# UNIT -4 PART-1

# INPUT/OUTPUT Management and Disk Scheduling

# Roadmap

**I/O Devices**

- Organization of the I/O Function
- I/O Buffering
- Disk Scheduling
- Raid

# Categories of I/O Devices

- Difficult area of OS design
  - Difficult to develop a consistent solution due to a wide variety of devices and applications
- Three Categories:
  - Human readable
  - Machine readable
  - Communications


## (1) Human readable

- Devices used to communicate with the user
- Printers and terminals
  - Video display
  - Keyboard
  - Mouse etc

## (2) Machine readable

•Used to communicate with electronic equipment like..

- Disk drives
- USB keys
- Sensors
- Controllers
- Actuators

## (3) Communication

- Used to communicate with remote devices
  - Digital line drivers
  - Modems

# Differences in I/O Devices

- Devices differ in a number of areas
  - Data Rate
  - Application
  - Complexity of Control
  - Unit of Transfer
  - Data Representation
  - Error Conditions

# (1) Data Rate

- May be massive difference between the data transfer rates of devices

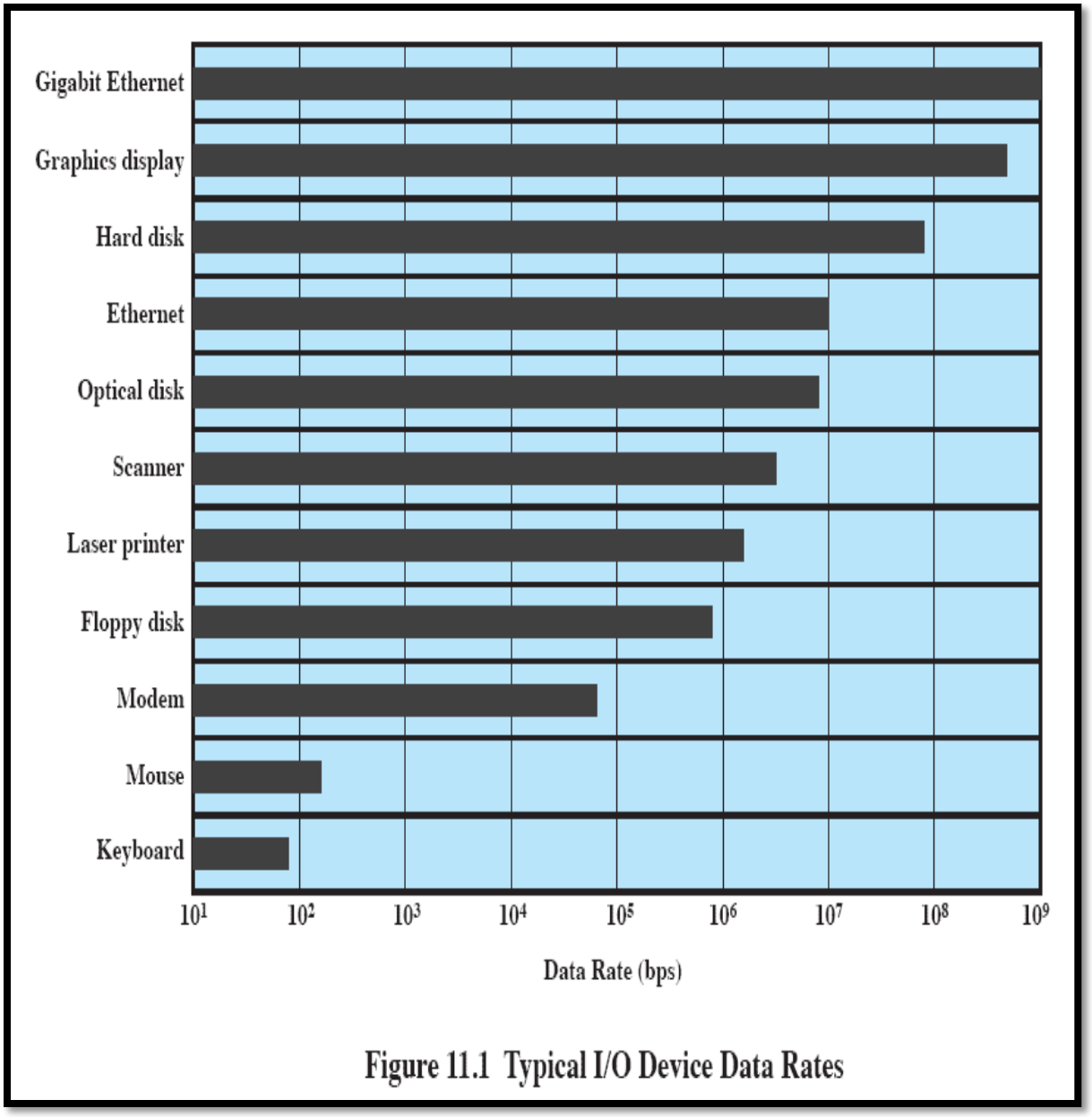




Figure 11.1 Typical I/O Device Data Rates

# (2) Application

The use to which a device is put has an influence on the software and policies in the operating system and supporting utilities.

Examples:

- disk used for files requires the support of file management software.
- disk used as a backing store for pages in a virtual memory scheme depends on the use of virtual memory hardware and software.

These applications have an impact on disk scheduling algorithms.

Another example, a terminal may be used by an ordinary user or a system administrator.

- implying different privilege levels and perhaps different priorities in the operating system.

## (3) Complexity of control

- A printer requires a relatively simple control interface.
- A disk is much more complex.
- This complexity is filtered to some extent by the complexity of the I/O module that controls the device.

## (4) Unit of transfer

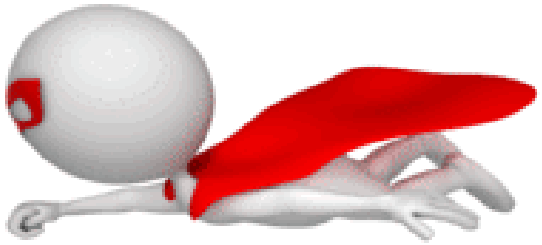
- Data may be transferred as
  - a stream of bytes or characters (e.g., terminal I/O)
  - or in larger blocks (e.g., disk I/O).

## (5) Data representation

- Different data encoding schemes are used by different devices,
  - including differences in character code and parity conventions.

## (6) Error Conditions

- The nature of errors differ widely from one device to another.
- Aspects include:
  - the way in which they are reported,
  - their consequences,
  - the available range of responses



# Roadmap

- I/O Devices
- **Organization of the I/O Function**
- I/O Buffering
- Disk Scheduling
- Raid

# Techniques for performing I/O

**Q-1: List and briefly define three techniques for performing I/O.**

Following summarized three techniques for performing I/O:

- **Programmed I/O:-**
  - The processor issues an I/O command, on behalf of a process, to an I/O module;

- **Interrupt-driven I/O:-**
  - The processor issues an I/O command on behalf of a process.
  - If the I/O instruction from the process is nonblocking, then the processor continues to execute instructions from the process that issued the I/O command.
  - Else the next instruction that the processor executes is from the OS, which will put
  - the current process in a blocked state and schedule another process.

# Techniques for performing I/O

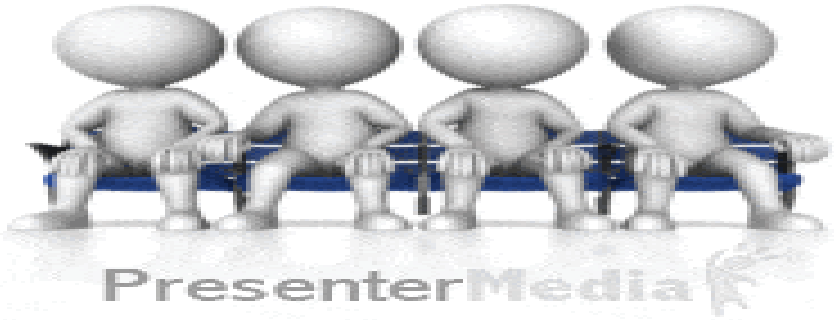
- **Direct memory access (DMA):-**
  - A DMA module controls the exchange of data between main memory and an I/O module.
  - The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

# Techniques for performing I/O

Table 11.1 indicates the relationship among these three techniques

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) |



# Evolution of the I/O Function

- As computer systems have evolved, there has been a pattern of increasing complexity

  and sophistication of individual components.

