I/O Management and Disk Scheduling

Chapter 11

Introduction

- The two main jobs of a computer are I/O and processing.
  - In many cases, the major job is I/O. (for example, text file editing, and web page browsing)
- The role of the OS in I/O is:
  - To issue commands to the devices
  - To catch interrupts
  - To handle errors
  - To provide an simple and easy-to-use interface (between the devices and the rest of the system)

Contents

- A brief discussion of I/O devices
- Organization of the I/O function
- Operating system design issues
- I/O Buffering
- Disk Scheduling
- RAID
- Disk Cache

I/O Devices

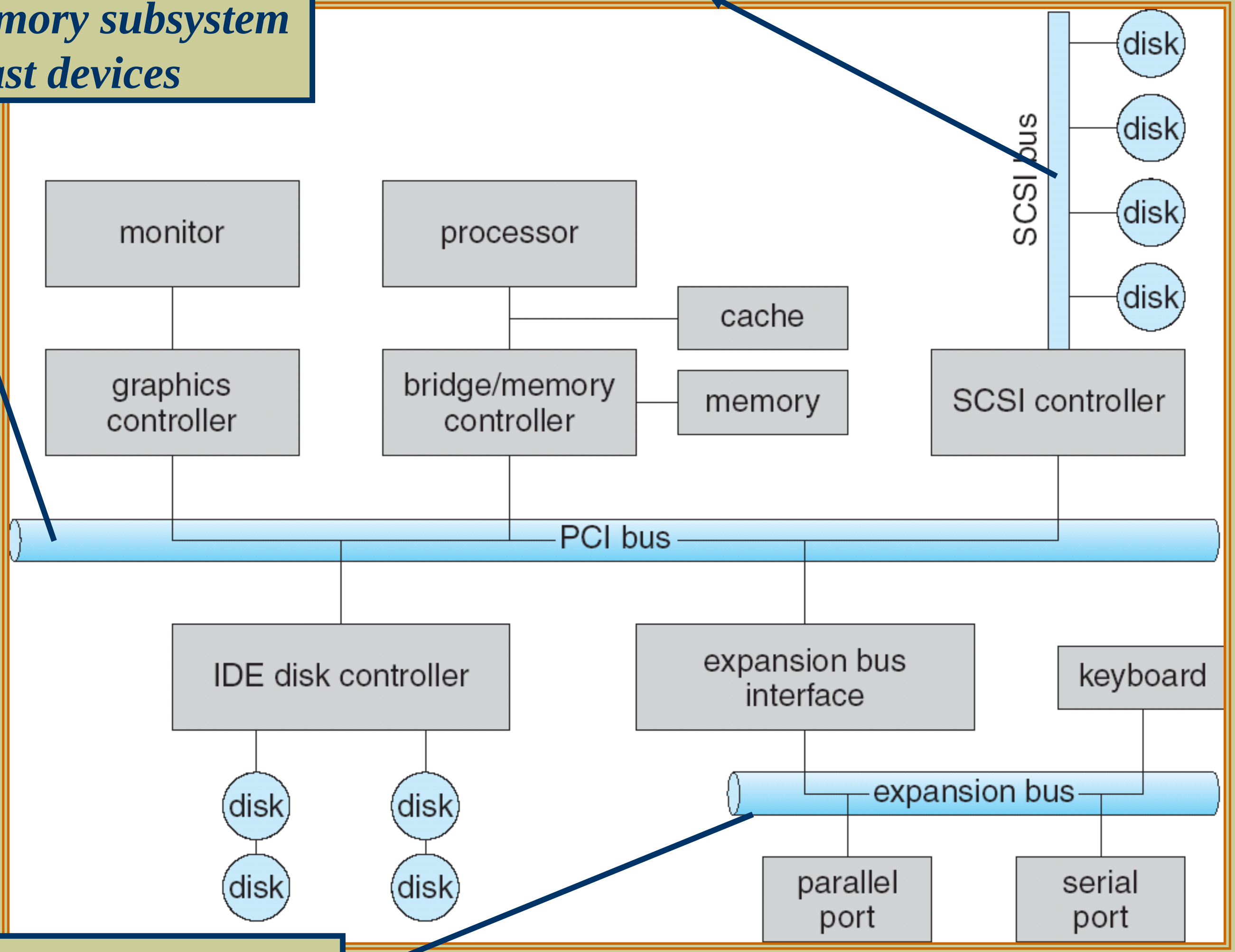
Buses
总线

- A device communicates with a machine via a connection point (or port) – for example: serial port (串口)
- If one or more devices use a common set of wires, the connection is called a bus (总线)
  - A bus is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.

A typical PC bus structure

PCI bus connects the processor-memory subsystem to the fast devices

SCSI bus is intended for fast disks, scanners and other devices needing considerable bandwidth.



The expansion bus connects relatively slow devices

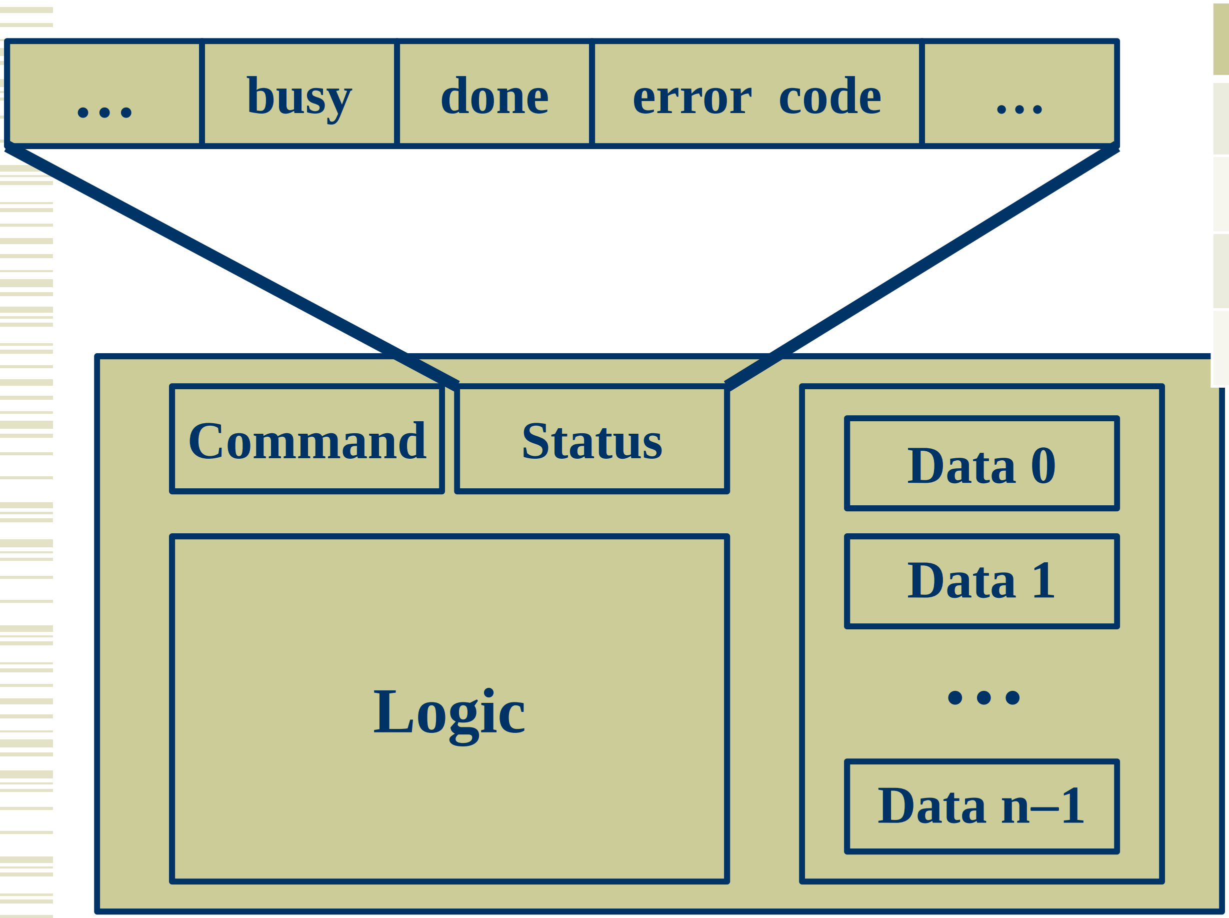
Controllers (1)
控制器

- A controller is a collection of electronics that can operate a port, a bus, or a device
  - A serial-port controller – a single chip that controls the signals on the wires of a serial port
  - SCSI bus controller – a separate circuit board that contains a processor, microcode, and some private memory to enable it to process the SCSI protocol messages
  - Disk controller – a circuit board attached to a disk drive (磁盘驱动器); it implements the disk side of the protocol for some kind of connection, SCSI or IDE

Controllers (2)

- Each controller has some (one or many) device registers
  - The processor can give the controller commands by writing some registers.
  - The processor can read out its status by reading some registers.
- The number of device registers and the nature of the commands vary radically from device to device.

The Conceptual Device Controller Interface



busy	done	
0	0	idle
0	1	finished
1	0	working
1	1	(undefined)

