CISC 7310X R6

Interrupts and I/O

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

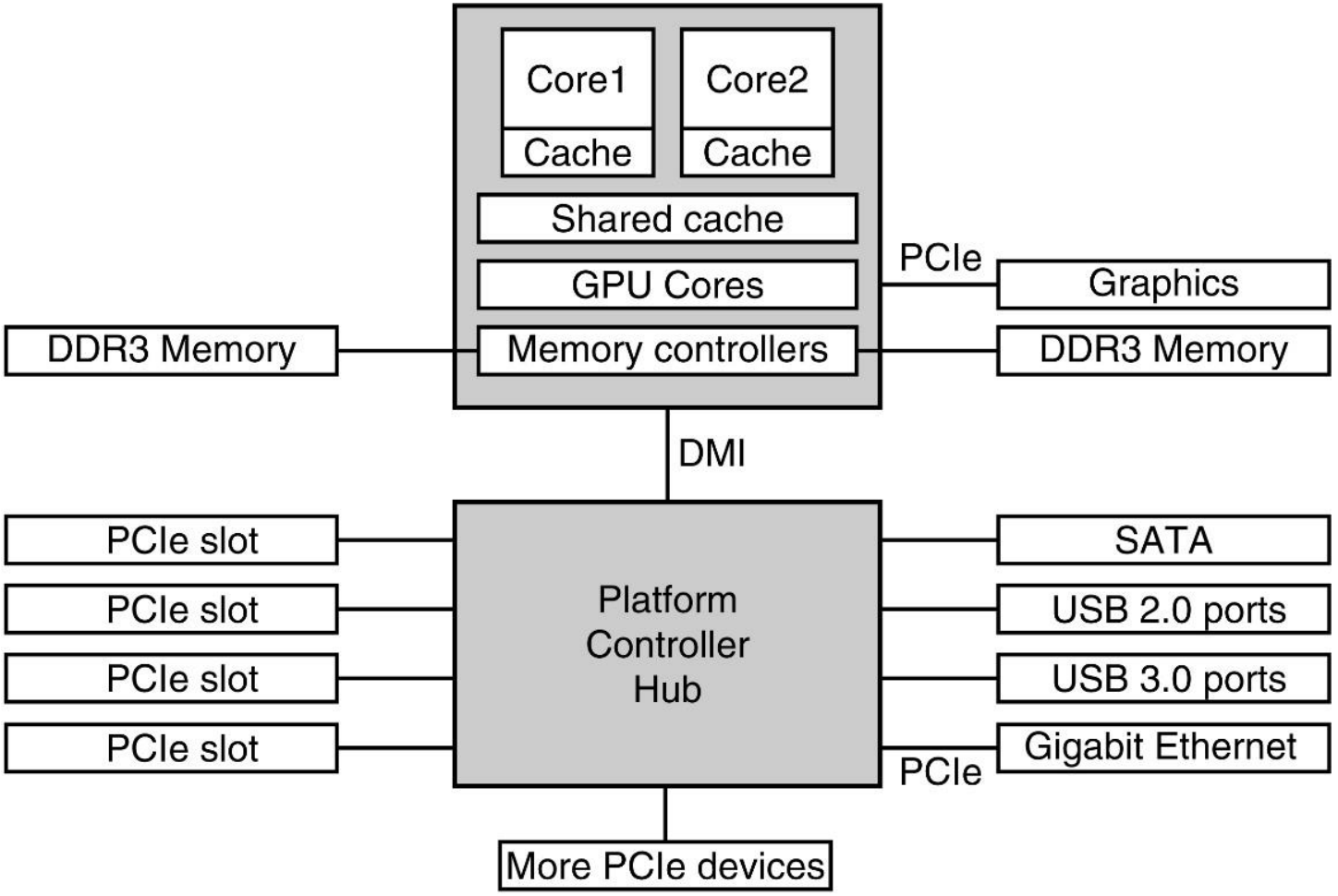
Outline

- **Overview of Computer Architecture**
- Overview of I/O devices
- Overview of Interrupts
- Overview of I/O schemes
- Overview of I/O software
- Experimenting with boot sector code for I/O and interrupts

