Chapter 13: I/O Systems

Chapter 13: I/O Systems

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

Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

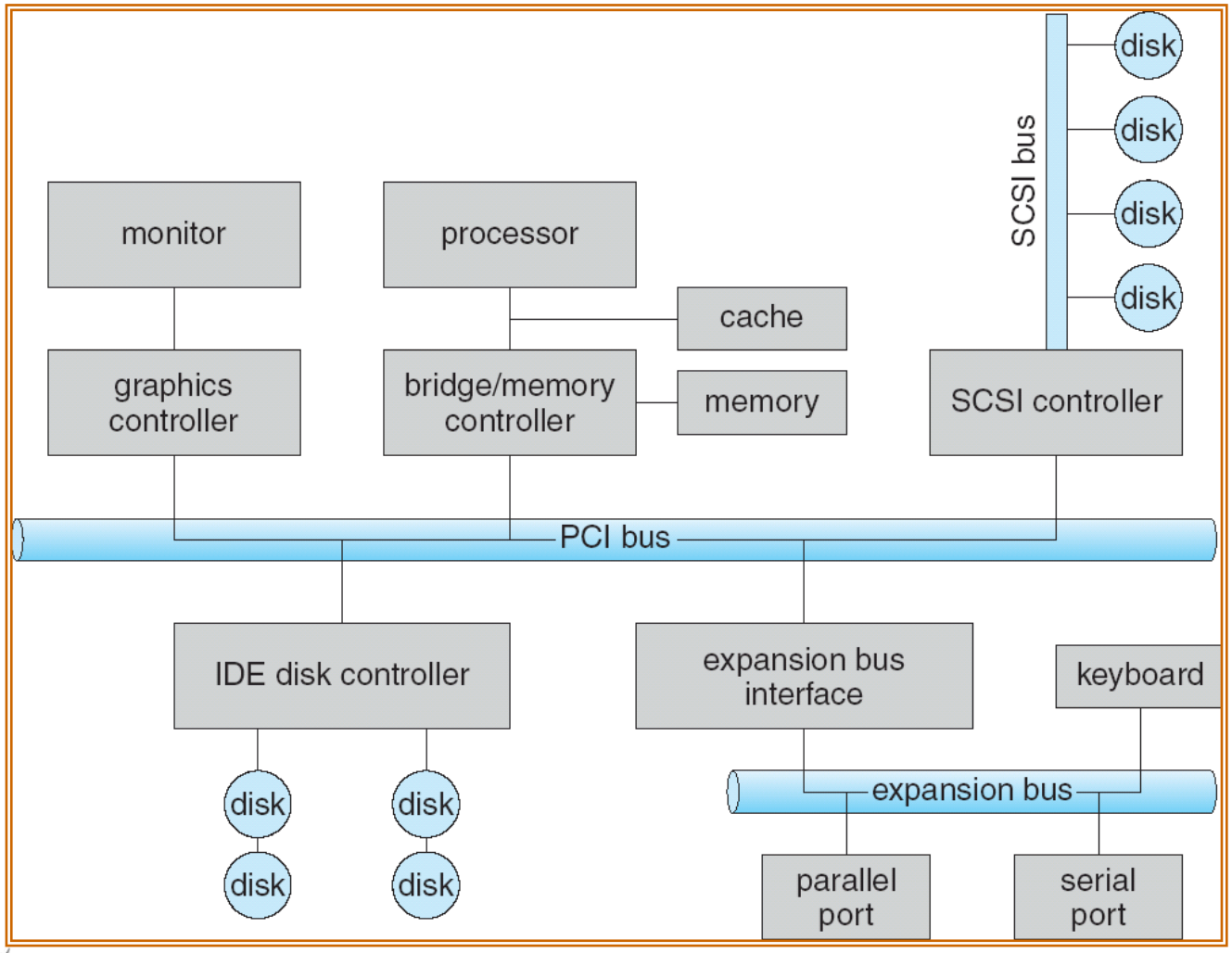
I/O Hardware

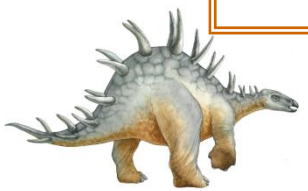
- **Incredible variety of I/O devices**
- Common concepts
  - **Port** 设备与计算机通信的连接点
  - **Bus** (**daisy chain** or shared direct access)
    一组线和一组严格定义的可以描述在线上传输信息的协议。
  - **Controller** (**host adapter**)
- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions（专有I/O通道）
  - **Memory-mapped I/O**（将I/O映射到内存里面，从而使得I/O和内存管理得到统一）

I/O Hardware

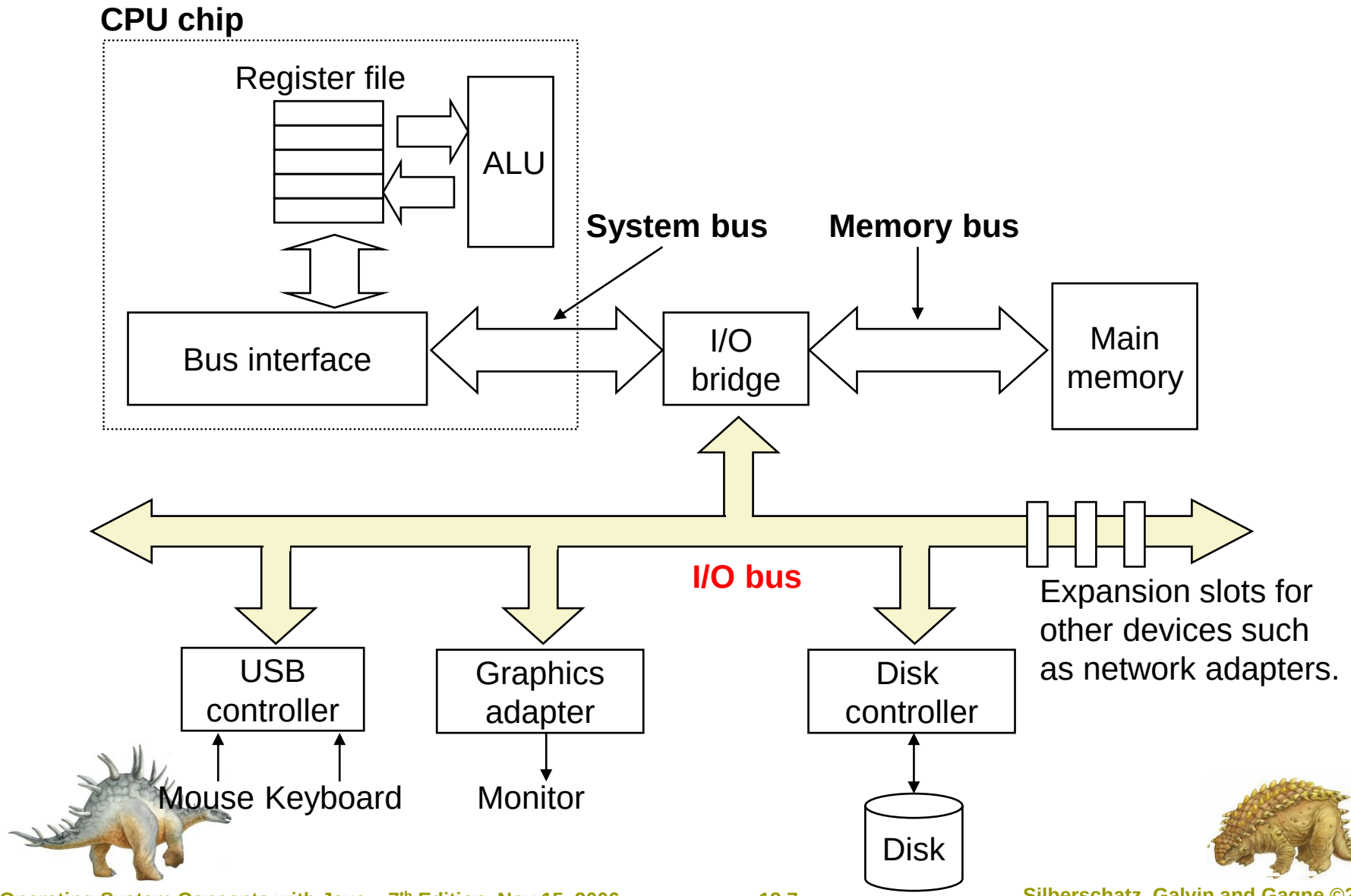
- **根据不同需求和场景，人们发明了大量专用设备**
  - 通信、存储、智能计算、安全协处理器等
- **每种设备有自己的协议、规范**
  - 如何标准化外设接口？
- **设备也可能产生错误**
  - 如何得知外设的状态并修复错误？

A Typical PC Bus Structure





I/O Bus


CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

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)

Polling

- Determines state of device
  - command-ready
  - busy
  - Error
- **Busy-wait** cycle to wait for I/O from device

Interrupts

- CPU IRL **(Interrupt-request line)** triggered by I/O device
- **Interrupt handler** receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some **nonmaskable**
- Interrupt mechanism also used for exceptions

Interrupts

中断嵌套

- **中断也能被“中断”！**
- **在处理当前中断（ISR）时：**
  - 更高优先级的中断产生；或者
  - 相同优先级的中断产生
- **那么该如何响应？**
  - 允许高优先级抢占
  - 同级中断无法抢占

如何禁止中断被抢占？

- **中断屏蔽：**
  - 屏蔽全局中断：不再响应任何外设请求
  - 屏蔽对应中断：只对对应IRQ停止响应
- **什么策略合适？**
  - 屏蔽全局中断：
    - 1. 系统关键步骤（原子性）
    - 2. 保证任务响应的实时性
  - 屏蔽对应中断：通常都是这种情况，对系统的整体影响最小

Interrupts

思考：

- **如果发端拼命发数据，收端首先收到什么数据？ 都可能**
  - 收端来不及处理数据，导致之前的数据的被覆盖了？ 轮询
  - 收端在处理中断上下文中，导致后来数据没有收到？ 中断

对策

- **静态配置法**
  - UART驱动需要在初始化时指定“波特率”
- **动态协商法**
  - TCP设计了流量控制机制，发端逐步试探出收端收包能力的上限
- **思考**
  - UDP没有流量控制，那么用户会收到什么数据包（last or first?）

