PART 4. STORAGE MANAGEMENT

# Chapter 13: I/O Systems

**WHAT'S AHEAD:**

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

**WE AIM:**

- Explore the structure of an OS's 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

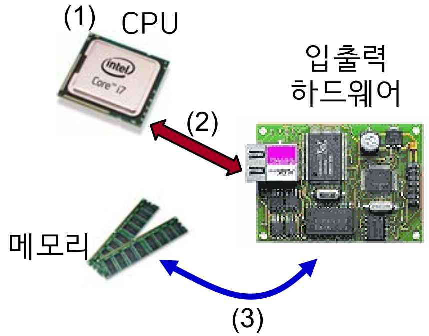
Note: These lecture materials are based on the lecture notes prepared by the authors of the book titled *Operating System Concepts*, 9e (Wiley)

핵심•요점

# 입출력 기법과 입출력 시스템

## 전형적인 입출력 기법 유형

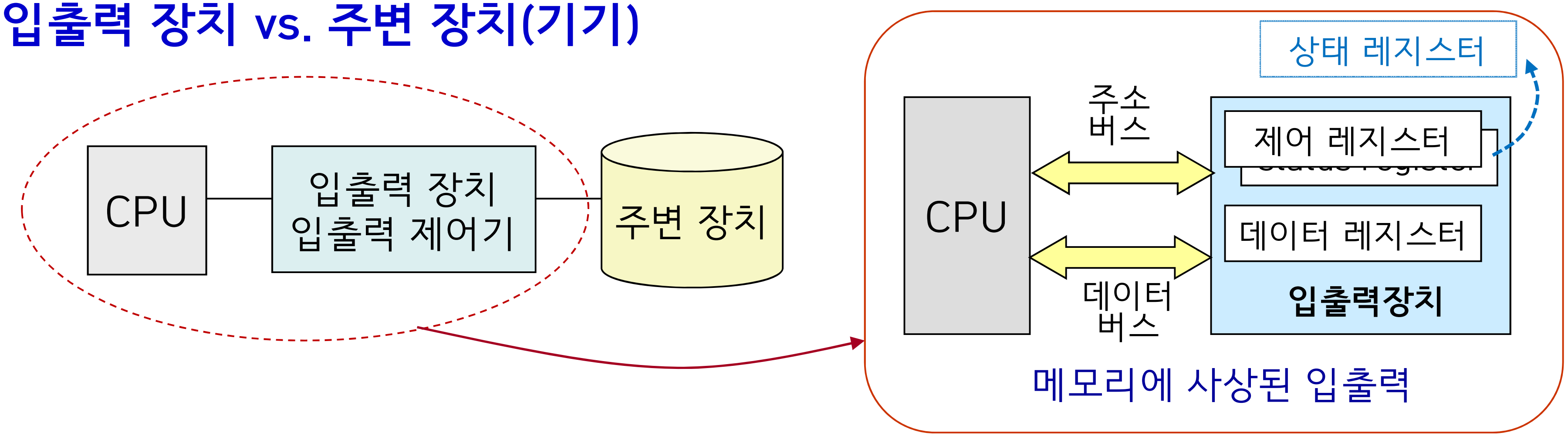
- ✓ 전적으로 프로그램에 의존 → **프로그램에 의한 입출력**(1)
- ✓ 인터럽트 하드웨어가 입출력 촉발 → **인터럽트에 의한 입출력**(2)
- ✓ CPU 도움 없이 → **직접 메모리 접근 (DMA)** (3)




## 입출력을 위한 주요 기능

- ✓ 목표 장치의 **초기화**
- ✓ 데이터 입출력 수행: **읽기/쓰기, 수신/송신**
- ✓ **상태** 정보 읽기, **제어** 정보 쓰기
- ✓ 입출력 **오류**의 검출 및 처리

## 입출력 장치 vs. 주변 장치(기기)

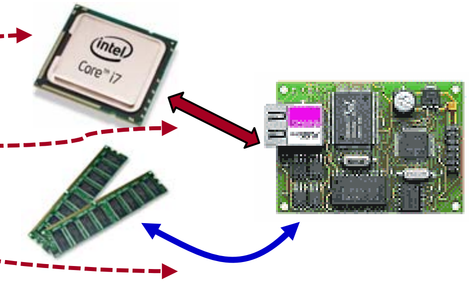


Core Ideas

# I/O Techniques & Systems

## Typical types of I/O techniques

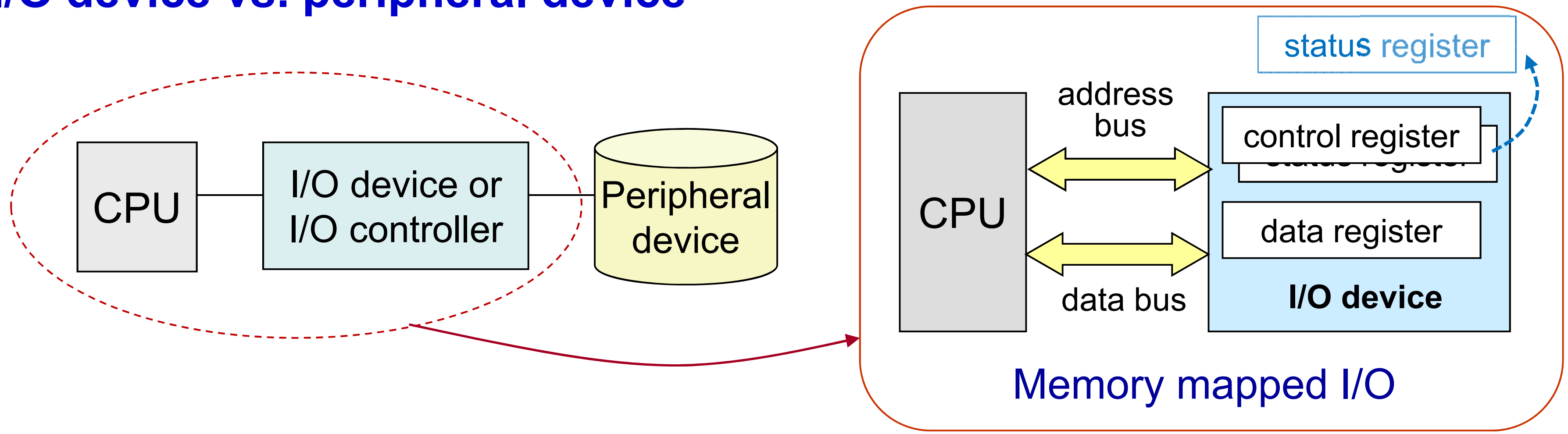
- ✓ The program does it all → Programmed I/O
- ✓ Interrupt hardware triggers I/O → Interrupt I/O
- ✓ CPU is left out → Direct memory access (DMA)



## Main functions for I/O

- ✓ Initialize the target device
- ✓ Perform data I/O such as read/write or send/receive
- ✓ Read status information and write control data
- ✓ Detect and handle I/O errors properly

