Operating System

Chapter 11. I/O Management and Disk Scheduling

Lynn Choi

School of Electrical Engineering

Categories of I/O Devices

- **I/O devices can be grouped into 3 categories**
  - Human readable devices
    - Suitable for communicating with the computer user
    - Printers, terminals, video display, keyboard, mouse
  - Machine readable devices
    - Suitable for communicating with electronic equipment
    - Disk drives, USB devices, sensors, controllers
  - Communication devices
    - Suitable for communicating with remote devices
    - Modems, digital line drivers

Data Rates

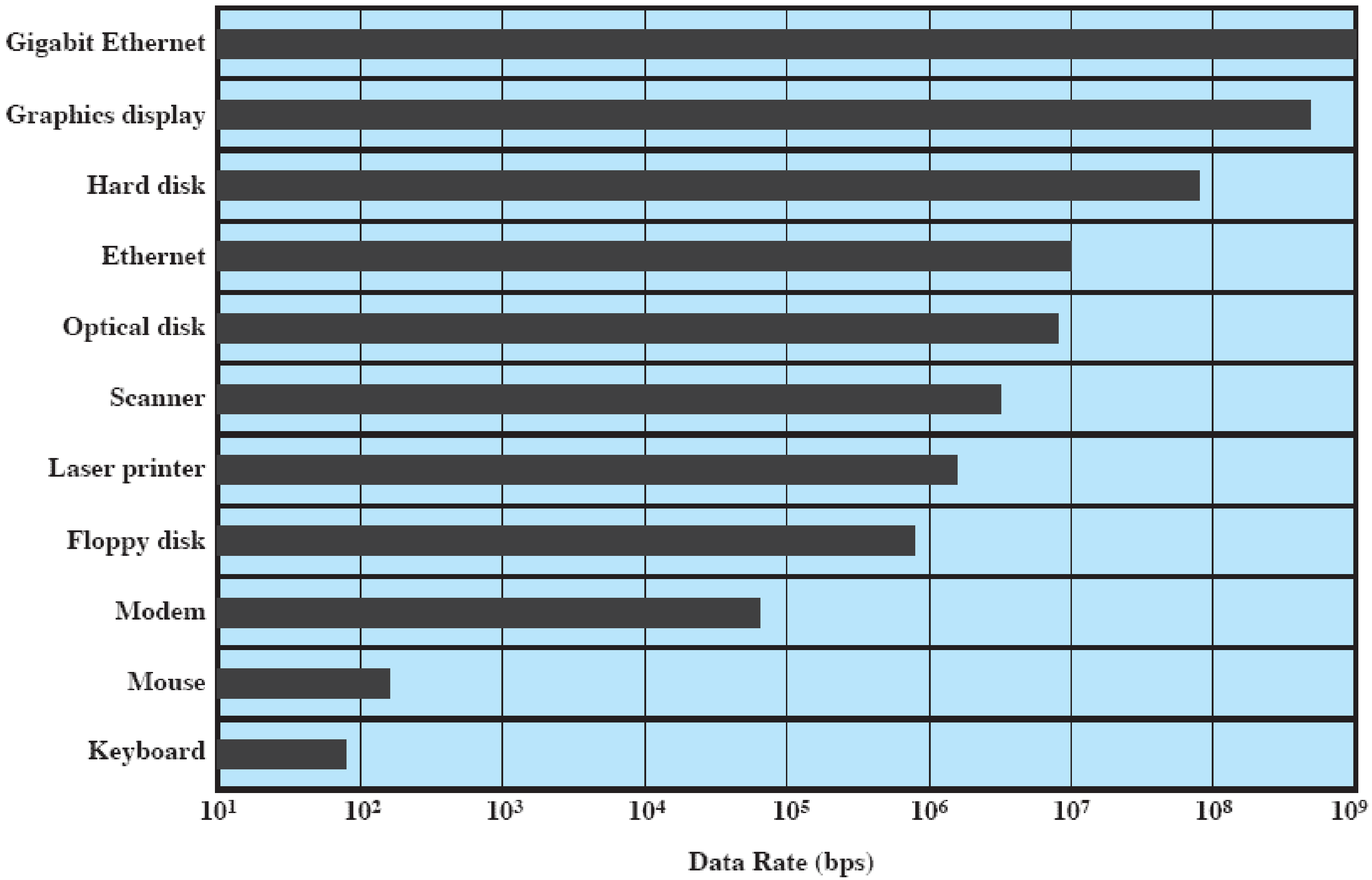


Figure 11.1 Typical I/O Device Data Rates

Source: Pearson

Organization of I/O Function

- **Three techniques for performing I/O are**
- **Programmed I/O**
  - The processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding
- **Interrupt-driven I/O**
  - The processor issues an I/O command on behalf of a process
    - If non-blocking – processor continues to execute instructions from the process that issued the I/O command
    - If blocking – the next instruction the processor executes is from the OS, which will put the current process in a blocked state and schedule another process
- **Direct Memory Access (DMA)**
  - The processor sends a request for a block transfer to the DMA module, which then controls the exchange of data between main memory and an I/O module. After the transfer, the DMA module interrupts the processor.

