

存储系统



第4章 存储体系

4.1 存储体系概念和并行存储系统

4.2 虚拟存储系统

4.3 高速缓冲存储器(Cache)

4.4 Cache—主存—辅存三级层次

4.5 ARM存储系统



现代计算机系统都以存储器为中心。

在计算机运行过程中，存储器是各种信息存储和交换的中心。

4.1.1 存储体系的引出

存储器容量 $S_M = W \cdot l \cdot m$ 。其中， W 为存储体的字长(单位为位或字节)， l 为每个存储体的字数， m 为并行工作的存储体个数。

存储系统和存储器是两个完全不同的概念。

在一台计算机中，通常有多种存储器。

种类：主存储器、**Cache**、通用寄存器、先行缓冲存储器、磁盘存储器、磁带存储器、光盘存储器等。

材料工艺：**ECL**、**TTL**、**MOS**、磁表面、激光，**SRAM**，**DRAM**。

访问方式：直接译码、先进先出、随机访问、相联访问、块传送、文件组。

4.1 存储体系概念和并行存储系统

存储器的主要性能：**速度、容量、价格**

速度用存储器的访问周期、读出时间、频带宽度等表示。

容量用字节**B**、千字节**KB**、兆字节**MB**和千兆字节**GB**等单位表示。

价格用单位容量的价格表示，如**\$/bit**。

存储系统的**关键**是如何组织好速度、容量和价格均不相同的存储器，使这个**存储器**的速度接近速度最快的那个存储器，存储容量与容量最大的那个存储器相等，单位容量的价格接近最便宜的那个存储器。

4.1 存储体系概念和并行存储系统

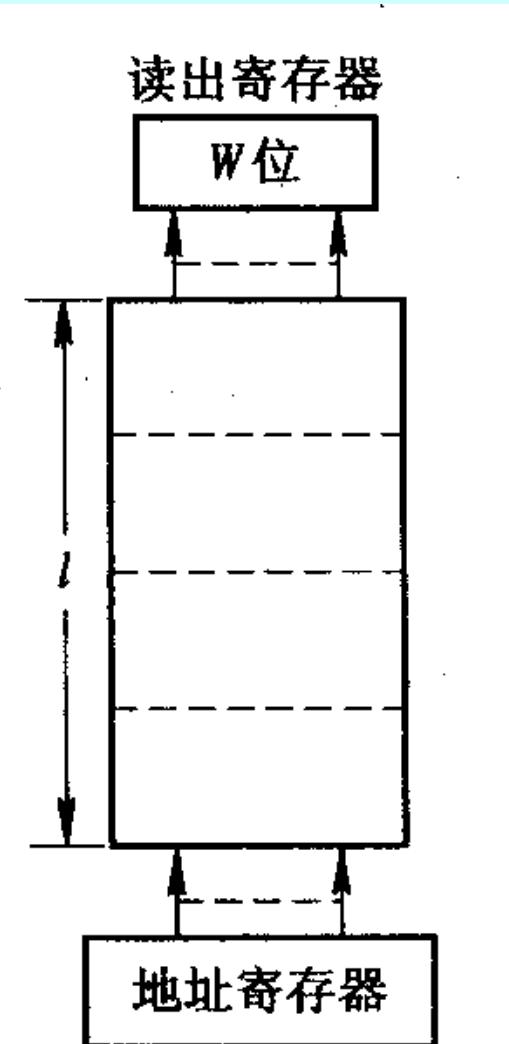
存储器的速度可以用访问时间 T_A 、存储周期 T_M 和频宽(也称带宽) B_m 来描述。 T_A 是存储器从接到访存读申请，到信息被读到数据总线上所需的时间。 T_M 则是连续启动一个存储体所需要的间隔时间，它一般总比 T_A 大。存储器频宽是存储器可提供的数据传送速率，一般用每秒钟传送的信息位数(或字节数)来衡量，又分最大频宽(或称极限频宽)和实际频宽。最大频宽 B_m 是存储器连续访问时能提供的频宽。单体的 $B_m = W/T_M$ 。 m 个存储体并行工作时可达到的最大频宽 $B_m = W \cdot m/T_M$ 。由于存储器不一定总能连续满负荷地工作，所以，实际频宽往往要低于最大频宽。

4.1 存储体系概念和并行存储系统

计算机系统总希望存储器能在尽可能低的价格下，提供尽量高的速度和尽量大的存储容量。在速度上应尽量和CPU匹配，否则CPU的高速性能难以发挥。容量上应尽可能放得下所有系统软件及多个用户软件。同时，存储器的价格又只能占整个计算机系统硬件价格中一个较小而合理的比例。然而，价格、速度和容量的要求是互相矛盾的。在存储器件一定的条件下，容量越大，因其延迟增大而使速度越低。容量越大，存储器总价格当然也就会越大。存取速度越高，价格也将越高。同等容量的情况下存储器的速度大体上按双极型、MOS型、电荷耦合器件(CCD)、磁泡、定头磁盘、动头磁盘、磁带的顺序依次下降。

4.1 存储体系概念和并行存储系统

4.1.2 并行存储系统



图北京理工大学单体单字节存储器

图4.1是一个字长为W位的单体主存，一次可以访问一个存储器字，所以主存最大频宽 $B_m=W/T_M$ 。假设，此存储器字长W与CPU所要访问的字(数据字或指令字，简称CPU字)的字长W相同，则CPU从主存获得信息的速率就为 W/T_M 。我们称这种主存是单体单字存储器。



4.1 存储体系概念和并行存储系统

要想提高主存频宽 B_m ，使之与CPU速度匹配，显然可以想到，在同样的器件条件(即同样的 T_M)下，只有设法提高存储器的字长W才行。例如，改用图4.2的方式组成，这样，主存在一个存储周期内就可以读出4个CPU字，相当于CPU从主存中获得信息的最大速率提高到原来的4倍，即 $B_m=4W/T_M$ 。我们称这种主存为单体多字存储器。

4.1 存储体系概念和并行存储系统

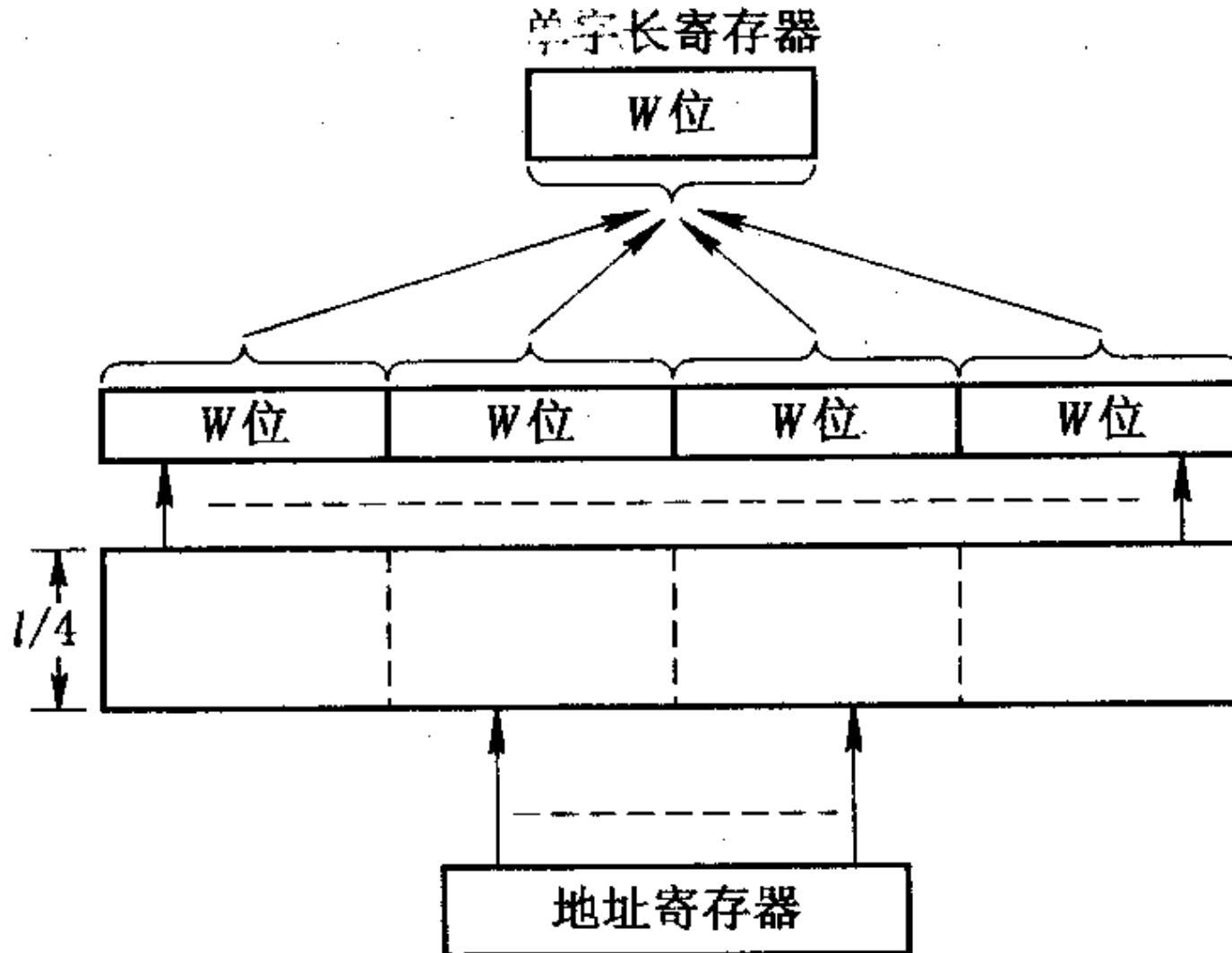


图 4.2 单体多字($m=4$)存储器
北京理工大学计算机学院

方法：把 m 字 w 位的存储器改变成为 m/n 字 $n \times w$ 位的存储器（单体多字）。

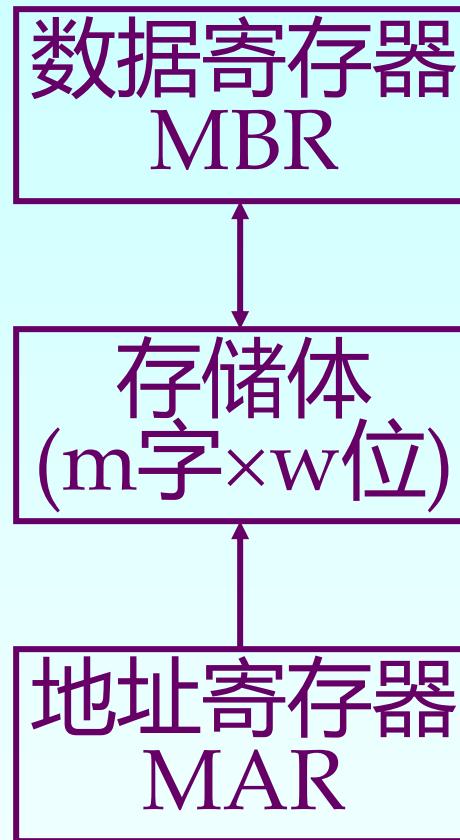
逻辑实现：

把地址码分成两个部分，一部分作为存储器的地址，另一部分负责选择数据。

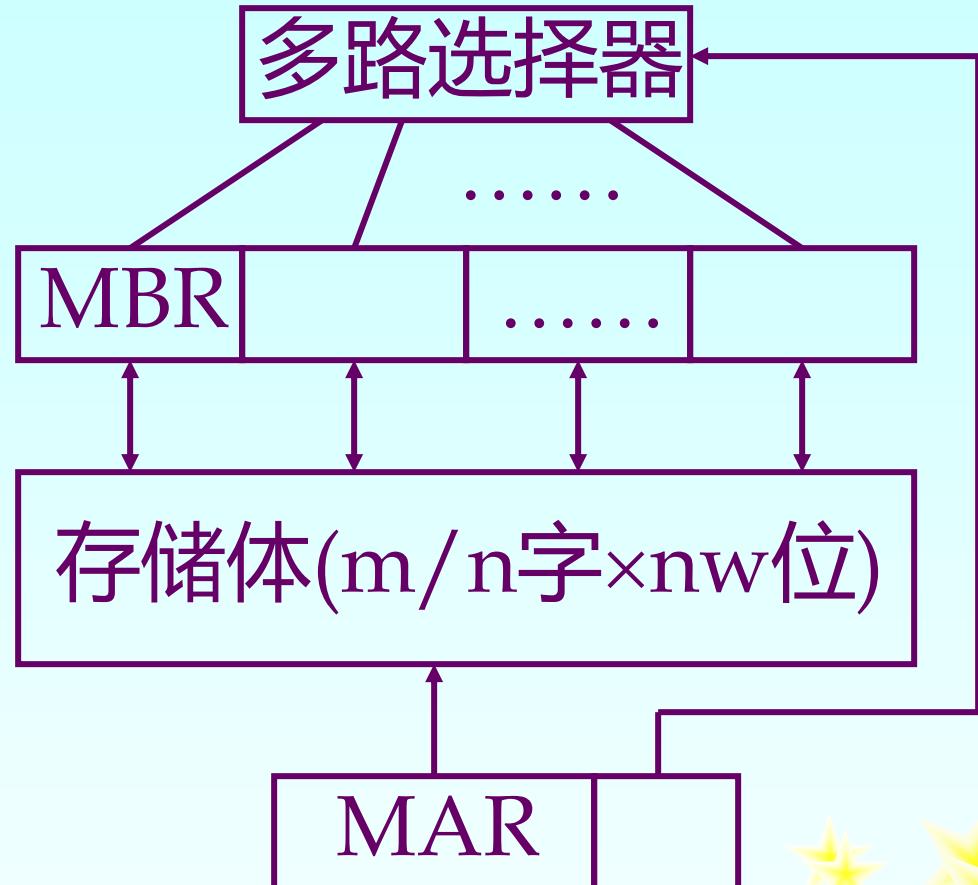
主要缺点：访问冲突大

- (1) 取指令冲突
- (2) 读操作数冲突
- (3) 写数据冲突
- (4) 读写冲突





一般存储器



并行访问存储器

4.1 存储体系概念和并行存储系统

多体交叉存储器：每个存储体都是CPU字的宽度，称为多体单字存取。

这种交叉又有低位交叉和高位交叉两种。



(1)高位交叉访问存储器

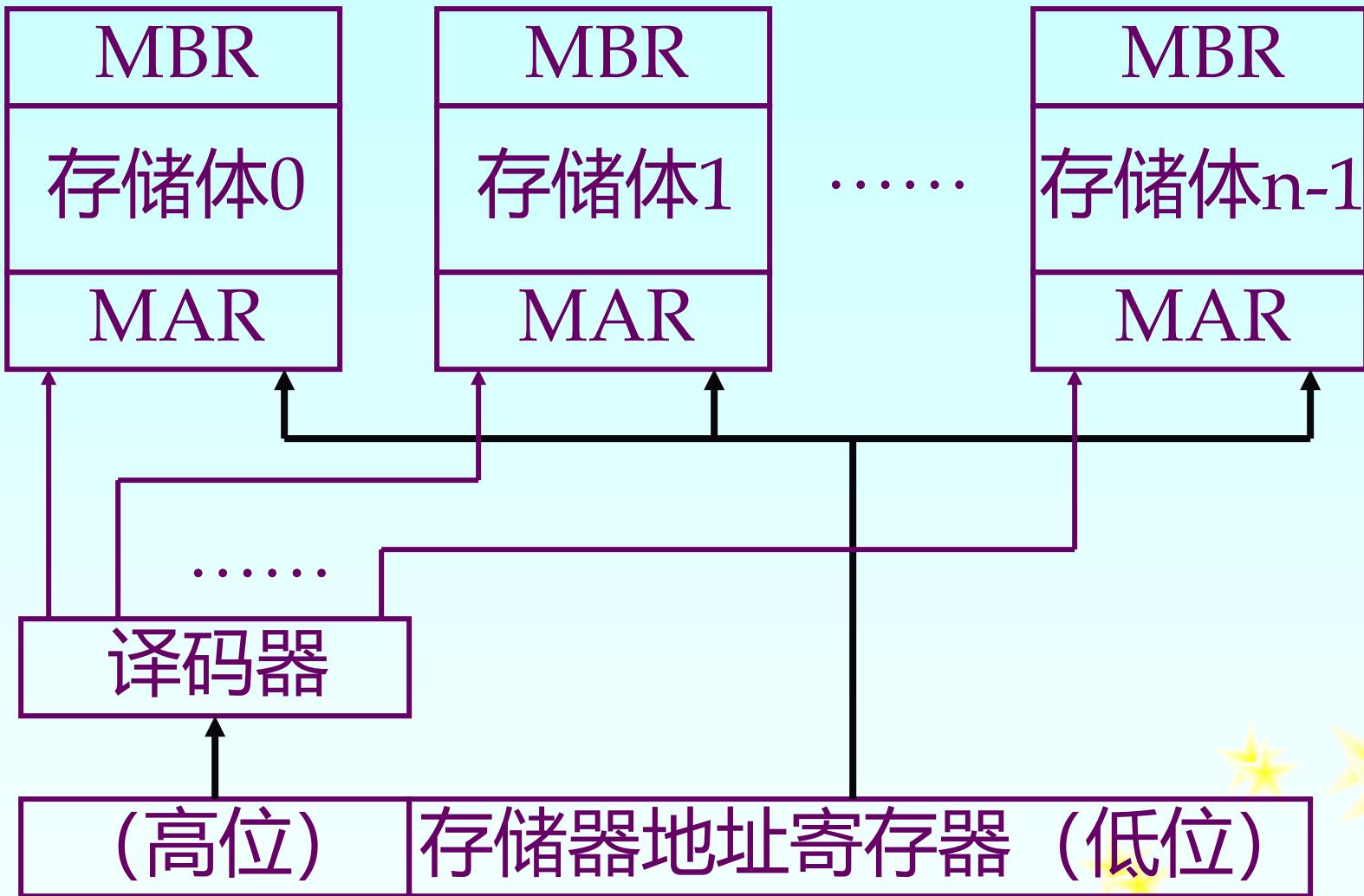
主要目的： 扩大存储器容量

实现方法： 用地址码的高位区分存储体号

每个存储模块都有各自独立的控制部件，每个存储模块可以独立工作。但由于程序的局部性原理，通常只有一个存储模块在不停地忙碌，其他存储模块是空闲的。



4.1 存储体系概念和并行存储系统



(2) 低位交叉访问存储器

主要目的：提高存储器访问速度

实现方法：用地址码的低位区分存储体号

n 个存储体分时启动

实际上是一种采用流水线方式工作的并行存储器。理论上，存储器的速度可望提高 n 倍。



4.1 存储体系概念和并行存储系统

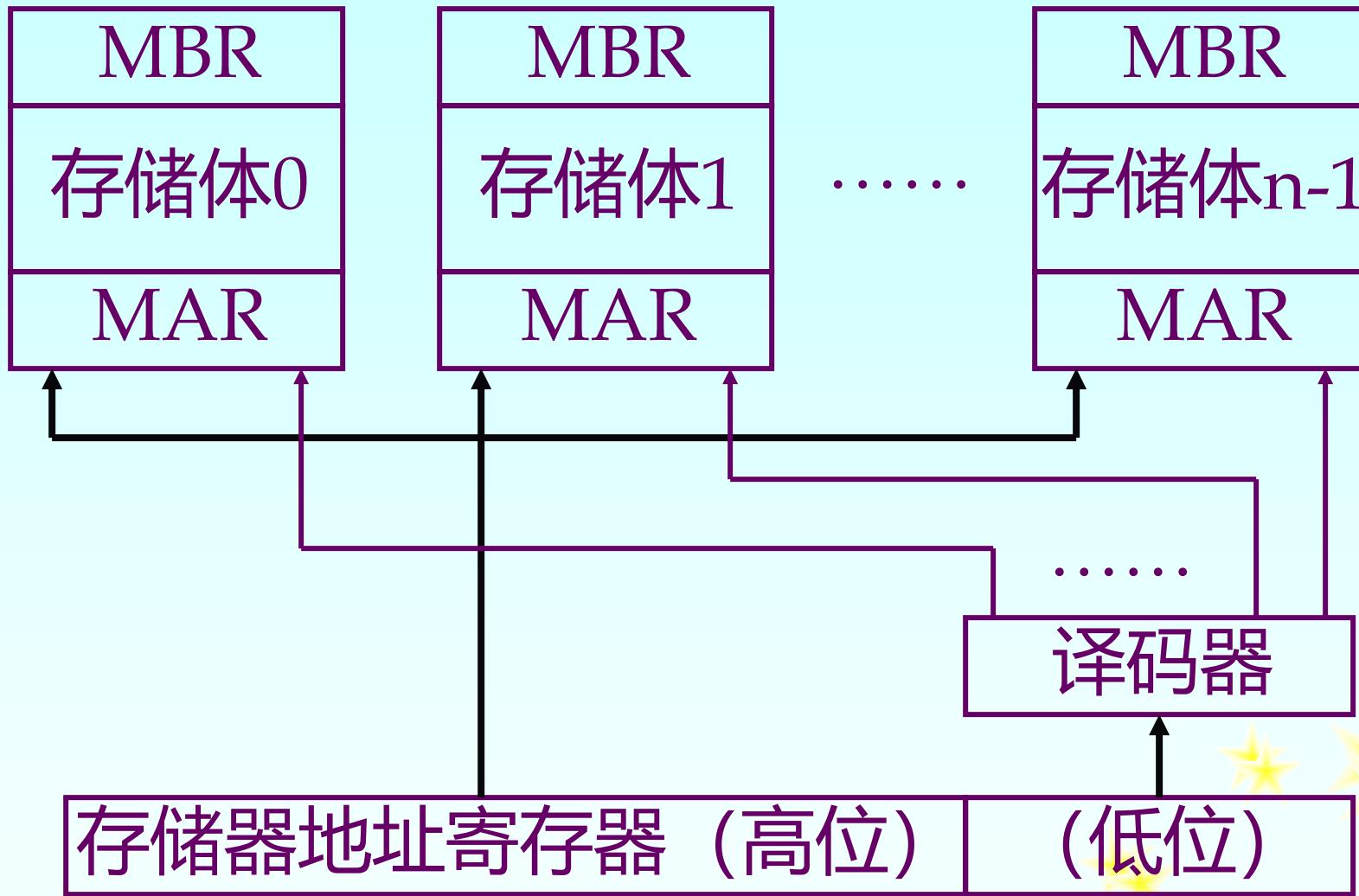


表 4.1 地址的模4低位交叉编址

模 体	地址编址序列	对应二进制地址码最末二位的状态
M_0	$0, 4, 8, 12, \dots, 4i+0, \dots$	00
M_1	$1, 5, 9, 13, \dots, 4i+1, \dots$	01
M_2	$2, 6, 10, 14, \dots, 4i+2, \dots$	10
M_3	$3, 7, 11, 15, \dots, 4i+3, \dots$	11



4.1 存储体系概念和并行存储系统



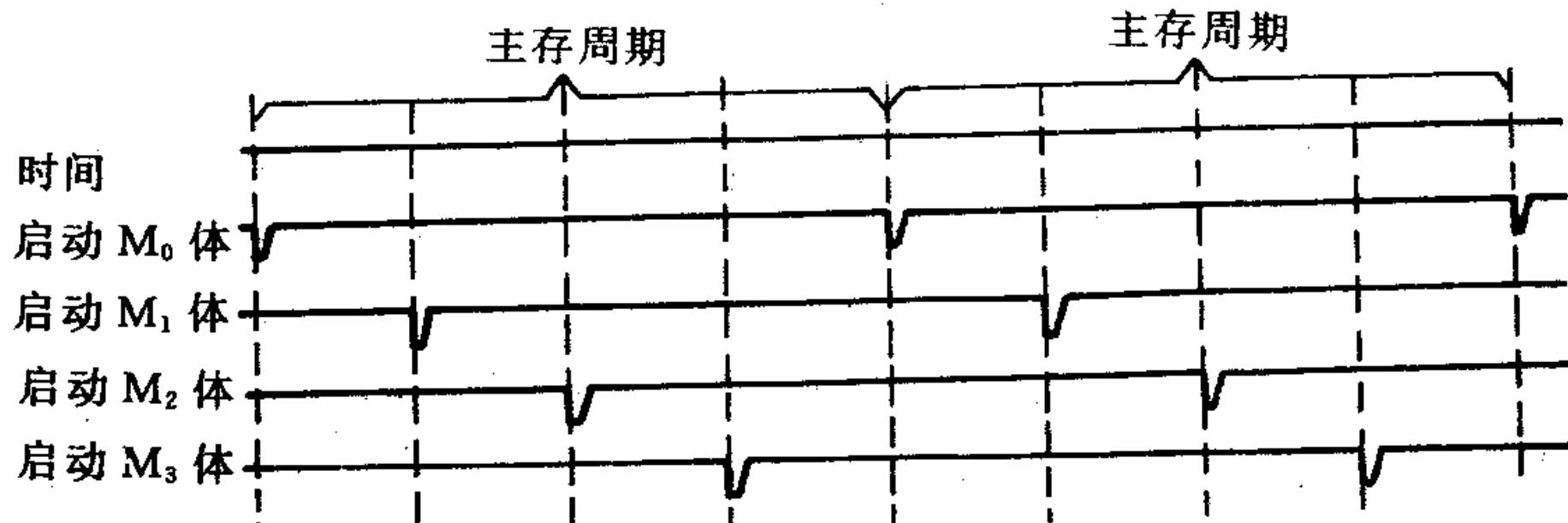


图 4.6 4个分体分时启动的时间关系



4.1.3 存储体系定义和分支

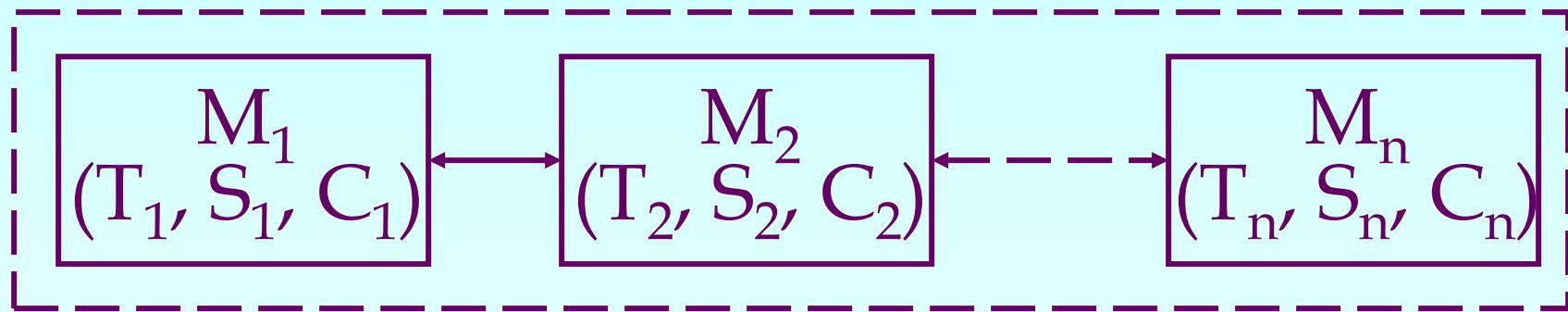
1. 存储系统(存储体系、存储层次)的定义

两个或两个以上速度、容量和价格各不相同的存储器用硬件、软件、或软件与硬件相结合的方法连接起来成为一个存储系统。这个系统对应用程序员透明，并且，从应用程序员看，它是一个存储器，这个存储器的速度接近速度最快的那个存储器，存储容量与容量最大的那个存储器相等，单位容量的价格接近最便宜的那个存储器。



4.1 存储体系概念和并行存储系统

从外部看：



$T \approx \min(T_1, T_2, \dots, T_n)$, 用存储周期表示

$S \approx \max(S_1, S_2, \dots, S_n)$, 用MB或GB表示

$C \approx \min(C_1, C_2, \dots, C_n)$, 用每位的价格表示

4.1 存储体系概念和并行存储系统

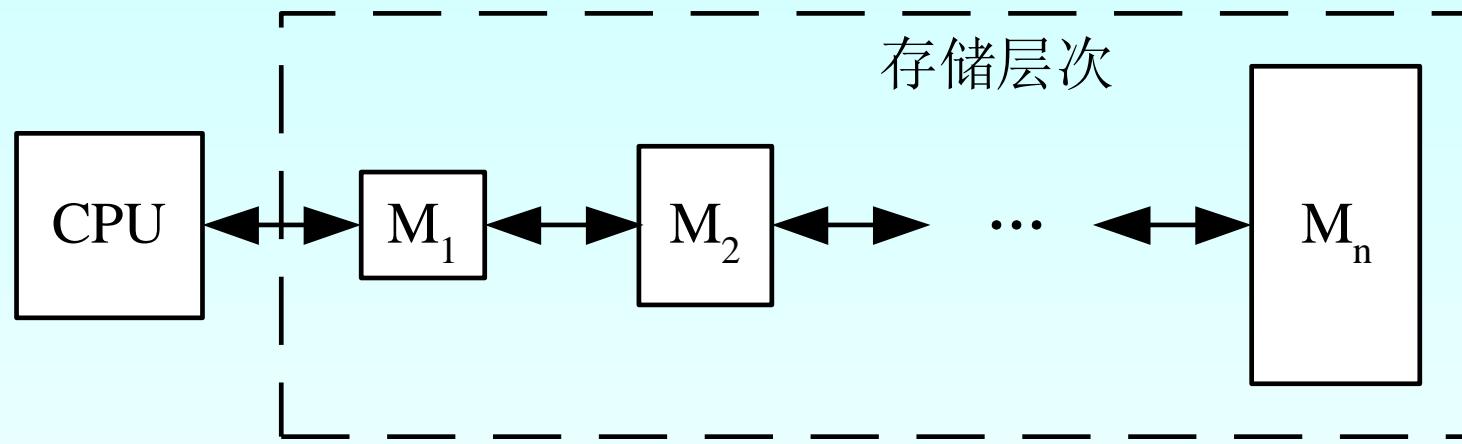


图4.8 多级存储层次



4.1 存储体系概念和并行存储系统

主存—辅存层次

又称虚拟存储系统，由主存储器和磁盘存储器构成。

主要目的：扩大存储器容量，弥补主存容量的不足。

在主存和辅存之间，增加辅助的软硬件，让它们构成一个整体。从**CPU**看，速度接近主存的速度，容量是虚拟的地址空间，每位价格是接近于辅存的价格。由于虚拟存储系统需要通过操作系统来调度，因此对**系统程序员是不透明的，但对应用程序员是透明的**。

4.1 存储体系概念和并行存储系统

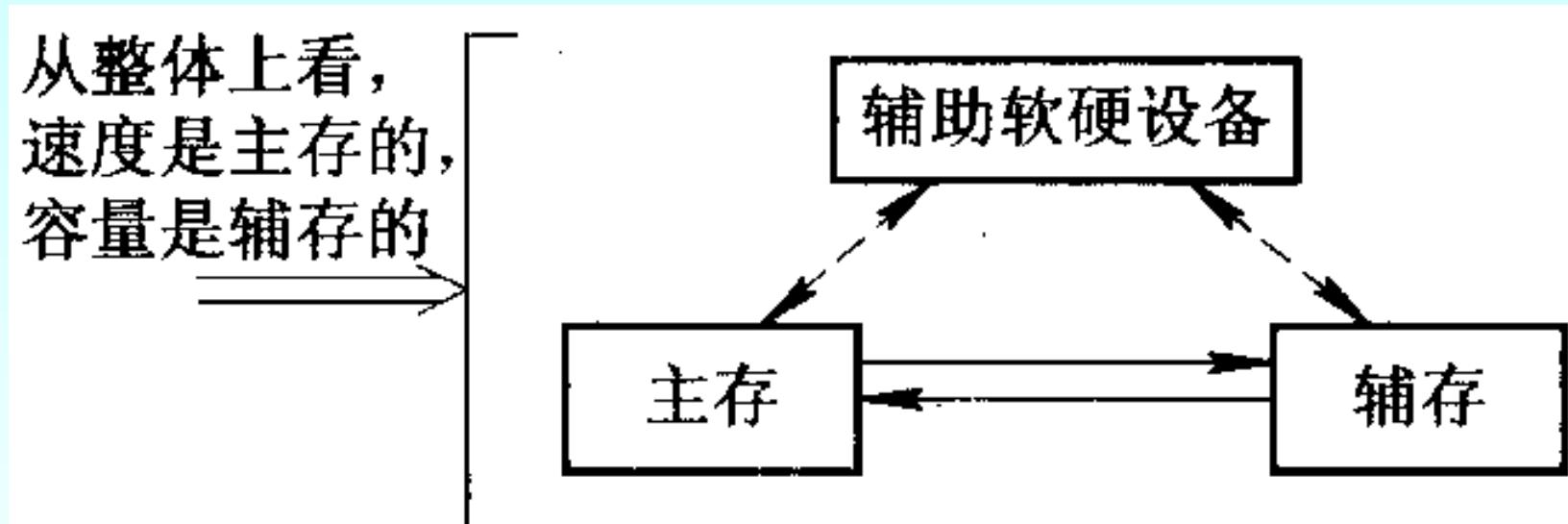


图 主存—辅存存储层次



Cache—主存层次：

又称**Cache**存储系统，由**Cache**和主存储器构成。

主要目的：提高存储器速度，弥补主存速度的不足。

在**Cache**和主存之间，增加辅助硬件，让它们构成一个整体。从**CPU**看，速度接近**Cache**的速度，容量是主存的容量，每位价格接近于主存的价格。由于**Cache**存储系统全部用硬件来调度，因此它对系统程序员和应用程序员都是透明的。

