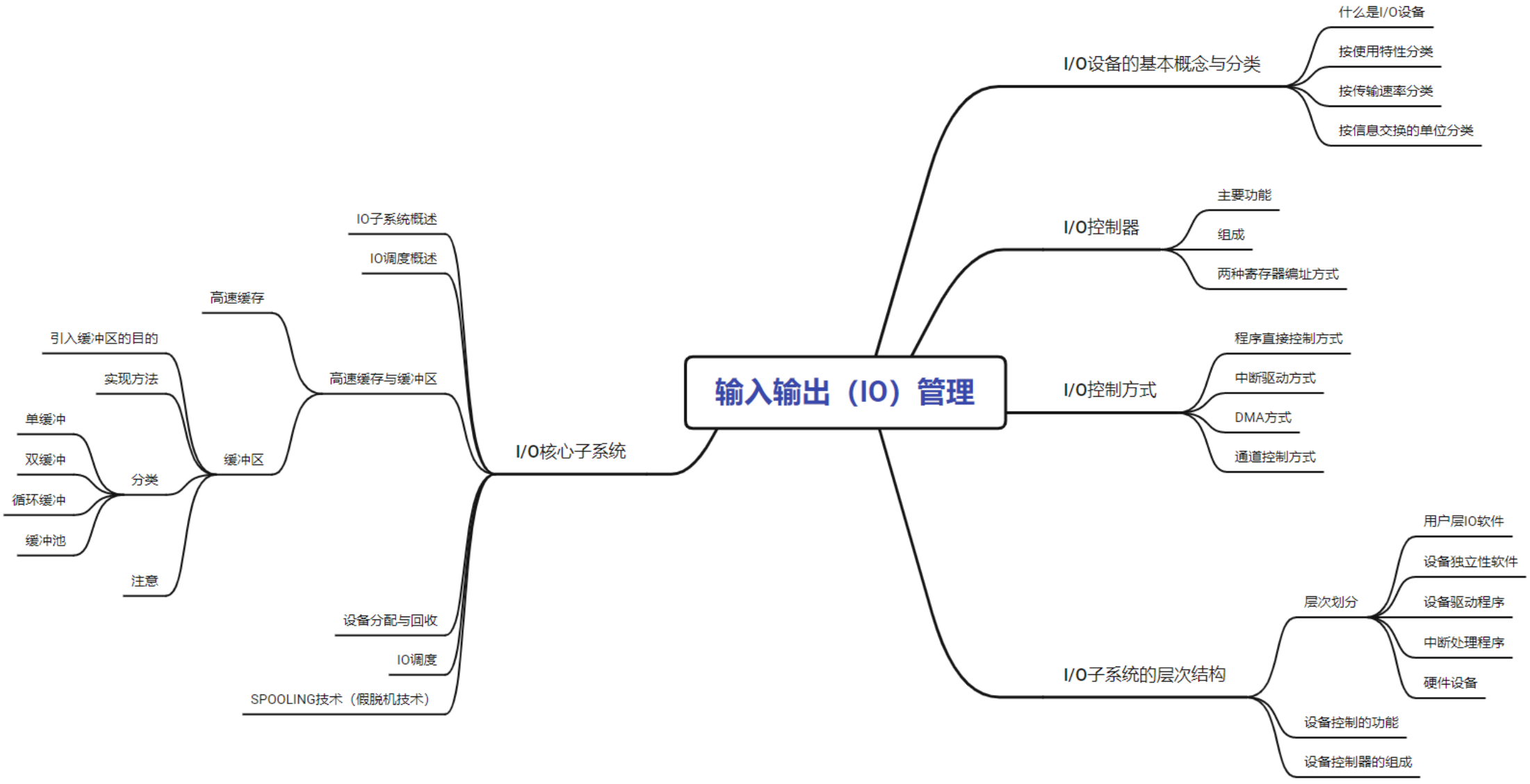
计算机基础-操作系统

# 输入输出（IO）管理

宿船长

2


输入输出（IO）管理
I/O设备的基本概念与分类
什么是I/O设备
按使用特性分类
按传输速率分类
按信息交换的单位分类
I/O控制器
主要功能
组成
两种寄存器编址方式
I/O控制方式
程序直接控制方式
中断驱动方式
DMA方式
通道控制方式
I/O子系统的层次结构
层次划分
用户层IO软件
设备独立性软件
设备驱动程序
中断处理程序
硬件设备
设备控制的功能
设备控制器的组成
I/O核心子系统
IO子系统概述
IO调度概述
高速缓存与缓冲区
高速缓存
引入缓冲区的目的
实现方法
缓冲区
分类
单缓冲
双缓冲
循环缓冲
缓冲池
注意
设备分配与回收
IO调度
SPOOLING技术（假脱机技术）

# IO设备的基本概念和分类

# I/O设备的基本概念和分类——什么是I/O设备

“I/O”就是“输入/输出”（Input/Output）
I/O设备就是可以将数据输入到计算机，或者可以接收计算机输出数据的外部设备，属于计算机中的硬件部件。

鼠标、键盘——典型的输入型设备

显示器——输出型设备

移动硬盘——即可输入、又可输出的设备

# I/O设备的基本概念和分类——什么是I/O设备

“I/O”就是“输入/输出”（Input/Output）
I/O设备就是可以将数据输入到计算机，或者可以接收计算机输出数据的外部设备，属于计算机中的硬件部件。

Write操作：向外部设备写出数据

Read操作：从外部设备读入数据

UNIX系统将外部设备抽象为一种特殊的文件，用户可以使用与文件操作相同的方式对外部设备进行操作。

# I/O设备的基本概念和分类——I/O设备的分类（按使用特性分类）

按使用特性分类
- 人机交互类外部设备：数据传输速度慢
- 存储设备：数据传输速度快
- 网络通信设备：数据传输速度介于上述二者之间

人机交互类外设：鼠标、键盘、打印机等——用于人机交互

存储设备：移动硬盘、光盘等——用于数据存储

网络通信设备：调制解调器等——用于网络通信

# I/O设备的基本概念和分类——I/O设备的分类（按传输速率分类）

按传输速率分类
- 低速设备
- 中速设备
- 高速设备

低速设备：鼠标、键盘等——传输速率为每秒几个到几百字节

中速设备：如激光打印机等——传输速率为每秒数千至上万个字节

高速设备：如磁盘等——传输速率为每秒数千字节至千兆字节的设备

# I/O设备的基本概念和分类——I/O设备的分类（按信息交换的单位分类）

按信息交换的单位分类

- 块设备（传输快，可寻址）
- 字符设备（传输慢，不可寻址，常采用中断驱动方式）

传输速率较高，可寻址，即对它可随机地读/写任一块

传输速率较慢，不可寻址，在输入/输出时常采用中断驱动方式

块设备：如磁盘等——数据传输的基本单位是“块”

字符设备：鼠标、键盘等——数据传输的基本单位是字符。

# IO控制器

# I/O控制器——I/O设备的电子部件（I/O控制器）

CPU无法直接控制I/O设备的机械部件，因此I/O设备还要有一个电子部件作为CPU和I/O设备机械部件之间的“中介”，用于实现CPU对设备的控制。

这个电子部件就是I/O控制器，又称设备控制器。CPU可控制I/O控制器，又由I/O控制器来控制设备的机械部件。

主要功能：

- 接受和识别CPU发出的命令（要有控制寄存器）
- 向CPU报告设备的状态（要有状态寄存器）
- 数据交换（要有数据寄存器，暂存输入输出的数据）
- 地址识别（由I/O逻辑实现）

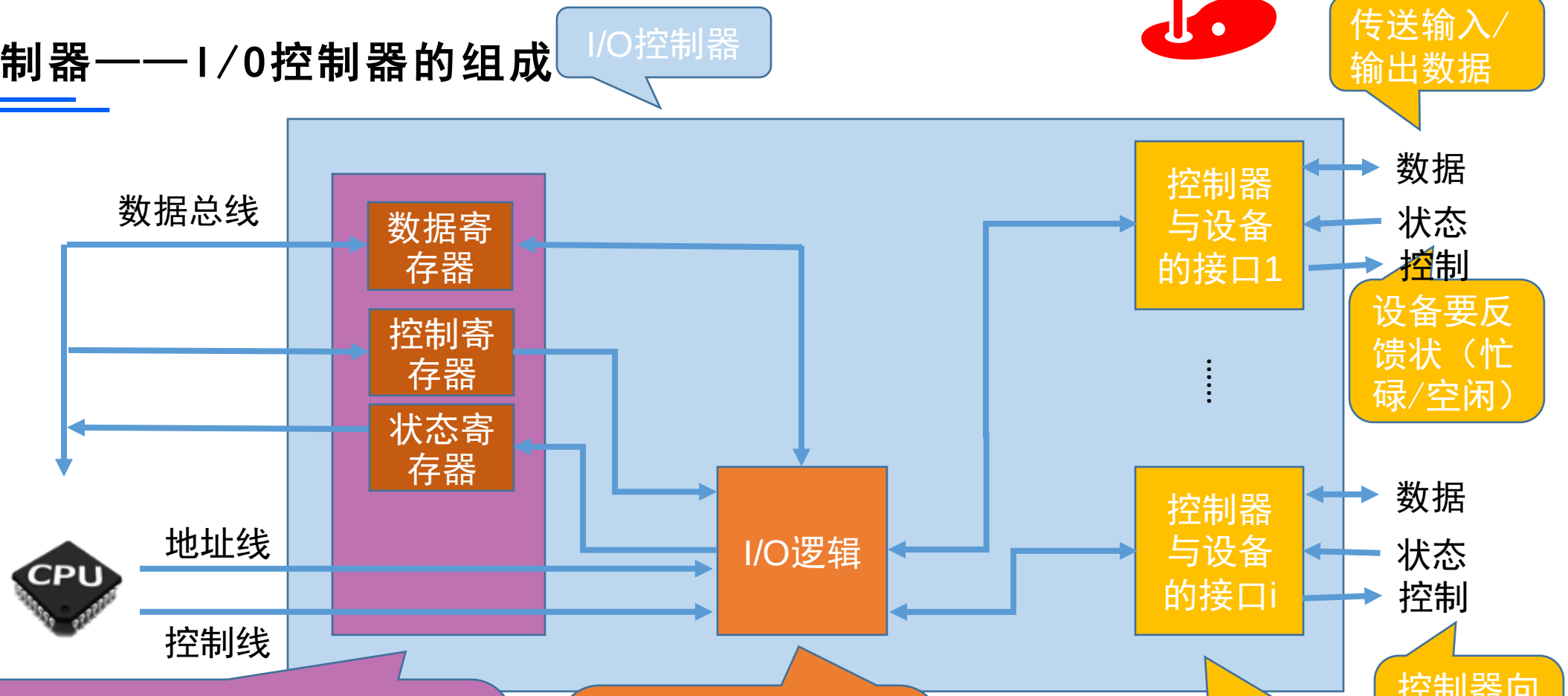
如CPU发来的read/write命令，I/O控制器中会有相应的控制寄存器来存放命令和参数

I/O控制器中会有相应的状态寄存器，用于记录I/O设备的当前状态。如：1表示空闲，0表示忙碌

I/O控制器中会设置相应的数据寄存器。输出时，数据寄存器用于暂存CPU发来的数据，之后再由控制器传送设备。输入时，数据寄存器用于暂存设备发来的数据，之后CPU从数据寄存器中取走数据。

类似于内存的地址，为了区分设备控制器中的各个寄存器，也需要给各个寄存器设置一个特定的“地址”。I/O控制器通过CPU提供的“地址”来判断CPU要读/写的是哪个寄存器

# I/O控制器——I/O控制器的组成




*CPU与控制器的接口*

用于实现CPU与控制器之间的通信。CPU通过控制线发出命令；通过地址线指明要操作的设备；通过数据线来取出（输入）数据，或放入（输出）数据

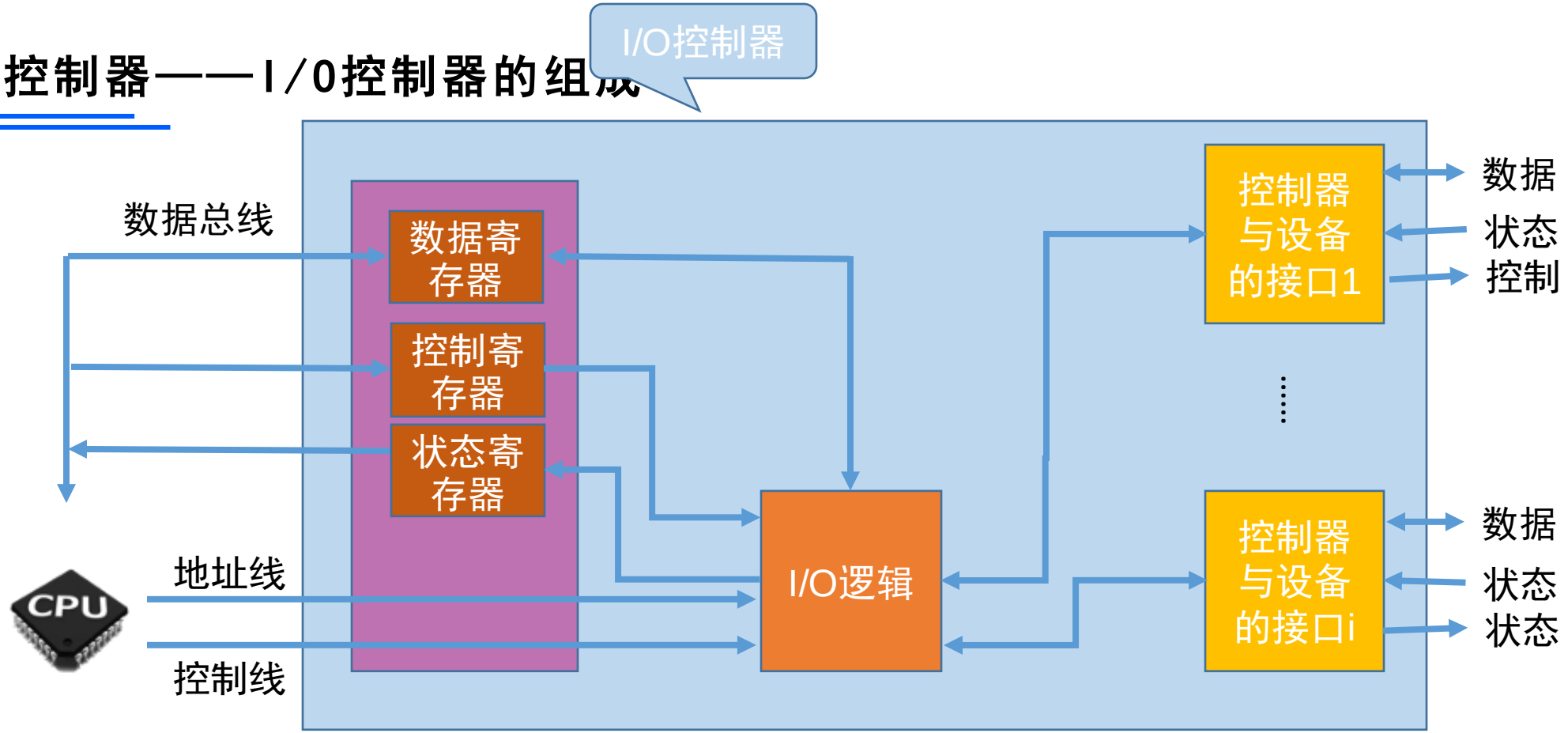
*I/O逻辑*

负责接收和识别CPU的各种命令（如地址译码），并负责对设备发出命令

*控制器与设备的接口*

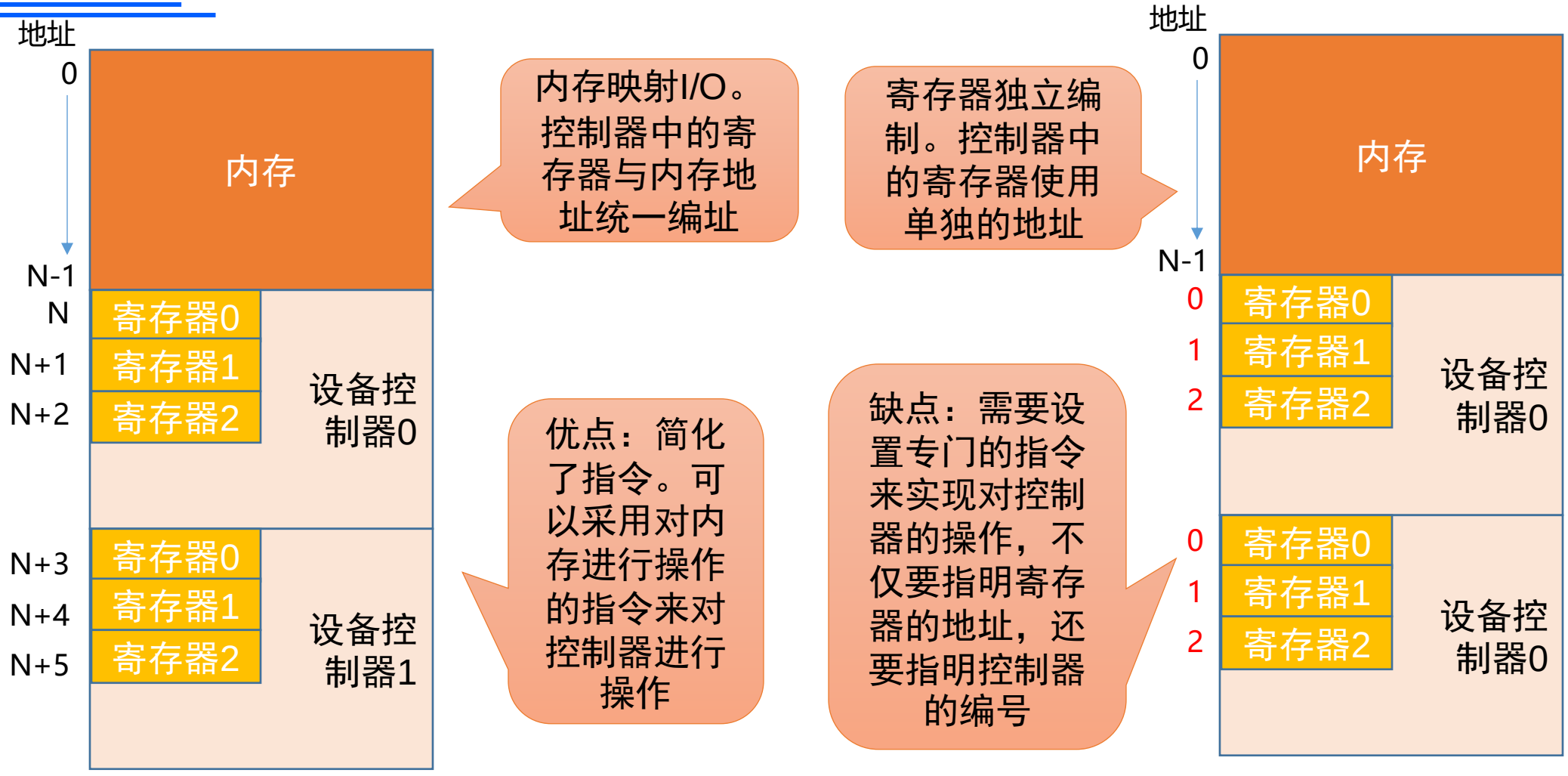
用于实现控制器与设备之间的通信

# I/O控制器——I/O控制器的组成




值得注意的小细节：①一个I/O控制器可能会对应多个设备；
②数据寄存器、控制寄存器、状态寄存器可能有多个（如：每个控制/状态寄存器对应一个具体的设备），且这些寄存器都要有相应的地址，才能方便CPU操作。有的计算机会让这些寄存器占用内存地址的一部分，称为内存映像I/O；另一些计算机则采用I/O专用地址，即寄存器独立编址。

