

CISC 3320

C05b: I/O Hardware and Schemes

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- This slides are a revision of the slides by the authors of the textbook

Outline

- I/O Device/Hardware
- Role of Device Driver
- Accessing I/O Devices
- I/O Schemes

Overview of I/O Management

- I/O management is a major component of operating system design and operation
 - Important aspect of computer operation
 - I/O devices vary greatly
 - Various methods to control them
- Performance management
 - New types of devices frequent
 - Ports, busses, device controllers connect to various devices
 - Device drivers encapsulate device details
 - Present uniform device-access interface to I/O subsystem

I/O Hardware

- A few general categories
 - Storage devices
 - Examples: Disks, tapes, solid state drives
 - Transmission devices
 - Examples: network adapters, modems
 - Human-interface devices
 - Examples: display screens, keyboard, mouse, touch screen
- Specialized devices
 - Examples: I/O devices in control cars, robots, aircrafts, spacecrafts

Common Concepts

- Signals from I/O devices interface with computer via:
 - Port
 - Bus
 - Device controller

Port

- Port: connection point for device
 - Devices communicate with a computer via this connection point
 - (Physical) port
 - Examples: USB port, serial port, parallel port
 - (Logical) port

Bus

- Daisy chain or shared direct access
 - A common set of wires with a protocol that specifies commands that can be transmitted
 - Examples:
 - PCI bus common in PCs and servers, PCI Express (PCIe)
 - Expansion bus connects relatively slow devices

