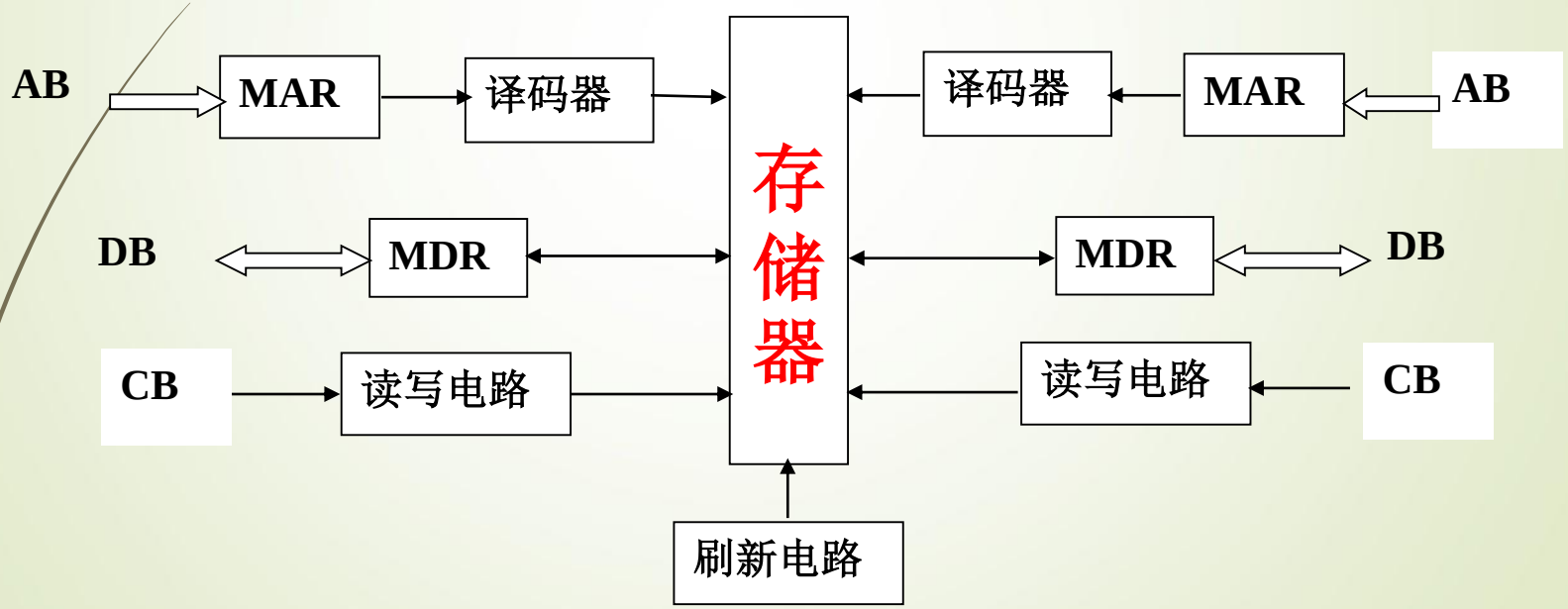
第4章　存 储 器

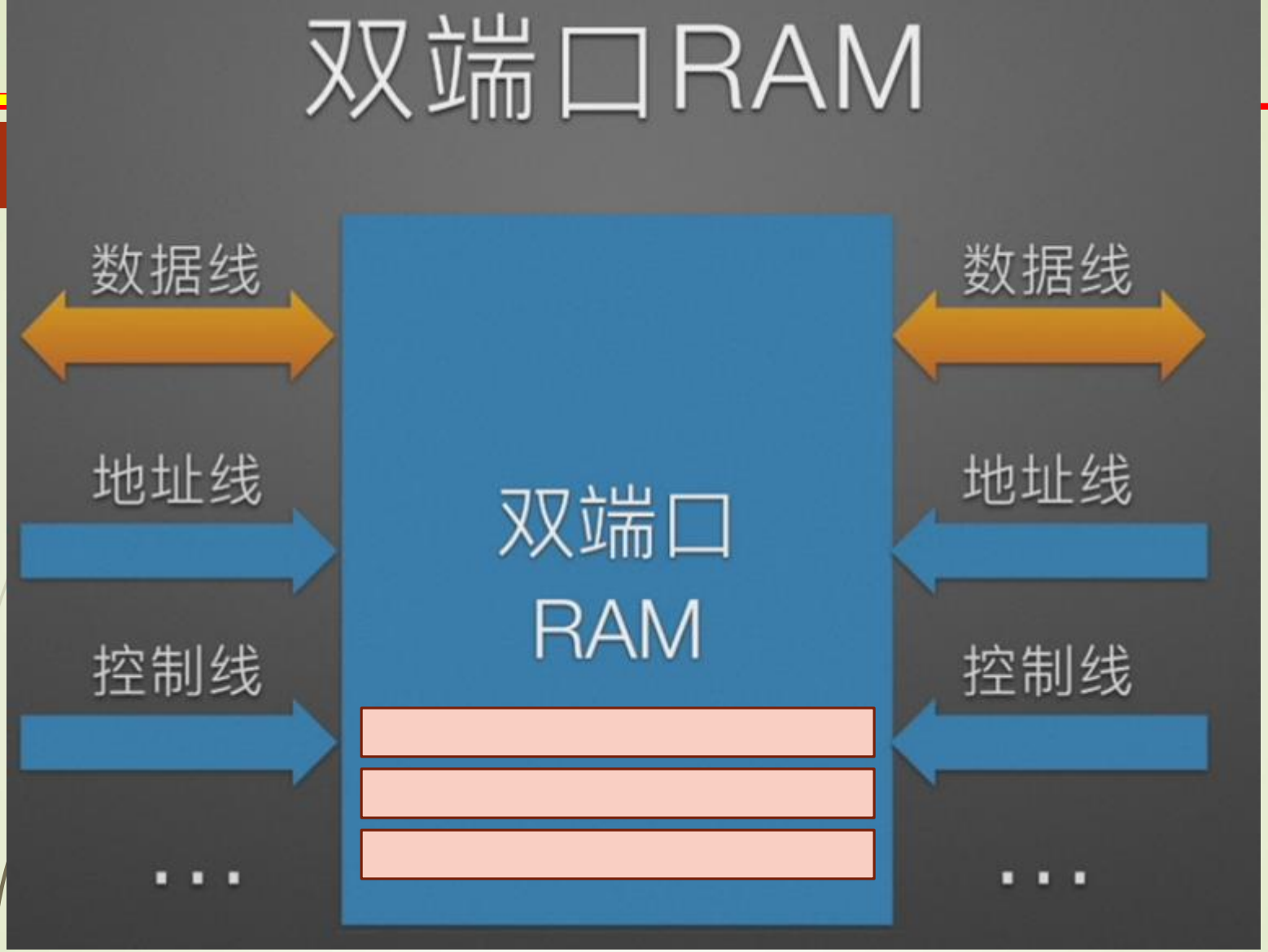
多端口存储器 (Multi-SRAM)

- 单端口存储器任一时刻只能接收来自CPU和I/O中一方的访存请求,是串行工作模式。
- Multi-SRAM：为需要数据共享而设计，
  - 进行数据共享的多个不同设备需要异步访问保存在同一存储体的信息
- 具有两个或多个端口，不同端口分别连接不同设备，通常具有输出允许OE、写允许WE和端口允许CE三个控制信号
- 对两端口同时读，不仲裁
- 对一个端口读，同时对另一个端口写，或同时对两端口写，需要仲裁。
  - 读写冲突使数据不可预测：读后写、写后读；同时写
  - 解决方法：增加额外读周期；使写操作的多组指定地址只通过一个端口输入。

双端口存储器

双端口随机存储器DPARAM(Dual-port Access RAM)由两个访问端口，即两套MAR、MDR、地址译码器和读写电路，两个端口分别连接两套独立总线，同时接收来自两方的访存请求，使存储器并行工作。





同一个存储器有两套独立的读写控制电路，就称为**双端口存储器。**

双端口存储器

- 两个访问端口独立工作互不干扰，只有当两个端口试图在同一时间访问同一地址单元，才会发生冲突。
- 存储器仲裁逻辑根据两端口访问请求到达存储器的微小时间差来决定首先为哪一方服务，被延缓访问的另一方被耽误的时间很短，小于一个存取周期。

双端口存储器的仲裁

●无冲突读写控制

(1) 当两个端口地址不同时，在两个端口上进行读写操作，一定不会发生冲突;

(2) 当两个端口地址相同时，在两个端口上不同时进行读写操作，一定不会发生冲突;

(3) 当两个端口地址相同时，在两个端口上同时进行读操作，一定不会发生冲突。

●有冲突读写控制

(1) 当两个端口地址相同时，在两个端口上同时进行写操作，一定会发生冲突;

(2) 当两个端口地址相同时，在两个端口上同时一读- -写，一定会发生冲突。解决办法:置busy为0，来避免对同一地址存储单元的同时访问。

单选题 0.5分

设置

[例]设存储器容量为32 字，字长64bit,模块数m=4,分别用顺序方式和交叉方式进行组织。存储周期T=200ns,数据总线宽度为64位，总线传送周期为50ns。若连续读出4个字，顺序存储器(高位交叉存储器)和交叉存储器(低位交叉存储器)的带宽各是多少？（　　）

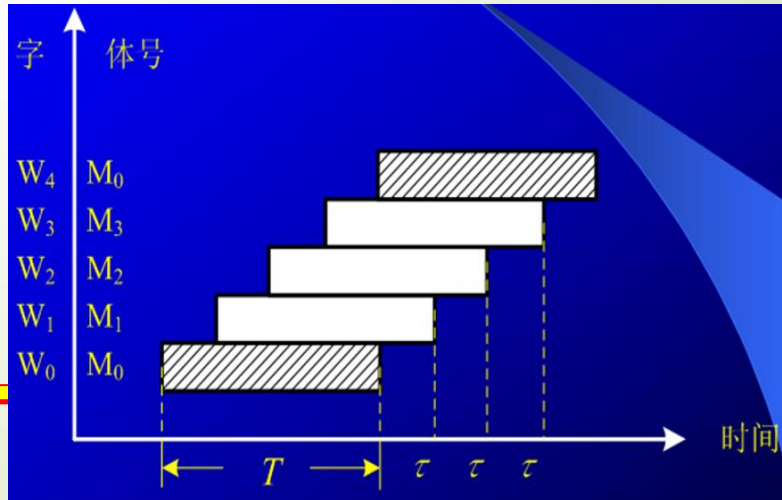
提交

A 731 Mbit/s， 320 Mbit/s

B 320 Mbit/s， 731 Mbit/s

C 320 Mbit/s， 320Mbit/s

D 731 Mbit/s， 731 Mbit/s



解:

交叉存储器和顺序存储器连续读出4个字(m=4)的信息总量都是t=64bitx4= 256bit

交叉存储器和顺序存储器连续读出4个字所需的时间分别是:

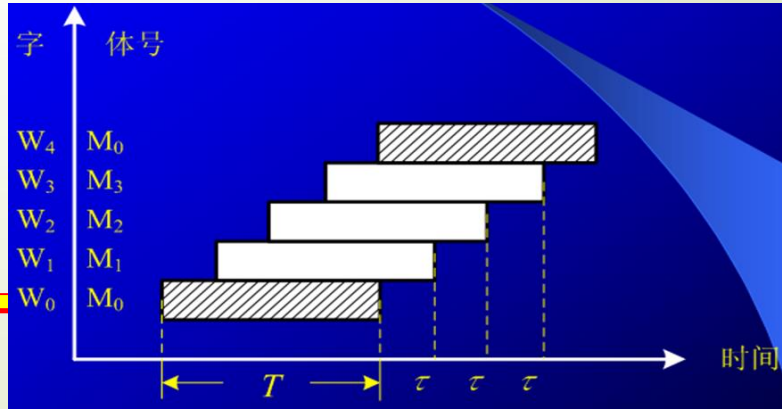
t1=T+50(m-1)=200ns+ 150ns= 350ns

t2=mT=4x200ns= 800ns

则交叉存储器和顺序存储器的带宽分别是

W1=t/t1=256bit/350ns=731Mbit/s

W2=V/z=256bit/800ns=320Mbit/s



单选题 0.5分

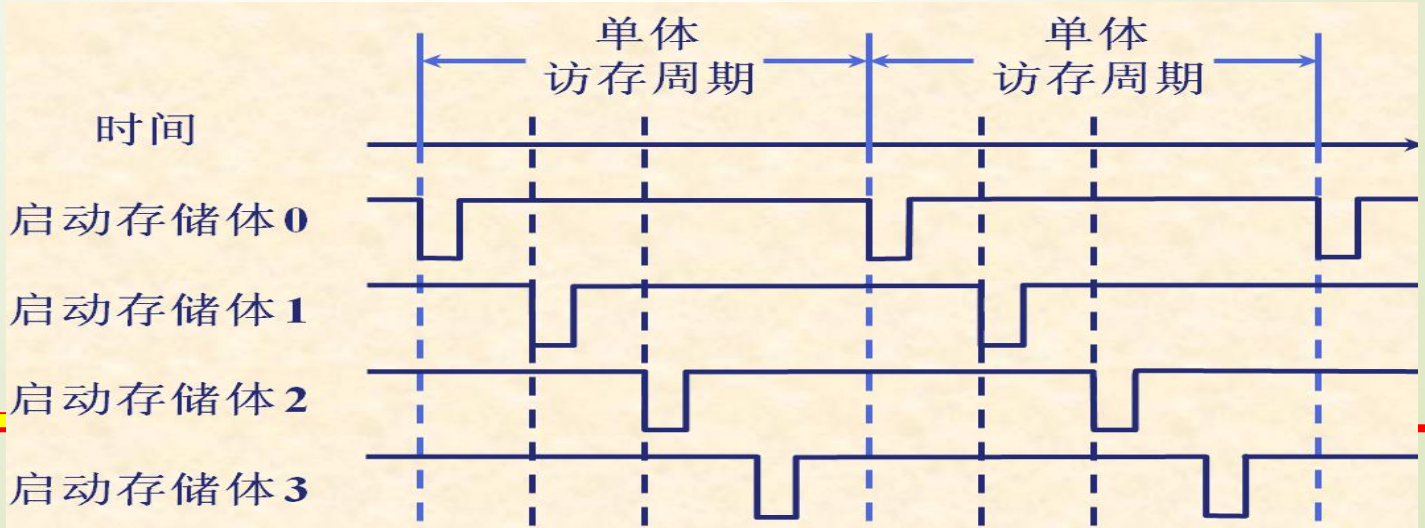
采用8体并行低位交叉存储器，设每个体的存储容量为32Kx16位，按16位字编址，存储周期为400ns，下述说法中正确的是：

A. 在400ns内，存储器可向CPU提供2^7位二进制信息

B. B. 在100ns内，每个体可向CPU提供2^7位二进制信息

C. 在400ns内，存储器可向CPU提供2^8位二进制信息

D. 在100ns内，每个体可向CPU提供2^8位二进制信息



单选题 0.5分

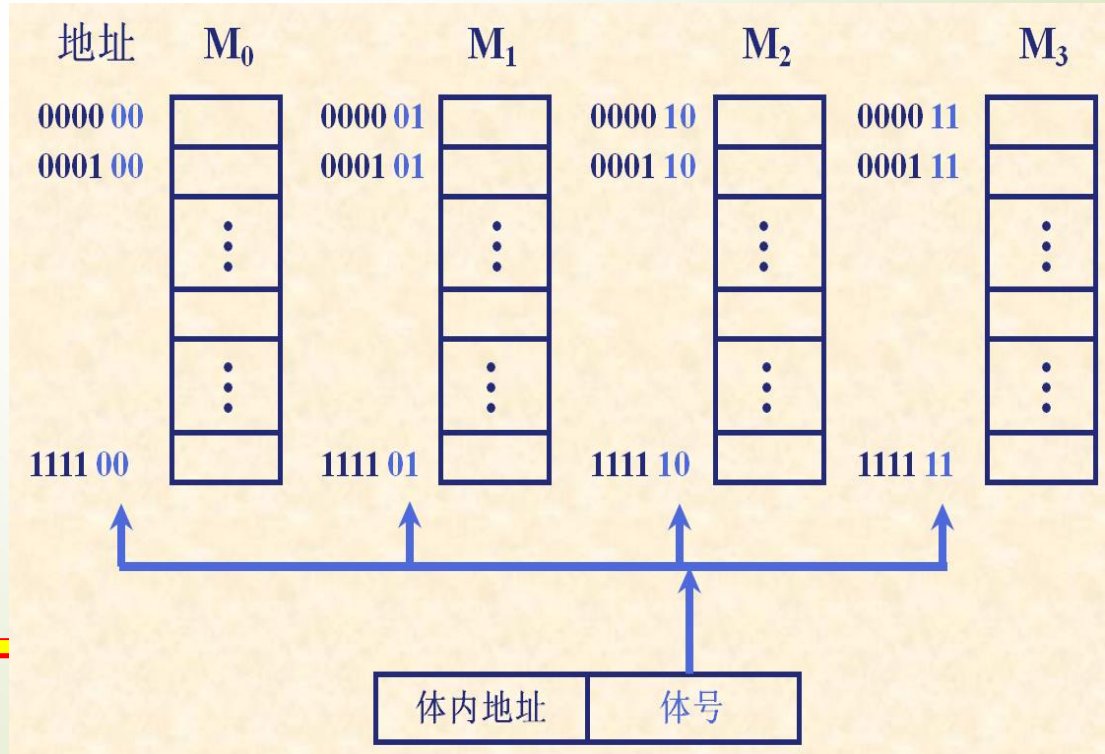
某计算机使用4体低位交叉编址存储器，假定在存储器总线上出现的主存地址（十进制）序列为8005，8006，8007,8008，8001，8002，8003，8004, 8000，则可能发生访存冲突的地址对是(　)。

A.8004和8008

B.8002和8007

C.8001和8008

D.8000和8004



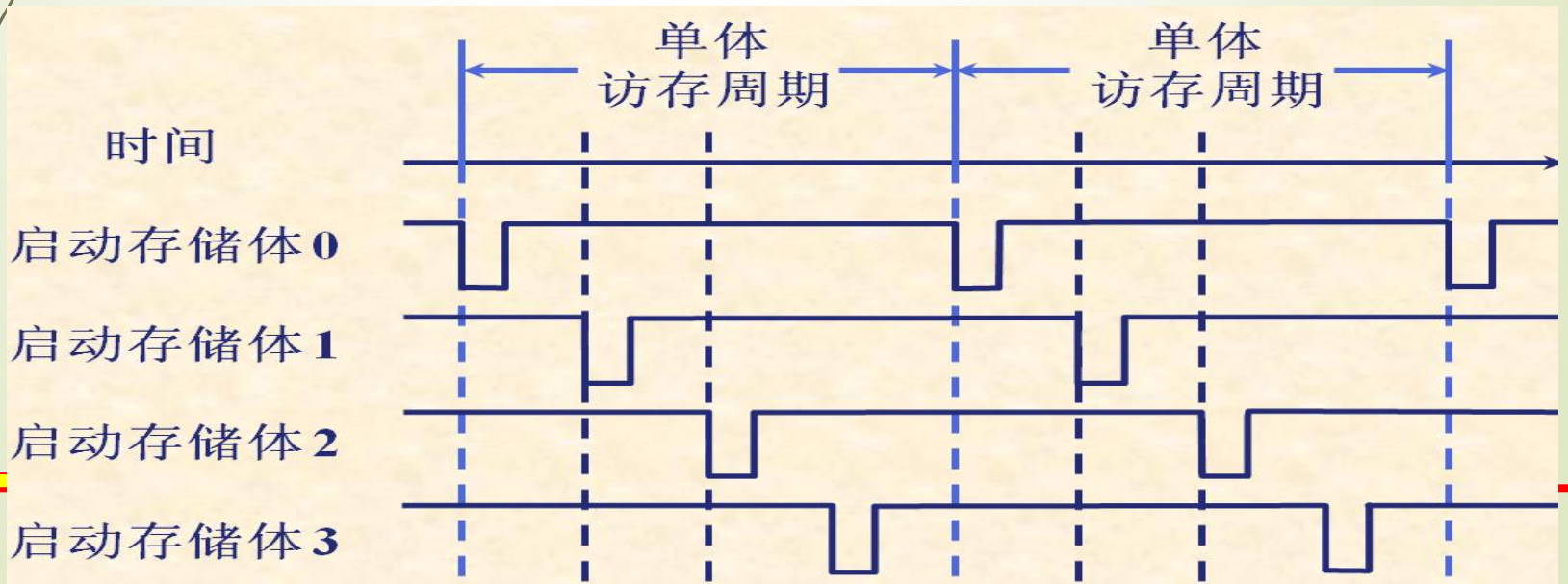
解答：

采用4体低位交叉编址方式：

8000,8004,8008在一个体中。8001，8005在一个体中。

8002,8006在一个体中。

8003, 8007在一个体中。



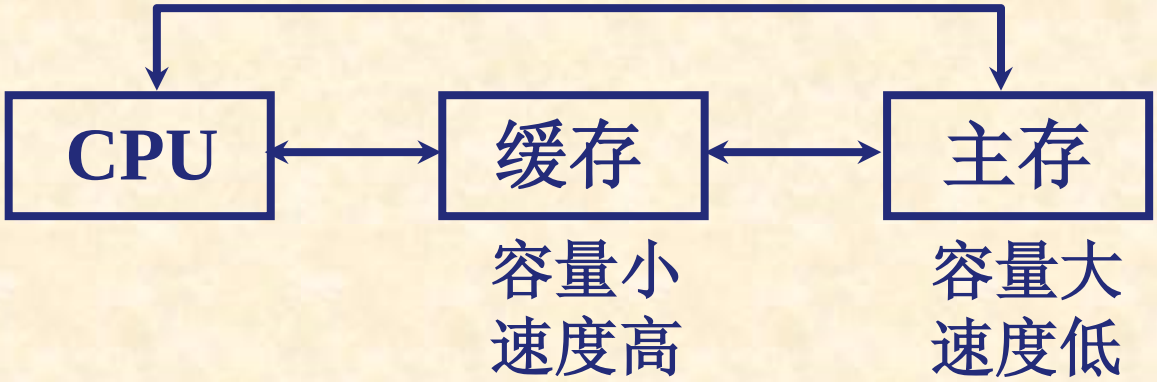
4.3 高速缓冲存储器

一、概述

1. 问题的提出

避免 CPU “空等” 现象

CPU 和主存（DRAM）的速度差异



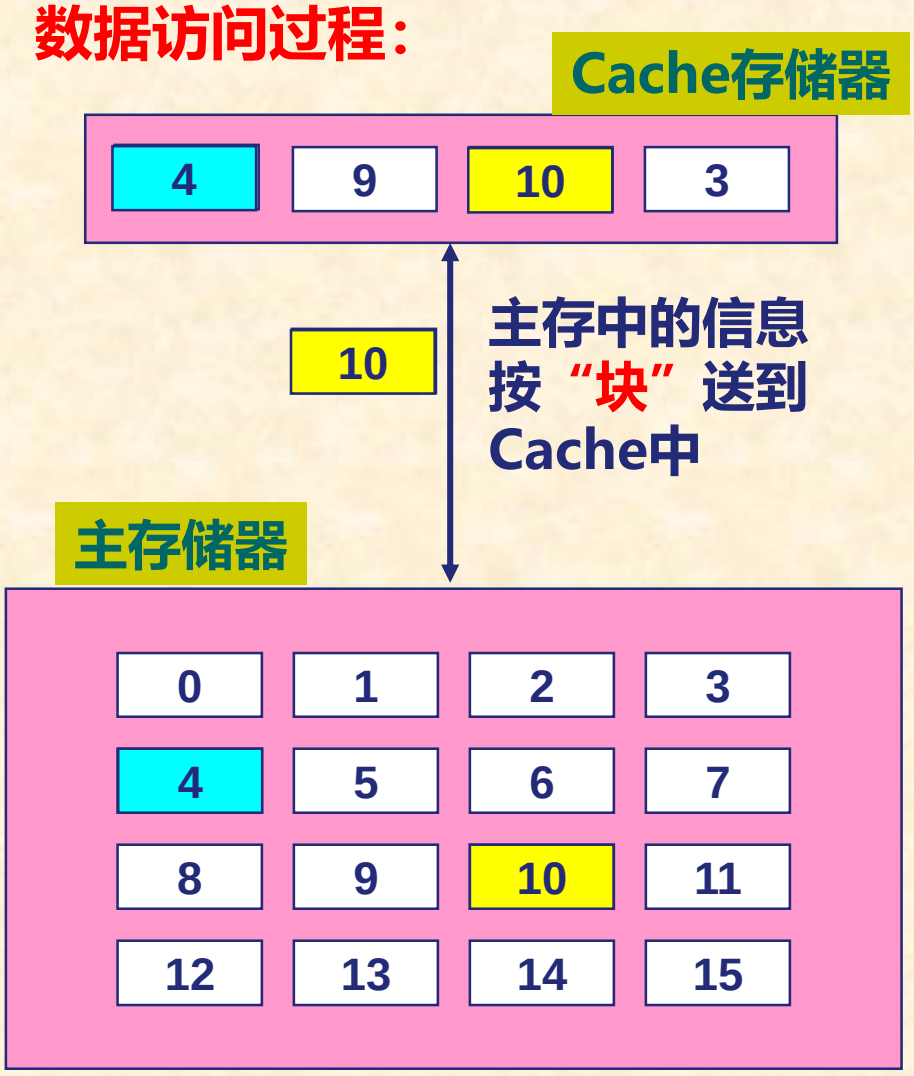
理论依据：程序访问的局部性原理。即：在一个较短的时间间隔内，CPU对局部范围的存储器地址频繁访问，而对此地址范围之外的地址访问很少。

