Chapter 13: I/O Systems

Zhi Wang
Florida State University

Content

- I/O hardware
- Application I/O interface
- Kernel I/O subsystem
- I/O performance

Objectives

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

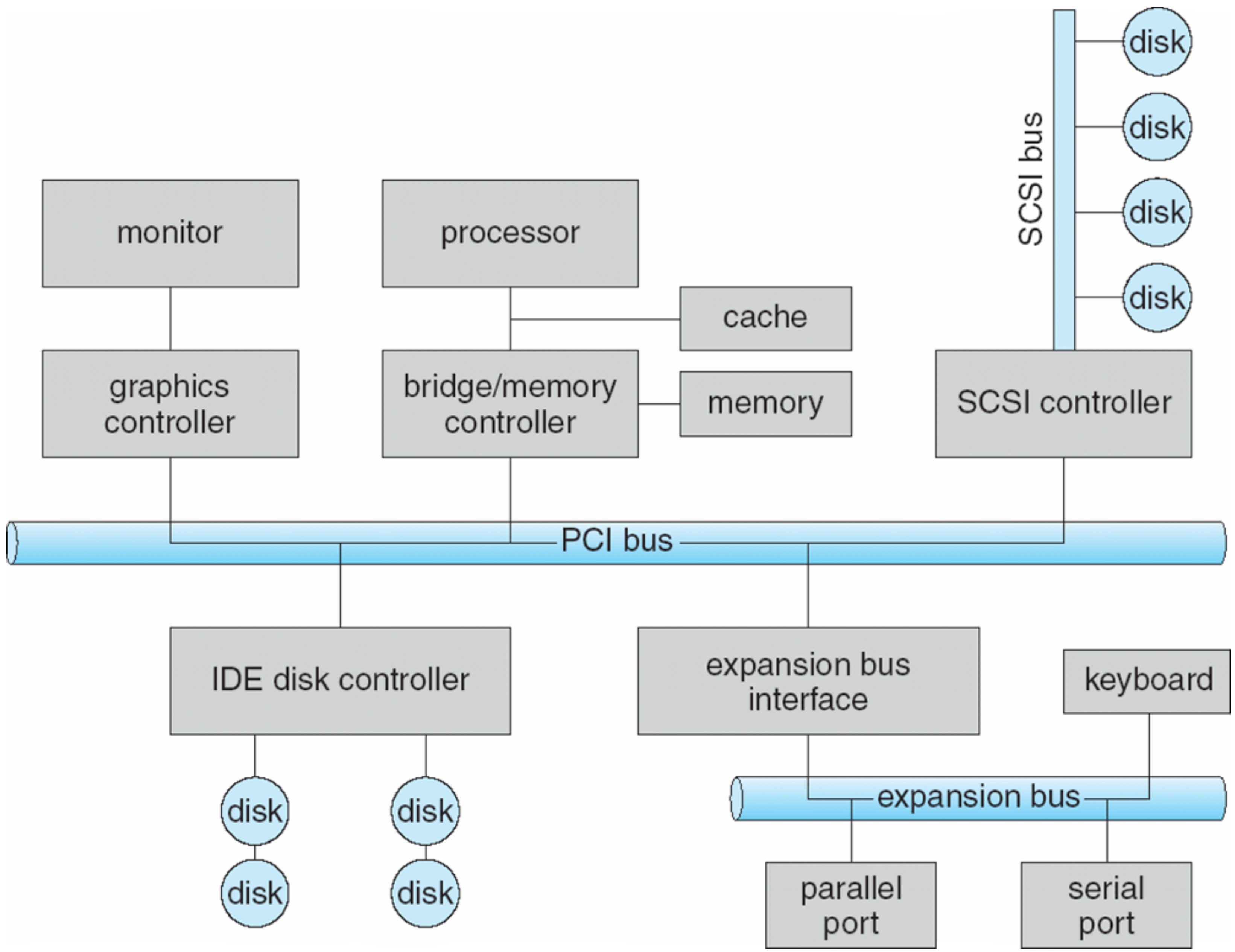
Overview

- I/O management is a major component of OS design and operation
  - important aspect of computer operation
    - I/O devices is the way computer to interact with user and other systems
  - I/O devices vary greatly
    - various methods to control them
    - performance varies
    - device drivers encapsulate device details; presents an uniform interface
  - new types of devices frequently emerges

I/O Hardware

- Incredible variety of I/O devices
  - storage, communication, human-interface
- Common concepts: signals from I/O devices interface with computer
  - **bus**: an interconnection between components (including CPU)
  - **port**: connection point for device
  - **controller**: component that control the device
    - can be integrated to device or separate circuit board
    - usually contains processor, microcode, private memory, bus controller, etc
- I/O access can use **polling** or **interrupt**

A Typical PC Bus Structure



I/O Hardware

- Some CPU architecture has dedicated I/O instructions
  - e.g., x86: in, out, ins, outs
- Devices usually provide registers for data and control I/O of device
  - device driver places (pointers to) commands and data to register
  - registers include **data-in**/**data-out**, **status**, **control** (or command) register
  - typically 1-4 bytes, or FIFO buffer
- Devices are assigned addresses for registers or on-device memory
  - **direct I/O instructions**
    - to access (mostly) registers
  - **memory-mapped I/O**
    - data and command registers mapped to processor address space
    - to access (large) on-device memory (graphics)

I/O Ports 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)