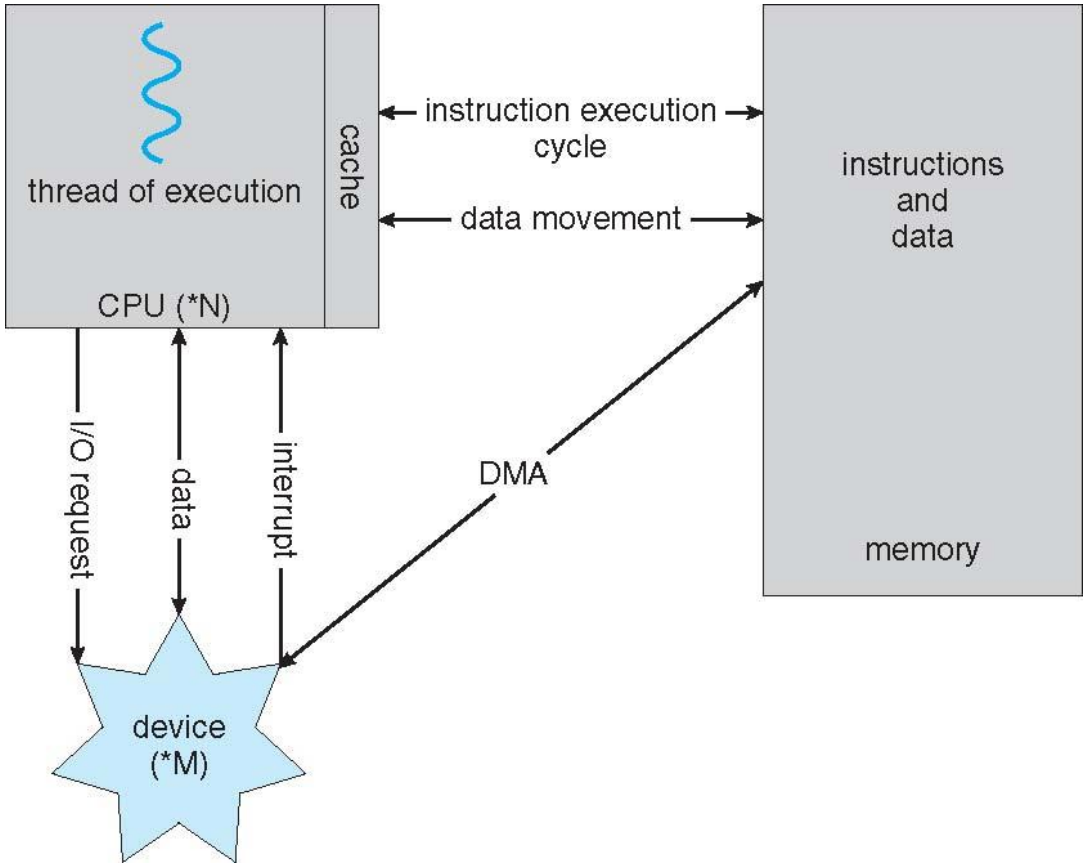
von Neumann Computers

- Process and memory connected by a bus (Von Neumann, 1945)
- Bus: A common set of wires with a protocol that specifies commands that can be transmitted

An x86 Realization



How does it work?



- How a modern computer system works [Figure 1.7 in Silberschatz et al., 2018]

Architecture, OS, and Programming

- Architecture underpins design of OS and programming
  - Observing how we wrote our boot sector code
  - How about the future?

"I propose to call this tube the von Neumann bottleneck. The task of a program is to change the contents of the store in some major way; when one considers that this task must be accomplished entirely by pumping single words back and forth through the von Neumann bottleneck, the reason for its name becomes clear."

– John Backus, 1977

Outline

- Overview of Computer Architecture
- **Overview of I/O devices**
- Overview of Interrupts
- Overview of I/O schemes
- Overview of I/O software
- Experimenting with boot sector code for I/O and interrupts

