Operating Systems: Internals and Design Principles

Chapter 11
I/O Management and Disk Scheduling

Eighth Edition
By William Stallings

11.1 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, and actuators

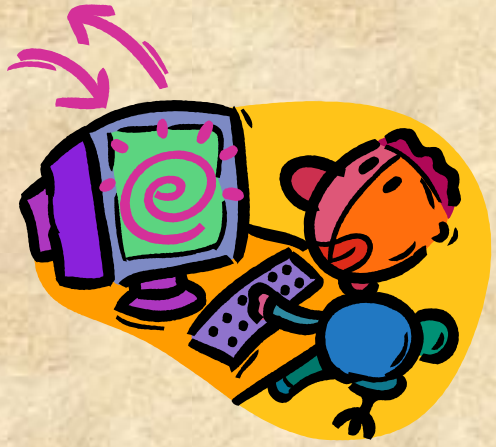
Communication

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

11.2 Organization of the I/O Function

- **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 –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
  - the processor sends a request for the transfer of a block of data to the DMA module
  - interrupted only after the entire block has been transferred

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. The processor directly controls a peripheral device (e.g., microprocessor-controlled devices)
2. • A controller or I/O module is added
   • The processor uses programmed I/O without interrupts
3. The same configuration as step 2, but now interrupts are employed for increasing efficiency
4. The I/O module is given direct control of memory via DMA to move data to or from memory without involving the processor
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

11.3 Operating System Design Issues

Design Objectives

Efficiency

- 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

- 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
- Three most important logical structures
  - local peripheral device
    - logical I/O allows user processes to deal with the device in terms of a device ID and simple commands like open, close, read and write
    - device I/O converts requested operations and data into appropriate sequence of I/O instructions, channel commands, and controller orders
    - scheduling and control perform actual queueing and scheduling of I/O operations, and actually interacts with the device hardware
  - communication port
  - file system

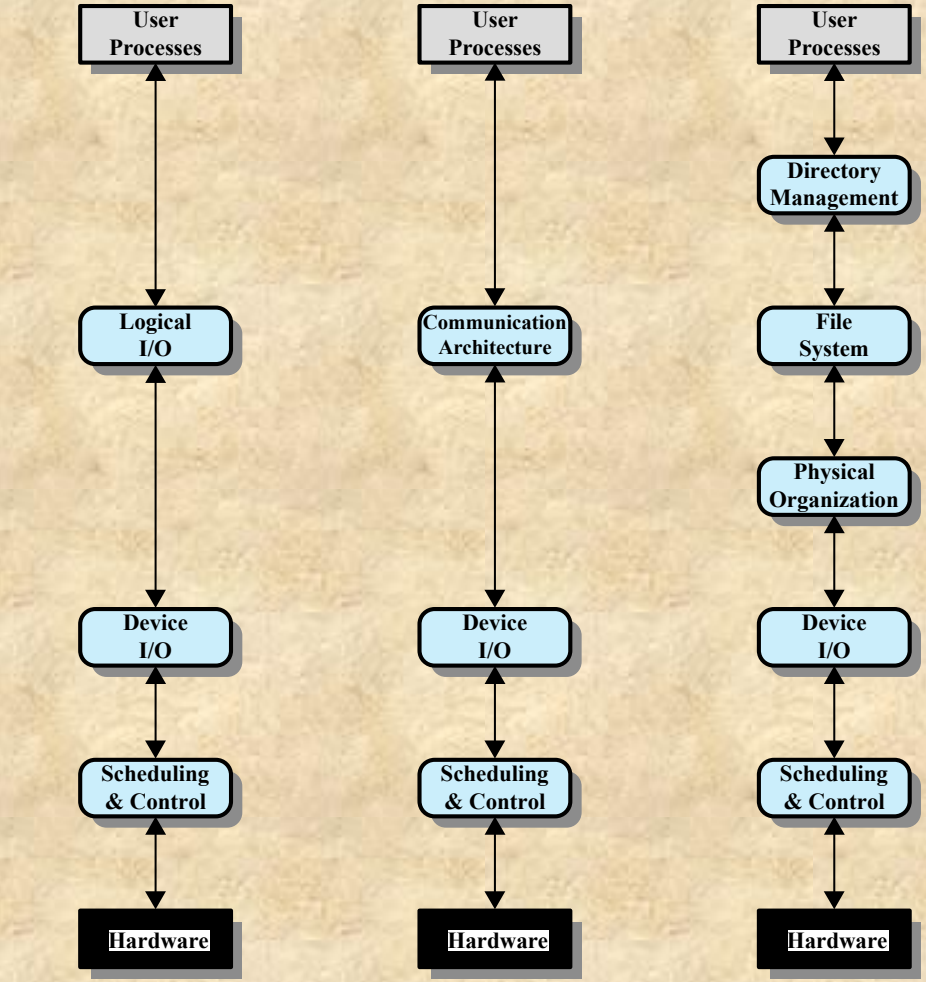


Figure 11.4 A Model of I/O Organization

11.4 I/O Buffering

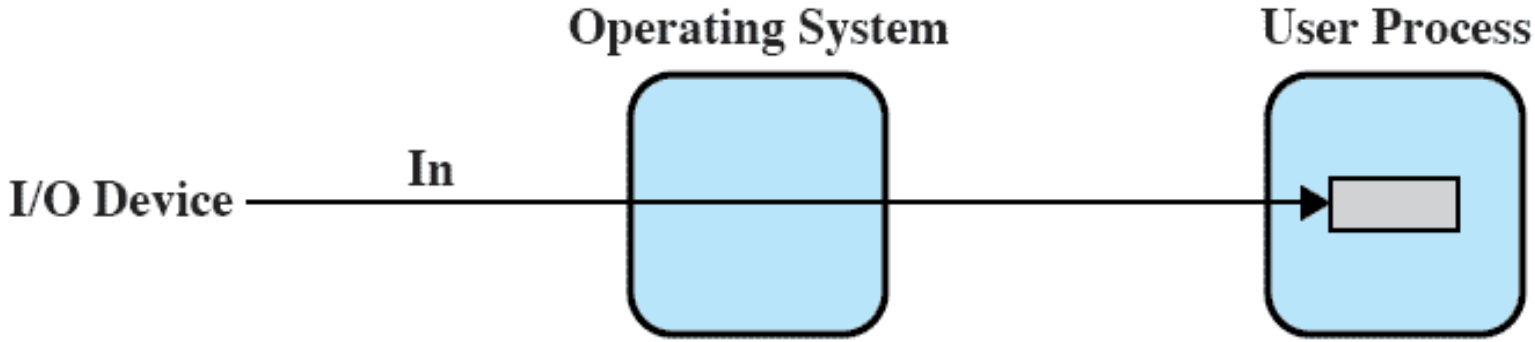
- After issuing I/O command to complete, we need to wait for the data to become available either by busy waiting or by process suspension
  - problem
    - the program is hung up waiting for the relatively slow I/O to complete
    - a risk of single-process deadlock if a process is suspended awaiting for I/O completion and then is swapped out before the beginning of I/O operation
      - the process is blocked waiting on the I/O event
      - the I/O operation is blocked waiting for the process to be swapped in

- To avoid these overheads and inefficiency, perform input transfers in advance of requests being made and perform output transfers some time after the request is made
- This technique is known as buffering
- Two types of I/O devices

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 with no block structure • terminals, printers, communications ports, and most other devices that are not secondary storage are examples

