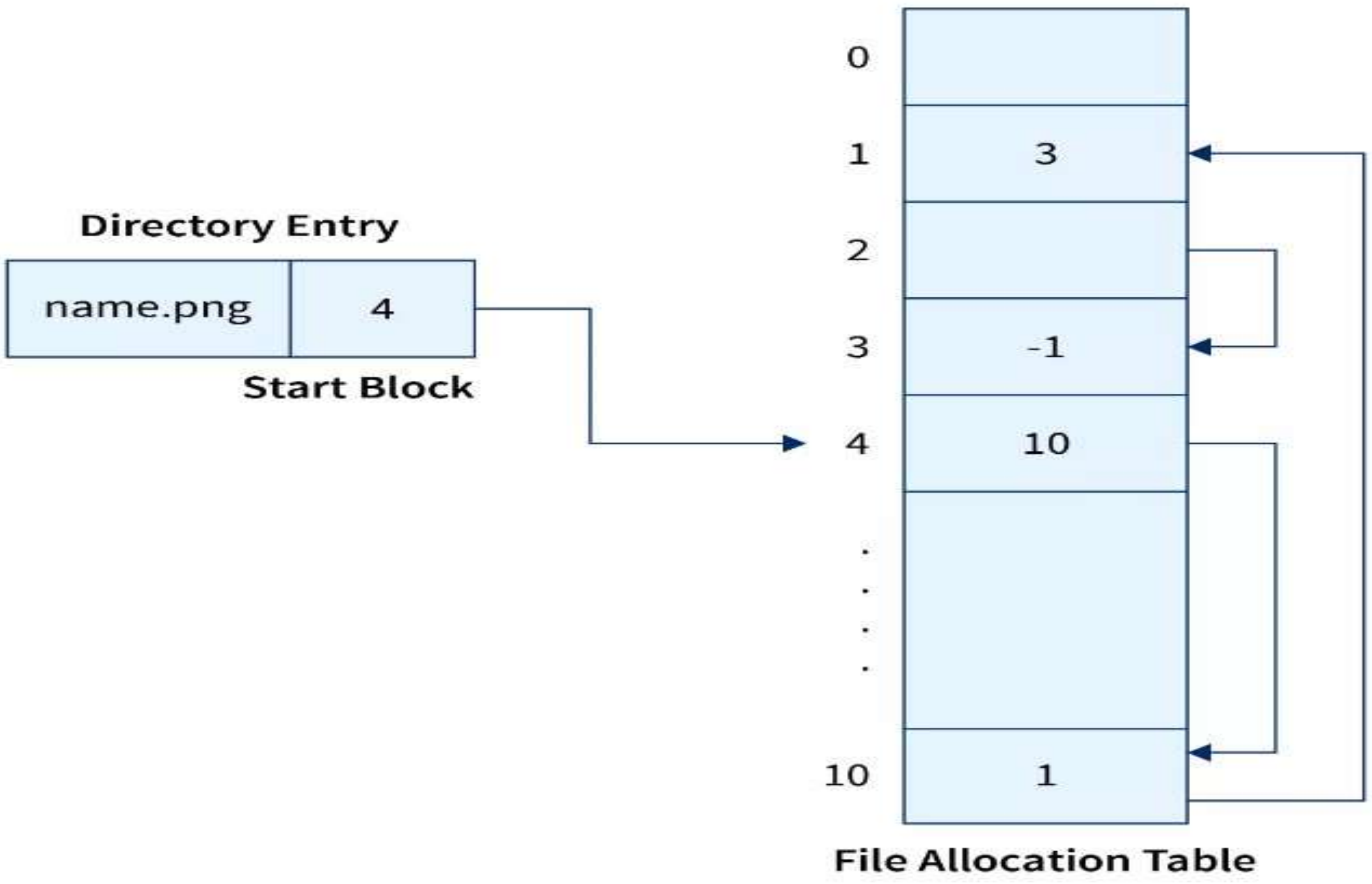
- In Combined Scheme, a special block is used to store all the information related to the file like name, authority, size, etc.
- The special block is called **inode**(information-node).
- Some space of this special block is used to store the information related to the field as mentioned above and the remaining space is used to store the addresses of blocks that contain the actual file.

File Allocation Table (FAT)

- The File Allocation Table (FAT) overcomes the drawback of Linked File allocation.
- The random access of a particular block is not possible in the linked file allocation. To access a particular block it is necessary to access all its previous blocks.
- Let's see how File Allocation Table works.

Explanation

- In the file allocation table, all disk block links are collected and maintained. Here the number of head seeks is reduced by caching the file allocation table so that the head does not need to go through all the disk blocks to access one particular block.
- The whole process of randomly accessing any block using FAT is completed by reading the desired entry of a block from the file allocation table and accessing that particular block.
- The diagrammatic representation of FAT is given below -


Directory Entry
name.png
4
Start Block
0
1
3
2
3
-1
4
10
10
1
File Allocation Table

Advantages

- Random Access to the block is possible in FAT.
- One bad/corrupted disk block cannot corrupt all the other blocks.
- It uses all the disk blocks for data as in linked file allocation it needs extra space for pointers.

Disadvantages

- If entries increase so the FAT size also increases.
- Each entry requires the FAT entry.
- If Entries increase the FAT size increases which increases the size of a block so there are chances of internal fragmentation.

Inode.

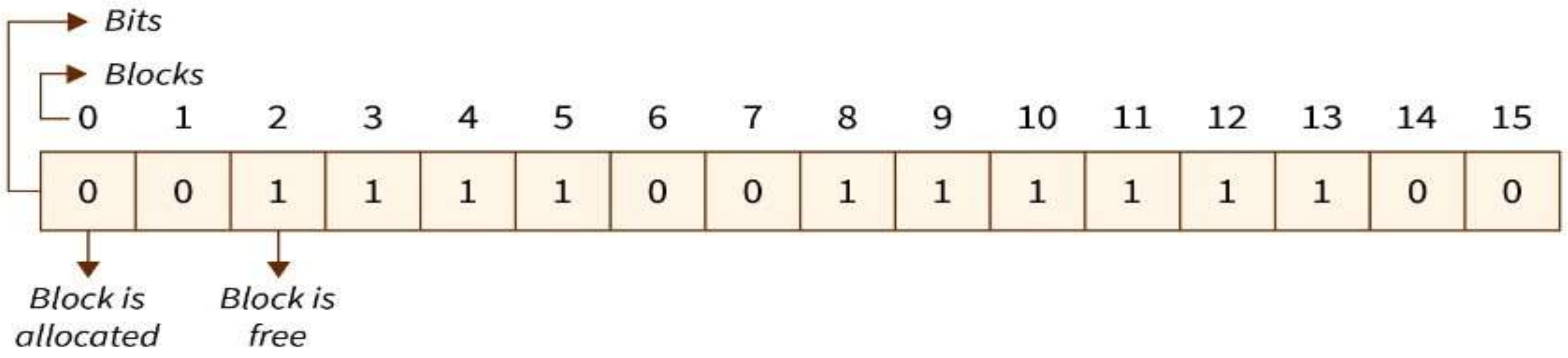
- In Operating systems based on UNIX, every file is indexed using **Inode(information-node)**. The inode is the special disk block that is created with the creation of the file system. This special block is used to store all the information related to the file like name, authority, size, etc along with the pointers to the other blocks where the file is stored.

Free space management

- It is not easy work for an operating system to allocate and de-allocate memory blocks (managing free space) simultaneously.
- The operating system uses various methods for adding free space and freeing up space after deleting a file.
- There are various methods using which a free space list can be implemented. We are going to explain them below-

Bitmap or Bit Vector :

- A bit vector is the most frequently used method to implement the free space list. A bit vector is also known as a **"Bit map"**. It is a series or collection of bits in which each bit represents a disk block. The values taken by the bits are either 1 or 0. If the block bit is 1, it means the block is empty and if the block bit is 0, it means the block is not free. It is allocated to some files. Since all the blocks are empty initially so, each bit in the bit vector represents 0.
- **Let us take an example :**
- Given below is a diagrammatic representation of a disk in which there are 16 blocks. There are some free and some occupied blocks present. The upper part is showing block number. Free blocks are represented by '1' and occupied blocks are represented by '0'.



- **"Free block number"** can be defined as that block which does not contain any value, i.e.*i*.*e*., they are free blocks. The formula to find a free block number is :
- [Block number = (number of bits per words)*(number of 0-value word) + Offset of first 1 bit]

- We will consider the first 8 bits group (00111100011110) to constitute a non-zero word since all bits are not 0 here. **"Non-zero word"** is that word that contains the bit value '1' (block that is not free). Here, the first non-zero word is the third block of the group. So, the offset will be '3'.
- Hence, the **block number** = 8 * 0 + 3 = 3=8∗0+3=3
- **Advantages of Bit vector method :**

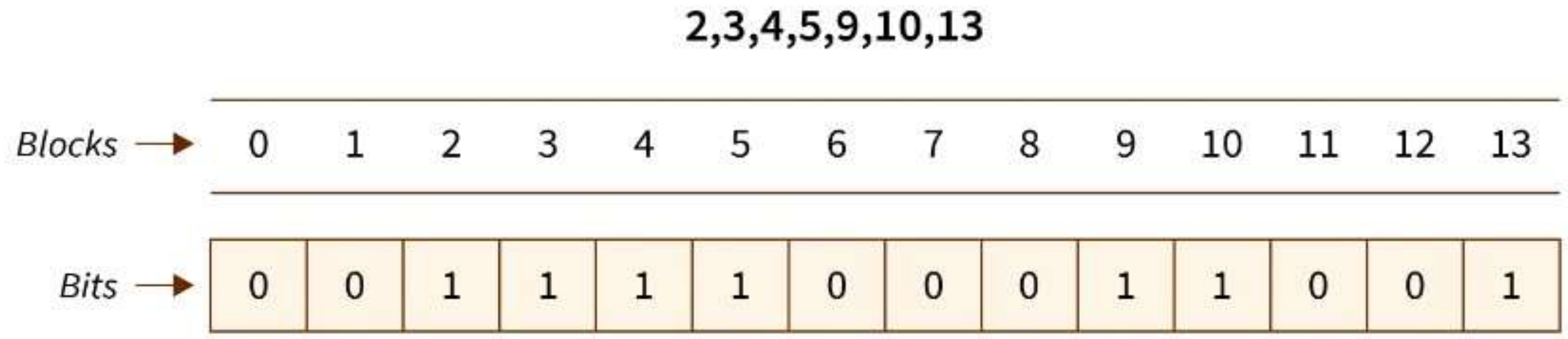
1. Simple and easy to understand.
2. Consumes less memory.
3. It is efficient to find free space.

- **Disadvantages of Bit vector method :**

1. The operating system goes through all the blocks until it finds a free block. (block whose bit is 0).
2. It is not efficient when the disk size is large.

Linked List :

- A **linked list** is another approach for free space management in an operating system.
- In it, all the free blocks inside a disk are linked together in a **"linked list"**.
- These free blocks on the disk are linked together by a pointer.
- These pointers of the free block contain the address of the next free block and the last pointer of the list points to null which indicates the end of the linked list.
- This technique is not enough to traverse the list because we have to read each disk block one by one which requires I/O time.



