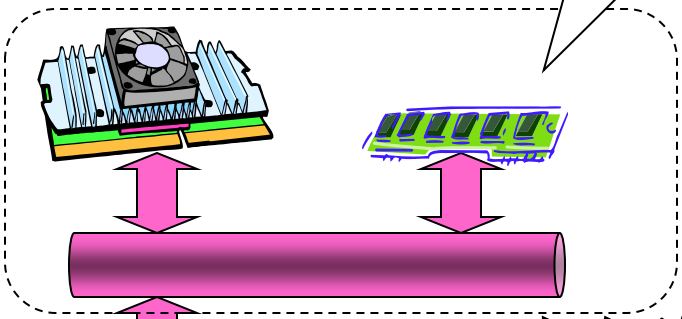
操作系统原理

黄俊杰

2024年9月

第7章 设备管理

认识计算机外设与计算机!



CPU-内存总线

图形控制器

总线控制器

PCI总线

IDE控制器

扩展总线接口

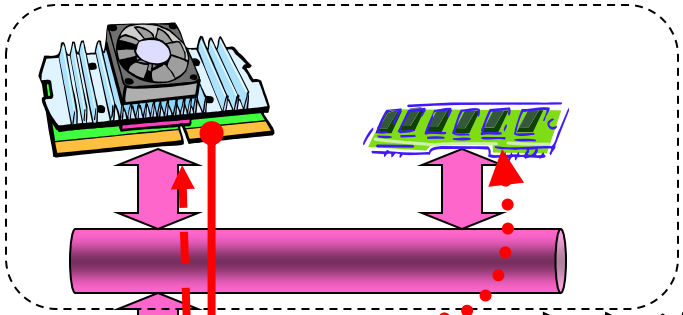
扩展总线

并行口

想一想外设怎么工作?

想让外设工作
并不复杂!

- CPU向控制器中的寄存器读写数据
- 控制器完成真正的工作，并向CPU发中断信号



CPU-内存总线

总线控制器

PCI总线

发出一个读命令

将数据送往内存

读完后向CPU发出中断

IDE控制器

读磁盘

I/O的特点

1. I/O性能经常成为系统性能的瓶颈

(1)CPU性能不等于系统性能。 响应时间也是一个重要因素。

(2)CPU性能越高，与I/O差距越大。弥补：装入更多的进程

(3)进程切换多，系统开销大。

2. 操作系统庞大复杂的原因之一是I/O的复杂性。外设种类繁多，结构各异；输入输出数据信号类型不同，速度差异很大。

设备管理的任务和功能

设备管理的基本任务是：

(1)按照用户要求来控制I/O设备操作，完成用户所希望的输入输出要求，以减轻用户编制程序的负担。

多道程序环境中，设备管理的另一个重要任务是：

(2)按照一定的算法把一个I/O设备分配给对该类设备提出请求的进程，以保证系统有条不紊地工作。

(3)充分而有效地使用这些设备，尽可能提高它们CPU的并行操作程度是设备管理的第三个重要而艰巨的任务。

设备管理功能

1)进行设备的分配：按照设备的类型(独享、共享或虚拟)和系统中所采用的分配算法，决定把一个I/O设备分配给哪一个要求该类设备的进程。

2)实现真正的I/O操作：为完成该功能，设备管理程序应具有下述三个子功能：

①在设置有通道的系统中，应根据用户提出的I/O要求，构成相应的通道程序，提供给通道去执行；

②启动指定的设备进行I/O操作；

③对通道发来的中断请求作出及时的响应和处理

设备管理功能

3)实现其它功能：如

①设备管理程序应具有对缓冲区进行管理的功能。

◆并行性　◆均衡性（使设备充分忙碌）

②向用户提供使用外部设备的方便接口，使用户摆脱繁琐的编程负担。

◆方便性　◆友好界面　◆透明性

③为改善系统的可适应性和可扩展性，应使用户程序与实际使用的物理设备无关。

7.1 I / O系统

1. I/O系统的结构

- 微机I/O系统
- 主机I/O系统

微型机I / O系统

早期微型机本身比较简单，其I/O系统多采用总线型I/O系统结构。如图所示

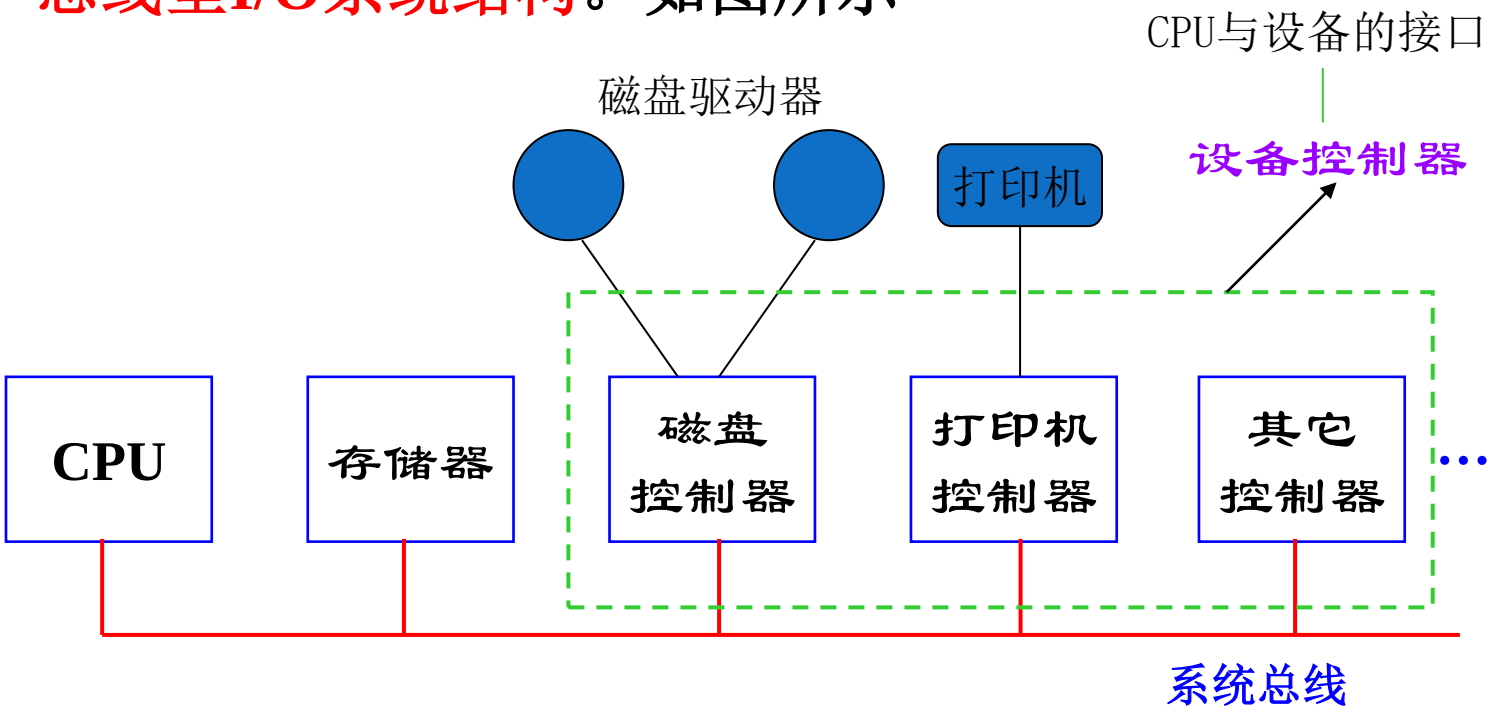


图　总线型I/O系统结构

主机I / O系统

由于主机所配置的I/O设备较多，为减轻CPU和总线的负担，采用的是具有通道的I/O系统结构。如图所示

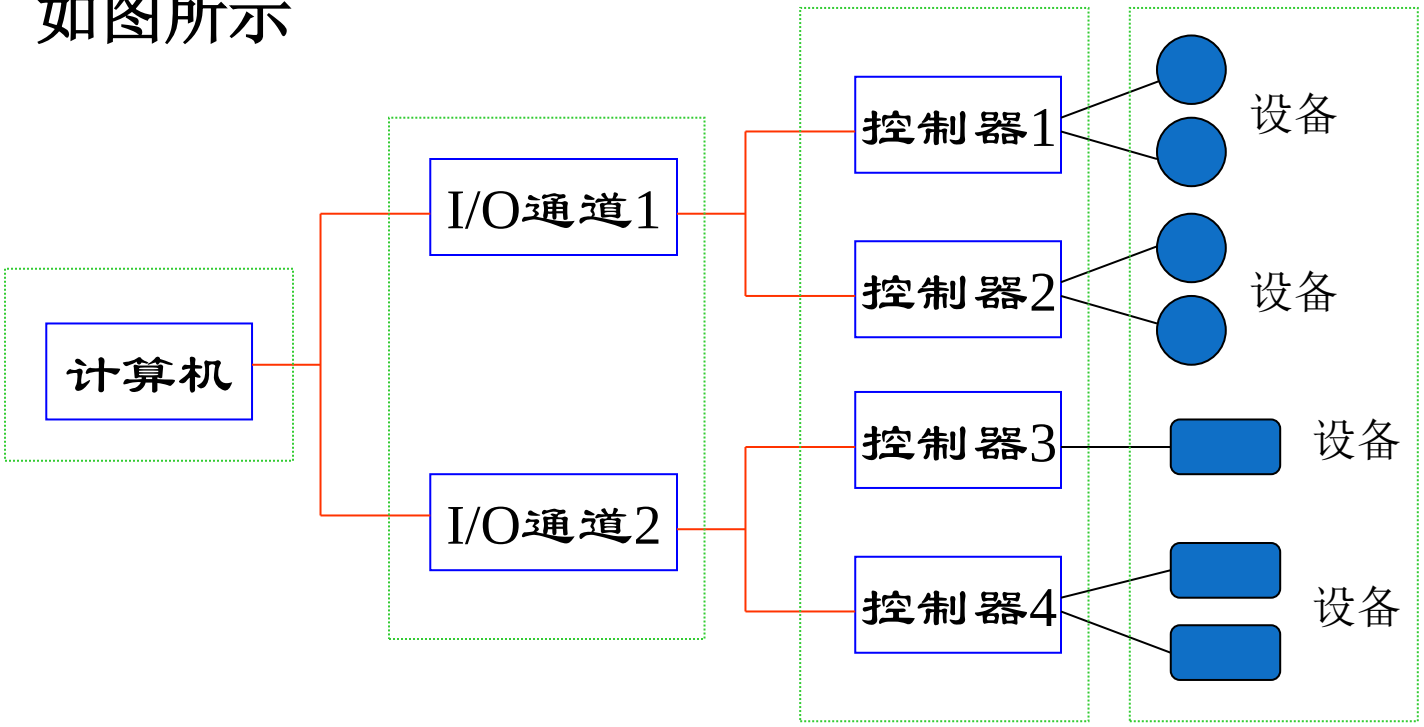


图　具有通道的I/O系统结构

7.1 I / O系统

2. I/O设备的类型

I / O设备的类型（按传输速率分类）

低速设备：是指传输速率为几字节/秒~几百字节/秒的一类设备。如键盘、鼠标器、语音的输入和输出等设备。

中速设备：是指传输速率为几千字节/秒~几十千字节/秒的一类设备。如行式打印机、激光打印机等。

高速设备：是指传输速率为几百千字节/秒~几兆字节/秒的一类设备。如：磁带机、磁（光）盘机等。

I / O设备的类型（按信息交换的单位分类）

块设备：是指信息存取是以**数据块**为单位的一类设备。主要用于存储信息，如磁盘。

磁盘设备的特征：
- 传输速率较高
- 可寻址存取
- I/O系统采用DMA方式

字符设备：是指信息存取是以**字符**为单位的一类设备。主要用于数据的输入和输出，如打印机、交互式终端等。

字符设备的特征：
- 传输速率较低
- 不可寻址存取
- I/O控制采用中断驱动方式

I/O设备的类型（按设备的共享属性分类）

独占设备：是指在一段时间内只允许一个用户（进程）访问的一类设备，即临界资源。如打印机、终端等。

共享设备：是指在一段时间内可允许多个用户（进程）访问的一类设备。如磁盘等。

虚拟设备：是指通过虚拟技术将一台独占设备变换为若干台逻辑设备，供若干用户（进程）同时使用，称经过虚拟技术处理后的一类设备为虚拟设备。

7.1 I/O系统

3. I/O设备控制器接口

4. I/O设备控制器

设备控制器的功能

接收和识别命令—是指CPU可以向设备控制器发送多种不同的命令，设备控制也能接收并识别这些命令（通过设置控制寄存器）。

数据交换—是指能实现CPU与设备控制器之间（通过数据总线）、设备控制器与设备之间的数据交换（通过设置数据寄存器）。

标识和报告设备的状态—是指设备控制器应记下与之相连的设备的状态供CPU了解（通过设置状态寄存器）。

地址识别—是指设备控制器必须能够识别它所控制的每个设备的地址，并且上述寄存器应具有唯一的地址。（通过配置地址译码器）。

设备控制器的功能

数据缓冲—由于I/O设备的速度与CPU和内存的速度相差悬殊，因此在控制器中必须设置缓冲器。

差错控制—是指设备控制器还兼管对由I/O设备传来的数据进行差错检测。

设备控制器的组成

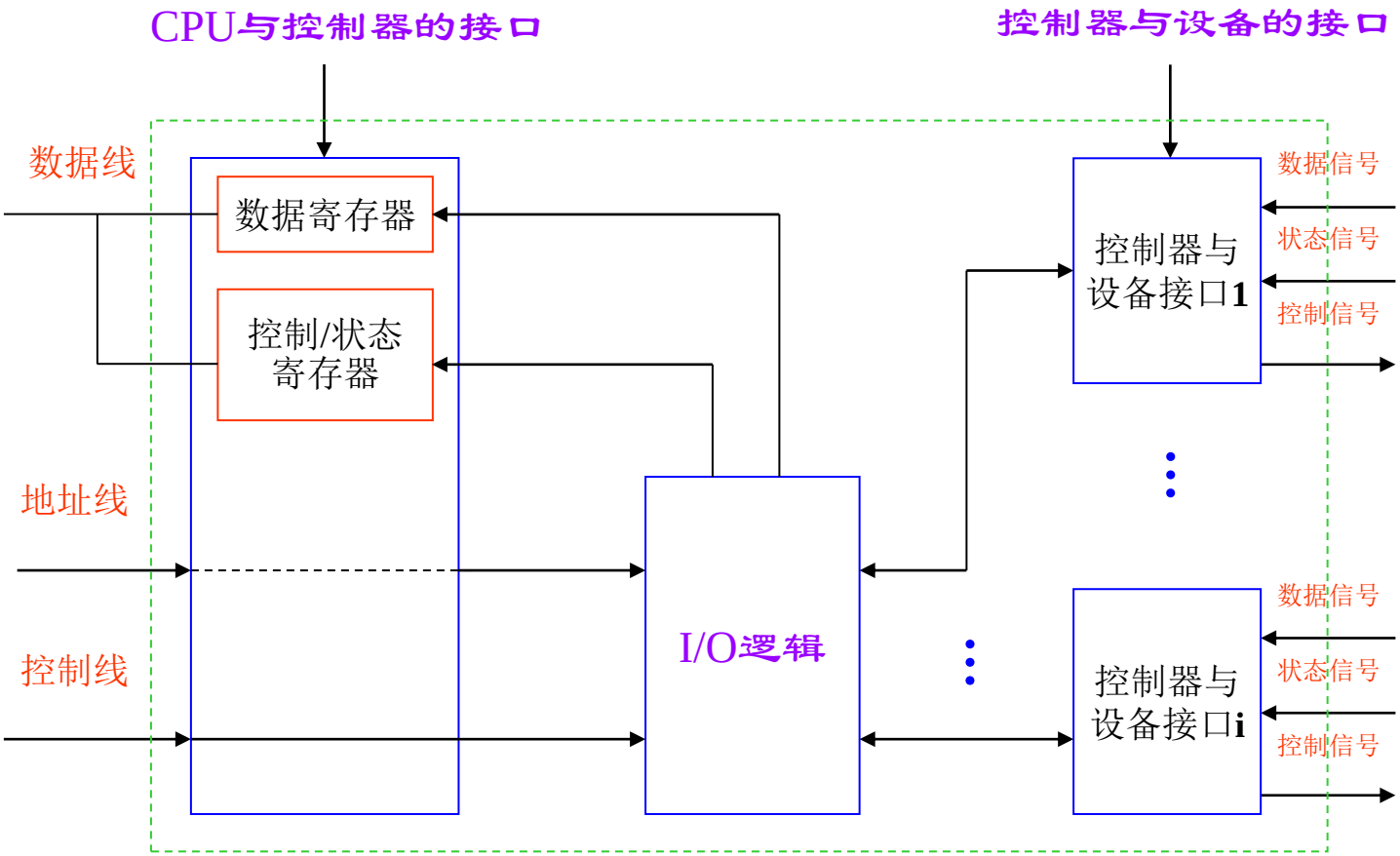
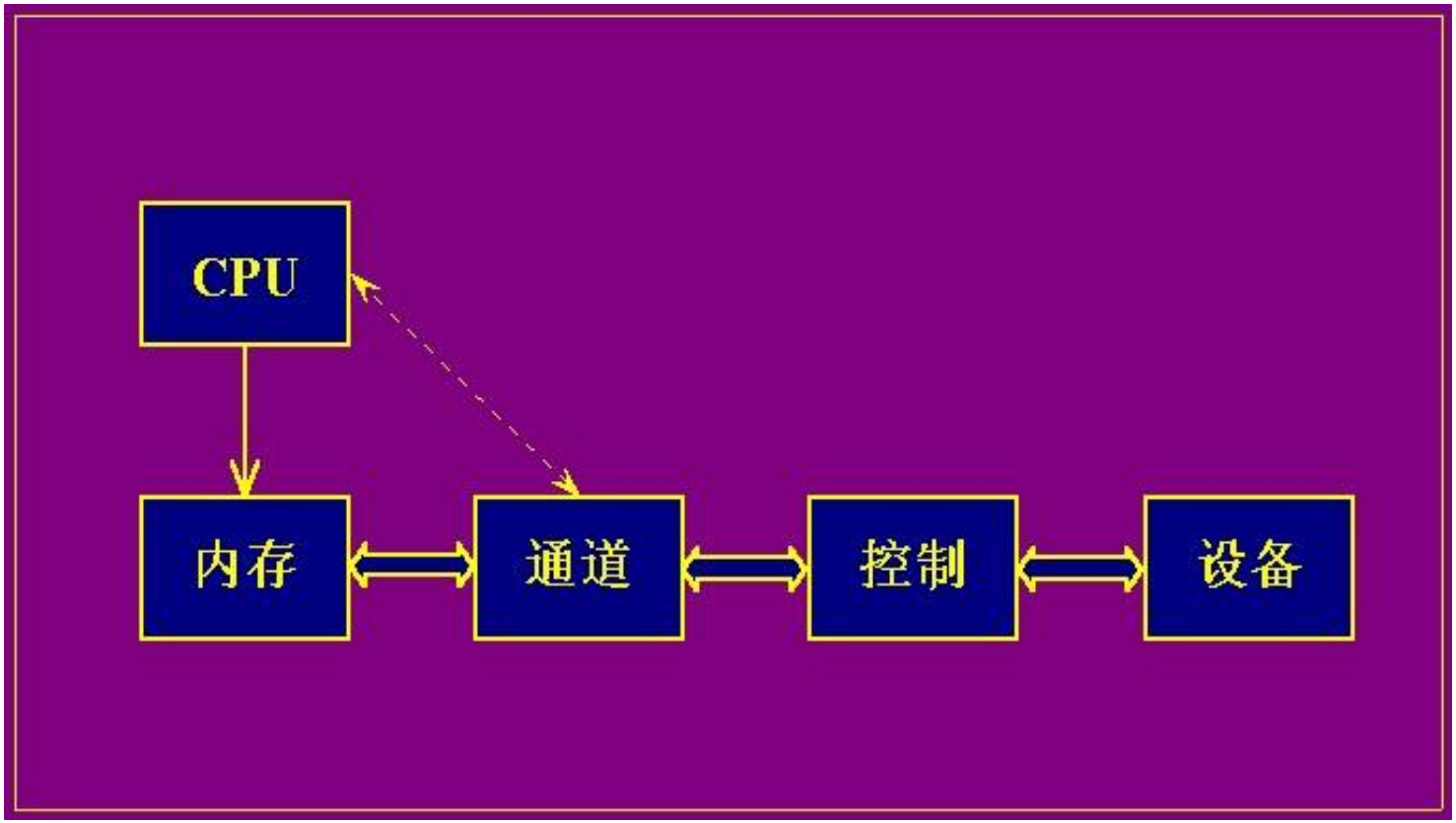