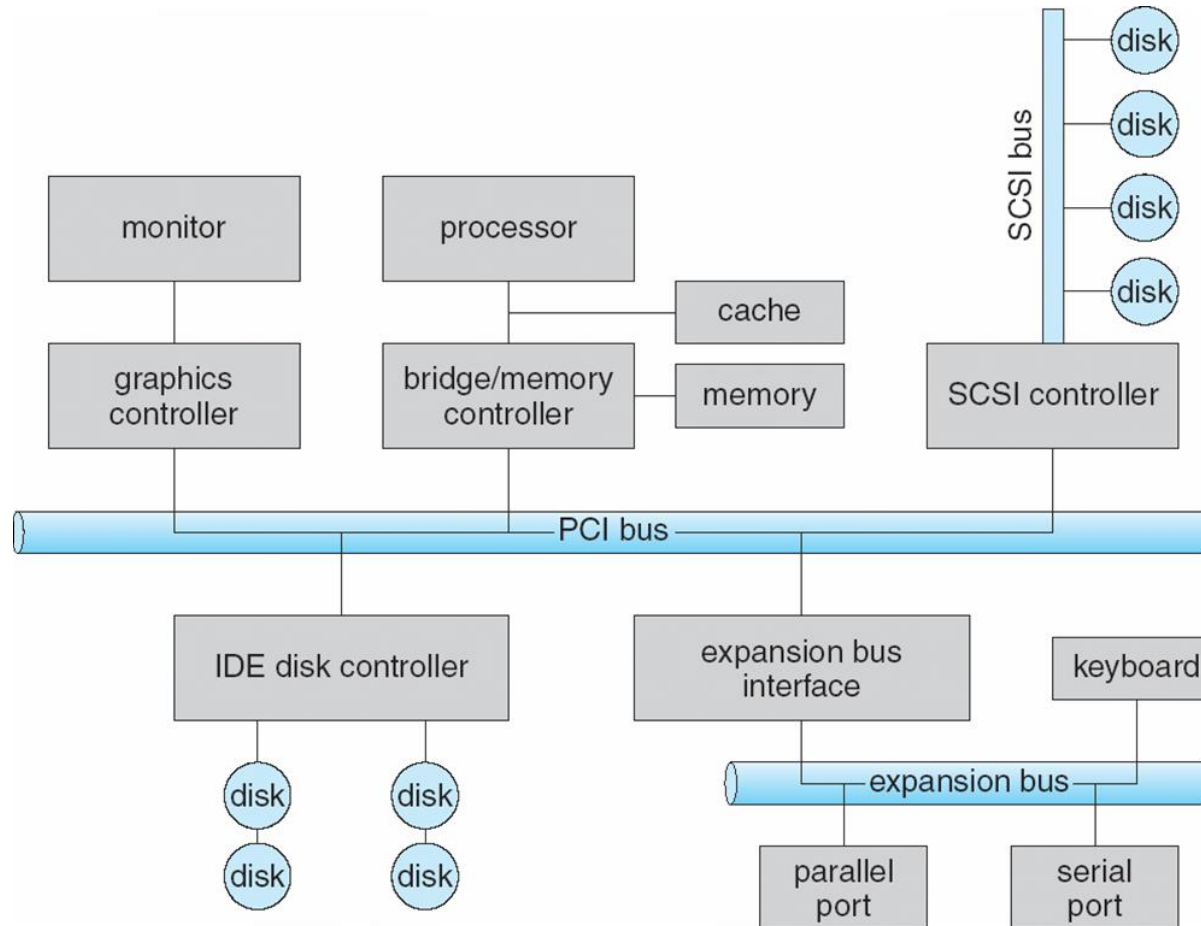
Device Controller

- Devices
 - Example: hard disk drives have motors, magnetic headers, and disks
- Controller, also called host adapter
 - A collection of electronics that operate a port, a bus, or a device (some contain small embedded computer)
 - Accept and act on commands from the OS
 - Present a simpler interface to the OS
 - Examples: SATA controller

Variety of Controllers

- Sometimes integrated
- Sometimes separate circuit board (host adapter)
- Contains processor, microcode, private memory, bus controller, etc
- Some talk to per-device controller with bus controller, microcode, memory, etc

A Typical PC Bus Structure



Questions?

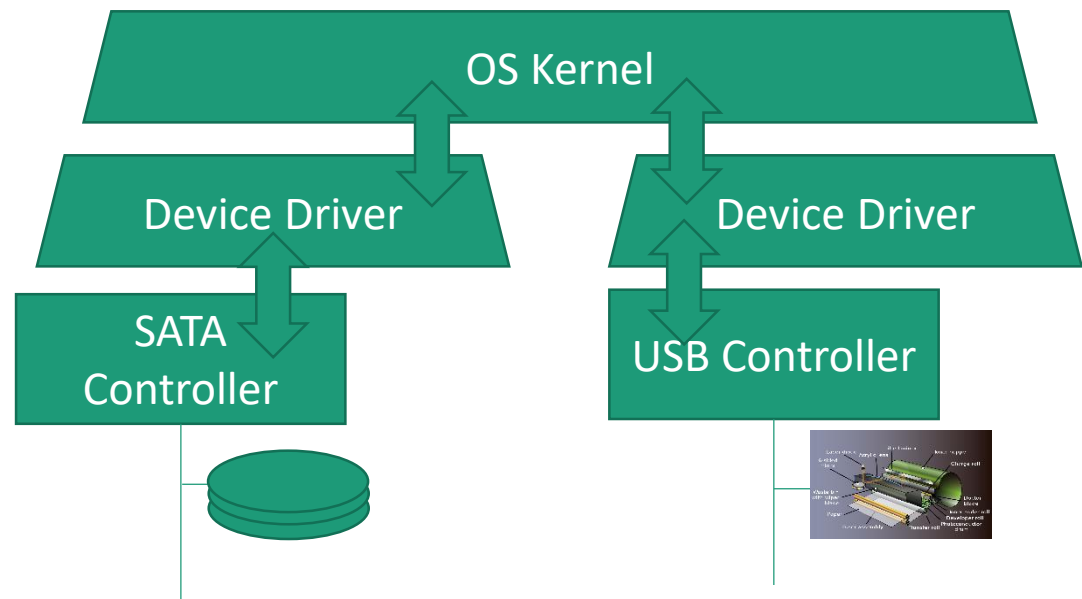
- Variety of devices
- Port, bus, and device controller

Needing Device Driver

- Each type of controller is different
- A piece of software called device driver communicates to the controller, and the OS
 - Adhere to some standard when communicating to the OS

Device, Driver, and OS

- Reduce complexity, increase uniformity and reliability



Questions?