No Buffer

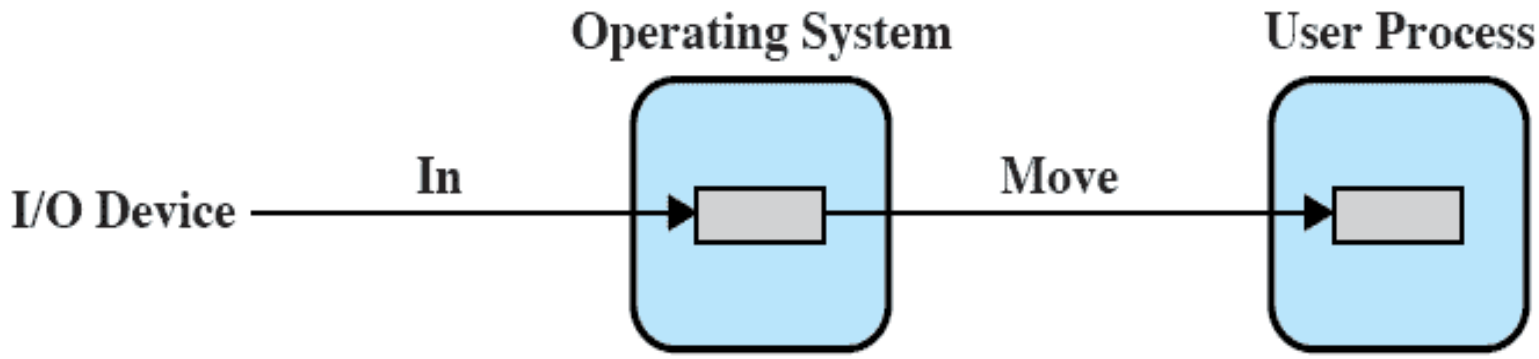
- Without a buffer, the OS directly accesses the device when it needs



(a) No buffering

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, since it must keep track of the assignment of system buffers to user processes
  - swapping logic is also affected if the I/O operation involves the same disk that is used for swapping

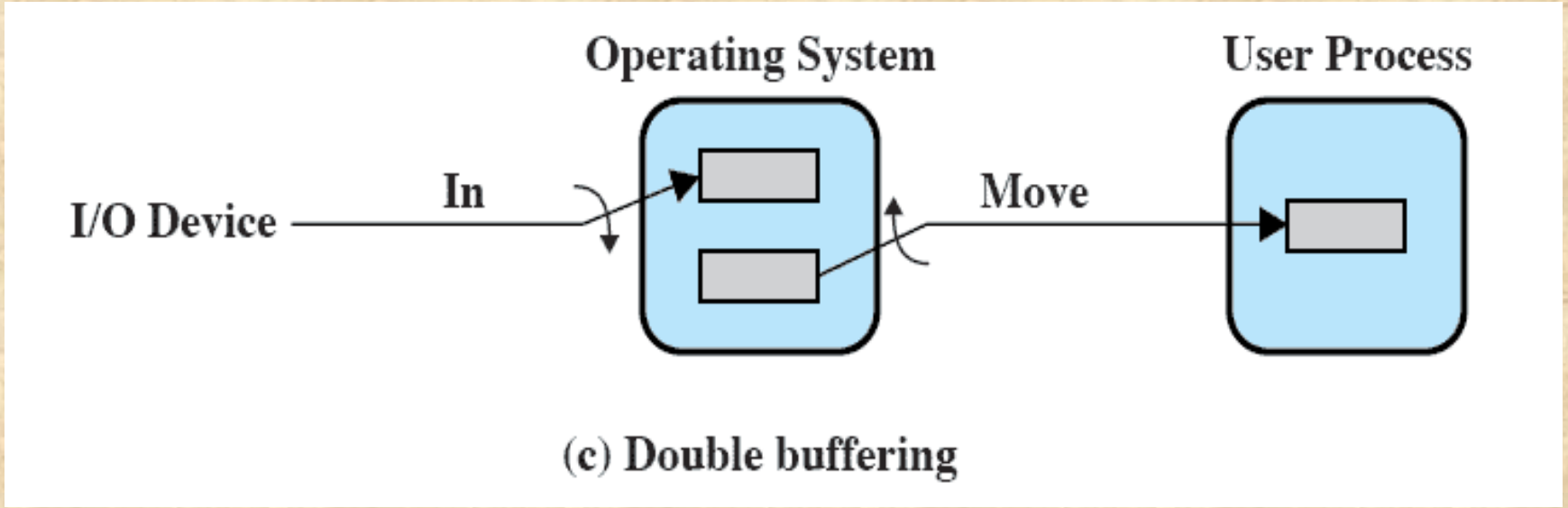
Stream-Oriented Single Buffer

- 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

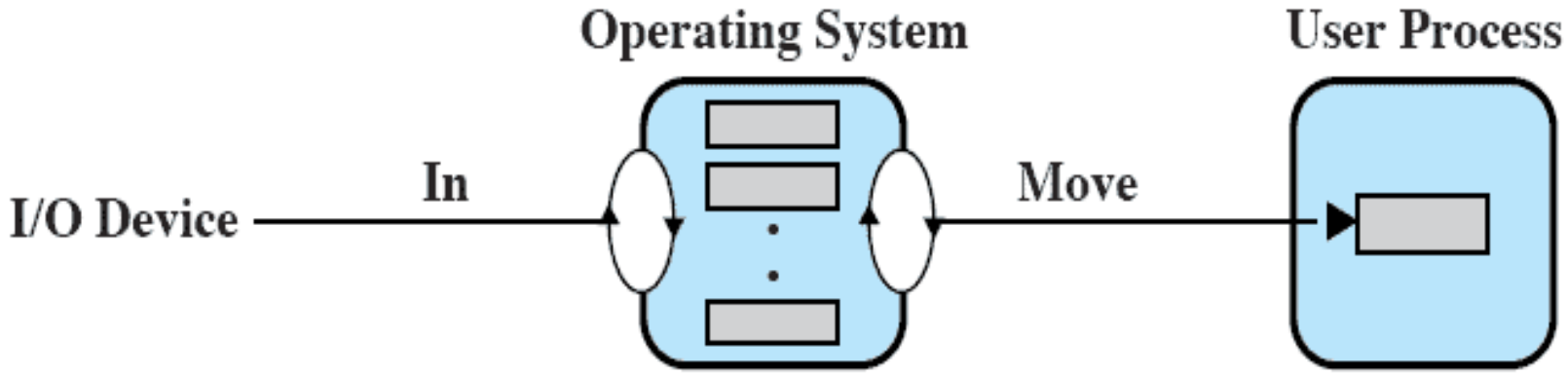
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer
- Also known as buffer swapping



(c) Double buffering

Circular Buffer

- Two or more buffers are used if a process performs rapid bursts of I/O
- Each individual buffer is one unit in a circular buffer
- Simply the bounded-buffer producer/consumer model



(d) Circular buffering

11.5 Disk Scheduling

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

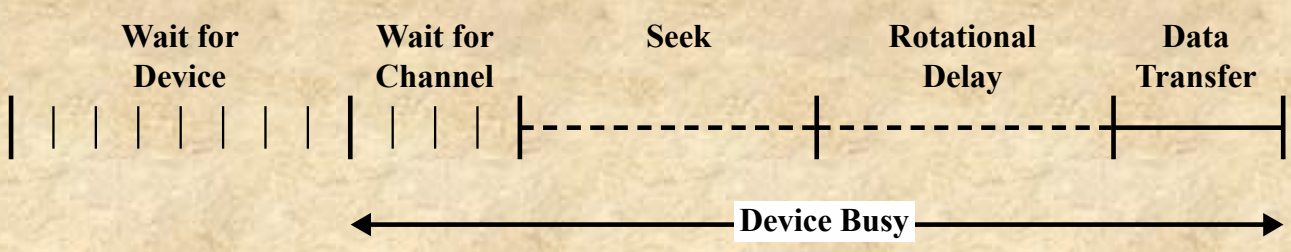


Figure 11.6 Timing of a Disk I/O Transfer

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** (※ a typical average seek time < 10ms)
- The time it takes for the beginning of the sector to reach the head is known as **rotational delay** (※ the average rotational delay = 2ms, assuming 15,000 rpm)
- **access time =** seek time + rotational delay

Positioning the Read/Write Heads

- Transfer time (T)

$$T = \frac{b}{rN}$$

where b = number of bytes to be transferred
N = number of bytes on a track
r = rotation speed per second

- Total average access time (T_a)