图　设备控制器的组成

5. I/O通道

I / O通道（通道的引入）

硬件连接结构



I / O通道（通道的引入）

引入通道的主要目的：是使一些原来由 CPU处理的I/O任务转由通道来承担，从而把CPU从繁杂的I/O任务中解脱出来，以保证CPU有更多的时间去进行数据的处理。

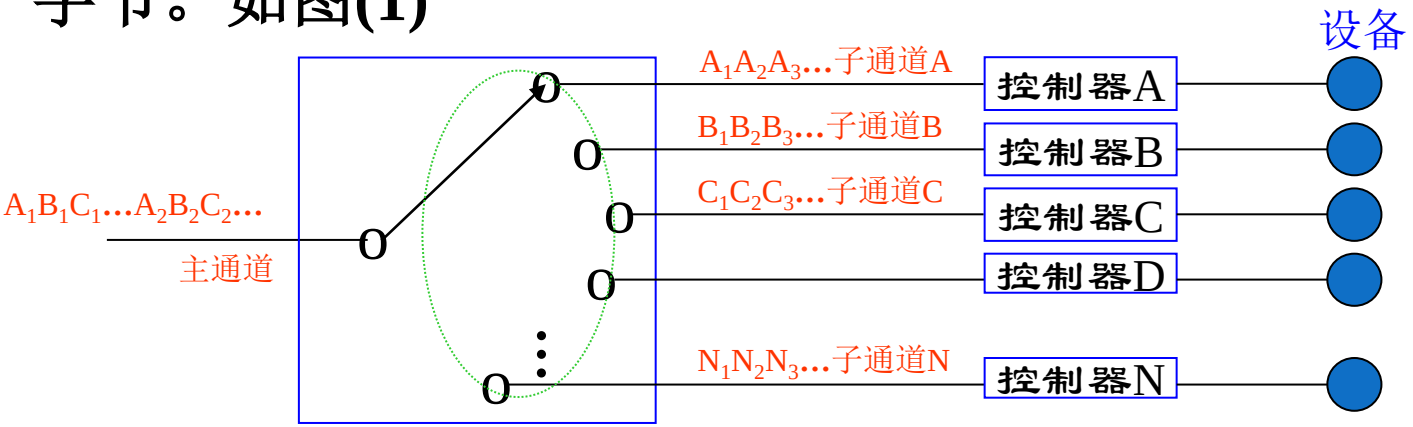
I/O通道的特点： I/O通道是一种特殊的处理机，与一般的处理机的区别主要表现在：

(1)它的指令类型单一。即：通道硬件简单，所执行的指令主要局限于与I/O操作有关的指令。

(2)它没有自己的内存，通道所执行的通道程序是存放在内存中的。即：通道与CPU共享内存。

I / O通道（通道的类型）

字节多路通道：它是为连接大量中、低速外围设备(如软驱、行式打印机等)设置的，是以字节为单位交叉地工作。当为一台设备传送一个字节后，立即转去为另一台设备传送字节。如图(1)



图(1)　字节多路通道的工作原理图

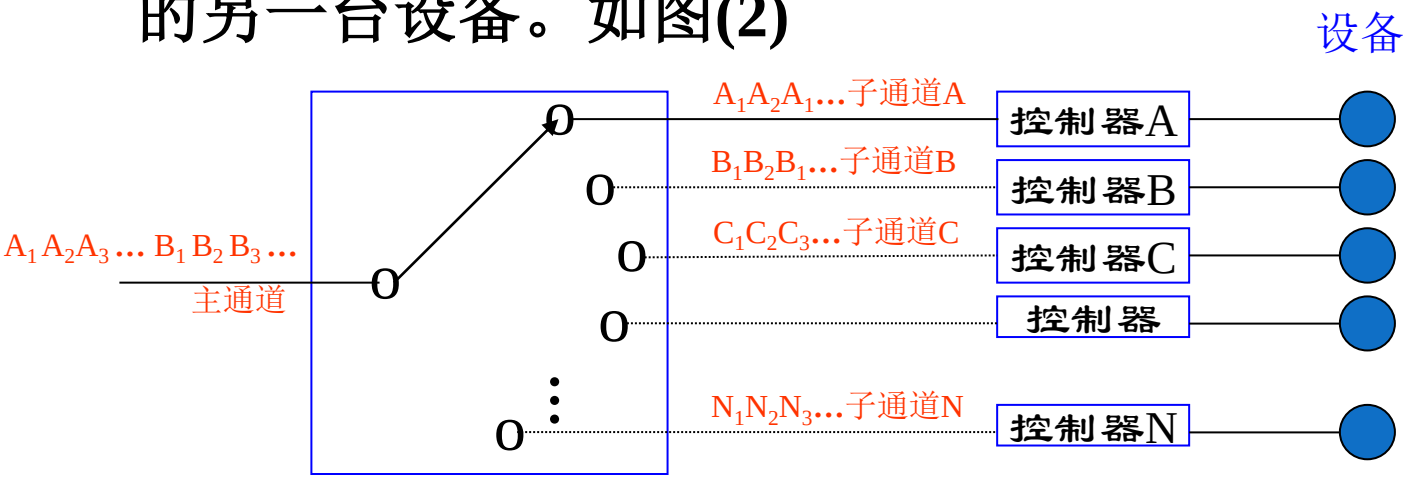
从图7-4(1)还可以看出，子通道是按时间片轮转方式共享主通道。

优点：简单，通道利用率较高；

缺点：传输速率低。

I / O通道（通道的类型）

数组选择通道：它是为连接大量快速外围设备(如硬盘、页式打印机等)设置的，是以成批的数据块为单位。但每次只能为一台设备服务，只有当该设备的输入输出完成后，才能再选择与通道相连接的另一台设备。如图(2)

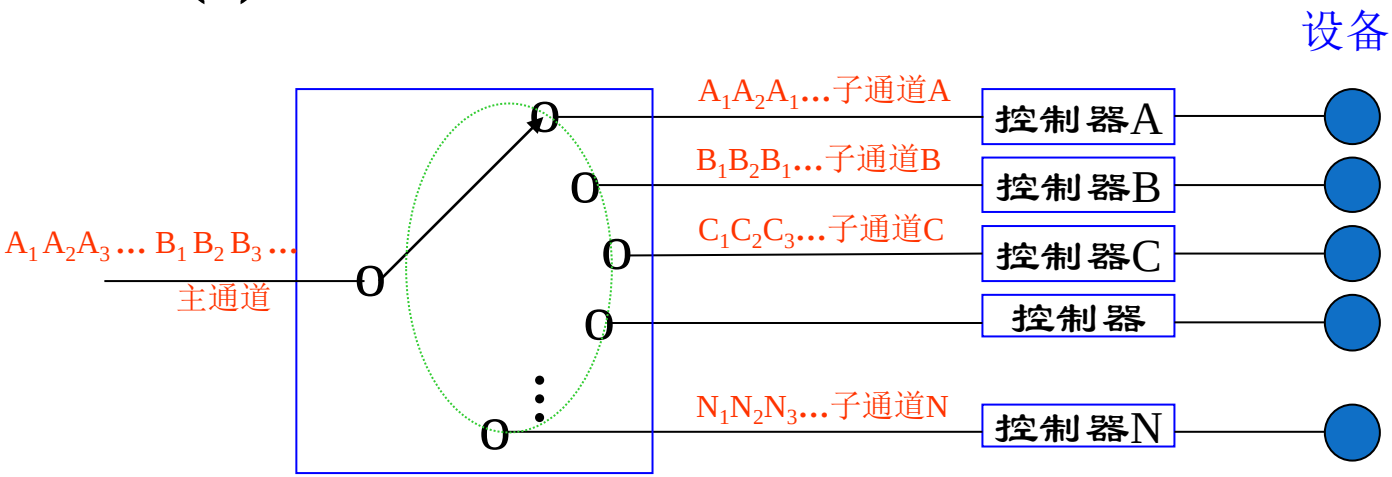


图(2)　数组选择通道的工作原理图

优点：传输速率高；**缺点：**通道利用率低。

I / O通道（通道的类型）

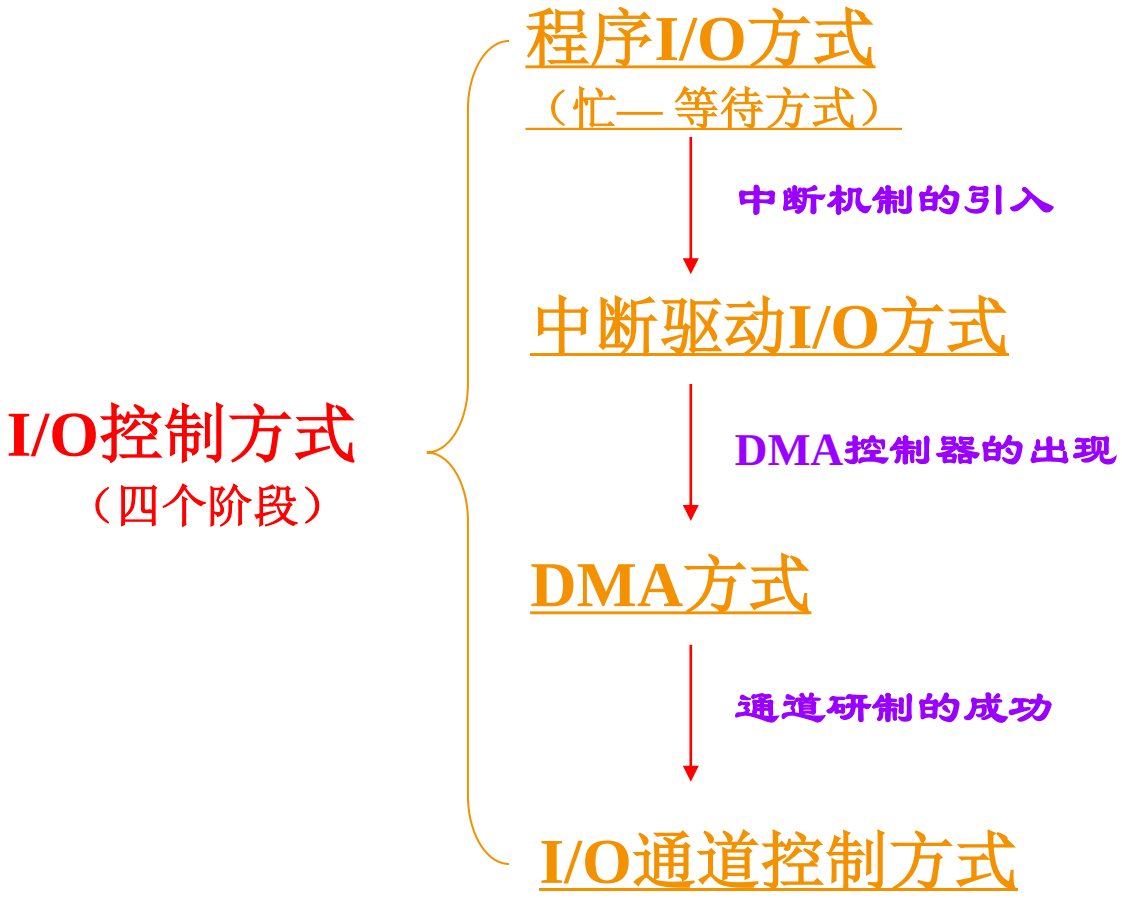
数组多路通道：它是将字节多路通道和数组选择通道各自的优点相结合而形成的一种新通道。如图(3)



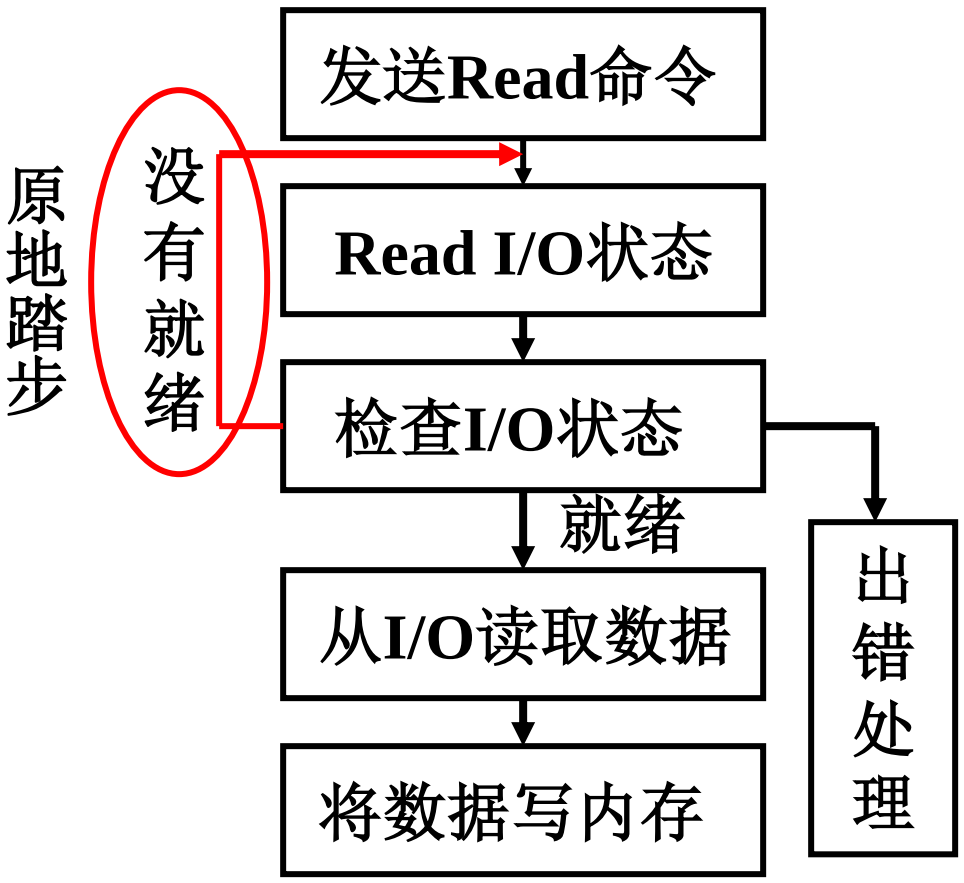
图(3)　数组多路通道的工作原理图

优点：能并发操作，通道利用率高，传输速率高；
缺点：硬件成本高，实现复杂。

7.2 I / O控制方式


I/O控制方式
（四个阶段）
程序I/O方式
（忙—等待方式）
中断机制的引入
中断驱动I/O方式
DMA控制器的出现
DMA方式
通道研制的成功
I/O通道控制方式

