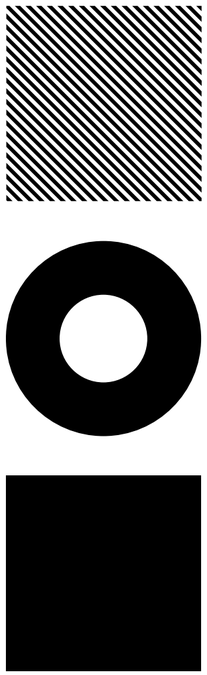
I/O Architecture

Agenda

1. Overview
2. Device Model
3. Device Drivers
4. Char/Block Device Interface

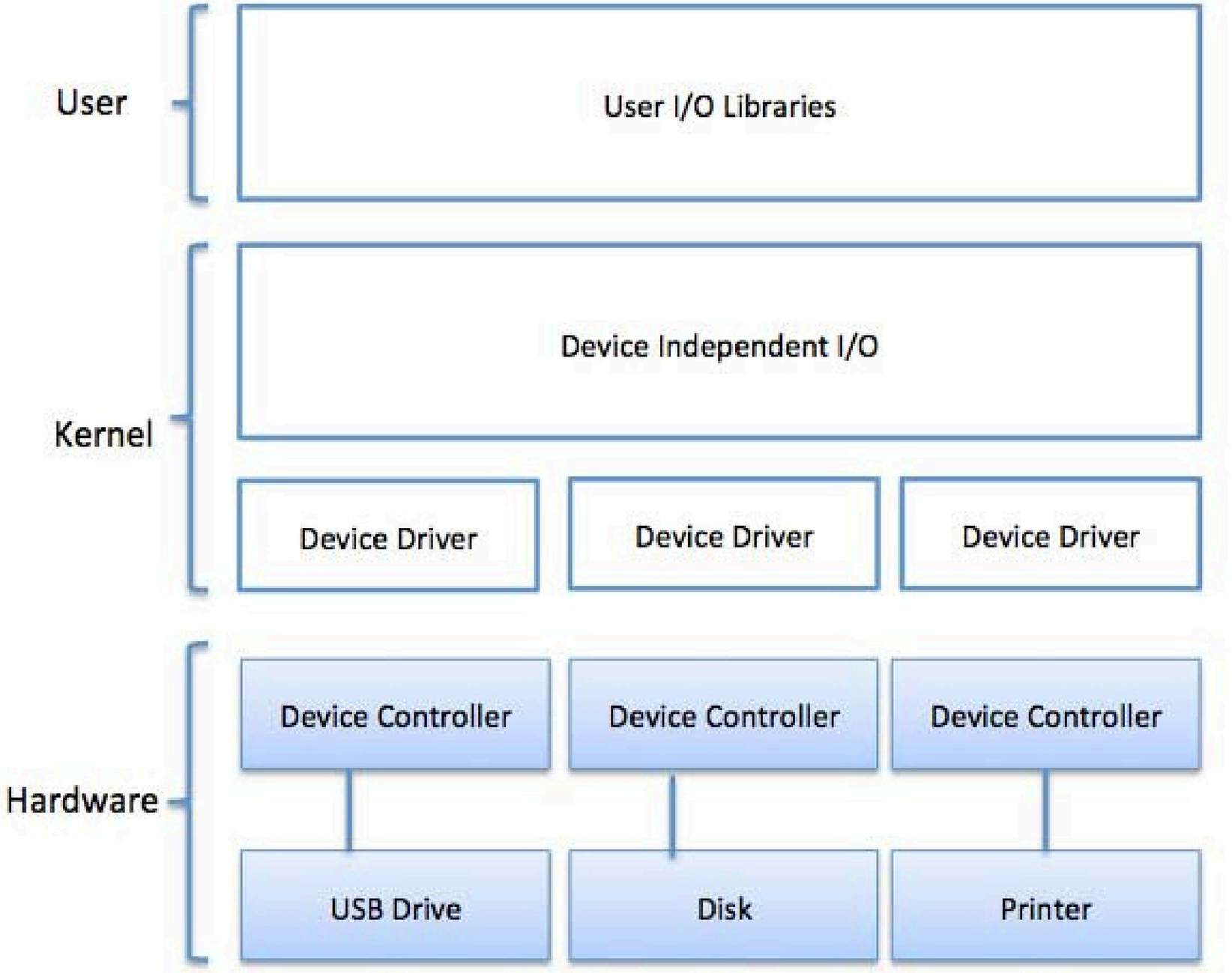


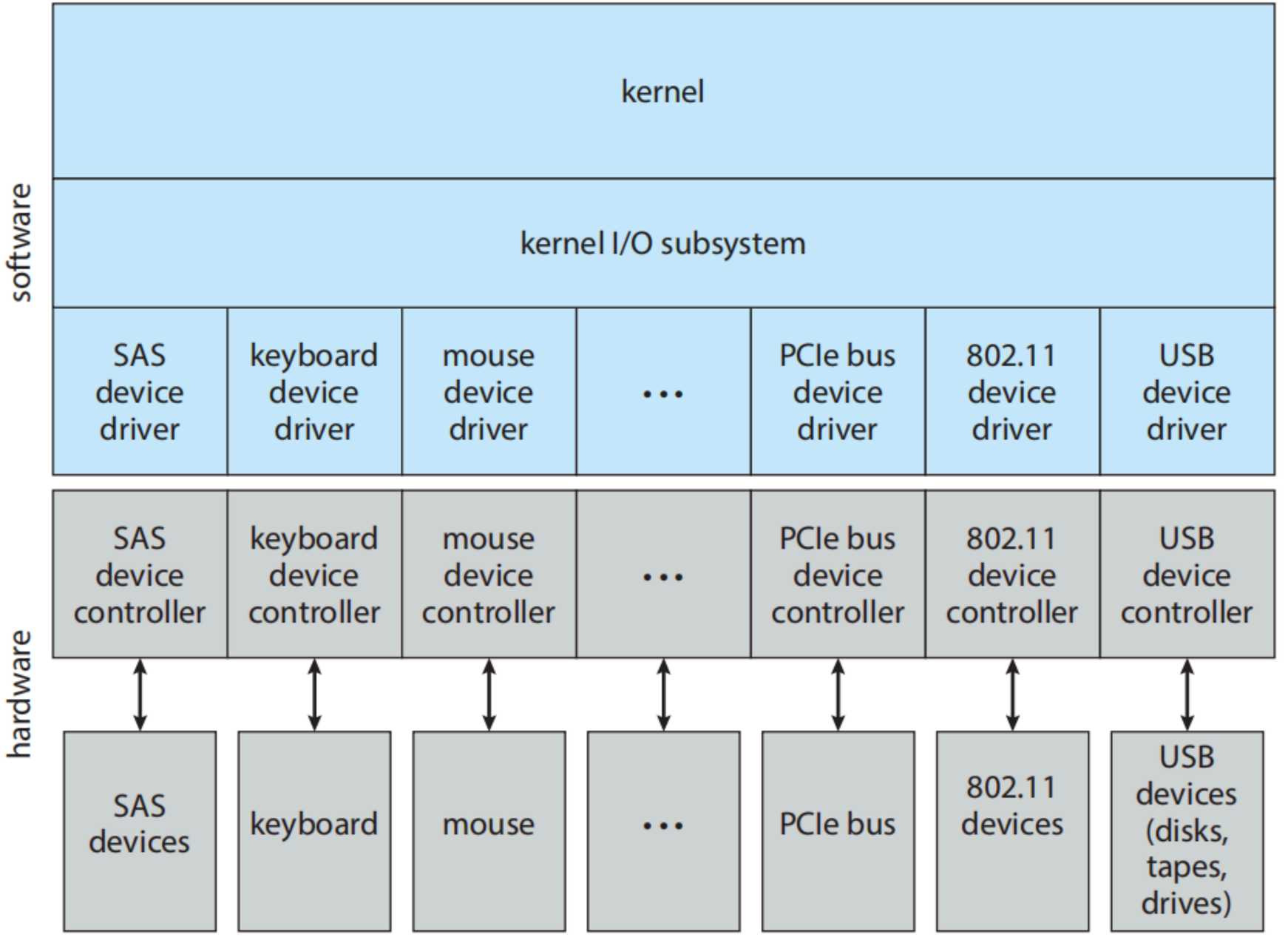
Overview

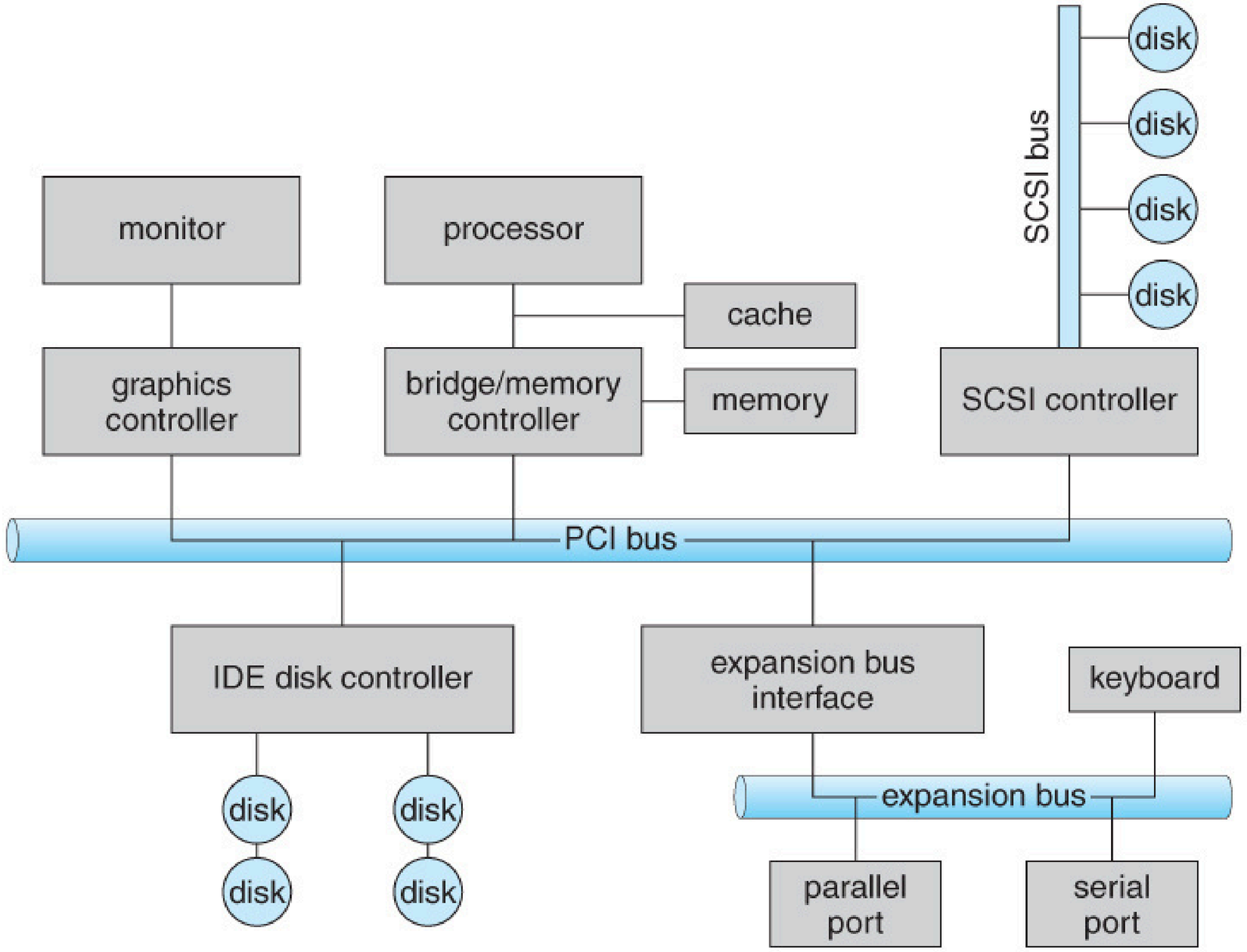
Definition, Diagram, Components explanantion

