Unit 5- Storage Management

J.Premalatha
Professor/IT
Kongu Engineering College
Perundurai

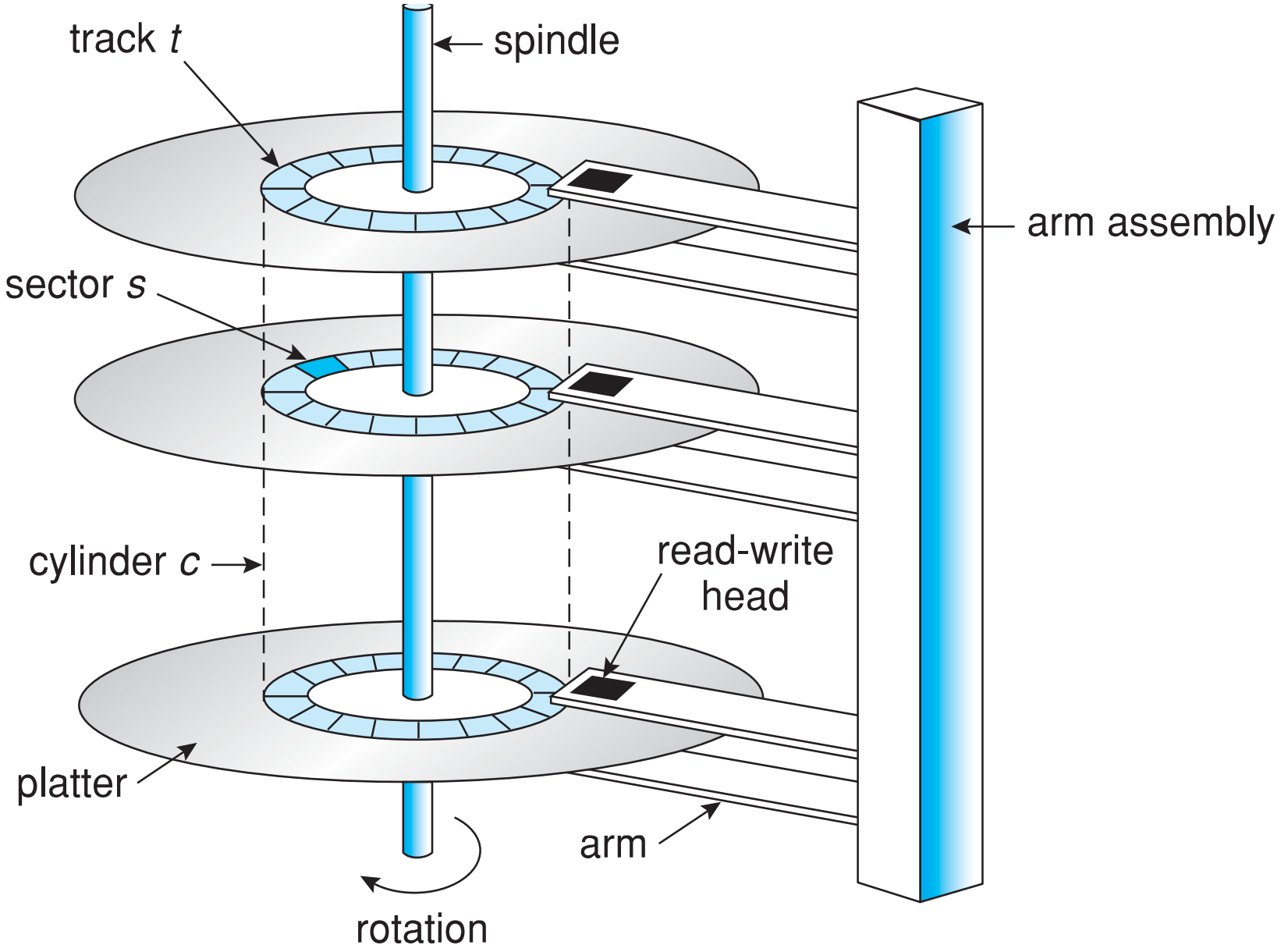
File system can be viewed logically as consisting of three parts.

File system Interface	**The user and programmer interface to the file system**
File system Implementation	**The internal data structures and algorithms used by the OS to implement the file system interface**
Mass storage structure	**The secondary and tertiary storage structure**

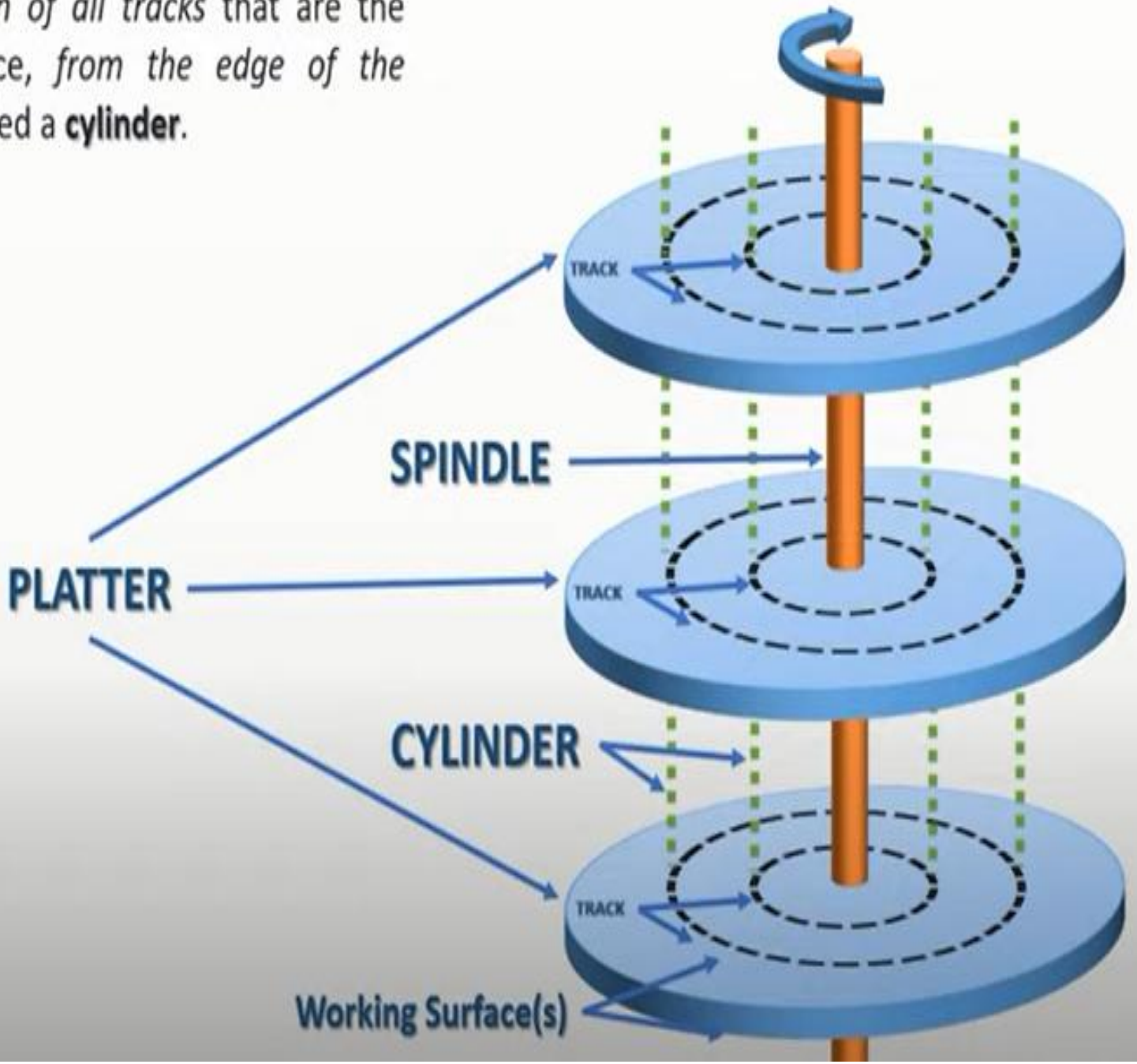
Overview of Mass Storage Structure

- Bulk of secondary storage for modern computers is **hard disk drives (HDDs)** and **nonvolatile memory (NVM)** devices
- **HDDs** spin **platters of magnetically-coated material under moving read-write heads**
  - Drives rotate at 60 to 250 times per second
  - Each **disk platter has a flat circular shape. A read –write head "flies" just above each surface of every platter.**
  - The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are subdivided into sectors.
  - The **set of tracks at a given arm position make up a cylinder**. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. Cylinders are vertically formed by tracks. **In other words, track 12 on platter 0 plus track 12 on platter 1 etc. is cylinder 12.**

Moving-head Disk Mechanism



The *collection of all tracks* that are the same distance, *from the edge of the platter*, is called a **cylinder**.



- Each sector has a fixed size and is the smallest unit of transfer. The sector size was commonly 512 bytes to 4KB.
- Rotation speed relates to transfer rates.
- The **transfer rate is the rate at which data flow between the drive(hard disk) and the computer(Main Memory).**
- Another performance aspect, the **positioning time or random-access time or access time**, consists of two parts:

1. Seek time: The time necessary to **move the disk arm to the desired cylinder**

2. Rotational latency: The time necessary for **the desired sector to rotate to the disk head**

  - **Head crash** results from disk head making contact with the disk surface -- That's bad

- Disks can be removable

Hard Disk Drives

- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from 30GB to 3TB per drive
- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency

Hard Disk Performance

- **Access Latency = Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

Nonvolatile Memory Devices(NVM)

- It is also called as **solid-state disks (SSDs)**
- Other forms include **USB drives** (thumb drive, flash drive), DRAM disk replacements, surface-mounted on motherboards, and main storage in devices like smartphones
- Can be more reliable than HDDs
- More expensive per MB
- May be have shorter life span – need careful management
- Less capacity
- But much faster - **No moving parts, so no seek time or rotational latency**

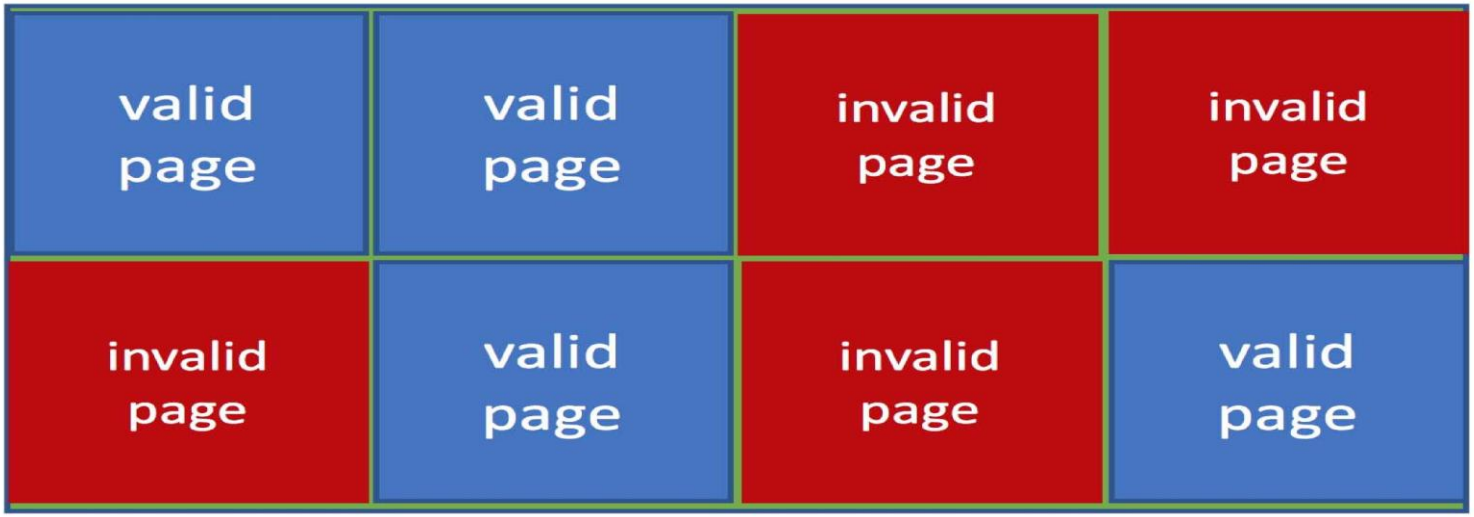
Nonvolatile Memory Devices

- Have characteristics that present challenges
- Read and written in “page” increments **but can’t overwrite in place**
  - **Must first be erased,** and erases happen in larger ”block” increments
  - Can only be erased a limited number of times before worn out – ~ 100,000
  - Life span measured in **drive writes per day (DWPD)**
    - A 1TB NAND drive with rating of 5DWPD is expected to have 5TB per day written within warrantee period without failing

NAND Flash Controller Algorithms

- NAND semiconductors cannot be overwritten once written, there are usually pages containing invalid data. But it allows erase .
- If no erase has occurred in the meantime, the page first written has the old data, which are now invalid, and the second page has the current, good version of the block. A NAND block containing valid and invalid pages
- To track which logical blocks are valid, controller maintains **flash translation layer (FTL) table**
- Also implements **Garbage Collection(GC)** to free invalid page space. Valid data could be copied to other locations, freeing up blocks that could be erased and could then receive the writes.
- Allocates **over provisioning** to provide working space for Garbage Collection. The device sets aside a number of pages (frequently 20 percent of the total) as an area always available to write to.

NAND block with valid and invalid pages



- Blocks that are totally invalid by garbage collection, or write operations invalidating older versions of the data, are erased and placed in the over-provisioning space if the device is full or returned to the free pool.
- Each cell has lifespan, so **wear leveling** needed to write equally to all cells. If some blocks are erased repeatedly, while others are not, the frequently erased blocks will wear out faster than the others, and the entire device will have a shorter lifespan than it would if all the blocks wore out concurrently.

Volatile Memory

- DRAM frequently used as mass-storage device
  - Not technically secondary storage because volatile, but can have file systems, be used like very fast secondary storage
- **RAM drives** (with many names, including RAM disks) present as raw block devices, commonly file system formatted
- Computers have buffering, caching via RAM
  - Caches / buffers allocated / managed by programmer, operating system, hardware
  - RAM drives under user control
  - Found in all major operating systems
- Used as high speed temporary storage
  - Programs could share bulk date, quickly, by reading/writing to RAM drive

Magnetic Tape

Magnetic tape was used as an early secondary-storage medium. Although it is nonvolatile and can hold large quantities of data, its access time is slow compared with that of main memory and drives. In addition, random access to magnetic tape is about a thousand times slower than random access to HDDs and about a hundred thousand times slower than random access to SSDs so tapes are not very useful for secondary storage. Tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.

A tape is kept in a spool and is wound or rewound past a read–write head. Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can read and write data at speeds comparable to HDDs. Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes. Some tapes have built-in compression that can more than double the effective storage. Tapes and their drivers are usually categorized by width, including 4, 8, and 19 millimeters and 1/4 and 1/2 inch. Some are named according to technology, such as LTO-6 (Figure 11.5) and SDLT.

Figure 11.5 An LTO-6 Tape drive with tape cartridge inserted.

Disk Attachment

- Host-attached storage accessed through I/O ports talking to **I/O busses**
- Several busses available, including **advanced technology attachment (ATA)**, **serial ATA (SATA)**, **eSATA**, **serial attached SCSI (SAS)**, **universal serial bus (USB)**, and **fibre channel (FC)**.
- Most common is SATA
- Because NVM much faster than HDD, new fast interface for NVM called **NVM express (NVMe)**, connecting directly to PCI bus
- Data transfers on a bus carried out by special electronic processors called **controllers** (or **host-bus adapters**, **HBAs**)
  - Host controller on the computer end of the bus, device controller on device end
  - Computer places command on host controller, using memory-mapped I/O ports
    - Host controller sends messages to device controller
    - Data transferred via DMA between device and computer DRAM

HDD Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- 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**
- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes

Disk Scheduling (Cont.)

- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists
- In the past, operating system responsible for queue management, disk drive head scheduling
  - Now, built into the storage devices, controllers

Disk Scheduling algorithms

- Several algorithms exist to schedule the servicing of disk I/O requests
- FCFS,SSTF, SCAN ,C-SCAN, LOOK and C-LOOK scheduling algorithms
- Illustrate scheduling algorithms with a request queue (0-199)

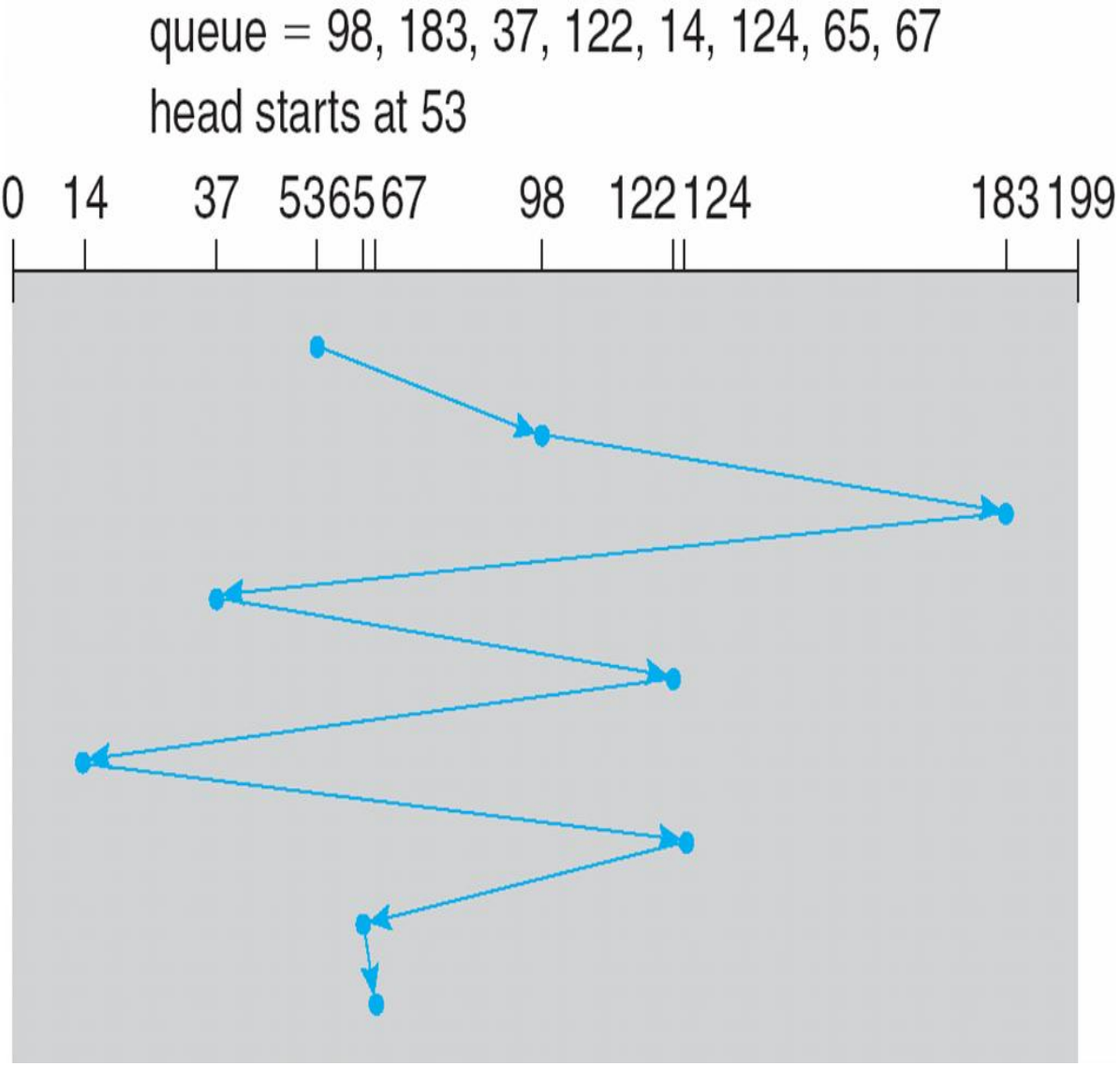
  98, 183, 37, 122, 14, 124, 65, 67 and Head pointer 53

FCFS scheduling : The requests are addressed in the order they arrive in the disk queue.

