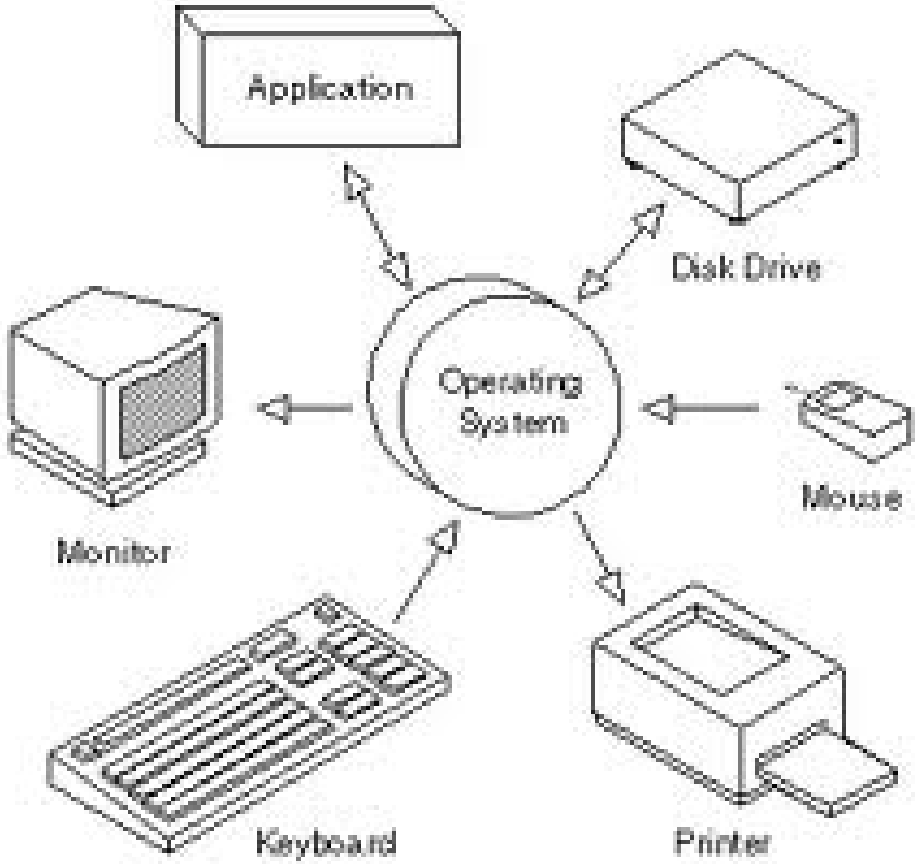
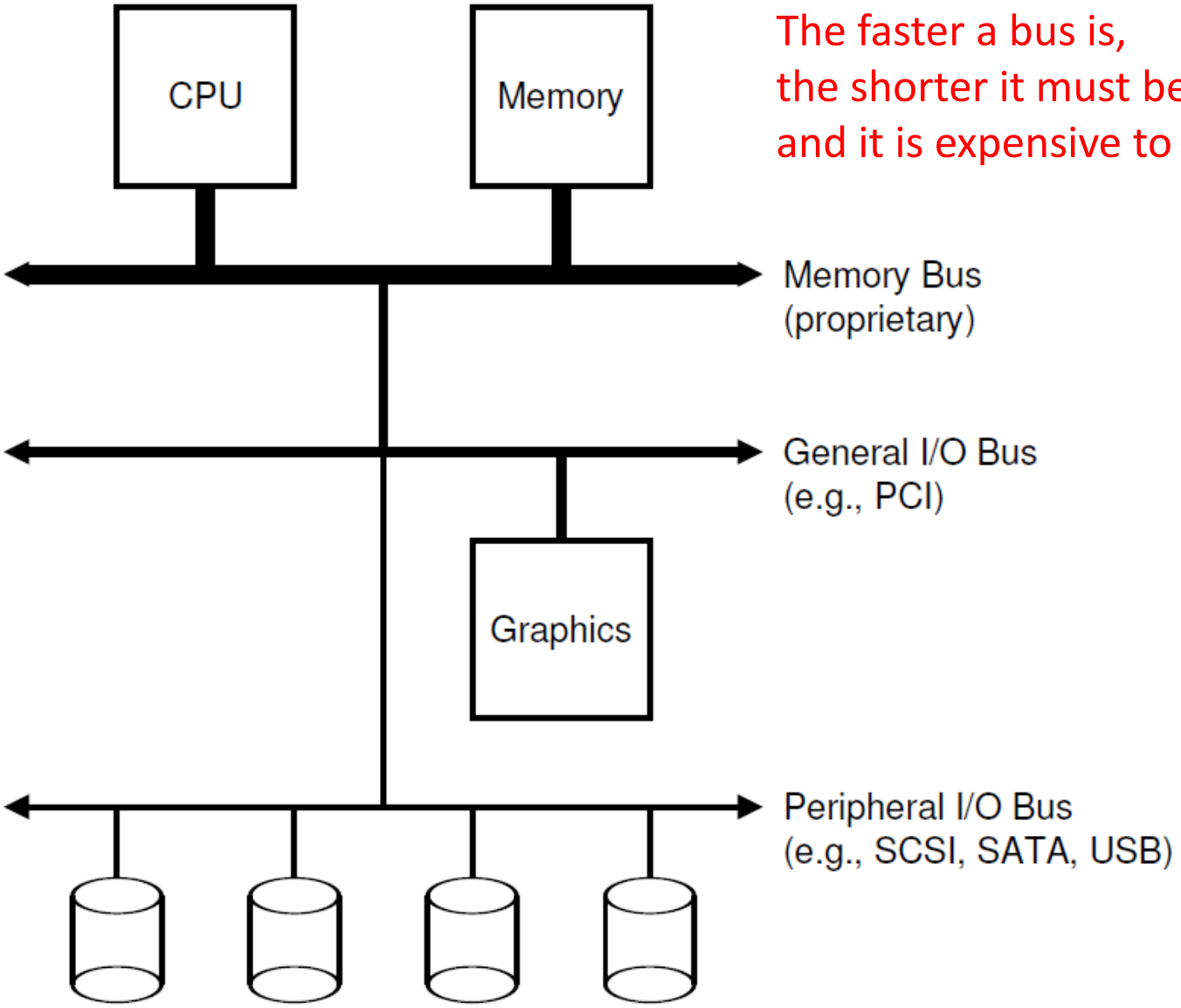
NYU
New York University

Operating Systems

I/O

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com


Application
Disk Drive
Operating System
Monitor
Mouse
Keyboard
Printer



The faster a bus is,
the shorter it must be,
and it is expensive to design too.

Categories of I/O Devices

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

Human readable

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

Machine readable

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

Communication

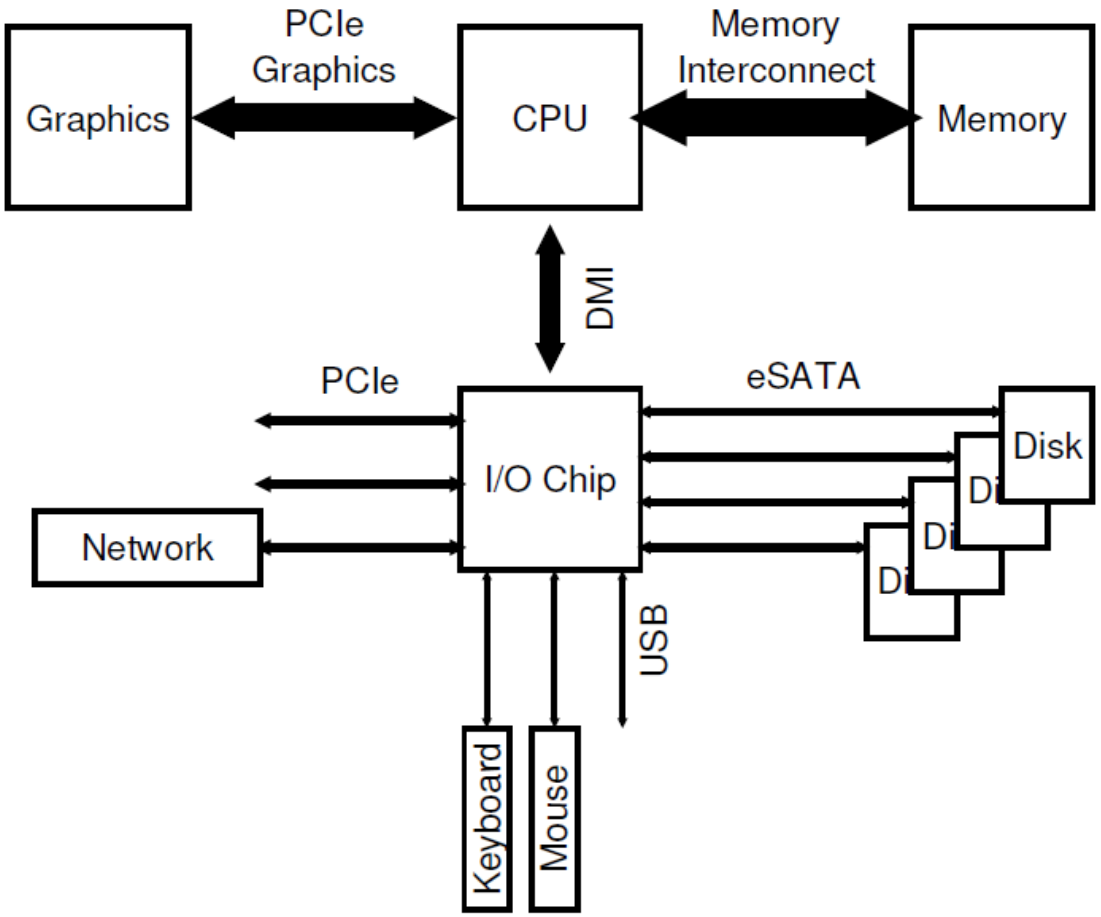
- suitable for communicating with remote devices
- ethernet cards, wifi adapters

A Simple Definition

- The main concept of I/O is to move data from/to I/O devices to the processor using some modules and buffers.

