ECS518U - Operating Systems
Week 5

Disks and I/O

Tassos Tombros

Outline

- Review from Week 4
  - Lab feedback
  - FAT, inodes
- **Disks**
  - Physical aspects and properties
  - Performance: Latency, bandwidth
- **I/O**
  - Memory-mapped I/O
  - Direct Memory Access
  - Scheduling of disk requests
- Reading:
  - **Tanenbaum:** Chapter 5, sections 5.1 - 5.4
  - **Stallings:** Chapter 11, sections 11.1 – 11.5

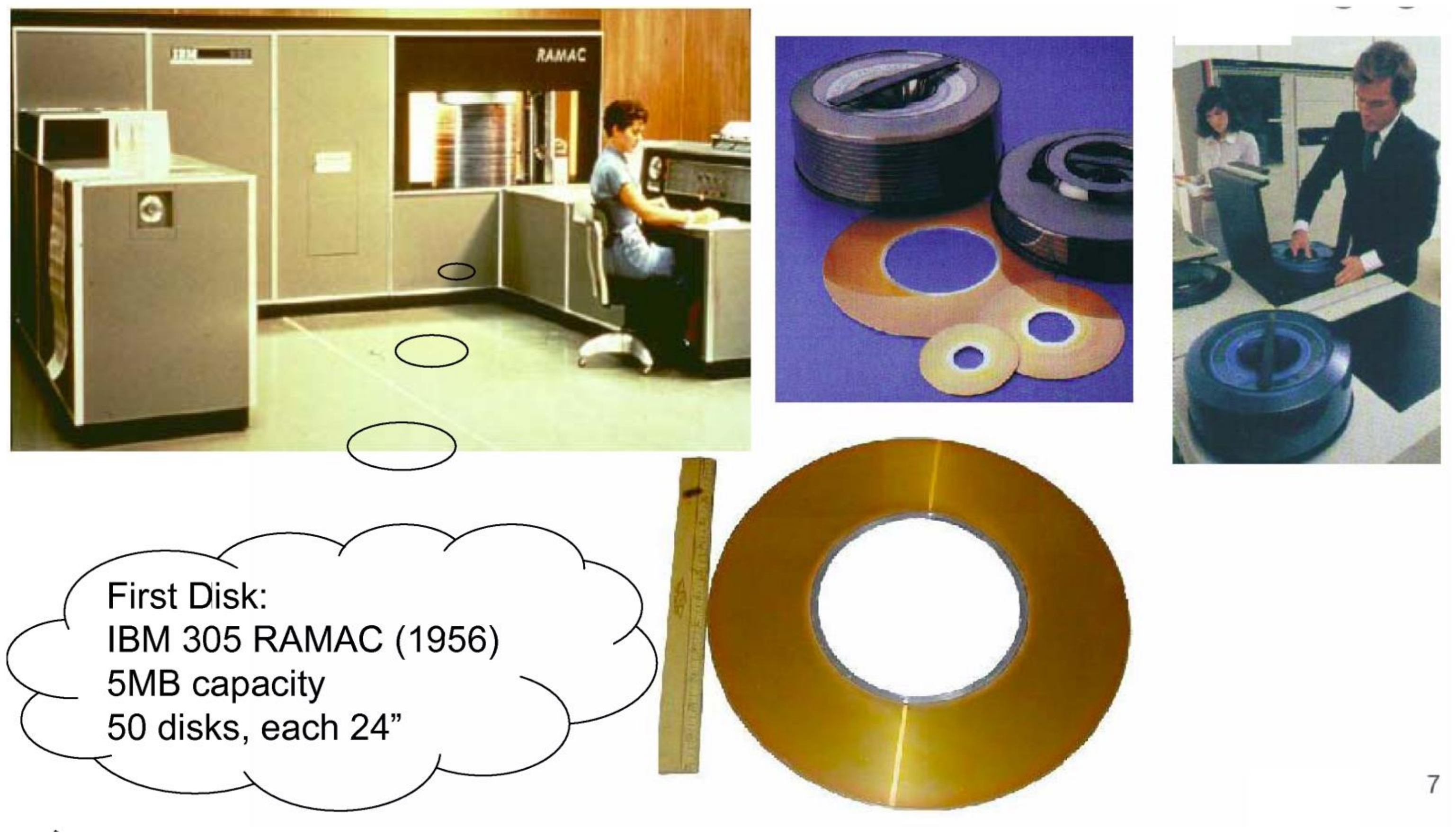
Things you will learn today

- How are **magnetic disk drives** organised and accessed
  - Tracks, sectors, cylinders
  - **Latency and bandwidth** (seek, rotate, transfer times)
  - and a little bit about **Solid State Drives (SSDs)**
- **Main components and methods of I/O**
  - Device controllers, bus hierarchy
  - Synchronous and asynchronous I/O
  - How the CPU communicates with devices (Memory-mapped I/O)
  - How data is transferred to/from device controllers (Direct Memory Access (DMA))
- **Scheduling of disk requests**
  - Why it is necessary
  - Specific scheduling algorithms

Context

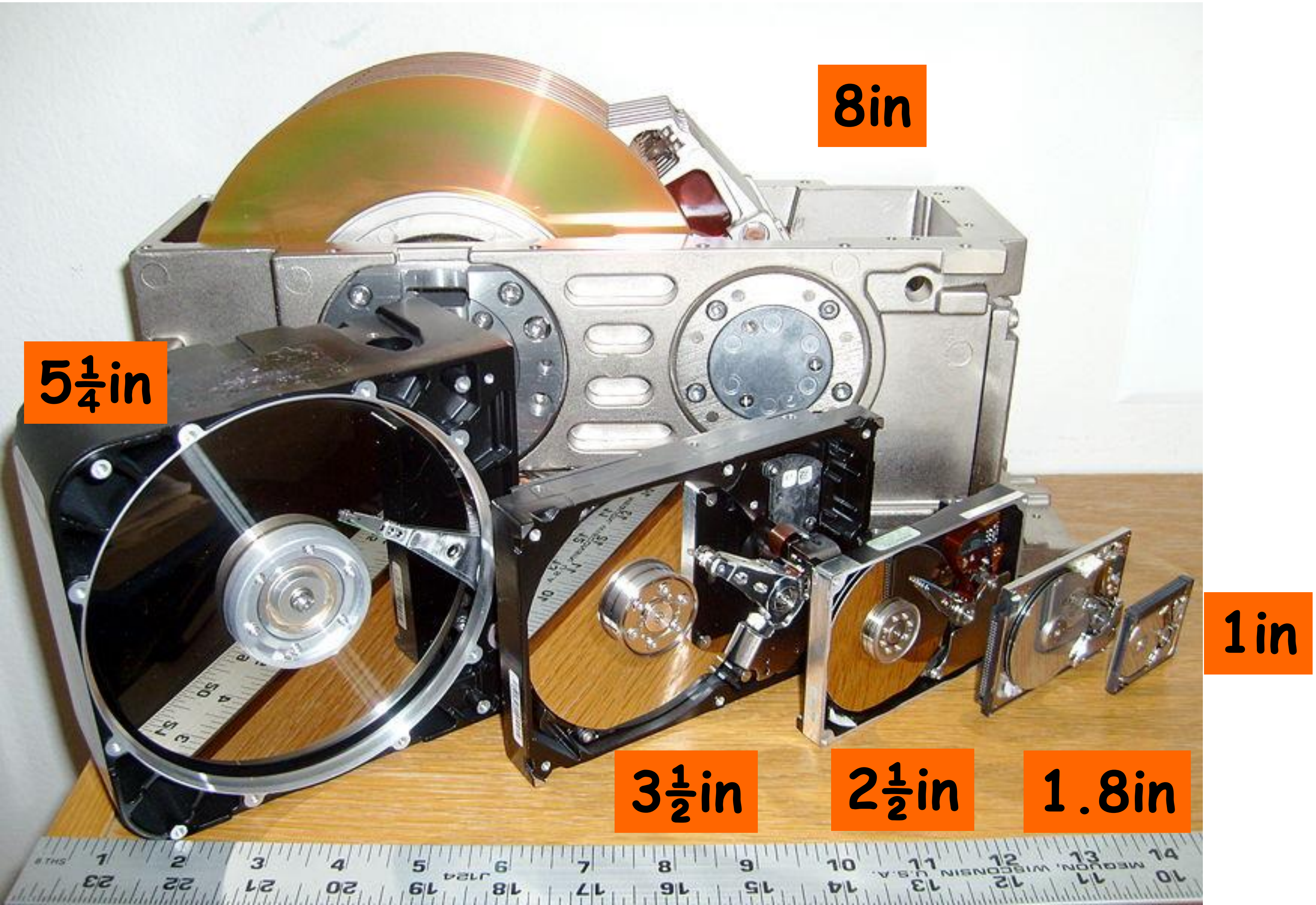
- **Files** as we see them and use them are really stored on some storage device
  - HDDs, SSDs, etc.
- **File systems** (Week 4) hide the complexity of storage devices from us
- Our files are really bits of data (most likely) spread over many different parts of the **storage device**
  - e.g. in different sectors of a HDD
  - there is **no guarantee** our files will be stored in contiguous locations on the disk!!!
- Some properties of storage device hardware make I/O operations challenging for the OS
  - What properties in particular? What is the challenge?
    - Time it takes for disk to rotate, time it takes for read/write head to move into place
    - These operations are very slow compared to CPU speed

