Computer

操作系统原理与实践

第四章 设备管理

高等教育出版社

北京汉众信息科技有限责任公司策划

第四章 设备管理

- 目的与要求：
  - 掌握I/O控制的硬件基础和基本原理、设备的管理和使用方法。
  - 理解设备管理子系统的层次，功能及技术、了解磁盘设备。
- 重点与难点：
  - I/O控制方式
  - 设备使用方法
  - I/O软件层次结构
  - 设备驱动程序
  - 缓冲技术
  - 磁盘调度方法
  - 盘阵选择
- 作业： 2，5，9，10，14

第四章 设备管理

- 4.1 I/O硬件
- 4.2 I/O软件
- 4.3 存储设备

4.1 I/O硬件

- **4.1.1 I/O总线**
- **4.1.2 设备控制器**
- **4.1.3 直接存储器访问控制器**
- **4.1.4 I/O通道**
- **4.1.5 I/O设备**
- **4.1.6 I/O 控制方式**

4.1.1 I/O总线

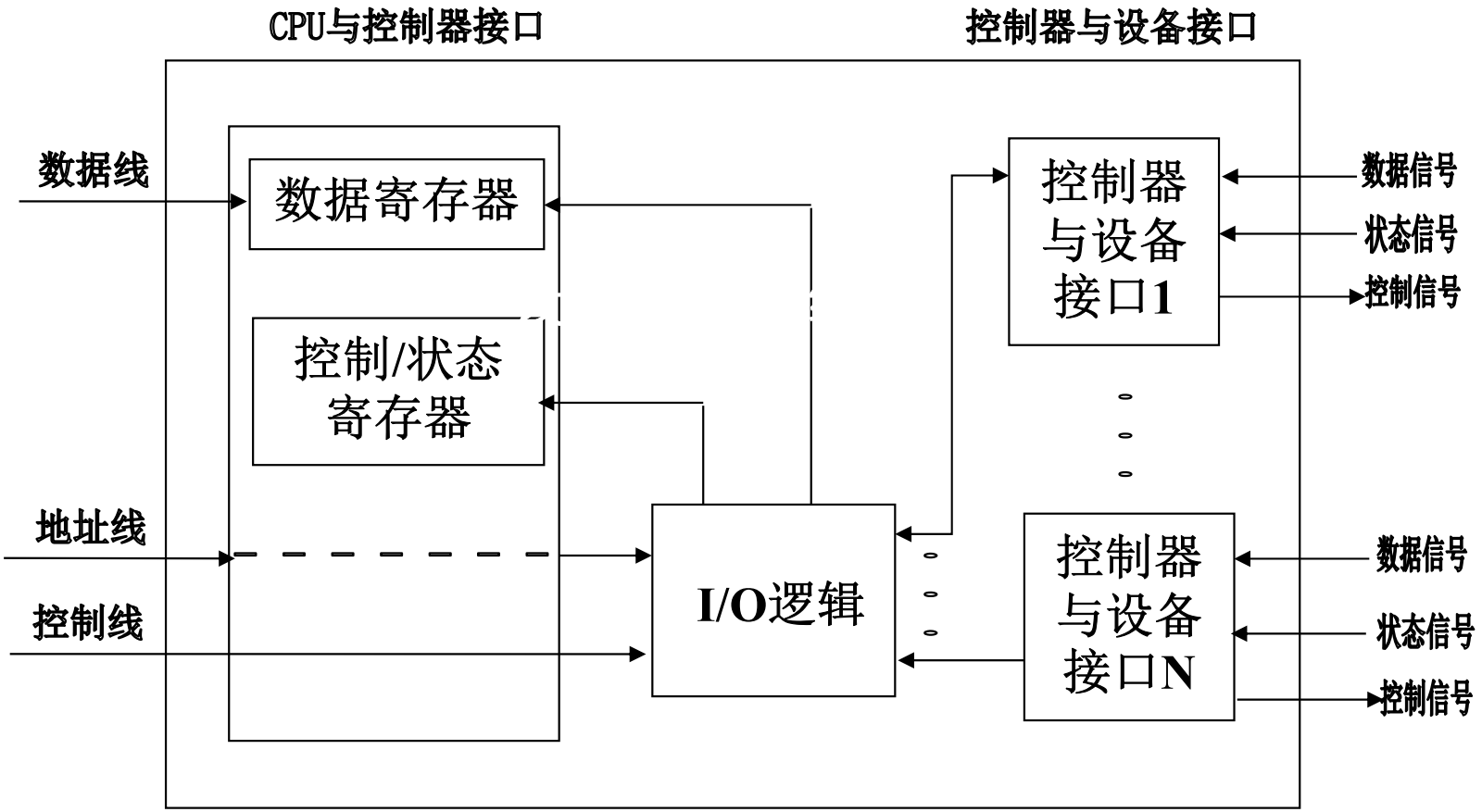
- 第一章所讲的总线的基本内容。
- **PCI**总线及其配置空间

4.1.2 设备控制器

- I/O设备通常包含一个机械部件和一个电子部件。电子部件被称作I/O部件或设备控制器。
- 操作系统一般只与控制器打交道，而非设备本身。
- 早期CPU是直接控制外部设备的，在引入I/O部件之后，I/O指令功能加强，才将CPU逐渐从与外设的交互细节中解放出来。

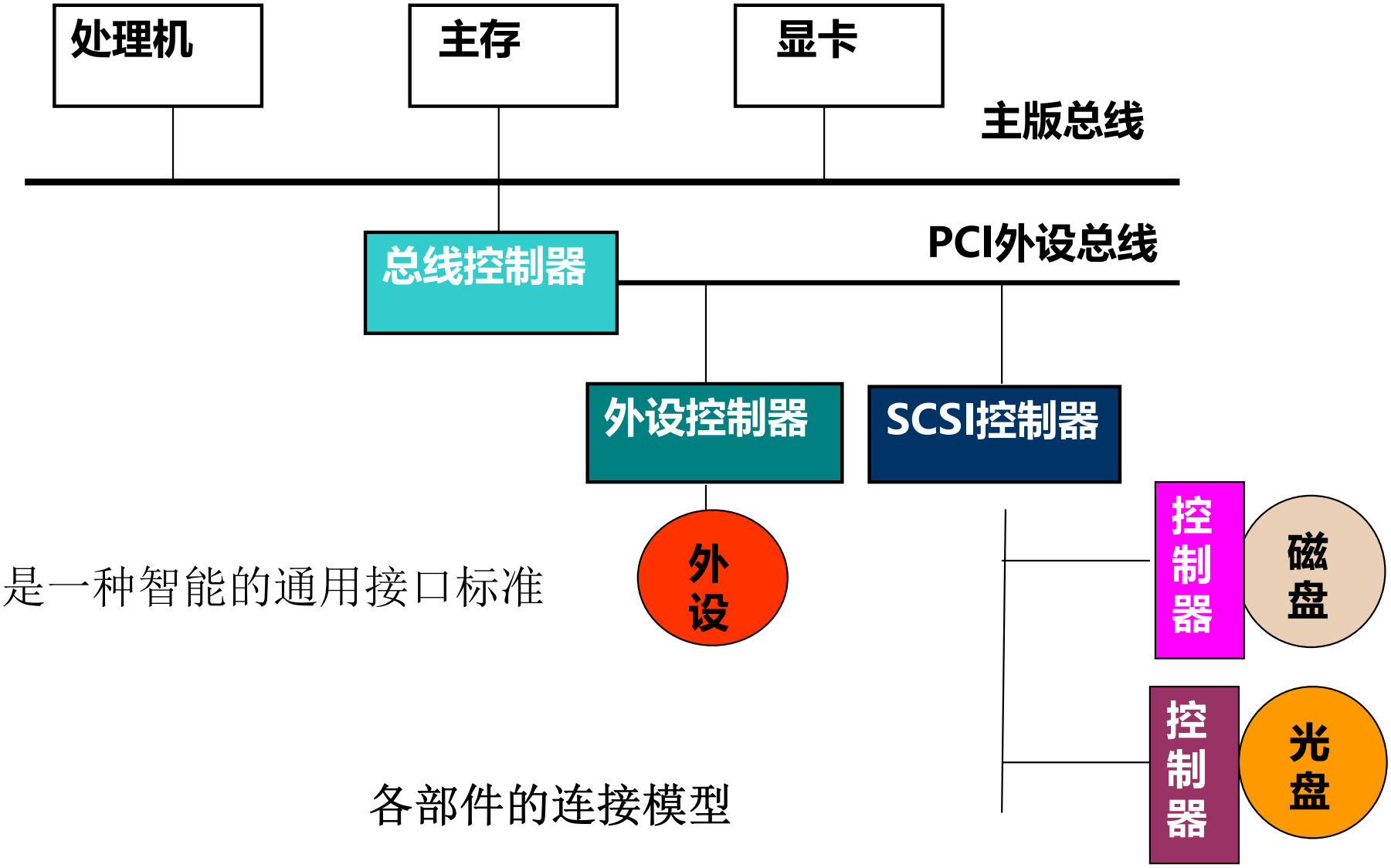
4.1.2 设备控制器

- **控制器的基本结构**



设备控制器的组成

4.1.2 设备控制器

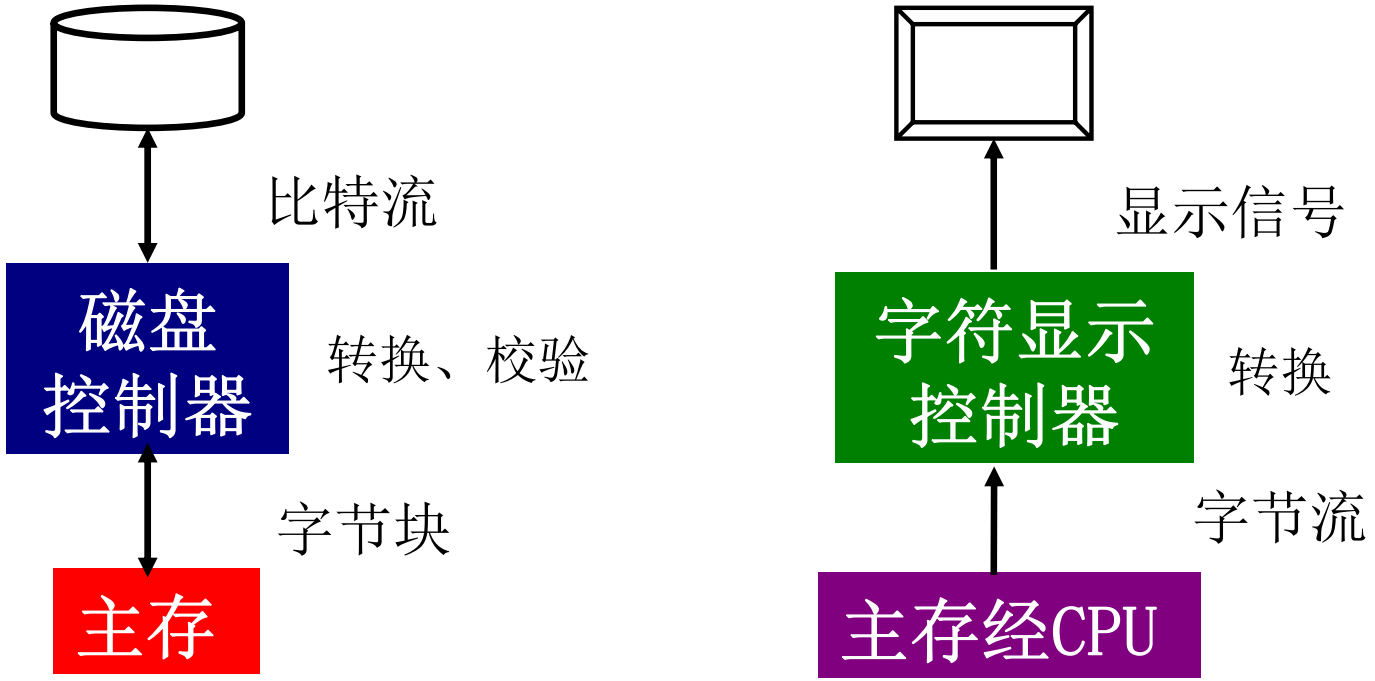


SCSI是一种智能的通用接口标准

各部件的连接模型

4.1.2 设备控制器

- 控制器的任务：
  - 在外部设备与内存（或CPU）之间完成比特流（或外部信号）和字节块（流）之间的转换。



4.1.2 设备控制器

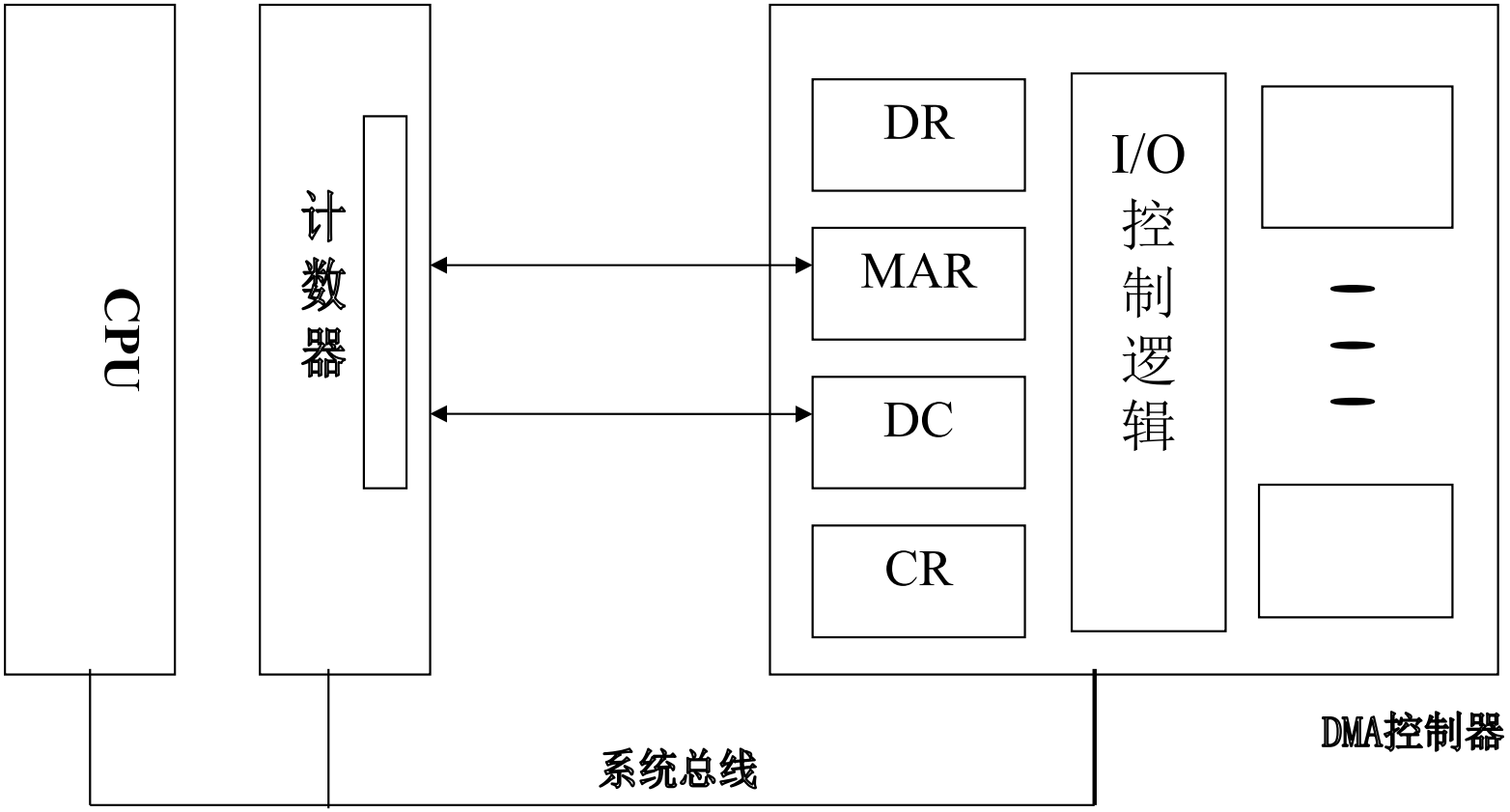
- 每个控制器都有一些用来与CPU通讯的I/O寄存器。操作系统通过向这些寄存器写命令字来实现I/O功能。