- Role and benefit of having device drivers?

I/O Instructions

- I/O instructions control devices via device controllers
- Device controller usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution

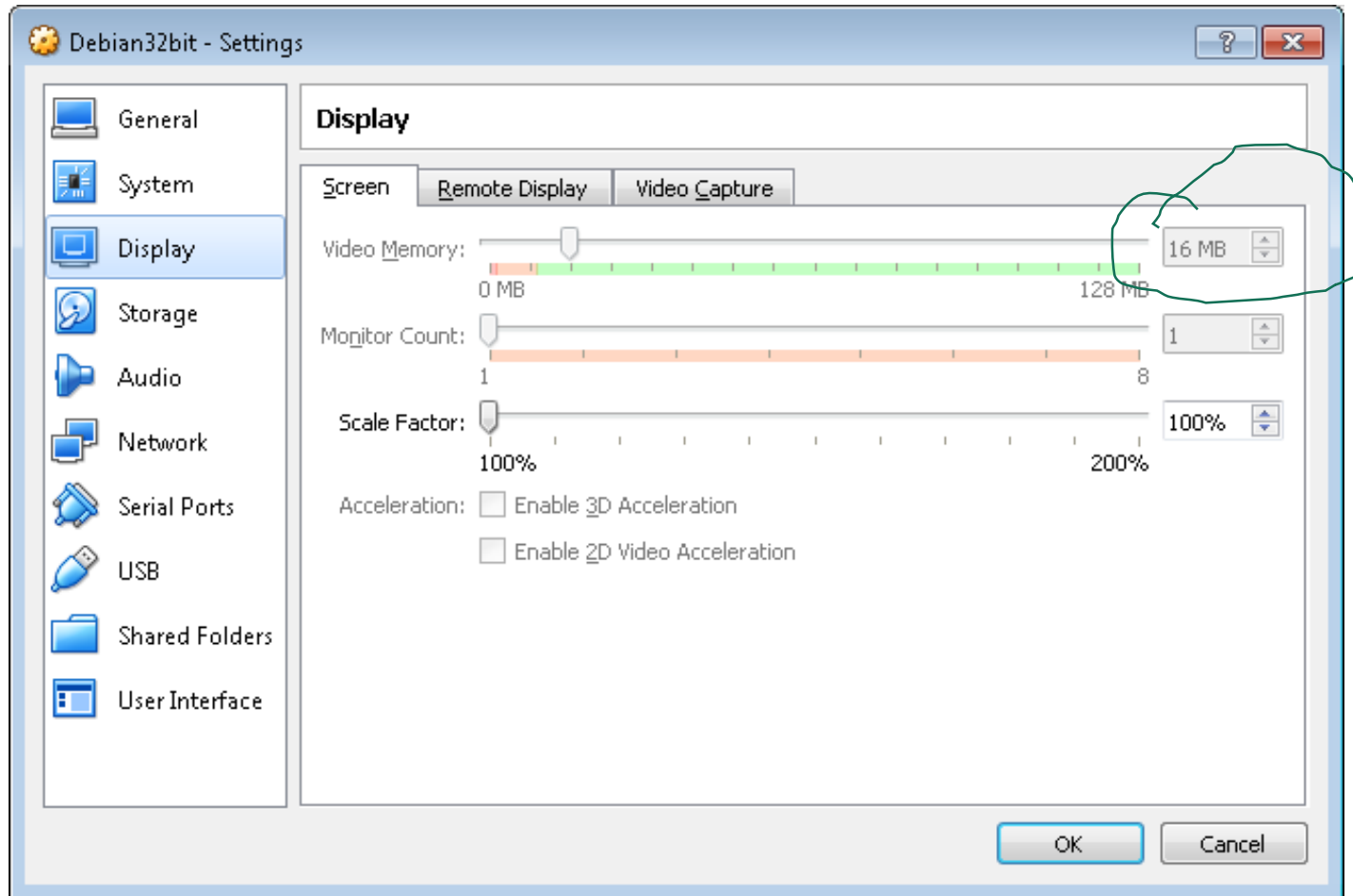
Controller Registers

- Typically have 4 registers or more
- Typically 1-4 bytes
 - Data-in register
 - Read by the host
 - Data-out register
 - Written by the host
 - Status register
 - A number of bits indicating the status of the device (e.g., busy, error)
 - Control register
 - A number of bits indicating the mode of the device

Controller Data Buffer

- May have a data buffer, e.g., FIFO buffer
 - Examples: video adapter (video memory)

VM VirtualBox: Allocating Device Buffer



Device Addresses

- Devices have addresses (logical port), used by
 - Direct I/O instructions
 - Memory-mapped I/O
