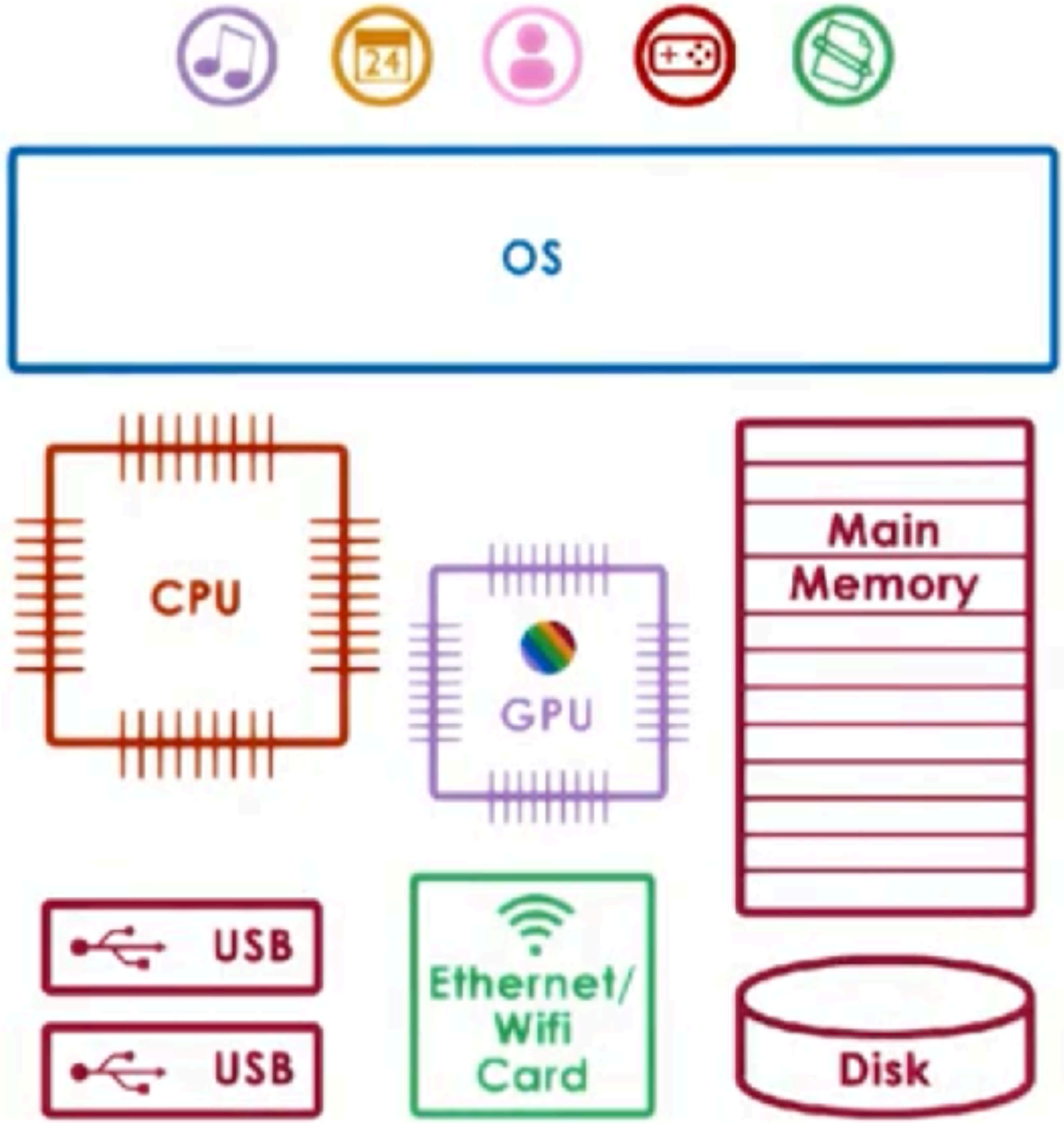
I/O Management

I/O Devices



Basic I/O Device Features

Control registers
- command
- data transfers
- status

Microcontroller == device's CPU

On device memory

Other logic
- e.g., analog to digital converters

