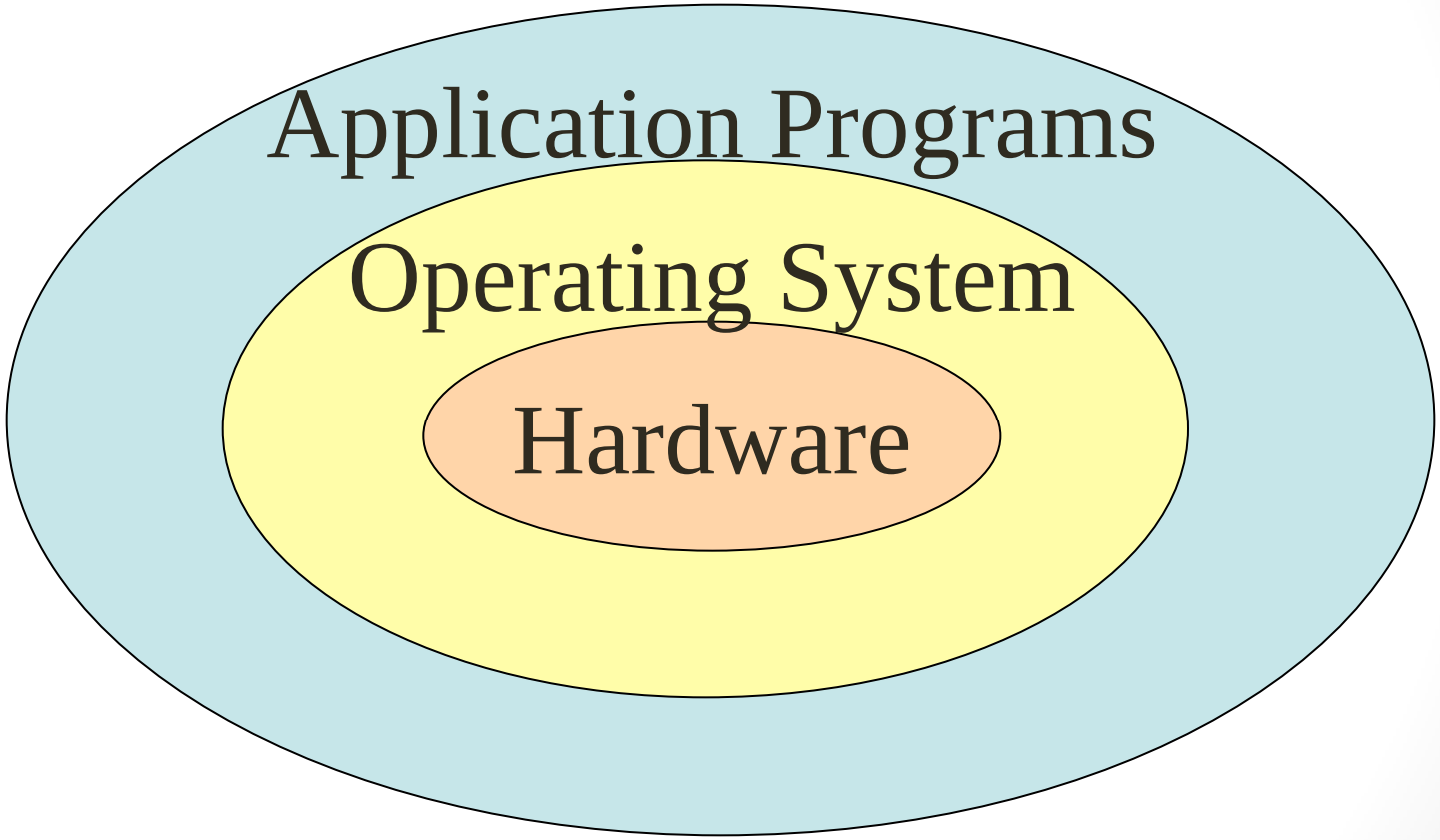
Operating System Design

I/O devices
Polling vs. Interrupt
DMA

Neda Nasiriani

Fall 2018

Macroscopic Abstract View of the Computer System



What is an Operating System?

A "program" that acts as an intermediary between a user of a computer and the computer hardware.

- The OS manages resources in the computer system.
- The OS controls the execution of programs.

