Operating Systems
CS 415

Lecture 2:

OS Structure & Interrupts

Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Announcements

- **Suyash Gupta**
  - Office Hours: Thursday, 10-11am, Deschutes 334

- **Nihal Balivada (TA)**
  - Office Hours: Wednesday/ Friday, 11-12pm, Deschutes 335

- **Ranjitha Rani (TA)**
  - Office Hours: Monday /Tuesday, 1-2pm, Deschutes 229

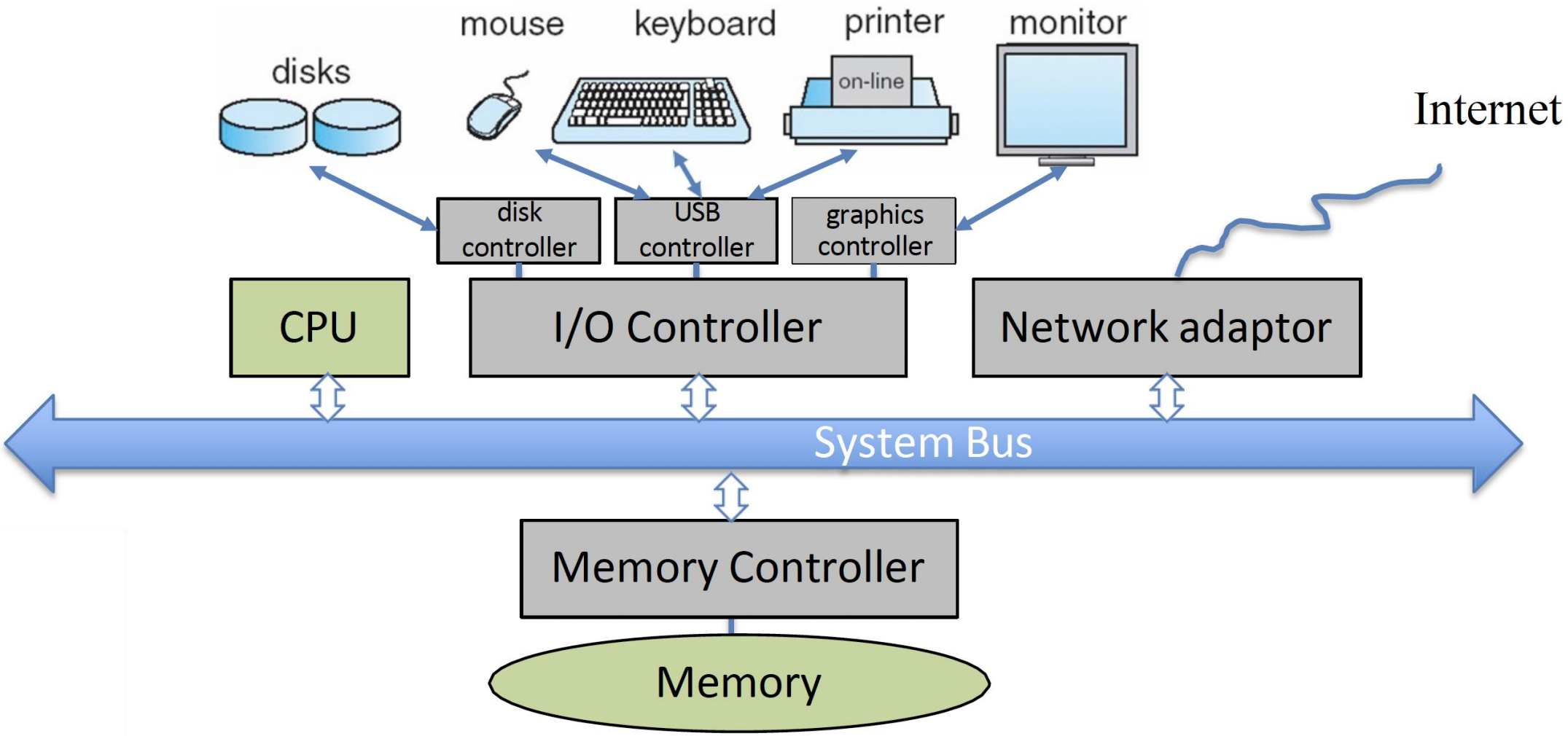
Assignment 1 is Out!

- **Assignment 1 is out!**
- **Please start working and talk to us if you are stuck.**
- **First Lab this Friday!**

Last Class

- What is an Operating System?
- OS overview.

Canonical System Structure



CPU

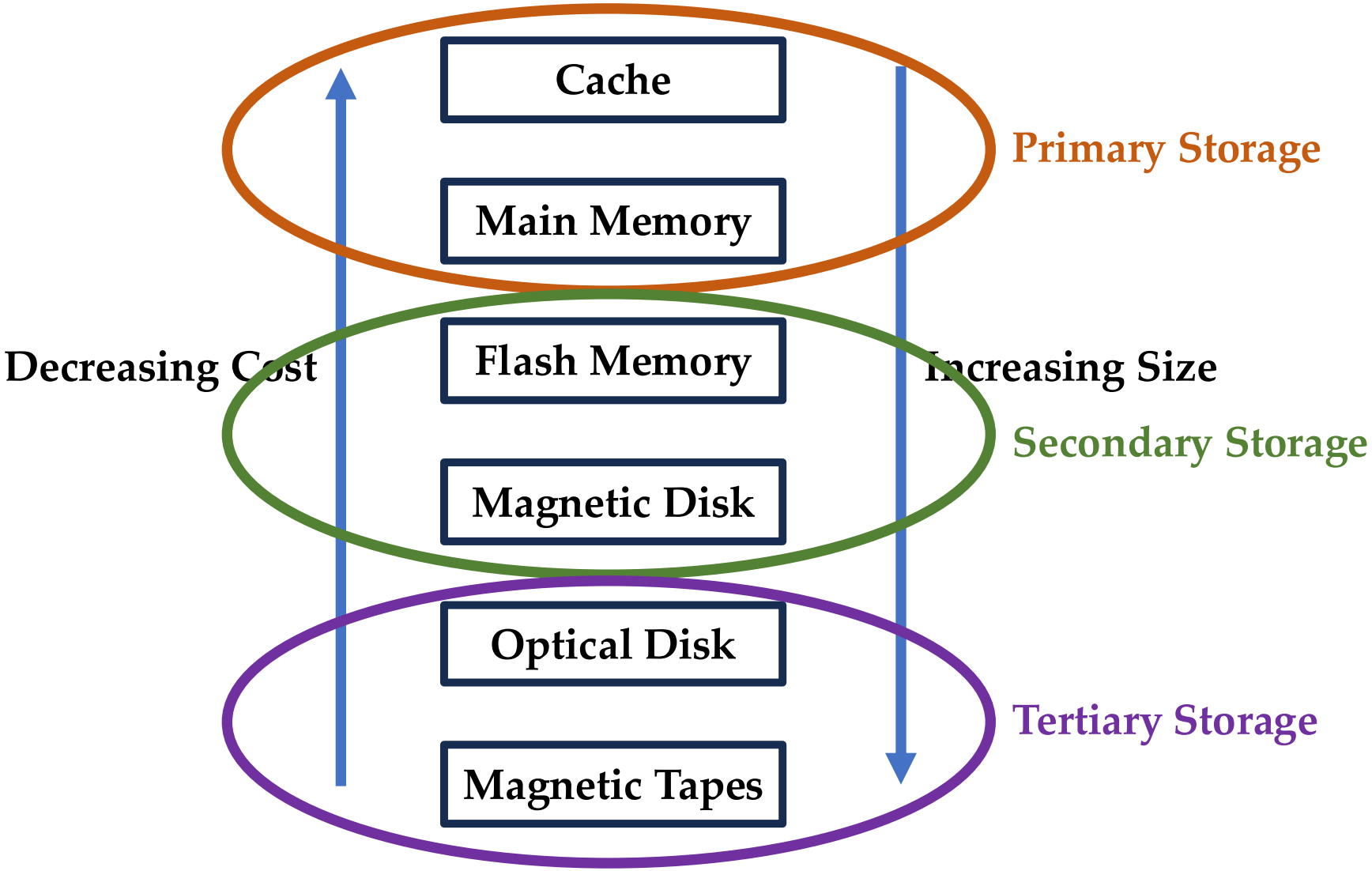
- CPUs are semiconductor devices with digital logic.
  - Instruction set architecture (ISA)
    - specifies the instructions a CPU can execute
    - n-bit architecture (e.g., 32-bit, 64-bit) relates to data width.
  - Arithmetic logical unit (ALU) → Helps perform computations.

CPU

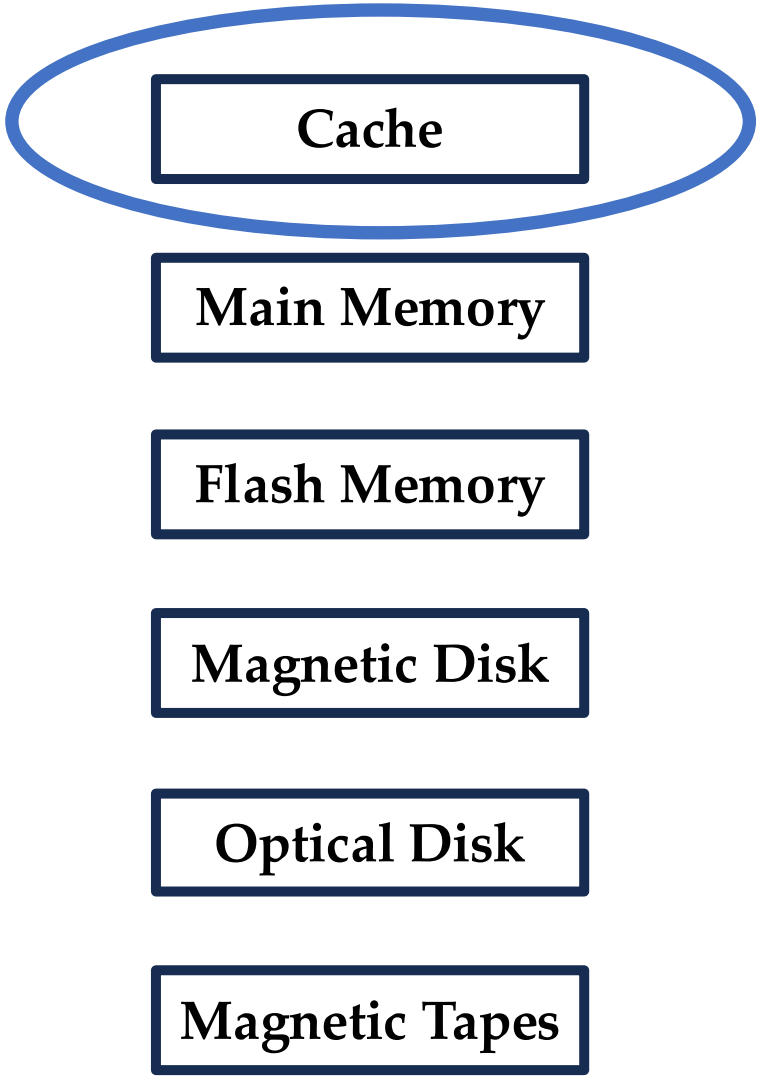
- CPUs are semiconductor devices with digital logic.
  - Instruction set architecture (ISA)
    - specifies the instructions a CPU can execute
    - n-bit architecture (e.g., 32-bit, 64-bit) relates to data width.
  - Arithmetic logical unit (ALU) → Helps perform computations.

