FILE-SYSTEM INTERFACE

OUTLINE

- FILE CONCEPT
- ACCESS METHODS
- DISK AND DIRECTORY STRUCTURE

OBJECTIVES

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

WHAT IS FILE?

- A file is a unit of storing data on a secondary storage device such as a hard disk or other external media.
- A collection of related information defined by its creator.
- Every file has a name and its data.
- Operating system associates various information with files. For example the date and time of the last modified file and the size of the file etc.
- This information is called as files attribute or metadata.
- The attributes varies considerably from system to system.
- In general file is a collection of bits, bytes and records.

FILE CONCEPT

- Contiguous logical address space
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider
    - **Text file:sequence of character organized into lines. Ex .Txt**
    - **Source file:sequence of subroutine and functions. Ex .C, .Java**
    - **Object file:collection of words, organized into loader record blocks. Ex .Obj**

FILE ATTRIBUTES

DETAILED INFORMATION ABOUT THE FILE STORED IN DIRECTORY STRUCTURE

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk

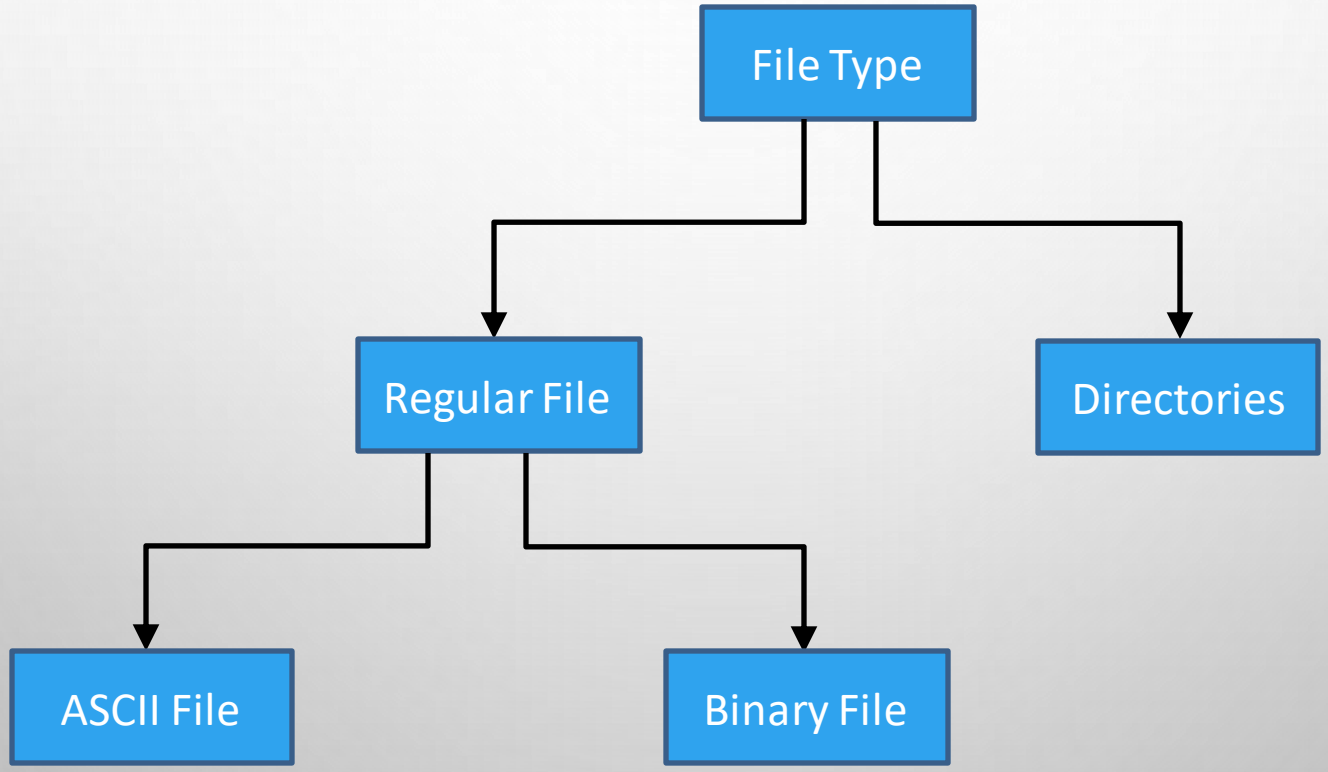
FILE OPERATIONS

- File is an **abstract data type**
- **Create-** There should be space in file system, entry for new file
- **Write** –Name of the file**, write pointer** location
- **Read** – name of the file**, read pointer** location
- **Reposition within file** – no actual I/O operation**. Seek**
- **Delete-**delete file from file system entry
- **Truncate-**deleting the content of the file
- ***Open*(f_i)** – search the directory structure on disk for entry f_i, and move the content of entry to memory
- ***Close*** (f_i) – move the content of entry f_i in memory to directory structure on disk

OPEN FILE LOCKING

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

TYPES OF FILES



REGULAR FILE VS DIRECTORIES

- **Regular File:**
  - Regular files are the ones that contains user information.
  - These may have text, databases or executable program.
  - The user can apply various operation on such files like add, modify, delete or even remove the entire file.
- **Directories:**
  - Directories are system files for maintaining the structure of the file system.
  - To keep track of files, file system normally have directories or folder.

DIFFERENT TYPES OF FILES

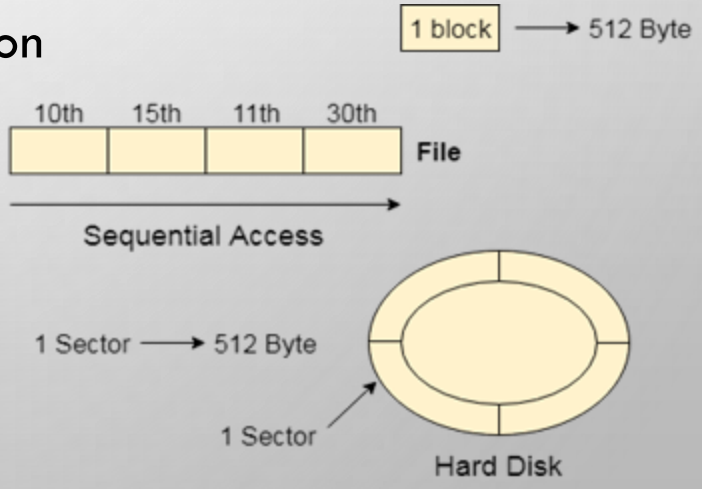
FILE TYPE	USUAL EXTENSION	FUNCTION
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, c++	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information

FILE ACCESS METHODS

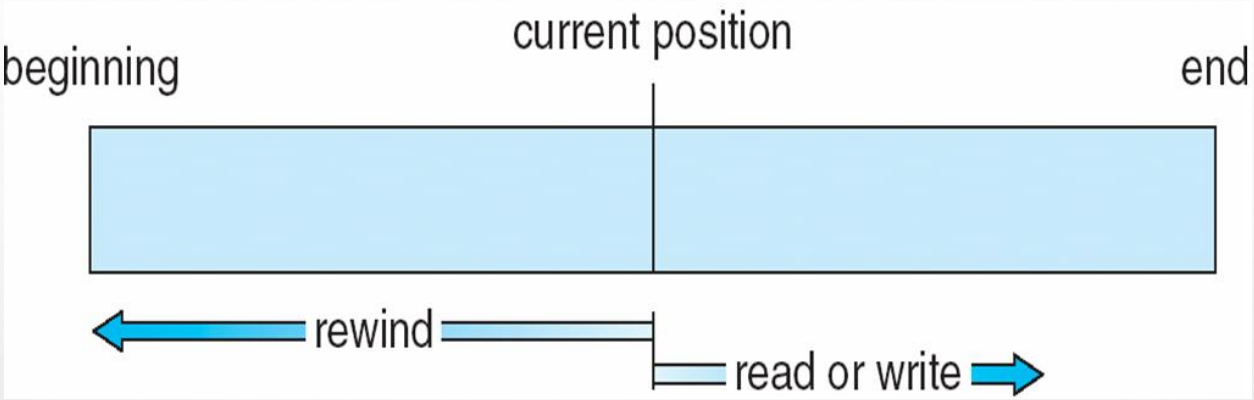
- **SEQUENTIAL ACCESS**
- **DIRECT/RANDOM ACCESS**
- **INDEXED ACCESS**

SEQUENTIAL ACCESS

- Emulates magnetic tape operation
- One record is processed after the other
- Editors and compilers usually access the file in this manner
- Supports following operations:
  - Read next: read record and move pointer to next position
  - Write next: write and advance the position
  - Rewind: moving back to the earlier location



SEQUENTIAL-ACCESS FILE



Sequential Access

```
read next
write next
reset
```

no read after last write
(rewrite)

DIRECT ACCESS

- IT IS BASED ON DISK MODEL
- ALSO KNOWS AS DIRECT ACCESS METHOD OR RELATIVE ACCESS METHOD.
- THERE IS NO RESTRICTION ON THE ORDER OF READING AND WRITING FOR A DIRECT ACCESS FILE.
- FILE WHOSE BYTES OR RECORDS CAN BE READ IN ANY ORDER ARE CALLED RANDOM ACCESS FILES.
- ESSENTIAL FOR MANY APPLICATIONS, FOR EXAMPLE: DATABASE SYSTEM
- FOLLOWING OPERATIONS ARE SUPPORTED:
  - **READ N:** READING RECORD 'N'
  - **WRITE N:**WRITING RECORD 'N'
  - **JUMP TO RECORD N:**JUMP TO RECORD NUMBER 'N'

- **DIRECT ACCESS –** FILE IS FIXED LENGTH **LOGICAL RECORDS**

```
READ N
WRITE N
POSITION TO N
      READ NEXT
      WRITE NEXT
REWRITE N
```

  N = **RELATIVE BLOCK NUMBER**

- RELATIVE BLOCK NUMBERS ALLOW OS TO DECIDE WHERE FILE SHOULD BE PLACED

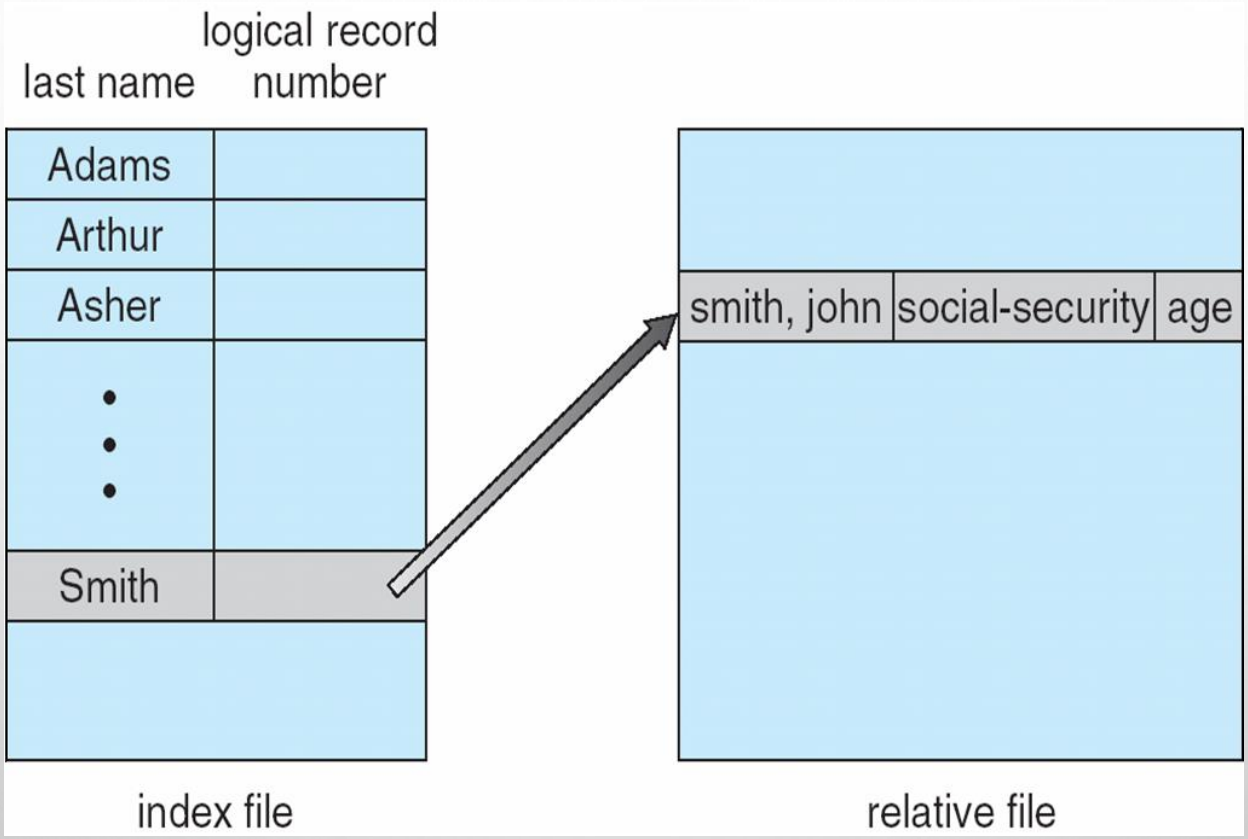
SIMULATION OF SEQUENTIAL ACCESS ON DIRECT-ACCESS FILE