# I/O控制器——内存映像I/O v. s. 寄存器独立编址

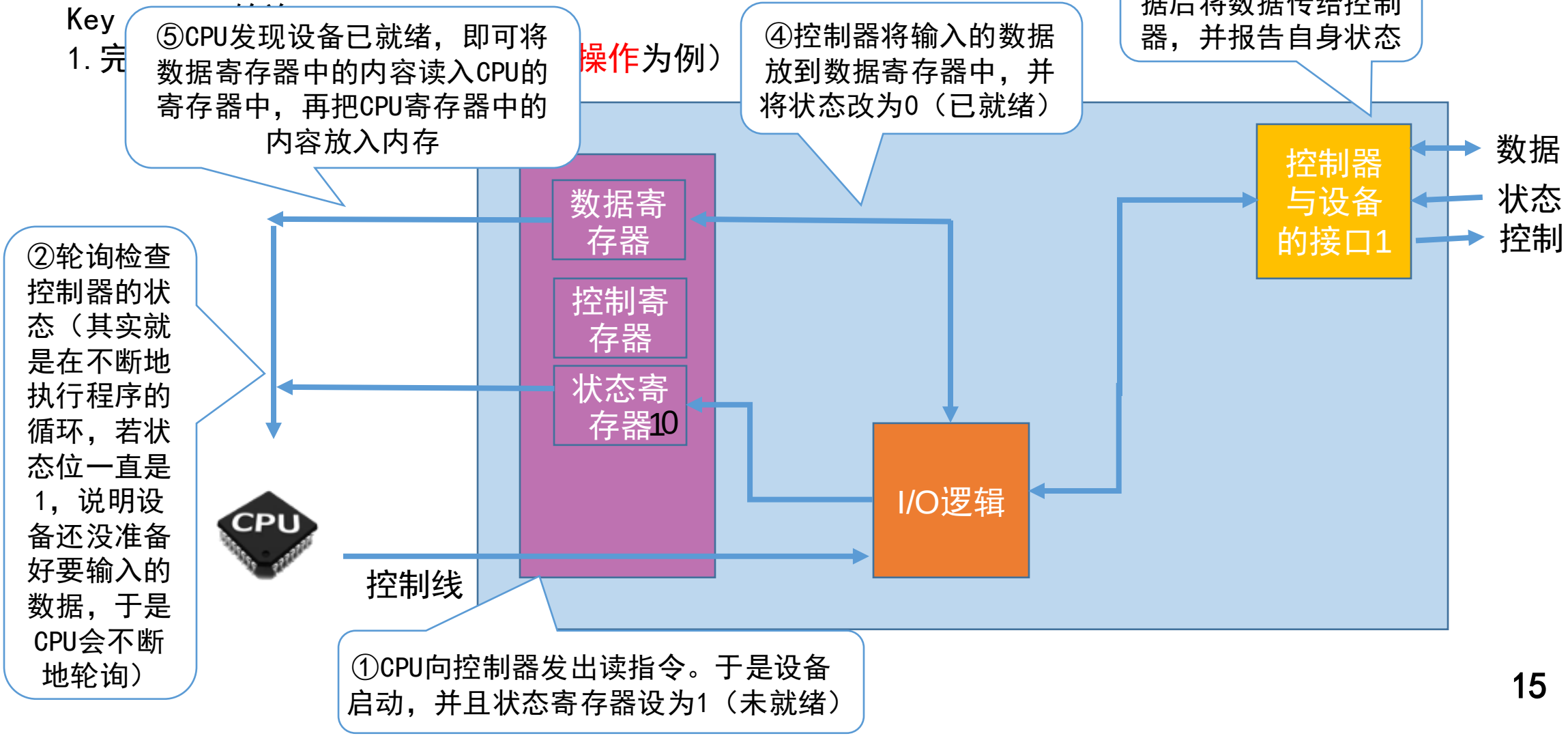


# IO控制方式

# I/O控制方式——程序直接控制方式

Key
1. 完

操作为例）



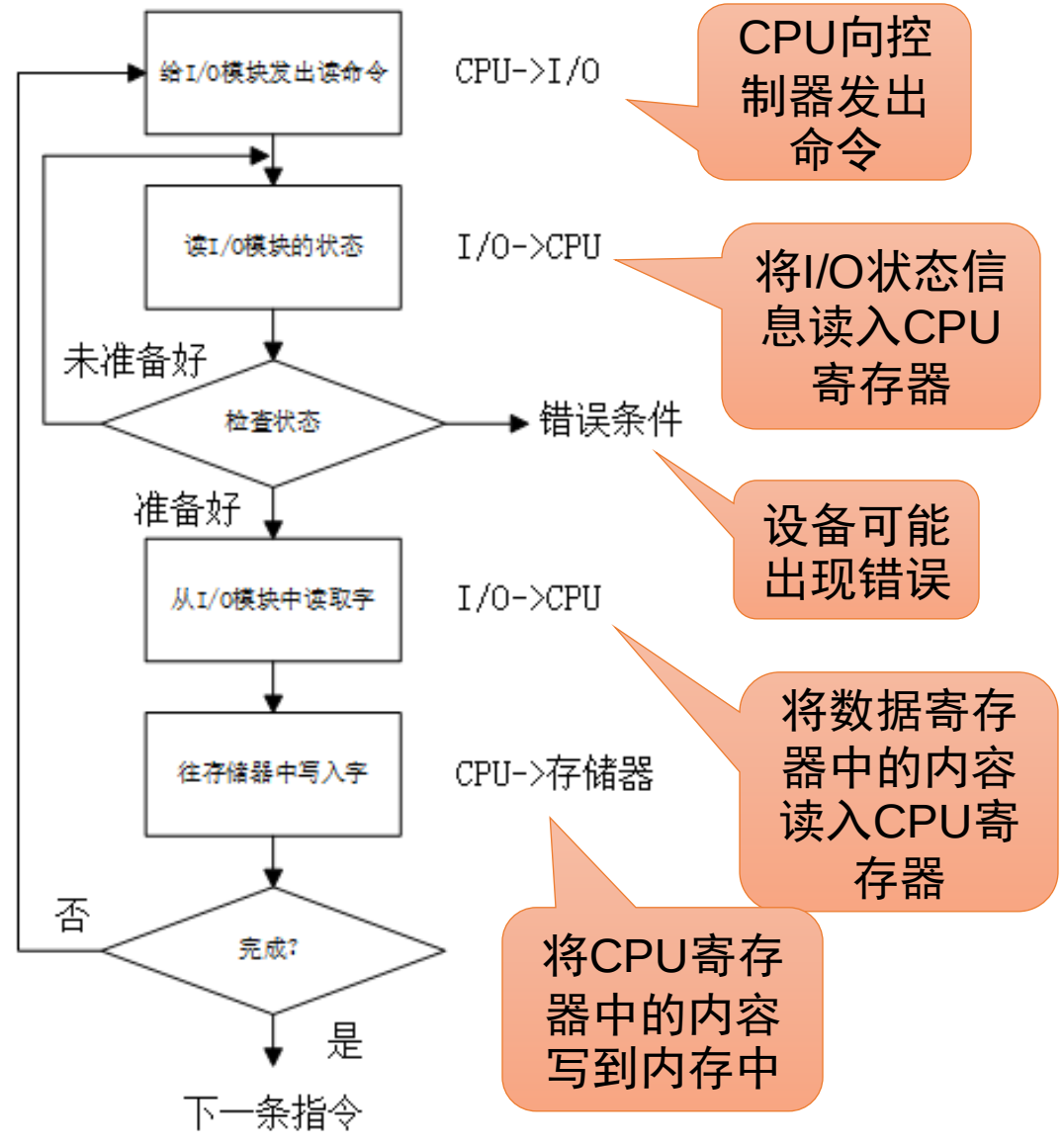
# I/O控制方式——程序直接控制方式

1. 完成一次读/写操作的流程（见右图，Key word：轮询）

```
01. #include <stdio.h>
02. #include <stdlib.h>
03. int main()
04. {
05.     int a, b, c, d;
06.     scanf("%d", &a);  //输入整数并赋值给变量a
07.     scanf("%d", &b);  //输入整数并赋值给变量b
08.     printf("a+b=%d\n", a+b);  //计算a+b的值
09.     scanf("%d %d", &c, &d);  //输入两个整数并分别赋值给c、d
10.     printf("c*d=%d\n", c*d);  //计算c*d的值
11.
12.     system("pause");
13.     return 0;
14. }
```

输入的数据最终要放到内存中（a/b/c/d变量存放在内存中）

同理，输出的数据也存放在内存中，需要从内存取出




（a）程序直接控制方式

# I/O控制方式——程序直接控制方式

1. 完成一次读/写操作的流程（见右图，Key word：轮询）

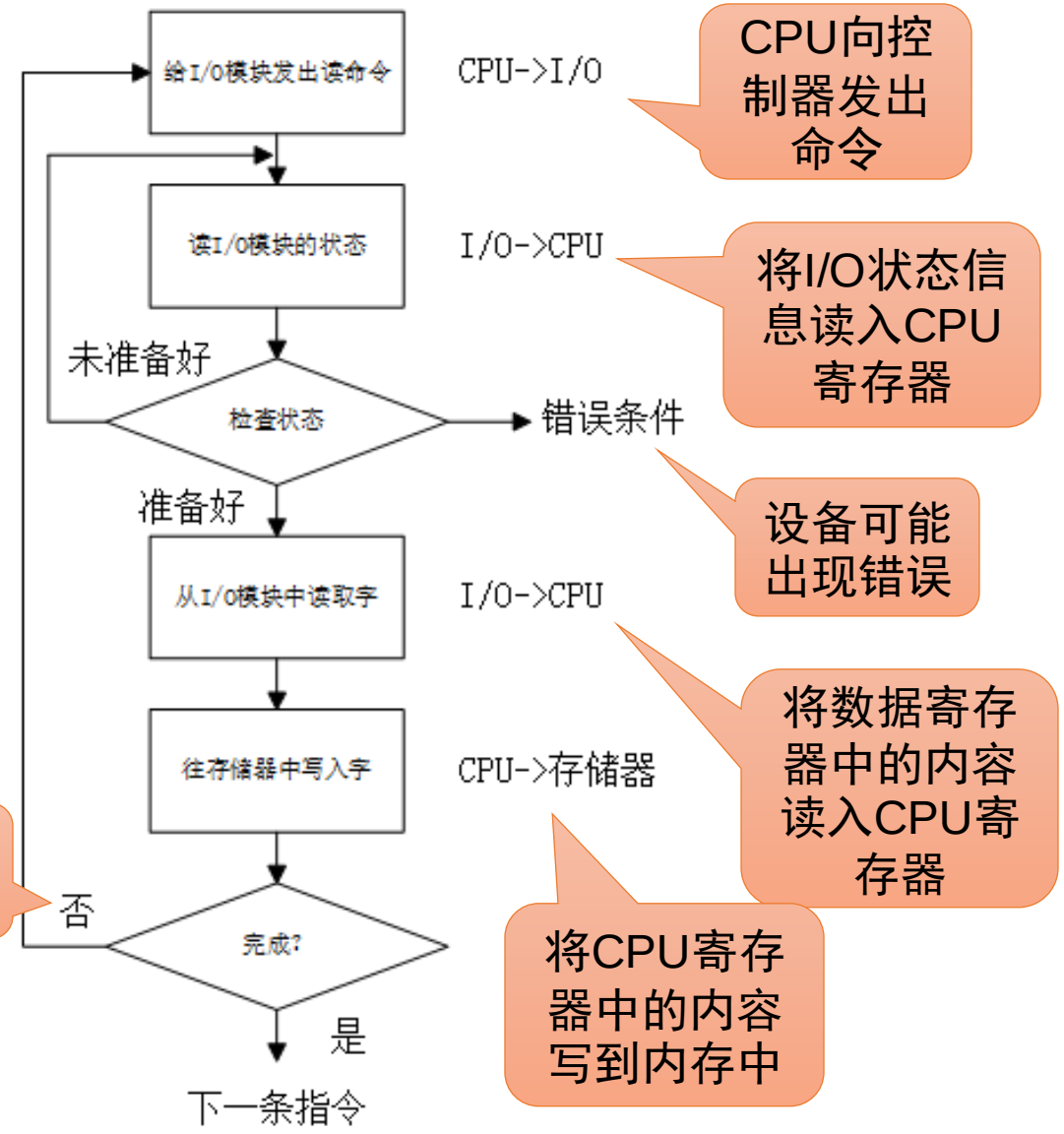
2. CPU干预的频率
很频繁，I/O操作开始之前、完成之后需要CPU介入，并且在等待I/O完成的过程中CPU需要不断地轮询检查。

3. 数据传送的单位
每次读/写一个字

指的是CPU的寄存器

4. 数据的流向
读操作（数据输入）：I/O设备->CPU->内存
写操作（数据输出）：内存->CPU->I/O设备
每个字的读/写都需要CPU的帮助

5. 主要缺点和主要优点
优点：实现简单。在读/写指令之后，加上实现循环检查的一系列指令即可（因此才称为“程序直接控制方式”）
缺点：CPU和I/O设备只能串行工作，CPU需要一直轮询检查，长期处于“忙等”状态，CPU利用率低。

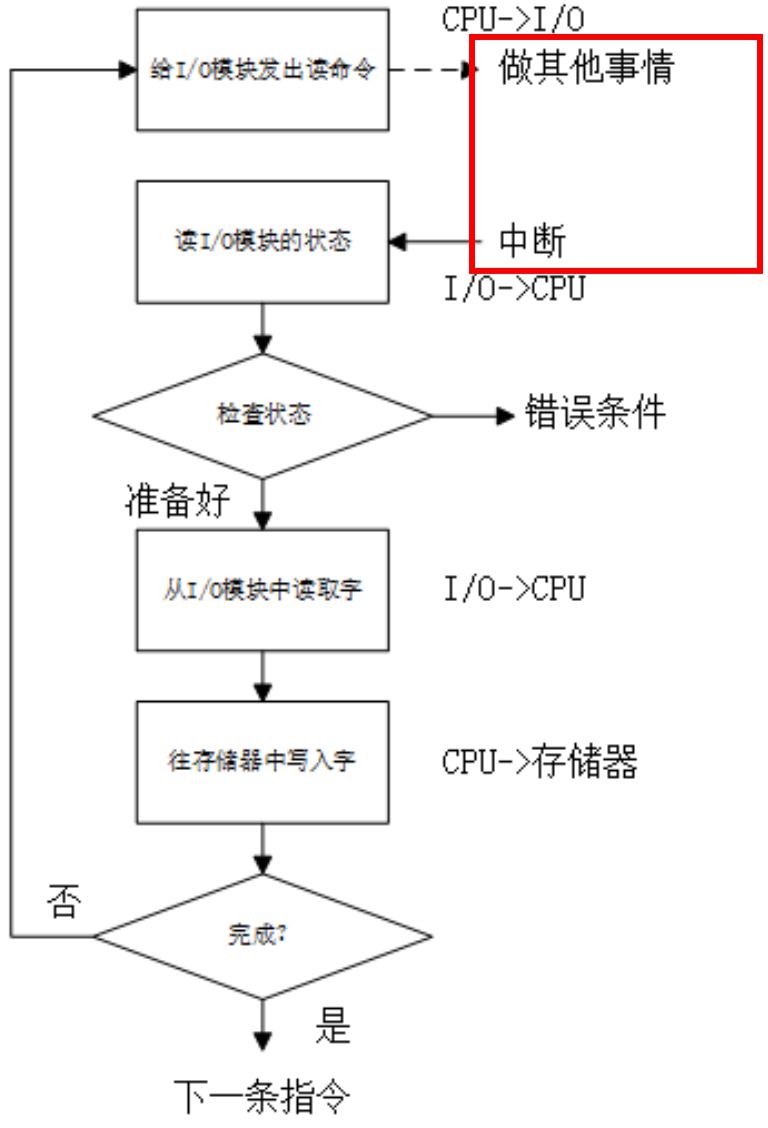



（a）程序直接控制方式

# I/O控制方式——中断驱动方式

引入中断机制。由于I/O设备速度很慢，因此在CPU发出读/写命令后，可将等待I/O的进程阻塞，先切换到别的进程执行。当I/O完成后，控制器会向CPU发出一个中断信号，CPU检测到中断信号后，会保存当前进程的运行环境信息，转去执行中断处理程序处理该中断。处理中断的过程中，CPU从I/O控制器读一个字的数据传送到CPU寄存器，再写入主存。接着，CPU恢复等待I/O的进程（或其他进程）的运行环境，然后继续执行。

注意：
①CPU会在每个指令周期的末尾检查中断；
②中断处理过程中需要保存、恢复进程的运行环境，这个过程是需要一定时间开销的。可见，如果中断发生的频率太高，也会降低系统性能。




（b）中断驱动方式

# I/O控制方式——中断驱动方式

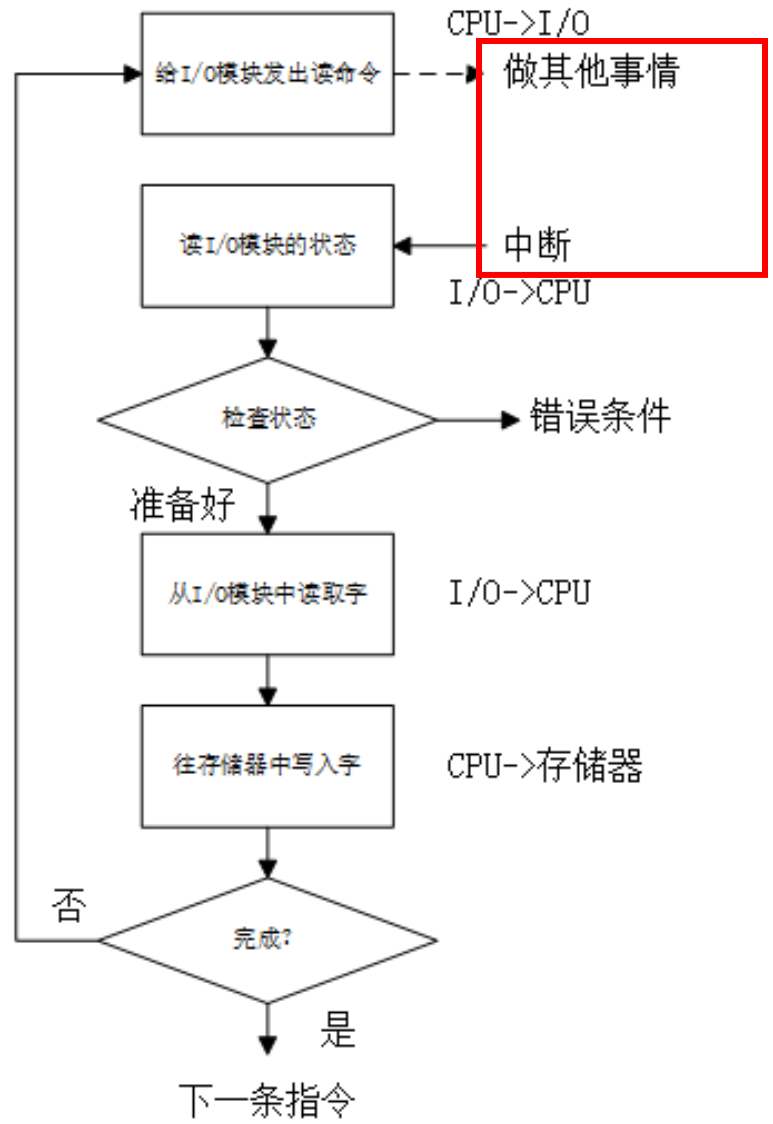
1. 完成一次读/写操作的流程（见右图，Key word：中断）

2. CPU干预的频率
每次I/O操作开始之前、完成之后需要CPU介入。
等待I/O完成的过程中CPU可以切换到别的进程执行。

3. 数据传送的单位
每次读/写一个字

4. 数据的流向
读操作（数据输入）：I/O设备->CPU->内存
写操作（数据输出）：内存->CPU->I/O设备

5. 主要缺点和主要优点
优点：与“程序直接控制方式”相比，在“中断驱动方式”中，I/O控制器会通过中断信号主动报告I/O已完成，CPU不再需要不停地轮询。
CPU和I/O设备可并行工作，CPU利用率得到明显提升。
缺点：每个字在I/O设备与内存之间的传输，都需要经过CPU。而频繁的中断处理会消耗较多的CPU时间。




