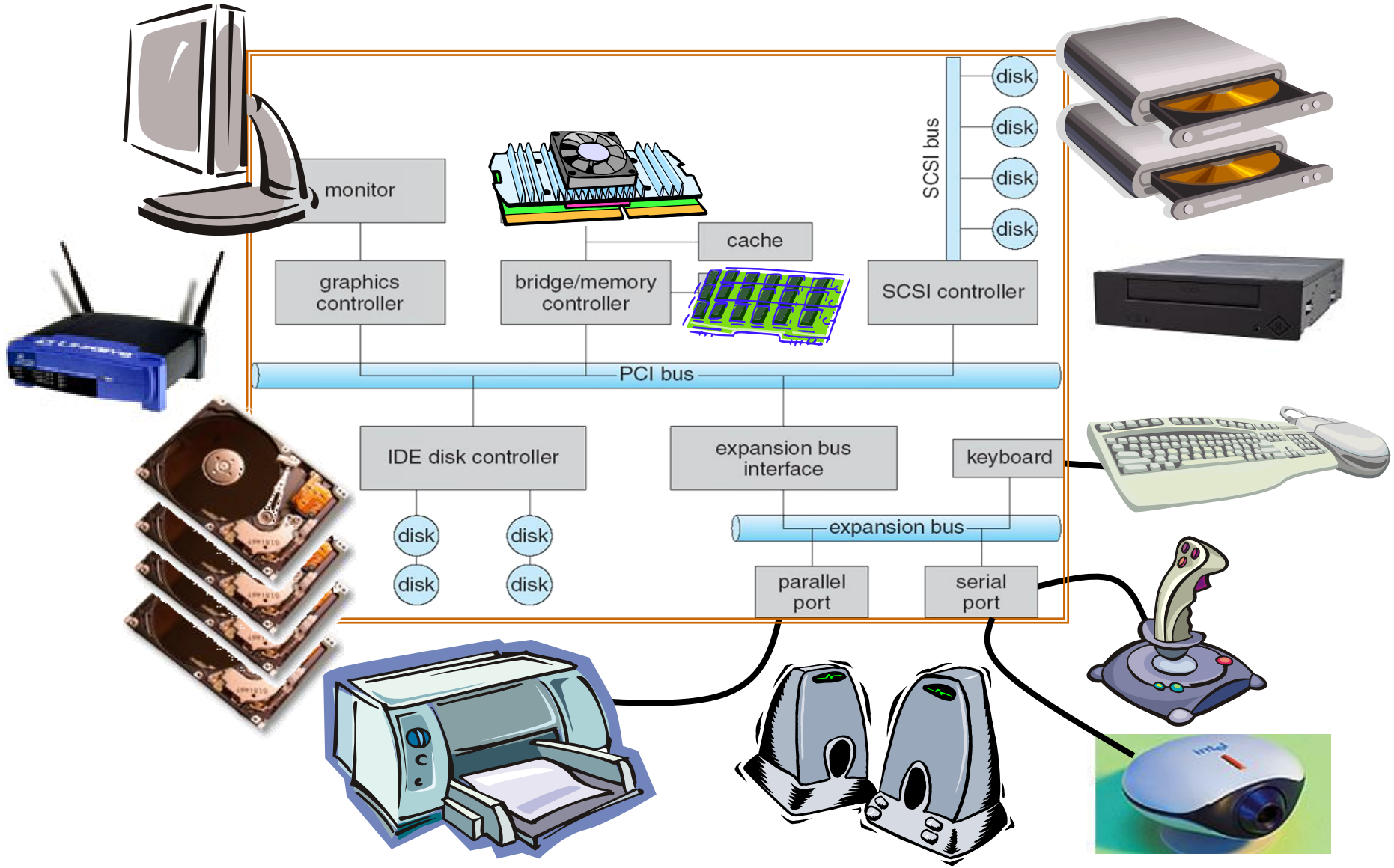
# I/O System

# Modern I/O Systems



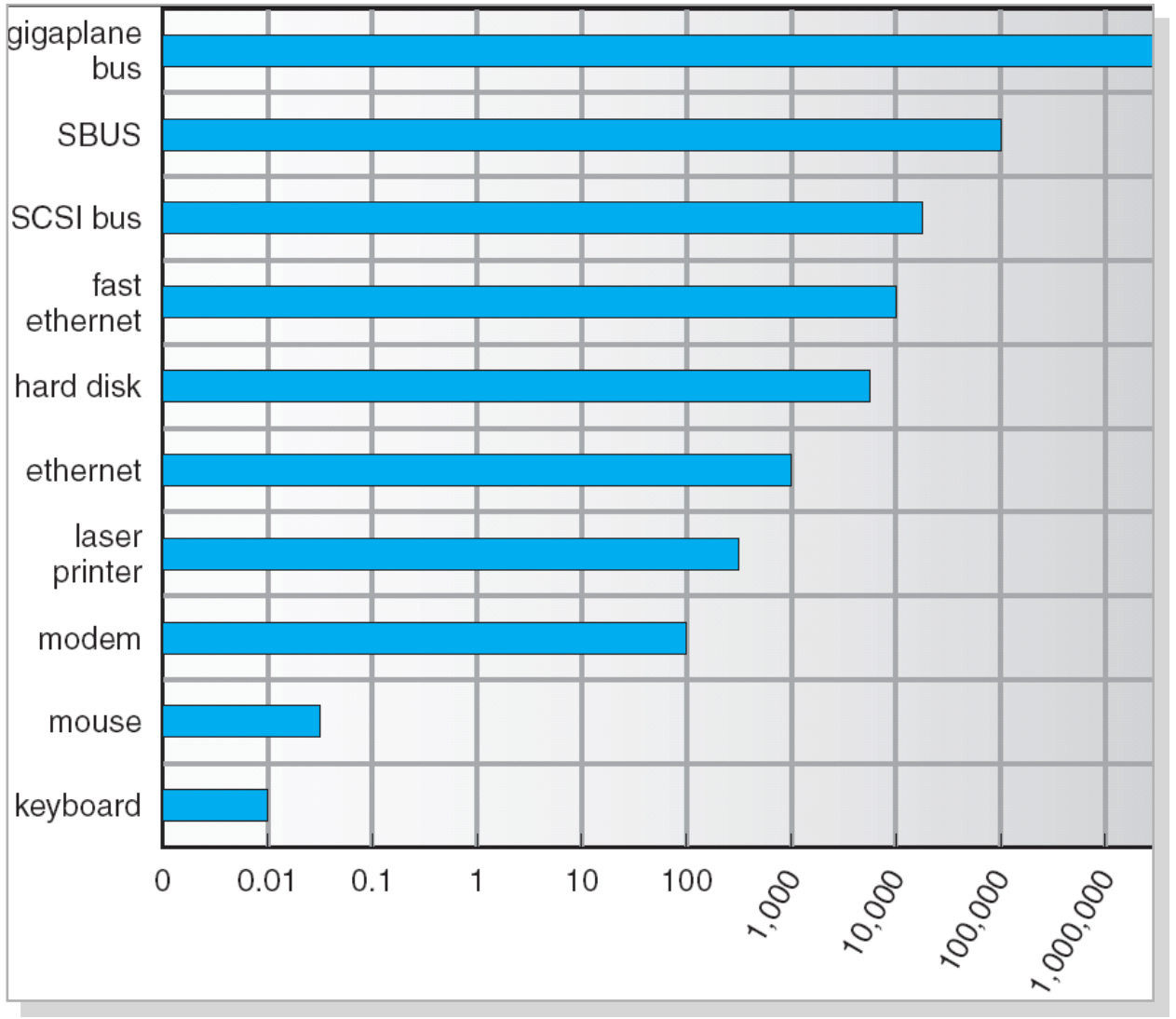
# The Requirements of I/O

- So far in this course:
  - We have learned how to manage CPU, memory……
- What about I/O?
  - Without I/O, computers are useless
  - But…
    - Thousands of devices, each slightly different
    - Devices unreliable: media failures and transmission errors
    - Devices unpredictable and/or slow

# Three Main Problems

- Thousands of devices, each slightly different
  - How can we standardize the interfaces to these devices?
- Devices unreliable: media failures and transmission errors
  - How can we make them reliable???
- Devices unpredictable and/or slow
  - How can we manage them if we don't know what they will do or how they will perform?

# Example: Device-Transfer Rates




Kbit/second

# I/O Devices

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

# I/O Devices

- Block Devices: *e.g.* disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Commands include `open()`, `read()`, `write()`, `seek()`
- Character Devices: *e.g.* keyboards, mouse, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
- Network Devices: *e.g.* Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface

# I/O Devices

- Sequential/Random
  - Some devices must be accessed sequentially (*e.g.* tape)
  - Others can be accessed randomly (*e.g.* Disk, CD, etc.)

# Characteristics of I/O Devices

| aspect | variation | example |
|---|---|---|
| data-transfer mode | character<br>block | terminal<br>disk |
| access method | sequential<br>random | modem<br>CD-ROM |
| transfer schedule | synchronous<br>asynchronous | tape<br>keyboard |
| sharing | dedicated<br>sharable | tape<br>keyboard |
| device speed | latency<br>seek time<br>transfer rate<br>delay between operations | |
| I/O direction | read only<br>write only<br>read–write | CD-ROM<br>graphics controller<br>disk |

