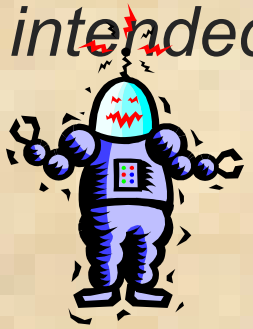
Operating Systems: Internals and Design Principles

Chapter 11 I/O Management and Disk Scheduling

Seventh Edition
By William Stallings

Operating Systems: Internals and Design Principles

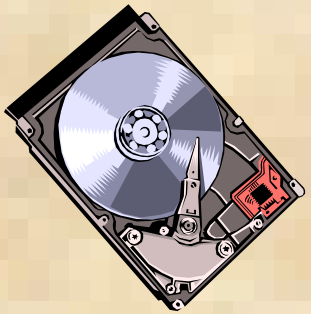
An artifact can be thought of as a meeting point—an "interface" in today's terms between an "inner" environment, the substance and organization of the artifact itself, and an "outer" environment, the surroundings in which it operates. If the inner environment is appropriate to the outer environment, or vice versa, the artifact will serve its intended purpose.



— THE SCIENCES OF THE ARTIFICIAL, Herbert Simon

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

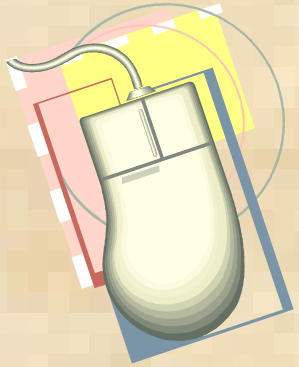
- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

Communication

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

Differences in I/O Devices

- Devices differ in a number of areas:



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

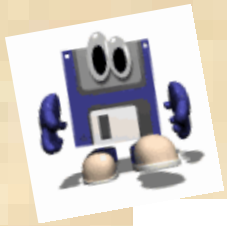
- data may be transferred as a stream of bytes or characters or in larger blocks

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

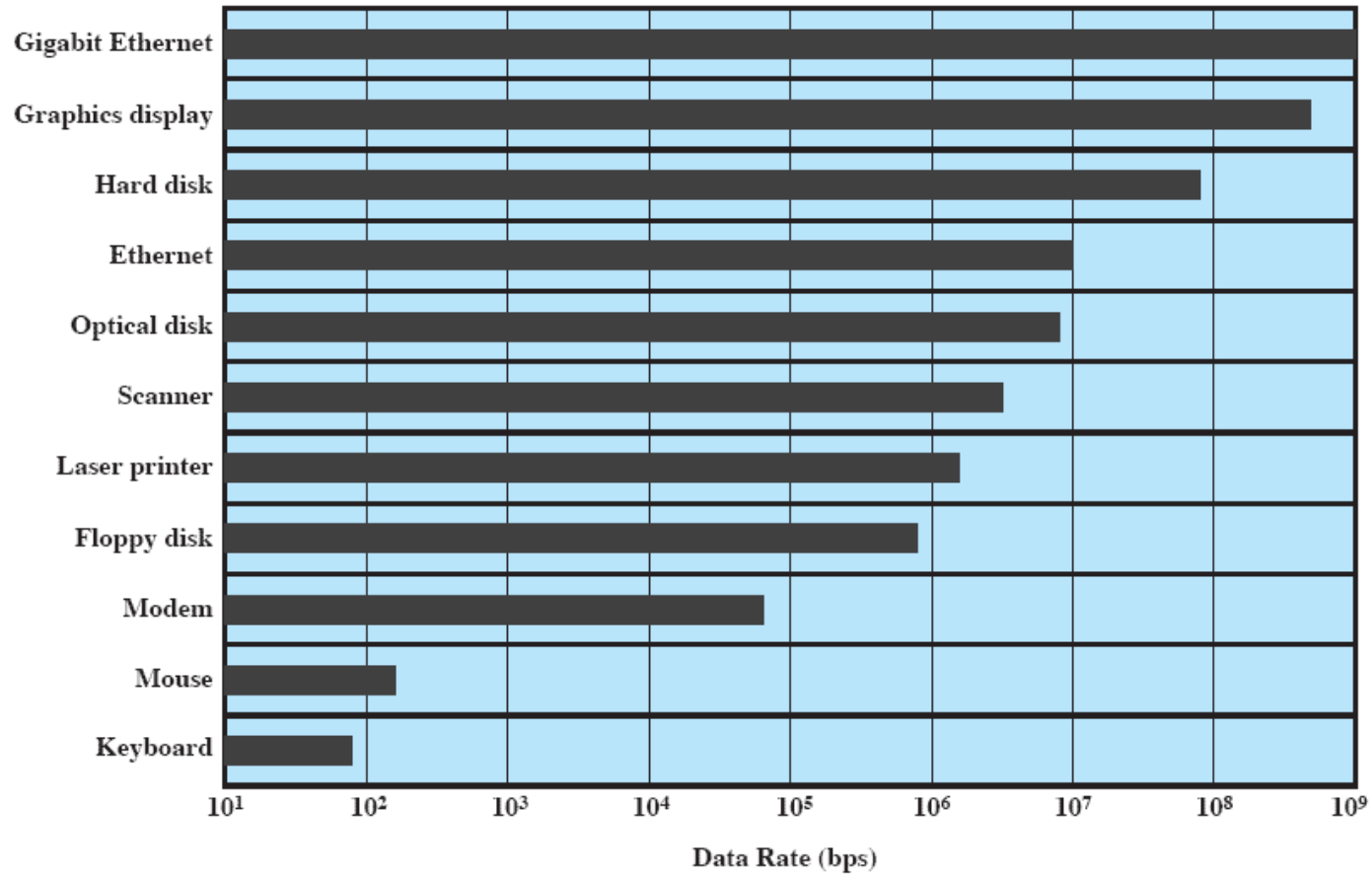


Figure 11.1 Typical I/O Device Data Rates

Organization of the 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)**
  - a DMA module controls the exchange of data between main memory and an I/O module

Techniques for Performing I/O

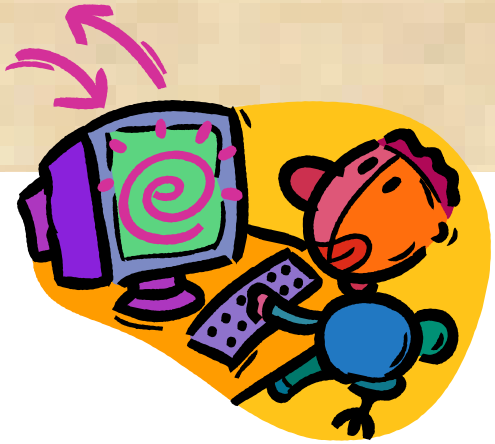


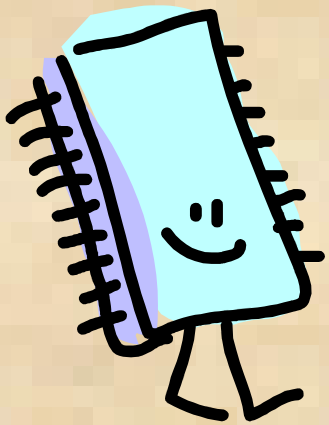
Table 11.1 I/O Techniques

	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)

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 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

Direct Memory Access



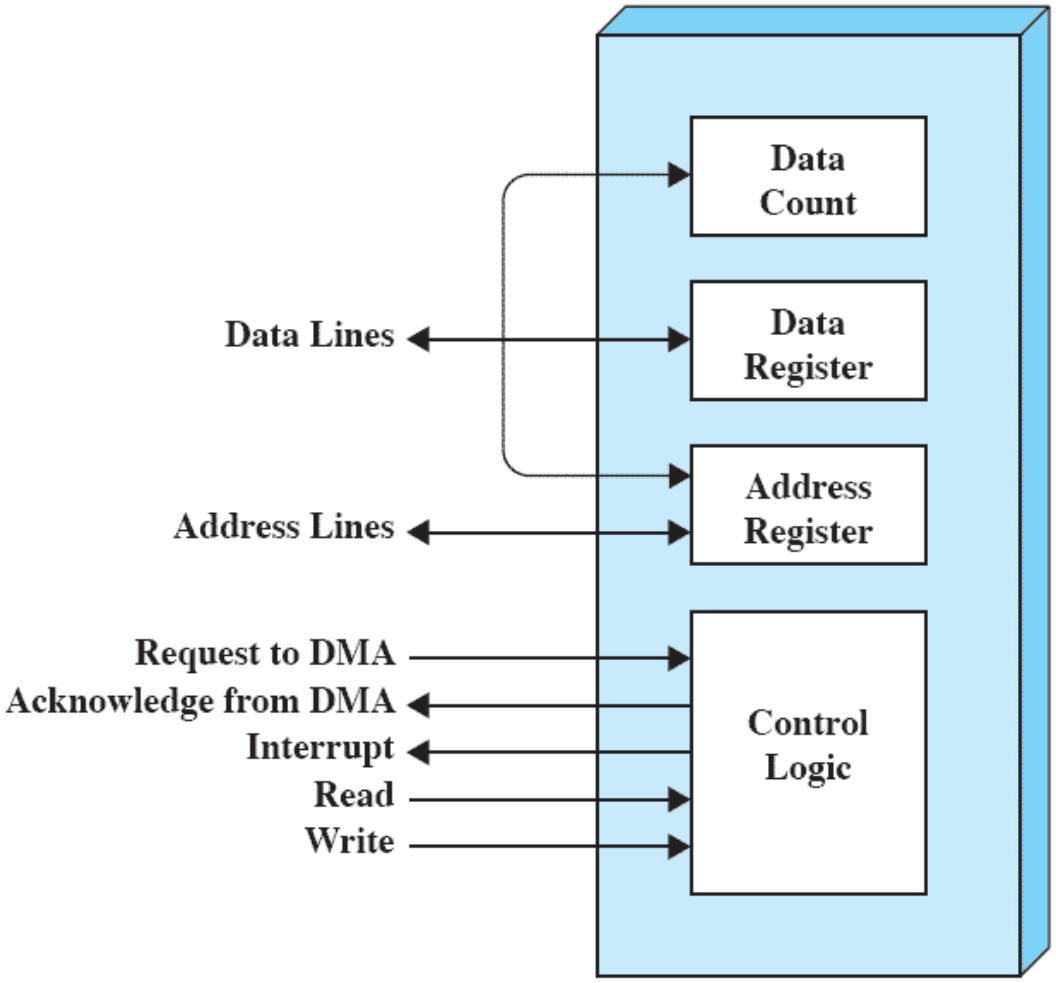
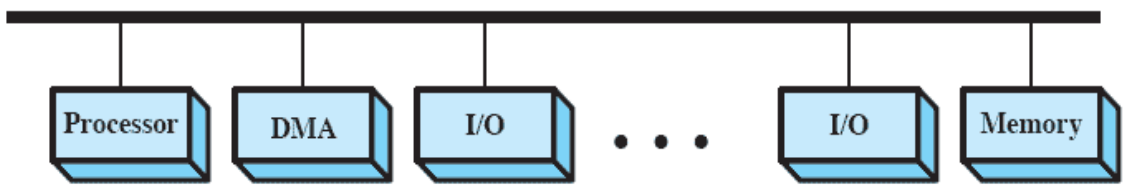