（b）中断驱动方式

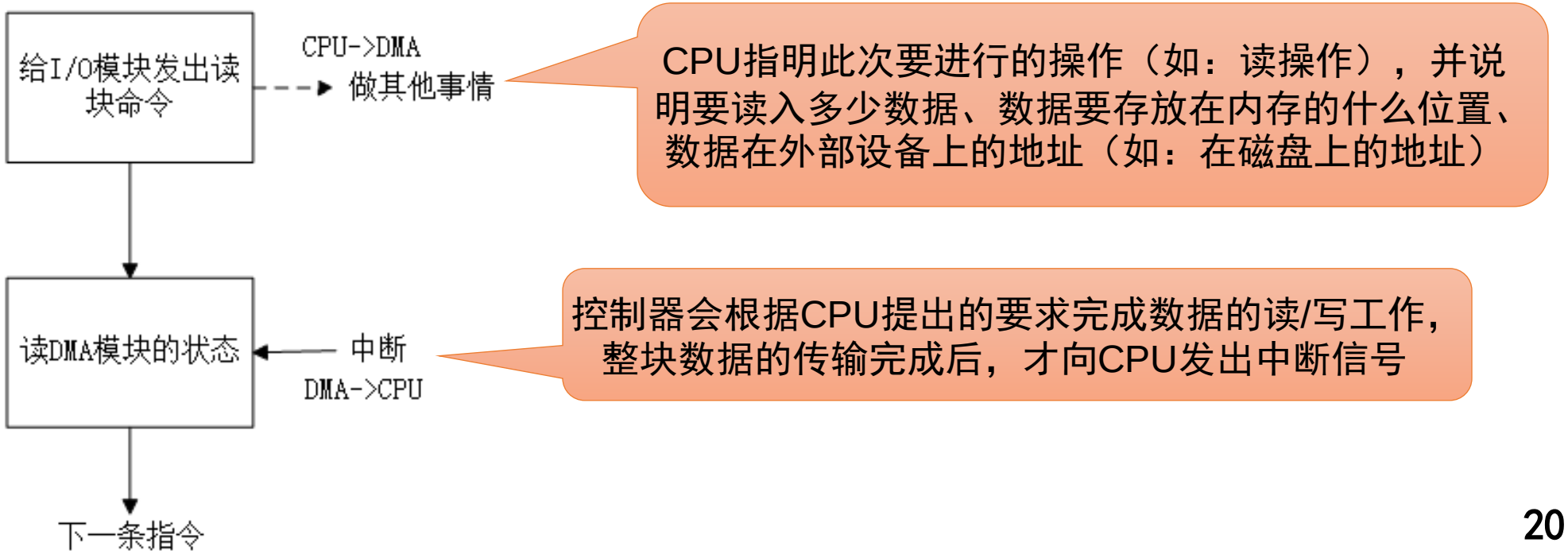
# I/O控制方式——DMA方式

与“中断驱动方式”相比，DMA方式（Direct Memory Access，直接存储器存取。主要用于块设备的I/O控制）有这样几个改进：

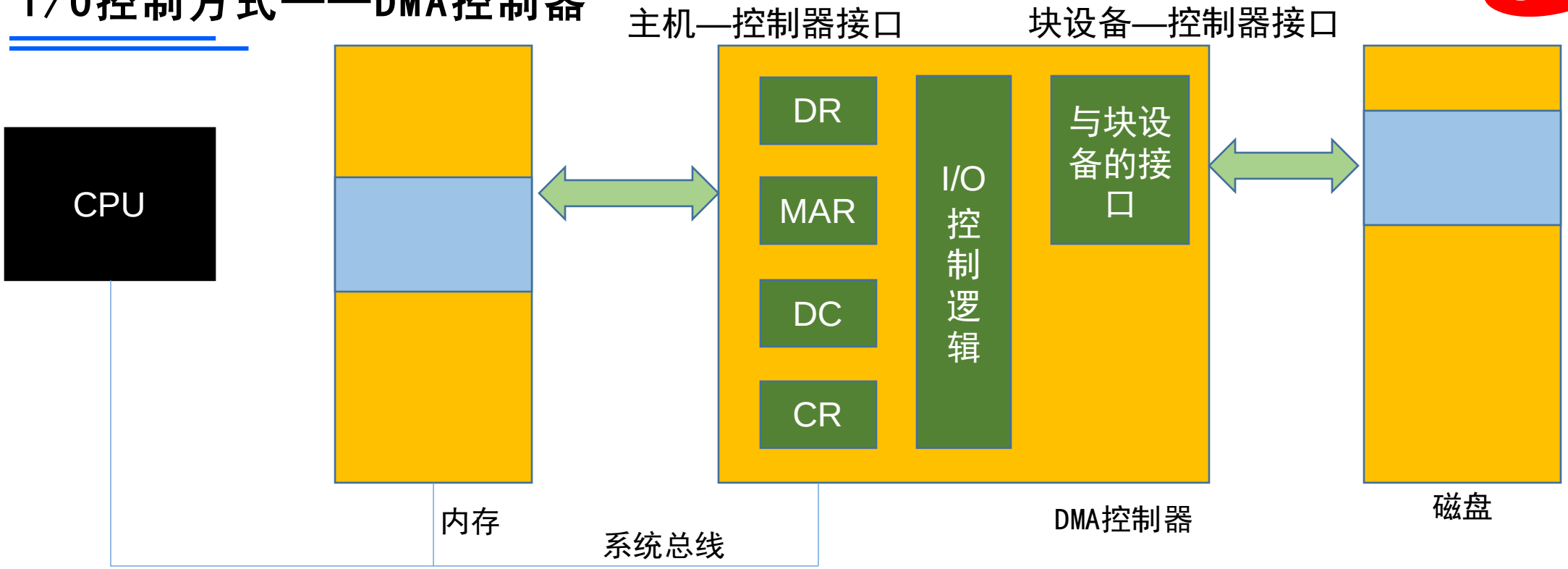
①数据的传送单位是“块”。不再是一个字、一个字的传送；

②数据的流向是从设备直接放入内存，或者从内存直接到设备。不再需要CPU作为“快递小哥”。

③仅在传送一个或多个数据块的开始和结束时，才需要CPU干预。

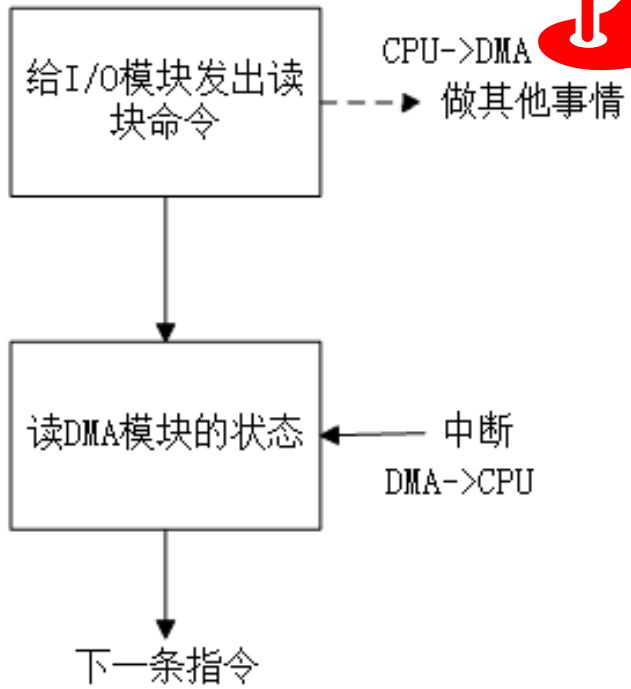


# I/O控制方式——DMA控制器




DR（Data Register，数据寄存器）：暂存从设备到内存，或从内存到设备的数据。
MAR（Memory Address Register，内存地址寄存器）：在输入时，MAR表示数据应放到内存中的什么位置；输出时MAR表示要输出的数据放在内存中的什么位置。
DC（Data Counter，数据计数器）：表示剩余要读/写的字节数。
CR（Command Register，命令/状态寄存器）：用于存放CPU发来的I/O命令，或设备的状态信息。

# I/O控制方式——DMA方式




1. 完成一次读/写操作的流程（见右图）

2. CPU干预的频率
仅在传送一个或多个数据块的开始和结束时，才需要CPU干预。

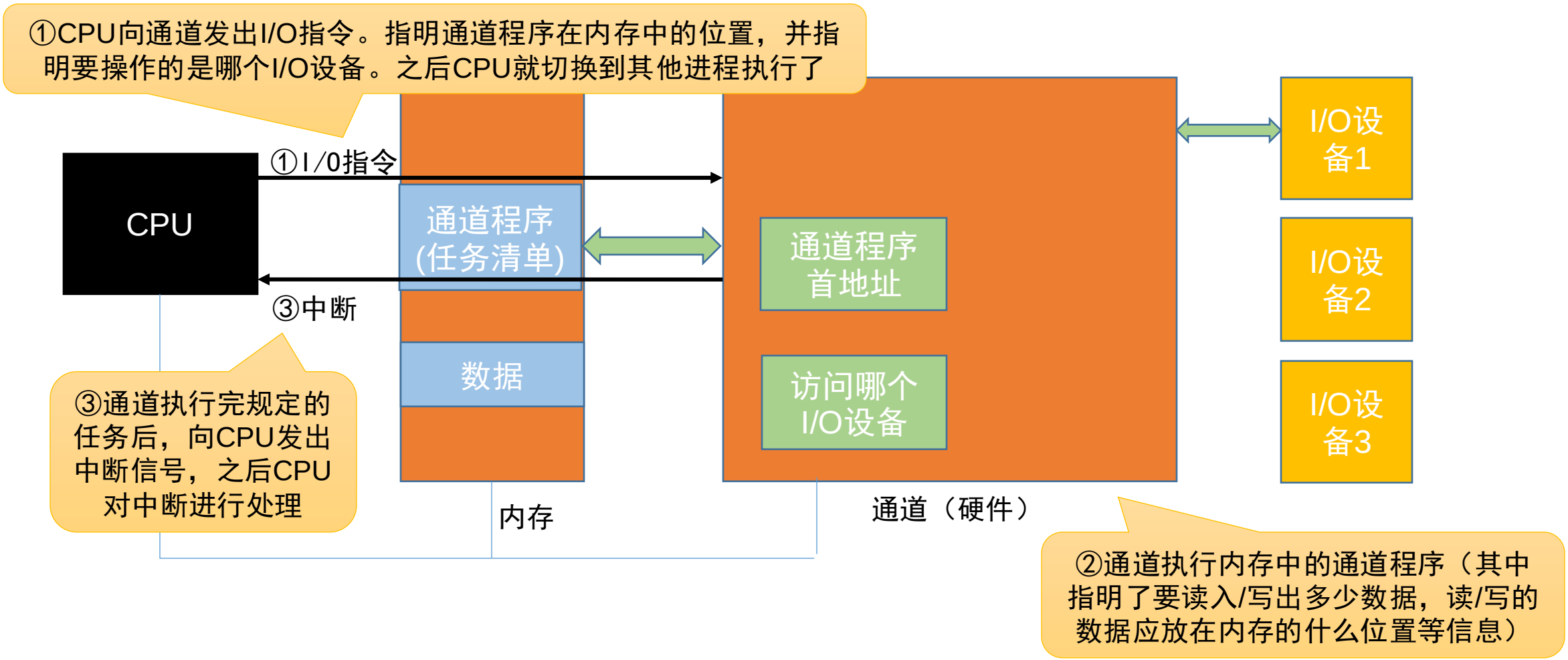
3. 数据传送的单位
每次读/写一个或多个块（注意：每次读写的只能是连续的多个块，且这些块读入内存后在内存中也必须是连续的）

4. 数据的流向（不再需要经过CPU）
读操作（数据输入）：I/O设备->内存
写操作（数据输出）：内存->I/O设备

5. 主要缺点和主要优点
优点：数据传输以“块”为单位，CPU介入频率进一步降低。数据的传输不再需要先经过CPU再写入内存，数据传输效率进一步增加。CPU和I/O设备的并行性得到提升。
缺点：CPU每发出一条I/O指令，只能读/写一个或多个连续的数据块。
如果要读/写多个离散存储的数据块，或者要将数据分别写到不同的内存区域时，CPU要分别发出多条I/O指令，进行多次中断处理才能完成。

# I/O控制方式——通道控制方式

通道：一种硬件，可以理解为是“简化版的CPU”。通道可以识别并执行一系列通道指令

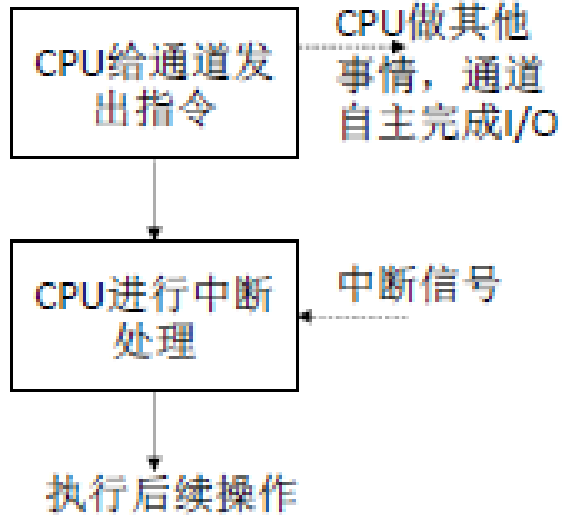


# I/O控制方式——通道控制方式

与CPU相比，通道可以执行的指令很单一，并且通道程序是放在主机内存中的，也就是说通道与CPU共享内存

通道：一种硬件，可以理解为是“简化版的CPU”。通道可以识别并执行一系列通道指令

1. 完成一次读/写操作的流程（见右图）




2. CPU干预的频率
极低，通道会根据CPU的指示执行相应的通道程序，只有完成一组数据块的读/写后才需要发出中断信号，请求CPU干预。

3. 数据传送的单位
每次读/写一组数据块

4. 数据的流向（在通道的控制下进行）
读操作（数据输入）：I/O设备->内存
写操作（数据输出）：内存->I/O设备

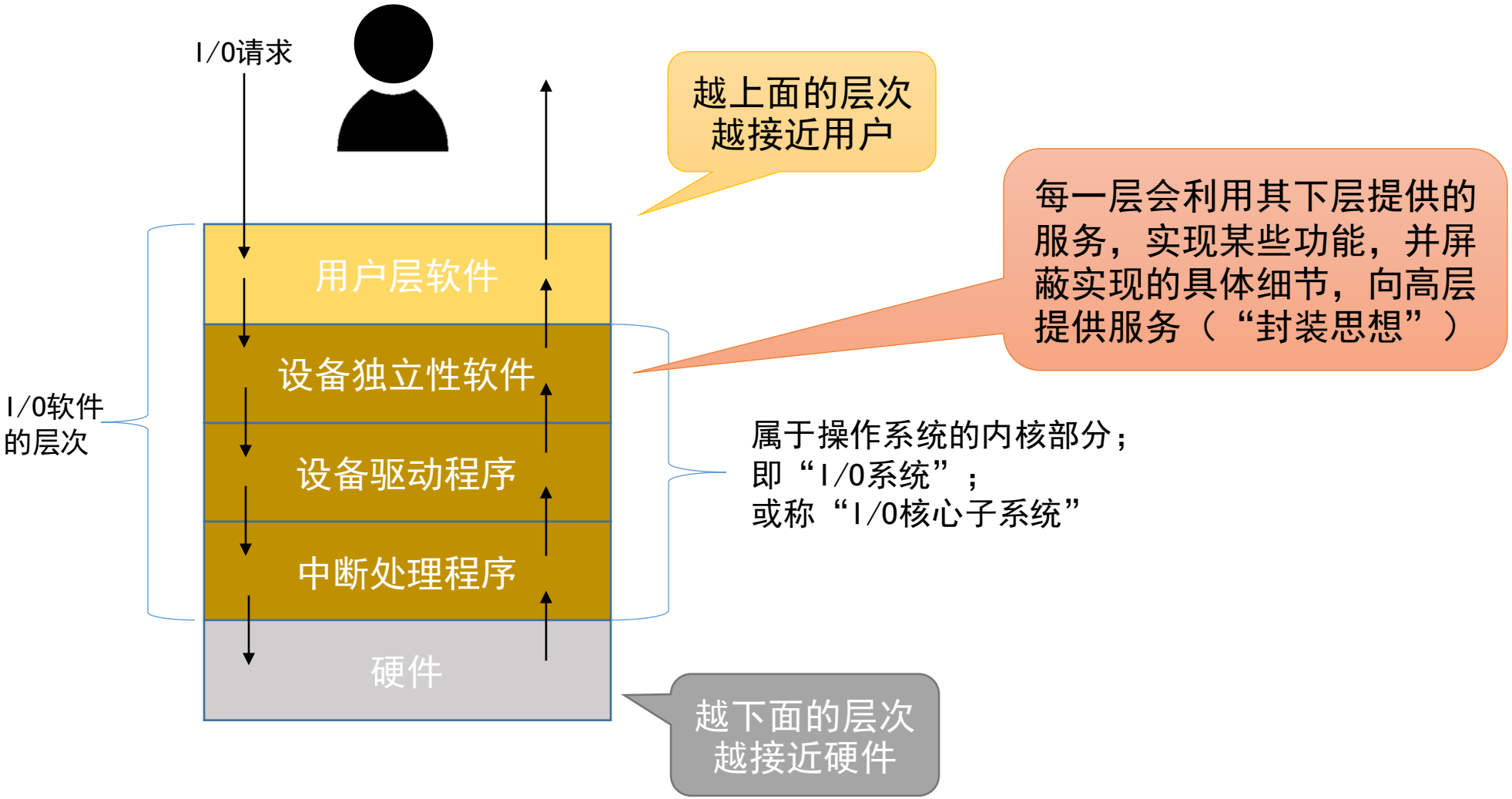
5. 主要缺点和主要优点
缺点：实现复杂，需要专门的通道硬件支持
优点：CPU、通道、I/O设备可并行工作，资源利用率很高。

# I/O控制方式

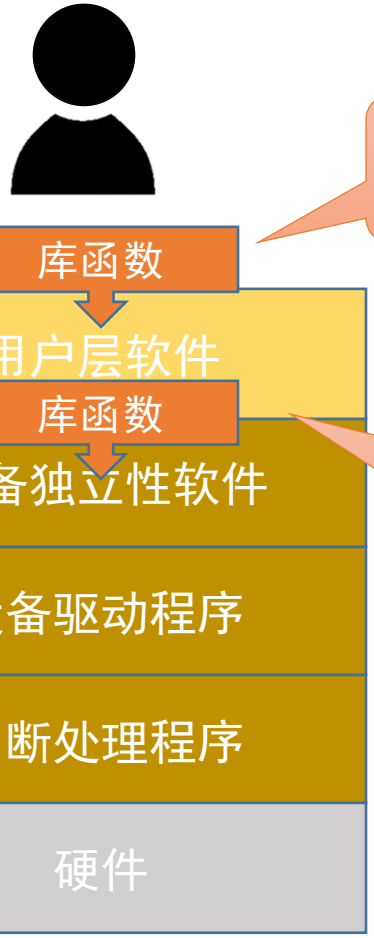
| | 完成一次读/写的过程 | CPU干预频率 | 每次I/O的数据传输单位 | 数据流向 | 优缺点 |
|---|---|---|---|---|---|
| 程序直接控制方式 | CPU发出I/O命令后需要不断轮询 | 极高 | 字 | 设备->CPU->内存<br>内存->CPU->设备 | 每一个阶段的优点都是解决了上一阶段的最大缺点。<br>总体来说，整个发展过程就是要尽量减少CPU对I/O过程的干预，把CPU从繁杂的I/O控制事务中解脱出来，以便更多地去完成数据处理任务。 |
| 中断驱动方式 | CPU发出I/O命令后可以做其他事，本次I/O完成后设备控制器发出中断信号 | 高 | 字 | 设备->CPU->内存<br>内存->CPU->设备 | |
| DMA方式 | CPU发出I/O命令后可以做其他事，本次I/O完成后DMA控制器发出中断信号 | 中 | 块 | 设备->内存<br>内存->设备 | |
| 通道控制方式 | CPU发出I/O命令后可以做其他事。通道会执行通道程序以完成I/O，完成后通道向CPU发出中断信号 | 低 | 一组块 | 设备->内存<br>内存->设备 | |

# IO系统的层次结构

# I/O软件层次结构


I/O请求
用户层软件
设备独立性软件
设备驱动程序
中断处理程序
硬件
I/O软件
的层次
越上面的层次
越接近用户
每一层会利用其下层提供的服务，实现某些功能，并屏蔽实现的具体细节，向高层提供服务（“封装思想”）
属于操作系统的内核部分；
即“I/O系统”；
或称“I/O核心子系统”
越下面的层次
越接近硬件

# I/O软件层次结构——用户层软件




属于操作系统内核部分

用户层软件实现了与用户交互的接口，用户可直接使用该层提供的、与I/O操作相关的库函数对设备进行操作

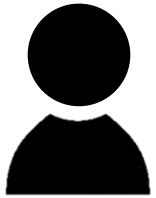
Eg：printf（“hello,world!");

用户层软件将用户请求翻译成格式化的I/O请求，并通过“系统调用”请求操作系统内核的服务

Eg：printf（“hello, world!”）；会被翻译成等价的write系统调用，当然，用户层软件也会在系统调用时填入相应参数。

Windows操作系统向外提供的一系列系统调用，但是由于系统调用的格式严格，使用麻烦，因此在用户层上封装了一系列更方便的库函数接口供用户使用（Windows API）

# I/O软件层次结构——设备独立性软件

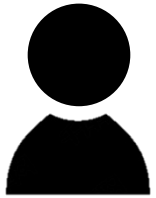


设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。

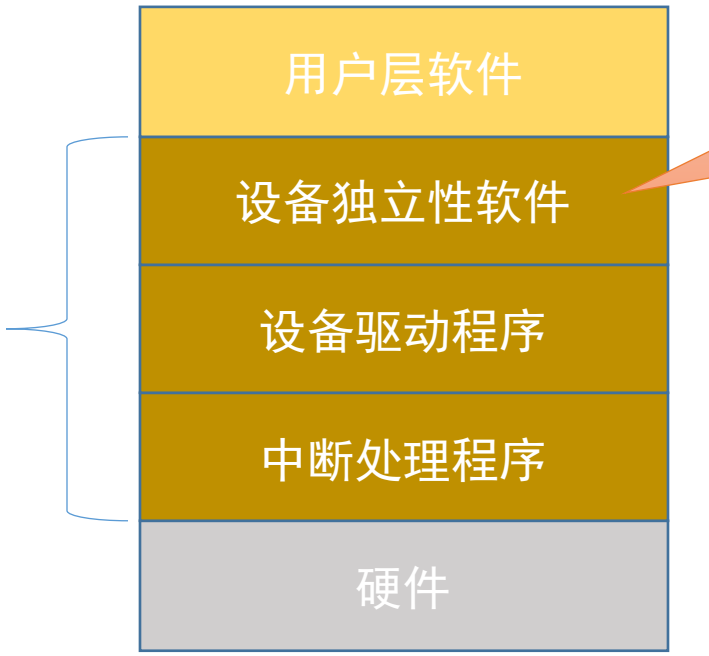

主要实现的功能：
①向上层提供统一的调用接口（如read/write系统调用）

属于操作系统内核部分

# I/O软件层次结构——设备独立性软件



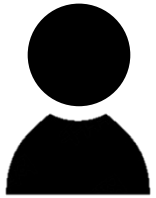
设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。