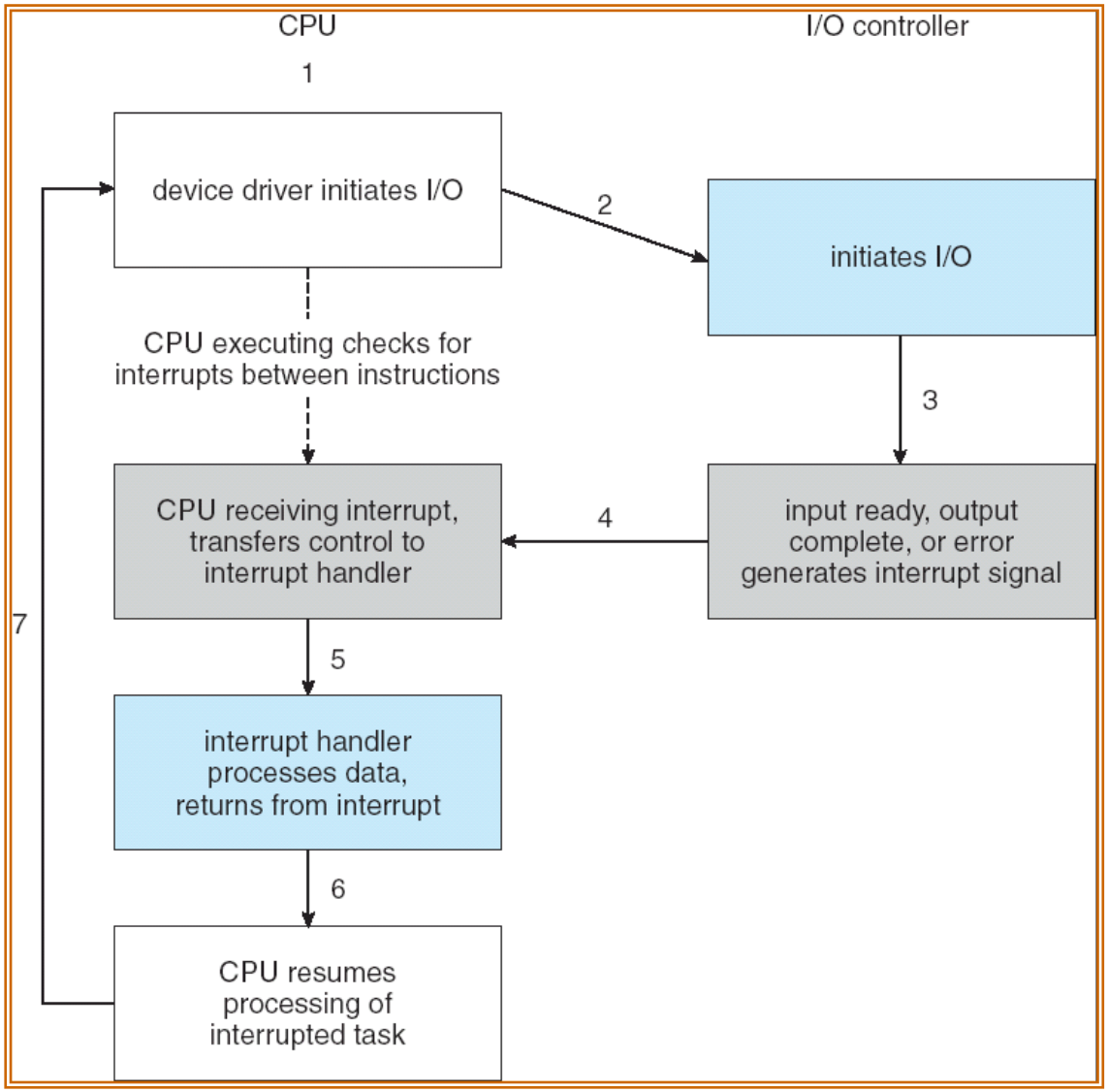
Interrupts

问题：如果有多个中断同时发生怎么办？

- **中断优先级：**
  - 当多个中断同时发生时（NMI、软中断、异常），CPU首先响应高优先级的中断
- **ARM Cortex-M 处理器的中断优先级如下所示：**

类型	优先级（值越低，优先级越高）
复位（reset）	-3
不可屏蔽中断（NMI）	-2
硬件故障（Hard Fault）	-1
系统服务调用（SVcall）	可配置
调试监控（debug monitor）	可配置
系统定时器（SysTick）	可配置
外部中断（External Interrupt）	可配置

Interrupt-Driven I/O Cycle



Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

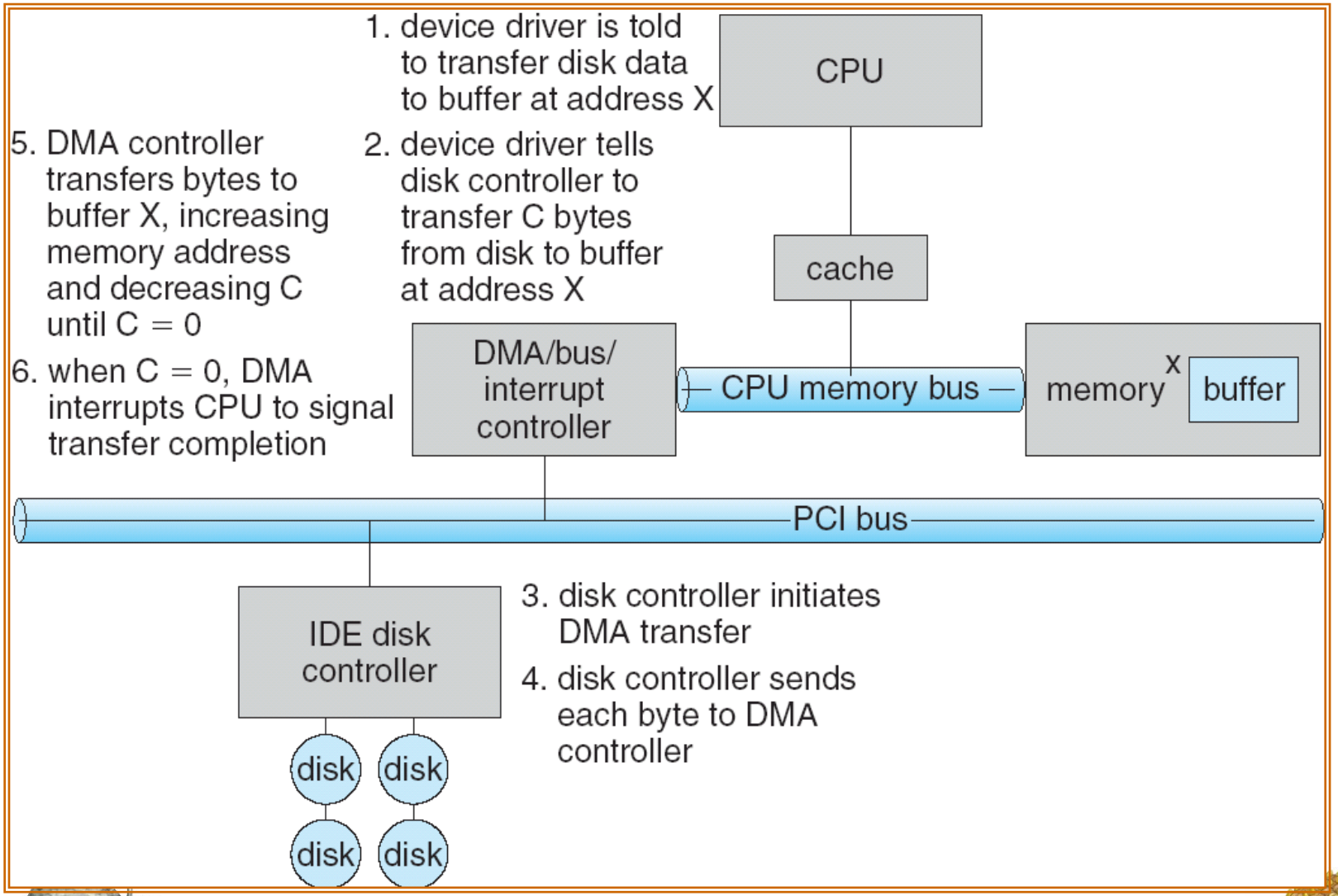
Direct Memory Access

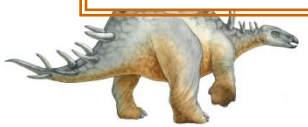
- Used to **avoid programmed I/O** for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory

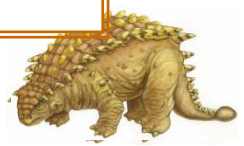
Direct Memory Access

- **可编程 I/O（Programmable I/O）**
  - 通过CPU in/out 或 load/store 指令
  - 消耗CPU时钟周期和数据量成正比
  - 适合于简单小型的设备
- **直接内存访问（DMA）**
  - 外设可直接访问总线
  - DMA与内存互相传输数据，传输不需要CPU参与
  - 适合于高吞吐量I/O

