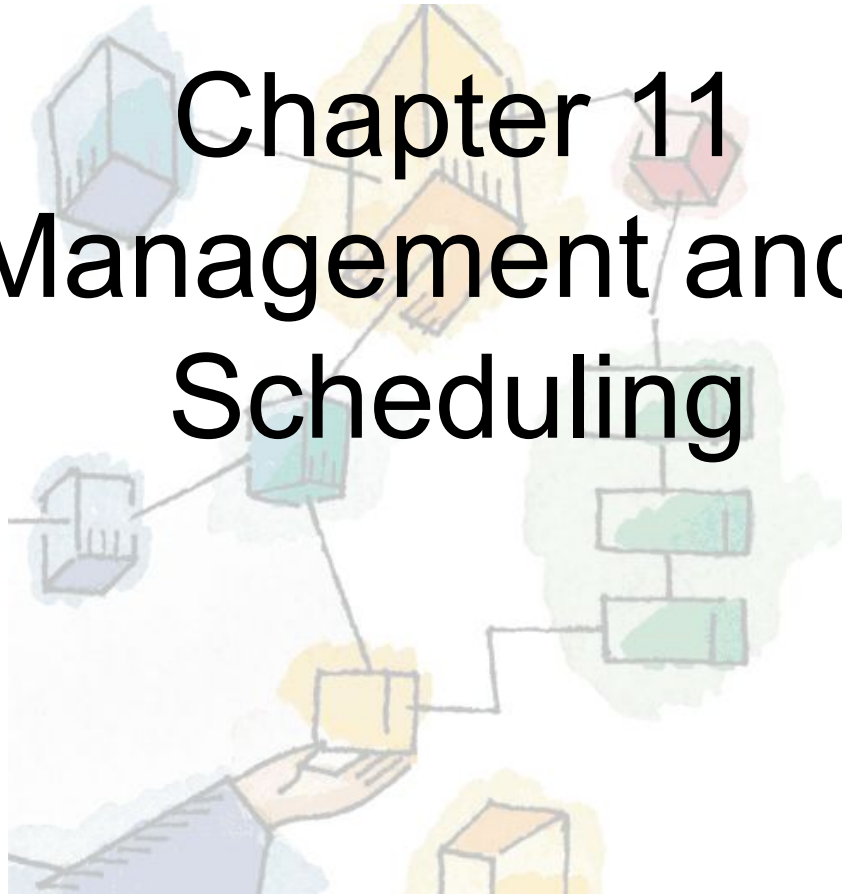


*Operating Systems:  
Internals and Design Principles, 6/E*  
William Stallings

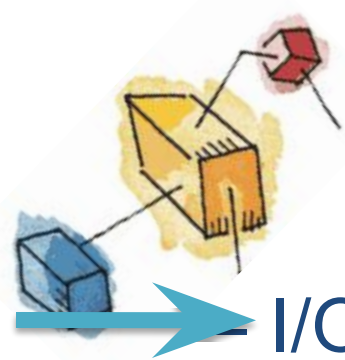
# Chapter 11

## I/O Management and Disk Scheduling



Dave Bremer  
Otago Polytechnic, NZ  
©2008, Prentice Hall

# Roadmap



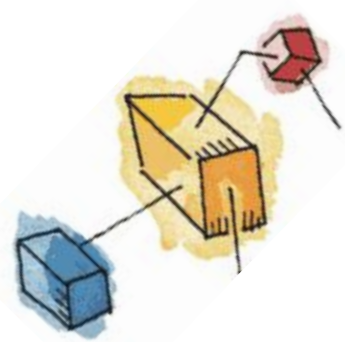
## I/O Devices

- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O



# Categories of I/O Devices

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

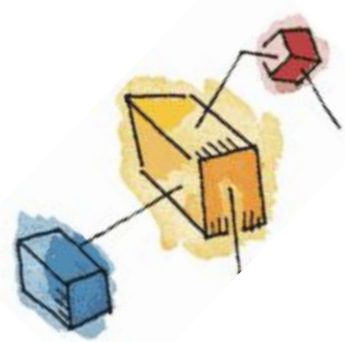




# Human readable

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





# Machine readable

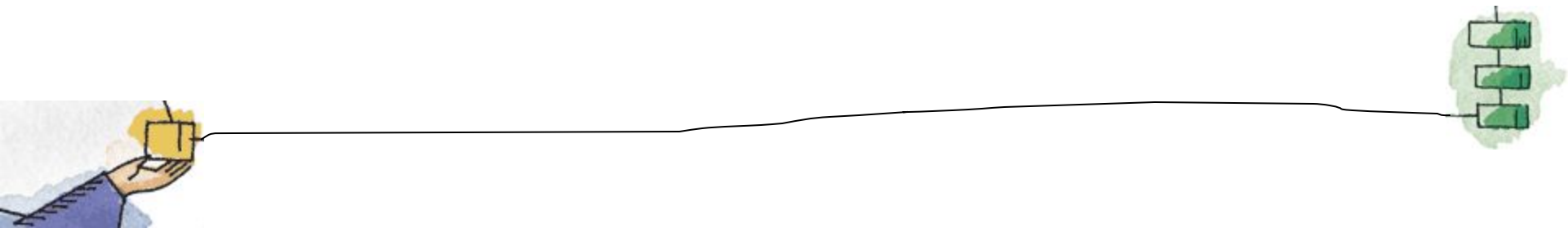
- Used to communicate with electronic equipment
  - Disk drives
  - USB keys
  - Sensors
  - Controllers
  - Actuators





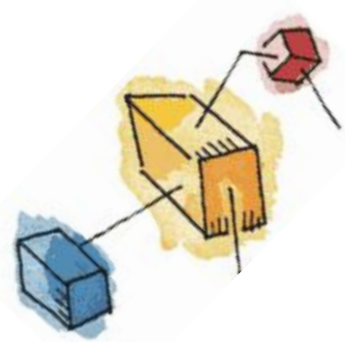
# Communication

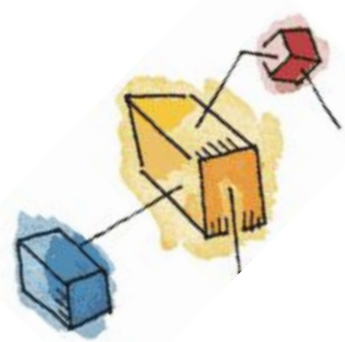
- Used to communicate with remote devices
  - Digital line drivers
  - Modems



# Differences in I/O Devices

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





# Data Rate

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

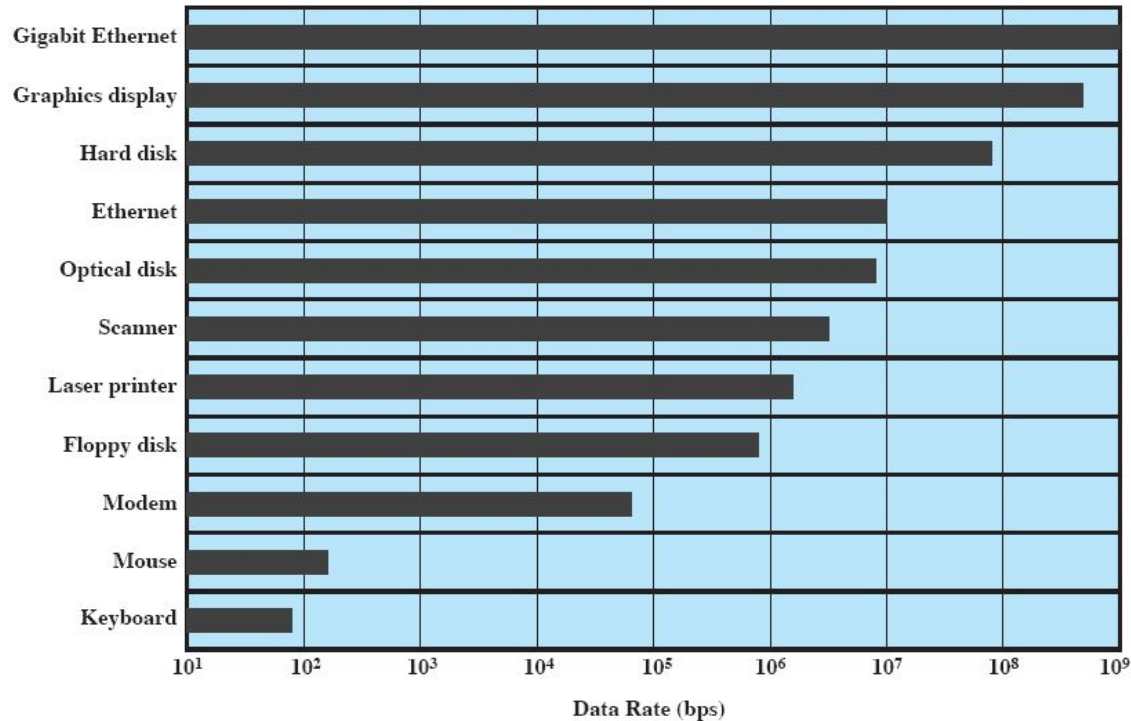


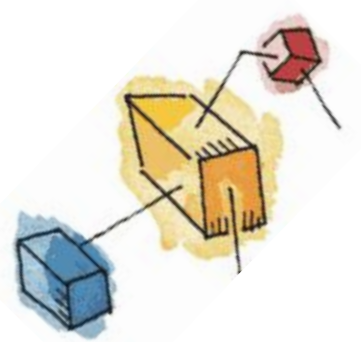
Figure 11.1 Typical I/O Device Data Rates

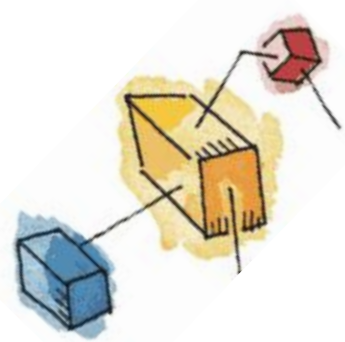




# Application

- Disk used to store files requires file management software
- Disk used to store virtual memory pages needs special hardware and software to support it
- Terminal used by system administrator may have a higher priority





# Complexity of control

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





# Unit of transfer

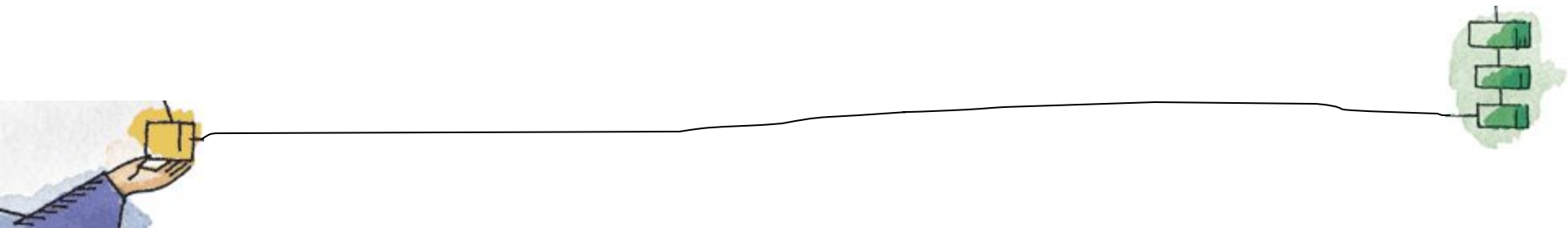
- Data may be transferred as
  - a stream of bytes or characters (e.g., terminal I/O)
  - or in larger blocks (e.g., disk I/O).





