Principles of Operating Systems

CS 446/646

5. Input/Output

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Spring 2006

Principles of Operating Systems

CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. Processes
3. Memory Management
4. CPU Scheduling
5. **Input/Output**
6. File System
7. Case Studies

Principles of Operating Systems

CS 446/646

5. Input/Output

a. **Overview of the O/S Role in I/O**

b. **Principles of I/O Hardware**

c. **I/O Software Layers**

d. **Disk Management**

Principles of Operating Systems

CS 446/646

5. Input/Output

a. **Overview of the O/S Role in I/O**

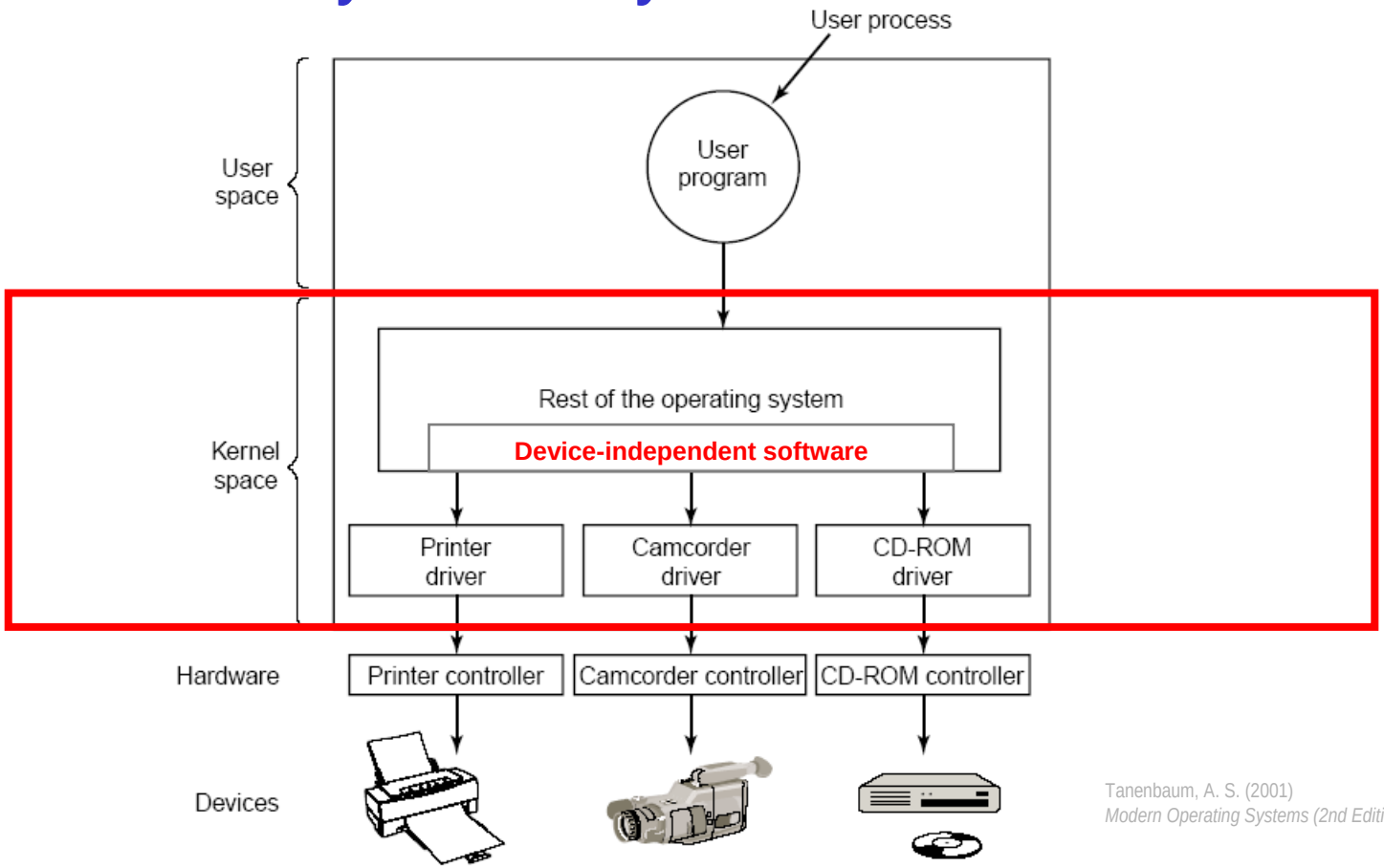
b. **Principles of I/O Hardware**

c. **I/O Software Layers**

d. **Disk Management**

5.a Overview of the O/S Role in I/O

- The I/O subsystem is layered

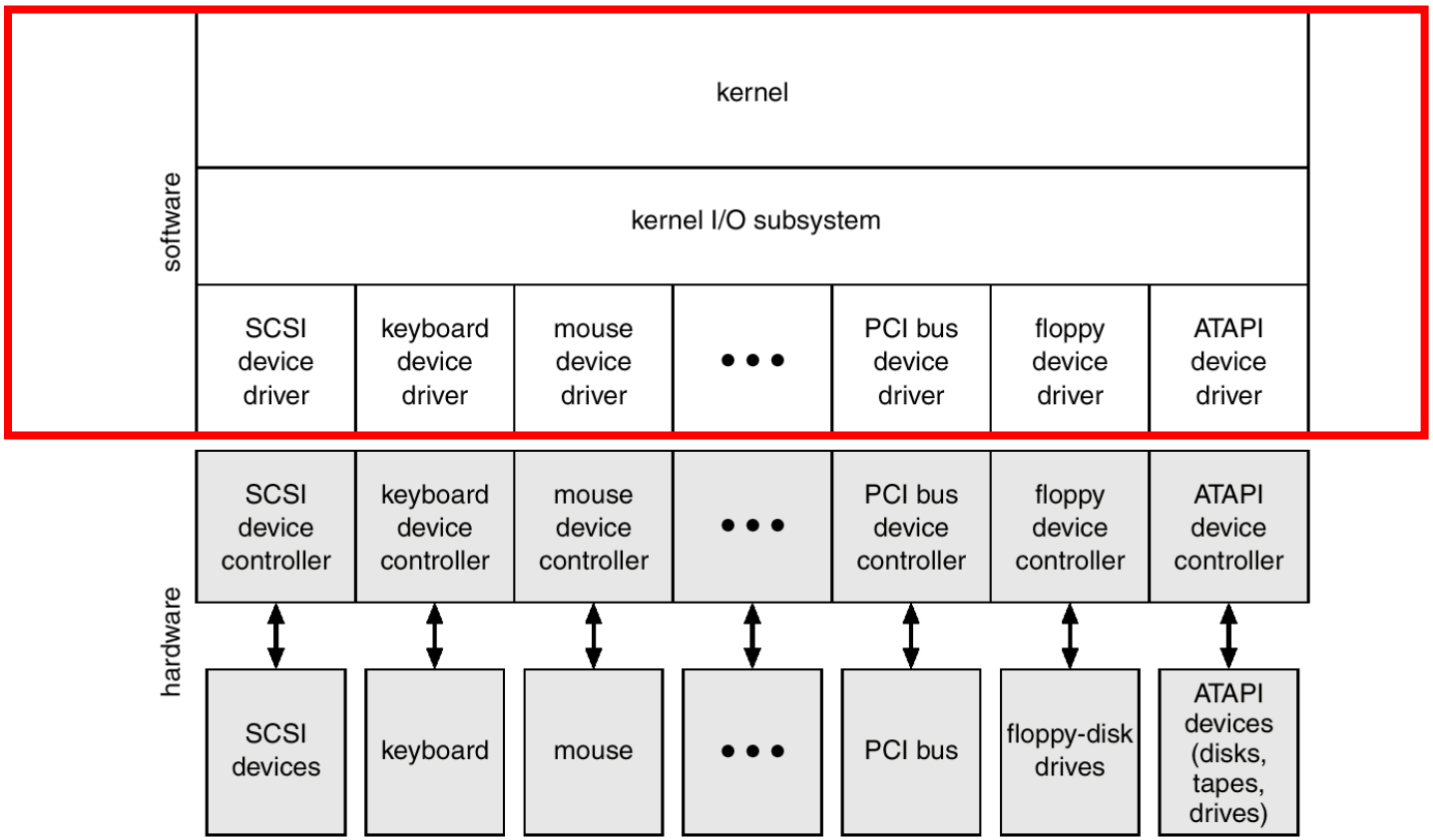


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Layers of the I/O system

5.a Overview of the O/S Role in I/O

- The I/O subsystem is layered



A kernel I/O structure

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

5.a Overview of the O/S Role in I/O

- Chart of operating system responsibilities in I/O

§D – The O/S is responsible for controlling access to all the I/O devices

- ✓ the O/S hides the peculiarities of specific hardware devices from the user
- ✓ the O/S issues the low-level commands to the devices, catches interrupts and handles errors
- ✓ the O/S relies on software modules called “device drivers”
- ✓ the O/S provides a device-independent API to the user programs, which includes buffering

Principles of Operating Systems

CS 446/646

5. Input/Output

a. **Overview of the O/S Role in I/O**

b. **Principles of I/O Hardware**

c. **I/O Software Layers**

d. **Disk Management**

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

- ✓ The diversity of I/O devices
- ✓ I/O bus architecture
- ✓ I/O devices & modules
- ✓ CPU-I/O communication

c. I/O Software Layers

d. Disk Management

5.b Principles of I/O Hardware

The diversity of I/O devices

- Great variety of I/O devices
  - ✓ storage devices
    - disks
    - tapes
  - ✓ transmission devices
    - network cards
    - modems
  - ✓ human-interface devices
    - screen
    - keyboard
    - mouse

5.b Principles of I/O Hardware

The diversity of I/O devices

- I/O devices vary in many dimensions (but these categories have fuzzy boundaries)
  - ✓ main distinction: character-stream vs. block
    - block devices transfer blocks of bytes as units
      - block devices store information in fixed-size blocks
      - blocks can be accessed independently from each other
      - disks are typical block devices; tapes not so typical
    - character devices transfer bytes one by one
      - accepts or delivers a stream of characters without block structure
      - not addressable, not seekable

5.b Principles of I/O Hardware

The diversity of I/O devices

- I/O devices vary in many dimensions (cont'd)
  - ✓ sequential vs. random-access
    - sequential devices transfer in a fixed order they determine
    - random-access devices can be "seeked" at any storage location
  - ✓ synchronous vs. asynchronous
    - synchronous devices have predictable transfer times
    - asynchronous devices are irregular

5.b Principles of I/O Hardware

The diversity of I/O devices

- I/O devices vary in many dimensions (cont'd 2)
  - ✓ sharable vs. dedicated
    - sharable devices may be used concurrently by several processes or threads
    - dedicated devices cannot
  - ✓ speed of operation
    - devices speed range from a few bytes to a few GB per second
  - ✓ read-write, read only, or write only
    - some devices are both input/output, others only one-way

5.b Principles of I/O Hardware

The diversity of I/O devices

- I/O devices vary in many dimensions

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 readÐwrite	CD-ROM graphics controller disk

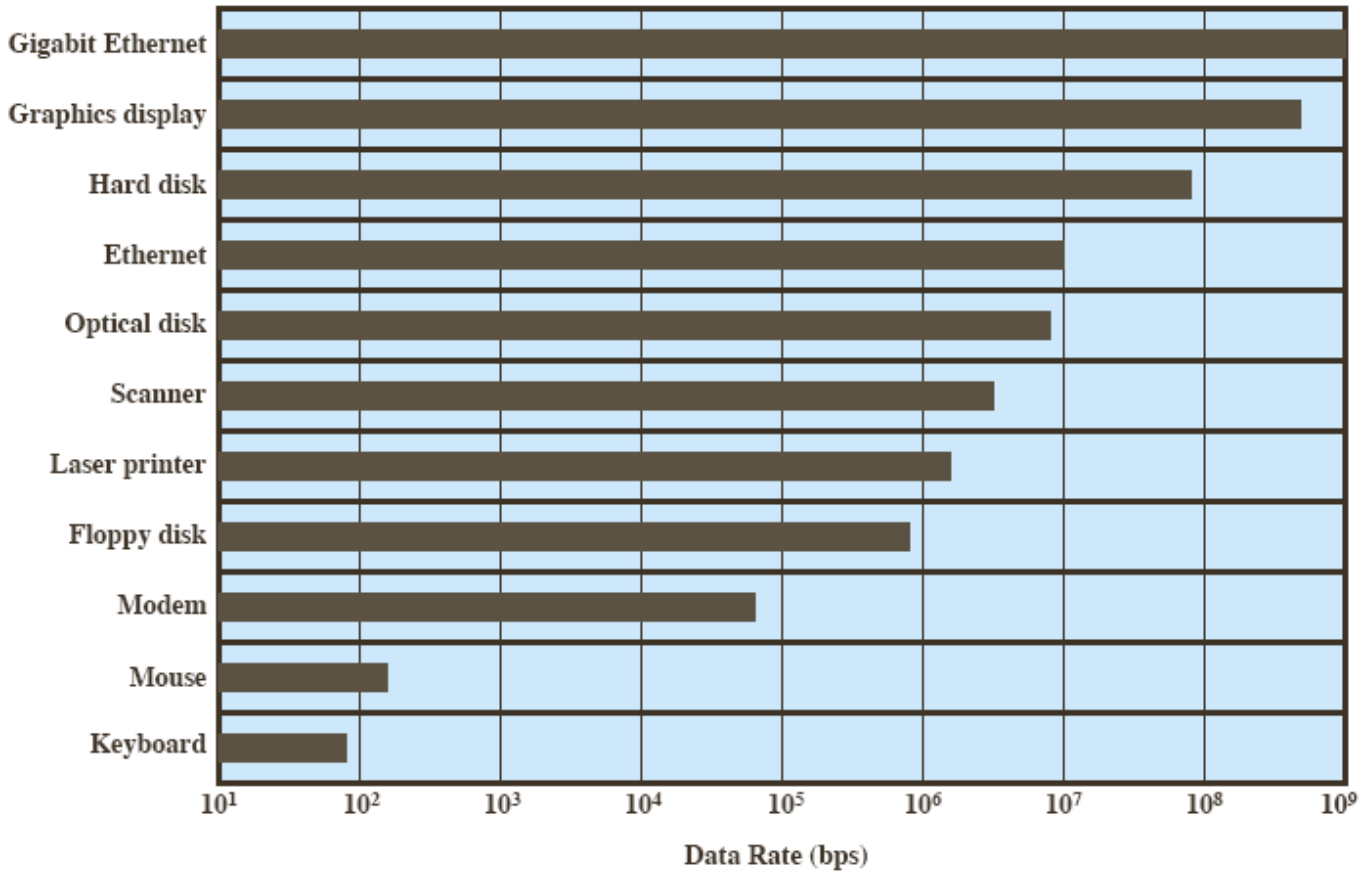
Characteristics of I/O devices

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

5.b Principles of I/O Hardware

The diversity of I/O devices

- I/O devices vary hugely in data transfer speed



Typical I/O device data rates

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

The diversity of I/O devices

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

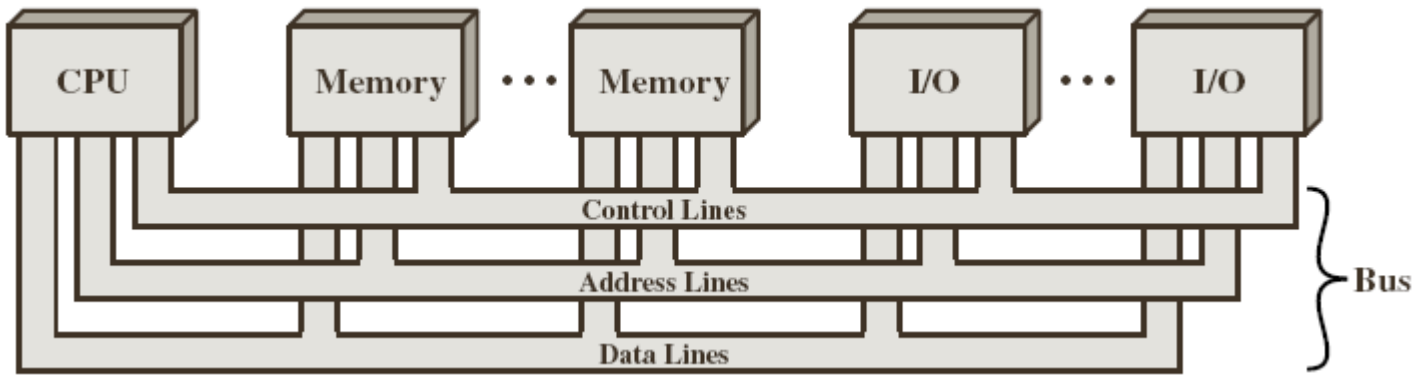
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Some typical device, network, and bus data rates