程序访问的局部性原理

- 引入Cache的理论依据：程序访问的局部性原理。
  - 大量典型程序的运行情况分析结果表明，在一个较短的时间间隔内，CPU对局部范围的存储器地址频繁访问，而对此地址范围之外的地址访问很少。
- 程序具有访问局部性特征的原因
  - 指令：指令按序存放，地址连续，循环程序段或子程序段重复执行
  - 数据：连续存放，数组元素重复、按序访问
- 程序访问局部性分为空间局部性和时间局部性
- 基于程序访问的局部性使访存要求能快速得到响应
  - 在CPU和主存之间设置一个快速小容量的存储器，其中总是存放最活跃（被频繁访问）的程序块和数据，由于程序访问的局部性特征，大多数情况下，CPU能直接从这个高速缓存中取得指令和数据，而不必访问主存。

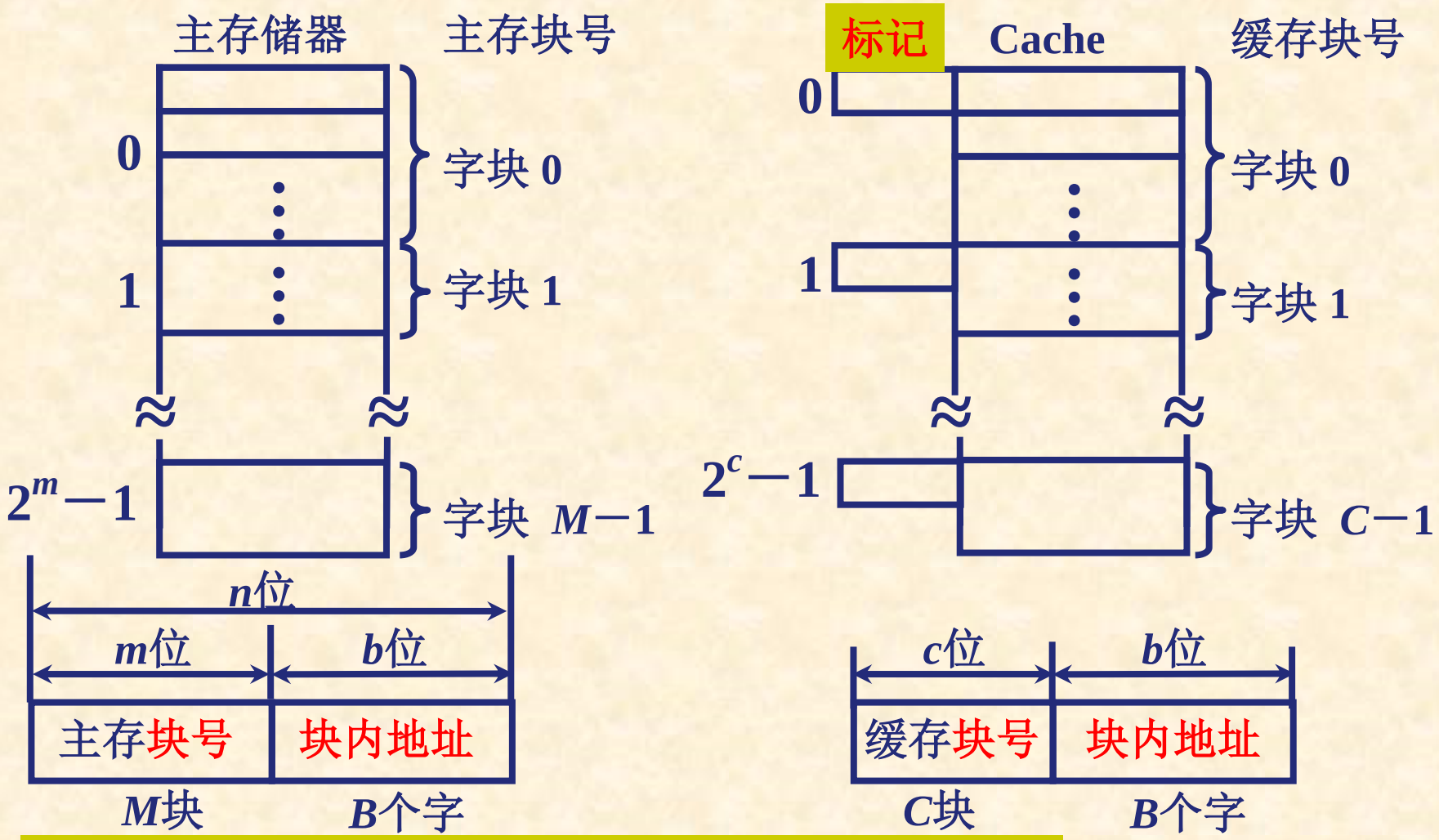
Cache(高速缓存)是什么样的？

- Cache是一种小容量高速缓冲存储器，它由SRAM组成。
- Cache直接制作在CPU芯片内，速度几乎与CPU一样快。
- 程序运行时，CPU使用的一部分数据/指令会预先成批拷贝在Cache中，Cache的内容是主存储器中部分内容的映象。
- 当CPU需要从内存读(写)数据或指令时，先检查Cache，若有，就直接从Cache中读取，而不用访问主存储器。



2. Cache 的工作原理

(1) 主存和缓存的编址



主存和缓存按块存储　　块的大小相同　　*B* 为块长

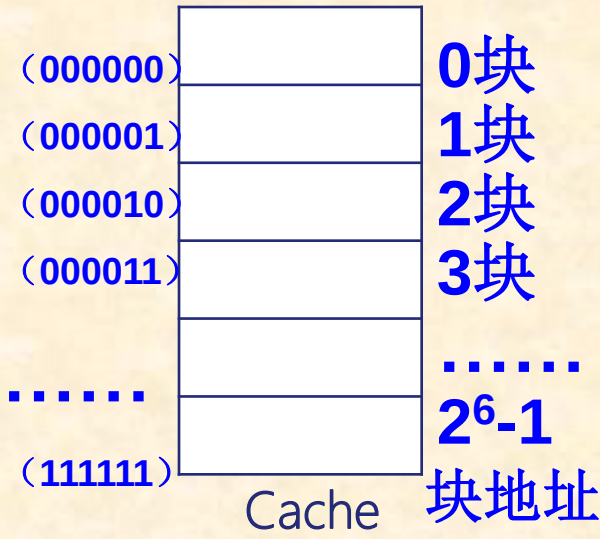
设置

[例]主存容量为512KB，Cache容量4KB,每个字块16个字，每个字32位(按字节寻址)。

(1) Cache地址多少位?有多少个字块?

(2) 主存地址多少位?有多少个字块?

(3) 画出当前主存地址各段位数。



Cache容量为4KB=2^{12}B，因此地址12位;
每个块的容量为16x4B=64B，因此有
2^{12}B/64B = 2^6个块;

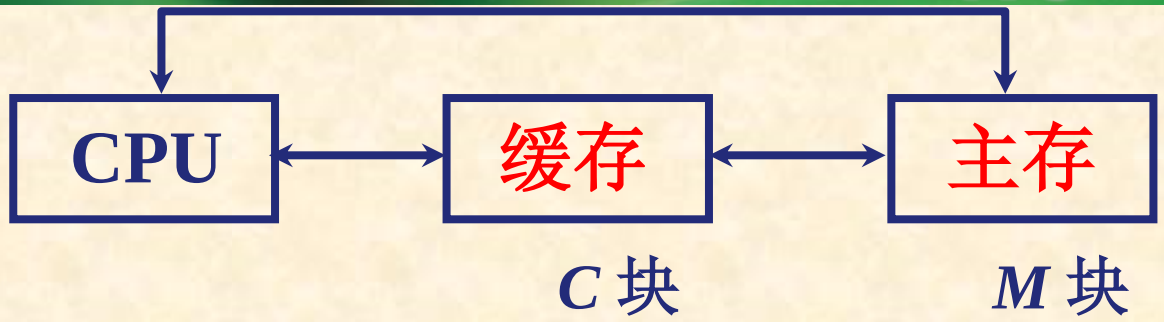
主存容量为512KB=2^{19}B，主存地址19位。
因此有2^{19}B/64B = 2^{13}个块;

(2) 命中与未命中

缓存共有 C 块

主存共有 M 块

$M >> C$



命中(hit)　　主存块 调入 缓存

主存块与缓存块 建立 了对应关系

用 标记记录 与某缓存块建立了对应关系的 主存块块号

未命中
(缺失 miss)　　主存块 未调入 缓存

主存块与缓存块 未建立 对应关系

(3) Cache 的命中率（hit rate）

CPU 欲访问的信息在 Cache 中的 比率

在程序执行期间，设*Nc*为访问Cache命中的次数，*Nm*为访问主存总次数，命中率为*h*，则

$$h = \frac{Nc}{Nc + Nm}$$

命中率 与 Cache 的 容量 与 块长 有关

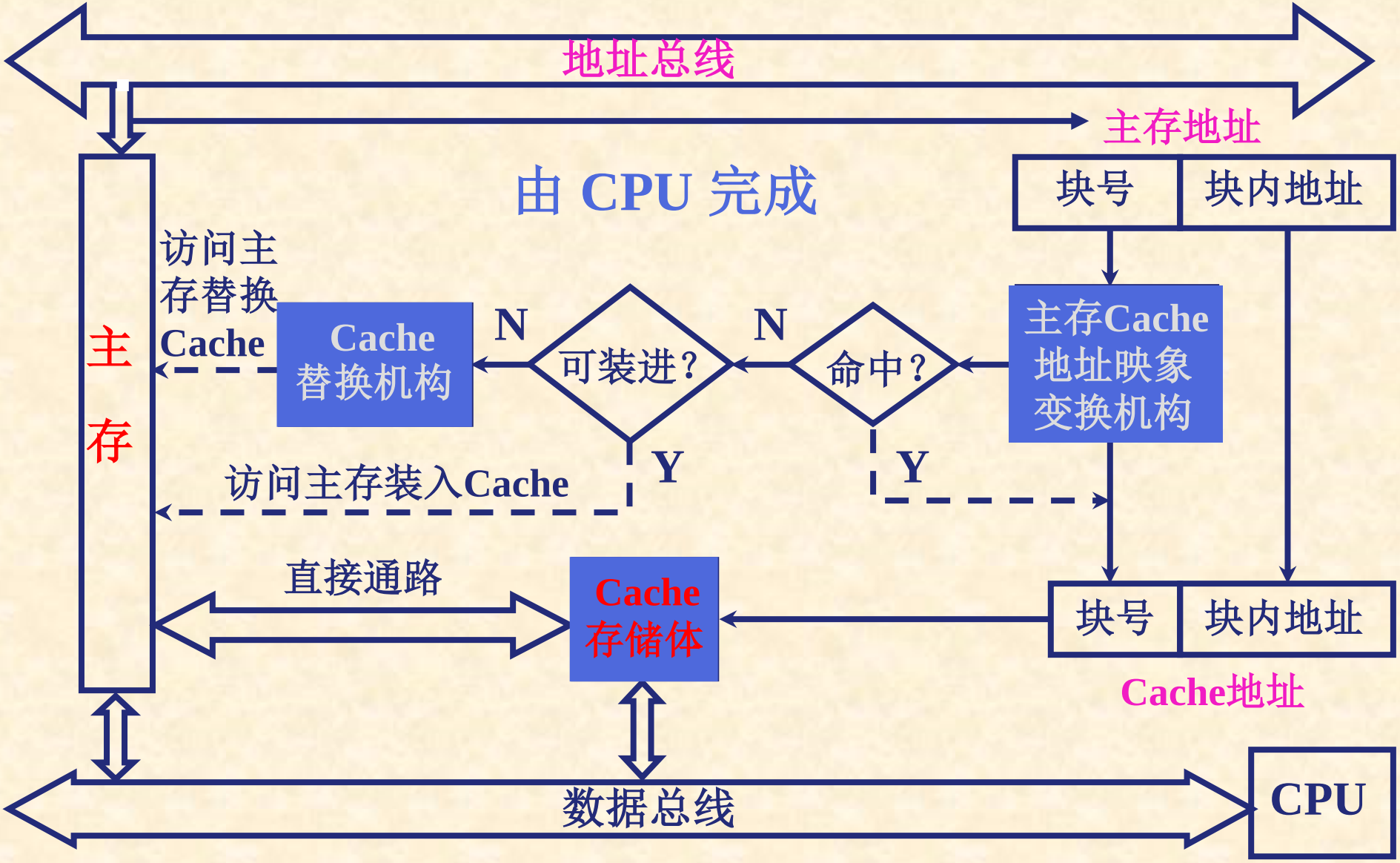
一般每块可取 4 至 8 个字　块长取一个存取周期内从主存调出的信息长度

CRAY_1　16体交叉　块长取 16 个存储字

IBM 370/168　4体交叉　块长取 4 个存储字

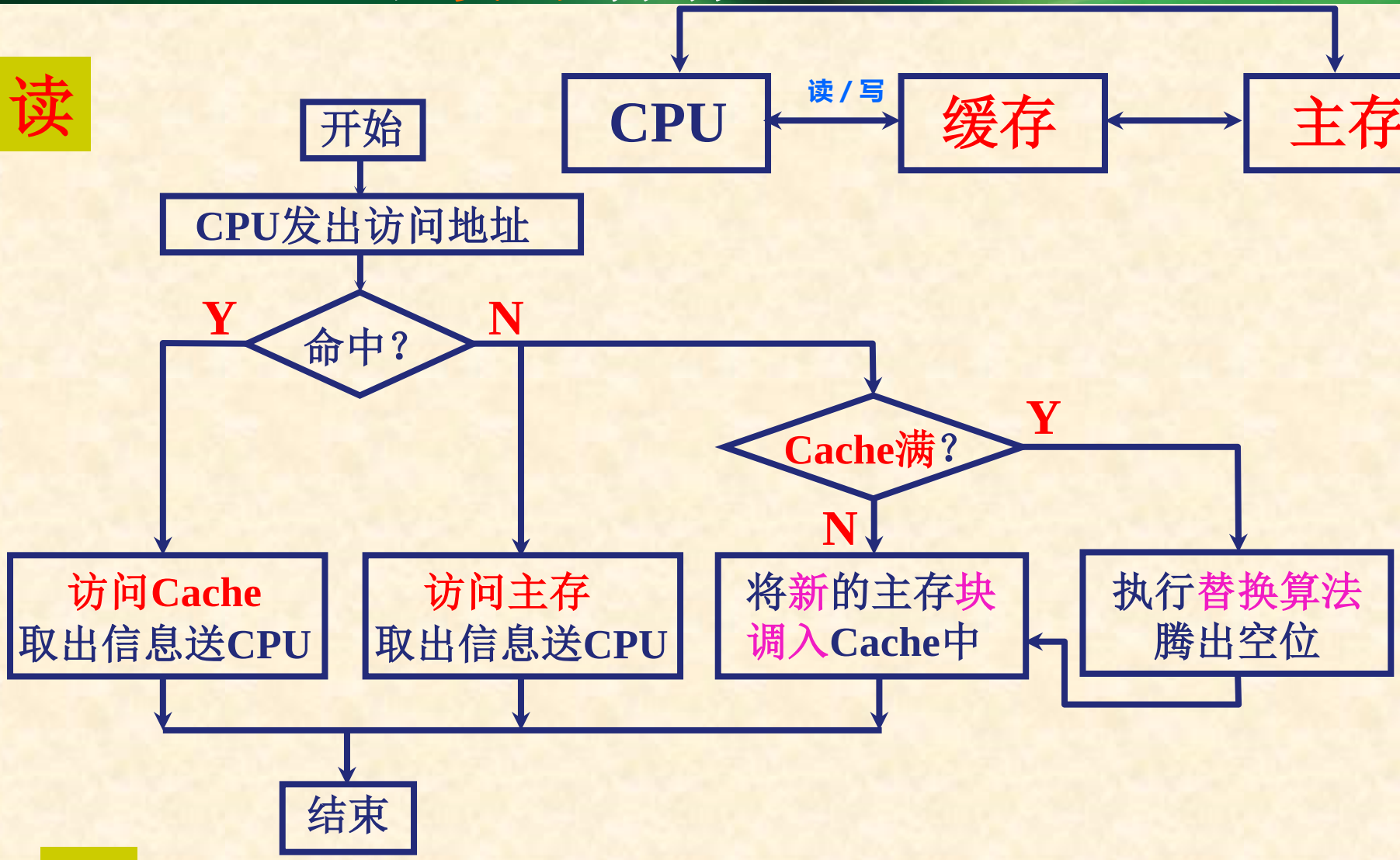
（64位×4 = 256位）

3. Cache 的基本结构



4. Cache 的 读写 操作

读



写

Cache 和主存的一致性

4. Cache 的写操作

- 对Cache块内写入的信息，必须与被映象的主存块内的信息完全一致。
  - 直写（write-through，或称store-through）：随时保证主存内容与Cache的数据始终一致。
    - 但有可能会增加访存次数，因每向Cache写入时，都需向主存写入。
  - 回写（write-back）又称标志交换式（Flag-Swap）：数据每次只是暂时写入Cache，并用标志将该块加以注明，直至该块从Cache替换出时，才写入主存。
    - 这种方法,其速度快，但因主存中的字块未经随时修改，可能失效。
- 信息只写入主存时，同时将相应的Cache块有效位置置“0”,表明此Cache块已失效，需要时从主存调入。还有一种可能，被修改的单元根本不在Cache内，因此写操作只对主存进行。

5. Cache 的改进

(1) 增加 Cache 的级数

片载（片内）Cache

片外 Cache

(2) 统一缓存和分开缓存

指令 Cache　　数据 Cache

与主存结构有关

与指令执行的控制方式有关　　是否流水

Pentium	8K 指令 Cache	8K 数据 Cache
PowerPC620	32K 指令 Cache	32K 数据 Cache

二、Cache 映射(Cache Mapping) 4.3

- 什么是Cache的**映射功能**?
  - 把访问的局部主存区域取到Cache中时，该放到Cache的何处?
  - Cache槽比主存块少，多个主存块映射到一个Cache槽中
- **地址映象**与变换
  - 在主存的地址和Cache地址之间建立一种确定的逻辑关系，也就是根据主存的地址来构成Cache地址。这种地址间的逻辑关系称为映射

二、Cache 映射(Cache Mapping)

- 如何进行映射/映象
  - 把主存空间划分成大小相等的主存块（Block）
  - Cache中存放一个主存块的对应单位称为槽（Slot）或行（line）或块（Block）
  - 将主存块和Cache行按照以下三种方式进行映射
    - 直接(Direct)：每个主存块映射到Cache的固定行中
    - 全相联(Full Associate)：每个主存块映射到Cache的任意行中
    - 组相联(Set Associate)：每个主存块映射到Cache的固定组中的任意一行中

1. 直接映像（direct mapped）

- 直接映像：把主存的每一块映射到Cache中的一个固定的Cache行（槽） 的方式。
- 最简单的地址映像方式，地址变换速度快，但块冲突的概率较高，当程序往返访问两个相互冲突的块中的数据时，Cache的命中率将急剧下降。
- 映射公式：

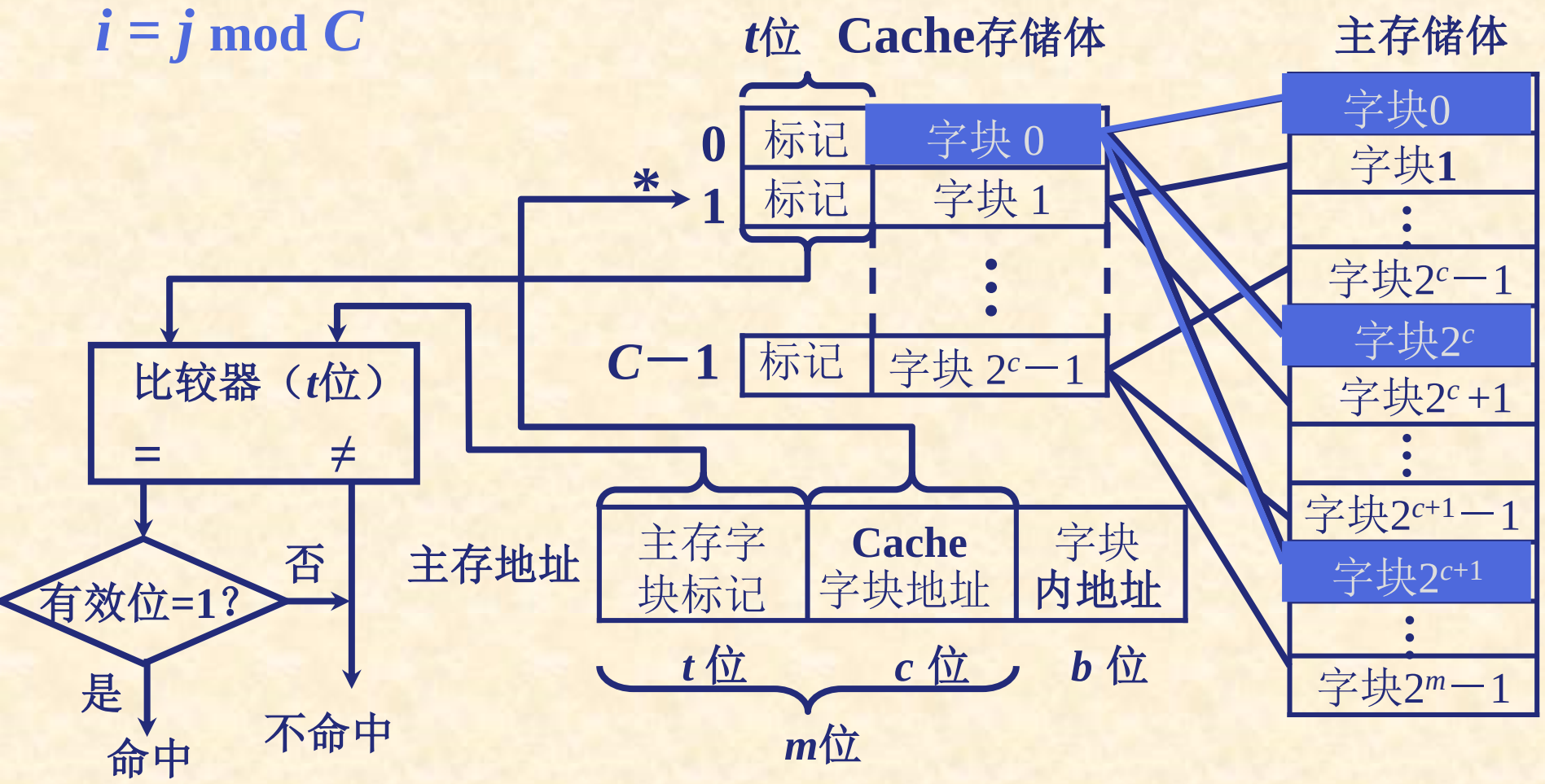
Cache行号=主存块号 mod Cache行数

举例：4=100 mod 16 （假定Cache共有16行）

(说明：主存第100块应映射到Cache的第4行中。)