# Data representation

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





# Error Conditions

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



# Roadmap



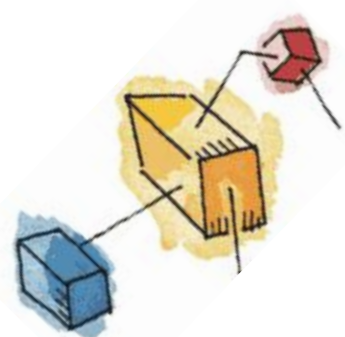
- I/O Devices



Organization of the I/O Function

- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O



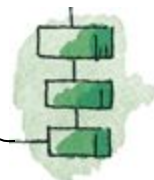


# Techniques for performing I/O

- Programmed I/O
- Interrupt-driven I/O
- Direct memory access (DMA)

**Table 11.1 I/O Techniques**

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



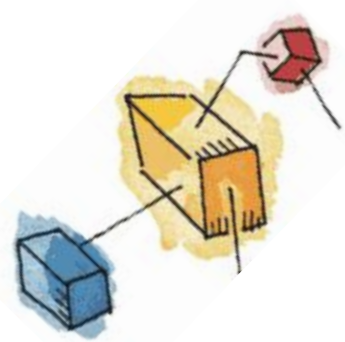
# Evolution of the I/O Function

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





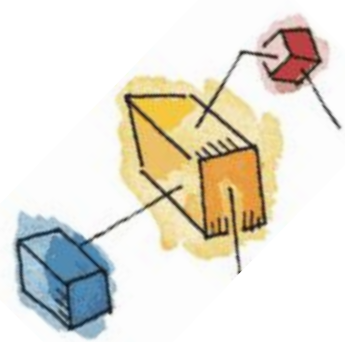
# Evolution of the I/O Function cont...



3. Controller or I/O module with interrupts
  - Efficiency improves as processor does not spend time waiting for an I/O operation to be performed
4. Direct Memory Access
  - Blocks of data are moved into memory without involving the processor
  - Processor involved at beginning and end only

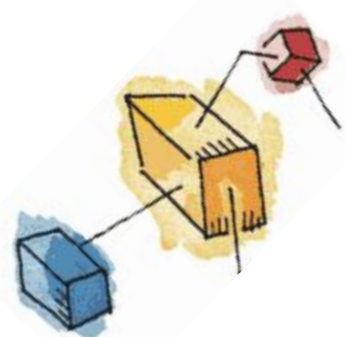


# Evolution of the I/O Function cont...



5. I/O module is a separate processor
  - CPU directs the I/O processor to execute an I/O program in main memory.
6. I/O processor
  - I/O module has its own local memory
  - Commonly used to control communications with interactive terminals





# Direct Memory Address

- Processor delegates I/O operation to the DMA module
- DMA module transfers data directly to or from memory
- When complete DMA module sends an interrupt signal to the processor

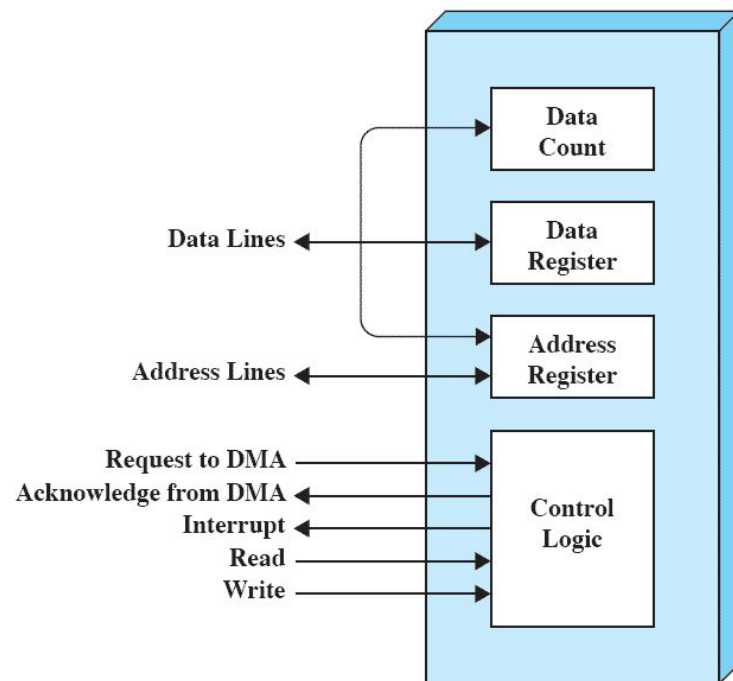
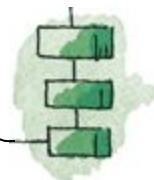
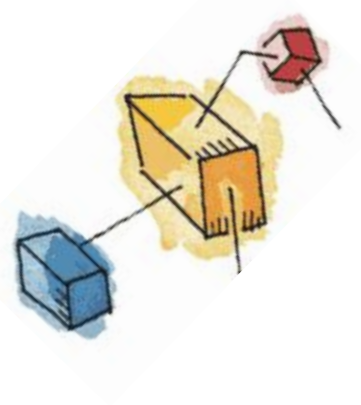
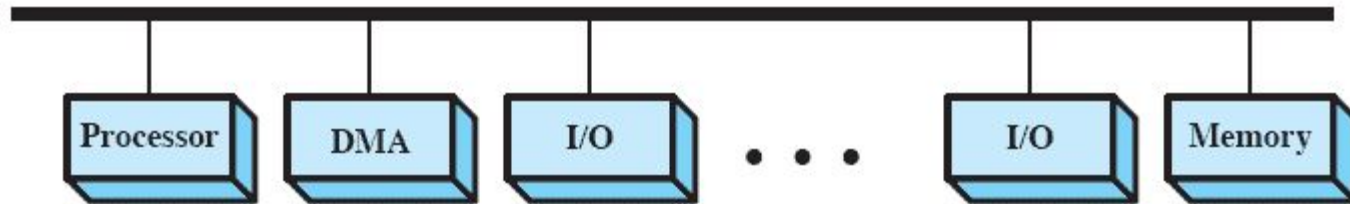


Figure 11.2 Typical DMA Block Diagram





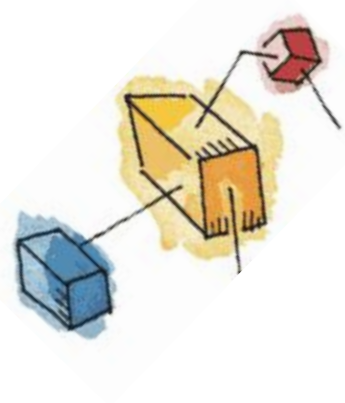
# DMA Configurations: Single Bus



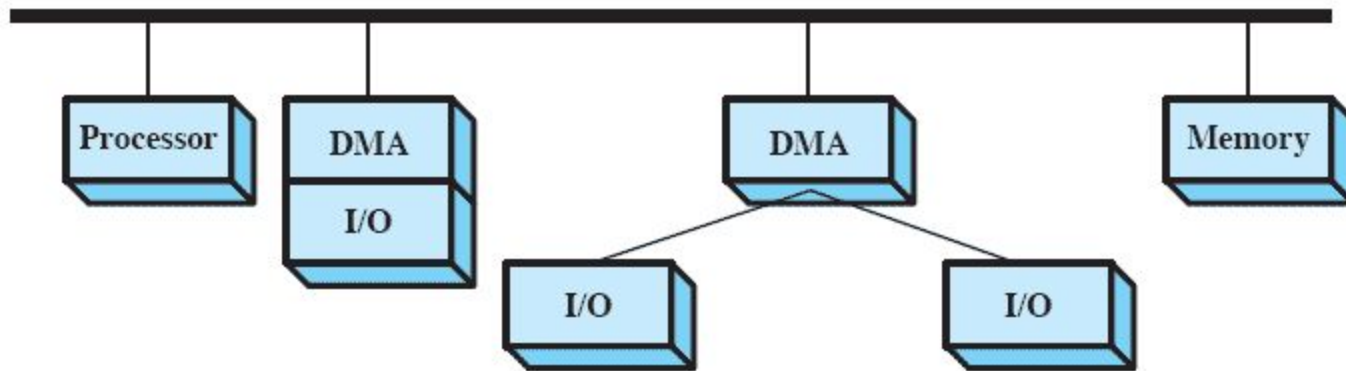
(a) Single-bus, detached DMA

- DMA can be configured in several ways
- Shown here, all modules share the same system bus





# DMA Configurations: Integrated DMA & I/O

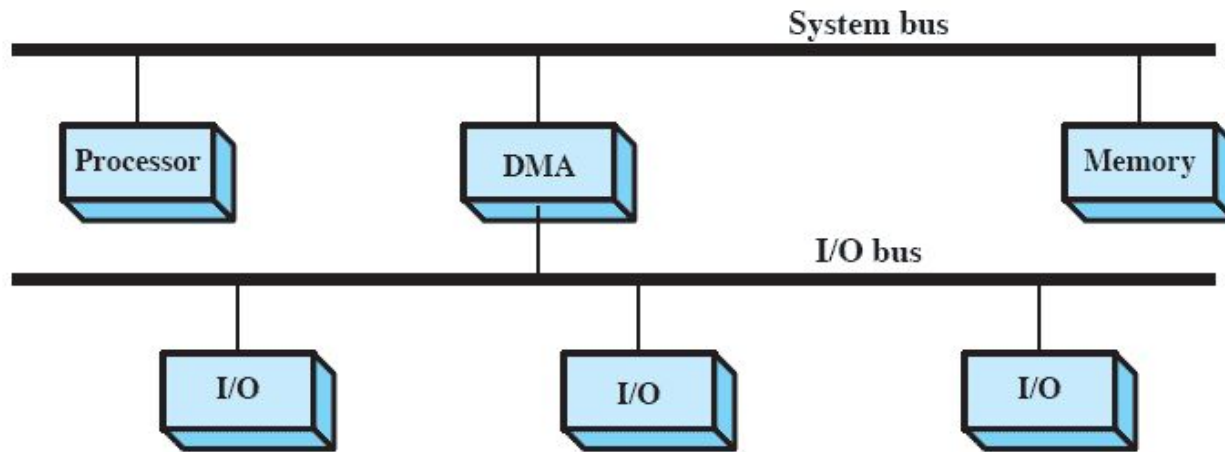


(b) Single-bus, Integrated DMA-I/O

- Direct Path between DMA and I/O modules
- This substantially cuts the required bus cycles



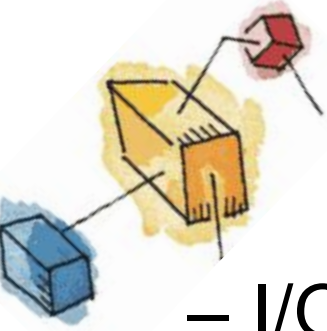

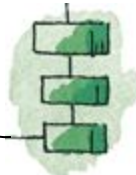
# DMA Configurations: I/O Bus



(c) I/O bus

- Reduces the number of I/O interfaces in the DMA module

# Roadmap

- 
- I/O Devices
  - Organization of the I/O Function
  - Operating System Design Issues
    - I/O Buffering
    - Disk Scheduling
    - Raid
    - Disk Cache
    - UNIX SVR4 I/O
    - LINUX I/O
    - Windows I/O
- 
- 



# Goals: Efficiency

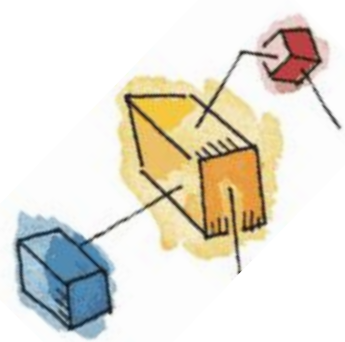
- Most I/O devices extremely slow compared to main memory
- Use of multiprogramming allows for some processes to be waiting on I/O while another process executes
- I/O cannot keep up with processor speed
  - Swapping used to bring in ready processes
  - But this is an I/O operation itself





