# Chap. 13) I/O Systems

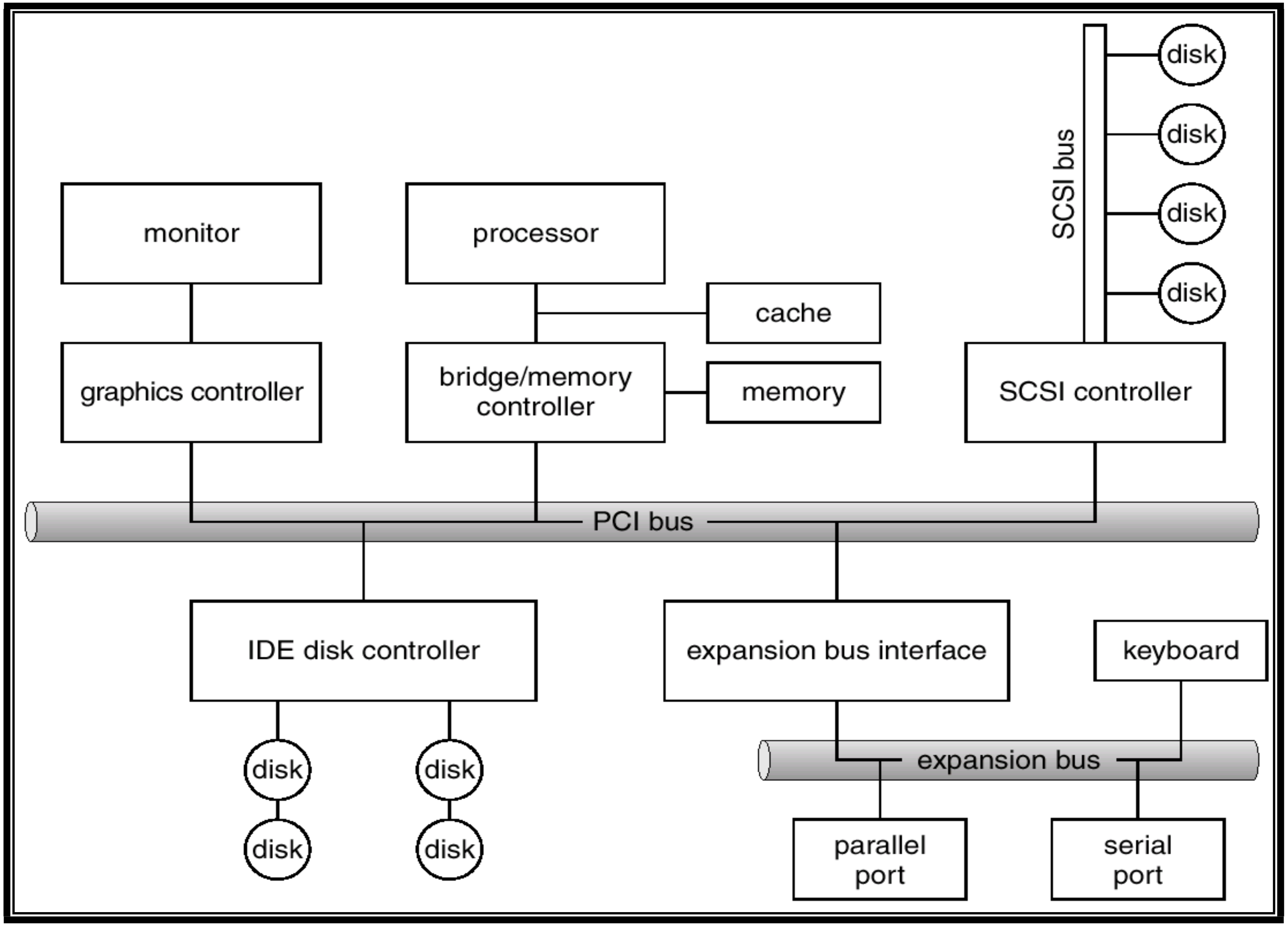
경희대학교  컴퓨터공학과

방 재 훈

# I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - ✓ Port
  - ✓ Bus (daisy chain or shared direct access)
  - ✓ Controller (host adapter)
- I/O instructions control devices
- Devices have addresses, used by
  - ✓ Direct I/O instructions
  - ✓ Memory-mapped I/O

# A Typical PC Bus Structure



# Device Controller

- Device controller (or host adapter)
  - ✓ I/O devices have components:
    - Mechanical component
    - Electronic component
  - ✓ The electronic component is the device controller
    - May be able to handle multiple devices
  - ✓ Controller’s tasks
    - Convert serial bit stream to block of bytes
    - Perform error correction as necessary
    - Make available to main memory

- Use special I/O instructions to an I/O port address

| 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) |

# Memory-Mapped I/O

- The device control registers are mapped into the address space of the processor
  - ✓ The CPU executes I/O requests using the standard data transfer instructions
- I/O device drivers can be written entirely in C
- No special protection mechanism is needed to keep user processes from performing I/O
  - ✓ Can give a user control over specific devices but not others by simply including the desired pages in its page table
- Reading a device register and testing its value is done with a single instruction
- Memory-mapped regions should be uncacheable
- Memory-mapped device register is vulnerable to accidental modification through the use of incorrect pointers
  - ✓ Protected memory helps to reduce this risk