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

OTHER ACCESS METHODS

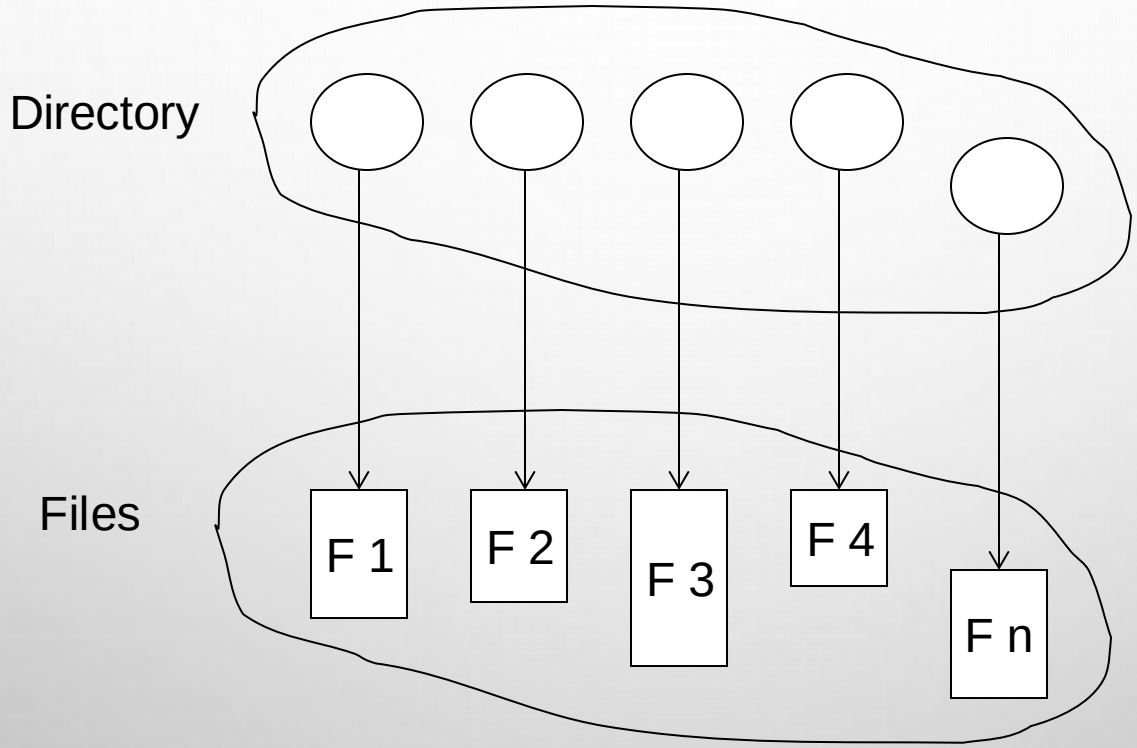
- Can be built on top of base methods
- General involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)

EXAMPLE OF INDEX AND RELATIVE FILES



DIRECTORY STRUCTURE

- A COLLECTION OF NODES CONTAINING INFORMATION ABOUT ALL FILES

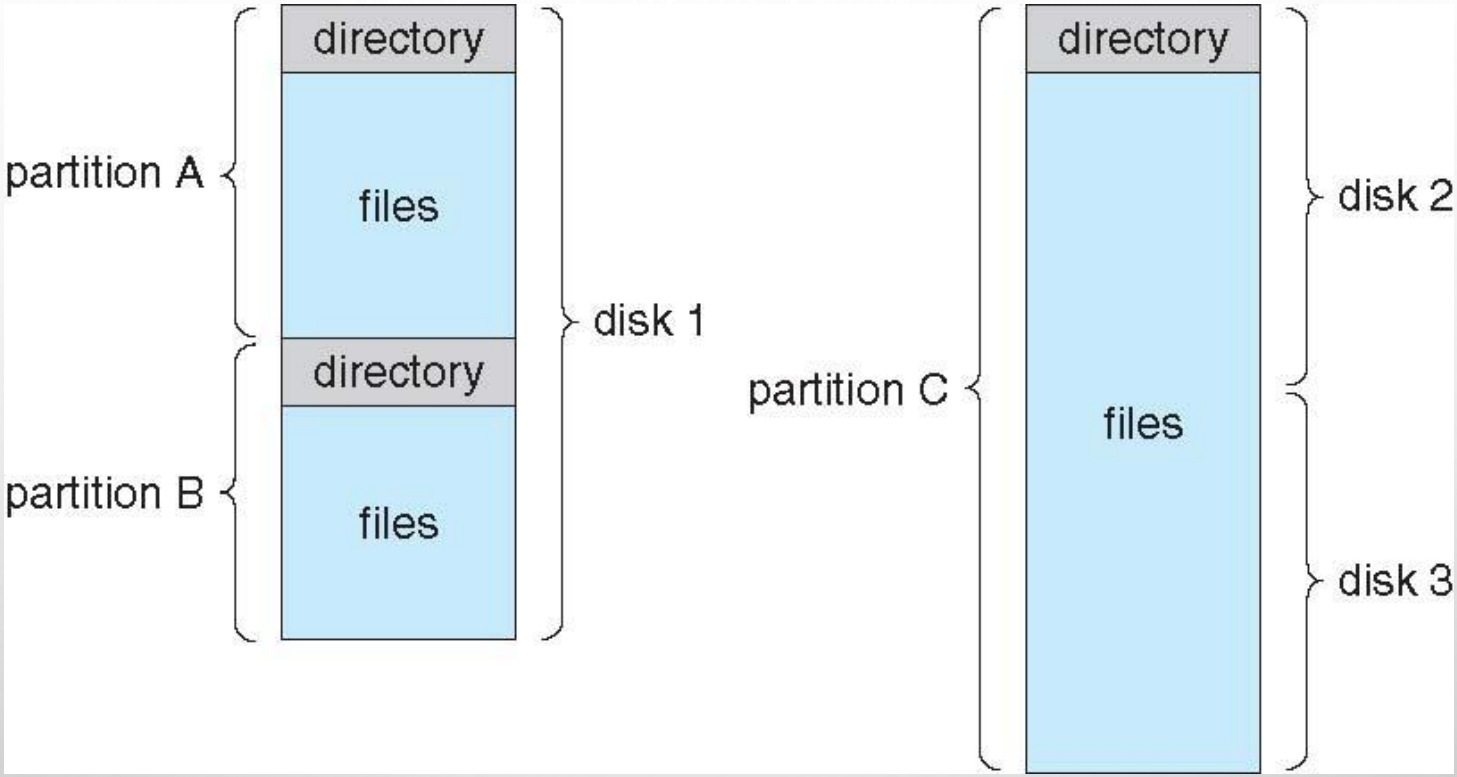


Both the directory structure and the files reside on disk

DIRECTORY STRUCTURE

- **FILE DIRECTORIES**
  - Collection of files is a file directory.
  - The directory contains information about the files, including attributes, location and ownership.
  - A directory can be viewed as a symbol table that translates file name into their entries.
  - Directory can be organized in many ways.
- **Advantages of maintaining directories are:**
  - **Efficiency:** A file can be located more quickly.
  - **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
  - **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

A TYPICAL FILE-SYSTEM ORGANIZATION

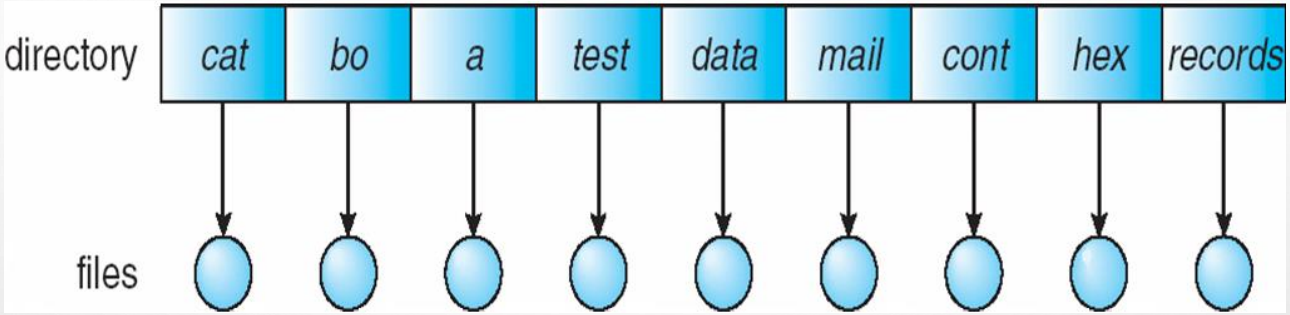


OPERATIONS PERFORMED ON DIRECTORY

- SEARCH FOR A FILE
- CREATE A FILE
- DELETE A FILE
- LIST A DIRECTORY
- RENAME A FILE
- TRAVERSE THE FILE SYSTEM

SINGLE-LEVEL DIRECTORY

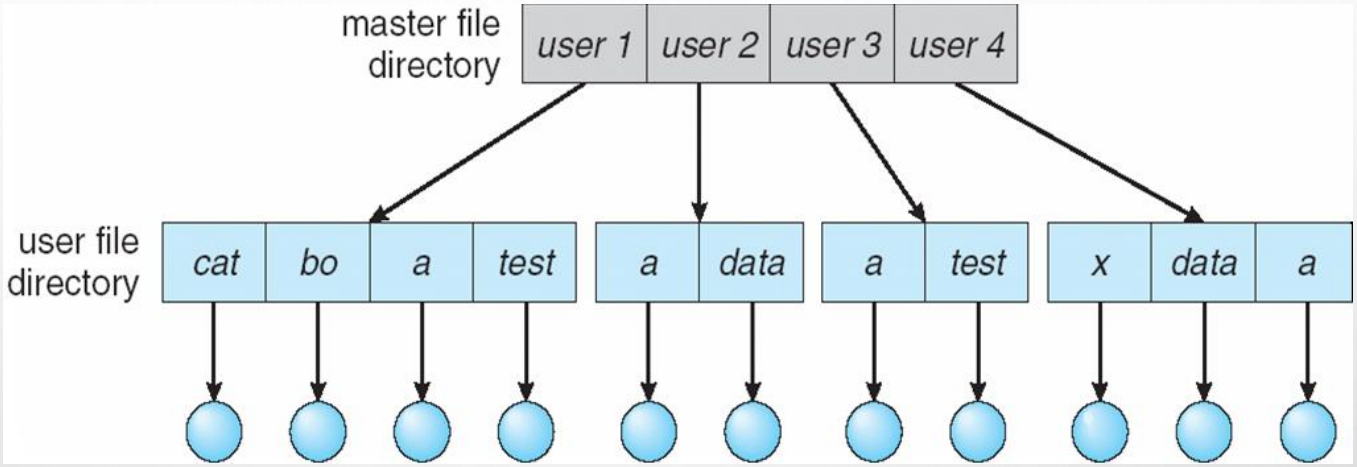
- A SINGLE DIRECTORY FOR ALL USERS



- Simplest directory structure
- All files are contained in the same directory..
- **Naming problem:** users cannot have same name for two files.
- **Grouping problem:** users cannot group files according to their need.

TWO-LEVEL DIRECTORY

- SEPARATE DIRECTORY FOR EACH USER



- Each user has its own user file directory(UFD)
- UFD's have similar structures, but each lists only the files of a single user.
- When user logs in, the system's master file directory is searched.
- MFD is indexed by user name or account number and each points to UFD for that user.