Techniques for Performing I/O

Table 11.1 I/O Techniques

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

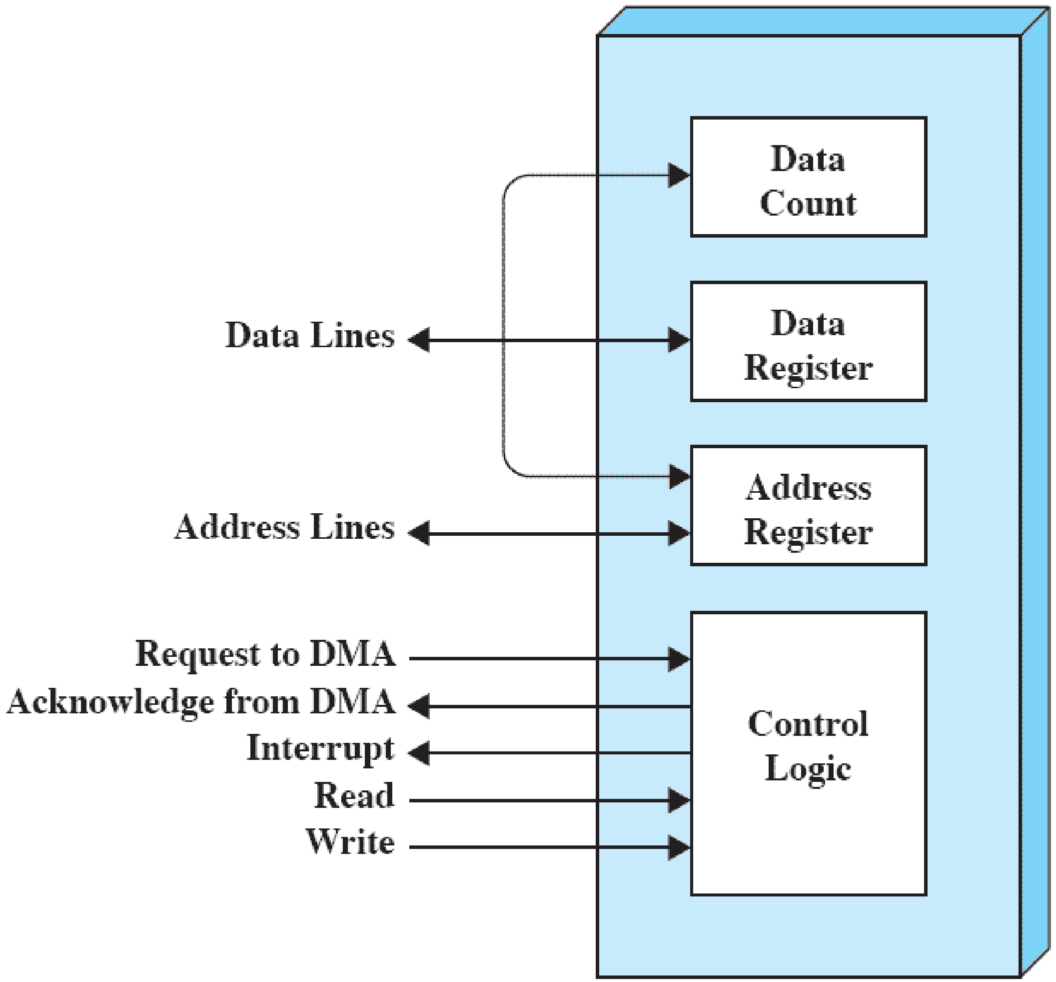
Source: Pearson

Evolution of I/O Function

- *Processor directly controls a peripheral device*
- *Programmed I/O without interrupt*
  - An I/O controller or I/O module is added
- *Programmed I/O with interrupt*
  - Same configuration as step 2, but now interrupts are employed
- *DMA*
  - The I/O module is given direct control of memory via DMA
- *I/O channel*
  - The I/O module is enhanced to become a separate processor, with a specialized instruction set tailored for I/O
- *I/O processor*
  - The I/O module has a local memory of its own and is, in fact, a computer in its own right

DMA Block Diagram

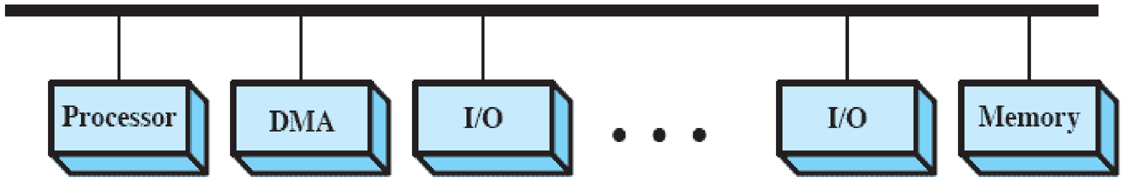
- ❑ *Processor issues a command to DMA module with the following information*
  - Read or Write
  - The address of IO device
  - The starting address of memory
  - The number of words to transfer
- ❑ *DMA module transfers the entire block and after completion, it interrupts the processor*