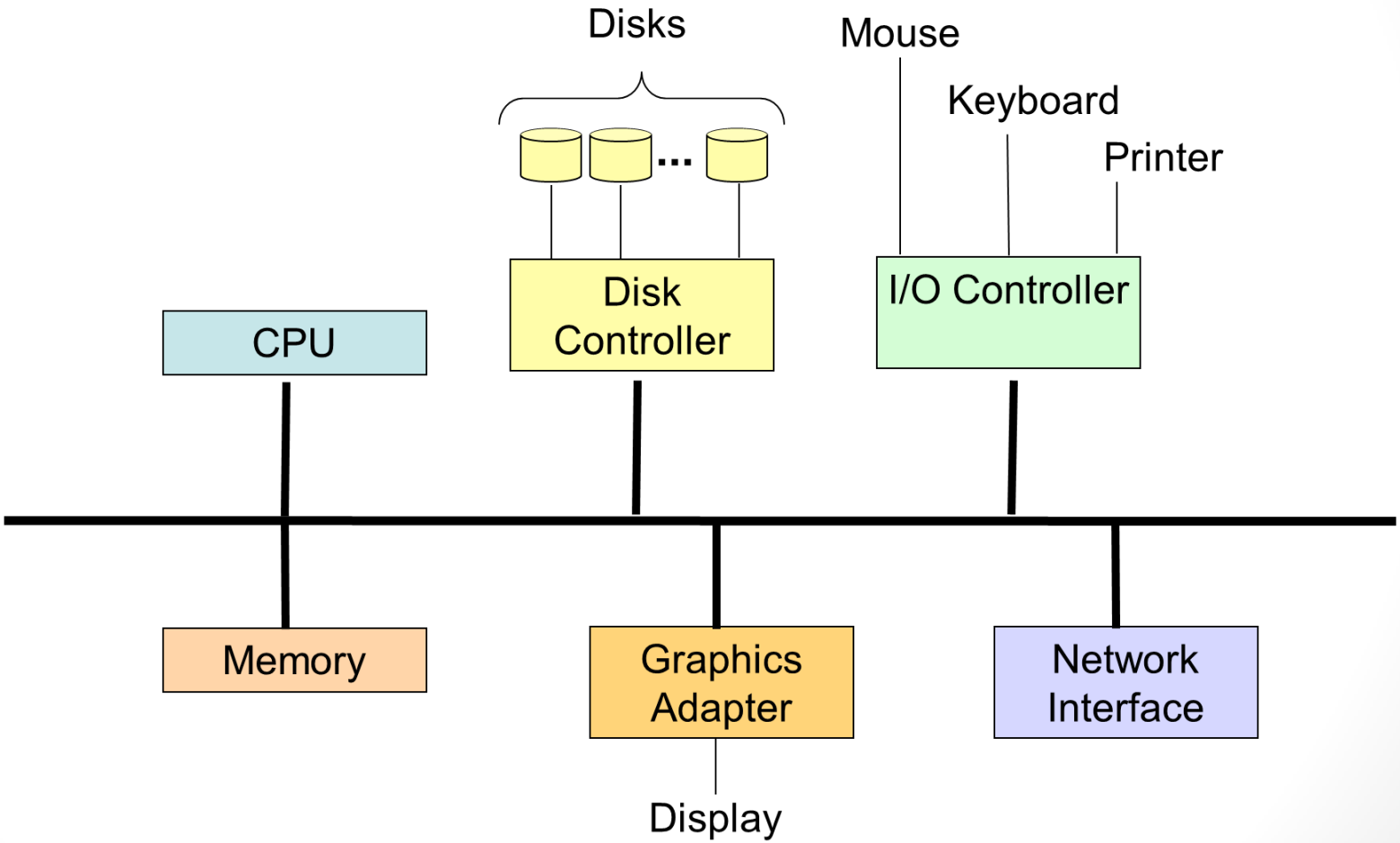
Operating System Definitions

- Resource allocator – manages and allocates resources.
- Control program – controls the execution of user programs and operations of I/O devices.
- Kernel – the one program “running” at all times (all else being application programs).

Small Quiz!

I/O Devices

Computer System



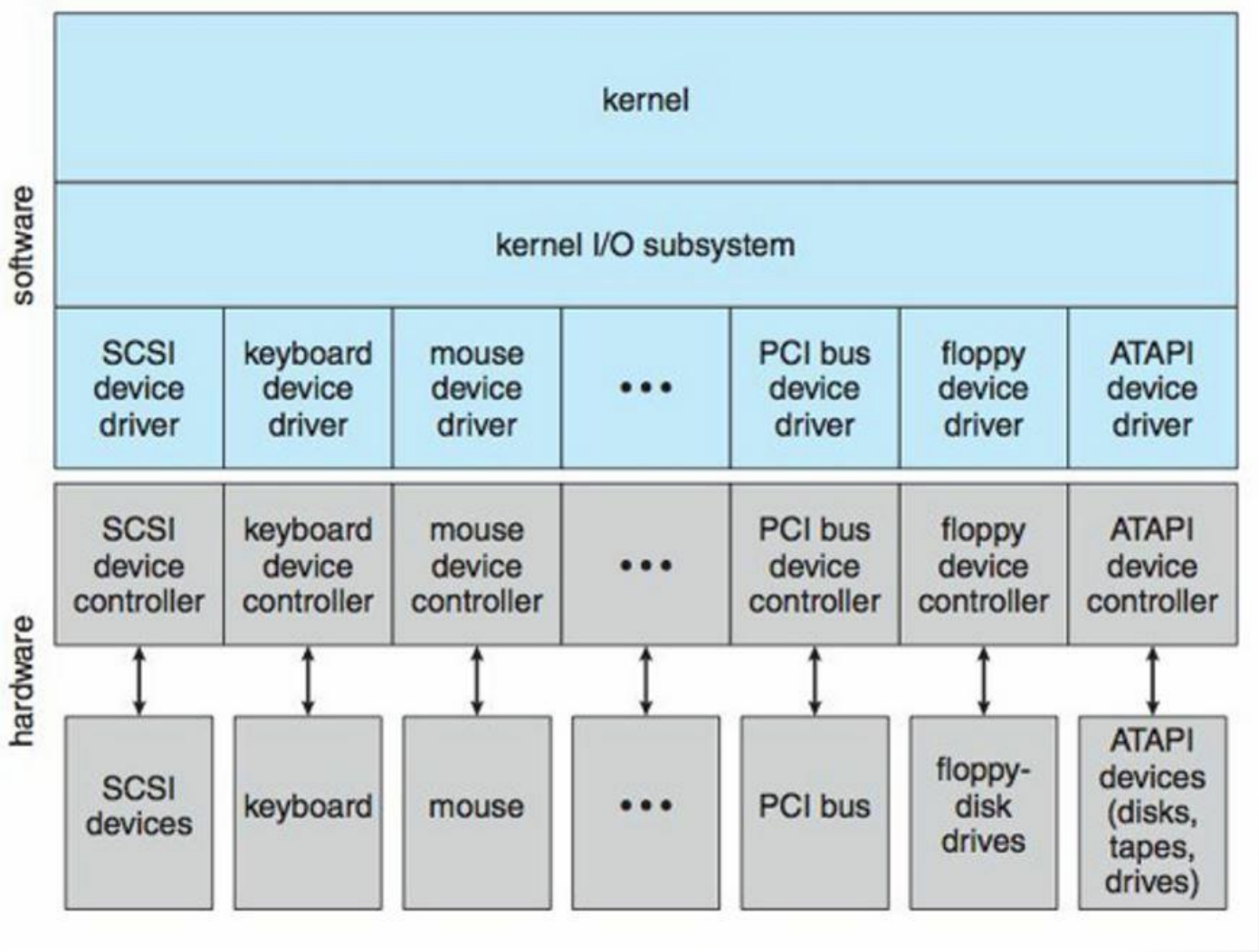
What kind of I/O devices?

Three main categories:

1) Storage devices
   - Disk, tapes
2) Transmission devices
   - Network connections, Bluetooth
3) Human-interface devices
   - Screen, keyboard, mouse, audio in and out

- How does these devices communicate with computer system?

