Bilkent University
Department of Computer Engineering
CS342 Operating Systems

Input/Output (I/O) Sub-System

Last Update: May 31, 2023

Objectives and Outline

Objectives

- Explore the structure of an operating systems I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

Outline

- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Performance

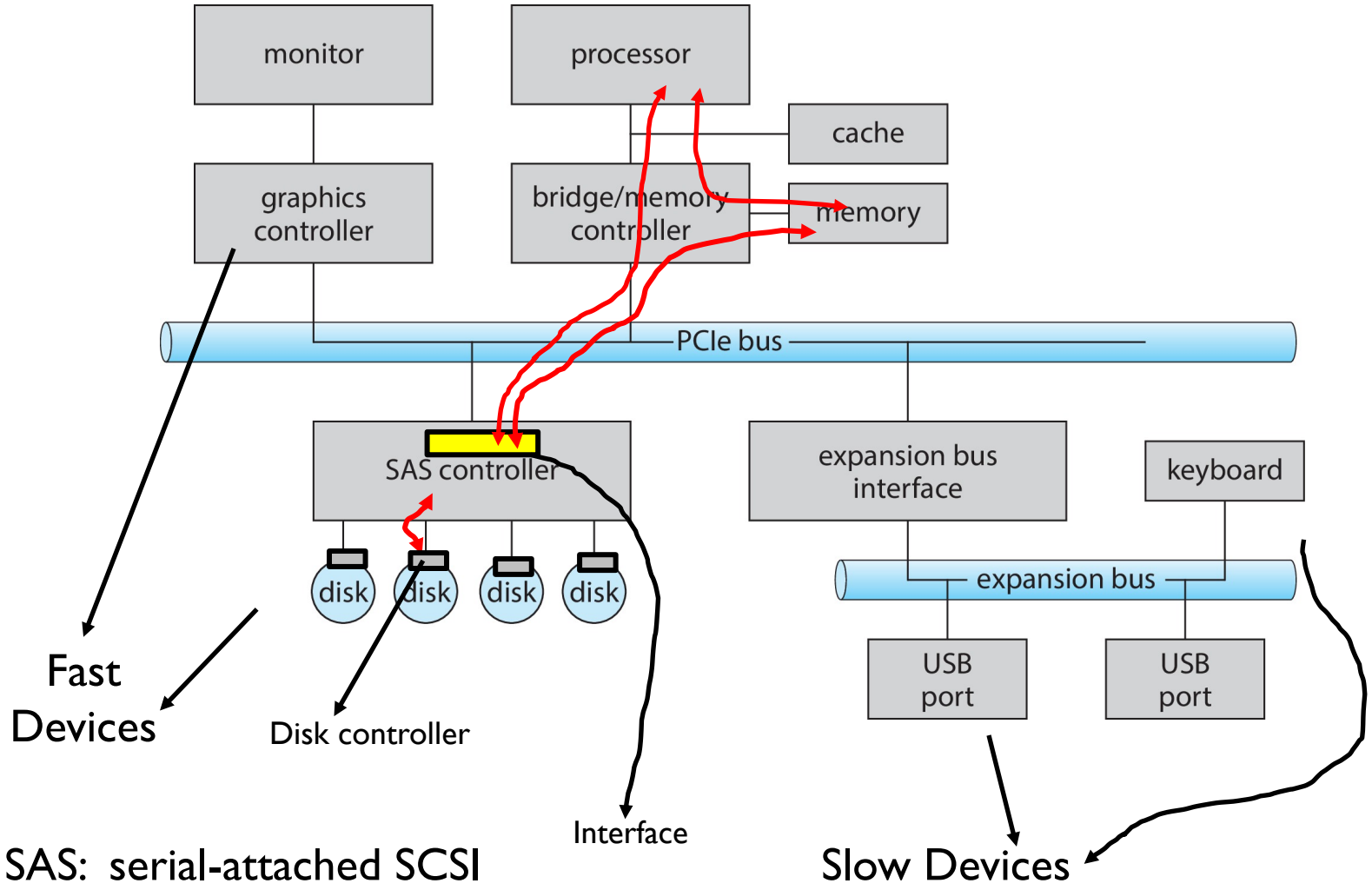
Input/Output

- A computer has two main functionality
  - Doing computation
  - Doing I/O (Input/Output)
- I/O:
  - Storage I/O: disk I/O
  - User Interaction: keyboard, mouse, screen I/O
  - Communication
  - ...
- OS: Manages and controls I/O devices. Performs I/O operations.

I/O Hardware

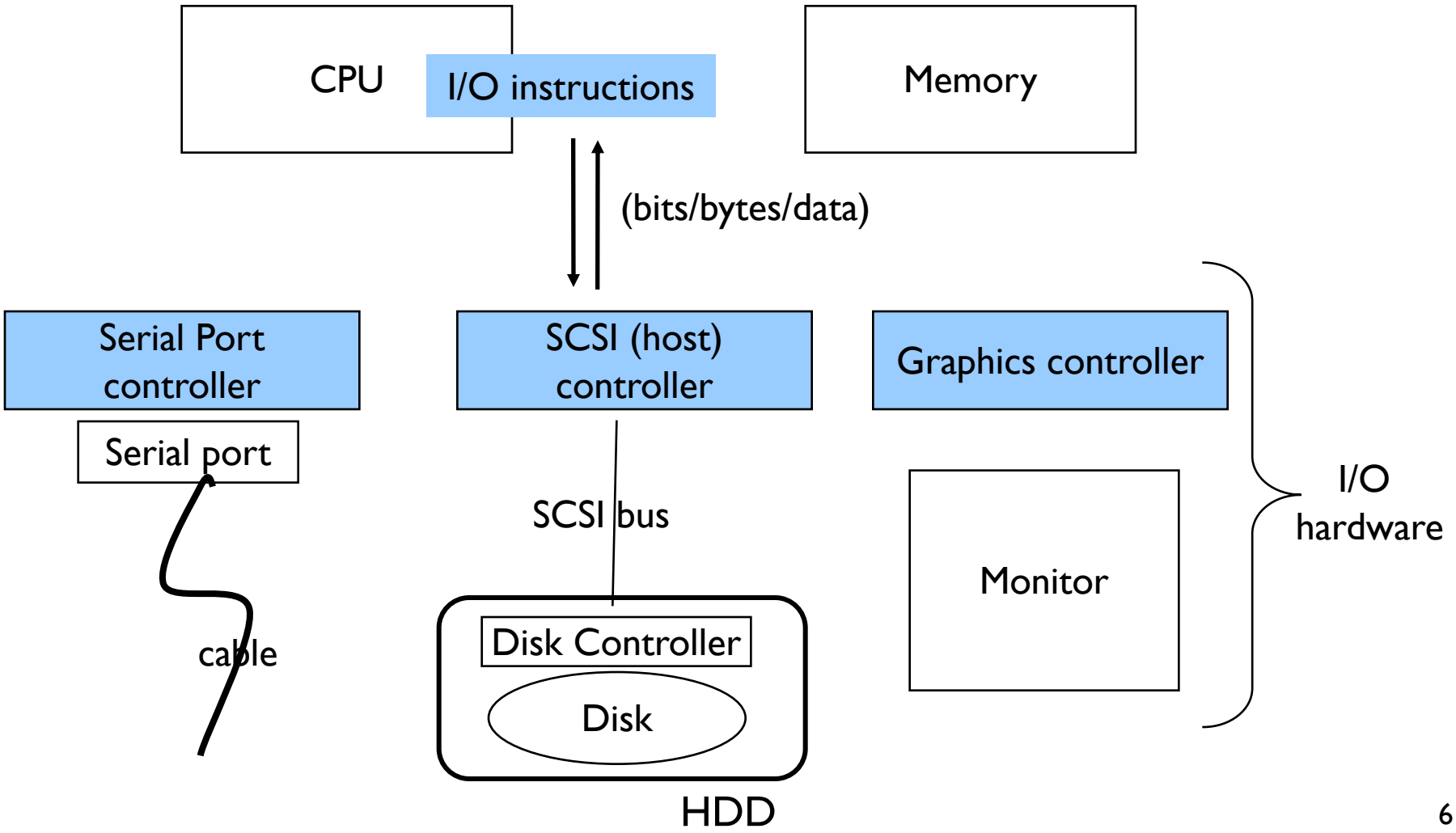
- Incredible variety of I/O devices
- But there are common concepts:
  - Port: a connection point, or a register set on the device controller and their addresses through which CPU accesses the device.
  - Bus: medium over which signals are sent/received – data transferred. For example: PCIe (system bus)
  - Device controller (host adapter): chip that can be accessed by CPU and that controls the device/physical-port/bus (PCI controller, hard disk controller, keyboard controller, …)
- I/O instructions used to control and interact with devices.