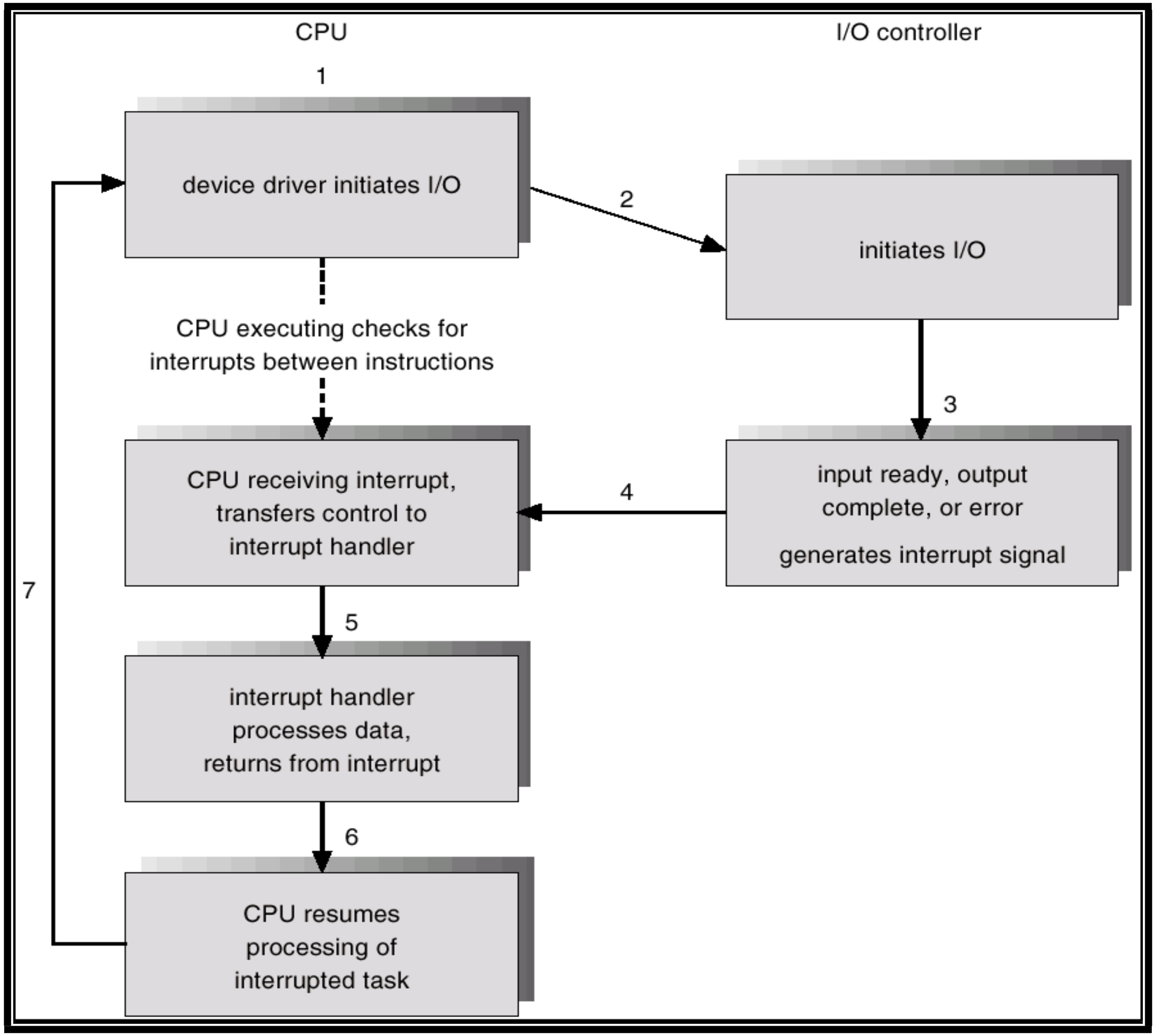
# Polling vs. Interrupts

- Polled I/O
  - ✓ CPU asks (“polls”) devices if need attention
    - ready to receive a command
    - command status, etc.
  - ✓ Advantages
    - Simple
    - Software is in control
    - Efficient if CPU finds a device to be ready soon
  - ✓ Disadvantages
    - Inefficient in non-trivial system (high CPU utilization)
    - Low priority devices may never be serviced

# Polling vs. Interrupts (Cont'd)

- Interrupt-driven I/O
  - ✓ I/O devices request interrupt when need attention
  - ✓ Interrupt service routines specific to each device are invoked
  - ✓ Interrupts can be shared between multiple devices
  - ✓ Advantages
    - CPU only attends to device when necessary
    - More efficient than polling in general
  - ✓ Disadvantages
    - Excess interrupts slow (or prevent) program execution
    - Overheads (may need 1 interrupt per byte transferred)