- This is the way the processor deals with the outside world.

Modern System Architecture

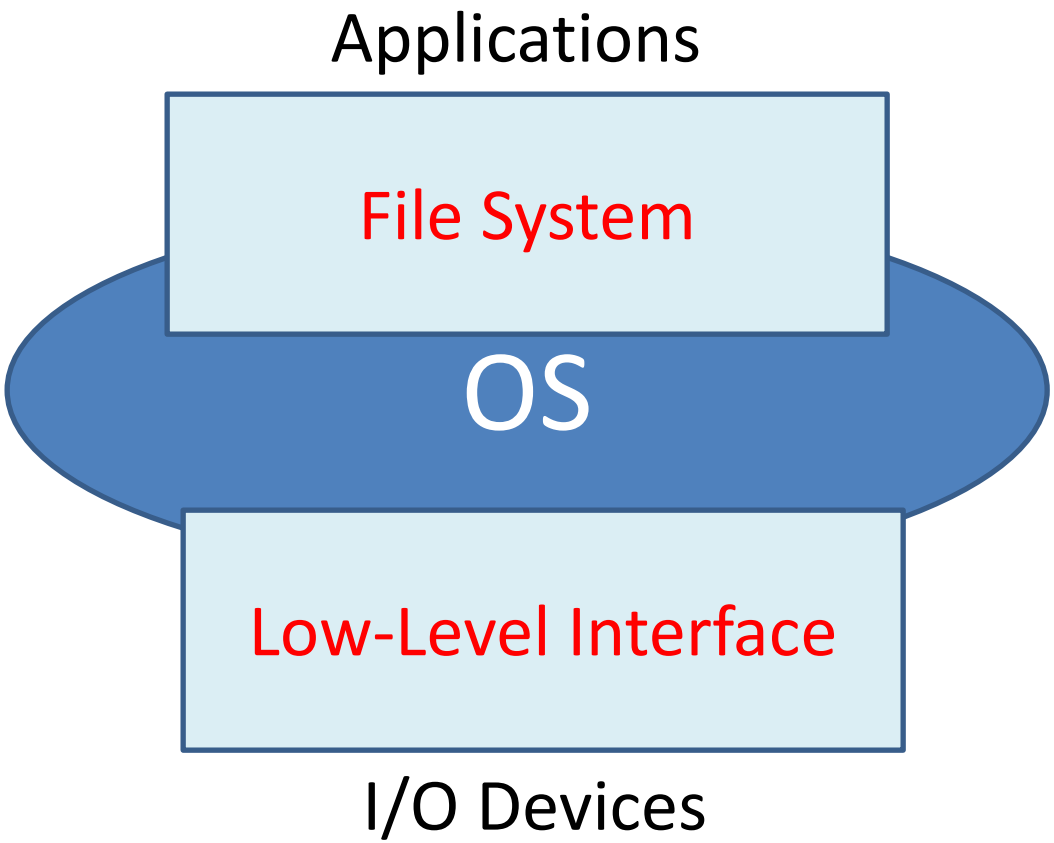


Example of one of Intel Chipsets.

DMI: Intel's proprietary Direct Media Interface.

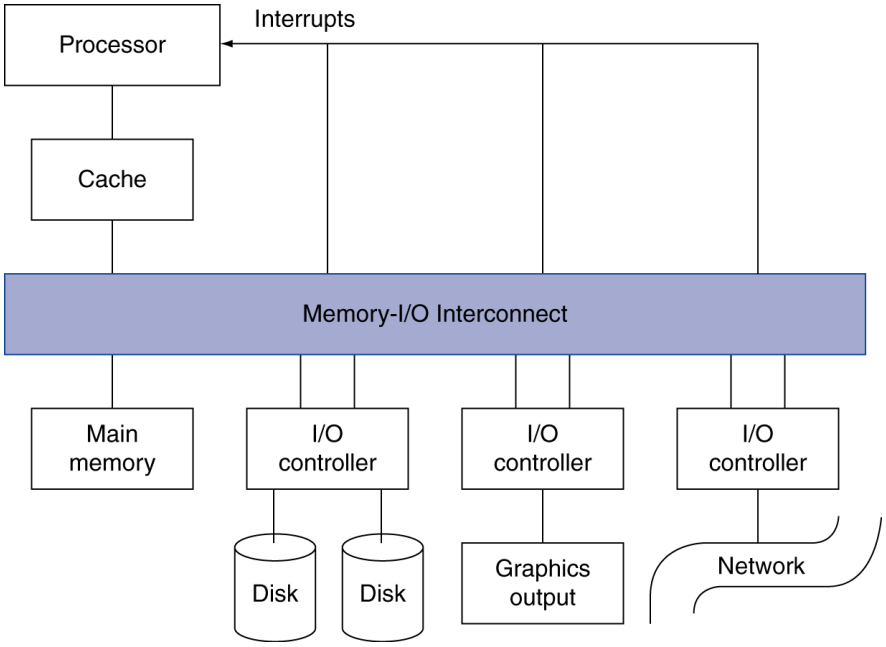
The OS and I/O

- The OS controls all I/O devices
- Provides an interface between the devices and the rest of the system.

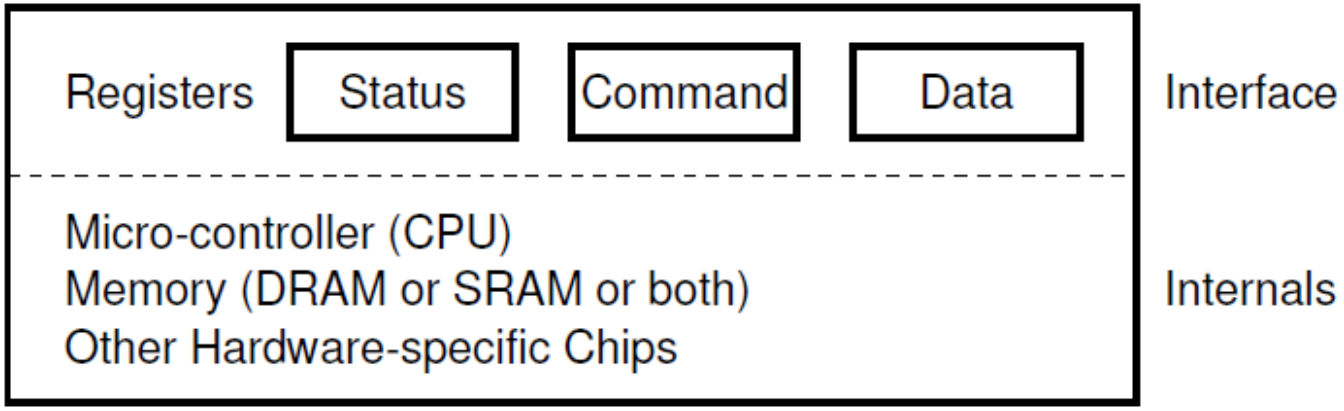

Applications
File System
OS
Low-Level Interface
I/O Devices

I/O Devices: Challenges

- Very diverse devices
  - behavior (i.e., input vs. output vs. storage)
  - partner (who is at the other end?)
  - data rate
- I/O Design affected by many factors (expandability, resilience)
- Performance:
  - access latency
  - throughput
  - connection between devices and the system
  - the memory hierarchy
  - the operating system
- A variety of different users



How Does a Generic I/O Device Look Like?



The OS can deal with the above device as follows:

```
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
```

Polling is not efficient.