 - Device data and command registers mapped to processor address space
 - Especially for large address spaces (graphics)

Device Address (I/O Port Space)

- Each register is assigned an address, sometimes called an I/O port number
 - Typically, a 8-bit or 16-bit integer
 - All I/O port numbers form the I/O port address space
- A CPU has I/O instructions
 - Example instruction (in an assembly language):
 - `IN REG, PORT`
 - `OUT PORT, REG`

Device I/O Port Locations on PCs (partial)

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)

Access Device Controller

- CPU read and write to the device controller registers and data buffer
 - (Logical) I/O ports
 - Memory mapped I/O

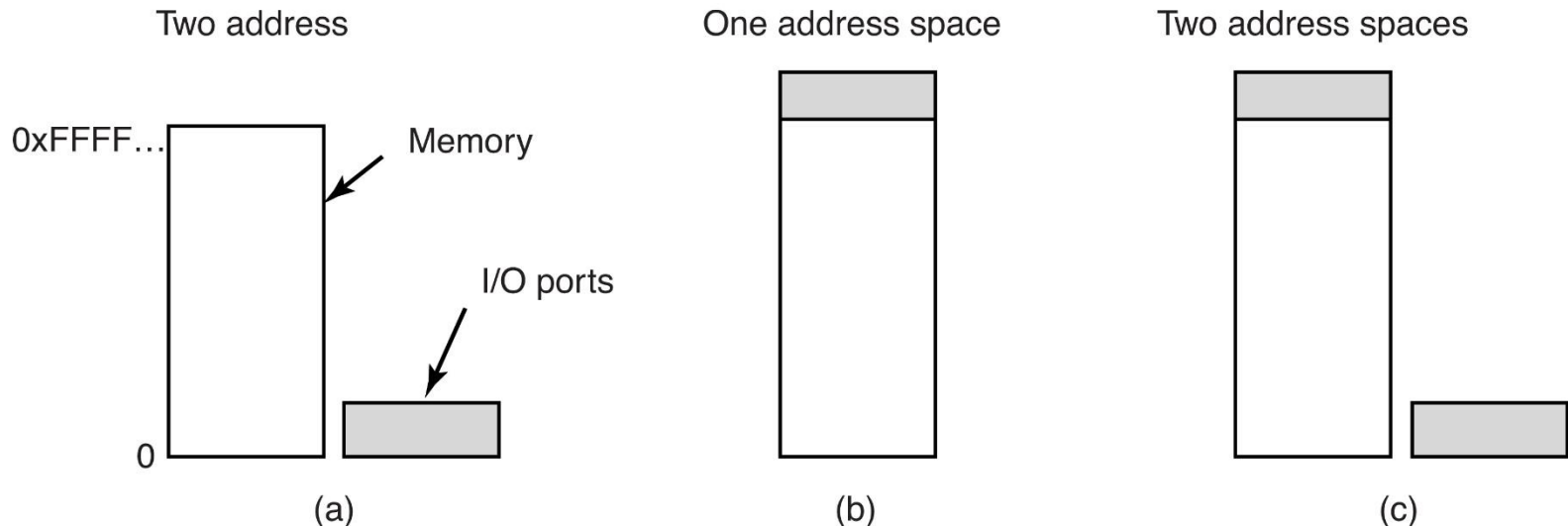
I/O Instruction: Example

- Example from
 - <http://www.tldp.org/HOWTO/text/IO-Port-Programming>
- Source
 - <https://github.com/CISC3320SP19/SamplePrograms/tree/master/C05bIO>