4.1 存储体系概念和并行存储系统

从CPU看，
速度是Cache的，
容量是主存的

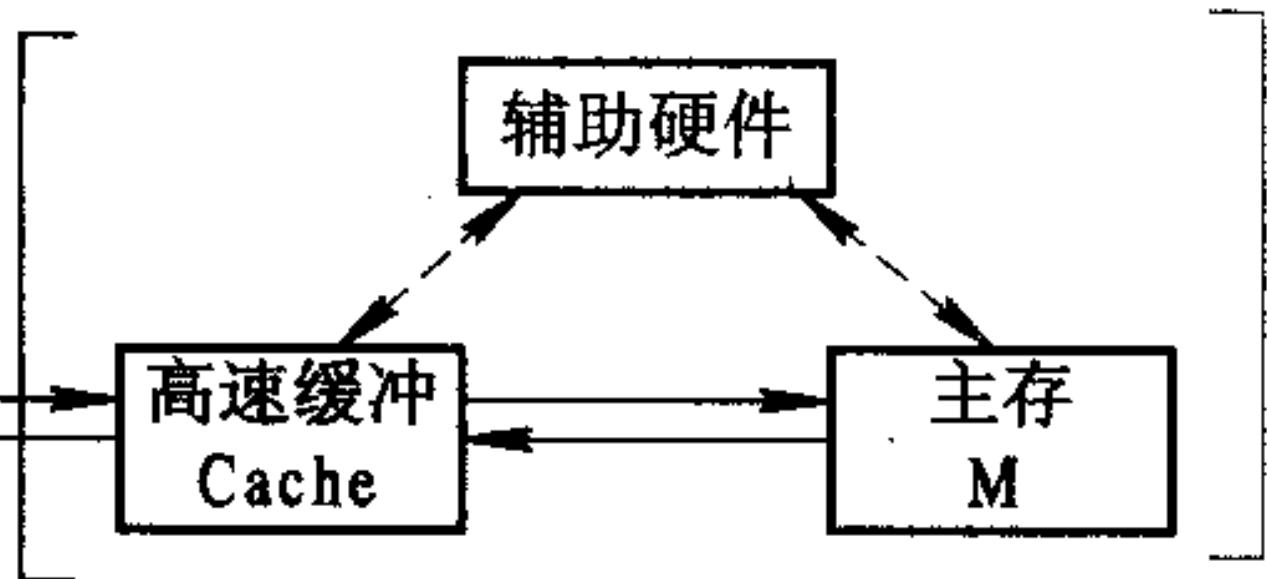


图 Cache—主存存储层次



4.1 存储体系概念和并行存储系统

从CPU看，
速度是Cache的，
容量是主存的

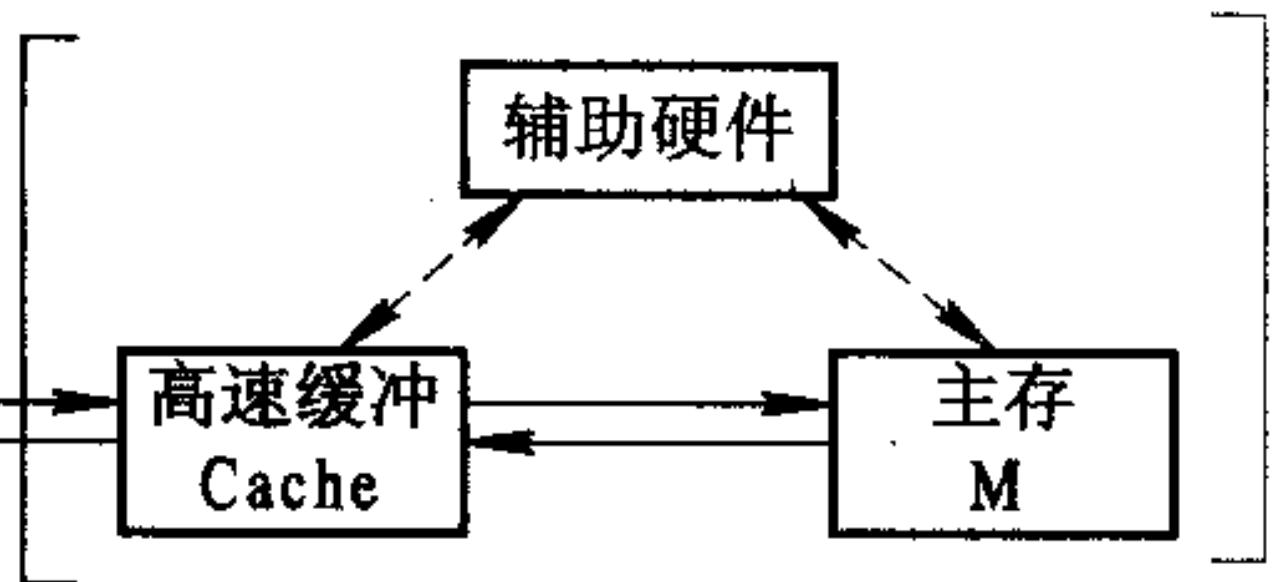


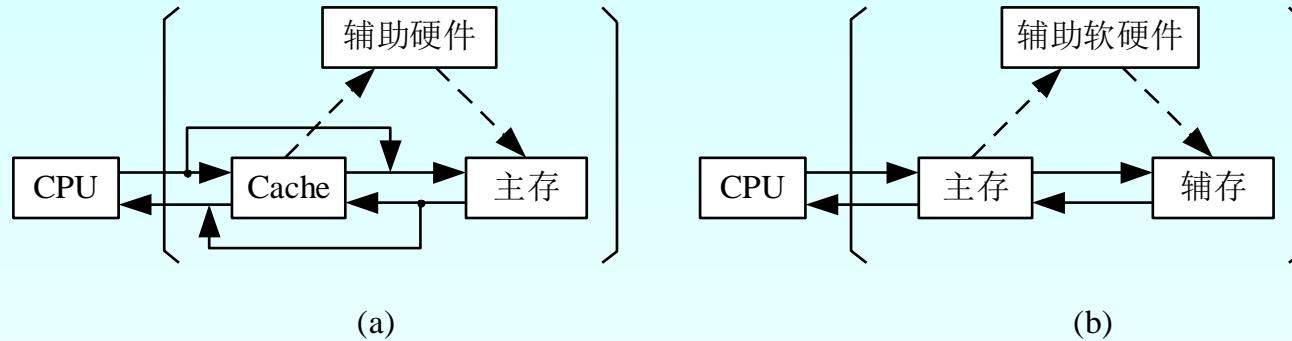
图 Cache—主存存储层次



4.1 存储体系概念和并行存储系统

2. 存储体系分支

基本的两级存储体系是虚拟存储系统和Cache存储系统，这是存储体系的两个不同分支。



(a) Cache存储系统 (b) 虚拟存储系统
图4.9 两种存储系统

程序局部性原理

局部性分为时间上的局部性和空间上的局部性。时间上的局部性是指最近访问的代码是不久将被访问的代码，这是由程序循环造成的。空间上的局部性是指那些地址上相邻近的代码可能会被一起访问，这主要是由于指令通常是顺序执行的，以及数据一般是以向量、阵列等形式簇聚地存储所致。

所以，程序在执行时所用到的指令和数据的地址分布不会是随机的，而是相对簇聚的。

4.1.4 存储体系的性能参数

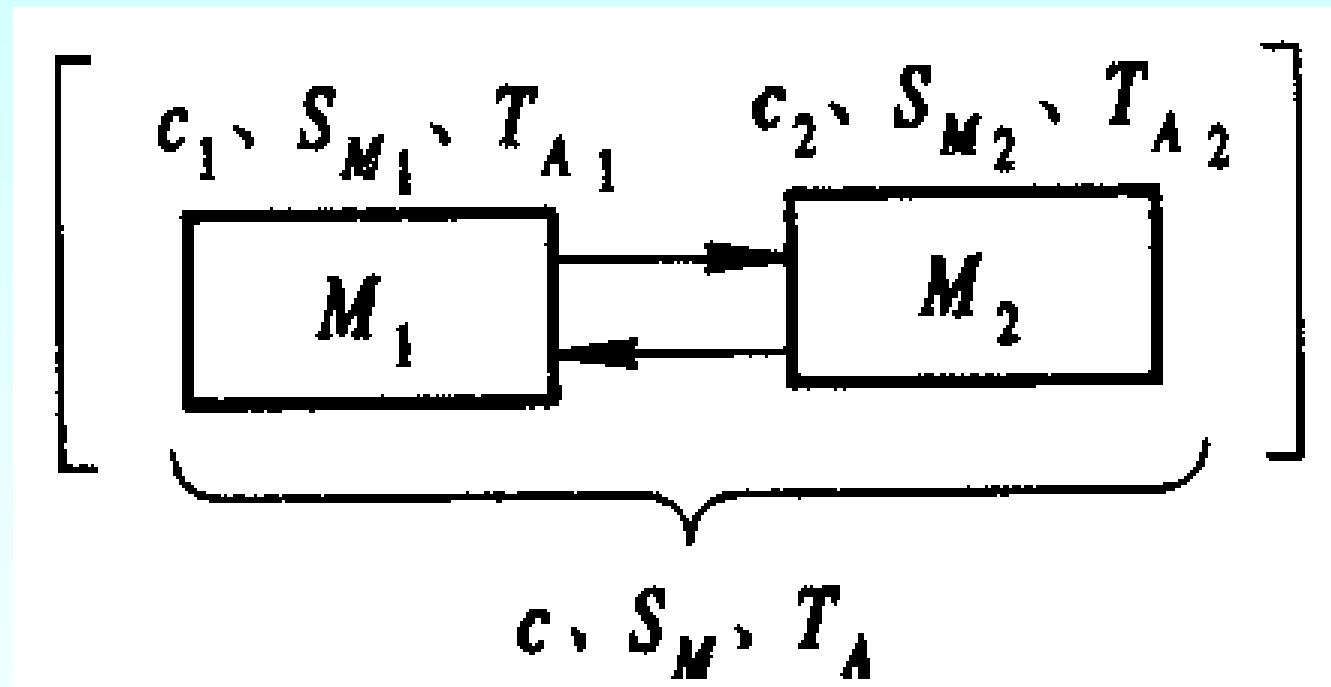


图 4.10 二级存储体系的评价

4.1 存储体系概念和并行存储系统

设 c_i 为 M_i 的每位价格， S_{M_i} 为 M_i 的以位计算的存储容量， T_{Ai} 为CPU访问到 M_i 中的信息所需要的时间。为评价存储层次的性能，引入存储层次的每位价格 c 、命中率 H 和等效访问时间 T_A 。

存储层次的每位平均价格

$$c = \frac{c_1 S_{M_1} + c_2 S_{M_2}}{S_{M_1} + S_{M_2}}$$

$S_2 >> S_1$ 时， $C \approx C_2$ ，但 S_2 与 S_1 不能相差太大

存储系统的容量

要求：

存储系统的容量等于**M2**存储器的容量。

提供尽可能大、能随机访问的地址空间。

方法有两种：

只对**M2**存储器进行编址，**M1**存储器只在内部编址。

另外设计一个容量很大的逻辑地址空间。

命中率 H 定义为CPU产生的逻辑地址能在 M_1 中访问到(命中到)的概率。命中率可用实验或模拟方法来获得，即执行或模拟一组有代表性的程序，若逻辑地址流的信息能在 M_1 中访问到的次数为 R_1 ，当时在 M_2 中还未调到 M_1 的次数为 R_2 ，则命中率

$$H = \frac{R_1}{R_1 + R_2}$$



4.1 存储体系概念和并行存储系统

不命中率为 H , 两级存储层次的等效访问时间 T_A 根据 M_2 的启动时间有:

假设 M_1 访问和 M_2 访问是同时启动的,

$$T_A = H \times T_{A1} + (1 - H) \times T_{A2}$$

假设 M_1 不命中时才启动 M_2 ,

$$\begin{aligned} T_A &= H \times T_{A1} + (1 - H) \times (T_{A1} + T_{A2}) \\ &= T_{A1} + (1 - H) \times T_{A2} \end{aligned}$$



4.1 存储体系概念和并行存储系统

存储层次的等效访问时间

$$T_A = HT_{A_1} + (1 - H)T_{A_2}$$

设CPU对存储层次相邻二级的访问时间比 $r=T_{A_2}/T_{A_1}$, 存储系统的访问效率

$$e = \frac{T_{A_1}}{T_A} = \frac{T_{A_1}}{HT_{A_1} + (1 - H)T_{A_2}} = \frac{1}{H + (1 - H)r}$$

存储系统的访问效率主要与命中率和两级存储器的速度之比有关。

4.1 存储体系概念和并行存储系统

访问效率

$$e = T_{A_1} / T_A$$

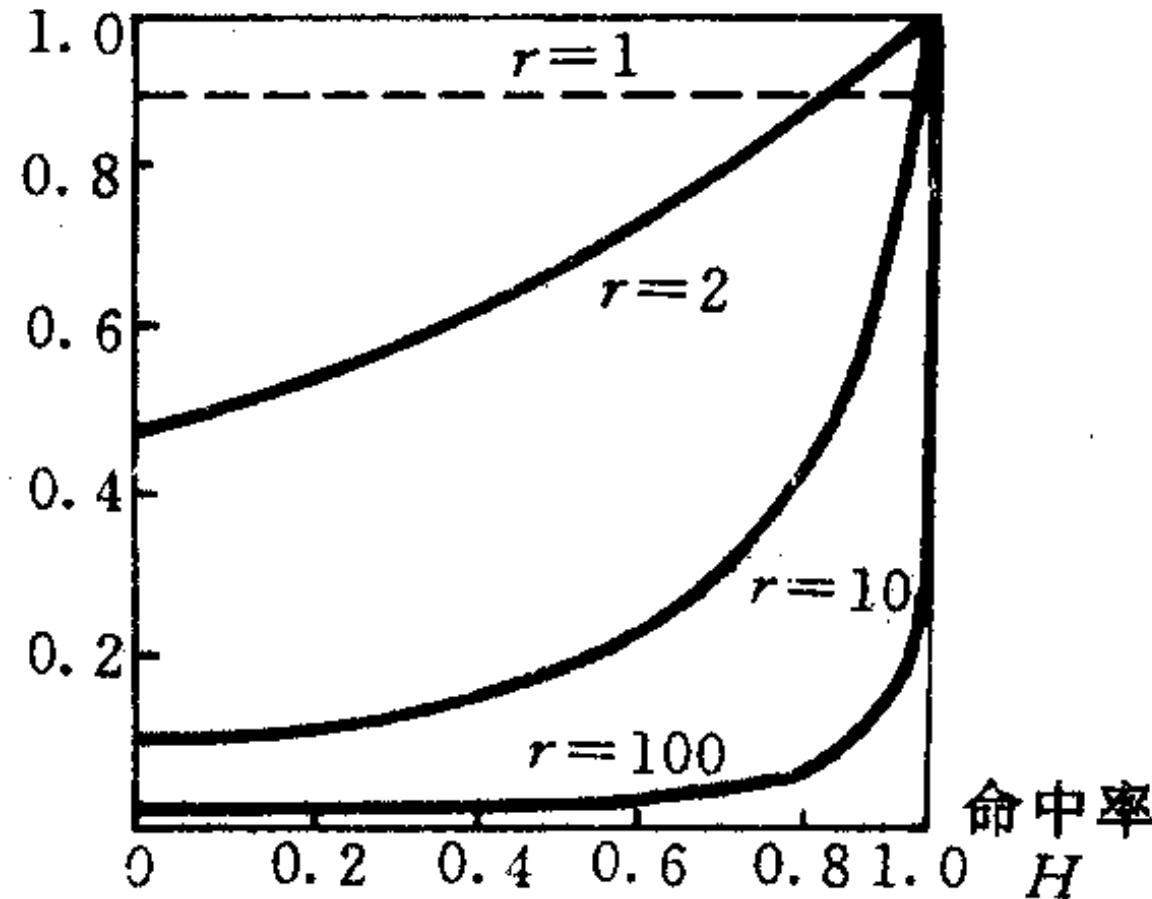


图 4.10 对于不同的 r , 命中率 H 与
访问效率 e 的关系

例1：假设某计算机的存储系统由**Cache**和主存组成。某程序执行过程中访存**1000**次，其中访问**Cache**失效（未命中）**50**次，则**Cache**的命中率是多少？

解：程序访存次数**N1+N2=1000**次，其中访问**Cache**的次数**N1**为访存次数减去失效次数。

$$H = \frac{1000 - 50}{1000} = 95\%$$



4.1 存储体系概念和并行存储系统

例2：CPU执行一段程序时，Cache完成存取的次数为**5000**次，主存完成存取的次数为**200**次。已知Cache存储周期TC为**40ns**，主存存取周期TM为**160ns**。分别求：

- (1) Cache的命中率H；
- (2) 等效访问时间TA；
- (3) Cache—主存系统的访问效率e。

解：(1) $H = \frac{5000}{5000 + 200} \approx 96\%$

(2) $TA = H \times TA_1 + (1-H) \times TA_2$

$= 0.96 \times 40\text{ns} + (1-0.96) \times 160\text{ns} = 44.8\text{ns}$

(3) $e = \frac{T_{A1}}{T_A} = 40/44.8 = 89.3\%$

例3：假设 $T_2=5T_1$ ，在命中率H为0.9和0.99两种情况下，分别计算存储系统的访问效率。

解：

当H=0.9时，

$$e_1 = 1 / (0.9 + 5(1 - 0.9)) = 0.72$$

当H=0.99时，

$$e_2 = 1 / (0.99 + 5(1 - 0.99)) = 0.96$$



提高存储系统速度的两条途径：

一是提高命中率H。

二是两个存储器的速度不要相差太大。

其中第二条有时做不到(如虚拟存储系统)，主要依靠提高命中率。



例4：在虚拟存储系统中，两级存储器的速度相差特别悬殊 $T_2=10^5T_1$ 。如果要使访问效率 $e=0.9$ ，问需要有多高的命中率？

解：

$$0.9 = \frac{1}{H + (1 - H) \cdot 10^5}$$

$$\begin{aligned}0.9H + 90000(1 - H) &= 1 \\89999.1H &= 89999\end{aligned}$$

得：

$$H = 0.999998888877777... \approx 0.999999$$



采用预取技术提高命中率

方法：不命中时，把**M2**存储器中相邻几个单元组成的一个数据块都取出来送入**M1**存储器中。

计算公式：

$$H' = \frac{H + n - 1}{n}$$

其中：**H'**是采用预取技术后的命中率；**H**是原来的命中率；**n**为数据块大小与数据重复使用次数的乘积。

4.1 存储体系概念和并行存储系统

证明：

采用预取技术之后，不命中率降低n倍：

$$H' = 1 - \frac{1 - H}{n} = \frac{H + n - 1}{n}$$

也可以采用另外一种证明方法：在原有命中率计算公式中，把访问次数扩大到n倍，这时，由于采用了预取技术，命中次数为：
nN₁+(n-1)N₂，不命中次数仍为**N₂**，因此新的命中率为：

$$H' = \frac{nN_1 + (n-1)N_2}{nN_1 + nN_2} = \frac{N_1 + (nN_1 + nN_2) - (N_1 + N_2)}{nN_1 + nN_2} = \frac{H + n - 1}{n}$$

例5：在一个**Cache**存储系统中，当**Cache**的块大小为一个字时，命中率**H=0.8**；假设数据的重复利用率为**5**，计算块大小为4个字时，**Cache**存储系统的命中率是多少？假设**T₂=5T₁**，分别计算访问效率。

解：

n=4×5=20, 采用预取技术后，命中率提高到：

$$H' = \frac{H + n - 1}{n} = \frac{0.8 + 20 - 1}{20} = 0.99$$



Cache块为1个字大时, **H=0.8**, 访问效率为:

$$e_1 = 1/(0.8 + 5(1 - 0.8)) = 0.55$$

Cache块为4个字大时, **H=0.99**, 访问效率为:

$$e_2 = 1/(0.99 + 5(1 - 0.99)) = 0.96$$



4.1 存储体系概念和并行存储系统

例6：在一个虚拟存储系统中， $T_2=10^5 T_1$ ，原来的命中率只有**0.8**，如果访问磁盘存储器的数据块大小为**4K字**，并要求访问效率不低于**0.9**，计算数据在主存储器中的重复利用率至少为多少？

解：

假设数据在主存储器中的重复利用率为**m**，根据前面的给出关系：

$$0.9 = \frac{1}{H' + (1 - H') \cdot 10^5}, \quad H' = \frac{0.8 + 4096m - 1}{4096m}$$

解方程组得 **$m=44$** ，即数据在主存储器中的重复利用率至少为**44次**。

存储器的层次结构

多个层次的存储器：

Register Files \Rightarrow Buffers(Lookahead)

\Rightarrow **Cache \Rightarrow Main Memory \Rightarrow**

Online Storage \Rightarrow Off-line Storage

如下页图所示，如果用*i*表示层数，则有：

工作速度： $T_i < T_{i+1}$

存储容量： $S_i < S_{i+1}$

单位价格： $C_i > C_{i+1}$

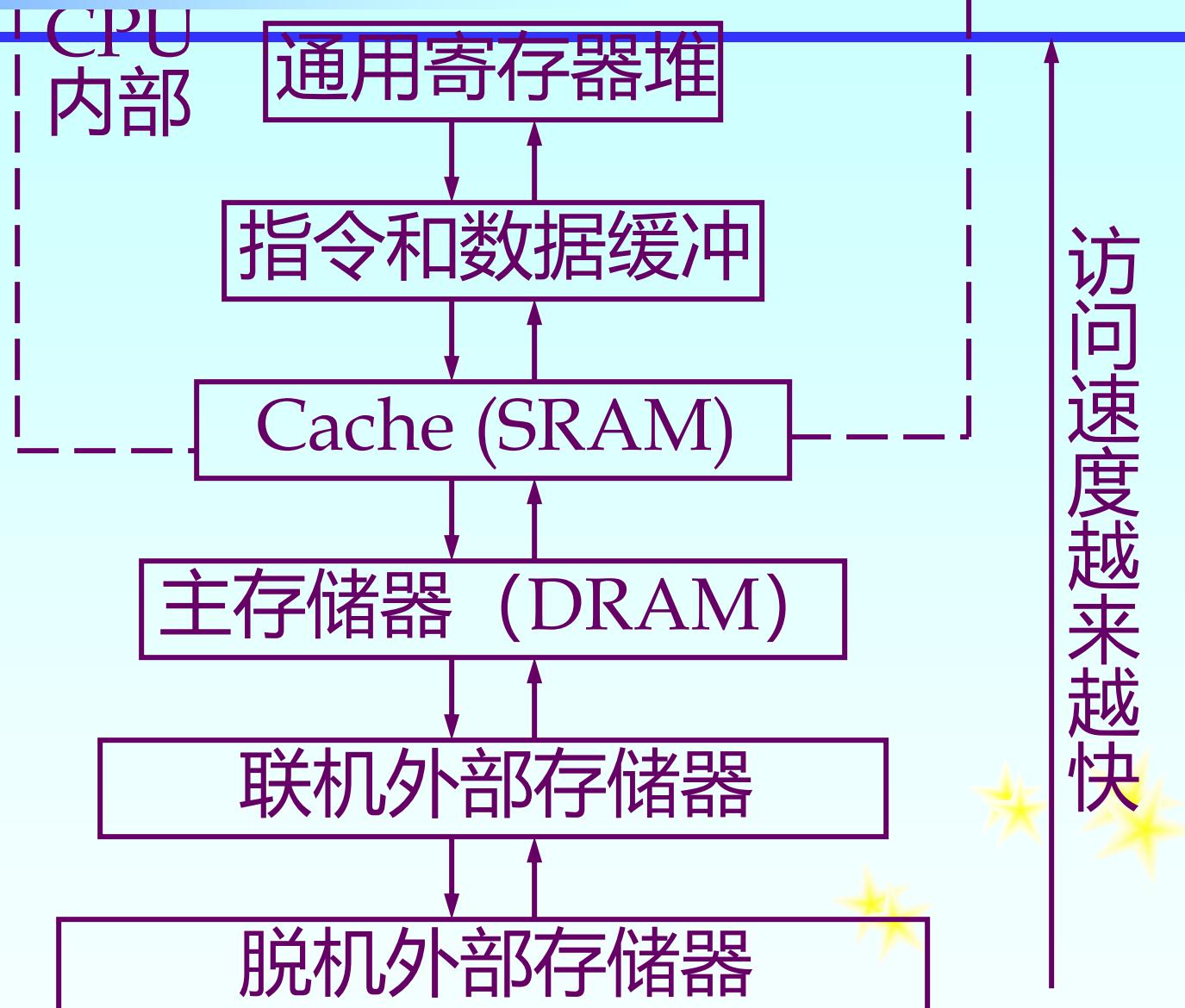


4.1 存储体系概念和并行存储系统

计算机体系结构

存储位的容量越来越大
价格越来越便宜

访问速度越来越快



各级存储器的主要性能特性

存储器层次	通用寄存器	缓冲栈	Cache
存储周期	< 10ns	< 10ns	10 - 60ns
存储容量	< 512B	< 512B	8K-2MB
价格\$c/KB	1200	80	3.2
访问方式	直接译码	先进先出	相联访问
材料工艺	ECL	ECL	SRAM
分配管理	编译器分配	硬件调度	硬件调度
带宽	400-8000	400-1200	200-800

(待续)

各级存储器的主要性能特性 (续)

存储器层次	主存储器	磁盘存储器	脱机存储器
存储周期	60-300ns	10 - 30ms	2 - 20 min
存储容量	32M-1GB	1G-1TB	5G-10TB
价格\$c/KB	0.36	0.01	0.0001
访问方式	随机访问	块访问	文件组
材料工艺	DRAM	磁表面	磁、光等
分配管理	操作系统	系统/用户	系统/用户
带宽	80-160	10-100	0.2 - 0.6

CPU与主存储器的速度差距越来越大

1955年，第一台大型机**IBM704**，
CPU和主存储器的工作周期均为**12微秒**，
目前，**CPU**的工作速度提高了**4个数量级**以
上，主存储器的工作速度仅提高两个数量
级。今后，**CPU**与主存储器的速度差距会
更大。

研究存储系统的目的就是要找出解决
这一问题的办法。



频带平衡

计算机中各级存储器频带应该达到平衡。

例如：一台速度为**500MIPS**的计算机系统，主存储器的各种访问源的频带宽度如下：

CPU取指令：500MW/s

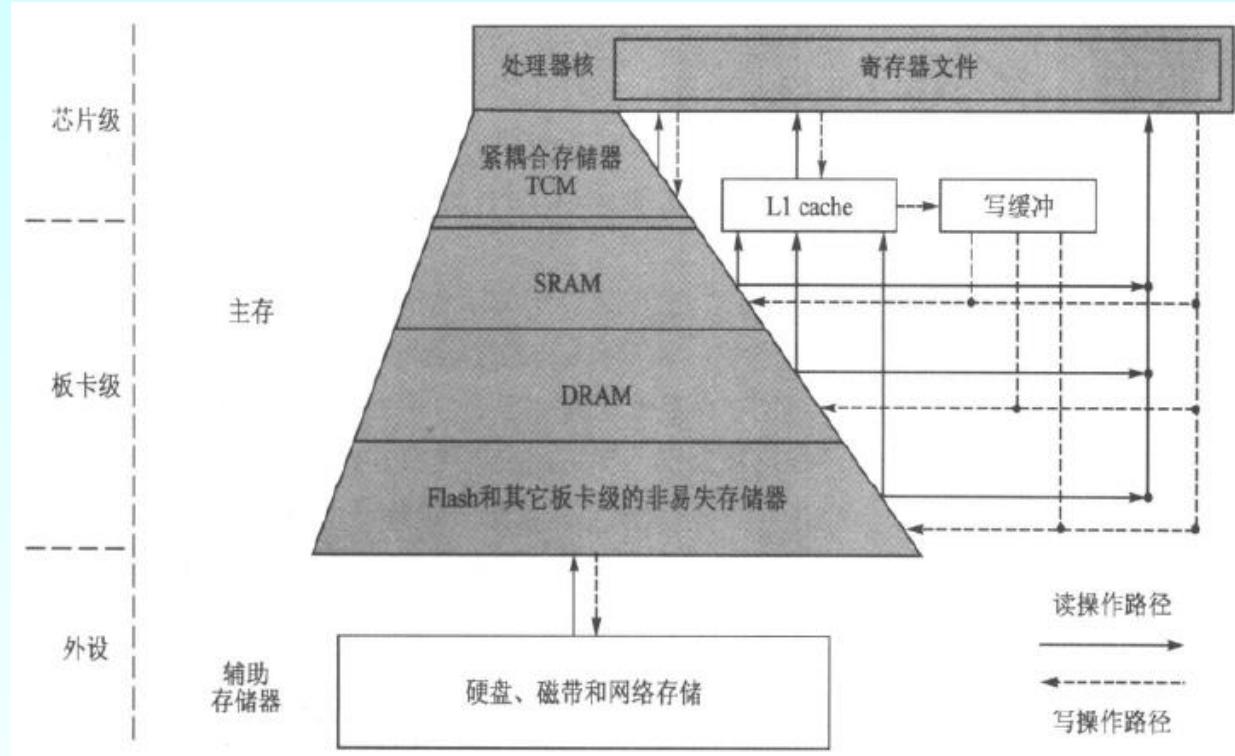
CPU取操作数和保存运算结果：1000MW/s

各种输入输出设备访问存储器：50MW/s

三项相加，要求存储器的频带宽度不低于
1550MW/s, 访问周期不大于**0.64ns**, 实际上
目前主存储器的工作周期为**100ns**左右，两者
相差**150**多倍。

4.1 存储体系概念和并行存储系统

使用ARM处理器核的嵌入式系统中的存储层次结构



4.1.5 双口RAM

双口**RAM**是指同一个存储器具有两组相互独立的读写控制电路，是一种高速工作的存储器。它有两个独立的端口，分别具有各自的地址线、数据线和控制线，可以对存储器中任何位置上的数据进行独立的存取操作。

双口**RAM**的核心部分是用于数据存储的存储器阵列，可为左、右两个端口所共用。当两个端口的地址不相同时，在两个端口上进行读写操作，一定不会发生冲突。当任一端口被选中驱动时，就可对整个存储器进行存取，每一个端口都有自己的片选控制和输出驱动控制。

4.1 存储体系概念和并行存储系统

当两个端口同时存取存储器的同一存储单元时，就会因数据冲突造成数据存储或读取错误。两个端口对同一主存操作有**4**种情况：

- ① 两个端口不同时对同一地址单元存取数据；
- ② 两个端口同时对同一地址单元读出数据；
- ③ 两个端口同时对同一地址单元写入数据；
- ④ 两个端口同时对同一地址单元，一个写入数据，另一个读出数据。

在第①、第②种情况时，两个端口的存取不会出现错误，第③种情况会出现写入错误，第④种情况会出现读出错误。为避免第③、④种错误情况的出现，双口**RAM**设计有硬件“**BUSY**”功能输出，其工作原理如下：当左、右端口不对同一地址单元存取时， $\overline{\text{BUSY}_R} = H$ ， $\overline{\text{BUSY}_L} = H$ ，可正常存储。

4.1 存储体系概念和并行存储系统

当左、右端口对同一地址单元存取时，有一个端口的 $\overline{\text{BUSY}} = \text{L}$ ，禁止数据的存取。此时，两个端口中，哪个存取请求信号出现在前，则其对应的 $\overline{\text{BUSY}} = \text{H}$ ，允许存取；哪个存取请求信号出现在后，则其对应的 $\overline{\text{BUSY}} = \text{L}$ ，禁止其写入数据。需要注意的是，两端口间的存取请求信号出现时间要相差在 **5ns** 以上，否则仲裁逻辑无法判定哪一个端口的存取请求信号在前；在无法判定哪个端口先出现存取请求信号时，两根控制线不会同时为低电平。这样，就能保证对应于 $\overline{\text{BUSY}} = \text{H}$ 的端口能进行正常存取，对应于 $\overline{\text{BUSY}} = \text{L}$ 的端口不存取，从而避免双端口存取出现错误。