# Interrupt-Driven I/O Cycle



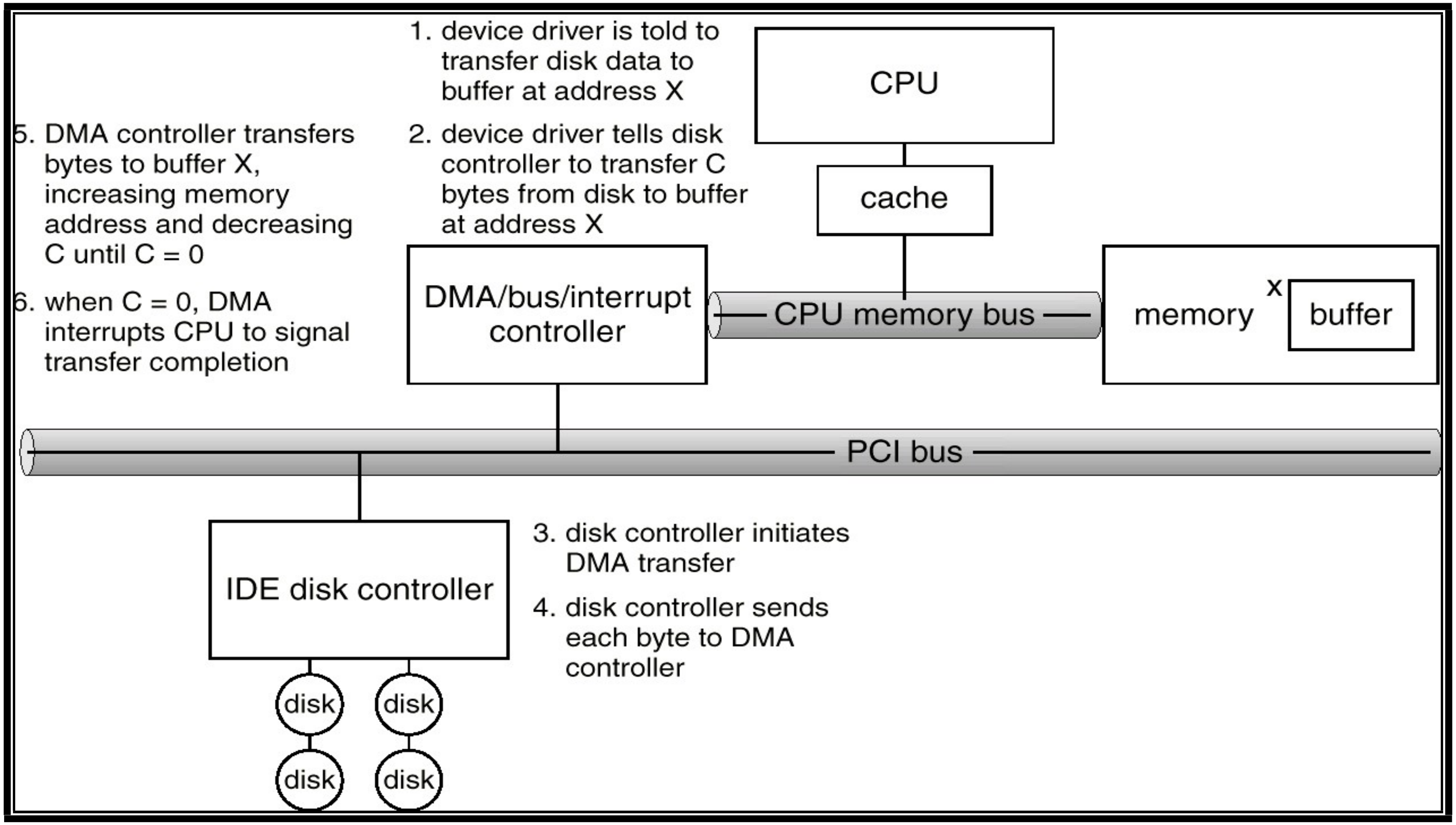
# Intel Pentium Processor Event-Vector Table

| 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 |
| 19Ð31 | (Intel reserved, do not use) |
| 32Ð255 | maskable interrupts |

# Direct Memory Access

- Used to avoid programmed I/O for large data movement
  - ✓ Programmed I/O vs. DMA
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

# Six Step Process to Perform DMA Transfer



- DMA modes
  - (1) Cycle stealing
    - The DMA controller sneaks in and steals an occasional bus cycle from the CPU once in a while, delaying it slightly
  - (2) Burst mode
    - The DMA controller acquires the bus, issues a series of transfers, then releases the bus
    - More efficient than cycle stealing: acquiring the bus takes time and multiple words can be transferred for the price of one bus acquisition
    - It can block the CPU and other devices too long

# DMA (Cont'd)

- Addressing in DMA
  - (1) Physical address
    - OS converts the virtual address of the intended memory buffer into a physical address and writes it into DMA controller's address register
  - (2) Virtual address
    - The DMA controller must use the MMU to have the virtual-to-physical translation done
    - Not common: only when the MMU is part of the memory rather than part of the CPU
  - ✓ In any case, the target memory region should be pinned (not paged out) during DMA

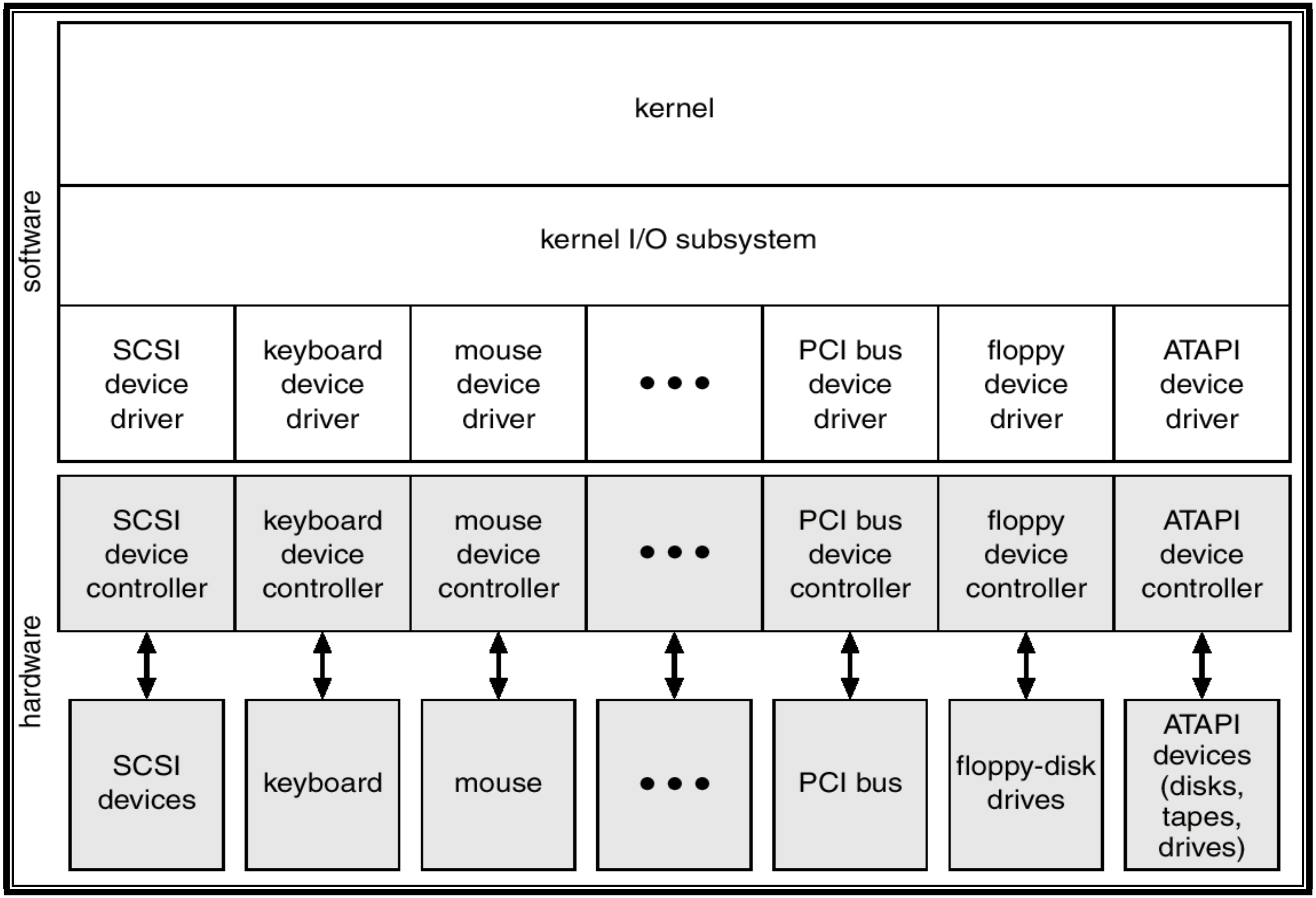
# Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes

