Fontys
UNIVERSITY OF
APPLIED SCIENCES

› FOR SOCIETY

Lesson 11 – Chapter 11
I/O Management and Disk Scheduling

Fontys
› FOR SOCIETY

Roadmap

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

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

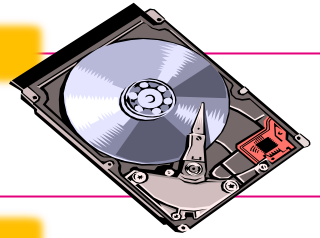
Fontys

Categories of I/O Devices

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

Human readable

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



Machine readable

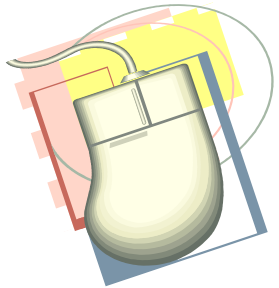
- suitable for communicating with electronic equipment
- disk drives, USB keys, sensors, controllers

Communication

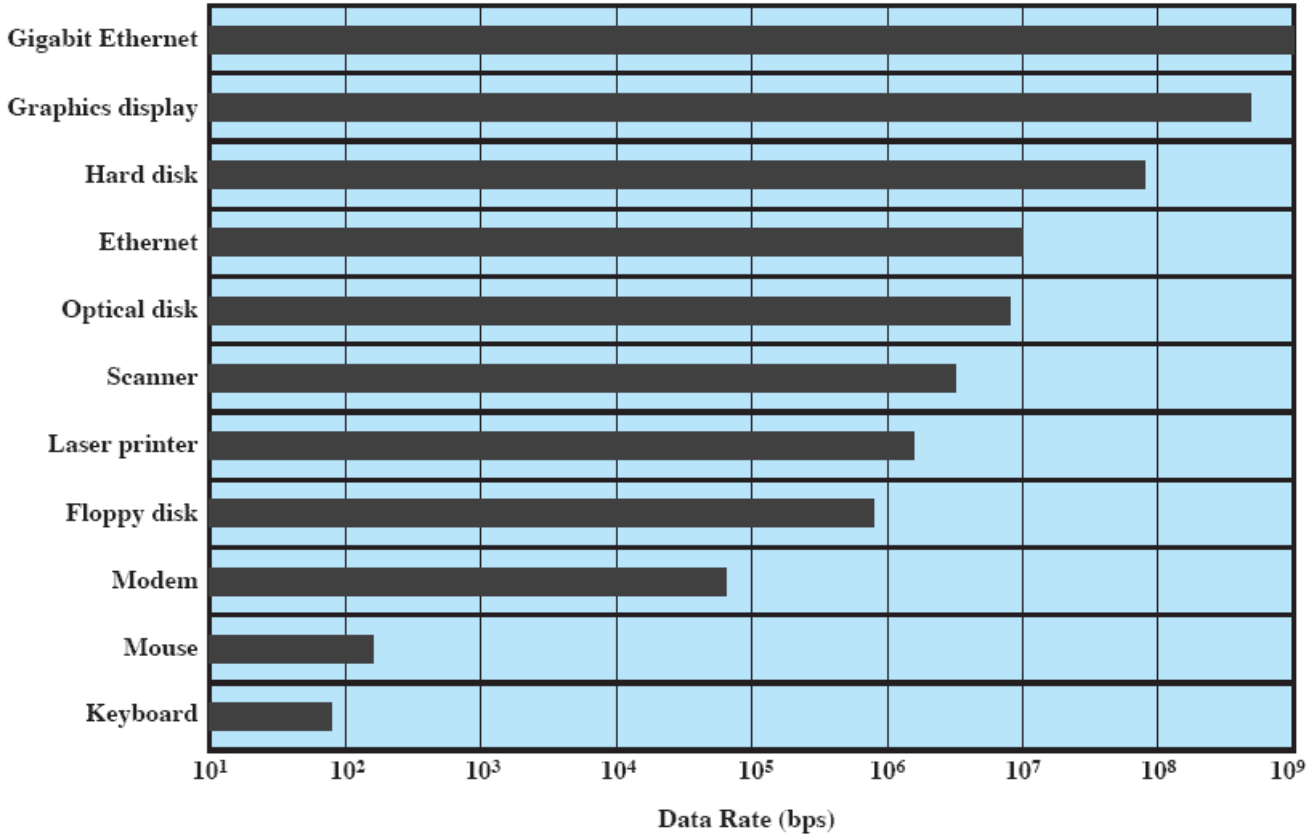
- suitable for communicating with remote devices
- modems, digital line drivers

Differences in I/O Devices

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



Data Rate - Differences



Fontys

Fontys
› FOR SOCIETY

Roadmap

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

Techniques for performing I/O

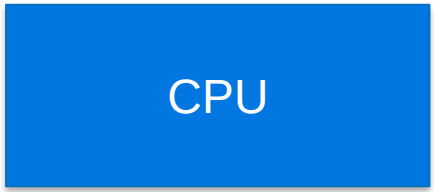
- Programmed I/O
- Interrupt-driven I/O
- Direct memory access (DMA)

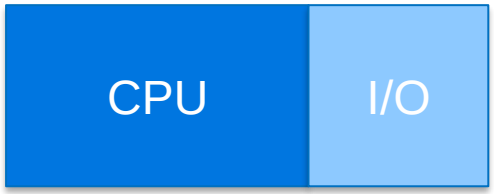
Table 11.1 I/O Techniques

	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

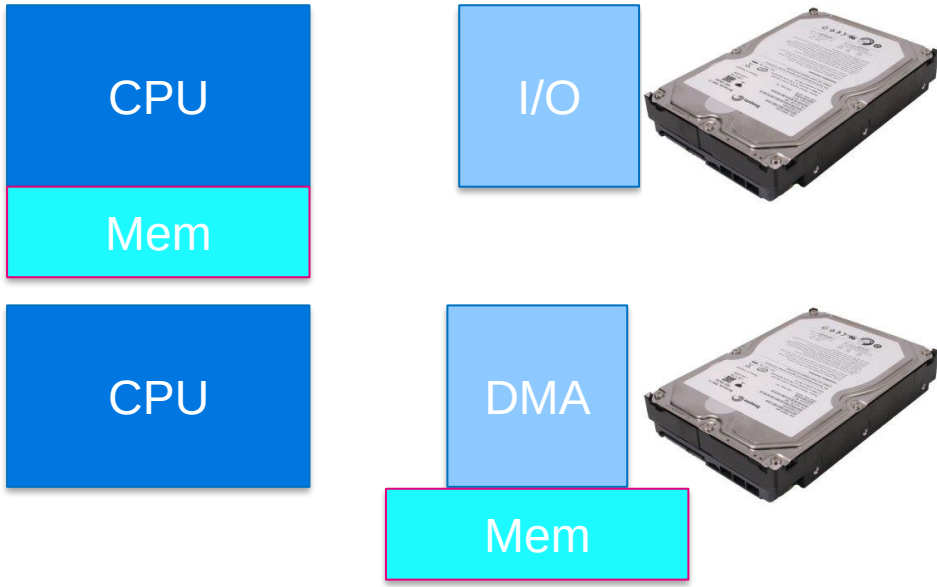
1. Processor directly controls a peripheral device
2. Controller or I/O module is added
   - Processor uses programmed I/O without interrupts
   - Processor does not need to handle details of external devices