TWO-LEVEL DIRECTORY

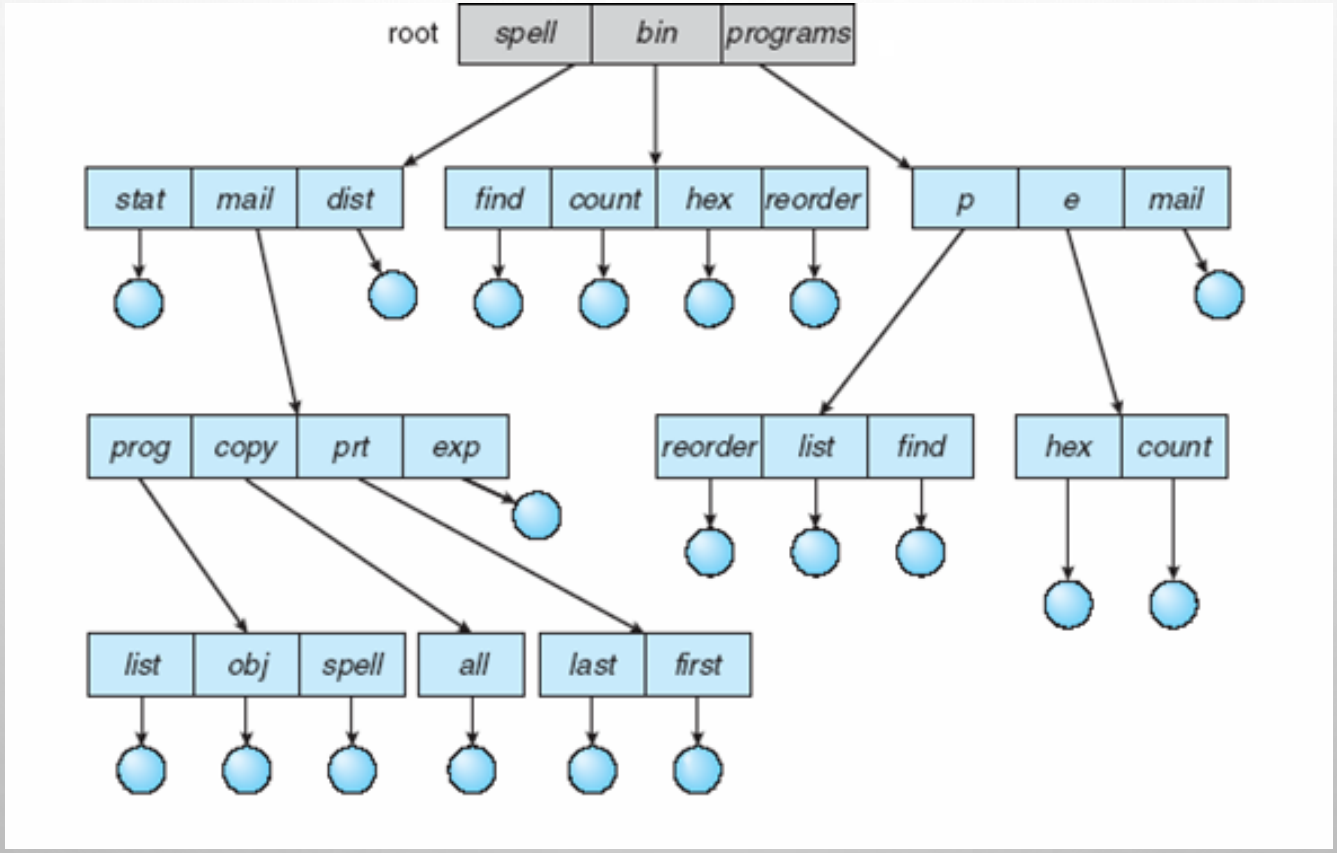
- In this separate directories for each user is maintained.
- **Path name:** due to two levels there is a path name for every file to locate that file.
- Now, we can have same file name for different user.
- Searching is efficient in this method.

Problem:

- When user wants to cooperate for on some task and to access one another's files.
  - If access has to be permitted one user must have the ability to name the file in another user directory.

TREE-STRUCTURED DIRECTORIES

- Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



TREE-STRUCTURED DIRECTORIES

- It allows user to create their own directories and organize their files accordingly.
- Every file in the system has unique path
- A directory contain set of files and subdirectories.
- With a tree structured directory system, user can access in addition to their file, the files of other users.

TREE-STRUCTURED DIRECTORIES (CONT.)

- EFFICIENT SEARCHING
- GROUPING CAPABILITY
- CURRENT DIRECTORY (WORKING DIRECTORY)
  - **`CD /SPELL/MAIL/PROG`**
  - **`TYPE LIST`**

FILE LOCATION

- **Absolute Path Name:**
  - An absolute path name consisting of the path from the root directory to the file.
- **Relative Path Name:**
  - User can designate one directory as current working directory, in which case, all path names not beginning at the root directory are taken relative to working directory.

TREE-STRUCTURED DIRECTORIES (CONT.)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

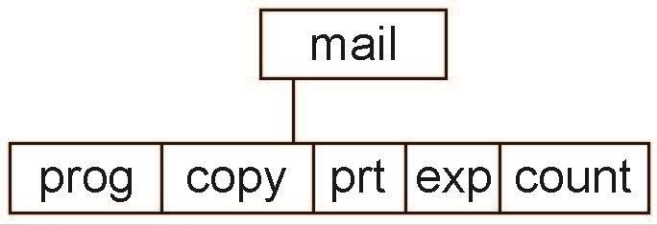
```
RM <FILE-NAME>
```

- Creating a new subdirectory is done in current directory

```
MKDIR <DIR-NAME>
```

Example: if in current directory **/MAIL**

```
MKDIR COUNT
```



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

ACCESS LISTS

- Specify the user name and the type of access allowed for each user

PROBLEMS:

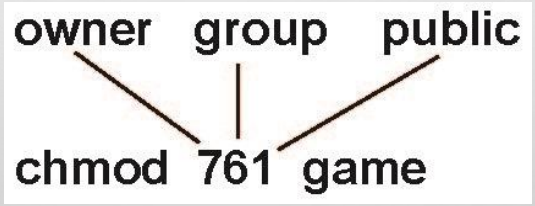
- LENGTH
- CONSTRUCTING SUCH A LIST IS TEDIOUS
- VARIABLE SIZE DIRECTORY ENTRY

ACCESS LISTS AND GROUPS

- MODE OF ACCESS: READ, WRITE, EXECUTE
- THREE CLASSES OF USERS ON UNIX / LINUX

			RWX
A) **OWNER ACCESS**	7	⇒	1 1 1
			RWX
B) **GROUP ACCESS**	6	⇒	1 1 0
			RWX
C) **PUBLIC ACCESS**	1	⇒	0 0 1

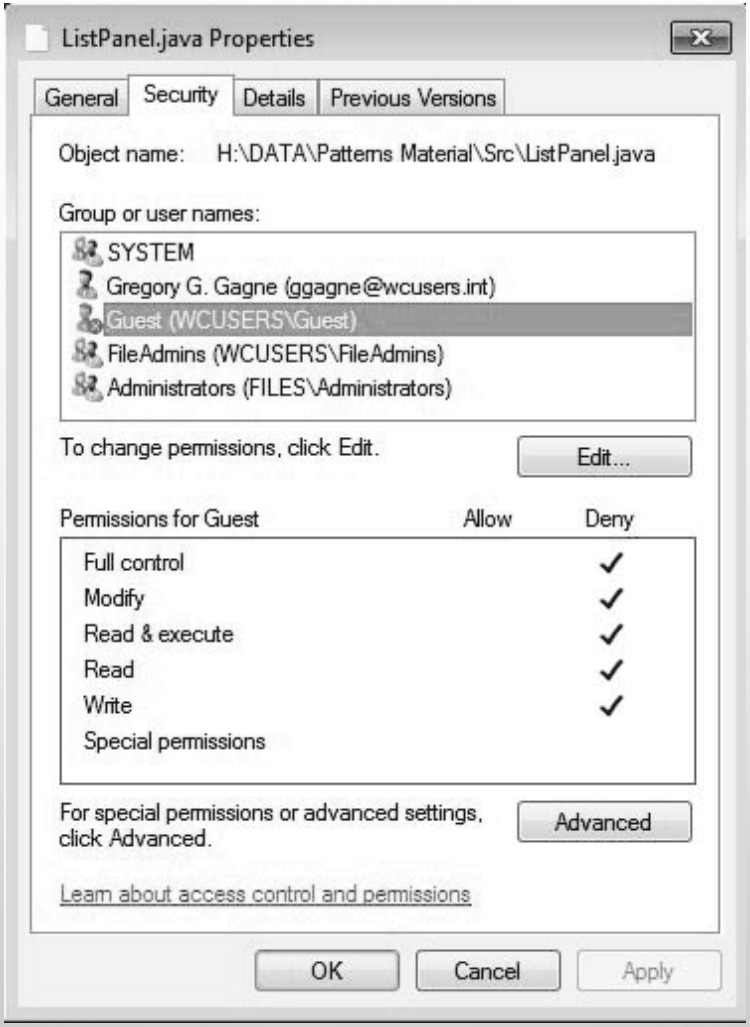
- ASK MANAGER TO CREATE A GROUP (UNIQUE NAME), SAY G, AND ADD SOME USERS TO THE GROUP.
- FOR A PARTICULAR FILE (SAY *GAME*) OR SUBDIRECTORY, DEFINE AN APPROPRIATE ACCESS.



Attach a group to a file

```
chgrp       G       game
```

WINDOWS 7 ACCESS-CONTROL LIST MANAGEMENT



A SAMPLE UNIX DIRECTORY LISTING

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

THANK YOU

File System Implementation

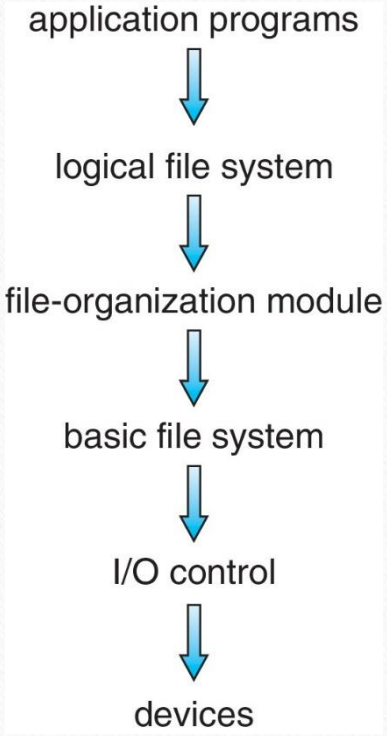
Objectives:

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** (**FCB**) – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

