

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 6

Input/Output Systems

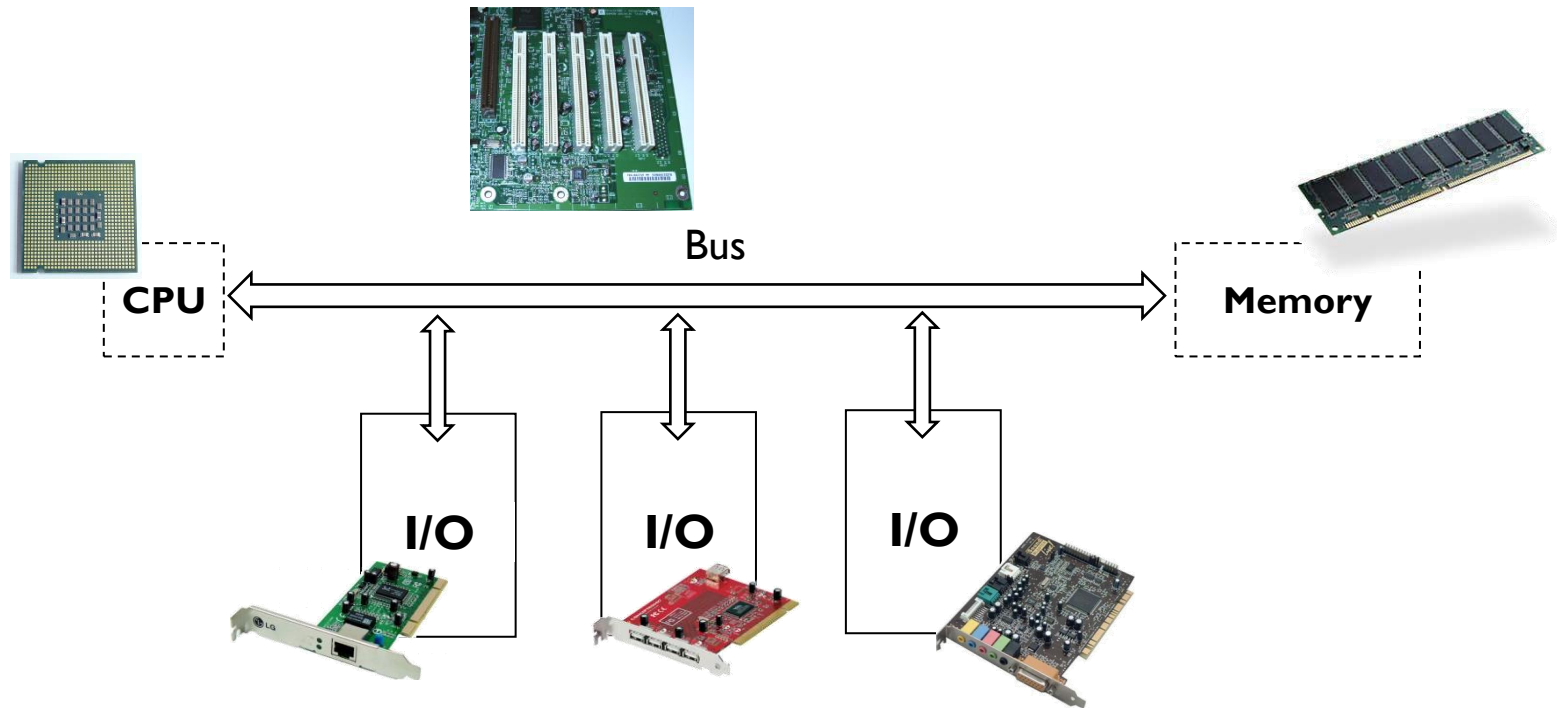
Computer Structure
Bachelor in Computer Science and Engineering



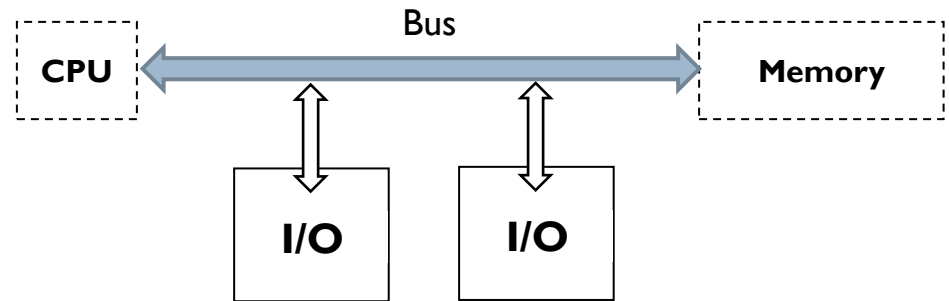
Contents

1. Introduction
2. Peripheral
 - ▶ Concept
 - ▶ Classification and types of peripherals
 - ▶ General structure of a peripheral
 - ▶ Case study: hard disk drive and solid state drives
3. Buses
 - ▶ Structure and operation
 - ▶ Bus hierarchy
4. I/O modules
 - ▶ Structure
 - ▶ Characteristics
 - ▶ I/O techniques

Introduction

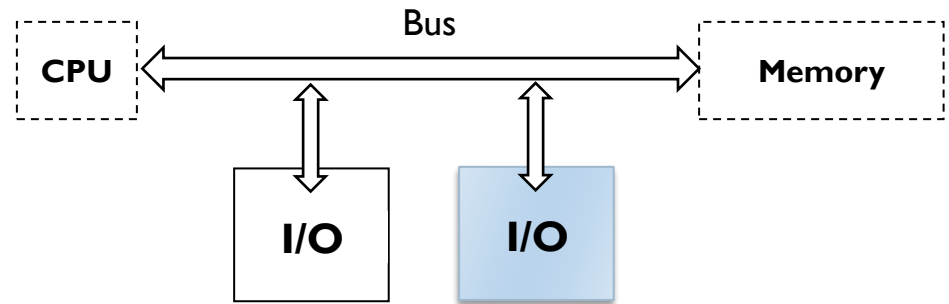


Introduction



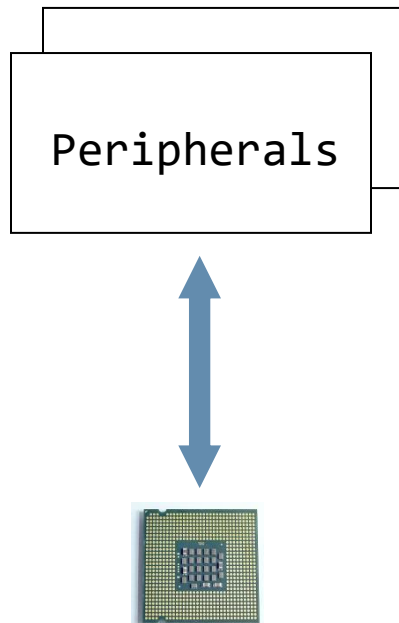
- What is an interconnection bus

Introduction



- ▶ What is a peripheral
- ▶ What is an input/output module
- ▶ How data is accessed from peripherals

Peripheral concept



► Peripheral:

- Any external device that connects to a processor through the **input/output (I/O) modules**.
- They allow storing information or communicating the computer with the outside world.

Classification of peripherals (by use)



► **Communication:**

► **Human-computer**

- (Terminal) keyboard, mouse, ...
- (Printer) plotter, scanner, ...

► **Computer-computer**

- Modem, network adapter

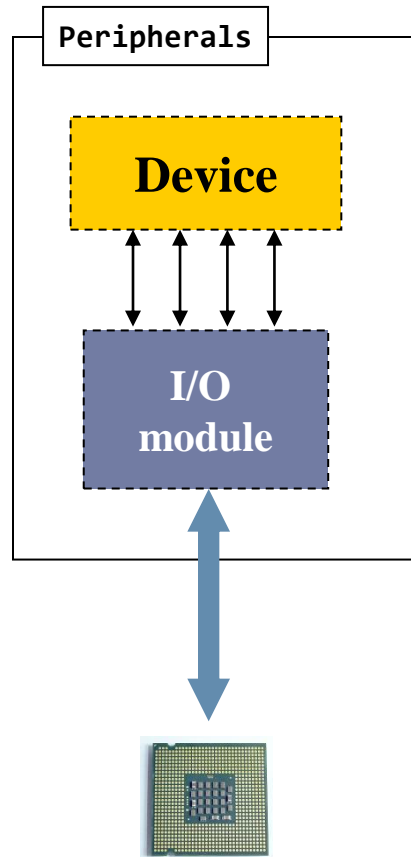
► **Physical environment**

- (read/action) x
(analogical/digital)

► **Storing:**

- Direct access (disks, DVD, ...)
- Sequential access (tapes)

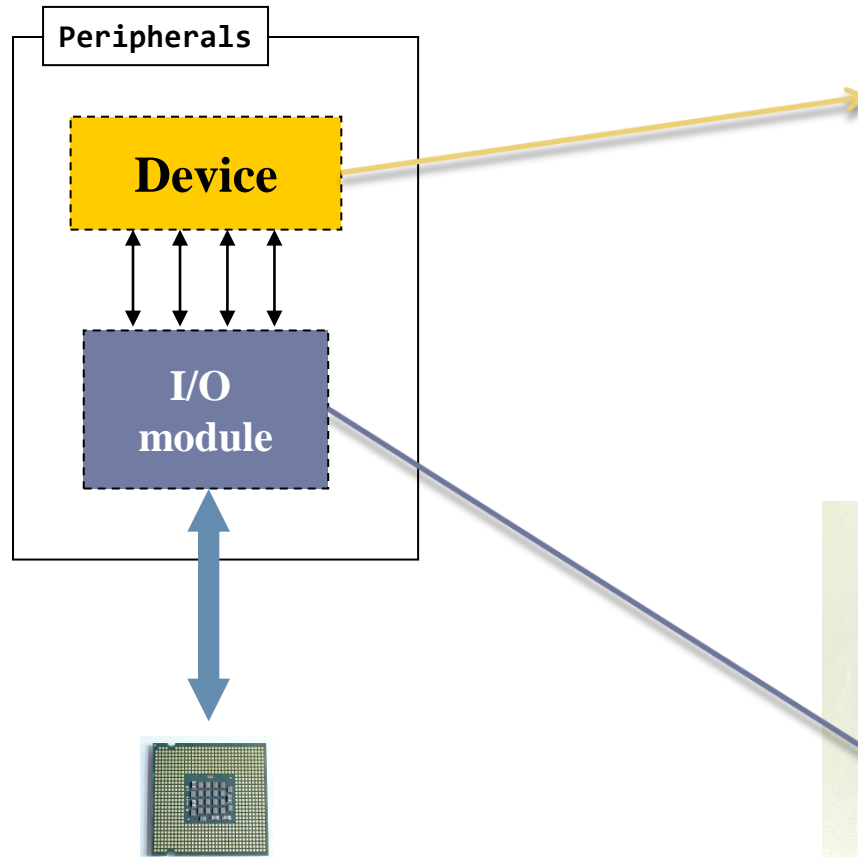
General structure of a peripheral



- ▶ Consisting of:
 - ▶ **Device**
 - ▶ Hardware that interacts with the environment
 - ▶ **I/O module (also I/O unit)**
 - ▶ Also called **controller**
 - ▶ Interface between the device and the processor, which hides the particularities of the processor.

Example:

Disk drive



A little bit of history...



- ▶ The first hard disk was introduced in 1956
 - ▶ First disk drive was built in 1956
 - ▶ It was called IBM RAMAC 305
 - ▶ 50 aluminum disks of 61 cm (24") diameter
 - ▶ 5 MB of data
 - ▶ Spun at 3,600 revolutions per minute (RPM)
 - ▶ Had a transfer speed of 8.8 Kbps
 - ▶ 35000\$ per year rental
 - ▶ Weighed about one ton

A little bit of history...

- ▶ In 1980 appeared the first 5 1/4" disk
 - ▶ 5 MB
 - ▶ 10 000 \$



- ▶ In 1997 appeared the first disk with 15000 RPM

A little bit of history...



1956

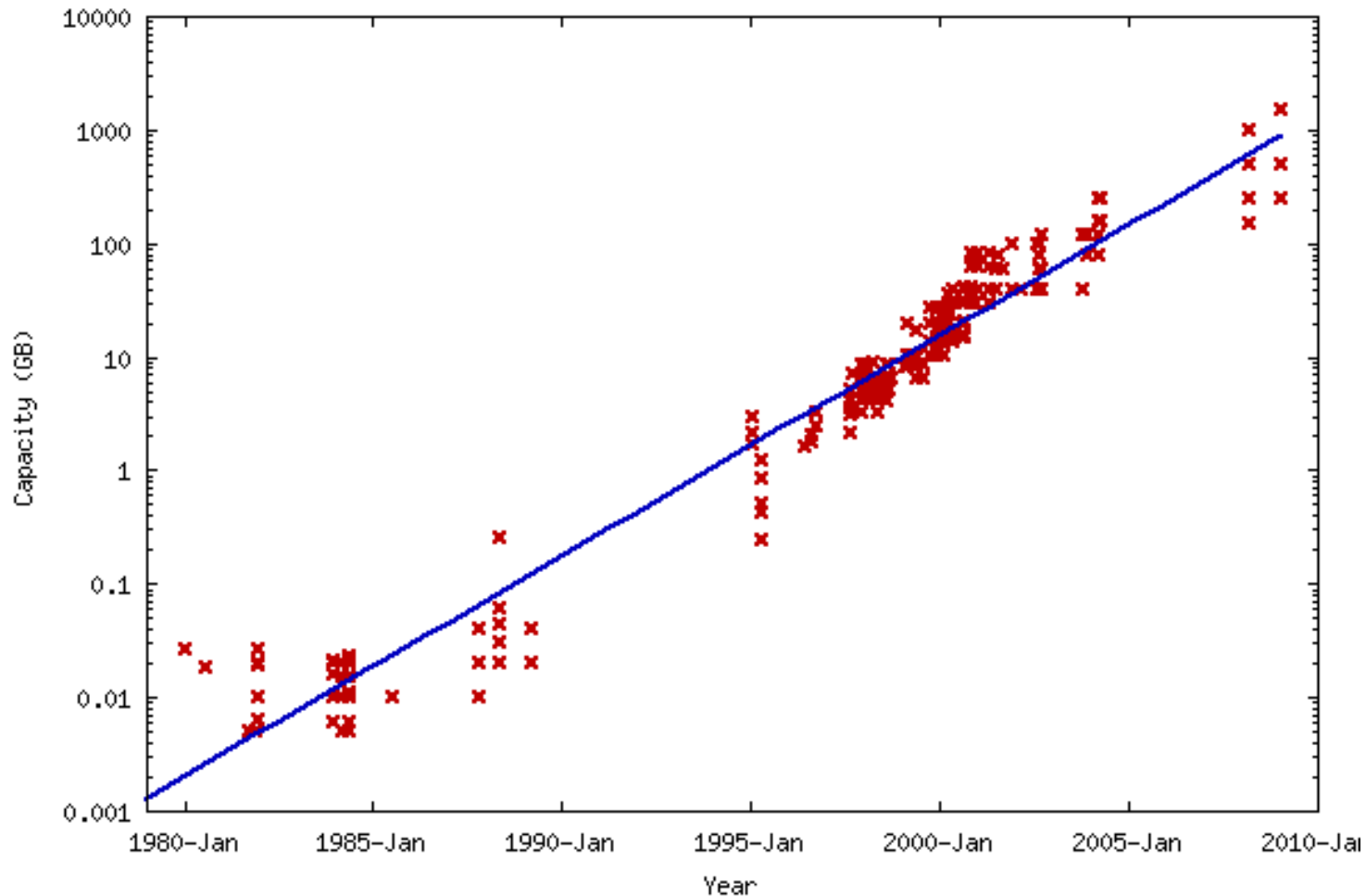


80's



2005

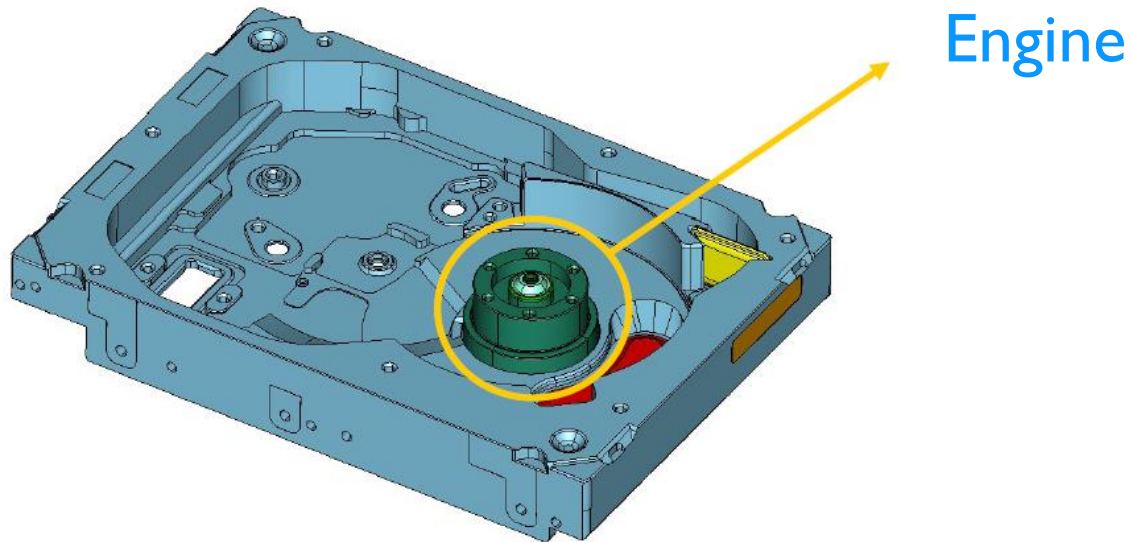
A little bit of history...



A little bit of history...

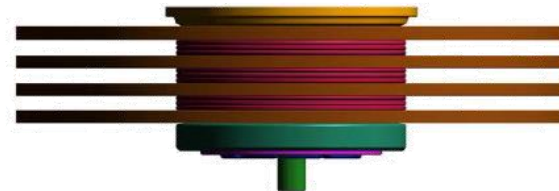
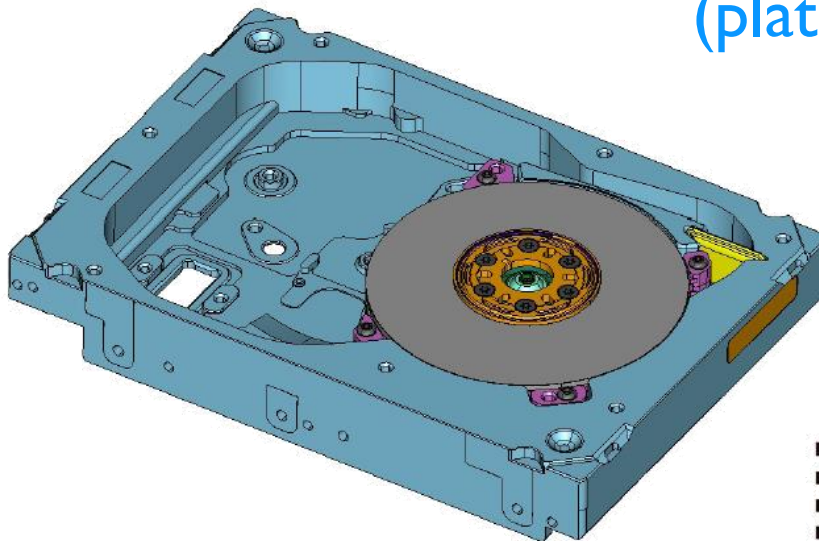
	Annual growth rate
Capacity	1.93 / year
Cost	0.60 / year
Performance	0.05 / year

Anatomy of a hard disk drive



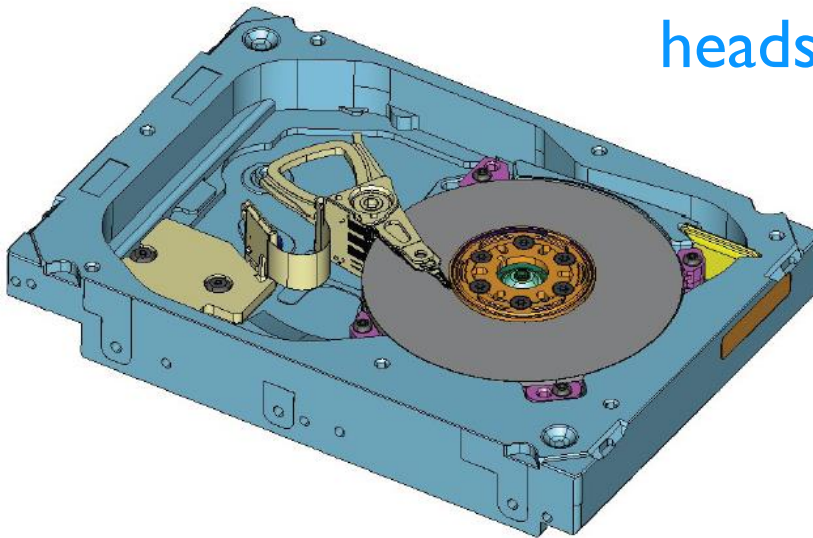
Anatomy of a hard disk drive

Disks
(plates)

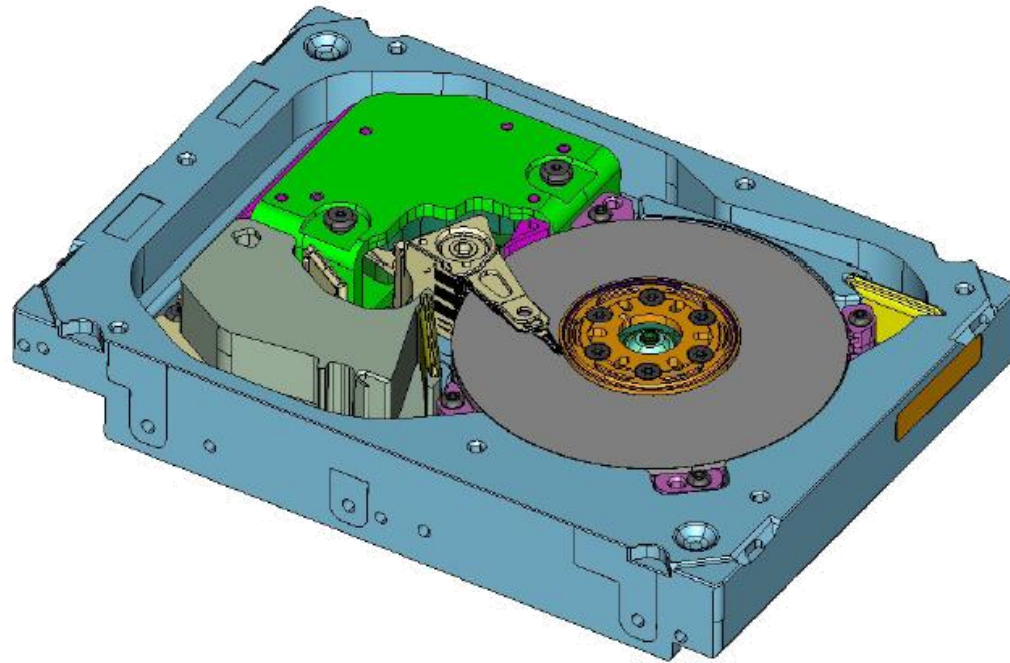


Anatomy of a hard disk drive

Read/write
heads



Anatomy of a hard disk drive



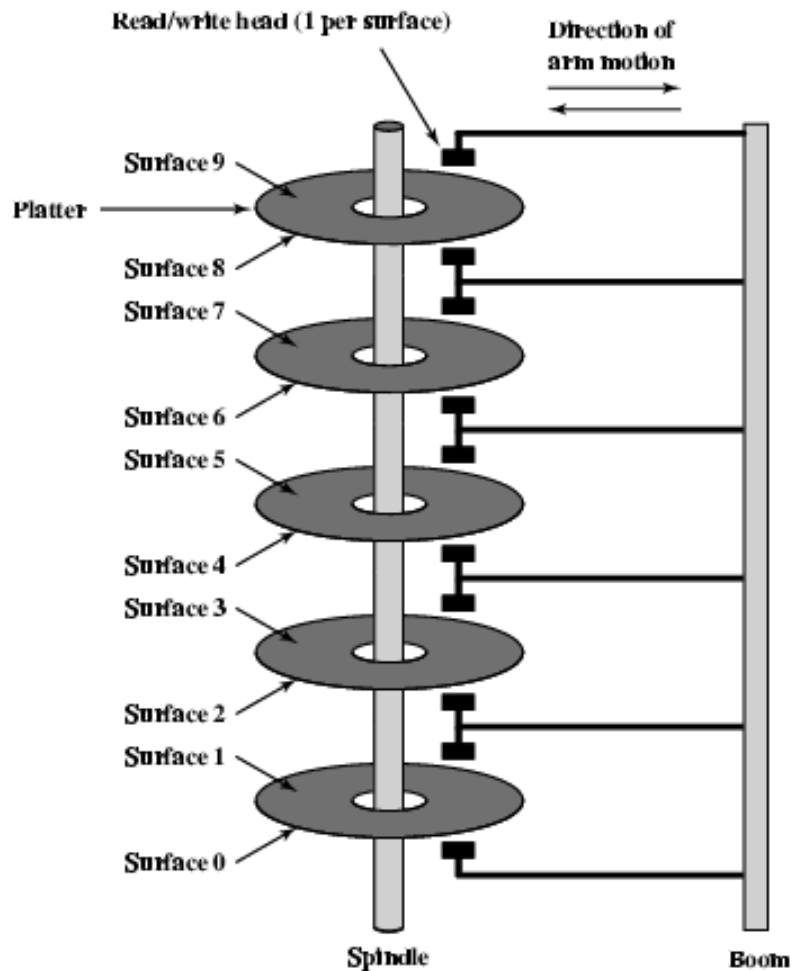
control and
mechanic module

Anatomy of a hard disk drive



- ▶ **Disk controller**
 - ▶ Command scheduling
 - ▶ Error correction
 - ▶ Optimization
 - ▶ Integrity check
 - ▶ Revolutions per minute (RPM) monitoring
 - ▶ Disk cache