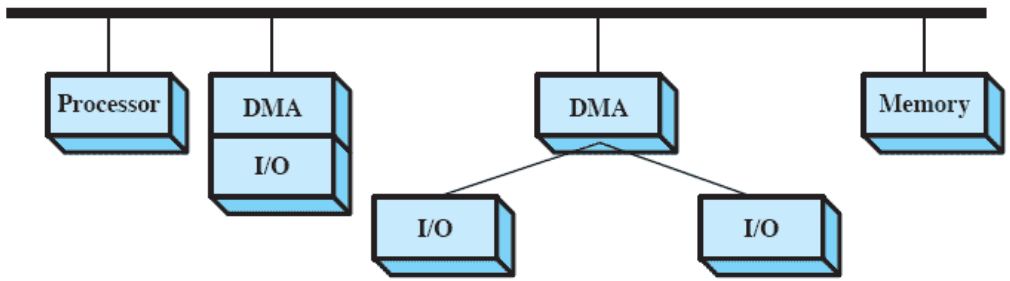
Figure 11.2 Typical DMA Block Diagram

DMA

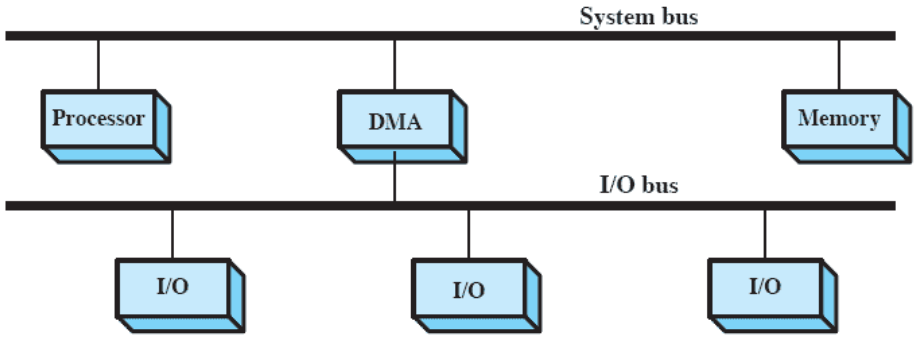


(a) Single-bus, detached DMA

Alternative



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



(c) I/O bus

Configurations

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

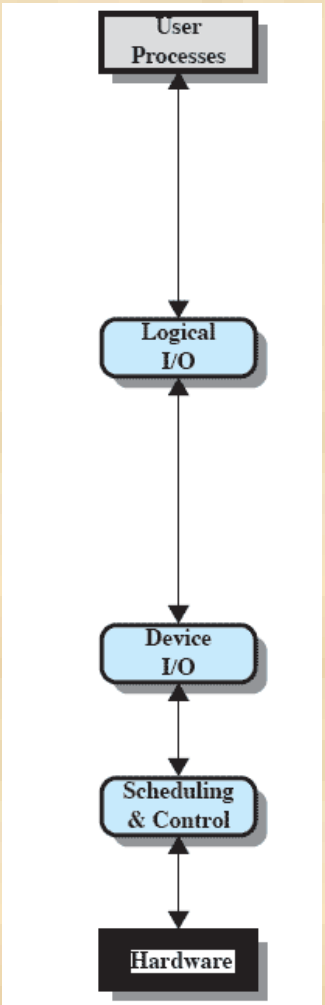
Generality

- 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

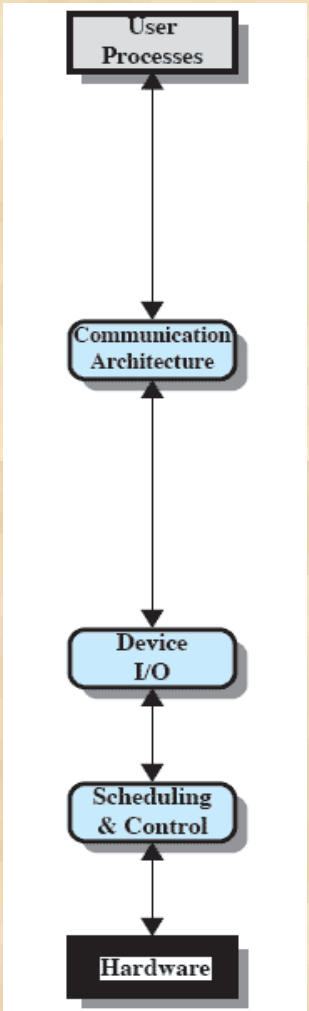
Hierarchical Design

- 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

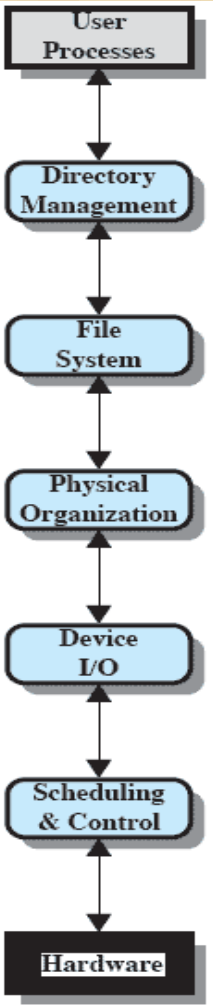
A Model of I/O Organization



(a) Local peripheral device



(b) Communications port



(c) File system

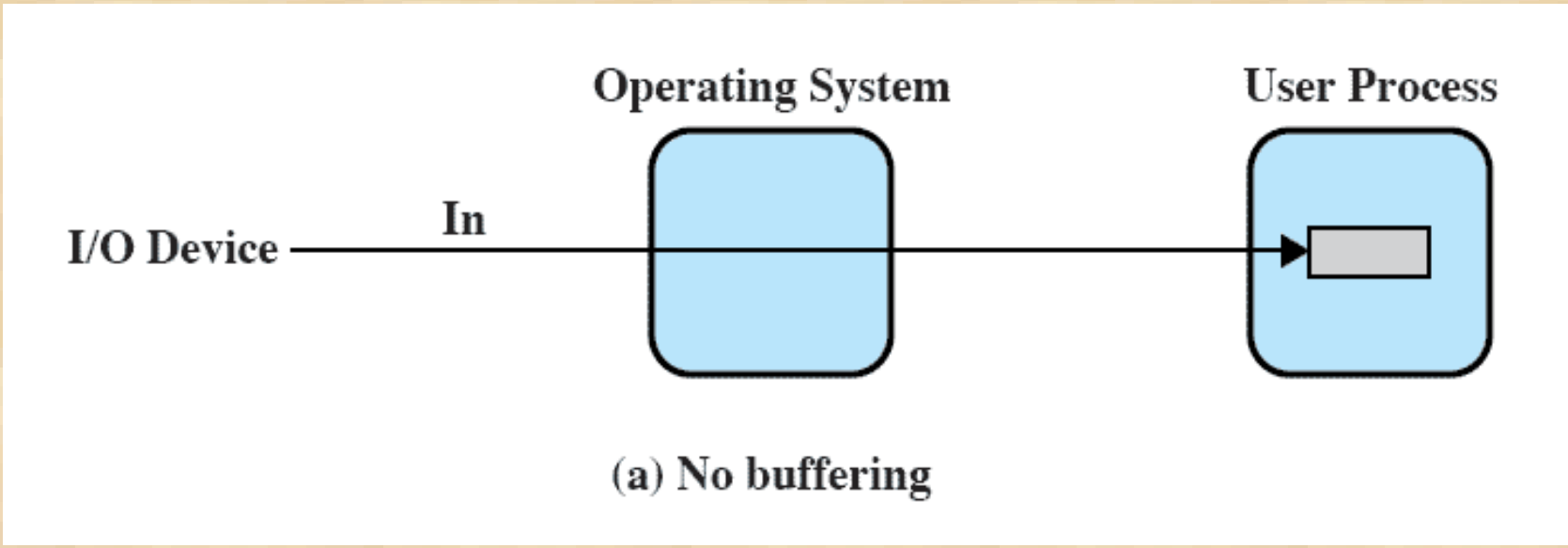
Buffering

- 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 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 keys are examples	• transfers data in and out as a stream of bytes • no block structure • terminals, printers, communications ports, and most other devices that are not secondary storage are examples