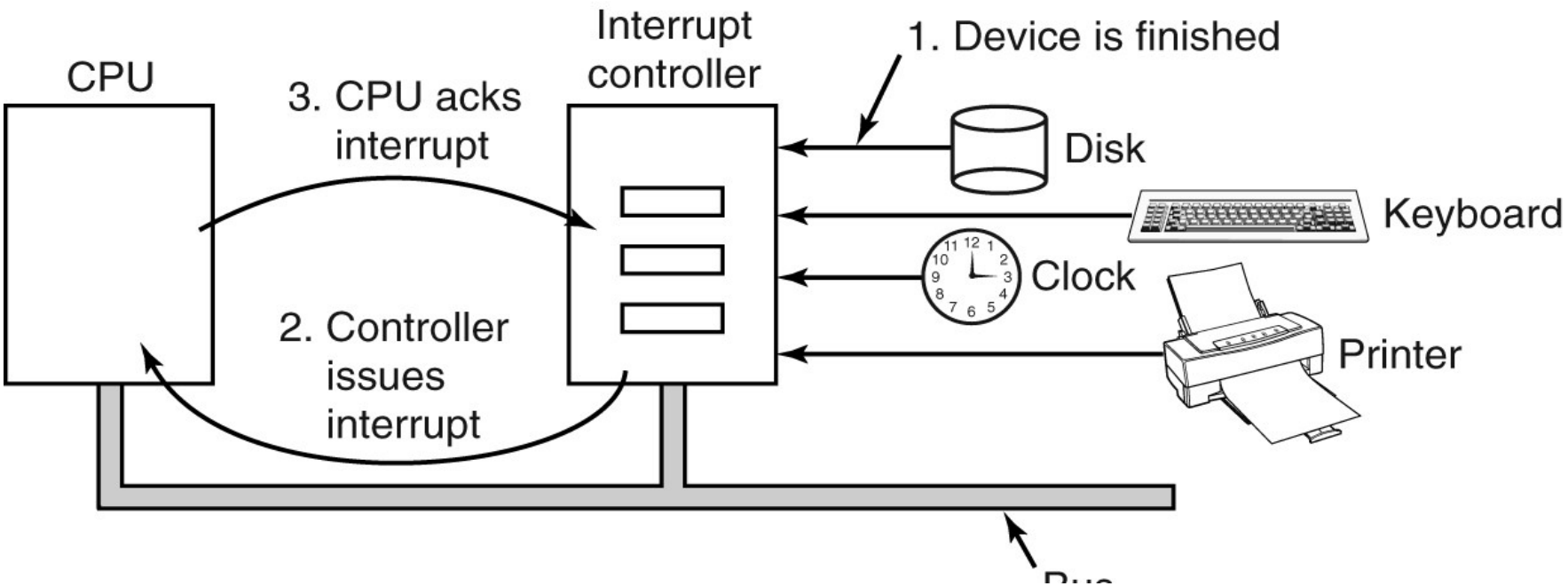
Efficient OS-Device Interaction

- Polling (see previous slide)
  - Called programmer I/O
- Interrupts
- DMA

Efficient OS-Device Interaction: Interrupts

- OS issues a request and puts the calling process to sleep.
- OS context switches to another process.
- When done, the device raises an interrupt.
- CPU moves to kernel mode and OS executes the needed interrupt service routine (ISR).
- The OS wakes the process that needed the I/O.

Interrupts


CPU
3. CPU acks interrupt
Interrupt controller
1. Device is finished
Disk
Keyboard
Clock
Printer
2. Controller issues interrupt

Polling vs Interrupts

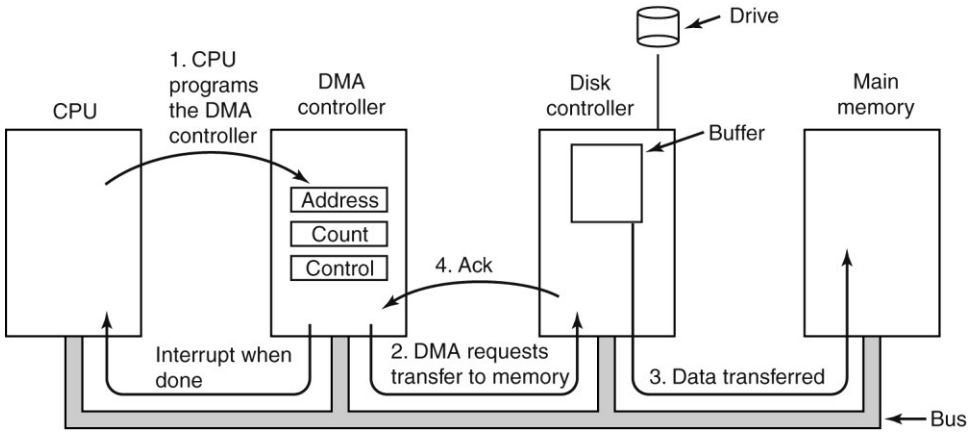
- If a device is fast, it may be best to poll.
  - The cost of interrupt handling and context switch may outweigh the benefit of interrupts.
- If a device is slow, then interrupts is better.

Efficient OS-Device Interaction: DMA

- It is better not to bug the OS with a lot of overhead when moving a large chunk of data.
- DMA = Direct Memory Access
  - A device in the system that orchestrates transfers between devices and main memory without CPU intervention.
- OS tells DMA where the data lives, how much to copy, etc. The OS is done with the transfer and can go do something else.
- When the DMA completes, it raises an interrupt telling the OS the operation is done.

Direct Memory Access (DMA)

- It is not efficient for the CPU to request data from I/O one byte at a time.
- DMA controller has access to the system bus independent from the CPU



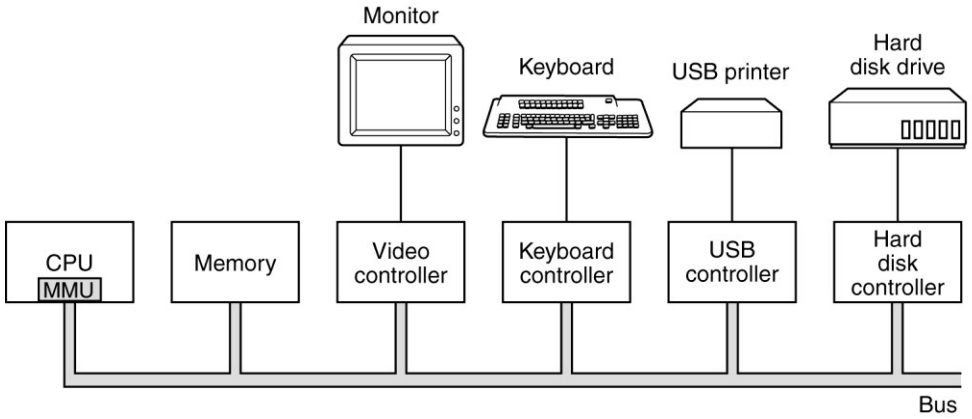
I/O Units

Mechanical component

Electronic Component

The Device Itself

Device Controller



Controller and Device

- Each controller has few registers used to communicate with CPU
- By writing/reading into/from those registers, the OS can control the devices.
- There are also data buffers in the device that can be read/written by the OS.

How does the OS deal with controllers and devices?

How does CPU communicate with control registers and data buffers?

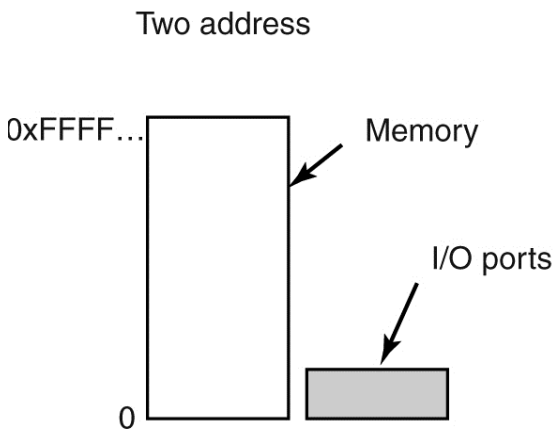
Two main approaches

- I/O Instructions
  - Privileged

- Memory-mapped I/O

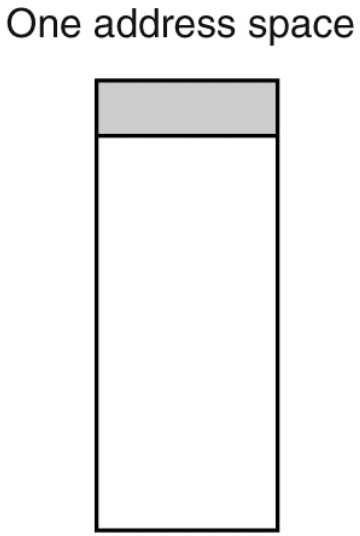
I/O Instructions

- Each control register is assigned an I/O port number
- The set of all I/O ports form the I/O port space
- I/O port is accessed with special instructions
- I/O port space is protected



Memory-Mapped I/O

- Map control registers into the memory space
- Each control register is assigned a unique memory address

