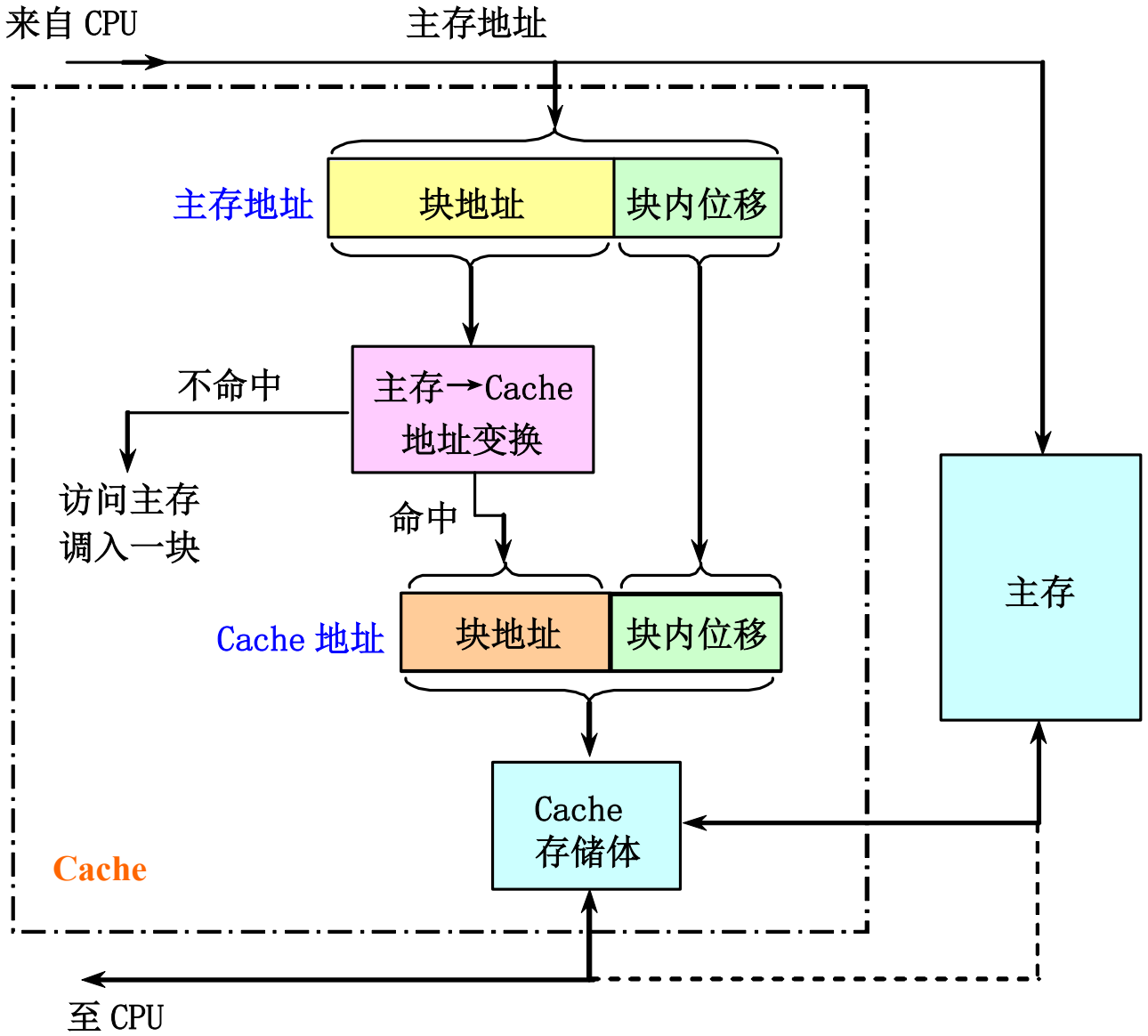
1.2 Cache基本知识

■ 基本结构和原理

- **Cache**是按块进行管理的。**Cache**和主存均被分割成大小相同的块。信息以块为单位调入**Cache**。
  - 主存块地址（块号）用于查找该块在**Cache**中的位置。
  - 块内位移用于确定所访问的数据在该块中的位置。

