

Operating Systems with Linux

(MCA-105)

Unit – 4


by

Dr. Sunil Pratap Singh

(Associate Professor, BVICAM, New Delhi)

2023


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.1



File

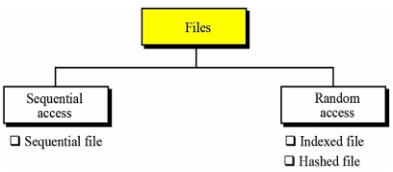
- A **file** is a collection of related data stored on mass storage (e.g., disk or tape).
 - The data is subdivided into **records** (e.g., student information).
 - Each record contains a number of **fields** (e.g., roll number, name).
 - One (or more) field is the **key** field (e.g., roll number).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.2



File Organizations

- A **file organization** refers to the way records are arranged on a storage device.
- How best the files be arranged for easy of access?



```

graph TD
    Files[Files] --> Sequential[Sequential access]
    Files --> Random[Random access]
    Sequential --- S["Sequential file"]
    Random --- R["Indexed file"]
    Random --- H["Hashed file"]
            
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.3

Sequential Files

- A sequential file is one in which records can only be accessed one after another from beginning to end.
- This file organization is the simplest way to store and retrieve records of a file.
- In this file, data records are stored in **some specific sequence**, e.g., order of arrival, value of key field, etc.

Record 1	Record 2	Record 3
001 Manoj 22	002 Aman 21	003 Mukesh 21
↑ Key	↑ Key	↑ Key

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.4

Sequential Files

- The records of a sequential file cannot be accessed at random, i.e., to **access the n^{th} record**, one must **traverse the preceding $(n-1)$ records**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.5

Sequential File Organization

- In sequential file organization, the actual storage of records might or might not be sequential:
 - On a tape, it usually is.
 - On a disk, it might be distributed across sectors and the operating system would use a linked list of sectors to provide the illusion of sequentially.
- **Editors and compilers usually access files in this fashion.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.6

Sequential File Organization

- **Advantages**
 - Easy to handle
 - Involves no overhead
 - Can be stored on tapes as well as disks
- **Disadvantages**
 - Records can only be accessed in sequence
 - Time consuming

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.7

Indexed File

- To access a record in a file randomly, we need to know the address of the record.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.8

Indexed File: Logical View

- An **indexed file** is made of a **data file** (also known as **relative file**), which is a sequential file, and an **index**.
- The index itself is a **very small file** with only two fields: **the key of the sequential file and the address of the corresponding record on the disk**.
- The index is sorted based on the key values of the data files.

Index		Data file			
Key	Addr.	Addr.	Key	Name	Distance
045128	300	000	379452	Mary Dook	1432.45
079918	001	001	079918	Sarah Terry	1400.22
112107	002	002	112107	Ilyana Deviana	11.48
166702	003	003	166702	Thery Eagle	14323.00
...
378845	007	007	378845	John Carter	7234.01
...
146212	009	009	146212	Evan Nigo	15121.10
379452	000	006	045128	Shaul Feldman	87922.95

Accessing indexed file: 166702 → 166702 Thery Eagle 14323.00
Extracted record

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.9

Indexed File: Accessing a Record

- Accessing a record in the file requires following steps:
 - The entire index file is loaded into main memory (the file is small and uses little memory).
 - The index entries are searched, using an efficient search algorithm such as a binary search, to find the desired key.
 - The address of the record is retrieved.
 - Using the address, the data record is retrieved and passed to the user.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.10

Inverted File

- One of the advantages of indexed files is that we can have **more than one index**, each with a different key.
- This type of indexed file is usually called an **inverted file**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.11

Direct File (Hashed File)

- A **hashed file** uses a **mathematical function** to **map the key to the address**.
- The user gives the key, the function maps the key to the address and passes it to the operating system, and the record is retrieved.

```

            graph LR
            Key --> Mapping[Address = HashFunction(Key)]
            Mapping -- Address --> File
            subgraph File
            direction TB
            A1[1]
            A2[2]
            A3[...]
            Ak[k]
            end
            File -- Record --> Record
            
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.12

Process of Working in Direct File (Hashed File)

Addr.	Key	Name	Balance
001	001	Mary Dodd	1432.45
002	002	Sarah Trapp	100.22
003	003	Bryan Devaux	11.45
004	004	Harry Eagle	14321.00
025	025	John Carver	7234.01
099	099	Tuan Ngo	15121.10
100	100	Shouli Feldman	87922.05

Accessing direct-hashed file

Extracted record

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.13

Text File versus Binary File

- A file stored on a storage device is a sequence of bits that can be interpreted by an application program as a **text file** or a **binary file**.

Interpreted as a text file

01000001 01000010

Two bytes represent two characters (A and B)

Interpreted as a binary file

01000001 01000010


Two bytes represent one number (16706)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.14

Text File

- A **text file** is a file of characters.
- It cannot contain integers, floating-point numbers, or any other data structures in their internal memory format.
 - To store these data types, they must be converted to their character equivalent formats.
- Example:** Character file sent to printer.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.15



Binary File

- A **binary file** is a collection of data stored in the internal format of the computer.
- Unlike text files, binary files contain data that is meaningful only if it is properly interpreted by a program.
- If the data is textual, one byte is used to represent one character.
- If the data is numeric, two or more bytes are considered for a data item.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.16



File System

- The file system consists of two distinct parts:
 - A collection of files (each storing related data)
 - A directory structure (which organizes and provides information about all the files in the system.
- Files are mapped by the operating system onto physical devices.
 - These storage devices are usually nonvolatile, so the contents are persistent through power failures and system reboots.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.17



File Attributes

- A file's attributes vary from one operating system to another but typically consist of these:
 - **Name** - The symbolic file name is the only information kept in human-readable form.
 - **Identifier** - This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
 - **Type** - This information is needed for systems that support different types of files.
 - **Location** - This information is a pointer to a device and to the location of the file on that device.
 - **Size** - The current size of the file.
 - **Protection** - Access-control information determines who can do reading, writing, executing, and so on.
 - **Time, date, and user identification** - This information may be kept for creation, last modification, and last use.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.18

File Attributes (contd...)

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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.19

File Operations

- **Creating a File**
 - Two steps are necessary to create a file.
 - First, space in the file system must be found for the file.
 - Second, an entry for the new file must be made in the directory.
- **Writing a File**
 - To write a file, we make a system call specifying both the name of the file and the information to be written to the file.
 - Given the name of the file, the system searches the directory to find the file's location.
 - The system must keep a write pointer to the location in the file where the next write is to take place.
 - The write pointer must be updated whenever a write occurs.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.20

File Operations (contd...)

- **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.
 - The directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place.
 - Once the read has taken place, the read pointer is updated.
 - Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file-position pointer.
 - Both the read and write operations use this same pointer, saving space and reducing system complexity.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.21

File Operations (contd...)

- **Repositioning within a File**
 - The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.
 - Repositioning within a file need not involve any actual I/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 the directory entry.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.22

File Operations (contd...)

- **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.
- **Other Operations - Appending new Information to the end of an Existing File, Renaming an Existing File**
- The operating system keeps a small table, called the **open-file table**, containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.23

Directory and Disk Structure

- Files are stored on random-access storage devices, including hard disks, optical disks, and solid state (memory-based) disks.
- **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 file system.
 - Storage devices can also be collected together into RAID sets that provide protection from the failure of a single disk.
 - Sometimes, disks are subdivided and also collected into RAID sets.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.24

Directory and Disk Structure (contd...)

- Partitions are also known as **slices** or **minidisks**.
- A **file system can be created** on each of these parts of the disk.
- Any entity containing a file system is generally known as a **volume**.
- The volume may be a **subset of a device**, a **whole device**, or **multiple devices linked together into a RAID set**.
- 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 that directory**) records information—such as name, location, size, and type—for all files on that volume.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.25

Directory and Disk Structure (contd...)

The diagram illustrates file-system organization across three disks. Disk 1 contains partition A (with a directory and files) and partition B (with a directory and files). Disk 2 contains partition C (with a directory and files). Disk 3 also contains partition C (with a directory and files). The text 'File-System Organization' is written below the diagram.

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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.26

Directory Structure

- The directory can be viewed as a symbol table that translates file names into their directory entries.
- Various operations such as: **search for a file**, **create a file**, **delete a file**, **list directory**, **renaming a file**, **traversing the file system**, etc. are performed on a directory
- The most common schemes for defining the logical structure of directory includes:
 - **Single-Level Directory**
 - **Two-Level Directory**
 - **Tree-Structured Directories**
 - **Acyclic-Graph Directories**
 - **General Graph Directory**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.27

Single-Level Directory

- It is the simplest directory structure.
- All files are contained in the same directory, which is easy to support and understand.
- This approach has limitations when the number of files increases or when the system has more than one user.
 - If two users call their data file "test", then the unique-name rule is violated.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.28

Two-Level Directory

- In the two-level directory structure, each user has his own user file directory.
 - 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.
- This structure effectively isolates one user from another.
- Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.29

Two-Level Directory (contd...)

- Programs provided as part of the system—loaders, assemblers, compilers, utility, routines, libraries, and so on—are generally defined as files.
- As the directory system is defined presently, this file name would be searched for in the current UFD.
- One solution would be to copy the system files into each UFD. However, copying all the system files would waste an enormous amount of space.
- Solution** – A special user directory is defined to contain the system files (for example, user 0). Whenever a file name is given to be loaded, the operating system first searches the local UFD.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.30

Tree-Structured Directory

- A generalization of Two-Level Directory structure is to extend the directory structure to a tree of arbitrary height.
- This generalization allows users to create their own subdirectories and to organize their files accordingly.
- A tree is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- A directory is simply another file, but it is treated in a special way.
- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.31

Tree-Structured Directory (contd...)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.32

Acyclic-Graph Directory

- An acyclic graph allows directories to share subdirectories and files.
- The same file or subdirectory may be in two different directories.
- The acyclic graph is a natural generalization of the tree-structured directory scheme.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.33

Directory Implementation

- **Linear List**
 - A linear list of file names is maintained with pointers to the data blocks.
 - **This method is simple to program but time-consuming to execute.**
 - To create a new file, we must first search the directory to be sure that no existing file has the same name.
 - To delete a file, we search the directory for the named file and then release the space allocated to it.
 - **To reuse the directory entry, we can do one of several things:**
 - We can mark the entry as unused (by assigning it a special name)
 - We can attach it to a list of free directory entries.
 - We can copy the last entry in the directory into the freed location and decrease the length of the directory.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.34

Directory Implementation (contd...)

- **Hash Table**
 - A linear list stores the directory entries, with the help of hash data structure.
 - The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
 - It decrease the directory search time.
 - Insertion and deletion are also fairly straightforward, although some provision must be made for collisions.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.35

Disk Structure (Magnetic Disks)

The diagram illustrates the physical structure of a magnetic disk. A central spindle is shown with a rotation arrow. Multiple platters are stacked on the spindle. Each platter has concentric tracks. Sectors are defined between tracks. Cylinders are formed by tracks at the same radial distance across different platters. An arm assembly is attached to the spindle, with read-write heads positioned over the tracks on each platter.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.36

Disk Structure (contd...)

- Drives generally rotate 60 to 200 times per second.
- **Transfer Rate** - The rate at which data flow between the drive and the computer.
- **Positioning Time (Seek Time)** – The time necessary to move the disk arm to the desired cylinder.
- **Rotational Latency** - The time necessary for the desired sector to rotate to the disk head.
- The disk head flies on an extremely thin cushion of air (measured in microns), there is a danger that the head will make contact with the disk surface. Although the disk platters are coated with a thin protective layer, the head can sometimes damage the magnetic surface. This accident is called a **head crash**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.37

Disk Structure (contd...)

- A disk drive is attached to a computer by a set of wires called an **I/O bus**.
- Several kinds of buses are available:
 - **Enhanced Integrated Drive Electronics (EIDE)**
 - **Advanced Technology Attachment (ATA)**
 - **Serial ATA (SATA)**
 - **Universal Serial Bus (USB)**
 - **Small Computer-Systems Interface (SCSI)**
- The data transfers on a bus are carried out by special electronic processors called **controllers**.
- The **host controller** is the controller at the computer end of the bus. A **disk controller** is built into each drive.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.38

Disk Structure (contd...)

- To perform a disk I/O operation, the computer places a command into the host controller.
- The host controller then sends the command via messages to the disk controller.
- The disk controller operates the disk-drive hardware to carry out the command.
- **Disk controllers usually have a built-in cache.**
- Data transfer at the disk drive happens between the cache and the disk surface.
- Data transfer to the host occurs between the cache and the host controller.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.39

Disk Structure (contd...)

- Disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
- The size of a logical block is usually 512 bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder.
- The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.40

Disk Structure (contd...)

- Constant Linear Velocity (CLV)
 - On media that use CLV, the density of bits per track is uniform.
 - The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold.
 - As we move from outer zones to inner zones, the number of sectors per track decreases.
 - Tracks in the outermost zone typically hold 40 percent more sectors than do tracks in the innermost zone.
 - The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head.
 - This method is used in CD-ROM and DVD-ROM drives.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.41

Disk Structure (contd...)

- Constant Angular Velocity (CAV)
 - The density of bits decreases from inner tracks to outer tracks to keep the data rate constant.
 - The disk rotation speed can stay constant.
 - This method is used in hard disks.
 - As disk technology improving, the number of cylinders per disk has been increasing.
 - Large disks have tens of thousands of cylinders.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.42

Allocation Methods

- The direct-access nature of disks allows us flexibility in the implementation of files.
- Three major methods of allocating disk space are in wide use:
 - Contiguous,
 - Linked, and
 - Indexed

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.43

Contiguous Allocation

- Each file occupy a set of contiguous blocks on the disk.
- With this ordering, assuming that only one job is accessing the disk, accessing block **b+1** after block **b** normally requires no head movement.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

count

0 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

28 29 30 31

directory


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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.44

Contiguous Allocation (contd...)

- The number of disk seeks required for accessing contiguously allocated files is minimal.
 - When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head needs only move from one track to the next.
- For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.
- For direct access to block **i** of a file that starts at block **b**, we can immediately access block **b + i**.
- Thus, both sequential and direct access can be supported by contiguous allocation.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.45



Contiguous Allocation (contd...)

- How to satisfy a request of size n from a list of free holes?
 - **First-fit** and **best-fit** are the most common strategies used to select a free hole from the set of available holes.
 - Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first fit is generally faster.
 - **All these algorithms suffer from the problem of external fragmentation.**
 - External fragmentation becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.46



Contiguous Allocation (contd...)