A Typical PC Bus Structure



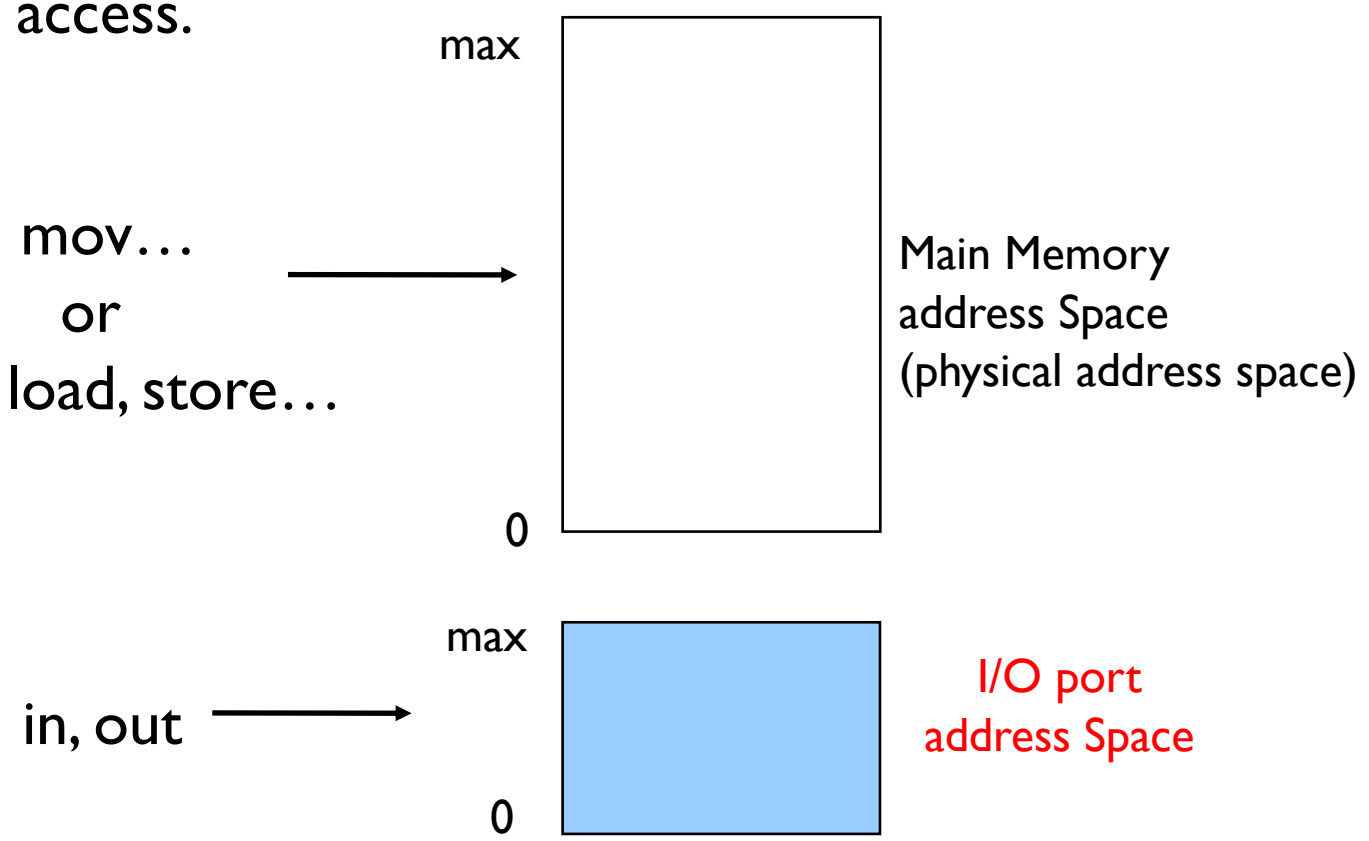
SAS: serial-attached SCSI

I/O hardware concepts

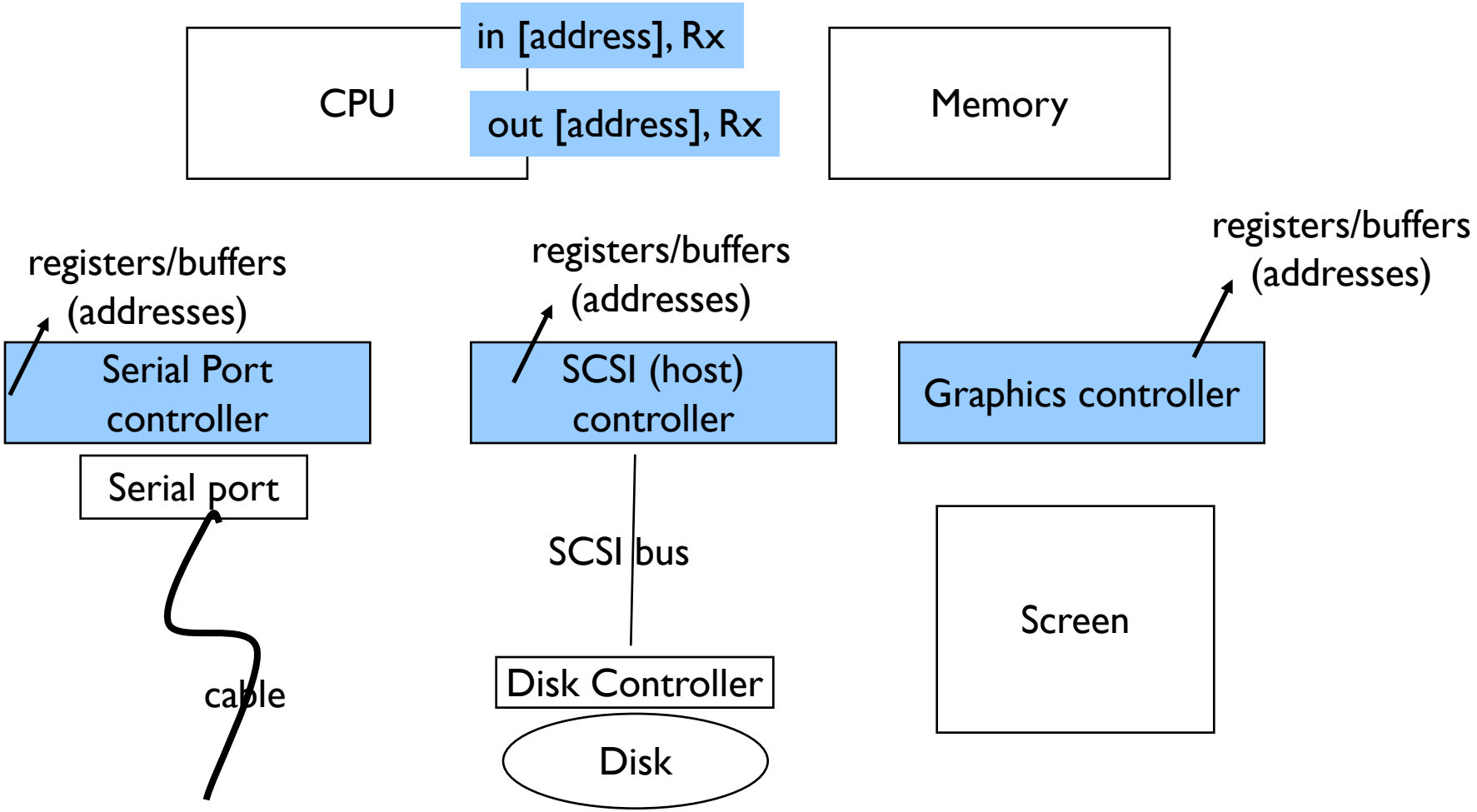


I/O hardware concepts: direct I/O (port-mapped I/O)

Device controller registers are addressed with a separate address space: I/O port address space. Special machine instructions are used to access.

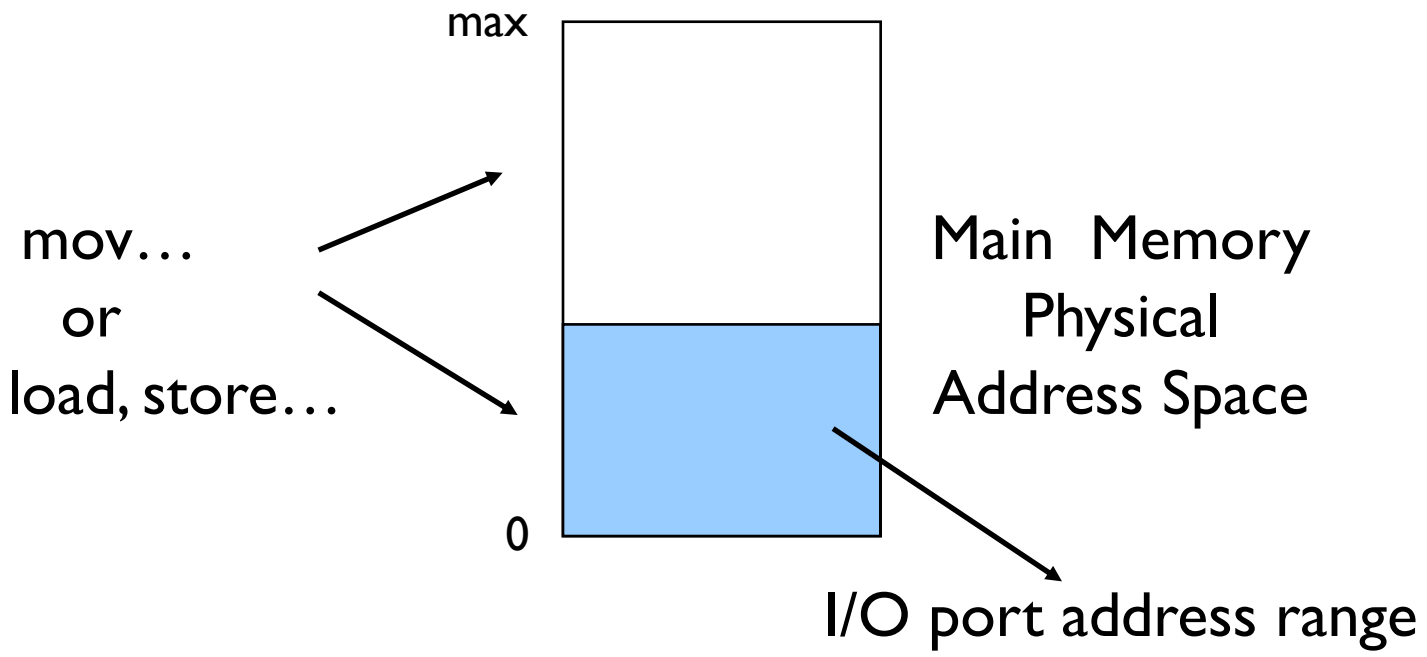


I/O hardware concepts: direct I/O (port-mapped I/O)



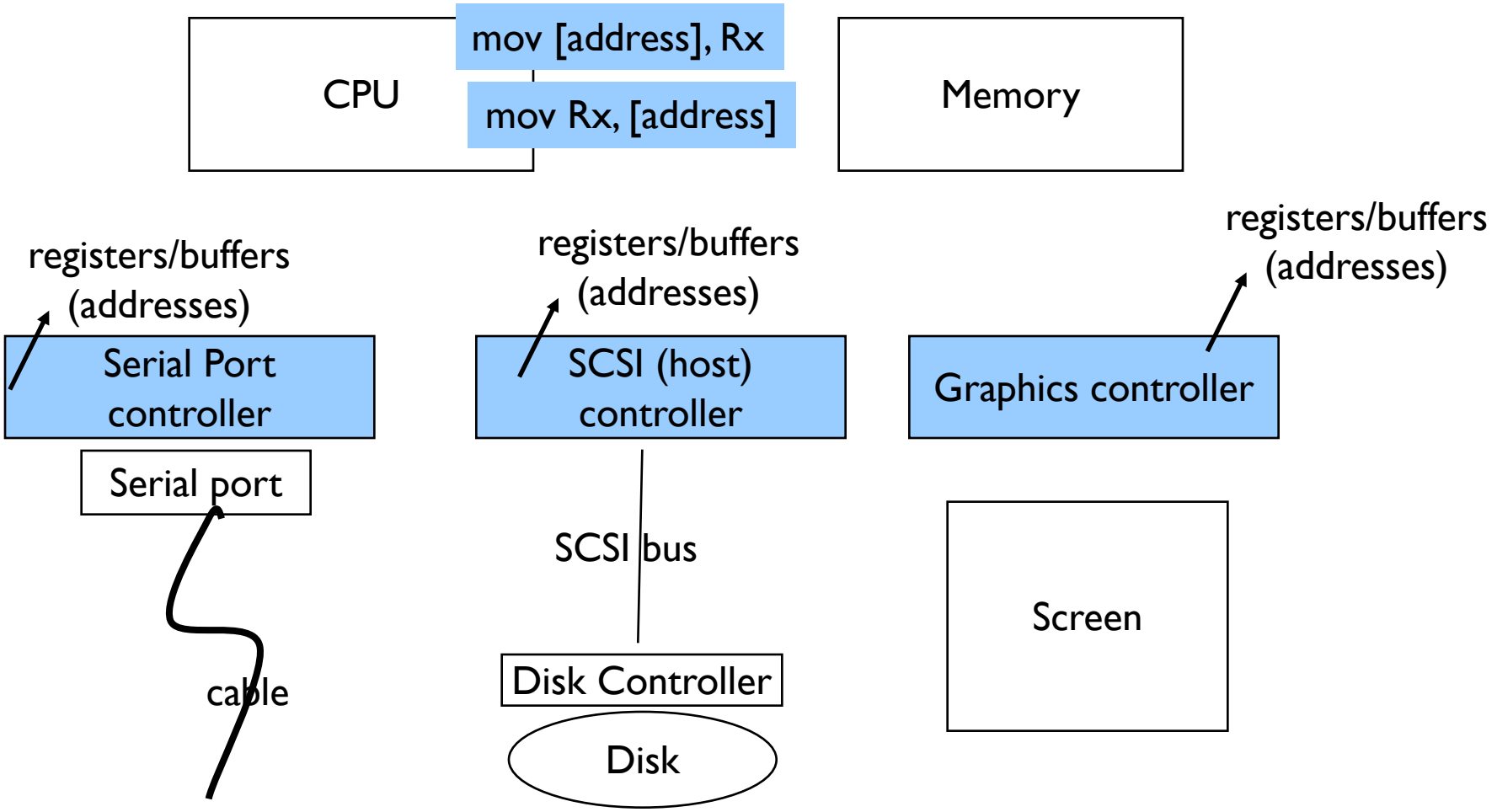
I/O hardware concepts: memory mapped I/O

Device controller is accessed with main memory addresses (device controller registers are mapped to a region of physical memory address space).