Six Step Process to Perform DMA Transfer







Application I/O Interface

- I/O system calls **encapsulate** device behaviors in **generic classes**

设备抽象（以Linux为例）

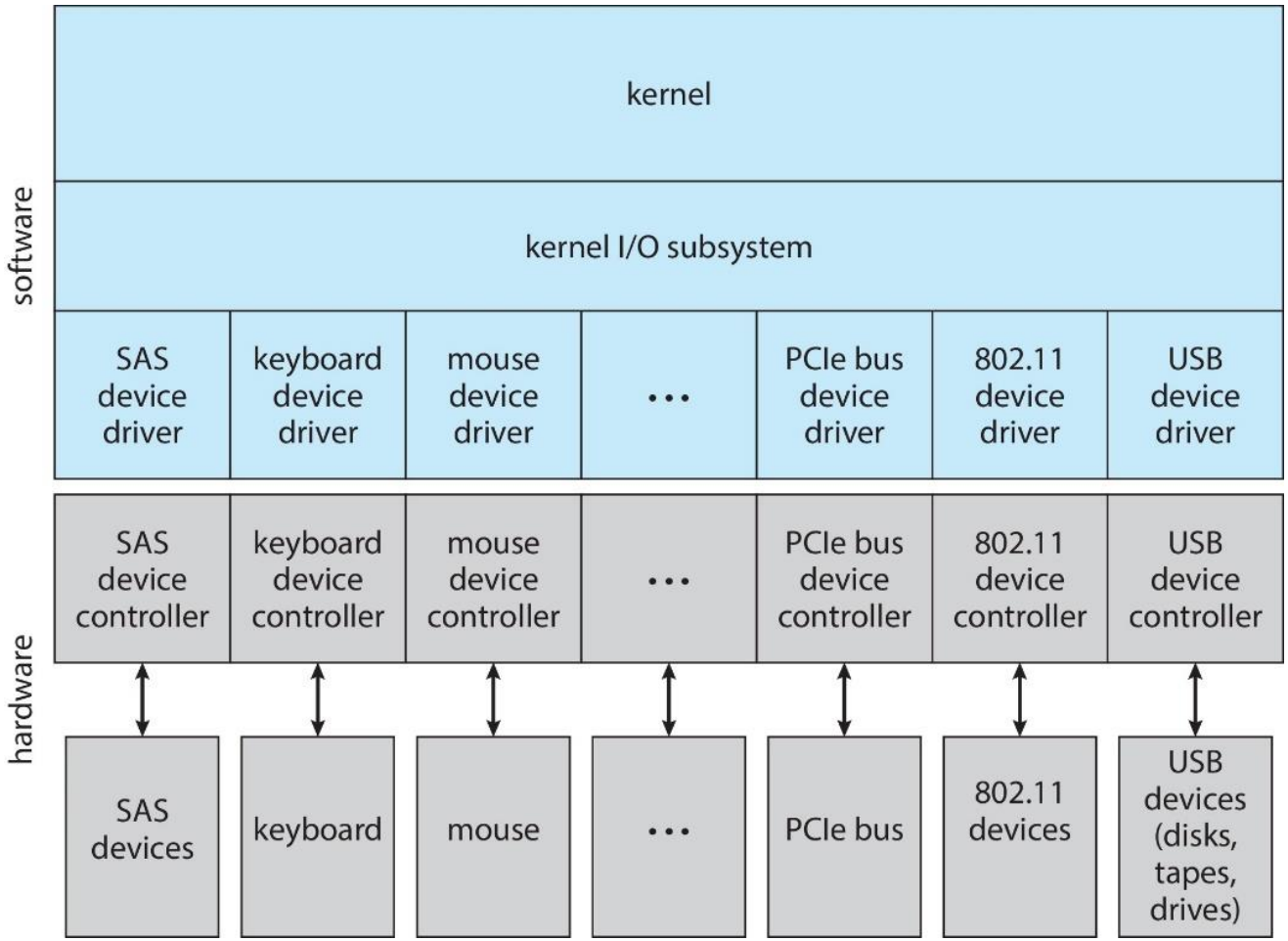
- **对设备进行分类**
  - 字符设备（char）：LED、键盘、串口等
  - 块设备（block）：闪存、硬盘等
  - 网络设备（network）：Ethernet网卡、蓝牙网卡等
- **对设备进行管理**
  - 字符抽象：文件系统（ read/write ）
  - 块抽象：文件系统（ read/write ），mmap
  - 网络抽象：socket，文件系统兼容（用read/write也可读写socket）

Application I/O Interface

操作系统都有其I/O子系统结构和设备驱动程序框架

1. 不同I/O设备中抽象出一些通用的类型，每种通用类型都可以通过一组标准化的功能和I/O接口进行访问。
2. 具体差别由内核模块（设备驱动程序）封装，设备驱动程序一方面可以定制以适合各种设备，一方面也提供一组标准接口。
3. I/O子系统独立于硬件，简化了操作系统开发工作，也有利于硬件制造商。

A Kernel I/O Structure



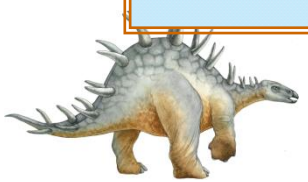
Characteristics of I/O Devices

I/O设备在许多方面都有所不同

- **数据传输模式**：字符流或块
- **访问模式**：顺序存取或随机存取
- **传输调度**：同步或异步（或两者兼有）
- **共享**：可共享的或专用的
- **运行速度**：延迟、寻道时间、传输速率、延迟等。
- **I/O方向**：读写、只读或只写。

Characteristics of I/O Devices

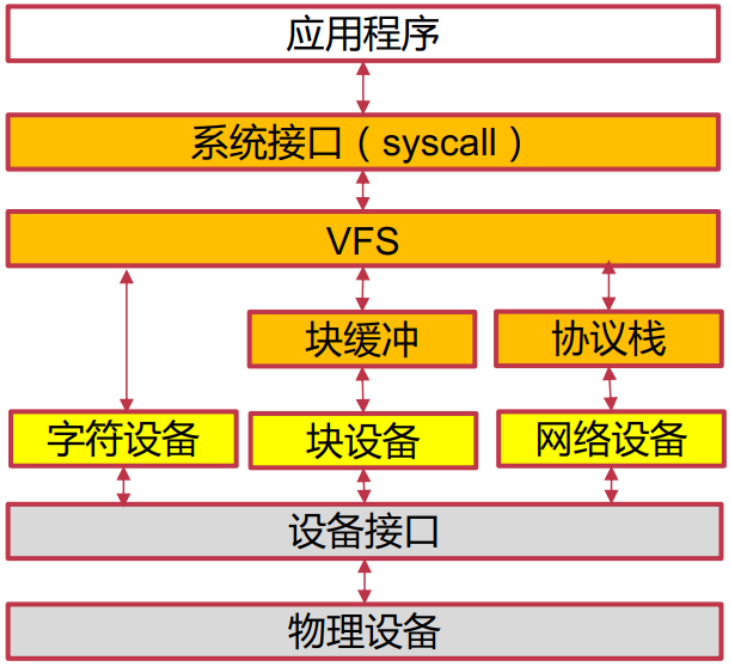
aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction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



Application I/O Interface

案例：Linux常见设备分类

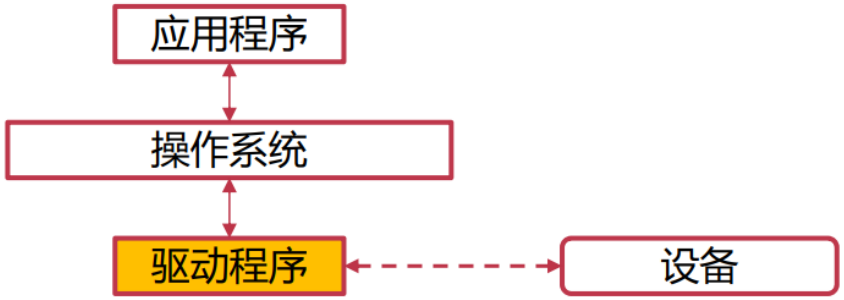
- 字符设备
- 块设备
- 网络设备



Device Driver

设备的代码——驱动

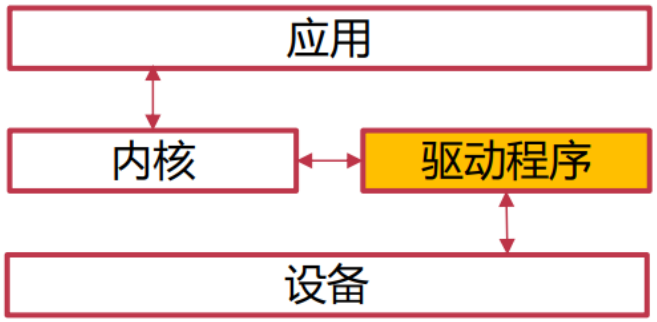
- **驱动**
  - 使操作系统和设备间能相互通信的特殊程序
- **例子：操作系统——CPU的“驱动”**



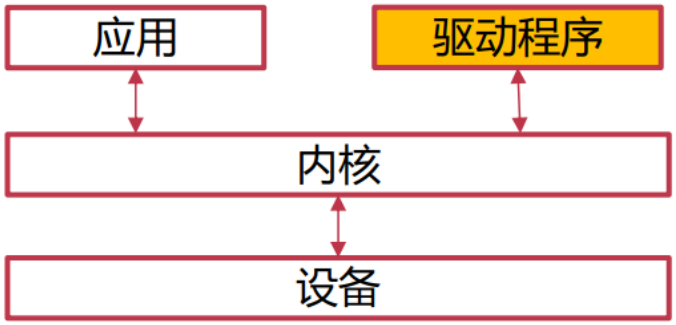
Device Driver

宏内核vs微内核的驱动

- **宏内核**
  - 驱动在内核态
  - 优势：性能更好
  - 劣势：容错性差



- **微内核**
  - 驱动在用户态
  - 优势：可靠性好
  - 劣势：性能开销（IPC）



Device Driver

驱动程序

- **系统接口（ioctl）：**
  - 让用户空间的应用通过驱动间接和设备进行交互
- **驱动模型：**
  - 让驱动程序可以在系统统一安排下和设备交互

Block and Character Devices

- **块设备：磁盘驱动器，…**
  - 数据存储在固定大小的块中，每个块都有一个地址；
  - 命令包括read（）、write（）、seek（）。
  - *原始I/O、直接I/O或文件系统访问*
  - 内存映射文件访问
    - 映射到虚拟内存和集群的文件，通过请求分页。
  - DMA
- **字符设备：键盘、鼠标、串行端口。**
  - 由一个个字符组成的流；
  - 命令包括get（）、put（）
  - 顶部分层的库允许行编辑。

Block and Character Devices

块设备

- **例子：**
  - 磁盘、U盘、闪存等（以存储设备为主）
- **访问模式：**
  - **随机访问**，以块粒度进行读写
  - 在驱动程序之上增加一层缓冲，避免和慢设备频繁交互
- **通常使用内存抽象：**
  - 内存映射文件(Memory-Mapped File)：直接访问数据
  - 同样可以使用文件抽象，但内存抽象更受欢迎（灵活性更好）

Block and Character Devices

字符设备

- **例子：**
  - 键盘、鼠标、串口、打印机等
- **访问模式：**
  - **顺序访问**，每次读取一个字符
  - 调用驱动程序和设备直接交互
- **通常使用文件抽象：**
  - open(), read(), write(), close()

Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows NT/9*x*/2000 include socket interface
  - Separates network protocol from network operation
  - Includes select functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Network Devices

- **例子：**
  - 以太网、WiFi、蓝牙等（以通信设备为主）
- **访问模式：**
  - 面向**格式化报文**的收发
  - 在驱动层之上维护多种协议，支持不同策略
- **通常使用套接字抽象：**
  - socket(), send(), recv(), close(), etc.