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

where T_s is the average seek time

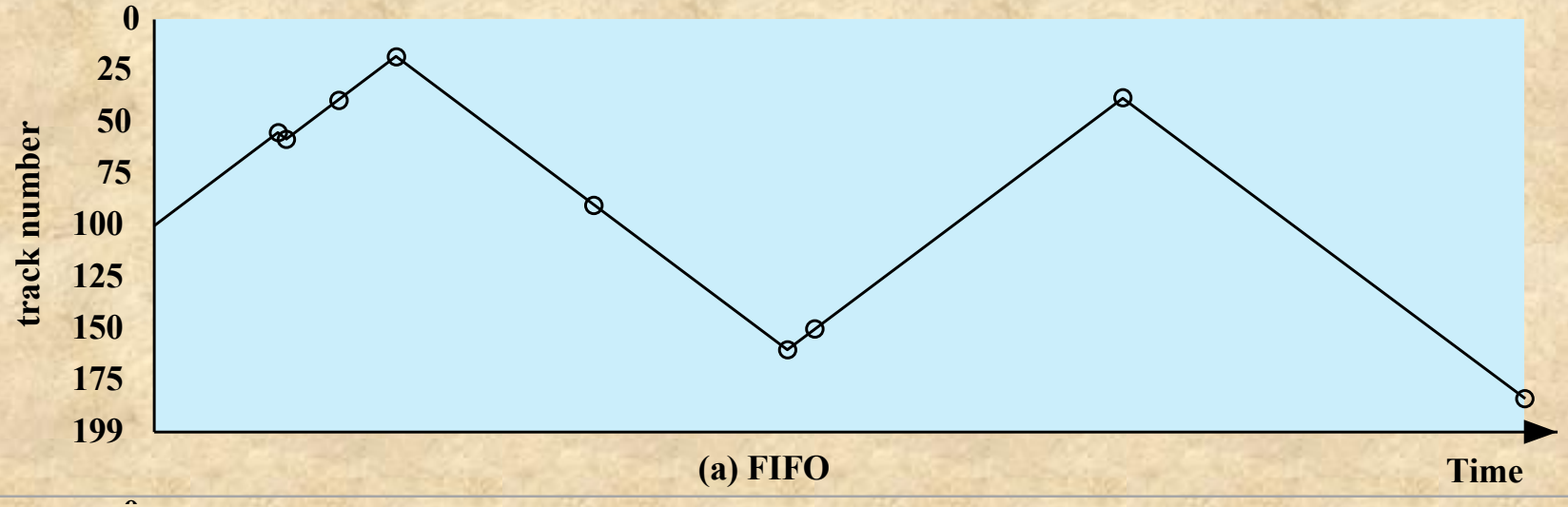
(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**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

Disk Scheduling Policies

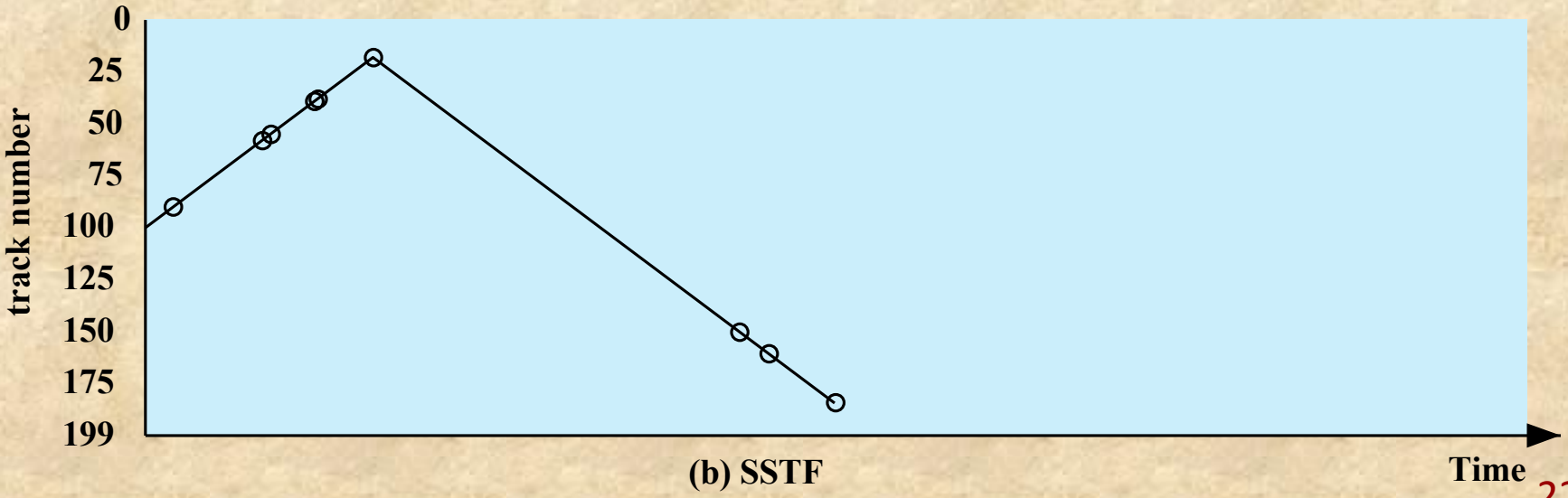
First-In, First-Out (FIFO)

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



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
- Random tie-breaking algorithm to resolve cases of equal distance



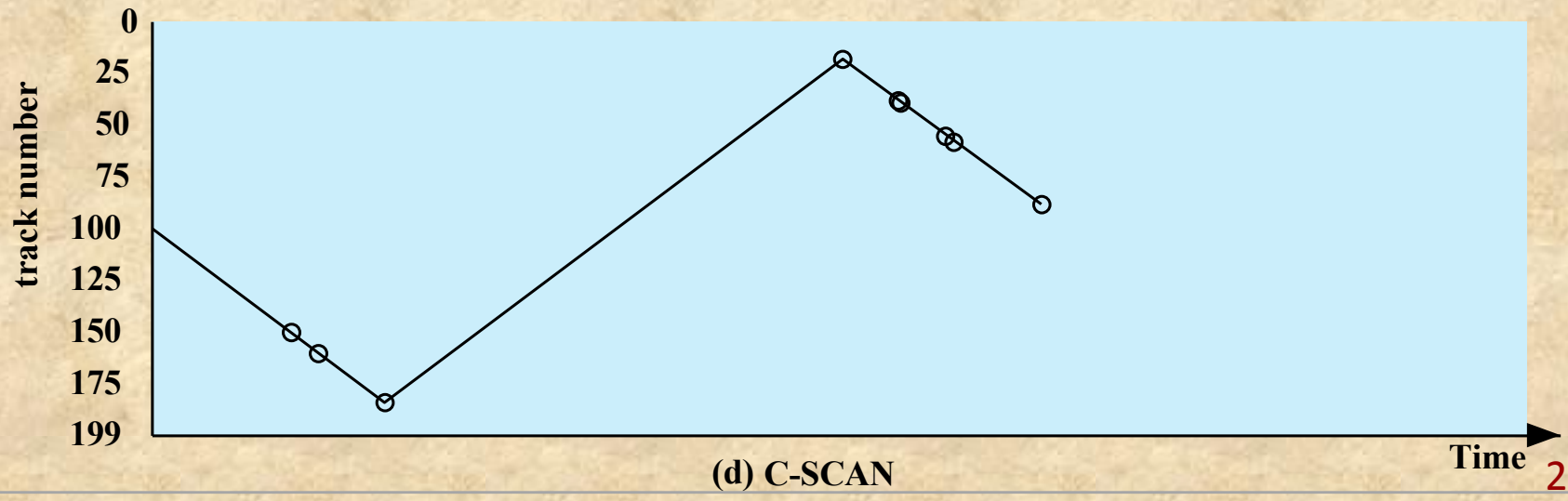
SCAN

- Also known as the elevator algorithm
- Arm moves in one direction only
  - satisfies all outstanding requests until it reaches the last track or until there are no more requests (referred to the *LOOK* policy) in that direction then the direction is reversed
- Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks

(c) SCAN

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
- Reduce the maximum delay experienced by new requests from *2t* to $t + s_{max}$, where s_{max} = maximum seek time and t = expected time for a scan from inner track to outer track