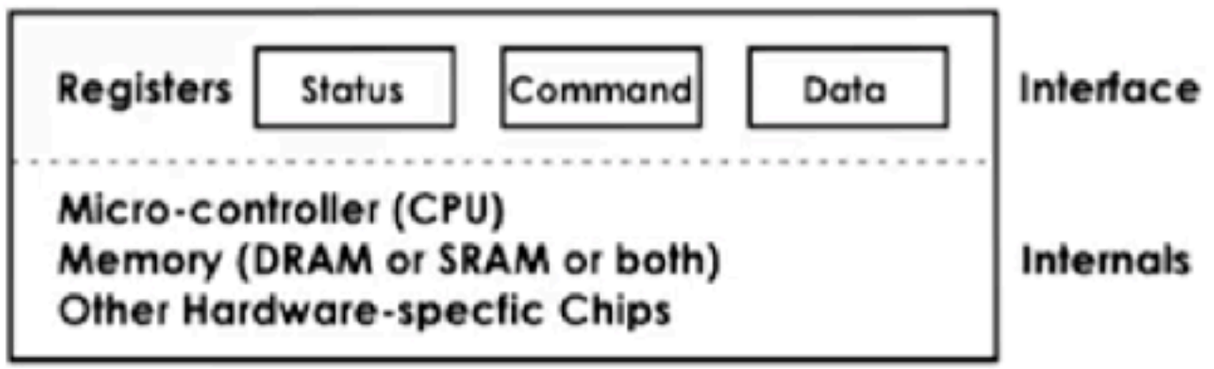
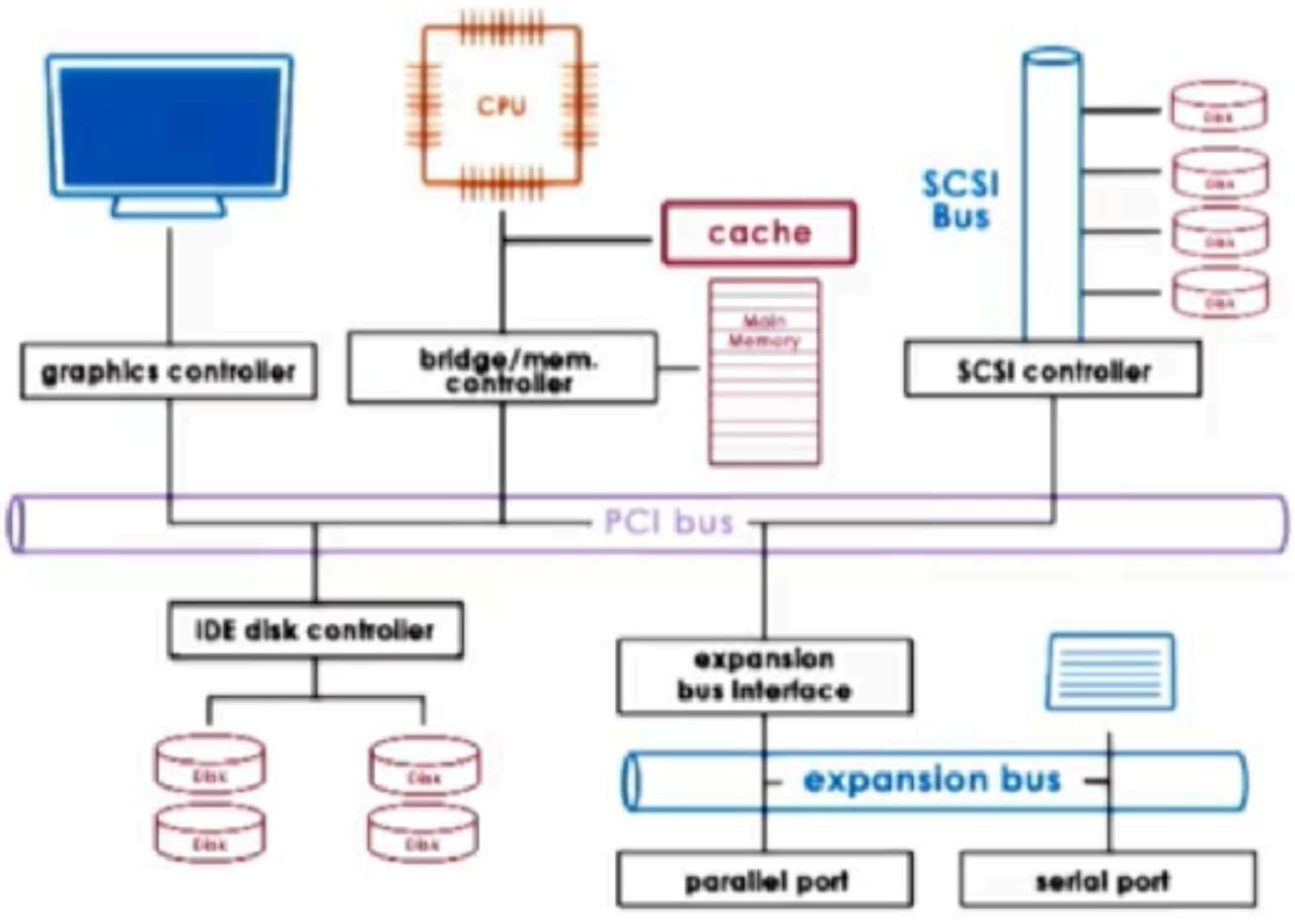


Figure 36.2: A Canonical Device

CPU - Device Interconnect



Peripheral Component Interconnect (PCI)

- PCI Express (PCIe)
  (> PCI-X > PCI)