Advantage : Algorithm is intrinsically[by naturally] fair, No starvation

Dis Advantage : Not the best performance

FCFS - Illustration shows total head movement of 640 cylinders



Total head movement =
(98-53)+ (183-98)+
(183-37)+(122-37)+
(122-14) + (124-14)+
(124-65)+(67-65)

= 45+85+146+85+108+
110+59+2

=640

SSTF – Shortest Seek Time First

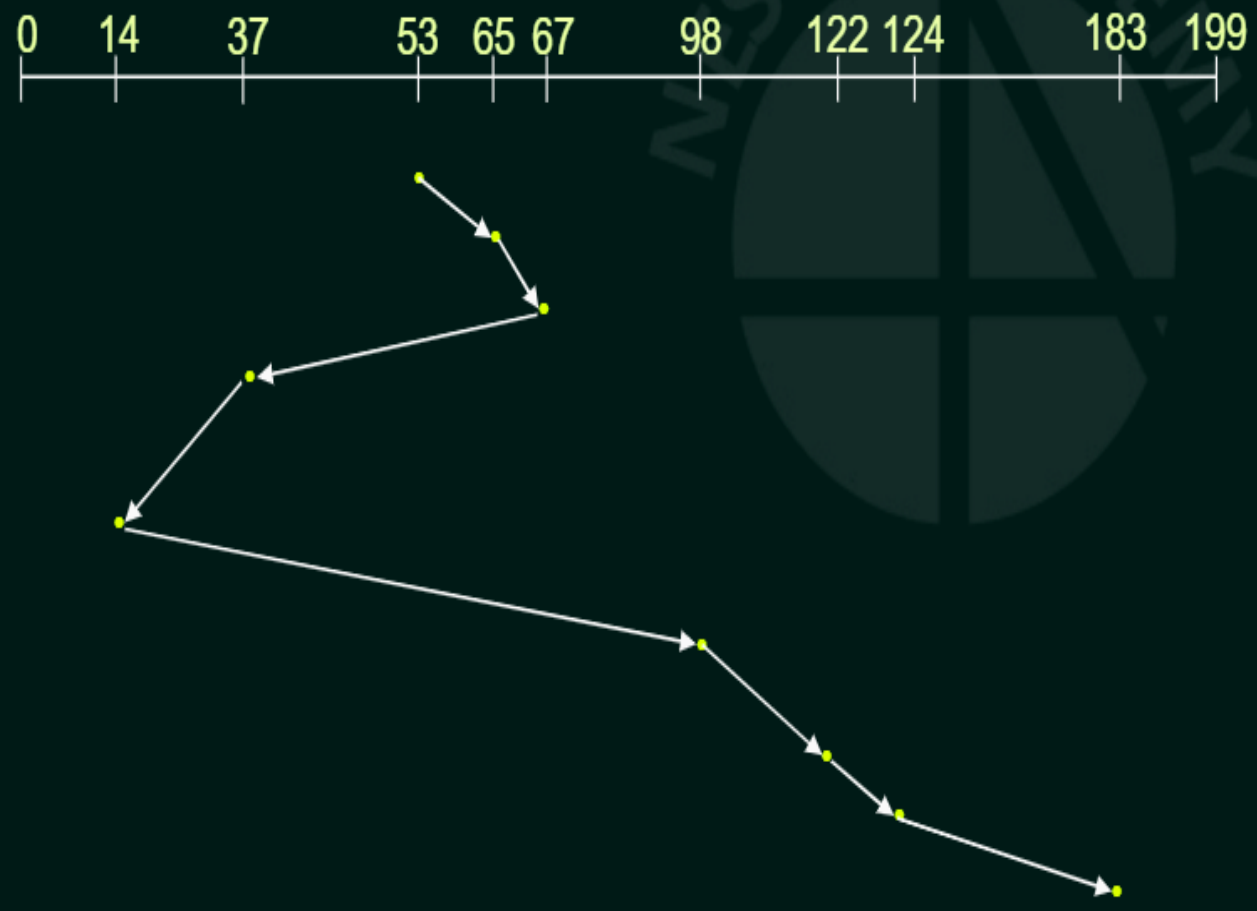
To service all the requests close to the current head position before moving the head for away to service other requests.

Advantage : Better performance than FCFS disk scheduling algorithm
Dis Advantage : Though it is better than FCFS, it is not optimal

SSTF disk scheduling

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

Head starts at: 53



Total head movement

=

(65-53) + (67-65) +
(67-37) + (37-14) +
(98-14) + (122-98) +
(124-122) + (183-124)

= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236

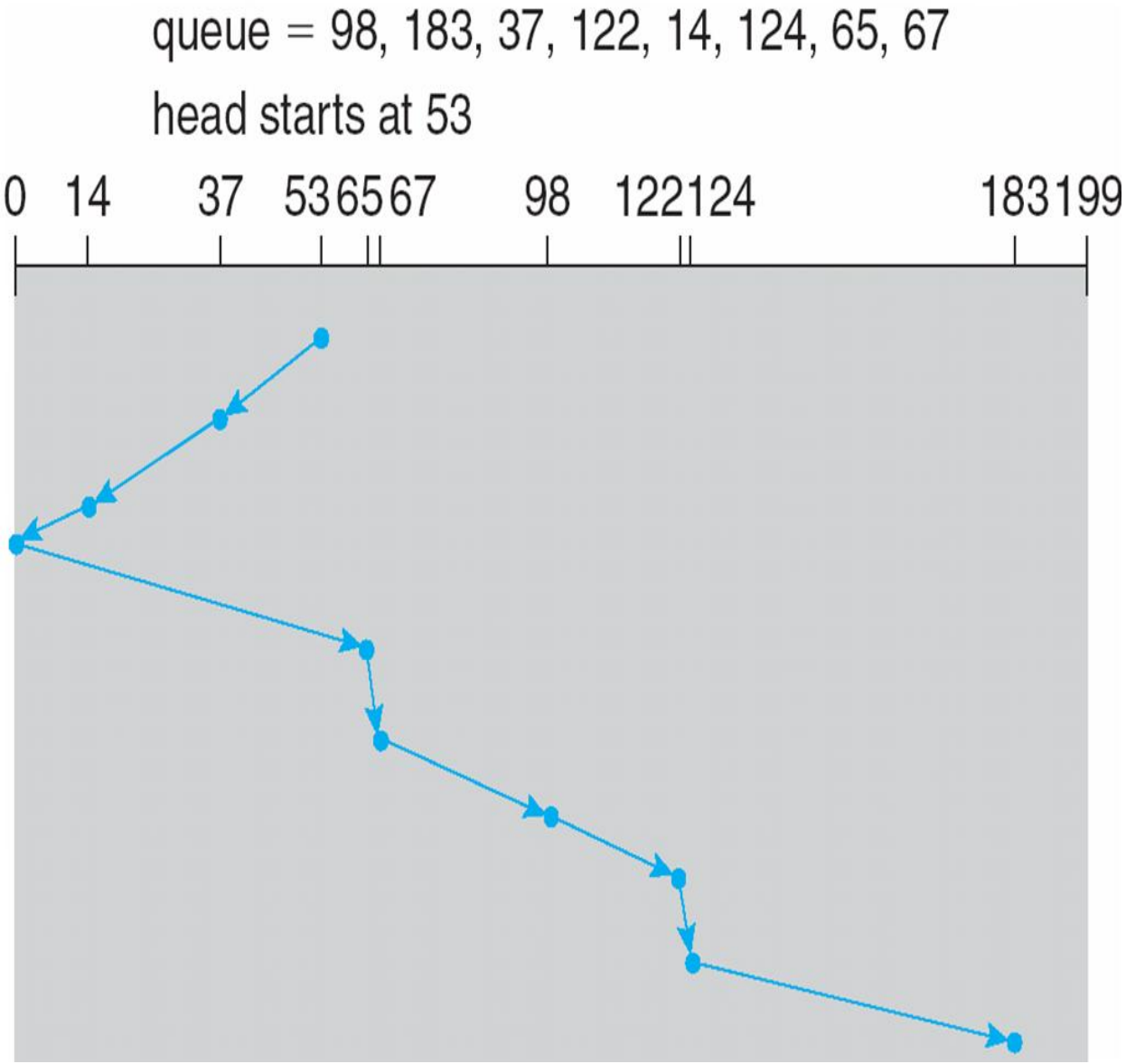
SCAN

- 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.
- The head continuously scans back and forth across the disk
- **SCAN algorithm** sometimes called as **elevator algorithm**

Advantages: Better than FCFS, No starvation

Dis Advantage : If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction and comes back

SCAN –Assume the direction of head movement is towards ‘0’



Total head movement
(53-37) +(37-14)+ (14-0) + (65-0)+ (67-65)+ (98-67)+ (122-98) (124-122)+ (183-124)

=16+23+14+65+2+31+24+2+59

=236

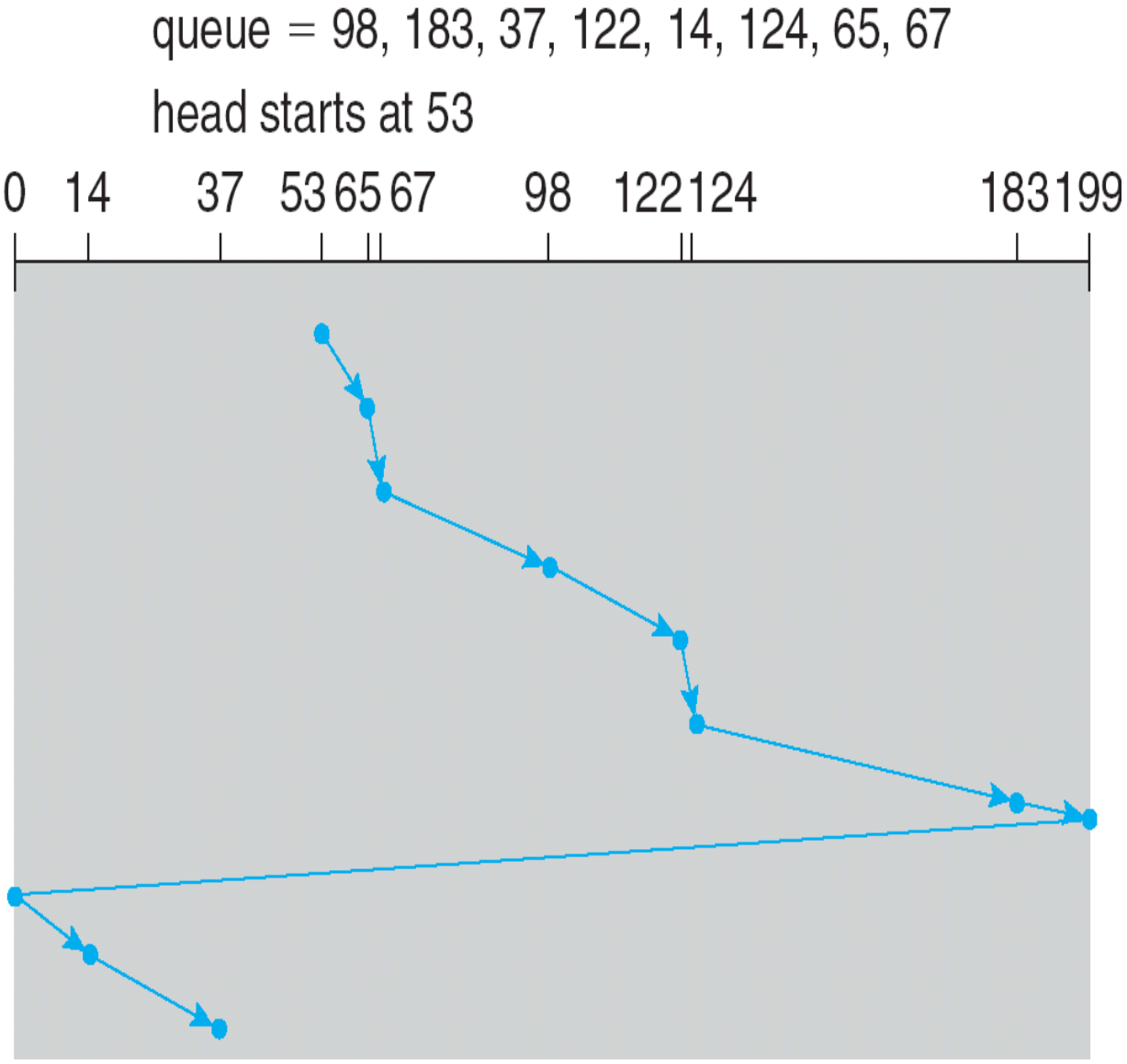
C-SCAN

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

Advantage : Uniform wait time

Dis Advantage : Total head movement is more compared to SCAN algorithm

C-SCAN : direction of head movement is towards 199



Total head movement
(65-53)+(67-65)+
(98-67) + (122-98)+
(124-122)+(183-124)+
(199-183)+(199-0)+
(14-0)+(37-14)

=12+2+31+24+2+59+1
6+199+14+23

= 382

LOOK

- Both the SCAN and C-SCAN move the disk arm across the full width of the disk.
- But in LOOK algorithm, 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.

Advantages: Seek time is lesser than all algorithms.

LOOK disk scheduling

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

Head starts at: 53 (and direction of head movement is towards 0)

Total head movement

=

(53-37) + (37-14) +
(65-14) + (67-65) +
(98-67) + (122-98) +
(124-122) + (183-124)

= 16 + 23 + 51 + 2 + 31 + 24 +
2 + 59

= 208

C-LOOK

- Like LOOK, C-LOOK algorithm moves the head in one direction servicing requests along the way.
- When the head reaches the last request, it immediately returns to the farthest request on the other end without servicing any requests on the return trip.
- Then from that position, it services the remaining request in the same direction like before.

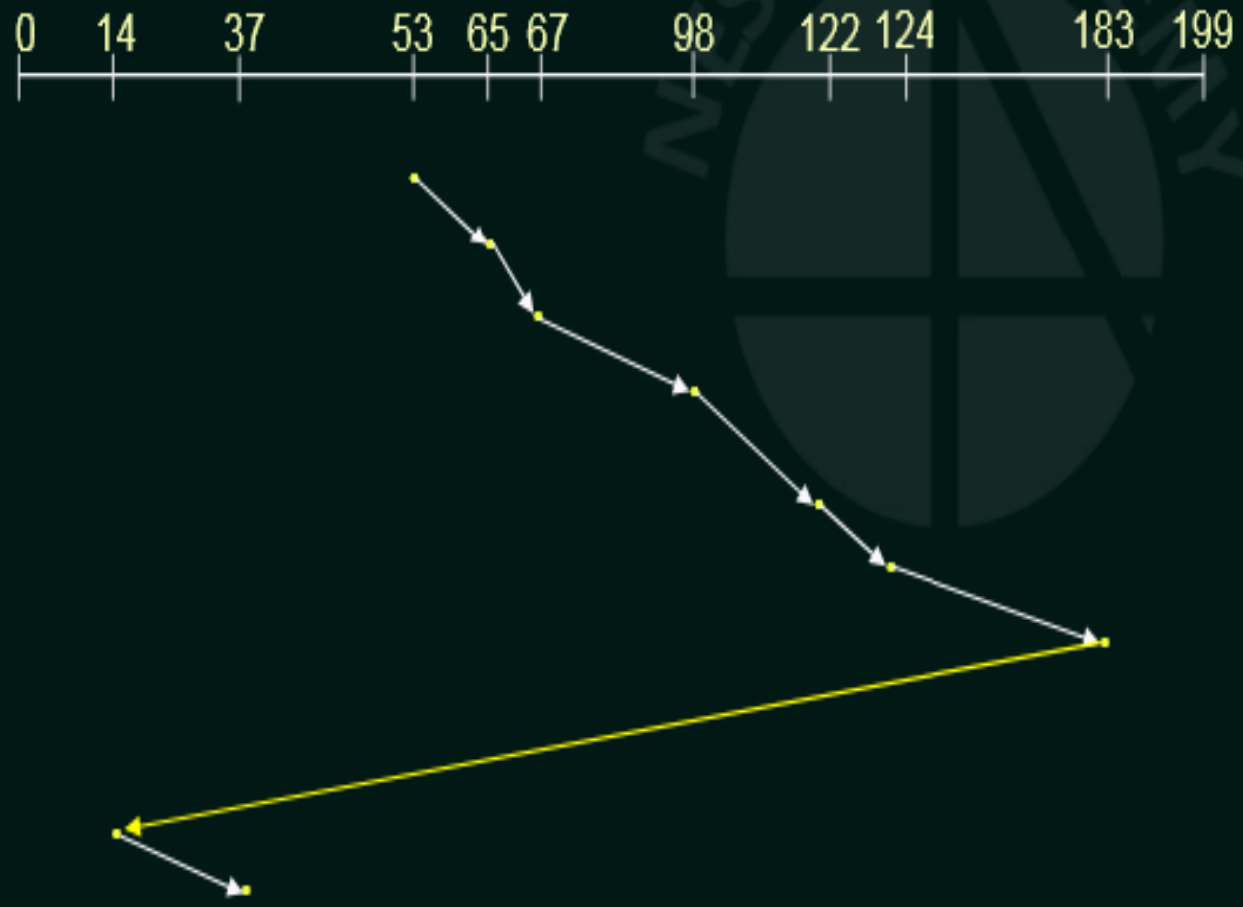
Advantage: More uniform wait time

Dis Advantage : Total head movement is more compared to LOOK algorithm.

C-LOOK disk scheduling

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

Head starts at: 53 (and direction of head movement is towards 199)



Total head movement

=

(65-53) + (67-65) +
(98-67) + (122-98) +
(124-122) + (183-124) +
(183-14) + (37-14)

= 12 + 2 + 31 + 24 + 2 + 59 +
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
- To avoid starvation Linux implements **deadline** scheduler
  - Maintains separate read and write queues, gives read priority
    - Because processes more likely to block on read than write
  - Implements four queues: 2 x read and 2 x write
    - 1 read and 1 write queue sorted in LBA order, essentially implementing C-SCAN
    - 1 read and 1 write queue sorted in FCFS order
    - All I/O requests sent in batch sorted in that queue's order
    - After each batch, checks if any requests in FCFS older than configured age (default 500ms)
      - If so, LBA queue containing that request is selected for next batch of I/O

EX1: Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, For each of the following disk-scheduling algorithms a. FCFS b. SSTF c. SCAN d. LOOK e. C-SCAN f. C-LOOK

The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081.

The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774. The total seek distance is 1745.

The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 130, 86. The total seek distance is 9769.

The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek distance is 3319.

The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 0, 86, 130. The total seek distance is 9985.

The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130. The total seek distance is 3363.