Cache的基本工作原理示意图



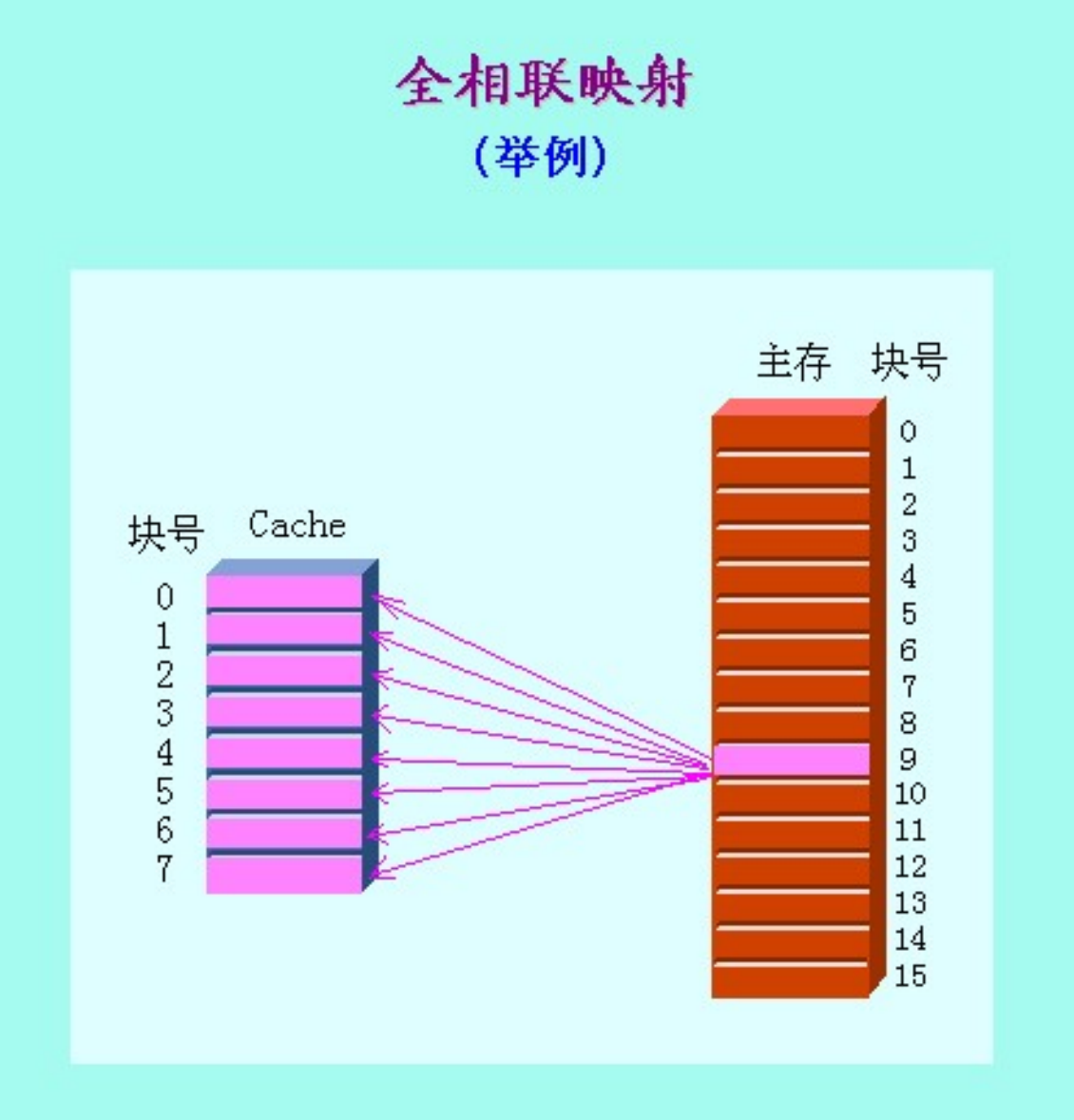
1.2 Cache基本知识

■ 映象规则

➢ 全相联映象

- 全相联：主存中的任一块可以被放置到 **Cache**中的任意一个位置。
- 特点：空间利用率最高，冲突概率最低，实现最复杂。

1.2 Cache基本知识



1.2 Cache基本知识

- 直接映象
  - 直接映象：主存中的每一块只能被放置到Cache中唯一的一个位置。
  - 特点：空间利用率最低，冲突概率最高，实现最简单。
  - 对于主存的第 i 块，若它映象到Cache的第 j 块，则：