块（行）都从0开始编号

1. 直接映像



每个缓存块 i 可以和 若干 个 主存块 对应

每个主存块 j 只能和 一 个 缓存块 对应

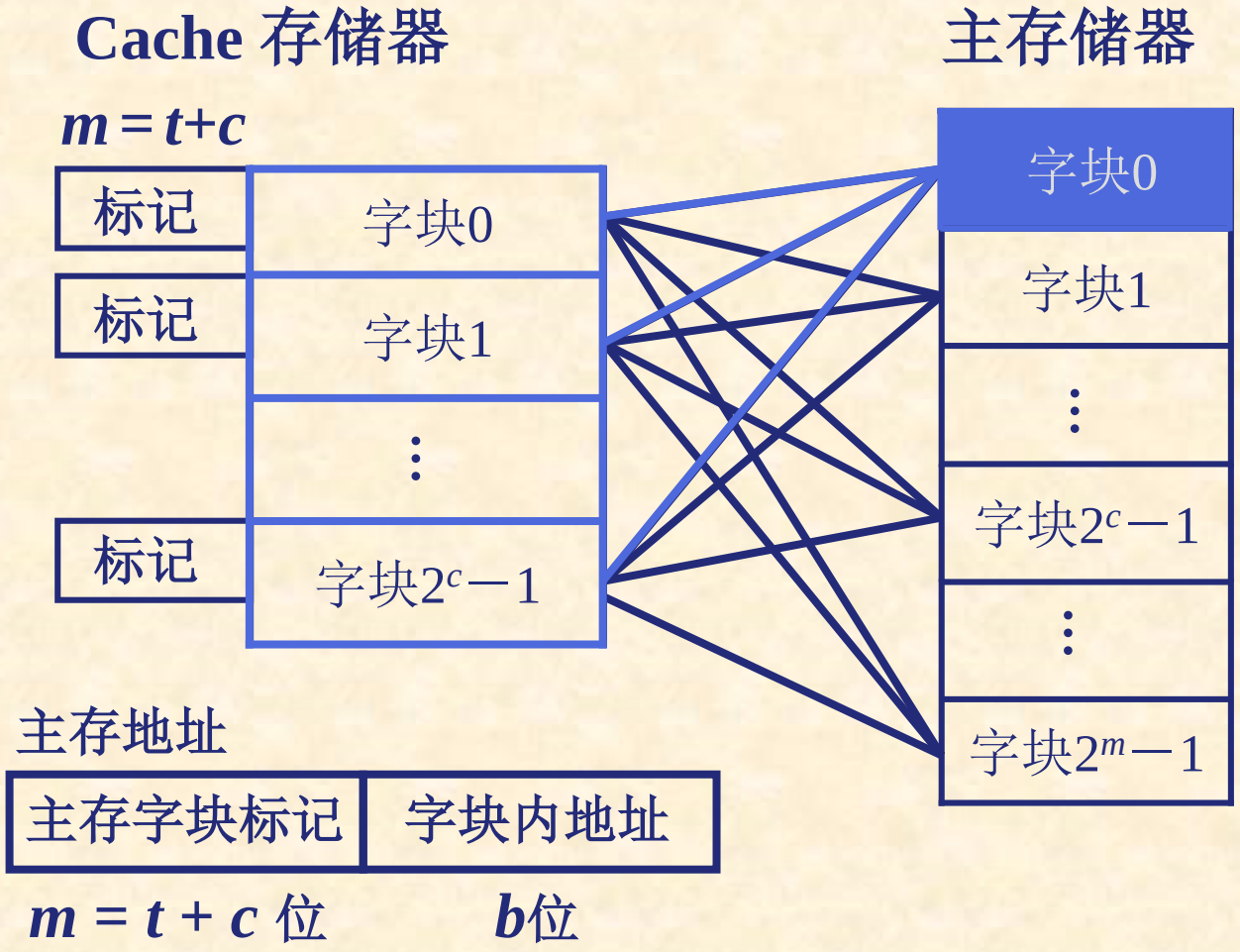
1. 直接映像

- 特点：
  - 容易实现，命中时间短
  - 无需考虑淘汰（替换）问题
  - 但不够灵活，Cache存储空间得不到充分利用，命中率低
  - 例如，需将主存第0块与第16块同时复制到Cache中时，由于它们都只能复制到Cache第0行，即使Cache其它行空闲，也有一个主存块不能写入Cache。这样就会产生频繁的 Cache装入。

2. 全相联映像（fully associative）

- 每个主存块都可映像到任何Cache块的地址映像方式。
- 在全相联映像方式下，主存中存储块的数据可调入Cache中的任意块。
- 命中率实现比较复杂。

2. 全相联映像



主存 中的 任一块 可以映像到 缓存 中的 任一块

3.组相联映射（Set Associative）

- 组相联映像指的是将存储空间分成若干组，主存块与Cache组之间是直接映像，而组内各块之间则是全相联映像。
- 将Cache所有行分组，把主存块映射到Cache固定组的任一行中。也即：组间直接映像(模映射)、组内全相联映象（全映射）。映射关系为：

  Cache组号=主存块号 mod Cache组数

  举例：假定Cache划分为：8K字=8组×2行/组×512字/行

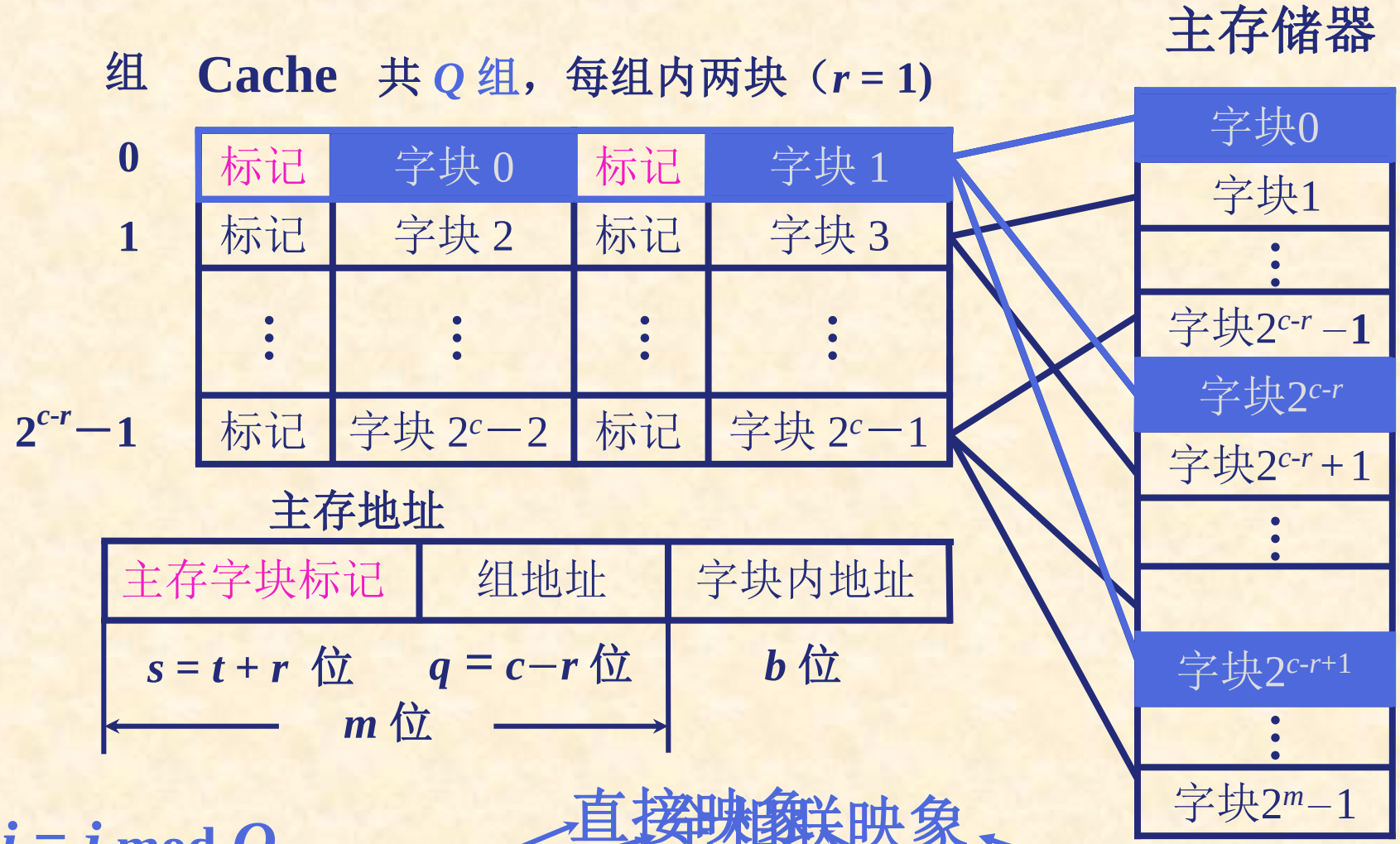
  4=100 mod 8

  (主存第100块应映射到Cache的第4组的任意行中。)

组相联映射（Set Associative）

- 特点：
  - 结合直接映射和全相联映射的优点。当Cache组数为1时，变为相联映射；当每组只有一个槽时，变为直接映射。
  - 每组2或4行（称为2-路或4-路组相联）较常用。通常每组4行以上很少用。在较大容量的L2 Cahce和L3 Cahce中使用4-路以上。

3. 组相联映像（set associative）



$i = j \bmod Q$

直接映象 全相联映象

某一主存块 j 按模 Q 映射到 缓存 的第 i 组中的 任一块

4. Cache中的位数（容量）

- 由于每个Cache的地址可能对应于存储器中不同的地址，因此需要在Cache中加标记(tag)，标记必须能判断Cache中的字是否为所请求的地址信息。
- 标记只包含存储器地址的高位部分（存储器地址的高位部分用作选定Cache地址）
- Cache中包含一位有效位(valid bit)用于说明Cache块是否含有有效地址
  - 有效位用于判断Cache块中是否有有效信息。例如当处理器启动时Cache是空的，此时标记字段中的值是没有意义的。即使执行了数条指令，Cache中的一些块仍为空，此时也需要用有效位说明这些单元的标记应被忽略。

Cache块内容

有效位	标记	数据

主存地址

直接映射：

标记	Cache行号	块内地址

全相联映射：

标记（主存块号）	块内地址

组相联映射：

标记	Cache组号	块内地址

一个Cache行的内容

有效位	标记	数据

4. Cache的容量

Cache块	有效位	标记	数据

- **Cache不仅存储数据，而且存储标记，故Cache中所需总位数是Cache的大小和地址位数的函数。**
- **例如： 假设一个32位字节的地址以及一个大小为2^n个字直接映像的Cache，每块有1个字（4个字节），则标记字段位数为32-n-2，其中n为Cache块地址，2位为块内地址。而块大小为1个字（32位）且地址大小也为32位，而这样一个Cache的位数为**

$$2^n \times [32+(32-n-2)+1] = 2^n \times (63-n)$$

4. Cache的容量

- 例题 假设一个直接映像的Cache 有64KB数据，地址为32位，块大小为1个字（4个字节），那么该Cache的容量是多少位？

$2^{14} \times [32+(32-14-2)+1] = 2^{14} \times 49 = 784 \times 2^{10} = 784K$位

例. 设某机主存容量为16MB，Cache数据区的容量为16KB。每个字块有8个字，每个字32位。设计一个四路组相联映像（即Cache每组内共有4个字块）的Cache组织，要求：

（1）画出主存地址字段中各段的位数。

解：根据每个字块有8个字，每个字32位，得出主存地址字段中字块内地址字段为5位。

根据Cache容量为16KB=2^{14}B，字块大小为2^5B，得Cache共有2^9块，故c=9。根据四路组相联映像2^r=4，得r=2，则q=c-r=7。

根据主存容量为16MB=2^{24}B，得出主存地址字段中主存字块标记位数为24-7-5=12。

主存地址字段格式如下图所示：主存字块标记组地址字块内地址

12位7位5位

主存字块标记	组地址	字块内地址
12位	7位	5位

设某机主存容量为16MB，Cache数据区的容量为16KB。每个字块有8个字，每个字32位。设计一个四路组相联映像（即Cache每组内共有4个字块）的Cache组织，要求：

（2）设Cache初态为空，CPU依次从主存第0、1、2、…、99号单元读出100个字（主存一次读出一个字），并重复此次序读8次，问命中率是多少？

解：由于每个字块中有8个字，而且初态Cache为空，因此CPU读第0单元时，未命中，必须访问主存，同时将该字所在的主存块调入Cache第0组中的任一块内，接着CPU读1～7号单元时，均命中。同理CPU读第8、16、…、96号单元时均未命中。可见CPU在连续读100个字中共有13次未命中，而后7次循环读100个字全部命中，命中率为：

$$\frac{100\times 8-13}{100\times 8}\times 100\%=98.375\%$$

设某机主存容量为16MB，Cache数据区的容量为16KB。每个字块有8个字，每个字32位。设计一个四路组相联映像（即Cache每组内共有4个字块）的Cache组织，要求：

（3）若Cache的速度是主存容量的6倍，试问有Cache和无Cache相比，速度提高多少倍？

解：根据题意，设主存存取周期为6t，Cache的存取周期为t，没有Cache的访问时间为6t×800，有Cache的访问时间为，则有Cache 和没有Cache相比，速度提高倍数为：

$$\frac{6t \times 800}{t(800-13)+6t \times 13}-1 \approx 4.5$$

命中率对平均访问时间的影响

- 设H是命中率，则平均访问时间$T = HT_C + (1 - H)(T_C + T_M)$
$= T_C + (1 - H)T_M$

- 例1. 若H=0.85, Tc=1ns, T_M=20ns，则T为多少?
  答： T = 4ns

- 例2. 若命中率H提高到0.95，则结果又如何?
  答： T = 2ns

- 例3. 若命中率为0.99呢?
  答： T = 1.2ns

访存速度与命中率的关系非常大!

The Need to Replace! (何时需要替换？)

- Direct Mapped Cache:
  - 映射唯一，毫无选择，无需考虑替换
- N-way Set Associative Cache:
  - 每个主存数据有N个Cache行可选择，需考虑替换
- Fully Associative Cache:
  - 每个主存数据可存放到Cache任意行中，需考虑替换

结论：若Cache miss in a N-way Set Associative or Fully Associative Cache，则可能需要替换。其过程为：

- 从主存取出一个新块
- 选择一个有映射关系的空Cache行
- 对应的Cache行已被占满而需要调入新的主存块时，必须考虑从cache行中调出一个主存块

三、替换算法

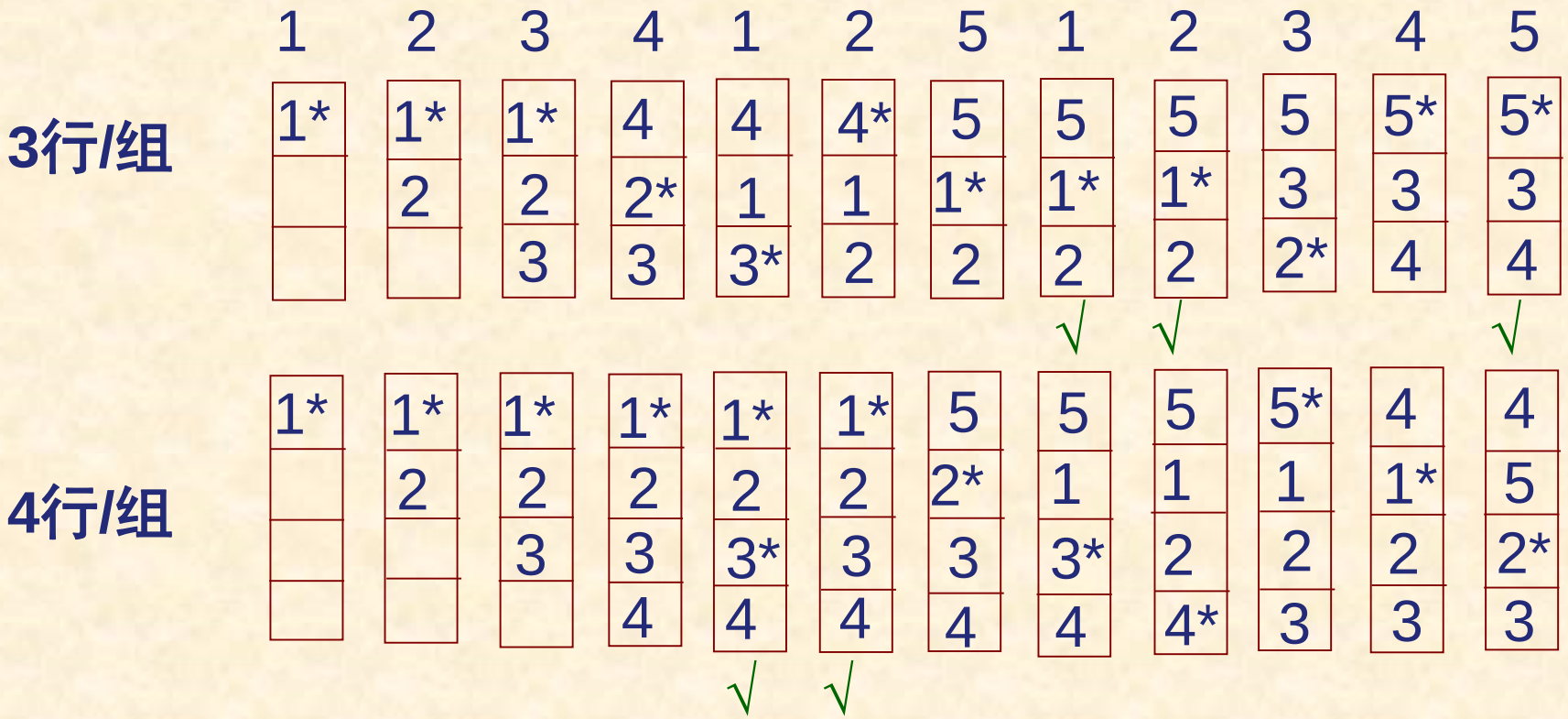
- 选择替换算法的依据是存储器总体的性能，主要是Cache的访问命中率。
- 常用替换算法有：
  - 先进先出FIFO （first-in-first-out）
  - 最近最少使用LRU （ least-recently used）
  - 最不经常使用LFU （ least-frequently used）
  - 随机替换算法（Random）

  等等

替换算法-先进先出（FIFO）

- 总是把最先进入的那一块淘汰掉。

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组的情况。

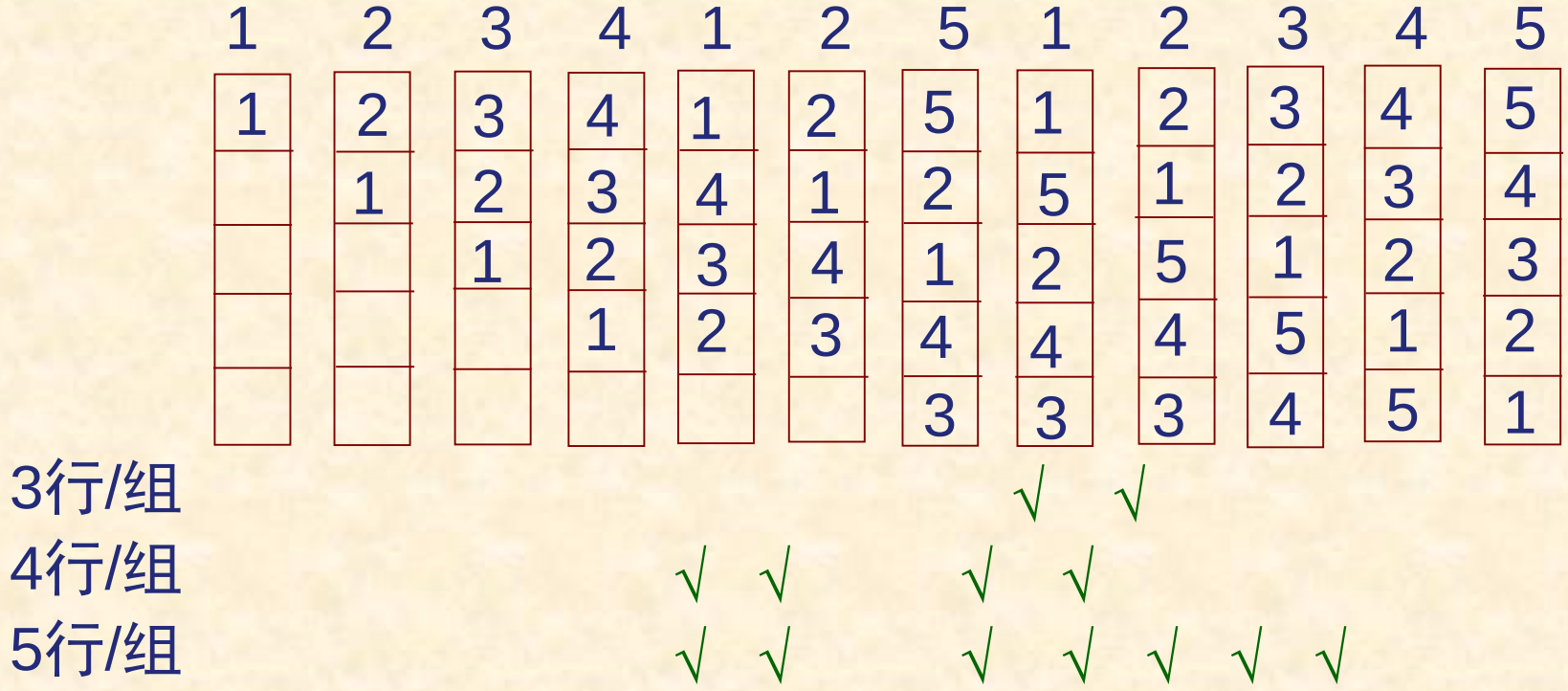


由此可见，FIFO不是一种堆栈算法，即命中率并不随组的增大而提高。

替换算法-最近最少用(LRU)

- 总是把最近最少用的那一块淘汰掉。

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组、 5行/组的情况。



替换算法-最近最少用

- 是一种堆栈算法，它的命中率随组的增大而提高。
- 当分块局部化范围(即：某段时间集中访问的存储区)超过了Cache存储容量时，命中率变得很低。极端情况下，假设地址流是1,2,3,4,1 2,3,4,1,……，而Cache每组只有3行，那么，不管是FIFO，还是LRU算法，其命中率都为0。这种现象称为颠簸(Thrashing / PingPong)。
- 该算法具体实现时，并不是通过移动块来实现的，而是通过给每个cache行设定一个计数器，根据计数值来记录这些主存块的使用情况。这个计数值称为**LRU位**。

具体实现

替换算法-最近最少用

- 计数器变化规则：
  - 每组4行时，计数器有2位。计数值越小则说明越被常用。
  - 命中时，被访问行的计数器置0，比其低的计数器加1，其余不变。
  - 未命中且该组未满时，新行计数器置为0，其余全加1。
  - 未命中且该组已满时，计数值为3的那一行中的主存块被淘汰，新行计数器置为0，其余加1。

1		2		3		4		1		2		5		1		2		3		4		5	
0	1	1	1	2	1	3	1	0	1	1	1	2	1	0	1	1	1	2	1	3	1	0	5
		0	2	1	2	2	2	3	2	0	2	1	2	2	2	0	2	1	2	2	2	3	4
				0	3	1	3	2	3	3	3	0	5	1	5	2	5	3	5	0	4	2	3
						0	4	1	4	2	4	3	4	3	4	3	4	0	3	1	3	1	2

替换算法-其他算法

- 最不经常用（LFU）算法：

  替换掉Cache中引用次数最少的块。LFU也用与每个行相关的计数器来实现。

  （这种算法与LRU有点类似，但不完全相同。）

- 随机算法：

  随机地从候选的cache行中选取一个淘汰，与使用情况无关。

  （模拟试验表明，随机替换算法在性能上只稍逊于基于使用情况的算法。而且代价低！）

程序的局部性原理举例1

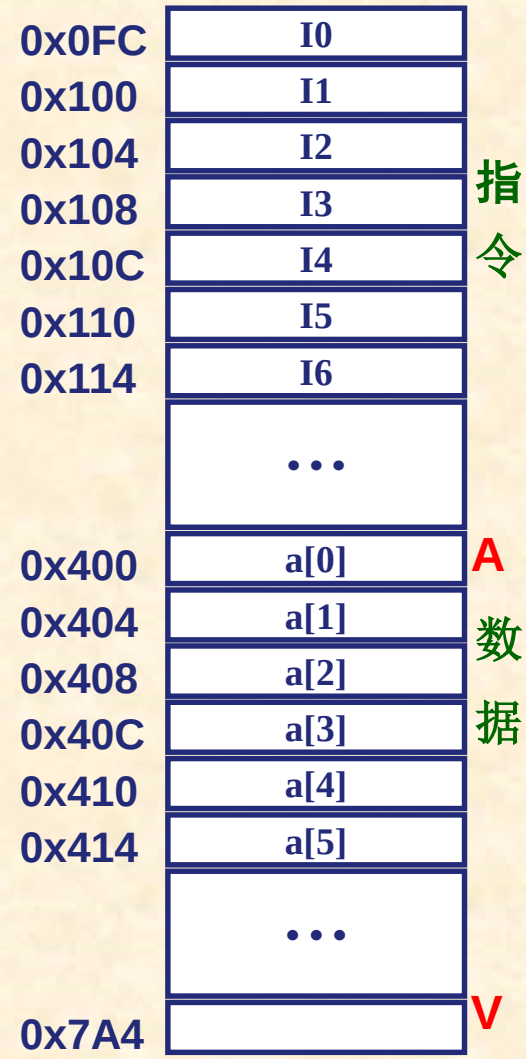
高级语言源程序

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
```

对应的汇编语言程序

```
I0:          sum  <-- 0                    *v = sum;
I1:          ap  <-- A   A是数组a的起始地址
I2:          i   <-- 0
I3:          if (i >= n) goto done
I4:    loop: t   <-- (ap) 数组元素a[i]的值
I5:          sum <-- sum + t   累计在sum中
I6:          ap  <-- ap + 4   计算下个数组元素地址
I7:          i   <-- i + 1
I8:          if (i < n) goto loop
I9:    done: V  <-- sum   累计结果保存至地址v
```

主存的布局：



每条指令4个字节；每个数组元素4字节

指令和数组元素在内存中均连续存放

sum, ap ,i, t 均为通用寄存器；A，V为内存地址

程序的局部性原理举例1

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
*v = sum;
```

问题：指令和数据的时间局部性和空间局部性各自体现在哪里?