# Generality

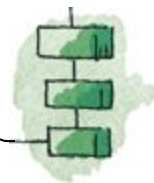
- For simplicity and freedom from error it is desirable to handle all I/O devices in a uniform manner
- Hide most of the details of device I/O in lower-level routines
- Difficult to completely generalize, but can use a hierarchical modular design of I/O functions





# Hierarchical design

- A hierarchical philosophy leads to organizing an OS into layers
- Each layer relies on the next lower layer to perform more primitive functions
- It provides services to the next higher layer.
- Changes in one layer should not require changes in other layers



# Local peripheral device

- Logical I/O:
  - Deals with the device as a logical resource
- Device I/O:
  - Converts requested operations into sequence of I/O instructions
- Scheduling and Control
  - Performs actual queuing and control operations

User  
Processes

Logical  
I/O

Device  
I/O

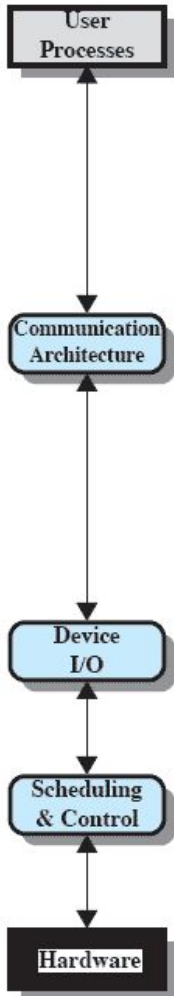
Scheduling  
& Control

Hardware

(a) Local peripheral device

# Communications Port

- Similar to previous but the logical I/O module is replaced by a communications architecture,
  - This consist of a number of layers.
  - An example is TCP/IP,



(b) Communications port



# File System

- Directory management
  - Concerned with user operations affecting files
- File System
  - Logical structure and operations
- Physical organisation]
  - Converts logical names to physical addresses



(c) File system





# Roadmap

- I/O Devices
- Organization of the I/O Function
- Operating System Design Issues



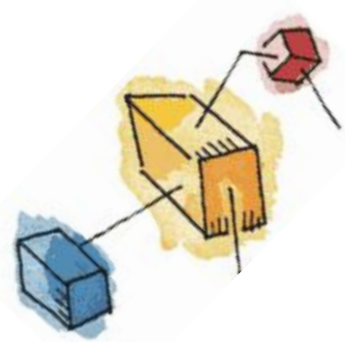
## I/O Buffering

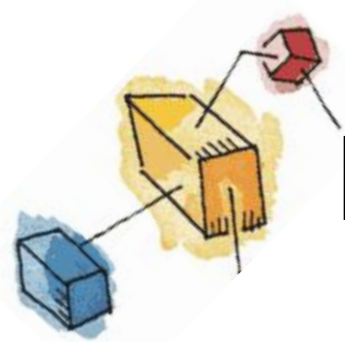
- Disk Scheduling
- Raid
- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O



# I/O Buffering

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





# Block-oriented Buffering

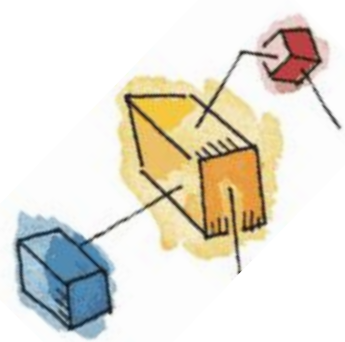
- Information is stored in fixed sized blocks
- Transfers are made a block at a time
  - Can reference data by block number
- Used for disks and USB keys





# Stream-Oriented Buffering

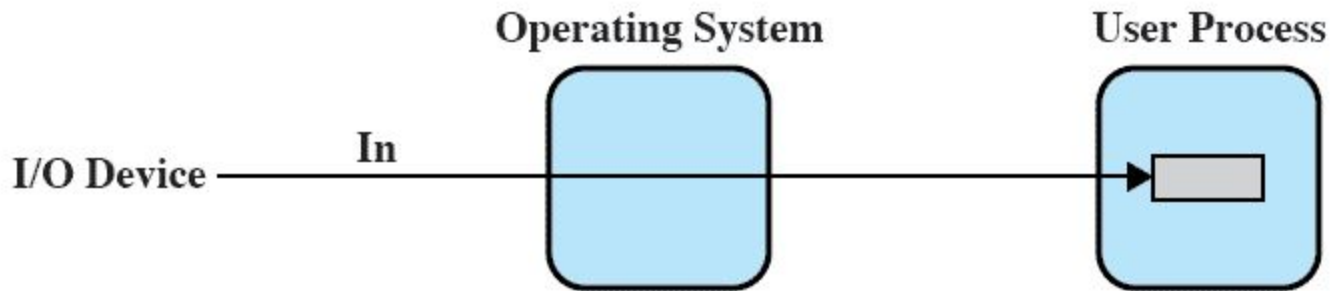
- Transfer information as a stream of bytes
- Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage





# No Buffer

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

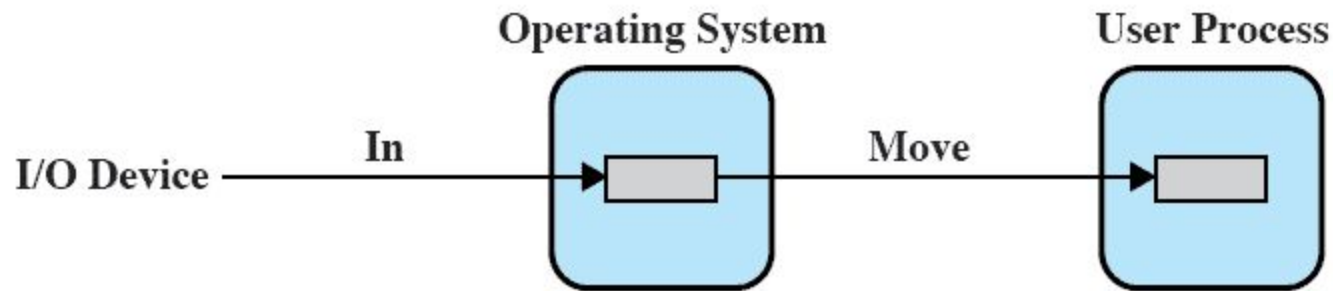


(a) No buffering



# Single Buffer

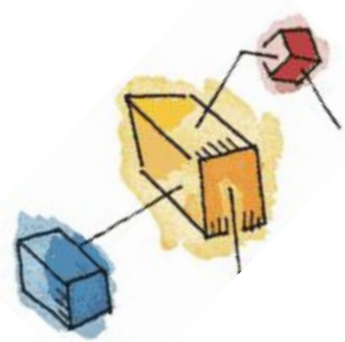
- Operating system assigns a buffer in main memory for an I/O request



(b) Single buffering

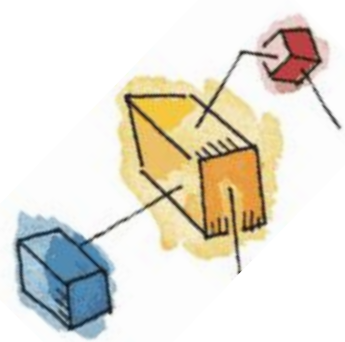
# Block Oriented Single Buffer

- Input transfers made to buffer
- Block moved to user space when needed
- The next block is moved into the buffer
  - *Read ahead or Anticipated Input*
- Often a reasonable assumption as data is usually accessed sequentially



# Stream-oriented Single Buffer

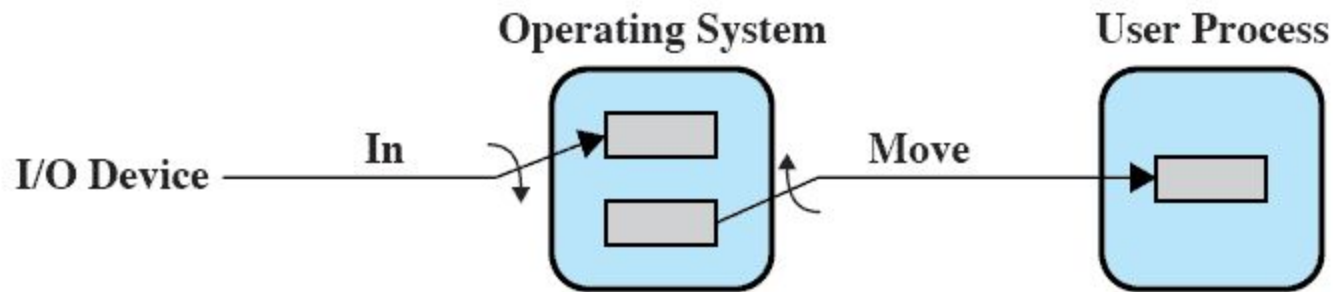
- Line-at-time or Byte-at-a-time
- Terminals often deal with one line at a time with carriage return signaling the end of the line
- Byte-at-a-time suites devices where a single keystroke may be significant
  - Also sensors and controllers





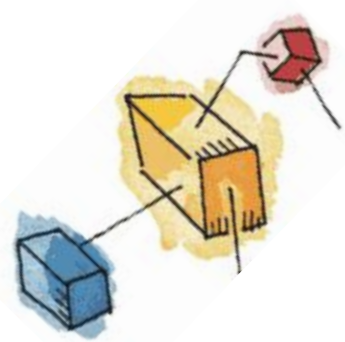
# Double Buffer

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



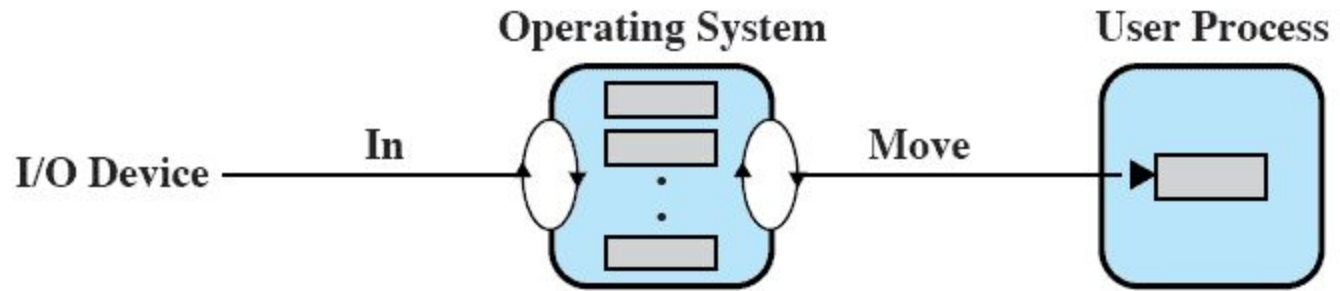
(c) Double buffering



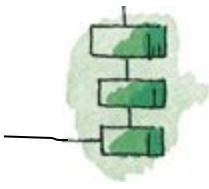


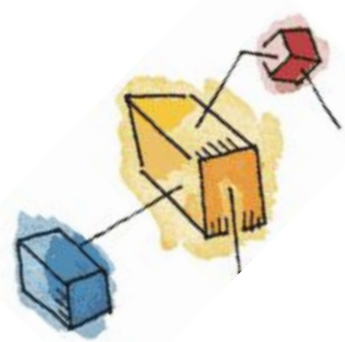
# Circular Buffer

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



(d) Circular buffering





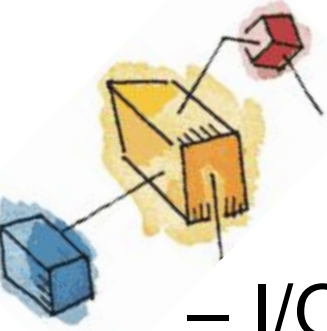
# Buffer Limitations

- Buffering smoothes out peaks in I/O demand.
  - But with enough demand eventually all buffers become full and their advantage is lost
- However, 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.