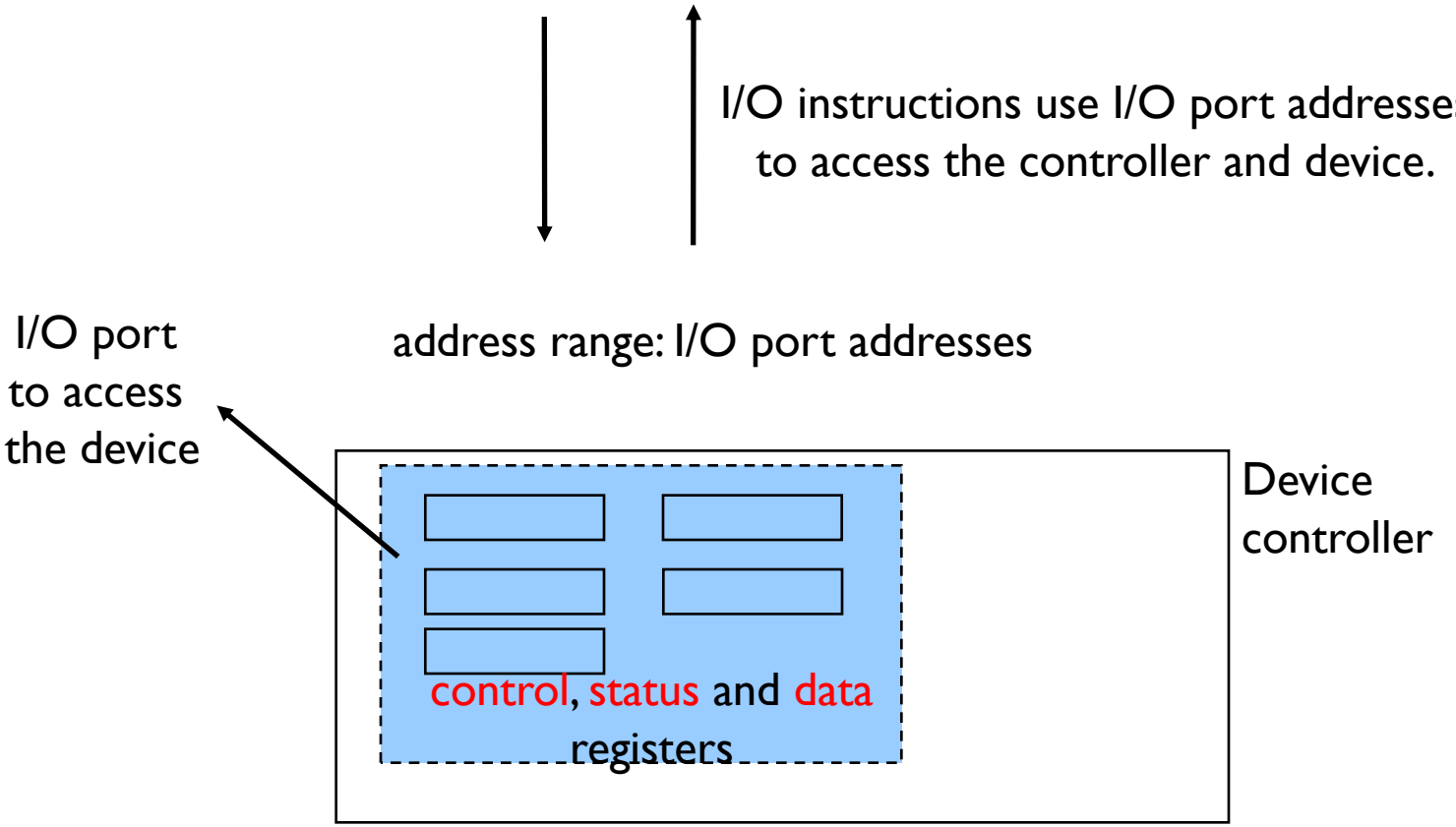
Memmory mapped I/O is more common now, in modern systems.

I/O hardware concepts: memory mapped I/O

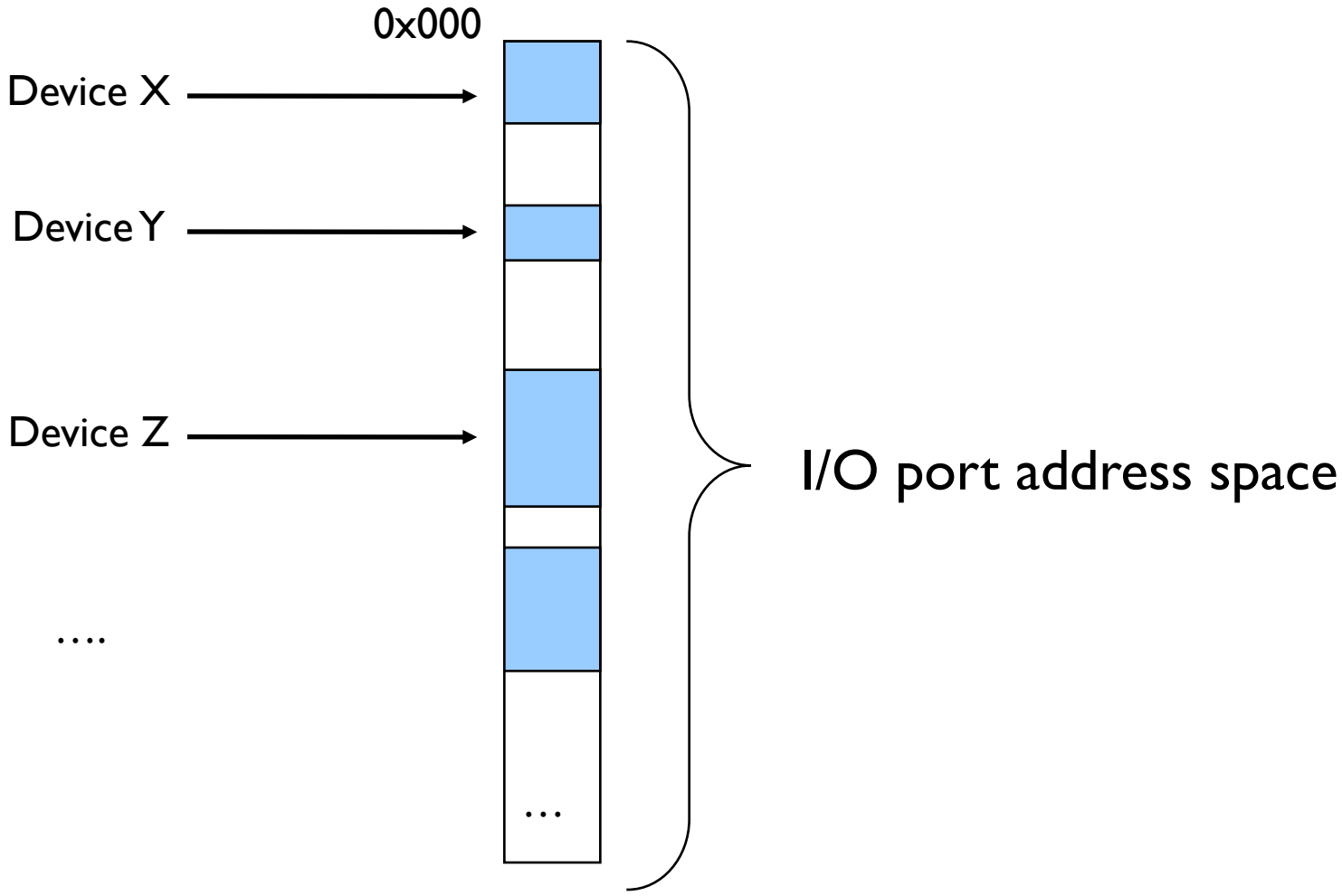


I/O port concept

I/O port for a device: set of controller registers and their addresses.
CPU can use them to access/control the device.



I/O port addresses



Device I/O Port Locations on PCs (partial)

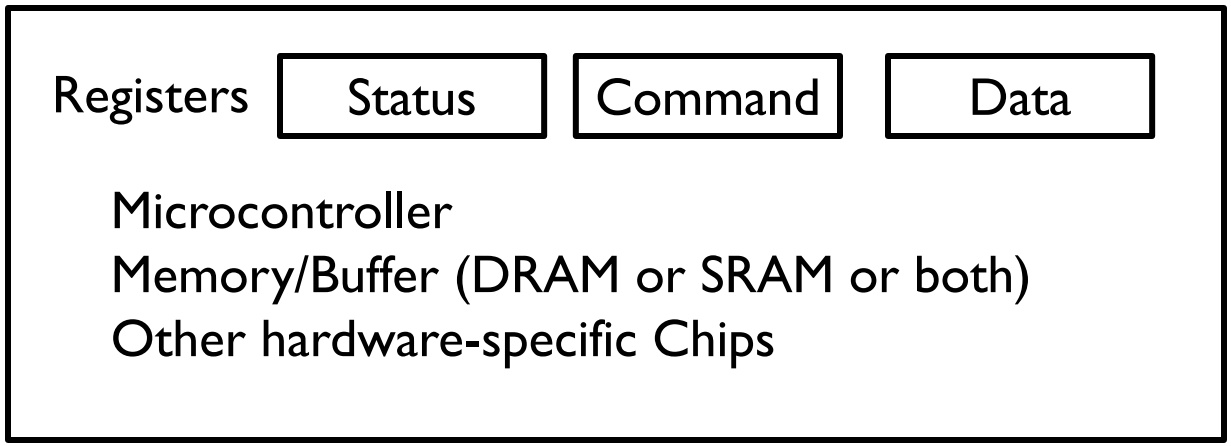
I/O address range	Device
00 – 1F	First DMA Controller, 8237 A-5
20 – 3F	First Programmable Interrupt Controller, 8259A
40 – 5F	Programmable Interval Timer (PIT) (System Timer), 8254
60 – 6F	Keyboard, 8042
70 – 7F	Real Time Clock (RTC)
170 – 177	Secondary Parallel ATA Disk Controller
1F0 – 1F7	Primary Parallel ATA Hard Disk Controller
200 – 20F	Game Port
220 – 233	Sound Card
2F8 – 2FF	Serial Port 2
300 – 31F	Ethernet network interface
320 – 323	ST-506 and compatible Hard Disk Drive interface
340 – 35F	Primary SCSI host adapter
378 – 37F	Parallel Port
3D0 – 3DF	Color Graphics Adaptor (CGA)
3F0 – 3F7	Primary IDE Controller (slave drive)
3F8 – 3FF	Serial Port I