Definition:

The I/O subsystem in a Linux operating system is a component that manages the input and output operations of the system. It facilitates communication between the hardware devices and the software applications, allowing data to be transferred between the two. The I/O subsystem includes various device drivers, interfaces, and protocols that enable the operating system to interact with different types of hardware, such as storage devices, network interfaces, and display adapters. By efficiently managing I/O operations, the Linux I/O subsystem helps ensure optimal system performance and stability.


User
User I/O Libraries
Kernel
Device Independent I/O
Device Driver
Device Driver
Device Driver
Hardware
Device Controller
Device Controller
Device Controller
USB Drive
Disk
Printer


kernel
kernel I/O subsystem
SAS device driver
keyboard device driver
mouse device driver
• • •
PCIe bus device driver
802.11 device driver
USB device driver
software
SAS device controller
keyboard device controller
mouse device controller
• • •
PCIe bus device controller
802.11 device controller
USB device controller
hardware
SAS devices
keyboard
mouse
• • •
PCIe bus
802.11 devices
USB devices (disks, tapes, drives)


monitor
processor
cache
graphics controller
bridge/memory controller
memory
SCSI bus
disk
disk
disk
disk
SCSI controller
PCI bus
IDE disk controller
expansion bus interface
keyboard
disk
disk
disk
disk
expansion bus
parallel port
serial port

Key Componenets :

- I/O Bus
- Controllers
- Ports
- I/O Modes (PIO, Interrupt, DMA)

I/O Bus:

Definition : It’s daisy chain or shared direct access.

Commonly Types:

- The **PCI bus** connects high-speed high-bandwidth devices to the memory subsystem (and the CPU.)
- The **expansion bus** connects slower low-bandwidth devices, which typically deliver data one character at a time (with buffering.)
- The **SCSI bus** connects a number of SCSI devices to a common SCSI controller.

Controllers

Definition

A controller is a hardware or software component that manages the communication and operation between a computer and peripheral devices

Some of Types

- **Device Controllers:** Hard Disk Controller (HDC), Display Controller.
- **Network Controllers:** Network Interface Controller (NIC).
- **I/O Controllers:** USB Controller.
- **Graphics Card Controllers**
- **DMA (Direct Memory Access) Controllers.**
- **IDE (Integrated Drive Electronics) Controllers.**

Functions of a Controller:

- Ensures the device operates as expected and efficiently.
- Converts system-level commands into device-specific instructions.
- Detects and handles errors in data transmission or device operation.
- Temporarily stores data to manage different speeds between devices and the CPU.
- Translates data formats or protocols for compatibility.

Ports

Definition

ports are logical or physical communication endpoints used for transferring data between the CPU and peripheral devices

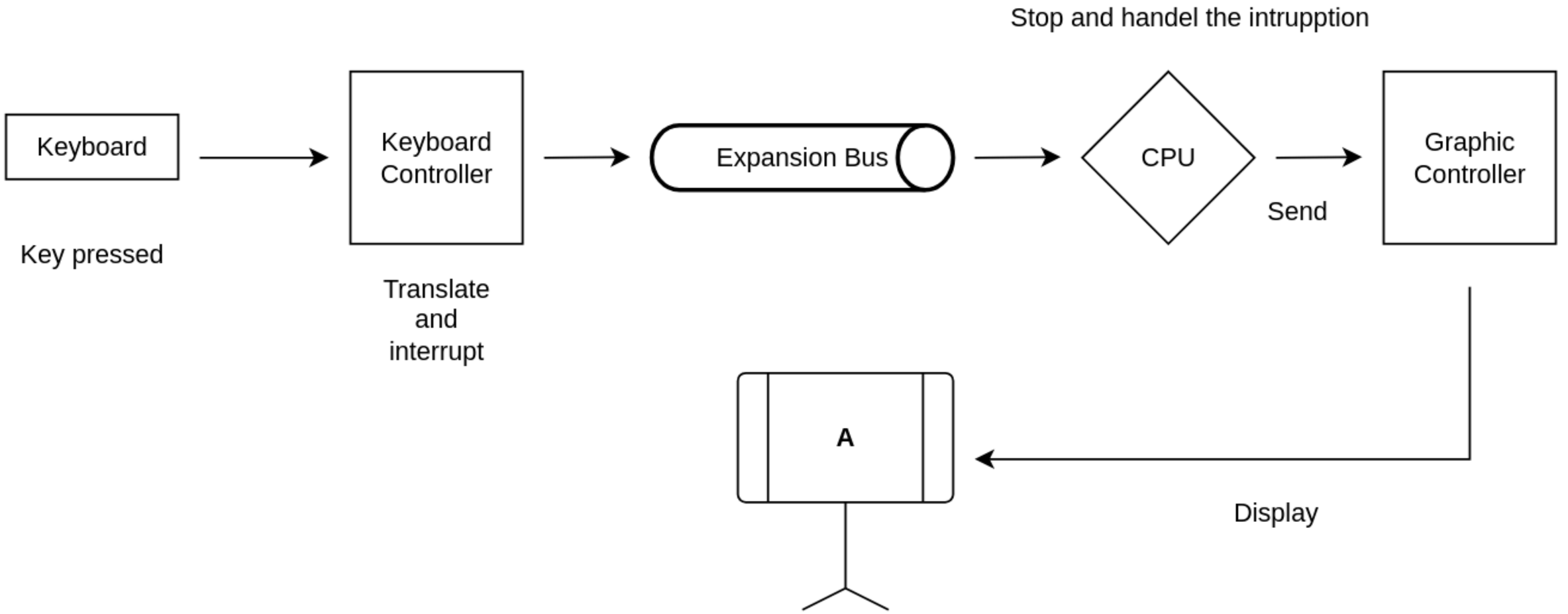
- Each port is identified by a unique address .
- Port addresses are stored in a dedicated **I/O address space**, separate from the system's main memory address space.
- I/O devices can be **mapped** into the system’s memory address space, allowing CPU instructions that access memory to also access I/O devices (memory-mapped I/O).