- How much space is needed for a file?
 - **When the file is created, the total amount of space it will need must be found and allocated.**
 - In copying a file, this determination maybe fairly simple.
 - In general, however, the size of an output file may be difficult to estimate.
 - If we allocate too little space to a file, we may find that the file cannot be extended.
 - Especially with a best-fit allocation strategy, the space on both sides of the file may be in use. Hence, we cannot make the file larger in place.
 - Even if the total amount of space needed for a file is known in advance, pre-allocation may be inefficient.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.47



Contiguous Allocation (contd...)

- To minimize these drawbacks, some operating systems use a **modified contiguous-allocation scheme**.
 - **A contiguous chunk of space is allocated initially.**
 - If that amount proves not to be large enough, another chunk of contiguous space, **known as an extent**, is added.
 - The location of a file's blocks is then recorded **as a location** and a **block count**, plus a **link to the first block of the next extent**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.48

Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks.
 - The disk blocks may be scattered anywhere on the disk.
 - The directory contains a pointer to the first and last blocks of the file.
 - There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.49

Linked Allocation (contd...)

- The major problem is that it can be used effectively only for sequential-access files.
 - To find the i^{th} block of a file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block.
 - Each access to a pointer requires a disk read, and some require a disk seek.
- The space required for the pointers is also a disadvantage.
- Another problem of linked allocation is reliability.
 - The files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer is lost or damaged?
- The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.50

Indexed Allocation

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation.
 - However, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
 - Indexed allocation solves this problem by bringing all the pointers together into one location: the index block.
 - Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file.
 - The directory contains the address of the index block.
 - To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.51

Indexed Allocation (contd...)

- When the file is created, all pointers in the index block are set to nil.
- When the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} index-block entry.
- Indexed allocation supports direct access without suffering from external fragmentation

The diagram illustrates indexed allocation. On the left, a cylinder represents a disk with 32 blocks numbered 0 to 31. A file named 'jeep' is shown with a directory entry pointing to an index block at address 19. The index block contains pointers to blocks 9, 16, 1, 10, 25, -1, and -1. Arrows indicate that the file 'jeep' is composed of blocks 9, 16, 1, 10, and 25.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.52

Indexed Allocation (contd...)

- Indexed allocation suffers from wasted space.
- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.
 - Consider a common case in which we have a file of only one or two blocks.
 - With linked allocation, we lose the space of only one pointer per block.
 - With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-nil.
 - This point raises the question of how large the index block should be.
 - The index block should be small. If the index block is too small, however, it will not be able to hold enough pointers for a large file.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.53

Disk Scheduling Algorithms

- Background**
 - Seek Time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
 - The **Rotational Latency** is the additional time for the disk to rotate the desired sector to the disk head.
 - The **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.
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Seek-Time-First (SSTF) Scheduling
- SCAN (Elevator Algorithm) Scheduling
- Circular SCAN (C-SCAN) Scheduling
- LOOK Scheduling

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.54

First-Come, First-Served (FCFS) Scheduling

Example: A disk queue requests for I/O to blocks on cylinders **98, 183, 37, 122, 14, 124, 65, 67**. Determine the total head movement (in cylinders) if the disk head is initially at cylinder 53.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

The diagram shows a horizontal axis representing disk cylinders from 0 to 199. The head starts at cylinder 53. It moves to 98, then to 183, then back to 37, then to 122, then to 14, then to 124, then to 65, and finally to 67. The path is a zig-zag across the disk.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.55

Shortest-Seek-Time-First (SSTF) Scheduling

- Service all the requests close to the current head position before moving the head far away to service other requests.
 - SSTF algorithm selects the request with the least seek time from the current head position.
 - Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
 - It may cause starvation of some requests.
 - The requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while the request from 14 is being serviced, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.56

SSTF Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders **98, 183, 37, 122, 14, 124, 65, 67**. Determine the total head movement (in cylinders) if the disk head is initially at cylinder 53.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

The diagram shows a horizontal axis representing disk cylinders from 0 to 199. The head starts at cylinder 53. It moves to 65, then to 67, then to 37, then to 122, then to 14, then to 124, then to 98, and finally to 183. The path is a series of small steps, always moving to the next closest request.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.57

SCAN (Elevator) Scheduling

- The disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- The SCAN algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.
- The **direction of head movement** in addition to the head's current position is important.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.58

SCAN (Elevator) Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders **98, 183, 37, 122, 14, 124, 65, 67**. Determine the total head movement (in cylinders) if the **disk head is initially at cylinder 53** and the **disk arm is moving toward 0**.

queue = 98, 183, 37, 122, 14, 124, 65, 67
 head starts at 53

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.59

C-SCAN Scheduling

- Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time.
- Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.
- When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
- The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.60

C-SCAN Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the **disk head is initially at cylinder 53** and the **disk arm is moving toward 199**.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.61

LOOK and C-LOOK Scheduling

- Both SCAN and C-SCAN move the disk arm across the full width of the disk.
 - In practice, neither algorithm is often implemented this way.
 - More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.
 - Versions of SCAN and C-SCAN that follow this pattern are called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in a given direction


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.62

C-LOOK Scheduling

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the **disk head is initially at cylinder 53** and the **disk arm is moving toward 199**.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.63




Question

Example: A disk queue requests for I/O to blocks on cylinders 82, 170, 43, 140, 24, 16, 190. Determine the total head movement, using the following algorithms, (in cylinders) if the disk head is initially at cylinder 50 and the disk arm is moving toward 199.

- FCFS
- SSTF
- SCAN
- C-SCAN
- LOOK
- C-LOOK


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.64



Device Management or I/O Management

- Computers operate many kinds of devices:
 - **Storage devices** (disks, tapes)
 - **Transmission devices** (network cards, modem)
 - **Human-interface devices** (screen, keyboards, mouse)
- To control the devices (which are connected to the Computer) is a **big concern because different I/O devices have different functions and different speed.**
- Different methods are required to control different devices.
- Different methods (required to control the devices) **form I/O Sub-system** for Kernel.
- To encapsulate the details and oddities of different devices, the Kernel of an OS is structured to use **device-driver** modules.
- The device drivers present a **uniform device-access interface** to I/O Sub-system.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.65



Port, Bus, Controller

- The device communicates with the machine via a connection point or port.
 - Serial Port
 - Parallel Port
 - USB Port
 - HDMI Port
- If some devices use a common set of wires, then it is called a Bus.
 - A Bus is set of wires in which messages are conveyed by pattern of electrical voltage.
- A Controller is a collection of electronics that can operate a Port, Bus or a Device.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.66



Character-Stream and Block Devices

- **A character-stream device transfers bytes one-by-one**
 - It can use a few bytes for their operations and it does not requires buffering.
 - The response time and processing speed are faster than the block devices.
 - In character oriented the I/O can be performed directly between the system and the user and as such, saves the kernel from the copying process and the buffering mechanisms overhead.
 - **Interface** → **Character** → Keyword → `put(), get()`.
- **A block device transfer a block of bytes as a unit.**
 - Block devices are storage devices that can provide data operations in fixed-size blocks for both reading and writing.
 - Hard drives, floppy disks, and optical drives, such as DVD-ROMs and CD-ROMs, are some examples of such machines.
 - To speed up the access during read and write operations it requires a buffering mechanism.
 - **Interface** → **Block** → Disk → `read(), write(), seek()`.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.67



Blocking and Non-blocking I/O

- **Blocking I/O means that a given thread/process cannot do anything more until the I/O is fully received.**
 - When an application/process issues blocking system call, the process execution Stops.
 - **Example:** With blocking I/O, when a client makes a request to connect with the server, the thread that handles that connection is blocked until there is some data to read, or the data is fully written.
 - **Until the relevant operation is complete that thread/process can do nothing else but wait in the wait queue.**
- **Some user-level processes perform non-blocking I/O.**
 - **Example:** Receiving data from Keyword or Mouse while displaying processed data on screen.
 - Video player is example of non-blocking I/O which reads data/frames from HDD and play the video on screen (after compression).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.68




Blocking and Non-blocking I/O

- **Blocking I/O means that a given thread/process cannot do anything more until the I/O is fully received.**
 - When an application/process issues blocking system call, the process execution Stops.
 - **Example:** With blocking I/O, when a client makes a request to connect with the server, the thread that handles that connection is blocked until there is some data to read, or the data is fully written.
 - **Until the relevant operation is complete that thread/process can do nothing else but wait in the wait queue.**
- **Some user-level processes perform non-blocking I/O.**
 - **Example:** Receiving data from Keyword or Mouse while displaying processed data on screen.
 - Video player is example of non-blocking I/O which reads data/frames from HDD and play the video on screen (after compression).

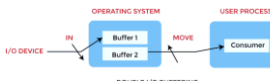
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.69

Buffering and Double Buffering

- The **buffer** is an area in the main memory used to store or hold the data temporarily.
 - In other words, buffer temporarily stores data transmitted from one place to another, either between two devices or an application.
 - The act of storing data temporarily in the buffer is called **buffering**.
 - Buffering helps in matching speed between two devices in which the data is transmitted.
 - It helps the devices with different sizes of data transfer to get adapted to each other.
 - ✓ In computer networking, the large message is fragmented into small fragments and sent over the network. The fragments are accumulated in the buffer at the receiving end and reassembled to form a complete large message.



SINGLE I/O BUFFERING



DOUBLE I/O BUFFERING

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.70

I/O Channels

- A channel is an independent hardware component that co-ordinates all I/O to a set of controllers.
 - Channels use separate, independent and low-cost processors for its functioning which are called **Channel Processors**.
 - Each channel supports one or more controllers or devices.
 - Channel programs contain list of commands to the channel itself and for various connected controllers or devices.
 - When OS want to perform I/O, the OS initiates an I/O machine instruction for the channel and then rest of the task to complete the I/O is performed by the channel.
 - Multiplexer (Byte Multiplexer and Block Multiplexer)
 - Selector

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.71

Disk Management

- Disk Formatting
- Boot Block
- Bad Blocks

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U4.72

Disk Formatting (Physical)

- A new magnetic disk is a blank slate (just a platter of a magnetic recording material).
- To store data, it must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting**, or **physical formatting**.
- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector typically consists of a **header**, a **data area** (usually 512 bytes in size), and a **trailer**.
- The header and trailer contain information used by the disk controller, such as a **sector number** and an **error-correcting code (ECC)**.
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- The controller automatically does the ECC processing whenever a sector is read or written.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.73

Disk Formatting (Logical)

- To use a disk to hold files, the OS still needs to record its own data structures on the disk.
- It does so in two steps:
 - The first step is to partition the disk into one or more groups of cylinders.
 - ✓ The OS can treat each partition as though it were a separate disk.
 - ✓ For instance, one partition can hold a copy of the OS's executable code, while another holds user files.
 - After partitioning, the second step is logical formatting (or creation of a file system).
 - ✓ In this step, the OS stores the initial file-system data structures onto the disk.
 - ✓ These data structures may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.74

Free-Space Management

- To keep track of free disk space, the system maintains a **free-space list**. The free-space list records all free disk blocks.
 - To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
 - This space is then removed from the free-space list.
 - When a file is deleted, its disk space is added to the free-space list.
- **Approaches for Free-Space Management**
 - **Bit Vector**
 - **Linked List**
 - **Grouping**
 - **Counting**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.75

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Free-Space Management: Bit Vector

- Each block is represented by 1 bit.
 - If the block is free, the bit is 1.
 - If the block is allocated, the bit is 0.
- **Example:** Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map or bit vector would be:
 - 001111001111110001100000011100000 ...
- Bit vectors are inefficient unless the entire vector is kept in main memory.
- Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.76

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Free-Space Management: Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
 - This first block contains a pointer to the next free disk block, and so on.
 - OS simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.
 - It is not efficient to give the address of a large number of free blocks since traversing in linked-list is sequential.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.77

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Free-Space Management: Grouping

- This approach stores the addresses of n free blocks in the first free block.
 - The first n-1 of these blocks are actually free.
 - The last block contains the addresses of another n free blocks, and so on.
 - The addresses of a large number of free blocks can now be found quickly unlike the situation when the standard linked-list approach is used.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.78

Free-Space Management: Counting

- Generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
 - Thus, rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block.
 - Each entry in the free-space list then consists of a disk address and a count.
 - These entries can be stored in a B-tree, rather than a linked list, for efficient lookup, insertion, and deletion.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.79

RAID Structure


- RAID - Redundant Arrays of Independent Disks (RAIDs)
- A variety of disk-organization techniques, collectively called RAIDs are commonly used to address the performance and reliability issues.
 - The solution to the problem of reliability is to introduce redundancy (duplicate every disk).
 - The technique of duplicating every disk is called mirroring.
 - With mirroring, a logical disk consists of two physical disks, and every write is carried out on both disks. The result is called a mirrored volume.



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.80

RAID Structure (contd...)

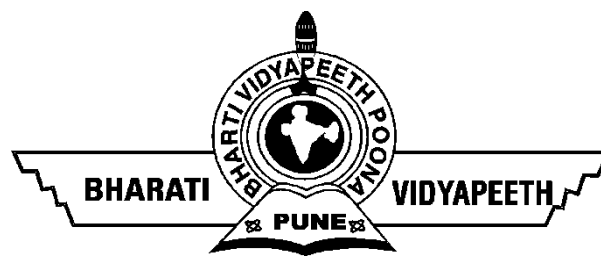
- Parallel access to multiple disks improves performance.
 - The number of reads per unit time has doubled.
 - The transfer rate of each read is the same as in a single-disk system.
 - With multiple disks, we can improve the transfer rate as well by striping data across the disks.
 - Data striping consists of splitting the bits of each byte across multiple disks; such striping is called bit-level striping.
 - In block-level striping, for instance, blocks of a file are striped across multiple disks; with n disks, block i of a file goes to disk $(i \bmod n) + 1$.
 - Block-level striping is the most common.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.81

 **RAID Levels**

- Mirroring provides high reliability, but it is expensive.
- Striping provides high data-transfer rates, but it does not improve reliability.
- There are various schemes to provide redundancy:
 - **RAID Level 0.** RAID level 0 refers to disk arrays with striping at the level of blocks but without any redundancy. 
 - **RAID Level 1.** RAID level 1 refers to disk mirroring. 
 - Other levels – Students should cover through home assignment.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U4.82



Operating Systems with Linux

(MCA-105)

Unit – 4

by

Dr. Sunil Pratap Singh

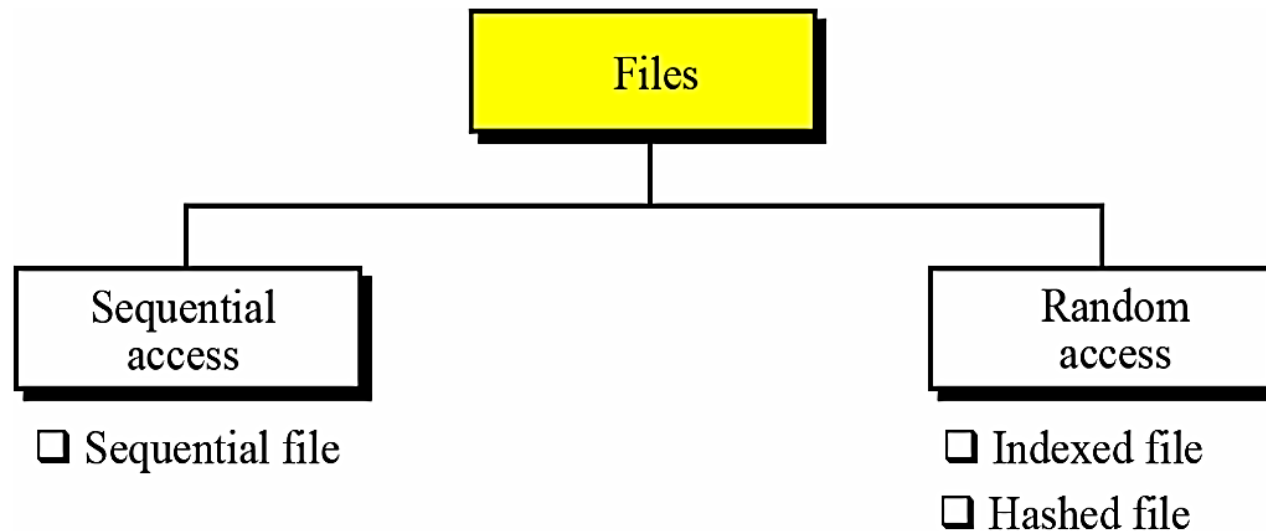
(Associate Professor, BVICAM, New Delhi)

2023

- A **file** is a collection of related data stored on mass storage (e.g., disk or tape).
 - The data is subdivided into **records** (e.g., student information).
 - Each record contains a number of **fields** (e.g., roll number, name).
 - One (or more) field is the **key** field (e.g., roll number).