# Roadmap

- 
- I/O Devices
  - Organization of the I/O Function
  - Operating System Design Issues
  - I/O Buffering



## Disk Scheduling

- Raid
- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O





# Disk Performance Parameters

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

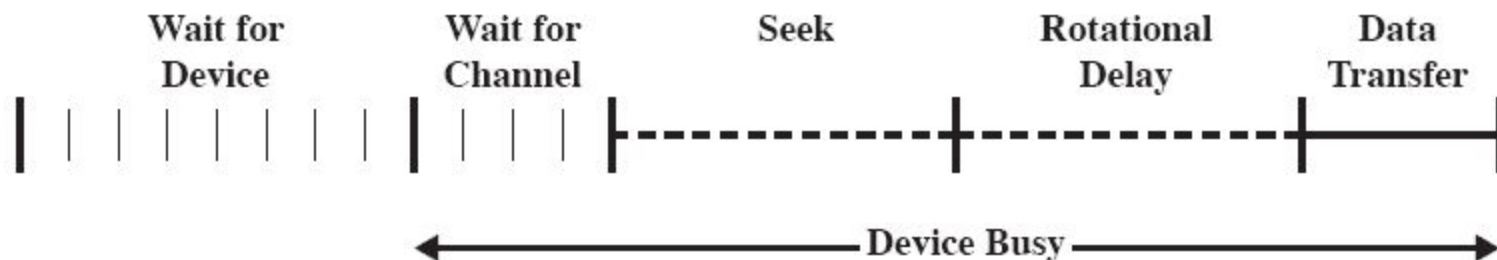
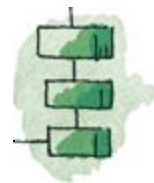
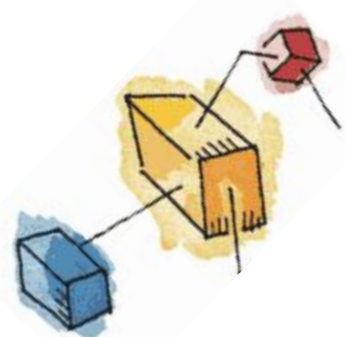


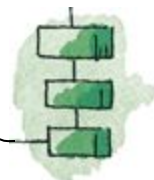
Figure 11.6 Timing of a Disk I/O Transfer

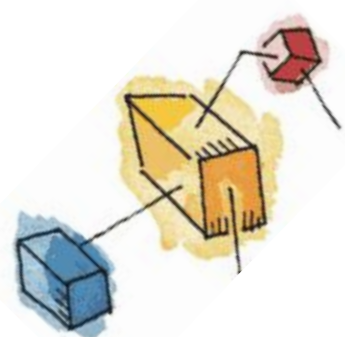




# Positioning the Read/Write Heads

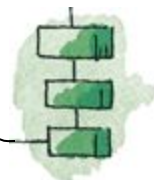
- When the disk drive is operating, the disk is rotating at constant speed.
- Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system.





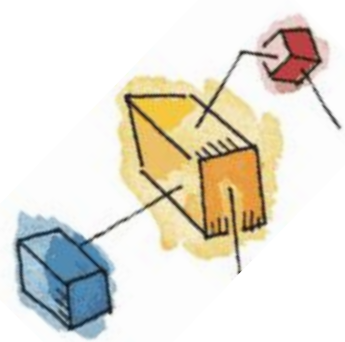
# Disk Performance Parameters

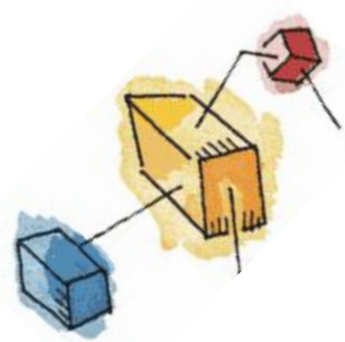
- **Access Time** is the sum of:
  - **Seek time:** The time it takes to position the head at the desired track
  - **Rotational delay** or **rotational latency:** The time it takes for the beginning of the sector to reach the head
- **Transfer Time** is the time taken to transfer the data.



# Disk Scheduling Policies

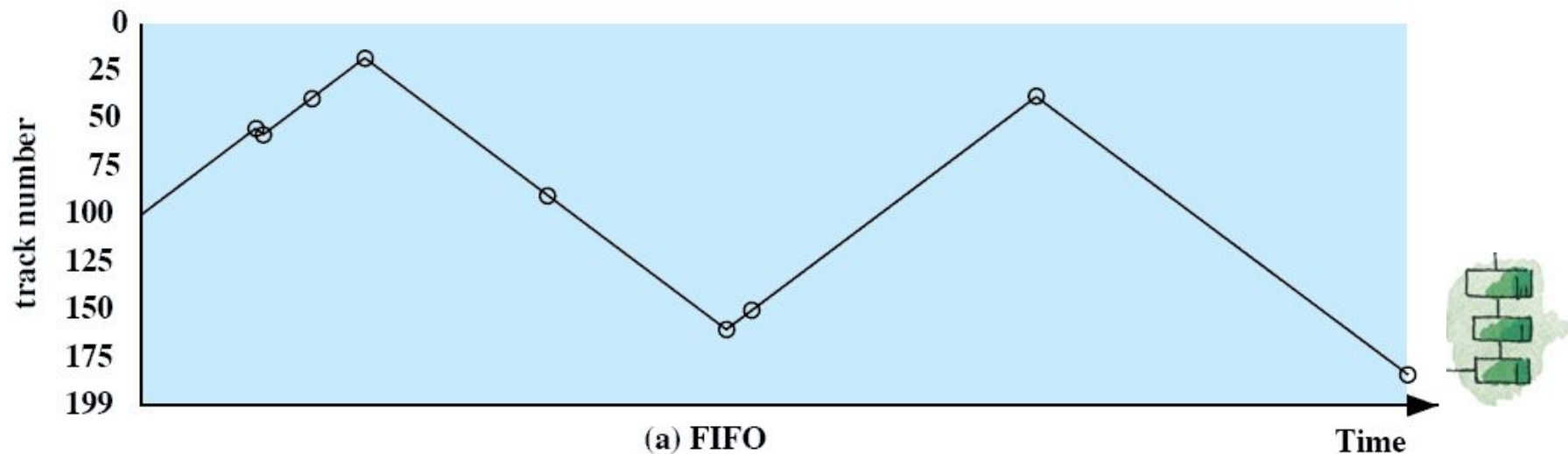
- To compare various schemes, consider a disk head is initially located at track 100.
  - assume a disk with 200 tracks and that the disk request queue has random requests in it.
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.

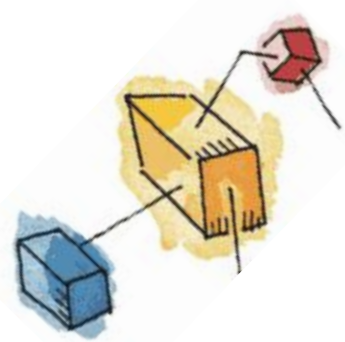




# First-in, first-out (FIFO)

- Process request sequentially
- Fair to all processes
- Approaches random scheduling in performance if there are many processes

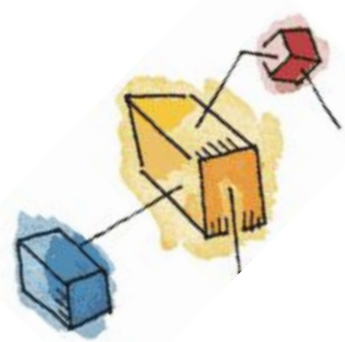




# Priority

- Goal is not to optimize disk use but to meet other objectives
- Short batch jobs may have higher priority
- Provide good interactive response time
- Longer jobs may have to wait an excessively long time
- A poor policy for database systems





# Last-in, first-out

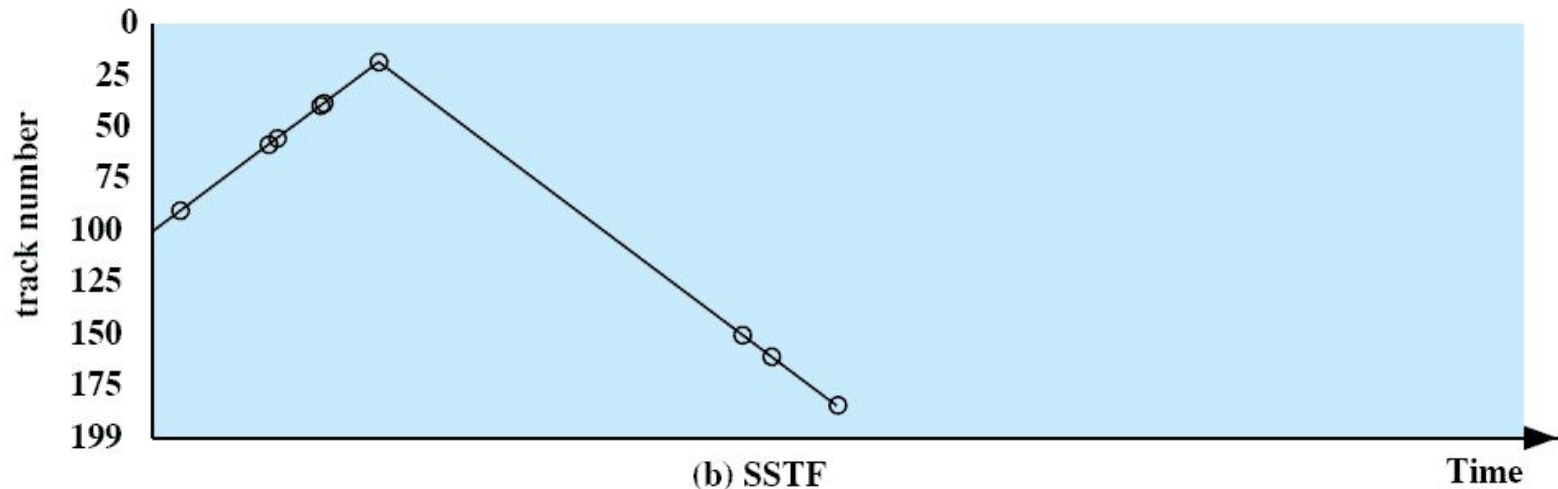
- Good for transaction processing systems
  - The device is given to the most recent user so there should be little arm movement
- Possibility of starvation since a job may never regain the head of the line