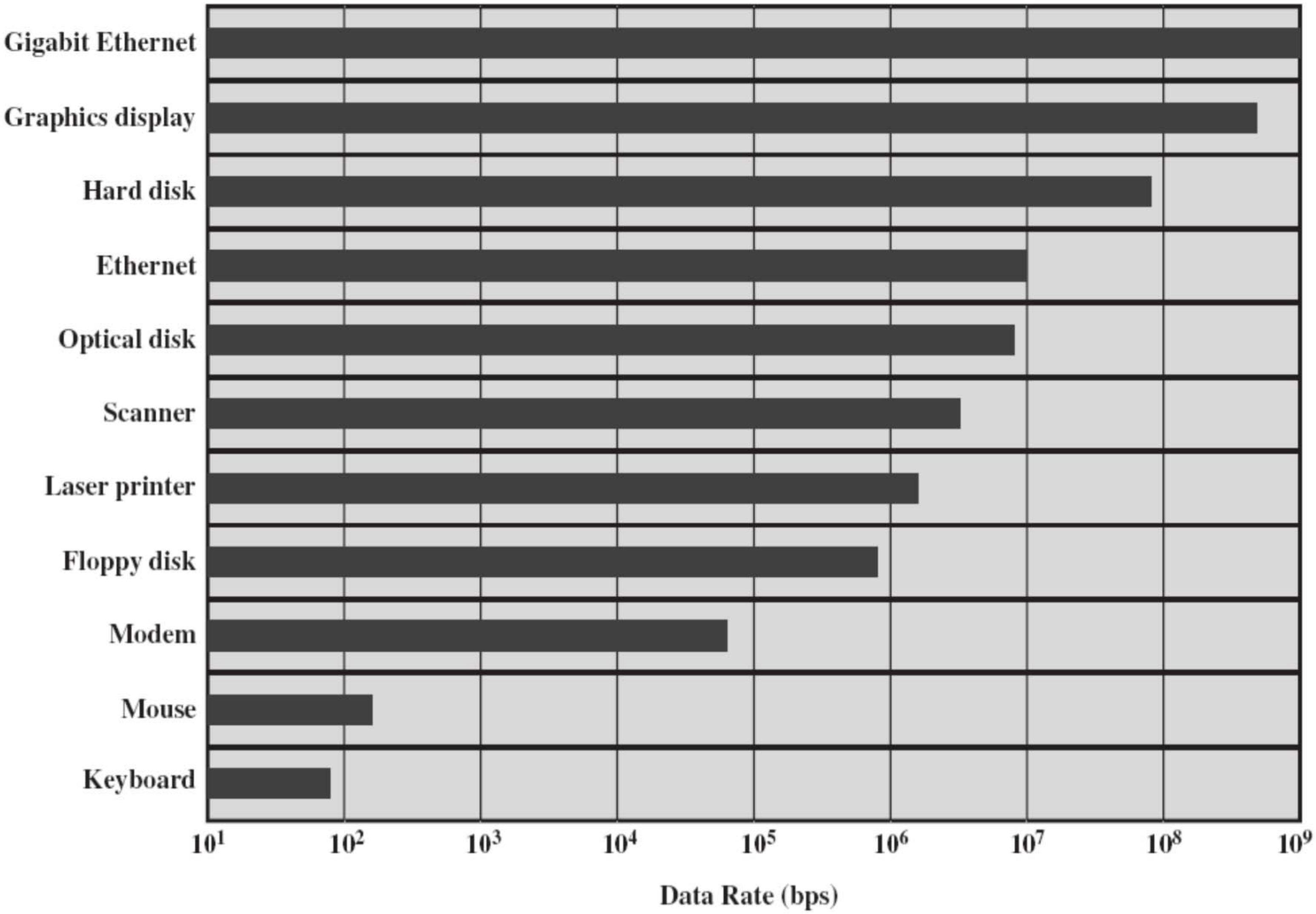
Categories of I/O Devices

- Three Categories
  - Human Readable: communicating with the computer user
    - Examples: printers and video display terminals (视频显示终端), the latter consisting of display,keyboard,mouse,and so on
  - Machine Readable: communicating with electronic equipment
    - Examples: disk and tape drives, sensors (传感器), controllers, and actuators (传动器)
  - Communication: communicating with remote devices
    - Examples: Digital line drivers and modems (数字线驱动器和调制解调器)

Differences in I/O Devices (1)

- There are great differences across classes and even within each class.
- The key differences include:
  - *Data rate*, application, complexity of control, *unit of transfer*, *data representation*, and error conditions

Differences in I/O Devices (2)– Data Rate



Differences in I/O Devices (3) Application

- Application: The use to which a device is put has an influence on the software and policies in the OS and supporting utilities
  - Disk used to store files requires file-management software
  - Disk used to store virtual memory pages needs special hardware and software to support it
  - Terminal used by system administrator may have a higher priority

Differences in I/O Devices (4)

- Complexity of control
- Unit of transfer
  - Data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
- Data representation
  - Encoding schemes
- Error conditions
  - Devices respond to errors differently

What do these differences cause?

- This diversity makes a uniform and consistent approach to I/O difficult to achieve

Another Classification Scheme

根据输入输出特性来分

- Block devices: a block device stores information in fixed-size blocks, each one with its address
  - Disks are the most common block devices
- Character devices: a character device delivers or accepts a stream of characters, without any regard to any block structure
  - It is not addressable and does not have any seek function
  - Printer, network interfaces, mice, etc.
- This scheme is not perfect. Some devices just do not fit in (for example: clocks – cause interrupts at well-defined intervals

Sharable vs. Dedicated Devices
共享设备与独占设备

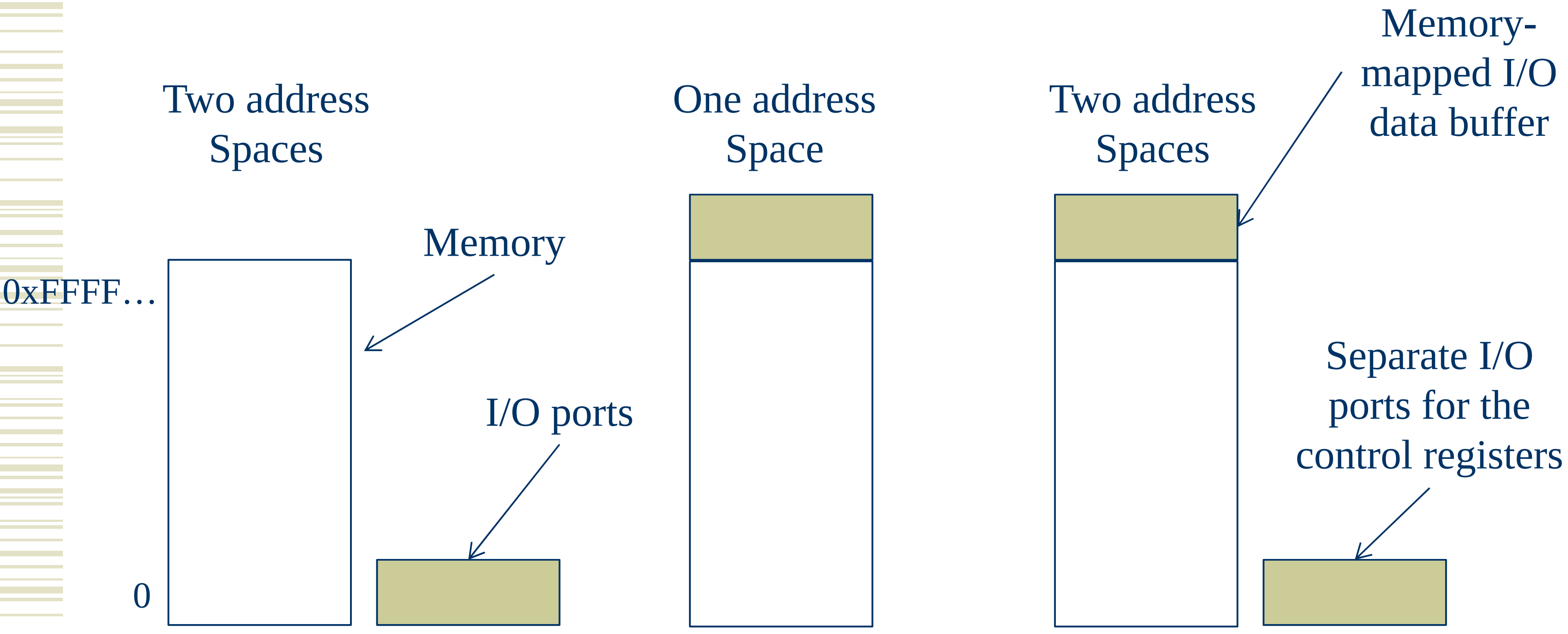
根据设备分配特性来分

- ***Sharable devices can be used by many users at the same time***
  - ***Example: disks – multiple users can have open files on the same disk at the same time***
- ***Dedicated devices have to be dedicated to a single user until that user is finished***
  - ***Example: printers – having two or more users printing their results at random to the same printer will not work***

Organization of the I/O Function

Memory-Mapped I/O

- Each controller has a few registers for communicating with the CPU.
- Two alternatives exist for the communication
  - Each register is assigned an I/O port number, an 8- or 16-bit integer
    - `IN REG, PORT` or `OUT PORT, REG`
  - I/O registers are part of the regular memory address space (called memory-mapped I/O)
    - Each register is assigned a unique memory address where no memory is assigned



Two address Spaces

Memory

0xFFFF…

0

I/O ports

One address Space

Two address Spaces

Memory-mapped I/O data buffer

Separate I/O ports for the control registers

Separate I/O and memory space

Memory-mapped I/O

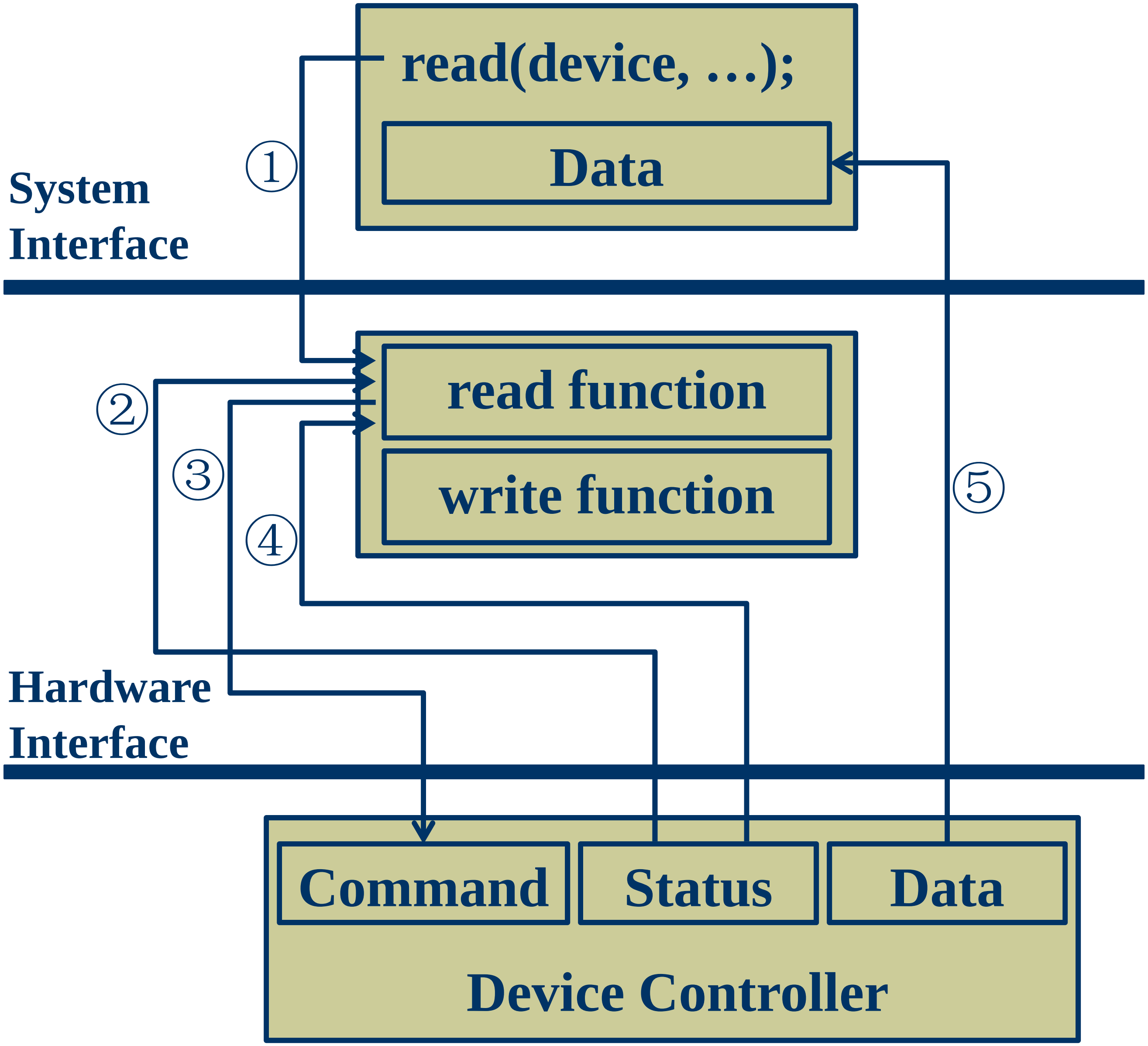
Hybrid

How I/O Can Be Done?

