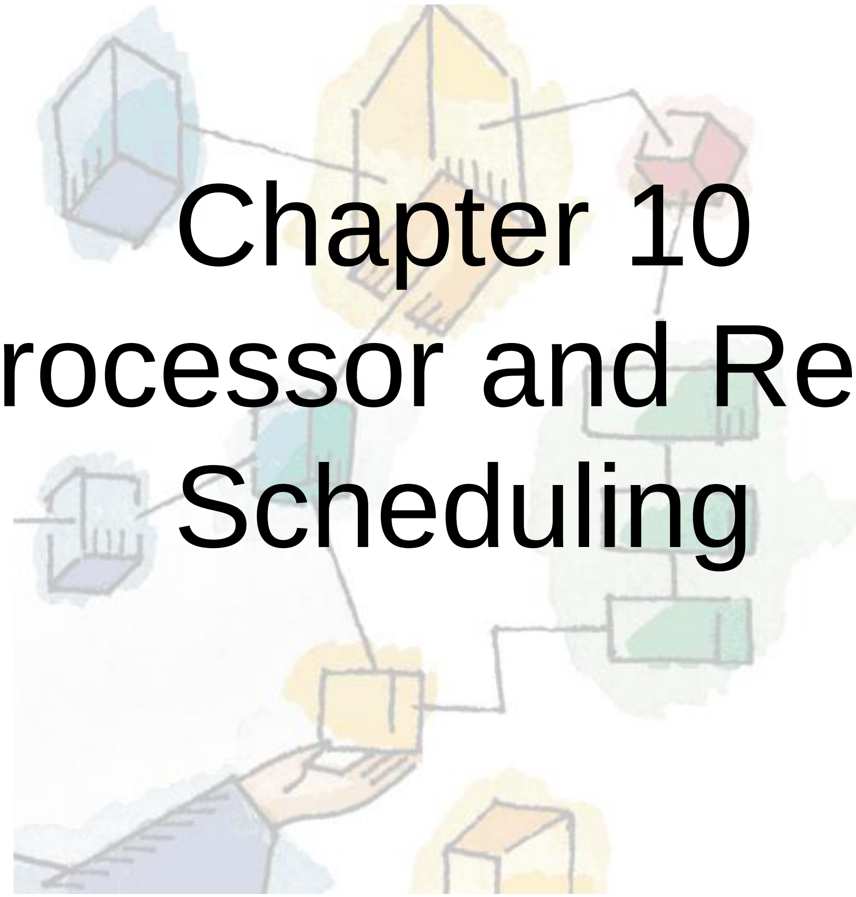
Operating Systems:
Internals and Design Principles, 6/E
William Stallings

Chapter 10
Multiprocessor and Real-Time Scheduling



Dave Bremer
Otago Polytechnic, N.Z.

Classifications of Multiprocessor Systems

- Loosely coupled processors,
  - Each has their memory & I/O channels
- Functionally specialized processors
  - Controlled by a master processor
  - Such as I/O processor
- Tightly coupled multiprocessing
  - Processors share main memory
  - Controlled by operating system

Granularity

- Or frequency of synchronization, between processes in a system.
- Five categories, differing in granularity:
  - Independent Parallelism
  - Coarse Parallelism
  - Very Coarse-Grained Parallelism
  - Medium-Grained Parallelism
  - Fine-Grained Parallelism

Independent Parallelism

- No explicit synchronization among processes
- Separate application or job
- Example is time-sharing system

Coarse and Very Coarse-Grained Parallelism

- Synchronization among processes at a very gross level
- Good for concurrent processes running on a multiprogrammed uniprocessor
  - Can by supported on a multiprocessor with little change

Medium-Grained Parallelism

- Single application is a collection of threads
- Threads usually interact frequently, affecting the performance of the entire application

Fine-Grained Parallelism

- Highly parallel applications
- Specialized and fragmented area

Synchronization Granularity and Processes

Table 10.1 Synchronization Granularity and Processes

Grain Size	Description	Synchronization Interval (Instructions)
Fine	Parallelism inherent in a single instruction stream.	<20
Medium	Parallel processing or multitasking within a single application	20-200
Coarse	Multiprocessing of concurrent processes in a multiprogramming environment	200-2000
Very Coarse	Distributed processing across network nodes to form a single computing environment	2000-1M
Independent	Multiple unrelated processes	not applicable

Valve Example

- Valve (half-life2 etc) found a hybrid approach works best for their games
- Some systems worked best assigned to a single processor. E.G sound mixing
- Others can be threaded so they work on single processors but greatly improve performance spread over multiple processors. E.g. scene rendering