for hard disk access

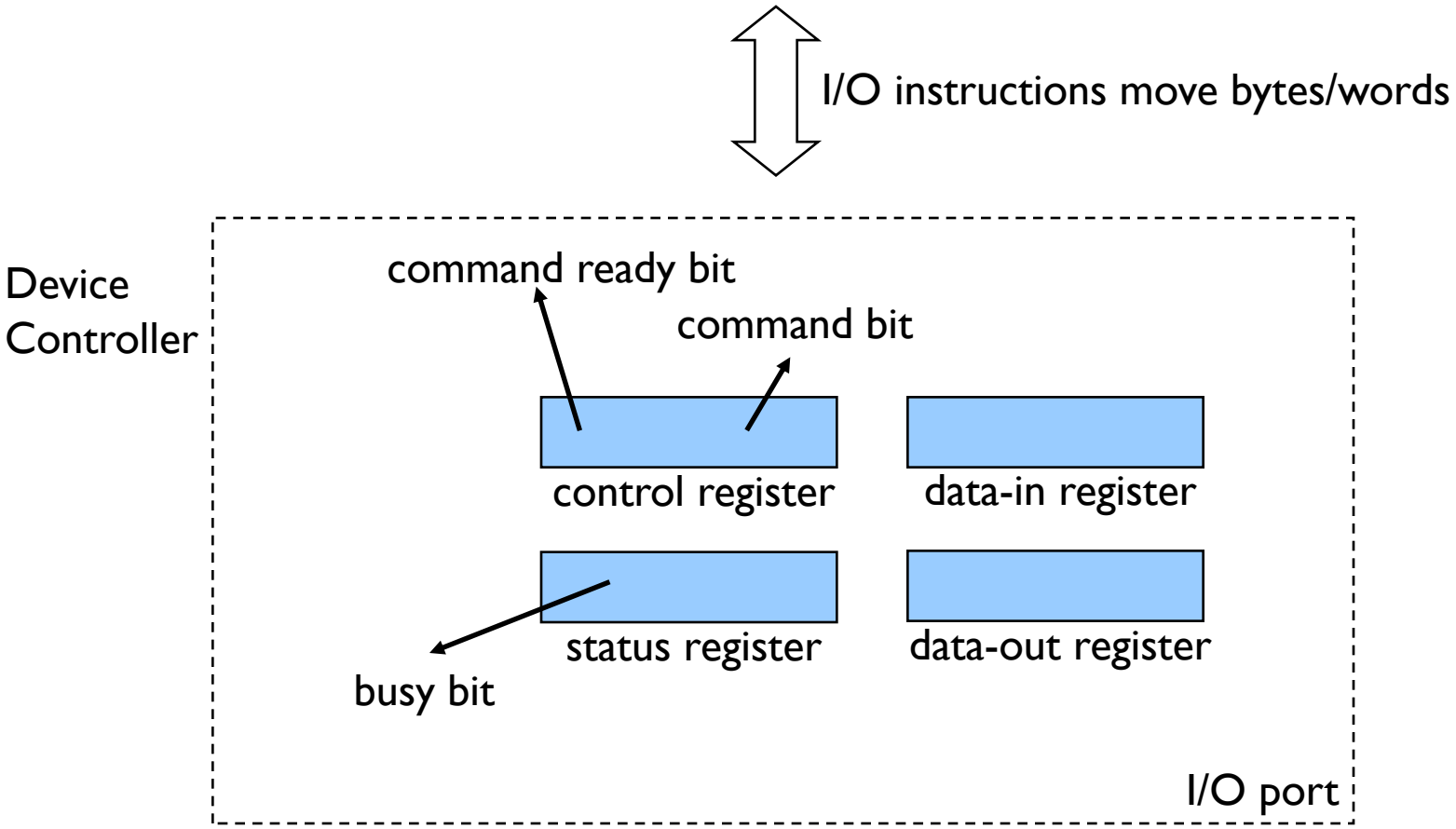
https://en.wikipedia.org/wiki/Input/output_base_address

A canonical device

- A device controller can be considered to have two parts:
  - Interface (I/O port): accessible to software running in CPU (i.e., operating system – device driver)
  - Internal structure
    - Microcontroller,
    - Memory,
    - ..



A typical device interface (I/O port)



Interacting with a device controller

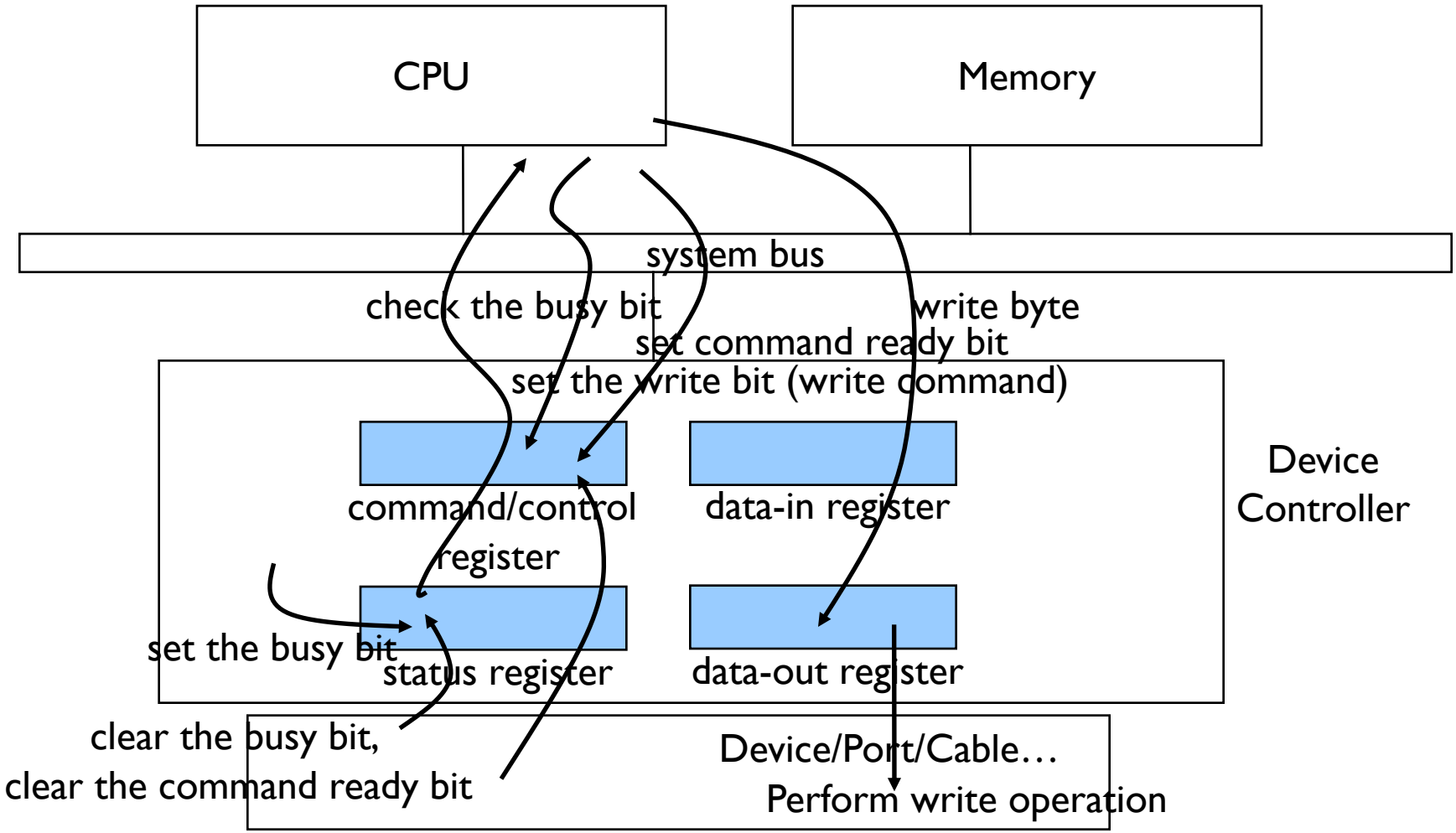
Interacting with the Device controller

- Host (CPU+Memory) and Device Controller interaction (data transfers and control) can be in one of 3 ways:
  - Polling
  - Interrupt driven I/O with PIO.
    - Use CPU to copy data between Memory and Device Controller. This is called Programmed I/O (PIO)
  - Interrupt driven I/O with DMA
    - Device Controller directly copies the data to/from Memory.

Transferring Data between host and device: Polling

- Busy-wait cycle to wait for I/O from or to device
  - Wait for input (Read operation) to arrive, or
  - Wait for output (Write operation) to complete
- State of device is checked in a loop, if device is busy or not
- If device is not busy, then I/O is started.
- Good for fast devices.
- Not good for slow devices. Keeping CPU busy in a loop for a long time.

Example: polling based writing



Example: polling based writing

CPU

1. read and check the busy bit
2. if busy go to 1.
3. set the write bit in command register
4. Write byte (word) into data out register
5. Set the command ready bit
6. Go to 1 (maybe after doing something else)

Controller

- notices command ready bit set
- sets the busy bit
- reads the command (it is write command), gets byte from data out register and writes the byte out (this may take time)
- clears the busy bit
- clears the command ready bit
- clears the error bit

Transferring Data between host and device: Interrupts

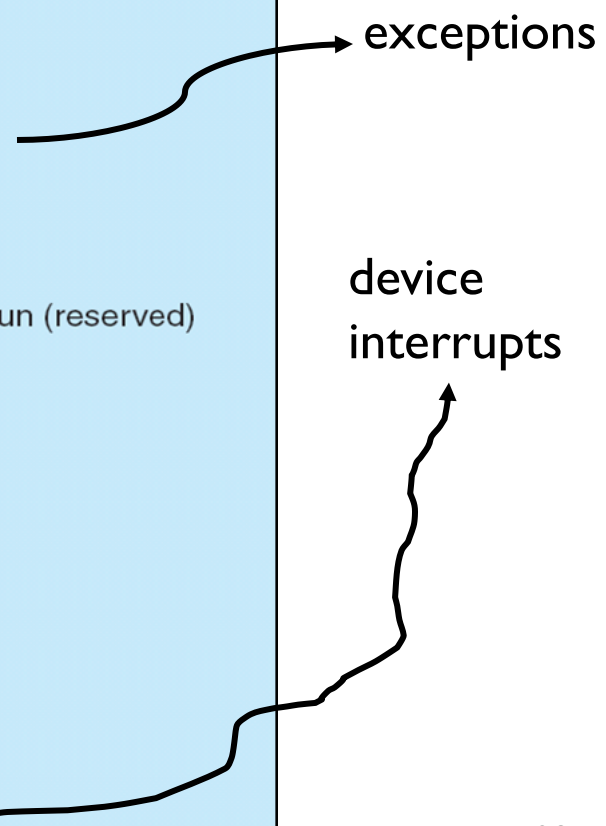
- Interrupt mechanism is used to do more efficient I/O.
- I/O device sends an interrupt to CPU:
  - When there is data to (R)ead (R interrupt)
  - or, when (W)rite (output) operation competed (W interrupt).
- An interrupt handler (service) routine (ISR) handles an interrupt.
- Interrupt (event) vector table is used by CPU to find the address of the correct interrupt handler.
  - indexed by an event/interrupt number (also called a *vector*).

Interrupts

- Interrupt mechanism is also used for exceptions.
  - exceptions: division by zero, page fault, illegal memory access, ...
  - exception handling: terminate process, for example.
  - some exceptions are not maskable.
- Dispatching to correct exception handler via event vector table.
- System calls are called via trap machine instruction, which is considered as a software interrupt. Control is transferred to the system call handler via an entry in event vector table (for example, entry `128: 0x80`).
- Multi-CPU systems can process interrupts concurrently.
- Interrupts are frequent, handlers must be fast.

