Operating System 2- CS402

Lecture 3
I/O Systems

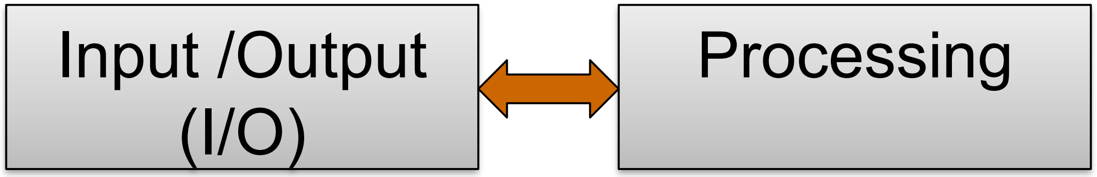
2025

Learning Objectives

- Explore the structure of an operating system’s **I/O subsystem**.
- Discuss the communication between **I/O Devices and the CPU**.
- Explain **Polling and Interrupts** in I/O Communication.

Jobs of a computer

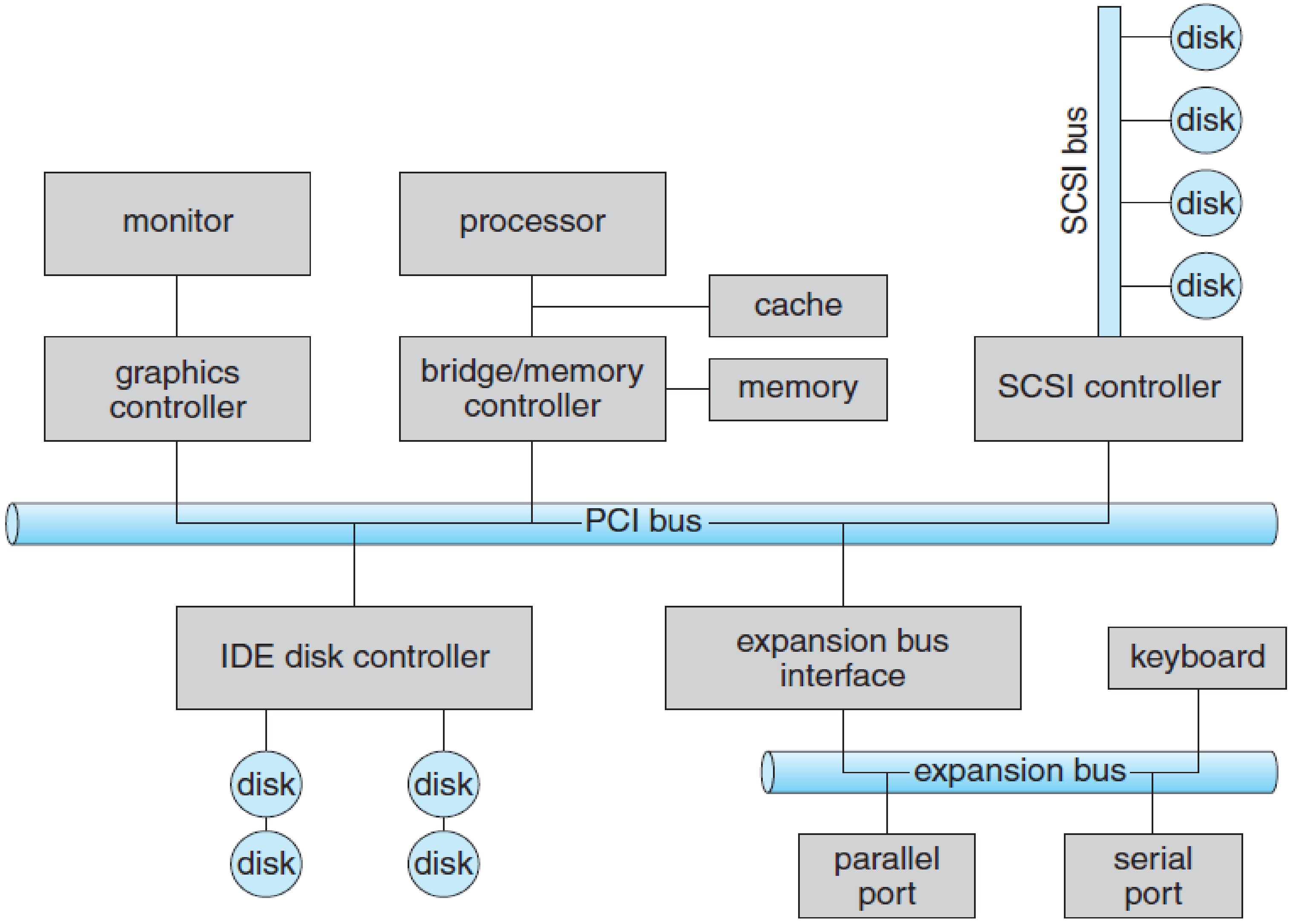
- The role of the **operating system** in computer I/O is to **manage** and **control** I/O operations and I/O devices.
- The control of devices connected to the computer is a major concern of operating-system designers.
- Because I/O devices **vary so widely in their function and speed**, varied methods are needed to control them.
- These methods form the **I/O subsystem** of the kernel, which separates the rest of the **kernel** from the complexities of managing I/O devices.