Thread Structure for Rendering Module

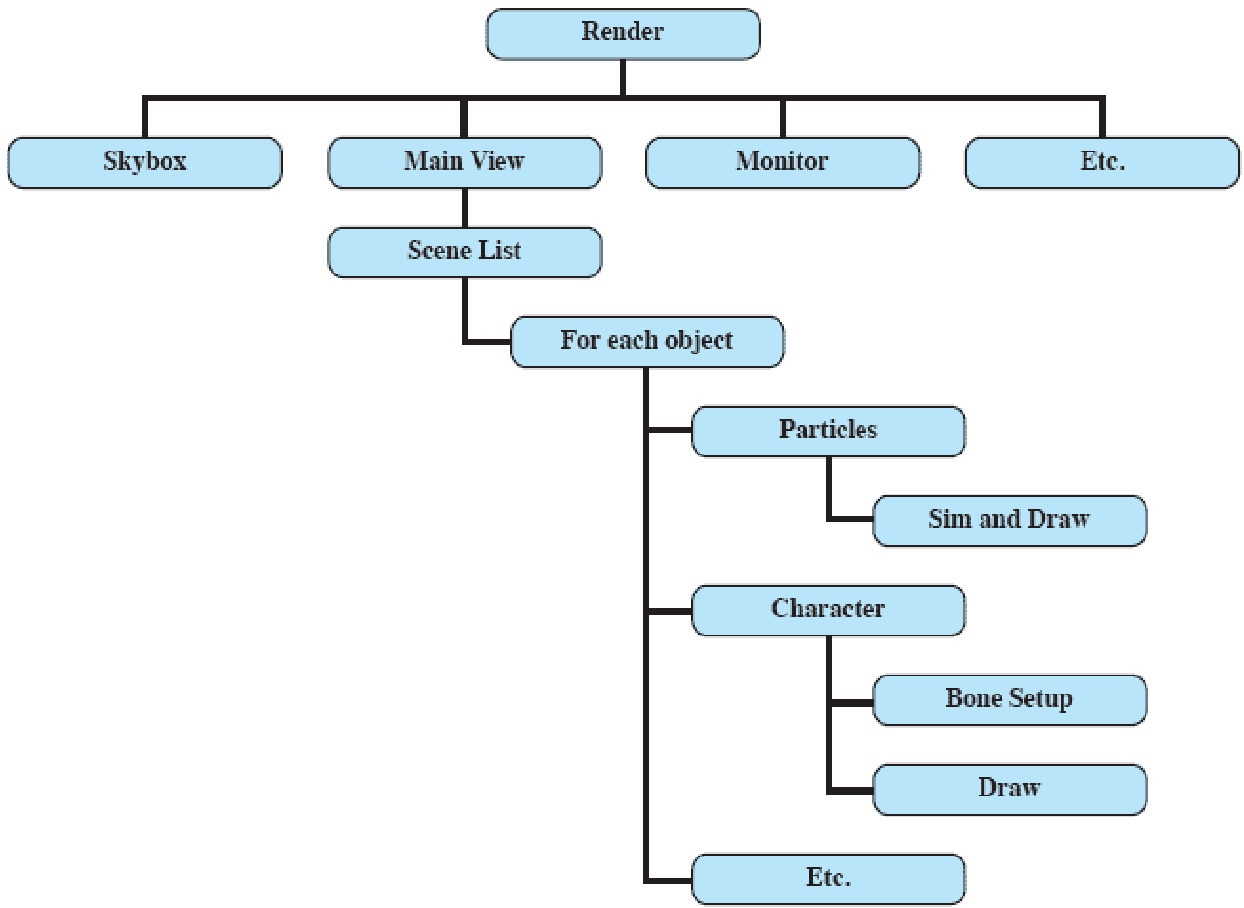


Figure 10.1 Hybrid Threading for Rendering Module

Scheduling Design Issues

- Scheduling on a multiprocessor involves three interrelated issues:
  - Assignment of processes to processors
  - Use of multiprogramming on individual processors
  - Actual dispatching of a process
- The approach taken will depend on the degree of granularity of applications and the number of processors available

Assignment of Processes to Processors

- Assuming all processors are equal, it is simplest to treat processors as a pooled resource and assign process to processors on demand.
  - Should the assignment be static or dynamic though?
- Dynamic Assignment
  - threads are moved for a queue for one processor to a queue for another processor;

Static Assignment

- Permanently assign process to a processor
  - Dedicate short-term queue for each processor
  - Less overhead
  - Allows the use of ‘group’ or ‘gang’ scheduling (see later)
- But may leave a processor idle, while others have a backlog
  - Solution: use a common queue

Assignment of Processes to Processors

- Both dynamic and static methods require some way of assigning a process to a processor
- Two methods:
  - Master/Slave
  - Peer
- There are of course a spectrum of approaches between these two extremes.

Master / Slave Architecture

- Key kernel functions always run on a particular processor
- Master is responsible for scheduling
- Slave sends service request to the master
- Disadvantages
  - Failure of master brings down whole system
  - Master can become a performance bottleneck

Peer architecture

- Kernel can execute on any processor
- Each processor does self-scheduling
- Complicates the operating system
  - Make sure two processors do not choose the same process

Process Scheduling

- Usually processes are not dedicated to processors
- A single queue is used for all processes
- Or multiple queues are used for priorities
  - All queues feed to the common pool of processors

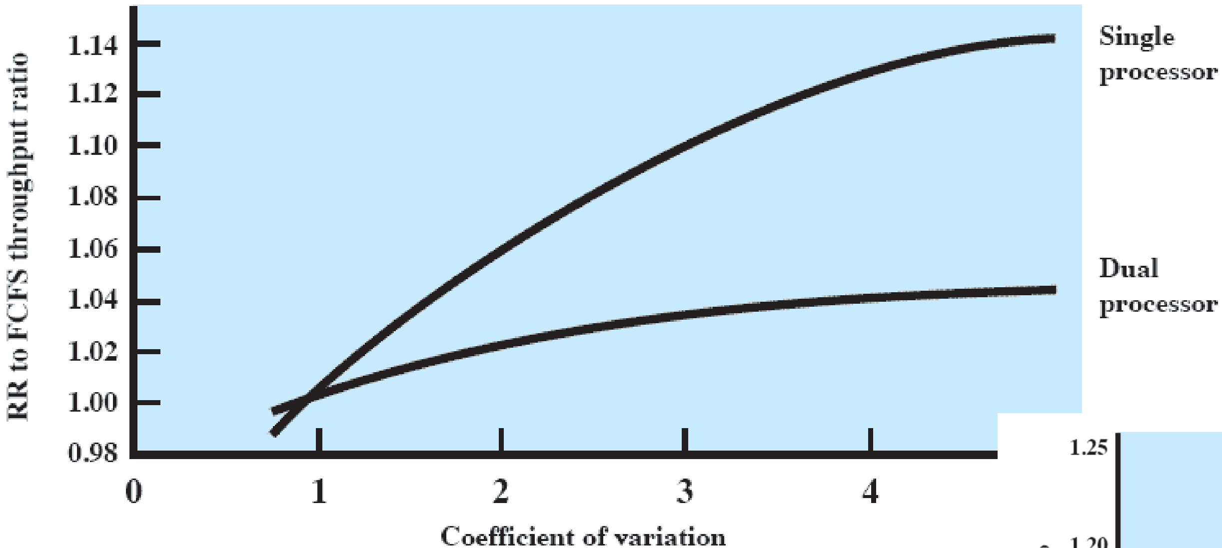
Thread Scheduling

- Threads execute separate from the rest of the process
- An application can be a set of threads that cooperate and execute concurrently in the same address space
- Dramatic gains in performance are possible in multi-processor systems
  - Compared to running in uniprocessor systems

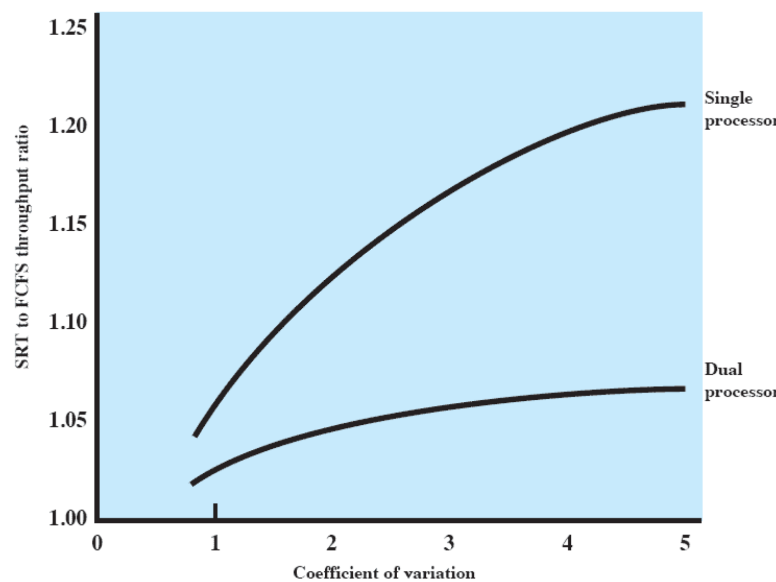
Approaches to Thread Scheduling

- Many proposals exist but four general approaches stand out:
  - Load Sharing
  - Gang Scheduling
  - Dedicated processor assignment
  - Dynamic scheduling

Comparison One and Two Processors



(a) Comparison of RR and FCFS



(b) Comparison of SRT and FCFS

Figure 10.2

Load Sharing

- Processes are not assigned to a particular processor
- Load is distributed evenly across the processors
- No centralized scheduler required
- The global queue can be organized and accessed using any of the schemes discussed in Chapter 9.