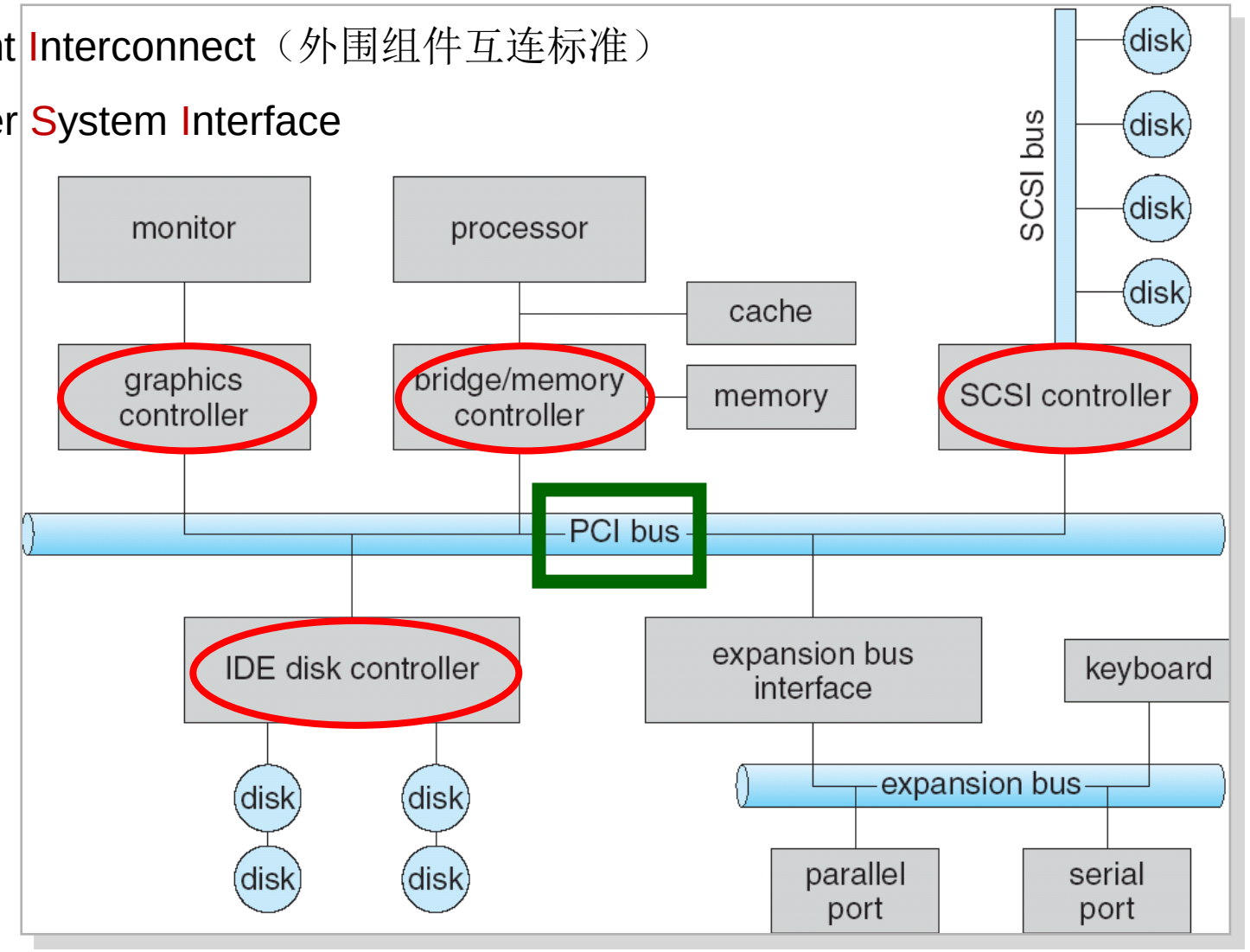
# I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - **Port** -- address
  - **Bus**
  - **Controller** (**host adapter**)
    - Contains a set of registers that can be read and written
    - May contain memory
- I/O instructions control devices
- Devices have addresses

# A Typical PCI Bus Structure

Peripheral Component Interconnect（外围组件互连标准）

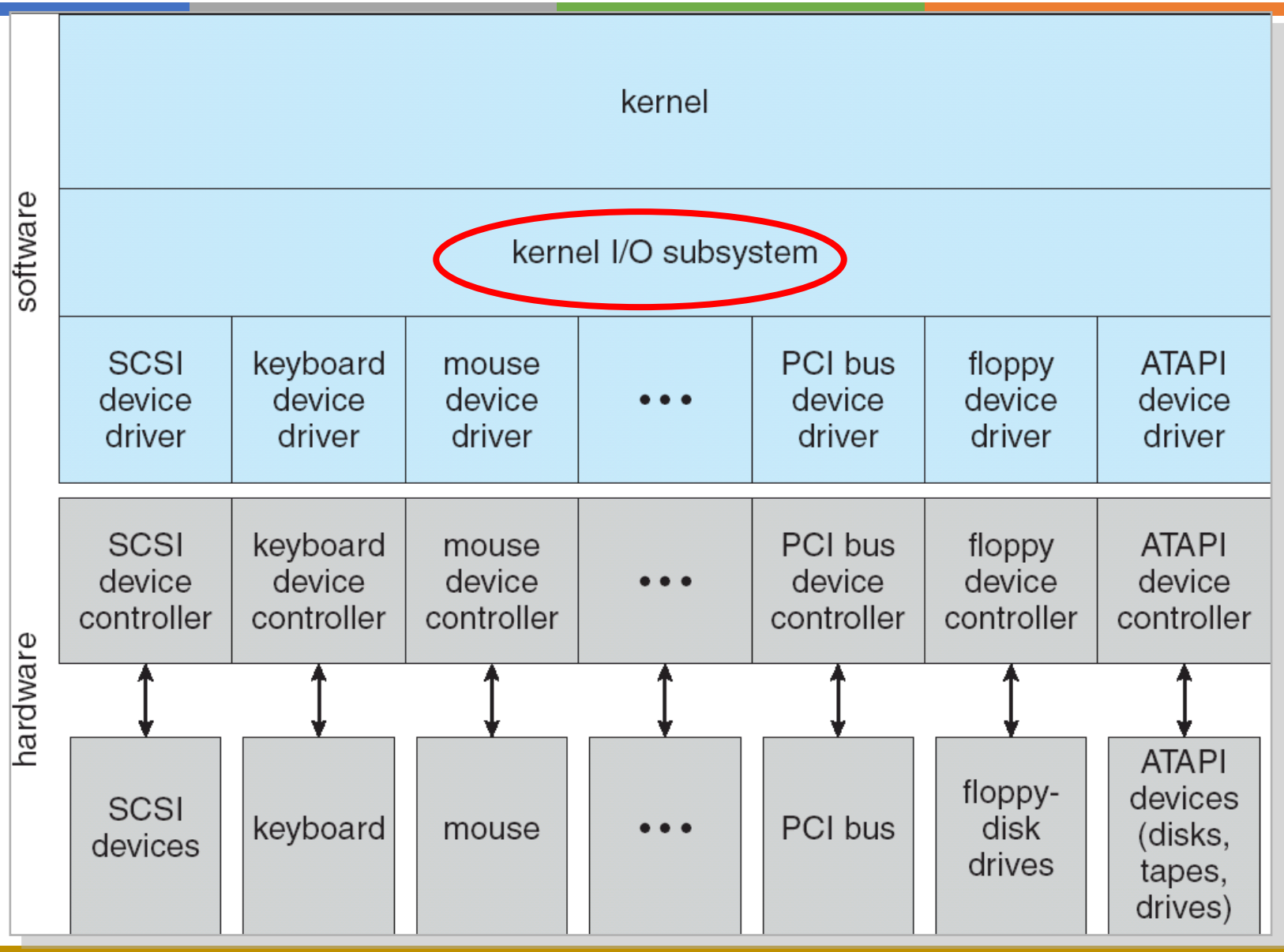
SCSI: Small Computer System Interface



# 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) |

# A Kernel I/O Structure




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

# The Goal of I/O Subsystem

- Provide Uniform Interfaces, Despite Wide Range of Different Devices
  - This code works on many different devices:

```
int fd = open("/dev/something");
for (int i = 0; i < 10; i++) {
  fprintf(fd,"Count %d\n",i);
}
close(fd);
```

  - Why? Because code that controls devices ("device driver") implements standard interface.

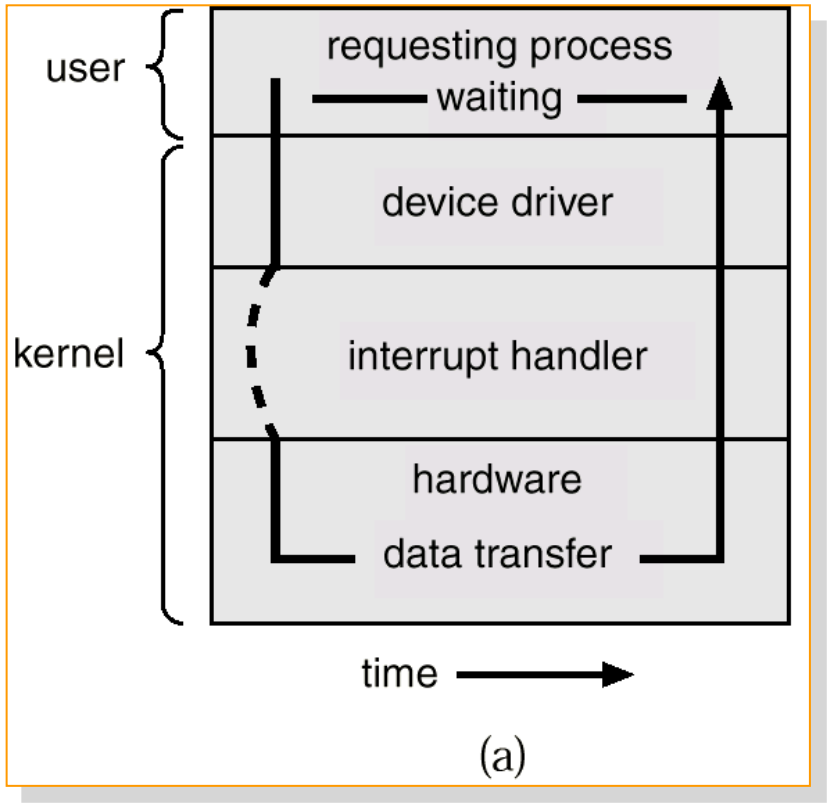
# Three Main Problems

- Incredible variety of I/O devices.
- Unpredictable and/or slow.
- Unreliable.

