Operating System UNIT V

UNIT–V: File System

- **Different Types of Files**
- **Access methods**
- **Directory Structures**
- **Various Allocation methods**
- **Disk Scheduling and Management**
  - **Disk Scheduling Algorithms**
  - **Selecting a Disk Scheduling Algorithm**
    a. FCFS
    b. SSTF
    c. SCAN
    d. C–SCAN
    e. LOOK
    f. C–LOOK
- **Introduction to Distributed File System (*DFS*)**
- **Exam Questions**

❖ File Concept

- 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.
- Commonly, files represent **programs** (both source and object forms) and **data.**
- Data files may be **numeric, alphabetic, alphanumeric, or binary**.
- In general, a file is a sequence of **bits, bytes, lines, or records**

- The information in a file is defined by its creator.
- Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on.
- 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 functions, each of which is further organized as declarations followed by executable statements.
- An **executable file** is a series of code sections that the loader can bring into memory and execute

❖ *File Attributes*

- **Name**
  - Only information kept in human-readable form
- **Identifier**
  - Unique tag (number) identifies file within file system
- **Type**
  - Needed for systems that support different types
- **Location**
  - Pointer to file location on device
- **Size**
  - Current file size
- **Protection**
  - Controls who can do reading, writing, executing
- **Time, date, and user identification**
  - Data for protection, security, and usage monitoring

- **Information about files**
  - Kept in the directory structure
  - Maintained on the disk

Figure 11.1 A file info window onMac OS X.

File Operations

- A file is an abstract data type.
- To define a file properly, we need to consider the operations that can be performed on files.
- Operating system can provide system calls to create, write, read, reposition, delete, and truncate files.
  - **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.
  - Both the read and write operations use this same pointer, saving space and reducing system complexity.
- **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 Types – Name, Extension

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 com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

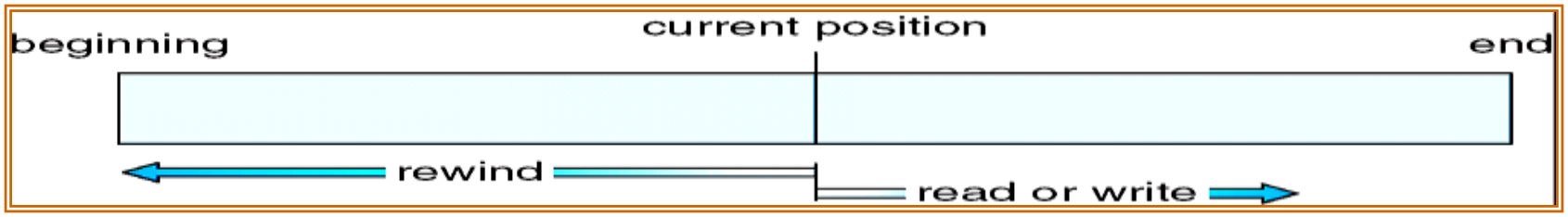
❖Access Methods

- 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. while others 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.
- 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 is based on tape model of a file and which is depicted in Figure 11.4



➢Direct Access

- Another method is **direct access (or relative access).**
- **A file is made up** of fixed-length **logical records that allow programs to read and write records** rapidly in no particular order.
- 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.
- 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

Simulation of Sequential Access on a Direct–Access File

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

➢Indexed access method

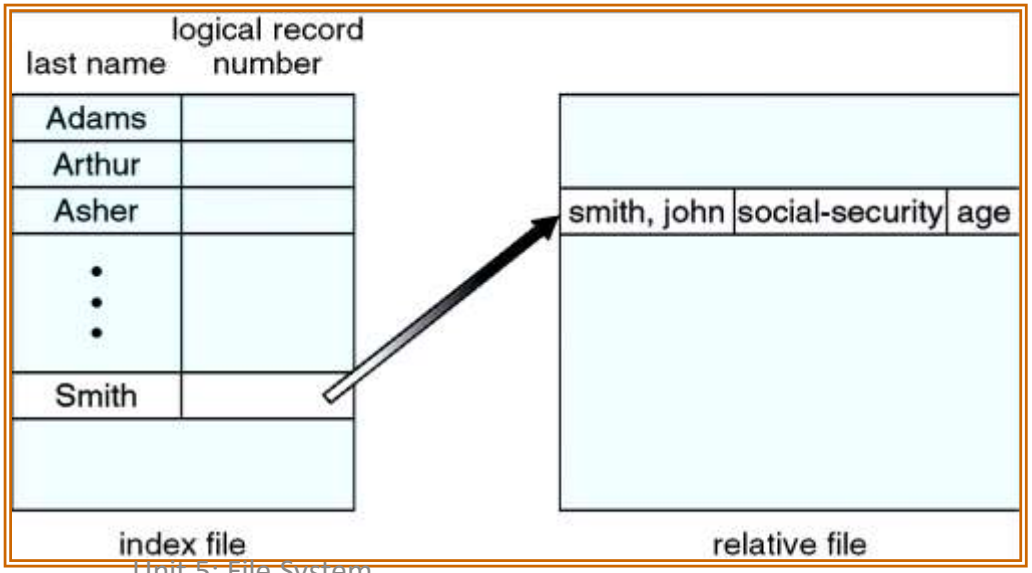
•In indexed accessed file contains a collection of blocks and pointers are assigned various blocks

•To find a record in a file , we first search the index and then use the pointer to access the file directory and to find the desired record.

•With large files, the index file itself may become too large to be kept in memory.

•One solution is to create an index for the index file.

•The primary index file contains pointers to secondary index files, which point to the actual data items.



❖ Directory and Disk Structure

- A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control. For example, a disk can be **partitioned** into quarters, and each quarter can hold a separate file system.

- Afile system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a **volume**.

- Each volume can be thought of as a virtual disk. Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.

- Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory** or **volume table of contents**.

- The device directory (more commonly known simply as the **directory**) records information—such as name, location, size, and type—for all files on that volume. Figure 11.7 shows a typical file-system organization.

A Typical File–System Organization

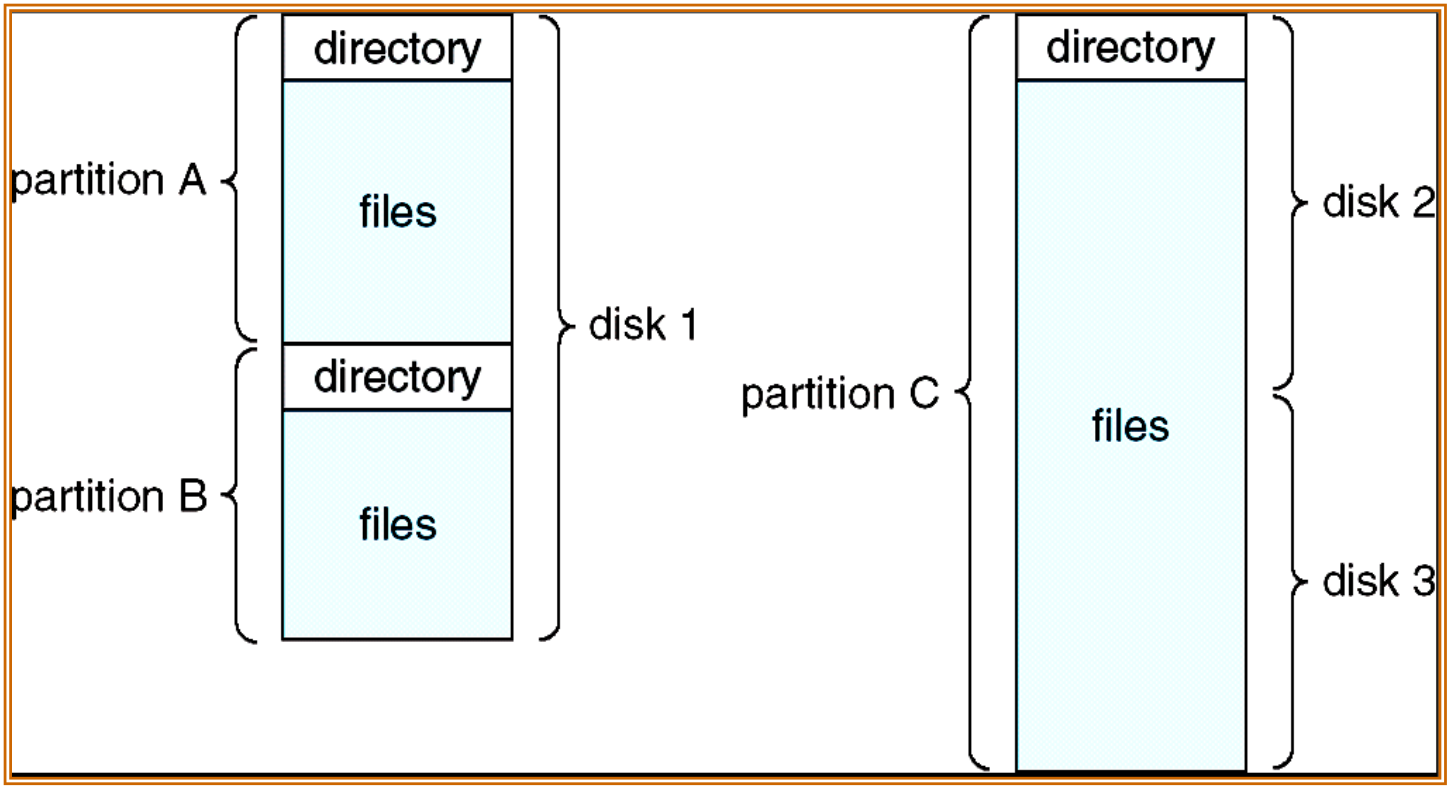


Figure 11.7 A typical file-system organization.

