Chapter 13 I/O Systems

LI Wensheng, SCST, BUPT

Strong points:

I/O Hardware

Application I/O Interface, Kernel I/O Subsystem

Transforming I/O Requests to Hardware Operations

Chapter Objectives

- **Explore the structure of an operating system's I/O subsystem.**
- **Discuss the principles and its complexities of I/O hardware.**
- **Explain the performance aspects of I/O hardware and software.**

Contents

13.1 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
- Two trends
  - Increasing standardization of software and hardware interfaces.
  - Increasingly broad variety of I/O devices.

标准化，统一接口
USB（Universal Serial Bus）
方便—用户、制造商。

Overview(Cont.)

- **The basic I/O hardware elements accommodate a wide variety of I/O devices.**
  - **Ports, busses, device controllers connect to various devices.**
- **To encapsulate the details and oddities of different devices, the kernel is structured to use device-driver modules.**
- **The device drivers present a uniform device access interface to the I/O subsystem.**
  - **much as system calls provide a standard interface between the application and the operating system.**

IO接口，
安全意识，资源保护，
遵章守法，服务意识。

行业标准很重要
标准—话语权

5G标准，华为

- 在5G标准中，3GPP（ **3rd Generation Partnership Project** ）定义了使用的三大场景：
  - eMBB（ **Enhanced Mobile Broadband**）：3D/超高清视频等大流量移动宽带业务；
  - mMTC（ **Massive MachineType Communication**）：大规模物联网业务；
  - URLLC（超可靠低延迟通信）：如无人驾驶、工业自动化等需要低时延、高可靠连接的业务。
- 在eMBB场景下的标准成为了各家争夺的焦点，在这个场景之中又形成了三大阵营：
  - 以高通为代表的美国企业阵营 LDPC code(低密度奇偶校验码)；
  - 以华为为代表的中国企业阵营 Polar code(极化码)；
  - 以法国为代表的欧洲企业阵营 Turbo code(涡轮码)。

5G标准，华为（续）

- 在5G发展初期，5G标准选择厘米波还是毫米波？
  - 美国认为毫米波更先进；
  - 华为任正非选择的是厘米波，因为他认为厘米波虽然不如毫米波先进，但覆盖范围大、建设成本低。
- 截至2020年1月1日，全球共有 21571 个5G标准专利项声明。
  - 华为拥有 3147 项，排名第一；
  - 领先第二名 三星 352件；
  - 中兴通讯拥有 2561 件排名第三；
  - LG电子2300件；
  - 诺基亚2149件；
  - 爱立信1494件。
- 华为在5G标准制定方面有较大的话语权，美国要进行5G，很难绕开华为。

5G标准，华为（续）

- 华为申请的5G专利数量全球第一，更为重要的是，华为还制定了5G标准，这让美国和一些西方国家无法绕开。
- 美国不爽了，于是开始污蔑华为5G技术不安全。
  - 美国本土率先公布要求本土运营商不得与华为合作建设5G网络；
  - 使用华为设备和技术的运营商，要替换掉华为设备等；
  - 为安全为由，威胁其他国家，放弃和华为合作。
- 任正非表示，华为可以将5G技术打包卖给美国企业，让美国企业在华为5G技术的基础上，生产5G设备等，甚至研发6G。

5G标准，华为（续）

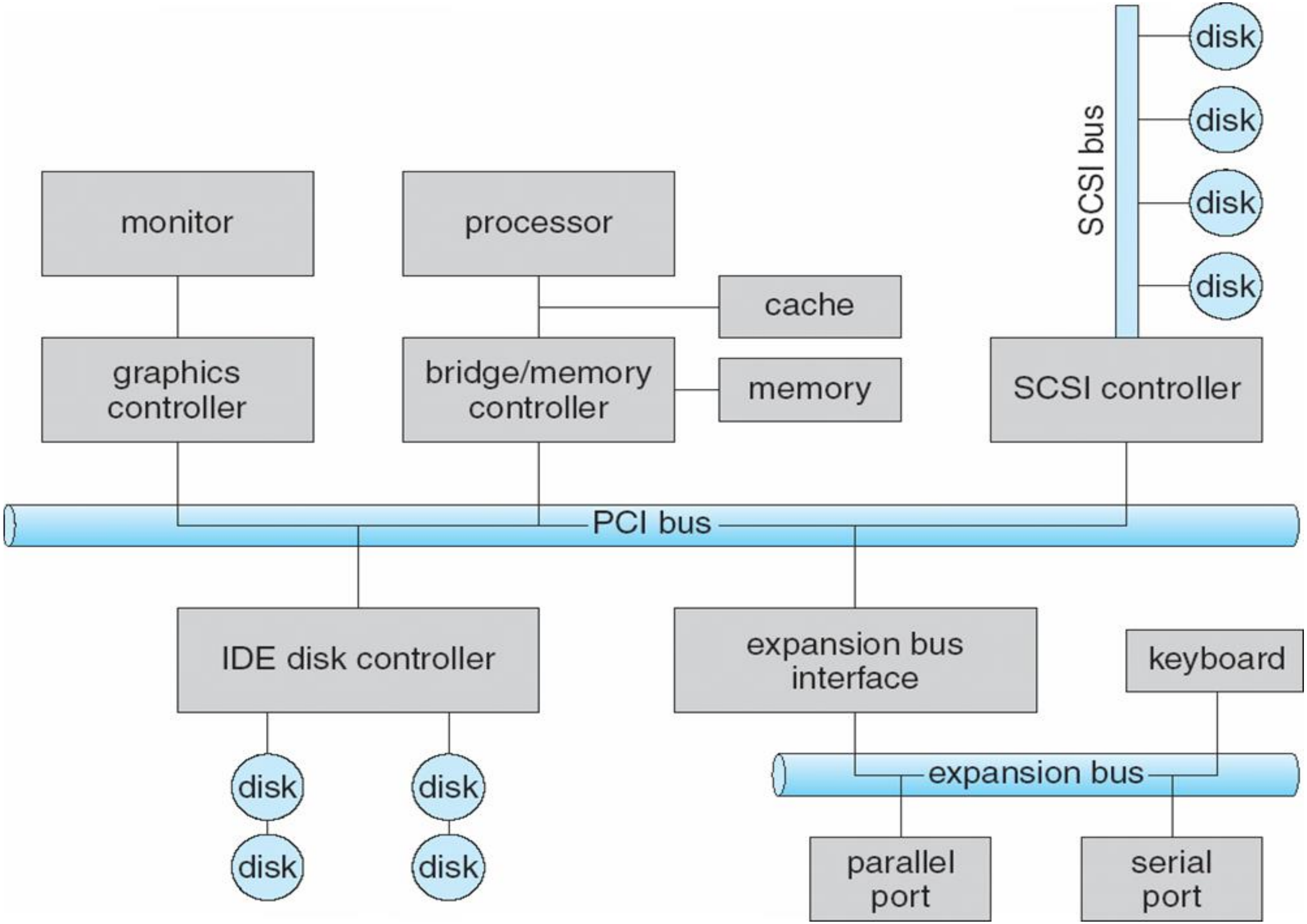
- 2020年10月4日消息，华为意大利分公司总裁路易吉 ·德 ·维奇斯（Luigi De Vecchis），在罗马网络安全中心的开幕式上公开表示：“ 我们将开放并展示我们的内部工作，我们可以接受解剖级的验证以回应所有这些外界压力。”
- 华为准备接受彻底的审查，以证明其技术不会对将在 5G 网络建设中使用其设备的国家构成任何风险。
- 真金不怕火炼！华为用行动打破谣言！
- 向外面传递了两个信息：
  - 第一，华为的5G技术和设备是绝对安全的，不存在国家安全隐患问题。
  - 第二，华为的5G技术已经领先了世界，即使给你们看，也无所谓。
- 标准的重要性。

end 1

13.2 I/O Hardware

- **Incredible variety of I/O devices, the general categories:**
  - **Storage (disks, tapes)**
  - **Transmission (network connections, Bluetooth)**
  - **Human-interface (screen, keyboard, mouse, audio in and out)**
- **Common concepts-- signals from I/O devices interface with computer.**
  - **Port : connection point for device, e.g. serial port.**
  - **Bus: a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires (daisy chain or shared direct access).**
    - **PCI bus, common in PCs and servers, PCI Express (PCIe)**
    - **expansion bus connects relatively slow devices.**
  - **Controller (host adapter): a collection of electronics that can operate a port, a bus, or a device.**
    - **Sometimes integrated, Serial-port controller**
    - **Sometimes separate circuit board, SCSI bus controller**
    - **Contains processor, microcode, private memory, etc.**

A Typical PC Bus Structure



I/O Hardware (Cont.)