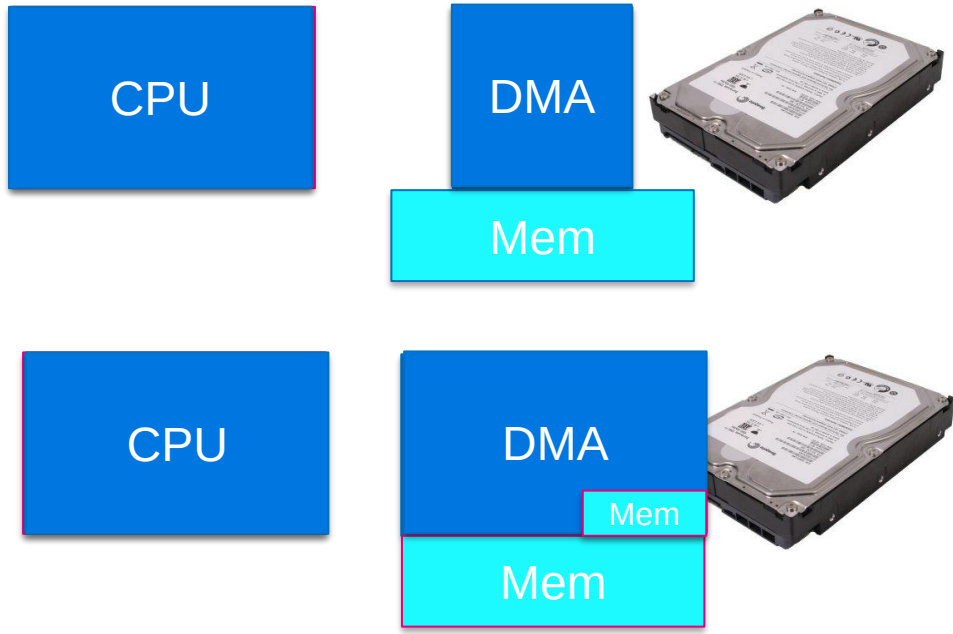
Evolution of the I/O Function cont…

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



Direct Memory Access

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

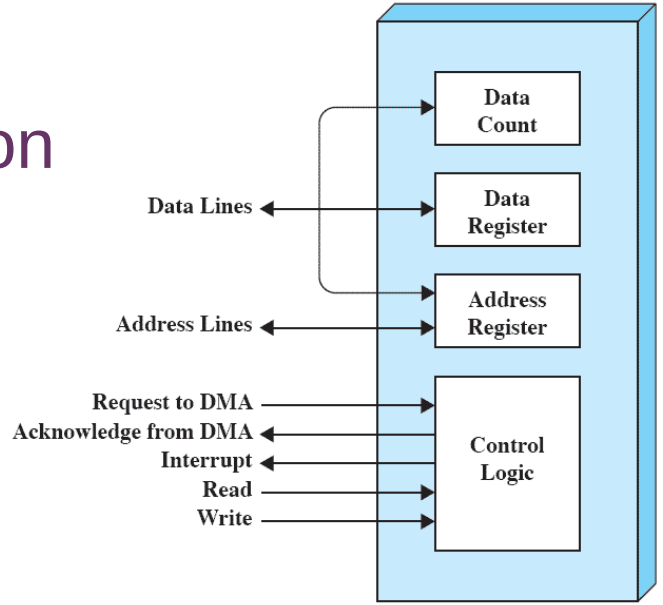
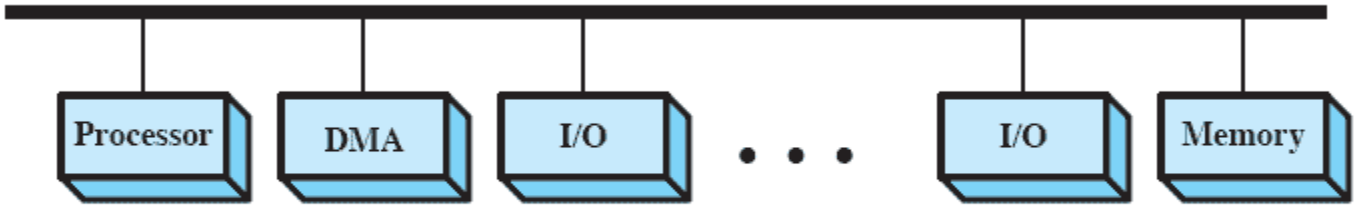