方案1: 原地踏步等待!（程序直接控制方式）



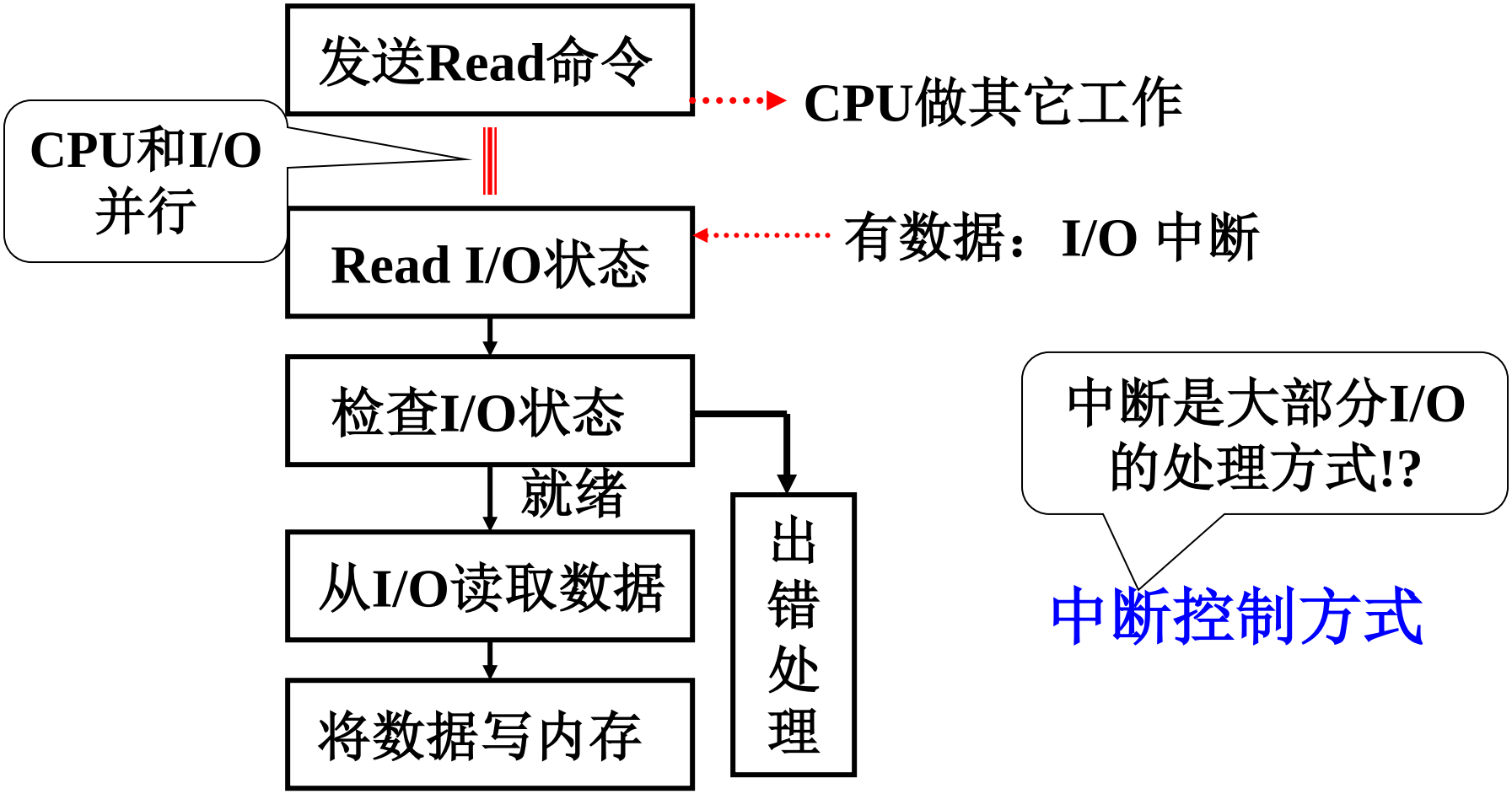
轮询

轮询!

```
in AL, 0x??
while(AL!=ready)
{
   in AL, 0x??
}
读数据..
```

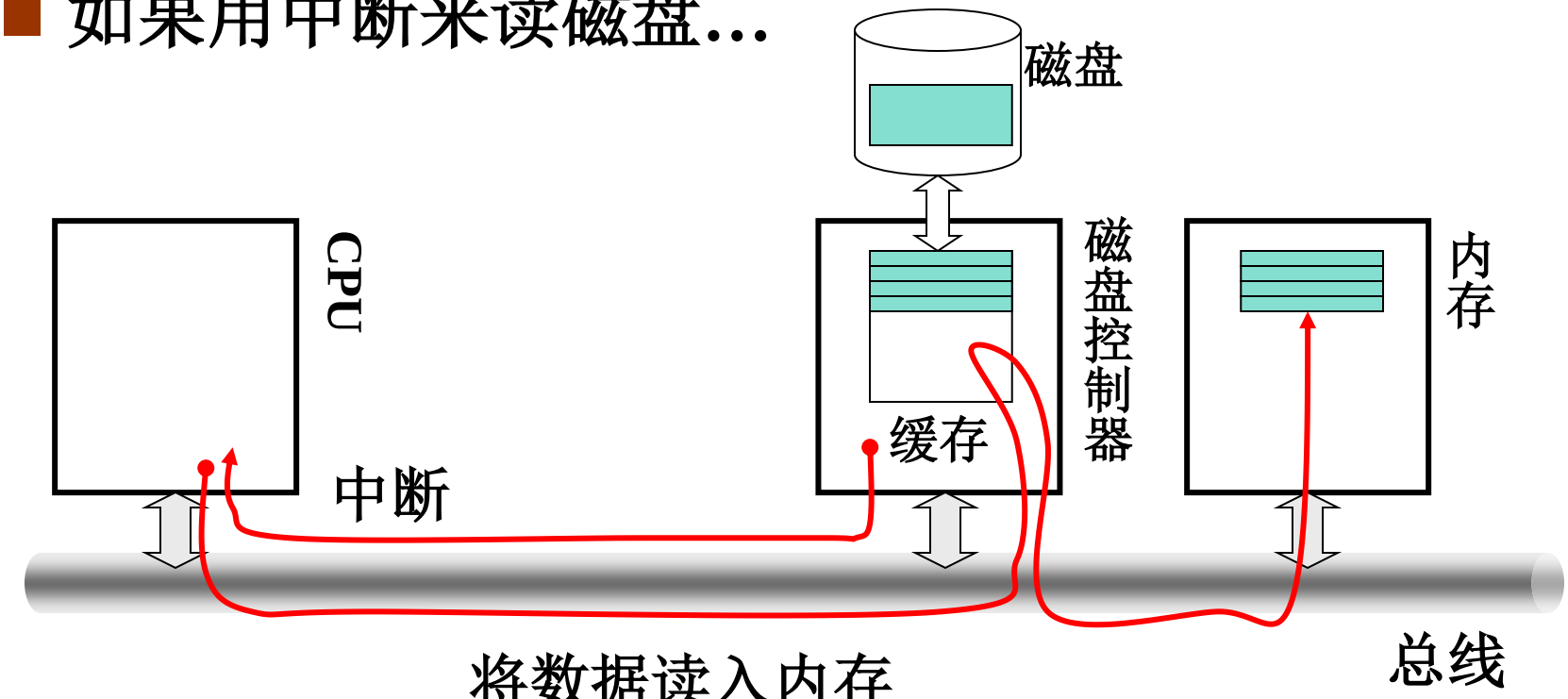
浪费CPU资源(CPU比外设快太多了)!

方案2: 设备就绪了告诉CPU一声!



中断在某些场合还不够!

- 如果用中断来读磁盘…



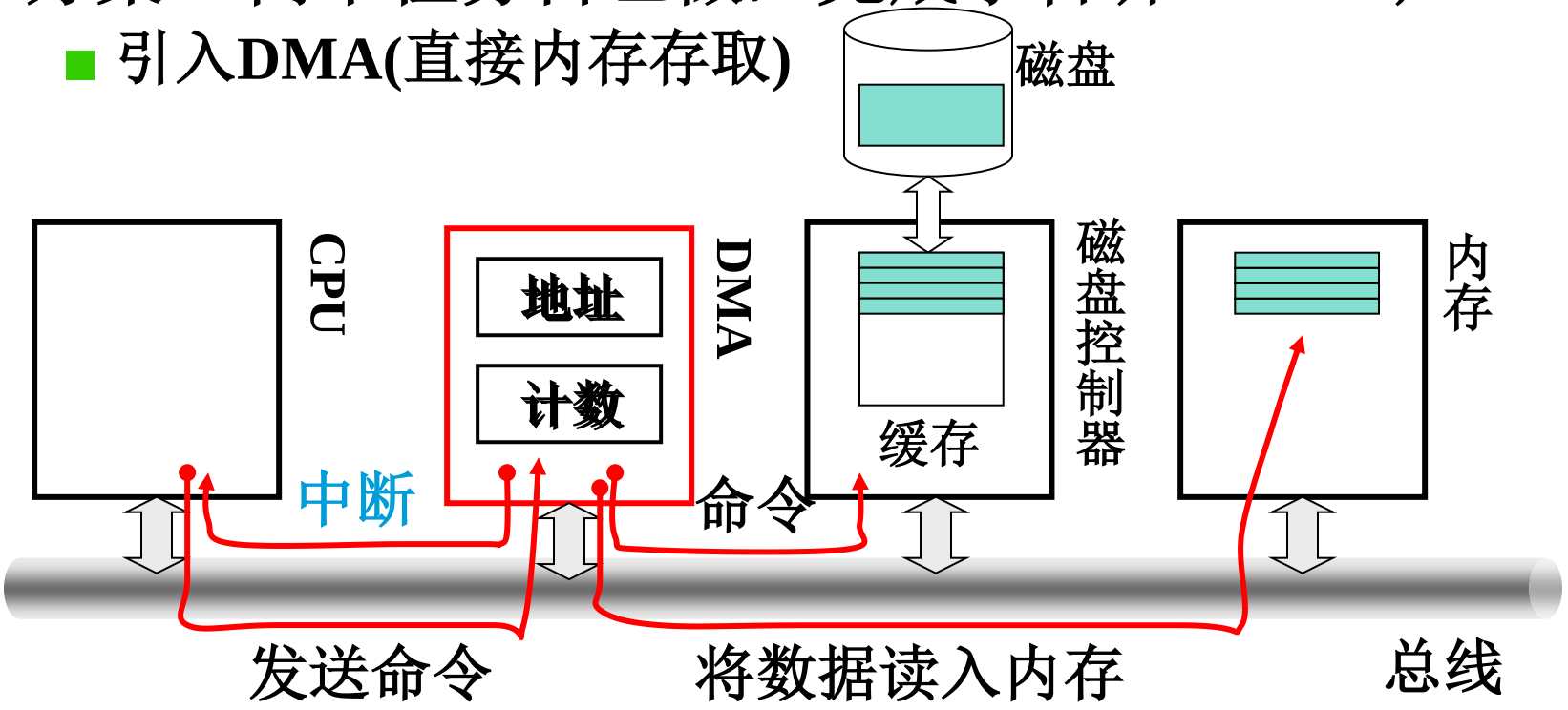
将数据读入内存

- 每个字节从缓存移动内存都由CPU负责完成

可以设计有一定处理能力的外围设备，
将一些简单任务交给它!

中断在某些场合还不够!

- 方案3: 简单任务自己做，完成了告诉CPU一声!
  - 引入DMA(直接内存存取)



  - 幸运的是: 该方式的细节由DMA设计者考虑，对于操作系统而言，考虑的仍然只是中断处理

DMA方式

1. DMA方式的引入

为减少CPU对I/O 的干预，而引入了直接存储器访问（Direct Memory Access）方式。

特点：(1)以数据块为基本单位进行I/O；

(2)数据直接从设备送入内存，或者直接从内存送入设备；

(3)仅在传送开始或结束时才需CPU干预，并行程度高。

DMA方式

2. DMA控制器的组成

DMA控制器由三部分组成。如图。

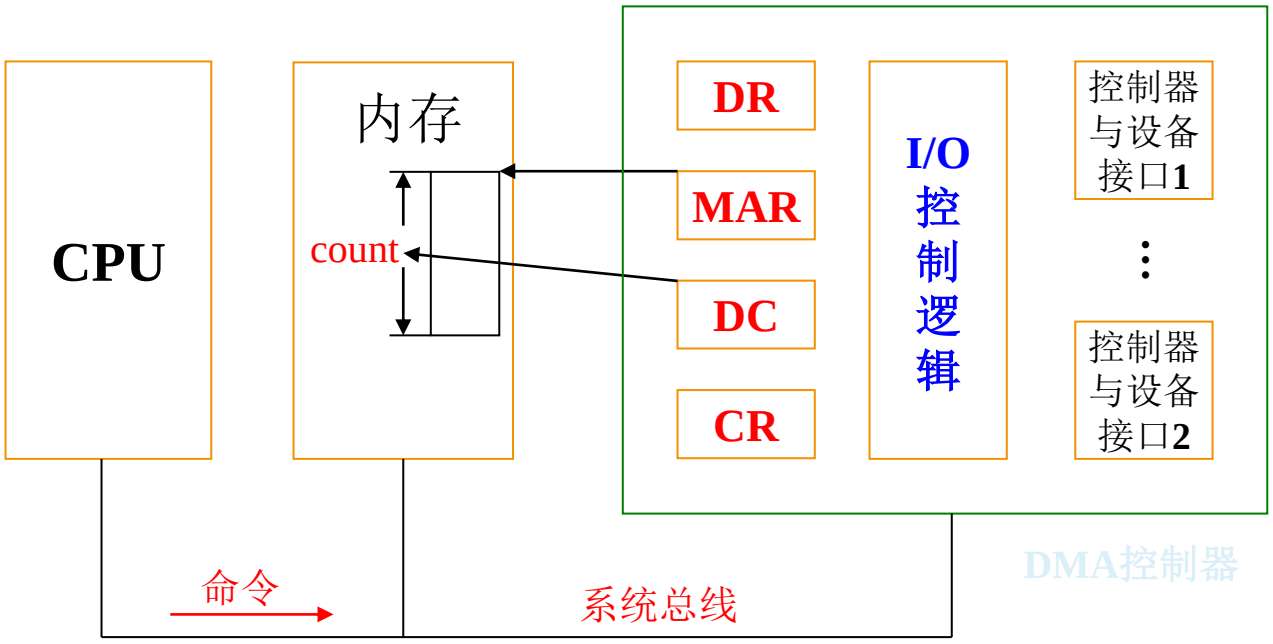


图　DMA控制器的组成

DMA方式

3. DMA工作过程（如图所示）

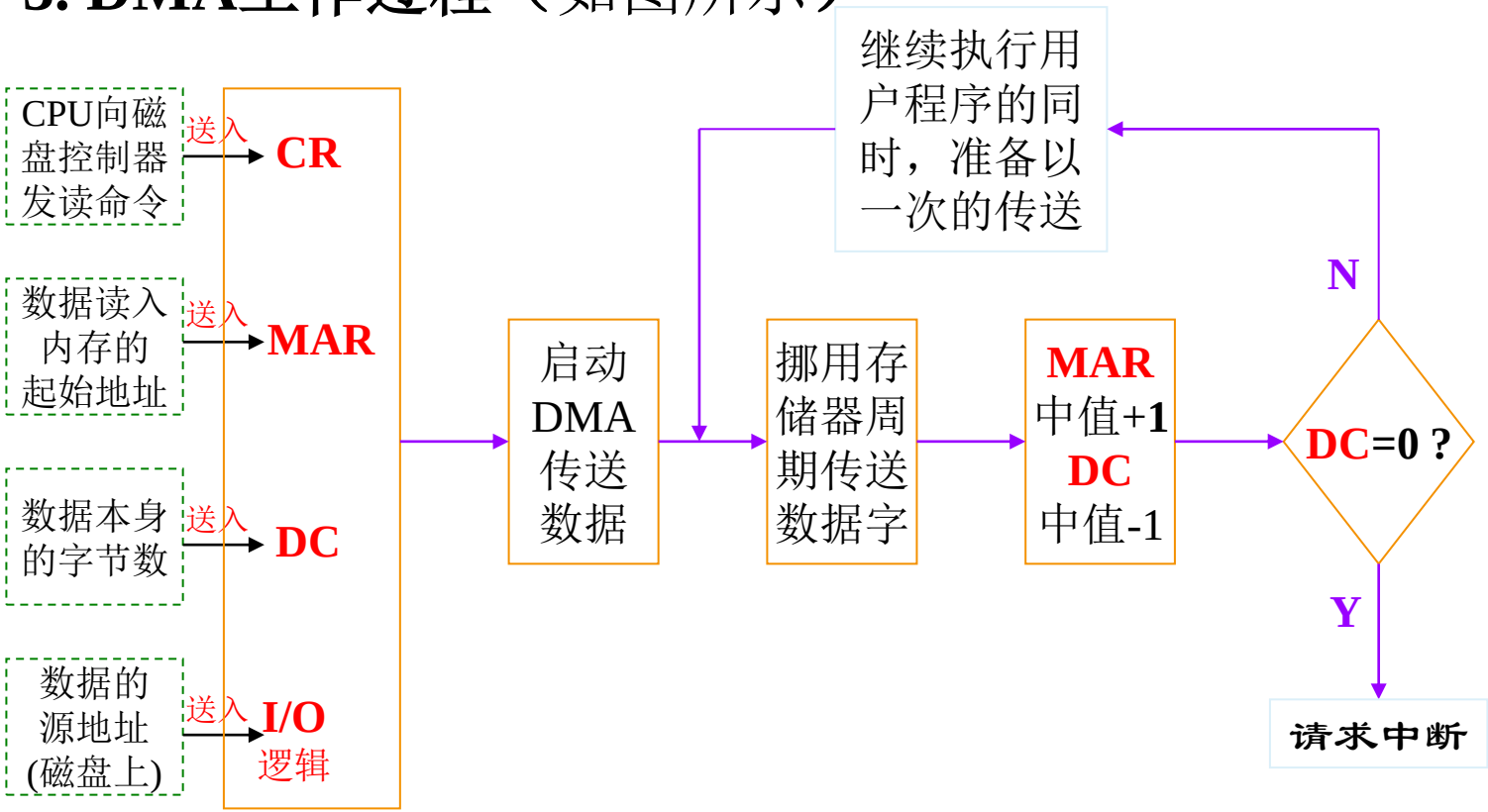


图　DMA方式的工作流程

方案4：I/O通道控制方式

1. 引入

I/O通道方式是DMA方式的发展，它可进一步减少CPU的干预。同时又可实现CPU、通道和I/O设备三者的并行工作，从而更有效地提高了整个系统的资源利用率。

通道：执行通道程序，向控制器发出命令，并具有向CPU发中断信号的功能。 一旦CPU发出指令，启动通道，则通道独立于CPU工作。由前面知，一个通道可连接多个控制器，一个控制器可连接多个设备，形成树形交叉连接。