- Device-driver layer hides differences among I/O controllers from kernel

- Devices vary in many dimensions
  - ✓ Character-stream or block
  - ✓ Sequential or random-access
  - ✓ Sharable or dedicated
  - ✓ Speed of operation
  - ✓ read-write, read only, or write only

# A Kernel I/O Structure



# Characteristics of I/O Devices

| aspect | variation | example |
|---|---|---|
| data-transfer mode | character<br>block | terminal<br>disk |
| access method | sequential<br>random | modem<br>CD-ROM |
| transfer schedule | synchronous<br>asynchronous | tape<br>keyboard |
| sharing | dedicated<br>sharable | tape<br>keyboard |
| device speed | latency<br>seek time<br>transfer rate<br>delay between operations | |
| I/O direction | read only<br>write only<br>readÐwrite | CD-ROM<br>graphics controller<br>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
- Character devices include keyboards, mice, serial ports
  - ✓ Commands include `get, put`
  - ✓ Libraries layered on top allow line editing

# Network Devices

- Varying enough from block and character to have own interface

- Unix and Windows NT/9*i*/2000 include socket interface
  - ✓ Separates network protocol from network operation
  - ✓ Includes `select` functionality

- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

# Clocks and Timers

- Provide current time, elapsed time, timer
- If programmable interval time used for timings, periodic interrupts
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers

# Blocking and Nonblocking I/O

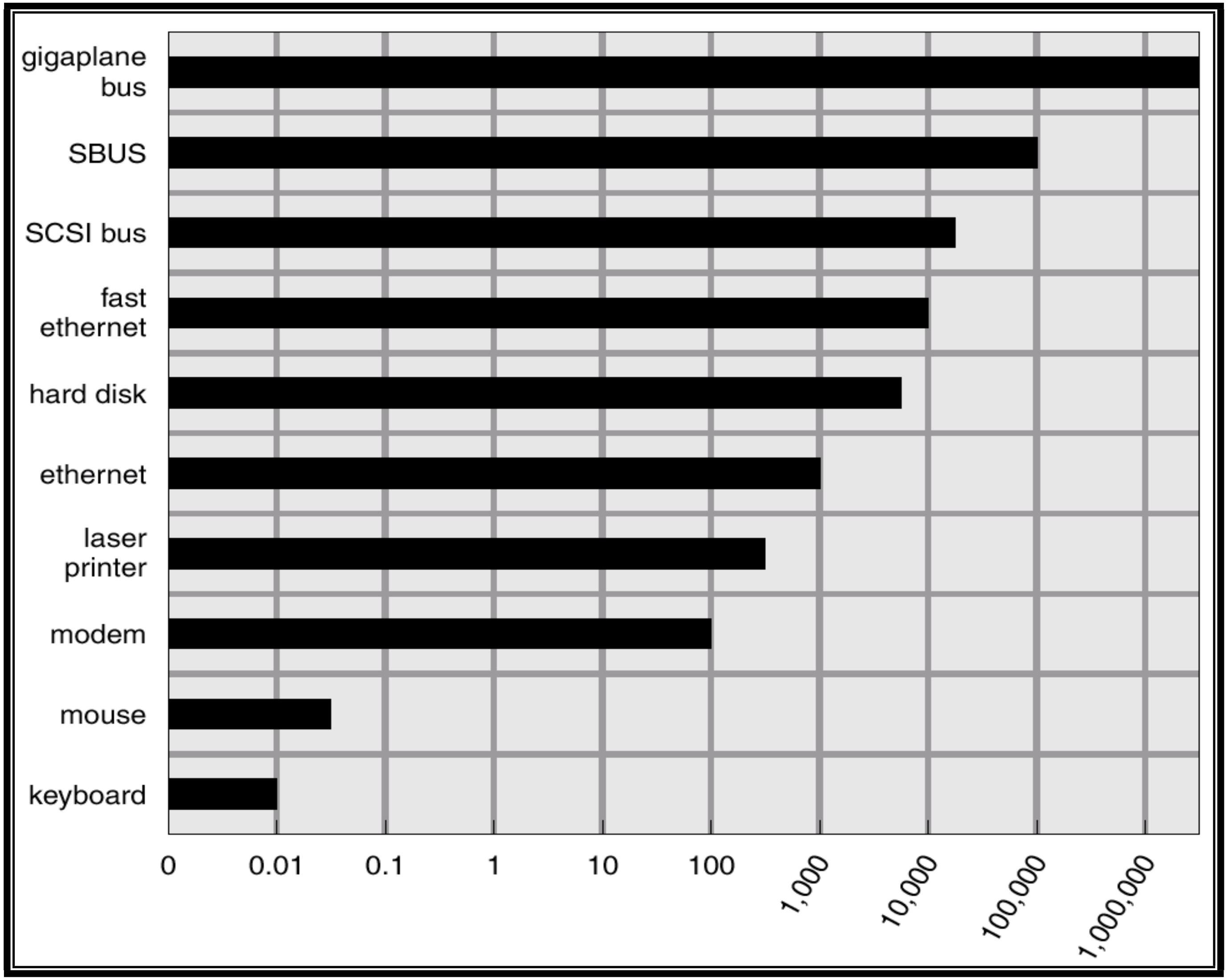
- Blocking - process suspended until I/O completed
  - ✓ Easy to use and understand
  - ✓ Insufficient for some needs