- Registers
  - Defined as part of ISA (how many, what type, length, …)
  - CPU's scratchpads for program execution
  - Fastest memory in a computer system → Stores instruction, data, address.
- Clock to synchronizes constituent circuits
- CPU state: Program Counter (PC) + registers values between

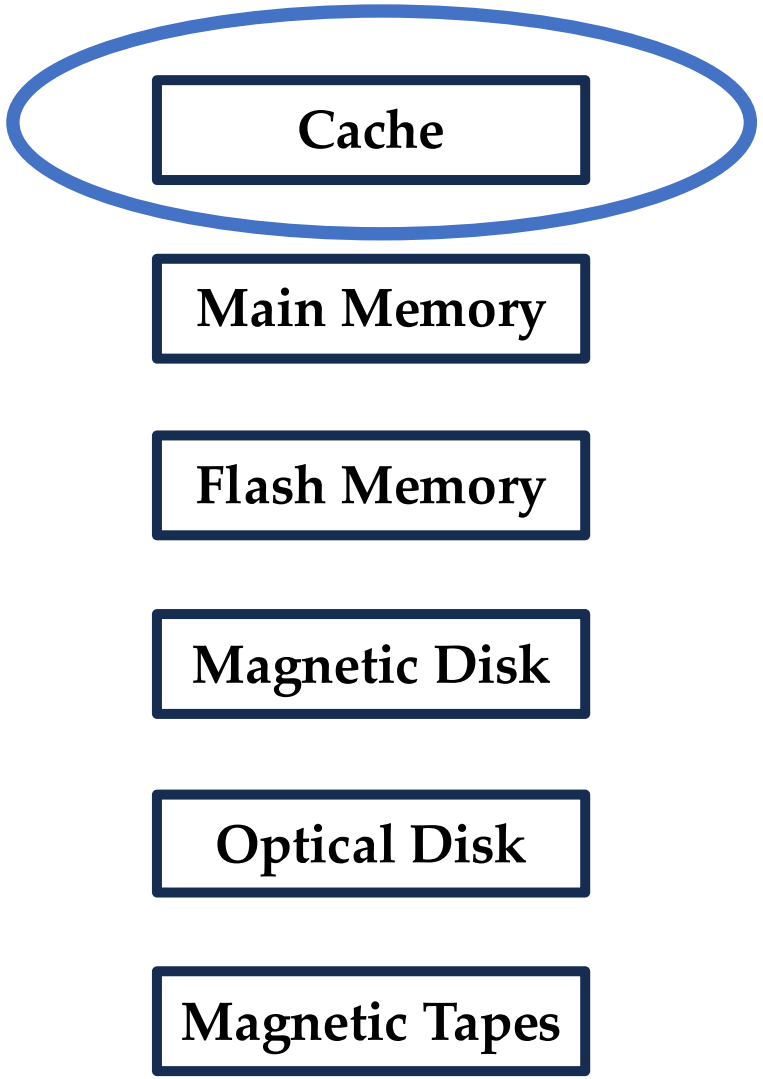
Storage Hierarchy



Storage Hierarchy



Storage Hierarchy



Nearest to the CPU.

Volatile!

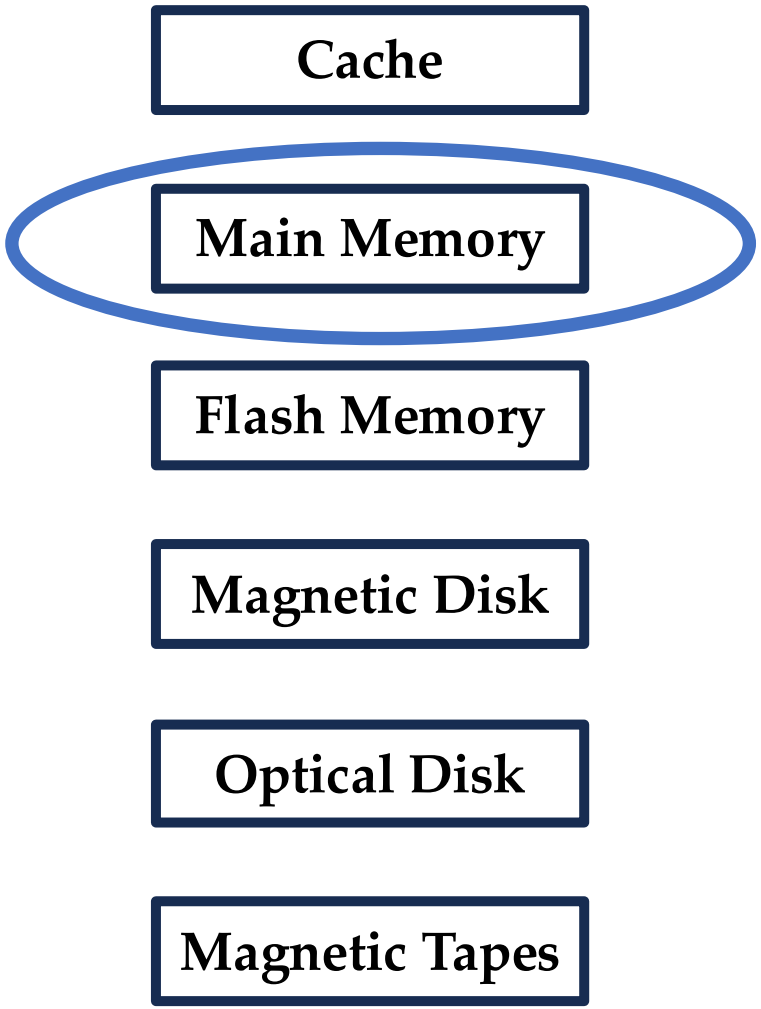
Always accessed first.

Often multiple levels: L1, L2, L3.

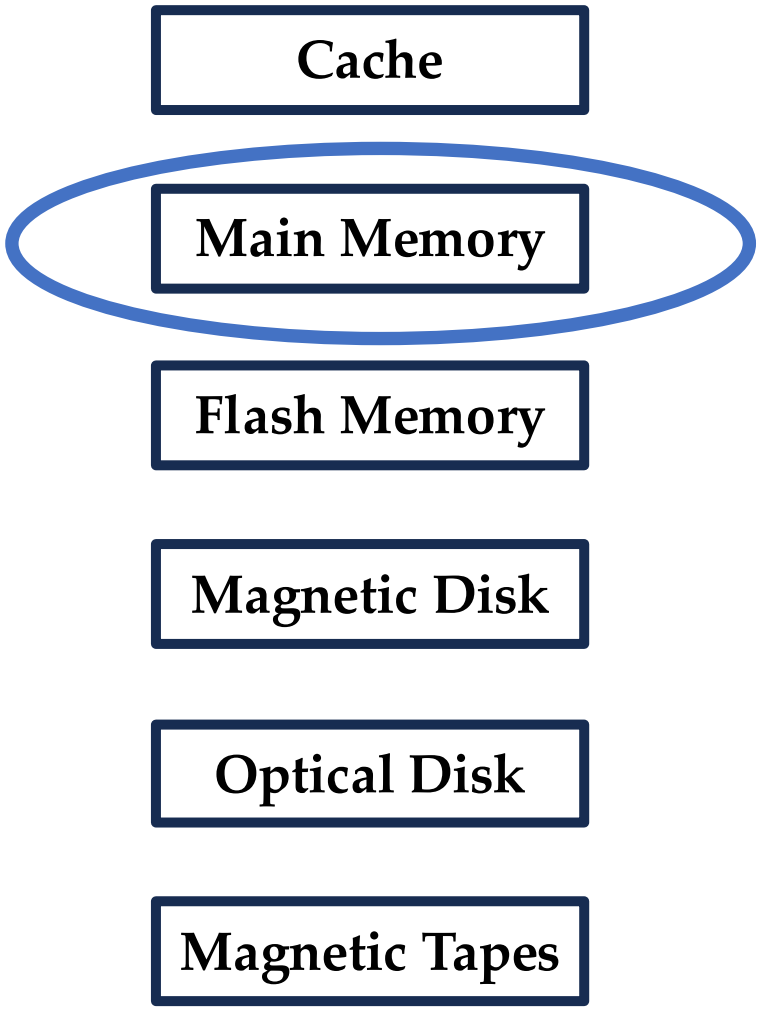
When CPU wants some data → it contacts **cache controller**, which searches for the data.

A good application or system design targets **cache hits**.

Storage Hierarchy



Storage Hierarchy



On cache miss, access memory.

Random Access Memory (RAM)

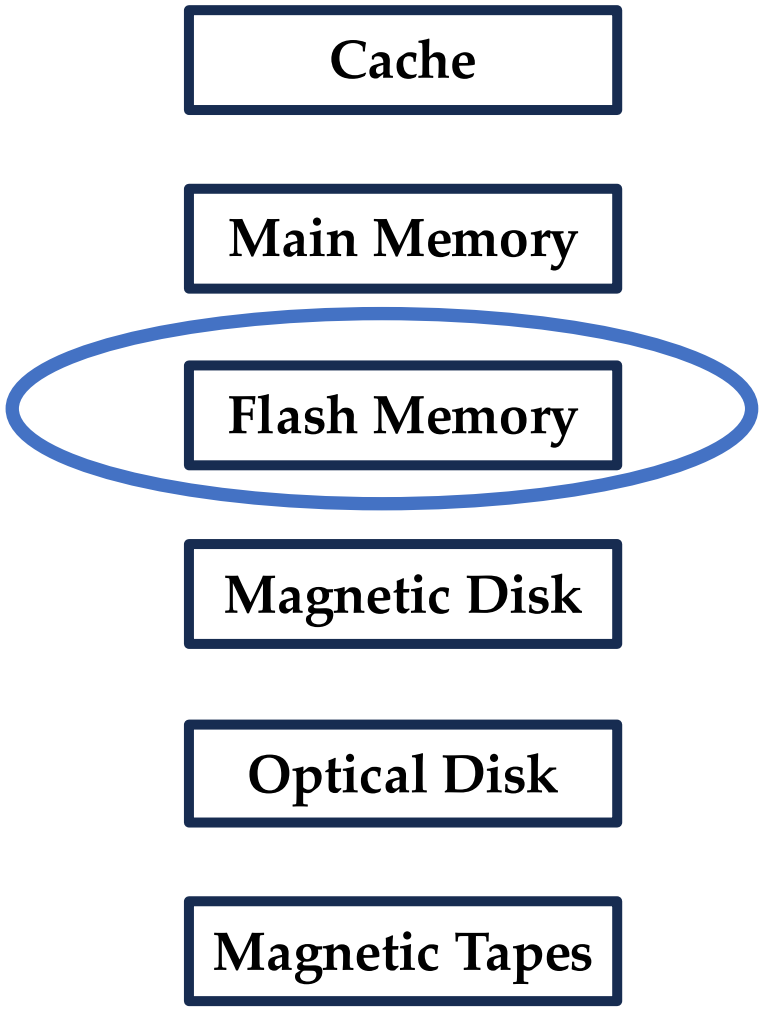
Volatile!

When CPU wants some data, it uses **Memory Management Unit** (MMU) to perform virtual address to physical address translation.

Next, **Memory Controller** fetches the data at the physical address.

Modern systems contact MMU at the same time as L1 cache (saves time).