4.2 虚拟存储系统

虚拟存储概念**1961**年由英国曼彻斯特大学**Kilbrn**等人提出。

70年代广泛地应用于大中型计算机系统中，目前许多微型机也开始使用虚拟存储系统。

虚拟存储系统由主存储器和联机工作的外存储器（磁盘存储器）共同组成。

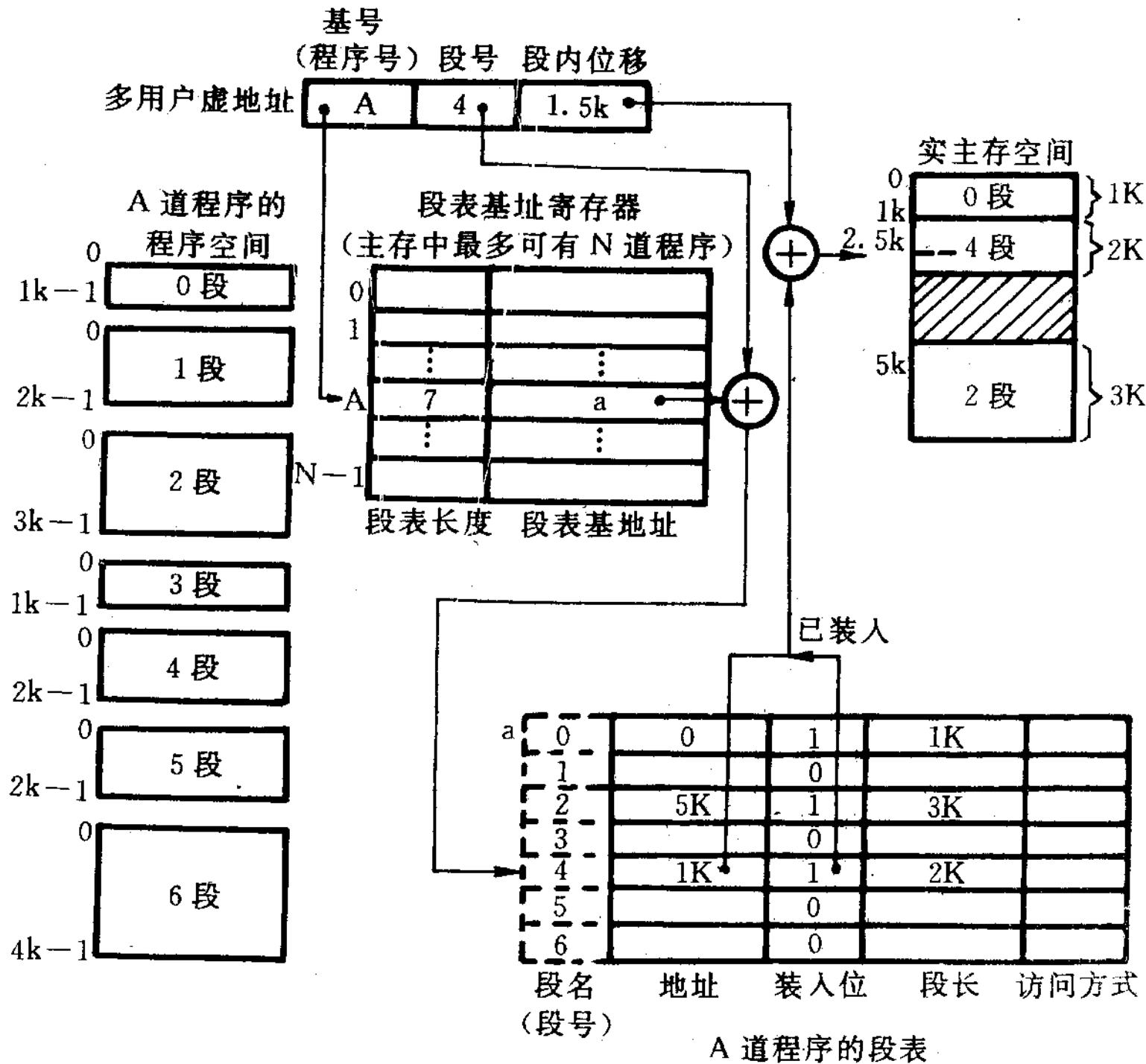
4.2.1 虚拟存储管理方式

1. 段式管理

特点：将程序按逻辑意义分成段，按段进行调入、调出和管理。各个段的长度因程序而异。

地址映像方法：每个程序段都从**0**地址开始编址，长度可长可短，可以在程序执行过程中动态改变程序段的长度。每个程序段可以映像到主存的任意位置，可以连续存放，也可以不连续存放，可以顺序存放，也可以前后倒置。

4.2 虚

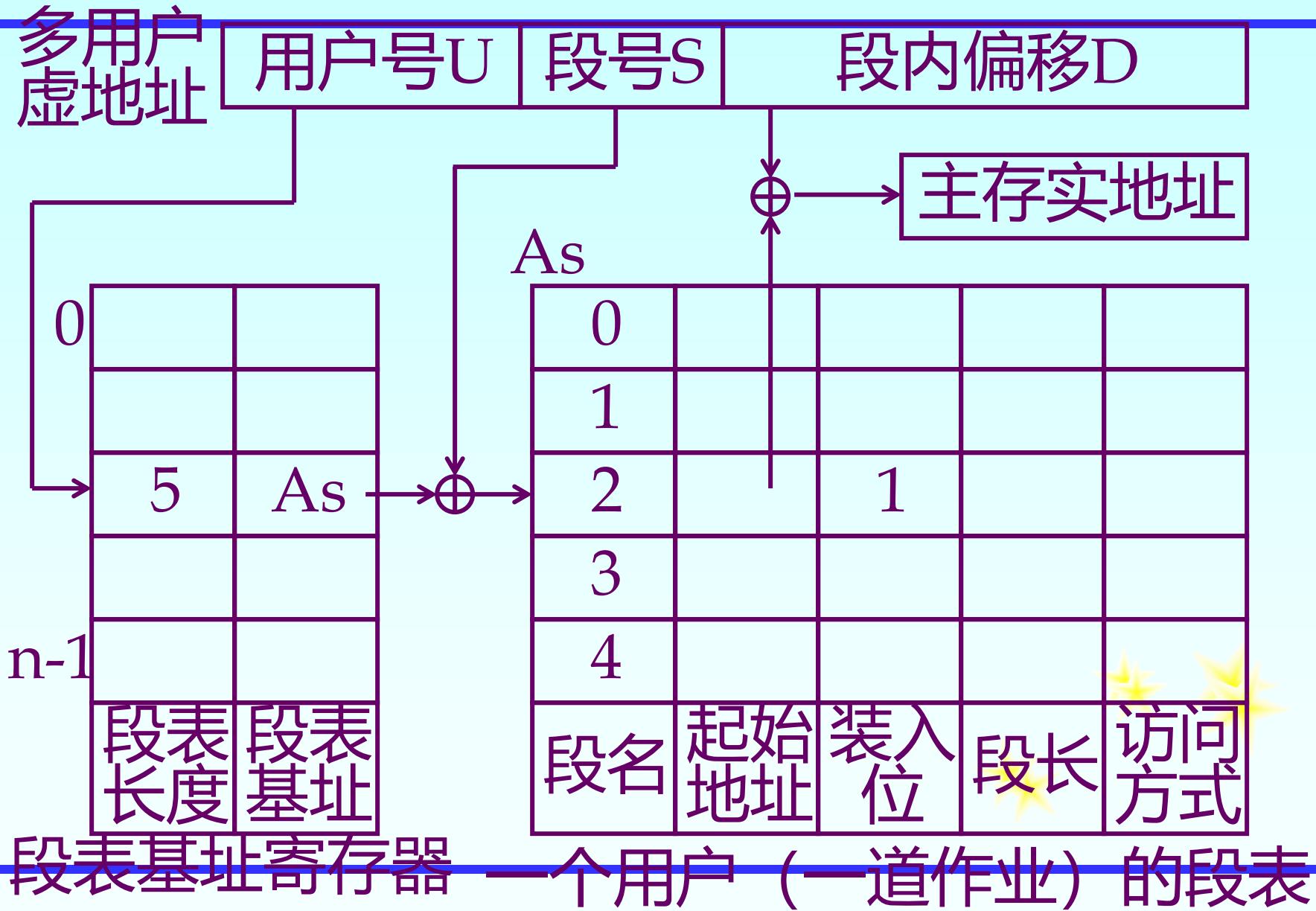


地址变换方法：

- 由用户号找到基址寄存器。
- 从基址寄存器中读出段表的起始地址。
- 把起始地址与多用户虚地址中段号相加得到段表地址。
- 把段表中给出的起始地址与段内偏移D相加就能得到主存实地址。



4.2 虚拟存储系统



4.2 虚拟存储系统

为了把程序虚地址变换成主存实地址，需要一个段表。段表中每一行记录了某个段对应的若干信息，包括段名（段号）、装入位、段起点、段长和访问方式等。装入位为“1”，表示该段已调入主存；装入位为“0”，则表示该段不在主存中。由于段的大小可变，所以在段表中要给出各段的起始地址与段的长度。段表本身也是一个段，一般驻留在主存中。

如果系统有N道程序，就有N个段表。用N个段表基址寄存器分别记录各道程序的段表在主存中的起始地址。

段式虚拟存储系统的主要优点:

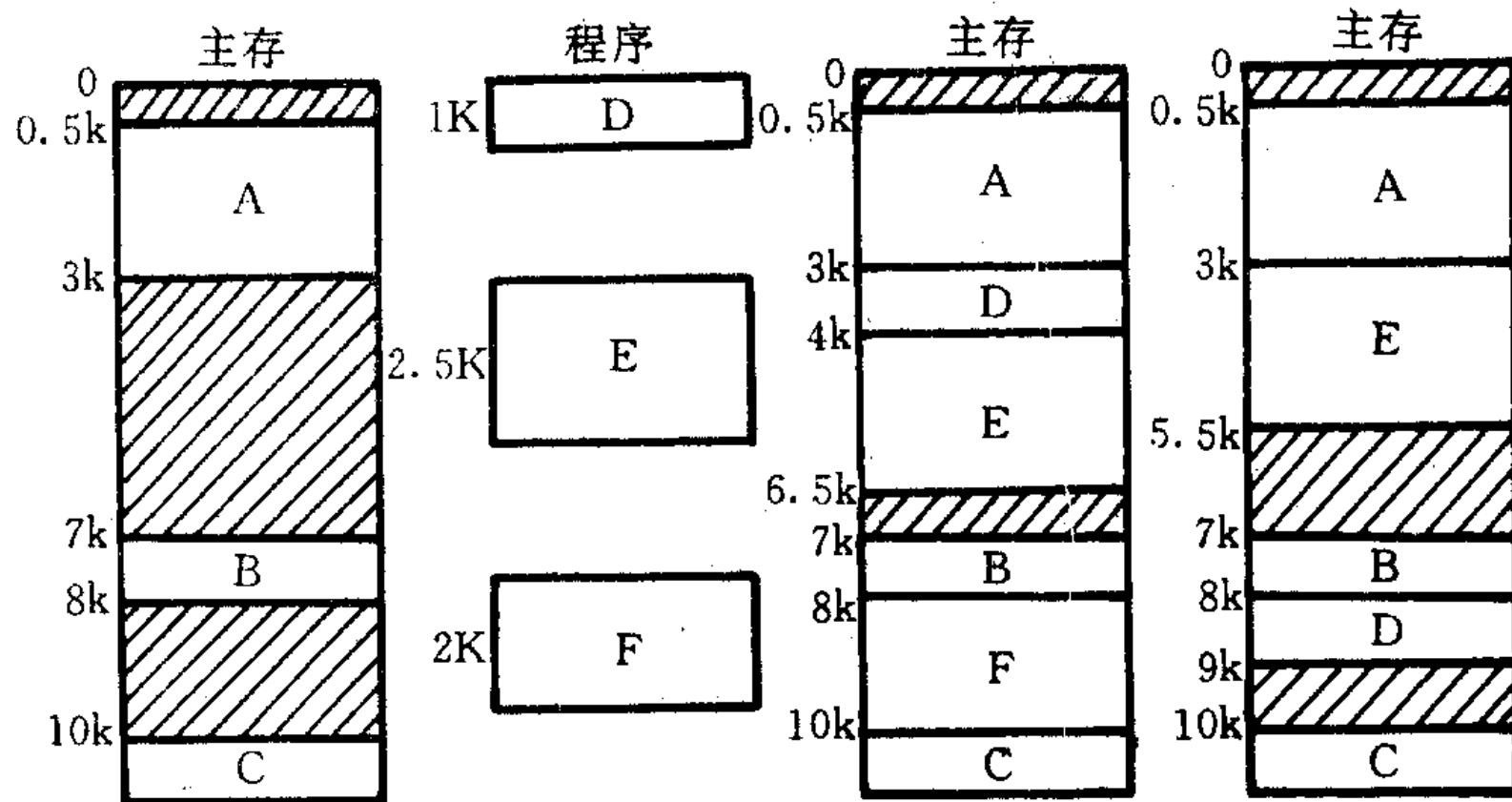
- (1) 程序的模块化性能好
- (2) 便于程序和数据的共享
- (3) 程序的动态链接和调度比较容易
- (4) 便于实现信息保护

段式虚拟存储系统的主要缺点:

- (1) 地址变换所花费的时间比较长，做两次加法运算
- (2) 主存储器的利用率往往比较低
- (3) 对辅存（磁盘存储器）的管理比较困难



4.2 虚拟存储系统



(a) 需依次调入
D、E、F 段

(b) 首先分配法
D、E、F 全被装入

(c) 最佳分配法
F 段无法装入

在图中，首先分配法优于最佳分配法，但不意味着首先分配法就一定好，完全可能出现相反的情况。

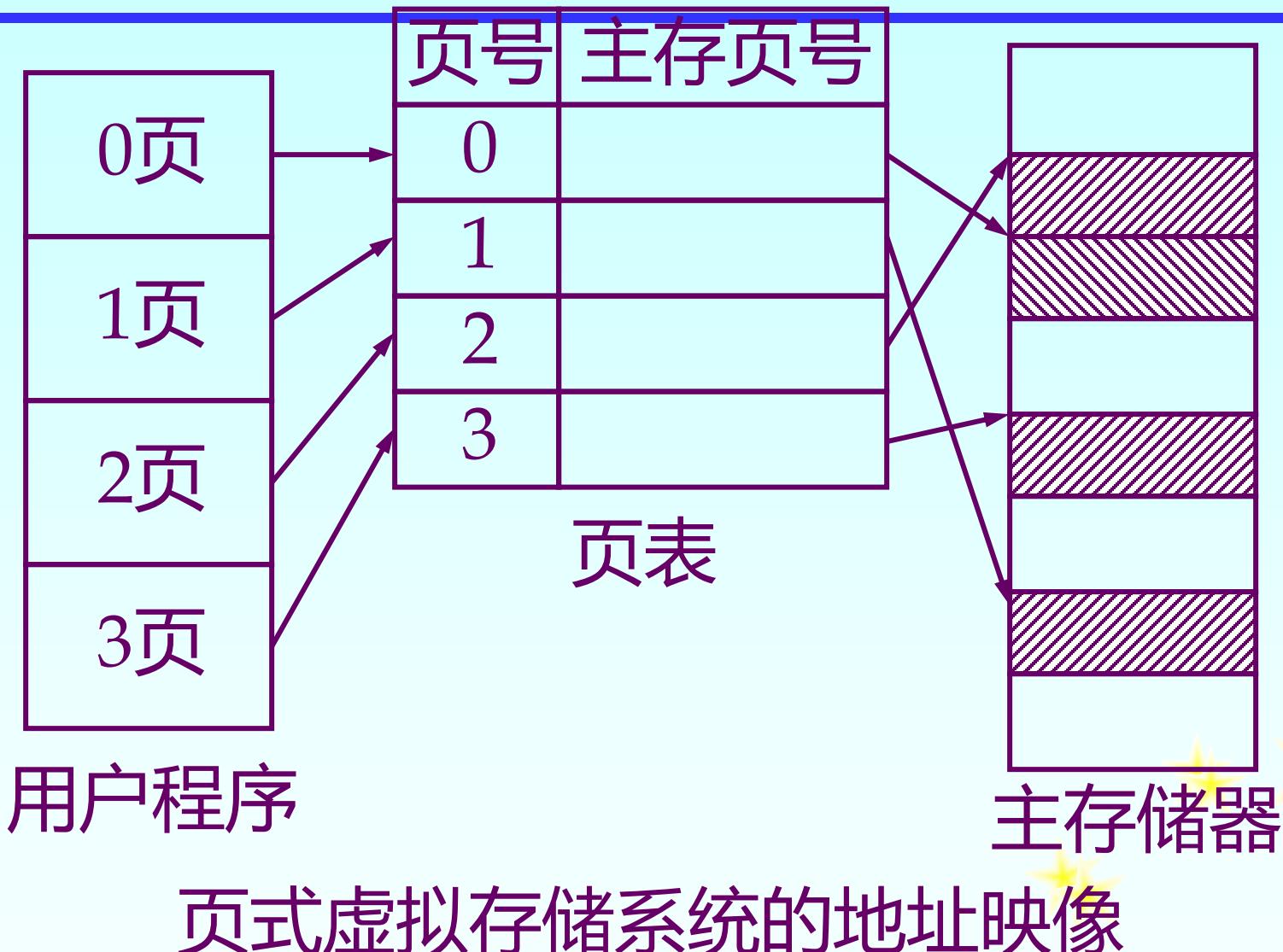


2. 页式管理

页式存储管理是将主存空间和程序空间都分成固定大小的页面（页面的大小随机器而异，一般为**512B**到几**KB**），让程序的起点必须处在主存中某一个页面位置的起点上。

主存即实存的页称为实页，虚存的页称为虚页，由地址映像机构将虚页号转换成主存的实际页号。

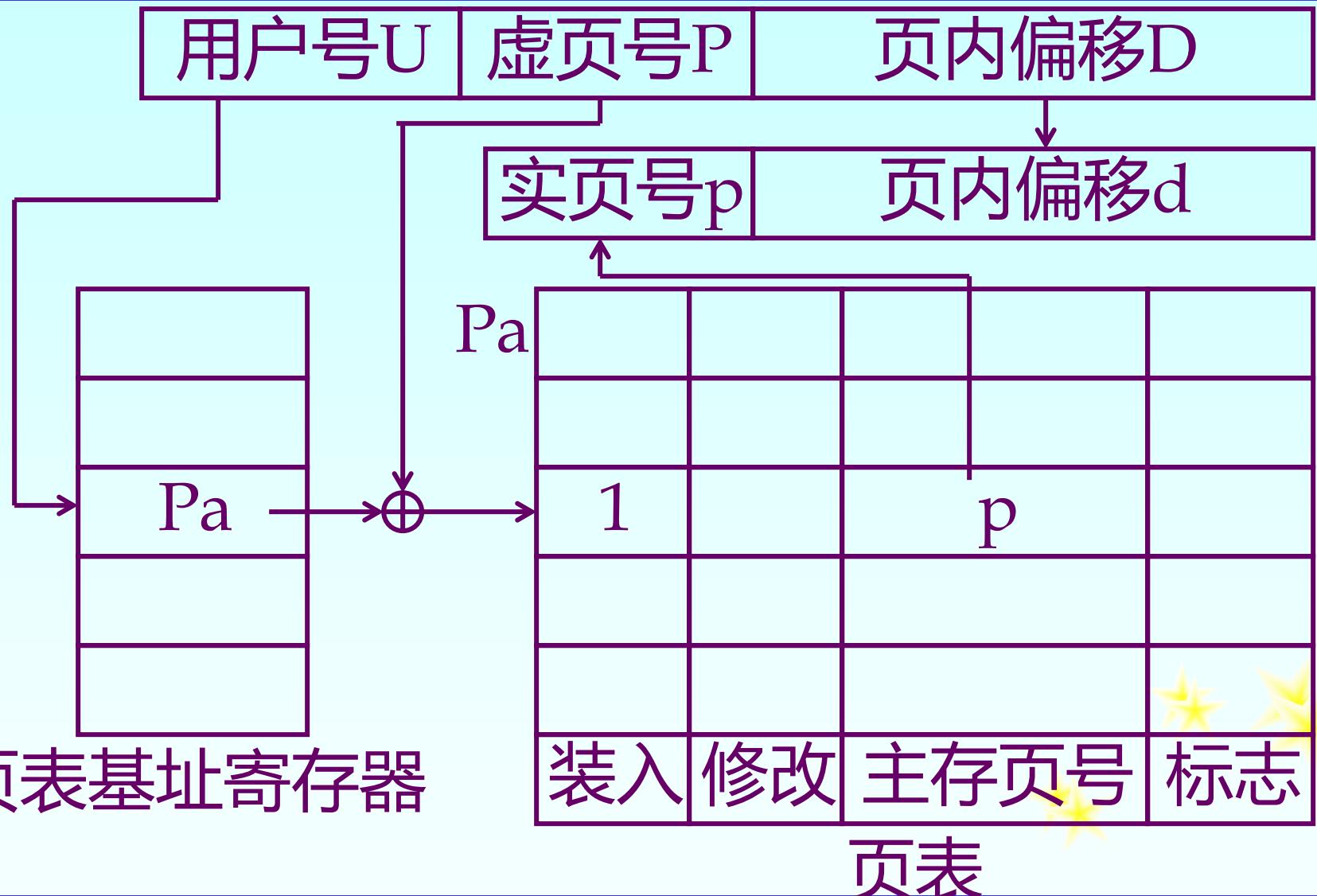
4.2 虚拟存储系统



页式管理需要一个页表。页表是一张存放在主存中的虚页号和实页号的对照表，页表中每一行记录了某个虚页对应的若干信息，包括虚页号、装入位和实页号等。若装入位为“1”，表示该页面已在主存中，将对应的实页号与虚地址中的页内地址相拼接就得到了完整的实地址；若装入位为“0”，表示该页面不在主存中。

与段式存储管理一样，也要配备N个页表基址寄存器，来存放N道程序各自所用页表在主存中的起始地址。

4.2 虚拟存储系统



北京理工大学计算机学院

页式虚拟存储系统主要优点：

- (1) 主存储器的利用率比较高
- (2) 页表相对比较简单
- (3) 地址变换的速度比较快
- (4) 对磁盘的管理比较容易

页式虚拟存储系统主要缺点：

- (1) 程序的模块化性能不好
- (2) 页表很长，需要占用很大的存储空间。

例如：虚拟存储空间**4GB**，页大小**1KB**，则
页表的容量为**4M字**，**16MB**。

3. 段页式管理

在段式、页式存储器的基础上，还有一种段页式虚拟存储系统。将程序按其逻辑结构分段，每段再划分为若干大小相等的页；主存空间也划分为若干同样大小的页。虚存和实存之间以页为基本传送单位。

段的长度必须是页长的整数倍，段的起点必须是某一页的起点。用户按照程序段来编写程序，每个程序段分成几个固定大小的页。

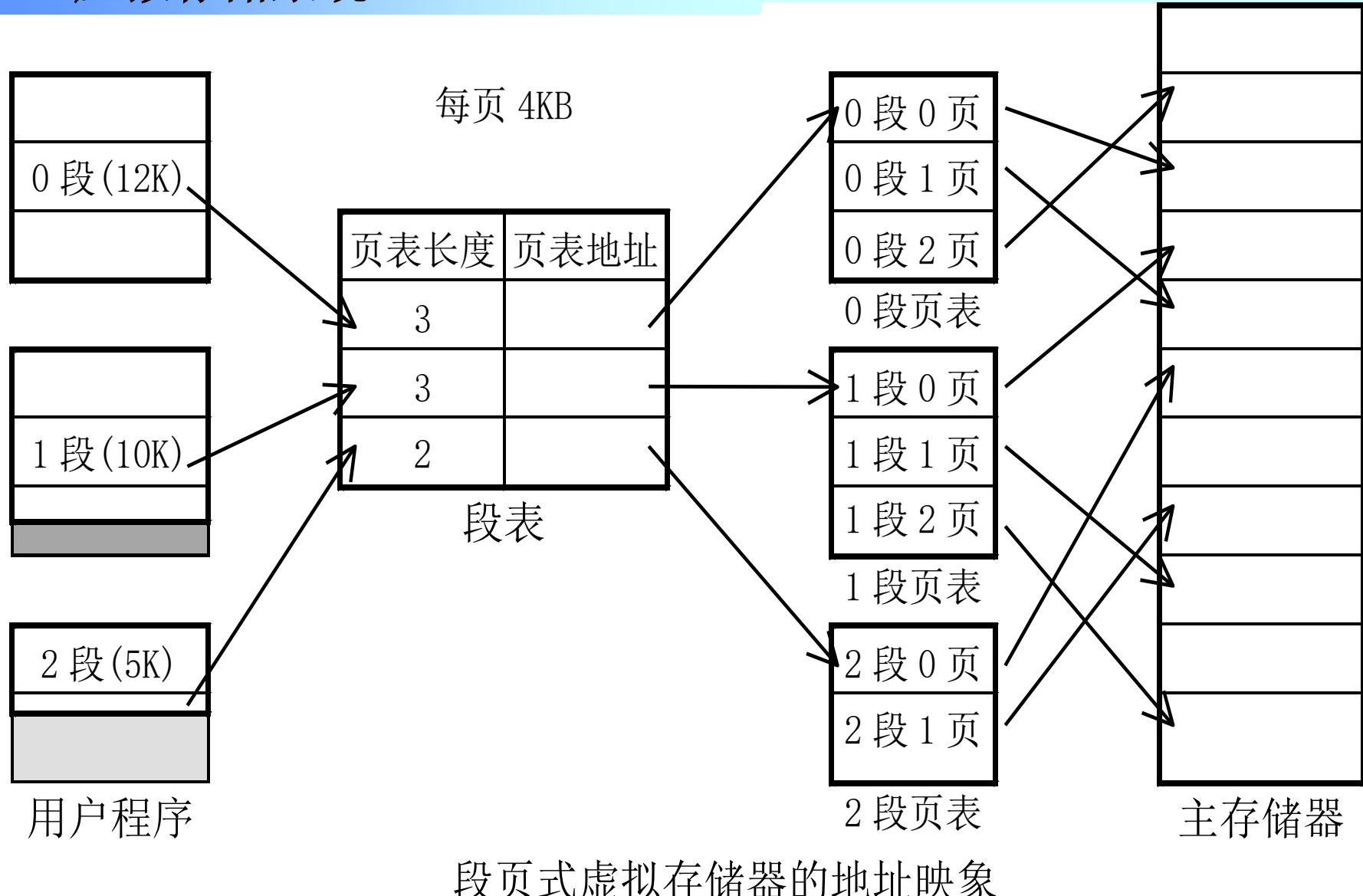
每道程序需要一张段表，多张页表。

对用户原来编写程序的虚拟存储空间采用分段的方法管理，而对主存的物理空间采用分页方法管理。

例如，一个用户程序由**3**个独立的程序段组成。一张段表，**3**张页表。段表中给出该程序段的页表长度和页表的起始地址。页表中给出了每一页在主存中的实页号。



4.2 虚拟存储系统



CPU访问时，虚地址包含用户号、段号、段内页号、页内地址四部分。地址变换方法：

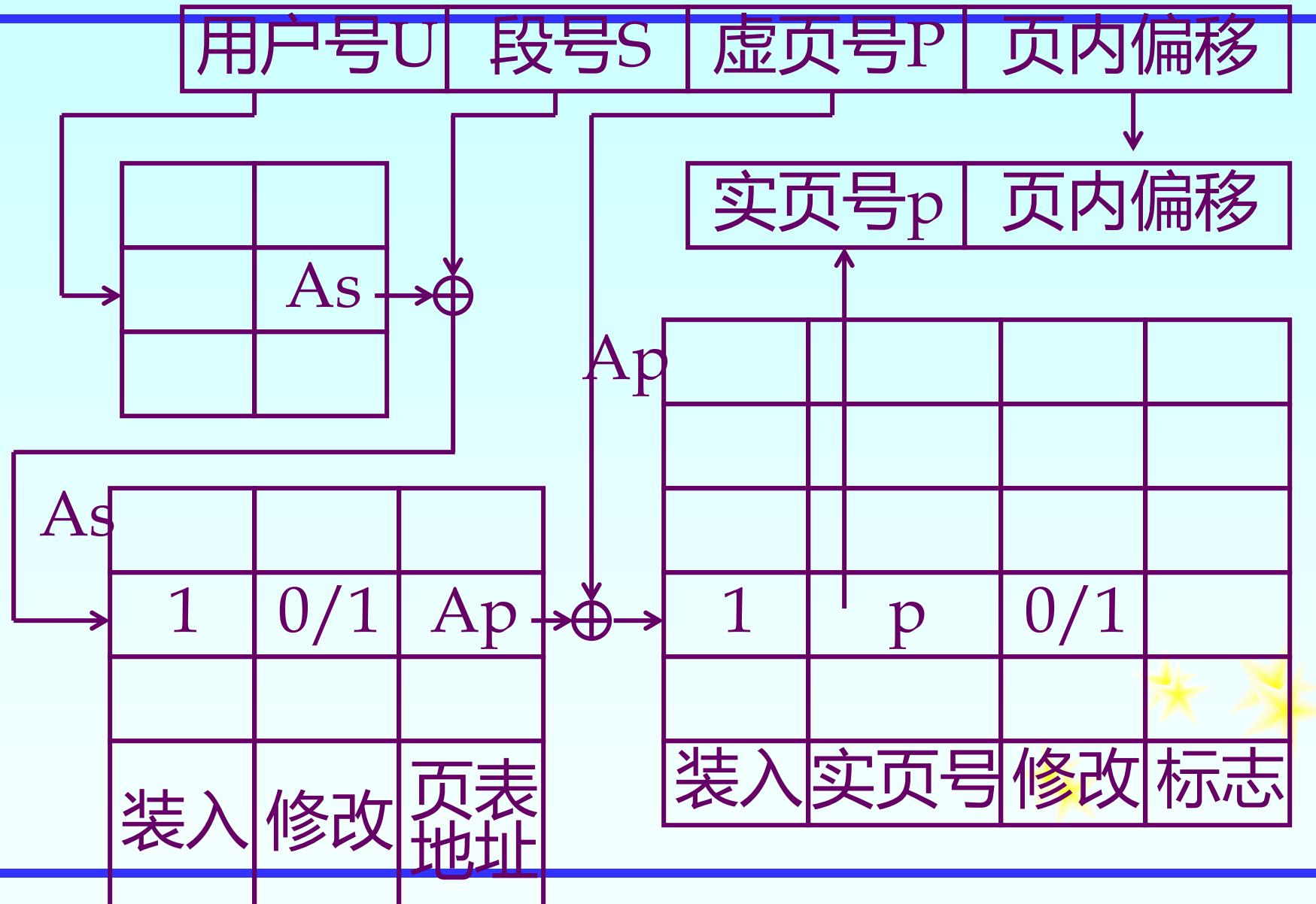
- (1) 由用户号找到段表基址寄存器，
- (2) 将段表起始地址与段号合成，得到段表地址，查段表，得到该程序段的页表起始地址和页表长度，
- (3) 将页表起始地址，与段内页号合成，得到页表地址，再查页表找到要访问的主存实页号，

(4) 最后把实页号 p 与页内偏移 d 拼接得到主存的实地址。

段页式存储器综合了前两种结构的优点，但要经过两级查表才能完成地址转换，费时要多些。



4.2 虚拟存储系统



4.2 虚拟存储系统

造成虚拟存储系统速度降低的主要原因：

- (1) 要访问主存储器须先查段表或页表
- (2) 可能需要多级页表

页表级数的计算公式：

其中：

Np为页面的大小

Nv为虚拟存储空间大小

Nd为一个页表存储字的大小

$$g = \left\lceil \frac{\log_2 Nv - \log_2 Np}{\log_2 Np - \log_2 Nd} \right\rceil$$



例7：虚拟存储空间大小**Nv=4GB**，
页的大小**Np=1KB**，每个页表存储字占用
4个字节。整个页表共有**4M**个表项，远大
于一个页面，所以需要建立多级页表。计
算得到页表的级数：

$$g = \left\lceil \frac{\log_2 4G - \log_2 1K}{\log_2 1K - \log_2 4} \right\rceil = \left\lceil \frac{32 - 10}{10 - 2} \right\rceil = 3$$