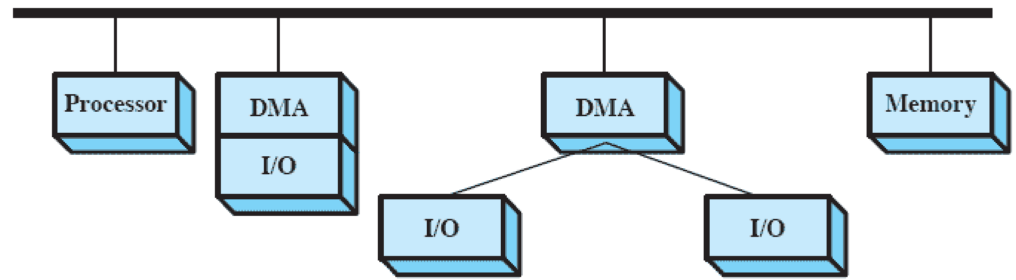
Source: Pearson

Figure 11.2 Typical DMA Block Diagram

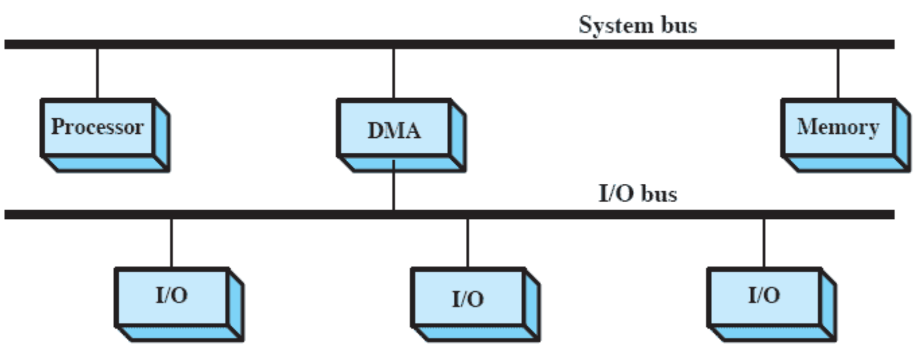
DMA Alternative Configurations



(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-I/O



(c) I/O bus

Source: Pearson

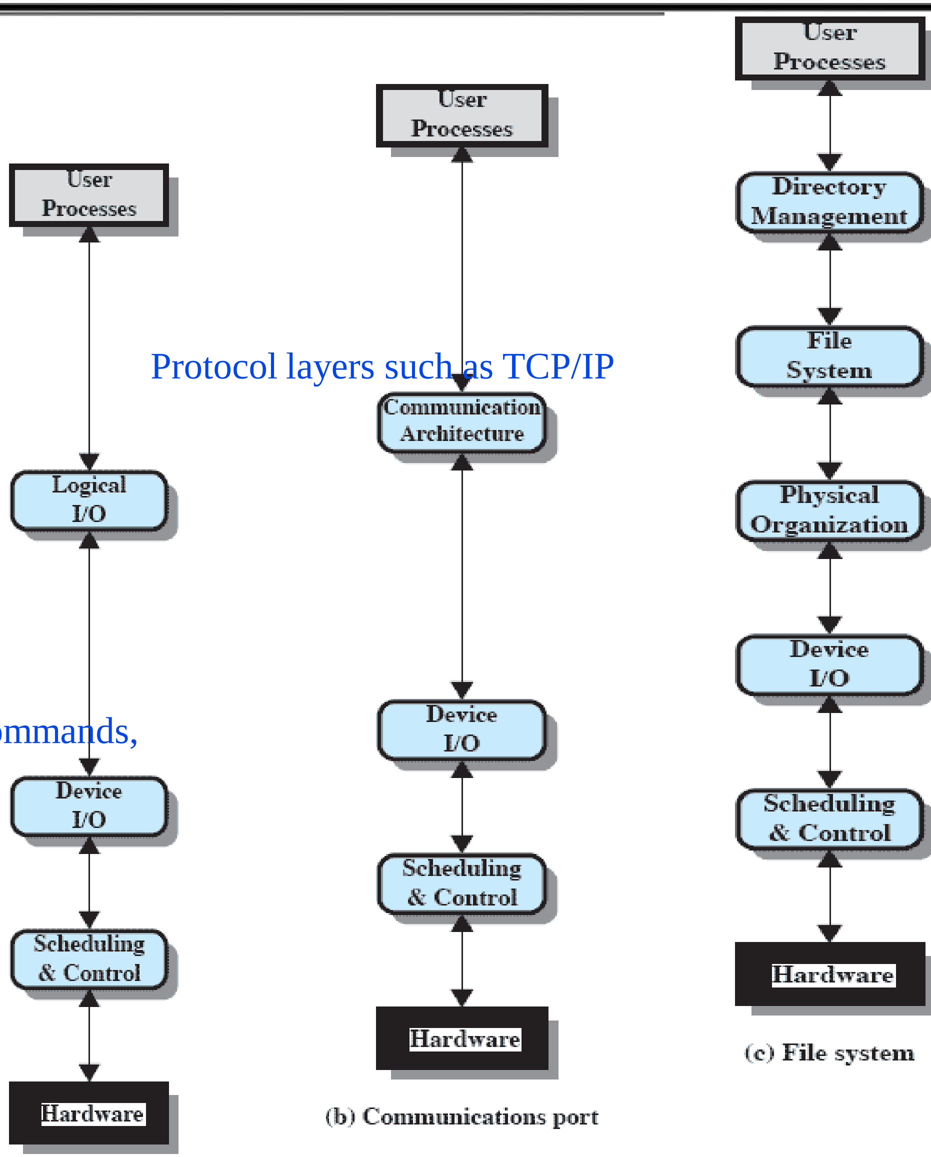
Design Objectives

- **Efficiency**
  - Major effort in I/O design
  - Important because I/O operations often form a bottleneck
  - Most I/O devices are extremely slow compared with main memory and the processor
  - The area that has received the most attention is disk I/O
- **Generality**
  - Desirable to handle all devices in a uniform manner
  - The way processes view I/O devices and the way the operating system manages I/O devices and operations
  - Hide the details of device I/O so that user processes and upper levels of OS see devices in terms of general functions such as read, write, open, and close
  - Diversity of devices makes it difficult to achieve true generality

Hierarchical Design

- **Hierarchical nature of modern operating systems**
  - Operating system functions should be separated according to their complexity, timescale, and their level of abstraction
  - Leads to an OS organization into a series of layers
  - Each layer performs a related subset of the functions and relies on the next lower layer to perform more primitive functions and to conceal the details of those functions. It provides services to the next higher layer.
  - Layers should be defined so that changes in one layer do not require changes in other layers

A Model of I/O Organization



(a) Local peripheral device

(b) Communications port

(c) File system

open, close, read, write

I/O instructions, channel commands, buffering techniques

interrupts, scheduling, and queuing

Protocol layers such as TCP/IP

symbolic file names are converted to identifiers add, delete

open, close, read, write

logical reference to files are converted to physical addresses (track, sector)

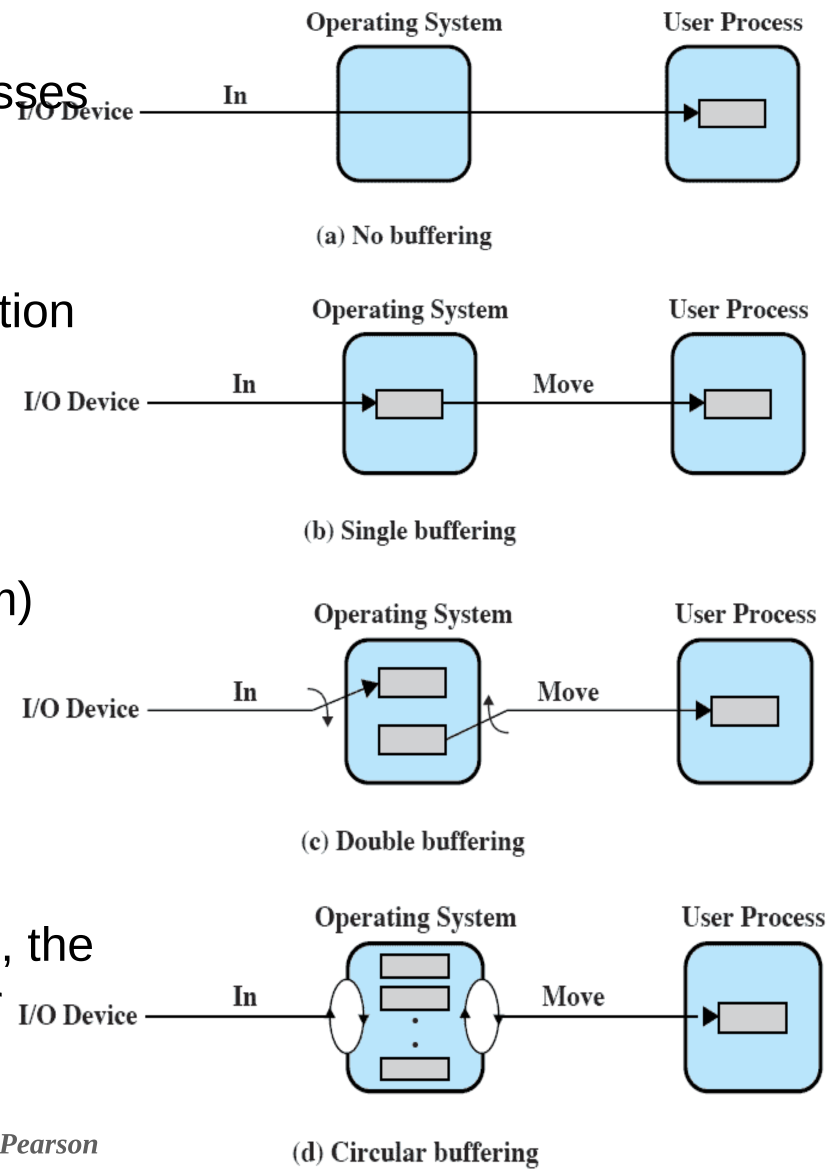
Source: Pearson

Buffering

- **Perform data transfers in advance of requests**
  - For both inputs and outputs
  - Can reduce time waiting for I/O to complete
  - Also, avoid I/O interferences with OS swapping decisions
- **Block-oriented device**
  - Stores information in blocks that are usually of fixed size
  - Transfers are made one block at a time
  - Possible to reference data by its block number
  - Disks and USB devices are examples
- **Stream-oriented device**
  - Transfers data as a stream of bytes
  - No block structure
  - Terminals, printers, keyboards, mouse, communications ports, and most other devices that are not secondary storage are examples

I/O Buffering Schemes

- **No buffering**
  - Without a buffer, the OS directly accesses the device when it needs
- **Single buffering**
  - OS assigns a buffer in the system portion of main memory
- **Double buffering**
  - Use two system buffers
  - A process can transfer data to (or from) one buffer while the operating system empties (or fills) the other buffer
  - Also known as buffer swapping
- **Circular buffering**
  - When more than two buffers are used, the collection of buffers is a circular buffer
  - Each individual buffer is one unit in a circular buffer