- **Processor give commands and data to a controller to accomplish an I/O transfer.**
- **I/O instructions control devices.**
- **Controller 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**
- **Devices have addresses, used by**
  - **Direct I/O instructions**
  - **Memory-mapped I/O**
    - **Device data and command registers mapped to processor address space**
    - **Especially for large address spaces (graphics)**

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)

I/O port

- **4 registers**
  - **Status: contains bits that can be read by the host.** **Idle, busy, ready, error**
  - **Control: can be written by the host to start a command or to change the mode of a device.**
  - **Data-in: read by the host to get input.**
  - **Data-out: written by the host to send output.**
- **Data registers are typically 1 to 4 bytes in size.**
- **Some controllers have FIFO chips to expand the capacity of the controller beyond the size of the data register.**
  - **FIFO chips can hold several bytes of input or output data.**

Techniques for Performing I/O

- Programmed I/O
  - Process is busy-waiting for the operation to complete
- Interrupt-driven I/O
  - I/O command is issued
  - Processor continues executing instructions
  - I/O module sends an interrupt when done
- Direct Memory Access (DMA)
  - DMA module controls exchange of data between main memory and the I/O device
  - Processor interrupted only after entire block has been transferred

Polling

busy-wait cycle to wait for I/O from device.
Reasonable if device is fast, but inefficient if device slow.
CPU switches to other tasks?

- **Determines state of device**
  - **command-ready, busy, Error**
- **Handshaking （host and controller, e.g. write out a byte）**
  1. **host repeatedly reads the busy bit until that bit becomes clear.**
  2. **host sets the write bit, and writes a byte into data-out register.**
  3. **host sets the command-ready bit.**
  4. **When the controller notices that the command-ready bit is set, it sets the busy bit.**
  5. **controller reads the command register and sees the write command.**
     It reads the data-out register to get the byte, and does the I/O to the device.
  6. **controller clears the command-ready bit, clears error bit in the status register, and clears the busy bit.**
- **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?**

Thinking the following question

- **The example of handshaking in polling used 2 bits: a busy bit and a command-ready bit.**
- **Is it possible to implement this handshaking with only 1 bit?**
  If it is, describe the protocol.
  If it is not, explain why 1 bit is insufficient.

Answer:

It is possible, using the following algorithm.

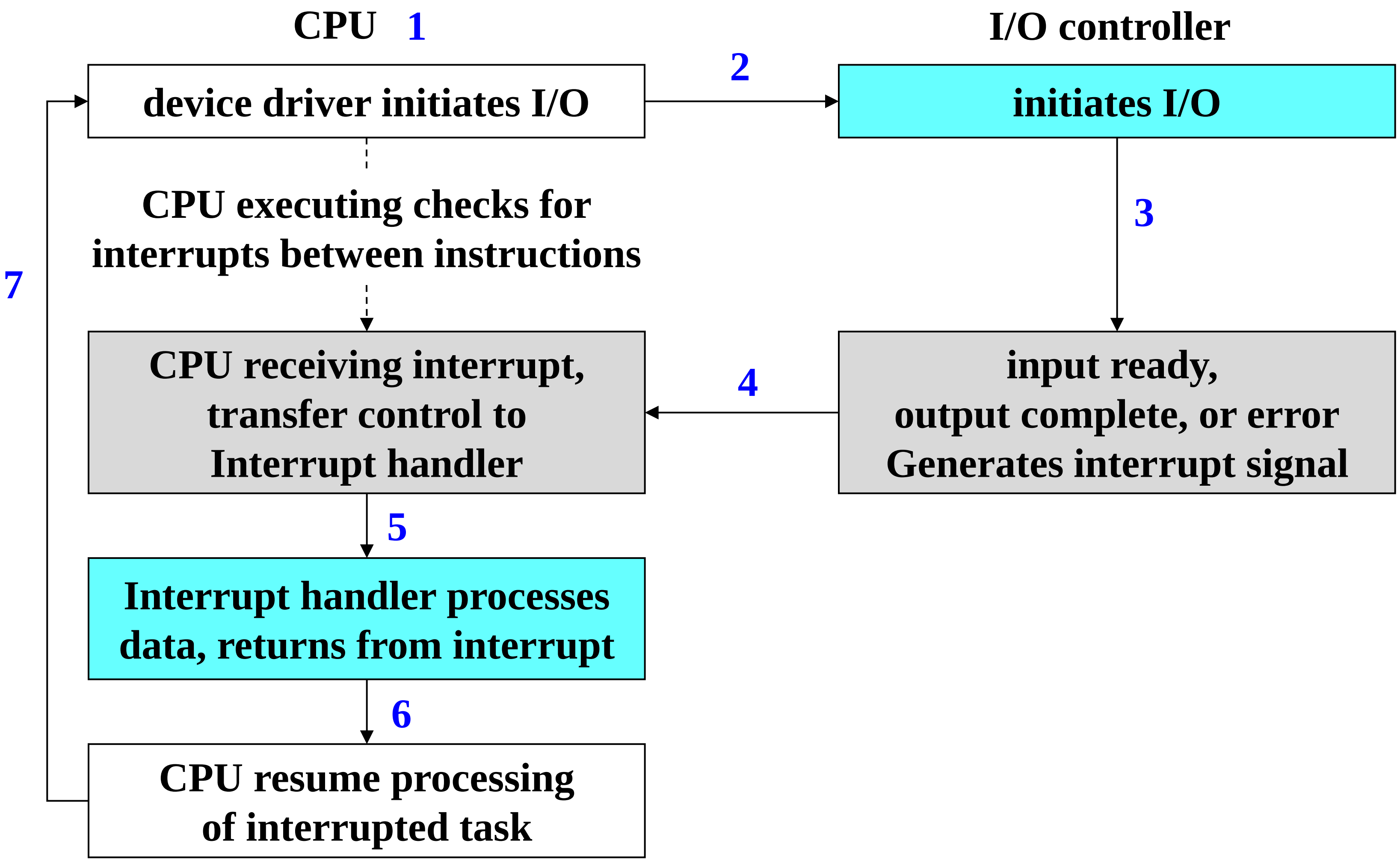
Let's assume we simply use the busy-bit (or the command-ready bit; this answer is the same regardless). When the bit is off, the controller is idle. The host writes to data-out and sets the bit to signal that an operation is ready (the equivalent of setting the command-ready bit). When the controller is finished, it clears the busy bit. The host then initiates the next operation.

This solution requires that both the host and the controller have read and write access to the same bit, which can complicate circuitry and increase the cost of the controller.

Interrupts

- CPU Interrupt-request line triggered by I/O device
  - Controller asserting a signal on the interrupt request line.
  - 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



Interrupts (Cont.)

- Two interrupts request lines
  - Nonmaskable, reserved for critical events.
  - Maskable, turned off to ignore or delay some interrupts.
  - Intel Pentium Processor Event-Vector Table:
- Interrupt priority levels
  - enable the CPU to defer the handling of low-priority interrupts without masking all interrupts.
  - makes it possible for a high priority interrupt to preempt the execution of a low-priority interrupt.

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 interrupt

Interrupts (Cont.)

- **Interrupt mechanism also used for exceptions:**
  - **Dividing by zero**
  - **accessing a protected or nonexistent memory address**
  - **Attempting to execute a privileged instruction from user mode**
- **Good uses:**
  - **System call**
  - **Page fault**
- **Used for time-sensitive processing, frequent, must be fast.**

Direct Memory Access

- Used to avoid programmed I/O (PIO) for large data movement.
- Requires DMA controller, a special-purpose processor.
- Bypasses CPU to transfer data directly between I/O device and memory.
  - Host writes a DMA command block into memory.
    - Source and destination addresses
    - Read or write mode
    - Count of bytes
  - CPU writes the address of this command block to the DMA controller.
  - DMA controller operates the memory bus directly, placing address on the bus to perform transfers.
  - When the entire transfer is finished, the DMA controller interrupts the CPU.