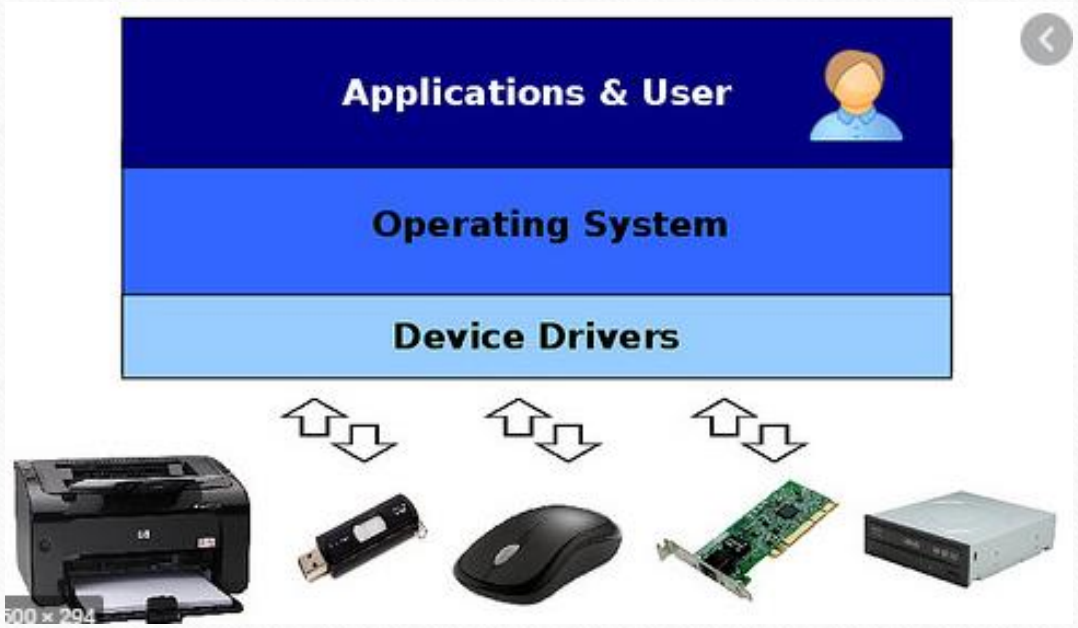
Layered File System



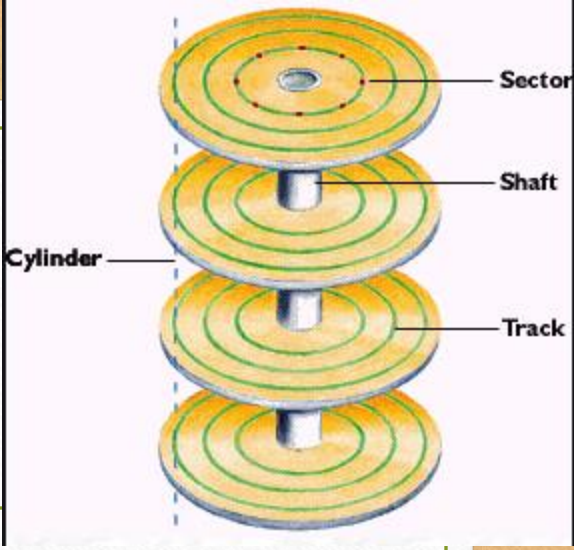
Device drivers

- **Device drivers** manage I/O devices at the I/O control layer
  - Can be thought of as a translator
  - Given commands like "retrieve block 123" outputs low-level hardware specific commands to hardware controller
  - Interfaces I/O devices to the rest of the system.
  - Tell the controller which device location to act on and what action to take

Device drivers



Basic file system



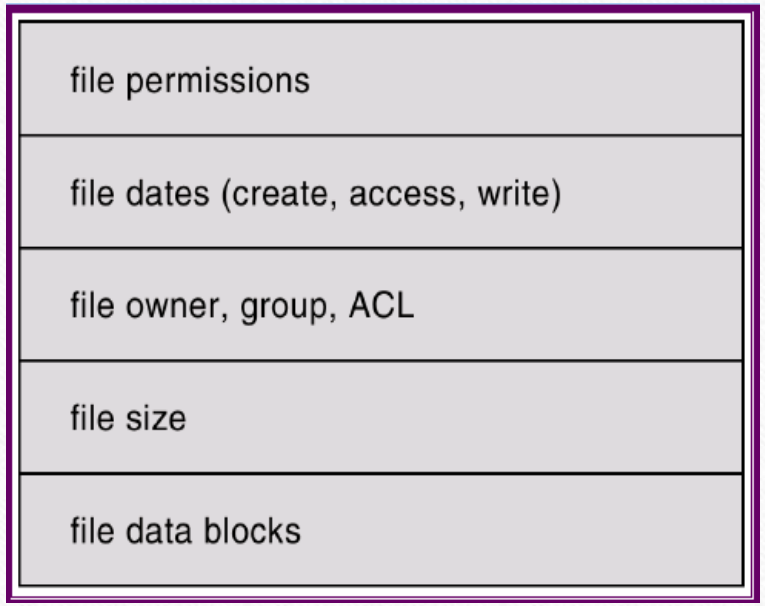
- Issues generic commands to appropriate device driver to read and write physical block on the disk.
- Each physical block is identified by its numeric disk address(For example , drive1, cylinder 72, track 2, sector 10).
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data

File organization module

- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

Logical file system

- **Logical file system** manages metadata information
  - Directory management
  - Manage file structure using FCB(file control block)
  - Protection and security



File-System Implementation

- Operating system implement open and close system calls for processes to implement to request access to file content.
- File system can be implemented
  - **On-disk**-file system contain information about how to boot operating system, total number of blocks, the number and location of the free blocks, the directory structure and individual files
  - **In-memory** used for both file system management and performance improvement via caching

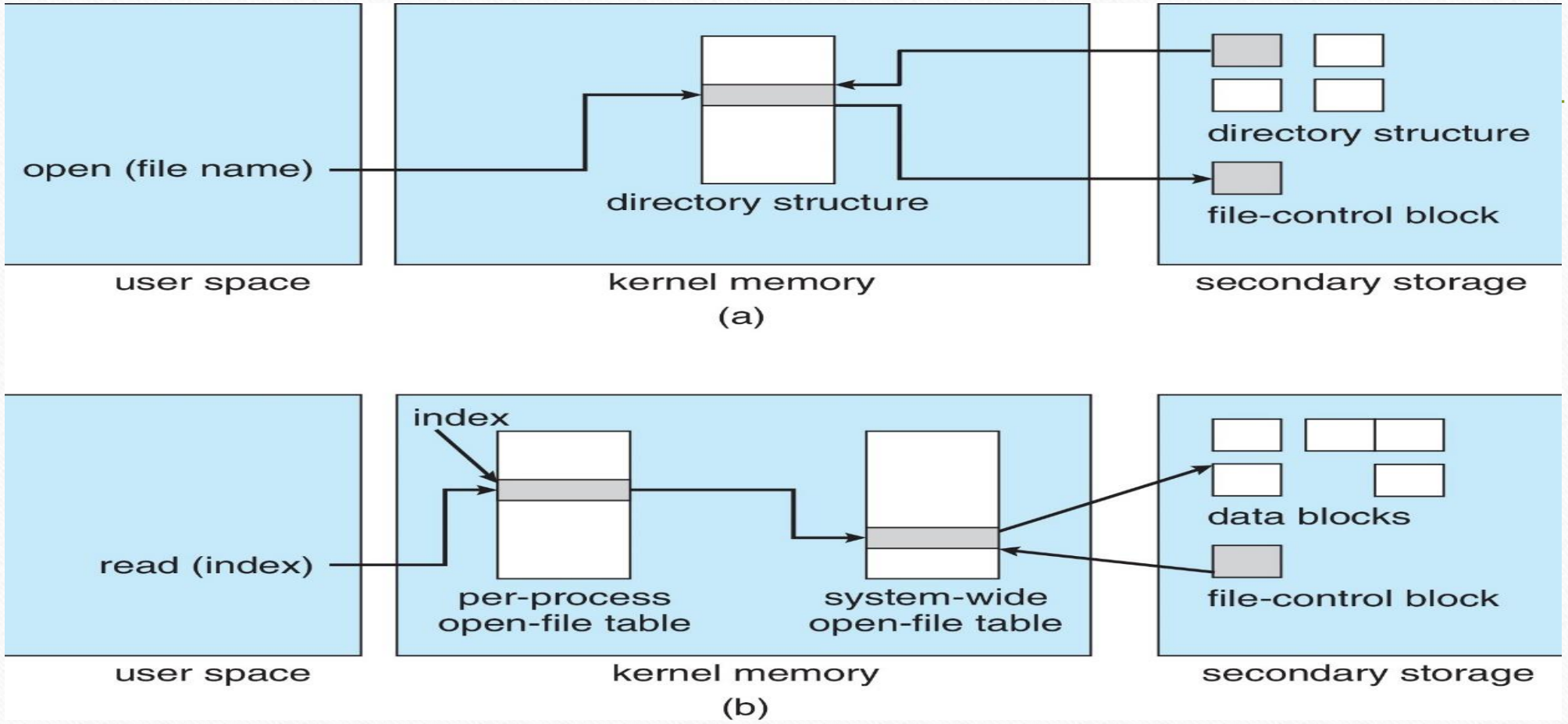
On-disk Structure

- **Boot control block** contains info needed by system to boot OS from that partition. Needed
- **Partition control block (superblock, master file table)**contains partition detail ,such as number of blocks in the partition ,free-block count and free block pointers.
- **Directory structure** organizes the files
- **File Control Block** (**FCB**) contains many file details including file permission, size and location of the data blocks.

In-memory Structure

- Contain information about each **mounted partition**
- Hold directory information of recently accessed directories
- **system-wide open-file table** contains a copy of the FCB of each file and other info
- **per-process open-file table** contains pointers to appropriate entries in system-wide open-file table as well as other information(current location in the file, access mode in which file is opened)

In-Memory File System Structures



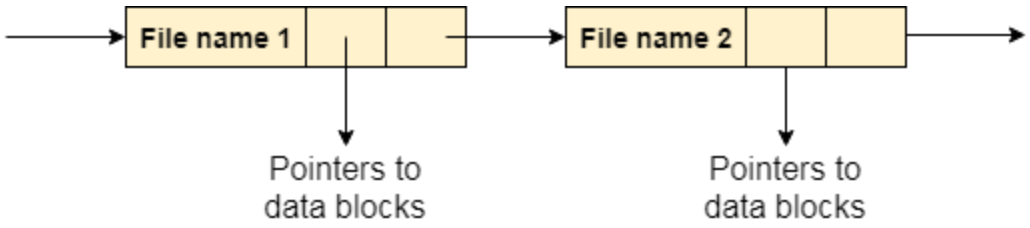
Implementing File System

Topics to be covered

- Directory Implementation
- Allocation Methods
- Free Space Management

Directory Implementation

- **Linear list:**
  - All the files in a directory are maintained as singly lined list.
  - Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

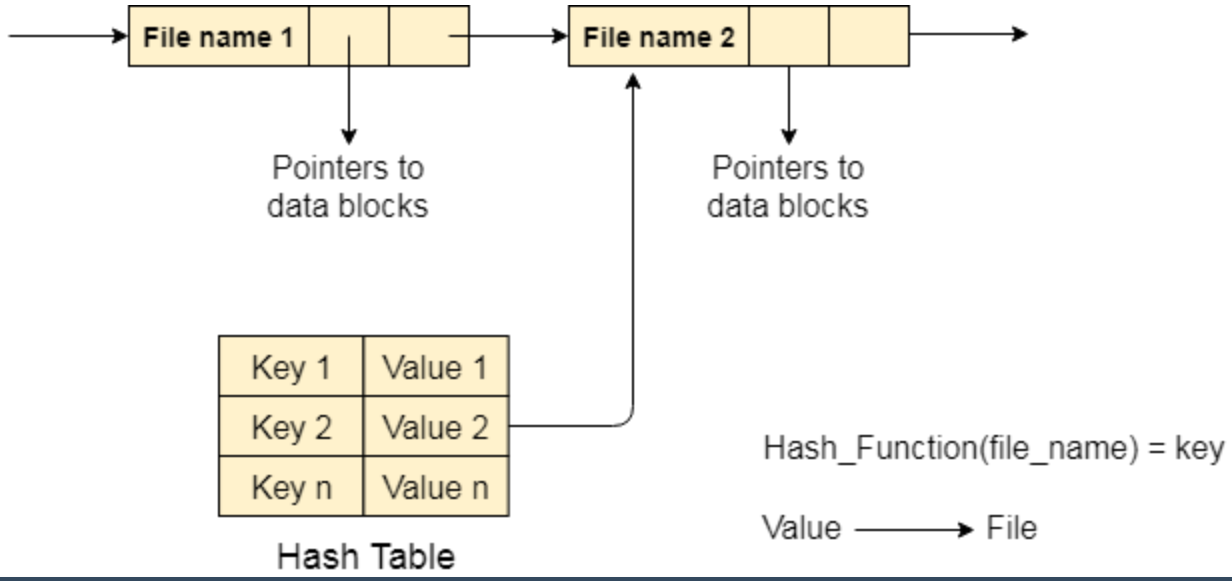


Linear List

  - The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

Directory Implementation

- **Hash table:**
  - This approach suggests to use hash table along with the linked lists.
  - A key-value pair for each file in the directory gets generated and stored in the hash table.
  - Searching becomes efficient compare to linear list.