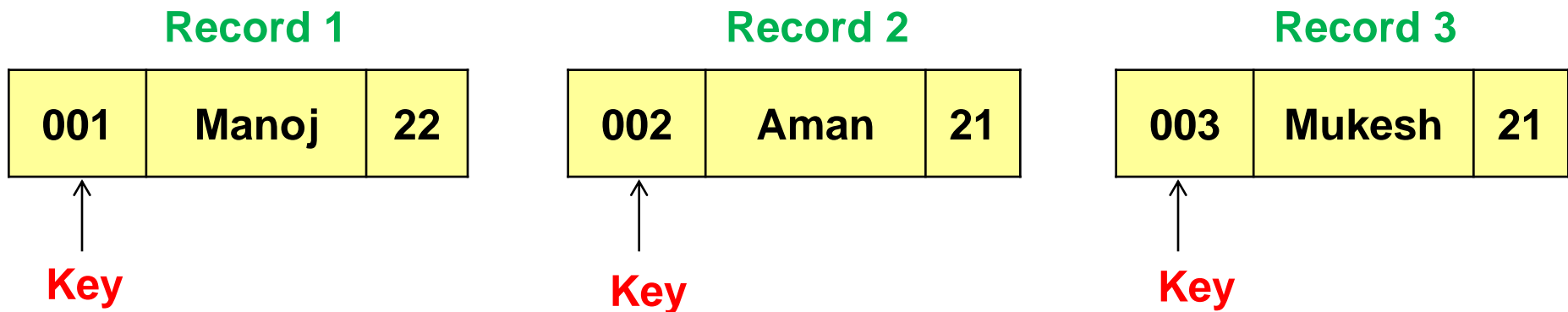
File Organizations

- A **file organization** refers to the way records are arranged on a storage device.
- How best the files be arranged for easy of access?



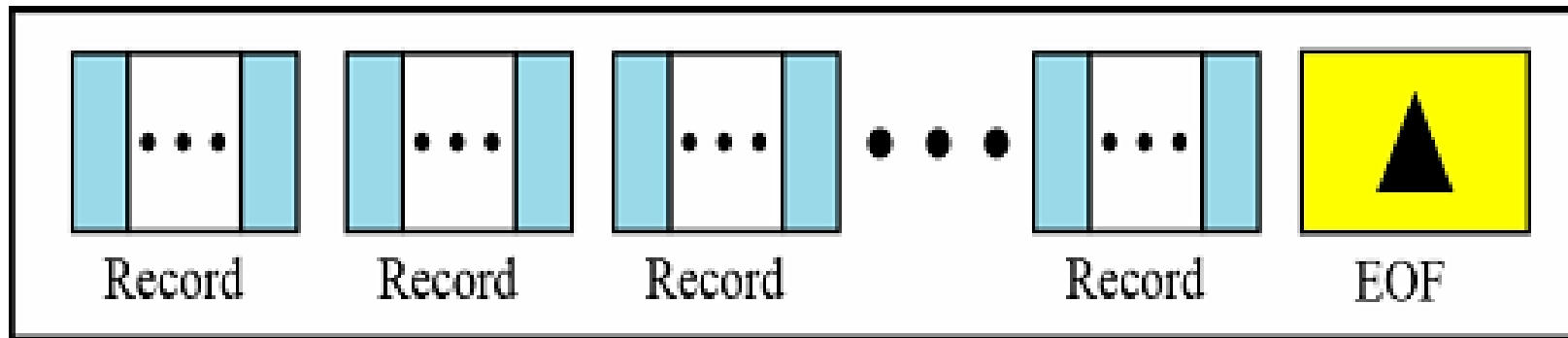
Sequential Files

- A sequential file is one in which records can only be accessed one after another from beginning to end.
- This file organization is the simplest way to store and retrieve records of a file.
- In this file, data records are stored in **some specific sequence**, e.g., order of arrival, value of key field, etc.



Sequential Files

- The records of a sequential file cannot be accessed at random, i.e., to access the n^{th} record, one must traverse the preceding $(n-1)$ records.



Sequential File Organization

- In sequential file organization, the actual storage of records might or might not be sequential:
 - On a tape, it usually is.
 - On a disk, it might be distributed across sectors and the operating system would use a linked list of sectors to provide the illusion of sequentially.
- Editors and compilers usually access files in this fashion.

Sequential File Organization

- **Advantages**

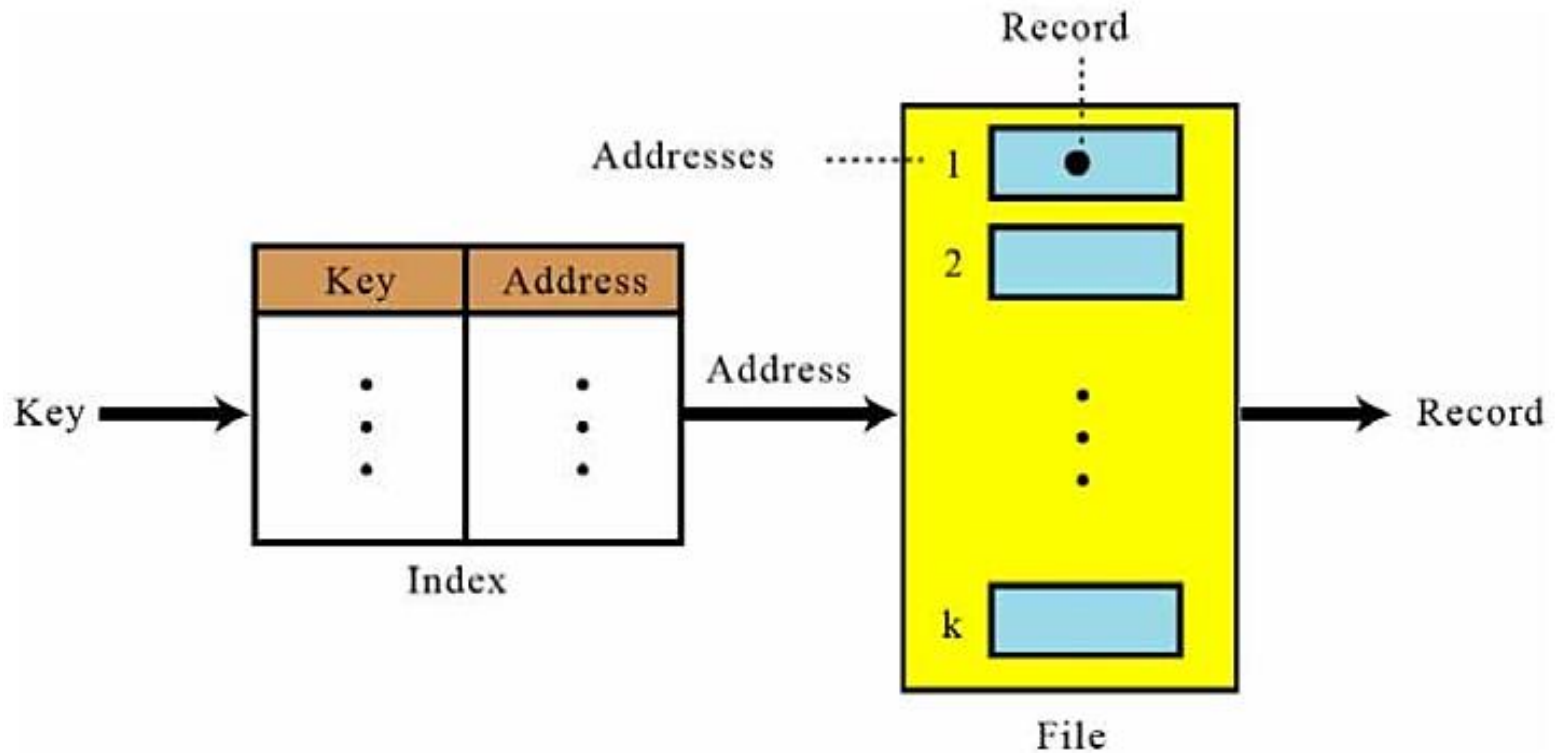
- Easy to handle
- Involves no overhead
- Can be stored on tapes as well as disks

- **Disadvantages**

- Records can only be accessed in sequence
- Time consuming

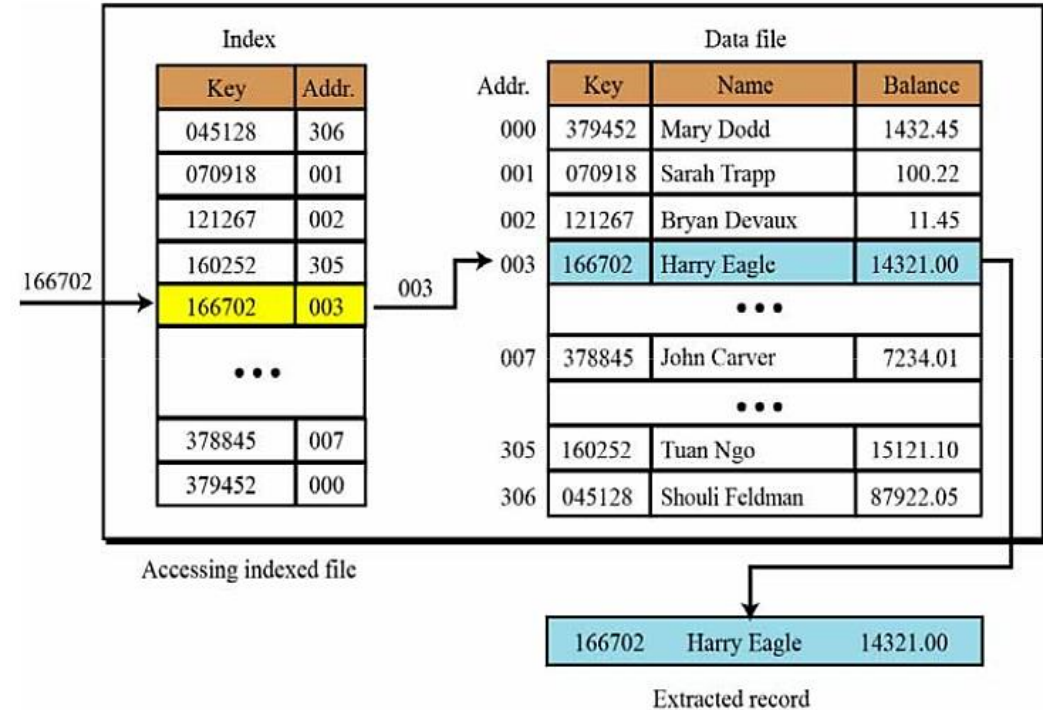
Indexed File

- To access a record in a file randomly, we need to know the address of the record.



Indexed File: Logical View

- An **indexed file** is made of a **data file** (also known as **relative file**), which is a sequential file, and an **index**.
- The index itself is a **very small file** with only two fields: **the key of the sequential file** and **the address of the corresponding record on the disk**.
- The index is sorted based on the key values of the data files.



Indexed File: Accessing a Record

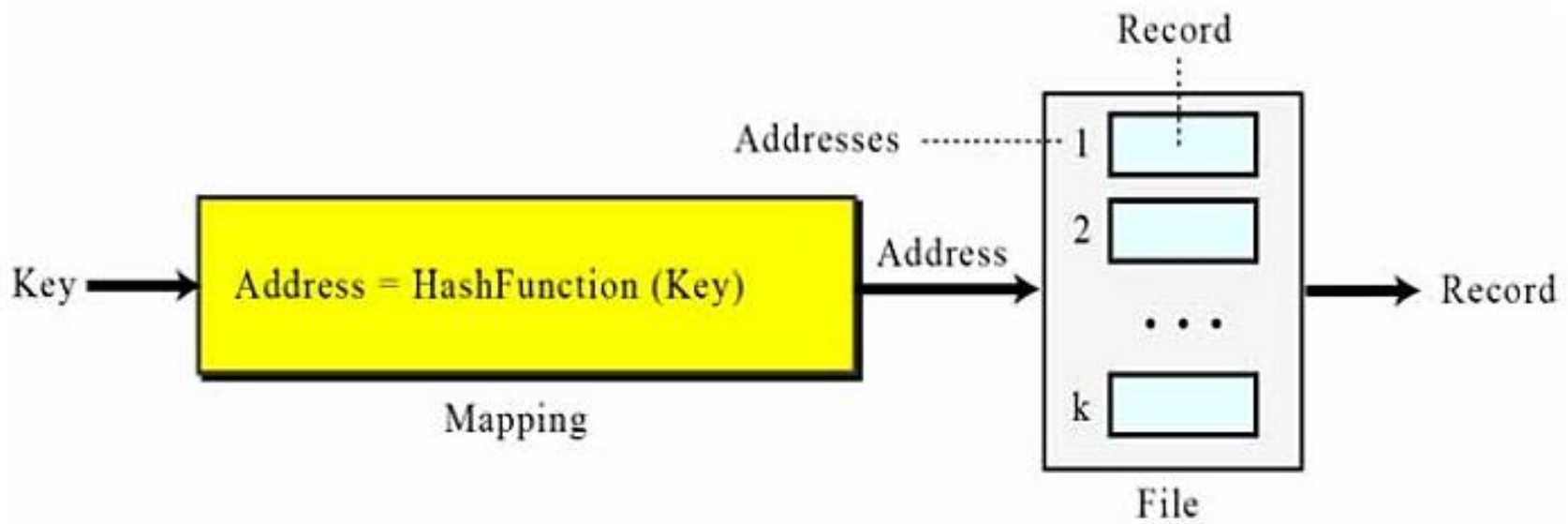
- Accessing a record in the file requires following steps:
 - The entire index file is loaded into main memory (the file is small and uses little memory).
 - The index entries are searched, using an efficient search algorithm such as a binary search, to find the desired key.
 - The address of the record is retrieved.
 - Using the address, the data record is retrieved and passed to the user.