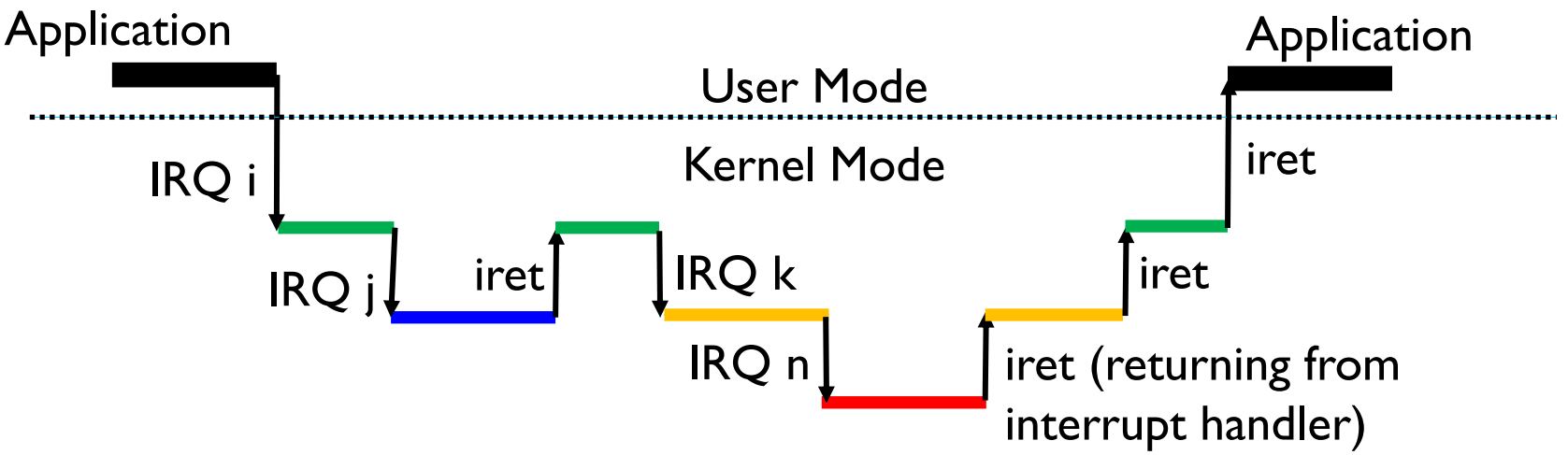
Intel Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts



Interrupts

- Possible to mask interrupts to ignore them or delay them for a while.
  - timer device interrupt can not be masked.
- Interrupts can be prioritized. (different levels).
- Interrupt nesting is possible.
  - While executing an ISR, it is possible to receive and handle a higher priority interrupt (jump to another ISR).

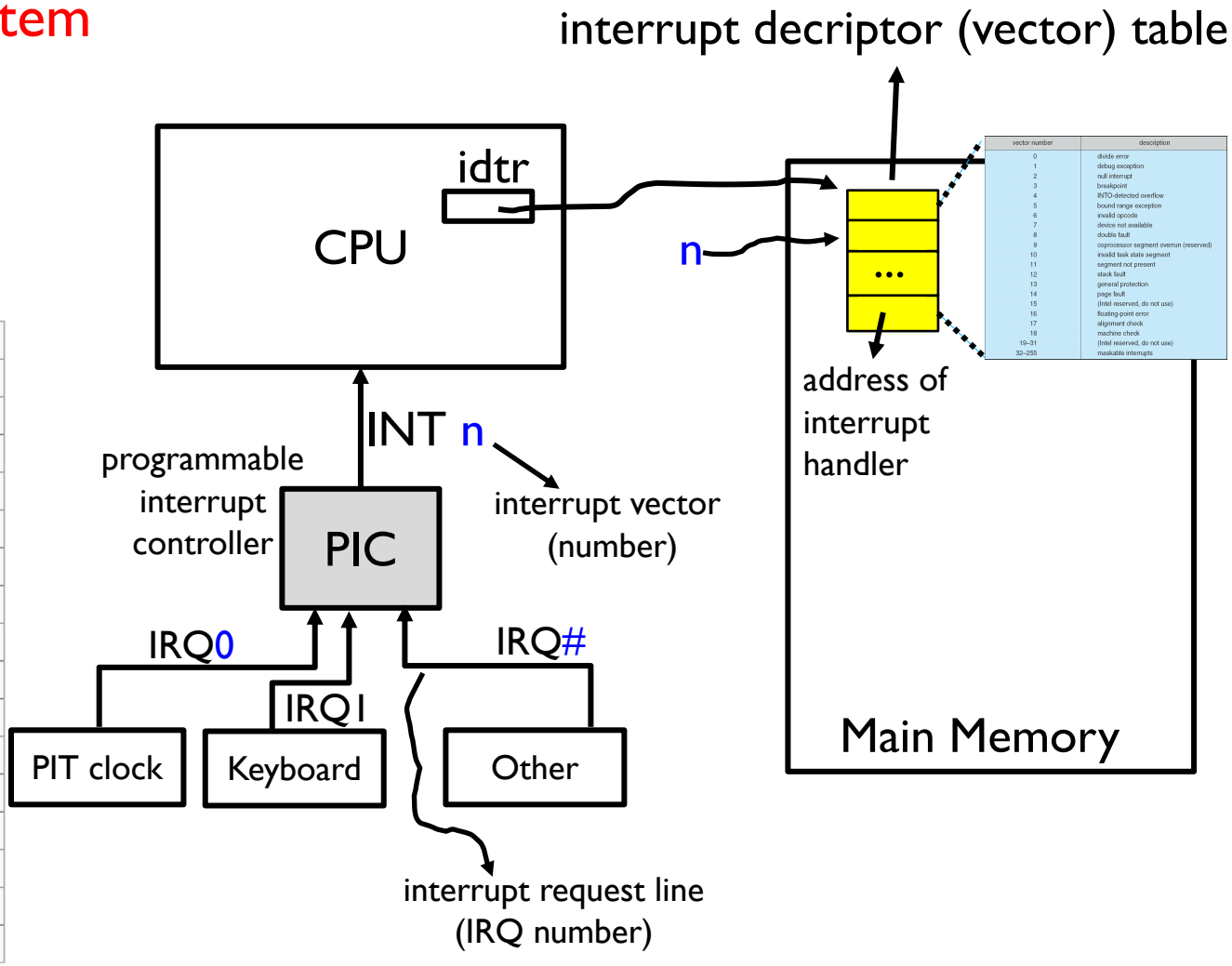


Interrupts

- Device interrupts are controlled and managed by programmable interrupt controller chip(s) (PIC or APIC, or...).
- Multiple interrupt lines go into the PIC chip.
- A device uses one of the lines.
  - Timer device, for example, uses line 0.
  - Keyboard uses line 1.
- Each line has an associated IRQ (interrupt request) number.
- PIC converts IRQ number to a vector (interrupt number).
  - Sends the vector to CPU.

Interrupts

Uniprocessor System



standard IRQ numbers

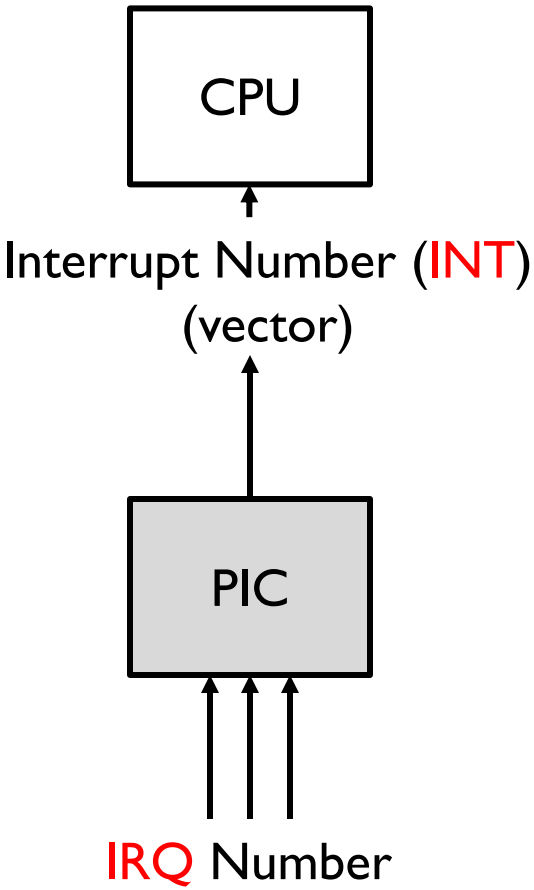
IRQ	Description
0	Programmable Interrupt Timer Interrupt
1	Keyboard Interrupt
2	Cascade (used internally by the two PICs. never raised)
3	COM2 (if enabled)
4	COM1 (if enabled)
5	LPT2 (if enabled)
6	Floppy Disk
7	LPT1 / Unreliable "spurious" interrupt (usually)
8	CMOS real-time clock (if enabled)
9	Free for peripherals / legacy SCSI / NIC
10	Free for peripherals / SCSI / NIC
11	Free for peripherals / SCSI / NIC
12	PS2 Mouse
13	FPU / Coprocessor / Inter-processor
14	Primary ATA Hard Disk
15	Secondary ATA Hard Disk

An example IRQ assignment by PIC

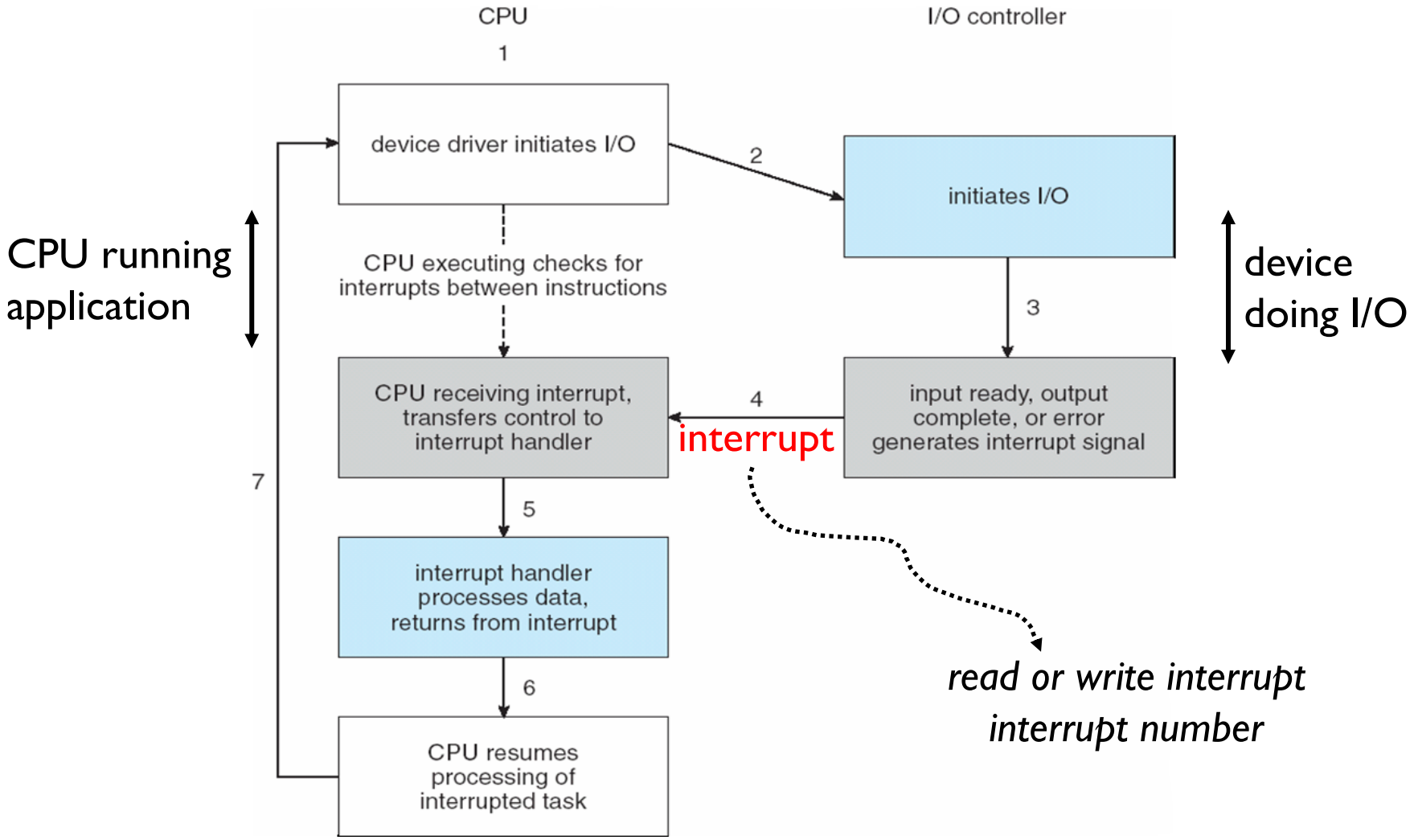
x86

By default: vector = 32 + IRQ# (but can be changed)

IRQ	INT	Hardware device
0	32	Timer (PIT)
1	33	Keyboard
2	34	PIC cascading
3	35	Second serial port
4	36	First serial port
6	38	Floppy disk
8	40	System clock (RTC)
10	42	Network interface
11	43	USB port, sound card
12	44	PS/2 mouse
13	45	Mathematical coprocessor
14	46	EIDE disk controller (ATA HDD)
15	47	EIDE disk controller (secondary ATA HDD)



Interrupt-Driven I/O Cycle



Latency

- We have lots of interrupts happening in a second.
  - Hundreds or thousands of interrupts per second in a PC.
  - Tens of thousands of interrupts per second in a server. A server has many CPUs (for example, 48 cores).
- For example, a macOS desktop generated 23,000 interrupts over 10 seconds.