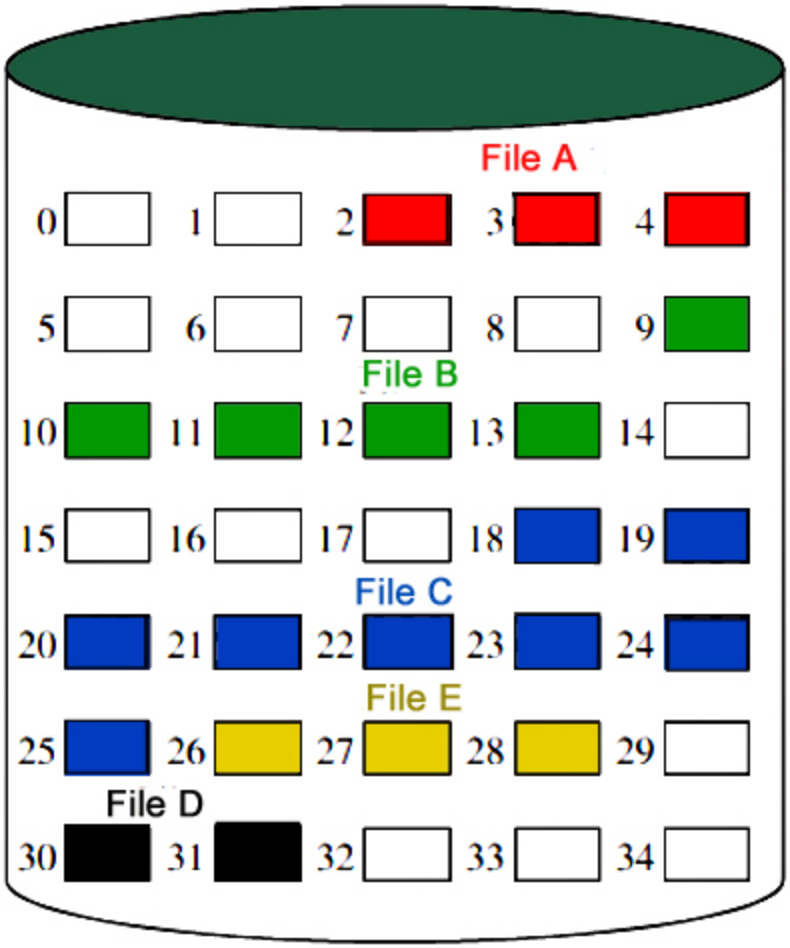
File Allocation

- Various methods to implement files is listed below,
  1. Continuous Allocation
  2. Linked Allocation(Non-contiguous allocation)
  3. Indexed Allocation

Continuous Allocation

- A single continuous set of blocks is allocated to a file at the time of file creation.
- It is simple to implement because we need to keep track of only the first block and the number of blocks in the file.
- Read performance is excellent because entire file can be read from the disk in a single operation.
  - Only one seek is needed to the first block.
- Once the disk is filled, reusing the space requires maintaining a list of hole.
  - However, when a new file is to be created, it is necessary to know its final size in order to choose a hole of the current size to place it in.

Continuous Allocation



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Continuous Allocation

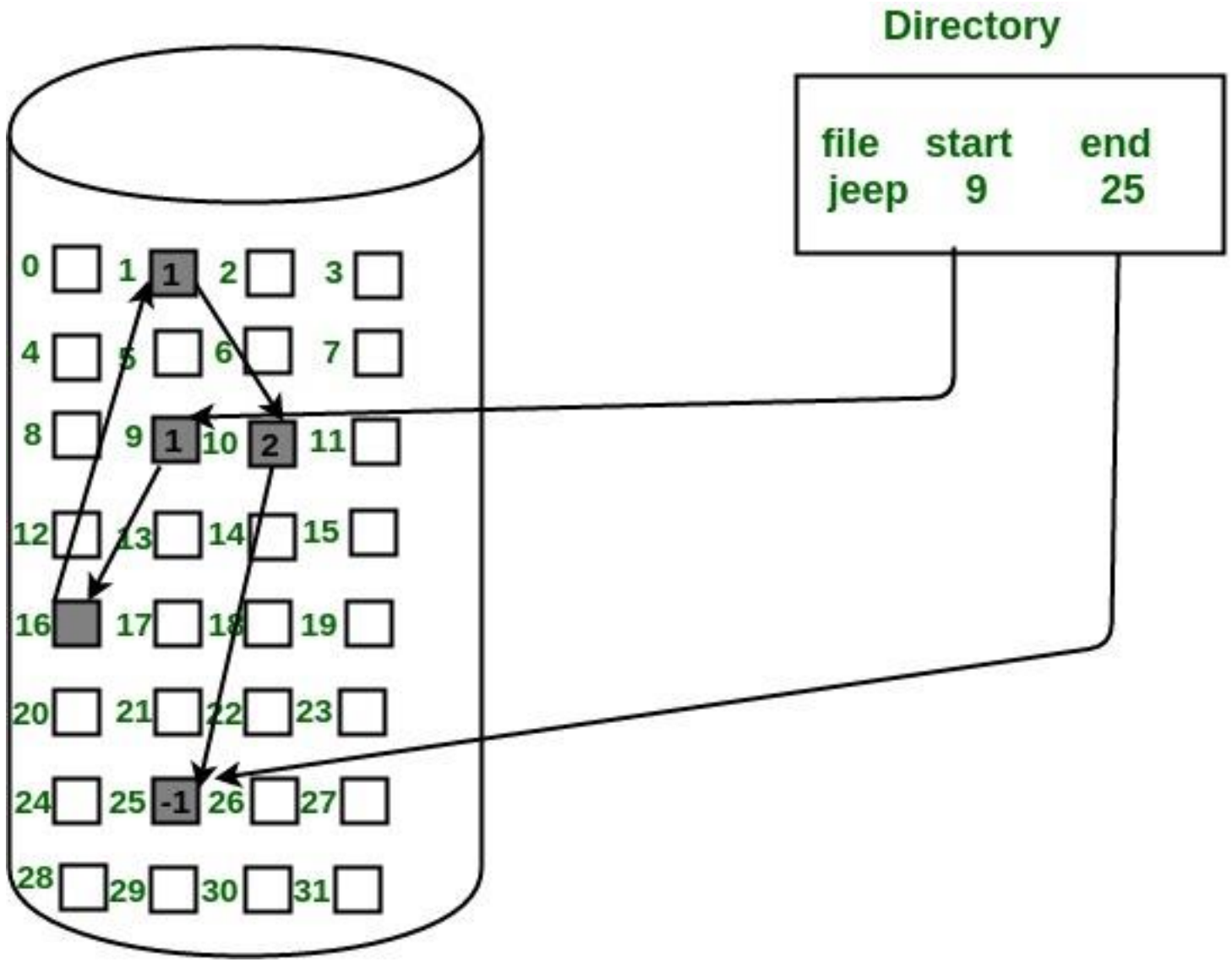
- **Advantages:**
  - Both the Sequential and Direct Accesses are supported by this.
  - This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.
- **Disadvantages:**
  - This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
  - Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked Allocation(Non-contiguous allocation)

- Each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.

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

Linked Allocation(Non-contiguous allocation)



Linked Allocation(Non-contiguous allocation)

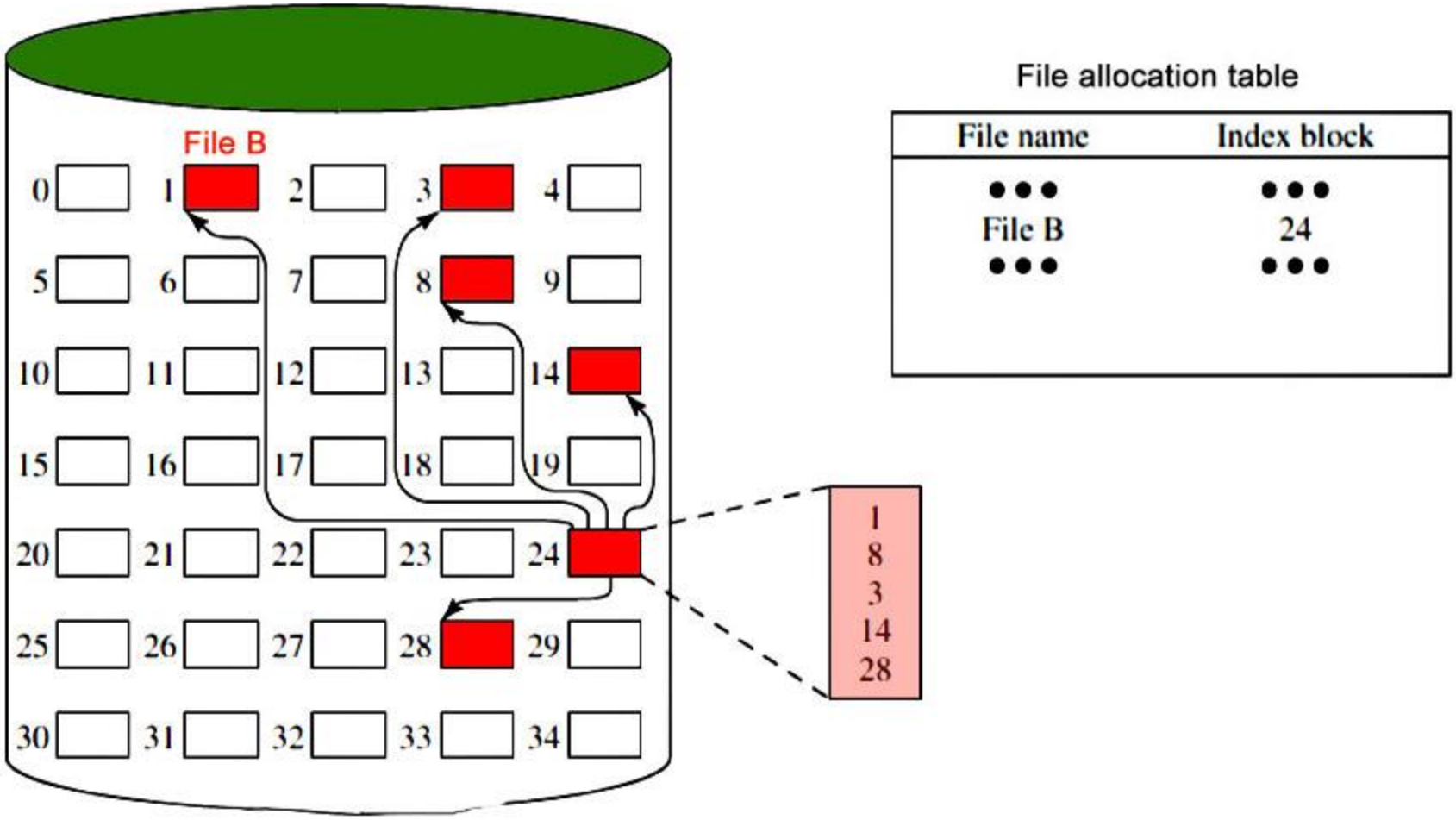
- **Disadvantages:**
  - Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
  - It does not support random or direct access. We can not directly access the blocks of a file only sequential access is possible.
  - Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

- A special block known as the Index block contains the pointers to all the blocks occupied by a file.
- Each file has its own index block.

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

Indexed Allocation



File allocation table

File name	Index block
•••	•••
File B	24
•••	•••

Indexed Allocation

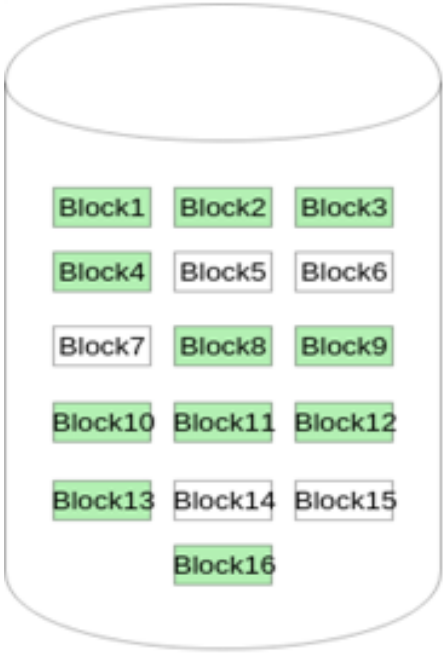
- **Disadvantages:**
  - The pointer overhead for indexed allocation is greater than linked allocation.