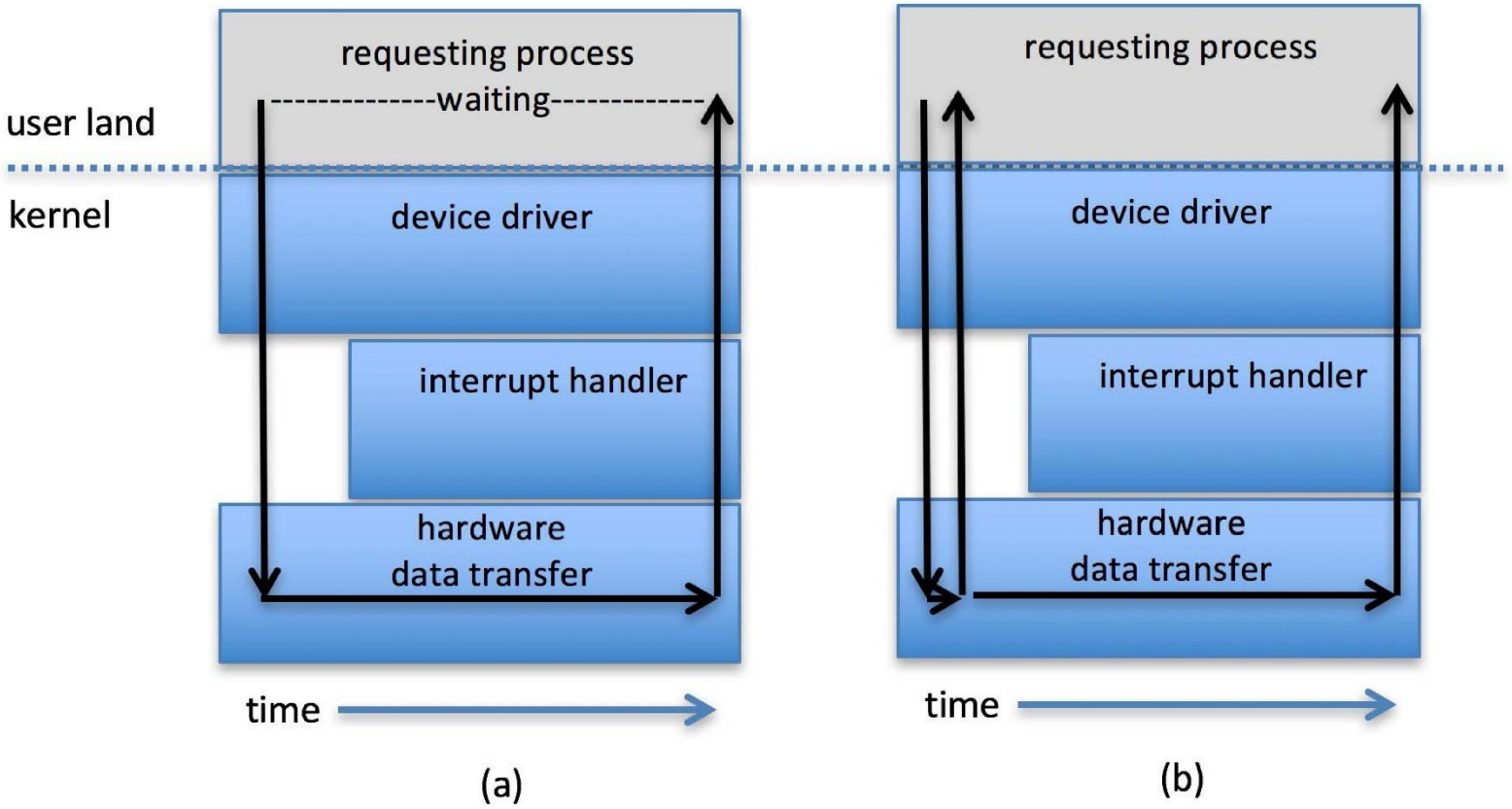
Clocks and Timers

- Provide current time, elapsed time, timer
- **Programmable interval timer** used for timings, periodic interrupts
- ioctl (on UNIX) covers odd aspects of I/O such as clocks and timers

Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Two I/O Methods



Synchronous

Asynchronous

Kernel I/O Subsystem

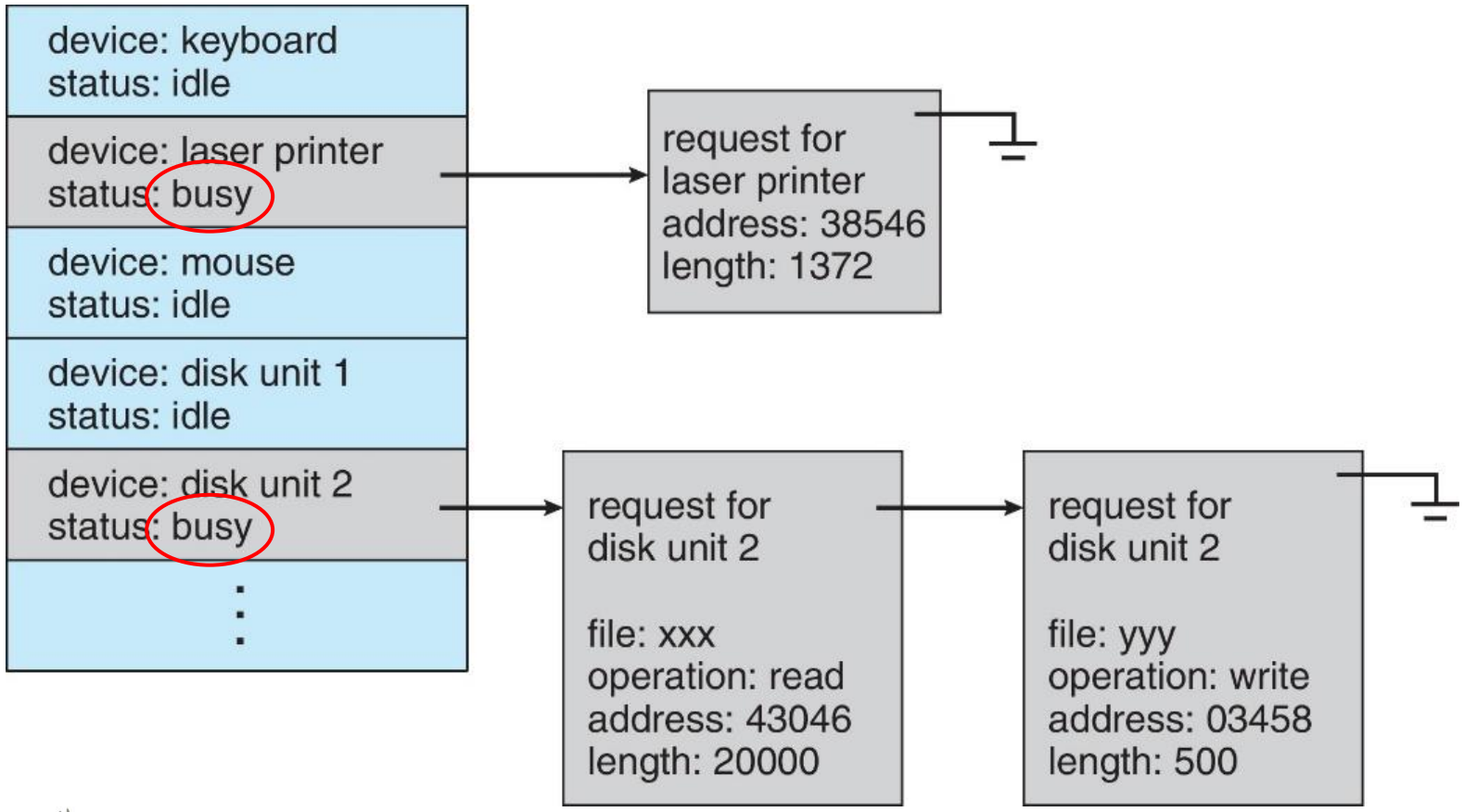
- **提供基于硬件和设备驱动程序的基本服务。**
  - 调度
  - 缓冲
  - 缓存
  - 假脱机
  - 设备预留
  - 错误处理。
- **负责保护自身免受错误进程和恶意用户的攻击。**

Kernel I/O Subsystem

- **I/O调度：**重新排列每个设备的请求等待队列的顺序。
  - 提高系统的整体性能
  - 在进程之间公平地共享设备访问
  - 减少I/O完成的平均等待时间。
- **设备状态表**
  - 包含每个I/O设备的条目，指示设备类型、地址和状态（不工作、空闲、忙碌）
  - 若设备忙于某一个请求，则请求的类型和其他参数将会保留在该设备的相应表条目中。
  - 内核管理该表，支持异步I/O，同时跟踪所有I/O请求，调度I/O操作。