A kernel I/O structure



I/O Devices

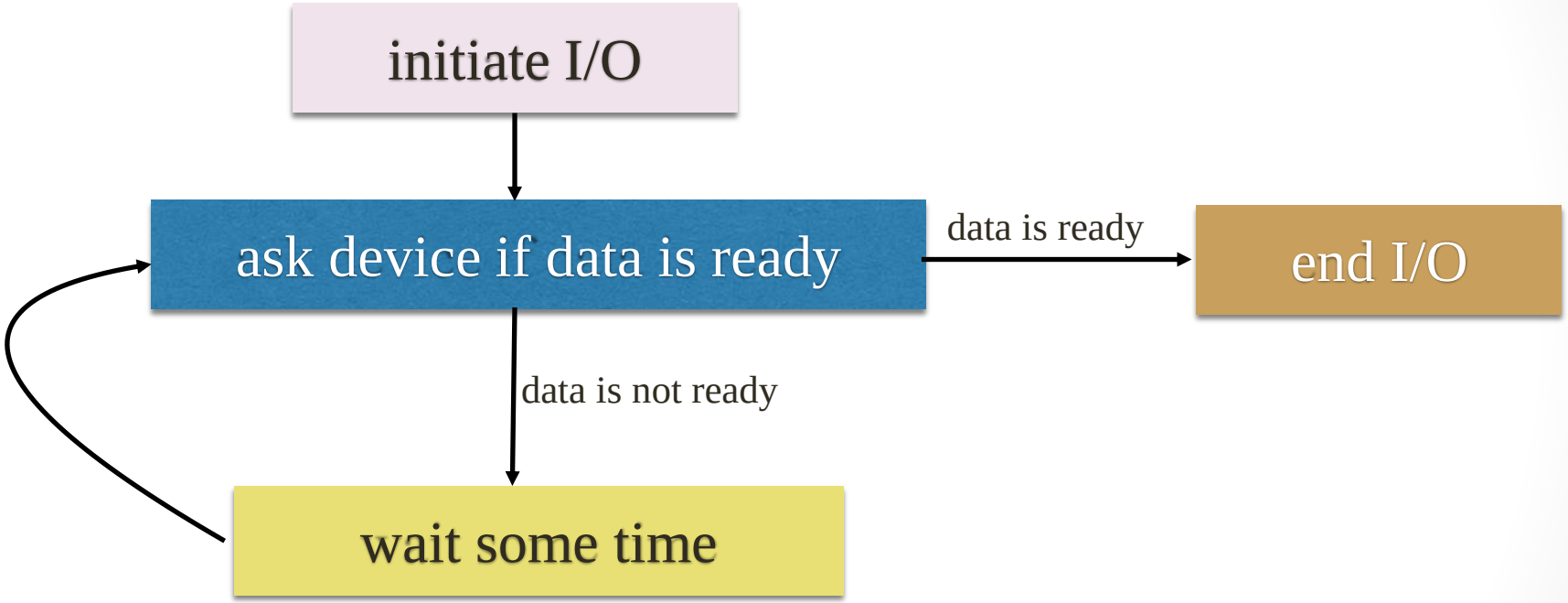
- Communication with computer system
  - **Port:** connection point for device
  - **Bus:** Shared wire among different devices
  - **Memory-mapped I/O:** device-control registers are mapped to the address space of the CPU. The CPU executes I/O requests using the standard data transfer instructions at their memory address
  - **Controller** (**host adapter**) – electronics that operate port, bus, device
    - Contains processor, microcode, private memory (buffer), bus controller, etc

Operating System Operations

Assumptions:

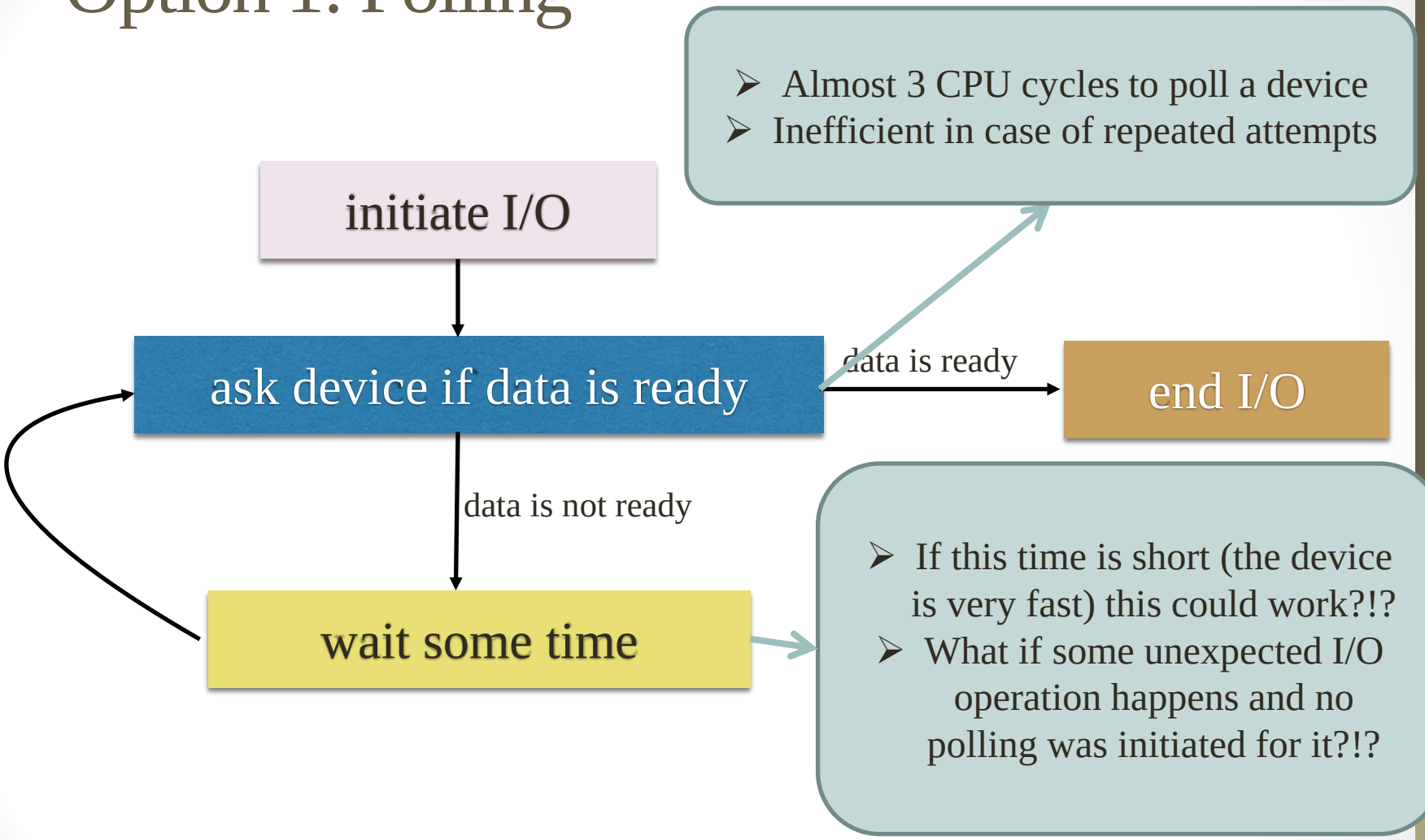
- I/O devices and the CPU can execute concurrently.
- Each *device controller* is in charge of a particular device type.
- Each device controller has a *local buffer*.
- There must be some mechanism to move data from/to main memory to/from local buffers.
- I/O operations move data from the device to a controller's local buffer.
- There must be some mechanism for the CPU to learn that an I/O operation has completed.