(a) No buffering

(b) Single buffering

(c) Double buffering

(d) Circular buffering

Source: Pearson

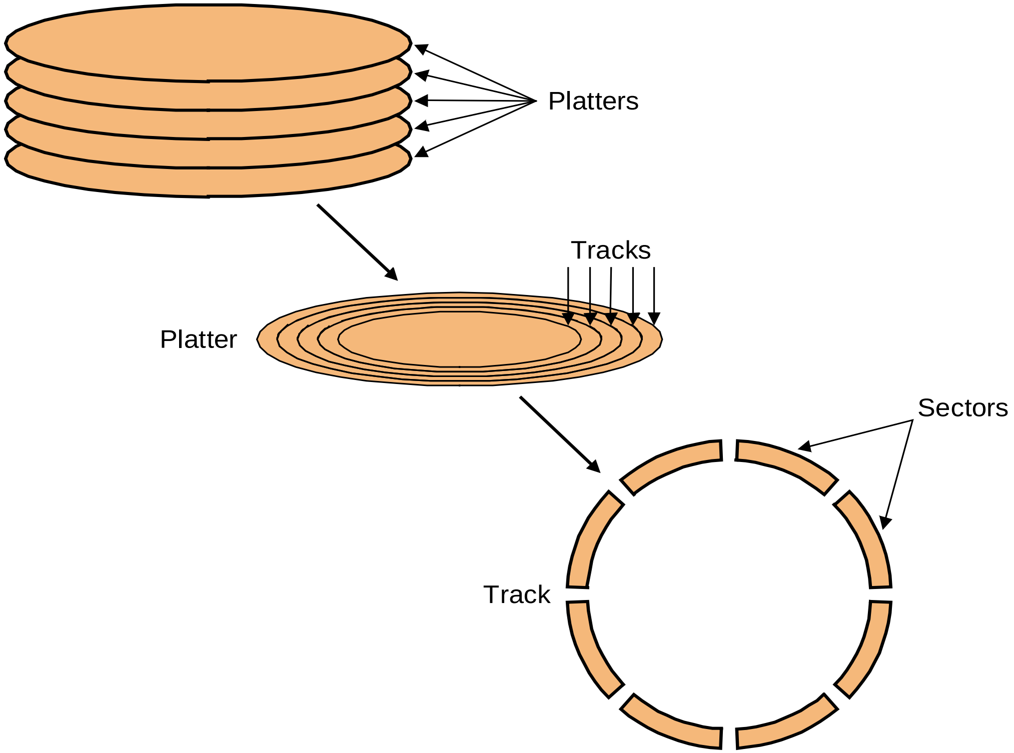
Single Buffering

- **For block-oriented devices**
  - Input transfers are made to the system buffer
  - When the transfer is complete, the process moves the block into user space and immediately requests another block
  - Can speed up I/O since data are usually accessed sequentially
- **For stream-oriented devices**
  - Line-at-a-time operation
    - Used for dumb terminals or line printers
    - User input is one line at a time with a carriage return
    - Output to the terminal is similarly one line at a time
  - Byte-at-a-time operation
    - Used on forms-mode terminals, sensors and controllers
    - When each keystroke is significant

Magnetic Disk

- **A magnetic disk consists of a collection of platters, each of which has two recordable surfaces.**
  - The stack of flatters rotate at 5400 RPM to 15000 RPM
  - The diameter of this aluminum platter is from 3 ~ 12 cm

- **Read/write heads**
  - To read or write, the read/write heads must be moved so that they are over the right track
  - Disk heads for each surface are connected together and move in conjunction



Magnetic Disk

- **Cylinder: a set of tracks at a given radial position**
  - All the tracks under the heads at a given point on all surfaces
- **Track: each surface is divided into concentric circles**
  - 10,000 to 50,000 tracks per surface
  - ZBR (Zone Bit Recording)
    - The number of sectors per track increases in outer zones
- **Sector - track is divided into fixed size sectors (100 ~ 500 sectors/track)**
  - Preamble - allows head to be synchronized before r/w
  - Data - 512B - 4KB
  - Error correcting code (ECC)
    - Hamming code or Reed-Solomon code
  - Inter-sector gap
  - Formatted capacity does not count preamble/ecc/gap

Magnetic Disk

- **Performance**
  - Seek time
    - To move the read/write head to the desired track
    - 3 ~ 14ms, consecutive tracks less than 1 ms
  - Rotational latency
    - To locate the desired sector under the read/write head
    - On average, it takes a half of a single rotation time
    - 5400 ~ 16200 rpm (90 ~ 270 rotations/s), 2 ~ 6ms avg.
  - Transfer time
    - Depends on the rotation speed and data density
    - 30 ~ 40MB/s, 512B sector takes 12 ~ 16us
- **Disk Controller**
  - Accept commands from CPU
    - read, write, format (write preambles), control the arm motion, detect/correct errors, convert byte to a serial bit pattern, buffering/caching,

Disk Access Time

- **Disk access time =**
  - Seek time + rotational latency + transfer time + controller overhead
- **For example,**
  - HDD with the following characteristics
    - 10,000 RPM
    - Average seek time 6ms
    - Transfer rate 50MB/s
    - Controller overhead 0.2ms
    - No disk idle time
  - Average acceess time for a 512B sector =
    - 6ms + 0.5 rotation / 10000RPM + 0.5KB/50MB/s + 0.2ms = 6 + 3 + 0.01 + 0.2 = 9.2ms
    - Usually seek time is only 25% ~ 33% of the advertised number due to locality of disk references
    - Most disk controllers have a built-in cache and transfer rates from the cache are typically much higher and up to 320MB/s

Timing Comparison

- **Consider a disk with**
  - Seek time of 4ms
  - Rotation speed of 7500 rpm
  - 512 byte sectors with 500 sectors per track
- **Read a file consisting of 2500 sectors (1.28MB)**
- **Sequential organization**
  - The file occupies all the sectors of 5 adjacent tracks.
  - Seek time = 4ms
  - Rotational latency = 4ms
  - Read 500 sectors = 8ms
  - Total time = 16 + 4 * 12 = 64ms
- **Random access**
  - Seek time = rotational latency = 4ms
  - Read 1 sector = 0.016ms
  - Total time = 2500 * 8.016 = 20.04s
- **Which sectors are read from the disk has a tremendous impact on I/O performance!**

Disk Scheduling Algorithms

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of *N* records at a time	Service guarantee
FSCAN	N-step-SCAN with *N* = queue size at beginning of SCAN cycle	Load sensitive

Source: Pearson

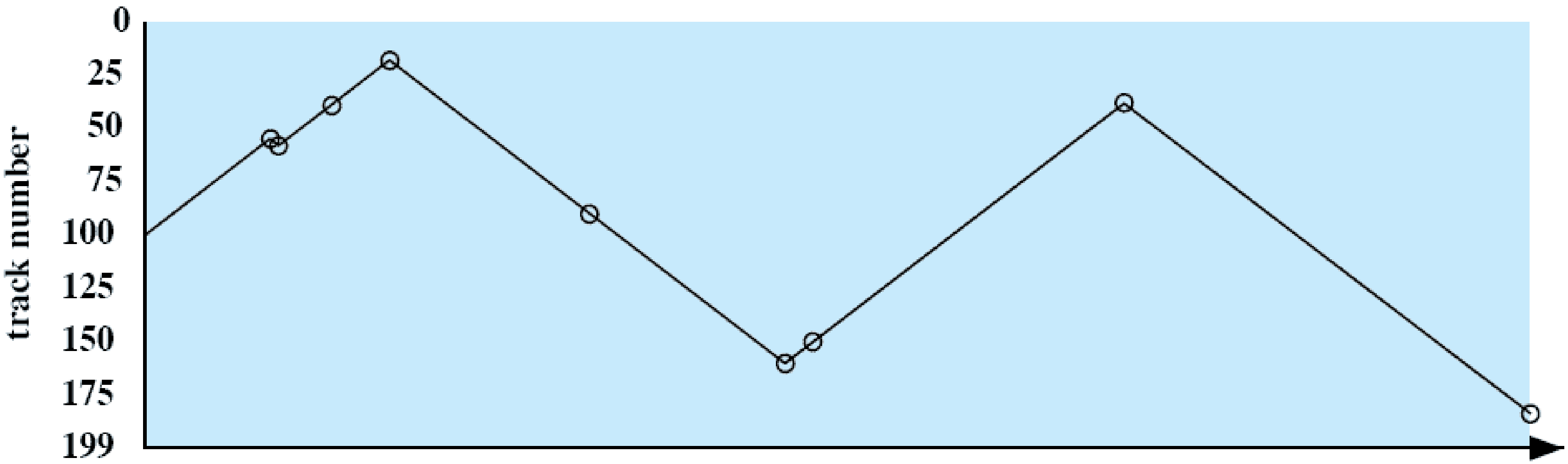
Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Source: Pearson