Inverted File

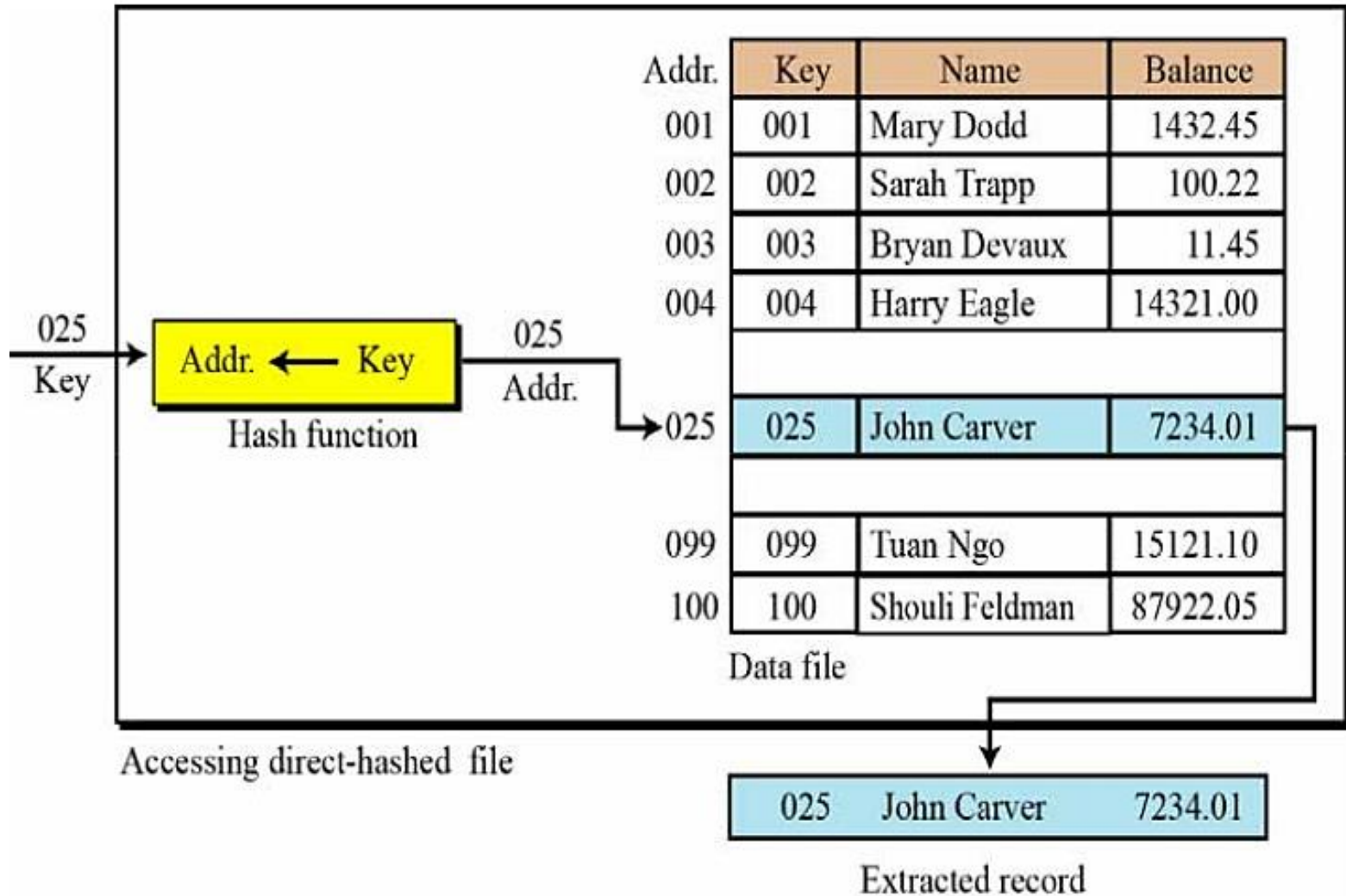
- One of the advantages of indexed files is that we can have **more than one index**, each with a different key.
- This type of indexed file is usually called an **inverted file**.

Direct File (Hashed File)

- A **hashed file** uses a **mathematical function** to **map the key to the address**.
- The user gives the key, the function maps the key to the address and passes it to the operating system, and the record is retrieved.



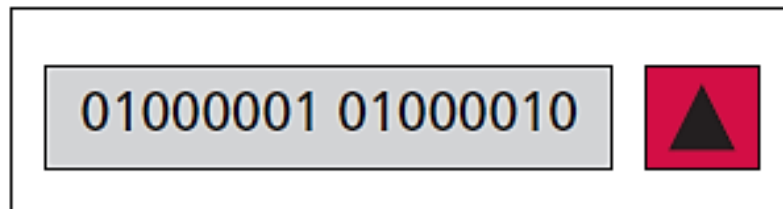
Process of Working in Direct File (Hashed File)



Text File *versus* Binary File

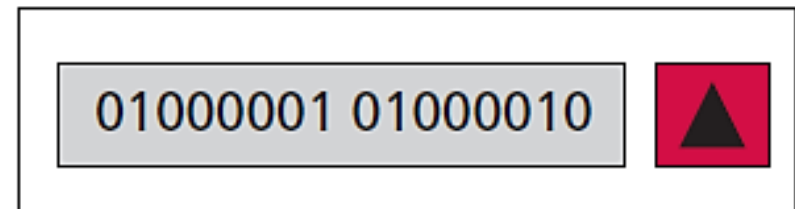
- A file stored on a storage device is a sequence of bits that can be interpreted by an application program as a **text file** or a **binary file**.

Interpreted as a text file



Two bytes represent two characters
(A and B)

Interpreted as a binary file



Two bytes represent one number
(16706)

Text File

- A **text file** is a file of characters.
- It cannot contain integers, floating-point numbers, or any other data structures in their internal memory format.
 - To store these data types, they must be converted to their character equivalent formats.
- **Example:** Character file sent to printer.

Binary File

- A **binary file** is a collection of data stored in the internal format of the computer.
- Unlike text files, binary files contain data that is meaningful only if it is properly interpreted by a program.
- If the data is textual, one byte is used to represent one character.
- If the data is numeric, two or more bytes are considered for a data item.

File System

- The file system consists of two distinct parts:
 - A collection of files (each storing related data)
 - A directory structure (which organizes and provides information about all the files in the system.
- Files are mapped by the operating system onto physical devices.
 - These storage devices are usually nonvolatile, so the contents are persistent through power failures and system reboots.

File Attributes

- A file's attributes vary from one operating system to another but typically consist of these:
 - **Name** - The symbolic file name is the only information kept in human-readable form.
 - **Identifier** - This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
 - **Type** - This information is needed for systems that support different types of files.
 - **Location** - This information is a pointer to a device and to the location of the file on that device.
 - **Size** - The current size of the file.
 - **Protection** - Access-control information determines who can do reading, writing, executing, and so on.
 - **Time, date, and user identification** - This information may be kept for creation, last modification, and last use.



File Attributes (contd...)

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

File Operations

- **Creating a File**

- Two steps are necessary to create a file.
- First, space in the file system must be found for the file.
- Second, an entry for the new file must be made in the directory.

- **Writing a File**

- To write a file, we make a system call specifying both the name of the file and the information to be written to the file.
- Given the name of the file, the system searches the directory to find the file's location.
- The system must keep a write pointer to the location in the file where the next write is to take place.
- The write pointer must be updated whenever a write occurs.

File Operations (contd...)

- **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.
- The directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place.
- Once the read has taken place, the read pointer is updated.
- Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file-position pointer.
- Both the read and write operations use this same pointer, saving space and reducing system complexity.

File Operations (contd...)

- **Repositioning within a File**

- The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.
- Repositioning within a file need not involve any actual I/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 the directory entry.

File Operations (contd...)

- **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.

- **Other Operations - Appending new Information to the end of an Existing File, Renaming an Existing File**

- The operating system keeps a small table, called the **open-file table**, containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table.



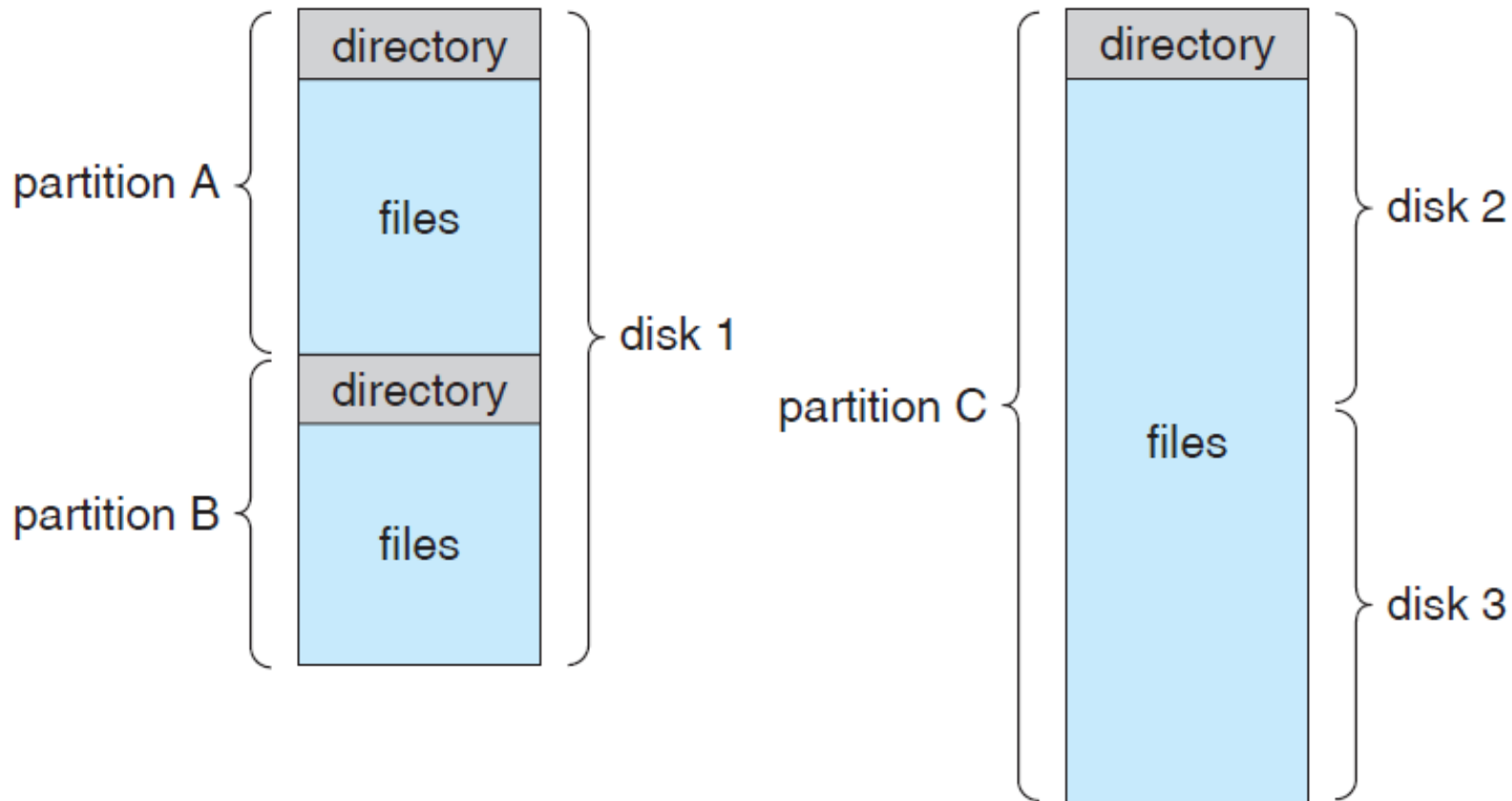
Directory and Disk Structure

- Files are stored on random-access storage devices, including hard disks, optical disks, and solid state (memory-based) disks.
- 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 file system.
 - Storage devices can also be collected together into RAID sets that provide protection from the failure of a single disk.
 - Sometimes, disks are subdivided and also collected into RAID sets.

Directory and Disk Structure (contd...)

- Partitions are also known as **slices** or **minidisks**.
- A **file system can be created** on each of these parts of the disk.
- Any entity containing a file system is generally known as a **volume**.
- The volume may be a **subset of a device**, a **whole device**, or **multiple devices linked together into a RAID set**.
- 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 that directory**) records information—such as name, location, size, and type—for all files on that volume.

Directory and Disk Structure (contd...)



File-System Organization

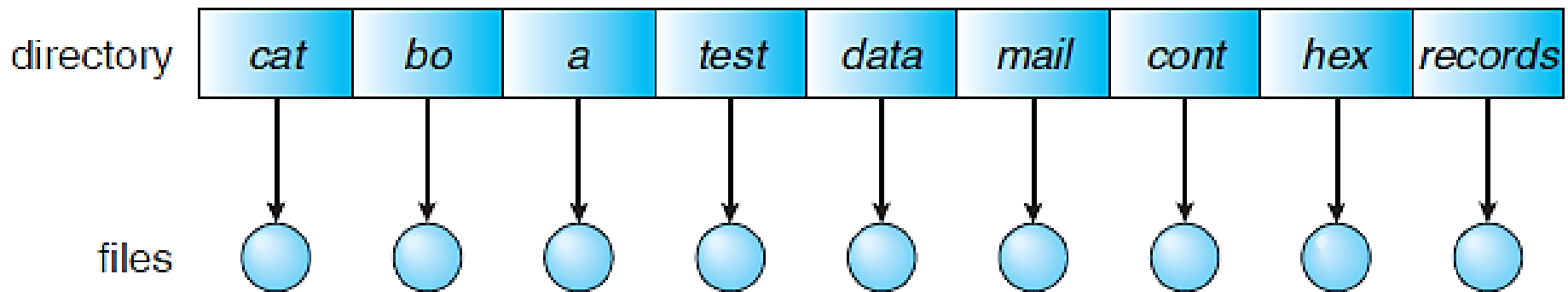
Computer systems may have zero or more file systems, and the file systems maybe of varying types.