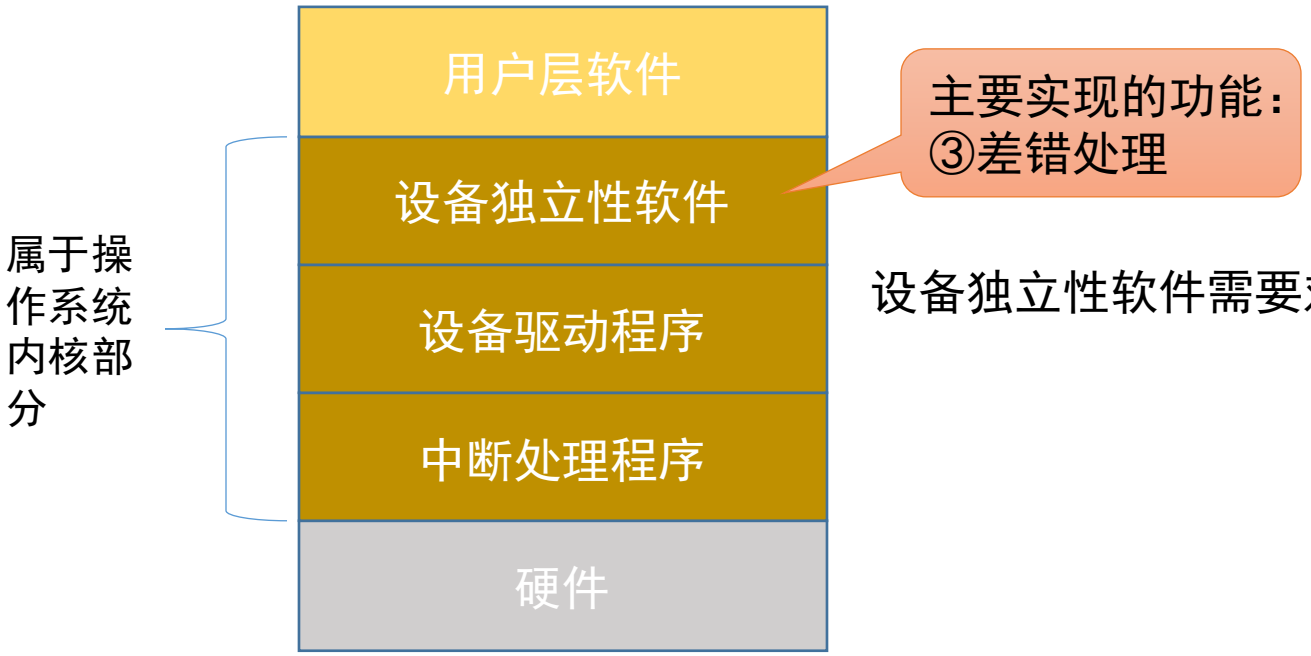

主要实现的功能：
②设备的保护

原理类似与文件保护。设备被看做是一种特殊的文件，不同用户对各个文件的访问权限是不一样的，同理，对设备的访问权限也不一样。

# I/O软件层次结构——设备独立性软件

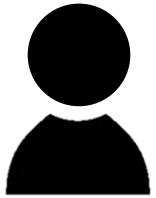


设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。

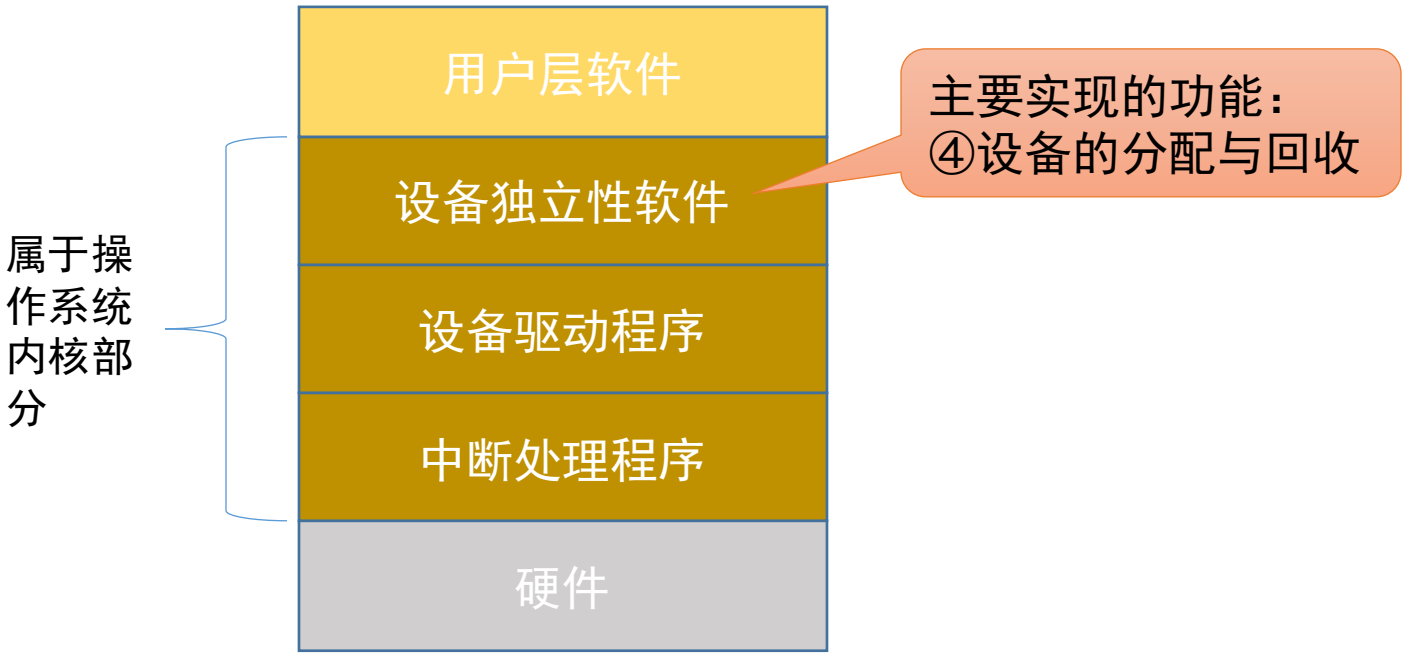



设备独立性软件需要对一些设备的错误进行处理

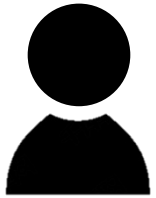
# I/O软件层次结构——设备独立性软件



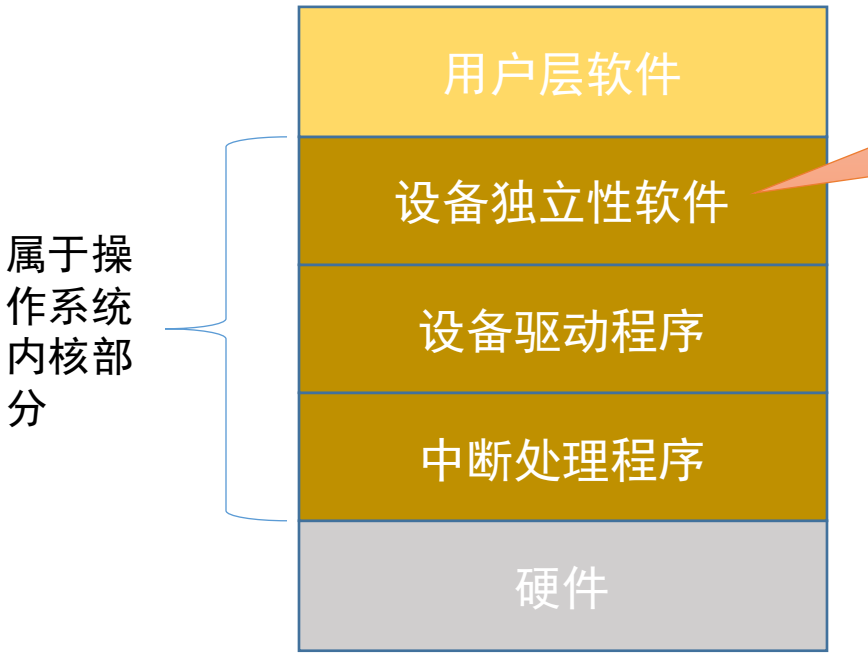
设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。



# I/O软件层次结构——设备独立性软件



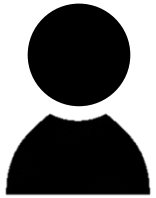
设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。



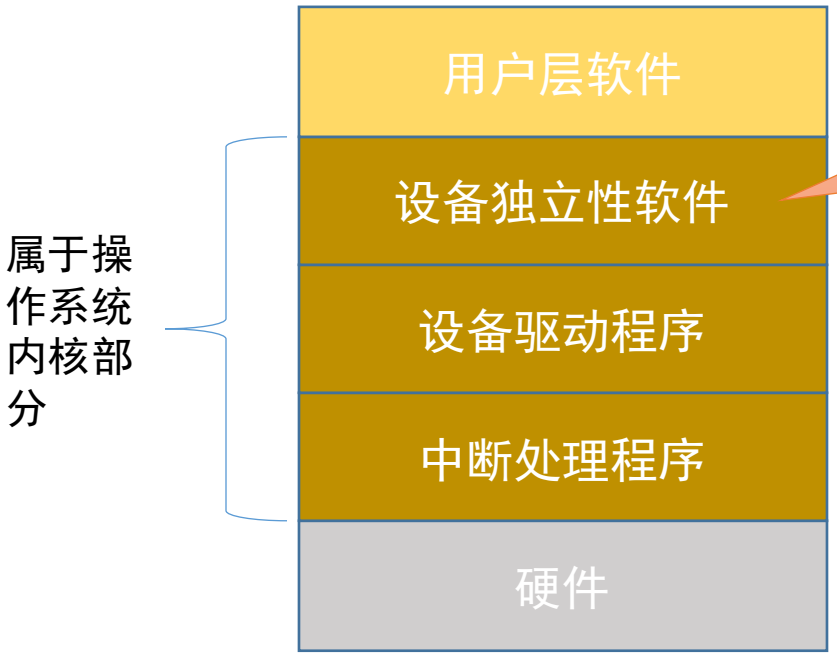

主要实现的功能：
⑤数据缓冲区管理

可以通过缓冲技术屏蔽设备之间数据交换单位大小和传输速度的差异

# I/O软件层次结构——设备独立性软件



设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。




主要实现的功能：
⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

用户或用户层软件发出I/O操作相关系统调用的系统调用时，需要指明此次要操作的I/O设备的逻辑设备名（eg：去学校打印店打印时，需要选择打印机1/打印机2/打印机3，其实这些都是逻辑设备名）

设备独立性软件需要通过“逻辑设备表（LUT，Logical UnitTable）”来确定逻辑设备对应的物理设备，并找到该设备对应的设备驱动程序

# I/O软件层次结构——设备独立性软件

设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现。


属于操作系统内核部分

主要实现的功能：
⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

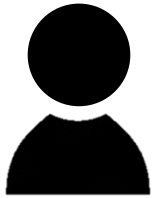
| 逻辑设备名 | 物理设备名 | 驱动程序入口地址 |
|---|---|---|
| /dev/打印机1 | 3 | 1024 |
| /dev/打印机2 | 5 | 2046 |
| …… | …… | …… |

I/O设备被当做一种特殊的文件

不同类型的I/O设备需要有不同的驱动程序处理

操作系统系统可以采用两种方式管理逻辑设备表（LUT）：
第一种方式，整个系统只设置一张LUT，这就意味着所有用户不能使用相同的逻辑设备名，因此这种方式只适用于单用户操作系统。
第二种方式，为每个用户设置一张LUT，各个用户使用的逻辑设备名可以重复，适用于多用户操作系统。系统会在用户登录时为其建立一个用户管理进程，而LUT就存放在用户管理进程的PCB中。

# I/O软件层次结构——设备驱动程序



逻辑设备表（LUT）

| 逻辑设备名 | 物理设备名 | 驱动程序入口地址 |
| --- | --- | --- |
| /dev/打印机1 | 3 | 1024 |
| /dev/打印机2 | 5 | 2046 |
| …… | …… | …… |


属于操作系统内核部分

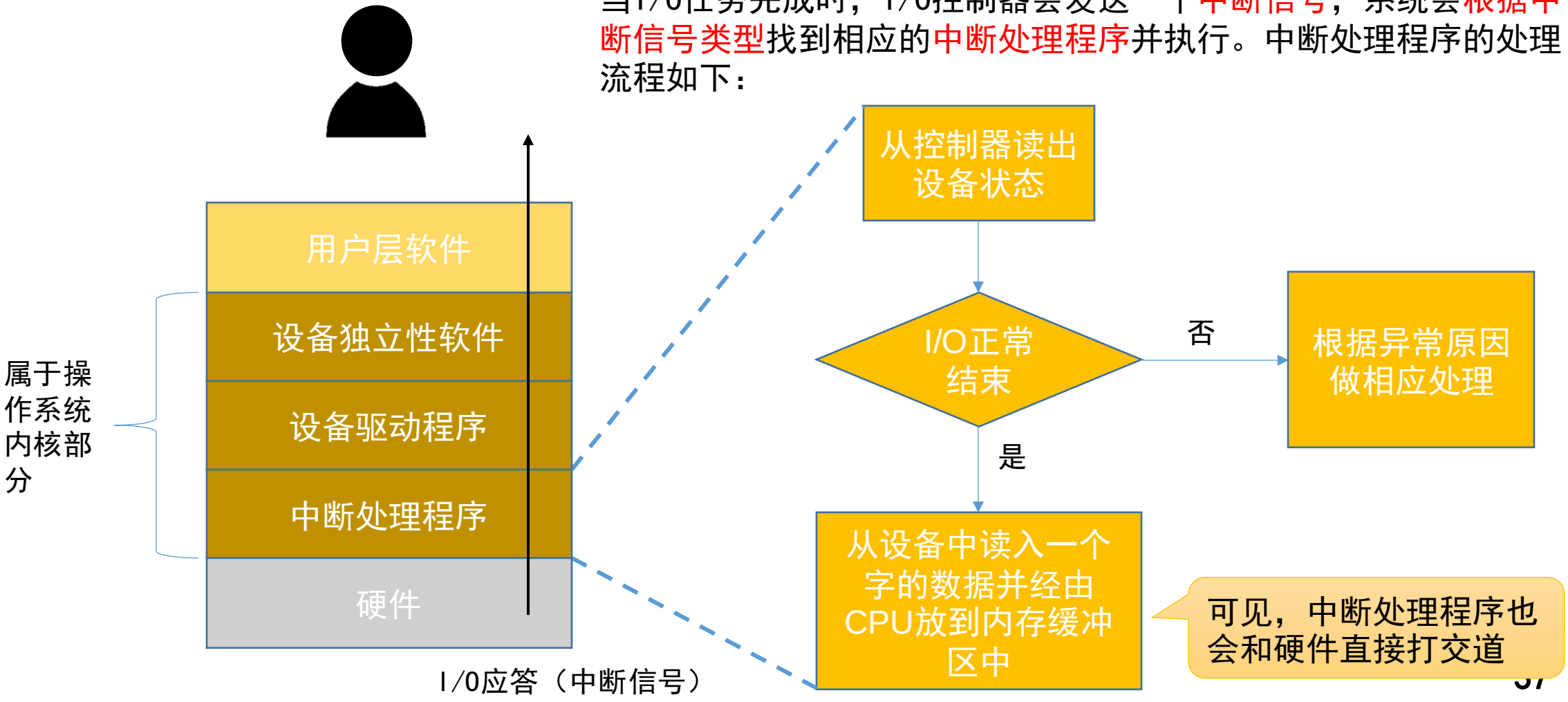
主要负责对硬件设备的具体控制，将上层发出的一系列命令（如read/write）转化成特定设备“能听得懂”的一系列操作。包括设置设备寄存器；检查设备状态等

不同的I/O设备有不同的硬件特性，具体细节只有设备的厂家才知道。因此厂家需要根据设备的硬件特性设计并提供相应的驱动程序。

注：驱动程序一般会以一个独立进程的方式存在。

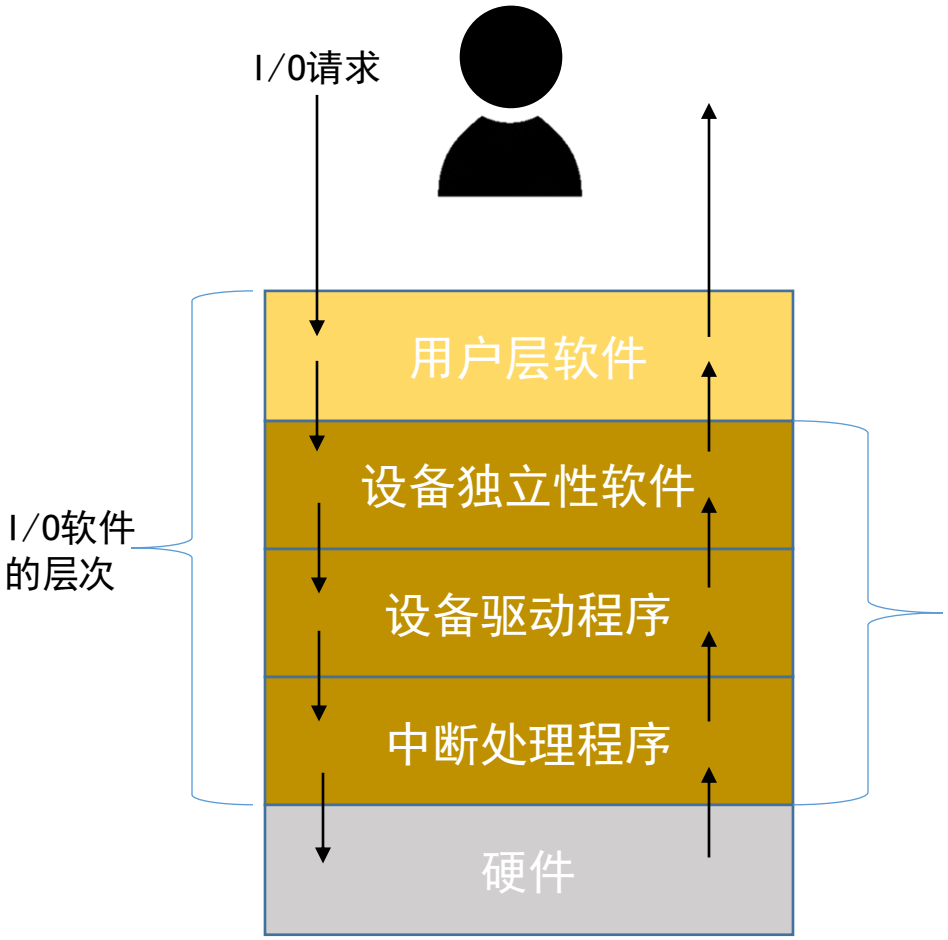
# I/O软件层次结构——中断处理程序

当I/O任务完成时，I/O控制器会发送一个中断信号，系统会根据中断信号类型找到相应的中断处理程序并执行。中断处理程序的处理流程如下：



# IO核心子系统

# I/O核心子系统

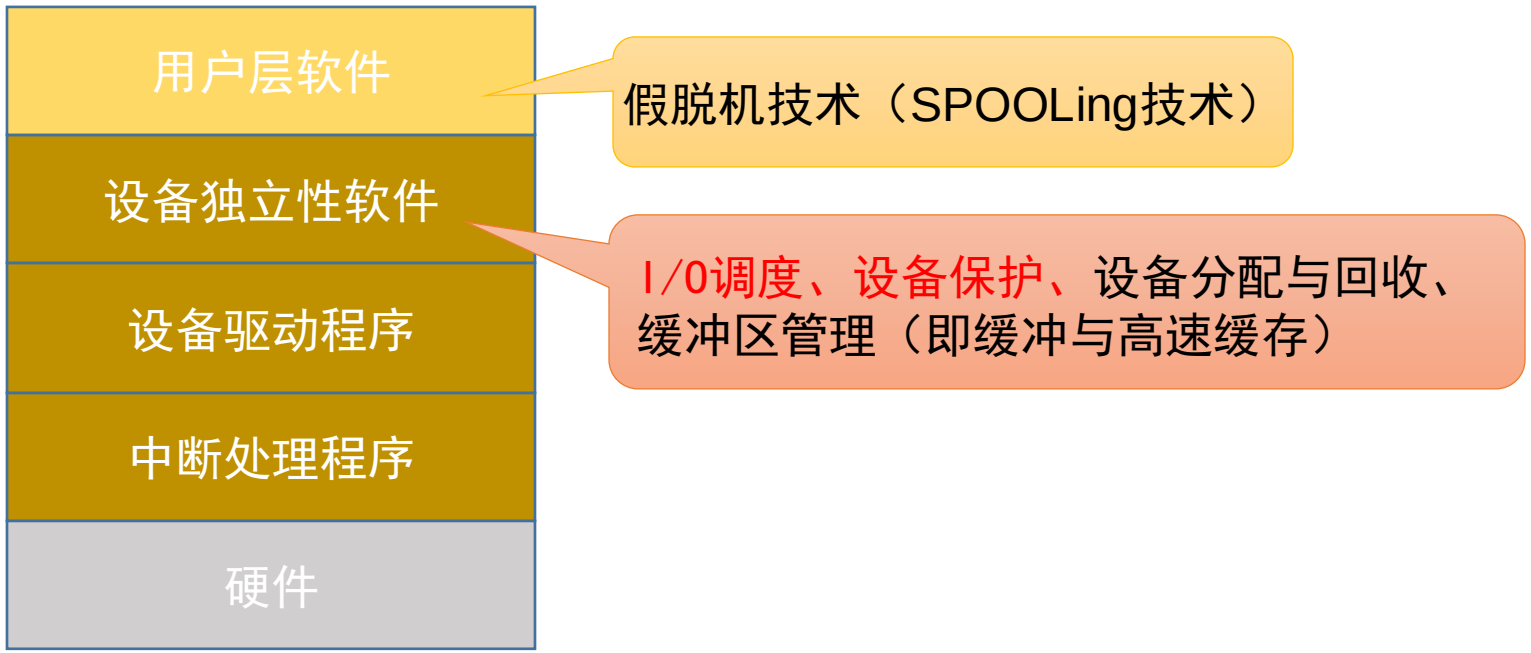



属于操作系统的内核部分；
即“I/O系统”；
或称“I/O核心子系统”

因此I/O核心子系统要实现的功能其实就是中间三层要实现的功能

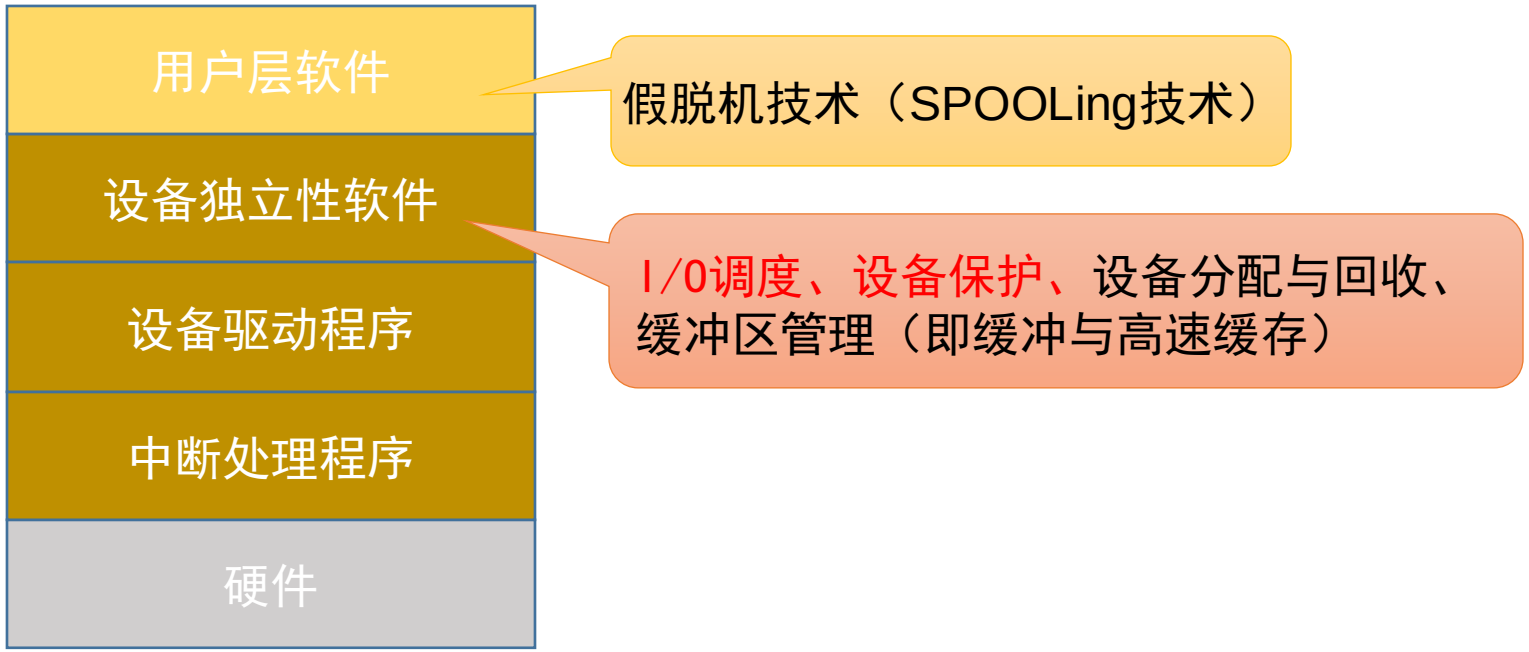
考研中，我们需要重点理解和掌握的功能是：I/O调度、设备保护、假脱机技术（SPOOLing技术）、设备分配与回收、缓冲区管理（即缓冲与高速缓存）