➢ Storage Structure

- General-purpose computer system has multiple storage devices, and those devices can be sliced up into volumes that hold file systems.

- Computer systems may have zero or more file systems, and the file systems maybe of varying types.

- For example, a typical Solaris system may have dozens of file systems of a dozen different types, as shown in the file system list in Figure 11.8.

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

Figure 11.8 Solaris file systems.

- **tmpfs**—a “temporary” file system that is created in volatile main memory and has its contents erased if the system reboots or crashes

- **objfs**—a “virtual” file system (essentially an interface to the kernel that looks like a file system) that gives debuggers access to kernel symbols
- **ctfs**—a virtual file system that maintains “contract” information to manage which processes start when the system boots and must continue to run during operation
- **lofs**—a “loop back” file system that allows one file system to be accessed in place of another one
- **procfs**—a virtual file system that presents information on all processes as a file system
- **ufs, zfs**—general-purpose file systems

❖ Directory Overview

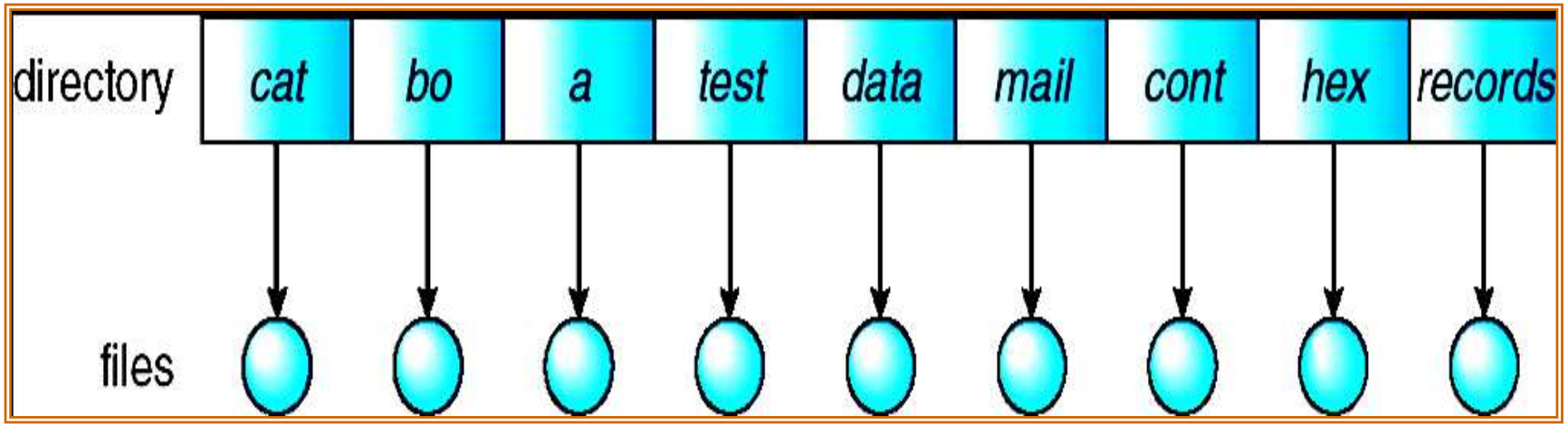
- The directory can be viewed as a symbol table that translates file names into their directory entries. If we take such a view, we see that the directory itself can be organized in many ways.

➢ ***Operations Performed on Directory***

- **Search for a file**. We need to be able to search a directory structure to find the entry for a particular file.
- **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.
- **Traverse the file system. We** may wish to access every directory and every file within

Single-Level Directory

- A single directory for all users
- The simplest directory structure is the single-level directory.
- All files are contained in the same directory, which is easy to support and understand



Advantages

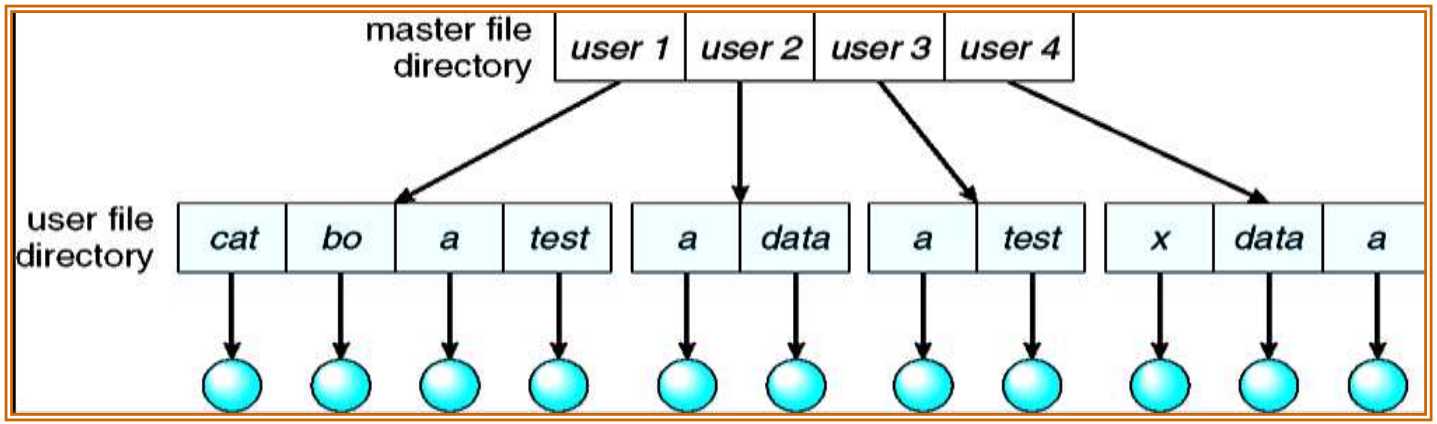
- Implementation is very simple.
- If the sizes of the files are very small then the searching becomes faster.
- File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

- We cannot have two files with the same name.
- The directory may be very big therefore searching for a file may take so much time.
- Protection cannot be implemented for multiple users.
- There are no ways to group same kind of files.
- 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

- Separate directory for each user
- In the two-level directory structure, each user has his own user file directory (UFD).
- The UFDs have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user (Figure 11.10).



Advantages:

•Users can have the same file name.

•Searching is easier as there is user-grouping.

Disadvantages:

•Two files of the same type cannot be grouped together.

•Users cannot share the files with each other

➢Tree-Structured Directories

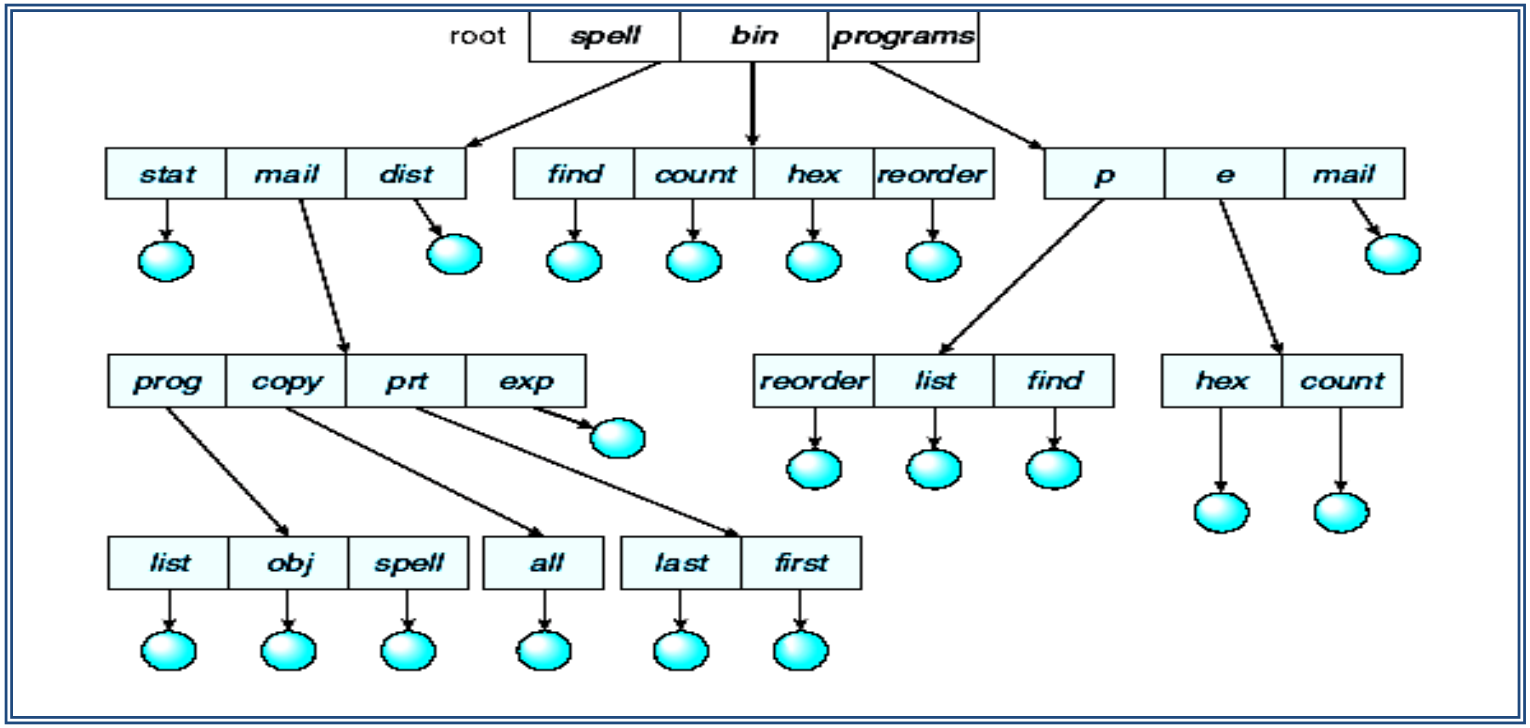
- In Tree structured directory system, any directory entry can either be a file or sub directory.