- The user process issues a system call
  - The kernel translates the system call into a procedure call to the appropriate driver
  - The procedure call to the appropriate driver may do the I/O work in three different way
    - Programmed I/O (程序直接控制I/O方式)
    - Interrupt-Driven I/O (中断驱动I/O方式)
    - DMA (内存直接存取方式)

Programmed I/O
程序直接控制的I/O

- I/O with Polling (轮询) or busy waiting (忙等)
  - Controller registers usually have one or more status bits that can be tested to determine if an output operation is complete or if new data is available from an input device.
  - A CPU can execute a loop, testing a status bit each time until a device is ready to accept or provide new data
  - CPU cycles are wasted to repeatedly test the status bits, which may lead to low CPU utilization



read(device, …);

Data

System Interface

read function

write function

Hardware Interface

Command

Status

Data

Device Controller

Programmed I/O

The OS usually copies the buffer with the string to an array *p* in kernel space

The OS checks to see if the printer is ready to accept a character

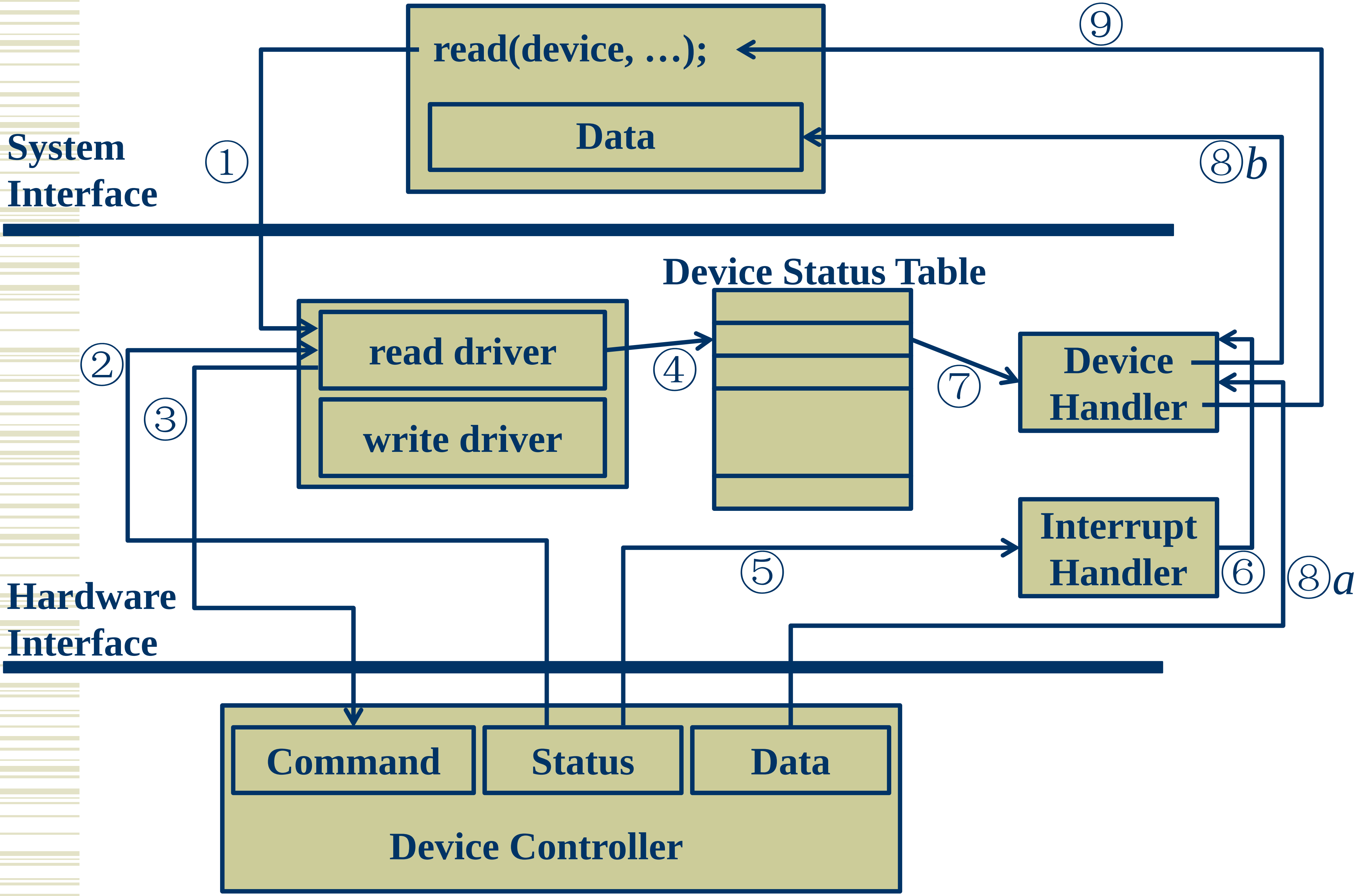
The OS writes a character to the data register of the printer

```
copy_from_user(buffer, p, count);
for (i=0; i<count; i++) {
    while (*printer_status_reg!=READY);
    *printer_data_register=p[i];
}
Return_to_user();
```

After the entire string has been printed, the control returns to the user process

Interrupt-Driven I/O

- The motivation for incorporating interrupts into the computer hardware
  - To eliminate the need for the device driver software to constantly poll the controller status register
  - Instead, many device controllers use interrupts to automatically notify the CPU when they are ready to have their registers read or written



read(device, …);

Data

System
Interface

Device Status Table

read driver

write driver

Device
Handler

Interrupt
Handler

Hardware
Interface

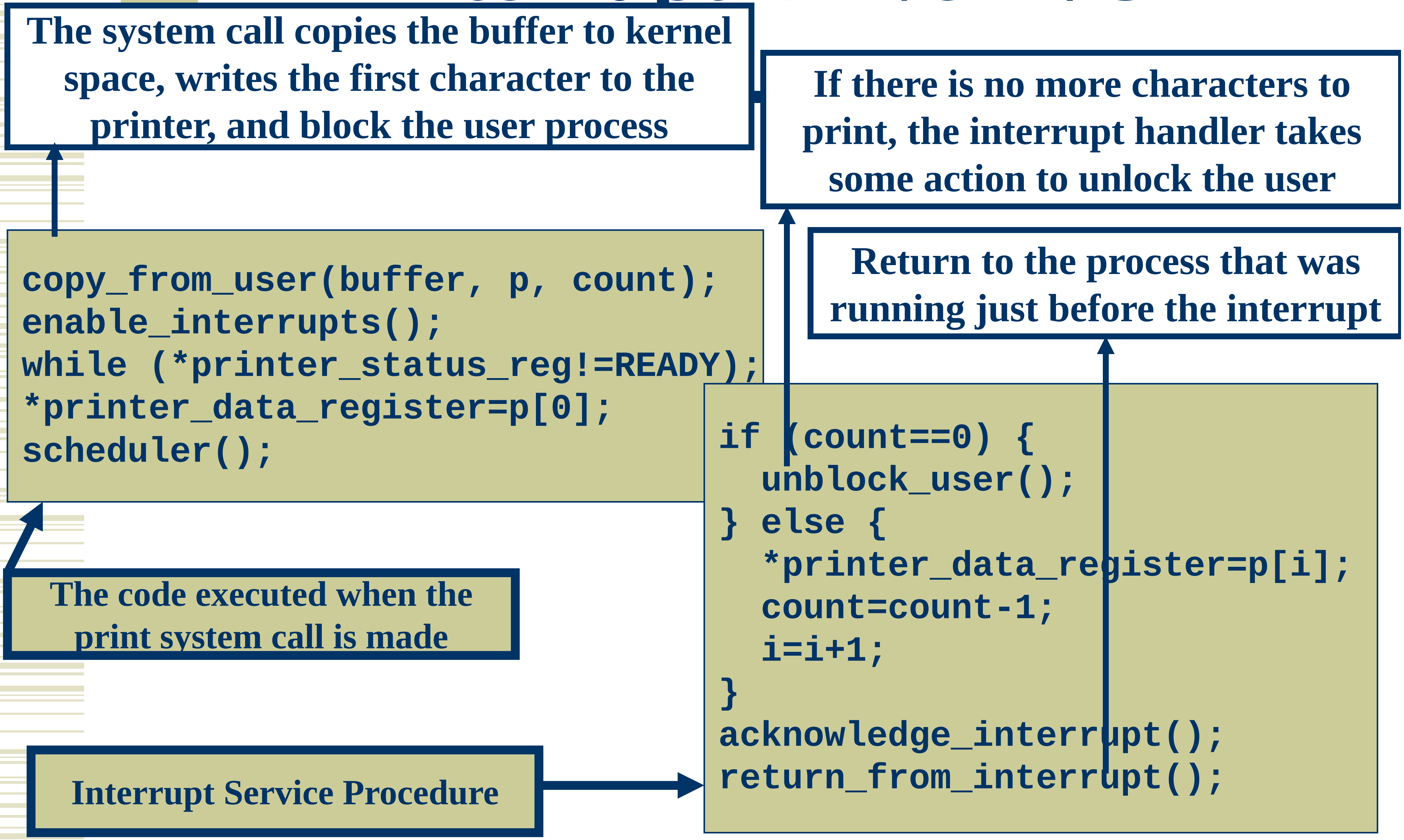
Command

Status

Data

Device Controller

Interrupt-Driven I/O



The system call copies the buffer to kernel space, writes the first character to the printer, and block the user process

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

The code executed when the print system call is made

If there is no more characters to print, the interrupt handler takes some action to unlock the user

Return to the process that was running just before the interrupt

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

Interrupt Service Procedure

I/O Using DMA

The big win with DMA is reducing the number of interrupts from one per character to one per buffer printed

On the other hand, the DMA controller is usually much slower than the main CPU. If the DMA controller is not capable of driving the device at full speed, or the CPU usually has nothing to do anyway while waiting for the DMA interrupt, then interrupt-driven I/O or even programmed I/O is better