Option 1: Polling



The Simpsons: https://www.youtube.com/watch?v=18AzodTPG5U

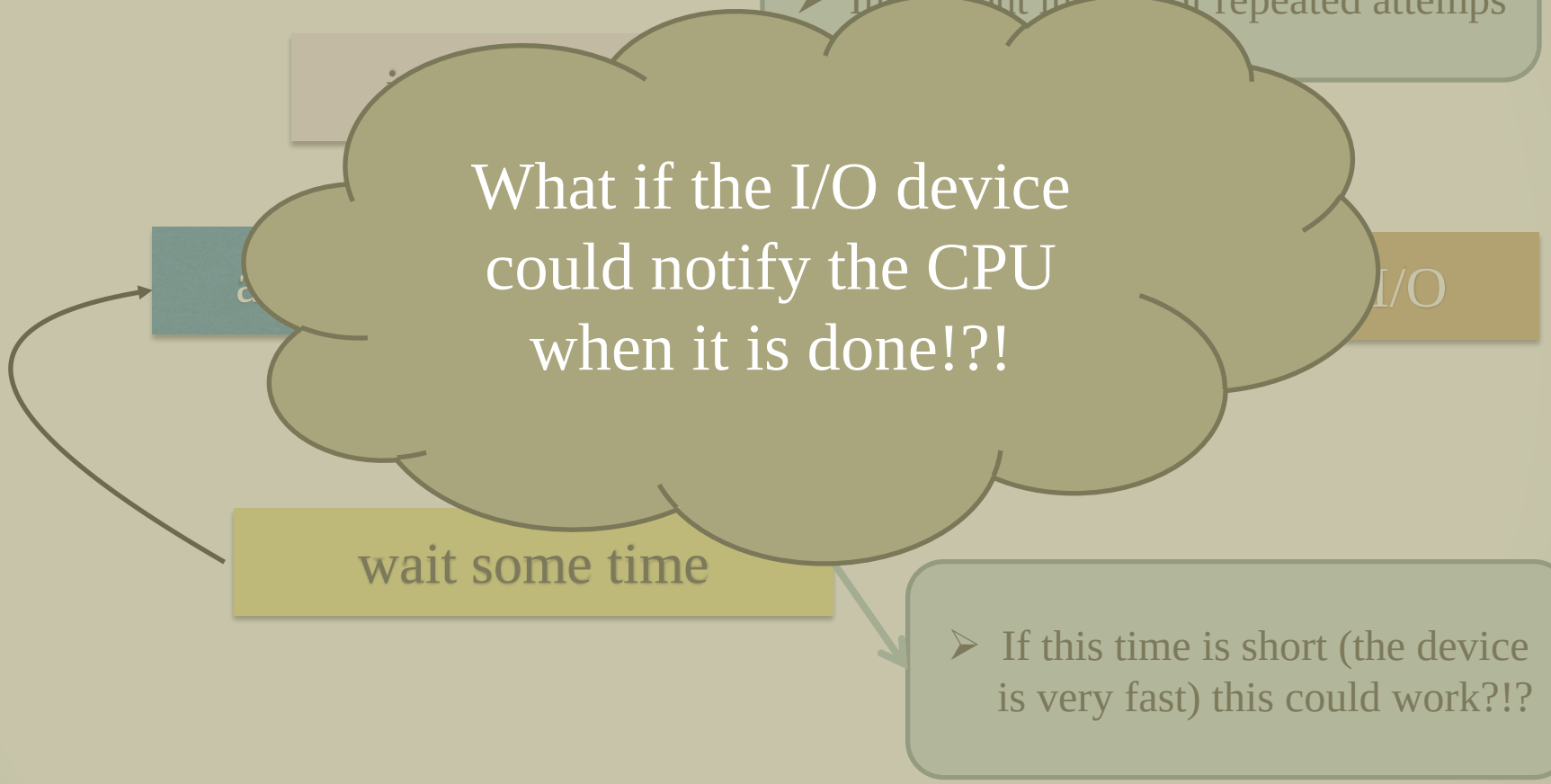
Option 1: Polling



The Simpsons: https://www.youtube.com/watch?v=18AzodTPG5U

Option 1: Polling

- Almost 3 CPU cycles to poll a device
- Inefficient in case of repeated attemps



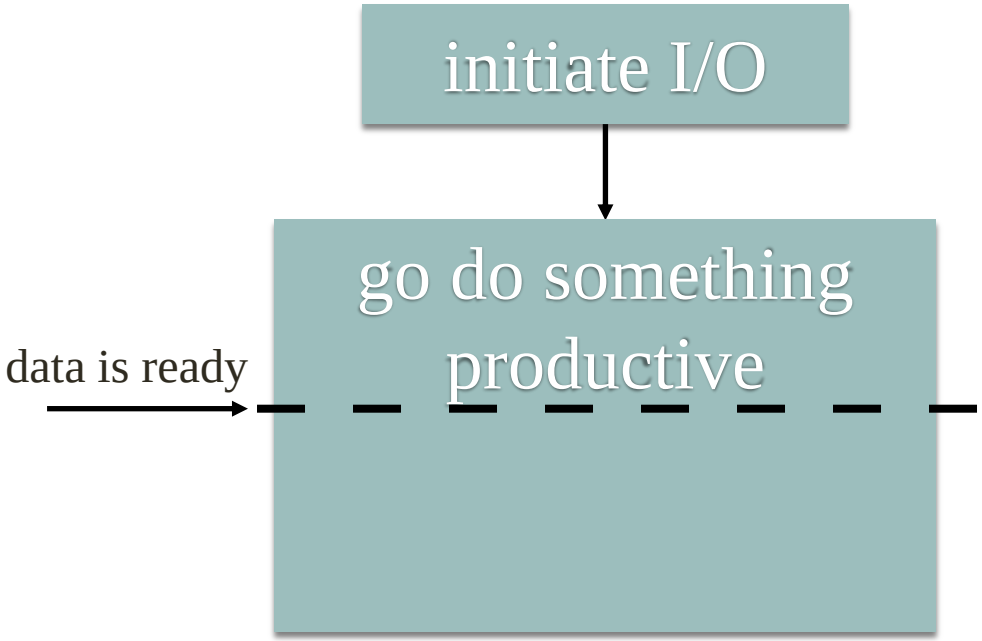