Ex2. Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 2150, and the previous request was at cylinder 1805. The queue of pending requests, in FIFO order, is 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms:

a. FCFS b. SSTF c. SCAN d. LOOK e. C-SCAN f. C-LOOK

a. FCFS service order

2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4956, 3681

Sum of movements per step, starting from 2150:

81 + 857 + 1084 + 504 + 2256 + 1074 + 1262 + 1167 + 3433 + 1275= 13011

b. SSTF service order

2069, 2296, 2800, 3681, 4956, 1618, 1523, 1212, 544, 356

Sum of movements per step, starting from 2150:

81 + 227 + 504 + 881 + 1275 + 3338 + 95 + 311 + 668 + 188= 7586

c. SCAN service order

2296, 2800, 3681, 4956, (4999), 2069, 1618, 1523, 1212, 544, 356

Sum of movements per step, starting from 2150:

146 + 504 + 881 + 1275 + 43 + 2930 + 451 + 95 + 311 + 668 + 188= 7492

d. LOOK service order

2296, 2800, 3681, 4956, 2069, 1618, 1523, 1212, 544, 356

Sum of movements per step, starting from 2150:

146 + 504 + 881 + 1275 + 2887 + 451 + 95 + 311 + 668 + 188 = 7424

e. C-SCAN service order

2296, 2800, 3681, 4956, (4999), (0), 356, 544, 1212, 1523, 1618, 2069

Sum of movements per step, starting from 2150:

146 + 504 + 881 + 1275 + 43 + 4999 + 356 + 188 + 668 + 311 + 95 + 451= 9917

f. C-LOOK service order

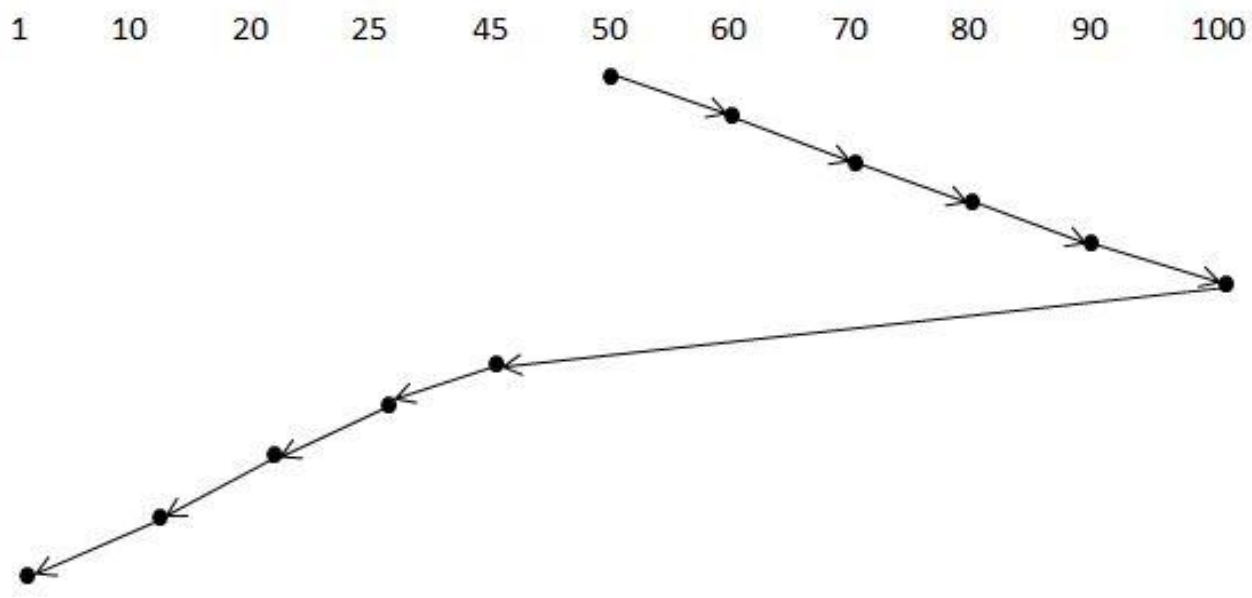
2296, 2800, 3681, 4956, 356, 544, 1212, 1523, 1618, 2069

Sum of movements per step, starting from 2150:

146 + 504 + 881 + 1275 + 4600 + 188 + 668 + 311 + 95 + 451= 9137

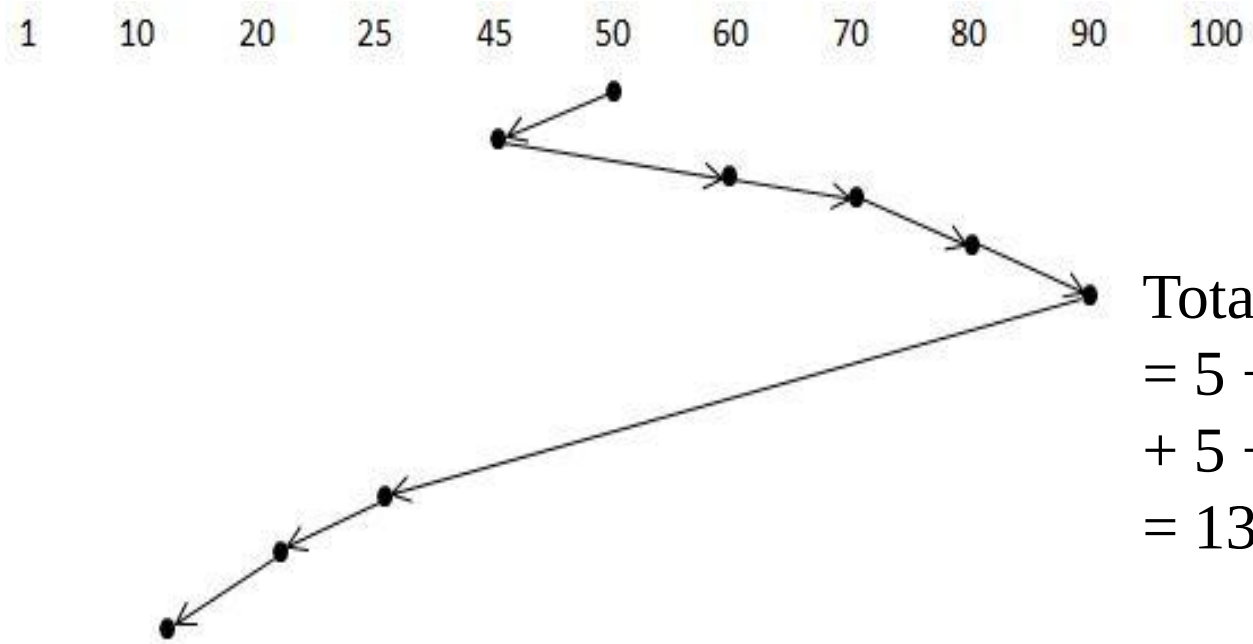
G1: Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is ____ tracks. (GATE 2015)

SCAN:



Total head movements,
= 10 + 10 + 10 + 10 + 10 + 55 + 20 + 5 + 10
= 140

SSTF:



Total head movements
= 5 + 15 + 10 + 10 + 65
+ 5 + 10 + 10
= 130

∴ Additional distance that will be traversed by R/W head is
= 140 - 130
= 10

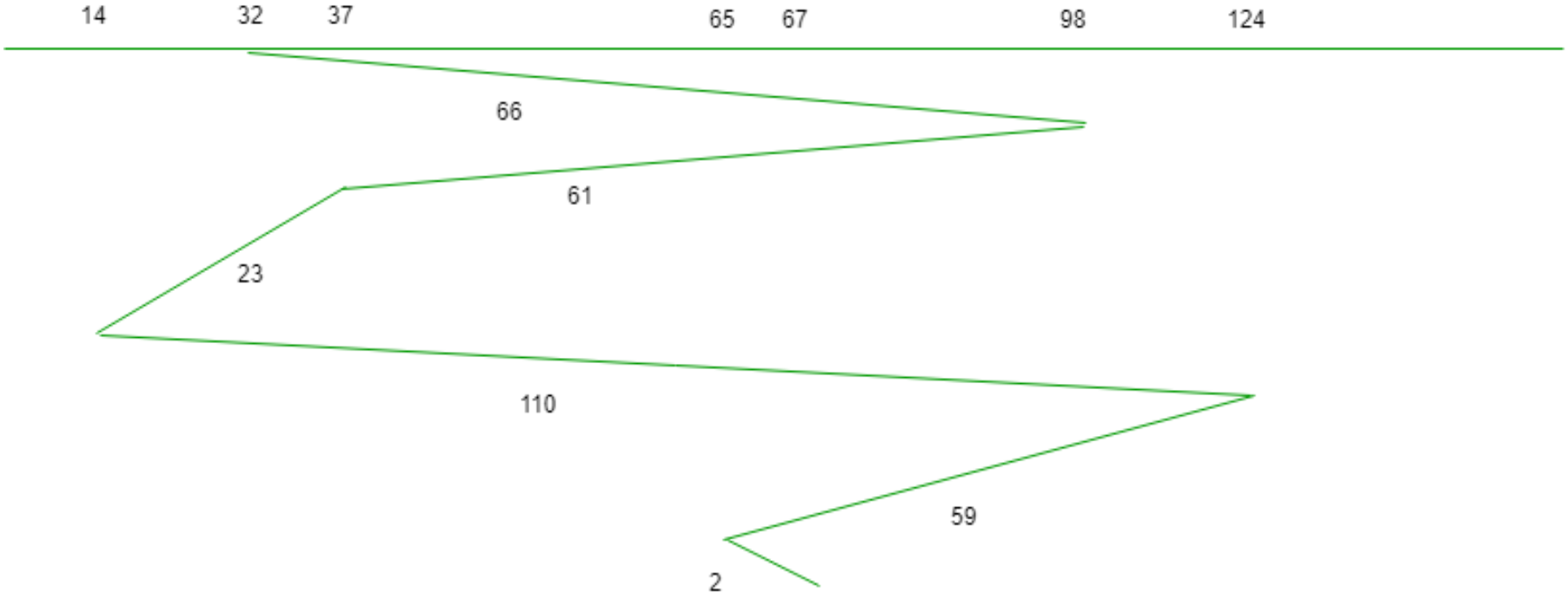
G2. A disk has 200 tracks (numbered 0 through 199). At a given time, it was servicing the request of reading data from track 120, and at the previous request, service was for track 90. The pending requests (in order of their arrival) are for track numbers. 30 70 115 130 110 80 20 25. How many times will the head change its direction for the disk scheduling policies SSTF(Shortest Seek Time First) and FCFS (First Come First Serve). (GATE 2004)

According to Shortest Seek Time First:
90->120-> 115->110-> 130-> 80-> 70-> 30-> 25-> 20
Change of direction(Total 3); 120->115; 110->130; 130->80

According to First Come First Serve:
90->120-> 30-> 70-> 115->130-> 110-> 80-> 20-> 25
Change of direction(Total 4); 120->30; 30->70; 130->110;20->25

G3. If the Disk head is located initially at track 32, find the number of disk moves required with FCFS scheduling criteria if the disk queue of I/O blocks requests are: 98, 37, 14, 124, 65, 67 (UGC NET 2016)

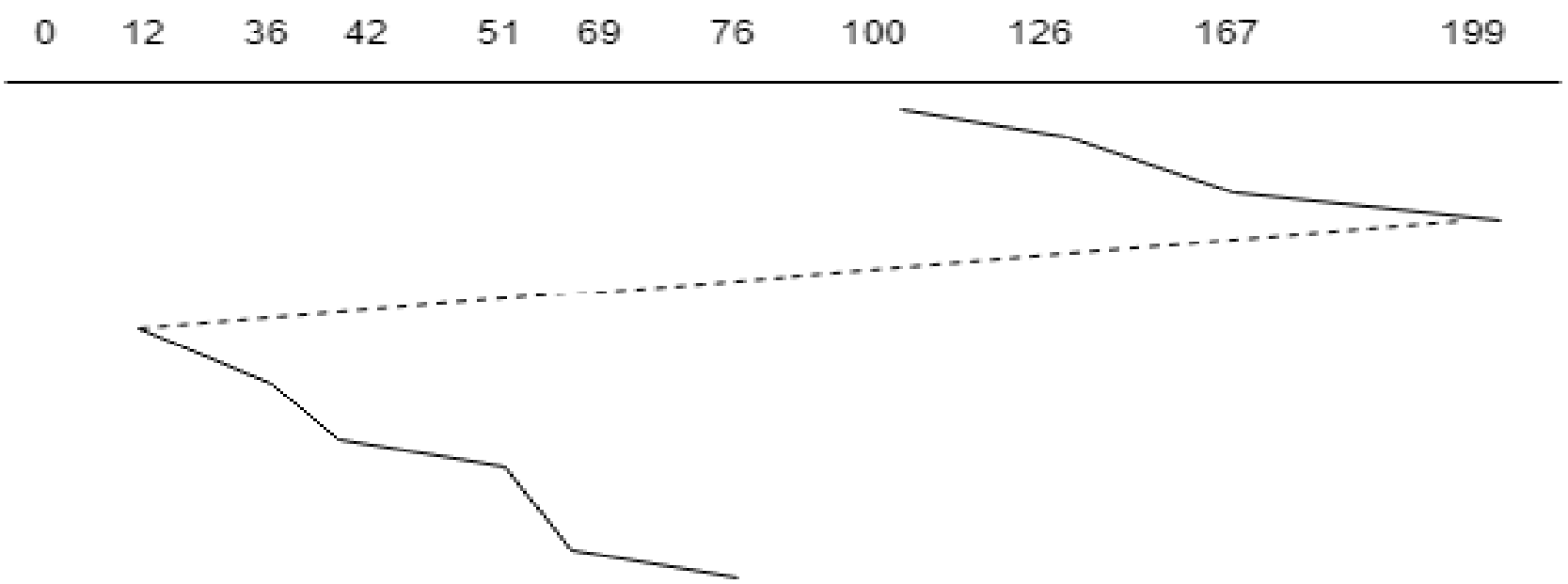
(A) 320 **(B)** 322 **(C) 321** **(D)** 319



Total disk movement = 66 + 61 + 23 + 110 + 2 = 321.

G4. There are 200 tracks on a disc platter and the pending requests have come in the order – 36, 69, 167, 76, 42, 51, 126, 12 and 199. Assume the arm is located at the 100 track and moving towards track 199. If sequence of disc access is 126, 167, 199, 12, 36, 42, 51, 69 and 76 then which disc access scheduling policy is used? (ISRO 2014)

(A) Elevator **(B)** Shortest seek-time first **(C) C-SCAN** **(D)** First Come First Served



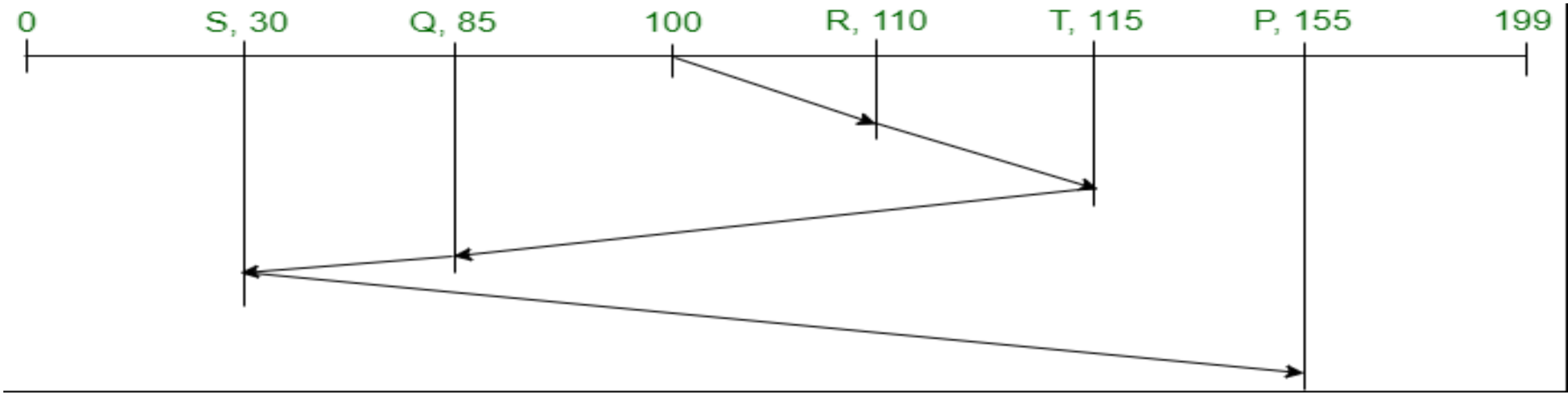
G5. Consider the following five disk access requests of the form (request id, cylinder number) that are present in the disk scheduler queue at a given time. (P, 155), (Q, 85), (R, 110), (S, 30), (T, 115) Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek Time First scheduling to service the requests. Which one of the following statements is FALSE? (GATE 2020)

(A) T is serviced before P

(B) Q is serviced after S, but before T

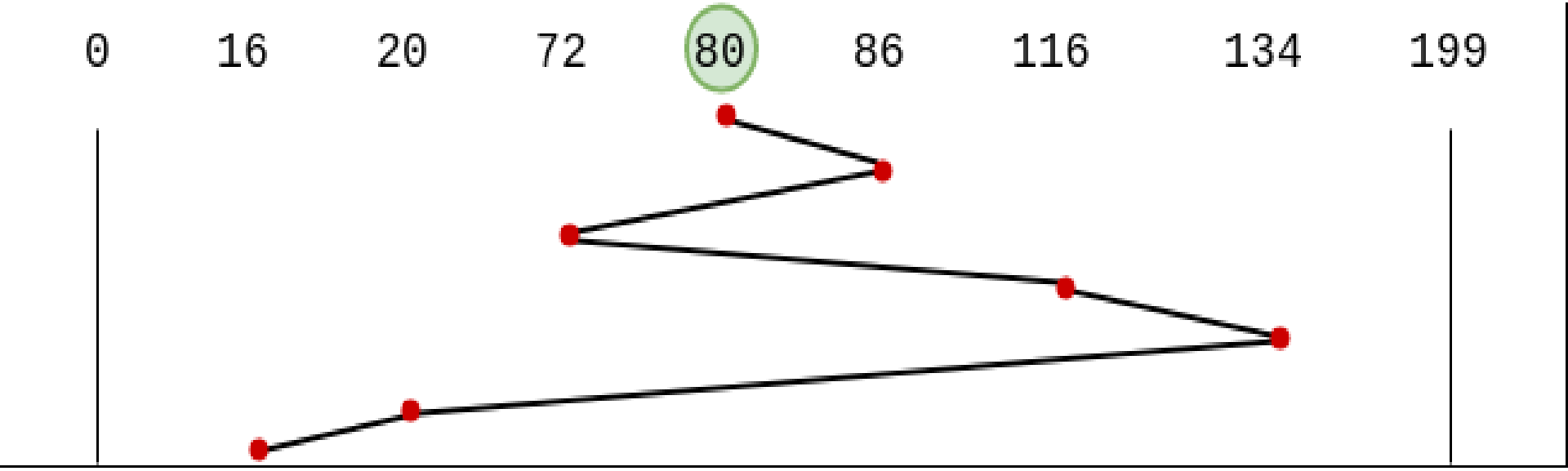
(C) The head reverses its direction of movement between servicing of Q and P

(D) R is serviced before P



G6. Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, … , 199), and 256 sectors per track (numbered as 0, 1, … 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time: [120, 72, 2], [180, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 16, 1] . Currently head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts. Power dissipation associated with rotational latency and switching of head between different platters is negligible. The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithm is _____ . (GATE 2018)

(A) 45 **(B)** 80 **(C) 85** **(D)** None of these



Head starts at 80. Then,Total Head movements in SSTF = (86-80) + (86-72) + (134-72) + (134-16) = 200

Power dissipated by 1 head movement = 20mW/100 = 0.2mW

Power dissipated by 200 movements : P1 = 0.2 mW * 200 = 40 mW

Power dissipated in reversing head direction once = 15 mW

Number of time Head changes its direction = 3

Power dissipated in reversing head direction: P2 = 3 * 15 = 45 mW

Total power consumption (in mW) is P1 + P2

= 40 mW + 45 mW = **85 mW**

File Concept

- A **file is a named collection of related information that is recorded on secondary storage.**
- From a user's perspective, a file is
  - the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- Commonly, files represent programs and data.
- Types:
  - Data –Numeric, Character ,Binary
  - Program
- Contents defined by file's creator
  - Many types
    - **text file, source file, executable file**

File Attributes

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

File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open*** **(F_i)**
- ***Close*** **(F_i)**

Open Files

- The operating system keeps a table, **called the open-file table, containing information about all open files**
  - **File pointer:** pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - **Disk location of the file**: cache of data access information
  - **Access rights:** per-process access mode information

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

File Types – Name, Extension

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

File Structure

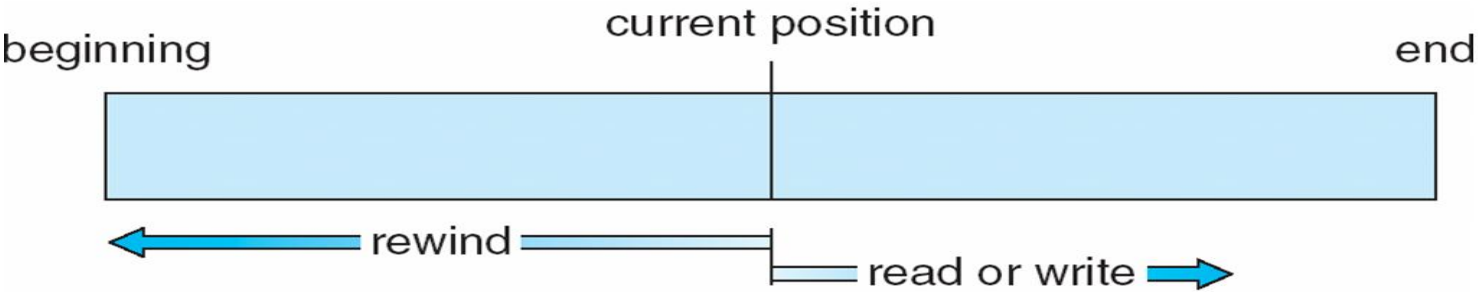
- Sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program

Access Methods

- A file is fixed length **logical records**
- **Sequential Access**
- **Direct Access**
- **Other Access Methods**

Sequential Access

- Operations
  - **read next**
  - **write next**
  - **Reset**
  - no read after last write (rewrite)