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

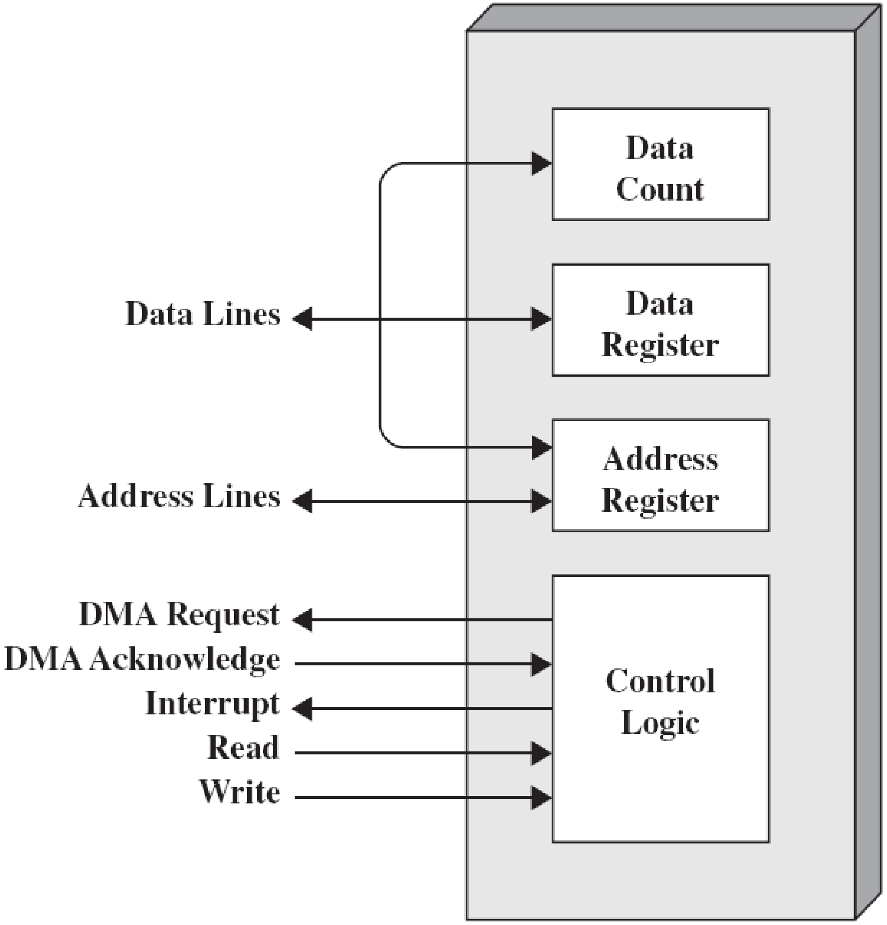
```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

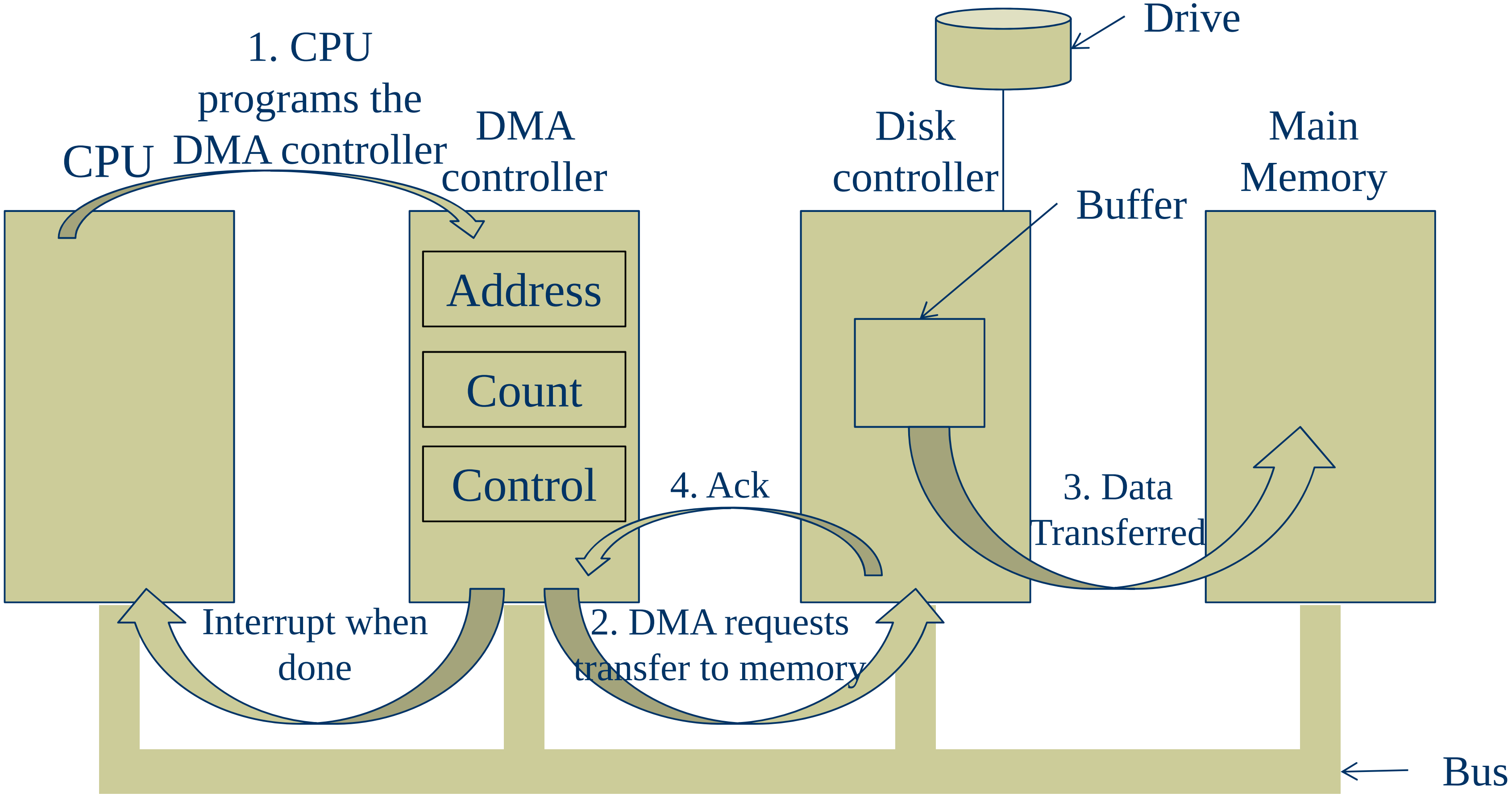
Interrupt Service Procedure

Direct Memory Access
直接内存存取

- Takes control of the system from the CPU to transfer data to and from memory over the system bus
- Cycle stealing (周期窃取) is used to transfer data on the system bus
  - The instruction cycle is suspended so data can be transferred
  - The CPU pauses one bus cycle
  - No interrupts occur
    - Do not save context

Typical DMA Block Diagram 典型的DMA结构简图




1. CPU programs the DMA controller
CPU
DMA controller
Address
Count
Control
Drive
Disk controller
Buffer
Main Memory
4. Ack
3. Data Transferred
Interrupt when done
2. DMA requests transfer to memory
Bus

	No interrupts	Use of interrupts
I/O to memory transfer through CPU	Programmed I/O	Interrupt-Driven I/O
Direct I/O-to-memory transfer		Direct Memory Access (DMA)

Operating System Design Issues

Design Objectives

- Two objectives are paramount in designing the I/O facility
  - Efficiency (效率)
  - Generality (通用性)

Design Objectives Efficiency

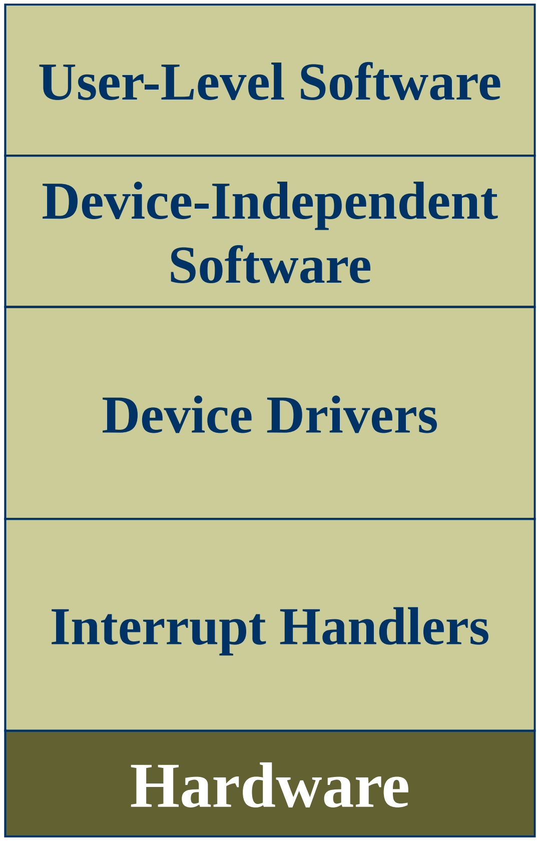
- Efficiency is important because I/O operations often form a bottleneck in a computing system
  - Most I/O devices extremely slow compared to main memory
  - One way to tackle this problem is multiprogramming
    - It allows for some processes to be waiting on I/O while another process executes
  - It still happens that I/O cannot keep up with processor speed, even with the vast size of main memory
    - Swapping is used to bring in additional Ready processes which is an I/O operation itself
- Much of this chapter is devoted to the disk I/O efficiency

Design Objectives
Generality

- A key concept in the design of I/O software is known as *device independence* (设备独立性)
  - It means that it should be possible to write programs that can access any I/O device without having to specify the device in advance
- It is desirable to handle all devices in a uniform manner.
  - It is difficult in practice to achieve true generality, because of the diversity of device characteristics
  - Using a hierarchical modular approach to hide most of the details of device I/O in lower-level routines
    - so that processes and upper levels see devices in general terms such as read, write, open, close, lock, unlock

I/O Software Organization

- I/O software is typically organized in four layers
  - The functionality and interfaces differ from system to system



User-Level Software

Device-Independent Software

Device Drivers

Interrupt Handlers

Hardware

Even if programmed I/O is occasionally useful, interrupts can not be avoided for most I/O

Interrupt Handlers

- **The driver starting an I/O operation gets blocked until the I/O has completed and the interrupt occurs**
  - **For example: The driver can block itself by doing *wait*() on a semaphore, a *receive*() on a message, etc.**
- **When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt**
- **Then, the interrupt handler can unblock the driver and started it**
  - **For example: the interrupt handler can do *singal*() on a semaphore, or send a message to the blocked driver, etc**

Device Drivers

- Device drivers contains the device-specific code for controlling I/O devices
  - Controlling a device means issuing a sequence of commands to it.
- The device driver normally has to be part of the OS kernel
  - In order to access the device's hardware

Device Drivers

- To accept abstract read/write requests from the device-independent software and see that they are carried out.
  - Check if the input parameters are valid
  - Check if the device is currently in use
  - Controlling the device means issuing a sequence of commands to it.
  - After the commands have been issued, one of two situations will apply
    - In many cases, the device driver must wait until the controller does some work for it, so it blocks itself until the interrupt comes in to unblock it
    - In other case, the operation finishes without delay, so the
  - After the operation completed, the driver must check for errors

Device-Independent I/O Software (1) Basic Functions

- The basic function of the device-independent software is:
  - to perform the I/O functions that are common to all devices (such as *buffering*, *error reporting*, *allocating and releasing dedicated devices*, and *providing a device-independent block size*)