指令：　0x0FC（I0）

…

→0x108（I3）

→0x10C（I4）

…

→0x11C（I8）

循环n次

→0x120（I9）

若n足够大，则在一段时间内一直在局部区域内执行指令，故循环内指令的时间局部性好；

数据：只有数组在主存中：

0x400→0x404→0x408

→0x40C→……→0x7A4

按顺序执行，故程序空间局部性好！

数组元素按顺序存放，按顺序访问，故空间局部性好；

每个数组元素都只被访问1次，故没有时间局部性。

主存的布局：

程序的局部性原理举例2

以下哪个对数组a引用的空间局部性更好？时间局部性呢？变量sum的空间局部性和时间局部性如何？对于指令来说，for循环体的空间局部性和时间局部性如何？

程序段A:

```
int sumarrayrows(int a[M][N])
{
   int i, j, sum=0;
       for  (i=0; i<M, i++)
              for (j=0; j<N, j++)  sum+=a[i][j];
       return sum;
}
```

程序段B:

```
int sumarraycols(int a[M][N])
{
    int i, j, sum=0;
        for  (j=0; j<N, j++)
               for (i=0; i<M, i++)  sum+=a[i][j];
        return sum;
}
```

M=N=2048时主存的布局:

数组在存储器中按行优先顺序存放

程序的局部性原理举例2

程序段A的时间局部性和空间局部性分析

(1) **数组a**：访问顺序为a[0][0], a[0][1] ,……, a[0][2047]; a[1][0], a[1][1],…… ,a[1][2047]; ……，与存放顺序一致，故空间局部性好！

因为每个a[i][j]只被访问一次，故时间局部性差！

(2) **变量sum**：单个变量不考虑空间局部性；每次循环都要访问sum，所以其时间局部性较好！

(3) **for循环体**：循环体内指令按序连续存放，所以空间局部性好！

循环体被连续重复执行2048x2048次，所以时间局部性好！

实际上 优化的编译器使循环中的sum分配在寄存器中，最后才写回存储器！

程序的局部性原理举例2

程序段B的时间局部性和空间局部性分析

(1) **数组a**：访问顺序为a[0][0], a[1][0] ,……, a[2047][0];
a[0][1],a[1][1],…… ,a[2047][1];……，与存放顺序不一致，每次跳过2048个单元，若交换单位小于2KB，则没有空间局部性!

(时间局部性差，同程序A)

(2) **变量sum**：（同程序A ）

(3) **for循环体**：（同程序A）

实际运行结果(2GHz Intel Pentium 4):

程序A：59,393,288 时钟周期

程序B：1,277,877,876 时钟周期

程序A比程序B快21.5 倍!!

Cache和程序性能举例

- 举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i, j, sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。

程序 A：

```
int a[256][256];
……
int sum_array1 ( )
{
        int i, j, sum = 0;
        for ( i = 0; i < 256; i++)
                for (j = 0; j < 256; j++)
                        sum += a[i] [j];
        return sum;
}
```

程序 B：

```
int a[256][256];
……
int sum_array2 ( )
{
        int i, j, sum = 0;
        for ( j = 0; j < 256; j++)
                for ( i = 0; i < 256; i++)
                        sum += a[i] [j];
        return sum;
}
```

（1）不考虑用于一致性和替换的控制位，数据cache的总容量为多少？

（2）a[0][31]和a[1][1]各自所在主存块对应的cache行号分别是多少？

（3）程序A和B的数据访问命中率各是多少？哪个程序的执行时间更短？

Cache和程序性能举例

- 举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i, j, sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。

(1) 主存地址空间大小为256MB，因而主存地址为28位，其中6位为块内地址，3位为cache行号（行索引），标志信息有28-6-3=19位。在不考虑用于cache一致性维护和替换算法的控制位的情况下，数据cache的总容量为：
8×(19+1+64×8)=4256位=532字节 。

(2) a[0][31]的地址为320+4×31=444，[444/64]=6（取整），因此a[0][31]对应的主存块号为6。6 mod 8=6，对应cache行号为6。

或: 444=0000 0000 0000 0000 000 110 111100B，中间3位110为行号（行索引），因此，对应的cache行号为6。 a[1][1]对应的cache行号为：
[(320+4×(1×256+1))/64] mod 8=5。

(3) A中数组访问顺序与存放顺序相同，共访问64K次，占4K个主存块；首地址位于一个主存块开始，故每个主存块总是第一个元素缺失，其他都命中，共缺失4K次，命中率为1-4K/64K=93.75%。

方法二：每个主存块的命中情况一样。对于一个主存块，包含16个元素，需访存16次，其中第一次不命中，因而命中率为15/16=93.75%。

B中访问顺序与存放顺序不同，依次访问的元素分布在相隔256×4=1024的单元处，它们都不在同一个主存块中,cache共8行，一次内循环访问16块，故再次访问同一块时，已被调出cache，因而每次都缺失，命中率为0。

4.4 辅助存储器

一、概述

1. 特点：不直接与 CPU 交换信息

2. 磁表面存储器的技术指标

(1) 记录密度

道密度 D_t：沿磁盘半径方向长度上的磁道数

道密度＝磁道数/存储区域的长度

单位：道/英寸（TPI）；道/毫米（TPM）

位密度 D_b：位密度是磁道单位长度上可以记录的二进制代码数位。

位密度＝磁道容量/内圈的周长

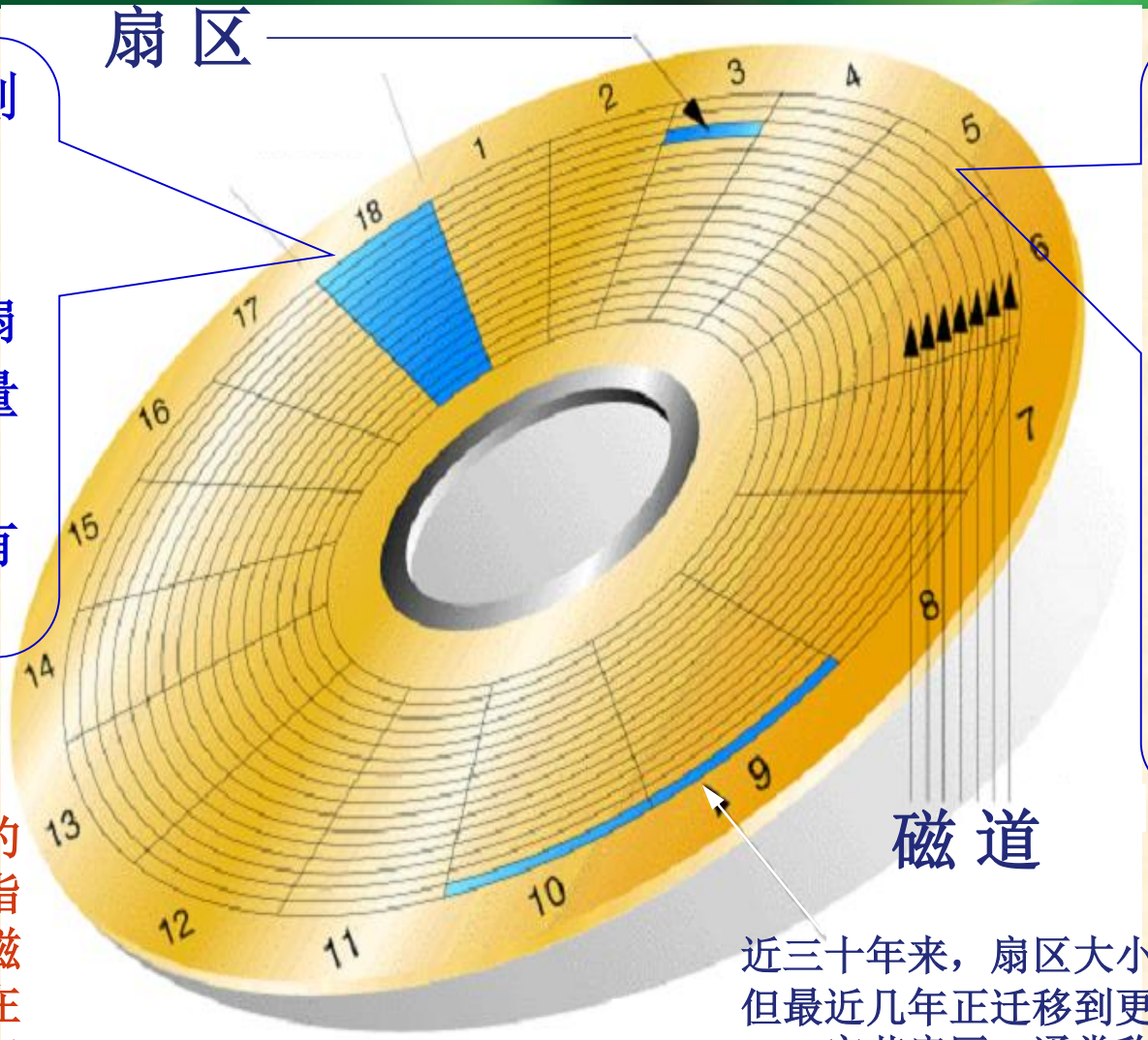
单位：位/英寸（BPI）；位/毫米（BPM）

几个概念

- 记录面：磁盘片表面称为记录面。
- 磁道：记录面上一系列同心圆称为磁道。它的编址是由外向内依次编号，最外一个同心圆叫做零磁道。
- 柱面：所有记录块上半径相等的磁道的集合称为柱面，一个磁盘组的柱面数等于其中一个记录面上的磁道数。
- 扇区：将每一个记录面分成若干个区域，每一个区域称为一个扇区。
- 记录块：将每一个磁道分成若干个段，每段称为一个记录块或扇段。

磁盘的磁道和扇区

每个磁道被划分为若干段（段又叫扇区），每个扇区的存储容量为512字节。每个扇区都有一个编号

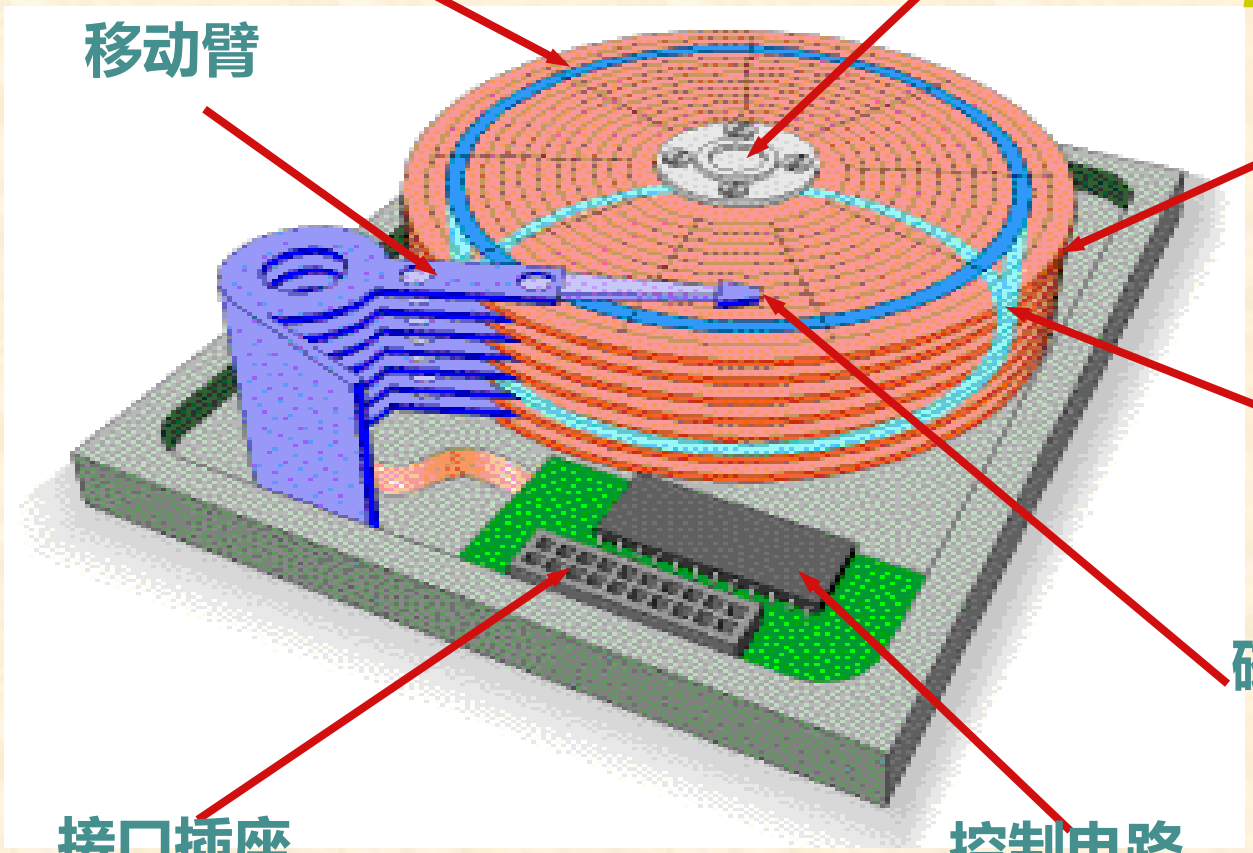


磁盘表面被分为许多同心圆，每个同心圆称为一个磁道。每个磁道都有一个编号，最外面的是0磁道

注：所谓磁盘的格式化操作，指在盘面上划分磁道和扇区，并在扇区中填写扇区号等信息的过程

近三十年来，扇区大小一直是512字节。但最近几年正迁移到更大、更高效的4096字节扇区，通常称为4K扇区。国际硬盘设备与材料协会（IDEMA）将之称为高级格式化。

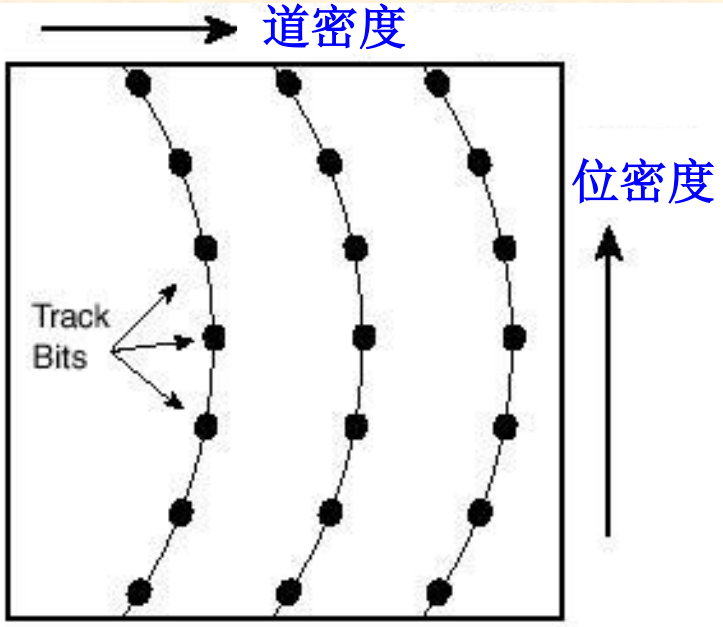
磁盘驱动器



磁道号就是柱面号、磁头号就是盘面号

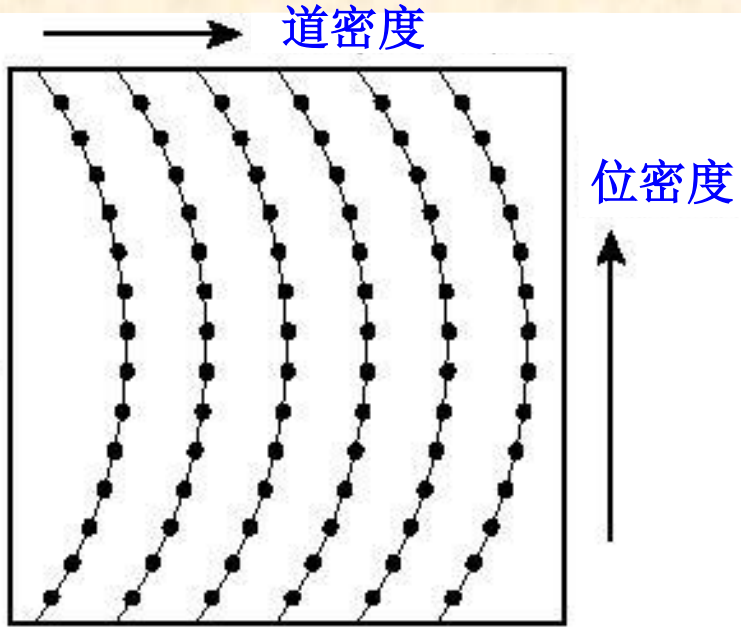
如何增大磁盘片的容量？

- 提高盘片上的信息记录密度！
  - 增加磁道数目——提高磁道密度
  - 增加扇区数目——提高位密度，并采用可变扇区数



低密度存储示意图

早期的磁盘所有磁道上的扇区数相同，所以位数相同，内道上的位密度比外道位密度高



高密度存储示意图

现代磁盘磁道上的位密度相同，所以，外道上的扇区数比内道上扇区数多，使整个磁盘的容量提高

磁盘磁道的格式

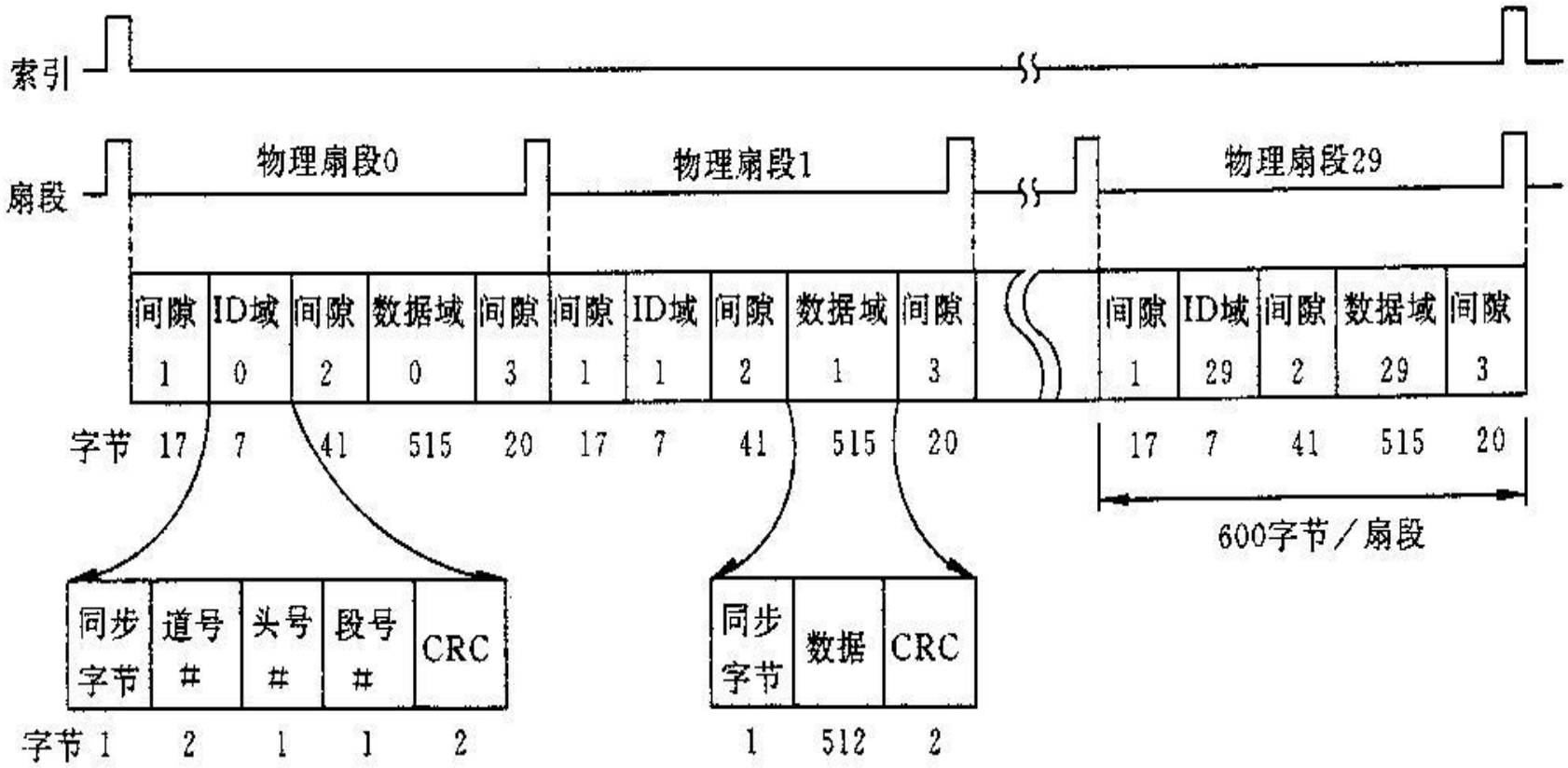


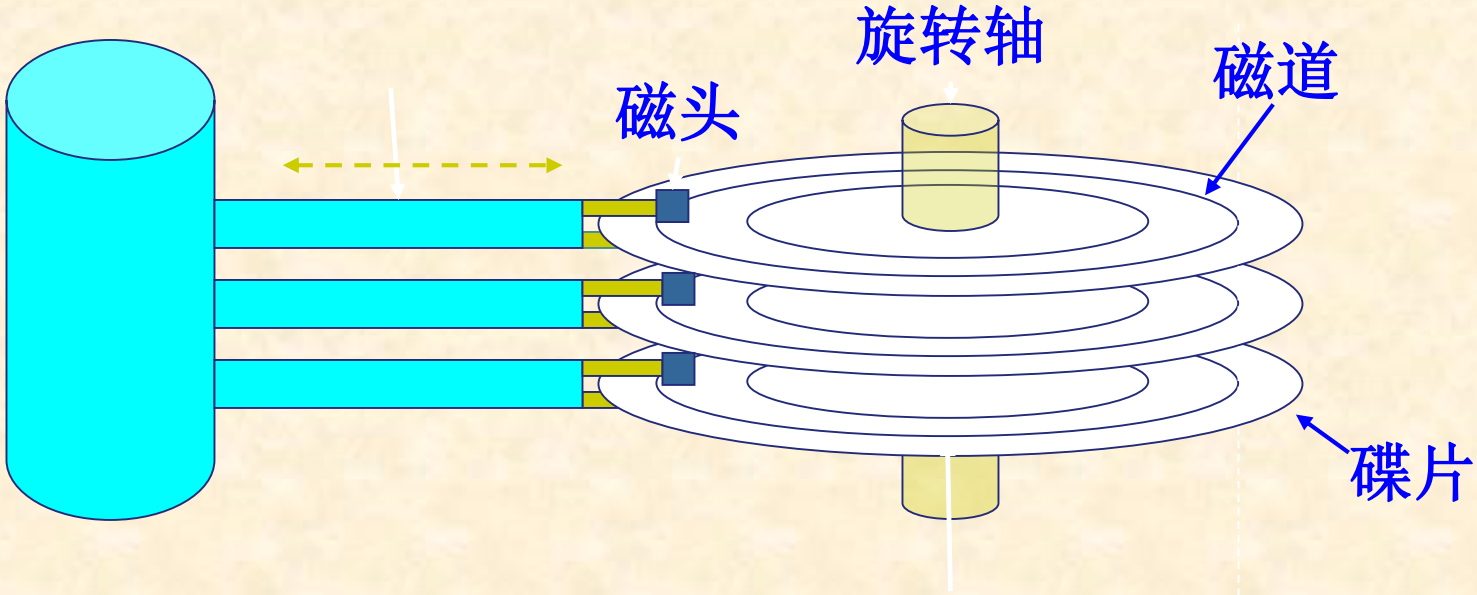
图 5.2 温彻斯特磁盘磁道格式(Seagate ST506)

