Ch 11

*Operating Systems: Internals and Design Principles*

# I/O Management and Disk Scheduling

I/O types
Buffering, buffering types
Disk performance parameters
FIFO, PRI
Shortest service time first
SCAN, C-SCAN, N-step-SCAN, FSCAN
Linus Elevator

RAID, RAID levels

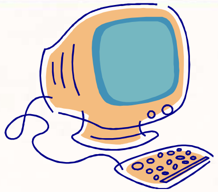
Disk cache
Least Frequently Used replacement
Frequency-Based replacement

# Categories of I/O Devices

External devices that engage in I/O with computer systems can be grouped into three categories:

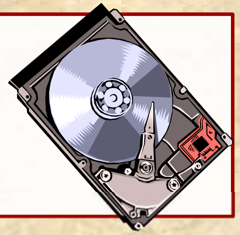
**Human readable**

- suitable for communicating with the computer user
- printers, terminals, video display, keyboard, mouse



**Machine readable**

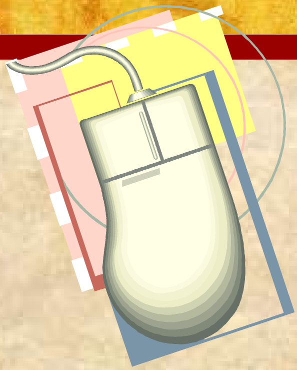
- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers



**Communication**

- suitable for communicating with remote devices
- modems, digital line drivers

# Differences in I/O Devices



*Data Rate*

- there may be differences of magnitude between the data transfer rates

*Application*

- the use to which a device is put has an influence on the software

*Complexity of Control*

- the effect on the operating system is filtered by the complexity of the I/O module that controls the device

*Unit of Transfer*

- data may be transferred as a stream of bytes or characters or in larger blocks

*Data Representation*

- different data encoding schemes are used by different devices

*Error Conditions*

- the nature of errors, the way in which they are reported, their consequences, and the available range of responses differs from one device to another

# Data Rates

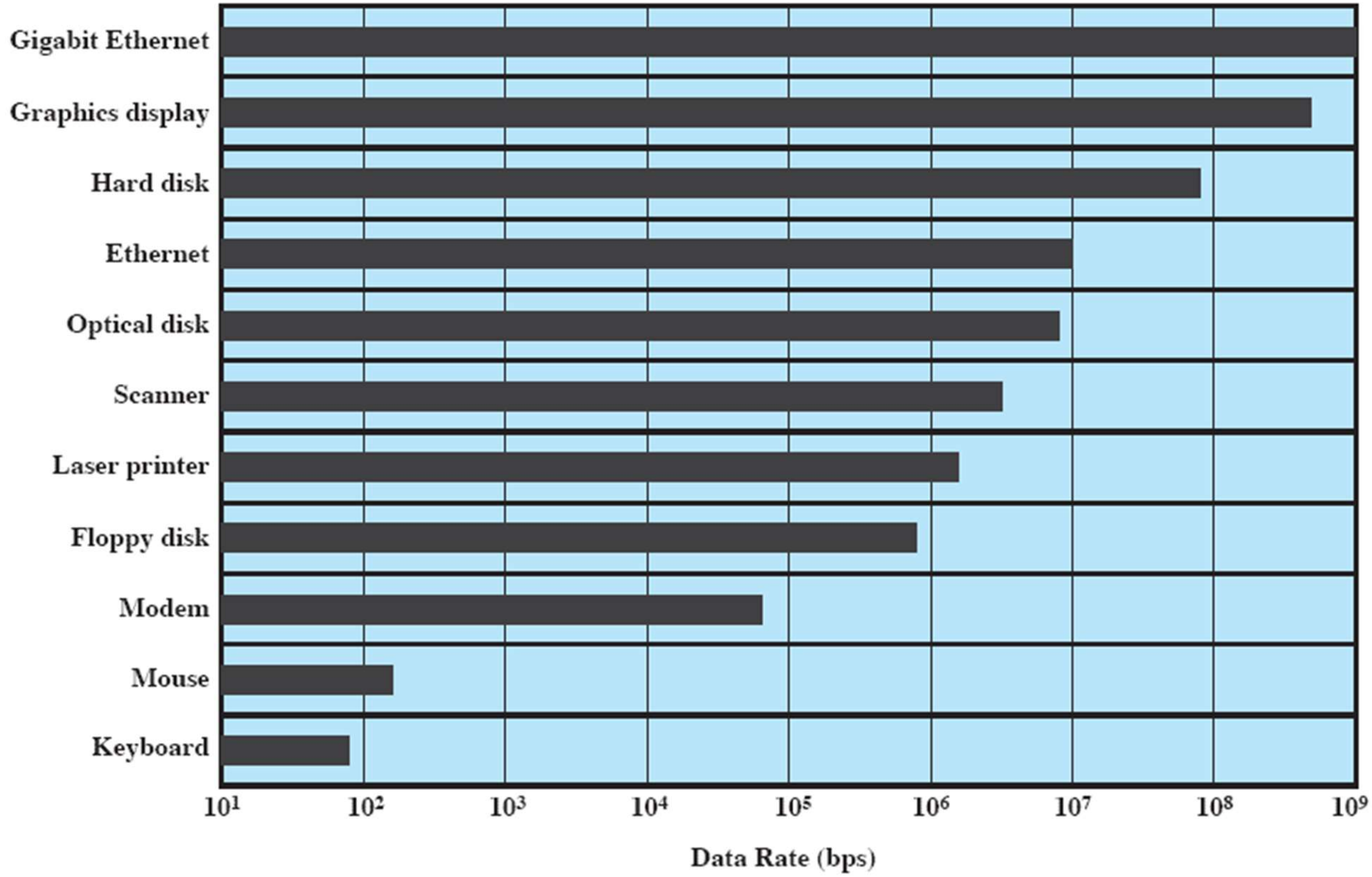



**Figure 11.1 Typical I/O Device Data Rates**

# Techniques for Performing I/O

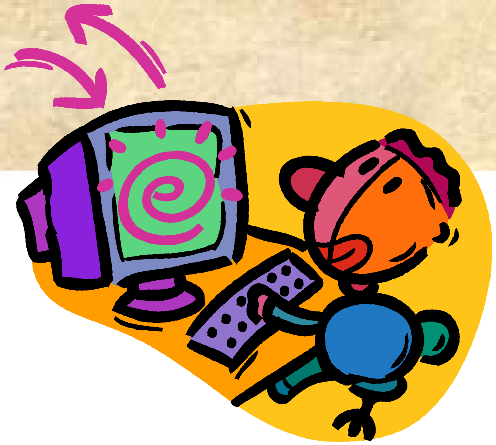


**Table 11.1 I/O Techniques**

| | No Interrupts | Use of Interrupts |
|---|---|---|
| I/O-to-memory transfer through processor | Programmed I/O<br>suora I/O | Interrupt-driven I/O<br>epäsuora I/O<br>keskeyttävä I/O |
| Direct I/O-to-memory transfer | | Direct memory access (DMA) |

# Evolution of the I/O Function

1
- Processor directly controls a peripheral device

2
- A controller or I/O module is added

3
- Same configuration as step 2, but now interrupts are employed

4
- The I/O module is given direct control of memory via DMA

5
- The I/O module is enhanced to become a separate I/O processor, with a specialized instruction set tailored for I/O

6
- The I/O module has a local memory of its own and is, in fact, a computer in its own right

# 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
- Applies to the way processes view I/O devices and the way the operating system manages I/O devices and operations
- Diversity of devices makes it difficult to achieve true generality
- Use a hierarchical, modular approach to the design of the I/O function

# Hierarchical Design

- Functions of the operating system should be separated according to their complexity, their characteristic time scale, and their level of abstraction
- Leads to an organization of the operating system into a series of layers
- Each layer performs a related subset of the functions required of the operating system
- Layers should be defined so that changes in one layer do not require changes in other layers