  - to provide a uniform interface to the user-level software

Device-Independent I/O Software (2)
Uniform Interfacing for Device Drivers

- The interface between the device drivers and the rest of the operating system should be as uniform as possible
- How I/O devices are named
  - The device-independent software takes care of mapping symbolic device names onto the proper driver

Device-Independent I/O Software (3) Buffering

- Discussed later

Device-Independent I/O Software (4) Error Reporting

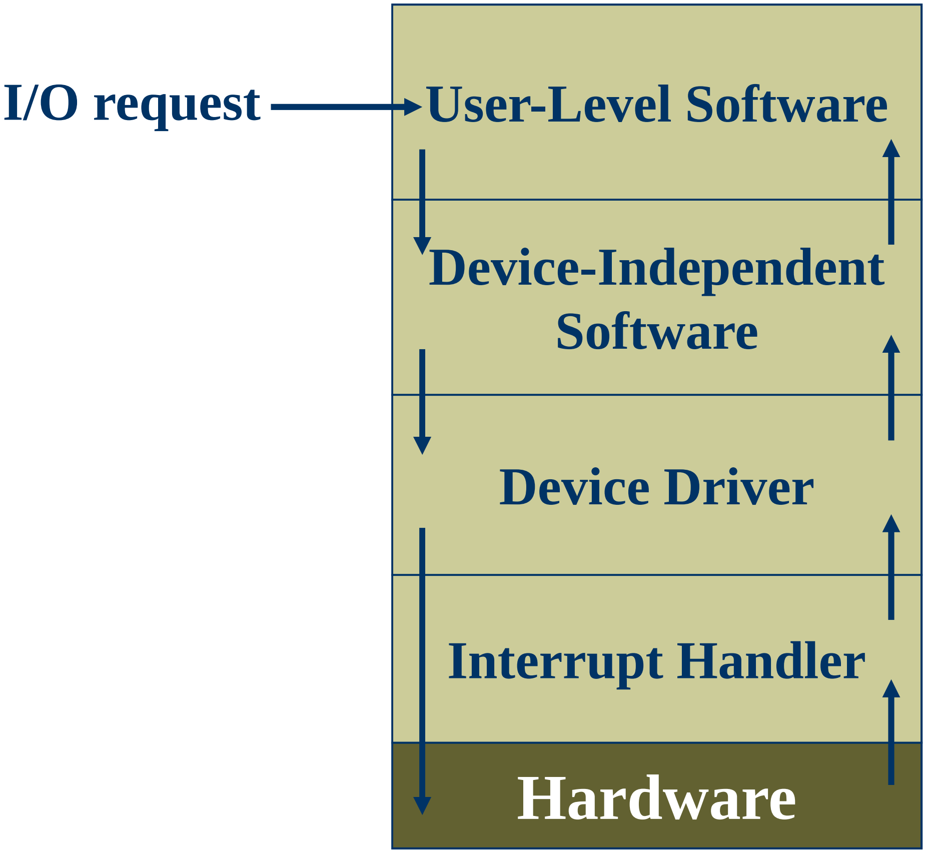
- Many errors are device-specific and must be handled by the appropriate driver
  - Programming error (such as writing to an input device)
  - Actual I/O error (such as trying to read from a camcorder that has been switched off)
- How to deal with error depends on the environment and the nature of the error
  - For a simple read error and there is an interactive user available, it may display a dialog box asking the user what to do
  - For some serious errors, the system may have to display an error message and terminate

Device-Independent I/O Software (5) Allocating and Releasing Dedicated Devices

- A simple way to handle these requests is to require processes to perform *open()* on the special files for devices directly
  - If the device is not available, the *open()* fails.
  - Closing such a dedicated device then releases it.
- An alternative approach is to have special mechanisms for requesting and releasing dedicated devices
  - An attempt to acquire a device that is not available blocks the caller (Blocked processes are put on a queue)
  - When the requested device becomes available, the first process on the queue is allowed to acquire it and continue execution

User-Space I/O Software

- A small portion of the I/O software consists of libraries linked together with user programs
  - System calls are normally made by library procedures.
  - Some library procedures simply put their parameters in the appropriate place for the system call
  - Other procedures do the formatting of input and output. (such as the *printf*() from C)
- Spooling: a way of dealing with dedicated I/O devices in a multiprogramming system (discussed later)



I/O request

Layer	I/O functions
User-Level Software	Make I/O call; format I/O; spooling
Device-Independent Software	Naming, protection, blocking, buffering, allocation
Device Driver	Set up device registers; Check status
Interrupt Handler	Wake up driver when I/O completed
Hardware	Perform I/O operation

I/O Buffering
I/O缓冲

I/O Buffering

- I/O buffering is a technique to overlap computation time and I/O time
  - Buffering tends to keep I/O devices running smoothly
  - A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application
- Without buffering,
  - Processes must wait (be blocked) for I/O to complete before proceeding
  - Certain pages must remain (be locked) in main memory during I/O

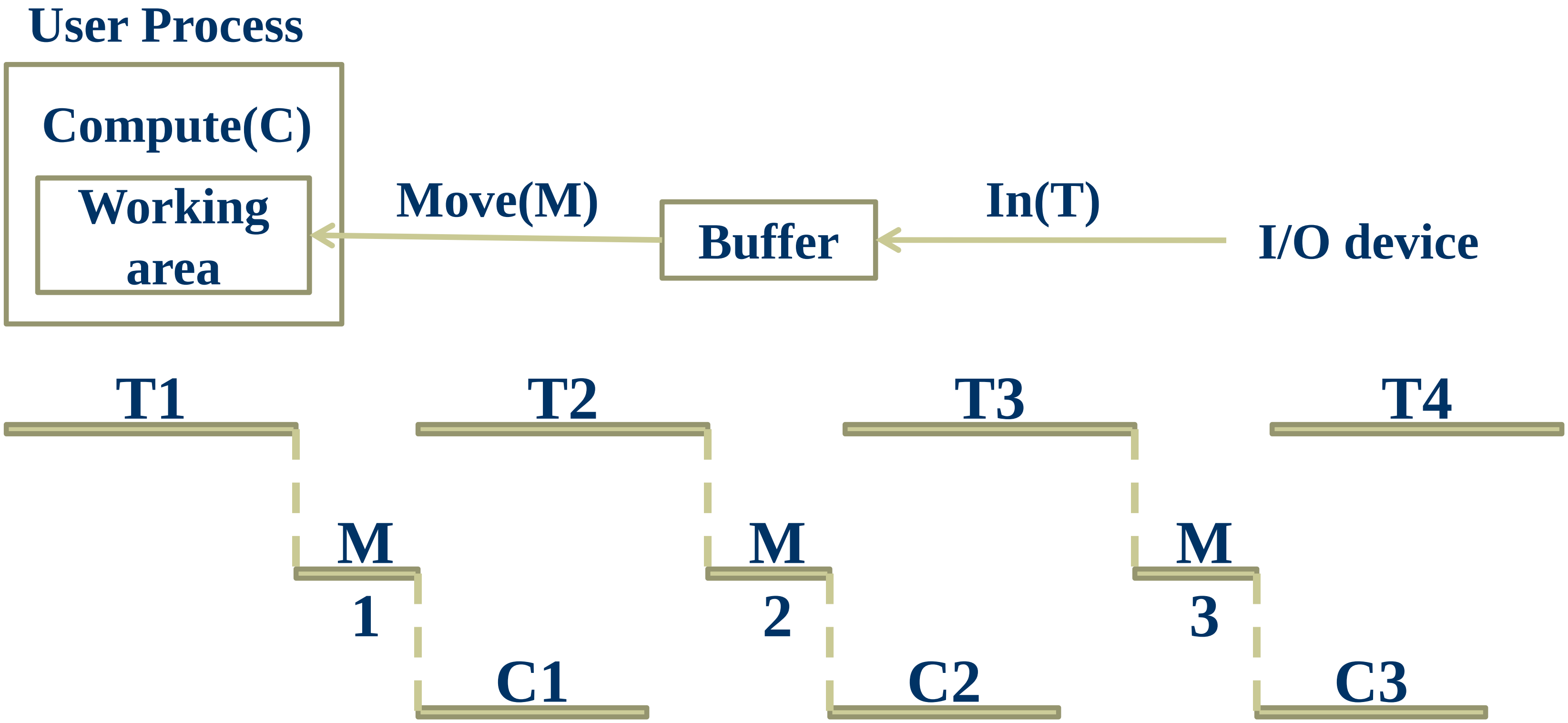
I/O Buffering
Block-Oriented & Stream-Oriented

- Two types of devices
  - Block-oriented devices stores information in blocks that are usually of fixed size, and transfers are made a block at a time
    - Generally, it is possible to reference data by its block number
    - Disks and tapes are examples
  - Stream-oriented devices transfer data in and out as a stream of bytes, with no block structure
    - Used for terminals, printers, communication ports, mouse, and most other devices that are not secondary storage

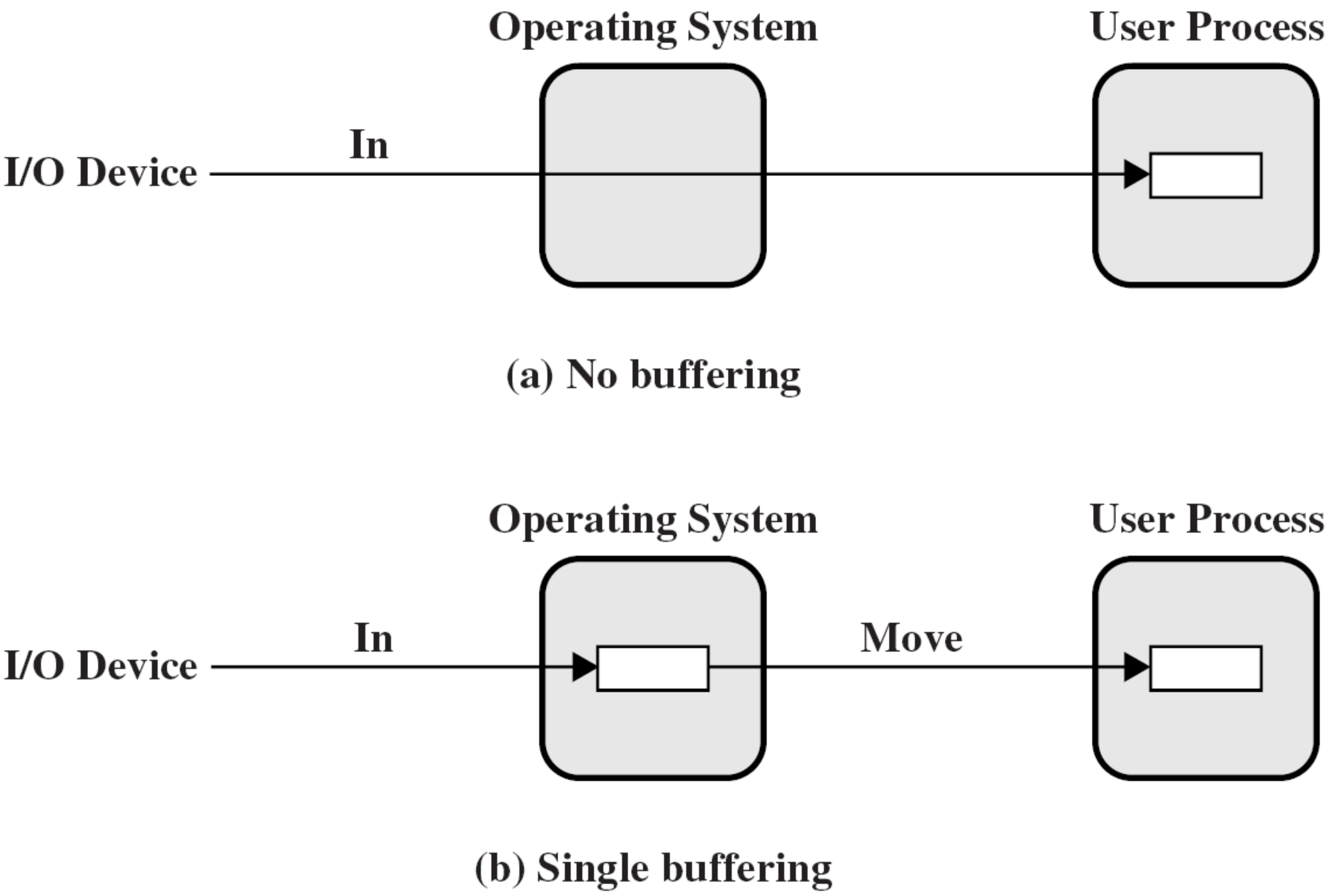
Single Buffer
单缓冲区

- When a user process issues an I/O request, the OS assigns a buffer in the system portion of main memory to the operation
  - For block-oriented input:
    - Input transfers are made to the system buffer
    - On transfer completion, the process moves the block into user space
    - Then, another block is fetched into the system buffer, in the expectation that the block will eventually be needed. [because data are usually accessed sequentially]

Single Buffer



Single Buffer



(a) No buffering

(b) Single buffering

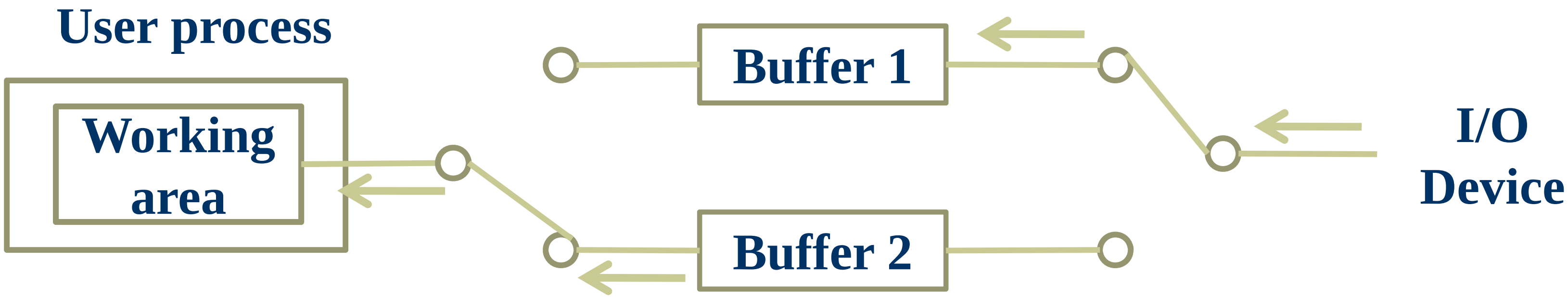
Single Buffer

- Suppose that *T* is the time required to input one block and that *C* is the computation time that intervenes between input requests
  - Without buffering, the execution per block is essentially *T*+*C*