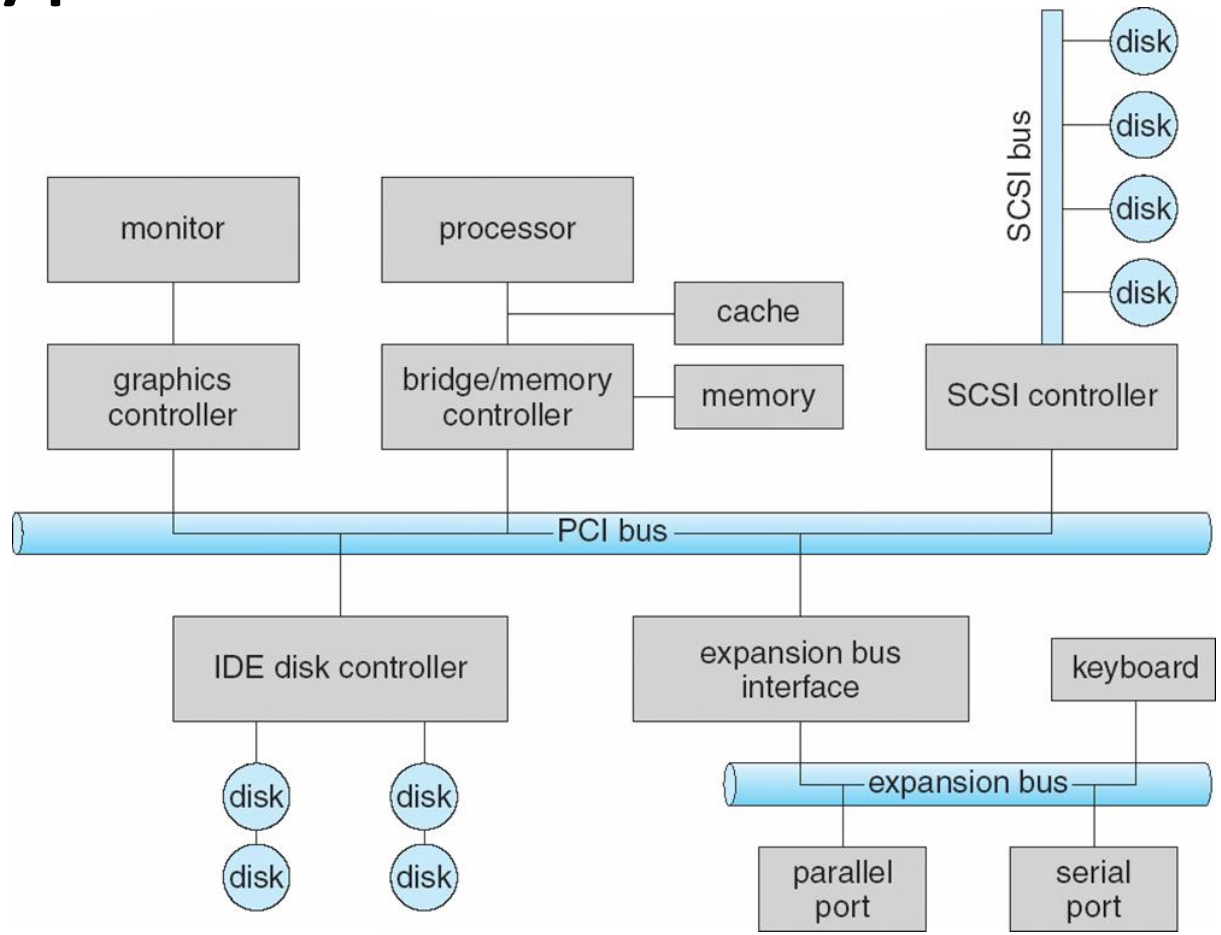
I/O Devices

- 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: control cars, robots, aircrafts, spacecrafts

Port and I/O Bus

- Devices communicate with a computer via a connection point
  - (Physical) port
    - Examples: USB port, serial port, parallel port
    - Not to confuse with (logical/data) ports
  - I/O Bus (or Expansion Bus)
    - Examples: PCI bus, SCSI bus

A Typical PC Bus Structure



[Figure 12-1 in Silberschatz et al., 2018]

Device Controller

- Devices
  - Example: hard disk drives have motors, magnetic headers, and disks
- Controller
  - 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

Design Consideration: Access Right

- A design consideration
  - What kind of access right should we give to device drivers?
  - Unrestricted
    - Kernel mode
    - Relatively easier to design, can affect the others
  - Restricted
    - User mode
    - More difficult to design, isolated from the others

Design Consideration: Load Device Drivers

- Relink the kernel with the new deriver
  - Require reboot
- Add to the kernel an entry indicating a new driver is needed
  - Load the driver during reboot
- Install and run the device driver on the fly
  - Hot-pluggable

Device Controller Registers

- Typically have 4 registers or more
  - 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
- May have a data buffer
  - Examples: video adapter (video memory)

Access Device Controller

- CPU read and write to the device controller registers and data buffer
  - (Logical) I/O ports
  - Memory mapped I/O
- See the tutorial for experimenting using bootsector code

I/O Port Space and Port-Mapped I/O

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

Example I/O Port Allocation

- Some default values on PCs

I/O Address Range	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)