在此例中，每个磁道包含30个固定长度的扇段，每个扇段有600个字节(17+7+41+515+20=600)。

（2）存储容量：磁盘存储器可以存储的总字节数

- 格式化容量：是可按照某个特定记录格式利用的磁化单元数，也是用户真正可以使用的容量
  - 格式化容量＝记录面数×每面的磁道数×扇区数×记录块的字节数，
  - 单位：KB或MB。
- 非格式化容量：是磁记录表面可以利用的磁化单元总数。
  - 非格式化容量＝记录面数×每面的磁道数×磁道容量，
  - 单位:KB或MB。
- 关系：格式化容量一般为非格式化60%～70%

平均存取时间



硬盘的操作流程如下：

所有磁头同步寻道（由柱面号控制）→ 选择磁头（由磁头号控制）→

被选中的磁头等待扇区到达磁头下方（由扇区号控制）→ 读写该扇区中的数据

2. 磁表面存储器的技术指标

（3）平均寻址时间：从读写命令发出后，磁头从某一起始位置出发移动到新的记录位置，到开始从盘片表面读出或写入信息所需要的时间。

- 寻道时间：将磁头定位到所要求的磁道上所需要的时间
- 等待时间：等待磁道上需要访问的信息到达磁头下的时间
- 平均寻址时间＝平均寻道时间＋平均等待时间。
  - 平均寻道时间＝（最大寻道时间＋最小寻道时间）/2；
  - 平均等待时间＝磁盘旋转一周所需时间的一半，即1/2×（1/转速）。

- 磁盘上的信息以扇区为单位进行读写，平均存取时间为：

  T = 寻道时间 + 旋转等待时间 + 数据传输时间（忽略不计）

  - 寻道时间——磁头寻找到指定磁道所需时间(大约5ms)
  - 旋转等待时间——指定扇区旋转到磁头下方所需要的时间(大约4～6ms) (转速： 4200 / 5400 / 7200 / 10000rpm)
  - 数据传输时间——(大约0.01ms / 扇区)

（4）数据传输率：磁盘存储器在单位时间能向主机传送的字节数。

- $D_r = D_b \times V$
- 如果磁盘的旋转速度为每秒n转，每条磁道的容量为N个字节，则数据传输率

$$Dr＝nN（B/s）$$

(5) 误码率:出错信息位数与读出信息的总位数之比

- CRC循环冗余校验码

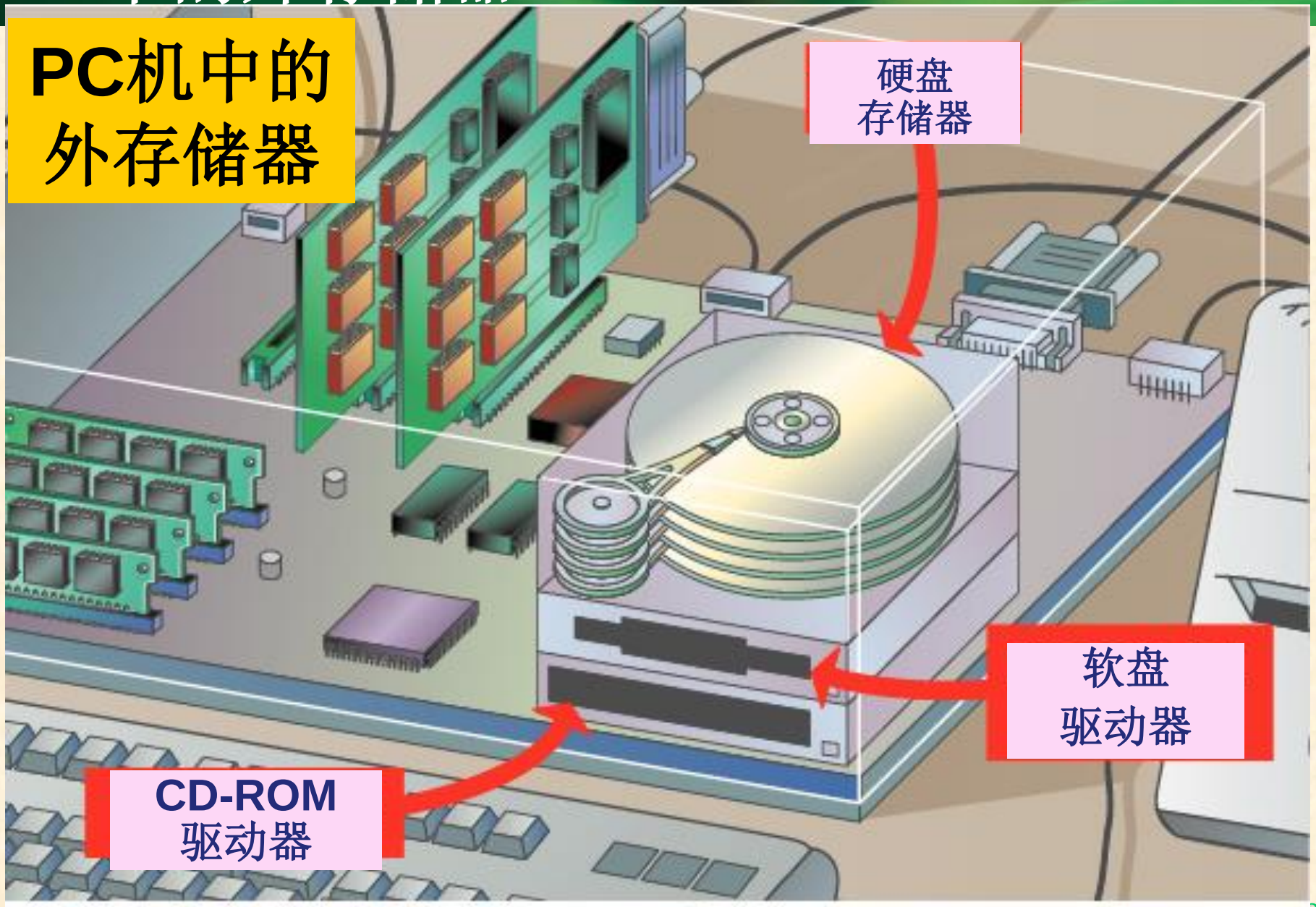
3、记录的地址格式

- 磁盘地址格式：对于活动头磁盘组，磁盘地址由记录面号（也称为磁头号）、磁道号和扇区号组成。一台主机如果配有几台磁盘机，则还要给它们编号，因此，磁盘地址格式为：

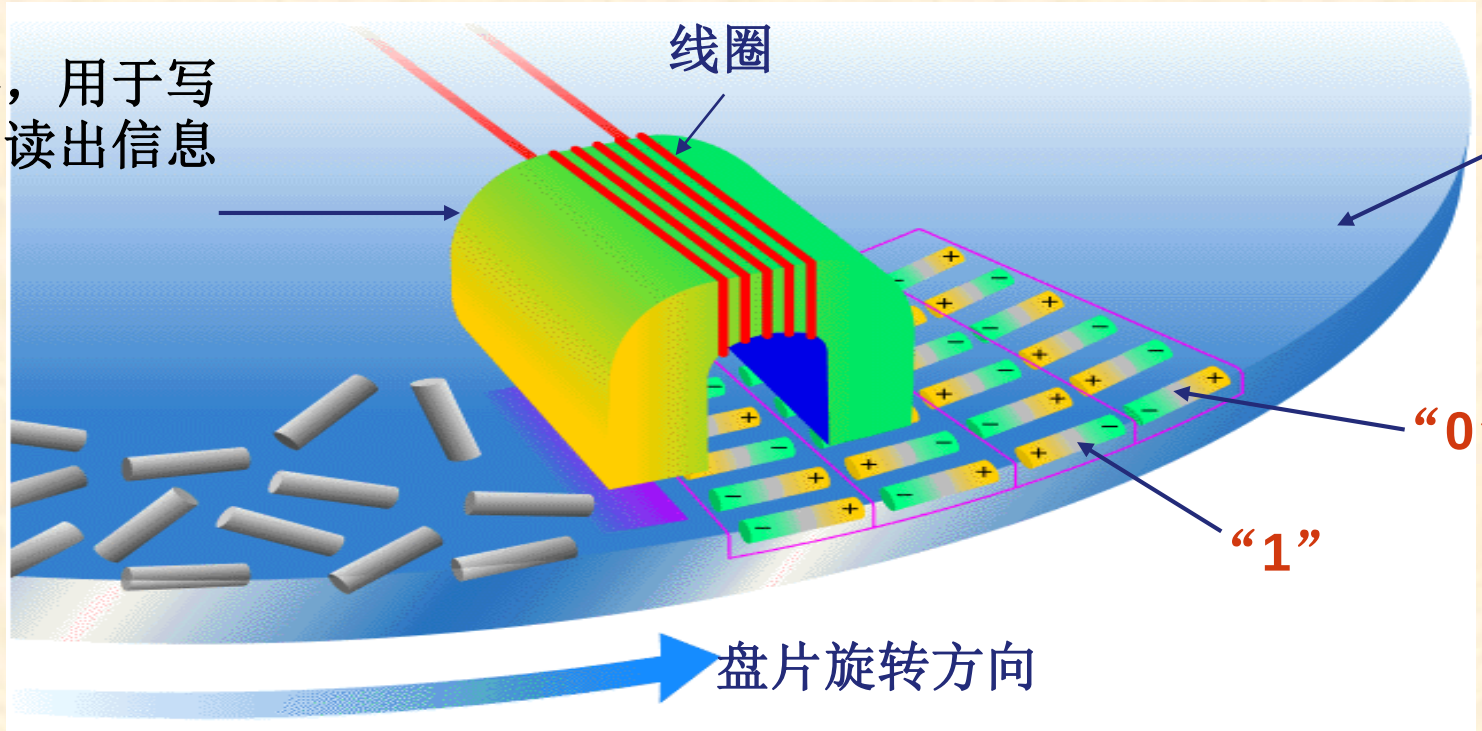
驱动器号	磁道号（柱面号）	记录面号（磁头号）	扇区号

- 注意磁道号（柱面号）与记录面号的顺序。
  - 如果有一个较大的文件，在某磁道、某记录面的所有扇区内存放不下时，应先改变记录面号（即换成与该磁道号对应的另一记录面存放），这样可避勉磁头的机械运动影响存取速度。只有当磁道号（柱面号）对应的所有记录都存放不下时，才改变磁道号（柱面号）。

PC中的外存储器



二、磁记录原理和记录方式



写1：线圈通以正向电流，使呈N-S状态

写0：线圈通以反向电流，使呈S-N状态

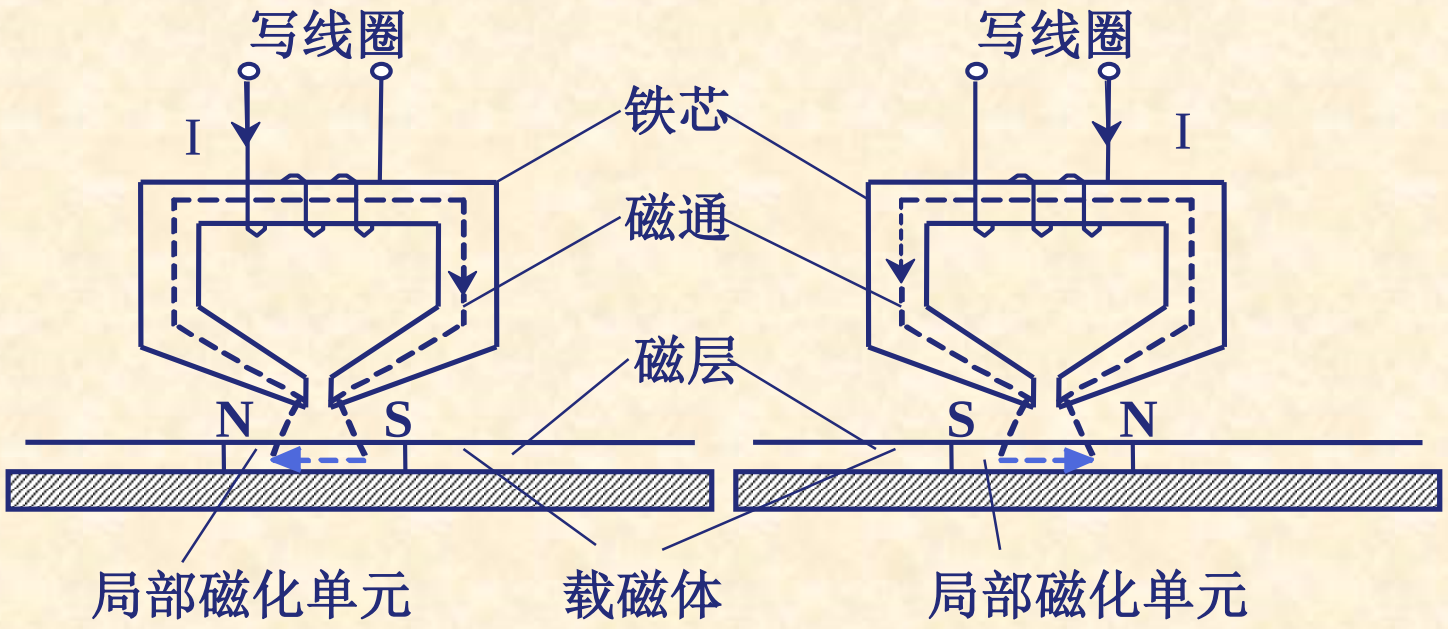
} 不同的磁化状态被记录在磁盘表面

读时：磁头固定不动，载体运动。因为载体上小的磁化单元外部的磁力线通过磁头铁芯形成闭合回路，在铁芯线圈两端得到感应电压。根据不同的极性，可确定读出为0或1。

二、磁记录原理和记录方式

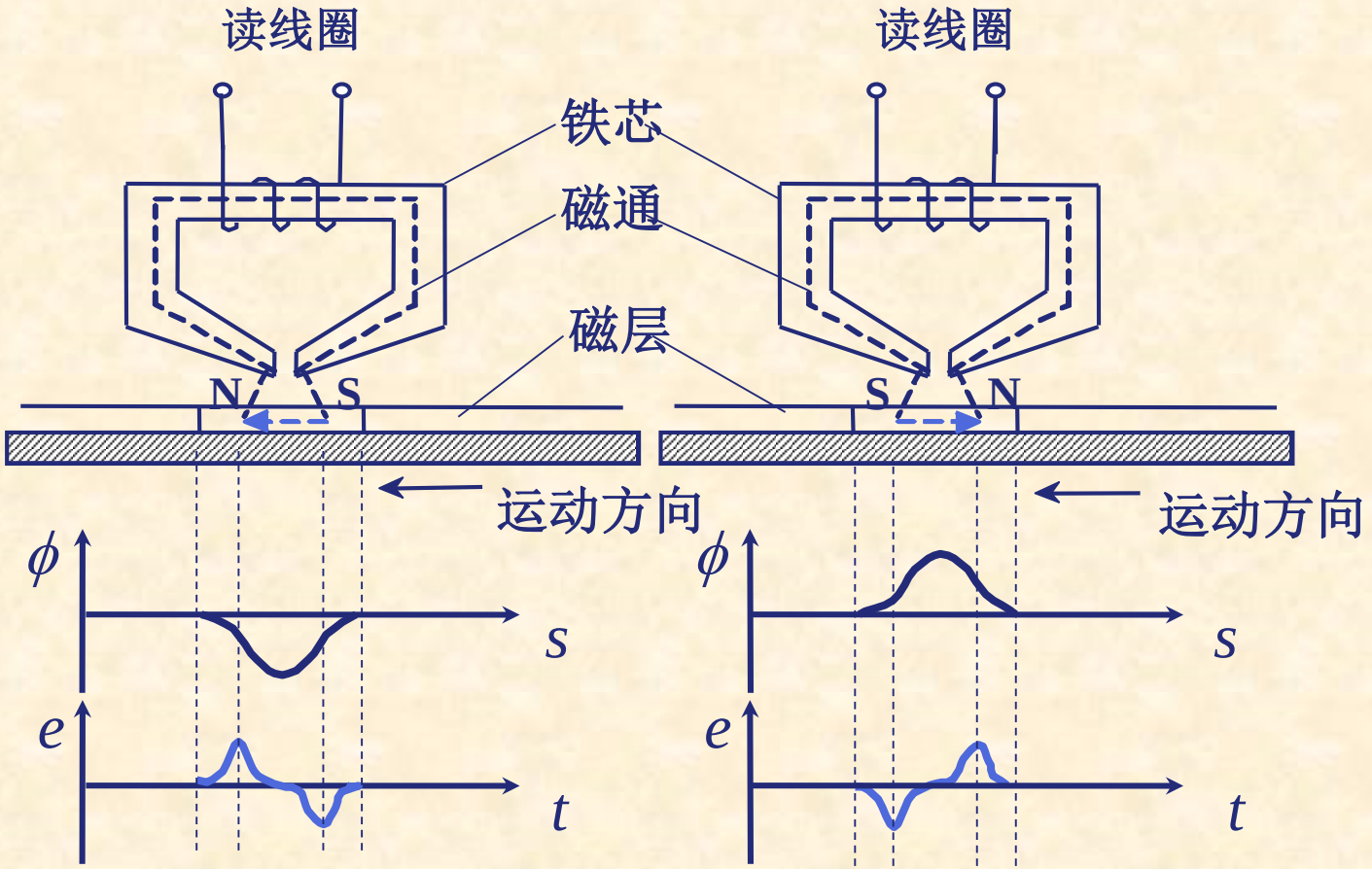
1. 磁记录原理

写



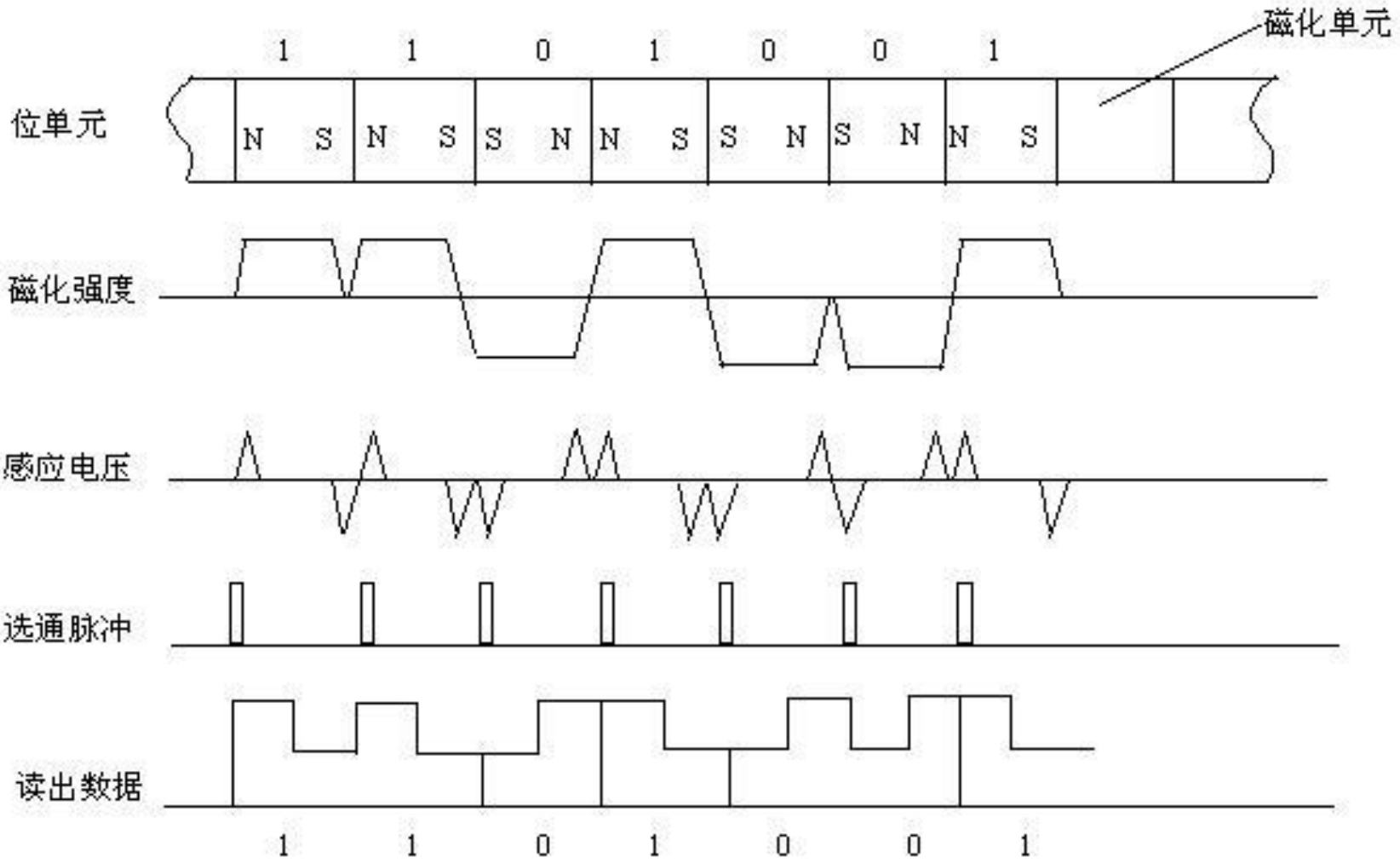
写入“0”　　写入“1”

读



读出“0”　　读出“1”

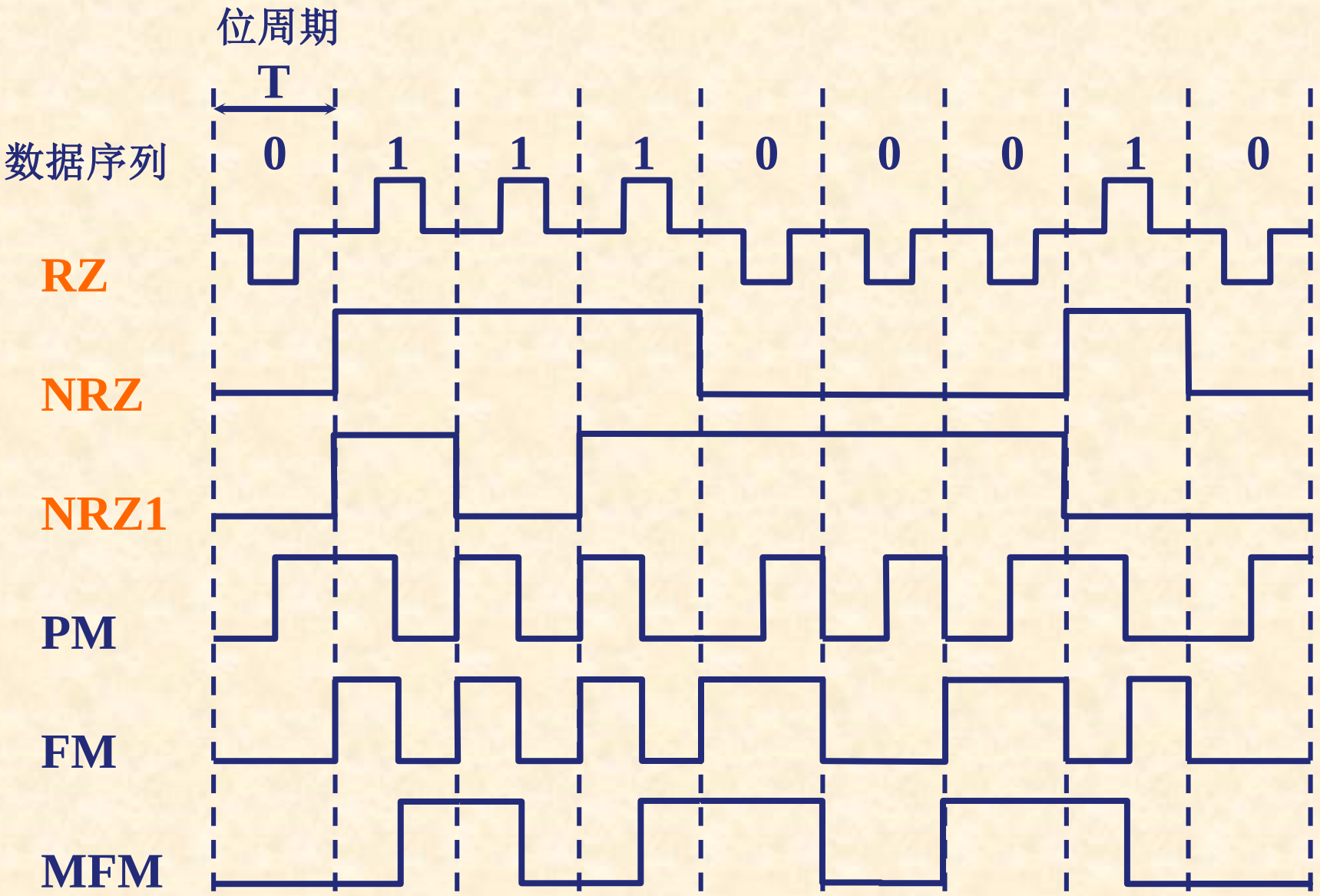
磁表面信息读出过程


磁化单元
1 1 0 1 0 0 1
位单元
N S N S S N N S S N S N N S
磁化强度
感应电压
选通脉冲
读出数据
1 1 0 1 0 0 1