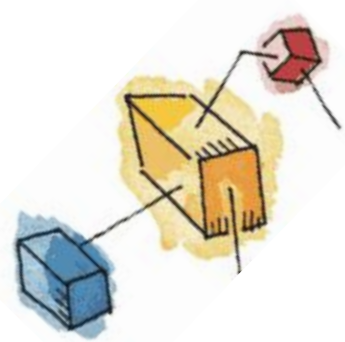




# Shortest Service Time First

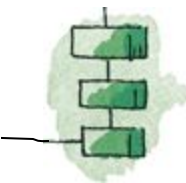
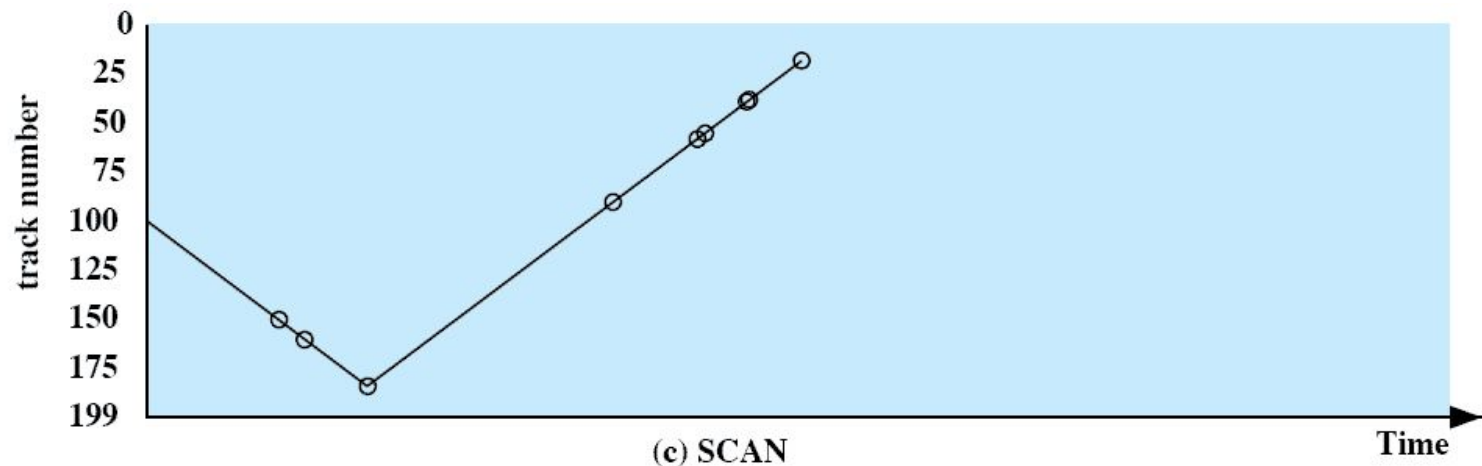
- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time

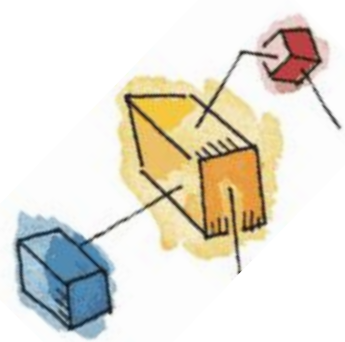




# SCAN

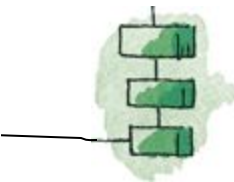
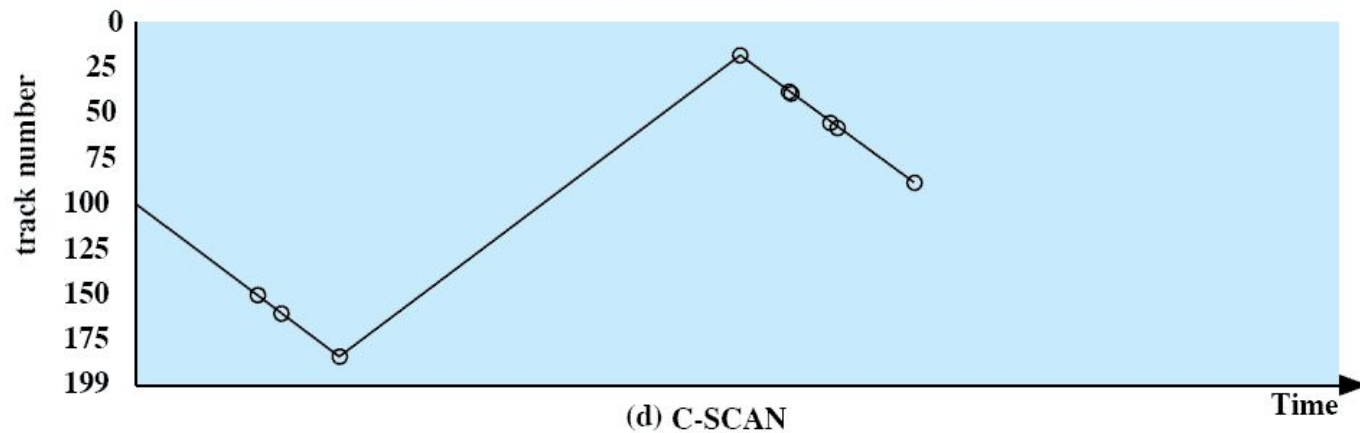
- Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction then the direction is reversed

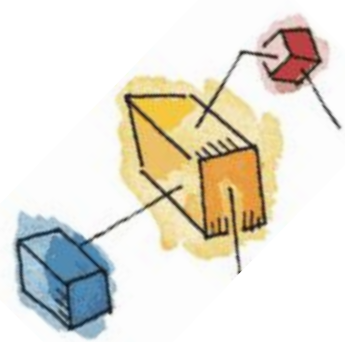




# C-SCAN

- 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





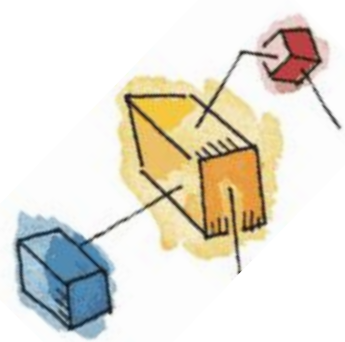
# N-step-SCAN

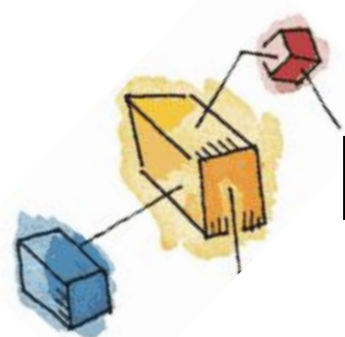
- Segments the disk request queue into subqueues of length  $N$
- Subqueues are processed one at a time, using SCAN
- New requests added to other queue when queue is processed



# FSCAN

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





# Performance Compared

## Comparison of Disk Scheduling Algorithms

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







# Disk Scheduling Algorithms

**Table 11.3** Disk Scheduling Algorithms

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





# Roadmap

- I/O Devices
- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling

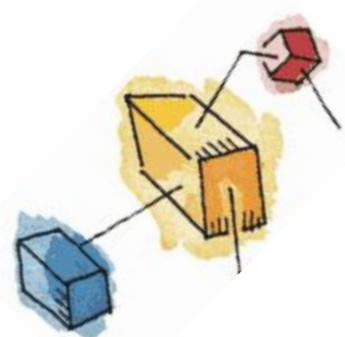


## Raid

- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O

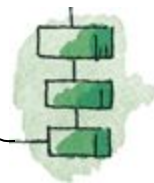






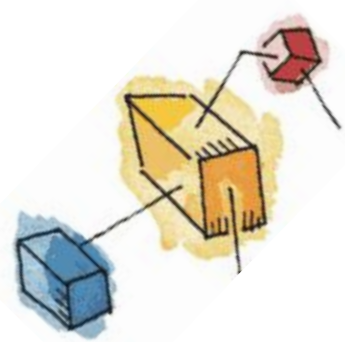
# Multiple Disks

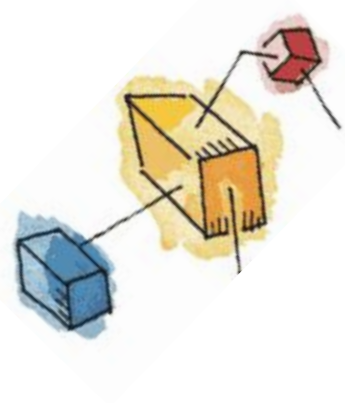
- Disk I/O performance may be increased by spreading the operation over multiple read/write heads
  - Or multiple disks
- Disk failures can be recovered if parity information is stored



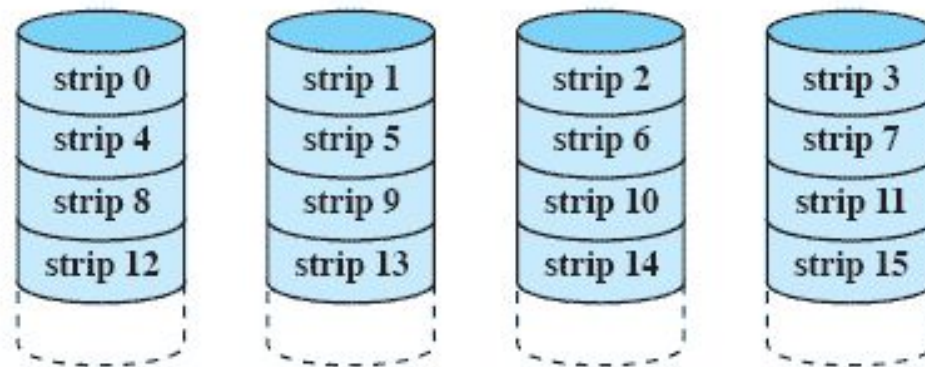
# RAID

- Redundant Array of Independent Disks
- 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
- Redundant disk capacity is used to store parity information which provides recoverability from disk failure



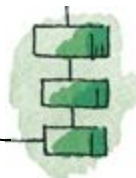


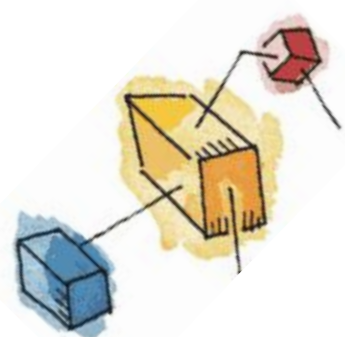
# RAID 0 - Stripped



(a) RAID 0 (non-redundant)

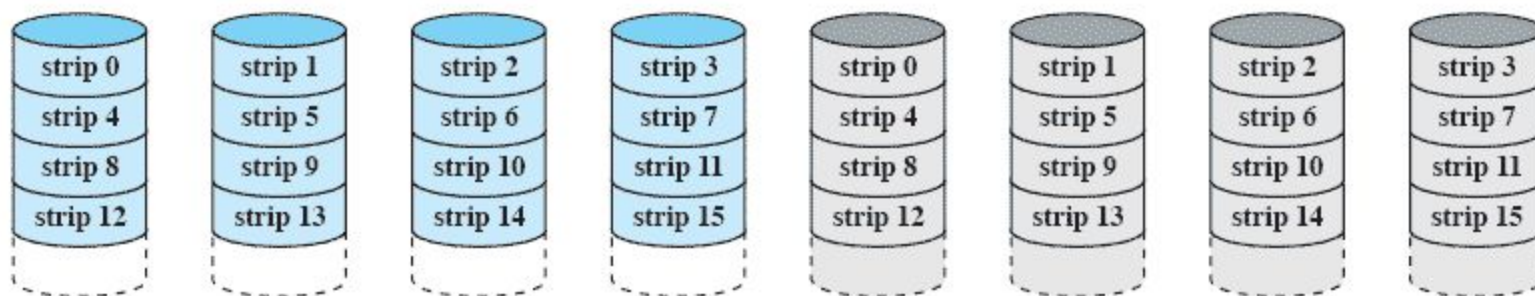
- Not a true RAID – no redundancy
- Disk failure is catastrophic
- Very fast due to parallel read/write





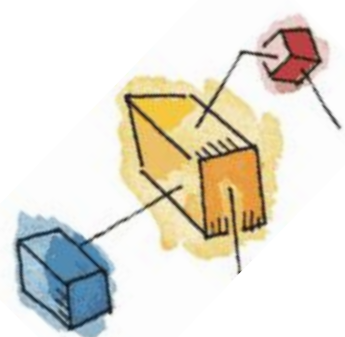
# RAID 1 - Mirrored

- Redundancy through duplication instead of parity.
- Read requests can be made in parallel.
- Simple recovery from disk failure



(b) RAID 1 (mirrored)

