Systems and Networking I

Applied Computer Science and Artificial Intelligence

2025-2026

SAPIENZA
UNIVERSITÀ DI ROMA

Gabriele Tolomei

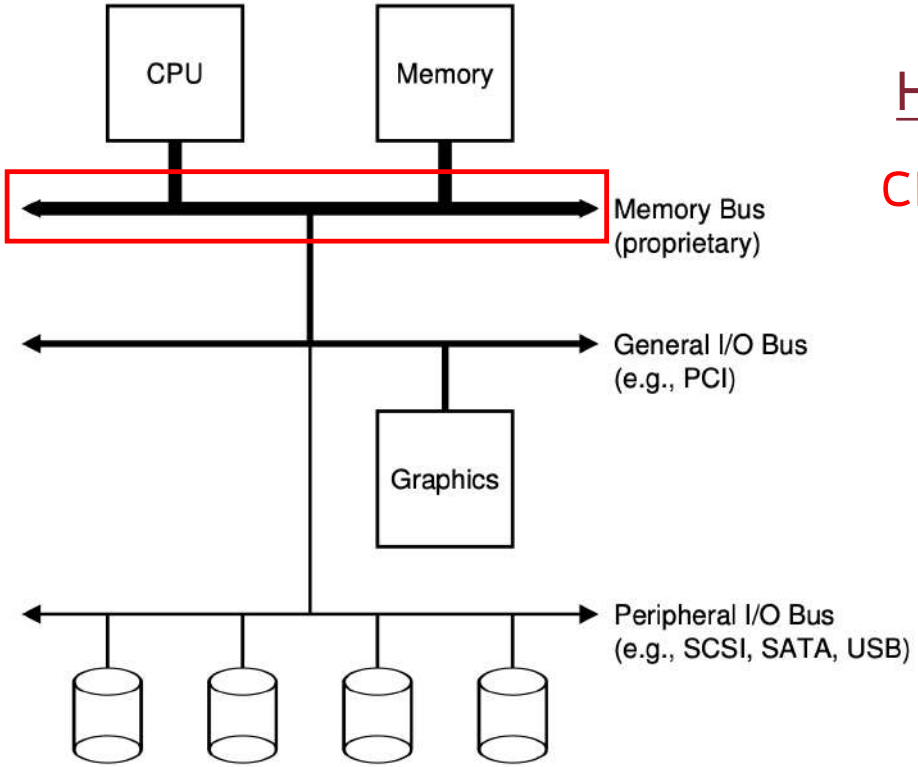
Computer Science Department

Sapienza Università di Roma

tolomei@di.uniroma1.it

A Quick Recap on I/O

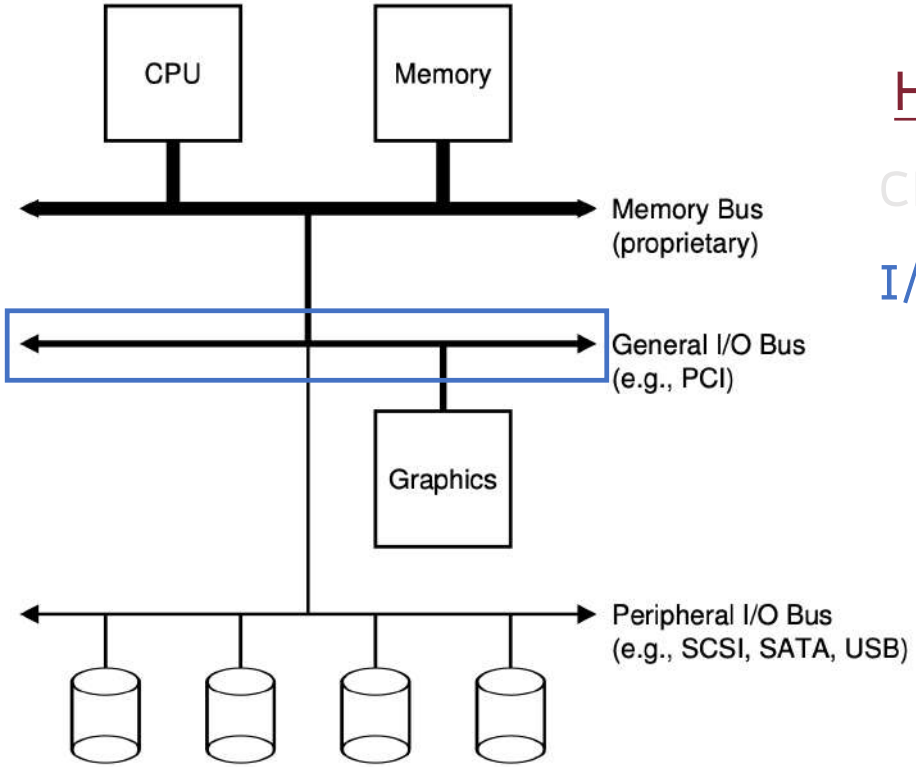
Remember the Basic System Architecture



Hierarchy of Communication Buses

CPU-Memory high-speed bus

Remember the Basic System Architecture

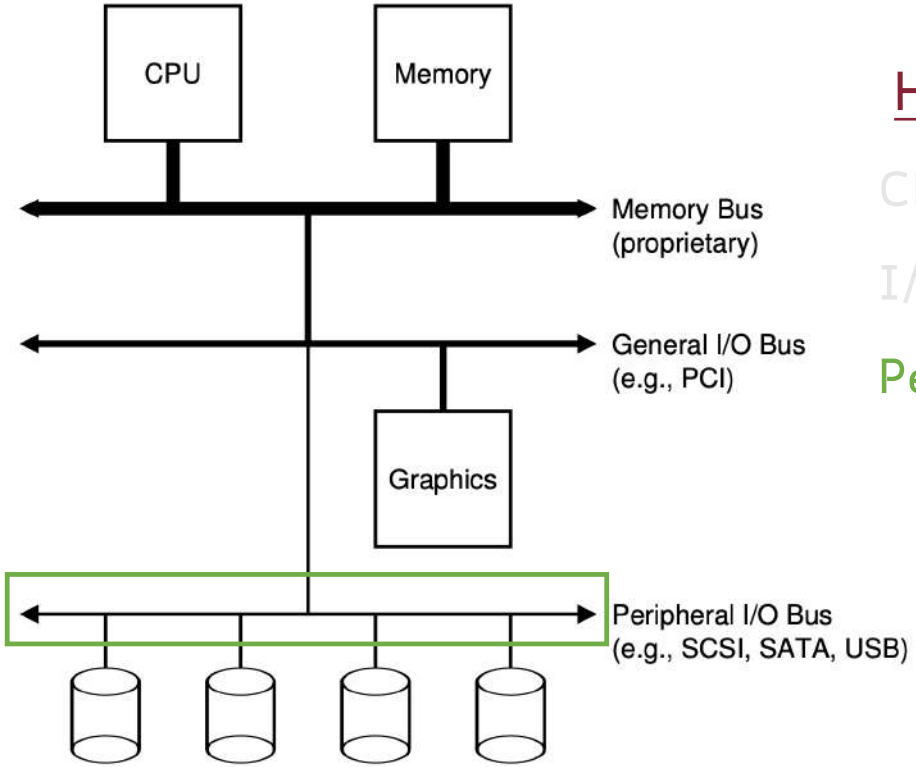


Hierarchy of Communication Buses

CPU-Memory high-speed bus

I/O bus

Remember the Basic System Architecture



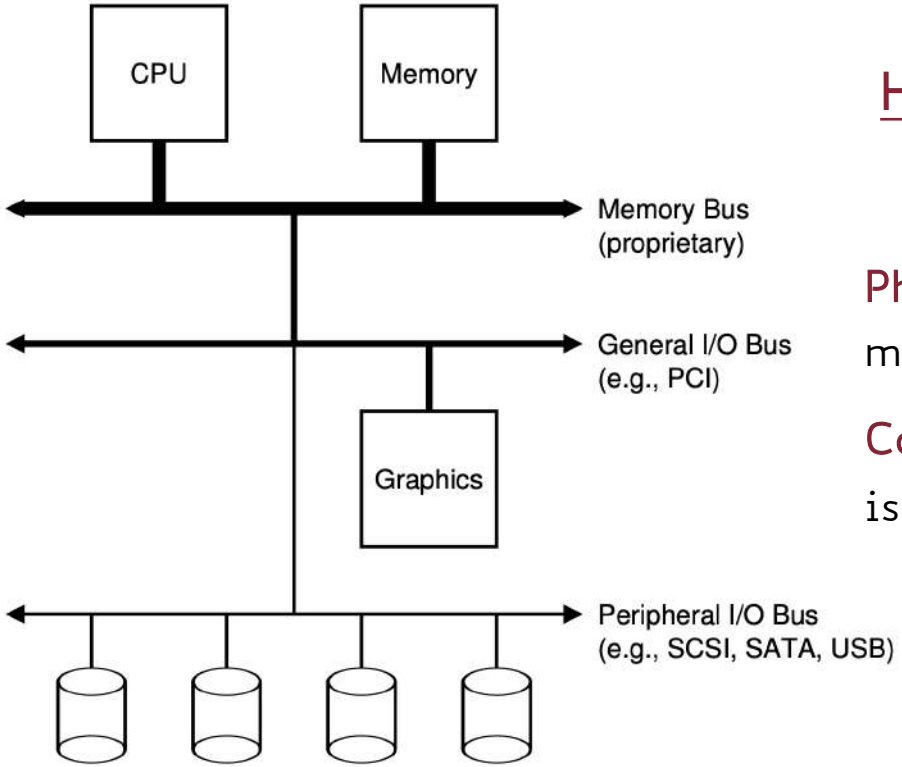
Hierarchy of Communication Buses

CPU-Memory high-speed bus

I/O bus

Peripheral bus

Remember the Basic System Architecture



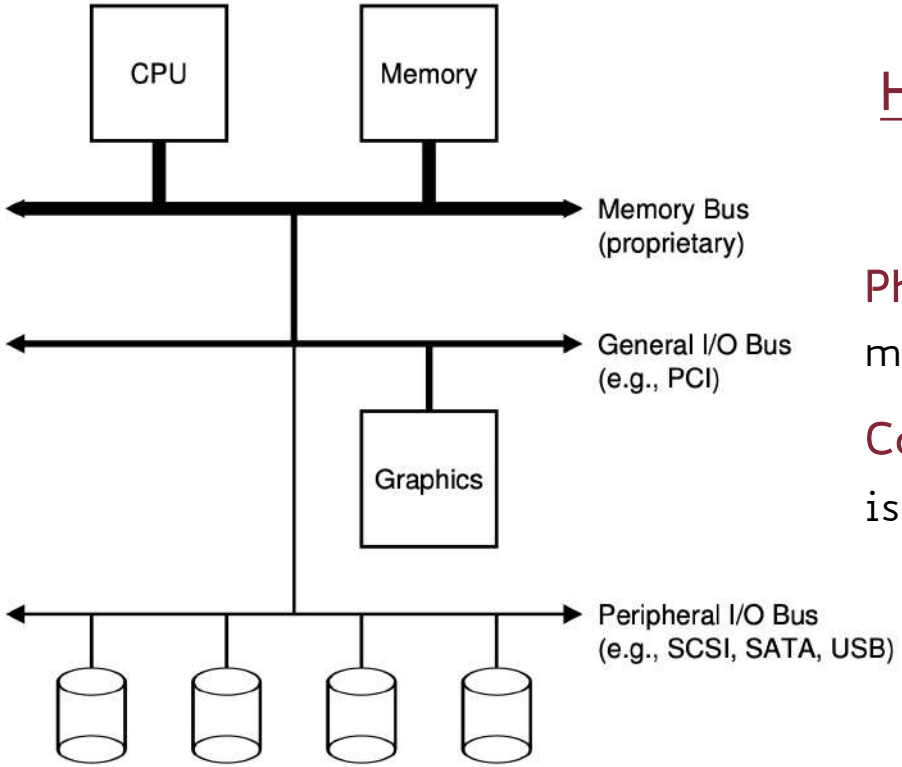
Hierarchy of Communication Buses

Why?

Physics: The faster a bus is, the shorter it must be!

Costs: Engineering a high-performance bus is expensive!

Remember the Basic System Architecture



Hierarchy of Communication Buses

Why?

Physics: The faster a bus is, the shorter it must be!

Costs: Engineering a high-performance bus is expensive!

High-speed I/O devices are closer to the CPU (e.g., graphics card)

Low-speed I/O devices are closer to the CPU (e.g., hard disks)

A Canonical I/O Device: Components

- Each I/O device is made of **2 parts**:
  - the **physical device** itself
  - the **device controller** (chip or set of chips controlling a family of physical devices)

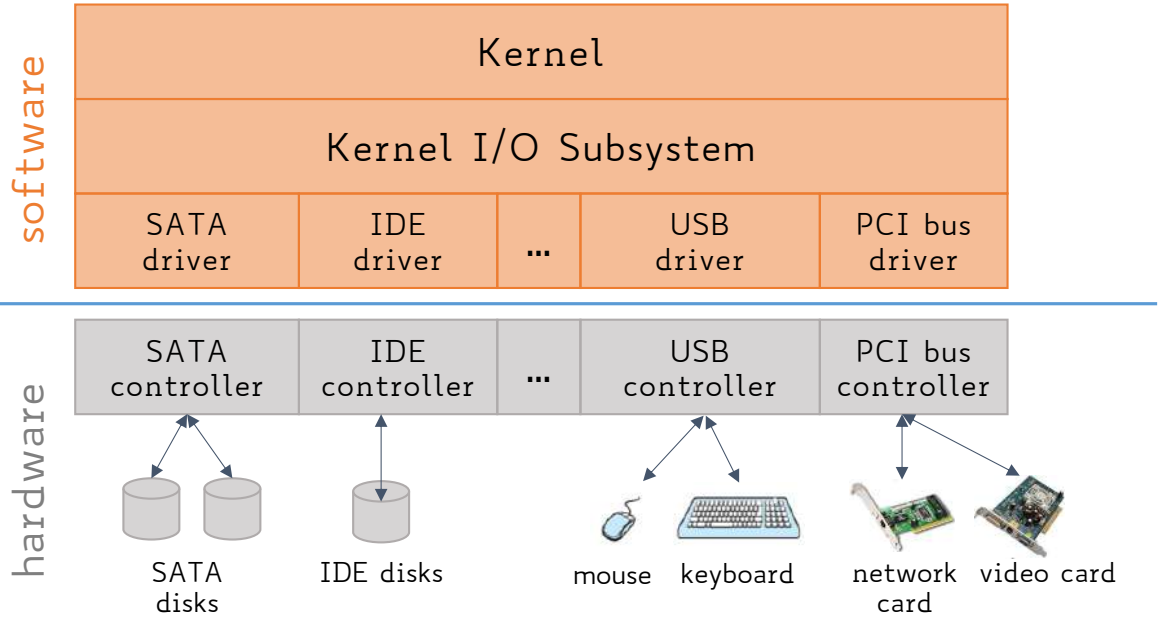
A Canonical I/O Device: Components

- Each I/O device is made of **2 parts**:
  - the **physical device** itself
  - the **device controller** (chip or set of chips controlling a family of physical devices)
- Can be categorized as:
  - storage, communications, user-interface, etc.