Free Space Management

- Techniques to manage free space are:
  1. Bitmap or Bit vector
  2. Linked list
  3. Grouping

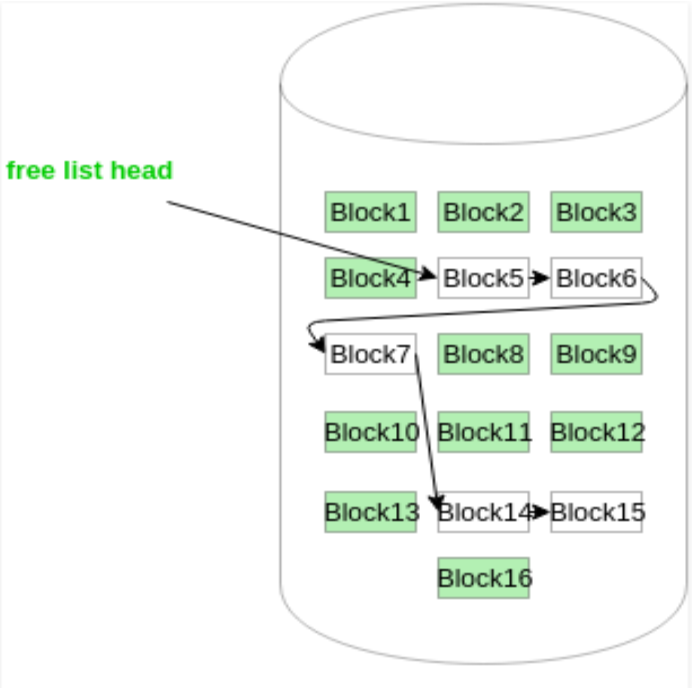
Bitmap or Bit Vector

- A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block.
- If the block is free the bit is set to 1.
- If the block is allocated the bit is set to zero.
- The given instance of disk blocks can be represented by a bitmap of 16 bits as: **0000111000000110**.
- Simple to understand and efficient.
- Easy to get contiguous files.



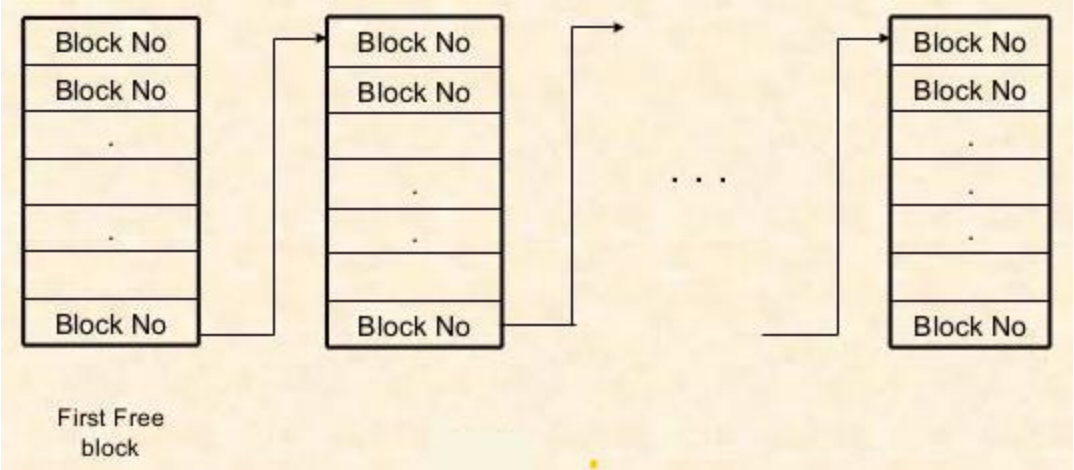
Linked list

- The free disk blocks are linked together i.e. a free block contains a pointer to the next free block.
- The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.
- A drawback of this method is the I/O required for free space list traversal.



Grouping

- This approach stores the address of the free blocks in the first free block.



- First n-1 of these block are address of free blocks.
- Last block contains the pointer of another free block.
- Large number of the free blocks are found quickly.

Thank You

SECONDARY STORAGE STRUCTURE

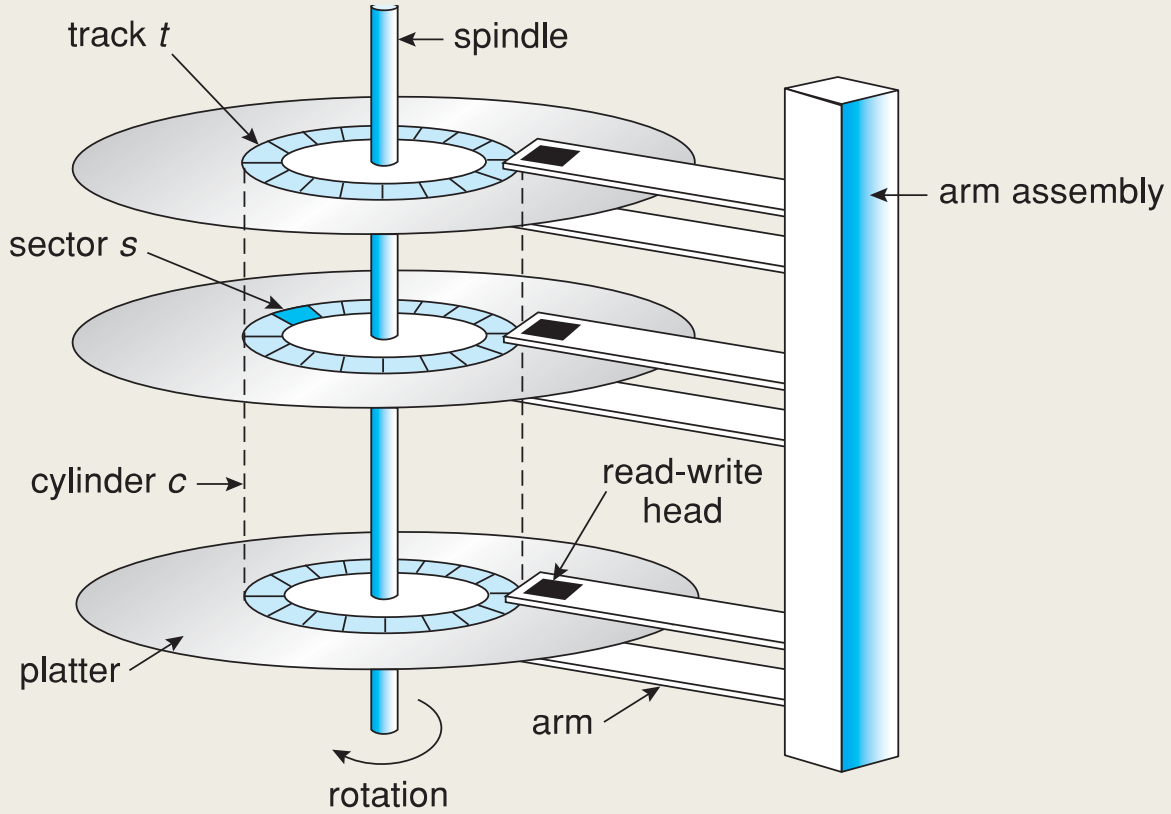
Mass Storage Structure

- Overview of Mass Storage Structure
- Disk Structure
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure

Overview of Mass Storage Structure

- Magnetic disks provide bulk of secondary storage of modern computers
  - *Drives rotate at 60 to 250 times per second*
  - *Transfer rate is rate at which data flow between drive and computer*
  - *Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency)*
  - *Head crash results from disk head making contact with the disk surface -- That's bad*
- Disks can be removable
- Drive attached to computer via I/O bus

Moving-head Disk Mechanism



Disk Access Time

- Seek Time(ST): Time taken by read/write head to reach to desired track
- Rotation time(RT): Time taken for one full rotation
- Rotational latency(RL): Time taken to reach to desired sector(RT/2)
- Transfer time(TT): data to be transfer/transfer rate
- Transfer rate: no of cylinder*capacity of one track*no of rotation each second
- Disk Access time=ST+RL+TT+Controller overhead + Queuing delay

Example:

- What is the average access time for transferring 512 bytes of data with the following specifications-
  - *Average seek time = 5 msec*
  - *Disk rotation = 6000 RPM*
  - *Data rate = 40 KB/sec*
  - *Controller overhead = 0.1 msec*
- Solution :
- Rotation time(RT): for 6000= 60 sec for 1 =60/6000=10 ms
- Rotation latency=RT/2=10/2=5ms
- TT=data to transfer/bit rate=512B/40KB=0.0125sec=12.5 ms
- Disk Access time=ST+RL+TT+Controller overhead

  =5 + 5 + 12.5 +0.1=22.6 ms

DISK SCHEDULING

DISK SCHEDULING

- To use the hardware efficiently is the responsibility of operating system, for disk drives meeting this responsibility require **fast access time and disk bandwidth**
- **Access time** has two major components seek time and rational latency
- **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
- Both access time and bandwidth can be improved by scheduling the disk I/O request in a good order
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

DISK SCHEDULING ALGORITHMS

- Several algorithms exist to schedule the servicing of disk I/O requests
  - FCFS Scheduling
  - SSTF Scheduling
  - SCAN Scheduling
  - C-SCAN Scheduling
  - LOOK Scheduling

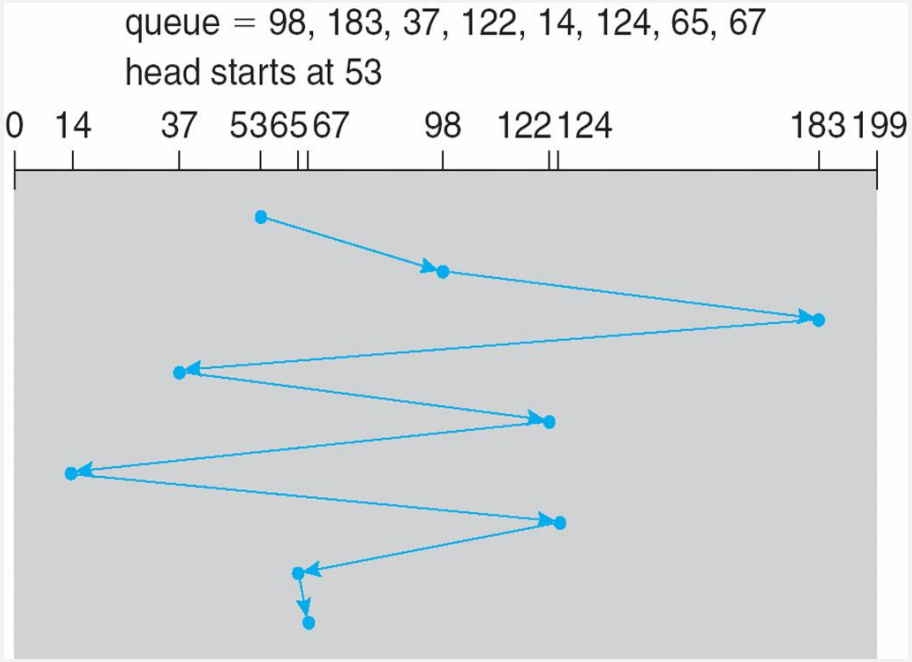
- We illustrate scheduling algorithms with a request queue (0-199)

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

  Head pointer 53

FCFS SCHEDULING

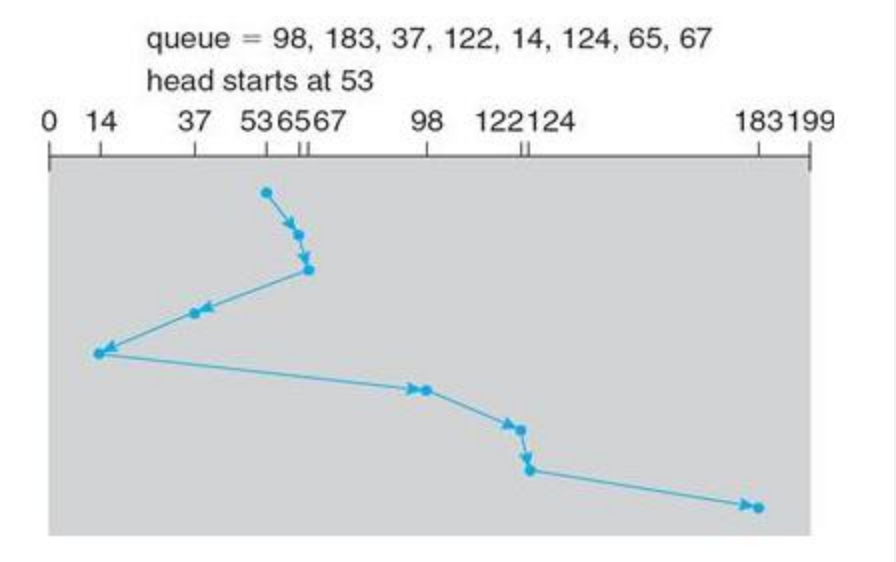
- Simplest form of disk scheduling



- Illustration shows total head movement of 640 cylinders (98-53)+(183-98)+(183-37)+(122-37)+(122-14)+(124-14)+(124-65)+(67-65)
- It does not provide fastest service
- Head movement can be decreased substantially and performance could by improved