3级页表：**256×256×64=4M字**

通常把**1**级页表驻留在主存储器中，**2、3**级页表只驻留一小部分在主存。

4.2 虚拟存储系统

4.2.2 页式虚拟存储系统构成

1. 地址的映像和变换

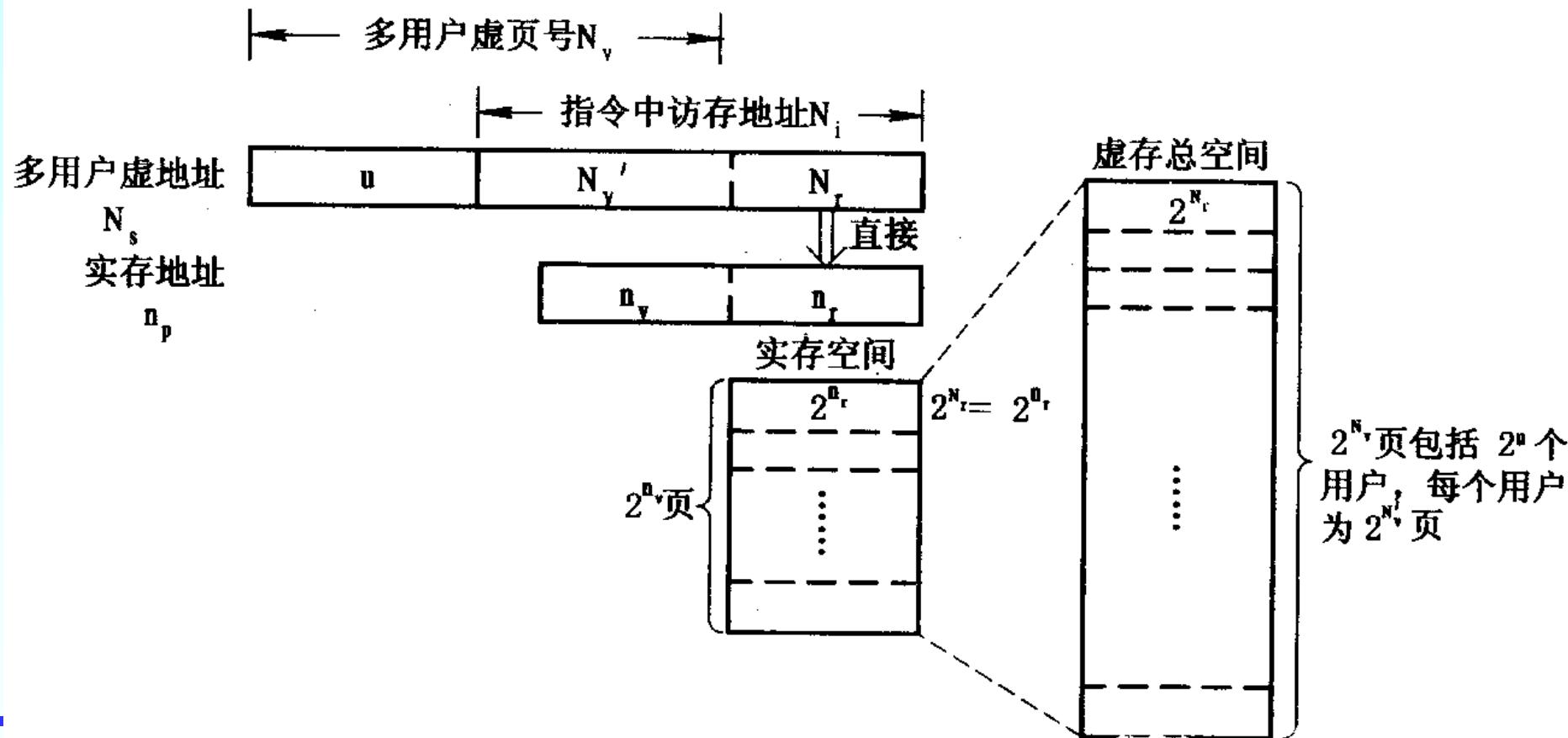


图 4.15 虚实地址对应关系及空间的压缩

4.2 虚拟存储系统

所谓地址的映像是将每个虚存单元按某种规则(算法)装入(定位于)实存，即建立多用户虚地址 N_s 与实主存地址 n_p 之间的对应关系。对于页式而言，实际上就是将多用户虚页号 N_v 的页可以装入主存中的哪些页面位置，建立起 N_v 与 n_p 的对应关系。而地址的变换则指的是程序按照这种映像关系装入实存后，在执行时，多用户虚地址 N_s 如何变换成对应的实地址 n_p 。对页式而言就是多用户虚页号 N_v 如何变换成实页号 n_v 。



4.2 虚拟存储系统

一个**主存地址A**由两部分组成，实页号**p**和页内偏移**d**。

实页号p	页内偏移d
------	-------

主存地址A的组成

一个**虚地址Av**由三部分组成，用户号**U**、虚页号**P**和页内偏移**D**。

用户号U	虚页号P	页内偏移D
------	------	-------

多用户虚拟地址**Av**的组成

内部地址变换：

虚页号变换成主存实页号，进而多用户虚拟地址**A_v**变换成主存实地址**A**。

多用户虚拟地址中的页内偏移**D**直接作为主存实地址中的页内偏移**d**。主存实页号**p**与它的页内偏移**d**直接拼接起来就得到主存实地址**A**。



外部地址变换：

虚页号变换成磁盘实地址，主要由软件实现。

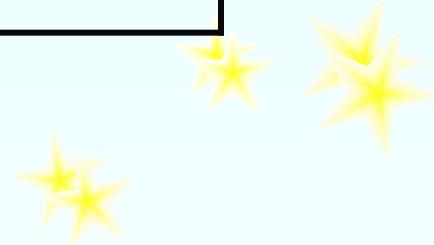
首先通过查外页表得到磁盘实地址，然后再查主存实页表，看主存是否有空页。若有空页，把磁盘存储器实地址和主存储器实页号送入输入输出处理机。在输入输出处理机的控制下，把要访问数据所在的一整页都从磁盘存储器调入到主存储器。



4.2 虚拟存储系统

要想把该道程序的虚页调入主存，必须给出该页在辅存中的实际地址。为了提高调页效率，辅存一般是按信息块编址的，而且让块的大小通常等于页面的大小。以磁盘为例，辅存实(块)地址 N_{vd} 的格式为

N_{vd}	磁盘机号	柱面号	磁头号	块号
----------	------	-----	-----	----



4.2 虚拟存储系统

内部地址变换失败（未命中—页面失效），要进行外部地址变换，其目的是要找到磁盘的实地址，并把需要的那一页调入主存。

在操作系统中，把页面失效当作一种异常故障来处理。

每个用户程序都有一张外页表，虚拟地址空间中的每一页，在外页表中都有对应的一个存储字。

每一个存储字除了磁盘存储器的地址之外，至少还包括一个装入位。

4.2 虚拟存储系统

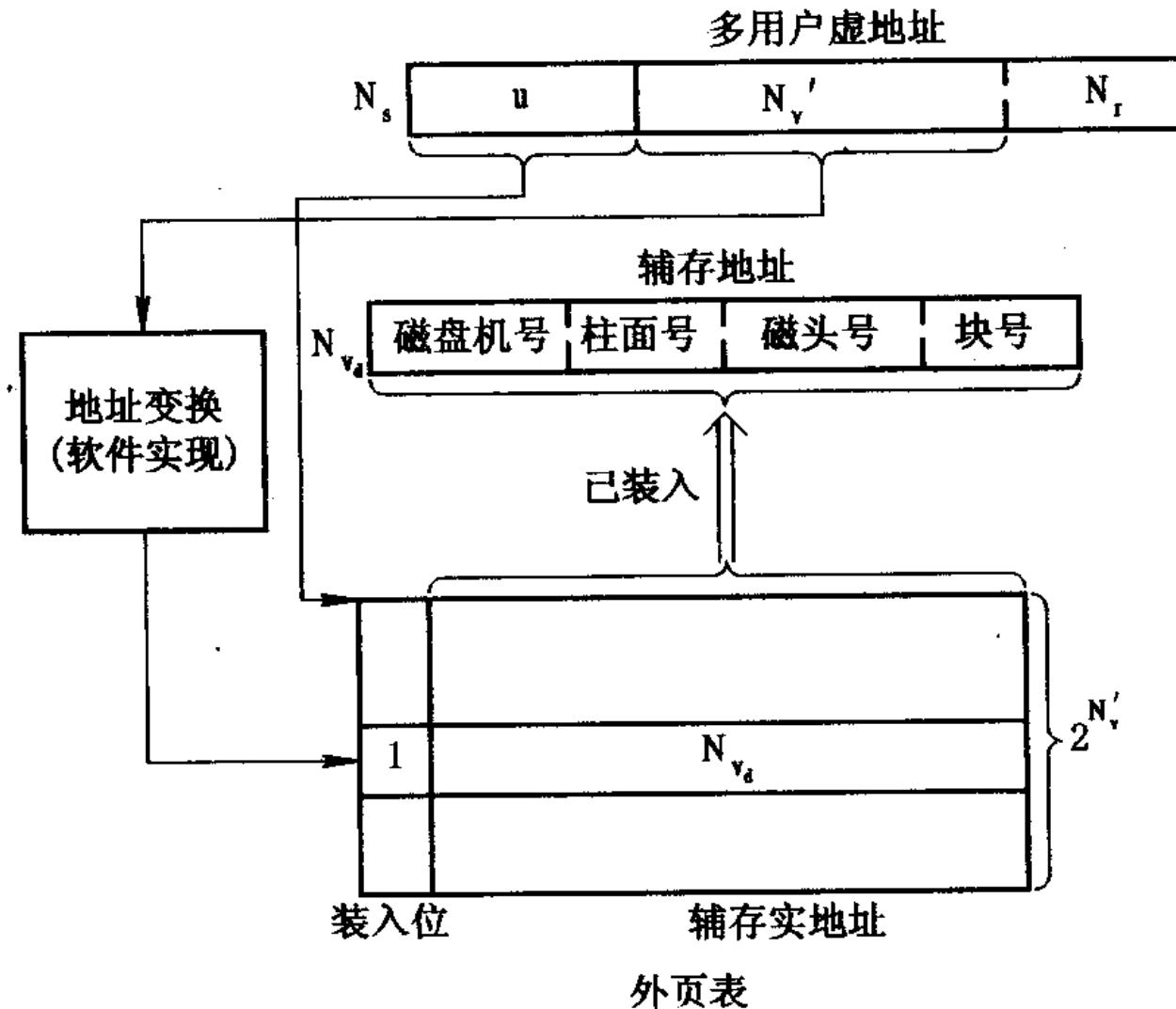


图 4.18 虚地址到辅存实地址的变换
北京理工大学计算机学院

2. 替换算法

替换算法的确定主要是看按这种替换算法替换是否有高的主存命中率，其次要看算法是否便于实现，辅助软、硬件成本是否低。到目前为止，已研究过各种替换算法，如随机法、先进先出法和近期最少使用(近期最久未用过)法等。

页面替换发生时间：

当发生页面失效时，要从磁盘中调入一页到主存。如果主存所有页面都已经被占用，必须从主存储器中淘汰掉一个不常使用的页面，以便腾出主存空间来存放新调入的页面。

评价页面替换算法好坏的标准：

- 一是命中率要高
- 二是算法要容易实现



页面替换算法的使用场合：

- (1) 虚拟存储系统中，主存页面的替换，一般用软件实现
- (2) Cache块替换一般用硬件实现
- (3) 虚拟存储系统的快慢表中，快表存储字的替换，用硬件实现
- (4) 虚拟存储系统中，用户基地址寄存器的替换，用硬件实现
- (5) 在有些虚拟存储系统中目录表的替换

页面失效

该页未装入主存，需从辅存中调页。

页面冲突（页面争用）

两个以上的虚页想要进入主存中同一个页面位置的现象。

页面失效时不一定发生页面冲突，但页面冲突一定是由页面失效引起的。



1、页面替换算法

(1) 随机算法(**RAND** Random algorithm):

算法简单，容易实现；没有利用历史信息，没有反映程序的局部性，命中率低。

(2) 先进先出算法 (**FIFO** First-In First-Out algorithm):

比较容易实现，利用了历史信息，没有反映程序的局部性。最先调入主存的页面，很可能也是经常要使用的页面。

(3) 近期最少使用算法 (**LFU Least Frequently Used algorithm**):

既充分利用了历史信息，又反映了程序的局部性，实现起来非常困难。

(4) 最久没有使用算法 (**LRU Least Recently Used algorithm**):

把**LFU**算法中的“多”与“少”简化成“有”与“无”，实现起来比较容易。



(5) 最优替换算法 (**OPT** OPTimal replacement algorithm):

是一种理想化的算法。用来作为评价其它页面替换算法好坏的标准。

在虚拟存储系统中，实际上有可能采用的只有**FIFO**和**LRU**两种算法。



4.2 虚拟存储系统

替换算法一般是通过用典型的页地址流模拟其替换过程，再根据所得到的命中率的高低来评价其好坏的。当然影响命中率的因素除了替换算法外，还因地址流、页面大小、主存容量等不同而不同。

设有一道程序，有1至5共5页，执行时的页地址流(即执行时依次用到的程序页页号)为：

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



4.2 虚拟存储系统

时间t	1	2	3	4	5	6	7	8	9	10	11	12
页地址流	2	3	2	1	5	2	4	5	3	2	5	2
先进先出 FIFO	2	2	2	2*	5	5	5*	5*	3	3	3	3*
命中3次	调进	调进	命中	调进	替换	替换	替换	命中	替换	命中	替换	替换
近期最少使用 LRU	2	2	2	2	2*	2	2	2*	3	3	3*	3*
命中5次	调进	调进	命中	调进	替换	命中	替换	命中	替换	命中	命中	命中
优化 OPT	2	2	2	2	2	2*	4*	4*	2	2	2	2
命中6次	调进	调进	命中	调进	替换	命中	替换	命中	命中	命中	命中	命中

图 4.20 3种替换算法对同一计算页地址流的替换过程

4.2 虚拟存储系统

由图4.20可见，**FIFO**的命中率最低，而**LRU**的命中率非常接近于**OPT**。

命中率也与页地址流有关。例如一个循环程序，当所需页数大于分配给它的页数时，无论**FIFO**还是**LRU**的命中率都明显低于**OPT**。图4.21，在**FIFO**和**LRU**算法中，总是发生下次就要使用的页面本次被替换出去的情况，这就是“颠簸”现象。



4.2 虚拟存储系统

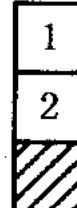
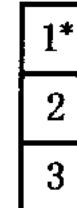
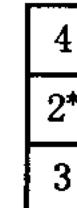
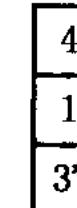
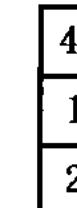
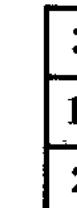
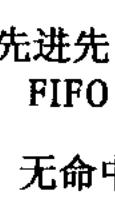
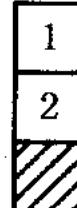
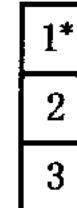
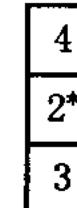
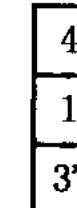
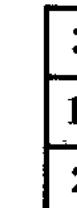
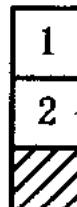
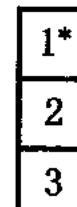
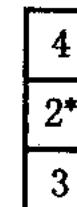
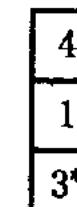
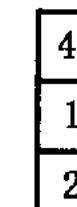
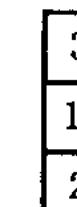
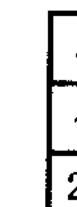
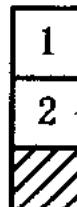
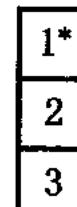
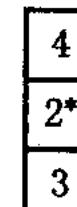
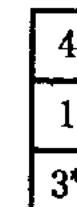
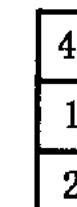
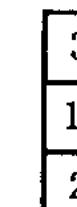
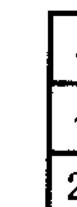
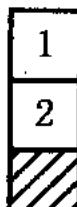
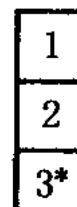
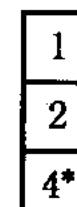
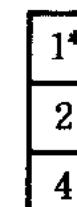
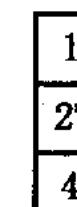
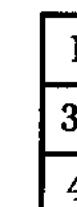
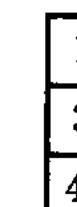
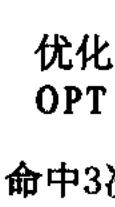
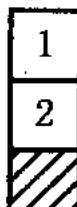
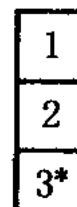
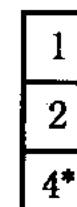
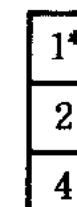
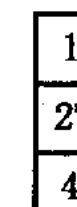
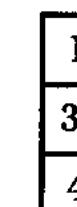
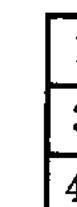
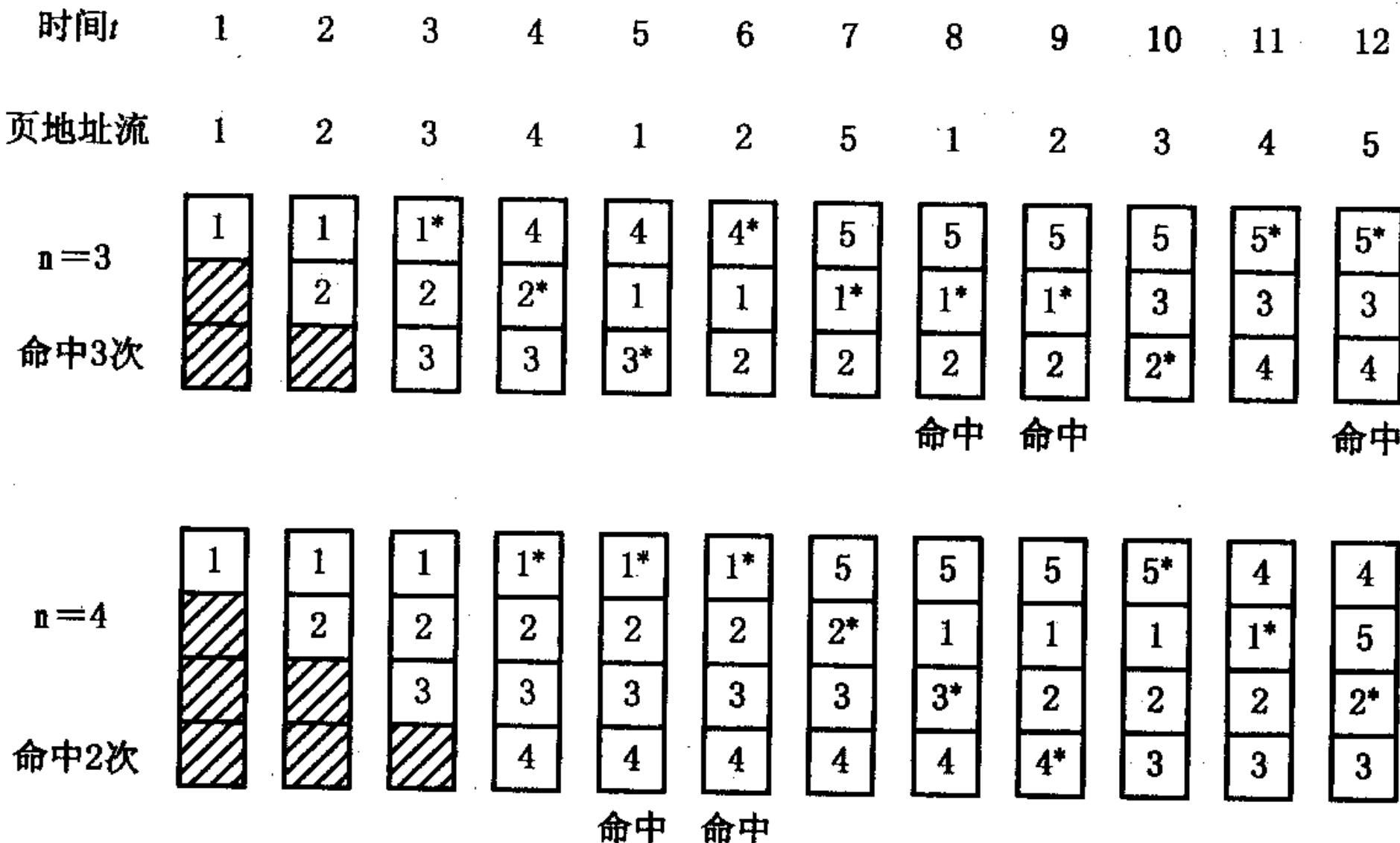
时间t	1	2	3	4	5	6	7	8
页地址流	1	2	3	4	1	2	3	4
先进先出 FIFO								
无命中								
近期最少使用 LRU								
无命中								
优化 OPT								
命中3次								
					命中	命中	命中	命中

图 4.21 北京理工大学页地址流有关

命中率还与分配给该道程序的实页数有关。一般来说，随着分配给程序的主存实页数越多，虚页装入主存的机会越多，但命中率是否提高与使用的替换算法还有关，如**FIFO**算法的实页数增加，命中率还有可能下降；而堆栈型算法，随着分配给程序的主存页面数增加，主存的命中率也提高，至少不下降。



4.2 虚拟存储系统

图 4.22 FIFO法的~~15页数增加~~计算机命中率反而有可能下降

4.2 虚拟存储系统

什么是堆栈型替换算法呢？设A是长度为L的任意一个页面地址流，t为已处理过t-1个页面的时间点，n为分配给该地址流的主存页面数， $B_t(n)$ 表示在t时间点、在n页的主存中的页面集合， L_t 表示到t时间点已遇到过的地址流中相异页的页数。如果替换算法具有下列包含性质：

$$n < L_t \text{ 时}, B_t(n) \subset B_t(n + 1)$$

$$n \geq L_t \text{ 时}, B_t(n) \subset B_t(n + 1)$$

则此替换算法属堆栈型的替换算法。

4.2 虚拟存储系统

LRU算法在主存中保留的是n个最近使用的页面，它们又总是被包含在n+1个最近使用的页面之中，所以LRU是堆栈算法。FIFO算法不具有以上特性。

以图4.22为例，从 t_7 可以看出， $n=3$ ，
 $B_7(3)=\{5,1,2\}$ ， $n=4$ ，
 $B_7(4)=\{5,2,3,4\}$ ， $B_7(3) \not\subseteq B_7(4)$ 。所以，
FIFO法不是堆栈替换算法，



4.2 虚拟存储系统

用堆栈处理技术对地址流进行模拟处理时，主存在 t 时间点的状况用堆栈 S_t 表示。 S_t 是 L_t 个不同页面号在堆栈中的有序集， $S_t(1)$ 是 t 时间点的 S_t 的栈顶项， $S_t(2)$ 是 t 时间点的 S_t 的次栈顶项，依次类推。按照堆栈型算法具有的包含性质，必有

$$n < L_t \text{ 时}, B_t(n) = \{S_t(1), S_t(2), \dots, S_t(n)\}$$

$$n \geq L_t \text{ 时}, B_t(n) = \{S_t(1), S_t(2), \dots, S_t(L_t)\}$$

4.2 虚拟存储系统

对不同的堆栈型替换算法， S_t 各项的改变过程是不同的。例如，LRU算法是把主存中刚访问过的页号置于栈顶，而把最久未被访问过的页号置于栈底。确切地说， t 时间点访问的页 A_t ，若 $A_t \notin S_{t-1}$ ，则把 A_t 压入堆栈使之成为 $S_t(1)$ ，而 $S_{t-1}(1)$ 成为 $S_t(2)$ ， $S_{t-1}(2)$ 成为 $S_t(3)$ ，……，即 S_{t-1} 各项都下推一个位置；若 $A_t \in S_{t-1}$ ，则把它由 S_{t-1} 中取出，压入栈顶成为 $S_t(1)$ ，在 A_t 之下各项的位置不动，而 A_t 之上的各项都下推一个位置。

LRU算法的堆栈实现

堆栈法 容量 2^{nv}

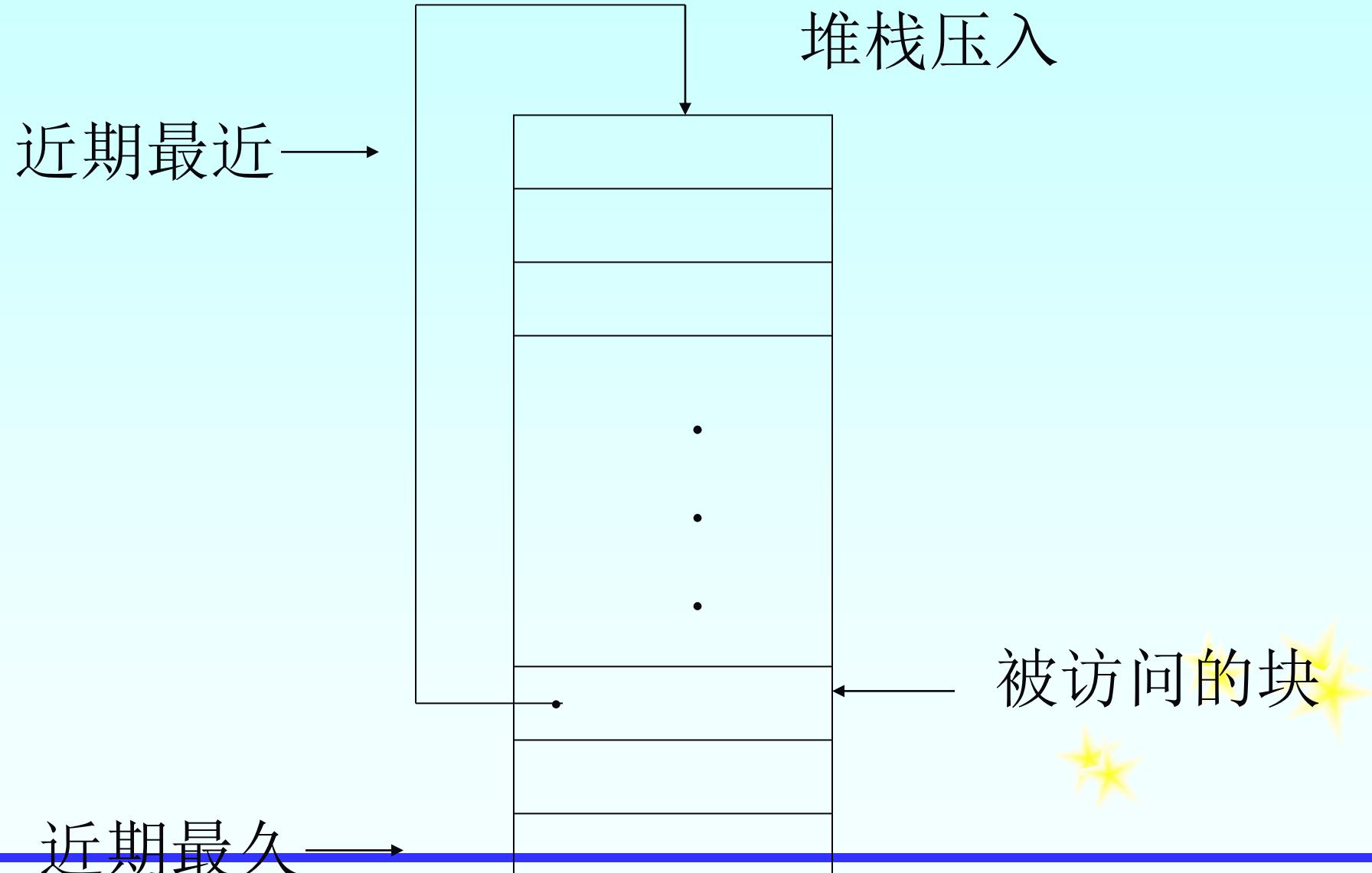
过程：页面是否在堆栈中？

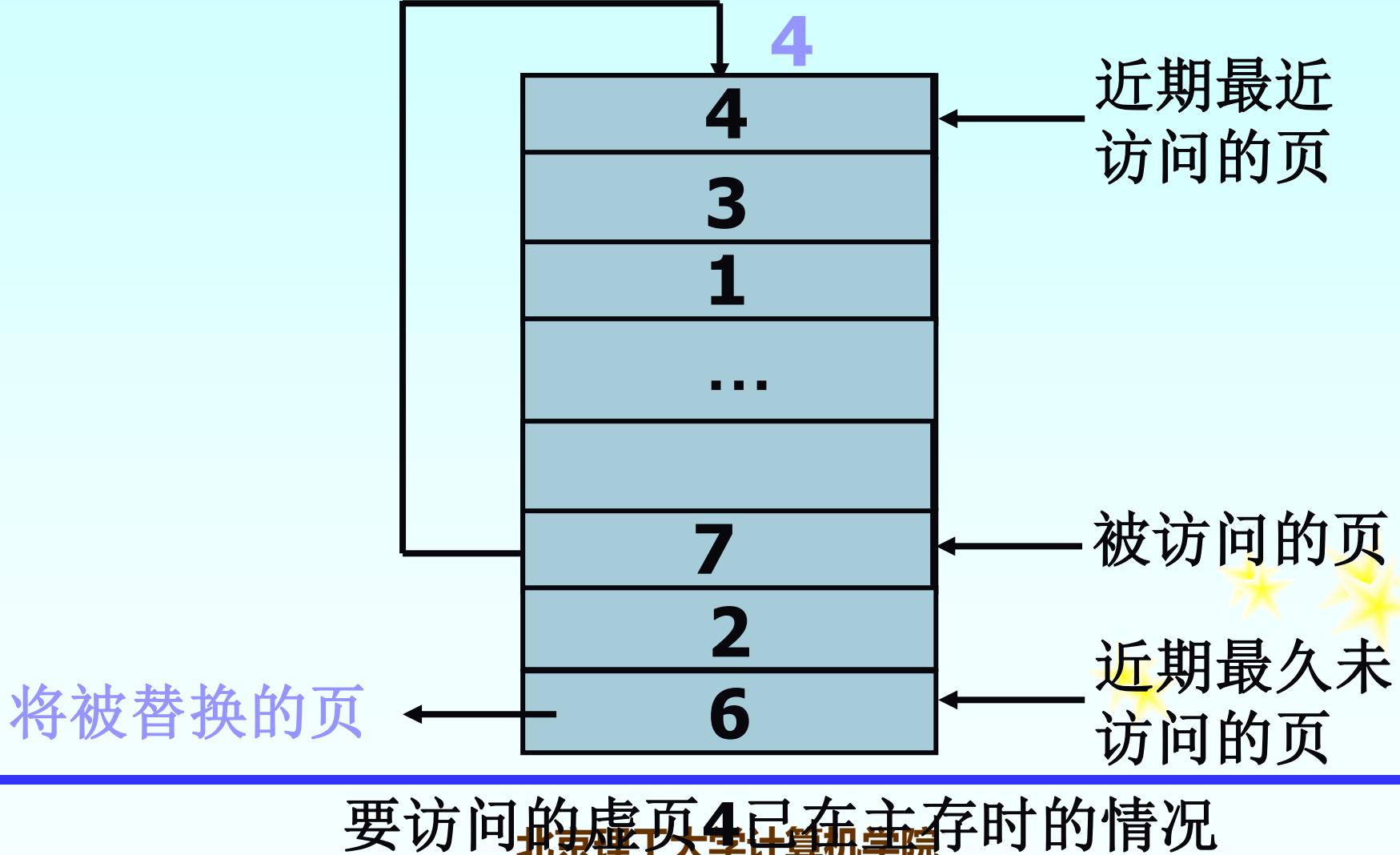
是：将该页面调置栈顶，并把该项上面的项下推一行，该项下面的不动。

否：新页调入堆栈，弹出栈底。



4.2 虚拟存储系统





结果：

栈顶恒为近期最近访问过的页号。

栈底恒为近期最久没有访问过的页号。

堆栈要求：

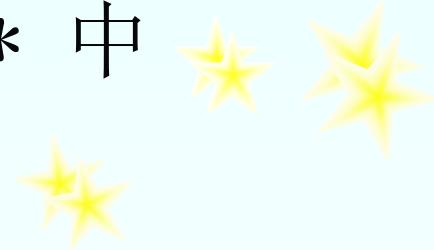
- 有相联比较功能
- 全下推或部分下推功能
- 从中间抽走一项的功能



4.2 虚拟存储系统

1	2	3	4	5	6	7	8	9	10
1	2	1	5	4	1	3	4	2	4
1	2	1	5	4	1	3	4	2	4
	1	2	1	5	4	1	3	4	2
			2	1	5	4	1	3	3