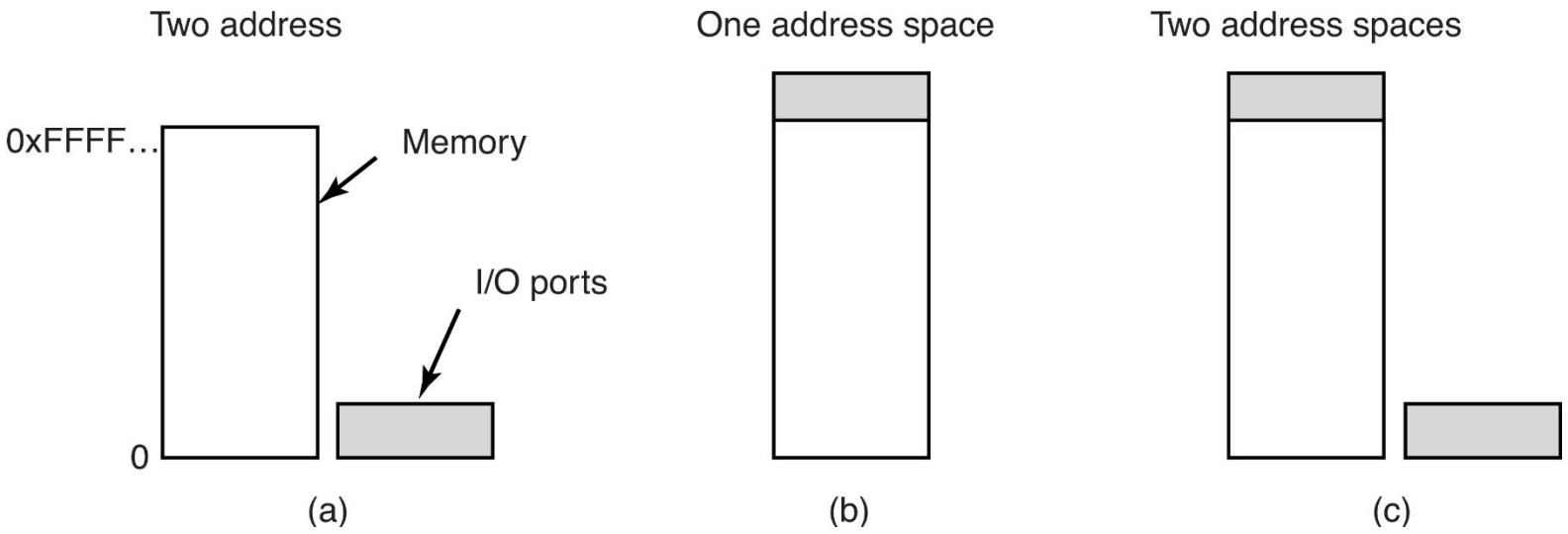
Memory-Mapped I/O

- 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

- Memory-mapped I/O for data buffer
  - Data buffers are mapped to memory address
- Port-mapped I/O for control registers
  - 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

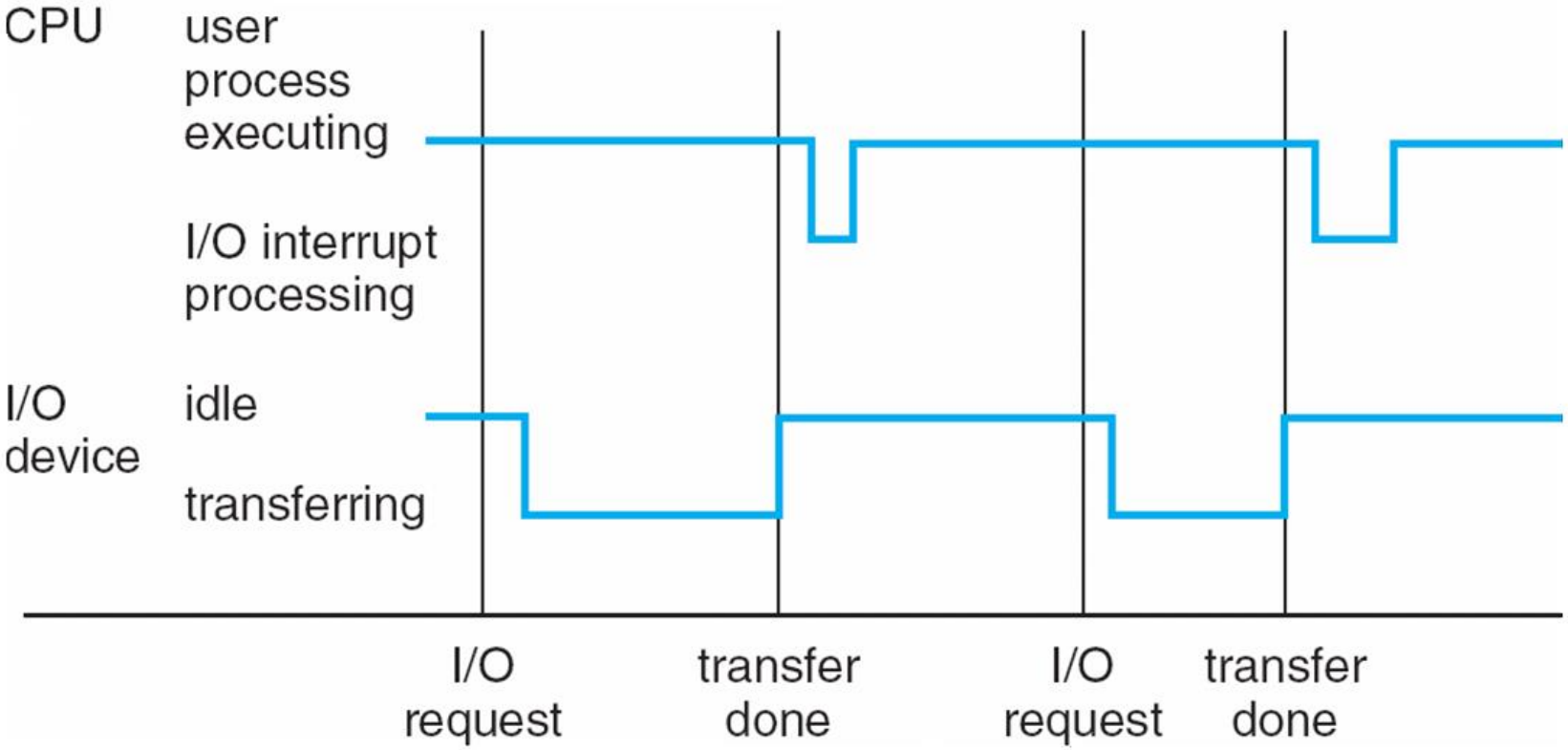
Outline

- Overview of Computer Architecture
- Overview of I/O devices
- **Overview of Interrupts**
- Overview of I/O schemes
- Overview of I/O software
- Experimenting with boot sector code for I/O and interrupts