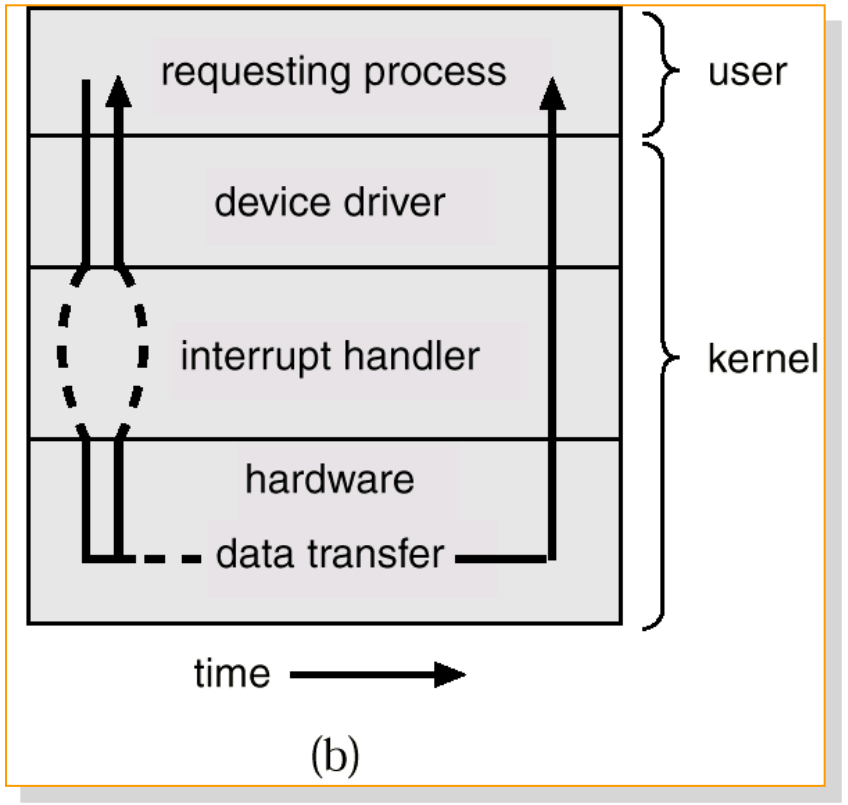
- I/O organization
- Device management

# I/O Organization

# Two I/O Methods




Synchronous




Asynchronous

# Two I/O Methods

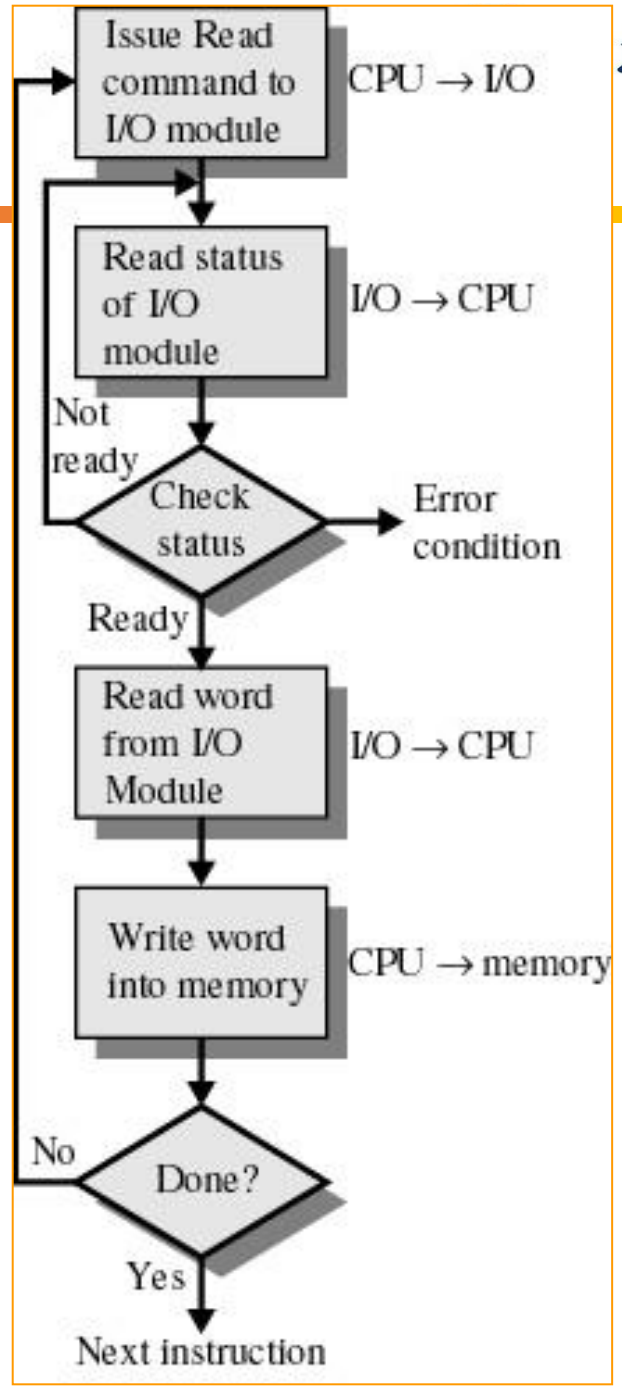
- **Synchronous** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

# Two I/O Methods

- Synchronous: "Wait"
  - When request data (e.g. `read()` system call), put process to sleep until data is ready
  - When write data (e.g. `write()` system call), put process to sleep until device is ready for data
- Asynchronous Interface: "Tell Me Later"
  - When request data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
  - When send data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

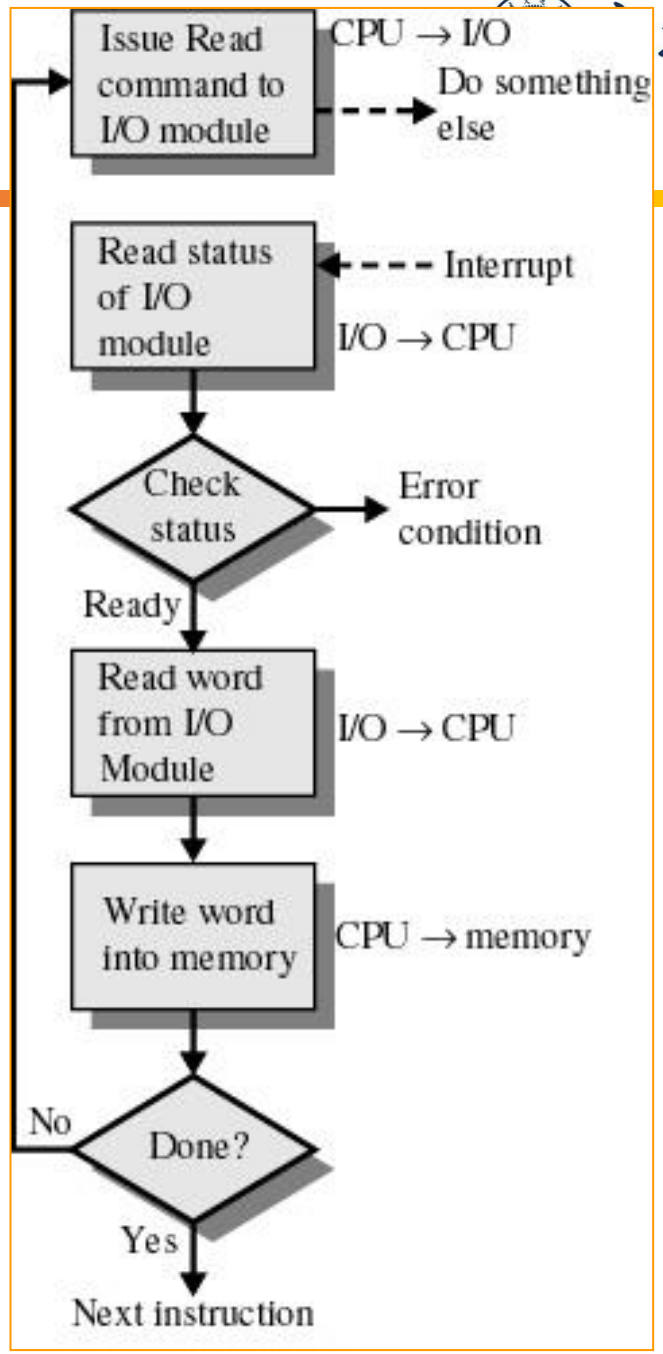
# Polling

- Determines state of device
  - Ready or Busy or Error
- The host repeatedly reads the busy bit until that bit becomes clear

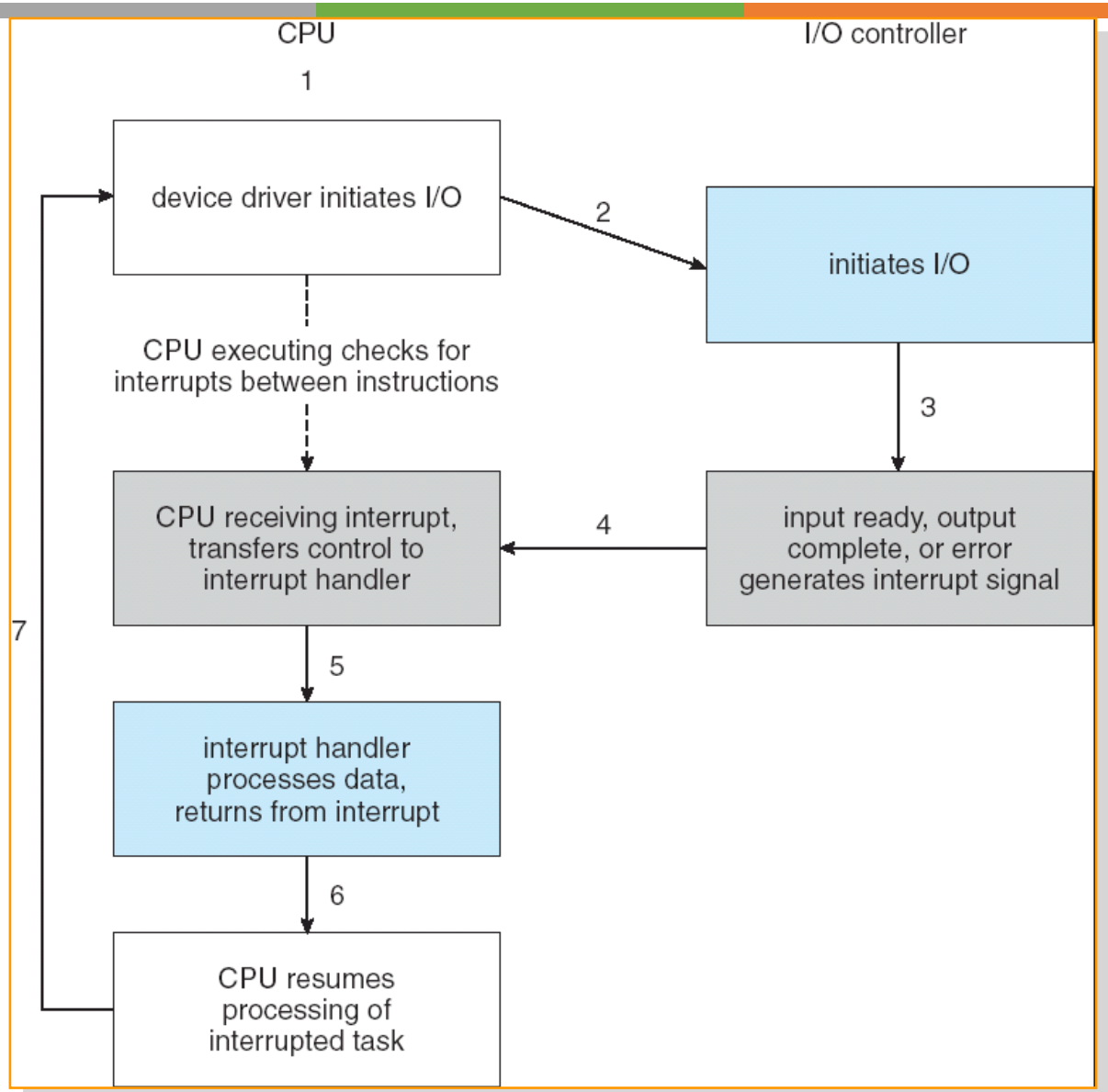


# Interrupts

- CPU is responsible for
  - moving chars to/from controller buffer, but
- Interrupt signal informs CPU when I/O operation completes