Direct Memory Access(Cont.)

- **Handshaking between the DMA controller and the device controller:**
  - **performed via a pair of wires called DMA-request and DMA-acknowledge.**
  - **The device controller places a signal on the DMA-request wire when a word of data is available for transfer.**
  - **This signal causes the DMA controller to**
    - **seize the memory bus**
    - **place the desired address on the memory-address wires**
    - **place a signal on the DMA-acknowledge wire.**
  - **When the device controller receives the DMA-acknowledge signal, it transfers the word of data to memory and removes the DMA-request signal.**

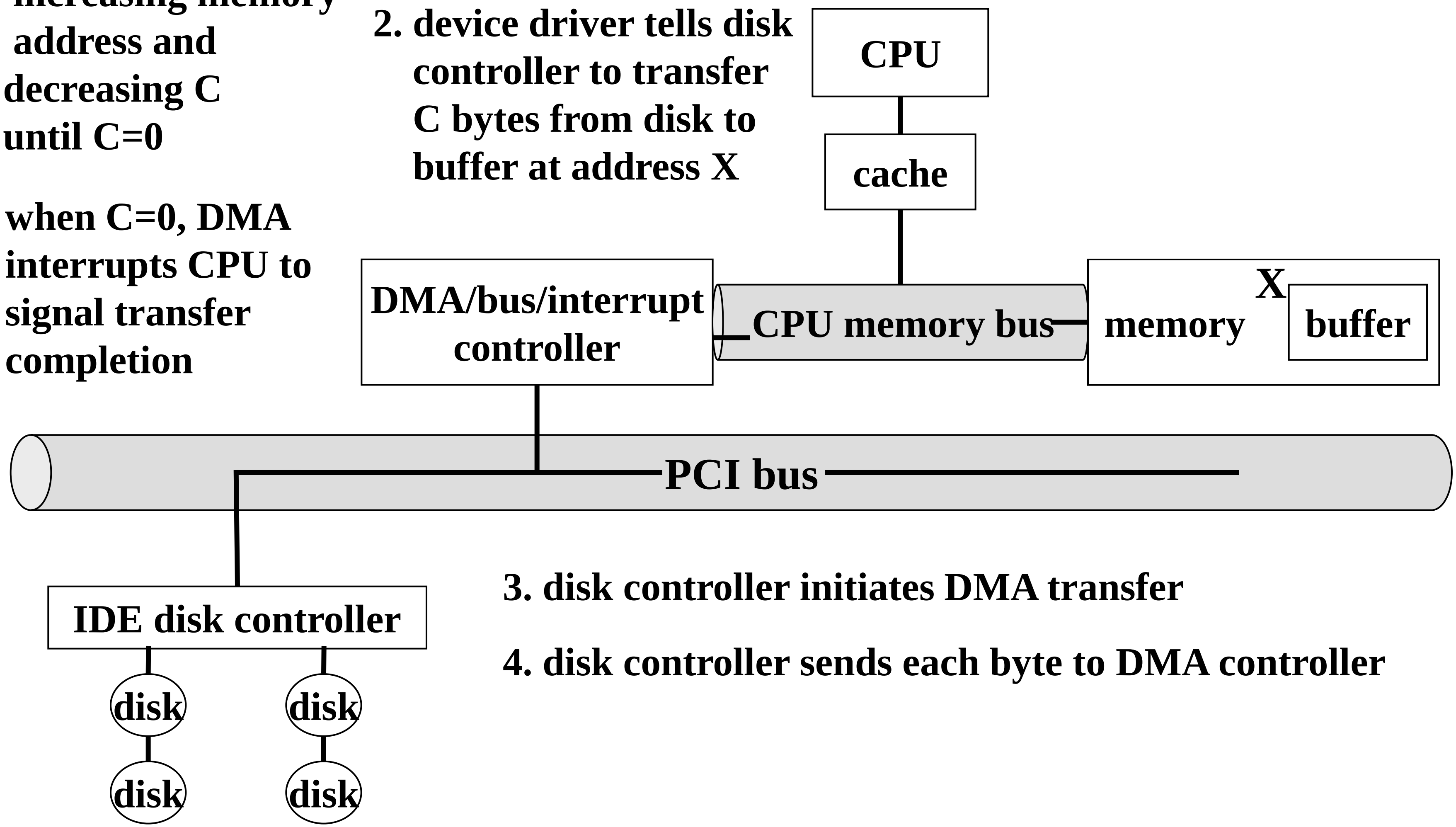
Six Step Process to Perform DMA Transfer

5. **DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C=0**

6. **when C=0, DMA interrupts CPU to signal transfer completion**

1. **device driver is told to transfer disk data to buffer at address X**

2. **device driver tells disk controller to transfer C bytes from disk to buffer at address X**



3. **disk controller initiates DMA transfer**

4. **disk controller sends each byte to DMA controller**

Direct Memory Access (Cont.)

- DMA controller seizes the memory bus, Cycle stealing is used to transfer data on the system bus.
  - CPU is prevented from accessing main memory.
  - CPU can still access data items in its caches.
- The instruction cycle is suspended so data can be transferred.
- The CPU pauses one bus cycle.
- No interrupts occur
  - Do not save context
- DVMA (direct virtual memory access) can perform a transfer between two memory-mapped devices without the intervention of the CPU or the use of main memory.

end 2

Thinking the following question

- How does DMA increase system concurrency? How does it complicate hardware design?

Answer:

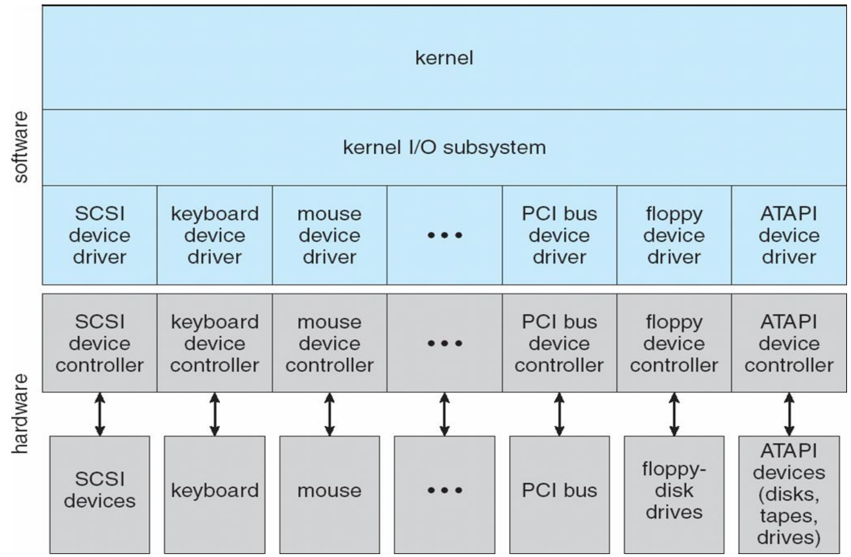
- **DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses.**
- **Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master.**

Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

end 2

13.3 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.
- 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
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only



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(Cont.)