# A Model of I/O Organization

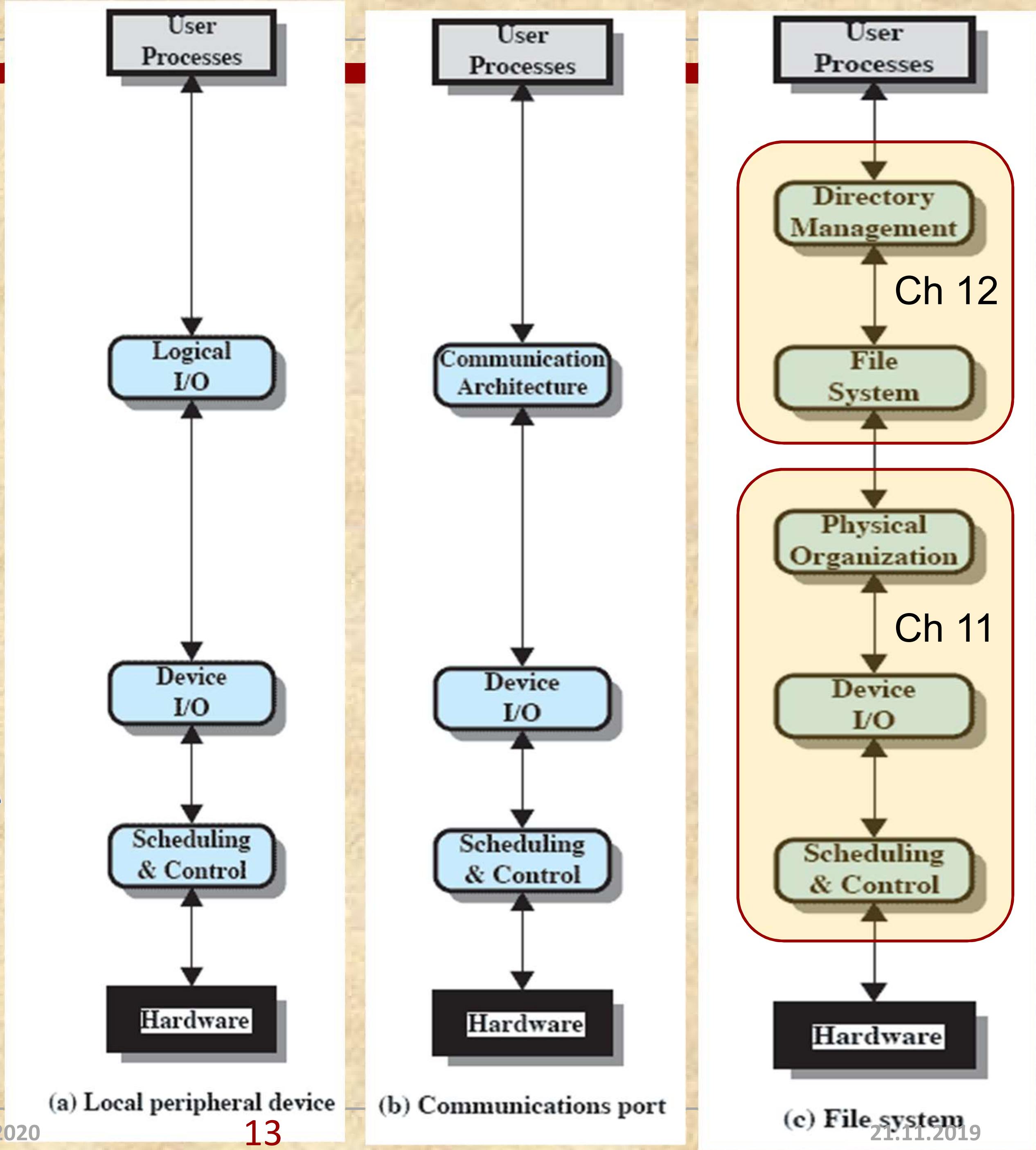



Fig. 11.4

# Buffering

- Perform input transfers in advance of requests being made and perform output transfers some time after the request is made

| Block-oriented device | Stream-oriented device |
|---|---|
| • stores information in <u>blocks</u> that are usually of <u>fixed size</u><br>• transfers are made one block at a time<br>• possible to reference data by its block number<br>• disks and USB keys are examples | • transfers data in and out as a stream of <u>bytes</u><br>• no block structure<br>• terminals, printers, communications ports, and most other devices that are not secondary storage are examples |

# No Buffer

- Without a buffer, the OS directly accesses the device when it needs

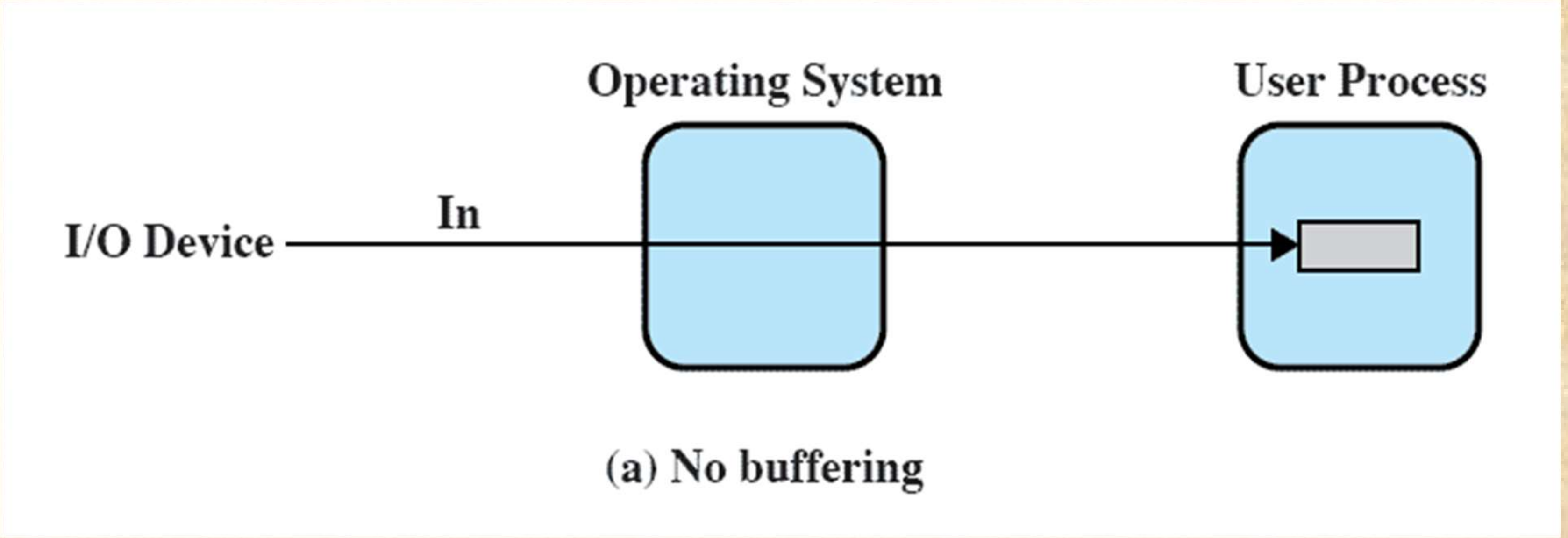



Fig. 11.5

# Single Buffer

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

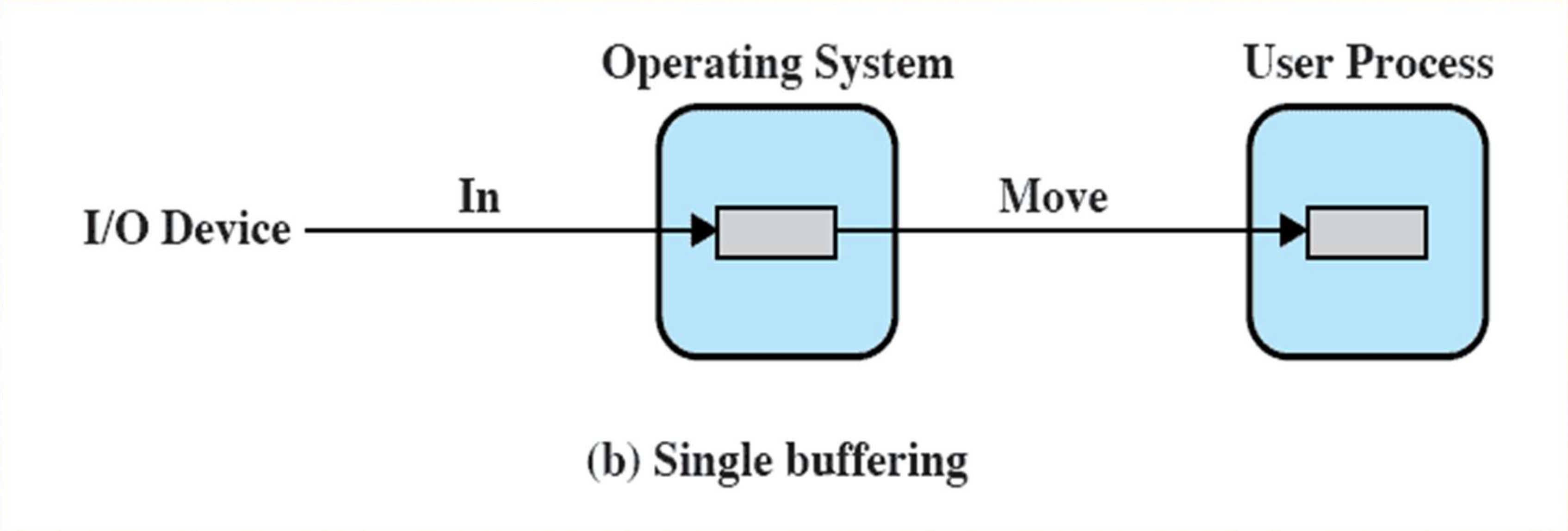


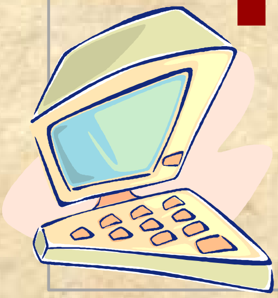

Fig. 11.5

# Block-Oriented Single Buffer

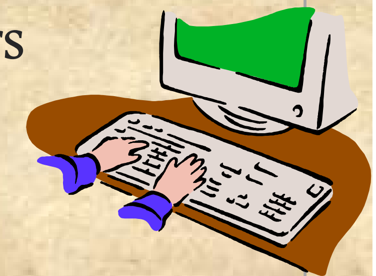
- Input transfers are made to the system buffer
- Reading ahead/anticipated input
  - is done in the expectation that the block will eventually be needed
  - when the transfer is complete, the process moves the block into user space and immediately requests another block
- Generally provides a speedup compared to the lack of system buffering
- Disadvantages:
  - complicates the logic in the operating system
  - swapping logic is also affected