# I/O核心子系统——功能在哪个层次实现




注：假脱机技术（SPOOLing技术）需要请求“磁盘设备”的设备独立性软件的服务，因此一般来说假脱机技术是在用户层软件实现的。但是408大纲又将假脱机技术归为“I/O核心子系统”的功能，因此考试时还是以大纲为准。

# I/O核心子系统——I/O调度

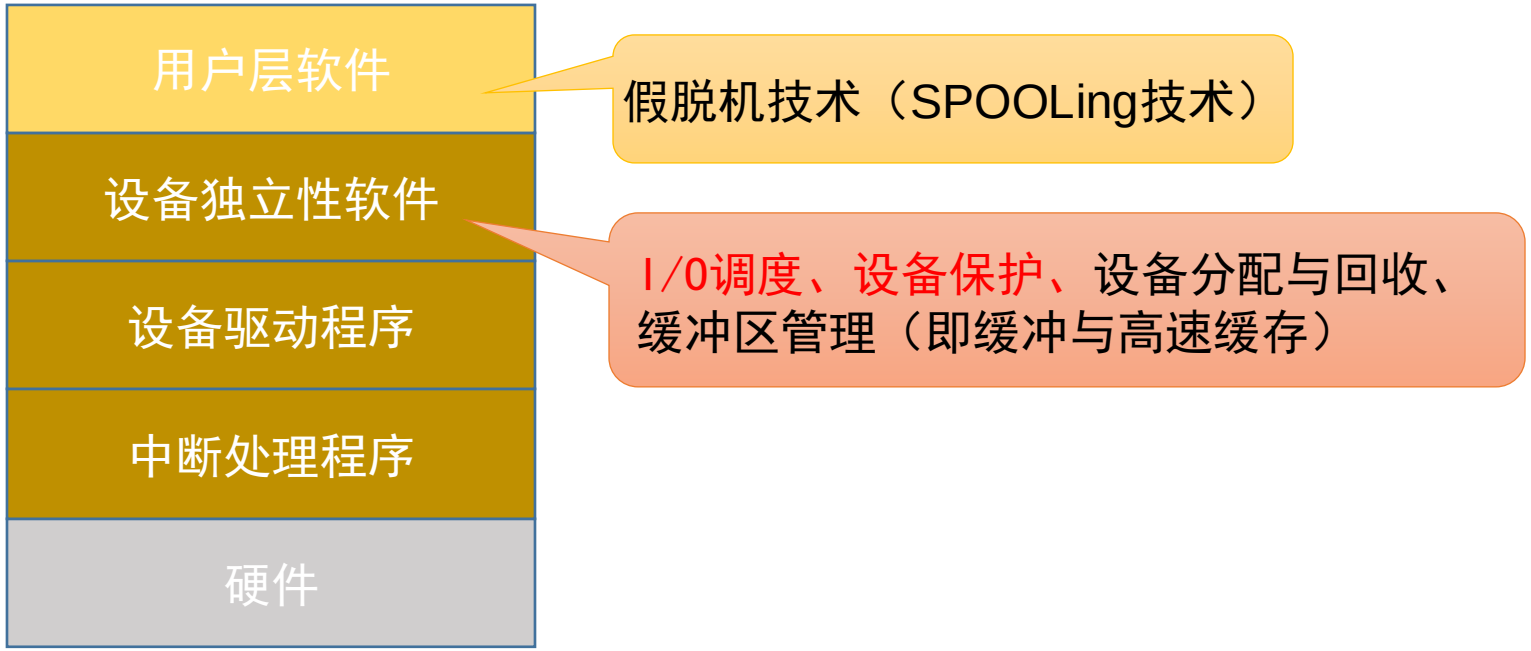



I/O调度：用某种算法确定一个好的顺序来处理各个I/O请求。

如：磁盘调度（先来先服务算法、最短寻道优先算法、SCAN算法、C-SCAN算法、LOOK算法、C-LOOK算法）。当多个磁盘I/O请求到来时，用某种调度算法确定满足I/O请求的顺序。

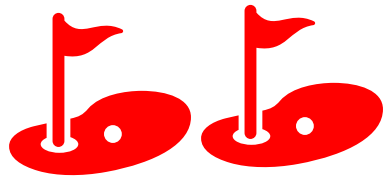
同理，打印机等设备也可以用先来先服务算法、优先级算法、短作业优先等算法来确定I/O调度顺序。

# I/O核心子系统——设备保护




操作系统需要实现文件保护功能，不同的用户对各个文件有不同的访问权限（如：只读、读和写等）。

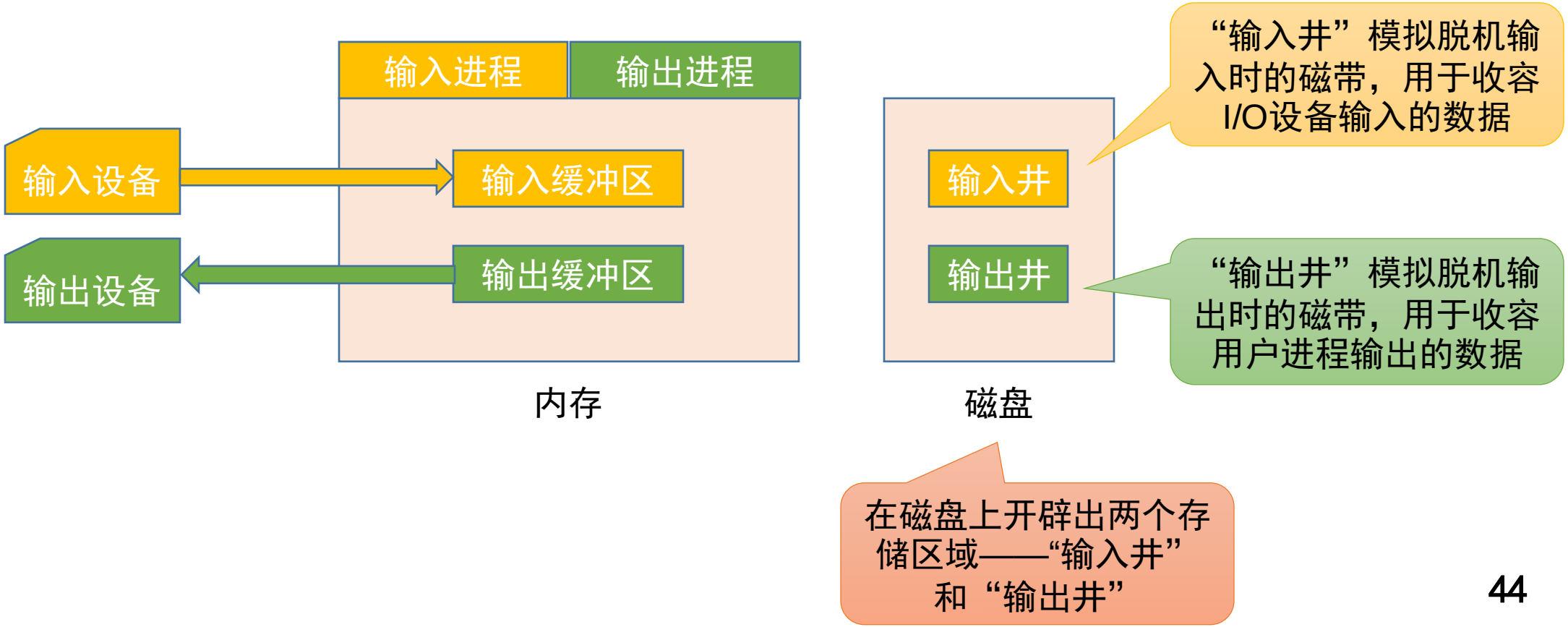
在UNIX系统中，设备被看做是一种特殊的文件，每个设备也会有对应的FCB。当用户请求访问某个设备时，系统根据FCB中记录的信息来判断该用户是否有相应的访问权限，以此实现“设备保护”的功能。（参考“文件保护”小节）



# SPOOLing技术

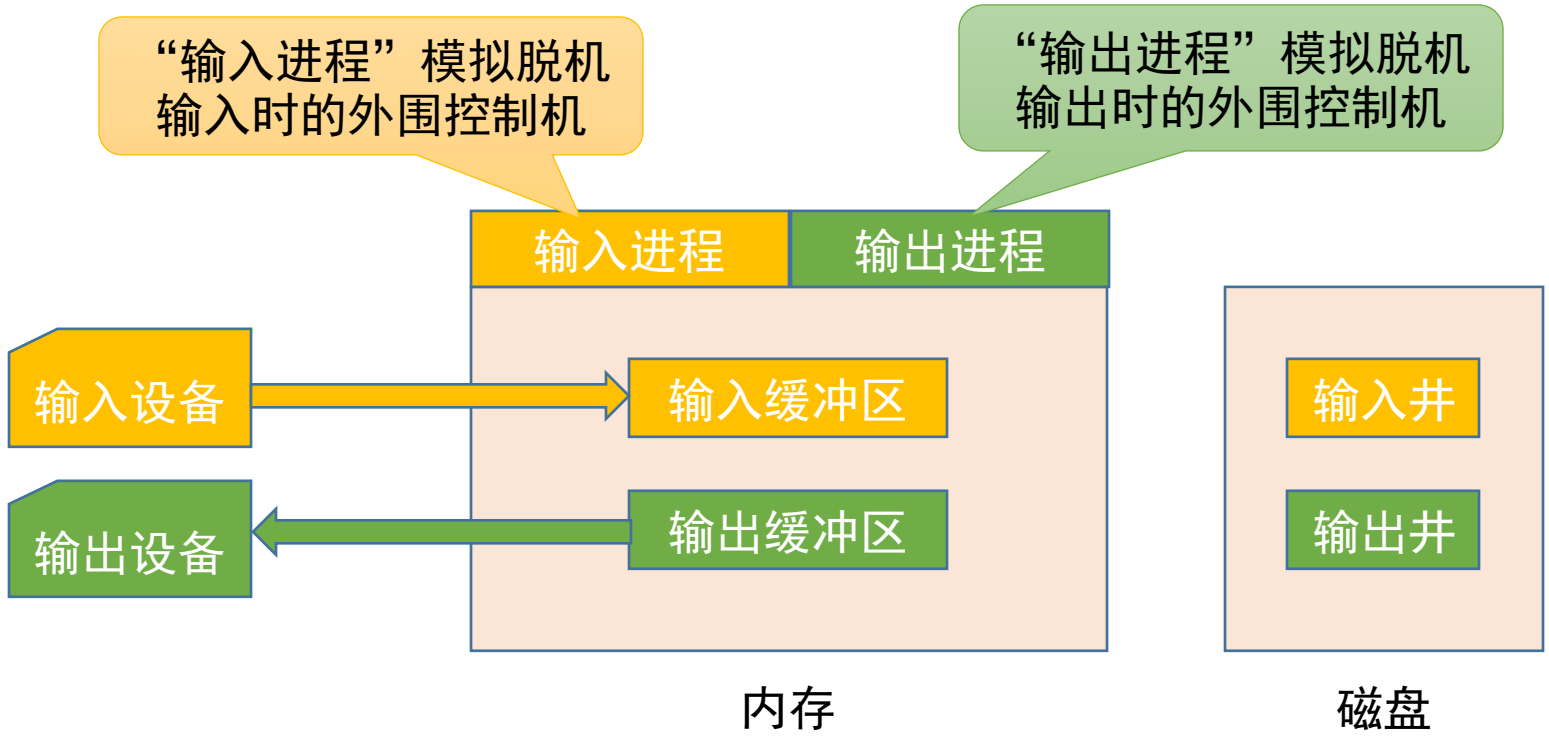
# 假脱机技术（SPOOLing技术）——输入井和输出井

“假脱机技术”，又称“SPOOLing技术”是用软件的方式模拟脱机技术。SPOOLing系统的组成如下：



# 假脱机技术（SPOOLing技术）——输入进程和输出进程

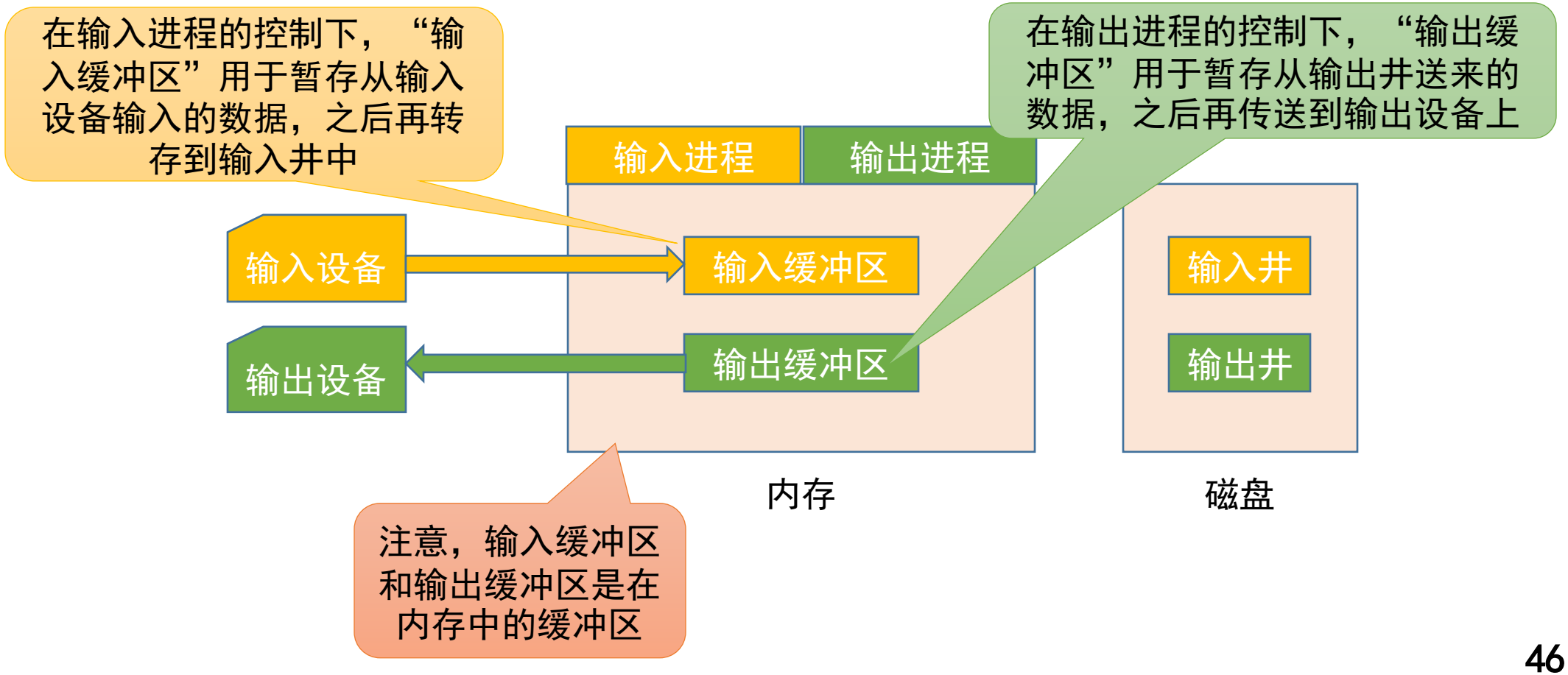
“假脱机技术”，又称“SPOOLing技术”是用软件的方式模拟脱机技术。SPOOLing系统的组成如下：




要实现SPOOLing技术，必须要有多道程序技术的支持。系统会建立“输入进程”和“输出进程”。

# 假脱机技术（SPOOLing技术）——输入/输出缓冲区

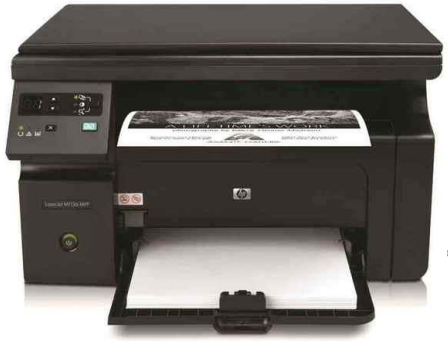
“假脱机技术”，又称“SPOOLing技术”是用软件的方式模拟脱机技术。SPOOLing系统的组成如下：



# 假脱机技术（SPOOLing技术）——共享打印机原理分析

独占式设备——只允许各个进程串行使用的设备。一段时间内只能满足一个进程的请求。

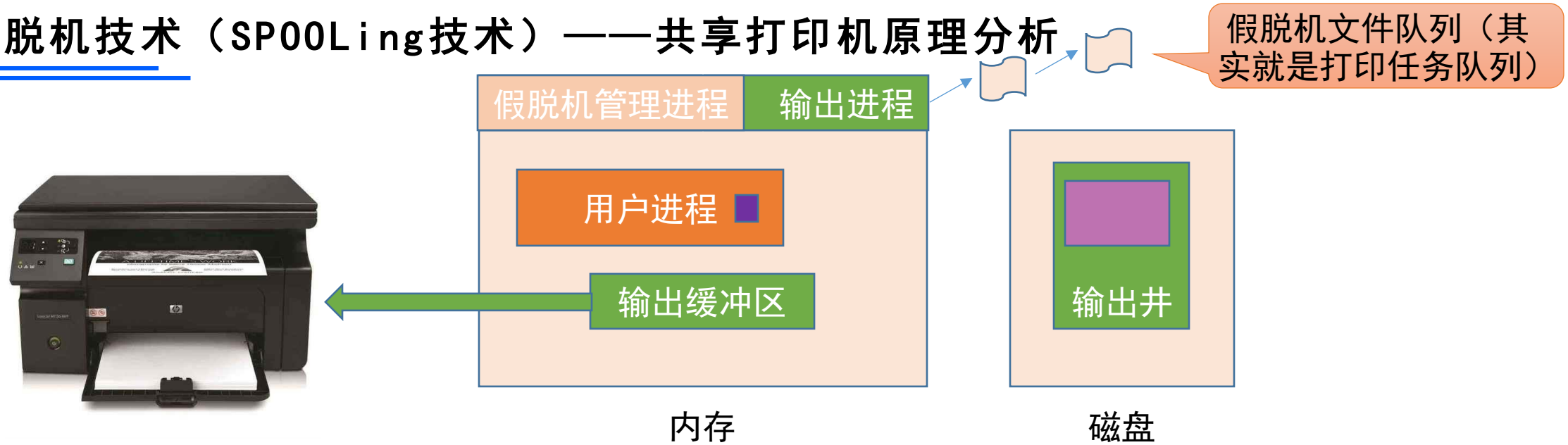
共享设备——允许多个进程“同时”使用的设备（宏观上同时使用，微观上可能是交替使用）。可以同时满足多个进程的使用请求。



打印机是种“独占式设备”，但是可以用SPOOLing技术改造成“共享设备”

独占式设备的例子：若进程1正在使用打印机，则进程2请求使用打印机时必然阻塞等待

# 假脱机技术（SPOOLing技术）——共享打印机原理分析



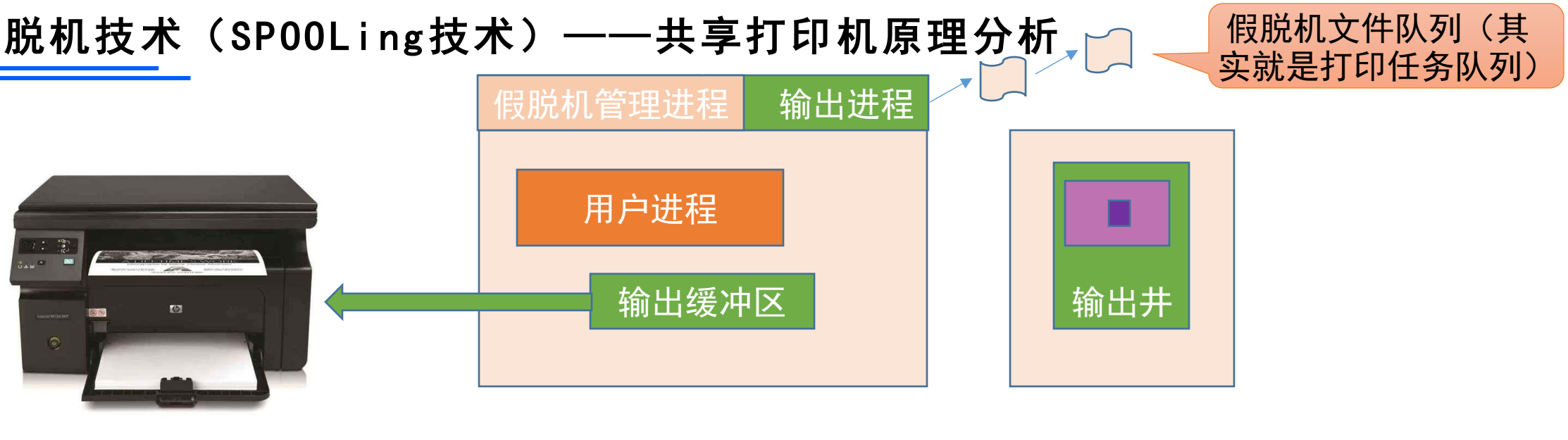

当多个用户进程提出输出打印的请求时，系统会答应它们的请求，但是并不是真正把打印机分配给他们，而是由假脱机管理进程为每个进程做两件事：

（1）在磁盘输出井中为进程申请一个空闲缓冲区（也就是说，这个缓冲区是在磁盘上的），并将要打印的数据送入其中；

（2）为用户进程申请一张空白的打印请求表，并将用户的打印请求填入表中（其实就是用来说明用户的打印数据存放位置等信息的），再将该表挂到假脱机文件队列上。

当打印机空闲时，输出进程会从文件队列的队头取出一张打印请求表，并根据表中的要求将要打印的数据从输出井传送到输出缓冲区，再输出到打印机进行打印。用这种方式可依次处理完全部的打印任务