Physical aspects: Disk Size

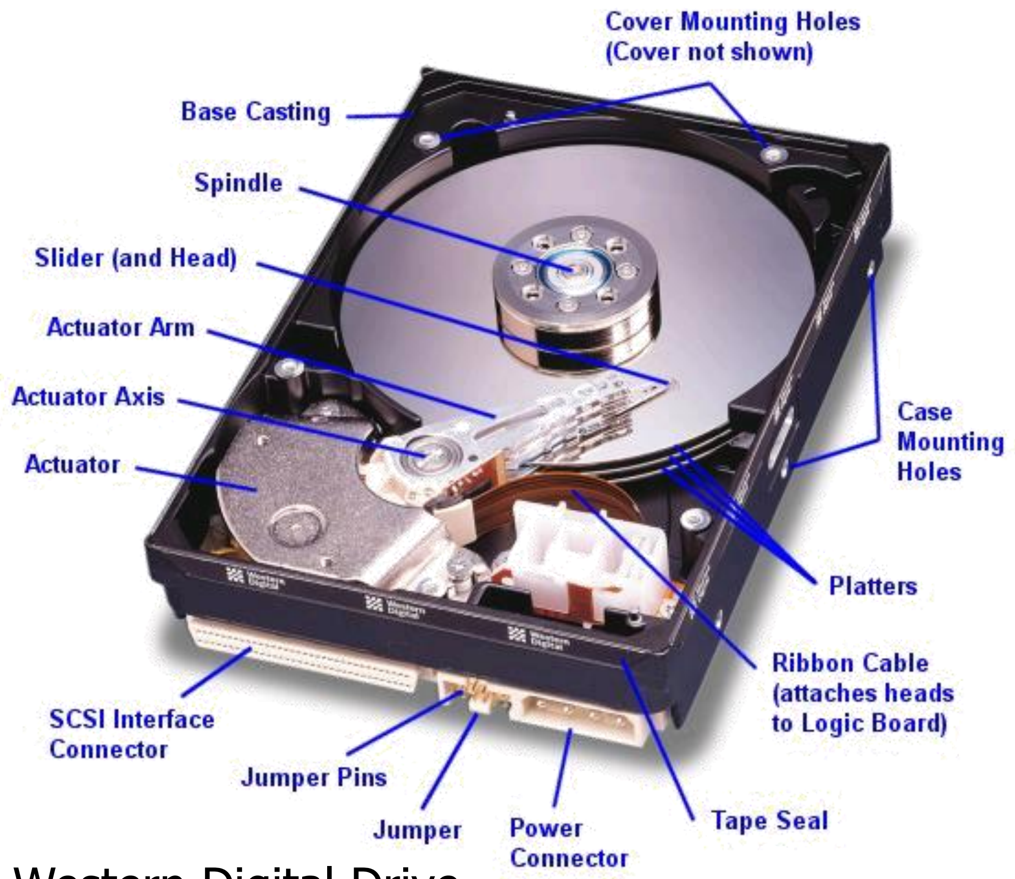


- 5MB, 50 disks, 24in diameter

Physical aspect: Disk Size



The (magic) Hard Disk



Western Digital Drive

http://www.storagereview.com/

Historical for comparison:

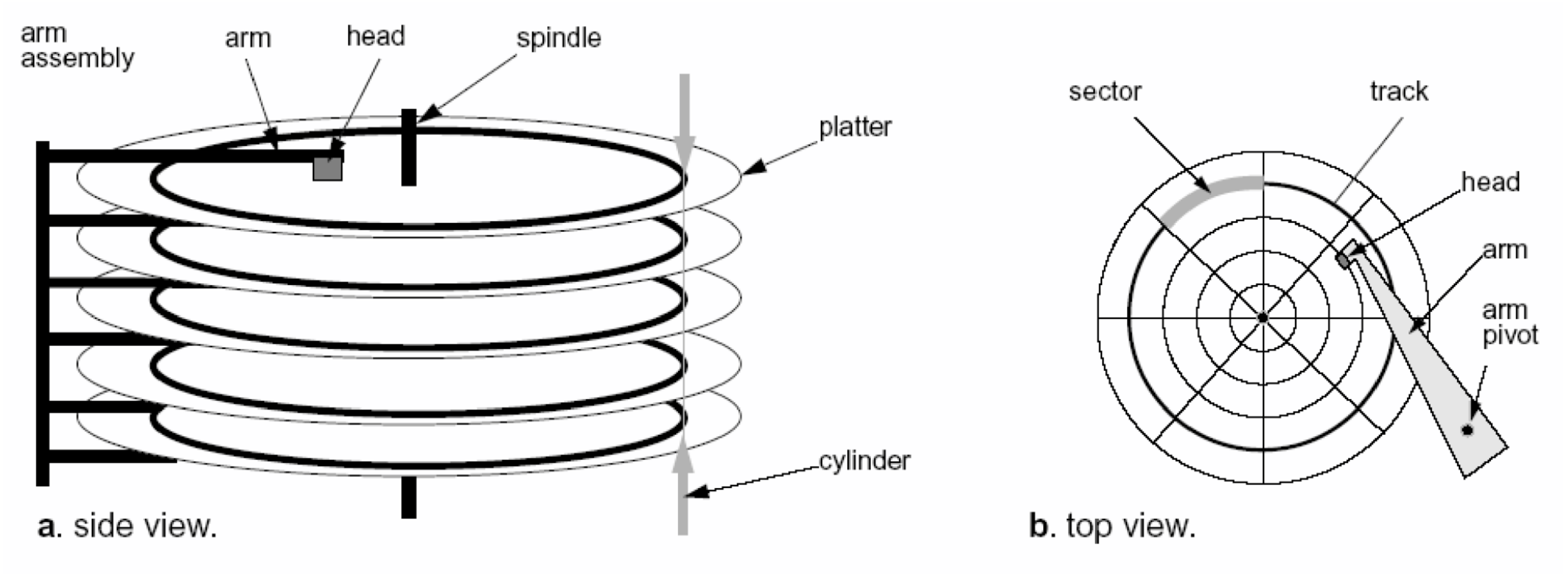
IBM Personal Computer/AT (1986)

- 30 MB hard disk - $500
- 30-40ms seek time
- 0.7-1 MB/s (est.)

Today's disks?

Tracks, Cylinder, Sector

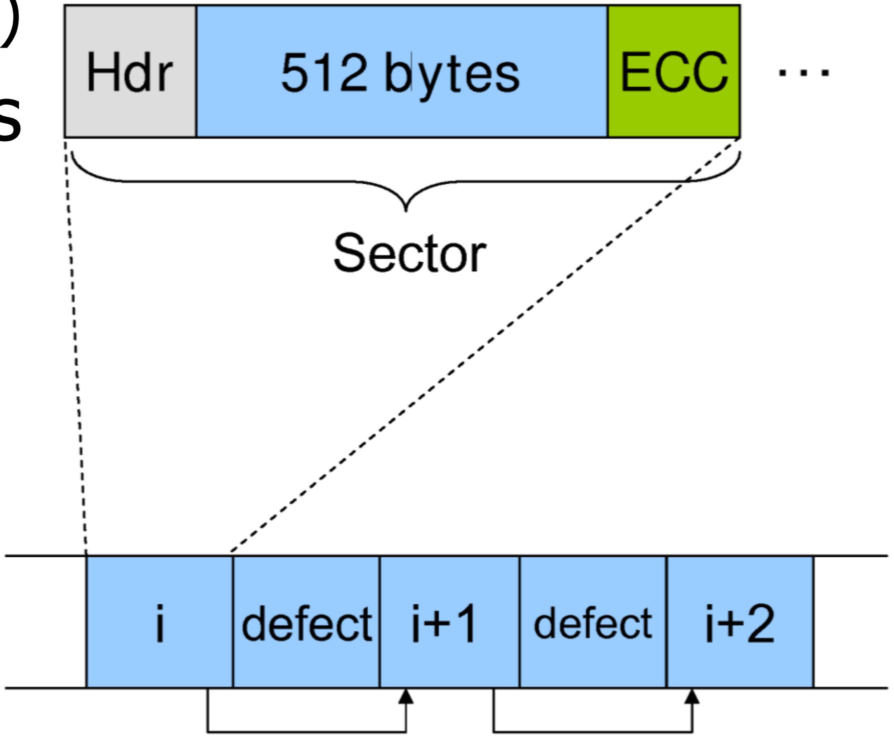
- **Track**: ring around the disk surface
- **Cylinder**: stack of tracks that are the same distance from the spindle
- **Sector**: segment (cake slice) of track
  - Sector is the unit of transfer



a. side view. b. top view.