5.b Principles of I/O Hardware

I/O bus architecture

- CPU, memory and I/O devices communicate via buses
  - ✓ a system bus is the “public transportation” of memory and I/O communication = a set of wires + a message protocol
  - ✓ typically contains hundreds of data, address and control lines
  - ✓ each line carries only 1 bit at a time, therefore the bus width *and* frequency are key factors in performance



Bus interconnection scheme

Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

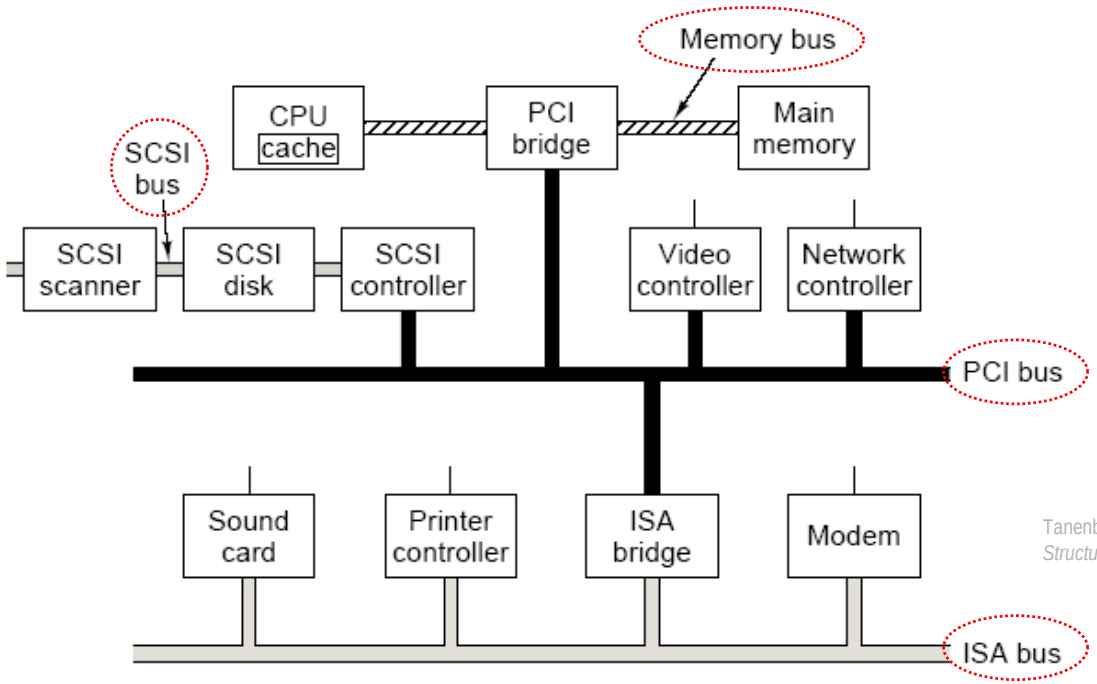
I/O bus architecture

- Typical bus structure
  - ✓ data lines
    - provide a path for moving data between system modules
  - ✓ address lines
    - used to designate the source or destination of the data
  - ✓ control lines
    - transmit commands and timing information between modules
    - memory read/write, I/O read/write, bus request/grant, etc.

5.b Principles of I/O Hardware

I/O bus architecture

- Typical bus interconnection layout
  - ✓ computer systems contain multiple types of buses at different levels of the hierarchy
  - ✓ memory bus, SCSI, ISA, PCI, USB, FireWire, etc.

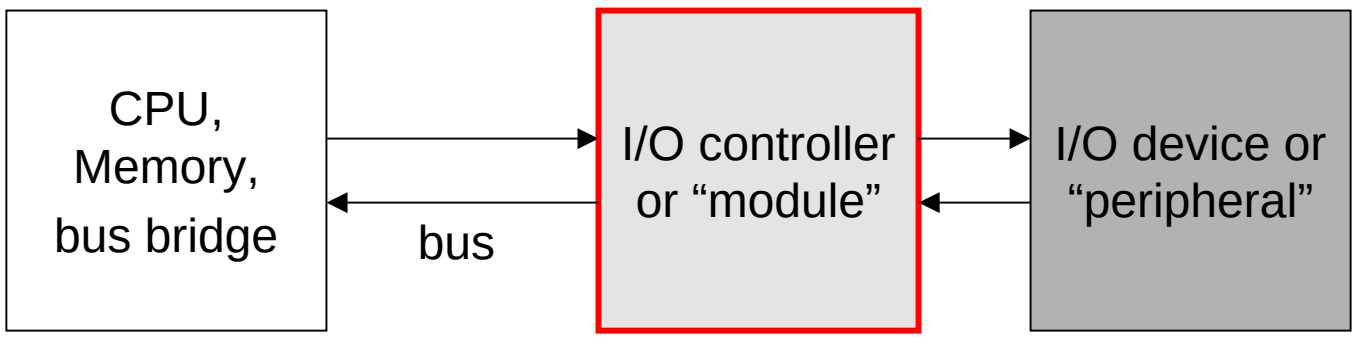


Tanenbaum, A. S. (2006)
Structured Computer Organization (5th Edition).

5.b Principles of I/O Hardware

I/O bus architecture

- Basic hardware I/O communication architecture
  - ✓ each I/O device consists of two parts:
    - the **controller** or **module**, containing most of the electronics
    - the device proper, such as a disk drive
  - ✓ the job of the controller is (a) to control the I/O and (b) handle bus access for it



5.b Principles of I/O Hardware

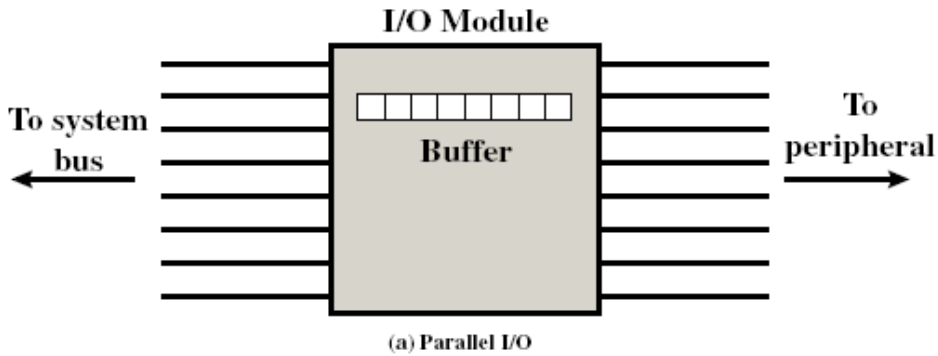
I/O devices & modules

- Why I/O modules? Why not connecting the devices directly to the bus?
  - ✓ wide variety of peripherals with various operation methods: don't want to incorporate heterogeneous logic into CPU
  - ✓ modules offer a more unified hardware command interface
  - ✓ data transfer rate slower or faster than memory or CPU
  - ✓ different data and word lengths
  - ✓ multiplexing: one module serving several devices (ex: SCSI)
- Functions of an I/O module
  - ✓ interface to CPU and memory via system bus
  - ✓ interface to one *or more* peripherals by custom data links

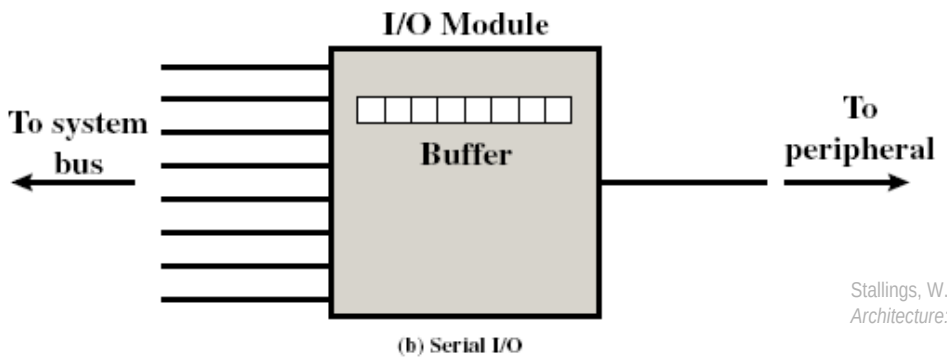
5.b Principles of I/O Hardware

I/O devices & modules

- Two main categories of I/O module-device interface



(a) Parallel I/O



(b) Serial I/O

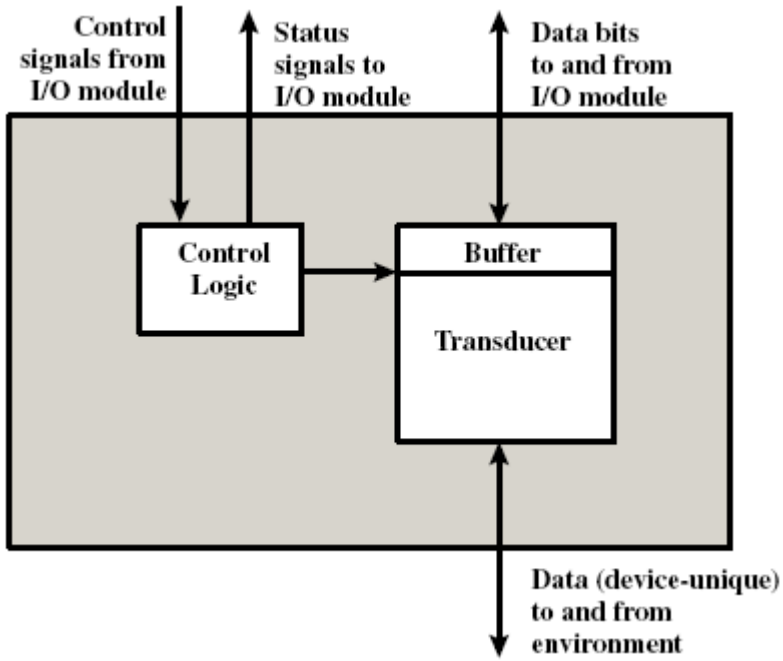
Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

Parallel and serial I/O

5.b Principles of I/O Hardware

I/O devices & modules

- Schematic structure of an I/O device
  - ✓ interface to the I/O module: control, data and status signals
  - ✓ interface with the physical/electrical apparatus



Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

Block diagram of an I/O device

5.b Principles of I/O Hardware

I/O devices & modules

- Typical I/O interface with the the I/O module ("host")
  - ✓ control registers
    - can be *written* by the host to start a command or change the mode of the device
  - ✓ status registers
    - contain bits *read* by the host that indicate whether a command has completed, a byte is available to be read from the data-in register, or there has been a device error
  - ✓ data registers (buffer)
    - data-in registers are read by the host to get input
    - data-out registers are written by the host to send output

5.b Principles of I/O Hardware

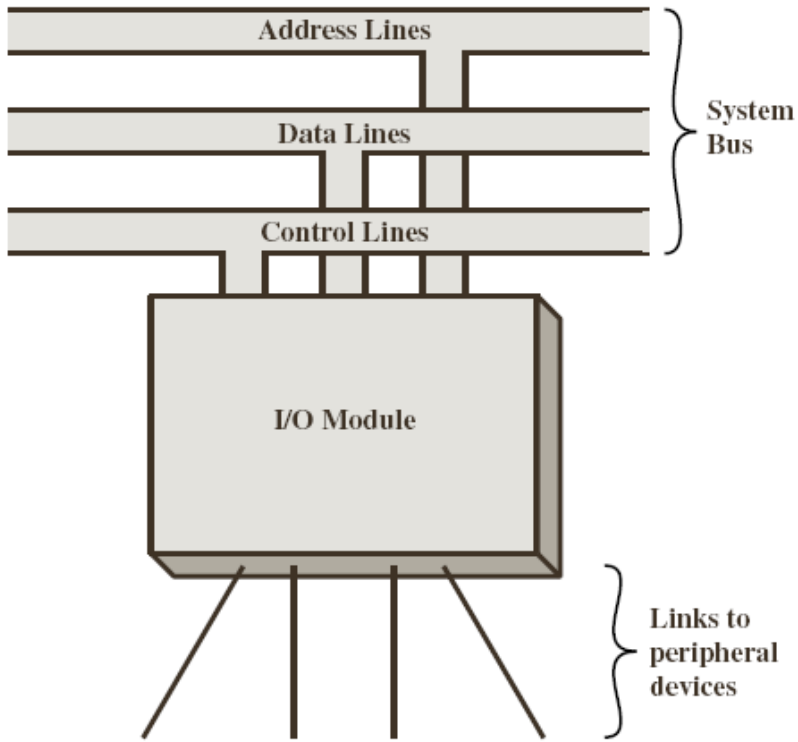
I/O devices & modules

- I/O interface with the physical/electrical apparatus
  - ✓ transducer
    - converts between binary data and analog electro-mechanical events specific to the device
    - ex: pressing a key on the keyboard generates an electronic signal and transduced into the ASCII bit pattern
    - ex: in a disk, bits in the device's buffer are transduced from/to magnetic patterns on the moving disk

5.b Principles of I/O Hardware

I/O devices & modules

- I/O controllers or "modules"
  - ✓ intermediate between the I/O device (peripheral) and CPU or memory



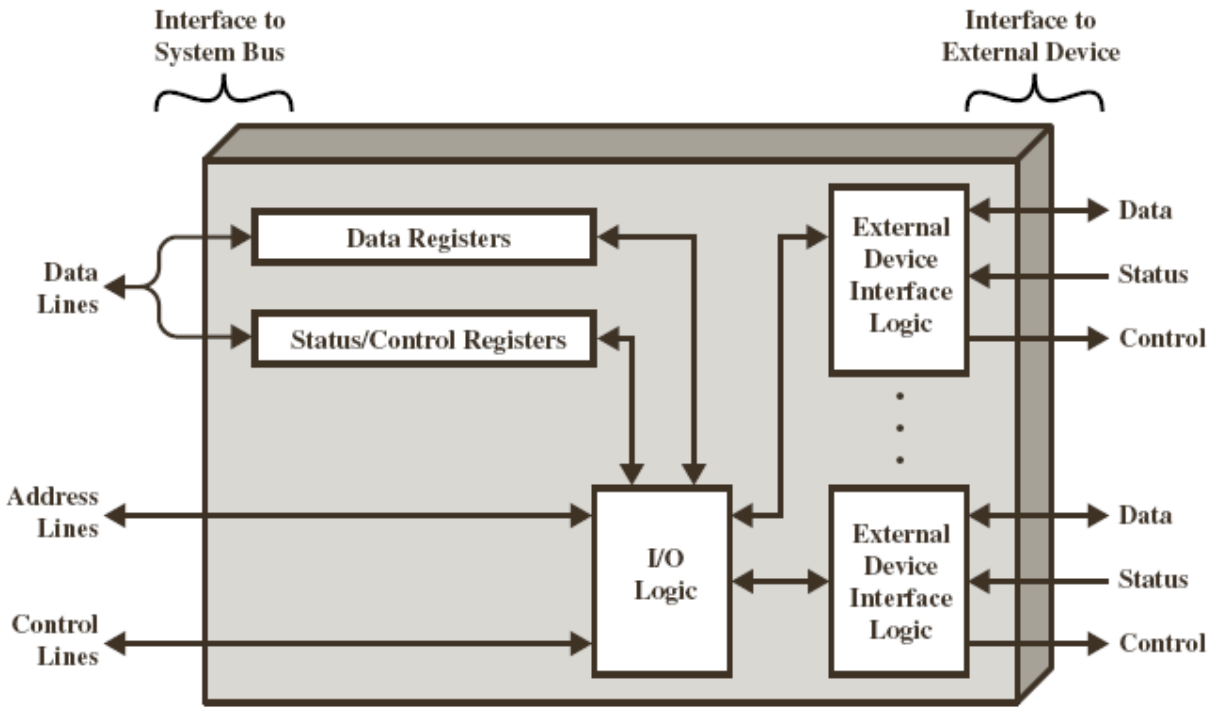
Generic model of an I/O module

Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

I/O devices & modules

- Schematic structure of an I/O module
  - ✓ interface also based on control, status and data lines
  - ✓ basically an adapter/multiplexer with a **buffer** (data registers)