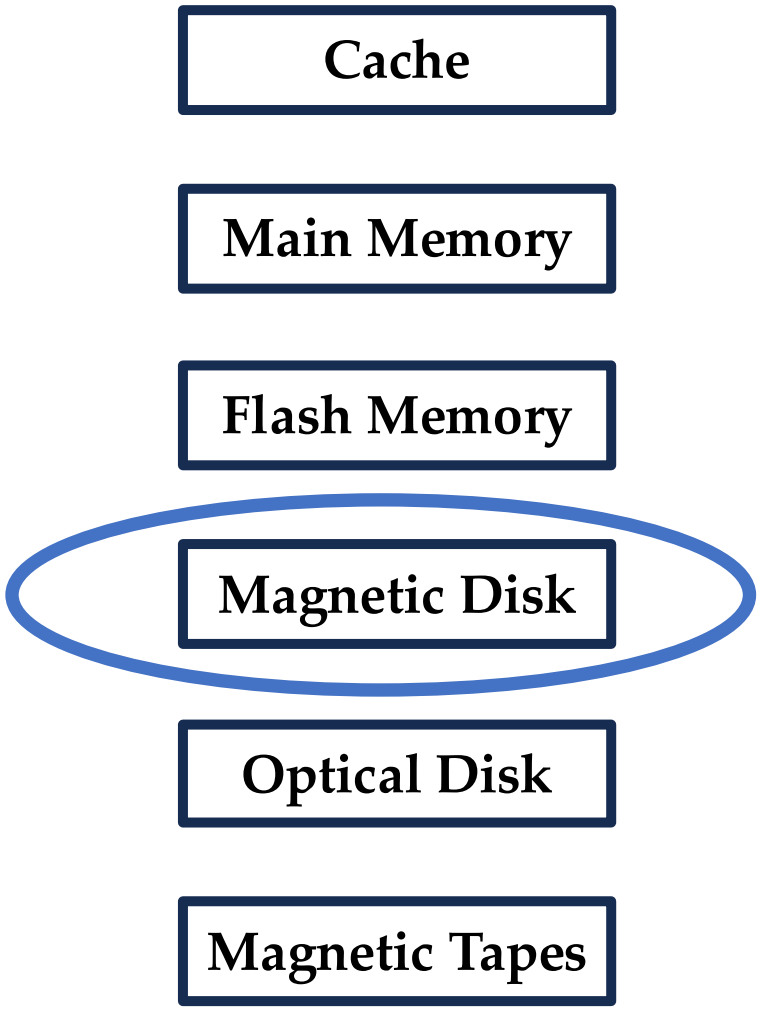
Storage Hierarchy



USB stick or external memory.

No data loss on turning off power.

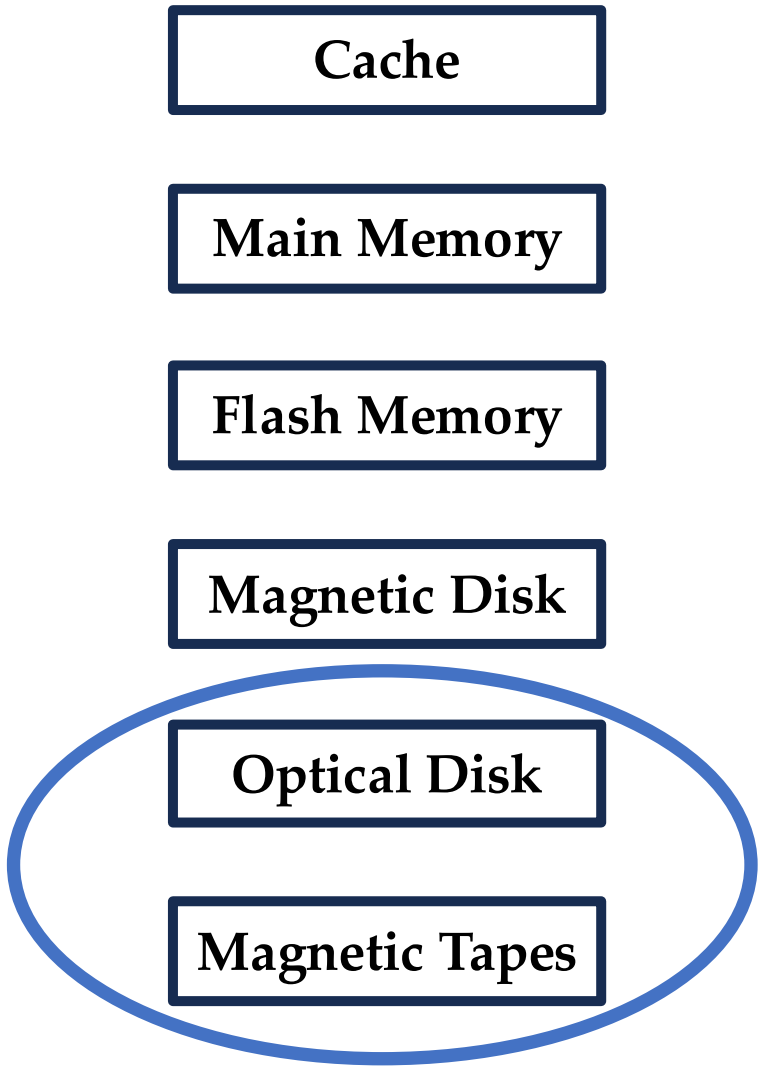
Storage Hierarchy



Useful for long-term storage of the database.

At the time of processing, CPU moves data from magnetic disk to the main memory.

Storage Hierarchy

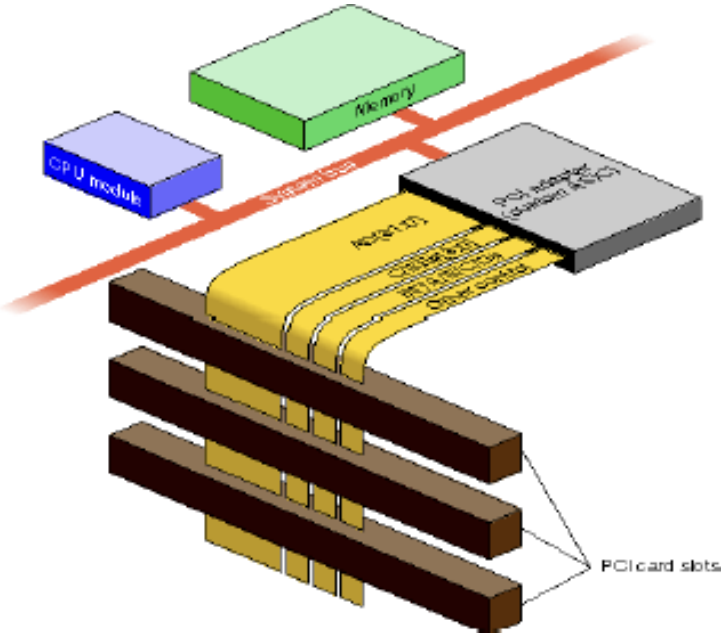


Older storage media; not common everywhere.

Interconnects

Interconnects

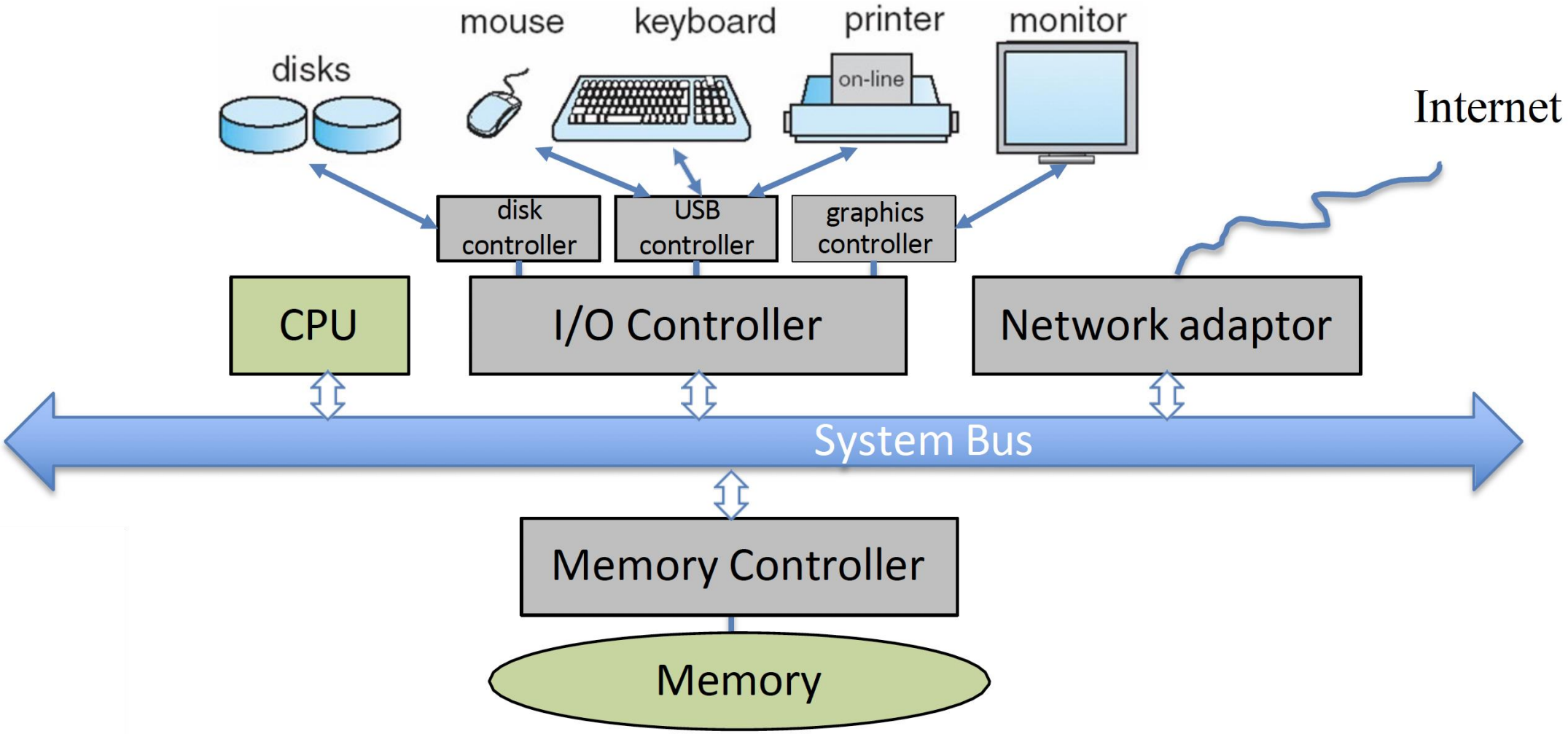
- A **bus** is hardware interconnect for supporting the exchange of data, control, signals, etc.
  - Physical specification
  - Defined by a protocol
  - Data and control arbitration



- System bus for CPU connection to memory.
- PCI bus for devices:
  - Connects CPU-memory subsystem to:
  - fast devices
  - expansion bus that connects slow devices

- Other device “bus” types: SCSI, IDE, USB.

Hardware Controllers



Hardware Controllers

- A **bus** connects to many devices over shared links
  - Disks, network cards, USB ports, etc.

- Controllers are the hardware endpoints on these buses.

Hardware Controllers

Common hardware controllers:

- **Memory controller (IMC)**
  - Lives inside the CPU package.
  - Handles all reads/writes between cores and DRAM; schedules memory accesses, refresh, etc.
- **PCIe controller**
  - Routes PCIe traffic between device controllers and the CPU/memory.
  - Often partly on the CPU (for GPU/NVMe) and partly in the chipset.
- **Storage controllers (disk controllers)**
  - Speak to physical disks/SSDs and present them to the OS as block devices.

Hardware Controllers

Common hardware controllers:

- **USB controller**
  - Manages USB ports and devices (mouse, keyboard, flash drive, etc.).
  - Handles all low-level USB protocol details.
- **Network controller (NIC)**
  - Talks to Ethernet/Wi-Fi hardware, sends/receives packets.
- **Interrupt controller (PIC/APIC/GIC)**
  - Collects interrupt requests (IRQs) from all the other controllers and routes them to CPU cores via interrupt vectors.

OS and Controllers

OS and Controllers

- OS views each controller as a device with registers mapped into memory.

- The OS kernel has a **driver** for each type of controller:
  - The driver is software that knows:
  - Which registers to write to configure the controller?
  - How to set up DMA so the controller transfers data directly between device and RAM?
  - How to handle the controller's interrupts?