FIFO

- Processes requests from the queue in sequential order
- Fair to all processes
- Approximate random scheduling in performance if there are many processes competing for the disk



(a) FIFO

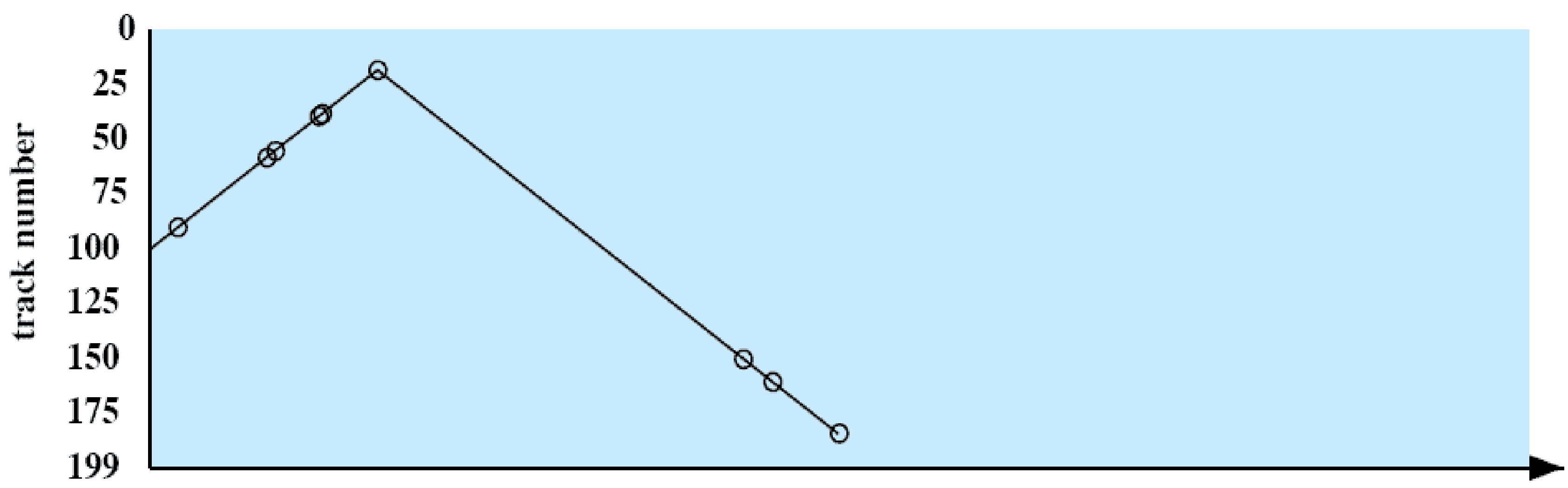
Source: Pearson

Priority (PRI)

- **The control of the scheduling is outside the control of disk management software**
- **Goal is not to optimize disk utilization but to meet other objectives**
- **Often short batch jobs and interactive jobs are given higher priority**
  - Provides good interactive response time
  - Longer jobs may have to wait an excessively long time

Shortest Service Time First (SSTF)

- **Select the disk I/O request that requires the least movement of the disk arm from its current position**
- **Always choose the minimum seek time**
  - Does not guarantee that the average seek time to be minimum

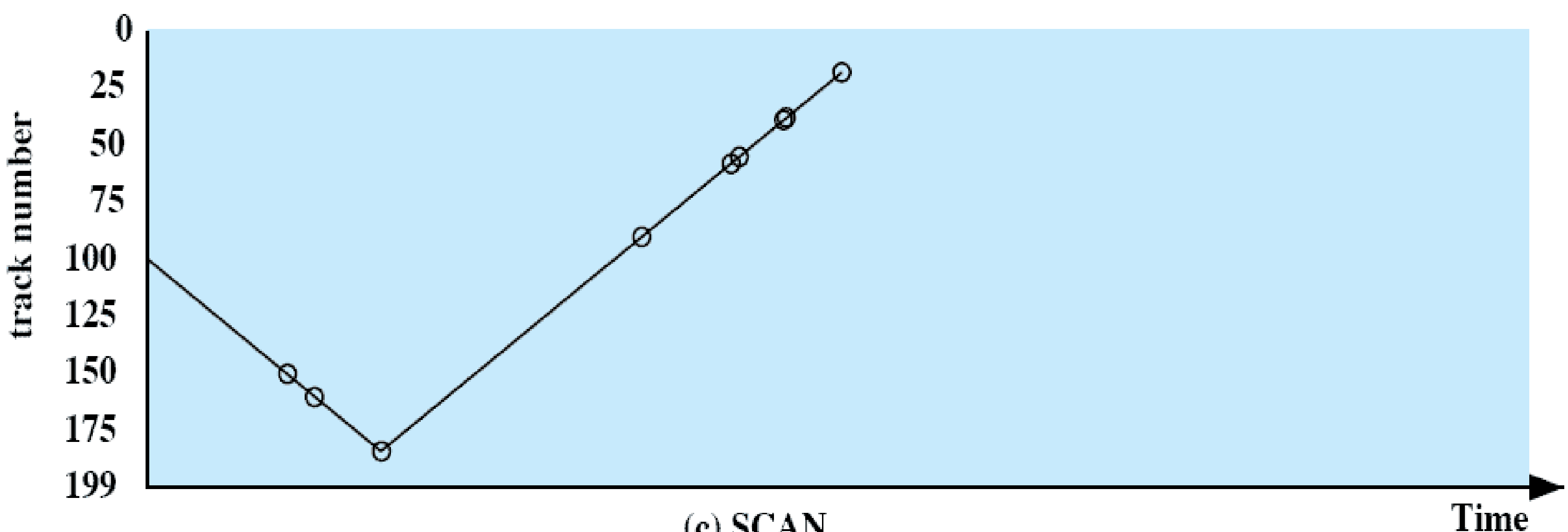


(b) SSTF

Source: Pearson

SCAN

- Also known as the elevator algorithm
- Arm moves in one direction only
  - Satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed
- Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks and favors the latest arriving jobs