  - With a single buffer, the time is max[*C*, *T*]+*M*
    - Here, *M* is the time required to move the data from the system buffer to user memory

Double Buffer

- An improvement over single buffering is to 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
- This technique is known as double buffering or buffer swapping

Double Buffer



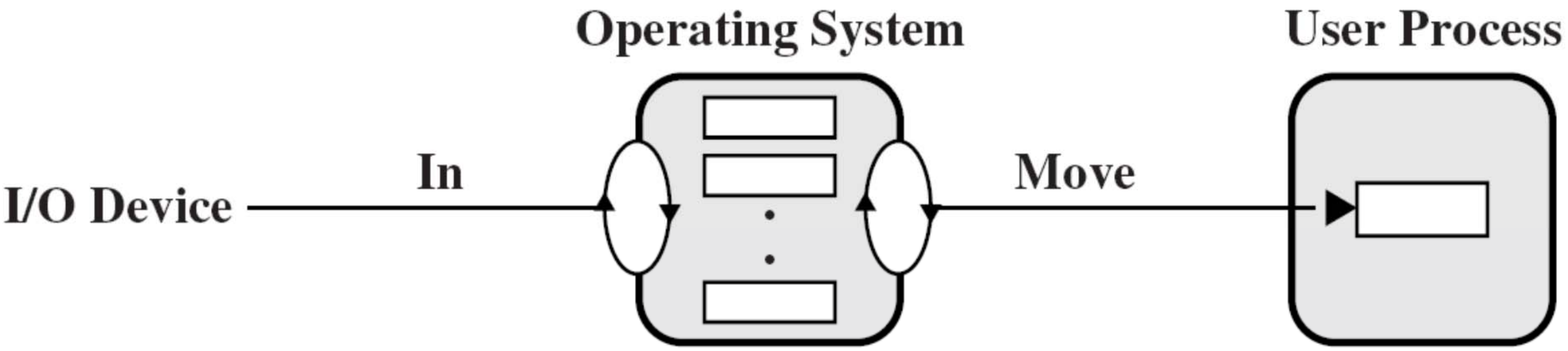
Double Buffer

- For block-oriented transfer, we can roughly estimate the execution time as max[*C*, *T*].
  - If $C \leq T$, it is possible to keep the block-oriented device going at full speed
  - If $C > T$, double buffering ensures that the process will not have to wait on I/O

Circular Buffer

- More than two buffers are used
  - Double buffering may be inadequate if the process performs rapid bursts of I/O
  - Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process
- This is simply the bounded-buffer producer/consumer model

Circular Buffer



(d) Circular buffering

The Utility of Buffering

- Buffering is a technique that smoothes out peaks in I/O demand
  - However, no amount of buffering will allow an I/O device to keep up with a process indefinitely
    - Even with multiple buffers, all of the buffers will eventually fill up and the process will have to wait after processing each chunk of data.

The Difference Between Buffer and Cache

- They are frequently combined; however, there is a difference in intent.
  - The success of cache exists mainly in that *the same datum will be read from cache multiple times*, or that *written data will soon be read*.
  - Buffering is to smooth out peaks in I/O demand

Spooling (1)
假脱机/伪脱机

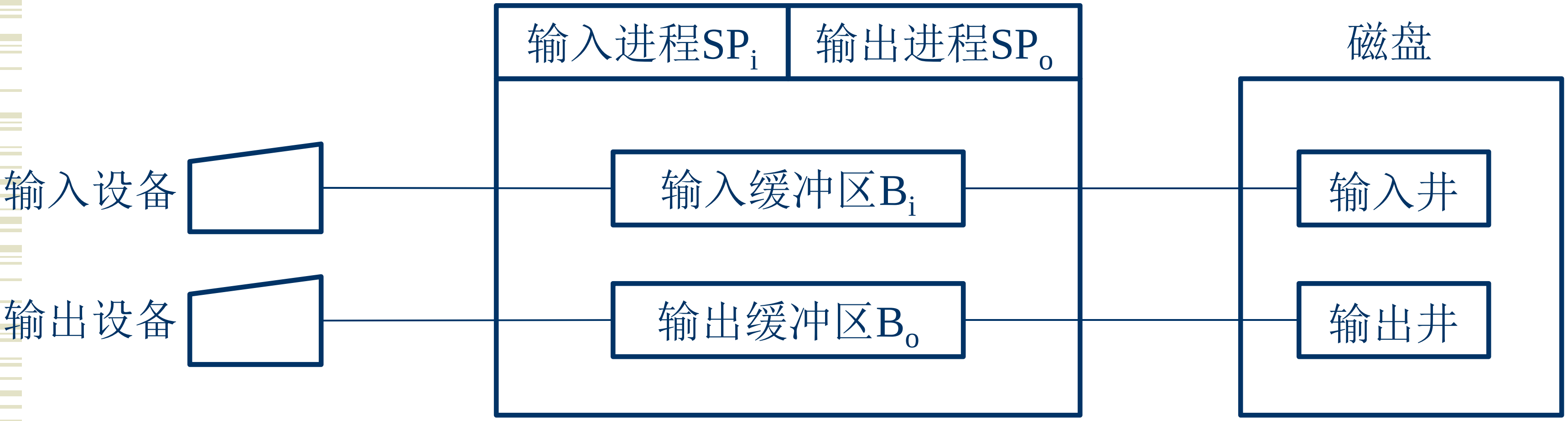
- **What is spooling?**
  - **Transferring data by placing it in a temporary working area where another program may access it for processing at a later point in time [from www.wikipedia.org]**
  - **The term "spool" is an acronym of "Simultaneous Peripheral Operation On-line"**

外部设备联机并行操作

Could be a file or storage device

SPOOLing系统的组成

- SPOOLing系统主要由三部分组成
  - 输入井和输出井；
  - 输入缓冲区和输出缓冲区；
  - 输入进程SP_i和输出进程SP_o；



输入井和输出井

- 输入井和输出井：这是在磁盘上开辟的两个大存储空间。
  - 输入井时模拟脱机输入时的磁盘设备，用于暂存I/O设备输入的数据
  - 输出井是模拟脱机输出时的磁盘设备，用于暂存用户进程的输出数据

输入缓冲区和输出缓冲区

- 为了缓和CPU和磁盘之间速度不匹配的矛盾，在内存中要开辟两个缓冲区：输入缓冲区和输出缓冲区
  - 输入缓冲区用于暂存由输入设备送来的数据，以后再传送到输入井
  - 输出缓冲区用于暂存从输出井送来的数据，以后再传送给输出设备

输入进程SP_i和输出进程SP_o

- 这里利用两个进程来模拟脱机I/O时的外围控制机。
  - 进程SP_i将用户要求的数据从输入设备通过输入缓冲区再送到输入井，当CPU需要输入数据时，直接从输入井读入内存
  - 进程SP_o把用户要求的输出数据，先从内存送到输出井，待输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备上

Spooling (2) Why spooling?

- The print spooling is the most common spooling application
  - Printers are relatively slow peripherals.
  - In comparison, disk devices are orders of magnitude faster.
  - *Without* spooling print data, the speed of program operation is constrained by the slowest device (printers) – this program is "print bounded"
- The key to spooling is asynchronous processing, where the process is not constrained by the speed of slow devices (particularly printers)

Spooling (3)
Behind the scenes

- A spooler contains two parts:
  - An operating system extension to trap data destined for a printer and buffers it.
  - A simple program that independently writes trapped data to the printer.
- With spooling
  - A spooling mechanism traps the I/O request, captures the output data, and releases the application to continue processing.
  - Afterwards, it writes the captured data to the printer, independent of the original application.

用共享设备来实现多个虚拟独占设备--“虚拟打印机”

Spooling (4) Summary

设备虚拟技术

- *Spooling is a way of dealing with dedicated I/O devices in a multiprogramming system*
  - *Without spooling, if a user process opened the character special file for the printer and then did nothing for hours, no other process could print anything*
- *Spooling can be managed by a system daemon process or an in-kernel thread.*
  - *To print a file, a process first generates the entire file to be printed and puts it in the spooling directory.*
  - *Only the daemon process has the permission to use the printer's special file to print the files in the directory.*

Magnetic Disk
磁盘

Magnetic Disk

- A disk is a circular platter (圆盘) constructed of metal or plastic, coated with a magnetizable (可磁化) material
- Data recorded on the disk are retrieved via a conducting coil (导体线圈) named the head (磁头)