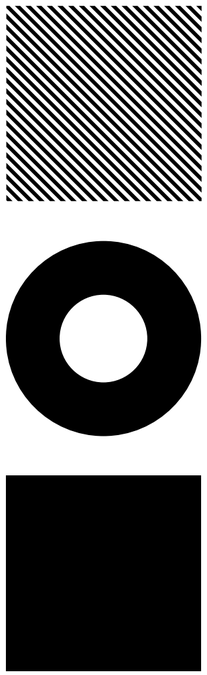
I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

I/O Modes

Input/Output (I/O) modes define how data is transferred between a computer's CPU and peripheral devices. Each mode varies in complexity, speed, and CPU involvement.

- **Programmed I/O (PIO)** the CPU is fully responsible for controlling data transfer between memory and the I/O device.The CPU executes a program that directly reads from or writes to the device.
- **Interrupt-Driven I/O** In this mode, the device notifies the CPU through an **interrupt** when it is ready for communication.
- **Direct Memory Access (DMA)** a dedicated hardware component called the **DMA controller** takes over the data transfer process, bypassing the CPU for most operations.


Stop and handel the intrupption
Keyboard
Keyboard Controller
Expansion Bus
CPU
Graphic Controller
Key pressed
Translate and interrupt
Send
A
Display



Device Model

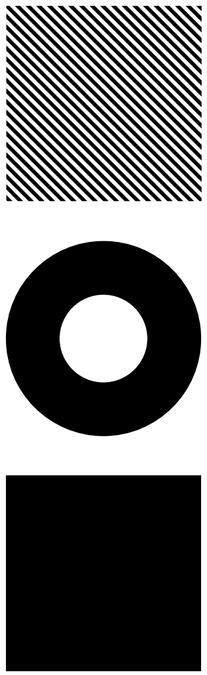
Definition, Components

Definitions

- Its a way the OS manages communication between the computer and hardware devices (like a keyboard, mouse, hard drive, or printer). It provides a structured approach to make sure the OS can handle various types of devices in a consistent way.
- **Devices are Different**, OS is **Consistent**: Every hardware device is different and has its own way of working. The OS uses the device model to standardize how it communicates with all devices, so programs don’t need to worry about hardware specifics.

Some of components

- **Device Drivers:** These are software pieces specific to each device, acting as translators between the OS and the device.
- **Device Abstraction:** The OS hides the complexity of hardware by providing a uniform interface. For example, reading a file from a USB drive feels the same as reading from a hard disk, even though the hardware is different.