A Canonical I/O Device: Components

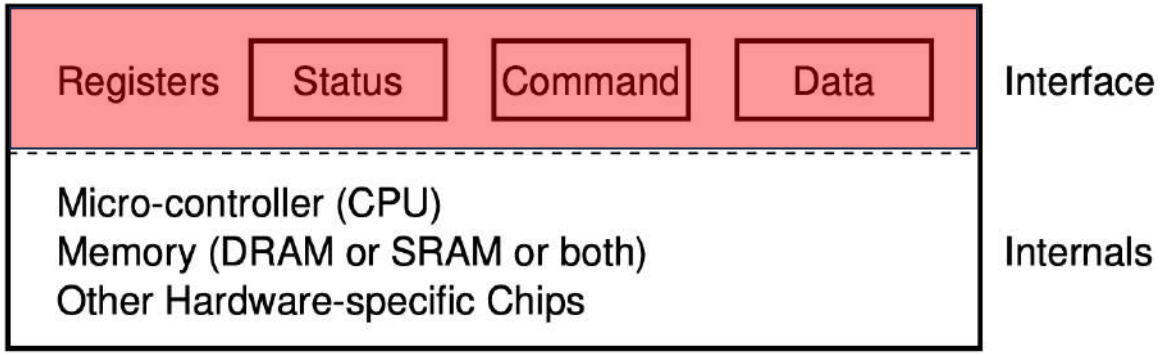
- Each I/O device is made of **2 parts**:
  - the **physical device** itself
  - the **device controller** (chip or set of chips controlling a family of physical devices)
- Can be categorized as:
  - storage, communications, user-interface, etc.
- OS talks to a device controller using a specific **device driver**

Device Drivers: OS Abstraction



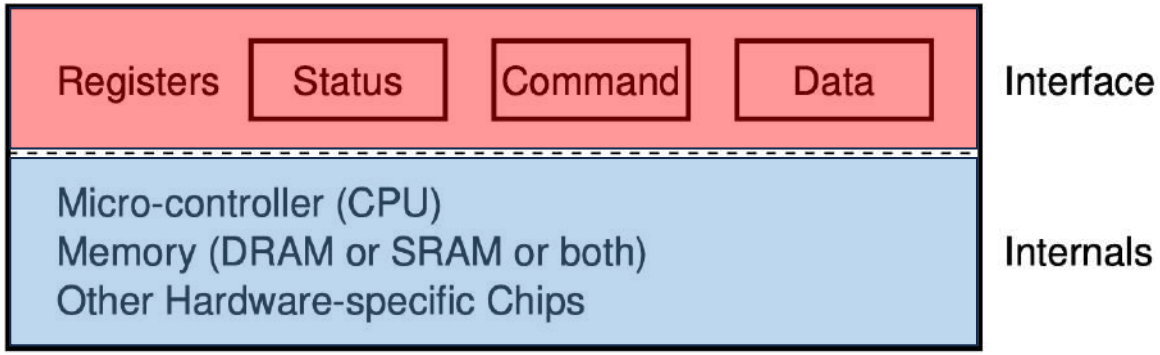
Device Controllers

- Every device controller has:
  - an **interface** → a number of dedicated registers to communicate with it



Device Controllers

- Every device controller has:
  - an **interface** → a number of dedicated registers to communicate with it
  - an **internal structure** → logic circuitry (vendor-specific)



Device Controllers

- **Status registers** → provide status information to the CPU about the I/O device (e.g., idle, ready for input, busy, error, transaction complete)
- **Command/Configuration/Control registers** → used by the CPU to configure and control the device
- **Data registers** → used to read data from or write data to the I/O device

The Basic Protocol: OS-I/O

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

The OS waits until the device is ready by repeatedly checking the STATUS register

polling

The Basic Protocol: OS-I/O

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

Send some data to the DATA register
(e.g., a 4KiB page to disk)