Device-status Table

设备状态表

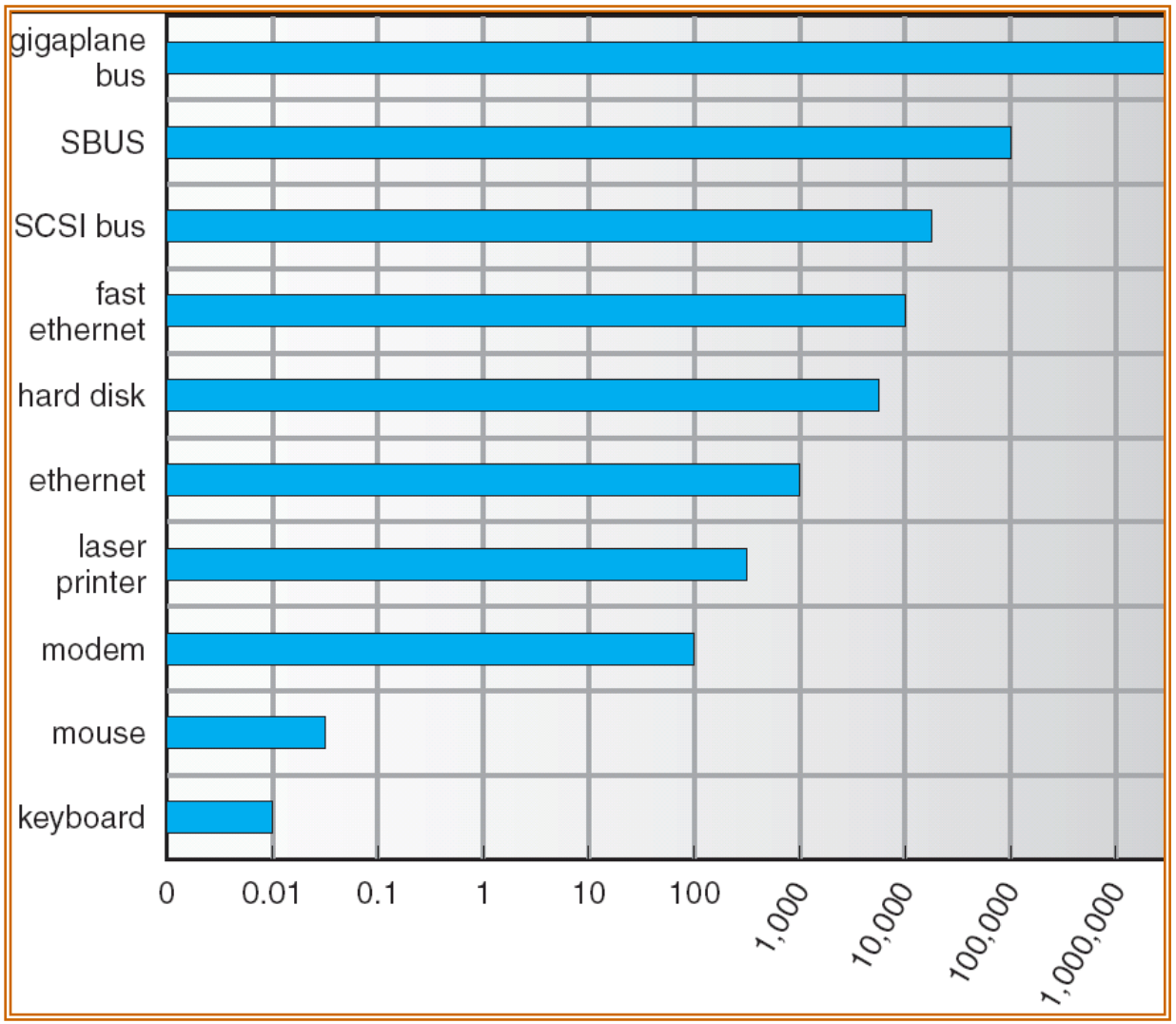


Kernel I/O Subsystem

缓冲（Buffering）

- 缓冲-在设备之间传输时，将数据存储在内存中。
  - 处理设备速度不匹配问题
  - 处理设备传输大小不匹配
  - 维护“复制语义”
- *复制语义—操作系统保证写入磁盘的数据版本是应用程序系统调用时的版本，与应用程序缓冲区中的任何后续更改无关。*
  - 如write（）系统调用，操作系统在将控制权返回应用程序之前，将应用程序数据复制到内核缓冲区。磁盘写入是从内核缓冲区执行的，因此对应用程序缓冲区的后续更改不会产生任何影响。
- *双缓冲–数据的两个副本。*
  - 内核和用户
  - 大小不一
  - 已满/正在处理中而未满/正在使用
  - 在某些情况下，写时复制可用于提高效率。

Sun Enterprise 6000 Device-Transfer Rates



Kernel I/O Subsystem

缓存（Caching）

- 缓存是一个快速内存区域，用于保存数据副本。
  - 对缓存副本的访问比对原始副本的访问更高效。
- 缓冲区和缓存是不同的。
  - 缓冲区可以保存数据项的唯一现有副本，而根据定义，缓存可以保存位于其他位置的项的更快存储上的副本。
- *缓存和缓冲是不同的功能，但有时一个内存区域可以用于这两个目的。*
  - 当内核接收到文件I/O请求时，内核首先访问缓冲区缓存，以查看文件的该区域是否已在主内存中可用。如果是，则可以避免或延迟物理磁盘I/O。
  - 磁盘写入在缓冲区缓存中累积几秒钟，以便收集大量传输以实现高效的写入计划。

Kernel I/O Subsystem

假脱机和设备保留

- SPOOL是一个缓冲区，用于保存无法接受交错数据流的设备（如打印机）的输出。
  - 某些设备（如磁带机和打印机）无法有效地多路传输多个并发应用程序的I/O请求。
- *假脱机是操作系统协调此类并发问题的一种方法。*
  - 每个应用程序输出都被假脱机到一个单独的辅助存储文件中。
  - 当应用程序完成打印时，假脱机系统将相应的假脱机文件排入队列，以便输出到打印机。
  - 操作系统提供了一个控制界面，允许用户和系统管理员显示和维护假脱机队列。
- *设备预约(设备预订) 是另一种通过提供显式协作设施来处理并发设备访问的方法。*
  - 应用程序应该注意死锁。

Kernel I/O Subsystem

- **Caching** - fast memory holding copy of data
  - Always just a copy （buffer）
  - Key to performance
- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing
- **Device reservation** - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

Error Handling

- **设备和I/O传输可能以多种方式失败。**
  - 暂时的(瞬时) 原因，如当网络过载时。操作系统通常可以通过重试请求来有效补偿瞬时故障。
  - 永久性的原因，如磁盘控制器出现故障。如果一个重要组件发生永久性故障，操作系统不太可能恢复。
- **通常，I/O系统调用将返回有关调用状态的信息，表示调用成功或失败。**
  - 例如，Linux中的errno。
- **系统错误日志保存问题报告。**

Error Handling

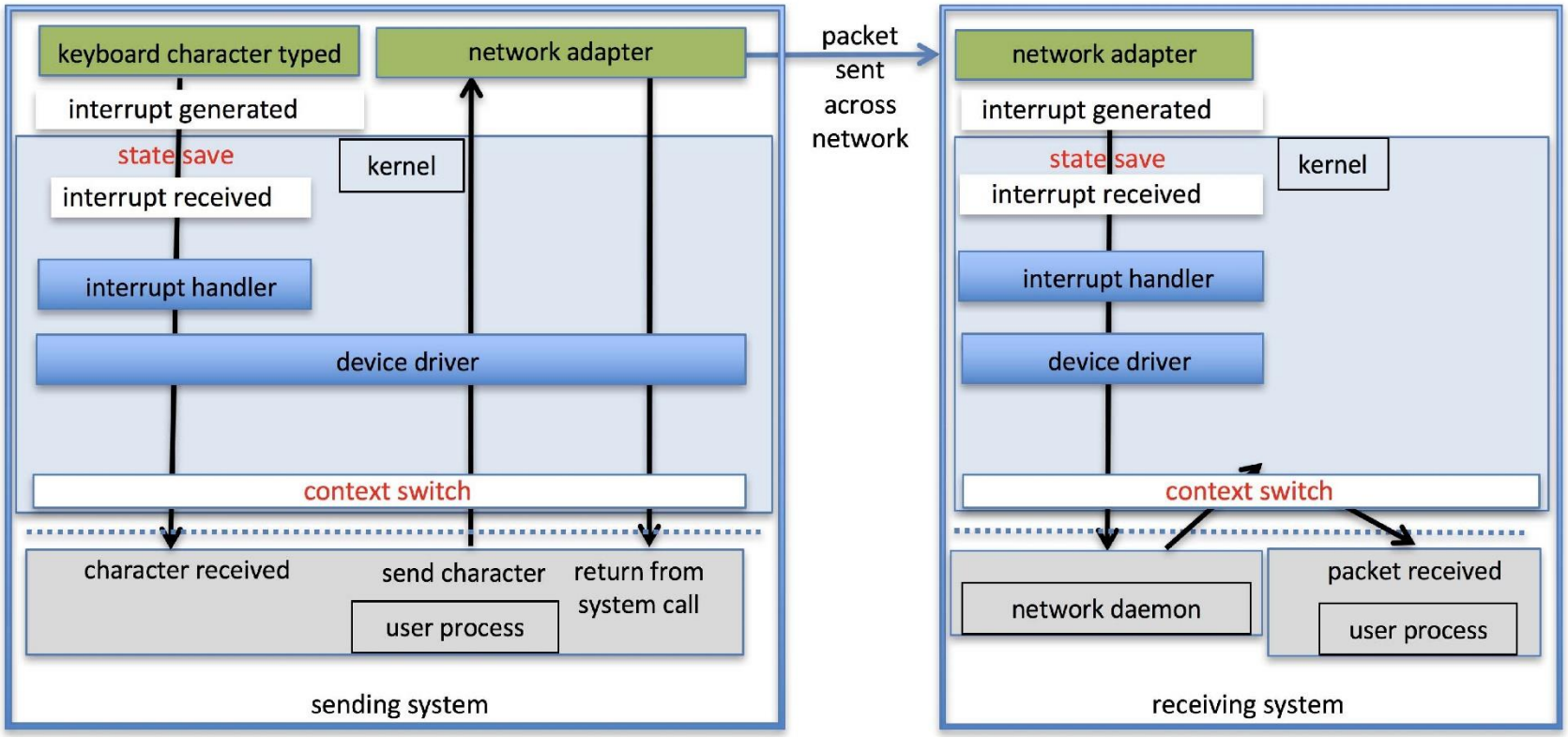
- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

Improving Performance

- **I/O是影响系统性能的主要因素。**
  - I/O对CPU提出了很高的要求，以执行设备驱动程序代码，并在进程被阻塞和解除阻塞时公平高效地调度进程。由此产生的上下文切换会给CPU及其硬件缓存带来压力。
  - I/O还暴露了内核中中断处理机制的低效性。中断处理是一项相对昂贵的任务。每个中断都会导致系统执行状态更改、执行中断处理程序，然后恢复状态。
  - 在控制器和物理内存之间的数据复制期间，以及在内核缓冲区和应用程序数据空间之间的复制期间，I/O会加载内存总线。如何快速处理所有这些需求是计算机架构师最关心的问题之一。
- **网络流量也会导致高上下文切换率。考虑计算机间的通信。**