In the above example, block 2 is the first free block and after that block 3 is the next free block, and then block 4 is free. Block 2 contains a pointer that indicates block 3 and blocks 3 contains a pointer that indicates the next free block 4. This continues until the pointer reached the last free block.

Advantage of the Linked list :

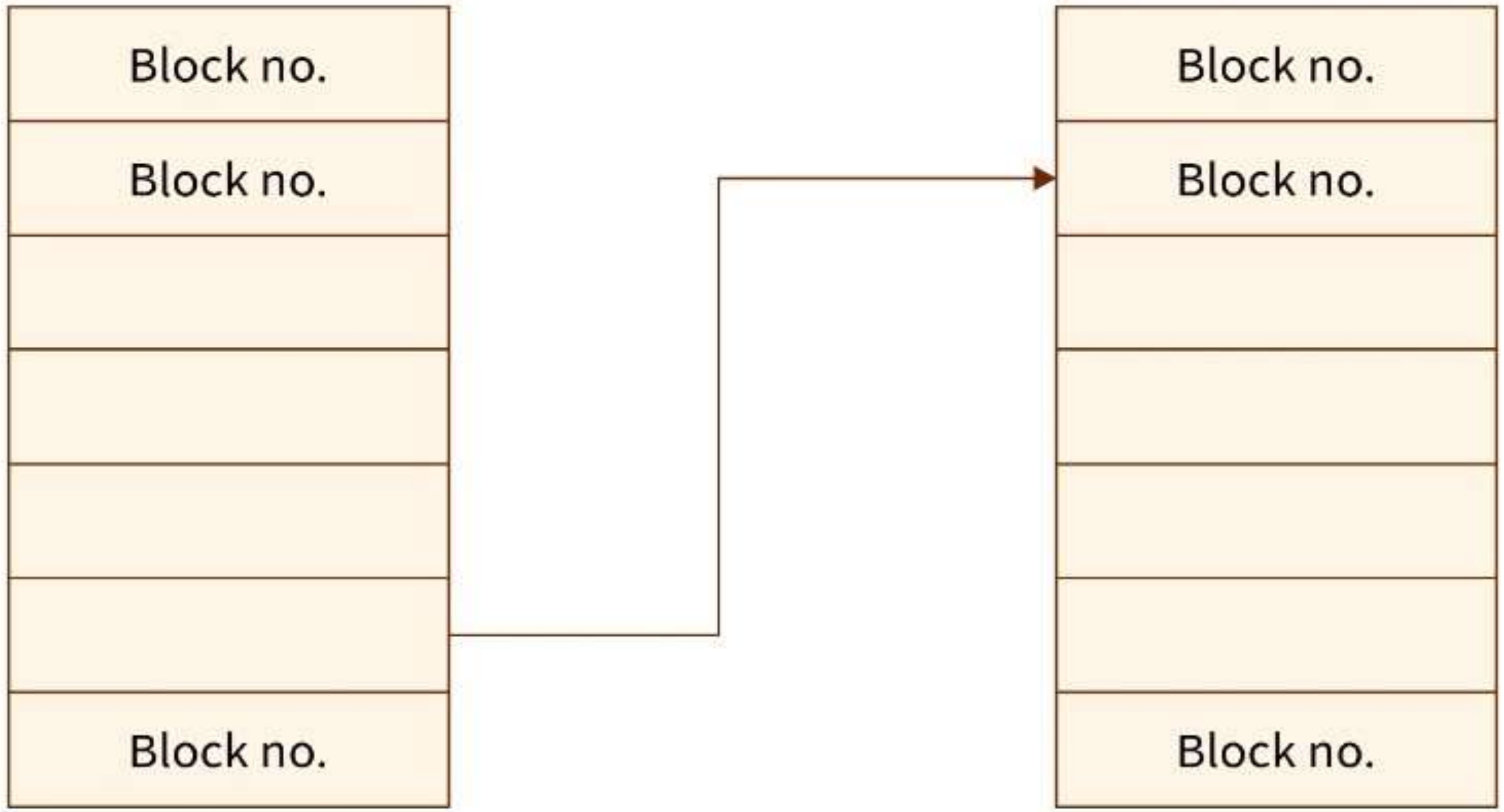
1. In this method, available space is used efficiently.
2. As there is no size limit on a linked list, a new free space can be added easily.

Disadvantages :

1. In this method, the overhead of maintaining the pointer appears.
2. the Linked list is not efficient when we need to reach every block of memory.

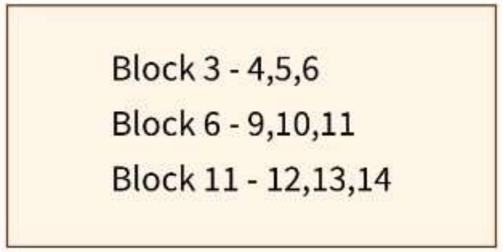
Grouping :

- The grouping technique is also called the **"modification of a linked list technique"**.
- In this method, first, the free block of memory contains the addresses of the n-free blocks.
- And the last free block of these n free blocks contains the addresses of the next n free block of memory and this keeps going on.
- This technique separates the empty and occupied blocks of space of memory.


Block no.
Block no.
Block no.
Block no.
Block no.
Block no.

Example :

- Suppose we have a disk with some free blocks and some occupied blocks. The free block numbers are 3, 4, 5, 6, 9, 10, 11, 12, 13,3,4,5,6,9,10,11,12,13, and 1414. And occupied block numbers are 1, 2, 7, 8, 15,1,2,7,8,15, and 1616 i.e.*i*.*e*., they are allocated to some files.



Block 3 - 4,5,6
Block 6 - 9,10,11
Block 11 - 12,13,14

- When the **"grouping technique"** is applied, block 3 will store the addresses of blocks 4, 5, and 6 because **'block 3'** is the first free block. In the same way, block 6 will store the addresses of blocks 9, 10, and 11 because block 6 is the first occupied block.
- **Advantage of the Grouping method :**

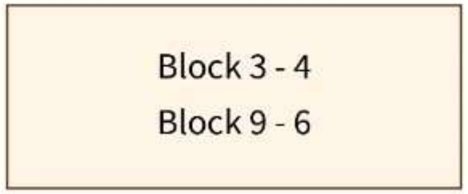
1. By using this method, we can easily find addresses of a large number of free blocks easily and quickly.

- **Disadvantage :**

1. We need to change the entire list if one block gets occupied.

Counting :

- In memory space, several files are created and deleted at the same time. For which memory blocks are allocated and de-allocated for the files.
- Creation of files occupy free blocks and deletion of file frees blocks. When there is an entry in the free space, it consists of two parameters- **"address of first free disk block (a pointer)"** and **" a number 'n' "**.
- **Example :**
- Let us take an example where a disk has 16 blocks in which some blocks are empty and some blocks are occupied as given below :


Block 3 - 4
Block 9 - 6

When the **"counting technique"** is applied, the block number 3 will represent block number 4 because block 3 is the first free block. Then, the block stores the number of free blocks i.e.*i.e*. - there are 4 free blocks together. In the same way, the first occupied block number 9 will represent block number 10 and keeps the number of rest occupied blocks i.e.- there are 6 occupied blocks as shown in the above figure.

Advantages :

1.In this method, a bunch of free blocks take place fastly.

2.The list is smaller in size.

Disadvantage :

1.In the counting method, the first free block stores the rest free blocks, so it requires more space.