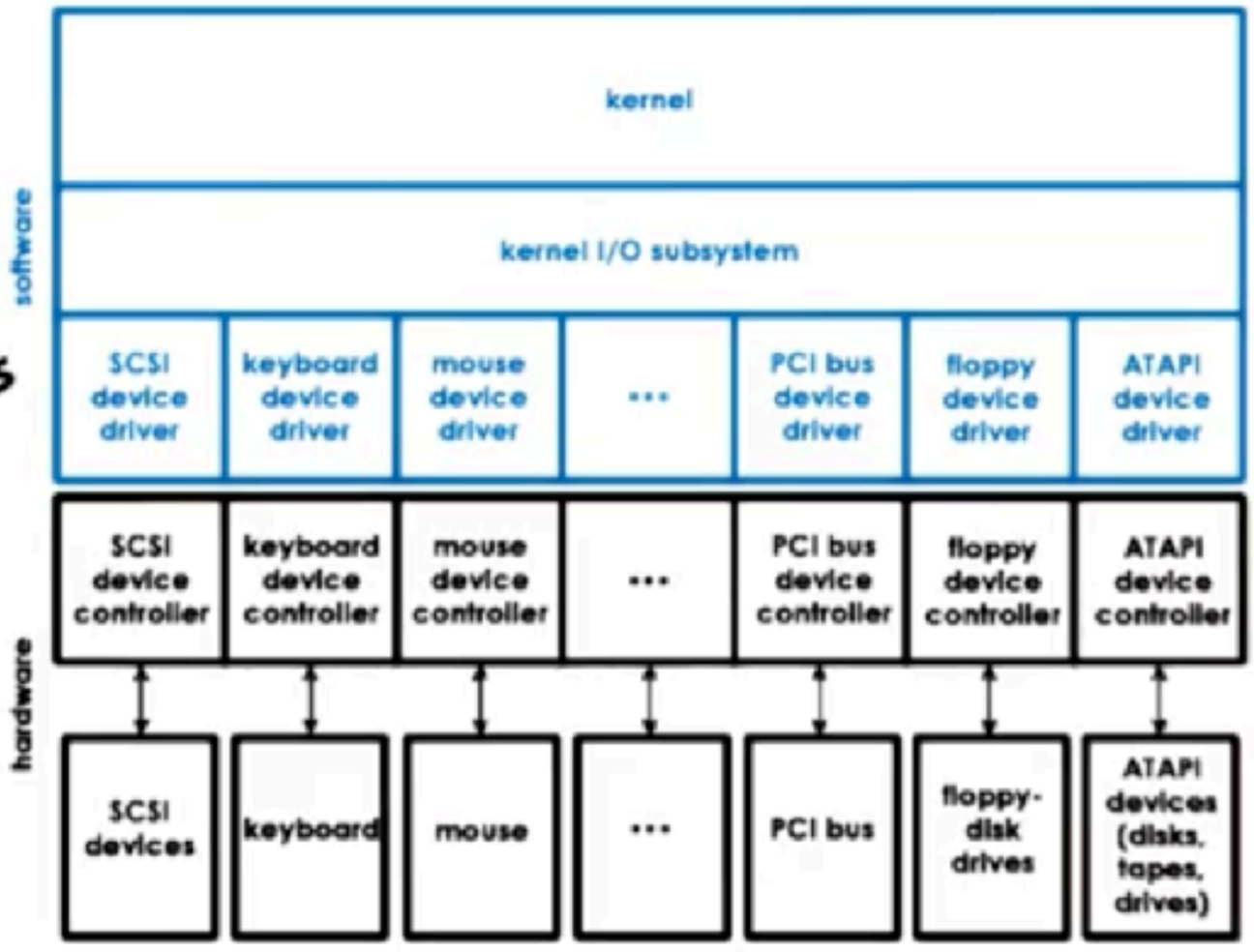
Other types of interconnects

- scsI bus
- peripheral bus
- bridges handle differences

Device Drivers

Device Drivers

- per each device type
- responsible for device access, management and control
- provided by device manufacturers per OS/version
- each OS standardizes interfaces
  - device independence
  - device diversity



Device Types

Block : disk
- read / write blocks of data
- direct access to arbitrary block

Character : keyboard
- get / put character

Network devices

OS representation of a device == special device file

UNIX-like systems

- /dev
- tmpfs
- devfs

Pseudo Devices Quiz

Linux supports a number of pseudo ("virtual") devices that provide special functionality to a system. Given the following functions name the pseudo device that provides that functionality.

- accept and discard all output (produces no output)

/dev/null

- produces a variable-length string of pseudo-random numbers

/dev/random

Note: Answer by using the full path (e.g., /dev/|p0)

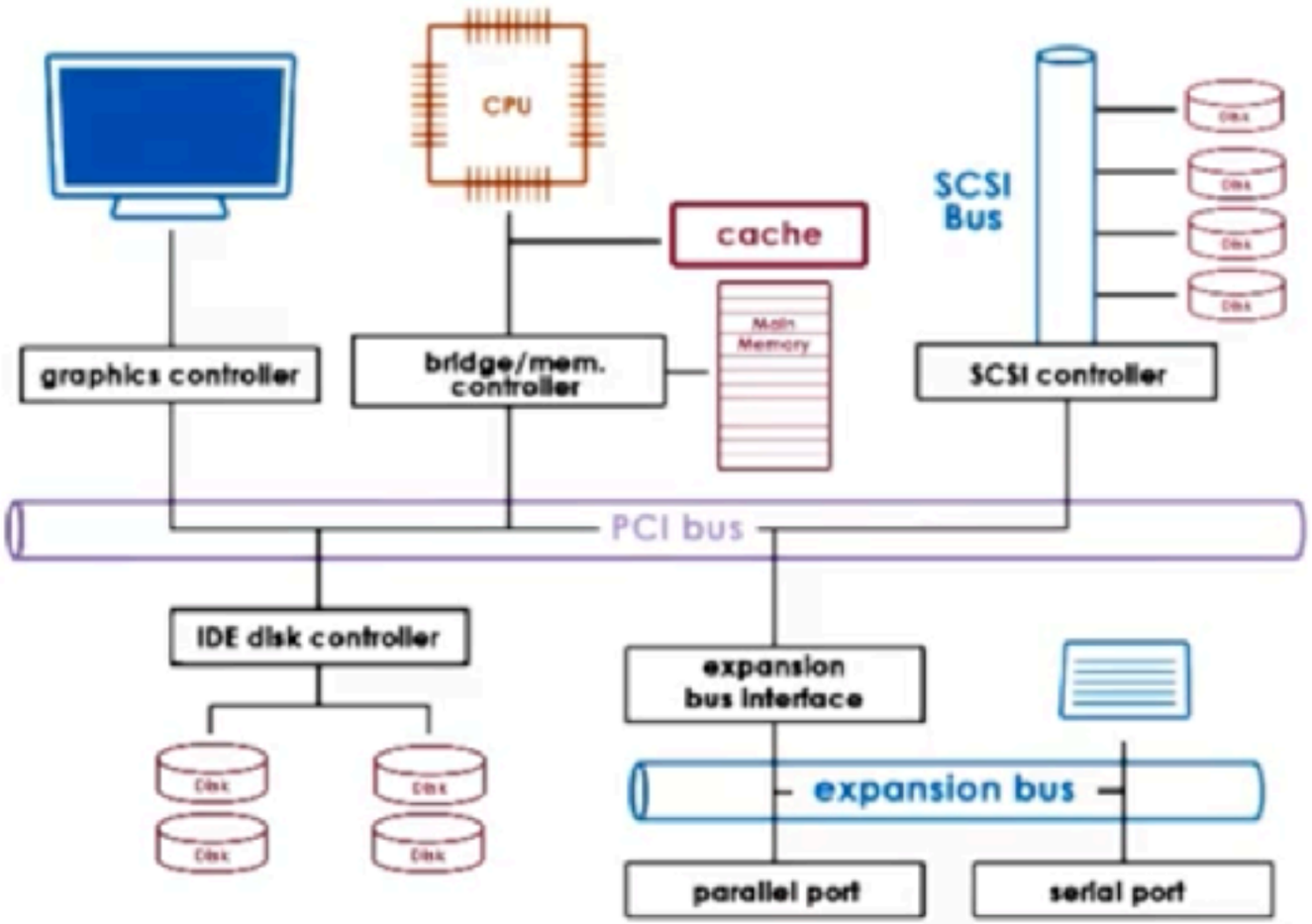
Looking at /Dev Quiz

Run the command 'ls -la /dev' in a Linux environment. What are some of the device names you see? Enter at least five device names in a comma-separated list in the text-box below.

hda, sda, tty, null, zero, ppp, lp, mem, console, autoconf, ...