(d) C-SCAN

N-Step-SCAN

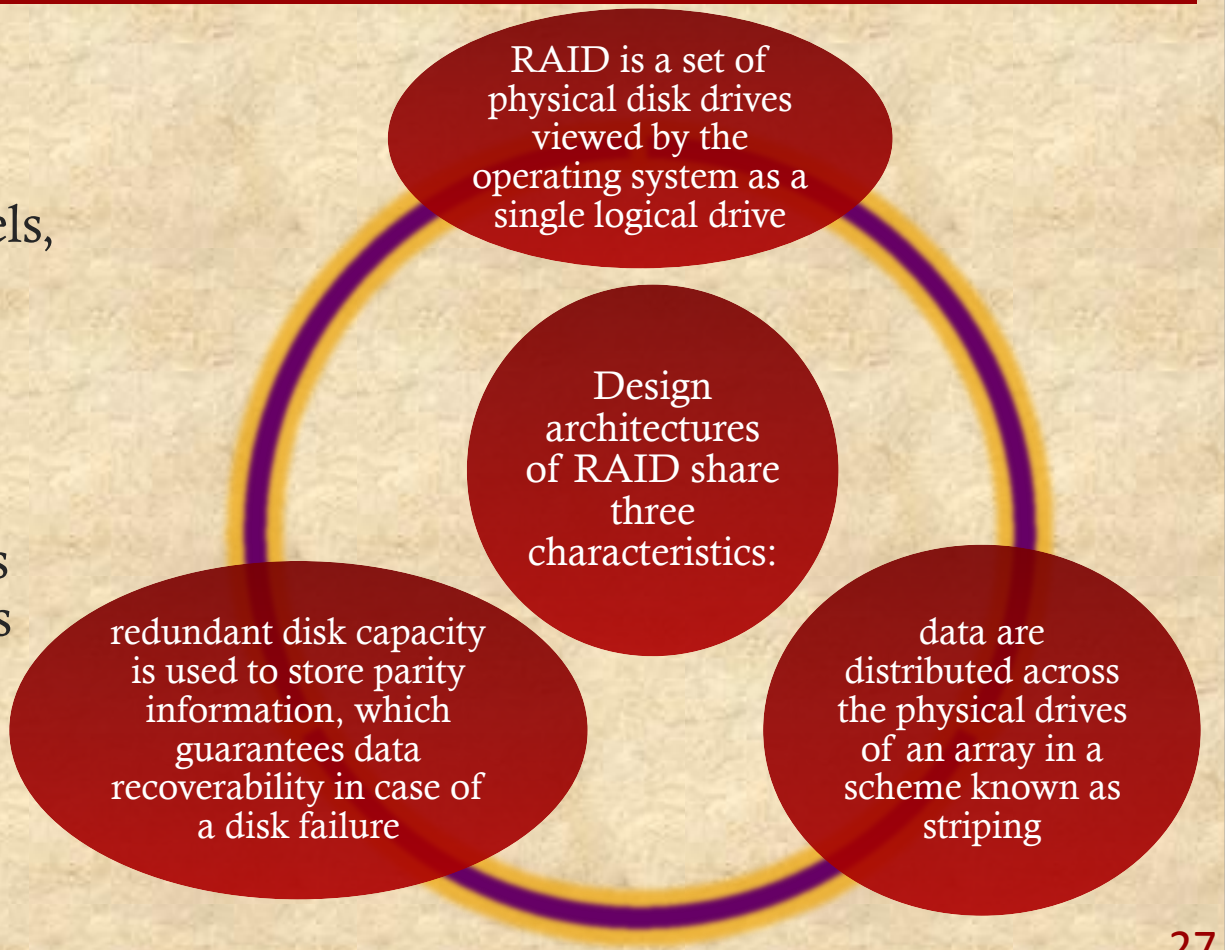
- With SSTF, SCAN, and C-SCAN, "arm stickiness" problem may occur
- Segments the disk request queue into subqueues of length *N*
- Subqueues are processed one at a time, using SCAN
- While a queue is being processed, new requests must be added to some other queue
- If fewer than *N* requests are available at the end of a scan, all of them are processed with the next scan
- With large values of N, N-step-SCAN → SCAN
- With a value of N = 1, N-step-SCAN → 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

11.6 RAID

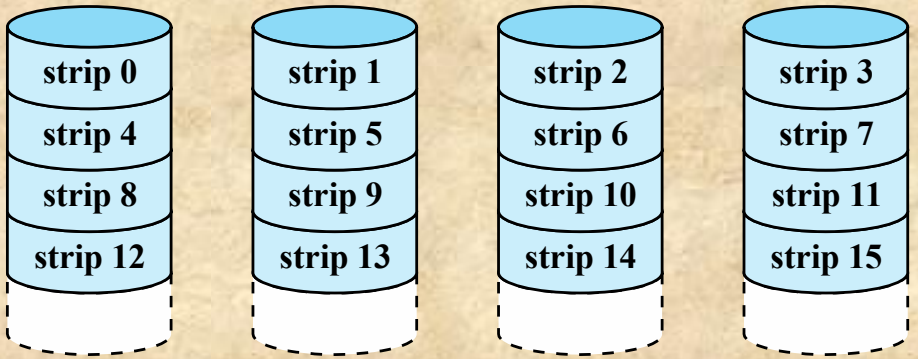
- Redundant Array of Independent Disks
- Consists of seven levels, zero through six
- Motivation: the performance improvement rate of secondary storage has been considerably less than the rate for processors and main memory



- Employs multiple disk drives and distributes data in such a way as to enable simultaneous access to data from multiple drives
  - improves I/O performance and allows easier incremental increases in capacity
- Makes use of stored parity information that enables the recovery of data lost due to a disk failure
  - the use of multiple devices increases the probability of failure
- Only four are commonly used: RAID levels 0, 1, 5, and 6

RAID Level 0

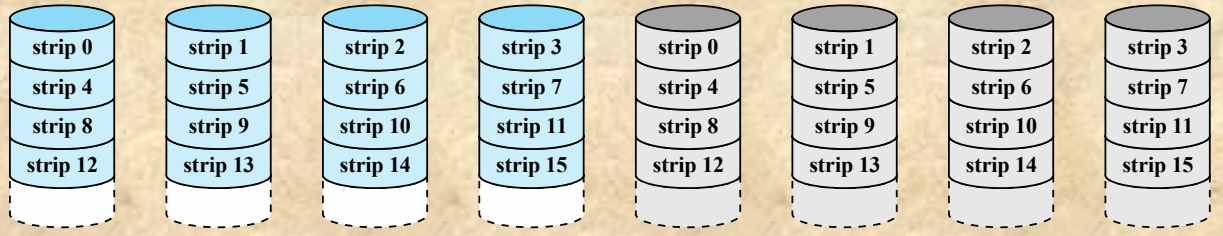
- Not a true RAID
  - no redundancy to improve performance or provide data protection
- The user and system data are distributed across all of the disks in the array
- Logical disk is divided into strips, which may be blocks or sectors
- Up to *n* logically contiguous strips for a single I/O request can be handled in parallel, greatly reducing the I/O transfer time.