- If this time is short (the device is very fast) this could work?!?

The Simpsons: https://www.youtube.com/watch?v=18AzodTPG5U

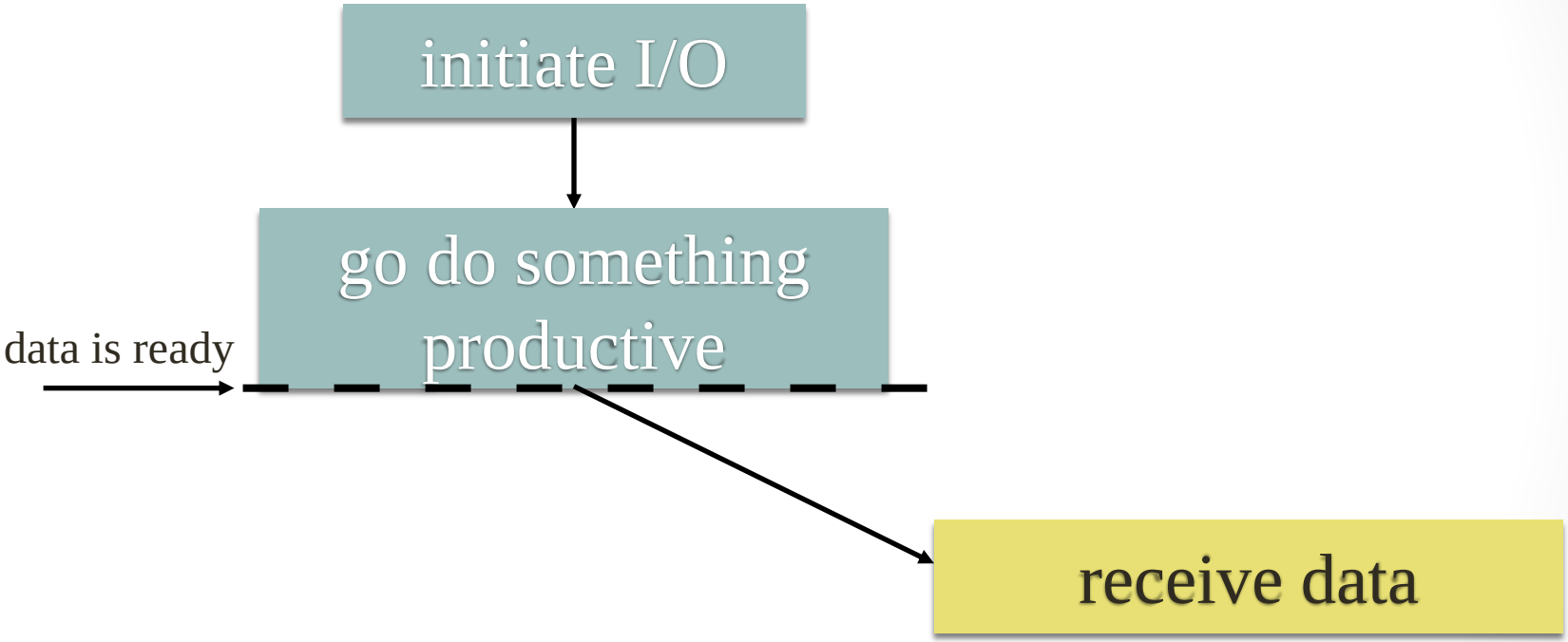
FYR: Polling

- For each byte of I/O
  1. Check if the I/O device is ready (Read busy bit from status register until 0)
  2. Host sets read or write bit and if write, copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done