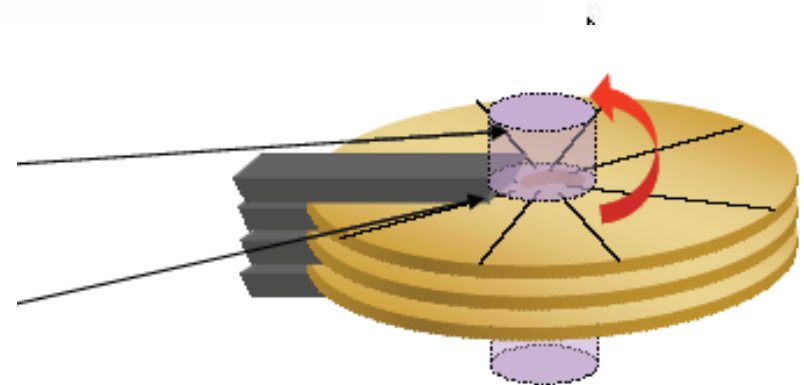
Multiple plates



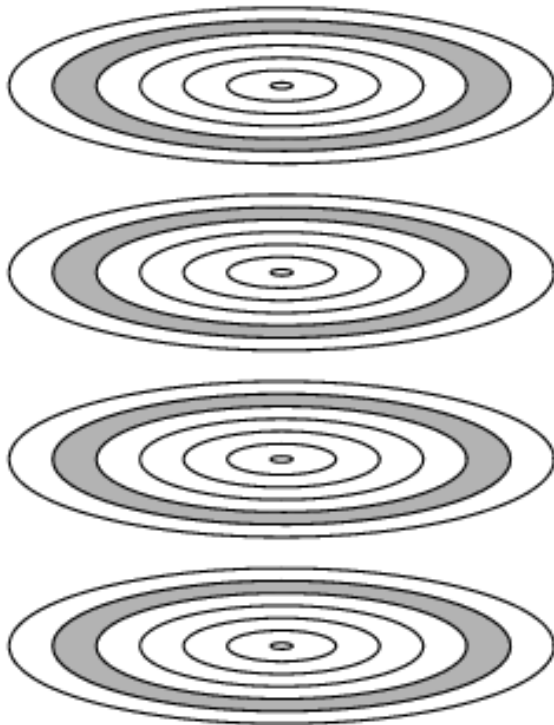
<http://www.snia.org>



Rotation

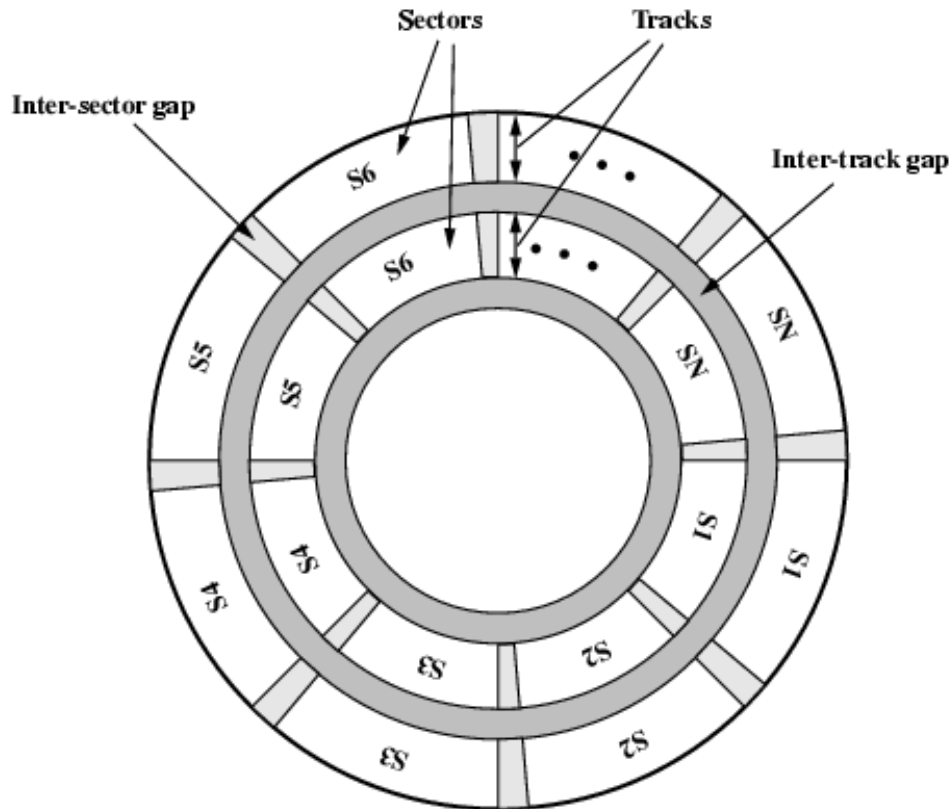


Cylinders



- **Cylinder:**
information accessed by
all heads in a rotation

Storage: tracks and sectors



Track:

- ▶ Concentric ring in plate

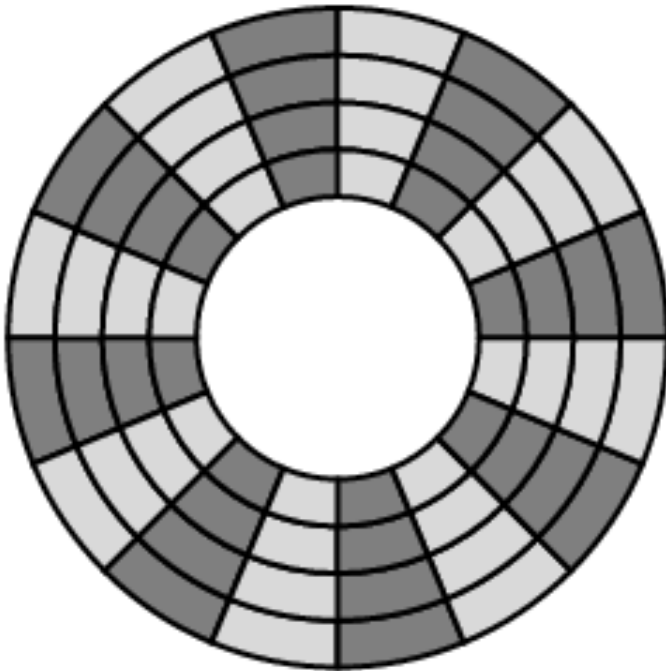
Sector:

- ▶ Disk area division performed on formatting (typically 512 bytes)

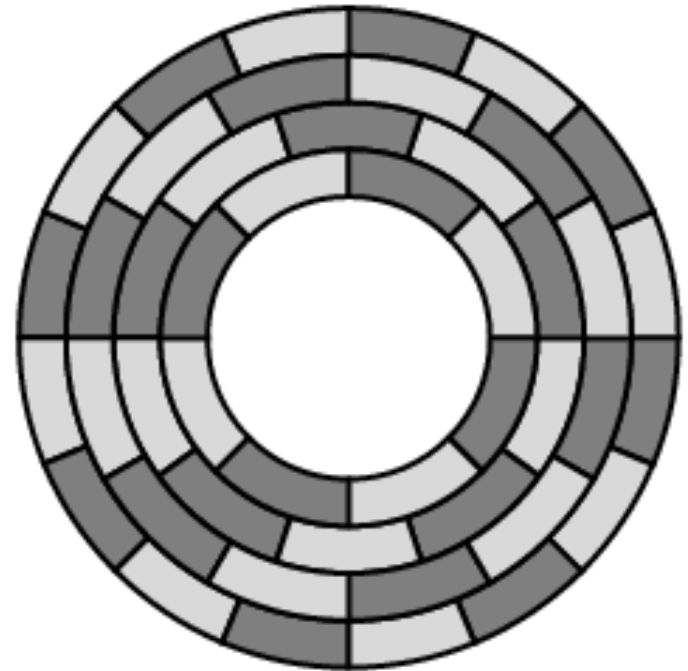
Block:

- ▶ File System writes in blocks
- ▶ Sector groups

Distribution of sectors

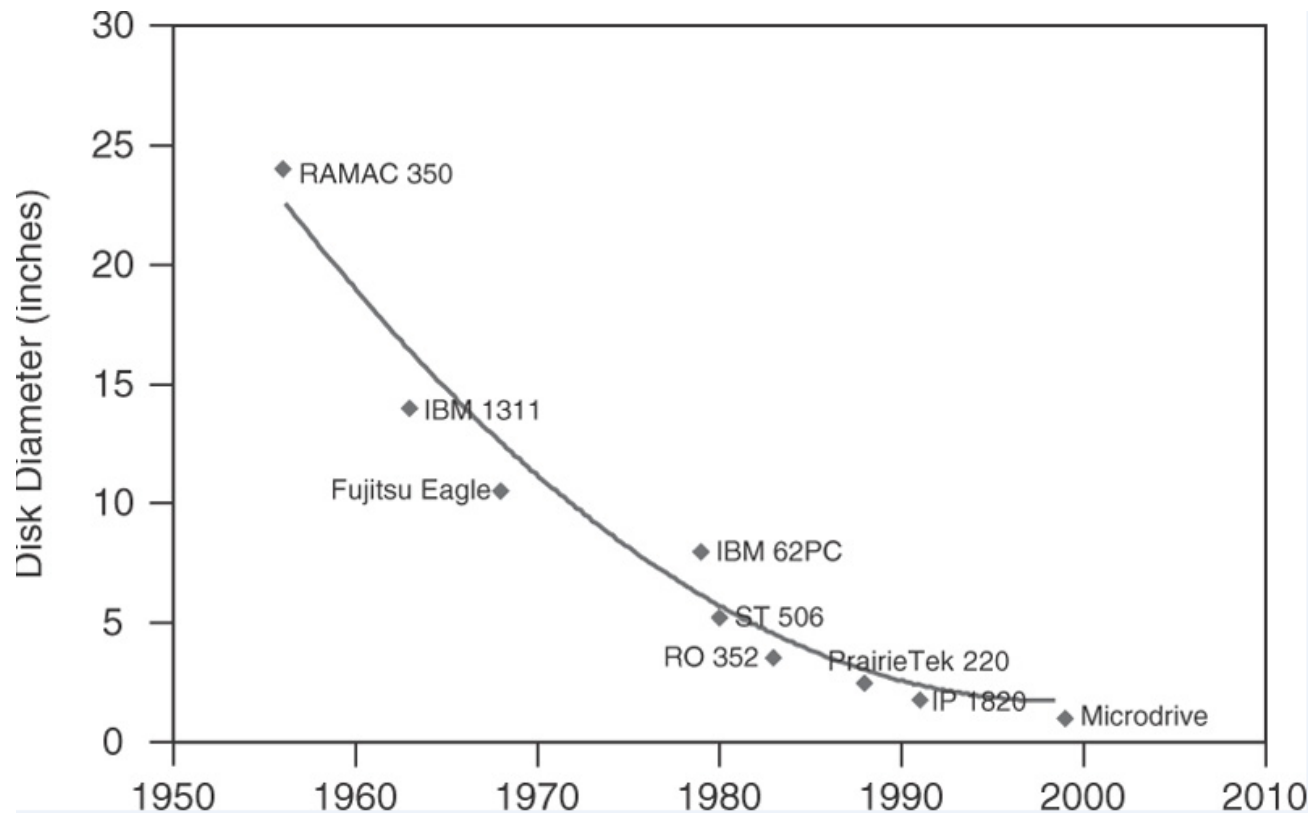


(a) Constant angular velocity



(b) Multiple zoned recording

Evolution of disk sizes



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Capacity

- ▶ Bits per inch
 - ▶ They depend on the read/write head, the recording medium, the rotation of the disk and the speed at which the bus can accept data.
- ▶ Tracks per inch
 - ▶ They depend on the read/write head, the recording medium, the precision with which the head can be positioned and the ability of the disk to rotate in a perfect circle.

Storage capacity

- ▶ For constant angular velocity disks:

- ▶ n_s : number of surfaces
- ▶ p : tracks per surfaces
- ▶ s : sectors per track
- ▶ t_s : bytes per sector

$$Capacity = n_s \times p \times s \times t_s$$

- ▶ For multiple zone recording:

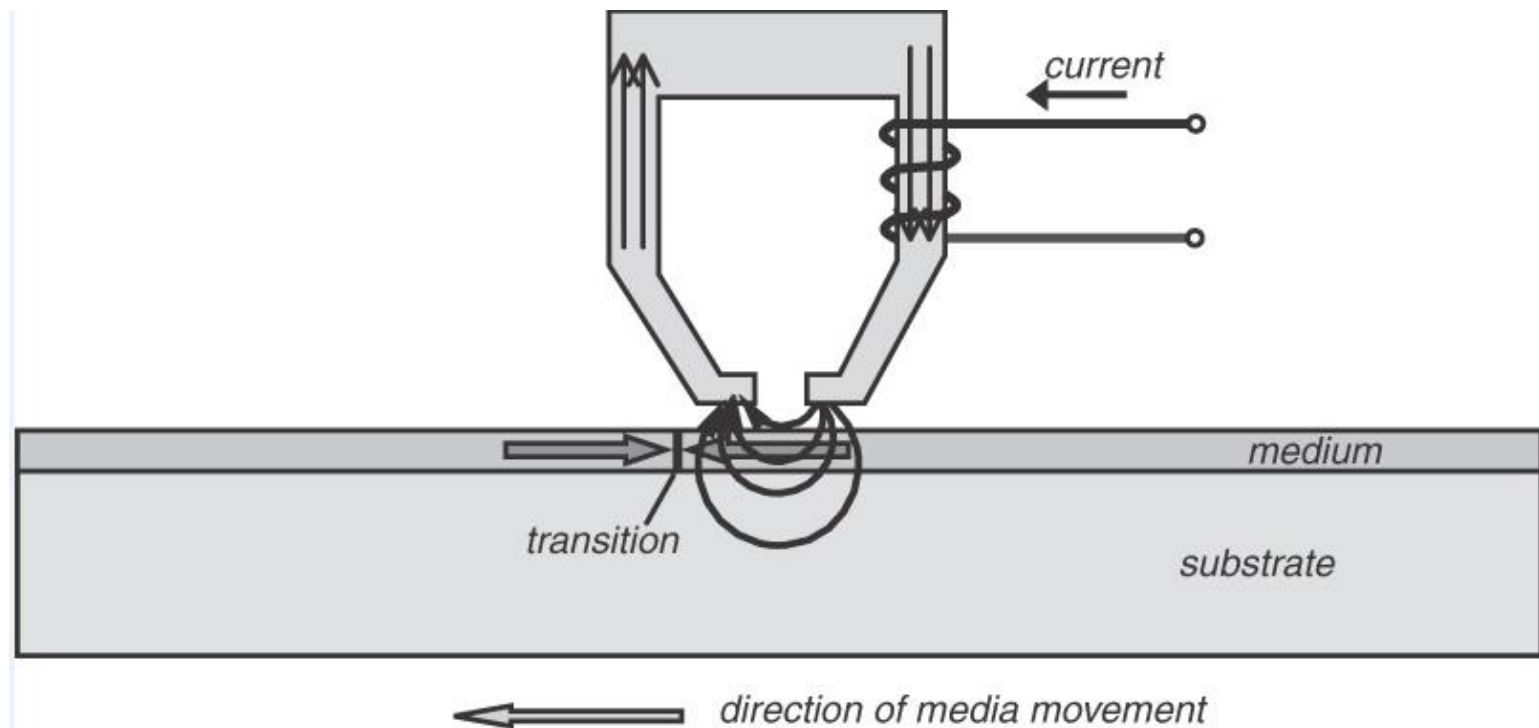
- ▶ z : number of zones
- ▶ p_i : number of track per zone i
- ▶ s_i : sectors per track in zone i

$$Capacity = n_s \times t_s \times \sum_{i=1}^z (p_i \times s_i)$$

Recording techniques

- ▶ Over the last decade the magnetic recording has achieved 100% growth of Areal Density (AD)
- ▶ Each bit cell in a track is composed of multiple magnetic grains
- ▶ The size or the number of magnetic grains in a bit cell cannot be scaled much below a diameter of ten nanometers due to:
 - ▶ Superparamagnetic effect
 - ▶ Ambient temperature would become magnetic grains unstable
- ▶ Recording techniques:
 - ▶ Longitudinal recording: store data in a longitudinal way over a horizontal plane
 - ▶ Perpendicular recording: data are stored in vertical way, increasing the disk capacity

Read/write head



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

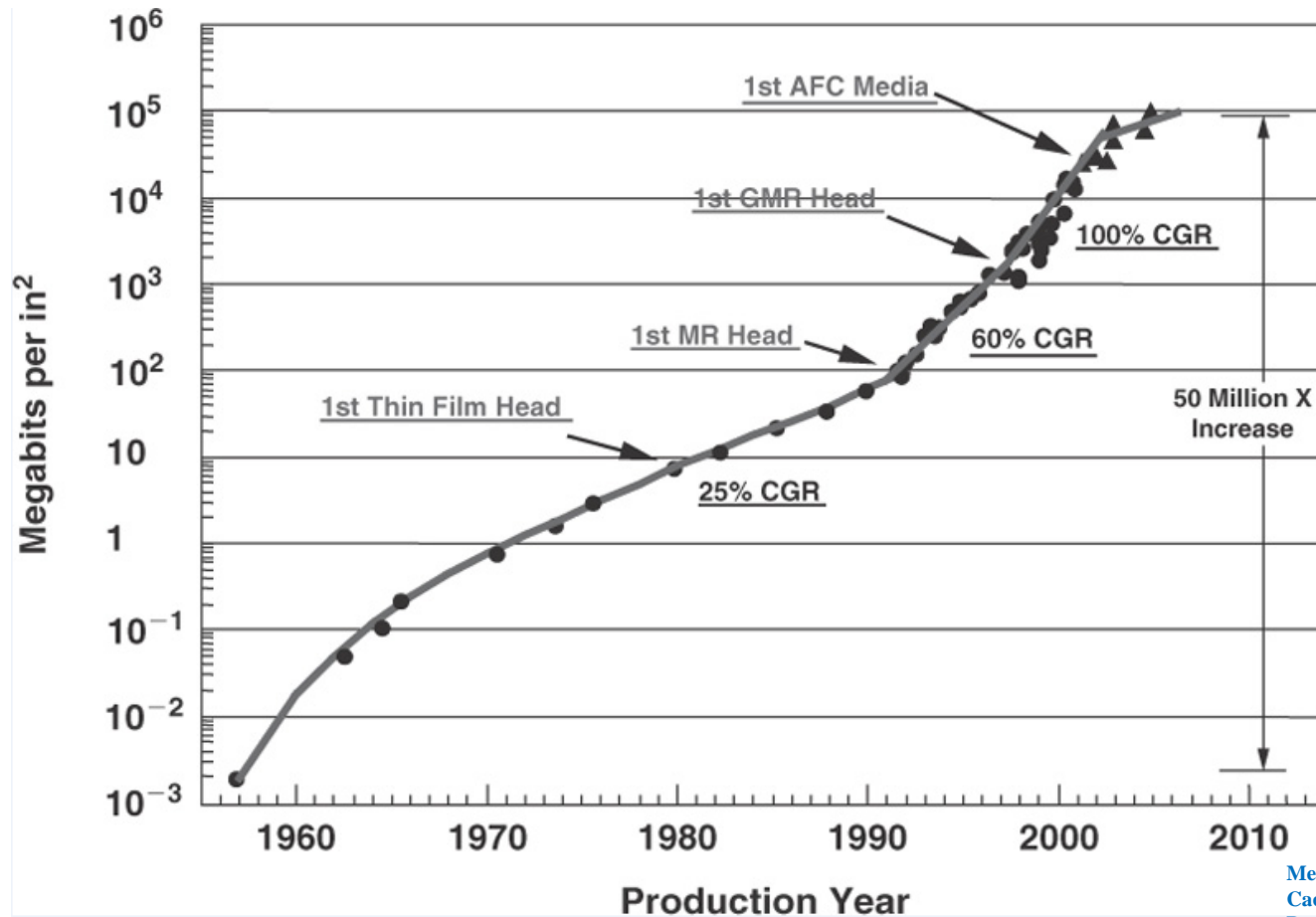
Areal density

- ▶ Improvements in disk capacity are expressed as an improvement in **areal density** (number of bits that can be recorded per square inch):

$$\text{Areal density (AD)} = \frac{\text{Tracks on a disk surface}}{\text{Inch}} \times \frac{\text{Bits on a track}}{\text{Inch}}$$

- ▶ Until 1998 the annual increase rate was 29%
- ▶ 1998-1997 the annual increase rate was 60%
- ▶ 1997-2003 the annual increase rate was 100%
- ▶ 2003-2011 the annual increase rate was 30%
- ▶ In 2011 the bigger areal density in commercial products was 400 billions of bits per square inch
- ▶ The cost per bit has improved in a factor of 1.000.000 between 1983 and 2011

Evolution of areal density

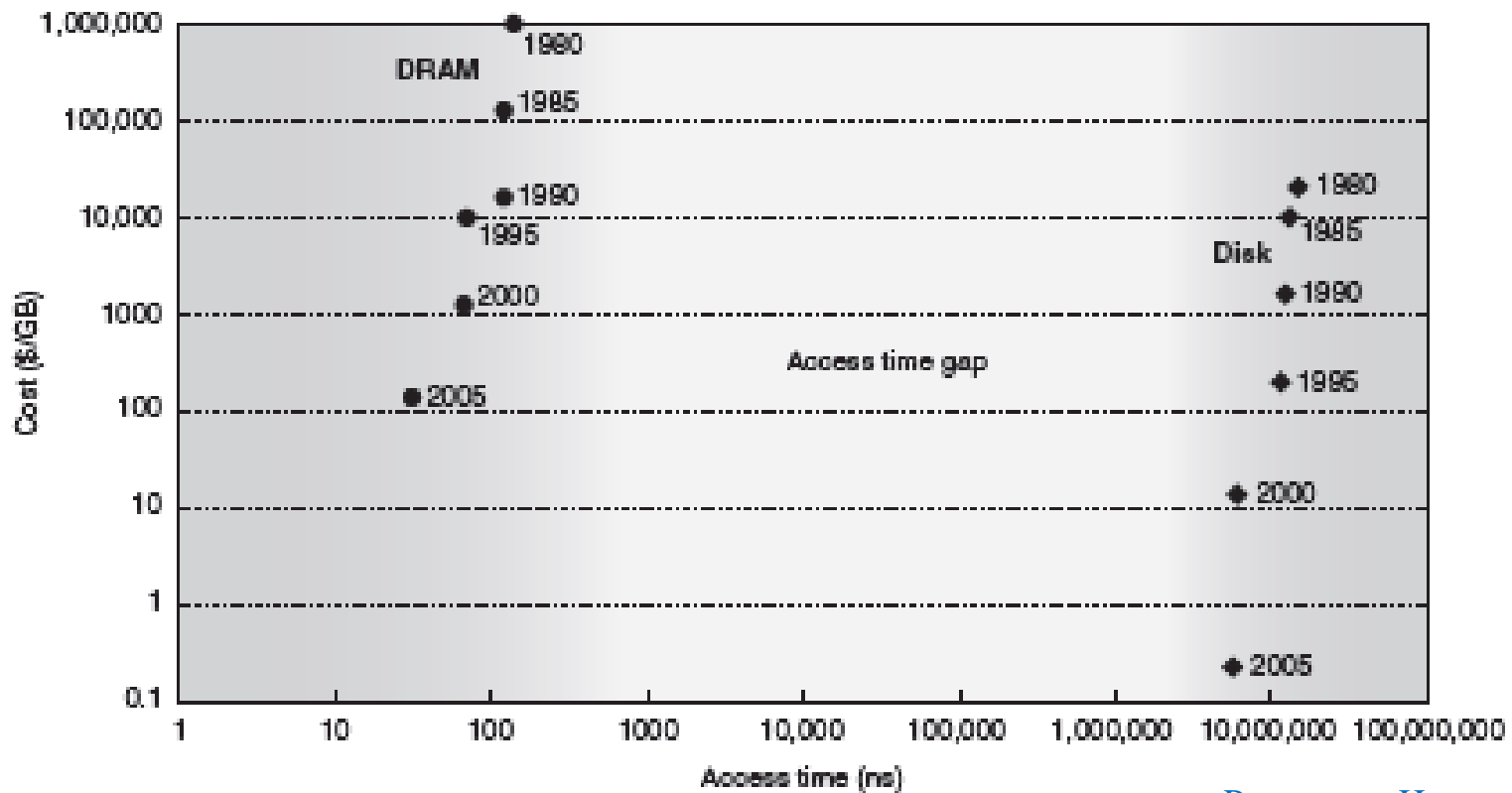


Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Disks and main memory

- ▶ The latency of a DRAM memory is 100.000 less than the latency of a disk
- ▶ The cost per GB in a DRAM memory is 30-150 times the cost per GB of a disk
- ▶ In 2015:
 - ▶ An 8 TB disk transfers 190 MB/s at a cost of 250 \$.
 - ▶ An 8 GB DDR4 module transfers 25 GB/s and costs approx. 70\$.