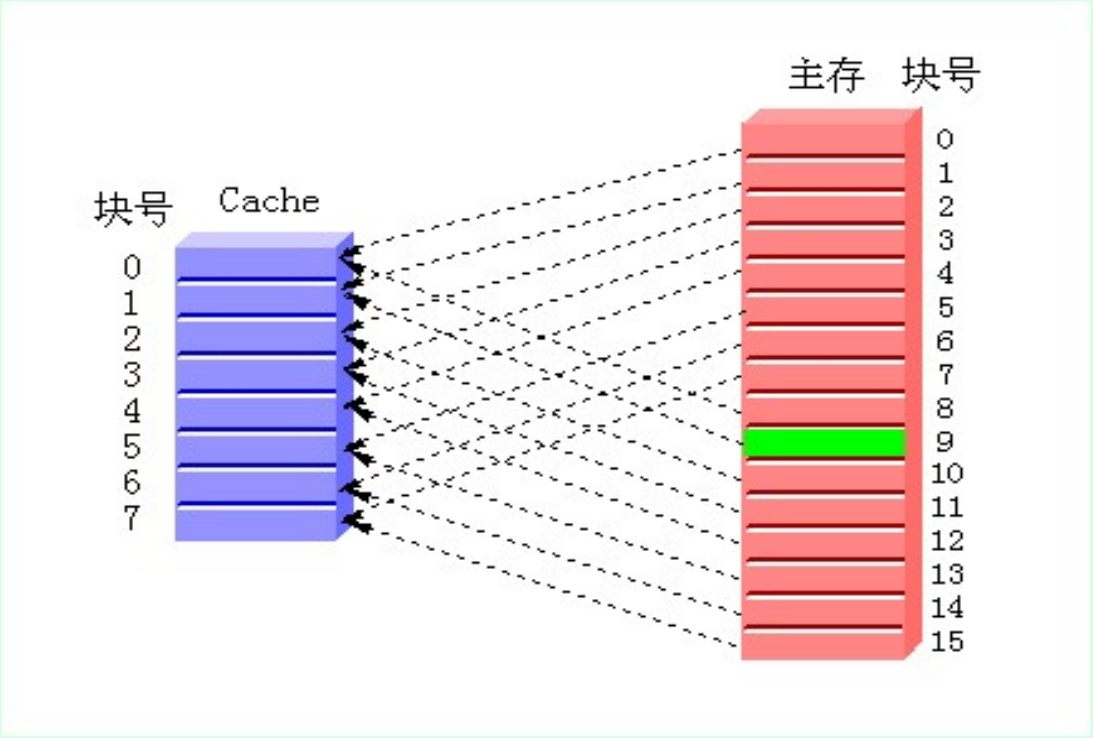
$$j = i \bmod (M) \quad （M 为Cache的块数）$$

设 $M=2^m$，则当表示为二进制数时，j 实际上就是 i 的低 m 位：

1.2 Cache基本知识

直 接 映 射

（举 例）



1.2 Cache基本知识

➢ 组相联映像

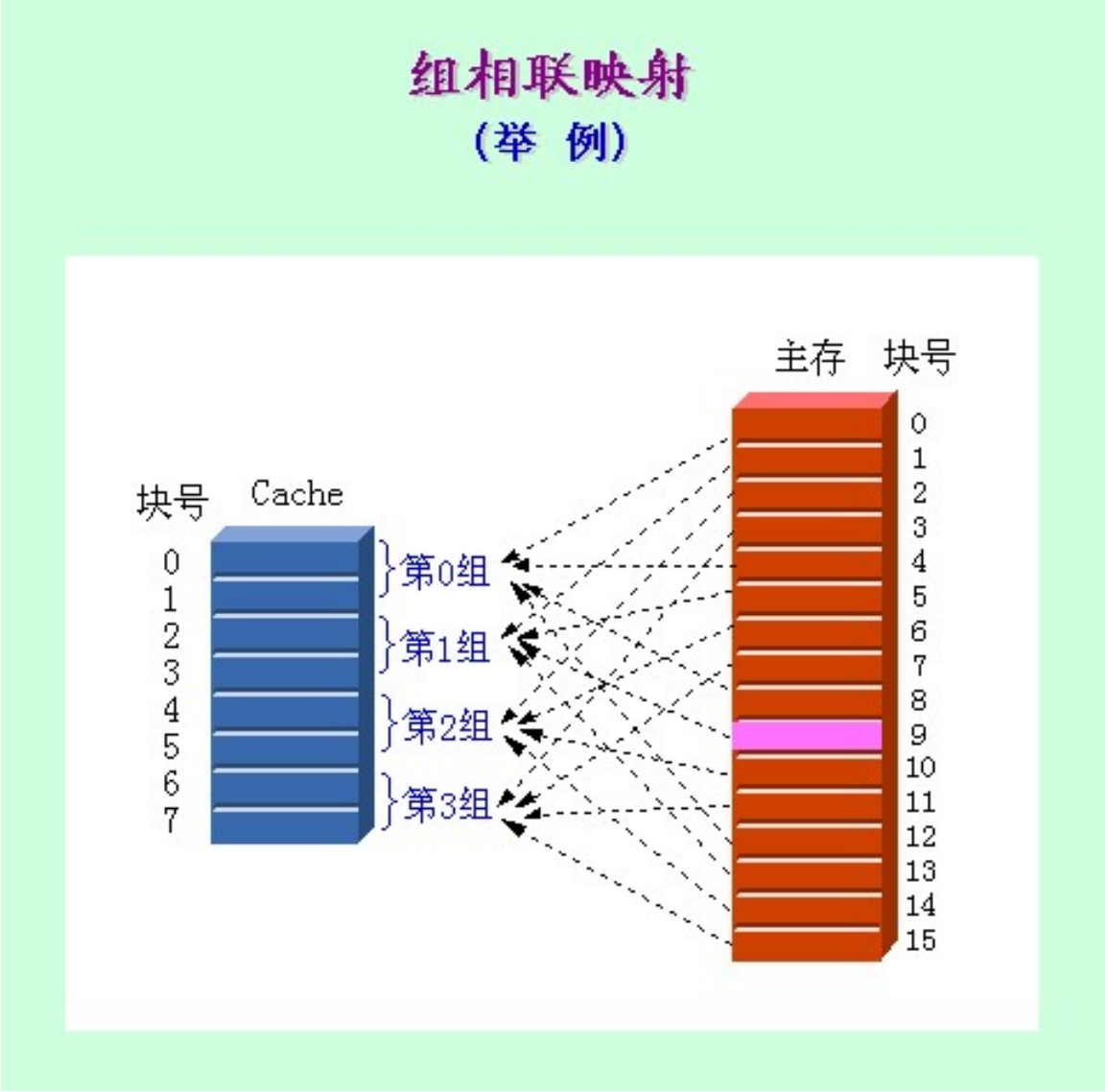
- 组相联：主存中的每一块可以被放置到Cache中唯一的一个组中的任何一个位置。
- 组相联是直接映像和全相联的一种折中
- 若主存第i 块映像到第k 组，则：