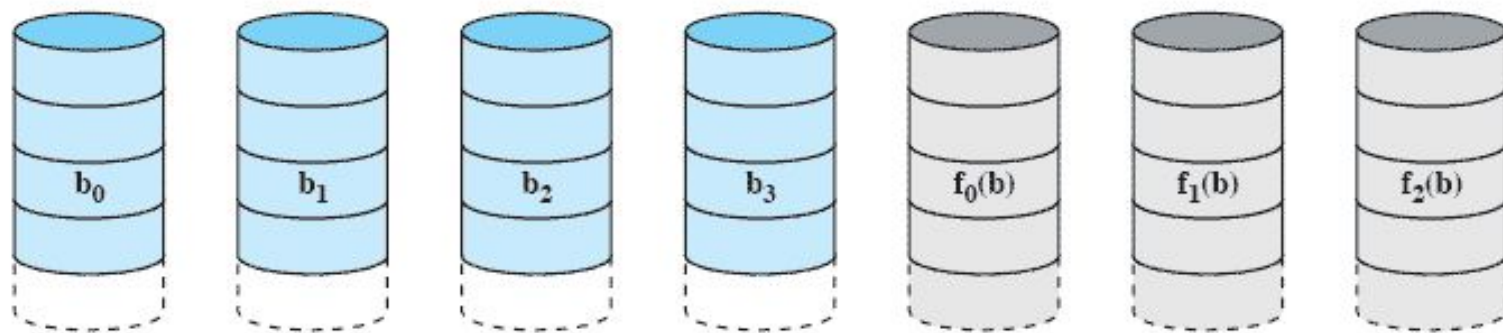




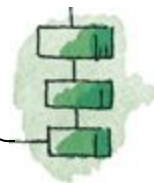
# RAID 2

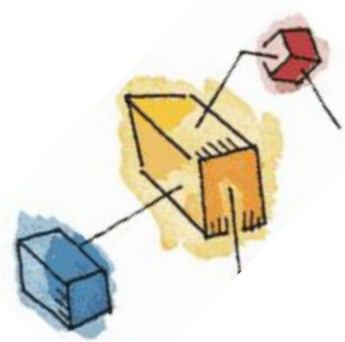
## (Using Hamming code)

- Synchronised disk rotation
- Data stripping is used (extremely small)
- Hamming code used to correct single bit errors and detect double-bit errors



(c) RAID 2 (redundancy through Hamming code)

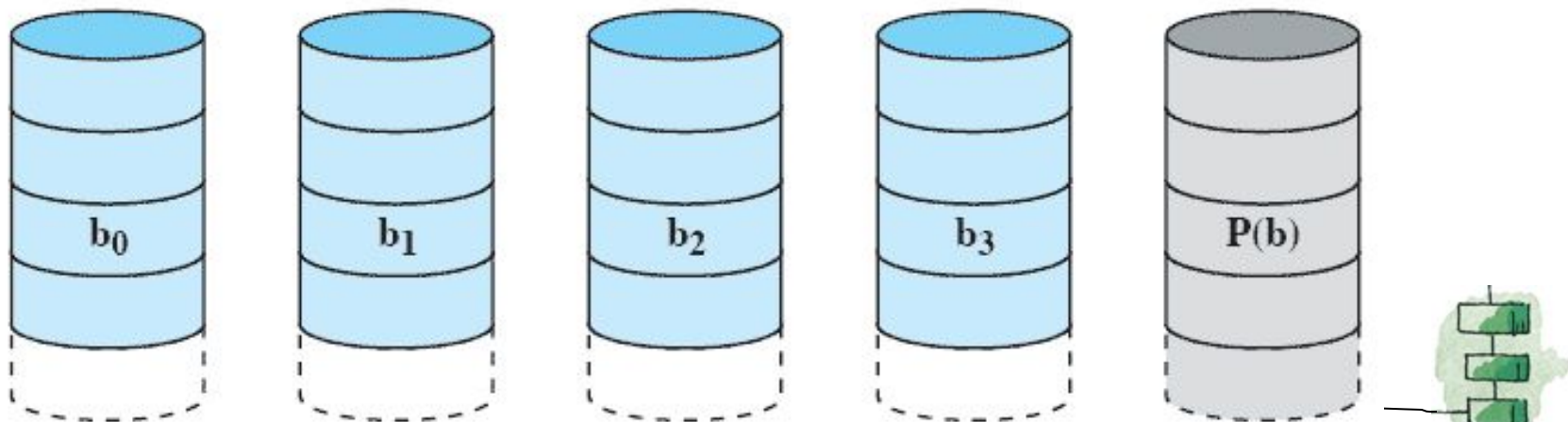




# RAID 3

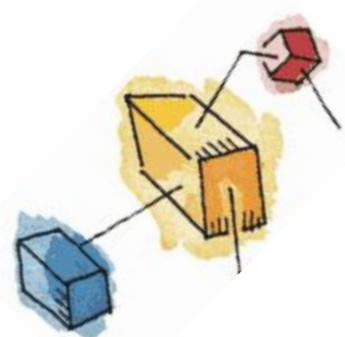
## bit-interleaved parity

- Similar to RAID-2 but uses all parity bits stored on a single drive



(d) RAID 3 (bit-interleaved parity)

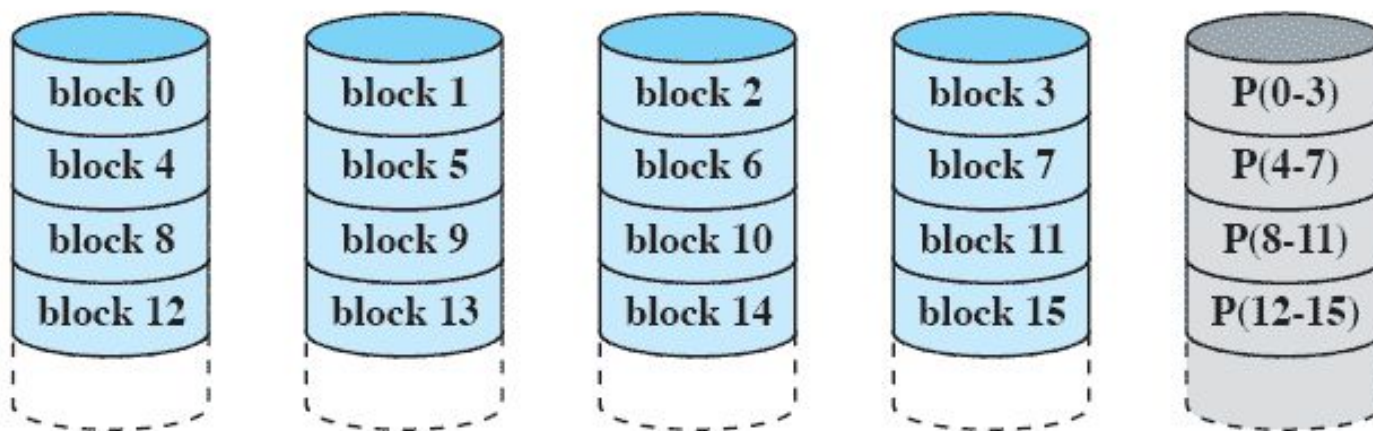




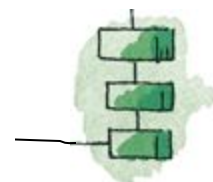
# RAID 4

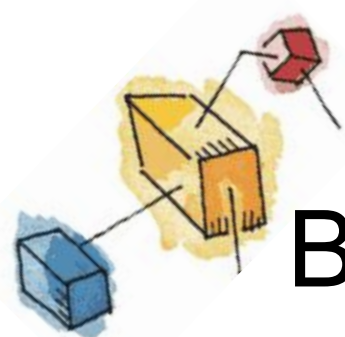
## Block-level parity

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



(e) RAID 4 (block-level parity)

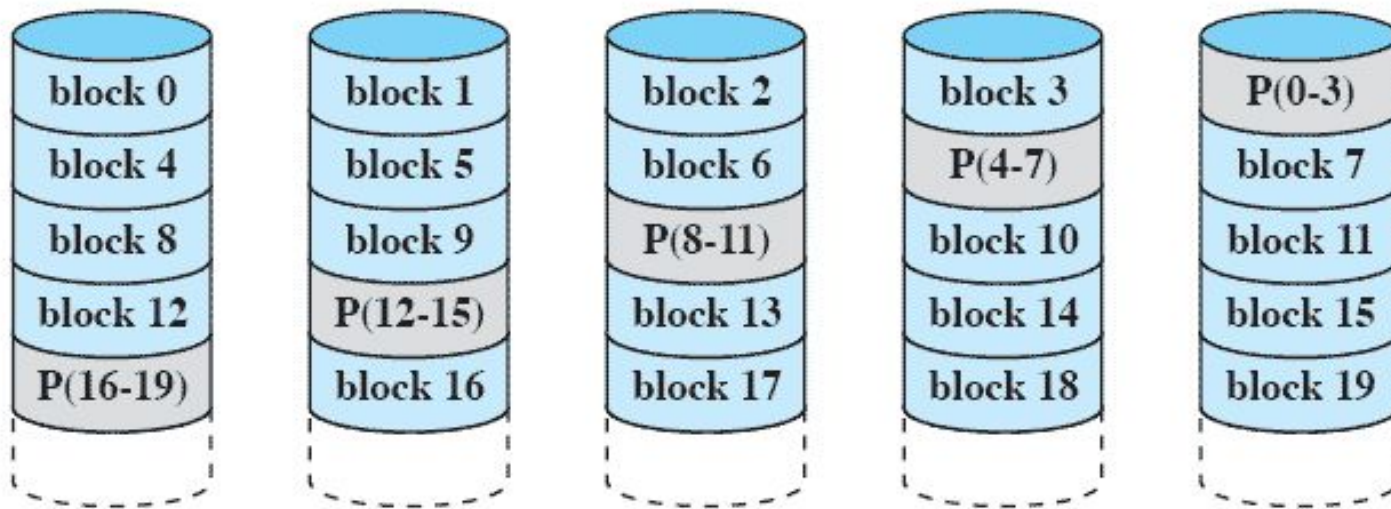




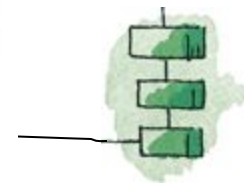
# RAID 5

## Block-level Distributed parity

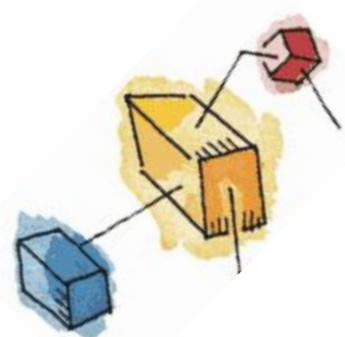
- Similar to RAID-4 but distributing the parity bits across all drives