I/O Elements

- **Ports:** connection points for devices.
- **Buses:** communication pathways that transfer data between **different computer components**, such as the CPU, memory, and peripherals.
- **Device controllers:** hardware components that **manage** specific I/O devices and **translate** commands from the CPU.
- **Device drivers:** present a **uniform device access** interface to the I/O subsystem.

PC Bus Structure



PC Bus Structure (Cont.)

- **PCIe (Peripheral Component Interconnect Express)** is a type of connection used for **high-speed data transfer** between electronic components.
- **Bus - daisy chain** (shared) or direct access.
  - **PCI bus** common in PCs and servers, PCI Express (**PCIe**).
  - **Expansion bus** connects relatively slow devices.

Controller

- A **controller** is a **hardware** component that manages the operation of a specific I/O device.
- It acts as an intermediary between the CPU and peripheral devices, **handling data transfer** and communication.
- Each I/O device (such as a keyboard, hard drive, or network card) has its own controller, which **interprets CPU commands** and converts them into device-specific actions.

Processor communication with I/O Controllers

- **How Does the Processor Communicate with I/O Controllers?**
  - The processor must **send commands and data** to a controller to perform I/O transfers.
  - Controllers **manage device-specific** operations and communicate with the CPU.
  - This communication occurs **through registers** that store data and control signals.

Controller Registers for I/O Transfer

- Controllers have one or more registers:
  - **Data Register**: Holds data being transferred.
  - **Control Register**: Stores commands from the CPU.
  - **Status Register**: Reports the state of the device.
- The CPU interacts with these registers using specific methods.

Communication Between I/O Devices and the CPU

- **There are three main methods:**
  - I/O Instructions.
  - Memory-Mapped I/O Communication.
  - Hybrid Approach (I/O Instructions and Memory-Mapped I/O).
- **Some systems use both techniques:**
    - PCs use I/O instructions for some devices.
    - Memory-mapped I/O for others.

    This hybrid approach optimizes performance and compatibility.
- Understanding these techniques helps in designing efficient I/O systems.