- **Subtleties of devices handled by device drivers.**
- **Broadly I/O devices can be grouped by the OS into**
  - **Block I/O**
  - **Character- stream I/O**
  - **Memory-mapped file access**
  - **Network sockets**
- **OS provide special system calls to access a few additional devices, such as time-of-day clock and a timer.**
- **OSs have an escape / back door, transparently passes arbitrary commands from an application to a device driver.**
  - **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.
- The socket interface, Linux, Unix and Windows
  - Separates network protocol from network operation
  - Includes select functionality: manages a set of sockets
  - The system calls in the socket interface enable an application
    - to create a socket
    - to connect a local socket to a remote address
    - to listen for any remote application to plug into the local socket
    - to send and receive packets over the connection
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

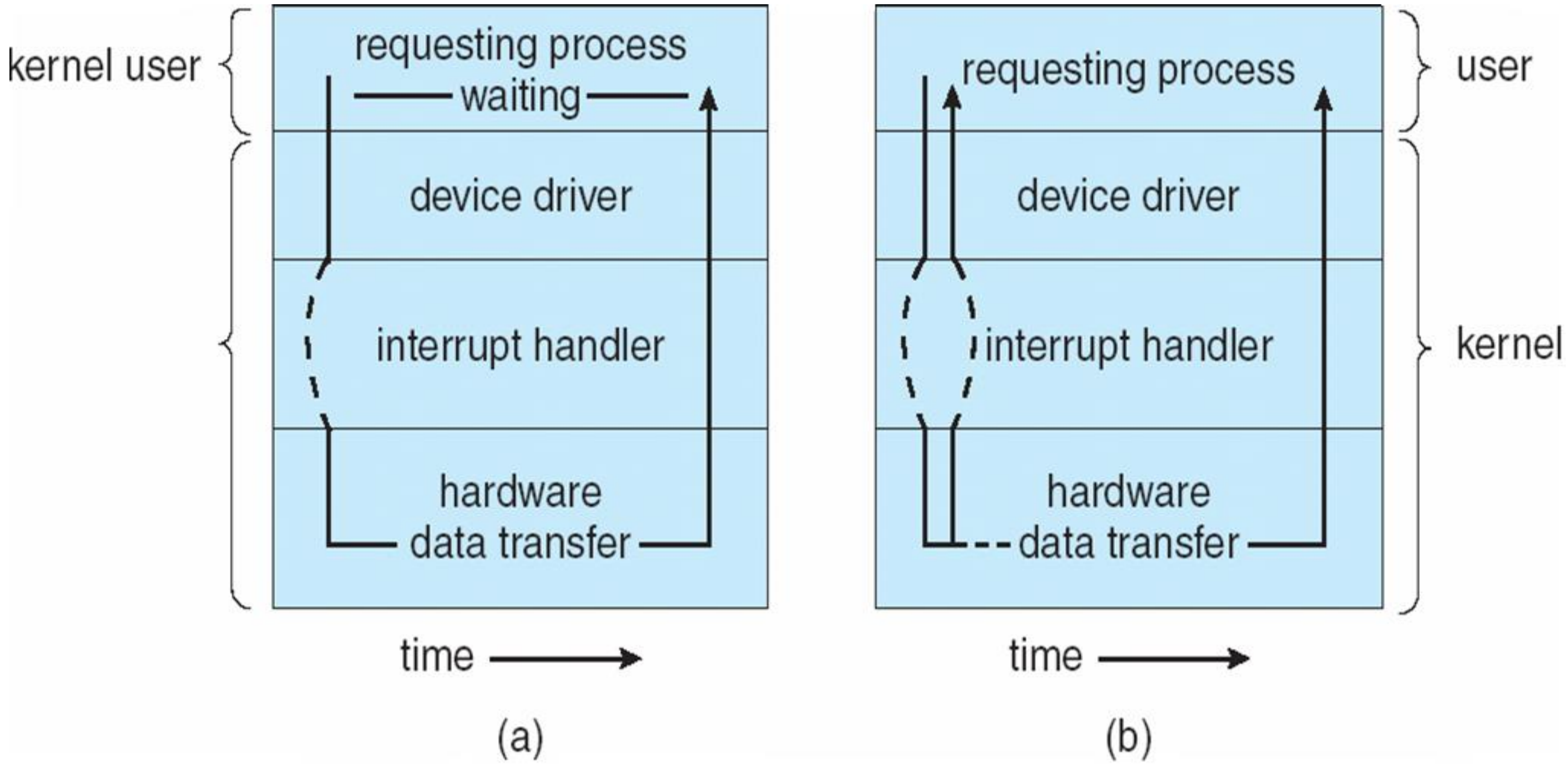
Clocks and Timers

- **Provide current time, elapsed time, timer**
- **Programmable interval timer used for timings, periodic interrupts.**
- **Normal resolution about 1/60 second.**
- **Some systems provide higher-resolution timers.**
- **`ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers.**

Blocking I/O 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
  - E.g. User interface, receives keyboard and mouse input while processing and displaying data on the screen. Video application, reads frames from a file on disk while simultaneously decompressing and displaying the output on the display.
  - Overlap; 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



Synchronous **Asynchronous**

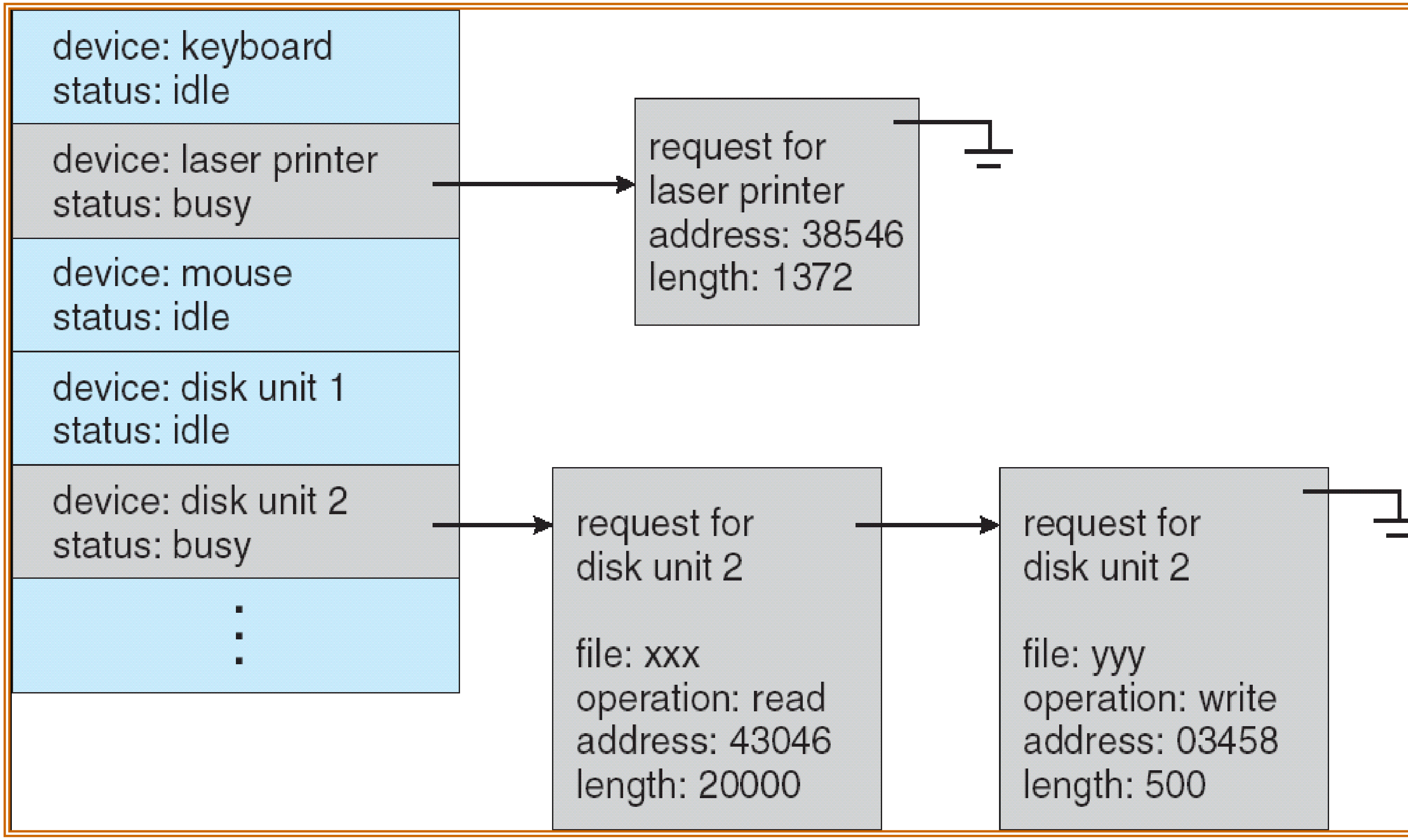
end 3

13.4 Kernel I/O Subsystem

- **Services provided by kernel I/O subsystem:**
  - **① Scheduling**
  - **② Buffering**
  - **③ Caching**
  - **④ Spooling**
  - **⑤ device reservation**
  - **⑥ error handling**