Disadvantages of Load Sharing

- Central queue needs mutual exclusion
  - Can lead to bottlenecks
- Preemptive threads are unlikely resume execution on the same processor
- If all threads are in the global queue, all threads of a program will not gain access to the processors at the same time

Gang Scheduling

- A set of related threads is scheduled to run on a set of processors at the same time
- Parallel execution of closely related processes may reduce overhead such as process switching and synchronization blocking.

Example
Scheduling Groups

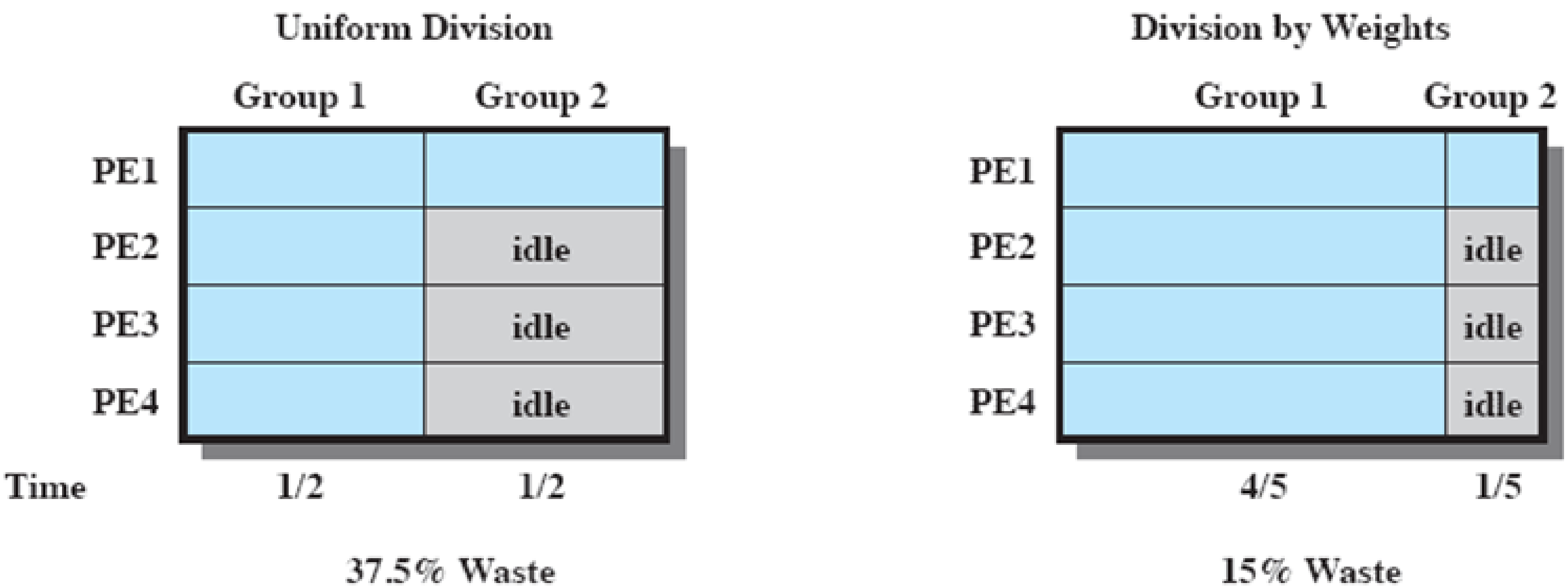


Figure 10.3 Example of Scheduling Groups with Four and One Threads [FEIT90b]

Dedicated Processor Assignment

- When application is scheduled, its threads are assigned to a processor
- Some processors may be idle
  - No multiprogramming of processors
- ***But***
  - In *highly* parallel systems processor utilization is less important than effectiveness
  - Avoiding process switching speeds up programs