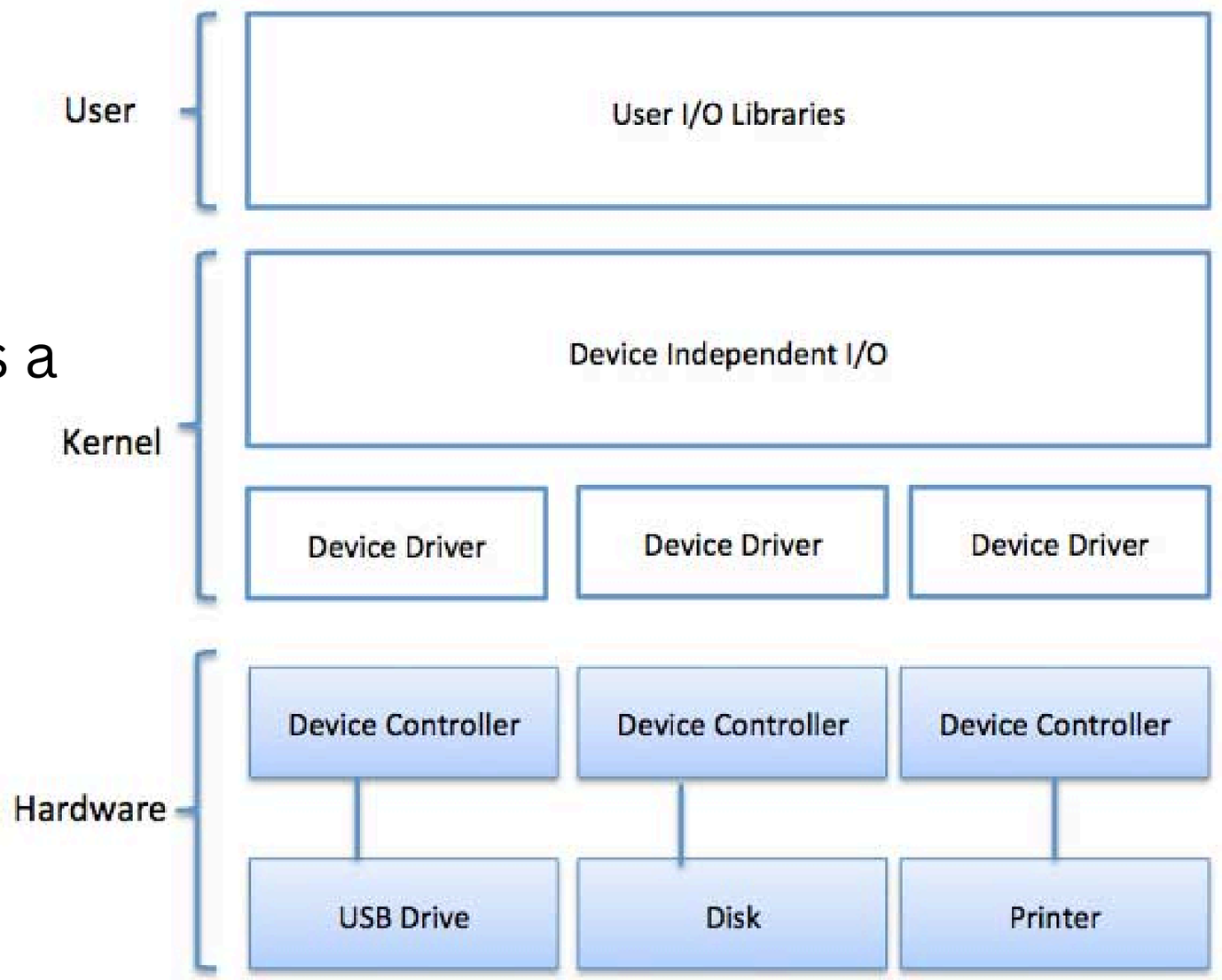


Device Drivers

your subtitles here

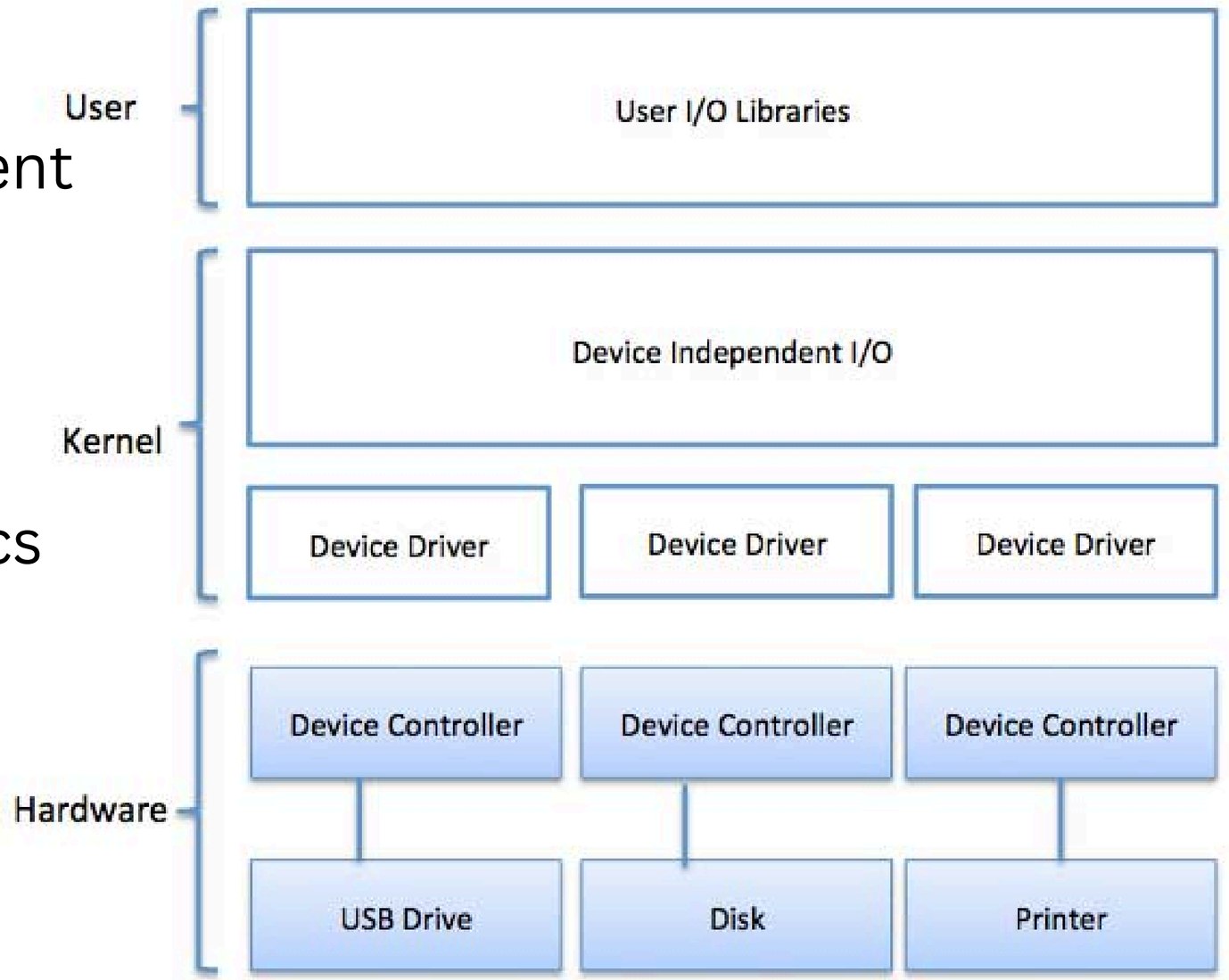
Definitions

- Device drivers are specialized software components that act as a bridge between the operating system and hardware devices.