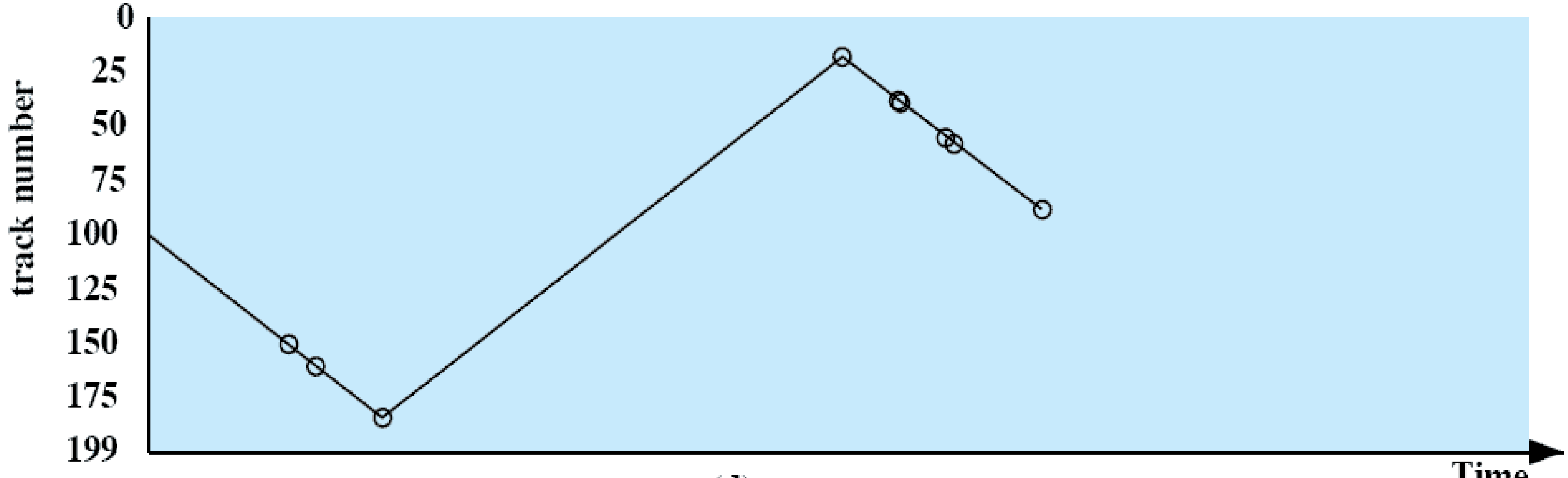


(c) SCAN

Source: Pearson

C-SCAN (Circular SCAN)

- Restricts scanning to one direction only
- When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan



(d) C-SCAN

Source: Pearson

N-Step-SCAN and FSCAN

- **N-Step-Scan**
  - Segment the disk request queue into subqueues of length *N*
  - Subqueues are processed one at a time, using SCAN
  - For a large value of N, the performance of N-Step-Scan approaches that of SCAN. For a value of N = 1, it is the same as FIFO.
- **FSCAN**
  - Uses two subqueues
  - When a scan begins, all of the requests are in one of the queues, with the other empty
  - During scan, all new requests are put into the other queue
  - Service of new requests is deferred until all of the old requests have been processed

RAID

❑ Motivation

- Disk seek time has continued to improve slowly over time
- 970 (50~100ms), 1990 (10ms), 2010 (3ms)

❑ Ideas

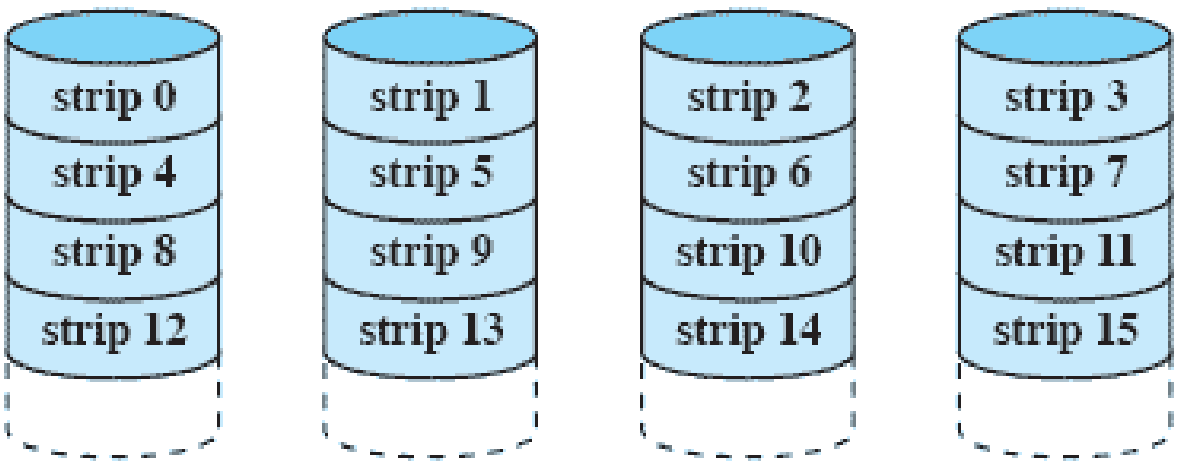
- Performance - parallel processing
- Reliability

❑ RAID (Redundant Array of Independent Disks)

- Consists of seven levels, zero through six
- These levels denote different design architectures that share 3 characteristics
  - RAID is a set of physical disk drives viewed by the operating system as a single logical drive
  - Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure
  - Data are distributed across the physical drives of an array in a scheme known as striping

RAID Level 0

- ***Stripping*** **- distribute data over multiple disks**
  - When a transferred block consists of 8 sectors, 2 sectors (*strip*) are distributed to different disk drive
  - If a block size is bigger than # drives * strip size, multiple requests are needed
  - If a single request consists of multiple logically contiguous strips, then up to n strips for that request can be handled in parallel
- **No redundancy and no error detection/correction but widely used**

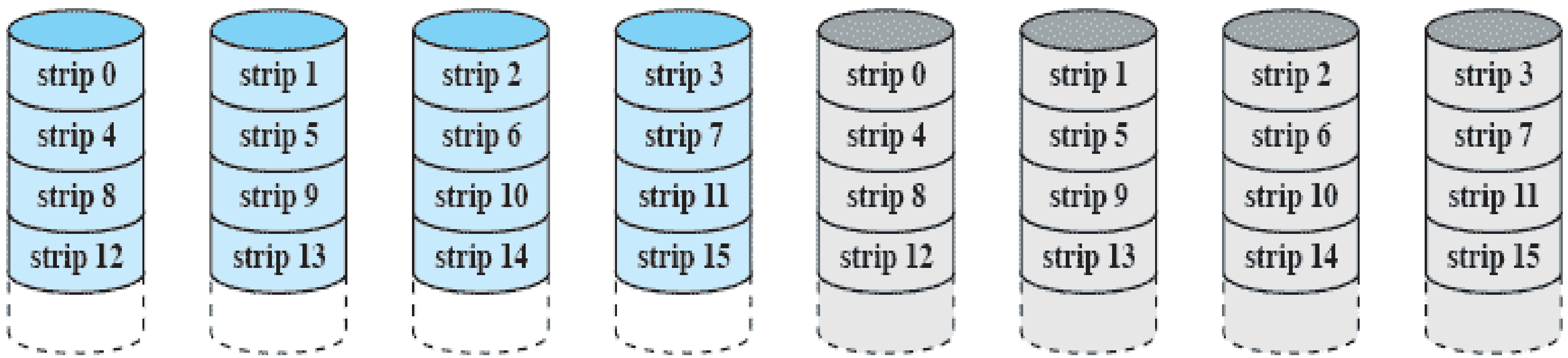


(a) RAID 0 (non-redundant)