Application Speedup

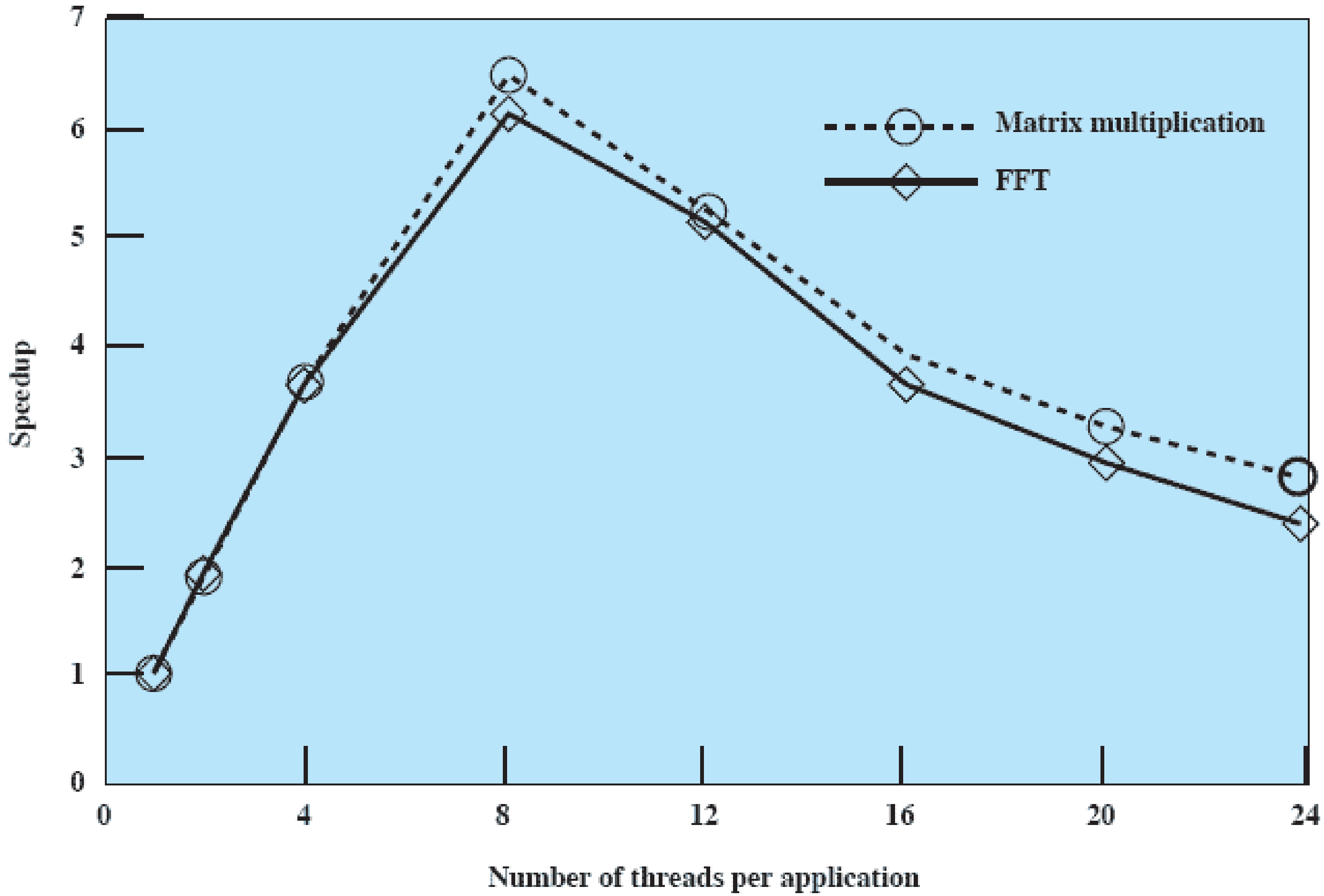
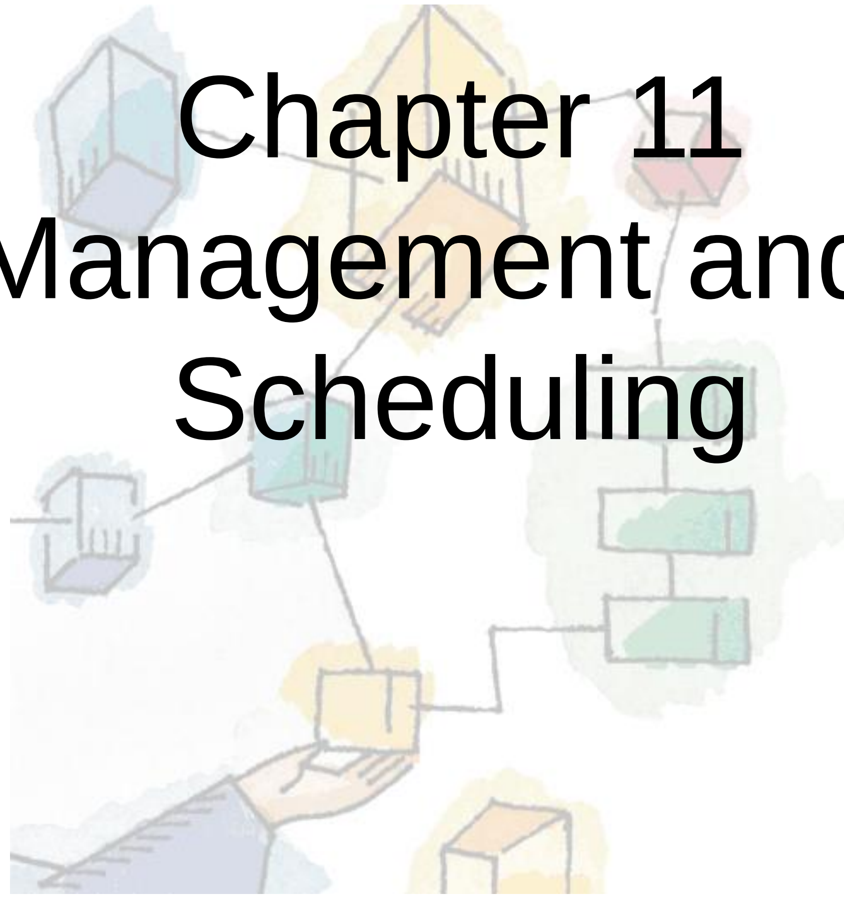


Figure 10.4 Application Speedup as a Function of Number of Threads



Chapter 11
I/O Management and Disk Scheduling

Dave Bremer
Otago Polytechnic, NZ

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

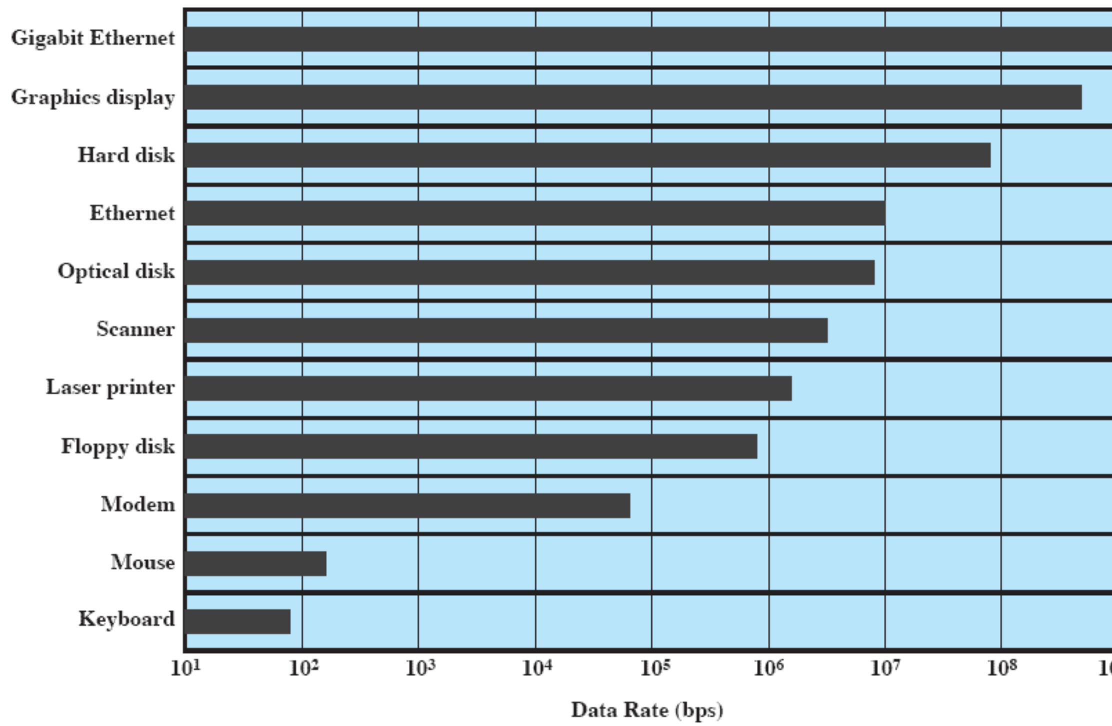


Figure 11.1 Typical I/O Device Data Rates

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**
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. 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