Sectors (the "cake slices")

- Each **sector** has
  - Header, ID
  - Space for data
  - Error Correcting Code (ECC)
- A sector can be marked as 'bad'
  - (e.g. due to physical damage)
- **Sector is min unit of transfer**
  - Block transfer



Key Concepts

Very slow in terms of CPU speed times

- Hardware factors
  - **Disk rotation**
    - Disk rotates at a fixed number of revs/min
  - Disk head movement (**seeking**)
    - Head moves across tracks
    - Reading of a sector depends on rotation and seek
- Performance factors
  - **Latency**
    - Time it takes to start a requested transfer
    - Measured in ms (typical 5 ms)
  - **Data rate / bandwidth**
    - Rate to transfer data
    - MBytes/sec (typical 100s of Mbytes/sec)

Performance: Latency

- **Seek time**
  - The time it takes to position head over the track/cylinder
  - Typical 3.5-9.5 ms
- **Rotational delay**
  - Time we need to wait for beginning of sector to rotate beneath the head
  - 6,000 RPM = 100 rotations/sec ➔ 5 ms **average** delay
- **Access time** = (seek time) + (rotational delay)
- **Transfer delay**
  - Time to read data
  - 1 sector (0.5 Kbytes) @ 100 Mbytes/sec ➔ 5 μs

Performance: bandwidth

- The actual bandwidth that we are going to get **depends** on how we read sectors off the disk
- Small transfers from different parts of the disk:
  - Access time (**seek + rotation**) dominates
  - 1 sector / 10 ms ➔ 50 KBytes / sec
  - 0.05 % of rated bandwidth
- Large sequential transfers to achieve high bandwidth
  - Sequence of sectors (one after another) as they pass under the head of the disk

60 Years' Progress …

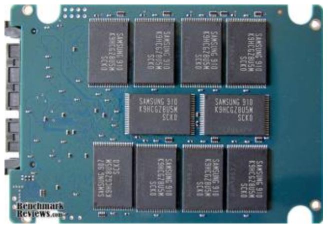
	IBM RAMAC (1956)	Current disks (2016)	Difference
Capacity	5MB	over 1TB	over 200,000
Areal Density	2 Kbits/in^2	over 1 Tbits/in^2	over 500,000,000
Price/GB	\$3,000,000	\$0.02	1 / 150,000,000
RPM	1,200	7,200	6
Seek Time	600 ms	10 ms	1 / 60
Data Rate	10 KB/s	over 100 MB/s	over 10,000
Power	5000 W	2 W	1 / 2,500
Weight	1000 Kg	100 g	1 / 10,000

Key points: **Latency is hard to improve** (delays due to physical limitations)

Bandwidth can improve (move more data at once – make data buses wider)

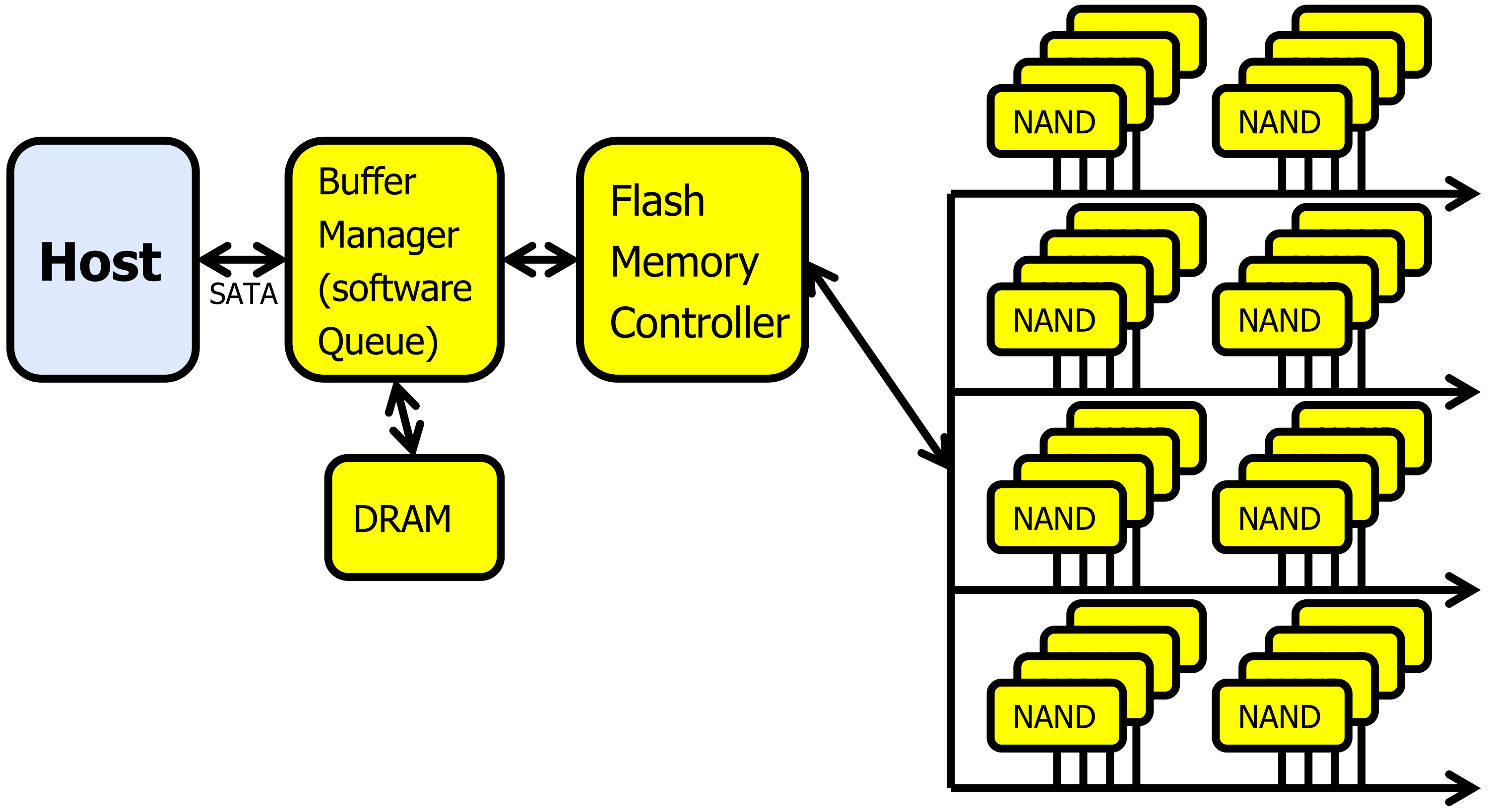
Flash Memory – Solid State Disks (SSDs)

- Flash memory replacing disks in some applications
  - Phones, cameras, etc.
- Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 “pages” per memory block
  - **Trapped electrons** distinguish between 1 and 0

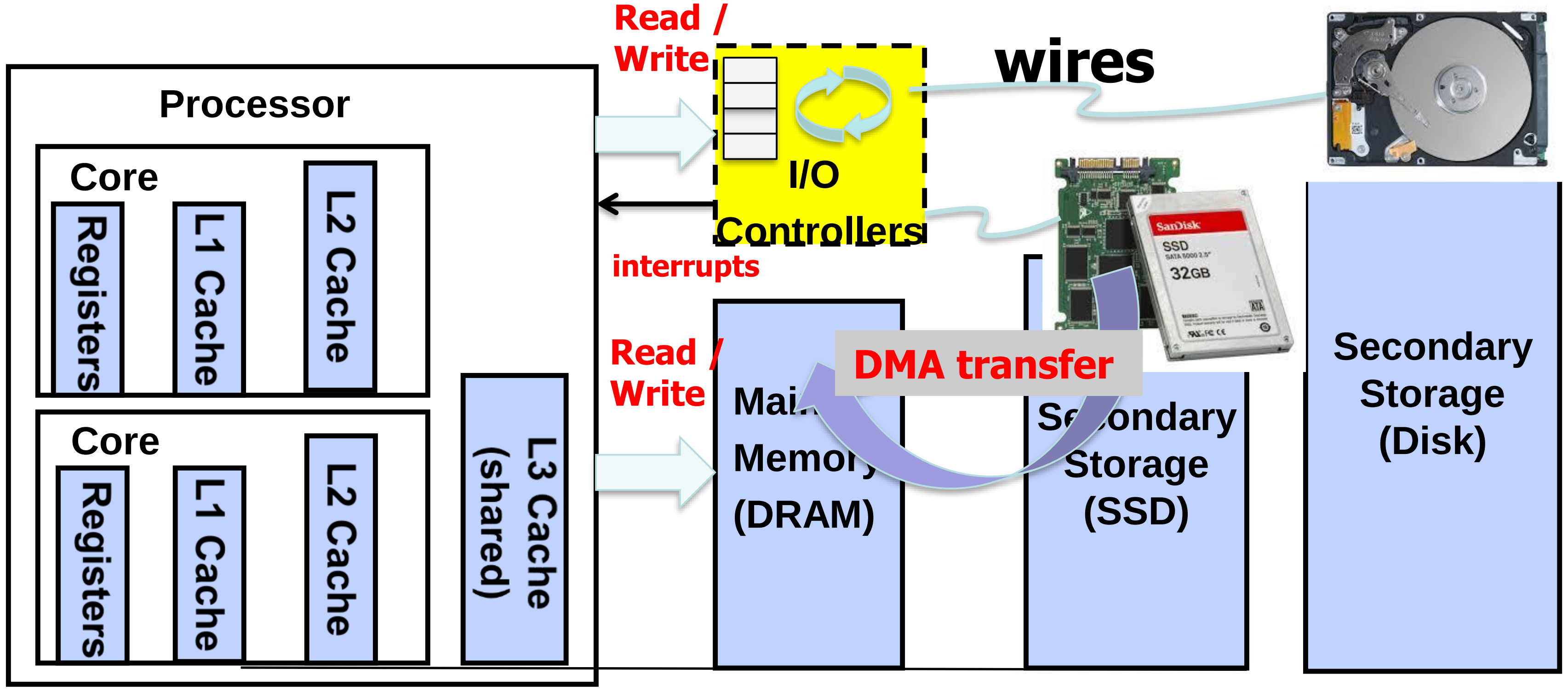


- **Pros**
  - No moving parts, low power, light weight
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
- **Cons**
  - More expensive
  - Write is slower (block erase)
  - Limited life (trapped electrons misbehave) – you lose storage capacity after some time