Block diagram of an I/O module

Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

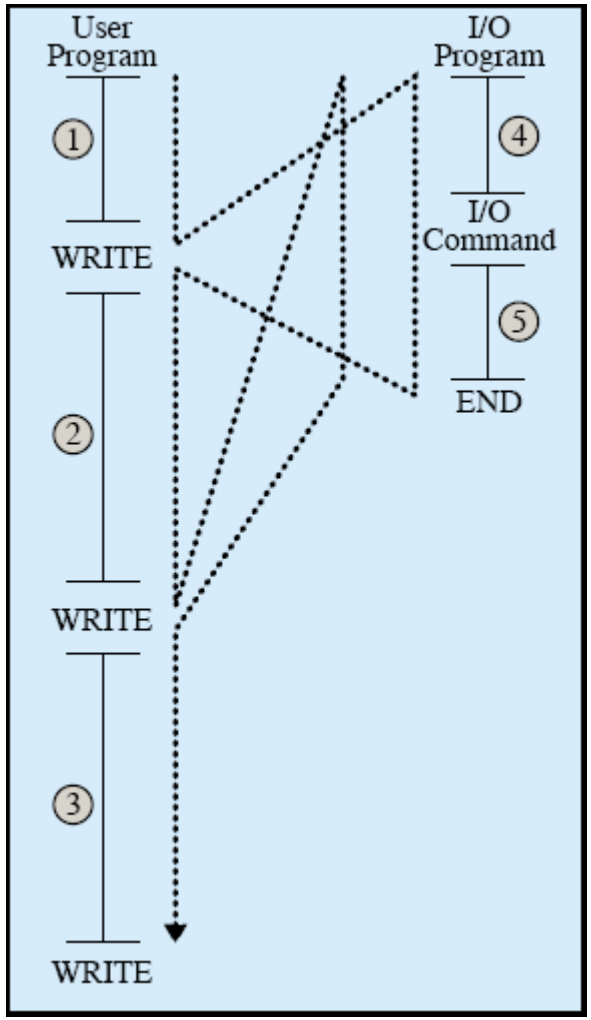
- Three communication protocols between CPU and I/O
  1. Programmed I/O (or “polling” or “busy waiting”)
     - the CPU must repeatedly poll the device to check if the I/O request completed
  2. Interrupt-driven I/O
     - the CPU can switch to other tasks and is (frequently) interrupted by the I/O device
  3. Direct Memory Access (DMA)
     - the CPU is involved only at the start and the end of the whole transfer; it delegates control to the I/O controller that accesses memory directly without bothering the CPU

5.b Principles of I/O Hardware

CPU-I/O communication

1. Programmed I/O

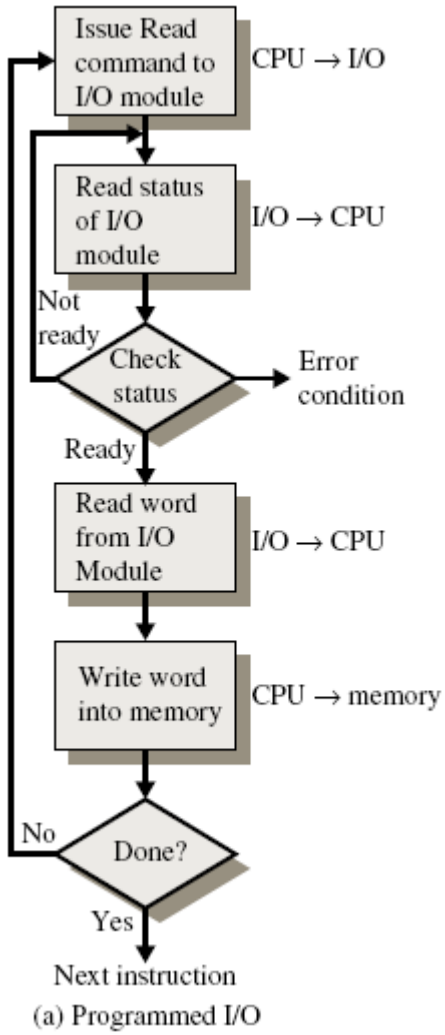
- ✓ the CPU issues an I/O command (on behalf of a process) to an I/O module
- ✓ the CPU (the process) then busy waits for completion before proceeding
- ✓ also called “busy waiting” or “polling”



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication



(a) Programmed I/O

Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

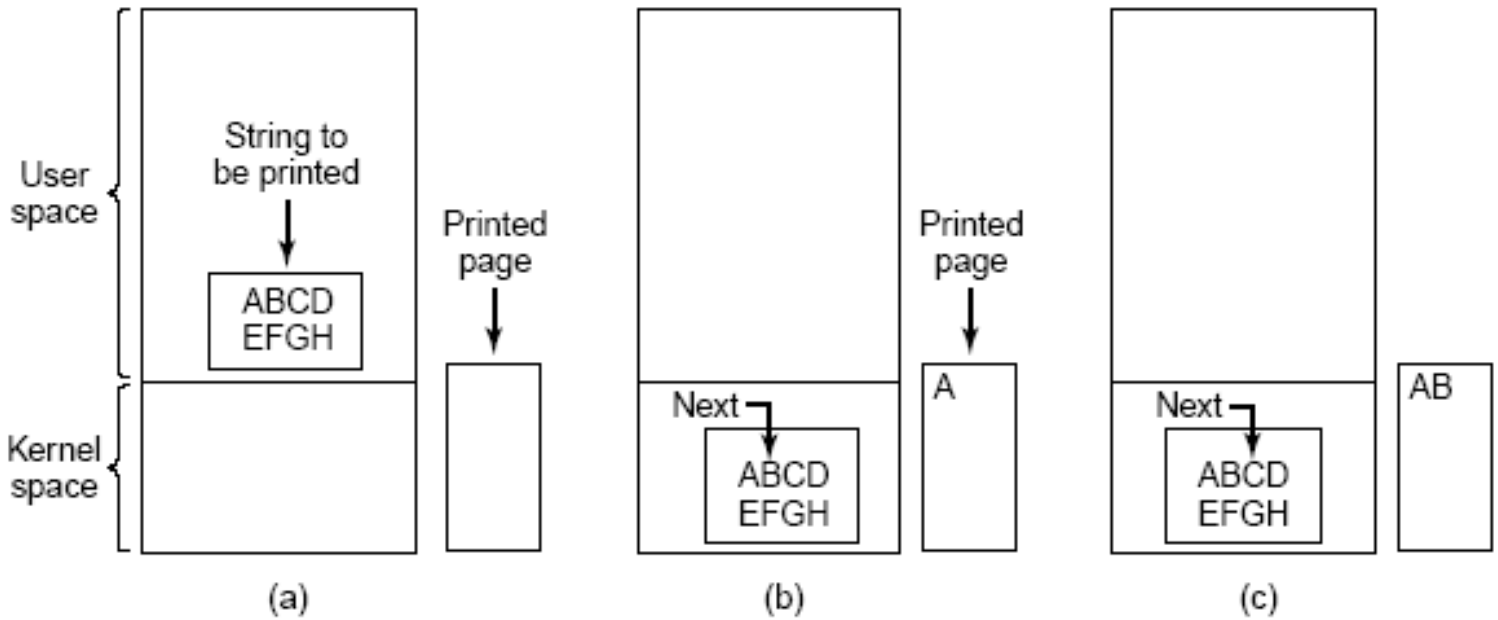
1. Programmed I/O

- ✓ basic handshake protocol between CPU and I/O module
  a. host repeatedly polls *occupied* bit in status register of I/O module until cleared (this is the busy waiting part)
  b. host sets *write* bit in *command* register of I/O module and writes byte into *data-out* register of I/O module
  c. host sets *command-ready* bit of I/O module
  d. I/O module notices *command-ready* bit and sets *occupied* bit
  e. I/O module reads *command* and *data-out* registers and orders I/O device to perform I/O
  f. when suceeded, I/O module clears *command-ready* bit, *error* bit and *occupied* bit

5.b Principles of I/O Hardware

CPU-I/O communication

- Example: writing a string to the printer



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Steps in printing a string

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of “driver” logic: writing a string to the printer

1. . . . using programmed I/O:

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
      while (*printer_status_reg != READY) ;/* loop until ready */
      *printer_data_register = p[i];       /* output one character */
}
return_to_user( );
```

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Programmed I/O code

5.b Principles of I/O Hardware

CPU-I/O communication

1. Programmed I/O problems

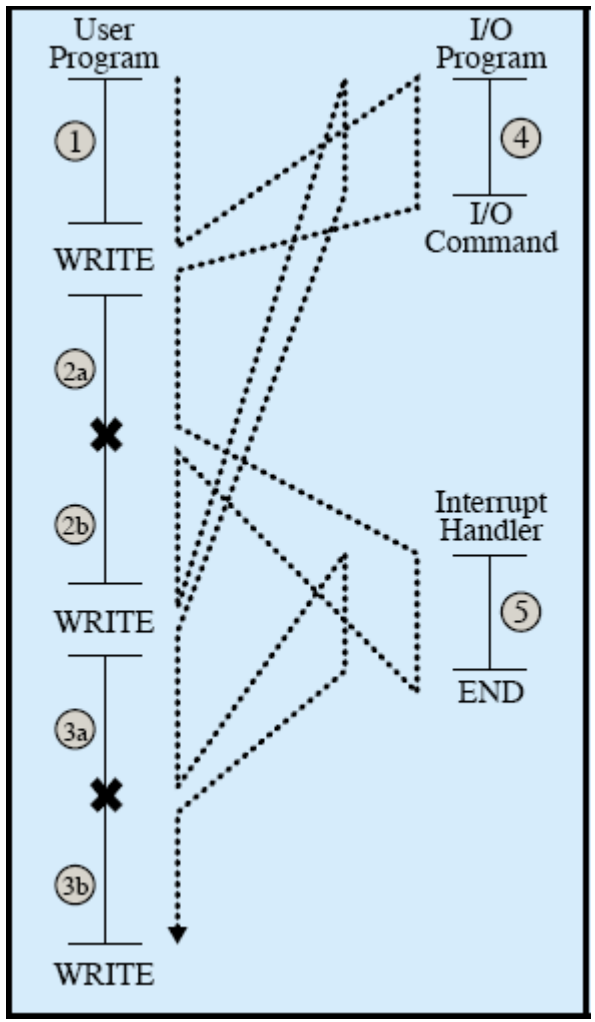
- ✓ the I/O device (module) is passive and needy: it does not alert the CPU that it is ready and does not transfer data to/from memory by itself
- ✓ the CPU needs to continually check the I/O status and data registers
  - to minimize the CPU waiting time
  - but also to avoid overflow in the small buffer of the controller: needs to be regularly cleared
- ✓ naturally this is a waste of CPU time if the I/O transfer is slower than the CPU. . . which it always is!
- ✓ no longer an option today

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

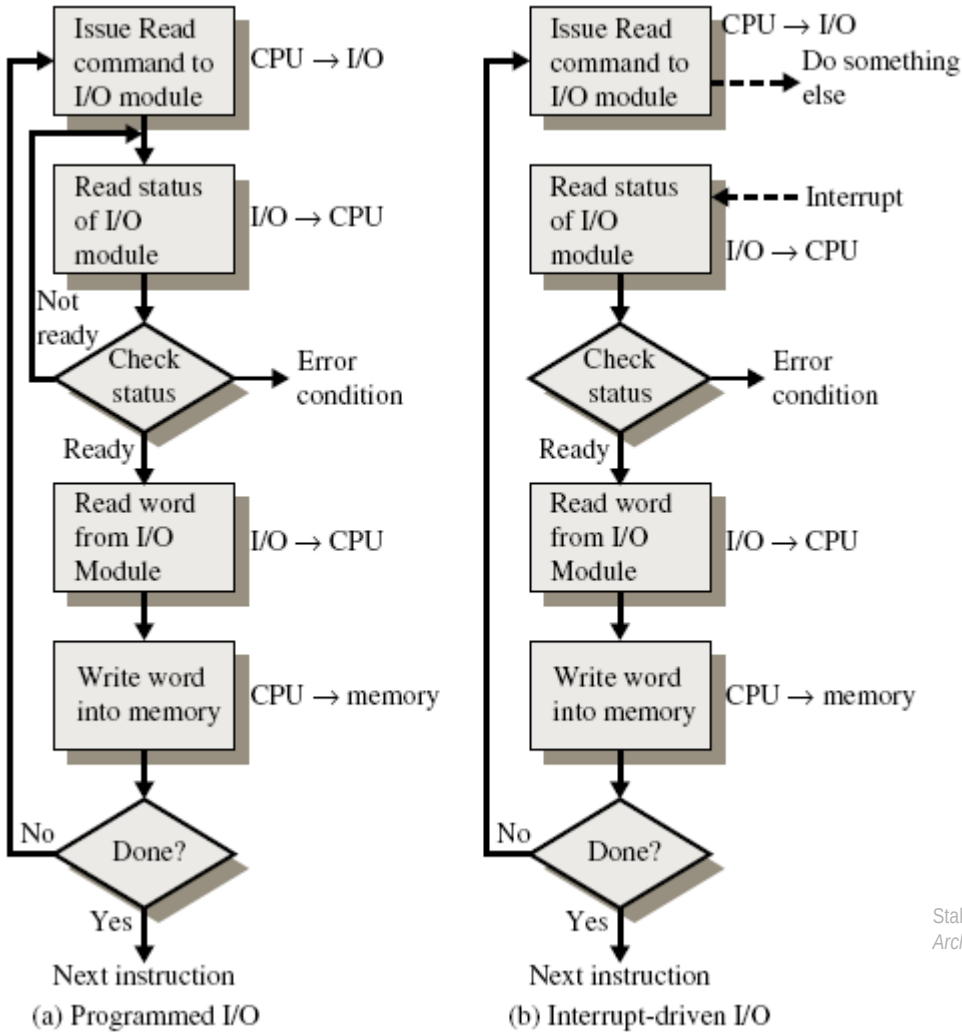
- ✓ the CPU issues an I/O command (on behalf of a process) to an I/O module
- ✓ . . . but does not wait for completion; instead, it continues executing subsequent instructions
- ✓ then, later, it is interrupted by the I/O module when work is complete
- ✓ note: the subsequent instructions may be in the same process or not, depending on whether I/O was requested asynchronously or not: *process wait ≠ CPU wait!*



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication



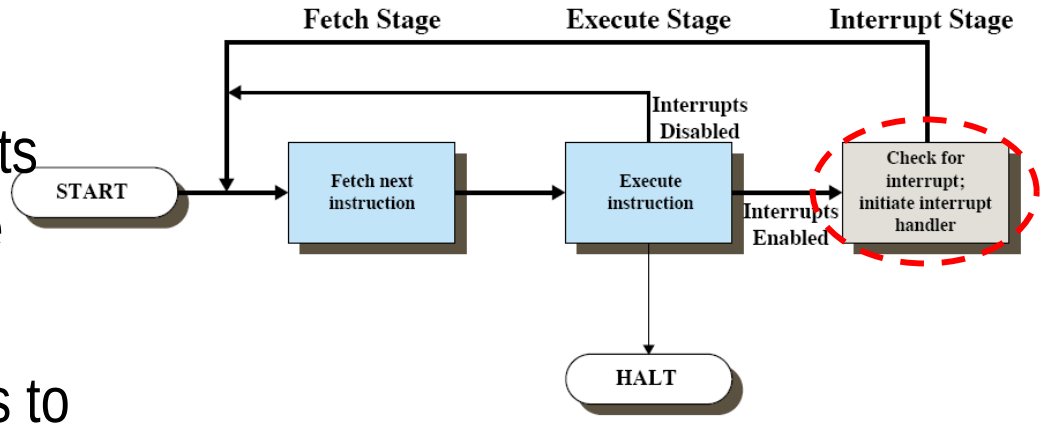
Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

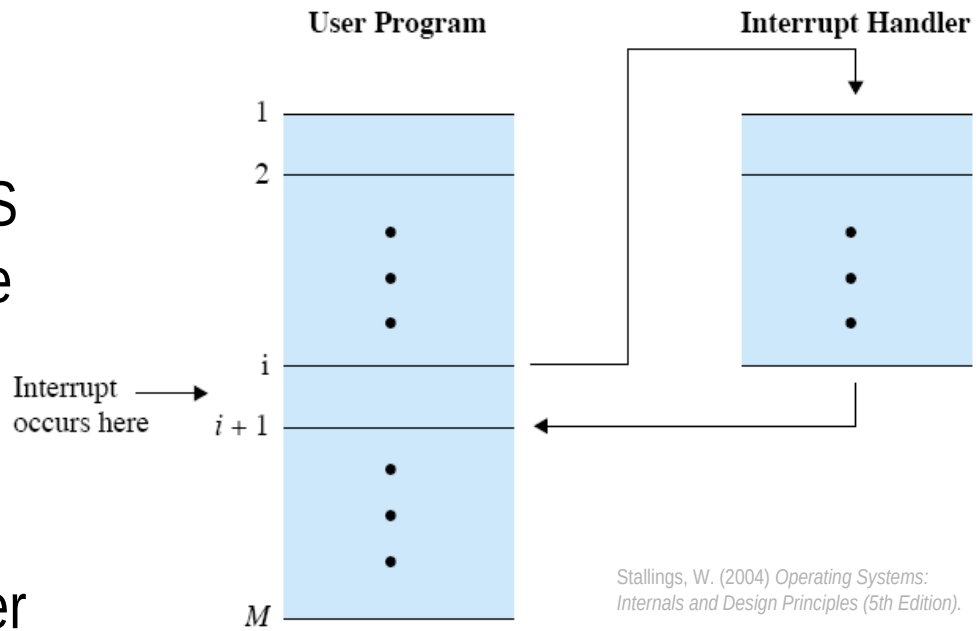
5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

- ✓ CPU senses interrupts in a third stage of the fetch/execute cycle
- ✓ control (PC) transfers to an interrupt handler in kernel space,
- ✓ which branches to O/S routines specific to the type of interrupt;
- ✓ the CPU is eventually returned to this user program . . . or another





Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of "driver" and "interrupt handler" logic: writing a string to the printer

2. . . . using interrupts:

driver logic