SSTF SCHEDULING

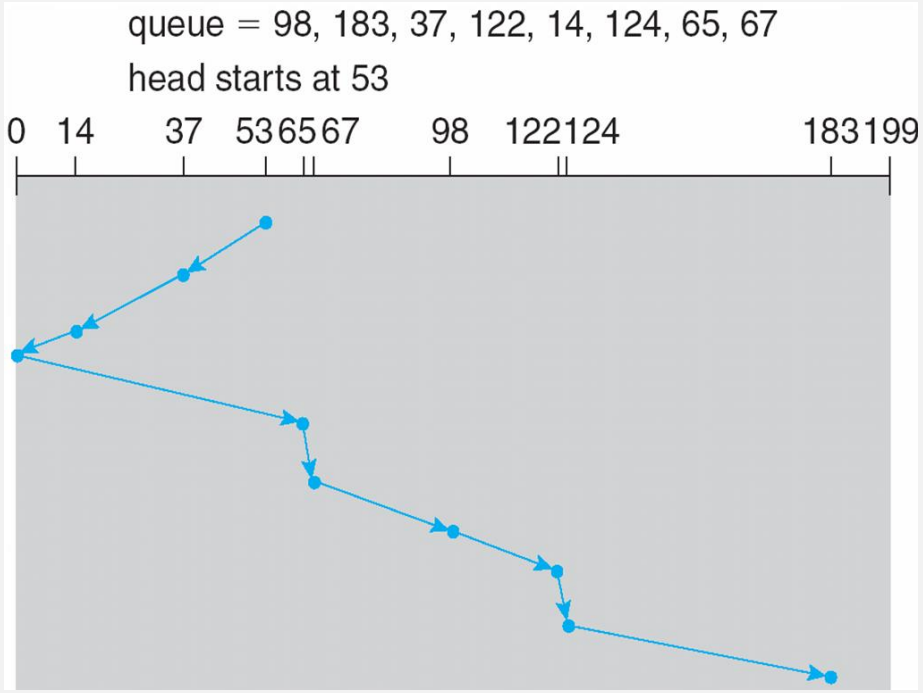
- Selects the request with the **minimum seek time** from the current head position



- Illustration shows total head movement of 236 cylinders
- Give substantial improvement in performance
- SSTF scheduling may cause **starvation** of some requests

SCAN SCHEDULING

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 236 cylinders
- (53-0)+(183-0)=236
- So it provide optimal solution as compared to **SSTF**

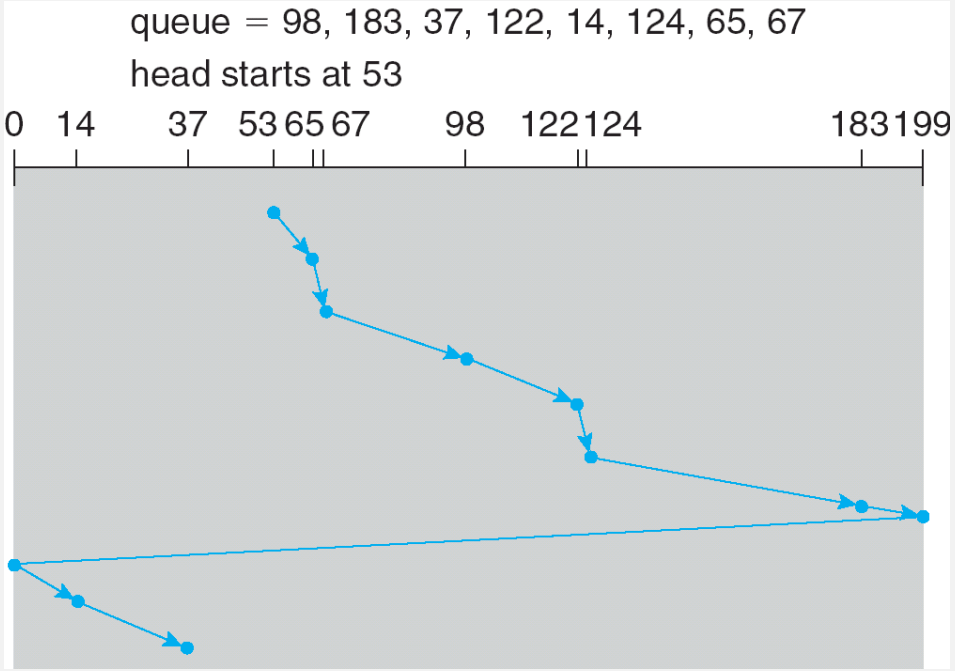


- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

C-SCAN SCHEDULING

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
  - total head movement=(199-53)+(199-0)+(37-0)

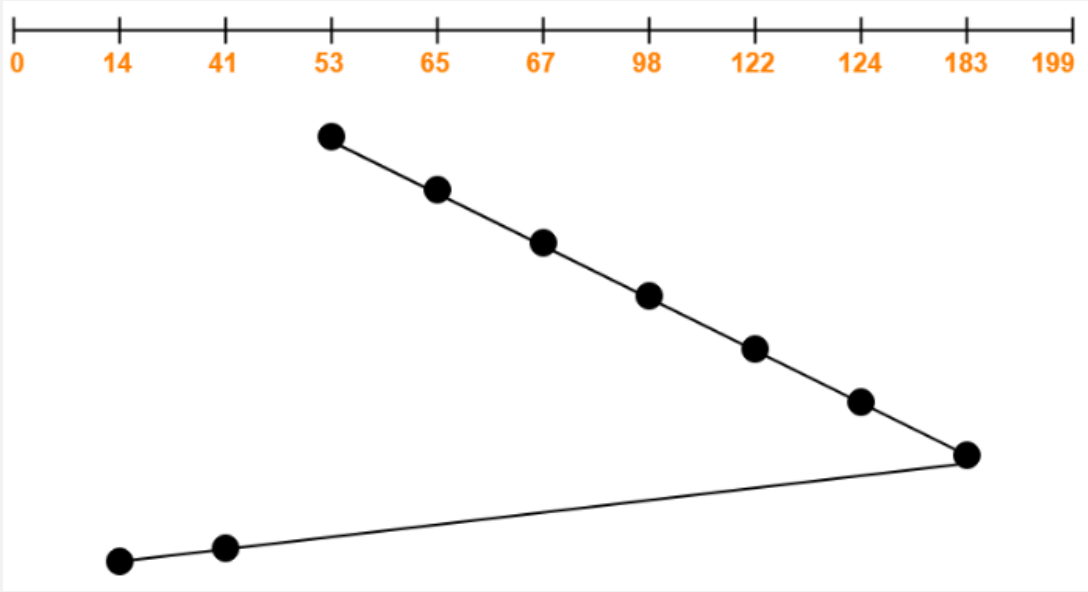
    =146+199+37=382



LOOK SCHEDULING

- Version of SCAN
- Arm only goes as far as the **last request** in each direction, the disk arm starts at last request at one end of the disk, and moves toward the other end, servicing requests until it gets to last request on the other end of the disk, where the head movement is reversed and servicing continues
- Total head movement=(183-53)+(183-41)+(41-14)

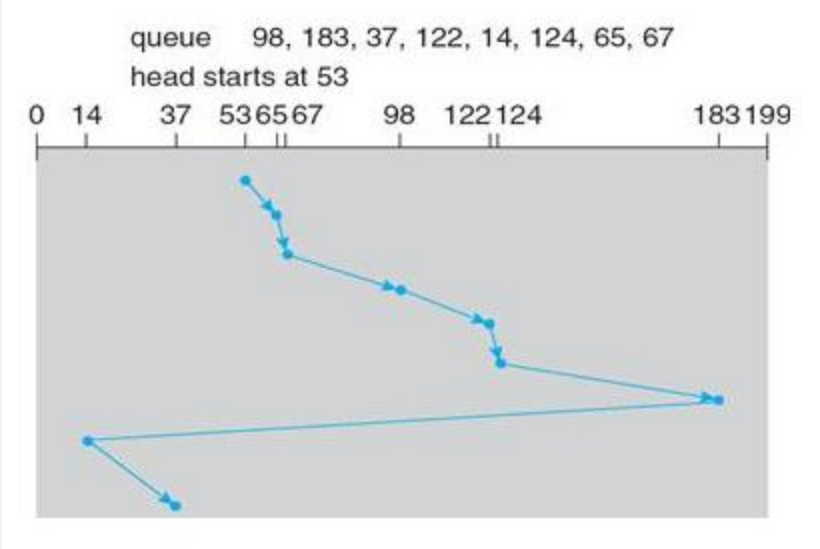
  =130+142+27=299



C-LOOK SCHEDULING

- Version of C-SCAN
- Arm only goes as far as the **last request** in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
- Total head movement=(183-53)+(183-14)+(37-14)

  =130+169+23=322



SELECTING A DISK-SCHEDULING ALGORITHM

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation, but still possible
- Performance depend heavily on number and type of request
- Request for disk service is highly influenced by the file allocation method
  - A program reading contiguously allocated file will generate several request that are close together on the disk, resulting in limited head movement
  - A link or indexed file, may include blocks that are widely scattered on the disk, resulting in greater head movement
- The location of directories and index block is also important
- Because of these complexities the disk –scheduling algorithm should be written as a separate module of OS. So, that it can be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for default algorithm.

QUESTION

- Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks

QUESTION

- Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50 × 10^6 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is

Disk latency = Seek Time + Rotation Time +
Transfer Time + Controller Overhead

Average Rotational Time = (0.5)/(15000 / 60) = 2 miliseconds
[On average half rotation is made]

It is given that the average seek time is twice the average rotational delay
So Avg. Seek Time = 2 * 2 = 4 miliseconds.

Transfer Time = 512 / (50×10^6 bytes/sec)
= 10.24 microseconds

Given that controller time is 10 times the average transfer time
Controller Overhead = 10 * 10.24 microseconds
= 0.1 miliseconds

Disk latency = Seek Time + Rotation Time +
Transfer Time + Controller Overhead
= 4 + 2 + $10.24 * 10^{-3}$ + 0.1 miliseconds
= 6.1 miliseconds

RAID

- **RAID** – **redundant array of independent disks**
  - multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**
  - Suppose that the **mean time to failure** of a single disk is 100,000 hours. Then the mean time to failure of some disk in an array of 100 disks will be 100,000/100 = 1,000 hours, or 41.66 days
- The solution to the problem of reliability is to introduce **redundancy**
- Improvement in Performance via Parallelism
- **Redundancy** can be introduced in two ways
  - Mirroring
  - Data Striping