## I/O device vs. peripheral device



# Overview

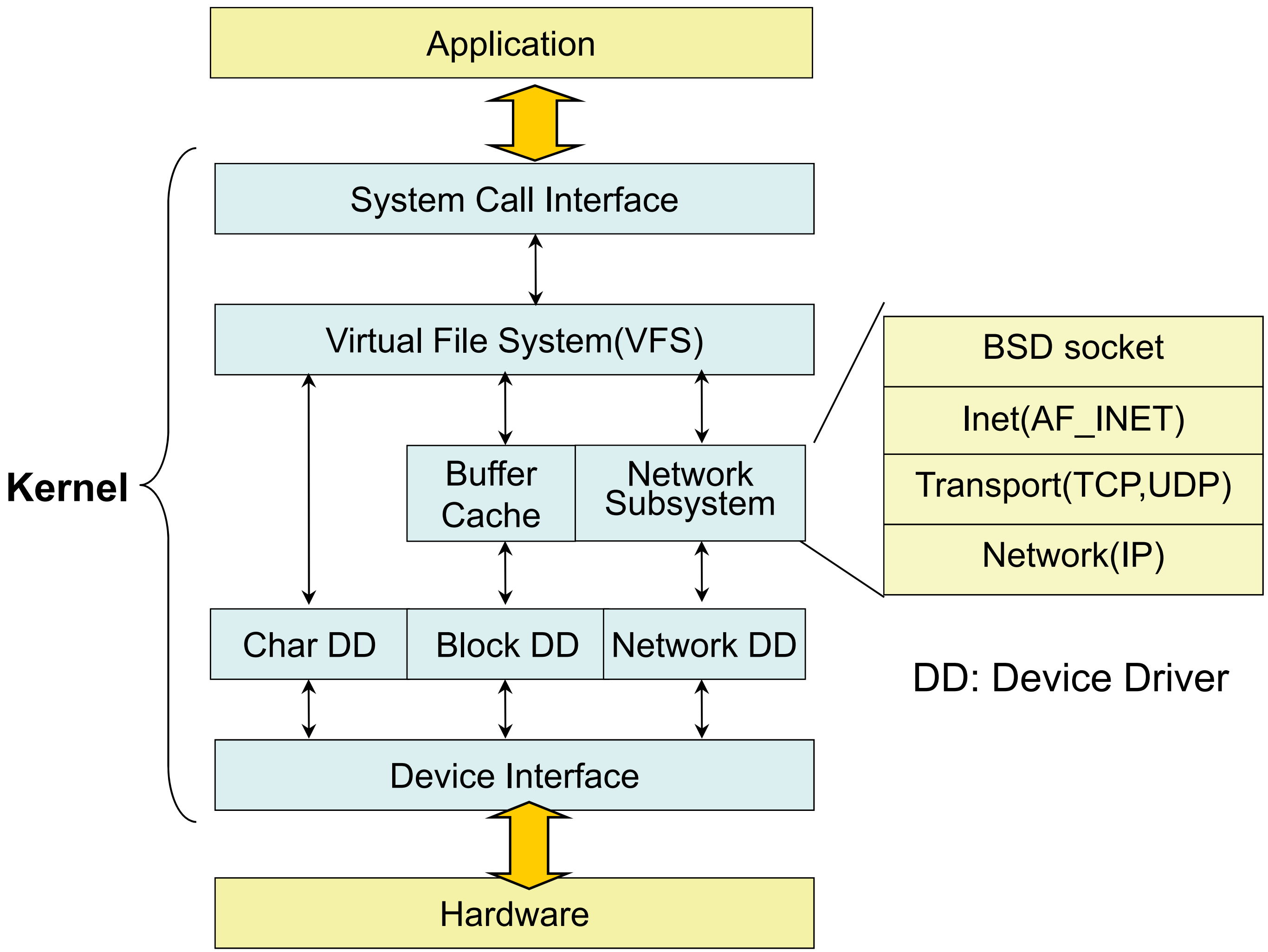
- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent

Caveat
**I/O device**
**≠ peripheral device**

- Ports, busses, device controllers connect to various devices

- Device drivers encapsulate device details
  - Present uniform device-access interface to I/O subsystem

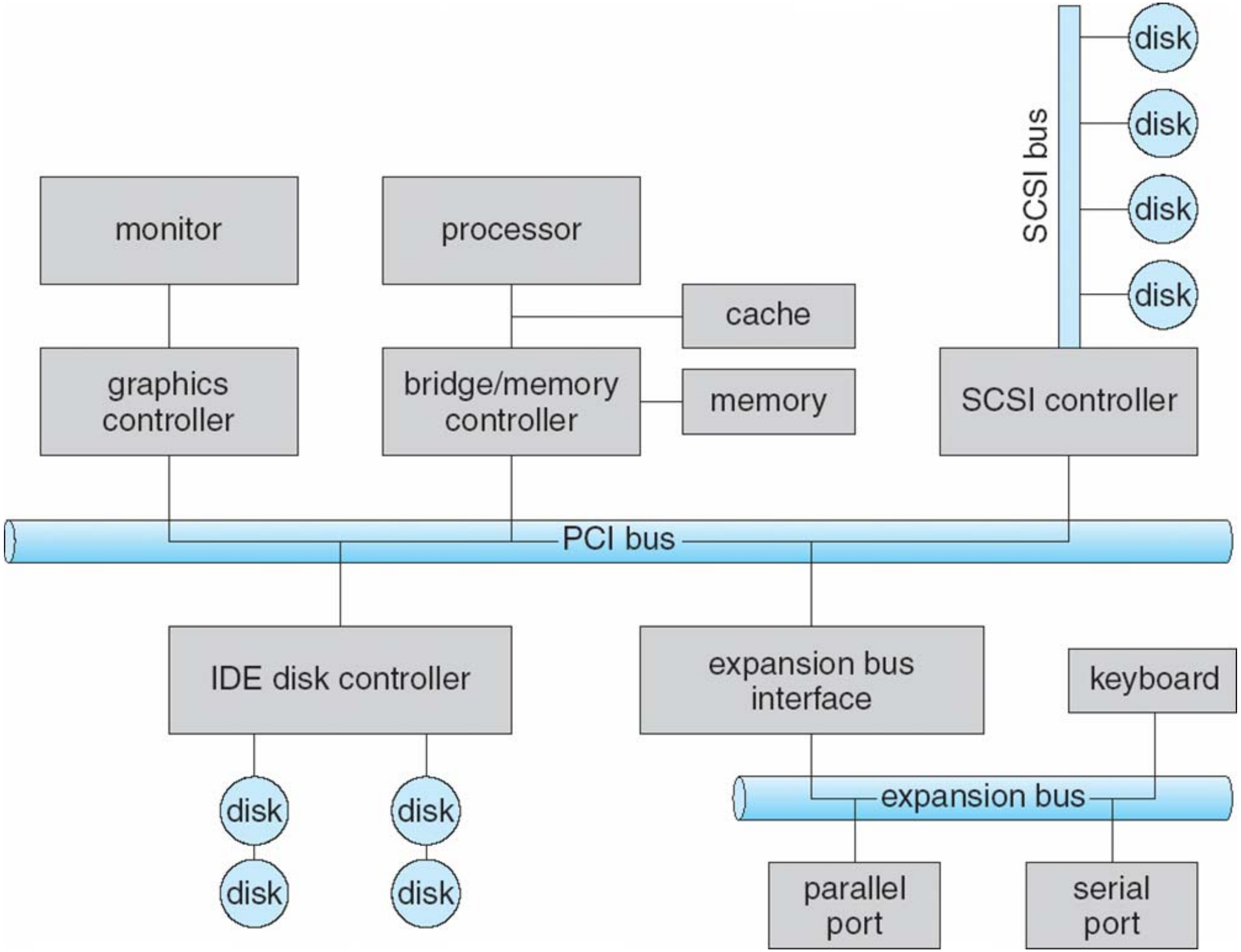
# Linux I/O Structure



# I/O Hardware

- Incredible variety of I/O devices
  - Storage
  - Communication, networking
  - Human-interface
  - Sensors
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus** - shared direct access
  - **Controller** (**host adapter**) – electronics that operate port, bus, device
    - 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