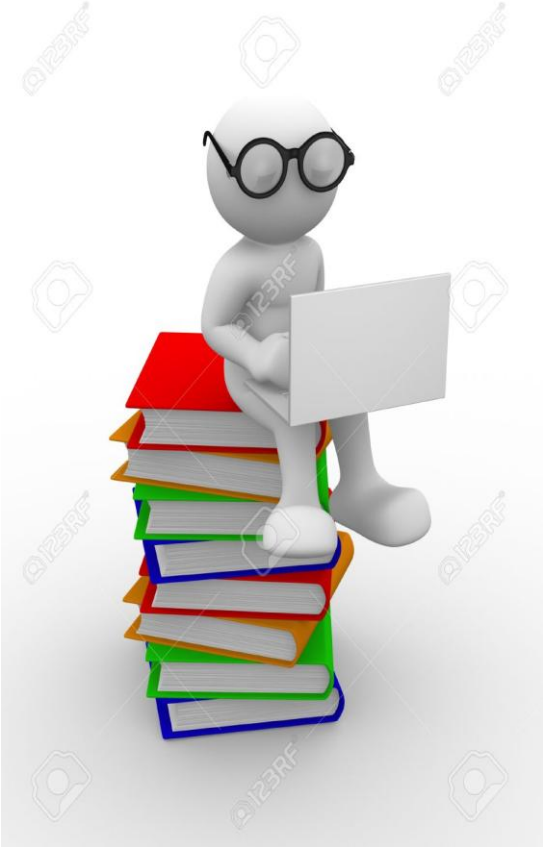
- The evolutionary steps can be summarized as follows:

1. Processor directly controls a peripheral device
2. Controller or I/O module is added
   - Processor uses programmed I/O without interrupts
   - Processor does not need to handle details of external devices

3. Now interrupts are employed.
   - The processor need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.

# Evolution of the I/O Function

4. The I/O module is given direct control of memory via DMA.
   - It can now move a block of data to or from memory without involving the processor, except at the beginning and end of the transfer.
5. I/O module is a separate processor
   - CPU directs the I/O processor to execute an I/O program in main memory.
6. I/O processor
   - I/O module has its own local memory
   - Commonly used to control communications with interactive terminals



# Basic Operation Of DMA

- When the processor wishes read or send a block of data, it issues a command to the DMA module by sending some information to DMA module. The information includes:
- read or write command, sending through read and write control lines.
- number of words to be read or written, communicated on the data lines and stored in the data count register.

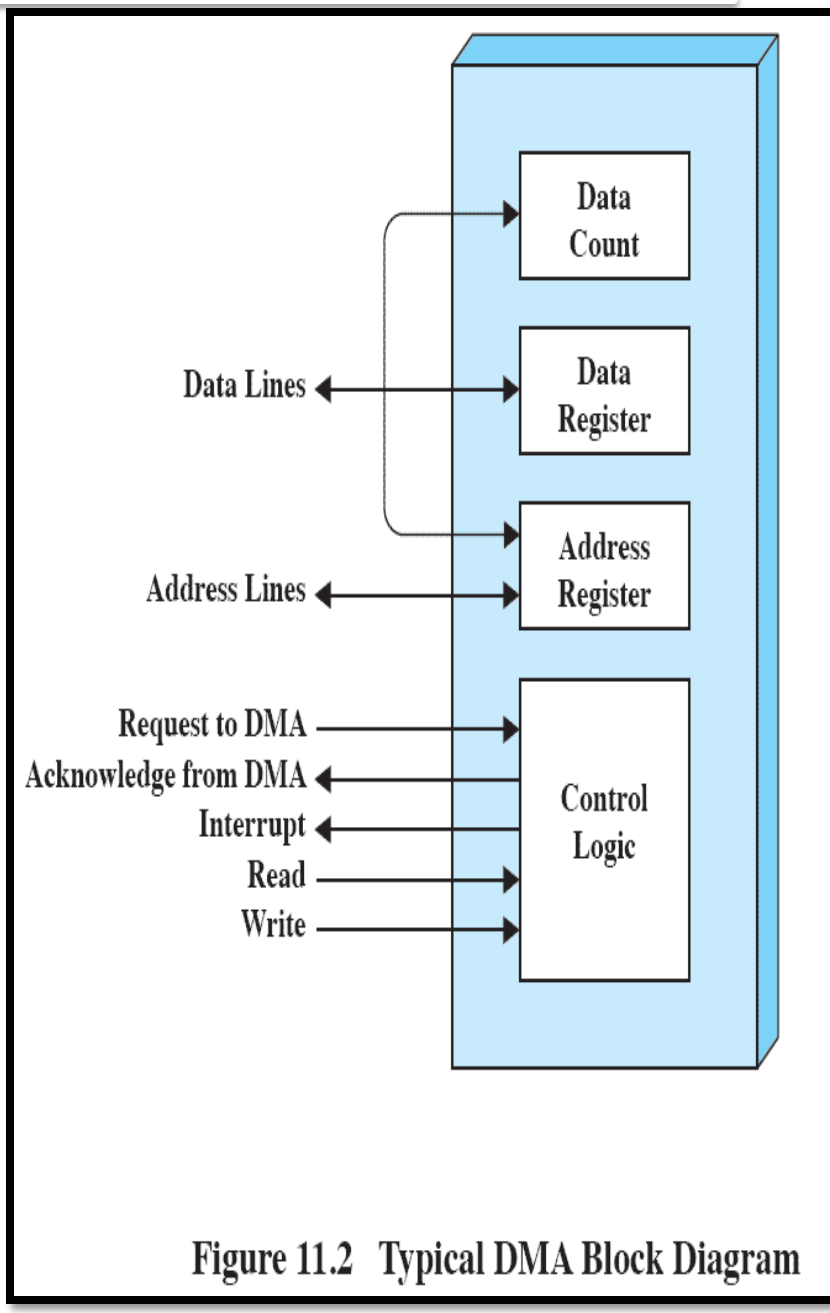


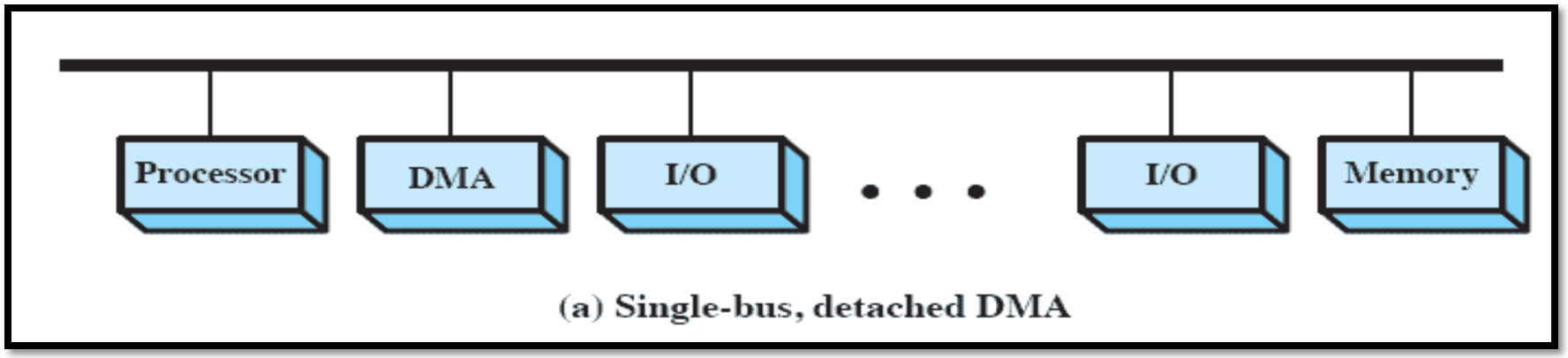

Figure 11.2 Typical DMA Block Diagram

# Direct Memory Access

•starting location in memory to read from or write to, communicated on data lines and stored in the address register.

- address of the I/O device involved, communicated on the data lines.
- After the information are sent, the processor continues with other work.
- The DMA module then transfers the entire block of data directly to or from memory without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor to inform that it has finish using the system bus.

# DMA Configurations:
# 1. Single Bus

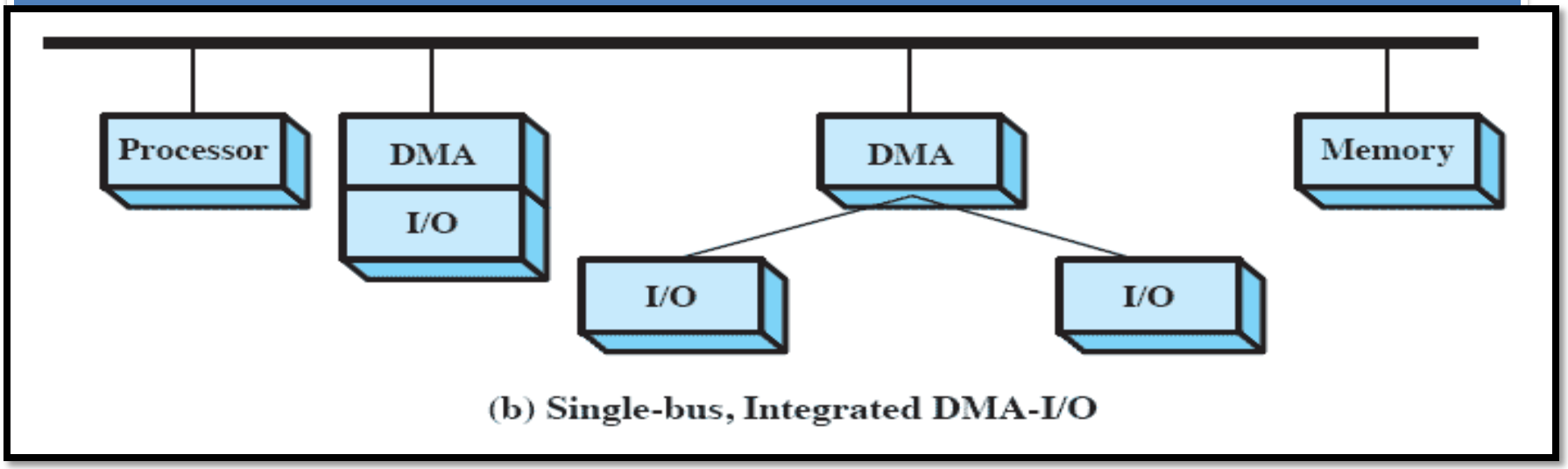



(a) Single-bus, detached DMA

Some possibilities are shown here In the first example, all modules share the same system bus.