Improving Performance

计算机之间的通信



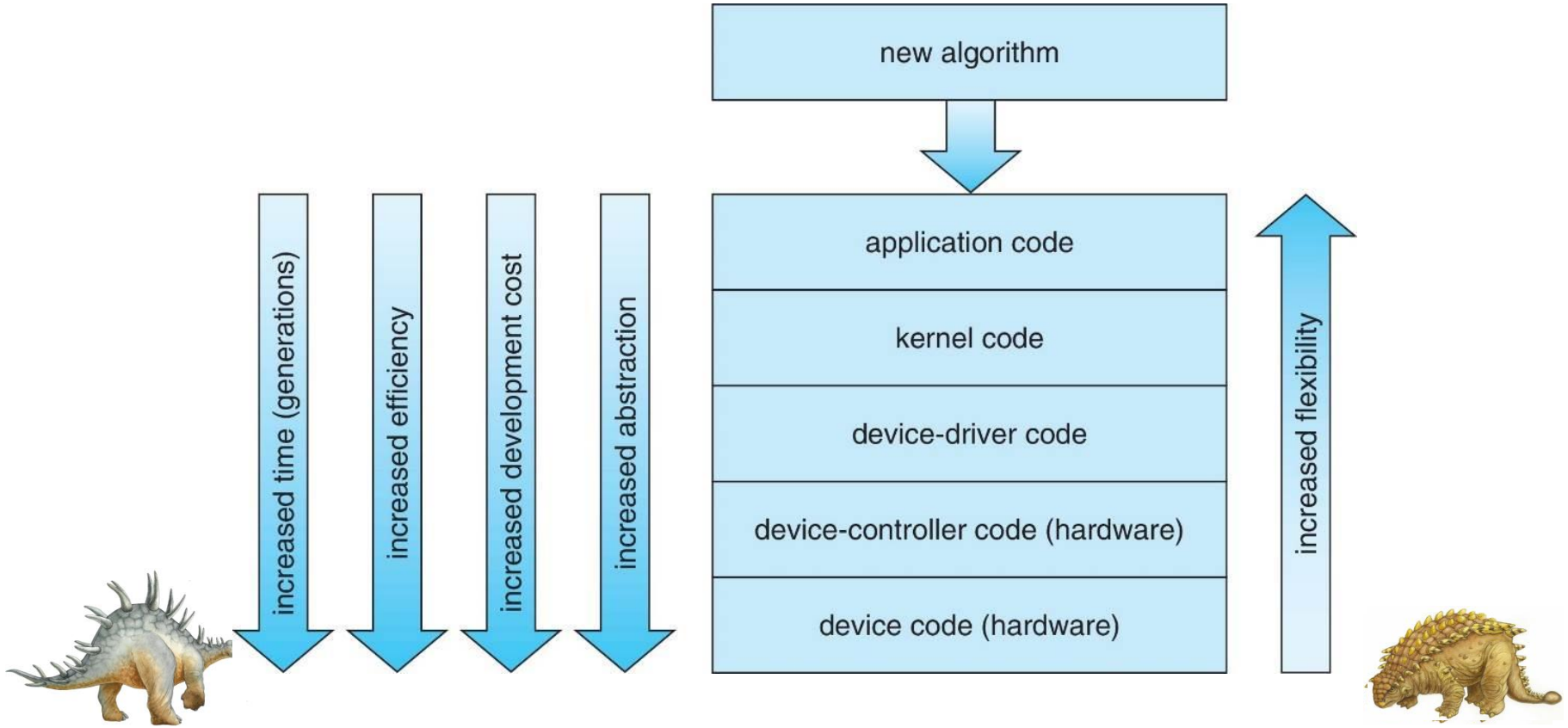
Improving Performance

- **一些系统为终端I/O使用单独的前端处理器，以减少在CPU上的中断负担**
- **可以采用几种方法提高I/O的效率**
  1. 减少上下文切换
  2. 减少在设备与应用程序间传输数据时在内存中复制的次数
  3. 通过使用DMA、通道为主CPU承担简单的数据拷贝任务，提高并发性。
  4. 将处理原语移动到硬件中，以允许其在设备控制器中的操作与CPU和总线操作并发。
  5. 平衡CPU、内存、总线和I/O性能，因为任一个部分的过载都会导致其他区域的空闲。

Improving Performance

设备功能的发展

I/O功能应该在**设备硬件**、**设备驱动程序**或**应用程序软件**中的何处实现?



Improving Performance

1. **在应用程序级别实现**
   - 应用程序代码灵活，应用程序错误不太可能导致系统崩溃。
   - 设备驱动程序不需要在每次代码更改后重新启动或重新加载。
   - 可能效率低下。
     - 上下文切换开销
     - 应用程序无法利用内部内核数据结构和内核功能
2. **在内核级别中实现**
   - 可以提高性能，但开发工作更具挑战性。
3. **在设备或控制器中的专用硬件实现**
   - 可以获得最高性能。
   - 随着开发时间的延长和灵活性的降低，该过程既困难又昂贵

Improving Performance

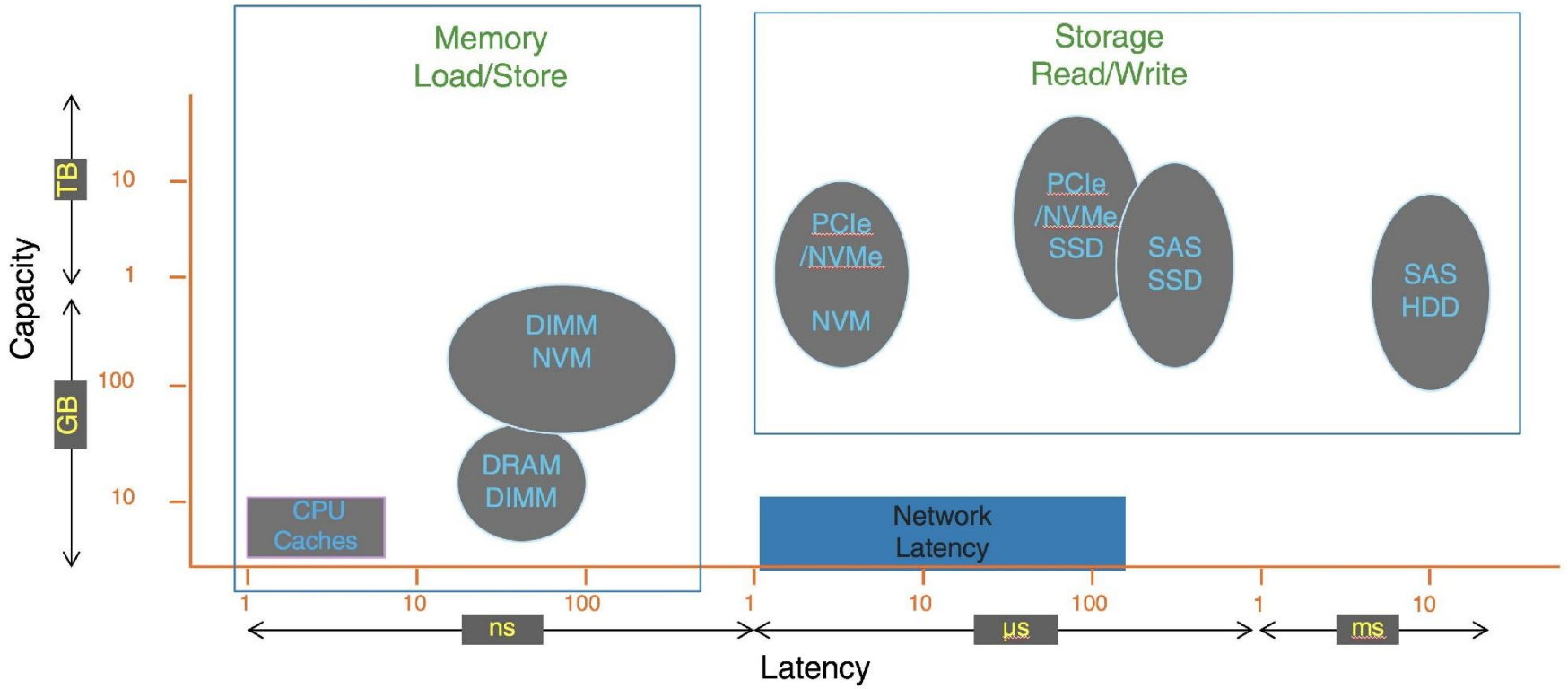
存储器的I/O性能

- I/O设备的速度一直在提高。
  - 非易失性存储设备越来越受欢迎，可用设备种类也越来越多。
  - NVM设备的速度从高到高不等，下一代设备的速度接近DRAM。
- 这些发展增加了I/O子系统以及操作系统算法利用现有读/写速度的压力。

Improving Performance

存储的I/O性能

- CPU缓存/SRAM、DIMM（双列直插存储器型号）DRAM、NVDIMM、PCIe/NVMe NVM、PCIe/NVMe SSD、SAS（串行连接SCSI）SSD、SAS HDD、SATA（串行高级技术连接）HDD…



End of Chapter 13

I/O 管理

设备管理：指操作系统对除了**CPU**和**内存**以外的所有**输入/输出设备**的管理，诸如设备控制器、通道、中断控制器等等。

为了提高计算机系统的整体效率，除了需要对CPU合理调度、对内存合理使用之外，对系统中的设备也要实施行之有效的管理，这样才能真正发挥计算机系统的整体效率。

I/O 外设分类

1、人可读

- 计算机用户间的交互；
- 打印机；
- 视频终端；
- 键盘、鼠标；

3、通信

- 远程设备间通信；
- 数字线路驱动器；
- 调制解调器；

2、机器可读

- 电子设备间通讯；
- 磁盘、磁带设备；
- 传感器；
- 控制器；
- 传动器；

I/O 功能的发展

1. **处理器直接控制外部设备；**

   由 CPU 负责在机器内存与设备控制器数据寄存器之间进行数据传送

2. **非中断可编程I/O**

3. **中断驱动I/O**

   为了减少设备驱动程序不断地询问控制器状态寄存器的开销当I/O 操作结束后，由设备控制器主动通知设备驱动程序

4. **DMA**

5. **通道**

6. **I/O处理器**

中断方式传送

为了减少CPU测试等待时间和提高CPU与外设的并行工作能力，引入了中断控制传输方式。

- 进程CPU发START命令启动外设准备数据；
- 进程放弃CPU，进程调度选择其它进程占有CPU;
- 外设准备好数据，通过**中断控制器**向CPU发中断请求信号，CPU接到该信号且**在允许中断**的情况下，**响应**该中断请求，转**中断处理程序**，在中断处理程序中完成数据输入。