I／O通道控制方式

2. 通道程序

通道是通过执行通道程序，并与设备控制器一起共同实现对I／O设备的控制。

1）通道运算控制部件

通道地址字CAW：记录通道程序在内存中的地址

通道命令字CCW：保存正在执行的通道指令

通道状态字CSW：存放通道执行后的返回结果

通道数据字CDW：存放传输数据

通道和CPU共用内存，通过周期窃取方式取得

I/O通道控制方式

2）通道命令及格式

通道程序是由一系列的通道指令(或称为通道命令)所构成。 每条通道指令中应包含下列诸信息：

(1)操作码：它规定了指令所执行的操作。

(2)内存地址：标明字符送入内存(读)和从内存取出(写)时的内存首址。

(3)计数：表示本条指令所要读(或写)数据的字节数。

(4)通道程序结束位P：{ P=1表示通道程序结束
P=0表示通道程序未结束

I / O通道控制方式

3) 工作原理

CPU：执行用户程序，当遇到I/O请求时，可根据该请求生成通道程序放入内存（也可事先编好放入内存），并将该通道程序的首地址放入CAW中；之后执行“启动I/O”指令，启动通道工作

通道：接收到“启动I/O”指令后，从CAW中取出通道程序的首地址，并根据首地址取出第一条指令放入CCW中，同时向CPU发回答信号，使CPU可继续执行其他程序，而通道则开始执行通道程序，完成传输工作。

7.3 缓 冲 管 理

7.3.1 缓冲的引入

- 缓和CPU与I/O设备间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽对中断响应时间的限制
- 提高CPU和I/O设备之间的并行性

缓冲管理的主要功能是：组织好缓冲区，并提供获得和释放缓冲区的手段。

缓冲区设置分为

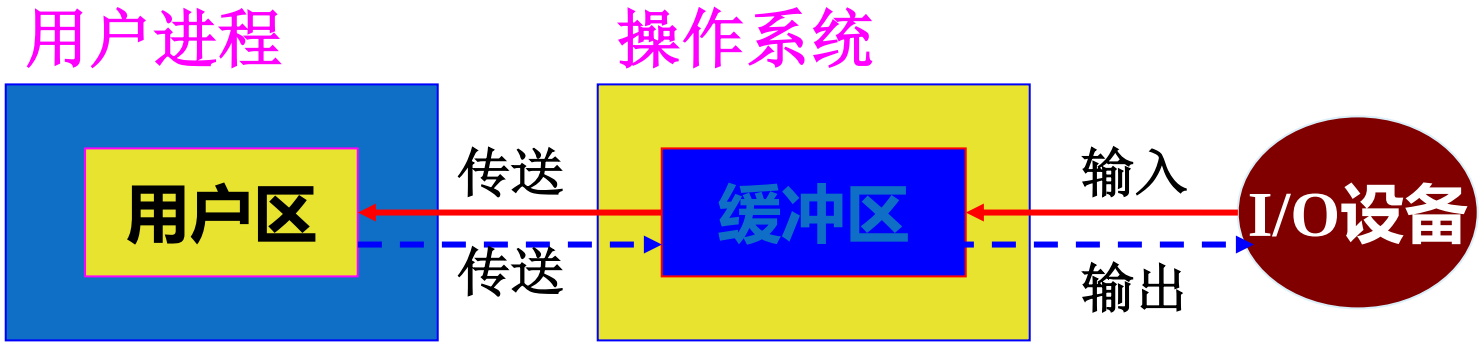
硬缓冲：在设备中设置缓冲区，由硬件实现

软缓冲：在内存中开辟一个空间，用作缓冲区

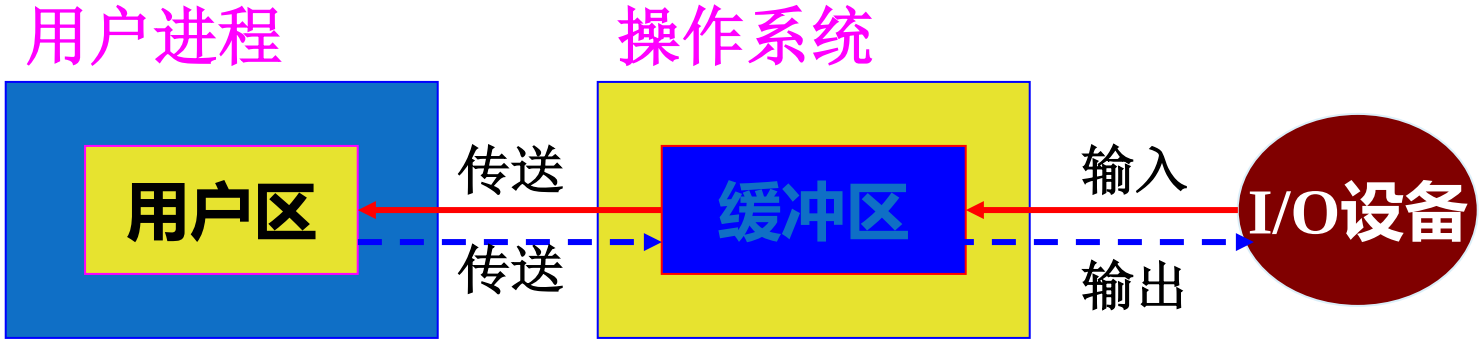
7.3.2 单缓冲和双缓冲

1. 单缓冲(Single Buffer)

是OS提供的最简单的一种缓冲形式。



7.3.2 单缓冲和双缓冲



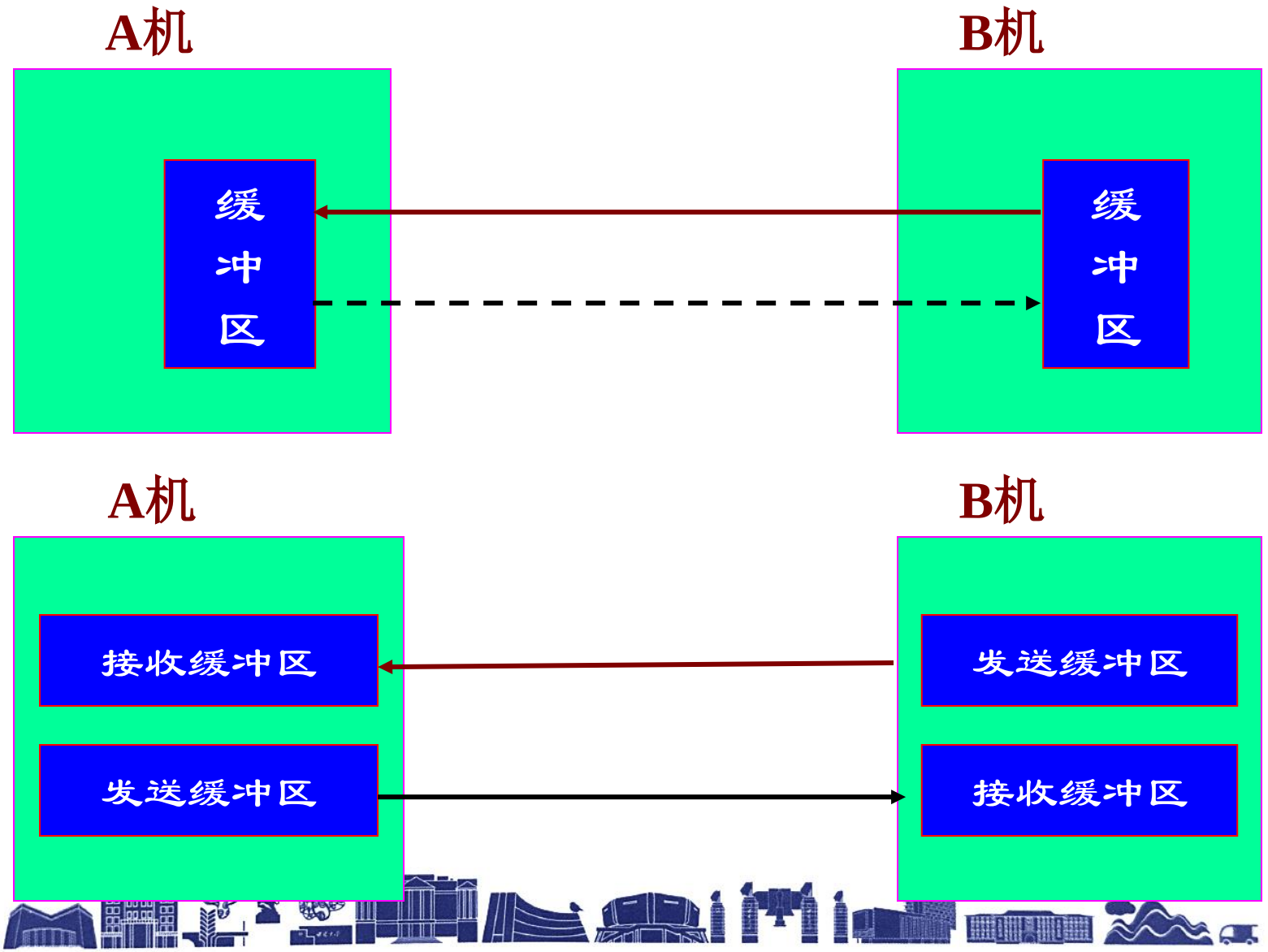
例：把一叠卡片读入并打印出来。设读卡机和打印机的速度分别是600张/m、600行/m，则每读入并打印出一张卡片上的记录需时为200ms。

说明：由于只有一个缓冲区，读卡机和打印机是串行工作的。即读卡机将数据读入缓冲区时，打印机空闲，打印机工作时，读卡机空闲。

为了能让两者更好地工作，引入了双缓冲区。

2. 双缓冲(Double Buffer)

两台计算机之间的通信：单缓冲与双缓冲



7.3.3 循环缓冲

一、循环缓冲的组成

1. 多个缓冲区

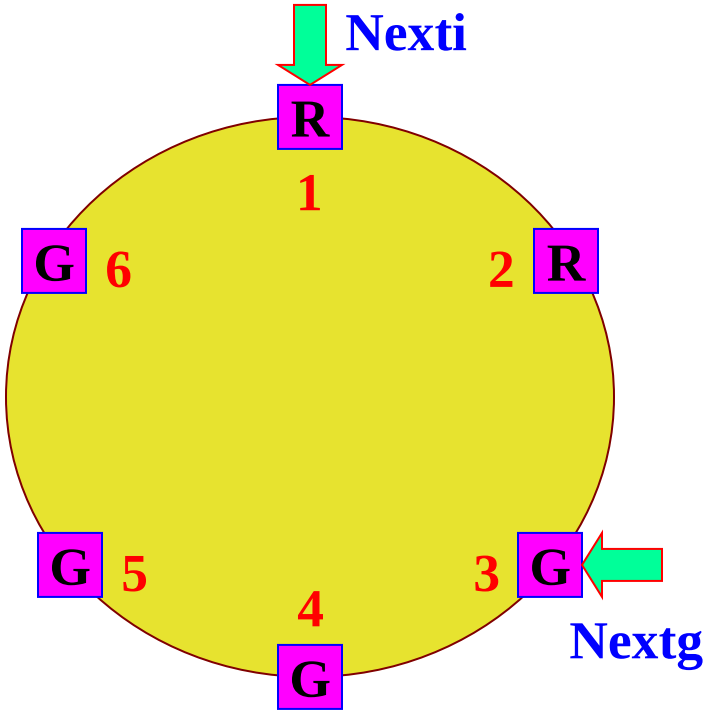
在循环缓冲中每个缓冲区的大小相同。缓冲区分为：

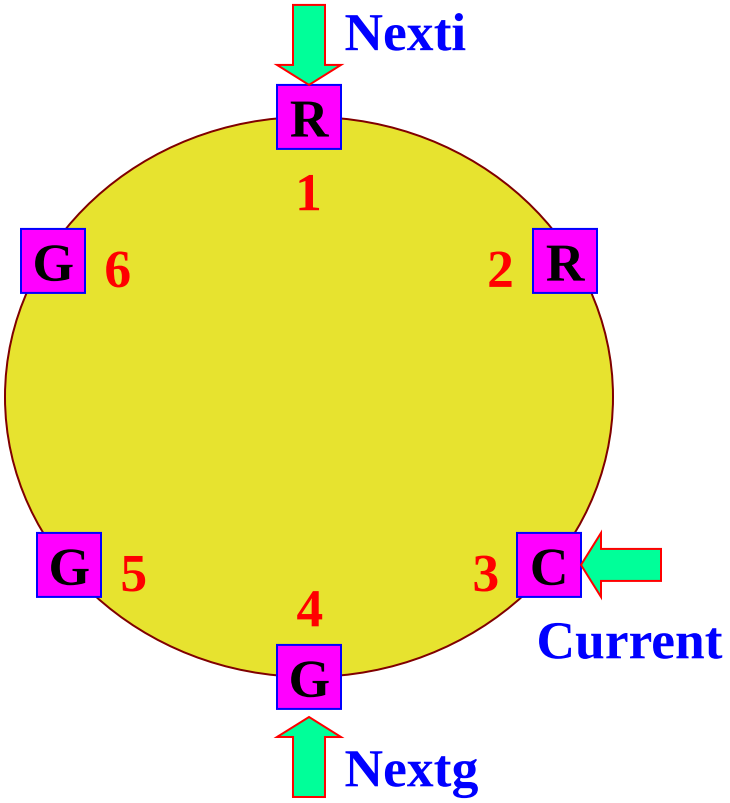
①空缓冲区R。用于存放输入数据

②满缓冲区G。其中存放的数据提供给计算进程使用

③现行工作缓冲区C。指计算进程正在使用的缓冲区

7.3.3 循环缓冲





2. 多个指针

对用于输入的多缓冲，应设置这样三个指针

①Nexti。指示输入进程下次可用的空缓冲区R

②Nextg。指示计算进程下次可用的满缓冲区G

③Current。指示计算进程正在使用的缓冲区单元。开始时指向第一单元，以后逐次指向第2，3等单元。

二、缓冲区的使用

①Getbuf过程。该过程对于计算进程而言是将指针Nextg 所指的满缓冲区G的数据提供给进程使用，用Current指针指向该缓冲区第一单元，同时将Nextg移向下一个G缓冲区；对于输入进程而言是将指针Nexti所指的空缓冲区R提供给进程使用，同时将Nexti移向下一个R缓冲区。

②Releasebuf过程。该过程对于计算进程而言是当G缓冲区的数据提取完时，便调用它来释放该缓冲区，此时将C缓冲区改为R缓冲区；对于输入进程而言当R缓冲区的数据已装满时，便调用它来释放该缓冲区，此时将C缓冲区改为G缓冲区。

三、进程同步

因此会有如下两种情况出现：

①Nexti指针追赶上Nextg指针。意味着输入进程输入数据的速度大于计算进程处理数据的速度，已把全部缓冲区(R)填满。此时，输入进程应阻塞，直到计算进程把某个缓冲区的数据全部提取完，使之成为空缓冲区，并用Releasebuf过程释放它，才将输入进程唤醒。称此种情况为系统受计算限制。

②Nextg指针追赶上Nexti指针。意味着计算进程处理数据的速度大于输入进程输入数据的速度，已把全部缓冲区(G)抽空。此时，计算进程应阻塞，直到输入进程装满某个缓冲区的数据，使之成为满缓冲区，并用Releasebuf过程释放它，才将计算进程唤醒。称此种情况为系统受I/O限制。

7.3.4 缓冲池(Buffer Pool)