- 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.

- Path names can be of two types: ***absolute and relative***.
- An **absolute path** name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A **relative path** name defines a path from the current directory.



- **Advantages:**
  - Efficient searching
  - Grouping capability
-

- **Disadvantages:**

•File may get saved into multiple directories.

•Files cannot be shared.

•Accessing a file may lead to different directories

➢Acyclic-Graph 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.

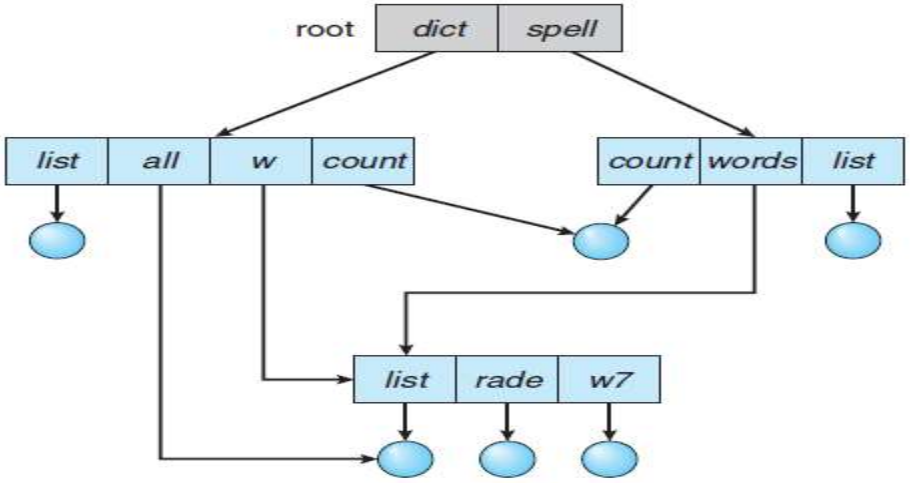


Figure 11.12 Acyclic-graph directory structure.

Advantages:

•Files can be shared.

•It makes searching easier as there are different paths.

•

Disadvantages:

•In the case of soft link, if the file is deleted, then only a dangling pointer is left.

•In the case of hard link, if we delete a file, then we also need to remove the reference connected to it.

•If the files are shared using a link, then deleting them might cause problem

❖ Various 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.

➢ *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.

Contiguous Allocation of Disk Space

Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- **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:**

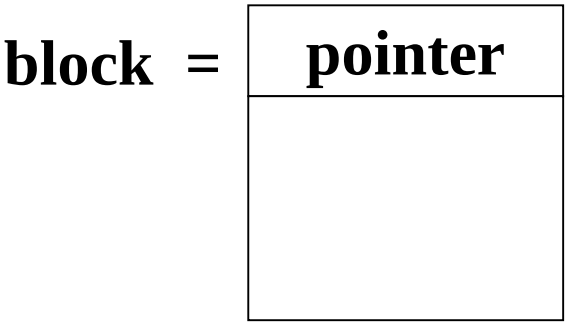
  •This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

  •Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

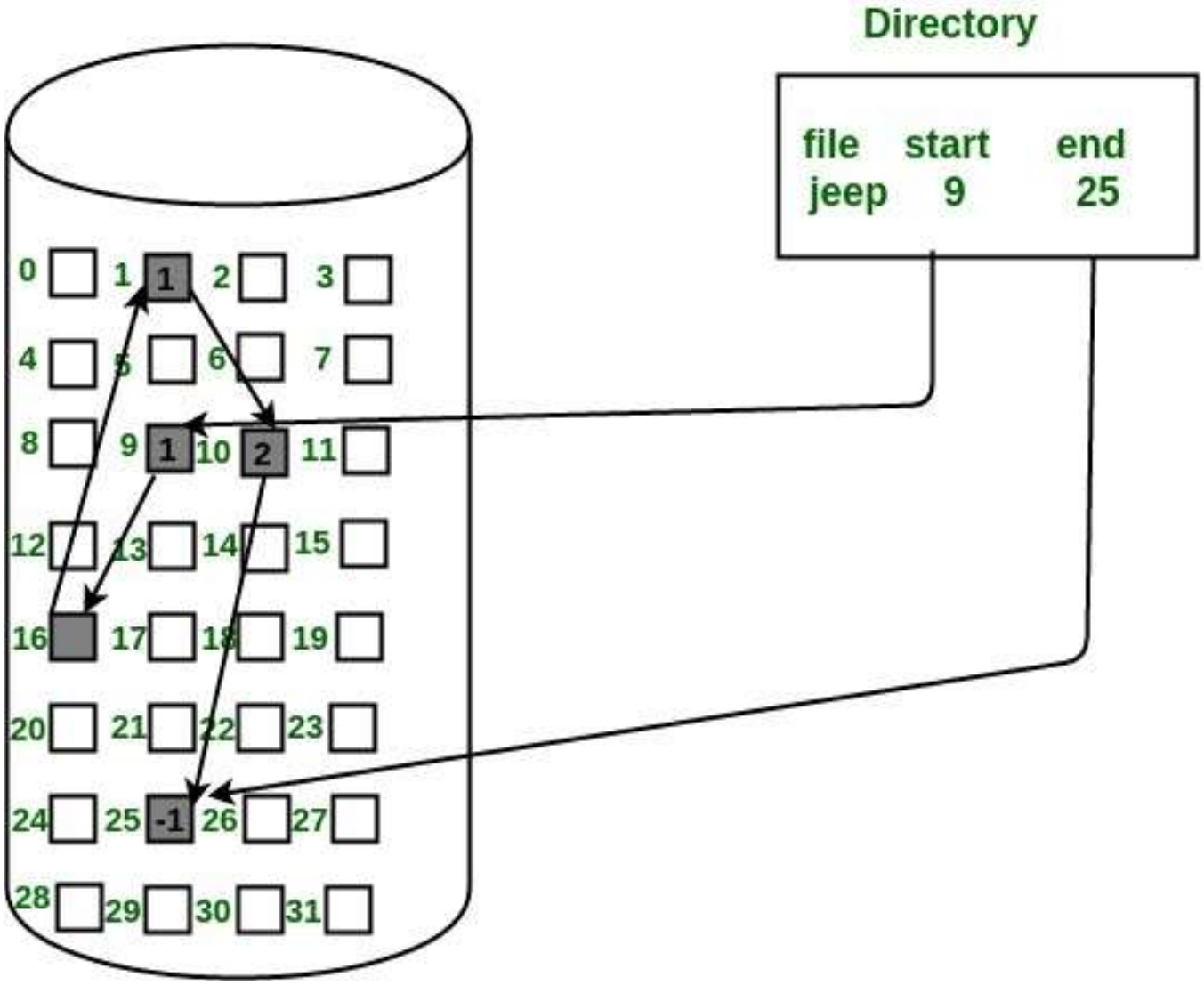
➢ *Linked Allocation*

- 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.



➢ *Linked 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.