键盘	060 - 063
硬盘	320 – 32F
打印机	378 – 37F
软盘	3F0 – 3F7
彩色显示器	3D0 – 3DF

IBM PC的I/O地址

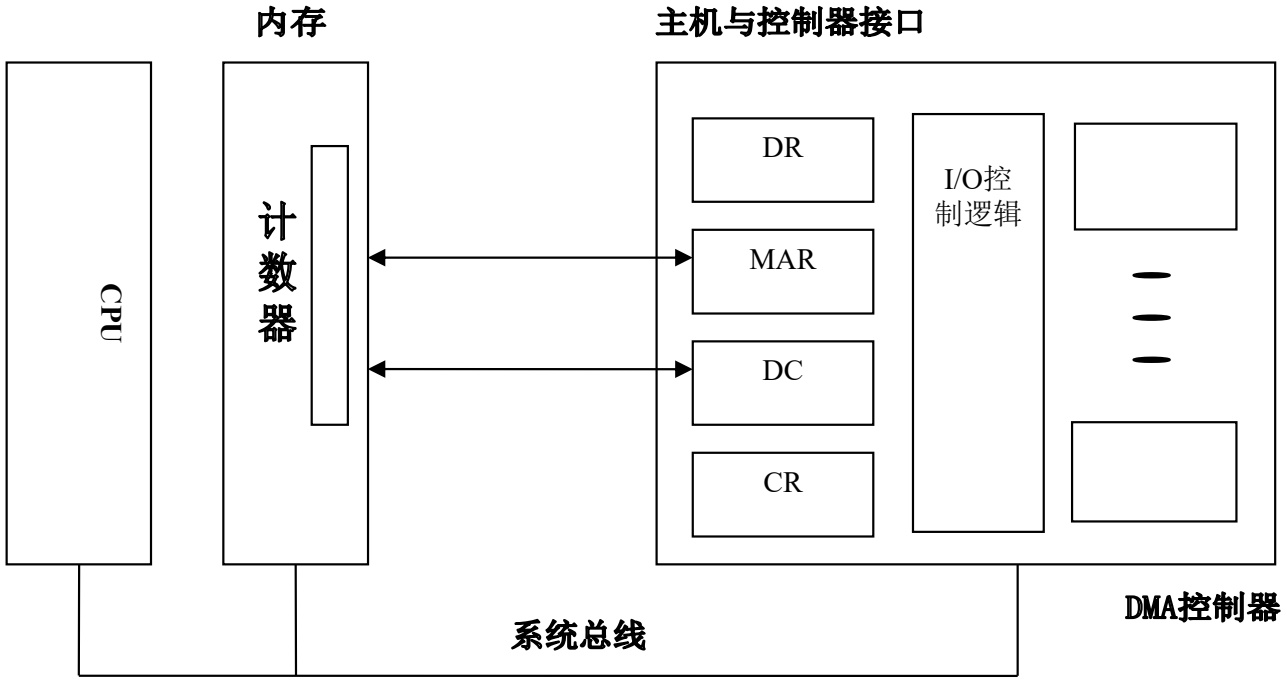
4.1.3 直接存储器访问控制器

- **DMA控制器内部结构及外部接口关系**



4.1.3 直接存储器访问控制器

- DMA的功能可以以独立的DMA部件在系统I/O总线上完成，也可整合到I/O部件中完成。
- 读写内存时，DMA部件需要控制总线，CPU可能在涉及存储访问时因此而忙等待。



4.1.4 I/O通道

- 定义：通道是独立于**CPU**的专门负责数据输入/输出传输工作的处理机，对外部设备实现统一管理，代替**CPU**对输入/输出操作进行控制，从而使输入，输出操作可与**CPU**并行操作。

  通道又称输入输出处理机，术语“通道”专指专门用来负责输入输出工作的处理机（简称Ｉ／Ｏ处理机）。比起中央处理机ＣＰＵ来，通道是一个比ＣＰＵ功能较弱、速度较慢、价格较为便宜的处理机。但“通道”一词在微型机中常指与ＤＭＡ或与Ｉ／Ｏ处理机相连设备的单纯的数据传送通路，它并不具有处理机的功能。

4.1.4 I/O通道

1. 引入通道的目的

为了使**CPU**从**I/O**事务中解脱出来，同时为了提高**CPU**与设备，设备与设备之间的并行工作能力

4.1.4　I/O通道

2.　通道类型

- 字节多路通道:通常按字节交叉的方式工作，适用于低速ＩＯ设备。
- 数组选择通道:按成组方式进行数据传输，适用于高速ＩＯ设备。
- 数组多路通道:综合前二个优点（数据传送率和通道利用率都较高）。

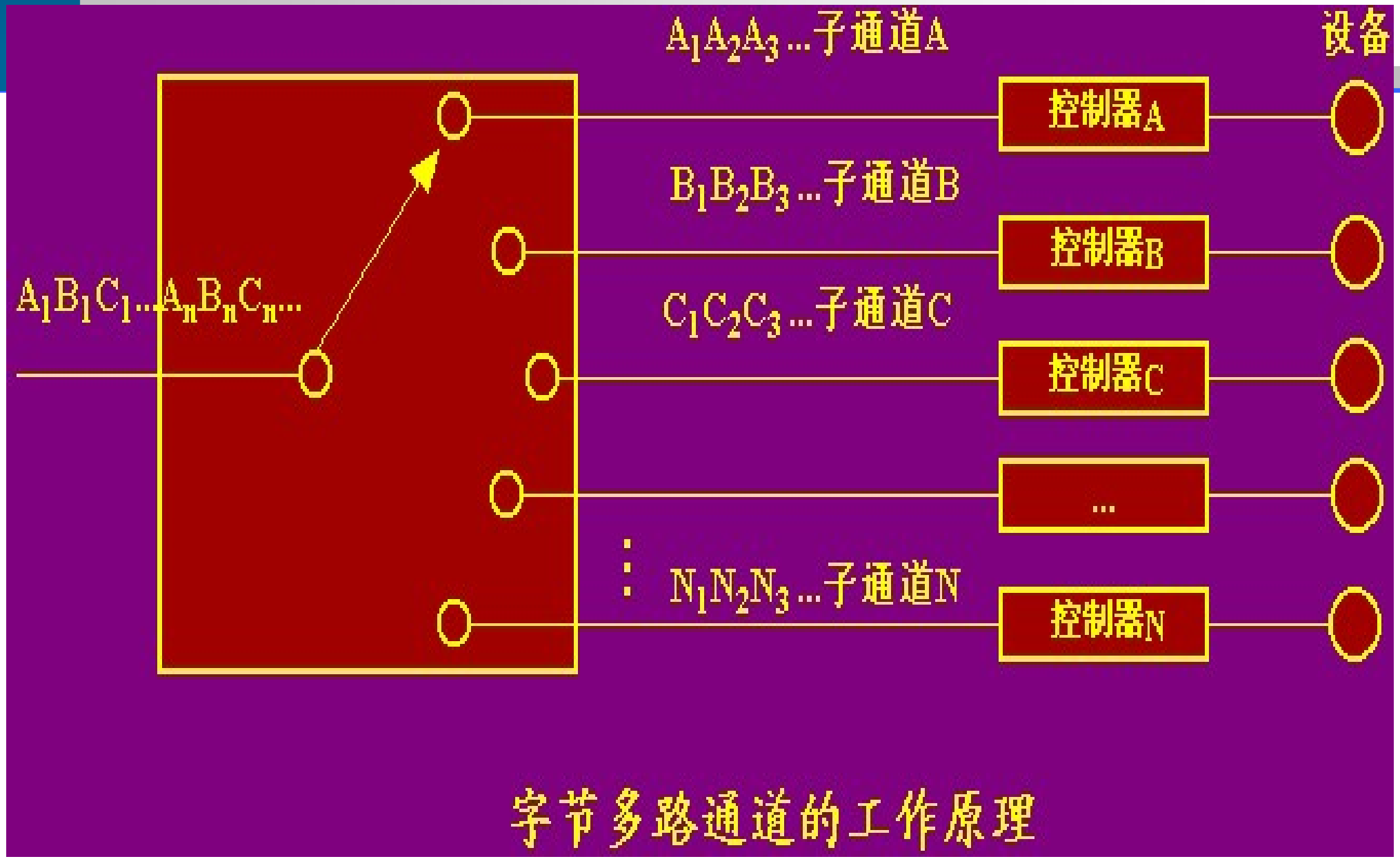
4.1.4　I/O通道

1）字节多路通道

字节多路通道以字节为单位传输信息，它可以分时地执行多个通道程序。当一个通道程序控制某台设备传送一个字节后，通道硬件就控制转去执行另一个通道程序，控制另一台设备传送信息。

主要连接以字节为单位的低速I/O设备。如打印机，终端。

以字节为单位交叉传输，当一台传送一个字节后，立即转去为另一台传送字节。



字节多路通道的工作原理

4.1.4 I/O通道

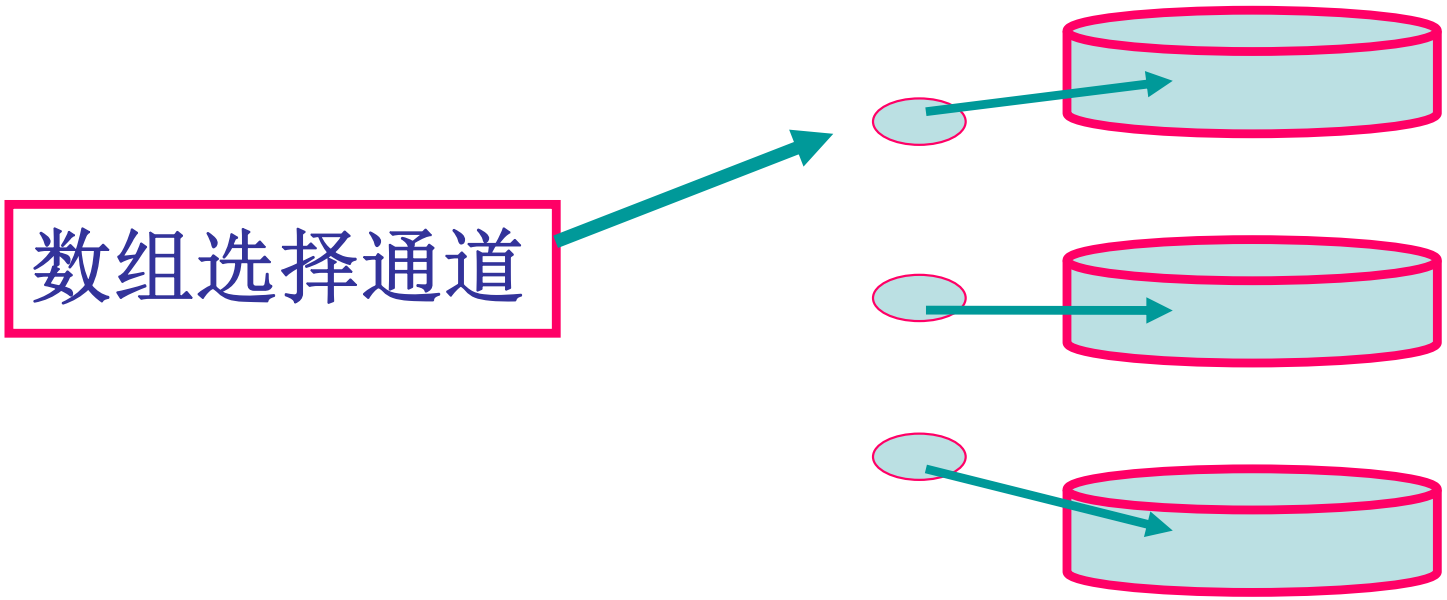
2) 数组选择通道

数组选择通道是以成组方式工作的，即每次传送一批数据，故传送速度很高。选择通道在一段时间内只能执行一个通道程序，只允许一台设备进行数据传输

当这台设备数据传输完成后，再选择与通道连接的另一台设备，执行它的相应的通道程序

主要连接磁盘，磁带等高速I/O设备

4.1.4　I/O通道



4.1.4 I/O通道

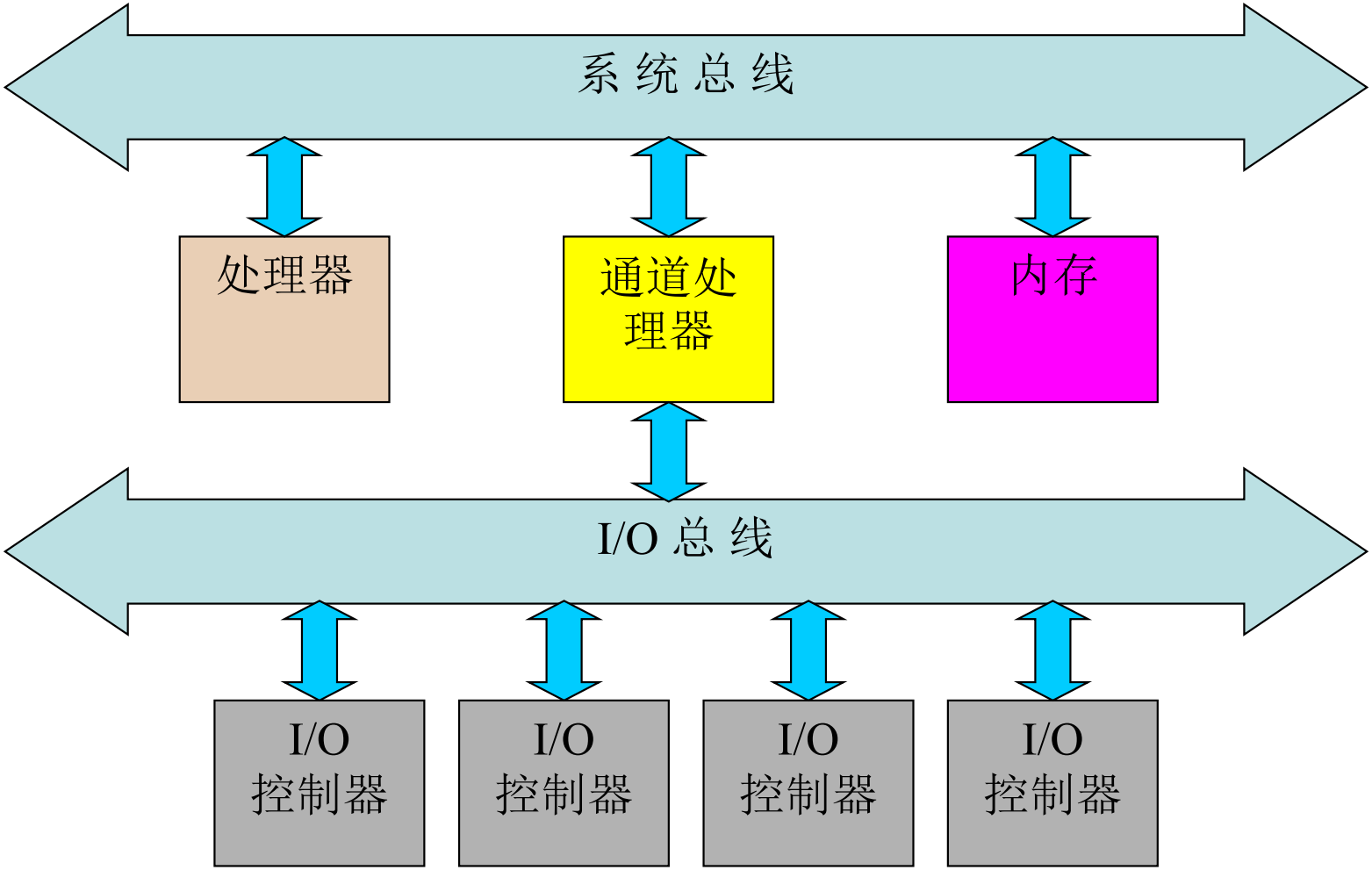
3)数组多路通道

数组多路通道结合了选择通道传送速度高和字节多路通道能进行分时并行操作的优点。它先为一台设备执行一条通道指令，然后自动转接，为另一台设备执行一条通道指令

主要连接高速设备

这样，对于连接多台磁盘机的数组多路通道，它可以启动它们同时执行移臂定位操作，然后，按序交叉地传输一批批数据。数据多路通道实际上是对通道程序采用多道程序设计的硬件实现

3. 硬件连接结构



4.1.4 I/O通道

4. 通道工作原理

通道相当于一个功能简单的处理机，包含通道指令（空操作，读操作，写操作，控制，转移操作），并可执行用这些指令编写的通道程序

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

2）通道命令

用于I/O操作的命令主要有两种：

I/O指令：启动通道程序

通道命令：对I/O操作进行控制

读、反读、写、测试设备状态的数据传输命令、用于设备控制的命令（磁带反绕、换页）、实现通道程序内部控制的转移命令

3）通道命令格式

命令格式一般包括：

操作码、数据传输内存地址、特征位、计数器

4)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。

4)工作原理

- 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。

  当通道传输完成最后一条指令时，向**CPU**发**I/O**中断，并且通道停止工作。**CPU**接收中断信号，从**CSW**中取得有关信息，决定下一步做什么

4.1.4　I/O通道

5.　通道与CPU的关系

- 主从关系
- 可并行工作
- 有通信方式
- 作用不同（通道——Ｉ/Ｏ；
  ＣＰＵ——计算）

4.1.4 I/O通道

- 通道传送与中断传送的区别：
  - 中断控制传送由中断控制器发出中断信息，中止CPU现行程序，转去执行中断服务程序。通道方式则是通过执行通道程序来实现。
  - 中断服务程序与CPU的现行程序是串行工作的，而通道程序的执行与CPU的现行程序是并行工作的。
  - 程序中断控制传送以CPU为中心，而通道则和DMA一样以内存为中心。