上述循环缓冲由于仅适合I/O进程和计算进程，是专用缓冲，缓冲区的利用率不高。目前广泛流行的公用缓冲池，由于可供多个进程共享，可提高缓冲区的利用率。

一、缓冲池(Buffer Pool)的组成

对于可用于I/O的缓冲池，包含的缓冲区类型有：

①空（闲）缓冲区； ②装满输入数据的缓冲区；

③装满输出数据的缓冲区。

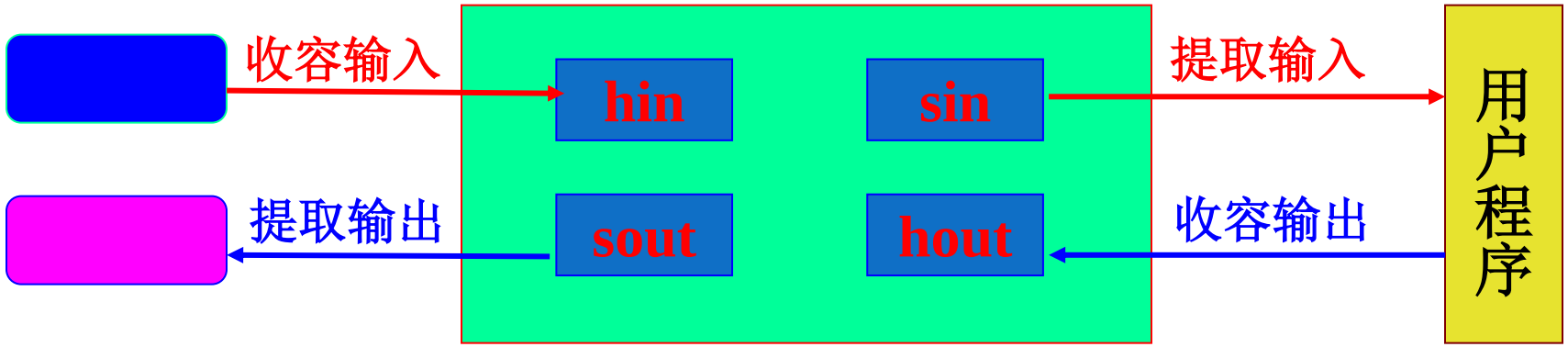
对应的有以下三个缓冲队列：

①空缓冲队列emq。由空缓冲区所链成的队列

②输入队列inq。由装满输入数据的缓冲区所链成的队列

③输出队列outq。由装满输出数据的缓冲区所链成的队列

二、缓冲区的工作方式



①收容输入工作方式
- 输入数据时，调用Getbuf (emq)
- 装满数据后，调用Putbuf (inq, hin)

②提取输入工作方式
- 输入数据时，调用Getbuf (inq)
- 提完数据后，调用Putbuf (emq, sin)

③收容输出工作方式
- 输出数据时，调用Getbuf (emq)
- 装满数据后，调用Putbuf (outq, hout)

④提取输出工作方式
- 输出数据时，调用Getbuf (outq)
- 提完数据后，调用Putbuf (emq, sout)

7.4 设 备 分 配

7.4.1 设备分配中的数据结构

设备管理程序对I/O设备进行分配和控制是借助一些表格。表格中记录了对I/O设备控制所需之信息。它们是设备管理程序实现管理功能的数据结构。其中包括：

(1)设备控制表(DCT)；

(2)控制器(控制)表(COCT)；

(3)通道(控制)表(CHCT)；

(4)系统设备表(SDT)。

一、设备控制表DCT

系统中为每一I/O设备都设置了一张用以记录该设备情况的设备控制表DCT，如下图所示。

设备类型Type
设备标识符deviceID
设备状态 等待/不等待 忙/闲
指向控制表的指针
重复执行次数或时间
挂起队列的队首指针
挂起队列的队尾指针

表中包含的内容有以下几项

一、设备控制表DCT

1. 挂起队列队首&队尾指针：凡因请求本设备未得到满足的进程，将其PCB按一定的策略排成一队列。

2.指向控制器COCT的指针：若I/O设备与内存之间仅有一条通道，此时设备仅连向一个控制器，该表目中便填上该控制器COCT的首址，为提高I/O系统的灵活性和可靠性，不少系统中把一个I/O设备与多个控制器相连，此时该表目中应填上相应的多个COCT的首址。

一、设备控制表DCT

3.重复执行次数或时间：在某些系统中，若发生数据传送错误，系统并不立即认为传送失败，而是重复执行该传送操作、只要在规定的重复执行次数。

4.设备状态：当某进程提出I/O请求时，系统除了给该进程分配I/O设备外。还需同时分配相应的控制器和通道。若此时与设备连接的所有控制器或通道都忙，则把该进程的PCB插入该控制器或通道的等待队列，并置位“等待 / 不等待”标志位；若I/O设备自身忙，则置位“忙 / 闲”标志位。

二、控制器控制表、通道表及系统设备表

1. 控制器控制表COCT

与设备控制表类似的，系统中为每一控制器都设置了一张用以记录该控制器情况的控制器控制表COCT，如下图所示。

控制器标识符controllerID
控制器状态　　忙/闲
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

COCT

2. 通道控制表CHCT

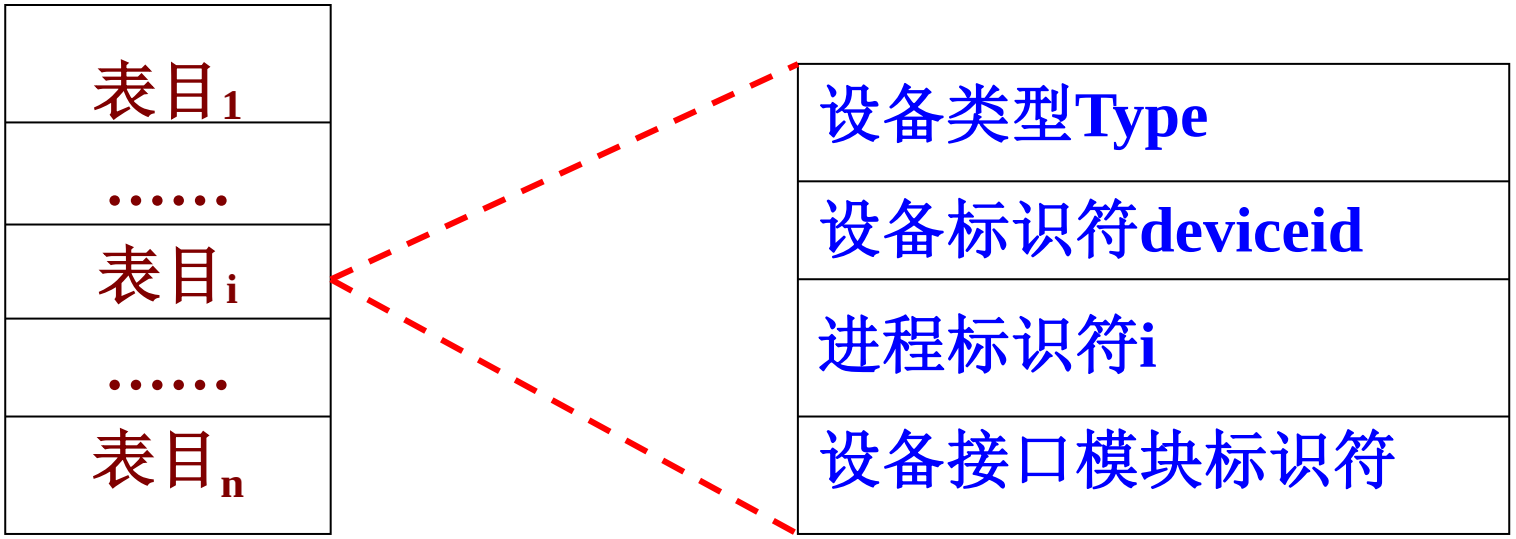
与设备控制表类似的，系统中为每一通道都设置了一张用以记录该通道情况的通道控制表COCT，如下图所示。

通道标识符channelID
通道状态　　忙/闲
指向控制器表首址
通道队列的队首指针
通道队列的队尾指针

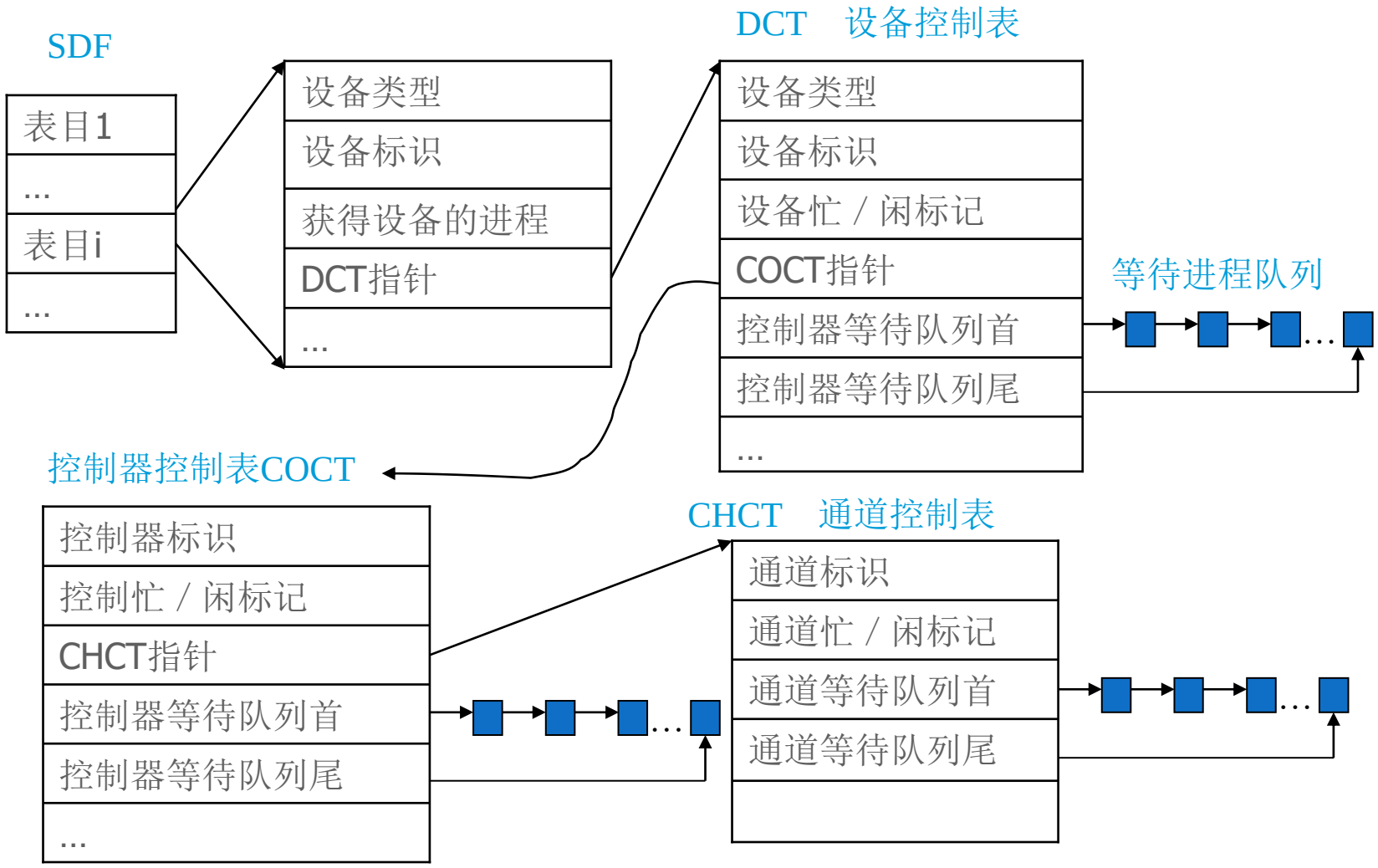
CHCT

3. 系统设备表SDT

系统设备表SDT是系统范围的数据结构，它记录了系统中所拥有的全部I/O设备，每一设备有一个表目，每个表目又包括若干项。下图示出了一种典型的系统设备表，其中每个表目包括四项：设备类、设备标识符、进程标识符、设备接口模块标识符。



设备分配数据结构及其关系图



7.4.2 设备分配时应考虑的因素

* I/O设备的固有属性：该设备仅适于某进程独占还是可供多个进程共享；

* I/O设备的分配算法：采用先请求先分配方式，还是按优先权最高者优先的方法；

* 设备分配的安全性：不合理的设备分配有可能导致死锁的发生；

* 与设备的独立性（无关性）：用户程序与实际分配的物理设备无关。

一、考虑设备的固有属性

设备固有属性是：**独占**和**共享**。

独享分配方式：即一个设备分配给某作业后便由该作业独占，直到该作业完成并释放后，其它作业方能使用。同样，像纸带穿孔机、卡片输入机等也都不宜由几个作业共享，而应采用独享分配方式。

这种分配方式的主要缺点是，**I/O设备通常都得不到充分的利用。**

一、考虑设备的固有属性

事实上，系统中独占型设备的数量是有限的，往往不能满足诸多进程的需求，成为系统中的“瓶颈”资源，使许多进程由于等待某些独占设备成为可用而被阻塞。而另一方面，使这些设备的利用率很低。为了克服这种缺点，人们常通过共享设备来模拟独占设备的动作，使独占型设备成为共享设备，从而提高它们的利用率。实现这一技术的软、硬件系统称为**假脱机系统 (SPOOLing系统)**。

二、I/O设备的分配算法

I/O设备的分配与进程调度相似，同样可采用如下的一些算法：

1.先请求先服务： 按发出请求的先后次序排成一个等待该设备的队列，设备分配程序把I/O设备分配给队列中第一个进程。

2.优先权最高者优先： 对它的I/O请求，也赋予高的优先权，显然有助于该进程尽快完成，从而尽早释放它所占用的资源。对于优先权相同的I/O请求，则按先请求先分配的原则排队。

三、设备分配中的安全性

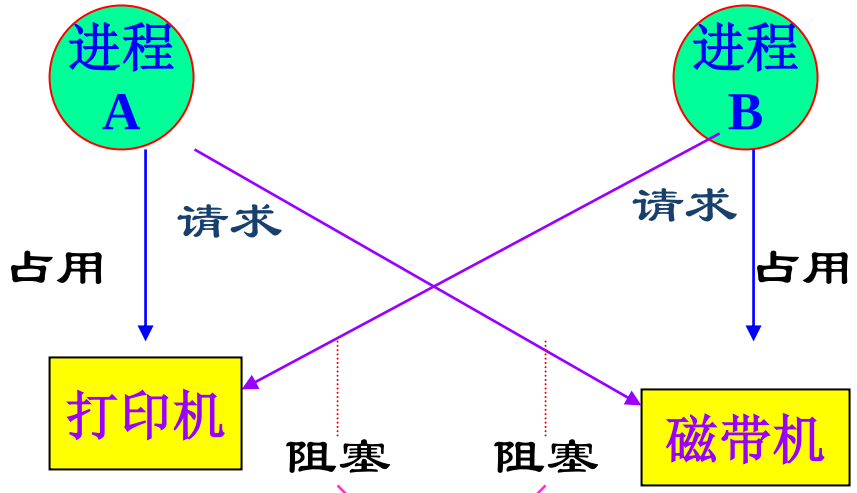
一个进程一次只能提出一个I/O请求，因而它不可能同时去操作多个I/O设备。这种方法的优点是：首先，它使程序的编制更为方便。例如，某程序发出一条命令要把一张卡片的内容读到主存中指定位置，则在下一条指令中就可认为卡片上的数据已存入指定位置。其次，这种方法对设备分配比较安全。由进程死锁判定可知，死锁的四个必要条件不可能具备(主要是“请求和保持”、“环路等待”条件不满足)。

其缺点是，进程进展缓慢，CPU和I/O设备之间是串行工作的。这就是所谓的单请求方式。

三、设备分配中的安全性

为了能同时操作多个I/O设备以加速进程的推进，应使得某进程以命令形式发出I/O请求后，仍可继续运行，需要时又可发出第二个、第三个I/O请求，仅当进程所请求之设备已为其它进程占用时，才进入阻塞状态。这样，一个进程就可同时操作多个外部设备。

多请求方式的缺点是，设备分配不安全。



因为它具备请求和保持条件，因而有可能产生死锁的情况。

7.4.3 设备独立性

一、设备独立性的概念

为了提高系统的可适应性和可扩展性，我们希望所编制的用户程序与实际使用的物理设备无关，这就是所谓与设备的独立性（无关性）。为此，我们将逻辑设备与物理设备区分，并引入之逻辑设备名称和物理设备名称的概念：