Note: You may use the Ubuntu VM provided for this course. See the Instructor Notes for a download link.

CPU - Device Interactions

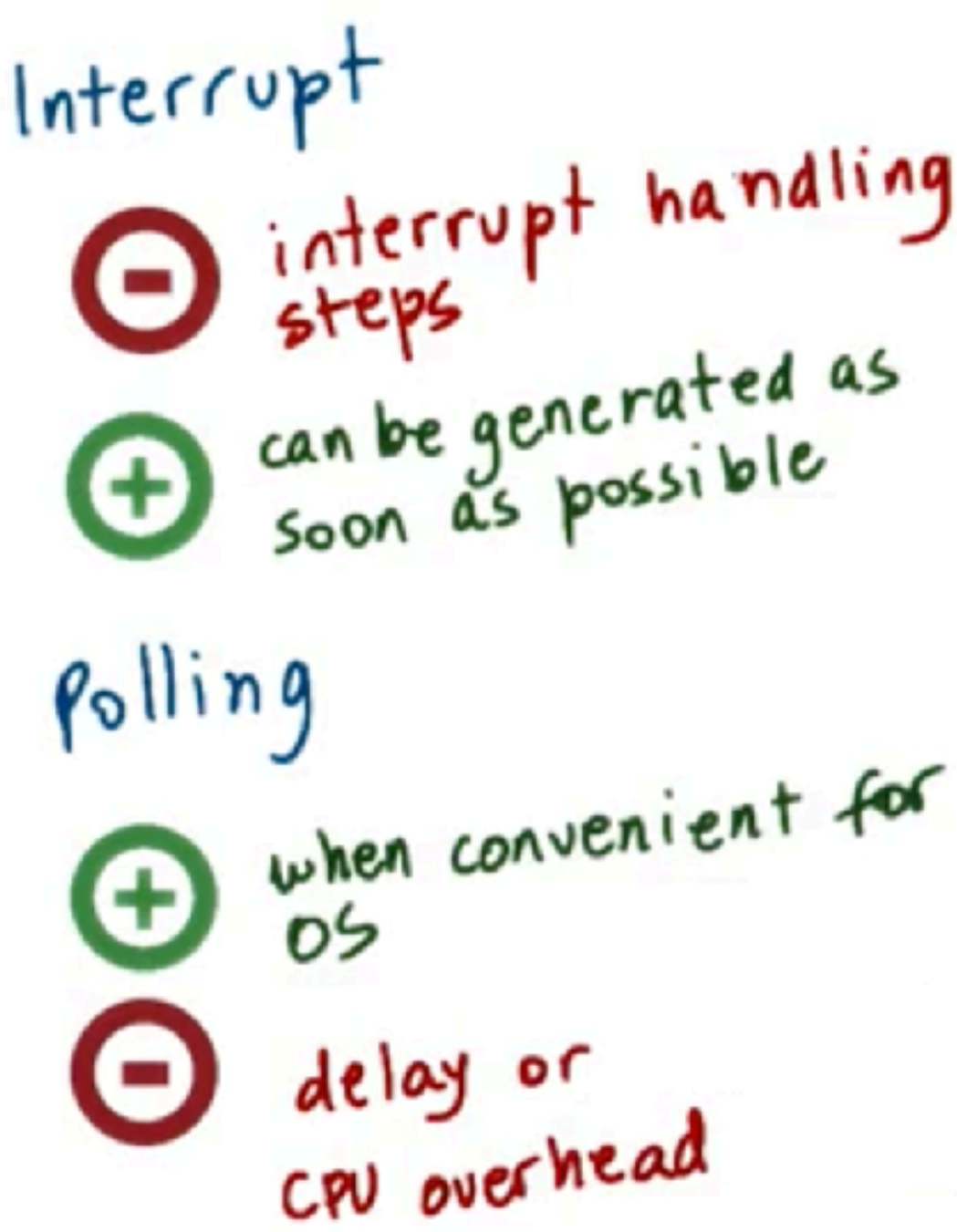


access device registers
== memory load/store

Memory-mapped I/O
- part of 'host' physical memory dedicated for device interactions
- Base Address Registers (BAR)

I/O port
- dedicated in/out instructions for device access
- target device (I/O port) and value in register