Interrupts

- Interrupt transfers control to the interrupt service routine generally
- Two sources of interrupts
  - External (hardware-generated) interrupts: interrupts are generally caused by hardware
  - Software generated interrupts: a trap or exception is a software-generated interrupt caused either by an error or a user request
- Interrupt vector (interrupt descriptor by Intel)
  - Interrupt service routine: interrupt handler, a program processes the interrupt
  - Interrupt vector table: consists of interrupt vectors
  - Interrupt vector: the address of an interrupt handler
- Interrupt architecture must save the address of the interrupted instruction

Interrupt Timeline: I/O Interrupts



Interrupt Vectors

- Address to interrupt routines
  - Some PC event/interrupt-vector numbering

Vector Number	Description
0	Divide Error
1	Debug Exception
…	
6	Invalid Opcode
…	
32-255	Maskable Interrupts (deviced generated)

Handling Interrupt

- CPU senses its interrupt-request line after each instruction
- When it is “lit”, CPU saves the current state
  - Example: push registers PSW and PC to the stack
- CPU jumps to the interrupt-handler routine at a fixed address in the memory
- Interrupt-handler routine completes its task

and restore the CPU state

  - Pop the registers from the stack

Design Consideration: Interrupts

- Maskable and nonmaskable interrupts
- Interrupt priorities and interrupt chaining
- Exceptions and software interrupts (traps)
- Precise and imprecise interrupts

Exceptions and Interrupts

- Interrupt mechanism also used for exceptions
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
  - System call executes via trap to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

Outline

- Overview of Computer Architecture
- Overview of I/O devices
- Overview of Interrupts
- **Overview of I/O schemes**
- Overview of I/O software
- Experimenting with boot sector code for I/O and interrupts

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; I/O done when interrupted;

Implementing Busy Waiting

- Illustrate it with writing a byte
  - Host

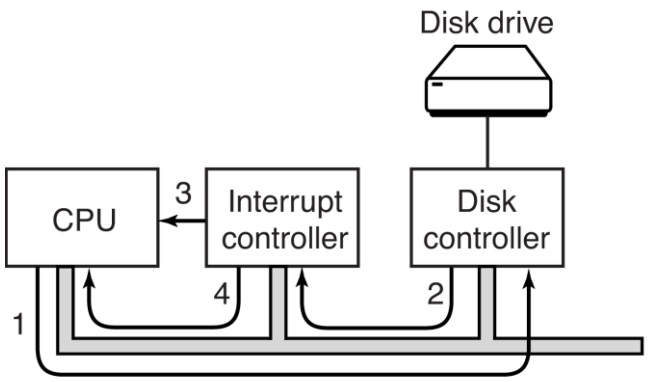
1. do
2. read the busy-bit in the device status register
3. while (busy)
4. set the write-bit in the control register
5. write a byte into the data-out register
6. set the command-ready bit in the control register