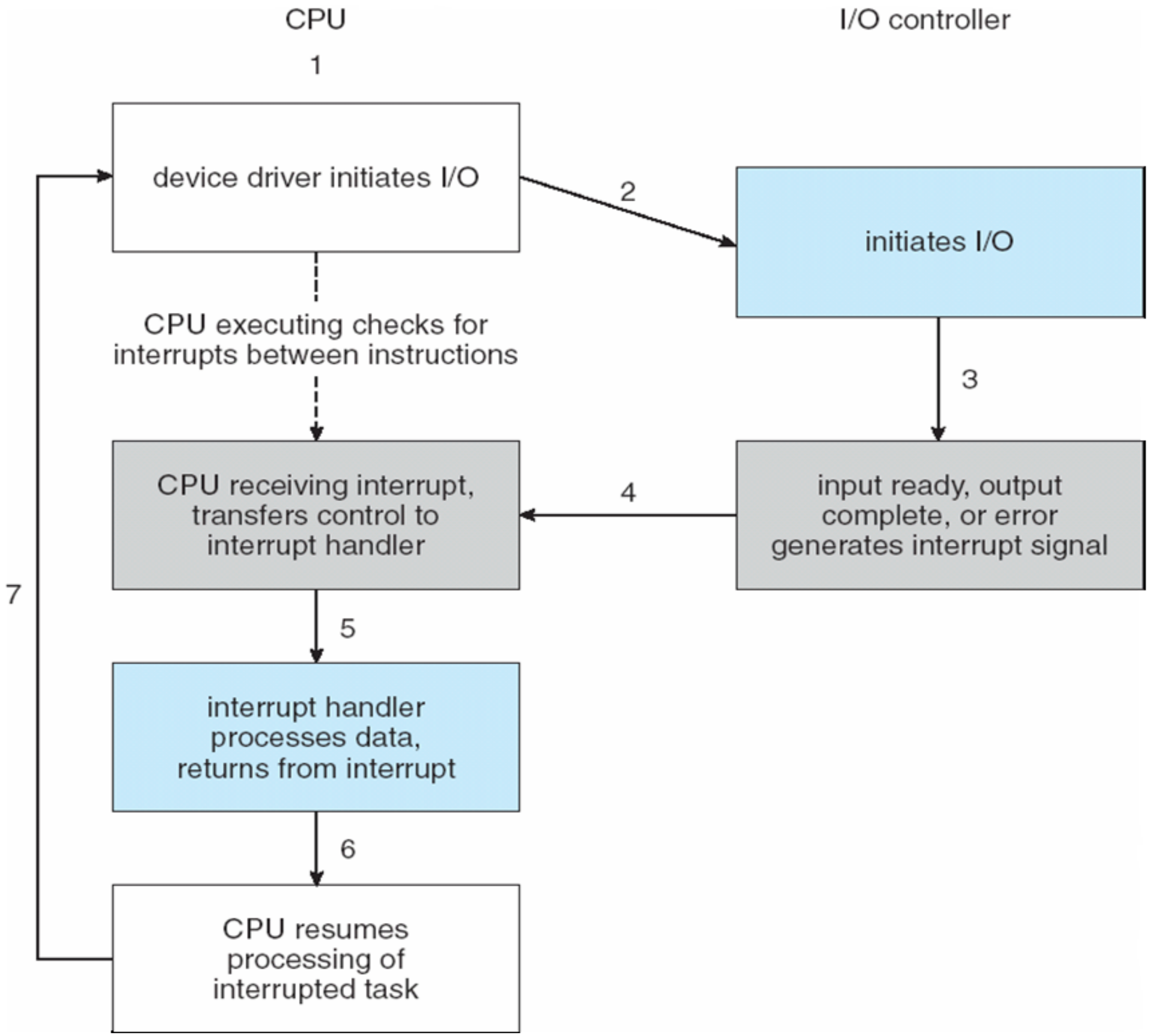
Polling

- For each I/O operation:
  - busy-wait if device is busy (status register)
  - send the command to the device controller (command register)
  - read status register until it indicates command has been executed
  - read execution status, and possibly reset device status
- Polling requires busy wait
  - reasonable if device is fast; inefficient if device slow

Interrupts

- Polling requires busy-wait, inefficient use of CPU resource
- Interrupts can avoid busy-wait
  - device driver send a command to the controller, and return
  - OS can schedule other activities
  - device will interrupt the processor when command has been executed
  - OS retrieves the result by handling the interrupt
- Interrupt-based I/O requires **context switch** at start and end
  - if interrupt frequency is extremely high, context switch wastes CPU time
  - solution: use polling instead
    - example: **NAPI** in Linux enables polling under very high network load

Interrupt-Driven I/O Cycle



Intel Pentium Interrupt 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

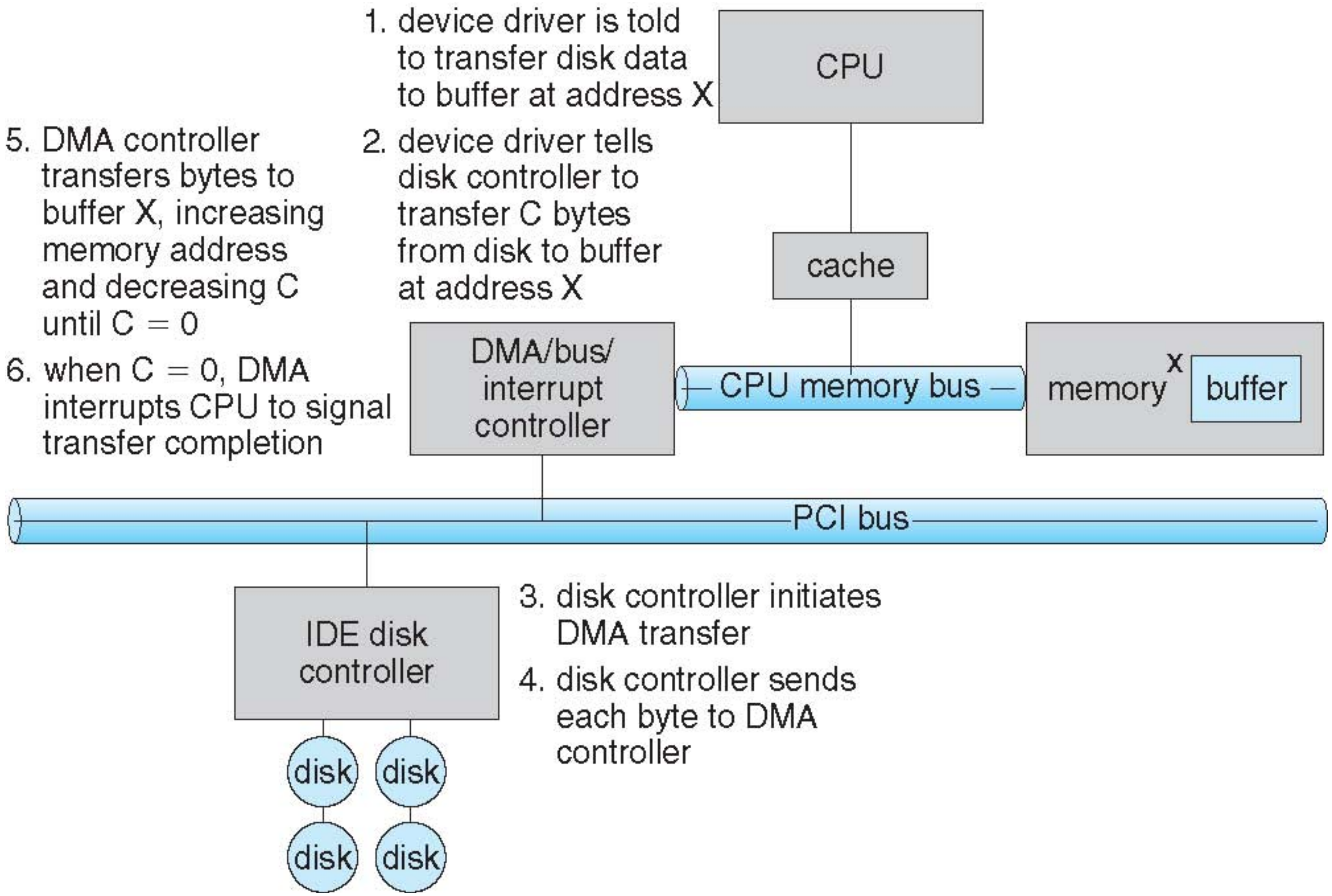
Interrupts

- Interrupt is also used for exceptions
  - protection error for access violation
  - page fault for memory access error
  - software interrupt for system calls
- Multi-CPU systems can process interrupts concurrently
  - sometimes a CPU may be dedicated to handle interrupts
  - interrupts can also have **CPU affinity**

Direct Memory Access

- DMA transfer data directly between I/O device and memory
  - OS only need to issue commands, data transfers bypass the CPU
  - no programmed I/O (one byte at a time), data transferred in large blocks
  - it requires DMA controller in the device or system
- OS issues commands to the DMA controller
  - a command includes: operation, memory address for data, count of bytes…
  - usually it is the pointer of the command written into the command register
  - when done, device interrupts CPU to signal completion