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

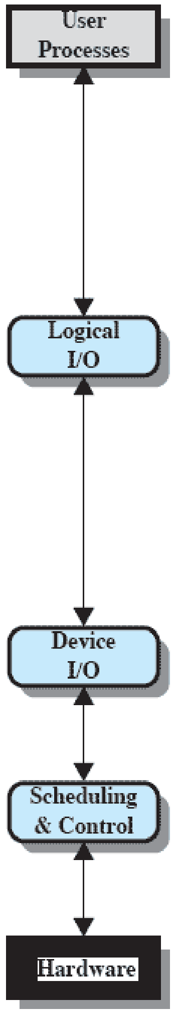
Generality

- 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

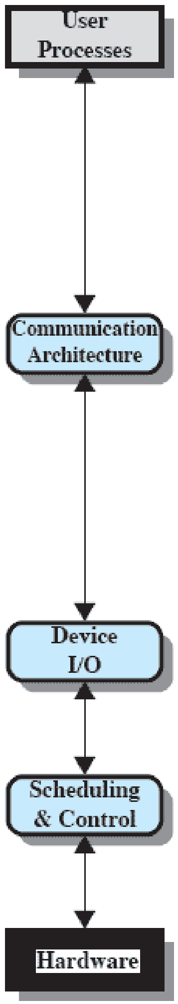
Local peripheral device



(a) 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

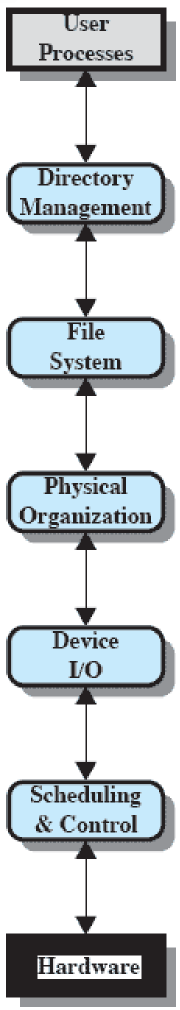
Communications Port



(b) 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,

File System



(c) File system

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

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.

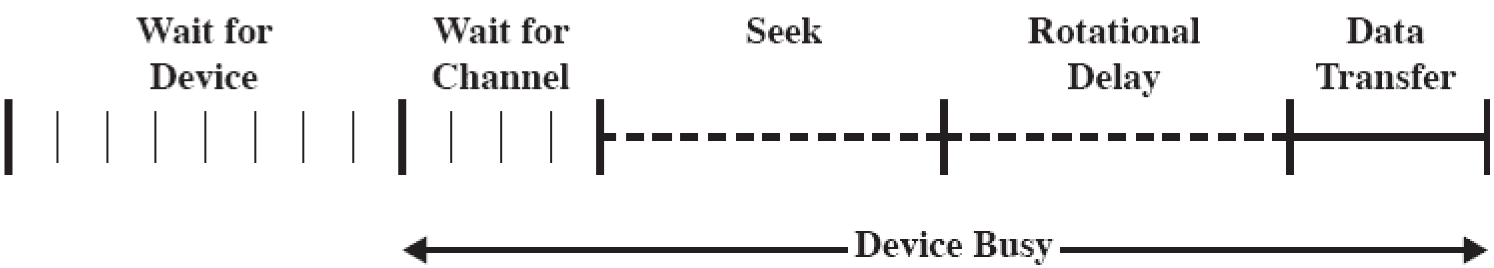


Figure 11.6 Timing of a Disk I/O Transfer

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

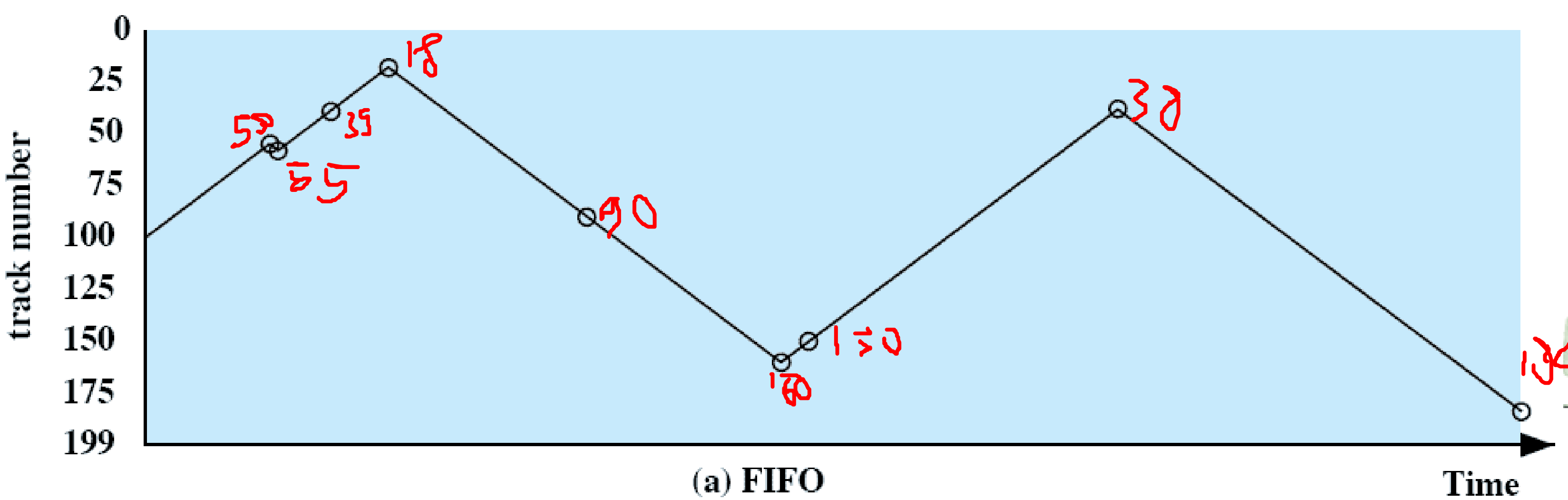
- ***Access Time*** is the sum of:
  - ***Seek time:*** The time it takes to position the head at the desired track
  - ***Rotational delay*** or ***rotational latency:*** The time its takes for the beginning of the sector to reach the head
- ***Transfer Time*** is the time taken to transfer the data.

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



(a) FIFO

Priority

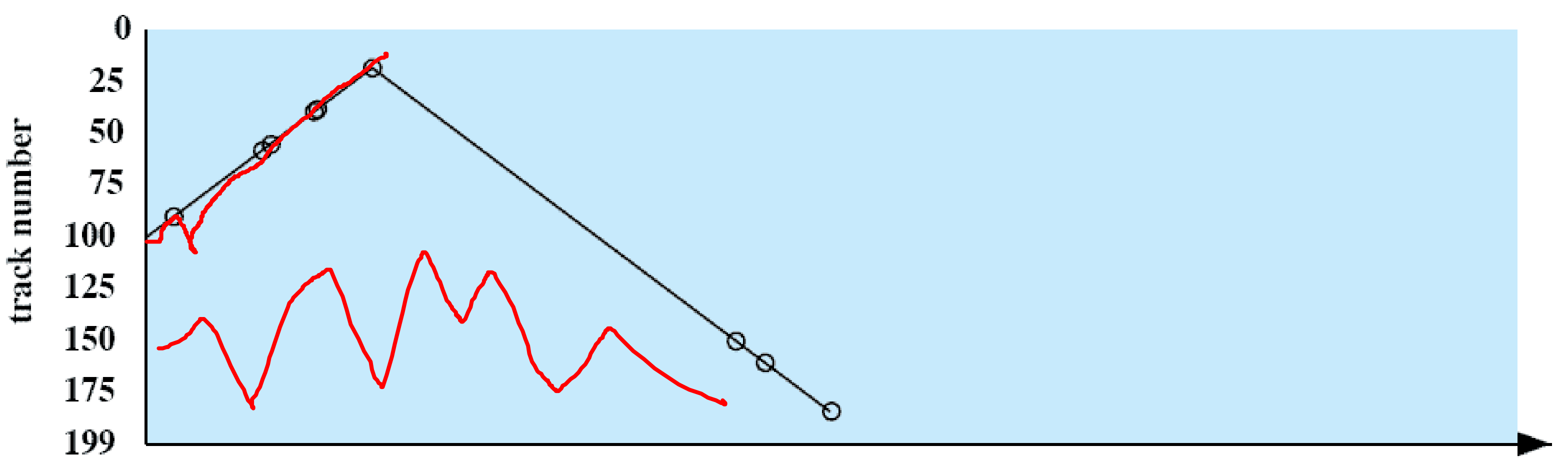
- Goal is not to optimize disk use but to meet other objectives
- Short batch jobs may have higher priority
- Provide good interactive response time
- Longer jobs may have to wait an excessively long time
- A poor policy for database systems