Disks and main memory



Patterson y Hennesy

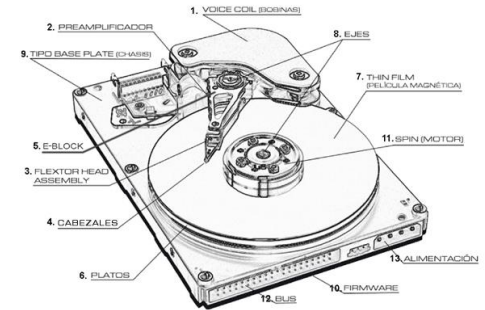
Addressing

- ▶ Types of addressing:
 - ▶ **Physical addressing**: cylinder-track-sector. A sector is determined by these three values.
 - ▶ **Logical blocks addressing (LBN)**
 - ▶ Each sector has a logical number and the mapping is done by the disk
- ▶ Current disk controllers do the mapping between LBN and physical addresses

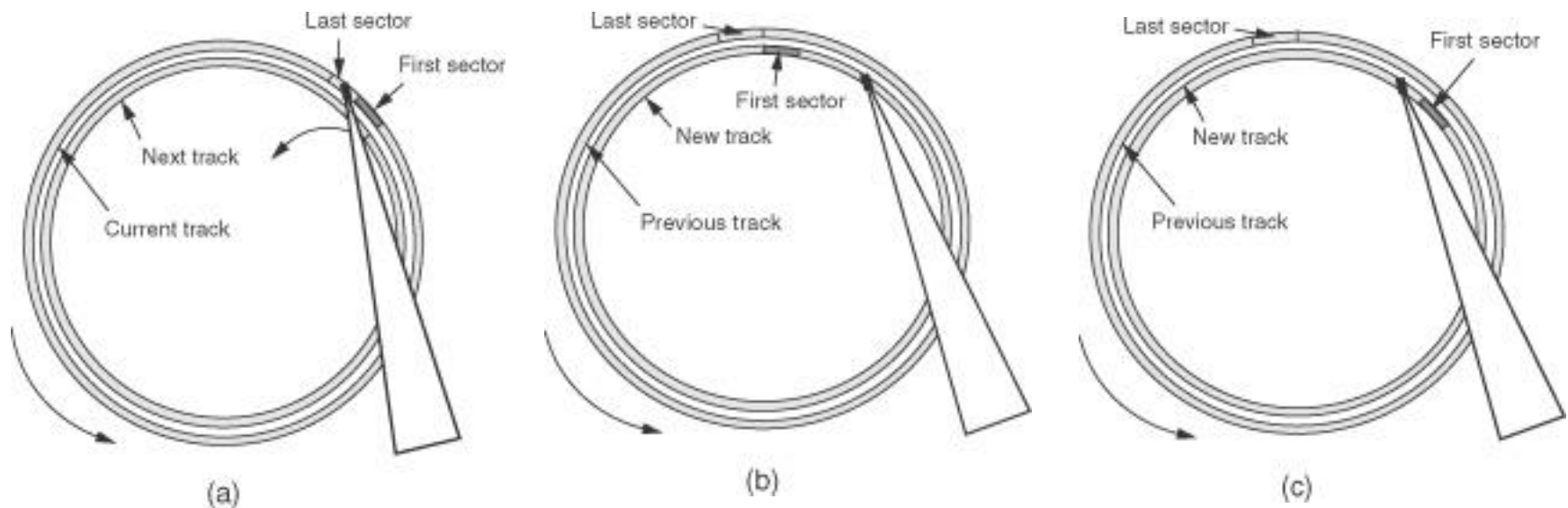
Access time

▶ $T_{\text{access}} = T_{\text{seek}} + T_{\text{latency}} + T_{\text{transfer}}$

- ▶ **Seek time** (T_{seek}): time to move the head from the current cylinder to the target cylinder
- ▶ **Rotational latency** (T_{latency}): time waiting for the rotation of the disk to bring the required sector under the read-write head
- ▶ $T_{\text{latency}} = \text{Tiempo medio para recorrer media pista}$
- ▶ **Data transfer time** (T_{transfer}): time required to traverse a sector and transfer the data from it.
- ▶ $T_{\text{transfer}} = \text{Amount of data} / \text{data transfer rate}$



Seek and rotation



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Seek time

- ▶ When the areal density increase the capacity of cylinders increase too:
 - ▶ The probability of reducing the number of seeks is increased
 - ▶ Increase the probability that the next data to request are in the same cylinder, reducing in this way the number of seeks

Rotational latency

- ▶ Rotational latency is generally calculated as half the time it takes the disk to do one revolution:

$$T_{rotate} = \frac{1}{2} \times \frac{60 \times 10^3}{RPM}$$

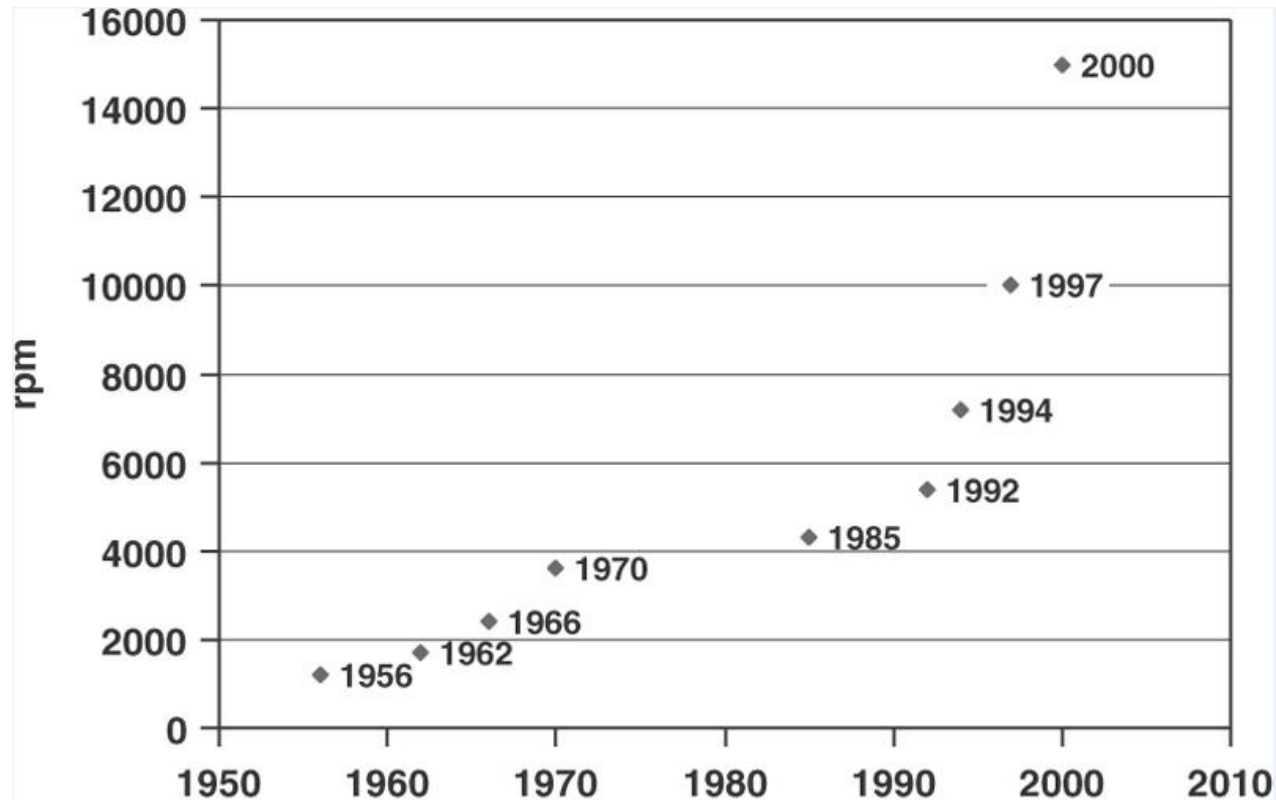
■ *Zero-latency access*

- ▶ Transfer the data as soon as the head is on the desired track to a buffer where the blocks are then reordered.

Rotational speed and rotational latency

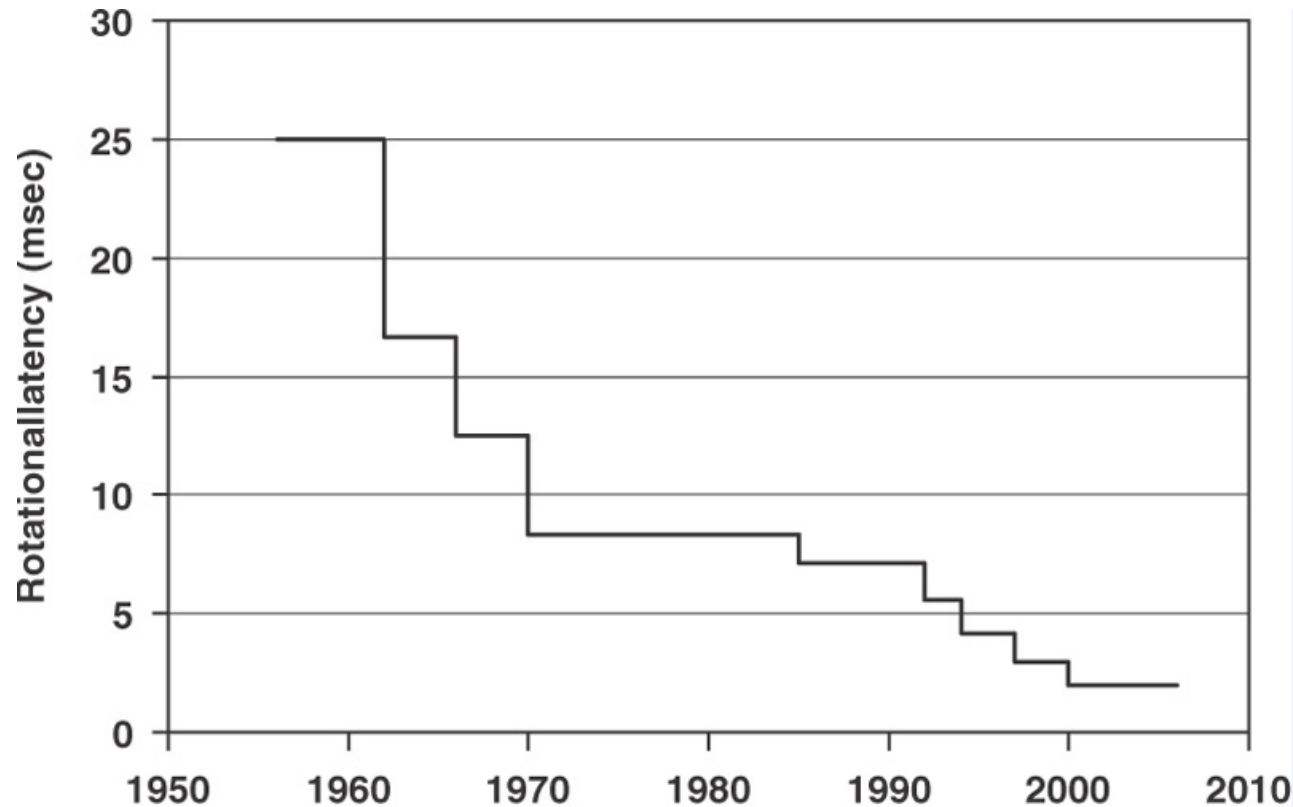
Rotational Speed	Rotational Latency (en ms)
5400	5.6
7200	4.2
10000	3.0
12000	2.5
15000	2.0

Evolution of RPM



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Evolution of rotational latency



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Data Transfer time

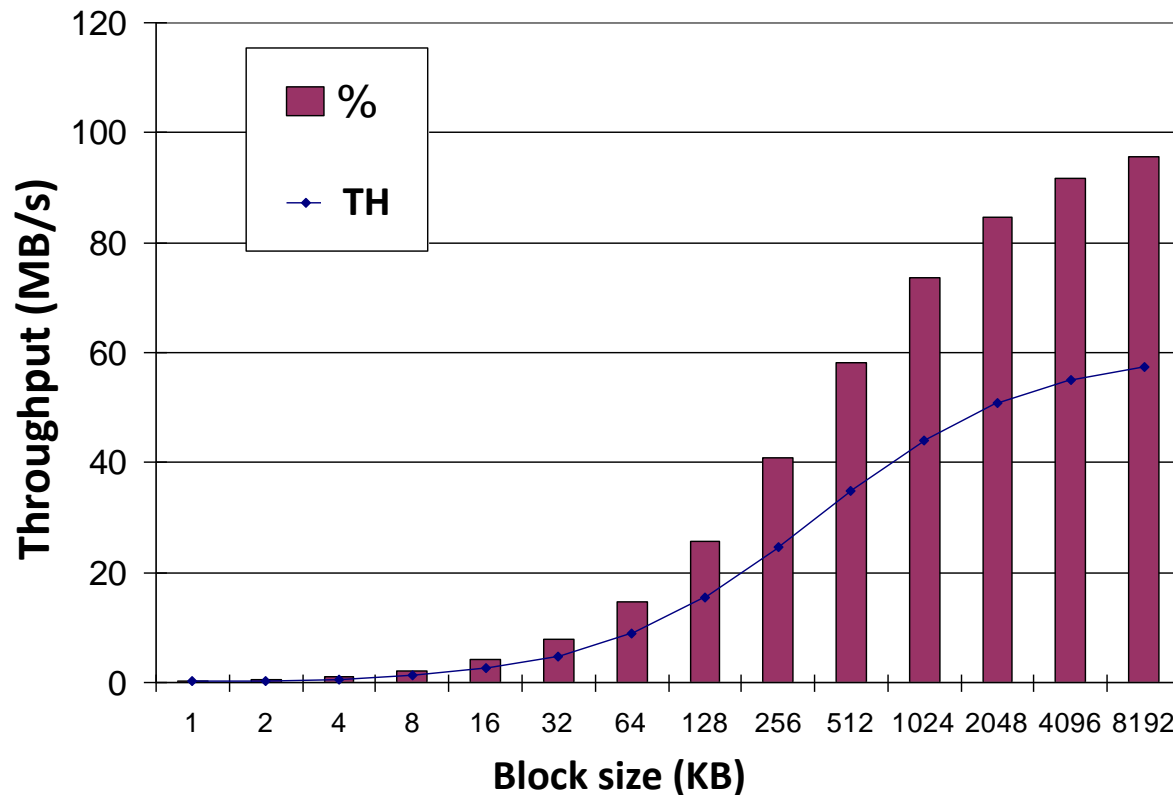
- ▶ Two elements:
 - ▶ External data rate to measure the transfer rate between memory and disk cache
 - ▶ Transfer rate between disk cache and disk storage media
- ▶ Data transfer time can be calculated as:

$$T_{transfer} = \frac{N_{request}}{N_{track}} \times \frac{60}{RPM}$$

- ▶ Where N_{track} denotes the number of sectors on a track, and $N_{request}$ the data length of a request measured in sectors.
- ▶ The ratio of the sectors of the outmost zone to that of the innermost zone ranges from 1.43 to 1.58.

Effect of the request size

- Effect of the request size ($t_a=6$ ms y $T_H = 60$ MB/s)



Disk controller

- ▶ Circuits and components to control the disk:
 - ▶ Storage interface
 - ▶ Disk sequencer
 - ▶ Error correction code(ECC)
 - ▶ Servo motor
 - ▶ Microprocessor
 - ▶ Buffer controller
 - ▶ Disk cache

Disk cache

- ▶ Exploit the locality
- ▶ Reduce physical access to disk
 - ▶ Reduce the heat dissipation
 - ▶ Increase the performance
- ▶ Usually does not exhibit temporal locality due to SSOO data caches.
- ▶ Typically implements read-ahead (prefetch) to improve spatial locality
- ▶ Disk caches are typically divided into independent segments corresponding to sequential data streams

Disk cache

- ▶ Replacement algorithms
 - ▶ Random Replacement
 - ▶ Least Frequently Used (LFU)
 - ▶ Least Recently Used (LRU)
- ▶ Systems designers generally believe that the size of a cache should be at least 0.1 to 0.3 % of the disk.

Disk scheduler

- ▶ Disk drives maintain a queue with pending requests and disk schedulers are designed to minimize the access time by reordering or rearranging pending request in the queue to reduce the seek time and rotational latency
- ▶ Scheduling algorithms:
 - ▶ First Come First Served (FCFS)
 - ▶ Shortest Seek Time First (SSTF)
 - ▶ SCAN
 - ▶ C-SCAN
 - ▶ LOOK

Other elements

- ▶ **Disk sequencer**: manages the data transfer between storage interface and data buffer
- ▶ **ECC**: responsible for appending ECC codes to the user data and also checking and correcting errors
- ▶ **Servo control**: detects the current position of the disk head and controls track following and seeking
- ▶ **Microprocessor**: controls the general disk behavior
- ▶ **Buffer controller**: provides arbitration and raw signal control of the buffer memory

Exercise

- ▶ How many bytes does a disk drive of 250 GB store?

Remainder

- ▶ 1 KB = 1024 bytes, but in the I.S. is 1000 bytes
- ▶ Manufactures of disk drives and telecommunications use I.S.:
 - ▶ A disk drive of 30 GB stores 30×10^9 bytes
 - ▶ A network of 1 Mbit/s transfers 106 bps.

Name	Abr	Factor	I.S.
Kilo	K	$2^{10} = 1,024$	$10^3 = 1,000$
Mega	M	$2^{20} = 1,048,576$	$10^6 = 1,000,000$
Giga	G	$2^{30} = 1,073,741,824$	$10^9 = 1,000,000,000$
Tera	T	$2^{40} = 1,099,511,627,776$	$10^{12} = 1,000,000,000,000$
Peta	P	$2^{50} = 1,125,899,906,842,624$	$10^{15} = 1,000,000,000,000,000$
Exa	E	$2^{60} = 1,152,921,504,606,846,976$	$10^{18} = 1,000,000,000,000,000,000$
Zetta	Z	$2^{70} = 1,180,591,620,717,411,303,424$	$10^{21} = 1,000,000,000,000,000,000,000$
Yotta	Y	$2^{80} = 1,208,925,819,614,629,174,706,176$	$10^{24} = 1,000,000,000,000,000,000,000,000$

Exercise

- ▶ A disk drive has a rotational speed of 7200 rpm and a constant areal density of 604 sectors per track. The average access time is 4ms
 - ▶ Compute the access time to a sector

Exercise

- ▶ Consider a disk with:
 - ▶ Rotational speed: 7200 rpm
 - ▶ Disk platters: 5, with 2 surfaces per plate
 - ▶ Number of tracks per plate: 30000
 - ▶ Sectors per plate: 600
 - ▶ Seek time: 1 ms per each 100 tracks
- ▶ If the disk head is in track 0 and the data requested are stored in track 600. Compute:
 - ▶ Capacity of the disk
 - ▶ Rotational latency
 - ▶ Transfer time needed to transfer a sector
 - ▶ Access time for a sector

Exercise (solution)

- ▶ Capacity:
 - ▶ $5 \text{ plates} * 2 \text{ sides/plates} * 30.000 \text{ tracks/side} * 600 \text{ sector/track} * 512 \text{ bytes/sector} = 85,8 \text{ GB}$
- ▶ Rotational latency:
 - ▶ $L_r = \text{Half turn/lap time of a track}$
 - ▶ $7.200 \text{ rotation/minute} \rightarrow 120 \text{ rotation/second}$
 $\rightarrow 0,0083 \text{ seconds/rotation} \rightarrow 4.2 \text{ milliseconds (half rotation)}$
- ▶ Sector transfer time:
 - ▶ $600 \text{ sectors per track and 1 track is read in } 8,3 \text{ milliseconds}$
 - ▶ $8,3 / 600 \rightarrow 0.014 \text{ milliseconds}$
- ▶ Seek time:
 - ▶ $\text{Every } 100 \text{ tracks } 1 \text{ ms, and it has to seek to the track } 600$
 - ▶ $600 / 100 = 6 \text{ milliseconds}$

Exercise

- ▶ Be a hard disk with an average seek time of 4 ms, a rotation speed of 15 000 rpm and 512 byte sectors with 500 sectors per track. We need to read a file consisting of 2500 sectors with a total of 1.22 MB. Estimate the time necessary to read this file in two scenarios:
 - ▶ The file is stored sequentially, i.e. the file occupies the sectors of 5 adjacent tracks.
 - ▶ The sectors of the file are randomly distributed on the disk.

Reliability

- ▶ MTTF: mean time to failure
- ▶ MTTR: mean time to repair
- ▶ Availability is defined as:

$$availability = \frac{MTTF}{MTTF + MTTR}$$

- ▶ What does a reliability of 99% mean?
 - ▶ In 365 consecutive days device works $99 \times 365 / 100 = 361.3$ days
 - ▶ It is out of service 3.65 days
- ▶ Failures in disk drives produce the 20-55% of the failures in the storage systems.

Energy consumption

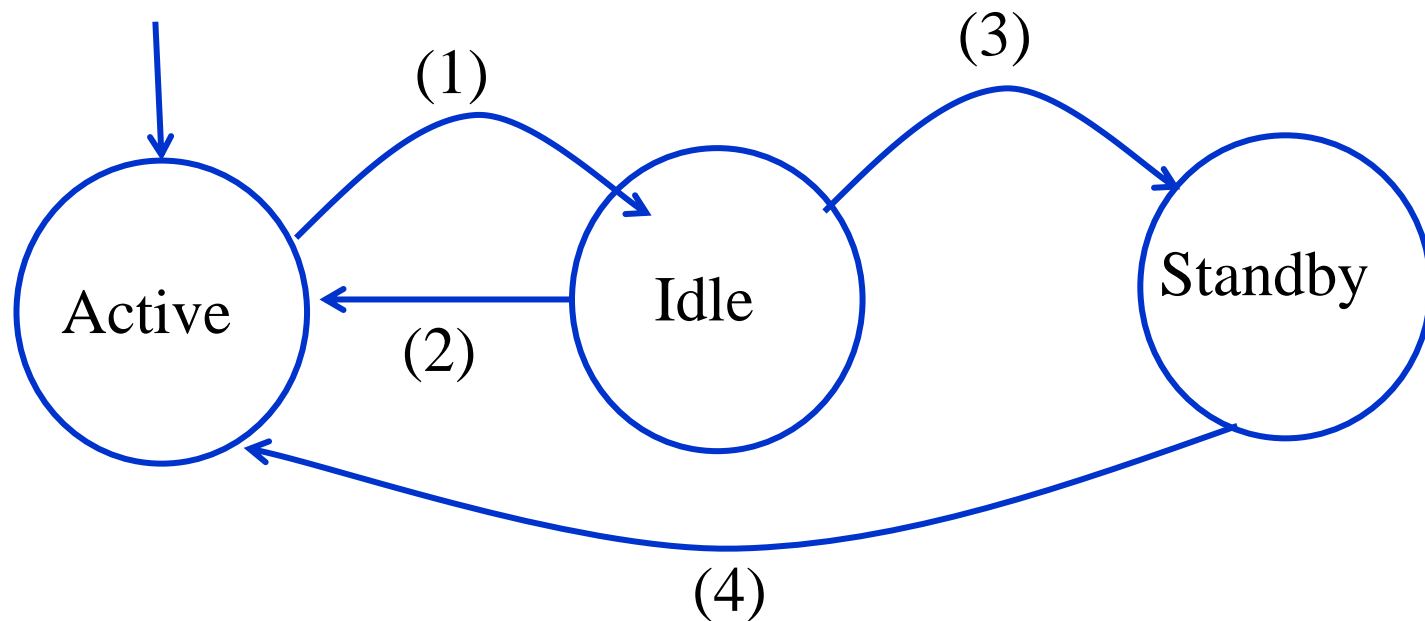
- ▶ The energy consumption in a typical ATA disk drive of 2011 is:
 - ▶ 9 w when is idle
 - ▶ 11 w when is reading or writing
 - ▶ 13 w in a seek operation
- ▶ Power consumed by a disk:

$$Power = N_{platter} \times D_{platter}^{4.6} \times RPM^{2.8}$$

- ▶ Where $N_{platter}$ is the number of disk patters y $D_{platter}$ the diameter for the platters
- ▶ Temperature is often the most important factor which affects the reliability of disk drives
 - ▶ Every 10° increase over 21° decreases the reliability by 50%

Power state transition of disk drives

R/W requests



Power state transitions

- ▶ (1): there is no pending request, the disk drive is transferred to the idle state where the disk platters are still spinning but the electronics may be partially unpowered
- ▶ (2): Disk drive receives a request
- ▶ (3): To conserve energy, the disk drive can be spun down to the standby state where the disk stops spinning, and the head is moved off the disk
- ▶ (4): To perform requests after entering the standby state, the disk drive must be transferred back from the standby state to the active state by spinning up

Energy conservation methods

- ▶ Based on timeout strategies. Once a disk drive is idle for a specific period of time, the disk drive is spun down to save energy
- ▶ Dynamic prediction. Based on the behaviors of application
- ▶ Stochastic mechanisms.
- ▶ Application-aware power management
 - ▶ Applications inform over the access pattern (in the source code or with compiler-driven methods)

Impacts of power state transitions

problems spinning down disks

- ▶ Increased consumption, when the platens have to rotate again.
- ▶ Reduces the reliability of the discs.
Manufacturers usually indicate the number of start/stop cycles a disk can withstand. Above this value the probability of failure increases by 50%.
- ▶ Power saving methods are usually applied to portable devices and are not applied to servers because of the intensive data loads.