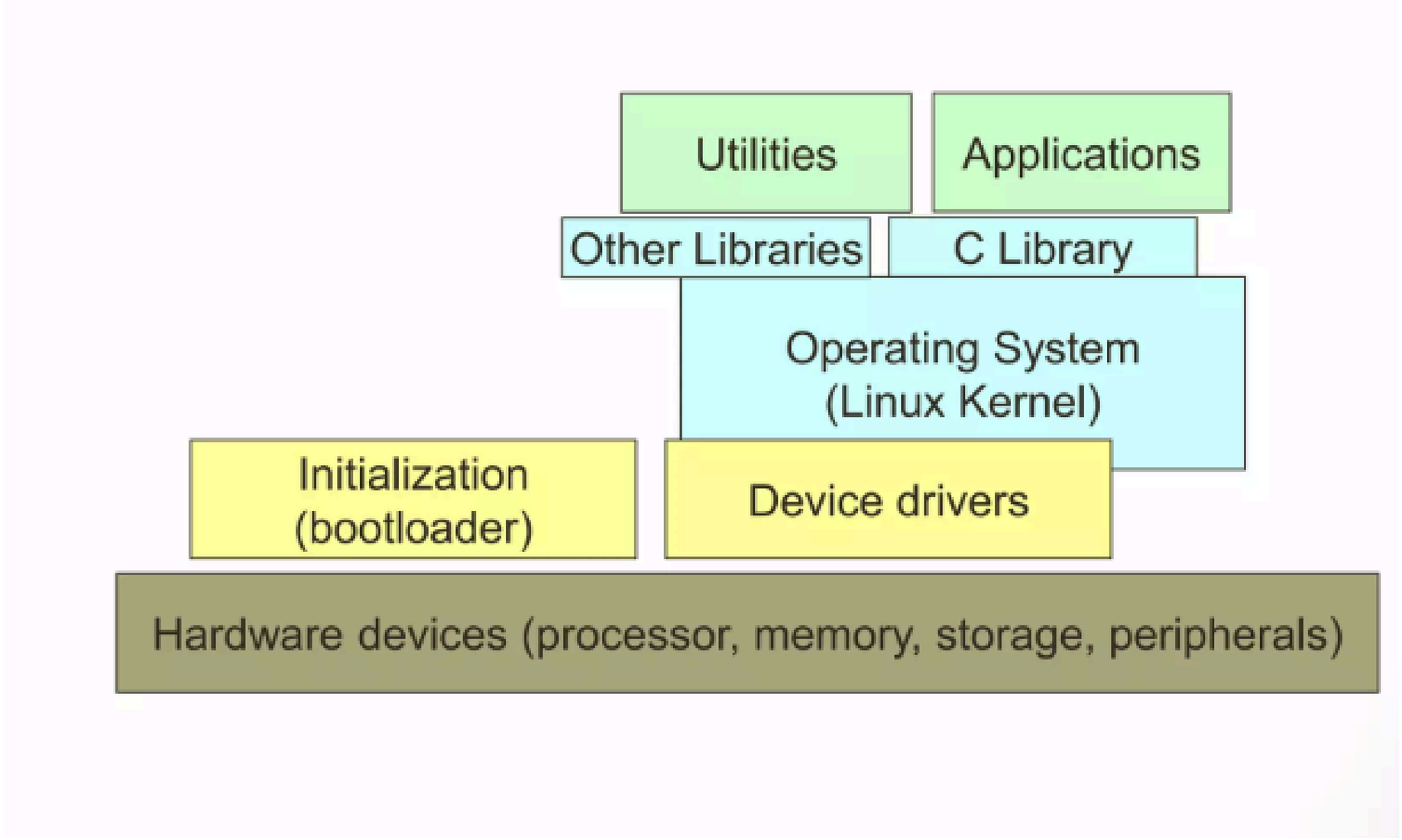
Definitions

- They provide a consistent interface for the OS to interact with hardware without needing to understand the specifics of the device.



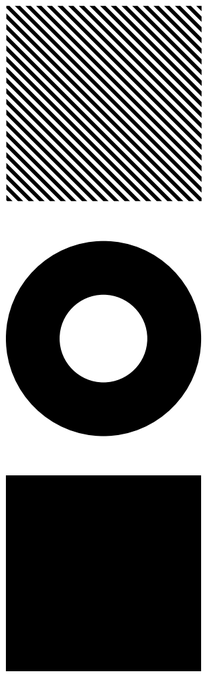
Roles and Responsibilities:

- Provide an abstraction layer, so the OS doesn’t need to understand hardware-specific details.
- Convert high-level OS commands (e.g., “write to disk”) into hardware-specific instructions.
- Manage interrupts and facilitate communication between devices and the OS kernel.


Utilities
Applications
Other Libraries
C Library
Operating System
(Linux Kernel)
Initialization
(bootloader)
Device drivers
Hardware devices (processor, memory, storage, peripherals)

Types of Drivers:

- **Character Device Drivers:** Handle data streams from character-based devices like keyboards or mice.
- **Block Device Drivers:** Manage data transfer for block-based storage devices like hard disks or SSDs.
- **Network Device Drivers:** Facilitate communication over networks via NICs (Network Interface Cards).



Char/Block Device Interface

Character Device, Block Device

What is a Character Device?

- A character device (or char device) is a hardware device that interacts with the operating system by transferring data as a stream of bytes, rather than in blocks. Examples include keyboards, mice, serial ports, and text consoles.

Character Device: Characteristics

1. **Stream-based Data Transfer:**
    - Data is sent or received sequentially as a stream of bytes.
    - Unlike block devices, it does not allow random access to specific portions of data.

Character Device: Characteristics

2.Driver Behavior:

- A char driver manages the communication between the device and the operating system.
- It implements basic system calls, such as:
  - open: To initialize and set up the device for communication.
  - close: To release the device once operations are complete.
  - read: To retrieve data from the device.
  - write: To send data to the device.