DMA方式

内存和**外设**之间开辟直接的数据交换通路，由**DMA控制器**完成数据交换。

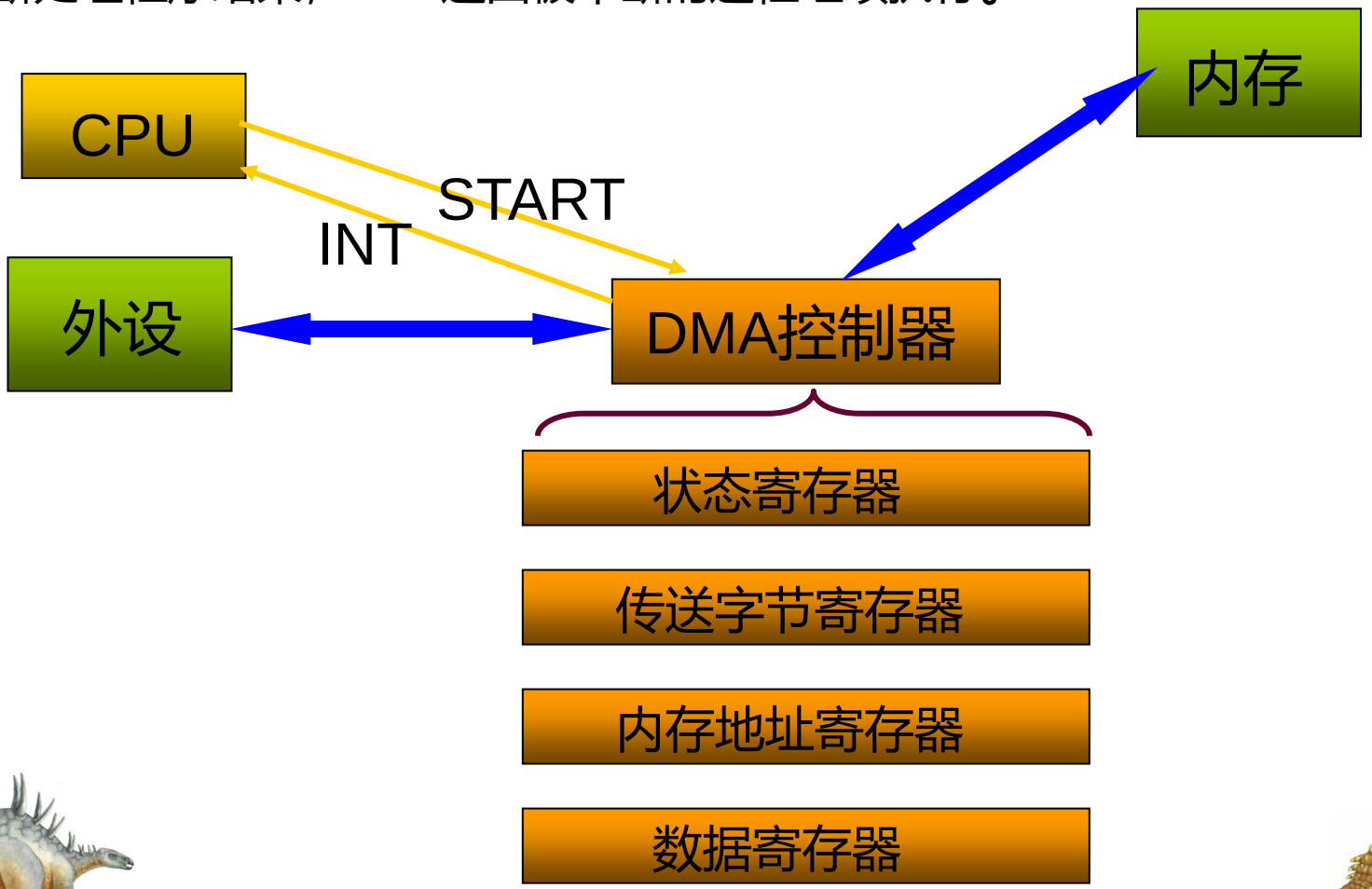
DMA方式 在传送开始需要CPU做一些初始化和传输结束做一些善后处理工作之外，在整个数据传输过程中，不需要CPU任何干预。

1. 进程要求输入数据，CPU将输入数据的内存地址及个数送入相应寄存器；
2. 进程进入等待状态，进程调度程序调度其他进程；
3. 在DMA控制器控制下，将数据寄存器的数据不断写入内存；

DMA方式

4. 传送完毕，DMA控制器向CPU发出中断请求，CPU转中断处理程序；

5. 中断处理程序结束，CPU返回被中断的进程继续执行。



DMA方式

1. 负责在高速外围设备与内存之间成批量的数据传输工作，但是不对数据作再加工处理，I/O操作类型简单；
2. 需要使用专门的DMA控制器：控制、状态寄存器、传送字数寄存器、内存地址寄存器和数据缓冲寄存器。
3. 采用“**偷窃总线控制权**”，不用CPU干预
4. **每传送一个数据并不产生中断**，只有本次DMA传送的数据全部完毕时，才产生中断。

DMA方式与中断的主要区别

- 中断方式是在**数据缓冲寄存区满后**，发中断请求， CPU 进行中断处理，DMA方式则是在所要求传送的数据块**全部传送结束**时要求 CPU 进行中断处理，大大减少了 CPU进行中断处理的次数。
- 中断方式的数据传送是由 CPU 控制完成的而 DMA方式则是在DMA控制器的控制下不经过 CPU 控制完成的。

操作系统设计问题-设计目标

1. 提高使用效率

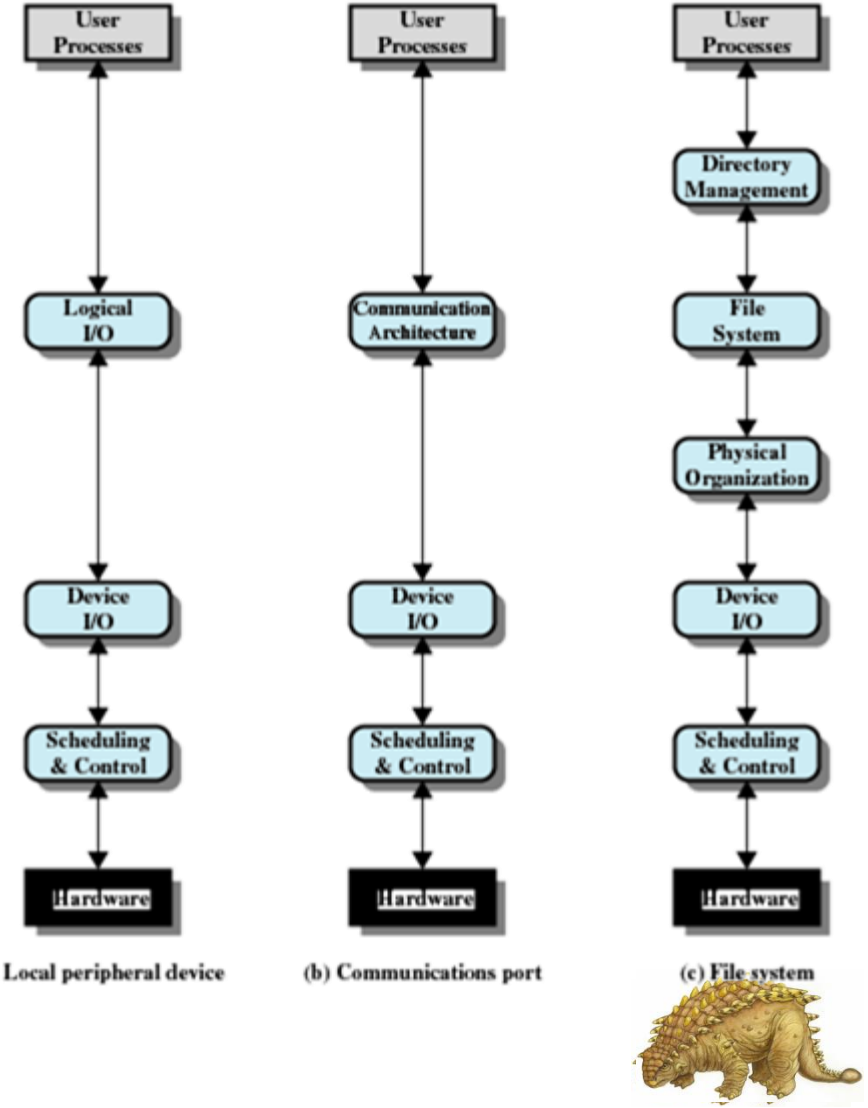
提高CPU与外设、外设与外设之间的并行工作的能力。中断和通道技术的引入提供了这种并行性。

2. 通用性

是使用户能独立于**具体设备的复杂物理特性**而方便的使用设备。也就是说避开各种物理设备的具体使用细节，呈现在用户面前的是**简单、易操作的逻辑设备**

I/O功能逻辑结构

- **逻辑I/O**：把设备当作一个逻辑资源来处理，并不关心实际控制设备的细节。
- **设备I/O**：请求的操作和数据被转换成适当的I/O指令序列、通道命令和控制命令。
- **调度和控制**：关于I/O操作的排队、调度和控制实际发生在这一层，该层与I/O模块和设备硬件**真正发生交互**的软件层。