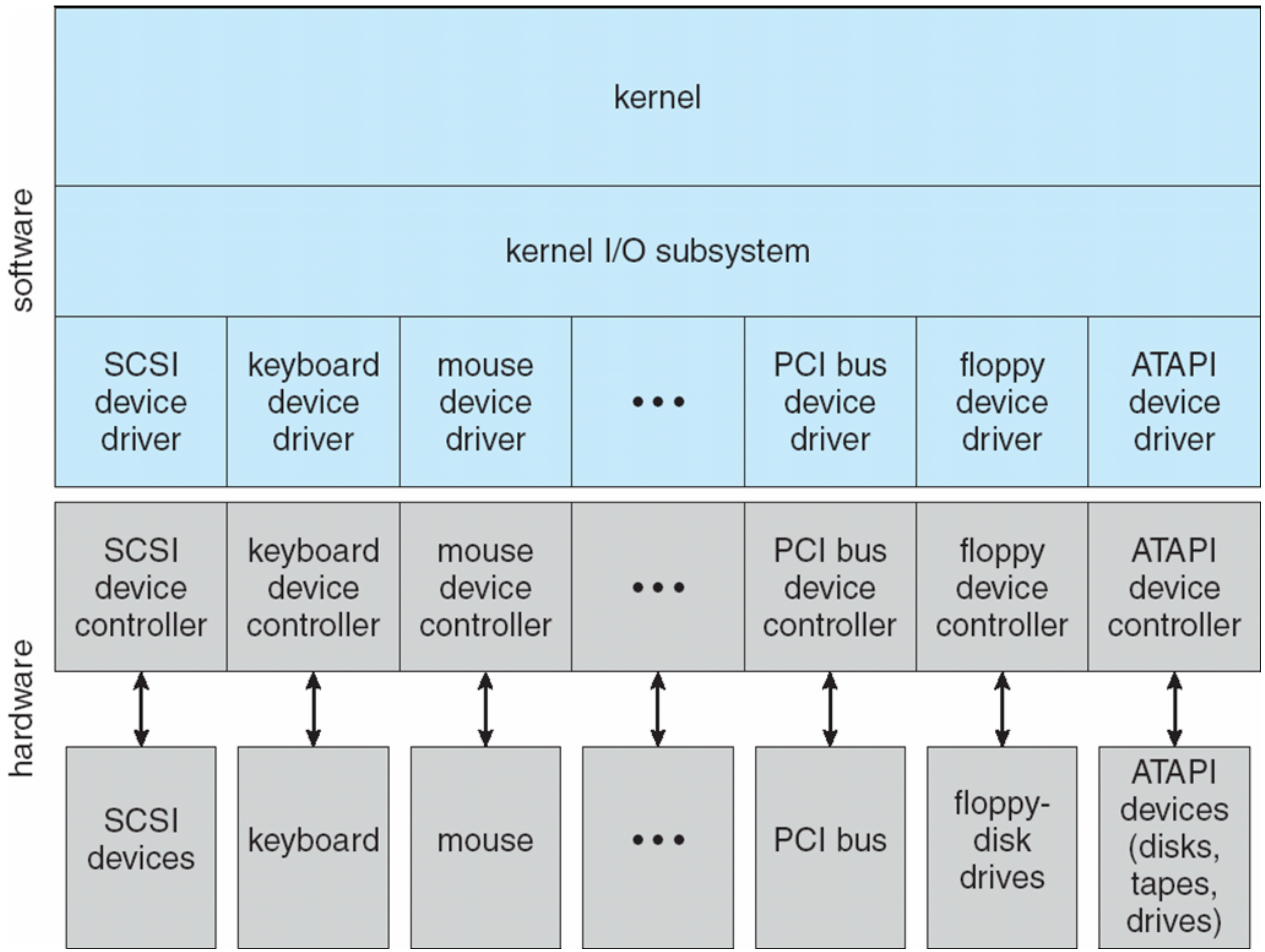
Six Steps of DMA Transfer



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
  - in Linux, devices can be accessed as files; low-level access with ioctl
- Device-driver layer hides differences among I/O controllers from kernel
  - each OS has its own I/O subsystem and device driver frameworks
  - new devices talking already-implemented protocols need no extra work
- Devices vary in many dimensions
  - character-stream or block
  - sequential or random-access
  - synchronous or asynchronous (or both)
  - sharable or dedicated
  - speed of operation
  - read-write, read only, or write only

Kernel I/O Structure



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

Characteristics of I/O Devices

- Broadly, I/O devices can be grouped by the OS into
  - **block I/O**
  - **character I/O** (Stream)
  - **memory-mapped file access**
  - **network sockets**
- Direct manipulation of I/O device usually an escape / back door
  - Linux's **ioctl** call to send commands to a device driver

Block and Character Devices

- Block devices access data in blocks, such as disk drives…
  - commands include read, write, seek
  - raw I/O, direct I/O, or file-system access
  - memory-mapped file access possible (e.g., memory-mapped files)
  - DMA
- Character devices include keyboards, mice, serial ports…
  - very diverse types of devices

Network Devices

- Varying enough from block and character to have own interface
  - very different from pipe, mailbox…
- Popular interface for network access is the **socket** interface
  - it separates network protocol from detailed network operation
  - some non-network operations are implemented as sockets
    - e.g., Unix socket