# Stream-Oriented Single Buffer

- Line-at-a-time operation
  - appropriate for scroll-mode terminals (dumb terminals)
  - user input is one line at a time with a carriage return signaling the end of a line
  - output to the terminal is similarly one line at a time



- Byte-at-a-time operation
  - used on forms-mode terminals
  - when each keystroke is significant
  - other 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
- Also known as buffer swapping

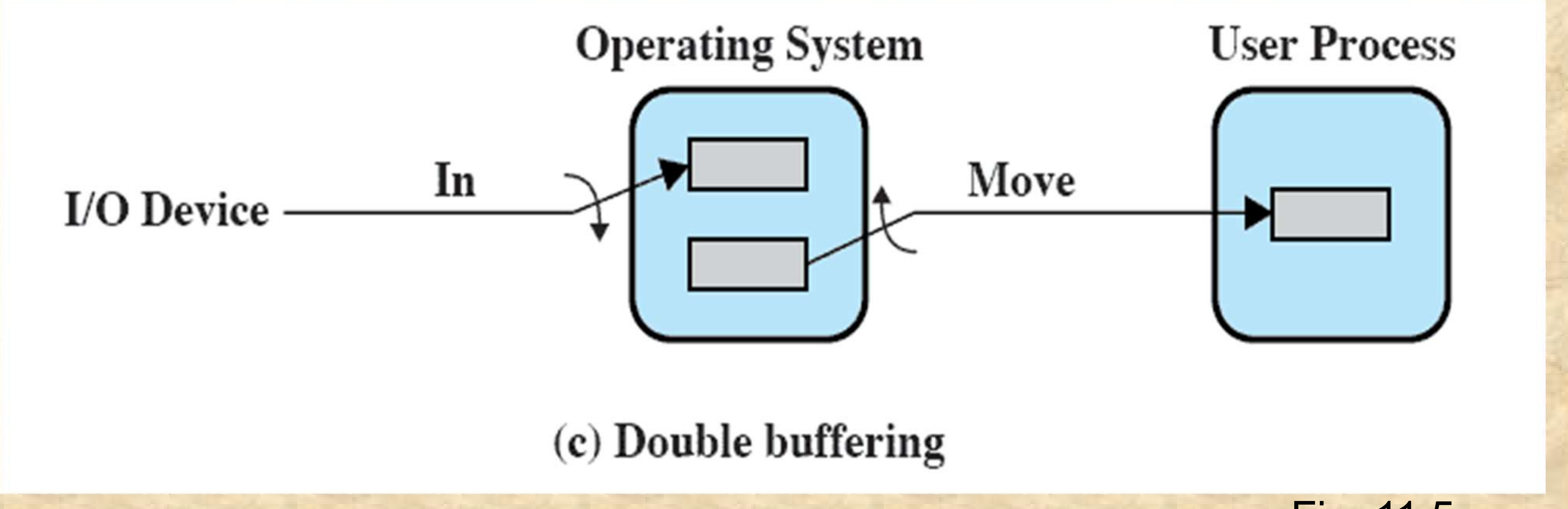
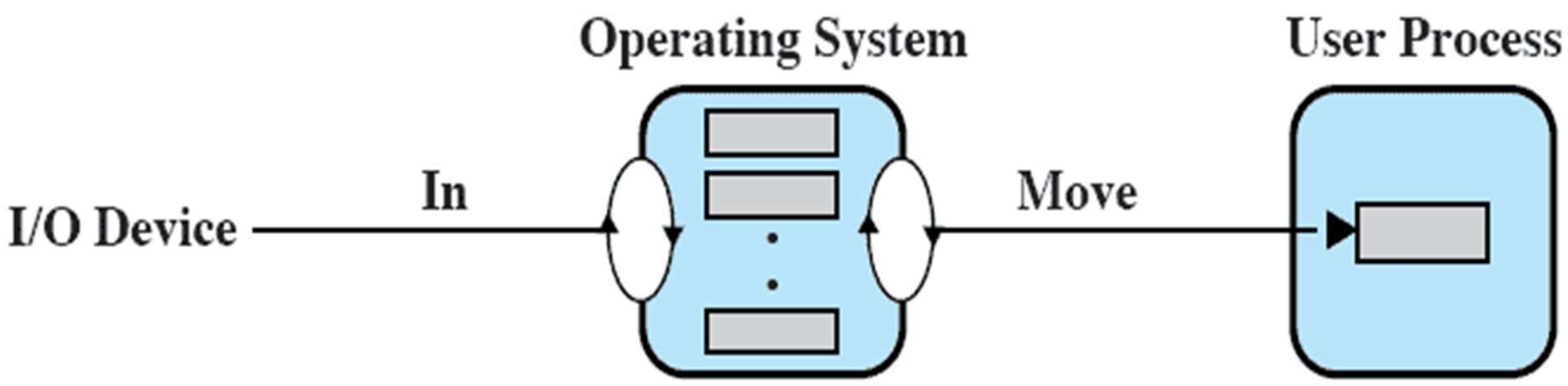



Fig. 11.5

# Circular Buffer

- Two or more 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

Producer-Consumer problem!

Fig. 11.5

# The Utility of Buffering

- Technique that smoothes out peaks in I/O demand
  - with enough demand eventually all buffers become full and their advantage is lost
- When there is a variety of I/O and process activities to service, buffering can increase the efficiency of the OS and the performance of individual processes
- Buffering balances communication between devices or processes of (very) different speed

Discuss

# Disk Performance Parameters

- The actual details of disk I/O operation depend on the:
  - computer system
  - operating system
  - nature of the I/O channel and disk controller hardware

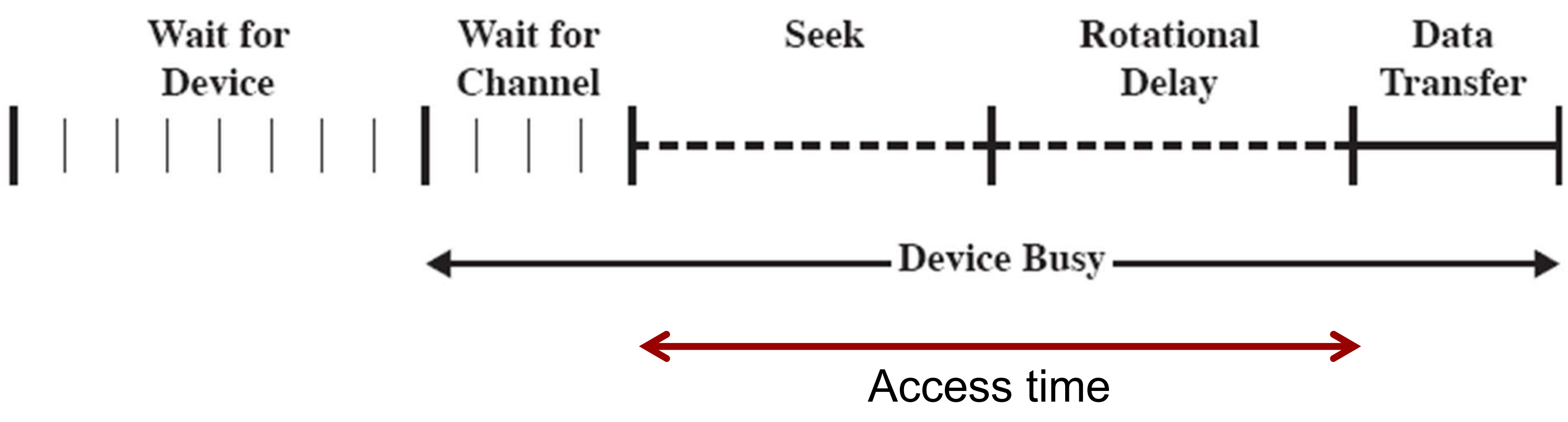



**Figure 11.6 Timing of a Disk I/O Transfer**

# Positioning the Disk Read/Write Heads

- 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
- On a movable-head system the time it takes to position the head at the track is known as **seek time**
- The time it takes for the beginning of the sector to reach the head is known as **rotational delay**
- The sum of the seek time and the rotational delay equals the **access time**