Memory Mapped I/O

- Map all the control registers into the memory address space
 - A register is assigned to a unique memory address to which no memory is assigned
 - Accessing these registers as if they were main memory
- Hybrid scheme
 - Data buffers are mapped to memory address
 - Control registers have dedicated I/O ports

Accessing Device Controllers



I/O Ports

Memory-Mapped

Hybrid

- Access controller registers [Figure 5-2 in Tanenbaum & Bos, 2014]

Strength and Weakness

- Strength of memory mapped I/O
 - Easier to program
 - Easier to protect
 - Faster to access
- Weakness (two addresses logically identical, but physically different)
 - More complex to design cache
 - More complex to design bus

Questions?

- Access devices controller registers
 - I/O ports
 - Memory-mapped I/O
 - Hybrid

I/O Schemes

- Busy waiting (polling)
 - while (busy) wait; do I/O;
- Interrupted I/O
 - do something else; when (interrupted) do I/O;
- Direct memory access (DMA)
 - initialize DMA; do something else; notified I/O completion or failure when interrupted;

Busy-Waiting (or Polling)

- Illustrate it with writing a byte

- Host

1. do
2. read the busy-bit in the device status register
3. while (busy)
4. set the write-bit in the control register
5. write a byte into the data-out register
6. set the command-ready bit in the control register

- Device Controller

1. do
2. read the command-ready bit
3. while (not set)
4. set the busy bit
5. read the byte in the data-out register
6. write the byte to the device
7. if (success) clear the command-ready bit and the busy bit
8. else set the error bit



Busy Waiting Occupies CPU

- Host:
 1. do
 2. [1] read the busy-bit in the device status register
 3. [3] while ([2] busy)
 4. ...
- Steps 1 - 3 are a busy-wait cycle to wait for I/O from device via polling the device controller
- Polling can happen in 3 instruction cycles
 - Read status, logical-and to extract status bit, branch if not zero