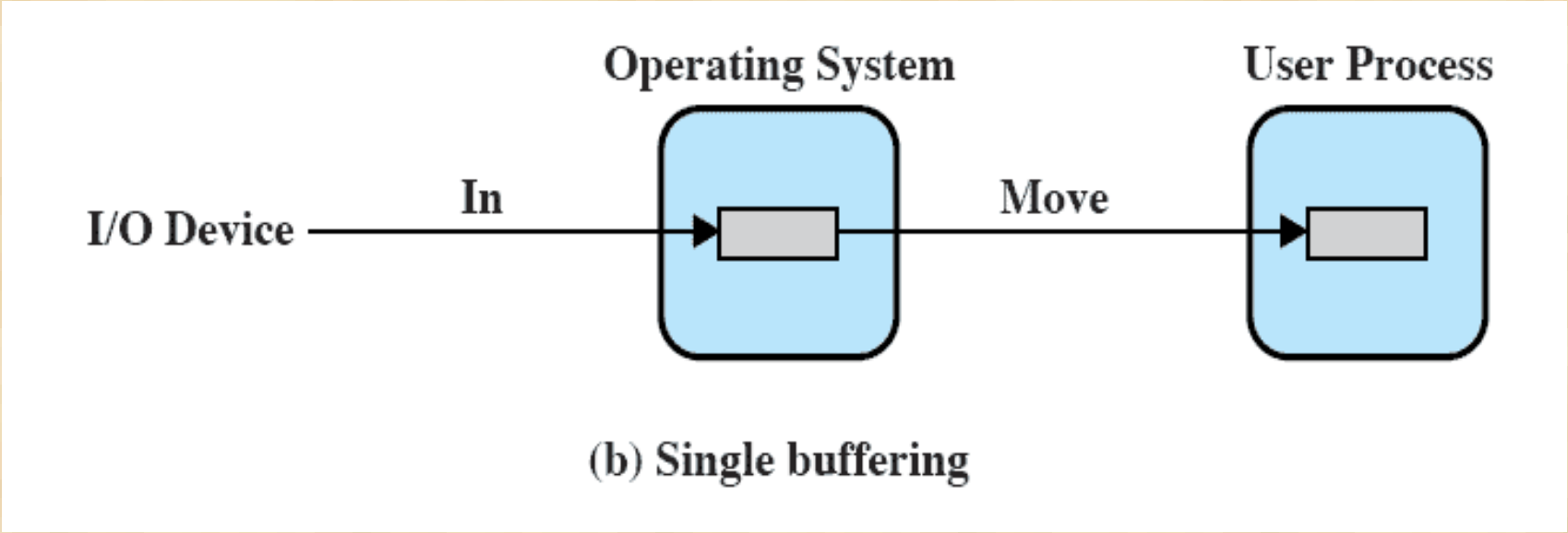
No Buffer

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



Single Buffer

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



(b) Single buffering

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

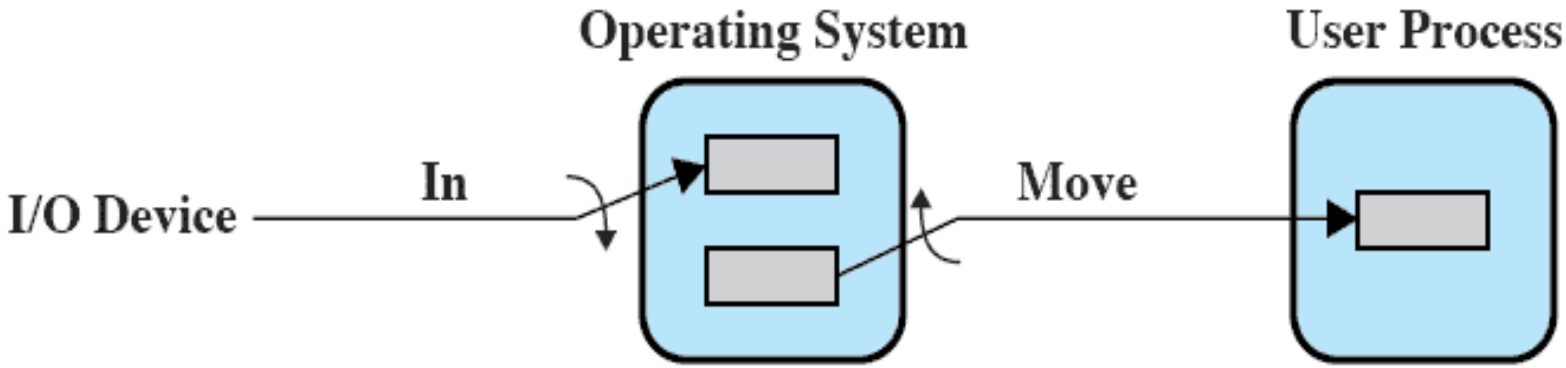
Stream-Oriented Single Buffer

- 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

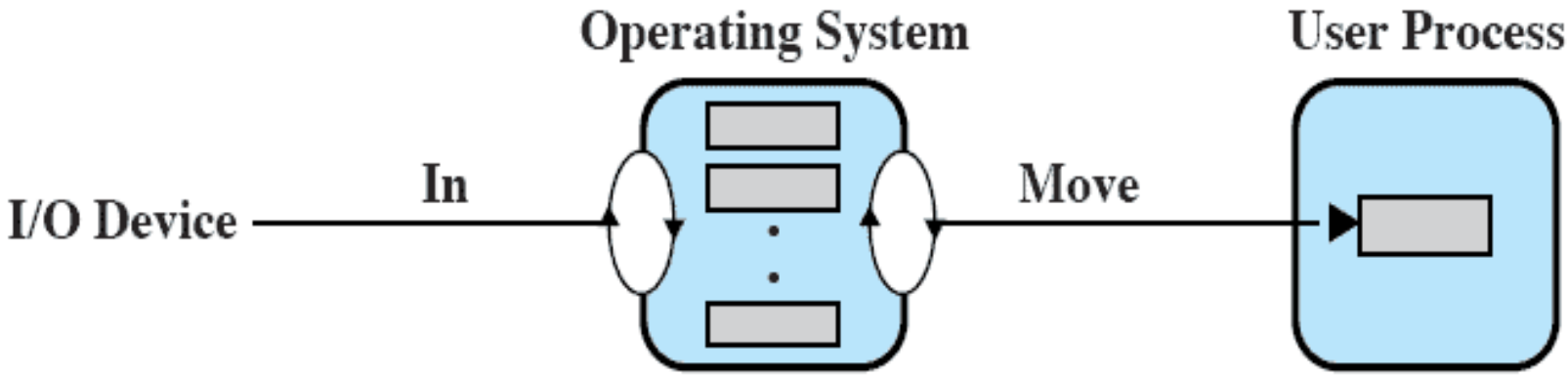
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer
- Also known as buffer swapping



(c) Double buffering

Circular Buffer

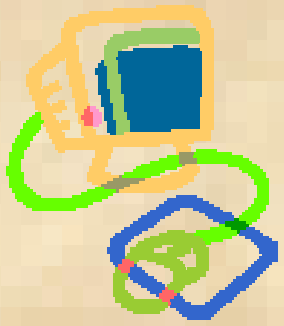
- Two or more buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process



(d) Circular buffering

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



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

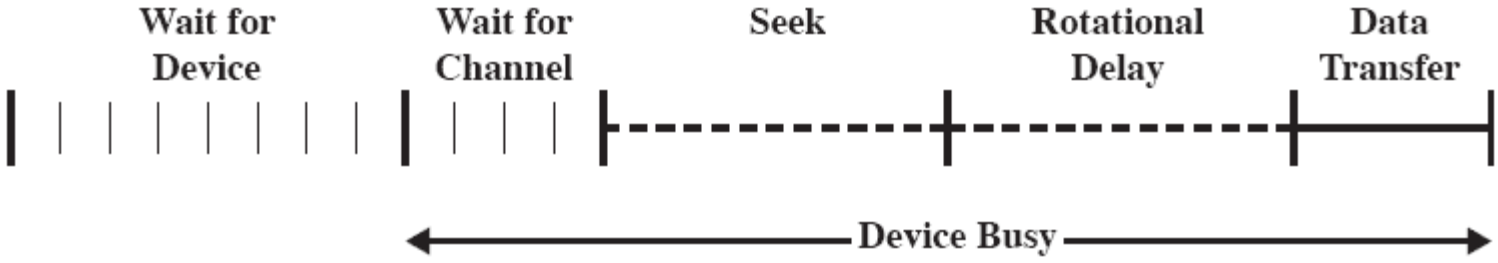


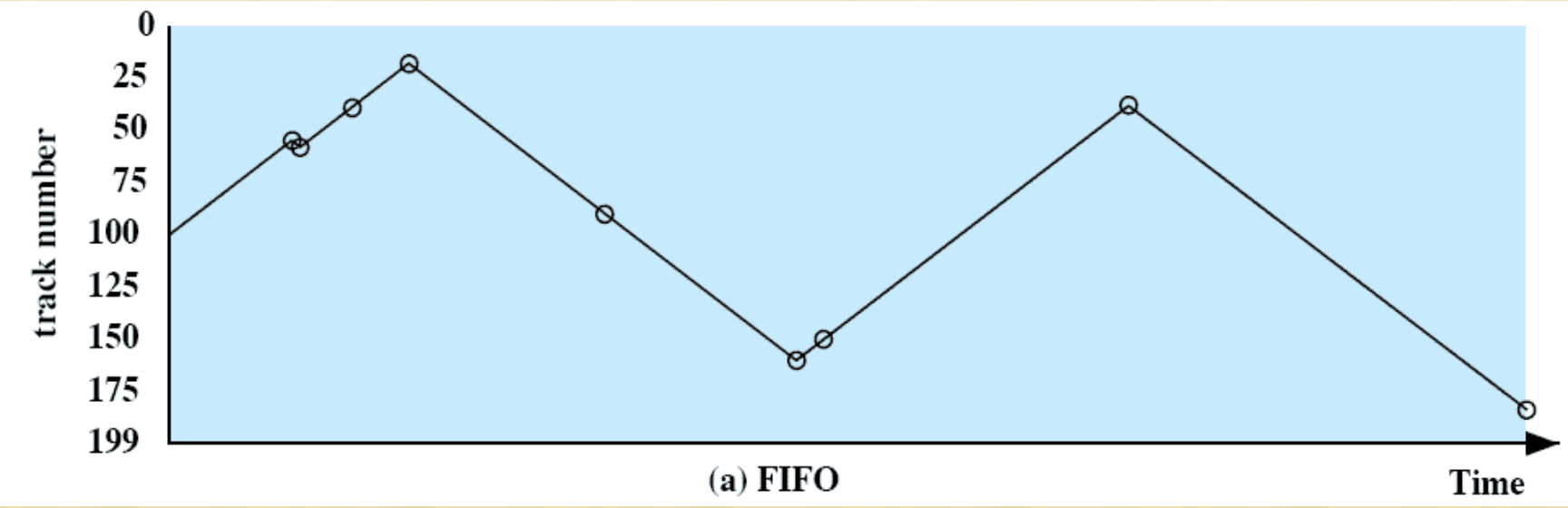
Figure 11.6 Timing of a Disk I/O Transfer

Positioning the 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**

First-In, First-Out (FIFO)