- The DMA module, uses programmed I/O to exchange data between memory and an I/O module through the DMA module.
- This is clearly inefficient: As with processor-controlled programmed I/O, each transfer of a word consumes two bus cycles (transfer request followed by transfer).

# DMA Configurations:
# 2.Integrated DMA & I/O



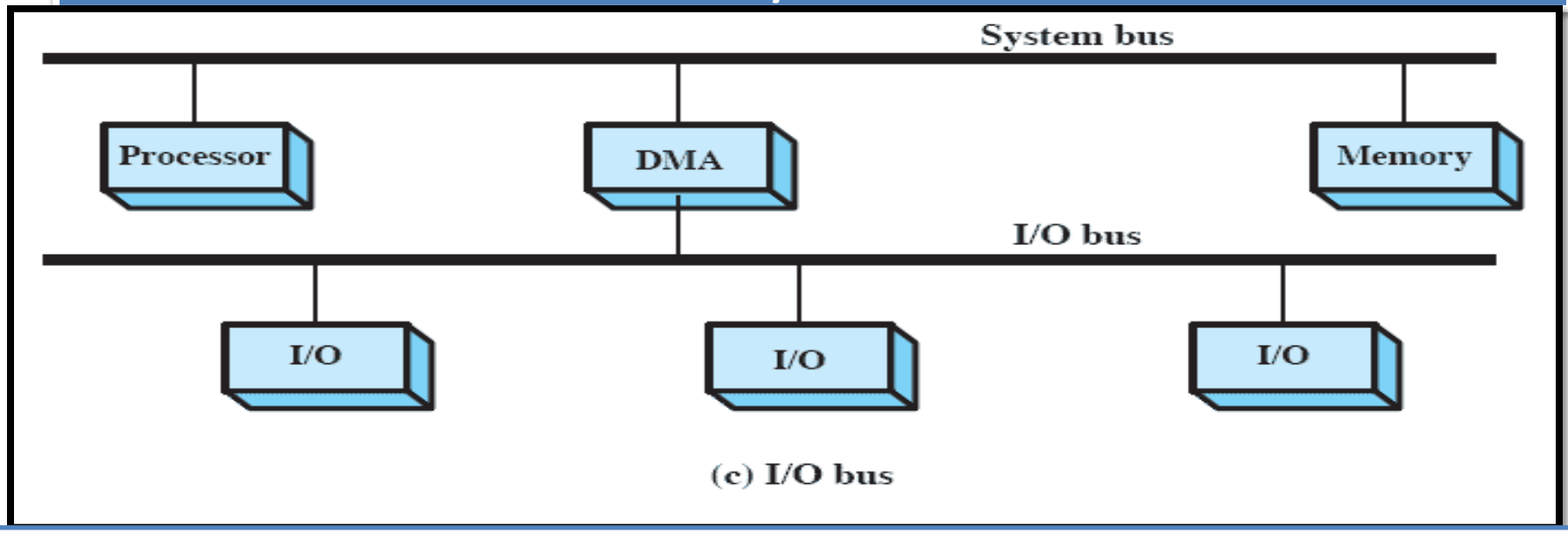

(b) Single-bus, Integrated DMA-I/O

The number of required bus cycles can be cut substantially by integrating the DMA and I/O functions.

This means that there is a path between the DMA module and one or more I/O modules that does not include the system bus.

The DMA logic may actually be a part of an I/O module, or it may be a separate module that controls one or more I/O modules.

# DMA Configurations:
# 3. I/O Bus




(c) I/O bus

This concept can be taken one step further by connecting I/O modules to the DMA module using an I/O bus

This reduces the number of I/O interfaces in the DMA module to one and provides for an easily expandable configuration.

In all of these cases the system bus that the DMA module shares with the processor and main memory is used by the DMA module only to exchange data with memory and to exchange control signals with the processor.

The exchange of data between the DMA and I/O modules takes place off the system bus.

# Summary flows of using DMA by processor

- Issue command to DMA module, by sending necessary information to DMA module.
- Processor does other work.
- DMA acquire control on the system, and transfers data to and from within memory and external device.
- DMA sends a signal to processor when the transfer is complete, system control is return to processor.

# Advantages

- Allows a peripheral device to read from/write to memory without going through the CPU
- Allows for faster processing since the processor can be working on something else while the peripheral can be populating memory

# Disadvantages

- requires a DMA controller to carry out the operation, which increases the cost of the system
- cache coherence problems

# Roadmap

- I/O Devices
- Organization of the I/O Function
- **I/O Buffering**
- Disk Scheduling
- Raid
- Disk Cache

# I/O Buffering

- Processes must wait for I/O to complete before proceeding
  - To avoid deadlock certain pages must remain in main memory during I/O
- It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made. This technique is known as buffering.

# Block-oriented Buffering

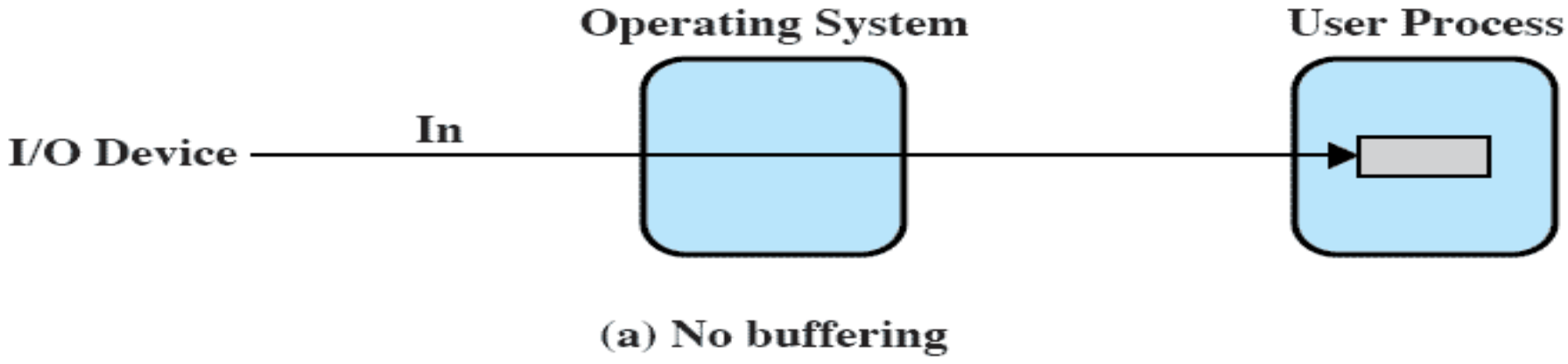
- Information is stored in fixed sized blocks
- Transfers are made a block at a time
  - Can reference data by block number
- Used for disks and USB keys

# Stream-Oriented Buffering

- Transfer information as a stream of bytes
- Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage

## No Buffer

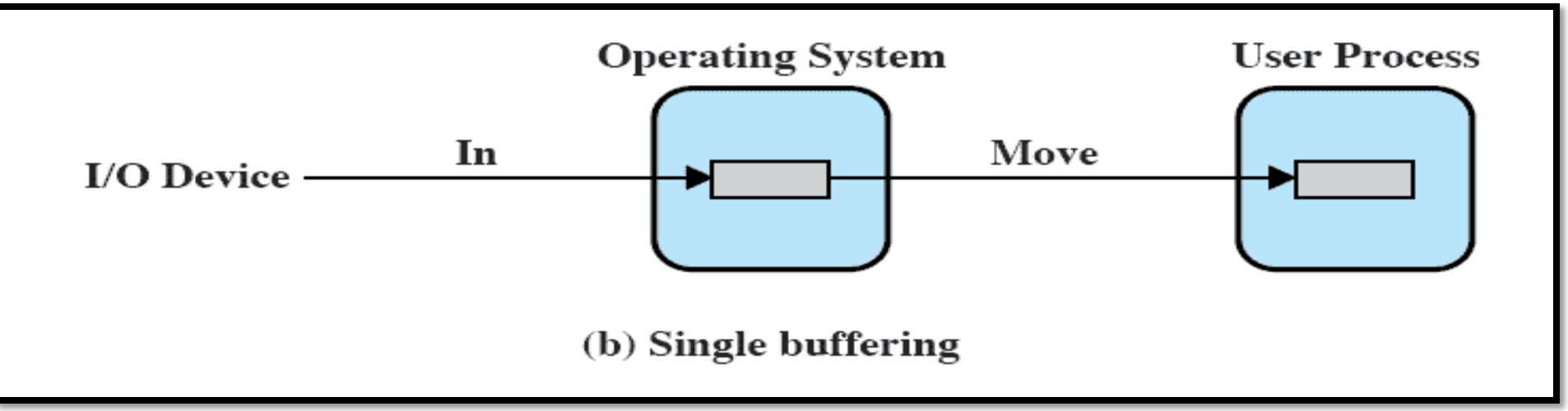
- Without a buffer, the OS directly access the device as and when it needs




(a) No buffering

# Single Buffer

- Operating system assigns a buffer in main memory for an I/O request




## Block Oriented Single Buffer

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.