# Disk Scheduling: First-In, First-Out (FIFO)

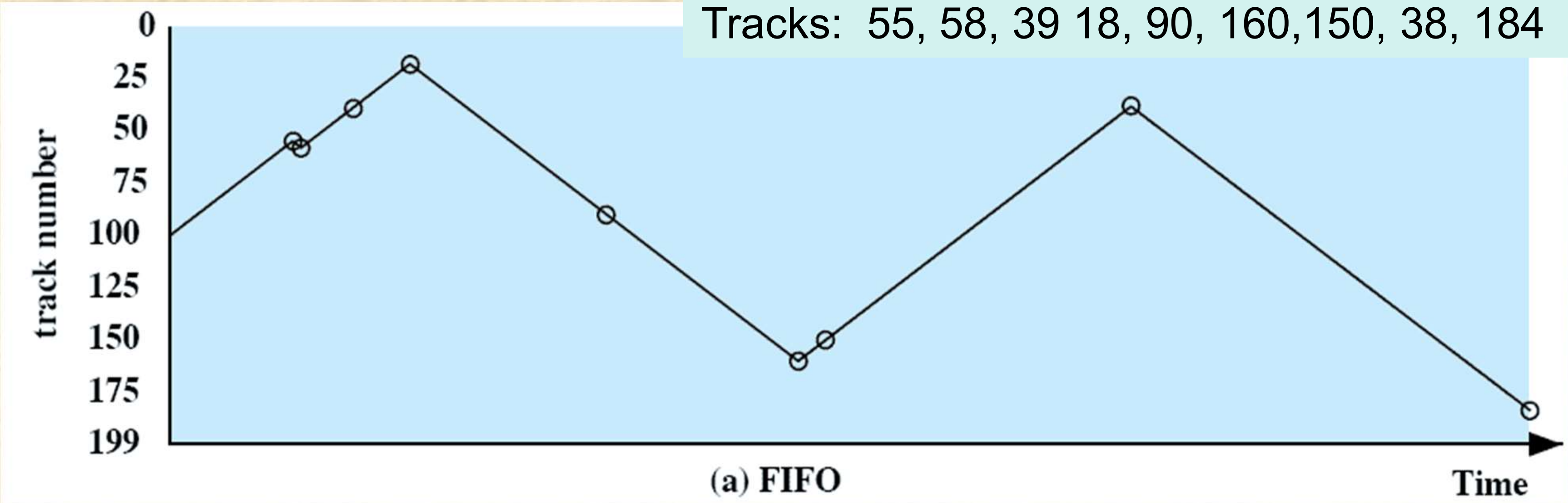
**<u>Disk Scheduling Problem</u>**
When <u>many</u> disk requests are waiting, in <u>which order</u> are they serviced?

- Processes I/O requests in sequential (arrival) order
- Fair to all processes
- Approximates random scheduling in performance if there are many processes competing for the disk

Why random?

What is good disk scheduling?

(Already in queue, arrived in this order)
Tracks:  55, 58, 39 18, 90, 160,150, 38, 184



# Disk Scheduling: Priority (PRI)

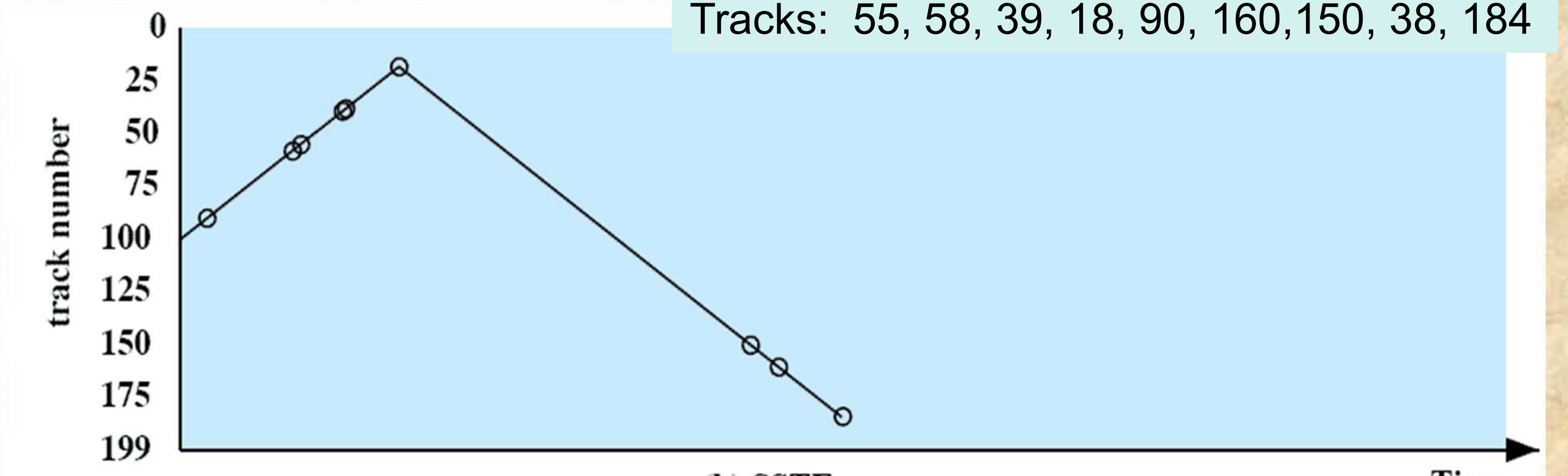
- Control of the scheduling is outside the control of disk management software
- Goal is not to optimize disk utilization but to meet other objectives
- Short batch jobs and interactive jobs are given higher priority
- Provides good interactive response time
- Longer jobs may have to wait an excessively long time
- A poor policy for database systems

# Disk Scheduling: 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
- Favors innermost tracks. Far away track may starve.

Why?

Tracks: 55, 58, 39, 18, 90, 160,150, 38, 184



(b) SSTF

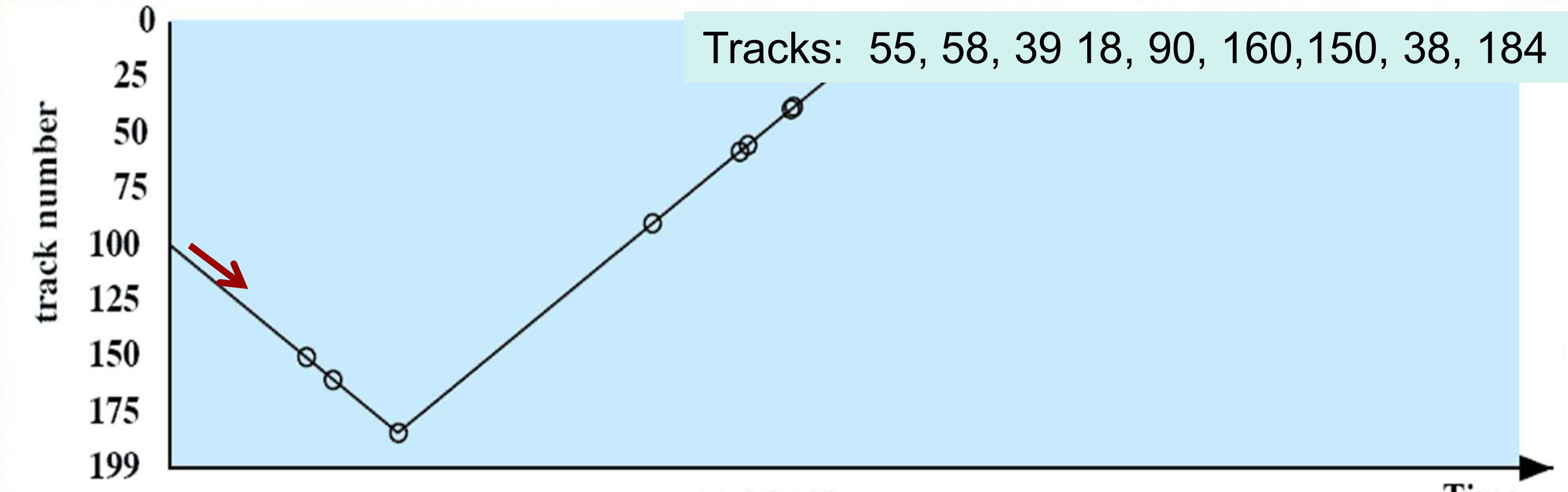
Fig. 11.7

# Disk Scheduling: SCAN

- Also known as the <u>elevator algorithm</u> hissi-algoritmi
- Arm moves up until all tracks serviced that direction, then down, etc.
  - satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed (sometimes called "LOOK" variant)
- Favors jobs whose requests are for tracks clustered nearest to both innermost and outermost tracks, and latest arriving jobs

Why?

Tracks: 55, 58, 39 18, 90, 160,150, 38, 184




(c) SCAN

Fig. 11.7

# Disk Scheduling: C-SCAN (Circular SCAN)

- Restricts scanning to one direction only (up or down)
- 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
- Reduces maximum delay

Why?