CPU/Device Interaction



Programmed I/O

no additional hardware support

CPU "programs" the device
- via command registers
- data movement

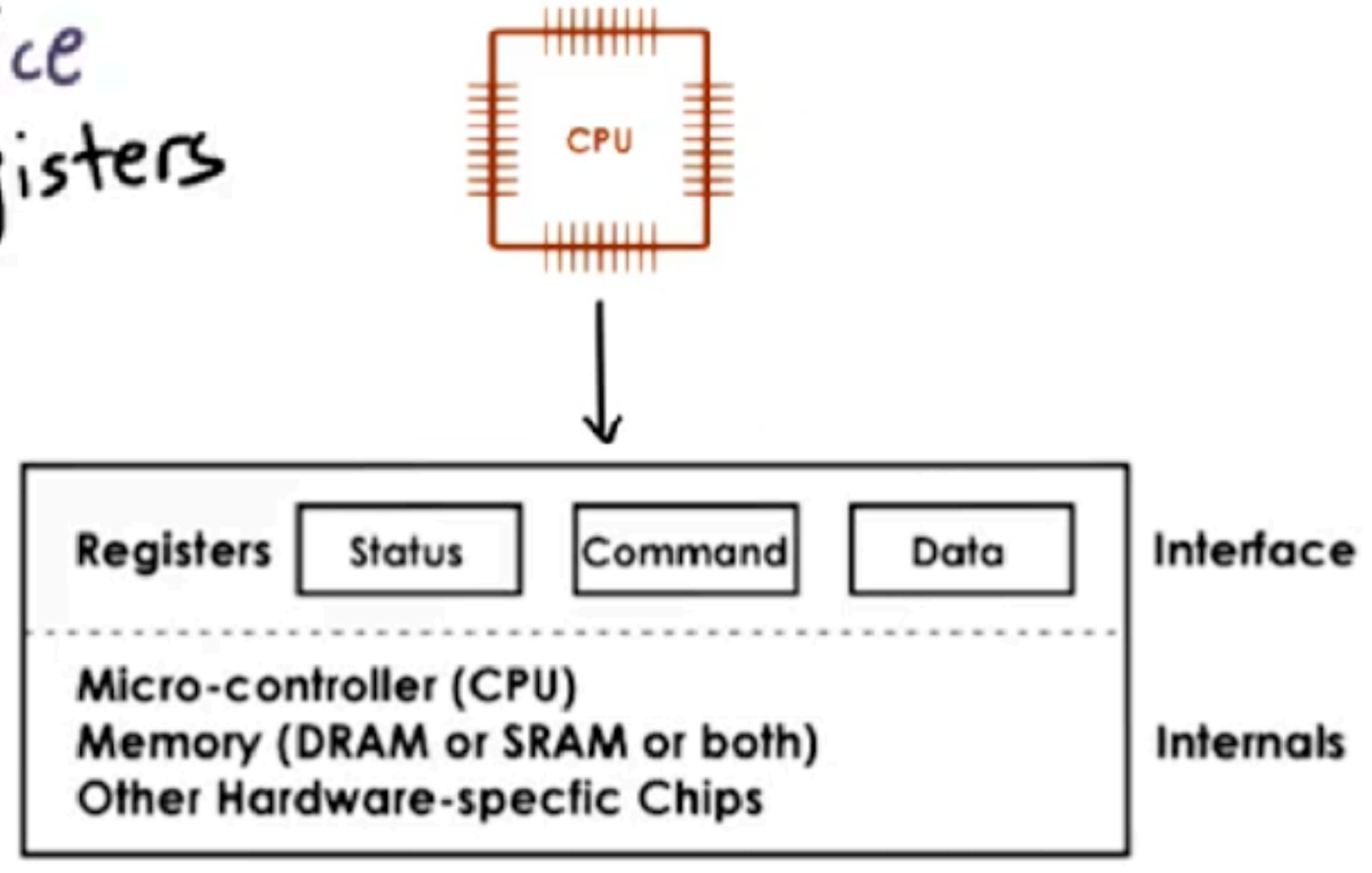


Figure 36.2: A Canonical Device

Programmed I/O

Example: NIC, data == network packet

- write command to request packet transmission
- copy packet to data registers
- repeat until packet sent

e.g., 1500 B packet; 8 byte regs or bus

=> 1 (for bus command) +
188 (for data)

=> 189 CPU store instructions

Direct Memory Access (DMA)

relies on DMA controller

CPU "programs" the device
- via command registers
- via DMA controls

For DMAs:
- data buffer must be in physical memory until transfer completes

=> pinning regions (non-swappable)