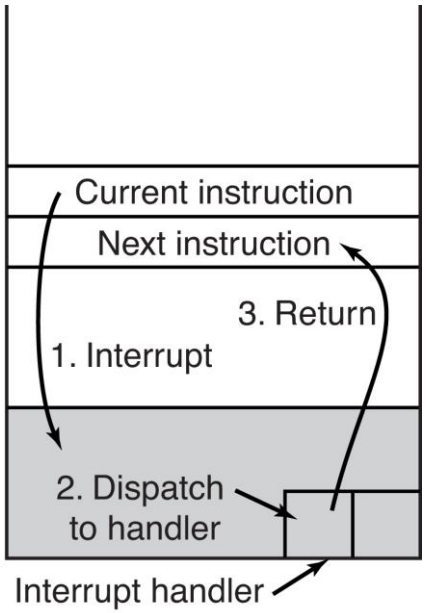
  - Device Controller

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

Interrupted I/O

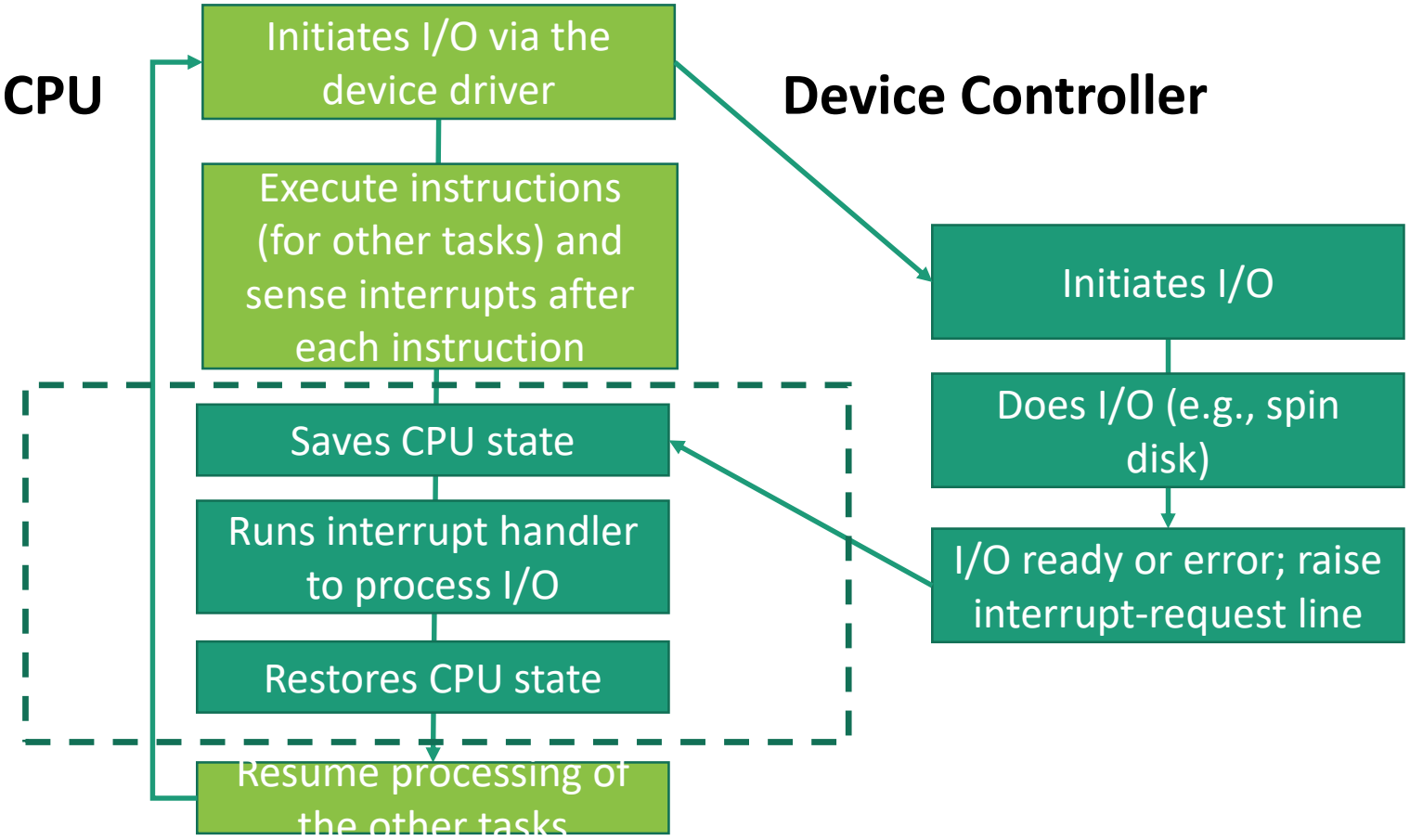
- Conduct I/O in an asynchronous fashion





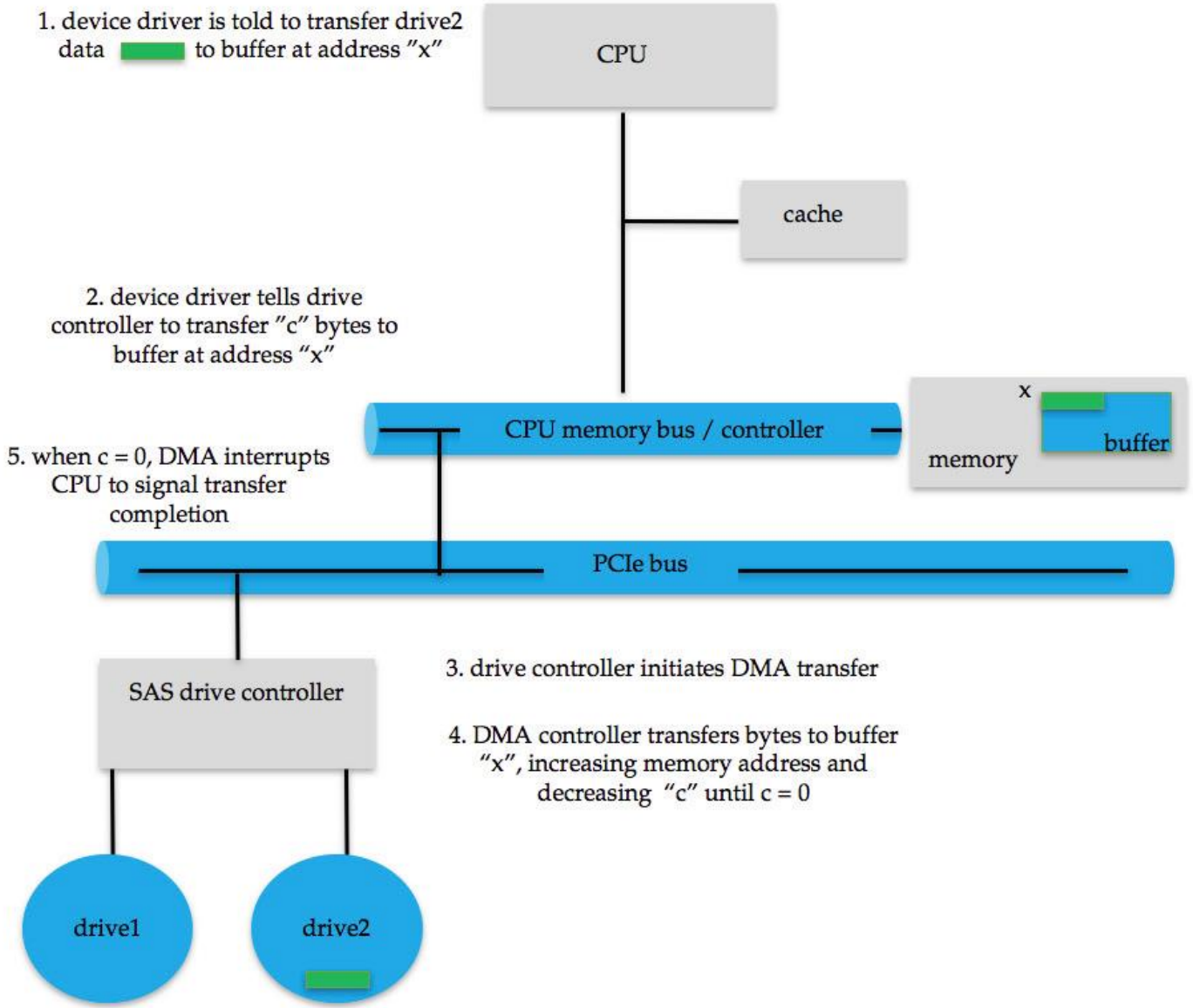
- Interrupted I/O[Figure 1-11 in Tanenbaum & Bos, 2014]