# Interrupt-Driven I/O Cycle

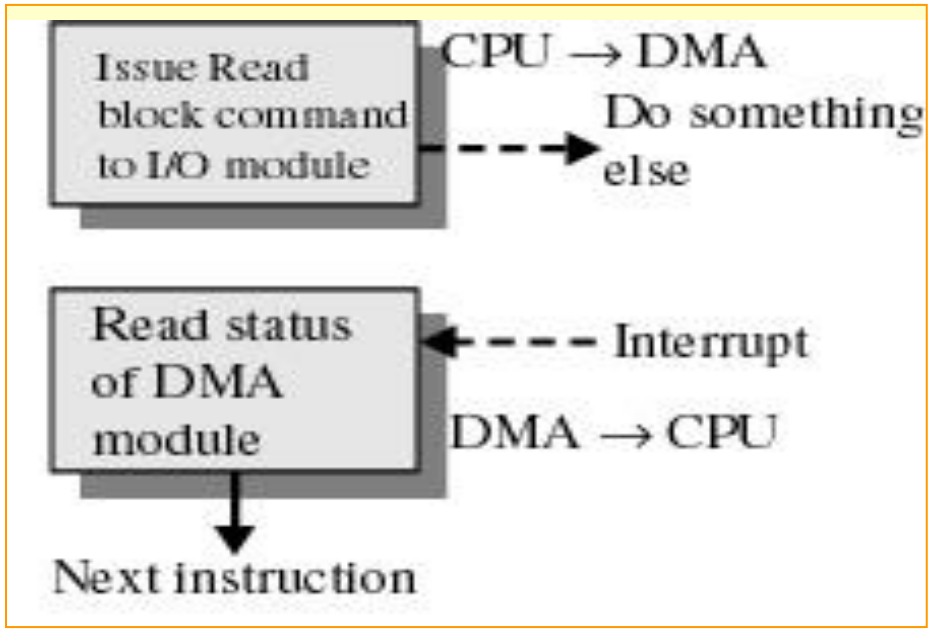


# Direct Memory Access

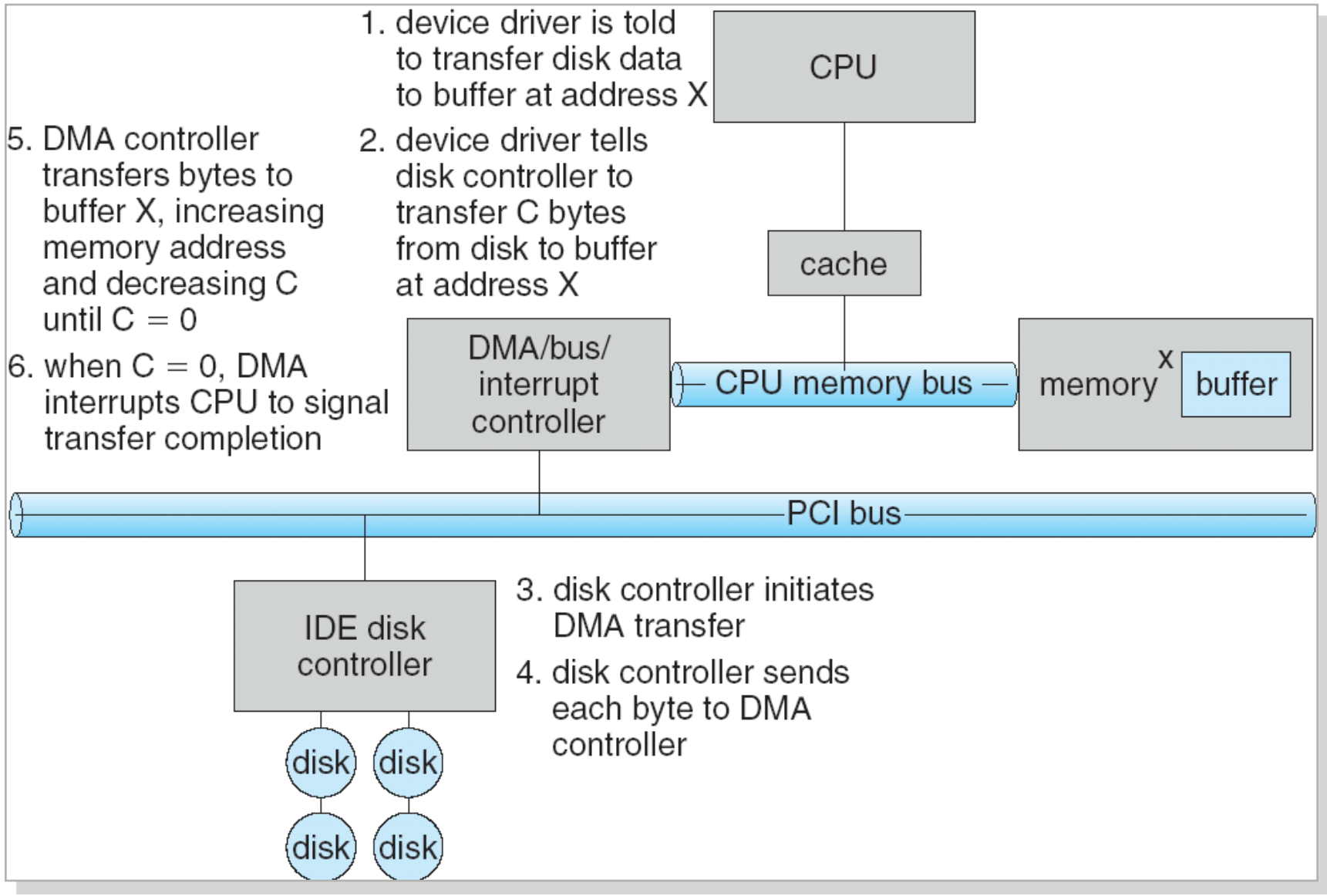
- Used to avoid **programmed I/O** for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory

# DMA

- CPU only initiates operation
- DMA controller transfers data directly to/from main memory
- Interrupt when transfer completed



# Six Step Process to Perform DMA Transfer



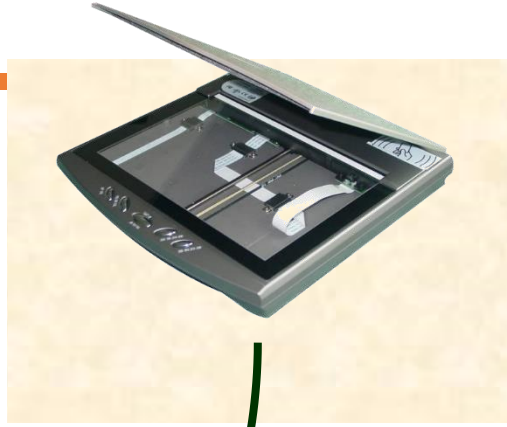
# Device Management

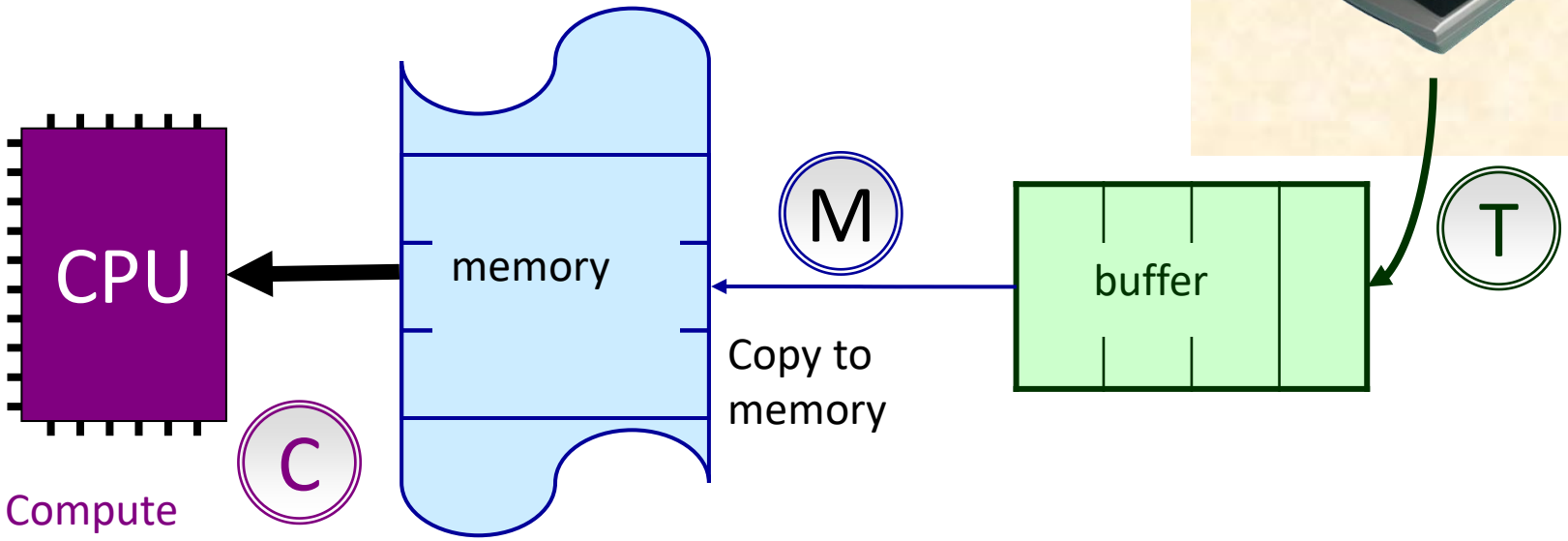
# Device Management

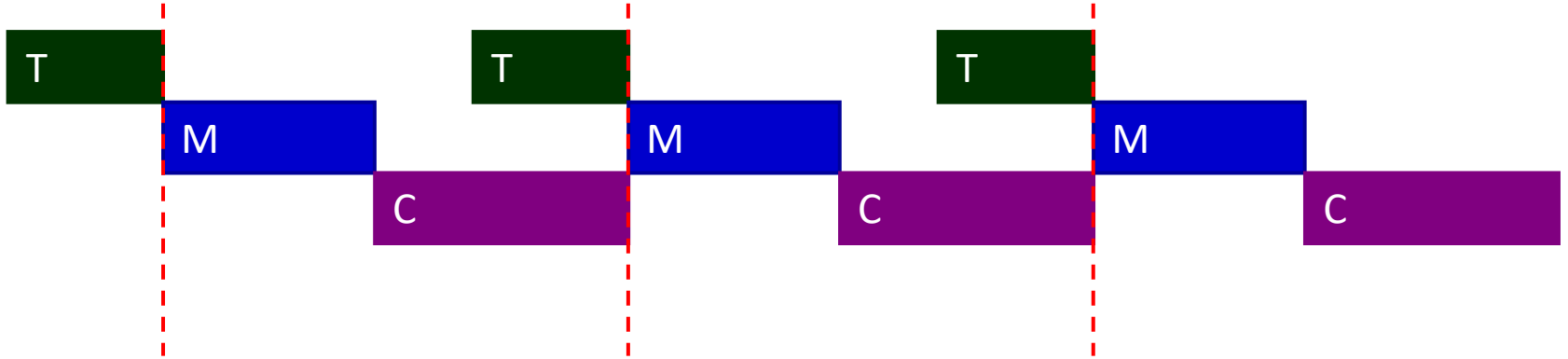
- Scheduling
- Error handling
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch

# Device Management

## Single Buffer

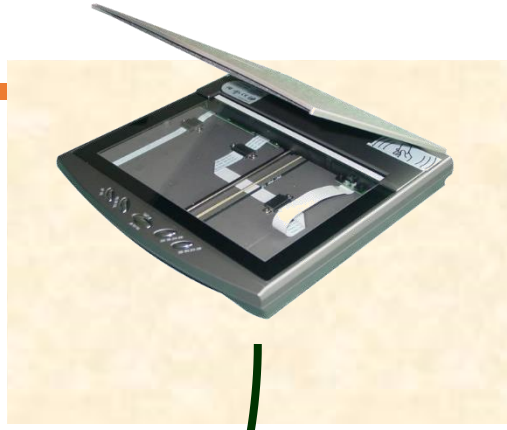


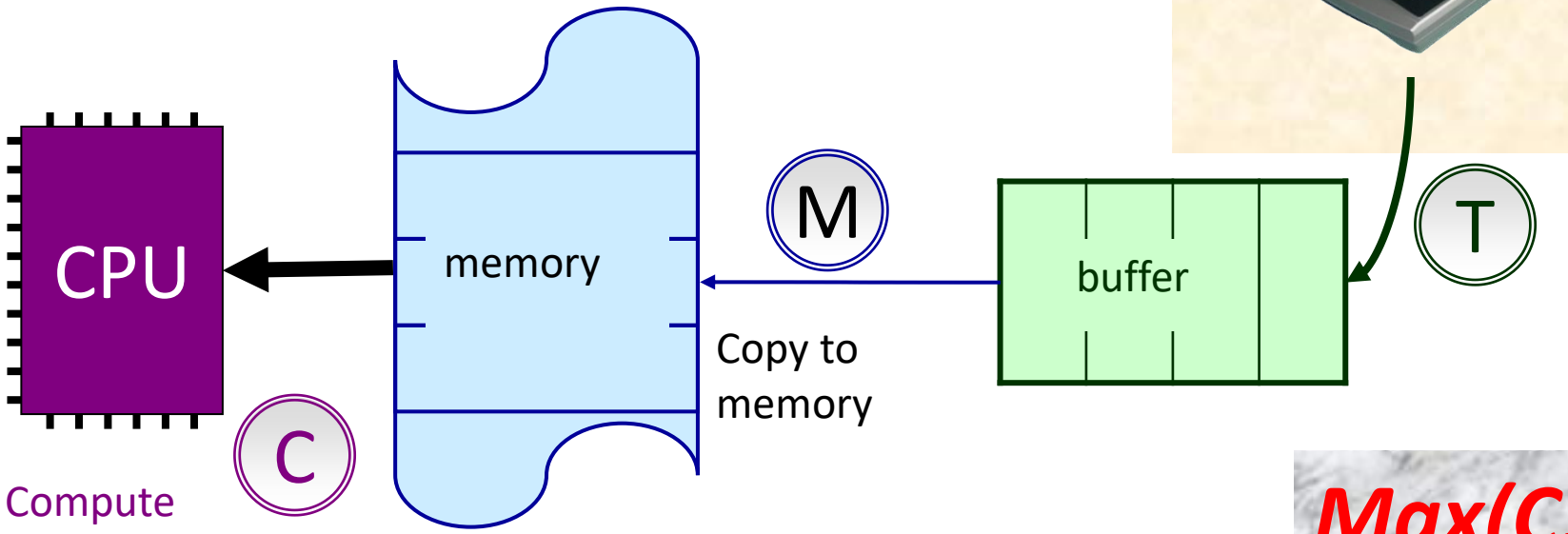


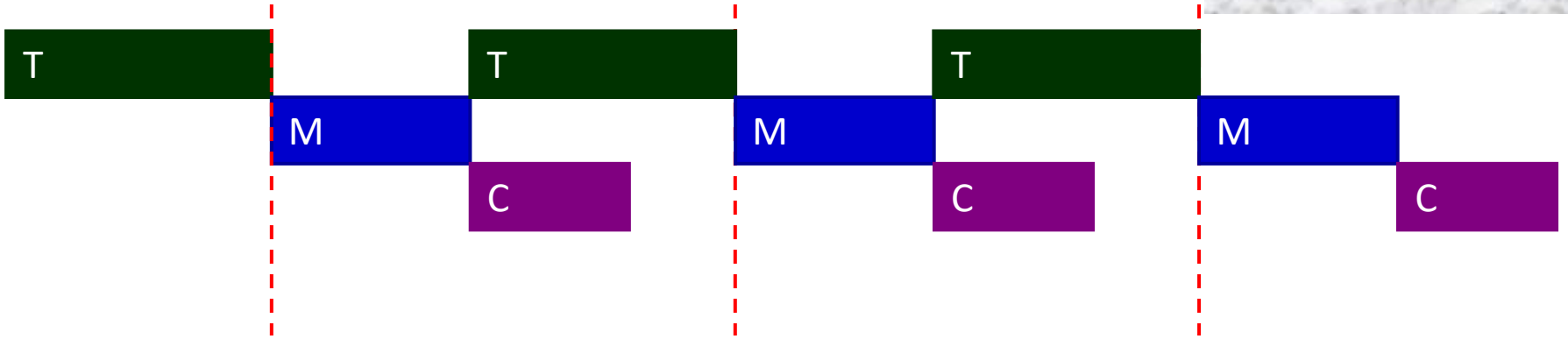
# Device Management

## Single Buffer





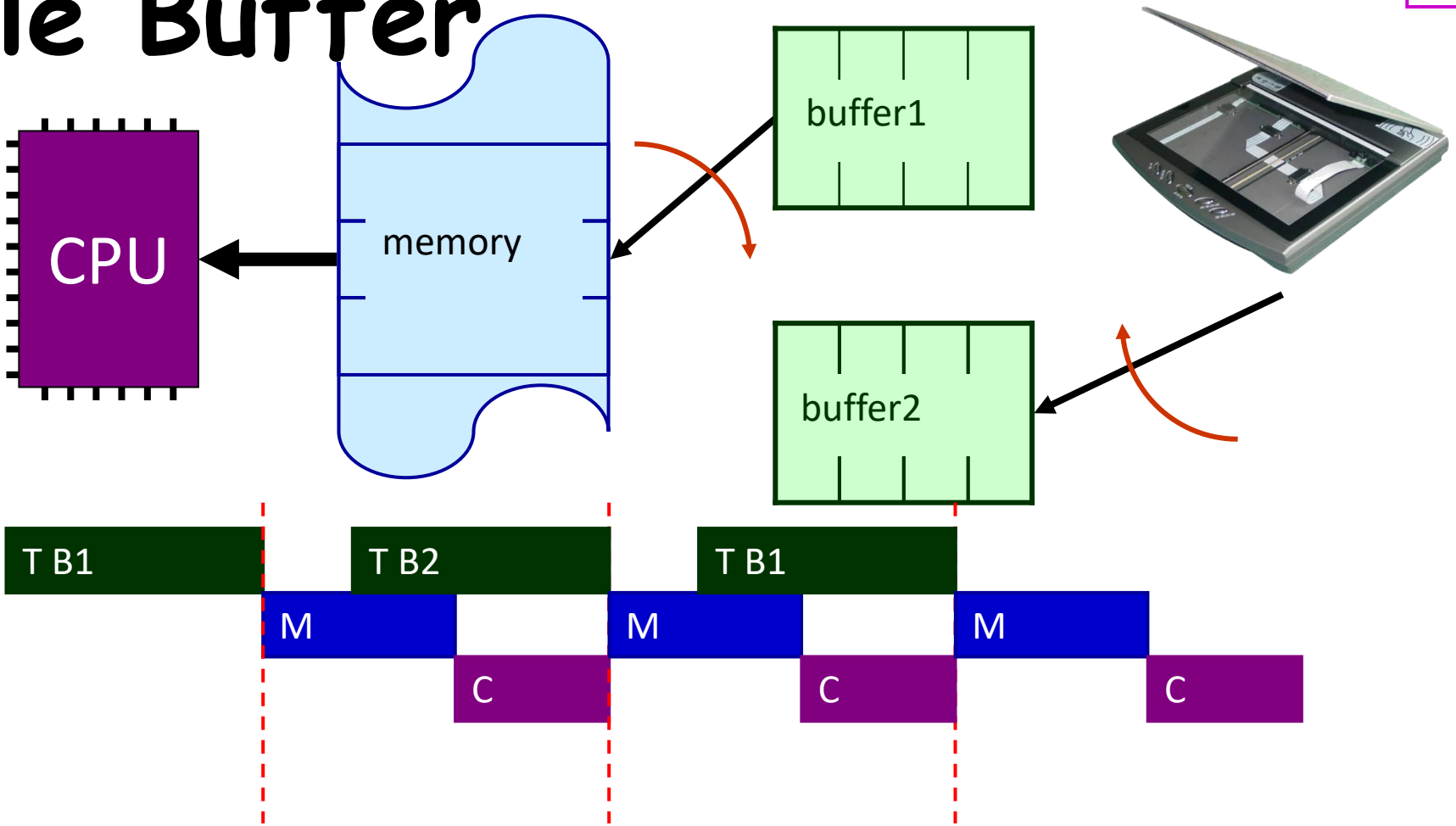

*Max(C,T)+M*



# Device Management

## Double Buffer

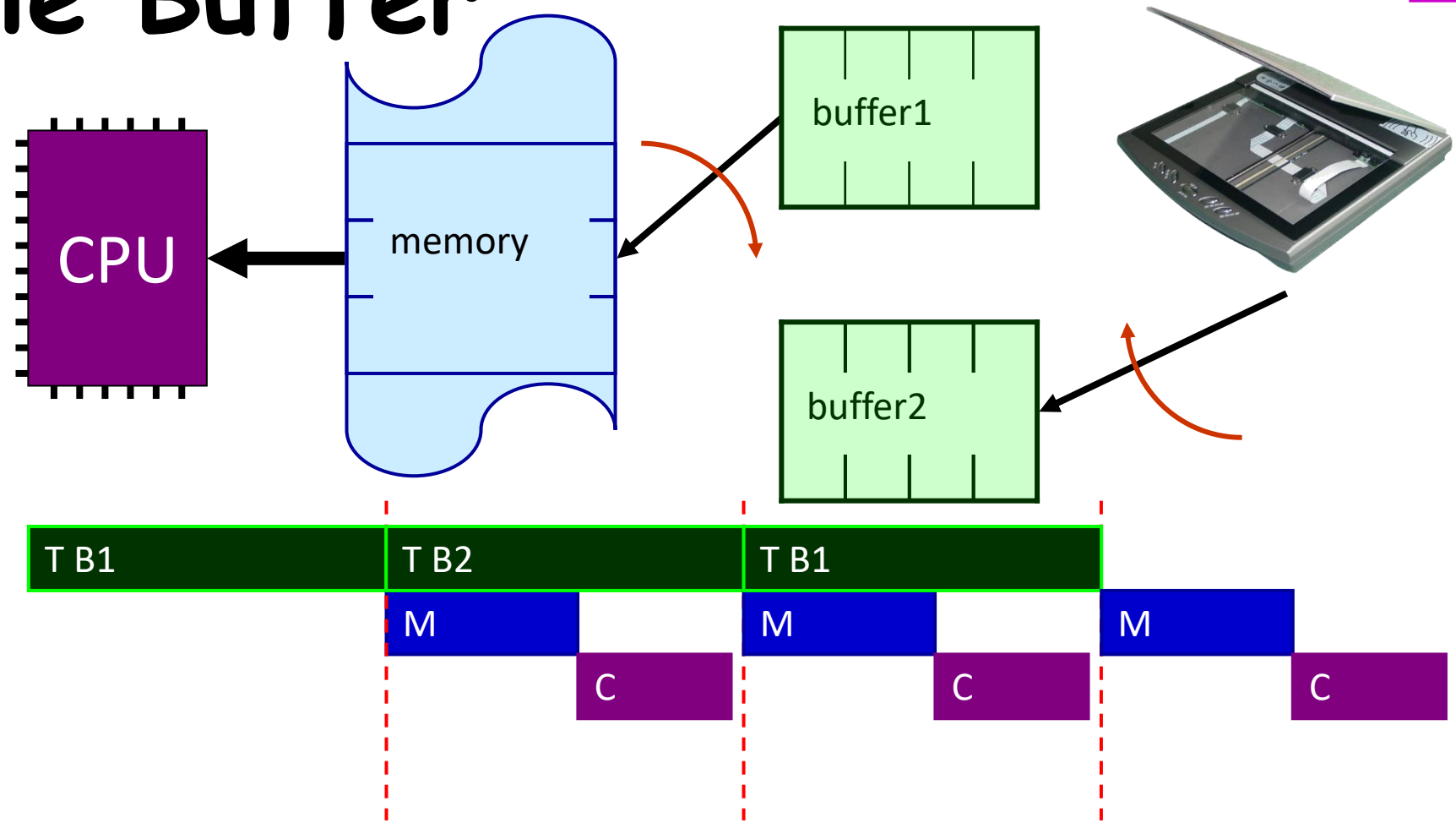
T>C



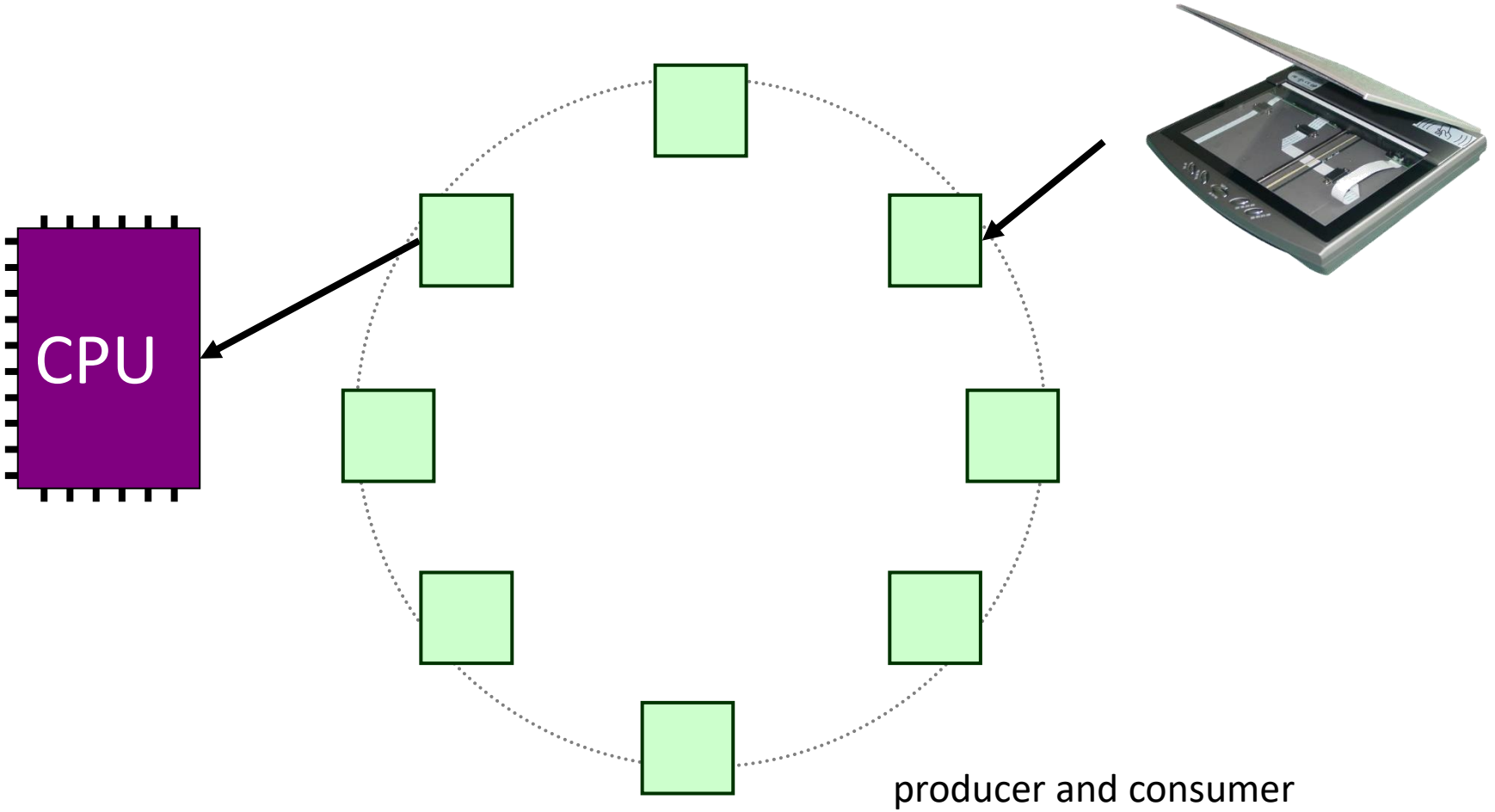
# Device Management

## Double Buffer

T>C



# Circular Buffer

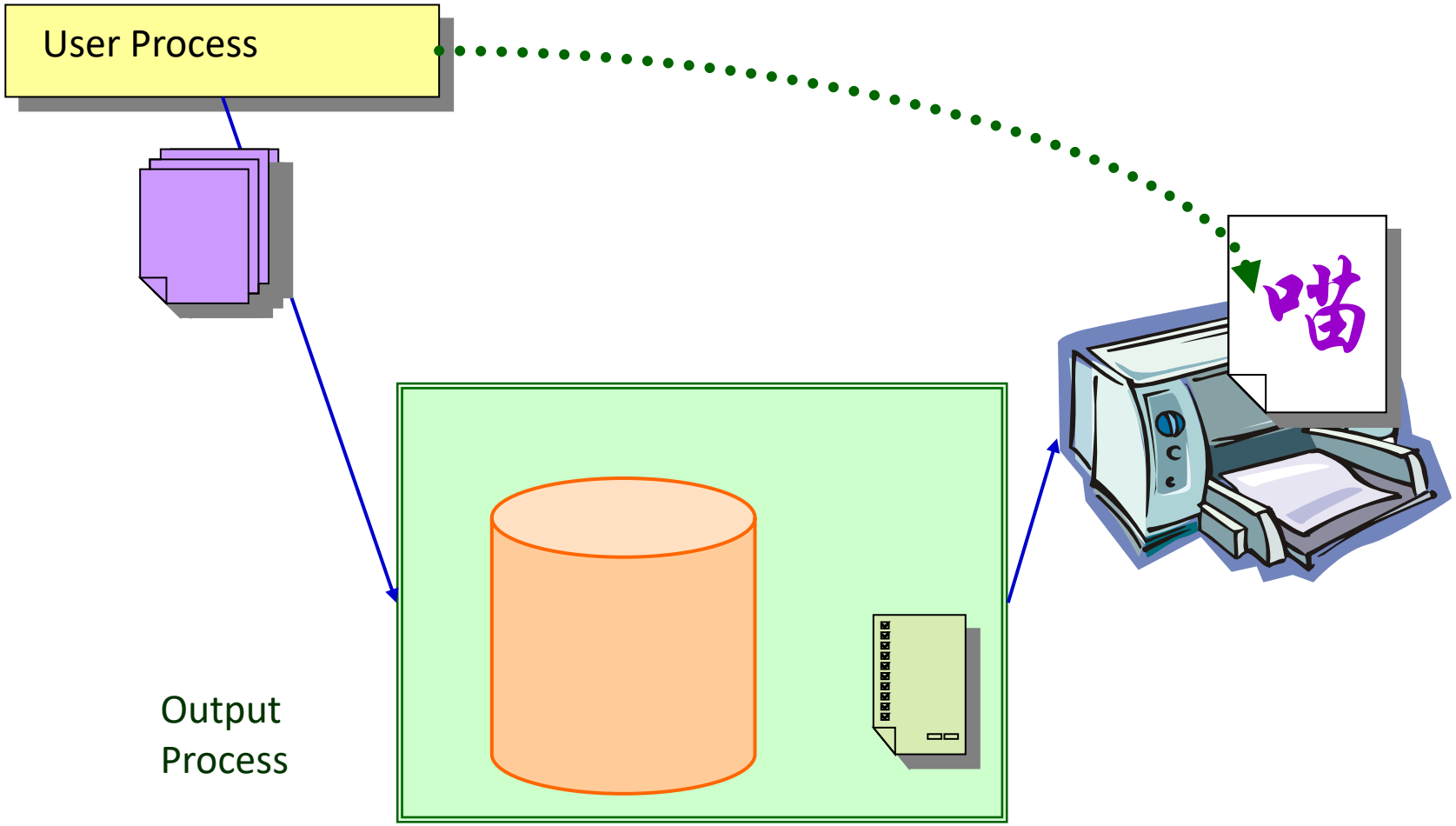


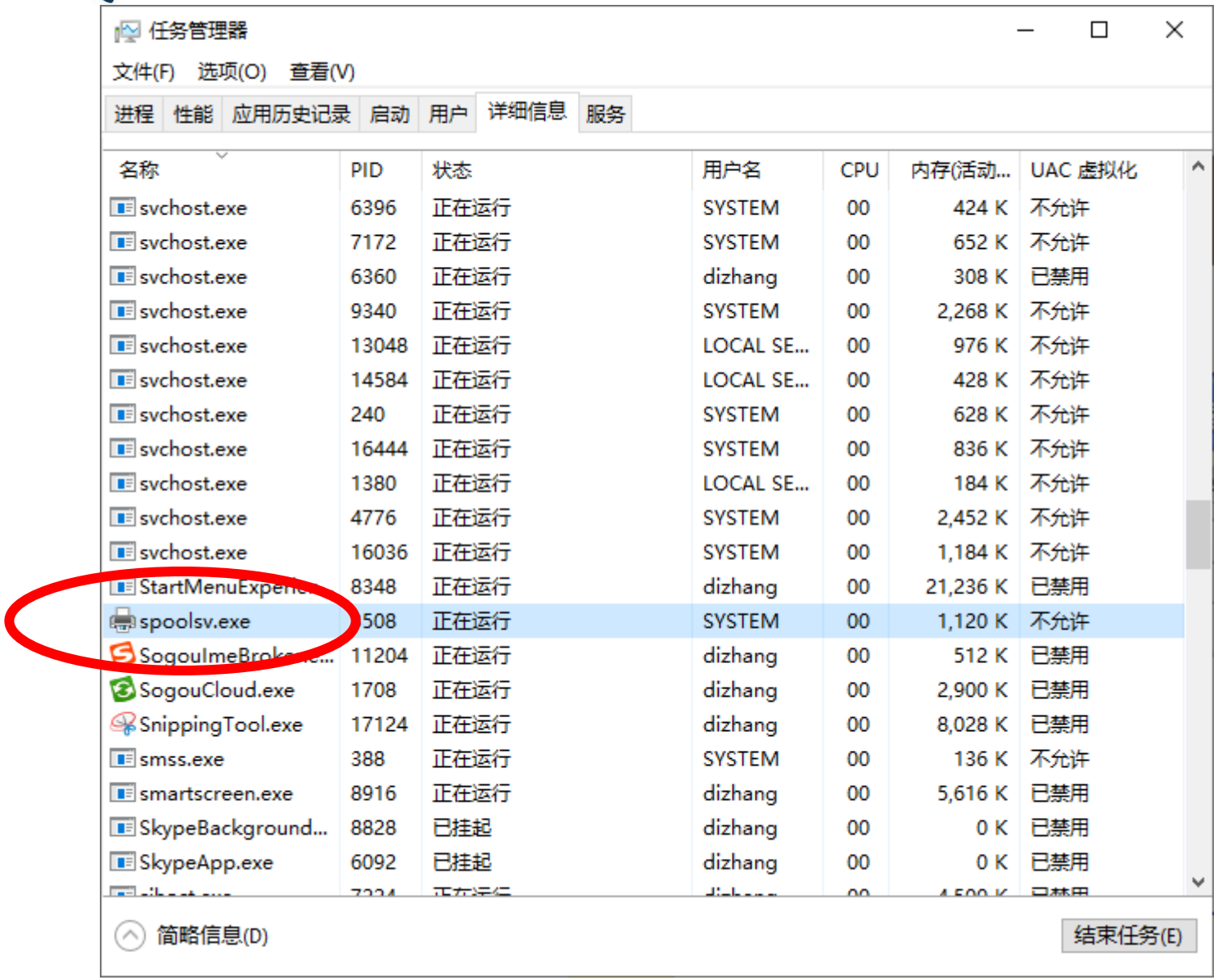

producer and consumer