Figure 11.2 Typical DMA Block Diagram

DMA Configurations: Single Bus

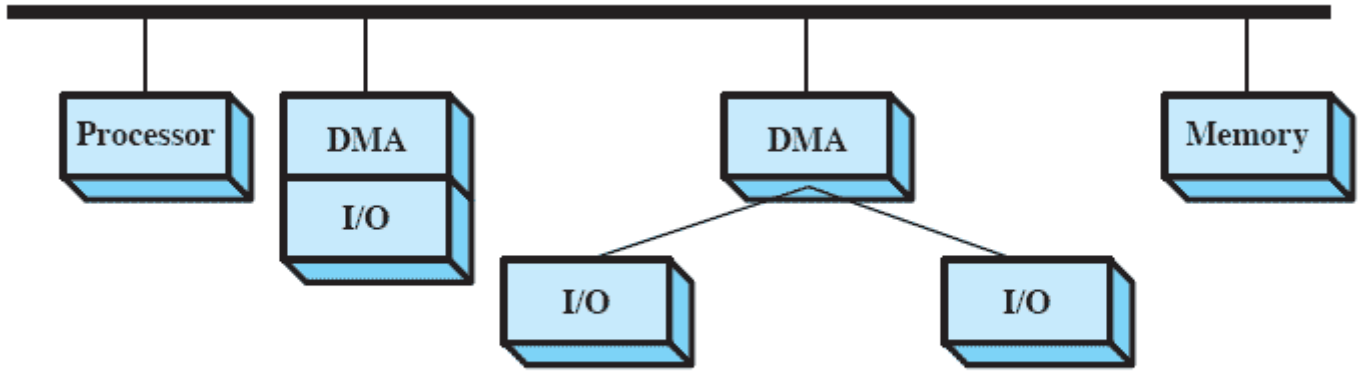


(a) Single-bus, detached DMA

DMA can be configured in several ways

- Shown here, all modules share the same system bus

DMA Configurations: Integrated DMA & I/O

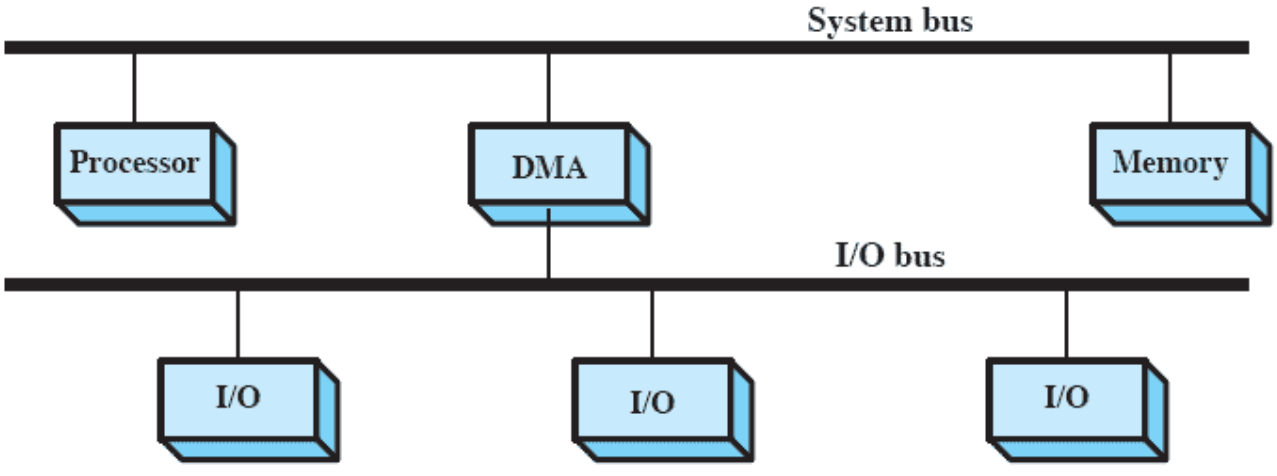


(b) Single-bus, Integrated DMA-I/O

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

Fontys
› FOR SOCIETY

Roadmap

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

Design - 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

Design - Goals: Generality

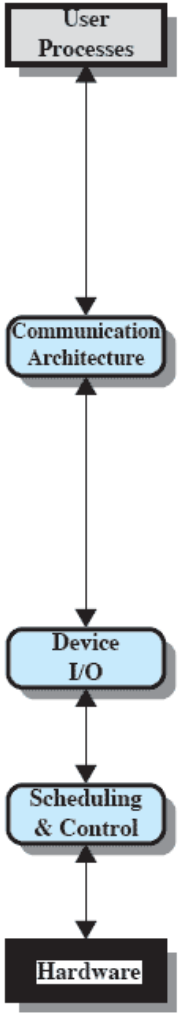
- For simplicity and freedom from error it is desirable to handle all I/O devices in a uniform manner
- Hide most of the details of device I/O in lower-level routines
- Difficult to completely generalize, but can use a hierarchical modular design of I/O functions

Hierarchical design

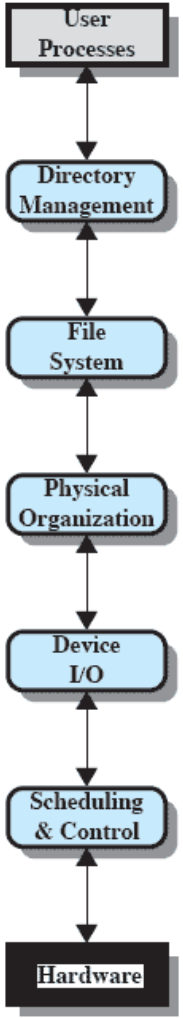
- A hierarchical philosophy leads to organizing an OS into layers
- Each layer relies on the next lower layer to perform more primitive functions
- It provides services to the next higher layer.
- Changes in one layer should not require changes in other layers

peripheral

communications



files



Roadmap

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

I/O Buffering - *Why?*

- Processes must wait for I/O to complete before proceeding
- It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers sometime after the request is made.

Buffering

Block-oriented Buffering

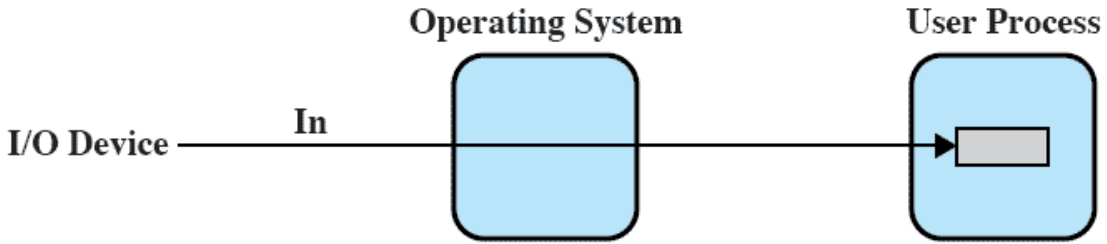
- Information is stored in fixed sized blocks
- Transfers are made a block at a time
- Used for disks and USB keys

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

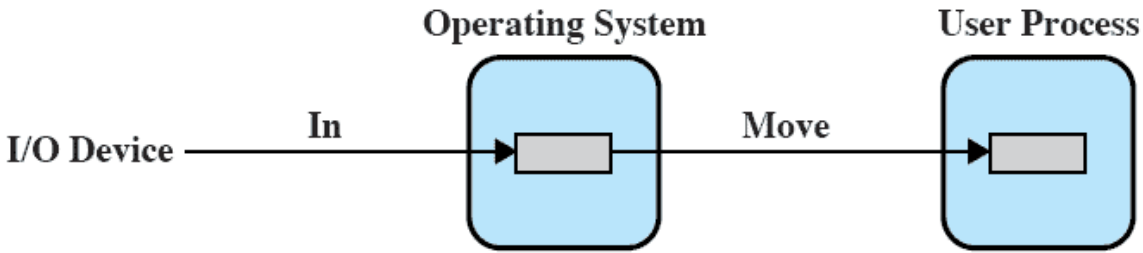
- Without a buffer, the OS directly access the device as and when it needs