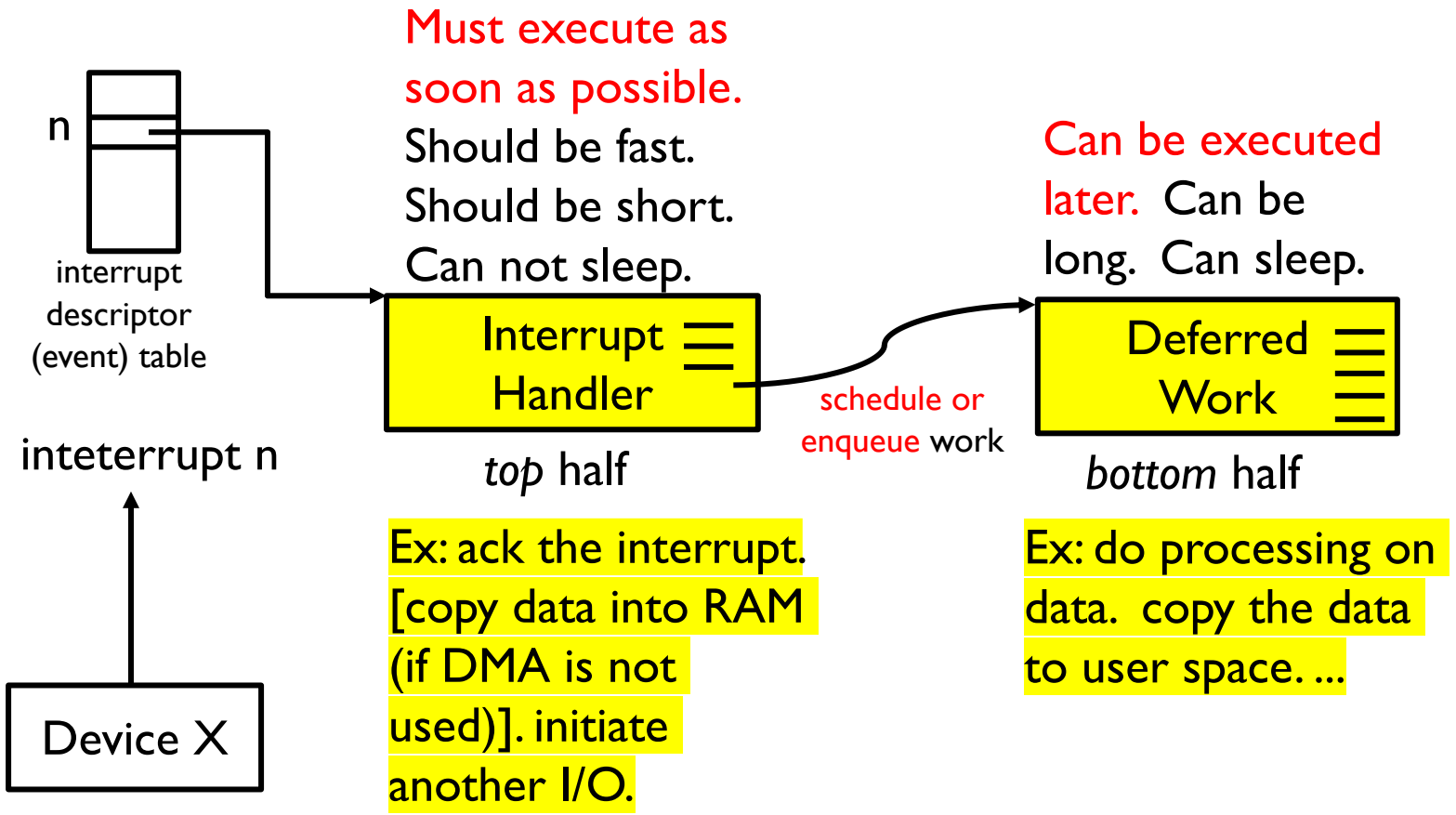
```
Fri Nov 25 13:55:59                                  0:00:10
                     SCHEDULER     INTERRUPTS
-----------------------------------------------
total_samples               13          22998

delays <  10 usecs          12          16243
delays <  20 usecs           1           5312
delays <  30 usecs           0            473
delays <  40 usecs           0            590
delays <  50 usecs           0             61
delays <  60 usecs           0            317
delays <  70 usecs           0              2
delays <  80 usecs           0              0
delays <  90 usecs           0              0
delays < 100 usecs           0              0
total  < 100 usecs          13          22998
```

In Linux, type:
cat /proc/interrupts
to see how many interrupts from devices.

Interrupt Processing

interrupt processing is usually done in two parts in Linux:
1) top half (immediate); 2) bottom half (later)



Direct Memory Access

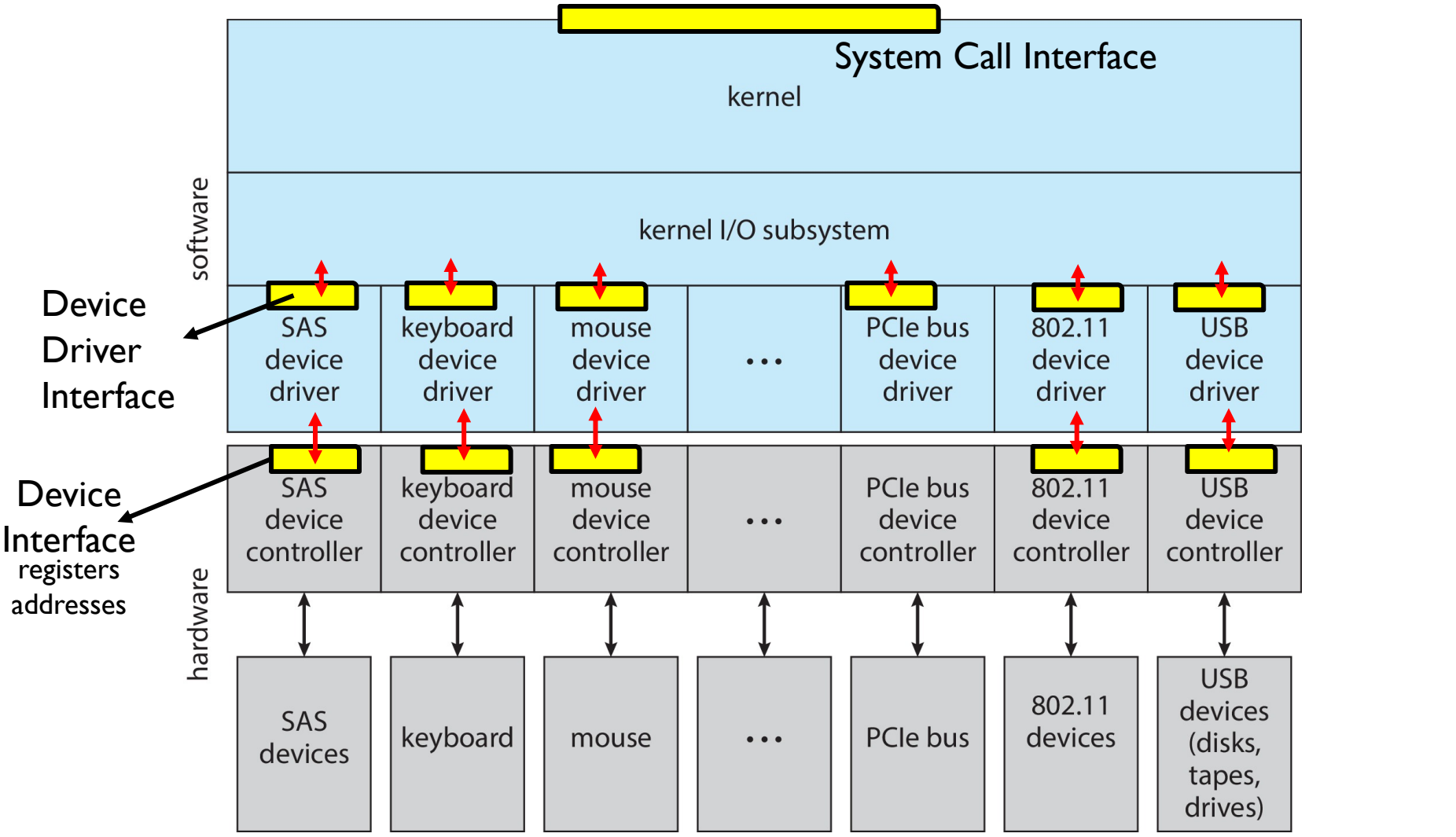
- Used to avoid programmed I/O (PIO) for large data movement
  - PIO: CPU moves data between device and memory
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory
- DMA chip orchestrates the transfer.
- Details depend on the architecture and DMA hardware.

Application I/O Interface

Application I/O Interface

- Devices vary in many dimensions
  - Character (stream) oriented or block oriented
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only
- Device-driver layer hides differences among I/O controllers from kernel.
- I/O system calls encapsulate device behaviors in generic classes (Application I/O Interface)

Application I/O Interface



Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Block and Character Devices

- Block devices include disk drives
  - Commands include read(), write(), seek()
  - Raw I/O or file-system access
  - Memory-mapped file access possible: mmap()
- Character devices include keyboards, mice, serial ports
  - Commands include get(), put() one character
  - Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows OS include socket interface
  - Create and use sockets to communicate: socket() system call.
  - Multiple sockets can be used: multiple connections.
  - A server may have many connections at the same time. connect() system call.
  - The select() system call enables a process to wait until data arrives from one or more of the many connections.
- In Unix we can use read(), write() system calls to send and receive data through a socket. Or recv() and send() system calls can be used.

Clocks and Timers

- Programmable interval timer (PIT) device is used for timings, periodic timer interrupts.
- RTC (real-time-clock) device is used to keep and get the current date/time information.
- The ioctl() system call can be used to manipulate the underlying device parameters. Many operating characteristics of devices (not only PIT and RTC) can be controlled via ioctl() sytem call from user level.