# 假脱机技术（SPOOLing技术）——共享打印机原理分析

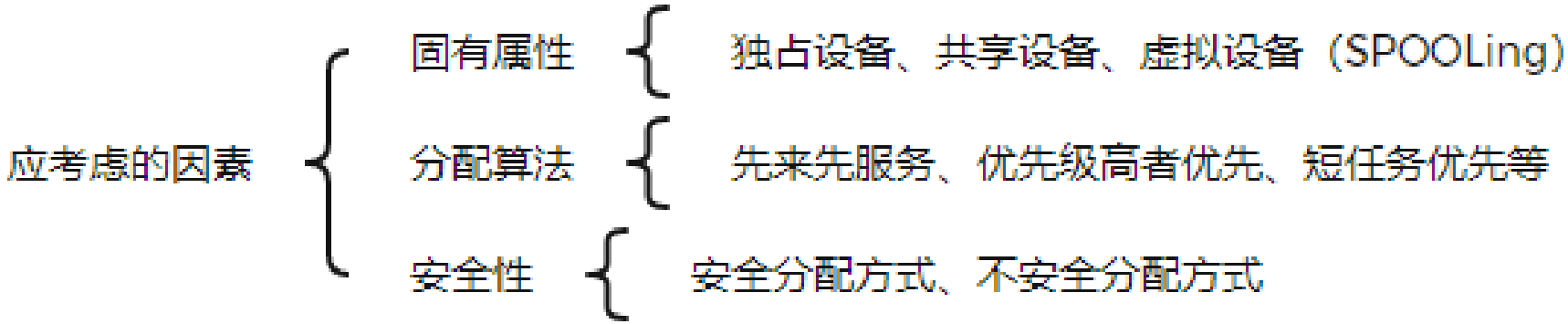



虽然系统中只有一个台打印机，但每个进程提出打印请求时，系统都会为在输出井中为其分配一个存储区（相当于分配了一个逻辑设备），使每个用户进程都觉得自己在独占一台打印机，从而实现对打印机的共享。

SPOOLing技术可以把一台物理设备虚拟成逻辑上的多台设备，可将独占式设备改造成共享设备。

# IO设备的分配与回收

# 设备的分配与回收——设备分配时应考虑的因素



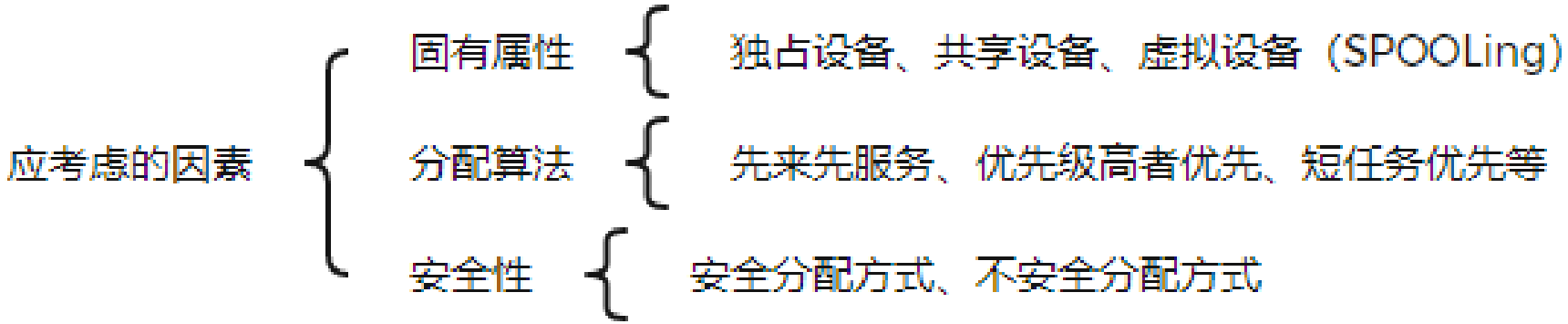

设备的固有属性可分为三种：独占设备、共享设备、虚拟设备。

独占设备——一个时段只能分配给一个进程（如打印机）

共享设备——可同时分配给多个进程使用（如磁盘），各进程往往是宏观上同时共享使用设备，而微观上交替使用。

虚拟设备——采用SPOOLing技术将独占设备改造成虚拟的共享设备，可同时分配给多个进程使用（如采用SPOOLing技术实现的共享打印机）

# 设备的分配与回收——设备分配时应考虑的因素




设备的分配算法：

先来先服务
优先级高者优先
短任务优先
……

# 设备的分配与回收——设备分配时应考虑的因素

- 应考虑的因素
  - 固有属性：独占设备、共享设备、虚拟设备（SPOOLing）
  - 分配算法：先来先服务、优先级高者优先、短任务优先等
  - 安全性：安全分配方式、不安全分配方式

从进程运行的安全性上考虑，设备分配有两种方式：

安全分配方式：为进程分配一个设备后就将进程阻塞，本次I/O完成后才将进程唤醒。（eg：考虑进程请求打印机打印输出的例子）

一个时段内每个进程只能使用一个设备
优点：破坏了“请求和保持”条件，不会死锁
缺点：对于一个进程来说，CPU和I/O设备只能串行工作

不安全分配方式：进程发出I/O请求后，系统为其分配I/O设备，进程可继续执行，之后还可以发出新的I/O请求。只有某个I/O请求得不到满足时才将进程阻塞。

一个进程可以同时使用多个设备
优点：进程的计算任务和I/O任务可以并行处理，使进程迅速推进
缺点：有可能发生死锁（死锁避免、死锁的检测和解除）

# 设备的分配与回收——静态分配和动态分配

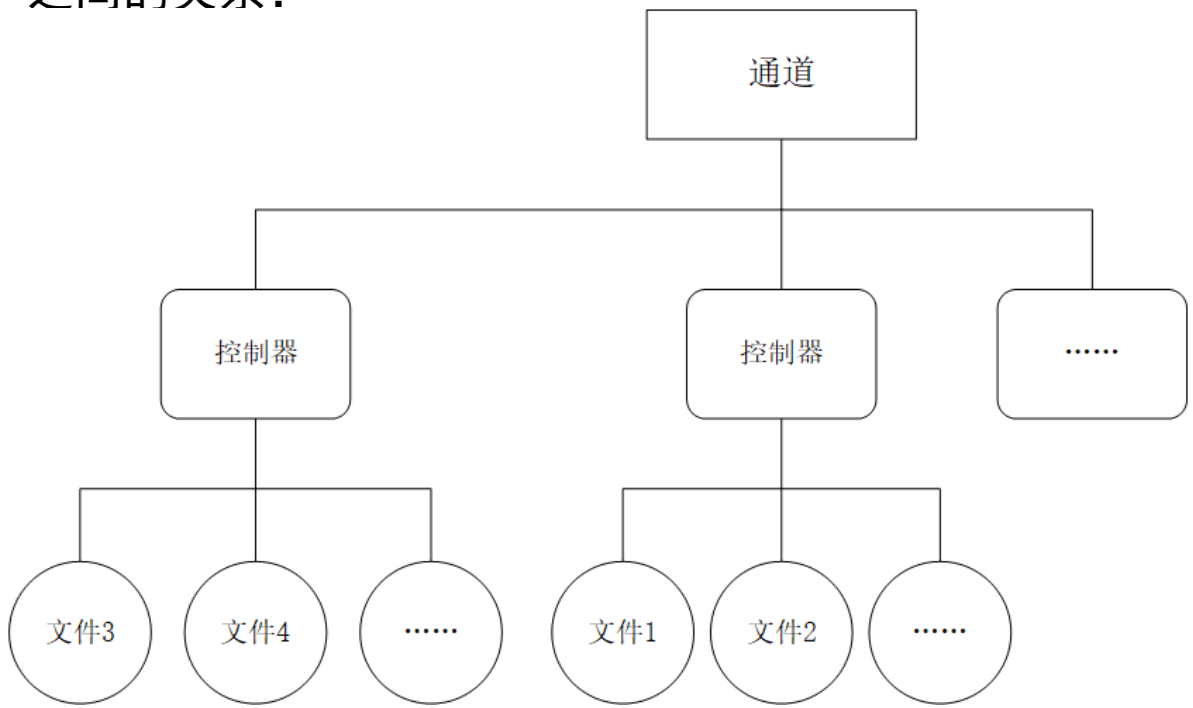
静态分配：进程运行前为其分配全部所需资源，运行结束后归还资源

动态分配：进程运行过程中动态申请设备资源

破坏了“请求和保持”条件，不会发生死锁

# 设备的分配与回收——设备分配管理中的数据结构

“设备、控制器、通道”之间的关系：




一个通道可控制多个设备控制器，每个设备控制器可控制多个设备。

# 设备的分配与回收——设备分配管理中的数据结构

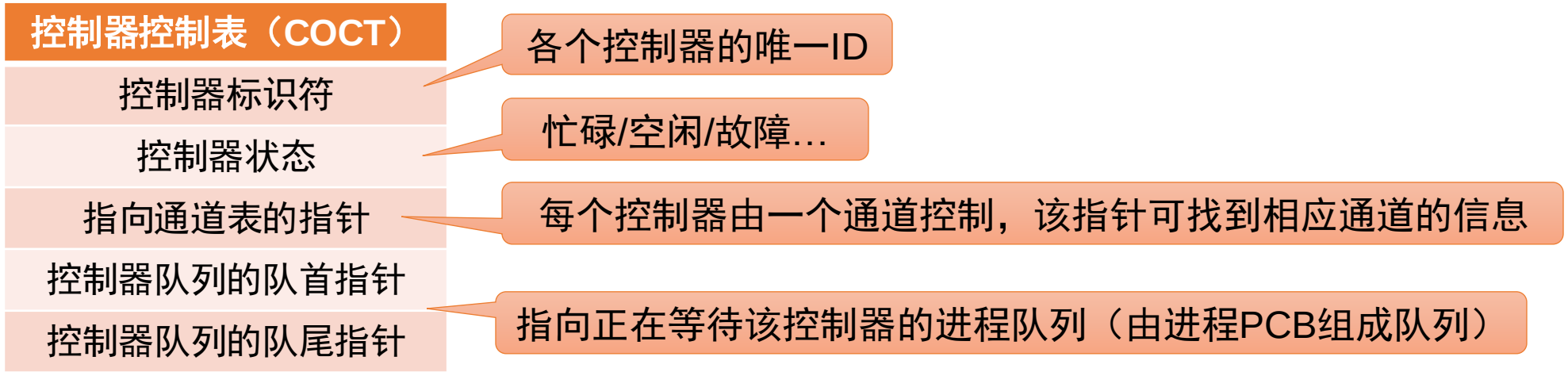
设备控制表（DCT）：系统为每个设备配置一张DCT，用于记录设备情况

| 设备控制表（DCT） | |
|---|---|
| 设备类型 | 如：打印机/扫描仪/键盘 |
| 设备标识符 | 即物理设备名，系统中的每个设备的物理设备名唯一 |
| 设备状态 | 忙碌/空闲/故障... |
| 指向控制器表的指针 | 每个设备由一个控制器控制，该指针可找到相应控制器的信息 |
| 重复执行次数或时间 | 当重复执行多次I/O操作后仍不成功，才认为此次I/O失败 |
| 设备队列的队首指针 | 指向正在等待该设备的进程队列（由进程PCB组成队列） |

注：“进程管理”章节中曾经提到过“系统会根据阻塞原因不同，将进程PCB挂到不同的阻塞队列中”

# 设备的分配与回收——设备分配管理中的数据结构

控制器控制表（COCT）：每个设备控制器都会对应一张COCT。操作系统根据COCT的信息对控制器进行操作和管理。



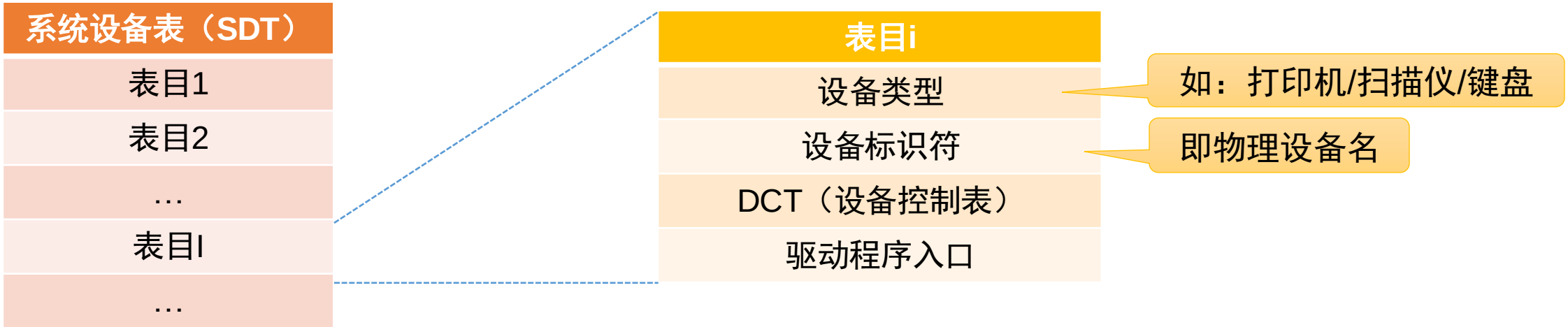
# 设备的分配与回收——设备分配管理中的数据结构

通道控制表（CHCT）：每个通道都会对应一张CHCT。操作系统根据CHCT的信息对通道进行操作和管理。

| 通道控制表（CHCT） | |
|---|---|
| 通道标识符 | 各个通道的唯一ID |
| 通道状态 | 忙碌/空闲/故障... |
| 与通道连接的控制器表首址 | 可通过该指针找到该通道管理的所有控制器相关信息（COCT） |
| 通道队列的队首指针 | 指向正在等待该通道的进程队列（由进程PCB组成队列） |
| 通道队列的队尾指针 | |

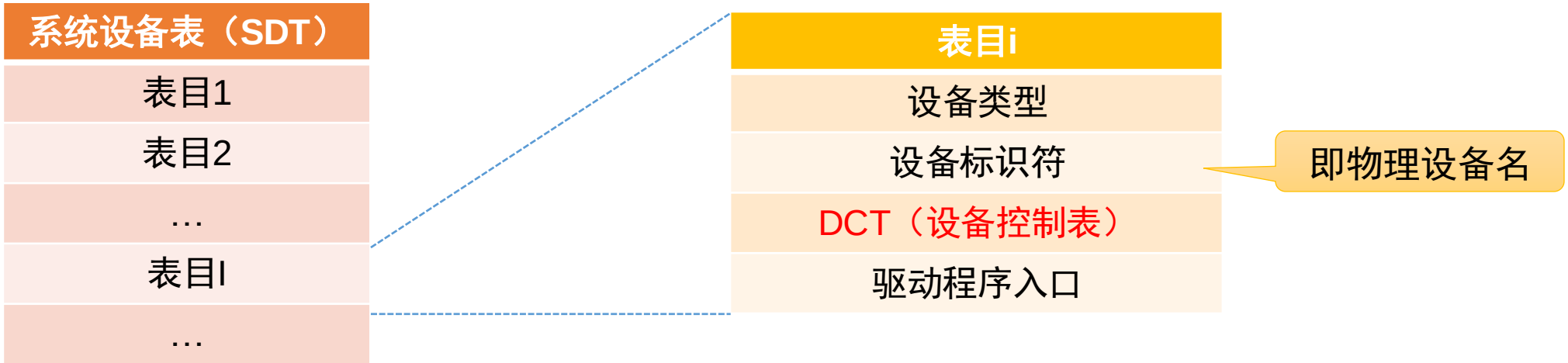
# 设备的分配与回收——设备分配管理中的数据结构

系统设备表（SDT）：记录了系统中全部设备的情况，每个设备对应一个表目。



# 设备的分配与回收——设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）



# 设备的分配与回收——设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。

| 设备控制表（DCT） |
| --- |
| 设备类型 |
| 设备标识符 |
| 设备状态 |
| 指向控制器表的指针 |
| 重复执行次数或时间 |
| 设备队列的队首指针 |

忙碌/空闲/故障...

每个设备由一个控制器控制，该指针可找到相应控制器的信息

指向正在等待该设备的进程队列（由进程PCB组成队列）

# 设备的分配与回收——设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。

| 控制器控制表（COCT） |
| --- |
| 控制器标识符 |
| 控制器状态 |
| 指向通道表的指针 |
| 控制器队列的队首指针 |
| 控制器队列的队尾指针 |

忙碌/空闲/故障...

每个控制器由一个通道控制，该指针可找到相应通道的信息

指向正在等待该控制器的进程队列（由进程PCB组成队列）

# 设备的分配与回收——设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

| 通道控制表（CHCT） |
| --- |
| 通道标识符 |
| 通道状态 |
| 与通道连接的控制器表首址 |
| 通道队列的队首指针 |
| 通道队列的队尾指针 |

忙碌/空闲/故障...

指向正在等待该通道的进程队列（由进程PCB组成队列）

注：只有设备、控制器、通道三者都分配成功时，这次设备分配才算成功，之后便可启动I/O设备进行数据传送

# 设备的分配与回收——设备分配步骤的改进

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

缺点：
①用户编程时必须使用“物理设备名”，底层细节对用户不透明，不方便编程
②若换了一个物理设备，则程序无法运行
③若进程请求的物理设备正在忙碌，则即使系统中还有同类型的设备，进程也必须阻塞等待

改进方法：建立逻辑设备名与物理设备名的映射机制，用户编程时只需提供逻辑设备名

# 设备的分配与回收——设备分配步骤的改进

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

| 系统设备表（SDT） |
| --- |
| 表目1 |
| 表目2 |
| … |
| 表目I |
| … |

即逻辑设备名

| 表目i |
| --- |
| 设备类型 |
| 设备标识符 |
| DCT（设备控制表） |
| 驱动程序入口 |

逻辑设备表（LUT）

| 逻辑设备名 | 物理设备名 | 驱动程序入口地址 |
| --- | --- | --- |
| /dev/打印机1 | 3 | 1024 |
| /dev/打印机2 | 5 | 2046 |
| …… | …… | …… |

# 设备的分配与回收——设备分配步骤的改进

逻辑设备表（LUT）

| 逻辑设备名 | 物理设备名 | 驱动程序入口地址 |
|---|---|---|
| /dev/printer | 3 | 1024 |
| /dev/tty | 5 | 2046 |
| …… | …… | …… |

逻辑设备表（LUT）建立了逻辑设备名与物理设备名之间的映射关系。

某用户进程第一次使用设备时使用逻辑设备名向操作系统发出请求，操作系统根据用户进程指定的设备类型（逻辑设备名）查找系统设备表，找到一个空闲设备分配给进程，并在LUT中增加相应表项。

如果之后用户进程再次通过相同的逻辑设备名请求使用设备，则操作系统通过LUT表即可知道用户进程实际要使用的是哪个物理设备了，并且也能知道该设备的驱动程序入口地址。

逻辑设备表的设置问题：
整个系统只有一张LUT：各用户所用的逻辑设备名不允许重复，适用于单用户操作系统
每个用户一张LUT：不同用户的逻辑设备名可重复，适用于多用户操作系统

# 缓冲区管理

# 缓冲区管理——什么是缓冲区？有什么作用？

缓冲区是一个存储区域，可以由专门的硬件寄存器组成，也可利用内存作为缓冲区。

使用硬件作为缓冲区的成本较高，容量也较小，一般仅用在对速度要求非常高的场合（如存储器管理中所用的联想寄存器，由于对页表的访问频率极高，因此使用速度很快的联想寄存器来存放页表项的副本）

一般情况下，更多的是利用内存作为缓冲区，“设备独立性软件”的缓冲区管理就是要组织管理好这些缓冲区

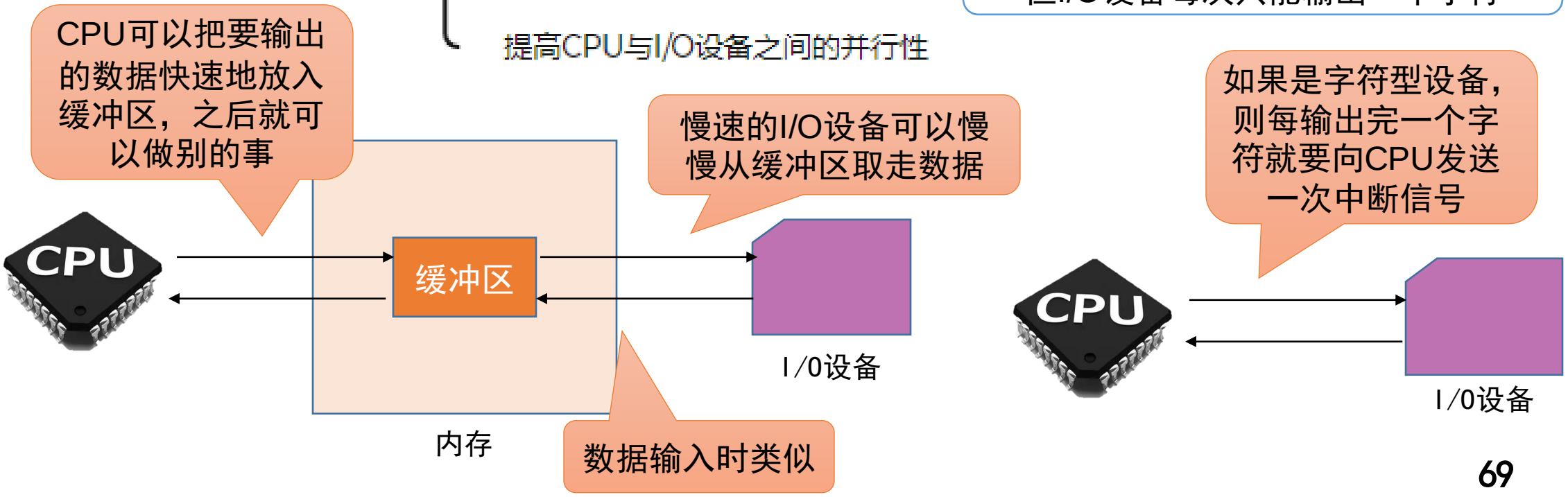
本节介绍的是“内存作为缓冲区”