- Nonblocking - I/O call returns as much as available
  - ✓ User interface, data copy (buffered I/O)
  - ✓ Implemented via multi-threading
  - ✓ Returns quickly with count of bytes read or written

# Kernel I/O Subsystem

- Scheduling
  - ✓ Some I/O request ordering via per-device queue
  - ✓ Some OSs try fairness
- 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”

# Sun Enterprise 6000 Device-Transfer Rates



# Kernel I/O Subsystem

- Caching - fast memory holding copy of data
  - ✓ Always just a copy
  - ✓ Key to performance

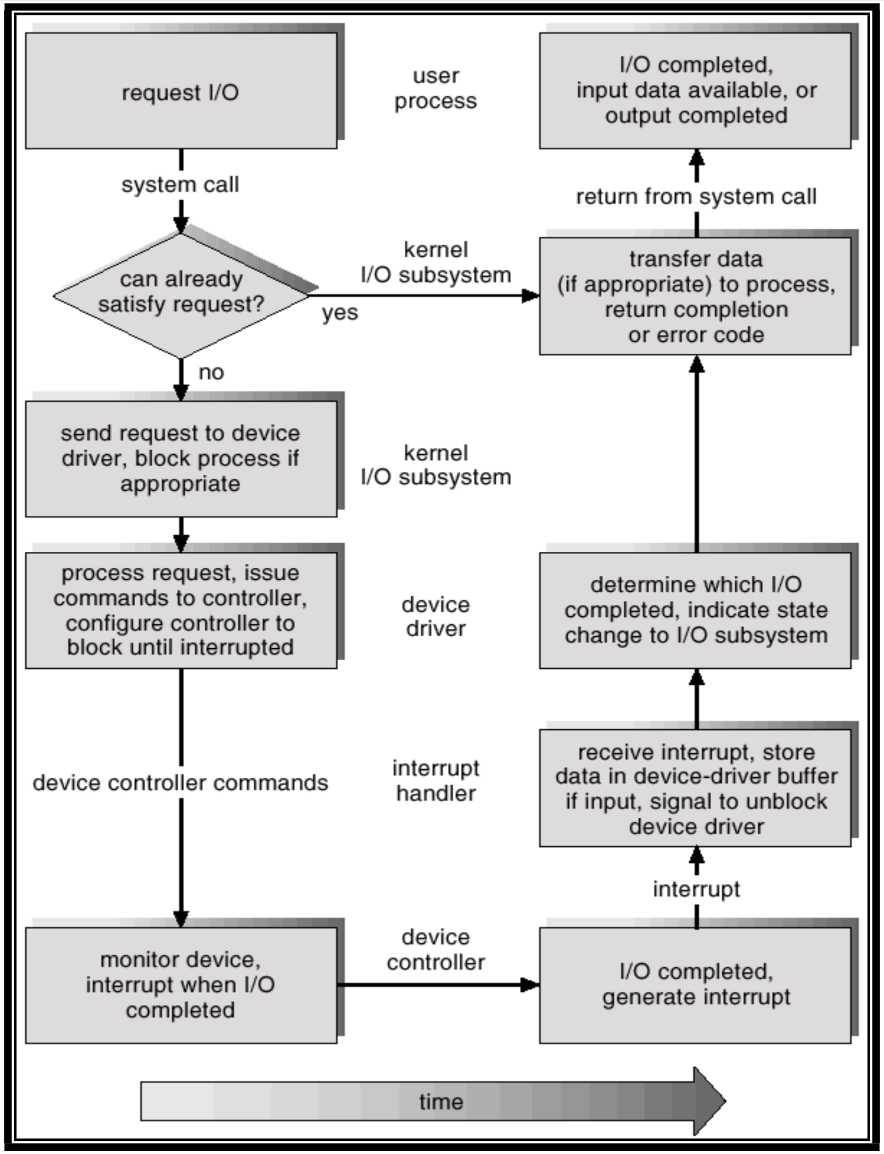
- Spooling - hold output for a device
  - ✓ If device can serve only one request at a time
  - ✓ i.e., Printing

- Device reservation - provides exclusive access to a device
  - ✓ System calls for allocation and deallocation
  - ✓ Watch out for deadlock