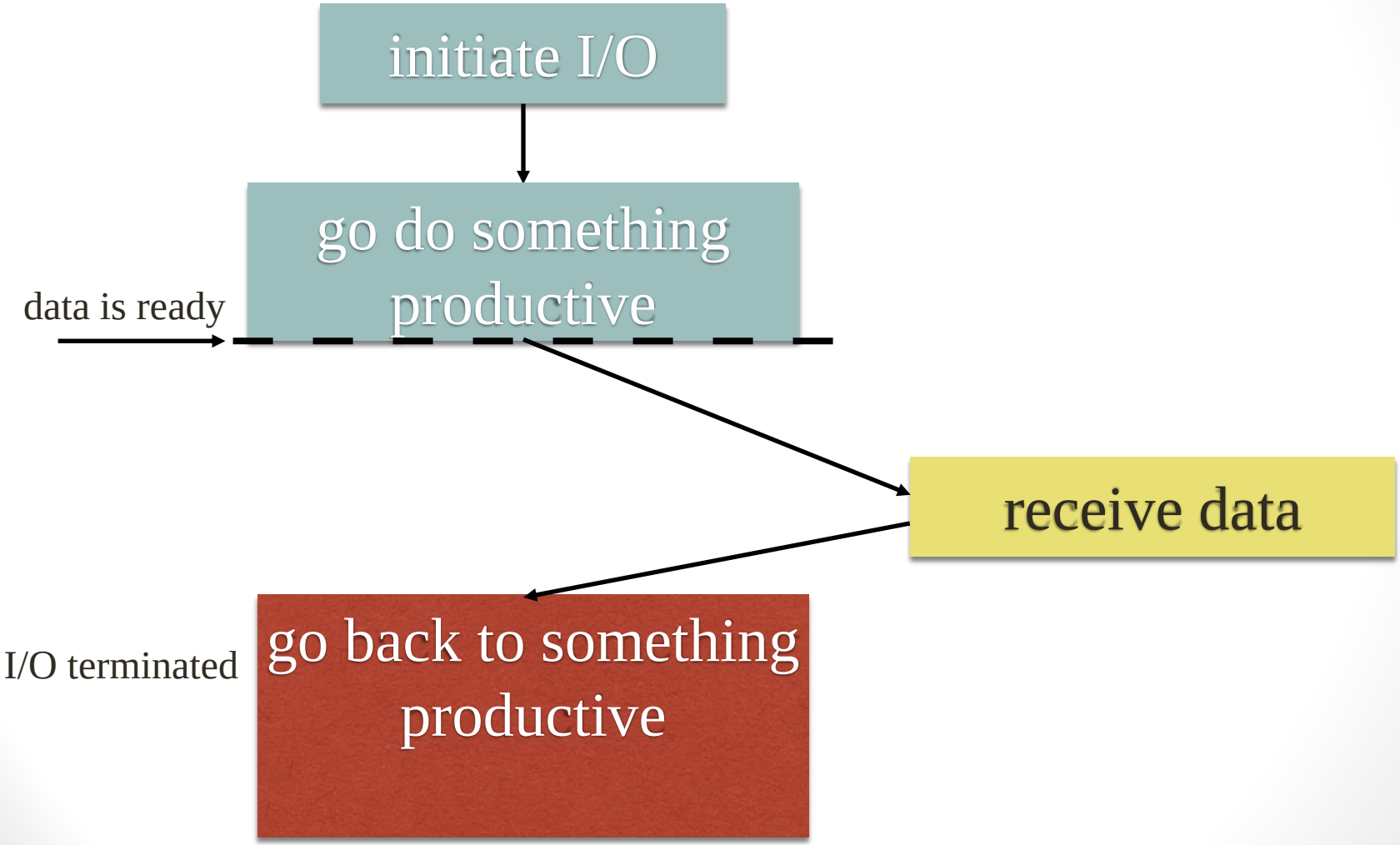
Option 2: Interrupt



Interrupt


initiate I/O
go do something productive
data is ready
receive data

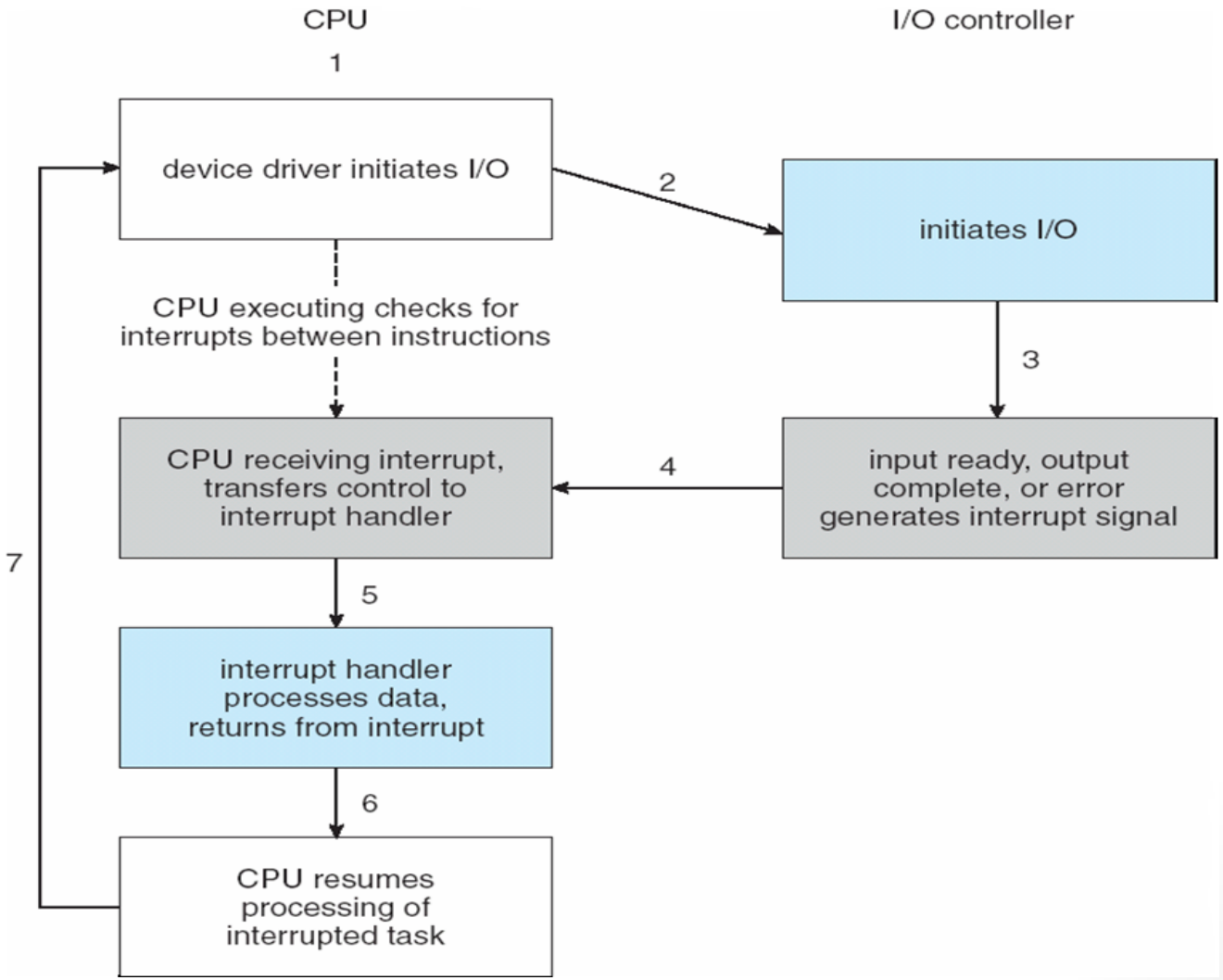
Option 2: Interrupt