Source: Pearson

RAID Level 1 (Mirroring)

- **Redundancy is achieved by duplicating all the data**
  - Every disk in the array has a mirror disk
    - When a drive fails the data may still be accessed from the second drive
- **Advantage**
  - A read request can be served by either of two disks.
  - There is no “write penalty”.
    - Write can be done in parallel. On a write, RAID levels 2-6 must compute and update parity bits as well as updating the actual strip.
- **Principal disadvantage is the cost**

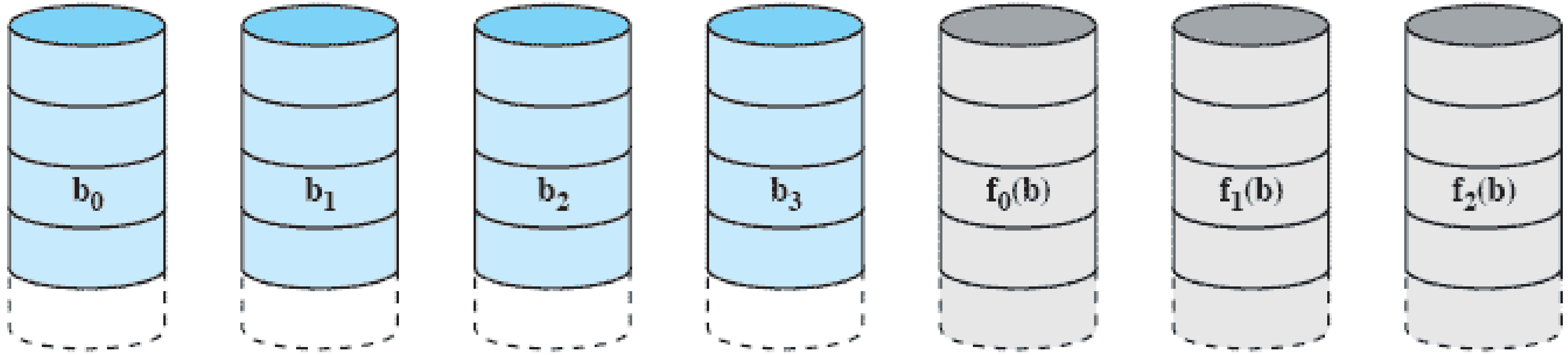


(b) RAID 1 (mirrored)

Source: Pearson

RAID Level 2

- ❑ **Distribute each byte/word over multiple disks**
- ❑ **Add hamming code**
  - For example, for 4b nibbles, 3b extra
- ❑ **Issues**
  - Require all drives to be rotationally synchronized
  - Require a substantial number of drives
  - On a write, all data disks and parity disk must be accessed
- ❑ **Effective choice where many disk errors occur**
  - Usually RAIS2 is a overkill and is not implemented

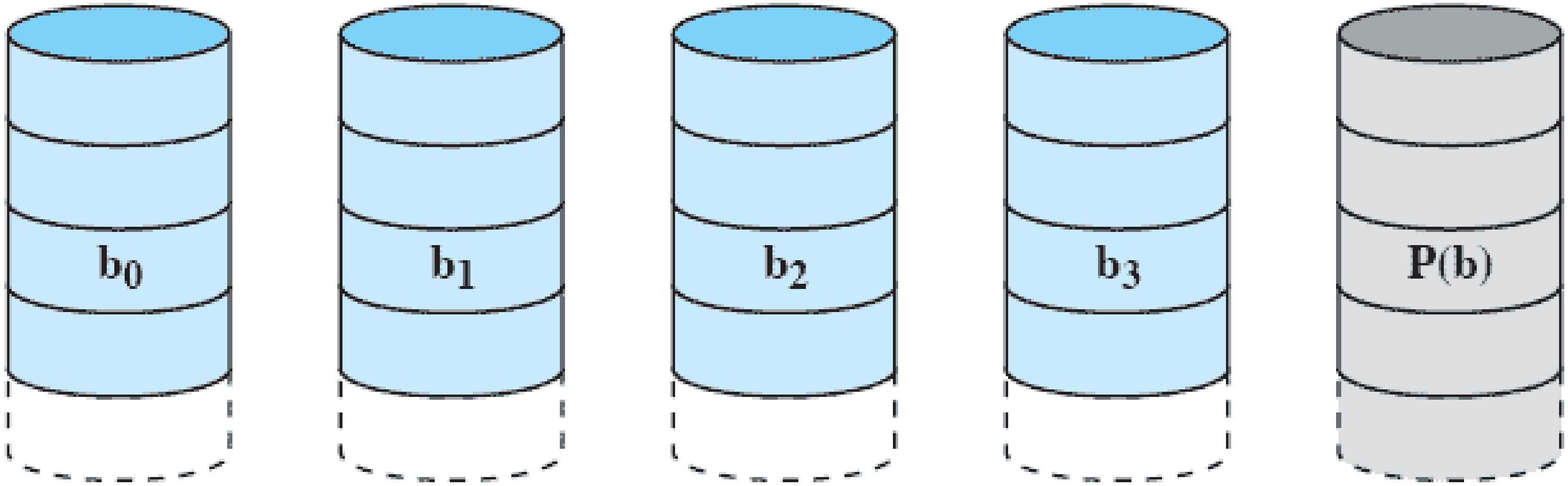


(c) RAID 2 (redundancy through Hamming code)

Source: Pearson

RAID Level 3

- **Distribute each byte/word over multiple disks**
- **Add parity bit (bit-interleaved parity)**
  - Requires only a single redundant disk, no matter how large the disk array
  - In case of a disk failure, the parity drive is accessed and data is reconstructed from the remaining devices.
- **Can achieve very high data transfer rates**

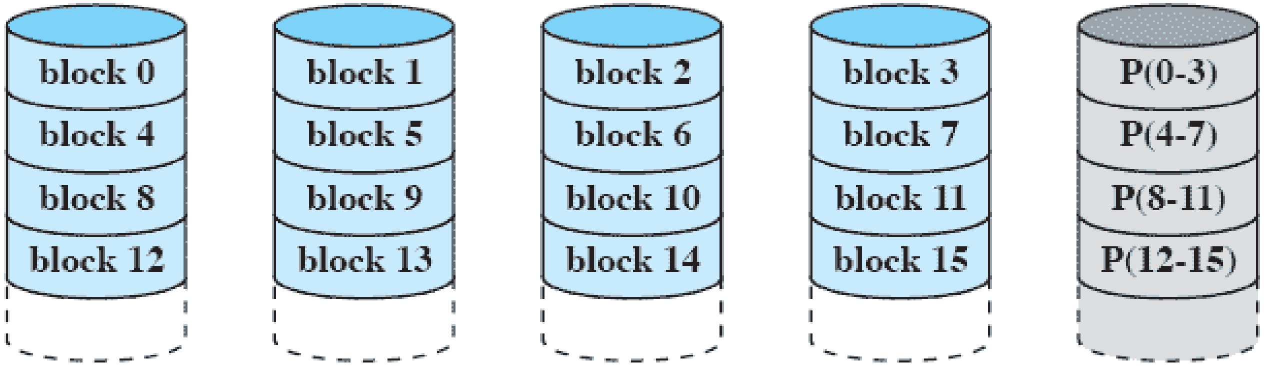


(d) RAID 3 (bit-interleaved parity)