① I/O Scheduling

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



Wensheng Li BUPT

②buffering

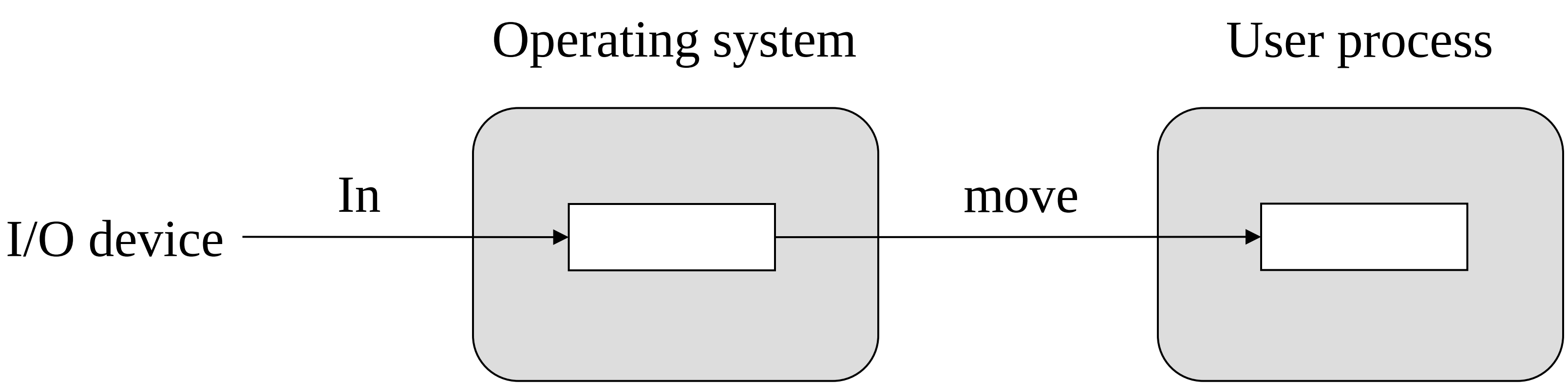
- **Buffering - store data in memory while transferring between devices**
  - **To cope with device speed mismatch**
    e.g. a file is being received via modem for storage on the hard disk.
  - **To cope with device transfer size mismatch**
    e.g. fragmentation and reassembly of message.
  - **To maintain "copy semantics"**
    e.g. an application write a buffer of data to disk.

* I/O Buffering

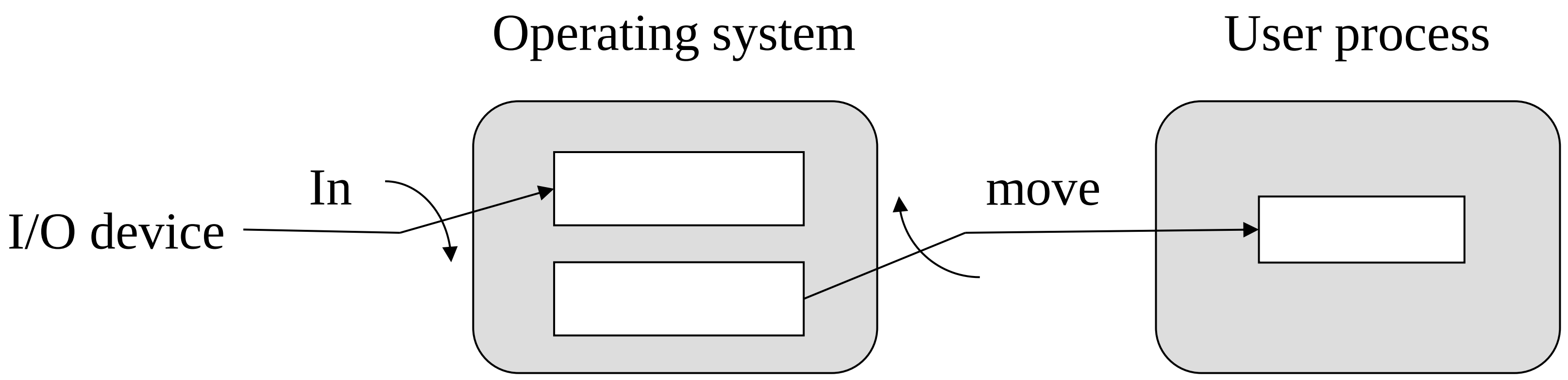
- **Block-oriented**
  - **Information is stored in fixed sized blocks.**
  - **Transfers are made a block at a time.**
  - **Used for disks and tapes.**
  - **User process can process one block of data while next block is read in.**
  - **Swapping can occur since input is taking place in system memory, not user memory**
  - **Operating system keeps track of assignment of system buffers to user processes.**
- **Stream-oriented**
  - **Transfer information as a stream of bytes.**
  - **Used a line at time.**
  - **Used for terminals, printers, communication ports, mouse, and most other devices that are not secondary storage.**
    - **User input from a terminal is one line at a time with carriage return signaling the end of the line.**
    - **Output to the terminal is one line at a time.**

* Single Buffer

- Operating system assigns a buffer in main memory for an I/O request.
- Block-oriented
  - Input transfers made to buffer
  - Block moved to user space when needed
  - Another block is moved into the buffer
    - Read ahead

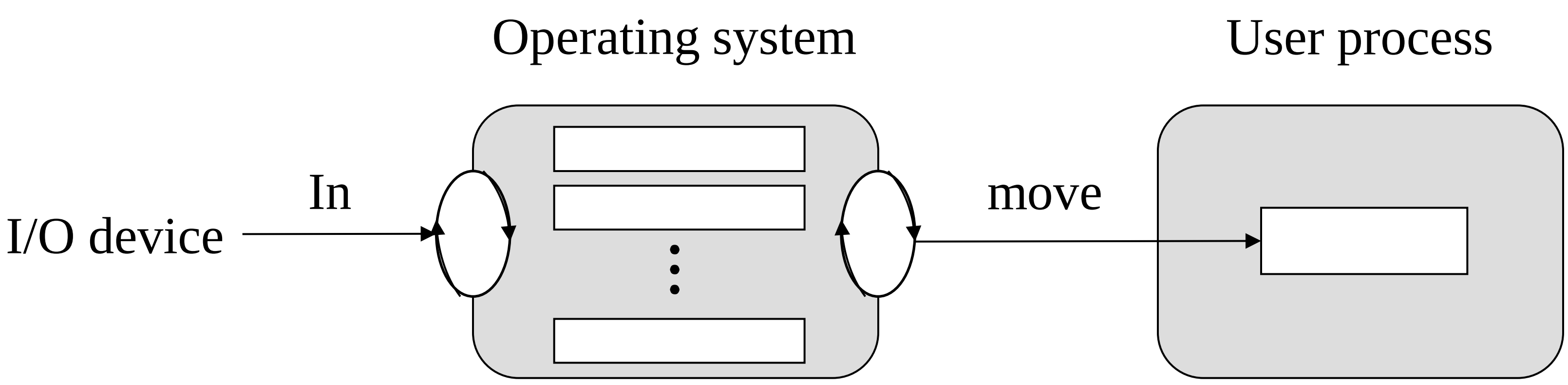


* Double Buffer



- **Use two system buffers instead of one**
- **A process can transfer data to or from one buffer while the operating system empties or fills the other buffer**

* Circular Buffer



- **More than two buffers are used**
- **Each individual buffer is one unit in a circular buffer**
- **Used when I/O operation must keep up with process**

③ caching

- **Caching - fast memory holding copy of data.**
- **Key to performance.**
- **Always just a copy.**
- **difference between a buffer and a cache**
  - **A buffer may hold the only existing copy of a data item.**
  - **A cache just holds a copy on faster storage of an item that resides elsewhere.**
- **Sometimes, a region of memory can be used for both purposes.**

④ Spooling & Device reservation

- **Spooling - hold output for a device**
  - **If device can serve only one request at a time**
  - **i.e., Printing**
  - **Each application's output is spooled to a separate disk file.**
  - **The spooling system copies the queued spool files to the printer one at a time.**
- **Device reservation(预约) - provides exclusive access to a device**
  - **System calls for allocation and deallocation**
  - **Watch out for deadlock**