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

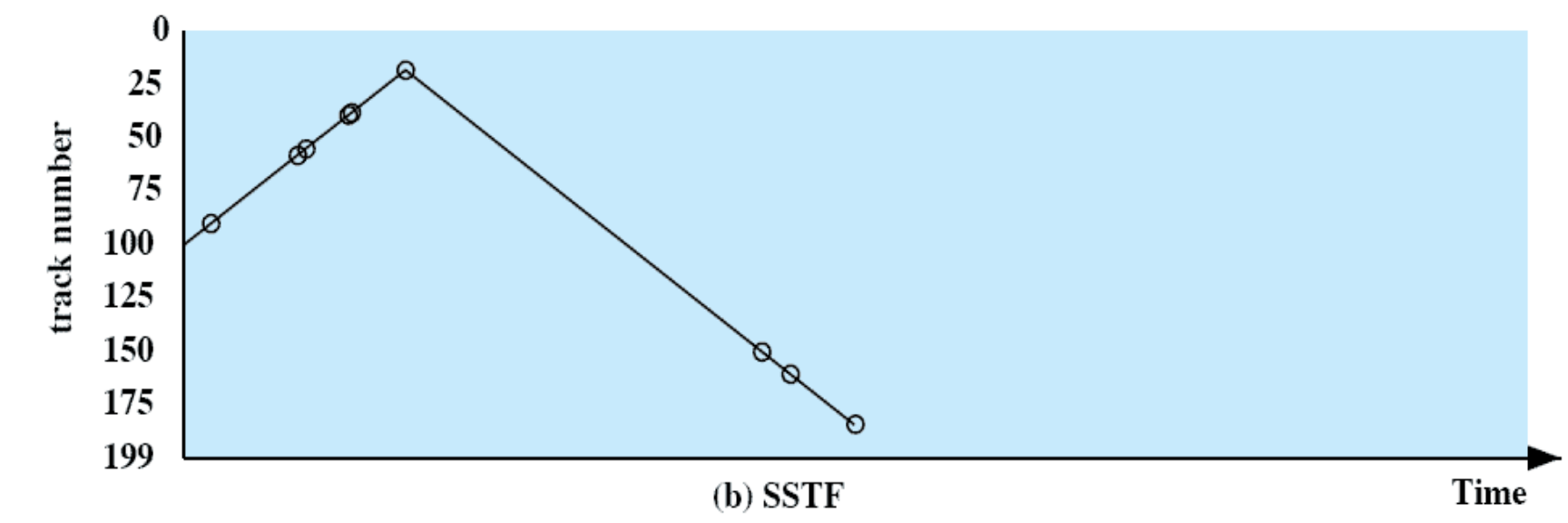


Priority (PRI)

- 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

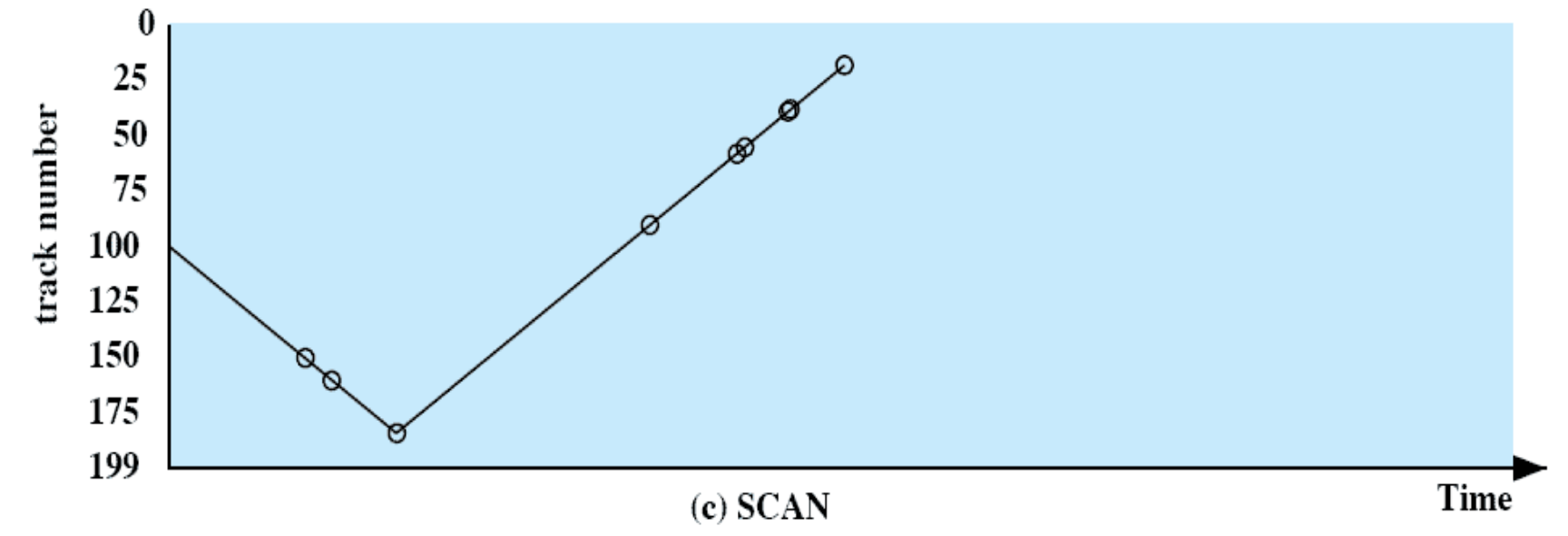
Shortest Service Time First (SSTF)

- 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



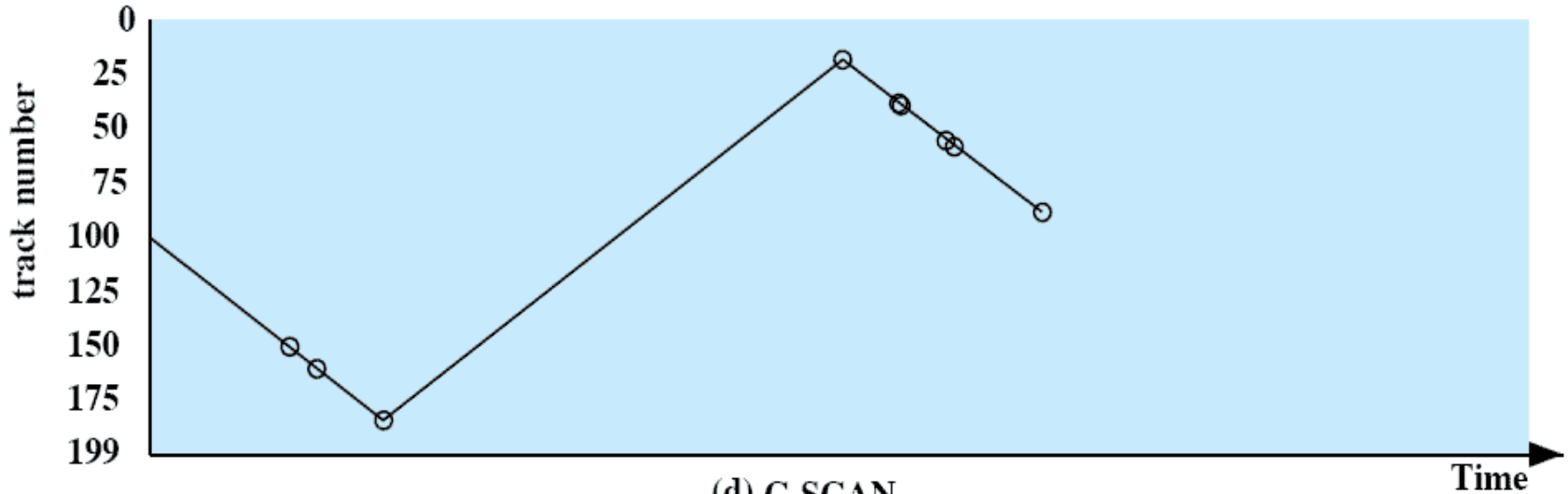
SCAN

- 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



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 begins again



(d) C-SCAN

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

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

(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

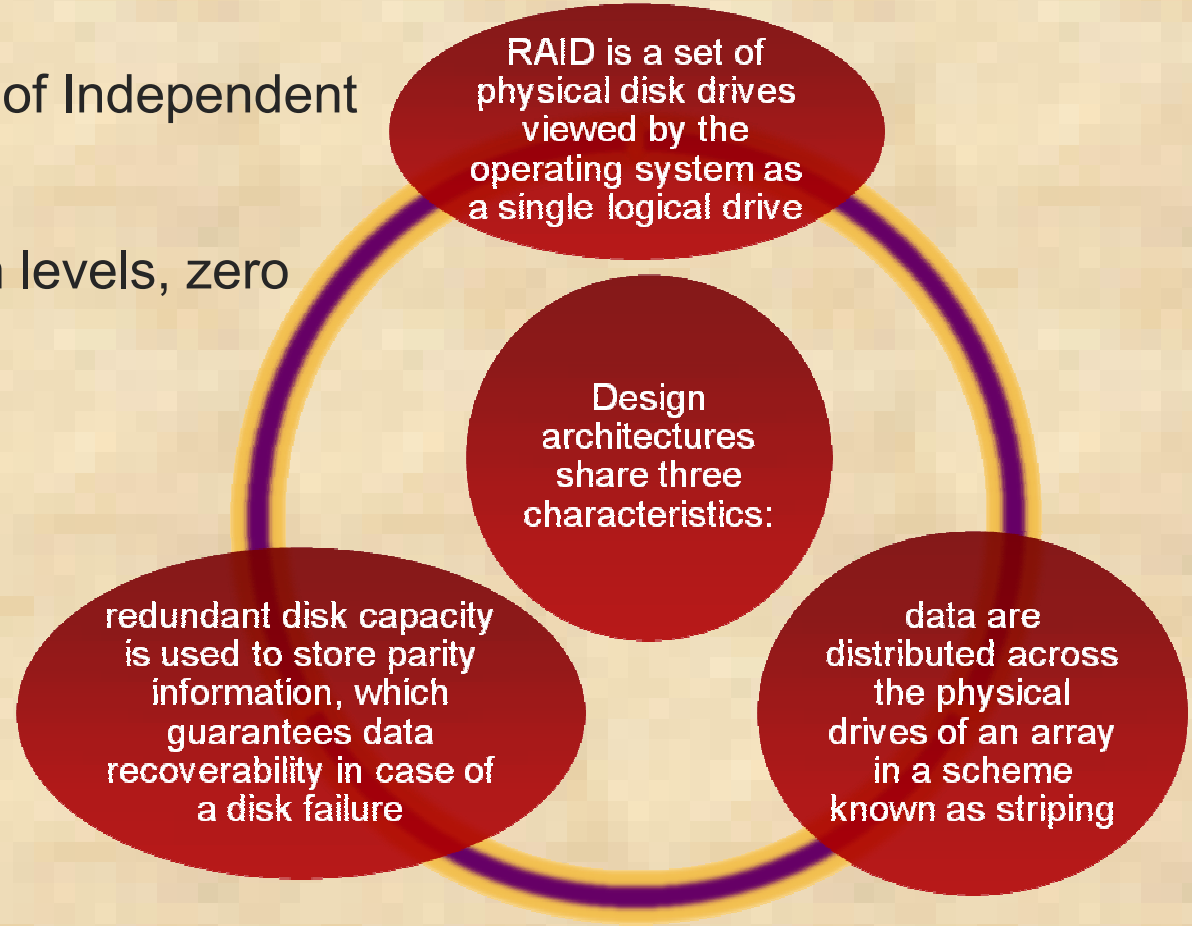
Table 11.2 Comparison of Disk Scheduling Algorithms