Directory Structure

- The directory can be viewed as a symbol table that translates file names into their directory entries.
- Various operations such as: search for a file, create a file, delete a file, list directory, renaming a file, traversing the file system, etc. are performed on a directory
- The most common schemes for defining the logical structure of directory includes:
 - Single-Level Directory
 - Two-Level Directory
 - Tree-Structured Directories
 - Acyclic-Graph Directories
 - General Graph Directory

Single-Level Directory

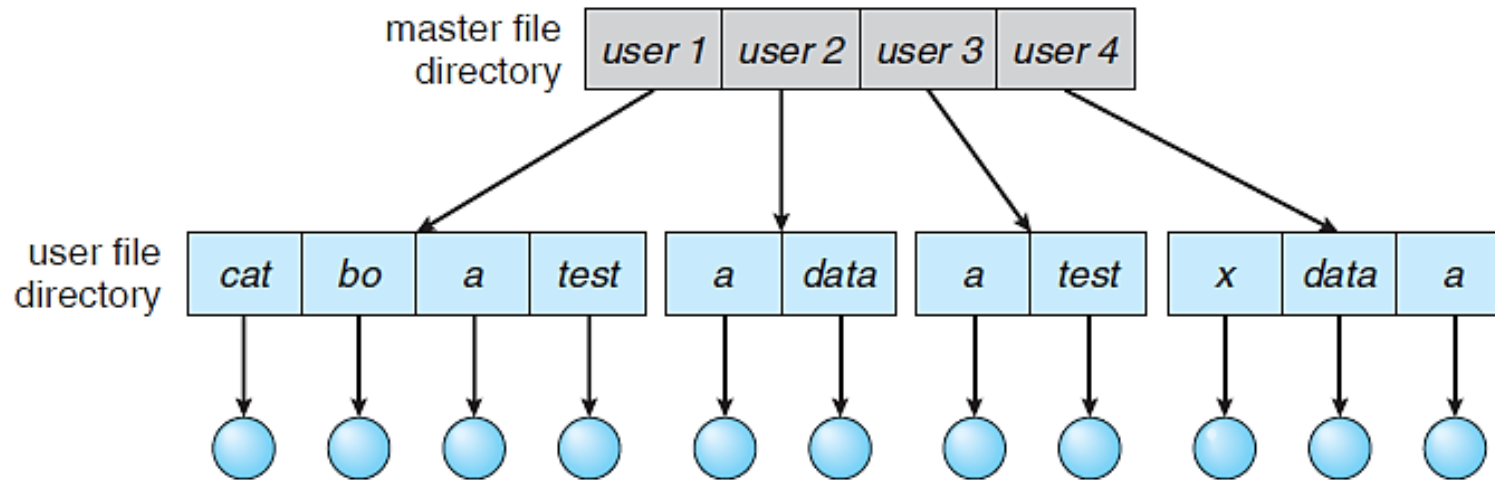
- It is the simplest directory structure.
- All files are contained in the same directory, which is easy to support and understand.
- This approach has limitations when the number of files increases or when the system has more than one user.
 - If two users call their data file “**test**”, then the unique-name rule is violated.



Two-Level Directory

- In the two-level directory structure, each user has his own user file directory.
 - 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
- This structure effectively isolates one user from another.
- Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files.

Two-Level Directory (contd...)

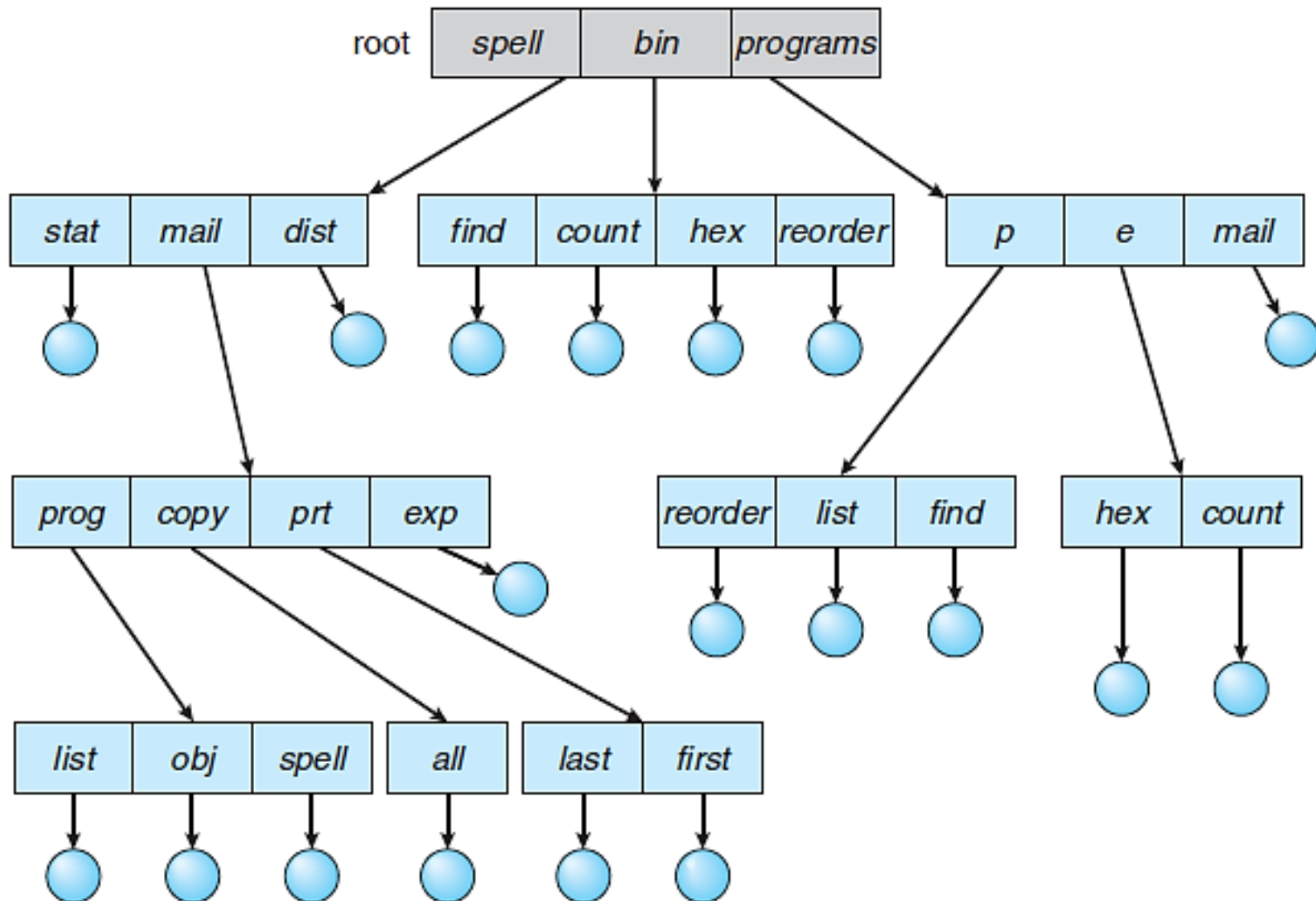


- Programs provided as part of the system—loaders, assemblers, compilers, utility, routines, libraries, and so on—are generally defined as files.
- As the directory system is defined presently, this file name would be searched for in the current UFD.
- One solution would be to copy the system files into each UFD. However, copying all the system files would waste an enormous amount of space.
- **Solution** – A special user directory is defined to contain the system files (for example, user 0). Whenever a file name is given to be loaded, the operating system first searches the local UFD.

Tree-Structured Directory

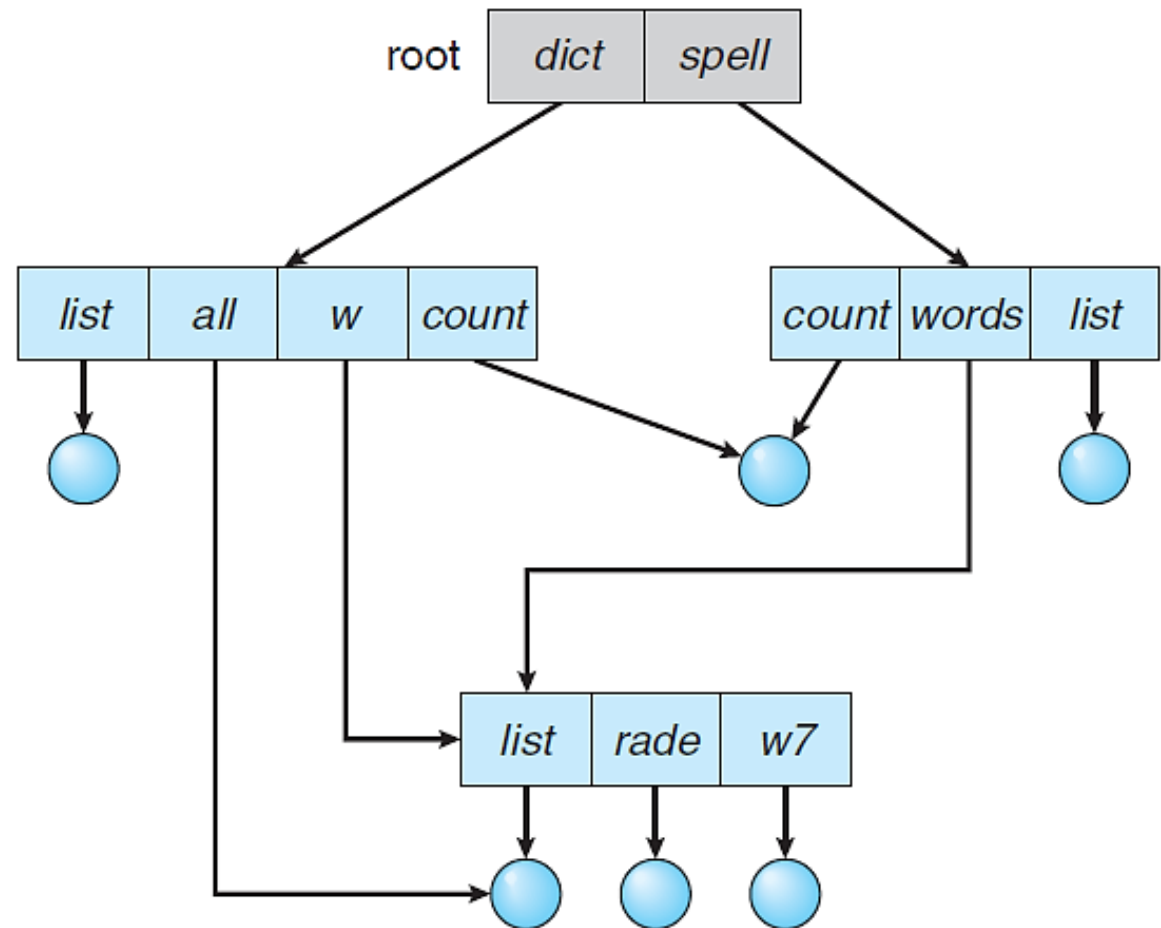
- A generalization of Two-Level Directory structure is to extend the directory structure to a tree of arbitrary height.
- This generalization allows users to create their own subdirectories and to organize their files accordingly.
- A tree is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- A directory is simply another file, but it is treated in a special way.
- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

Tree-Structured Directory (contd...)



Acyclic-Graph Directory

- An acyclic graph allows directories to share subdirectories and files.
- The same file or subdirectory may be in two different directories.
- The acyclic graph is a natural generalization of the tree-structured directory scheme.



Directory Implementation

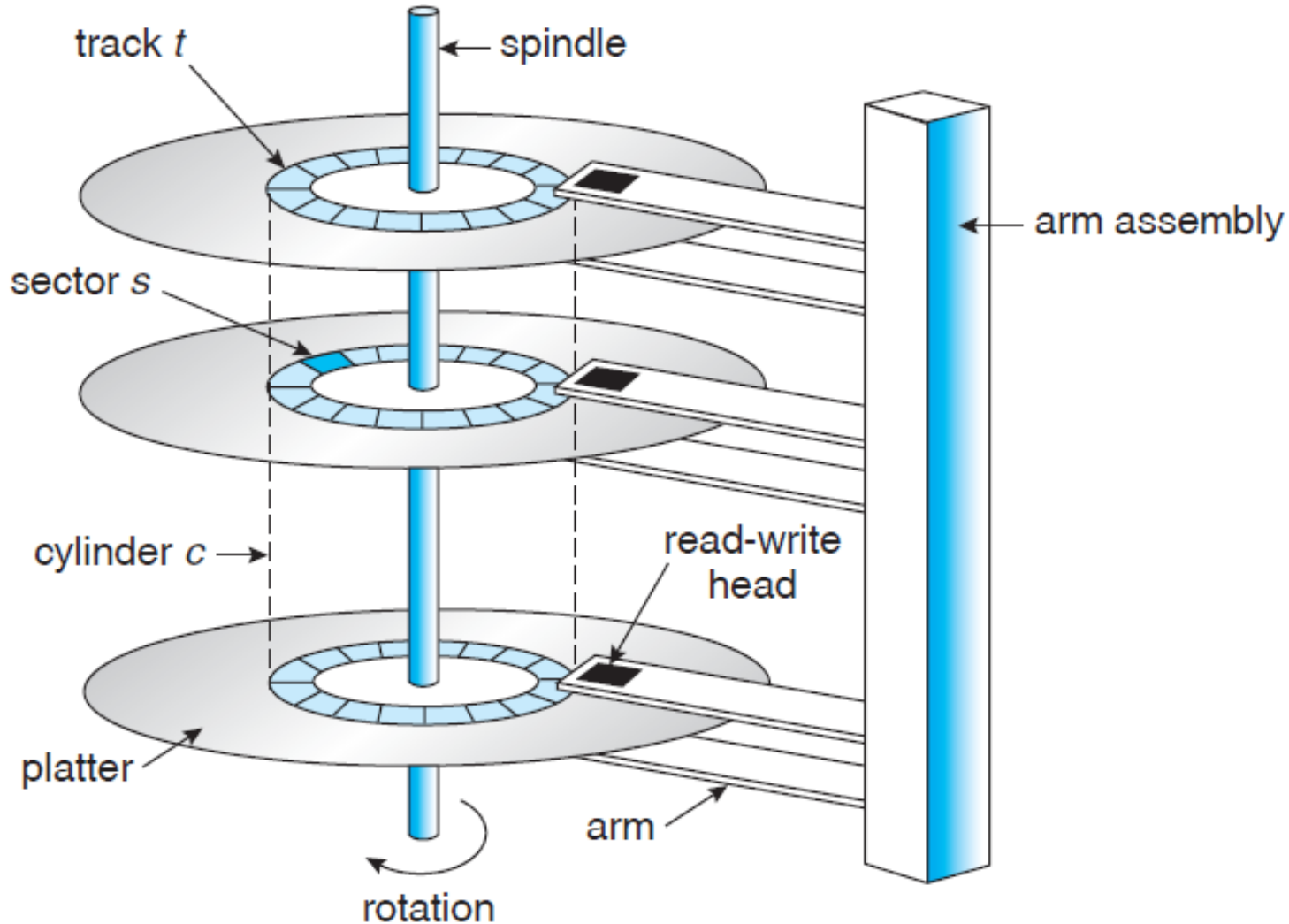
- Linear List

- A linear list of file names is maintained with pointers to the data blocks.
- This method is simple to program but time-consuming to execute.
 - To create a new file, we must first search the directory to be sure that no existing file has the same name.
 - To delete a file, we search the directory for the named file and then release the space allocated to it.
- To reuse the directory entry, we can do one of several things:
 - We can mark the entry as unused (by assigning it a special name)
 - We can attach it to a list of free directory entries.
 - We can copy the last entry in the directory into the freed location and decrease the length of the directory.