# 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

# Life Cycle of An I/O Request



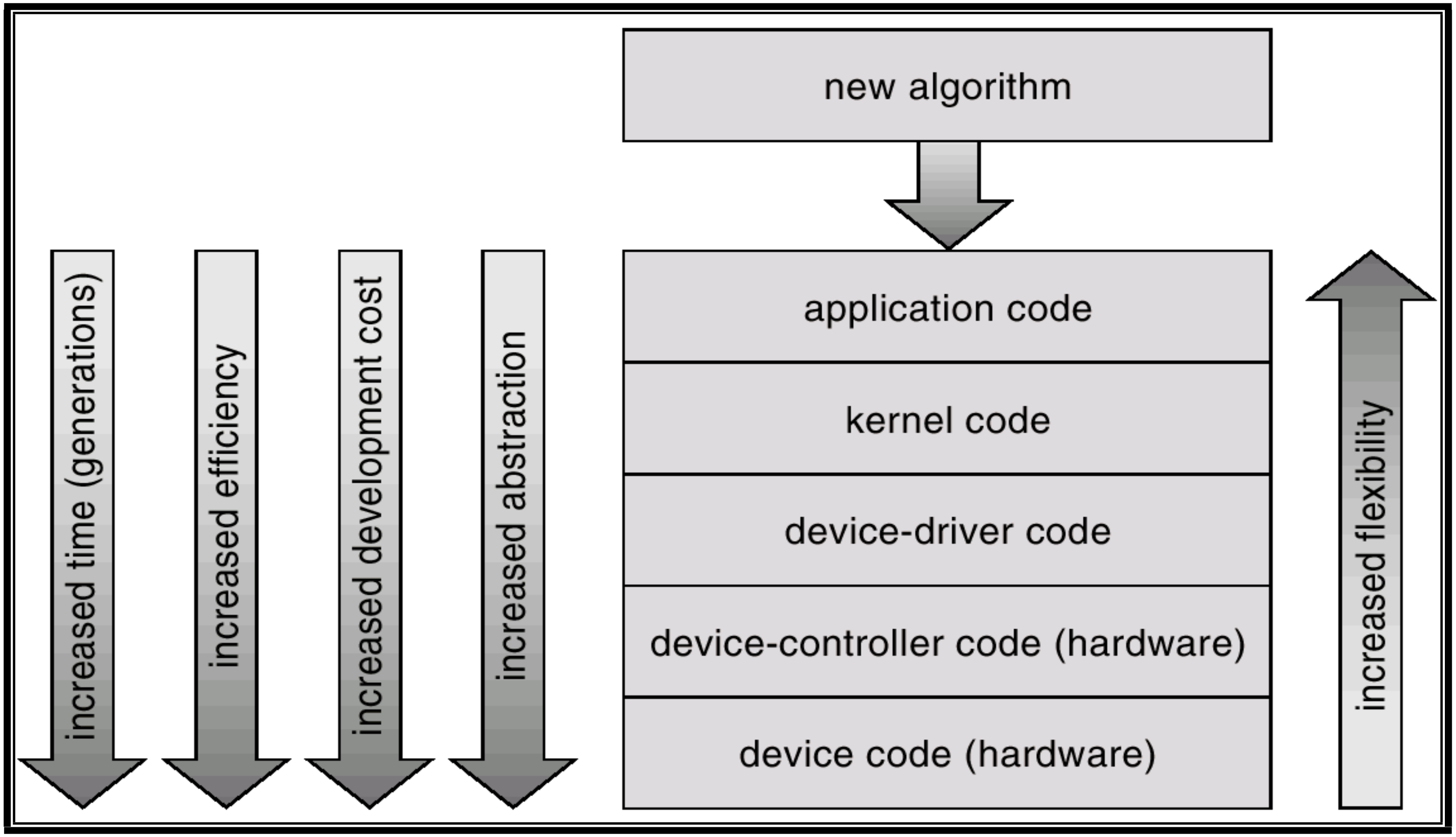
# Performance

- I/O a major factor in system performance:
  - ✓ Demands CPU to execute device driver, kernel I/O code
  - ✓ Context switches due to interrupts
  - ✓ Data copying
  - ✓ 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