中 * 中 * 中 * 中



4.2 虚拟存储系统

时间t	1	2	3	4	5	6	7	8	9	10	11	12
页地址流A	2	3	2	1	5	2	4	5	3	2	5	2
$S_t(1)$	2	3	2	1	5	2	4	5	3	2	5	2
$S_t(2)$	2	3	2	2	1	5	2	4	5	3	2	5
$S_t(3)$				3	2	1	5	2	4	5	3	3
$S_t(4)$					3	1	1	1	2	4	4	4
$S_t(5)$						3	3	1	1	1	1	
$S_t(6)$												
$n=1$												
$n=2$												
$n=3$												
$n=4$												
$n=5$												

图 使用LRU算法对页地址流进行堆栈处理

4.2 虚拟存储系统

由图的 S_t 可确定对应这个页地址流和主存页数n取不同值时的命中率。只要对不同的n值，当 $A_t \in S_{t-1}$ ，则命中；当 $A_t \notin S_{t-1}$ ，则不命中。例如，对n=4，其 $S_5=\{5, 1, 2, 3\}$ ，因为 $A_6=2 \in S_5$ ，所以命中；但对n=2，其 $S_5=\{5, 1\}$ ，因为 $A_6=2 \notin S_5$ ，所以不命中。这样就可算出各个n值的命中率H*如下所示：

N	1	2	3	4	5	>5
H^*	0.00	0.17	0.42	0.50	0.58	0.58

4.2 虚拟存储系统

3. 虚拟存储系统工作的全过程

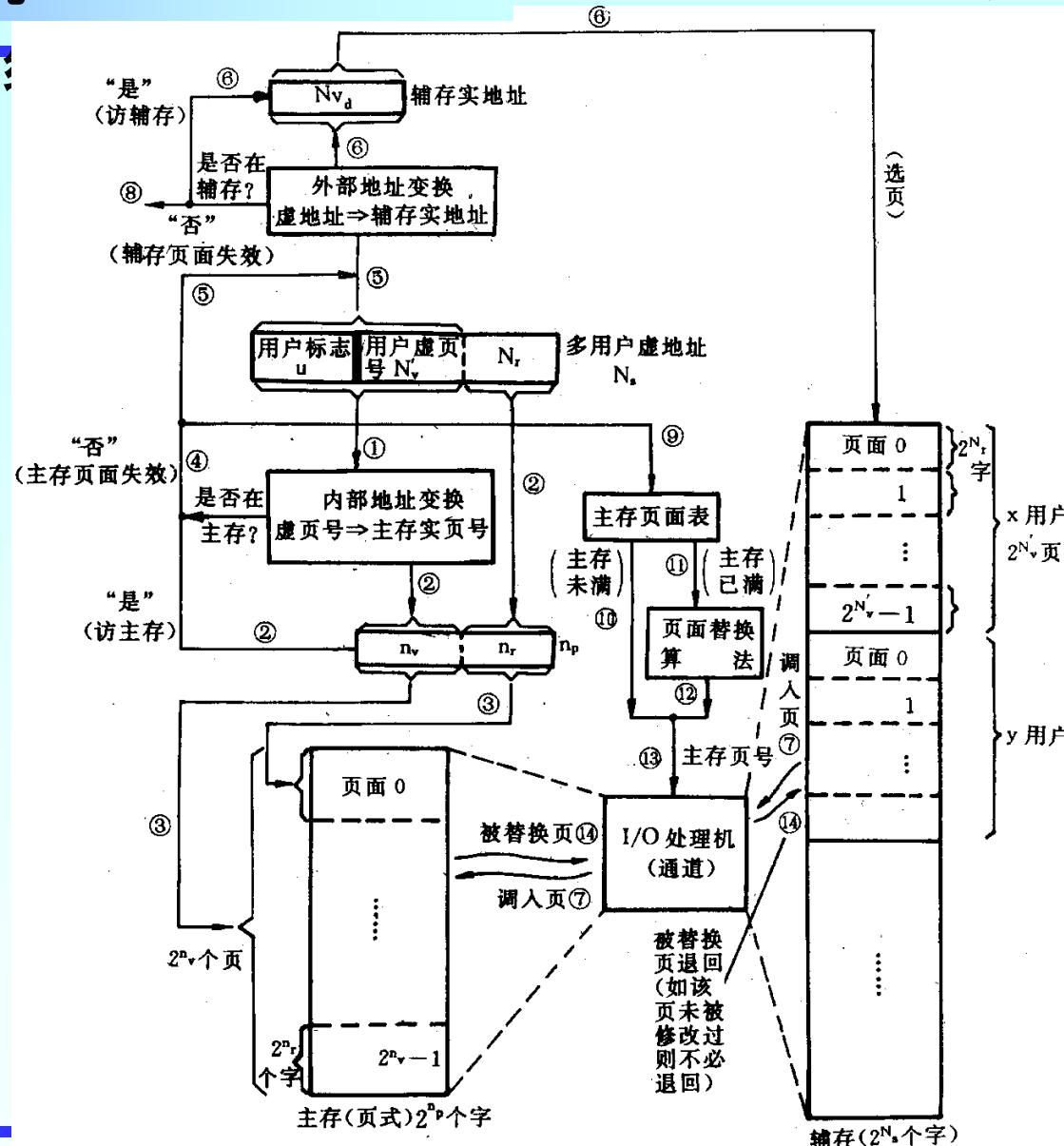


图 页式虚拟内存系统工作的全过程

页式虚拟存储系统的访问过程中，可能会用到**3**张表：内页表、外页表和主存页面表。

- (1) 内页表：在内部地址变换时使用。
- (2) 外页表：在外部地址变换时使用。
- (3) 主存页面表：看主存中是否有空页。这张表是对主存而言的，整个主存只有一个。



4.2.3 页式虚拟存储系统实现中的问题

1. 页面失效的处理

页面失效会在一条指令的分析和执行过程发出。页面失效不能按一般的中断对待，应看作一种故障，一旦出现，处理机必须立即予以响应和处理。



2. 提高虚拟存储系统等效访问速度的措施

要想使虚拟存储系统的等效访问速度提高到接近于主存的访问速度是不容易的。从存储层次的等效访问速度公式可以看出，这一方面要求能有很高的主存命中率，另一方面要求能有尽可能短的访主存时间。



(1) 目录表

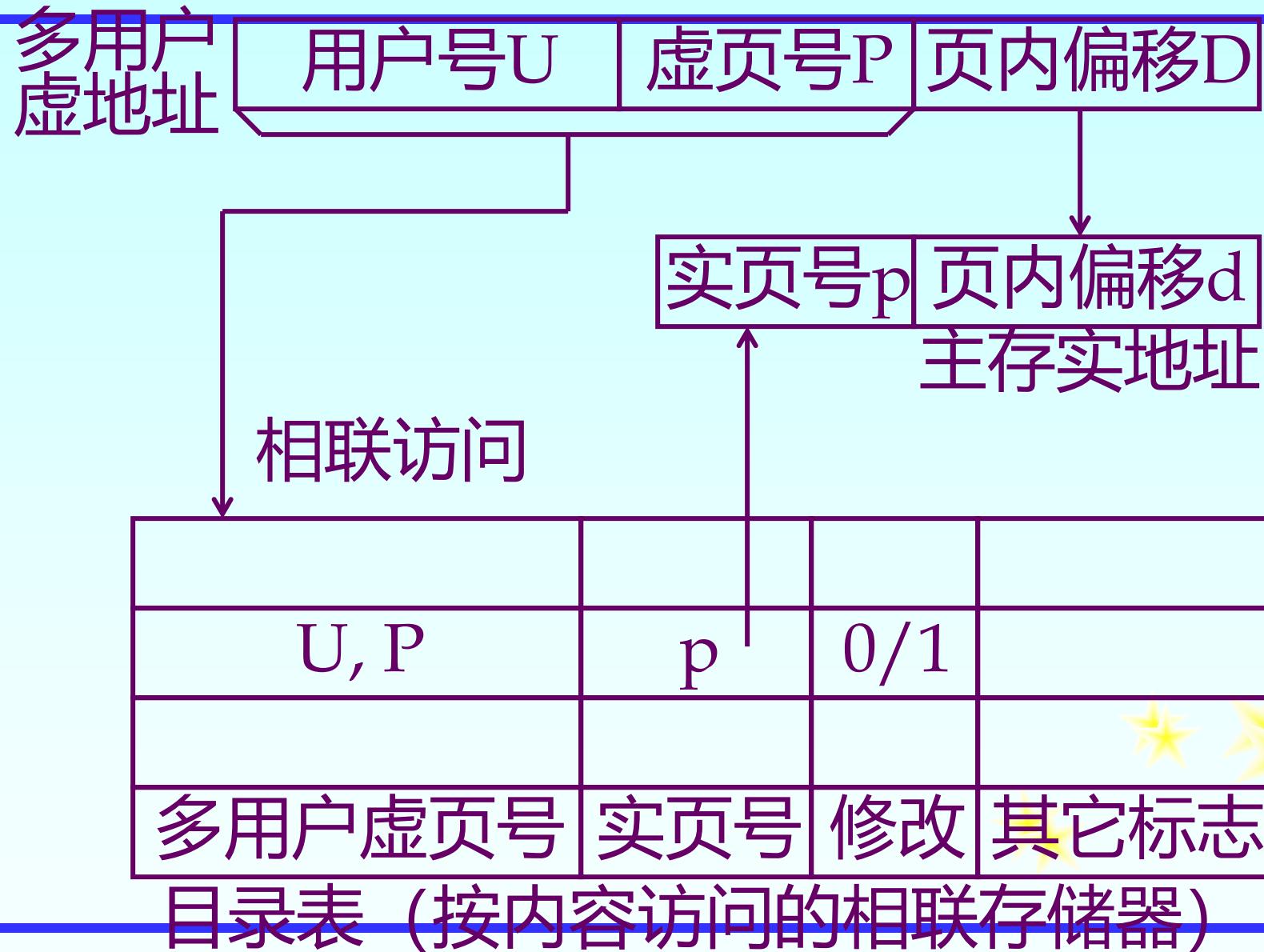
基本思想：用一个小容量高速存储器存放页表。

地址变换过程：把多用户虚地址中**U**与**P**拼接起来，相联访问目录表。读出主存实页号**p**，把**p**与多用户虚地址中的**D**拼接得到主存实地址。如果相联访问失败，发出页面失效请求。

主要优点：与页表放在主存中相比，查表速度快。

主要缺点：可扩展性比较差。主存储器容量增加时，目录表的造价高，速度降低。

4.2 虚拟存储系统



目录表是相联访问的，容量愈大，命中率愈高。但容量愈大，相联查找的速度也愈慢。目录表的命中率与查表速度是有矛盾的。



(2) 快慢表

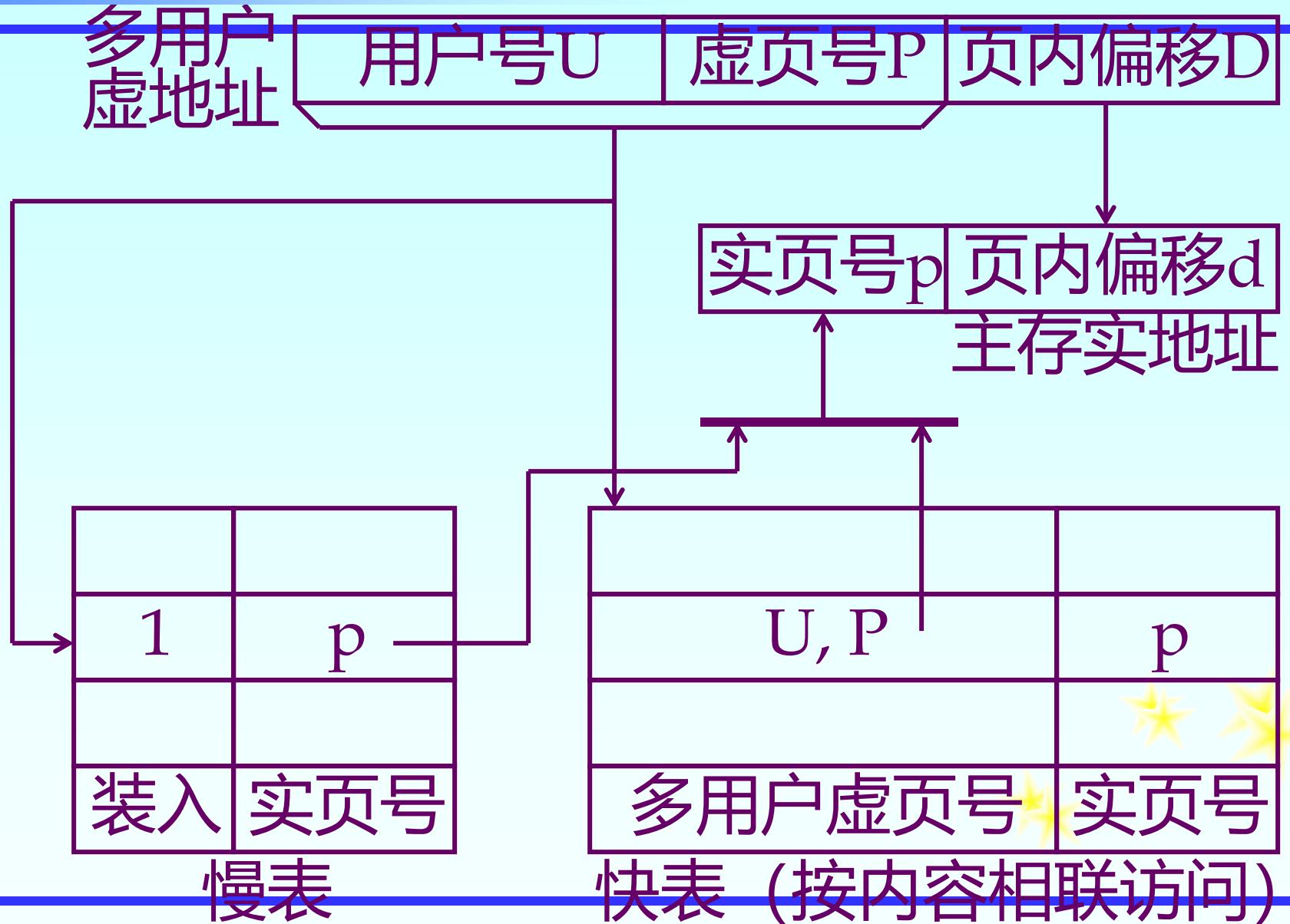
快表TLB(Translation Lookaside Buffer):

小容量(几~几十个字), 高速硬件实现,
采用相联方式访问。

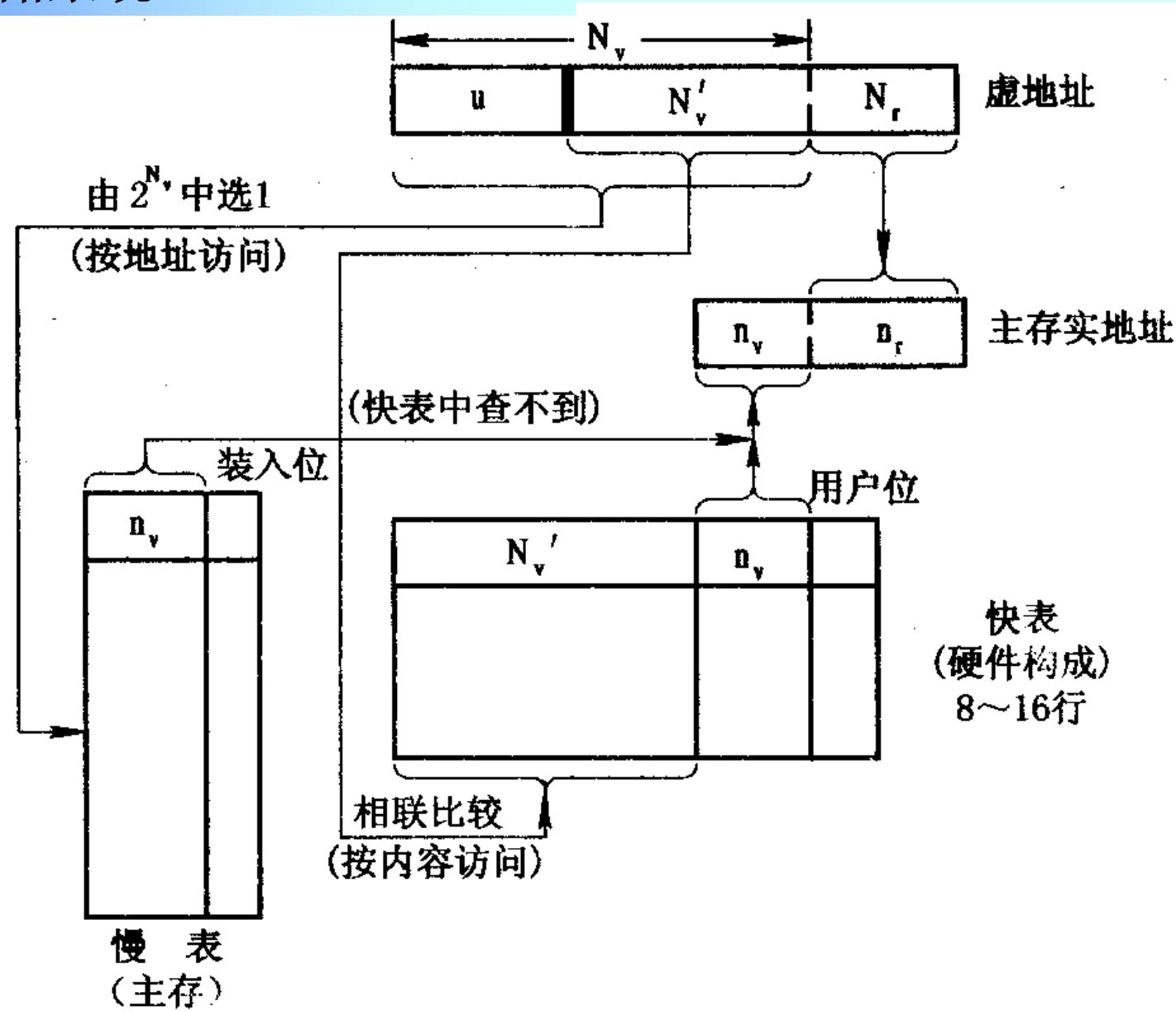
慢表: 当快表中查不到时, 从存放在
主存储器中的慢表中查找按地址访问, 用
软件实现。

快表与慢表也构成了一个两级存储系
统。

4.2 虚拟存储系统



4.2 虚拟存储系统



(3) 散列函数

目的：把相联访问变成按地址访问，从而加大快表容量。

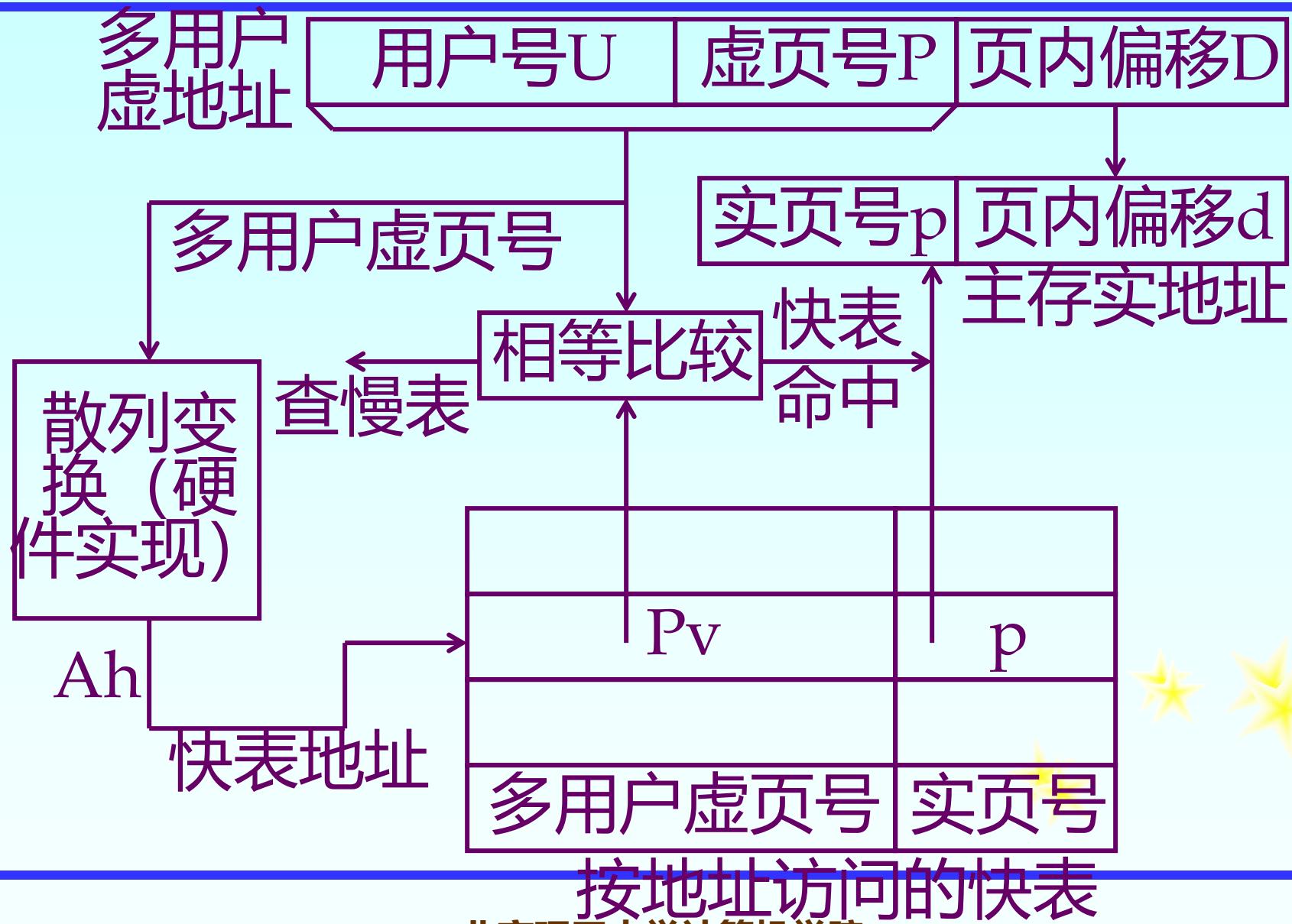
散列（**Hashing**）函数： $A_h = H(P_v)$,
20位左右 \Rightarrow 5~8位

采用散列变换实现快表按地址访问

避免散列冲突：采用相等比较器

地址变换过程：相等比较与访问存储器同时进行

4.2 虚拟存储系统



快表改为按地址访问，容量要比按内容访问的相联存储器大。进一步提高了快表的命中率，而且仍能有很高的查表速度。



例：

IBM370/168计算机的虚拟存储系统快表结构及地址变换过程。虚拟地址共长**48位**，页面大小为**4KB**，每个用户最多占用**4K**个页面，最多允许**16M**个用户，但同时上机的用户数一般不超过**6**个。

采用了两项新的措施：

一是采用两个相等比较器

二是用相联寄存器组把**24位**用户号**U**压缩成**3位**



4.2 虚拟存储

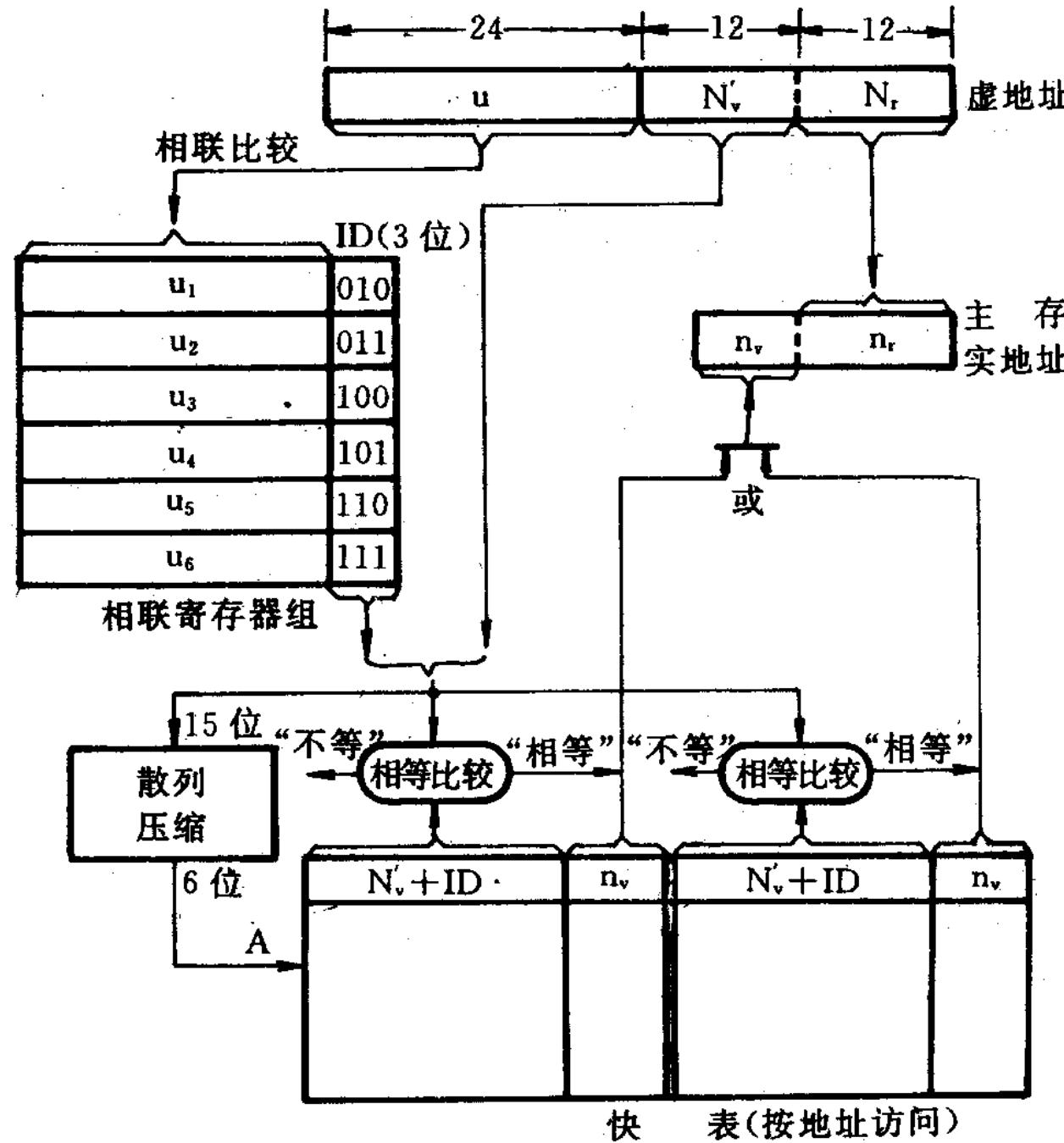


图 4.27 IBM 370/168 虚拟存储系统的快表

3. 影响主存命中率和CPU效率的某些因素

影响主存命中率的主要因素：

- (1) 程序执行过程中的页地址流分布情况；
- (2) 所采用的页面替换算法；
- (3) 页面大小；
- (4) 主存储器的容量；
- (5) 所采用的页面调度算法。

以下，对后三个因素进行分析。



(1) 页面大小与命中率的关系

页面大小为某个值时，命中率达到最大。

页面大小与命中率关系的解释：

假设 At 和 $At+1$ 是相邻两次访问主存的逻辑地址， $d = |At - At+1|$ 。

如果 $d < Sp$ ，随着 Sp 的增大， At 和 $At+1$ 在同一页面的可能性增加，即 H 随着 Sp 的增大而提高。

如果 $d > Sp$ ， At 和 $At+1$ 一定不在同一个页面内。随着 Sp 的增大，主存页面数减少，页面替换将更加频繁。 H 随着 Sp 的增大而降低。

4.2 虚拟存储系统

当**Sp**比较小的时候，前一种情况是主要的，**H**随着**Sp**的增大而提高。

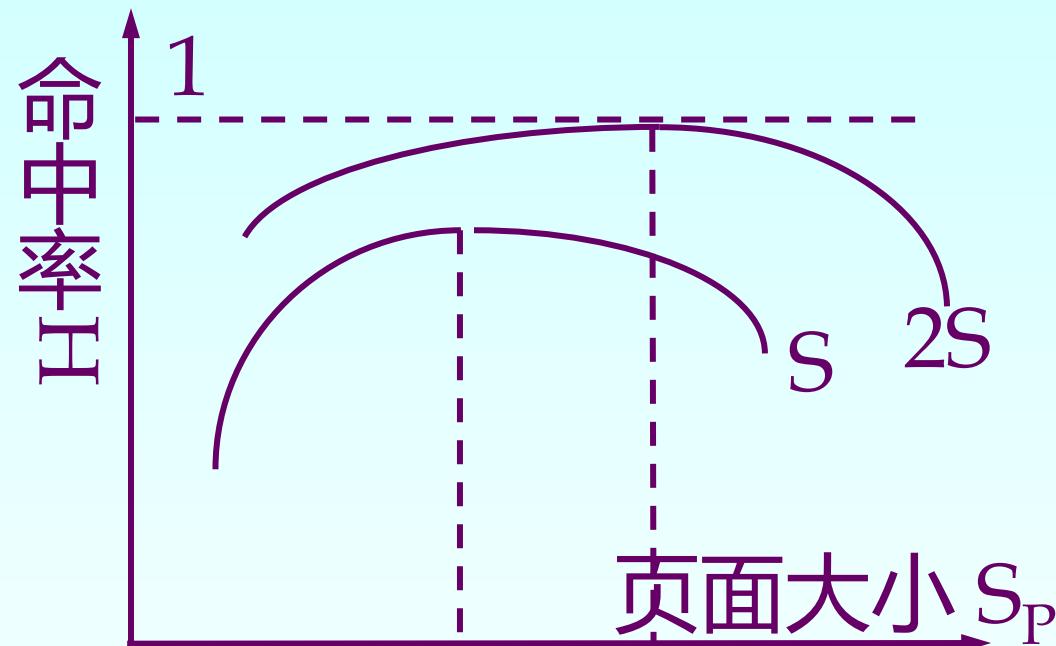
当**Sp**达到某一个最大值之后，后一种情况成为主要的，**H**随着**Sp**的增大而降低。

当页面大小增大时，造成的浪费也要增加。

当页面大小减小时，页表和页面表在主存储器中所占的比例将增加。



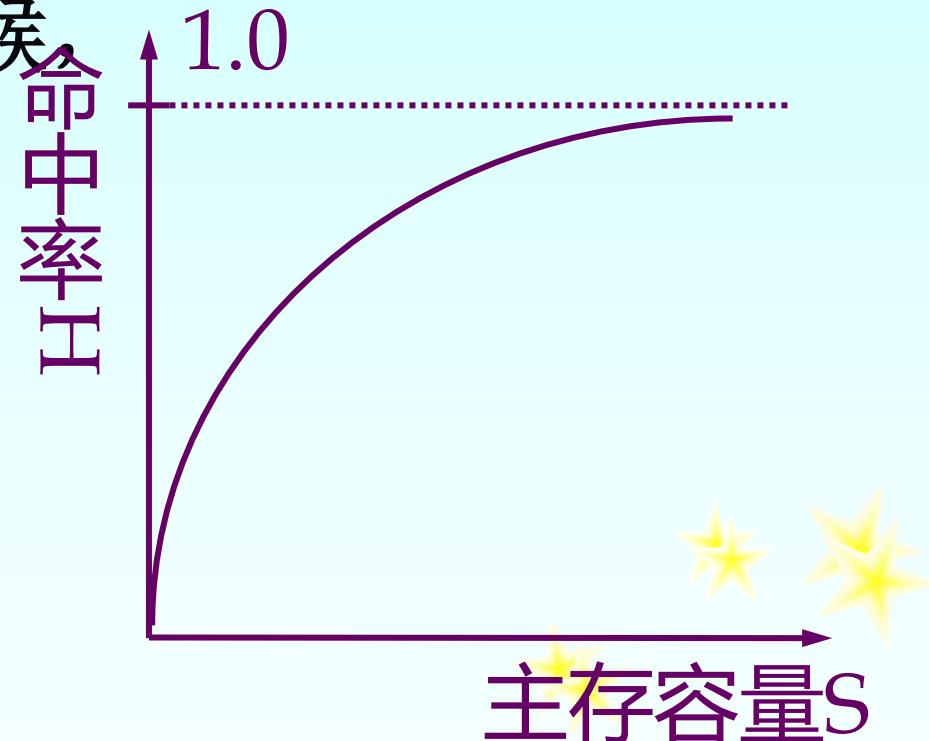
4.2 虚拟存储系统



(2) 主存容量与命中率的关系

主存命中率 H 随着分配给该程序的主存容量 S 的增加而单调上升。

在 S 比较小的时候， H 提高得非常快。随着 S 的逐渐增加， H 提高的速度逐渐降低。当 S 增加到某一个值之后， H 几乎不再提高。



(3) 页面调度方式与命中率的关系

请求式:

当使用到的时候，再调入主存。

优点: 主存利用率高。

缺点: 经常发生页面失效，尤其在程序开始的一段时间内。

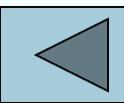


预取式：

在程序重新开始运行之前，把上次停止运行前一段时间内用到的页面先调入到主存储器，然后才开始运行程序。

优点：可以避免在程序开始运行时，频繁发生页面失效的情况。

缺点：如果调入的页面用不上，不仅浪费了调入的时间，而且还占用了主存资源。



4.3 高速缓冲存储器(Cache)

Cache存储系统与虚拟存储系统比较

存储系统	Cache	虚拟存储器
要达到的目标	提高速度	扩大容量
实现方法	全部硬件	软件为主 硬件为辅
两级存储器速度比	3~10倍	10^5 倍
页(块)大小	1~16字	1KB~16KB
等效存储容量	主存储器	虚拟存储器
透明性	对系统和 应用程序员	仅对应用 程序员
不命中时处理方式	等待主存储器	任务切换

4.3 高速缓冲存储器(Cache)

4.3.1 Cache存储系统的基本结构

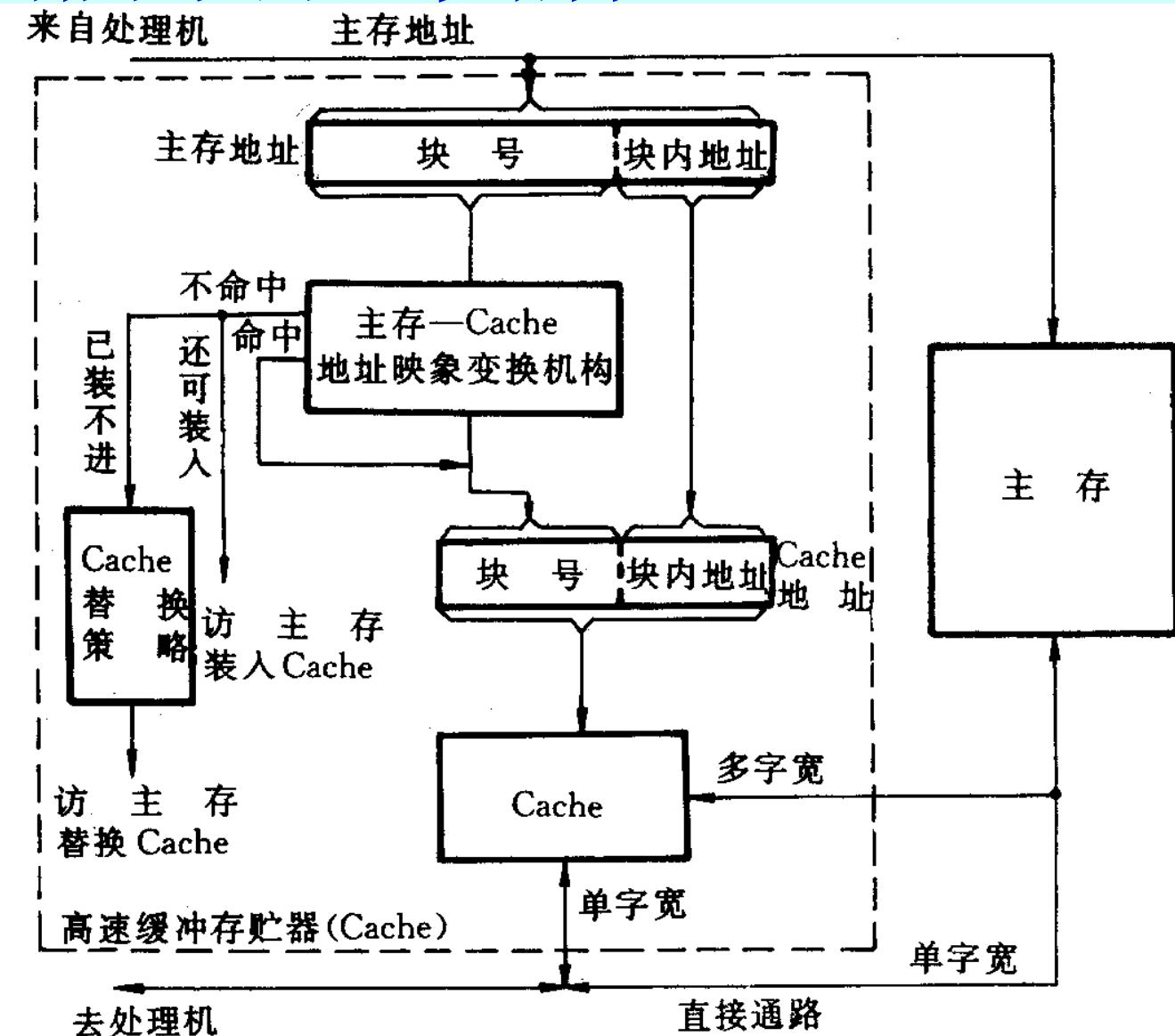


图 Cache存储器的基本结构

4.3 高速缓冲存储器(Cache)

在**Cache**存储系统中，把**Cache**和主存都划分成相同大小的块。

如果**Cache**命中，把主存地址变换成**Cache**地址，直接访问**Cache**。如果**Cache**失效（不命中），用主存地址访问主存，从主存中读出一个字送往**CPU**，同时，把包括该字在一整块都装入**Cache**。



4.3 高速缓冲存储器(Cache)

目前，访问Cache的时间一般可以是访主存时间的 $1/4$ 到 $1/10$ 。如IBM3033、Amdahl470V/7等许多机器的主存周期为300~600ns，而访问Cache的时间只需要50~100ns。因此，只要Cache的命中率足够高，就相当于能以接近于Cache的速度来访问大容量的主存。Cache存储器已在大、中、小以及微型机上普遍采用。

4.3 高速缓冲存储器(Cache)

为了加速调块，一般让每块的容量等于在一个主存周期内由主存所能访问到的字数，因此在有Cache存储器的主存系统都采用多体交叉存储器，例如，IBM 370/168的主存是模4交叉，每个分体是8个字节宽，所以Cache的每块为32个字节；CRAY-1的主存是模16交叉，每个分体是单字宽，所以其指令Cache(专门存放指令的Cache)的块容量为16个字。

4.3 高速缓冲存储器(Cache)

另外，主存被机器的多个部件所共用，应尽量提高Cache的访主存优先级，一般应高于通道的访主存级别，这样在采用Cache存储器的系统中，访存申请响应的优先顺序通常安排成Cache、通道、写数、读数、取指。因为Cache的调块时间只占用1~2个主存周期，这样做不会对外设访主存带来太大的影响。



4.3 高速缓冲存储器(Cache)

4.3.2 地址的映像与变换

地址映像：

把存放在主存中的程序按照某种规则装入到**Cache**中，并建立主存地址与**Cache**地址之间的对应关系。

地址变换：

当程序已经装入到**Cache**之后，在实际运行过程中，把主存地址变换成**Cache**地址。

在选取地址映像方法要考虑的主要因素：

地址变换的硬件要容易实现；地址变换的速度要快；主存空间利用率要高；发生块冲突的概率要小。

4.3 高速缓冲存储器(Cache)

1、全相联映像和变换

映像规则：主存中的任意一块都可以映像到**Cache**中的任意一块。

如果**Cache**的块数为 C_b ，主存的块数为 M_b ，映像关系共有： $C_b \times M_b$ 种。

用硬件实现非常复杂。

在虚拟存储系统中，全部用软件实现。



4.3 高速缓冲存储器(Cache)

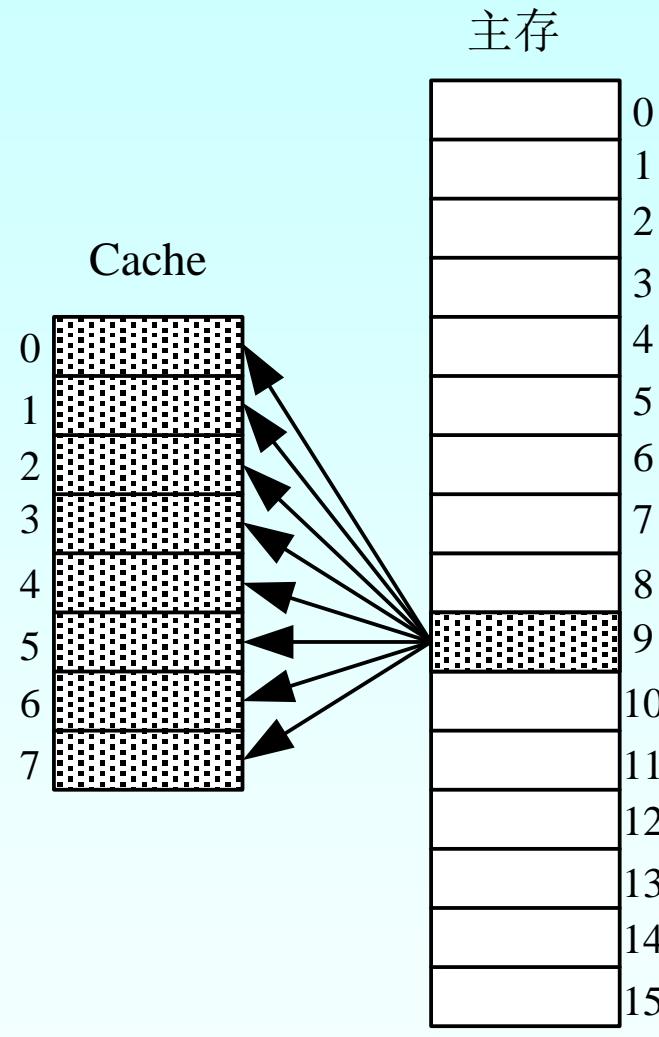


图 4.31 全相联映像规则
北京理工大学计算机学院

4.3 高速缓冲存储器(Cache)

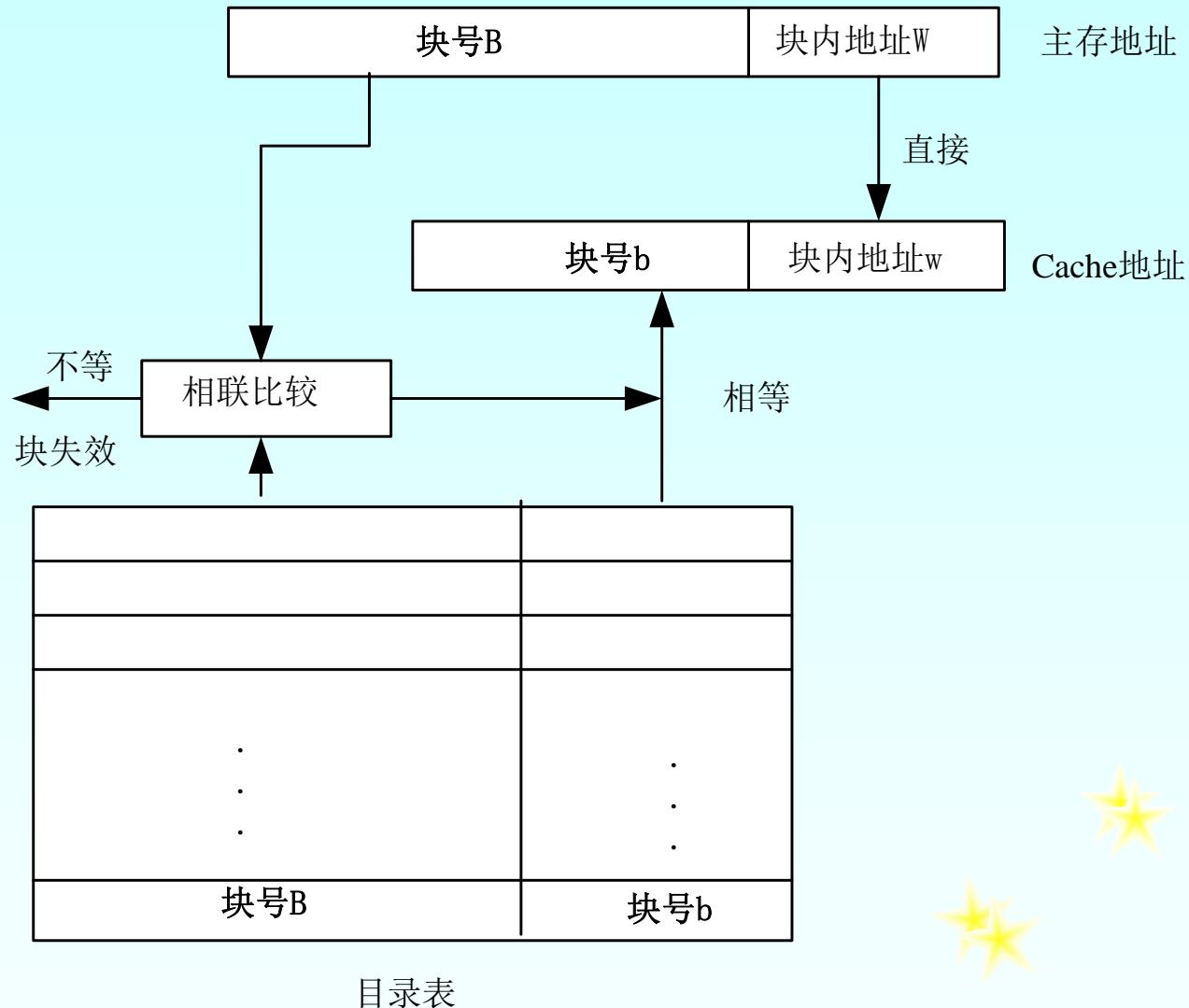


图 4.32 北京相联映像的地址变换过程

4.3 高速缓冲存储器(Cache)

全相联映像方式的主要优点：

块冲突概率比较小。

Cache的利用率高。

全相联映像方式的主要缺点：需要一个相联存储器，其代价很高。相联比较所花费的时间将影响**Cache**的访问速度。



2、直接映像及其变换

映像规则：主存中一块只能映像到**Cache**的一个特定的块中。

计算公式： $b=B \bmod C_b$, 其中：

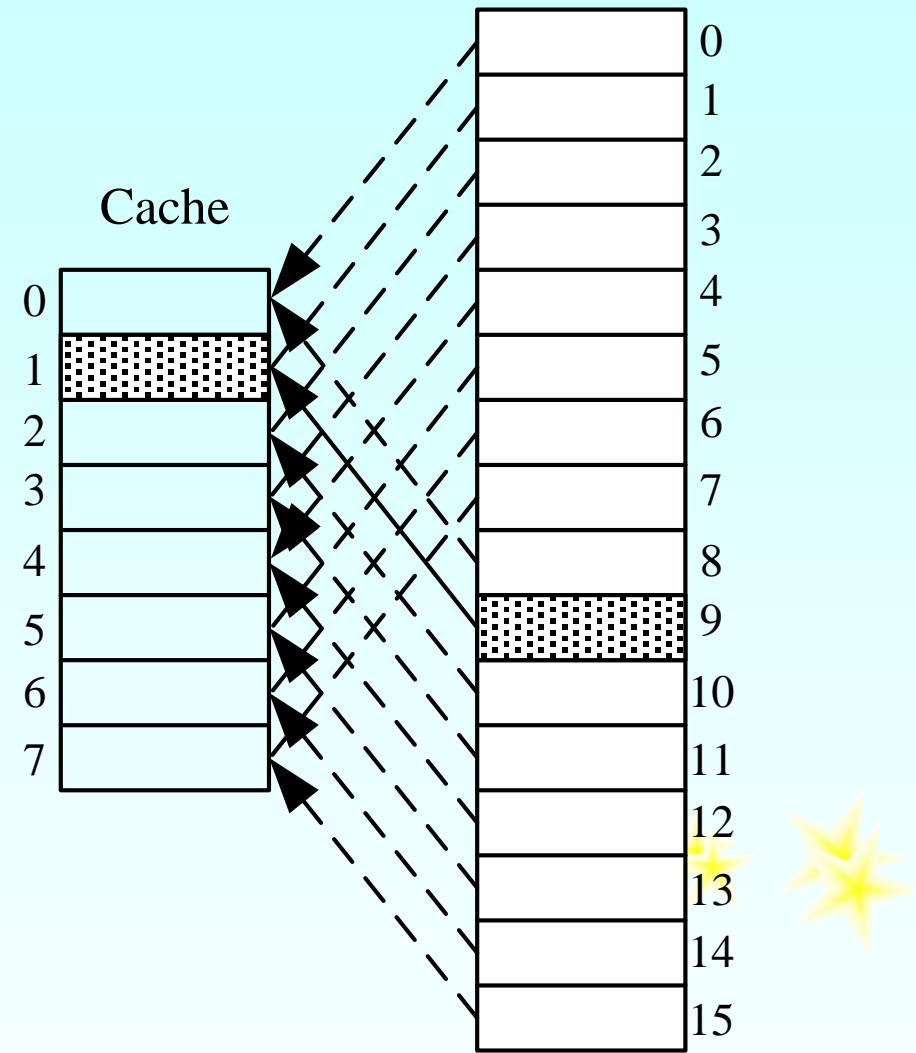
b为**Cache**的块号，

B是主存的块号，

C_b是**Cache**的块数。

整个**Cache**地址与主存地址的低位部分完全相同。

4.3 高速缓冲存储器(Cache)



直接映像方式

4.3 高速缓冲存储器(Cache)

地址变换过程：

用主存地址中的块号**B**去访问区号存储器；

把读出来的区号与主存地址中的区号**E**进行比较；

比较结果相等，且有效位为**1**，则**Cache**命中；

比较结果相等，有效位为**0**，表示**Cache**中的这一块已经作废；

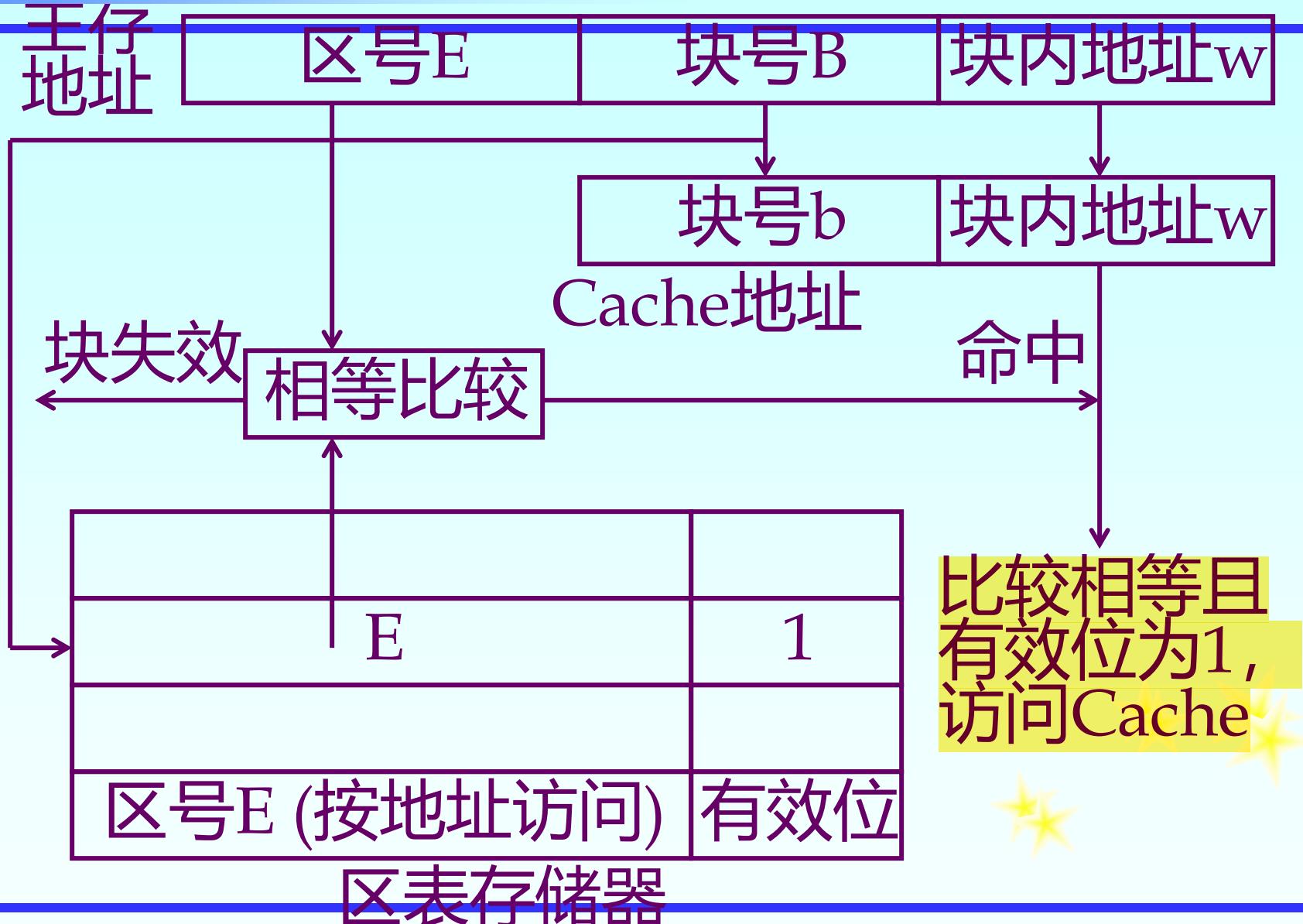
4.3 高速缓冲存储器(Cache)

比较结果不相等, 有效位为**0**, 表示**Cache**中的这一块是空的;

比较结果不相等, 有效位为**1**, 表示原来在**Cache**中的这一块是有用的。



4.3 高速缓冲存储器(Cache)



4.3 高速缓冲存储器(Cache)

直接映像方式的主要优点:

硬件实现很简单, 不需要相联访问存储器。
访问速度也比较快, 实际上不做地址变换。

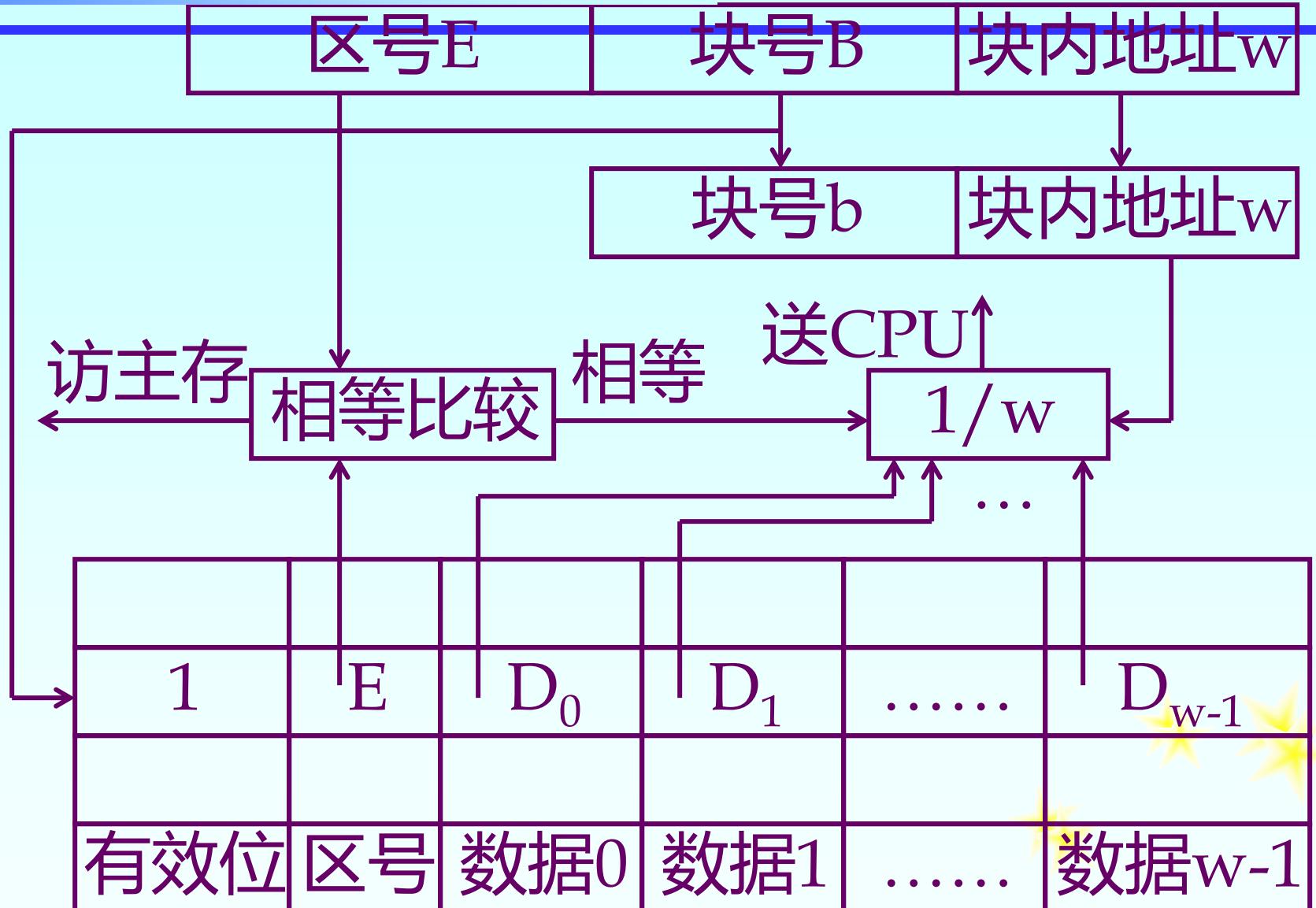
直接映像方式的主要缺点: 块的冲突率较高。

提高Cache速度的一种方法:

把区号存储器与Cache合并成一个存储器。



4.3 高速缓冲存储器(Cache)



按地址访问的Cache
北京理工大学计算机学院

4.3 高速缓冲存储器(Cache)

3、组相联映像及其变换

组相联映像实际上是对全相联映像和直接映像的折衷方案，当组数等于**1**（不再分组），组相联映像就变成为全相联映像；当组数等于**Cache**中块的数目，组相联映像就变成为直接映像。所以其优点和缺点介于全相联和直接映像方式的优缺点之间。

映像规则：

主存和**Cache**按同样大小划分成块，还按同样大小划分成组。

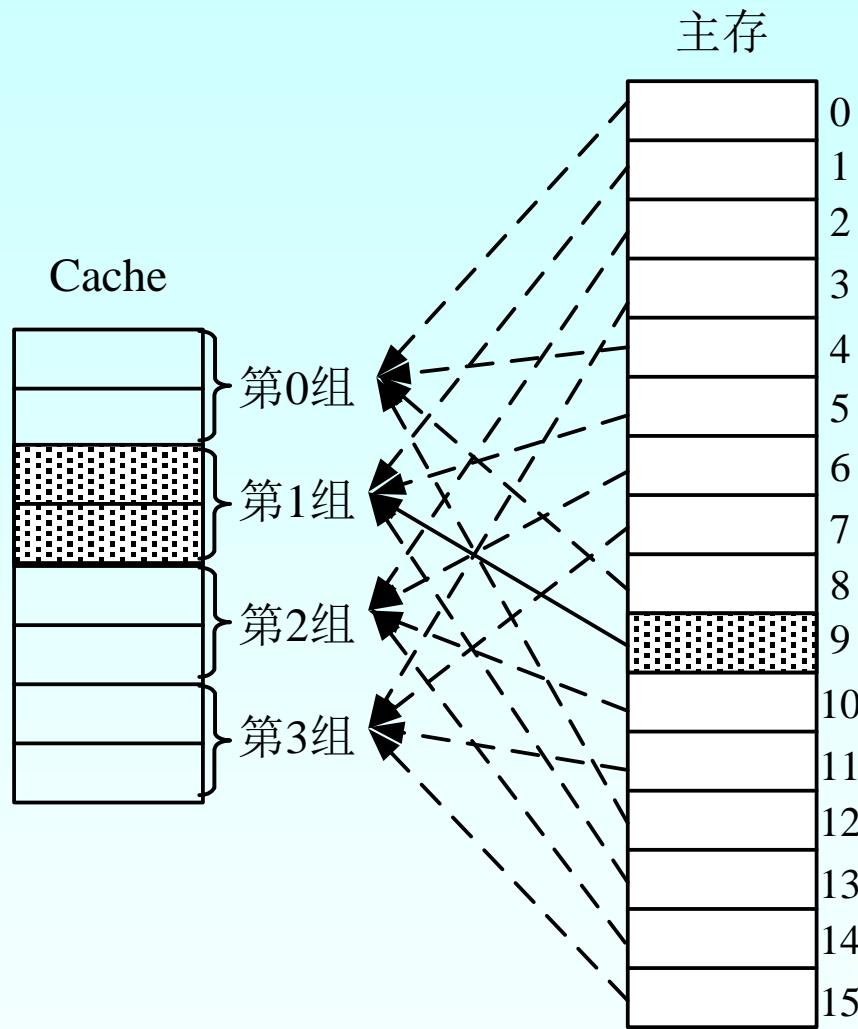
从主存的块到**Cache**的组之间采用直接映像方式。在对应的组内部采用全相联映像方式，组内随便放。

4.3 高速缓冲存储器(Cache)

通常将组内**2**块的组相联映像称为二路组相联，组内**4**块的组相联映像称为四路组相联。关于组相联映像方式的具体映像实现方案，以二路组相联为例，**Cache**分成**4**组，组相联映像方式如图所示。块**9**直接映像到**Cache**的组**1** ($1=9 \bmod 4$)，块**9**可映像到组**1**的块**0**和块**1**。



4.3 高速缓冲存储器(Cache)



4.3 高速缓冲存储器(Cache)

假设**Cache**空间分成 C_g 组 ($C_g = 2^g$)，每组为 G_b 块 ($G_b = 2^b$)。主存地址分为三部分：标记、组号、块内地址；**Cache**地址分为三部分：组号、组内块号、块内地址。主存地址的组号由**G**来表示，它的宽度**Cache**地址的组号 g 是一致的。**Cache**地址和主存地址的格式如图所示。

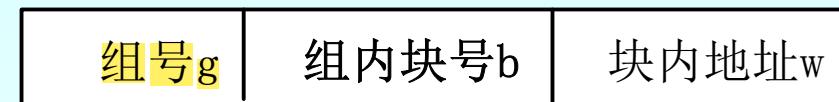


4.3 高速缓冲存储器(Cache)