  k＝i mod（G）　（G为Cache的组数）

  设$G=2^g$，则当表示为二进制数时，k 实际上就是 i 的低 g 位：

- 低g位以及直接映像中的低m位通常称为索引。

1.2 Cache基本知识



1.2 Cache基本知识

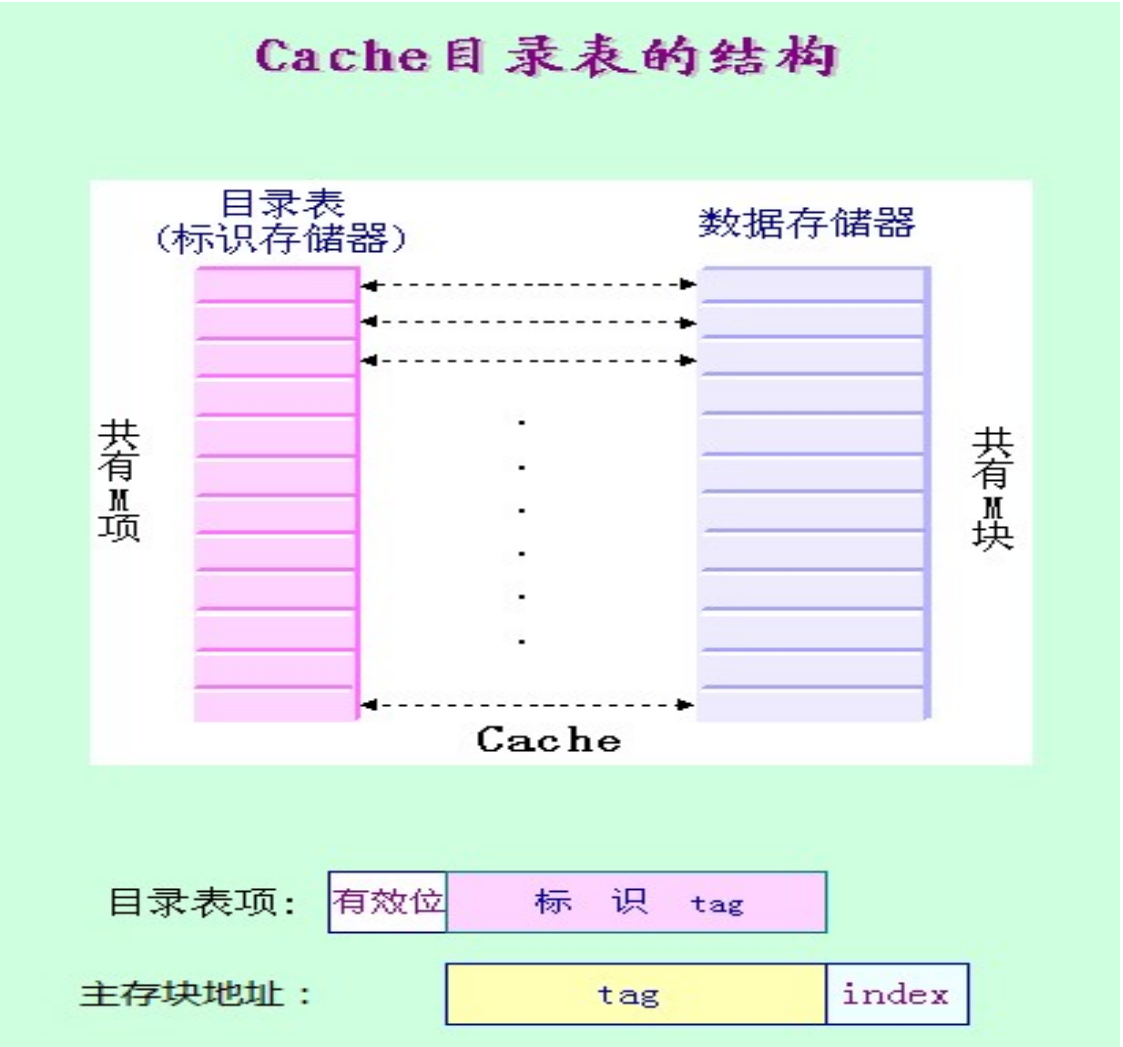
■ 查找方法

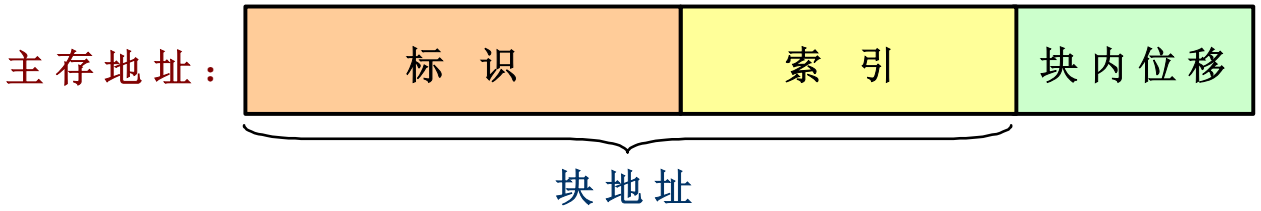
所要解决的问题：如何确定**Cache**中是否有所要访问的块？若有的话如何确定其位置？

➢ 通过查找目录表来实现：

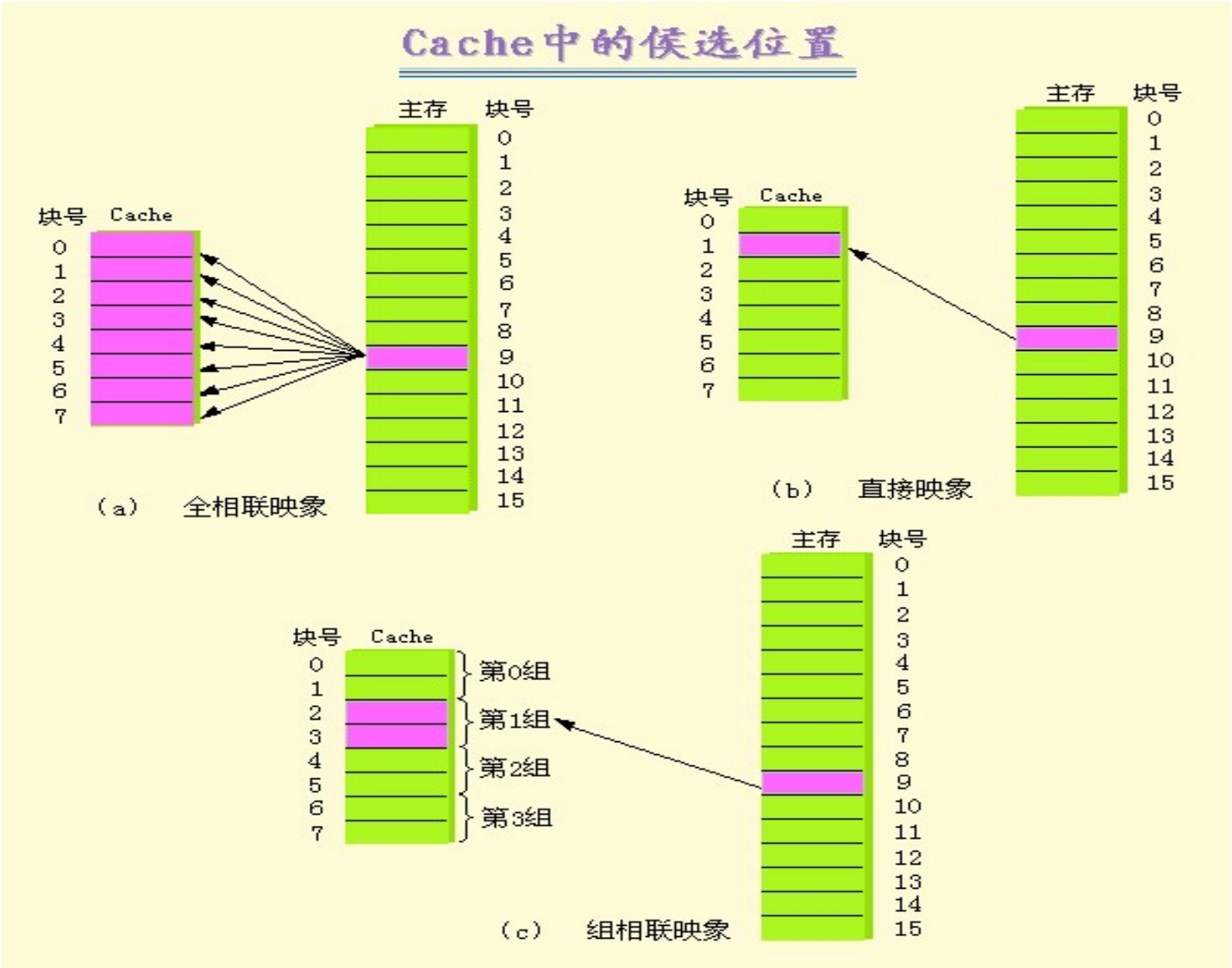
在**cache**中设有一个目录表，该表共有**M**项，每一项对应于**cache**中的一个块，用于指出当前该块的信息是哪个主存块的。