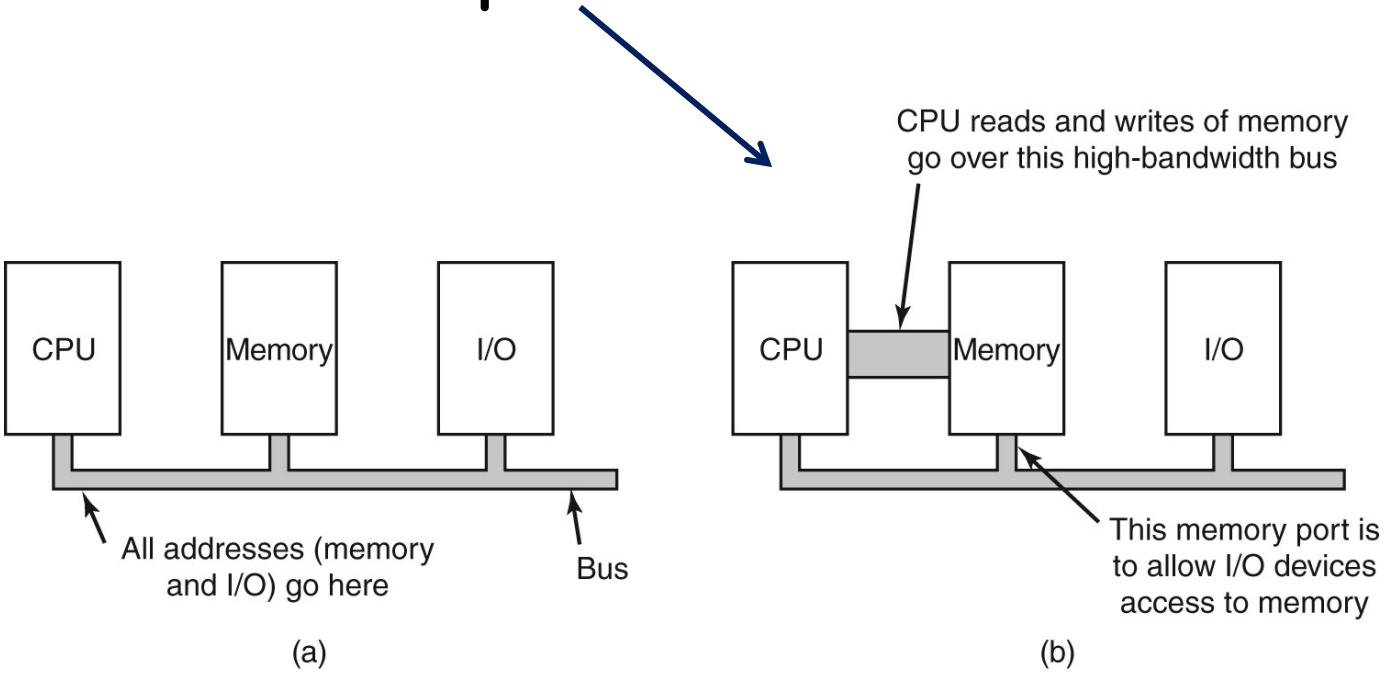


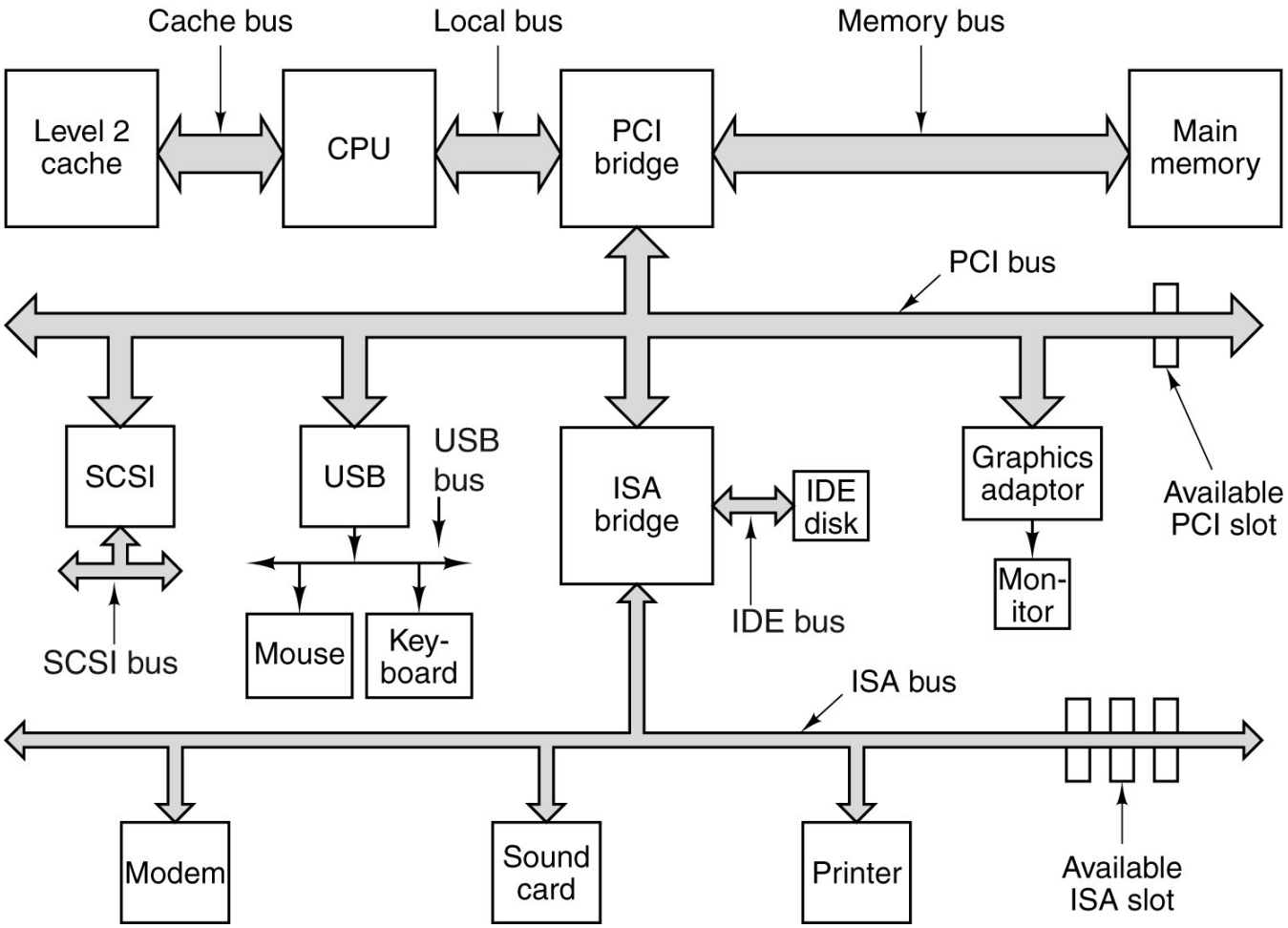
Advantages of Memory-Mapped I/O

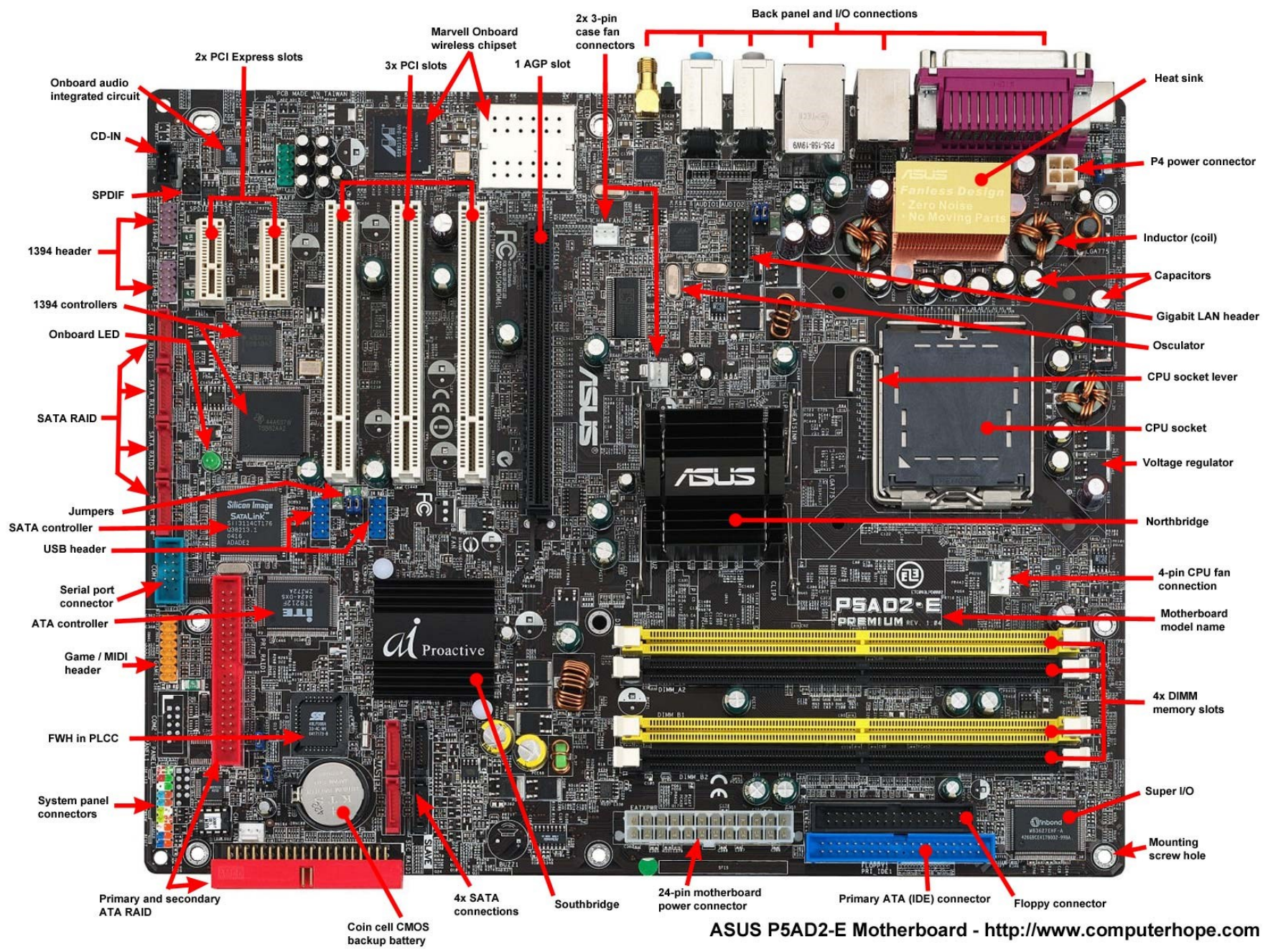
- Device drivers can be written entirely in C (since no special instructions are needed)
- No special protection is needed from OS, just refrain from putting that portion of the address space in any user's virtual address space.
- Every instruction that can reference memory can also reference control registers.