```
                copy_from_user(buffer, p, count);
                enable_interrupts( );
only            while (*printer_status_reg != READY) ;
1st char        *printer_data_register = p[0];
                scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

interrupt handler logic

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Interrupt-driven I/O code: (a) system call and (b) interrupt service procedure

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

- ✓ relies on an efficient hardware mechanism that saves a small amount of CPU state, then calls a privileged kernel routine
- ✓ note that this hardware mechanism is put to good use by the O/S for other events:
  - in virtual memory paging, a page fault is an exception that raises an interrupt
  - system calls execute a special instruction (TRAP), which is a software interrupt

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O problems

- ✓ the I/O device (module) is more active but still very needy
- ✓ wasteful to use an expensive general-purpose CPU to feed a controller 1 byte at a time

5.b Principles of I/O Hardware

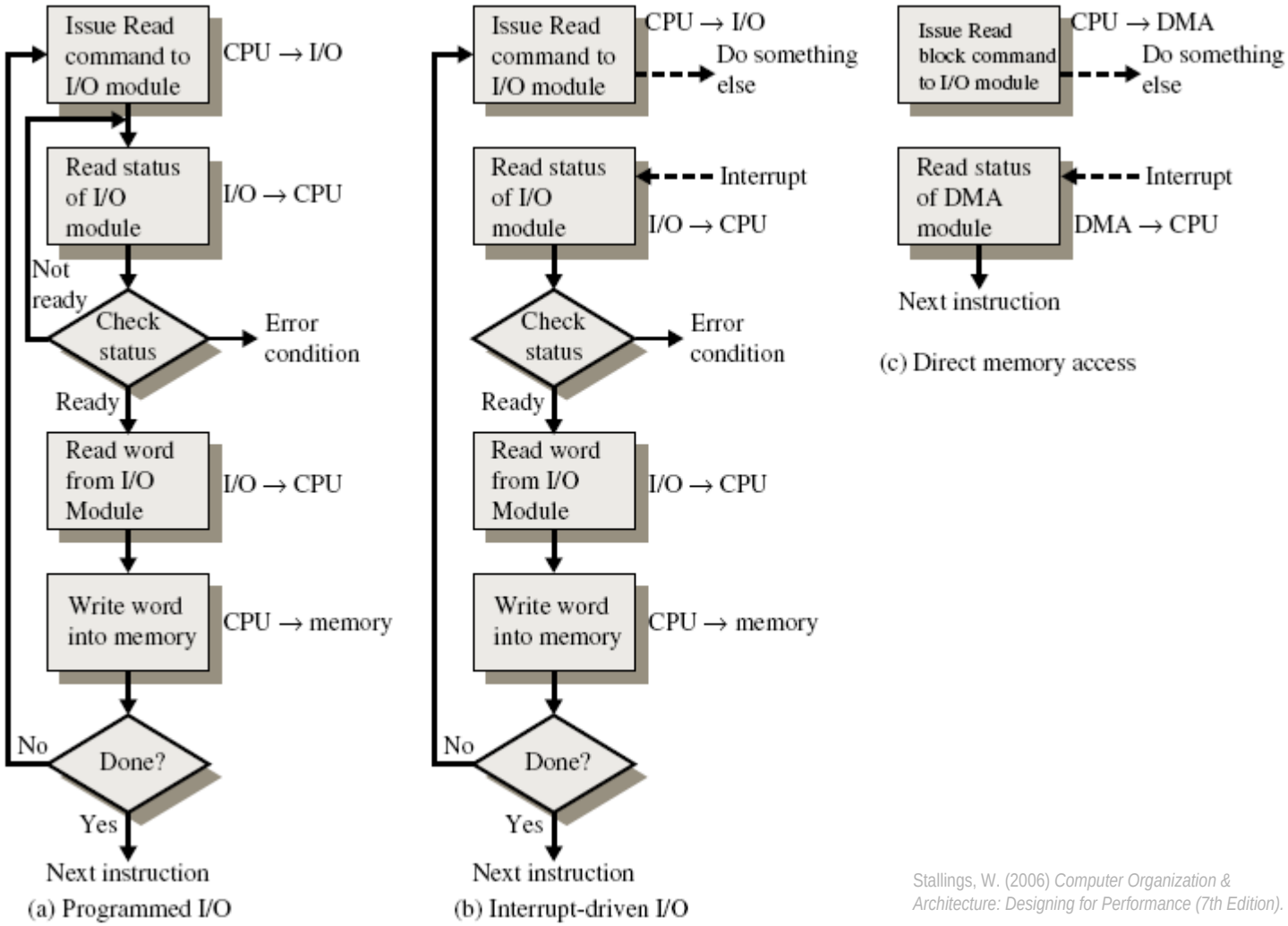
CPU-I/O communication

3. Direct Memory Access (DMA)

- ✓ avoids programmed/interrupted I/O for large data movement
- ✓ requires a special-purpose processor called DMA controllerbypasses CPU to transfer data directly between I/O device and memory
- ✓ the handshaking is performed between the DMA controller and the I/O module: in essence, the DMA controller is going to do the programmed I/O instead of the CPU
- ✓ only when the entire transfer is finished does the DMA controller interrupt the CPU

5.b Principles of I/O Hardware

CPU-I/O communication

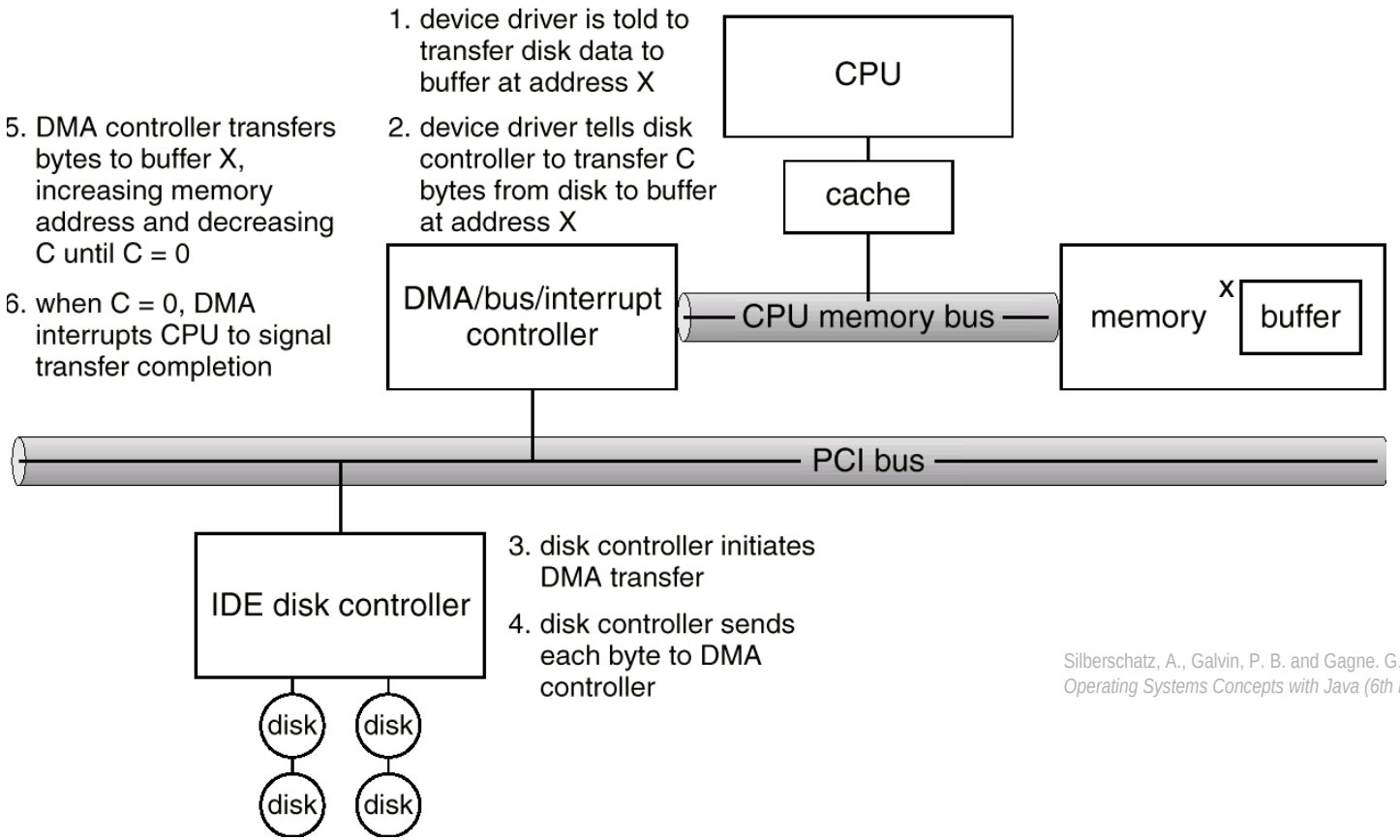


Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

3. Direct Memory Access (DMA)



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Steps in a DMA transfer

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of "driver" logic: writing a string to the printer

3. . . . using DMA:

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

DMA-supported I/O code: (a) system call and (b) interrupt service procedure

5.b Principles of I/O Hardware

CPU-I/O communication

- Summary

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

- ✓ The diversity of I/O devices
- ✓ I/O bus architecture
- ✓ I/O devices & modules
- ✓ CPU-I/O communication

c. I/O Software Layers

d. Disk Management

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

c. I/O Software Layers

- ✓ Overview of the I/O software
- ✓ Interrupt handlers
- ✓ Device drivers
- ✓ Device-independent I/O software
- ✓ User-level I/O system calls

d. Disk Management

5.c I/O Software Layers

Overview of the I/O software

- Goals and services of the I/O software
  - ✓ device independence
    - write programs that can access I/O devices without specifying them or knowing them in advance
    - ex: reading a file from a disk, whether floppy, magnetic, CD-ROM, etc.
    - no need to modify the program if a new device comes in
  - ✓ uniform naming (“mounting”)
    - abstract naming space independent from physical device
    - naming should be a string and/or integer ID, again without device awareness

5.c I/O Software Layers

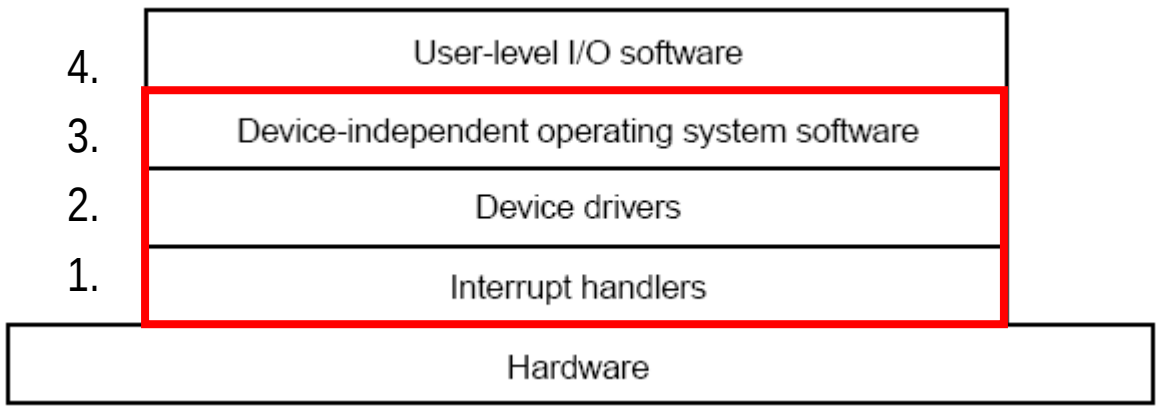
Overview of the I/O software

- Goals and services of the I/O software
  - ✓ error handling
    - lower layers try to handle the error before upper levels
    - controller hardware should correct error first; if it cannot, then driver software (for ex. by reissuing the command), etc.
    - upper levels can remain unaware of “bumps” at lower levels
  - ✓ synchronous vs. asynchronous transfers
    - most physical I/O is asynchronous (interrupt-driven)
    - O/S should make it look synchronous (blocking) to processes
  - ✓ buffering
    - decouple transfer rates and insulate data from swapping

5.c I/O Software Layers

Overview of the I/O software

- The I/O component of the O/S is organized in layers
  1. interrupt handlers
  2. device drivers
  3. device-independent I/O
  4. user-level I/O system calls



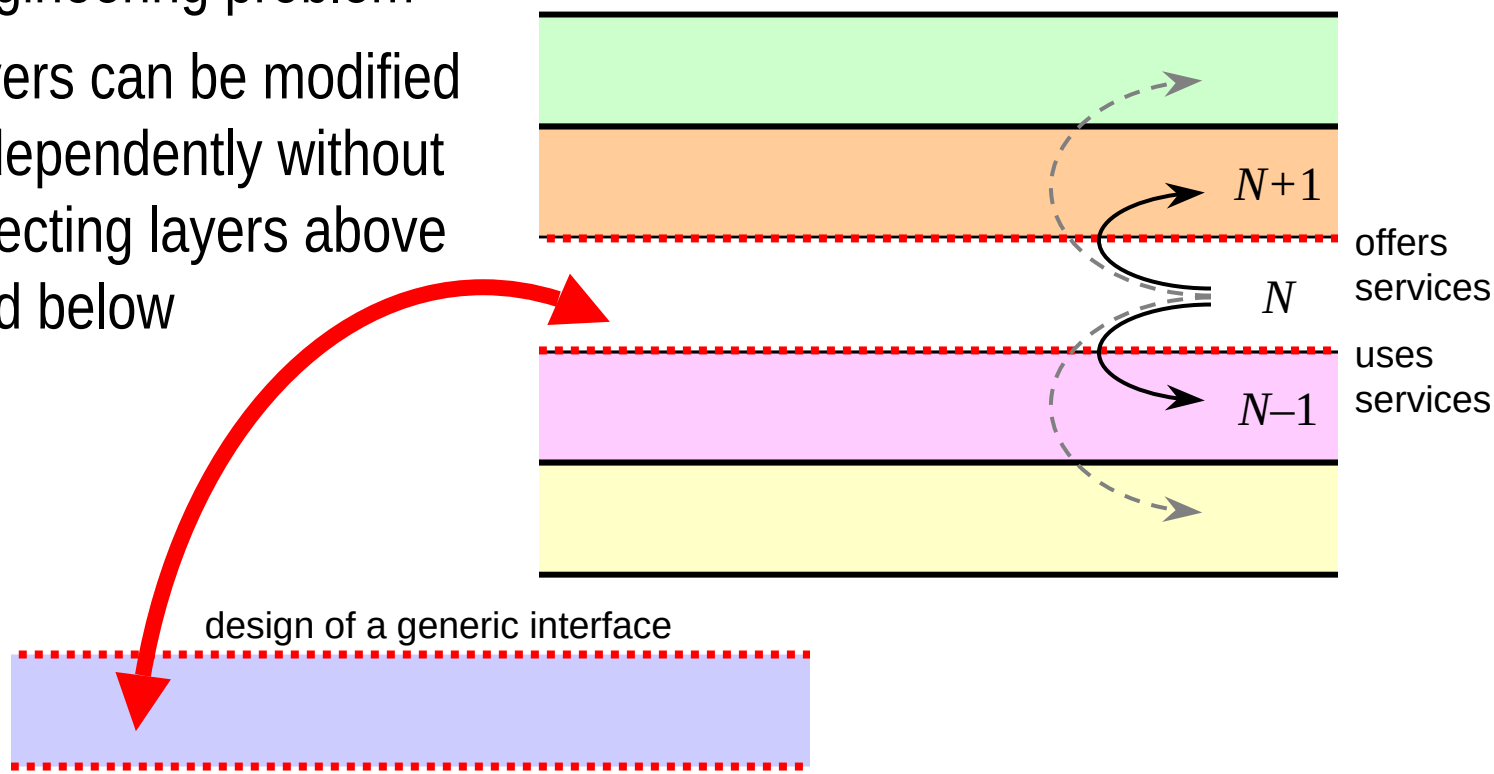
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Typical layers of the I/O software subsystem