Efficiency and Performance

Mass Storage Structure

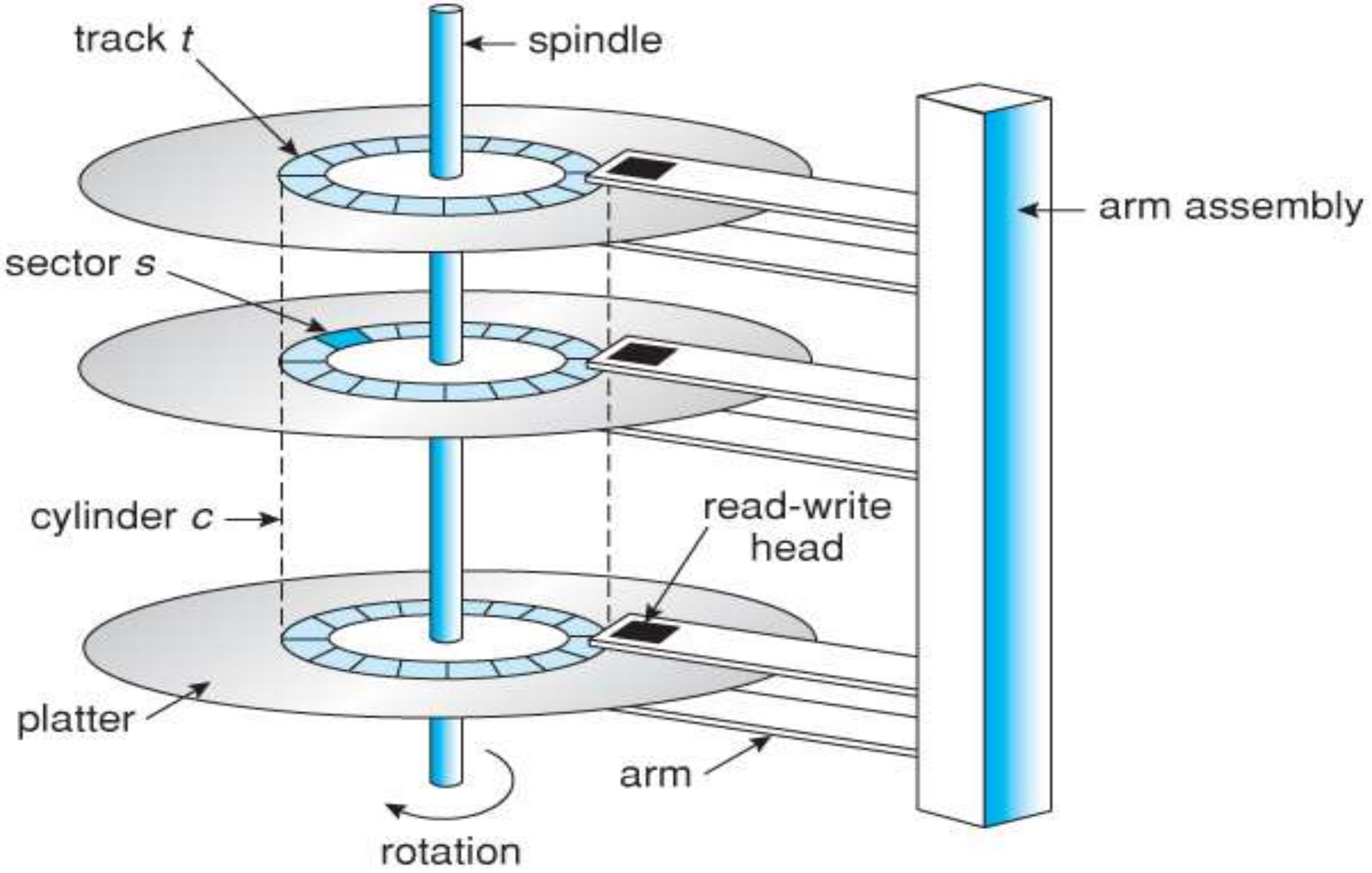
Magnetic Disks

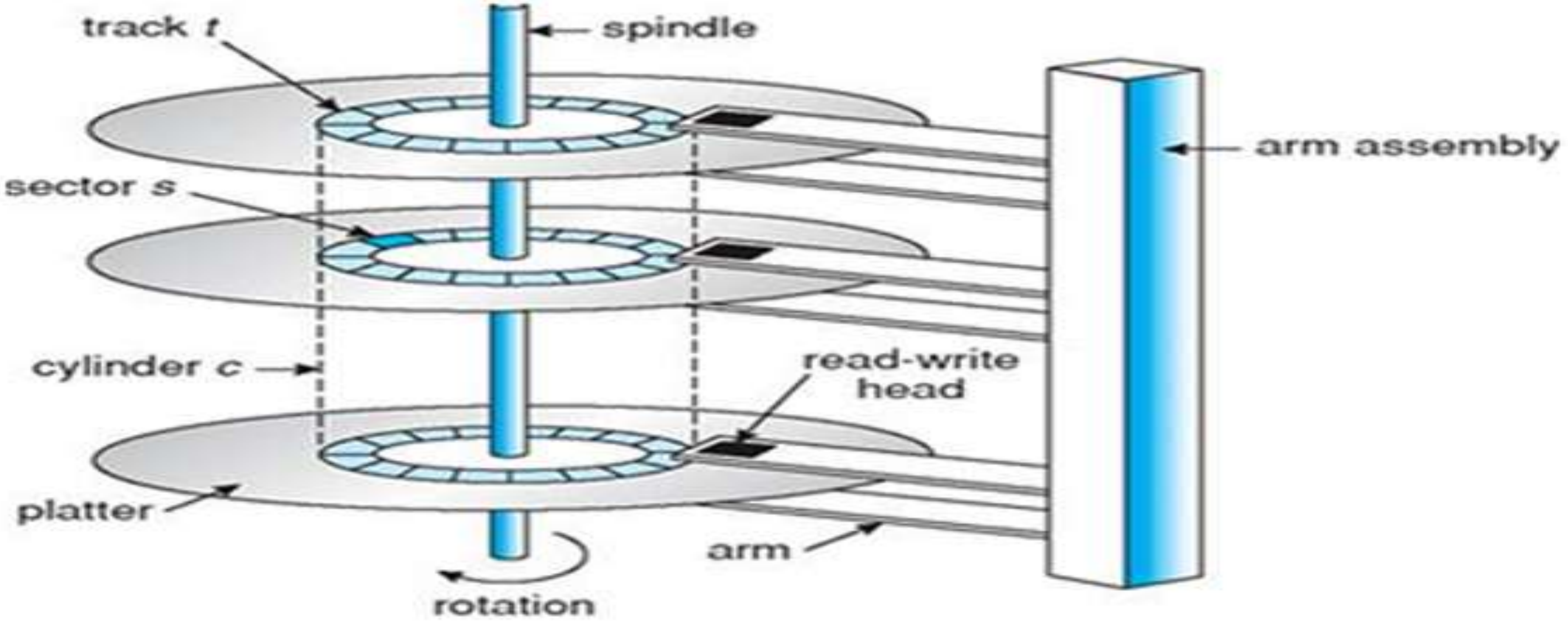
Traditional magnetic disks have the following basic structure:

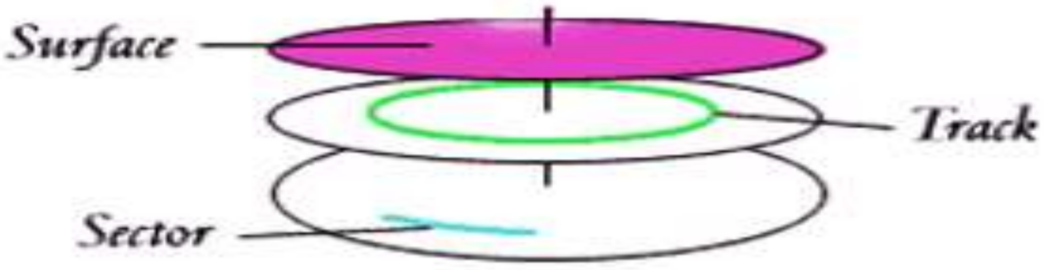
- One or more ***platters*** in the form of disks covered with magnetic media. ***Hard disk*** platters are made of rigid metal, while "***floppy***" disks are made of more flexible plastic.
- Each platter has two working ***surfaces.*** Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.
- Each working surface is divided into a number of concentric rings called ***tracks.*** The collection of all tracks that are the same distance from the edge of the platter, (i.e. all tracks immediately above one another in the following diagram) is called a ***cylinder***.

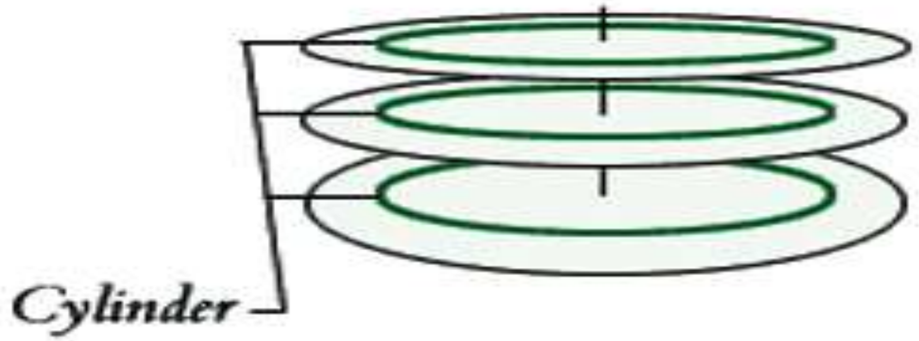
- Each track is further divided into ***sectors,*** traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. (Sectors also include a header and a trailer, including checksum information among other things.
- Larger sector sizes reduce the fraction of the disk consumed by headers and trailers, but increase internal fragmentation and the amount of disk that must be marked bad in the case of errors.)
- The data on a hard drive is read by read-write ***heads.*** The standard configuration (shown below) uses one head per surface, each on a separate ***arm***, and controlled by a common ***arm assembly*** which moves all heads simultaneously from one cylinder to another.
- (Other configurations, including independent read-write heads, may speed up disk access, but involve serious technical difficulties.)

- The storage capacity of a traditional disk drive is equal to the number of heads (i.e. the number of working surfaces), times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector.
- A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located.


track *t*
spindle
arm assembly
sector *s*
cylinder *c*
read-write head
platter
arm
rotation







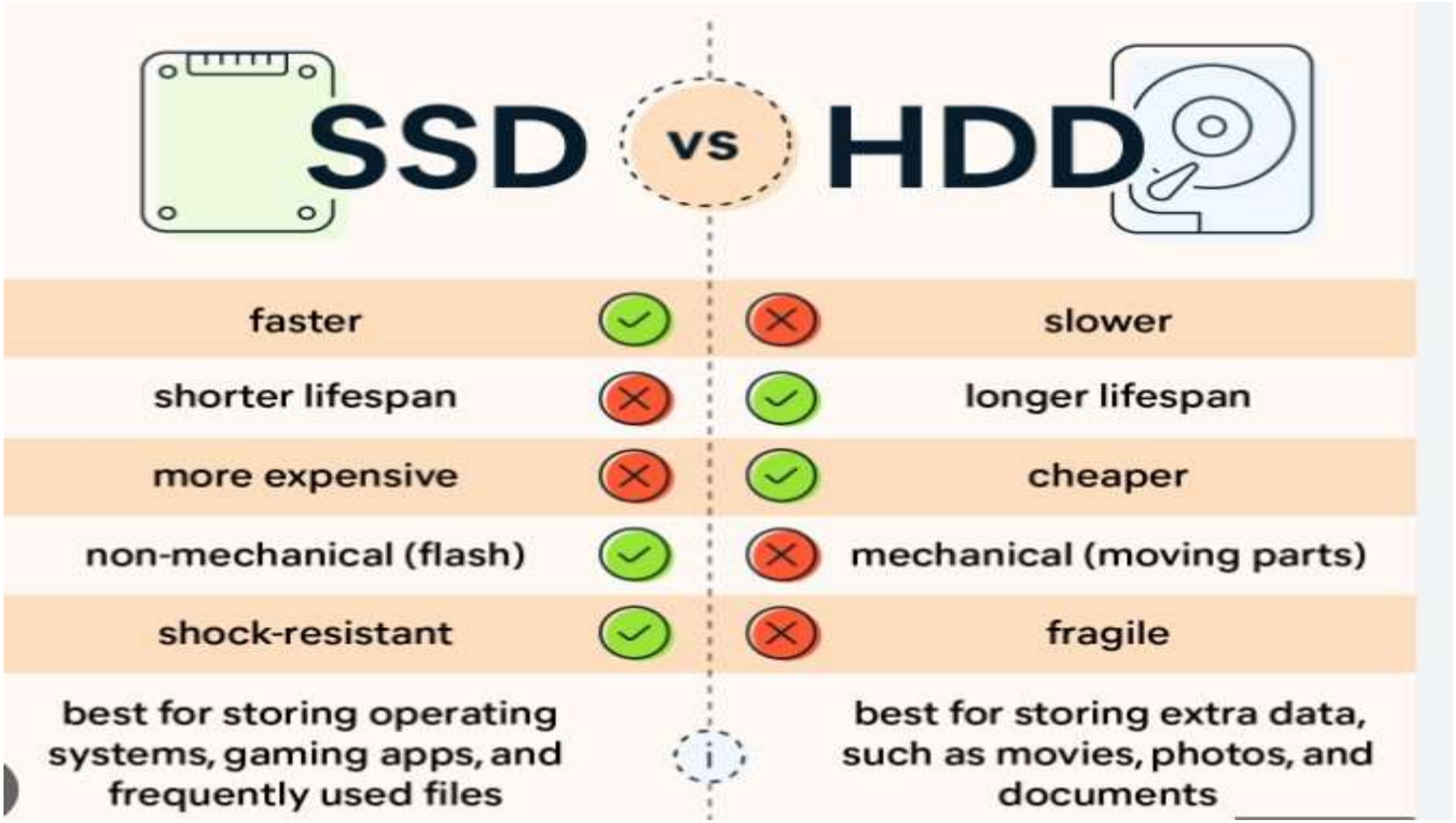
physical disk structure

Solid State Disk

- Old technologies are frequently employed in new ways as economic conditions and technology evolve. The growing usage of solid state drives, or SSDs, is one illustration of this.
- SSDs function as a tiny, quick hard disk using memory technology. To maintain the information over power cycles, certain implementations may employ either flash memory or DRAM chips protected by a battery.
- Due to the lack of moving components, SSDs operate far more quickly than conventional hard drives, and some issues, such the scheduling of disk accesses, simply do not exist.
- SSDs do have certain drawbacks, too, including the fact that they cost more than hard drives, are often smaller, and may have shorter life spans.