  - During a read or write operation, the head is stationary, while the platter rotates beneath it.

Data Organization

- Data on the platter are organized in a concentric set of rings, called track (磁道).
  - Each track is the same width as the head.
- Adjacent tracks are separated by gaps (间隙)
- Data are transferred to and from the disk in blocks.
  - Typically, the block is smaller than the capacity of the track
  - Accordingly, data are stored in block-size regions known as sectors (扇区)

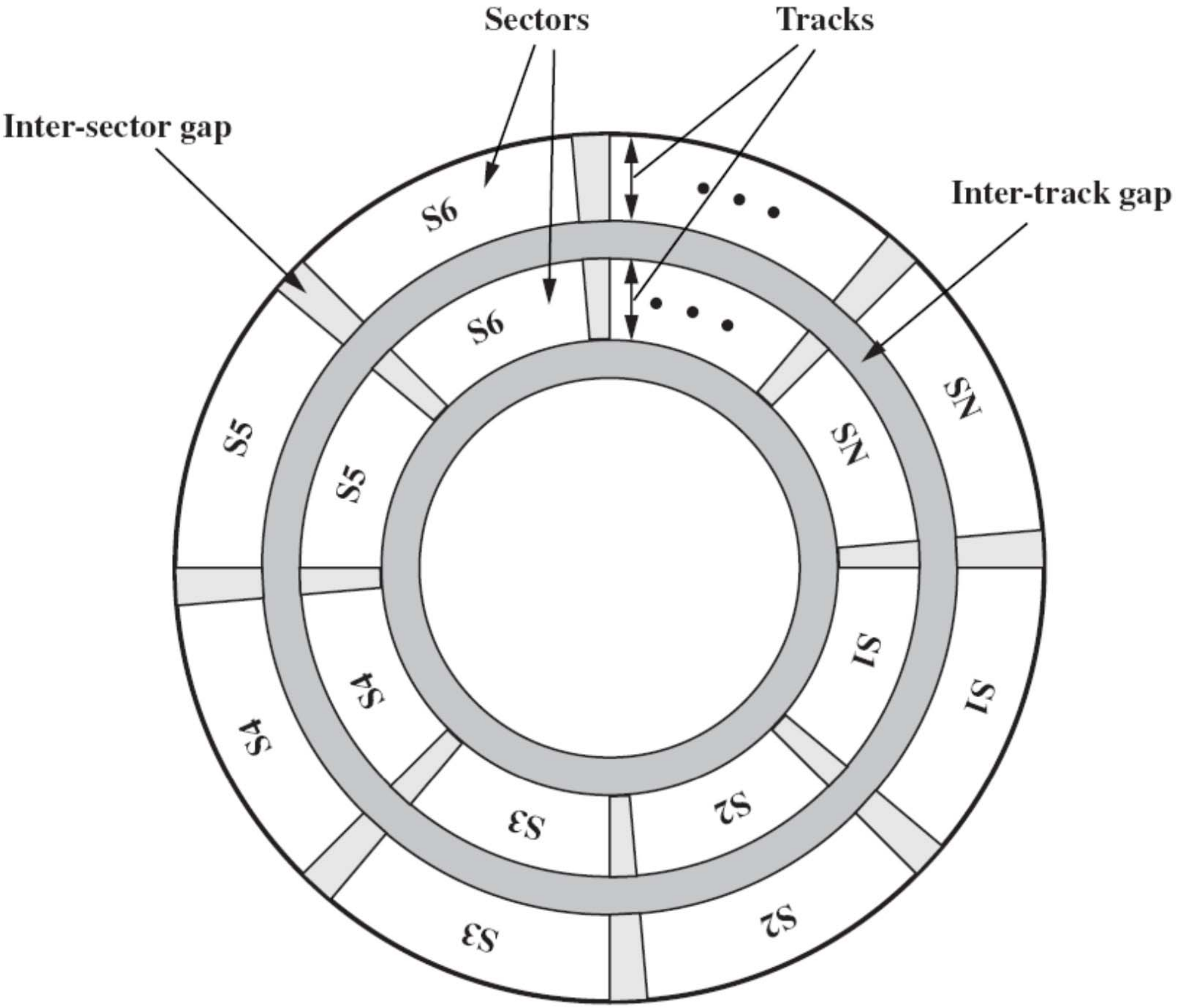


Figure 11.17 Disk Data Layout

Physical Characteristics
Fixed-Head versus Movable-Head

- In a fixed-head (固定头) disk, there is one read/write head per track
  - All the heads are mounted on a rigid arm (刚性的磁头臂) that extends across all tracks
- In a movable-head (活动头) disk, there is only one head.
  - The arm can be extended or retracted to position the head above any track

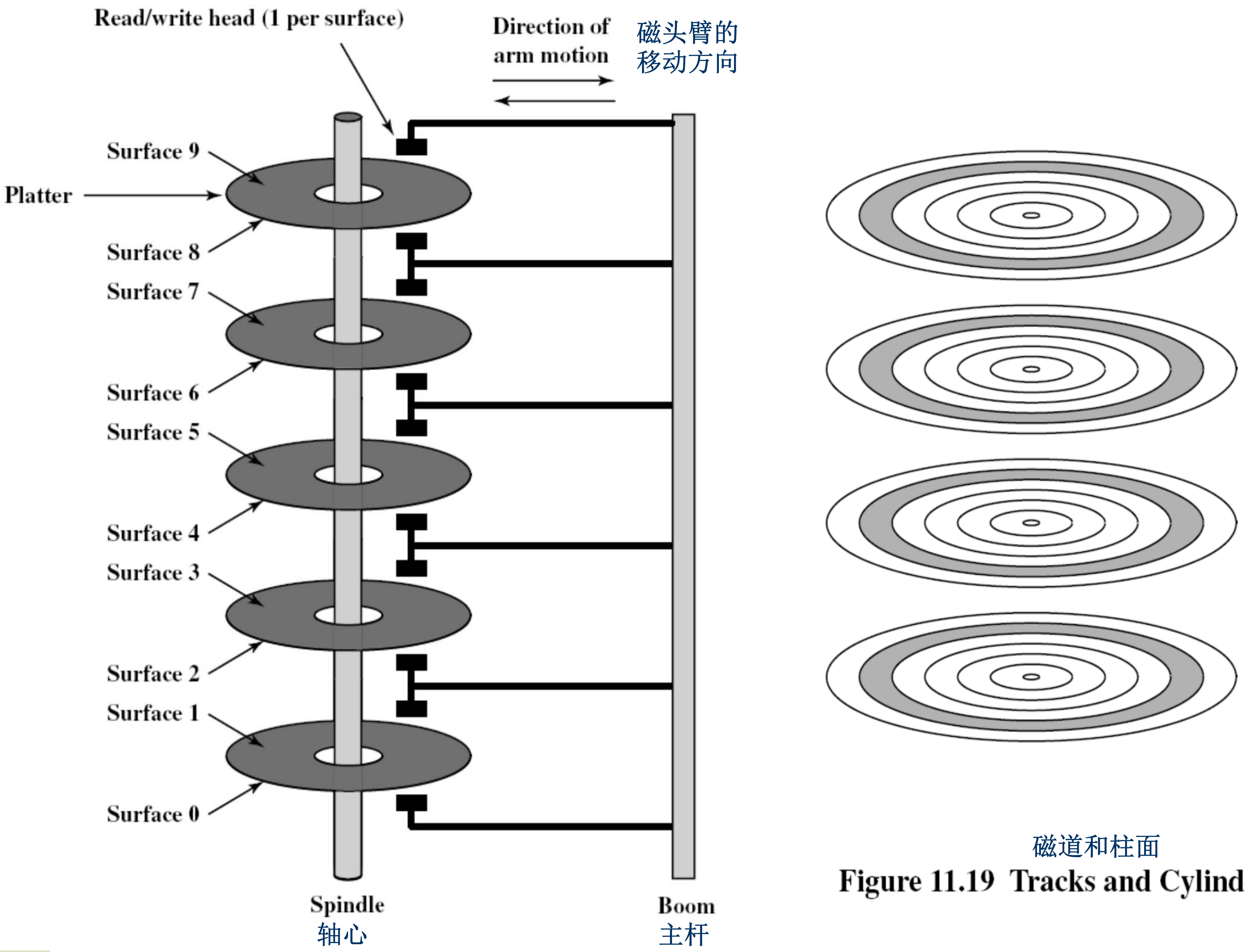
Physical Characteristics
Nonremovable versus Removable

- The disk itself is mounted in a disk drive
  - The disk drive consists of the arm, a spindle (轴) that rotates the disk, and the electronics needed for the I/O
- A nonremovable (不可移动) disk is permanently mounted in the disk drive
- A removable disk can be removed and replace with another disk
  - Such as floppy disk or ZIP disk

Physical Characteristics

Single Platter versus Multiple Platter

- Some disk drives accommodate multiple platters stacked vertically about an inch apart
  - Multiple arms are provided.
  - The platters come as a unit known as a disk pack (磁盘组)
  - One head per platter surface. All heads are at the same distance from the center of the disk and move together.
- The combination of all the tracks in the same relative position on the platter is referred to as a cylinder (柱面)



磁道和柱面

Figure 11.19 Tracks and Cylinders

Disk Scheduling
磁盘调度

Disk Performance Parameters 磁盘性能参数

- To read or write, the disk head (磁头) must be positioned at the desired track(磁道) and at the beginning of the desired sector(扇区)
- Seek time (寻道时间)
  - On a **movable-head** system, the time it takes to position the head at the desired track
- Rotational delay or rotational latency (旋转延迟)
  - The time it takes for the beginning of the sector to reach the head

Disk Performance Parameters

- Access time
  - Sum of seek time and rotational delay
  - The time it takes to get in position to read or write
- Data transfer occurs as the sector moves under the head

Disk Performance Parameters
Seek Time

- The seek time consists of two key components:
  - The initial startup time
  - The time taken to traverse the tracks
- The traversal time is not a linear function of the number of tracks
- A typical average seek time on contemporary disks is 5 to 10ms

Disk Performance Parameters Rotational Delay

- Magnetic disks, other than floppy disk, have rotational speeds in the range 5400 to 10,000 rpm
  - 10,000 rpm equivalent to one revolution per 6 ms
    - Thus, at 10,000 rpm, the average rotational delay will be 3ms
- Floppy disks typically rotates at between 300 and 600 rpm.
  - Thus, the average delay will be between 100 and 200 ms

Disk Performance Parameters Transfer Time

- The transfer time to/from the disk depends on the rotation speed of the disk: $T=b/(rN)$
  - T: transfer time
  - b: number of bytes to be transferred
  - N: number of bytes on a track
  - r: Rotation speed, in revolutions per second

Disk Performance Parameters Total Average Access Time

- $T_a = T_s + 1/(2r) + b/(rN)$
  - T_s is the average seek time

An Example - Description

- A typical disk
  - average seek time of 10ms
  - rotation speed of 10,000 rpm
  - 512-byte sectors with 320 sectors per track
- Suppose that we wish to read a file consisting of 2560 sectors for a total of 1.3Mbytes
  - How about the total time for the transfer?

An Example
One Situation

- If the file is stored as compactly as possible on the disk
  - The file occupies all the sectors on 8 adjacent tracks (8 tracks*320 sectors/track = 2560 sectors)
  - The time to read the first track is about 19ms: 10ms(average seek)+3ms(rotational delay)+6ms(read 320 sectors)
  - Suppose that the remaining tracks can now be read with essentially no seek time. Each successive track is read in 3+6=9ms.
  - Total time to read the entire file is 19+7*9=82ms=0.082 seconds

An Example
The Other Situation