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

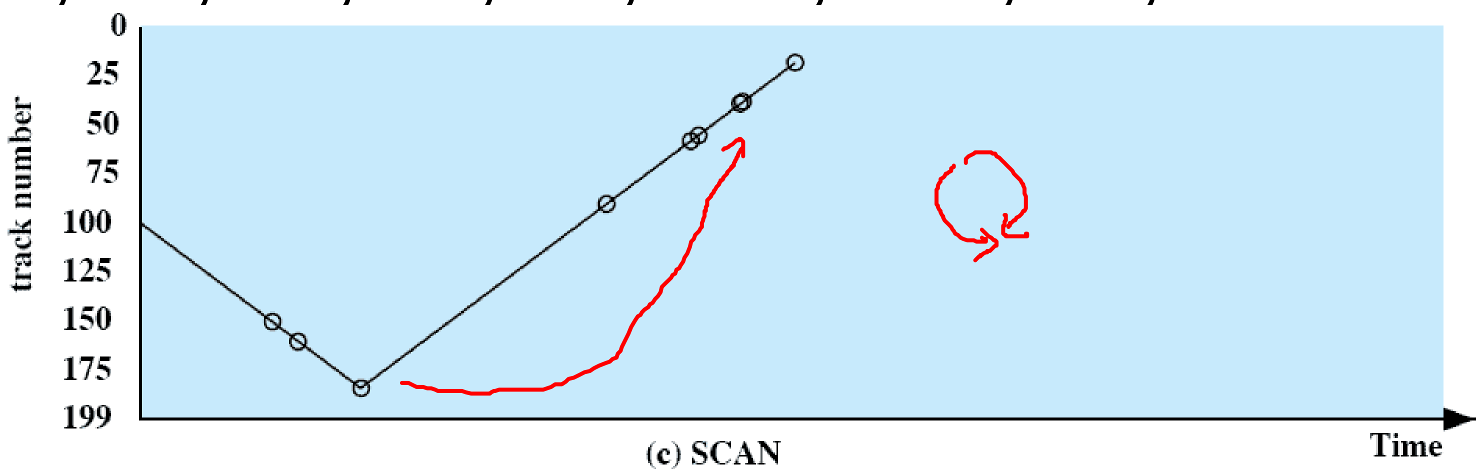
- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time
- 55, 58, 39, 18, 90, 160, 150, 38, 184, 112



(b) SSTF

SCAN

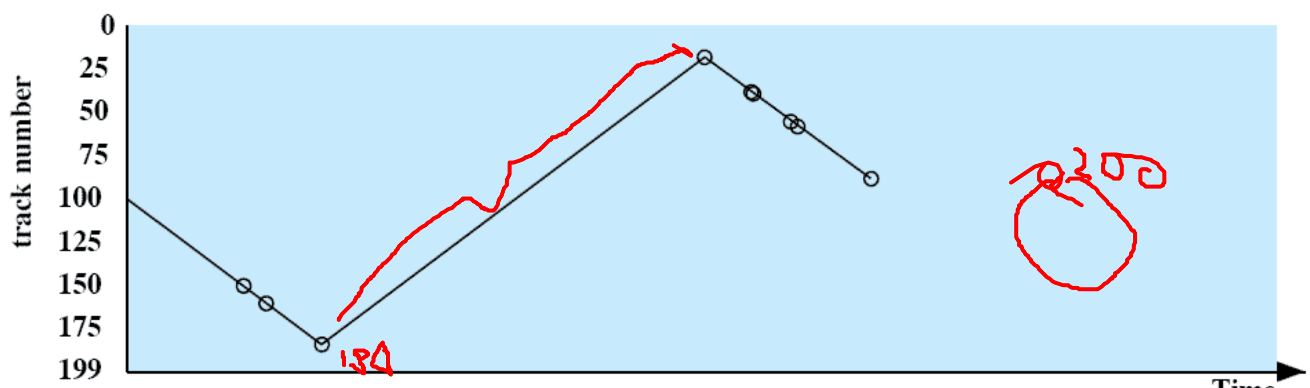
- Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction then the direction is reversed
- 55, 58, 39, 18, 90, 160, 150, 38, 184



(c) SCAN

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
- 55, 58, 39, 18, 90, 160, 150, 38, 184



(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
- 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
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Disk Scheduling Algorithms

Table 11.3 Disk Scheduling Algorithms

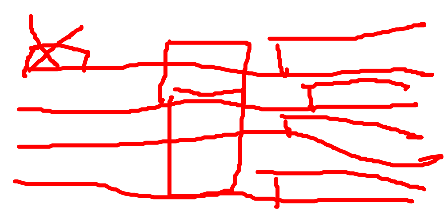
Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according 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

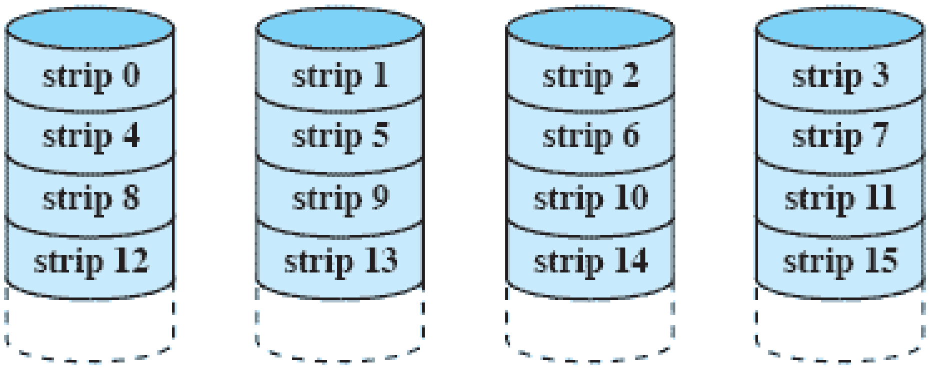
- Disk I/O performance may be increased by spreading the operation over multiple read/write heads
  - Or multiple disks
- Disk failures can be recovered if parity information is stored