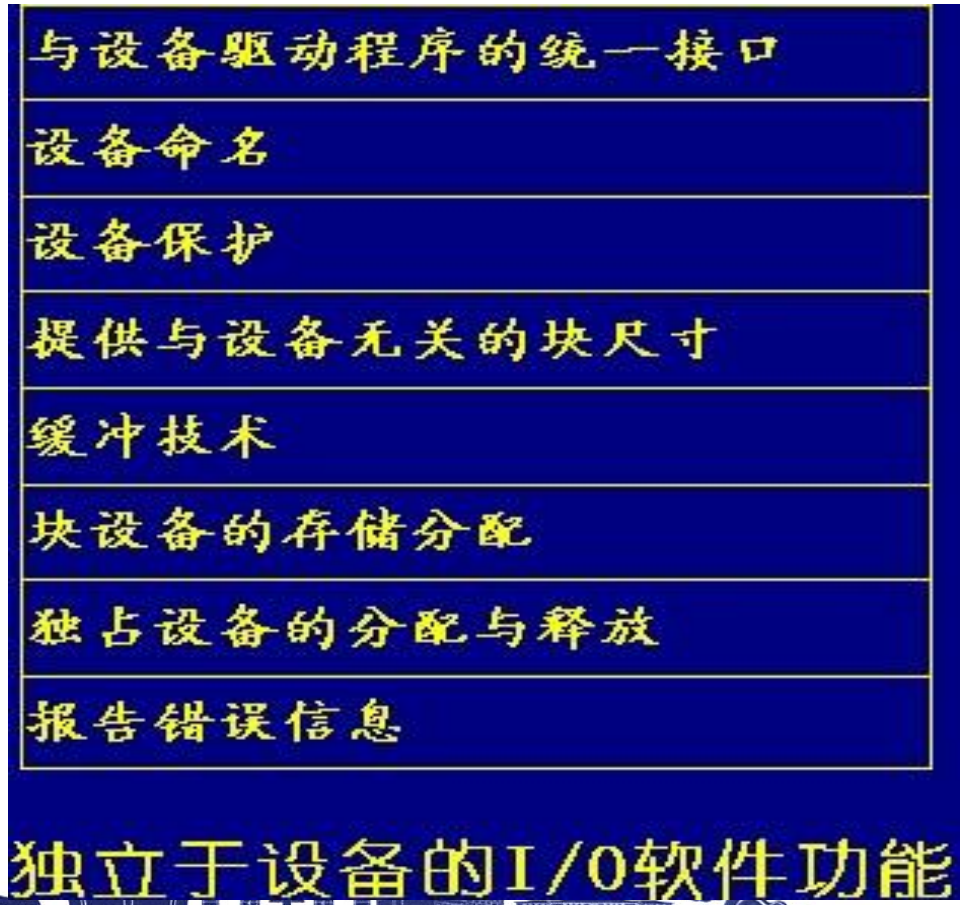
- 进程在实际执行时使用的一定是具体的物理设备
- 用户程序中应避免直接使用物理设备名称，而应使用逻辑设备名称

在实现设备独立性的功能后，有两个好处：

- 设备分配时的灵活性
- 易于实现I/O设备重定向

二、设备独立性软件

为了实现设备独立性，在设备驱动程序之上设置一层软件，称为设备独立性软件。设备驱动程序与设备独立软件之间的确切界限是依赖于具体的系统。主要功能有：



7.4.4 独占设备的分配程序

一、基本的设备分配程序

当系统中已经设置前述的数据结构，且确定一定的分配原则后，如果某进程提出了I/O请求，便可按下述步骤进行设备分配：

1.分配设备：

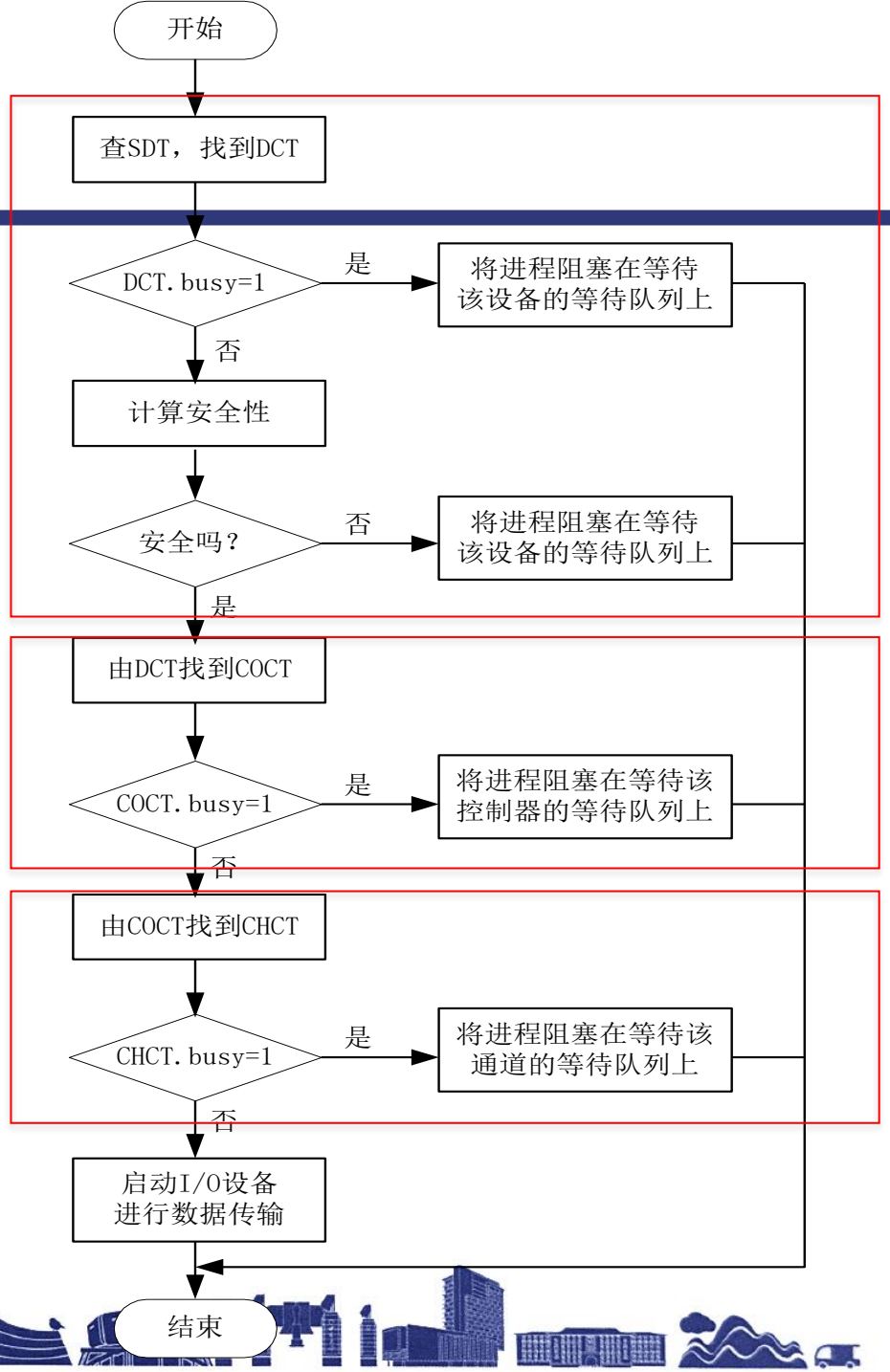
- 根据与进程n提出的逻辑设备名称对应的物理设备名称去检索系统设备表SDT，从中找到该物理设备之设备控制表DCT。
- 根据DCT中的状态信息Busy了解到该设备是否忙碌。若忙，让申请者待；若设备空闲，系统便按一定算法计算分配设备的安全性；安全时分配设备给请求者。

7.4.4 独占设备的分配程序

2.分配控制器：从设备控制表DCT中的控制器表指针COCT ptr，可得知与此设备连接的控制器的COCT，通过检查该表中的状态信息Busy来判别控制器是否空闲，若忙，将要求I/O的进程插入等待该控制器的队列COL中。

3.分配通道：如果控制器也空闲，则把控制器分配给它。之后，再通过控制器表COCT中的通道表指针CHCT ptr，检查与此控制器连接的通道忙否？若通道表中的状态信息Busy为忙，则把该进程插入等待通道的队列CHL中；若通道空闲，则表明本次I/O请求所需之I/O设备开始数据传送。

设备分配程序

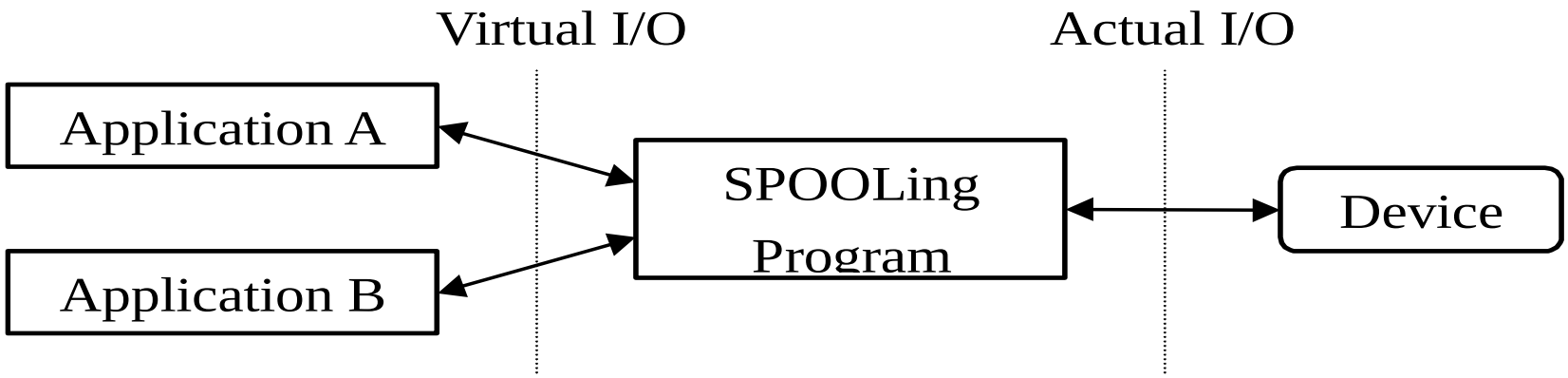


单通路情况下的设备分配程序

7.4.5 SPOOLing技术

利用假脱机技术(SPOOLing, Simultaneous Peripheral Operation On Line, 也称为虚拟设备技术)可把独享设备转变成具有共享特征的虚拟设备，从而提高设备利用率。

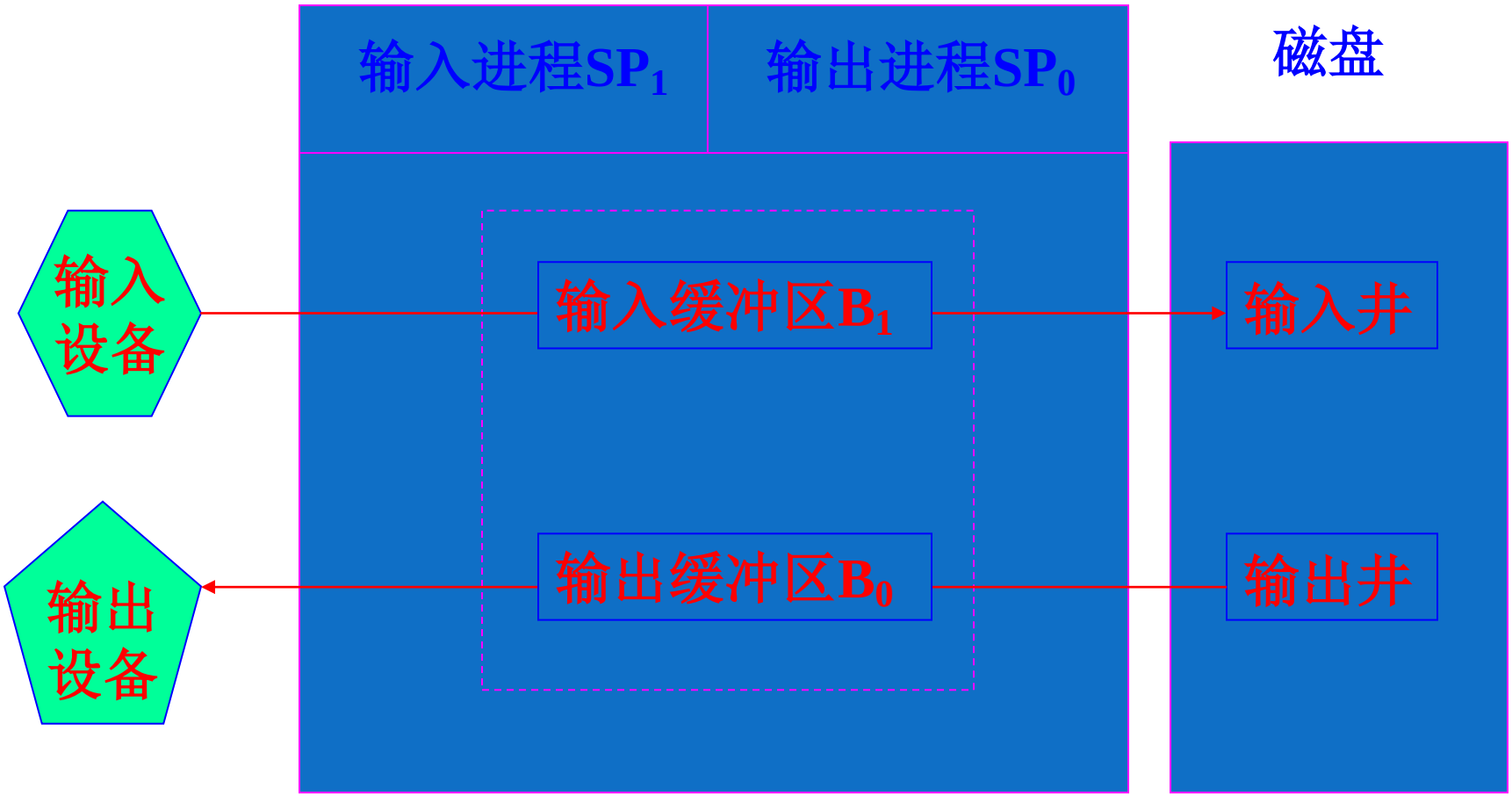
- **引入：在多道批处理系统中，专门利用一道或几道程序来完成对设备的I/O操作。无需使用外围I/O处理机。**



假脱机的原理

- SPOOLing程序和外设进行数据交换，可以称为“实际I/O”。一方面，SPOOLing程序预先从外设输入数据并加以缓冲，在以后需要的时候输入到应用程序；另一方面，SPOOLing程序接受应用程序的输出数据并加以缓冲，在以后适当的时候输出到外设。
- 在SPOOLing程序中，需要管理两级缓冲区：内存缓冲区和磁盘上的缓冲池，后者可以暂存多批I/O操作的较多数据。
- 应用程序进行I/O操作时，只是和SPOOLing程序交换数据，可以称为"虚拟I/O"。这时虚拟I/O实际上是从SPOOLing程序的缓冲池中读出数据或把数据送入缓冲池，而不是跟实际的外设进行I/O操作。

SPOOLing系统的组成



SPOOLing系统的组成

- **优点：**
  - **提高了I/O速度**：应用程序的虚拟I/O比实际I/O速度提高，缩短应用程序的执行时间。另一方面，程序的虚拟I/O操作时间和实际 I/O操作时间分离开来。
  - **实现对独享设备的共享**：由SPOOLing程序提供虚拟设备，可以对独享设备共享使用。
  - **实现了虚拟设备功能**
- **举例：打印机设备和可由打印机管理器管理的打印作业队列。**
  - 如：Windows NT中，应用程序直接向针式打印机输出需要15分钟，而向打印作业队列输出只需要1分钟，此后用户可以关闭应用程序而转入其他工作，在以后适当的时候由打印机管理器完成15分钟的打印输出而无需用户干预。

7.5 设 备 处 理

设备处理程序即设备驱动程序（进程），它是设备控制器与I/O进程之间的通信程序。它通常是用汇编语言或系统编程语言写的。

主要任务：

⑴ 接收上层软件发来的抽象要求，再将其转换为具体要求后，发送给设备控制器，启动设备去执行。

⑵ 将由设备控制器发来的信号传送给上层软件。

由于设备的种类的不同，应为每一类设备配置一种驱动程序，有时也可为非常相似的两类设备配置一个驱动程序。

7.5.1 设备驱动程序的功能和特点

一、设备驱动程序的功能

⑴将接收到的抽象要求转换为具体要求。

⑵检查用户I/O请求的合法性，了解I/O设备的状态，传递有关参数、设置设备的工作方式。

⑶发出I/O命令，启动分配到的I/O设备，完成指定的I/O操作。

⑷及时响应由控制器或通道发来的中断请求，并根据中断类型调用相应的中断处理程序进行处理。

⑸对于设置有通道的计算机系统，驱动程序还应能够根据用户的I/O请求，自动地构成通道程序。

二、设备驱动程序的特点

- 驱动程序主要是指在请求I/O的进程与设备控制器之间的一个通信和转换程序。
- 驱动程序与设备控制器和I/O设备的硬件特性紧密相关，因而对不同类型的设备应配置不同的驱动程序。
- 驱动程序与I/O设备所采用的I/O控制方式紧密相关。
- 由于驱动程序与硬件紧密相关，因而其中的一部分必须用汇编语言书写。