Interrupted I/O Cycle



Direct Memory Access

- Aided by a special purpose processor called direct-memory-access (DMA) controller
  - CPU writes a DMA command block into memory
    - Pointer to the source of transfer
    - Pointer to the destination of transfer
    - A count of the number of bytes to be transferred
  - CPU writes the address of this block to the DMA controller
  - The DMA controller does I/O by directly access devices and system bus
  - CPU is interrupted when the DMA controller completes the transfer or encounters an error

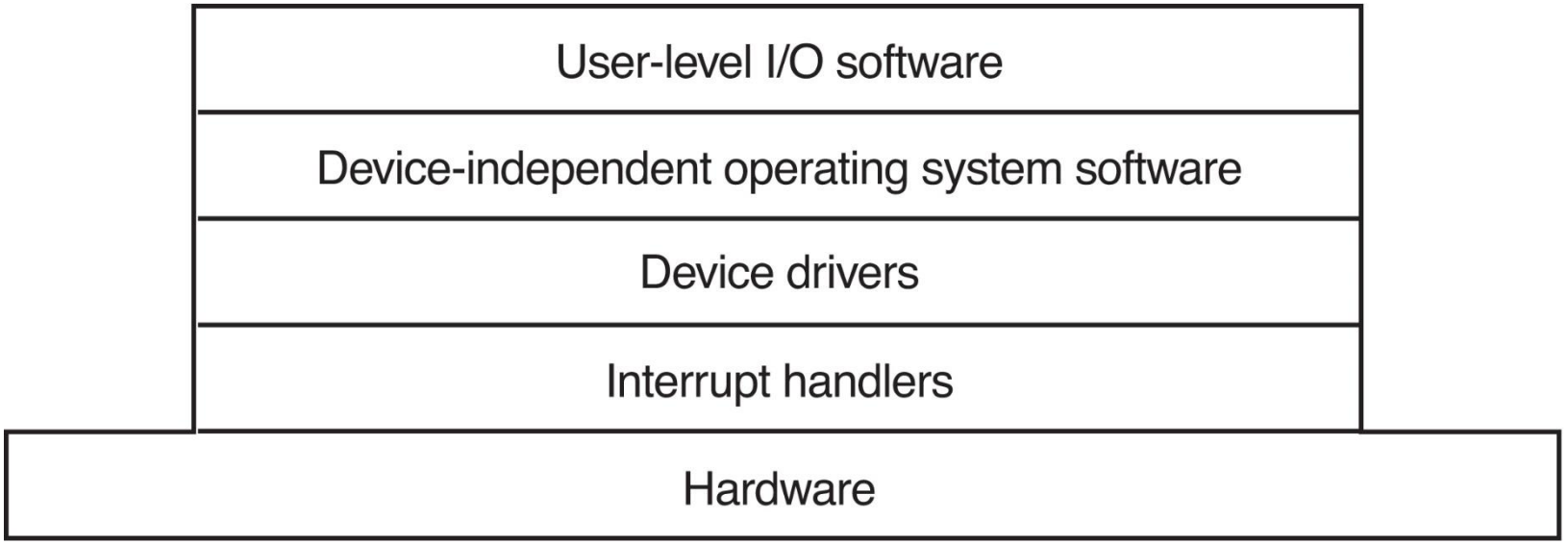


[Figure 12.6 in Silberschatz et al., 2018]

Outline

- Overview of Computer Architecture
- Overview of I/O devices
- Overview of Interrupts
- Overview of I/O schemes
- **Overview of I/O software**
- Experimenting with boot sector code for I/O and interrupts