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

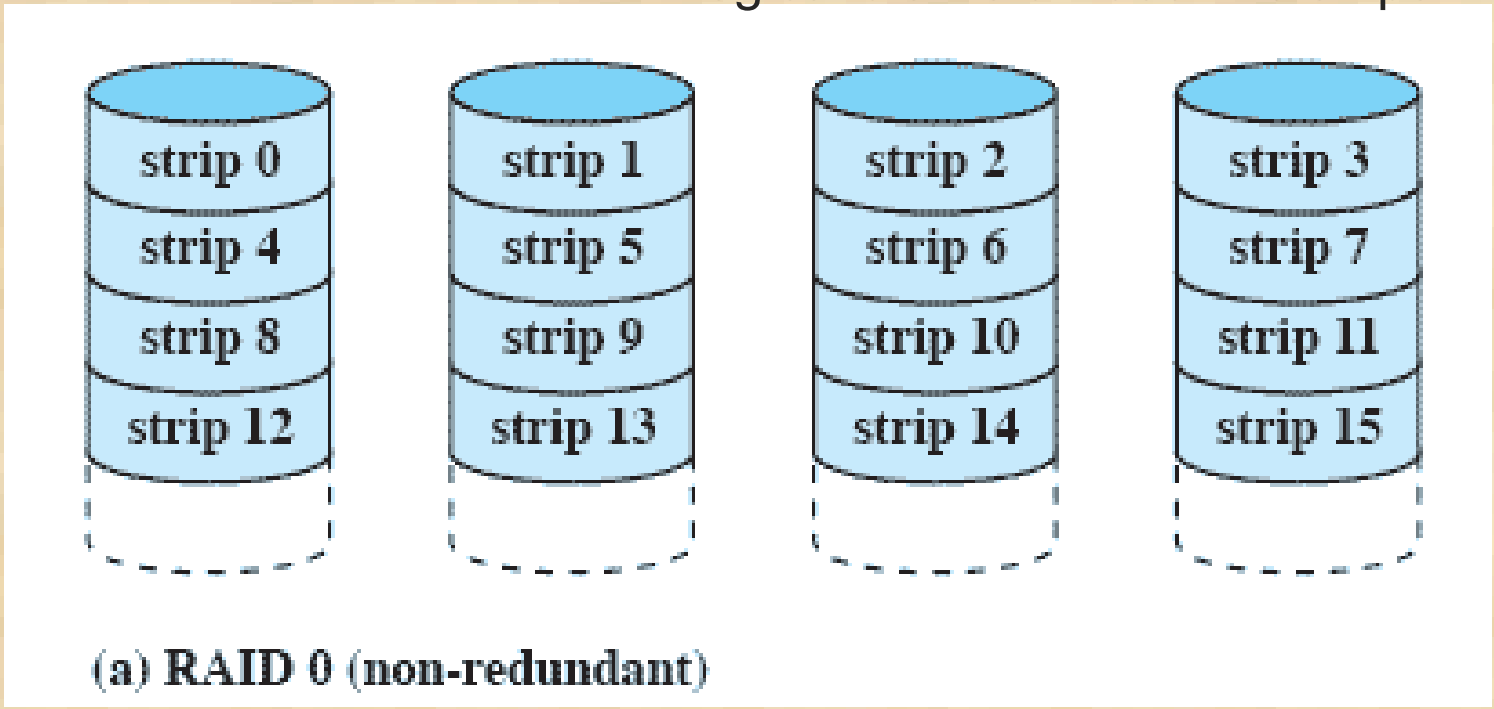
RAID

- Redundant Array of Independent Disks
- Consists of seven levels, zero through six



RAID Level 0

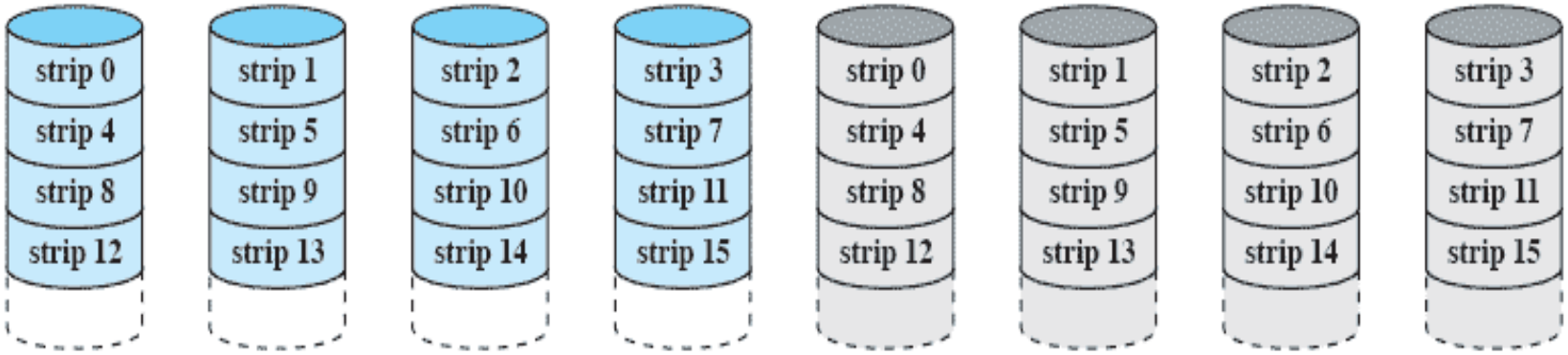
- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips



(a) RAID 0 (non-redundant)

RAID Level 1

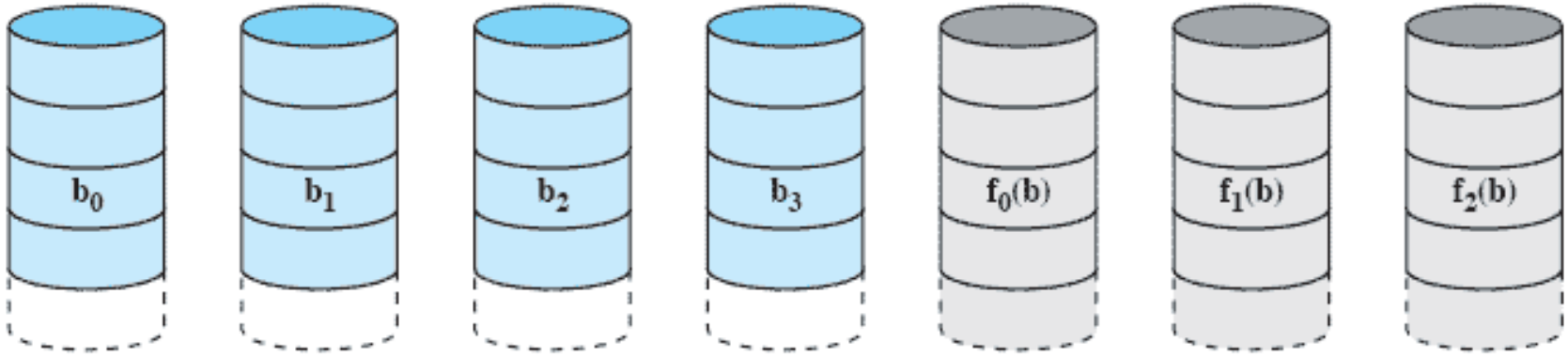
- Redundancy is achieved by the simple expedient of duplicating all the data
- There is no “write penalty”
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost



(b) RAID 1 (mirrored)

RAID Level 2

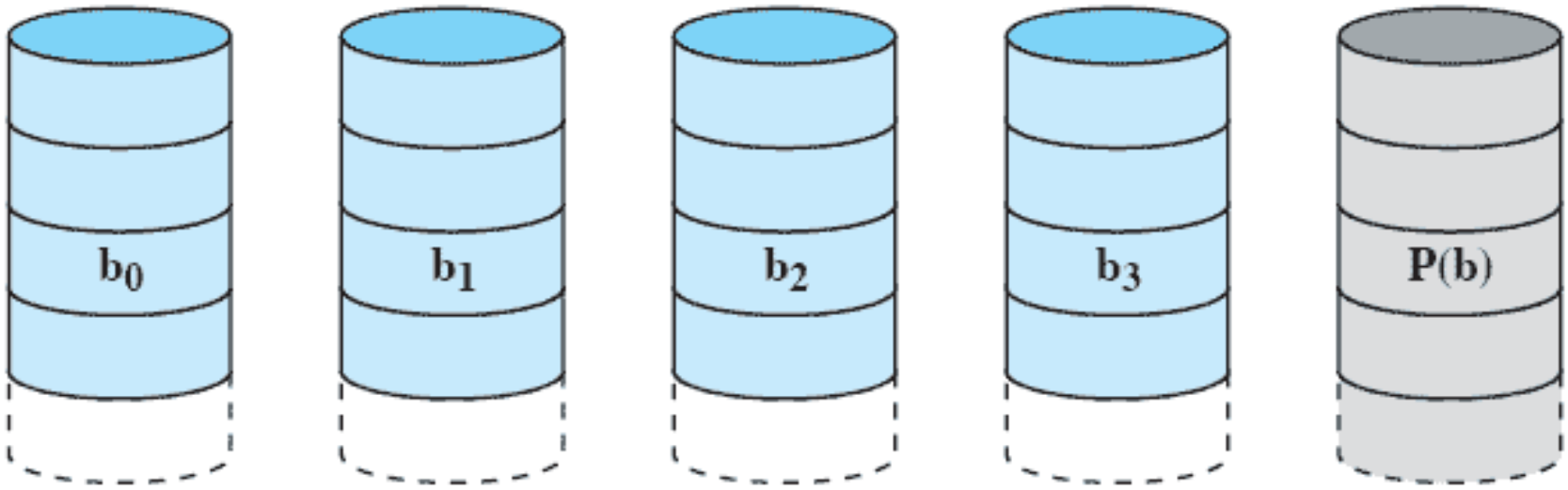
- Makes use of a parallel access technique
- Data striping is used
- Typically a Hamming code is used
- Effective choice in an environment in which many disk errors occur



(c) RAID 2 (redundancy through Hamming code)