- As a high-speed cache for information on hard disks that has to be retrieved fast, SSDs are particularly helpful.
- One use is to store frequently requested file system meta-data, such as directory and I node information.
- A boot drive is another version that has the OS and certain application executables but no essential user data.
- In order to make laptops thinner, lighter, and quicker, SSDs are also employed in them.
- The throughput of the bus may become a limiting problem due to how much quicker SSDs are than conventional hard drives, which leads to certain SSDs being linked directly to the system PCI bus.

SUPER ★ TALENT
64GB
SATA 2.5"
SOLID STATE DRIVE
SAM64GM25S



SSD		vs		HDD
faster	✓		✗	slower
shorter lifespan	✗		✓	longer lifespan
more expensive	✗		✓	cheaper
non-mechanical (flash)	✓		✗	mechanical (moving parts)
shock-resistant	✓		✗	fragile
best for storing operating systems, gaming apps, and frequently used files		i		best for storing extra data, such as movies, photos, and documents

Magnetic Tapes

- Prior to the advent of hard disk drives, magnetic tapes were frequently utilized for secondary storage; today, they are mostly used for backups.
- It might take a while to get to a specific location on a magnetic tape, but once reading or writing starts, access rates are on par with disk drives.
- Tape drive capacities may be anywhere from 20 and 200 GB, and compression can increase that capacity by double.

Disk Attachment

There are two ways for computers to access disc storage.

One method is to use I/O ports, also known as **host-attached storage,** on small systems. Another method is through a remote host in a distributed file system, which is known as **network-attached storage.**

Host-Attached Storage

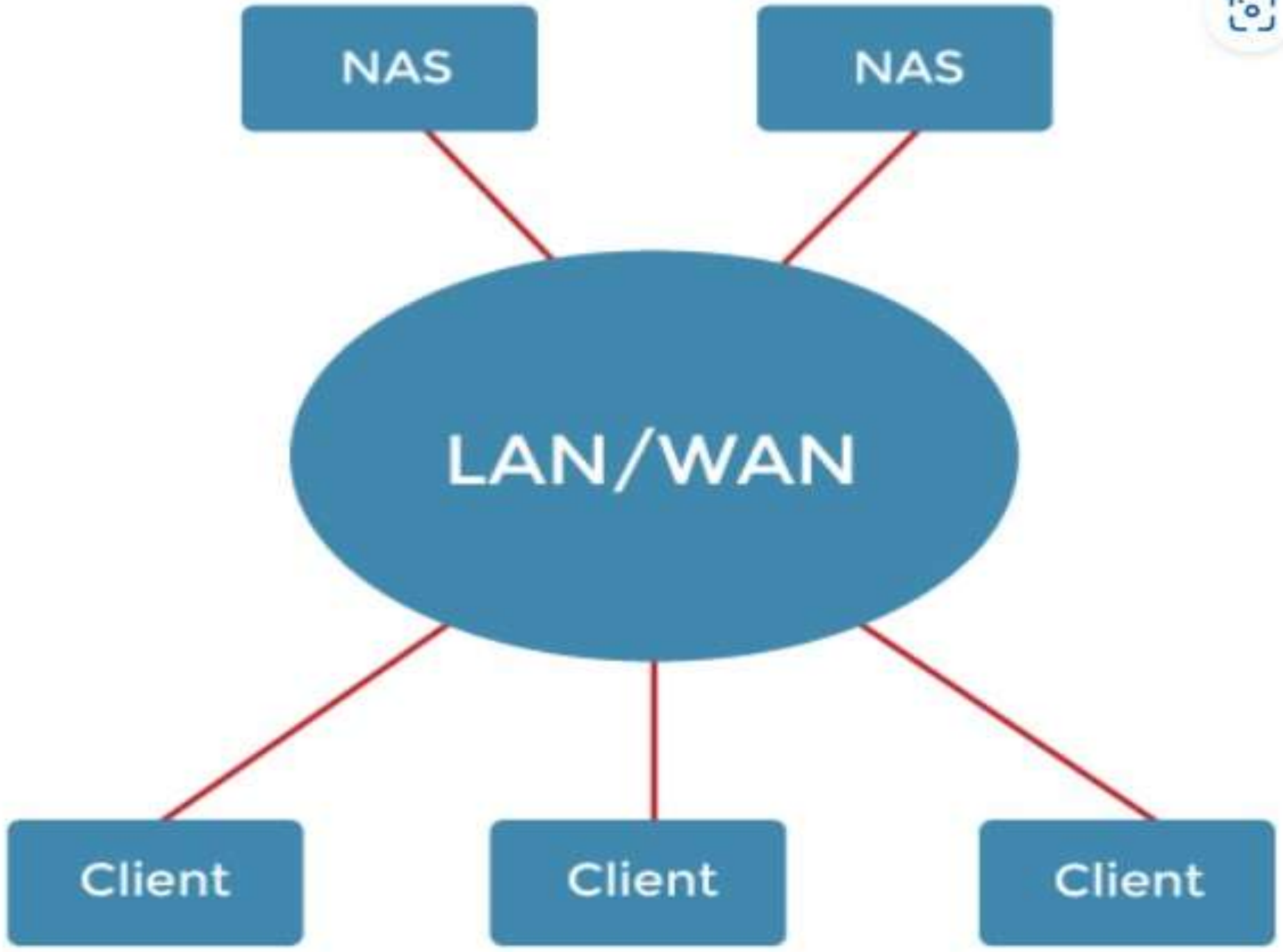
- **Host-attached storage (HAS)** is storage accessed via local I/O ports. These ports make use of various technologies.
- IDE or ATA is the I/O bus architecture used by most desktop PCs. This architecture allows for up to two drivers per I/O bus.
- SATA (Serial Advanced Technology Attachment) is a new related standard that has simplified cabling. High-end workstations and servers typically use more advanced I/O architectures such as Small Computer System Interface (SCSI) and fibre channel (FC).
- **SCSI (Small Computer System Interface)** is bus architecture. Its physical medium is typically a ribbon wire with many conductors.
- A maximum of **16** devices may be attached to the bus using the SCSI protocol.

- The devices typically comprise of one controller card (SCSI initiator) on the host and up to **15** storage devices (SCSI targets).
- A SCSI disk is a typical SCSI target. Although, the protocol permits every SCSI target to address up to 8 logical units.
- Logical unit addressing is widely used to direct commands to the RATD array or portable media library components.
- A Fibre Channel is a high-speed serial architecture that may use optical fibre or a four-conductor copper wire.
- It comes in two varieties. One type of fabric is a big switched fabric with a **24-bit** address space.
- This variant is projected to take the lead in the future and will serve as the foundation for storage-area networks (SANs).

- Many hosts and storage devices may be connected to the fibre due to the large address space and switched nature of the communication that provides better flexibility in I/O communication.
- Another PC version is an arbitrated loop (FC-AL), which may address up to **126** devices, including drives and controllers.
- As host-attached storage, a wide range of storage devices are suitable. Hard disc devices, RAID arrays, CDs, DVDs, and tape drives are among them.
- The I/O commands that start data transfer to a host-attached storage device can read and write logical data blocks routed to specially designated storage units.

Network-Attached Storage

- A **network-attached storage (NAS)** device is a dedicated storage system that can be accessed via the data network.
- Clients access NAS through a remote procedure call interface, like a CIFS for Windows computers or a network file system for Unix.
- **Remote procedure calls (RPCs)** are transmitted through TCP or UDP via an IP network, which is often the same LAN that delivers all data traffic to the clients.
- Generally, the network-attached storage unit is constructed as a RAID array with software that supports the RPC interface.
- It is easy to consider NAS as just another storage-access protocol. For example, a NAS system may access storage through RPC via TCP/IP rather than a SCSI device driver and SCSI protocols.

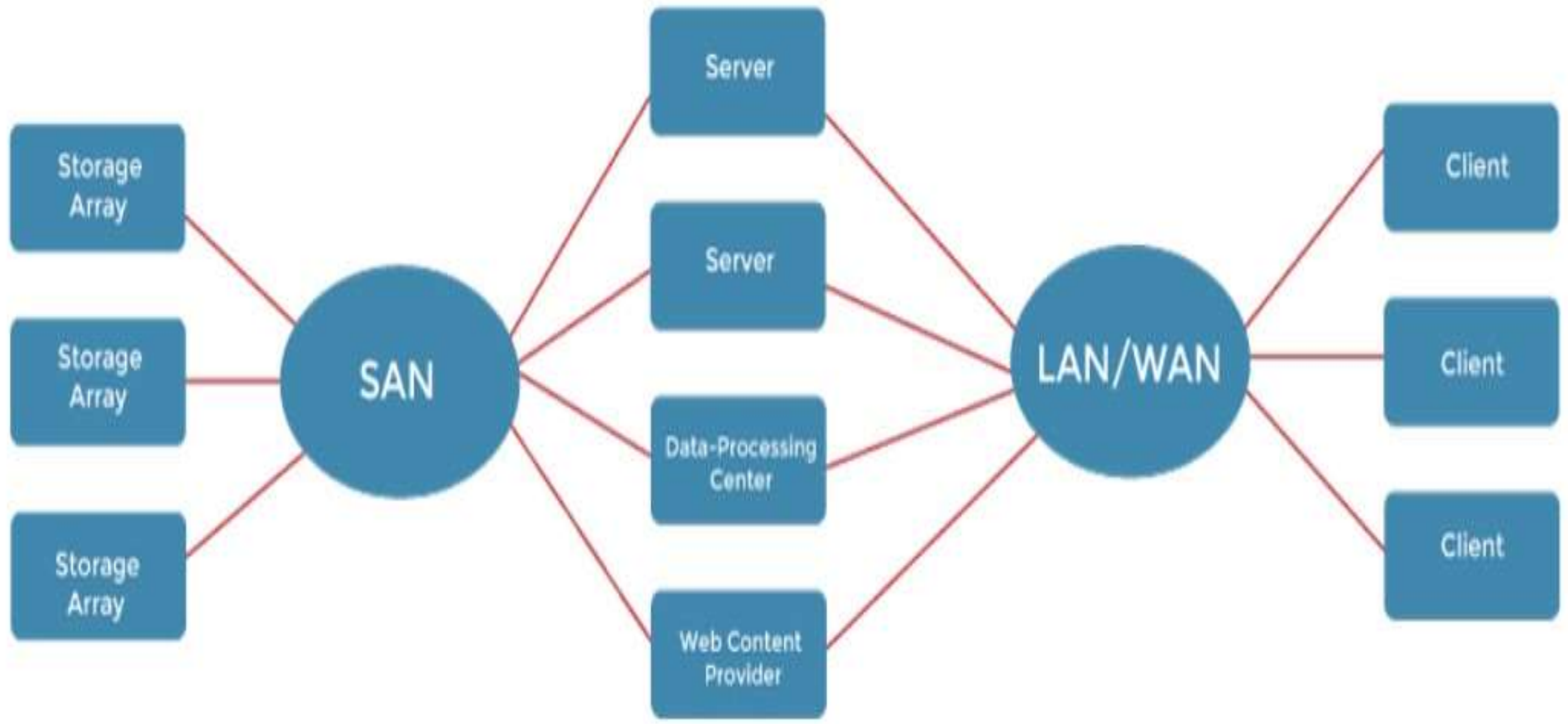


Network - Attached Storage

- NAS allows all systems on a local area network to share a pool of storage with the same simplicity of naming and access as local host-attached storage provides.
- However, it is less efficient and performs worse than other direct-attached storage solutions.
- **ISCSI (Internet Small Computer Systems Interface)** is the latest network-attached storage protocol.
- The SCSI protocol is carried across the IP network protocol.
- As a result, the SCSI protocol uses the networks to interconnect hosts and their storage than SCSI cables.