4.1.4 I/O通道

- 通道传送与DMA传送的区别：
  - DMA主要靠专用接口硬件实现数据传送；通道则靠执行通道程序实现数据传送。中断服务程序与CPU的现行程序是串行工作的，而通道程序的执行与CPU的现行程序是并行工作的。
  - DMA一般用来控制高速外设成组传送，通道既可控制高速外设成组传送，也可控制低速外设进行字或字节交叉传送。

4.1.5 I/O设备

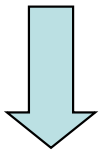
- 按数据组织分类：
  - 块设备
  - 字符设备
- 从资源分配角度分类：
  - 独占设备
  - 共享设备
  - 虚拟设备
- 按传输速率分类：
  - 低速设备
  - 中速设备
  - 高速设备
- 按其他方法分类：如按输入/输出对象进行，或者按是否可交互来进行 。

4.1.6 I/O控制方式

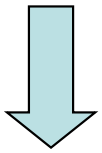
输入输出控制方式的发展过程

CPU直接控制外部设备

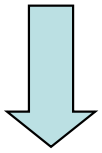
引入I/O部件，CPU直接控制I/O部件



引入中断驱动方式



引入DMA



I/O通道或I/O处理机

4.1.6 I/O控制方式

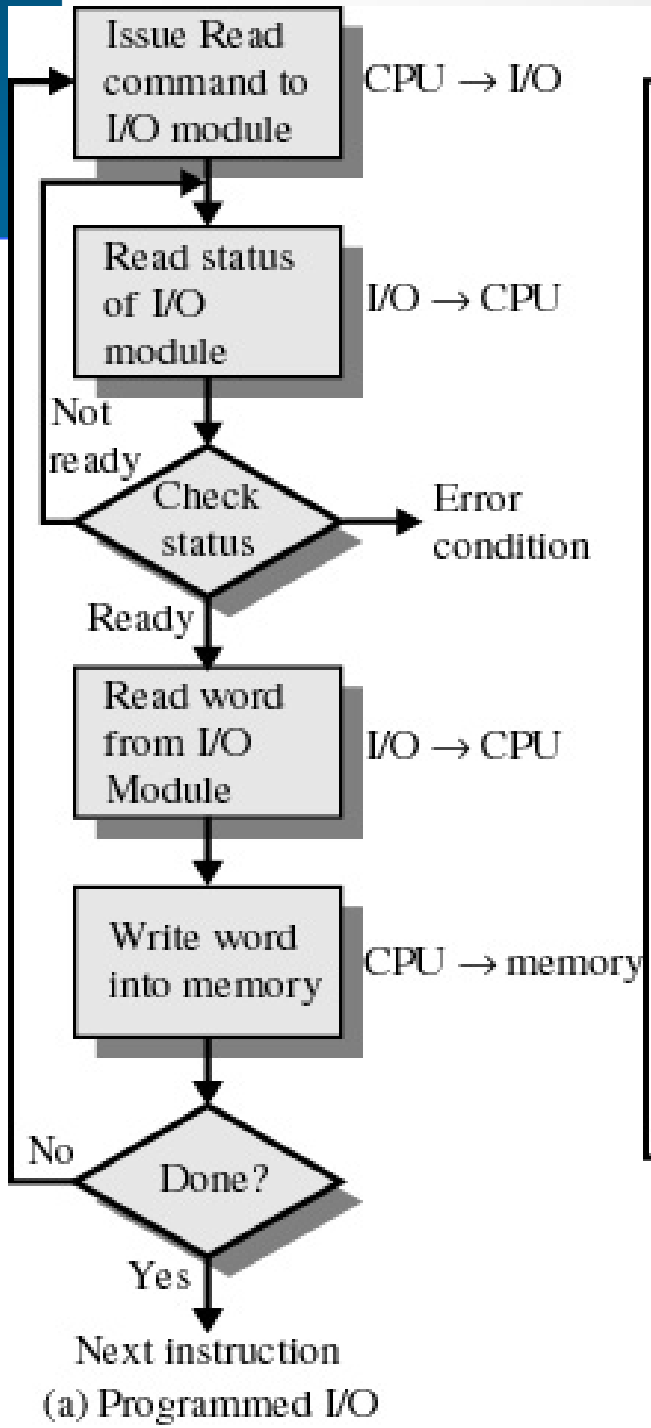
- 程序直接控制方式：
  - 需要CPU直接控制I/O操作的全过程，包括发送读写命令、传输数据、测试设备状态 。
  - 处理机指令集应包括控制类、测试类、读写类I/O指令 。
  - I/O部件接收到相应的指令后，将I/O状态写在寄存器的相应位置上。随着操作的执行更改状态位，由CPU执行相应指令读取I/O完成状态。I/O数据通过CPU寄存器转发。

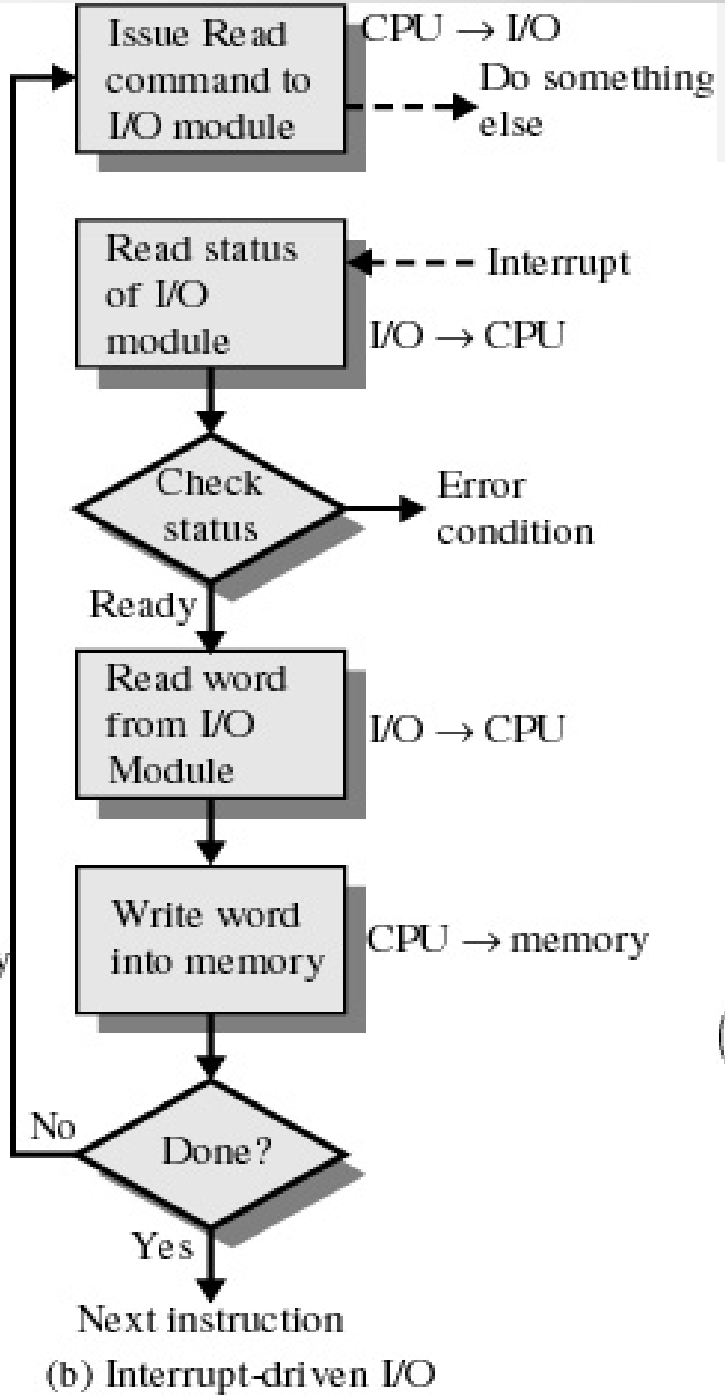
4.1.6 I/O控制方式

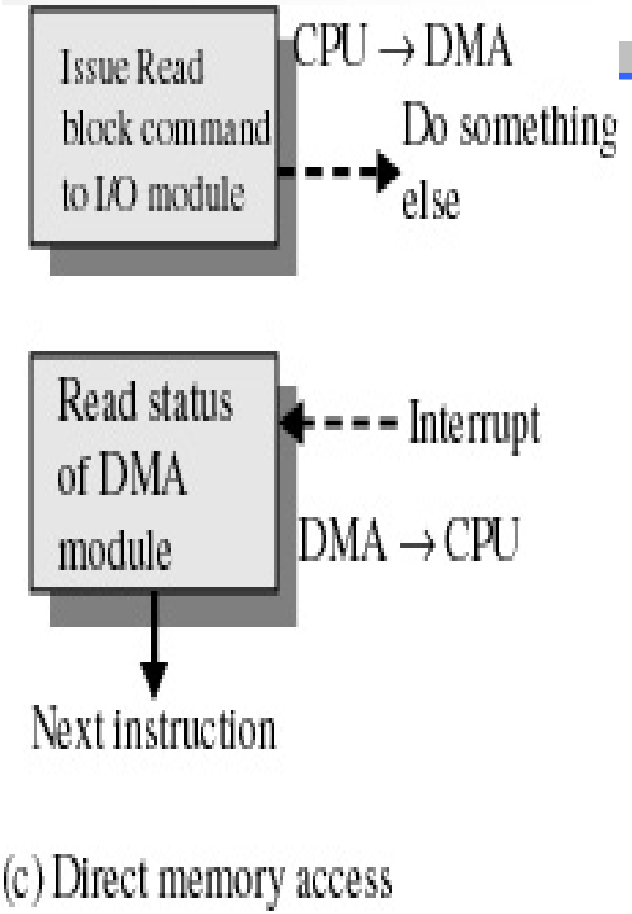
- 针对程序直接方式的不足，提出了中断方式：
  - CPU向I/O部件发出指令后，转去做其他有用的工作。当I/O部件准备好数据后，利用中断通知CPU，再由CPU完成数据传输。
  - 优点：
    - CPU不必反复测试寄存器状态，节约了时间。
  - 缺点：
    - 中断控制方式仍然消耗大量的CPU时间，因为每个字的数据传输都必须经过CPU寄存器转发。

4.1.6 I/O控制方式

- 程序直接控制方式与中断方式的缺陷：
  - I/O的传输速率受CPU测试或中断响应的速度限制
  - CPU为管理I/O耗费大量时间
- 更有效的方式—DMA（直接内存存取）：
  - 负责完成整个I/O操作，无需再经CPU寄存器转发，并在全部传输结束后向CPU发中断信号。
  - CPU向DMA部件发送I/O指令后，即可进行其他工作。给DMA的指令中应包括：操作类别、I/O设备的地址、读写数据在内存中的首地址、字数。







DMA

程序I/O 中断I/O

4.1.6 I/O控制方式

- 通道控制方式

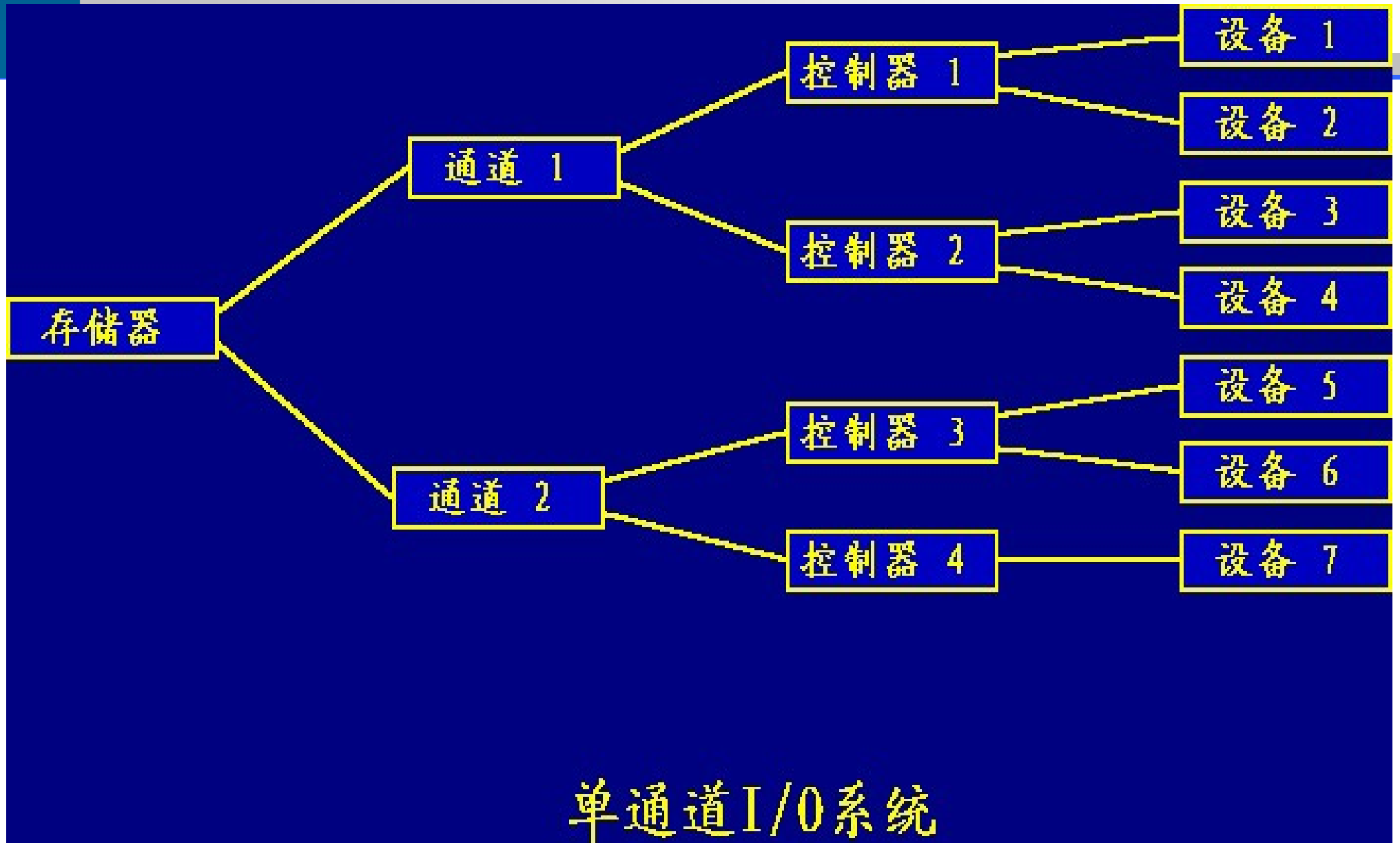
通道：执行通道程序，向控制器发出命令，并具有向CPU发中断信号的功能。 一旦CPU发出指令，启动通道，则通道独立于CPU工作。一个通道可连接多个控制器，一个控制器可连接多个设备，形成树形交叉连接

主要目的是启动外设时：

a 提高了控制器效率

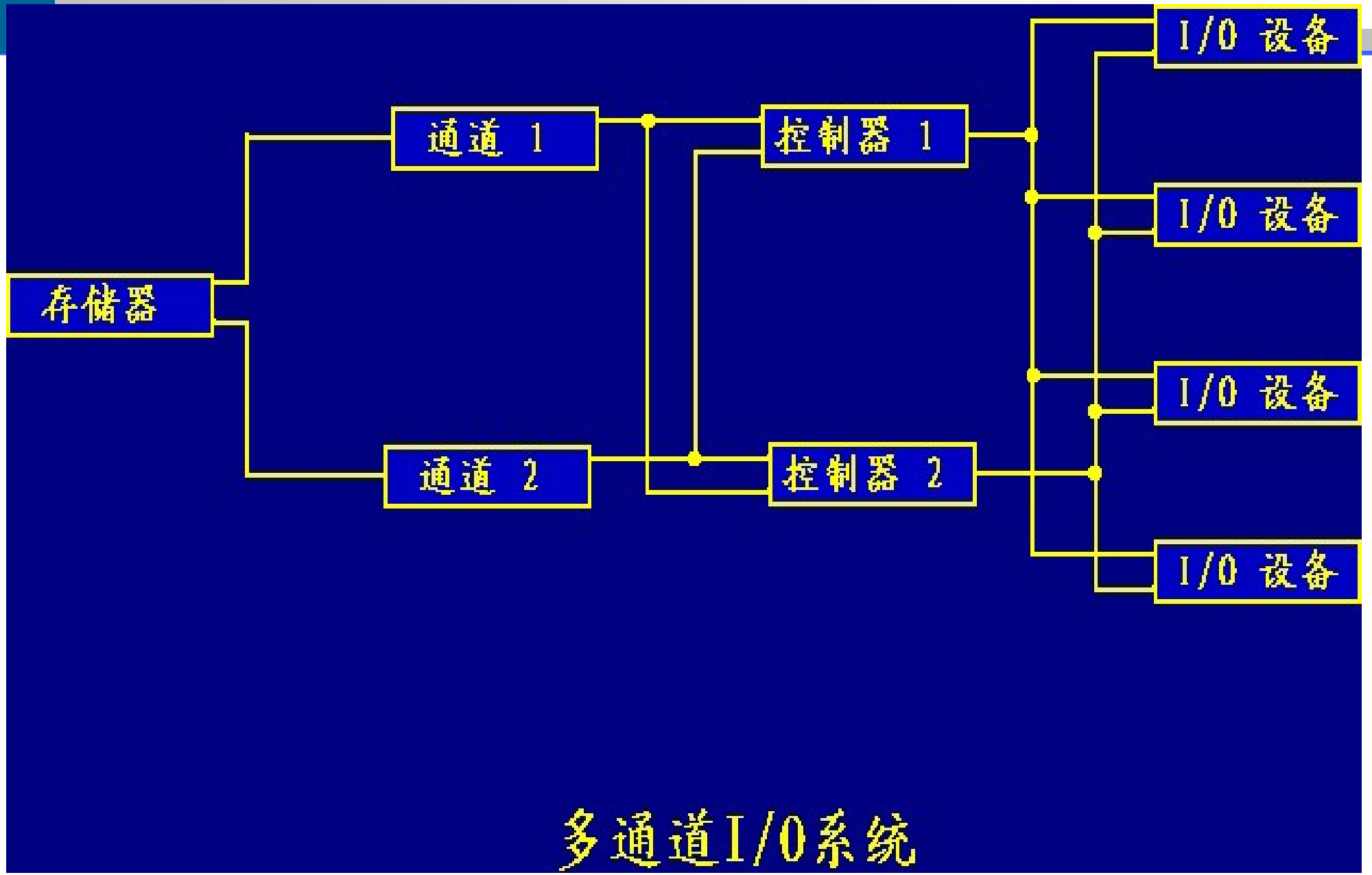
b 提高可靠性

c 提高并行度



单通道I/O系统

交叉连接



4.2 I/O软件

- **4.2.1 设备的使用与管理**
- **4.2.2 I/O软件层次结构**
- **4.2.3 缓冲管理**
- **4.2.4 设备驱动程序**
- **4.2.5 中断处理程序**