Called **reading ahead**, or **anticipated input**;

- it is done in the expectation that the block will eventually be needed.

Often this is a reasonable assumption most of the time because data are usually accessed sequentially.

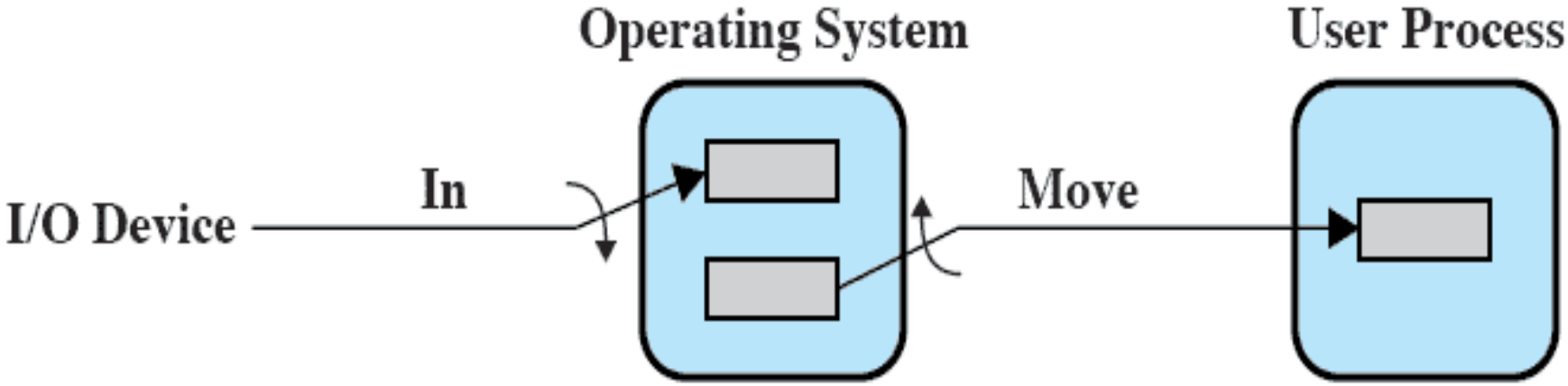
- Only at the end of a sequence of processing will a block be read in unnecessarily.

# Stream-oriented Single Buffer

- The single buffering scheme can be used in a line-at-a-time fashion or a byte-at-a-time fashion.
  - Line-at-a-time operation is appropriate for scroll-mode terminals (sometimes called dumb terminals).
  - Byte-at-a-time operation is used on where each keystroke is significant, or for peripherals such as sensors and controllers.

# Double Buffer

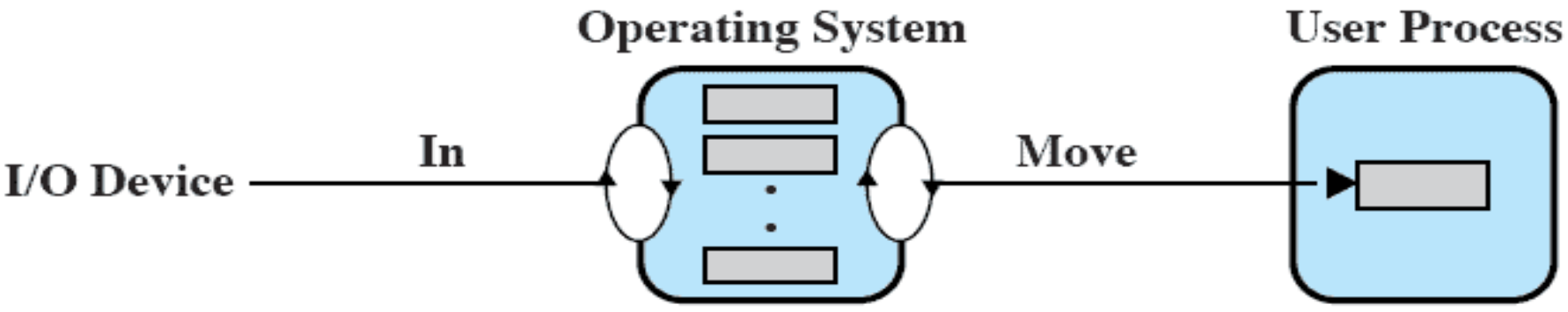
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer




(c) Double buffering

# Circular Buffer

- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process




(d) Circular buffering

# Buffer Limitations

- Buffering is a technique that smoothes out peaks in I/O demand.

- However, no amount of buffering will allow an I/O device to keep pace with a process indefinitely when the average demand of the process is greater than the I/O device can service.
  - Even with multiple buffers, all of the buffers will eventually fill up and the process will have to wait after processing each chunk of data.

- However, in a multiprogramming environment, when there is a variety of I/O activity and a variety of process activity to service, buffering is one tool that can increase the efficiency of the operating system and the performance of individual processes.

# Roadmap

- I/O Devices
- Organization of the I/O Function
- I/O Buffering
- **Disk Scheduling**
- Raid

# Disk Performance Parameters

**Q-2: Briefly define the disk scheduling policies with example.**

- When the disk drive is operating, the disk is rotating at constant speed.

- To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track.

- Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed-head system.

# Disk Performance Parameters

**Access Time = (SEEK TIME + ROTATIONAL DELAY)**

- **Seek time:-**
  - On a movable-head system, the time it takes to position the head at the track is Known as **seek time.**
- **Rotational delay** or **rotational latency:-**
  - The time its takes for the beginning of the sector to reach the head is known as **rotational delay.**
- Seek time + rotational delay = **access time,**
  - The time it takes to get into position to read or write.
- **Transfer Time:-**
- Once the head is in position then the time taken to transfer the data is transfer time.

# Disk Performance Parameters

- The actual details of disk I/O operation depend on many things
  - A general timing diagram of disk I/O transfer is shown here.