# Device Management

- **SPOOLing** (Simultaneous Peripheral Operation On Line) hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

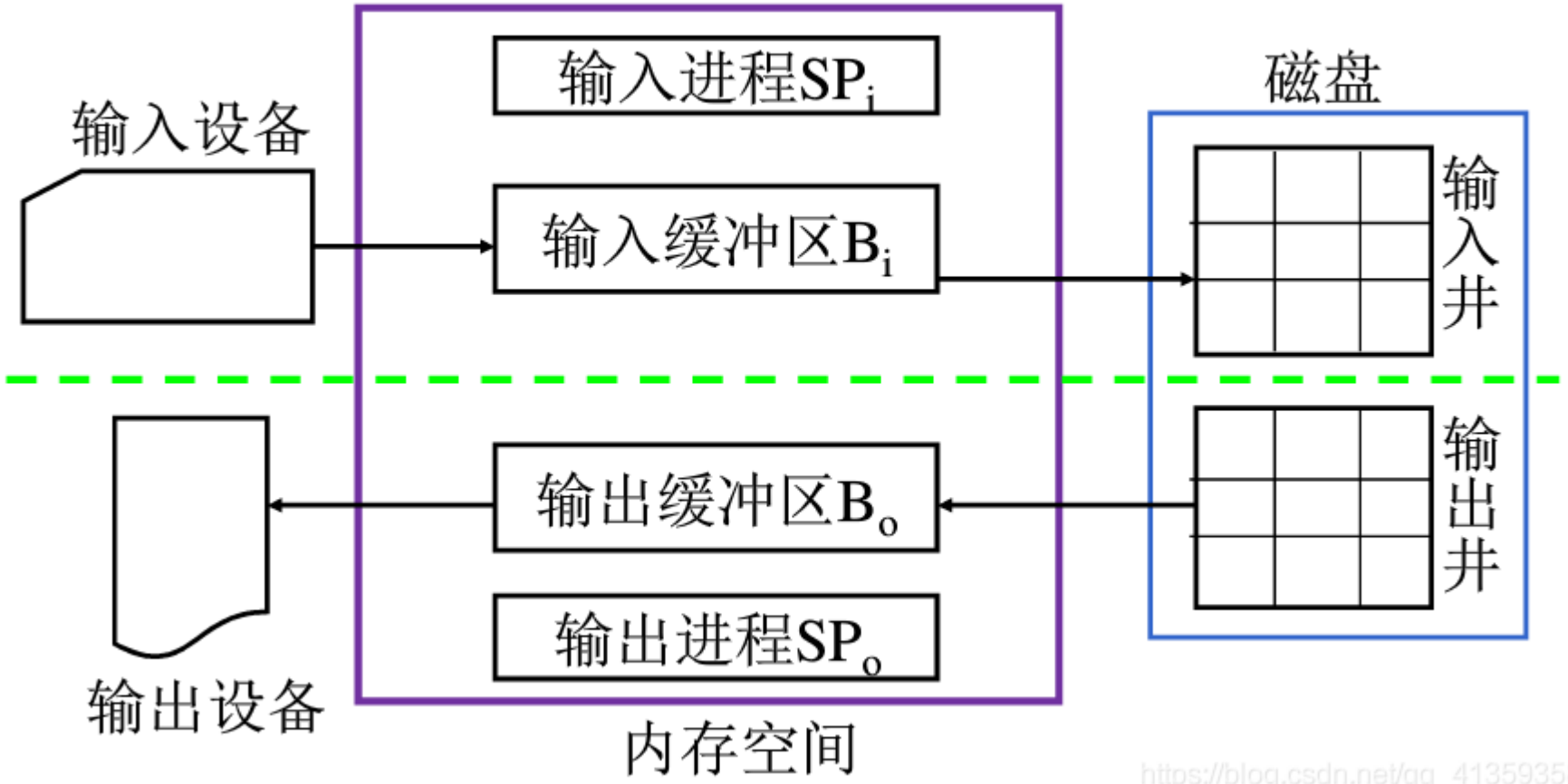
# Spooling




任务管理器
文件(F) 选项(O) 查看(V)
进程 性能 应用历史记录 启动 用户 详细信息 服务
名称 PID 状态 用户名 CPU 内存(活动... UAC 虚拟化
svchost.exe 6396 正在运行 SYSTEM 00 424 K 不允许
svchost.exe 7172 正在运行 SYSTEM 00 652 K 不允许
svchost.exe 6360 正在运行 dizhang 00 308 K 已禁用
svchost.exe 9340 正在运行 SYSTEM 00 2,268 K 不允许
svchost.exe 13048 正在运行 LOCAL SE... 00 976 K 不允许
svchost.exe 14584 正在运行 LOCAL SE... 00 428 K 不允许
svchost.exe 240 正在运行 SYSTEM 00 628 K 不允许
svchost.exe 16444 正在运行 SYSTEM 00 836 K 不允许
svchost.exe 1380 正在运行 LOCAL SE... 00 184 K 不允许
svchost.exe 4776 正在运行 SYSTEM 00 2,452 K 不允许
svchost.exe 16036 正在运行 SYSTEM 00 1,184 K 不允许
StartMenuExperi... 8348 正在运行 dizhang 00 21,236 K 已禁用
spoolsv.exe 1508 正在运行 SYSTEM 00 1,120 K 不允许
SogouImeBroker... 11204 正在运行 dizhang 00 512 K 已禁用
SogouCloud.exe 1708 正在运行 dizhang 00 2,900 K 已禁用
SnippingTool.exe 17124 正在运行 dizhang 00 8,028 K 已禁用
smss.exe 388 正在运行 SYSTEM 00 136 K 不允许
smartscreen.exe 8916 正在运行 dizhang 00 5,616 K 已禁用
SkypeBackground... 8828 已挂起 dizhang 00 0 K 已禁用
SkypeApp.exe 6092 已挂起 dizhang 00 0 K 已禁用
简略信息(D)
结束任务(E)