The Basic Protocol: OS-I/O

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

Issue the command

The Basic Protocol: OS-I/O

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

The OS waits for the device to finish again by polling in a loop

polling

The Basic Protocol: OS-I/O

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

The CPU is responsible for the whole data transfer

Programmed I/O

The Basic Protocol: OS-I/O

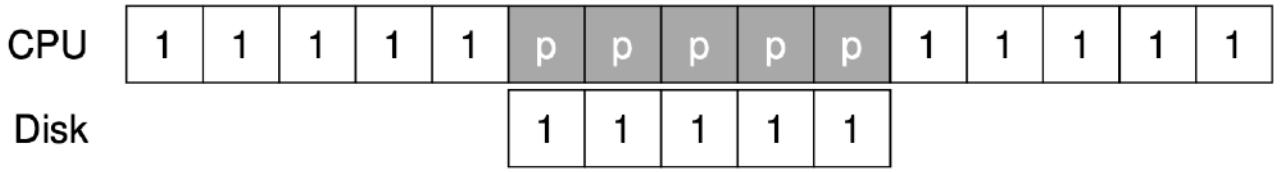
```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

The CPU wastes a lot of time checking a possibly slow I/O device

Programmed I/O + Polling

Lowering CPU Waste: How?

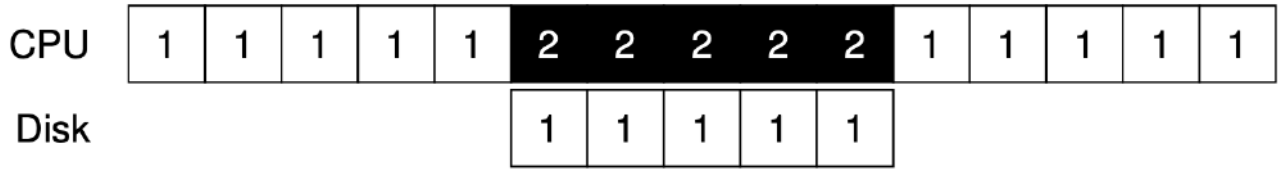
Polling: wastes CPU cycles waiting on a condition



Process 1 runs for some time, issues the I/O request and, while the request is being served, the OS repeatedly checks the status of the device (**p**)

Lowering CPU Waste: Interrupts!

Interrupt-driven I/O: allows overlap between CPU and I/O

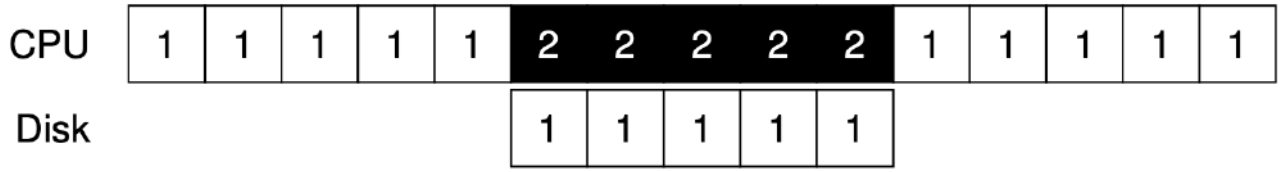


Process 1 runs for some time, issues the I/O request and, while the request is being served, the OS switches to Process 2

The I/O device will send an interrupt when done!

Lowering CPU Waste: Interrupts!

Interrupt-driven I/O: allows overlap between CPU and I/O

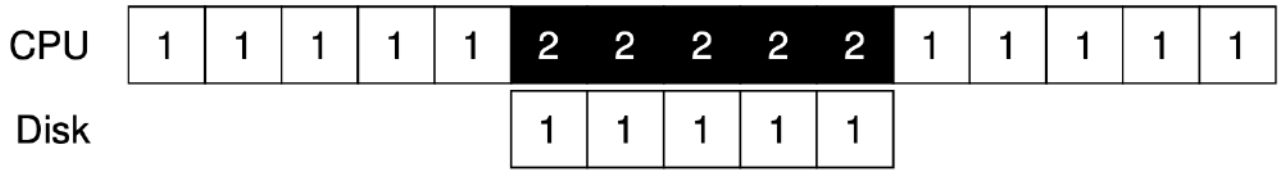


Process 1 runs for some time, issues the I/O request and, while the request is being served, the OS switches to **Process 2**

Is Interrupt-driven I/O **always** better than Programmed I/O + Polling?

Lowering CPU Waste: Interrupts!

Interrupt-driven I/O: allows overlap between CPU and I/O



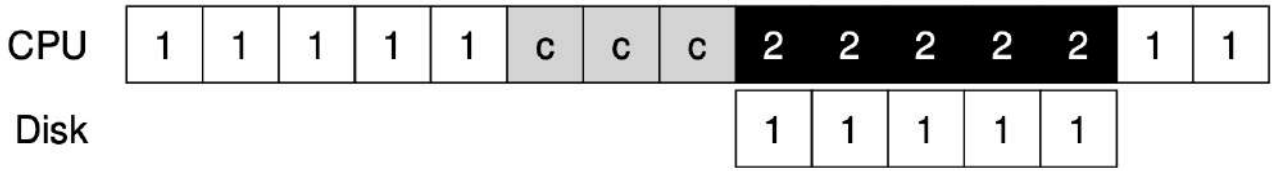
Process 1 runs for some time, issues the I/O request and, while the request is being served, the OS switches to **Process 2**

Is Interrupt-driven I/O **always** better than Programmed I/O + Polling?

NO! It depends on the speed of the I/O device/task

Beyond Programmed-I/O

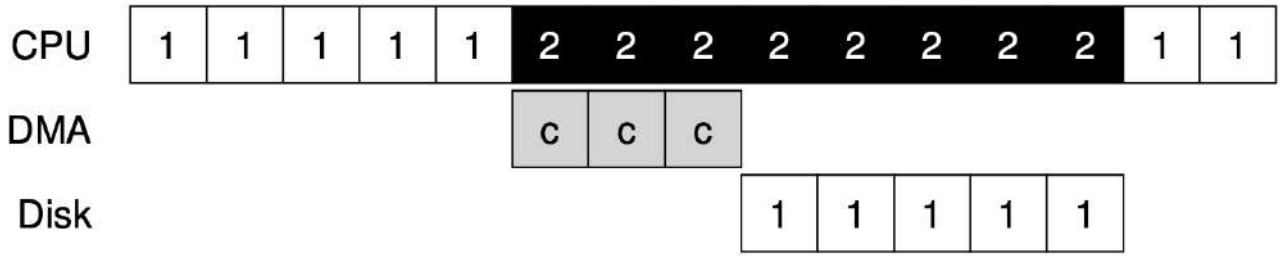
Programmed I/O: the CPU actually transfers data to I/O devices



Process 1 runs for some time, starts the I/O request by copying data from main memory to the I/O device one word at a time (c); when this is done the I/O begins and the OS switches to **Process 2**

Direct Memory Access (DMA)

DMA: a dedicated component to orchestrate memory-I/O transfers



Process 1 runs for some time, the OS starts the I/O by issuing a DMA request, then immediately switches to Process 2. Once the whole I/O task is done, the DMA controller sends an interrupt

To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status

To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status
- **Interrupt-driven**
  - CPU receives an interrupt from the controller (device or DMA) once the I/O task is done (either successfully or abnormally)

To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status
- **Interrupt-driven**
  - CPU receives an interrupt from the controller (device or DMA) once the I/O task is done (either successfully or abnormally)
- **Programmed I/O**
  - CPU does the actual work of moving data

To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status
- **Interrupt-driven**
  - CPU receives an interrupt from the controller (device or DMA) once the I/O task is done (either successfully or abnormally)
- **Programmed I/O**
  - CPU does the actual work of moving data
- **Direct Memory Access (DMA)**
  - CPU delegates off the work to a dedicated DMA controller

To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status
- **Interrupt-driven**
  - CPU receives an interrupt from the controller (device or DMA) once the I/O task is done (either successfully or abnormally)

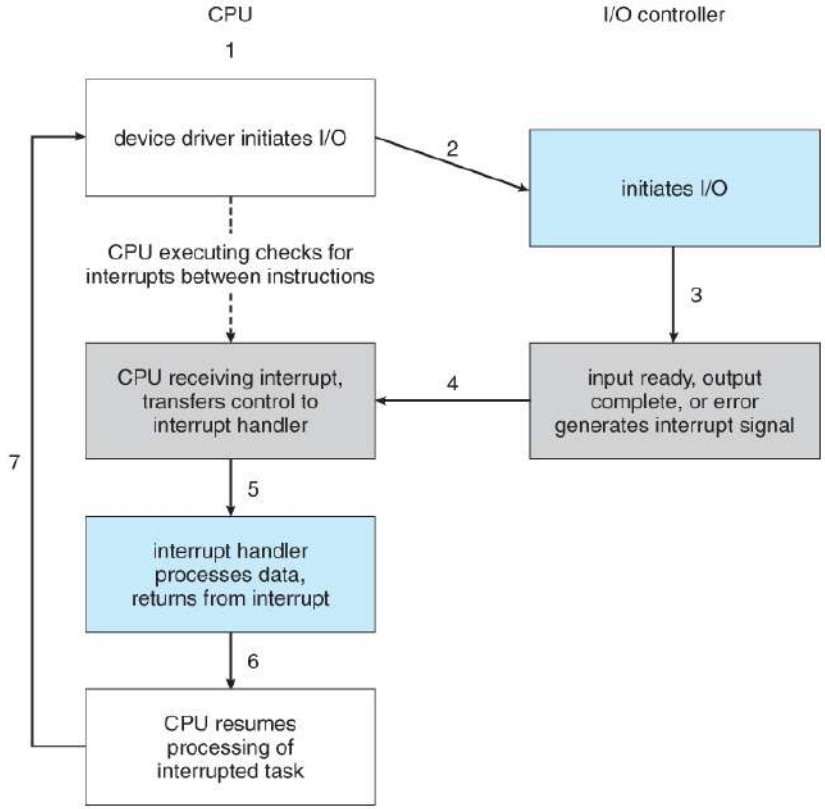
HOW?

- **Programmed I/O**
  - CPU does the actual work of moving data
- **Direct Memory Access (DMA)**
  - CPU delegates off the work to a dedicated DMA controller

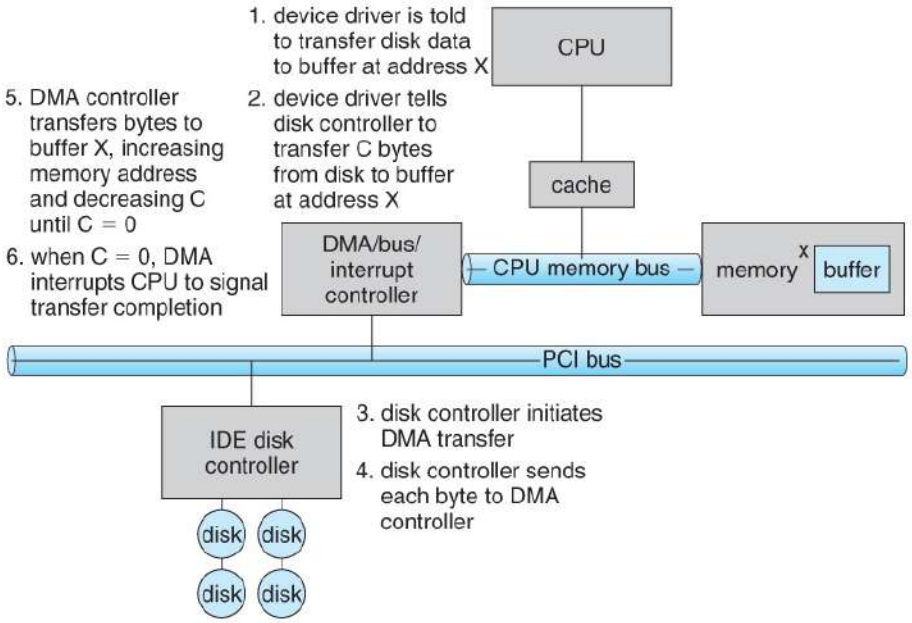
To Wrap Up: Performing I/O Tasks

- **Polling**
  - CPU periodically checks for the I/O task status
- **Interrupt-driven**
  - CPU receives an interrupt from the controller (device or DMA) once the I/O task is done (either successfully or abnormally)
- **Programmed I/O** WHO?
  - CPU does the actual work of moving data
- **Direct Memory Access (DMA)**
  - CPU delegates off the work to a dedicated DMA controller

How: Interrupt-driven I/O



Who: Direct Memory Access (DMA)



Overcome the limitation of Programmed I/O

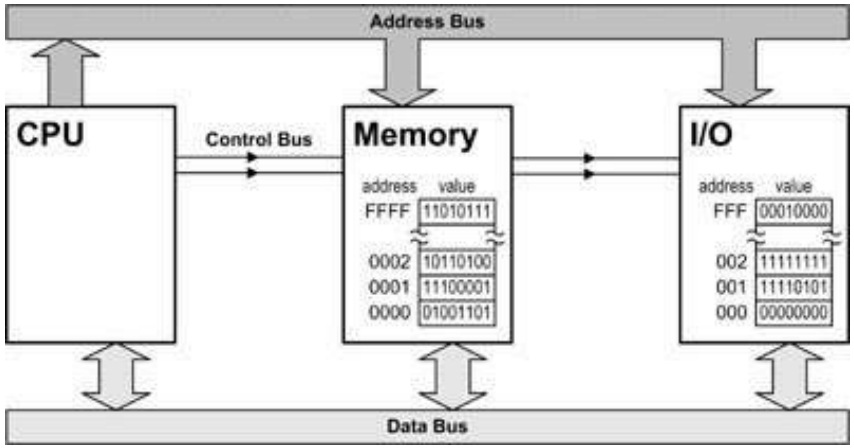
Maybe wasteful to tie up the CPU transferring data in and out of registers one word at a time

Useful for devices that transfer large quantities of data (such as disk controllers)

Typically, used in combination with interrupt-driven I/O

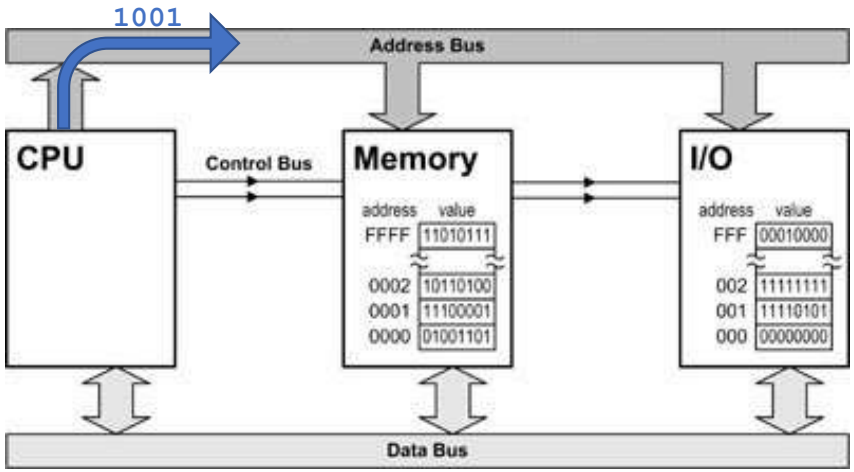
But how does the CPU/OS actually communicate with I/O devices?

Addressing Using the System Bus



How does CPU reference Memory addresses?

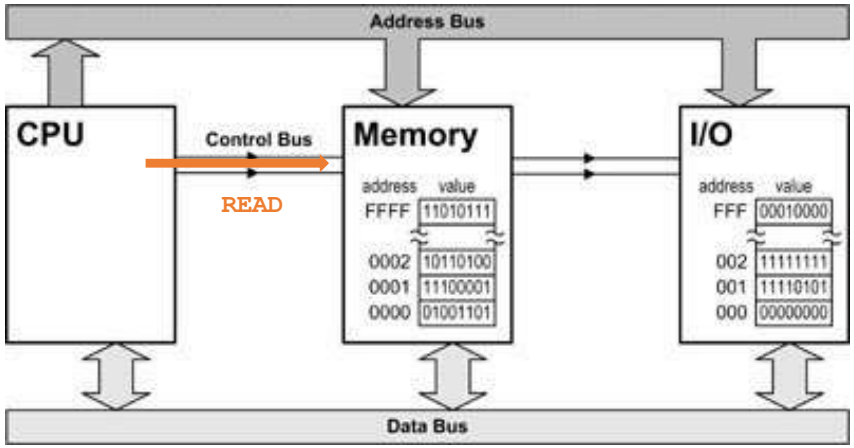
Addressing Using the System Bus



How does CPU reference Memory addresses?

It puts the address of a byte of memory on the address bus

Addressing Using the System Bus

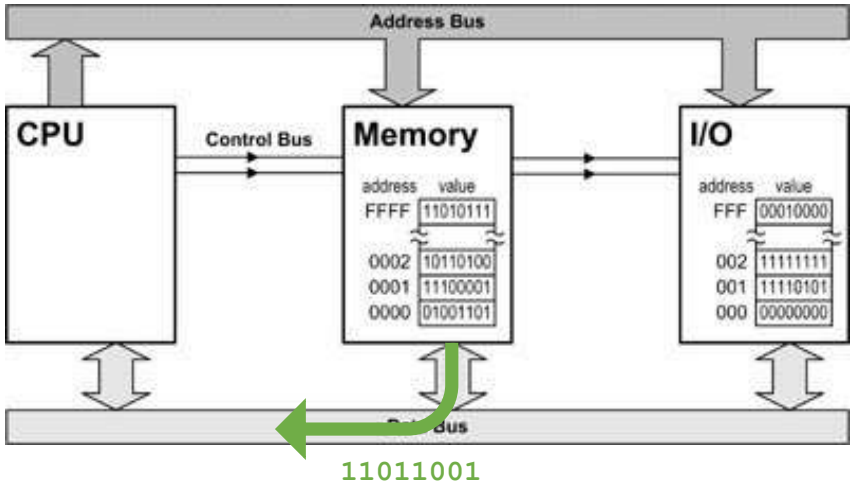


How does CPU reference Memory addresses?

It puts the address of a byte of memory on the address bus

It raises the **READ** signal on the control bus

Addressing Using the System Bus



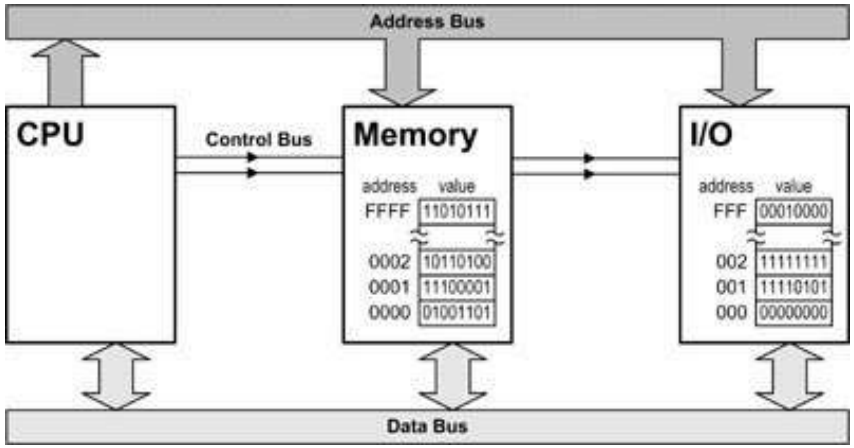
How does CPU reference Memory addresses?

It puts the address of a byte of memory on the address bus

It raises the **READ** signal on the control bus

Eventually, the RAM replies with the memory content on the data bus (usually a fixed-sized chunk of data, e.g., 8 ÷ 64 bytes)

Addressing Using the System Bus



How does CPU reference Memory addresses?

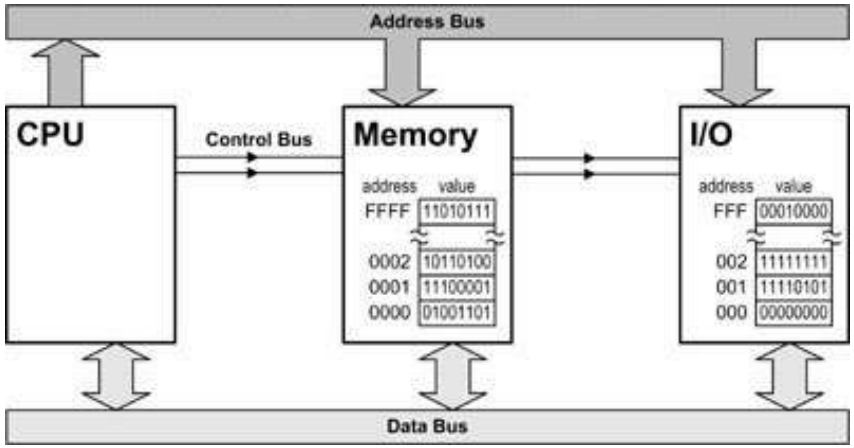
It puts the address of a byte of memory on the address bus

It raises the **READ** signal on the control bus

Eventually, the RAM replies with the memory content on the data bus (usually a fixed-sized chunk of data, e.g., 8 ÷ 64 bytes)

How about I/O devices? How to distinguish between Memory and I/O devices?

Addressing Using the System Bus



How does CPU reference Memory addresses?

It puts the address of a byte of memory on the address bus

It raises the **READ** signal on the control bus

Eventually, the RAM replies with the memory content on the data bus (usually a fixed-sized chunk of data, e.g., 8 ÷ 64 bytes)

If the **address bus is shared** between memory and I/O there is a special pin called "**M/IO**" that asserts whether the CPU wants to talk to memory (**M/IO = 0**) or an I/O device (**M/IO = 1**)

Port- vs. Memory-Mapped I/O

- CPU can talk to a device controller in **2 ways**:

Port- vs. Memory-Mapped I/O

- CPU can talk to a device controller in **2 ways**:
  - **Port-mapped I/O (PMIO)** → referencing controller's registers using a separate I/O address space

Port- vs. Memory-Mapped I/O

- CPU can talk to a device controller in **2 ways**:
  - Port-mapped I/O (PMIO) → referencing controller's registers using a separate I/O address space
  - **Memory-mapped I/O (MMIO)** → mapping controller's registers to the same address space used for main memory

Port-Mapped I/O (PMIO)

- Each I/O device controller's register is mapped to a specific port (address) at boot-up time
  - typically, 16-bit on x86 → 65,536 ports

Port-Mapped I/O (PMIO)

- Each I/O device controller's register is mapped to a specific port (address) at boot-up time
  - typically, 16-bit on x86 → 65,536 ports
- Requires special class of CPU instructions (e.g., `IN`/`OUT`)
  - The `IN` instruction reads from an I/O device, `OUT` writes to it

Port-Mapped I/O (PMIO)

- Each I/O device controller's register is mapped to a specific port (address) at boot-up time
  - typically, 16-bit on x86 → 65,536 ports
- Requires special class of CPU instructions (e.g., **IN**/**OUT**)
  - The **IN** instruction reads from an I/O device, **OUT** writes to it
- With the **IN** or **OUT** instructions, **M/IO = 1**, so memory does not respond and the I/O chip does

Port-Mapped I/O (PMIO)

- Each I/O device controller's register is mapped to a specific port (address) at boot-up time
  - typically, 16-bit on x86 → 65,536 ports
- Requires special class of CPU instructions (e.g., **IN**/**OUT**)
  - The **IN** instruction reads from an I/O device, **OUT** writes to it
- With the **IN** or **OUT** instructions, **M/IO = 1**, so memory does not respond and the I/O chip does
- Mostly used for x86 legacy devices, much less common today

Port-Mapped I/O (PMIO): Example

```
IN   AL, 0x3F8        ; read from the serial port (COM1)
OUT  0x64, AL         ; write to keyboard controller
```

For **legacy devices**, the **port numbers** are historically **standardized**:

- COM1 → 0x3F8
- COM2 → 0x2F8
- Keyboard controller → 0x60/0x64
- Programmable Interrupt Controller (PIC) → 0x20 and 0xA0

These port ranges became *de facto* standards, so vendors conform to them for compatibility

Memory-Mapped I/O (MMIO)

- Memory-mapped I/O "wastes" some address space but doesn't need any special instruction

Memory-Mapped I/O (MMIO)

- Memory-mapped I/O "wastes" some address space but doesn't need any special instruction
- To the CPU, I/O device ports are just like normal memory addresses mapped into RAM at boot-up time

Memory-Mapped I/O (MMIO)

- Memory-mapped I/O "wastes" some address space but doesn't need any special instruction
- To the CPU, I/O device ports are just like normal memory addresses mapped into RAM at boot-up time
- The CPU uses `MOV`-like instructions to access I/O device registers

Memory-Mapped I/O (MMIO)

- Memory-mapped I/O "wastes" some address space but doesn't need any special instruction
- To the CPU, I/O device ports are just like normal memory addresses mapped into RAM at boot-up time
- The CPU uses `MOV`-like instructions to access I/O device registers
- The **`M/IO`** is not needed as every address requested by the CPU refers to main memory

Memory-Mapped I/O (MMIO): Example

```
MOV RAX, [0xF0000000]   ; read NIC register
MOV [0xF0000000], RAX   ; write NIC register
```

These addresses are **not hardcoded by the vendor**, they are discovered and allocated by firmware/OS

The OS enumerates PCI/PCIe devices and assigns MMIO addresses **dynamically**

The device advertises the size and number of required MMIO regions in its **PCI Base Address Registers (BARs)**

Port- vs. Memory-Mapped I/O

```
MOV DX,1234h
MOV AL,[DX]     ;reads memory address 1234h (memory address space)
IN AL,DX        ;reads I/O port 1234h (I/O address space)
```

Both put the value **1234h** on the CPU address bus, and both assert a **READ** operation on control bus

Port- vs. Memory-Mapped I/O

```
MOV DX,1234h
MOV AL,[DX]     ;reads memory address 1234h (memory address space)
IN AL,DX        ;reads I/O port 1234h (I/O address space)
```

The first one will set **M/IO = 0** to indicate that the address belongs to memory address space

Port- vs. Memory-Mapped I/O