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

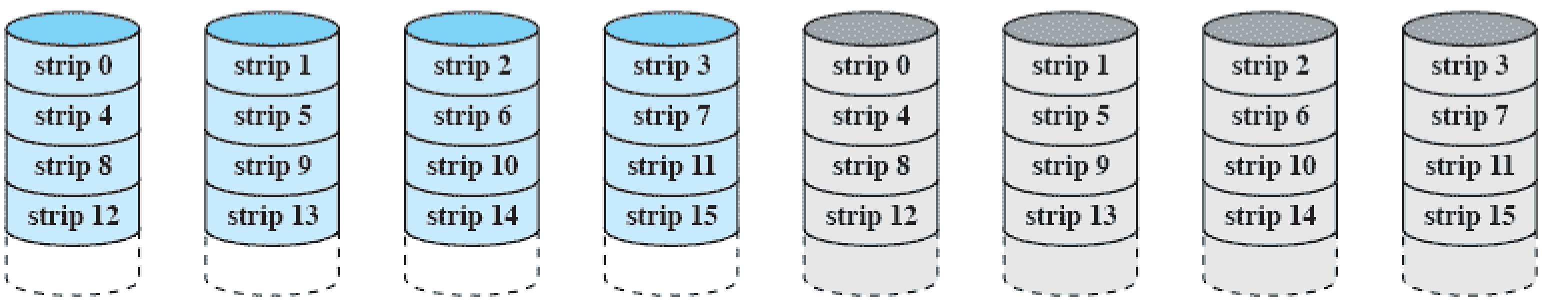


(a) RAID 0 (non-redundant)

- 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 made in parallel.
- Simple recovery from disk failure

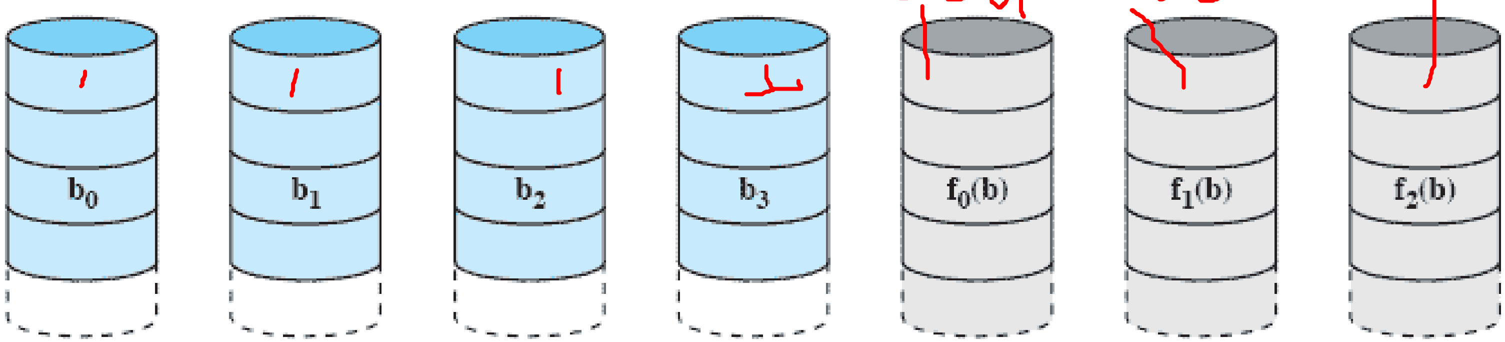


(b) RAID 1 (mirrored)

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

$b_0' \oplus b_1'$ $b_1' \oplus b_2'$ $b_2' \oplus b_3'$



(c) RAID 2 (redundancy through Hamming code)