2. 磁表面存储器的记录方式

- 磁记录方式是一种编码方式，即按照某种规律，把待写入的二进制信息转换成磁表面相应的磁化状态。实际就是将二进制信息转换成对应的写电流脉冲序列，写入电流波形的组成方式称为记录方式。
- 归零制（RZ，Return to Zero）记录方式中，写“1”时磁头线圈中加正向脉冲，写“0”时加负向脉冲。由于脉冲电流均要回到零，故称归零制
- 不归零制（NRZ，Non Return to Zero）：磁头线圈中始终有电源。写“1”时有正向电流，写“0”时有负向电流。由于磁头中电流不回到零，故称为不归零制。
- 见“1”就翻的不归零 - （NRZ - 1）这是一种改进的不归零制，记录“1”时，在位周期开始写电流改变方向；而记录“0”时，写电流方向维持不变，所以称之为见“1”就翻的不归零制。

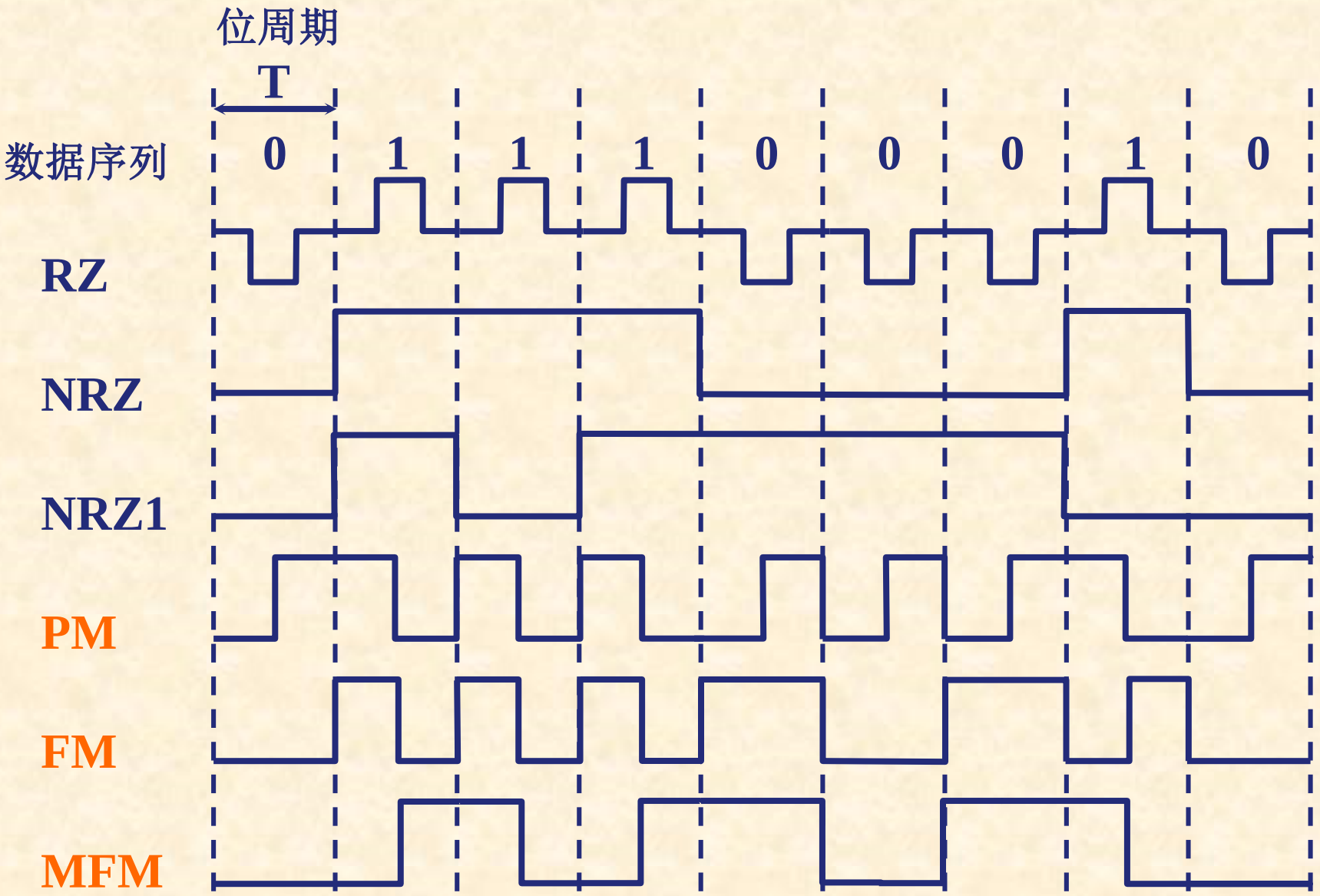
2. 磁表面存储器的记录方式



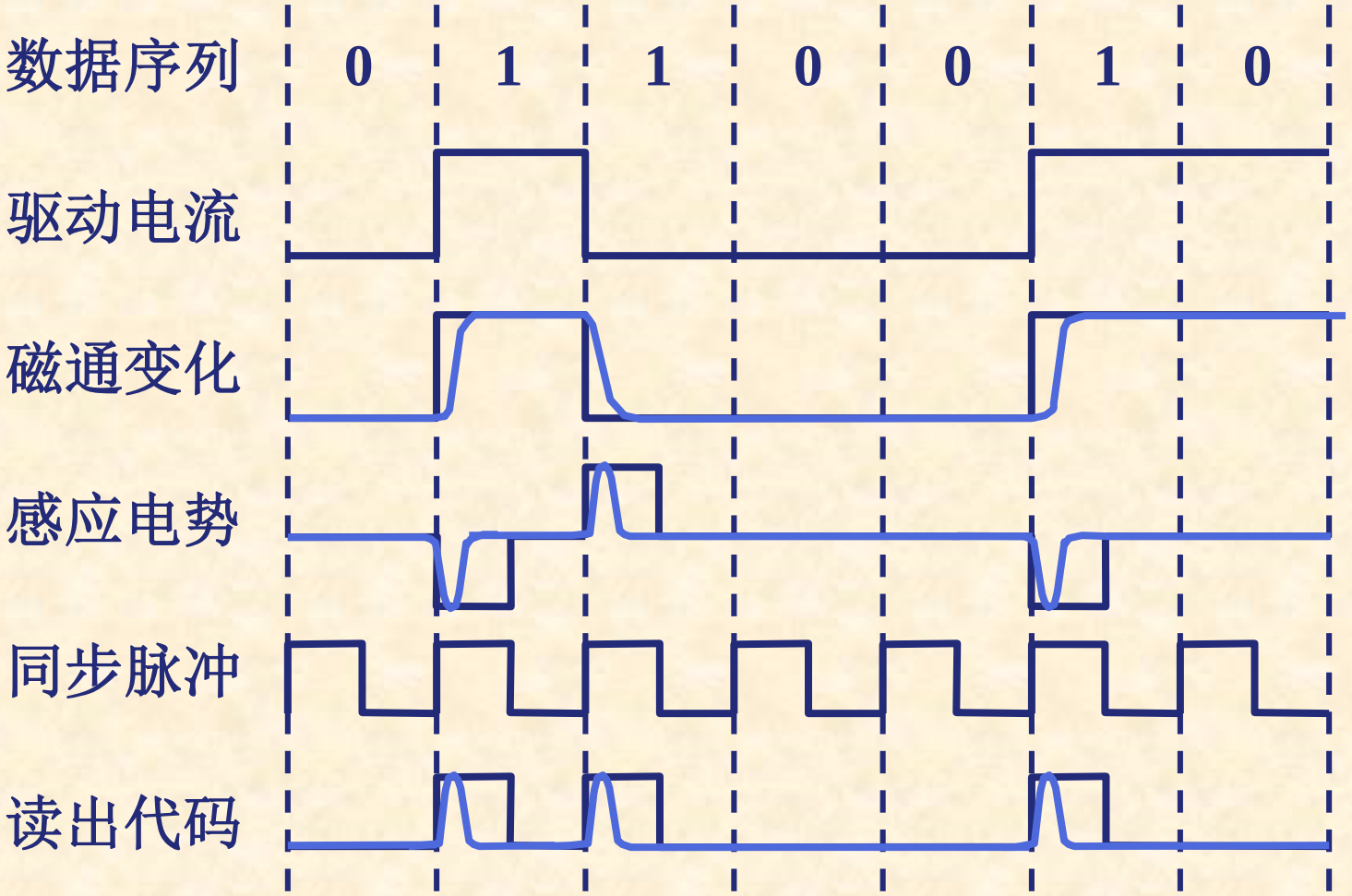
2. 磁表面存储器的记录方式

- 调相制（PE，Phase Encoding）：写“1”时，磁头线圈中的电流先正后负；写“0”时，电流先负后正。
- 调频制（FM，Frequency Modulation）：写“1”时，磁头线圈中的电流在位周期开始时改变一次方向，在位周期中间还要改变一次方向；写“0”时，磁头线圈中的电流只在位周期开始时改变一次方向。（差分曼彻斯特）
- 改进的调频制（MFM） 是对FM制改进，编码规则：记录“1”时，写电流在位周期中间改变方向；记录单独的一个“0”时，写电流不改变方向；记录连续的两个“0”时，写电流在位周期边界改变方向。

2. 磁表面存储器的记录方式



例 NRZ1 的读出代码波形



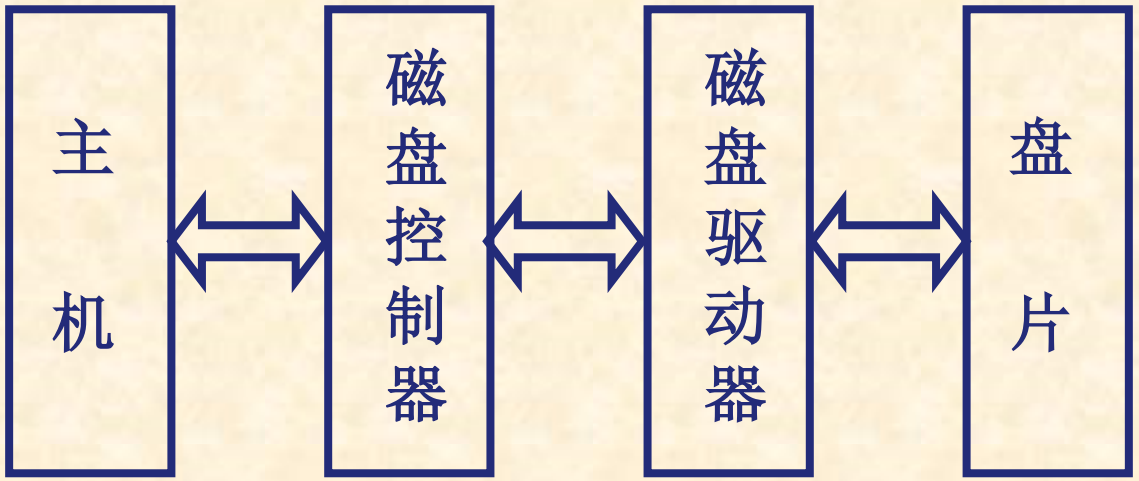
三、硬磁盘存储器

1. 硬磁盘存储器的类型

(1) 固定磁头和移动磁头

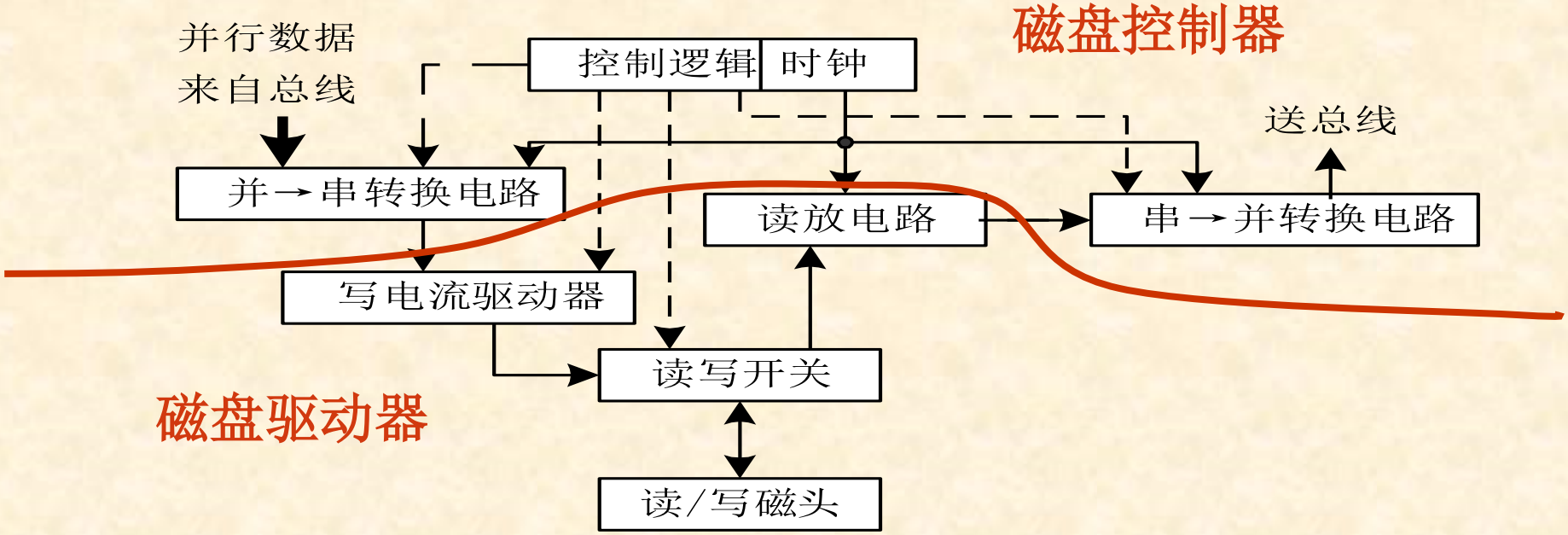
(2) 可换盘和固定盘

2. 硬磁盘存储器结构



硬盘存储器的组成

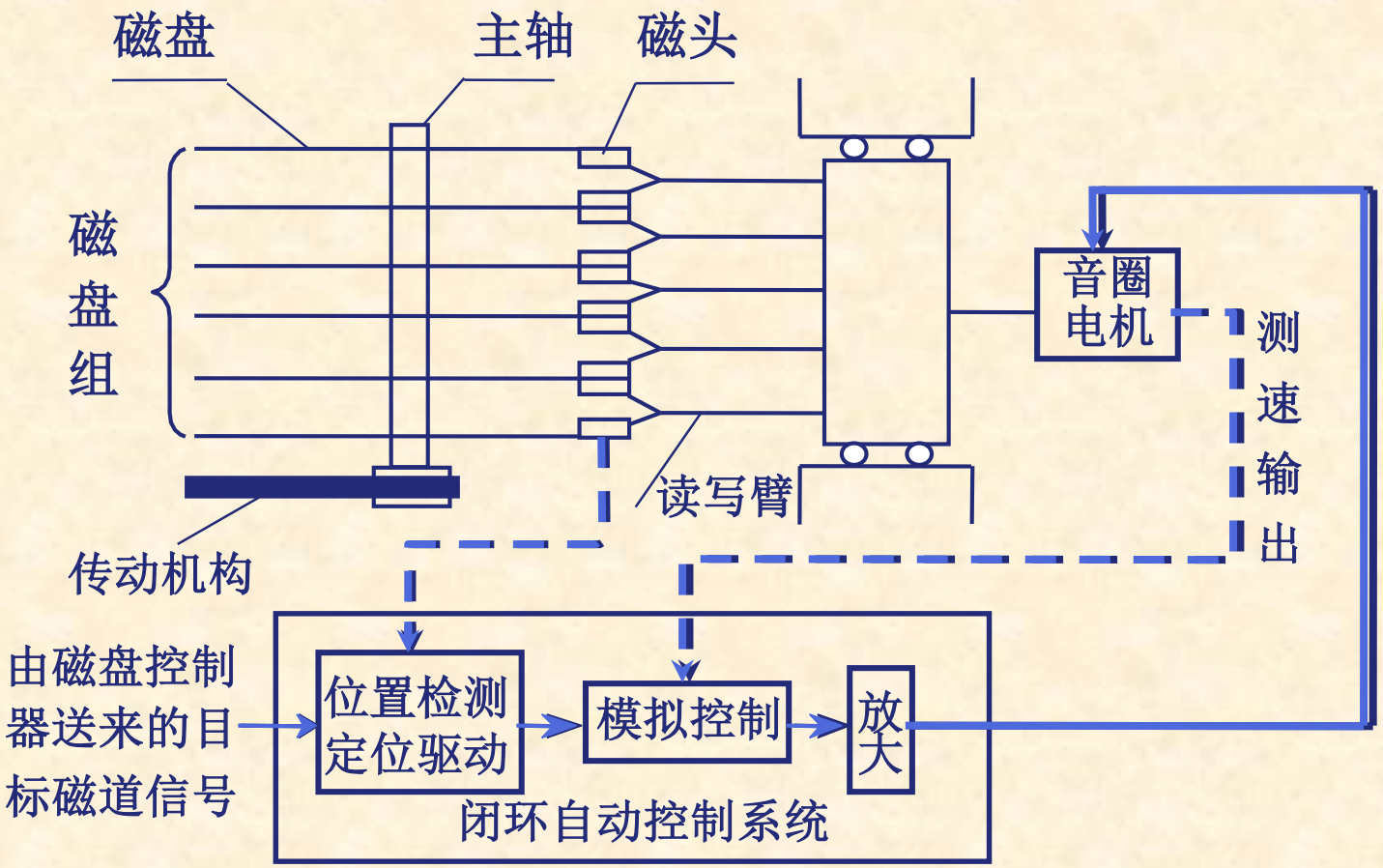
- 磁记录介质：用来保存信息
- 磁盘驱动器：包括读写电路、读/写转换开关、读写磁头与磁头定位伺服系统等
- 磁盘控制器：包括控制逻辑、时序电路、“并→串”转换和“串→并”转换电路等。（用于连接主机与盘驱动器）



硬盘存储器的逻辑结构

(1) 磁盘驱动器

主轴　　定位驱动　　数据控制



(2) 磁盘控制器

磁盘控制器 是

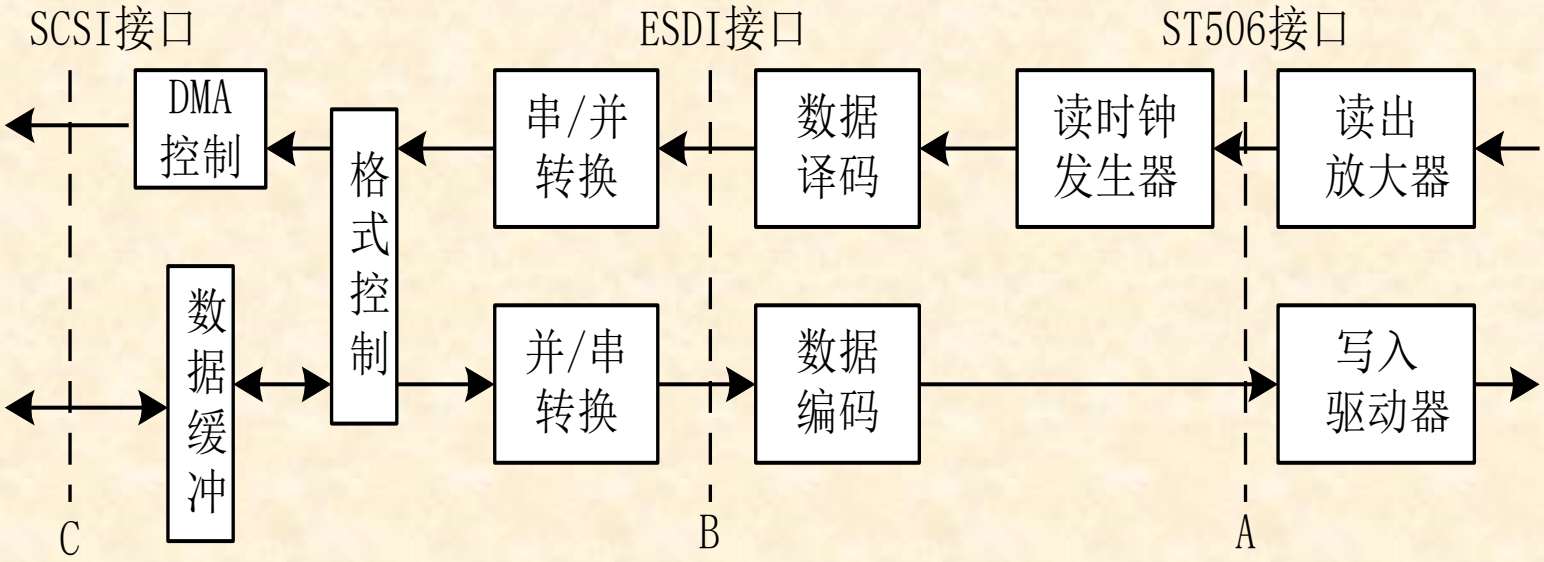
主机与磁盘驱动器之间的 接口 { 对主机 通过总线
对硬盘（设备）

- 接受主机发来的命令，转换成磁盘驱动器的控制命令
- 实现主机和驱动器之间的数据格 式转换
- 控制磁盘驱动器读写

硬盘控制器的逻辑结构

- 磁盘控制器是主机与磁盘驱动器之间的接口
- 磁盘控制器与磁盘驱动器之间并没有明确的界线
  (可以在 A点 / B点 / C点)

通过总线与主机连接　　通过接口电缆与磁盘驱动器连接



服务器大多使用SCSI接口
PC机前几年大多使用IDE(ATA)接口
近两年PC机开始大量使用SATA接口

(3) 盘片

磁层由非矩形剩磁特性的导磁材料（如氧化铁、镍钴合金）构成。导磁材料制成的磁胶涂敷或镀在载磁体上，厚度通常在0.1~5μm。载磁体可以是金属合金（硬质载磁体）或者塑料（软质载磁体）

冗余磁盘阵列(RAID)

- 系统总体性能的提高不匹配
  - 处理器和主存性能改进快
  - 辅存性能性能改进慢

*可靠性(Reliability)

- 所用措施：RAID-Redundant Arrays of Inexpensive Disk （磁盘冗余阵列）

- **RAID的基本思想：**

  将多个独立操作的磁盘按某种方式组织成磁盘阵列(Disk Array)，以增加容量，利用类似于主存中的多体交叉技术，将数据存储在多个盘体上，通过使这些盘并行工作来提高数据传输速度，并用冗余(redundancy)磁盘技术来进行错误恢复(error correction)以提高系统可靠性。

- **RAID特性：**

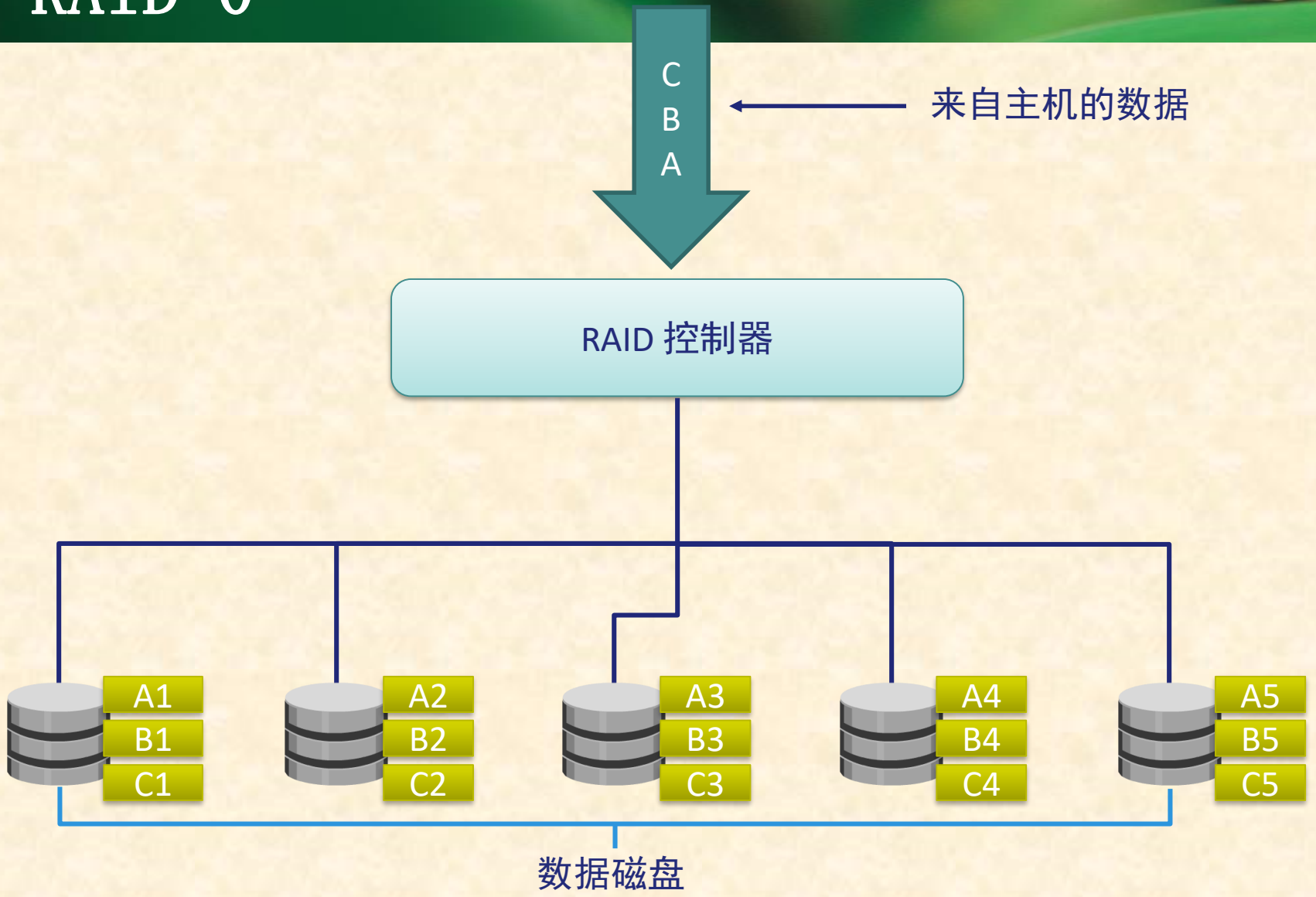
  （1）RAID是一组物理磁盘驱动器，在操作系统下被视为一个单个逻辑驱动器。

  （2）数据分布在一组物理磁盘上。

  （3）冗余磁盘用于存储校验信息，保证磁盘万一损坏时能恢复数据。

- **RAID级别**
  - **目前已知的RAID方案分为8级（0-7级），以及RAID10（结合0和1级）和RAID30（结合0和3级）和 RAID50 （结合0和5级） 。但这些级别不是简单地表示层次关系，而是表示具有上述3个共同特性的不同设计结构。**