4.2.1 设备的使用与管理

- 设备管理的目标
  - 提高设备的利率
  - 为用户提供方便、统一的界面
- 设备管理的任务
  - 动态地掌握并记录设备的状态
  - 设备分配和释放
  - 缓冲区管理
  - 实现物理I/O设备的操作

4.2.1 设备的使用与管理

- 设备相关系统调用：
  - 申请设备：该系统调用中有参数说明了要申请的设备名称，操作系统处理该系统调用时，会按照设备特性（是独占还是分时共享式使用）及设备的占用情况来分配设备，返回申请是否成功标志。
  - 将数据写入设备。
  - 从设备读取数据。
  - 释放设备。这是申请设备的逆操作。

4.2.1 设备的使用与管理

- 设备管理中的数据结构
  - 在多通路的Ｉ／Ｏ系统中，为了满足一个Ｉ／Ｏ请求，不仅仅是分配一个Ｉ／Ｏ设备的问题，还应分配相应的控制器和通道，以确保ＣＰＵ与Ｉ／Ｏ设备之间能进行通信，在存储器与Ｉ／Ｏ设备之间能进行数据的直接存取。
  - 设备管理程序对Ｉ／Ｏ设备进行分配和控制是借助于一些表格；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。

数据结构（续）

- 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。如下表：

设备控制表（ＤＣＴ）	每个设备一个
控制器表（ＣＯＣＴ）	每个控制器一个
通道表（ＣＨＣＴ）	每个通道一个
系统设备表（ＳＤＴ）	整个系统一个

数据结构（续）

1） 系统设备表SDT

整个系统一张表，记录系统中所有I/O设备的信息，表目包括：

设备类型、设备标识符、进程标识符、DCT表指针等。

数据结构（续）

2） 设备控制表DCT

主要内容：设备类型、设备标识符、设备状态、与此设备相连的COCT、重复执行的次数或时间、等待队列的队首和队尾指针、I/O程序地址

3)控制器控制表COCT

4)通道控制表CHCT

COCT和CHCT与DCT类似

三种控制块

设备控制块（DCB）	通道控制块（CHCB）	控制器控制块（COCB）
设备标示符	通道标示符	控制器标示符
设备状态	通道状态	控制器状态
与设备相连的控制器表	与通道相连的控制器表	与控制器相连的通道表
等待此设备的进程表	等待此通道的进程表	等待控制器的进程表

4.2.1 设备的使用与管理

- **I/O设备的使用方式：**
  - 独占式共享使用设备
  - 分时式共享使用设备
  - 以**SPOOLing**方式使用外设

4.2.1 设备的使用与管理

- 独占式共享使用设备：
  - 是指在申请设备时，如果设备空闲，就将其独占，不再允许其他进程申请使用，一直等到该设备被释放，才允许被其他进程申请使用。
  - 独占式使用设备时，设备利用率很低。
  - 如果一个逻辑上完整的数据可以用设备的一次I/O操作完成，那么我们就不必要独占该设备。反过来说，如果一次I/O操作的数据逻辑上完整，我们就不必要对该设备进行独占方式的申请使用。在申请这种设备时，不必检查是否已被占用，只要简单累加设备使用者计数即可。

4.2.1 设备的使用与管理

- 分时式共享使用设备：
  - 以一次I/O为单位分时使用设备，不同进程的I/O操作请求以排队方式分时地占用设备进行I/O 。
  - 从用户程序系统调用界面来看，I/O操作是并发的。

4.2.1 设备的使用与管理

- **以SPOOLing方式使用外设：**
  - SPOOLing 技术是在批处理操作系统时代引入的，即所谓假脱机输入输出技术。原来这种技术输入、输出是针对磁盘上的输入、输出，现在是针对磁盘上的文件。
  - 例如:同一进程所有输出数据在进程运行时被写到同一文件当中，文件排到打印输出队列，打印进程申请占用打印机后，成批读出文件中数据，并送打印机打印出去。

4.2.1 设备的使用与管理

- **什么是Spooling?**

在联机情况下实现的同时外围操作称为Spooling，或称为假脱机操作。

- **Spooling系统的组成：**

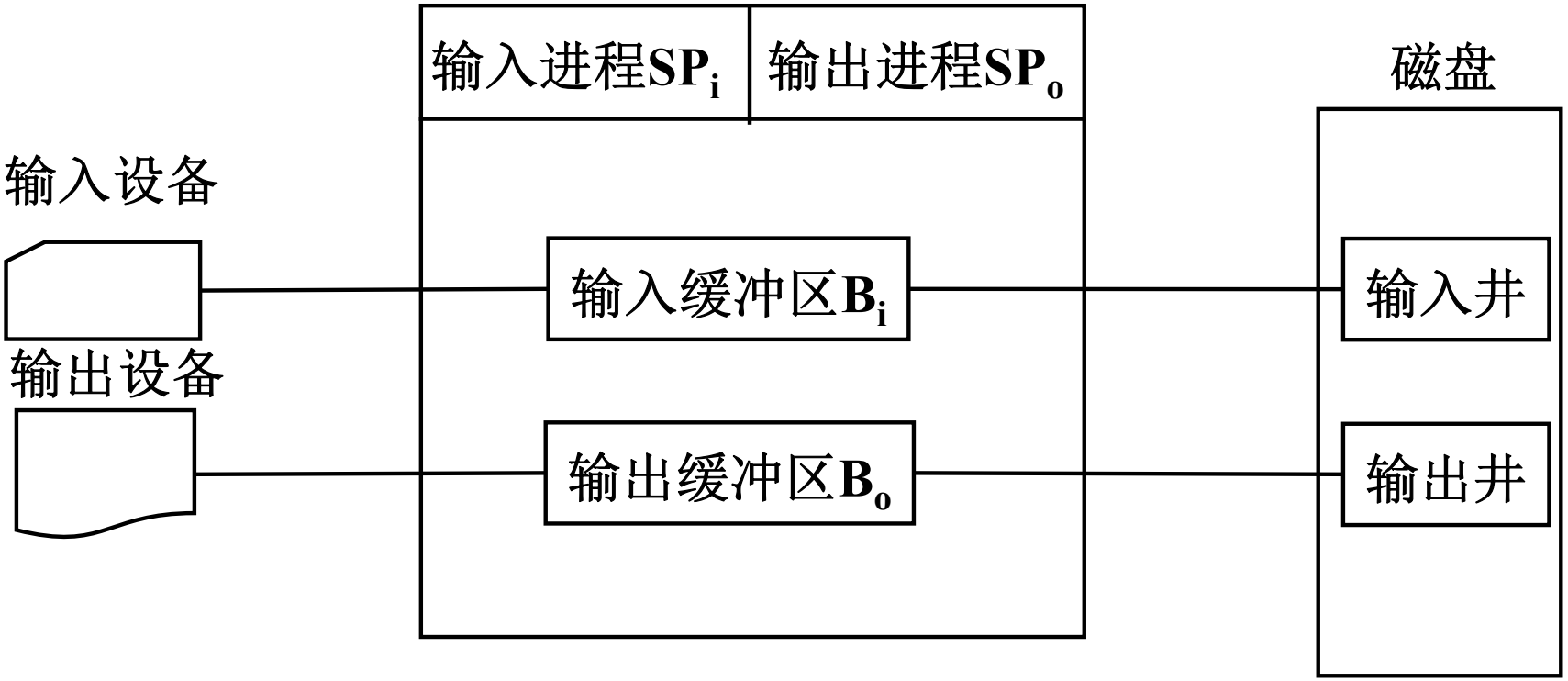
1、输入井和输出井，磁盘上开辟的两个大存储空间；

2、输入缓冲区和输出缓冲区；

3、输入进程SP_i和输出进程SP_o；

4、请求打印队列。

4.2.1 设备的使用与管理



Spooling系统的组成

4.2.1 设备的使用与管理

- Spooling系统的特点：

  1、提高了I/O速度；

  2、设备并没有分配给任何进程.在输入井或输出井中,分配给进程的是一存储区和建立一张I/O请求表；

  3、实现了虚拟设备功能。

4.2.1 设备的使用与管理

- 共享打印机：
  - 系统对于用户的打印输出，但并不真正把打印机分配给该用户进程，而是先在输出井中申请一个空闲盘块区，并将要打印的数据送入其中；然后为用户申请并填写请求打印表，将该表挂到请求打印队列上。若打印机空闲，输出程序从请求打印队首取表，将要打印的数据从输出井传送到内存缓冲区，再进行打印，直到打印队列为空。

4.2.1 设备的使用与管理

以SPOOLing方式使用外设

4.2.2 I/O软件层次结构

- 通常，操作系统将设备管理系统划分并组织成三个层次：
  - 用户层I/O
  - 与设备无关的I/O
  - 设备驱动及中断处理

用户层I/O
系统调用接口，设备无关的操作系统软件
设备驱动及中断处理
硬件

4.2.2 I/O软件层次结构

- 用户层**I/O**：
  - 用户层与设备的控制细节无关，不直接与设备打交道。
  - 它将所有的设备看作逻辑资源，为用户进程提供各类**I/O**函数。用户以设备标识符和一些简单的函数来使用设备，如打开、关闭、读、写等。如**C**库中的函数**fopen(); fread(); fwrite(); fclose();printf()**等 。

4.2.2 I/O软件层次结构

- 与设备无关的I/O软件层：

  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，这就是所谓与设备无关性。

  为此，我们将逻辑设备与物理设备区分，并引入逻辑设备名称和物理设备名称的概念。为了实现与设备的无关性，系统中必须有一张联系逻辑设备名称和物理设备名称的映射表（LUT表）。

4.2.2 I/O软件层次结构

逻辑设备号	物理设备号	驱动程序地址
1	7	20420
2	7	20420
3	2	20E00
4	4	1FC10
6	1	20D02
7	7	20420
15	10	1FC10
16	11	1FC120
…	…	……

4.2.2 I/O软件层次结构

这个ＬＵＴ中为三个不同的逻辑设备号列出了同样的物理设备和驱动程序的地址。这说明逻辑设备１，２，７目前均得到同一个物理设备７的服务（这或许是因为激光打印机及字母型打印机正在被修理，因此本来输出到这些设备上的请求都移到行式打印机）。在这个映射表中我们还可以看到物理设备４和１０都是由同一个驱动程序服务的。这是假定它们是同一类型的终端。

4.2.2 I/O软件层次结构

- 与设备无关的I/O软件层：
  - 它对上层提供系统调用的接口，对下通过设备驱动程序接口调用设备驱动程序。

4.2.2 I/O软件层次结构

- 该层软件的基本功能包括：
  - （1）设备名与设备驱动程序的映射
  - （2）设备保护
  - （3）逻辑块
  - （4）缓冲
  - （5）设备空闲空间管理与分配。
  - （6）错误报告
  - （7）分配及设备释放

4.2.2 I/O软件层次结构

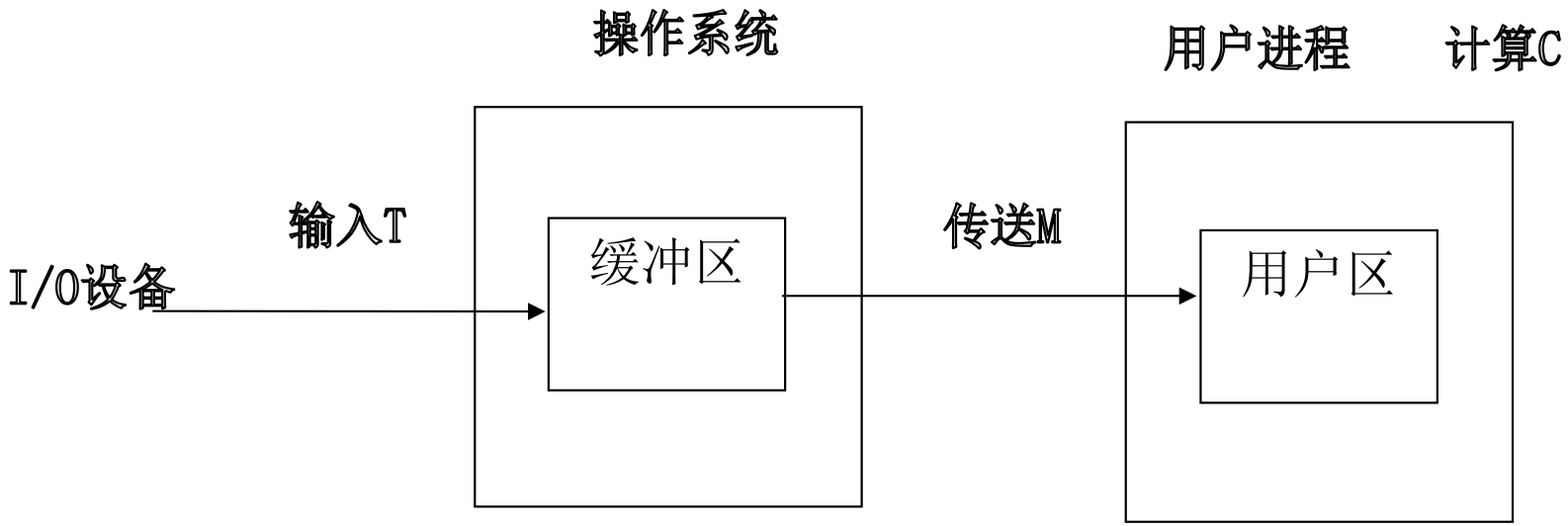
- 设备驱动与中断处理：
  - 设备驱动程序包括了所有与设备相关的代码，其功能是将设备无关层中的抽象请求转换成对底层设备的具体操作。
  - 当进程进行**I/O**操作时，将其阻塞至**I/O**操作结束并发生中断。中断发生时，由中断处理程序启动请求排队的下一请求并解除等**I/O**进程的阻塞状态，使其能够继续执行。

4.2.3 缓冲管理

- 引入缓冲的主要原因有以下几个方面：
  - 缓解**CPU**与**I/O**设备间速度不匹配的矛盾。
  - 减少对**CPU**的中断频率，放宽对中断处理时间的限制。
  - 提高**CPU**和**I/O**设备之间的并行性。

4.2.3 缓冲管理

- 单缓冲（**Single Buffer**）：
  - 是操作系统缓冲支持中最为简单的一种形式，每当用户进程发出一个**I/O**请求时，操作系统便在内核区域为之分配一个缓冲区。