Solid State Disks Architecture

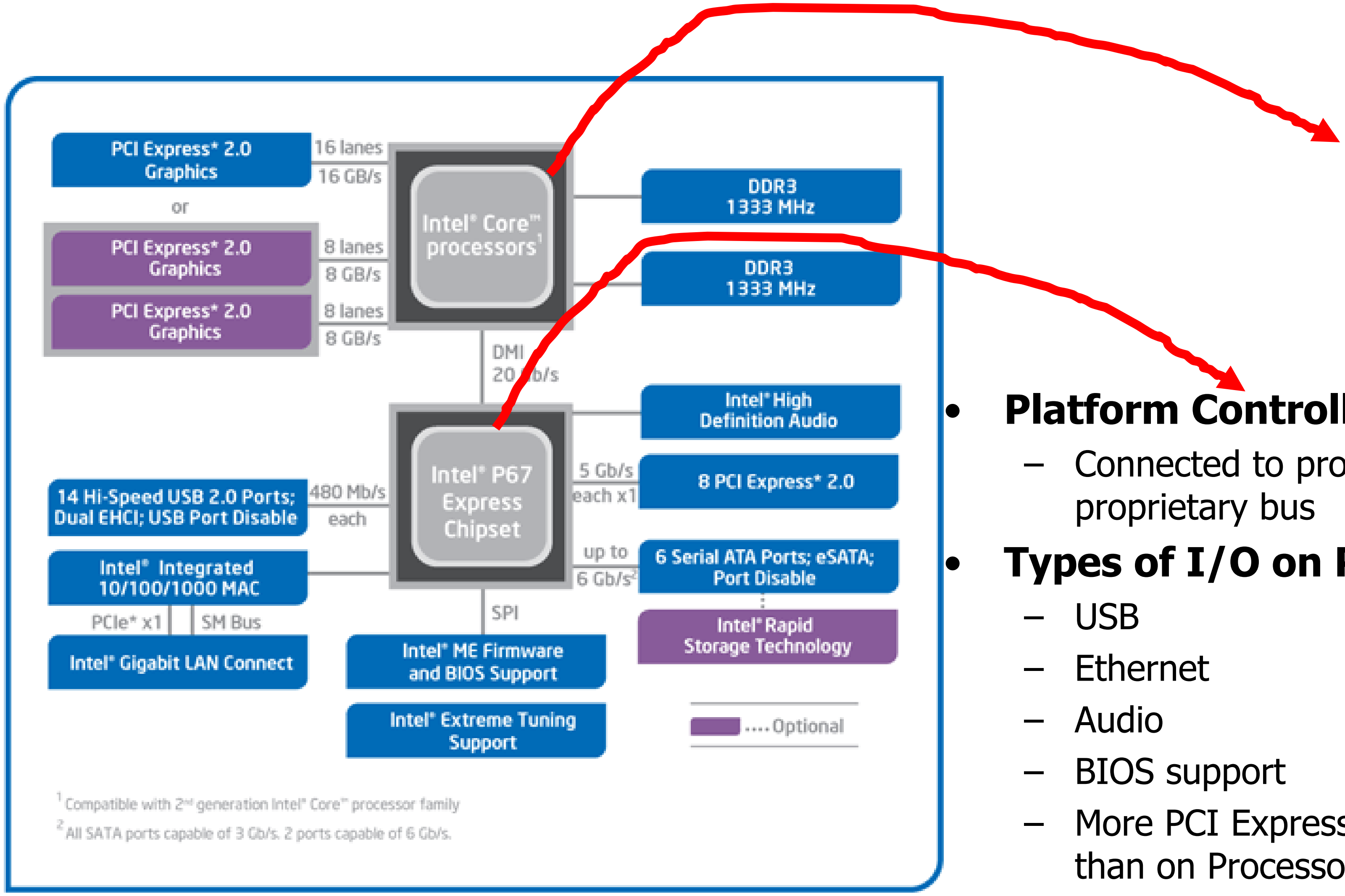


I/O in a picture



- I/O devices are supported by I/O Controllers
- Processors access them by reading and writing I/O registers as if they were memory
  - Write commands and arguments, read status and results

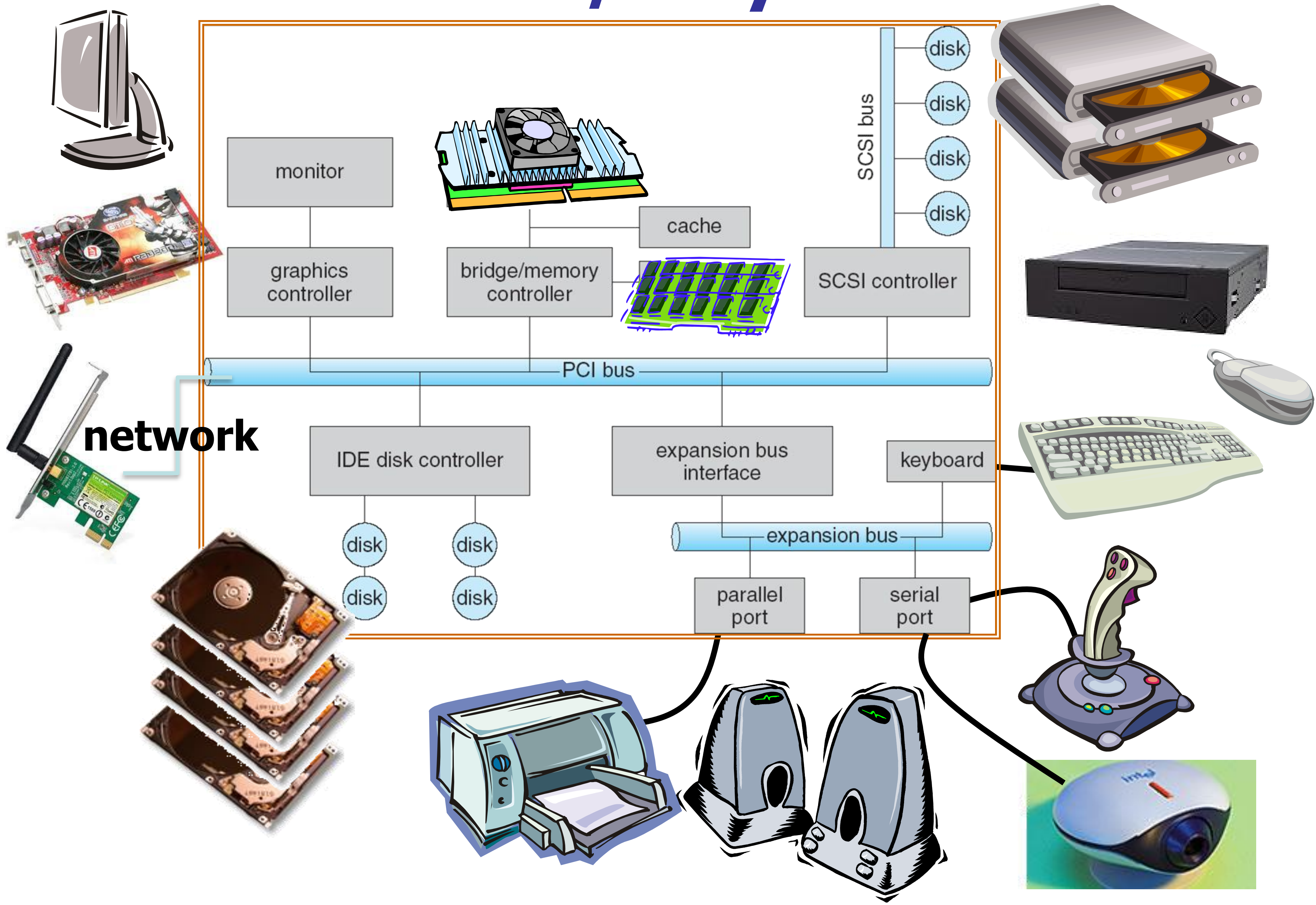
Example: Intel Sandy Bridge Platform Controller Hub (PCH)