7.6 磁盘存储器管理

磁盘存储在现代计算机系统中，都配备了磁盘存储器，并以它为主存放文件。

磁盘存储器管理的主要任务是：

⑴为文件分配必要的存储空间；

⑵合理地组织文件的存取方式；

⑶提高磁盘存储空间的利用率；

⑷提高对磁盘的I/O速度，以改善文件系统的性能；

⑸采取必要的冗余措施，来确保文件系统的可靠性。

由于随机存取的文件都是存放在磁盘上，因此，如何改善磁盘I/O的性能．已成为提高文件系统性能的关键。

从存储角度，与内存比较起来，磁盘有三个主要的优点：

1. 可用的存储容量非常大
2. 每位的价格非常低
3. 电源关掉后信息不会丢失

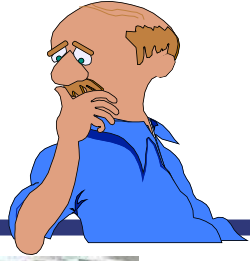
提高磁盘I/O速度的主要途径有：

（l）选择性能好的磁盘

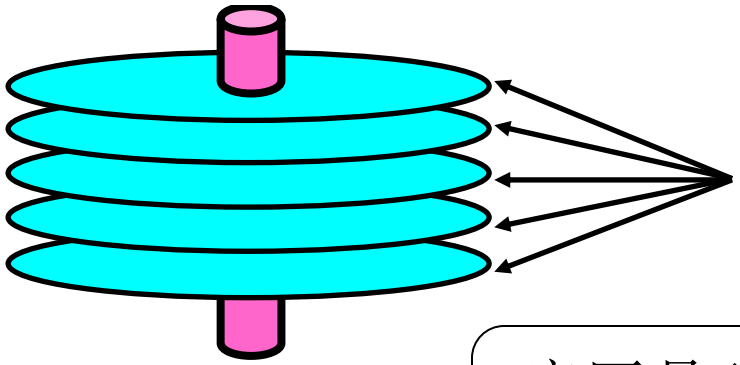
（2）采用好的磁盘调度算法

（3）设置磁盘高速缓冲

一、认识一下磁盘

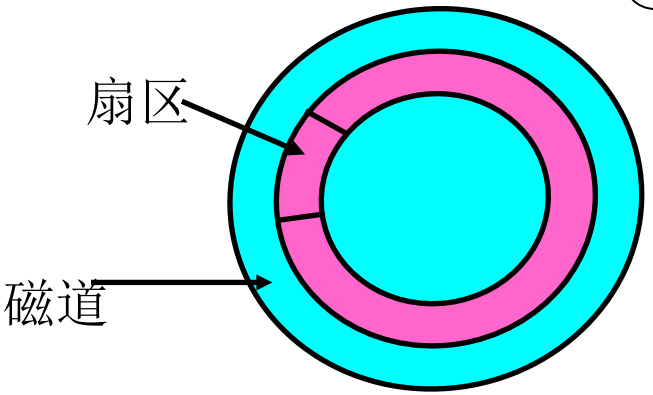


- 画一个示意图:



- 看看俯视图:

扇区是磁盘的寻址单位、访问单位

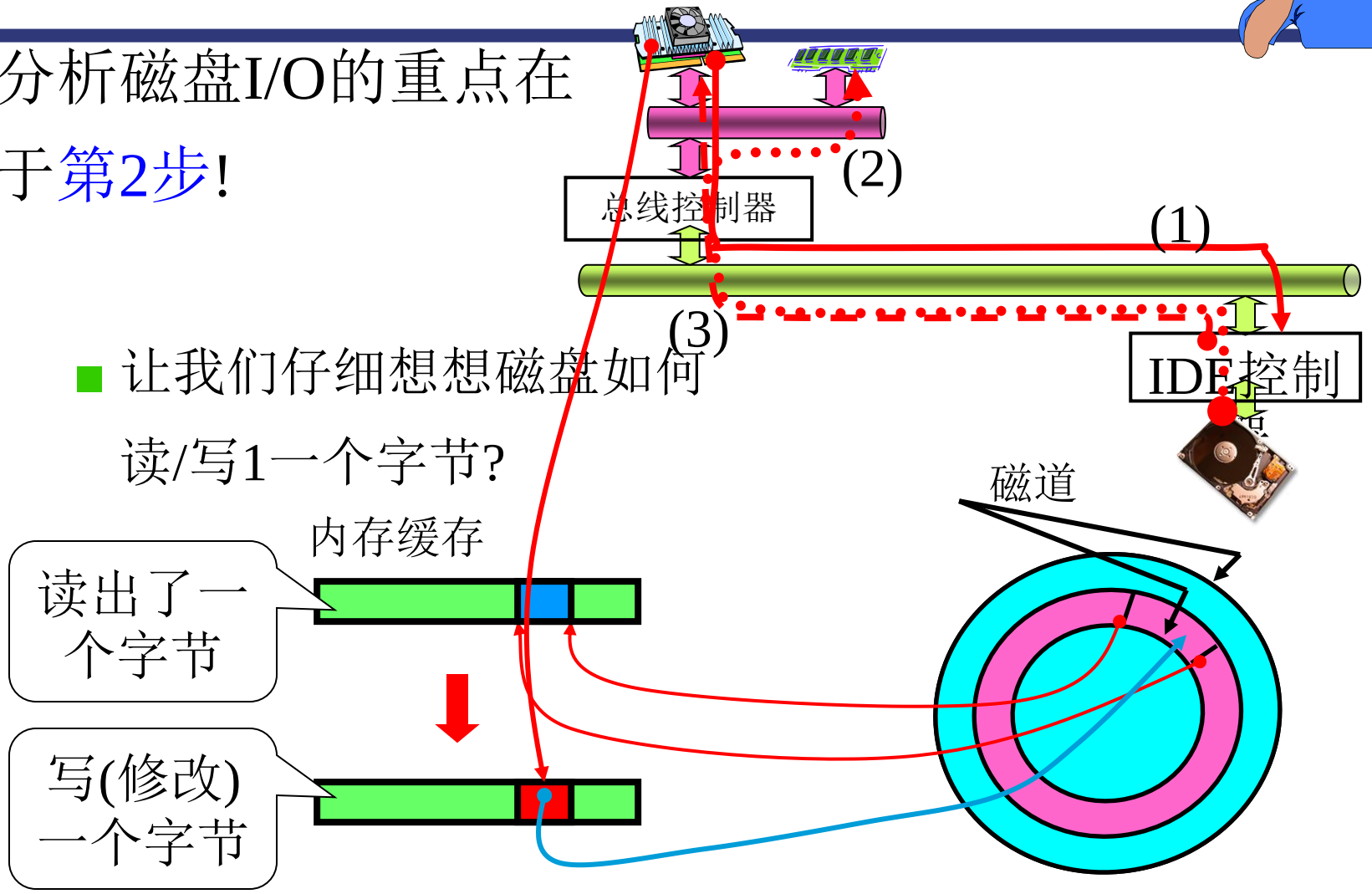


- 磁盘的数据单位是扇区
- 扇区大小：512字节
- 扇区的大小是传输时间和碎片浪费的折衷

- 所以，磁盘被称为块设备!

磁盘的I/O

- 分析磁盘I/O的重点在于第2步!
  - 让我们仔细想想磁盘如何读/写1一个字节?



- 磁盘I/O: 缓存队列→控制器→寻道→旋转→传输!

二、磁盘的类型

对磁盘可从不同的角度进行分类：

- 硬盘和软盘
- 单片盘和多片盘
- 固定头磁盘和活动头磁盘

1）固定头磁盘

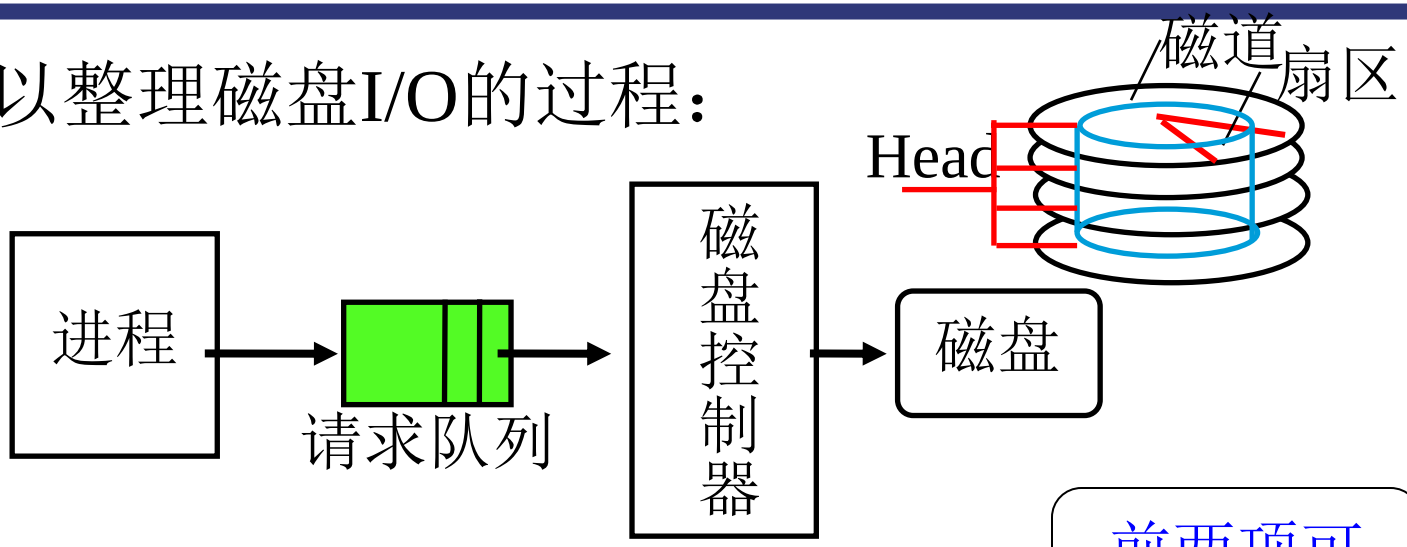
这种磁盘在每条磁道上都有一个读/写磁头，有效地提高了磁盘的I/O速度。

2）移动头磁盘

每一个盘面仅配有一个磁头，也被装入磁臂中，为能访问该盘面上的所有磁道，该磁头必须能移动以进行寻道。

三、磁盘访问时间

- 可以整理磁盘I/O的过程：



我们最关心的磁盘什么时候读/写完?

前两项可以忽略!

磁盘访问延迟 = 队列时间 + 控制器时间 + 寻道时间 + 旋转时间 + 传输时间

把磁臂(磁头)从当前位置移动到指定磁道上所需时间

是指定扇区移动到磁头下面所经历的时间

是指把数据从磁盘读出，或向磁盘写入数据所经历的时间。

- 关键所在：最小化寻道时间和旋转延迟!

7.6.2 磁盘调度

- 磁盘调度：

前两项可以忽略!

磁盘访问延迟 = 队列时间 + 控制器时间 +
寻道时间 + 旋转时间 + 传输时间

T_s: 10 ms to 5 ms

T_r: 10 ms to 8 ms

$T_t = b/(rN)$

- 多个磁盘访问请求出现在请求队列怎么办？ **调度**
- 调度的目标是什么？调度时主要考察什么？

目标当然是平均访问延迟小!

寻道时间是主要矛盾!

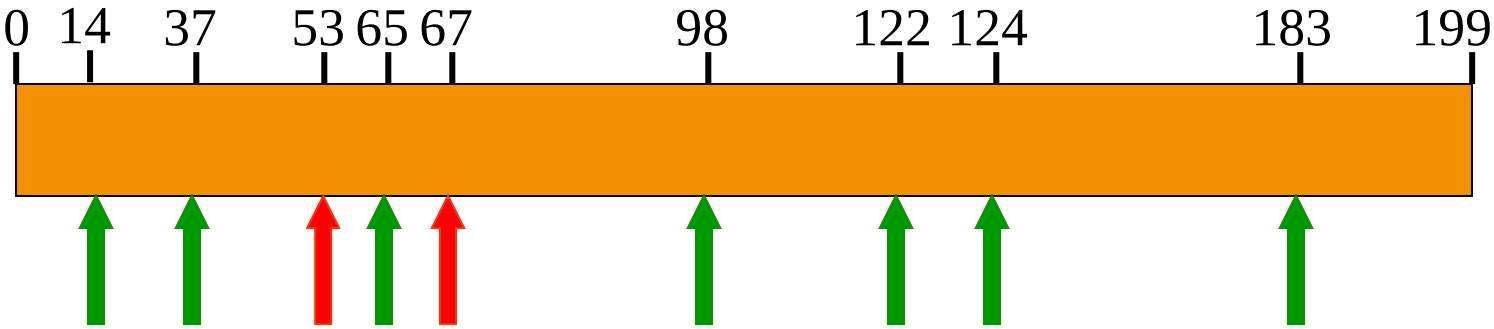
- **磁盘调度：输入多个磁道请求，给出服务顺序!**

一、FCFS磁盘调度

- 最直观、最公平的调度：
  - 一个实例: 磁头开始位置=53；

请求队列=98, 183, 37, 122, 14, 124, 65, 67

FCFS: 磁头共移动640磁道!



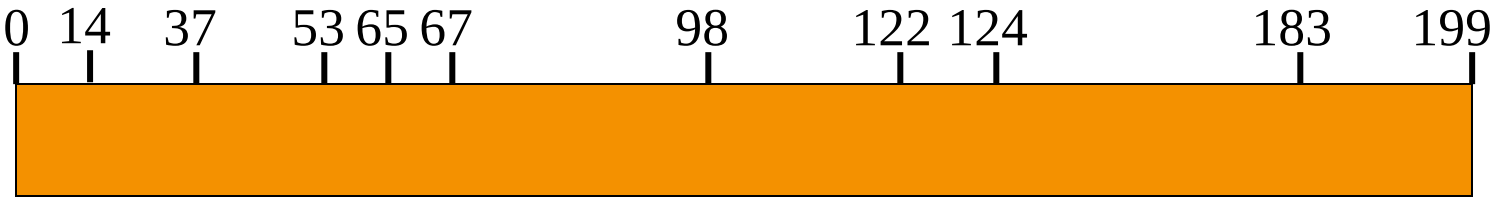
在移动过程中把经过的请求处理了!

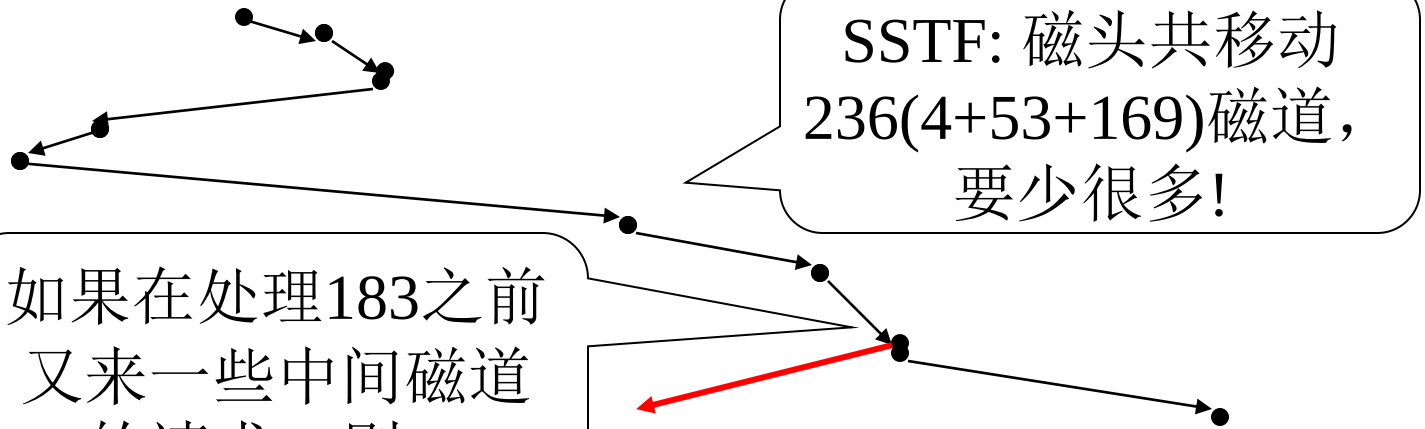
磁头在长途奔袭!

二、SSTF磁盘调度

- Shortest-seek-time First：
  - 继续该实例: 磁头开始位置=53；

    请求队列=98, 183, 37, 122, 14, 124, 65, 67



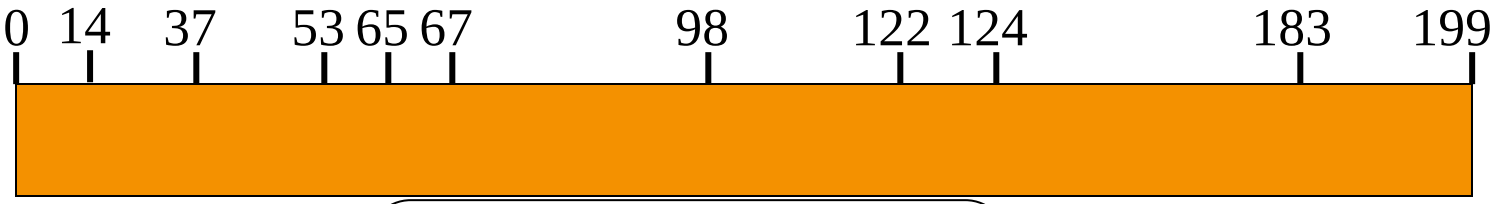


  - SSTF存在饥饿问题

三、SCAN磁盘调度

- SSTF+中途不回折：每个请求都有处理机会
  - 继续该实例: 磁头开始位置=53；

    请求队列=98, 183, 37, 122, 14, 124, 65, 67



SCAN: 磁头共移动53+183=236磁道，和SSTF一样!

这些请求的等待时间较长，只因所在方向不够幸运!“申”不逢时啊！

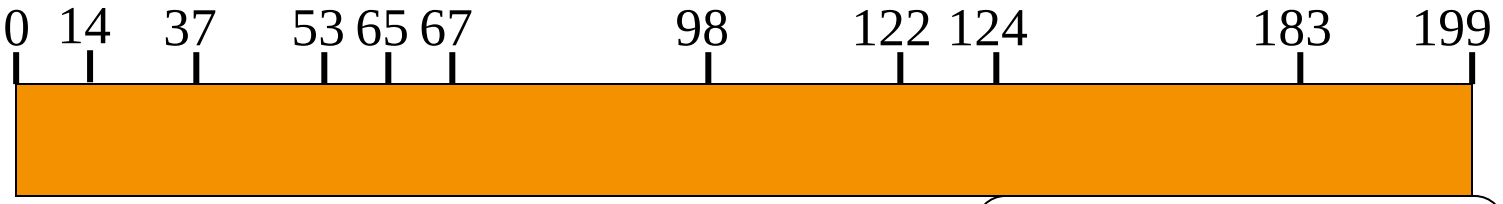
根据其特征，SCAN也被称为电梯算法!