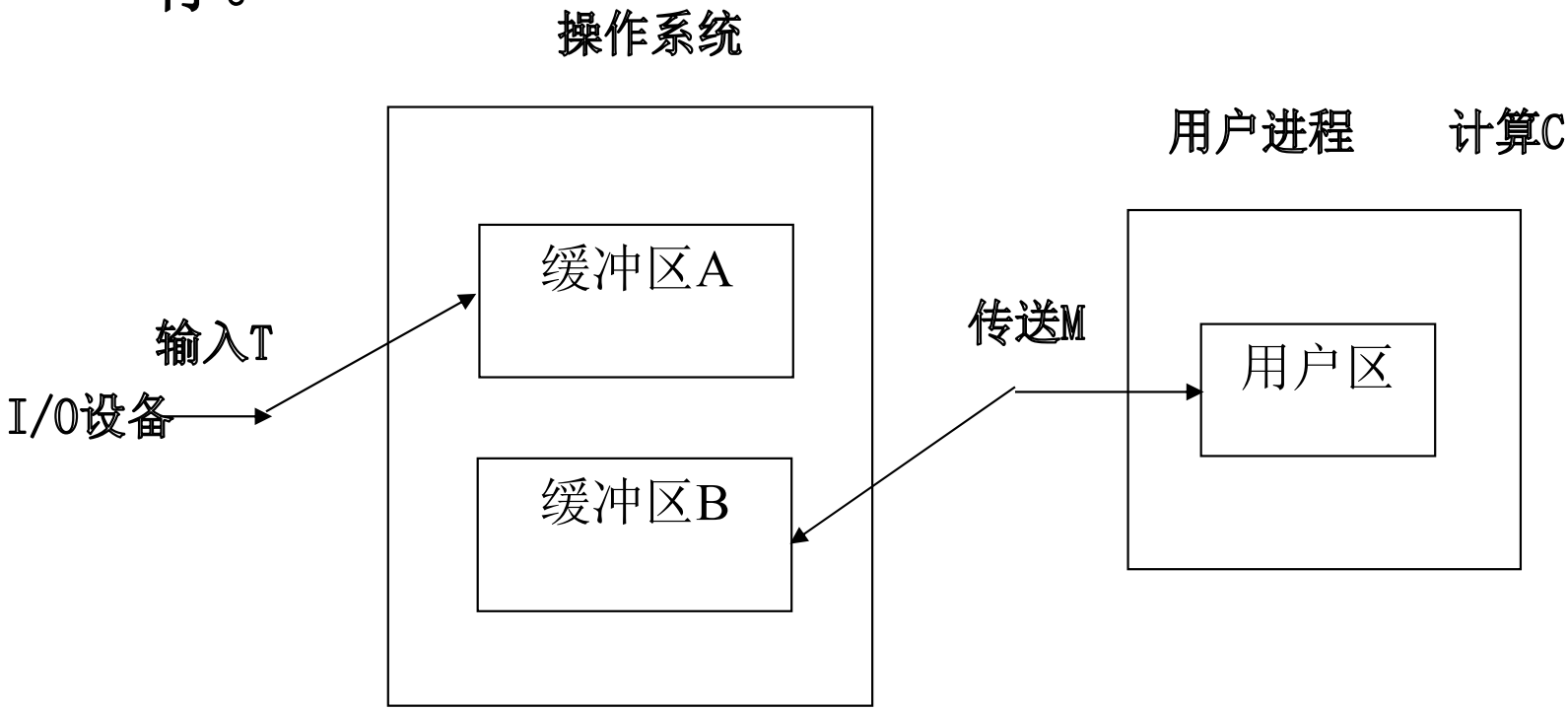


4.2.3 缓冲管理

- 例如，CPU从磁盘上读入N个数据块，然后对这些数据块进行计算，采用单缓冲机制的工作过程为：先从磁盘把一块数据读入到缓冲区中，所花时间为T；在读取一块以后由操作系统将缓冲区的数据传送到用户区，所花时间为M；最后由CPU对这一块数据进行计算，所花费的时间为C。如果不采用缓冲，将数据直接从磁盘读入用户区，每批数据的处理时间为（T+C）×N。采用单缓冲技术时，数据读入缓冲与计算C是可以并发执行的（第一和最后一块除外，采用前面讲到的预先读取方式），这样，每批数据的最大处理时间为MAX（C，T）×N+M×N。通常M远小于T或C，这样就提高了CPU和外设的利用率。

4.2.3 缓冲管理

- 双缓冲（**Double Buffer**）：
  - 是对单缓冲方式的改进。可以实现用户数据区—缓冲区之间交换数据和缓冲区—外设之间交换数据并行。

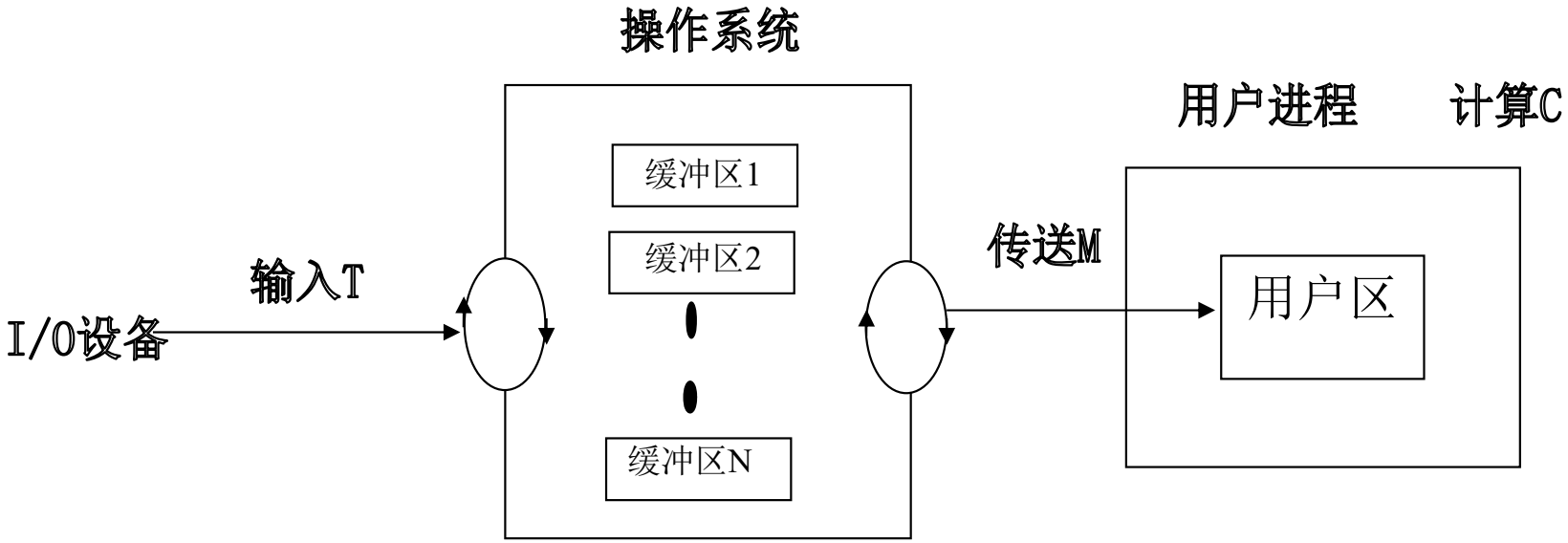


4.2.3 缓冲管理

- 双缓冲工作方式的基本方法是：在设备进行输入时，先将数据输入到缓冲区**A**，装满后转向缓冲区**B**。此时操作系统可以从缓冲区**A**中提取数据传送到用户区，最后由**CPU**对数据进行计算或其他处理。
- 系统处理一块数据的处理时间可粗略地认为是**MAX（C，T）**。若**C>T**，可使块设备连续输入；若**C<T**，可使**CPU**不必等待设备输入。

4.2.3 缓冲管理

- 循环缓冲（**Circular Buffer**）：
  - 指采用有限缓冲区的生产者/消费者模型对缓冲池中的缓冲区进行循环使用 。
  - 缓冲区结合预读和滞后写技术对具有重复性及阵发性**I/O**进程提高**I/O**速度很有帮助。



4.2.4 设备驱动程序

- 设备驱动程序的功能：
  - 向有关I/O设备的各种控制器发出控制命令，并且监督它们的正确执行，进行必要的错误处理。
  - 对各种设备排队、挂起、唤醒等操作进行处理。
  - 执行确定的缓冲区策略。
  - 进行比寄存器接口级别层次更高的一些特殊处理，如代码转换，ESC处理等。这些特殊处理均依赖于具体设备，不适合放在高层次的软件中处理。

4.2.4 设备驱动程序

- 设备驱动程序的特性
  - 设备驱动程序的突出特点是，它与**I/O**设备的硬件结构密切相关，是操作系统底层中惟一知道各种**I/O**设备的控制器细节及其用途的部分。

4.2.4 设备驱动程序

- 设备驱动程序的结构
  - 设备驱动程序的结构同**I/O**设备的硬件特性有关。一台彩色显示器的设备驱动程序的结构，显然同磁盘设备驱动程序的结构不同。通常，一个设备驱动程序对应处理一种设备类型，或者至多是一类密切相关、而差异性较少的设备类型。

4.2.4 设备驱动程序

- 设备驱动程序的实现策略
  - （1）确定是否发送新的请求。
  - （2）针对具体设备和操作，确定发送的内容。
  - （3）执行底层具体操作。
  - （4）中断与后续处理工作。在一条或多条指令发出以后，存在着两种做法。

4.2.4 设备驱动程序

- 设备驱动程序接口函数主要包括：
  - 驱动程序初始化函数：是为了使驱动程序其它函数能被上层正常调用，而做一些针对驱动程序本身的初始化工作。如向操作系统登记该驱动程序的接口函数，该初始化函数在系统启动时或驱动程序安装入内核时执行。
  - 驱动程序卸载函数：是驱动程序初始化函数的逆过程，在支持驱动程序可动态加载卸载的系统中才需要。
  - 申请设备函数：该函数申请一个驱动程序所管理的设备，按照设备特性进行独占式占用或者分时共享式占用，如果是独占式申请成功还应该对设备做初始化工作。
  - 释放设备函数：是申请设备函数的逆过程。

4.2.5 中断处理程序

- 中断基本概念：
  - 指**CPU**暂时终止现行程序，转去执行其他紧急事件或特殊请求，处理完后自动返回原来被中断处继续执行或调度新的进程执行的过程。
  - 一旦**CPU**响应中断，转入中断处理程序，系统就开始进行中断处理。

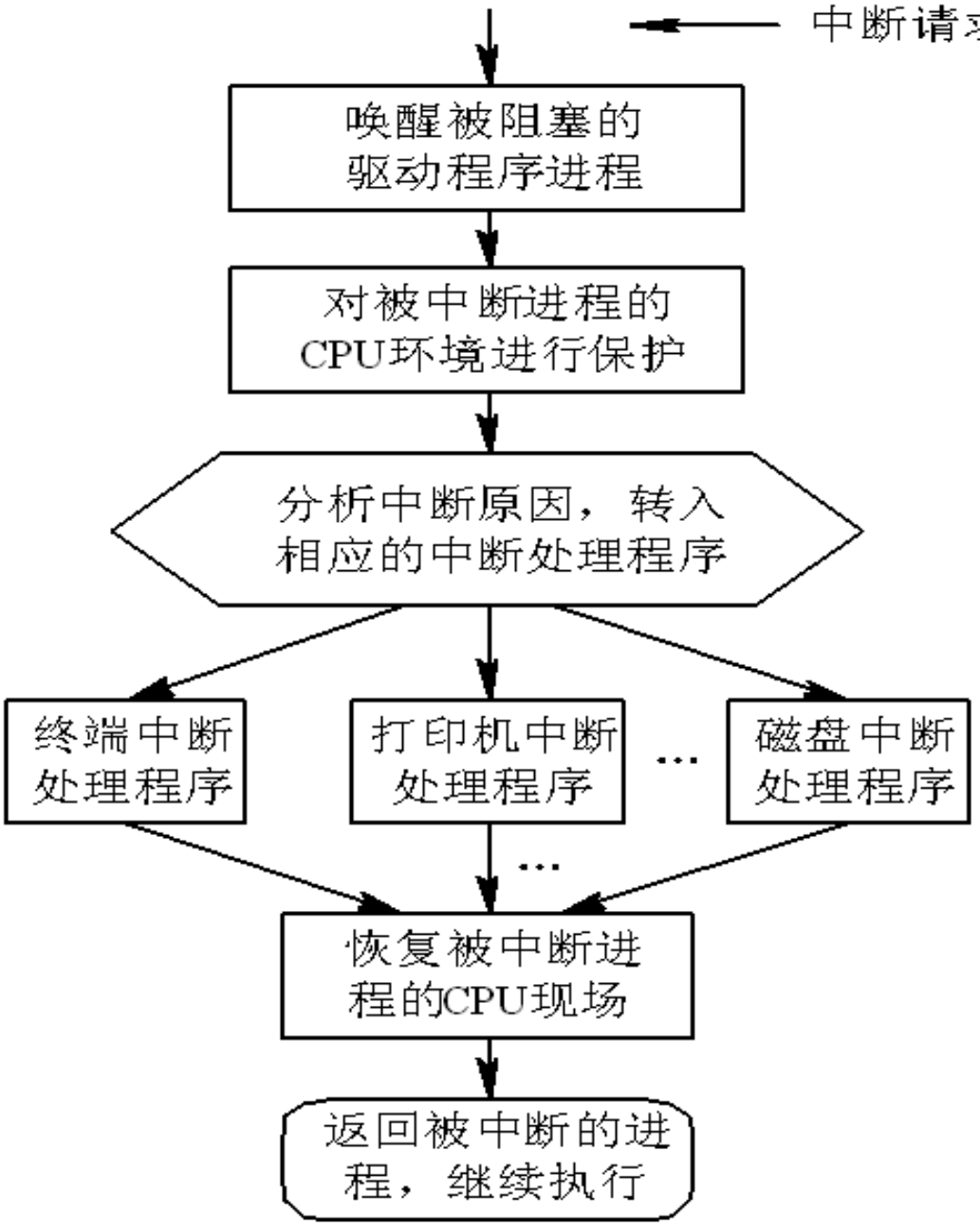
4.2.5 中断处理程序

- 中断处理过程：
  - （1）CPU检查响应中断的条件是否满足。CPU响应中断的条件是，有来自于中断源的中断请求及CPU允许中断。如果中断响应条件不满足，则中断处理无法进行。
  - （2）如果CPU响应中断，则CPU关中断，使其进入不可再次响应中断的状 态。
  - （3）保存被中断进程的现场。为了在中断处理结束后能使进程正确地返回到被中断点，系统必须保存当前处理机状态字PSW和程序计数器PC等的信息。

    这些信息通常保存在特定堆栈或寄存器中。

4.2.5 中断处理程序

- （4）分析中断原因，调用中断处理子程序。作为中断处理过程的核心工作之一，就是如何确定和寻找到中断服务程序的入口地址，该工作可由软件和硬件的方法来完成。
- （5）执行中断处理子程序。在有些系统中的异常是通过异常指令向当前执行进程发出软中断信号后,调用对应的处理子程序执行。
- （6）退出中断，恢复被中断进程的现场或调度新进程占据CPU。
- （7）开中断，CPU继续执行。


中断请求信号
唤醒被阻塞的
驱动程序进程
对被中断进程的
CPU环境进行保护
分析中断原因，转入
相应的中断处理程序
终端中断
处理程序
打印机中断
处理程序
...
磁盘中断
处理程序
...
恢复被中断进
程的CPU现场
返回被中断的进
程，继续执行

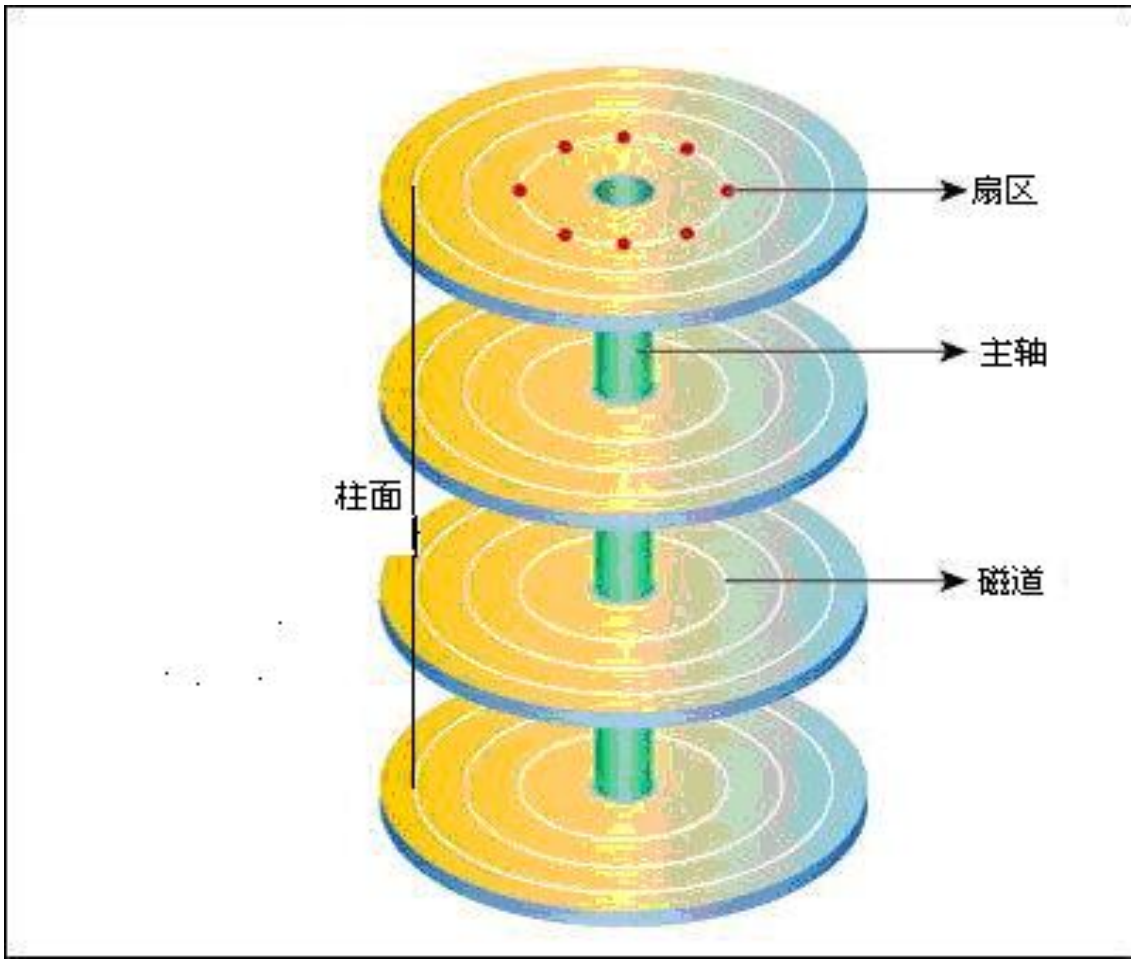
中断处理流程

4.3 存储设备

- **4.3.1** 常见的存储外设
- **4.3.2** 磁盘调度
- **4.3.3** 存储出错处理
- **4.3.4 RAM**盘
- **4.3.5** 磁盘阵列