Storage-Area Network

- The one downside of network-attached storage systems is bandwidth usage on the data network by storage I/O activities, which increases network latency.
- This problem is most acute in large client-server configurations because server-to-client communications compete for bandwidth with server-to-storage communications.



Storage - Area Network

- SAN is a private network that uses storage protocols to connect servers and storage devices instead of networking standards. T
- he flexibility of a SAN is its strength. Multiple hosts and storage arrays may connect to the same SAN, and storage may be allocated dynamically to hosts.
- A SAN switch permits or prohibits access between hosts and storage. For example, if a host's disk space is running short, the SAN may be set to allocate more storage to that server.
- SANs enable server clusters to share the same storage and storage arrays to contain several direct host connections.

- SANs often have more ports at a lower cost than storage arrays. The most common is FC.
- InfiniBand is a new option for high-speed connectivity networks for servers and storage units.
- It is a special-purpose bus architecture that provides hardware and software support for servers and storage units high-speed connectivity networks.

Disk Scheduling

- As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.
- However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.
- The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.
- Let's discuss some important terms related to disk scheduling.

Seek Time

- Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

- It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

- It is the time taken to transfer the data.

Disk Access Time

- Disk access time is given as,
- Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

- It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

- The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages.

The limitation of each algorithm leads to the evolution of a new algorithm.

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

FCFS Scheduling Algorithm

- It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Advantages:

- Every request gets a fair chance
- No indefinite postponement

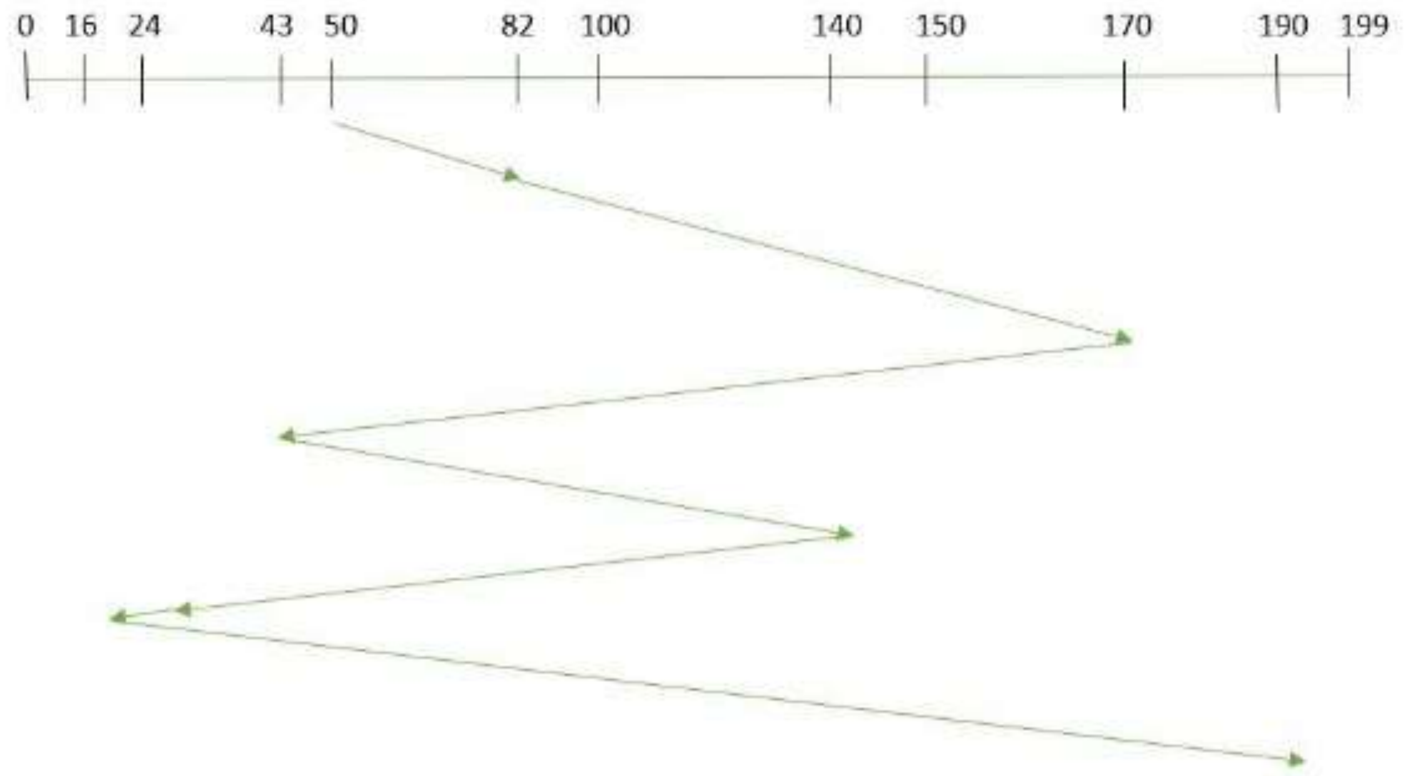
Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

Example:

Total number of tracks are 200. Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is : 50



total seek time:

=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)

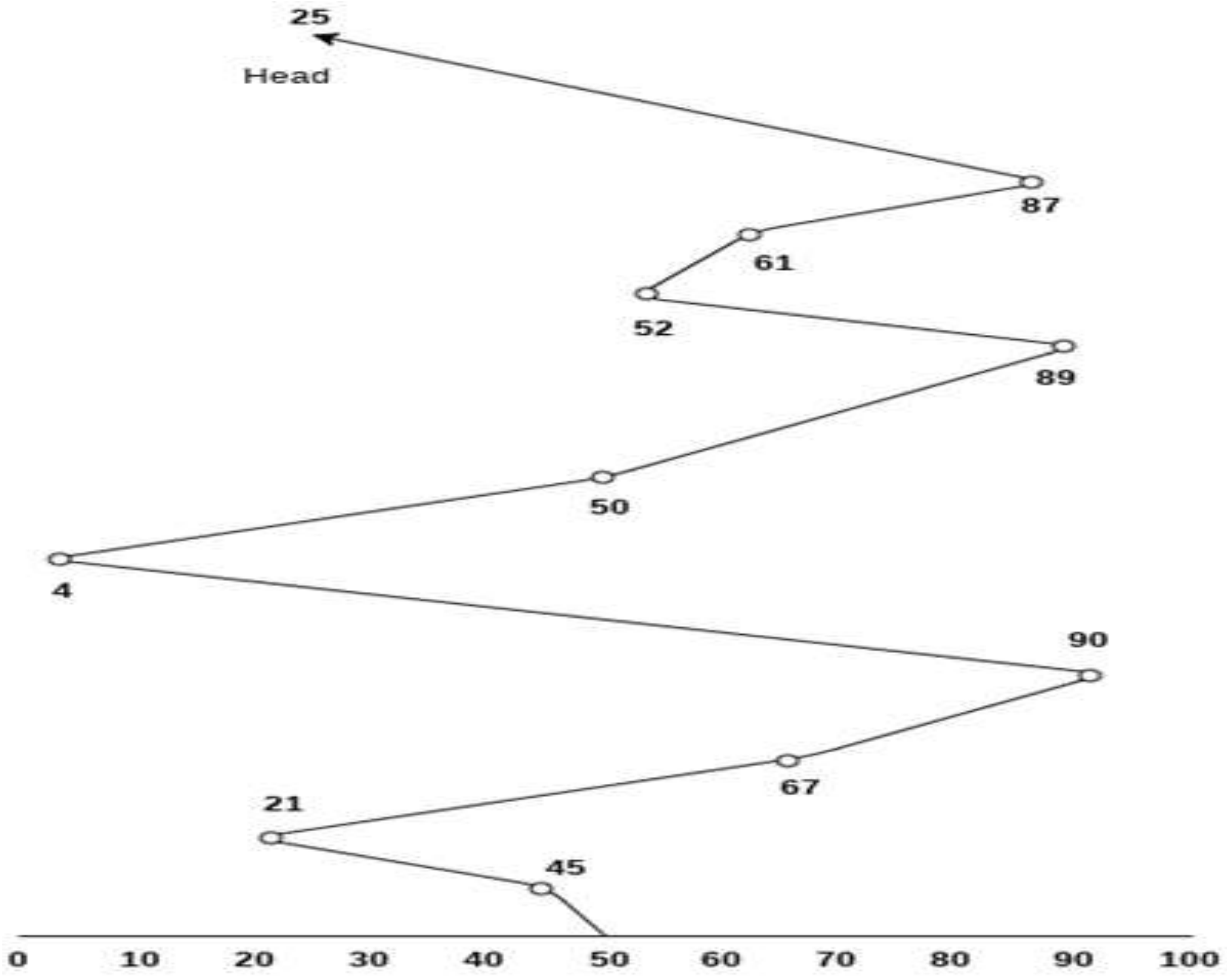
=642

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25 .

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Solution:


25
Head
87
61
52
89
50
4
90
67
21
45
0
10
20
30
40
50
60
70
80
90
100

Number of cylinders moved by the head

=(50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

SSTF Scheduling Algorithm

- Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.
- It allows the head to move to the closest track in the service queue.

Advantages:

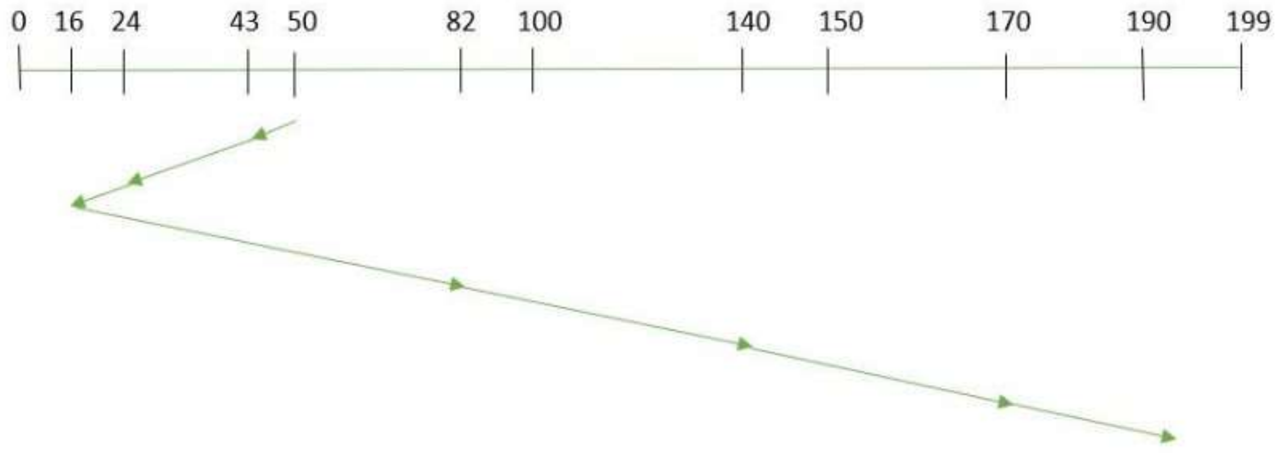
- Average Response Time decreases
- Throughput increases