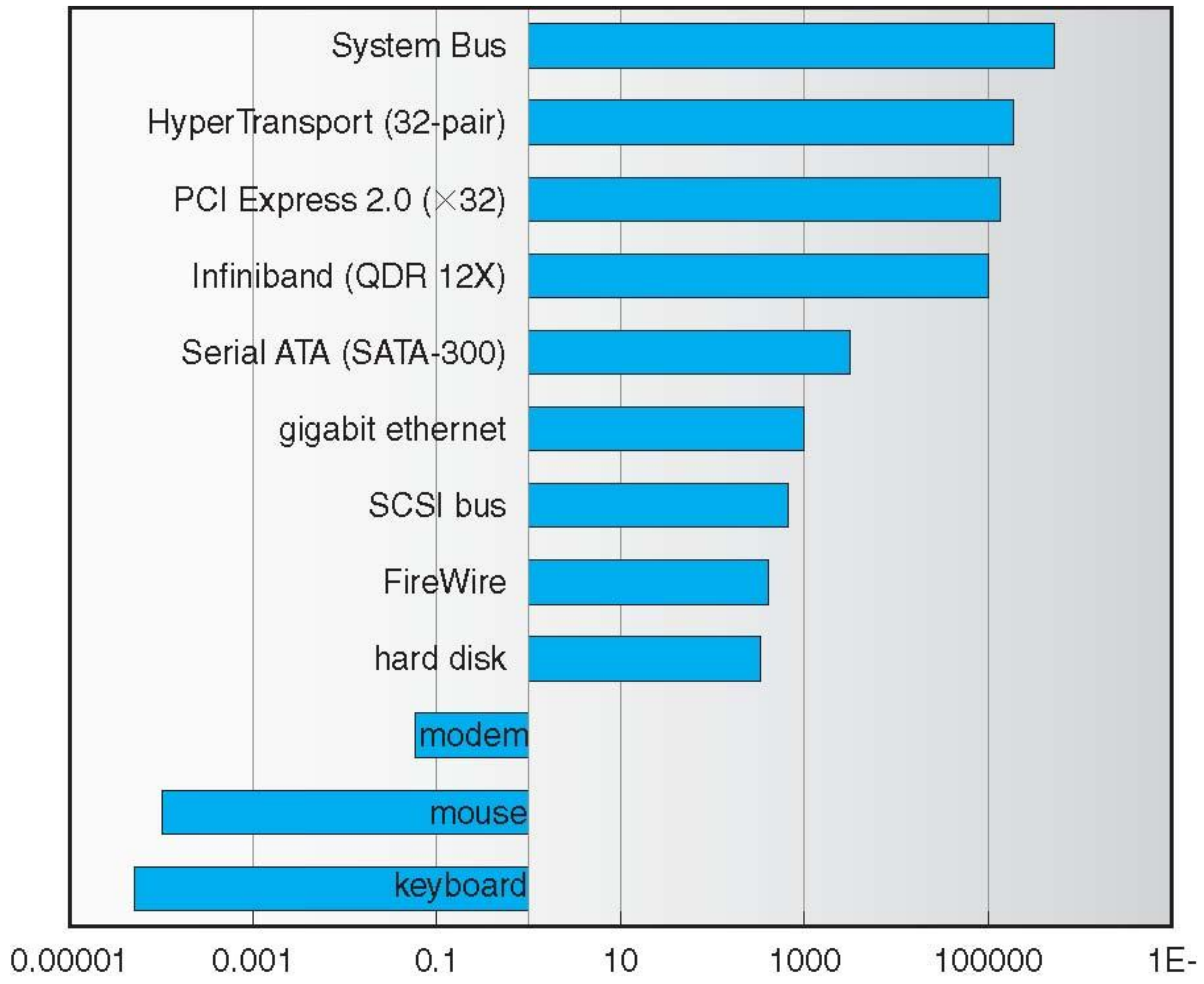
PCI Express* 2.0 Graphics
16 lanes
16 GB/s
or
PCI Express* 2.0 Graphics
8 lanes
8 GB/s
PCI Express* 2.0 Graphics
8 lanes
8 GB/s
Intel® Core™ processors[1]
DDR3 1333 MHz
DDR3 1333 MHz
DMI 20 Gb/s
Intel® High Definition Audio
14 Hi-Speed USB 2.0 Ports; Dual EHCI; USB Port Disable
480 Mb/s each
Intel® P67 Express Chipset
5 Gb/s each x1
8 PCI Express* 2.0
Intel® Integrated 10/100/1000 MAC
up to 6 Gb/s[2]
6 Serial ATA Ports; eSATA; Port Disable
PCIe* x1
SM Bus
SPI
Intel® Rapid Storage Technology
Intel® Gigabit LAN Connect
Intel® ME Firmware and BIOS Support
Intel® Extreme Tuning Support
▬ ....Optional
[1] Compatible with 2nd generation Intel® Core™ processor family
[2] All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.
DMI
PCI Express*
System Agent
IMC
Display
Core
LLC
Core
LLC
Core
LLC
Core
LLC
Graphics

- **Platform Controller Hub**
  - Connected to processor with proprietary bus
- **Types of I/O on PCH:**
  - USB
  - Ethernet
  - Audio
  - BIOS support
  - More PCI Express (lower speed than on Processor)
  - SATA (for Disks)

Modern I/O Systems



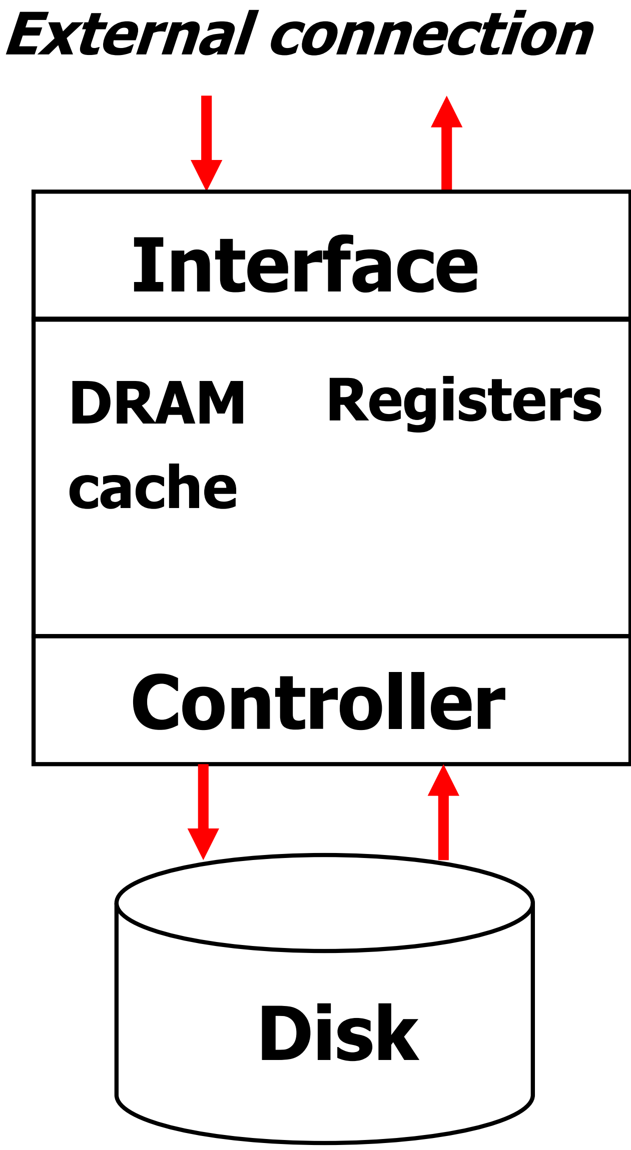
Example Device Transfer Rates (Mb/s)



- **Device rates vary over 12 orders of magnitude !!!**
- **What challenges does this introduce?**
  - System should be able to handle this wide range
  - Should not have high overhead/byte for fast devices!
  - Should not waste time waiting for slow devices

Disk Controller

- External connection following universal **bus standards**
  - IDE/ATA, SATA, SCSI → Ultra-320 SCSI, PCI Express
- The OS will have **drivers** that are able to 'speak' to different disk interfaces
- The controller typically has some data buffers (**cache**)
  - Can store neighbouring sectors from where head is reading for future use (why? Because of locality)
  - Can store recently accessed blocks
  - Are writes reliable? What happens if data is in cache and power goes out?
- **Controller's typical tasks**
  - Read/write operations
  - Performs validation / any error correction necessary
  - Communicates with the CPU



Memory-Mapped I/O

- **How does the CPU communicate with the device?**
  - Especially, how does it communicate with the controller's data buffers (cache) where the actual data is?
- **Memory mapped I/O (load/store instructions)**
  - I/O happens by reading from and writing to special locations in (main) memory
  - Essentially some **locations in main memory are reserved to store I/O device 's buffers and registers**
  - The part of main memory mapped for I/O is typically not available for any other operations apart from communicating with that I/O device

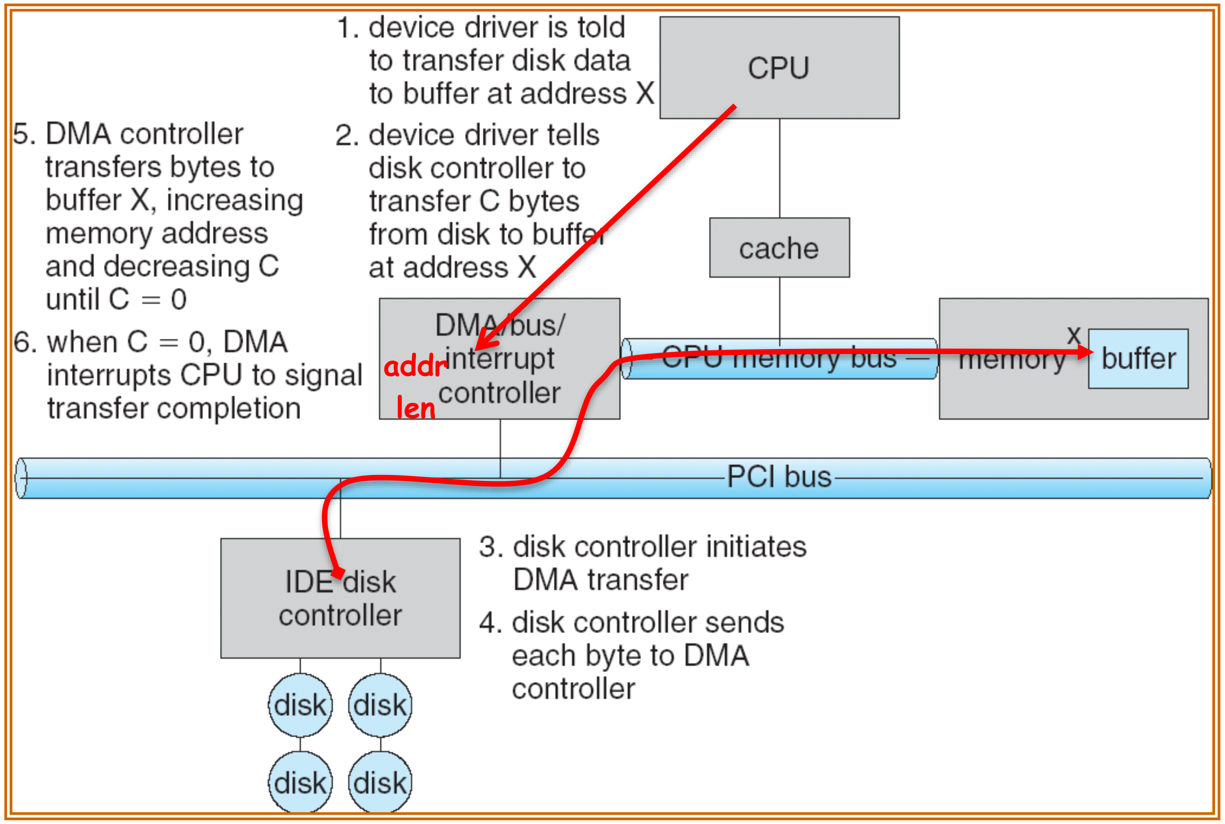
Transferring Data To/From Controller: Direct Memory Access (DMA)