4.3.1 常见的存储外设

- 磁带存储设备。
- 磁盘存储设备。
- 光盘存储设备。
  - **CD-ROM、**
  - **CD-R、**
  - **CD-RW、**
  - **DVD。**



磁盘示意图

4.3.2 磁盘调度

- 磁盘为共享设备，面临多个进程同时向磁盘提出磁盘访问操作的要求。但是，系统在任何时刻只允许一个对磁盘的I/O操作，其余操作只能等待。
- 磁盘访问：
  - 寻道时间：磁头花费在柱面定位上的时间。

    $T_s=m\times n+s$　　(m<0.1，s=0.2)
  - 旋转延迟时间：指定扇区移动到磁头下面所需的时间。(5.5ms)
  - 传输时间：数据写入磁盘或从磁盘读出的时间。

4.3.2 磁盘调度

- 磁盘调度：
  - 分为寻道调度和旋转调度两类，并且是先进行寻道调度，然后再进行旋转调度。
  - 就目前来讲，访问磁盘最耗时的还是寻道，因此，磁盘调度的大多数算法仍限定在追求平均寻道时间最少这个目标上。

4.3.2 磁盘调度

- 面向寻道的磁盘调度算法：
  - **FCFS（First Come , First Served，先来先服务）**
  - **SSTF（Shortest Seek Time First，最短寻道时间优先）**
  - **SCAN（扫描）调度，也称电梯调度**
  - **C-SCAN（Circular SCAN，循环扫描）调度算法**

4.3.2 磁盘调度

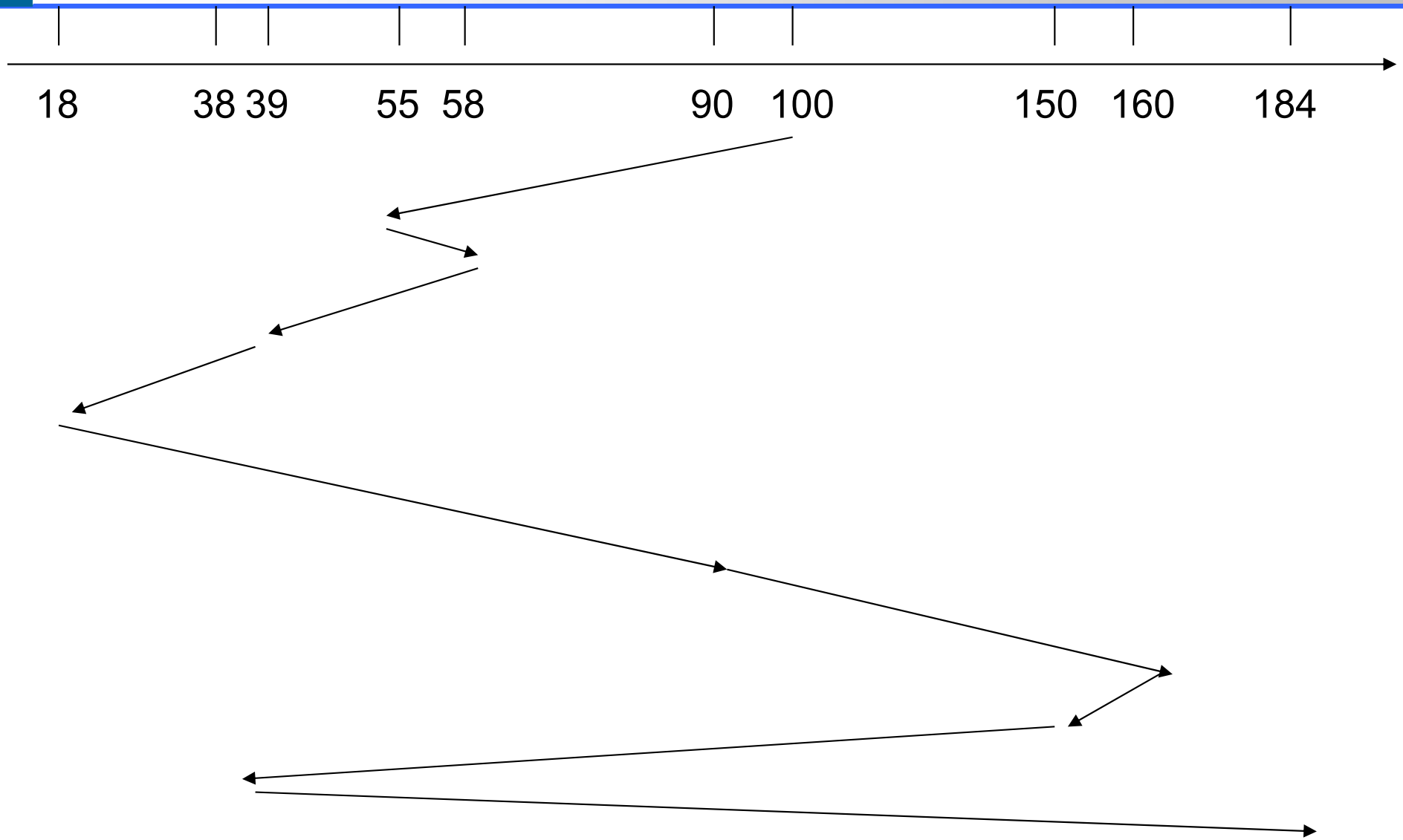
- （1）先来先服务：
  - 是所有磁盘调度算法中最简单的。根据所有进程访问磁盘请求的先后顺序进行调度。每个进程的请求都能依次得到处理，不会出现某进程的请求长期得不到满足的情况。
  - 此算法由于未对寻道进行优化，导致平均寻道时间较长。

4.3.2 磁盘调度

FCFS调度算法

（从 100 号磁道开始）	
被访问的下一个磁道号	移动距离（磁道数）
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度：55.3	

磁头访问次序 55 58 39 18 90 160 150 38 184，当前磁道100



4.3.2 磁盘调度

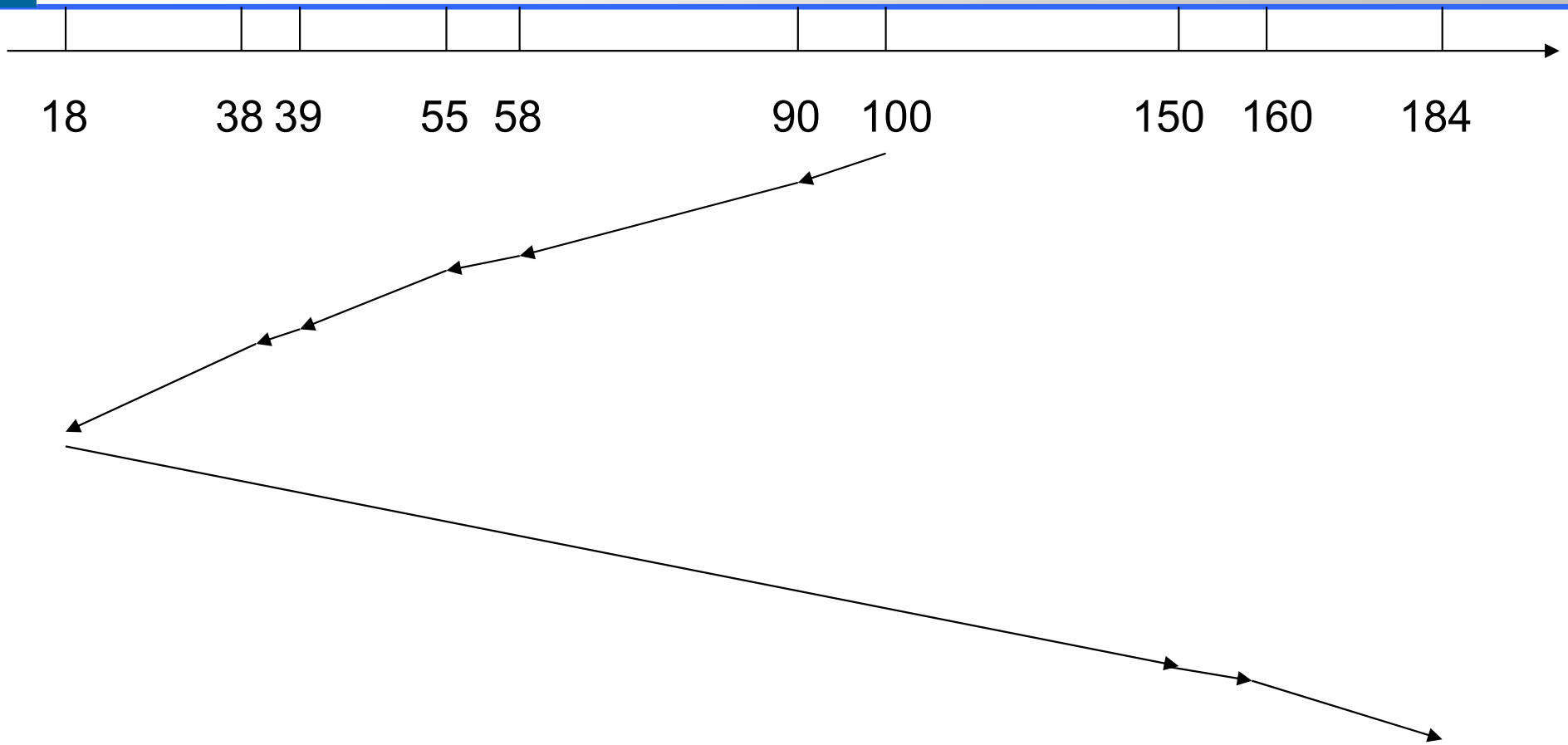
- （2）最短寻道时间优先：
  - 最短寻道时间优先算法选择磁道与当前磁头所在磁道距离最近的下一I/O操作进行，使每次的寻道时间最短，但却不能保证平均寻道时间最短。

4.3.2 磁盘调度

SSTF调度算法

（从 100 号磁道开始）	
被访问的下一个磁道号	移动距离（磁道数）
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
平均寻道长度：27.5	

磁头访问次序 55 58 39 18 90 160 150 38 184，当前磁道100



4.3.2 磁盘调度

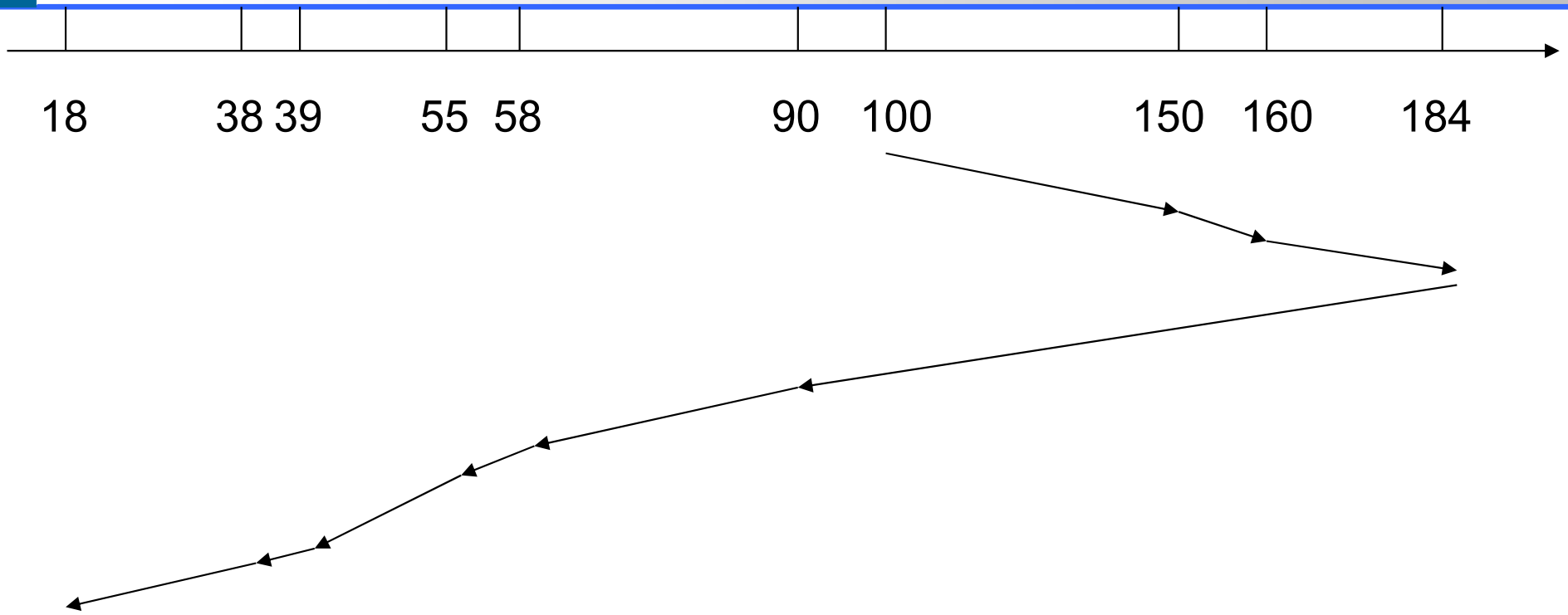
- （3）扫描算法（也称电梯算法）
  - 扫描算法不但要考虑欲访问的磁道与当前访问磁道的距离，而且更优先考虑磁头当前的移动方向。这样，相对**SSTF**算法而言，将不存在迟迟得不到响应的**I/O**操作。也就是说在当前移动方向上选择与当前磁道最近的进行先处理。
  - 这种算法中，有如高楼里上下运动的电梯，故又常称为电梯调度算法。

4.3.2 磁盘调度

SCAN调度算法示例

（从100#磁道开始，向磁道号增加方向访问）	
被访问的下一个磁道号	移动距离（磁道数）
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
平均寻道长度：27.8	