Solid State Drive (SSD)

- ▶ Semiconductor-based block storage device that acts as a disk drive
 - ▶ Based on Flash memories
 - ▶ Non-volatile storage
 - ▶ Based on DDR memories
 - ▶ Requires batteries and disk backup for non-volatile storage

HDD

with moving parts



SDD

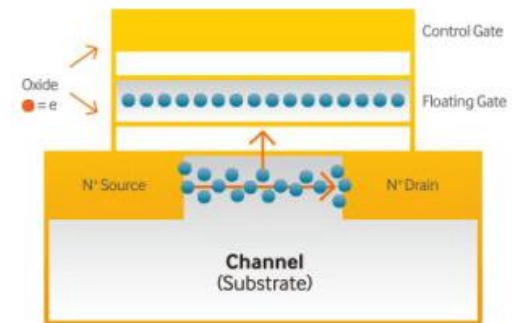
Without moving parts

Flash Memories

- ▶ Non-volatile memories that can be deleted and recorded electrically
- ▶ Types:
 - ▶ Flash NOR
 - ▶ Based on NOR doors
 - ▶ Allows access to 1-byte data
 - ▶ Good for high-speed random access
 - ▶ Used in BIOS memory (boot function)
 - ▶ Faster reading operations
 - ▶ NAND Flash
 - ▶ Based on NAND doors
 - ▶ Reads and writes at high speed in sequential mode in block sizes
 - ▶ Cannot access individual bytes
 - ▶ Higher density and cheaper. Used in SSD
 - ▶ More durable, less expensive, denser, faster write/erase operations

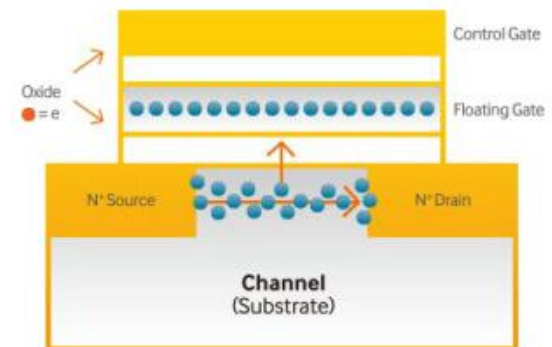
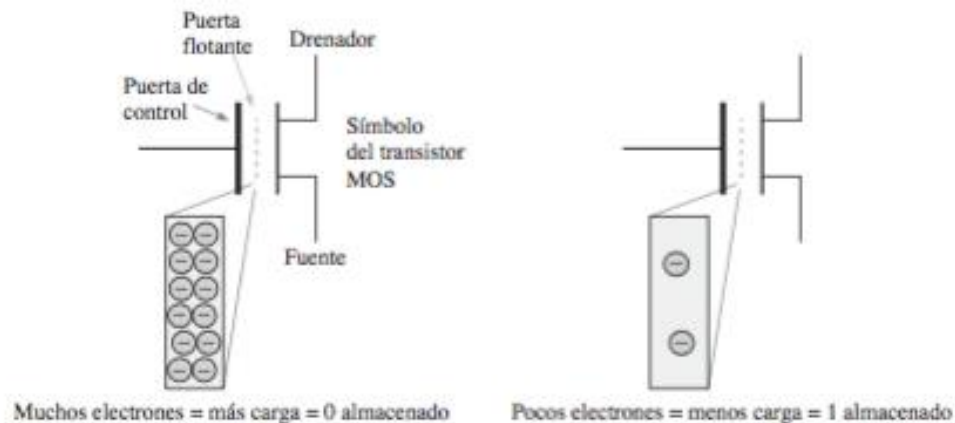
Memory cells

- ▶ Each storage cell consists of a floating gate MOS transistor
- ▶ There are two gates insulated by a layer of oxide
 - ▶ Control gate
 - ▶ Floating gate
- ▶ The electrons flow freely between the two gates
- ▶ The floating door is electrically insulated, trapping the electrons



Memory Cells

- ▶ The data bit is stored as a charge or no charge in the floating gate, depending on whether you want to store a 0 or a 1.

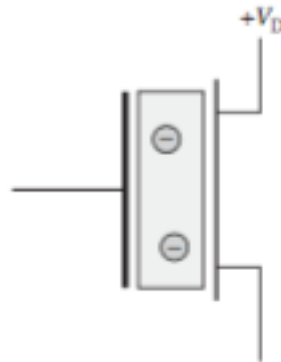
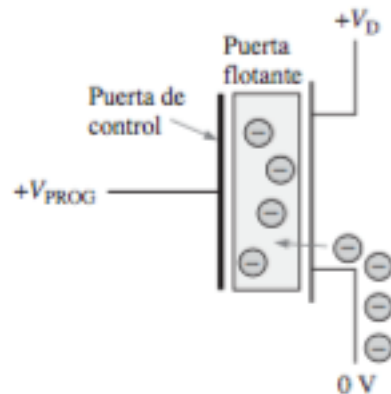


Fundamentos de Sistemas Digitales
Thomas L. Floyd

Basic operations of a flash memory cell

- ▶ Programming
 - ▶ Initially all cells are in state 1, because the load is removed
- ▶ Read operation
- ▶ Delete operation

Programming a Flash memory cell

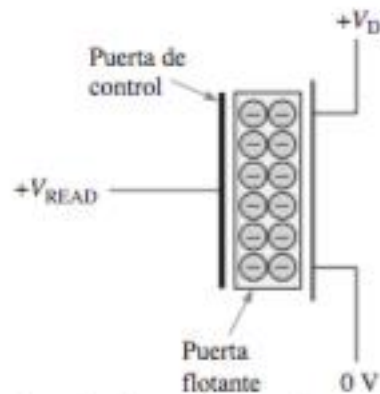


Fundamentos de Sistemas Digitales
Thomas L. Floyd

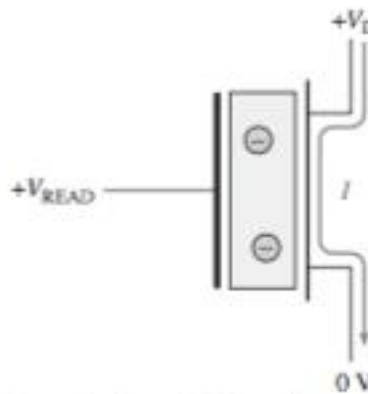
- ▶ To store a 0, a sufficiently positive voltage is applied to the control gate with respect to the source, to add charge to the floating gate during programming (attracts electrons)
- ▶ In the writing process, electrons are added to those doors that should store a 0 and not added to those that should store a 1.
- ▶ Only "0" is written
- ▶ To store a 1, no charge is added, leaving the cell in the deleted state

Reading a Flash memory cell

- Positive voltage is applied to the control gate



Cuando se lee un 0, el transistor permanece desactivado, porque la carga de la puerta flotante impide a la tensión de lectura exceder el umbral de activación.

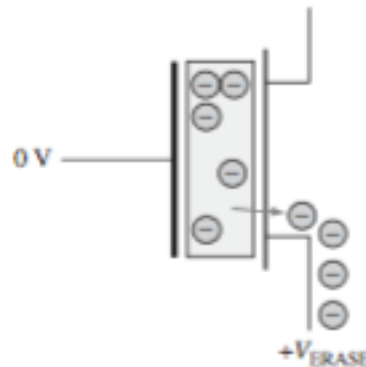


Cuando se lee un 1, el transistor se activa, porque la ausencia de carga en la puerta flotante permite que la tensión de lectura exceda el umbral de activación.

Fundamentos de Sistemas Digitales
Thomas L. Floyd

Delete

- ▶ During the delete operation, the load is removed from all memory cells. A voltage is applied in the opposite direction to remove the electrons



Para borrar una célula, se aplica a la fuente una tensión suficientemente positiva con respecto a la puerta de control, con el fin de extraer la carga de la puerta flotante durante la operación de borrado.

Fundamentos de Sistemas Digitales
Thomas L. Floyd

NAND Flash memory types

- ▶ Single-level cell flash (SLC)
 - ▶ Store 1 bit per cell
 - ▶ 50000 - 100000 writings per cell
 - ▶ Used primarily in military and industrial applications
- ▶ Multi-level cell flash (MLC)
 - ▶ Store several bits per cell, depending on the number of electrons stored in the cell
 - ▶ They offer more capacity but less duration (they wear more)
 - ▶ < 10000 writings per cell
 - ▶ Used in consumer electronics
 - ▶ Lower cost
 - ▶ Half the performance of SLCs

Wear leveling

- ▶ A NAND flash memory can only write a certain number of times in each block (or cell)
- ▶ When the limit is exceeded, the cell wears out (its oxide layer) and no longer stores electrons properly
- ▶ Wear Leveling - a process used by an SSD controller to maximize the life of the flash memory
- ▶ This technique levels out the wear and tear on all blocks by distributing the data writing across all blocks
 - ▶ When a block is to be modified, it is written in a new one

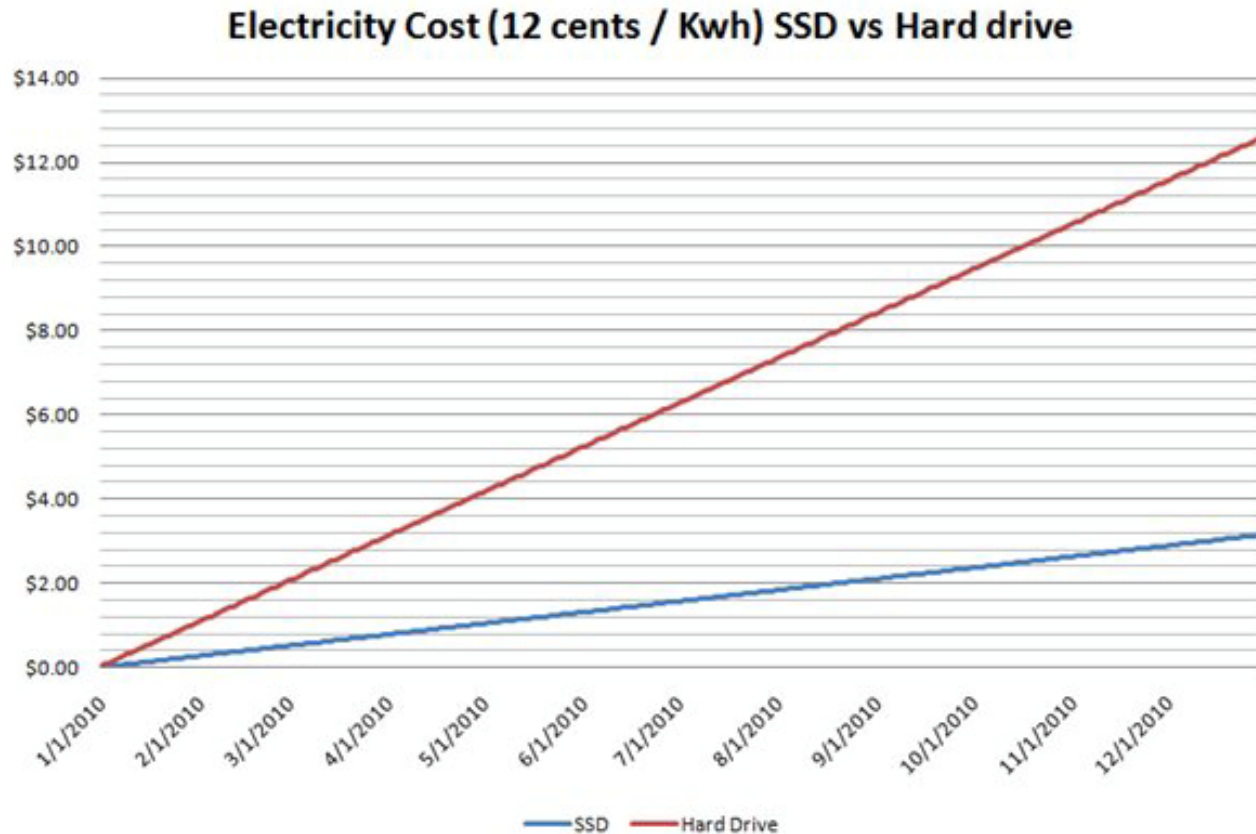
Structure of a Nand Flash

- ▶ NAND flashes are divided into 128 KB blocks that are subdivided into 2KB pages
- ▶ The deletion is done from a complete block
- ▶ Programming and reading a page

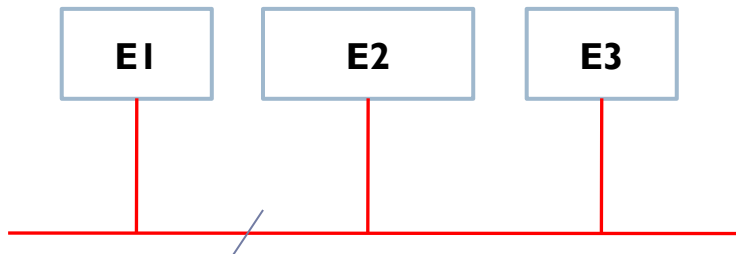
SDD vs HDD

	SDD	HDD
Access time	0.1 ms	5-8 ms
I/O operations/sec	6000 io/s	400 io/s
consumption	2-5 watos	6-15 watos

Energy consumption

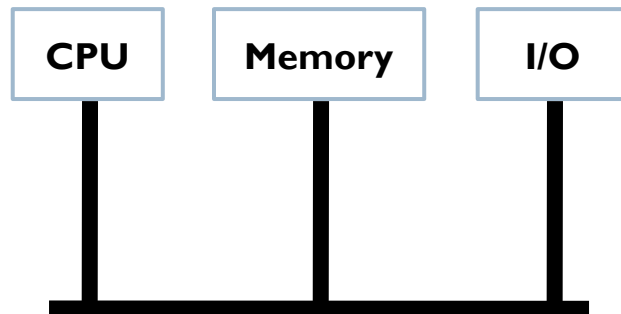


Bus



- ▶ A bus is a communication **path** between two or more devices.
- ▶ It **consists** of **several bit** transmission **lines**.
- ▶ **Shared medium**, univocal.
- ▶ Allows to transmit several bits between two elements connected to it

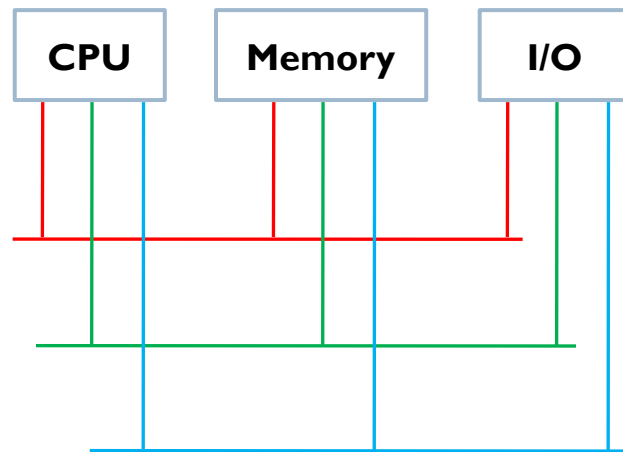
System bus



► **System bus**

- Connects the main components of the computer
- It represents the union of three buses:
 - Control
 - Addresses
 - Data

Buses



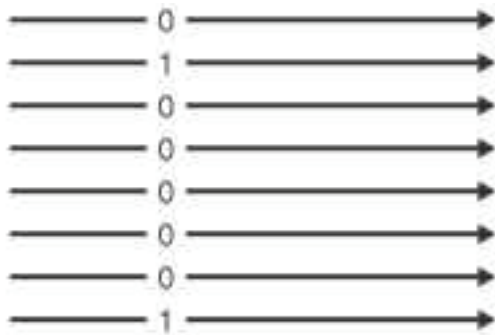
- ▶ **Data** bus
 - ▶ Transmits data
 - ▶ Its width and speed have a great influence on the performance
- ▶ **Address** bus
 - ▶ Memory addresses and I/O devices
 - ▶ Its width determines the maximum memory capacity
- ▶ **Control** bus
 - ▶ Control and timing signals

Characteristics of a bus

- ▶ Bus width
- ▶ Frequency
- ▶ Transfer speed
- ▶ Bandwidth

Characteristics of a bus

- **Bus width:** determines the number of bits that can be transmitted simultaneously



Parallel bus



serial bus

Characteristics of a bus

- ▶ **Frequency**: clock frequency with which it can operate
- ▶ **Transfer rate**: number of bytes per clock cycle
- ▶ **Bandwidth** (transfer rate): transmitted bytes per second
 - ▶ Transfer rate X frequency

Exercise

- ▶ Calculate the bandwidth in MBps of a 32-bit bus and a frequency of 66 MHz

Exercise (solution)

- ▶ Calculate the bandwidth in MBps of a 32-bit bus and a frequency of 66 MHz

$$\text{Bandwidth} = \frac{32 \text{ bits} \times 66 \text{ MHz}}{8 \text{ bits por byte}} = \frac{32 \times 66 \cdot 10^6}{8} = 264 \text{ MBps}$$

Arbitration method (bus protocol)

- ▶ Determines which of the elements connected to the bus can access the bus
- ▶ **Centralized** scheme: a bus controller grants the use of the bus
 - ▶ When an element wants to access the bus, it requests permission from the controller through the control lines (BUSRQ)
 - ▶ When the bus is free the controller grants the use (BUSACK)
- ▶ **Distributed** scheme: each element connected to the bus includes an access control logic that allows the joint use of the bus (access protocol)

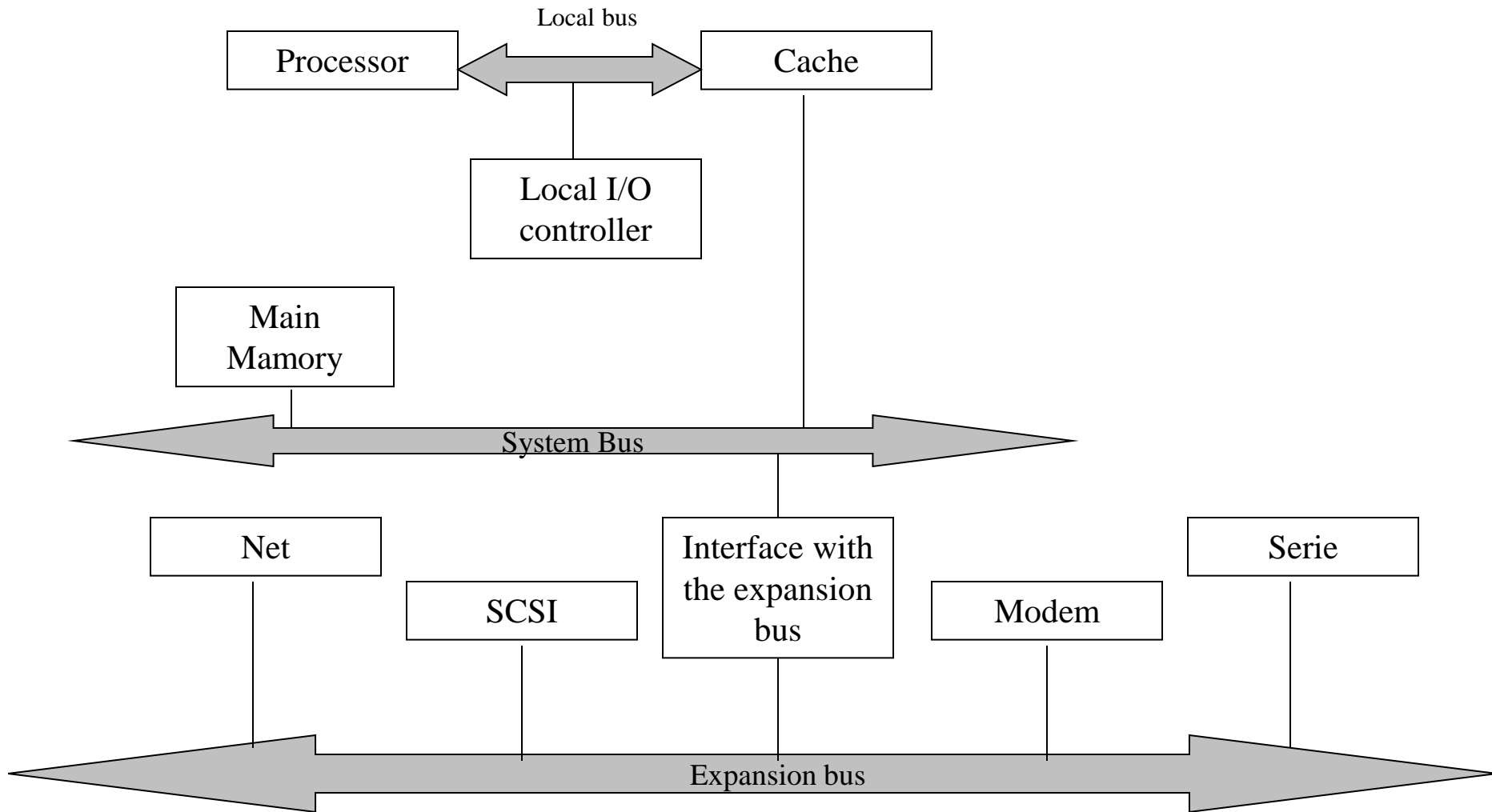
Synchronous and asynchronous buses

- ▶ A **synchronous** bus is governed by a clock signal and a communication protocol set to the operation of the clock
 - ▶ Fast
 - ▶ All devices connected to it must operate at the same clock frequency
- ▶ An **asynchronous** bus does not use a clock, the communication is done by sending orders through the control lines of the bus

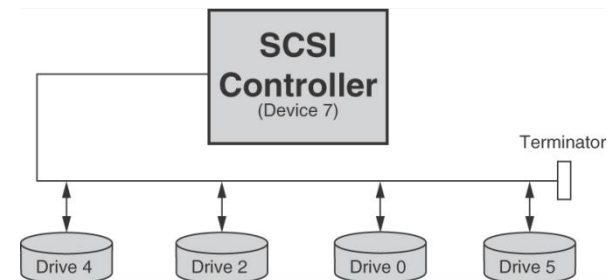
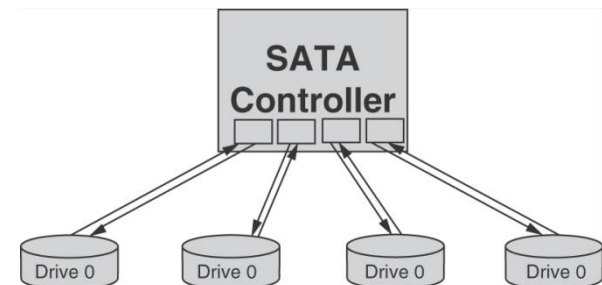
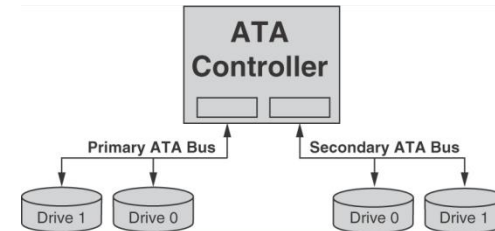
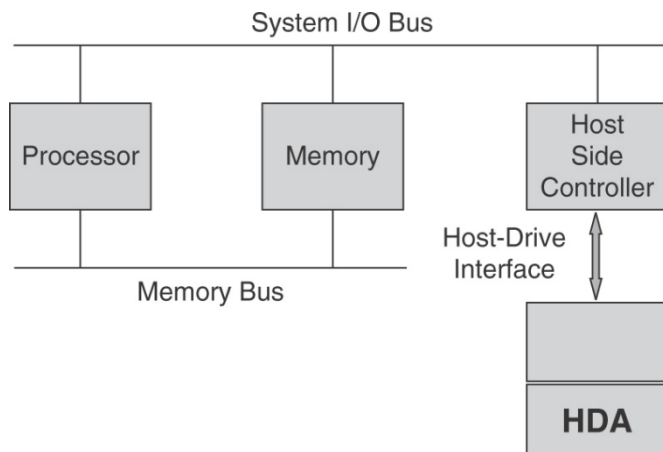
Bus hierarchies

- ▶ The more devices connected to the bus, the longer the propagation delay.
- ▶ As the number of transfer requests increases, a bottleneck can occur.
- ▶ Solutions:
 - ▶ Increase data transmission speed with wider buses.
 - ▶ Use more data buses, organized hierarchically.