# Polling for Programmed I/O

- Programmed I/O
  - Direct connection between CPU and I/O devices
  - All I/O operations performed on CPU by a program
- Request for I/O
  - Read or input: check to see if data is available in I/O device by polling
  - Write or output: check to see if output device is ready by polling
- Example: Polling loop

```
repeat
    for 1 millisecond   /* input from keyboard */
         check i/o control register to see if a key has been pressed
             then
                input a character into a CPU data register
                process the character
    end for
    if we need to output to screen  then
        check i/o control register to see if screen is ready
            then output a character
    if we are printing  then
        check i/o control register to see if the printer is ready
            then output a character
until forever
```

# I/O Hardware (Cont.)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Isolated I/O: Use direct I/O instructions
    - Separate I/O address space
  - **Memory-mapped I/O**
    - No separate address space for I/O
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

# Device I/O Port and Memory Locations

Device I/O Port and Memory Locations
on Linux (partial)

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) |

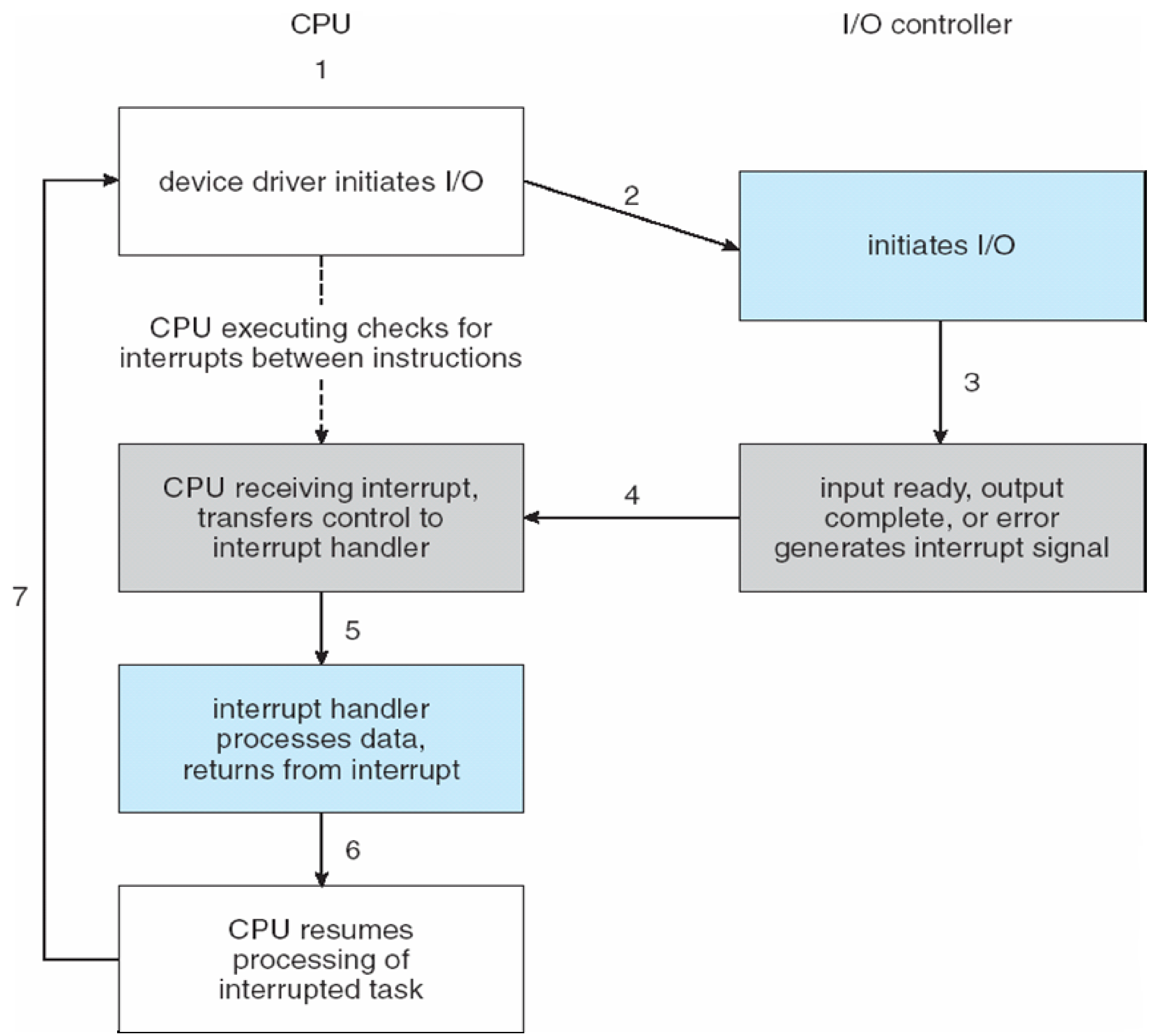
```
[root]# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
```

```
[root]# cat /proc/iomem
00000000-0009b7ff : System RAM
0009b800-0009ffff : reserved
000a0000-000bffff : Video RAM area
…
000f0000-000fffff : System ROM
00100000-3fedffff : System RAM
00100000-002d1daf : Kernel code
002d1db0-0038e2ff : Kernel data
…
d0100000-d02fffff : PCI Bus #02
```

# Interrupts

- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently? → Use hardware mechanism (i.e., interrupt)
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining if more than one device at same interrupt number

# 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 |
| 19–31 | (Intel reserved, do not use) |
| 32–255 | maskable interrupts |

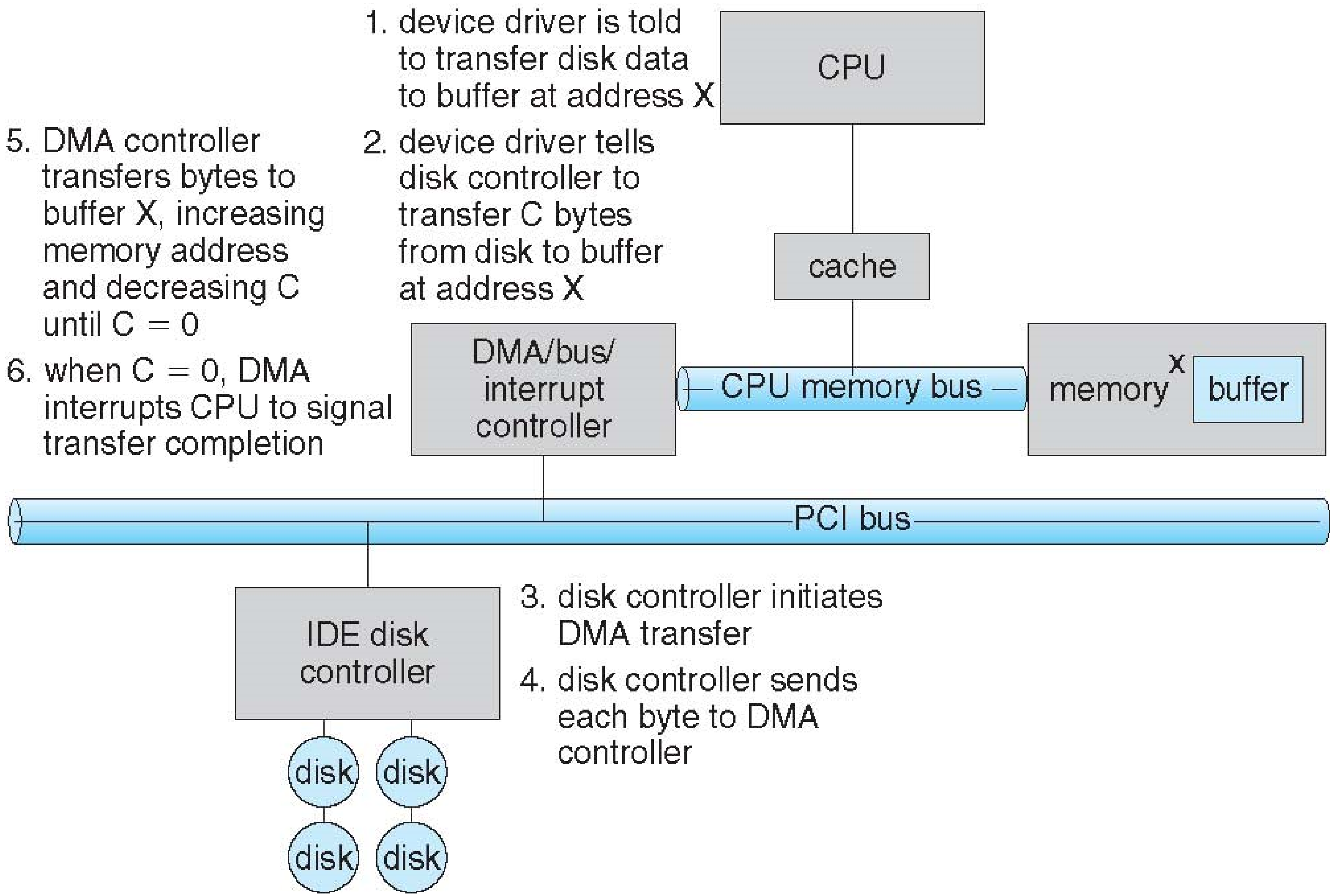
# Interrupts (Cont.)