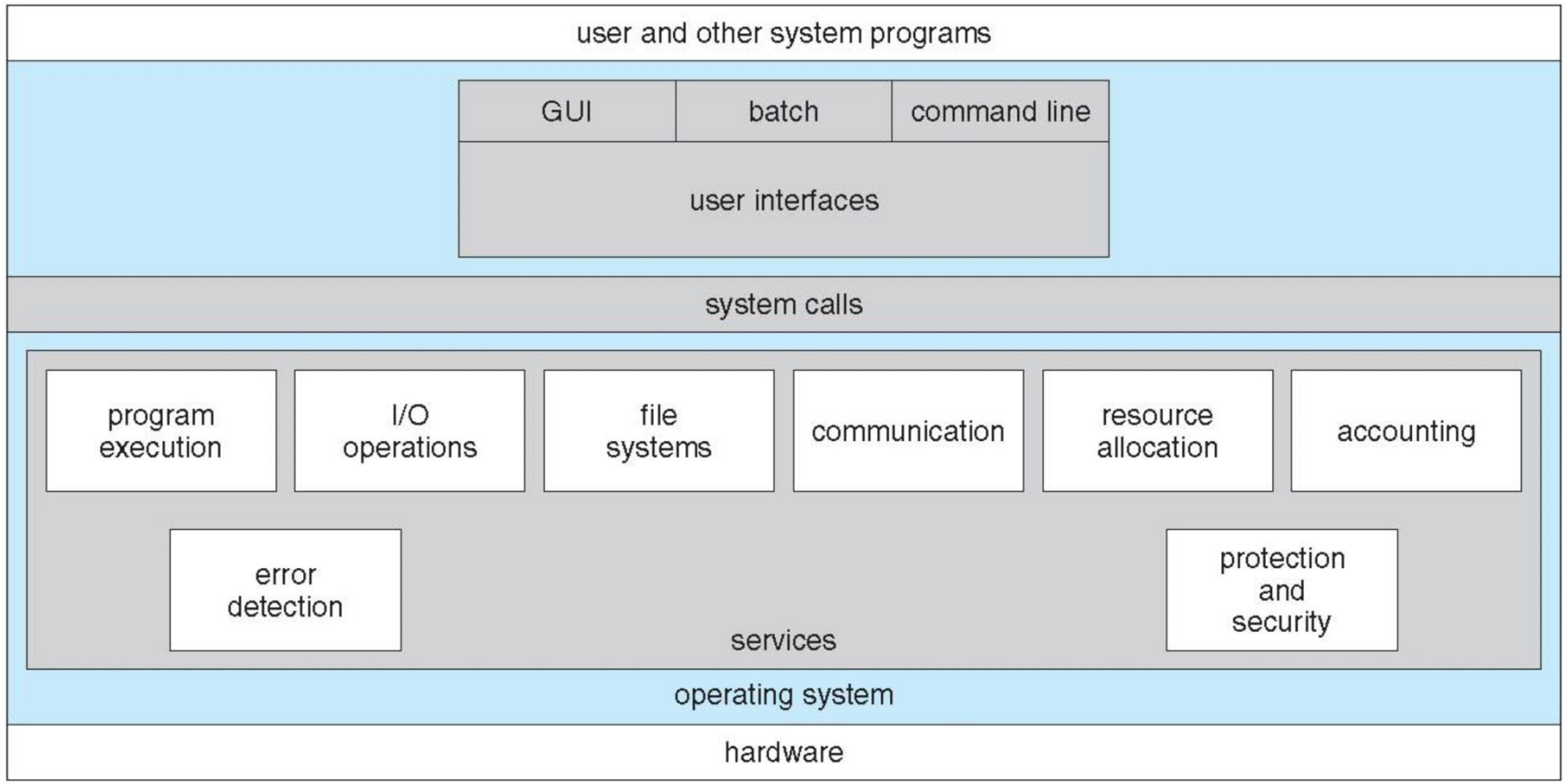
OS and Controllers

- OS views each controller as a device with registers mapped into memory.

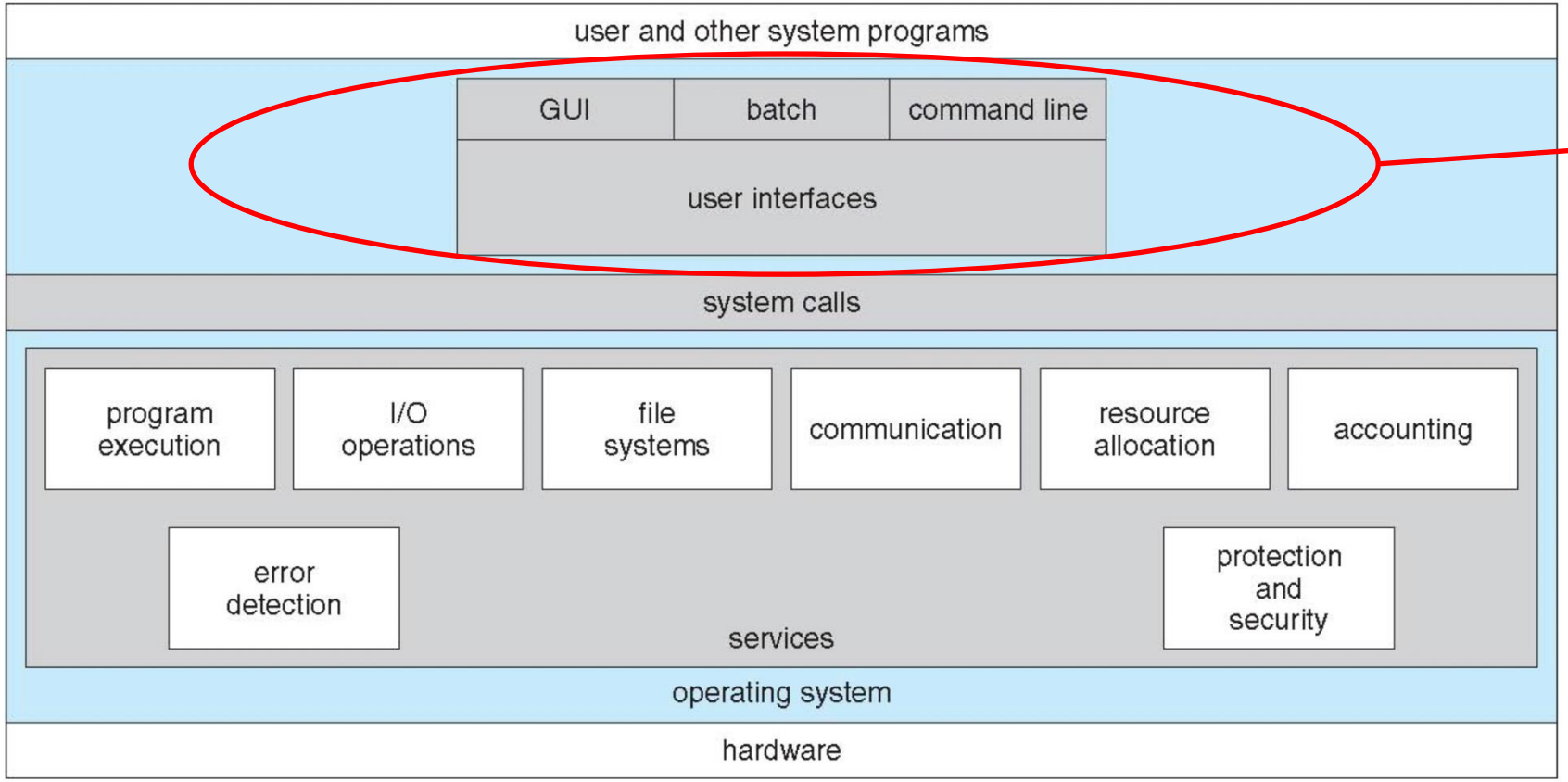
- The OS kernel has a **driver** for each type of controller:
  - The driver is software that knows:
  - Which registers to write to configure the controller?
  - How to set up DMA so the controller transfers data directly between device and RAM?
  - How to handle the controller's interrupts?

- When the OS **boots**, it:
  - Enumerates devices on buses (PCI/PCIe, etc.).
  - Finds controllers (e.g., "this is a SATA controller at PCI slot X").
  - Loads/initializes the appropriate controller driver for each one.