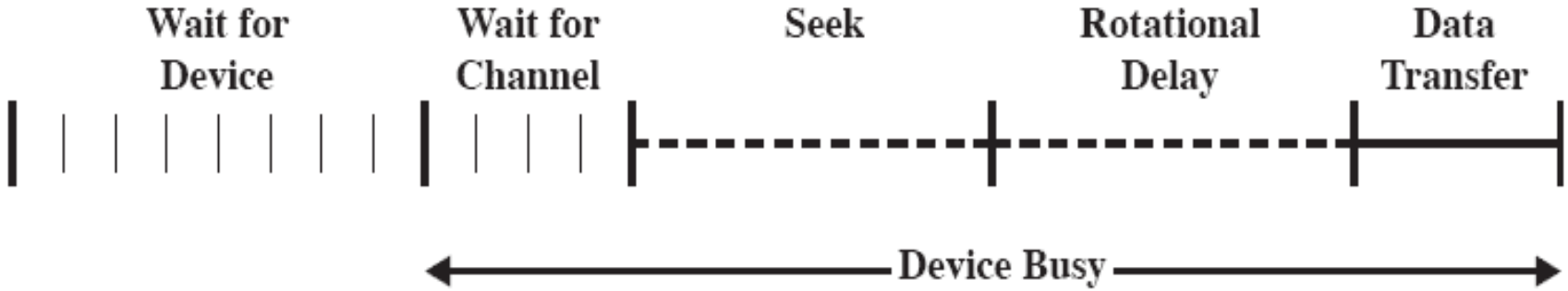



Figure 11.6 Timing of a Disk I/O Transfer

# Disk Scheduling Policies

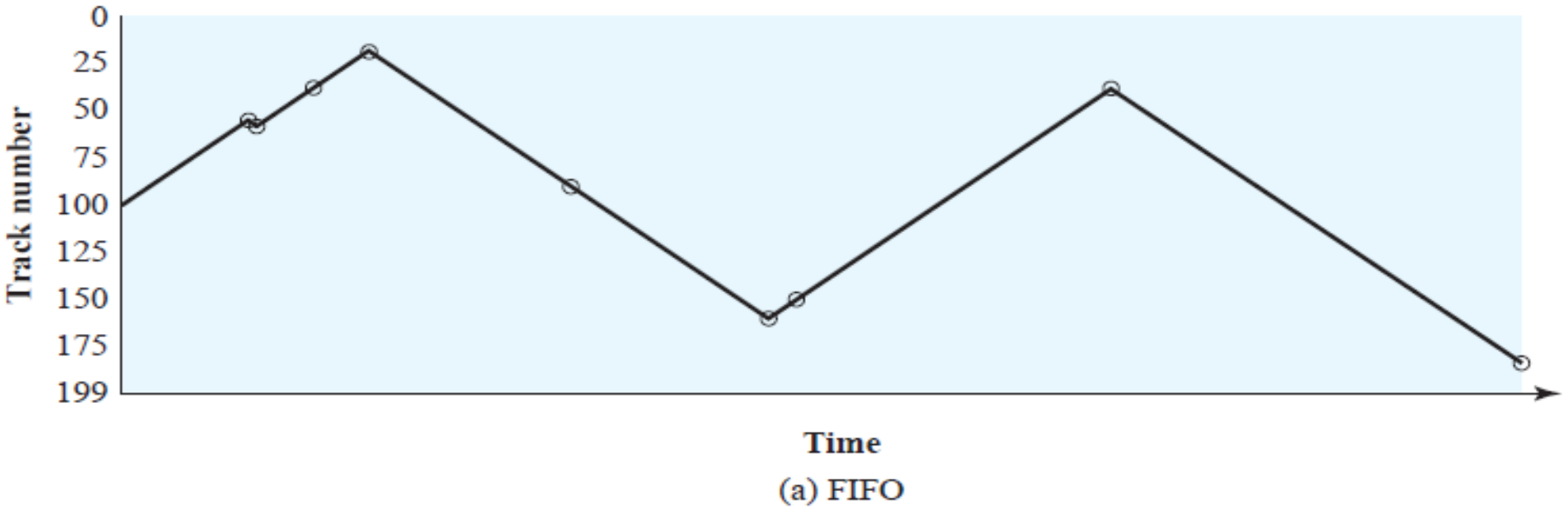
- **RANDOM SHCHEDULING:-**
  - If we selected items from the queue in random order, then we can expect that the tracks to be visited will occur randomly, giving poor performance.
  - This is called **random scheduling**

- To compare various schemes, consider a disk head is initially located at track 100.
  - assume a disk with 200 tracks and that the disk request queue has random requests in it.
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.

# First-in, first-out (FIFO)

- The simplest form of scheduling is first-in-first-out (FIFO) scheduling, which processes items from the queue in sequential order.
- This strategy has the advantage of being fair, because every request is honored in the order received.
- With FIFO, if there are only a few processes that require access and if many of the requests are to clustered file sectors, then we can hope for good performance.
  - However, this technique will often approximate random scheduling in performance, if there are many processes competing for the disk

TRACKS:- 55, 58, 39, 18, 90, 160, 150, 38, 184.




(a) FIFO

Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

# FIFO - PERFORMANCE

**(a) FIFO (starting at track 100)**

| Next track accessed | Number of tracks traversed |
|---|---|
| 55 | 45 |
| 58 | 3 |
| 39 | 19 |
| 18 | 21 |
| 90 | 72 |
| 160 | 70 |
| 150 | 10 |
| 38 | 112 |
| 184 | 146 |
| Average seek length | 55.3 |

Table 11.2 Comparison of Disk Scheduling Algorithms

- Figure 11.7a illustrates the disk arm movement with FIFO.
  - This graph is generated directly from the data in Table 11.2a

# Shortest Service Time First

- The SSTF policy is to 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.

TRACKS:- 55, 58, 39, 18, 90, 160, 150, 38, 184.

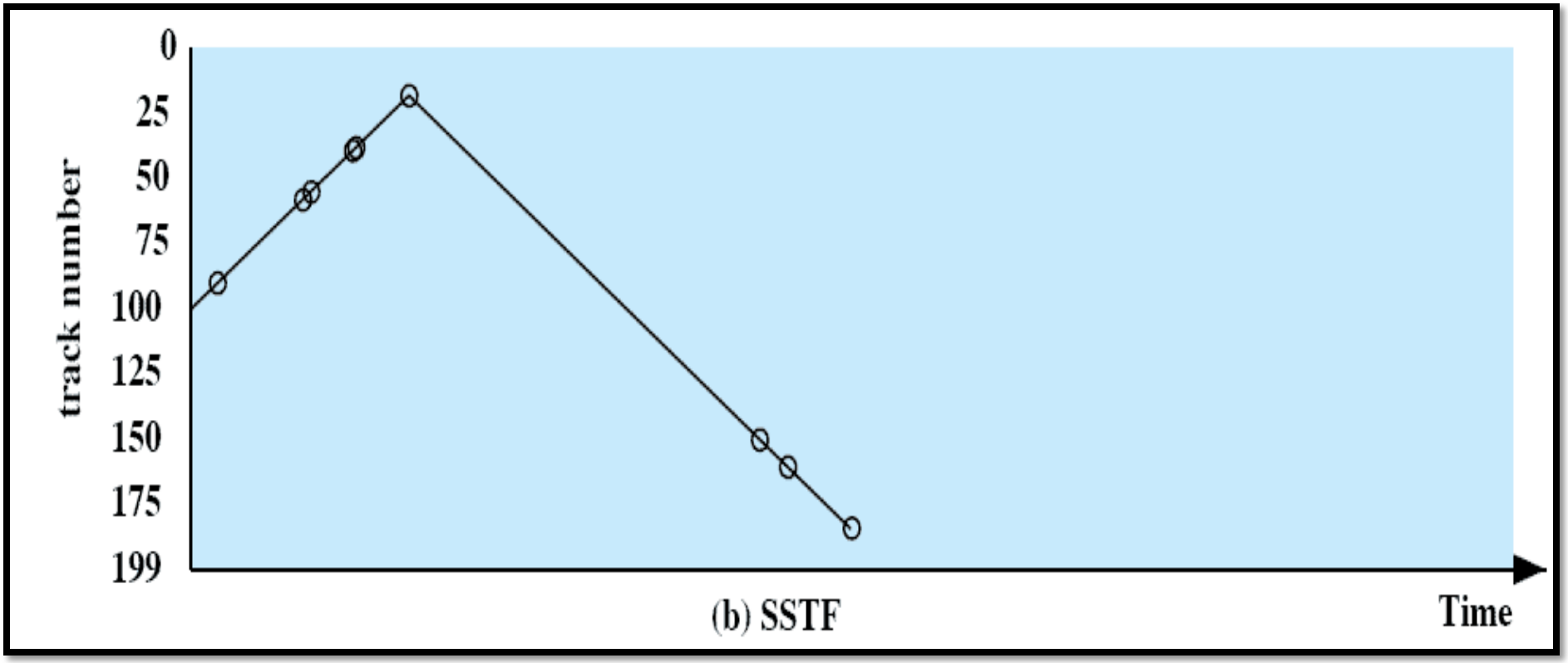



Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

# Shortest Service Time First

**(b) SSTF (starting at track 100)**

| Next track accessed | Number of tracks traversed |
|---|---|
| 90 | 10 |
| 58 | 32 |
| 55 | 3 |
| 39 | 16 |
| 38 | 1 |
| 18 | 20 |
| 150 | 132 |
| 160 | 10 |
| 184 | 24 |
| Average seek length | 27.5 |

Table 11.2 Comparison of Disk Scheduling Algorithms

- Always choosing the minimum seek time does not guarantee that the average seek time over a number of arm movements will be minimum.
- However, this should provide better performance than FIFO.
- Because the arm can move in two directions, a random tie-breaking algorithm may be used to resolve cases of equal distances.

# SCAN

- Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction then the direction is reversed

TRACKS:- 55, 58, 39, 18, 90, 160, 150, 38, 184.