- Data transfer between memory and device without the CPU
  - The hardware must have a DMA controller for this to happen, typically on the parentboard
- The DMA controller has access to memory bus independent of the CPU
- **The bus reads/writes directly from/to the CPU's memory**
  - CPU starts I/O operation by instructing the DMA controller to get some data
  - ... DMA transfers block of data to the memory
  - finally the CPU receives an interrupt from the DMA controller when the operation is done
- **The CPU is interrupted only at start and end of the transaction**
  - rather than for each memory read / write if the CPU was doing the job itself
- **Without DMA** the CPU would typically be occupied only with the read/write operation for the whole duration (**Programmed I/O**)

DMA Bus Sharing

- The DMA controller shares the bus with the CPU
  - Use the bus when the processor is not using it, **or**
  - Force processor to suspend operation temporarily (**cycle stealing**)

Slows down the CPU but not as much as if when CPU is doing the data transfer itself


1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/interrupt controller
addr
len
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk

Blocking vs. Non-blocking and Asynchronous I/O

- **Blocking (synchronous) – "Wait"**
  - "I want the data and I am going to stop until it comes"
  - Put the requesting process to sleep until the read/write is done
  - It is simple to manage
- **Non-blocking – "Don't wait"**
  - Returns quickly from read or write request with count of bytes successfully transferred
  - Read may return nothing, write may write nothing, we keep trying (polling)
- **Asynchronous (also non-blocking) – "Tell me later"**
  - When process requests/sends data, give pointer to buffer for kernel to fill/take data, return immediately to process;
  - When the kernel fills the buffer/takes the data then it notifies the process

I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- **I/O Interrupt:**
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupt has relatively high overhead (saving/loading contexts, etc.)
- **Polling**:
  - OS periodically checks a device-specific status register ("are you done yet?") - I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts

Life Cycle of an I/O Request

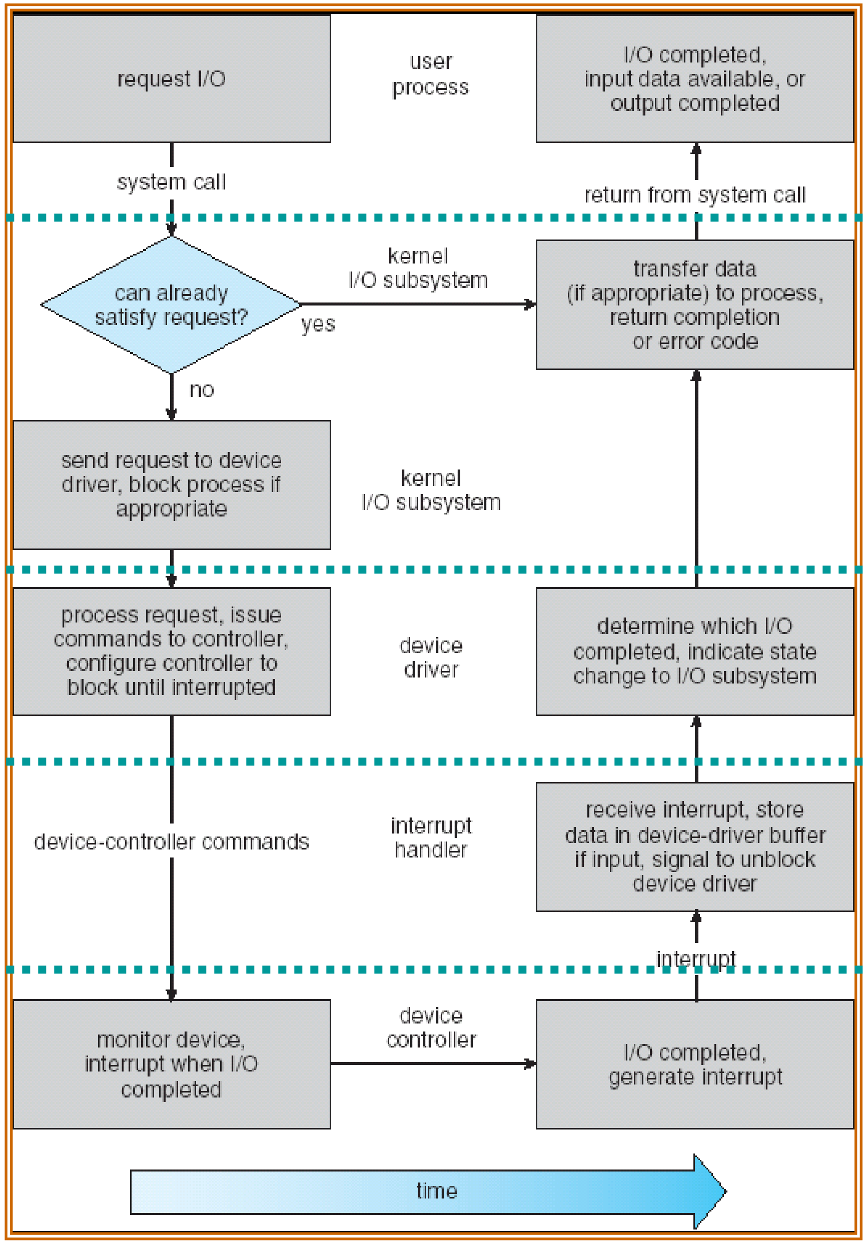
User Program

Kernel I/O Subsystem

Device Driver Top Half

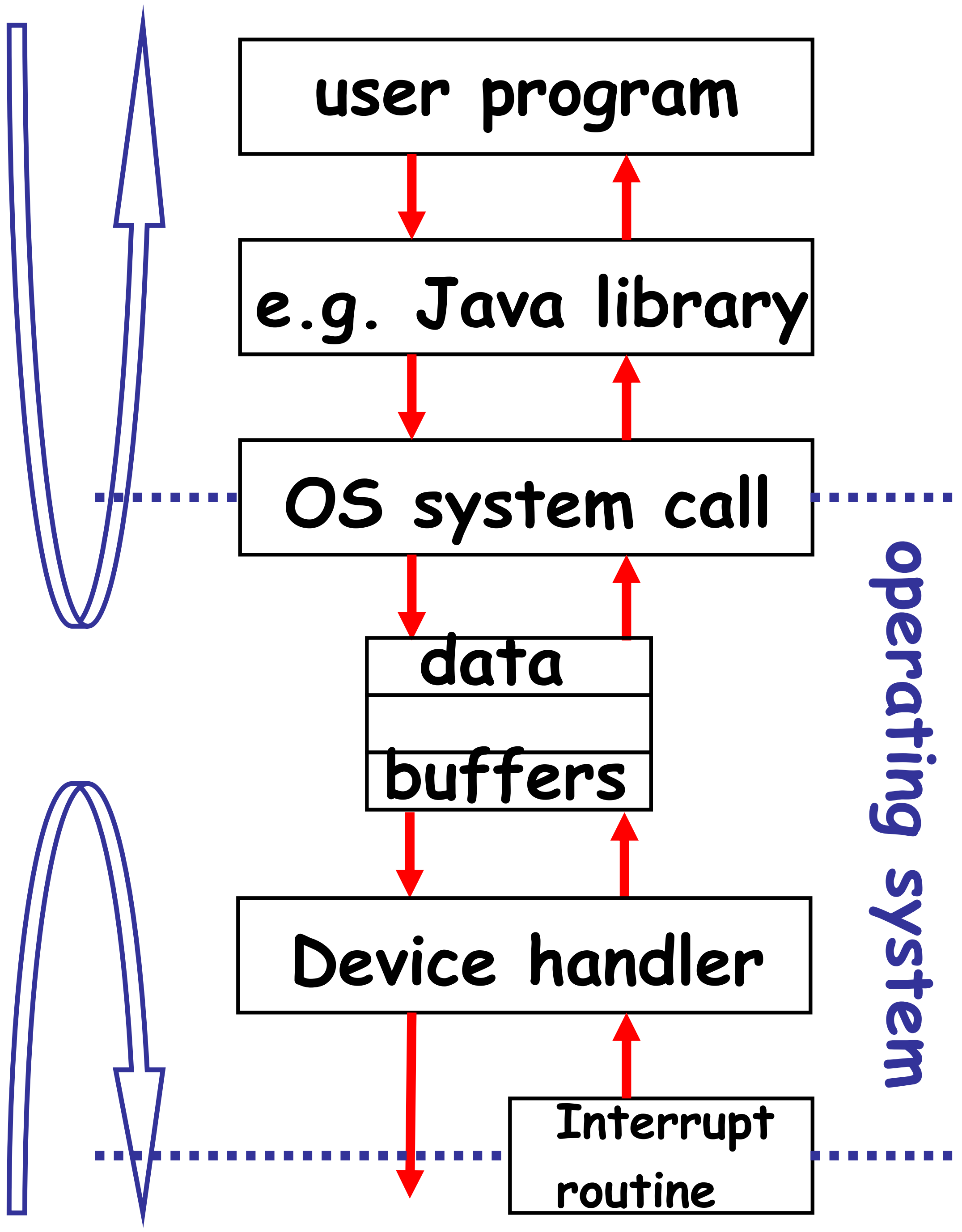
Device Driver Bottom Half

Device Hardware


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

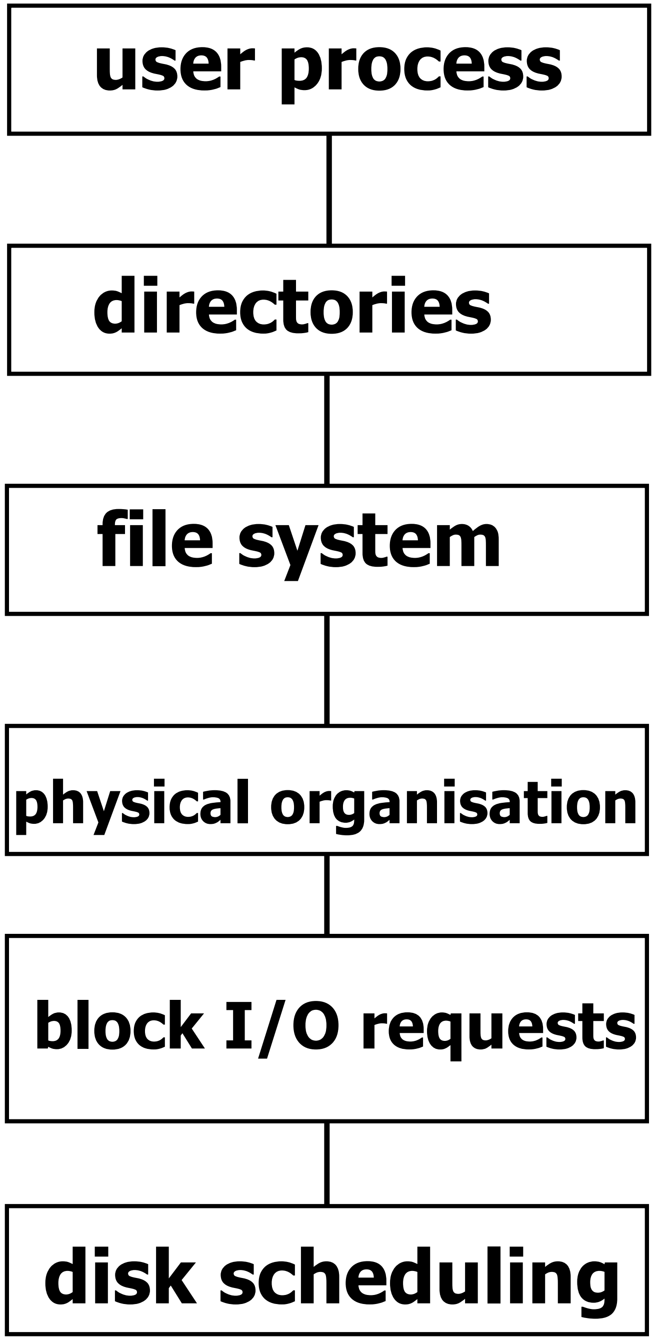
Software for I/O

- You want to read some bytes of data from e.g. a Java method
- Your code will communicate with some I/O system call
- S/W **trap** for a **mode switch** will take place
- **Buffers** in memory may have the data
  - If this is the case most likely there will be **no context switch**
- If the data is not in memory and interrupt will occur
  - This will lead to a **context switch**



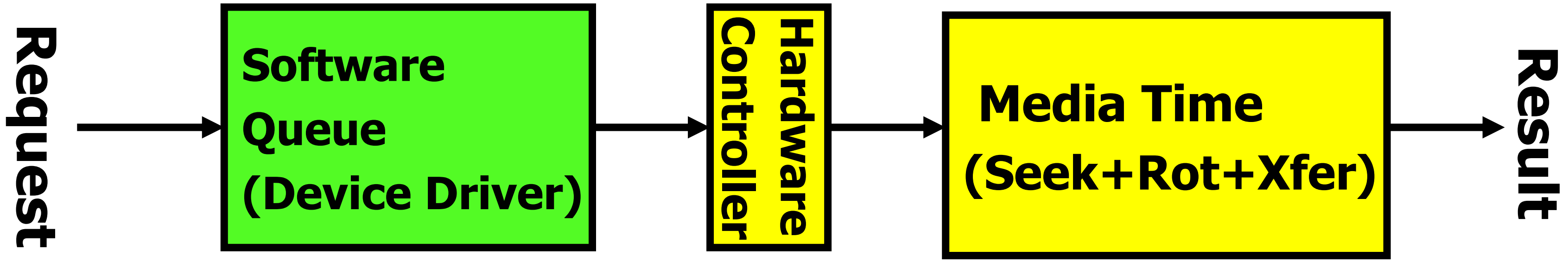
Software Hierarchy

- Directory management
- File system
- Physical addresses and buffers
- Device level I/O and cache
- **Disk scheduling**
  - It may be advantageous to change the order in which I/O requests come to the device
  - Can you think why?
    - Remember: (**Seek time** + rotational delay) + transfer delay



Disk Scheduling

- The main problem is that **disk access is slow**
  - 5 ms @ 2 GHz ➔ 10 million instructions!
- **Seek times** are in the order of 10ms and they dominate the time taken for read/write operations
- **The device driver** can use different methods for scheduling requests for the disk
- In all examples that follow **we assume a static queue**, but in reality requests will be arriving all the time for processes to do I/O

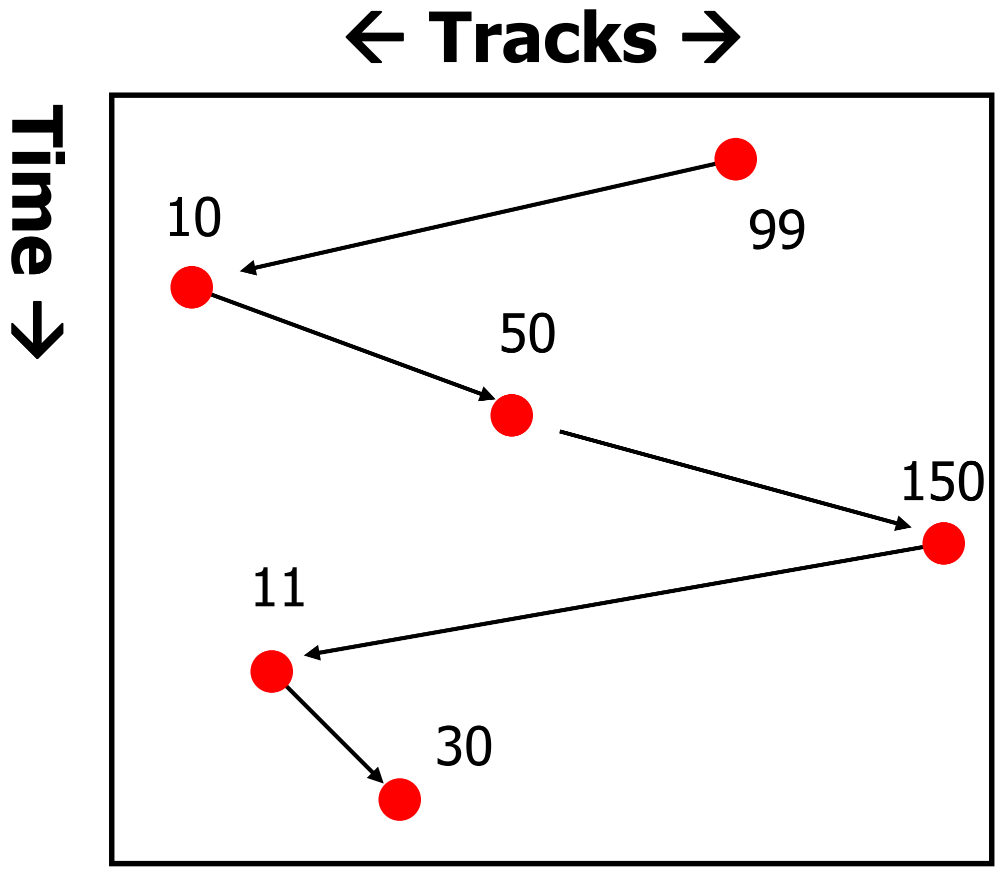


FIFO

Tracks requested →

99, 10, 50, 150, 11, 30

- First come first served
- The most fair
- BUT order of arrival may be to random spots on the disk ⇒ **very long seeks**