(a) No buffering

Single Buffer

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

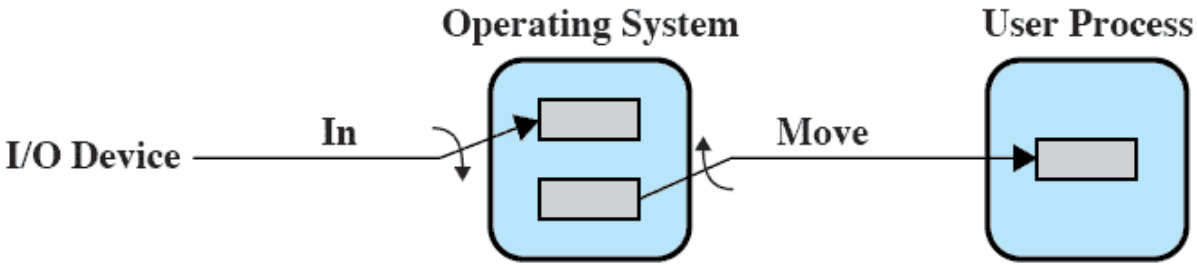


(b) Single buffering

Fontys

Double Buffer

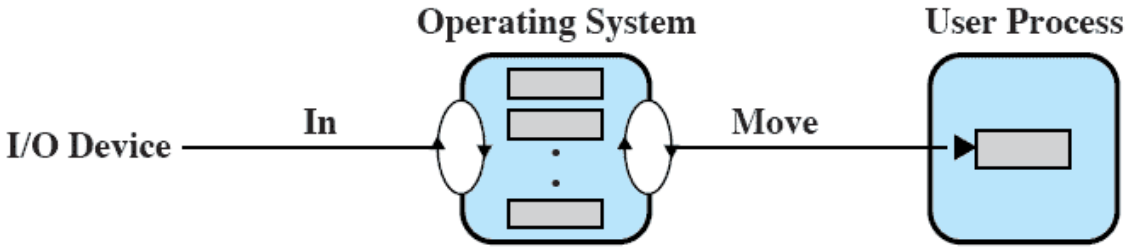
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



(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.
- 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.
- But…

  Size and numbers are limited

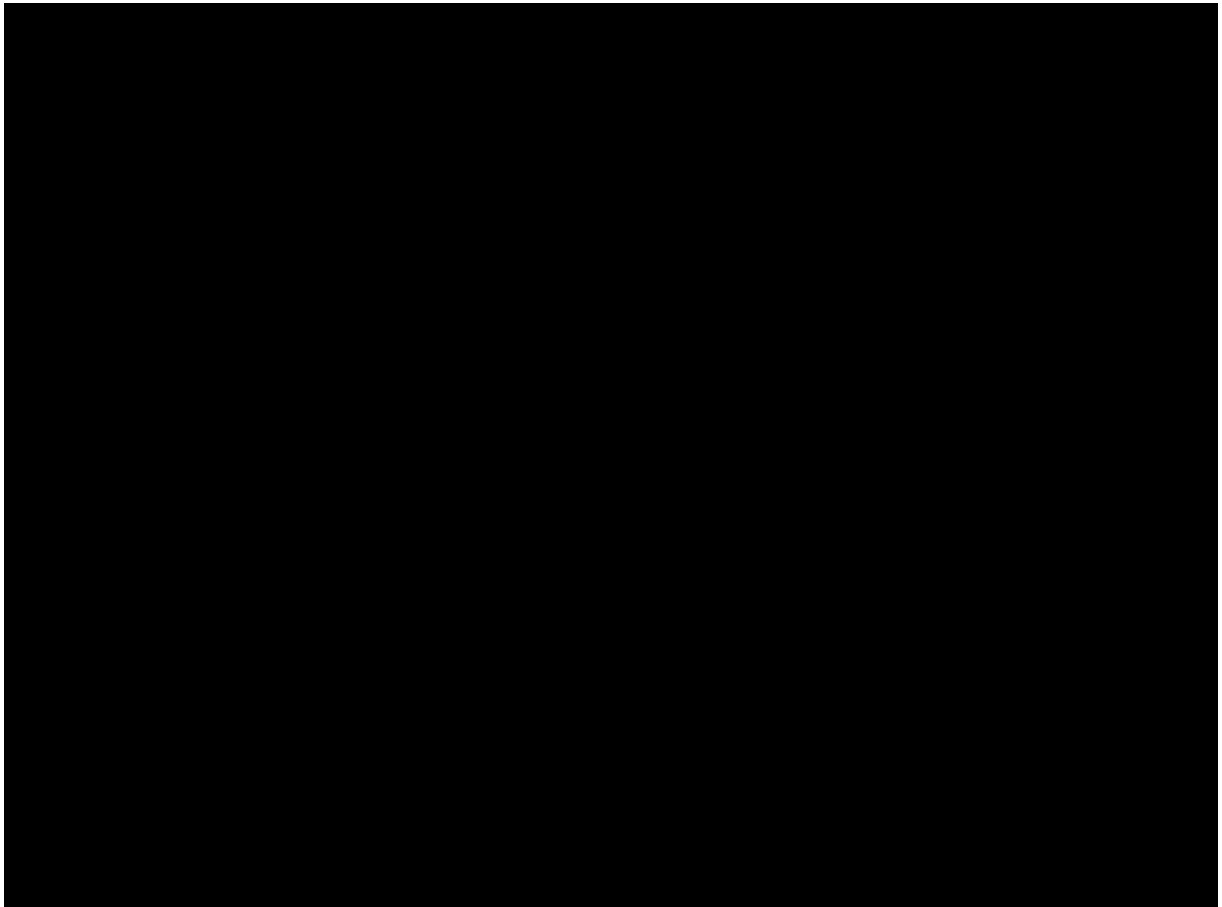
Roadmap

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


SAMSUNG
WU32543A
WARNING
DELICATE PRODUCT
www.Jetheonline.net
LeThe
Joshua Marius

Fontys

Inside a harddisk



Fontys

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:

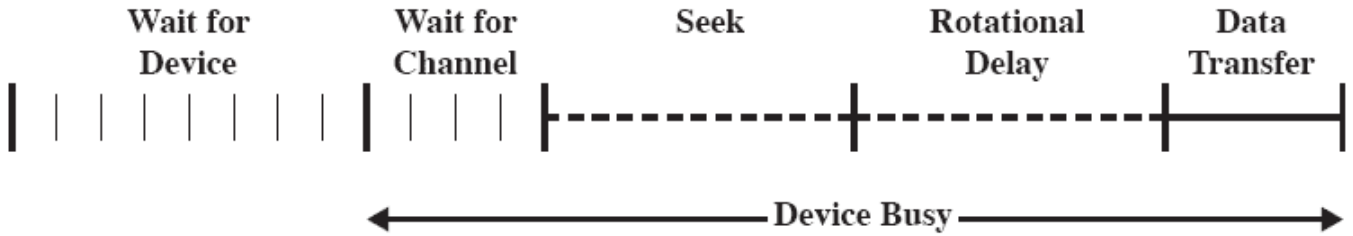
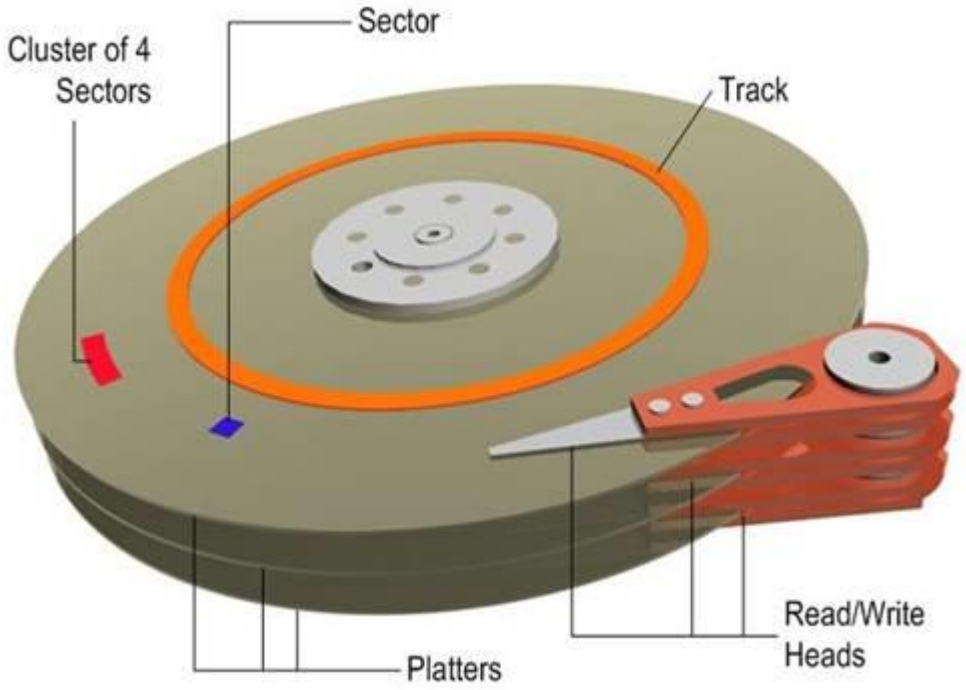


Figure 11.6 Timing of a Disk I/O Transfer

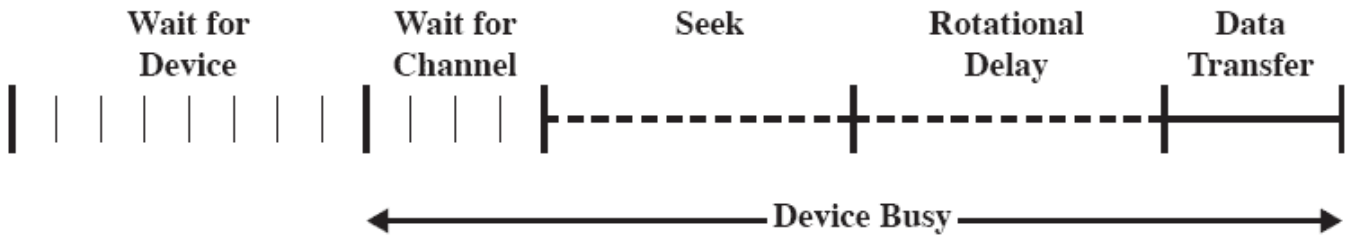
Positioning the Read/Write Heads

- The disk is rotating at constant speed.
- Track selection involves moving the head in a movable-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.



Fontys

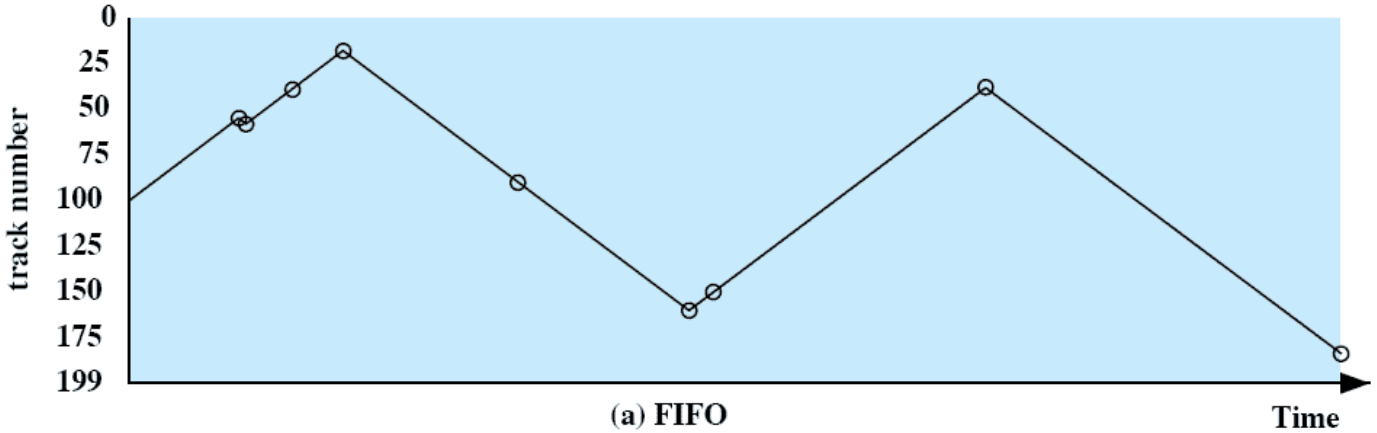
Disk Scheduling Policies

- 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.