磁头访问次序 55 58 39 18 90 160 150 38 184，当前磁道100



4.3.2 磁盘调度

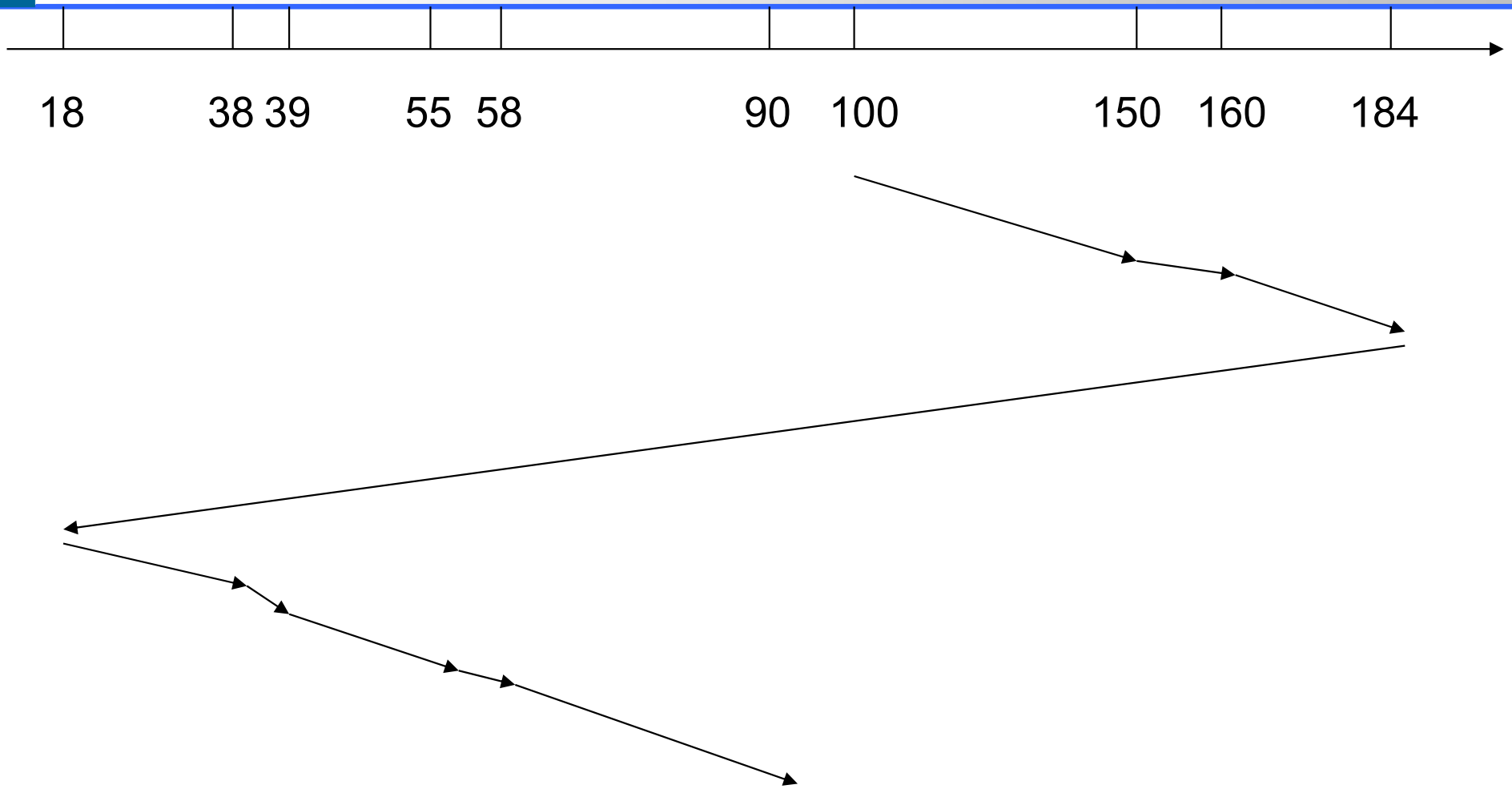
- （4）循环扫描算法
  - **SCAN**算法既能获得较好的寻道性能，又可以防止进程的“饥饿”现象，所以在大、中、小型机和网络中的硬盘调度中被广泛用。
  - 缺陷：当磁头刚从外向里移动过某一磁道时，正好又有一个进程请求访问此磁道，那么这时该进程必须等待，待磁头从外向里、然后再从里向外扫描完所有要访问的磁道后，才去处理该进程的请求，使得该进程的请求被严重地推迟。为了减少这种延迟，**CSCAN**算法规定磁头只做单向移动。 例如只作自里向外移动，当磁头访问完最外的被访问磁道时，立即返回到最里的欲访问磁道，最小磁道号紧接着最大磁道号构成循环扫描。

4.3.2 磁盘调度

CSCAN调度算法示例

（从100#磁道开始，向磁道号增加方向访问）	
被访问的下一个磁道号	移动距离（磁道数）
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度：27.5	

磁头访问次序 55 58 39 18 90 160 150 38 184，当前磁道100



4.3.2 磁盘调度

- **（5）FSCAN算法（First SCAN）**
  - 把**SCAN**经过简化，便形成了**FSCAN**算法。**FSCAN**算法只将磁盘请求访问队列分成两个子队列。一个是当前所有请求磁盘**I/O**的进程形成的队列，由磁盘调度按**SCAN**算法进行处理。另一个则是在扫描期间新出现的所有请求磁盘**I/O**进程的队列，它们将插入另一个等待处理的请求队列中。这样，所有的新请求都将被推迟到下一次扫描时处理。

4.3.3 存储出错处理

- 从磁盘驱动程序的角度来讲，需要关注和进行以下错误处理：
  - 程序性错误（例如，申请不存在的磁道）；
  - 瞬时校验错误（例如，磁头上有灰尘引起的读写错误）；
  - 永久性校验错误（例如，磁盘块物理介质损坏）；
  - 寻道错误（例如，寻找柱面**2**，磁臂却定位到柱面**4**）；
  - 控制器错误（例如，控制器拒绝执行命令）。

4.3.4 RAM盘

- 当物理内存较小时，我们可以将一部分磁盘空间虚拟成物理内存，使得内存看起来变得更大，方便程序地址空间管理、大程序的编写和多道程序的执行。随着内存容量的增加，能不能反过来，将一部分内存空间当作磁盘来使用呢？这就是**RAM**盘概念的由来。
- **RAM**盘的思想是就是将一块内存区域虚拟成磁盘块设备，同样支持写数据块和读数据块操作。

4.3.5 磁盘阵列

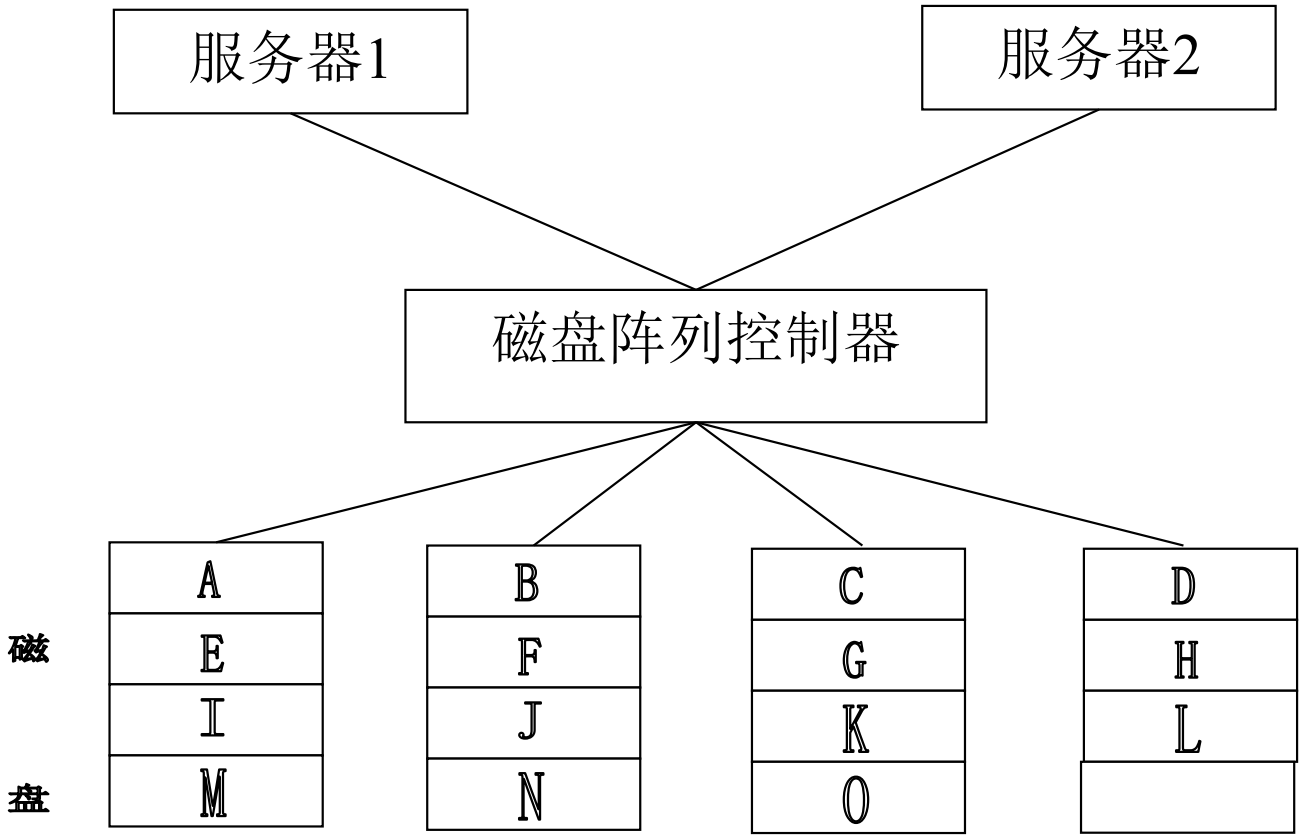
- 作用：
  - 通过冗余提高可靠性：如建立镜像盘。
  - 通过并行性提高性能：如将原来在一个物理盘连续的数据分条分布到多盘。

1987年，**Patterson**、**Gibson**和**Katz**这三位工程师在加州大学伯克利分校发表了题为《**A Case of Redundant Array of Inexpensive Disks**（廉价磁盘冗余阵列方案）》的论文，其基本思想就是将多个容量较小、相对廉价的硬盘驱动器进行有机组合，使其性能超过一个昂贵的大硬盘。这一设计思想很快被接受，从此**RAID**技术得到了广泛应用，数据存储进入了更快速、更安全、更廉价的新时代。

4.3.5 磁盘阵列

- **RAID 0**
  - 用块级条带化分割数据，并行地读/写于多个磁盘， 因此具有较高的数据传输率。
  - 没有任何的冗余，没有数据可靠性保障。因此**RAID 0**也称为无容错功能的条带磁盘阵列（**Striped Disk Array without Fault Tolerance**）。

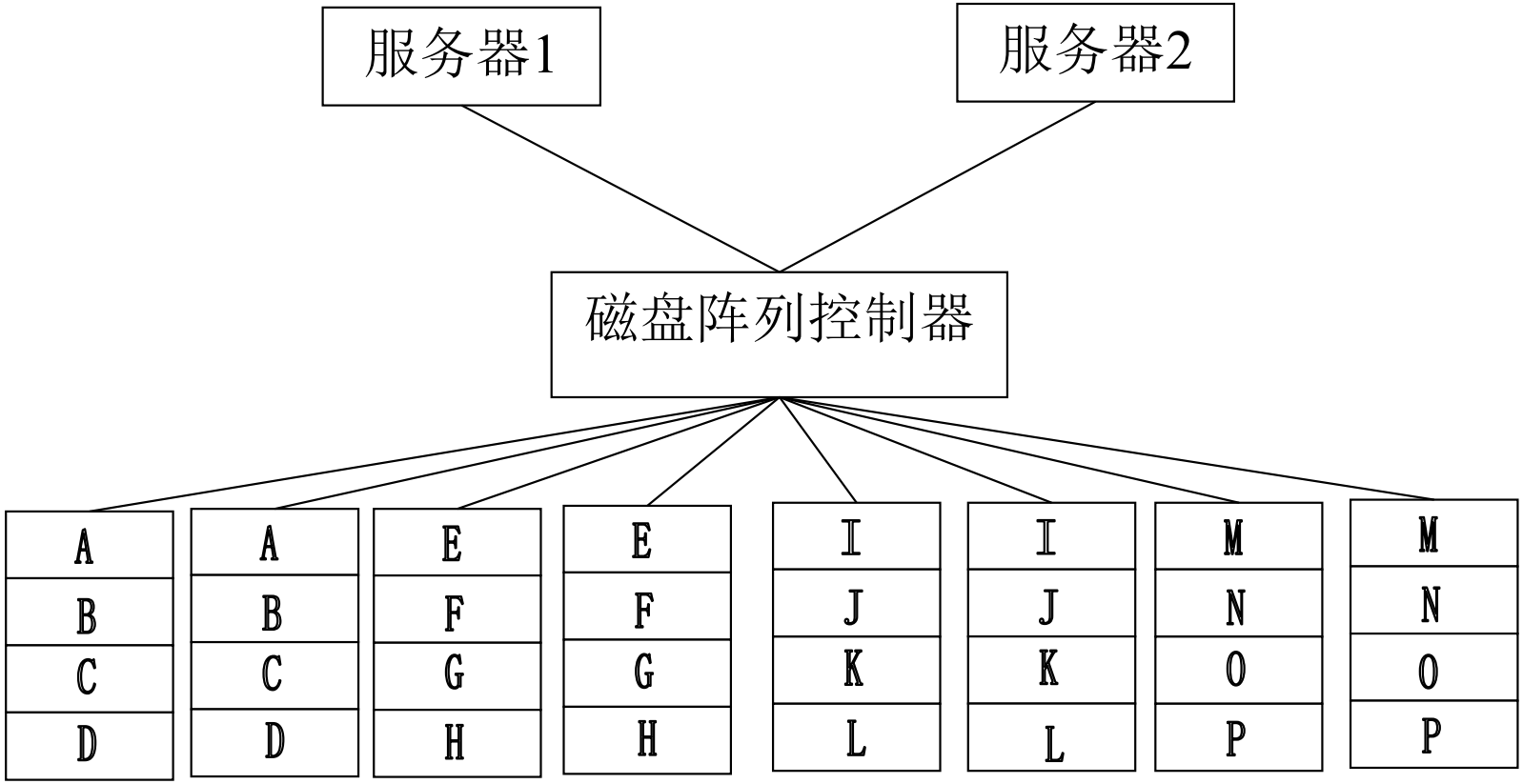
4.3.5 磁盘阵列



4.3.5 磁盘阵列

- **RAID 1**
  - 采用磁盘镜像技术，阵列中的单元为一对互为备份的磁盘。
  - 进行数据读操作时，可从镜像盘中的任一盘中进行读取，提高了读操作的性能。
  - 一半的磁盘用于备份，其成本较高。
  - 当一个磁盘失效，系统可以自动地交换到镜像磁盘上，而不需要重组失效的数据，可靠性好。

4.3.5 磁盘阵列

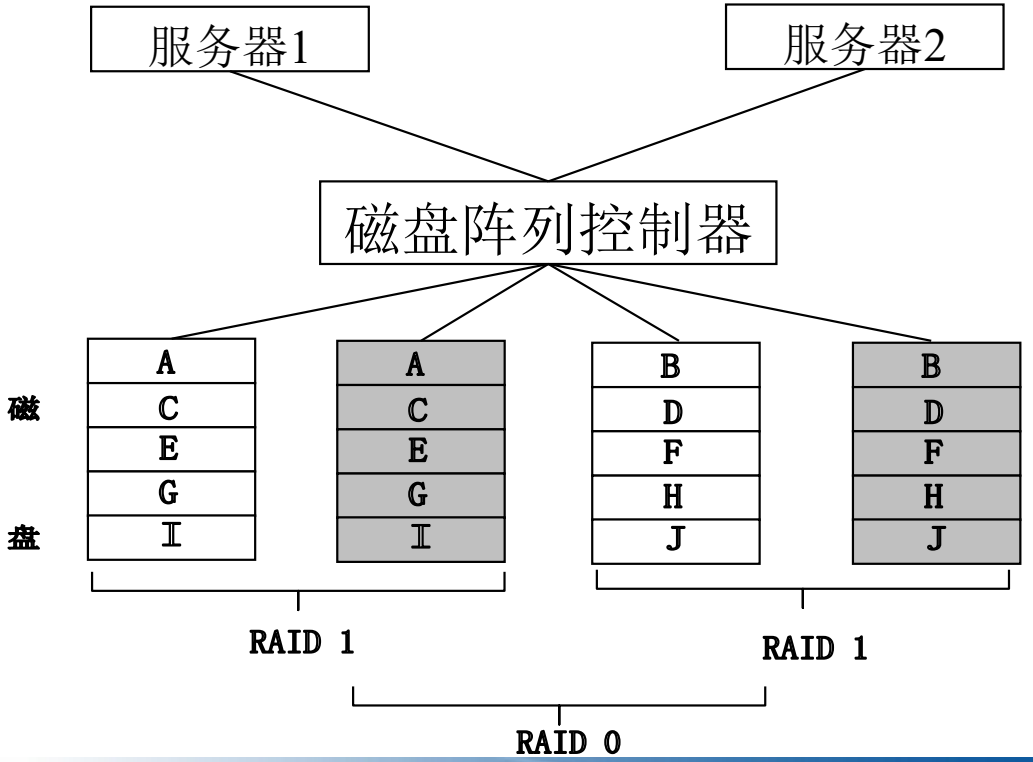


4.3.5 磁盘阵列

- **RAID 0+1:**
  - 是将**RAID 0**和**RAID 1**进行结合的产物，一组磁盘被条带化，然后条带被被镜像到另一个相等的条带。
  - 同时拥有**RAID 0**的超凡速度和**RAID 1**的数据高可靠性。
  - 磁盘利用率较低