5.c I/O Software Layers

Overview of the I/O software

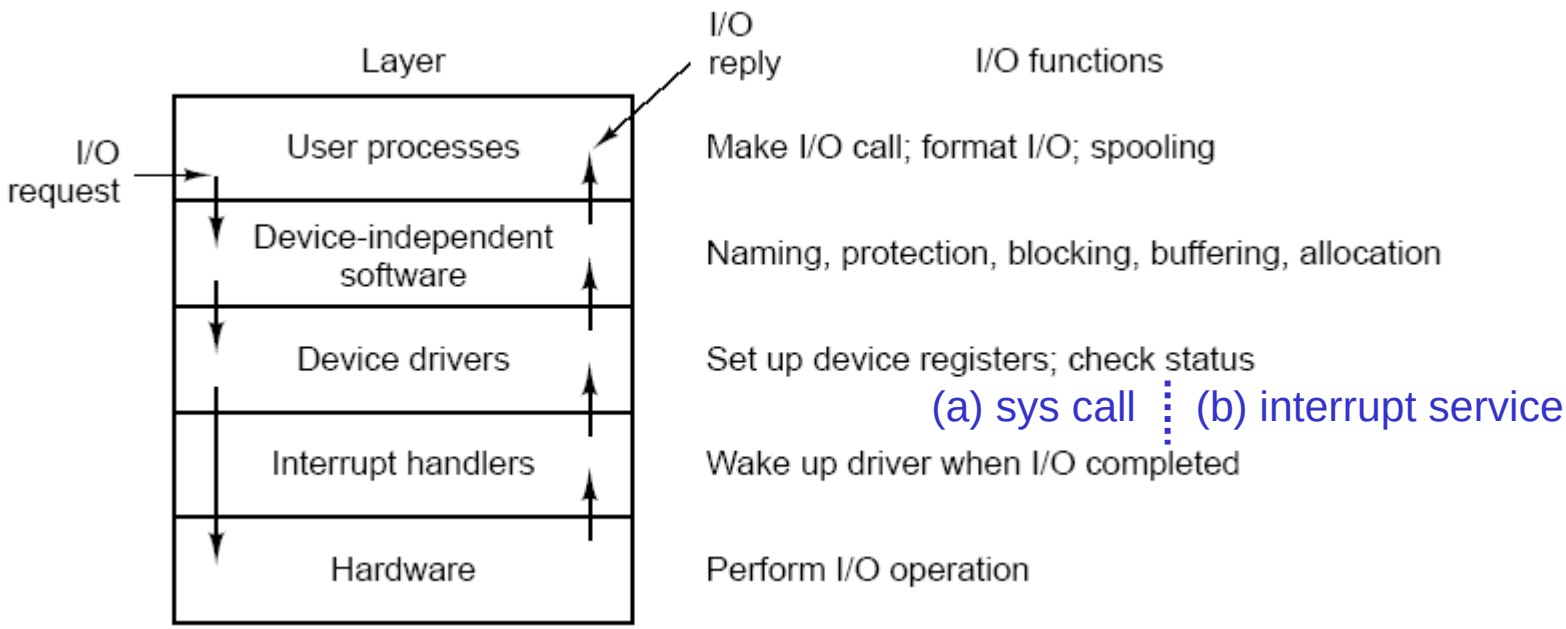
- Abstraction, encapsulation and layering
  - ✓ any complex software engineering problem
  - ✓ layers can be modified independently without affecting layers above and below



5.c I/O Software Layers

Overview of the I/O software

- Typical flow of control through the I/O layers upon an I/O request



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

5.c I/O Software Layers

Interrupt handlers

1. Interrupt handler routines

- ✓ interrupts (asynchronous, external to process) basically use the same mechanism as exceptions and traps (synchronous, internal to process)
- ✓ when an interrupts happen, the CPU saves a small amount of state and jumps to an interrupt-handler routine at a fixed address in memory
- ✓ the interrupt routine's location is determined by an interrupt vector

5.c I/O Software Layers

Interrupt handlers

1. Interrupt handler routines (cont'd)

nonmaskable, used for various error conditions

maskable, used for device-generated interrupts

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

Intel Pentium processor event-vector table

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

5.c I/O Software Layers

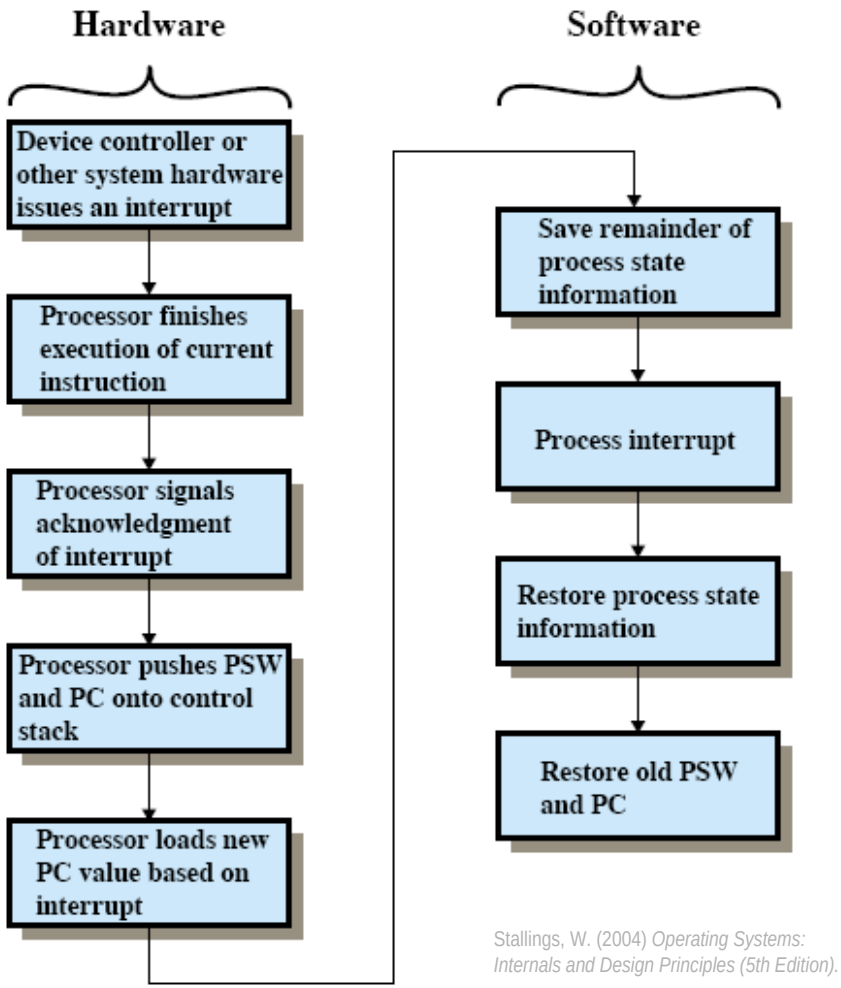
Interrupt handlers

1. Interrupt handler routines

- ✓ typical steps followed by an interrupt routine:
  a. save any registers not saved by the interrupt hardware
  b. set up a context (TLB, MMU, page table) for the routine
  c. set up a stack for the routine
  d. acknowledge the interrupt controller
  e. extract information from the I/O device controller's registers
  f. etc.
- ✓ interrupt processing is a complex operation that takes a great number of CPU cycles, especially with virtual memory

5.c I/O Software Layers

Interrupt handlers



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Simple interrupt processing

5.c I/O Software Layers

Device drivers

2. Device drivers

- ✓ each I/O device needs a **device-specific code** to control it
- ✓ device manufacturers supply drivers for several popular O/S
- ✓ a driver handles one type of device or one class (ex: SCSI)
- ✓ the driver logic is generally executed in kernel space (although microkernel architectures might push it in user space)
- ✓ drivers should "snap into place" in the kernel through device-independent interfaces (see next section)
- ✓ two main categories of drivers (two higher-level interfaces)
  - block-device drivers: disks, etc.
  - character-device drivers: keyboards, printers, etc.

5.c I/O Software Layers

Device drivers

2. Device drivers (cont'd)

- ✓ a driver has several functions
  - accept abstract read/write requests from the device-independent software above and translate them into concrete I/O-module-specific commands
  - schedule requests: optimize queued request order for best device utilization (ex: disk arm)
  - initialize the device, if needed
  - manage power requirements
  - log device events

5.c I/O Software Layers

Device drivers

2. Device drivers (cont'd)

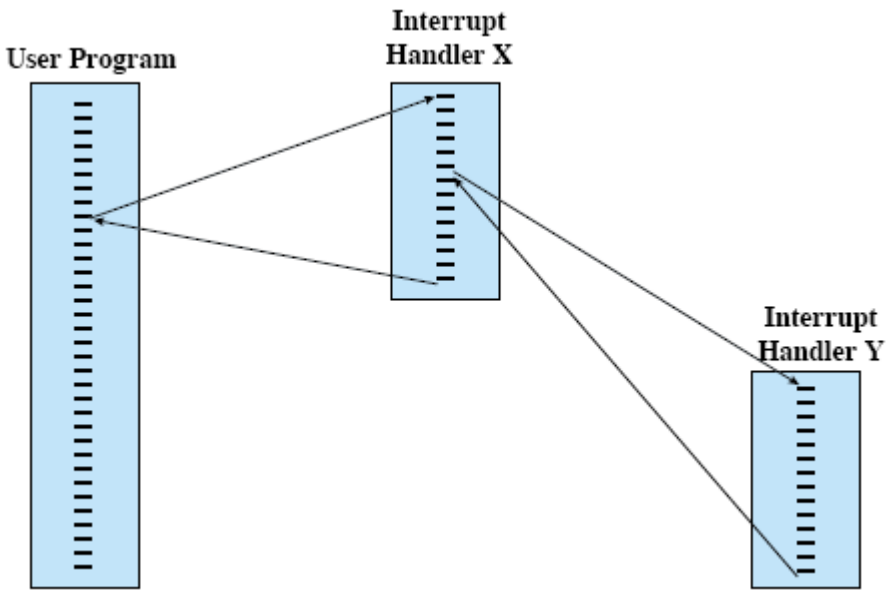
- ✓ typical code organization of a device driver:
  a. check validity of input parameters coming from above
  b. if valid, translate to concrete commands, e.g., convert block number to head, track & sector in a disk's geometry
  c. check if device currently in use; if yes, queue request; if not, possibly switch device on, warm up, initialize, etc.
  d. issue appropriate sequence of commands to controller
  e. if needs to wait, block
  f. upon interrupted from blocking, check for errors and pass data back
  g. process next queued request

5.c I/O Software Layers

Device drivers

2. Device drivers (cont'd)

- ✓ a driver code must be reentrant to allow for nested interrupts
- ✓ a driver must expect to be called a 2nd time before the 1st call is finished



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Nested interrupt processing

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software

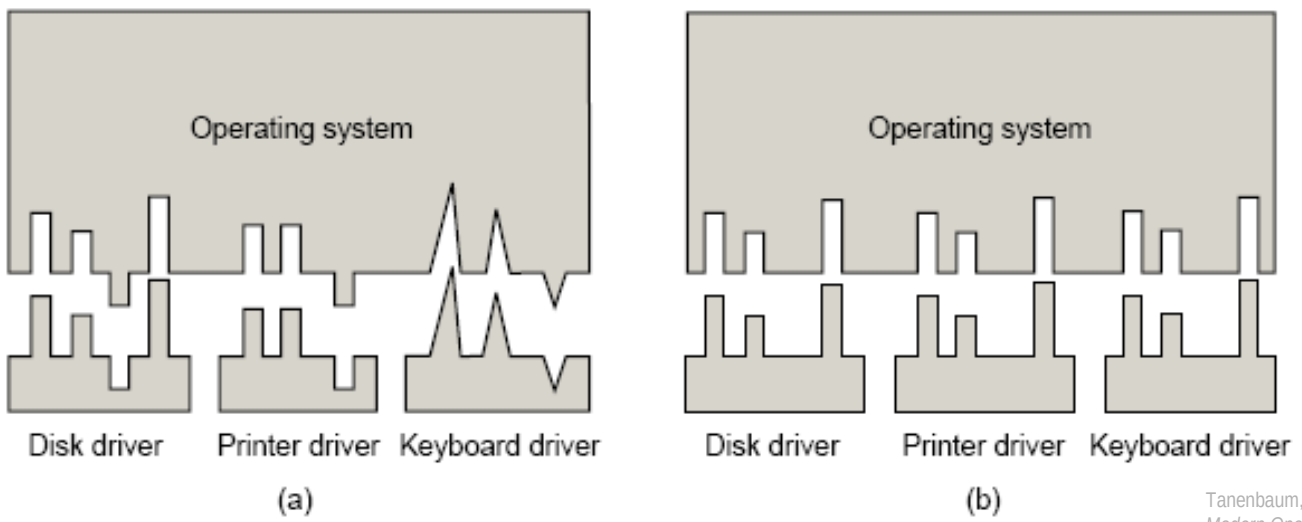
- ✓ generic functions provided by the kernel I/O subsystem:
  - uniform interfacing for device drivers
  - buffering
  - error reporting
  - providing a device-independent block size

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

- ✓ uniform interfacing
  - make all I/O devices look more or less the same, so that the O/S doesn't need to be hacked every time a new device comes along



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

(a) Without and (b) with a standard driver interface

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

- ✓ uniform interfacing
  - therefore, generally one unified interface
  - possibly additional specialized extensions for the main device categories
    - block devices: **`read()`**, **`write()`**
    - random-access block devices: **`seek()`**
    - character-stream devices: **`get()`**, **`put()`**
    - network devices: network socket interface similar to file system

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

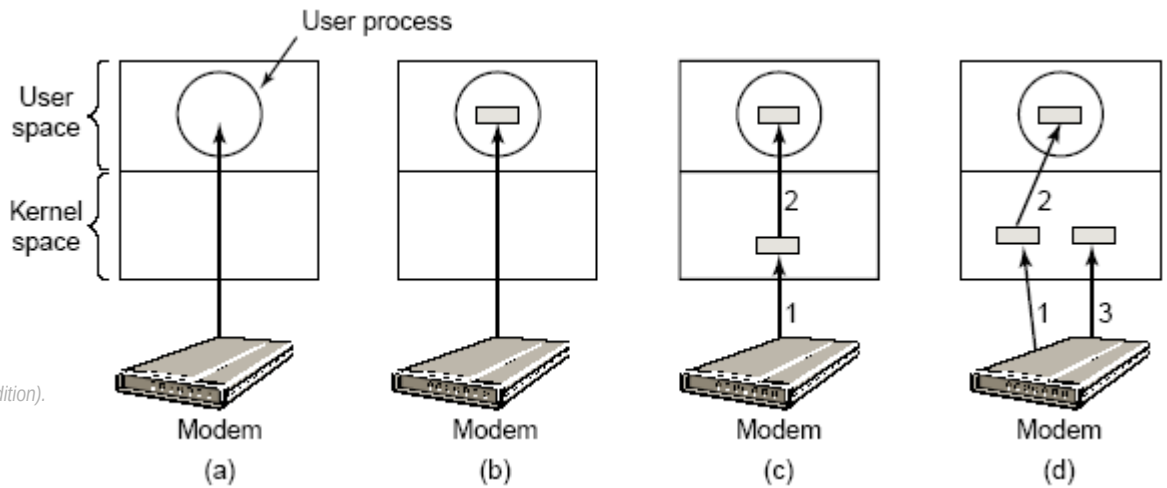
- ✓ buffering = "decoupling"
  - memory area that stores data in kernel space while transferred between device and application
  - cope with a speed mismatch between producer and consumer (ex: modem thousand times slower than disk)
  - adapt between services with different data-transfer sizes (ex: fragmentation and reassembly of network packets)
  - "copy semantics": cache data while transferred so it is not affected by changes from application or swapping
  - read ahead (locality principle)

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

- ✓ buffering
  - a) unbuffered input → *context switch for each transferred byte*
  - b) buffering in user space → *what happens if paged out?*
  - c) buffering in kernel, copy to user space → *what if buffer full?*
  - d) <u>double-buffering in kernel</u>



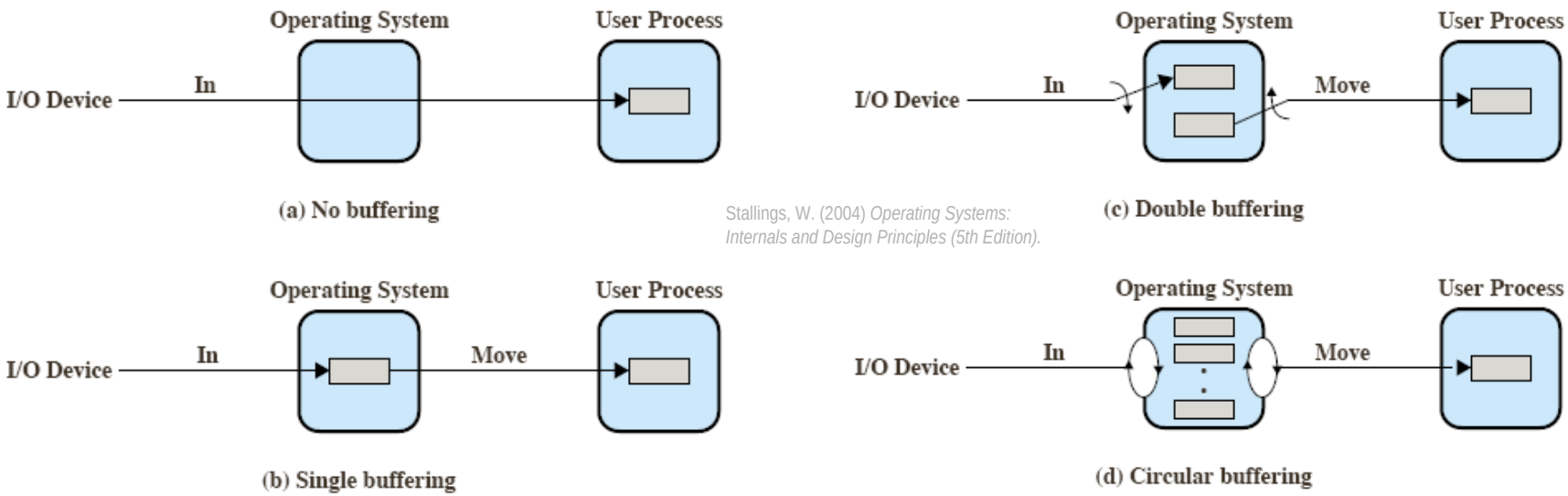
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