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



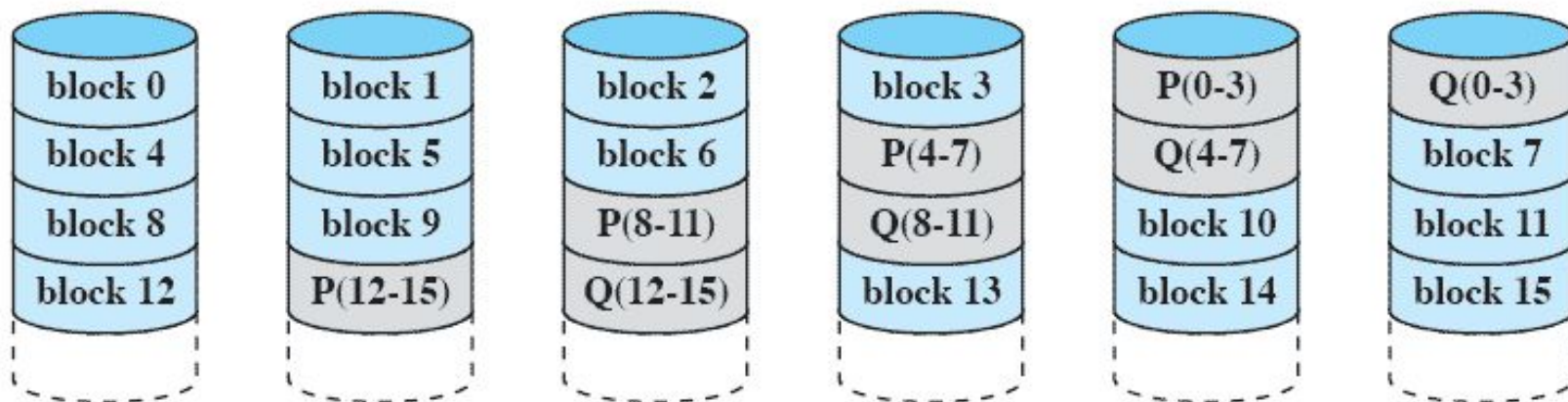




# RAID 6

## Dual Redundancy

- Two different parity calculations are carried out
  - stored in separate blocks on different disks.
- Can recover from two disks failing



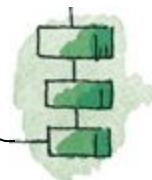
(g) RAID 6 (dual redundancy)





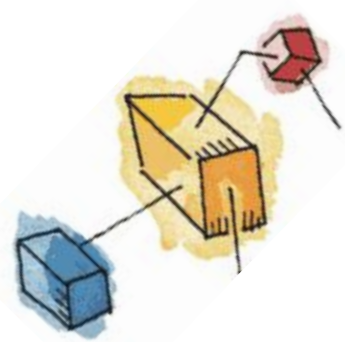
-  Disk Cache

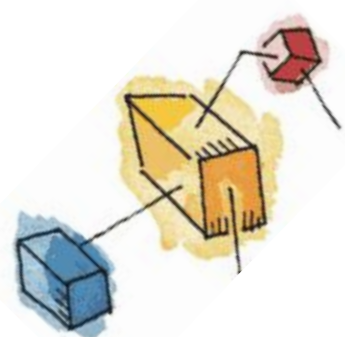
- 



# Disk Cache

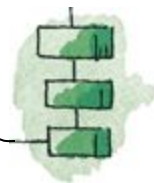
- Buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk
- When an I/O request is made for a particular sector,
  - a check is made to determine if the sector is in the disk cache.
- A number of ways exist to populate the cache

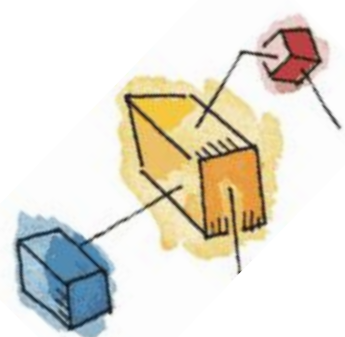




# Least Recently Used

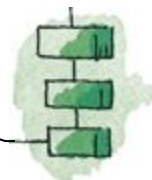
- The block that has been in the cache the longest with no reference to it is replaced
- A stack of pointers reference the cache
  - Most recently referenced block is on the top of the stack
  - When a block is referenced or brought into the cache, it is placed on the top of the stack



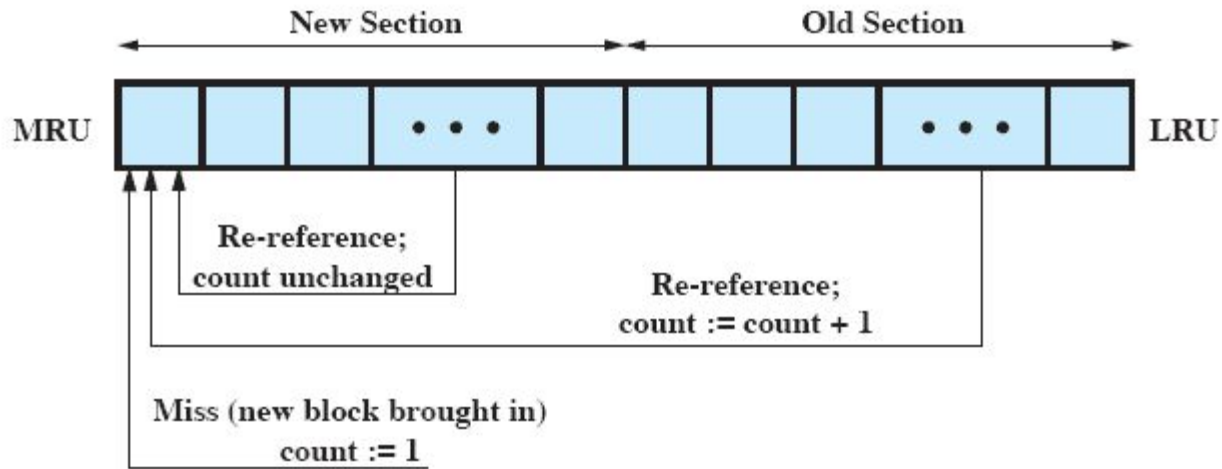


# Least Frequently Used

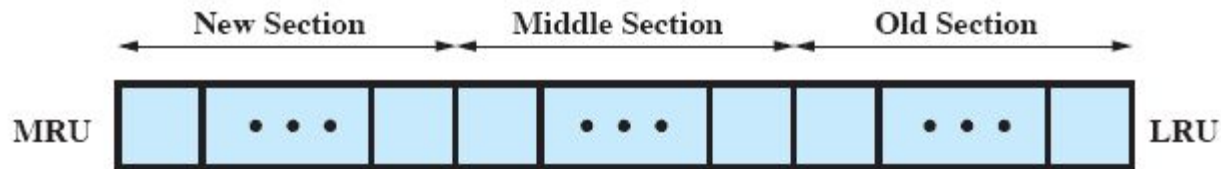
- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block accessed
- When replacement is required, the block with the smallest count is selected.



# Frequency-Based Replacement



(a) FIFO



(b) Use of three sections

# LRU Disk Cache Performance

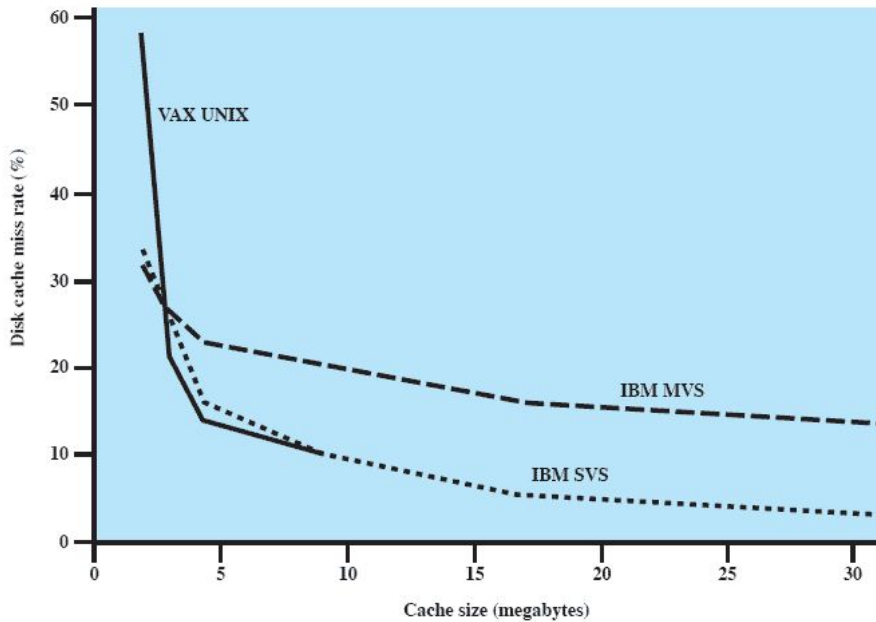


Figure 11.10 Some Disk Cache Performance Results Using LRU

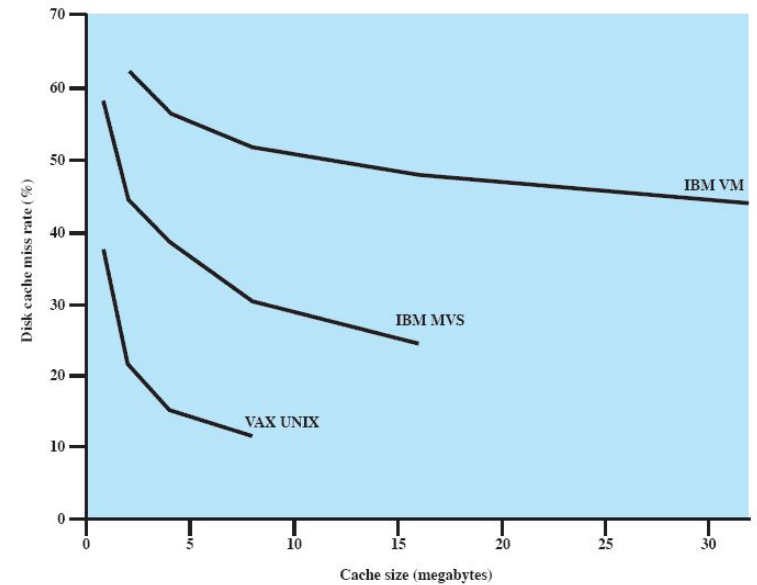


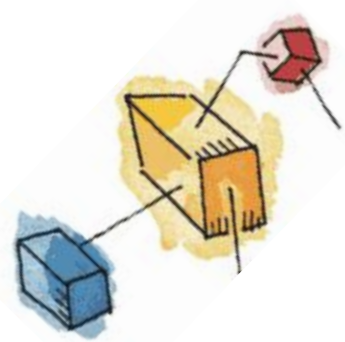
Figure 11.11 Disk Cache Performance Using Frequency-Based Replacement [ROBI90]

# Roadmap

- I/O Devices
- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache
- – UNIX SVR4 I/O
- LINUX I/O
- Windows I/O







# Devices are Files

- Each I/O device is associated with a special file
  - Managed by the file system
  - Provides a clean uniform interface to users and processes.
- To access a device, read and write requests are made for the special file associated with the device.