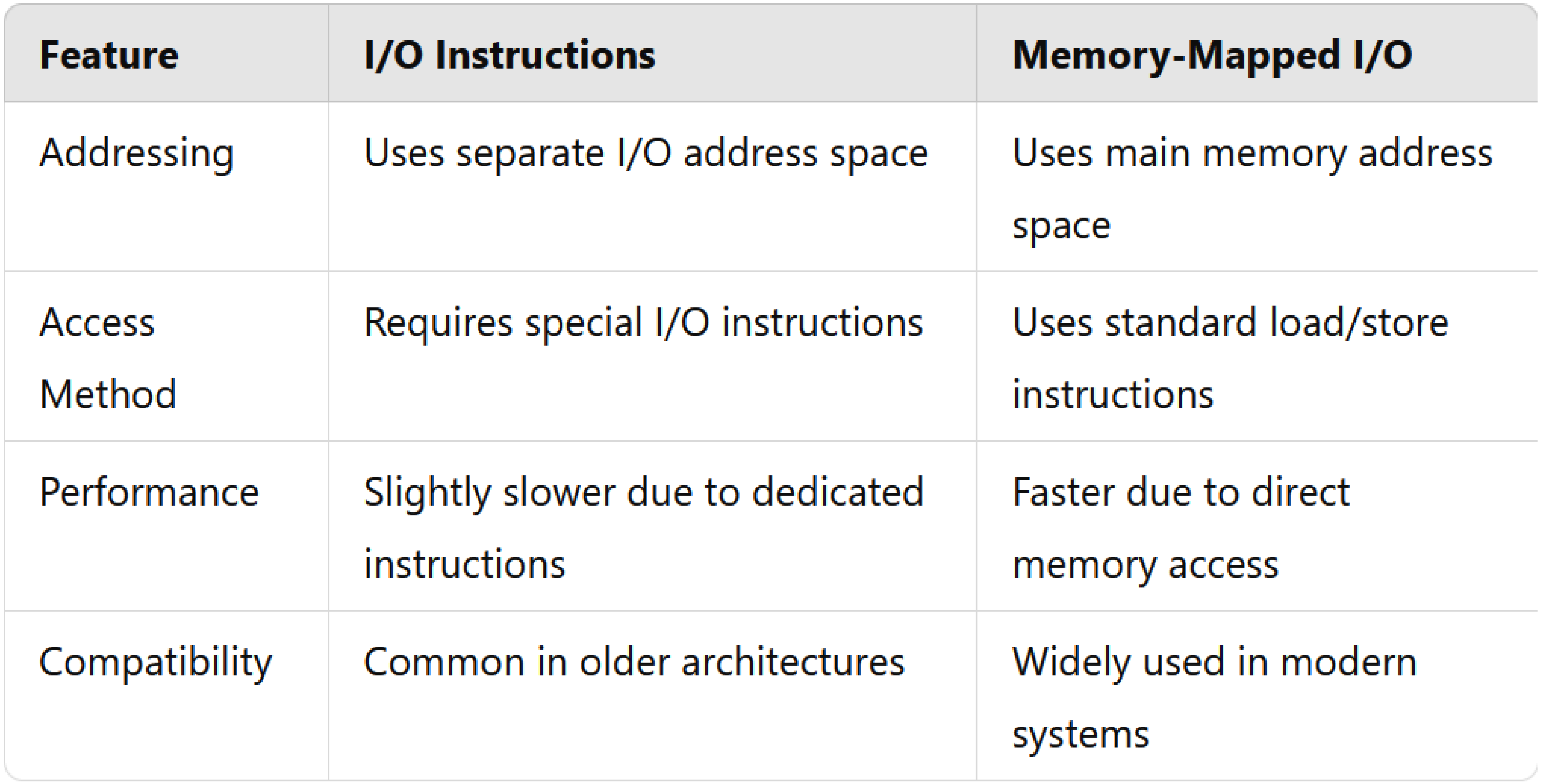
I/O Instructions Communication

- Special I/O **instructions** transfer data between CPU and I/O port.
- Key functions:
  - **Specify** the I/O port address.
  - **Trigger** bus lines to select the correct device.
  - **Move bits** in or out of the device register.
- Used in systems where dedicated I/O space is preferred.

Memory-Mapped I/O Communication

- Device-control registers are **mapped** into the CPU's address space.
- The CPU reads/writes to these registers using **standard data-transfer** instructions.
- No need for special I/O instructions.
- Provides a **unified addressing scheme** but may consume memory space.
- **Advantages**: Faster access, simplified instruction set, and direct integration with main memory operations.