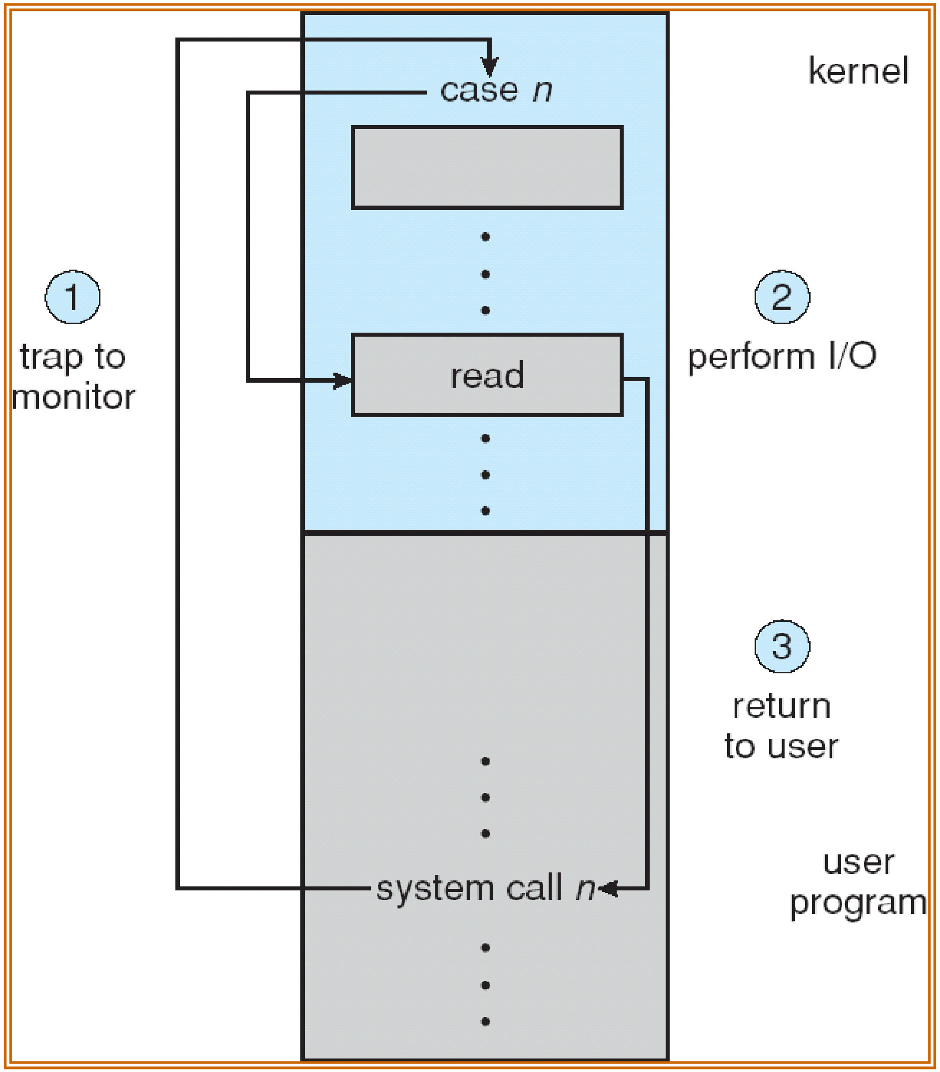
⑤ Error Handling

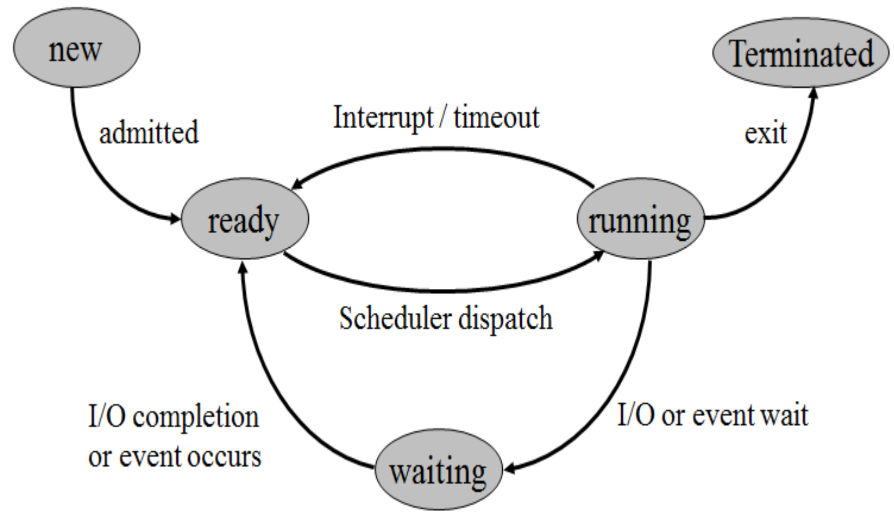
- OS can recover from disk read, device unavailable, transient write failures.
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code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

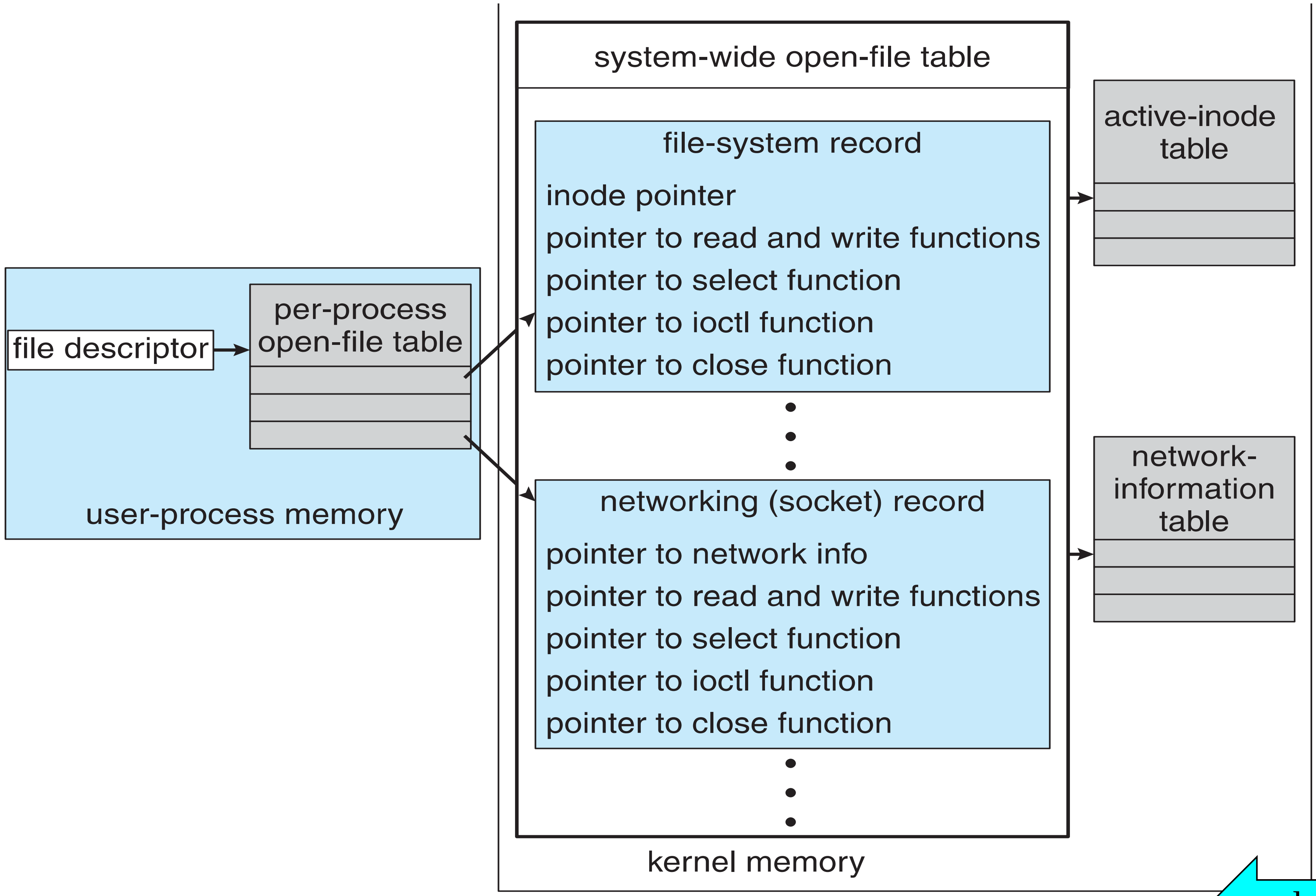




Kernel Data Structures

- Kernel keeps state information 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.
  - Message with I/O information passed from user mode into kernel.
  - Message modified as it flows through to device driver and back to process.
  - Pros / cons?（利弊？）