(a) RAID 0 (non-redundant)

RAID Level 1

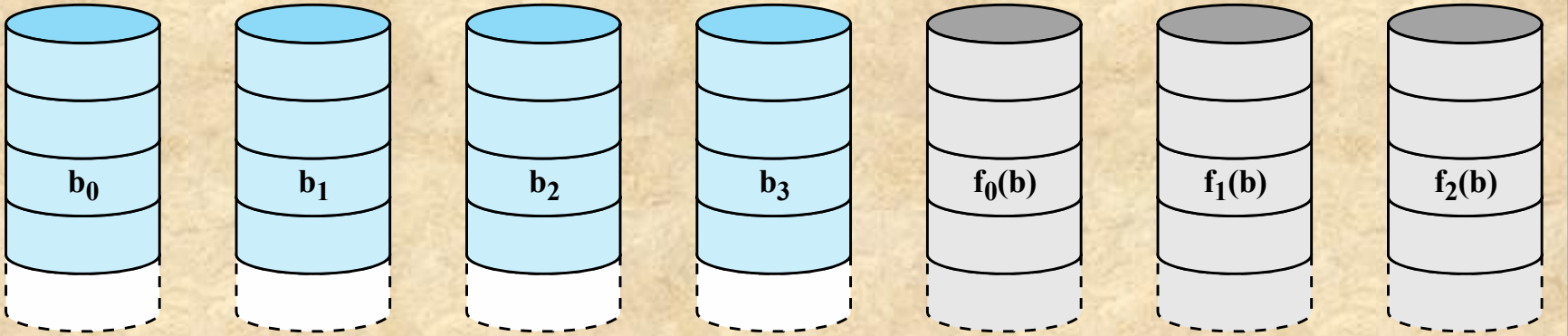
- Redundancy is achieved by duplicating all the data (called *data mirroring*) rather than parity calculation
- There is no "write penalty" due to no use of parity bits
- When a drive fails, the data may still be accessed from the second drive (simple recovery)
- In case of read requests, the performance of RAID1 can approach double of that of RAID 0
- In case of write requests, no significant performance gain over RAID0
- Principal disadvantage is the cost
  - require twice the disk space
  - limited to drives that store system software and data and other highly critical files



(b) RAID 1 (mirrored)

RAID Level 2

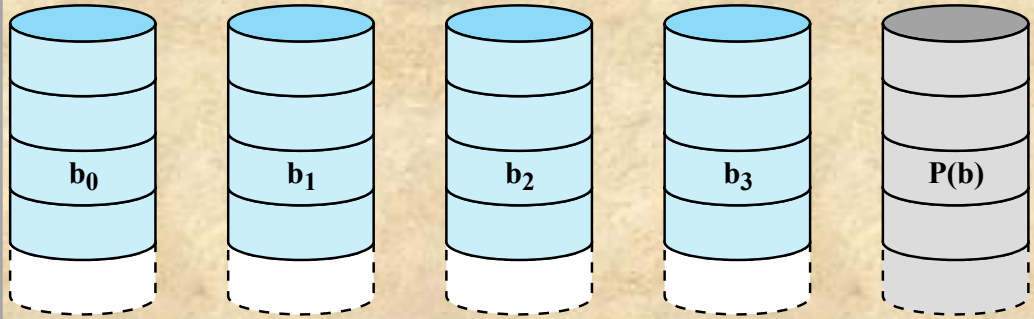
- Makes use of a parallel access technique
- Data striping is used, where the strips are very small, often as small as a single byte or word
- Typically a Hamming code is used
  - Correct single-bit errors and detect double-bit errors
  - $n = 2^m - 1$, where n = total bits and m = parity bits
- Only effective choice in an environment in which many disk errors occur



(c) RAID 2 (redundancy through Hamming code)

RAID Level 3

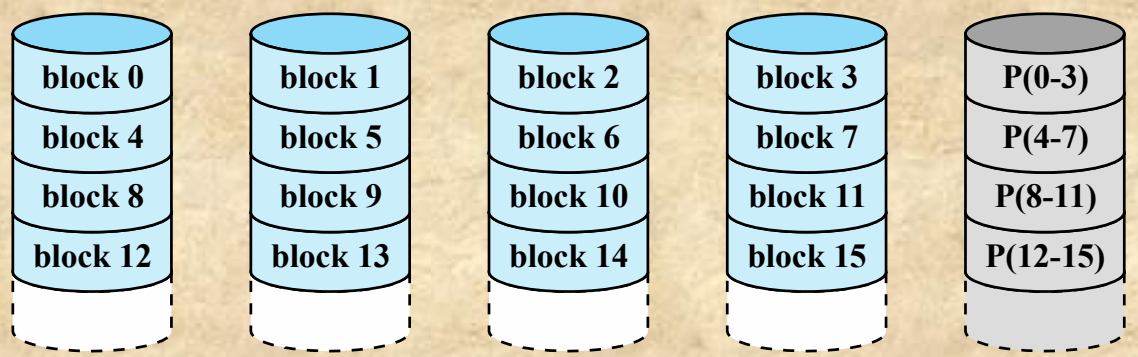
- Requires only a single redundant disk, no matter how large the disk array
  - instead of an error correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks
  - in the event of a drive failure, the parity drive is accessed and data is reconstructed from the remaining devices in reduced mode
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates, because data are striped in very small stripes, but only one I/O request can be executed at a time



(d) RAID 3 (bit-interleaved parity)

RAID Level 4

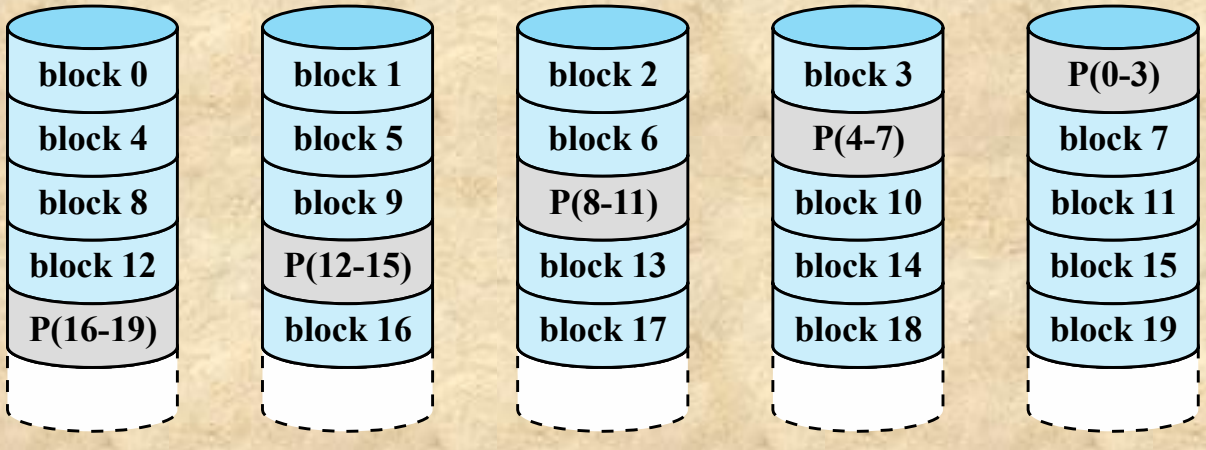
- RAID levels 4 through 6 makes use of an independent access technique
  - separate I/O requests can be satisfied in parallel
  - suitable for applications that require high I/O request rates rather than high data transfer rate
- A bit-by-bit parity strip is calculated across corresponding strips on 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
  - each time a write occurs, the user data and the corresponding parity bits must be updated



(e) RAID 4 (block-level parity)

RAID Level 5

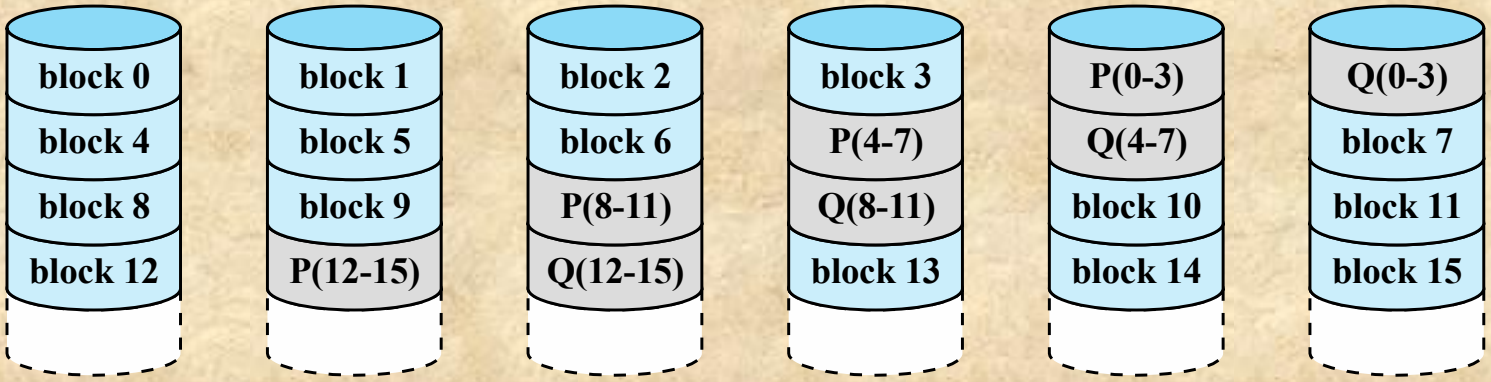
- Similar to RAID-4 but distributes the parity bits across all disks
  - Avoid the potential I/O bottleneck of the parity disk
- Typical allocation is a round-robin scheme
- The loss of any one disk does not result in data loss