I/O缓冲

为解决计算机外设的和中央处理器之间速度不匹配，所产生的“瓶颈”现象，提高系统性能，因此引入了缓冲技术。

- 改善中央处理器与外围设备之间速度不配的矛盾，
- 提高CPU和I/O设备的并行性。
- **面向块的I/O设备；**
- **面向流的I/O设备；**

I/O缓冲-缓冲区管理

- **单缓冲**

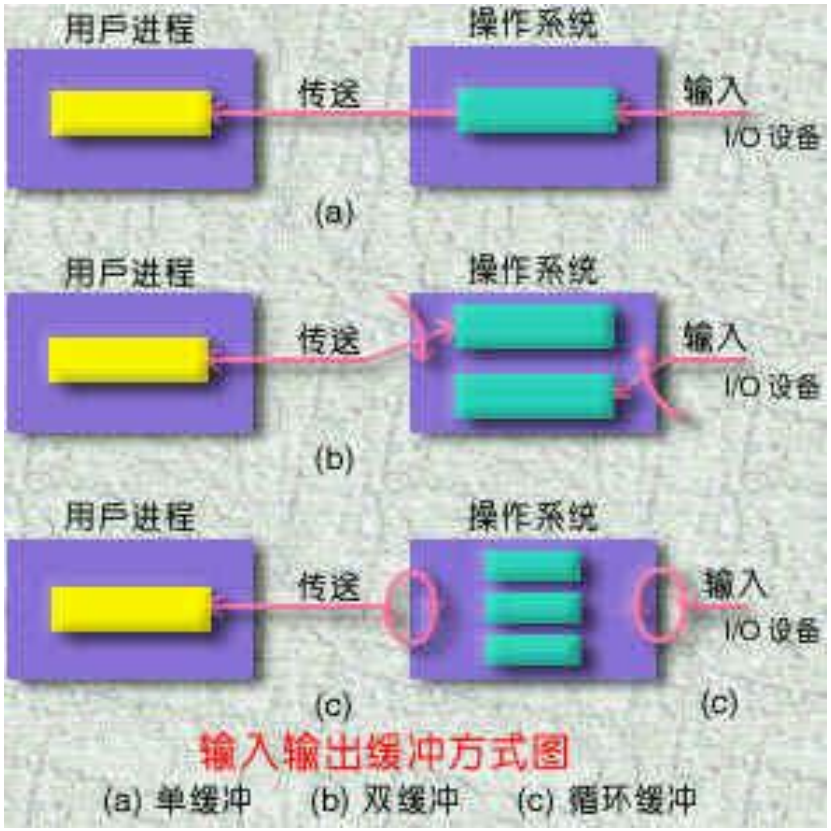
  当用户进程发出I/O请求时，操作系统在主存的系统空间为该操作分配一个缓冲区，可以实现预读和滞后写。

- **双缓冲**

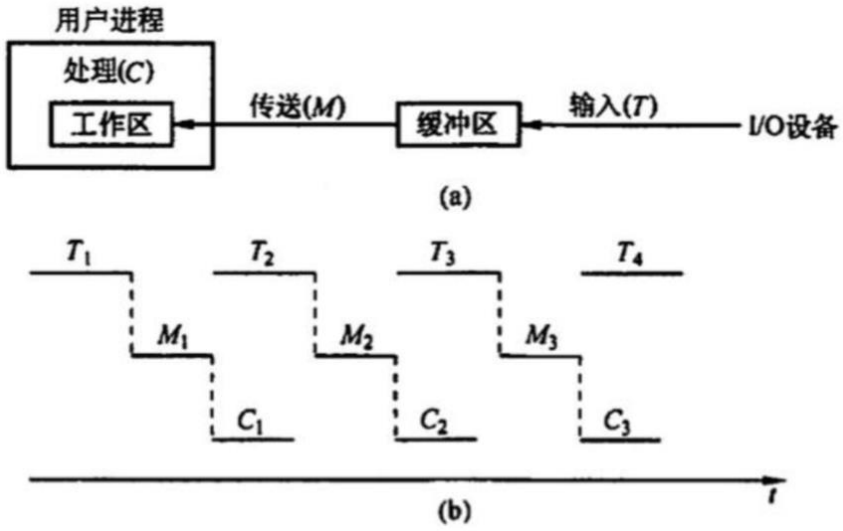
  可以实现用户数据区—缓冲区之间交换数据和**缓冲区—外设**之间交换数据的并行。

- **循环缓冲**

  多个缓冲区连接起来统一管理



I/O缓冲-缓冲区管理（单缓冲）



初始状态：工作区是满的，缓冲区是空的

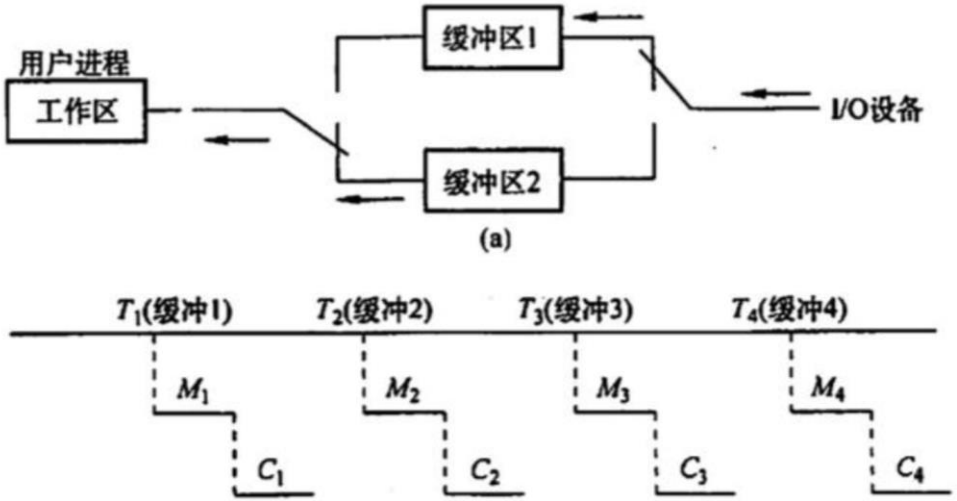
T:磁盘把一块数据输入到缓冲区的时间，

M:缓冲区中的数据传送到工作区的时间，

C :CPU对这一块数据处理的时间。

对于一块数据的处理时间是?

I/O缓冲-缓冲区管理（双缓冲）



初始状态：工作区是空的，一个缓冲区是满的，一个缓冲区是空的。

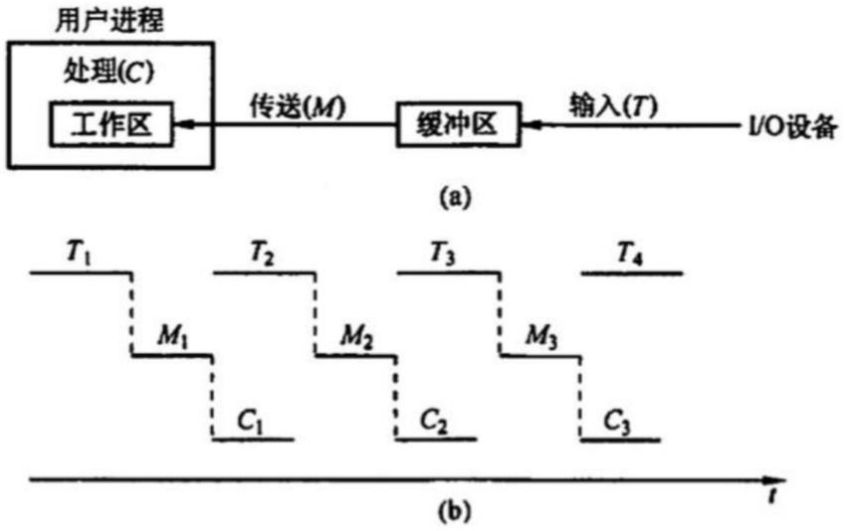
T:磁盘把一块数据输入到缓冲区的时间，

M:缓冲区中的数据传送到工作区的时间，

C :CPU对这一块数据处理的时间

对于一块数据的处理时间是?

I/O缓冲-缓冲区管理（单缓冲）



初始状态：工作区是满的，缓冲区是空的

T:磁盘把一块数据输入到缓冲区的时间，

M:缓冲区中的数据传送到工作区的时间，

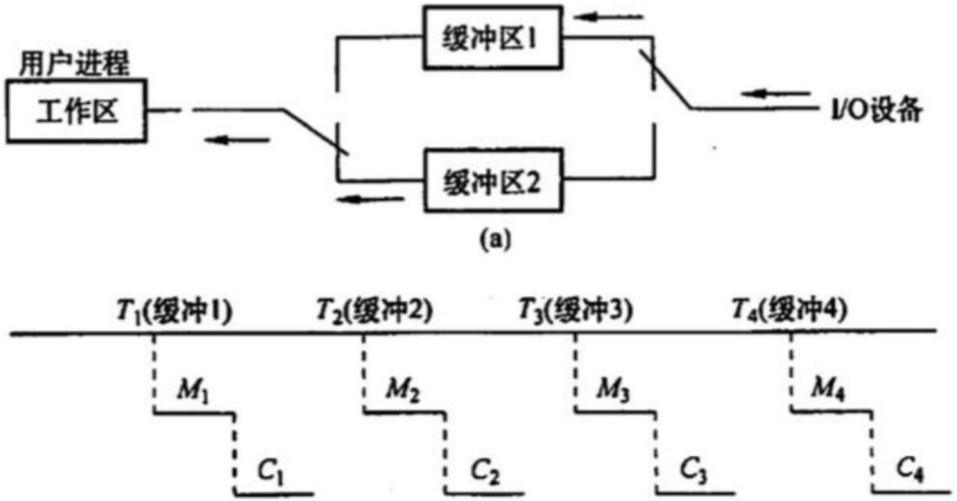
C :CPU对这一块数据处理的时间。

当第一块数据送入用户工作区后，缓冲区空闲，可以传送第二块数据。

这样第一块数据的处理C1与第二块数据的输入T2可并行，依次类推。

系统对每一块数据的处理时间为：**Max**（C，T）+M。

I/O缓冲-缓冲区管理（双缓冲）



初始状态：工作区是空的，一个缓冲区是满的，一个缓冲区是空的。

T:磁盘把一块数据输入到缓冲区的时间，

M:缓冲区中的数据传送到工作区的时间，

C :CPU对这一块数据处理的时间

T<C+M：缓冲区2开始向工作区传数据，I/O设备向缓冲区1开始传数据。当工作区传满数据后（用时M），缓冲区2为空。然后工作区开始处理数据， I/O设备继续向缓冲区1传入数据，当缓冲区1满数据后（用时T），工作区的数据还没有处理完毕，当工作区的数据处理完毕后（共用时C+M），此时工作区为空，缓冲区1满，2空，达到下一个初始状态；

T>C+M：同样可得用时为T，故处理时间为**Max**(C+M, T)。