Disadvantages of Memory-Mapped I/O

- Caching a device control register can be disastrous.
- Hardware complications



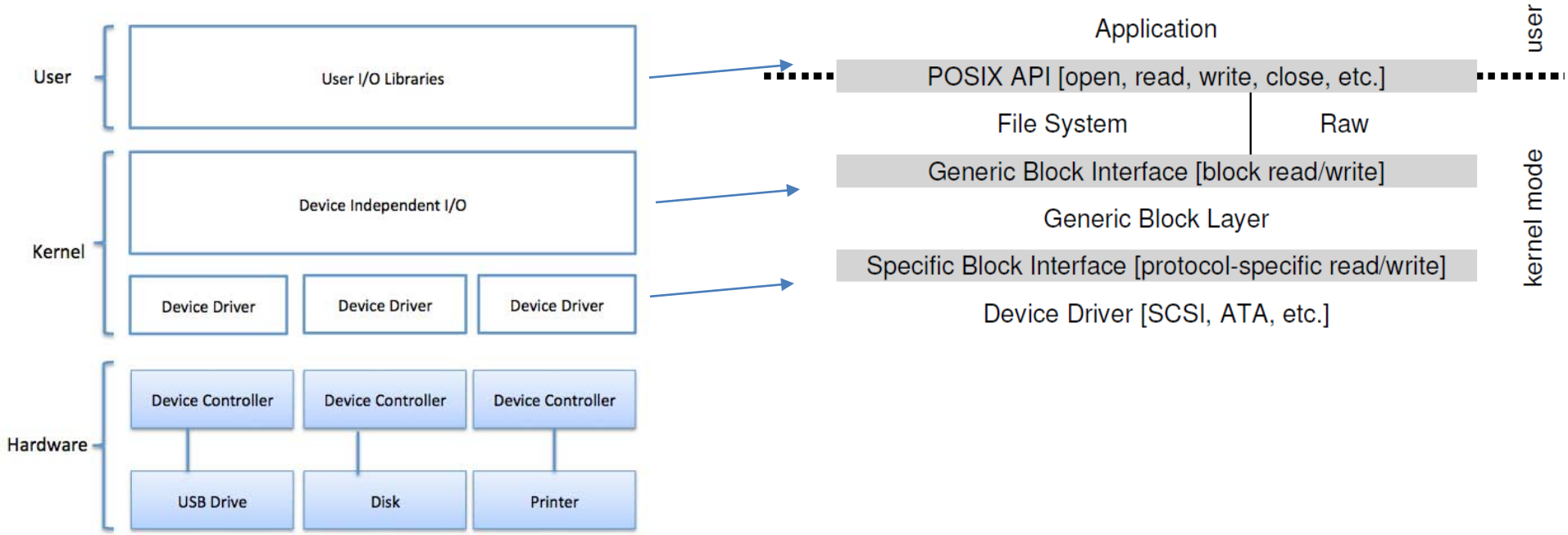

Cache bus
Local bus
Memory bus
Level 2 cache
CPU
PCI bridge
Main memory
PCI bus
SCSI
USB
USB bus
ISA bridge
IDE disk
Graphics adaptor
Available PCI slot
SCSI bus
Mouse
Key-board
IDE bus
Mon-itor
ISA bus
Modem
Sound card
Printer
Available ISA slot



Source: https://www.computerhope.com/cdn/bigmb.jpg

The Software Part

The Big Picture



Source: https://www.tutorialspoint.com/operating_system/images/io_software.jpg

Device Independent I/O Software

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Device Independent I/O Software

- Uniform interfacing for device drivers
  - Trying to make all devices look the same
  - For each class of devices, the OS defines a set of functions that the driver must supply.
  - This layer of OS maps symbolic device names onto proper drivers

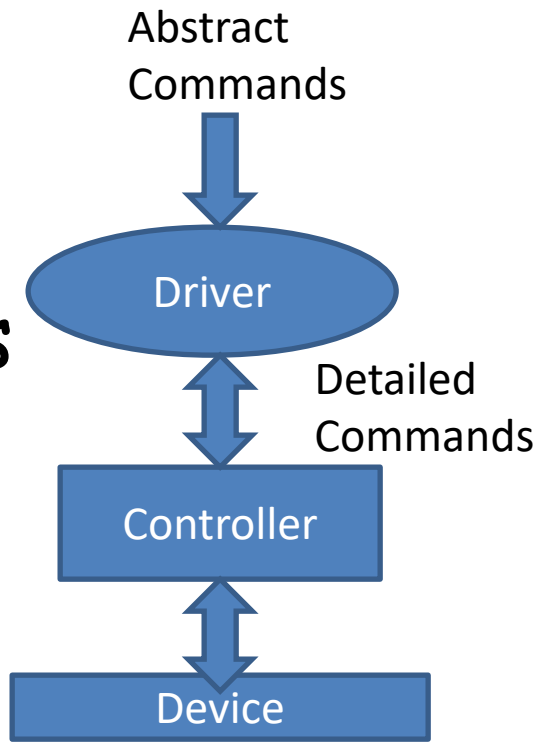
How About Device Drivers?

Device Drivers

- Device specific code for controlling the device
  - Read device registers from controller
  - Write device registers to issue commands
- Usually supplied by the device manufacturer.
- Can be part of the kernel or at user-space (with system calls to access controller registers).
- OS defines a standard interface that drivers for block devices must follow and another standard for driver of character devices.

Device Drivers

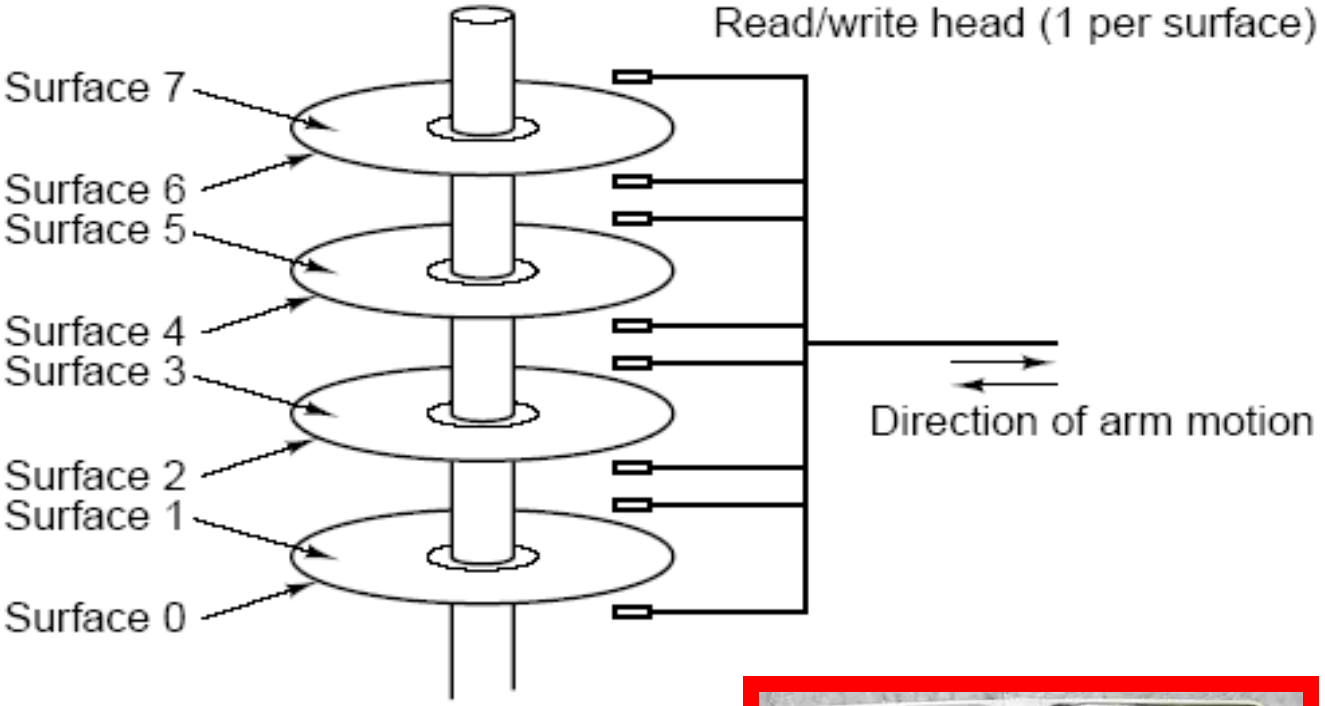
- Main functions:
  - Receive abstract read/write from layer above and carry them out
  - Initialize the device
  - Log events
  - Manage power requirements
- Drivers must deal with events such as a device removed or plugged