Tracks: 55, 58, 39 18, 90, 160,150, 38, 184

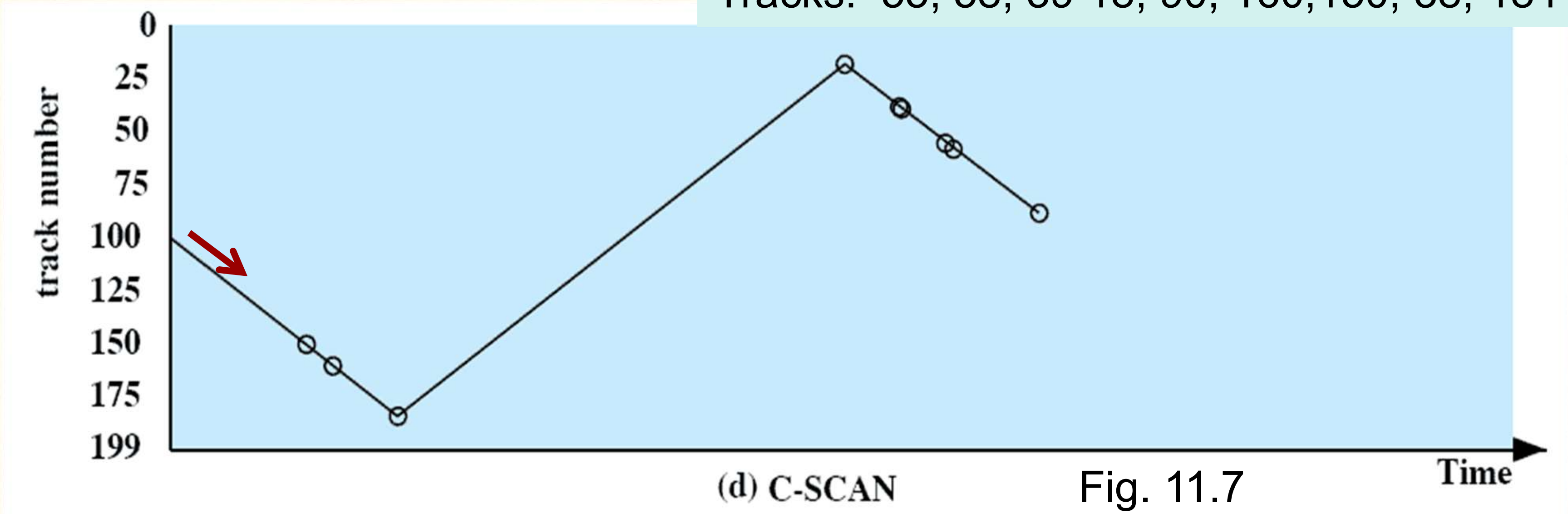



Fig. 11.7

# Disk Scheduling: N-Step-SCAN

- Segments the disk request queue into subqueues 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, all of them are processed with the next scan

More predictable service time?

"fixed size FIFO batches with SCAN"?

# Disk Scheduling: 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

More predictable service time?

"two FIFO batches with SCAN"?

(~random)

| (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 |

Table 11.2 Comparison of 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 |

Table 11.3 Disk Scheduling Algorithms

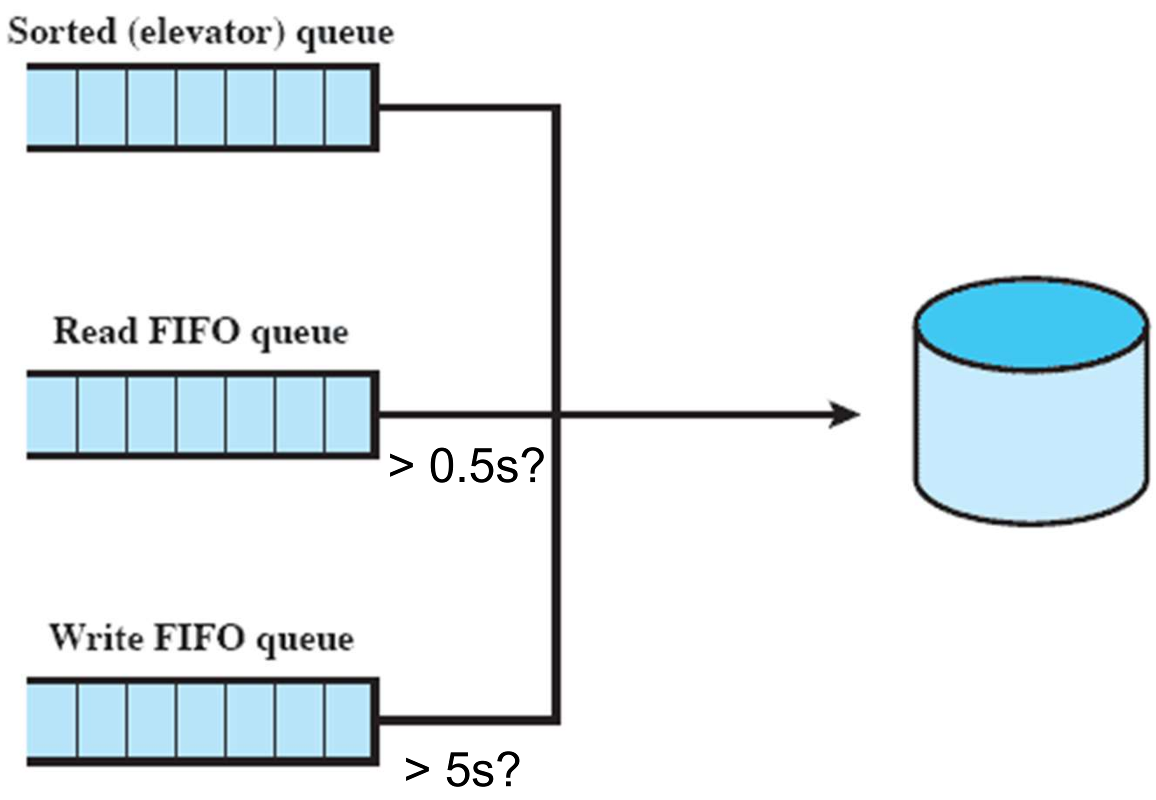
Discuss

# Linus Elevator (Linux I/O Scheduler)

- Very similar to other UNIX implementation
- Associates a special file with each I/O device driver
- Block, character, and network devices are recognized
- Default disk scheduler in Linux 2.4
  - Same or adjacent sector of pending request? → merge
  - Some request already old? → new request put last
    - Do not slow down pending requests too much
  - Good location on this direction → place it there
  - o/w → new request put last
- Optimizations: deadline & anticipatory scheduling

# Linux Deadline Scheduler

- Uses three queues:
  - incoming requests
  - read requests go also to the tail of a Read FIFO queue
  - write requests go also to the tail of a Write FIFO queue
- Each request has an expiration time
  - Get priority if time expires




If request delayed long, it gets priority even though overall performance suffers!

**Fig 11.14 [Sta 15]** The Linux Deadline I/O Scheduler

# Linux Anticipatory I/O Scheduler (for rotating disks)

- Elevator and deadline scheduling can be counterproductive if there are numerous synchronous read requests
- Is superimposed on the deadline scheduler
- When a read request is dispatched, the anticipatory scheduler causes the scheduling system to delay (e.g. 6 ms)
  - there is a good chance that the application that issued the last read request will issue another read request to the same region of the disk
    - that request will be serviced immediately
    - otherwise the scheduler resumes using the deadline scheduling algorithm
  - 8x speedup for large file reads?

# RAID

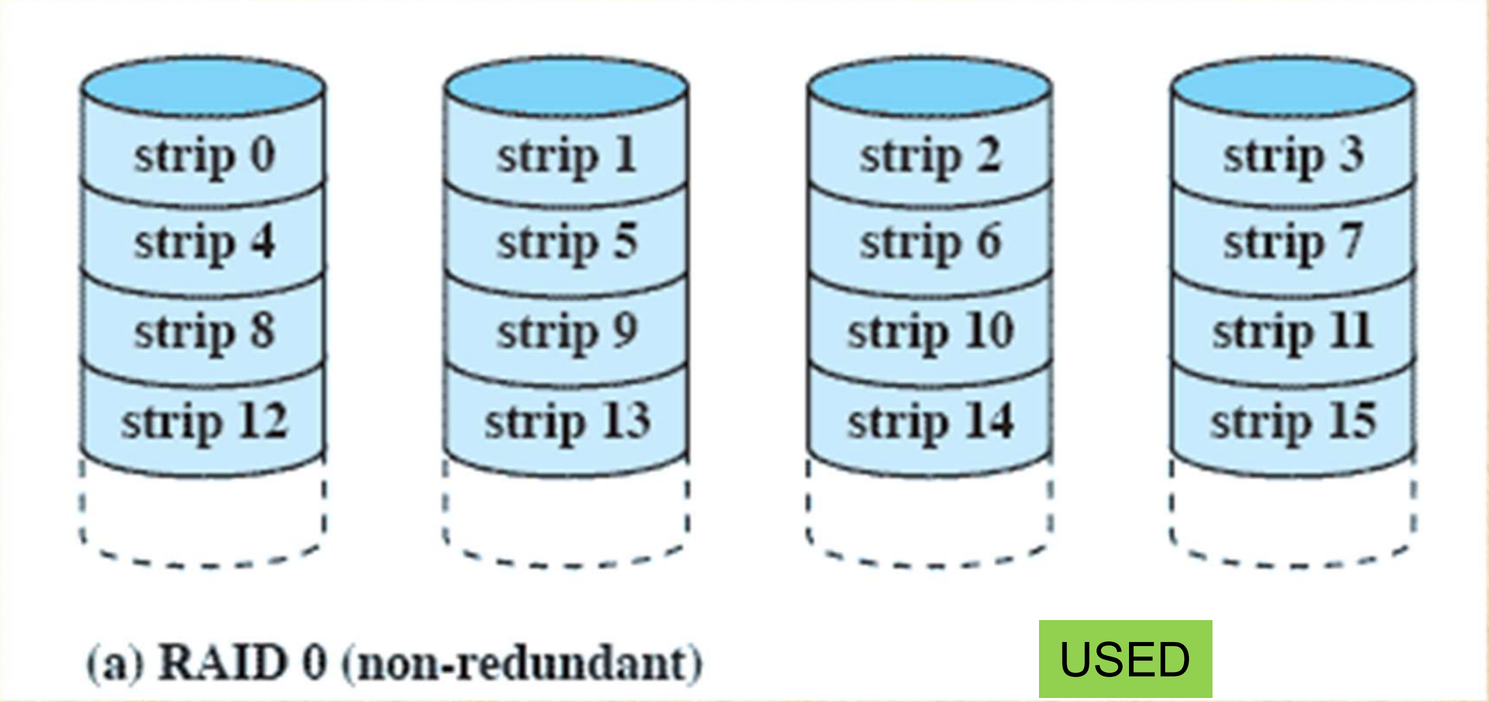
- **R**edundant **A**rray of **I**ndependent **D**isks (~~Redundant Array of Inexpensive Disks~~)
- Consists of seven levels, zero through six