$A \oplus C = B$

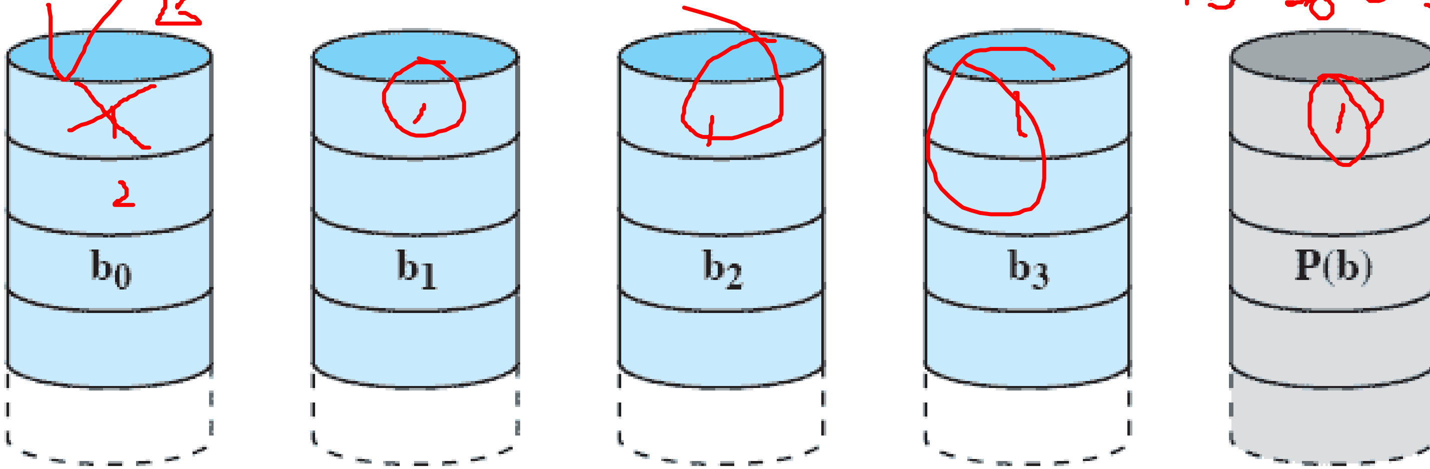
$A \oplus B = C$ $B \oplus C = A$

RAID 3
bit-interleaved parity

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

$b_0' = P(b)_1 \oplus b_1 \oplus b_2 \oplus b_3$

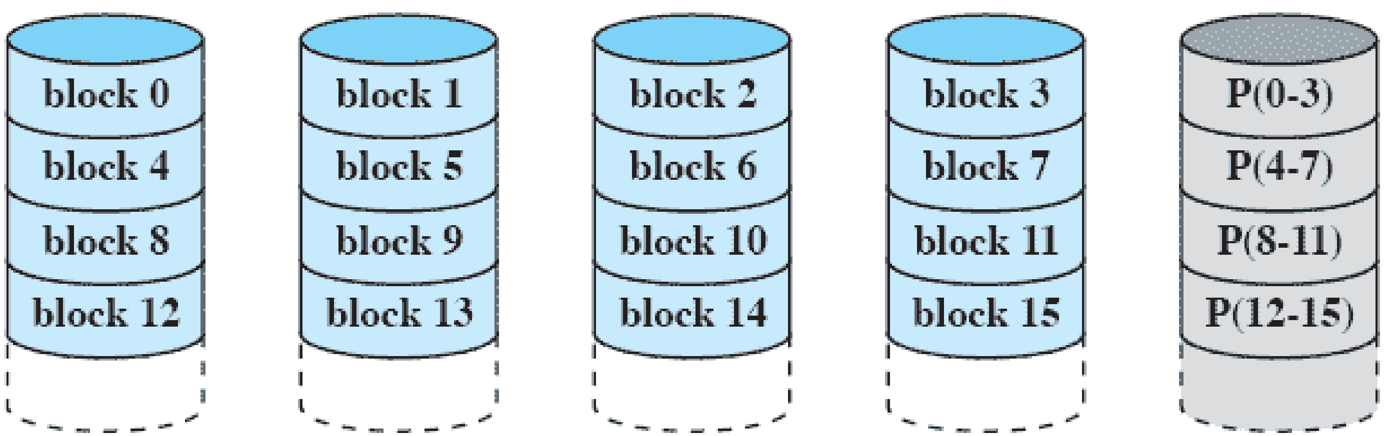
$Pb' = b_0' \oplus b_1 \oplus b_2 \oplus b_3$



(d) RAID 3 (bit-interleaved parity)

RAID 4
Block-level parity

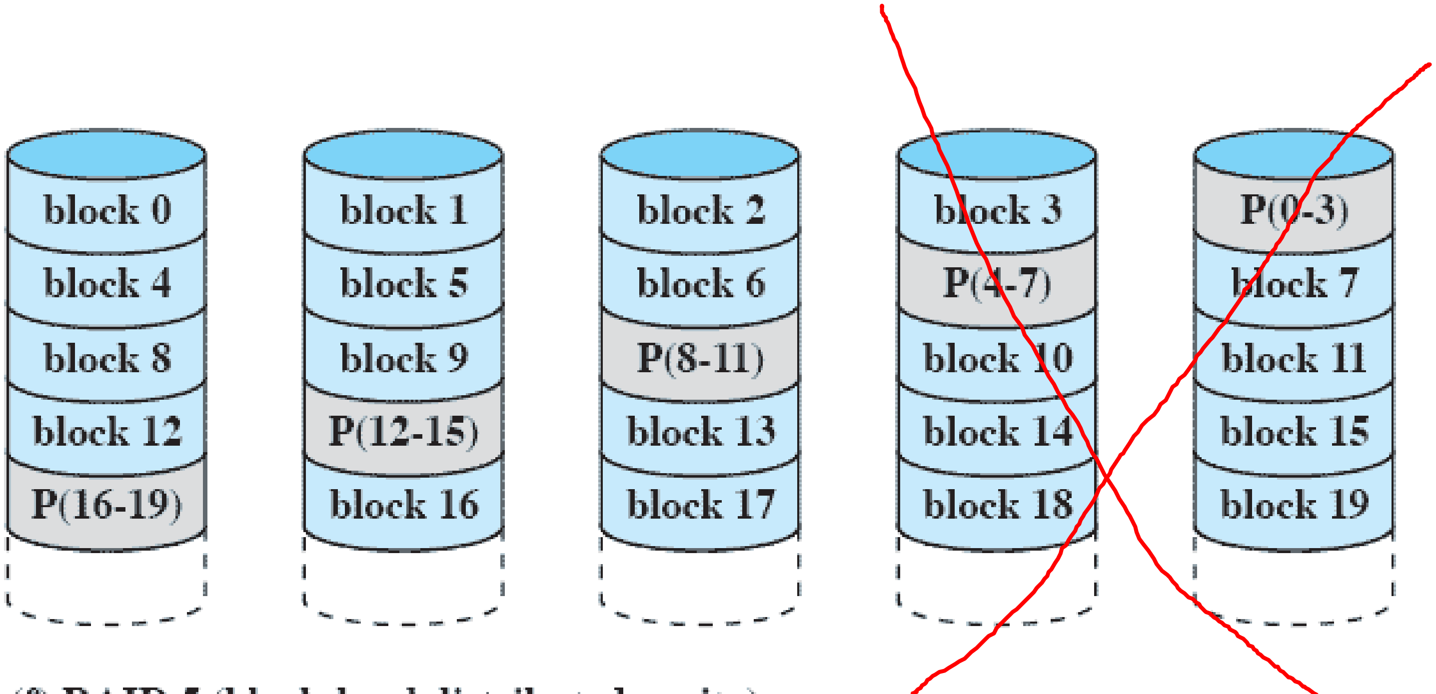
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk
- The parity bits are stored in the corresponding strip on the parity disk.



(e) RAID 4 (block-level parity)

RAID 5
Block-level Distributed parity

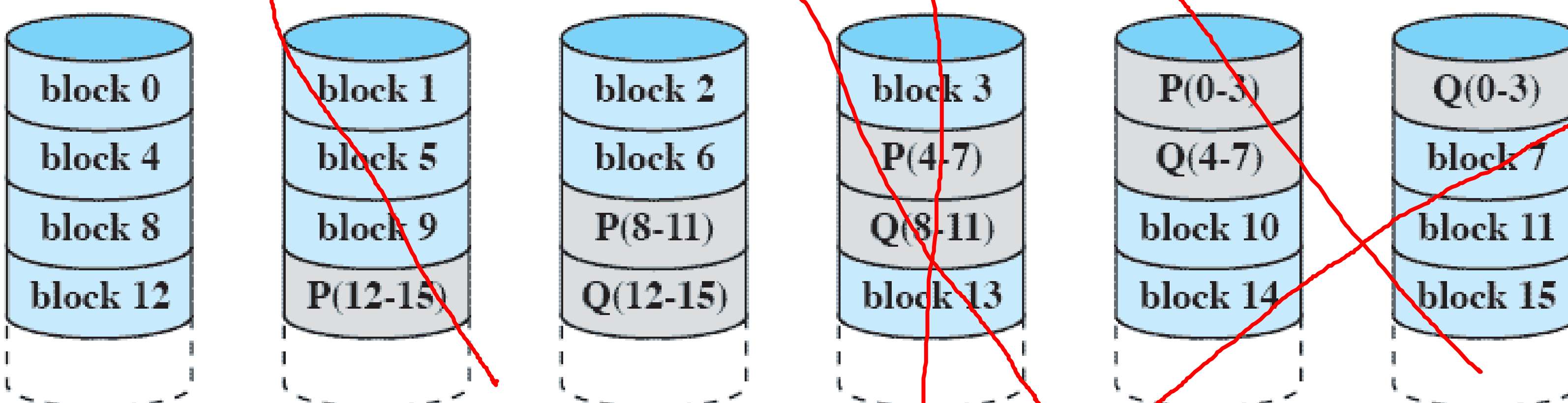
- Similar to RAID-4 but distributing the parity bits across all drives



(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)