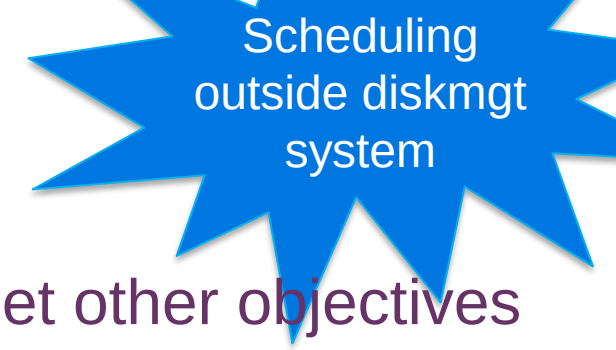
First-in, first-out (FIFO)

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

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



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

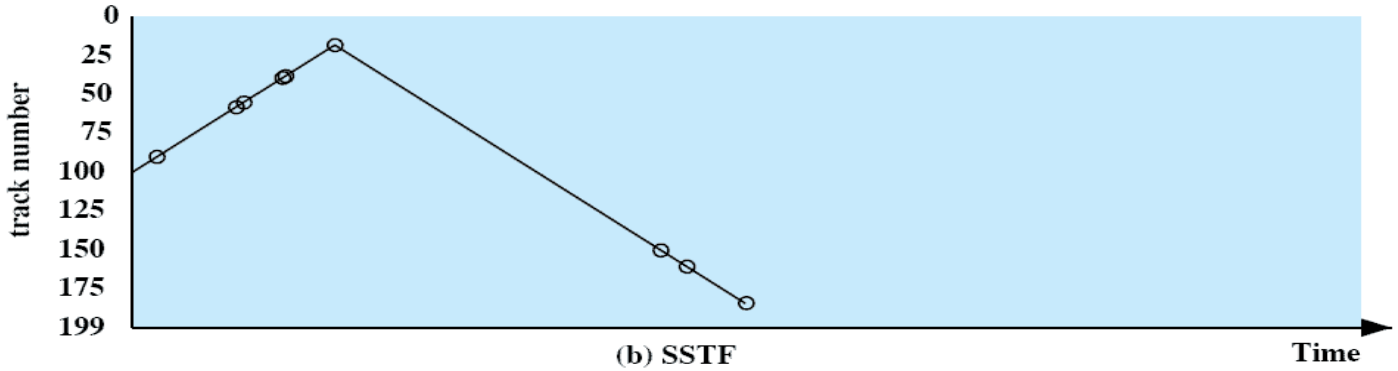
Last-in, first-out

- 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

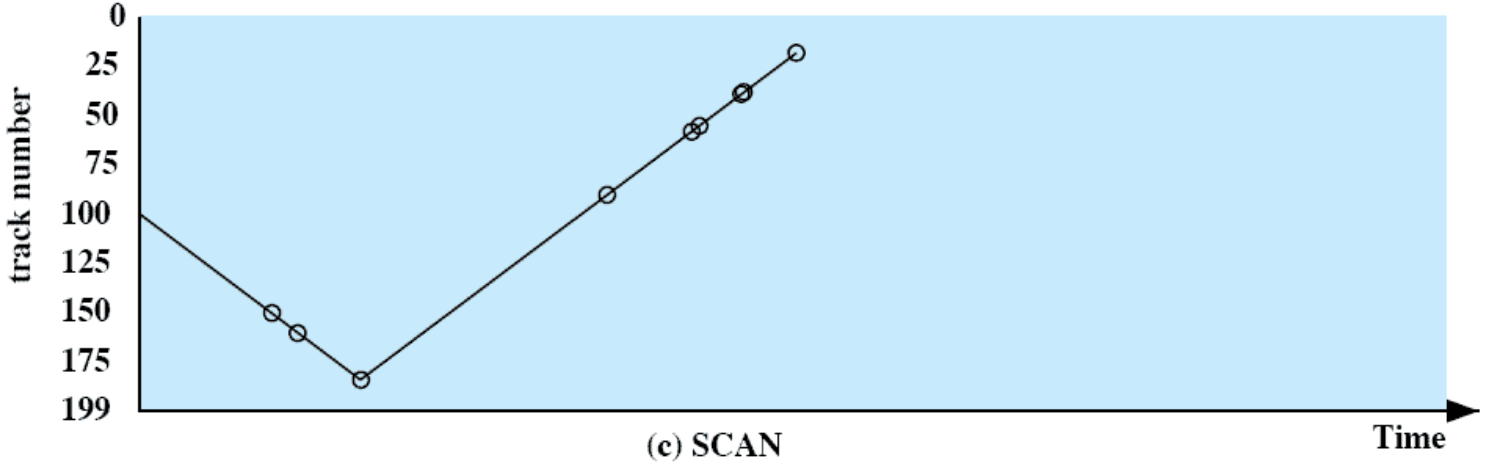
55,
58,
39,
18,
90,
160,
150,
38,
184



SCAN *"elevator" algorithm*

- 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

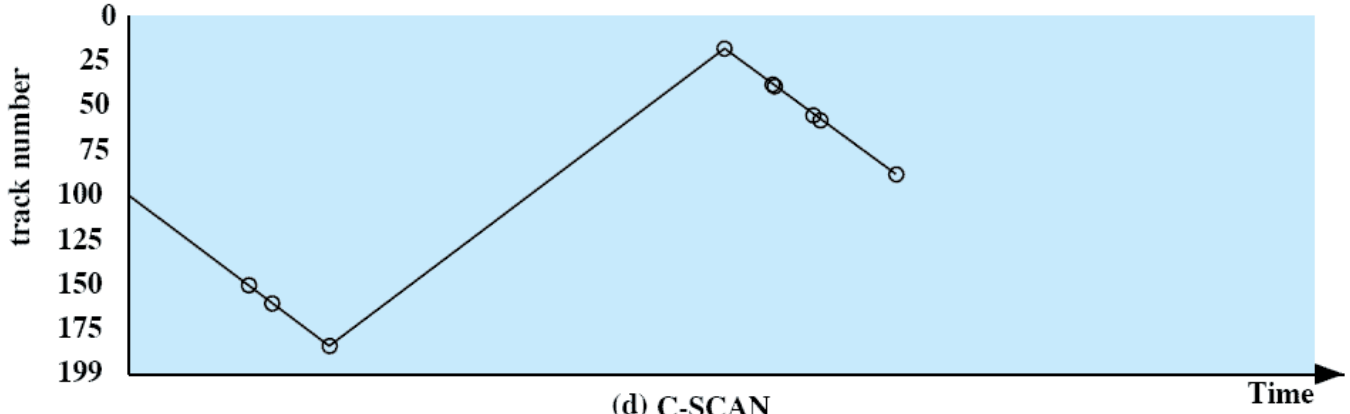


(c) SCAN

C(ircular)-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
- **Advantage**: Reduces maximum delay experienced by new requests