(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
  - may regenerate data even if two disks fail
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks
  - more than a 30% drop in overall write performance



(g) RAID 6 (dual redundancy)

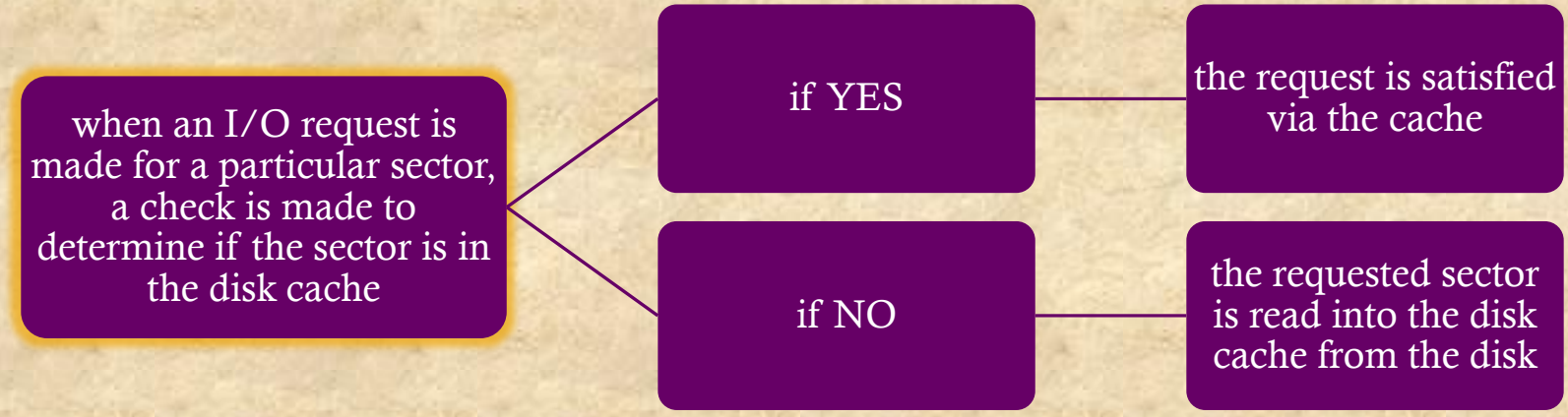
Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single 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

Table 11.4 RAID Levels

11.7 Disk Cache

- *Cache memory*
  - a memory that is smaller and faster than main memory
  - 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



Design Considerations

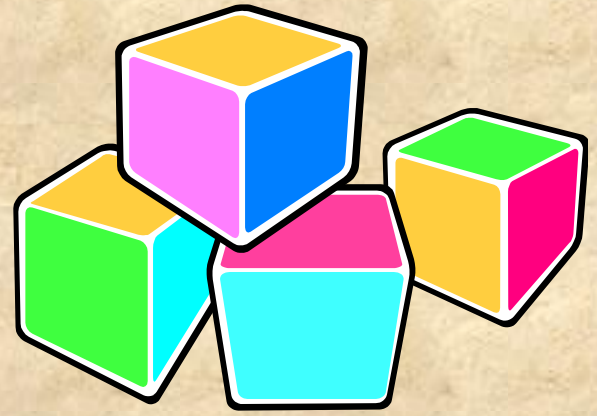
- First issue: how to deliver the data in the disk cache to the requesting process
  - transfer the block of data within main memory from the disk cache to memory assigned to the user process
  - use a shared memory capability and pass a pointer to the appropriate slot in the disk cache
    - save the time of a memory-to-memory transfer
    - allow shared access by other processes
- Second issue: replacement strategy when a new sector is brought into the disk cache

Least Recently Used (LRU)

- Most commonly used algorithm for 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
- When a block is to be brought in from secondary memory, remove the block that is on the bottom of the stack

Least Frequently Used (LFU)

- The block that has experienced the fewest references is replaced
- A counter is associated with each block
  - incremented each time block is accessed
- When replacement is required, the block with the smallest count is selected



11.8 UNIX SVR4 I/O

- Each individual I/O device is associated with a special file
- Managed by the file system and are read and written in the same manner as user data files
  - provides a clean, uniform interface to users and processes
- The file subsystem serves as the process interface to devices, because these are treated as files
- Types of I/O
  - buffered I/O passes through system buffers
    - system buffer caches
    - character queues
  - unbuffered I/O typically involves the DMA facility
    - the transfer takes place directly between the I/O module and the process I/O area

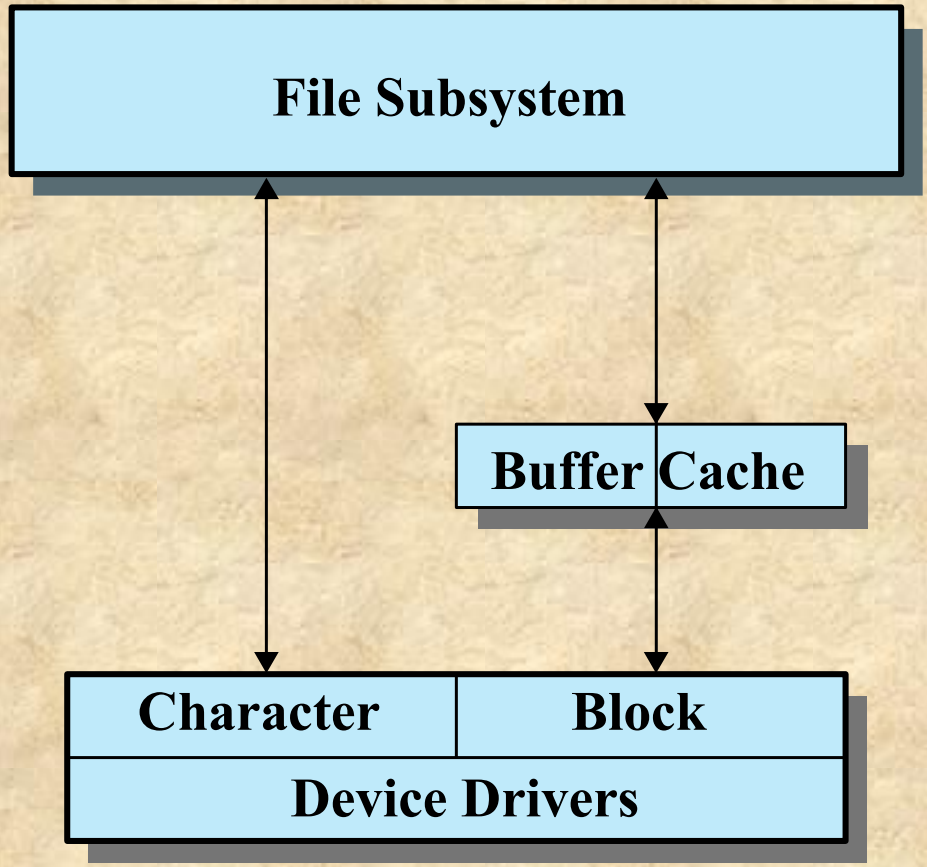


Figure 11.12 UNIX I/O Structure

UNIX Buffer Cache

- A disk cache for block-oriented devices (e.g., disk, USB key)
  - I/O operations with disk are handled through the buffer cache
  - may be read multiple times
- The data transfer between the buffer cache and the user process space always occurs using DMA
  - perform a memory-to-memory copy
  - not use up any processor cycles, but consume bus cycles
- Three lists are maintained for buffer cache management:
  - **free list**: list of all slots (※ one disk sector) in the cache that are available for allocation
  - **device list**: list of all buffers currently associated with each disk
  - **driver I/O queue**: list of buffers that are actually undergoing or waiting for I/O on a particular device