# UNIX SVR4 I/O

- Each individual device is associated with a special file
- Two types of I/O
  - Buffered
  - Unbuffered

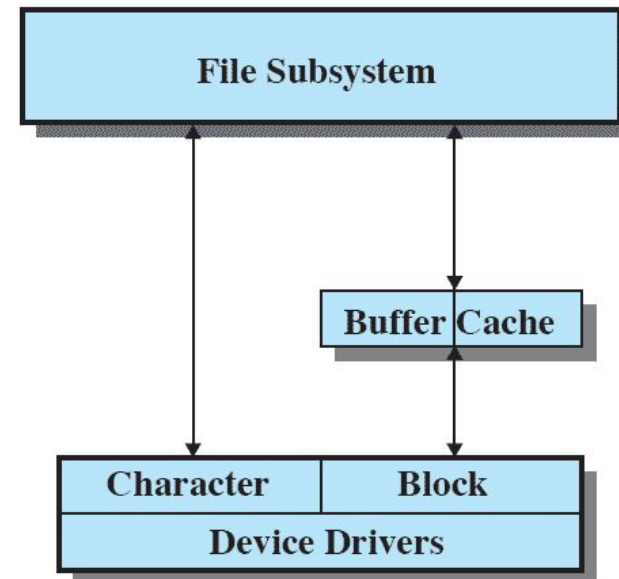
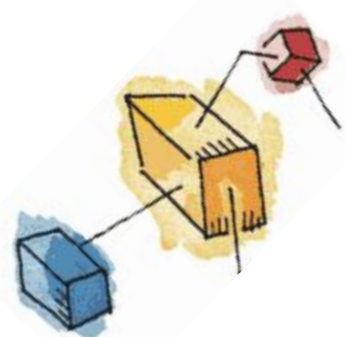


Figure 11.12 UNIX I/O Structure



# Buffer Cache

- Three lists are maintained
  - Free List
  - Device List
  - Driver I/O Queue

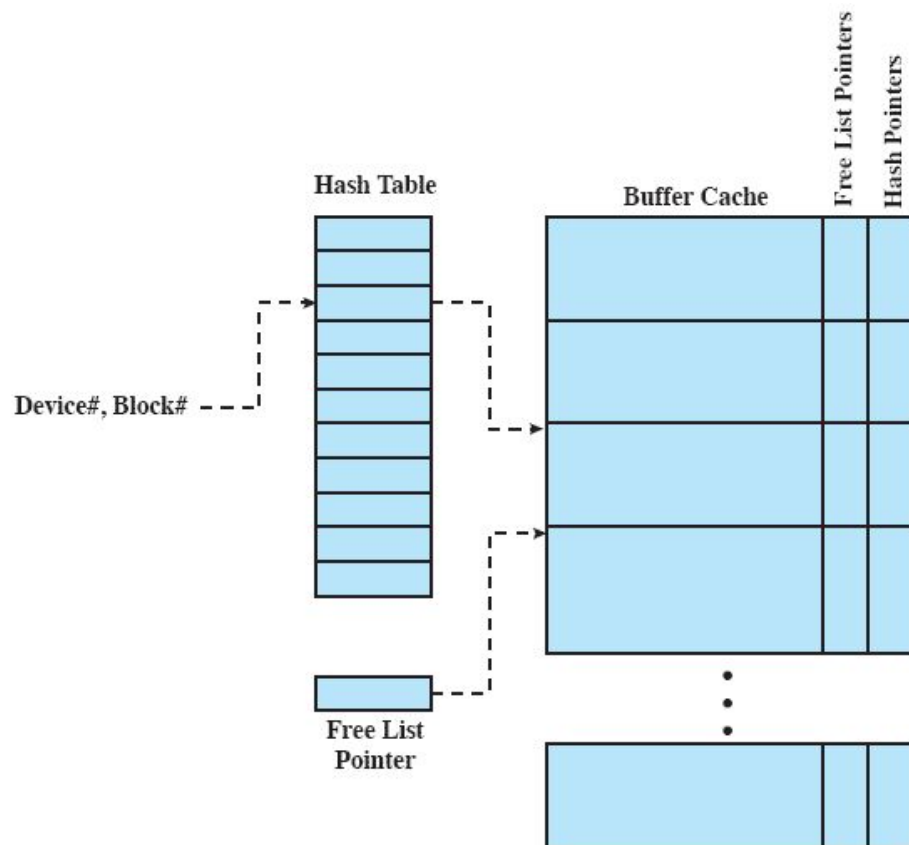


Figure 11.13 UNIX Buffer Cache Organization



# Character Cache

This illustration shows a central yellow box representing a character cache. A blue box on the left is connected to it by a line, representing an I/O device. A red box on the right is also connected to the central box, representing a process. This visualizes the flow of data between a device, a cache, and a process.

- Used by character oriented devices
  - E.g. terminals and printers
- Either written by the I/O device and read by the process or vice versa
  - Producer/consumer model used

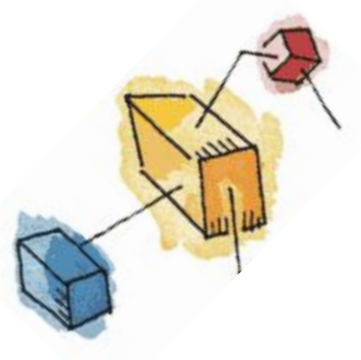




# Unbuffered I/O

- Unbuffered I/O is simply DMA between device and process
  - Fastest method
  - Process is locked in main memory and can't be swapped out
  - Device is tied to process and unavailable for other processes





# I/O for Device Types

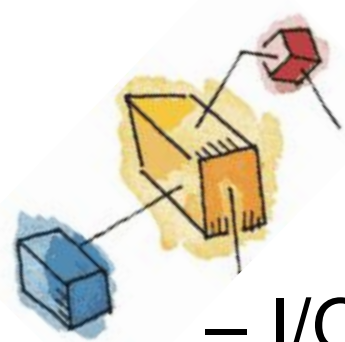
**Table 11.5** Device I/O in UNIX

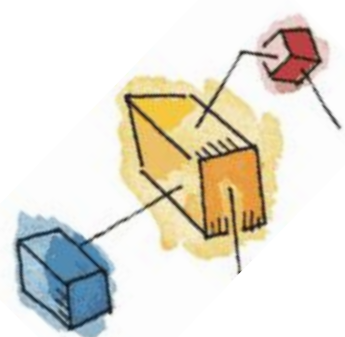
	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X



# Roadmap

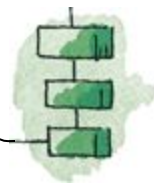
- I/O Devices
- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache
- UNIX SVR4 I/O
- **LINUX I/O**
- Windows I/O



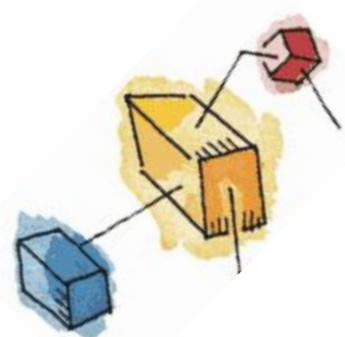


# Linux/Unix Similarities

- Linux and Unix (e.g. SVR4) are very similar in I/O terms
  - The Linux kernel associates a special file with each I/O device driver.
  - Block, character, and network devices are recognized.

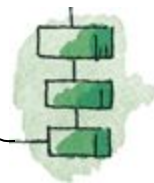






# The Elevator Scheduler

- Maintains a single queue for disk read and write requests
- Keeps list of requests sorted by block number
- Drive moves in a single direction to satisfy each request





# 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

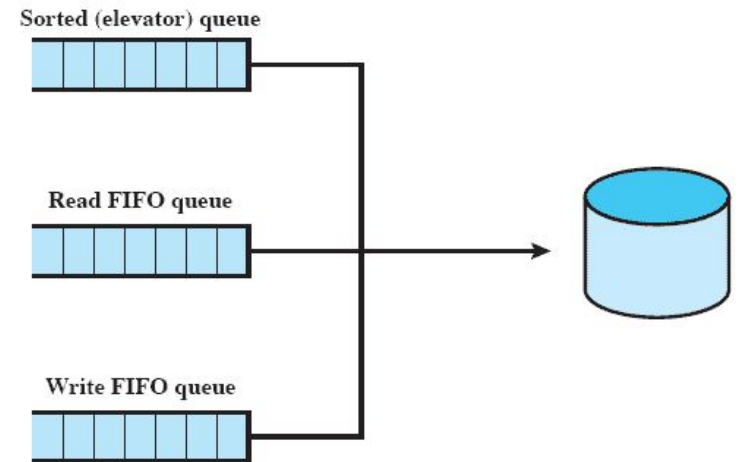
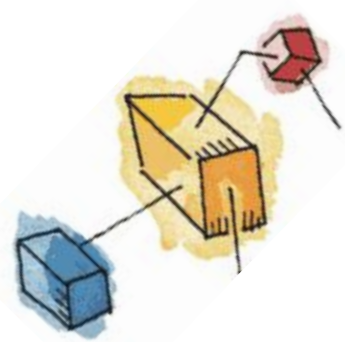


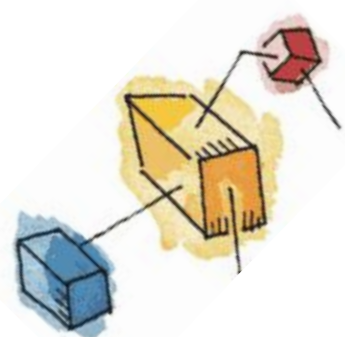
Figure 11.14 The Linux Deadline I/O Scheduler



# Anticipatory I/O scheduler

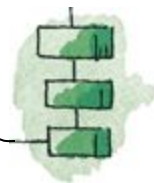
- Elevator and deadline scheduling can be counterproductive if there are numerous synchronous read requests.
- Delay a short period of time after satisfying a read request to see if a new nearby request can be made





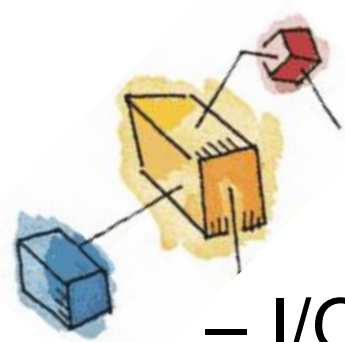
# Linux Page Cache

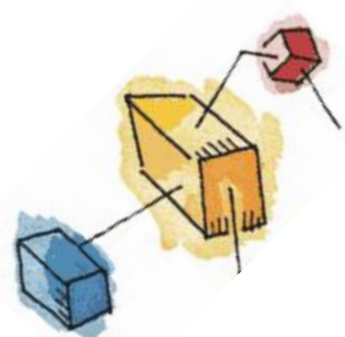
- Linux 2.4 and later, a single unified page cache for all traffic between disk and main memory
- Benefits:
  - Dirty pages can be collected and written out efficiently
  - Pages in the page cache are likely to be referenced again due to temporal locality



# Roadmap

- I/O Devices
- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache
- UNIX SVR4 I/O
- LINUX I/O
- Windows I/O





# Windows I/O Manager

- The I/O manager is responsible for all I/O for the operating system
- It provides a uniform interface that all types of drivers can call.

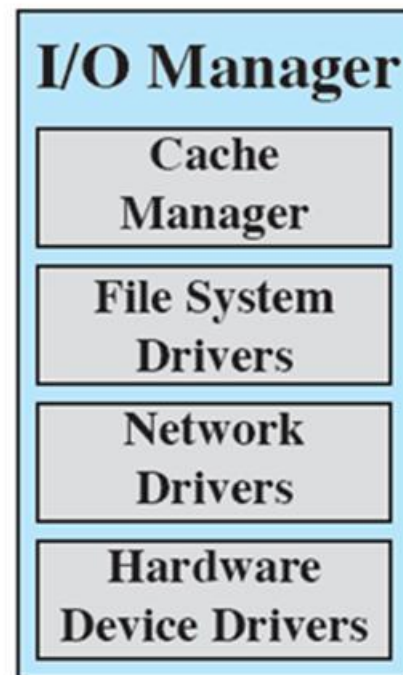
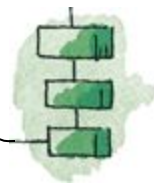


Figure 11.15 Windows I/O Manager





# Windows I/O

- The I/O manager works closely with:
  - Cache manager – handles all file caching
  - File system drivers - routes I/O requests for file system volumes to the appropriate software driver for that volume.
  - Network drivers - implemented as software drivers rather than part of the Windows Executive.
  - Hardware device drivers





# Asynchronous and Synchronous I/O

- Windows offers two modes of I/O operation:
  - asynchronous and synchronous.
- Asynchronous mode is used whenever possible to optimize application performance.







# Software RAID

- Windows implements RAID functionality as part of the operating system and can be used with any set of multiple disks.
- RAID 0, 1 and RAID 5 are supported.
- In the case of RAID 1 (disk mirroring), the two disks containing the primary and mirrored partitions may be on the same disk controller or different disk controllers.





# Volume Shadow Copies

- Shadow copies are implemented by a software driver that makes copies of data on the volume before it is overwritten.
- Designed as an efficient way of making consistent snapshots of volumes so that they can be backed up.
  - Also useful for archiving files on a per-volume basis