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



(d) C-SCAN

Fontys

N-step-SCAN *Danger of “arm stickeness”*

- **Problem**: SSTF, SCAN,C-SCAN risk monopolisation
- **Solution**: 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

Fontys

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

(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

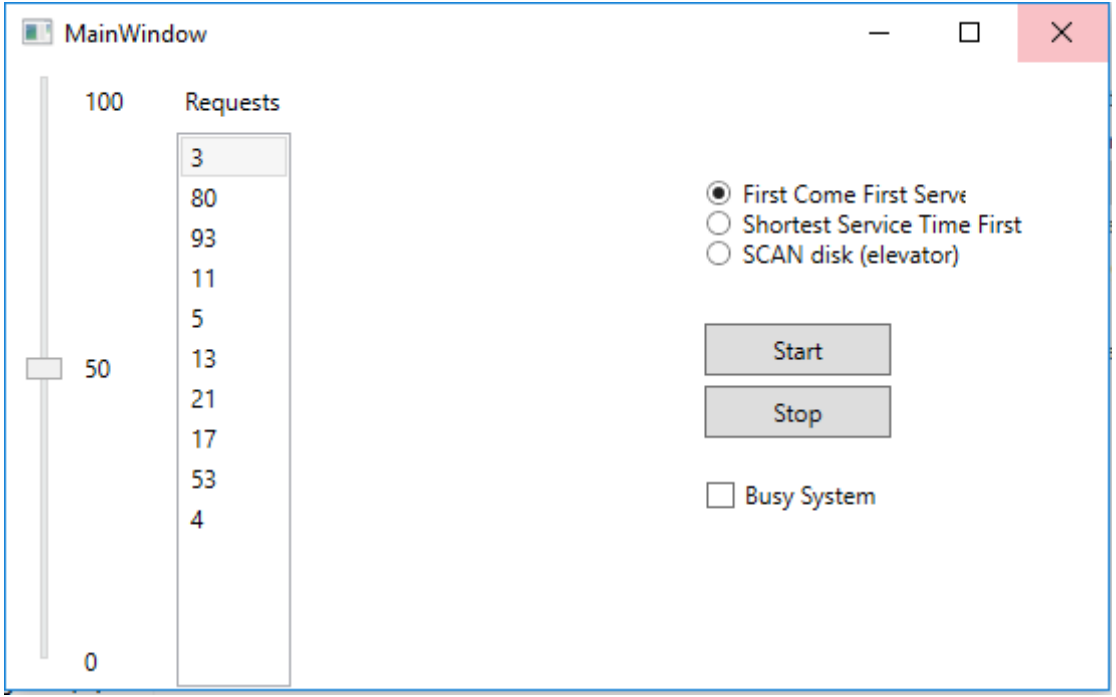
Table 11.3 Disk Scheduling Algorithms

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

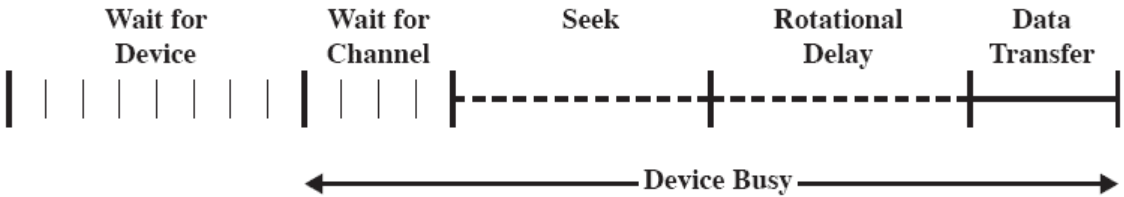
Fontys

› FOR SOCIETY

Demo



SSD – The solution to all problems?



Fontys
> FOR SOCIETY

Roadmap

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

Multiple Disks: *2 distinct improvements*

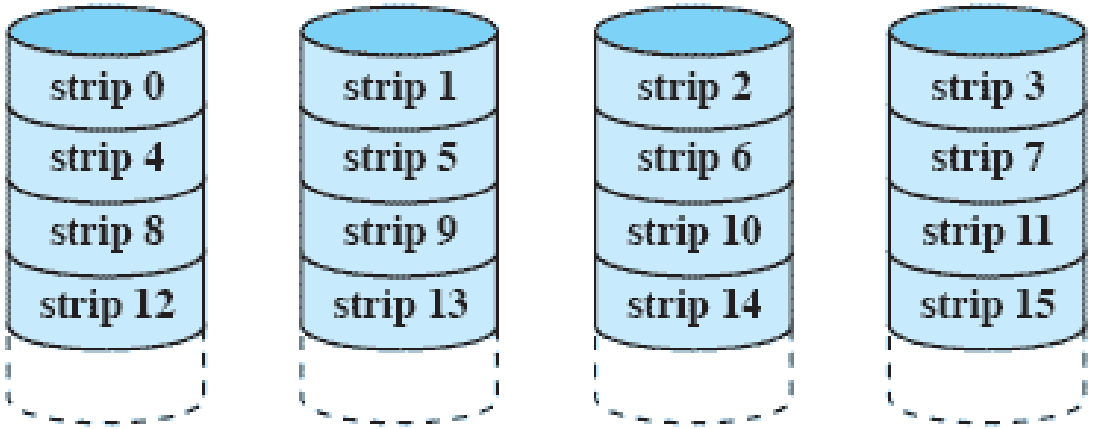
- Disk I/O performance may be increased by spreading the operation over multiple read/write heads
  - Or multiple disks
- Disk failures can be recovered if parity information is stored

RAID

Redundant Array of Independent Disks

- Set of physical disk drives
- Single logical drive
- Data are distributed across the physical drives
- Redundant disk capacity is used to store parity information which provides recoverability from disk failure