- 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

# 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
- 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
  - When done, interrupts to signal completion

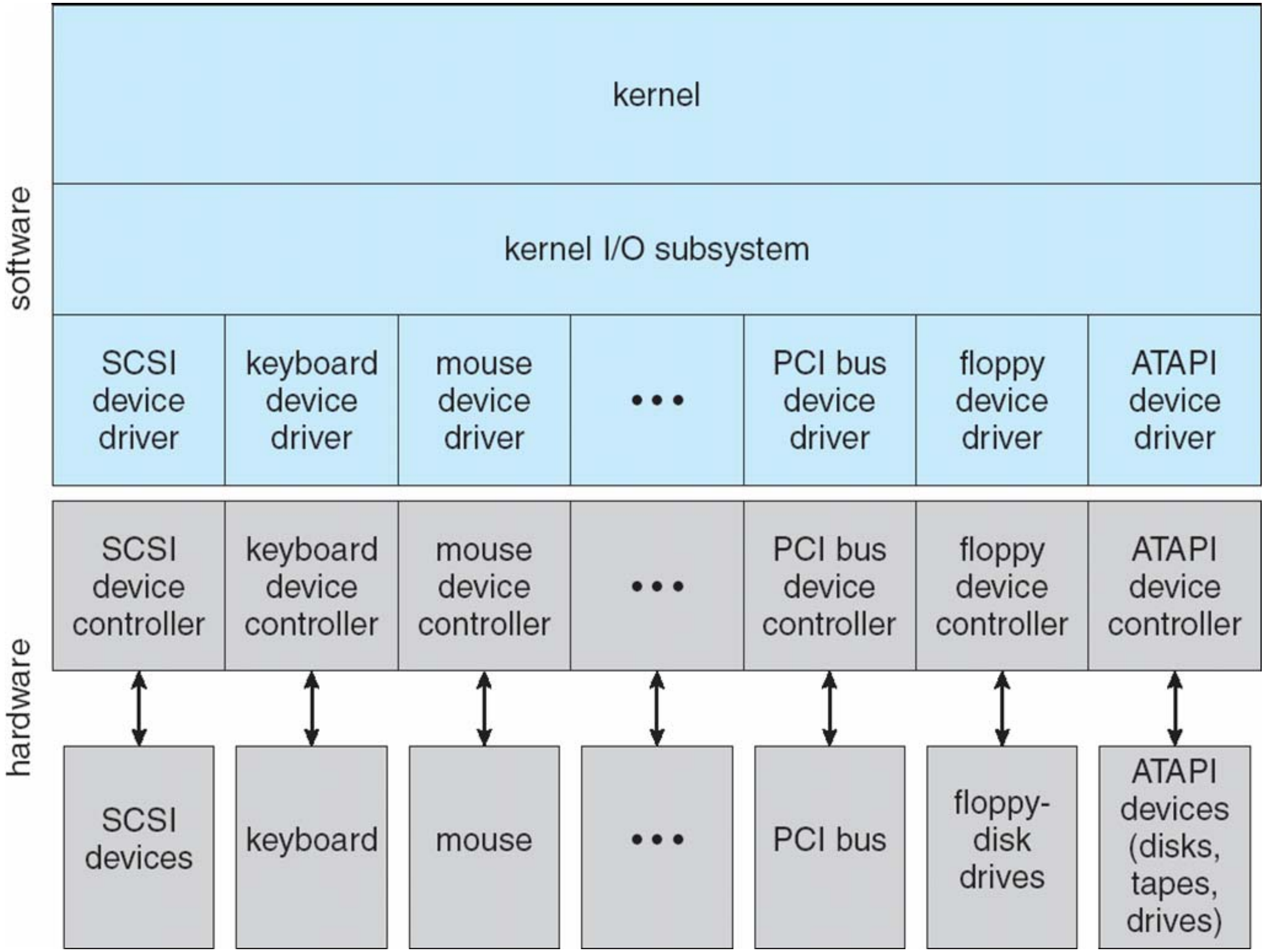
# Steps in a DMA Transfer



# 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
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- 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**

# 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>read–write | CD-ROM<br>graphics controller<br>disk |

# Characteristics of I/O Devices (Cont.)

- Subtleties of devices handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix **`ioctl()`** call to send arbitrary bits to a device control register and data to device data register

# Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - **Raw I/O**, **direct I/O**, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- 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*x*/2000 include **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