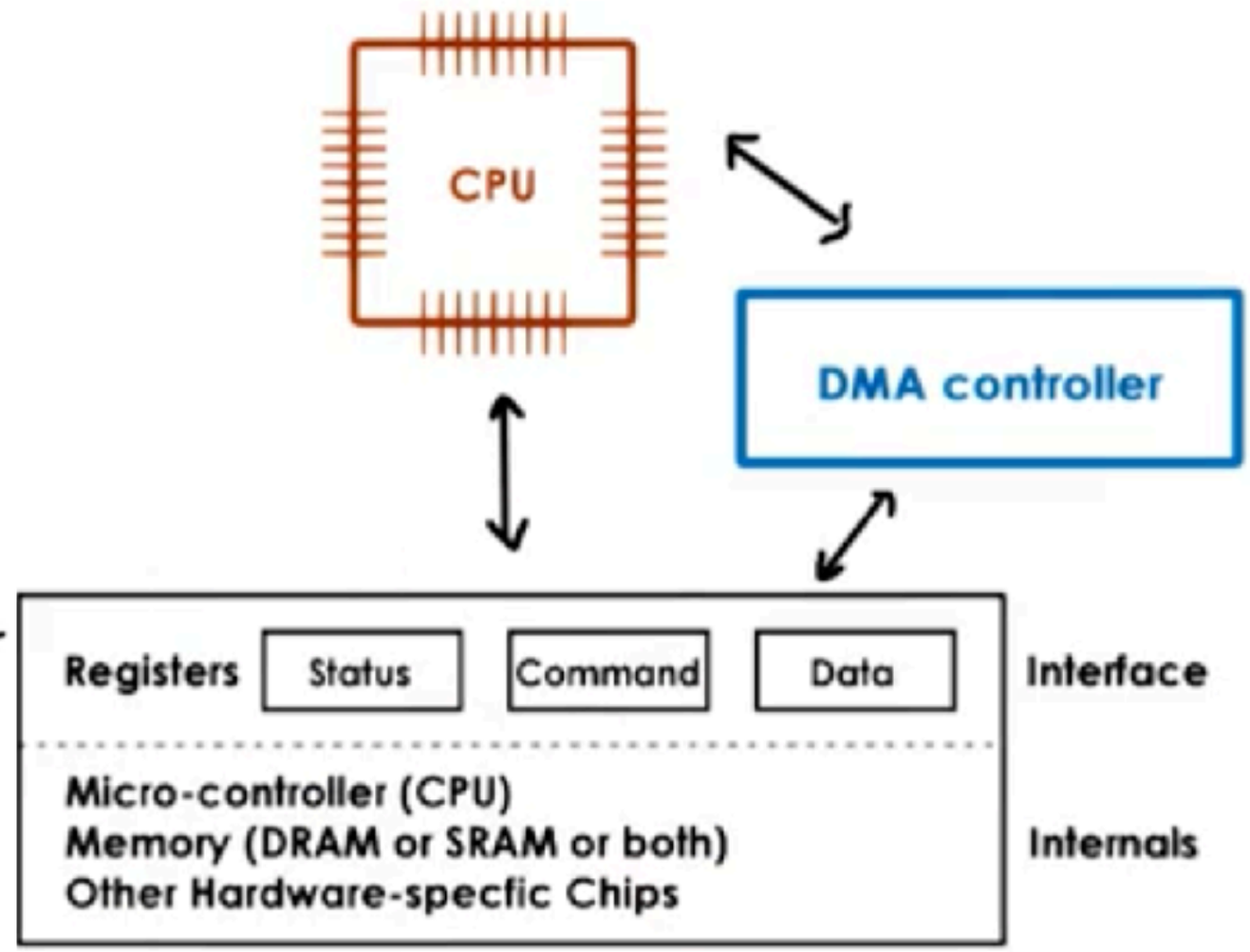


Figure 36.2: A Canonical Device

Example: NIC, data == network packet

- write command to request packet transmission
- configure DMA controller with in-memory address and size of packet buffer

e.g., 1500B w/ 8Byte regs or bus
=> 1 store instruction +
1 DMA configure

DMA vs. PIO Quiz

For a hypothetical system, assume the following:

- it costs 1 cycle to run a store instruction to a device register
- it costs 5 cycles to configure a DMA controller
- the PCI-bus is 8 bytes wide
- all devices support both DMA and PIO access

Which device access method is best for the following devices?

KEYBOARD :

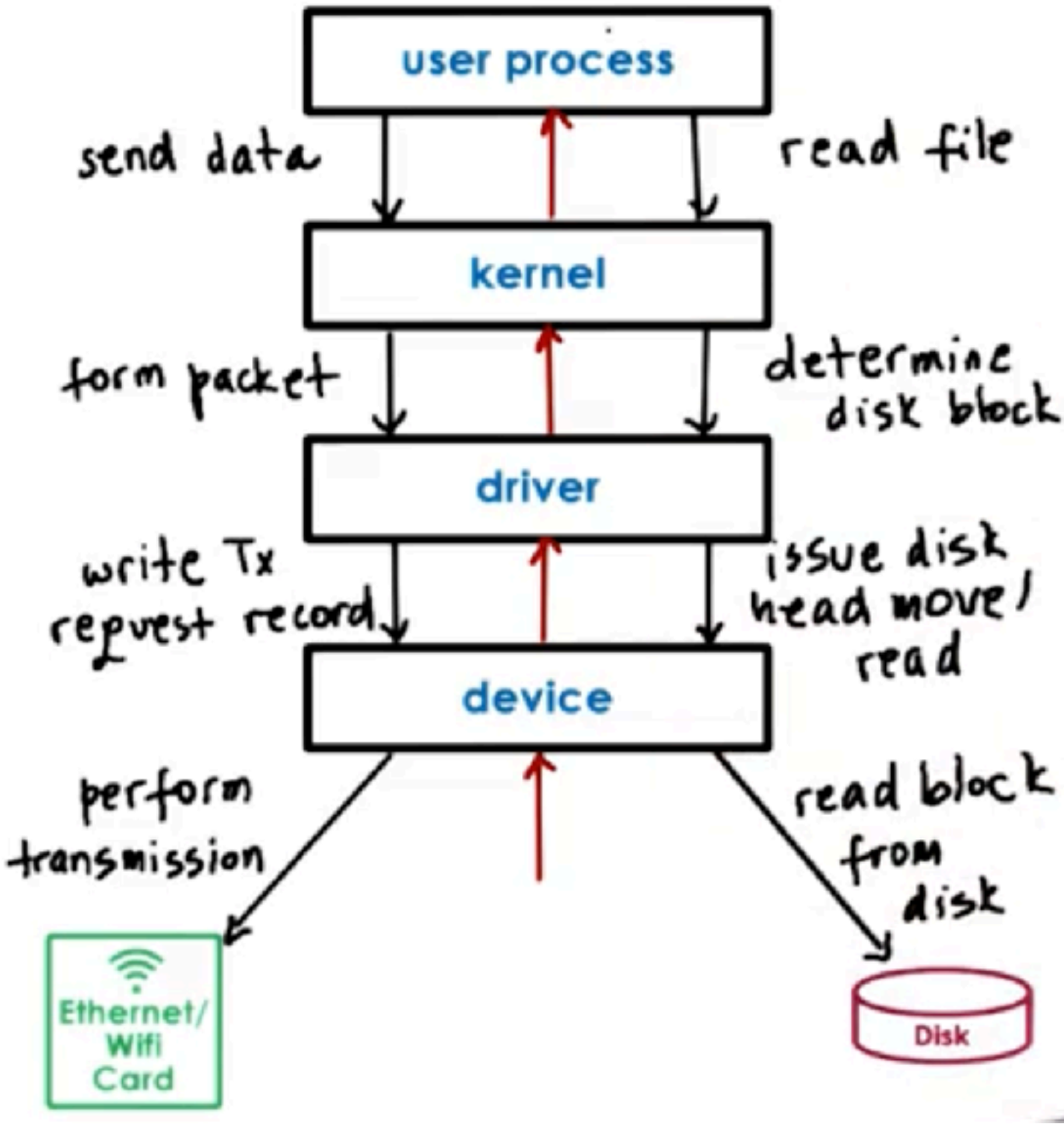
✓ PIO	○ DMA	○ Depends

NIC :