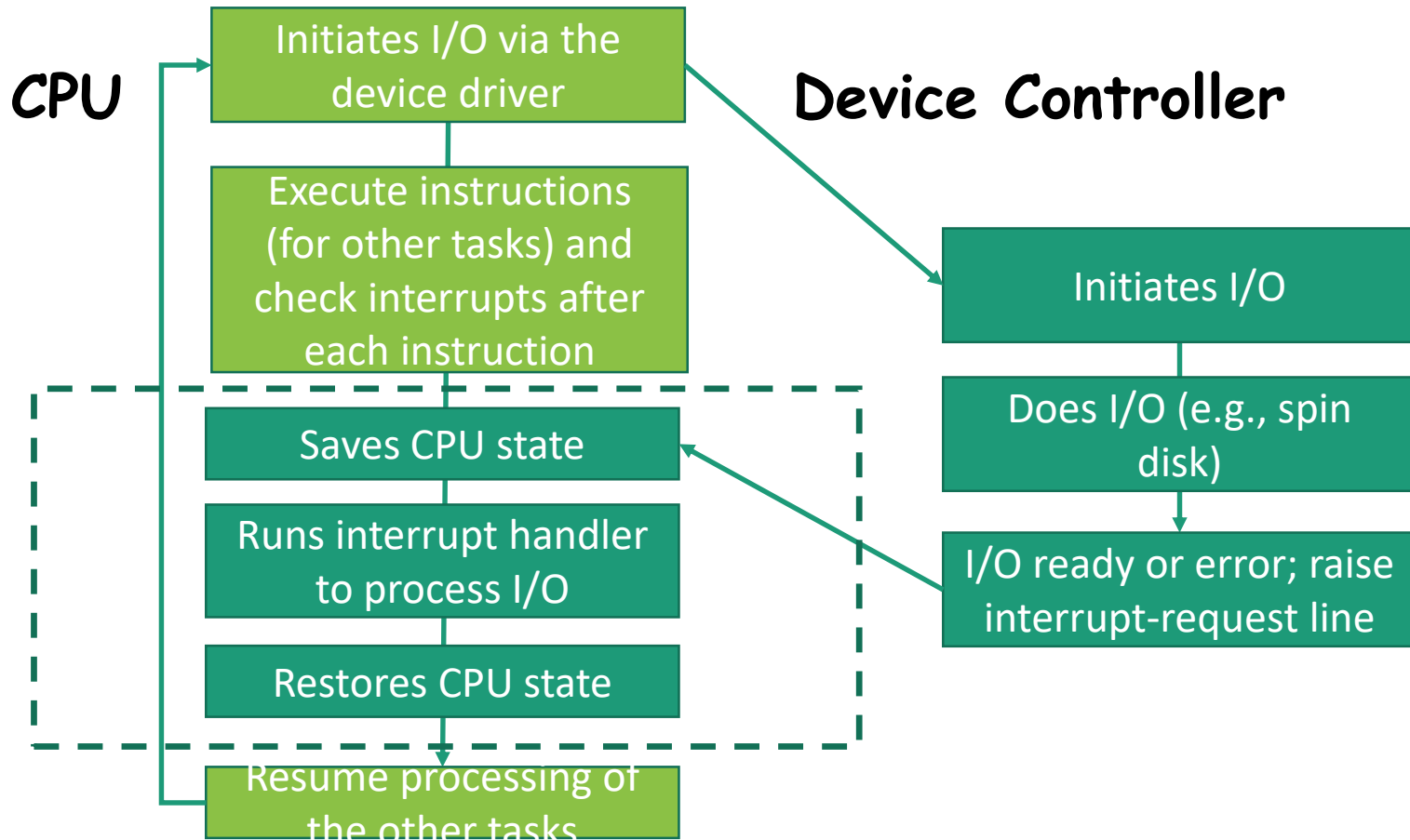
Busy-Waiting: Discussion

- Host:
 1. do
 2. read the busy-bit in the device status register
 3. while (busy)
 4. ...
- Steps 1 - 3 are a busy-wait cycle to wait for I/O from device
- Reasonable if device is fast (e.g., 1 or 2 cycles).
- But inefficient if device slow (e.g., many cycles).
- CPU switches to other tasks?
- But if miss a cycle data overwritten / lost.