Direct Access

- Operations
  - **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

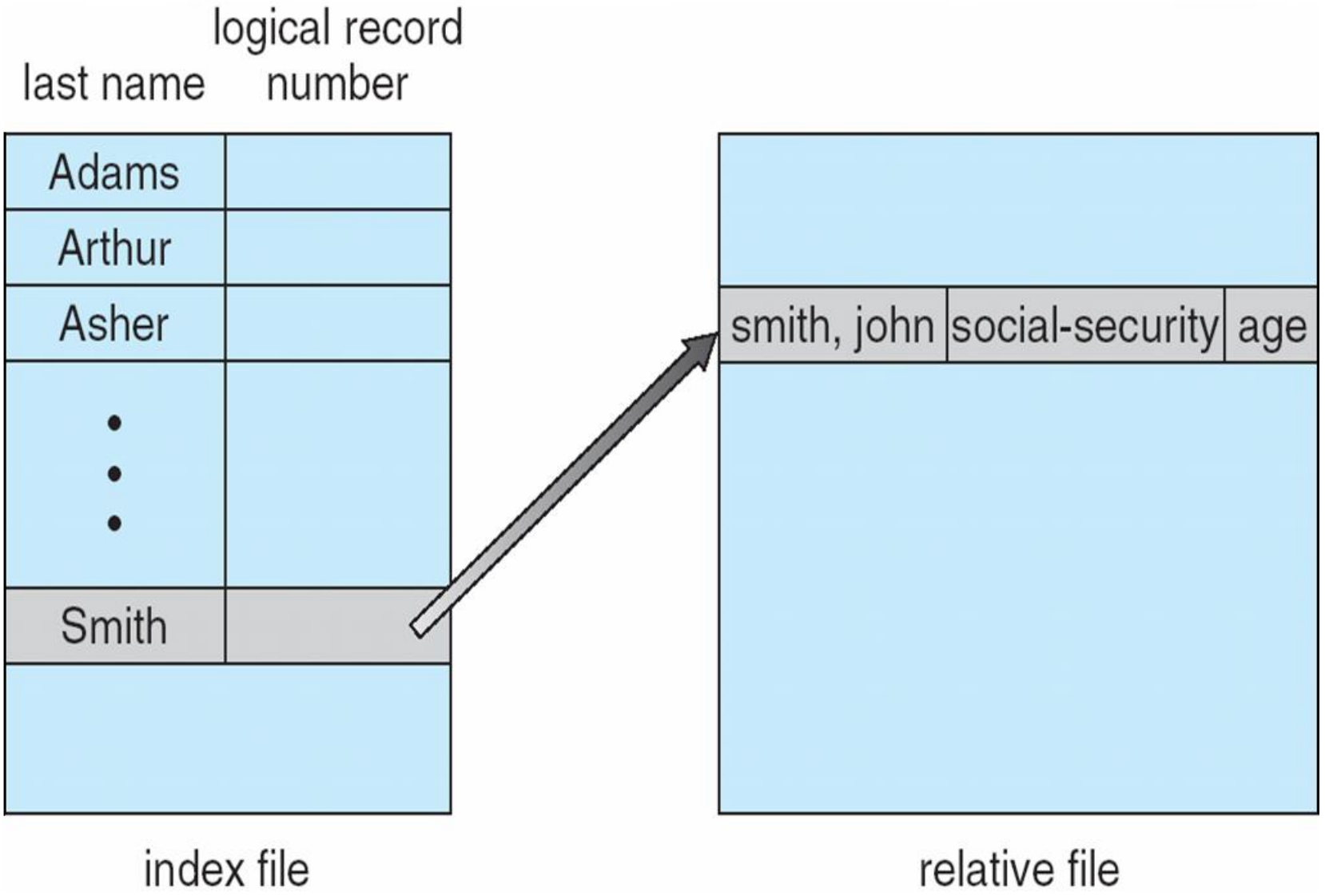
Simulation of Sequential Access on Direct-access File

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

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file. The index, like an index in the back of a book, contains pointers to the various blocks.
- To find a record in the file, first search the index and then use the pointer to access the file directly and to find the desired record
- If the index is too large, create an in-memory index, which an index of a disk index
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS

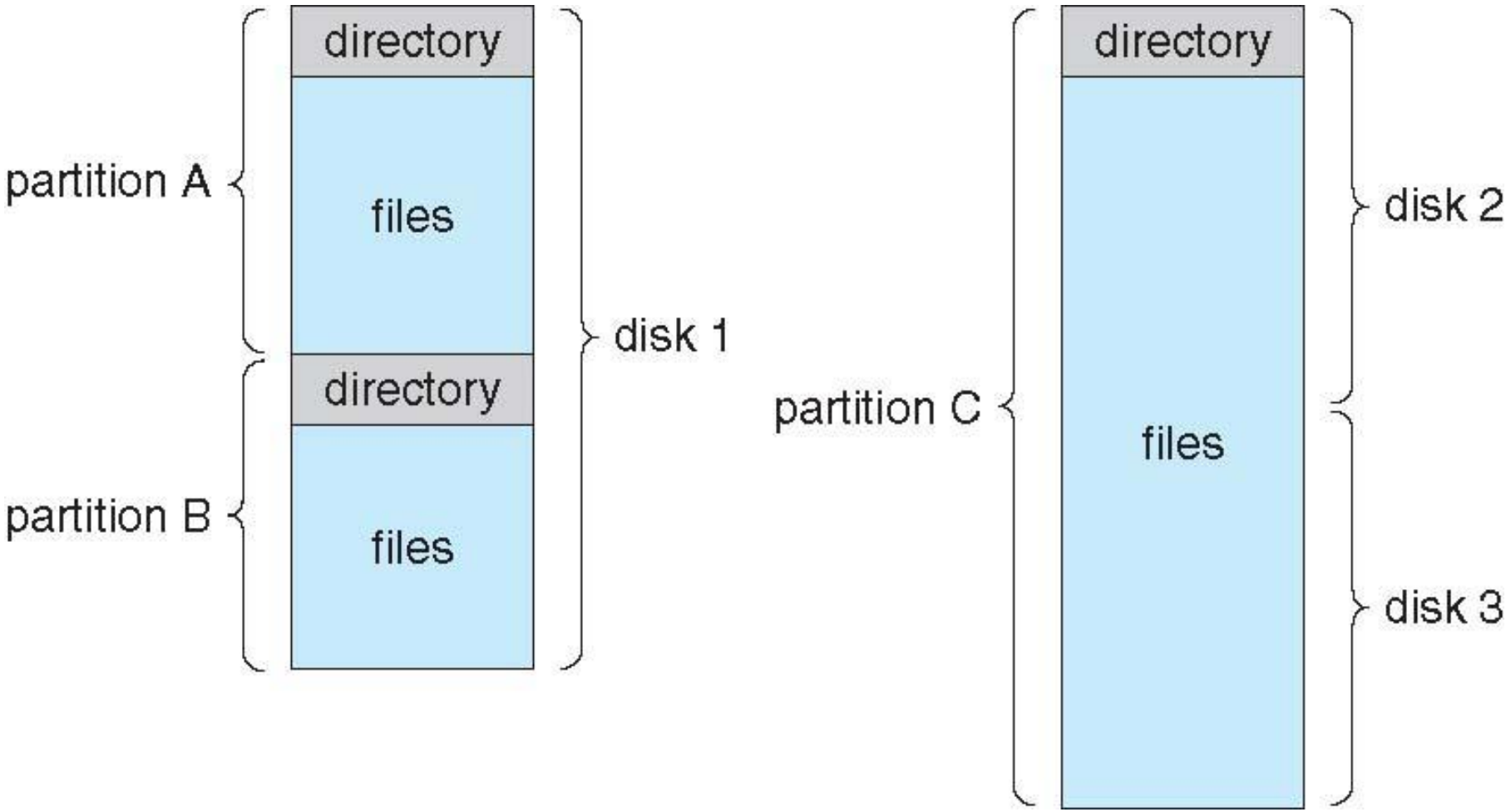
Example of Index and Relative Files



Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used both **raw** – without a file system and **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **volume**
- Each volume containing a file system also tracks that file system's info in **device directory** or **volume table of contents**
- In addition to **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization



Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

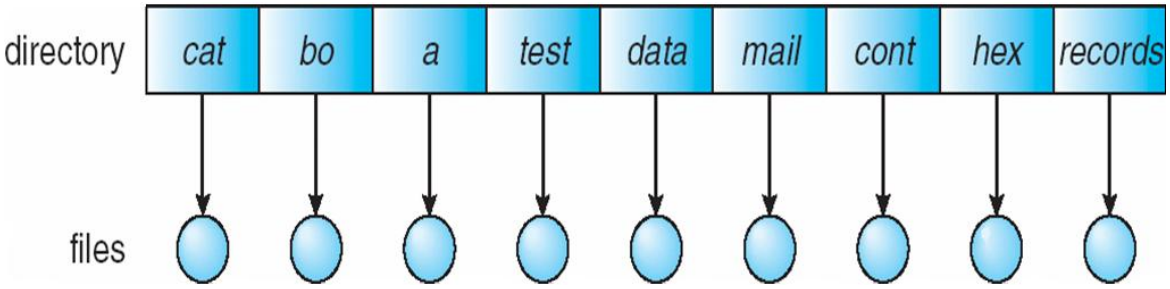
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A **single directory for all users**



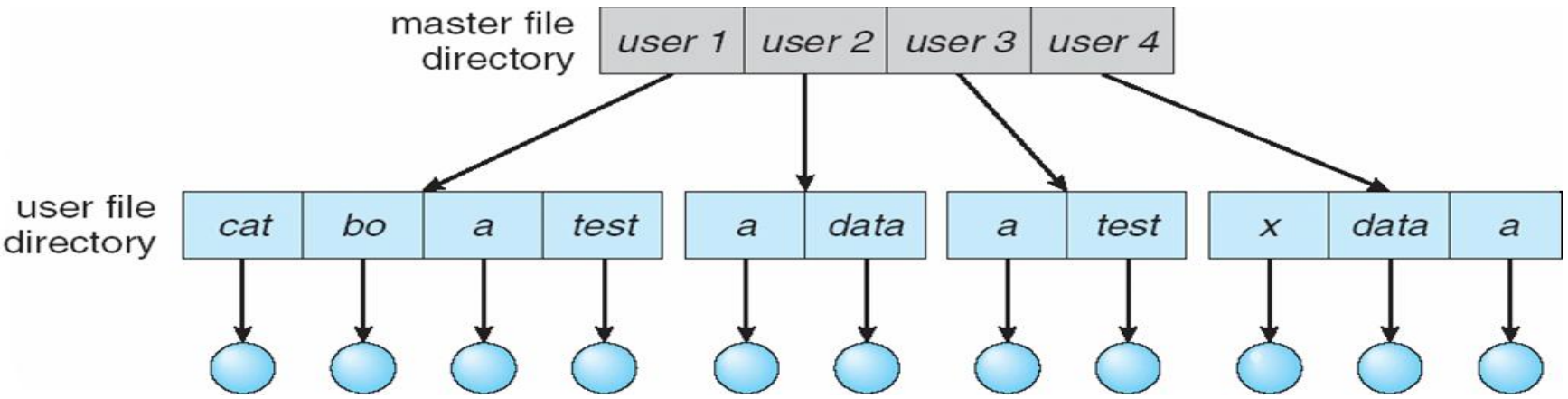
Advantages: Implementation is easy and File operations are easily done. Searching for files will be fast

Dis Advantages: 1. **Naming problem** – All files must have unique names since all the files are in the same directory

2. **Grouping problem :** Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system. Keeping track of so many files is a daunting task.

Two-Level Directory

- **Separate directory for each user**



In the two-level directory structure, each user has his own **user file directory (UFD).** The UFDs have similar structures, but each lists only the files of a single user.

Information about each UFD is stored in a system directory called as Master File Directory(MFD). 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

- Path name [eg] /user1/test or /user3/test or /user2/data
- Can have the same file name for different user
- Efficient searching

Disadvantages: 1.Users are isolated from each other. When two or more users wants to work on a task and access the same file, it becomes difficult because system doesn't permit this

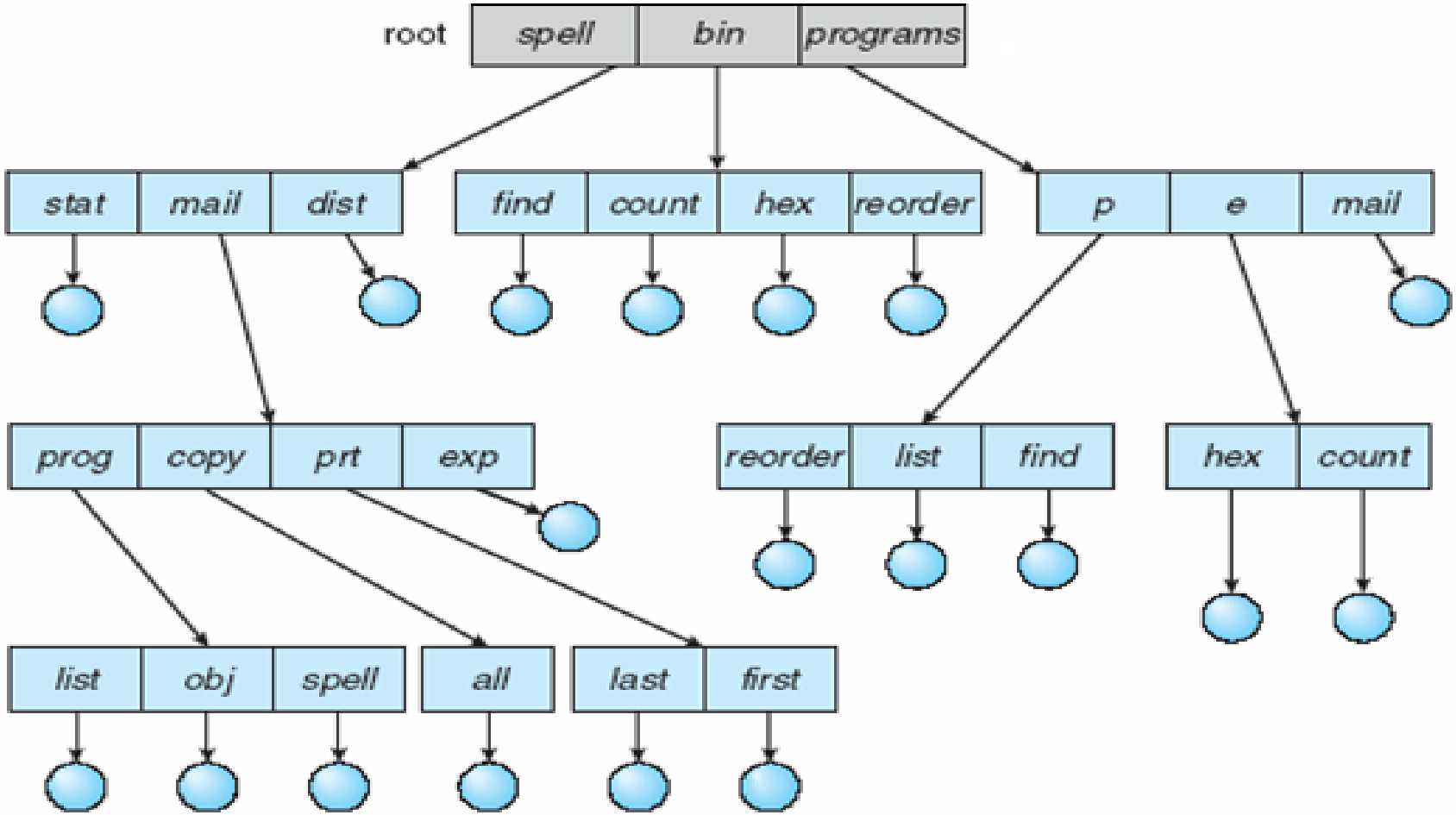
2. **Every system has its own syntax** for naming files and directories.

[ex] In MS-DOS C:\user1\test (ie) Drivename username directoryname.

3. **There are system files that has to be used by several users.** When these files are to be accessed by user, it will be searched in that user's UFD. But it wont be available.

Solution to this problem: 1.Copy the system files to each UFD – Waste large amount of space/memory .**2. A special user directory is defined to contain the system files[eg:user 0].**Whenever a file name is given, OS first searches the local UFD. If the file is found, it is used. If it is not found, the system automatically searches the special user directory that contain the system files.

Tree-Structured Directories



- In normal use, each process has a current directory. The current directory should contain most of the files that are of current interest to the process. When reference is made to a file, the current directory is searched.
- If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.
- Path names can be of two types: **absolute and relative**.
- In UNIX and Linux, an **absolute path name begins at the root and follows a path down to the specified file,** giving the directory names on the path.
- A **relative path name defines a path from the current directory**.
- For example, if the current directory is /spell/mail, then the relative path name prt/first refers to the same file as does the absolute path name /spell/mail/prt/first

Deletion of a directory:

If a directory is empty, its entry in the directory that contains it can simply be deleted.

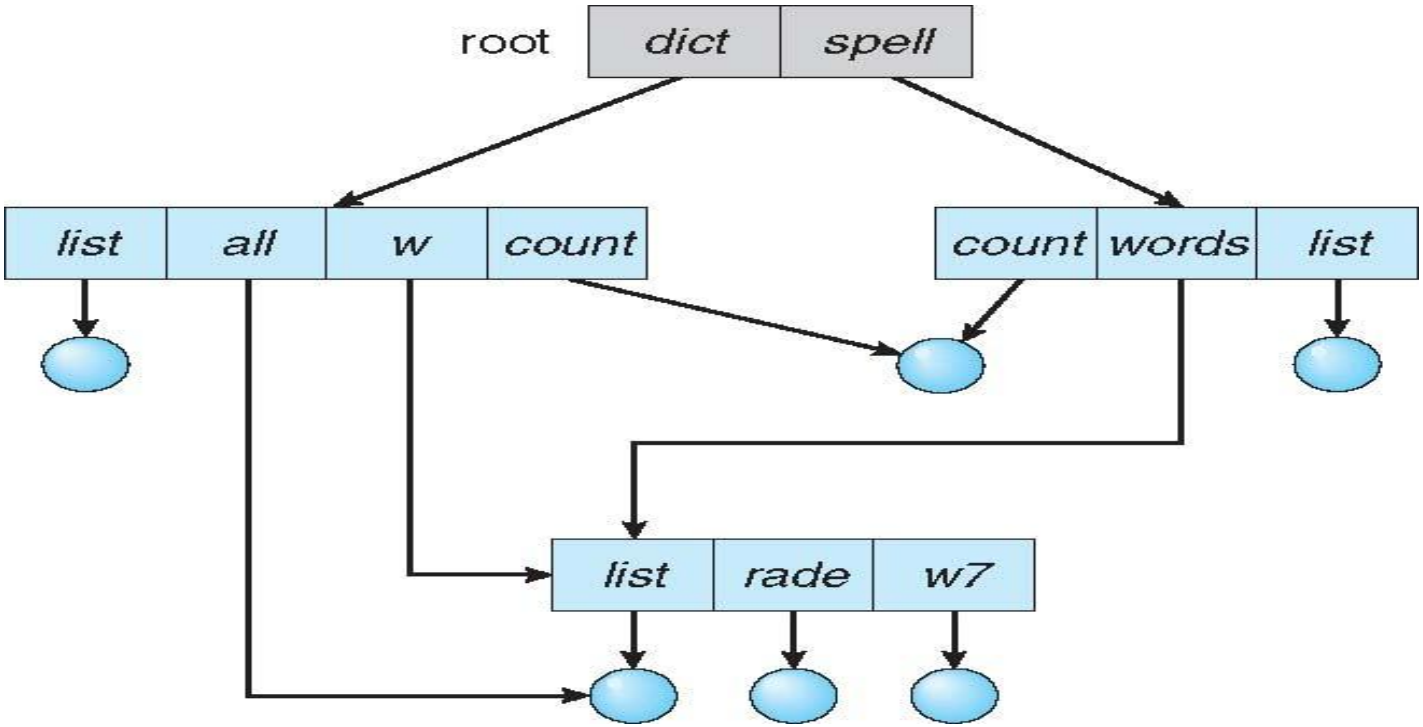
However, suppose the directory to be deleted is not empty but contains several files or subdirectories. **Some systems will not delete a directory unless it is empty.**

One of two approaches can be taken.

1. To delete a directory, the user must first delete all the files in that directory. If any subdirectories exist, this procedure must be applied recursively to them, so that they can be deleted also. This approach can result in a substantial amount of work.
2. An alternative approach, such as that taken by the UNIX rm command, is to provide an option: when a request is made to delete a directory, all that directory's files and subdirectories are also to be deleted

Acyclic-Graph Directories

- Have shared subdirectories and files
- Example



Acyclic-Graph Directories (Cont.)