- ✓ buffering
  - double buffering: further decouples producer from consumer (ex: modem fills 2nd buffer while 1st buffer is written to disk)
  - circular buffering: extension suitable for rapid bursts of I/O



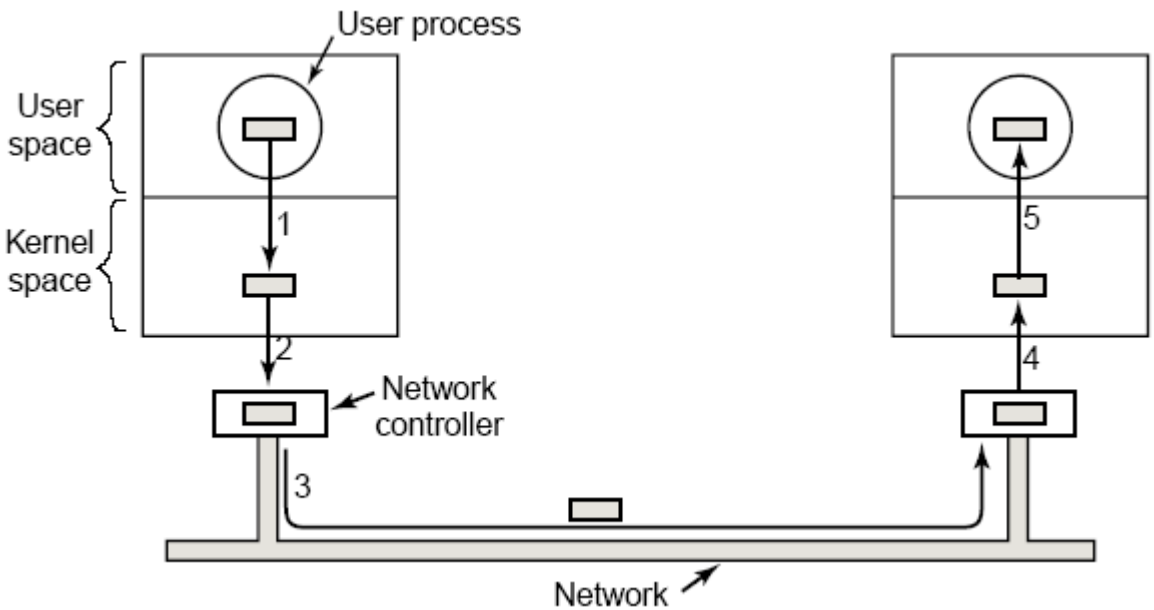
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.c I/O Software Layers

Device-independent I/O software

3. Device-independent I/O software (cont'd)

- ✓ buffering in networking



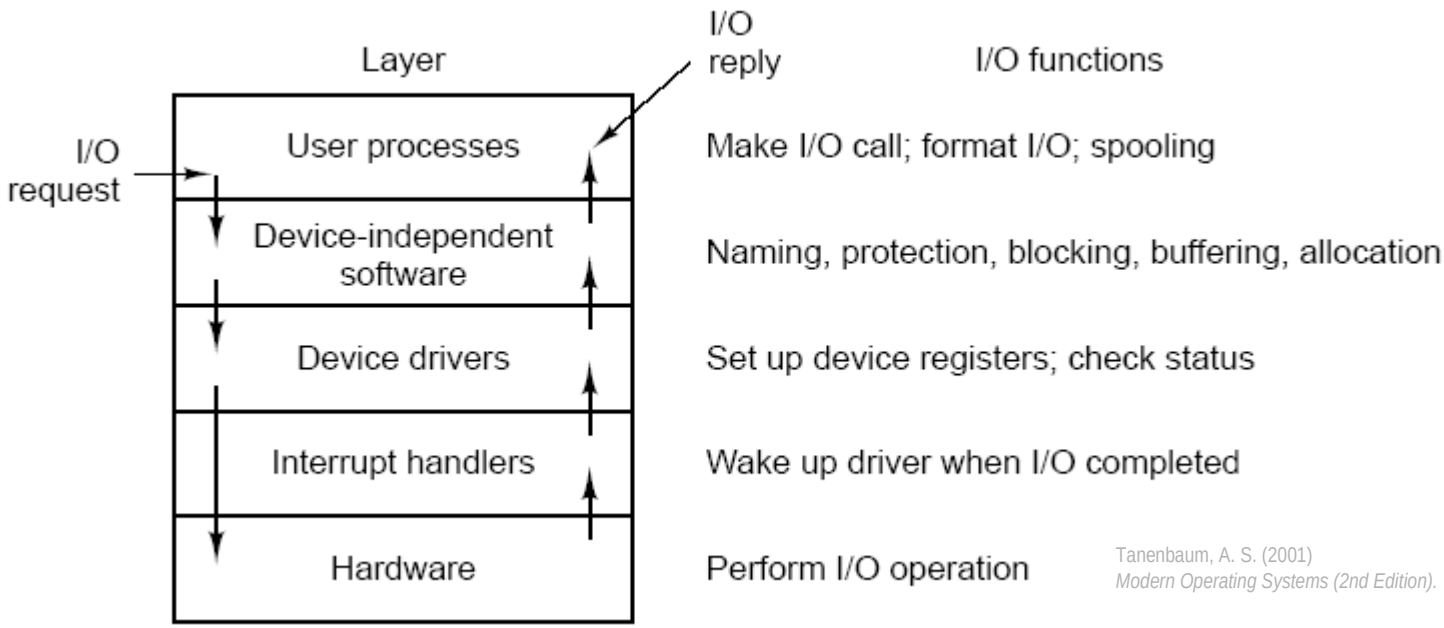
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

5.c I/O Software Layers

User-level I/O system calls

4. User-level I/O system calls

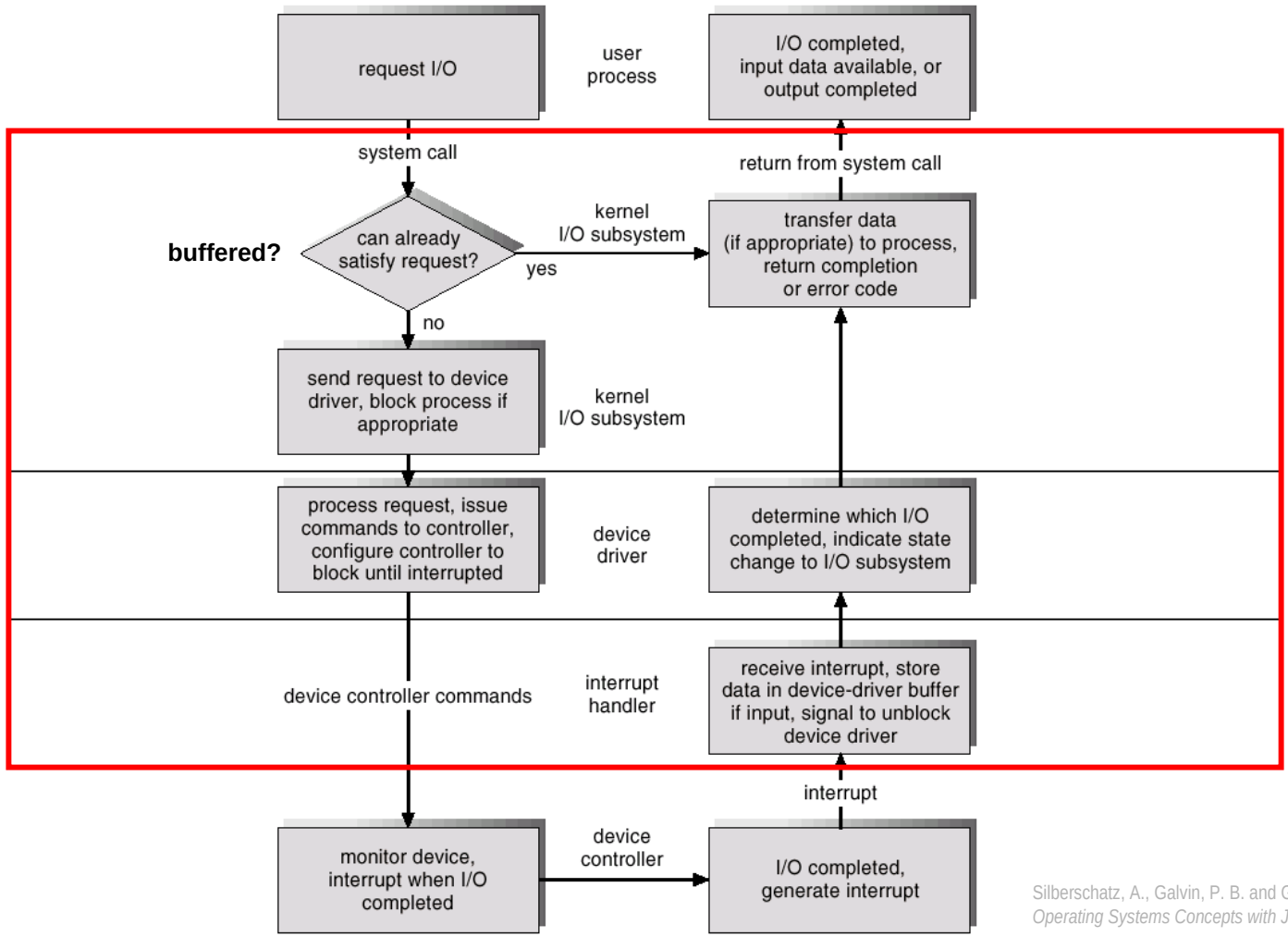
- ✓ **utility library** procedures wrapping system calls; for example, formatting: printf(), scanf()
- ✓ **spooling**: a daemon centralizes access requests to printer and other devices



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

5.c I/O Software Layers

User-level I/O system calls



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

The life-cycle of an I/O request

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

c. I/O Software Layers

- ✓ Overview of the I/O software
- ✓ Interrupt handlers
- ✓ Device drivers
- ✓ Device-independent I/O software
- ✓ User-level I/O system calls

d. Disk Management

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

c. I/O Software Layers

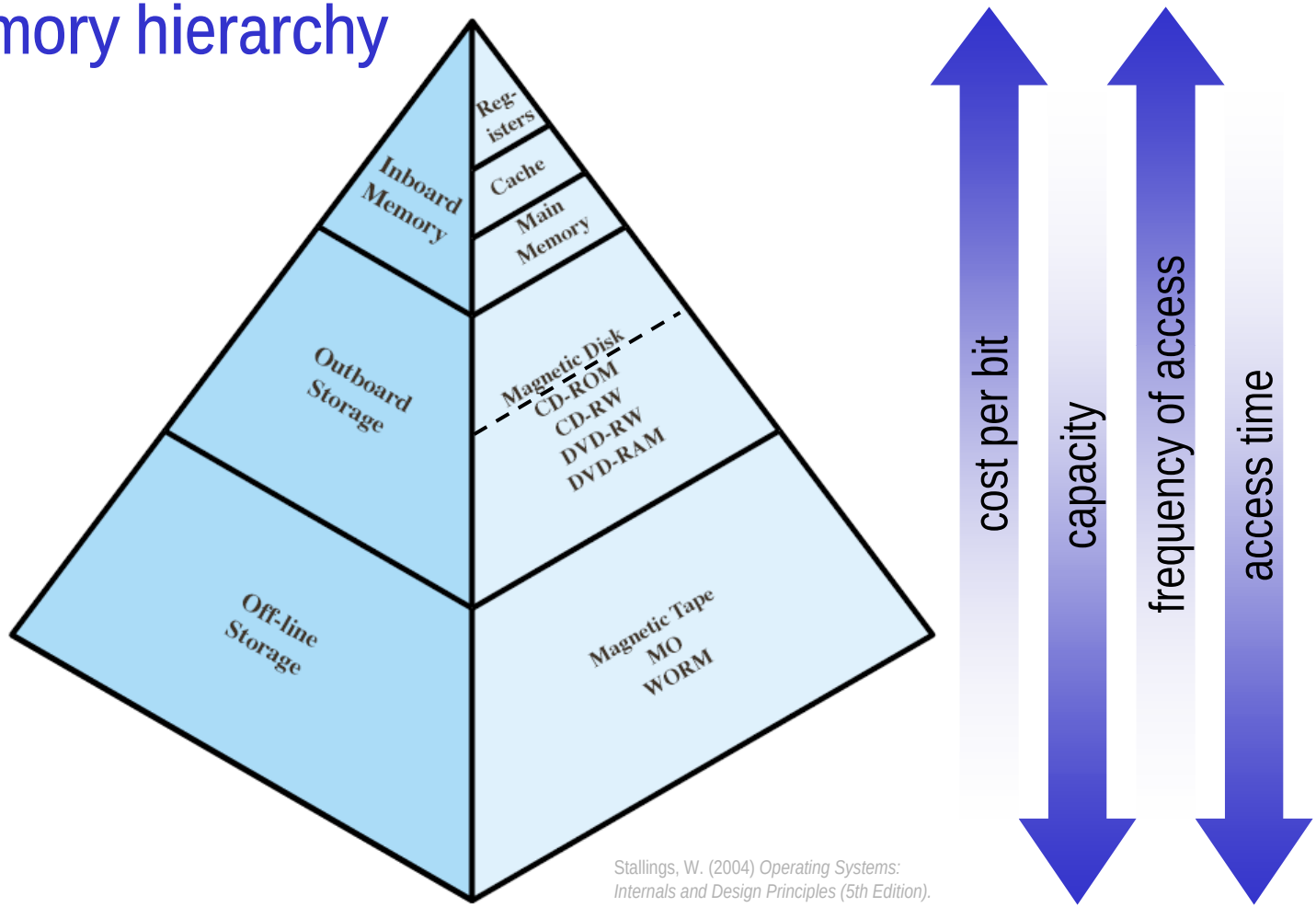
d. Disk Management

- ✓ Physical disk characteristics
- ✓ Disk formatting
- ✓ Disk scheduling

5.d Disk Management

Physical disk characteristics

- The memory hierarchy



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

The memory hierarchy

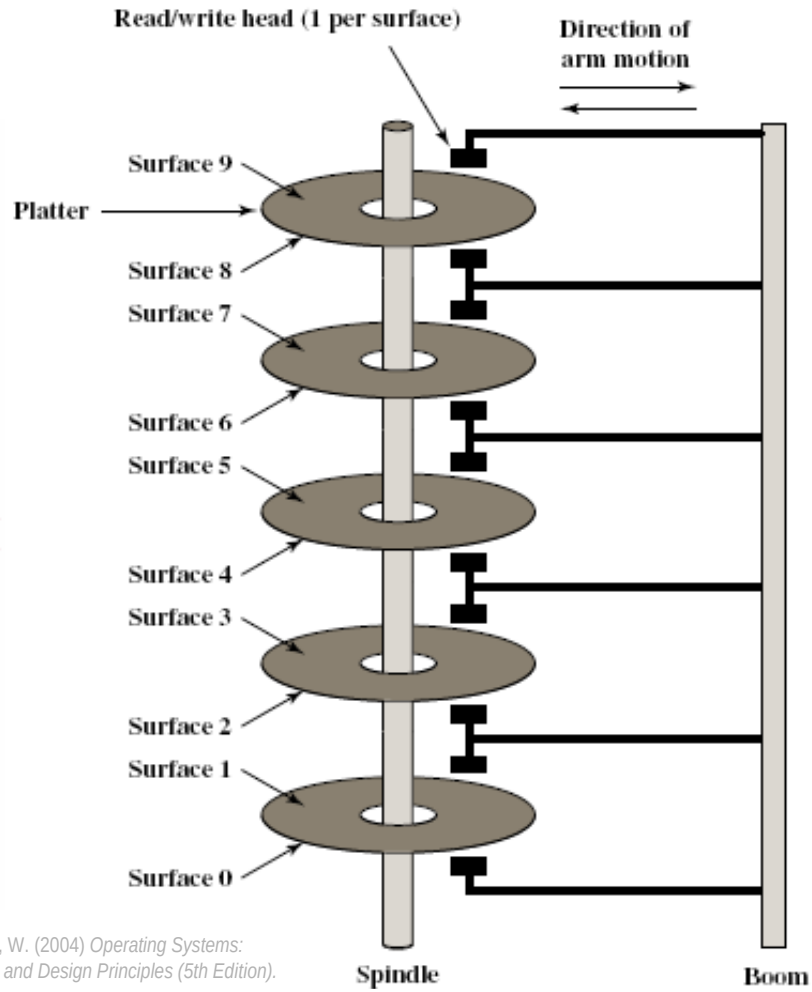
5.d Disk Management

Physical disk characteristics

- Rigid ("hard") magnetic disks
  - ✓ remain today the most important secondary memory (although the gap between CPU and disk performance has increased)
  - ✓ diameter shrunk from 50 cm down to 12 or 3 cm (notebooks)
  - ✓ "Winchester" disks are sealed
  - ✓ components of a disk drive:
    - one or several aluminum platters stacked vertically
    - platters have magnetizable coating on both sides
    - one pair of read/write movable heads per platter surface (heads hover on air cushion, don't make contact)
    - all heads mechanically fixed so they move together and are all at same distance from center