RAID 0

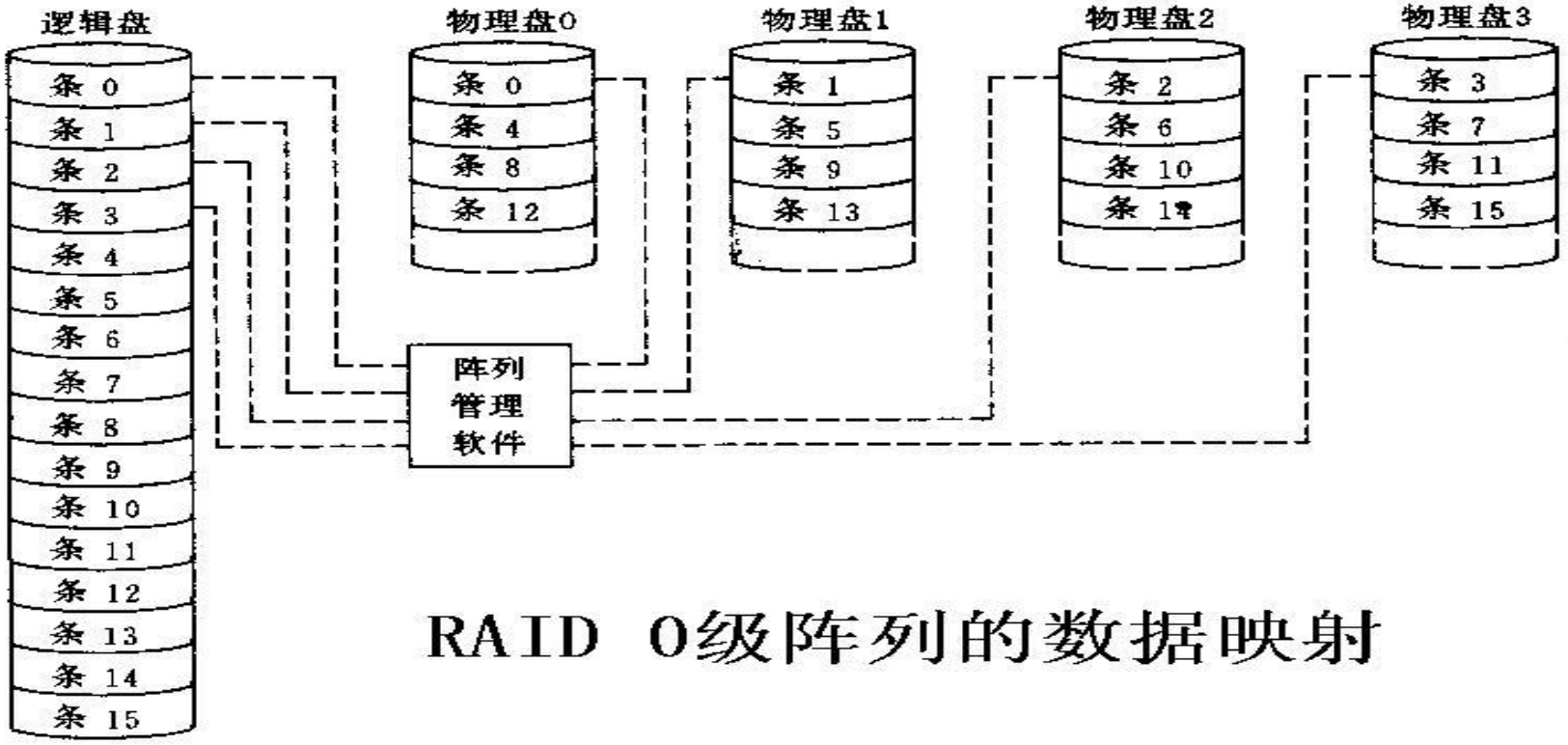

C
B
A
来自主机的数据
RAID 控制器
A1
B1
C1
A2
B2
C2
A3
B3
C3
A4
B4
C4
A5
B5
C5
数据磁盘

冗余磁盘阵列（ RAID 0 ）

- 不遵循特性(3)，所以无冗余。适用于容量和速度要求高的非关键数据存储的场合
  - 与单个大容量磁盘相比有两个优点：

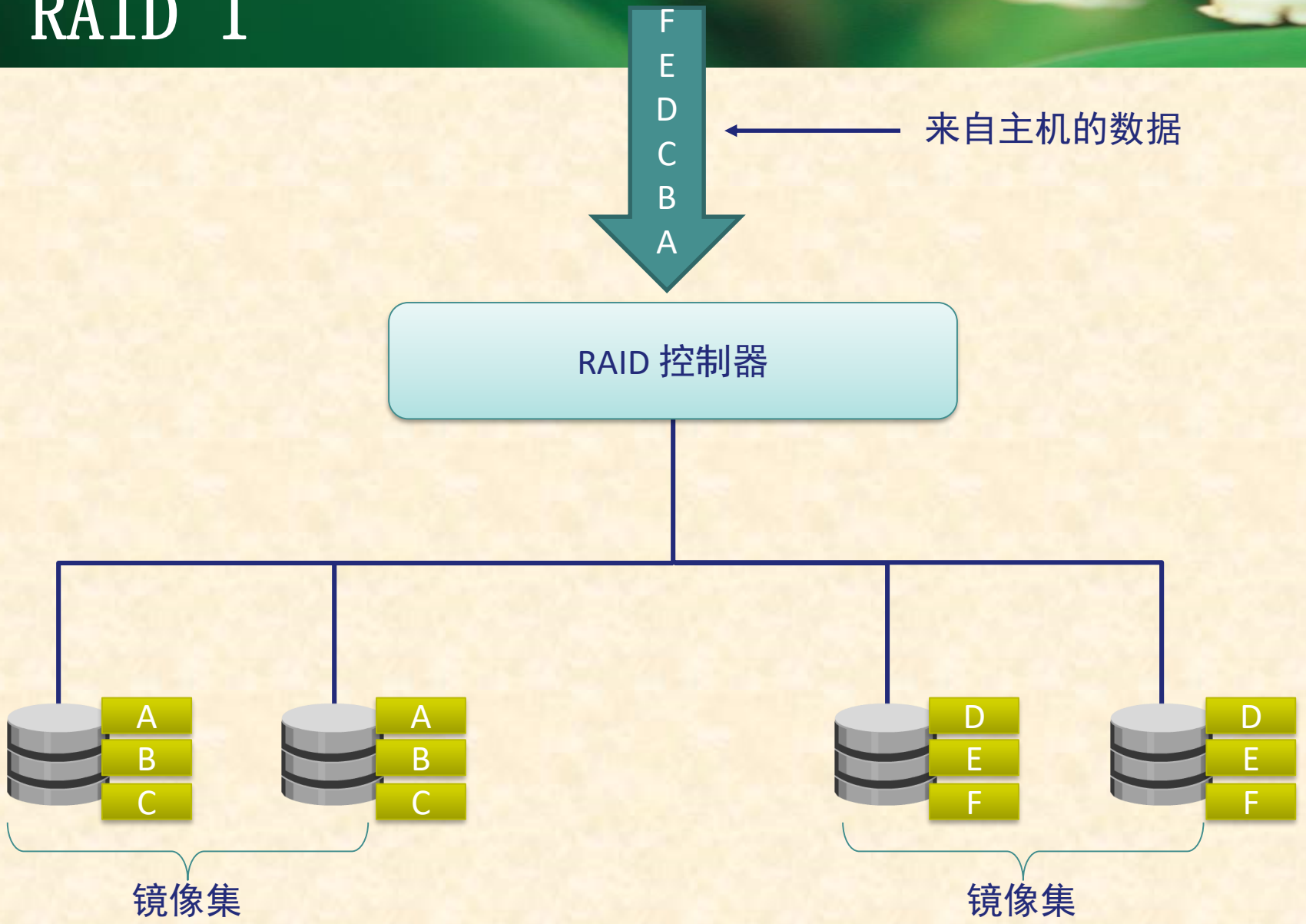
(1) 连续分布或大条区交叉分布时，如果两个I/O请求访问不同盘上的数据，则可并行发送。减少了I/O排队时间。具有较快的I/O响应能力。

(2) 小条区交叉分布时，同一个I/O请求有可能并行传送其不同的数据块(条区)，因而可达较高的数据传输率。例如，可以用在视频编辑和播放系统中，以快速传输视频流



RAID 0级阵列的数据映射

RAID 1



冗余磁盘阵列（ RAID 1 ）

- 镜像盘实现1对1冗余(100% redundancy)

（1）读：一个读请求可由其中一个定位时间更少的磁盘提供数据。

（2）写：一个写请求对应的两个磁盘并行更新。故写性能由两次中较慢的一次写来决定，即定位时间更长的那一次。

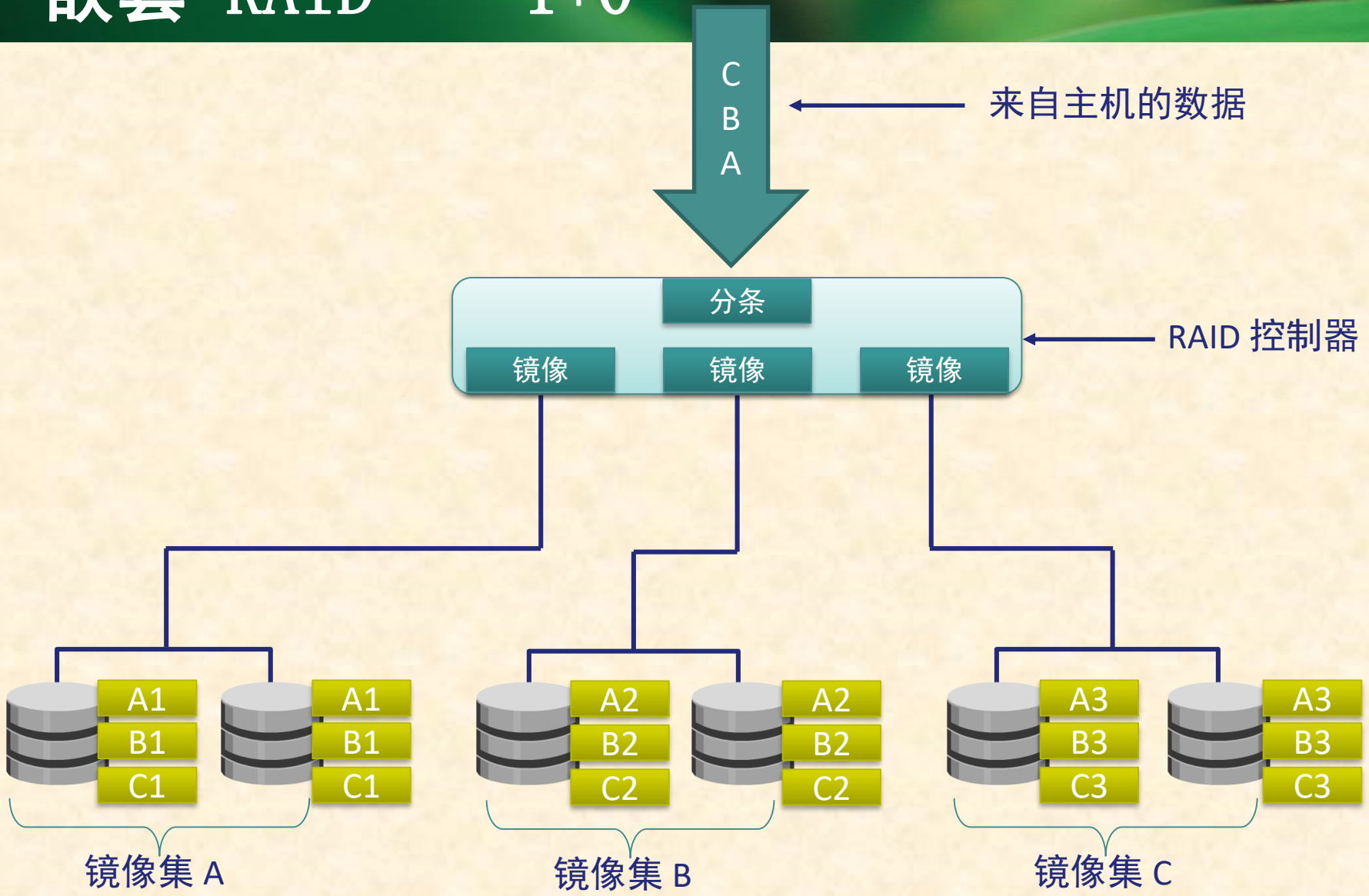
（3）检错：数据恢复很简单。当一个磁盘损坏时，数据仍能从另一个磁盘中读取。

- 特点；可靠性高，但价格昂贵。

常用于可靠性要求很高的场合，如系统软件的存储，金融、证券等系统。

(b) RAID 1(镜像)

嵌套 RAID - 1+0

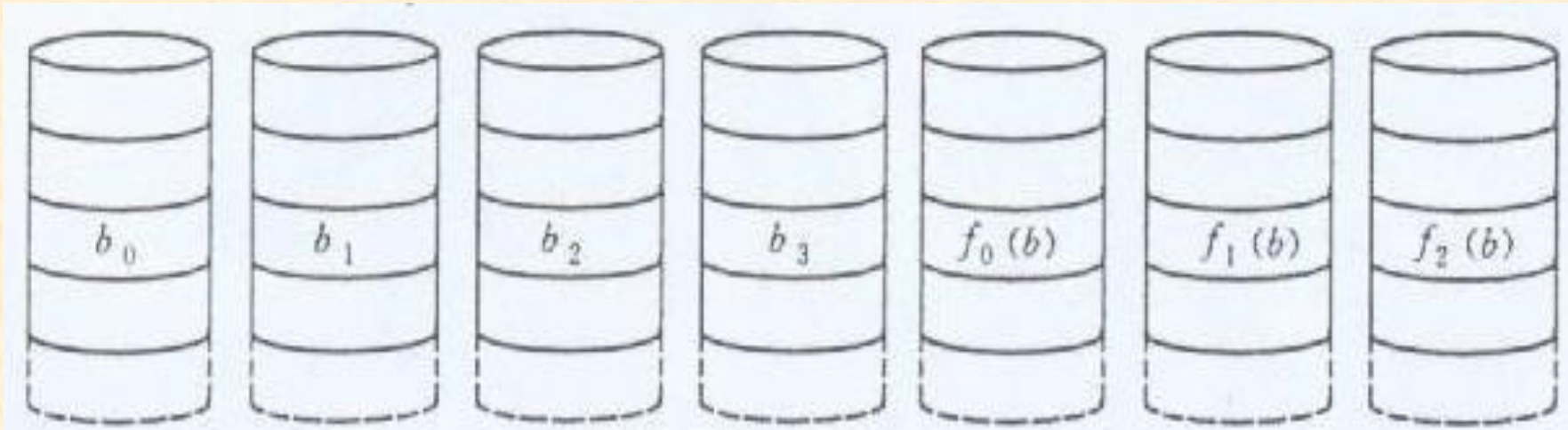

C
B
A
来自主机的数据
分条
镜像
镜像
镜像
RAID 控制器
A1
B1
C1
A1
B1
C1
A2
B2
C2
A2
B2
C2
A3
B3
C3
A3
B3
C3
镜像集 A
镜像集 B
镜像集 C

冗余磁盘阵列（ RAID2 ）

- 用海明校验法生成多个冗余校验盘，实现纠正一位错误、检测两位错误的功能。
- 采用条区交叉分布方式，且条区非常小（有时为一个字或一个字节）。这样，可获得较高的数据传输率，但I/O响应时间差。
- 采用海明码，虽然冗余盘的个数比RAID1少，但校验盘与数据盘成正比。所以冗余信息开销还是太大，价格也较贵
- 读操作性能高（多盘并行）。
- 写操作时要同时写数据盘和校验盘。

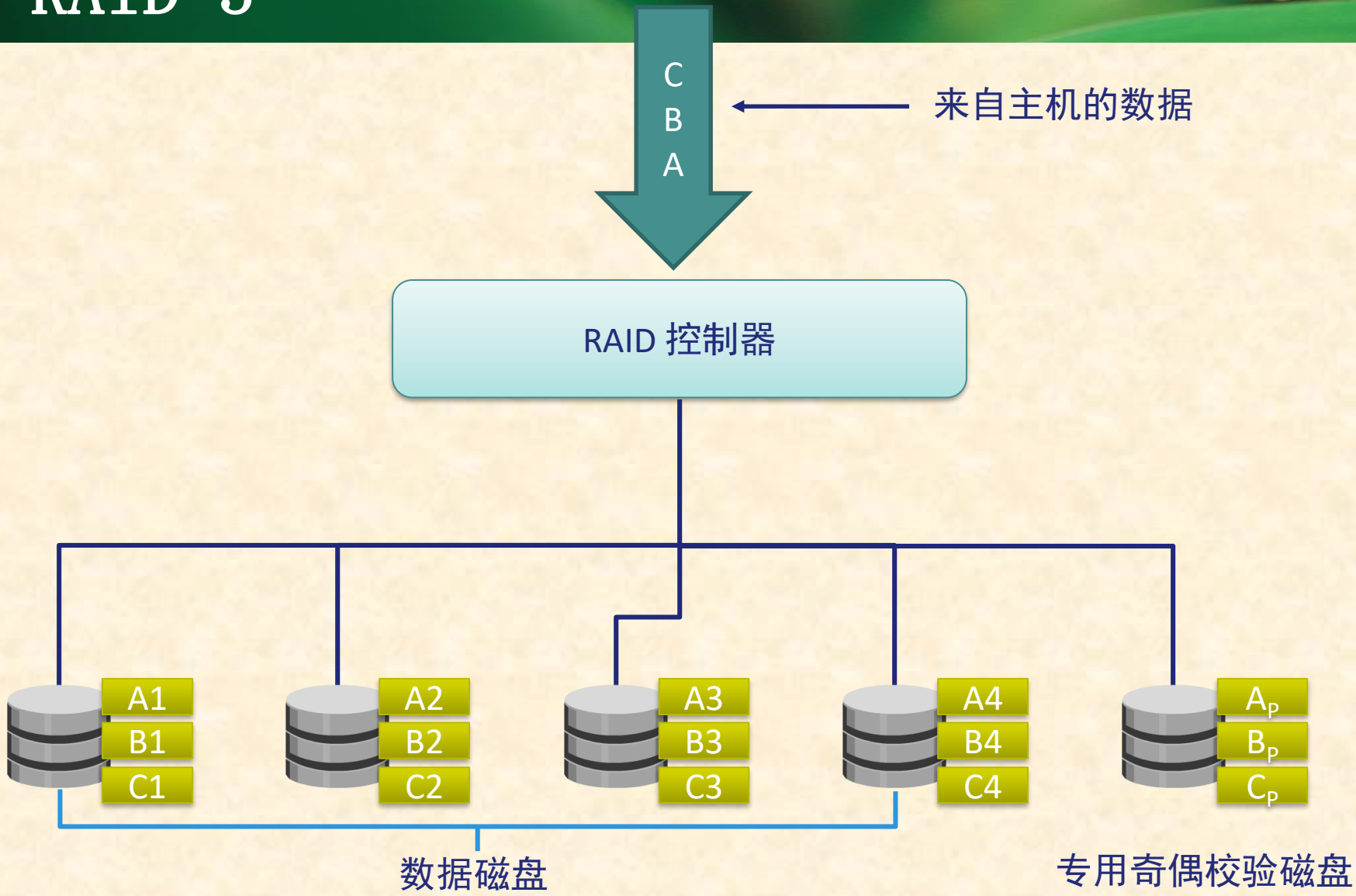
RAID2已不再使用！

为什么？



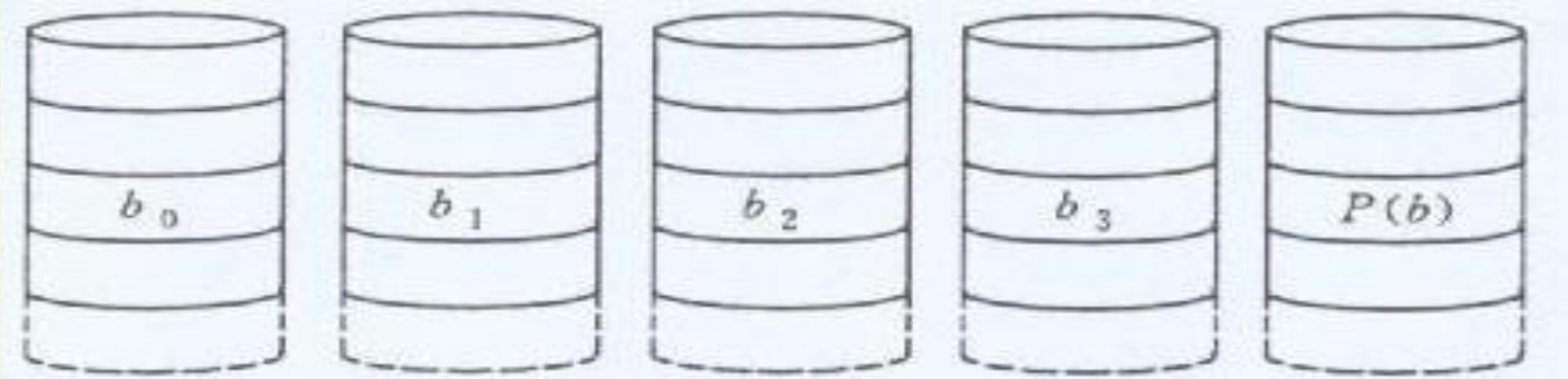
(c) RAID 2(冗余用于海明码)

RAID 3


C
B
A
来自主机的数据
RAID 控制器
A1
B1
C1
A2
B2
C2
A3
B3
C3
A4
B4
C4
A_P
B_P
C_P
数据磁盘
专用奇偶校验磁盘