```
MOV DX,1234h
MOV AL,[DX]     ;reads memory address 1234h (memory address space)
IN AL,DX        ;reads I/O port 1234h (I/O address space)
```

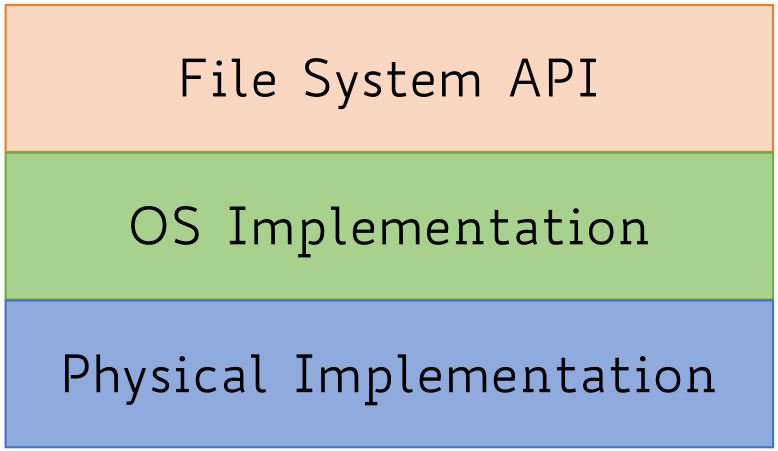
The second one will assert **M/IO = 1** to indicate that the address belongs to I/O address space

Port- vs. Memory-Mapped I/O

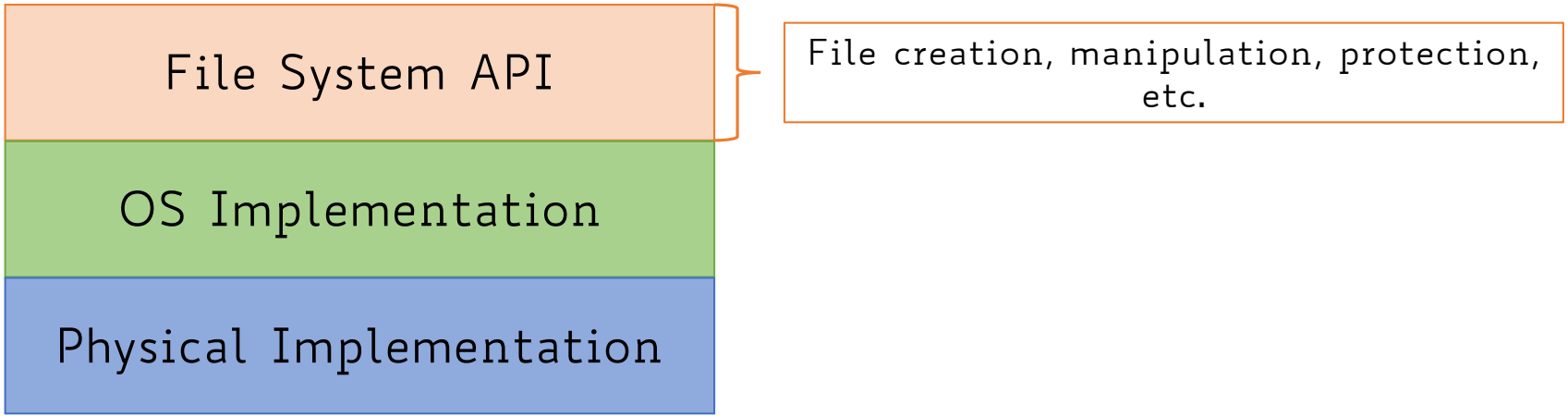
Feature	Port-Mapped I/O	Memory-Mapped I/O
Address space	Uses **separate I/O port space**	Uses **memory address space**
CPU instructions	`IN, OUT`	Normal loads/stores (`MOV`)
Hardware complexity	Separate I/O bus decoding	Requires memory bus decoding
Modern usage	Legacy (x86)	**Dominant** (PCI/PCIe)
Who assigns addresses?	Port numbers mostly **predefined historically**	OS/firmware assigns MMIO ranges based on device BARs

Part V: Storage Management

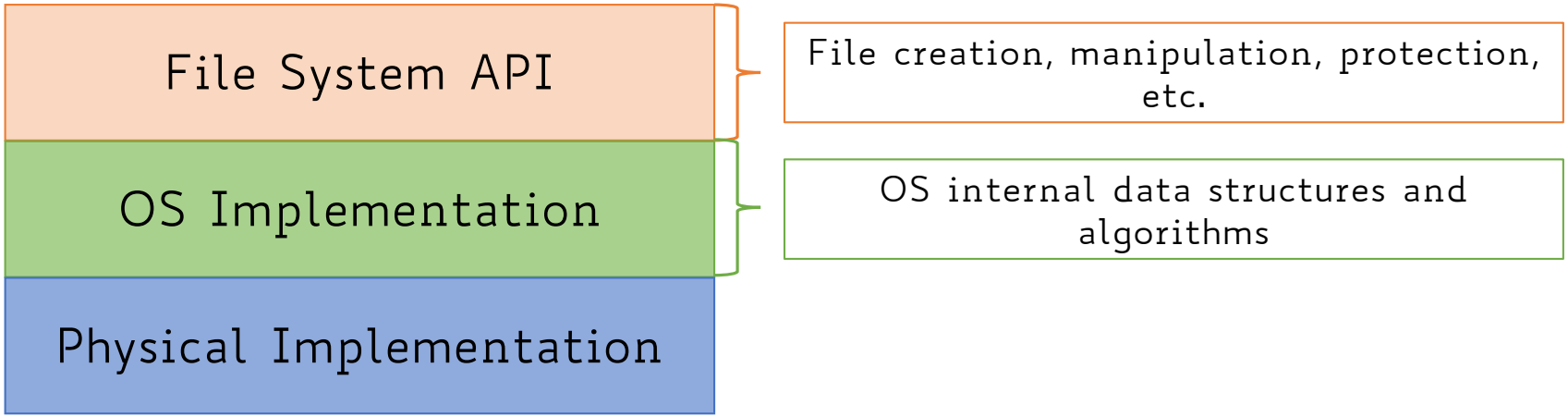
File System's Logical View



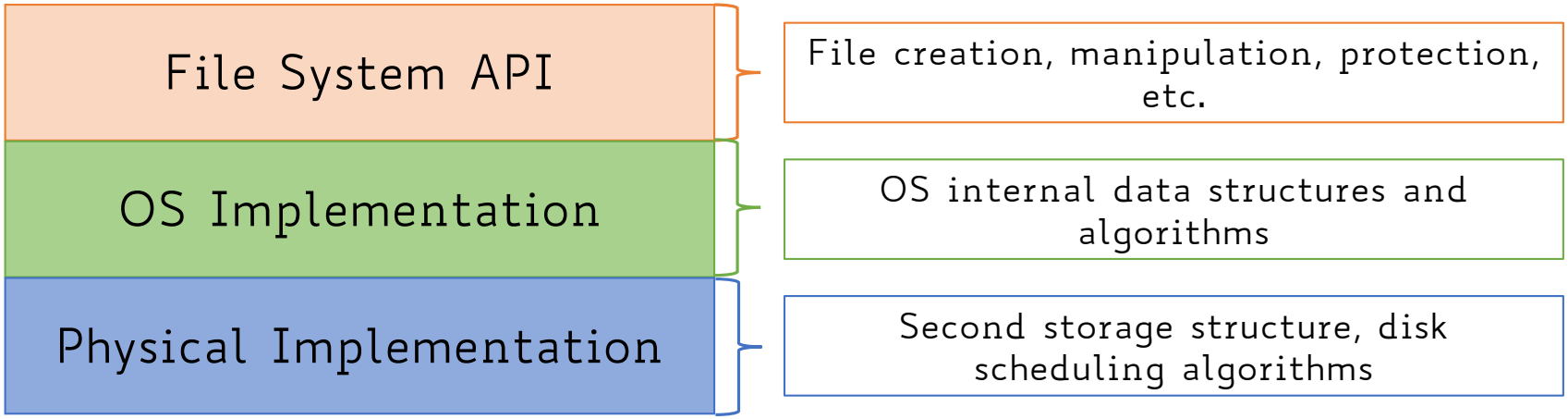
File System's Logical View



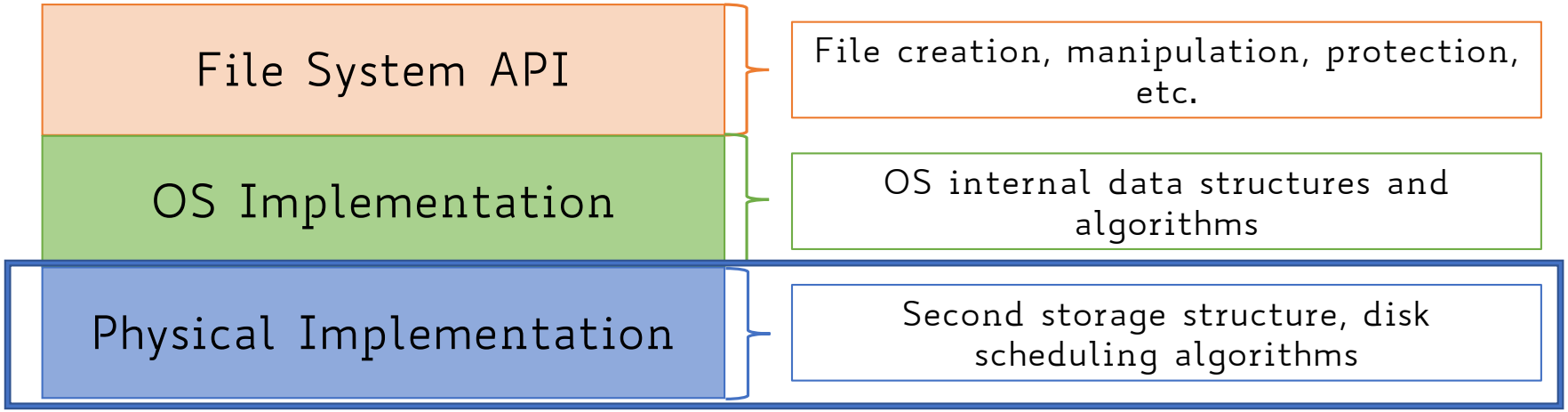
File System's Logical View



File System's Logical View



File System's Logical View



Overview of Mass-Storage Structure

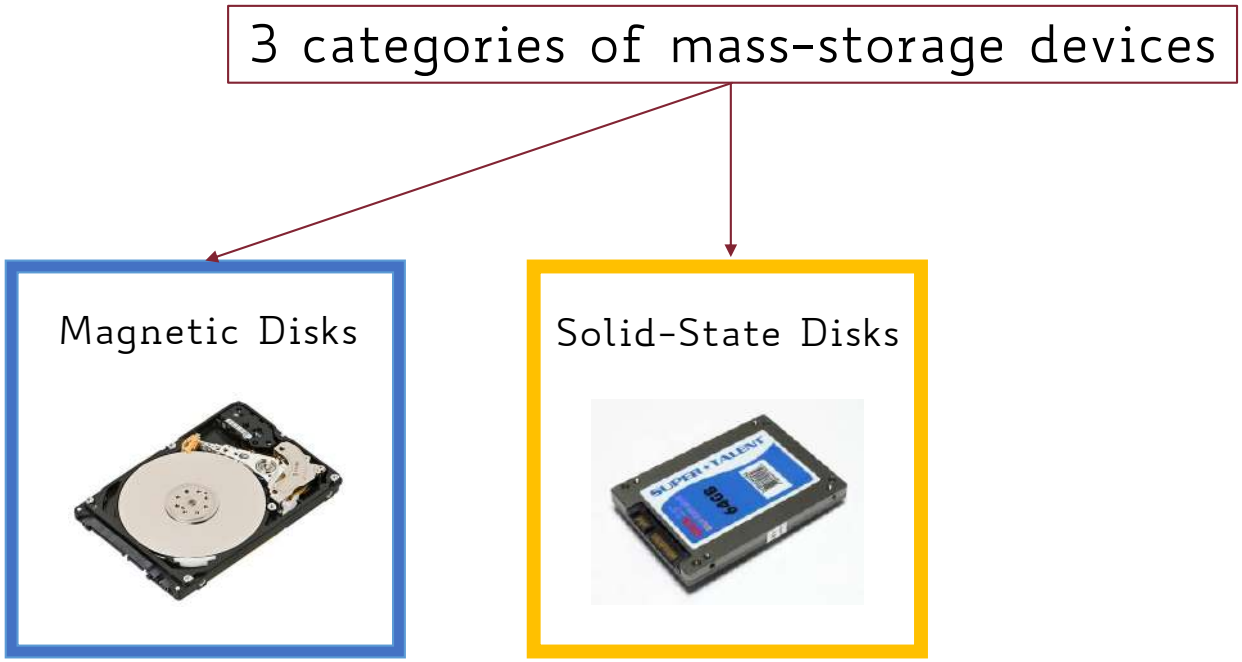
3 categories of mass-storage devices

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disks

Overview of Mass-Storage Structure



Overview of Mass-Storage Structure

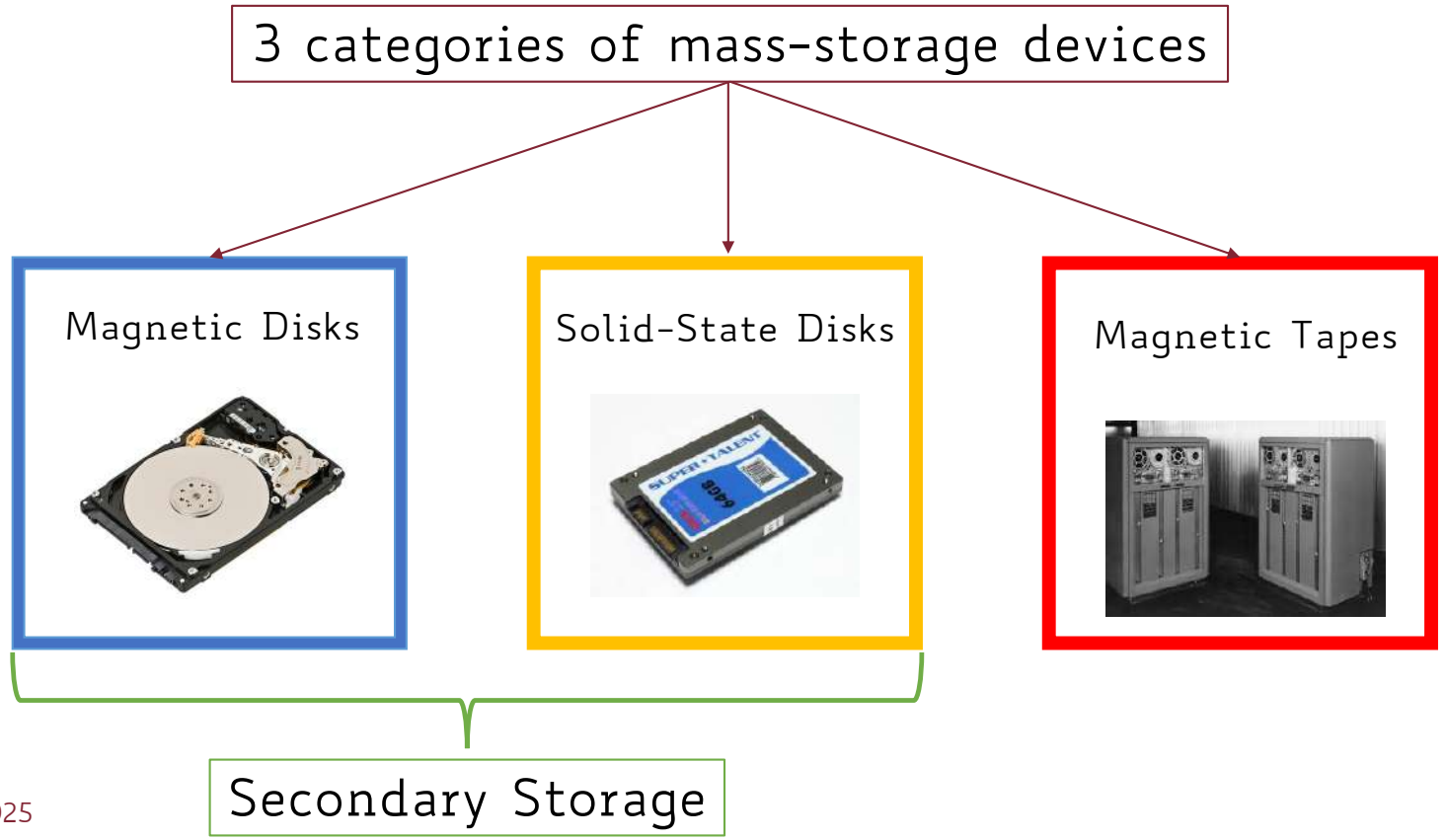
3 categories of mass-storage devices

Magnetic Disks

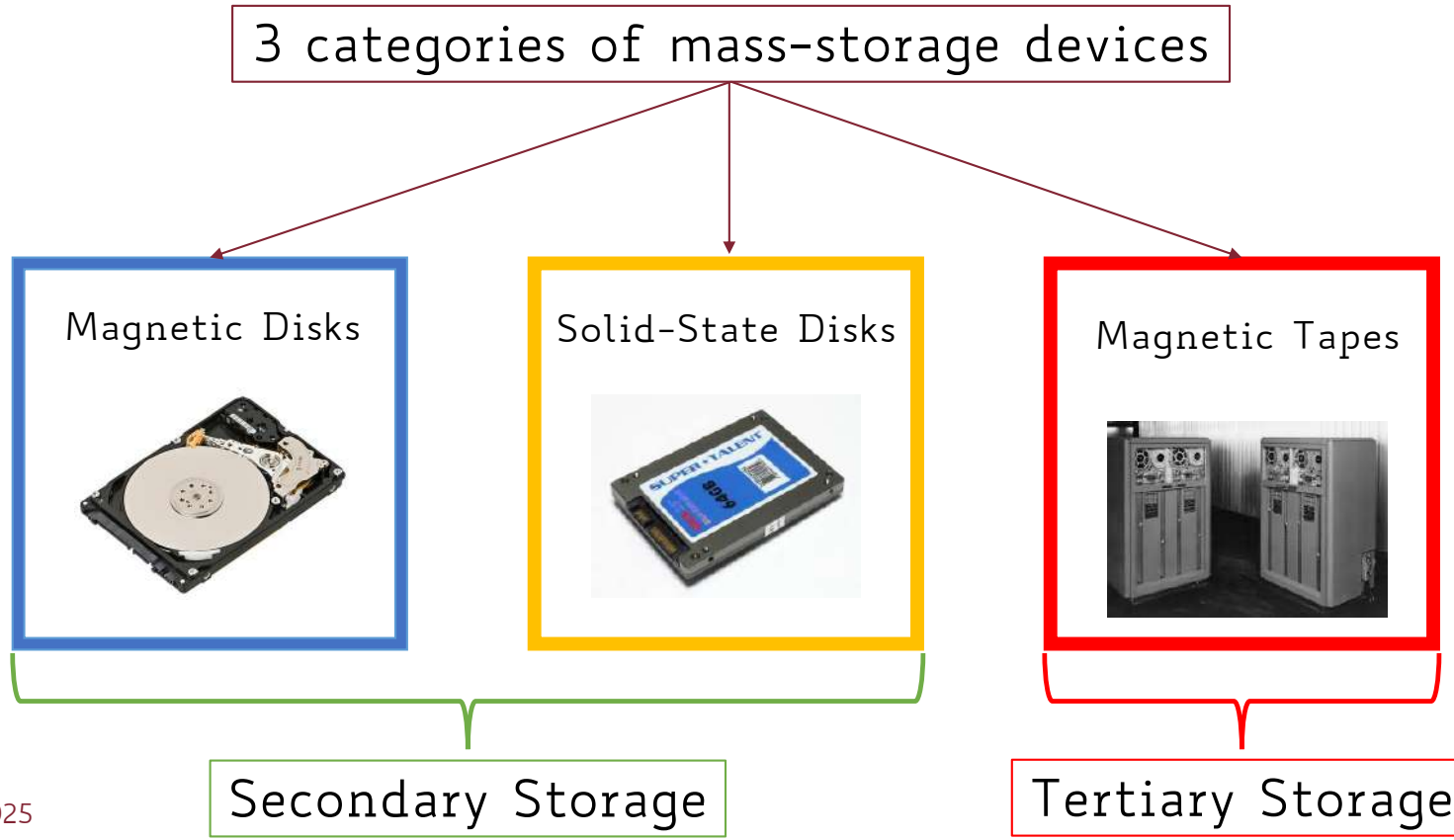
Solid-State Disks

Magnetic Tapes

Overview of Mass-Storage Structure



Overview of Mass-Storage Structure



Overview of Mass-Storage Structure

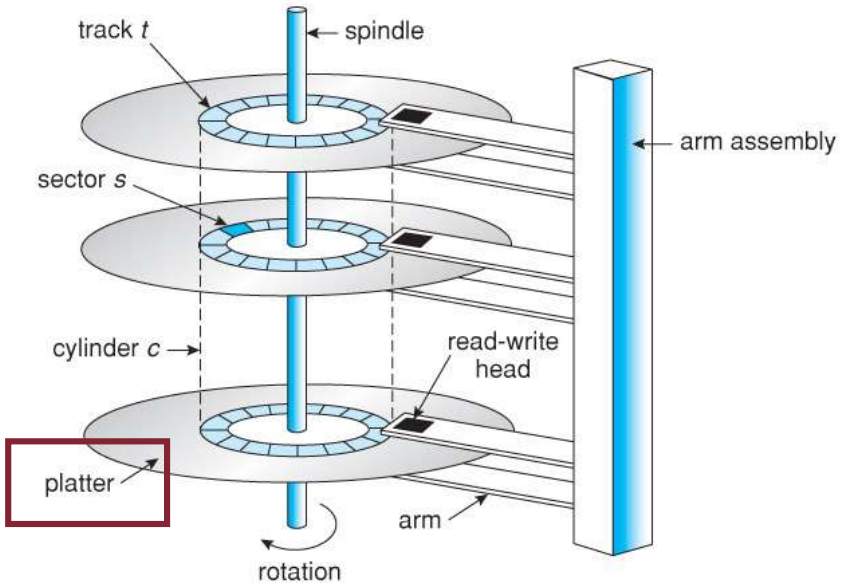
3 categories of mass-storage devices

Magnetic Disks

Solid-State Disks

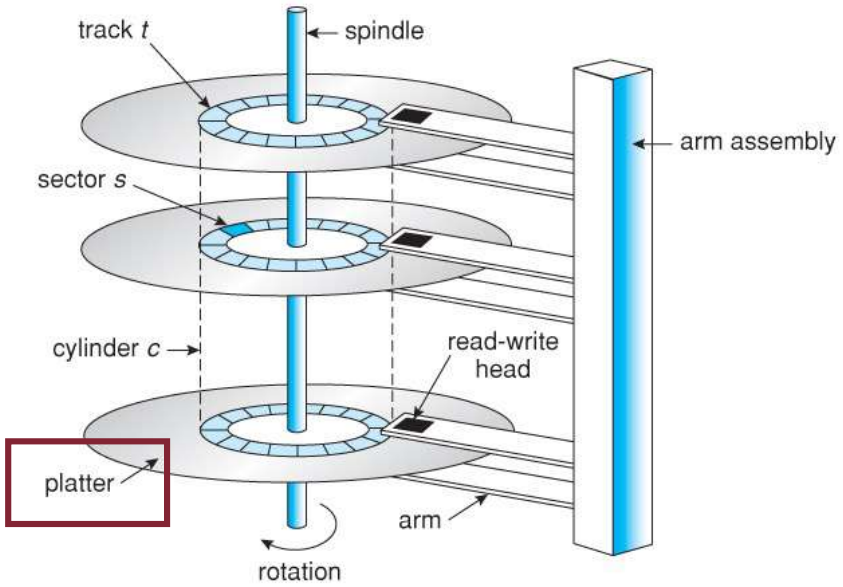
Magnetic Tapes

Magnetic Disks: Structure



One or more platters covered with magnetic media

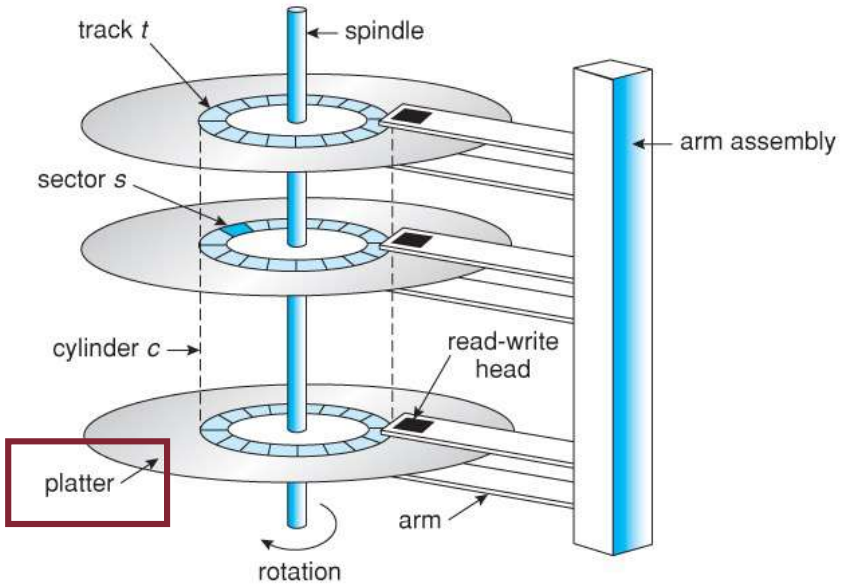
Magnetic Disks: Structure



One or more platters covered with magnetic media

Hard disk
rigid metal

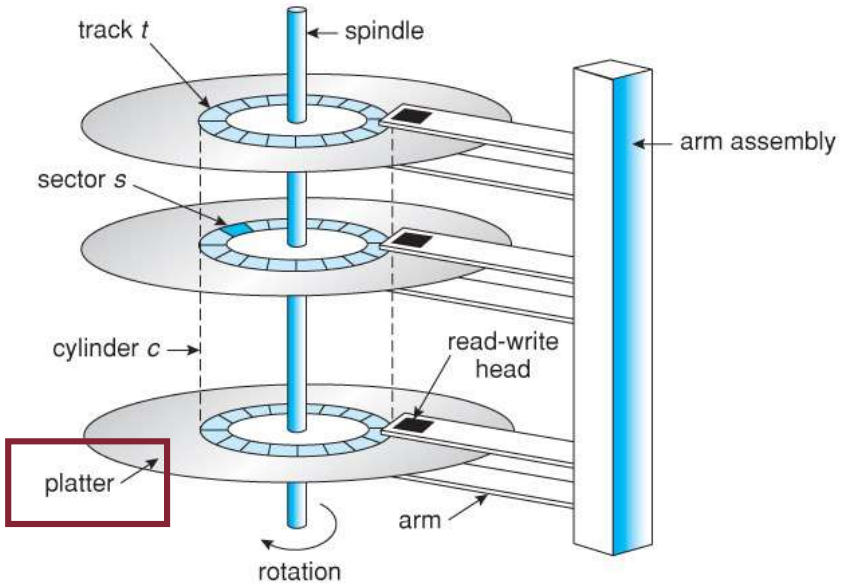
Magnetic Disks: Structure



One or more platters covered with magnetic media

- Hard disk: rigid metal
- Floppy disk: flexible plastic

Magnetic Disks: Structure



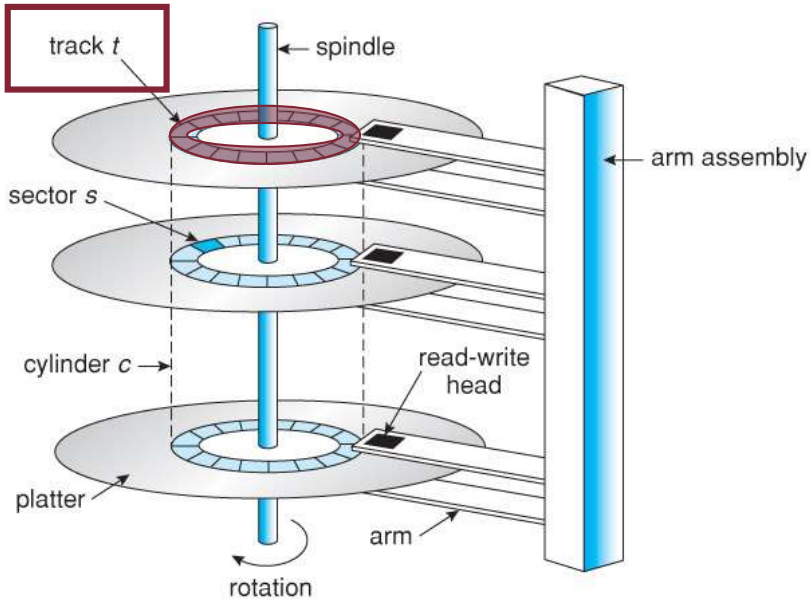
One or more platters covered with magnetic media

Hard disk
rigid metal

Floppy disk
flexible plastic

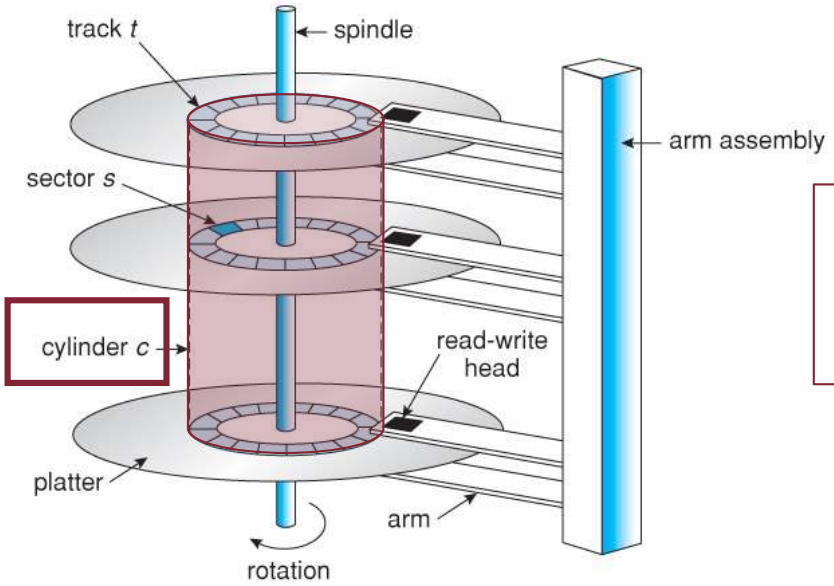
Each platter has 2 working surfaces

Magnetic Disks: Tracks and Cylinders



Each surface is divided into a number of concentric rings, called tracks

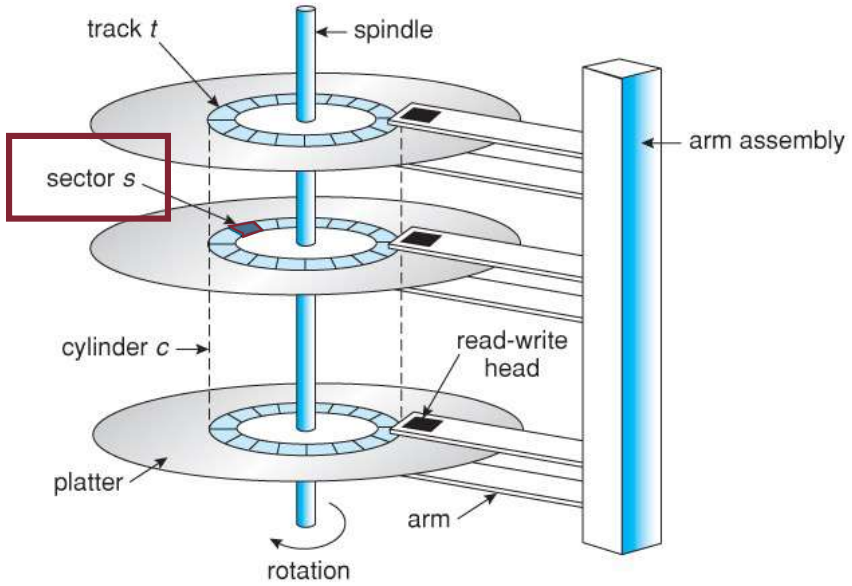
Magnetic Disks: Tracks and Cylinders



Each surface is divided into a number of concentric rings, called tracks

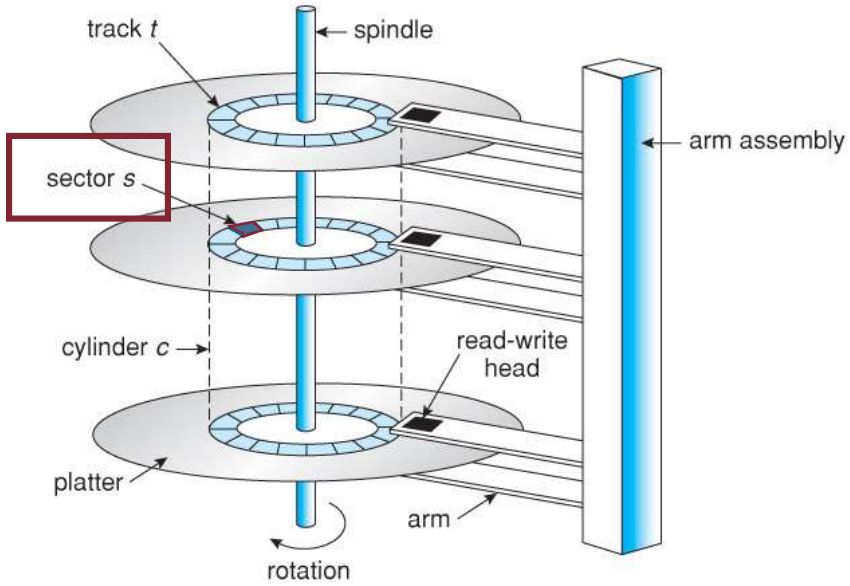
The set of all tracks that are the same distance from the edge of the platter is called a cylinder

Magnetic Disks: Sectors



Each track is further divided into sectors

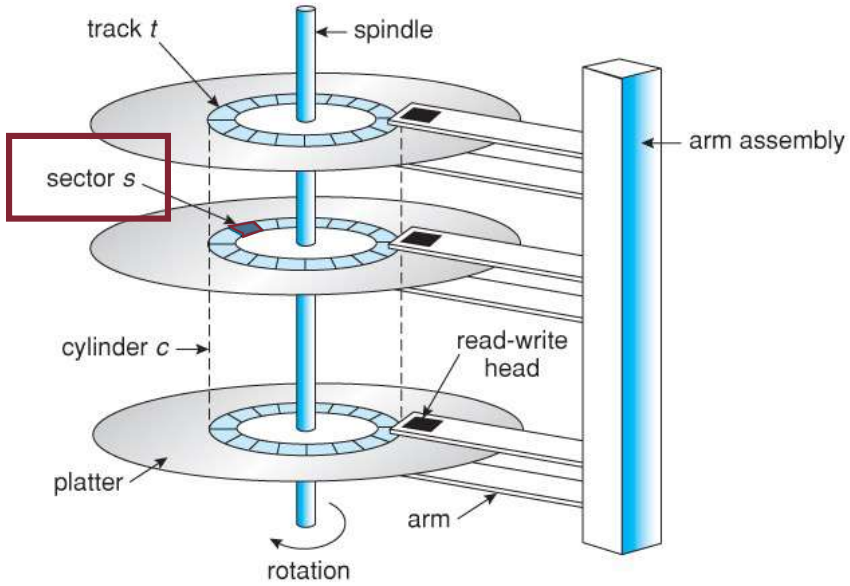
Magnetic Disks: Sectors



Each track is further divided into sectors

Each sector usually contains 512 B ÷ 4 KiB worth of data