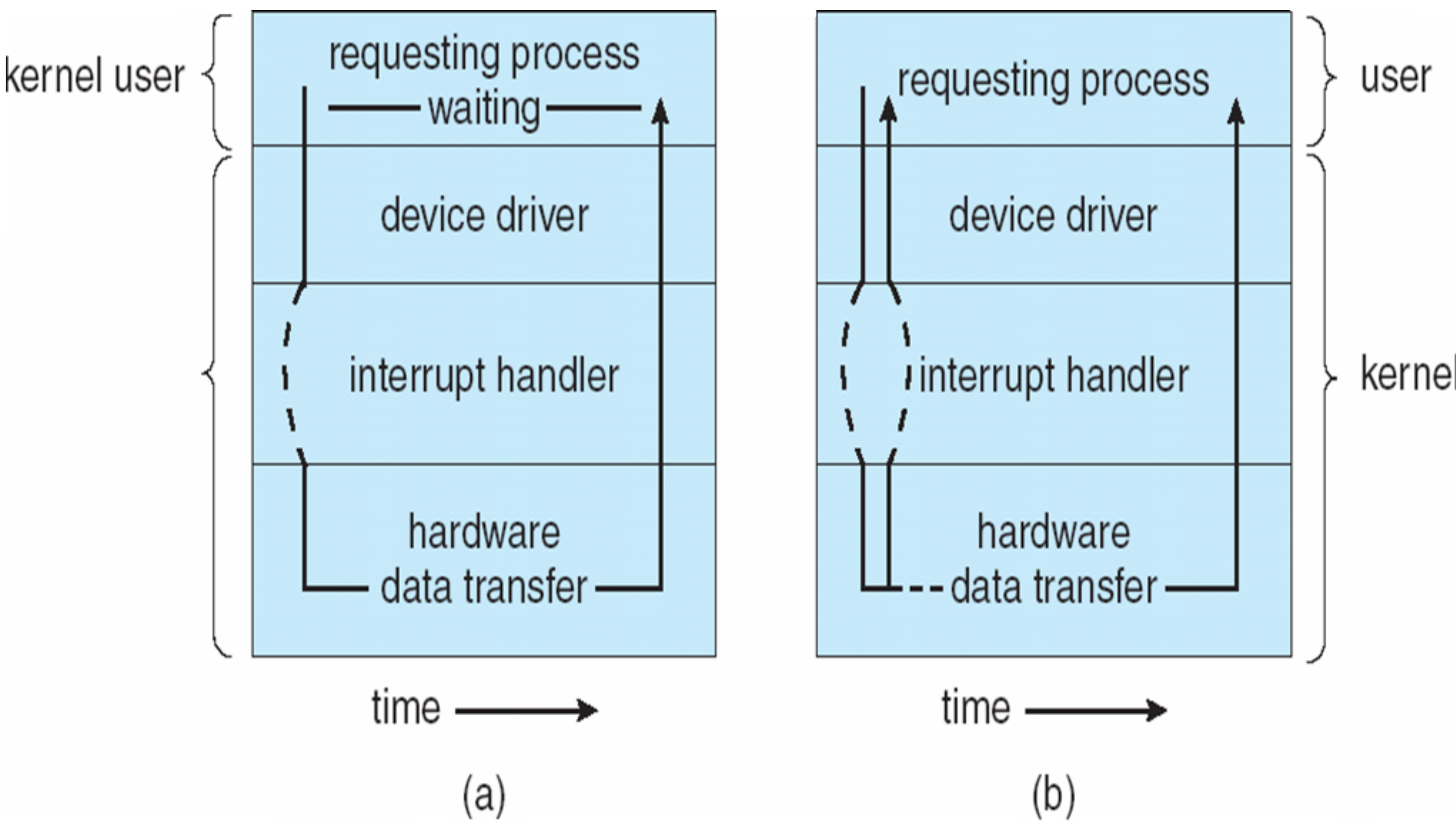
Clocks and Timers

- Clocks and timers can be considered as character devices
  - very important devices as they provide current time, elapsed time, timer
- Normal resolution about 1/60 second, some OS provides higher-resolution ones

Synchronous/Asynchronous I/O

- **Synchronous I/O** includes blocking and non-blocking I/O
  - **blocking I/O**: process suspended until I/O completed
    - easy to use and understand, but may be less efficient
    - insufficient for some needs
  - **non-blocking I/O**: I/O calls return as much data as available
    - process does not block, returns whatever existing data (read or write)
    - use select to find if data is ready, then use read or write to transfer data
- **Asynchronous I/O**: process runs while I/O executes,
  - I/O subsystem signals process when I/O completed via signal or callback
  - difficult to use but very efficient

Two I/O Methods



synchronous

asynchronous

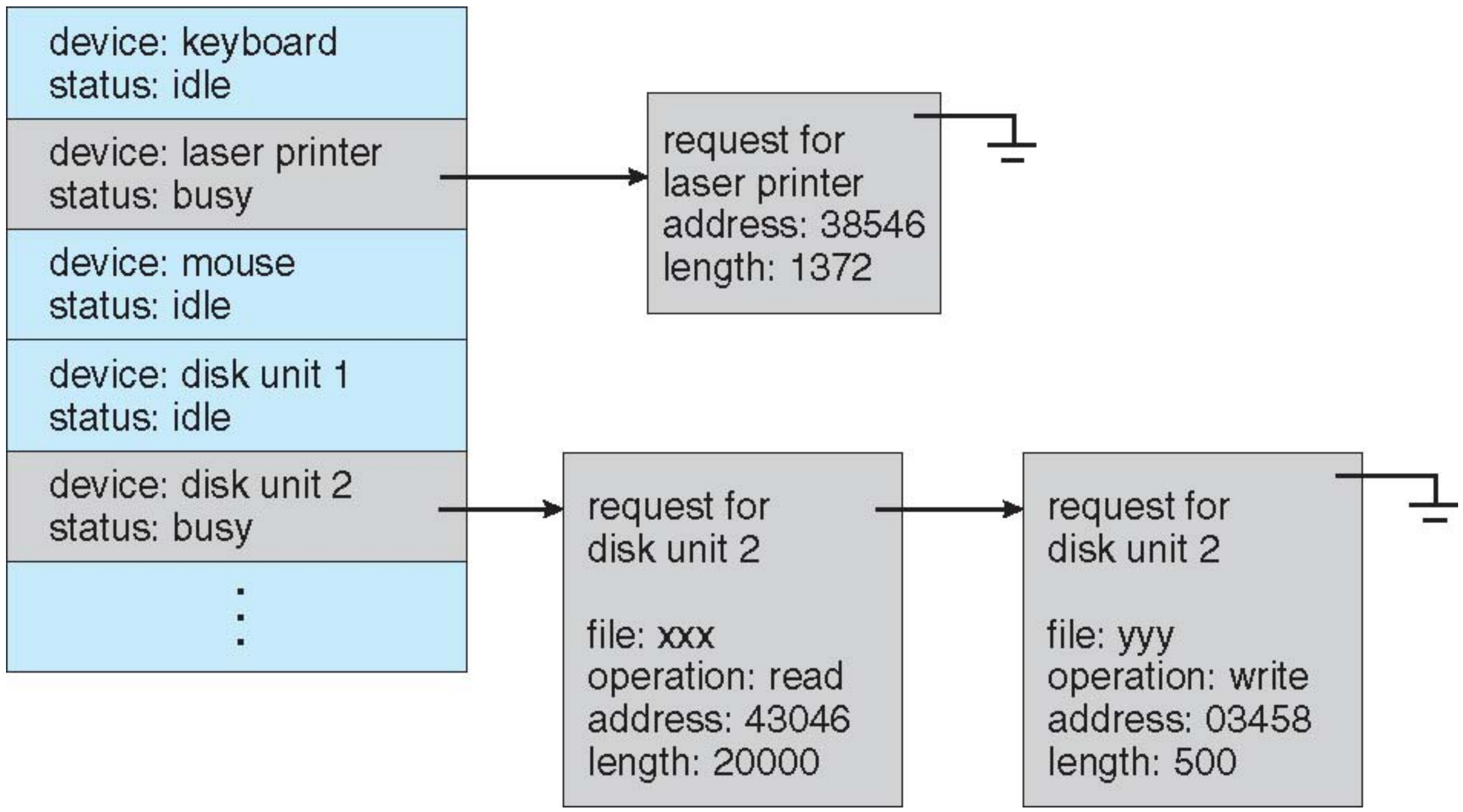
Kernel I/O Subsystem

- **I/O scheduling**
  - to queue I/O requests via per-device queue
  - to schedule I/O for fairness and quality of service
- **Buffering** - store data in memory while transferring between devices
  - to cope with device speed mismatch
  - to cope with device transfer size mismatch
  - to maintain “copy semantics”
  - to improve performance (double buffering in video)