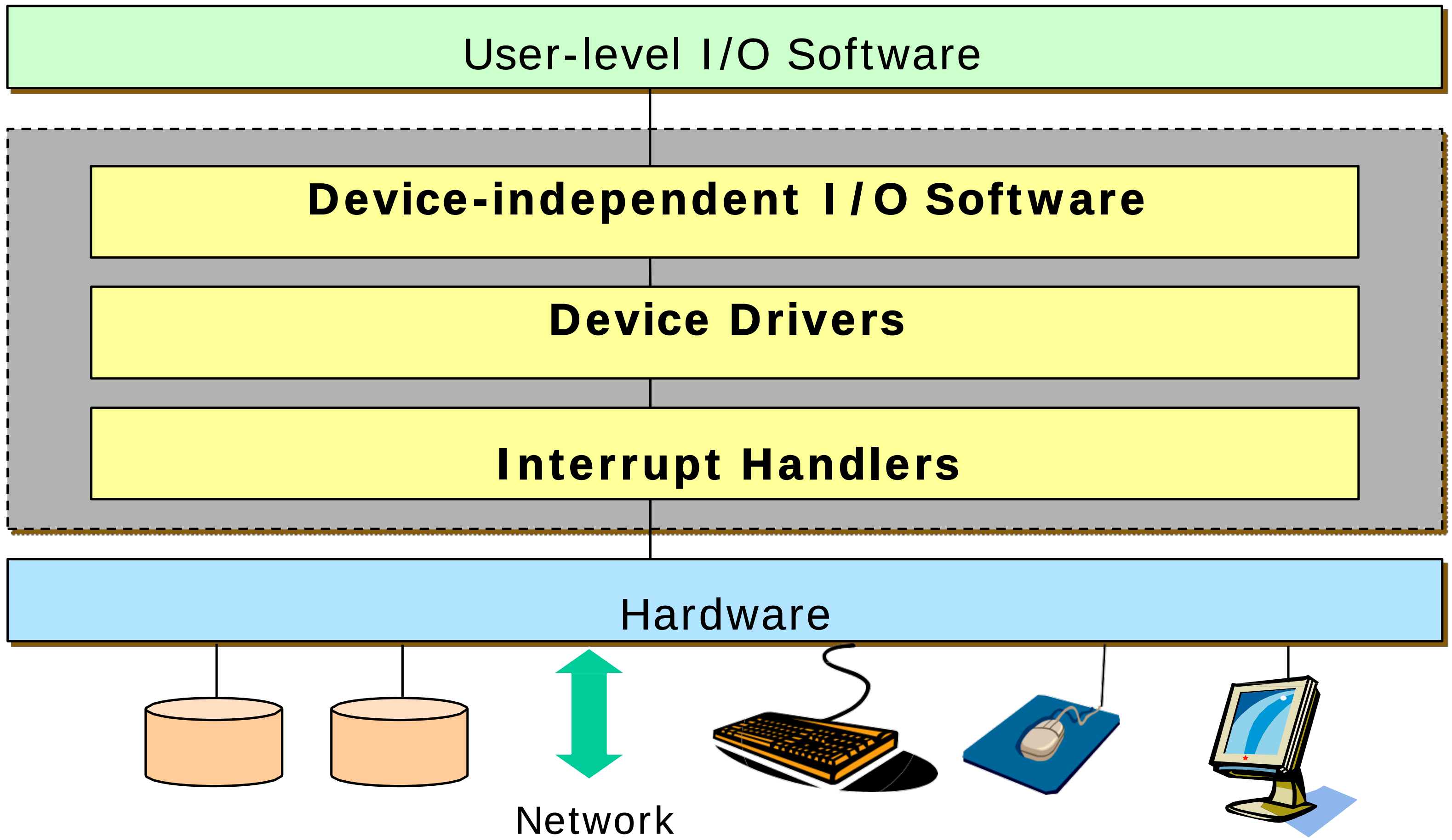
# Device-Functionality Progression



# Goals of I/O Software

- Goals
  - ✓ Device independence
    - Programs can access any I/O device without specifying device in advance
  - ✓ Uniform naming
    - Name of a file or device should simply be a string or an integer
  - ✓ Error handling
    - Handle as close to the hardware as possible
  - ✓ Synchronous vs. asynchronous
    - blocked transfers vs. interrupt-driven
  - ✓ Buffering
    - Data coming off a device cannot be stored in final destination
  - ✓ Sharable vs. dedicated devices
    - Disks vs. tape drives
    - Unsharable devices introduce problems such as deadlocks

# I/O Software Layers



- Handling interrupts

Critical actions
: Acknowledge an interrupt to the PIC.
: Reprogram the PIC or the device controller.
: Update data structures accessed by both the device and the processor.

Reenable interrupts

Noncritical actions
: Update data structures that are accessed only by the processor.
(e.g., reading the scan code from the keyboard)

Return from interrupts

Noncritical deferred actions
: Actions may be delayed.
: Copy buffer contents into the address space of some process (e.g., sending the keyboard line buffer to the terminal handler process).
Bottom half (Linux)

# Device Drivers

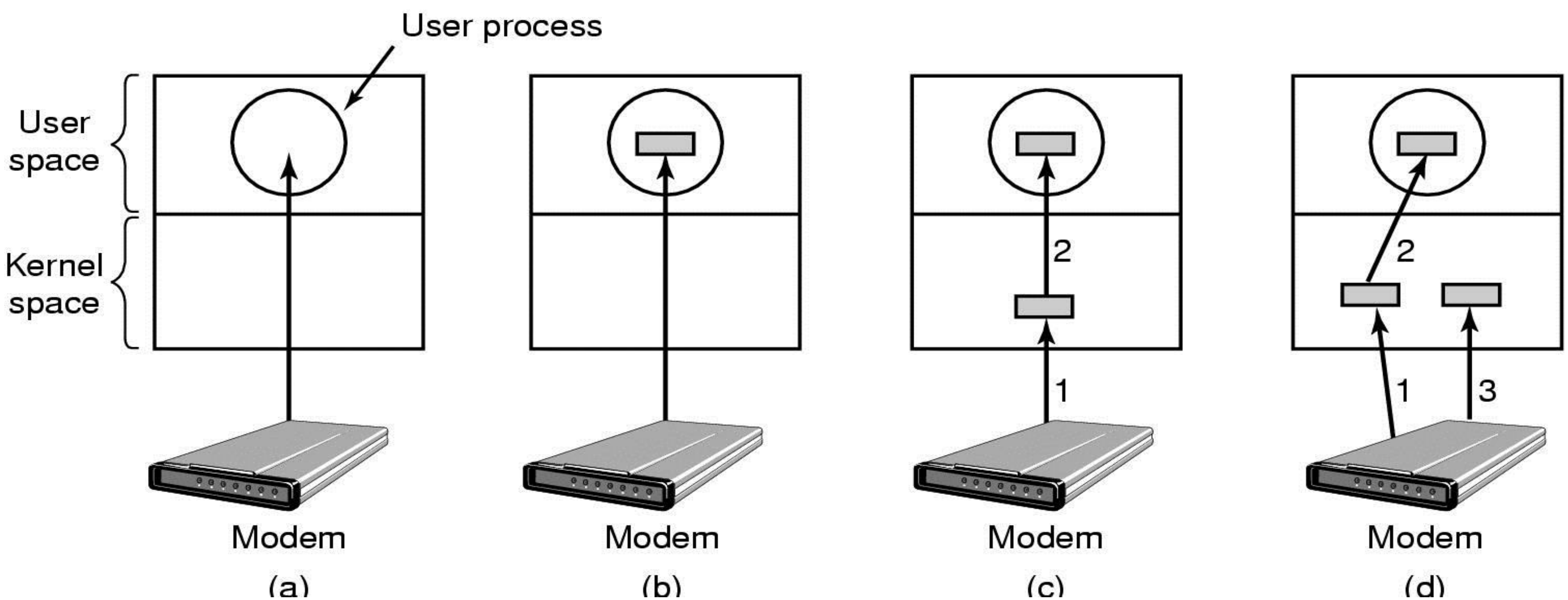
- Device drivers
  - ✓ Device-specific code to control each I/O device interacting with device-independent I/O software and interrupt handlers
  - ✓ Requires to define a well-defined model and a standard interface of how they interact with the rest of the OS
  - ✓ Implementing device drivers:
    - Statically linked with the kernel
    - Selectively loaded into the system during boot time
    - Dynamically loaded into the system during execution (especially for hot pluggable devices)