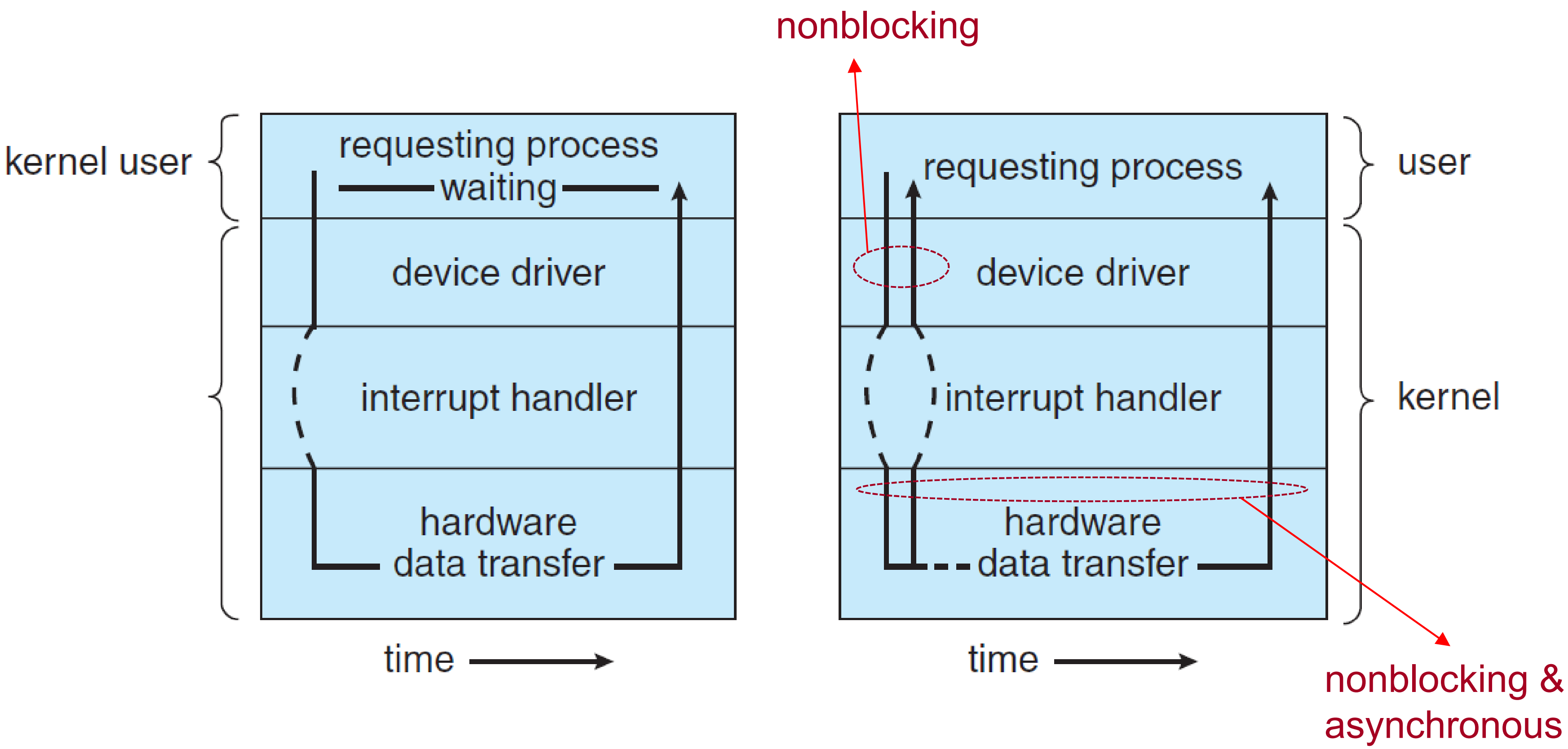
# Clocks and Timers

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- **Programmable interval timer** 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 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
  - **`select()`** to find if data ready then **`read()`** or **`write()`** to transfer
- ***Asynchronous*** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

# Two I/O Methods

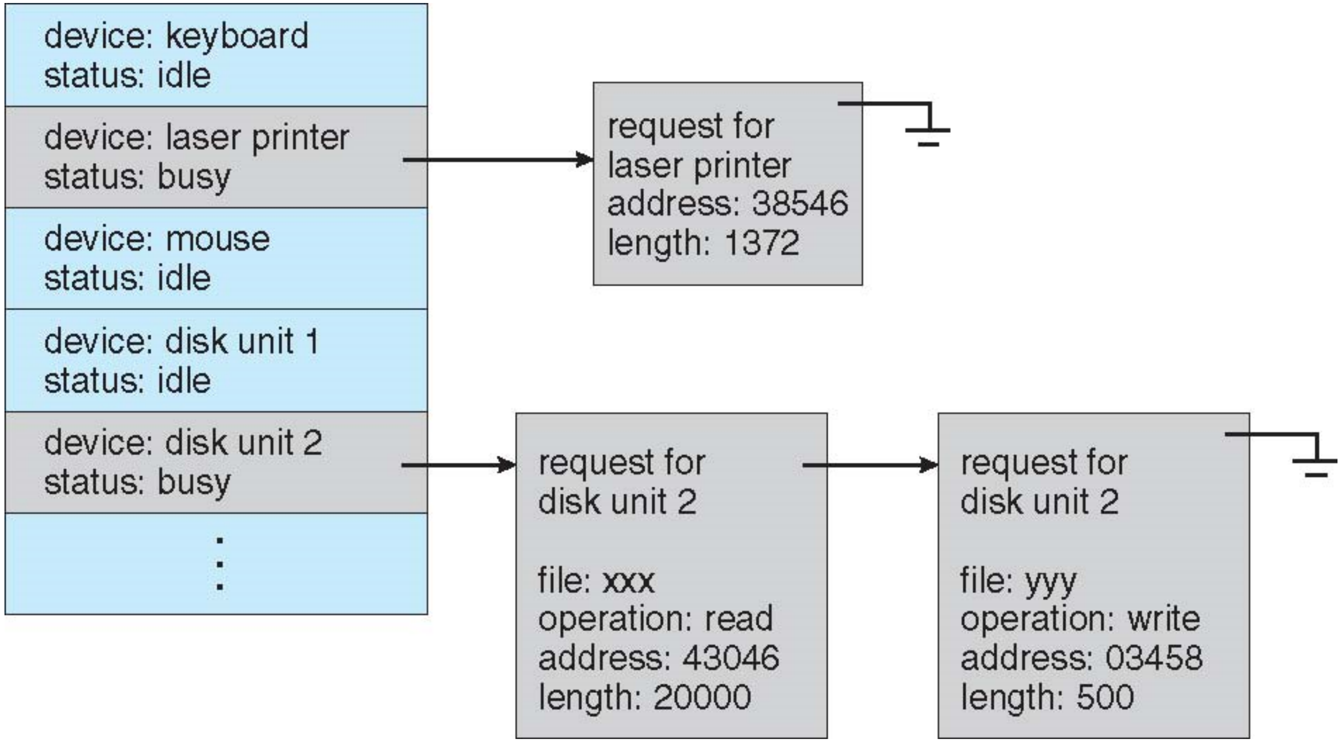



**(a) Synchronous & blocking** **(b) Asynchronous**

# Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
  - Some implement Quality of Service (i.e. IP QoS)

- Device status table



# Kernel I/O Subsystem (Cont.)

- Buffering
  - Store data in memory while transferring between devices
  - To cope with device speed mismatch (say, $10^{-5}$ ~ $10^6$ bytes/s)
    - Mitigate the difference in transfer rates in various devices
    - Double buffering – two copies of the data
      - Kernel and user
      - Decouple the producer of data from the consumer
  - To cope with device transfer size mismatch
  - To maintain "copy semantics"
    - E.g., What if the application modifies the contents of a buffer while the kernel is transferring them to disk? → Copy semantics is not guaranteed ➔ This problem can be resolved by maintaining a separate buffer in the kernel, so the buffer contents are copied into the kernel buffer in advance

# Kernel I/O Subsystem (Cont.)

- **Caching** - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

- **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 de-allocation
  - Watch out for deadlock