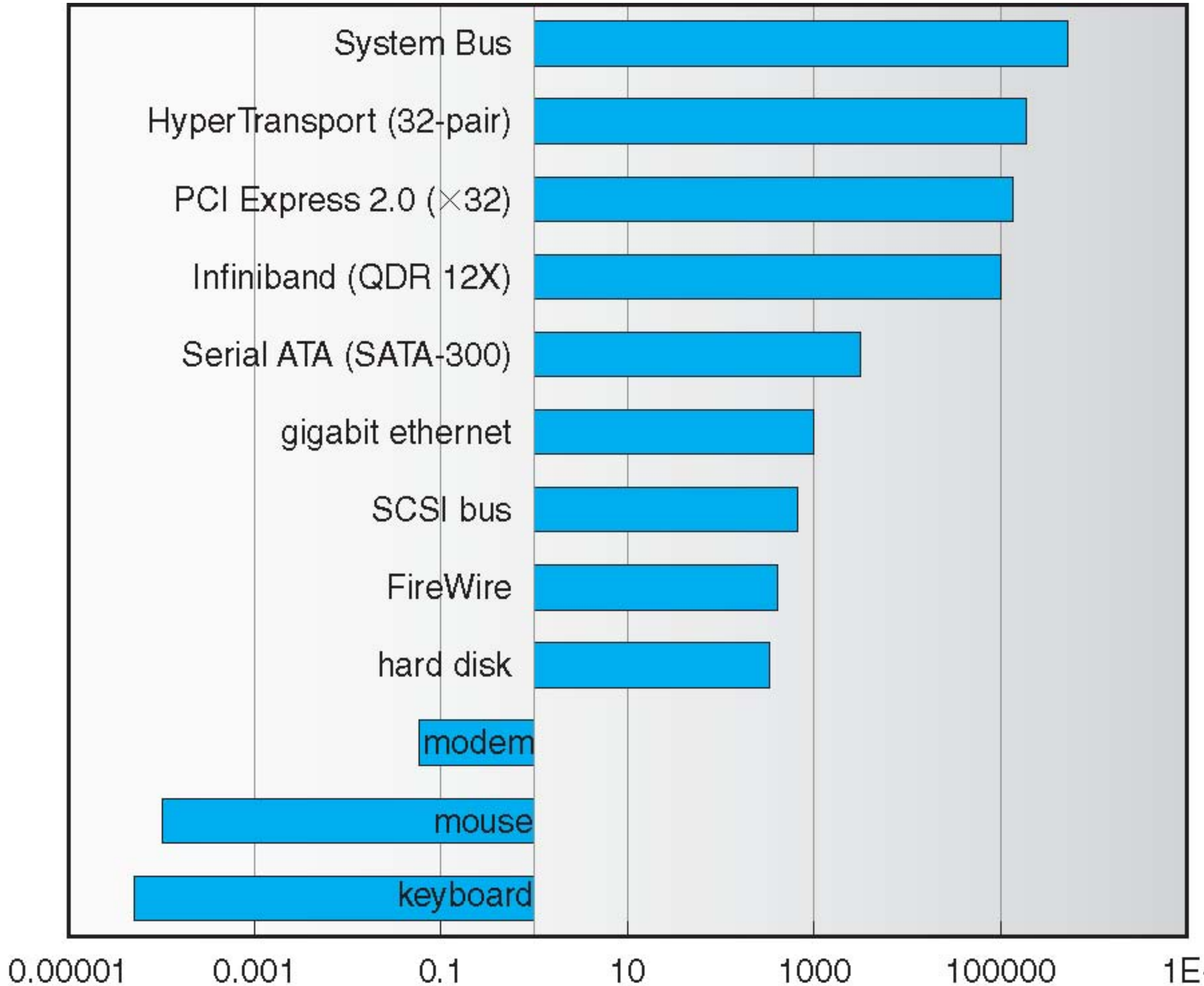
Kernel I/O Subsystem

- **Caching**: hold a copy of data for fast access
  - key to performance
  - sometimes combined with buffering
- **Spooling**: hold output if device can serve only one request at a time
  - i.e., printing
- **Device reservation**: provides exclusive access to a device
  - system calls for allocation and de-allocation
  - watch out for deadlock

Device-status Table



Sun Enterprise 6000 Device-Transfer Rates



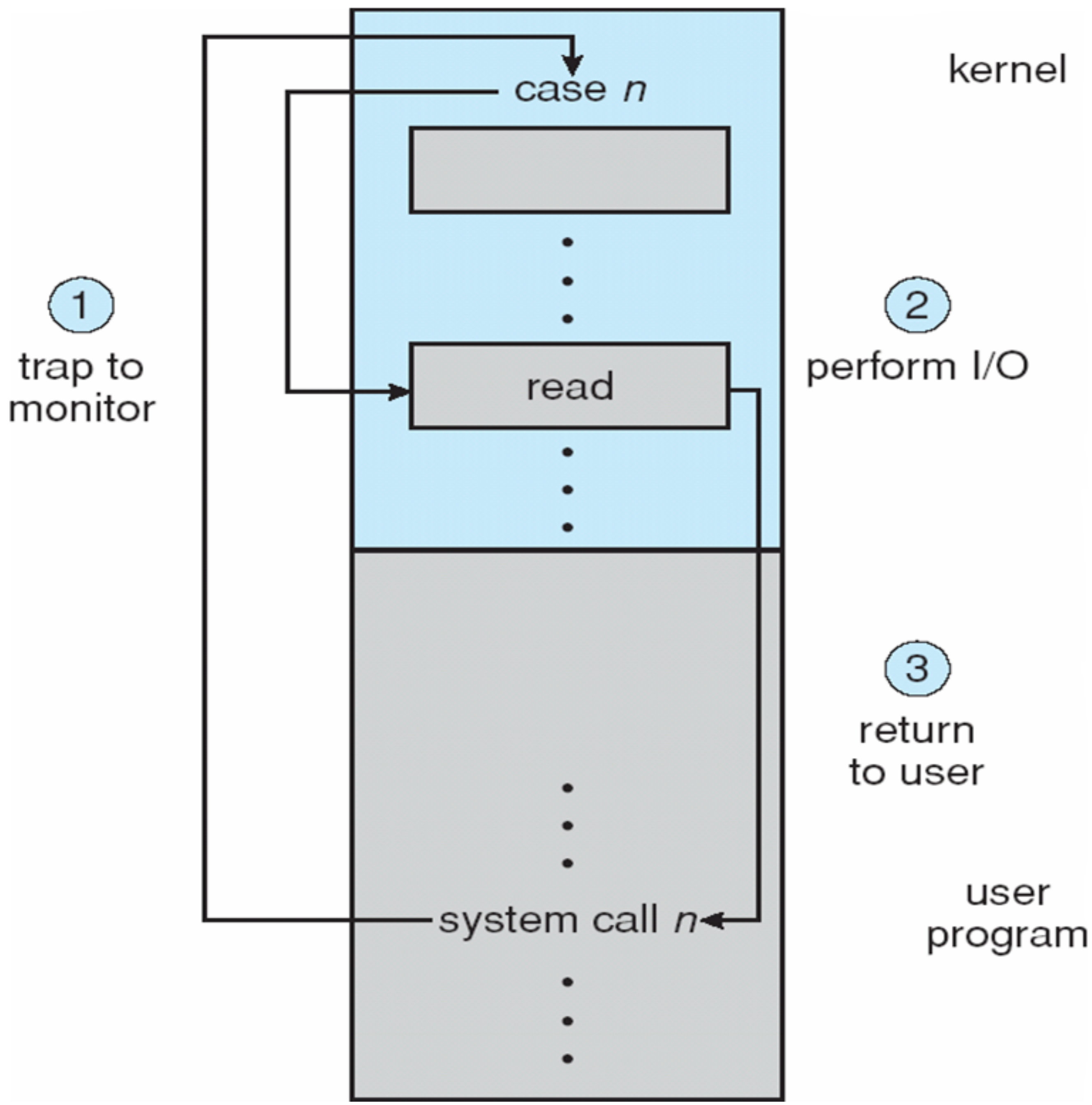
Error Handling

- Some OSes try to recover from errors
  - e.g., device unavailable, transient write failures
  - sometimes via retrying the read or write
  - some systems have more advanced error handling
    - track error frequencies, stop using device with high error frequency
- Some OSes just return an error number or code when I/O request fails
  - system error logs hold problem reports