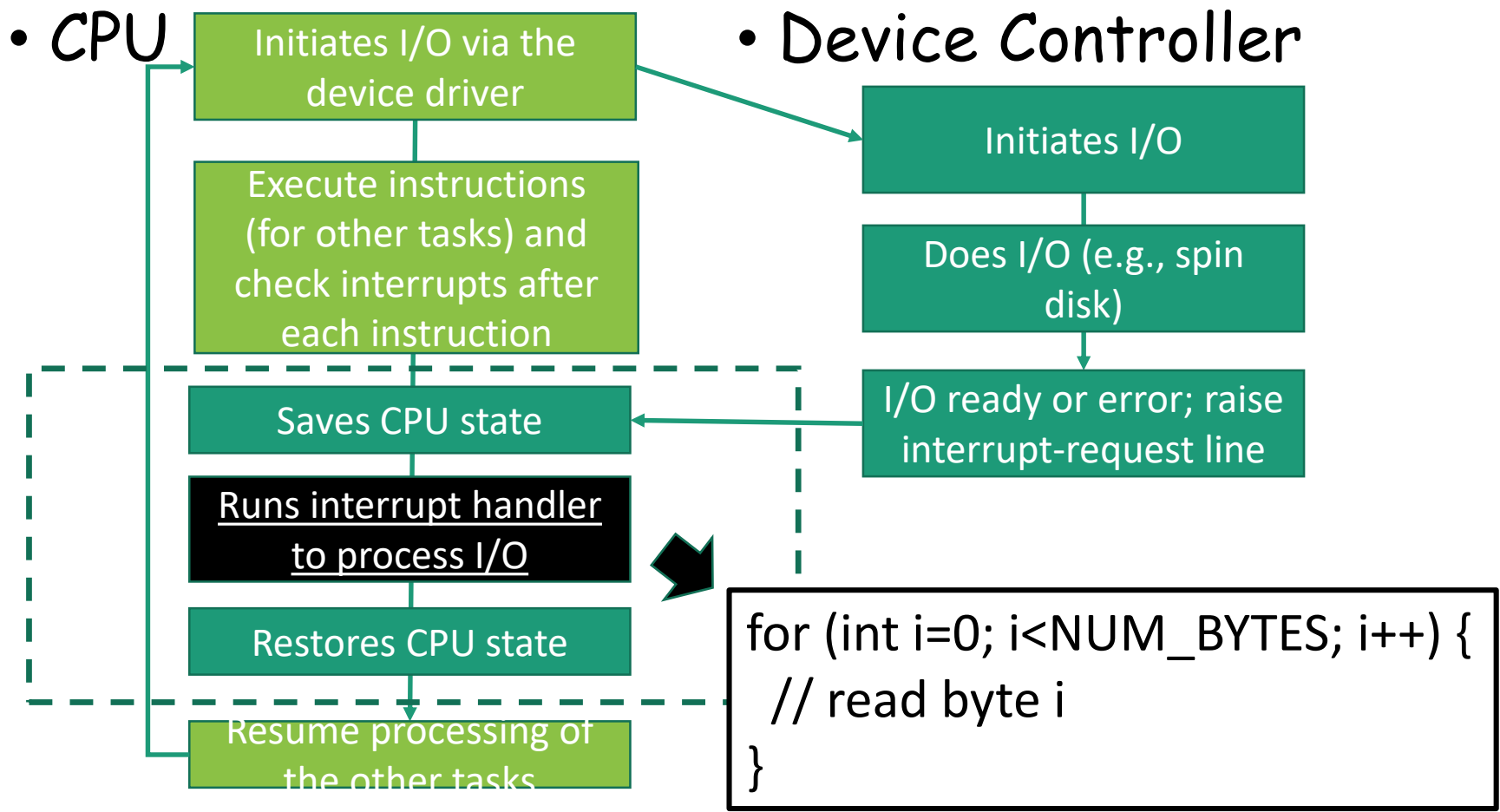
Interrupt-Driven I/O

- Host:
 1. do
 2. [1] read the busy-bit in the device status register
 3. [3] while ([2] busy)
 4. ...
- Polling can happen in 3 instruction cycles
 - Read status, logical-and to extract status bit, branch if not zero
- How to be more efficient?
 - Design with interrupts
 - Design I/O device to trigger CPU Interrupt-request line, and
 - Program accordingly

Interrupt-Driven I/O Cycle



Processing I/O: What if Many Bytes to Read?