UNIX I/O Kernel Structure

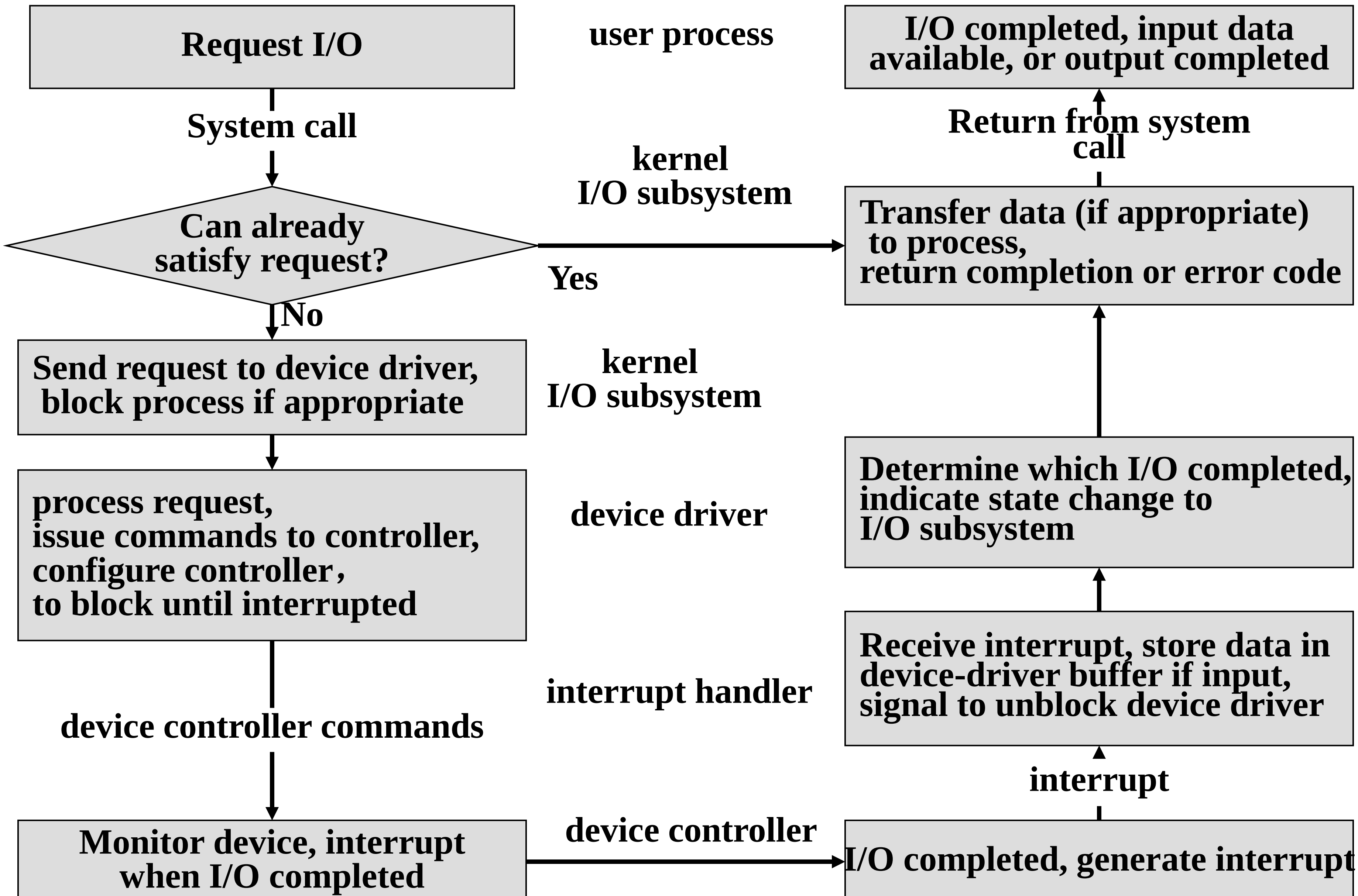


end 4

13.5 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

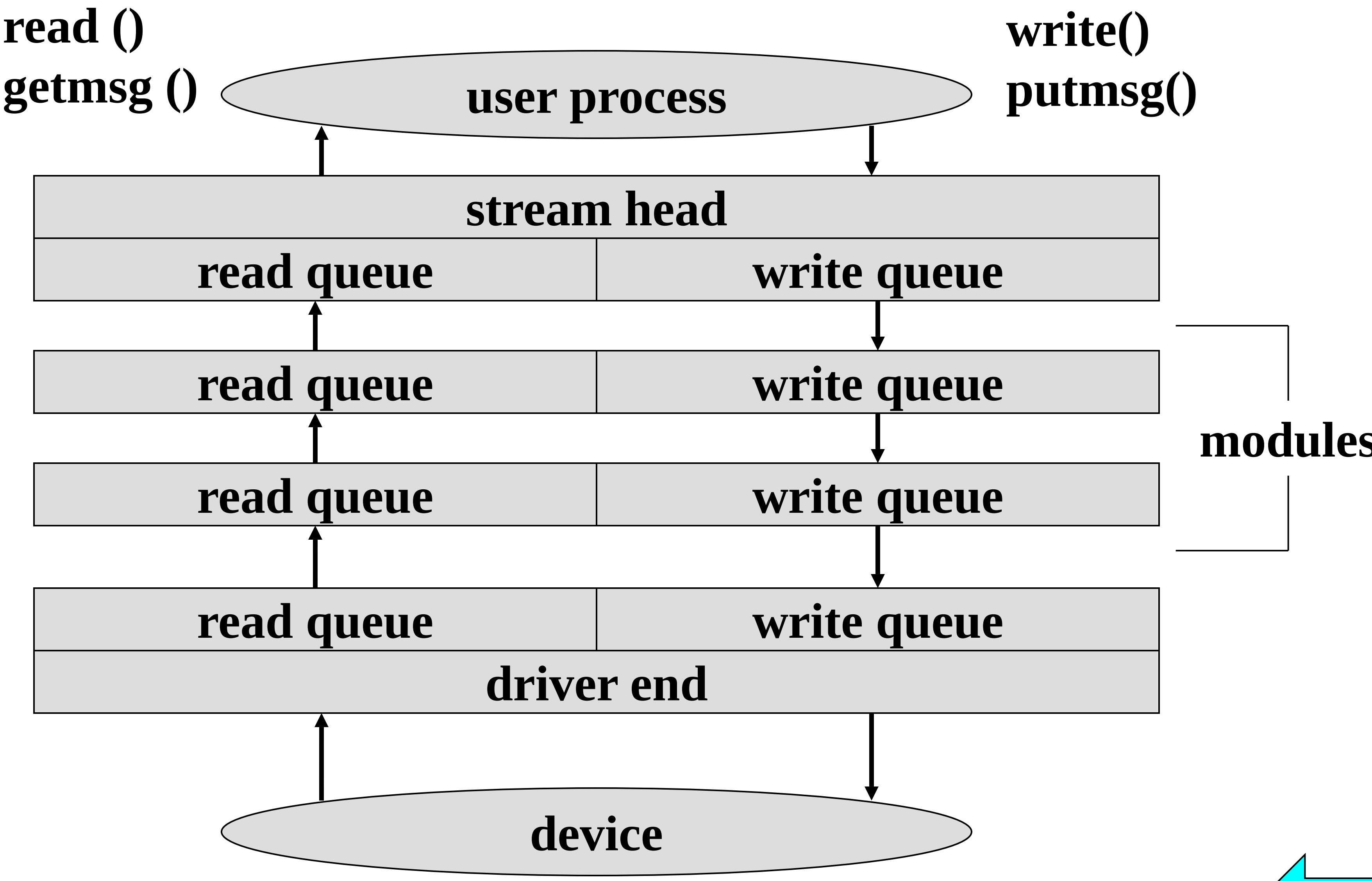
Life Cycle of An I/O Request



13.6 STREAMS

- STREAM – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond.
- A STREAM consists of:
  - STREAM head interfaces with the user process
  - driver end interfaces with the device
  - zero or more STREAM modules between them.
- Each module contains a read queue and a write queue.
- Message passing is used to communicate between queues.
  - Flow control option to indicate available or busy.
- Asynchronous internally, synchronous where user process communicates with stream head.