Magnetic Disks: Sectors

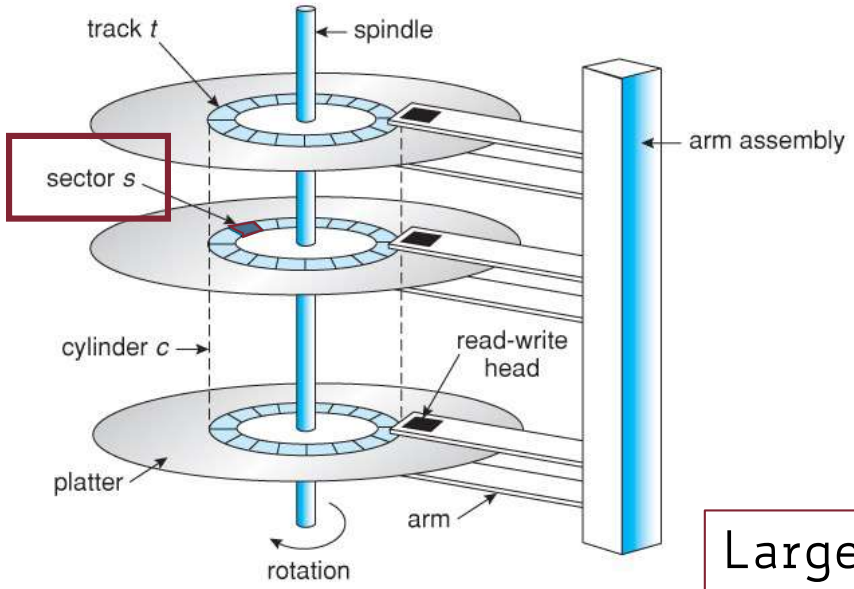


Each track is further divided into sectors

Each sector usually contains 512 B ÷ 4 KiB worth of data

Sectors also include a header and a trailer, and checksum information

Magnetic Disks: Sectors



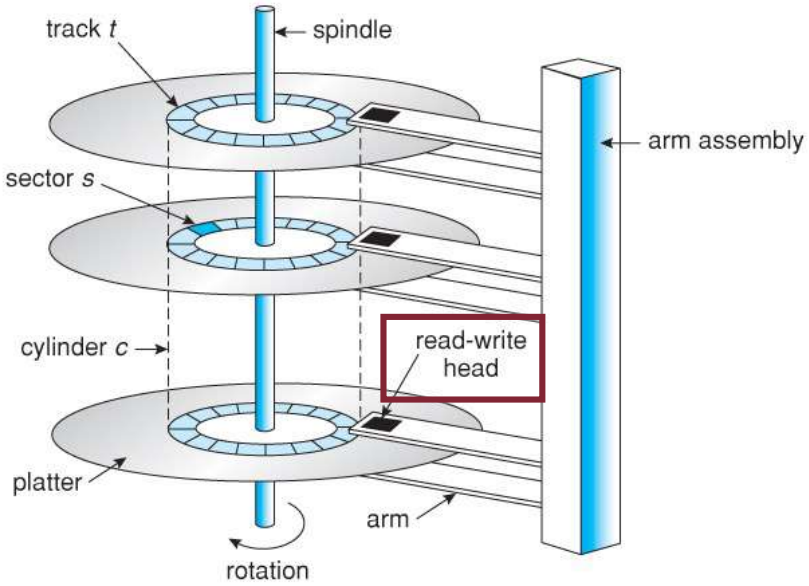
Each track is further divided into sectors

Each sector usually contains 512 B ÷ 4 KiB worth of data

Sectors also include a header and a trailer, and checksum information

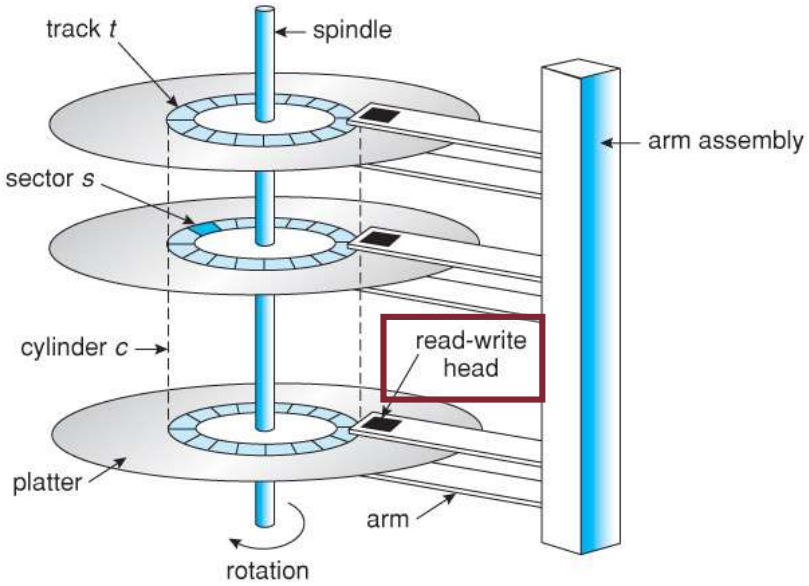
Larger sector sizes reduce the space wasted by headers and trailers, but increase internal fragmentation

Magnetic Disks: Heads



Data on hard drive is read by read-write heads

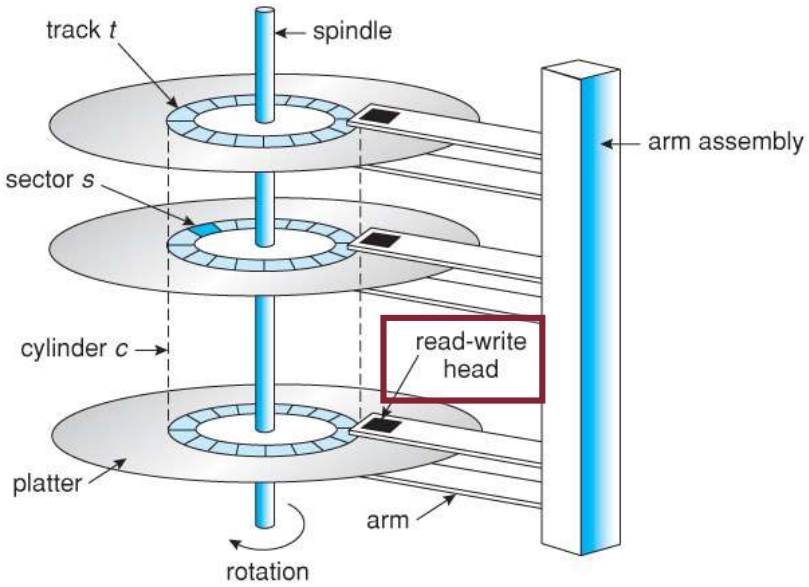
Magnetic Disks: Heads



Data on hard drive is read by read-write heads

Standard configuration uses one head per surface

Magnetic Disks: Heads

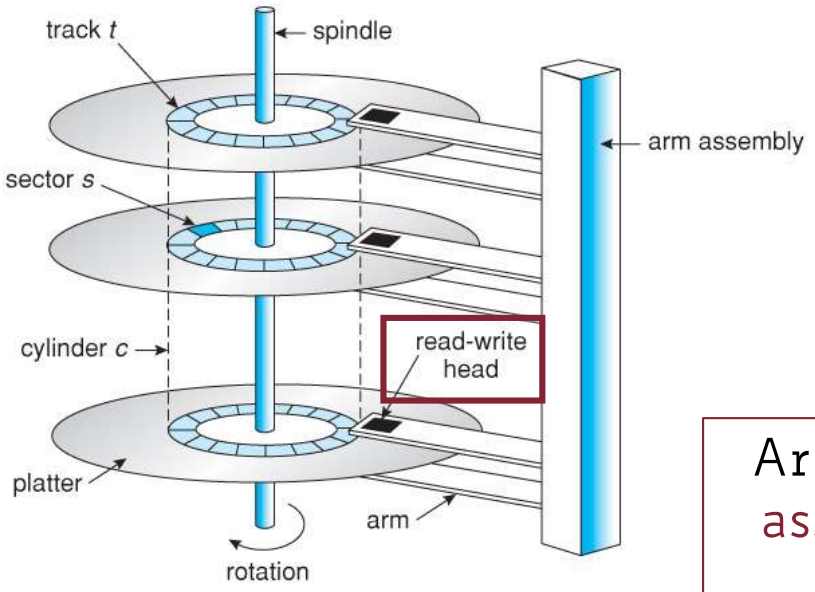


Data on hard drive is read by read-write heads

Standard configuration uses one head per surface

Each head is placed on a separate arm

Magnetic Disks: Heads



Data on hard drive is read by read-write heads

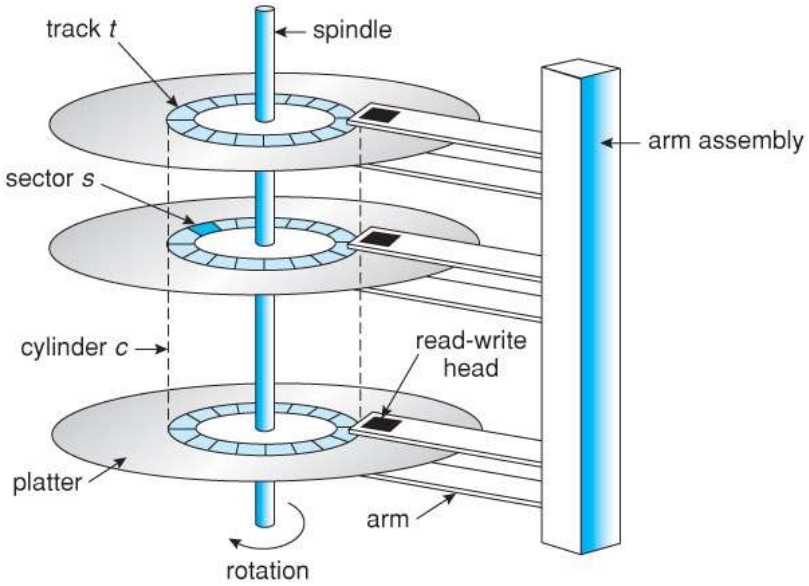
Standard configuration uses one head per surface

Each head is placed on a separate arm

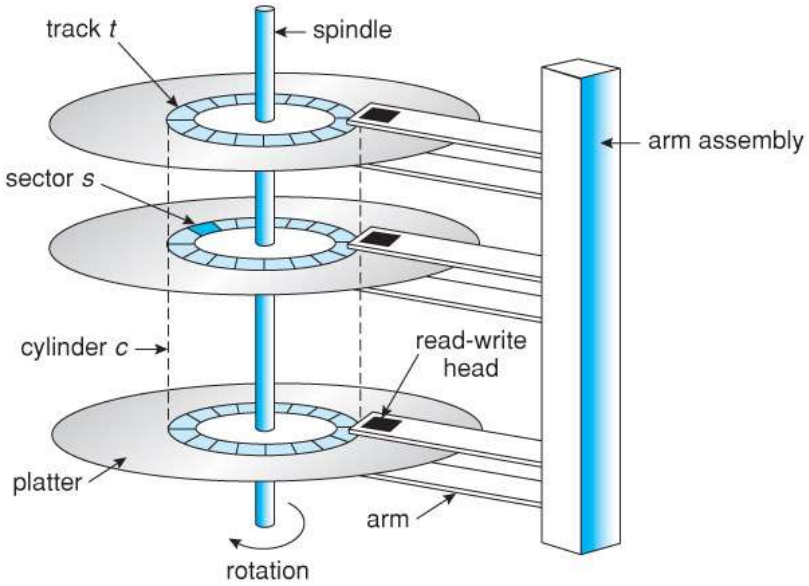
Arms are controlled by a common arm assembly moving simultaneously from one cylinder to another

Magnetic Disks: Storage Capacity

H = number of heads (working surfaces)



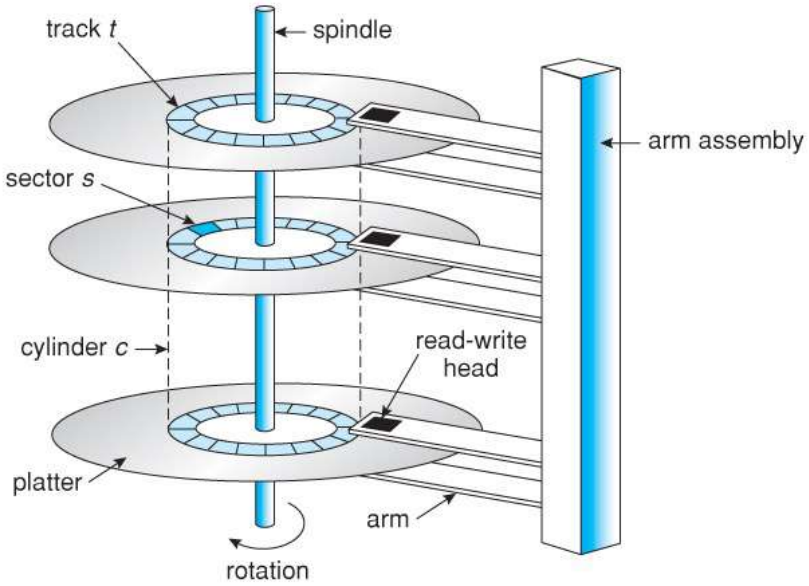
Magnetic Disks: Storage Capacity



H = number of heads (working surfaces)

T = number of tracks per surface

Magnetic Disks: Storage Capacity

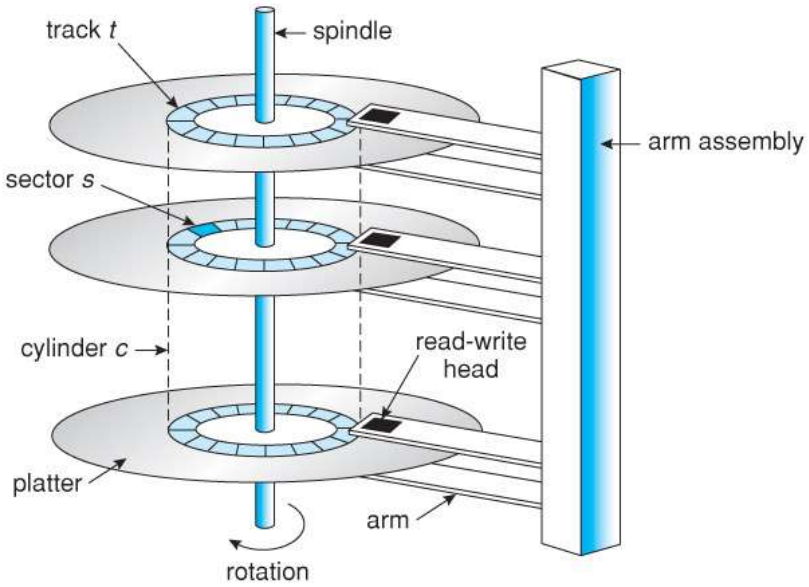


H = number of heads (working surfaces)

T = number of tracks per surface

S = number of sectors per track

Magnetic Disks: Storage Capacity



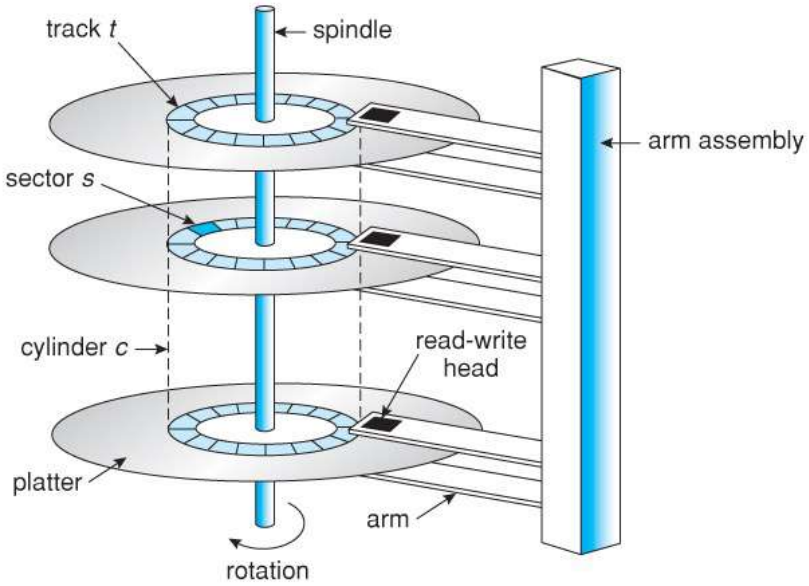
H = number of heads (working surfaces)

T = number of tracks per surface

S = number of sectors per track

B = number of bytes per sector

Magnetic Disks: Storage Capacity



H = number of heads (working surfaces)

T = number of tracks per surface

S = number of sectors per track

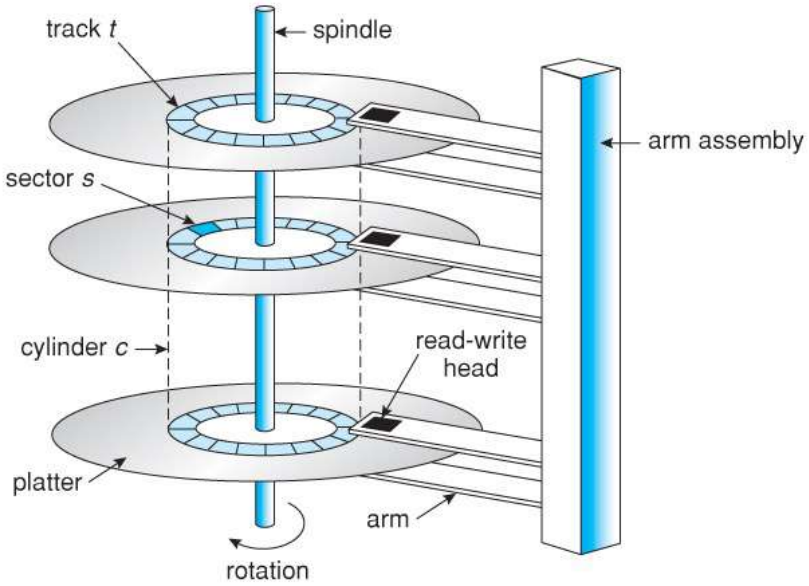
B = number of bytes per sector

C = H * T * S * B

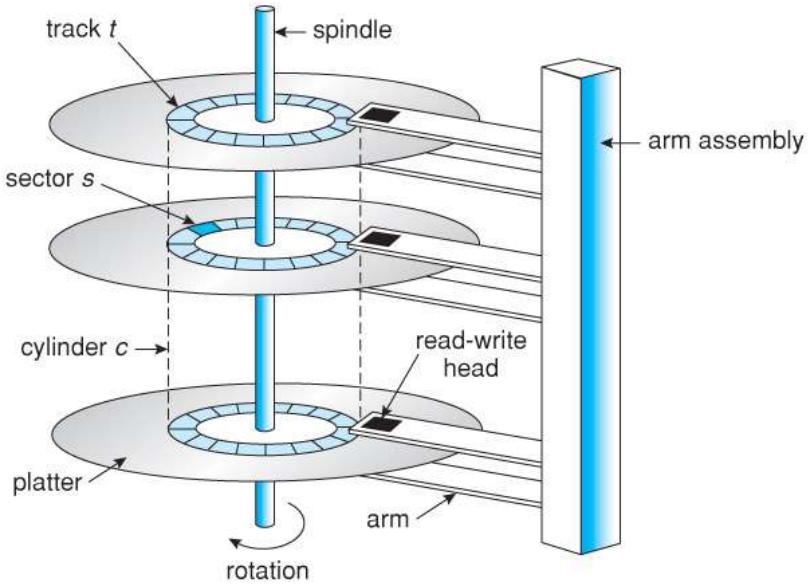
OVERALL CAPACITY

Magnetic Disks: Storage Capacity

Until the end of 1980s, every track had the same number of sectors with the same number of bits



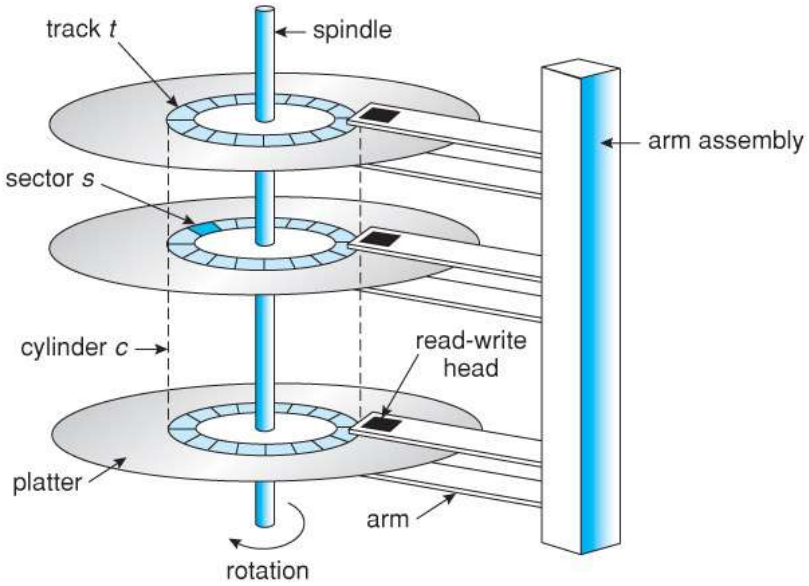
Magnetic Disks: Storage Capacity



Until the end of 1980s, every track had the same number of sectors with the same number of bits

Therefore, the bit density in the inner sectors was much higher than in the outer sectors

Magnetic Disks: Storage Capacity

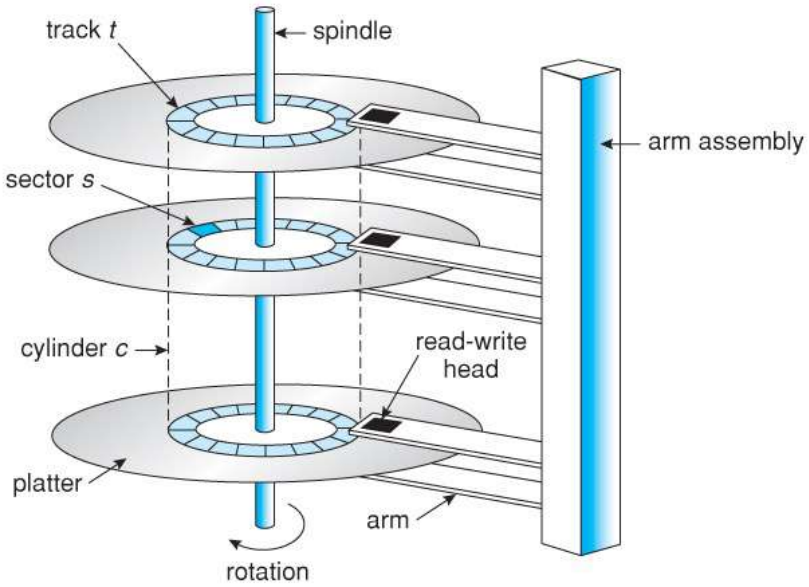


Until the end of 1980s, every track had the same number of sectors with the same number of bits

Therefore, the bit density in the inner sectors was much higher than in the outer sectors

Disk controllers have no "intelligence"