○ PIO	○ DMA	✓ Depends

Typical Device Access

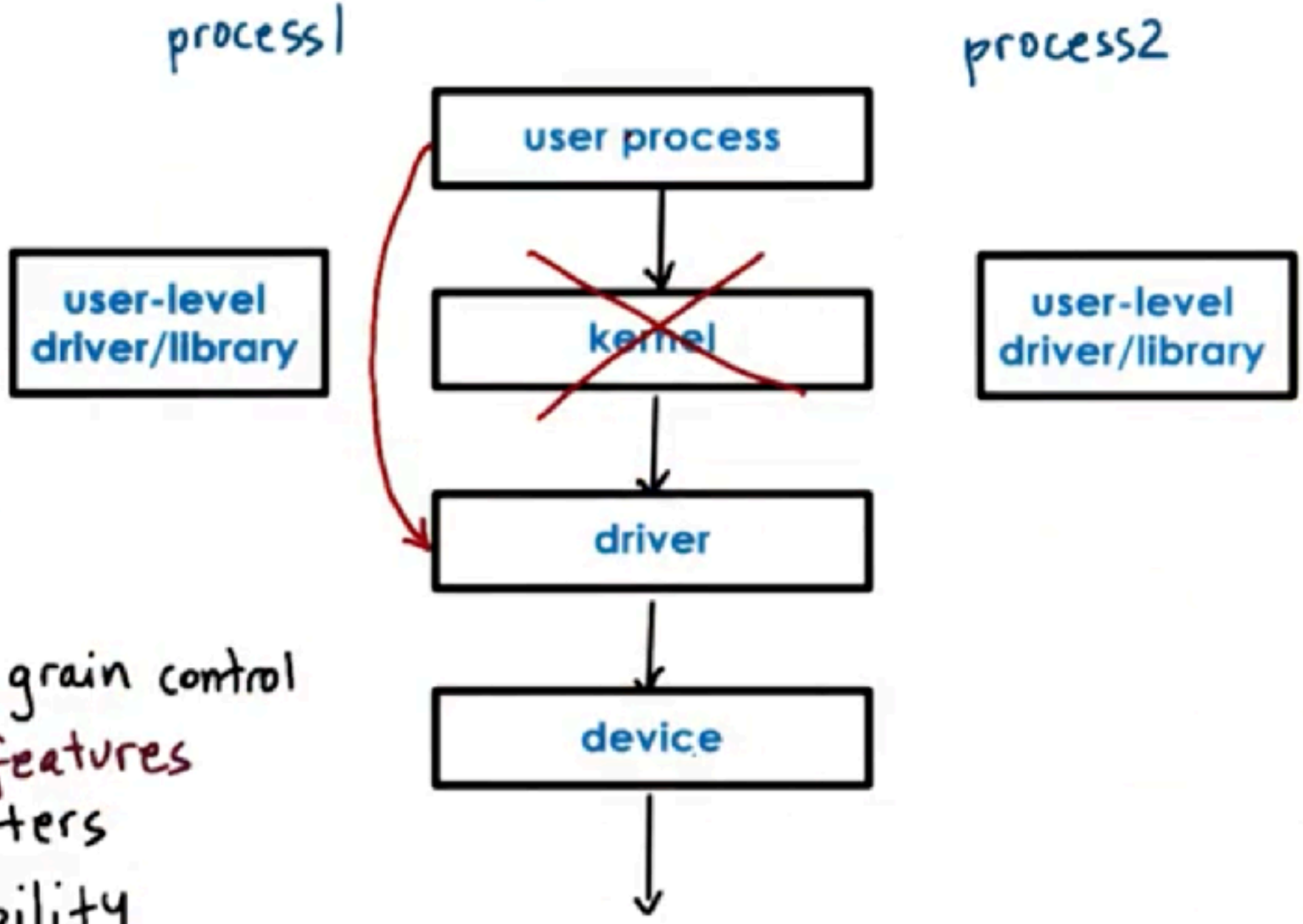
- system call
- in-kernel stack
- driver invocation
- device repuest configuration
- device performs request



Do You Have to Go Through the OS ?

OS Bypass

- device regs/data directly accessible
- OS configures then out-of-the way
- "user-level driver" (~ library)
- OS retains coarse-grain control
- relies on device features
  - sufficient registers
  - demux capability