4.3.5 磁盘阵列

- **RAID 1+0：**
  - 是将**RAID 1**和**RAID 0**进行结合的产物，其中磁盘被镜像成对，最后所得到的镜像对被条带化。

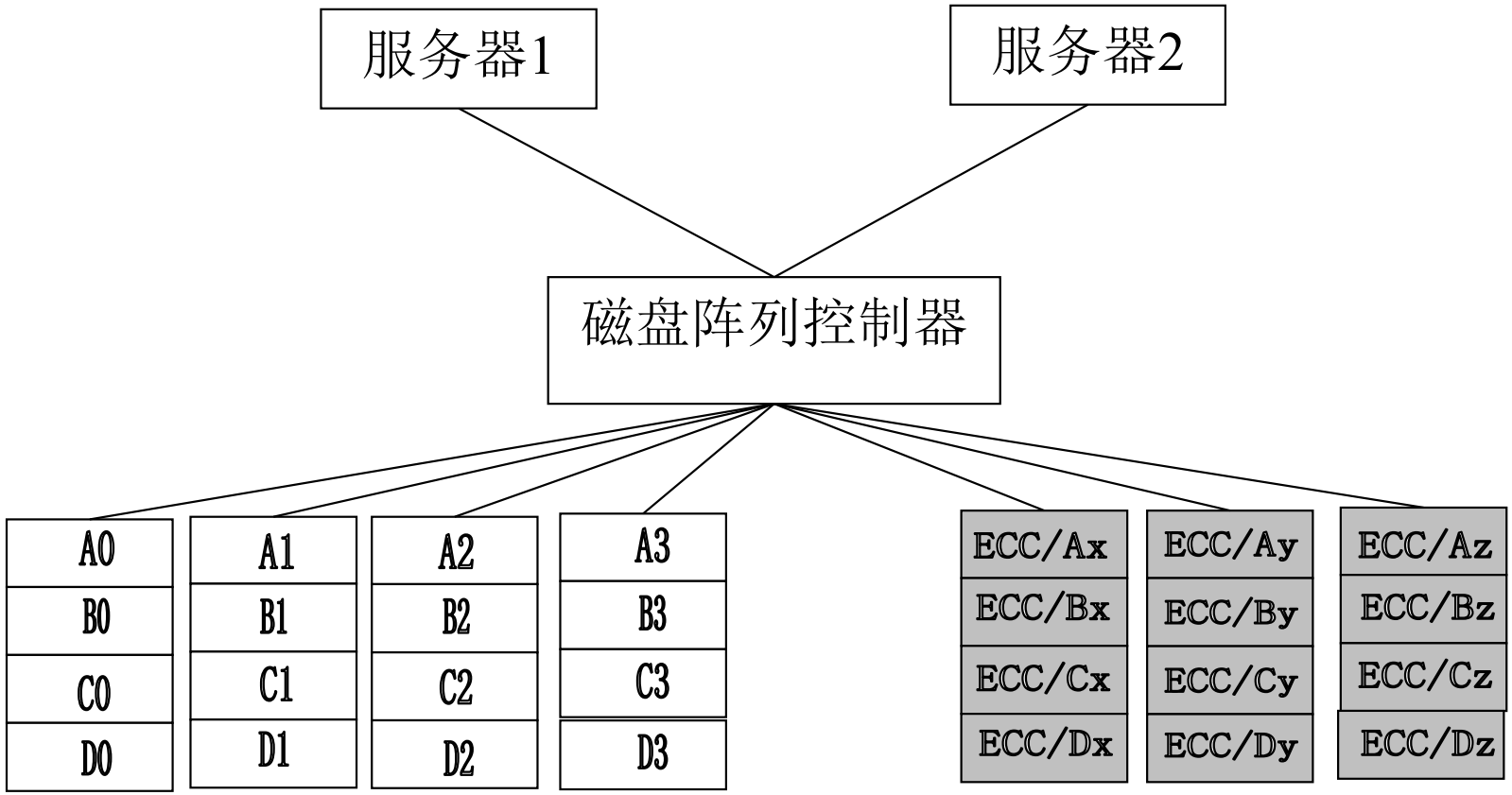


4.3.5 磁盘阵列

- **RAID 2**
  - 使用位条带化。对每个数据磁盘中的相应位都计算一个海明错误校正码，并且这个码位保存在多个奇偶校验磁盘中相应的位中，因此，**RAID级别2也称为海明码错误检测与修正方法（Hamming Code ECC，ECC（Error Correcting Code））。**

4.3.5 磁盘阵列

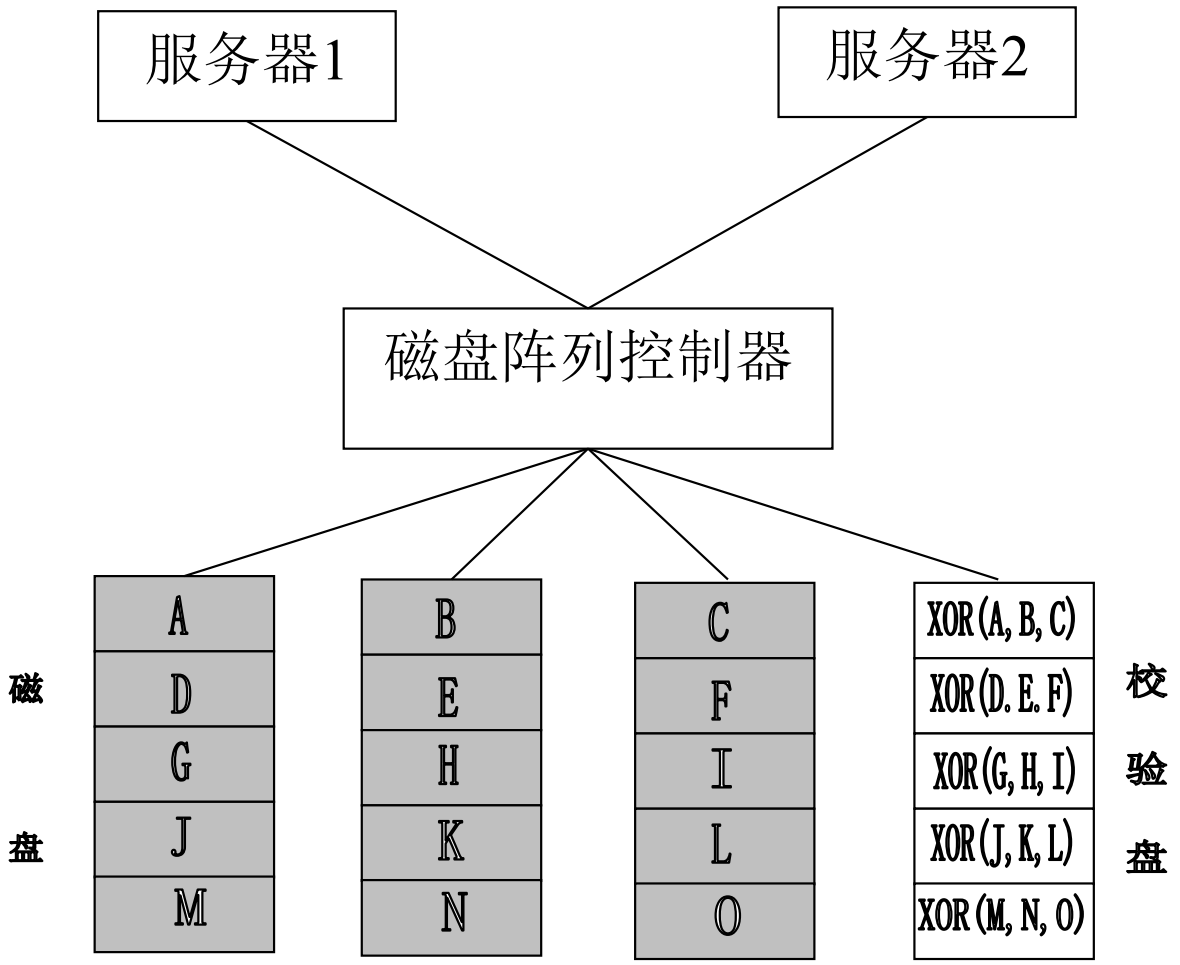
- 4位数据宽度，3个纠错盘时数据的存储示例 。



4.3.5 磁盘阵列

- **RAID 3**
  - 是**RAID 2**思想的发展，它采用相对简单的异或逻辑运算校验代替了复杂的海明码校验，条带宽度为字节。
  - **RAID3**只需一块额外的校验盘，成本得以降低。

4.3.5 磁盘阵列


服务器1
服务器2
磁盘阵列控制器
磁盘
A
D
G
J
M
B
E
H
K
N
C
F
I
L
O
XOR(A, B, C)
XOR(D. E. F)
XOR(G, H, I)
XOR(J, K, L)
XOR(M, N, O)
校验盘

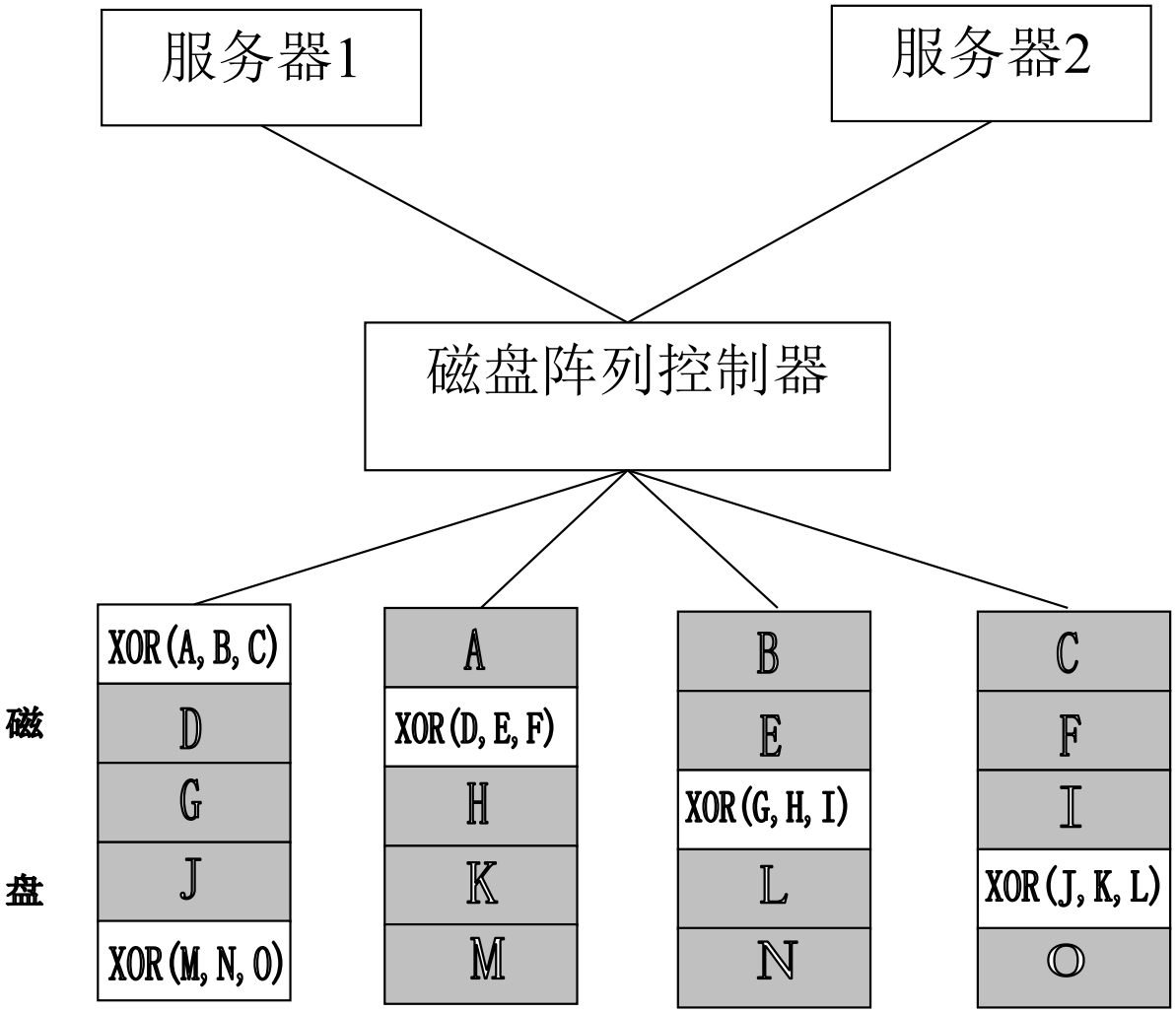
4.3.5 磁盘阵列

- **RAID 4**
  - 也称为块交错（**block-interleaved**）奇偶校验结构，运用块级条带化，像在**RAID 0**中一样。此外对于**N**个磁盘上的对应块，在另外单独的磁盘上保存奇偶校验块。如果**N**个磁盘中的某一个失效了，奇偶校验块可与其他磁盘上保存的对应块一起来恢复存储失效的块。

4.3.5 磁盘阵列

- **RAID 5**
  - 独立的数据磁盘与分布式校验（**Independent Data disks with distributed parity blocks**），与级别**4**不同的是，把数据和奇偶校验分布到所有的**N+1**个磁盘上，而不是把数据存储在**N**个磁盘上，把奇偶校验值存储在另一个磁盘上。
  - 各盘阵生产厂家的奇偶块分布规则可能不同。通过把奇偶校验位分布于阵列中的所有磁盘上，**RAID 5**避免了在**RAID 4**中可能出现的过度使用单独一个奇偶校验磁盘的情况。

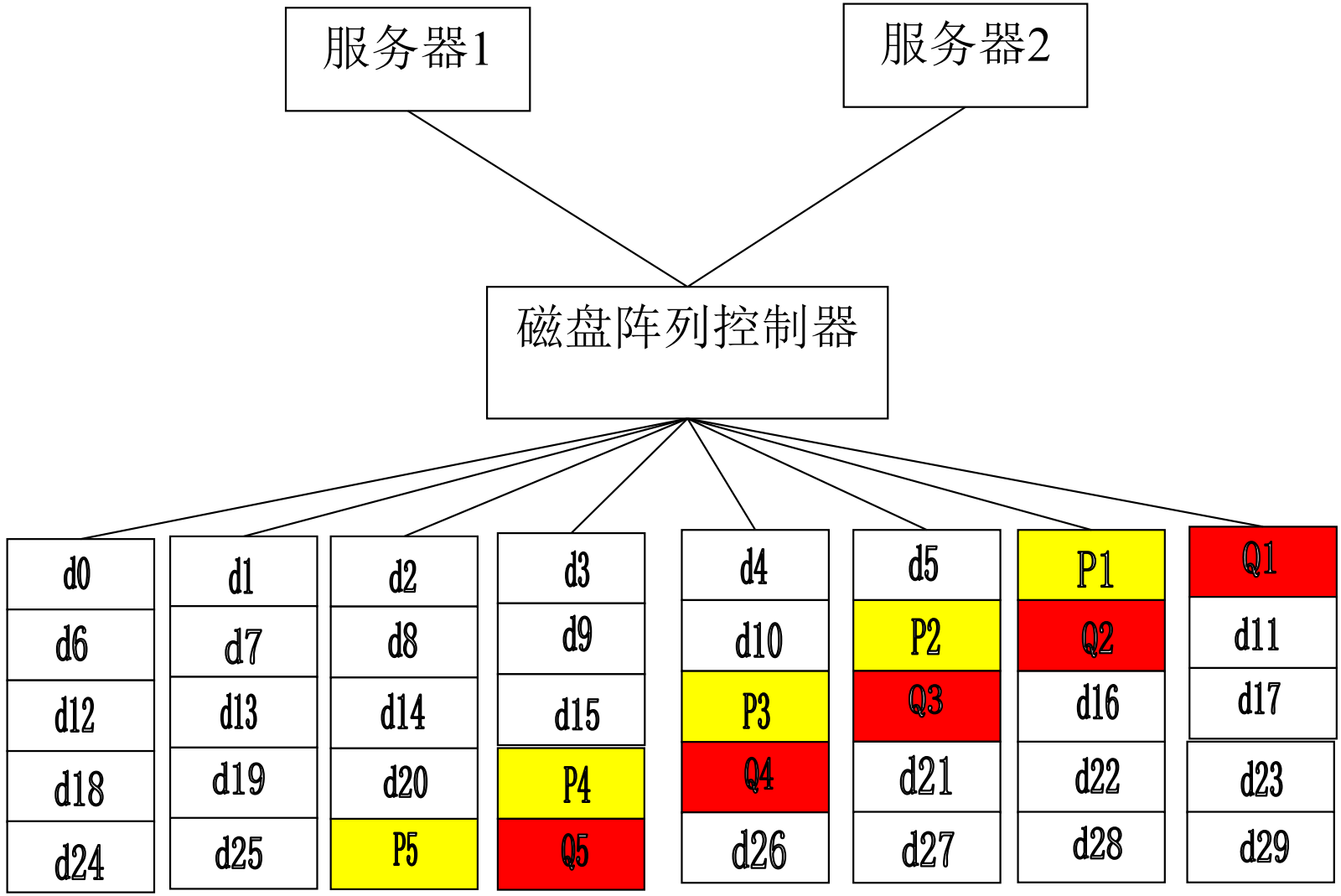
4.3.5 磁盘阵列


服务器1
服务器2
磁盘阵列控制器
磁
盘
XOR(A, B, C)
D
G
J
XOR(M, N, O)
A
XOR(D, E, F)
H
K
M
B
E
XOR(G, H, I)
L
N
C
F
I
XOR(J, K, L)
O

4.3.5 磁盘阵列

- **RAID 6**
  - **P+Q**冗余模式，与**RAID 5**相比，增加了第二个独立的奇偶校验信息块。
  - 即使两块磁盘同时失效，也不会影响数据的使用。
  - 但需要分配给校验信息更大的磁盘空间，相对于**RAID 5**有更大的“写损失”。因此也称为独立的数据硬盘与两个独立分布式校验方案（**Independent Data disks with two independent distributed parity schemes**）。

4.3.5 磁盘阵列


服务器1
服务器2
磁盘阵列控制器
d0
d6
d12
d18
d24
d1
d7
d13
d19
d25
d2
d8
d14
d20
P5
d3
d9
d15
P4
Q5
d4
d10
P3
Q4
d26
d5
P2
Q3
d21
d27
P1
Q2
d16
d22
d28
Q1
d11
d17
d23
d29