缓冲区的作用
- 缓和CPU与I/O设备之间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
- 解决数据粒度不匹配的问题
- 提高CPU与I/O设备之间的并行性

# 缓冲区管理——缓冲区有什么作用？

缓冲区的作用
- 缓和CPU与I/O设备之间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
- 解决数据粒度不匹配的问题
- 提高CPU与I/O设备之间的并行性

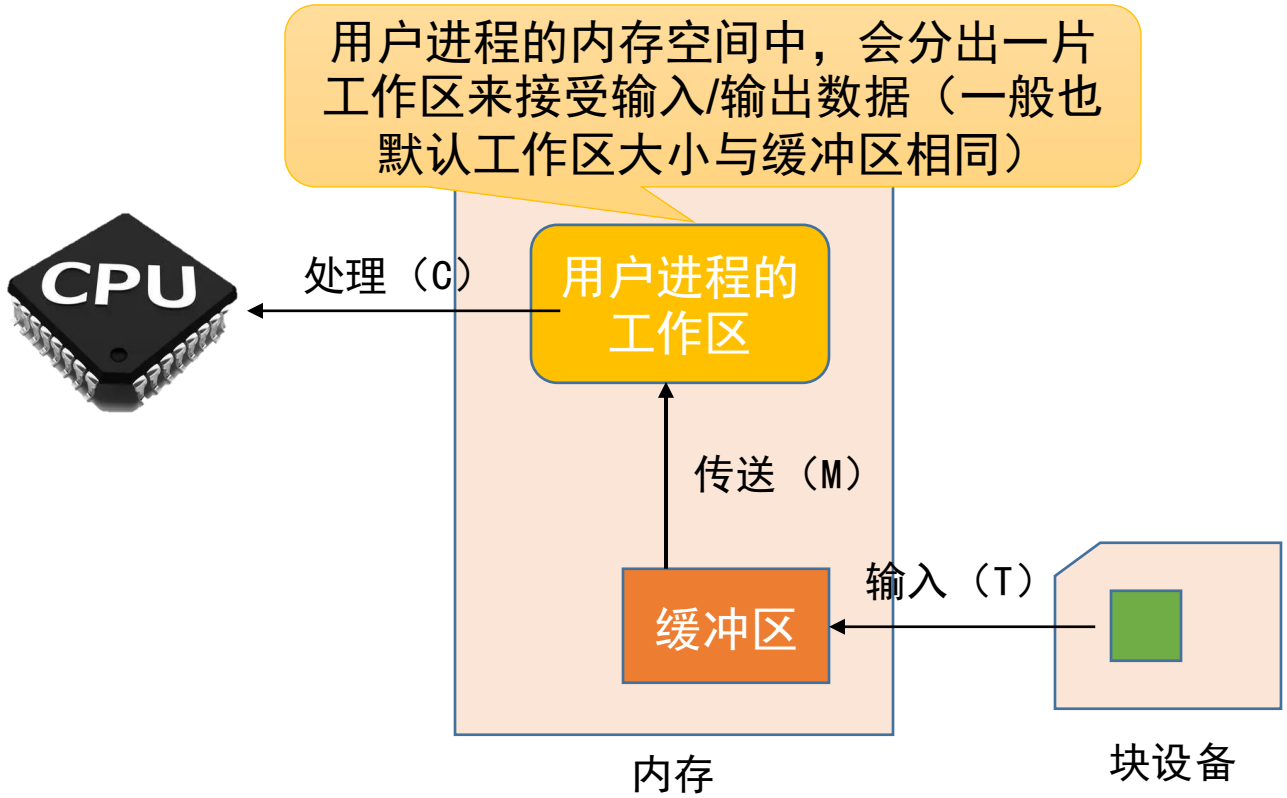
如：输出进程每次可以生成一块数据，但I/O设备每次只能输出一个字符



# 缓冲区管理——单缓冲

假设某用户进程请求某种块设备读入若干块的数据。若采用单缓冲的策略，操作系统会在主存中为其分配一个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）。

注意：当缓冲区数据非空时，不能往缓冲区冲入数据，只能从缓冲区把数据传出；当缓冲区为空时，可以往缓冲区冲入数据，但必须把缓冲区充满以后，才能从缓冲区把数据传出。




常考题型：计算每处理一块数据平均需要多久？

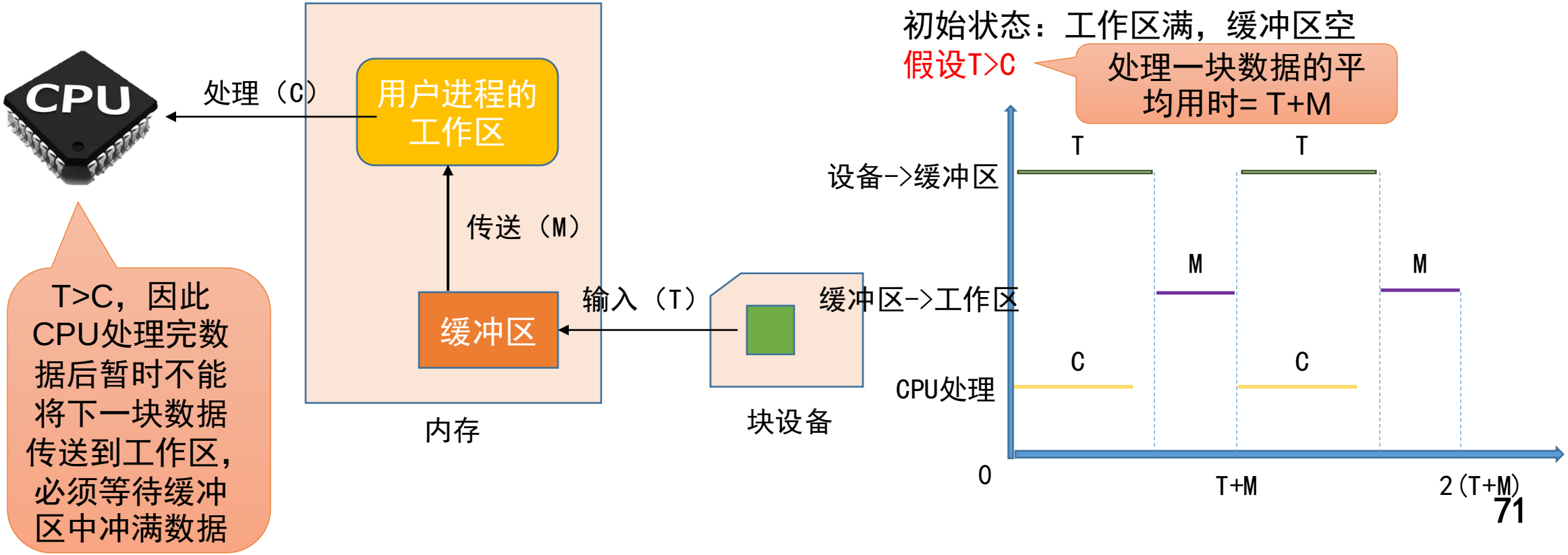
技巧：假定一个初始状态，分析下次到达相同状态需要多少时间，这就是处理一块数据平均所需时间。

在“单缓冲”题型中，可以假设初始状态为工作区满，缓冲区空。

# 缓冲区管理——单缓冲

假设某用户进程请求某种块设备读入若干块的数据。若采用单缓冲的策略，操作系统会在主存中为其分配一个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）。

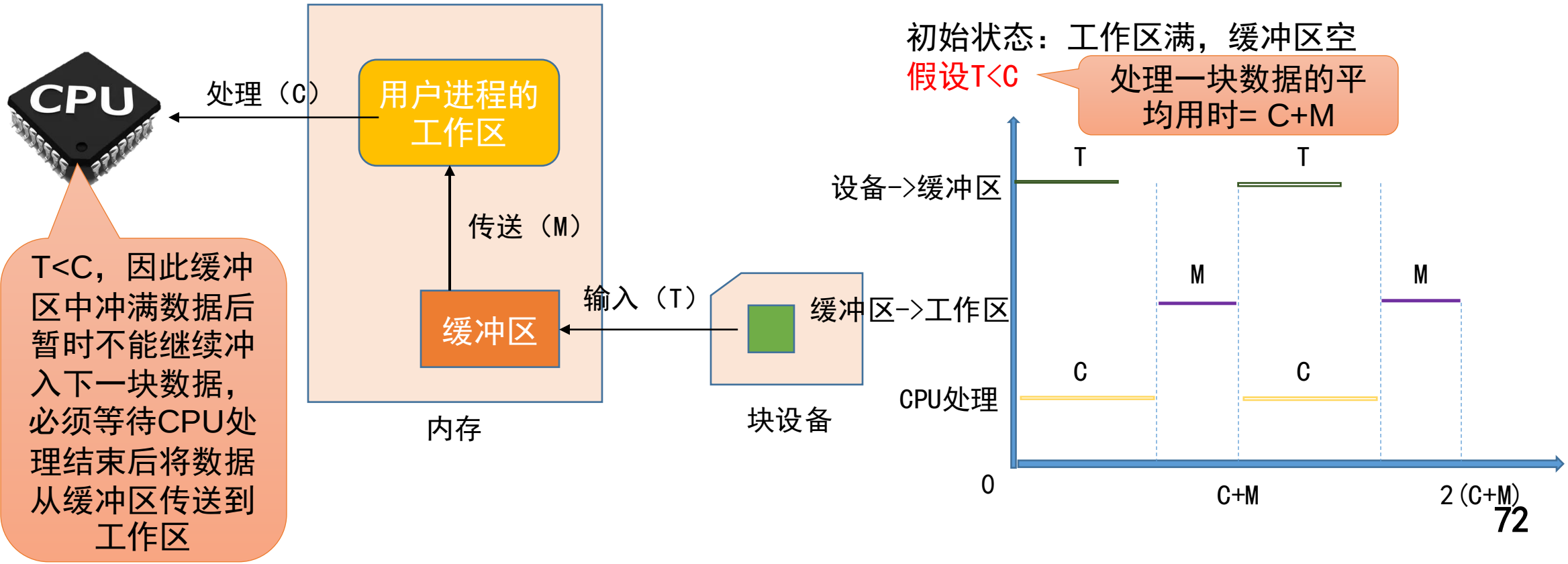
注意：当缓冲区数据非空时，不能往缓冲区冲入数据，只能从缓冲区把数据传出；当缓冲区为空时，可以往缓冲区冲入数据，但必须把缓冲区充满以后，才能从缓冲区把数据传出。



# 缓冲区管理——单缓冲

假设某用户进程请求某种块设备读入若干块的数据。若采用单缓冲的策略，操作系统会在主存中为其分配一个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）。

注意：当缓冲区数据非空时，不能往缓冲区冲入数据，只能从缓冲区把数据传出；当缓冲区为空时，可以往缓冲区冲入数据，但必须把缓冲区充满以后，才能从缓冲区把数据传出。



# 缓冲区管理——单缓冲

假设某用户进程请求某种块设备读入若干块的数据。若采用单缓冲的策略，操作系统会在主存中为其分配一个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）。
注意：当缓冲区数据非空时，不能往缓冲区冲入数据，只能从缓冲区把数据传出；当缓冲区为空时，可以往缓冲区冲入数据，但必须把缓冲区充满以后，才能从缓冲区把数据传出。

结论：采用单缓冲策略，处理一块数据平均耗时Max(C, T)+M

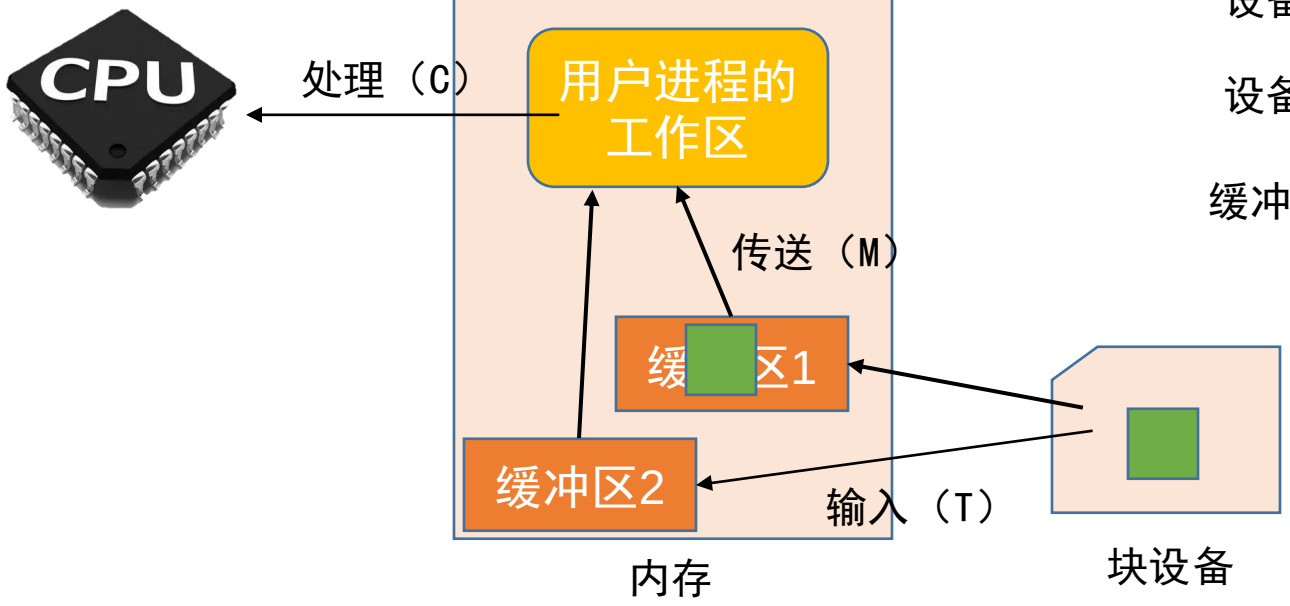
# 缓冲区管理——双缓冲

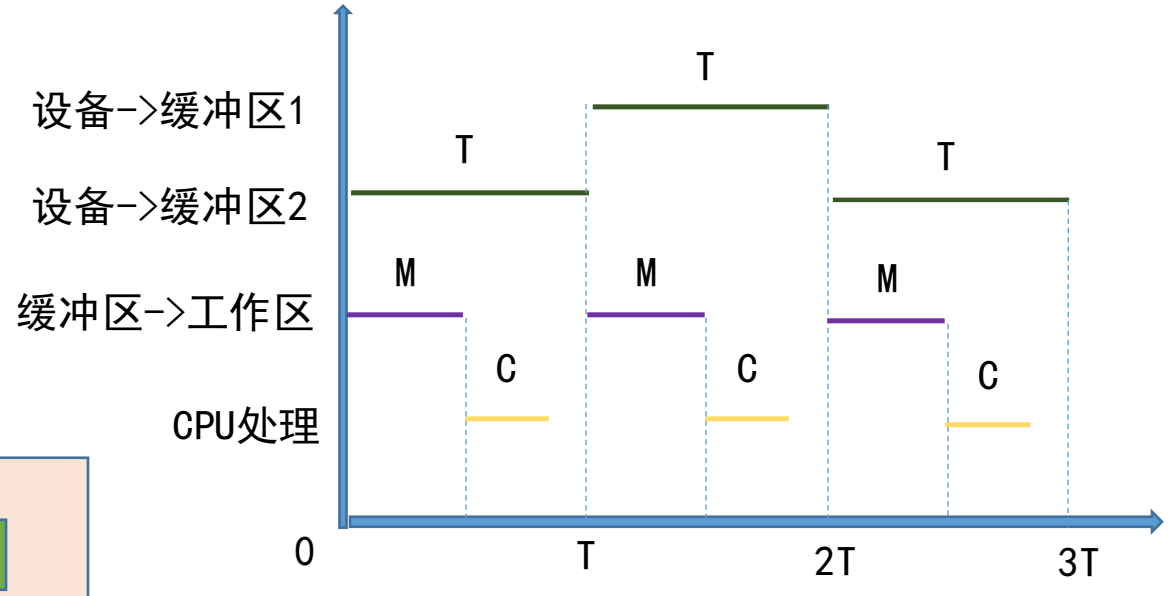
假设某用户进程请求某种块设备读入若干块的数据。若采用双缓冲的策略，操作系统会在主存中为其分配两个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）

双缓冲题目中，假设初始状态为：工作区空，其中一个缓冲区满，另一个缓冲区空

假设T>C+M

处理一块数据的平均用时= T



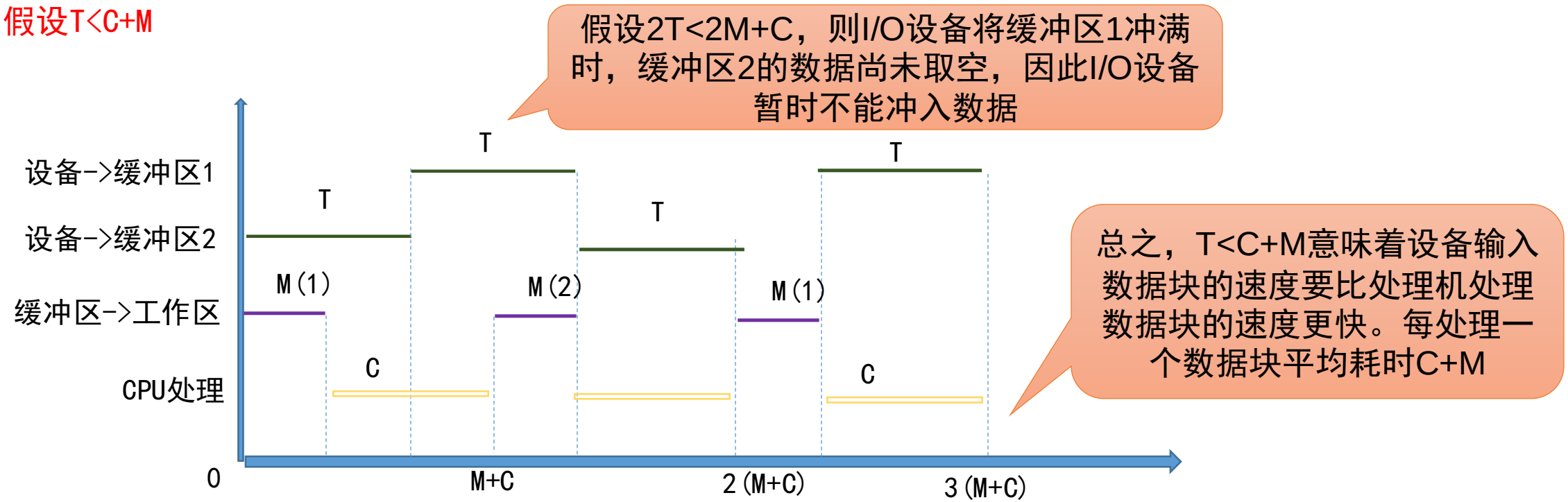



# 缓冲区管理——双缓冲

假设某用户进程请求某种块设备读入若干块的数据。若采用双缓冲的策略，操作系统会在主存中为其分配两个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）

双缓冲题目中，假设初始状态为：工作区空，其中一个缓冲区满，另一个缓冲区空

假设T<C+M




注：M(1)表示“将缓冲区1中的数据传送到工作区”；M(2)表示“将缓冲区2中的数据传送到工作区”

# 缓冲区管理——双缓冲

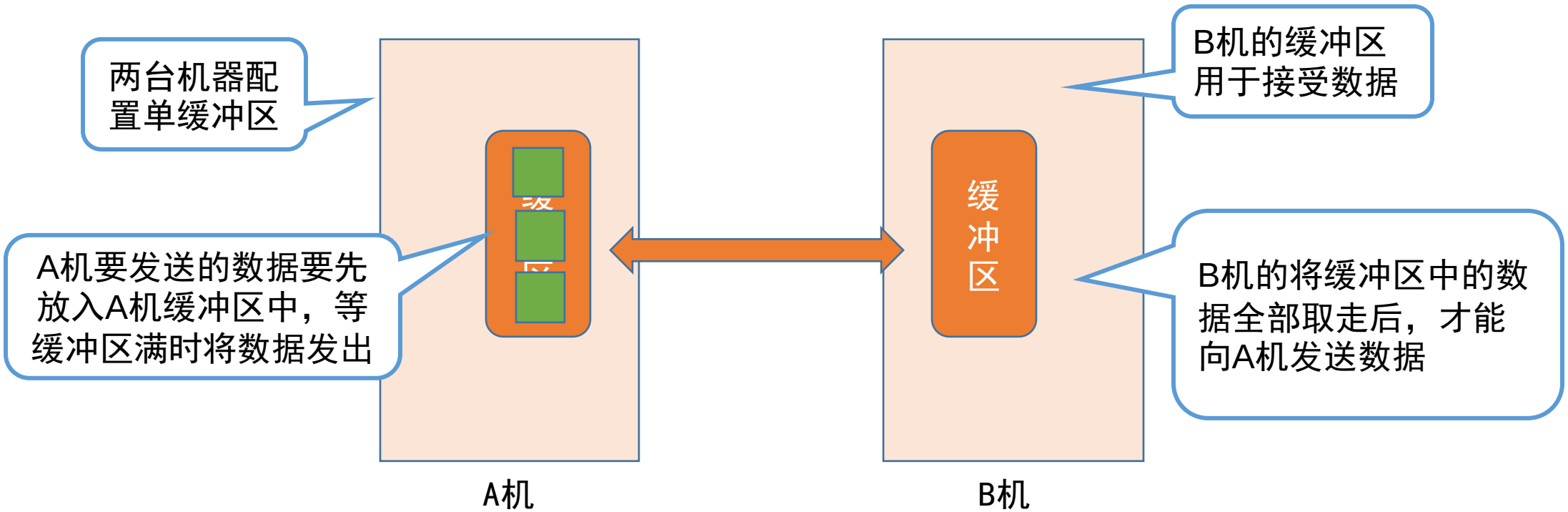
假设某用户进程请求某种块设备读入若干块的数据。若采用双缓冲的策略，操作系统会在主存中为其分配两个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）