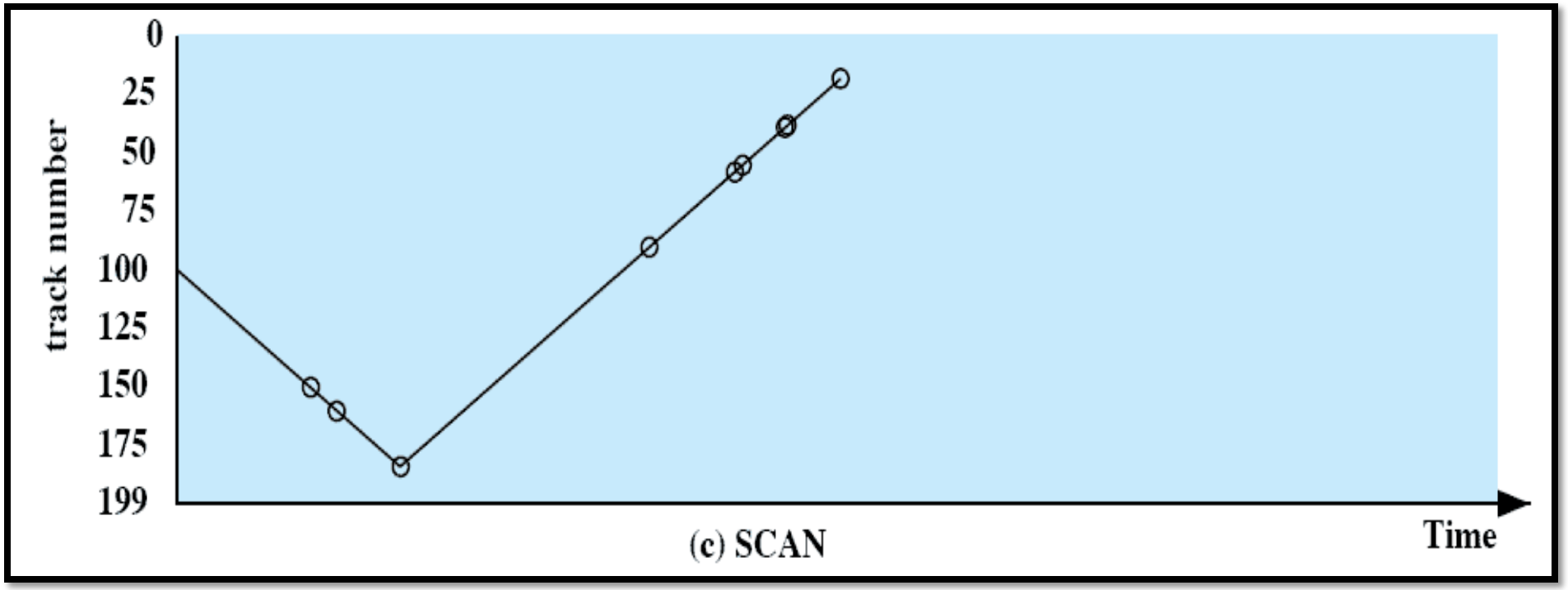



Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

# SCAN

**(c) SCAN (starting at track 100, in the direction of increasing track number)**

| Next track accessed | Number of tracks traversed |
|---|---|
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 90 | 94 |
| 58 | 32 |
| 55 | 3 |
| 39 | 16 |
| 38 | 1 |
| 18 | 20 |
| Average seek length | 27.8 |

Table 11.2 Comparison of Disk Scheduling Algorithms

- The arm is required to move in one direction only until there are no more requests in that direction.
- The service direction is then reversed and the scan proceeds in the opposite direction, again picking up all requests in order.

This latter refinement is sometimes referred to as the LOOK policy.

The SCAN policy favors jobs whose requests are for tracks nearest to both innermost and outermost tracks and favors the latest-arriving jobs.

# C-SCAN

- The C-SCAN (circular SCAN) policy restricts scanning to one direction only.
  - Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again.
- This reduces the maximum delay experienced by new requests.

TRACKS:- 55, 58, 39, 18, 90, 160, 150, 38, 184.

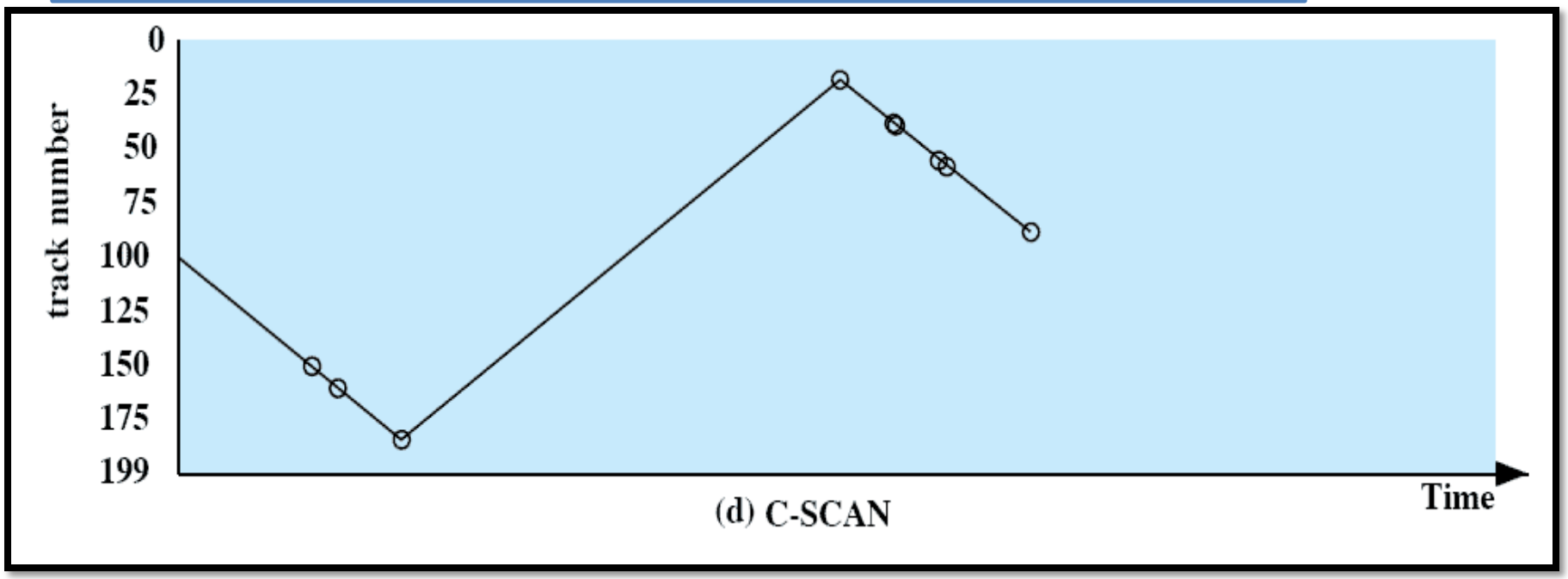



Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

# C-SCAN

**(d) C-SCAN (starting at track 100, in the direction of increasing track number)**

| Next track accessed | Number of tracks traversed |
|---|---|
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 18 | 166 |
| 38 | 20 |
| 39 | 1 |
| 55 | 16 |
| 58 | 3 |
| 90 | 32 |
| **Average seek length** | 35.8 |


Table 11.2 Comparison of Disk Scheduling Algorithms

# Performance Compared

**Table 11.2** 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 |

# N-step-SCAN

- The N-step-SCAN policy segments the disk request queue into sub queues of length N.

- Subqueues are processed one at a time, using SCAN.

- While a queue is being processed, new requests must be added to some other queue.

- If fewer than N requests are available at the end of a scan, then all of them are processed with the next scan.

# FSCAN

- Two sub queues
- When a scan begins, all of the requests are in one of the queues, with the other empty.
- All new requests are put into the other queue.
  - Service of new requests is deferred until all of the old requests have been processed.

# Disk Scheduling Algorithms

Table 11.3 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 |

# Roadmap

- I/O Devices
- Organization of the I/O Function
- I/O Buffering
- Disk Scheduling
- **Raid**

# RAID

**Q-3: Briefly define the seven RAID levels.**

- Redundant Array of Independent Disks

- The RAID scheme consists of seven levels, zero through six.
- These levels do not imply a hierarchical relationship but designate different design architectures that share three common characteristics:
  1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
  2. Data are distributed across the physical drives of an array in a scheme known as striping, described subsequently.
  3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

**RAID**

**Parity** data is used by some RAID levels to achieve redundancy. using XOR function to reconstruct the missing data.

For example, suppose two drives in a three-drive RAID 51 array contained the following data:

Drive 1: **01101101**

Drive 2: **11010100**

To calculate parity data for the two drives, an XOR is performed on their data:

**01101101**

XOR **11010100**

**10111001**

The resulting parity data, **10111001**, is then stored on Drive 3.

If Drive 2 were to fail, its data could be rebuilt using the XOR results of the contents of the two remaining drives, Drive 1 and Drive 3:

Drive 1: **01101101**

Drive 3: **10111001** as follows:

**10111001**

XOR **01101101**

**11010100**

The result of that XOR calculation yields Drive 2's contents. **11010100** is then stored on Drive 2, fully repairing the array.

In the case of a RAID 3 array of 12 drives, 11 drives participate in the XOR calculation shown above and yield a value that is then stored on the dedicated parity drive.