Problems:

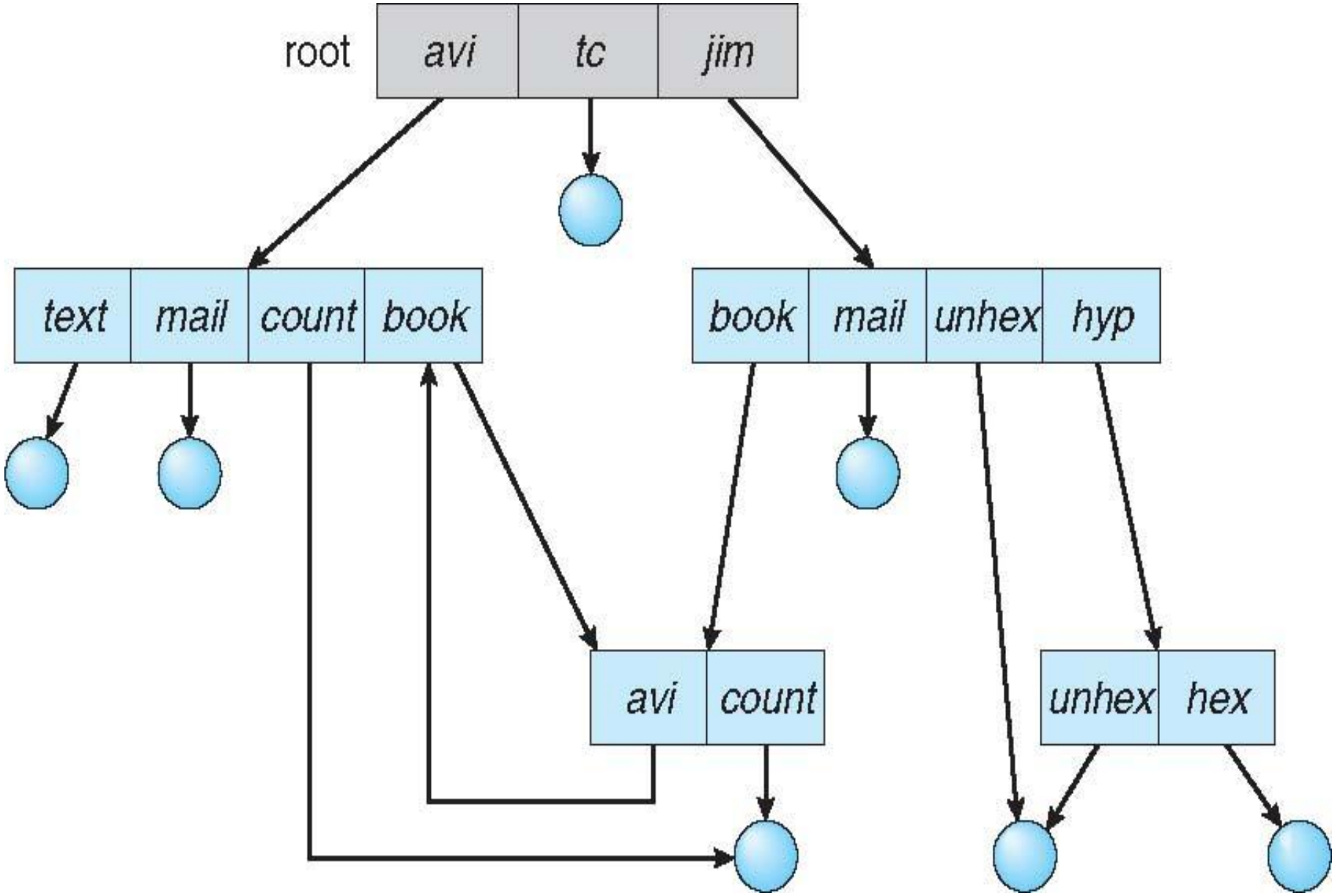
- **Same file with many paths and names:** A file may now have multiple absolute path names. Consequently, distinct file names may refer to the same file. This situation is similar to the aliasing problem for programming languages. To find a file, to accumulate statistics on all files, or to copy all files to backup storage— this problem becomes significant.
- **Deleting of shared files:**

  If the file is deleted by someone, it will lead to gangling pointers to the non – existent file.

  If shared files are implemented using symbolic links, then deletion of a file by one user will delete only the link and won't affect the original file. But, if the original file is deleted, it will leave the links dangling. Later, these links can be removed.

  Another approach: preserve the file until all references to it are deleted

General Graph Directory



General Graph Directory (Cont.)

- No guarantee for no cycles in Acyclic graph directory

Problems in Graph directory:

Searching for files: 1. If cycles are allowed to exist in the directory, avoid searching any component twice for the reason of correctness

2. A poorly designed search algorithm result in an infinite loop continually searching through the cycle and never terminating

Deleting files: With acyclic-graph directory structures, a value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted. However, when cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file because of the possibility of self – referencing(cycle). To avoid this, use Garbage collection scheme.

Garbage collection:

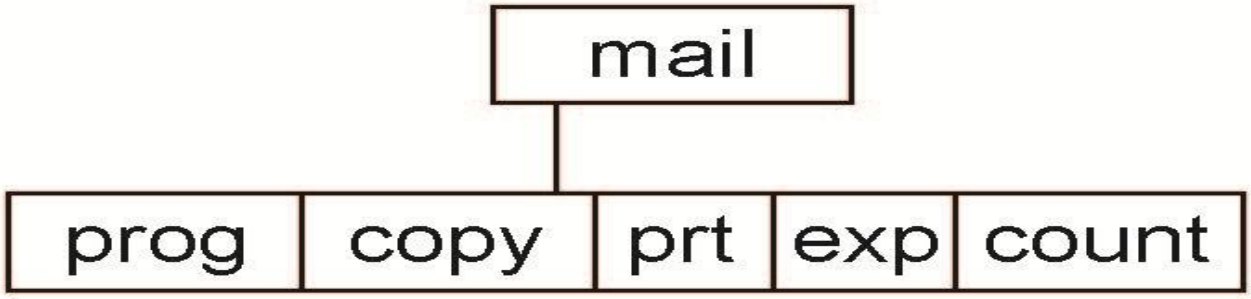
- Traversing the entire file system, marking everything that can be accessed.
- Then, a second pass collects everything that is not marked onto a list of free space.
- Garbage collection for a disk-based file system, however, is extremely time consuming and is thus seldom used.

Current Directory

- Can designate one of the directories as the current (working) directory
  - **cd /spell/mail/prog**
  - **type list - view the files**
- Creating and deleting a file is done in current directory
- Example of creating a new file
  - If in current directory is **/mail**
  - The command

mkdir <dir-name>

  - Results in:



  - Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Protection

- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups in Unix

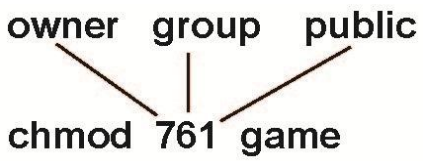
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			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 file (say *game*) or subdirectory, define an appropriate access.[chgrp- change the group ownership]

```
chgrp      G      game
```

- Attach a group to a file



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/

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** or **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

Per – File FCB

It contain many details about the file such as

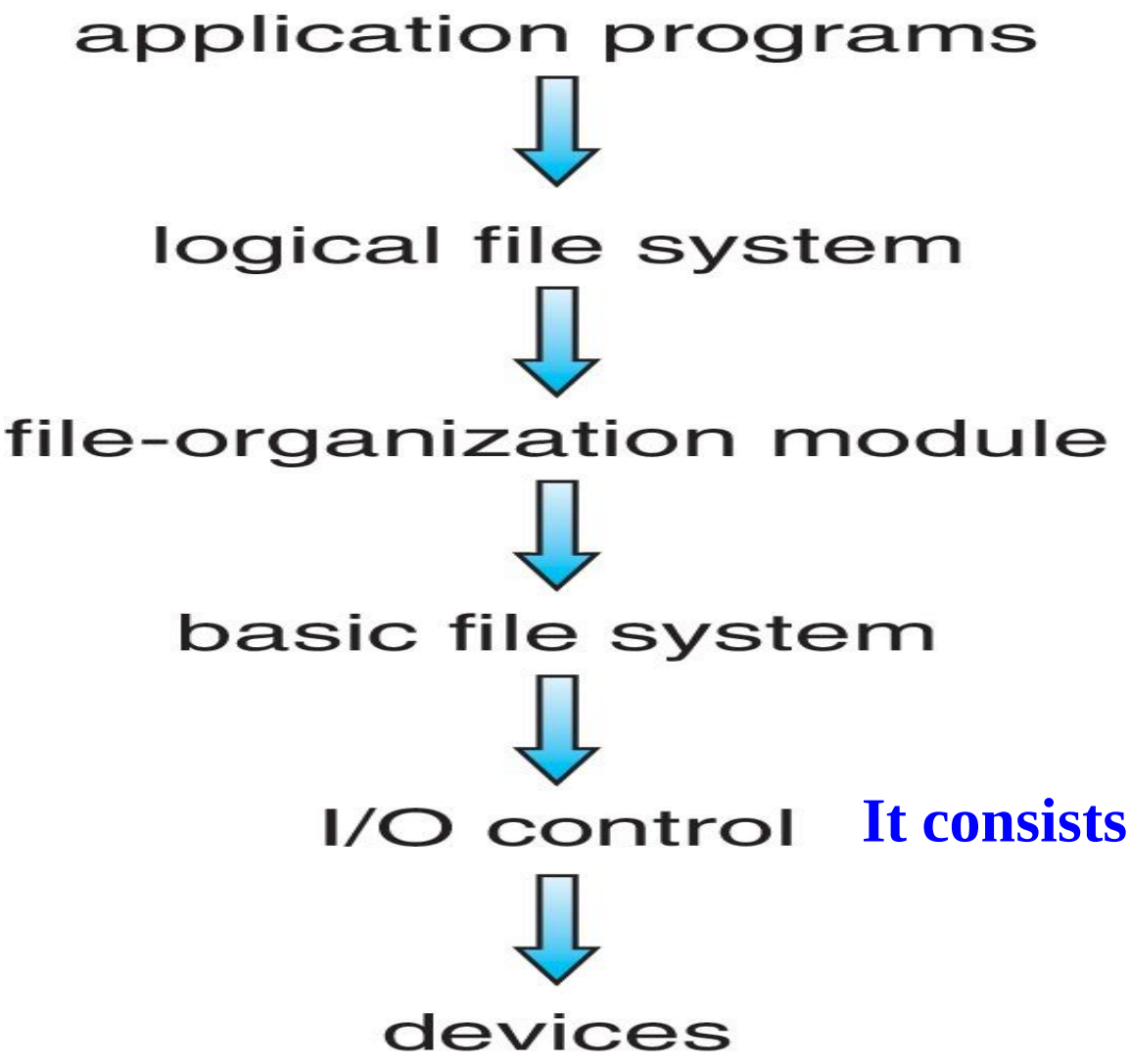
- File permissions
- Ownership
- Size
- Location of data block

File permissions
File dates (create, access, write)
File owner, ACL
File size
File data blocks

In Unix it is called as inode.

In NTFS it is stored in the master file table.

Layered File System



It consists of Device Drivers

File System Layers

- **I/O control level** consists of device drivers to **transfer information between the main memory and the disk system**. A device driver can act as a translator. Its input consists of high level commands, such as "retrieve block 123." Its output consists of low-level, hardware-specific instructions that are used by the hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver. Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Logical layers can be implemented by any coding method according to OS designer

File System Layers (Cont.)

- Many file systems, sometimes many within an operating system
  - Each with its own format:
  - Unix has **UFS(Unix File System), FFS(Fast File System)**
  - Windows has FAT, FAT32, NTFS (Windows NT File System) as well as floppy, CD, DVD Blu-ray,
  - Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS(Z File System by sun micro systems for solaris), GoogleFS, FUSE(File system and USE for unix without editing Kernal code)

File-System Operations

- Several **on-storage** and **in-memory** structures are used to implement a file system
- **On storage-** The file system contain information about how to boot an operating system , the total number of blocks, the number and location of free blocks, the directory structure, and individual files.
- **In-memory** - Information is used for **both file-system management and performance improvement** via caching.

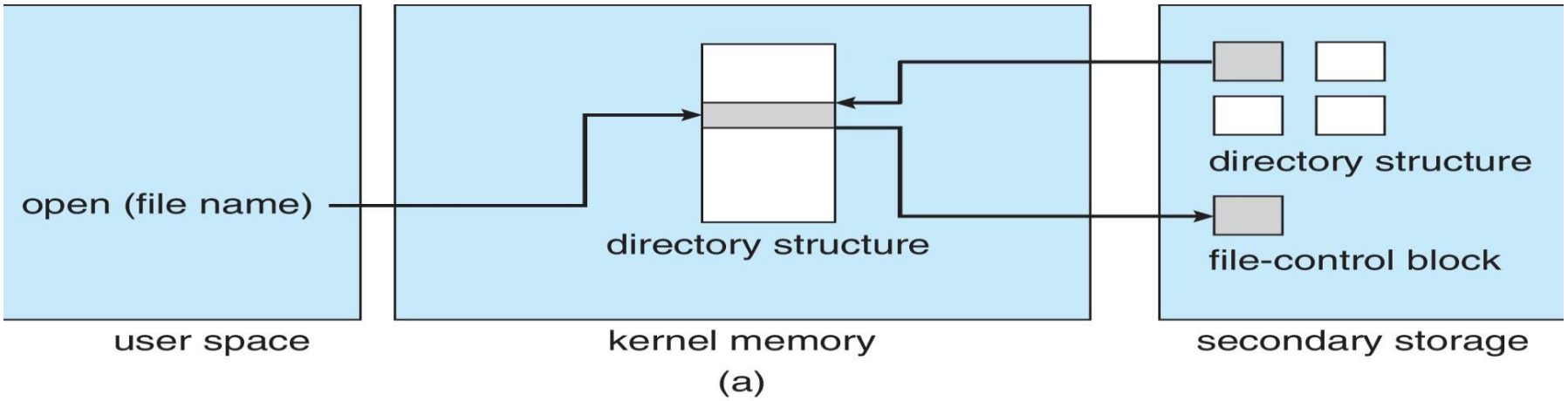
On-Storage Structures

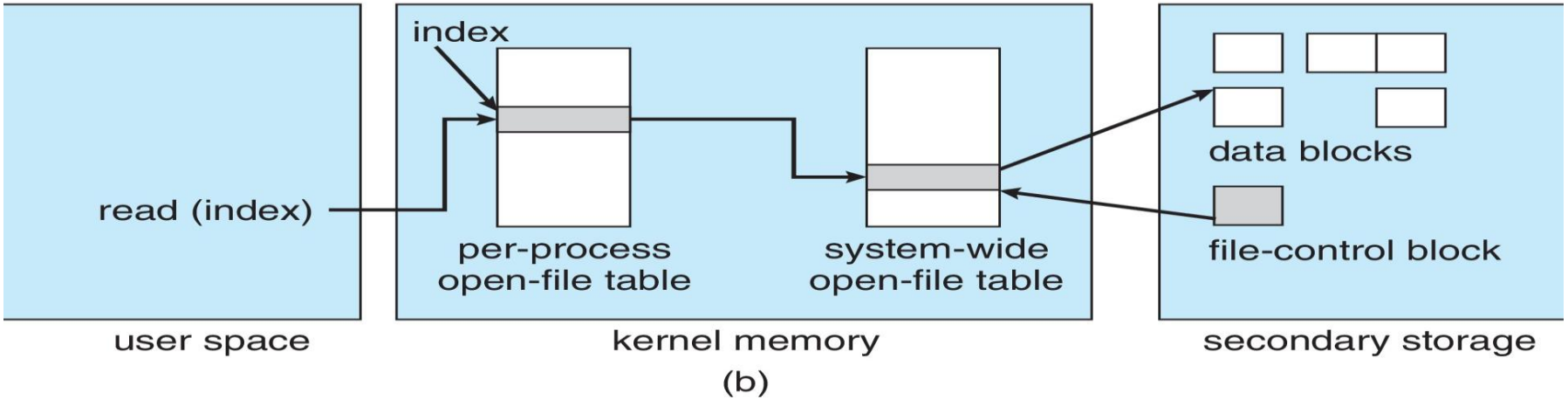
- **Boot control block** contains information needed by system to boot OS from that volume and usually first block of volume. **In UFS, it is called the boot block. In NTFS, it is the partition boot sector.**
- **Volume control block (In UFS called as superblock and in NTFS called as master file table)** contains volume details, Total # of blocks, # of free blocks, block size, free block pointers
- **Directory structure** organizes the files along with Names and inode numbers. In NTFS it is stored in master file table.

In-Memory File System Structures

- **Mount table** storing file system mounts, mount points, file system types
- **System-wide open-file table** contains a copy of the FCB of each file and other information
- **Per-process open-file table** contains pointers to appropriate entries in system-wide open-file table as well as other information

In-Memory File System Structures (Cont.)





- Figure (a) refers to opening a file
- Figure (b) refers to reading a file

Directory Implementation by two techniques

1. Linear list of file names with pointer to the data blocks

- Simple to implement but Time-consuming to execute;Linear search time

To create a new file : Search the directory to ensure that no existing file has the same name; Add a new file at the end of the directory

To delete a file : Search the directory for the named file; Release the space allocated to it.

Dis Advantage : Linear search is used to find the files and this will slow down the entire process

2. Hash Table – linear list with hash data structure and it decreases directory search time.

Dis Advantage: 1. Collisions – situations where two file names hash to the same location. Solution is use chained-overflow method

2. Hash tables are fixed size and hash function depends on that size

Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous Disk space allocation
  - Linked Disk space allocation
  - File Allocation Table (FAT)
  - Indexed Allocation Method

Contiguous Allocation Method

- Each file occupies set of contiguous blocks on the disk.
- Contiguous allocation of a file is defined by the disk address and length(in block units) [Eg] – If a file is ‘n’ blocks and starts at location ‘b’ , then it occupies the blocks b,b+1,b+2,---- b+n-1.
- The directory entry for each file indicates the address of the starting block and length of the area allocated for this file.

Advantages: Best performance in most cases

- Simple – only starting location (block #) and length (number of blocks) are required

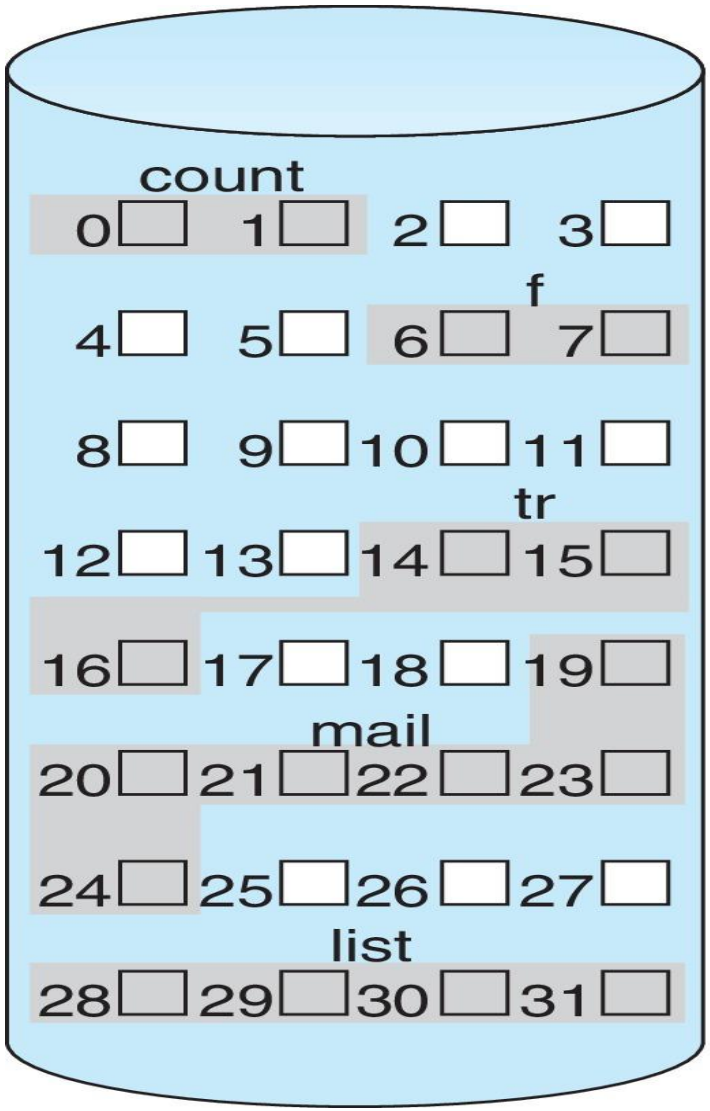
For Sequential Access:

The file system remembers the disk address of the last block referenced. Then it reads the next block.