5.d Disk Management

Physical disk characteristics



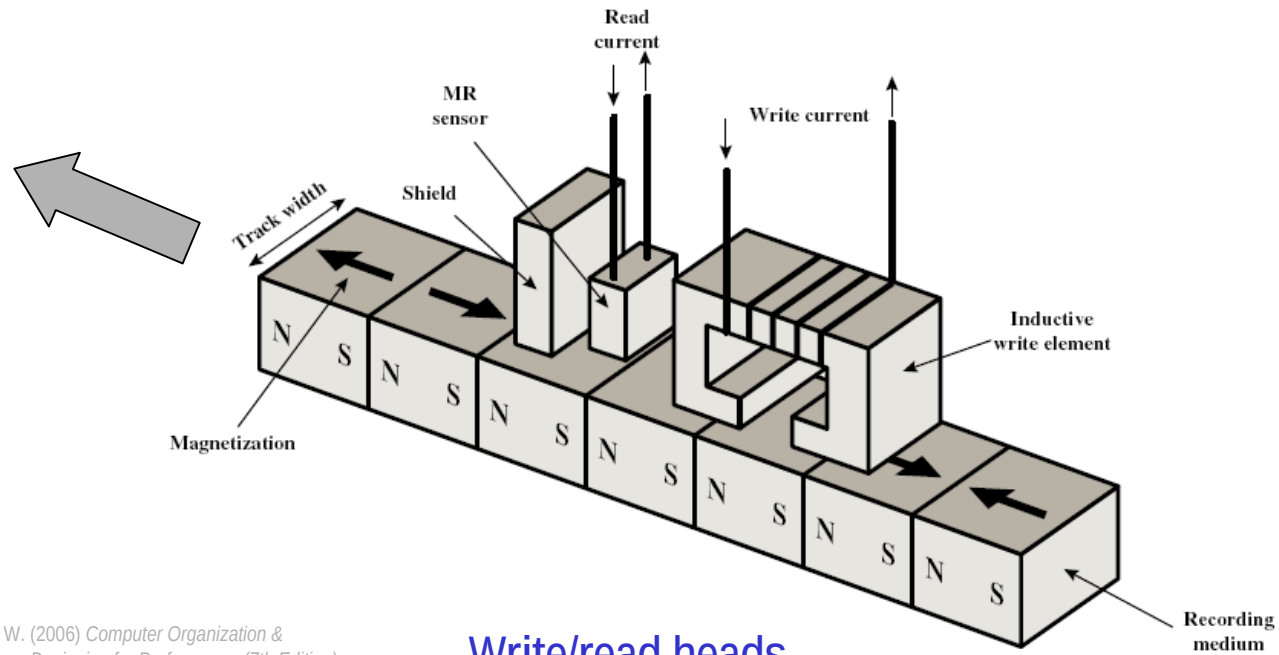
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Components of a disk drive

5.d Disk Management

Physical disk characteristics

- In modern systems, read and write heads are separate
  - ✓ the write head is an induction coil: produces a magnetic field
  - ✓ the read head is a magnetoresistive (MR) sensor: resistance depends on magnetic field, thus generates variable voltage



Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

Write/read heads

5.d Disk Management

Disk formatting

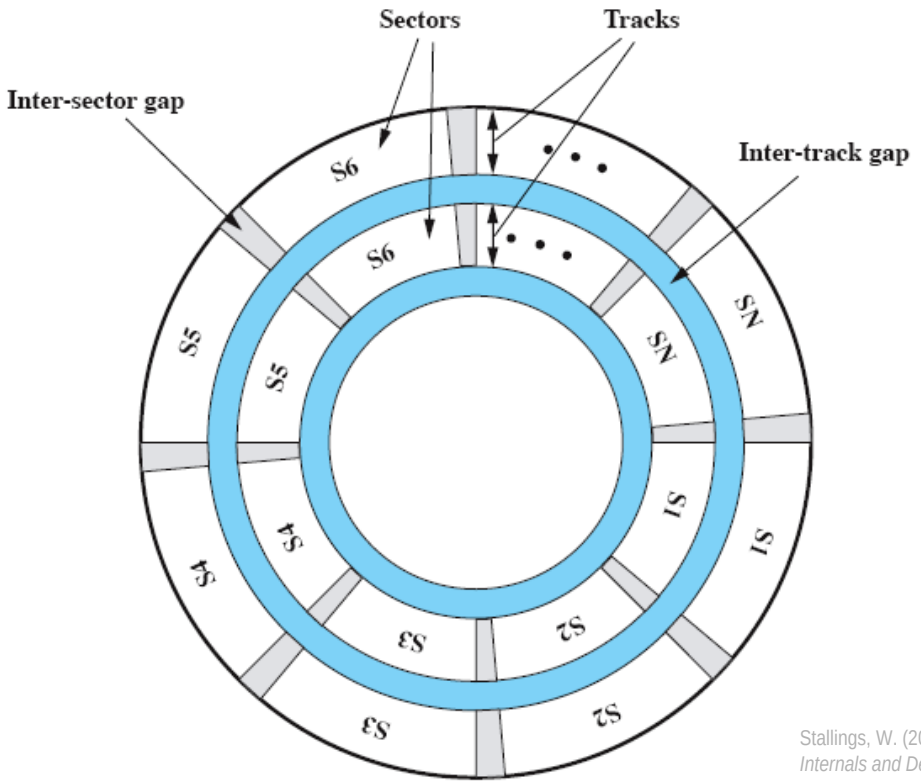
- Data organization and formatting
  - ✓ after manufacturing, there is no information on the disk: just a blank slate (continuous surface of magnetizable metal oxide)
  - ✓ before a disk can be used, each platter must receive a **low-level format** ("physical format") done by code in I/O controller:
    - series of concentric **tracks** (not grooves)
    - each tracks contains **sectors**, separated by short gaps
  - ✓ then the disk may be **partitioned**
  - ✓ finally, each partition receives a **high-level format** ("logical"):
    - boot sector, free storage map, file allocation table, etc.

  → *we'll see more of this in the File System chapter*

5.d Disk Management

Disk formatting

- A disk is addressed as a 1-D array of logical blocks
  - ✓ translation between logical block # and track # + sector #



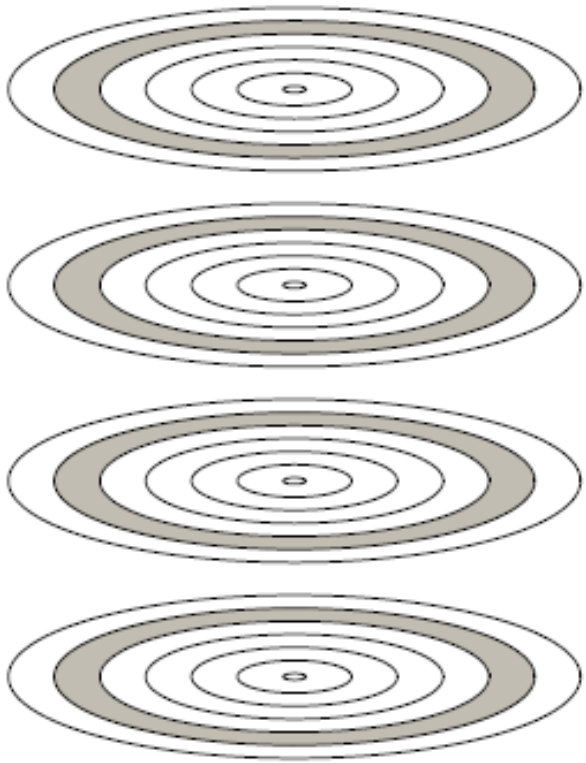
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Disk data layout

5.d Disk Management

Disk formatting

- Vertically aligned tracks on multiple platters are called "cylinders"



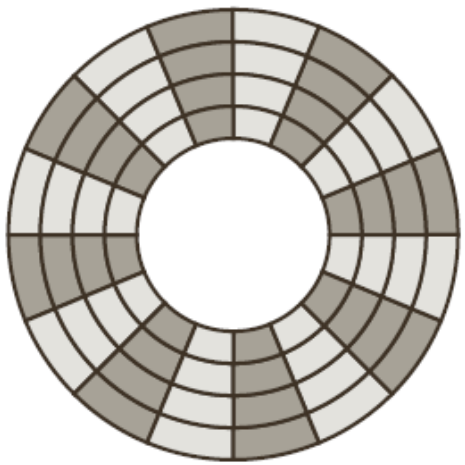
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Tracks and cylinders

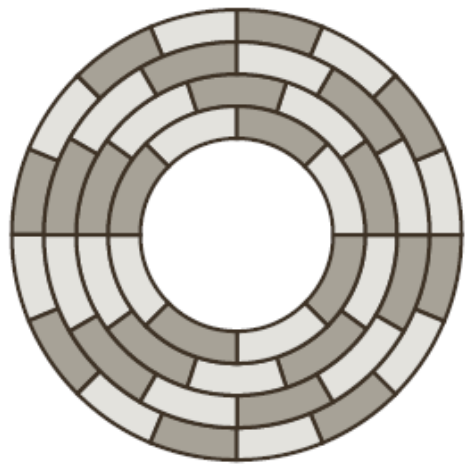
5.d Disk Management

Disk formatting

- Disk layout methods
  - ✓ **constant angular velocity**: pie-shaped sectors, same number per track → simple but wasted space on the long outer tracks
  - ✓ **multiple zone recording**: fixed-length sectors, variable number per track → greater data density but more complicated access



(a) Constant angular velocity



(b) Multiple zoned recording

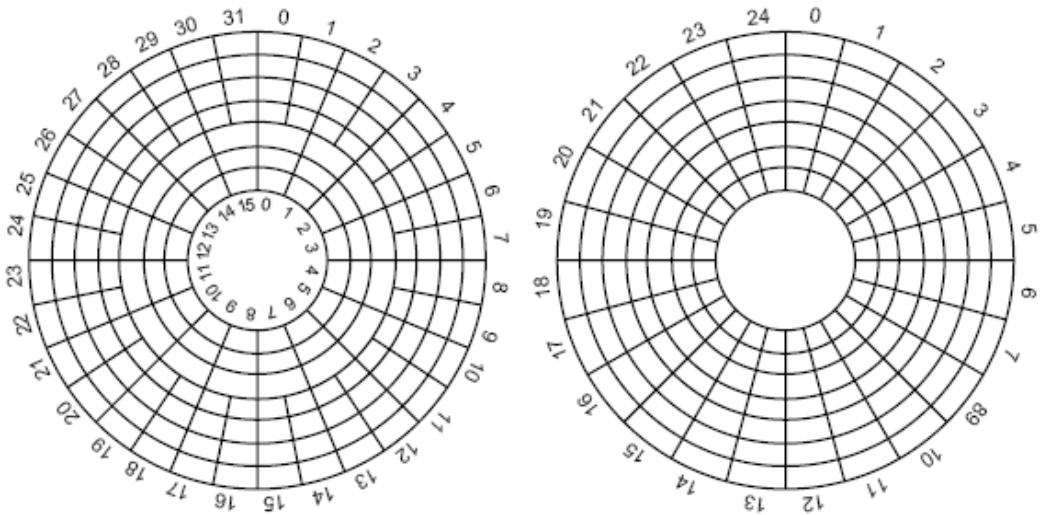
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Comparison of disk layout methods

5.d Disk Management

Disk formatting

- Virtual disk geometry
  - ✓ most disks are physically MZR but may still present a simpler, virtual CAV geometry to the O/S
  - ✓ the O/S driver uses cylinder, track, sector coordinates (*x*, *y*, *z*) which are remapped into zones by the I/O controller



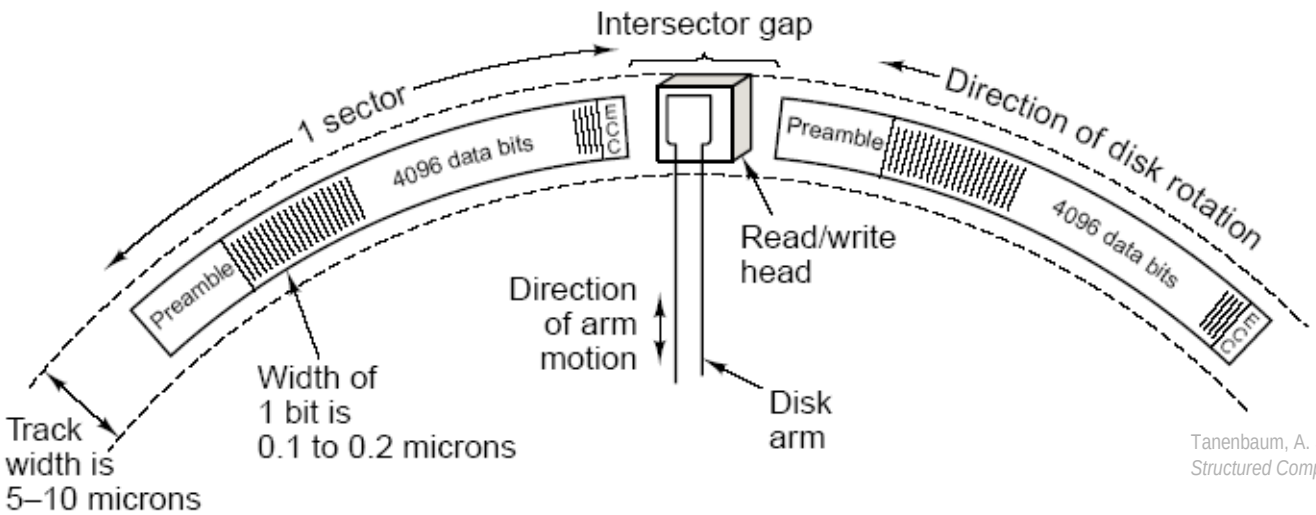
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Physical geometry vs. virtual geometry

5.d Disk Management

Disk formatting

- Tracks are divided into fixed-length sectors
  - ✓ each sector typically contains
    - 512 bytes of data
    - preceded by a preamble (for head synchronization)
    - followed by an error-correcting code (ECC)



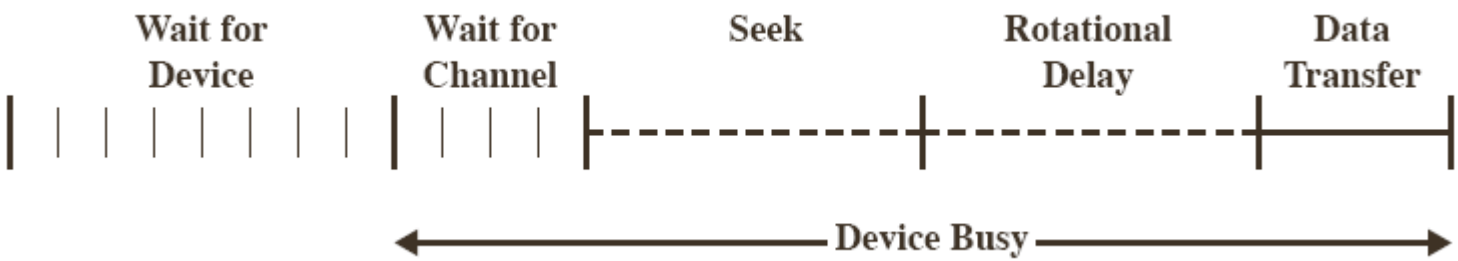
Tanenbaum, A. S. (2006)
Structured Computer Organization (5th Edition).