目录表的结构：

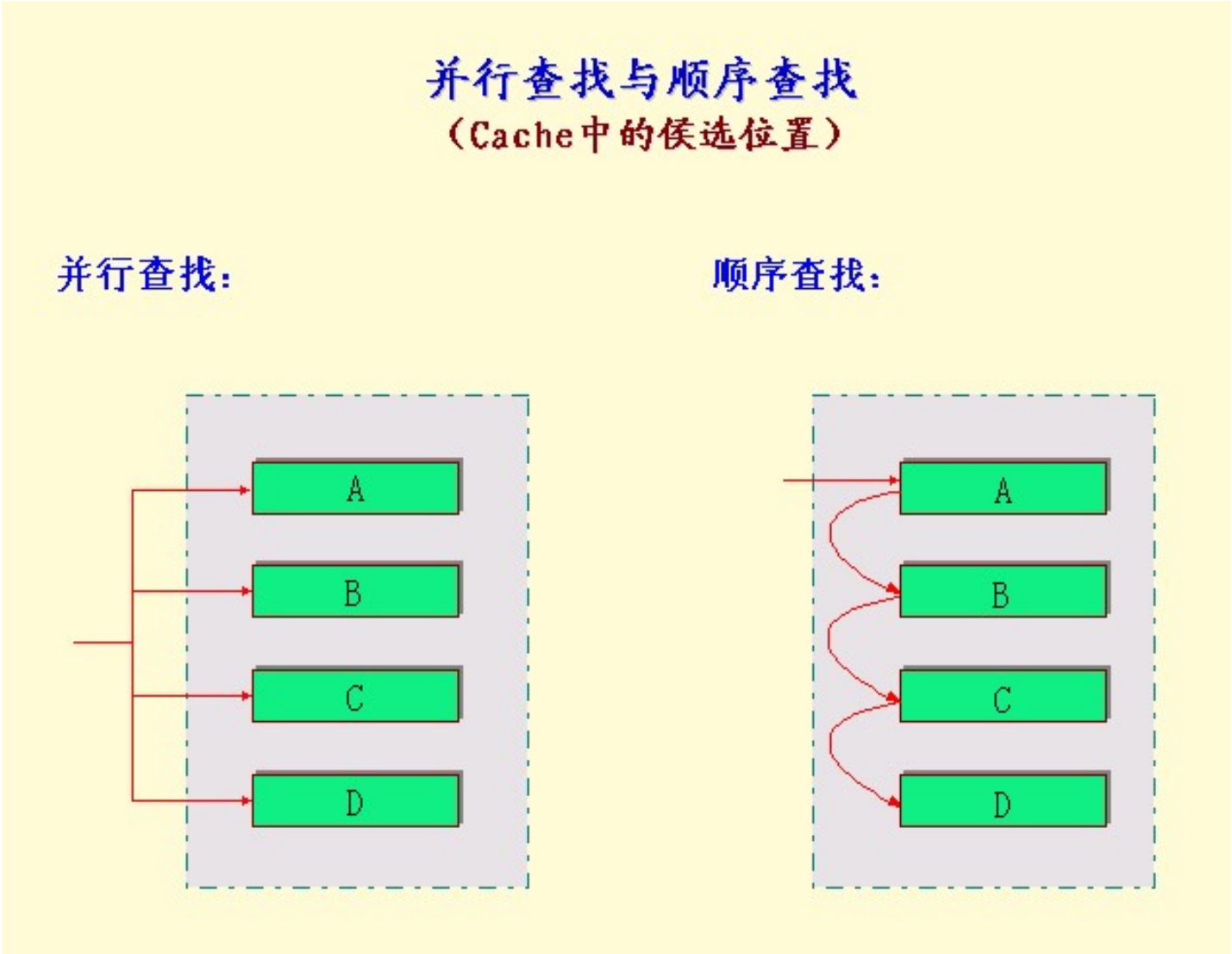

Cache目录表的结构
目录表
(标识存储器)
数据存储器
共有M项
共有M块
Cache
目录表项：
有效位
标 识 tag
主存块地址：
tag
index


主存地址：
标 识
索 引
块内位移
块地址

查找范围：
候选位置
所对应的
目录表项

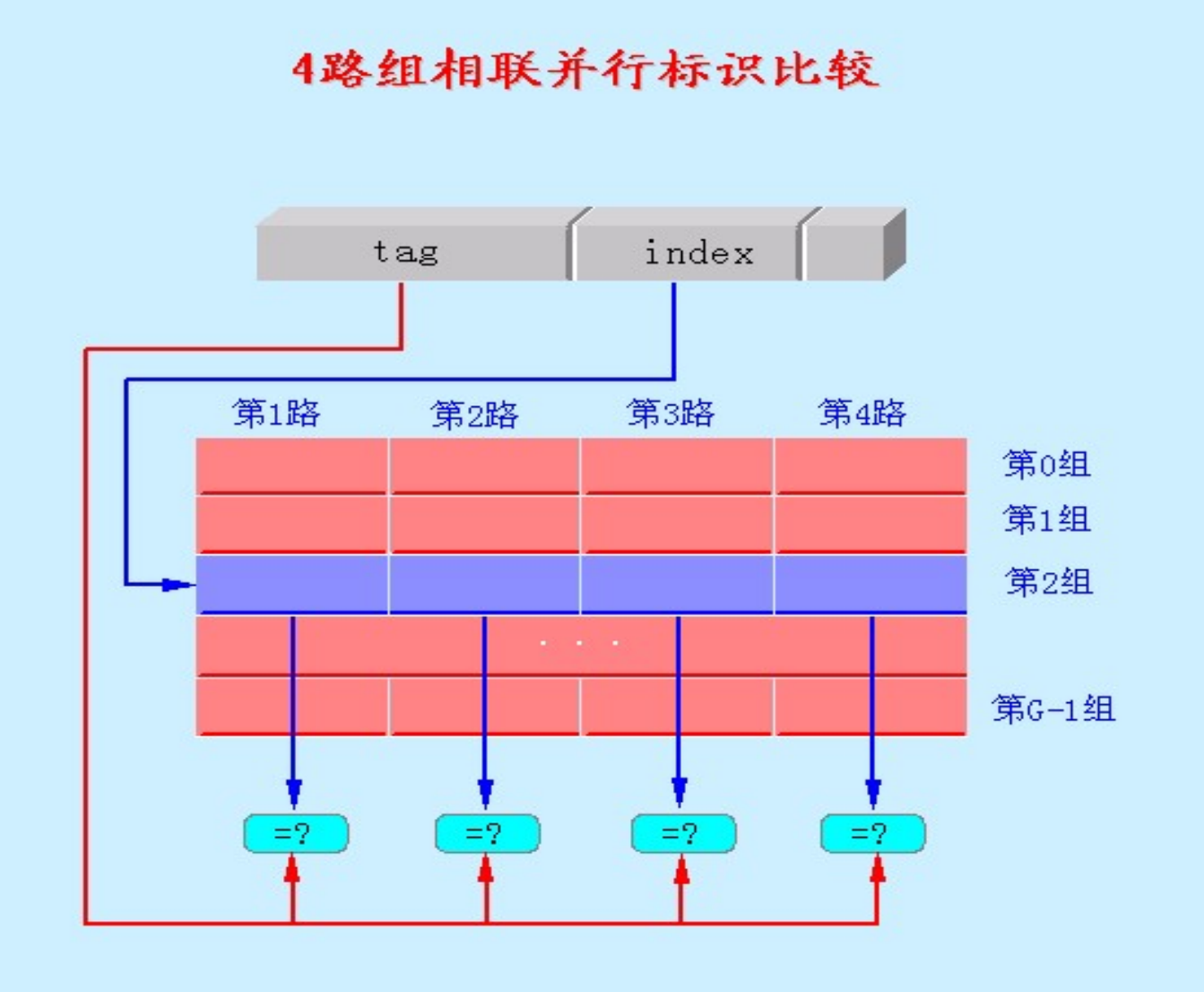


并行查找与顺序查找

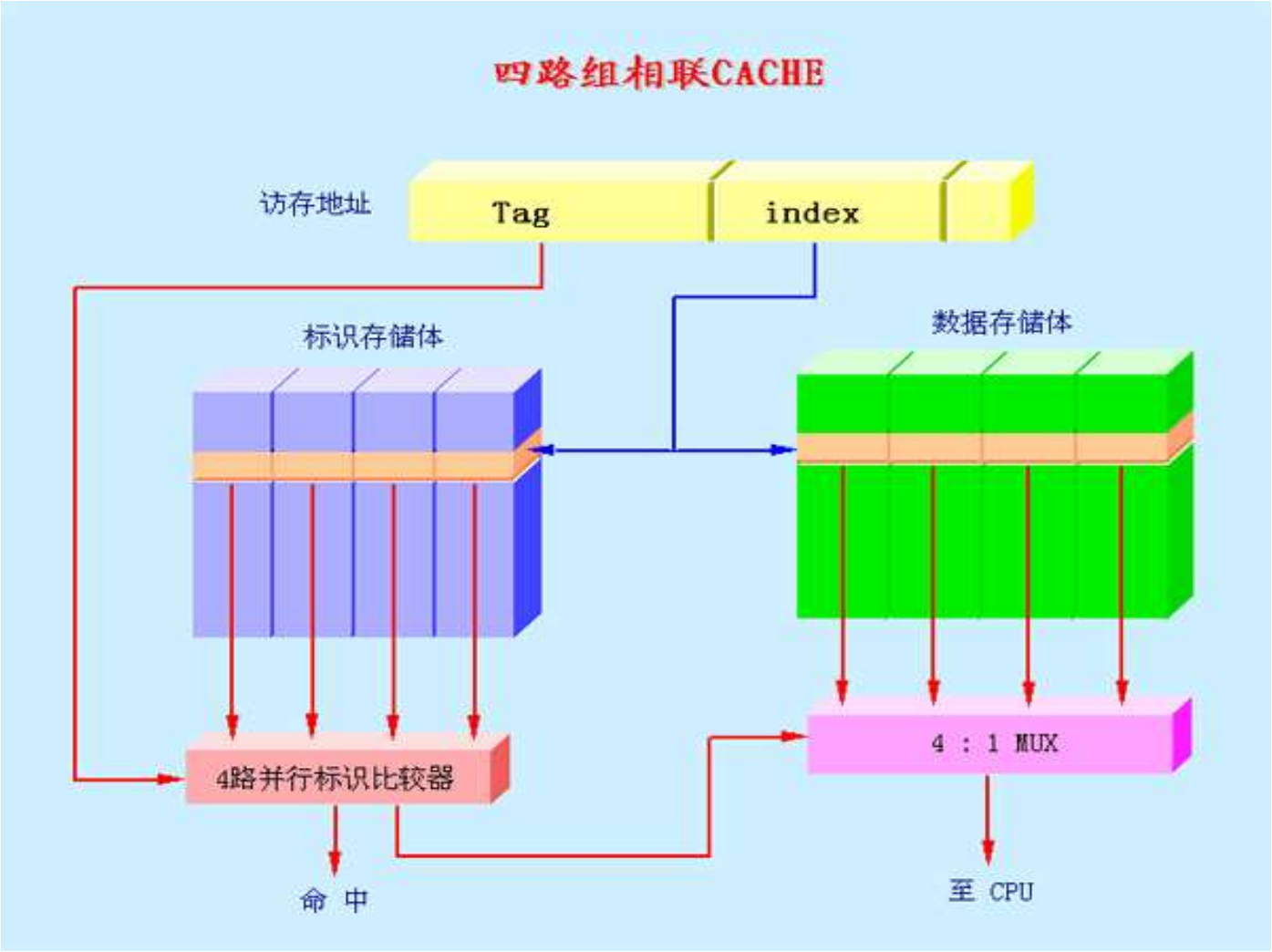


举例：

4 路组相联并行标识比较



4 路相联Cache的查找过程



1.2 Cache基本知识

■ 替换算法

- 所要解决的问题：当新调入一块，而Cache又已被占满时，替换哪一块？
  - 直接映像Cache中的替换很简单，因为只有一个块，别无选择。
  - 在组相联和全相联Cache中，则有多个块供选择。
- 主要的替换算法有三种
  - 随机法
  - 先进先出法FIFO
  - 最近最少使用法LRU

1.2 Cache基本知识

- **LRU**和随机法分别因其不命中率低和实现简单而被广泛采用。
  - 模拟数据表明，对于容量很大的**Cache**，**LRU**和随机法的命中率差别不大。

1.2 Cache基本知识

■ 写策略

写策略是区分不同**Cache**设计方案的一个重要标志。

- 两种写策略
  - 写直达法：执行“写”操作时，不仅写入Cache，而且也写入下一级存储器。
  - 写回法：执行“写”操作时，只写入Cache。仅当Cache中相应的块被替换时，才写回主存。（设置“修改位”）
    - 两种写策略的比较
  - 写回法的优点：速度快，所使用的存储器带宽较低。
  - 写直达法的优点：易于实现，数据一致性好。