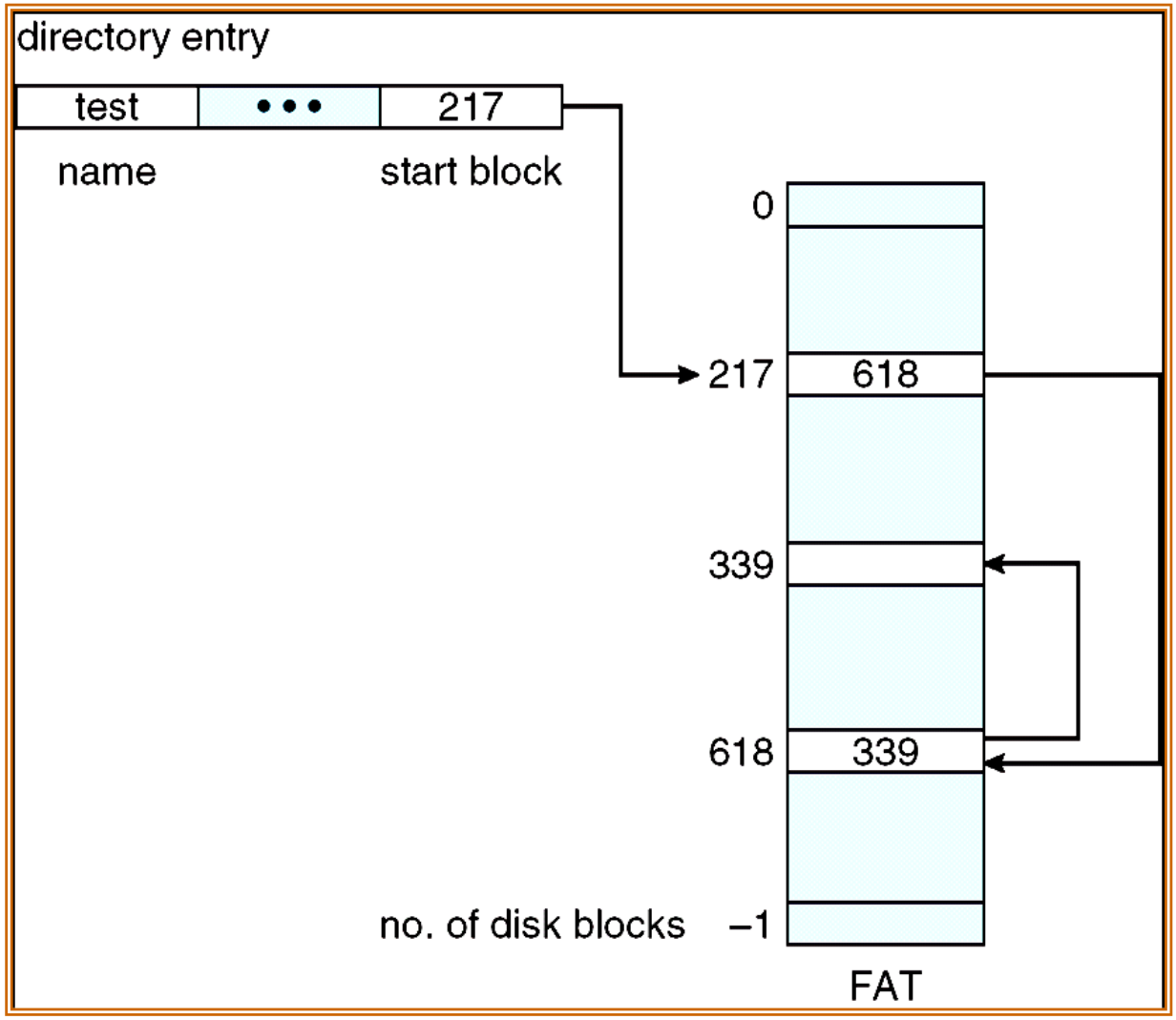
•Pointers required in the linked allocation incur some extra overhead.

➢ Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.

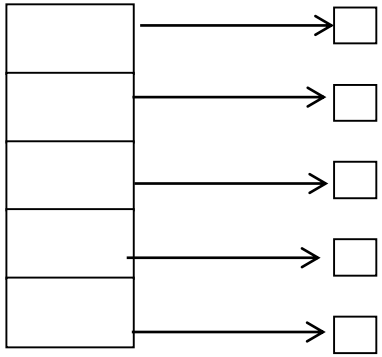
- Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:

File–Allocation Table



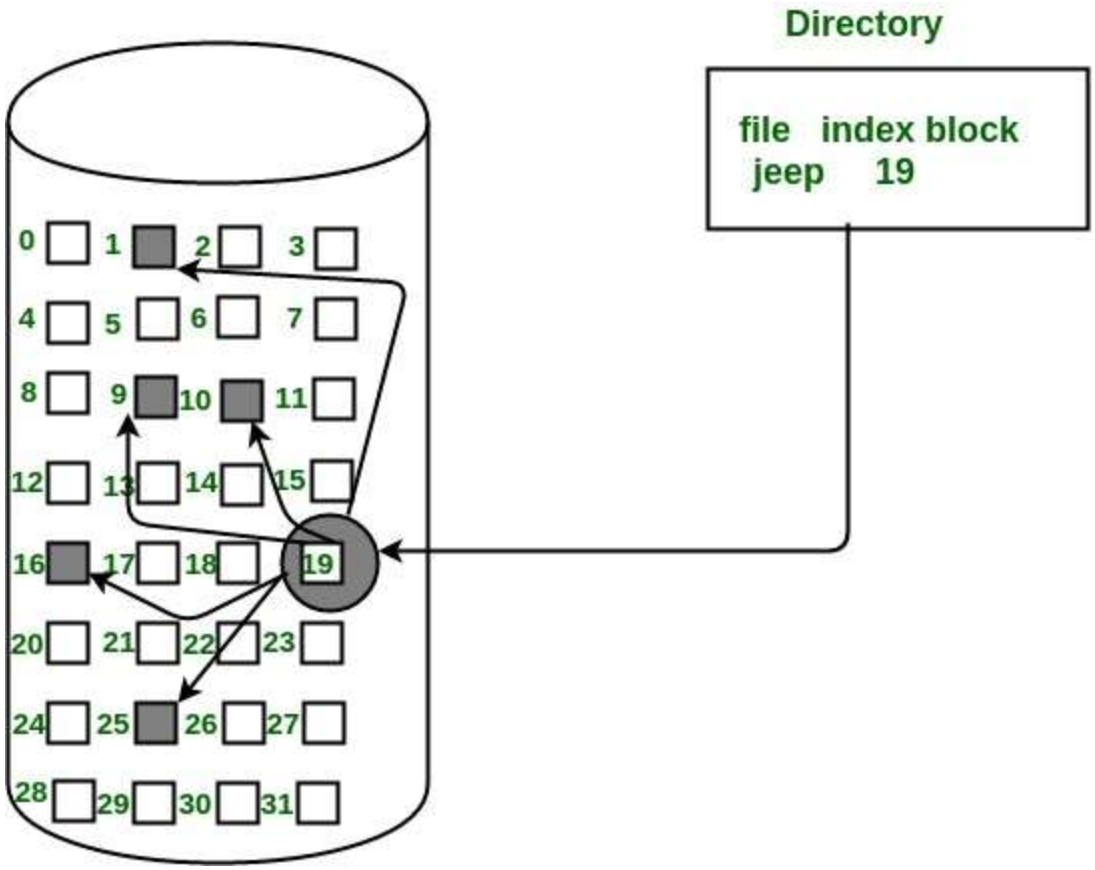
Indexed Allocation

- Brings all pointers together into the *index block*.
- Logical view.



index table

Example of 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.

- For files that are very large, single index block may not be able to hold all the pointers.

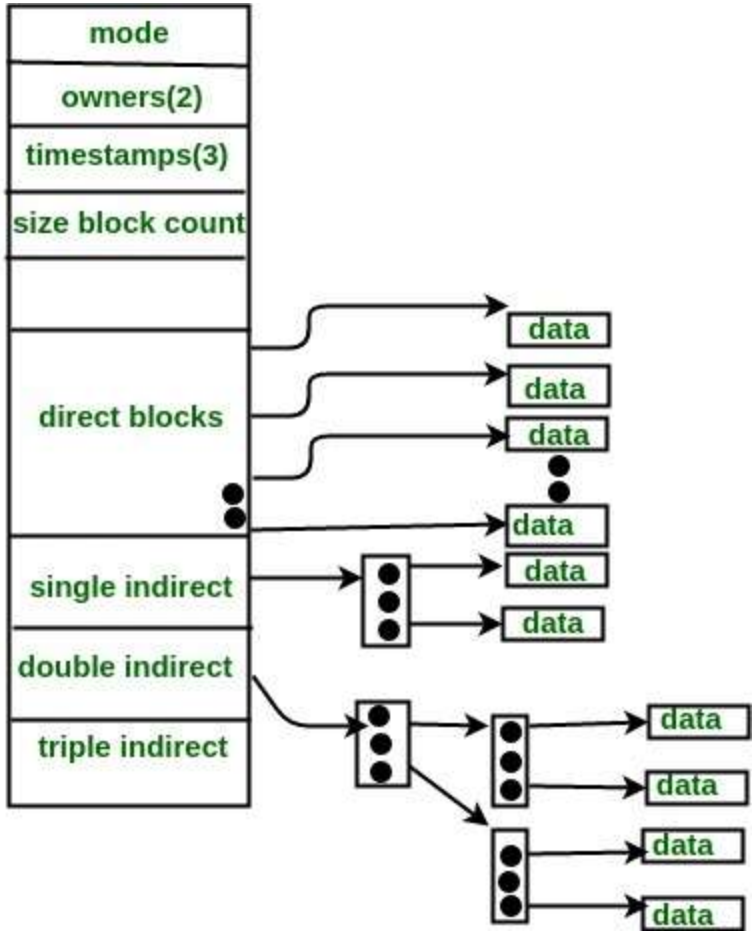
  Following mechanisms can be used to resolve this:

  1.Linked scheme: This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.