- Hash Table

- A linear list stores the directory entries, with the help of hash data structure.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- It decrease the directory search time.
- Insertion and deletion are also fairly straightforward, although some provision must be made for collisions.

Disk Structure (Magnetic Disks)



Disk Structure (contd...)

- Drives generally rotate 60 to 200 times per second.
- **Transfer Rate** - The rate at which data flow between the drive and the computer.
- **Positioning Time (Seek Time)** – The time necessary to move the disk arm to the desired cylinder.
- **Rotational Latency** - The time necessary for the desired sector to rotate to the disk head.
- The disk head flies on an extremely thin cushion of air (measured in microns), there is a danger that the head will make contact with the disk surface. Although the disk platters are coated with a thin protective layer, the head can sometimes damage the magnetic surface. This accident is called a **head crash**.

Disk Structure (contd...)

- A disk drive is attached to a computer by a set of wires called an **I/O bus**.
- Several kinds of buses are available:
 - Enhanced Integrated Drive Electronics (EIDE)
 - Advanced Technology Attachment (ATA)
 - Serial ATA (SATA)
 - Universal Serial Bus (USB)
 - Small Computer-Systems Interface (SCSI)
- The data transfers on a bus are carried out by special electronic processors called **controllers**.
- The **host controller** is the controller at the computer end of the bus. A **disk controller** is built into each drive.

Disk Structure (contd...)

- To perform a disk I/O operation, the computer places a command into the host controller.
- The host controller then sends the command via messages to the disk controller.
- The disk controller operates the disk-drive hardware to carry out the command.
- **Disk controllers usually have a built-in cache.**
- Data transfer at the disk drive happens between the cache and the disk surface.
- Data transfer to the host occurs between the cache and the host controller.

Disk Structure (contd...)

- Disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
- The size of a logical block is usually 512 bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder.
- The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

- **Constant Linear Velocity (CLV)**

- On media that use CLV, the density of bits per track is uniform.
- The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold.
- As we move from outer zones to inner zones, the number of sectors per track decreases.
- Tracks in the outermost zone typically hold 40 percent more sectors than do tracks in the innermost zone.
- The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head.
- **This method is used in CD-ROM and DVD-ROM drives.**

- **Constant Angular Velocity (CAV)**

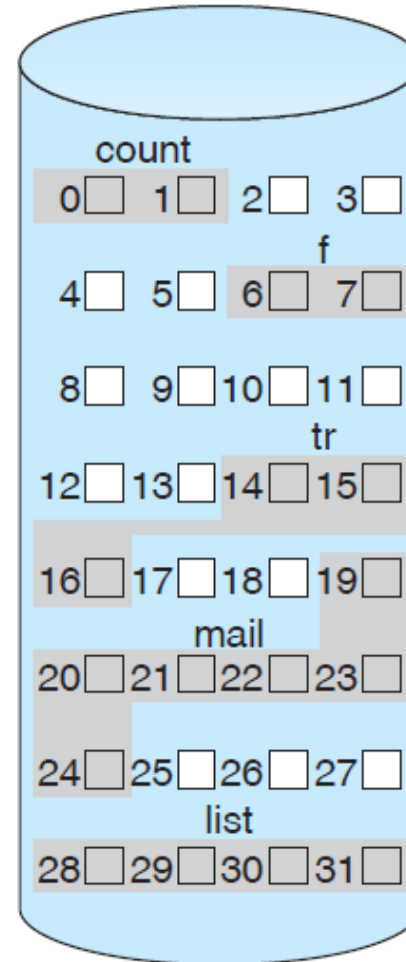
- The density of bits decreases from inner tracks to outer tracks to keep the data rate constant.
- The disk rotation speed can stay constant.
- **This method is used in hard disks.**
- **As disk technology improving, the number of cylinders per disk has been increasing.**
- Large disks have tens of thousands of cylinders.

Allocation Methods

- The direct-access nature of disks allows us flexibility in the implementation of files.
- Three major methods of allocating disk space are in wide use:
 - Contiguous,
 - Linked, and
 - Indexed

Contiguous Allocation

- Each file occupy a set of contiguous blocks on the disk.
- With this ordering, assuming that only one job is accessing the disk, accessing block $b+1$ after block b normally requires no head movement.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.



directory

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

Contiguous Allocation (contd...)

- The number of disk seeks required for accessing contiguously allocated files is minimal.
 - When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head needs only move from one track to the next.
- For **sequential access**, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.
- For **direct access** to block **i** of a file that starts at block **b**, we can immediately access block **b + i**.
- Thus, both sequential and direct access can be supported by contiguous allocation.

Contiguous Allocation (contd...)

- How to satisfy a request of size n from a list of free holes?
 - **First-fit** and **best-fit** are the most common strategies used to select a free hole from the set of available holes.
 - Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first fit is generally faster.
 - **All these algorithms suffer from the problem of external fragmentation.**
 - External fragmentation becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.

Contiguous Allocation (contd...)

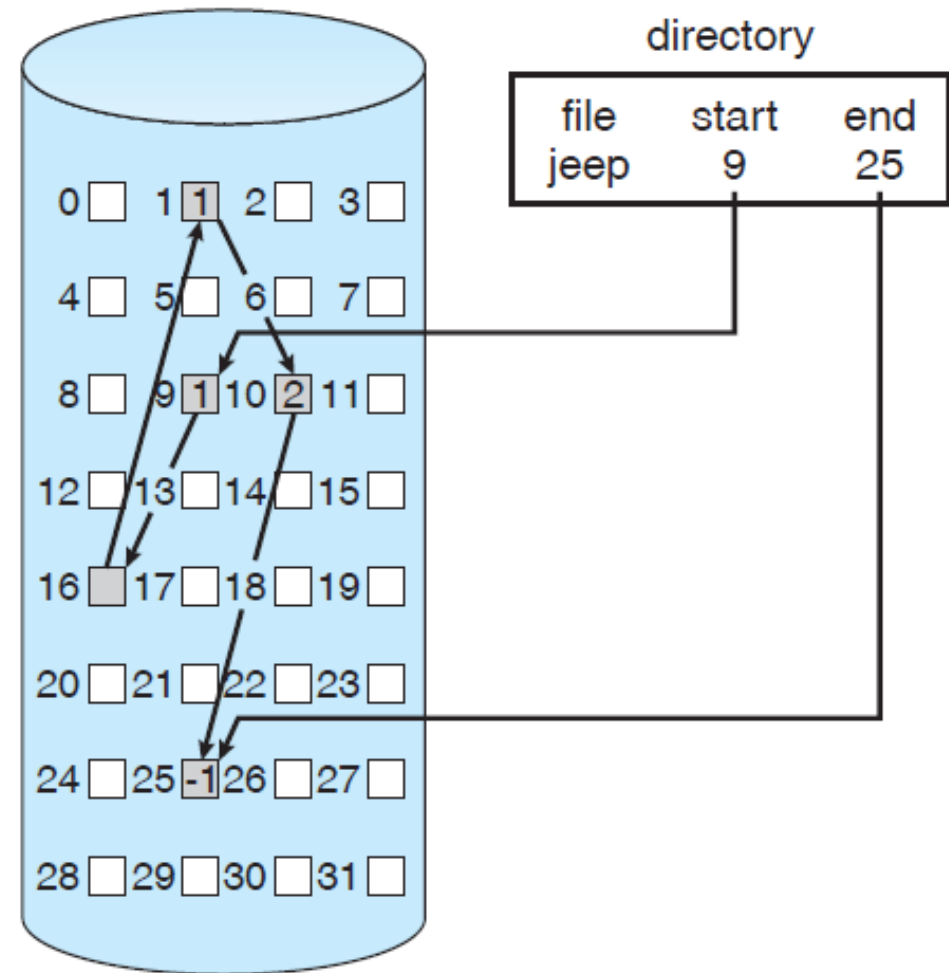
- How much space is needed for a file?
 - When the file is created, the total amount of space it will need must be found and allocated.
 - In copying a file, this determination maybe fairly simple.
 - In general, however, the size of an output file may be difficult to estimate.
 - If we allocate too little space to a file, we may find that the file cannot be extended.
 - Especially with a best-fit allocation strategy, the space on both sides of the file may be in use. Hence, we cannot make the file larger in place.
 - Even if the total amount of space needed for a file is known in advance, pre-allocation may be inefficient.

Contiguous Allocation (contd...)

- To minimize these drawbacks, some operating systems use a **modified contiguous-allocation scheme**.
 - A contiguous chunk of space is allocated initially.
 - If that amount proves not to be large enough, another chunk of contiguous space, **known as an extent**, is added.
 - The location of a file's blocks is then recorded **as a location** and **a block count**, plus **a link to the first block of the next extent**.

Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks.
 - The disk blocks may be scattered anywhere on the disk.
 - The directory contains a pointer to the first and last blocks of the file.
 - There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.



Linked Allocation (contd...)

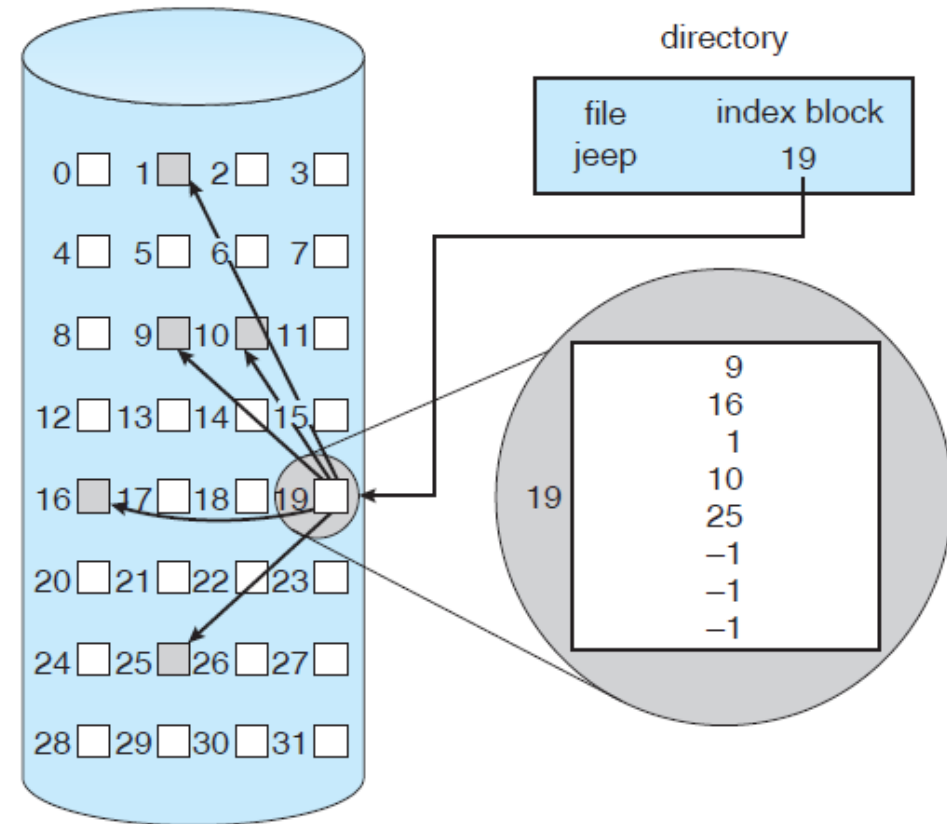
- The major problem is that it can be used effectively only for sequential-access files.
 - To find the i^{th} block of a file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block.
 - Each access to a pointer requires a disk read, and some require a disk seek.
- The space required for the pointers is also a disadvantage.
- Another problem of linked allocation is reliability.
 - The files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer is lost or damaged?
- The usual solution to this problem is to collect blocks into multiples, called clusters, and to allocate clusters rather than blocks.

Indexed Allocation

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation.
 - However, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
 - Indexed allocation solves this problem by bringing all the pointers together into one location: the index block.
 - Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file.
 - The directory contains the address of the index block.
 - To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.

Indexed Allocation (contd...)

- When the file is created, all pointers in the index block are set to nil.
- When the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} index-block entry.
- Indexed allocation supports direct access without suffering from external fragmentation



Indexed Allocation (contd...)

- Indexed allocation suffers from wasted space.
- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.
 - Consider a common case in which we have a file of only one or two blocks.
 - With linked allocation, we lose the space of only one pointer per block.
 - With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-nil.
 - This point raises the question of how large the index block should be.
 - The index block should be small. If the index block is too small, however, it will not be able to hold enough pointers for a large file.