冗余磁盘阵列（ RAID 3 ）

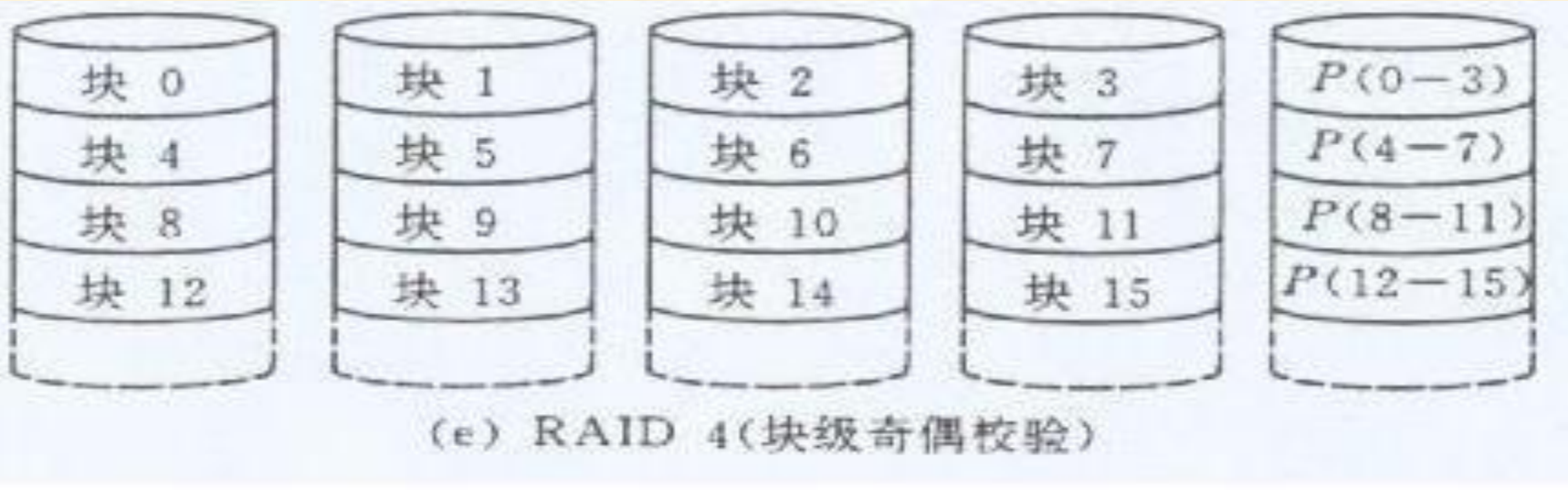
- 采用奇偶校验法生成单个冗余盘。
- 与RAID 2相同，也采用条区交叉分布方式，并使用小条区。这样，可获得较高的数据传输率，但I/O响应时间差。为什么？
- 用于大容量的 I/O请求的场合，如：图像处理、CAD 系统中。
- 某个磁盘损坏但数据仍有效的情况，称为简化模式。此时损坏的磁盘数据可以通过其它磁盘重新生成。数据重新生成非常简单，这种数据恢复方式同时适用于RAID3、4、5级。



(d) RAID 3(位交错奇偶校验)

冗余磁盘阵列（ RAID 4 ）

- 用一个冗余盘存放相应块（较大的数据条区）的奇偶校验位。
- 采用独立存取技术，每个磁盘的操作独立进行，所以可同时响应多个I/O请求。因而它适合于要求I/O响应速度块的场合。
- 对于写操作，校验盘成为I/O瓶颈，因为每次写都要对校验盘进行。
  - 少量写（只涉及个别磁盘）时，有“写损失”，因为一次写操作包含两次读和两次写
  - 大量写（涉及所有磁盘的数据条区）时，则只需直接写入奇偶校验盘和数据盘。因为奇偶校验位可全部用新数据计算得到。而无须读原数据



(e) RAID 4(块级奇偶校验)

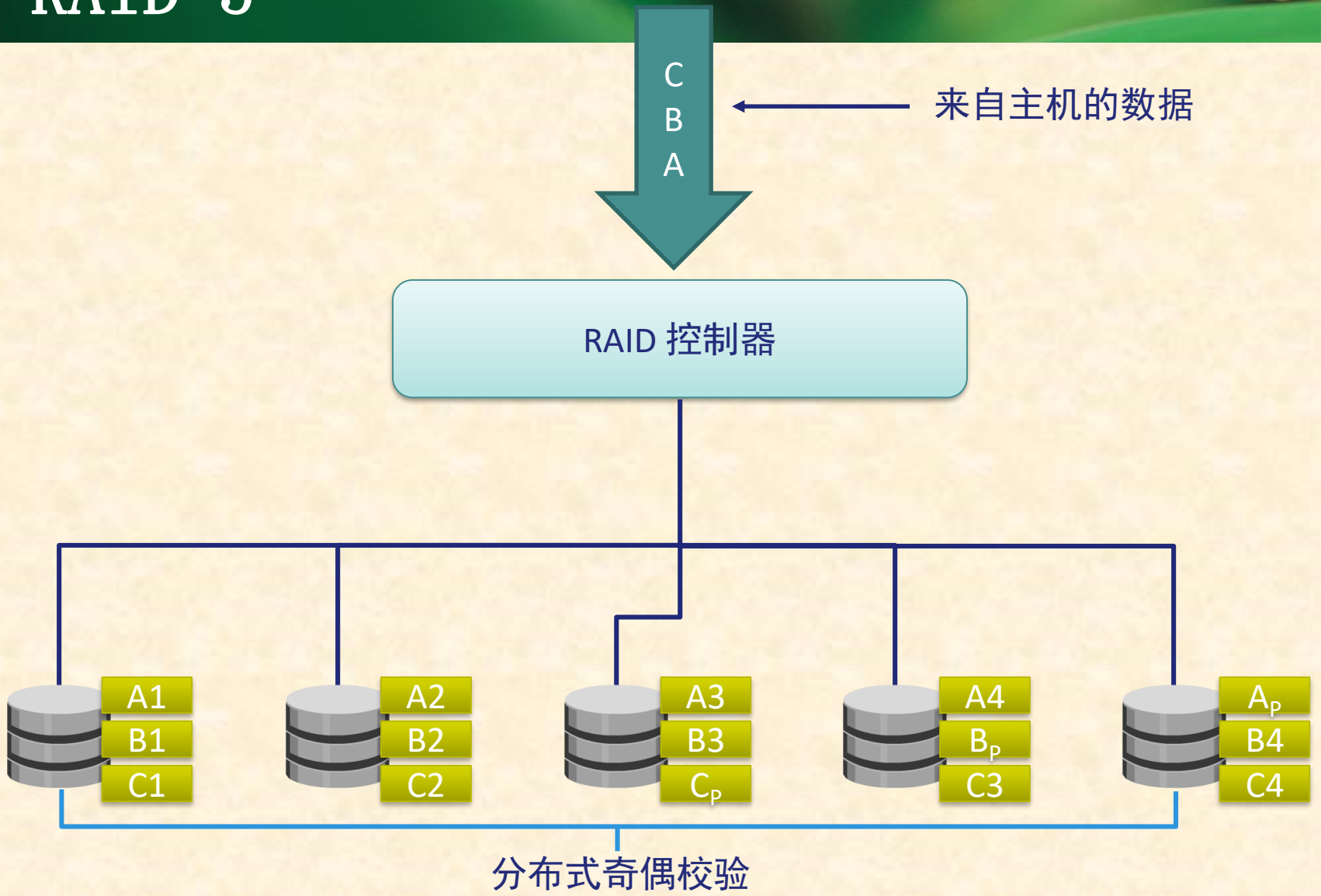
冗余磁盘阵列（ RAID 5 ）

- 与RAID 4组织方式类似，只是奇偶校验块分布在各个磁盘中，所以所有磁盘地位等价，这样可提高容错性，并且避免了使用专门校验盘时潜在的I/O瓶颈。
- 与RAID 4一样，采用独立的存取技术，因而有较高的I/O响应速度。
- 小数据量的操作可以多个磁盘并行操作
- 成本不高但效率高，所以被广泛使用

(f) RAID 5(块级分布式奇偶校验)

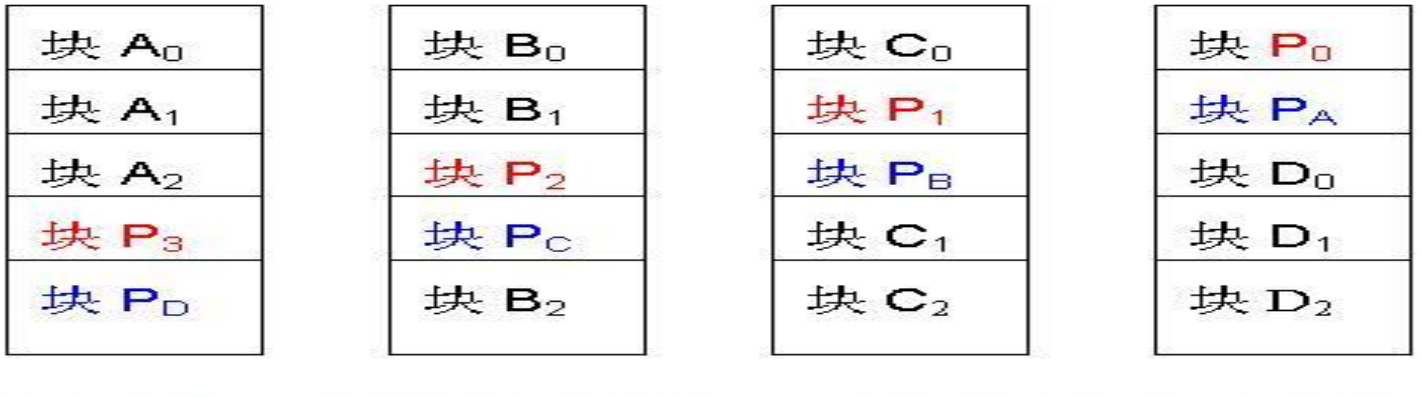
P块为校验块，分布在不同的磁盘中

RAID 5


C
B
A
来自主机的数据
RAID 控制器
A1
B1
C1
A2
B2
C2
A3
B3
C_P
A4
B_P
C3
A_P
B4
C4
分布式奇偶校验

冗余磁盘阵列（ RAID 6 ）

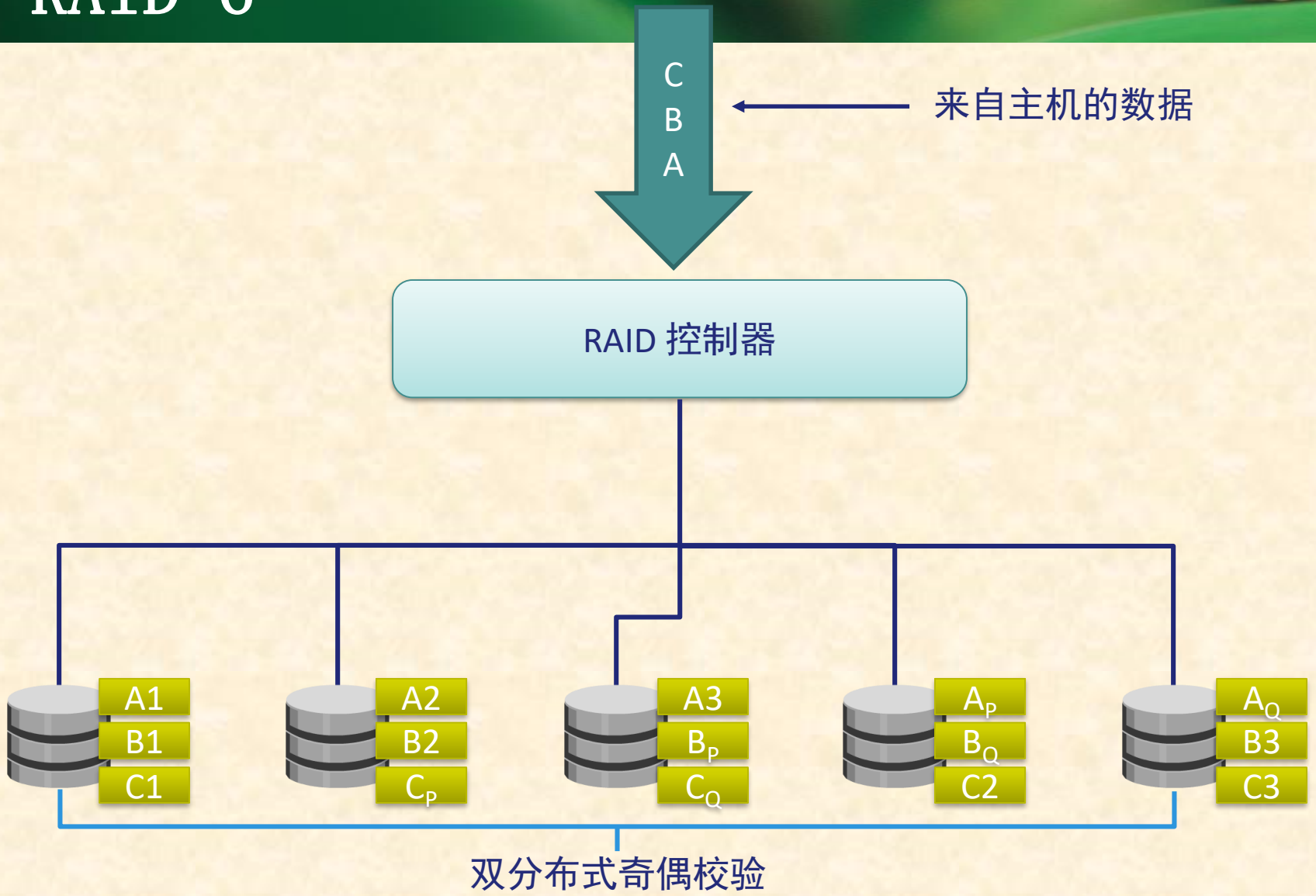
- 冗余信息均匀分布在所有磁盘上，而数据仍以块交叉方式存放
- 双维块交叉奇偶校验独立存取盘阵列，容许双盘出错
- 它是对RAID 5的扩展，主要是用于要求数据绝对不能出错的场合
- 由于引入了第二种奇偶校验值，对控制器的设计变得十分复杂，写入速度也比较慢，用于计算奇偶校验值和验证数据正确性所花费的时间比较多
- RAID 6级以增大开销的代价保证了高度可靠性



RAID 6 级双维块交叉奇偶校验独立存取盘阵列示意图

P0代表第0条区的奇偶校验值，而PA代表数据块A的奇偶校验值

RAID 6


C
B
A
来自主机的数据
RAID 控制器
A1
B1
C1
A2
B2
C_P
A3
B_P
C_Q
A_P
B_Q
C2
A_Q
B3
C3
双分布式奇偶校验

冗余磁盘阵列（ RAID 7 ）

- 带Cache的盘阵列
- 在RAID6的基础上，采用Cache技术使传输率和响应速度都有较大提高
- Cache分块大小和磁盘阵列中数据分块大小相同，一 一对应
- 有两个独立的Cache，双工运行。在写入时将数据同时分别写入两个独立的Cache，这样即使其中有一个Cache出故障，数据也不会丢失
- 写入磁盘阵列以前，先写入Cache中。同一磁道的信息在一次操作中完成
- 读出时，先从Cache中读出，Cache中没有要读的信息时，才从RAID中读

Cache和RAID技术结合，弥补了RAID的不足（如：分块的写请求响应性能差等），从而以高效、快速、大容量、高可靠性，以及灵活方便的存储系统提供给用户

四、软磁盘存储器

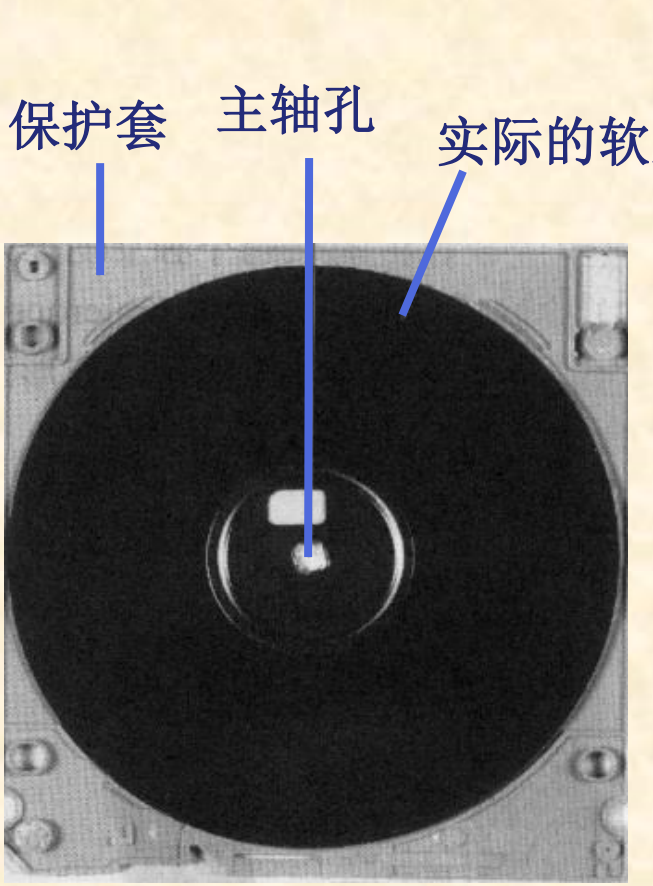
4.4

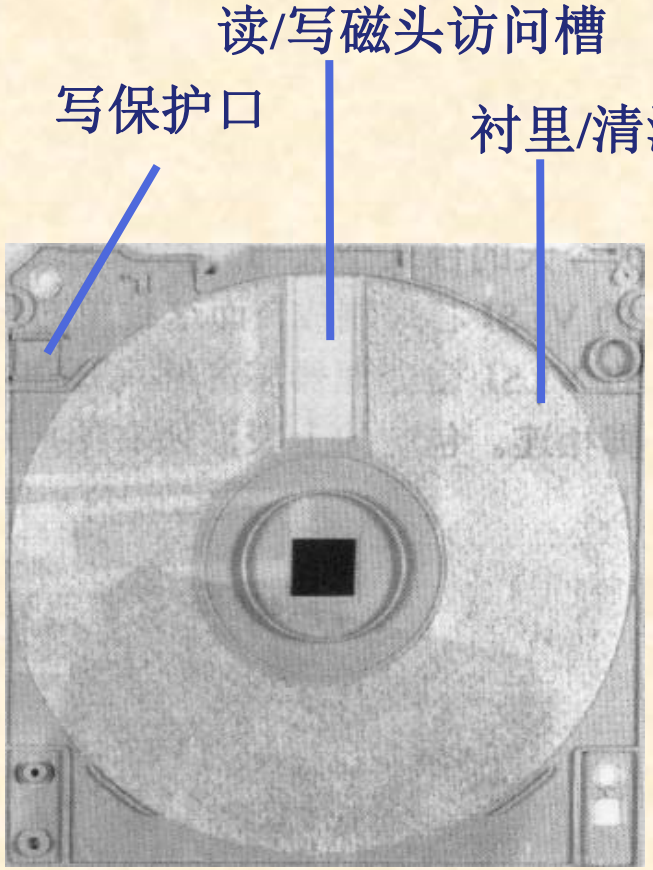
1. 概述

	硬盘	软盘
速度	高	低
磁头	固定、活动	活动
	浮动	接触盘片
盘片	固定盘、盘组 大部分不可换	可换盘片
价格	高	低
环境	苛刻	

2. 软盘片

由聚酯薄膜制成





五、光盘

4.4

1. 概述

采用光存储技术　　利用激光写入和读出

- 第一代光存储技术　　采用非磁性介质　　不可擦写
- 第二代光存储技术　　采用磁性介质　　可擦写

2. 光盘的存储原理

只读型和只写一次型　热作用（物理或化学变化）

可擦写光盘　　热磁效应

循环冗余码

循环冗余校验码（Cyclic Redundancy Check），简称CRC码

- 具很强的检错、纠错能力。
- 用于大批量数据存储和传送中的数据校验。是目前磁表面存储器中应用最广泛的一种校验方法，也是多机通信中常用的校验方法
- 可以发现并纠正信息串行读写、存储或传送过程中出现的一位或多位错误。
- 为什么大批量数据不用奇偶校验?

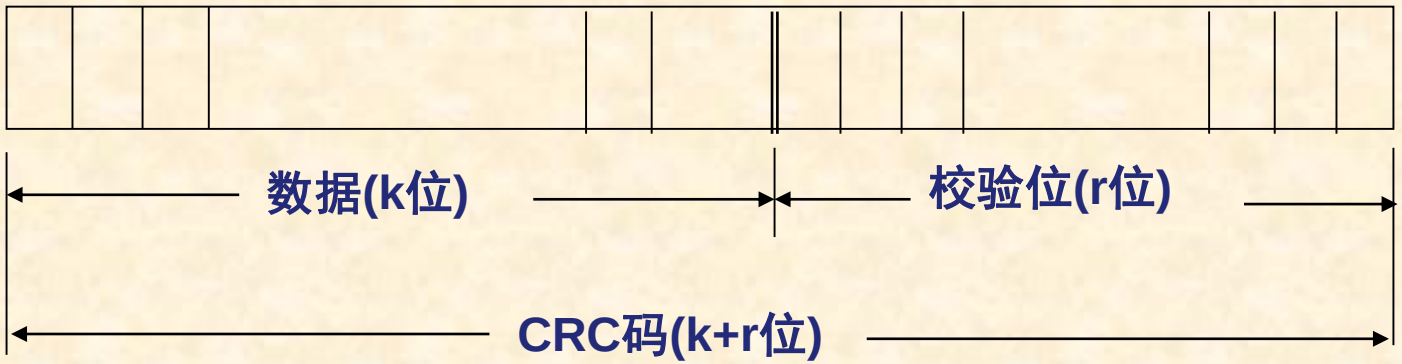
  在每个字符后增加一位校验位会增加大量的额外开销；尤其在网络通信中，对传输的二进制比特流没有必要再分解成一个个字符，因而无法采用奇偶校验码。
- 通过某种数学运算来建立数据和校验位之间的约定关系。

  奇偶校验码和海明校验码都是以奇偶检测为手段的。

CRC码的编码方法

基本思想：

- 数据信息M(x)为一个K位的二进制数据，将M(x)左移r位后，用一个约定的“生成多项式”G(x)相除，G(x)是一个r+1位的二进制数，相除后得到的r位余数就是校验位。校验位拼接到M(x)后，形成一个K+r位的代码，称该代码为循环冗余校验（CRC）码，也称（k+r,k）码。



CRC码的编码方法

- 从k位信息位得到r位校验位的方法：

(1) 将待编码k位有效信息写成多项式M(x)

$$M(x)=C_{k-1}x^{k-1}+C_{k-2}x^{k-2}+\ldots+C_1x+C_0$$

(2) 将M(x)左移r位，得到$M(x)\cdot x^r$。目的是空出r位，以便拼接r位校验位

(3) 选取一个r+1位的生成多项式G(x)。对$M(x)\cdot x^r$做模2除运算

$$\frac{M(x)\times x^r}{G(x)}=Q(x)+\frac{R(x)}{G(x)}$$

（要产生r位余数，所以除数应为r+1位）

(4) 将左移r位的待编码信息与余数R(x)做模2加，得到CRC码

$$M(x)\cdot x^r+R(x)=Q(x)\cdot G(x)$$

说明CRC码能被生成多项式G(x)整除（此结论是CRC译码纠错的依据）

以上说明CRC码能被生成多项式整除，这是译码与纠错的依据。

CRC码的校验方法

- 一个CRC码一定能被生成多项式整除，当数据和校验位一起送到接受端后，只要将接受到的数据和校验位用约定好的同样的生成多项式除，如果能除尽，表明没有发生错误；若除不尽，则表明某些数据位发生了错误，余数将指明出错位所在的位置。

CRC码计算

例：信息码为1100，生成多项式$G(x)=x^3+x+1$，求CRC码。

解：$M(x)=x^3+x^2$　　$G(x)=1011$　　$r=3$

$M(x)\cdot x^3=1100\ 000$

$$\frac{M(x)\cdot x^3}{G(x)}=\frac{1100000}{1011}=1110+\frac{010}{1011}$$

CRC码为1100 010

循环冗余码举例

$X^3 \cdot M(x) \div G(x) = (x^8 + x^4 + x^3) \div (x^3 + 1)$

```
          100111
      ------------
1001 | 100011000
       1001
       ----
        0011
        0000
        ----
         0111
         0000
         ----
          1110
          1001
          ----
          1110
          1001
          ----
          1110
          1001
          ----
           111
```

（模 2 运算不考虑加法进位和减法借位，上商的原则是当部分余数首位是 1 时商取 1，反之商取 0。然后按模 2 相减原则求得最高位后面几位的余数。这样当被除数逐步除完时，最后的余数位数比除数少一位。这样得到的余数就是校验位，此例中最终的余数有3位。）

校验位为111，CRC码为100011 111。如果要校验CRC码，可将CRC码用同一个多项式相除，若余数为0，则说明无错；否则说明有错。例如，若在接收方的CRC码也为100011 111时，用同一个多项式相除后余数为0。若接收方CRC码不为100011 111时，余数则不为0。

表 4.5 对应 $G(x)=1011$ 的（7，4）循环的出错模式

序号	N_1	N_2	N_3	N_4	N_5	N_6	N_7	余数	出错位
正确	1	1	0	0	0	1	0	000	无
错误	1	1	0	0	0	1	1	001	7
	1	1	0	0	0	0	0	010	6
	1	1	0	0	1	1	0	100	5
	1	1	0	1	0	1	0	011	4
	1	1	1	0	0	1	0	110	3
	1	0	0	0	0	1	0	111	2
	0	1	0	0	0	1	0	101	1

Thank You !

Institute of Computer Architecture