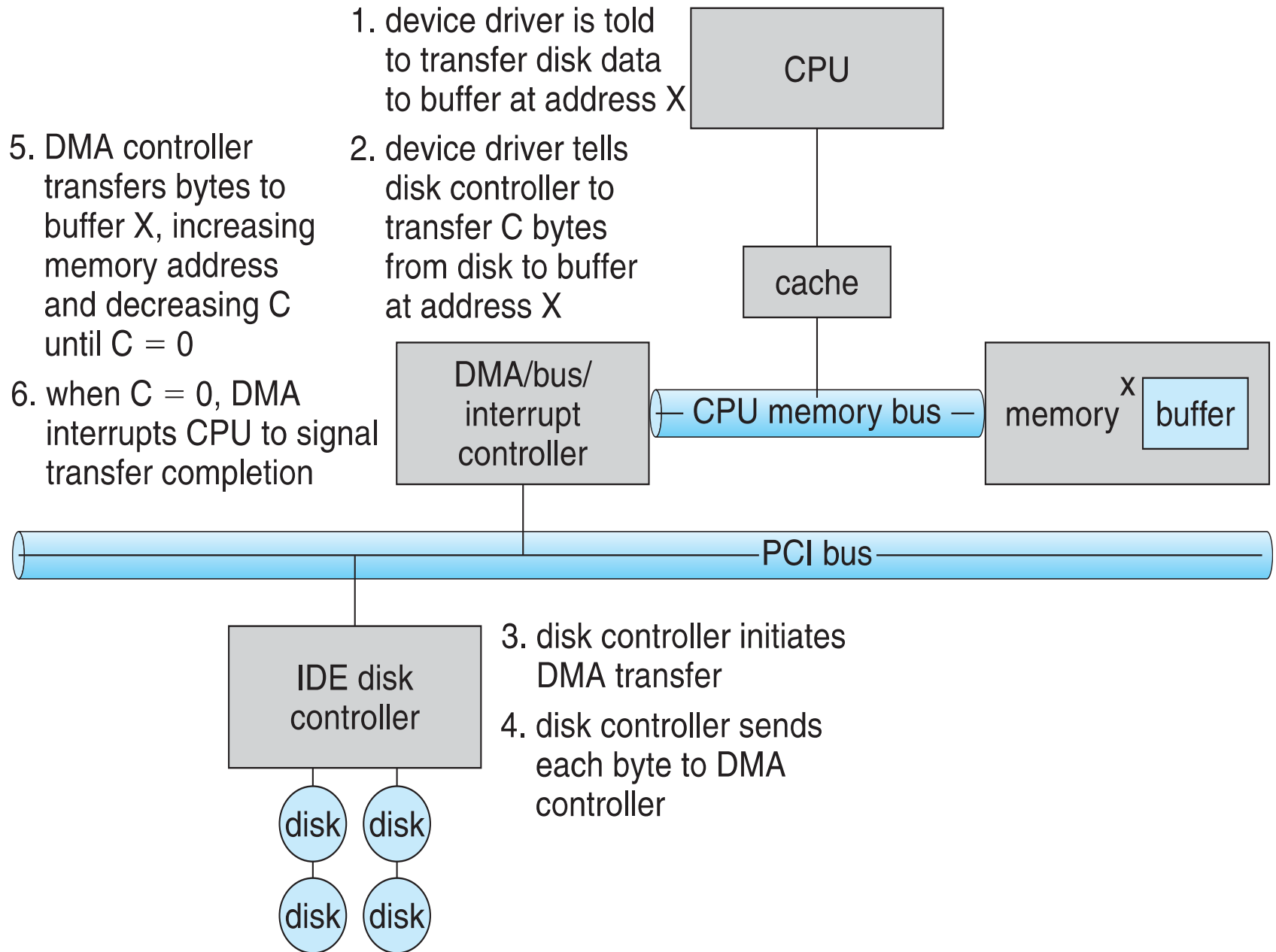


Direct Memory Access

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory
- Version that is aware of virtual addresses can be even more efficient
 - Direct Virtual Memory Access (DVMA)

Six Step Process to Perform DMA Transfer

- OS writes DMA command block into memory
 - Source and destination addresses
 - Read or write mode
 - Count of bytes
 - Writes location of command block to DMA controller
 - Bus mastering of DMA controller
 - grabs bus from CPU
 - Cycle stealing from CPU but still much more efficient
 - When done, interrupts to signal completion



1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X

5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller

Questions?

- I/O instructions
- I/O schemes
 - Busy-waiting (polling)
 - Interrupt-driven (interrupted) I/O
 - Direct memory access