RAID Level 3

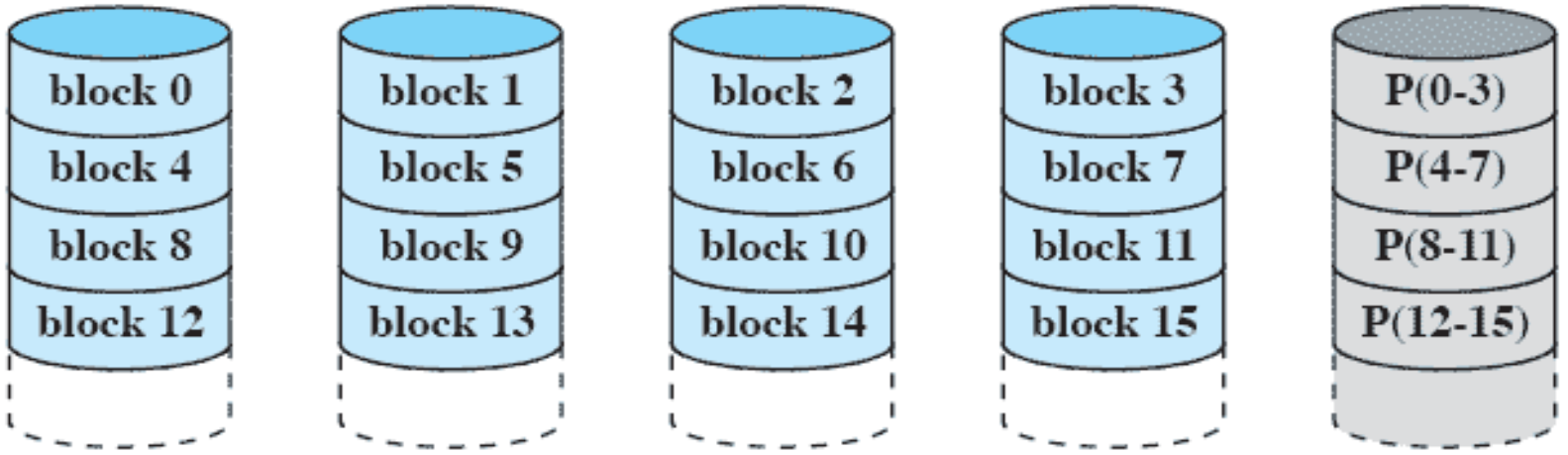
- Requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates



(d) RAID 3 (bit-interleaved parity)

RAID Level 4

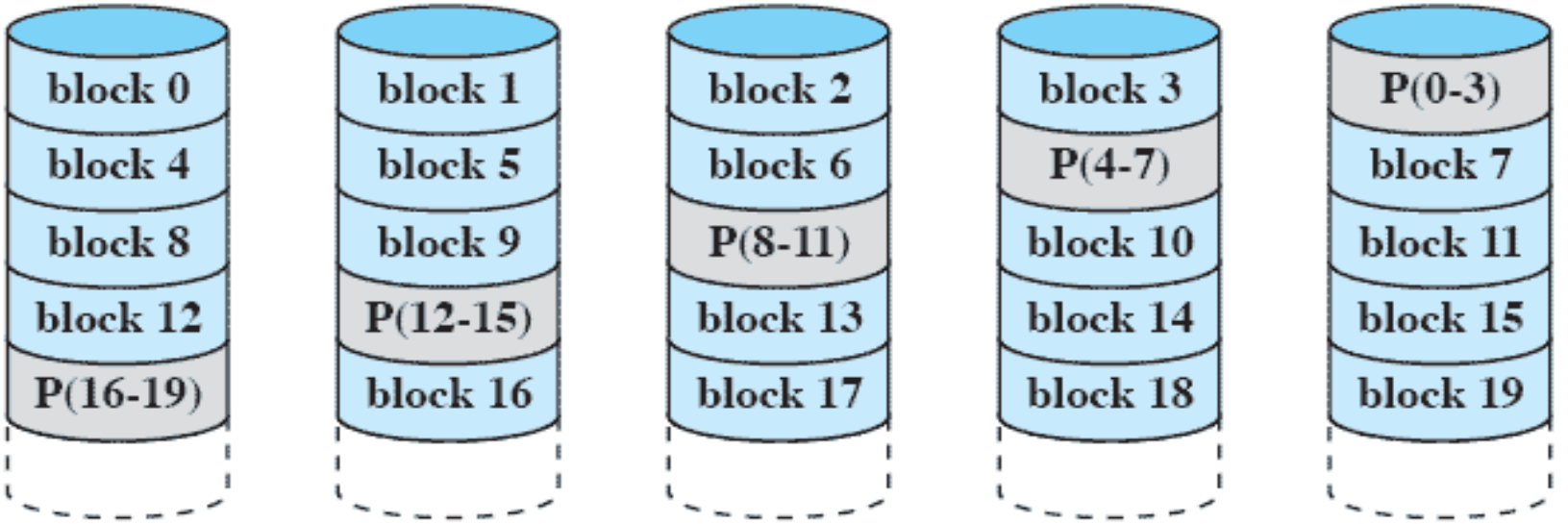
- Makes use of an independent access technique
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed



(e) RAID 4 (block-level parity)

RAID Level 5

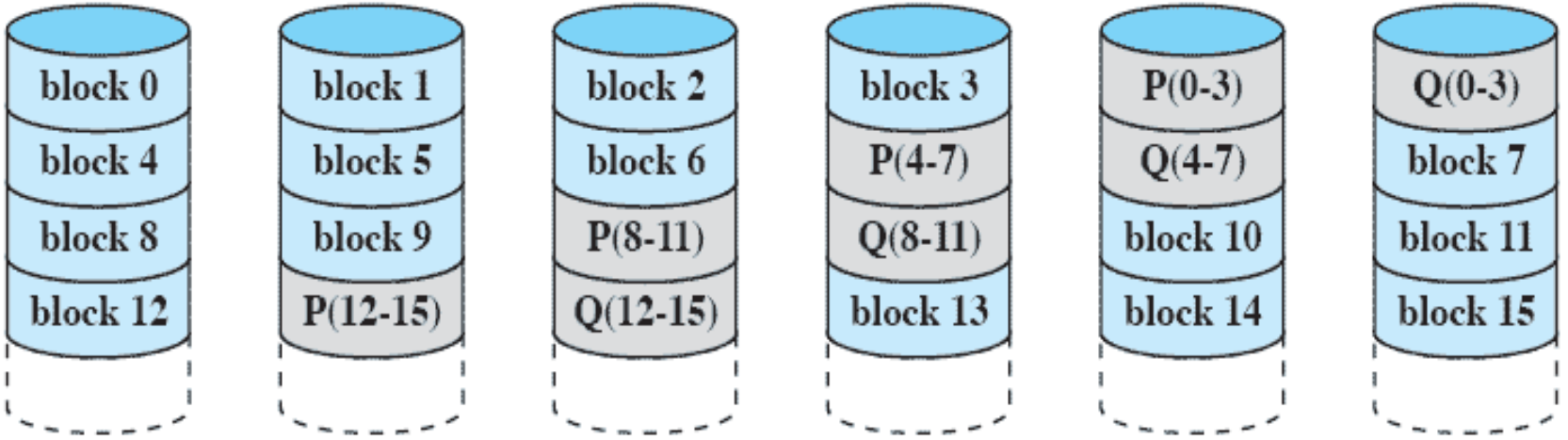
- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss



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

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks



(g) RAID 6 (dual redundancy)

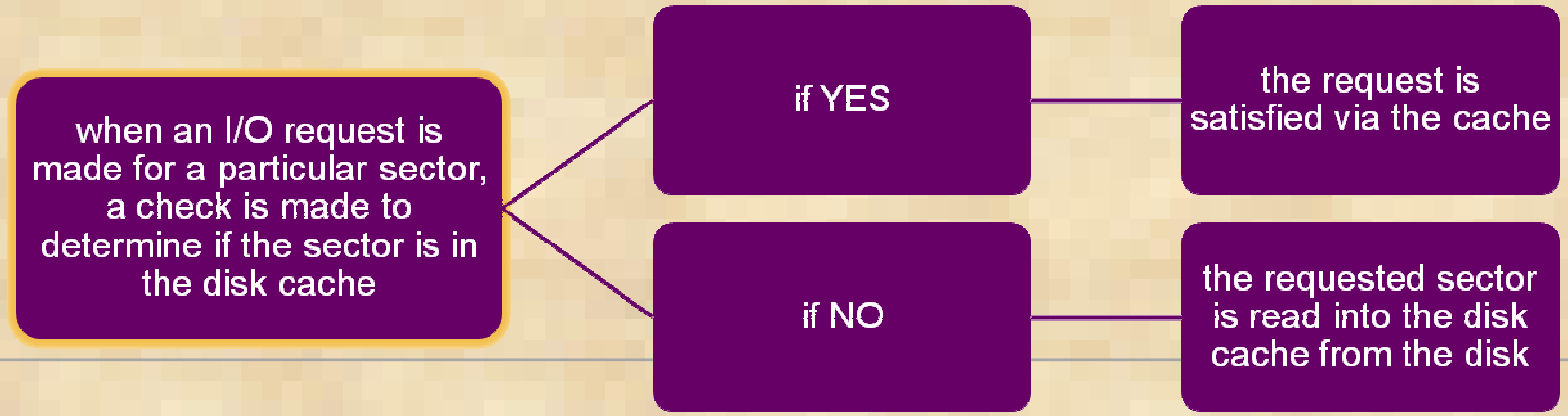
Table 11.4 RAID Levels

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

Note: N, number of data disks; m, proportional to log N.