Bus hierarchies

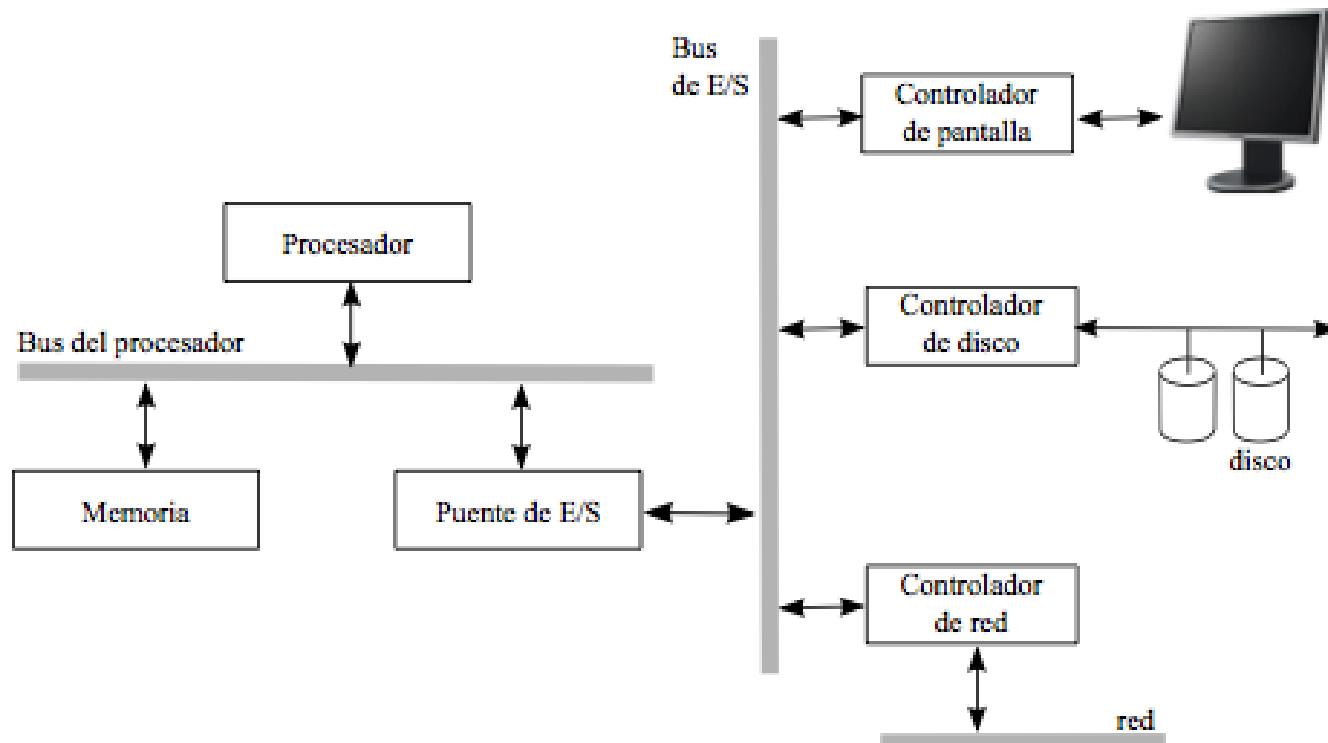


Disks Controllers



Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

Bus diagram in a typical computer system



Curiosity:

USB family



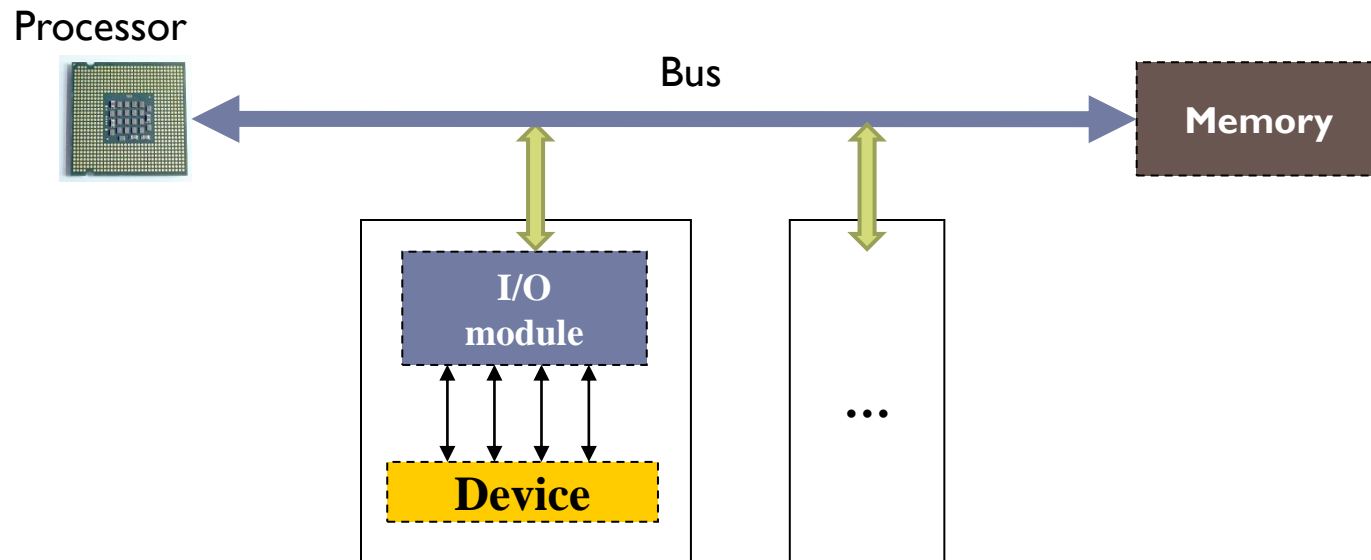
	Transfer (per sec.)	Introduction
USB 3.0	600 MB/s	2010
USB 2.0	60 MB/s	2000
USB 1.0	1.5 MB/s and 187 KB/s	1996

	Song / Pic	256 Flash	USB Flash	SD-Movie	USB Flash	HD-Movie
	4 MB	256 MB	1 GB	6 GB	16 GB	25 GB
USB 1.0	5.3 sec	5.7 min	22 min	2.2 hr	5.9 hr	9.3 hr
USB 2.0	0.1 sec	8.5 sec	33 sec	3.3 min	8.9 min	13.9 min
USB 3.0	0.01 sec	0.8 sec	3.3 sec	20 sec	53.3 sec	70 sec

<http://www.unp.co.in/f140/comparison-of-usb-3-0-port-with-usb-2-0-and-usb-1-0-a-70063/>

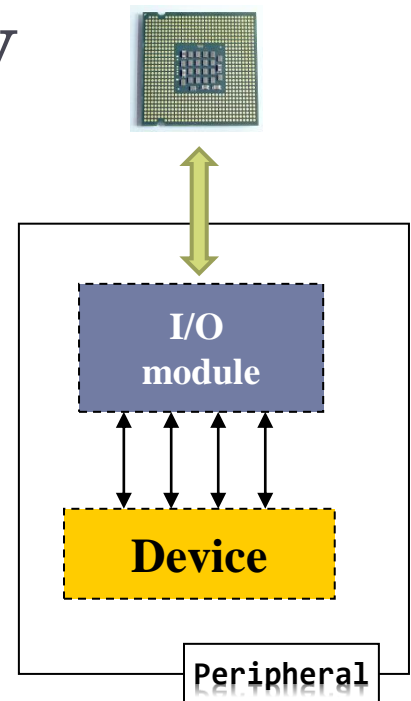
I/O module

- ▶ **I/O modules** perform the connection among the peripheral devices and the processor (or the memory)



Why I/O modules are necessary

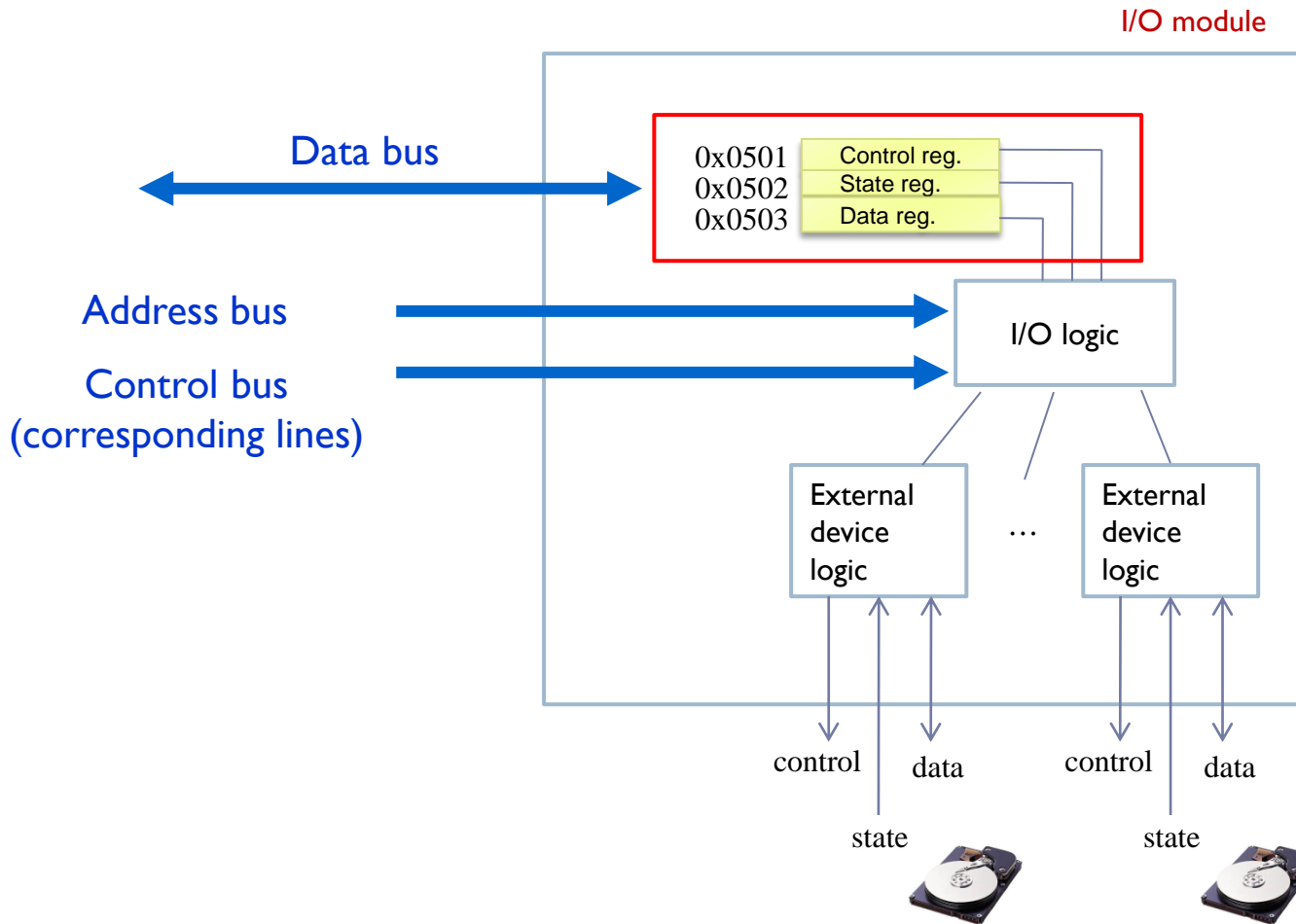
- ▶ They are necessary due to:
 - ▶ Wide variety of peripherals.
 - ▶ Peripherals are 'peculiar'
 - ▶ The data transfer rate of peripherals is much slower than that of the memory or the processor.
 - ▶ Peripherals are 'very slow'.
 - ▶ Formats and word sizes of peripherals different from those of the computer to which they are connected.



Why I/O modules are necessary

- ▶ Control and timing
- ▶ Processor communication
- ▶ Device communication
- ▶ Data buffering
- ▶ Error detection

Simplified I/O module model



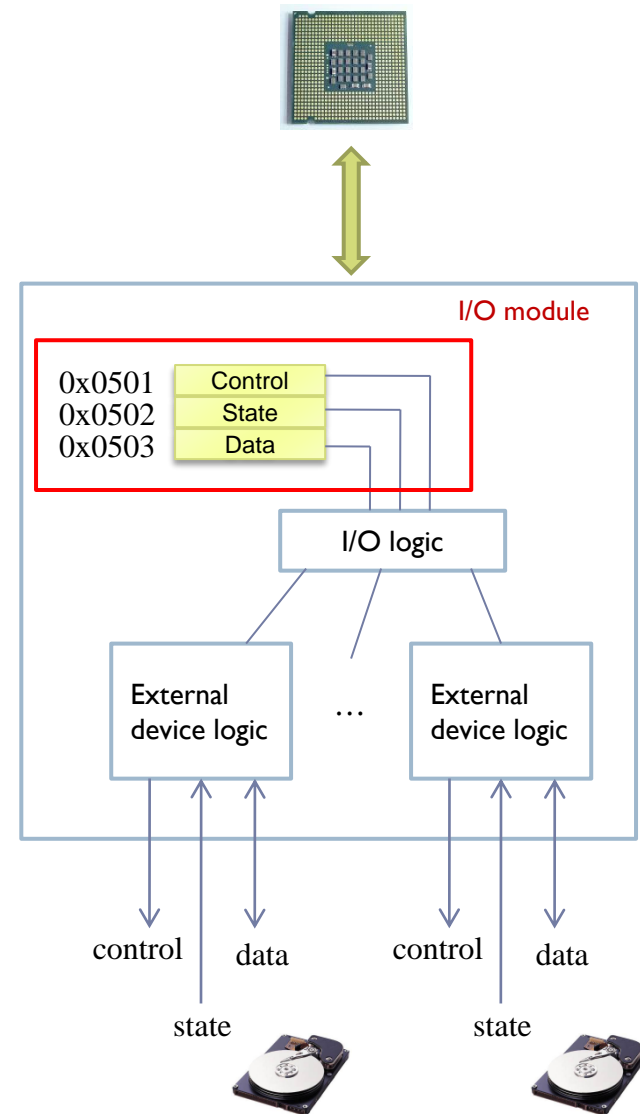
Simplified I/O module model

- ▶ Interaction between processor and I/O module through 3 registers:

- ▶ The **control** register
 - ▶ Commands for the peripheral

- ▶ The **state** register
 - ▶ Status of the last command

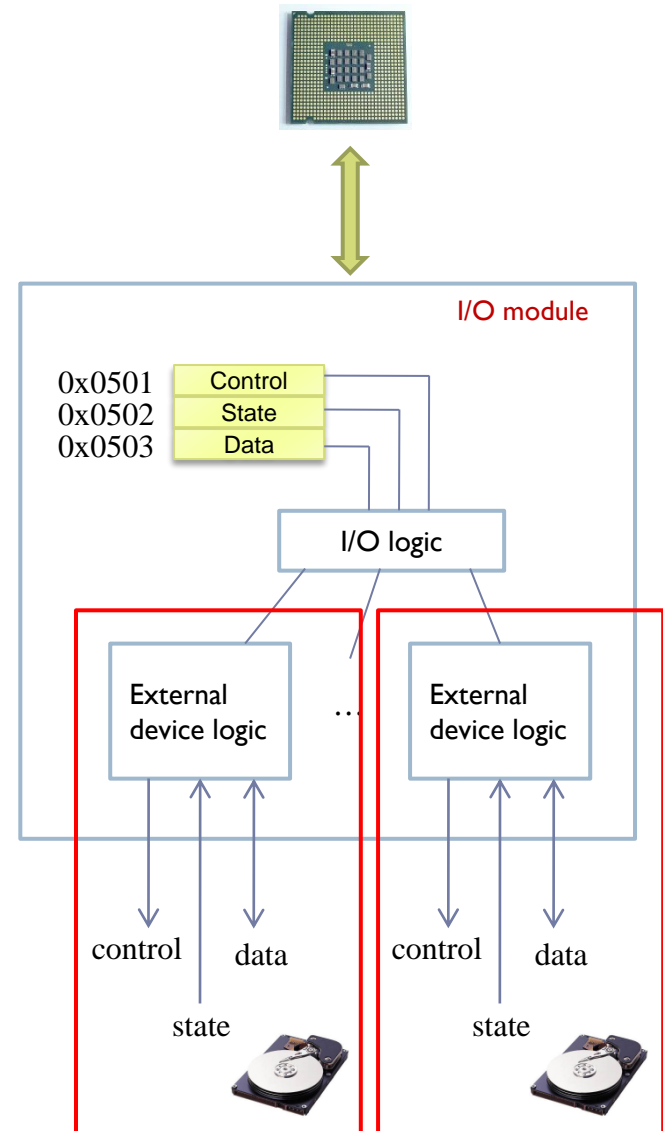
- ▶ The **data** register
 - ▶ Data exchanged Processor/Perif.



Simplified I/O module model

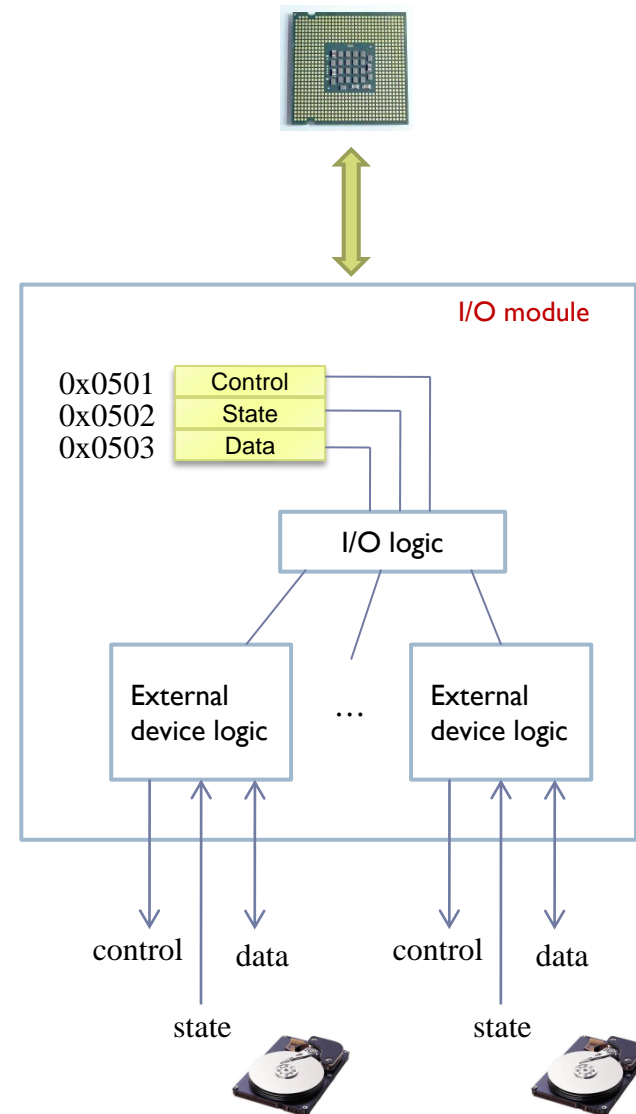
- ▶ Interaction between peripheral and I/O module:

- ▶ **Data signals (lines):**
for transferring information
- ▶ **State signals (lines):**
information about the device
 - ▶ Examples:
 - New data available
 - Peripheral on/off
 - Peripheral busy
 - Peripheral up and running
 - Error in last command
 - ...
- ▶ **Control signals (lines):**
to control the peripheral/device
 - ▶ Examples:
 - Power on/off
 - Skip page in a printer
 - Seek in a hard disk
 - ...



I/O module: functions

- ▶ **Attending to the CPU:**
 - ▶ Order decoding
 - ▶ Status information
 - ▶ Control and timing
 - ▶ E.g.: data to M.M.
- ▶ **Control peripheral(s):**
 - ▶ Communication with peripherals
 - ▶ Error detection
 - ▶ Temporary data storage
 - ▶ peripheral->processor

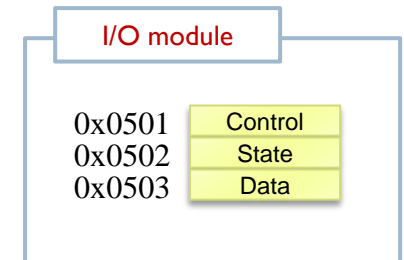


I/O module: types

- ▶ Types of modules by complexity:

- ▶ **Channel I/O** or **I/O processor**:
handles most of the processing details.

- ▶ **I/O controller** or **device driver**:
simpler module, which requires the processor to have detailed control of the device.



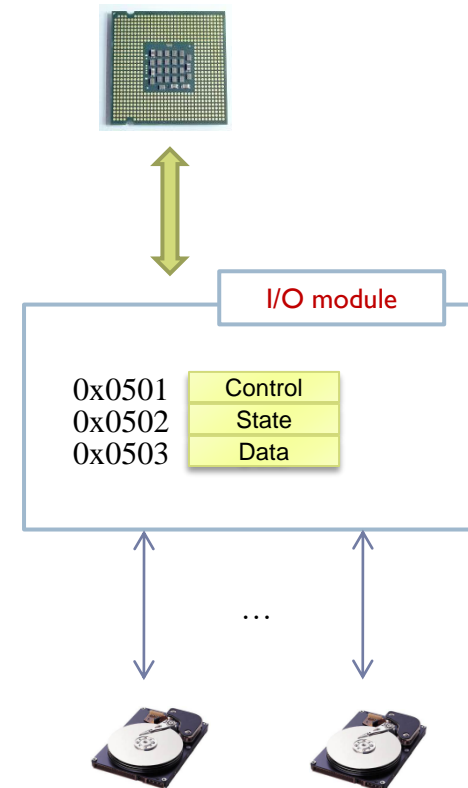
I/O module: characteristics

- ▶ Main characteristics:

- ▶ **Transfer unit**

- ▶ **Addressing**

- ▶ **I/O techniques**



Characteristics (1 / 3):

Transfer unit

▶ **Block devices:**

- ▶ Unit: **block** of bytes
- ▶ Access: **sequential** or **random**
- ▶ Operations: `read`, `write`, `seek`, ...
- ▶ Examples: “tapes” and disks



▶ **Character devices:**

- ▶ Unit: **chars** (ASCII, Unicode, etc.)
- ▶ Access: **sequential** to characters
- ▶ Operations : `get`, `put`,
- ▶ Example: terminals, printers, etc.



Characteristics (2/3):

I/O addressing

- ▶ **Memory-mapped I/O (MMIO)**

- ▶ I/O registers are mapped in memory using a set of memory addresses for these registers. Same machine instructions are used for memory and I/O.