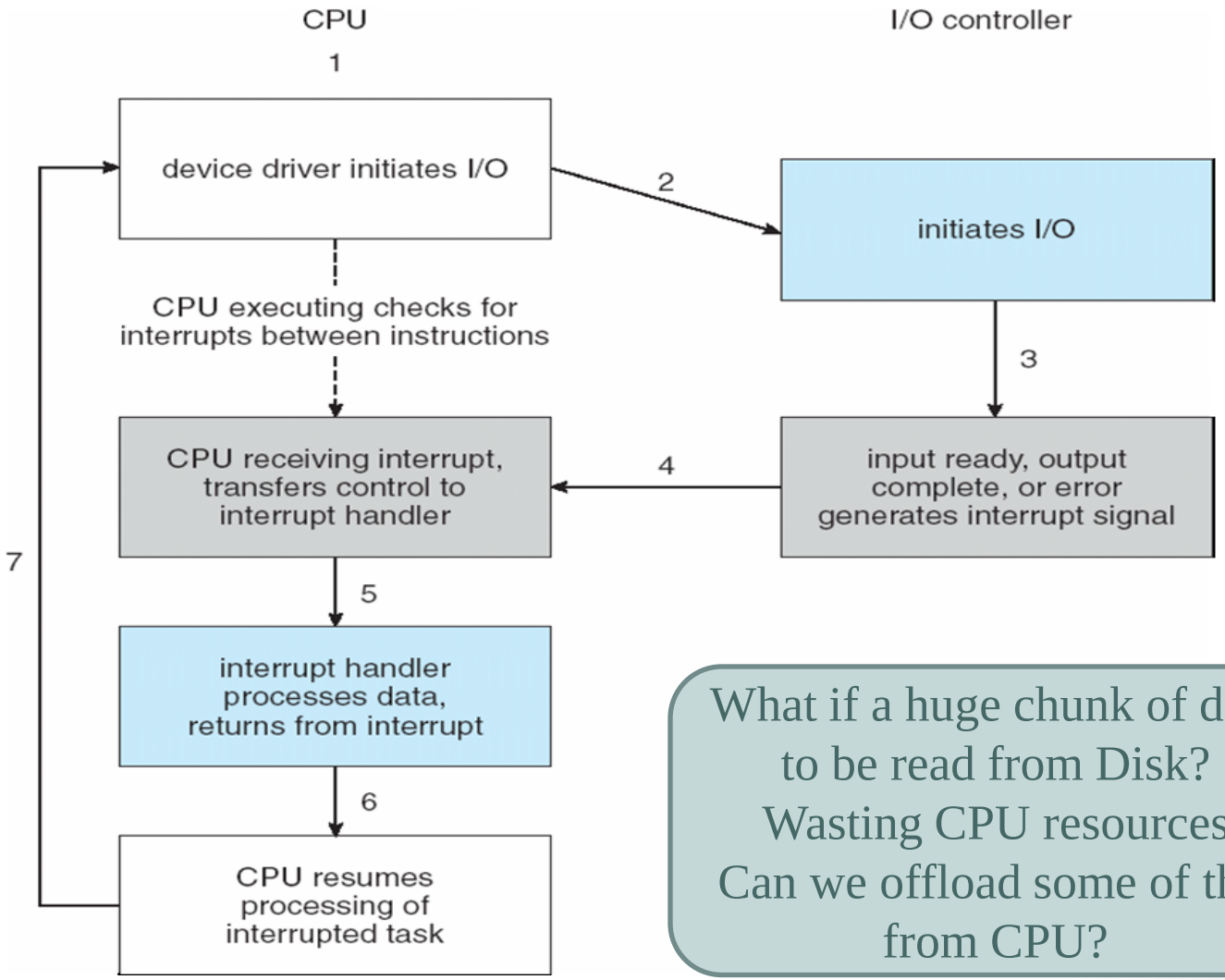
Interrupts

- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining if more than one device at same interrupt number
- Remember: **Traps** are software interrupts