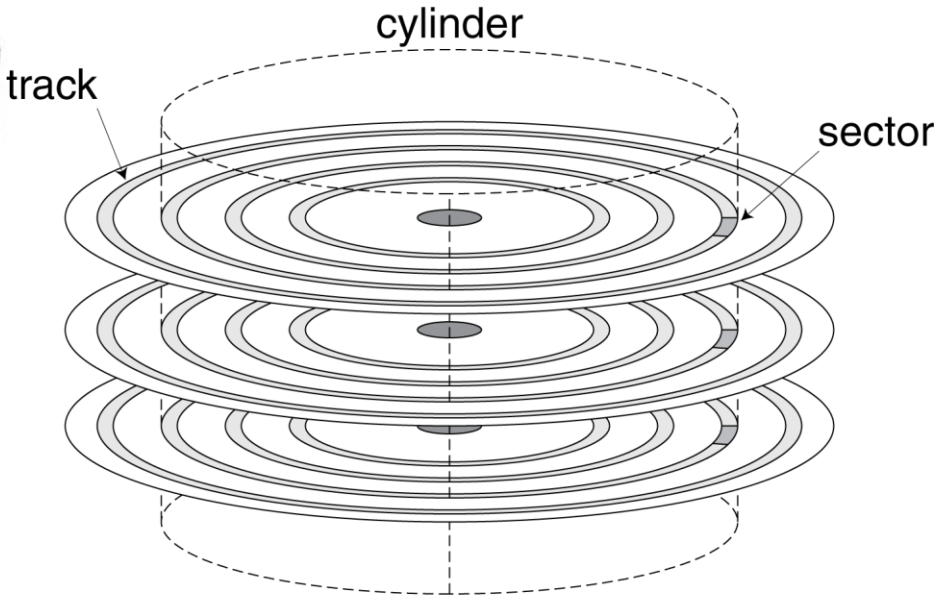


Example of I/O Devices

Hard-Disk Drive (HDD)

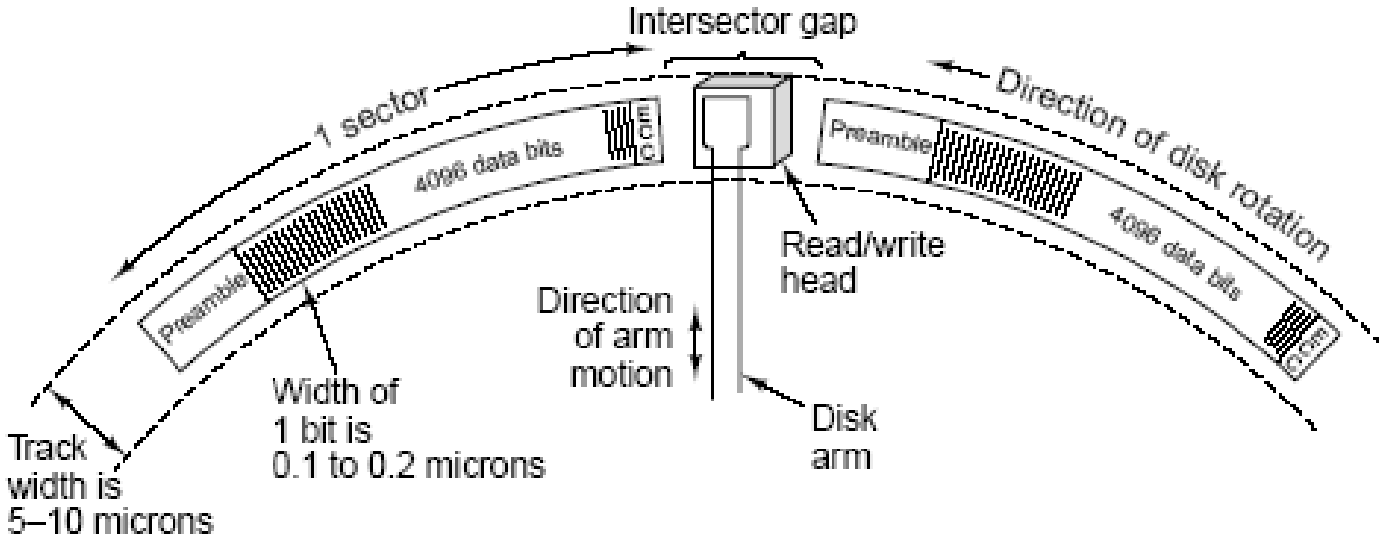
A Conventional Hard Disk (Magnetic) Structure




cylinder
track
sector

Hard Disk (Magnetic) Architecture

- **Surface** = group of tracks
- **Track** = group of sectors
- **Sector** = group of bytes
- **Cylinder**: several tracks on corresponding surfaces

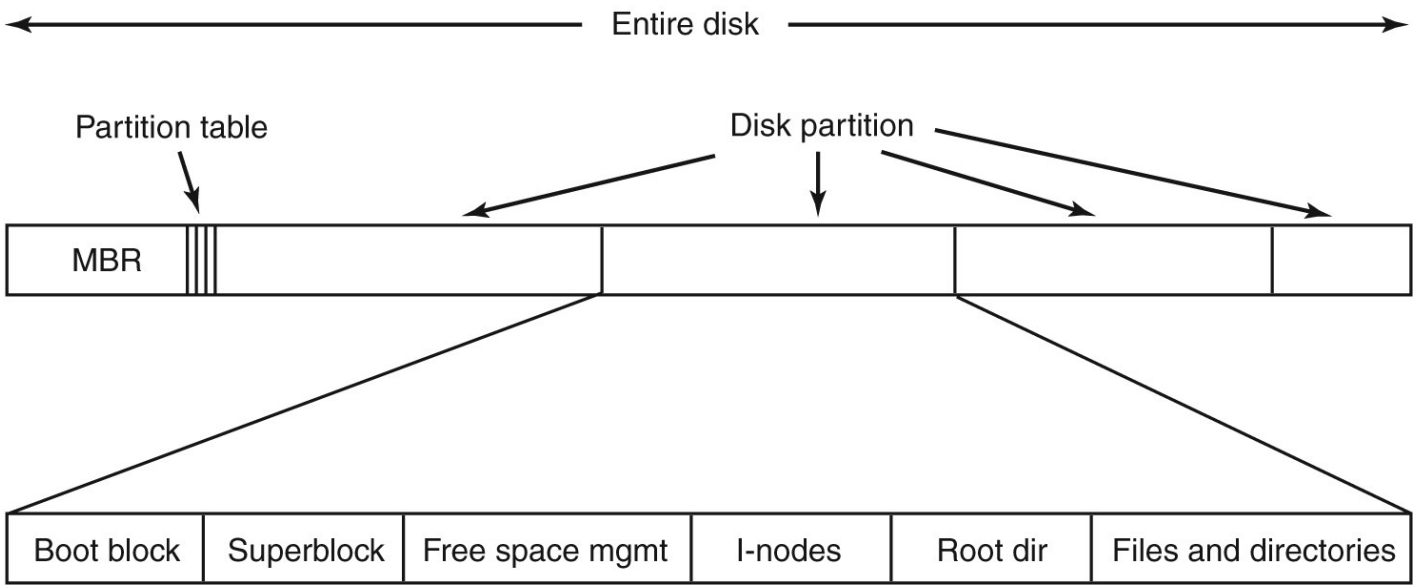


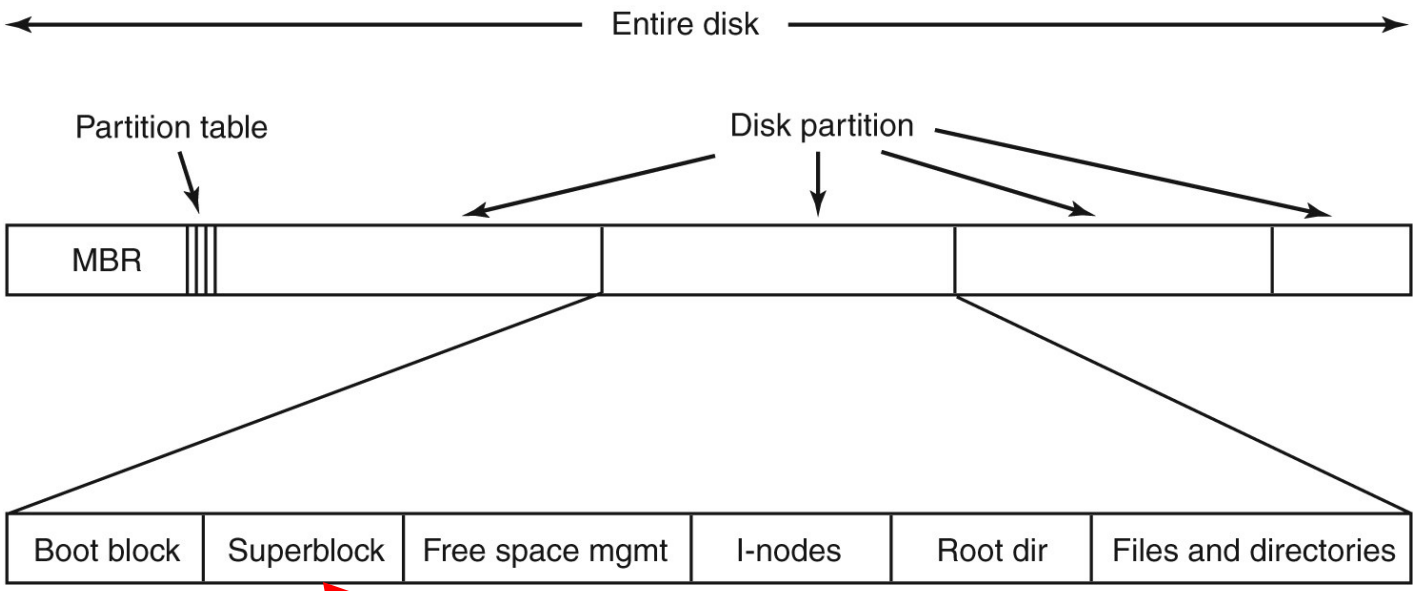
File System Layout

- Stored on disks
- Disks can have partitions with different file systems
- Sector 0 of the disk called MBR (Master Boot Record)
- MBR used to boot the computer
- The end of MBR contains the partition table