  2.Multilevel index: In this policy, a first level index block is used to point to the second level index blocks which inturn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.

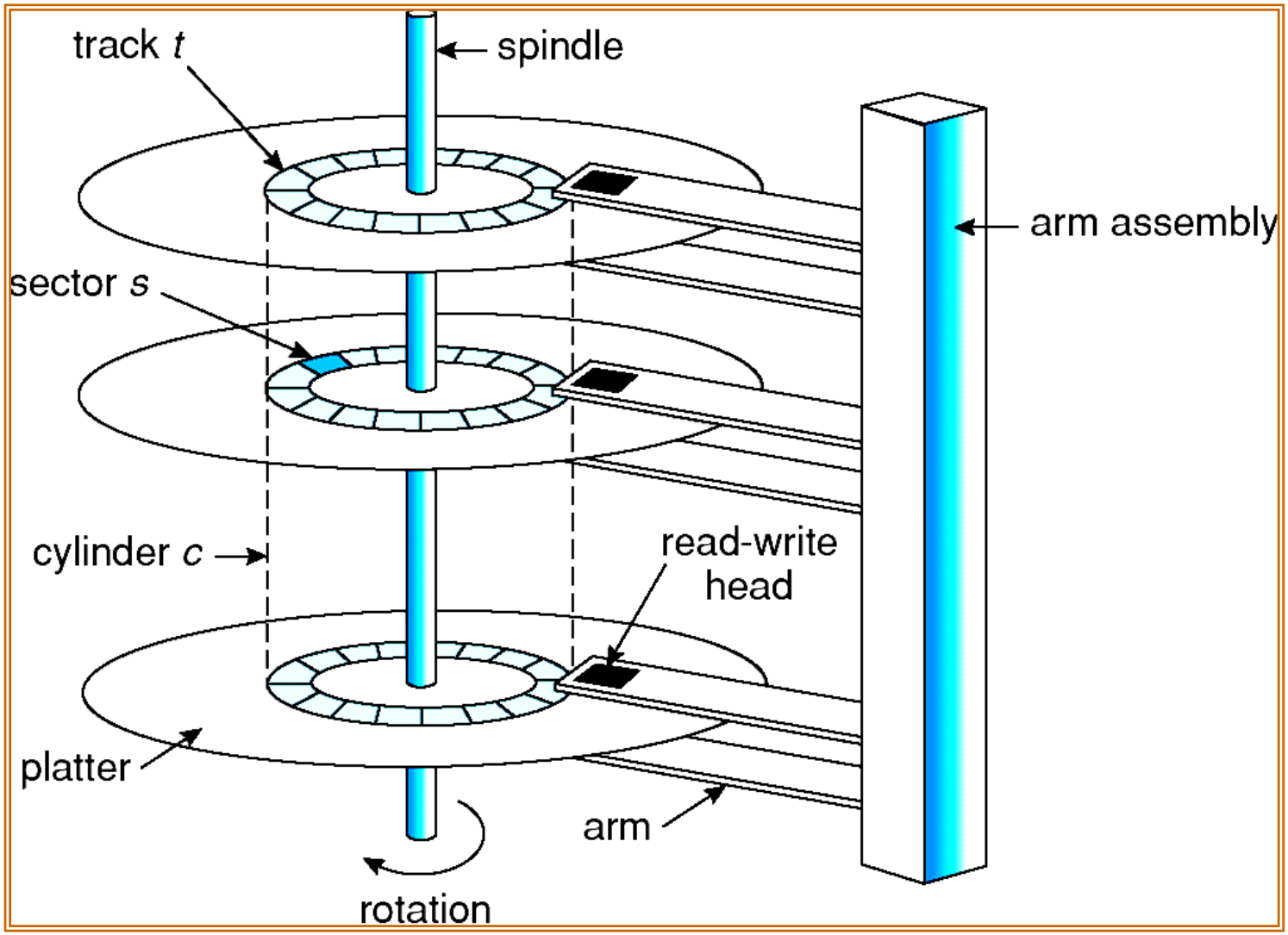
3. Combined Scheme:

- In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file *as shown in the image below.*
- The first few of these pointers in Inode point to the **direct blocks** i.e the pointers contain the addresses of the disk blocks that contain data of the file.
- The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect.
- **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data.
- Similarly, **double indirect blocks** do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.


mode
owners(2)
timestamps(3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

❖Disk Scheduling and Management

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a **fast access time** and **disk bandwidth**.
- Access time has two major components
  - ***Seek time*** is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - ***Rotational latency*** is the additional time waiting for the disk to rotate the desired sector to the disk head.
- **Minimize seek time**
- Seek time ≈ seek distance
- **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.



Moving Head Disk Mechanism

➢ Disk Scheduling – Example 1

1. Suppose that a disk drive has 200 cylinders, numbered 0 to 199. The drive current position of Read/Write head is: 50. The queue of pending requests in FIFO order is 82,170,43,140,24,16,190 . Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following *disk scheduling* algorithms?

a. FCFS

b. SSTF

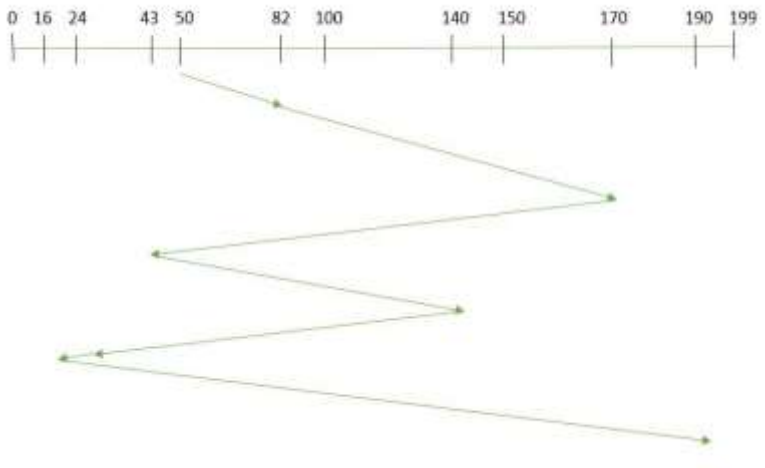
c. SCAN

d. C–SCAN

e. LOOK

f. C–LOOK

- **FCFS:FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.



Advantages:

- ✓Every request gets a fair chance
- ✓No indefinite postponement

Disadvantages:

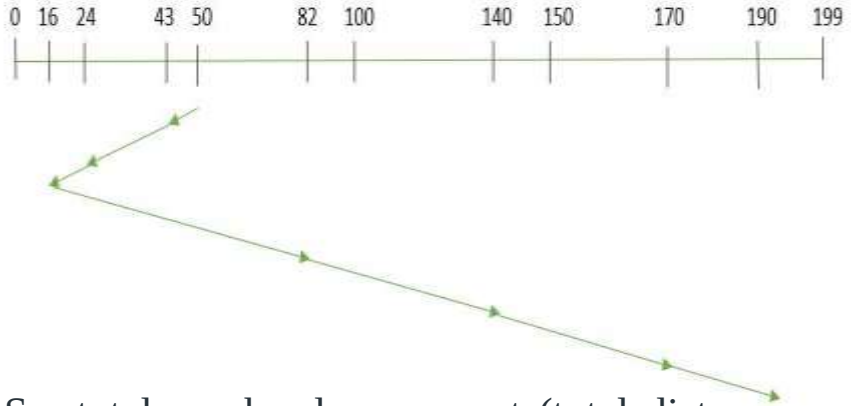
- ✓Does not try to optimize seek time
- ✓May not provide the best possible service

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

2.**SSTF:** In SSTF (Shortest Seek Time First), requests having 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 system.Let us understand this with the help of an example.



So, 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

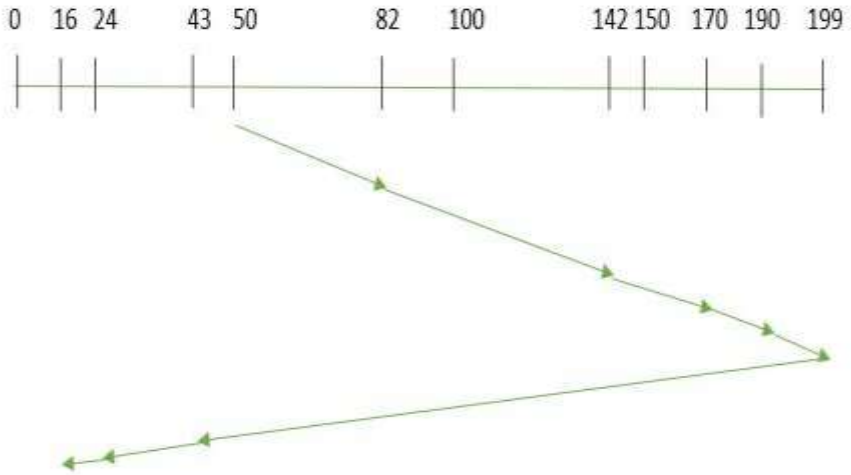
Advantages:

- ✓Average Response Time decreases
- ✓Throughput increases

Disadvantages:

- ✓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
- ✓High variance of response time as SSTF favors only some request

- **3. SCAN:** In 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.



Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50)+(199-16) =332

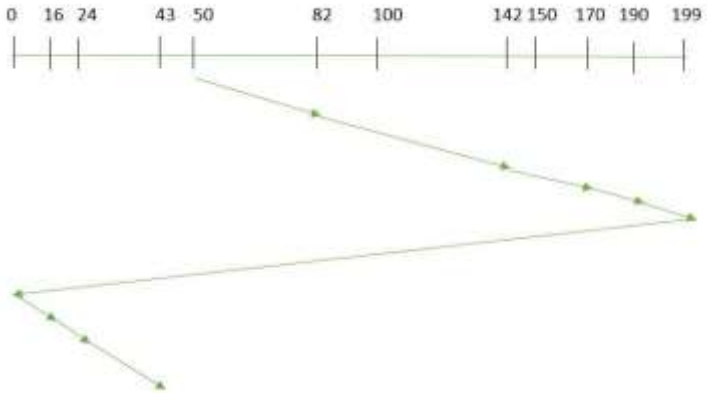
Advantages:

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

Disadvantages:

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

- 4.CSCAN: 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:

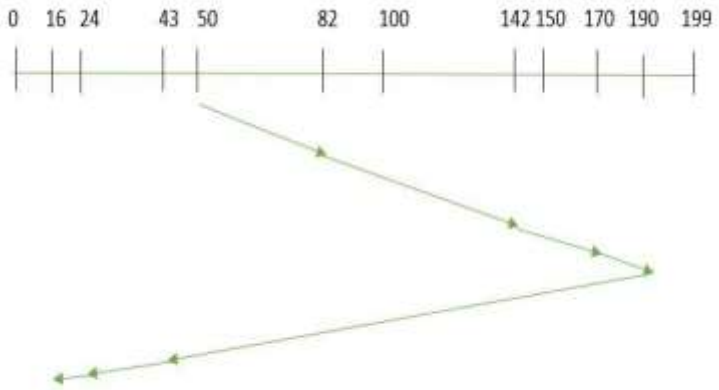
✓Provides more uniform wait time compared to SCAN

so, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50)+(199-0)+(43-0)

=391

5. 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

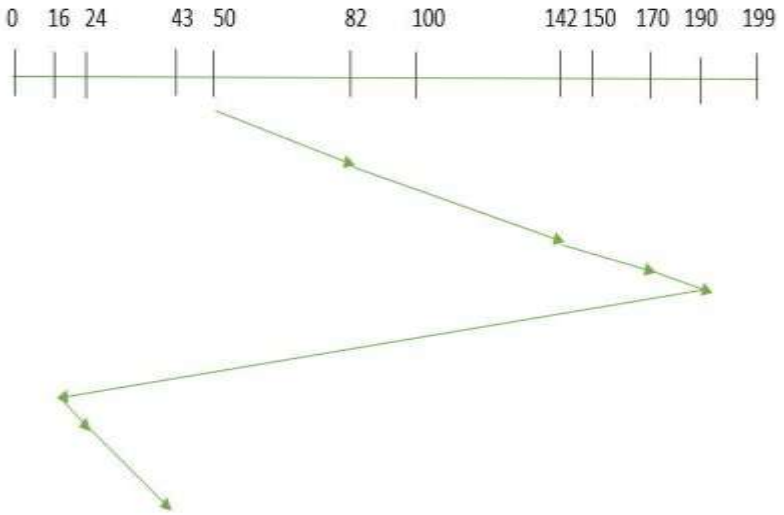


So, the total overhead movement (total distance covered by the disk arm) is calculated as:

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

=314

6. 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.



So, the total overhead movement (total distance covered by the disk arm) is calculated as:

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

=341

➢ Disk Scheduling – Example 2

1. Suppose that a disk drive has 200 cylinders, numbered 0 to 199. The drive is currently serving a request at cylinder 53, and the previous request was at cylinder 45. The queue of pending requests in FIFO order is 98, 183, 37, 122, 14, 124, 65, 67. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following *disk scheduling* algorithms?

a. FCFS

b. SSTF

c. SCAN

d. C–SCAN

e. LOOK

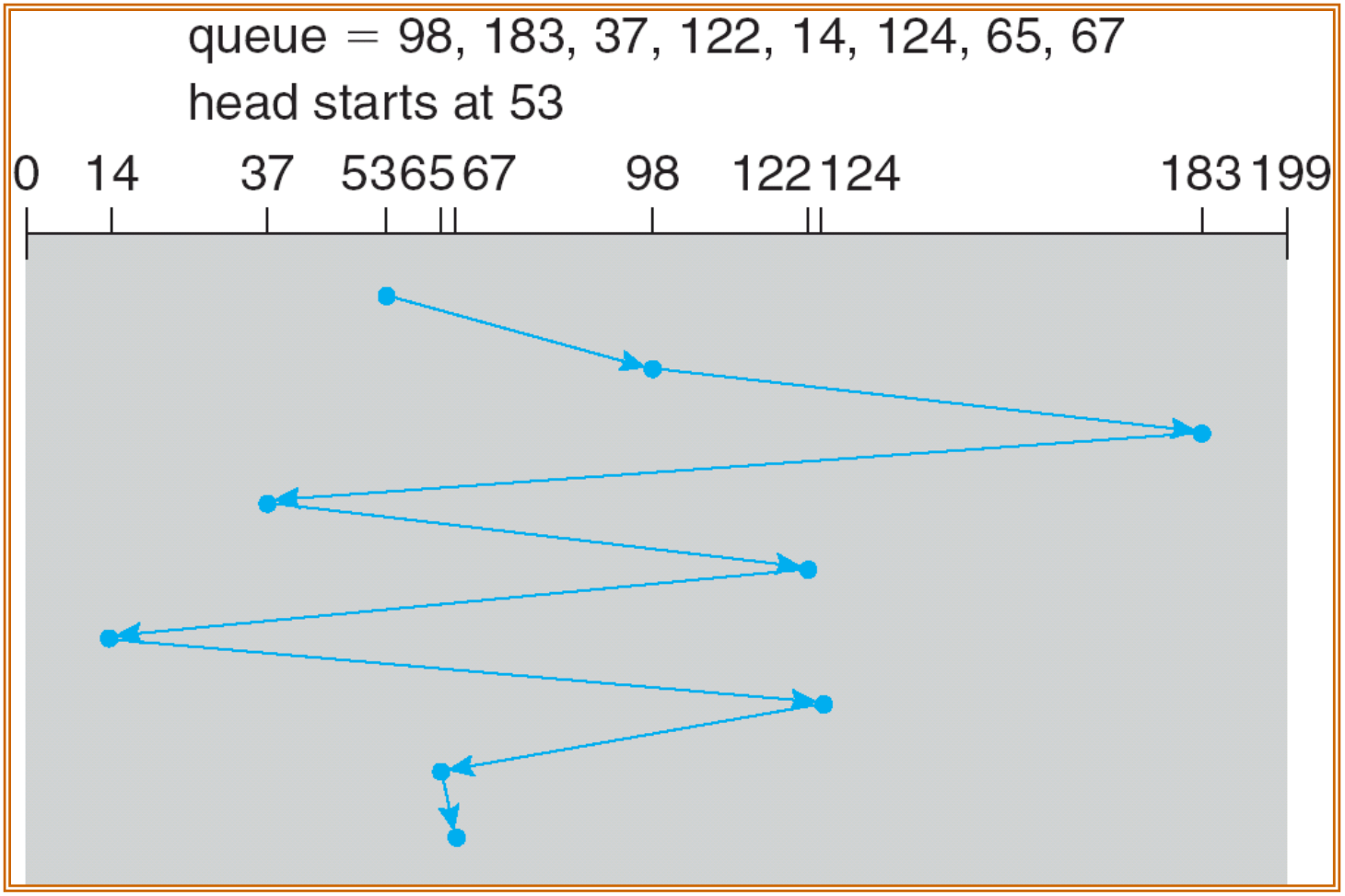
f. C–LOOK

➢ FCFS Disk Scheduling

- Request queue (0-199).