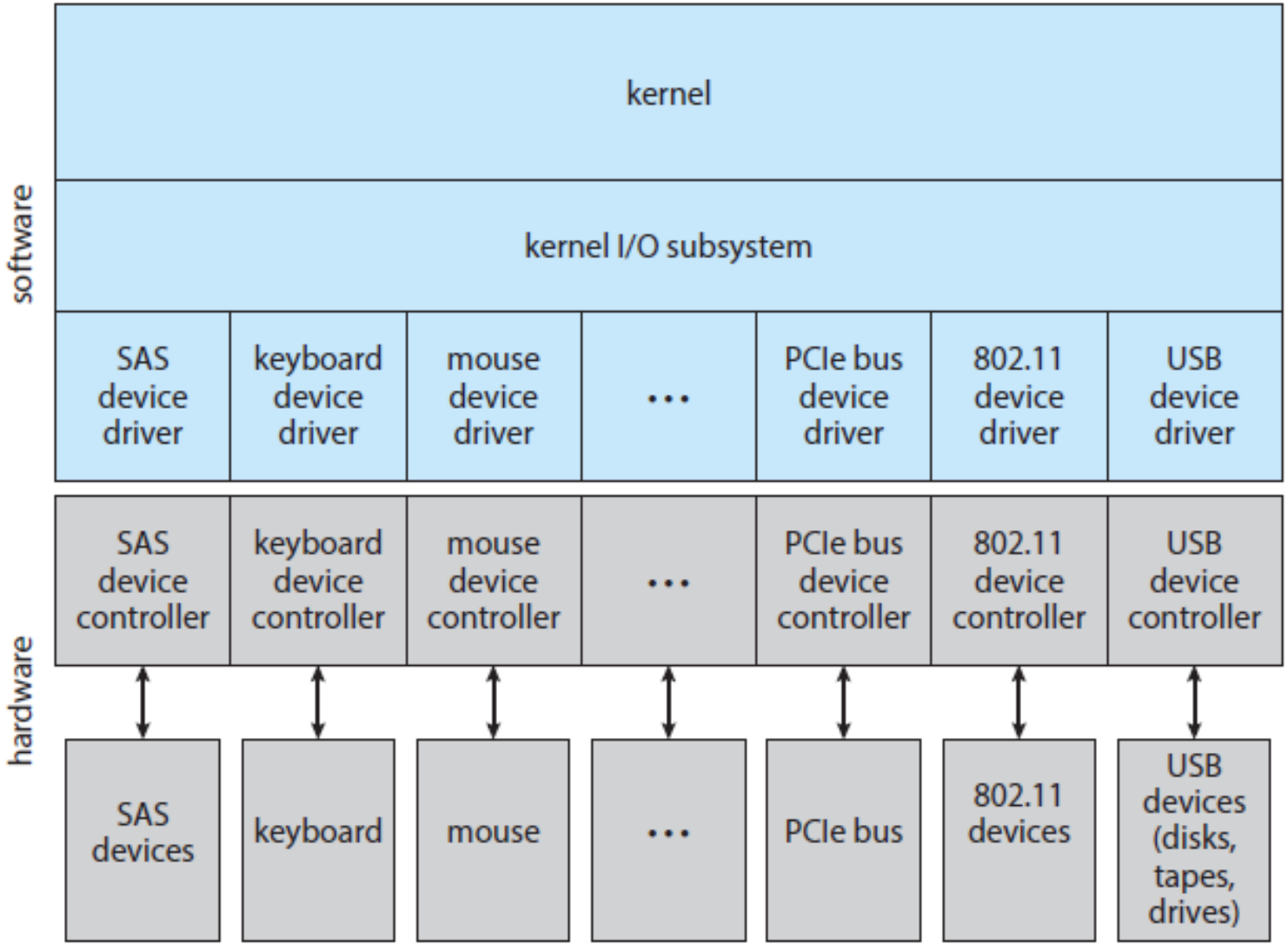
I/O Software Layers



- [Figure 5-11 in Tanenbaum & Bos, 2014]

Device Driver

- Each type of controller is different
- Software communicates to the controller, and the OS
- Adhere to some standard when communicating to the OS
- Reduce complexity, increase uniformity and reliability



- [Figure 12.7 in Silberschatz et al., 2018]

Device Types and Device Independency

- To achieve device independent, categorize devices based on general characteristics.
- A couple of dimensions
  - Size of transfer: Character-stream or block
  - Access order: sequential or random access
  - Predictability and responsiveness: Synchronous and asynchronous
  - Shared or dedicated
  - Speed of operation, e.g., latency, seek time, transfer rate
  - Read-write, read only, or write only

Block Devices

- Naming
  - Examples on Linux
    - by label, by uuid, by id, and by path
    - Running examples
      - lsblk –f
      - ls /dev/disk/
- Read and write a block a time
- Essential behavior
  - read(), write()
  - For random-access block devices
    - seek()

Character Devices

- Read and write a character a time
- Essential behavior
  - get(), put()

Outline

- Overview of Computer Architecture
- Overview of I/O devices
- Overview of Interrupts
- Overview of I/O schemes
- Overview of I/O software
- **Experimenting with boot sector code for I/O and interrupts**