The STREAMS Structure

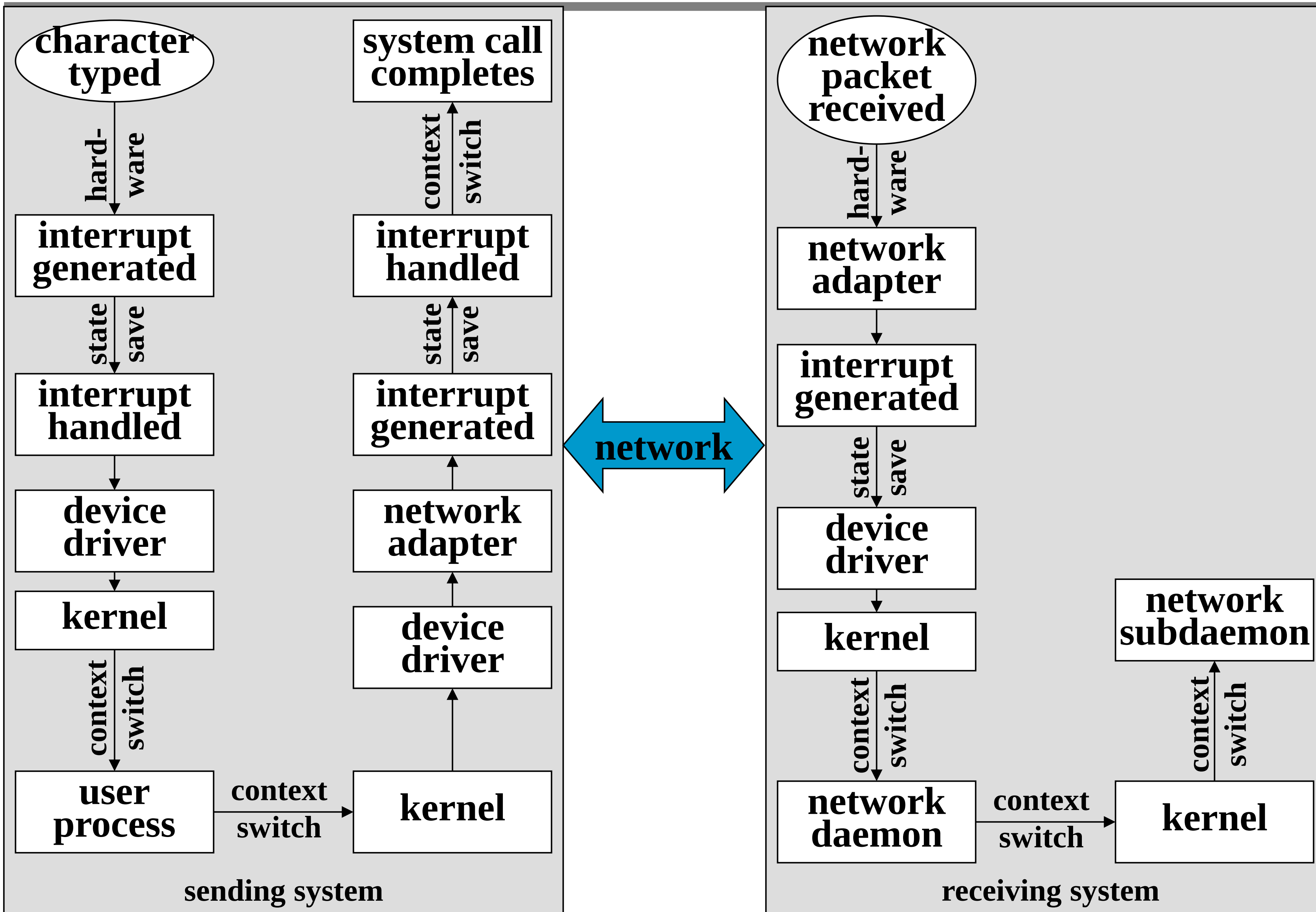


end 6

13.7 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**
  - **Data copying**
  - **Network traffic especially stressful**

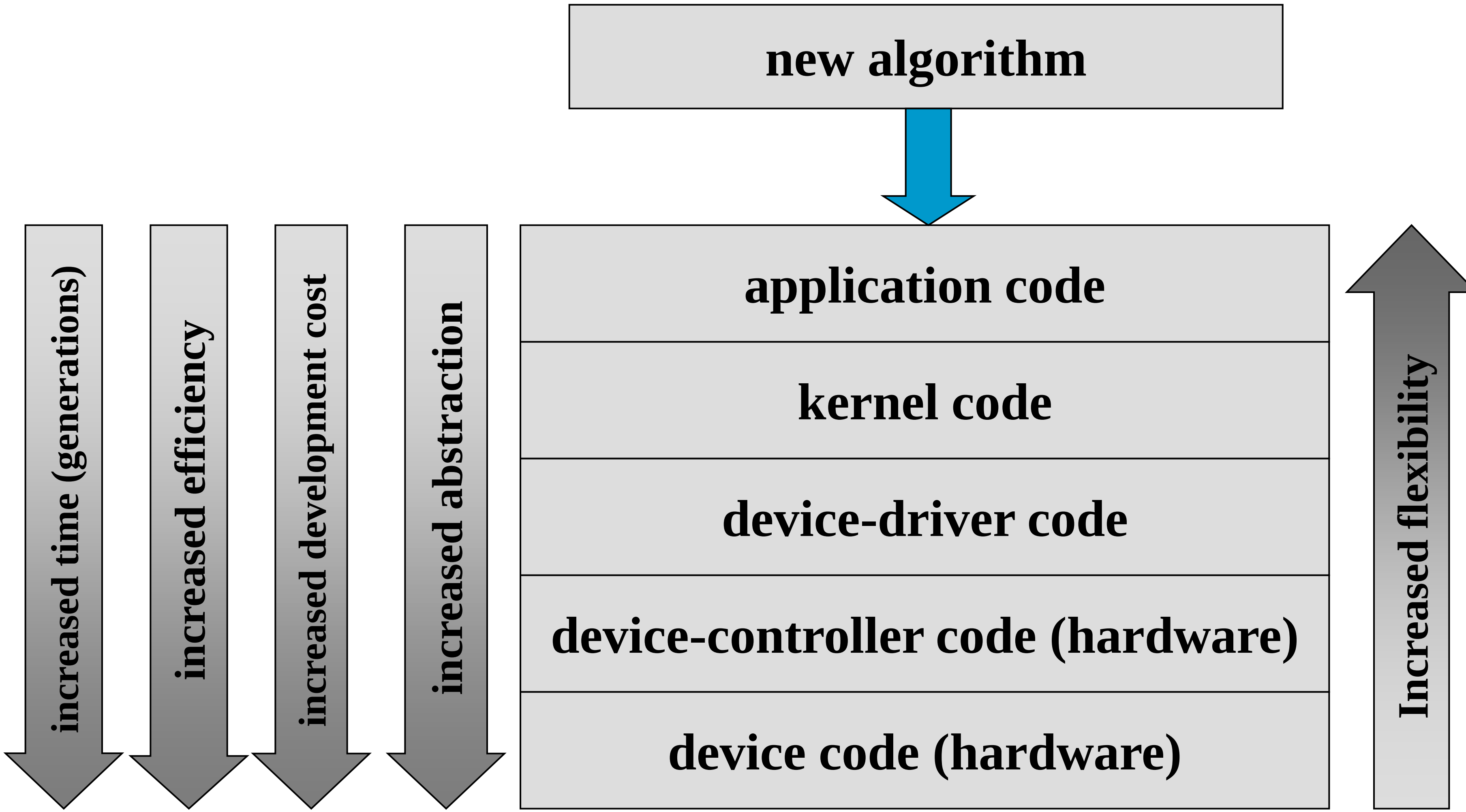
Intercomputer Communications



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



index7

Homework (page 526)

- **13.6**