Disadvantages

- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

<u>**Example:**</u>

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



total seek time:

=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170)

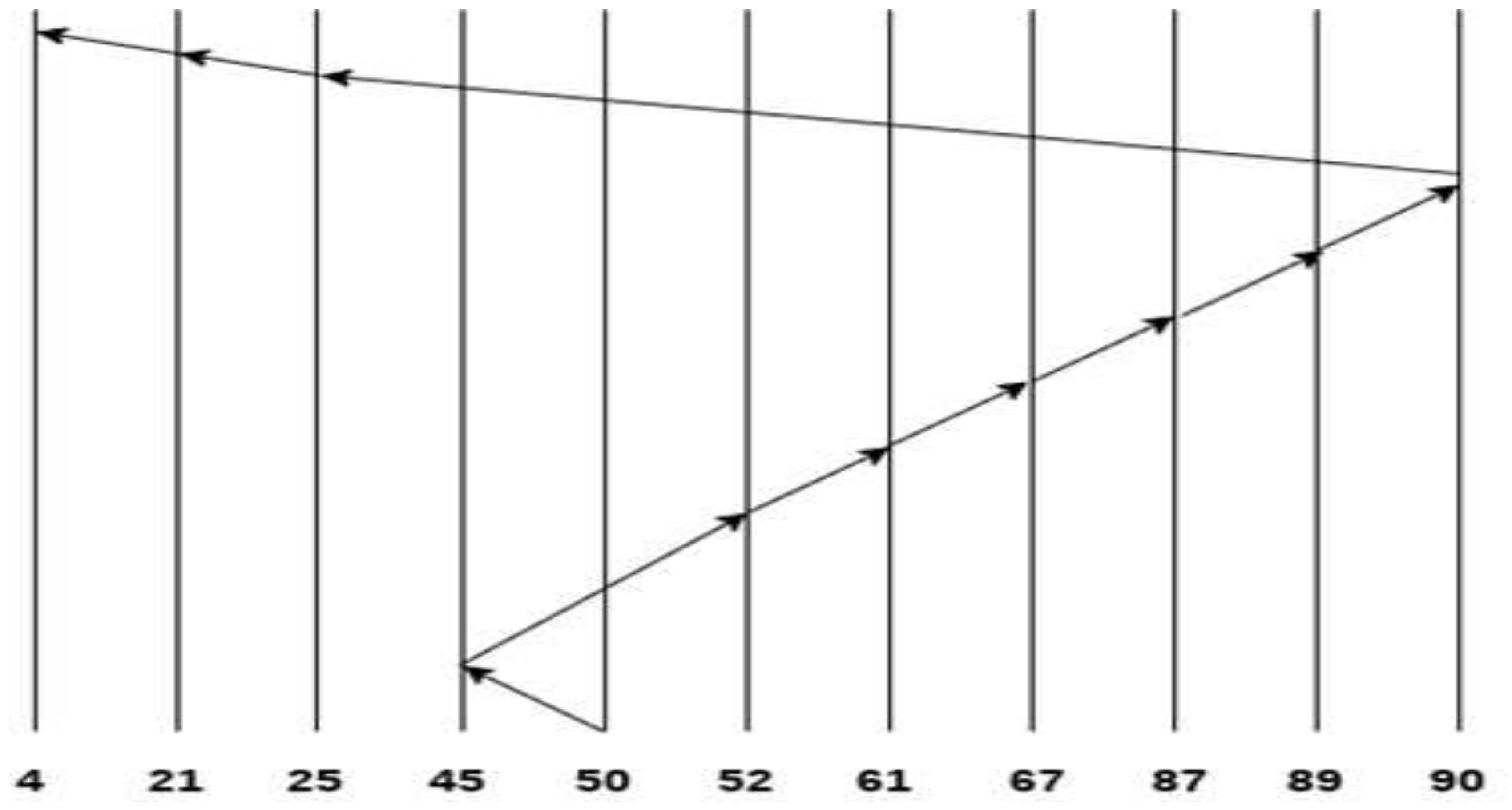
=208

Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.



Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

SCAN

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

Advantages:

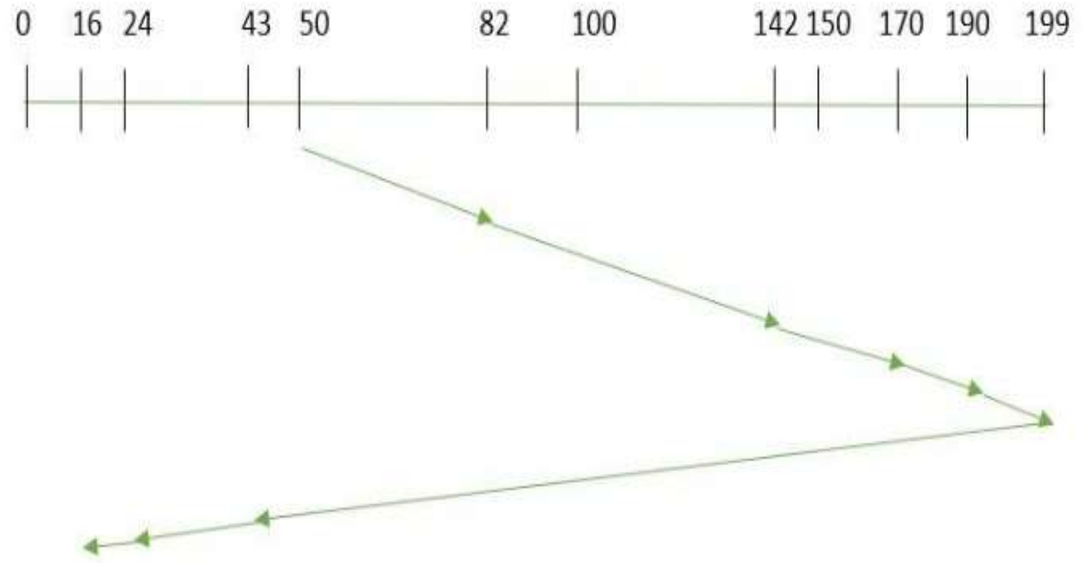
- High throughput
- Low variance of response time
- Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

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".**



the seek time is calculated as: =(199-50)+(199-16)

=332

CSCAN

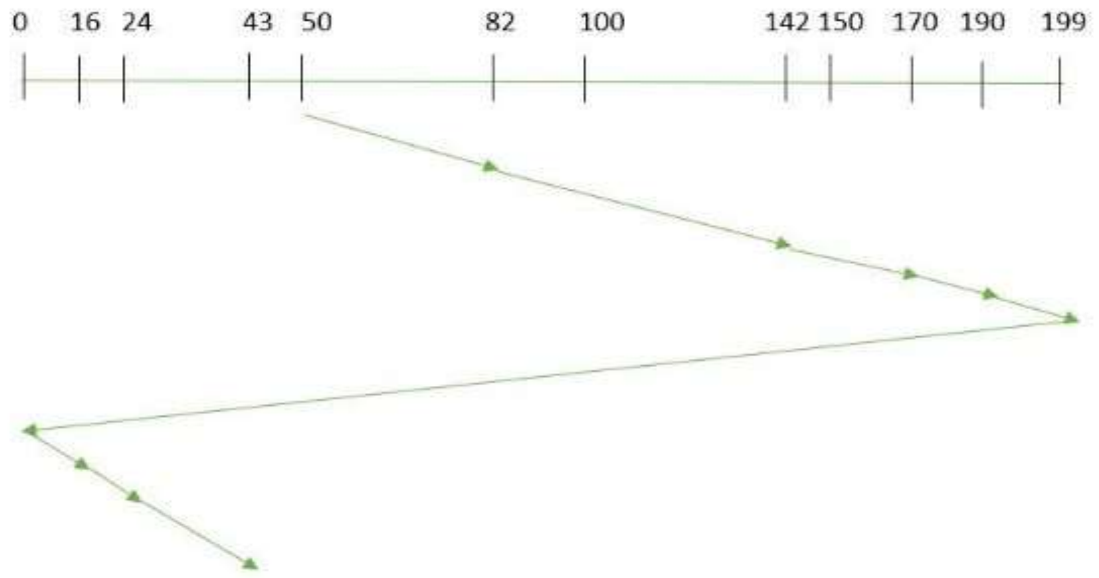
- In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
- These situations are avoided in *CSCAN* algorithm 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 SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

- Provides more uniform wait time compared to SCAN

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”.**



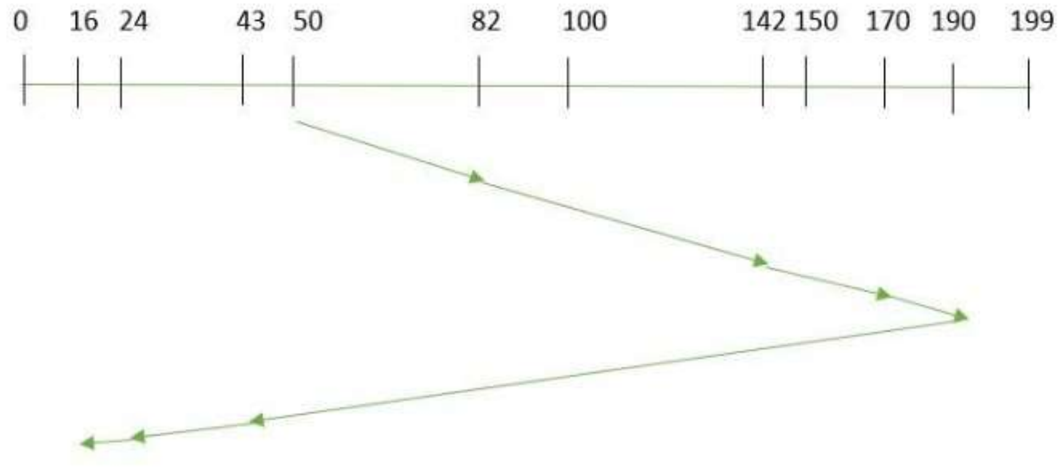
Seek time is calculated as:
=(199-50)+(199-0)+(43-0)
=391

LOOK

- It 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.