Character Device: Characteristics

3.Access via Filesystem Nodes:

- Char devices are represented in the file system, usually in the /dev directory.
- Example nodes:
  - /dev/tty1: Represents a terminal or console.
  - /dev/lp0: Represents a parallel printer port.

Character Device: Characteristics

4.Sequential Access:

- Unlike regular files, where you can move freely within the file (random access), most char devices allow only sequential access.
- For instance, data from a serial port is consumed in the order it arrives.

Examples of Character Devices

1. **Text Console (/dev/console):**
    - Used for displaying system messages or interacting with the system in text mode.
2. **Serial Ports (/dev/ttyS0, /dev/ttyS1, etc.):**
    - Often used for debugging or communicating with external devices like modems or microcontrollers.
3. **Keyboards and Mice:**
    - Handle real-time user inputs, represented as a stream of bytes corresponding to keystrokes or mouse movements.

What is a Block Device?

- A block device is a hardware device, such as a disk drive, that transfers data in fixed-size chunks called blocks. These blocks are typically 512 bytes or larger (in powers of two), depending on the device.

Block Device: Characteristics

1. Filesystem Hosting:
    - Block devices are primarily designed to host filesystems, enabling the storage and retrieval of structured data.
    - Examples include hard drives, solid-state drives (SSDs), and optical drives.

Block Device: Characteristics

2. Fixed-Size Data Transfers:
    - In most Unix systems, block devices handle I/O operations that involve reading or writing entire blocks of data at a time.
    - Block devices are ideal for applications where random access to large amounts of data is required, such as databases or file systems.

Block Device: Characteristics

3.Linux Flexibility:

- Unlike traditional Unix systems, Linux allows byte-level access to block devices, making them behave similarly to character devices in some scenarios. This allows the transfer of arbitrary byte sizes, not strictly multiples of the block size.

Block Device: Characteristics

4.Kernel Differences:

- While block and character devices can appear similar to users (both represented as filesystem nodes in /dev), the kernel handles them differently:
    - Block Devices: The kernel buffers and caches data in memory for optimized read/write performance. It manages the device as a random-access storage medium.
    - Character Devices: Data is not buffered; it is processed immediately as a stream.

Accessing Block Devices

1. Filesystem Nodes:
    - Block devices, like character devices, are represented by special files in the /dev directory.
    - Examples:
        - /dev/sda: A standard block device representing the first hard disk.
        - /dev/loop0: A loopback device used for mounting disk images.
2. Transparent to the User:
    - From the user’s perspective, accessing a block device is similar to accessing a character device. The distinction lies in how the kernel and drivers manage the data.

Examples of Block Devices

1. Hard Drives and SSDs (/dev/sda, /dev/nvme0n1):
    - Store operating systems, filesystems, and user data.
2. Optical Drives (/dev/sr0):
    - Devices for reading/writing CDs, DVDs, or Blu-rays.
3. Loopback Devices (/dev/loopX):
    - Virtual devices used to mount disk images.

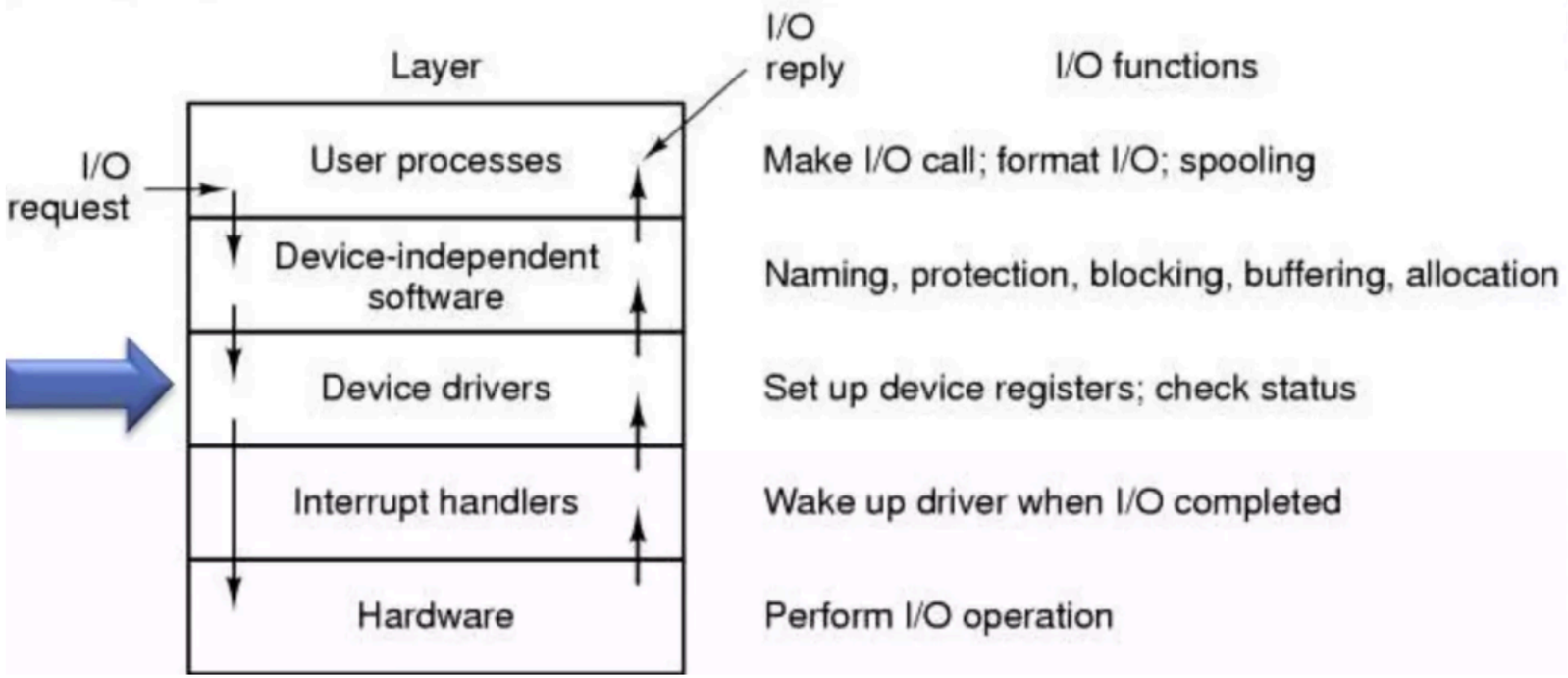
Comparison: Block Devices vs Character Devices