Magnetic Disks: Storage Capacity



Until the end of 1980s, every track had the same number of sectors with the same number of bits

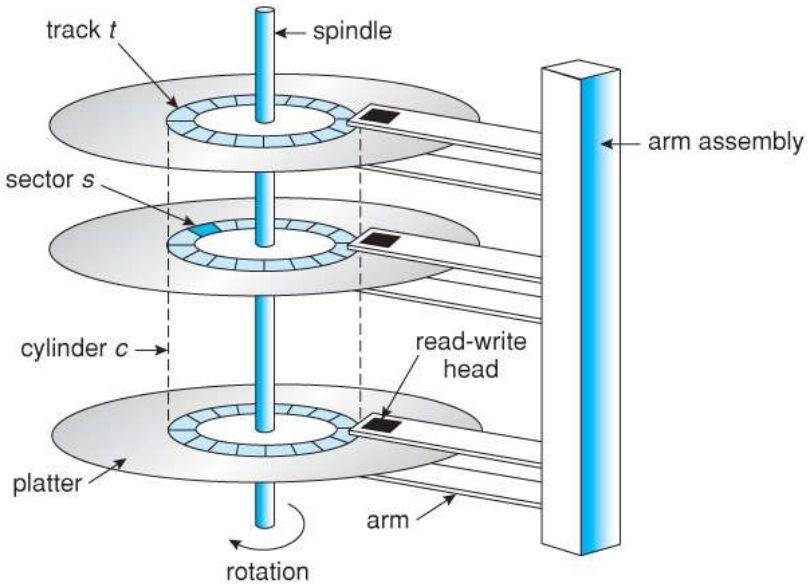
Therefore, the bit density in the inner sectors was much higher than in the outer sectors

Disk controllers have no "intelligence"

Drawbacks:

- The capacity of the disk was determined by the maximum bit density a controller could handle

Magnetic Disks: Storage Capacity



Until the end of 1980s, every track had the same number of sectors with the same number of bits

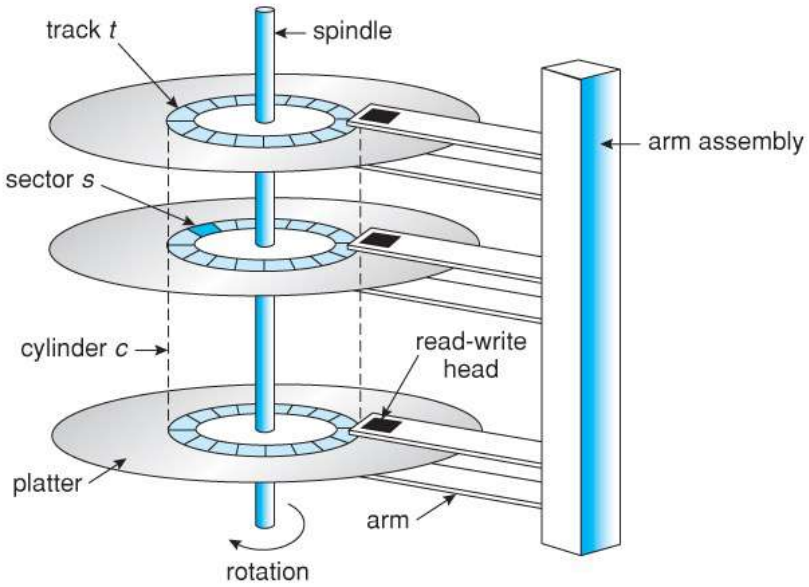
Therefore, the bit density in the inner sectors was much higher than in the outer sectors

Disk controllers have no "intelligence"

Drawbacks:

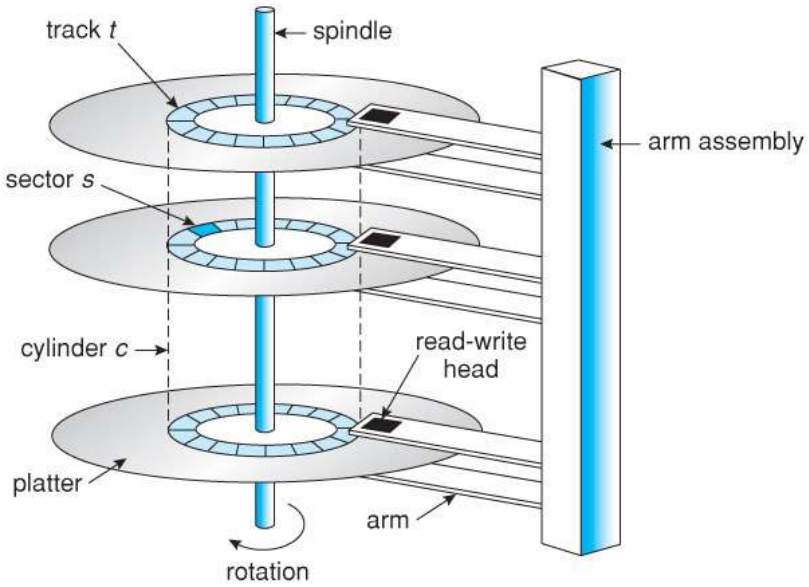
- The capacity of the disk was determined by the maximum bit density a controller could handle
- Different frequencies and timing from innermost to outermost tracks

Magnetic Disks: Storage Capacity



The number of sectors per track (S) varies with the radius of the track on the platter

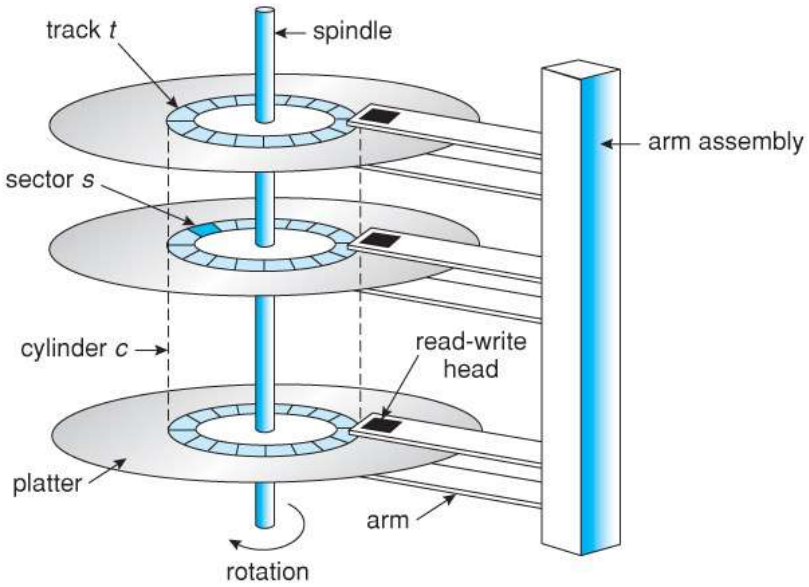
Magnetic Disks: Storage Capacity



The number of sectors per track (S) varies with the radius of the track on the platter

The outermost track is larger and can hold more sectors than the inner ones

Magnetic Disks: Storage Capacity

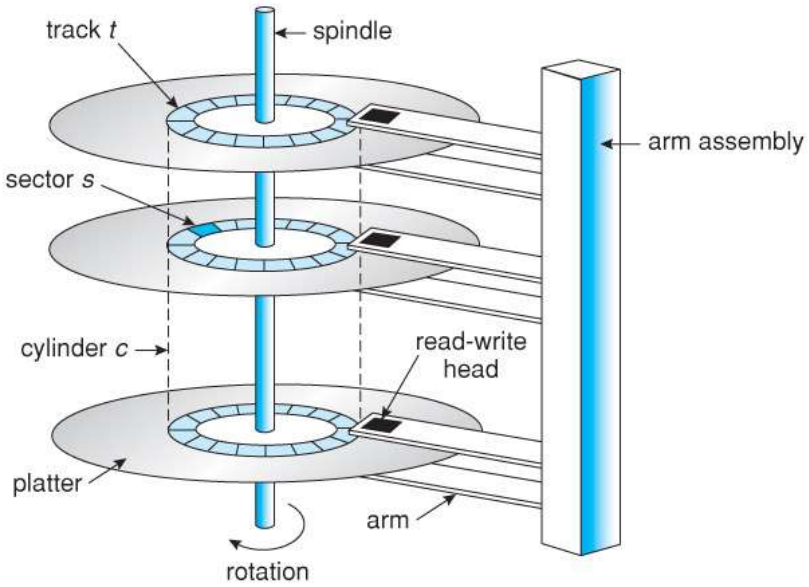


The number of sectors per track (S) varies with the radius of the track on the platter

The outermost track is larger and can hold more sectors than the inner ones

Tracks are divided into "zones" composed of multiple tracks

Magnetic Disks: Storage Capacity



The number of sectors per track (S) varies with the radius of the track on the platter

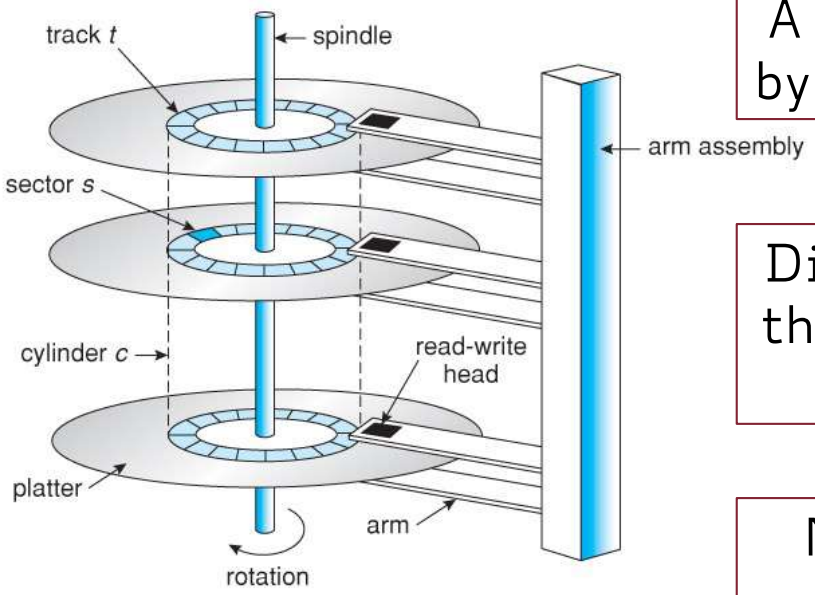
The outermost track is larger and can hold more sectors than the inner ones

Tracks are divided into "zones" composed of multiple tracks

Bit density within the same zone is constant

Zone Bit Recording (ZBR)

Magnetic Disks: (Logical) Referencing



A physical block of data is specified by the (head, cylinder, sector) number

Disk blocks are numbered starting at the outermost cylinder, identified by 0

Note that cylinder coincides with track

Magnetic Disks: Data Transfer

- The disk rotates at **constant angular speed** (e.g., 7200 rpm = 120 rps)

Magnetic Disks: Data Transfer

- The disk rotates at **constant angular speed** (e.g., 7200 rpm = 120 rps)
- Outer tracks spin **faster** than inner tracks (more sectors traversed in the same amount of time due to larger radius → more sectors per zone in ZBR)

Magnetic Disks: Data Transfer

- Data transfer from the disk to memory is made of **3 steps**:
  - positioning time (seek time or random access time)
  - rotational delay
  - transfer time

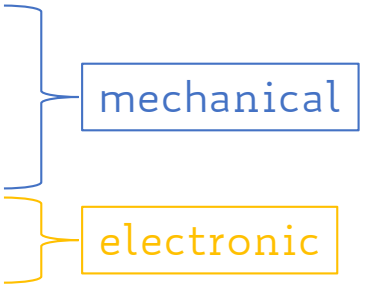
Magnetic Disks: Data Transfer

- Data transfer from the disk to memory is made of **3 steps**:
  - positioning time (seek time or random access time)
  - rotational delay
  - transfer time

mechanical

Magnetic Disks: Data Transfer