主存地址



Cache地址

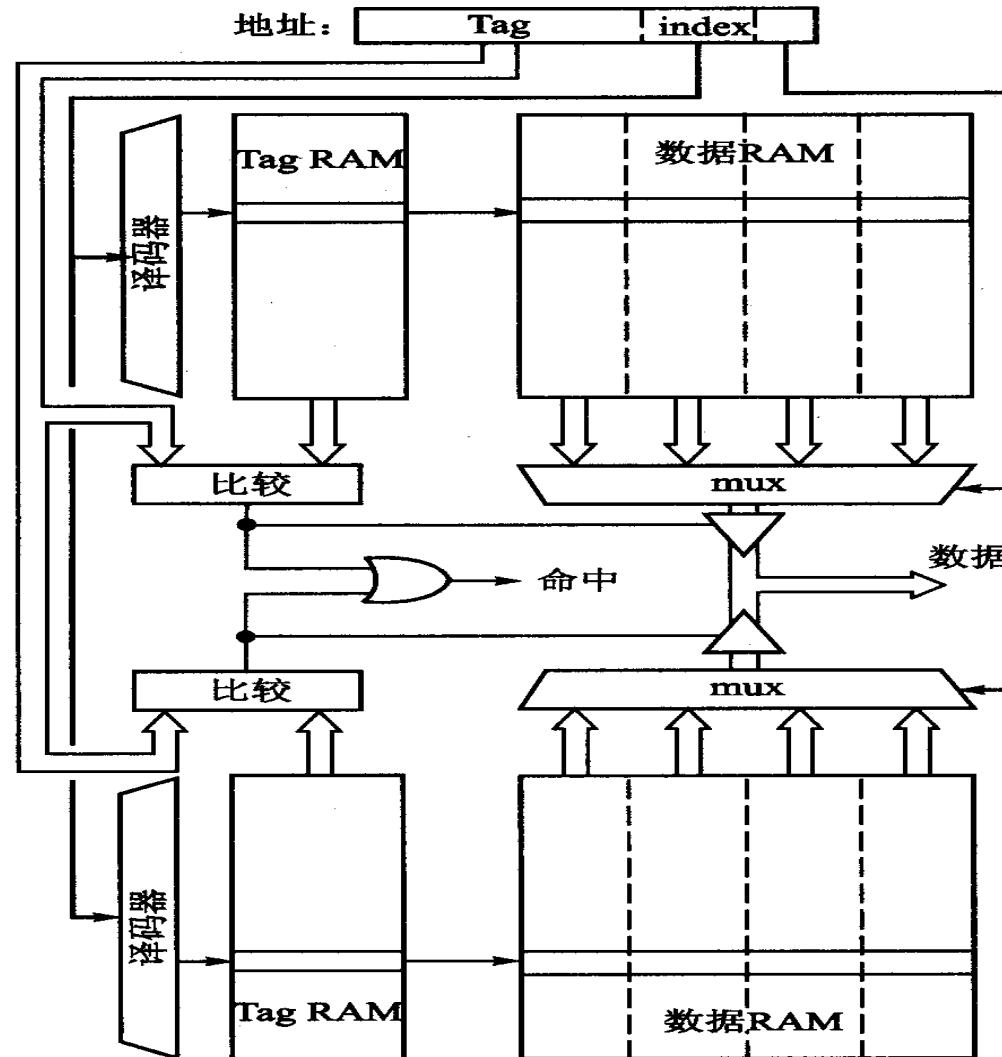


4.3 高速缓冲存储器(Cache)

组相联映像也使用低位地址直接访问**Cache**块，但它选中的是一个组，组内包含有2块或多个块。给定的内存块可以放在选中组中的任意一块内。一组内的块数，一般称为相联度或相联路数。选中一组后，组内所有块的标识(**tag**)同时进行比较，如果有任何一个匹配，则“命中”。组相联映像实际上是靠比较器的个数及增宽**Cache**位来降低**Cache**块的冲突。下图是**ARM**处理器2路组相联的示意图。



4.3 高速缓冲存储器(Cache)



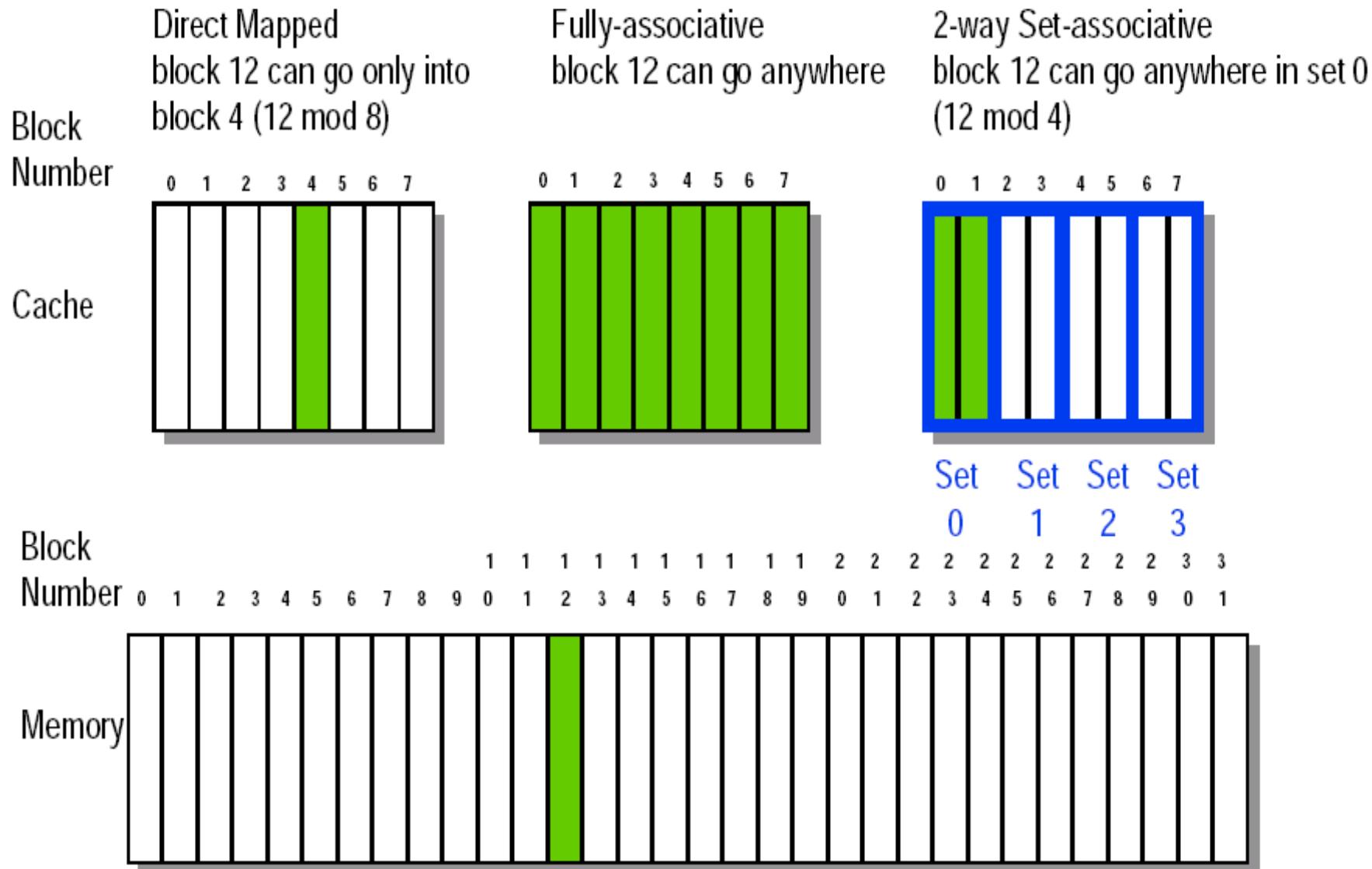
4.3 高速缓冲存储器(Cache)

图中的**Cache**为**8KB**，每行**16字节**，分为**2**个组直接映像**Cache**，每个为**256**行。32位地址中的**4**位作为行内的字节选择，组号(**Index**)只需**8**位，其余的高**20**位作为地址标识。组相联是通过“单体多字存储器”来并行查询，**2**路组相联即通过单体双字存储器来查询，有**2**个比较器。地址码字段如下：

Tag (20位)	Index (8位)	字节选择 (4位)
-----------	------------	-----------



4.3 高速缓冲存储器(Cache)



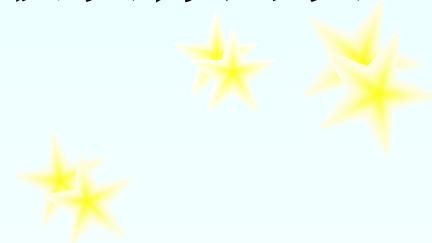
4.3 高速缓冲存储器(Cache)

例8：某计算机的**Cache**共有**16**块，采用**2路组相联**映像方式（即每组**2**块）。每个主存块大小为**32**字节，按字节编址。主存**129**号单元所在主存块应装入到的**Cache**组号是多少？

解：由于每个主存块大小为**32**字节，按字节编址。

根据计算主存块号的公式，主存块号 =

$\lfloor \text{主存地址} / \text{块大小} \rfloor = \left\lfloor \frac{129}{32} \right\rfloor = 4$ ，所以主存**129**号单元所在的主存块应为第**4**块。若**Cache**共有**16**块，采用**2路组相联**映像方式，可分为**8**组。根据组相联映像的映像关系，主存第**4**块转入**Cache**第**4**组。



4.3 高速缓冲存储器(Cache)

组相联映像方式的优点：

块的冲突概率比较低；
块的利用率大幅度提高；
块失效率明显降低。

组相联映像方式的缺点：

实现难度和造价要比直接映像方式高。



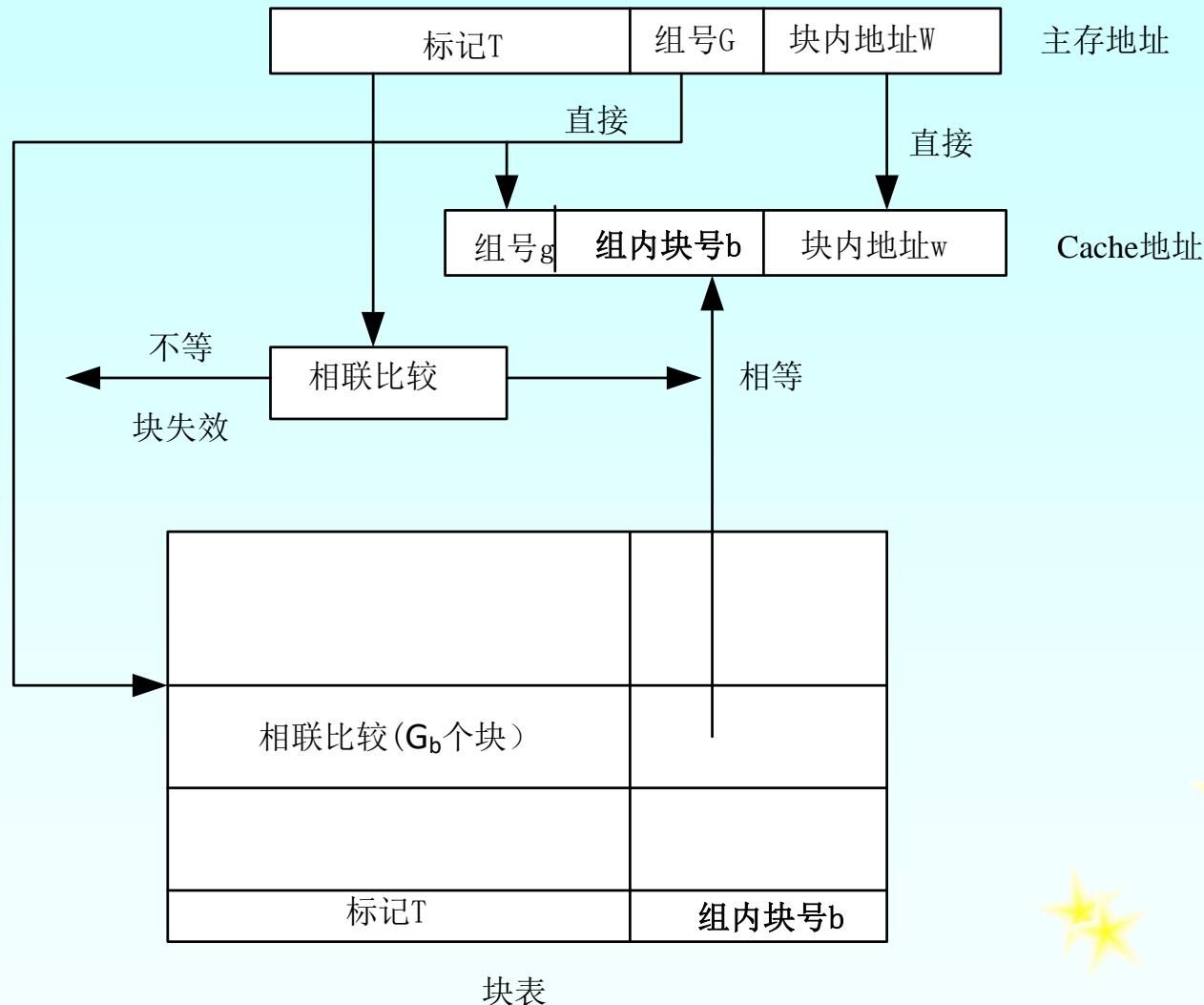
4.3 高速缓冲存储器(Cache)

地址变换过程：

用主存地址的组号**G**按地址访问块表存储器。把读出来的标记与主存地址中的标记进行相联比较，如果有相等的，表示**Cache**命中，访问**Cache**；如果没有相等的，表示**Cache**没有命中，访问主存，块装入**Cache**，修改块表。

通常，**Cache**存储器的地址映像规则使用组相联或直接映像，而不采用全相联映像。否则，主存—**Cache**的地址映像表太大，查表速度太慢，硬件无法实现。

4.3 高速缓冲存储器(Cache)



4.3 高速缓冲存储器(Cache)

- 共用**C_g**个目录表，每个目录表只需**G_b**行。
用高速小容量存储器做成块表存储器，组内采用全相联映像方式，在组之间采用按地址访问。块表容量与**Cache**的块数相等。
- 两个以上的虚页要想进入主存中同一个实页位置时，就会产生实页冲突（页面争用）。地址映像方式的选择应考虑尽量降低实页冲突发生的概率，同时希望辅助硬件较少，成本较低，以及地址变换的速度较快。虚拟存储系统一般都采用全相联的映像规则。

4.3 高速缓冲存储器(Cache)

4.3.3 替换算法的实现

Cache替换算法使用的时间：

发生块失效，且可以装入新调入块的几个Cache块都已经被装满时。

直接映像方式实际上不需要替换算法。

全相联映像方式的替换算法最复杂。



4.3 高速缓冲存储器(Cache)

Cache替换算法要解决的问题：

- 1、记录每次访问Cache的块号；
- 2、管理好所记录的Cache块号，为找出被替换的块号提供方便；
- 3、根据记录和管理的结果，找出被替换的块号。

Cache替换算法的主要特点：

全部用硬件实现。



4.3 高速缓冲存储器(Cache)

1. 堆栈法

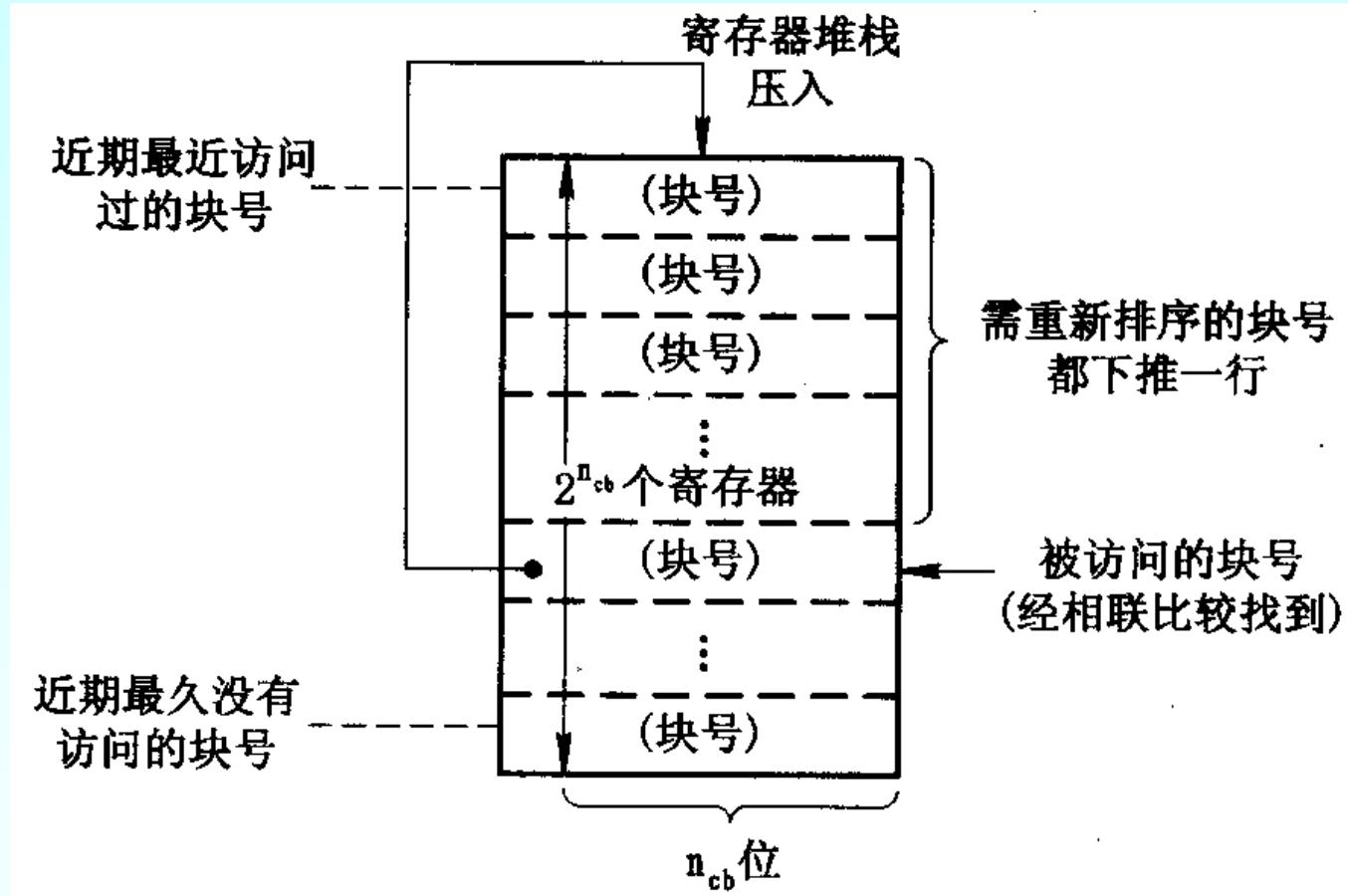


图 4.39 全相联映像LRU法经堆栈实现(需要有相联比较功能)
北京理工大学计算机学院

4.3 高速缓冲存储器(Cache)

2. 比较对法

比较对法的基本思路是让各个块成对组合，用一个触发器的状态来表示该比较对内两块访问的远近次序，再经门电路就可找到LRU块。例如有A、B、C3块，互相之间可组合成AB、BA、AC、CA、BC、CB6对，其中AB和BA、AC和CA、BC和CB是重复的，所以只需取AB、AC、BC 3对。各对内块的访问顺序分别用“对触发器” T_{AB} 、 T_{AC} 、 T_{BC} 表示。 T_{AB} 为“1”，表示A比B更近被访问过； T_{AB} 为“0”，表示B比A更近被访问过。 T_{AC} 、 T_{BC} 也类似定义。这样，当访问过的次序为A B C，即最近访问过的为A，最久未被访问过的为C，则这三个触发器状态分别必为 $T_{AB}=1$ ， $T_{AC}=1$ ， $T_{BC}=1$ 。

4.3 高速缓冲存储器(Cache)

如果访问过的次序为B A C， C为最久未被访问过的块，则此时必有 $T_{AB}=0$, $T_{AC}=1$, $T_{BC}=1$ 。因此以最久未被访问过的块C作为被替换掉的块的话，用布尔代数式必有

$$C_{LRU} = T_{AB} \cdot T_{AC} \cdot T_{BC} + \overline{T_{AB}} \cdot T_{AC} \cdot T_{BC} = T_{AC} \cdot T_{BC}$$

$$B_{LRU} = T_{AB} \cdot \overline{T_{BC}}$$

$$A_{LRU} = \overline{T_{AB}} \cdot \overline{T_{AC}}$$



4.3 高速缓冲存储器(Cache)

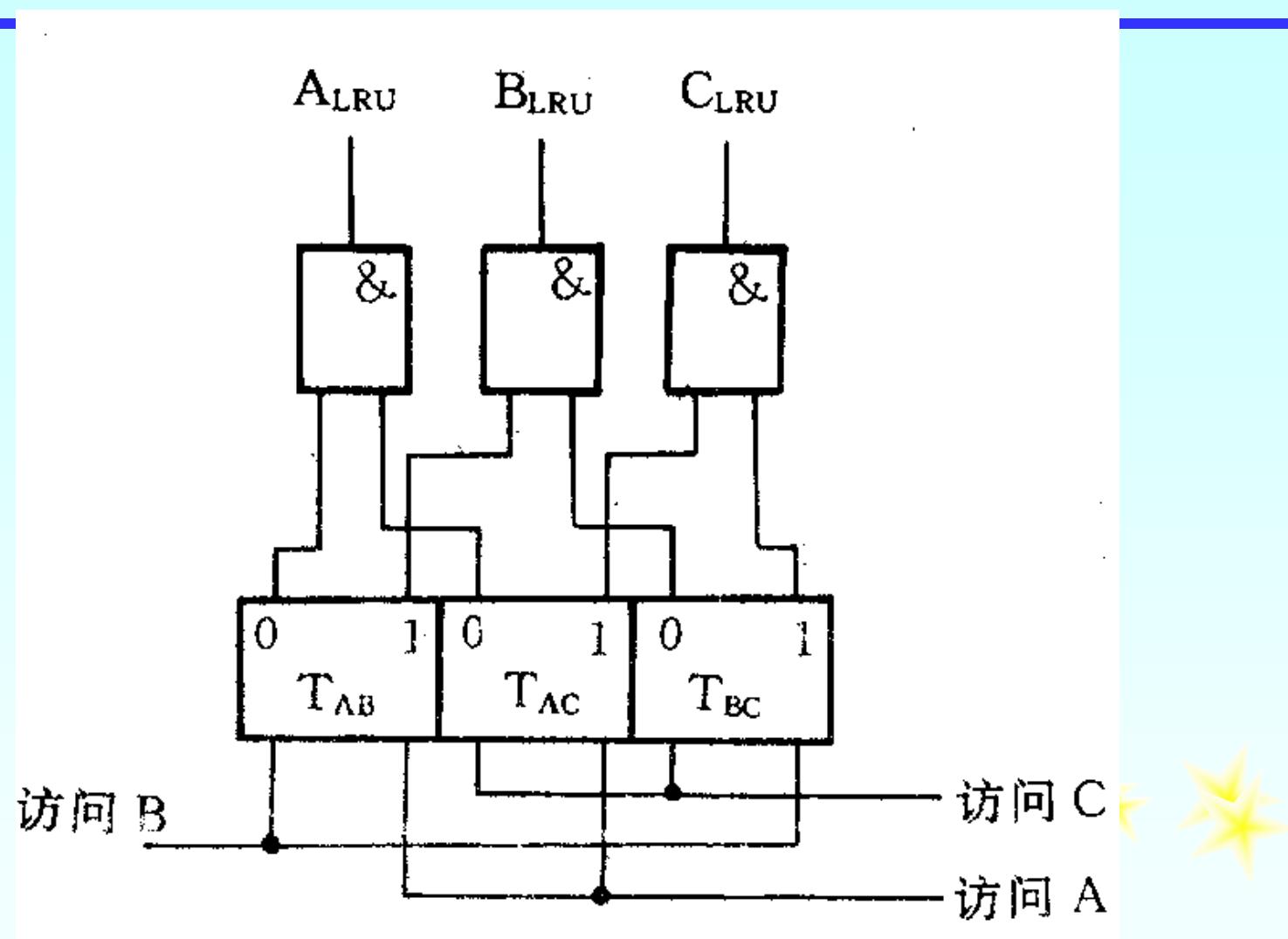


图 用比较对法实现LRU算法

4.3 高速缓冲存储器(Cache)

现在来分析比较对法所用的硬件量。由于每块均可能作为LRU块，其信号需要用一个与门产生，所以有多少块，就得有多少个与门；每个与门接收与它有关的触发器来的输入，例如 A_{LRU} 与门要有从 T_{AB} 、 T_{AC} 来的输入， B_{LRU} 要有从 T_{AB} 、 T_{BC} 来的输入，而与每块有关的对数为块数减去1，所以与门的扇入数是块数减去1。若 p 为块数，两两组合，比较对触发器的个数应为 C_p^2 ，即为 $p \cdot (p-1)/2$ 。表4.2给出了比较对法块数 p 的取值与门数、门的输入端数及比较对触发器数的关系。

4.3 高速缓冲存储器(Cache)

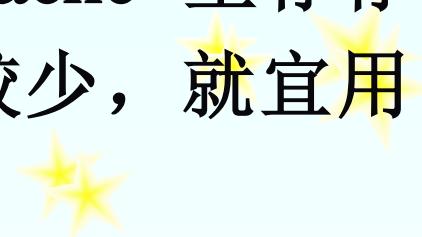
表 比较对触发器数、门数、门的输入端数与块数的关系

块数	3	4	8	16	64	256	...	p
比较对触发器数	3	6	28	120	2016	32 640	...	$p(p-1)/2$
门数	3	4	8	16	64	256	...	p
门输入端数	2	3	7	15	63	255	...	$p-1$



4.3 高速缓冲存储器(Cache)

综上所述，替换算法实现的设计是围绕下述两点来考虑的：一是如何对每次访问进行记录(使用位法、堆栈法、比较对法所用的记录方法都不同)；二是如何根据所记录的信息来判定近期内哪一块是最久没有被访问过的。由此可见，实现方法和所用的映像方法密切相关。例如，对于主存—辅存存储层次的全相联映像宜于采用使用位法或类似的方法，而不宜采用堆栈法和比较对法；但对于Cache—主存存储层次的组相联映像，因为组内块数较少，就宜用比较对法或堆栈法。



4.3.4 Cache的透明性及性能分析

1. Cache的透明性分析

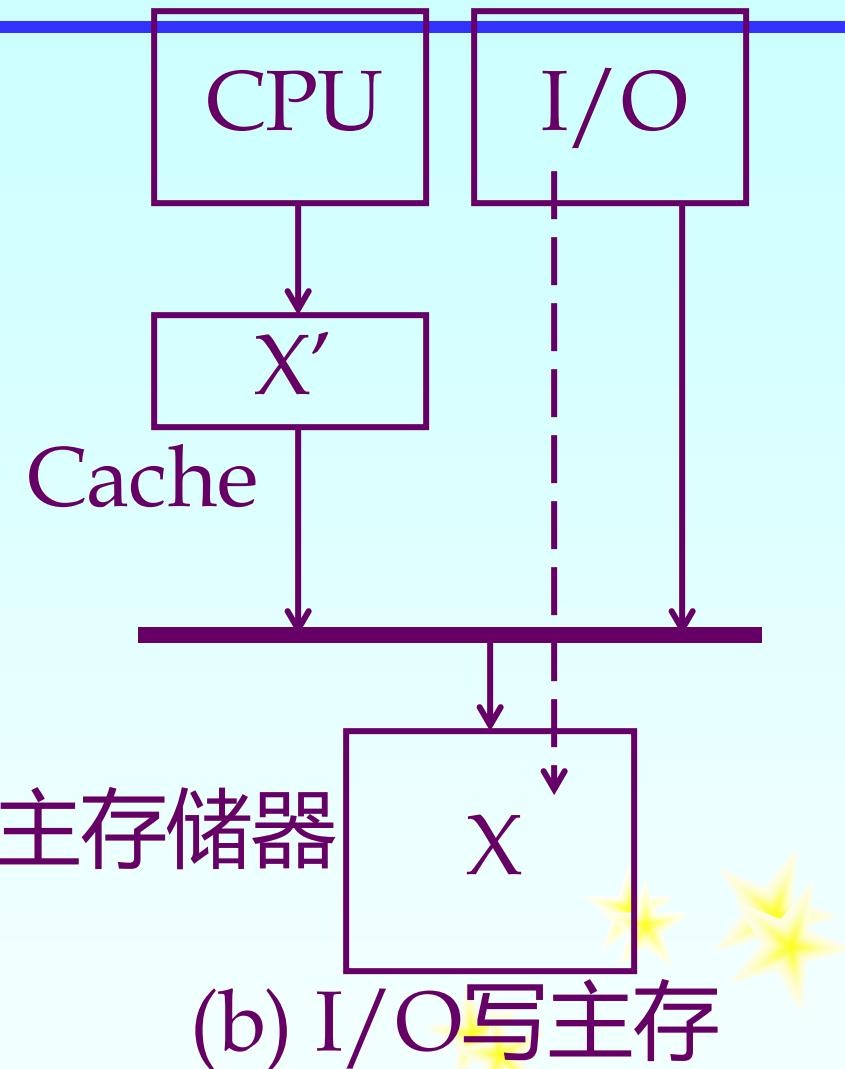
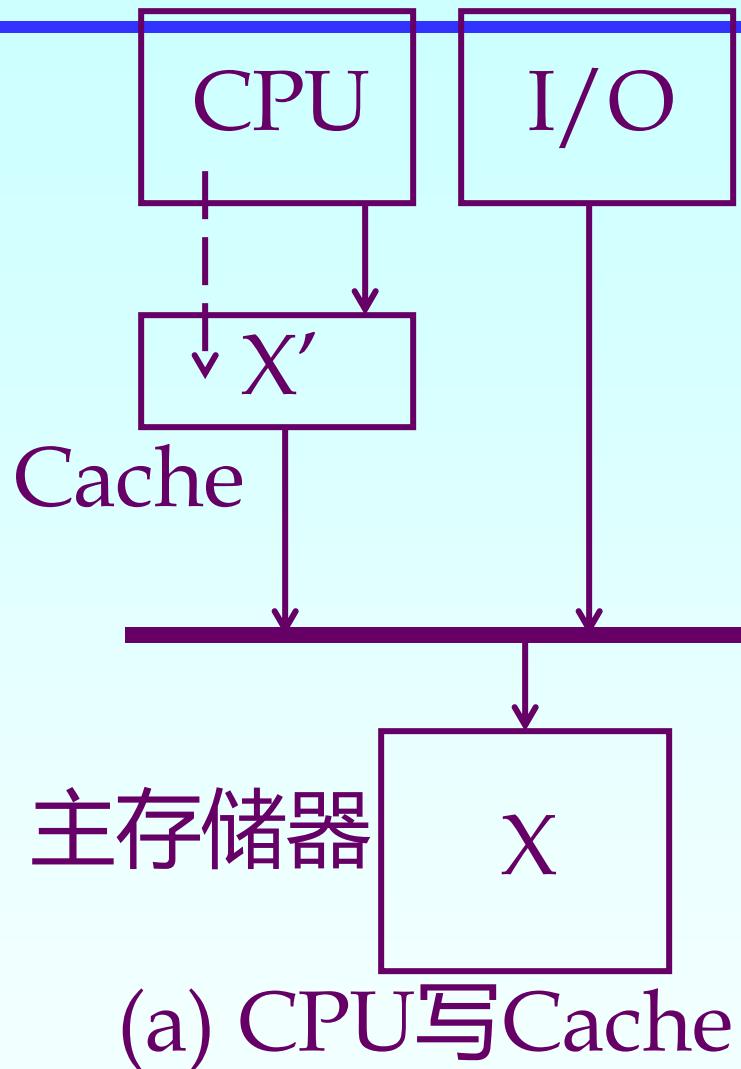
Cache存储层次对系统程序员和应用程序员都是透明的。

Cache是主存的一部分副本。造成Cache与主存的不一致的原因：

- (1) 由于CPU写Cache，没有立即写主存。
- (2) 由于I/O处理机或I/O设备写主存。



4.3 高速缓冲存储器(Cache)



Cache与主存不一致的两种情况

Cache存储系统的更新算法：

(1) 写直达法(写通过法), **Write-through**

CPU在执行写操作时，把数据同时写入**Cache**和主存。

(2) 写回法 (抵触修改法)**Write-Back**

CPU数据只写入**Cache**，不写入主存，仅当替换时，才把修改过的**Cache**块写回到主存。



写回法与写直达法的优缺点比较：

(1) 可靠性，写直达法优于写回法。

(2) 与主存的通信量，写回法少于写直达法。

例如：写操作占总访存次数的**20%**, Cache命中率为**99%**, 每块**4**个字。当Cache发生块替换时, 有**30%**块需要写回主存, 其余的因未被修改过而不必写回主存。则对于**WT**法, 写主存次数占总访存次数的**20%**。而**WB**法为 $(1-99\%) \times 30\% \times 4 = 1.2\%$ 。因此,**WB**法与主存的通信量要比**WT**法少**10**多倍。

4.3 高速缓冲存储器(Cache)

- (3) 控制的复杂性，写直达法比写回法简单。
- (4) 硬件实现的代价，写回法要比写直达法好。

写Cache的两种方法：

- (1) 不按写分配法：在写**Cache**不命中时，只把所要写的字写入主存。
- (2) 按写分配法：在写**Cache**不命中时，还把一个块从主存读入**Cache**。

目前，在写回法中采用按写分配法，在写直达法中采用不按写分配法。

4.3 高速缓冲存储器(Cache)