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".**



the seek time is calculated as:

=(190-50)+(190-16)

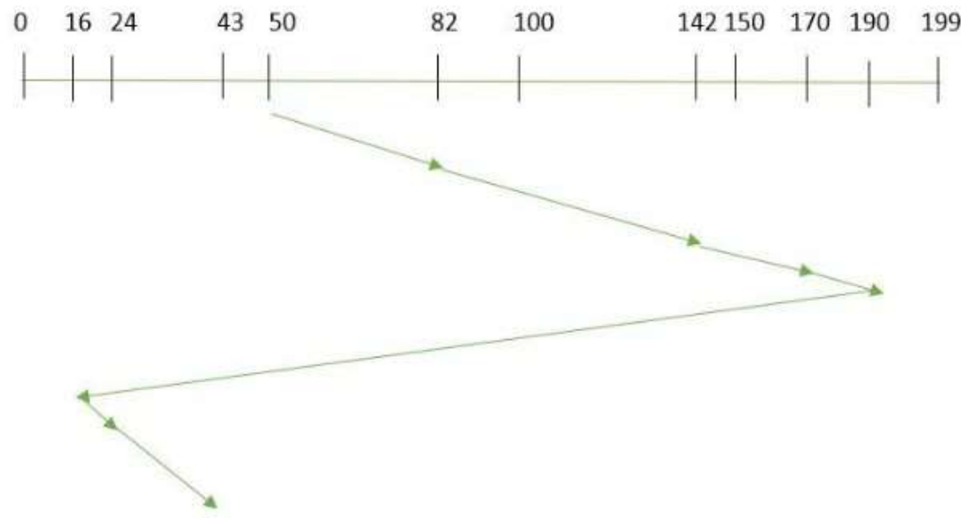
=314

CLOOK

As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to 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"**



the seek time is calculated as:

= (190-50)+(190-16)+(43-16)

= 341

Disk management

Disk management of the operating system includes:

- Disk Format
- Booting from disk
- Bad block recovery

Disk Formatting

- A new magnetic disk is mainly a blank slate. It is platters of the magnetic recording material.
- Before a disk may hold data, it must be partitioned into sectors that may be read and written by the disk controller.
- It is known as **physical formatting** and **low-level formatting.**
- **Low-level formatting** creates a unique data structure for every sector on the drive.
- A data structure for a sector is made up of a header, a data region, and a trailer.
- The disk controller uses the header and trailer to store information like an error-correcting code (ECC) and a sector number.

- The OS must require recording its own data structures on the disk drive to utilize it as a storage medium for files.
- It accomplishes this in two phases. The initial step is to divide the disk drive into one or more cylinder groups.
- The OS may treat every partition as it were a separate disk.
- For example, one partition could contain a copy of the OS executable code, while another could contain user files.
- The second stage after partitioning is **logical formatting.**
- The operating store stores the initial file system data structure on the disk drive in this second stage.

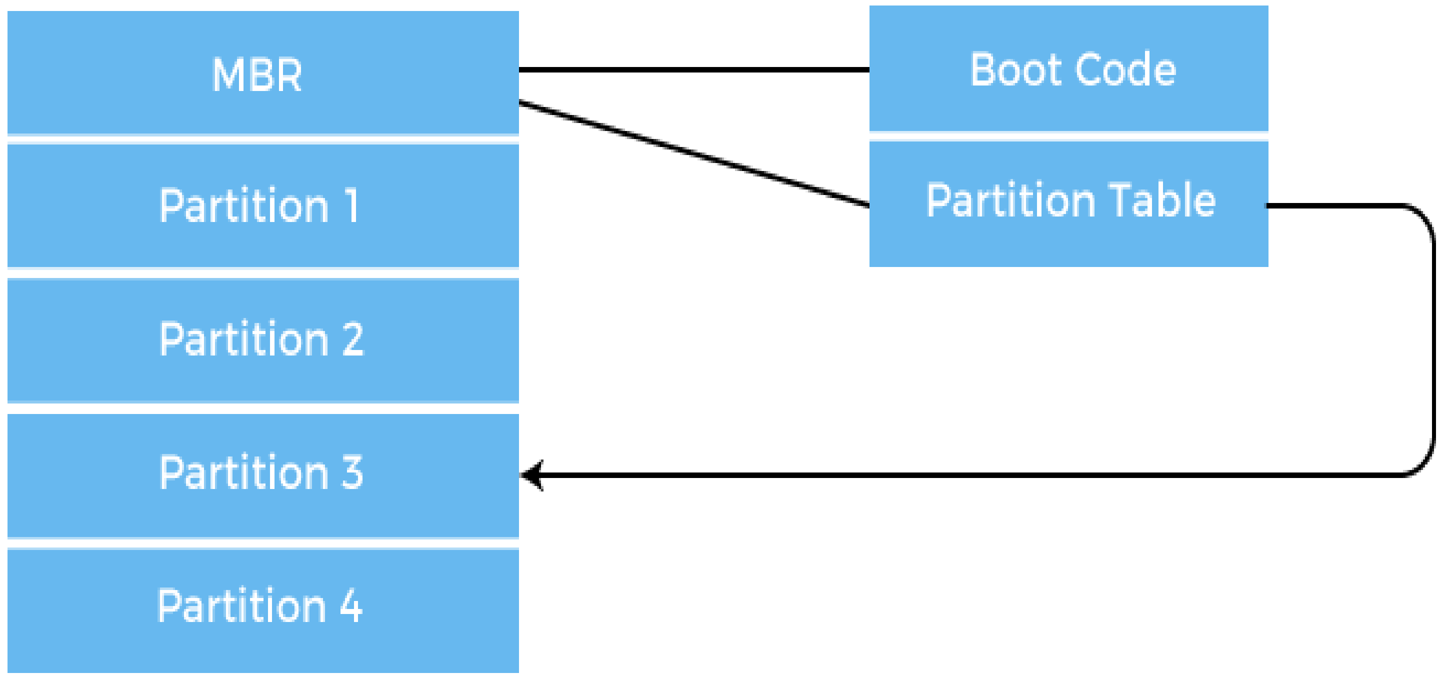
Boot block:

- When a system is turned on or restarted, it must execute an initial program. The start program of the system is called the bootstrap program.
- It starts the OS after initializing all components of the system.
- The bootstrap program works by looking for the OS kernel on disk, loading it into memory, and jumping to an initial address to start the OS execution.
- The bootstrap is usually kept in read-only memory on most computer systems.
- It is useful since read-only memory does not require initialization and is at a fixed location where the CPU may begin executing whether powered on or reset.
- Furthermore, it may not be affected by a computer system virus because ROM is read-only.
- The issue is that updating this bootstrap code needs replacing the ROM hardware chips.

- As a result, most computer systems include small bootstrap loader software in the boot ROM, whose primary function is to load a full bootstrap program from a disk drive.
- The entire bootstrap program can be modified easily, and the disk is rewritten with a fresh version.
- The bootstrap program is stored in a partition and is referred to as the **boot block.**
- A **boot disk** or **system disk** is a type of disk that contains a boot partition.

How Boot Block Works?

- Let's try to understand this using an example of the boot process in Windows 2000.
- The Windows 2000 stores its boot code in the first sector on the hard disk. The following image shows the booting from disk in Windows 2000.



- Moreover, Windows 2000 allows the hard disk to be divided into one or more partitions. This one partition is identified as the ***boot partition***, containing the operating system and the device drivers.
- In Windows 2000, booting starts by running the code placed in the system's ROM memory.
- This code allows the system to read code directly from the Master Boot Record or MBR.
- The MBR also contains the table that lists the partition for the hard disk and a flag indicating which partition is to be boot from the system.
- Once the system identifies the boot partition, it reads the first sector from memory, known as a ***boot sector***. It continues the process with the remainder of the boot process, which includes loading various system services.

Bad Blocks:

- Disks are prone to failure due to their **moving parts and tight tolerances**.
- When a disk drive fails, it must be replaced and the contents transferred to the replacement disk using backup media.
- For some time, one or more sectors become faulty.
- Most disks also come from the company with bad blocks.
- These blocks are handled in various ways, depending on the use of disk and controller.
- On the disk, the controller keeps a list of bad blocks.
- The list is initialized during the factory's low-level format and updated during the disk's life.
- Each **bad sector** may be replaced with one of the **spare sectors** by directing the controller.
- This process is referred to as sector sparing.

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 :

The area on the disk where the swapped-out processes are stored is called swap space.

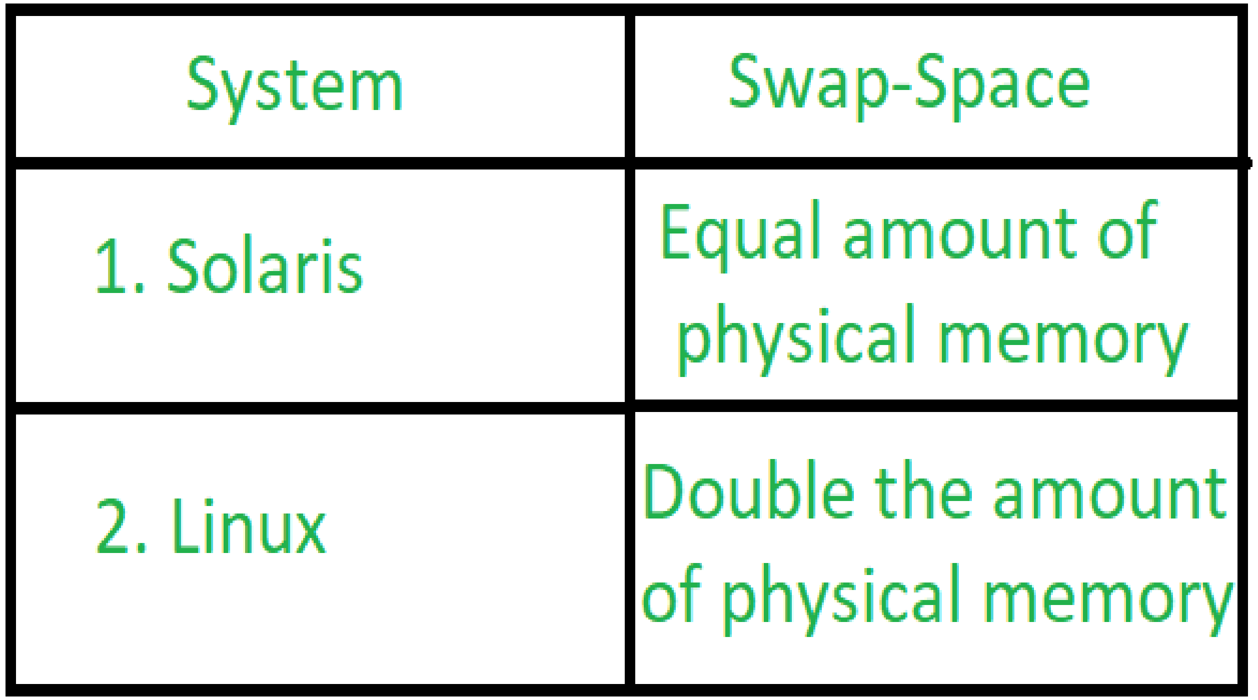
Swap-Space Management :

- 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.
- In these article, we are going to discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

Swap-Space Use :

- Swap-space is used by the different operating-system in various ways.
- The systems which are implementing swapping may use swap space to hold the entire process which may include image, code and data segments.
- Paging systems may simply store pages that have been pushed out of the main memory.
- The need of swap space on a system can vary from a megabytes to gigabytes but it also depends on the amount of physical memory, the virtual memory it is backing and the way in which it is using 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 otherwise be used for files, but it does not harm other.