MIRRORING

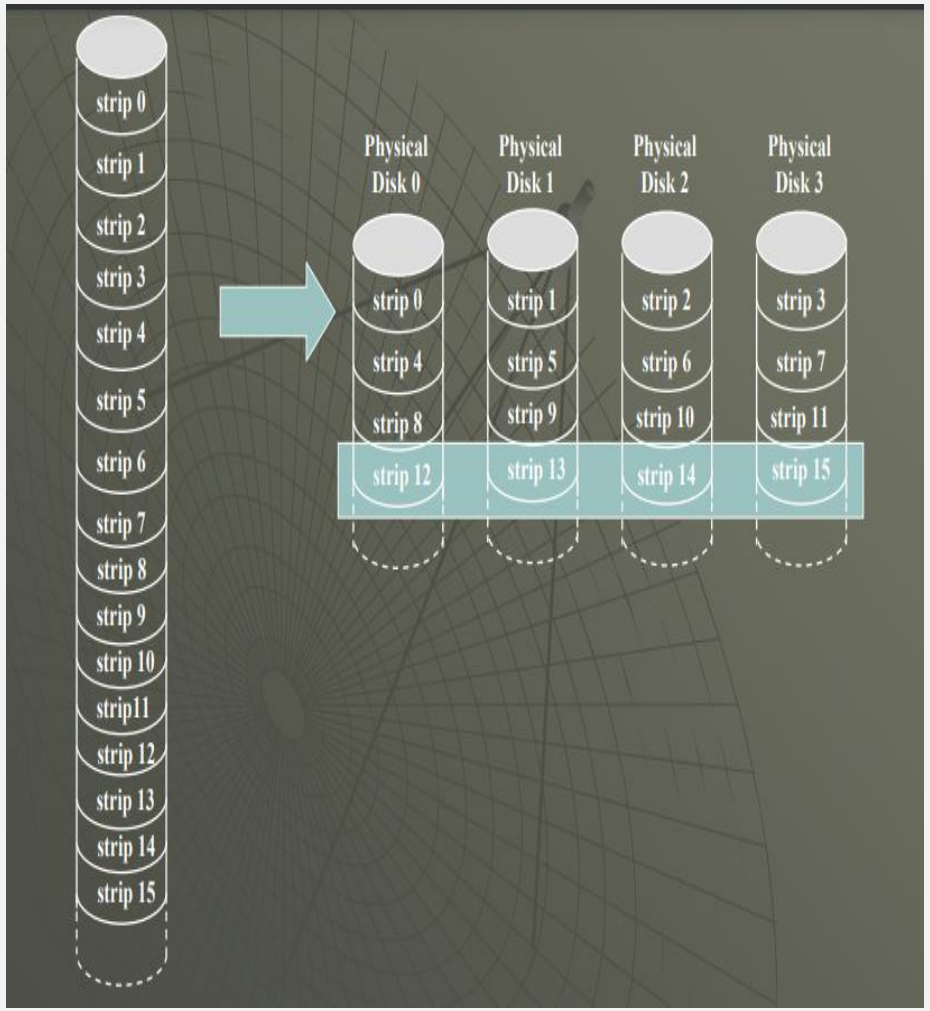
- Duplicate every disk
- Logical disk consists of two physical disks.
- Every write is carried out on both disks.
- If one of the disk fails, data read from the other
- Data permanently lost only if the second disk fails before the first failed disk is replaced.
  - Suppose mean time to repair is 10 hrs , the mean time to data loss of a mirrored disk system is 100,000 ^ 2 / (2 * 10) hrs ~ 57,000 years !
- Main disadvantage :
  - Most expensive approach .

DATA STRIPING

- It is the process of dividing a body of data into blocks and spreading the data blocks across multiple storage devices in a redundant array of independent disks (RAID) .
- Because striping spreads data across more physical drives, multiple disks can access the contents of a file, enabling writes and reads to be completed more quickly.
- Parallelism in a disk system, as achieved through striping, has two main goals:

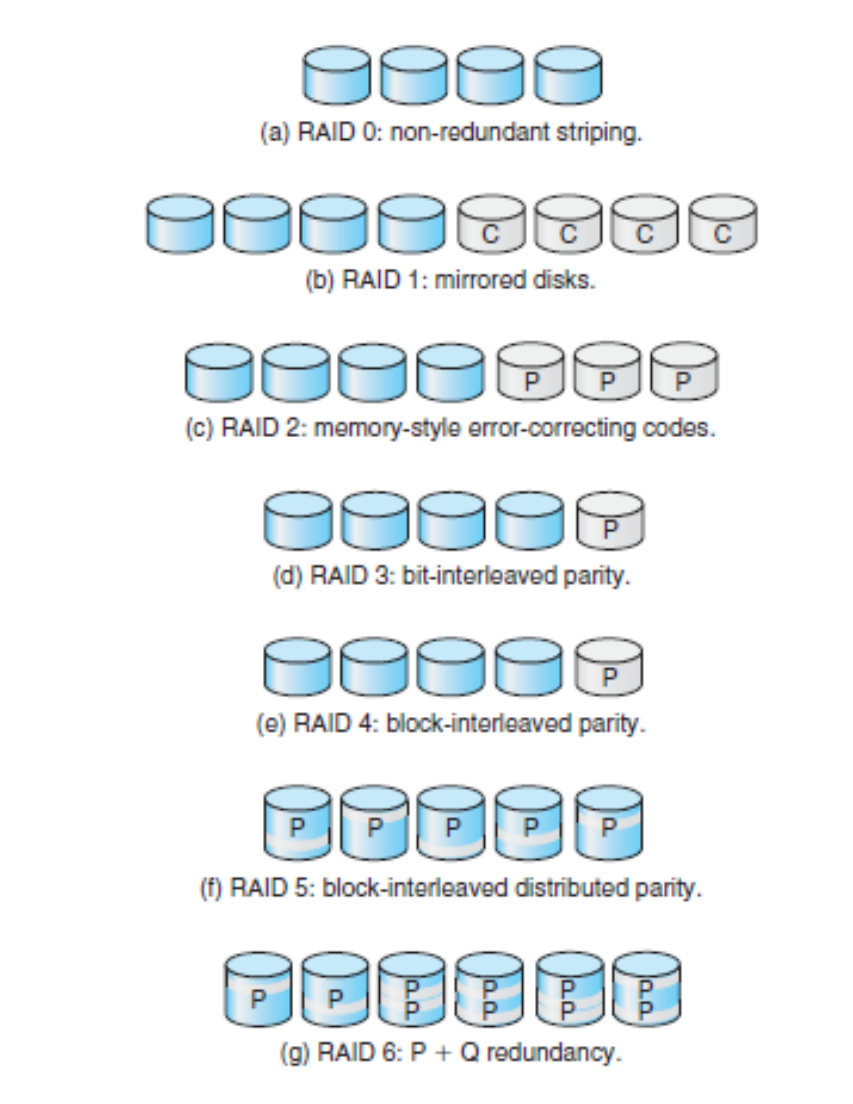
  1. Increase the throughput of multiple small accesses (that is, page accesses) by load balancing.

  2. Reduce the response time of large accesses.



RAID LEVELS

- Mirroring provides high reliability, but it is expensive. Striping provides high data-transfer rates, but it does not improve reliability.
- Numerous schemes to provide redundancy at lower cost by using disk striping combined with "parity" bits (which we describe shortly) have been proposed. These schemes have different cost–performance trade-offs and are classified according to levels called **RAID levels**.



RAID LEVEL 0

- **RAID level 0**. RAID level 0 refers to disk arrays with striping at the level of blocks but without any redundancy (such as mirroring or parity bits),

- Blocks are "striped" across disks.

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- **Reliability:** 0
  There is no duplication of data. Hence, a block once lost cannot be recovered.
- **Capacity:** N*B
  The entire space is being used to store data. Since there is no duplication, N disks each having B blocks are fully utilized.

RAID LEVEL 1

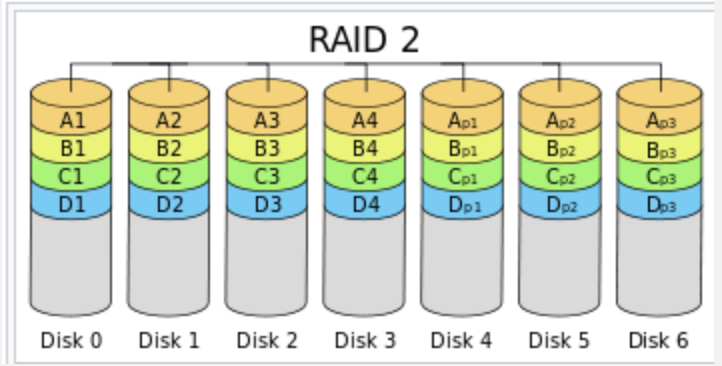
- RAID level 1 refers to disk mirroring.

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

- **Reliability:** 1 to N/2
  1 disk failure can be handled for certain, because blocks of that disk would have duplicates on some other disk. If we are lucky enough and disks 0 and 2 fail, then again this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, N/2 disk failures can be handled.
- **Capacity:** N*B/2
  Only half the space is being used to store data. The other half is just a mirror to the already stored data.

RAID LEVEL 2

- It stripes data at the bit (rather than block) level, and uses a Hamming code for error correction.
- The disks are synchronized by the controller to spin at the same angular orientation
- However, depending with a high rate Hamming code, many spindles would operate in parallel to simultaneously transfer data so that "very high data transfer rates" are possible

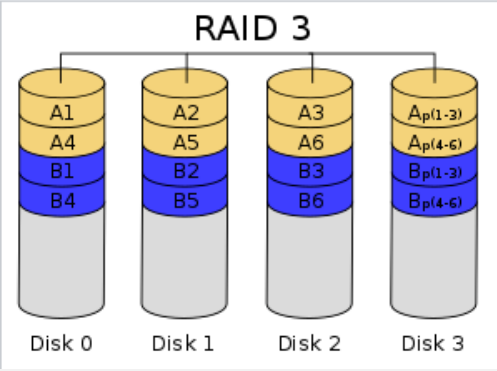


- If one of the disks fails, the remaining bits of the byte and the associated error-correction bits can be read from other disks and used to reconstruct the damaged data

RAID LEVEL 3

- RAID level 3, or bit-interleaved parity organization, improveson level 2
- A single parity bit is be used for error correction as well as for detection
- If one of the sectors is damaged, we know exactly which sector it is, and we can figure out whether any bit in the sector is a 1 or a 0 by computing the parity of the corresponding bits from sectors in the other disks. If the parity of the remaining bits is equal to the stored parity, the missing bit is 0; otherwise, it is 1.

0	1	0	0
1	1	0	1



- RAID level 3 has two advantages

  1. the storage overhead is reduced because only one parity disk is needed for several regular disks

  2. the transfer rate for reading or writing a single block is *N* times as fast as with RAID level 1
- All parity based RAID levels—is the expense of computing and writing the parity. This overhead results in significantly slower writes than with non-parity RAID arrays.

RAID LEVEL 4

- RAID level 4, or block-interleaved parity organization, uses block-level striping, as in RAID 0, and in addition keeps a parity block on a separate disk for corresponding blocks from *N* other disks.
- Parity is calculated using a simple XOR function. If the data bits are 0,0,0,1 the parity bit is XOR(0,0,0,1) = 1. If the data bits are 0,1,1,0 the parity bit is XOR(0,1,1,0) = 0. A simple approach is that even number of ones results in parity 0, and an odd number of ones results in parity 1.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

- **Reliability:** 1
  RAID-4 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data.
- **Capacity:** (N-1)*B
  One disk in the system is reserved for storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

RAID LEVEL 5

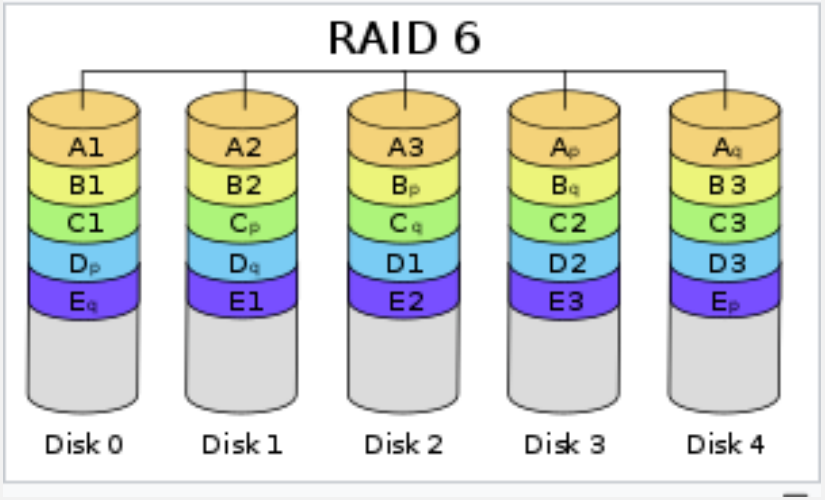
- This is a slight modification of the RAID-4 system where the only difference is that the parity rotates among the drives.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

- **Reliability:** 1
  RAID-4 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data.
- **Capacity:** (N-1)*B
  One disk in the system is reserved for storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

RAID LEVEL 6

- RAID level 6, also called the **P + Q redundancy scheme**, is much like RAID level 5 but stores extra redundant information to guard against multiple disk failures
- 2 bits of redundant data are stored for every 4 bits of data—compared with 1 parity bit in level 5—and the system can tolerate two disk failures.



- Advantage:
  - Multiple disk failure can be handled
- Disadvantage:
  - Parity overhead

RAID LEVELS 10

- For very high performance and reliability combination of RAID level 0 and Level 1 can be used.
- Highest performance
- Expensive