A portion of a disk track showing two sectors

5.d Disk Management

Disk scheduling

- Disk performance parameters
  - ✓ **seek time**: time it takes to position the head at the track
  - ✓ **rotational delay**: time it takes for the beginning of the sector to reach the head
  - ✓ **access time** = seek time + rotational delay
  - ✓ **transfer time**: time required for sector data transfer



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Timing of a disk I/O transfer

5.d Disk Management

Disk scheduling

- Disk performance parameters
  - ✓ average seek time typically < 10 ms (thanks to small diameter)
  - ✓ rotational speed $r \approx 7{,}500$ rpm = 1r / 8 ms $\rightarrow$ 4 ms rot. delay
  - ✓ transfer time $T = b / rN$ with b / N = transferred bytes / track

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 µsec

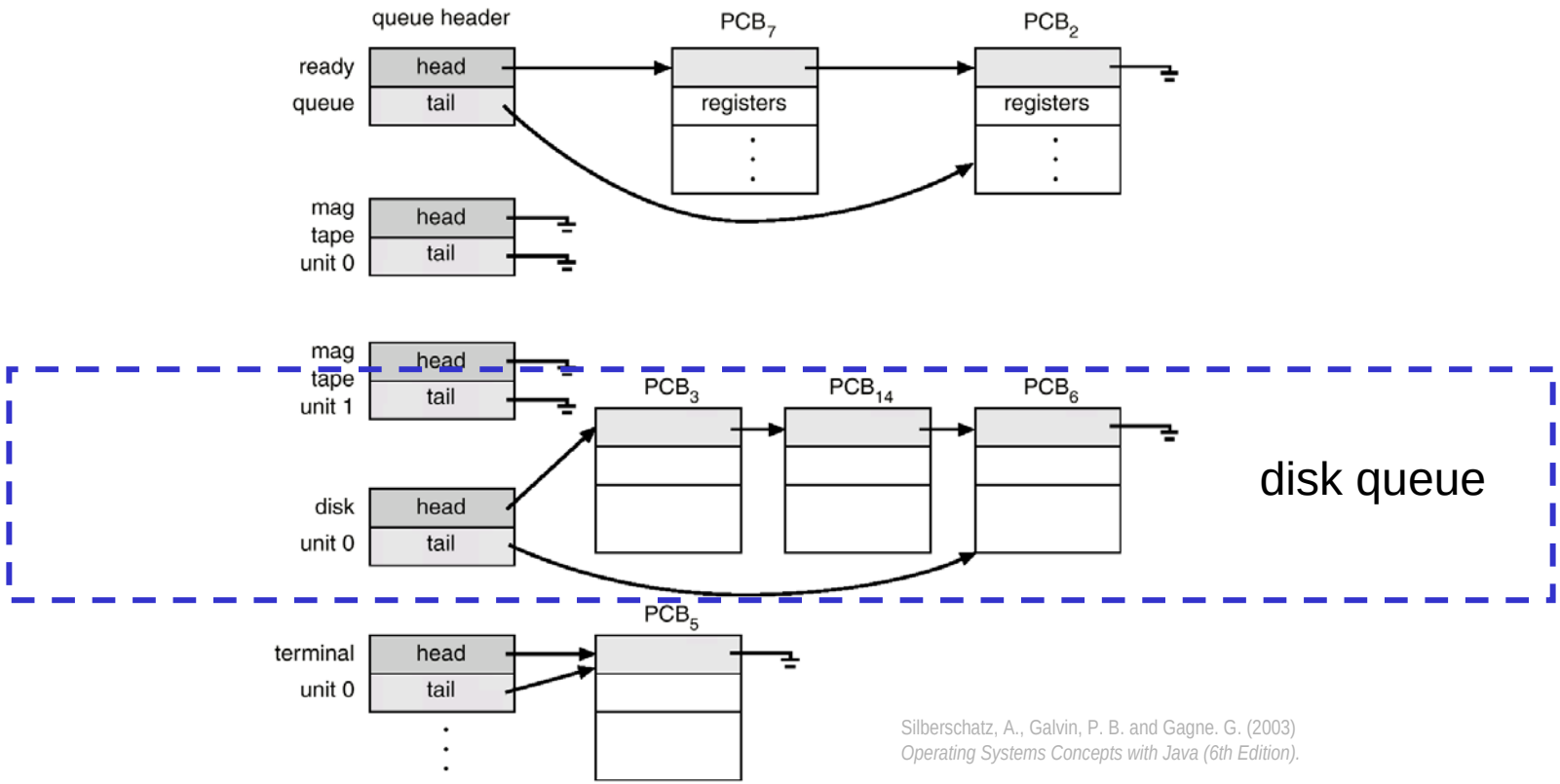
Two opposites on the historical scale of disk parameters

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

5.d Disk Management

Disk scheduling

- Additional waiting time for device availability
  - ✓ processes blocked for I/O are put into device-specific queues



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Various I/O device queues

5.d Disk Management

Disk scheduling

- Why disk scheduling matters: a timing comparison
  - ✓ total average service time

    $T_{service} = T_{seek} + T_{rotational} + T_{transfer}$

    $= T_{seek} + 1 / 2r + b / rN$
  - ✓ assume T_{seek} = 4 ms, r = 7,500 rpm, 500 sectors per track × 512 bytes per sector → $T_{transfer}$ = 0.016 ms / sector
  - ✓ first case: reading 2,500 randomly scattered sectors

    $T_{service}$ = 2,500 × (4 ms + 4 ms + 0.016 ms) = 20 seconds
  - ✓ first case: reading 2,500 contiguous sectors (in 5 tracks)

    $T_{service}$ = 4 ms + 5 × 4 ms + 2,500 × 0.016 ms = 64 ms
  - → *the order of sector access requests can make a big difference!*

5.d Disk Management

Disk scheduling

- Overview of disk scheduling policies
  - ✓ **kernel-level scheduling**: based on requestor process
    - control of scheduling outside of disk management software
    - not intended to optimize disk utilization
    - main objective is process priorities defined by the O/S
    - or following a blind, generic policy such as FIFO (no starvation) or LIFO (locality)
  - ✓ **driver-level scheduling**: based on requested item
    - goal is to optimize disk utilization
    - the disk-specific software has expertise on how requests should be ordered

5.d Disk Management

Disk scheduling

- Overview of disk scheduling policies
  - ✓ kernel-level (process) vs. driver-level (request) scheduling

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load sensitive

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.d Disk Management

Disk scheduling

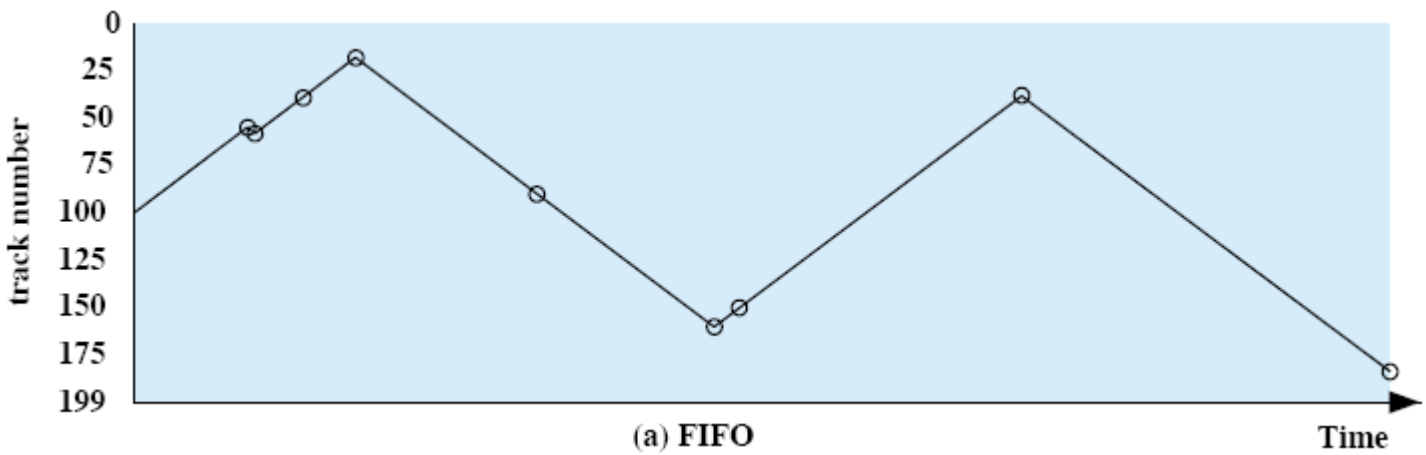
- Comparing performance of scheduling policies
  - ✓ assume disk with 200 tracks
  - ✓ assume sequence of requested tracks in order received by disk scheduler: 55, 58, 39, 18, 90, 160, 150, 38, 184
  - ✓ assume disk head initially located at track #100
  - ✓ we will compare FIFO, SSTF, SCAN, C-SCAN

5.d Disk Management

Disk scheduling

- First-In-First-Out (FIFO)
  - ✓ requests are processed in arrival order
  - ✓ fair and no risk of starvation
  - ✓ ok if few processes and requests cluster file sectors (locality)
  - ✓ generally bad, though, as interleaving causes random seek jumps and waste of time

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*



(a) FIFO (starting at track 100)

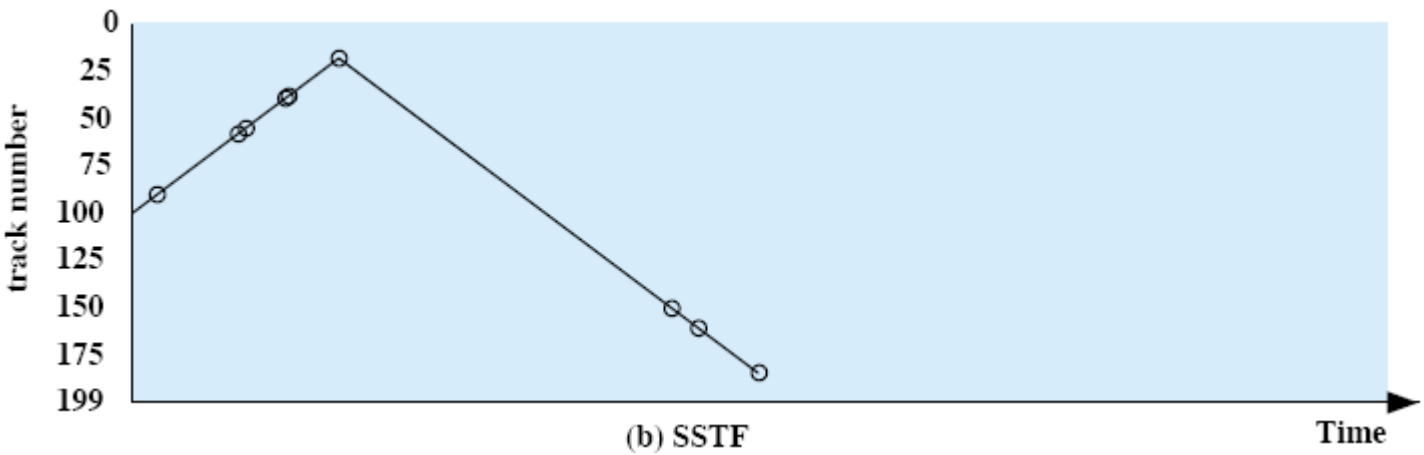
Next track accessed	Number of tracks traversed
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
Average seek length	55.3

5.d Disk Management

Disk scheduling

- Shortest Service (Seek) Time First (SSTF)
  - ✓ select the request that requires the least arm movement, i.e., the shortest seek time
  - ✓ much better than random or FIFO, however greater risk of starvation: requests in remote disk area may remain unfulfilled as long as there are shorter ones

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*



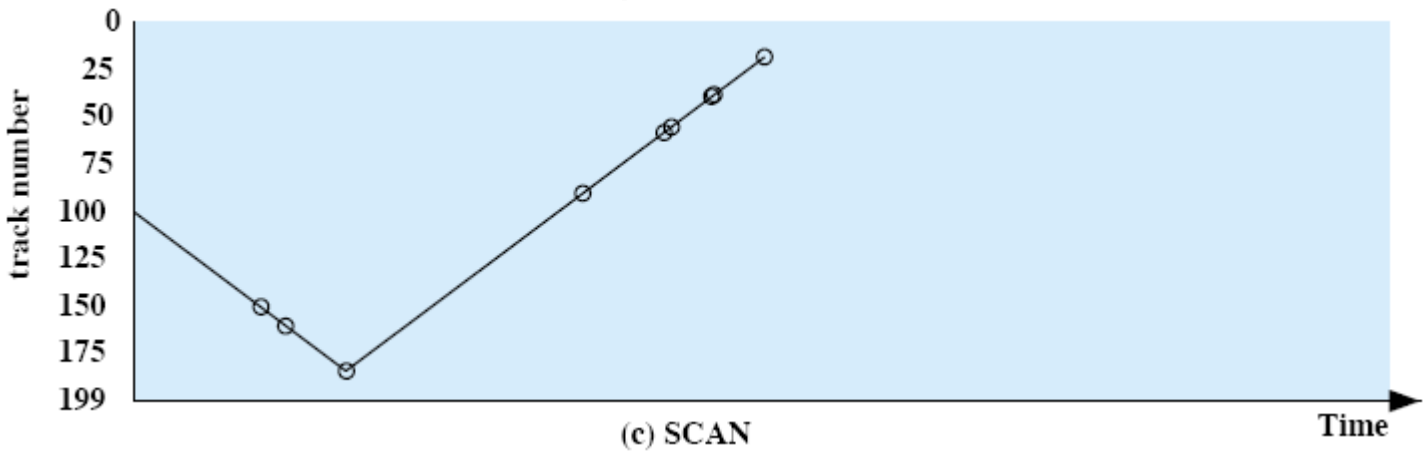
(b) SSTF
(starting at track 100)

Next track accessed	Number of tracks traversed
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
Average seek length	27.5

5.d Disk Management

Disk scheduling

- Scan or "elevator" algorithm (SCAN)
  - ✓ to prevent starvation, the arm moves in one direction only and satisfies requests "en route"
  - ✓ arm direction is reversed when reaching the last track (innermost or outermost)
  - ✓ . . . or as soon as reaching last request (LOOK: the variant implemented in Linux)



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

(c) SCAN
(starting at track 100, in the direction of increasing track number)

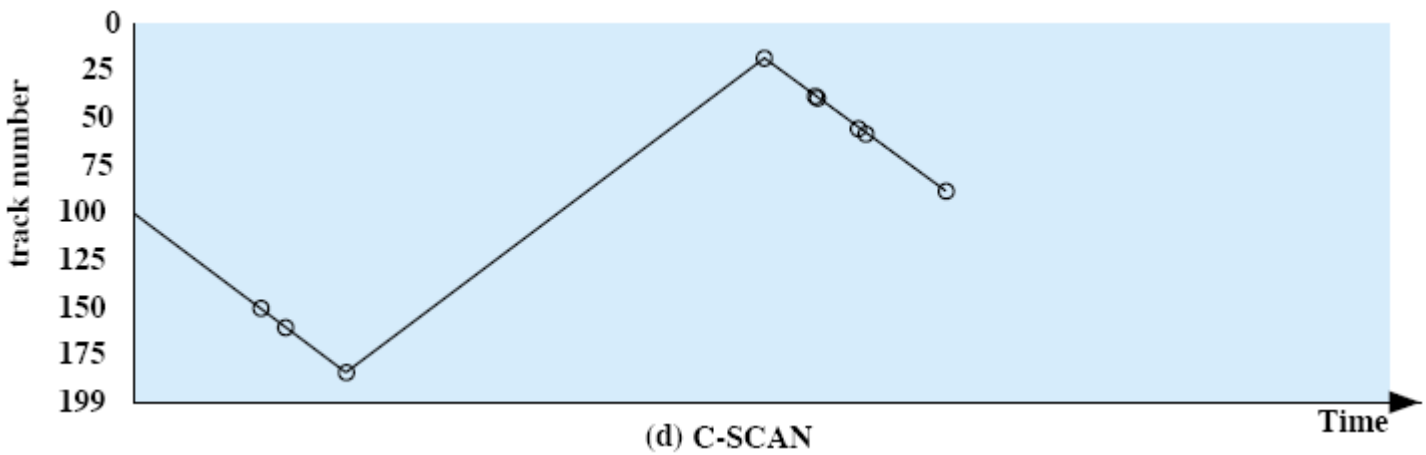
Next track accessed	Number of tracks traversed
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
Average seek length	27.8

5.d Disk Management

Disk scheduling

- Circular scan (C-SCAN)
  - ✓ same as SCAN except the arm direction of movement is never reversed
  - ✓ this reduces the maximum delay experienced by new requests that arrived at the opposite end of the disk

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*



(d) C-SCAN

(starting at track 100, in the direction of increasing track number)

Next track accessed	Number of tracks traversed
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
Average seek length	35.8

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

c. I/O Software Layers

d. Disk Management

- ✓ Physical disk characteristics
- ✓ Disk formatting
- ✓ Disk scheduling

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

c. I/O Software Layers

d. Disk Management

Principles of Operating Systems

CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. Processes
3. Memory Management
4. CPU Scheduling
5. Input/Output
6. **File System**
7. **Case Studies**