I/O Protection

- OS need to protect I/O devices
  - e.g., keystrokes can be stolen by a **keylogger** if keyboard is not protected
  - always assume user may attempt to obtain illegal I/O access
- To protect I/O devices:
  - define all I/O instructions to be privileged
    - I/O must be performed via system calls
  - memory-mapped I/O and I/O ports must be protected too

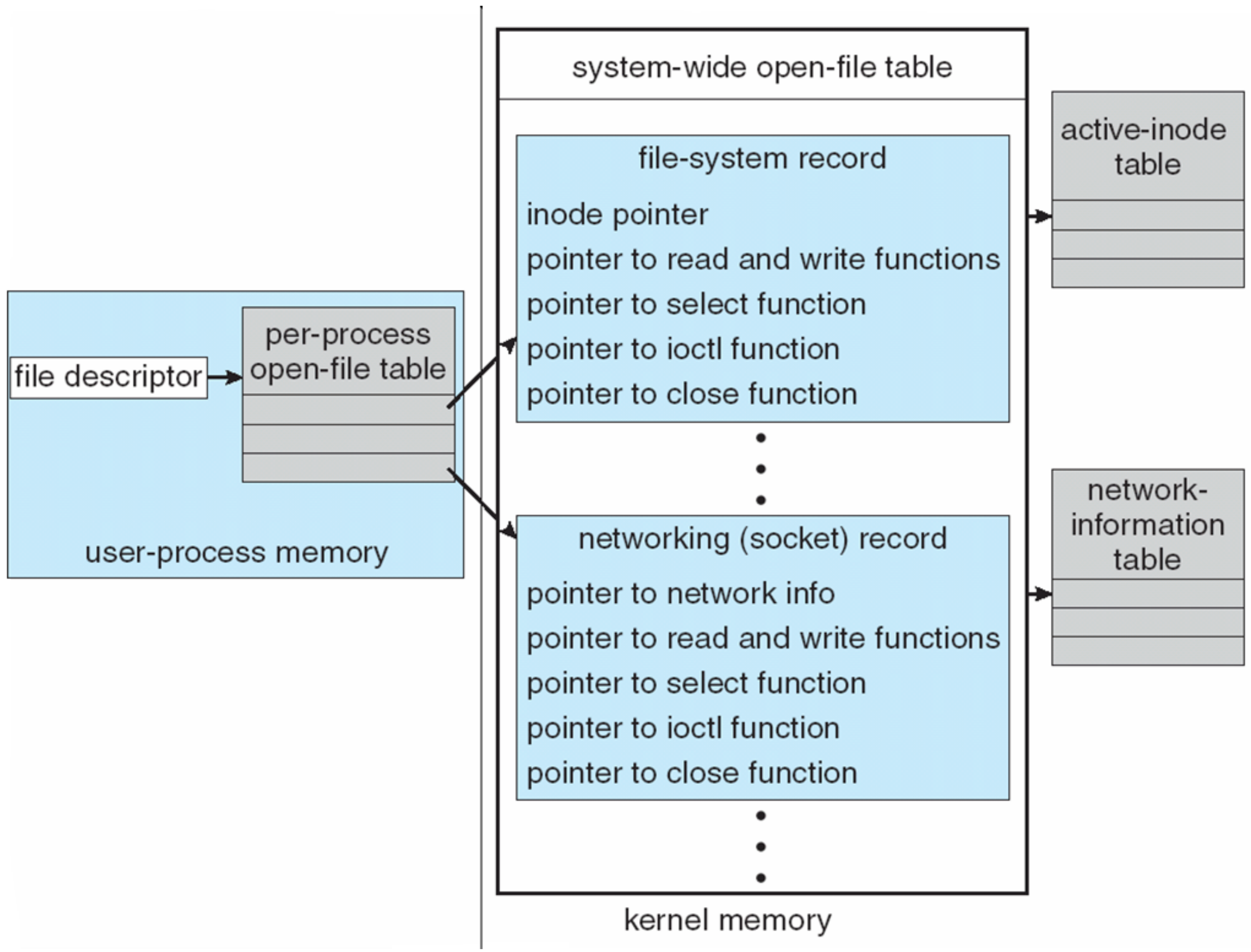
Use System Call to Perform I/O



Kernel Data Structures

- Kernel keeps state info for I/O components
  - e.g., open file tables, network connections, character device state
  - many data structures to track buffers, memory allocation, “dirty” blocks
    - sometimes very complicated
- Some OS uses message passing to implement I/O, e.g., Windows
  - message with I/O information passed from user mode into kernel
  - message modified as it flows through to device driver and back to process