Interrupt Driven I/O Cycle



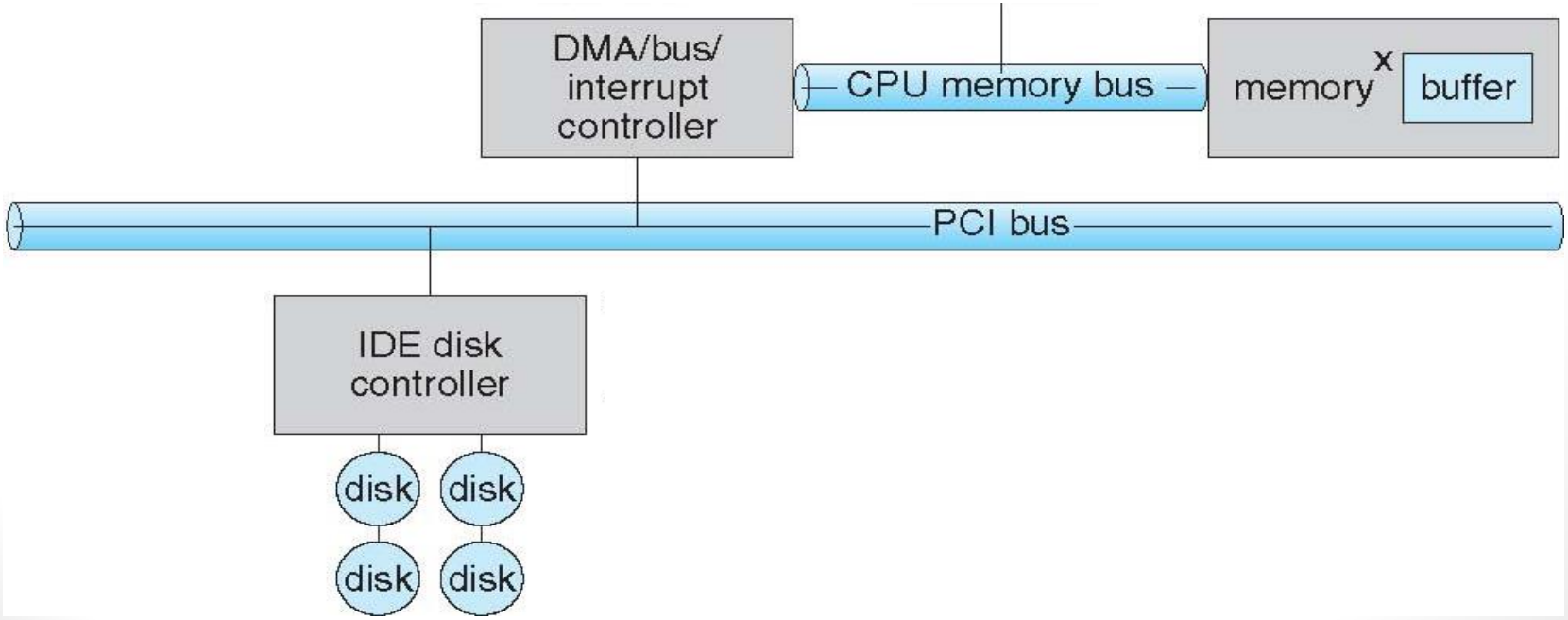
Interrupt Driven I/O Cycle



What if a huge chunk of data to be read from Disk?
Wasting CPU resources
Can we offload some of this from CPU?

Direct Memory Access (DMA)

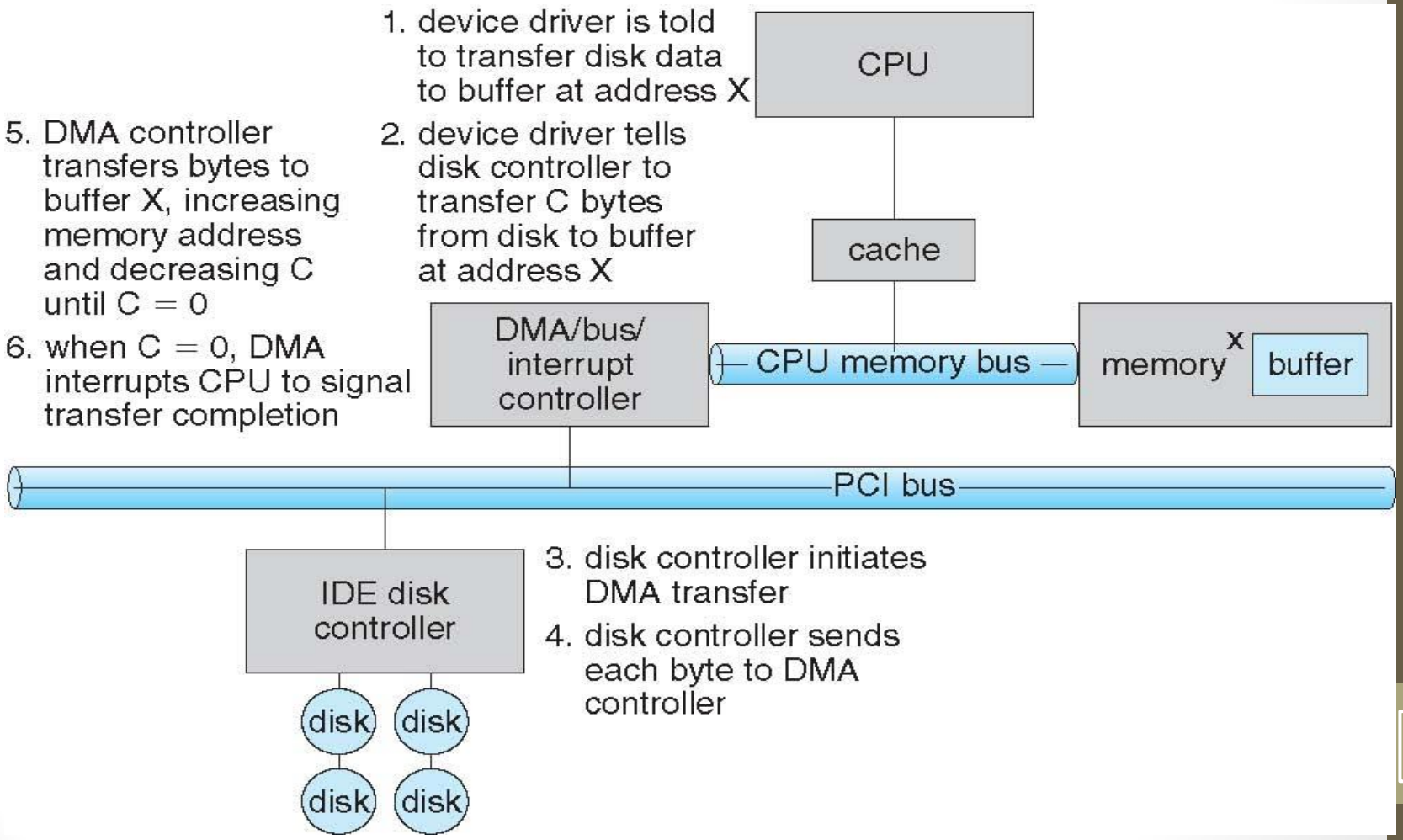
- What if we have a simpler controller that knows
  - the data location to be read from the disk
  - the memory location that it should be copied too
  - And can take care of this…



Direct Memory Access (DMA)

- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
- The CPU writes location of command block to DMA controller
- Handshaking between DMA controller and Device controller
  - DMA-request and DMA-acknowledge (for each word of data transfer)
  - Bus mastering of DMA controller – grabs bus from CPU
    - **Cycle stealing** from CPU but still much more efficient
- When done, DMA controller interrupts to signal completion

DMA: How it works



Activity!

Next Session

- Booting the OS