# Device-Independent I/O Software

- Uniform interfacing for device drivers
  - ✓ In Unix, devices are modeled as special files
    - They are accessed through the use of system calls such as open(), read(), write(), close(), ioctl(), etc.
    - A file name is associated with each device
  - ✓ Major device number locates the appropriate driver
    - Minor device number (stored in i-node) is passed as a parameter to the driver in order to specify the unit to be read or written
  - ✓ The usual protection rules for files also apply to I/O devices

# Device-Independent I/O Software (Cont'd)

- Buffering
  - ✓ (a) Unbuffered
  - ✓ (b) Buffered in user space
  - ✓ (c) Buffered in the kernel space
  - ✓ (d) Double buffering in the kernel



# Device-Independent I/O Software (Cont'd)

- Error reporting
  - ✓ Many errors are device-specific and must be handled by the appropriate driver, but the framework for error handling is device independent
  - ✓ Programming errors vs. actual I/O errors
  - ✓ Handling errors
    - Returning the system call with an error code
    - Retrying a certain number of times
    - Ignoring the error
    - Killing the calling process
    - Terminating the system

# Device-Independent I/O Software (Cont'd)

- Allocating and releasing dedicated devices
  - ✓ Some devices cannot be shared
  - (1) Require processes to perform open()'s on the special files for devices directly
    - The process retries if open() fails
  - (2) Have special mechanisms for requesting and releasing dedicated devices
    - An attempt to acquire a device that is not available blocks the caller

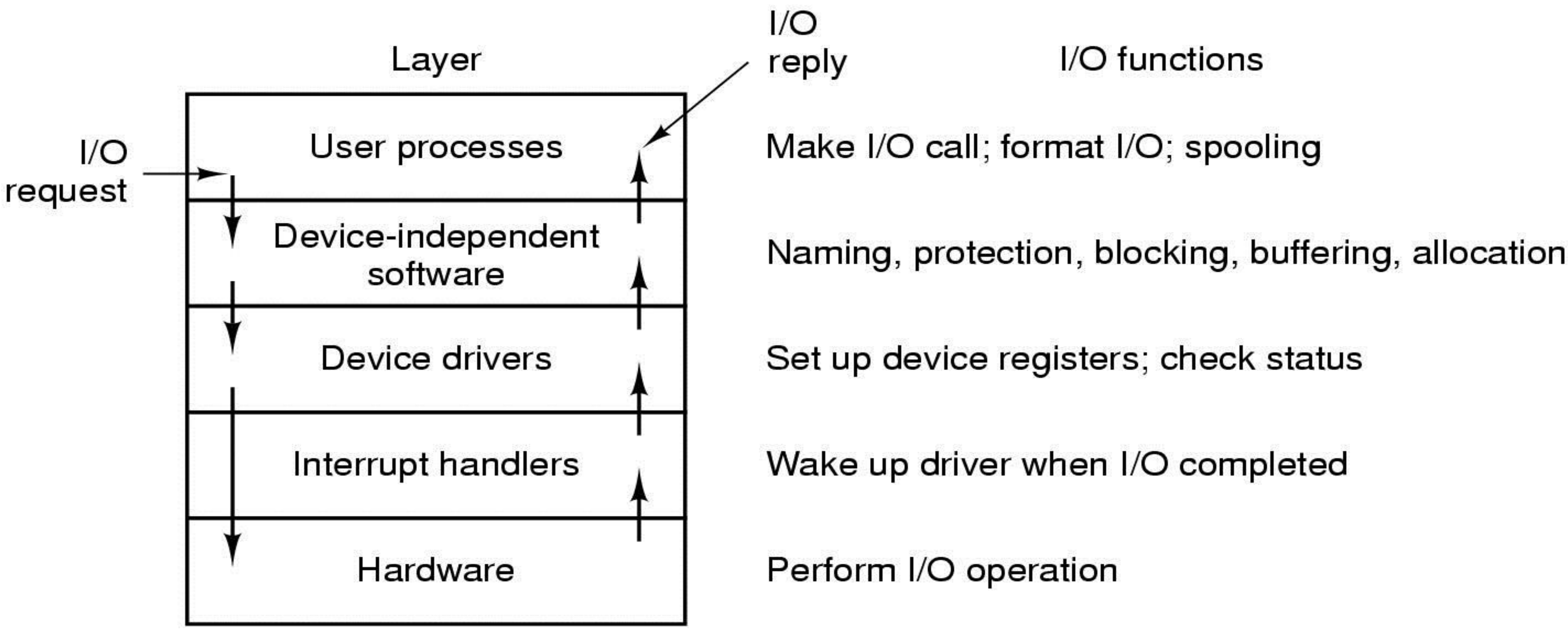
- Device-independent block size
  - ✓ Treat several sectors as a single logical block
  - ✓ The higher layers only deal with abstract devices that all use the same block size

# User-Space I/O Software

- Provided as a library
  - ✓ Standard I/O library in C
    - fopen() vs. open()

- Spooling
  - ✓ A way of dealing with dedicated I/O devices in a multiprogramming system
  - ✓ Implemented by a daemon and a spooling directory
  - ✓ Printers, network file transfers, USENET news, mails, etc.

# I/O Systems Layers



# Summary

- I/O hardware
  - ✓ I/O instructions control I/O devices
    - Direct I/O vs. Memory-mapped I/O
  - ✓ I/O events notification
    - Polled I/O vs. Interrupt-driven I/O
  - ✓ DMA (Direct Memory Access)

- I/O software
  - ✓ Application I/O interface: I/O system call
    - open, read, write, close, ioctl, …
  - ✓ Kernel I/O structure
    - Device-independent I/O subsystem
    - Device drivers (device specific)
    - Interrupt handlers (device specific)

# Summary (Cont'd)

- Performance issues
  - ✓ Reduce number of context switches
  - ✓ Reduce data copying
  - ✓ Reduce interrupts
  - ✓ Use DMA
  - ✓ Balance CPU, memory, bus, and I/O performance for higher system-level throughtput