OS Services

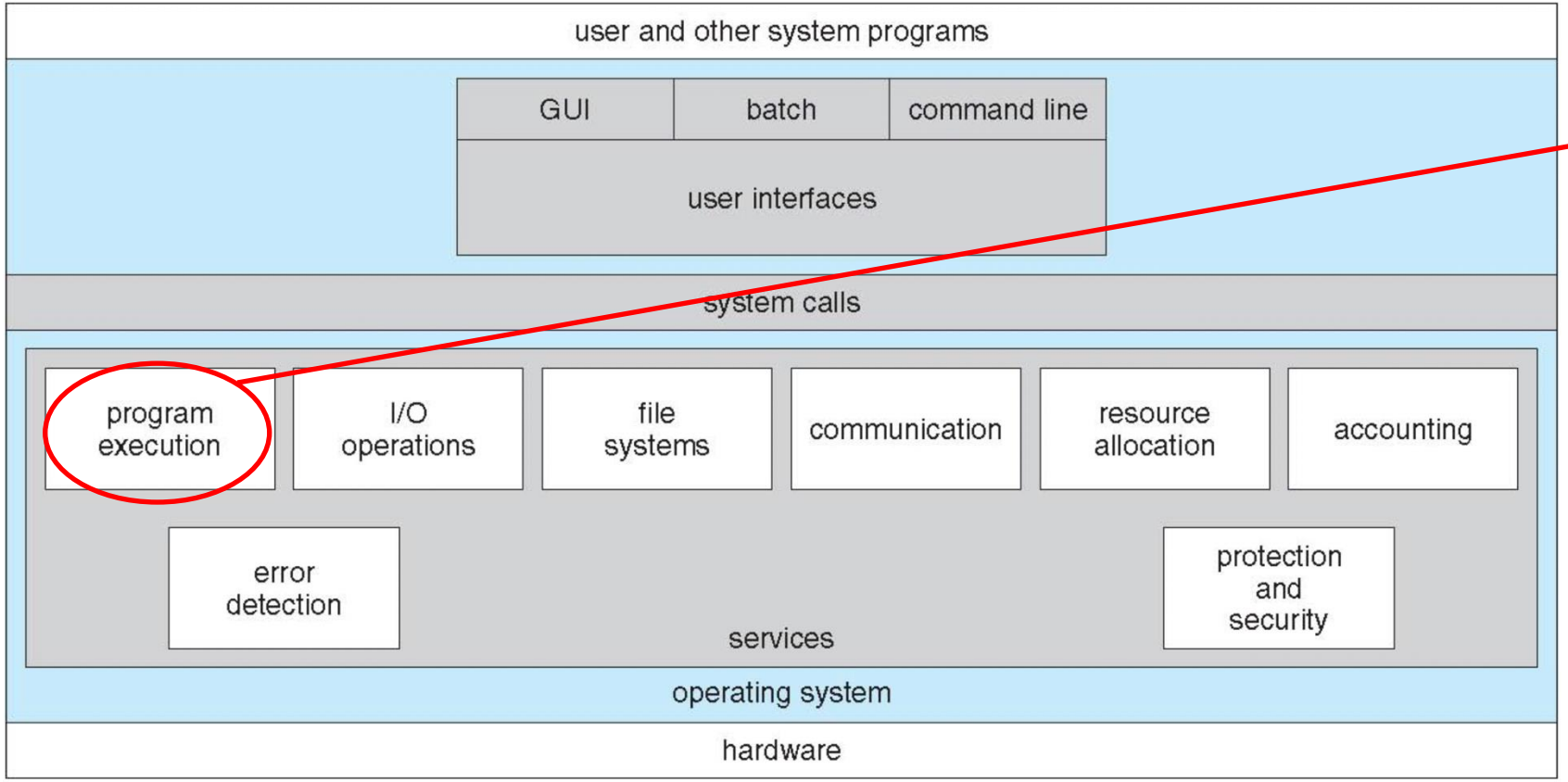

user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

OS Services



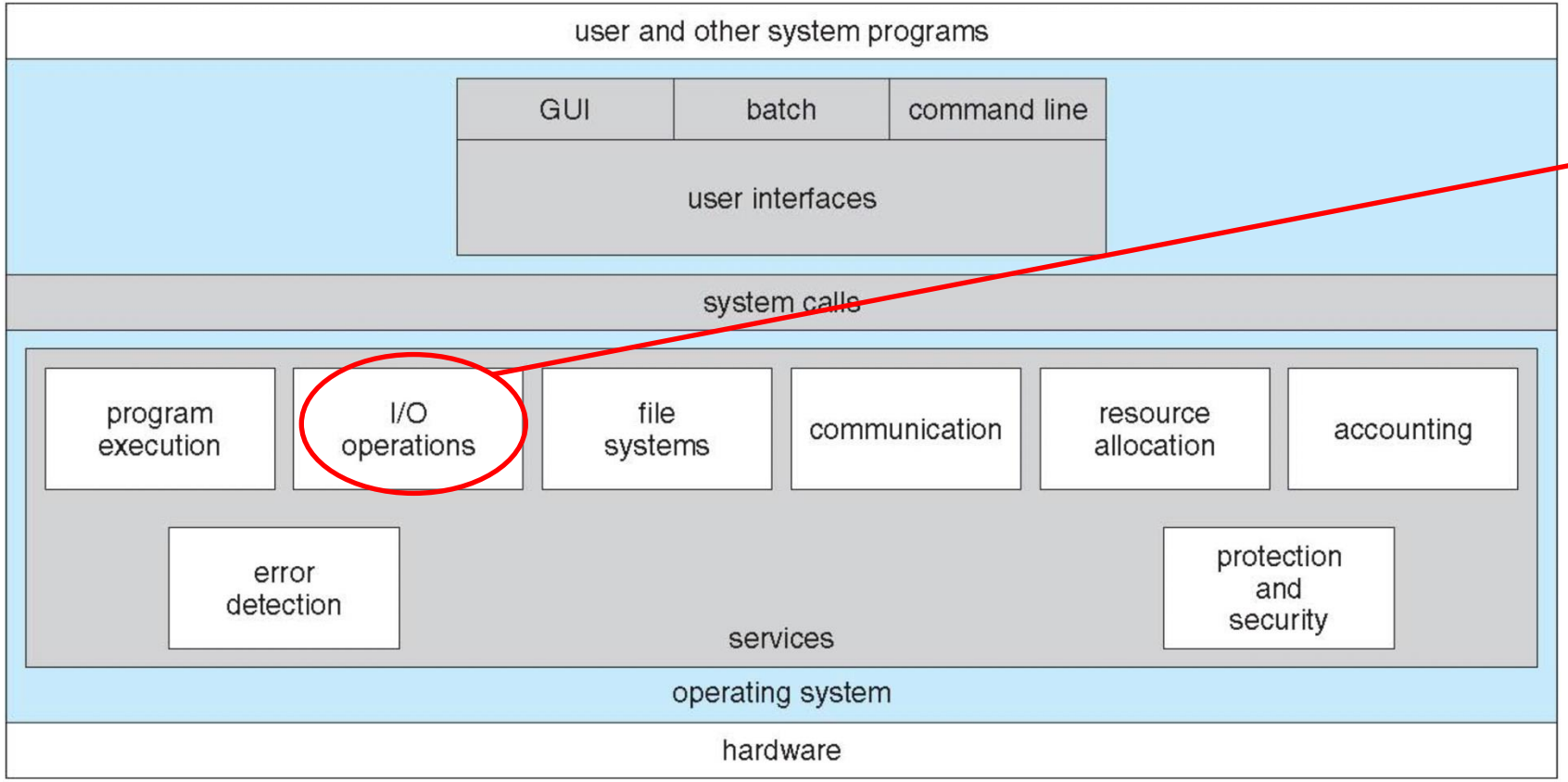
Allows you to easily use the computer system.

OS Services



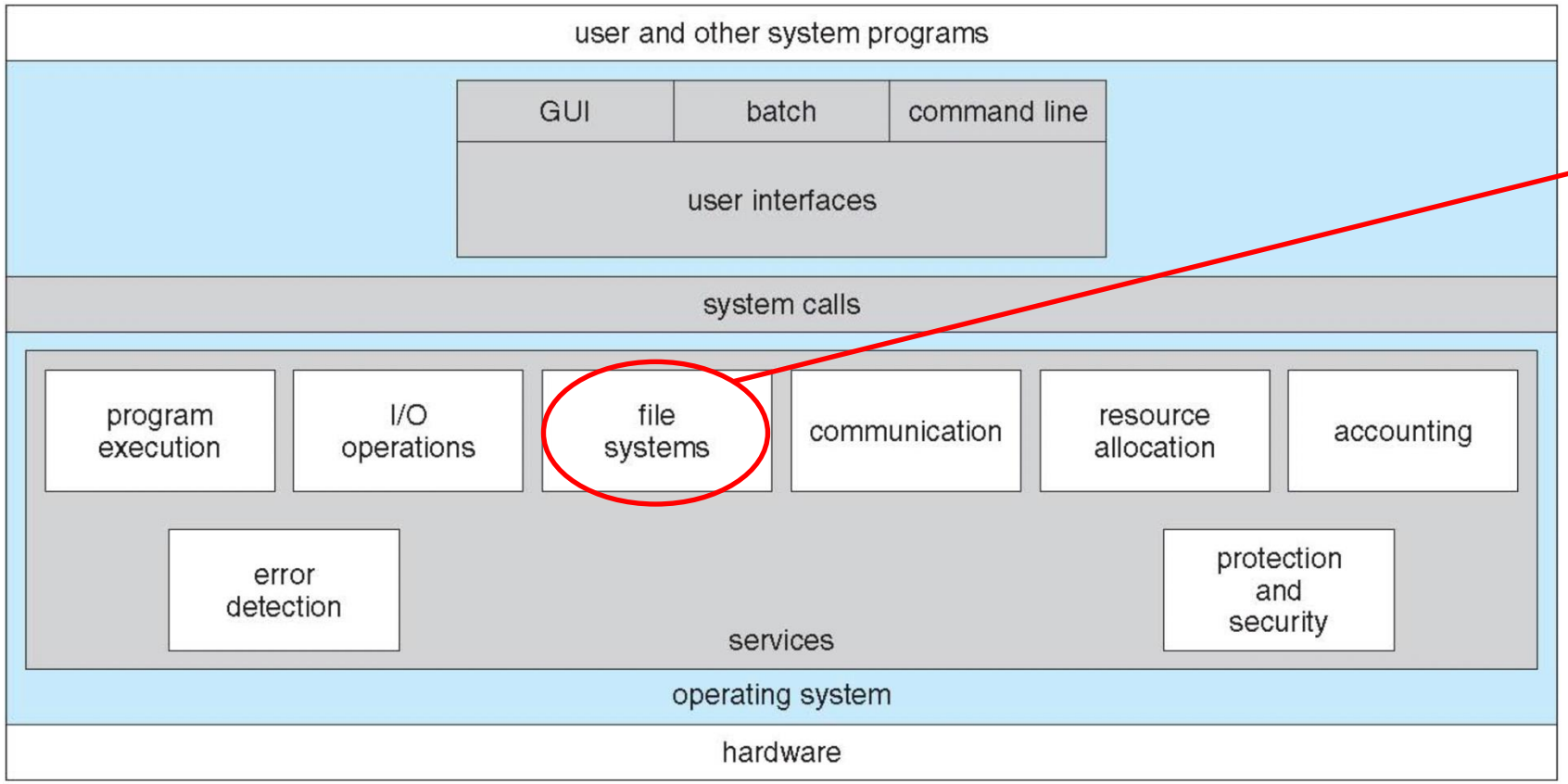
Execute the user program.

OS Services



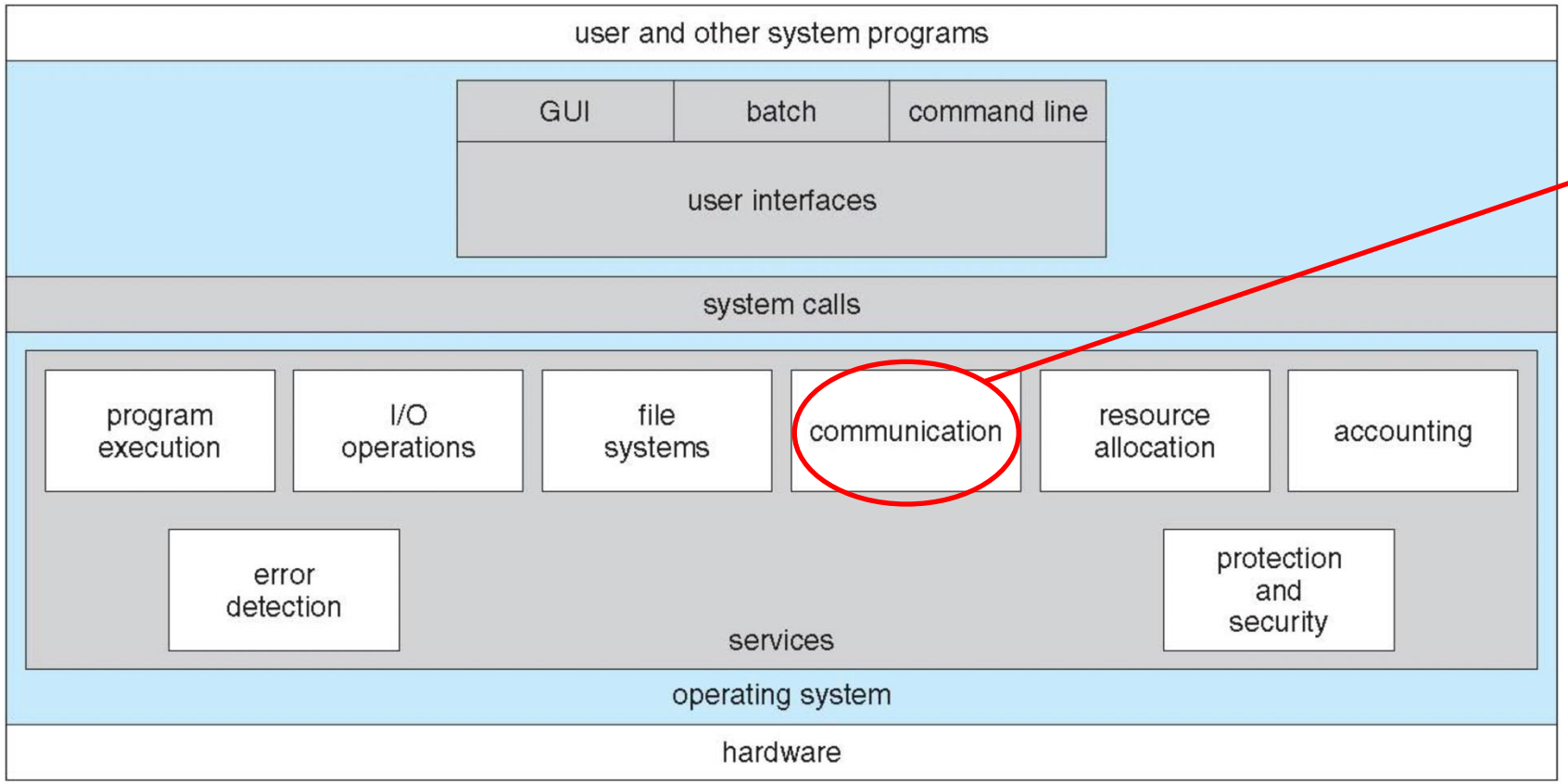
Interacting with I/O devices.