- Hash table contains pointers into the buffer cache
  - each pointer points to the first buffer in the chain
  - each reference to a (device#, block#) maps into a particular entry in the hash table
- A hash pointer associated with each buffer in buffer cache points to the next buffer in the chain for that hash table entry
- For block replacement, a least-recently-used algorithm is used
- The free list preserves this least-recently-used order

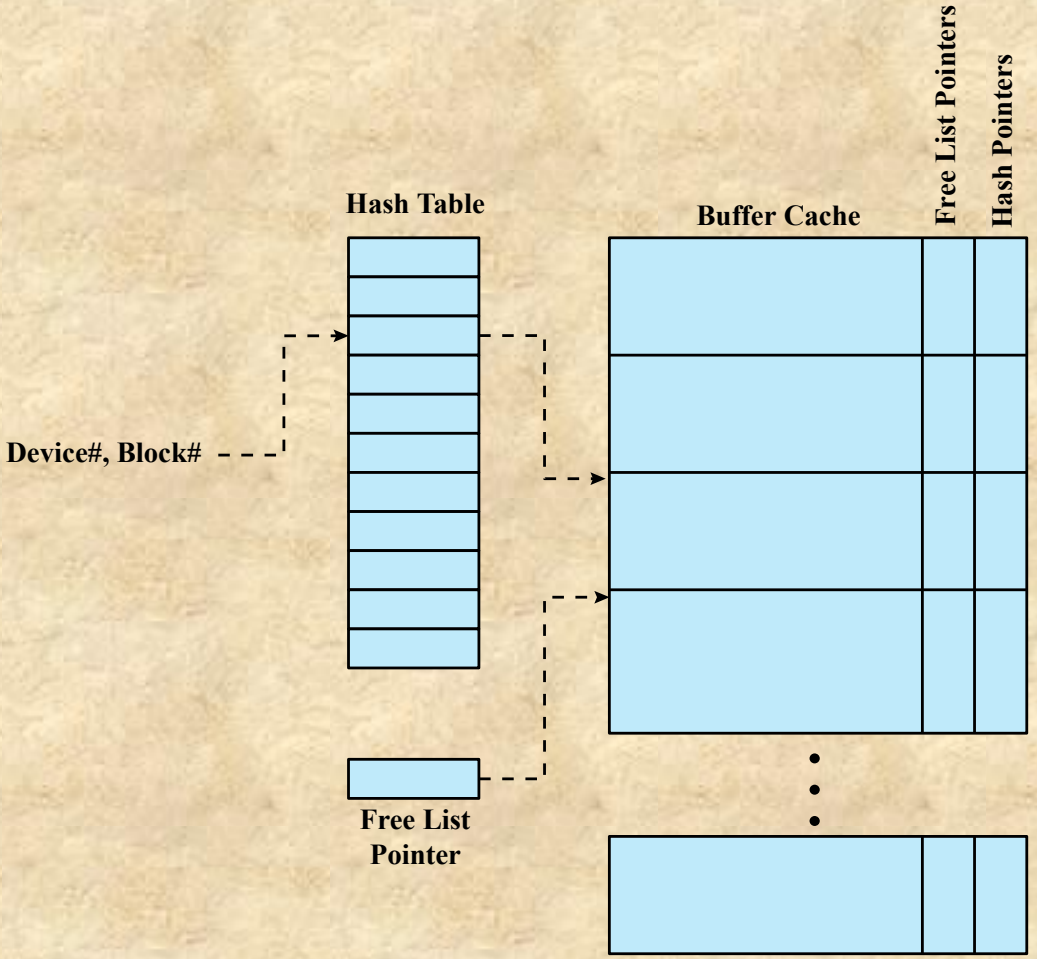


Figure 11.13 UNIX Buffer Cache Organization

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

the producer/consumer model is used

Character queues may only be read once

as each character is read, it is effectively destroyed

Unbuffered I/O

- Simply DMA between device and process space
- Always the fastest method for a process to perform I/O
- A process that is performing unbuffered I/O is locked in main memory and cannot be swapped out
  - reduce the opportunities for swapping by tying up part of main memory, thus reducing the overall system performance
  - I/O device is tied up with the process for the the transfer making it unavailable for other processes

	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers (depending on their speeds)	X		X

Table 11.5 Device I/O in UNIX

11.9 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

The Elevator Scheduler

- Maintain a single queue for disk read and write requests
- Keep the list of requests sorted by block number
- When a new request is added to the queue, four operations are considered in order:

1) If the request is to the same on-disk sector or an immediately adjacent sector to a pending request in the queue, then the existing request and the new request are merged into one request
2) If a request in the queue is sufficiently old, the new request is inserted at the tail of the queue to prevent starvation of a request
3) If there is a suitable location, the new request is inserted in sorted order
4) If there is no suitable location, the new request is placed at the tail of the queue

Deadline Scheduler

- Two problems of the elevator scheme
  - a distance block request can be delayed for a substantial time because the queue is dynamically updated
  - a stream of write requests (e.g., to place a large file on the disk) can block a read request for a considerable time
- Three queues are maintained
  - each incoming request is placed in sorted (elevator) queue
  - the same request is placed at the tail of a read FIFO queue for a read request or a write FIFO queue for a write request
  - associated with each request is an expiration time, with a default value of 0.5 seconds for a read request and 5 seconds for a write request

- Ordinarily, the scheduler dispatches from the sorted queue
  - when a request is satisfied, it is removed from the head of the sorted queue and also from the appropriate FIFO queue
- When the item at the head of one of the FIFO queues becomes older than its expiration time, then the scheduler next dispatches from that FIFO queue, taking the expired request, plus the next few requests from the queue.
  - As each request is dispatched, it is also removed from the sorted queue.
- The deadline I/O scheduler scheme overcomes the starvation problem and also the read versus write problem