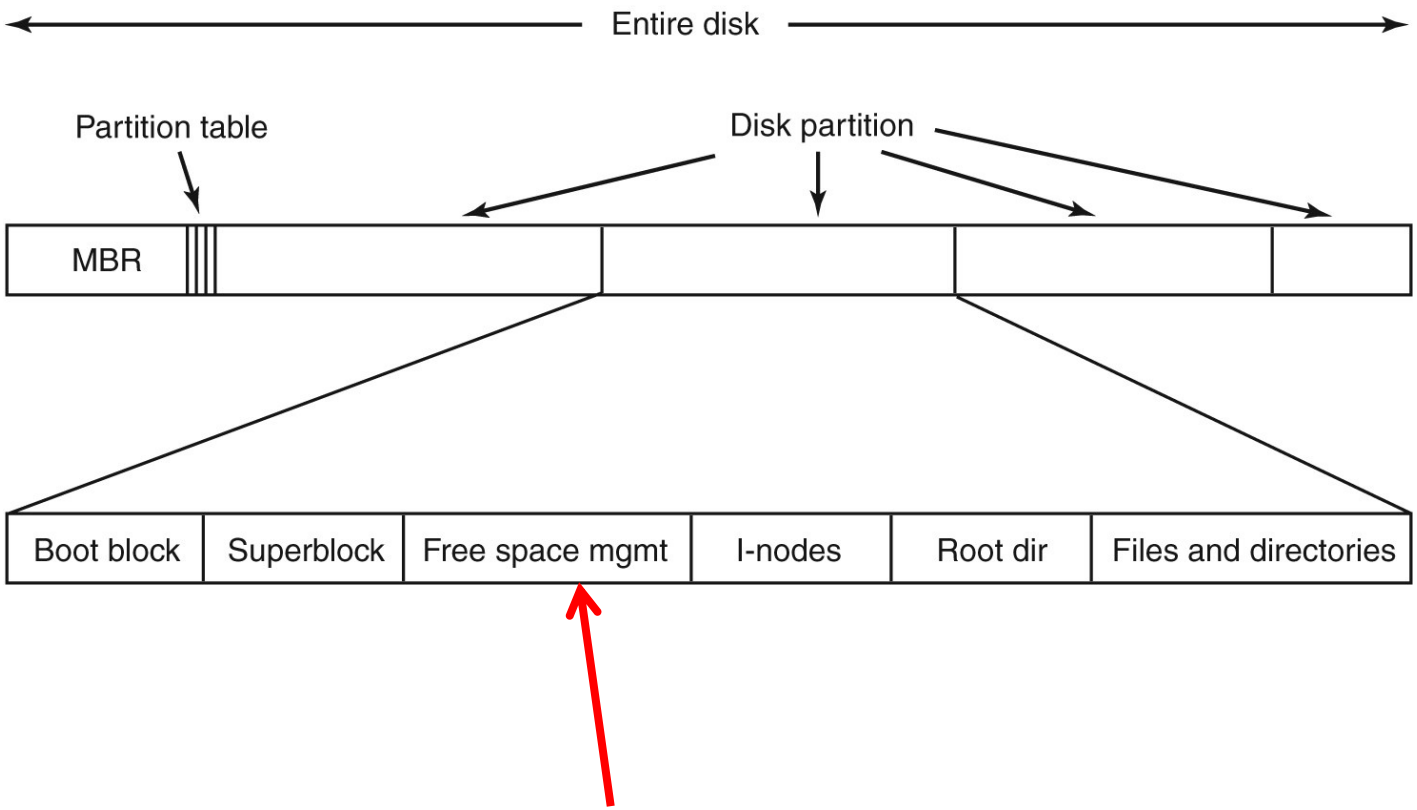
MBR and Partition Table

- Gives the starting and ending addresses of each partition
- One partition in the table is marked as active.
- When the computer is booted, BIOS executes MBR.
- MBR finds the active partition and reads its first block (called boot block) and executes it.
- Boot block loads the OS contained in that partition.

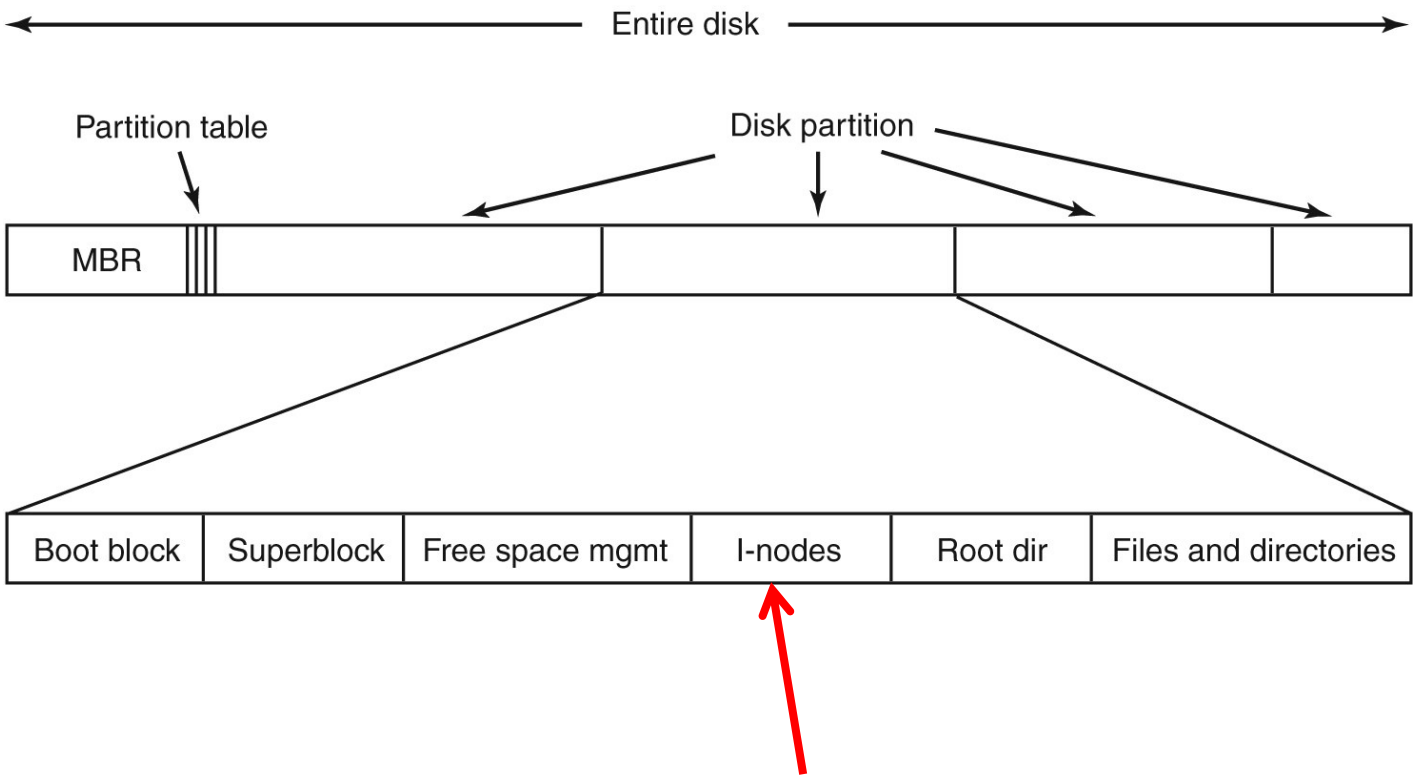

Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories



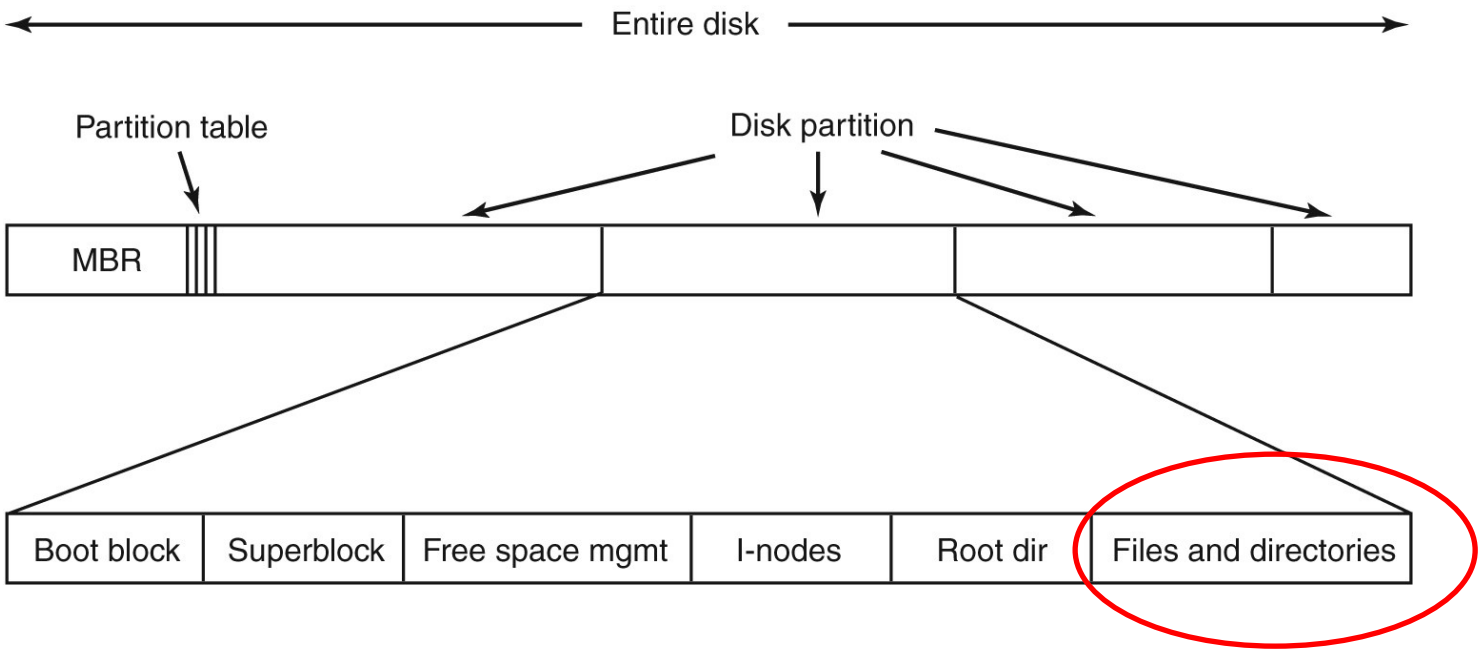
- Contains all key parameters about the file system.
- Is read into memory when computer is booted or file system is touched.