Design architectures share three characteristics:

- RAID is a set of physical disk drives viewed by the operating system as a single logical drive
- data are distributed across the physical drives of an array in a scheme known as striping
- redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure

# RAID Level 0

- Not a true RAID because it has no redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
  - Concurrency gives speed
- Logical disk is divided into (large) strips



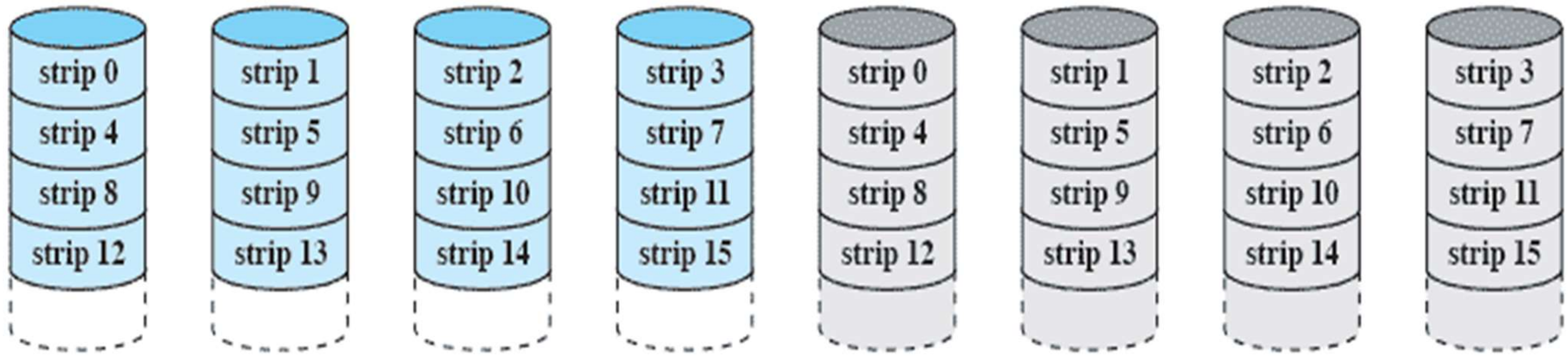

USED

Fig. 11.8

# RAID Level 1

Usually:
RAID 1: No stripes, mirroring
RAID 10: Stripes, mirroring

- Redundancy is achieved by the simple expedient of duplicating all the data
- There is no "write penalty"
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost
  - Data recovery cost/time?



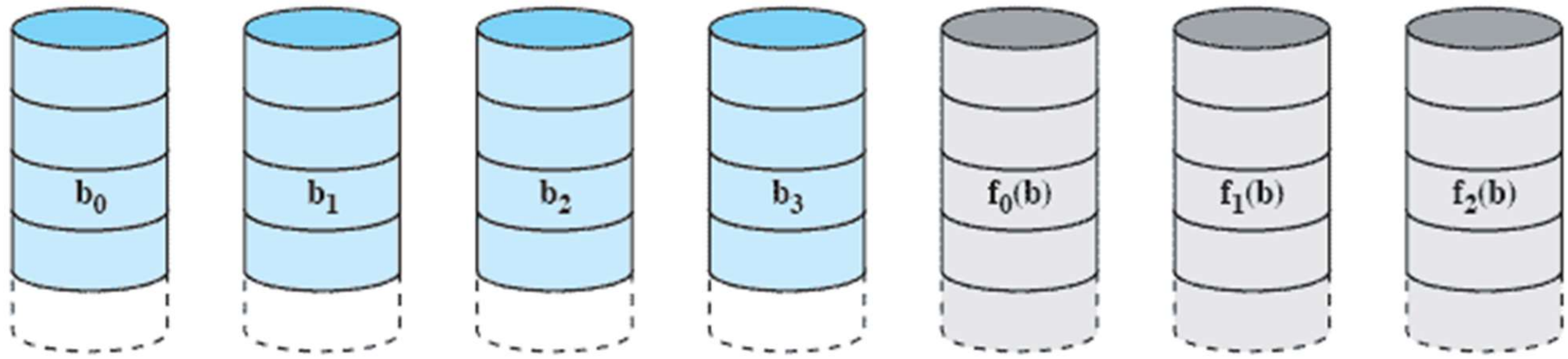

(b) RAID 1 (mirrored)

USED

Fig. 11.8

# RAID Level 2

- Makes use of a parallel access technique
- Data striping is used (byte or word size)
- Typically a Hamming code is used
- Effective choice in an environment in which many disk errors occur



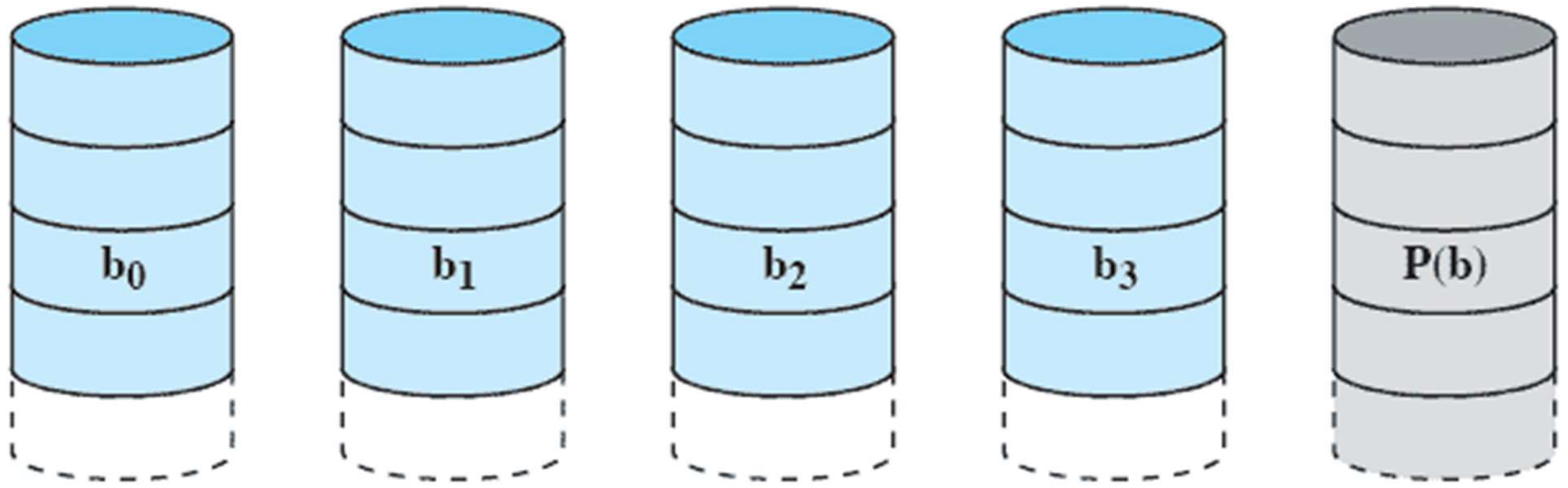

(c) RAID 2 (redundancy through Hamming code)

NOT USED

Fig. 11.8

# RAID Level 3

- Requires only a single redundant disk, no matter how large the disk array
  - Uses just one parity bit
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates

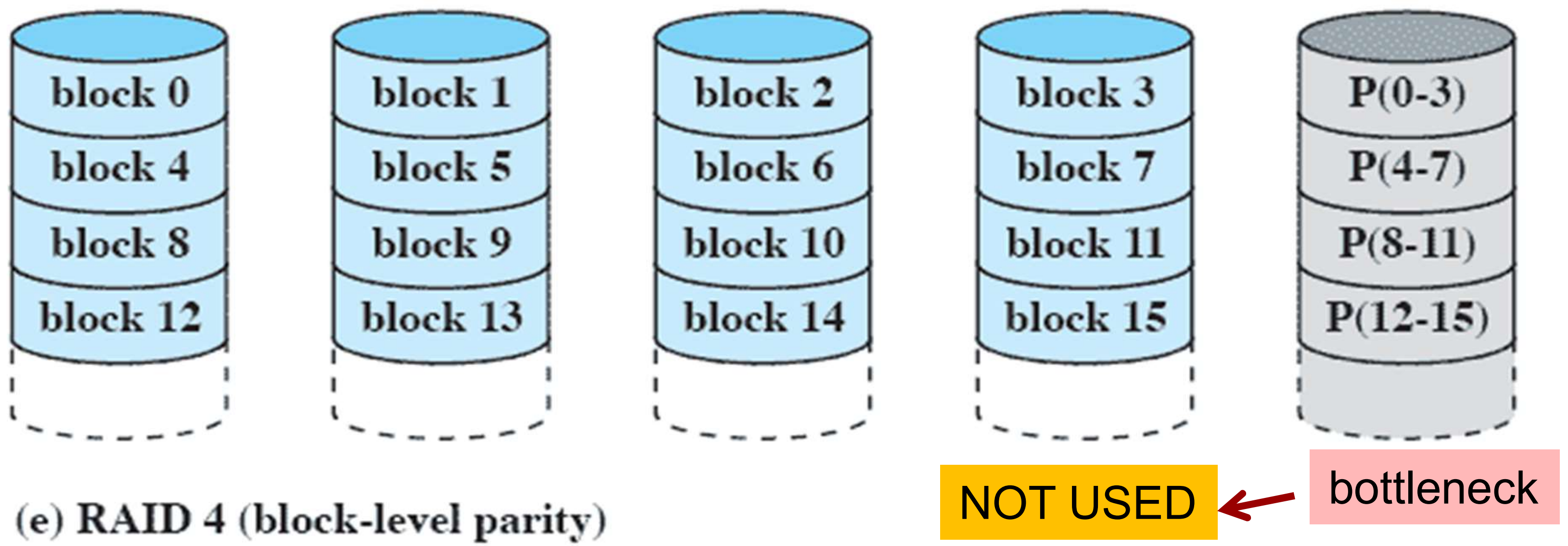



(d) RAID 3 (bit-interleaved parity)

Fig. 11.8

# RAID Level 4

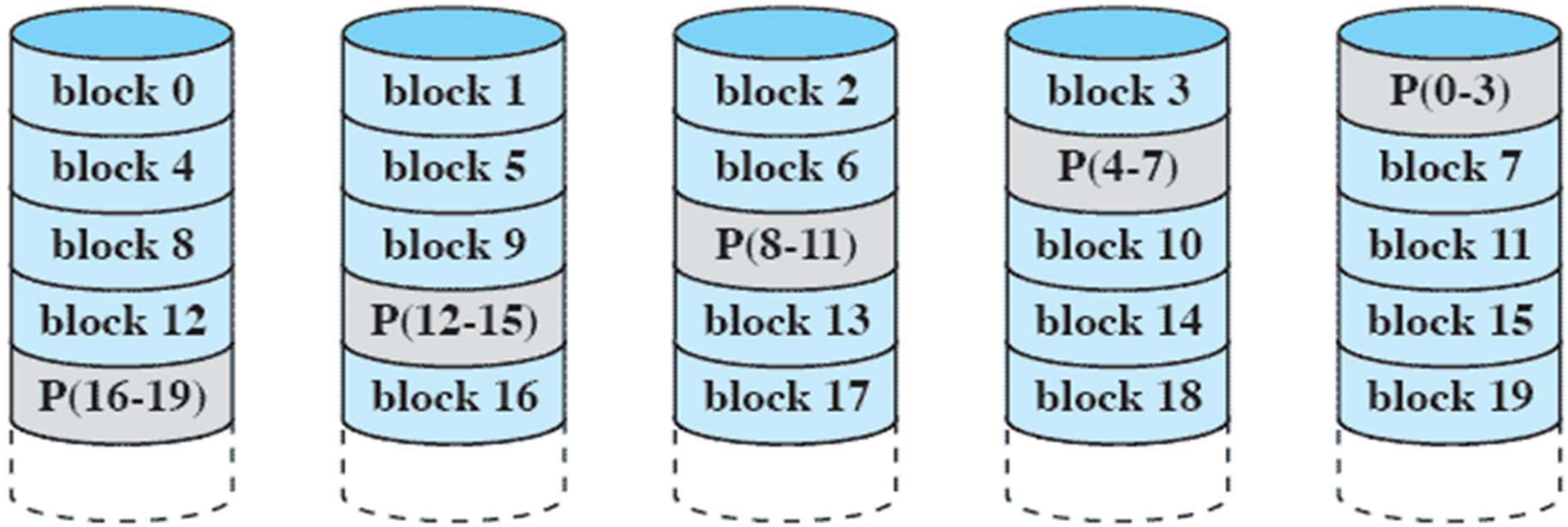
- Makes use of an <u>independent access</u> technique (large block size strips)
- A bit-by-bit <u>parity strip</u> is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed




(e) RAID 4 (block-level parity)

# RAID Level 5

- Similar to RAID-4 but distributes the parity bits (blocks) 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
  - High data recovery cost



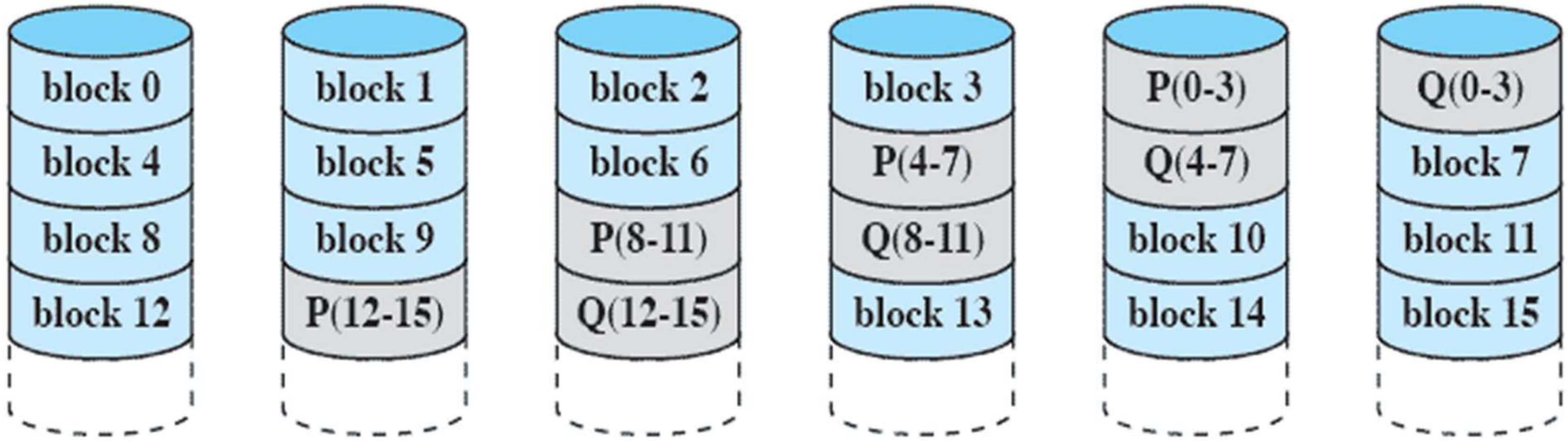

(f) RAID 5 (block-level distributed parity)

USED

Fig. 11.8

# RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
  - Can sustain 2 disk failures at a time
- Incurs a substantial write penalty because each write affects also two parity blocks
  - High data recovery cost




(g) RAID 6 (dual redundancy)

Fig. 11.8

# Table 11.4 RAID Levels

| Category | Level | Description | Disks required | Data availability | Large I/O data transfer capacity | Small I/O request rate |
|---|---|---|---|---|---|---|
| Striping | 0 | Nonredundant | $N$ | Lower than single disk | Very high | Very high for both read and write |
| Mirroring | 1 | Mirrored | $2N$ | Higher than RAID 2, 3, 4, or 5; lower than RAID 6 | Higher than single disk for read; similar to single disk for write | Up to twice that of a single disk for read; similar to single disk for write |
| Parallel access | 2 | Redundant via Hamming code | $N + m$ | Much higher than single disk; comparable to RAID 3, 4, or 5 | Highest of all listed alternatives | Approximately twice that of a single disk |
| | 3 | Bit-interleaved parity | $N + 1$ | Much higher than single disk; comparable to RAID 2, 4, or 5 | Highest of all listed alternatives | Approximately twice that of a single disk |
| Independent access | 4 | Block-interleaved parity | $N + 1$ | Much higher than single disk; comparable to RAID 2, 3, or 5 | Similar to RAID 0 for read; significantly lower than single disk for write | Similar to RAID 0 for read; significantly lower than single disk for write |
| | 5 | Block-interleaved distributed parity | $N + 1$ | Much higher than single disk; comparable to RAID 2, 3, or 4 | Similar to RAID 0 for read; lower than single disk for write | Similar to RAID 0 for read; generally lower than single disk for write |
| | 6 | Block-interleaved dual distributed parity | $N + 2$ | Highest of all listed alternatives | Similar to RAID 0 for read; lower than RAID 5 for write | Similar to RAID 0 for read; significantly lower than RAID 5 for write |

$N$ = number of data disks; $m$ proportional to log $N$

Discuss

# Windows RAID Configurations

- Windows supports two sorts of RAID configurations:

**Hardware RAID**

separate physical disks combined into one or more logical disks by the disk controller or disk storage cabinet hardware

**Software RAID**

noncontiguous disk space combined into one or more logical partitions by the fault-tolerant software disk driver, FTDISK