- Data transfer from the disk to memory is made of 3 **steps**:
  - positioning time (seek time or random access time)
  - rotational delay
  - transfer time



Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder

Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder
- Includes the time needed for the heads to settle

Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder
- Includes the time needed for the heads to settle
- Depends on how fast the hardware moves the arm

Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder
- Includes the time needed for the heads to settle
- Depends on how fast the hardware moves the arm
- Typically, the slowest step in the entire process

Bottleneck of overall disk data transfer

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head
- Can range from 0 up to one full revolution
  - 0 → the sector is already underneath the head
  - full revolution → the sector is the one before but in the opposite direction

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head
- Can range from 0 up to one full revolution
  - 0 → the sector is already underneath the head
  - full revolution → the sector is the one before but in the opposite direction
- On average, **0.5 revolutions** (r)
  - E.g., for a 7200 rpm (120 rps) disk this equals to 0.5 r/120 rps **~4 msec**

The second highest bottleneck

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory
- This is sometimes expressed as **transfer rate** (bandwidth) in bytes per second

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory
- This is sometimes expressed as **transfer rate** (bandwidth) in bytes per second

Data Transfer Time = Seek Time + Rotational Delay + Transfer Time

Sometimes the term transfer rate is used to refer to the overall data transfer time

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**
- Each logical block is the smallest addressable unit of transfer (e.g., **512 B**)

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**
- Each logical block is the smallest addressable unit of transfer (e.g., **512 B**)
- Blocks are mapped onto physical disk sectors (**512 B ÷ 4 KiB**)

Magnetic Disks: Structure

- Sector 0 is the first sector of the first track of the outermost cylinder

Magnetic Disks: Structure

- Sector 0 is the first sector of the first track of the outermost cylinder
- The mapping proceeds in order through that track

Magnetic Disks: Structure

- Sector 0 is the first sector of the first track of the outermost cylinder
- The mapping proceeds in order through that track
- Then through the rest of tracks in the same cylinder

Magnetic Disks: Structure

- Sector 0 is the first sector of the first track of the outermost cylinder
- The mapping proceeds in order through that track
- Then through the rest of tracks in the same cylinder
- Then through other cylinders (from the outermost to innermost)

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs
- Head crash may permanently damage the disk or even destroy it

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs
- Head crash may permanently damage the disk or even destroy it
- To avoid such a risk, disk heads are "parked" when the computer is turned off

Magnetic Disks: Interfaces

- Hard drives may be removable as floppy disks, and some are even hot-swappable
  - they can be removed while the computer is running

Magnetic Disks: Interfaces

- Hard drives may be removable as floppy disks, and some are even hot-swappable
  - they can be removed while the computer is running
- Disk drives are connected to the computer via the I/O bus

Magnetic Disks: Interfaces

- Hard drives may be removable as floppy disks, and some are even hot-swappable
  - they can be removed while the computer is running
- Disk drives are connected to the computer via the I/O bus
- Some of the common interface formats include:
  - Enhanced Integrated Drive Electronics (EIDE);
  - Advanced Technology Attachment (ATA) and Serial ATA (SATA);
  - Universal Serial Bus (USB);
  - Fiber Channel (FC);
  - Small Computer Systems Interface (SCSI)

Magnetic Disks: Controllers

- The **host controller** is at the computer's end of the I/O bus

Magnetic Disks: Controllers

- The **host controller** is at the computer's end of the I/O bus
- The **disk controller** is built into the disk itself

Magnetic Disks: Controllers

- The **host controller** is at the computer's end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)

Magnetic Disks: Controllers

- The **host controller** is at the computer's end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)
- Data is transferred between the magnetic surface and onboard **cache** by the disk controller

Magnetic Disks: Controllers

- The **host controller** is at the computer's end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)
- Data is transferred between the magnetic surface and onboard **cache** by the disk controller
- Finally, data is transferred from that cache to the host controller and the motherboard memory at electronic speeds

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay
- To minimize data transfer time from disk we need to minimize those

Minimize Data Transfer Time

- Smaller disks → lower seek time, since arms have to travel smaller distance

Minimize Data Transfer Time

- Smaller disks → lower seek time, since arms have to travel smaller distance
- Fast-spinning disks → lower rotational delay

Hardware Optimization

Minimize Data Transfer Time

- How can the OS help minimize data transfer time?
- Schedule disk operations so as to minimize head movement
- Lay out data on disk so that related data are located on close tracks
- Place commonly-used data on a specific portion of the disk
- Pick carefully the block size contained on each sector:
  - Too small → more seeks are needed to transfer the same amount of data
  - Too large → more internal fragmentation and space wasted

Summary

- Hard disks are slow devices compared to CPUs (and main memory)
- Manage those device efficiently is crucial
- Minimize seek and rotational delay on magnetic disks
- HW optimizations are limited → OS needs to take the lead here!