OS Services



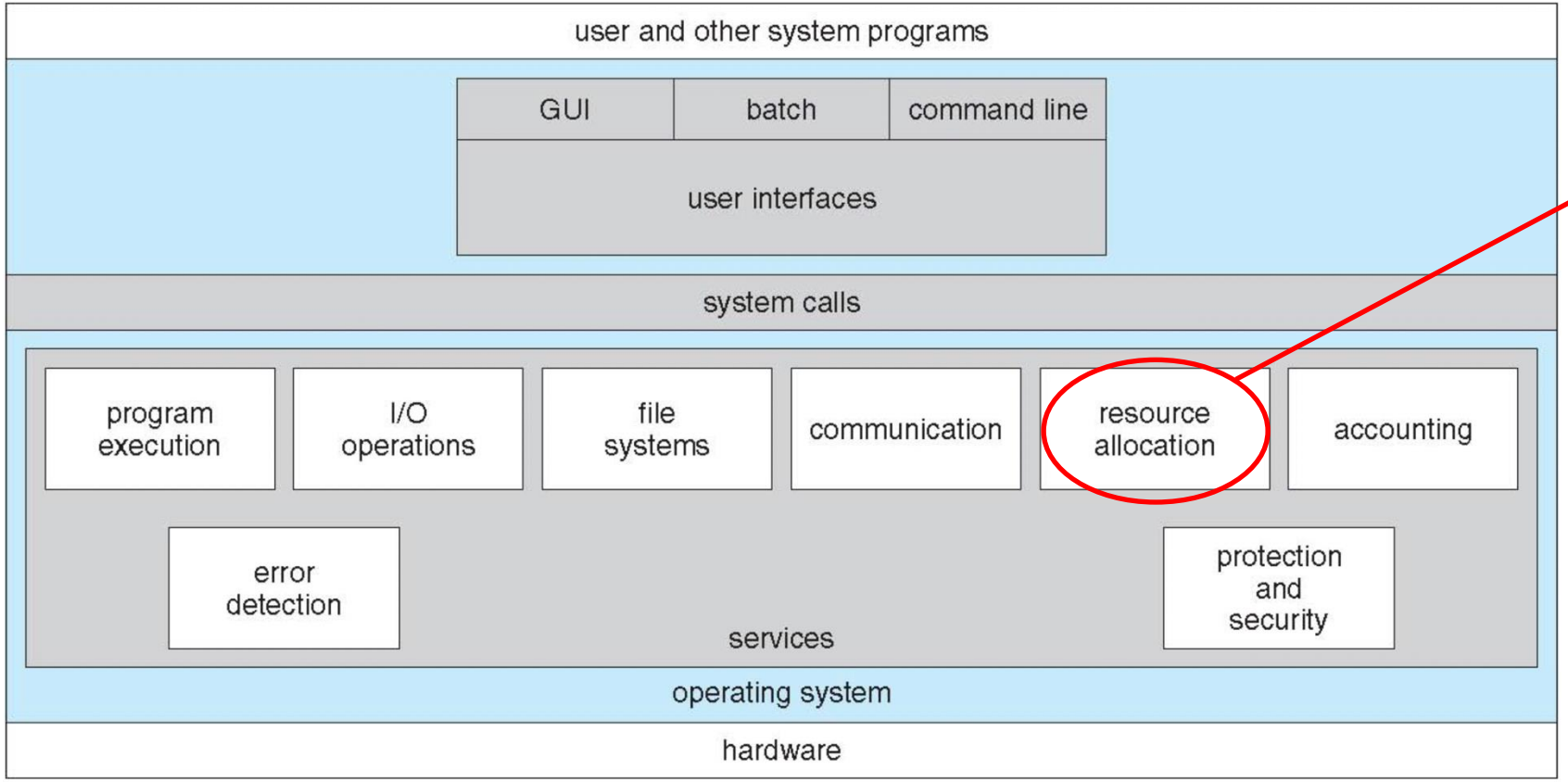
Create, delete, update, file manipulations.

OS Services



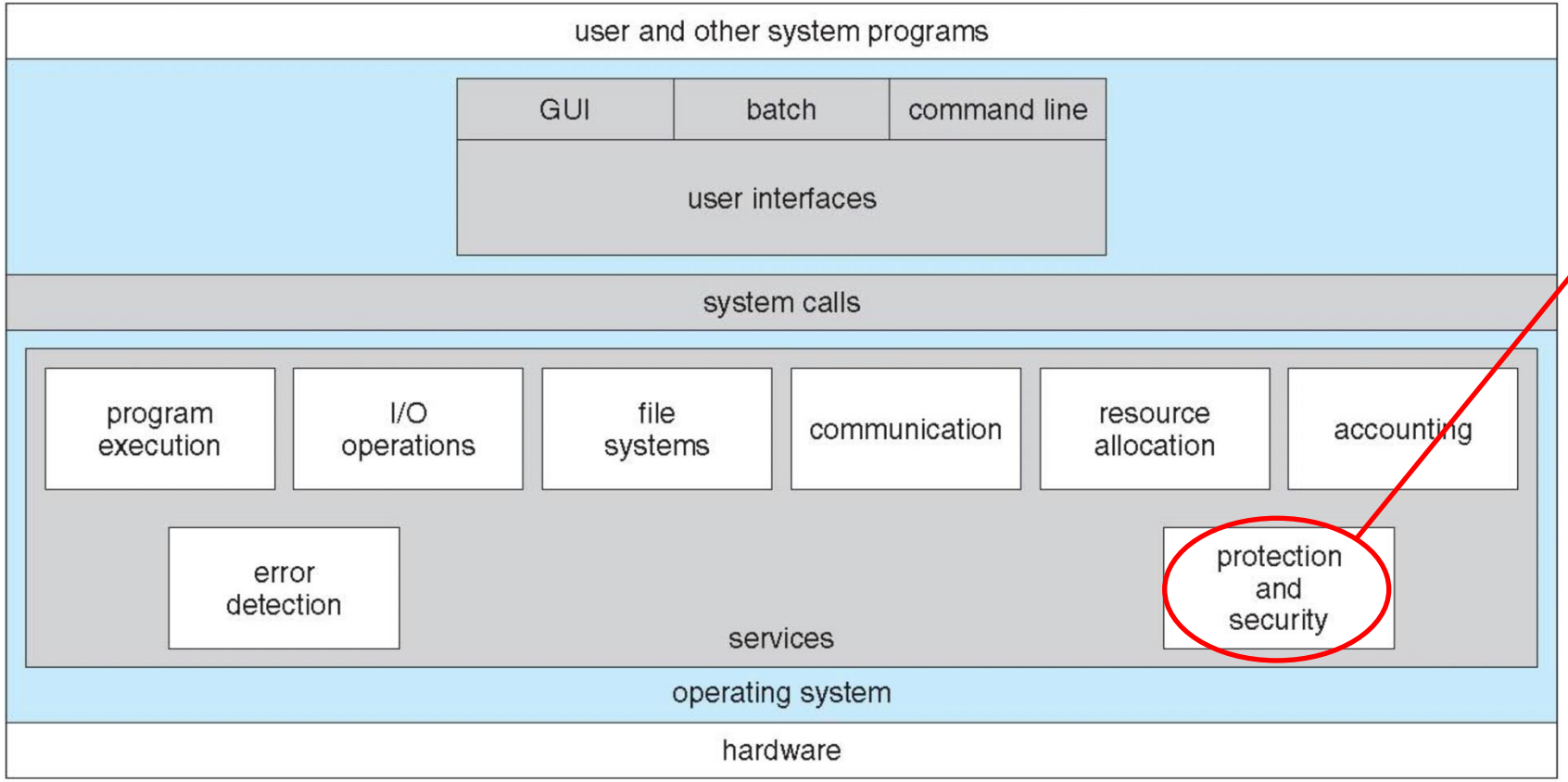
Network communication

OS Services



Scheduling, concurrency

OS Services



Protecting users and resources

Kernel vs. User Mode

Kernel vs. User Mode

- A modern CPU has at least two execution modes.
  - Indicated by status bit in a protected CPU register.
  - OS kernel runs in privileged mode → kernel or supervisor mode.
  - Applications run in non privileged mode → normal or user mode.

- Certain operations need to run in privileged mode.
  - Must switch the processor to privileged mode to allow processor to execute certain privileged instructions.
  - Example: interfacing with devices
  - Example: error handling

- OS can switch the processor back to user mode.

Protected Instructions

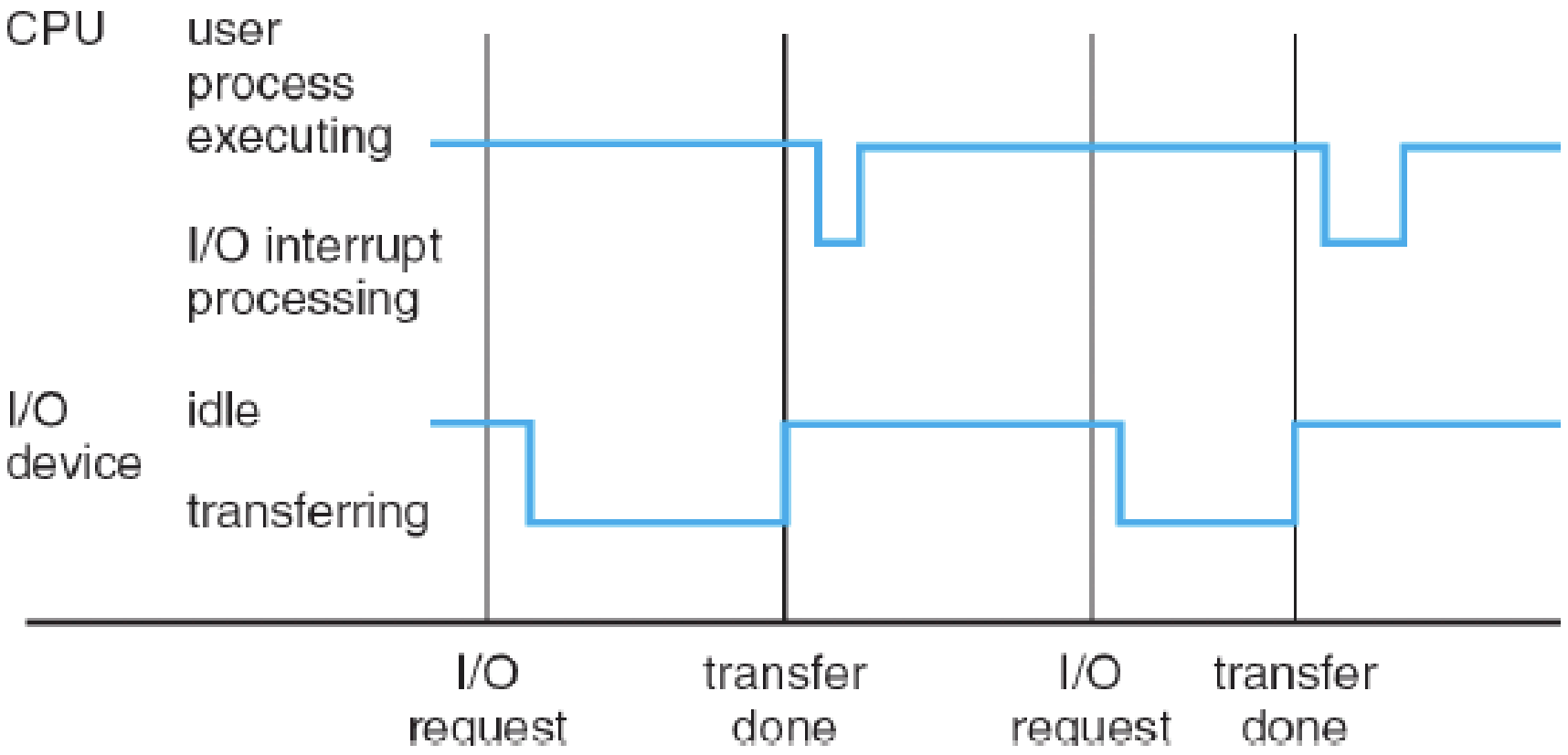
Protected Instructions

- A user program/process may run some instructions that require privilege:
  - Direct access to I/O
  - Modify page table pointers or the TLB
  - Enable and disable interrupts
  - Halt the machine
  - Access sensitive registers or perform sensitive operations

- These instructions are permitted only in privileged mode.
  - Otherwise, random user programs could crash the machine

Interrupts

Interrupt Timeline



System Call / Interrupts / Exceptions

System Call

- A software-triggered entry into the kernel.
- **Explicitly** requested by the program.
- Typically done via a special instruction like `syscall`, `svc`, or `int 0x80` on x86.