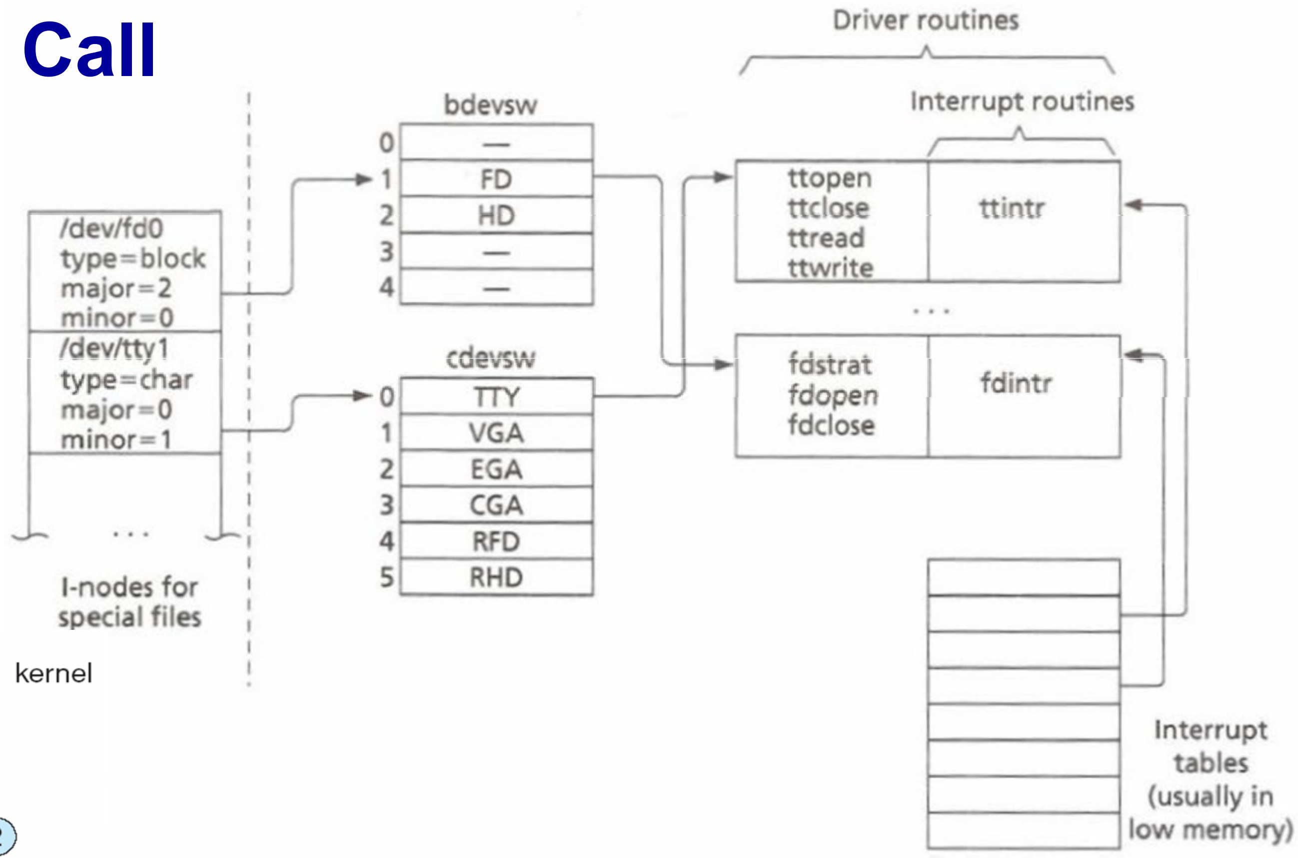
# Error Handling

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

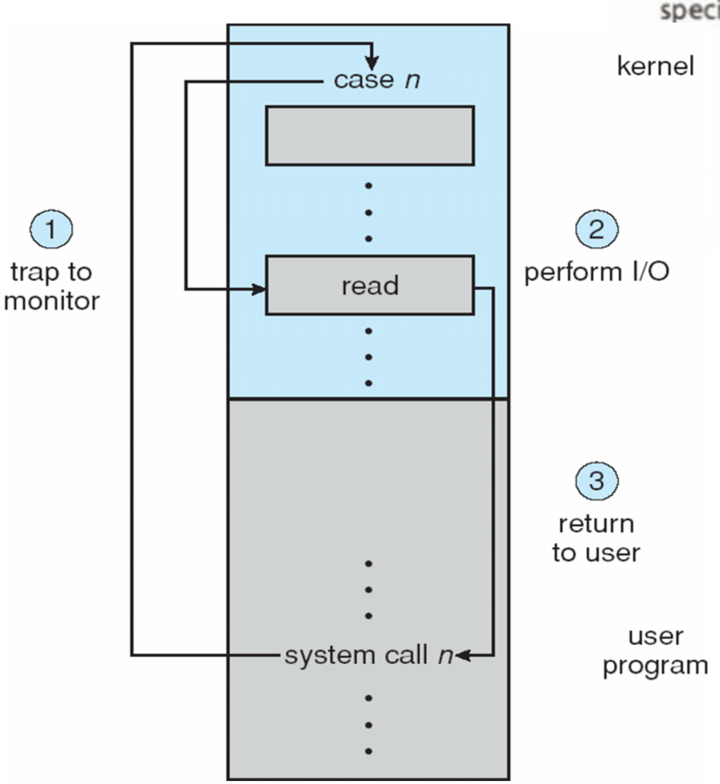
# I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

# Use of a System Call and Kernel Data Structures to Perform I/O




↑ Kernel data structures for accessing peripheral devices in Unix (K. Christian, S. Richter, *The UNIX Operating System, 3e.*, Wiley, 1994)