# Spooling

设备分为独占式设备，共享使用设备和虚拟设备（即为以spooling使用的外部设备）。独占设备申请后只有到被释放才能被其他进程申请使用，为了让独占设备能**逻辑上**像共享设备一样使用，是一种**将独占式设备改造成共享设备的技术**（逻辑上）。



# Spooling

- SPOOLING技术（Simultaneous Peripheral Operating On Line)同时联机外围操作技术，它是关于慢速字符设备如何与计算机主机进行数据交换的一种技术，通常又称假脱机技术。在多道程序环境下，利用多道程序中的一道或者两道程序来模拟脱机输入/输出中的外围控制机的功能，以达到“脱机”输入/输出的目的。利用这种技术可把独占设备转变成共享的虚拟设备，从而提高独占设备的利用率和进程的推进速度。

- SPOOLING系统 是对脱机输入/输出工作的模拟，它必须有大容量的且可随机存取的存储器的支持。其主要思想是在联机的条件下，进行两个方向的操作，在数据输入时，将数据从输入设备传送到磁盘或磁带（块设备），然后把这些块设备与主机相连；反过来，在数据输出时，将输出数据传送到磁盘或磁带上，再从磁盘或磁带传送到输出设备。这样，可以将一台独占的物理设备虚拟为并行使用的多态逻辑设备，从而使该物理设备被多个进程共享。

# Spooling

- 输入进程SPI是模拟脱机输入时的外围控制机，它将用户要求处理的数据从输入设备通过输入缓冲区再送到输入井（磁盘上开辟的一块区域），当CPU处理这些数据数据时，就直接从输入井读入内存。输出进程SPO是模拟脱机输出时的外围控制机，把用户要求输出的数据，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通过输出缓冲区（内存中一块区域）传送到输出设备上。

# Spooling

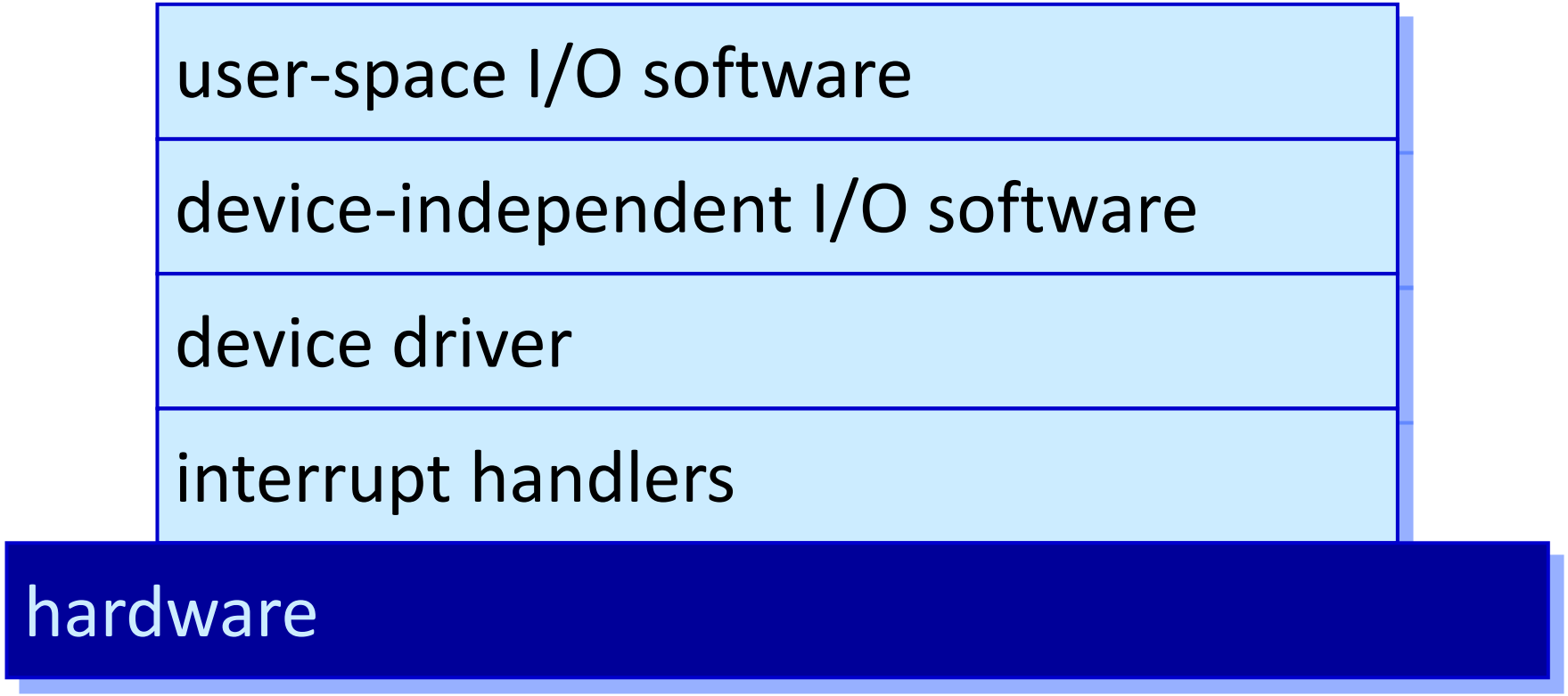
- 实例——利用打印机实现打印机共享已经被广泛用于多用户系统和计算机网络中，它实际上就是利用SPOOLING技术将独占的打印机改造为一台供做个用户共享的设备，只要有足够的外存空间和多道程序操作系统的支持即可。

1、当用户进程请求打印输出时，SPOOLING系统立即同意为该进程执行打印输出，但并不是真正地把打印机分配给该用户进程，而只是为该进程做两项工作： 一项是由输出进程SPO在输出井中为之申请一个空闲的存储空间，并将要打印的数据传送其中存放；另一项工作就是由输出进程SPO再为用户进程申请一张空白的用户请求打印表，并将用户的打印请求填入其中，然后将该表挂到打印机的请求队列上。这时，如果还有另一个进程请求打印机时，则系统仍同意为该进程执行打印输出，当然，系统所做的工作仍是以上两项内容。

2、在打印机执行实际打印时，如果打印机空闲，输出进程SPO将从请求打印队列的队首取出一张打印表，根据打印表中的要求将要打印的数据从输出井传送到内存输出缓冲区，再传送到打印机打印。打印完后，输出进程SPO将再检查请求打印队列中是否还有待打印的请求表，若有则再取出一张请求打印表，将新的但因要求继续打印。如此反复，直到请求打印队列空为止，输出进程才将自己阻塞起来，并在下次再有打印请求时被唤醒。

# At last

# Layered I/O software system



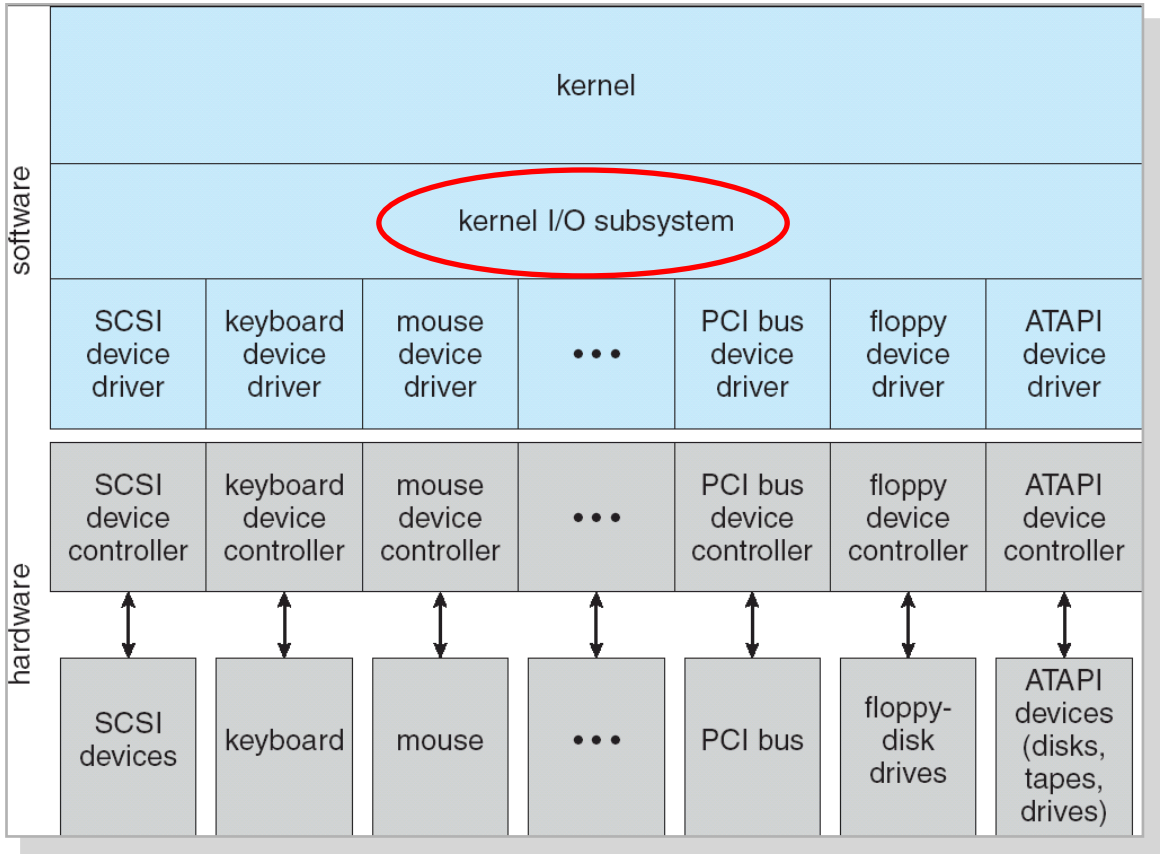
# Summary

# Summary

- Thousands of devices, varying in many dimensions
  - **Character-stream** or **block**
  - **Sequential** or **random-access**
  - **Sharable** or **dedicated**
  - **Speed** of **operation**
  - **read-write, read only,** or **write only**
- In common
  - **Port**
  - **Bus** (**daisy chain** or shared direct access)
  - **Controller**
  - I/O instructions control devices
  - Devices have addresses

# Summary

- Hides differences from kernel
  - Done by Kernel I/O Subsystem



# Three Main Problems

- Thousands of devices, each slightly different
  - **Provide Uniform Interfaces, Despite Wide Range of Different Devices**
- Devices unreliable: media failures and transmission errors
- Devices unpredictable and/or slow
  - **Devices Controlling and Management**

# Summary

- Device management
  - Scheduling
  - Buffering
  - Error handling
- Spooling

北京交通大学

# Thank you!

Q & A