Comparison: I/O Instructions vs. Memory-Mapped I/O



Feature	I/O Instructions	Memory-Mapped I/O
Addressing	Uses separate I/O address space	Uses main memory address space
Access Method	Requires special I/O instructions	Uses standard load/store instructions
Performance	Slightly slower due to dedicated instructions	Faster due to direct memory access
Compatibility	Common in older architectures	Widely used in modern systems

Discussion Questions

- Why might a system prefer memory-mapped I/O over I/O instructions?
- How does the hybrid approach benefit modern computers?
- Can you think of a device that would work better with I/O instructions rather than memory-mapped I/O?

Hybrid Approach in Modern Systems

- **Optimized Performance:** High-speed peripherals use memory-mapped I/O, while I/O instructions handle legacy devices.
- **Efficient Resource Management:** Balances **memory** use while maintaining dedicated I/O spaces for simpler devices.
- **Backward Compatibility:** Supports both old and new hardware efficiently.
- **Reduced Instruction Overhead:** Standard load/store operations work with memory-mapped I/O, simplifying processor design.
- **Real-World Example:**
  - PCs use **I/O instructions** for legacy peripherals like keyboards.
  - **Memory-Mapped I/O** is used for modern GPUs and SSDs.

Polling in I/O Communication

- **What is Polling?**
  - CPU continuously checks a device's status register for readiness.
- **Process:**
  - OS **sends** a request to a device.
  - CPU repeatedly **checks** the device controller's status register.
  - When the device is ready, data is transferred.
  - CPU processes the retrieved data.

Example: Polling Using Controller Registers

- **Scenario:** CPU reads data from a disk controller.
- **Polling Process:**
  - OS **sends** read request to disk controller.
  - CPU continuously **checks** disk controller’s status register.
  - When the status register indicates "ready," CPU reads data from the data register.