For Direct Access :

For accessing block ‘i’ of a file that starts at block ‘b’. Then immediately access block b + i block

Contiguous Allocation (Cont.)



directory

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

Dis Advantages:

1. **Finding space on the disk for a new file is difficult**

   - First fit and best fit algorithms are used to select a free hole from the set of available holes

   - Both these algorithms suffer from external fragmentation. So need for **compaction off-line (downtime)** or **on-line**

2. **Knowing how much space is needed for a file is a difficult task.**

   - If the space allocated for a file is too less and it needs more space later, then

   - Either the user program is terminated and the user allocate more space to run the program again (OR)
   - the system finds a larger hole, copies the content to the new space and releases the previous space

Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Here, a **contiguous chunk of space is allocated initially. Then, 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
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks- A file consists of one or more extents

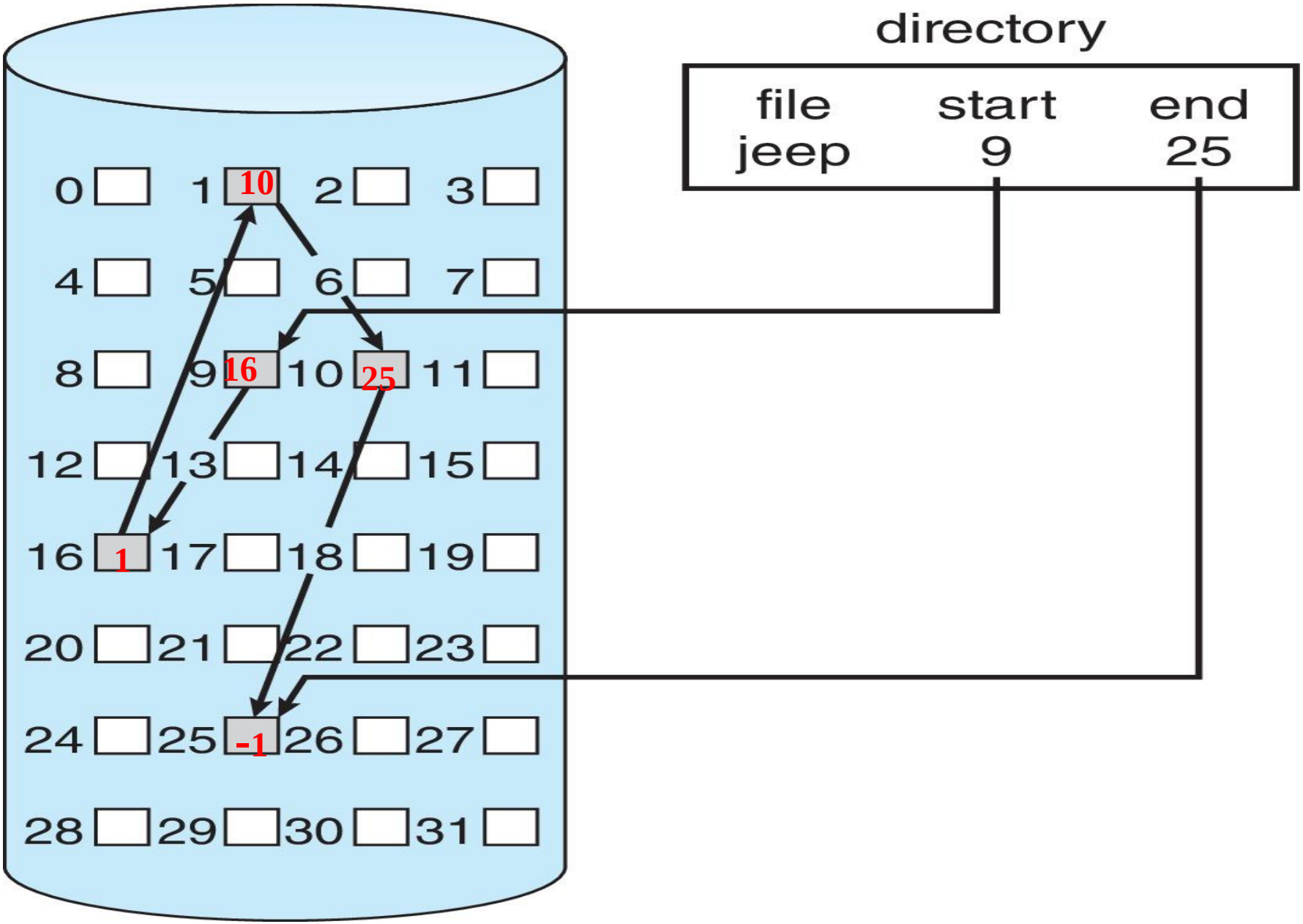
Linked Allocation

- **Each file is a linked list of blocks which may be scattered anywhere on the disk**
- The **directory contains a pointer to the first and last blocks of the file**
- Each block contains a pointer to the next block and File ends at nil pointer . If each block is 512 bytes in size, and a **disk address(the pointer) requires 4 bytes,** then the user see the block of 508 bytes

Advantages:

- No external fragmentation and No compaction
- Each block contains pointer to next block
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups

Linked Allocation Example



Linked Allocation (Cont.)

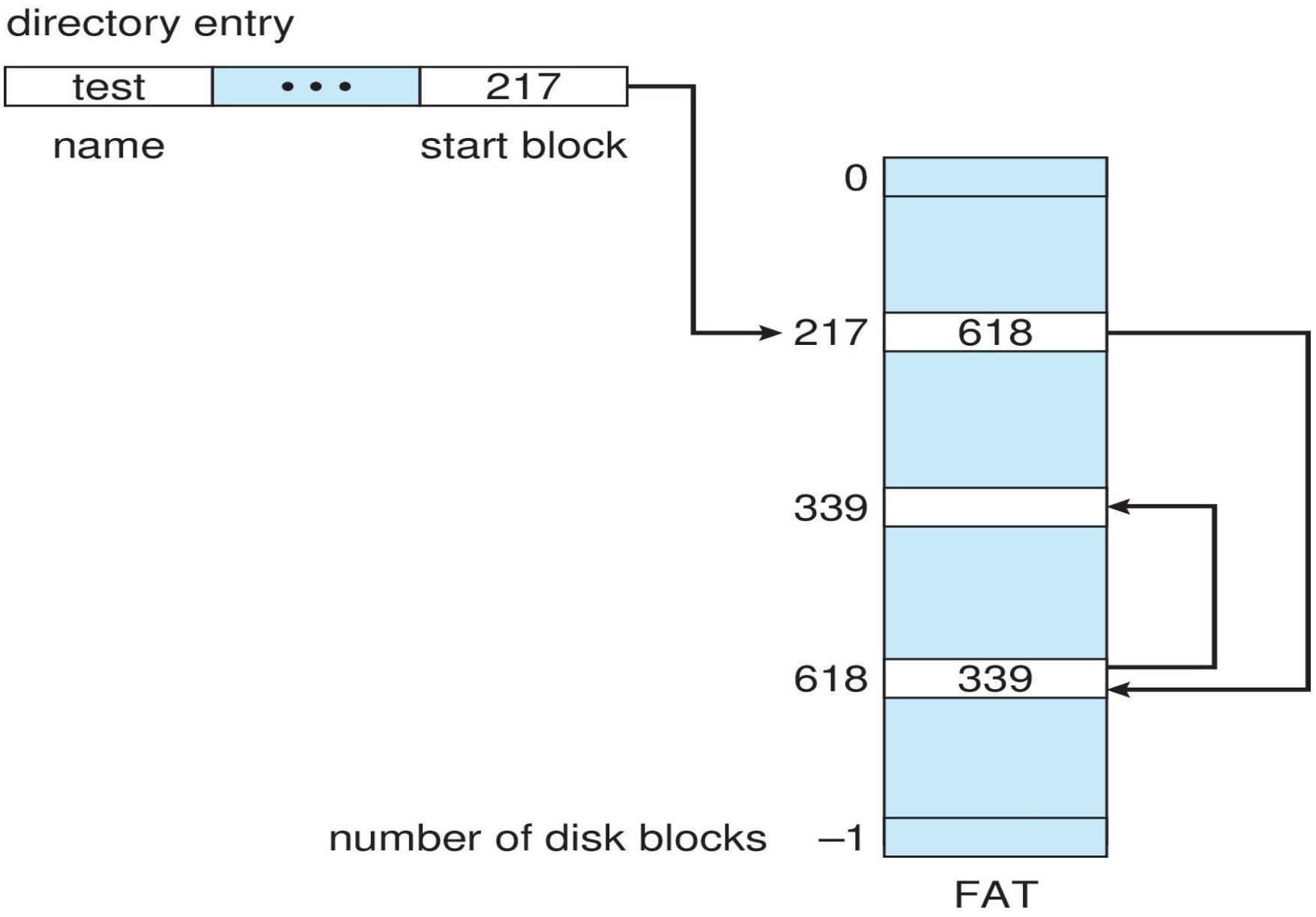
Dis Advantages:

1. **It can be used effectively only for sequential access not for direct access**. To find the i^{th} block , start at beginning of that file and follow the pointers until to get the i^{th} block .
2. **Space required for the pointers** - If a pointer requires 4bytes out of a 512 byte block, then 0.78 percent of the disk is used for pointers
3. **Reliability problem -** If the pointers are used for linking the files gets damaged or lost or wrong pointers are picked up leads to problems

FAT Allocation Method

- This method is used by MS-DOS
- Beginning of volume has table**, the table has one entry for each disk block and indexed by block number**
- The **directory entry contains the block number of the first block of the file**
- The **table entry indexed by that block number contains the block number of the next block in the file. The chain continues until the last block, which has a special end –of- file value as the table entry**
- Unused blocks are indicated by a ‘0’ table value

File-Allocation Table



Allocating a new block to a file:

- Find the first 0-valued table entry
- Replace the previous end-of-file value with the address of the new block
- Replace ‘0’ with the end-of-file value

Advantage : Random access time is improved

Dis Advantage : Not suitable for Direct access

Indexed Allocation Method

- The **main disadvantage of linked disk allocation is, it can not support for direct access**.
- **Indexed allocation method** solves this problem by **bringing all the pointers together into one location called as index block**
- **Each file has its own index block-** 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, use the pointer in the i^{th} index block entry

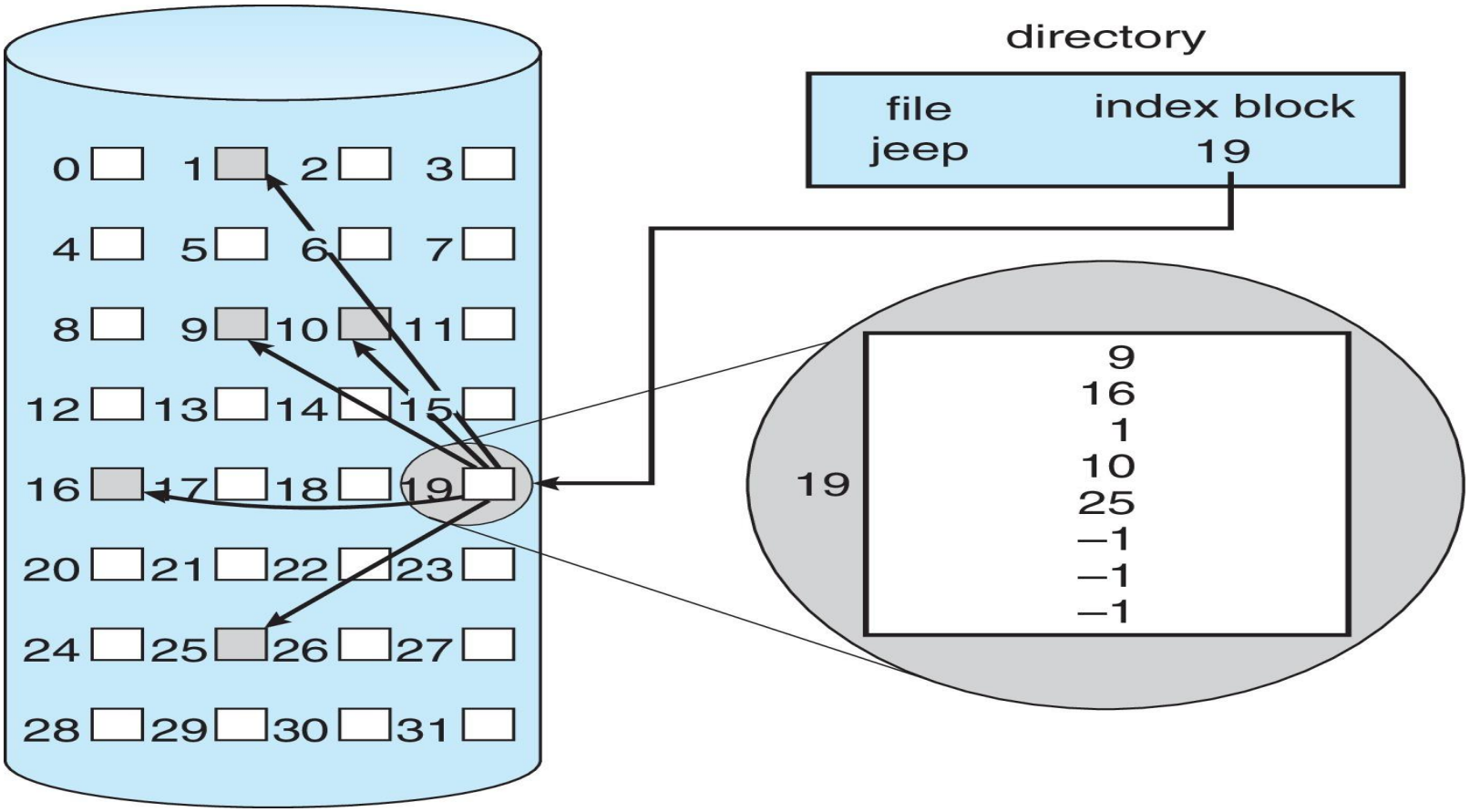
Create a new file :

Step1 : All pointers in the index block are set to null.

Step2 : When the i^{th} block is first written, a block is obtained from the free space manager.

Step3 : Its address is put in the i^{th} index block entry

Example of Indexed Allocation



Advantages:

1. Support direct access, without suffering external fragmentation
2. Any free block on the disk can satisfy a request for more space

DisAdvantages:

1. More wasted space
2. Pointer overhead of the index block is generally greater than the pointer overhead of linked allocation

How large should the index blocks be?

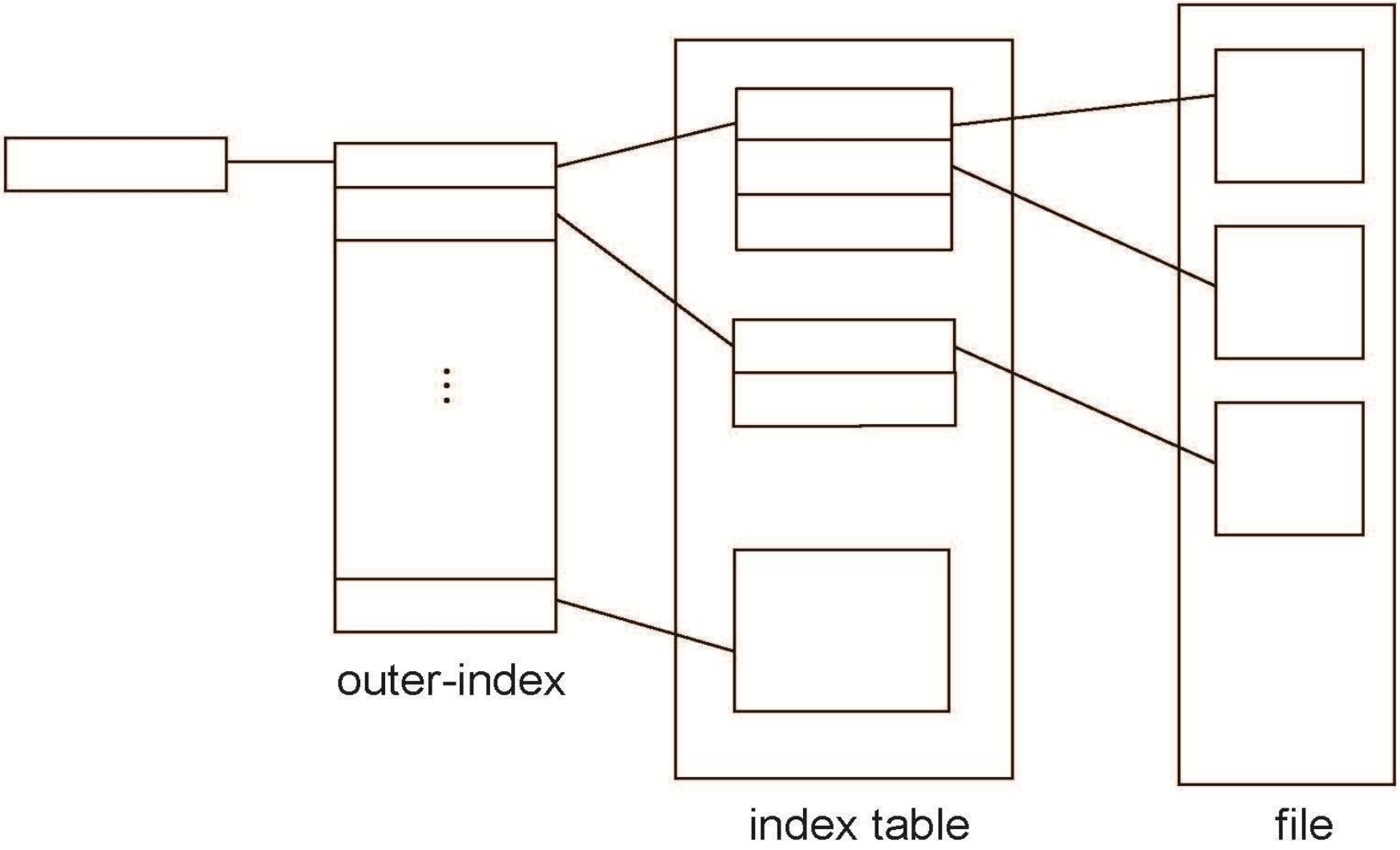
Every file must have an index block, so the index block to be as small as possible. If the index block is too small, it will not be able to hold enough pointers for a large file.

Mechanisms for index block to hold pointers for large file:

1. **Linked scheme:** An index block is normally one storage block. Thus, it can be read and written directly by itself. To allow for large files, link together several index blocks. For example, an index block might contain a header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is null (for a small file) or is a pointer to another index block (for a large file).