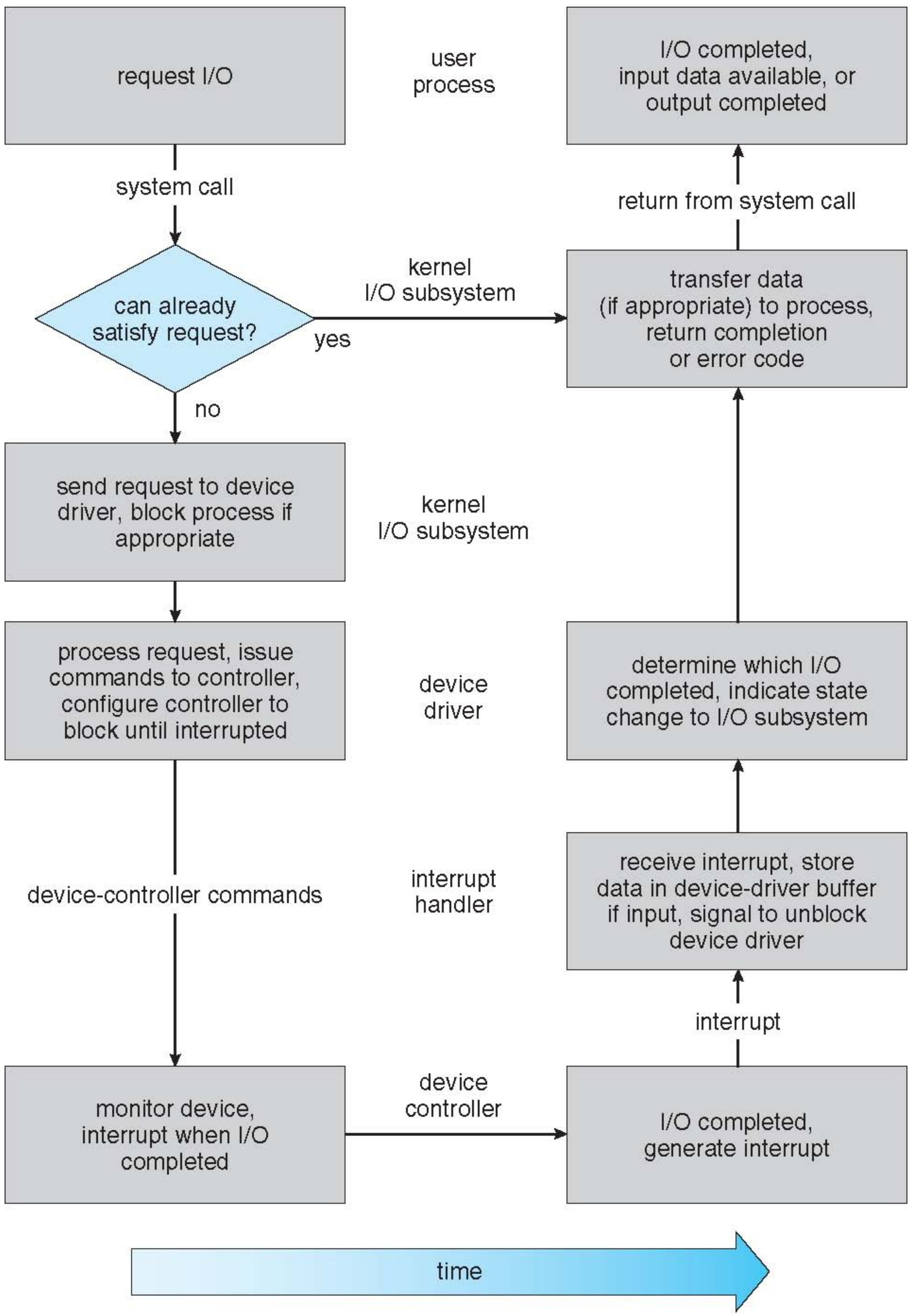
UNIX I/O Kernel Structure



I/O Requests to Hardware

- System resource access needs to be mapped to hardware
- Consider reading a file from disk for a process:
  - determine device holding file
  - translate name to device representation
  - physically read data from disk into buffer
  - make data available to requesting process
  - return control to process

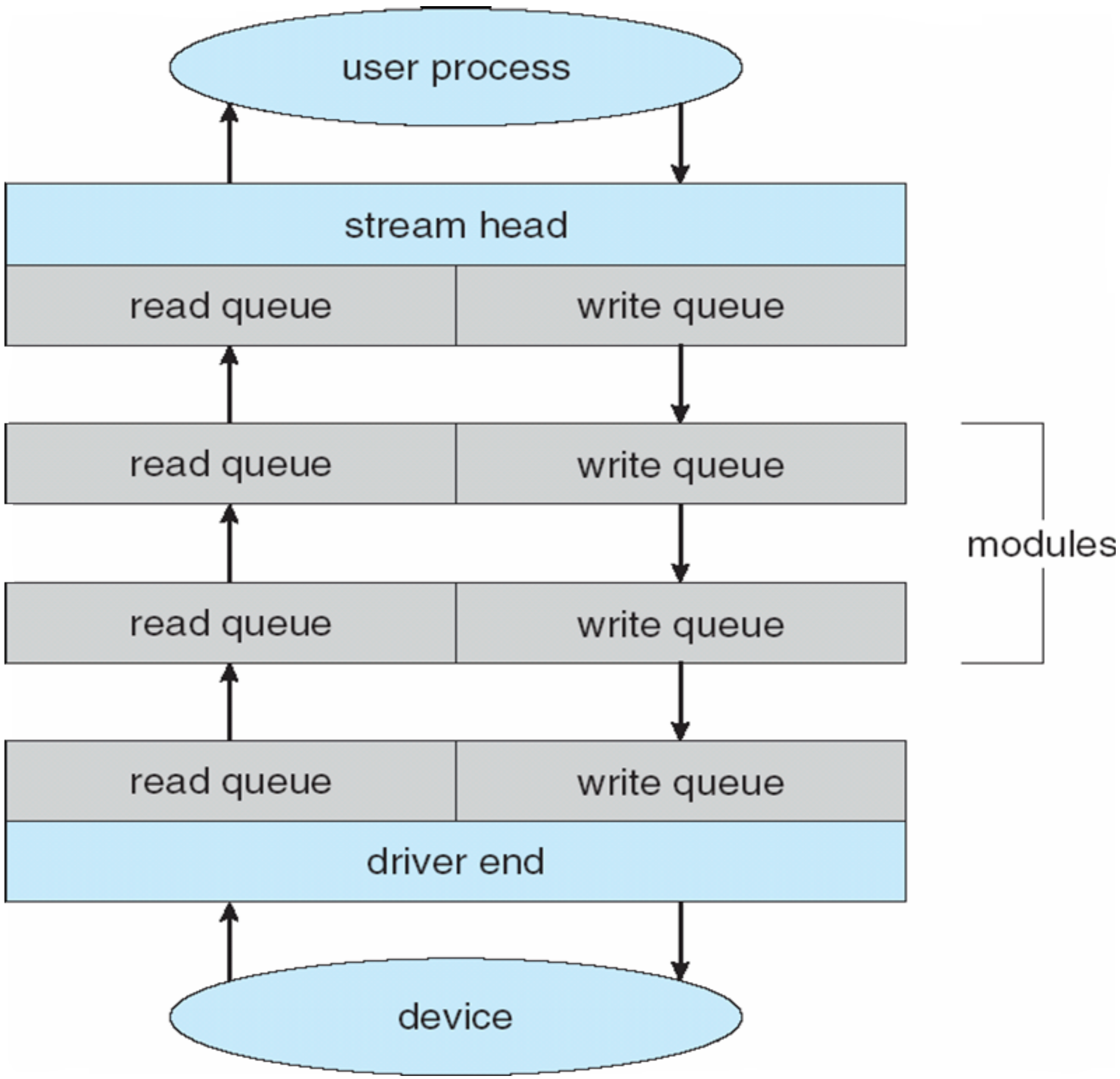
Life Cycle of An I/O Request



Streams

- Stream is a full-duplex communication channel between a user-level process and a device in Unix systems
- A stream consists of:
  - **stream head** interfaces with the user process
  - **driver end** interfaces with the device
  - zero or more stream modules between them (stacked)
    - each module contains a read queue and a write queue
- Message passing is used to communicate between queues
  - asynchronous internally, synchronous for user interface