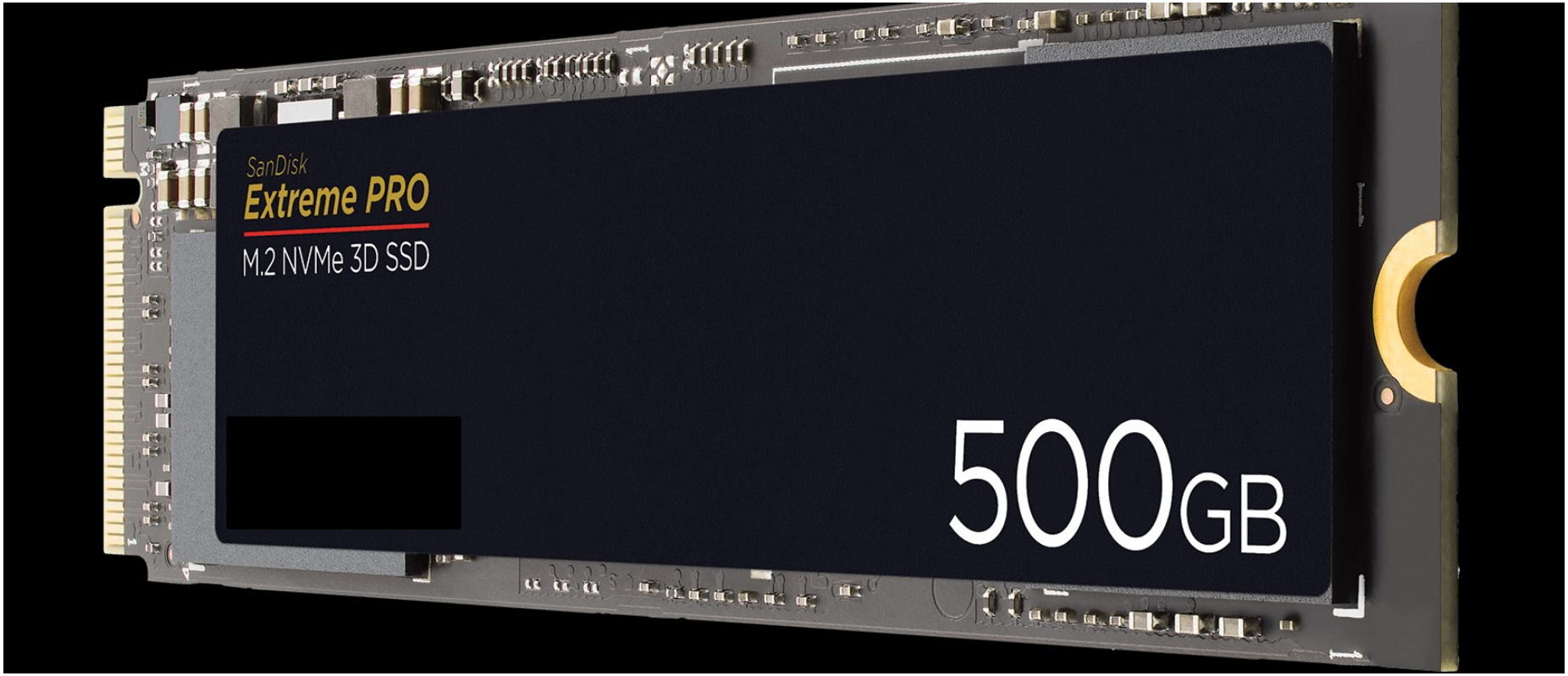
Bitmap or linked list



An array of data structures, one per file, telling about the file



Which disk blocks go with which files?



Solid State Disks (SSD)

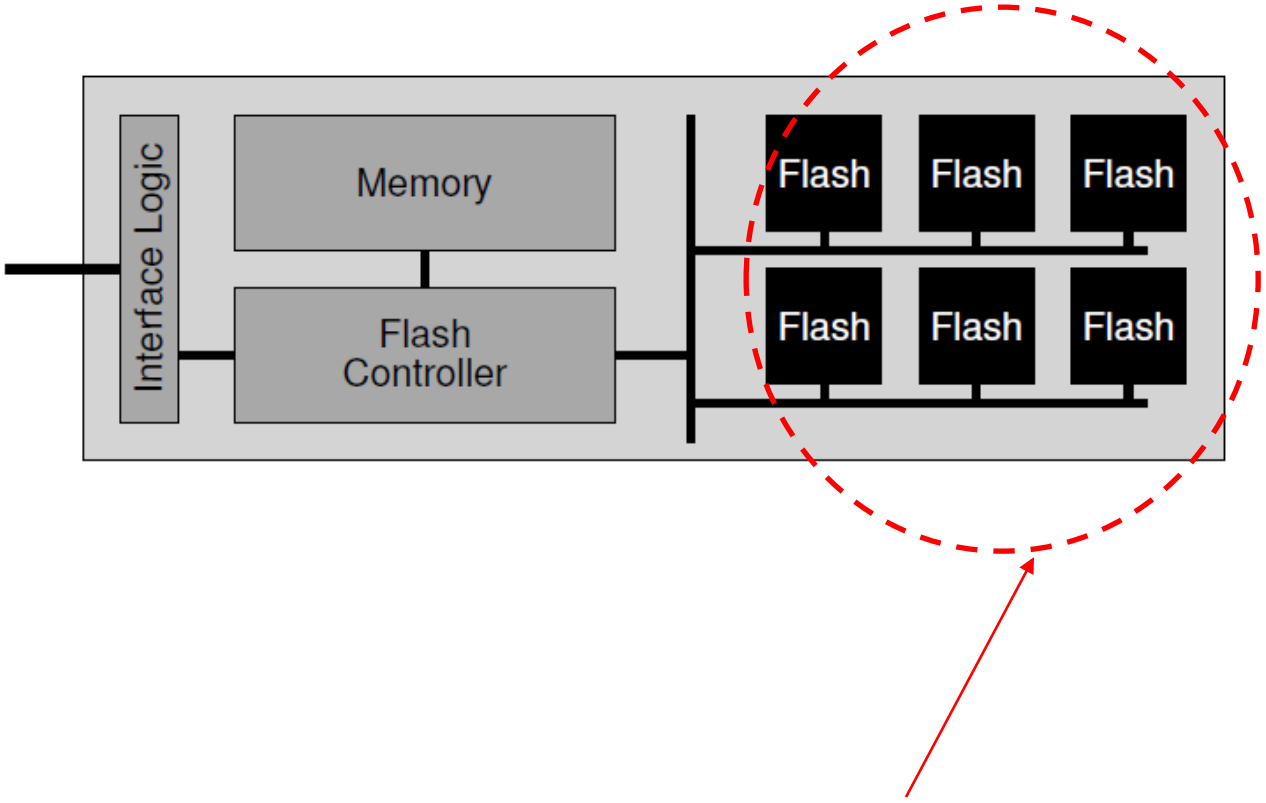
Advantages over traditional Hard Disk Drive (HDD)

- No moving parts, so:
  - no noise
  - Faster start up
- Deterministic read performance
  - The performance does not depend on the location of data.

Disadvantages relative to traditional Hard Disk Drive (HDD)

- More expensive
- Write latency is high
- Limited number of writes (~100,000)
  - after wish, the cell becomes unusable.

Internal Logic



This is where the data are store.
But what is inside those flash packages?

The Flash

- Data saved to a pool of NAND flash.
- Made of transistors, BUT:
  - Unlike designs used in DRAM, the ones used in flash retain their charges even when not powered up → non-volatile.
- Data stored in NAND cells.
  - One cell can store 1 (single-level-cell or SLC), 2 (Multi-Level Cell MLC), 3, or 4 bits.
  - SLC achieves higher performance.
- The NAND flash cells organized as a grid.
  - The entire grid is called a block.
  - Each row of the grid is called a page.

The Flash

- Page size varies from 2KB to 16KB.
- Each block has 128 to 256 pages.
- A block is then 256KB to 4MB,

Conclusions

- The OS provides an interface between. the devices and the rest of the system.
- OS must *expand* as new I/O devices are added.
- For any I/O device the OS uses device driver to deal with device controllers that control the device itself.
  - software: device driver
  - hardware: device controller
  - hardware: the device