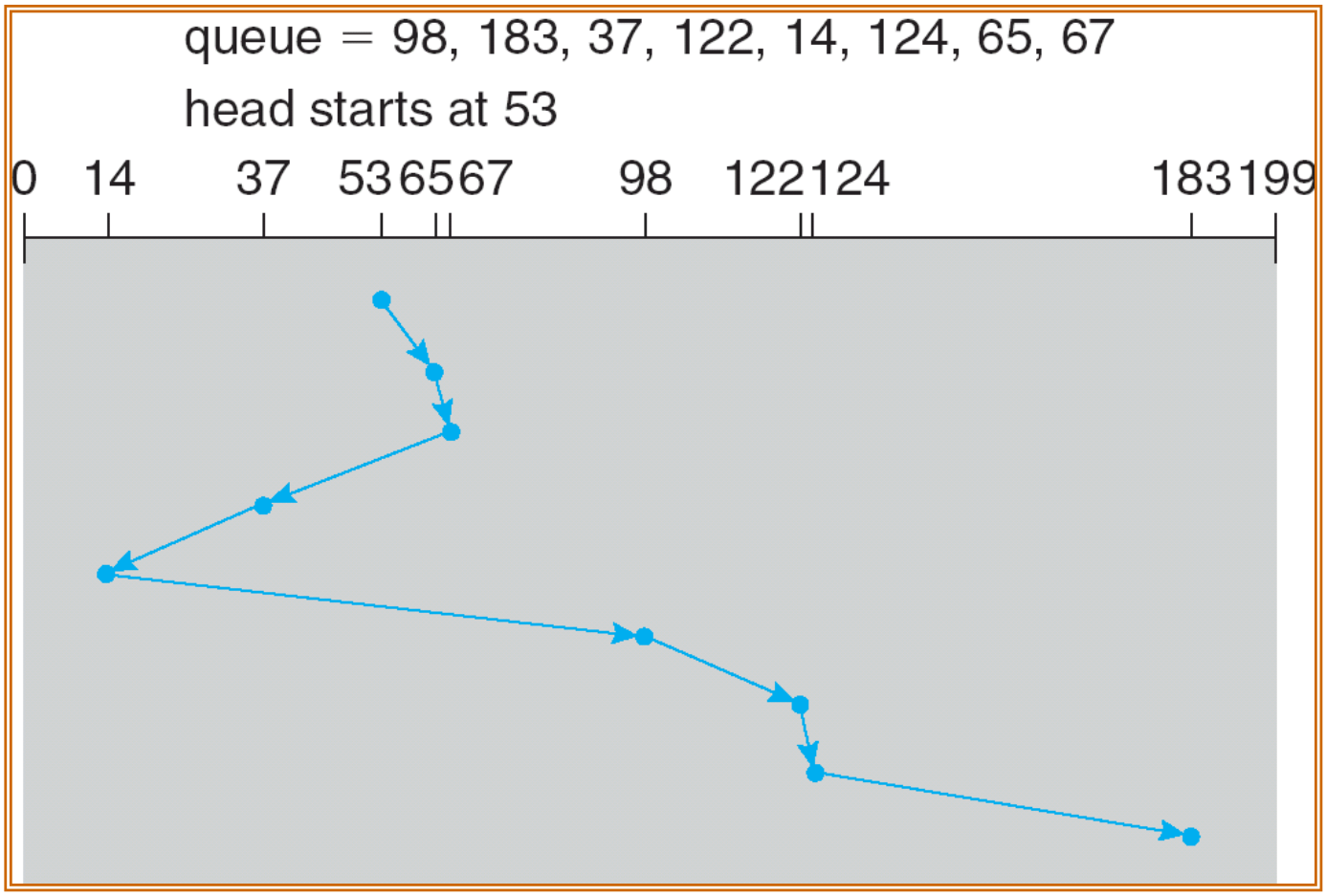
  98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53



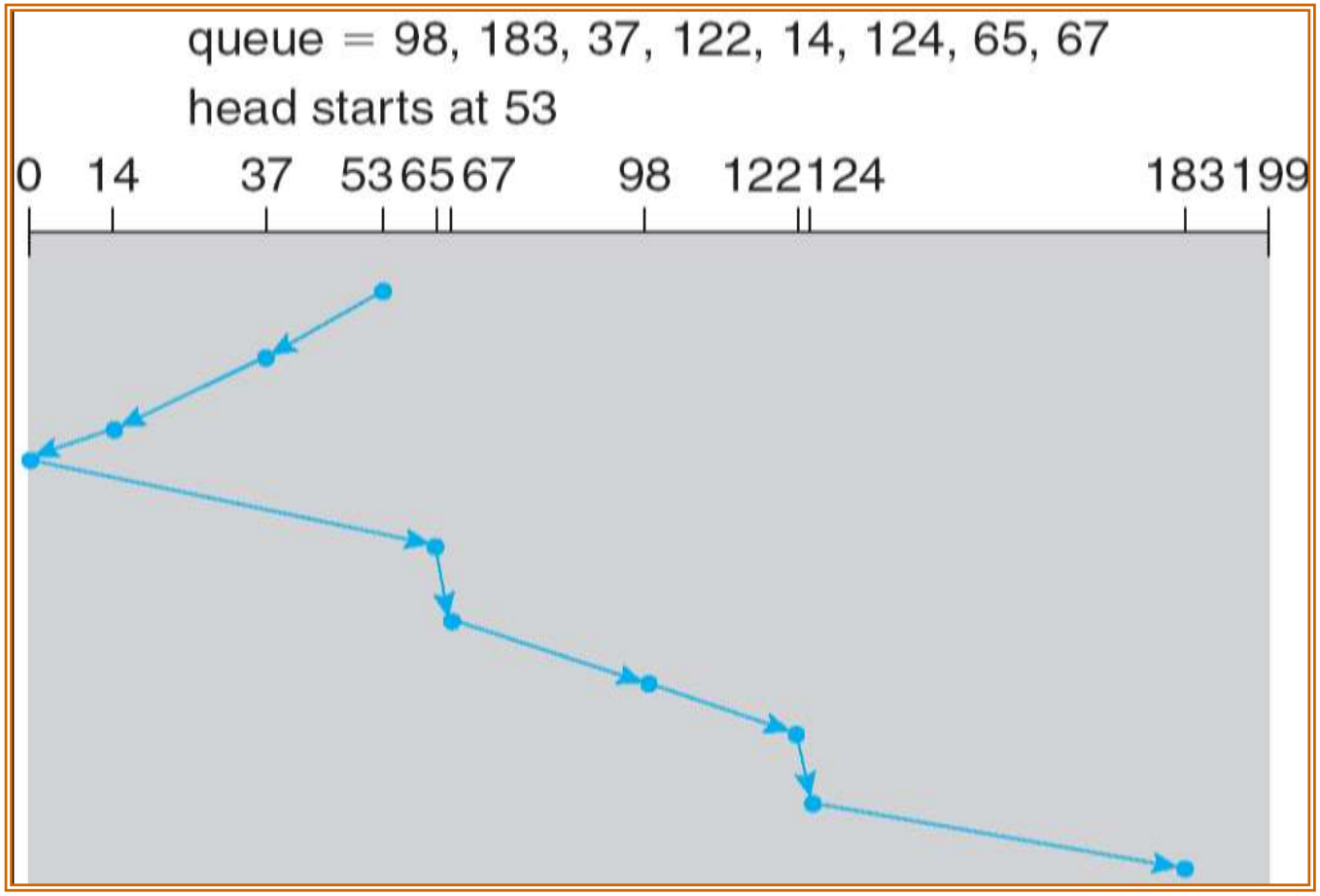
➢ SSTF Disk Scheduling Algorithm

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.



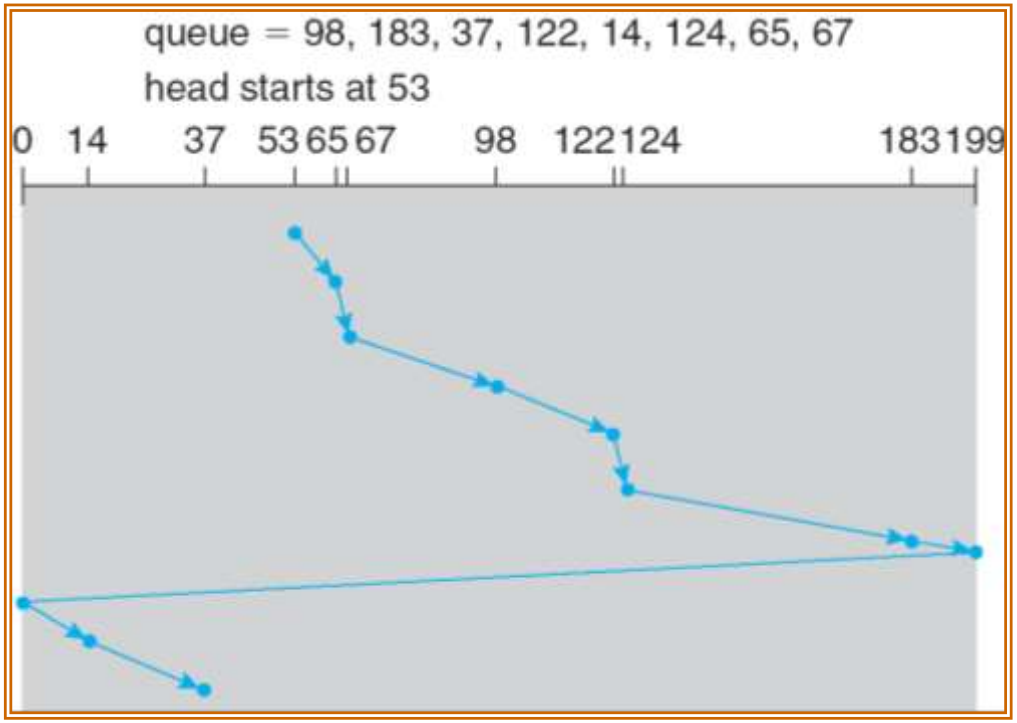
➢ SCAN / Elevator Disk Scheduling Algorithm

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.