Source: Pearson

RAID Level 4

- **RAID 4~6 make use of an** ***independent access technique***
  - Each member disk operates independently. Separate IO requests can be satisfied in parallel.
  - Suitable for applications with high IO request rates but not suitable for applications with high data transfer rates
- **Block-interleaved parity**
  - A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- **A write to disk X1 requires 2 reads of disk X1 and X4(parity) and 2 writes of disk X1 and X4**



(e) RAID 4 (block-level parity)

Source: Pearson

RAID 4 Level

- **Initially, the following relationship holds for each bit I**
  - $X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$
- **After the write**
  - $X4'(i) = X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i)$

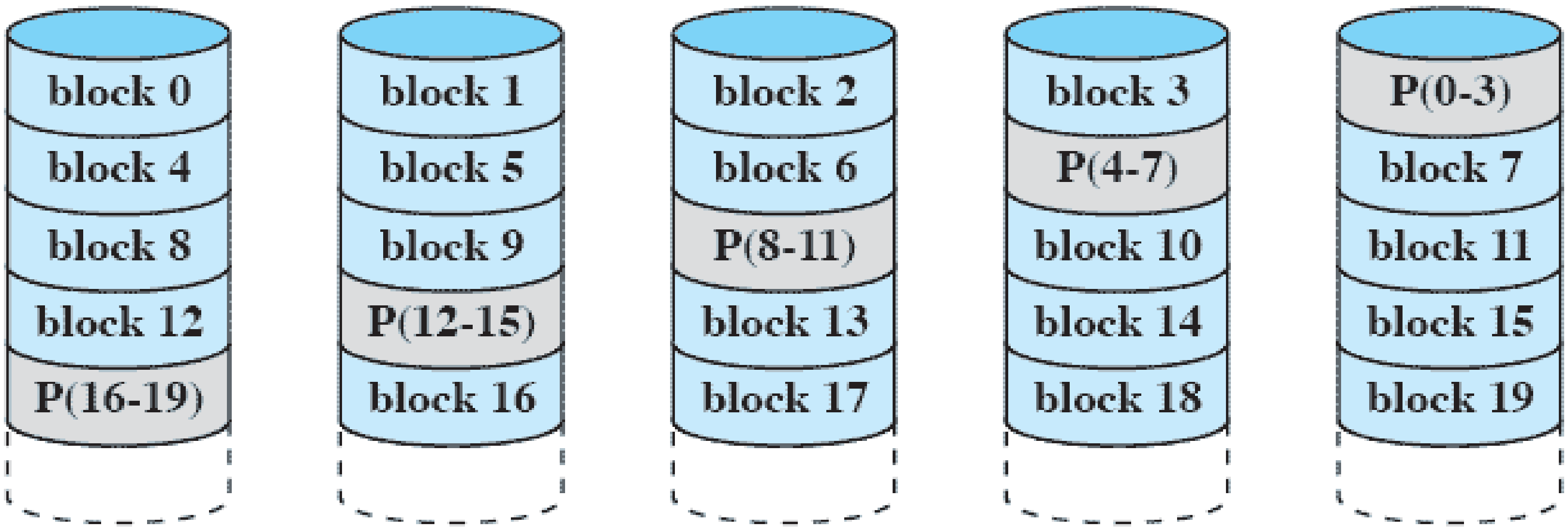
    $= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i)$

    $= X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i)$

    $= X4(i) \oplus X1(i) \oplus X1'(i)$
- **Therefore, to calculate the new parity, it must read the old user data and the old user parity**
  - Every write operation must involve the parity disk, which can become a bottleneck.

RAID Level 5

- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss
- Widely used

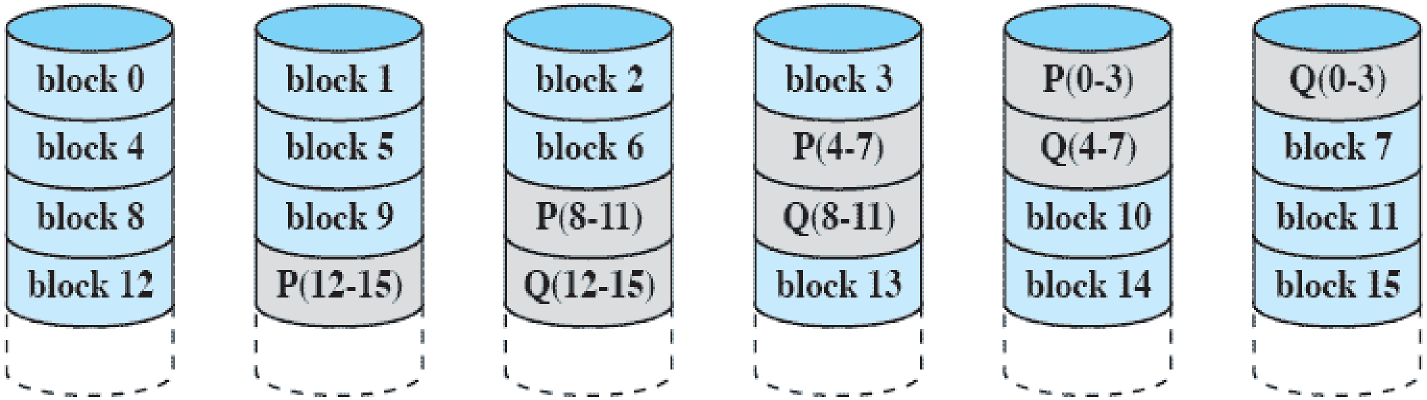


(f) RAID 5 (block-level distributed parity)

Source: Pearson

RAID Level 6

- **Two different parity calculations are carried out and stored in separate blocks on different disks**
  - One may use parity (exclusive-OR) and the other can be an independent algorithm
- **Provides extremely high data availability**
- **Incurs a substantial write penalty because each write affects two parity blocks**
  - Compared to RAID5, RAID6 can suffer more than a 30% drop in write performance



(g) RAID 6 (dual redundancy)

Source: Pearson

Disk Cache

- ***Disk cache* is a buffer in main memory for disk sectors**
  - Contains a copy of some of the sectors on the disk
- **When an I/O request is made for a particular sector, a check is made to determine if the sector is in the disk cache**
  - If Yes, the request is satisfied via the cache
  - If No, the requested sector is read into the disk cache from the disk

LRU

- The most commonly used algorithm
- The block that has not been referenced for the longest time is replaced
- A stack of pointers reference the cache
  - Most recently referenced block is on the top of the stack
  - When a block is referenced or brought into the cache, it is placed on the top of the stack

LFU (Least Frequently Used)

- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block is accessed
- When replacement is required, the block with the smallest count is selected
- Problematic when
  - Certain blocks are referenced relatively infrequently overall, but when they are referenced, there are short intervals of repeated references due to locality, building up high reference counts. After such interval is over, the reference count may be misleading.

Homework 10

- Exercise 11.1
- Exercise 11.4
- Exercise 11.6
- Exercise 11.8