Trap/Exceptions

- Illegal or invalid events caused by some instructions.

- Example: divide-by-zero, page fault, and invalid instruction.

System Call vs Trap

- Both are synchronous events → triggered due to an instruction.
- Think of “system call” as why, and “trap/exception” as how.
- You dial 911 on purpose → system call (you intentionally ask the OS for help).
- Your smoke detector auto-calls 911 when it senses smoke → that’s a fault/exception (hardware detected a problem and forced the OS to step in).
- Both end up reaching “911” (the kernel), but the cause is different

Hardware Interrupts

- Generated by an external device.
- Example: timer, disk, NIC, keyboard.
- Asynchronous!
- Can arrive between any two instructions, independent of what the current program is doing.
- The CPU finishes the current instruction, then jumps into the kernel's interrupt handler.

How to handle Interrupts?

Interrupt Handling

- An **interrupt/exception descriptor table (IDT)** contains:
  - Entries (or interrupt vectors) for different interrupts.
  - One single table, all cases (system calls, traps, hardware interrupts…)
  - IDT may include extra metadata per entry (gate type, privilege level, etc.).

- An **interrupt vector** is a pointer (address) to the handler routine for a given interrupt/exception number.

Interrupt Descriptor Table

	Interrupt Type	Handler
1	Page Fault	0x000244
2	…	
3	Timer Interrupt	0xAEEEFB
4	…	
5	syscall	0x899473

Address where the interrupt's handler exists.

Interrupt Request (IRQ)

Interrupt Request (IRQ)

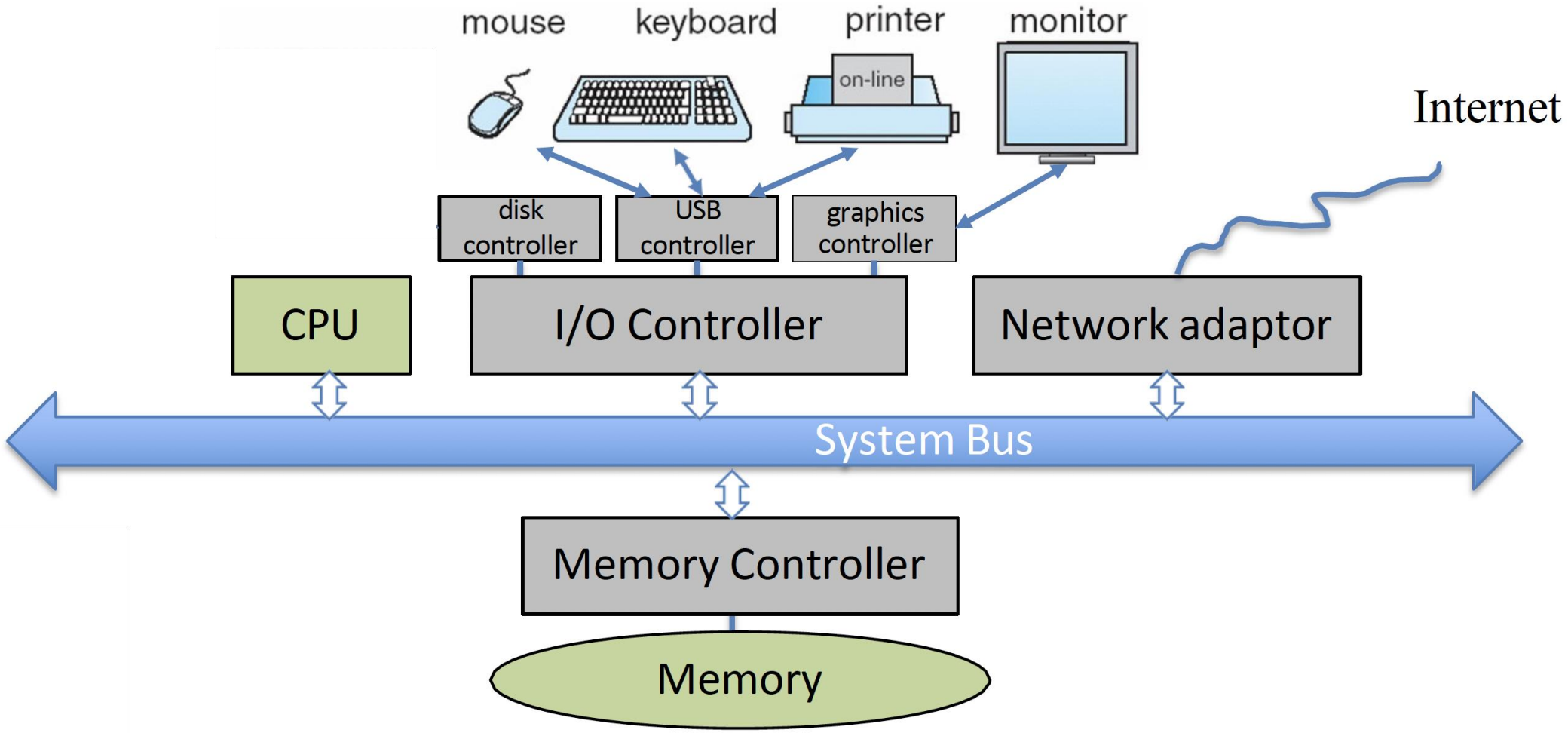
- A signal from a device to the CPU meaning.
- It has a number (IRQ 0, 1, 14,…), which OS uses to identify the source.
  - OS maintains a table of different IRQ's.
- A device triggers an interrupt line → the physical IRQ signal.
  - The signal goes into the interrupt controller, and then to the CPU.
- The OS treats "IRQ N fired" as <*some controller's*> interrupt happened and runs the handler registered for IRQ N.

Interrupt Controller

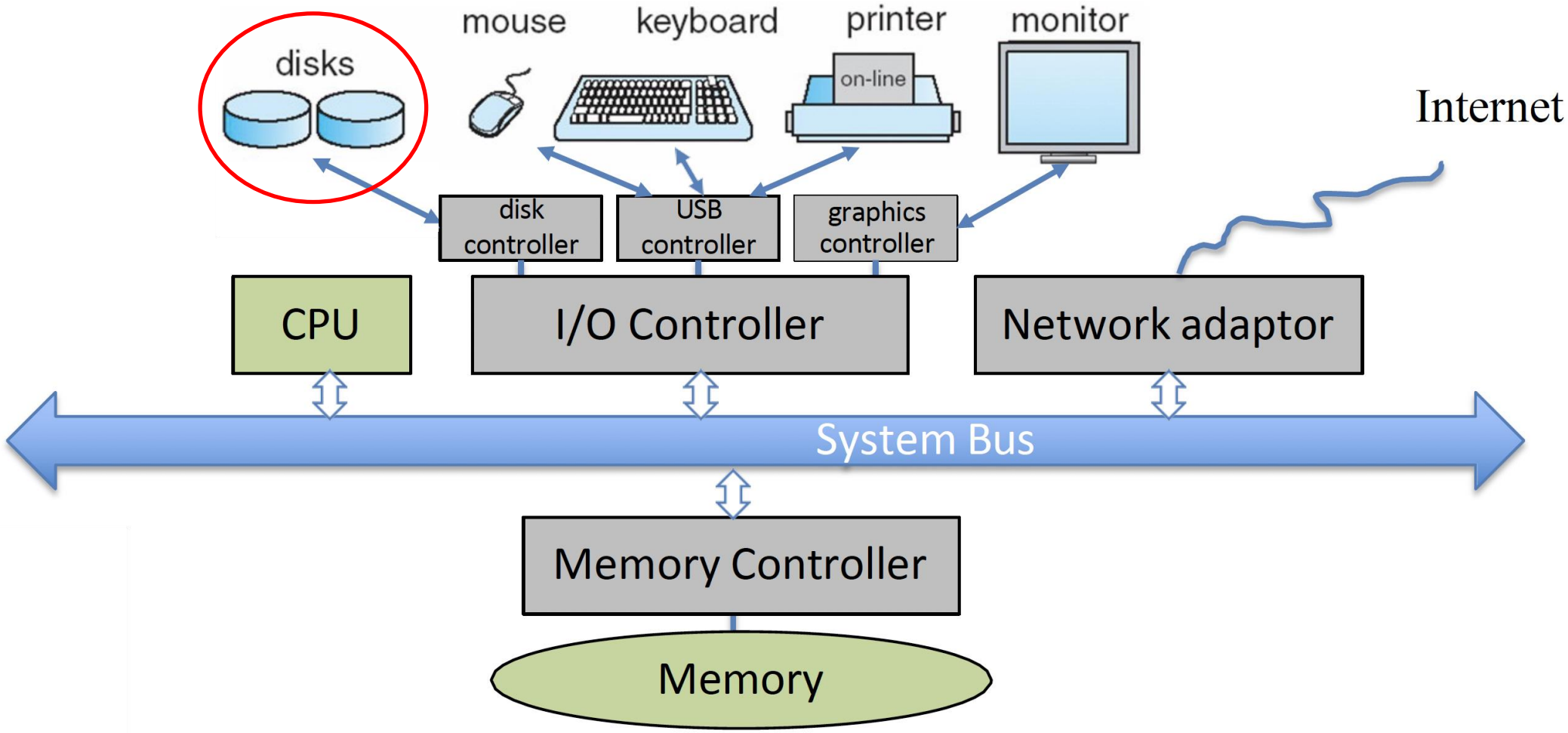
Interrupt Controller

- The interrupt controller is a hardware block that:
  - Has multiple IRQ inputs from devices.
  - Decides which IRQ is pending, its priority, and whether it's allowed.
  - Tells the CPU "you have an interrupt, use vector V" when ready.
- It is the hardware traffic cop for IRQ signals.