双缓冲题目中，假设初始状态为：工作区空，其中一个缓冲区满，另一个缓冲区空

结论：采用双缓冲策略，处理一个数据块的平均耗时为Max（T，C+M）

# 缓冲区管理——使用单/双缓冲在通信时的区别

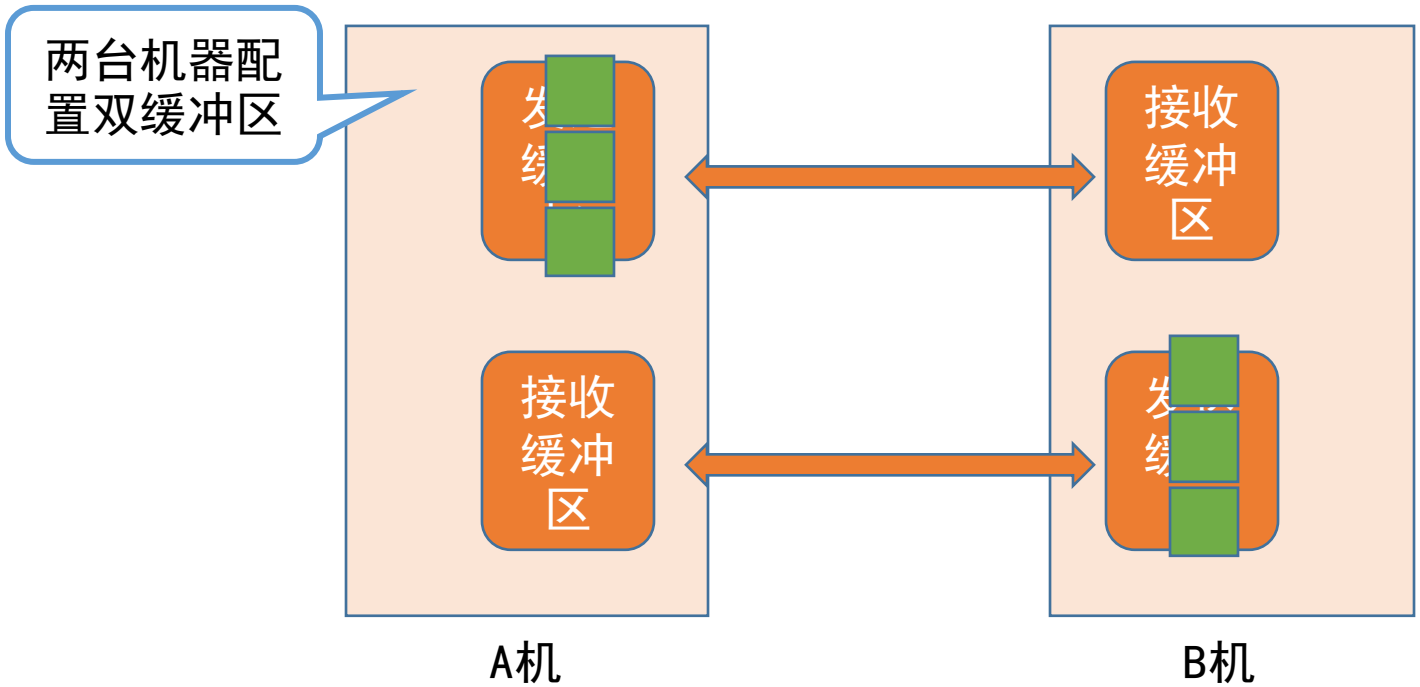
两台机器之间通信时，可以配置缓冲区用于数据的发送和接受。




显然，若两个相互通信的机器只设置单缓冲区，在任一时刻只能实现数据的单向传输。

# 缓冲区管理——使用单/双缓冲在通信时的区别

两台机器之间通信时，可以配置缓冲区用于数据的发送和接受。

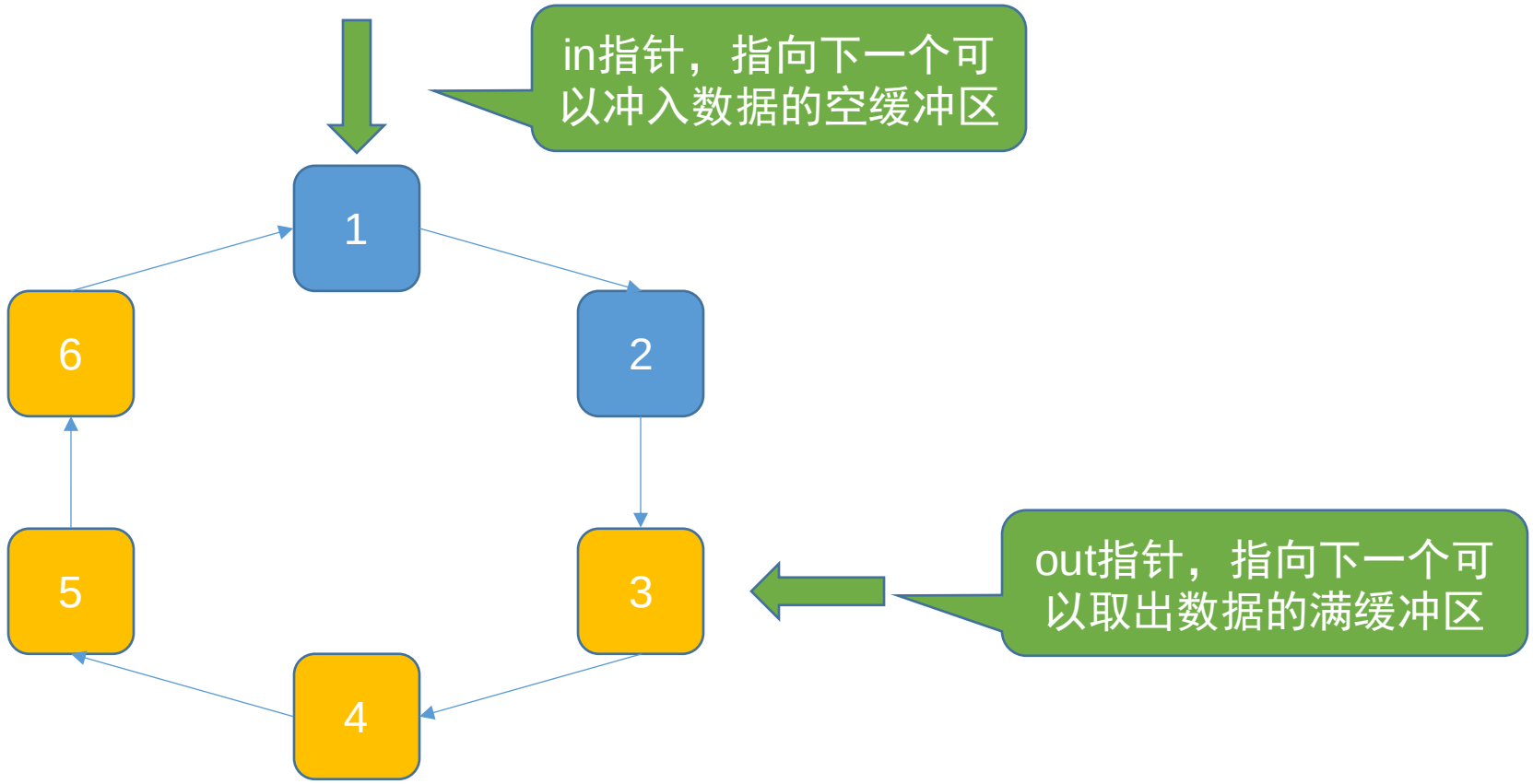



若两个相互通信的机器设置双缓冲区，则同一时刻可以实现双向的数据传输。

注：管道通信中的“管道”其实就是缓冲区。要实现数据的双向传输，必须设置两个管道

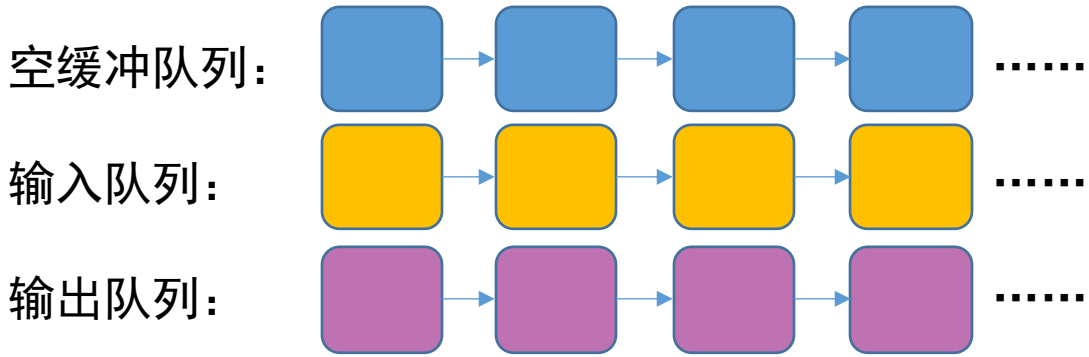
# 缓冲区管理——循环缓冲区

将多个大小相等的缓冲区链接成一个循环队列。
注：以下图示中，橙色表示已充满数据的缓冲区，绿色表示空缓冲区。

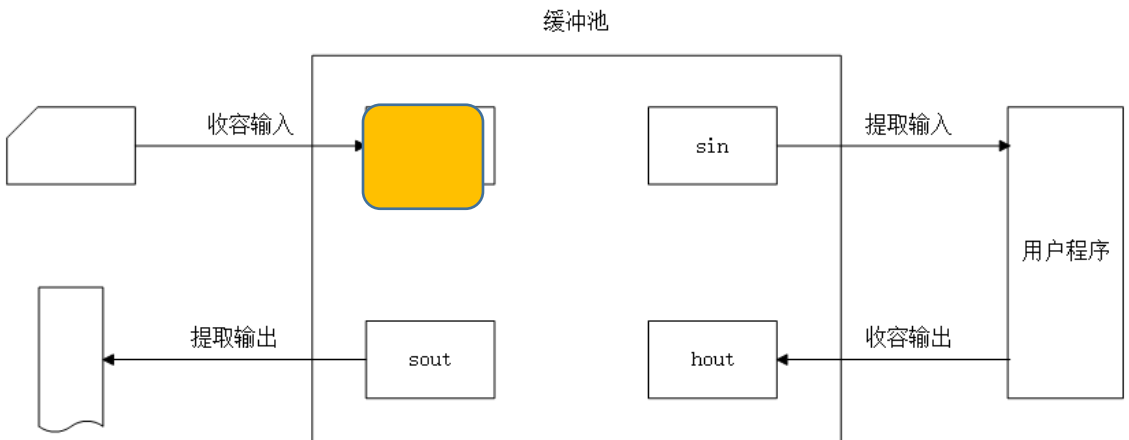


# 缓冲区管理——缓冲池

缓冲池由系统中共用的缓冲区组成。这些缓冲区按使用状况可以分为：空缓冲队列、装满输入数据的缓冲队列（输入队列）、装满输出数据的缓冲队列（输出队列）。
另外，根据一个缓冲区在实际运算中扮演的功能不同，又设置了四种工作缓冲区：用于收容输入数据的工作缓冲区（hin）、用于提取输入数据的工作缓冲区（sin）、用于收容输出数据的工作缓冲区（hout）、用于提取输出数据的工作缓冲区（sout）



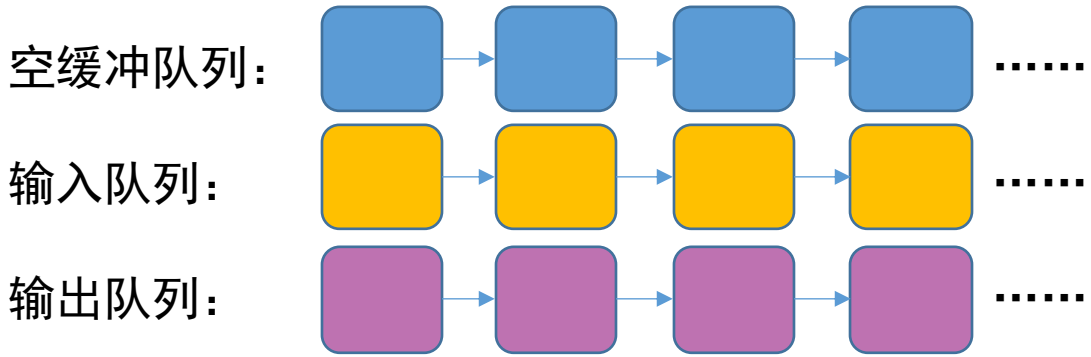

①输入进程请求输入数据



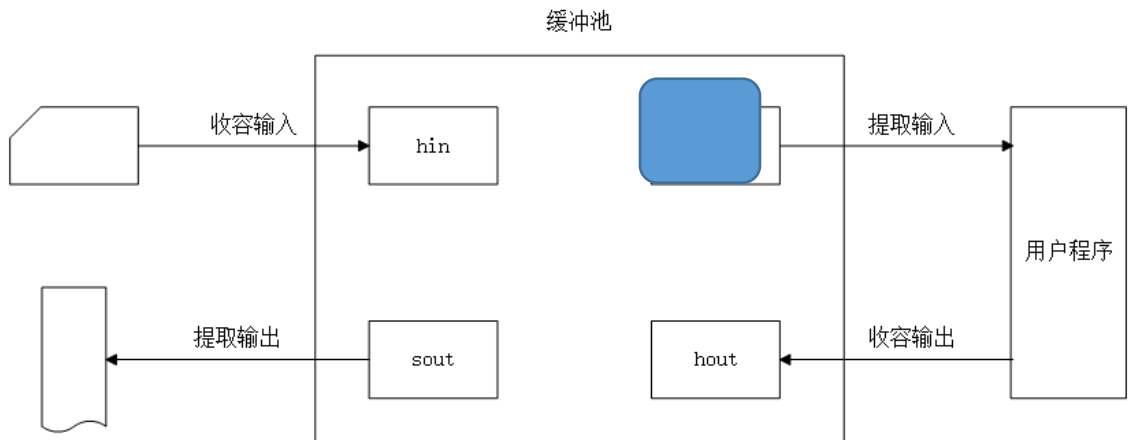

从空缓冲队列中取出一块作为收容输入数据的工作缓冲区（hin）。冲满数据后将缓冲区挂到输入队列队尾

# 缓冲区管理——缓冲池

缓冲池由系统中共用的缓冲区组成。这些缓冲区按使用状况可以分为：空缓冲队列、装满输入数据的缓冲队列（输入队列）、装满输出数据的缓冲队列（输出队列）。
另外，根据一个缓冲区在实际运算中扮演的功能不同，又设置了四种工作缓冲区：用于收容输入数据的工作缓冲区（hin）、用于提取输入数据的工作缓冲区（sin）、用于收容输出数据的工作缓冲区（hout）、用于提取输出数据的工作缓冲区（sout）



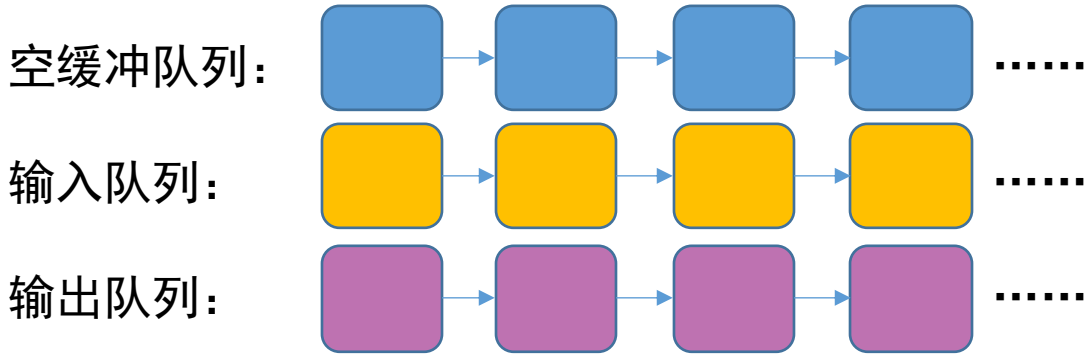

①输入进程请求输入数据
②计算进程想要取得一块输入数据



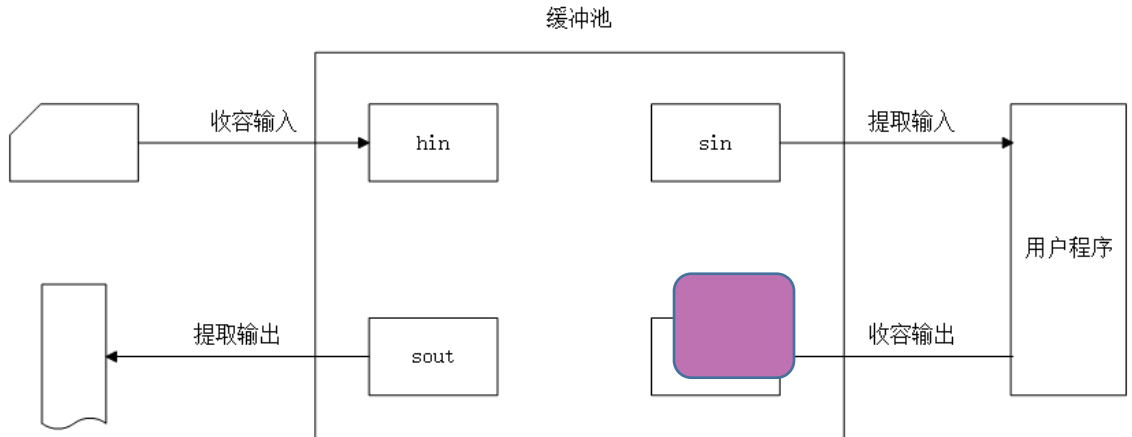

从输入队列中取得一块冲满输入数据的缓冲区作为“提取输入数据的工作缓冲区（sin）”。缓冲区读空后挂到空缓冲区队列

# 缓冲区管理——缓冲池

缓冲池由系统中共用的缓冲区组成。这些缓冲区按使用状况可以分为：空缓冲队列、装满输入数据的缓冲队列（输入队列）、装满输出数据的缓冲队列（输出队列）。
另外，根据一个缓冲区在实际运算中扮演的功能不同，又设置了四种工作缓冲区：用于收容输入数据的工作缓冲区（hin）、用于提取输入数据的工作缓冲区（sin）、用于收容输出数据的工作缓冲区（hout）、用于提取输出数据的工作缓冲区（sout）



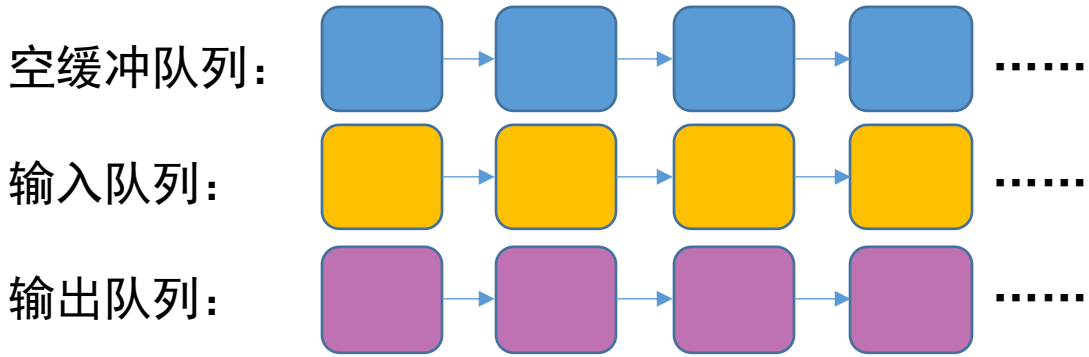

①输入进程请求输入数据
②计算进程想要取得一块输入数据
③计算进程想要将准备好的数据冲入缓冲区



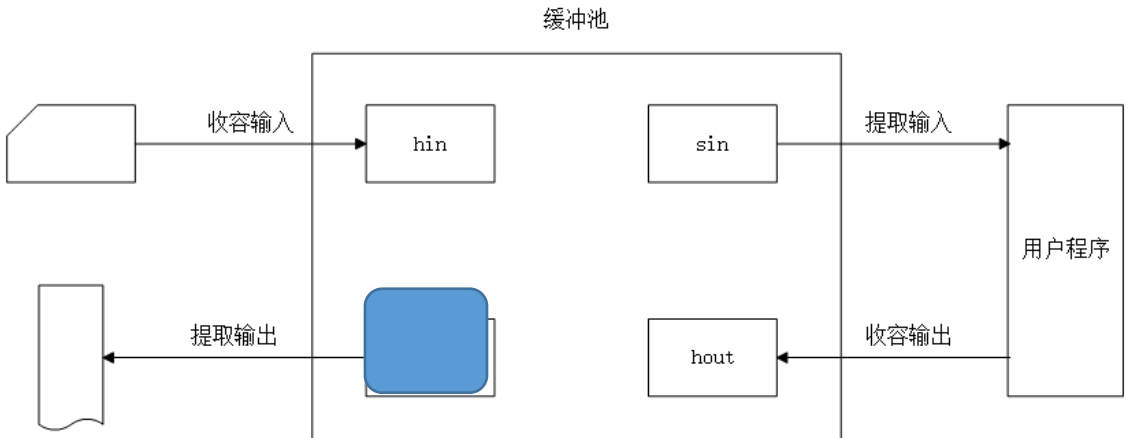

从空缓冲队列中取出一块作为“收容输出数据的工作缓冲（hout）”。数据冲满后将缓冲区挂到输出队列队尾

# 缓冲区管理——缓冲池

缓冲池由系统中共用的缓冲区组成。这些缓冲区按使用状况可以分为：空缓冲队列、装满输入数据的缓冲队列（输入队列）、装满输出数据的缓冲队列（输出队列）。
另外，根据一个缓冲区在实际运算中扮演的功能不同，又设置了四种工作缓冲区：用于收容输入数据的工作缓冲区（hin）、用于提取输入数据的工作缓冲区（sin）、用于收容输出数据的工作缓冲区（hout）、用于提取输出数据的工作缓冲区（sout）




①输入进程请求输入数据
②计算进程想要取得一块输入数据
③计算进程想要将准备好的数据冲入缓冲区
④输出进程请求输出数据




从输出队列中取得一块冲满输出数据的缓冲区作为“提取输出数据的工作缓冲区（sout）”。缓冲区读空后挂到空缓冲区队列

# 谢谢观看