- SCAN导致延迟不均

四、C-SCAN磁盘调度

- SCAN+直接移到另一端：两端请求都能很快处理
  - 继续该实例: 磁头开始位置=53;

    请求队列=98, 183, 37, 122, 14, 124, 65, 67



CSCAN中的C是环的意思!

CSCAN: 磁头共移动188+200磁道!
其中200会较快!

- 14→0(183→199)没有必要

例1：假定某移动磁盘上，处理了访问56号柱面的请求后，现在正在70号柱面上读信息，目前有下面的请求访问磁盘柱面的序列：

73, 68, 100, 120, 60, 108, 8, 50

请写出下列各小题的响应的次序及移动的柱面数。

⑴用最短查找时间优先算法

⑵电梯调度算法

解：⑴用SSTF，响应的次序为

70、68、73、60、50、8、100、108、120

移动的柱面数：2+5+13+10+42+92+8+12=184

⑵用电梯调度算法，响应的次序为

70、73、100、108、120、68、60、50、8

移动的柱面数：3+27+8+12+52+8+10+42=162

例2： 若干个等待访问磁盘者依次要访问的柱面为 20, 44, 40, 4, 80, 12, 76。 假设每移动一个柱面需要3毫秒时间，移动臂当前位于40号柱面，请按下列算法分别计算为完成上述各次访问总共花费的寻找时间。

(1)先来先服务算法： (2)最短寻找时间优先算法。

(3)电梯调度算法（分往数小和往数大的方向）

解：（1）先来先服务算法：

40 → 20 → 44 → 40 → 4 → 80 → 12 → 76

（20） （24） （4） （36） （76） （68） （64）

共移动292柱面

(20+24+4+36+76+68+64)*3=292*3=876 ms

（2）最短寻找时间算法：

40 →40 → 44 → 20 → 12 → 4 → 76 → 80

（0） （4） （24） （8） （8） （72） （4）

共移动120柱面

(0+4+24+8+8+72+4)*3=120*3=360 ms

（3）电梯调度算法：

由于未指明开始移动的方向，分成两种情形：

数小的方向：

40 → 40 → 20 → 12 → 4 → 44 → 76 → 80

（0） （20） （8） （8） （40） （32） （4）

共移动112柱面

(0+20+8+8+40+32+4)*3=112*3=336 ms

数大的方向

40 → 40 → 44 → 76 → 80 → 20 → 12 → 4

（0） （4） （32） （4） （60） （8） （8）

共移动116柱面

(0+4+32+4+60+8+8)*3=116*3=348 ms

7.6.3 磁盘高速缓存(Disk Cache)

1. 磁盘高速缓存的形式

利用内存中的存储空间，来暂存从磁盘中读出的一系列盘块中的信息。高速缓存在内存中可分成两种形式：

- 在内存中开辟一个单独的存储空间来作为磁盘高速缓存，其大小是固定的，不会受应用程序多少的影响；
- 把所有未利用的内存空间变为一个缓冲池，供请求分页系统和磁盘I/O时(作为磁盘高速缓存)共享。

2. 数据交付方式

系统可以采取两种方式，将数据交付给请求进程：

- 数据交付。这是直接将高速缓存中的数据，传送到请求者进程的内存工作区中。
- 指针交付。只将指向高速缓存中某区域的指针， 交付给请求者进程。

后一种方式由于所传送的数据量少，因而节省了数据从磁盘高速缓存存储空间到进程的内存工作区的时间。

3. 置换算法

由于请求调页中的联想存储器与高速缓存(磁盘I/O中)的工作情况不同，因而使得在置换算法中所应考虑的问题也有所差异：

- 访问频率。
- 可预见性。
- 数据的一致性。

4. 周期性地写回磁盘

- 在UNIX中会强制性地将所有在高速缓存中已修改的盘块数据写回磁盘。一般是把两次调用SYNC的时间间隔定为30s。
- 在MS-DOS中所采用的方法是：只要高速缓存中的某盘块数据被修改，便立即将它写回磁盘，并将这种高速缓存称为“写穿透、高速缓存”(write-through cache)

7.6.4 提高磁盘I/O速度的其它方法

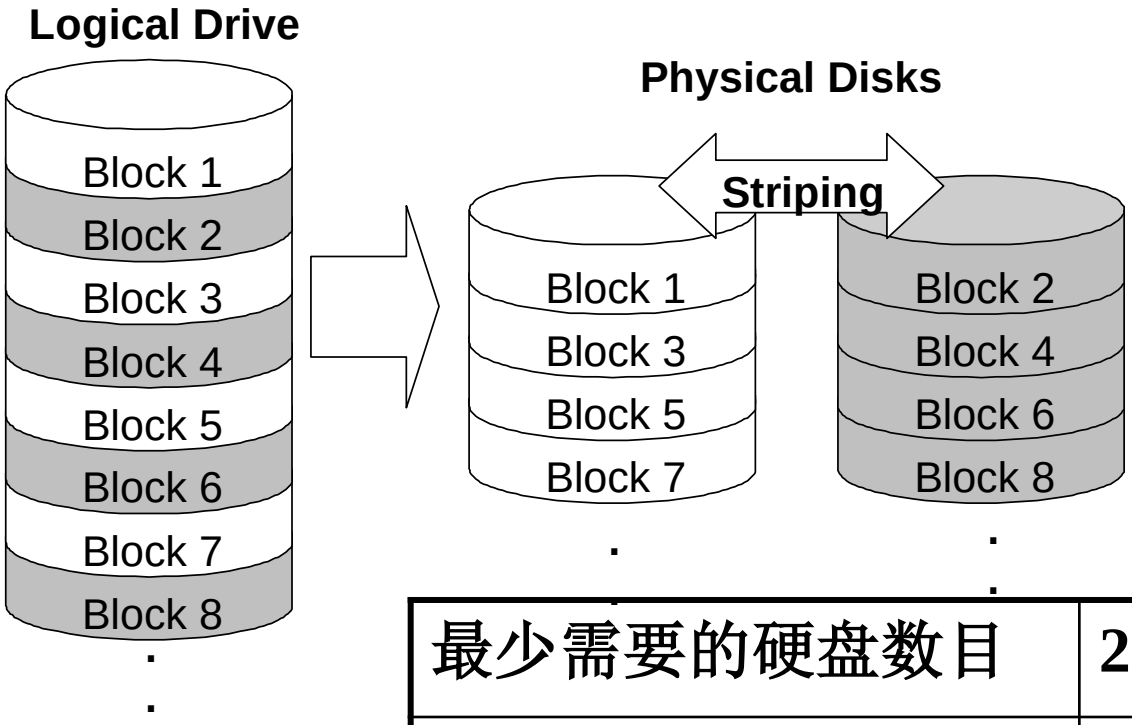
1. 提前读 (Read-Ahead)
2. 延迟写 (Lazy Writing)
3. 优化物理块的分布
4. 虚拟盘

7.6.5 RAID 廉价冗余磁盘阵列 Redundant Arrays of Inexpensive Disks

- **采用若干硬磁盘驱动器按照一定要求组成一个整体，整个磁盘阵列由阵列控制器管理。**
- **可以提高**高磁盘的访问速度和可靠性，**一共有7级，RAID0-RAID6。**
- **Parity是应用于RAID中的另一种冗余技术**
  - 比如你的一个数据单位有x位数字，那么你可以使用这x位数字产生一个奇偶校验位，并且把这个奇偶校验位作为这个数据单位的第x＋1位，如果这x＋1位中的任何一个丢失，剩下的x位仍能修复这个数据

RAID级别

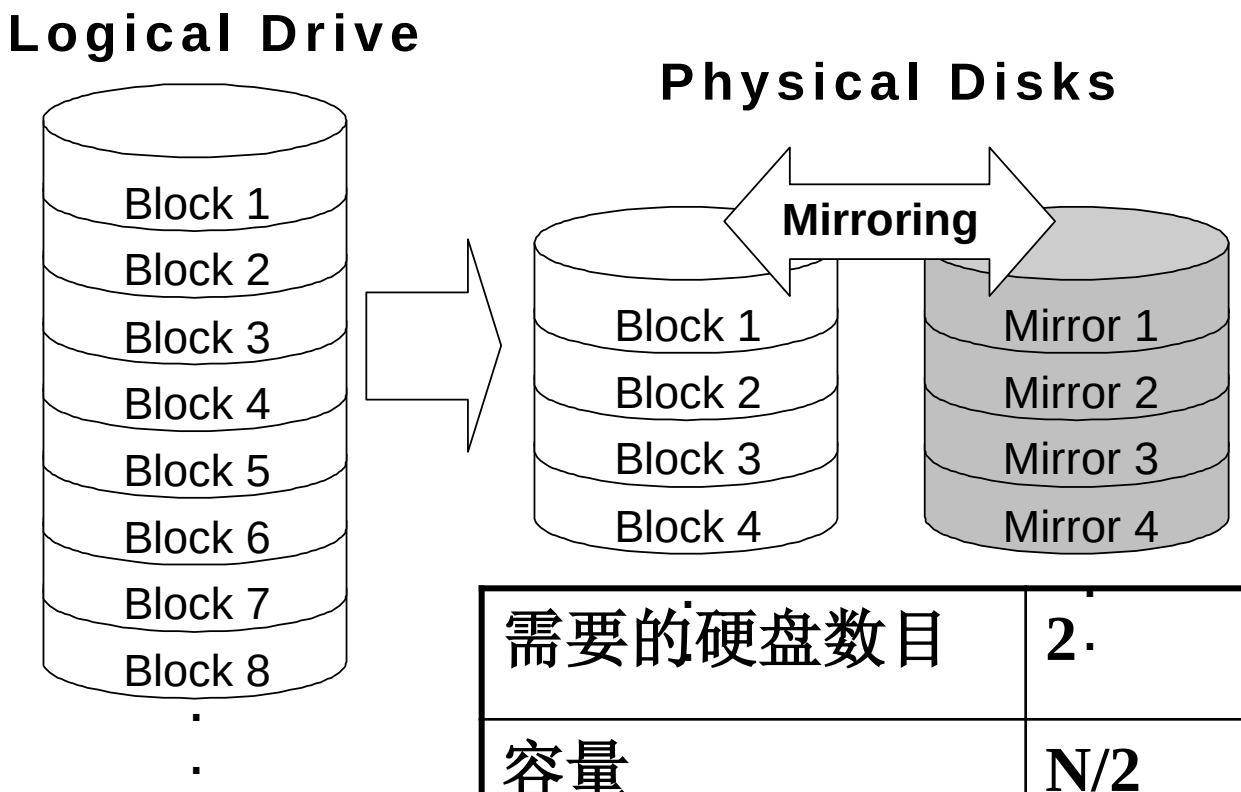
- **RAID 0 带区集(Striping Data)**



最少需要的硬盘数目	2
容量	N
冗余	NO

RAID级别

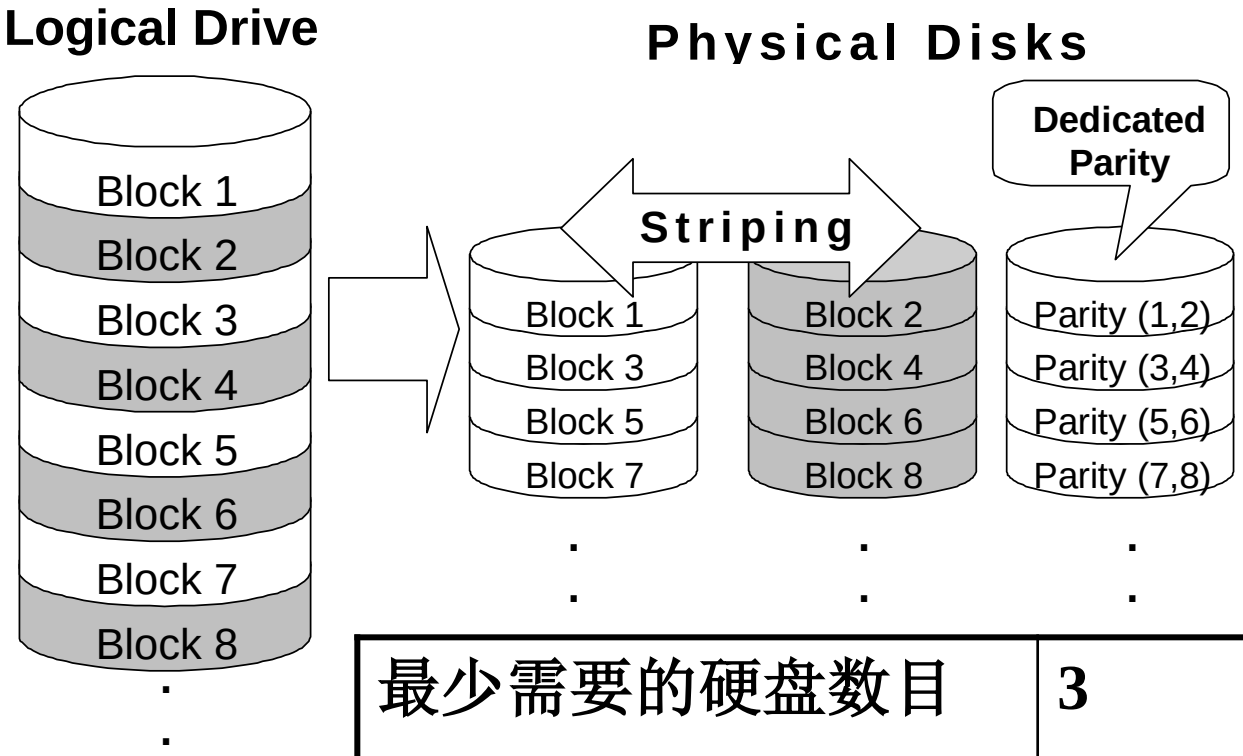
- **RAID 1** 镜像集



需要的硬盘数目	2
容量	N/2
冗余	YES

RAID级别

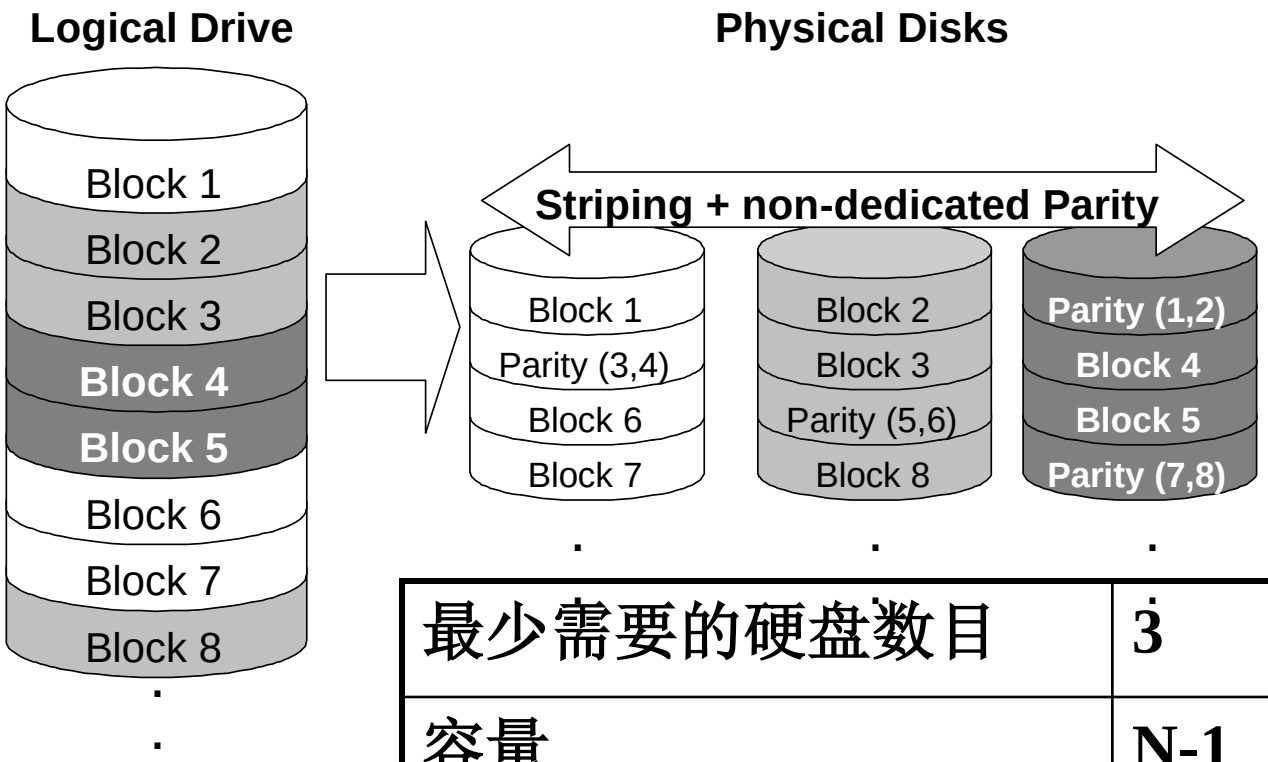
- **RAID**



最少需要的硬盘数目	3
容量	N-1
冗余	YES

RAID级别

- **RAID 5**



最少需要的硬盘数目	3
容量	N-1
冗余	YES

参考资料

- 廖剑伟，操作系统，西南大学，重庆，2023