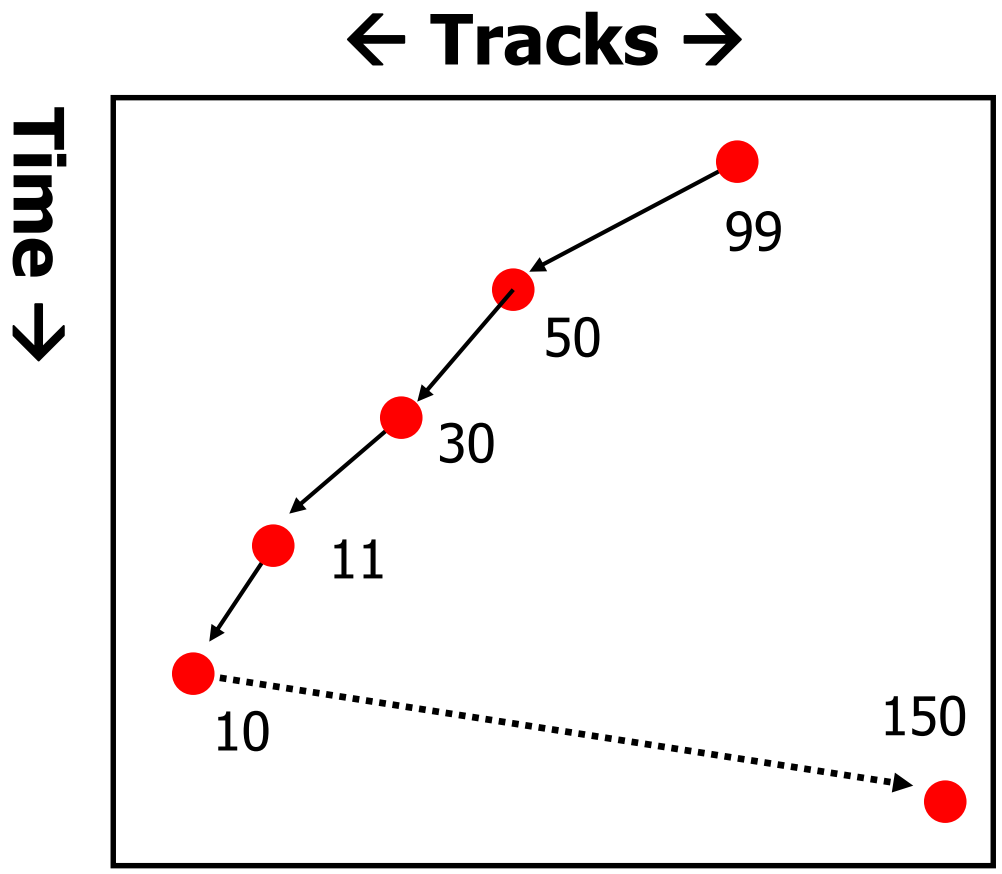
SSTF

- **Shortest Seek Time First**
- Closest to where we are currently on the disk
  - we must also include rotational delay
- SSTF is quite good at reducing seek times
- Unbounded wait - **it may lead to starvation** (see example with request for track 150)
  - If new requests keep arriving that are close to track 10, track 150 will be made to wait a very long time (or possible forever…)

Tracks requested →

99, 10, 50, 150, 11, 30

(we assume head is on track 99 when we start)



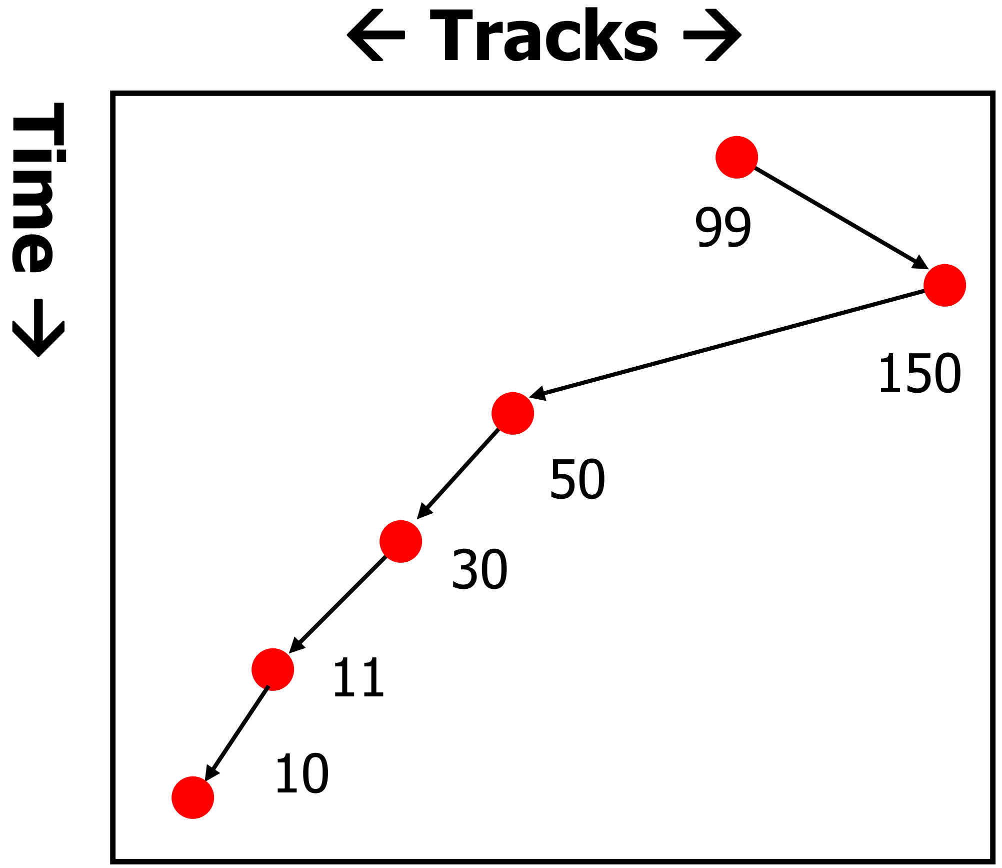
Scan

- Also called **elevator** algorithm
- Take the **closest request in the direction of travel**
- When we reach the 'end' of the tracks we reverse the direction of travel and satisfy remaining requests
- Bounded wait -> It avoids the problem of starvation
- But, it is not fair (see example of request for track 10)
  - It is also biased against the area of the disk that the head has most recently passed

Tracks requested →

99, 10, 50, 150, 11, 30

(we assume head is on track 99, direction of travel towards higher track numbers)



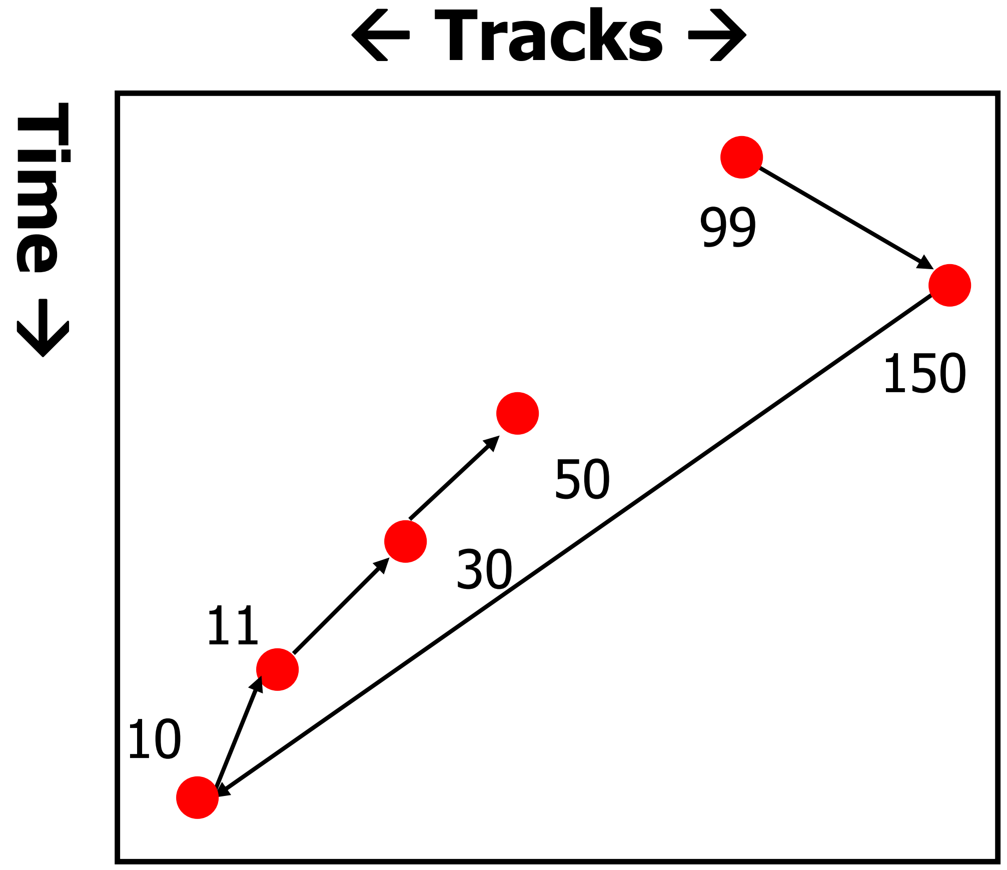
C-Scan

- **Circular scan algorithm**
- Take the **closest request in the direction of travel**
- When we reach the 'end' of the tracks we return the head to the opposite end of the disk and the scan begins again
- It skips any requests on the way back (when we move the head back to the start it does not satisfy requests)
- Fairer than SCAN

Tracks requested →

99, 10, 50, 150, 11, 30

(we assume head is on track 99, direction of travel towards higher track numbers)



Anticipatory schedulers

- **A further problem with the elevator algorithm**
  - Works **against locality of reference**
  - If we get a read request for a track/sector we are very likely to get another request nearby (why?)
  - **If we scan too fast in an elevator algorithm**, then that other request will most likely be left behind
- **Anticipatory I/O scheduler**
  - Delay after satisfying a read request to see if a new nearby request can be made
  - By introducing some delay, we may actually get faster overall I/O from the disk

Lab 5 - Assessed

- **Lab 5: File systems (Assessed)**
  - Work with inodes and links
  - Sample code will be given, extend to complete tasks
- **Some questions ask you to explain / describe**
  - e.g. look at info in the inode (use `stat` command), explore, read and answer the questions about access times
  - Create hard and soft links in your directory, learn and study how they behave (delete, move, rename, watch the inodes, etc.)
  - etc…
- Some ask you to **implement PHP scripts** based on sample code
  - Understand the code we give you
  - Read carefully the task descriptions
  - Experiment with the functions we suggest in the lab sheet, understand what they do
  - Complete the tasks

Summary

- Disks have high latency (rotation / seek)
  - Cache memory to store recent data
  - Clever algorithms for scheduling requests
  - SSDs are becoming increasingly popular, but …
- **I/O is a central part of an OS**
  - CPU without I/O -> disembodied brain
  - Main (physical) memory is used for communication between CPU and device controllers
  - DMA allows reads/writes without the CPU being tied up the whole time