Disk Scheduling Algorithms

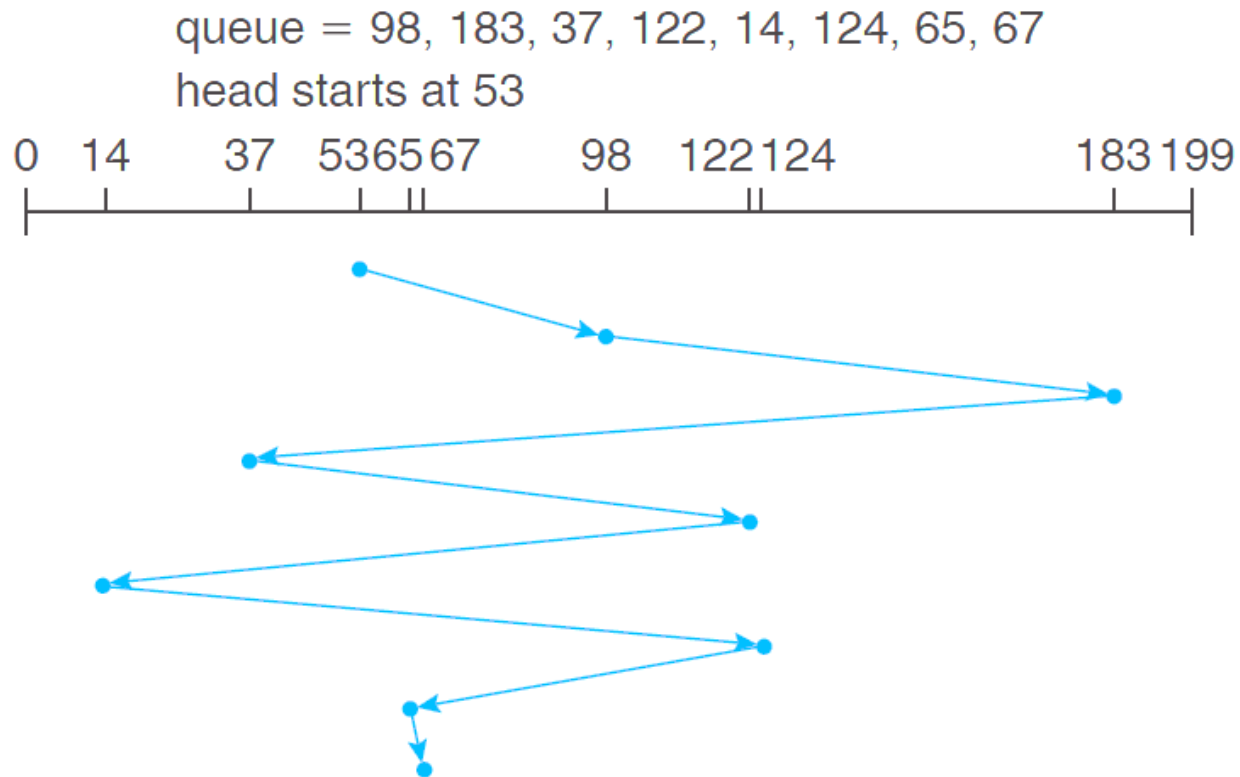
- Background

- **Seek Time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- The **Rotational Latency** is the additional time for the disk to rotate the desired sector to the disk head.
- The **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.

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Seek-Time-First (SSTF) Scheduling
- SCAN (Elevator Algorithm) Scheduling
- Circular SCAN (C-SCAN) Scheduling
- LOOK Scheduling

First-Come, First-Served (FCFS) Scheduling

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the disk head is initially at cylinder 53.

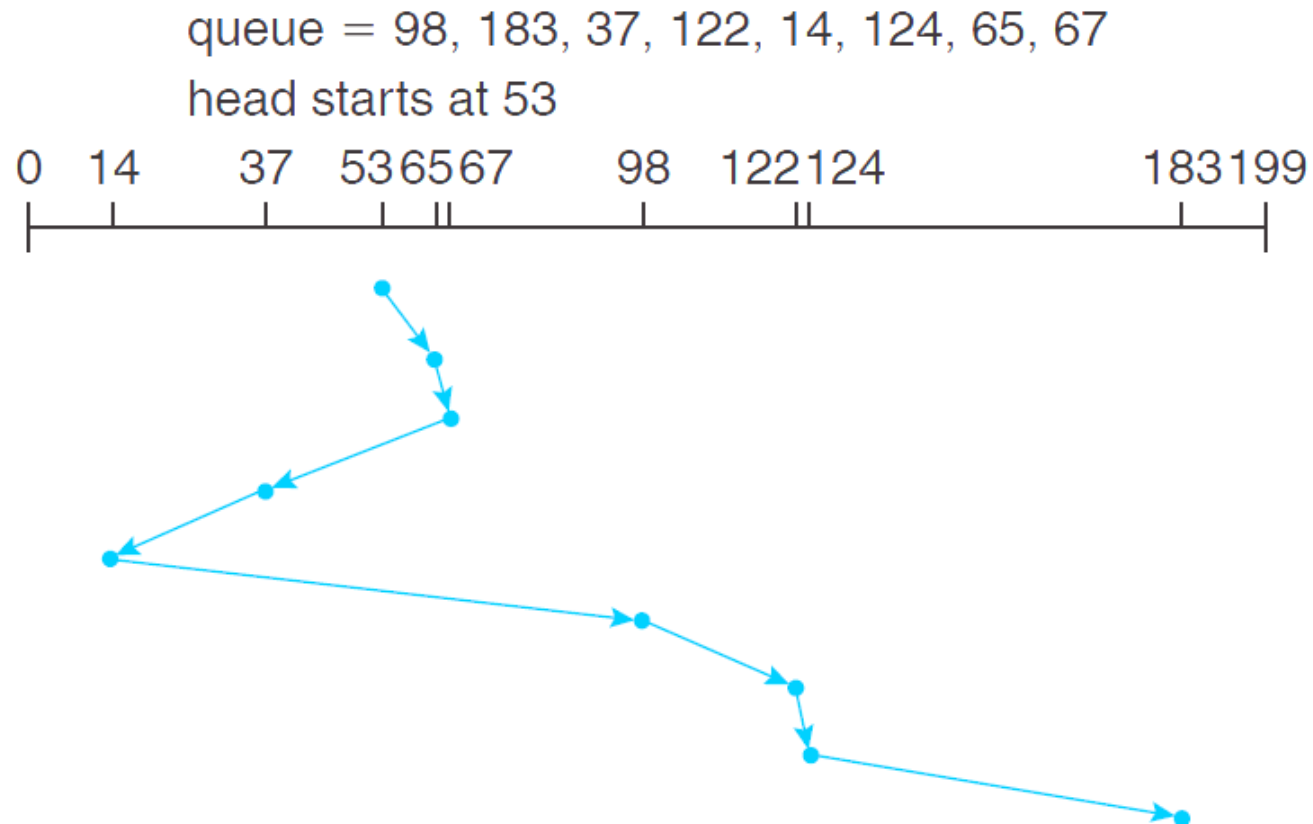


Shortest-Seek-Time-First (SSTF) Scheduling

- Service all the requests close to the current head position before moving the head far away to service other requests.
 - SSTF algorithm selects the request with the least seek time from the current head position.
 - Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
 - It may cause starvation of some requests.
 - The requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while the request from 14 is being serviced, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive.

SSTF Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the disk head is initially at cylinder 53.

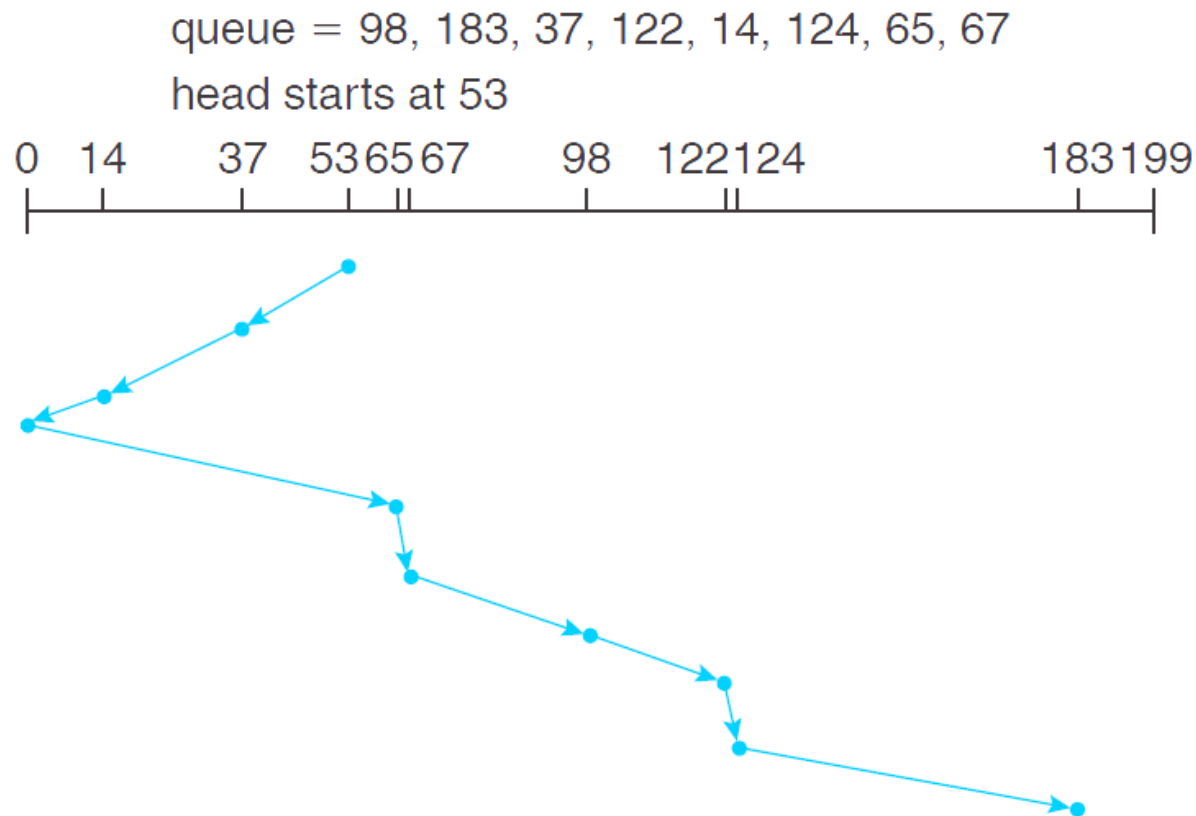


SCAN (Elevator) Scheduling

- The disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- The SCAN algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.
- The **direction of head movement** in addition to the head's current position is important.

SCAN (Elevator) Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the **disk head** is **initially at cylinder 53** and the **disk arm is moving toward 0**.

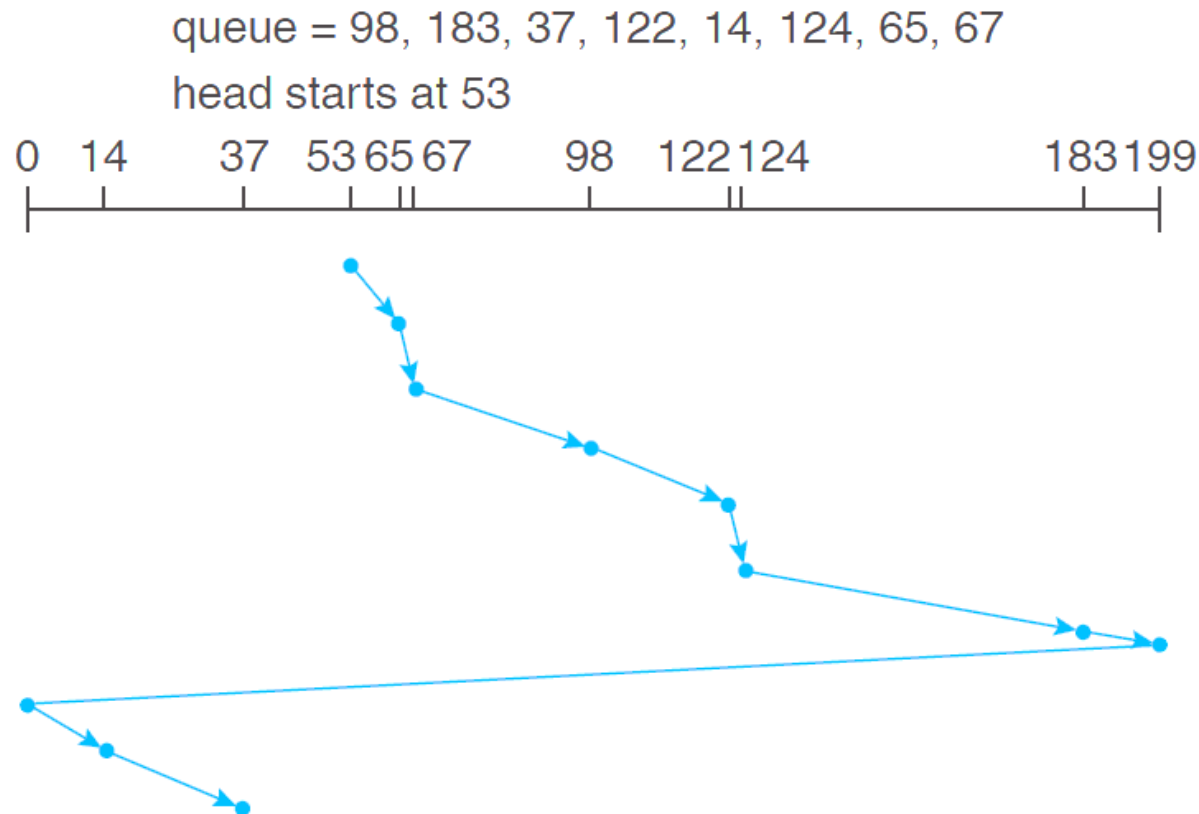


C-SCAN Scheduling

- Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time.
- Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.
- When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
- The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

C-SCAN Scheduling (contd...)

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the **disk head** is **initially at cylinder 53** and the **disk arm is moving toward 199**.