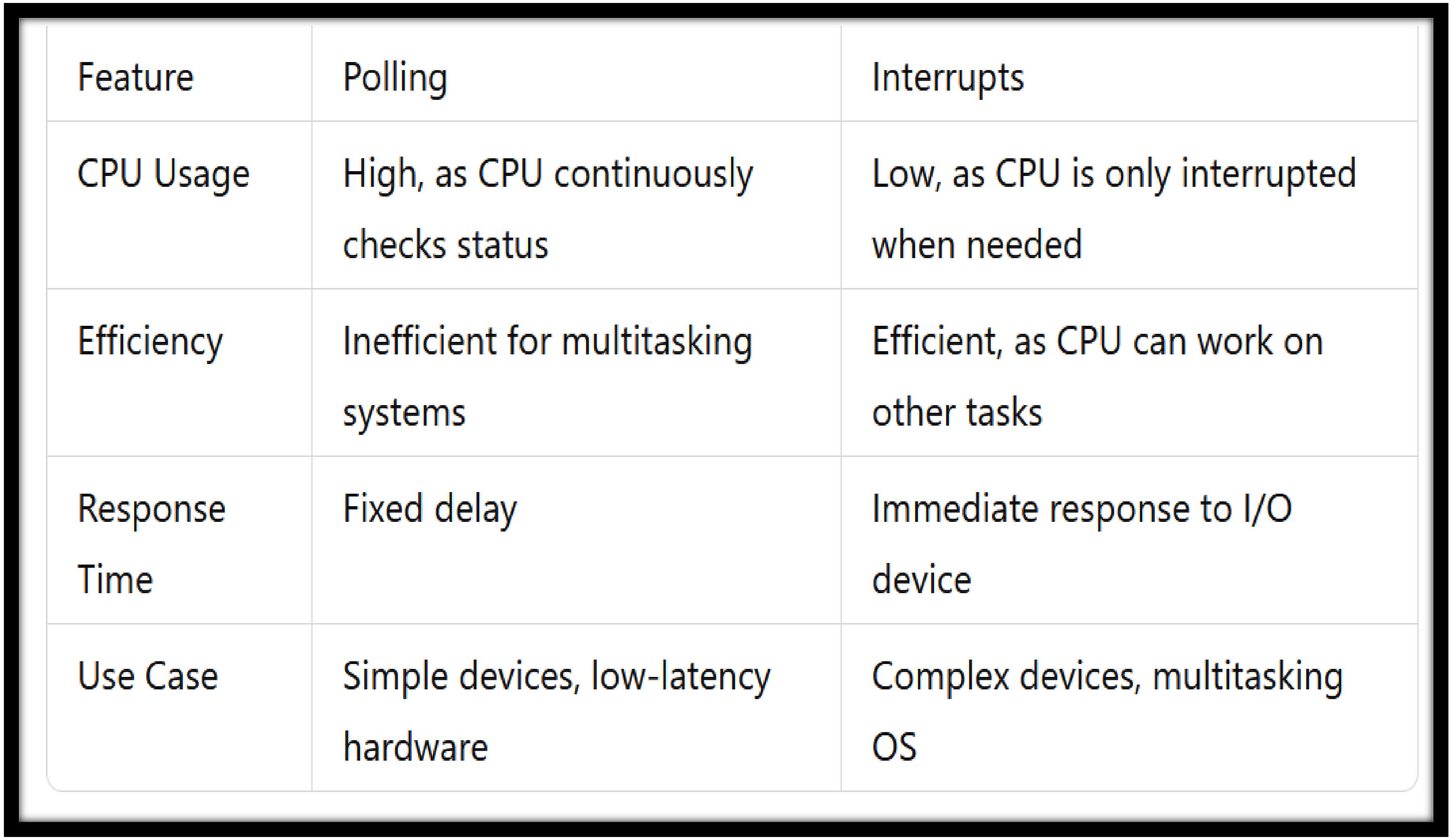
Polling: Advantages & Disadvantages

- **Advantages**:
  - Simple to implement.
  - Suitable for low-latency devices.
- **Disadvantages**:
  - **Wastes** CPU time (inefficient in multitasking systems).
  - CPU is **blocked** while waiting for device.
  - Not **scalable** for multiple devices.

Interrupts in I/O Communication

- **What are Interrupts?**
  - Device **notifies** CPU when it’s ready, instead of continuous polling.
- **Process:**
  - CPU **initiates** I/O request and moves on to other tasks.
  - When the device is **ready**, it sends an **interrupt** signal to the CPU.
  - The CPU **pauses** its current task, **processes** the interrupt, and **resumes** execution.

Polling vs. Interrupts



Feature	Polling	Interrupts
CPU Usage	High, as CPU continuously checks status	Low, as CPU is only interrupted when needed
Efficiency	Inefficient for multitasking systems	Efficient, as CPU can work on other tasks
Response Time	Fixed delay	Immediate response to I/O device
Use Case	Simple devices, low-latency hardware	Complex devices, multitasking OS

Real-World Examples of Polling & Interrupts

- **Polling** is ideal for **events that occur at consistent and predictable intervals**, allowing the system to check for updates periodically.
  - **For example**, a smart home device such as a thermostat may use polling to periodically check the sensor data to monitor changes in the room temperature and adjust the temperature based on user settings.
- **Interrupts** are more ideal when **immediate responses to external events** are critical, such as real-time systems. They are also best for event-driven applications where variable event timing is common.

Categorization of Interrupts Based on Priority

- **Maskable Interrupts:** Can be disabled or ignored by the CPU (e.g., user input, disk I/O completion).

- **Non-Maskable Interrupts (NMI):** Cannot be ignored by the CPU; used for critical events like hardware failures or system crashes.

Direct Memory Access (DMA)

- **What is DMA?**
  - Allows data transfers between **devices and memory** without CPU involvement.
  - **DMA controller** handles data movement.
- **Steps of DMA Transfer:**
  - CPU issues a **request** to the DMA controller.
  - DMA controller transfers data directly from device to RAM.
  - DMA controller signals completion with an interrupt.

Real-World Example

- **Hard Disk Data Transfer**: When a hard drive reads data, the OS can use DMA to copy large chunks directly into memory.
- **Network Data Processing**: High-speed network cards use DMA to transfer packets without burdening the CPU.
- **Graphics Processing**: GPUs use DMA to quickly transfer textures and graphics data from RAM to VRAM.

Discussion Questions

- Why might a system prefer interrupts over polling?
- Can you think of a scenario where polling is still beneficial?
- How does an OS manage multiple I/O interrupts efficiently?

The End