Case Study I: New Device Plugged In



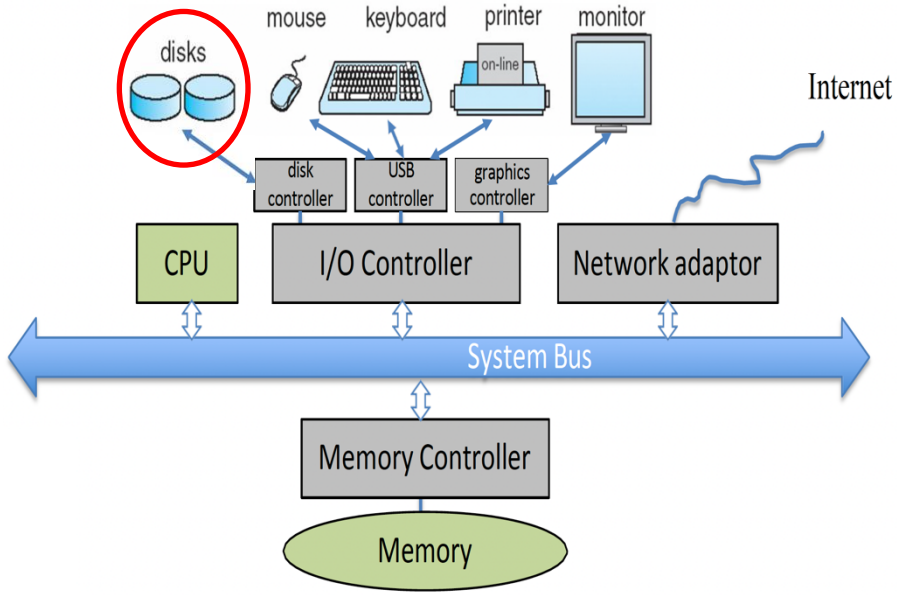
Case Study I: New Device Plugged In



Case Study I: New Device Plugged In

Step 1: Plugging in

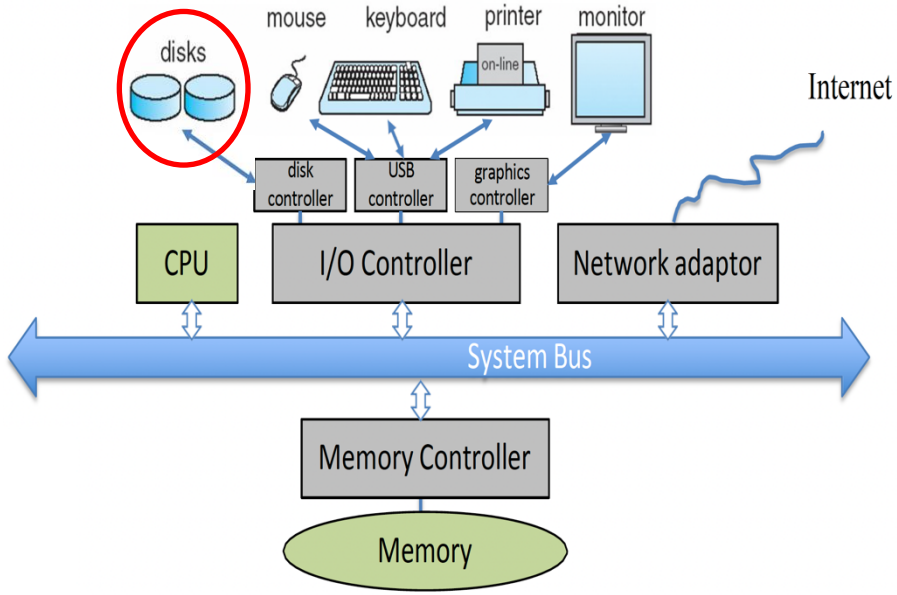
- You plug in a device to a port on some bus controller.
- The disk/bus controller detects a new disk device is present.
- It raises an interrupt.



Case Study I: New Device Plugged In

Step 2: OS discovers and identifies the device

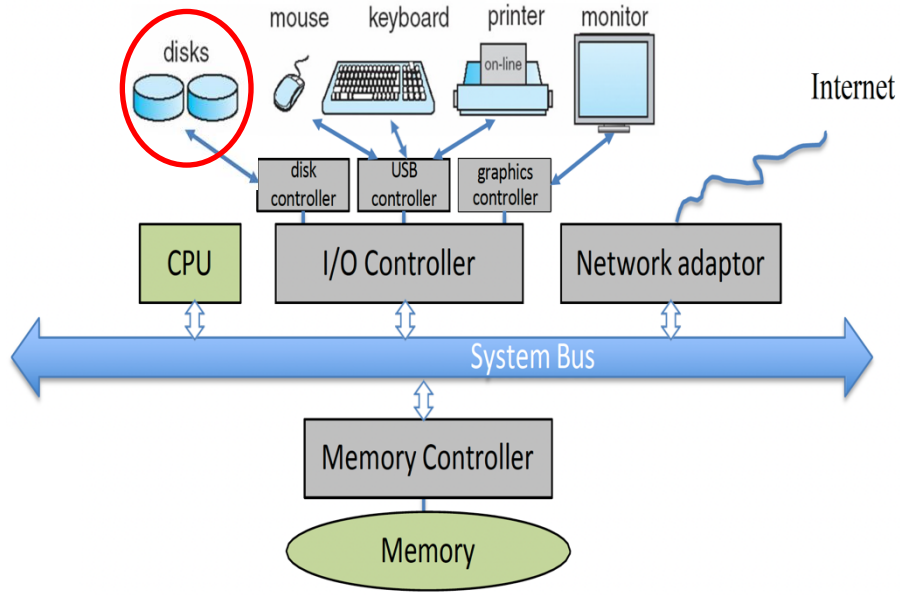
- The bus driver runs and queries the controller to enumerate attached devices:
  - Reads identifiers like vendor ID, device ID, class, etc.
- The kernel creates a "device object" that represents "disk at controller C, port P" and fills it with the disk's IDs and capabilities.



Case Study I: New Device Plugged In

Step 3: OS finds the device driver

- The OS now asks: "Which driver should handle this kind of disk device?"
- It matches the disk's IDs and protocol to a suitable disk driver.
- If that driver is not yet in memory, the OS:
  - Reads the driver's binary (kernel module) from storage into RAM,
  - Links and relocates it into the kernel's address space so its functions (e.g. disk interrupt handler) become executable kernel code.

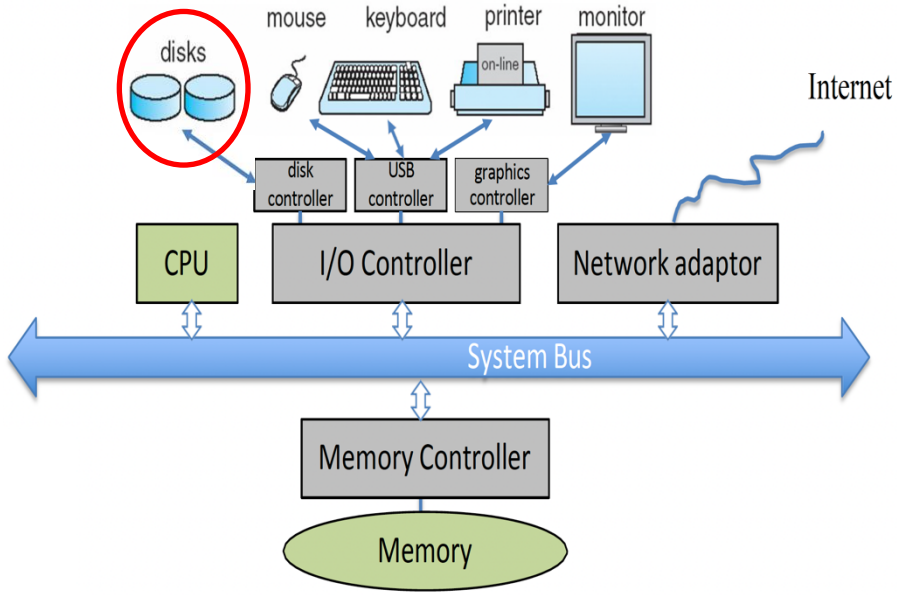


Case Study II: Disk Access and Interrupt

Case Study II: Disk Access and Interrupt

Step 1: OS issues disk I/O and disk uses DMA

- User process needs some data that is not in memory.
  - Page-fault!
- OS asks disk to fetch the data.
- Disk uses DMA to load data to the memory.

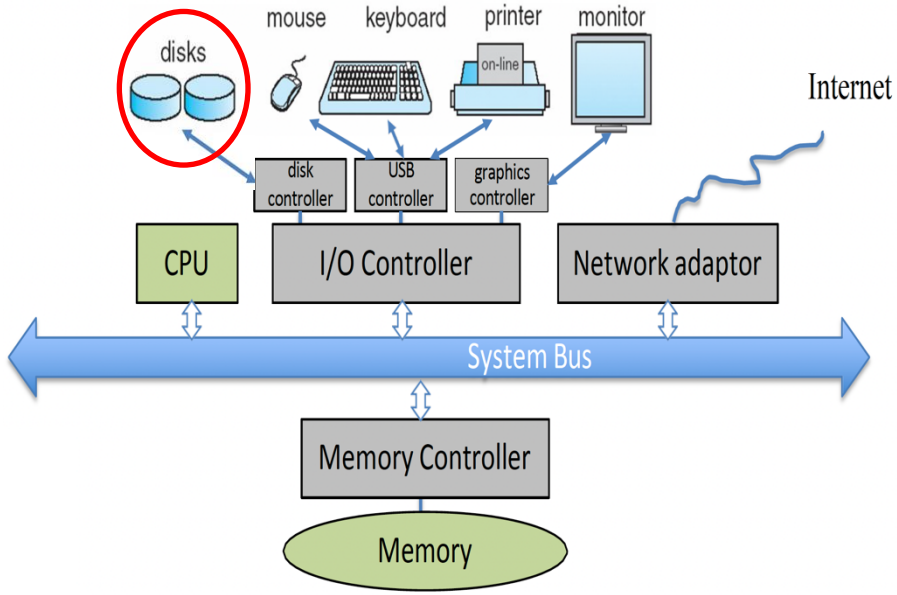


Case Study II: Disk Access and Interrupt

Step 2: Disk/controller signals completion

- When the disk I/O completes, the disk controller asserts its **IRQ** to signal the CPU that "DMA done."

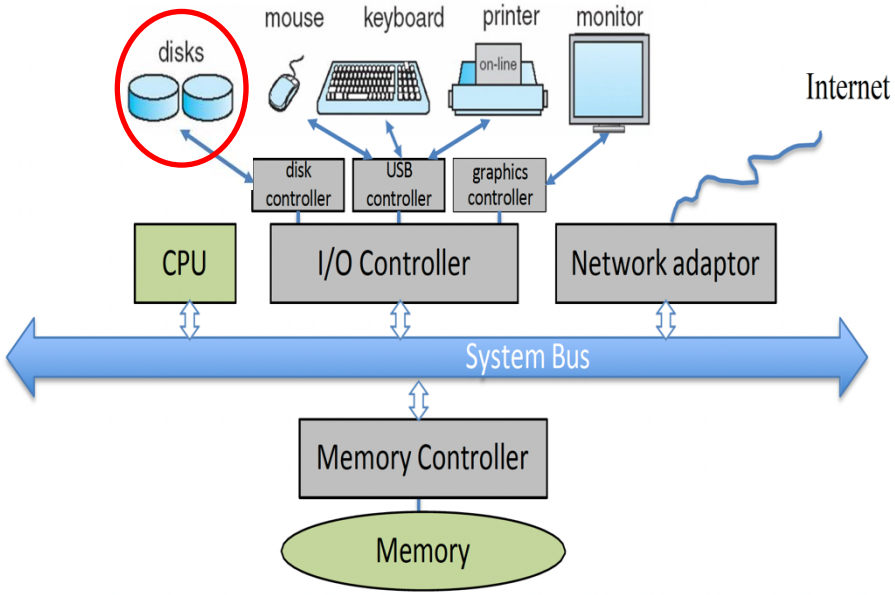
- A hardware request for the CPU to be interrupted.



Case Study II: Disk Access and Interrupt

Step 3: H/w interrupt controller (traffic cop)

- The interrupt controller sees that the disk controller's IRQ N is active:
  - Marks "IRQ N is pending."
  - When the CPU acknowledges an interrupt, the controller provides the interrupt vector number associated with that IRQ.
    - e.g., vector 0x2E for IRQ N.



Case Study II: Disk Access and Interrupt

Step 4: CPU switches to interrupt handling

- Meanwhile, the CPU is executing some user code (possibly unrelated to the disk I/O).
- After completing an instruction, its hardware checks: "Is there a pending interrupt?"
- If yes, the CPU saves the current state and switches to kernel mode, and fetches interrupt vector number from the interrupt controller.
- Uses that vector as an index into IDT, which gives the address of the interrupt entry code and begins executing.