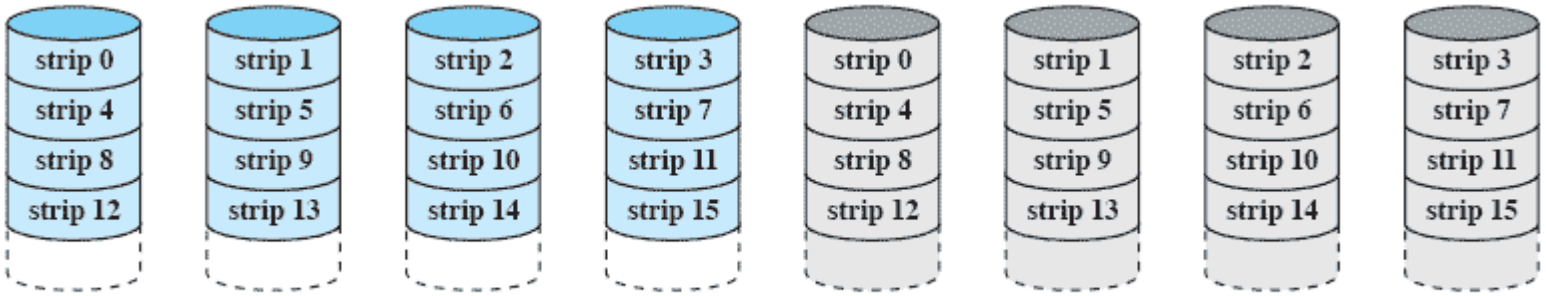
RAID 0 - Striped



- 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)

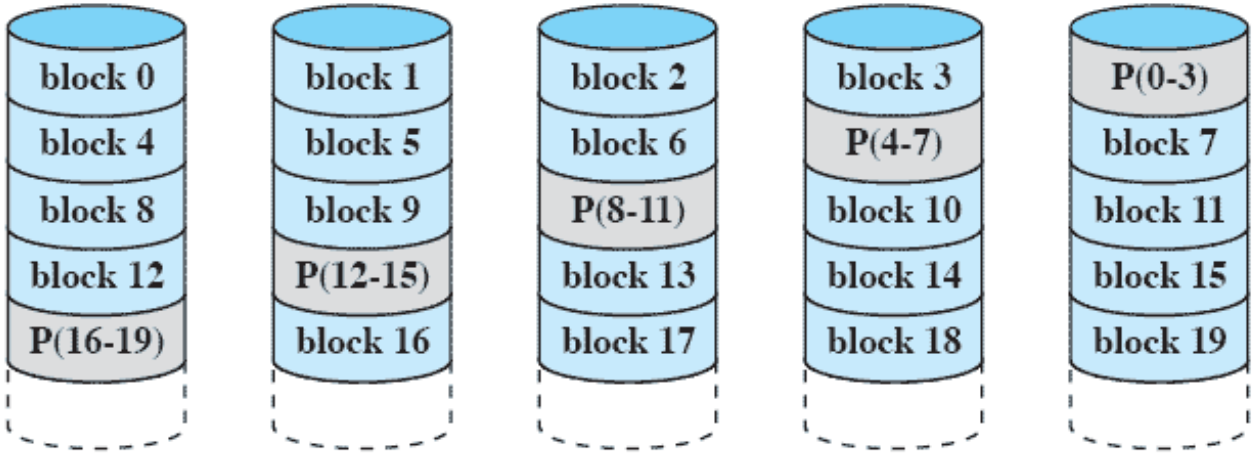
Disadvantage?

RAID 5

Block-level Distributed parity

Avoids bottleneck for parity disk

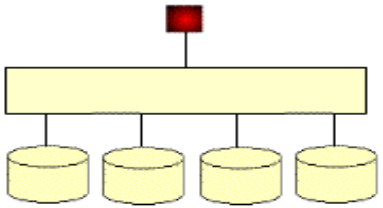
- A bit-by-bit parity strip is calculated across corresponding strips on all data blocks
- The parity bits are stored in the corresponding strip on one of the disks
- Independent disk



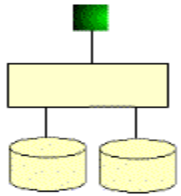
(f) RAID 5 (block-level distributed parity)

Fontys

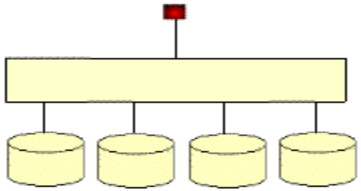
Raid - Animations



Raid 0



Raid 1



Raid 5

Fontys

How does the parity for RAID5 work?

- XOR function result is equal 1 if both arguments are different.
  XOR (0, 1) = 1
  XOR (1, 0) = 1
- XOR function output is equal 0 if both arguments are same.
  XOR (0, 0) = 0
  XOR (1, 1) = 0

- Calculate the XOR of the following data
  | 101 | 010 | 011 |
- And we calculate XOR of those data and place it on 4th drive
  XOR (101, 010, 011) = 100
  XOR (101,010) = 111 and then
  XOR (111, 011) = 100
- So the data on the four drives looks like this below:
  | 101 | 010 | 011 | **100** |

Fontys

What happens when a disk fails?

| 101 | ~~***010***~~ | 011 | 100 |

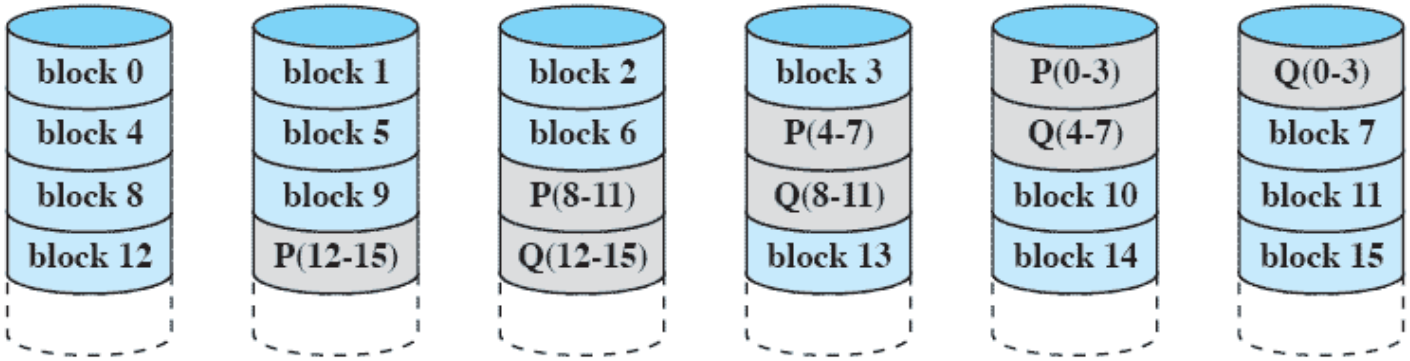
XOR (101, 011, 100) = 010

You can check the missing other drives and XOR of the remaining data will always give you exactly the data of your missing drive.

Fontys

RAID 6 *Dual Redundancy*

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



(g) RAID 6 (dual redundancy)

Fontys

Number Disks

- Raid 0-2 – Minimum 2 disks
- Raid 3-5 – Minimum 3 disks
- Raid 6-10 – Minimum 4 disks

Fontys

Questions?