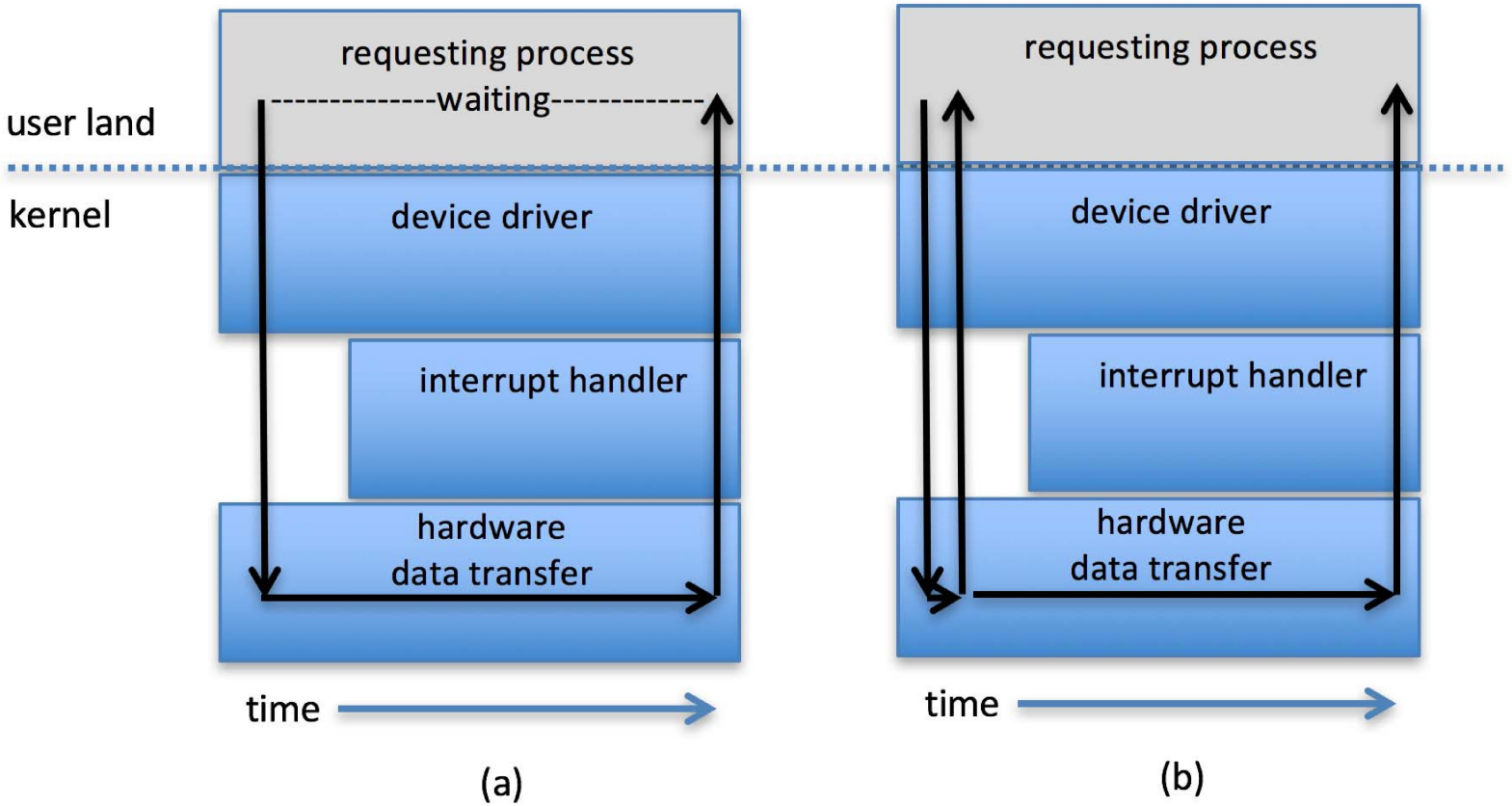
Example:

```
struct rtc_time current;
// read the RTC time/date
retval = ioctl(fd, RTC_RD_TIME, &current);
```

Blocking and Non-blocking I/O

- Blocking I/O: process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some need
- Non-blocking I/O: I/O call returns with as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
- Asynchronous I/O: process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Two I/O Methods



Blocking/Synchronous I/O

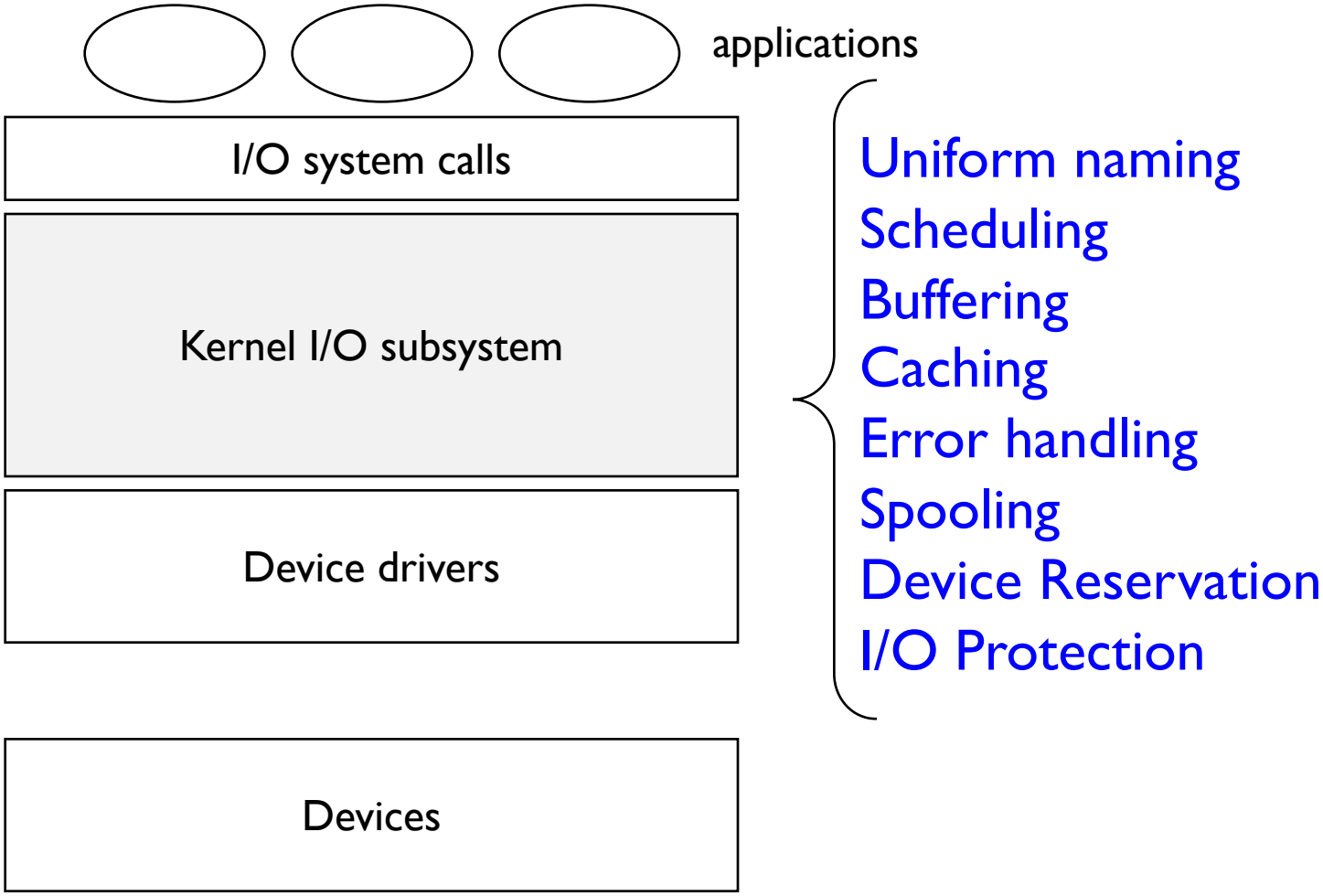
Asynchronous I/O

Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.
- For example, Unix readve() accepts a vector of multiple buffers to read into or write from
- It is also called scatter-gather I/O.
- This method better than multiple individual I/O calls:
  - Decreases context switching and system call overhead
  - Some versions provide atomicity
    - Avoid for example worry about multiple threads changing data as reads / writes occurring

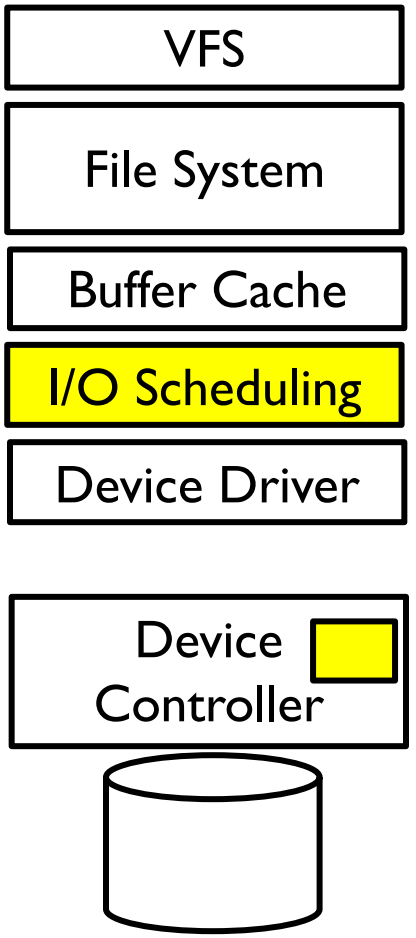
Kernel I/O Subsystem

Kernel I/O Subsystem



Kernel I/O Subsystem

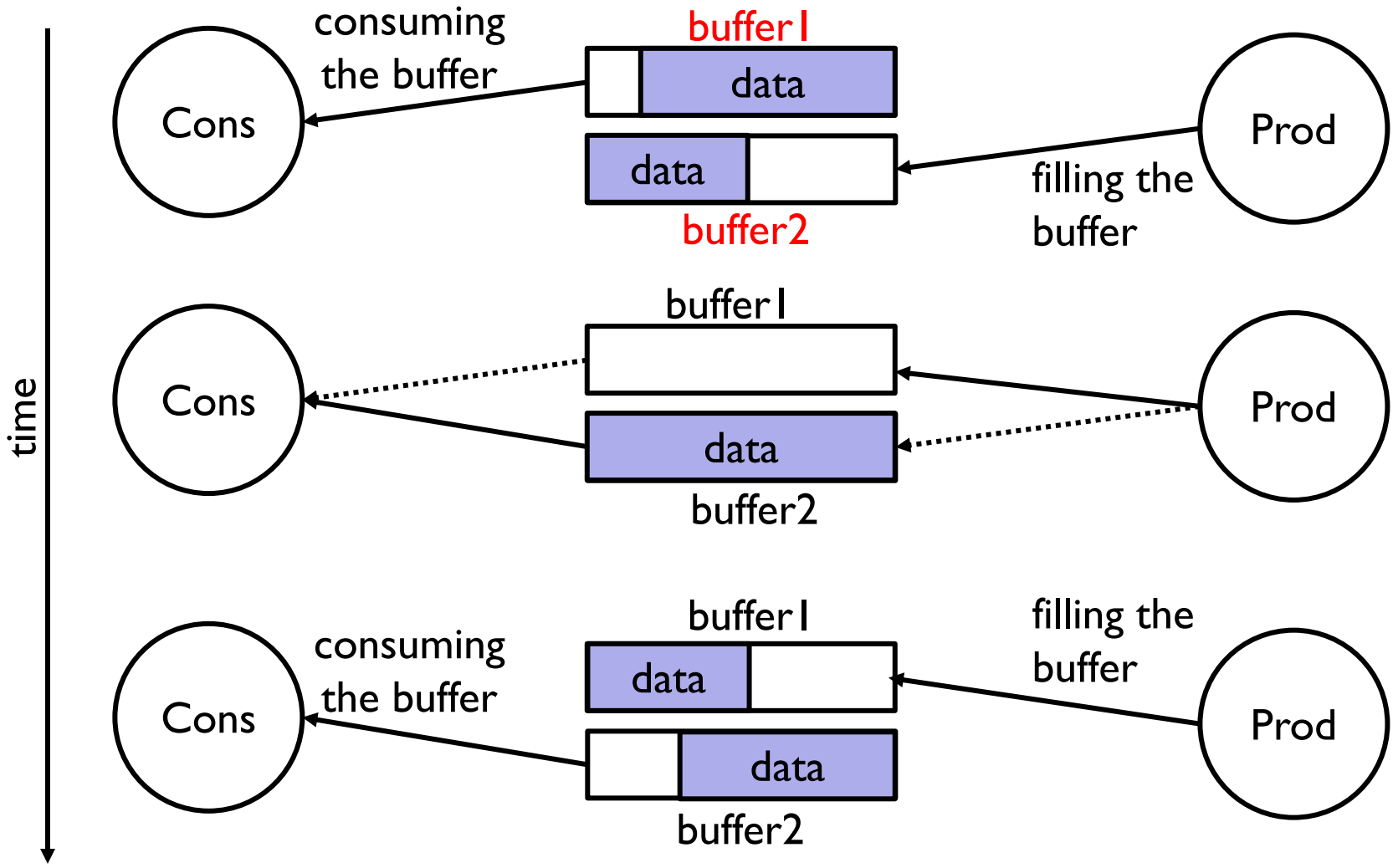
- Scheduling
  - I/O requests for a device may be enqueued.
  - Multiple processes requesting I/O.
  - For a disk (block device), for example, we have a request queue.
    - A lot of blocks requested by one or more processes (for reading or writing).
  - Requests can be ordered by the kernel (scheduling)
    - to increase performance or to be fair
  - Various scheduling algorithms can be used.