Following table shows different system using amount of swap space:

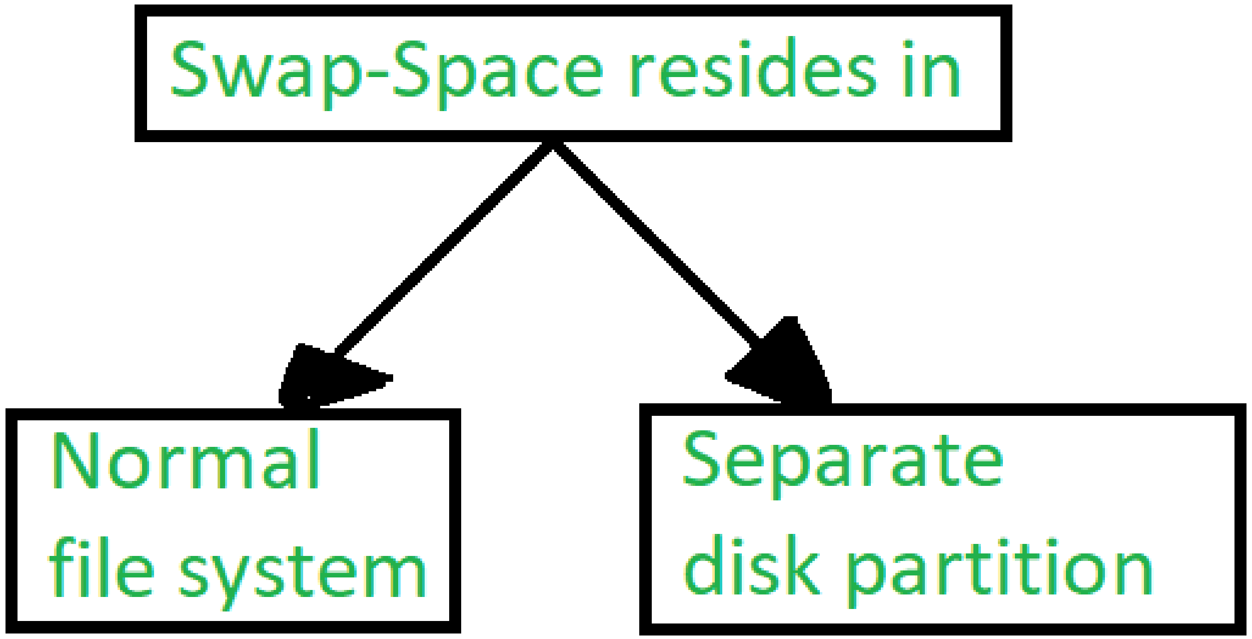


System	Swap-Space
1. Solaris	Equal amount of physical memory
2. Linux	Double the amount of physical memory

<u>Explanation of above table</u> :

- Solaris, setting swap space equal to the amount by which virtual memory exceeds page-able physical memory.
- In the past Linux has suggested setting swap space to double the amount of physical memory.
- Today, this limitation is gone, and most Linux systems use considerably less swap space.
- Including Linux, some operating systems; allow the use of multiple swap spaces, including both files and dedicated swap partitions.
- The swap spaces are placed on the disk so the load which is on the I/O by the paging and swapping will spread over the system's bandwidth.

Swap-Space Location :



Location of swap-space

A swap space can reside in one of the two places –

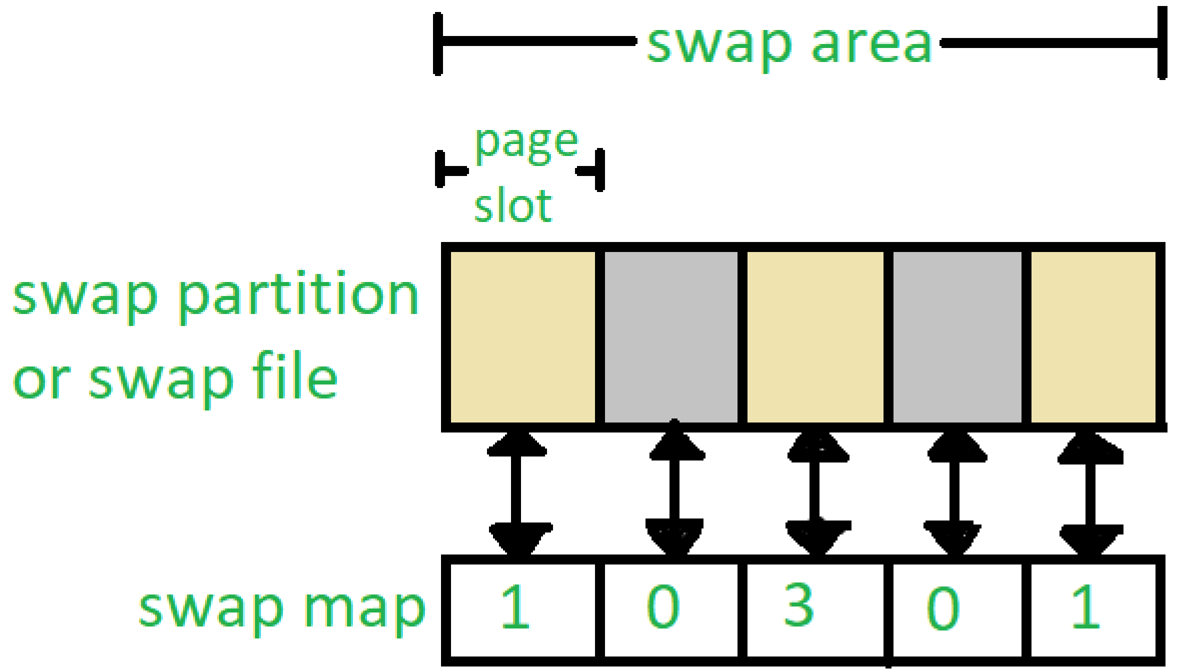
1. Normal file system
2. Separate disk partition

- Let, if the swap-space is simply a large file within the file system.
- To create it, name it and allocate its space **normal file-system** routines can be used. This approach, through easy to implement, is inefficient.
- Navigating the directory structures and the disk-allocation data structures takes time and extra disk access.
- During reading or writing of a process image, **external fragmentation** can greatly increase swapping times by forcing multiple seeks.

- There is also an alternate to create the swap space which is in a separate **raw partition**.
- There is no presence of any file system in this place.
- Rather, a swap space storage manager is used to allocate and de-allocate the blocks. from the raw partition.
- It uses the algorithms for speed rather than storage efficiency, because we know the access time of swap space is shorter than the file system.
- By this **Internal fragmentation** increases, but it is acceptable, because the life span of the swap space is shorter than the files in the file system.
- Raw partition approach creates fixed amount of swap space in case of the **disk partitioning**.
- Some operating systems are flexible and can swap both in raw partitions and in the file system space, example: **Linux**.

<u>**Swap-Space Management: An Example**</u> –

- The traditional UNIX kernel started with an implementation of swapping that copied entire process between contiguous disk regions and memory.
- UNIX later evolve to a combination of swapping and paging as paging hardware became available.
- In Solaris, the designers changed standard UNIX methods to improve efficiency. More changes were made in later versions of Solaris, to improve the efficiency.
- Linux is almost similar to Solaris system. In both the systems the swap space is used only for anonymous memory, it is that kind of memory which is not backed by any file.
- In the Linux system, one or more swap areas are allowed to be established.
- A swap area may be in either in a swap file on a regular file system or a dedicated file partition.



Data structure for swapping on Linux system

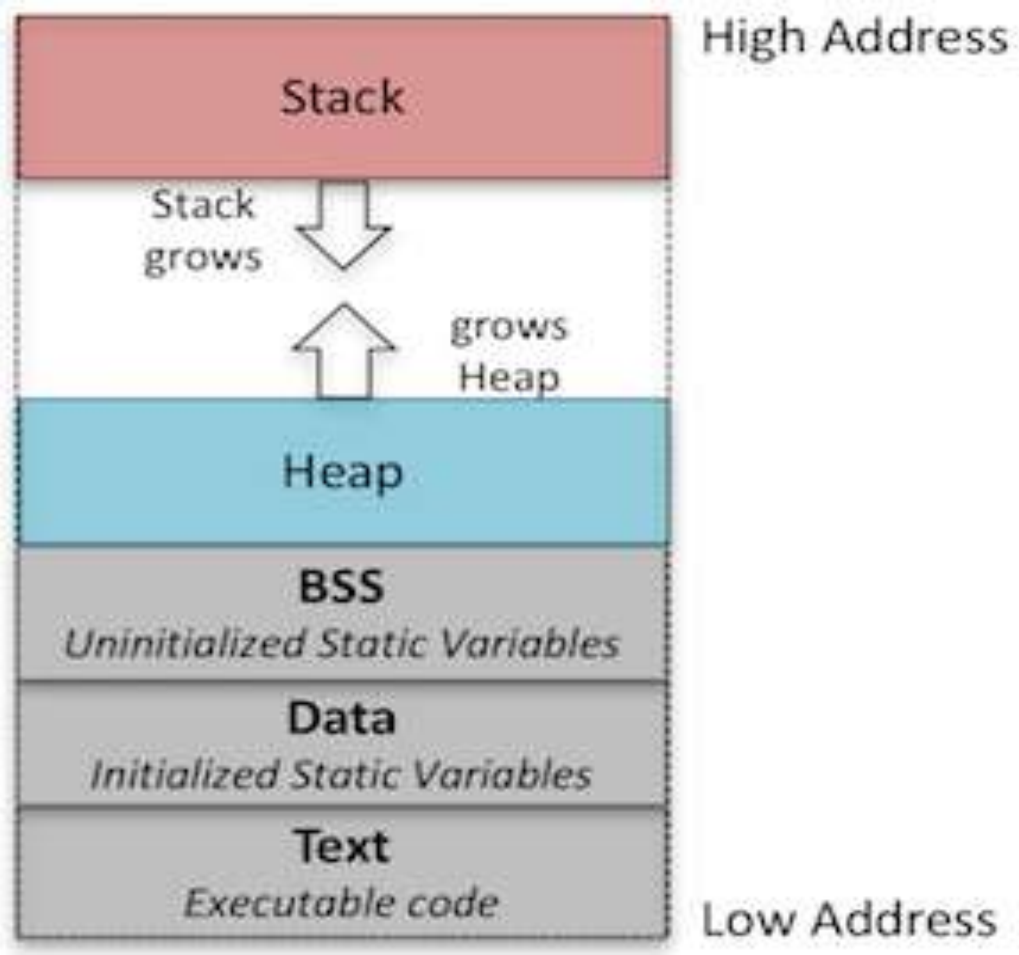
- Each swap area consists of 4-KB **page slots**, which are used to hold the swapped pages.
- Associated with each swap area is a **swap-map-** an array of integers counters, each corresponding to a page slot in the swap area.
- If the value of the counter is 0 it means page slot is occupied by swapped page.
- The value of counter indicates the number of mappings to the swapped page.
- For example, a value 3 indicates that the swapped page is mapped to the 3 different processes.

Dynamic Memory Allocation

- Memory allocation is a very important part of software development. When the program is loaded into the system memory, the memory region allocated to the program is divided into three broad regions: stack, heap, and code.
- **Stack region** is used for storing program's local variables when they're declared. Also, variables and arrays declared at the start of a function, including main, are allocated stack space. Stacks grow from high address to low address.
- Heap region is exclusively for dynamic memory allocation. Unlike stacks, heaps grow from low address to high address.

Code region can be further divided as follows:

- *BSS segment*: stores uninitialized static variables
- *Data segment*: stores static variables that are initialized
- *Text segment*: stores the program's executable instructions



Program Memory Layout