# RAID 0 - Stripped

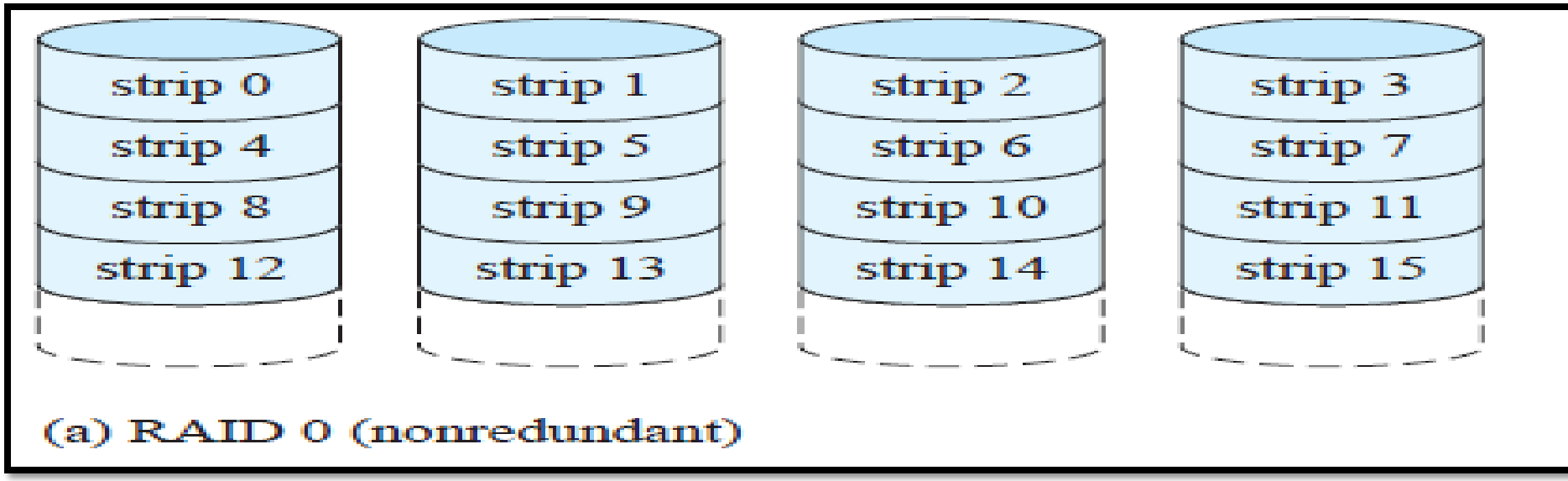



Figure 11.8 RAID Levels

RAID level 0 is not a true member of the RAID family, because it does not include redundancy.

The advantage of this layout is that if a single I/O request consists of multiple logically contiguous strips, then up to n strips for that request can be handled in parallel, greatly reducing the I/O transfer time.

# RAID 1 - Mirrored

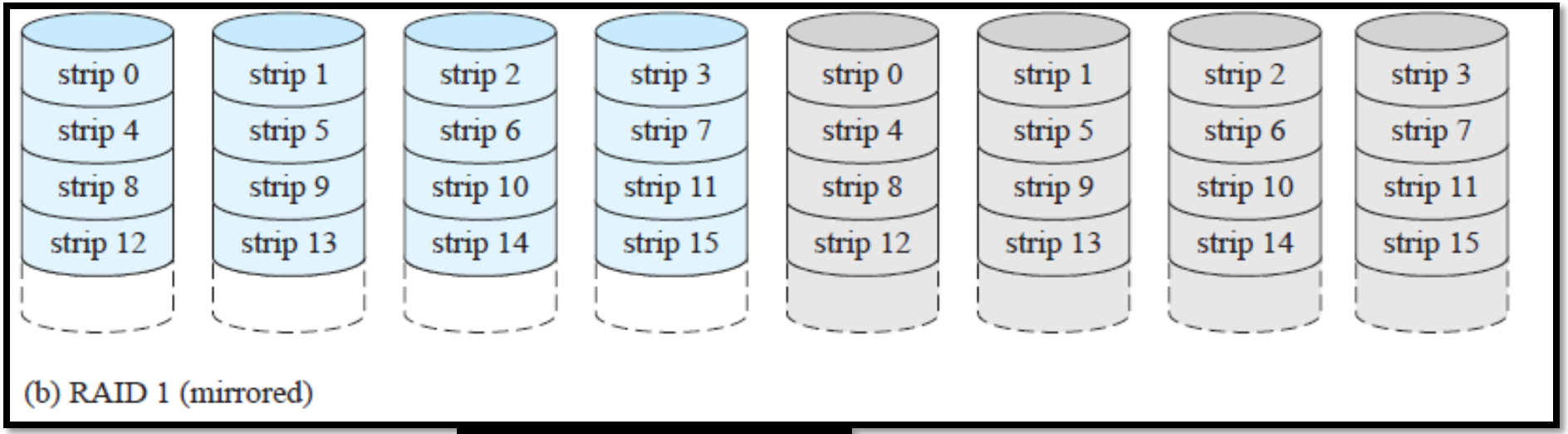


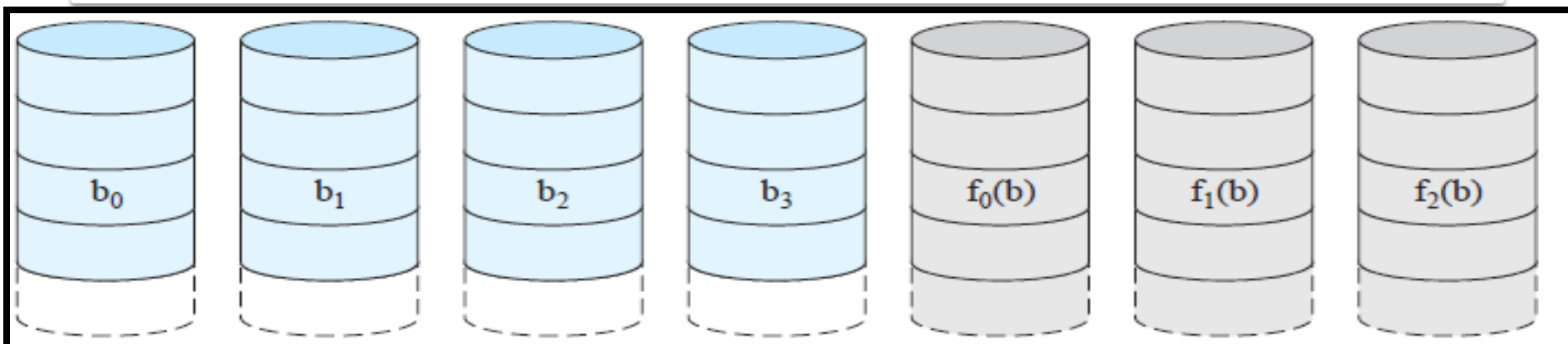

Figure 11.8 RAID Levels

- Each logical strip is mapped to two separate physical disks so that every disk in the array has a mirror disk that contains the same data.
- A read request can be serviced by either of the two disks that contains the requested data, whichever one involves the minimum seek time plus rotational latency.

- A write request requires that both corresponding strips be updated, but this can be done in parallel.
  - Thus, the write performance is dictated by the slower of the two writes

- Recovery from a failure is simple.
  - When a drive fails, the data may still be accessed from the second drive.

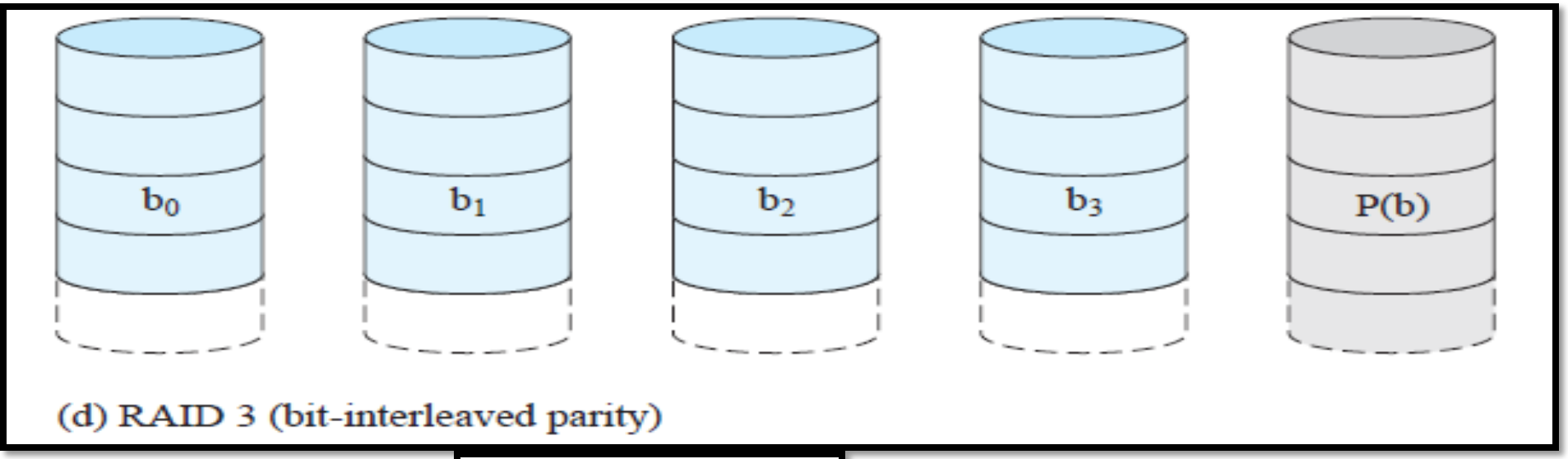
# RAID 2 (Using Hamming code)




(c) RAID 2 (redundancy through Hamming code)

- In a parallel access array, all member disks participate in the execution of every I/O request.
- Typically, the spindles of the individual drives are synchronized so that each disk head is in the same position on each disk at any given time.
- As in the other RAID schemes, data striping is used.
- In the case of RAID 2 and 3, the strips are very small, often as small as a single byte or word.
- With RAID 2, an error-correcting code is calculated across corresponding bits on each data disk, and the bits of the code are stored in the corresponding bit positions on multiple parity disks.
- Typically, a Hamming code is used, which is able to correct single-bit errors and detect double-bit errors.