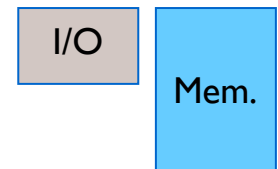
- ▶ E.g.: `sw $a0 label_disk`



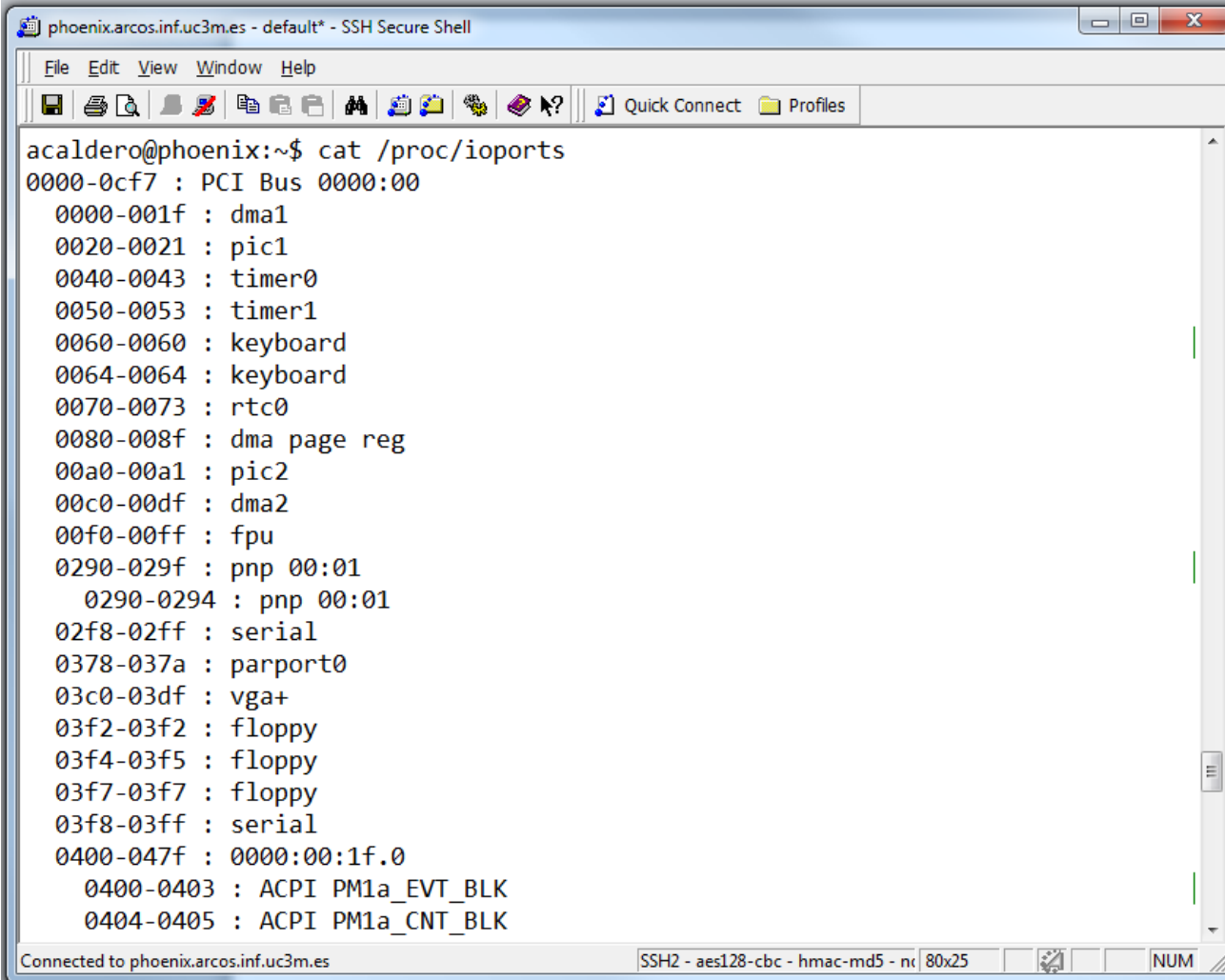
- ▶ **Port-mapped I/O (PMIO):**

- ▶ The address space for I/O is isolated from the address space used for the memory. It uses special machine instructions to access the I/O registers to access the I/O registers (ports).

- ▶ E.g.: `out $a0 0x105A`



Addressing Linux

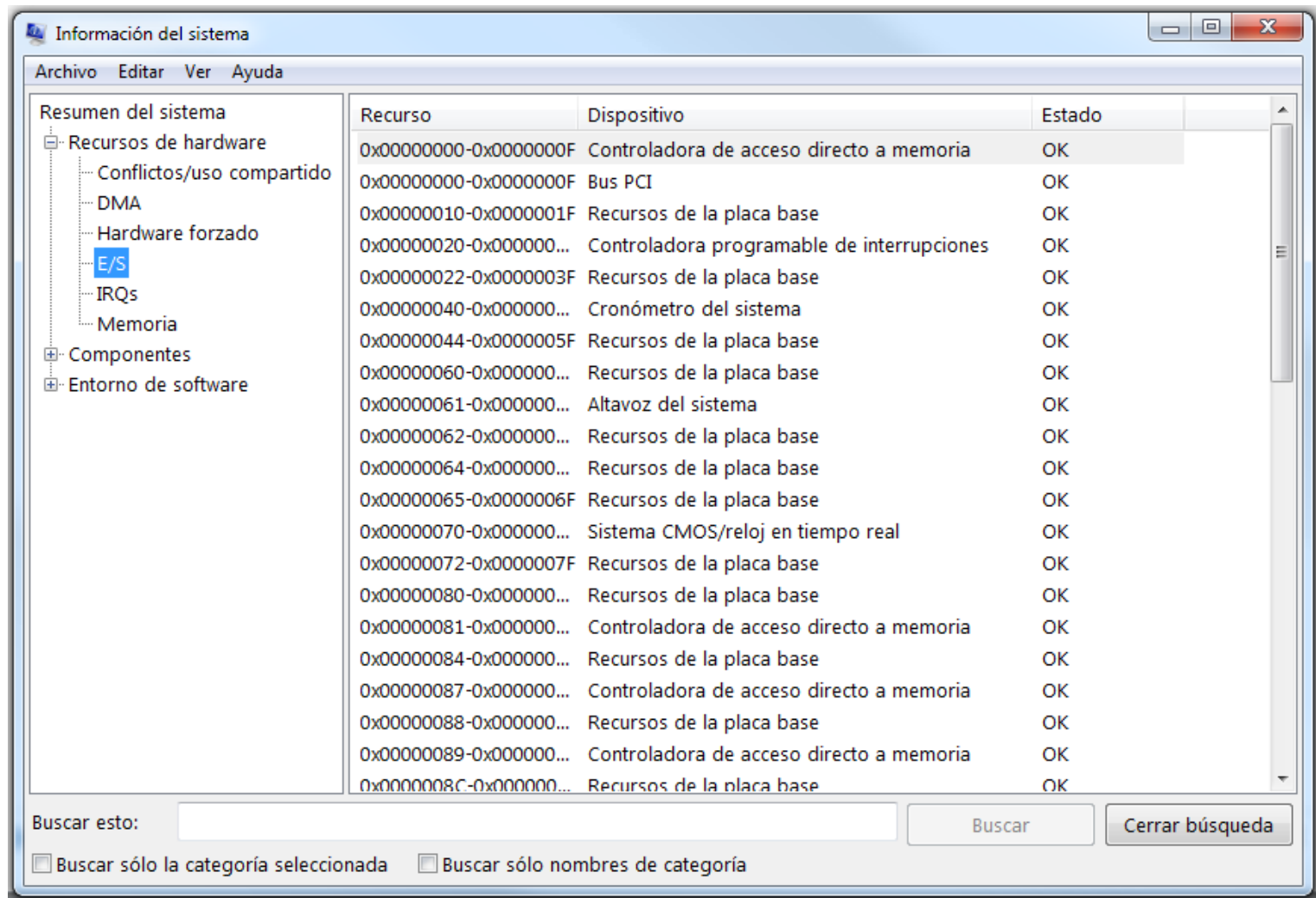


The screenshot shows an SSH terminal window titled "phoenix.arcos.inf.uc3m.es - default* - SSH Secure Shell". The terminal displays the command `cat /proc/ioports` and its output, which lists various I/O ports and their associated hardware components. The output is as follows:

```
acaldero@phoenix:~$ cat /proc/ioports
0000-0cf7 : PCI Bus 0000:00
  0000-001f : dma1
  0020-0021 : pic1
  0040-0043 : timer0
  0050-0053 : timer1
  0060-0060 : keyboard
  0064-0064 : keyboard
  0070-0073 : rtc0
  0080-008f : dma page reg
  00a0-00a1 : pic2
  00c0-00df : dma2
  00f0-00ff : fpu
  0290-029f : pnp 00:01
    0290-0294 : pnp 00:01
  02f8-02ff : serial
  0378-037a : parport0
  03c0-03df : vga+
  03f2-03f2 : floppy
  03f4-03f5 : floppy
  03f7-03f7 : floppy
  03f8-03ff : serial
  0400-047f : 0000:00:1f.0
    0400-0403 : ACPI PM1a_EVT_BLK
    0404-0405 : ACPI PM1a_CNT_BLK
```

The terminal window also shows a status bar at the bottom indicating the connection details: "Connected to phoenix.arcos.inf.uc3m.es" and "SSH2 - aes128-cbc - hmac-md5 - n".

Addressing Windows



Characteristics (3/3): **I/O techniques**

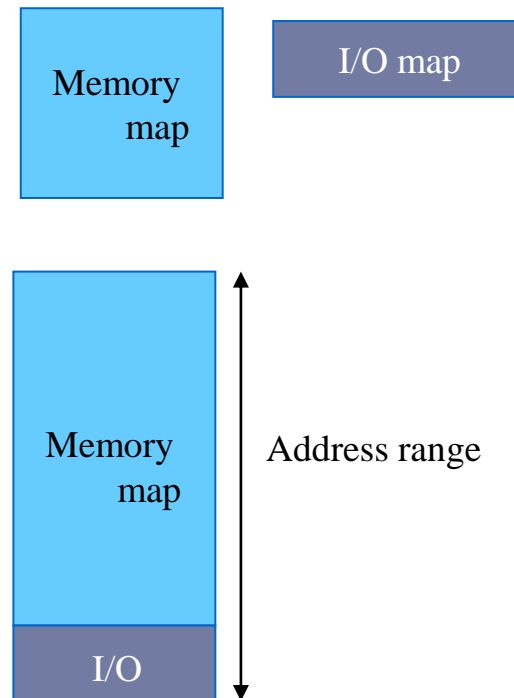
- ▶ **I/O techniques**: Processor-I/O_module interaction
 - ▶ **Programmed I/O**
 - ▶ **Interrupt I/O**
 - ▶ **DMA(Direct Memory Access) I/O**

Programmed I/O

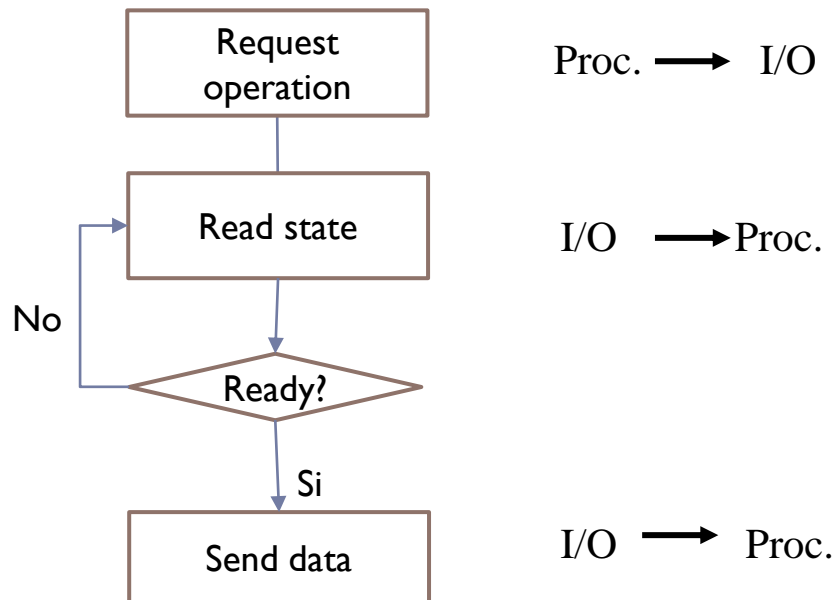
- ▶ The transfers between the processor (memory) and the I/O module is controlled by the processor that executes **I/O machine instructions**
- ▶ I/O instructions:
 - ▶ Special instructions (similar to lw and sw)
 - ▶ Privileged instructions
- ▶ Example of hypothetical I/O instructions:
 - ▶ **IN Reg, address**
 - ▶ Load in the processor register “Reg” the value stored in the I/O register identified by a given address
 - ▶ **OUT Reg, address**
 - ▶ To write an item in an I/O register from the I/O module

I/O map

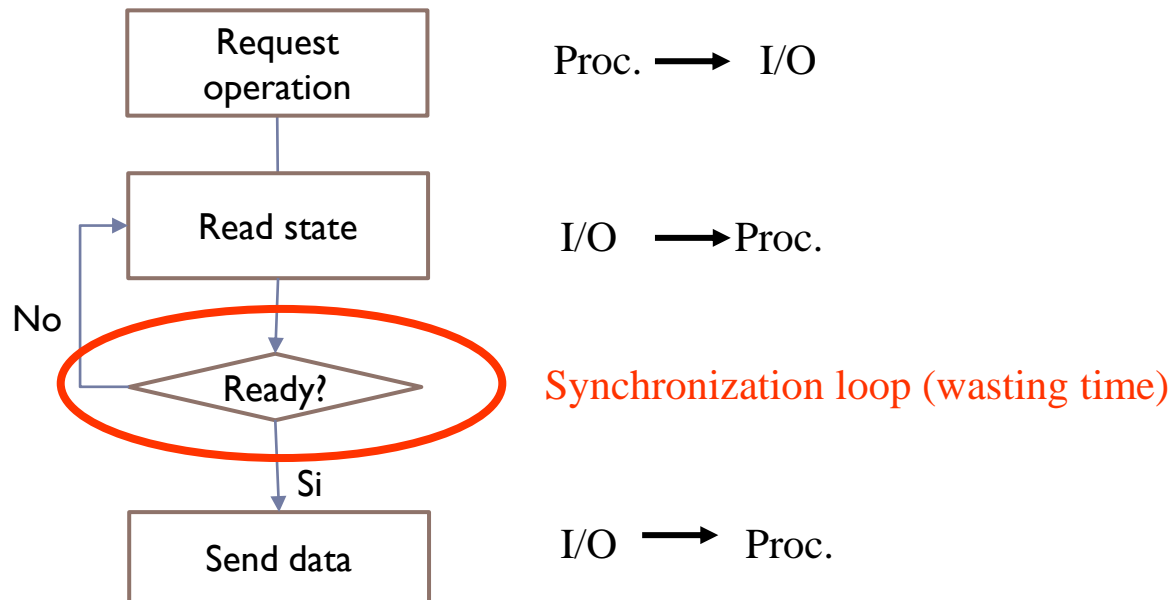
- ▶ I/O map: space address for I/O
 - ▶ With p bits, 2^p possible addresses
- ▶ Types:
 - ▶ **Isolate I/O map**
 - ▶ Port-mapped I/O
 - ▶ Includes special I/O instructions (IN, OUT)
 - ▶ **Shared I/O map**
 - ▶ Memory-mapped I/O
 - ▶ Same machine instructions for I/O and for memory (LOAD, STORE)



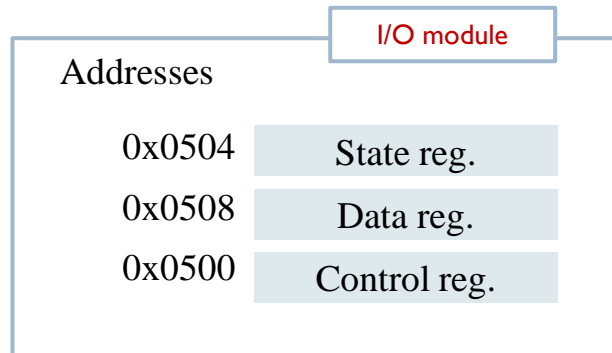
Interaction via programmed I/O



Interaction via programmed I/O

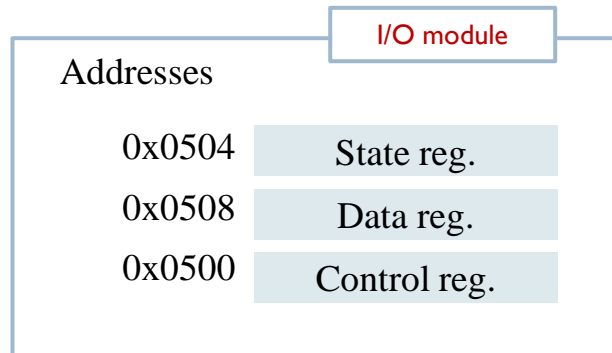


Example



- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

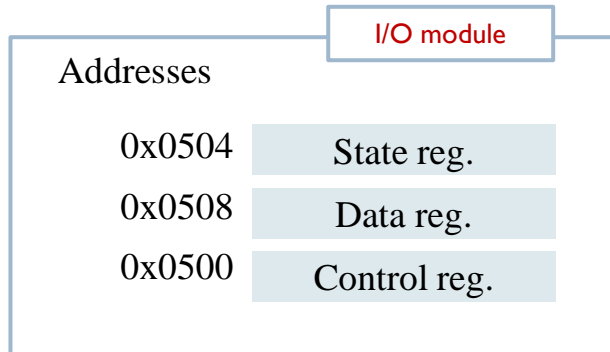
Example



- ▶ Instructions to write a 1 to register 0x0508 (data)?

- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

Example

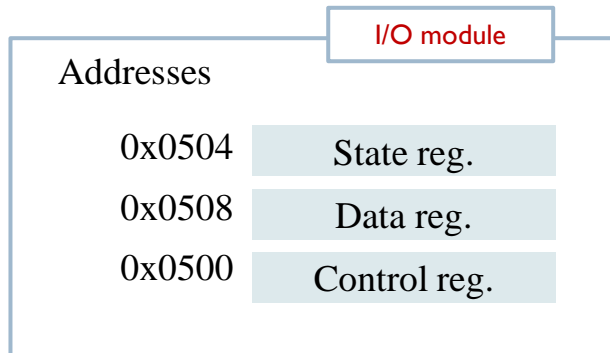


- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

- ▶ Instructions to write a 1 to register 0x0508 (data)?

```
li $t0, 1
sw $t0, 0x0508
```

Example

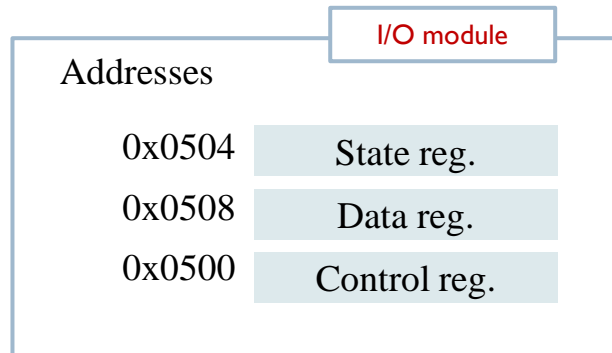


- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

- ▶ How to read a data (word)?



Example



- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

▶ How to read a data (word)?

1. Send the command

```
li $t0, 0  
sw $t0, 0x0500
```

2. Read state

```
bucle: lw $t0, 0x0504
```

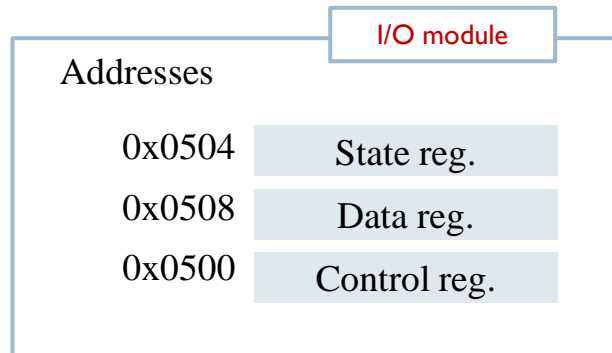
3. Check state

```
beqz $t0, bucle
```

4. Read the data (word)

```
lw $t0, 0x0508
```

Example



- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

▶ How to write a data (word)?

1. Send the data (word)

```
li $t0, 123  
sw $t0, 0x0508
```

2. Send the command

```
li $t0, 1  
sw $t0, 0x0500
```

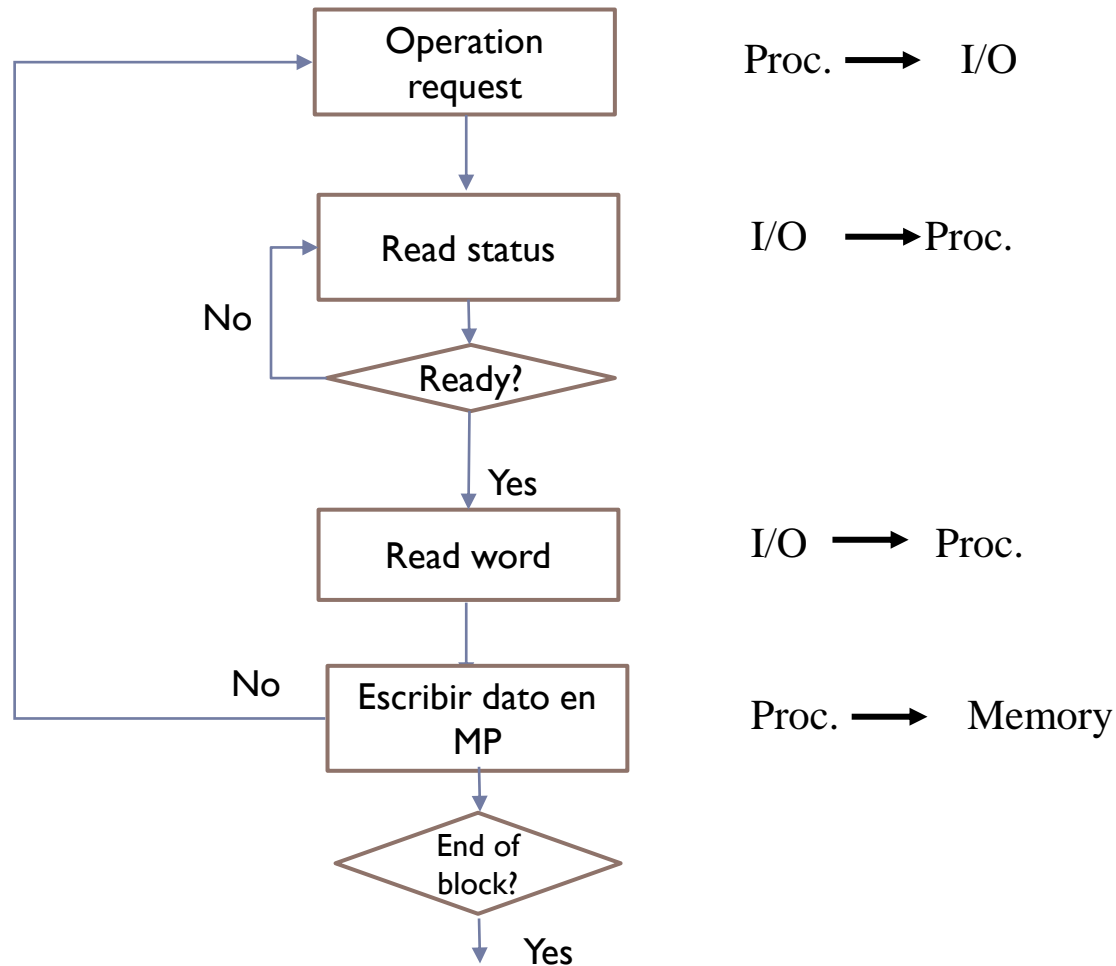
3. Read state

```
bucle: lw $t0, 0x0504
```

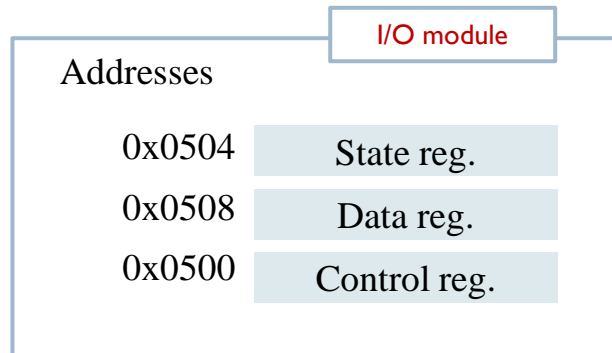
4. Check state

```
beqz $t0, bucle
```

Reading a data block



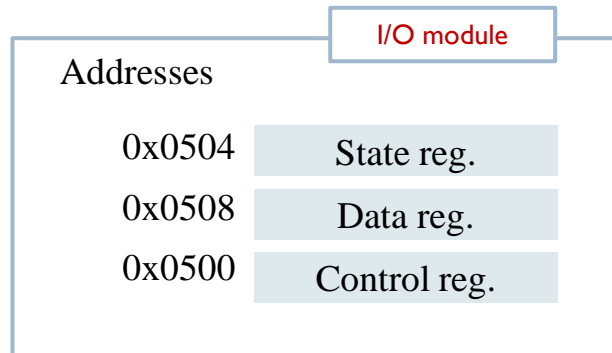
Exercise



Code an assembler program that reads 100 integers using the described I/O module, and stores them in the main memory at address given by the 'dataI' label.

- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

Exercise (solution)



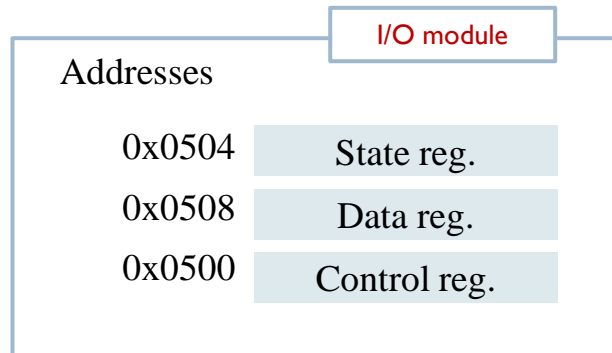
- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

```
.data
    data1: .space 400

.text
.globl main
main:    li $t3 0
        loop1: li $t0 0
            sw $t0 0x500

        loop2: lw $t1 0x504
            beqz $t1 loop2
            lw $t2 0x508
            sw $t2 data1($t3)
            add $t3 $t3 4
            bne $t3 400 loop1
```

Exercise (solution)



- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

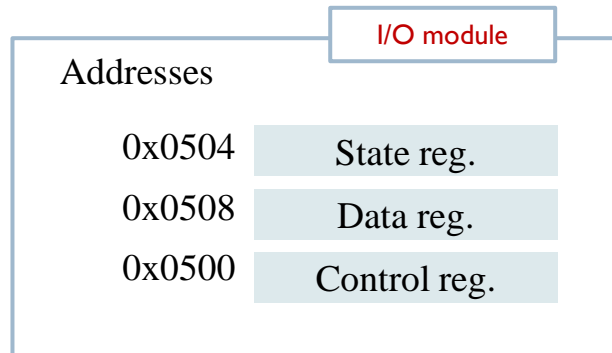
```
.data
    data1: .space 400

.text
.globl main
main:    li $t3 0
        loop1: li $t0 0
            sw $t0 0x500

        loop2: lw $t1 0x504
            beqz $t1 loop2
            lw $t2 0x508
            sw $t2 data1($t3)
            add $t3 $t3 4
            bne $t3 400 loop1
```

} Synchronization loop

Exercise (solution)



- ▶ Control information:
 - ▶ 0: read
 - ▶ 1: write
- ▶ State information:
 - ▶ 0: device not ready
 - ▶ 1: device (data) ready
- ▶ Memory-mapped I/O:
 - ▶ lw and sw MIPS instructions

```
.data
    data1: .space 400

.text
.globl main
main:    li $t3 0
        loop1: li $t0 0
                sw $t0 0x500

        loop2: lw $t1 0x504
                beqz $t1 loop2
                lw $t2 0x508
                sw $t2 data1($t3)
                add $t3 $t3 4
                bne $t3 400 loop1
```

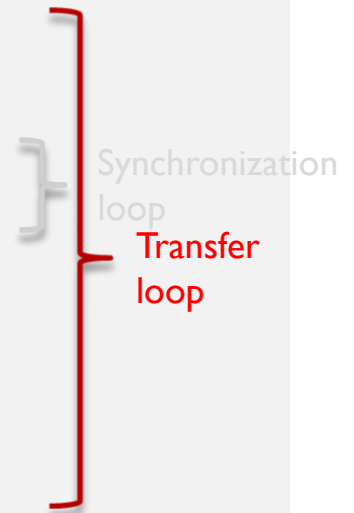
} Synchronization loop

} Transfer loop

Exercise

- ▶ Be a computer with the capacity to execute 200 million instructions per second (200 MIPS).
- ▶ The I/O module described above is connected with an average read timeout of 5 ms.
- ▶ Calculate how many instructions are executed in the synchronization loop and in the transfer loop for the program shown.

```
.data
    data1: .space 400
.text
.globl main
main:    li $t3 0
        loop1: li $t0 0
            sw $t0 0x500
        loop2: lw $t1 0x504
            beqz $t1 loop2
            lw $t2 0x508
            sw $t2 data1($t3)
            add $t3 $t3 4
            bne $t3 400 loop1
```



Exercise (solution)

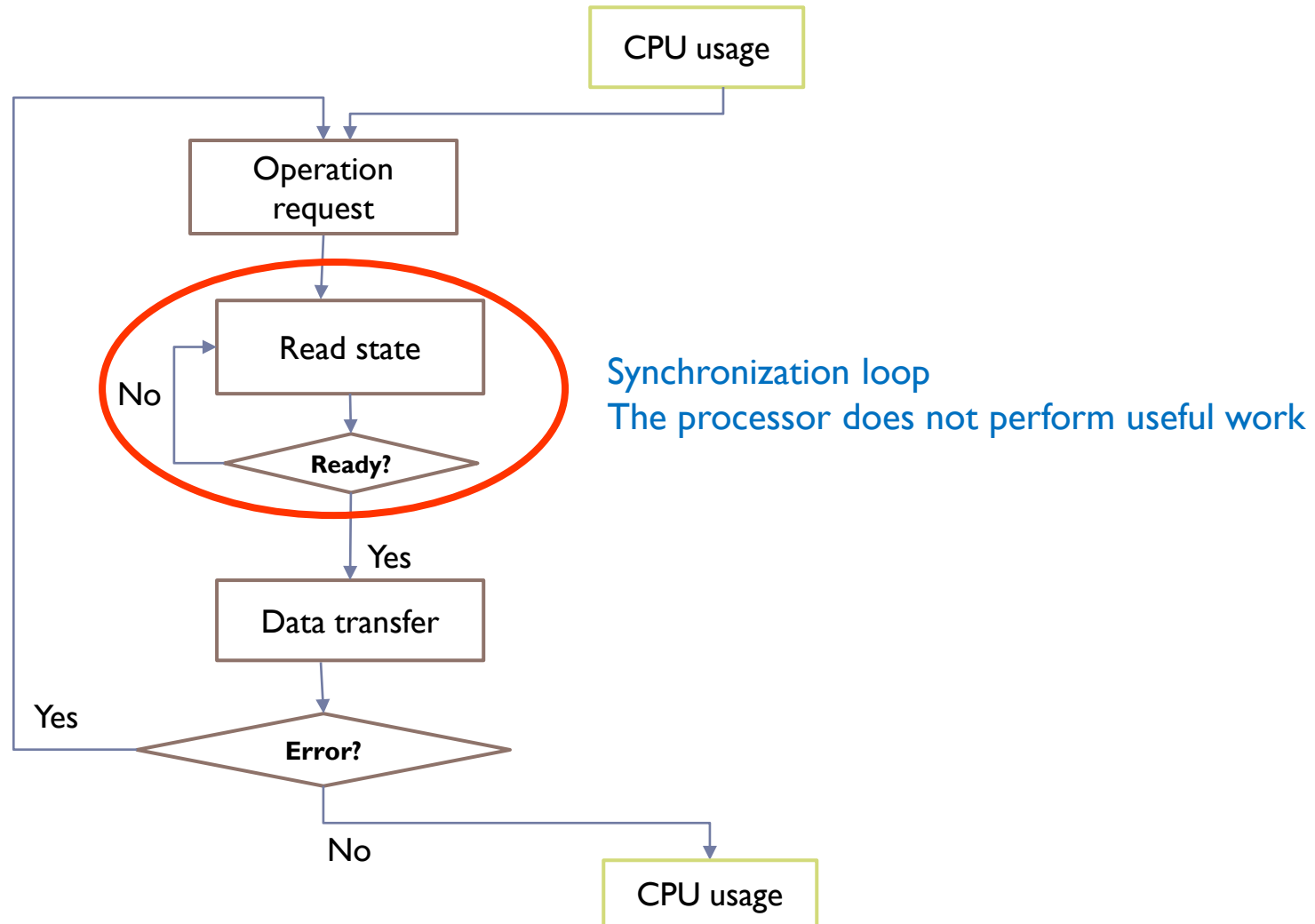
- ▶ Bucle de sincronización:
 - ▶ In average 5 ms
 - ▶ 200 MIPS are executed
 - ▶ $I_{bs} = 200 * 10^6 * 5 * 10^{-3} = 10^6$
- ▶ Bucle de transferencia:
 - ▶ $I = (li \$t3\ 0) + 6 * 100 + 10^6 (I_{bs})$
- ▶ 1,000,601 instructions are executed, and 1,000,000 are instructions executed in the synchronization loop (el 99,9%)
 - ▶ It is a waste of processor cycles
 - ▶ The CPU does not perform useful work

```
.data
    data1: .space 400
.text
.globl main
main:    li $t3 0
        loop1: li $t0 0
            sw $t0 0x500
        loop2: lw $t1 0x504
            beqz $t1 loop2
            lw $t2 0x508
            sw $t2 data1($t3)
            add $t3 $t3 4
            bne $t3 400 loop1
```

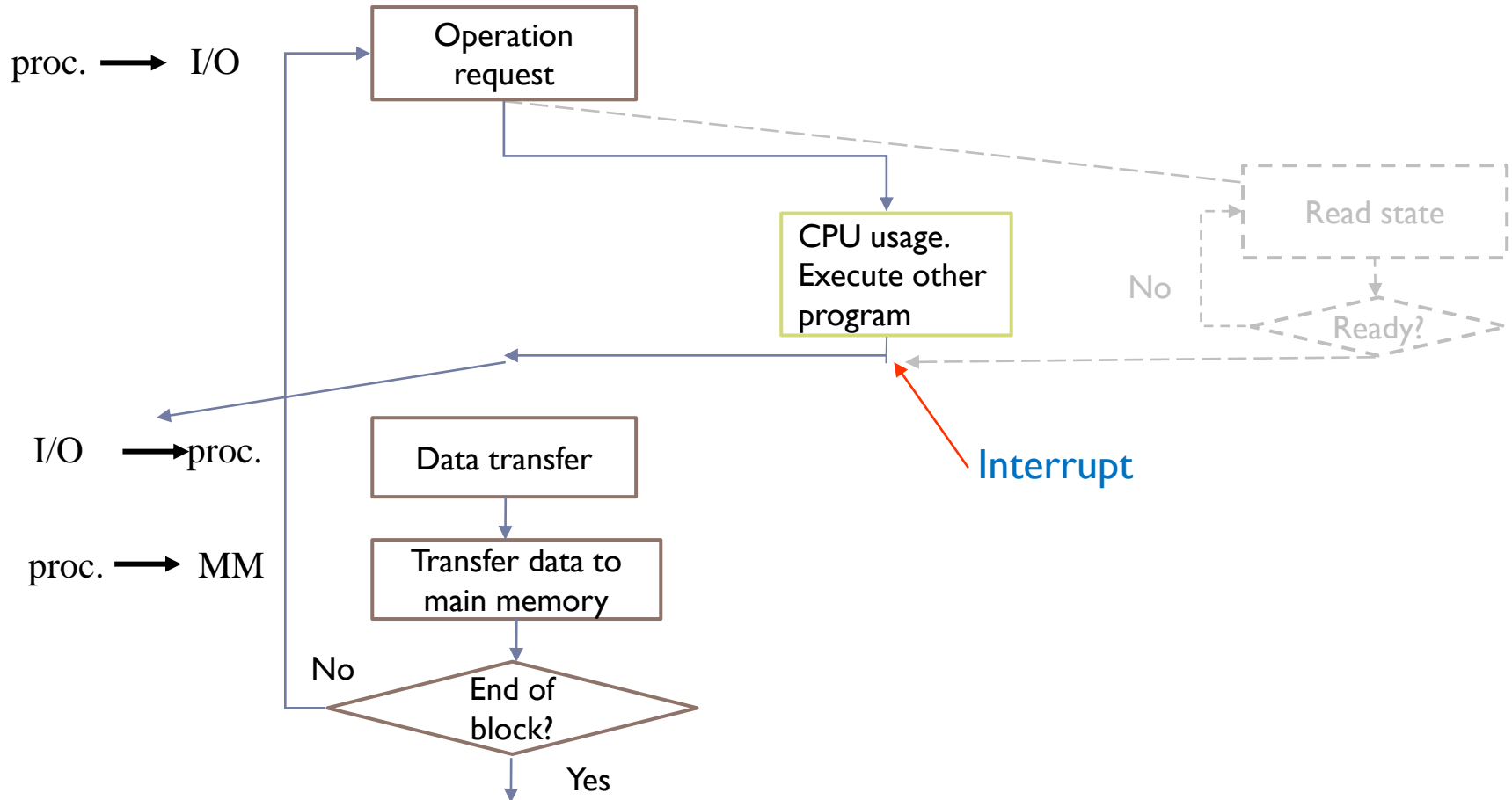
Synchronization loop

Transfer loop

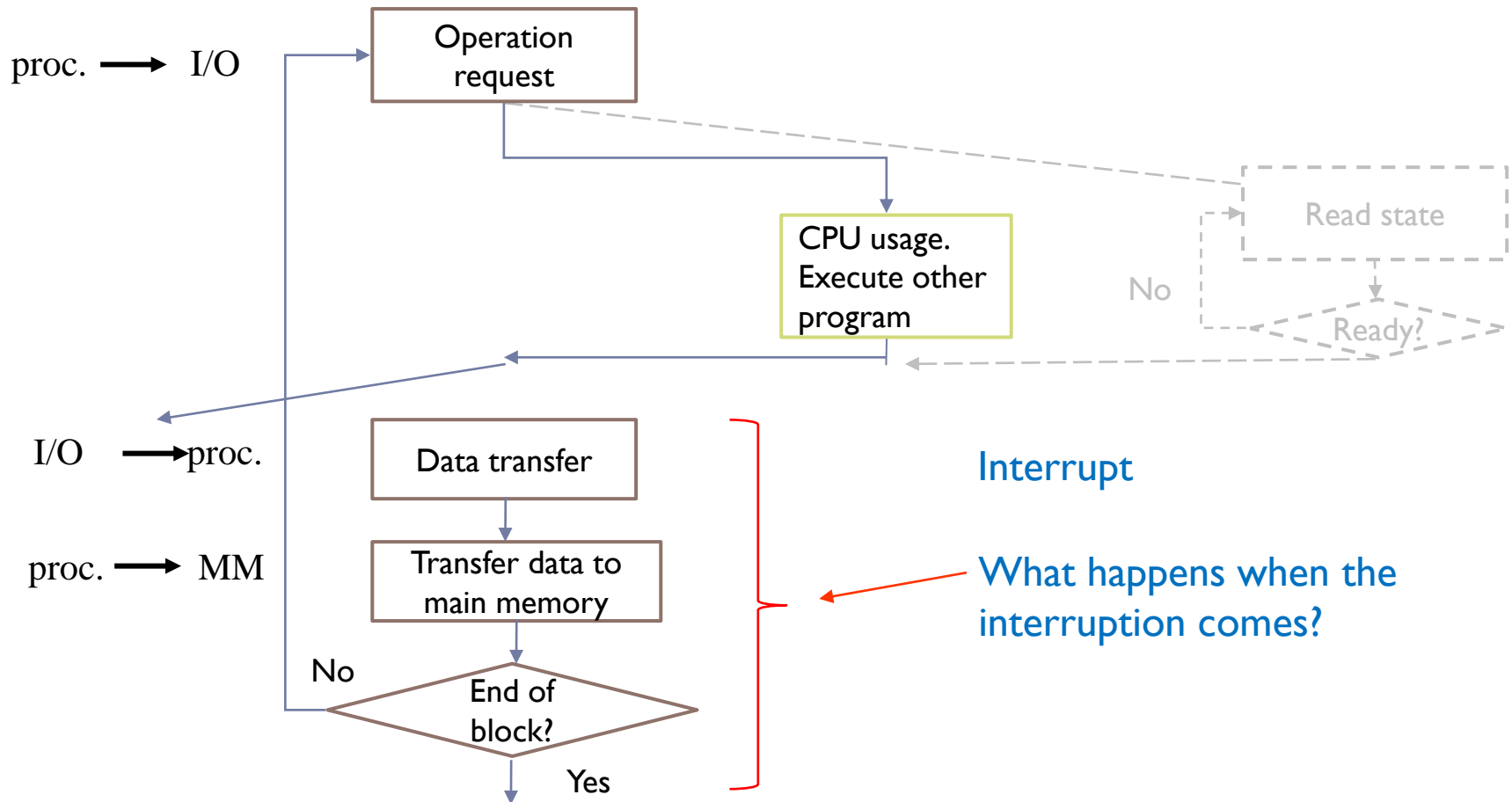
Main problem of the programmed I/O



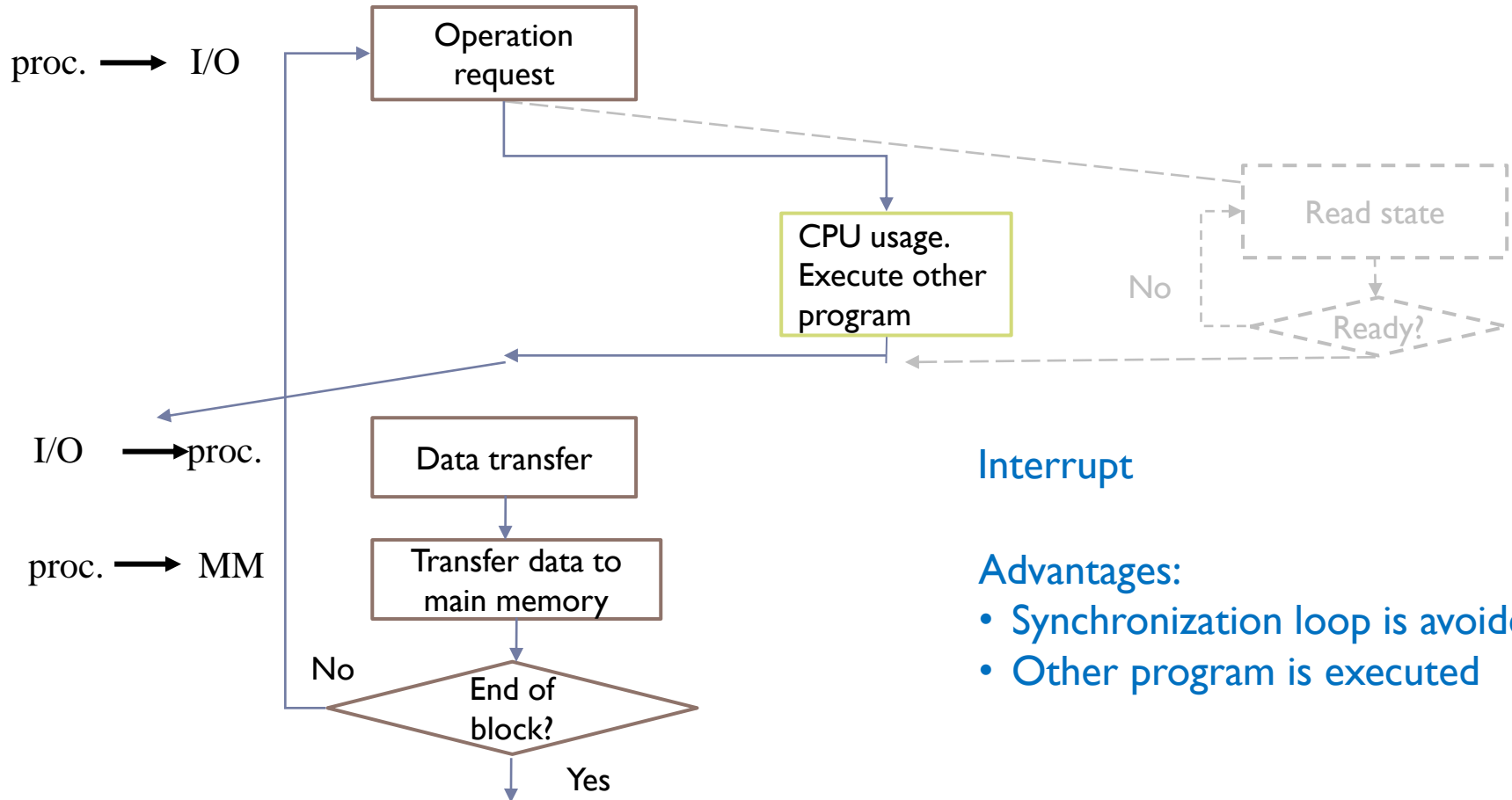
Interrupt I/O



Interrupt I/O



Interrupt I/O

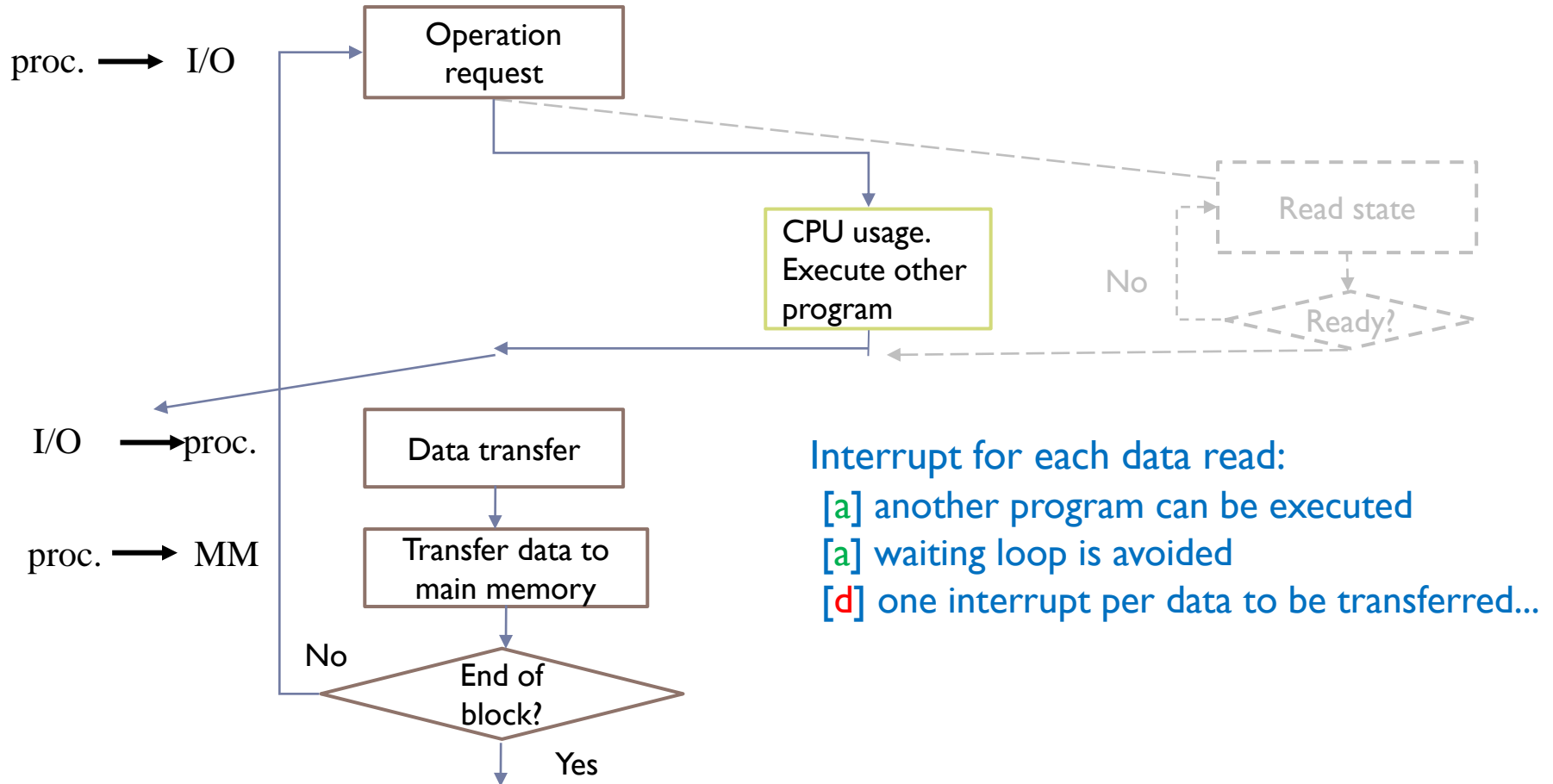


Interrupt

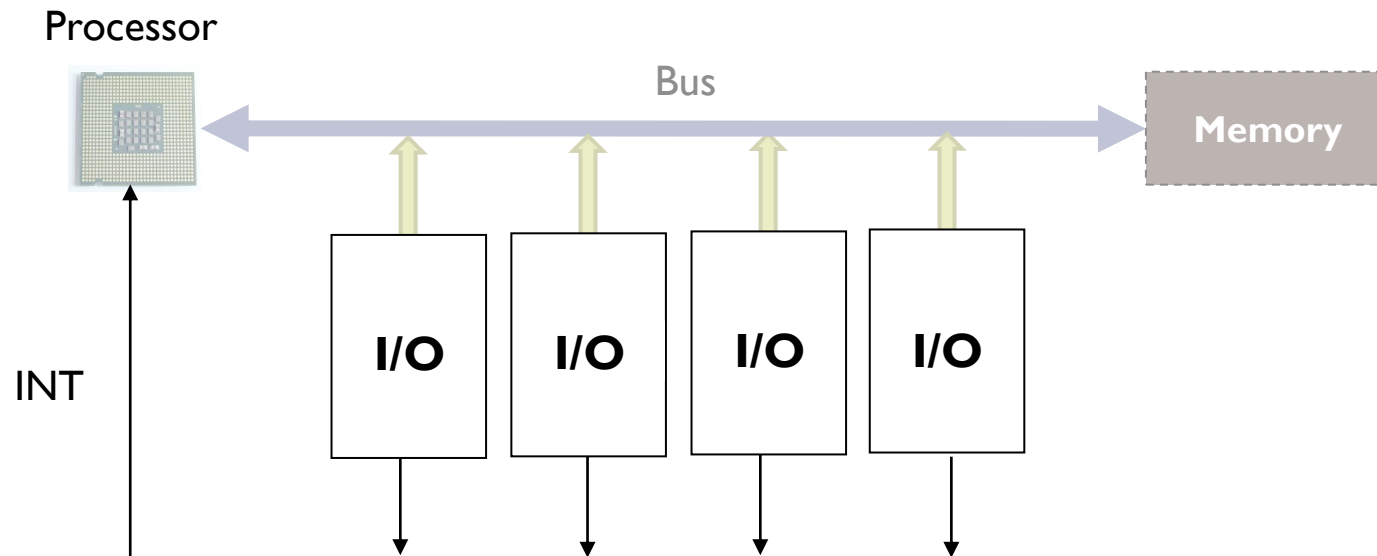
Advantages:

- Synchronization loop is avoided
- Other program is executed

Interrupt I/O

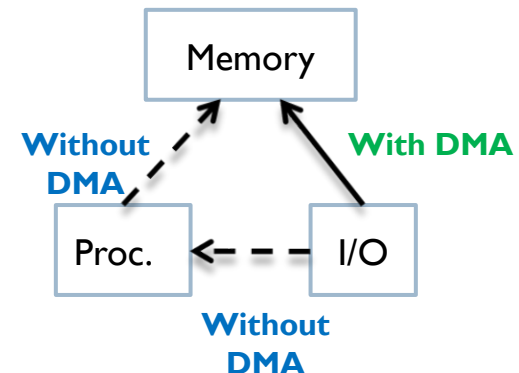


Interrupt driven I/O



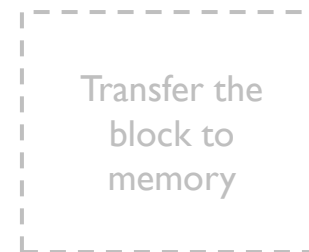
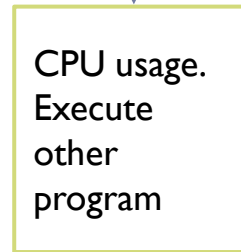
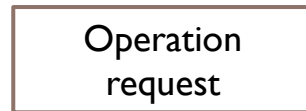
DMA I/O

- ▶ DMA: Direct Memory Access
- ▶ CPU does not carry out the transfer between the I/O module and the memory
 - ▶ With interrupts the synchronization loop is avoided, but the transfer is carry out by CPU
 - ▶ For a block with N bytes, N interrupts are needed
- ▶ Using DMA, the whole transfer is done by the I/O module
 - ▶ Only **one** interrupt at the end

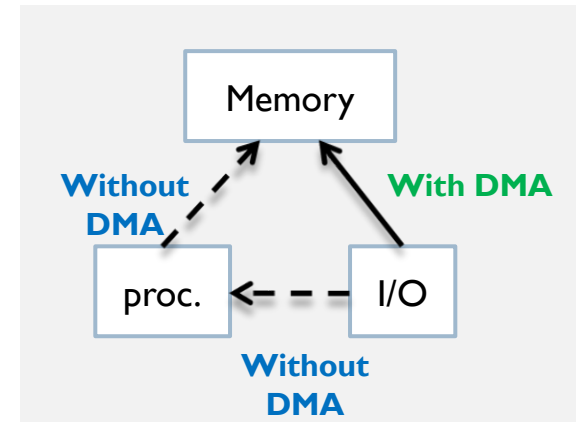


Transfer a block using DMA

Proc. → I/O

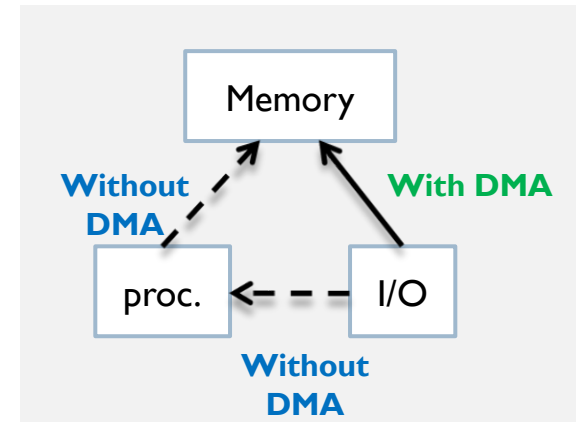
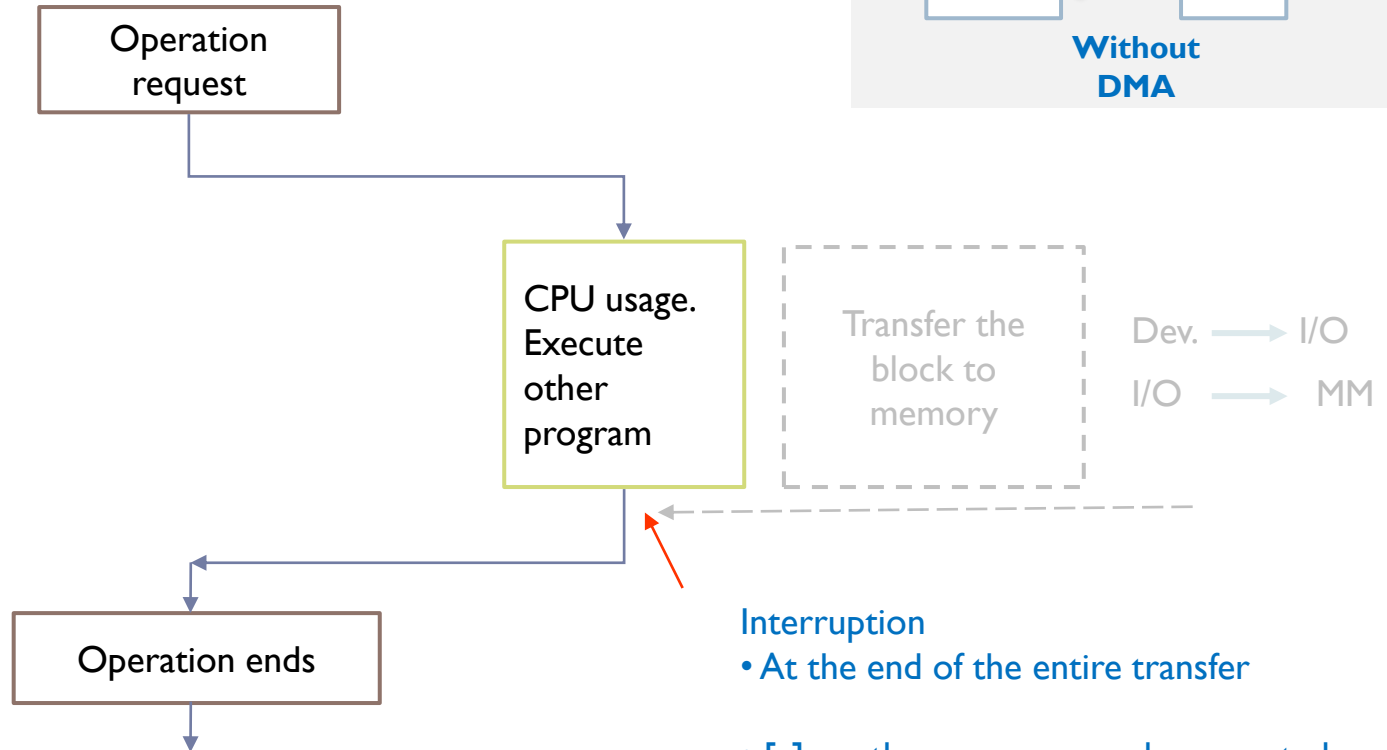


Dev. → I/O
I/O → MM



Transfer a block using DMA

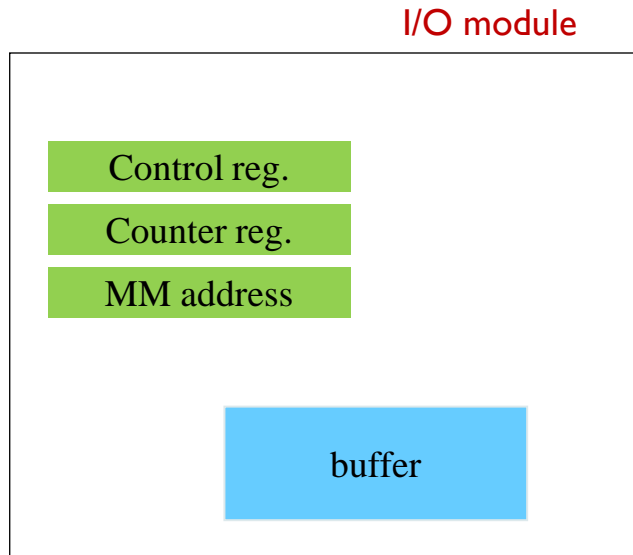
Proc. → I/O



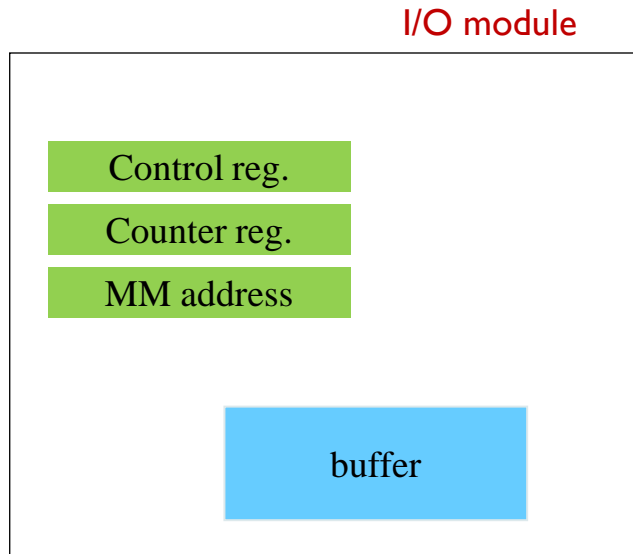
Interrupt

- At the end of the entire transfer
- [v] another program can be executed
- [v] a single interrupt

Simplified structure of I/O module for DMA



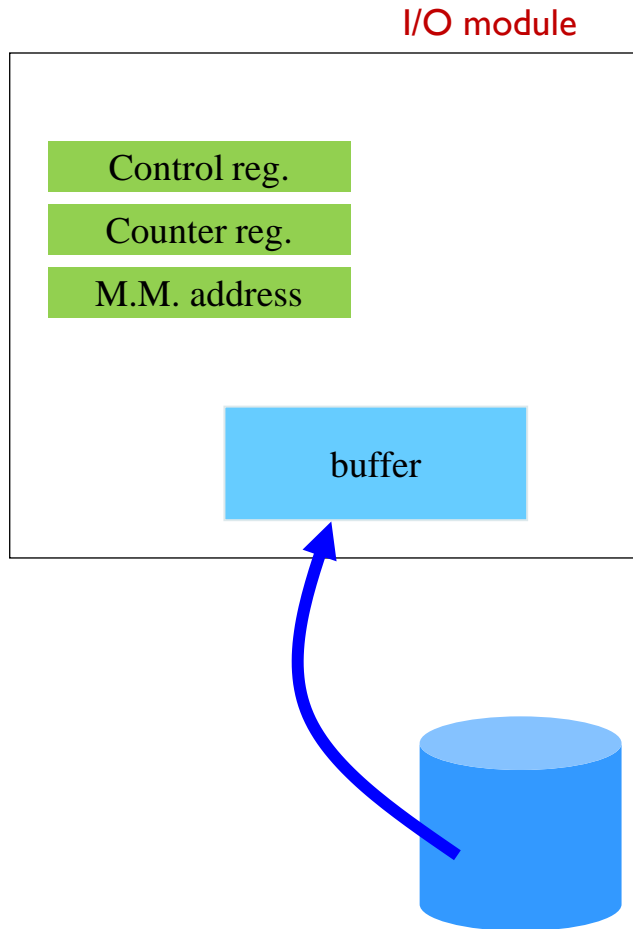
Data transfer with DMA



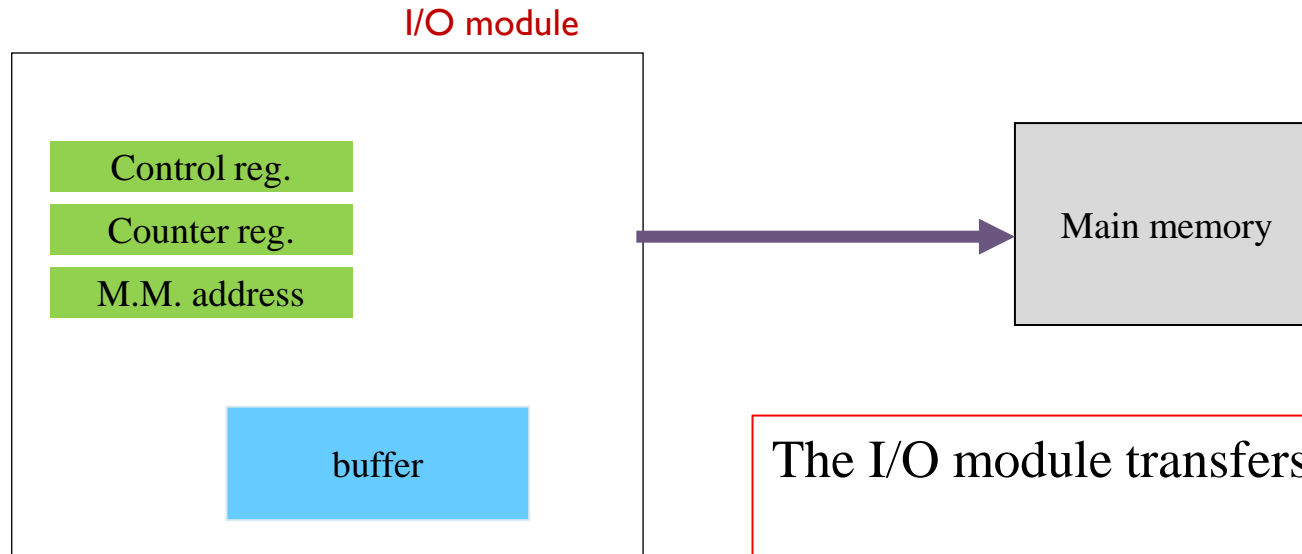
- ▶ The processor writes in I/O registers (using I/O instructions)
 - ▶ Operation (**control reg.**)
 - ▶ Read, write, etc.
 - ▶ The number of bytes to transfer (**counter reg.**)
 - ▶ **Memory address** where:
 - ▶ Data are stored (write in device)
 - ▶ Store the data (reading from device)

Data transfer with DMA

- ▶ I/O module transfers the data block from the device to the internal buffer inside the I/O module (in a reading operation)



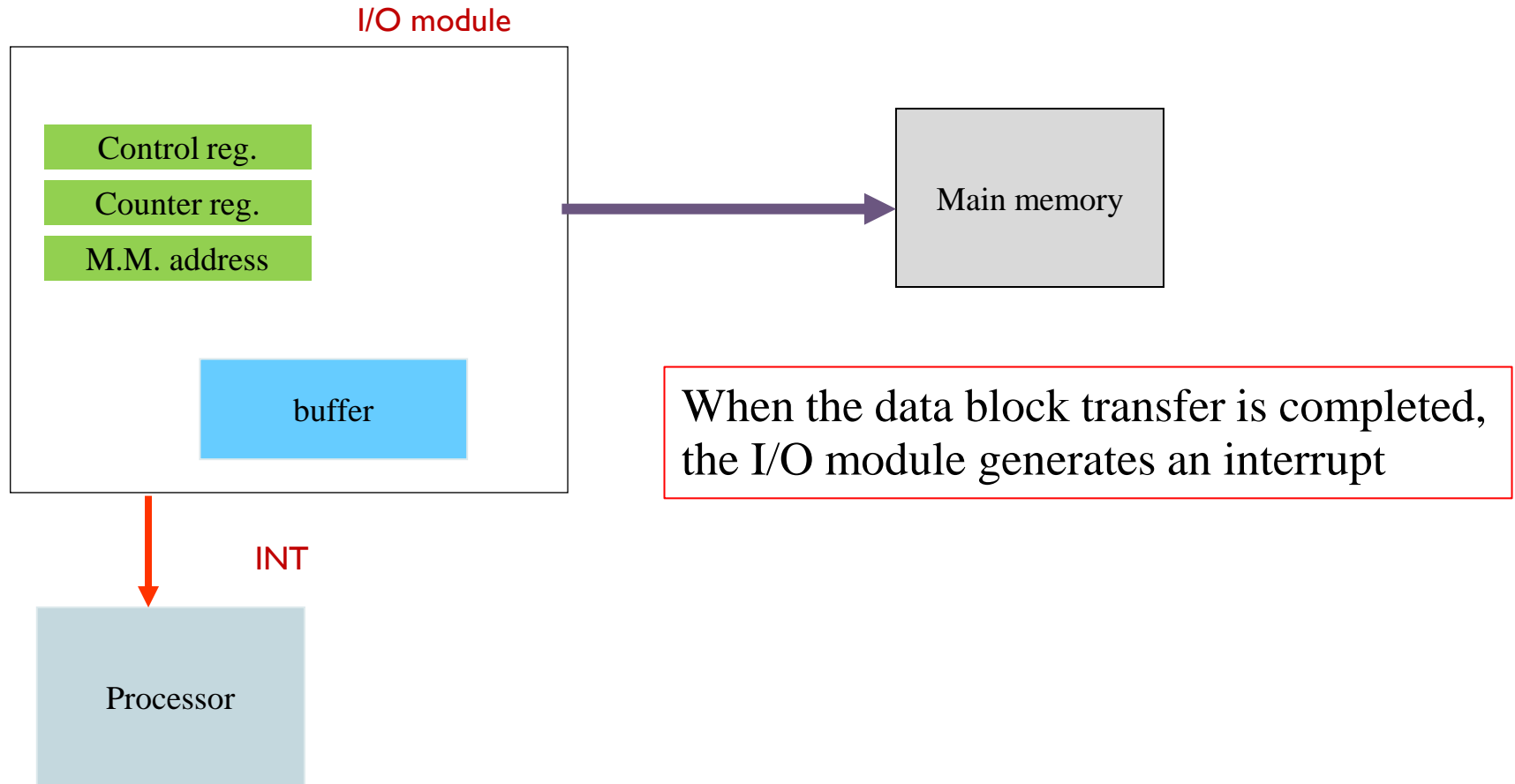
Data transfer with DMA



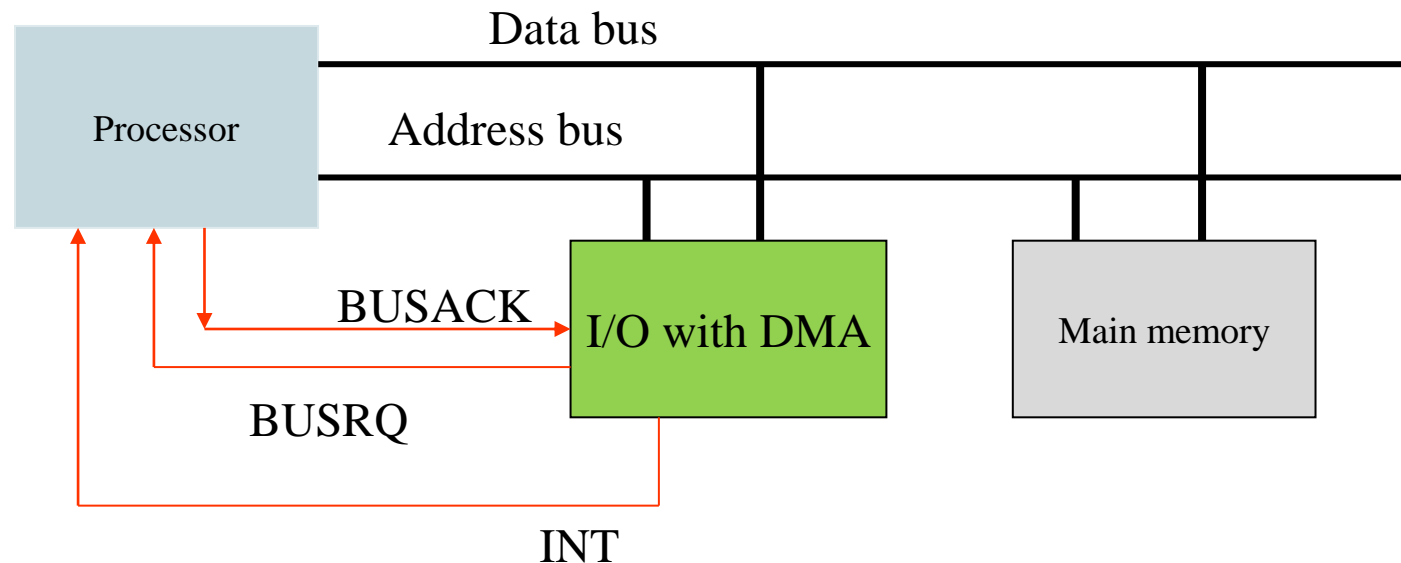
The I/O module transfers the data block:

```
while (counter > 0)
{
    Byte (word) -> MP[MM_address]
    MM_address++
    counter--
}
```

Data transfer with DMA

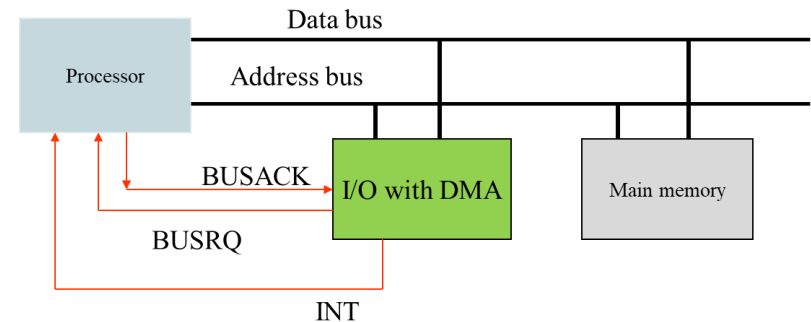


I/O module access to M.M.



- ▶ A coordination is needed to control the access to memory from the processor and I/O modules

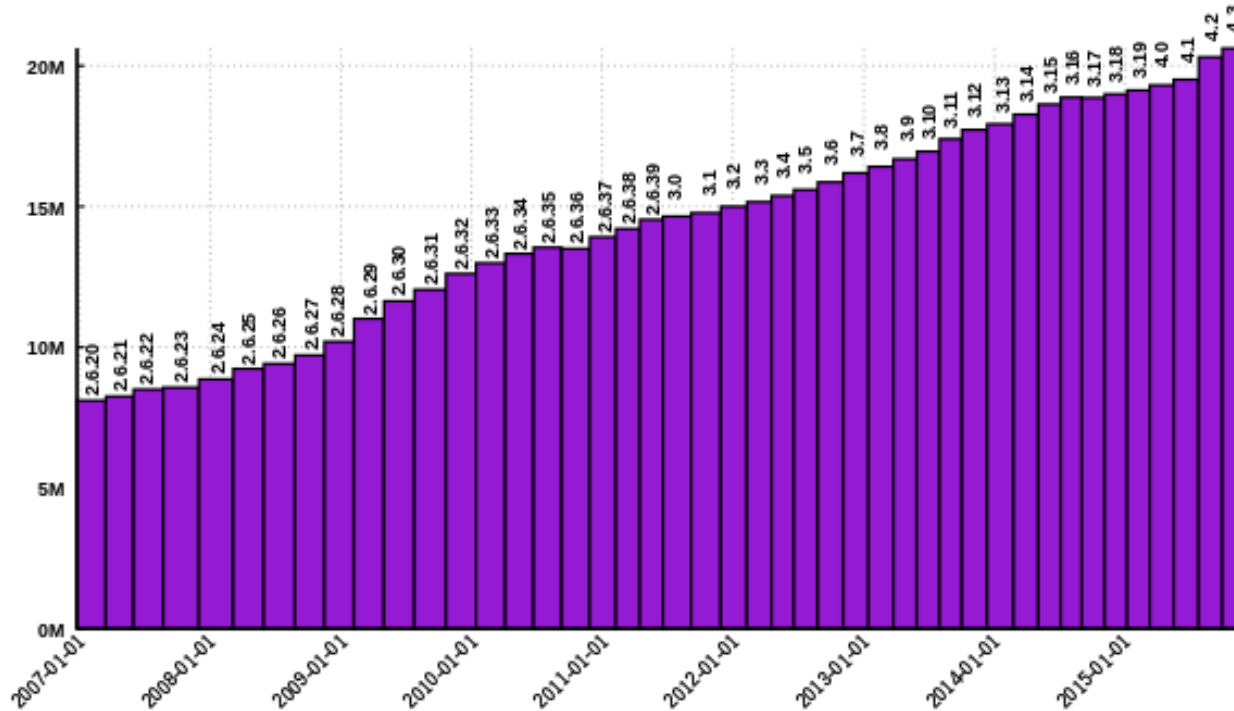
I/O module access to MM: Cycle stealing



- ▶ When the I/O module is ready to transfer a word:
 - ▶ Activates **BUSRQ** signal to request bus access
 - ▶ At the end of each phase of an instruction, the processor checks this signal. If this signal is activated, the processor does not use the buses and activate the **BUSACK** signal
 - ▶ The I/O module access to memory and then deactivate **BUSRQ** signal
 - ▶ The processor then can use the buses
 - ▶ At the end of the data block transfer, the I/O module sends an interrupt signal to the processor.

Curiosity: the importance of drivers

Linux kernel



Lines of code
of the
Linux kernel

- ▶ 70% of Linux code is related to device drivers.

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 6

Input/Output Systems

Computer Structure
Bachelor in Computer Science and Engineering