1.2 Cache基本知识

- 采用写直达法时，若在进行“写”操作的过程中CPU必须等待，直到“写”操作结束，则称CPU写停顿。
  - 减少写停顿的一种常用的优化技术：采用写缓冲器
- “写”操作时的调块
  - 按写分配(写时取)：写不命中时，先把所写单元所在的块调入Cache，再行写入。
  - 不按写分配(绕写法)：写不命中时，直接写入下一级存储器而不调块。
- 写策略与调块
  - 写回法 —— 按写分配
  - 写直达法 —— 不按写分配

1.2 Cache基本知识

例　5个存储器操作组成的序列：

Write Mem[100]

Write Mem[100]

Read Mem[200]

Write Mem[200]

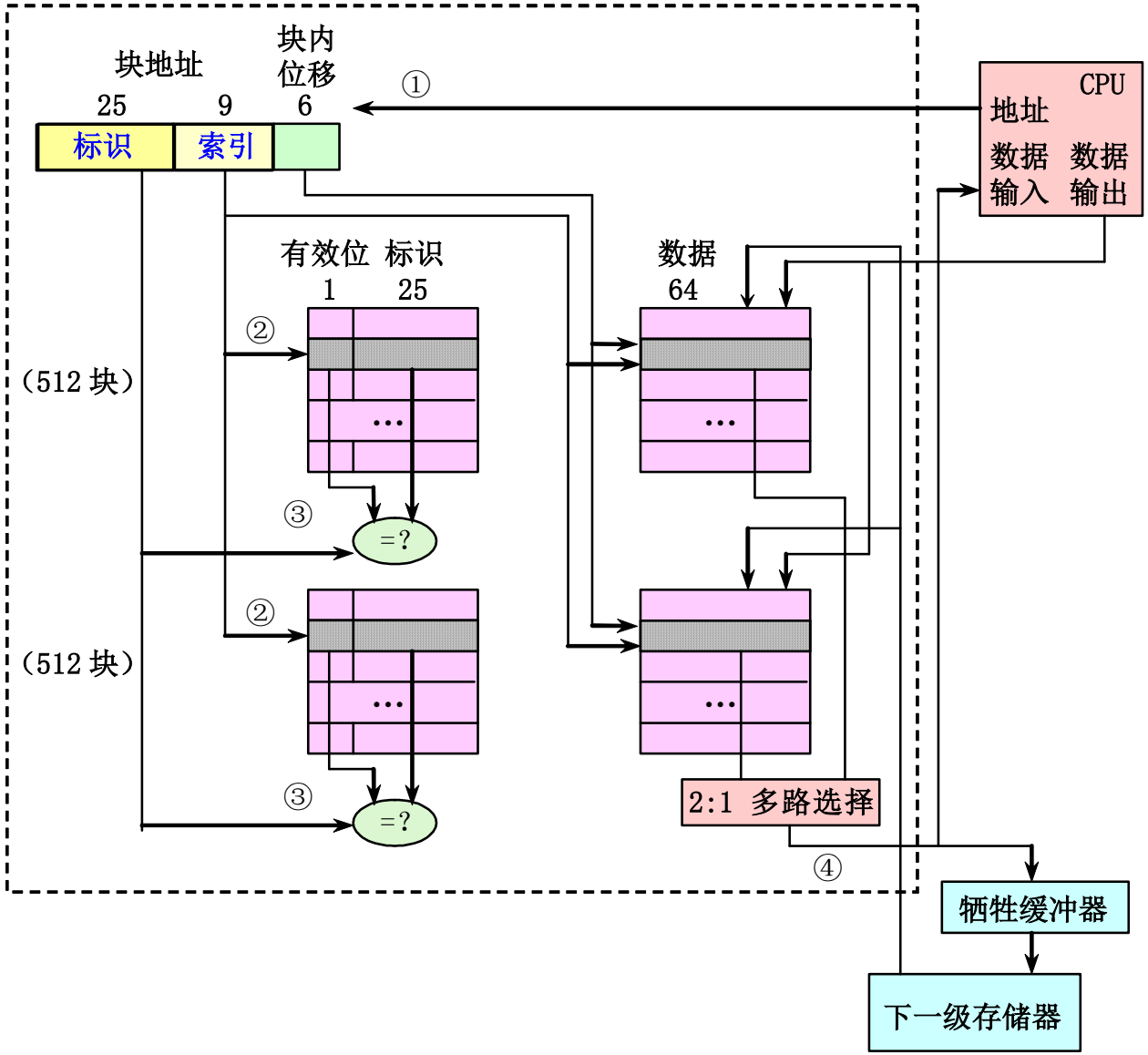
Write Mem[100]

在按写分配和不按写分配时，命中数和缺失数为多少？

解：不按写分配：**4**次缺失，**1**次命中

按写分配：**2**次缺失，**3**次命中

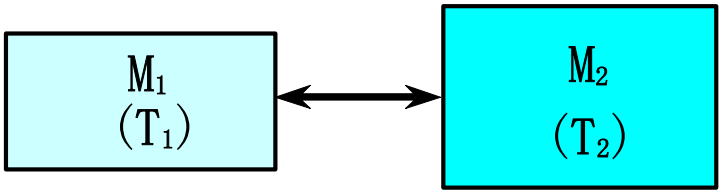
1.3 Opteron数据缓存



1.4 缓存的性能分析

■ 命中率H和不命中率F

考虑由**M1**和**M2**构成的两级存储层次：



T_1 ：M_1的访问时间，T_2：M_2的访问时间

□命中率：**CPU**访问存储系统时，在**M1**中找到所需信息的概率。

$$H=\frac{N_1}{N_1+N_2}$$

N_1 ：访问M_1的次数，N_2：访问M_2的次数

□ 缺失率（不命中率）：**F＝1－H**

1.4 缓存（Cache）的性能分析

■ 平均访问时间T_A

$$T_A = HT_1 + (1-H)(T_1+T_M)$$
$$= T_1 + (1-H)T_M$$
$$或\ T_A = T_1 + FT_M$$

分两种情况来考虑**CPU**的一次访存：

- 当命中时，访问时间即为T_1（命中时间）
- 当M_1不命中时，从M_2取数据：

  不命中时的访问时间为：$T_2+T_B+T_1=T_1+T_M$

  $$T_M = T_2+T_B$$

  - ✓ 不命中代价T_M：从向M_2发出访问请求到把整个数据块调入M_1中所需的时间。
  - ✓ 传送一个信息块所需的时间为T_B。

1.4 缓存（Cache）的性能分析

■ CPU执行时间

CPU执行时间＝（CPU执行周期数+存储器停顿周期数）× 时钟周期时间

其中：

- 存储器停顿时钟周期数＝缺失次数×缺失代价
- 存储器停顿时钟周期数＝“读”的次数×读缺失率×读缺失代价＋“写”的次数×写缺失率×写缺失代价
- 存储器停顿时钟周期数＝访存次数×缺失率×缺失代价