# RAID 3 bit-interleaved parity




(d) RAID 3 (bit-interleaved parity)

Figure 11.8 RAID Levels

RAID 3 is organized in a similar fashion to RAID 2.

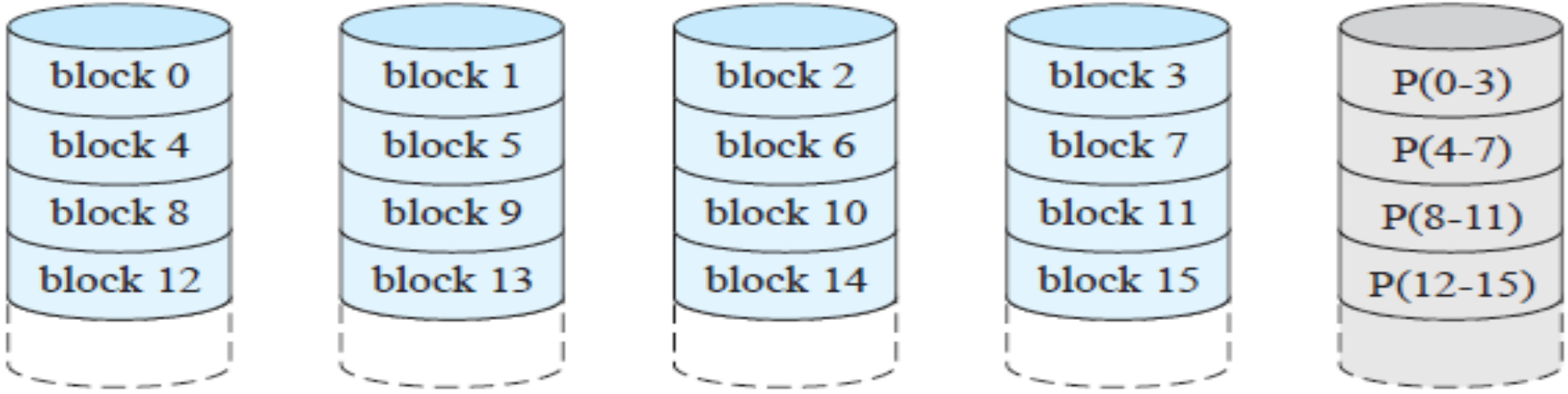
The difference is that RAID 3 requires only a single redundant disk, no matter how large the disk array.

RAID 3 employs parallel access, with data distributed in small strips.

Instead of an error-correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.

# RAID 4 Block-level parity

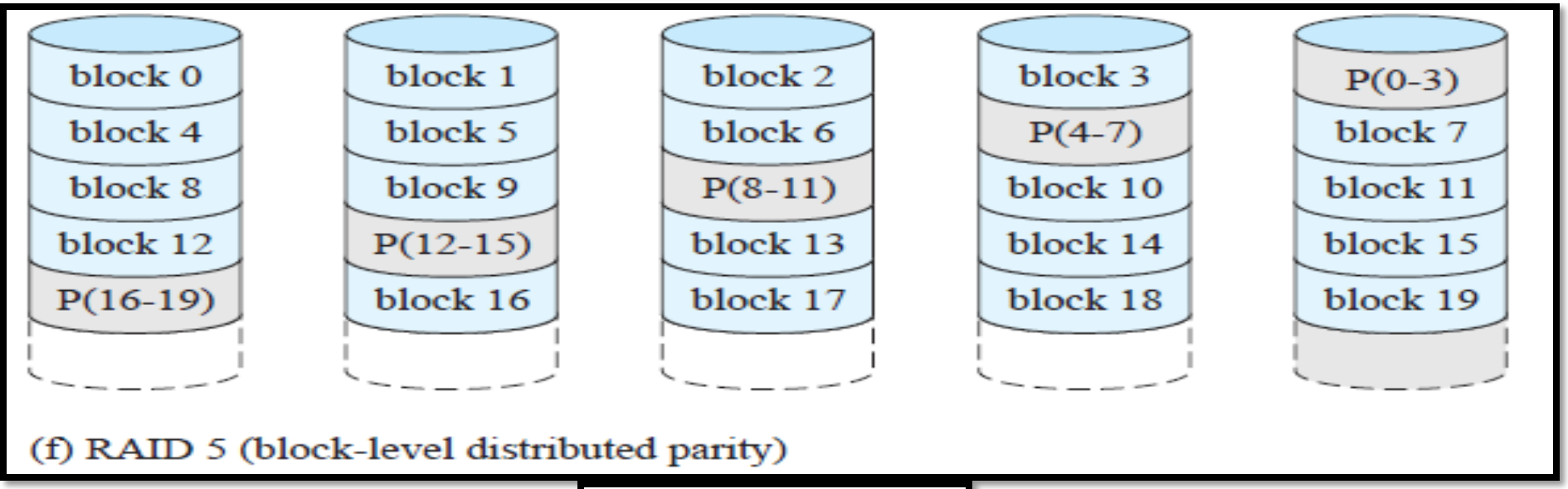
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk
- The parity bits are stored in the corresponding strip on the parity disk.




(e) RAID 4 (block-level parity)

Figure 11.8 RAID Levels

# RAID 5 Block-level Distributed parity




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

Figure 11.8 RAID Levels

RAID 5 is organized in a similar fashion to RAID 4.

RAID 5 distributes the parity strips across all disks.

- A typical allocation is a round-robin scheme
- For an n-disk array, the parity strip is on a different disk for the first n stripes, and the pattern then repeats.

The distribution of parity strips across all drives avoids the potential I/O bottleneck of the single parity disk found in RAID 4.

# RAID 6 Dual Redundancy

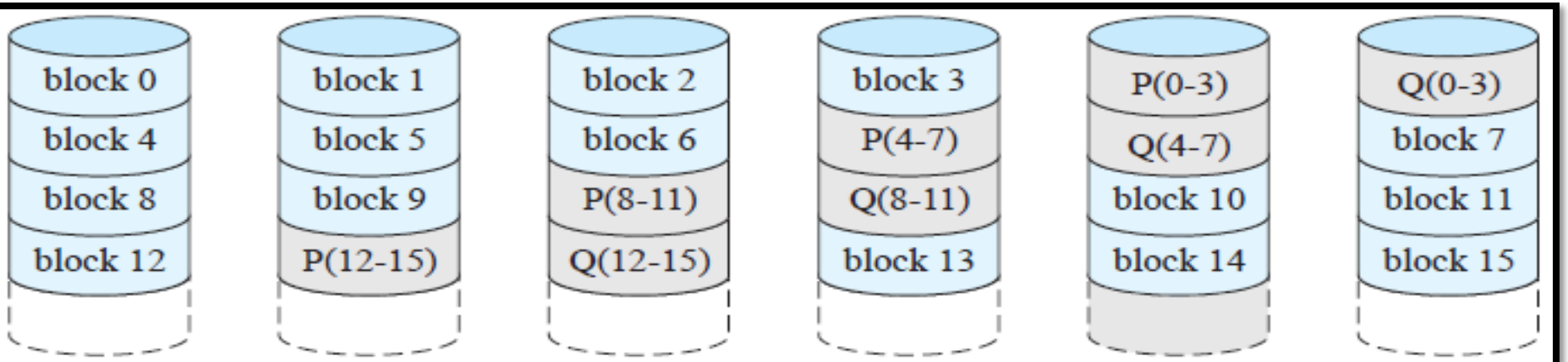



Figure 11.8 RAID Levels

Two different parity calculations are carried out and stored in separate blocks on different disks.

- Thus, a RAID 6 array whose user data require N disks consists of N+2 disks.

P and Q are two different data check algorithms.

- One of the two is the exclusive-OR calculation used in RAID 4 and 5.
- The other is an independent data check algorithm.

This makes it possible to regenerate data even if two disks containing user data fail.


PresenterMedia