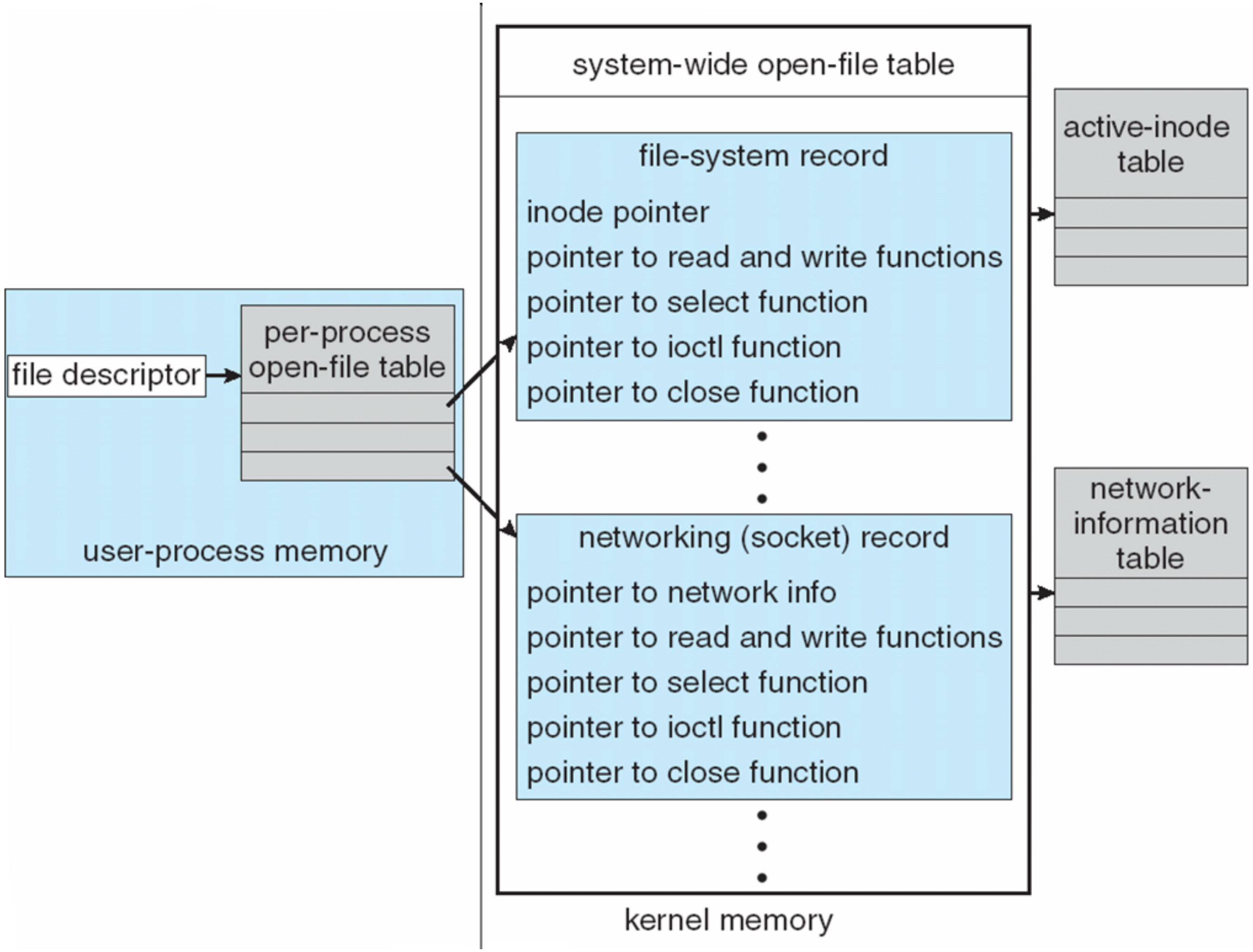



← Use of a system call to perform I/O

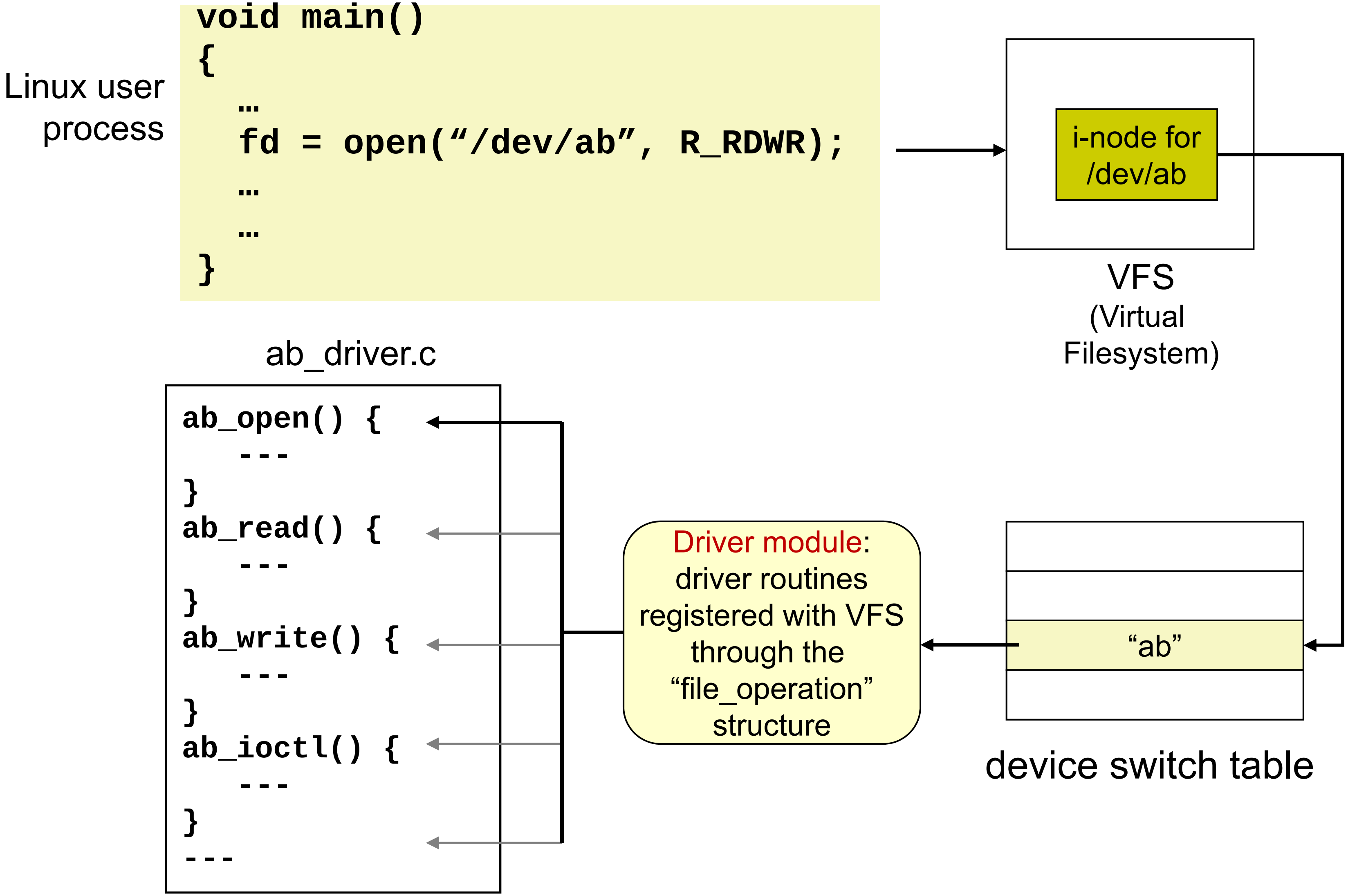
# Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, "dirty" blocks
- Some use object-oriented methods and message passing to implement I/O
  - Windows uses message passing
    - Message with I/O information passed from user mode into kernel
    - Message modified as it flows through to device driver and back to process
    - For input, the message contains the buffer to receive the data, and for output, the data to be written
    - Pros: Simplify the structure and the design of I/O system. Add flexibility
    - Cons: Overhead caused by shared data structures