2. Multilevel index : A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size.

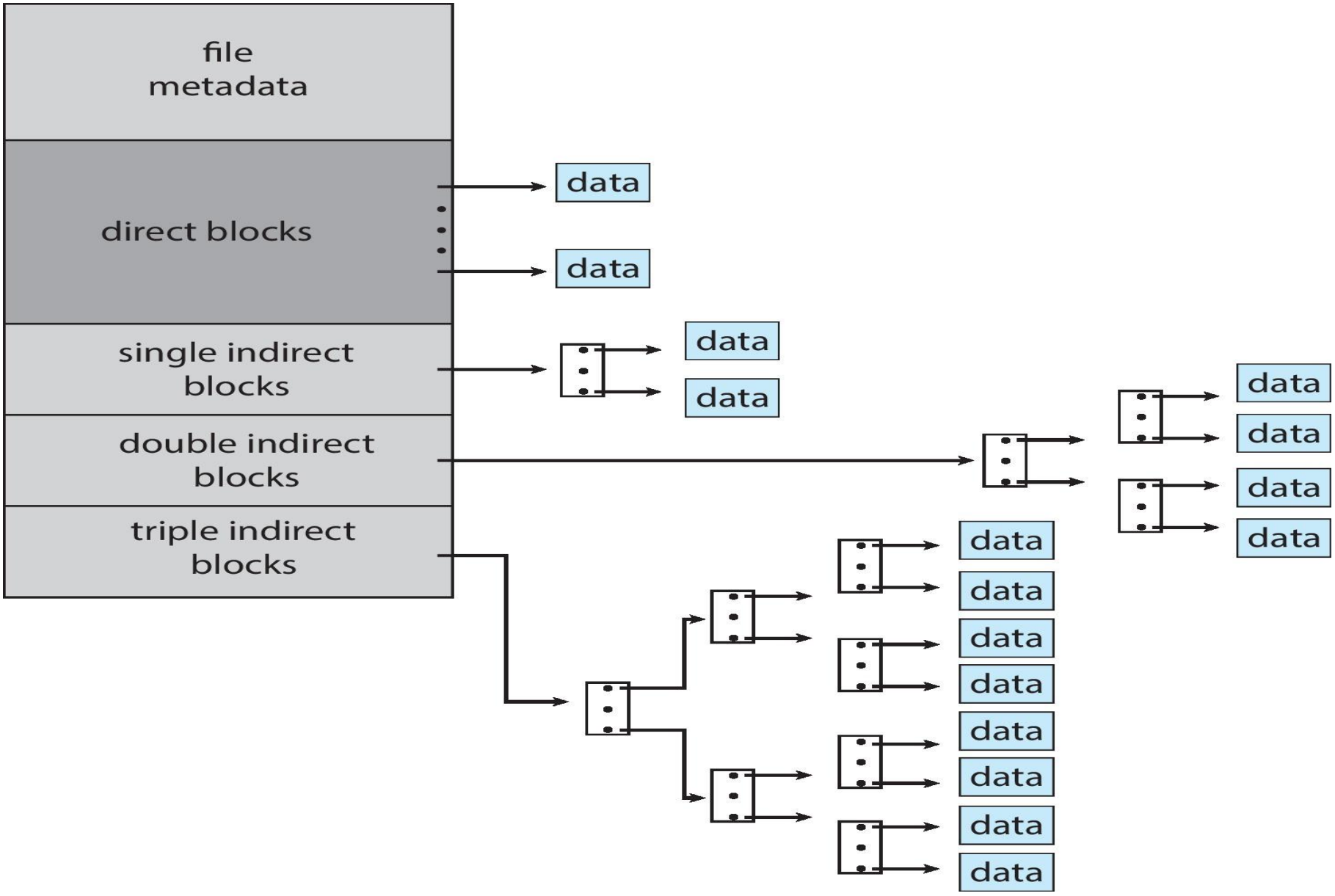
Indexed Allocation – Two-Level Scheme



How large should the index blocks be?

3.Combined scheme : In **UNIX-based file systems**, there are pointers [ex :15] of the index block in the file's inode. The first 12 of these **pointers point to direct blocks**; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files (of no more than 12 blocks) do not need a separate index block. If the block size is 4 KB, then upto 48 KB of data can be accessed directly. The next three **pointers point to indirect blocks**. The **first points to a single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The **second points to a double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The **last pointer contains the address of a triple indirect block.**

Combined Scheme : UNIX UFS



Performance

- Best method depends on file access type
  - **Contiguous great for sequential and random**
- **Linked good for sequential, not random**
- Declare access type at creation
  - Select either contiguous or linked
- **Indexed more complex**
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead
- For **NVM, no disk head so different algorithms and optimizations needed**
  - Using old algorithm uses many CPU cycles trying to avoid non-existent head movement
  - Goal is to reduce CPU cycles and overall path needed for I/O

Free-Space Management

- File system maintains **free-space list** to track available blocks

The techniques are

1. Bit vector or bit map

2. Linked Free Space List

3. Grouping

4.Counting

Bit vector or bit map Technique

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.

For 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 bitmap would be

001111001111110001100000011100000 ...

Advantages: simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.

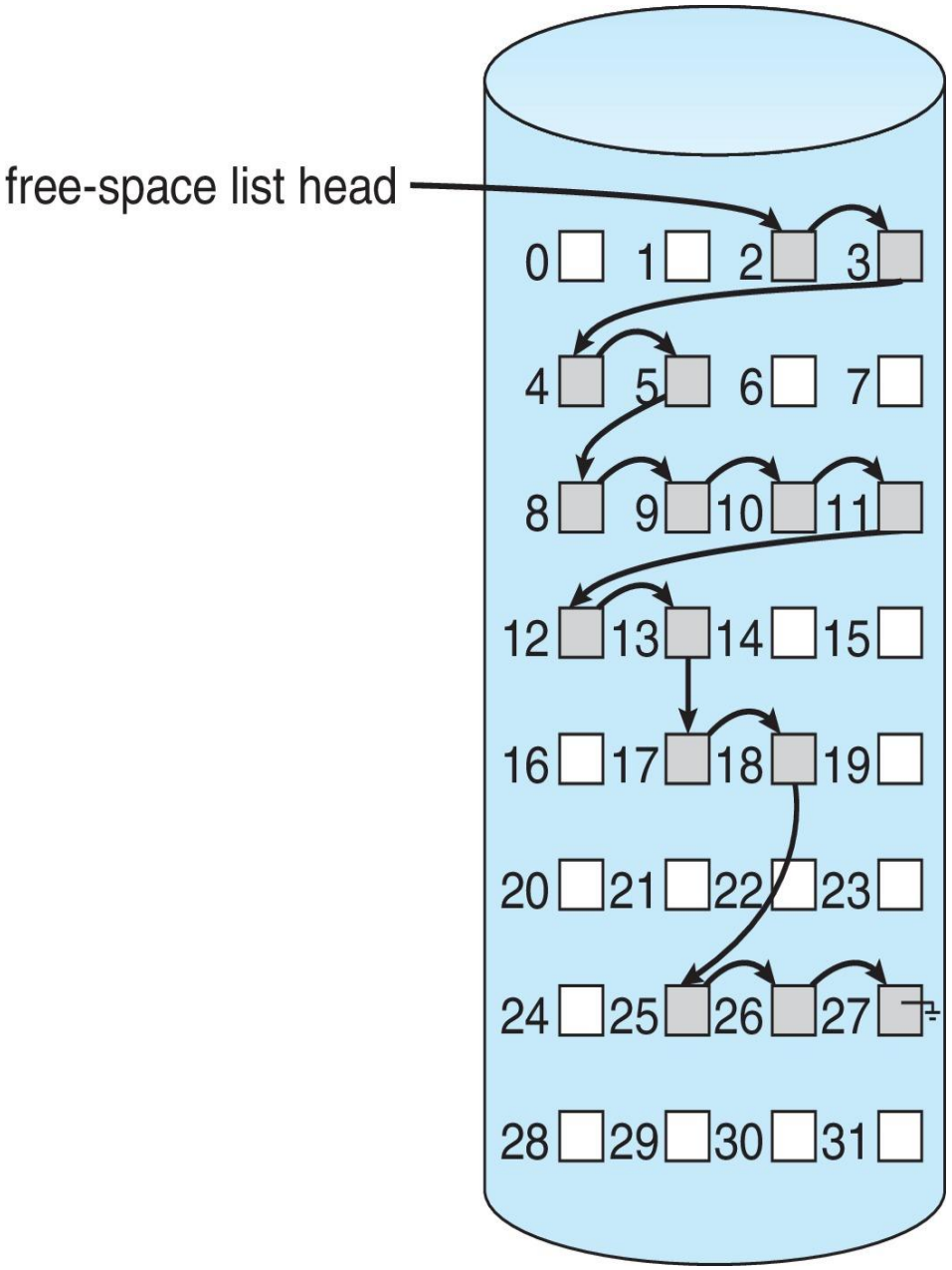
DisAdvantages: The entire vector is kept in main memory; bitmaps must be modified both when blocks are allocated and when they are freed.

Linked Free Space List on Disk

- All the free disk blocks are linked together
- A pointer pointing to the first free block is kept in a special location on the disk.
- It is cached in memory
- The first block contains a pointer to the next free disk block and so on.

Advantage : No need to traverse the entire list (if # free blocks recorded)

Dis Advantage: Cannot get contiguous space easily



Grouping : It is the **modification of the free linked-list 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.

Counting : 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 block addresses, 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 device address and a count. Although each entry requires more space than would a simple disk address, the overall list is shorter, as long as the count is generally greater than 1.

Space Maps

- Used in Oracle **ZFS**
- Consider meta-data I/O on very large file systems
  - Full data structures like bit maps cannot fit in memory
- Divides device space into **metaslab** units and manages metaslabs
  - Given volume can contain hundreds of metaslabs
- Each **metaslab has associated space map**
  - Uses counting algorithm
- But **records to log file** rather than file system
  - Log of all block activity, in time order, in counting format
- Metaslab activity ➔ load space map into memory

- The in-memory space map is then an accurate representation of the allocated and free space in the metaslab.
- ZFS also condenses the map as much as possible by combining contiguous free blocks into a single entry. Finally, the free-space list is updated on disk

G1.A Unix-style i-node has **10 direct pointers** and **one single, one double and one triple indirect pointers**. **Disk block size is 1 Kbyte**, **disk block address is 32 bits**, and 48-bit integers are used. What is the maximum possible file size ? (GATE 2004)

(A) 2^{24} bytes (B) 2^{32} bytes **(C) 2^{34} bytes** **(D)** 2^{48} bytes

Size of the disk block = 1KB= 1024 bytes

Disk block address = 32bits = 4 bytes

Number of addresses per block = Block size/ space occupied by each address = 1024/4 = 256 = 2^8

Maximum size of file = (10 direct pointers $\times$1KB) + (1 single Indirect pointer $\times$1KB) + (1 Double Indirect pointer $\times$1KB) + (1 Triple Indirect pointer $\times$1KB)

$=(10 \times 2^{10}) + (2^8 \times 2^{10}) + (2^8 \times 2^8 \times 2^{10}) + (2^8 \times 2^8 \times 2^8 \times 2^{10})$

$= 2^{13} + 2^{18} + 2^{26} + 2^{34} = 2^{34}$ bytes $= 2^4 \times 2^{30} = 16$GB

$(10 \times 2^{10})$is written as $(10 \times 1024) = 10240 = 10100000000000 = 13$bits

G2. The index node (inode) of a **Unix-like file system has 12 direct**, one **single-indirect** and one **double-indirect** pointer **The disk block size is 4kB and the disk block addresses 32-bits long.** The maximum possible file size is (rounded off to 1 decimal place)_____ GB.(GATE 2019)

(A)4 **(B)**2 **(C)**1 **(D)** 0.50

Size of the disk block = 4KB= 4096 bytes = 2^{12}

Disk block address = 32bits = 4 bytes

Number of addresses per block = Block size/ space occupied by each address = 4096/4 = 1024 = 2^{10}

Maximum size of file = (12 direct pointers $\times$4KB) + (1 single Indirect pointer $\times$4KB) + (1 Double Indirect pointer $\times$4KB)

$=(12 \times 2^{12}) + (2^{10} \times 2^{12}) + (2^{10} \times 2^{10} \times 2^{12})$

$= 2^{15} + 2^{22} + 2^{32} = 2^{32}$ bytes $= 2^{2} \times 2^{30} = 4GB$

$(12 \times 2^{12})$is written as (12 $\times$4096) = 49152 = In binary it is equal to 15bits

G3. A **FAT** (file allocation table) based file system is being used and the **total overhead of each entry in the FAT is 4 bytes in size.** Given a **100 x 10^6 bytes disk** on which the file system is stored and data **block size is 10^3 bytes**, the **maximum size of a file** that can be stored on this disk in units of 10^6 bytes is _____. (GATE 2014)

Total disk size = **100 x 10^6 bytes**
Si ze of disk block = **10^3 bytes**
Overhead for each FAT entry = 4 bytes
Number of entries in FAT = Total disk size/Size of disk block
= **100 x 10^6 bytes/ 10^3 bytes = 10^5 entries**
Total space occupied by FAT = Number of entries in FAT x Overhead for each FAT entry
= **10^5 x 4 bytes = 0.4x 10^6 bytes**

Maximum file size that can be stored on the disk= Total disk space- Total space occupied by FAT
= (100 x 10^6 - 0.4x 10^6) bytes = 99.6 x 10^6 bytes

G4. A file system with **300 GB disk** uses a file descriptor with **8 direct** block addresses, **1 indirect** block address and **1 doubly indirect** block address. The size of each **disk block is 128 Bytes** and the size of each **disk block address is 8 Bytes**. The **maximum possible file size** in this file system is (GATE 2012)

(A) 3 Kbytes **(B) 35 Kbytes** **(C)** 280 Bytes

(D) Dependent on the size of the disk

Size of the disk block = 128 bytes

Disk block address = 8 bytes

Number of addresses per block = Block size/ space occupied by each address = 128/8 = 16

Maximum size of file = [(8 direct pointers × 128) + (1 single Indirect pointer × 128) + (1 Double Indirect pointer × 128)] bytes

= [(8 × 128) +(16 × 128) + (16 × 16 × 128)] bytes

= [1024 + 2048+ 32768] bytes

= 35840bytes = 35KB

EX1 :Consider a file currently consisting of 100 blocks. Assume that the file-control block (and the index block, in the case of indexed allocation) is already in memory. **Calculate how many disk I/O operations are required for contiguous, linked, and indexed (single-level) allocation** strategies, if, for one block, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow at the beginning but there is room to grow at the end. Also assume that the block information to be added is stored in memory.

a. The block is added at the beginning.

b. The block is added in the middle.

c. The block is added at the end.

d. The block is removed from the beginning.

e. The block is removed from the middle.

f. The block is removed from the end.

a. The block is added at the beginning.

contiguous allocation : Each block must be shifted over to the next block. This involved one read and one write per block. Then the new block must be written. So, **201 I/O operations**.

linked allocation: Just write the new block making it point to the next block. Update the *first block* pointer in memory. So, only **1 I/O operation.**

indexed allocation: Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

The results are:

	Contiguous	Linked	Indexed
a.	201	1	1
b.	101	52	1
c.	1	3	1
d.	198	1	0
e.	98	52	0
f.	0	100	0

b. The block is added to the middle.

contiguous allocation: 50 blocks (the second half) must be shifted over one block and the new block must be written. As before, the shift takes one read operation and one write operation So, **101 I/O operations**.

linked allocation: Read 50 blocks to find the middle. Then , write new block somewhere with the *next block* pointing to the block after the 50th block. Then , write the 50th block to point to this new block. So, **52 I/O operations**.

indexed allocation: Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

c. The block is added at the end

contiguous allocation: Just write the new block. So, only **1 I/O operation.**

linked allocation: [Assuming that in order to modify the a block's *next block* pointer, first read the whole block in, then write the whole block out just with that pointer changed]. First read the last block as given by the *last block* pointer, then write new block, then write the last block back modifying its *next block* pointer to point to new block, then update *last block* pointer in memory. So, **3 I/O operations.**

indexed allocation1. Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

d. The block is removed from the beginning.

contiguous allocation: Just point the *first block* pointer to the second block. So, there is **‘0’ I/O operation.**

linked allocation: Read in the first block to get the second block's pointer and then set the *first block* pointer to point to the second block. So, only **1 I/O operation.**

indexed allocation: Just remove the block from the index in memory. So, there is **‘0’ I/O operation**

e. The block is removed from the middle

contiguous allocation: Assuming that removing the 51st block, then move 49 blocks. The 49 blocks takes a read and a write operation for each block. So, there is **98 I/O operations.**

linked allocation: It takes 51 reads to find the pointer to the 52nd block, then update the 50th block's *next block* pointer with this value. So, there is **52 I/O operations.**

indexed allocation: Just remove the block from the index in memory. So, there is **‘0’ I/O operation.**

f. The block is removed from the end.

contiguous allocation: Just update the *length* information stored in memory. So, there is **'0' I/O operation.**

linked allocation: update the *last block* pointer with the second to last block and the only way to get the second to last block is to read the preceding 99 blocks. Then probably mark the 99th block (the new last block) as having a null pointer and this would give the 100th operation. Save the new last block in *last block* pointer in memory. So, there is **100 I/O operations.**

indexed allocation: Just remove the block from the index in memory. So, there is **'0' I/O operation.**

The Linux System - History

- Linux is a modern, free operating system based on UNIX standards
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility, released as open source
- Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet
- **Linux system** has many, varying **Linux distributions** including the kernel, applications, and management tools

The Linux Kernel

- Kernels with odd version numbers are **development kernels**, those with even numbers are **production kernels**

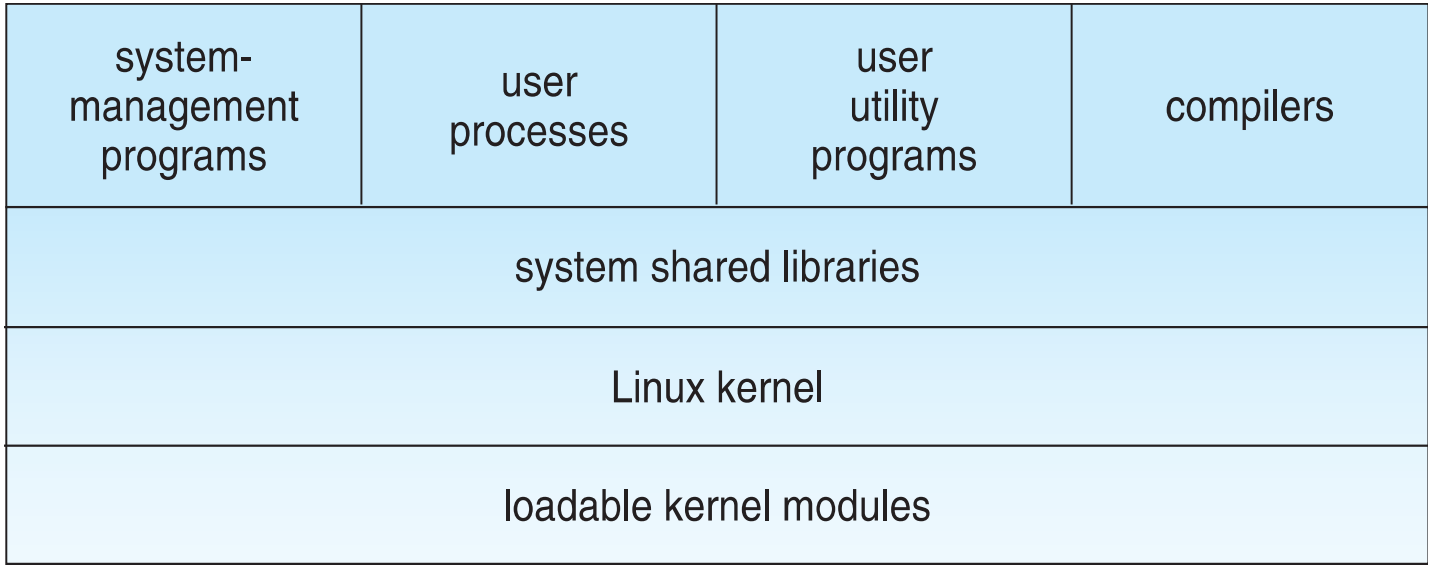
Linux 2.0

- **New features included:**
  - Improved memory-management code
  - Improved TCP/IP performance
  - Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand
  - Standardized configuration interface

Linux Licensing

Anyone using Linux, or creating their own derivative of Linux, may not make the derived product proprietary. Can sell distributions, but must offer the source code too

Components of a Linux System



Components of a Linux System

- Like most UNIX implementations, Linux is composed of three main bodies of code; the most important distinction between the kernel and all other components.
- The **kernel** is responsible for maintaining the important abstractions of the operating system
- The **system libraries** define a standard set of functions through which applications interact with the kernel
- The **system utilities** perform individual specialized management tasks

Kernel Modules

- Four components to Linux module support:
  - **module-management system**
  - **module loader and unloader**
  - **driver-registration system**
  - **conflict-resolution mechanism**

Module Management

- Supports loading modules into memory and letting them talk to the rest of the kernel

Driver Registration

- Allows modules to tell the rest of the kernel that a new driver has become available

Conflict Resolution

- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

Process Management

- The fork() system call creates a new process
- A new program is run after a call to exec()

Under Linux, process properties fall into three groups: the process's identity, environment, and context

Process Identity

- **Process ID (PID)** - The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process
- **Credentials** - Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files

- **Personality** - Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls
  - Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX
- **Namespace** – Specific view of file system hierarchy
  - Most processes share common namespace and operate on a shared file-system hierarchy
  - But each can have unique file-system hierarchy with its own root directory and set of mounted file systems

Process Environment

- The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
  - The **argument vector** lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself.
  - The **environment vector** is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.