Disk Cache

- *Cache memory* is used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor
- Reduces average memory access time by exploiting the principle of locality
- *Disk cache* is a buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk



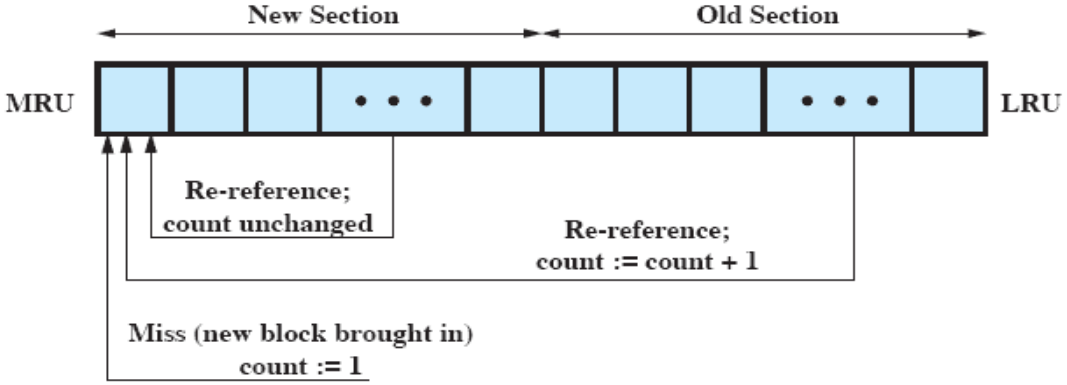
Least Recently Used (LRU)

- 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

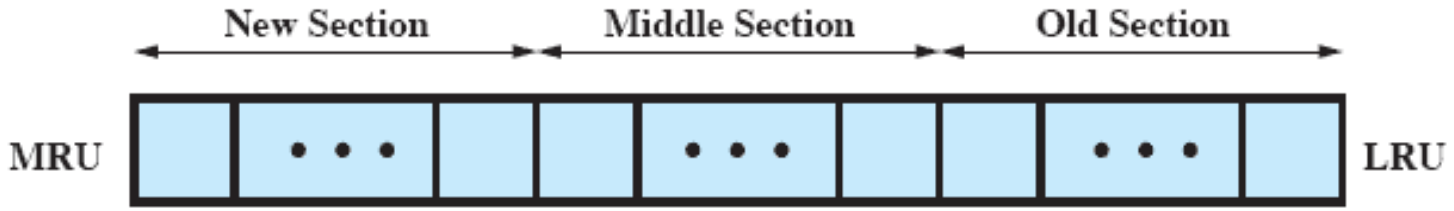
Least Frequently Used (LFU)

- 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



(b) Use of three sections

LRU Disk Cache Performance

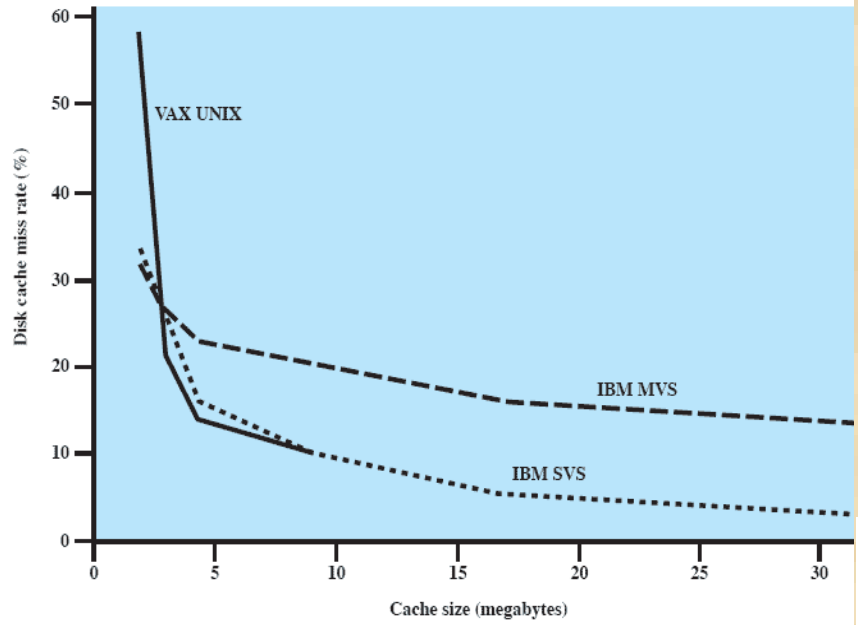


Figure 11.10 Some Disk Cache Performance Results Using LRU

Frequency-Based Replacement

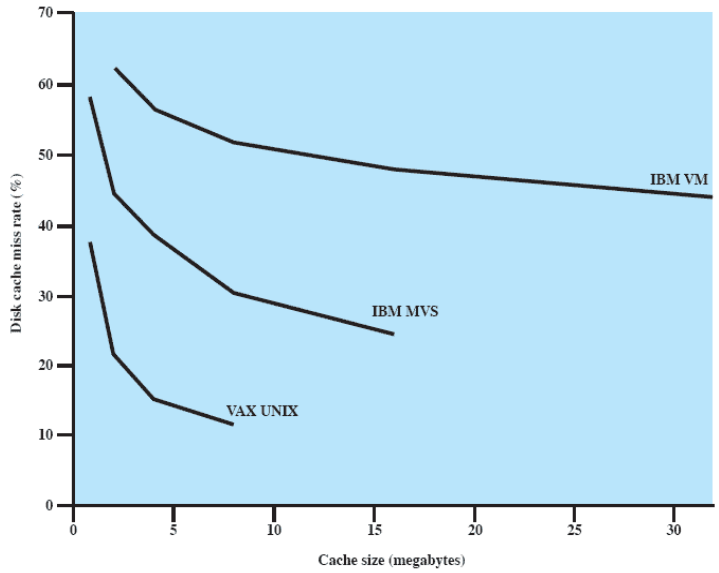


Figure 11.11 Disk Cache Performance Using Frequency-Based Replacement [ROBI90]

UNIX SVR4 I/O

- Two types of I/O
  - Buffered
    - system buffer caches
    - character queues
  - Unbuffered

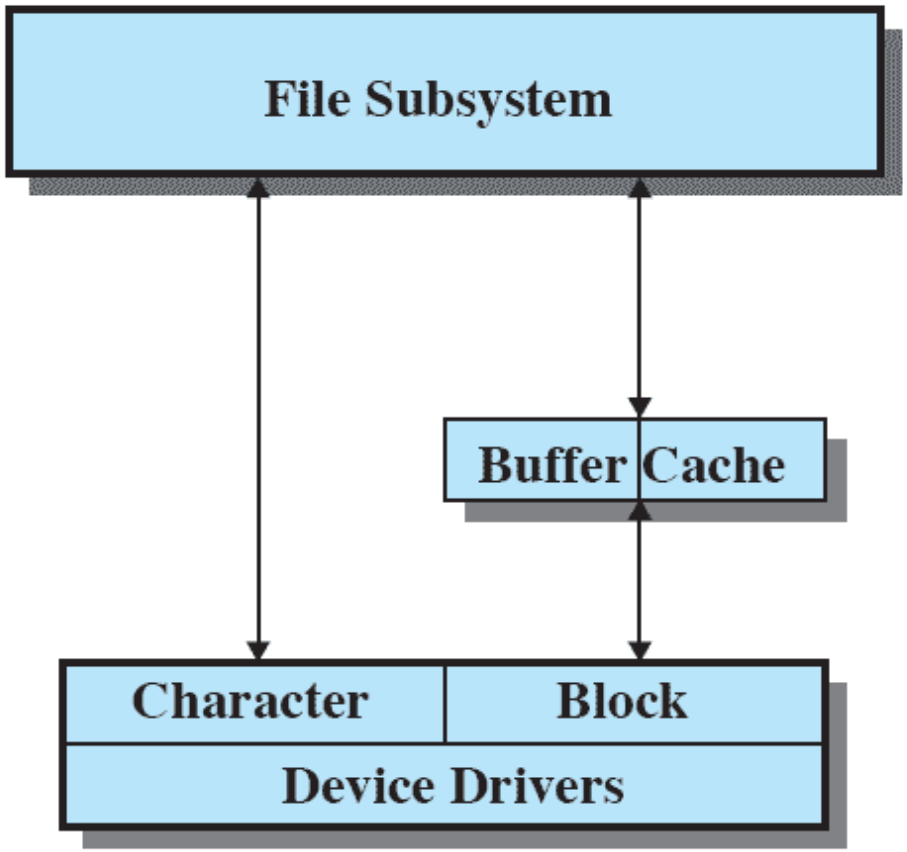


Figure 11.12 UNIX I/O Structure

Buffer Cache

- Three lists are maintained:
  - free list
  - device list
  - driver I/O queue

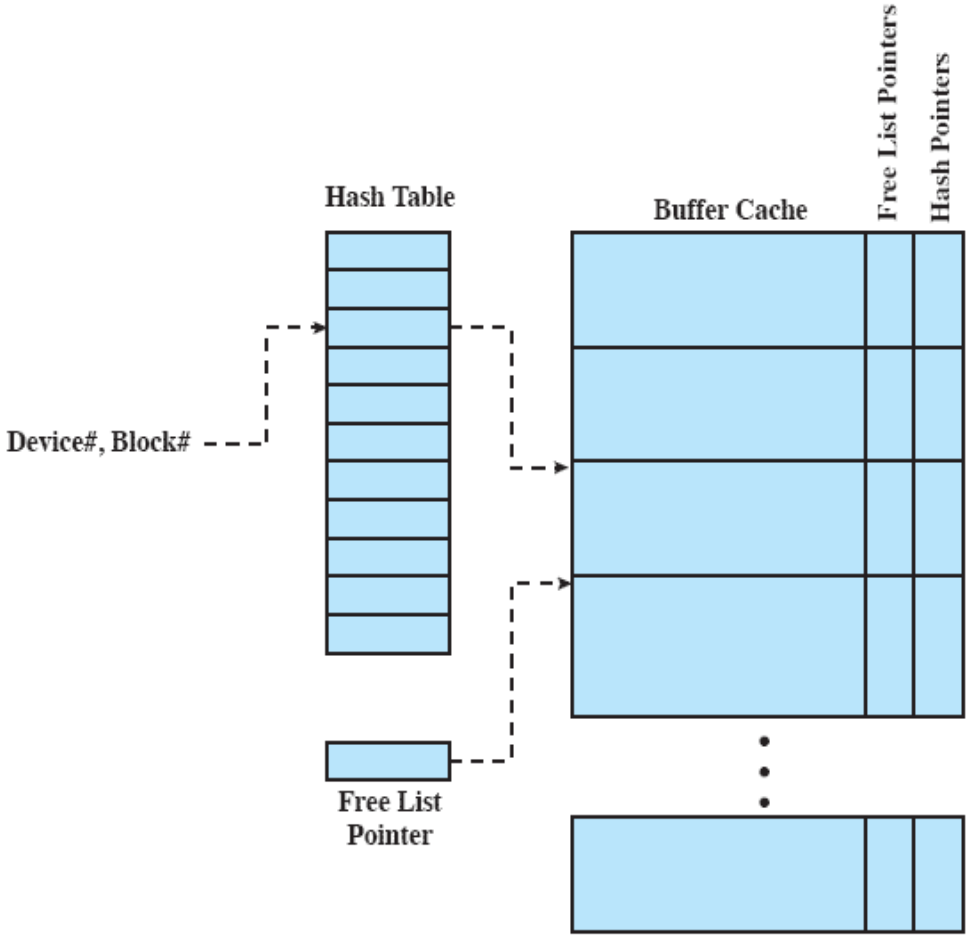


Figure 11.13 UNIX Buffer Cache Organization

Character Queue

Used by character oriented devices

terminals and printers

Either written by the I/O device and read by the process or vice versa

producer/consumer model is used

Character queues may only be read once

as each character is read, it is effectively destroyed

Unbuffered I/O

- Is simply DMA between device and process space
- Is always the fastest method for a process to perform I/O
- Process is locked in main memory and cannot be swapped out
- I/O device is tied up with the process for the duration of the transfer making it unavailable for other processes