对于单处理机系统的Cache，多数采用写回法。但对共享主存的多处理机系统，为保证各处理机经主存交换信息时不出错，较多采用写直达法。

对于共享主存的多处理机系统，各CPU都有自己的Cache。

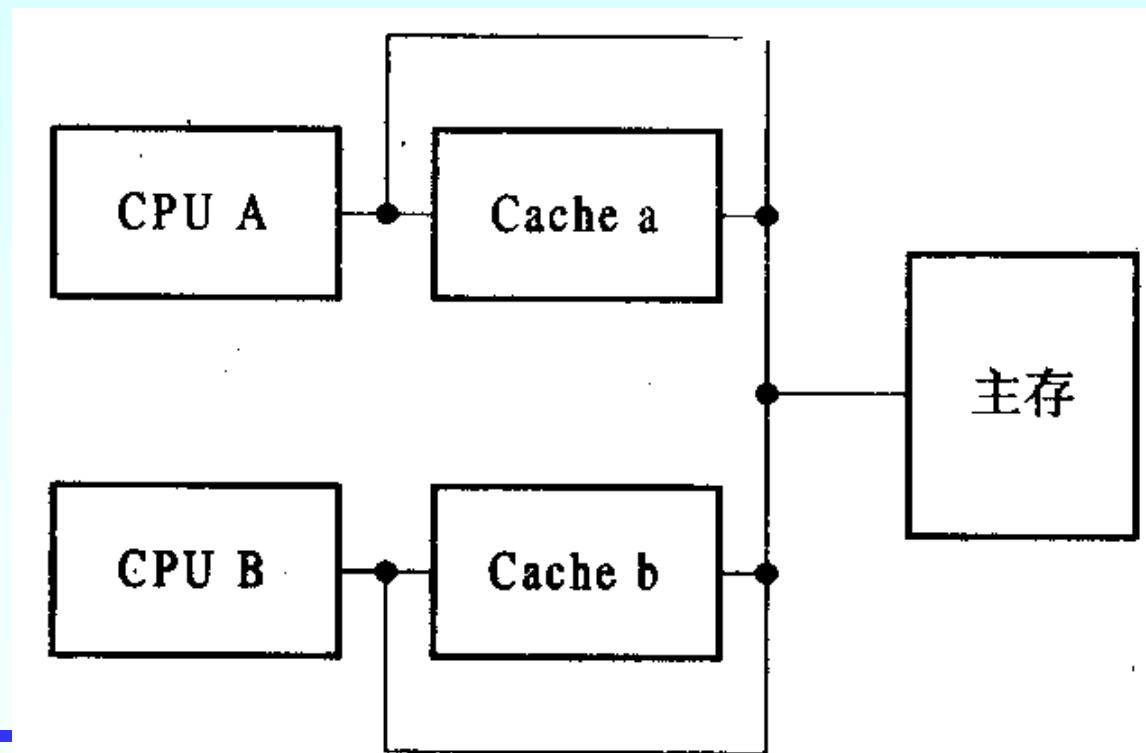


图 每个处理机都有Cache的共享主存多处理机系统

4.3 高速缓冲存储器(Cache)

仅靠写直达法不能保证一致性。例如，CPUA写入CacheA的同时，采用写直达法也写入主存，如果CacheB中也有此单元，则其内容并未变，此时CPU B读出的内容就是原来的内容。

解决的方法：

1. 播写法：任何处理机在要写入Cache时，不仅写入自己的Cache，还把信息播写到所有Cache有此单元的地方或让所有Cache有此单元的块作废。
2. 控制某些共享信息不得进入Cache。
3. 目录表法：在不命中时，先查目录表，以判定目标块是否在别的Cache中，以及是否正在被修改等，然后再决定如何读写。

4.3 高速缓冲存储器(Cache)

2. Cache的预取算法

预取算法有如下几种：

- (1) 按需取：在出现Cache不命中时，把一个块取到Cache中来。
- (2) 恒预取：无论Cache是否命中，都把下一块取到Cache中。
- (3) 不命中预取：当Cache不命中，把本块和下一块一起取到Cache中。

主要考虑因素：

命中率的提高；

Cache与主存之间通信量的增加。



4.3 高速缓冲存储器(Cache)

从模拟实验的结果看：

采用恒预取能使**Cache**的不命中率降低
75~85%。

采用不命中预取能使**Cache**的不命中率
降低**30~40%**。



4.3 高速缓冲存储器(Cache)

Cache所用的取算法基本上仍是按需取进法，即在出现Cache块失效时，才将要访问的字所在的块(行)取进。由于程序存在局部性，只要适当选择好Cache的容量、块的大小、组相联的组数和组内块数，是可以保证有较高的命中率的。然而如辅之以采用在未用到某信息块之前就将其预取进Cache的预取算法，还有进一步提高命中率的可能。

4.3 高速缓冲存储器(Cache)

为了便于硬件实现，通常只预取直接顺序的下一块，即在访问到主存的第*i*块(不论是否已取进Cache)时，只有第*i*+1块才是可能的预取块。至于何时将该块取进，可以有恒预取和不命中时预取两种不同的方法。恒预取指的是只要访问到主存第*i*块的某个字，不论Cache是否命中，恒发预取命令。不命中时预取仅当访问第*i*块不命中时，才发预取命令。Amdahl 470V/8采用的就是不命中时预取法。

4.3 高速缓冲存储器(Cache)

采用预取法并非一定能提高命中率，它还和其他因素有关。

一是块的大小。若每块的字节数过少，预取的效果不明显。从预取需要出发，希望块尽可能增大。但若每块的字节数过多，一方面可能会预取进不需要的信息，另一方面由于Cache的容量有限，又可能把正在使用或近期内就要用到的信息给替换出去，反而降低了命中率。从已有模拟结果来看，每块的字节数如果超过256，就会出现这种情况。

二是预取开销。要预取就要有访主存开销和将它取进Cache的访Cache开销，还要加上把被替换块写回主存的开销。这些开销会增加主存和Cache的负担，干扰和延缓程序的执行。

3. 任务切换对失效率的影响

受限于Cache的容量，多个进程的工作区很难同时保留在Cache内。因此，造成Cache失效的一个重要原因是任务切换。失效率的高低当然就和任务切换的频度有关，或者说与任务切换的平均时间间隔 Q_{sw} 的大小有关。



4.3 高速缓冲存储器(Cache)

由于任务切换引起的Cache失效率可以通过下述几种办法来解决：增大Cache容量；修改调度算法，使任务切换回来之前，有用的信息仍能保留在Cache中不被破坏；设置多个Cache，例如设置两个Cache，一个专用于管理程序，一个专用于用户程序。这样，在管态和目态之间切换时，不会破坏各自Cache中的内容。此外，对于某些操作，例如长的向量运算、长的字符行运算等，可以不经过Cache直接进行，以避免这些操作由于使用Cache，而从Cache中置换出大量更有希望将重新使用的数据。

4.3 高速缓冲存储器(Cache)

4. 影响Cache存储器性能的因素

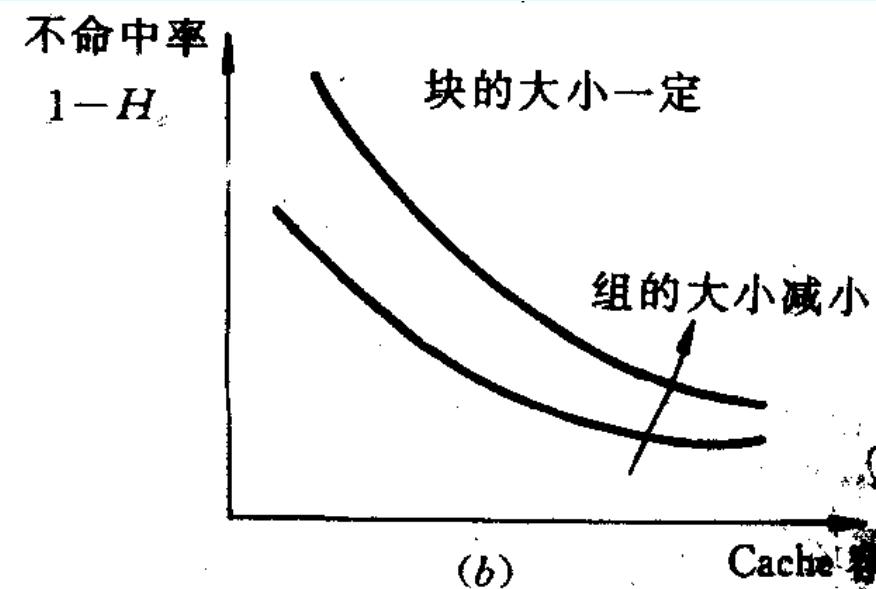
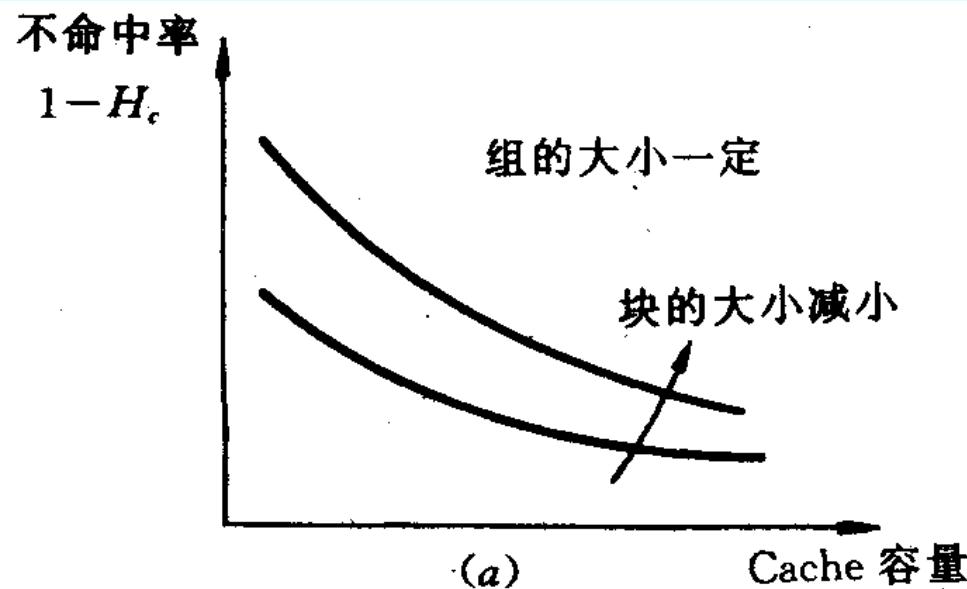
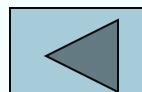


图 块的大小、组的大小与Cache容量对Cache命中率的影响



4.4 Cache—主存—辅存三级层次

以地址变换为例，CPU提供访存的虚地址就可能需要变换成Cache地址、主存地址和辅存地址。如果对应此虚地址的单元已在Cache中，就需把虚地址直接变换成Cache地址，访Cache，而不是先把虚地址变换成主存实地址，再由主存实地址变换成Cache地址，这样可以缩短地址变换的时间。如果对应单元已在主存但尚未调入Cache时，则需把虚地址经快表和慢表变换成主存实地址去访主存，对读访问以及采用按写分配法的写访问还必须进行虚地址到Cache地址的映像或变换，以便把包含对应此单元所在的一块调入或替换进Cache。

4.4 Cache—主存—辅存三级层次

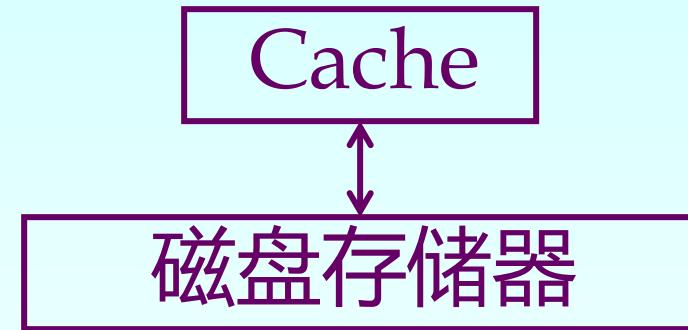
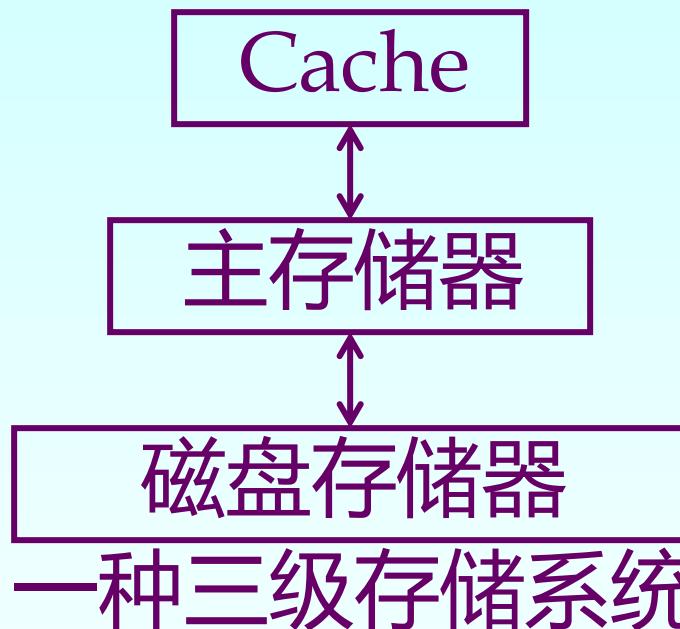
如果对应单元还不在主存，就要把虚地址变换成辅存实地址，去辅存调页，同时，还要将虚地址映像变换成主存实地址将页调入主存，以及把虚地址映像变换成Cache地址，将其中的一块装入Cache。在这种三级层次中通常总是让页的大小恰好是块的2的幂倍，每一块的大小又是字的2的幂倍。而且每次用虚页号查快表和慢表以取得主存实地址和用虚地址对应Cache块号位置的虚块号经组相联去访Cache(Cache中每个单元存放有主存实地址和对应的数据)同时进行。若能在快表中找到，就用由快表来的主存实地址与由Cache中读出的主存实地址相比较。当两者相符，存在Cache中该单元的数据就是要访问的虚、实地址的内容。写Cache的过程与此类似。

4.4 Cache—主存—辅存三级层次

由于每次访Cache，都要查快表，因此，查快表的速度必须尽可能快，不能因为它使访Cache周期延长过多。前面讲过，快表的内容是能随任务切换而变的，因此，Cache的内容也就能正确反映任务的切换。当然，在实际实现中，可以有很多技巧。例如，Cache中每个单元所存的不必是整个主存实地址而只是其某种压缩。也有的机器是和主存实地址完全无关地用虚地址访问Cache。此外也还有一些其他的方案。

4.4 Cache—主存—辅存三级层次

存储系统的组织方式

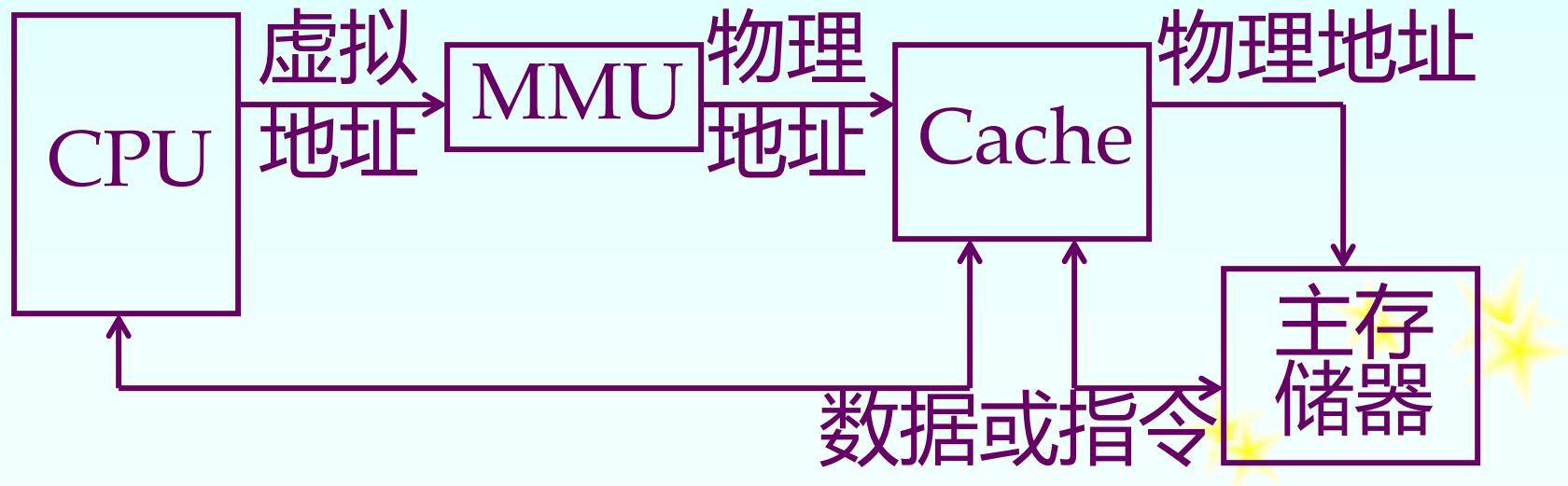


4.4 Cache—主存—辅存三级层次

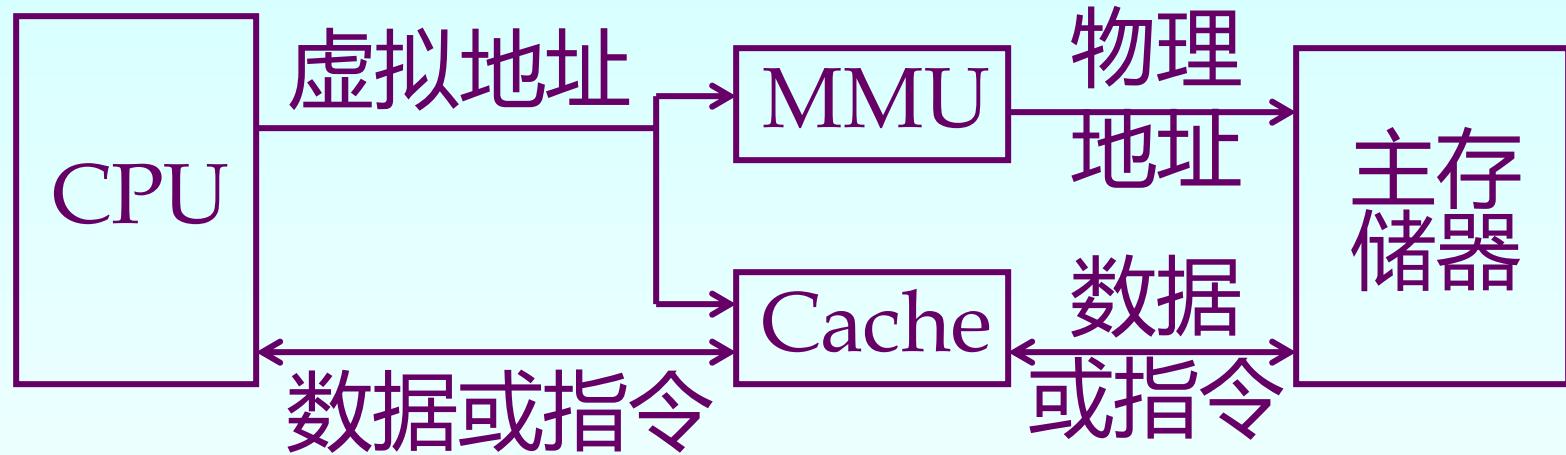
存储系统的几种组织方式：

1、两个存储系统的组织方式：物理地址

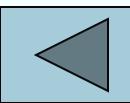
Cache存储系统；目前的大部分处理机均采用这种两级存储系统。



2、一个存储系统组织方式：虚拟地址**Cache**存储系统；如**Intel**公司的*i860*等处理机采用这种组织方式。

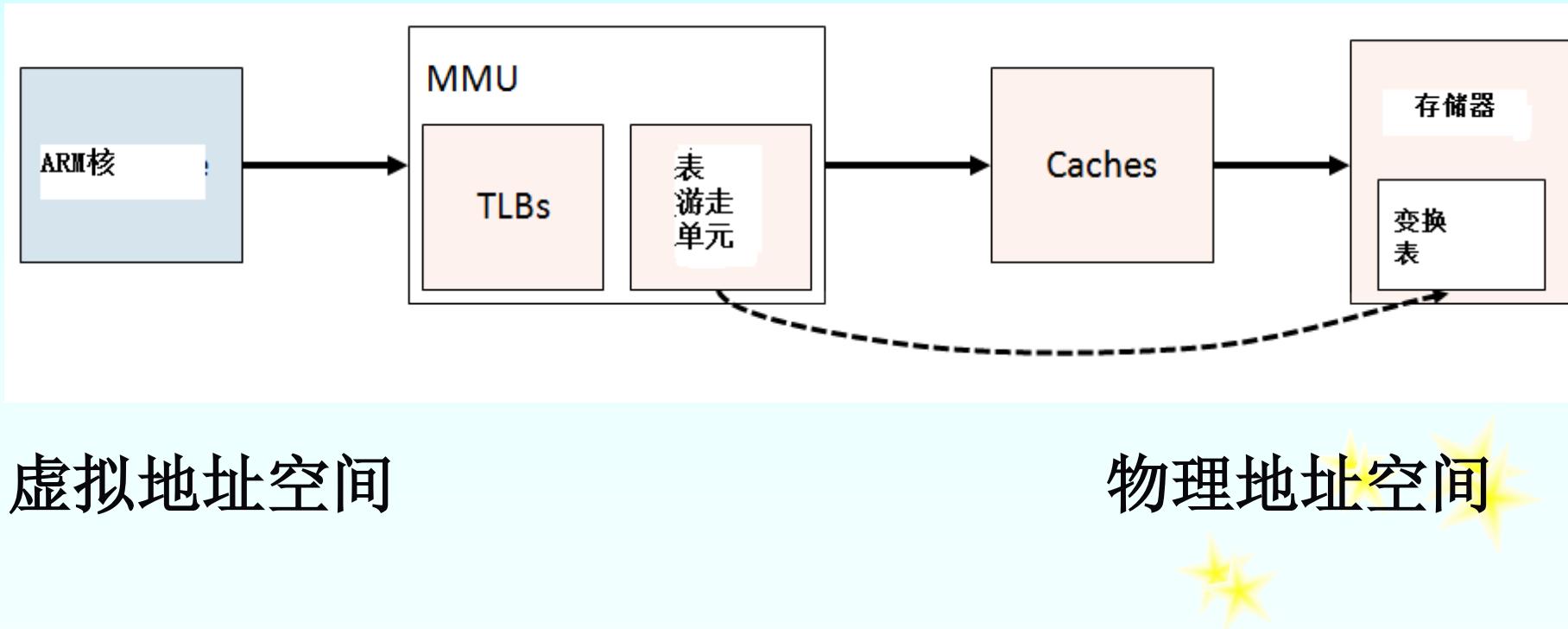


3、全**Cache**系统。没有主存储器，**Cache**—磁盘存储系统。



采用存储器管理单元（The Memory Management Unit, MMU）管理虚拟地址到物理地址的变换，如下图。MMU提供读取存储器中地址变换表的硬件（Table walking），CP15表基址寄存器（Table Base Registers , TTBR）存储表的物理基址，地址变换旁视缓冲器（Translation Look-aside Buffers ， TLBs）缓存（Cache）最近的变换关系。内核可以有分开的指令和数据TLB，或一个共享的统一TLB。当MMU使能时，内核的所有访问都经过MMU。MMU使用TLB缓存（cached）的变换或完成表游走。

4.5 ARM存储系统

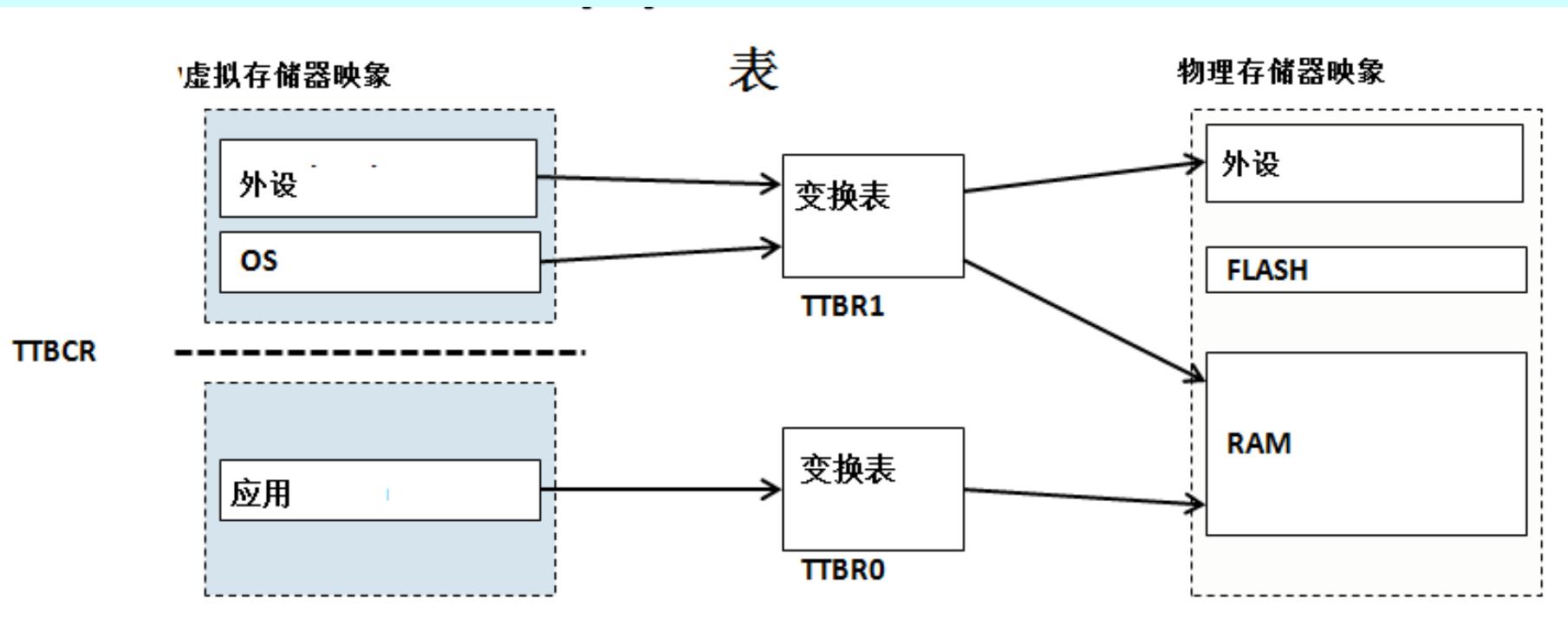


4.5 ARM存储系统

ARM使用的操作系统（OS）和应用变换表如下图所示。虚拟地址空间可以分离（split）成两部分，上半部分由TTBR1指向的表来控制，下半部分由TTBR0指向的表来控制。分离是由变换表基址控制寄存器（**translation Table Base Control Register, TTBCR**）来控制的。TTBR1典型用于操作系统和外设，TTBR0典型用于应用。



4.5 ARM存储系统

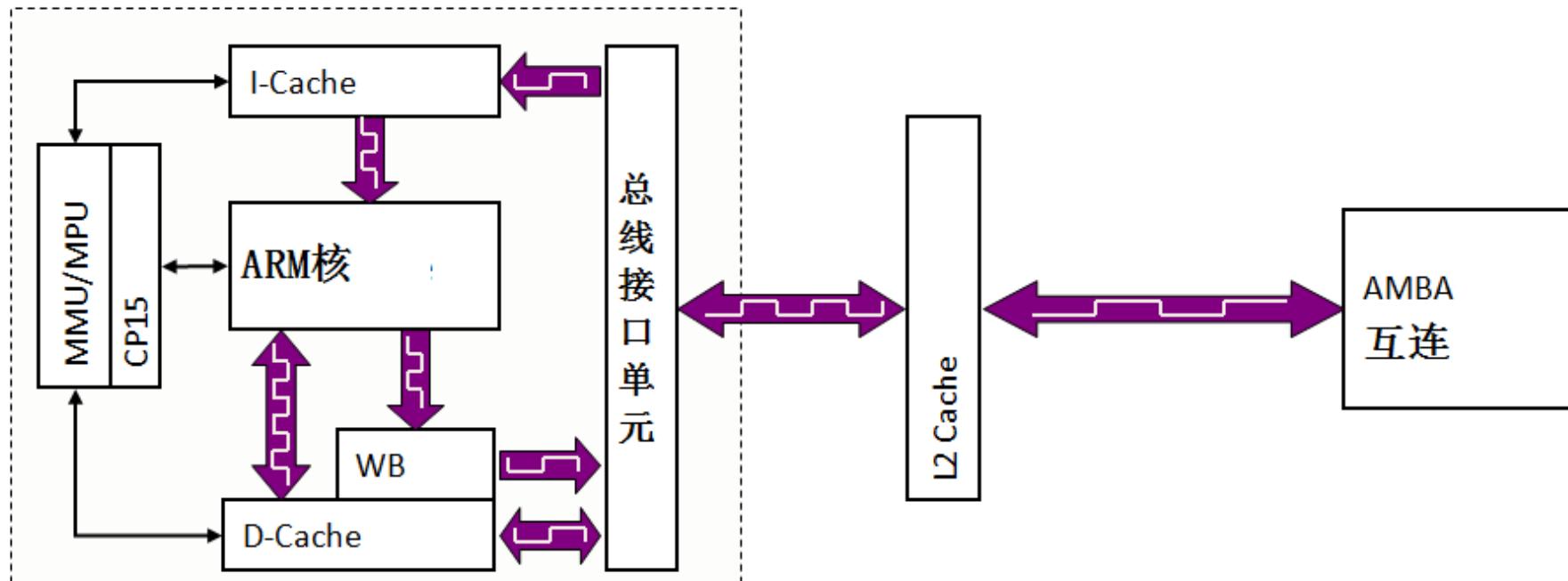


4.5 ARM存储系统

带Cache的ARM宏核（Cached ARM Macrocell）如下图所示。对于存储器管理，ARM核可包含MMU或MPU。存储器管理单元（MMU）实现虚拟存储系统体系结构（Virtual Memory System Architecture，VMSA），存储器保护单元实现物理存储器系统体系结构（Physical Memory System Architecture，PMSA）。



4.5 ARM存储系统



43. (9分) 某32位计算机，**CPU**主频为**800MHz**，**Cache**命中时的**CPI**为4，**Cache**块大小为**32**字节；主存采用**8**体交叉存储方式，每个体的存储字长为**32**位、存储周期为**40ns**；存储器总线宽度为**32**位，总线时钟频率为**200MHz**，支持突发传送总线事务。每次读突发总线事务的过程包括：送首地址和命令、存储器准备数据、传送数据。每次突发传送**32**字节，传送地址或**32**位数据均需要一个总线时钟周期。请回答下列问题，要求给出理由或计算过程。

- ① (1) CPU和总线的时钟周期各为多少？总线的带宽（即最大数据传输率）为多少？
- ② (2) Cache缺失时，需要用几个读突发传送总线事务来完成一个主存块的读取？
- ③ (3) 存储器总线完成一次读突发传送总线事务所需的时间是多少？
- ④ (4) 若程序BP执行过程中，共执行了100条指令，平均每条指令需进行1.2次访存，Cache的缺失率为5%，不考虑替换等开销，则BP的CPU执行时间是多少？

答：

(1) CPU的时钟周期：

$1/800\text{MHz} = 1.25\text{ns}$ 或 秒， 或 秒。 (1分)

总线的时钟周期： $1/200\text{MHz} = 5\text{ns}$ 或 秒，
或 秒。 (1分)

总线的带宽： $4B \times 200\text{MHz} = 800\text{MB/s}$ 或
者 $4B/5\text{ns} = 800\text{MB/s}$

或者 (1分)

(2) 需要1个突发传送总线事务 (1分)。



(3) $40\text{ns}/8=5\text{ns}$ 启动一个体工作，用8个总线时钟周期读突发传送总线事务。

$$5\text{ns} + 40\text{ns} + 8 \times 5\text{ns} = 85\text{ns} \quad (2\text{分})$$

(4) 命中时指令执行时间：

$$100 \times 4 \times 1.25\text{ns} = 500\text{ns} \quad (1\text{分})$$

指令执行过程中Cache缺失时的额外开销

$$1.2 \times 100 \times 5\% \times 85\text{ns} = 510\text{ns}$$

BP的CPU执行时间为 $500 + 510 = 1010\text{ns}$
(2分)。

所有指令的执行时间+Cache缺失时额外开销。

本章重点

并行存储器和交叉访问存储器的工作原理

存储系统的定义

存储系统的性能参数

段式、页式、段页式虚拟存储管理的特点

虚拟存储系统的页面替换算法

页式虚拟存储系统的工作原理



虚拟存储系统中加快地址变换的方法

Cache存储系统地址映像及变换方法

Cache存储系统的块替换算法

Cache存储系统的一致性问题



术语解释

地址的变换、存储层次（体系）、程序局部性