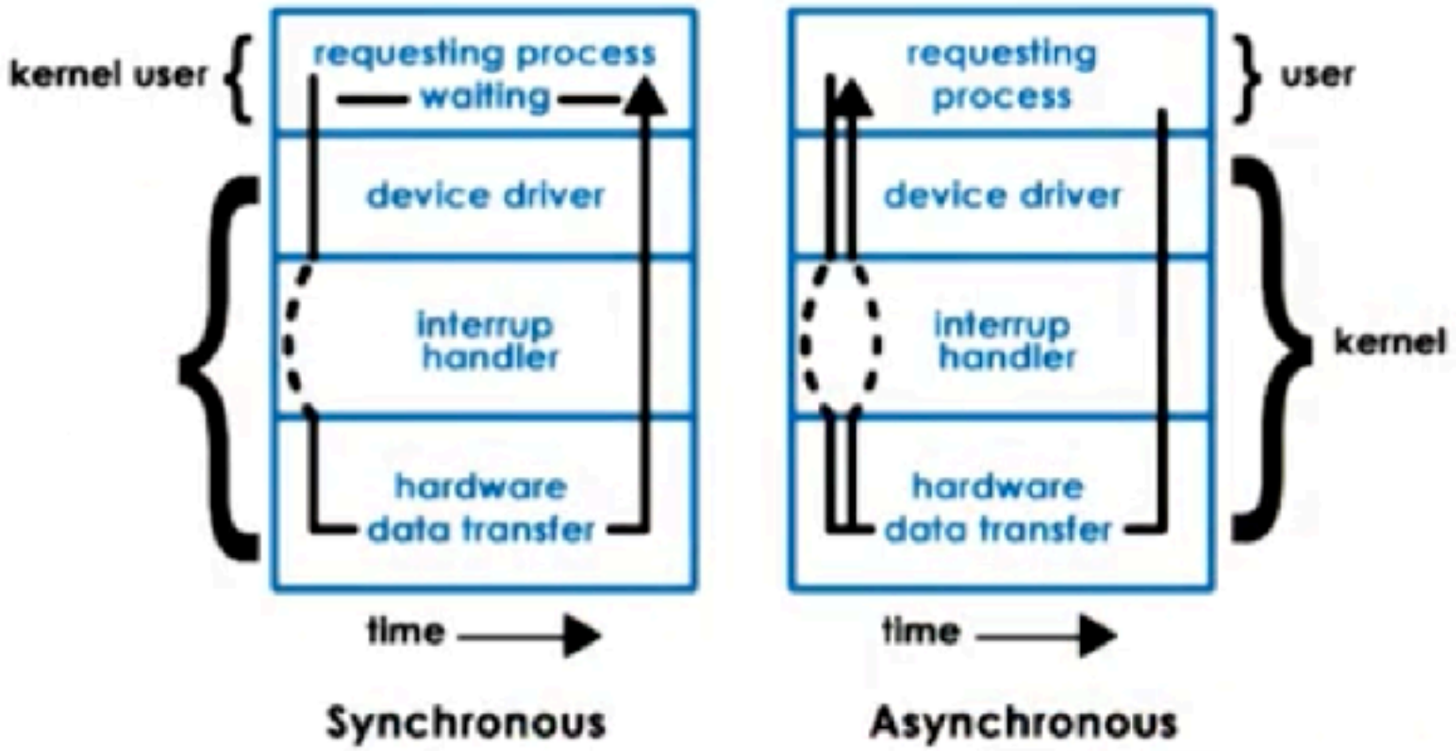
Sync vs Async I/O

Synchronous I/O operations
=> process blocks

Asynchronous I/O operations
=> process continues

Later ...
- process checks and retrieves result

OR
- process is notified that the operation completed and results are ready



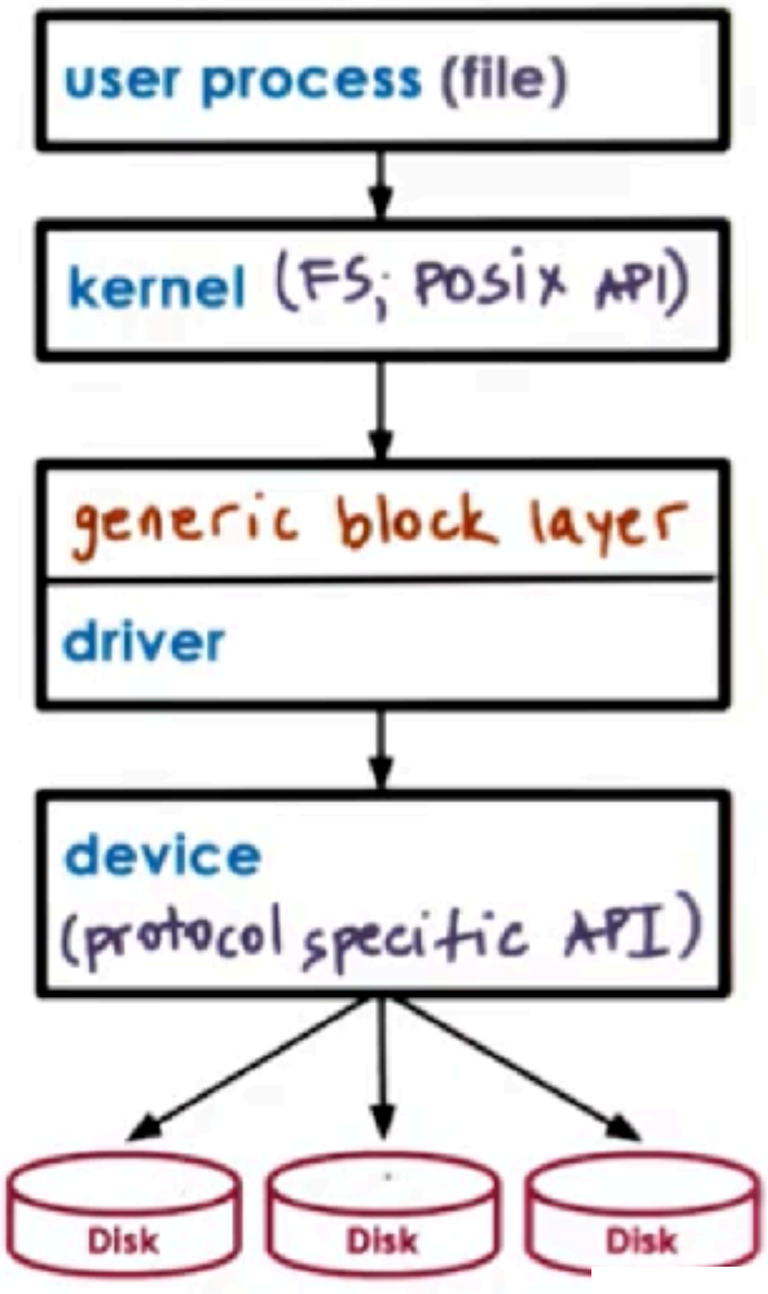
Block Device Stack

processes use files => logical storage unit

kernel file system (FS)
- where, how to find and access file
- OS specifies interface

generic block layer
- OS standardized block interface

device driver



block device
typical storage for files

Block Device Quiz

In Linux, the `ioctl()` command can be used to manipulate devices.

Complete the code snippet, using `ioctl()`, to determine the size of a block device.

```
// ...
int fd;
unsigned long numblocks = 0;

fd = open(argv[1], O_RDONLY);

ioctl(fd, BLKGETSIZE, &numblocks);

close(fd);

// ...
```