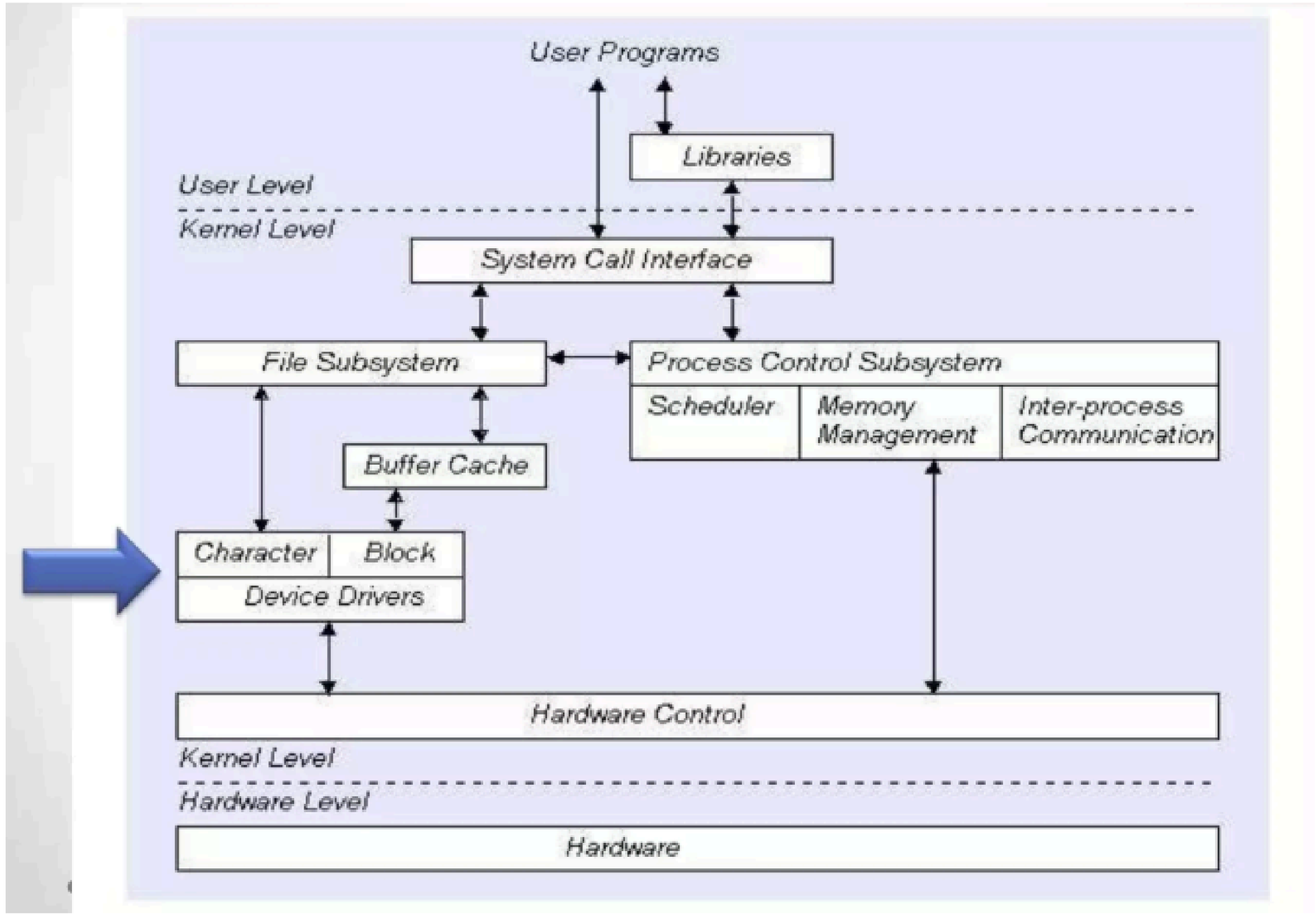
Feature	Block Devices	Character Devices
Data Transfer	Fixed-size blocks	Stream of bytes
Caching	Buffered for performance	Unbuffered, real-time
Access	Random-access	Sequential access
Use Case	Storage (e.g., disks, SSDs)	Input/output (e.g., keyboard)

References:

- Operating System Concepts, 10th Edition
- Understanding the Linux Kernel.(2005)
- Operating Systems: Three Easy Pieces
- Operating Systems: Internals and Design Principles - Focuses on architecture and driver management.

Thank you!


Layer
I/O reply
I/O functions
I/O request
User processes
Make I/O call; format I/O; spooling
Device-independent software
Naming, protection, blocking, buffering, allocation
Device drivers
Set up device registers; check status
Interrupt handlers
Wake up driver when I/O completed
Hardware
Perform I/O operation


User Programs
Libraries
User Level
Kernel Level
System Call Interface
File Subsystem
Process Control Subsystem
Scheduler
Memory Management
Inter-process Communication
Buffer Cache
Character
Block
Device Drivers
Hardware Control
Kernel Level
Hardware Level
Hardware