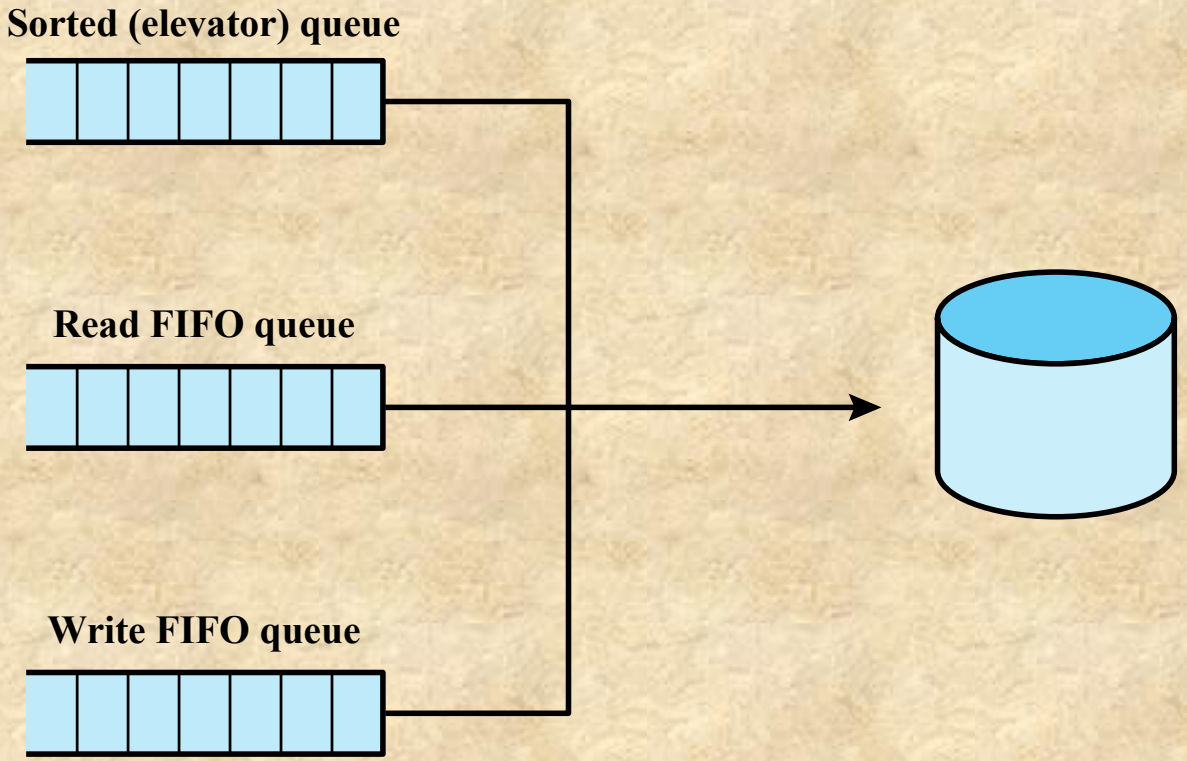


Figure 11.14 The Linux Deadline I/O Scheduler

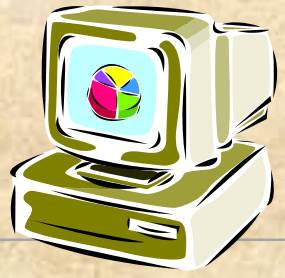
Anticipatory I/O Scheduler

- Elevator and deadline scheduling
  - designed to dispatch a new request as soon as the existing request is satisfied, thus keeping the disk as busy as possible
  - can be counterproductive if there are numerous synchronous read requests
    - an application will wait until a read request is satisfied and the data available before issuing the next request because of the principle of locality
  - the small delay between receiving the data for the last read and issuing the next read enables the scheduler to turn elsewhere for a pending request and dispatch that request
  - if the scheduler were to delay a short period of time after satisfying a read request, to see if a new nearby read request is made, the overall performance of the system could be enhanced

- The anticipatory scheduler 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 due to the principle of locality
    - if so, that request will be serviced immediately
    - if no such read request occurs, the scheduler resumes using the deadline scheduling algorithm

Linux Page Cache

- In Linux 2.2 and earlier release,
  - a page cache for reads and writes from regular file, system files and for virtual memory pages
  - a separate buffer cache for block I/O
- For Linux 2.4 and later, maintain a single unified page cache for all traffic between disk and main memory
- Two benefits:
  - pages in the page cache are likely to be referenced again due to the principle of temporal locality, thus saving a disk I/O operation
  - dirty pages can be collected and written out efficiently



11.10 Windows I/O

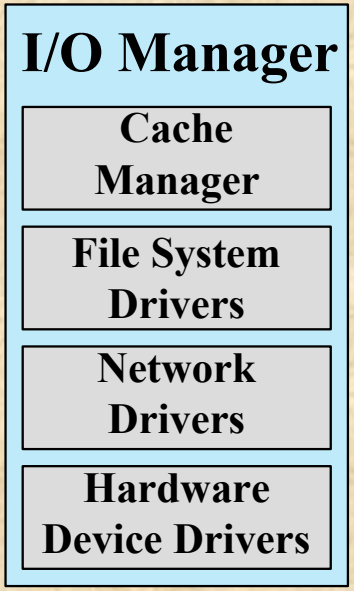
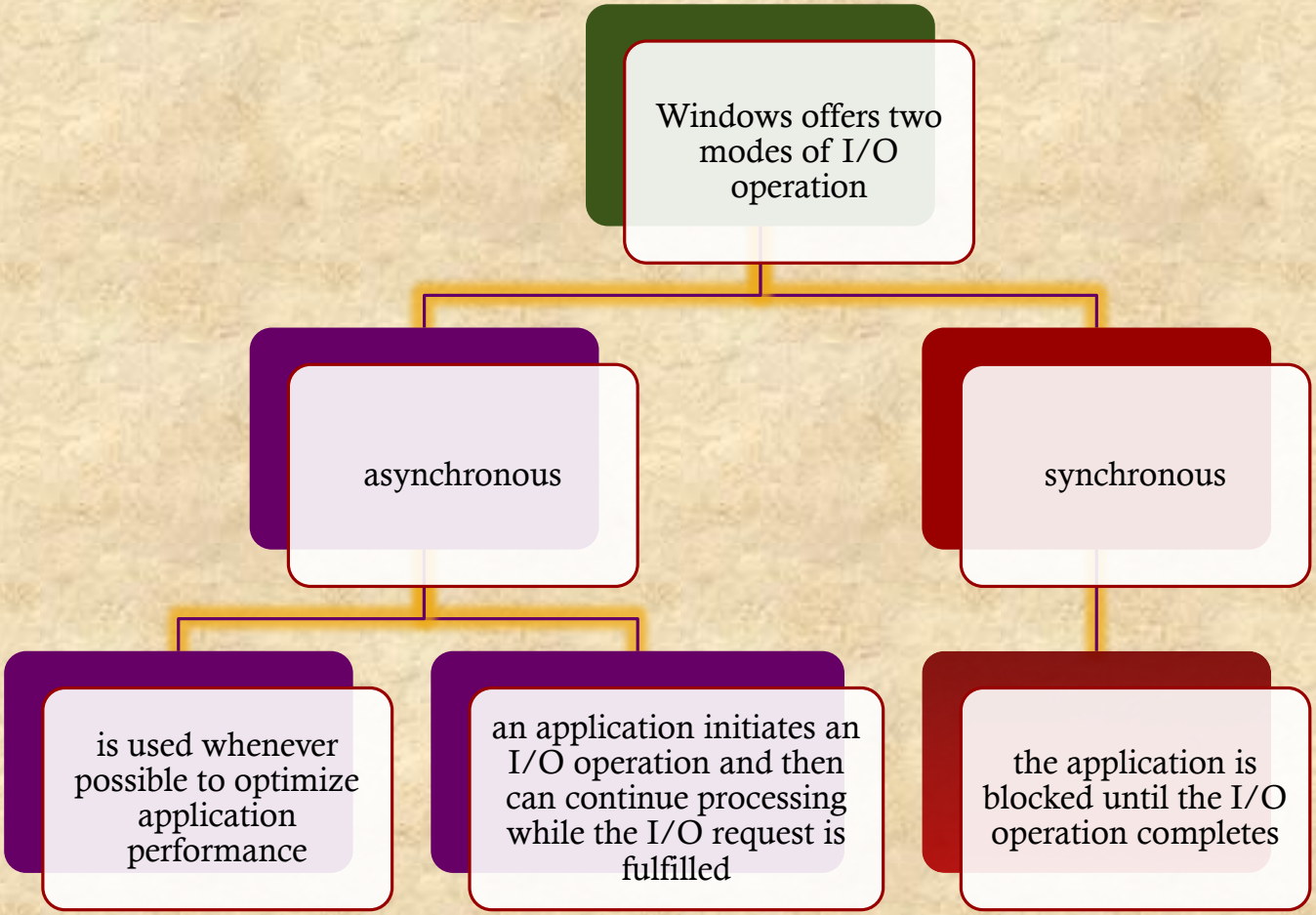


Figure 11.15 Windows I/O Manager

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 devices
- Organization of the I/O function
  - The evolution of the I/O function
  - Direct memory access
- Operating system design issues
  - Design objectives
  - Logical structure of the I/O function
- I/O Buffering
  - Single buffer
  - Double buffer
  - Circular buffer
- Disk scheduling
  - Disk performance parameters
  - Disk scheduling policies
- Raid
- Raid levels 0 – 6
- Disk cache
  - Design and performance considerations
- UNIX SVR4 I/O
  - Buffer cache
  - Character queue
  - Unbuffered I/O
  - UNIX devices
- Linux I/O
  - Disk scheduling
  - Linux page cache
- Windows I/O
  - Basic I/O facilities
  - Asynchronous and Synchronous I/O
  - Software RAID
  - Volume shadow copies/encryption