Kernel I/O Subsystem

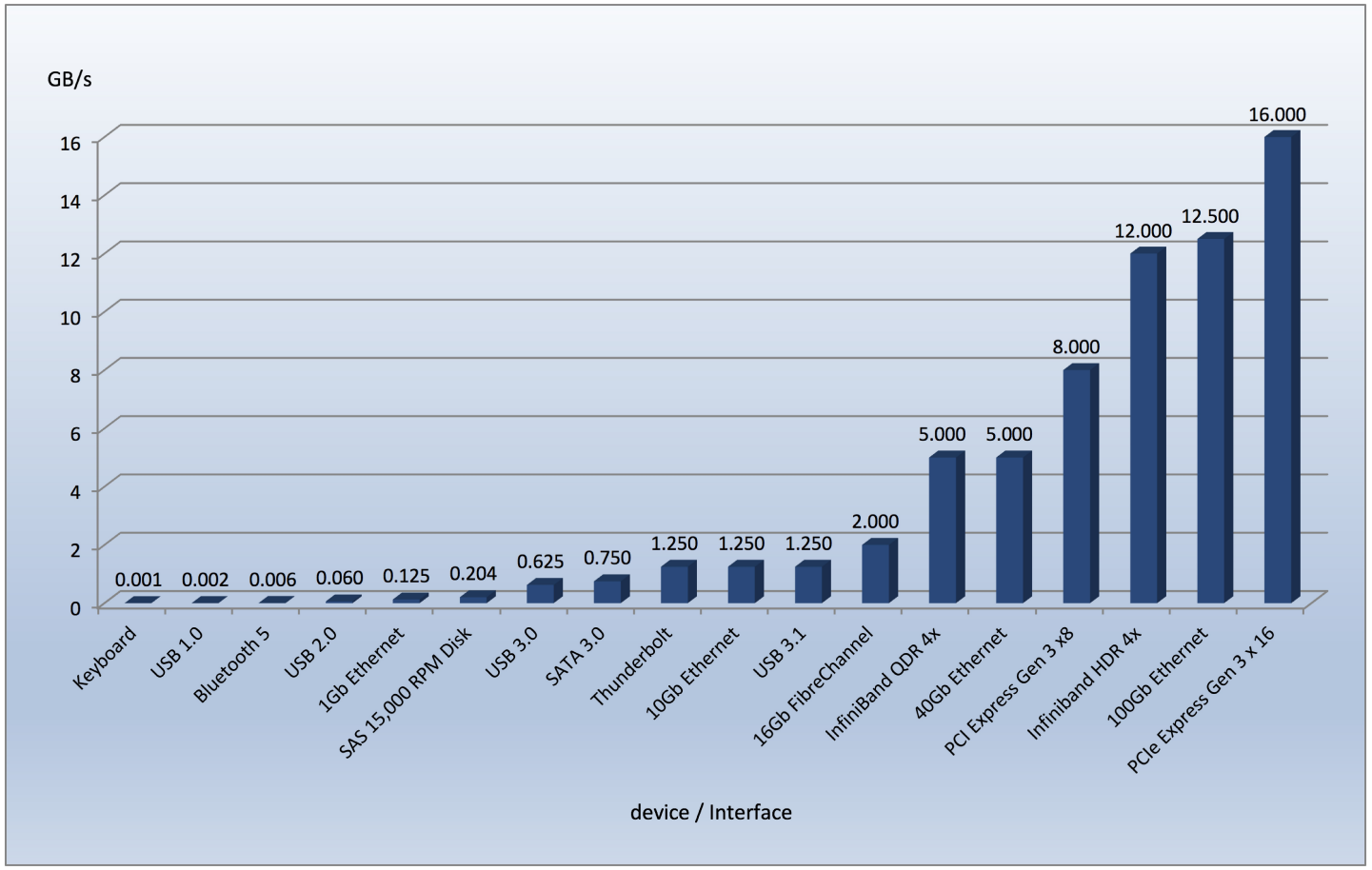
- Buffering:
  - Buffer: memory area that stores data being transferred between two devices or between a device and a process.
  - For a disk: a block read from disk or a block to write to a device is buffered in the kernel.
  - Buffering is done for 3 reasons:
    - To cope with device speed mismatch
      - Fast and slow device
    - To cope with device transfer size mismatch
    - To maintain "copy semantics": application issues write(), data is copied to kernel in a buffer.

Double buffering



Double buffering relaxes timing requirements

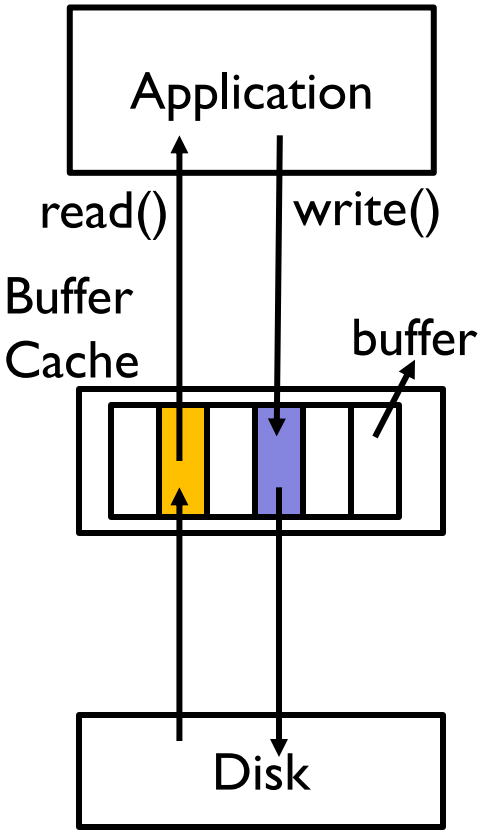
Common PC and Data-center I/O devices and Interface Speeds



Device speeds vary a lot.

Kernel I/O Subsystem

- Caching - fast memory that is holding a copy of data
  - Always just a copy.
  - Key to performance.
  - For example, "disk cache" is caching the frequently accessed disk blocks in memory.
  - Disk Cache: consists of a set of buffers. Each buffer is storing a block of data that is read from disk or that is to be written to disk.
  - Since disk cache is doing both buffering and caching, it is called Buffer Cache.



Kernel I/O Subsystem

- Spooling: hold output for a device
  - if device can serve only one request at a time
    - i.e., printing
  - Many processes can send output to the spooler at the same time
  - Spooler sends the outputs to the device one at a time.

Kernel I/O Subsystem

- Device reservation: provides exclusive access to a device
  - System calls for allocation and de-allocation
  - We need to be careful about deadlocks

Error Handling

- OS can recover from disk read, device unavailable, transient write failures.
- Most return an error number or code when I/O request fails.
- System error logs hold problem reports.

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
    - Kernel can execute I/O instructions (not the processes)
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too.

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many complex data structures to track buffers, memory allocation, "dirty" blocks
- Some use object-oriented methods and message passing to implement I/O.

I/O lifecycle and performance

I/O Requests to Hardware Operations

- Consider reading a file from a process:
  - fd = open(X, "r"); X is a filename (or pathname)
  - read(fd, buf, N)
- open()
  - Determine device holding file
    - disk? partition? CD? (look to mount table)
  - Translate name to file representation
    - Find out the inode for the file
- read()
  - Read data from disk into kernel buffer
    - Using inode and index nodes, data blocks are found
  - Make data available to requesting process
    - Copy requested data to the buffer of the user application
  - Return and resume execution of the process

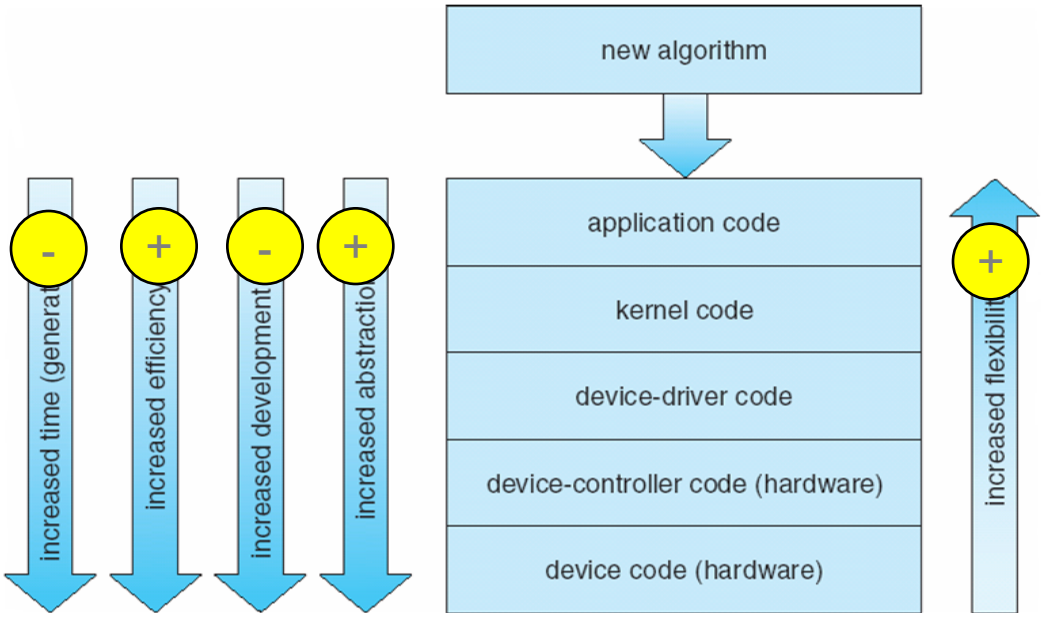
Performance

- I/O is a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
    - Software or hardware interrupts
  - Data copying
    - From device to device-driver/kernel, to application (vice versa)
- Network traffic especially stressful

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression



Where should a new I/O functionality or feature be implemented?

References

- Operating System Concepts, Silberschatz et al.
- Modern Operating Systems, Andrew S. Tanenbaum et al.
- OSTEP: Remzi Arpaci-Dusseau et al.
- Operating Systems: Principals and Practice, Anderson et al.
- Understanding the Linux Kernel. D. Bovet et al.
- https://wiki.osdev.org/Interrupts
- Linux Kernel Development, R. Love.
- xv6 Operating System book. R. Cox et al.