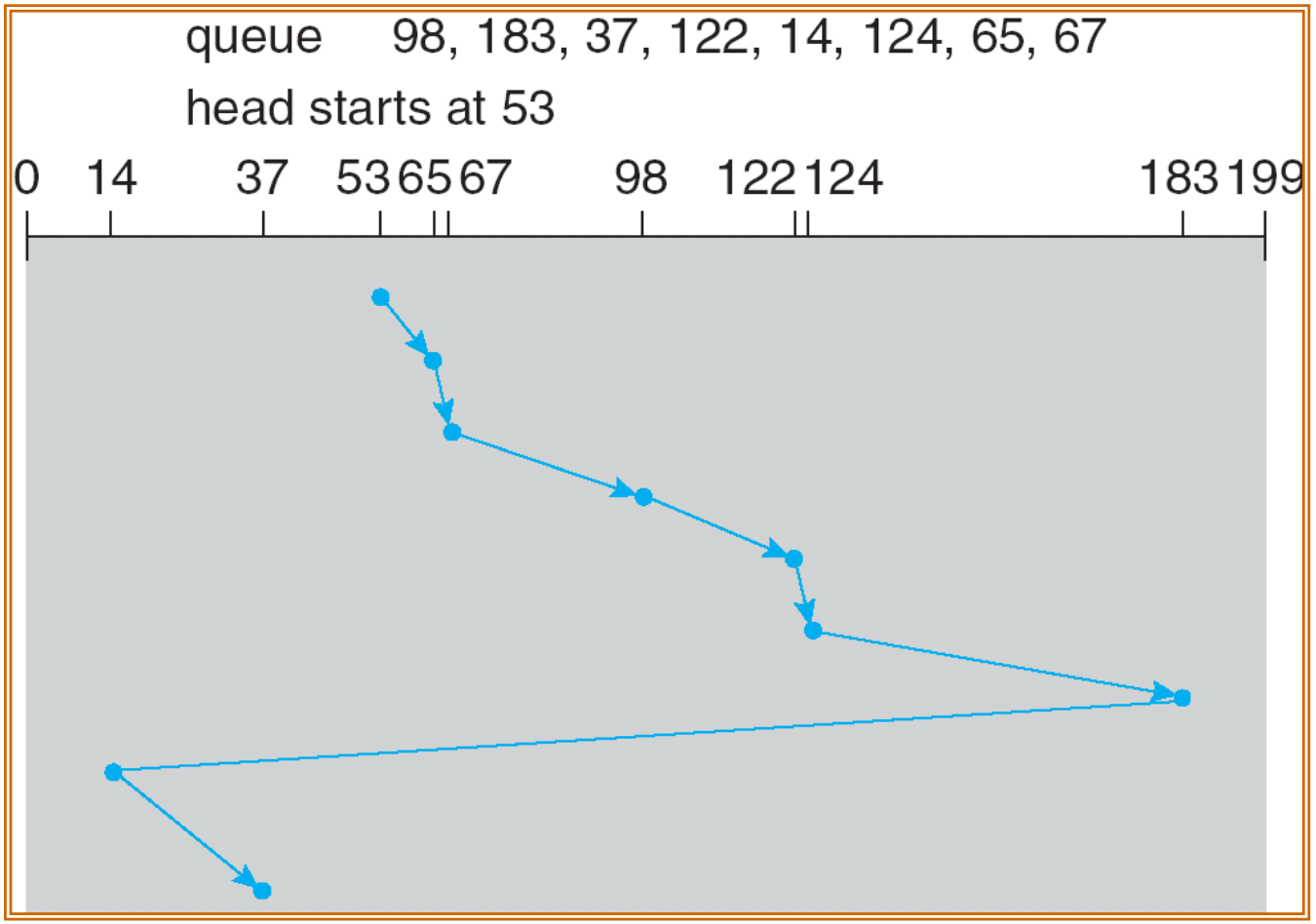
➢ C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



➢ C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



➢ *Disk Scheduling – Example 3*

2. Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following *disk scheduling* algorithms?

a. FCFS

b. SSTF

c. SCAN

d. C–SCAN

e. LOOK

f. C–LOOK

❖ Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

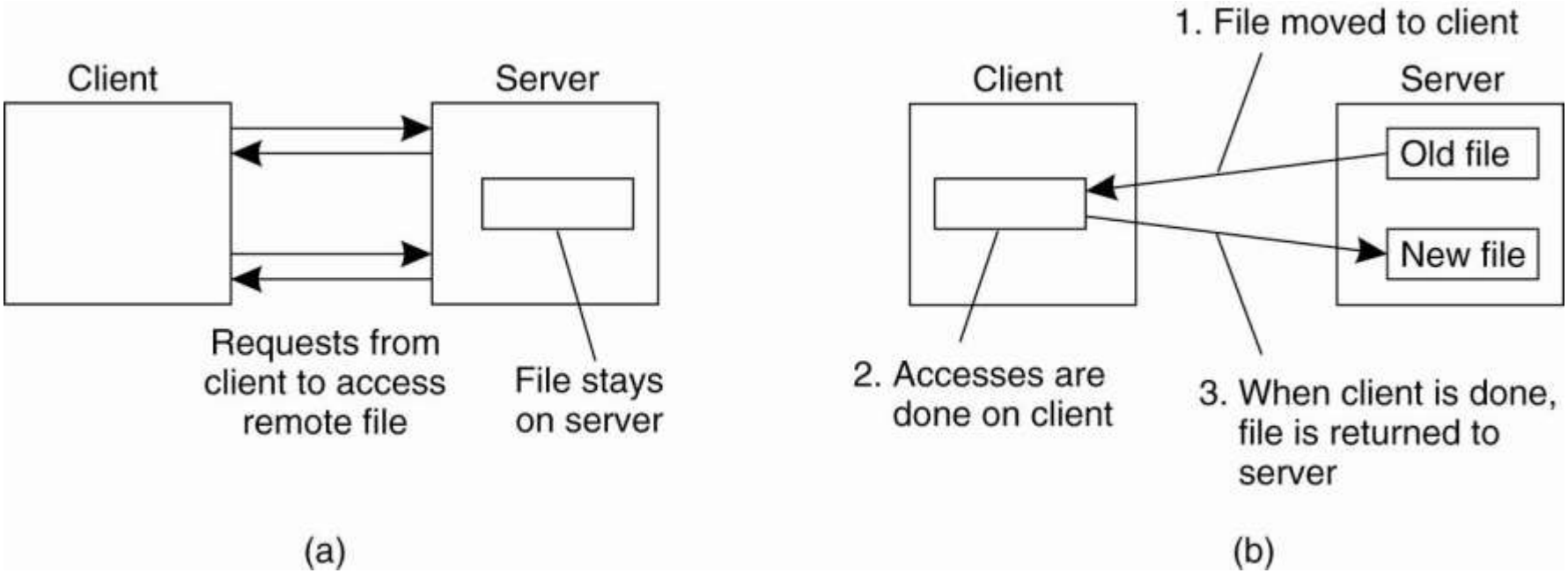
❖ Distributed / Network File System (DFS) / (NFS)

- Any File System that allows Access to Files from Multiple Hosts Sharing via a Computer Network.
  - Makes it Possible for Multiple Users on Multiple Machines to Share Files and Storage Resources.
- Client Nodes do not have Direct Access to the underlying Block Storage but interact over the Network using a Protocol.
  - Makes it possible to Restrict Access to the File System Depending on Access lists or Capabilities on both the Servers and the Clients, Depending on How the Protocol is Designed.
- Include facilities for Transparent replication and Fault tolerance.
  - When a limited no. of Nodes in a File System go offline, the System Continues to Work without Any Data loss.

DFS / NFS

Requirements

- Heterogeneity
- Replication
- Transparency
- Consistency
- Concurrency
- Fault tolerance
- Security
- Efficiency



(a) The remote access model **(b) The upload / download model**

Exam Questions

1. Describe the various *attributes of files*.
2. Write short note on *file sharing*.
3. Explain *Directory Structure*.
4. List the *file allocation methods*.
5. Explain the various *file allocation methods*, their *advantages and disadvantages*.
6. Discuss various *Disk Scheduling algorithms*.
7. Explain briefly about *disk scheduling algorithms* and its *types*.