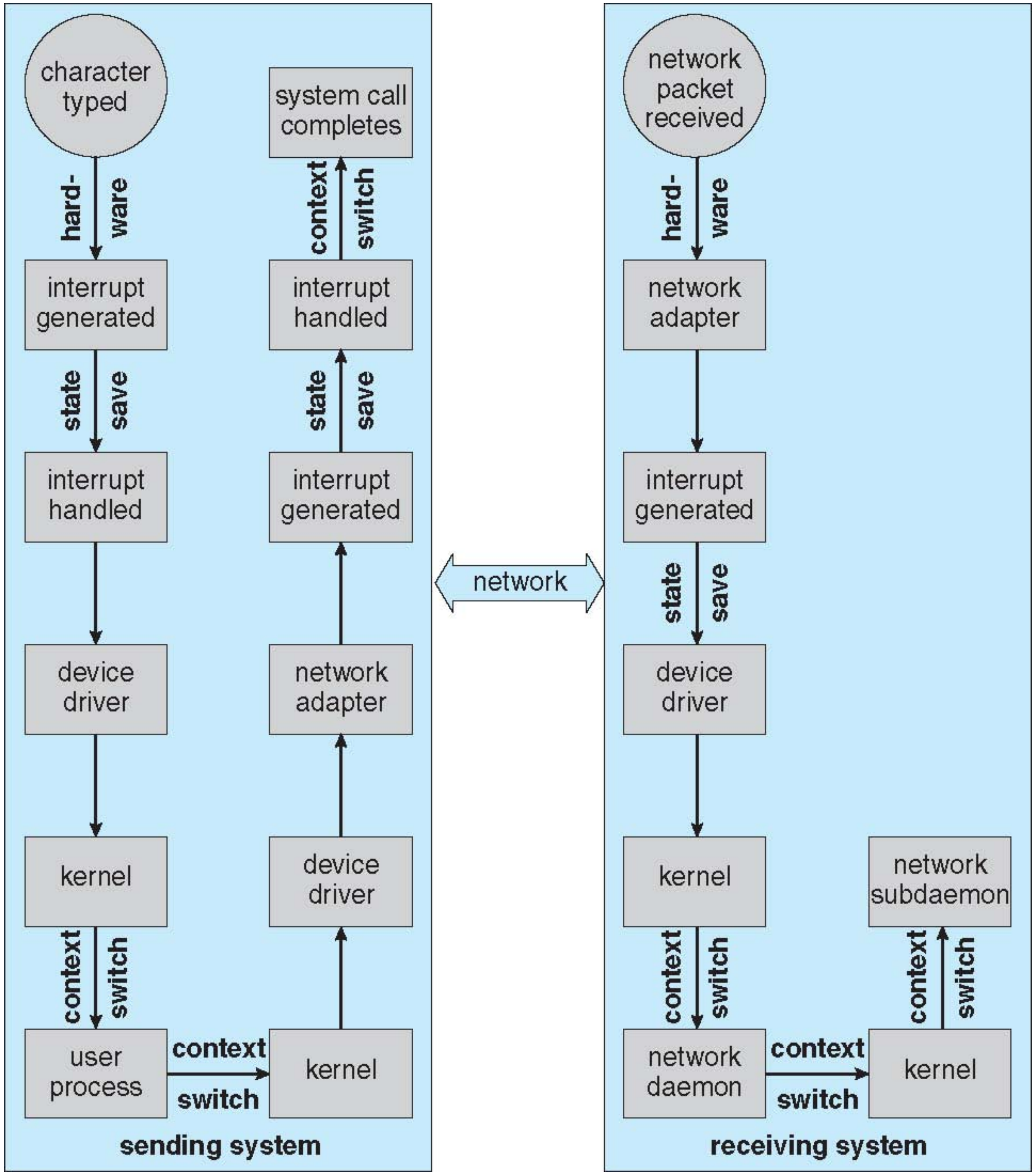
Streams Structure



Performance

- I/O is a major factor in system performance:
  - CPU to execute device driver, kernel I/O code
  - context switches due to interrupts
  - data buffering and copying
    - network traffic especially stressful

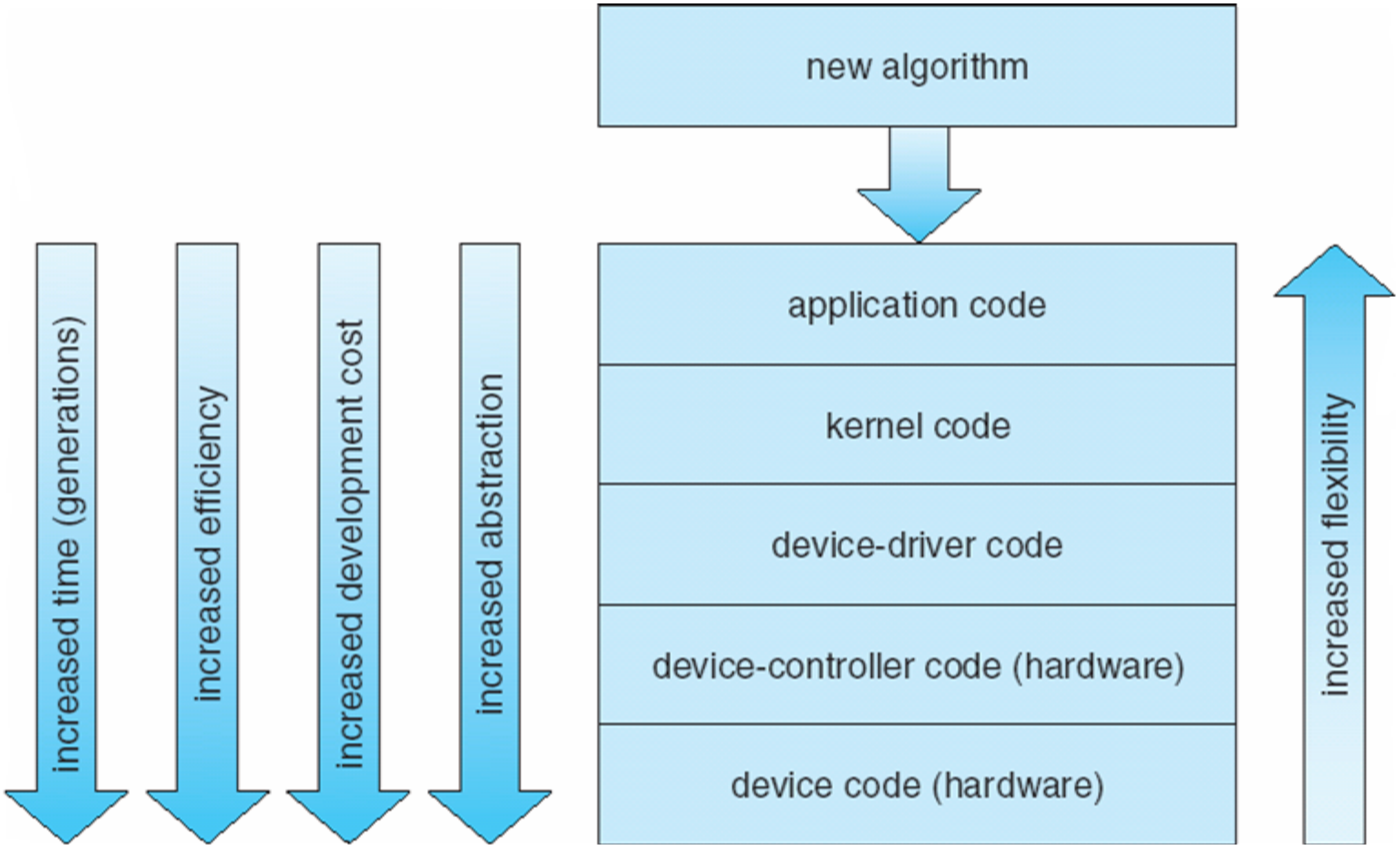
Intercomputer Communications



Performance

- To improve performance
  - reduce number of context switches
  - reduce data copying
  - reduce interrupts by using large transfers, smart controllers, polling
  - use DMA
  - use smarter hardware devices
  - move user processes to kernel threads

Device-Functionality Progression



End of Chapter 13