- If the accesses to the sectors are distributed randomly over the disk
  - For each sector, we have 10ms(average seek)+3ms(rotational delay)+0.01875ms(read 1 sector)=13.01875ms
  - Total time=2560*13.01875=33.328 seconds

Conclusion

- It is clear that the order in which sectors are read from the disk has a tremendous effect on I/O performance

Disk Scheduling Policies

- Seek time is the main reason for differences in performance
- In multiprogramming, the OS maintains a queue of requests for each I/O device
  - For a single disk there will be a number of I/O requests from various processes
- If requests are selected randomly, we will get the worst possible performance

Disk Scheduling Policies First-In-First-Out

- The simplest scheduling would be first-in-first-out (FIFO)
  - We process requests from the queue in sequential order
  - It is fair to all processes
  - It approaches random scheduling in performance if there are many processes competing for the disk
    - If there are only a few processes that require access and if many of the requests are to the clustered file sectors, then we can hope good performance for FIFO

Disk Scheduling Policies
Priority

- Priority
  - Goal is not to optimize disk use but to meet other objectives
  - Short batch jobs may have higher priority
    - Longer jobs may have to wait excessively long times
  - Provide good interactive response time

Disk Scheduling Policies
Shortest Service Time First

- The SSTF policy select the disk I/O request that requires the least movement of the disk arm from its current position
  - Always choose the minimum Seek time
  - The arm can move in two directions, a random tie-breaking algorithm may be used
- It does not guarantee that the average seek time over a number of arm movements will be minimum
  - Can you give out an example?

Disk Scheduling Policies
SCAN

This refinement is called the **LOOK** policy

- SCAN
  - Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction or until there is no more requests in that direction
  - The service direction is then reversed and the scan proceeds in the opposite direction
  - It is also called the elevator (电梯) algorithm

Disk Scheduling Policies C-SCAN

- **C-SCAN (circular SCAN循环SCAN)**
  - **It restricts scanning to one direction only**
  - **When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again**
- **This reduces the maximum delay experienced by new requests. It provides a more uniform wait time**
  - **With SCAN, if the expected time for a scan from inner track to outer track is t, the maximum delay of a request $2t$**
  - **With C-SCAN, the maximum delay of a request is $t+s_{max}$, where s_{max} is the maximum seek time**

An Example

- Assume a disk with 200 tracks
- The requested tracks, in the order received, are 55, 58, 39, 18, 90, 160, 150, 38, 184.
- Starting at track 100

Arm Stickiness
磁头臂的粘性(粘滞性)

- With SSTF, SCAN and C-SCAN, it is possible that the arm may not move for a considerable period of time
  - For example, if one or a few processes have high access rates to one track, they can monopolize (垄断) the entire device by repeated requests to that track.
- High-density multisurface disks are more likely to be affected by this characteristic than low-density disks and/or disks with only one or two surfaces.
- To avoid arm stickiness, let us have a look at two approaches: N-step-SCAN and FSCAN

Disk Scheduling Policies N-Step-SCAN

- N-step-SCAN
  - Segments the disk request queue into subqueues of length N
  - Subqueues are processed one at a time, using SCAN
  - New requests added to other queue when queue is processed
- With large values of N, N-step-SCAN approaches SCAN
- With a value of N=1, it is the same as FIFO

Disk Scheduling Policies FSCAN

- FSCAN
  - Two queues
  - When a scan begins, all the requests are in one queue, with another queue empty
  - During the scan, all new requests are put into the other queue

Disk Scheduling Algorithms

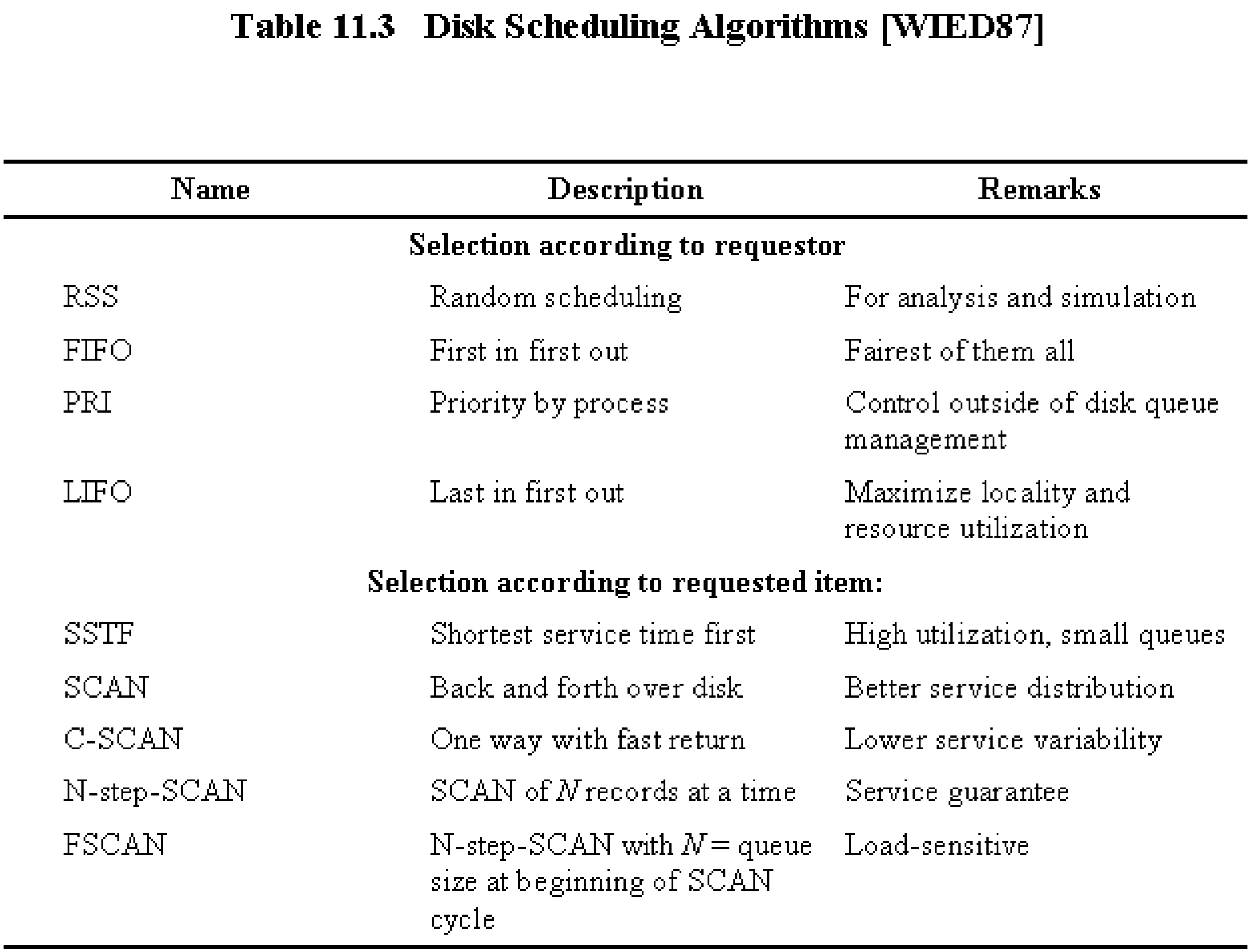
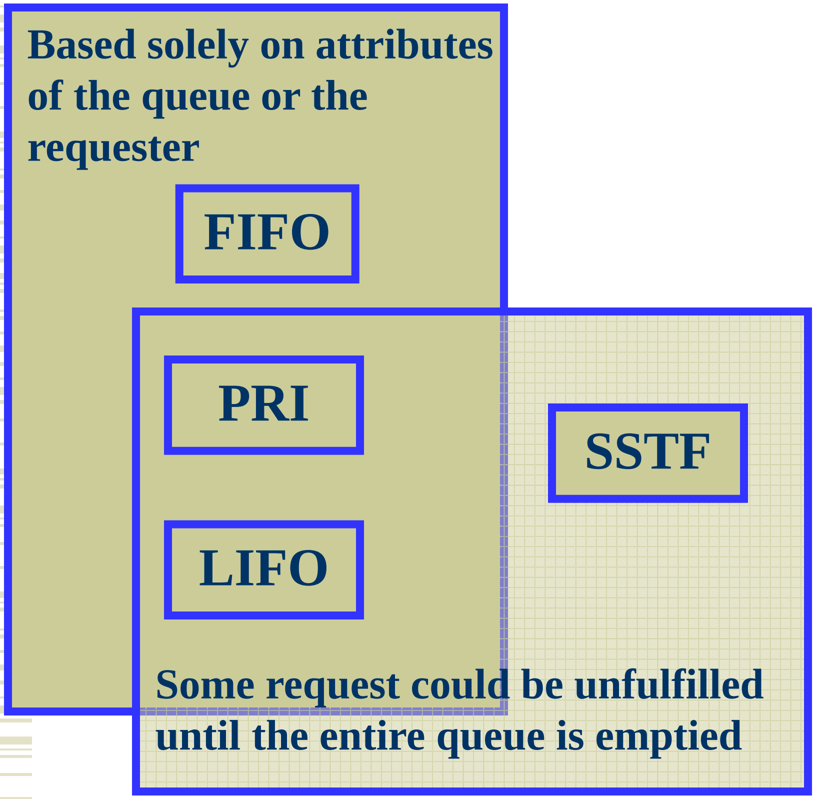


Table 11.3 Disk Scheduling Algorithms [WIED87]

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

Summary



Based solely on attributes of the queue or the requester

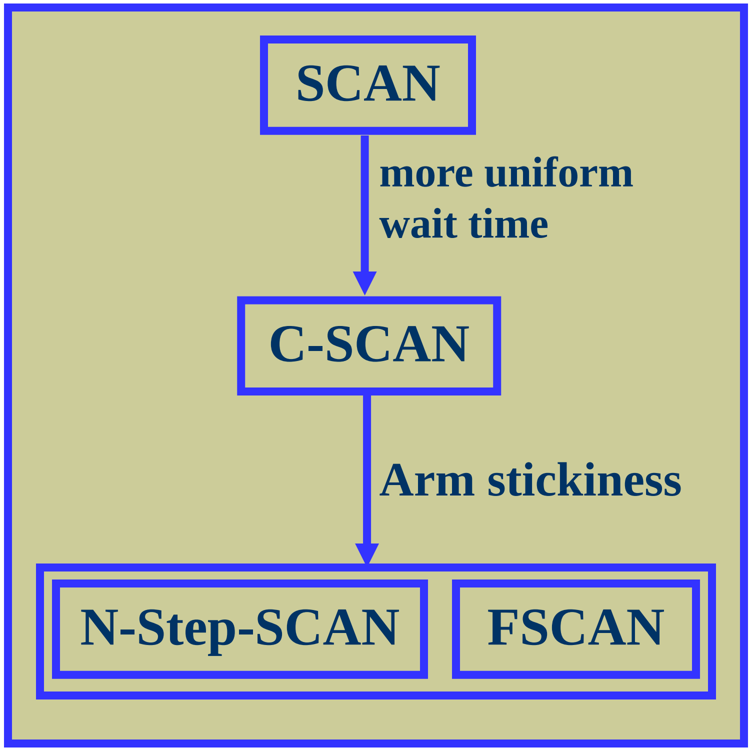
FIFO

PRI

LIFO

SSTF

Some request could be unfulfilled until the entire queue is emptied



SCAN

more uniform wait time

C-SCAN

Arm stickiness

N-Step-SCAN

FSCAN