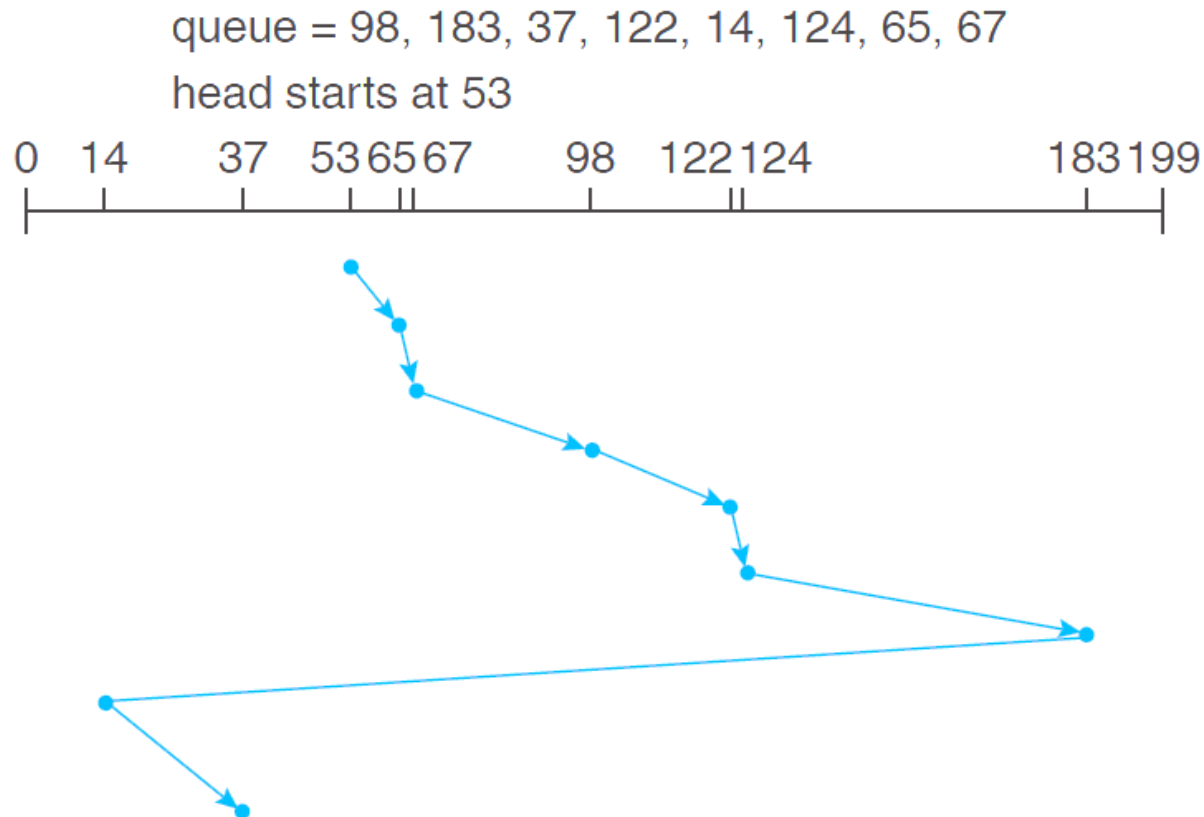


LOOK and C-LOOK Scheduling

- Both SCAN and C-SCAN move the disk arm across the full width of the disk.
 - In practice, neither algorithm is often implemented this way.
 - More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.
 - Versions of SCAN and C-SCAN that follow this pattern are called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in a given direction

C-LOOK Scheduling

Example: A disk queue requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Determine the total head movement (in cylinders) if the **disk head** is **initially at cylinder 53** and the **disk arm is moving toward 199**.



Question

Example: A disk queue requests for I/O to blocks on cylinders 82, 170, 43, 140, 24, 16, 190. Determine the total head movement, using the following algorithms, (in cylinders) if the disk head is initially at cylinder 50 and the disk arm is moving toward 199.

- FCFS
- SSTF
- SCAN
- C-SCAN
- LOOK
- C-LOOK

Device Management or I/O Management

- Computers operate many kinds of devices:
 - **Storage devices** (disks, tapes)
 - **Transmission devices** (network cards, modem)
 - **Human-interface devices** (screen, keyboard, mouse)
- To control the devices (which are connected to the Computer) is a **big concern because different I/O devices have different functions and different speed.**
- Different methods are required to control different devices.
- Different methods (required to control the devices) **form I/O Sub-system** for Kernel.
- **To encapsulate the details and oddities of different devices, the Kernel of an OS is structured to use device-driver modules.**
- The device drivers present a **uniform device-access interface** to I/O Sub-system.

Port, Bus, Controller

- The device communicates with the machine via a connection point or port.
 - Serial Port
 - Parallel Port
 - USB Port
 - HDMI Port
- If some devices use a common set of wires, then it is called a Bus.
 - A Bus is set of wires in which messages are conveyed by pattern of electrical voltage.
- A Controller is a collection of electronics that can operate a Port, Bus or a Device.

Character-Stream and Block Devices

- A character-stream device transfers bytes one-by-one
 - It can use a few bytes for their operations and it does not requires buffering.
 - The response time and processing speed are faster than the block devices.
 - In character oriented the I/O can be performed directly between the system and the user and as such, saves the kernel from the copying process and the buffering mechanisms overhead.
 - Interface → Character → Keyword → `put()`, `get()`.
- A block device transfer a block of bytes as a unit.
 - Block devices are storage devices that can provide data operations in fixed-size blocks for both reading and writing.
 - Hard drives, floppy disks, and optical drives, such as DVD-ROMs and CD-ROMs, are some examples of such machines.
 - To speed up the access during read and write operations it requires a buffering mechanism.
 - Interface → Block → Disk → `read()`, `write()`, `seek()`.

Blocking and Non-blocking I/O

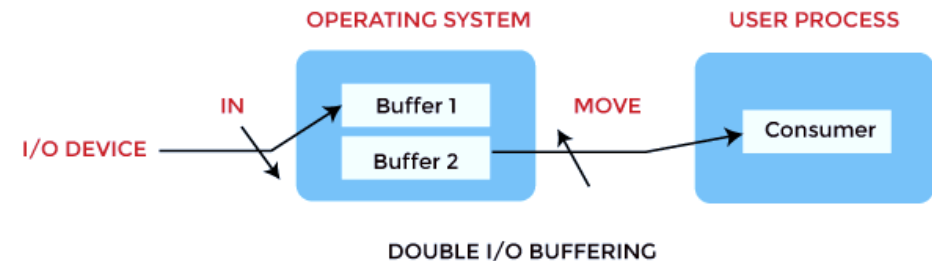
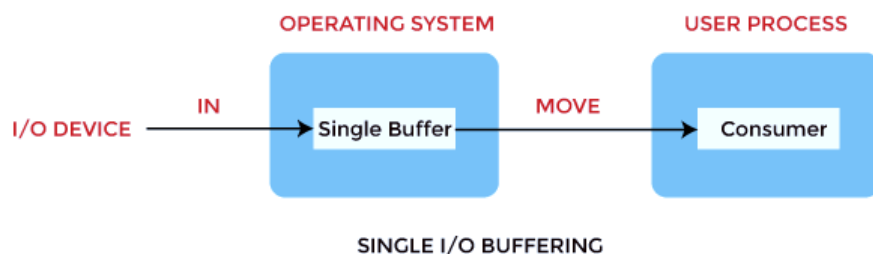
- Blocking I/O means that a given thread/process cannot do anything more until the I/O is fully received.
 - When an application/process issues blocking system call, the process execution stops.
 - **Example:** With blocking I/O, when a client makes a request to connect with the server, the thread that handles that connection is blocked until there is some data to read, or the data is fully written.
 - Until the relevant operation is complete that thread/process can do nothing else but wait in the wait queue.
- Some user-level processes perform non-blocking I/O.
 - **Example:** Receiving data from Keyboard or Mouse while displaying processed data on screen.
 - Video player is example of non-blocking I/O which reads data/frames from HDD and play the video on screen (after compression).

Blocking and Non-blocking I/O

- Blocking I/O means that a given thread/process cannot do anything more until the I/O is fully received.
 - When an application/process issues blocking system call, the process execution stops.
 - **Example:** With blocking I/O, when a client makes a request to connect with the server, the thread that handles that connection is blocked until there is some data to read, or the data is fully written.
 - Until the relevant operation is complete that thread/process can do nothing else but wait in the wait queue.
- Some user-level processes perform non-blocking I/O.
 - **Example:** Receiving data from Keyboard or Mouse while displaying processed data on screen.
 - Video player is example of non-blocking I/O which reads data/frames from HDD and play the video on screen (after compression).

Buffering and Double Buffering

- The **buffer** is an area in the main memory used to store or hold the data temporarily.
 - In other words, buffer temporarily stores data transmitted from one place to another, either between two devices or an application.
 - The act of storing data temporarily in the buffer is called **buffering**.
 - **Buffering helps in matching speed between two devices in which the data is transmitted.**
 - **It helps the devices with different sizes of data transfer to get adapted to each other.**
- ✓ In computer networking, the large message is fragmented into small fragments and sent over the network. The fragments are accumulated in the buffer at the receiving end and reassembled to form a complete large message.



I/O Channels

- A channel is an independent hardware component that co-ordinates all I/O to a set of controllers.
 - Channels use separate, independent and low-cost processors for its functioning which are called **Channel Processors**.
 - Each channel supports one or more controllers or devices.
 - Channel programs contain list of commands to the channel itself and for various connected controllers or devices.
 - **When OS want to perform I/O, the OS initiates an I/O machine instruction for the channel and then rest of the task to complete the I/O is performed by the channel.**
 - **Multiplexer** (Byte Multiplexer and Block Multiplexer)
 - **Selector**



Disk Management

- Disk Formatting
- Boot Block
- Bad Blocks

Disk Formatting (Physical)

- A new magnetic disk is a blank slate (just a platter of a magnetic recording material).
- To store data, it must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting**, or **physical formatting**.
- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector typically consists of a **header**, a **data area** (usually 512 bytes in size), and a **trailer**.
- The header and trailer contain information used by the disk controller, such as a **sector number** and an **error-correcting code (ECC)**.
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- The controller automatically does the ECC processing whenever a sector is read or written.

Disk Formatting (Logical)

- To use a disk to hold files, the OS still needs to record its own data structures on the disk.
- It does so in two steps:
 - The first step is to partition the disk into one or more groups of cylinders.
 - ✓ The OS can treat each partition as though it were a separate disk.
 - ✓ For instance, one partition can hold a copy of the OS's executable code, while another holds user files.
 - After partitioning, the second step is logical formatting (or creation of a file system).
 - ✓ In this step, the OS stores the initial file-system data structures onto the disk.
 - ✓ These data structures may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

Free-Space Management

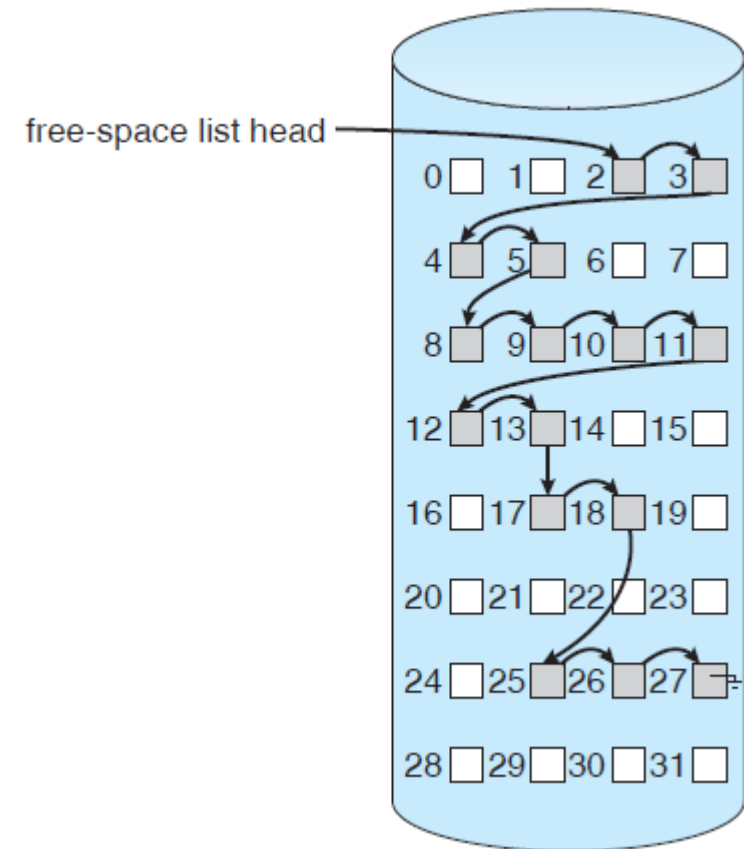
- To keep track of free disk space, the system maintains a **free-space list**. The free-space list records all free disk blocks.
 - To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
 - This space is then removed from the free-space list.
 - When a file is deleted, its disk space is added to the free-space list.
- **Approaches for Free-Space Management**
 - **Bit Vector**
 - **Linked List**
 - **Grouping**
 - **Counting**

Free-Space Management: Bit Vector

- Each block is represented by 1 bit.
 - If the block is free, the bit is 1.
 - If the block is allocated, the bit is 0.
- **Example:** Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map or bit vector would be:
 - 001111001111110001100000011100000 ...
- Bit vectors are inefficient unless the entire vector is kept in main memory.
- Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.

Free-Space Management: Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
 - This first block contains a pointer to the next free disk block, and so on.
 - OS simply needs a free block so that it can allocate that block to a file, so **the first block in the free list is used.**
 - **It is not efficient to give the address of a large number of free blocks since traversing in linked-list is sequential.**



Free-Space Management: Grouping

- This approach stores the addresses of n free blocks in the first free block.
 - The first $n-1$ of these blocks are actually free.
 - The last block contains the addresses of another n free blocks, and so on.
 - The addresses of a large number of free blocks can now be found quickly unlike the situation when the standard linked-list approach is used.

Free-Space Management: Counting

- Generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
 - Thus, rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block.
 - Each entry in the free-space list then consists of a disk address and a count.
 - These entries can be stored in a B-tree, rather than a linked list, for efficient lookup, insertion, and deletion.

RAID Structure

- RAID - Redundant Arrays of Independent Disks (RAIDs)
- A **variety of disk-organization techniques**, collectively called RAIDs are commonly used to address the **performance** and **reliability issues**.
 - The solution to the problem of reliability is to introduce **redundancy (duplicate every disk)**.
 - The technique of duplicating every disk is called **mirroring**.
 - With mirroring, a logical disk consists of **two physical disks**, and every write is carried out on both disks. **The result is called a mirrored volume.**

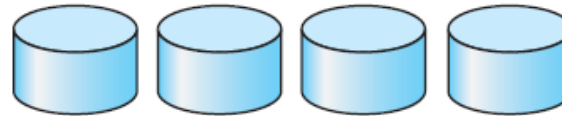
RAID Structure (contd...)

- Parallel access to multiple disks improves performance.
 - The number of reads per unit time has doubled.
 - The transfer rate of each read is the same as in a single-disk system.
 - **With multiple disks, we can improve the transfer rate as well by striping data across the disks.**
 - Data striping consists of splitting the bits of each byte across multiple disks; such striping is called bit-level striping.
 - In block-level striping, for instance, blocks of a file are striped across multiple disks; with n disks, block i of a file goes to disk $(i \bmod n) + 1$.
 - Block-level striping is the most common.

RAID Levels

- Mirroring provides high reliability, but it is expensive.
- Striping provides high data-transfer rates, but it does not improve reliability.
- There are various schemes to provide redundancy:

- **RAID Level 0.** RAID level 0 refers to disk arrays with striping at the level of blocks but without any redundancy.



- **RAID Level 1.** RAID level 1 refers to disk mirroring.



- Other levels – Students should cover through home assignment.