# UNIX I/O Kernel Structure



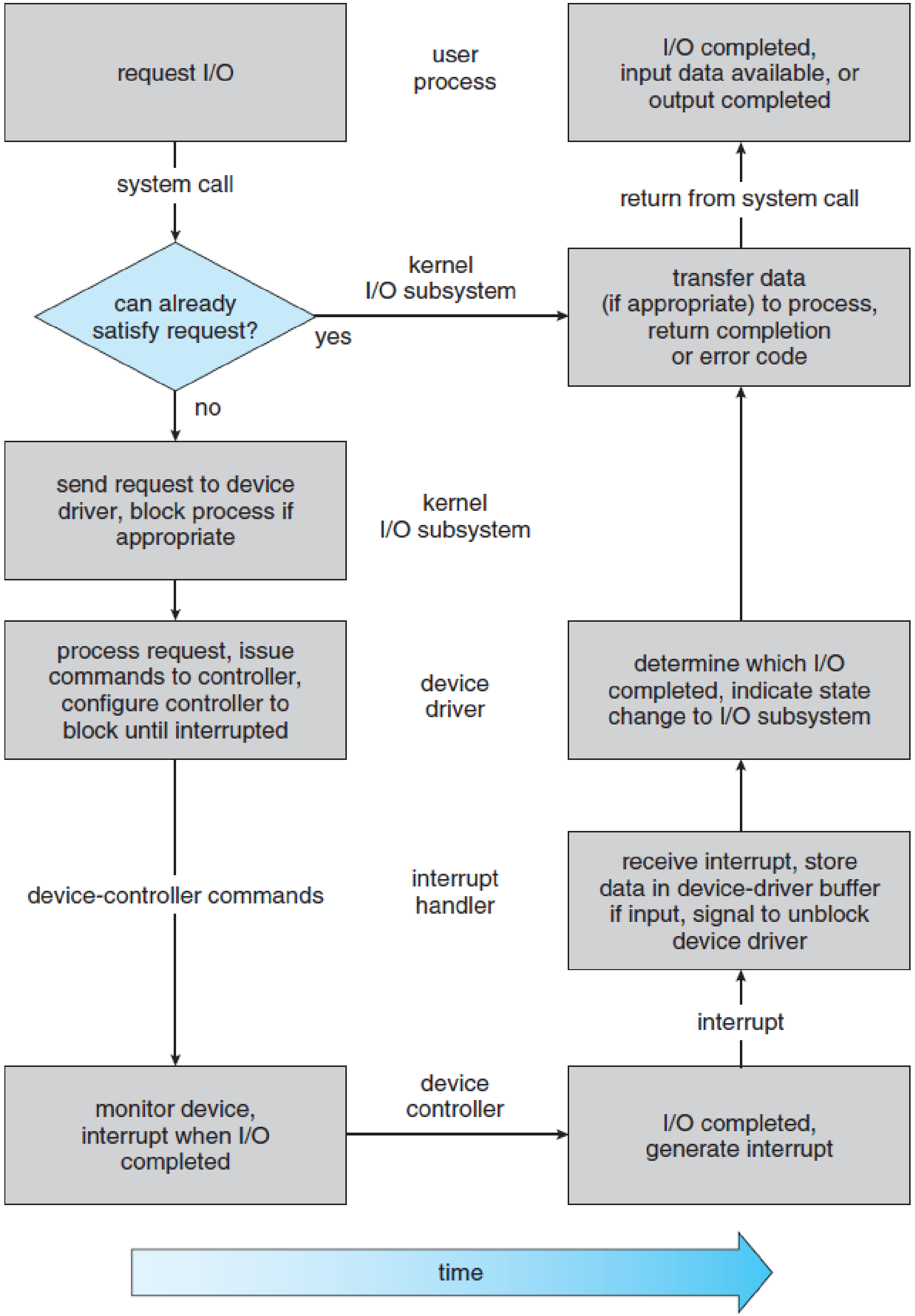
# Conceptual Description of Linux I/O



# Transforming I/O Requests to Hardware Operations

- 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

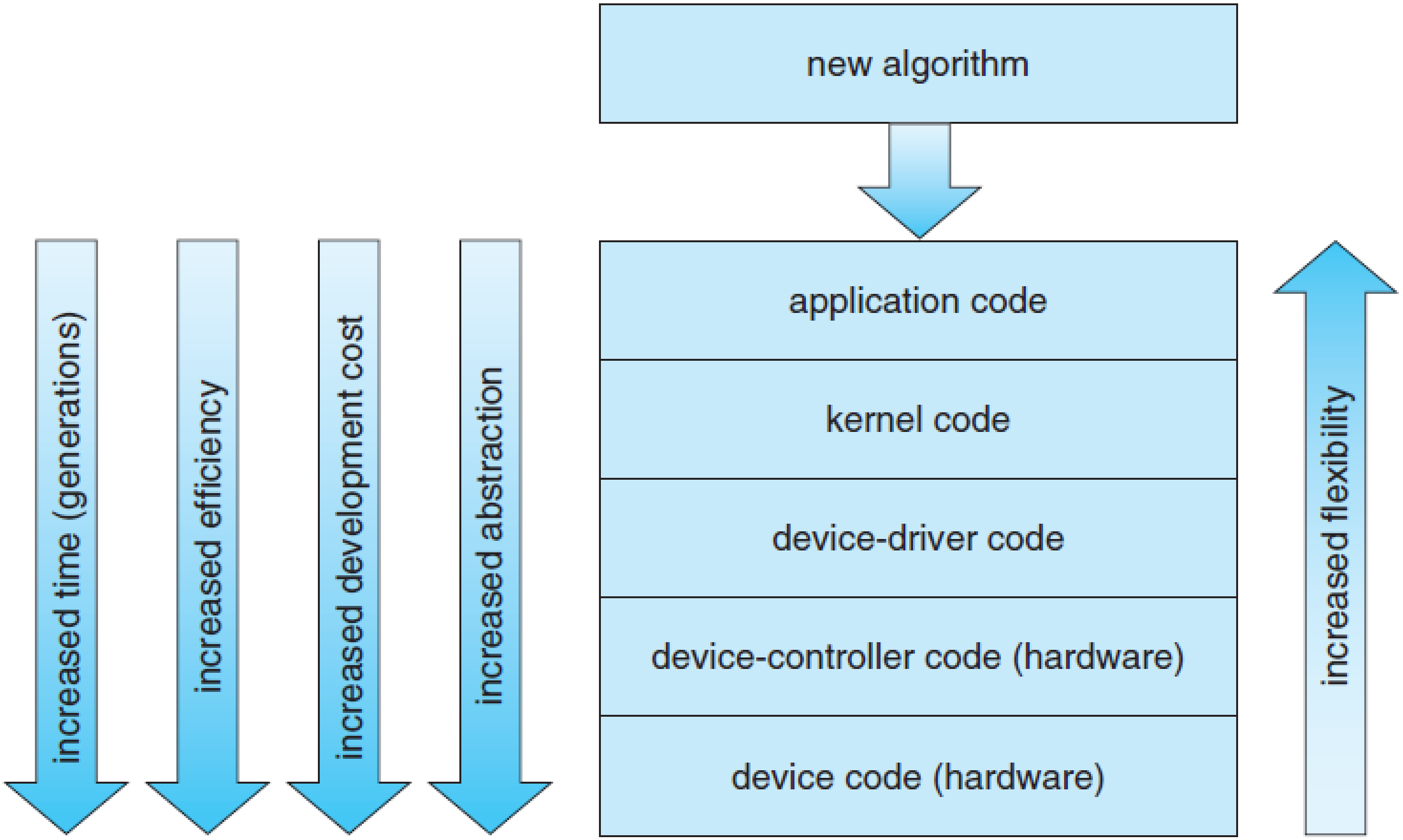


# Performance

- 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
  - Use smarter hardware devices
  - Balance CPU, memory, bus, and I/O performance for highest throughput
  - Move user-mode processes / daemons to kernel threads

# Device-Functionality Progression



# Summary

- 입출력시스템 개관
  - 입출력장치란?
  - 디바이스 드라이버 역할
- 입출력 하드웨어
  - 입출력시스템의 구성
  - 입출력장치의 구성, 역할
  - 입출력기법: 폴링, 인터럽트, DMA
- 응용 레벨의 입출력 인터페이스
  - 입출력장치의 유형 및 특징
  - block vs. character vs. network devices
  - clocks, timers
  - blocking/nonblocking, synchronous/asynchronous

- Overview of I/O systems
  - what is an I/O device?
  - role of device drivers
- I/O hardware
  - organization of I/O system
  - configuration and role of I/O device
  - I/O techniques: polling, interrupt, DMA
- Application I/O interface
  - types, characteristics of I/O devices
  - block vs. character vs. network devices
  - clocks, timers
  - blocking/nonblocking, synchronous/asynchronous

# Summary (Cont.)

- 커널 입출력시스템
  - 스케줄링, 버퍼링, 캐싱, 스풀링
  - 에러 처리, 입출력 보호
  - 커널 데이터 구조
- 입출력 요청부터 하드웨어 동작까지
  - 입출력의 계층적인 구조상에서 명령과 데이터의 흐름
- 성능 개선
  - 문맥교체 수 줄임
  - 데이터 복사 줄임
  - 인터럽트 수 줄임
  - DMA 사용
  - 커널 쓰레드 사용
  - CPU 부터 입출력장치 사이의 데이터 흐름의 최적화

- Kernel I/O subsystem
  - scheduling, buffering, caching, spooling
  - error processing, protection of I/O
- Transforming I/O requests to hardware operations
  - flows of data and commands in the I/O hierarchical structure
- Performance enhancement
  - reduce the number of context switches
  - reduce data copying
  - reduce the number of interrupts
  - use DMA
  - use kernel threads
  - optimize the data flow between CPU through I/O devices