Device I/O in UNIX

Linux I/O

- 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 is the Linux Elevator

For Linux 2.6 the Elevator algorithm has been augmented by two additional algorithms:

- the deadline I/O scheduler
- the anticipatory I/O scheduler

Deadline Scheduler

- Uses three queues:
  - incoming requests
  - read requests go to the tail of a FIFO queue
  - write requests go to the tail of a FIFO queue
- Each request has an expiration time

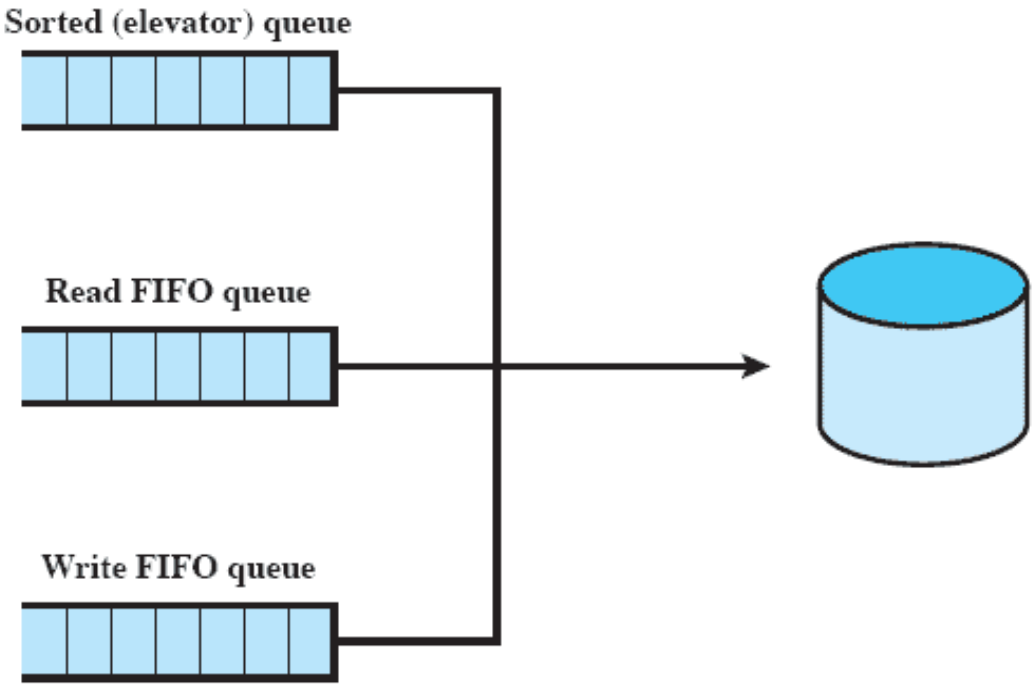


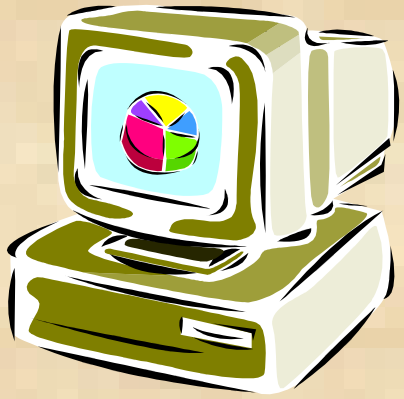
Figure 11.14 The Linux Deadline I/O Scheduler

Anticipatory I/O Scheduler

- 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
  - 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

Linux Page Cache

- For Linux 2.4 and later there is a single unified page cache for all traffic between disk and main memory
- Benefits:
  - dirty pages can be collected and written out efficiently
  - pages in the page cache are likely to be referenced again due to temporal locality



Windows I/O Manager

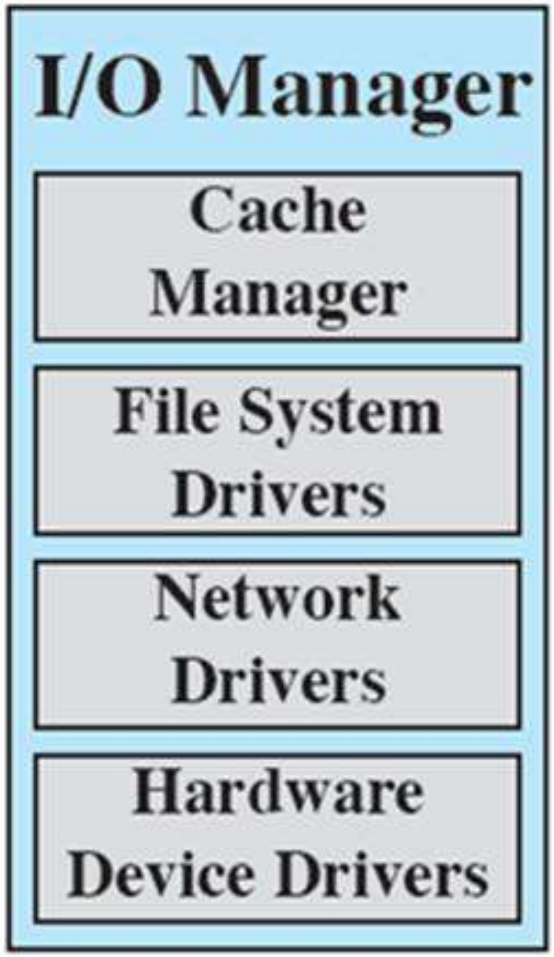
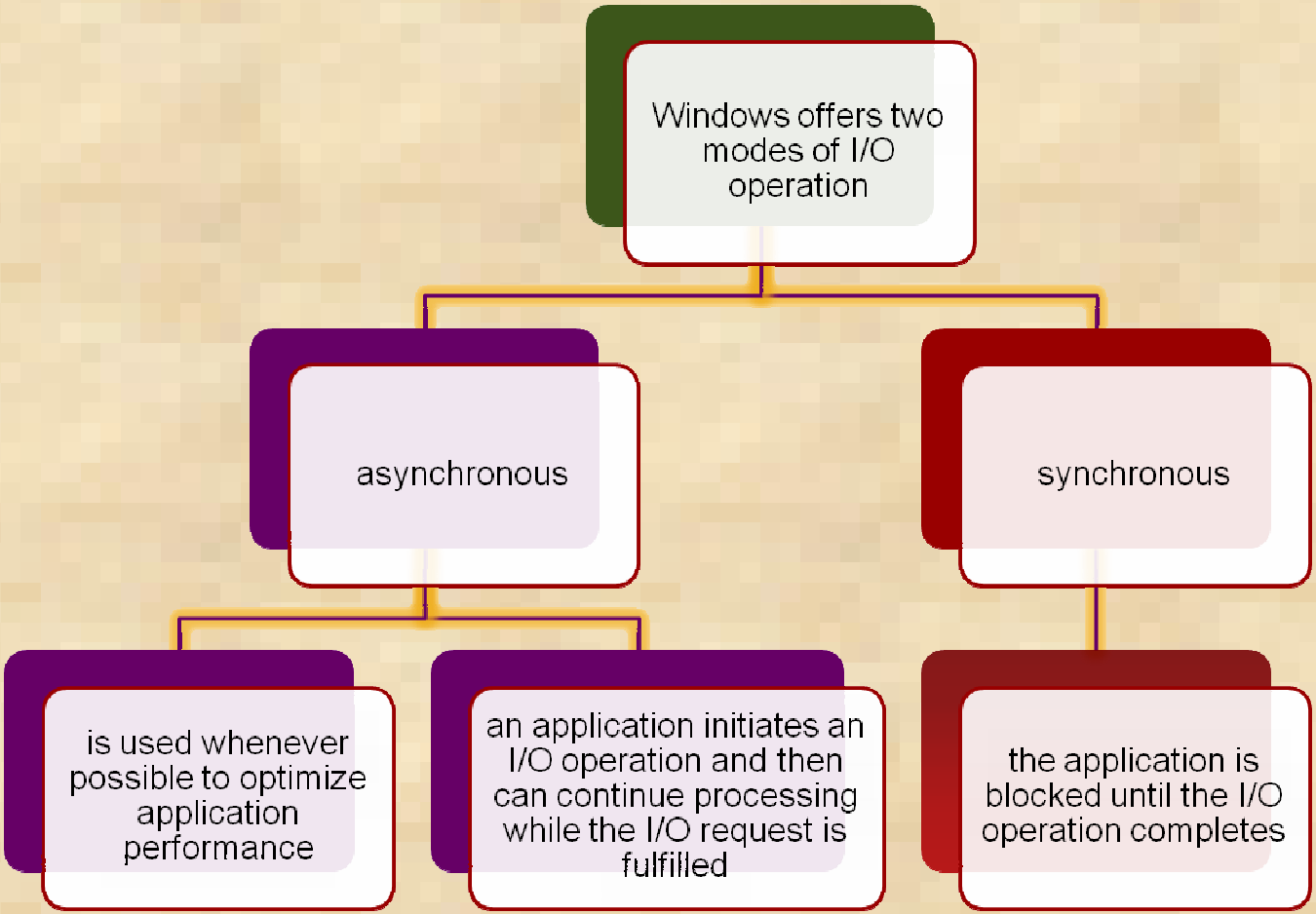


Figure 11.15 Windows I/O Manager

Basic I/O Facilities

- **Cache Manager**
  - maps regions of files into kernel virtual memory and then relies on the virtual memory manager to copy pages to and from the files on disk
- **File System Drivers**
  - sends I/O requests to the software drivers that manage the hardware device adapter
- **Network Drivers**
  - Windows includes integrated networking capabilities and support for remote file systems
  - the facilities are implemented as software drivers
- **Hardware Device Drivers**
  - the source code of Windows device drivers is portable across different processor types

Asynchronous and Synchronous I/O



I/O Completion

- Windows provides five different techniques for signaling I/O completion:

1. Signaling the file object
2. Signaling an event object
3. Asynchronous procedure call
4. I/O completion ports
5. Polling

Windows RAID Configurations

- Windows supports two sorts of RAID configurations:

Hardware RAID	Software RAID
separate physical disks combined into one or more logical disks by the disk controller or disk storage cabinet hardware	noncontiguous disk space combined into one or more logical partitions by the fault-tolerant software disk driver, FTDISK

Volume Shadow Copies and Volume Encryption

- **Volume Shadow Copies**
  - efficient way of making consistent snapshots of volumes so they can be backed up
  - also useful for archiving files on a per-volume basis
  - implemented by a software driver that makes copies of data on the volume before it is overwritten

- **Volume Encryption**
  - Windows uses BitLocker to encrypt entire volumes
  - more secure than encrypting individual files
  - allows multiple interlocking layers of security

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