# Disk Cache

levyvälimuisti

- *Cache memory* is HW memory that is smaller and faster than main memory and that is interposed between main memory and the processor
- ***Disk cache*** is a buffer in main memory for disk sectors

levyvälimuisti

  - 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 disk cache
- if NO, … → the requested sector is read into the disk cache from the disk

  - ***Disk buffer*** is buffer on hard disk controller (for the same purpose as disk cache)

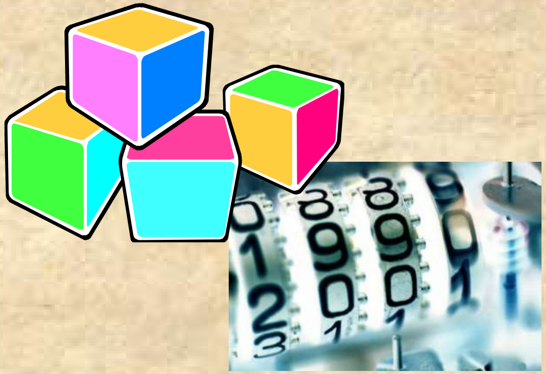
levypuskuri

# Disk Cache Replacement Algorithm: Least Recently Used (LRU)

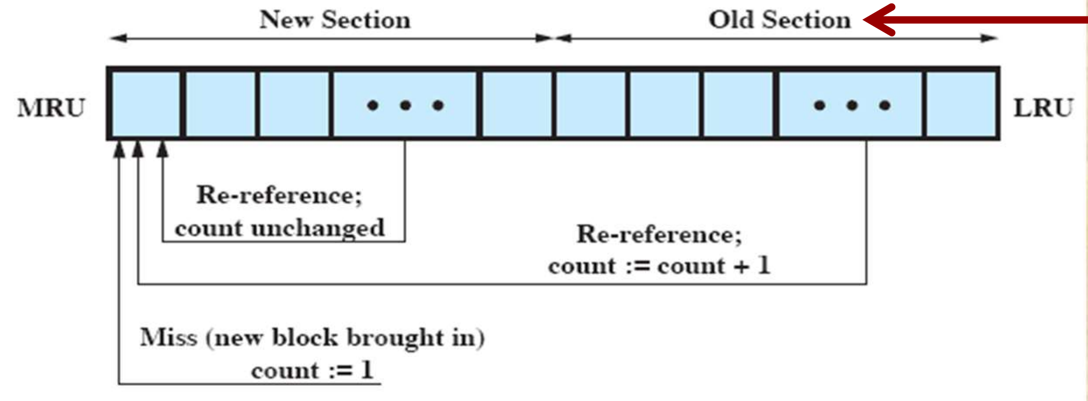
- Most commonly used algorithm that deals with the design issue of replacement strategy
- The block that has been in the cache the longest with no reference to it is replaced
- A stack of pointers reference the cache
  - most recently referenced block is on the top of the stack
  - when a block is referenced or brought into the cache, it is placed on the top of the stack

# Disk Cache Replacement Algorithm: Least Frequently Used (LFU)

- 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

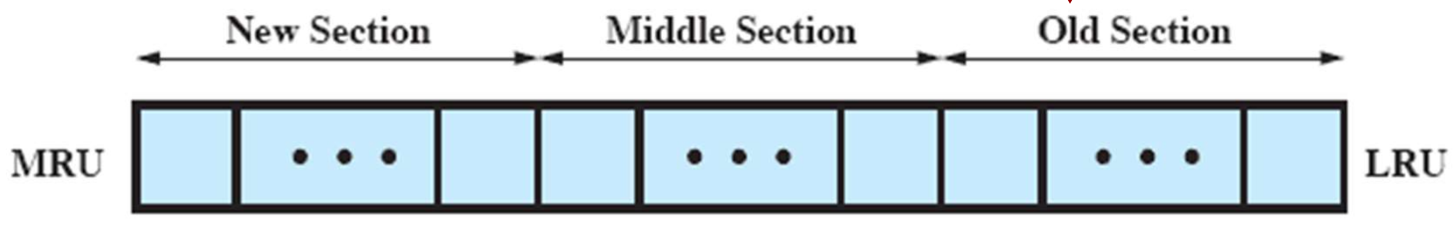


# Frequency-Based Replacement




(a) FIFO

Replace from here
(using smallest reference count)




(b) Use of three sections

Fig. 11.9

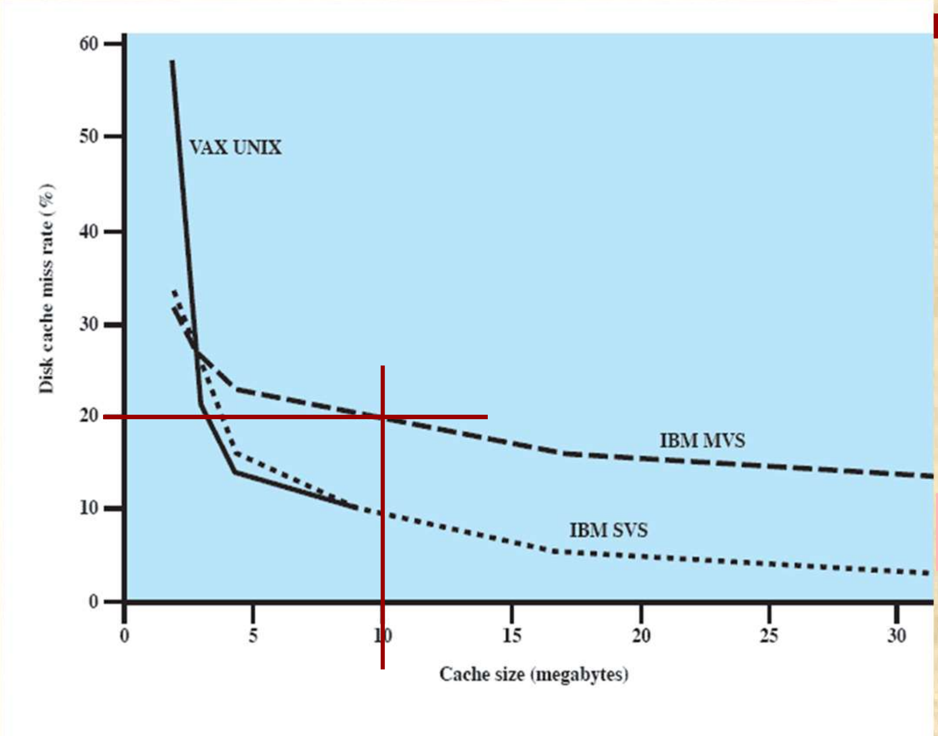



Figure 11.10 Some Disk Cache Performance Results Using LRU

# Frequency-Based Replacement

Note: different data sets

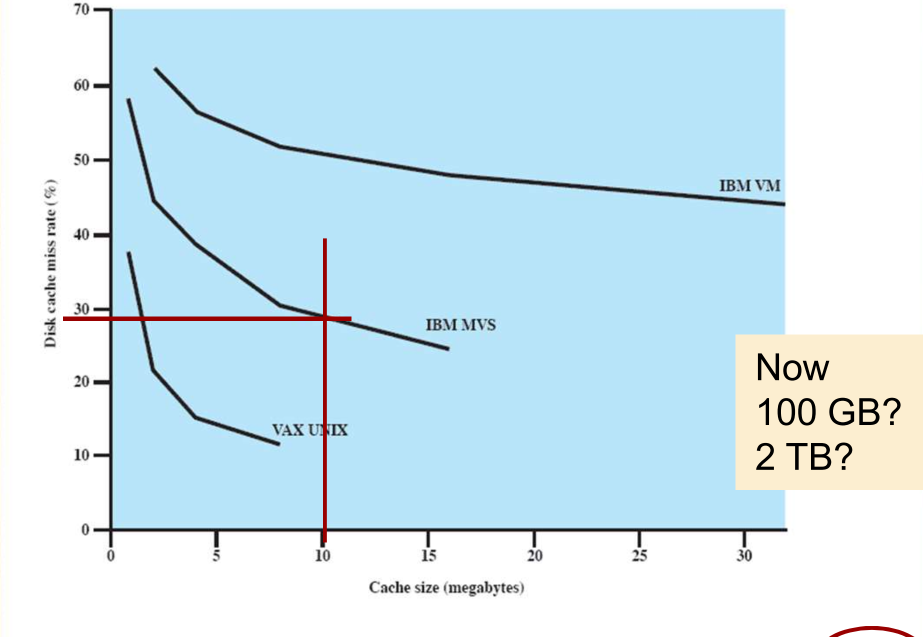



Figure 11.11 Disk Cache Performance Using Frequency-Based Replacement [ROBI90]

Now
100 GB?
2 TB?

# LRU Disk Cache Performance

Discuss

# Summary

- I/O architecture is the computer system's interface to the outside world
- I/O functions are generally broken up into a number of layers
- A key aspect of I/O is the use of buffers that are controlled by I/O utilities rather than by application processes
- Buffering smoothes out the differences between the speeds
- The use of buffers also decouples the actual I/O transfer from the address space of the application process
- Disk I/O has the greatest impact on overall system performance
- Two of the most widely used approaches are disk scheduling and the disk cache
- A disk cache is a buffer, usually kept in main memory, that functions as a cache of disk block between disk memory and the rest of main memory