Process Context

- The (constantly changing) state of a running program at any point in time
- The **scheduling context** is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process
- The kernel maintains **accounting** information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far
- The **file table** is an array of pointers to kernel file structures

Processes and Threads

- A thread is simply a new process that happens to share the same address space as its parent
  - Both are called ***tasks*** by Linux
- A distinction is only made when a new thread is created by the `clone()` system call
  - `fork()` creates a new task with its own entirely new task context
  - `clone()` creates a new task with its own identity, but that is allowed to share the data structures of its parent

flag	meaning
`CLONE_FS`	File-system information is shared.
`CLONE_VM`	The same memory space is shared.
`CLONE_SIGHAND`	Signal handlers are shared.
`CLONE_FILES`	The set of open files is shared.

Scheduling

- As of 2.5, new scheduling algorithm – preemptive, priority-based, known as *O*(1)
  - Real-time range
  - nice value
  - Had challenges with interactive performance
- 2.6 introduced **Completely Fair Scheduler (CFS)**

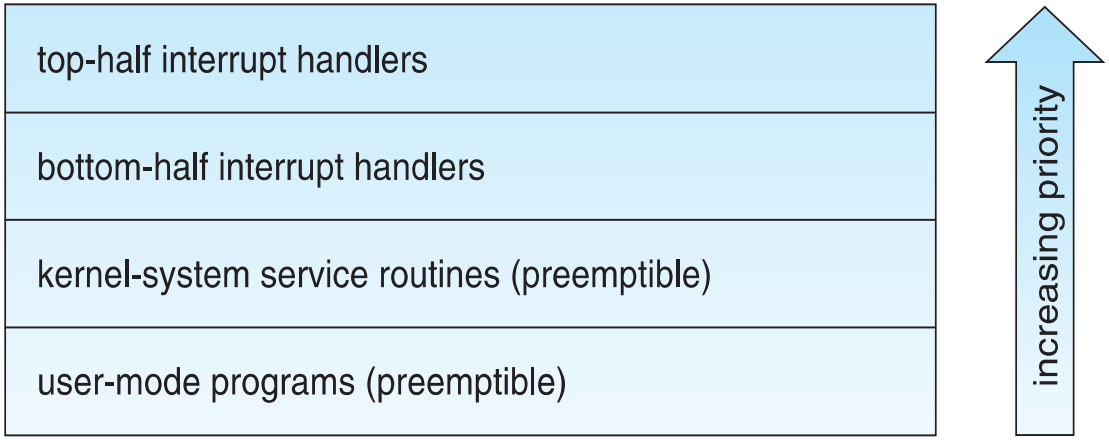
CFS

- CFS calculates how long a process should run as a function of total number of tasks
- *N* runnable tasks means each gets 1/*N* of processor's time
- Then weights each task with its nice value
  - Smaller nice value -> higher weight (higher priority)

Kernel Synchronization

- A request for kernel-mode execution can occur in two ways:
  - A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs
  - A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt

Interrupt Protection Levels



- Each level may be interrupted by code running at a higher level, but will never be interrupted by code running at the same or a lower level
- User processes can always be preempted by another process when a time-sharing scheduling interrupt occurs

Symmetric Multiprocessing

- Linux 2.0 was the first Linux kernel to support **SMP** hardware; separate processes or threads can execute in parallel on separate processors
- Version 3.0 adds even more fine-grained locking, processor affinity, and load-balancing

Memory Management

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- Splits memory into four different **zones** due to hardware characteristics

Memory Management

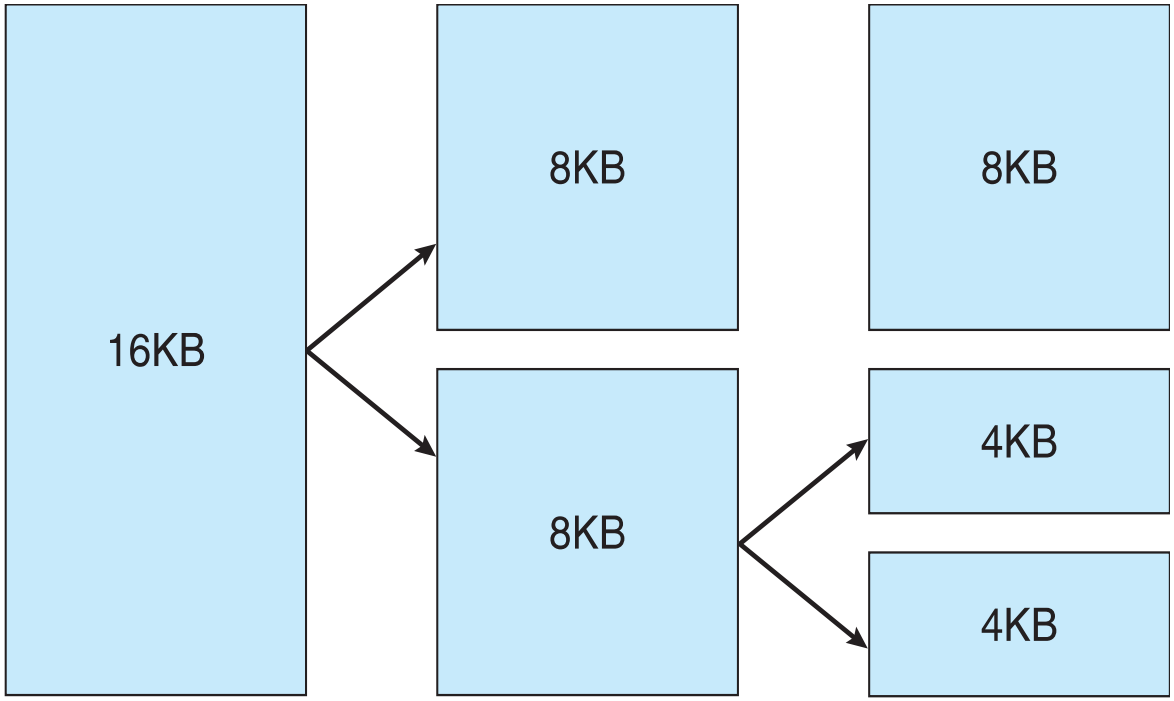
- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes
- Splits memory into four different **zones** due to hardware characteristics
  - Architecture specific, for example on x86:

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

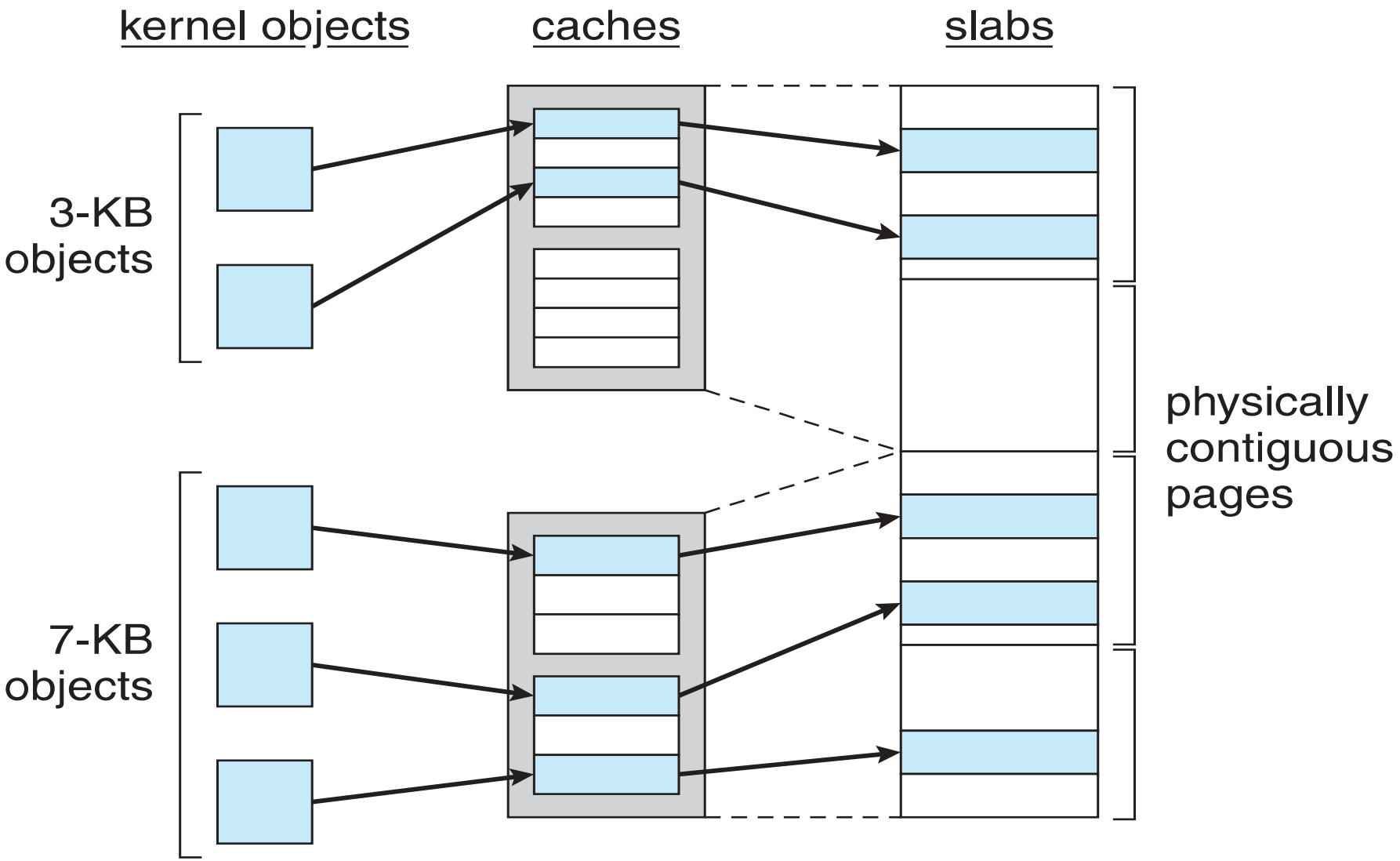
Managing Physical Memory

- The allocator uses a **buddy-heap algorithm** to keep track of available physical pages
  - Each allocatable memory region is paired with an adjacent partner
  - Whenever two allocated partner regions are both freed up they are combined to form a larger region
  - If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request

Splitting of Memory in a Buddy Heap



Slab Allocator in Linux



Virtual Memory

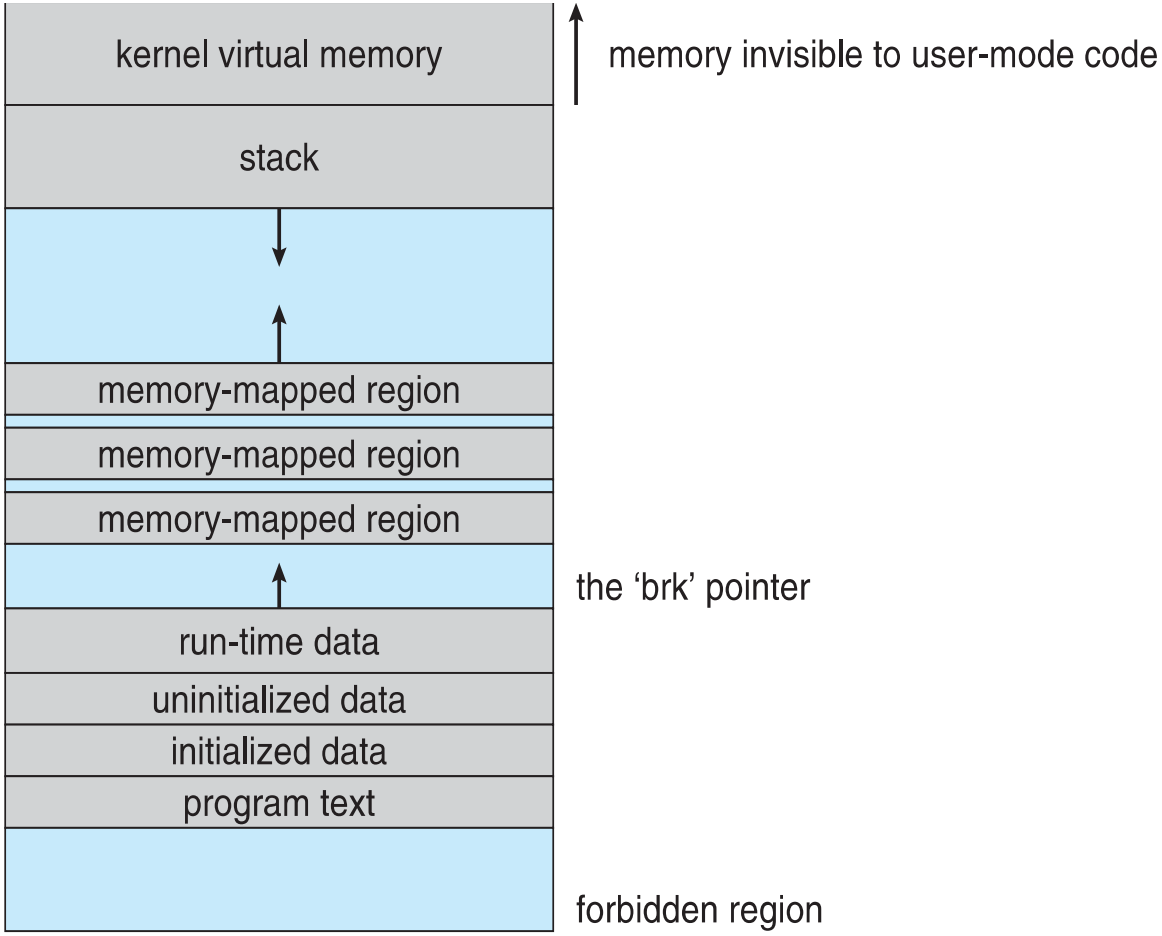
- The VM manager maintains two separate views of a process's address space:
  - A logical view describing instructions concerning the layout of the address space
  - A physical view of each address space which is stored in the hardware page tables for the process

- Virtual memory regions are characterized by:
  - The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (**demand-zero memory**)
  - The region's reaction to writes (page sharing or copy-on-write

Swapping and Paging

- The VM paging system can be divided into two sections:
  - The **pageout-policy** algorithm decides which pages to write out to disk, and when
  - The **paging mechanism** actually carries out the transfer, and pages data back into physical memory as needed

Memory Layout for ELF Programs



Static and Dynamic Linking

- A program whose necessary library functions are embedded directly in the program's executable binary file is ***statically*** linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions
- *Dynamic* linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once
- Shared libraries compiled to be **position-independent code (PIC)** . so can be loaded anywhere

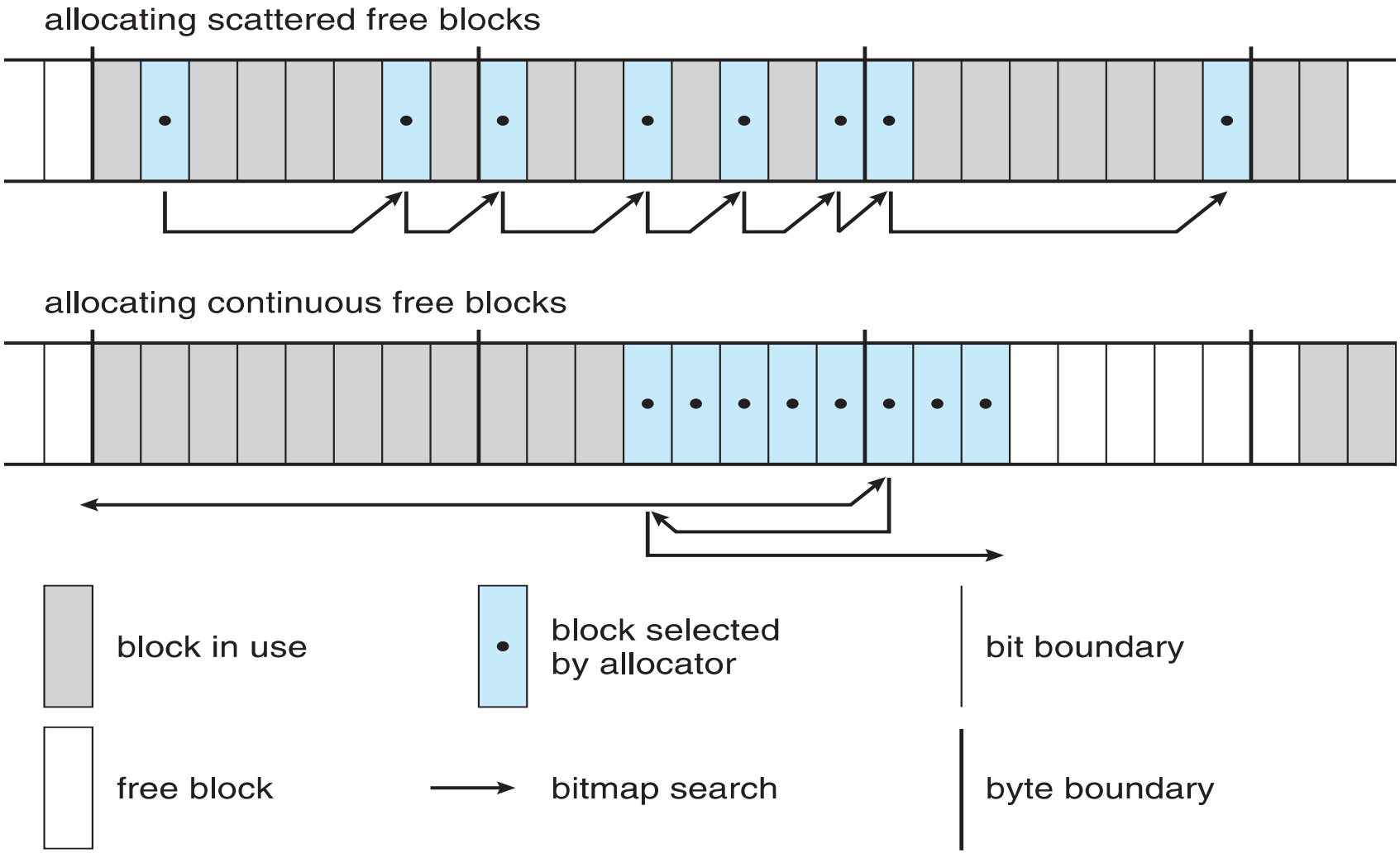
File Systems

- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and is composed of four components:
  - The **inode object** structure represent an individual file
  - The **file object** represents an open file
  - The **superblock object** represents an entire file system
  - A **dentry object** represents an individual directory entry

The Linux ext3 File System

- **ext3** is standard on disk file system for Linux
  - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
  - Supersedes older **extfs**, **ext2** file systems
  - Work underway on ext4 adding features like extents
  - Of course, many other file system choices with Linux distros
- **The main differences between ext2fs and FFS concern their disk allocation policies**
  - In FFS, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
  - ext3 does not use fragments; it performs its allocations in smaller units

Ext2fs Block-Allocation Policies



Journaling

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions**
  - Once in log file, considered committed
  - Over time, log file transactions replayed over file system to put changes in place

The Linux Proc File System

- The **proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains

Input and Output

- Linux splits all devices into three classes:
  - **Block devices** allow random access to completely independent, fixed size blocks of data
  - **Character devices** include most other devices; they don't need to support the functionality of regular files
  - **Network devices** are interfaced via the kernel's networking subsystem

Block Devices

- The block buffer cache serves two main purposes:
  - it acts as a pool of buffers for active I/O
  - it serves as a cache for completed I/O
- The **request manager** manages the reading and writing of buffer contents to and from a block device driver

Character Devices

- A character device driver must register a set of functions which implement the driver's various file I/O operations
- The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device
- The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface
- **Line discipline** is an interpreter for the information from the terminal device

Inter process Communication

- Like UNIX, Linux informs processes that an event has occurred via **signals**
- The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait_queue** structures
- Also implements System V Unix semaphores
  - Process can wait for a signal or a semaphore
  - Semaphores scale better
  - Operations on multiple semaphores can be atomic

Passing Data Between Processes

- The **pipe** mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read a the other
- Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
- To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism

Network Structure

- Networking is a key area of functionality for Linux
  - It supports the standard Internet protocols for UNIX to UNIX communications
- Most important set of protocols in the Linux networking system is the internet protocol suite
  - It implements routing between different hosts anywhere on the network
  - On top of the routing protocol are built the UDP, TCP and ICMP protocols
- Packets also pass to **firewall management** for filtering based on **firewall chains** of rules

Security

- The **pluggable authentication modules** ***(PAM)*** system is available under Linux
- PAM is based on a shared library that can be used by any system component that needs to authenticate users
- Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers **(uid** and **gid)**
- Linux augments the standard UNIX **setuid** mechanism in two ways:
- Linux provides another mechanism that allows a client to selectively pass access to a single file to some server process without granting it any other privileges