CPU执行时间＝（**CPU执行周期数+缺失次数×缺失代价**）

× 时钟周期时间

$$CPU\text{执行时间} = IC \times \left(CPI_{execution} + \frac{\text{缺失次数}}{\text{指令数}} \times \text{缺失代价} \right) \times \text{时钟周期时间}$$

1.4 缓存（Cache）的性能分析

CPU执行时间＝（**CPU执行周期数+访存次数×缺失率×缺失代价**）

× 时钟周期时间

$$CPU\text{执行时间} = IC \times \left(CPI_{execution} + \frac{\text{访存次数}}{\text{指令数}} \times \text{缺失率} \times \text{缺失代价} \right) \times \text{时钟周期时间}$$

=*IC*×（*CPI*execution＋每条指令的平均访存次数×缺失率

×缺失代价）× 时钟周期时间

1.4 缓存（Cache）的性能分析

例1 16KB指令Cache加上16KB数据Cache，与一个32KB统一Cache相比，哪种缺失率较低？利用表B-3中的缺失率数据，假定36%的指令为数据传输指令。假定一次命中需要1个时钟周期，缺失代价为100个时钟周期。对于统一缓存，如果仅有一个缓存端口，无法同时满足两个请求，一次 load 或 store 操作访问 Cache 的命中时间都要增加一个时钟周期，按照有关流水线的术语，统一Cache会导致结构冒险。又假设采用写直达策略，且有一个写缓冲器，并且忽略写缓冲器引起的等待。请问上述两种情况下平均访存时间各是多少？

1.4 缓存（Cache）的性能分析

- 分离Cache与统一Cache平均缺失率的比较

前提：指令Cache容量+数据Cache容量

= 统一Cache容量

- 分离Cache平均缺失率的计算：

访问指令Cache的百分比×指令Cache的缺失率

＋访问数据Cache的百分比×数据Cache的缺失率

访问指令Cache的百分比：

100%/(100%＋26%＋10%)≈74%

访问数据Cache的百分比：

(26%＋10%)/(100%＋26%＋10%)≈26%

1.4 缓存（Cache）的性能分析

解：

$$\text{缺失率} = \frac{\text{每条指令的缺失数}}{\text{每条指令的访存数}} = \frac{\text{每千条指令的缺失数}/1000}{\text{访存数}/\text{指令数}}$$

$$\text{缺失率}_{16KB\text{指令}} = \frac{3.82/1000}{1.0} = 0.004$$

$$\text{缺失率}_{16KB\text{数据}} = \frac{40.9/1000}{0.36} = 0.114$$

$$\text{缺失率}_{32KB\text{统一}} = \frac{43.3/1000}{1.0+0.36} = 0.0318$$

分离 **Cache** 的总体缺失率为：

(74%×0.004)＋(26%×0.114)＝0.0326

比容量为 **32KB** 的统一 **Cache** 的缺失率略高一些，

1.4 缓存（Cache）的性能分析

存储器平均访问时间＝指令所占的百分比×
(指令命中时间＋指令缺失率×缺失代价)＋
数据所占的百分比×
(数据命中时间＋数据缺失率×缺失代价)

平均访存时间$_{分离}$＝74%×(1＋0.004×100)＋
26%×(1＋0.114×100)
＝(74%×1.4)＋(26%×12.4)=4.26

平均访存时间$_{统一}$＝74%×(1＋0.0318×100)＋
26%×(1＋1 ＋ 0.0318×100)
＝(74%×4.18)＋(26%×5.18)=4.44

1.4 缓存（Cache）的性能分析

例2 假设Cache缺失代价为200个时钟周期，当不考虑存储器停顿时，所有指令的执行时间都是1.0个时钟周期，访问Cache缺失率为2%，平均每条指令访存1.5次，每千条指令的平均缓存缺失数为30。试分析Cache对性能的影响。用每条指令的缺失数及缺失率来计算。

解：用缺失率计算

$$\text{CPU时间}_{\text{有cache}} = IC \times (CPI\text{execution} + \text{每条指令的平均访存次数} \times \text{缺失率} \times \text{缺失代价}) \times \text{时钟周期时间}$$

$$= IC \times (1.0 + 1.5 \times 2\% \times 200) \times \text{时钟周期时间}$$

$$= IC \times 7 \times \text{时钟周期时间}$$

1.4 缓存（Cache）的性能分析

用缺失数计算

$$CPU\text{执行时间} = IC \times \left(CPI_{execution} + \frac{\text{缺失次数}}{\text{指令数}} \times \text{缺失代价} \right) \times \text{时钟周期时间}$$

$$\text{CPU时间}_{\text{有cache}} = \text{IC} \times (1.0 + 30/1000 \times 200) \times \text{时钟周期时间}$$

$$= \text{IC} \times 7 \times \text{时钟周期时间}$$

理想CPI=1，实际CPI =7, CPU时间是理想情况的7倍

但若不采用Cache，则：

$$\text{CPI} = 1.0 + 200 \times 1.5 = 301$$

是有Cache系统的CPU时间的40多倍。

1.4 缓存（Cache）的性能分析

例3 考虑两种不同组织结构的Cache：直接映像Cache和两路组相联Cache，试问它们对CPU的性能有何影响？先求平均访存时间，然后再计算CPU性能。分析时请用以下假设：

(1)理想Cache（命中率为100%）情况下的CPI为1.6，时钟周期为0.35ns，平均每条指令访存1.4次。

(2)两种Cache容量均为128KB，块大小都是64字节。

(3)在组相联Cache中，由于多路选择器的存在而使**CPU**的时钟周期增加到原来的**1.35**倍，才能与组相联Cache的多路选择器相适应。

(4) 这两种结构Cache的缺失代价都是65ns。（在实际应用中，应取整为整数个时钟周期）

(5) 命中时间为1个时钟周期，128KB直接映像Cache的缺失率为2.1%，相同容量的两路组相联Cache的缺失率为1.9%。

1.4 缓存（Cache）的性能分析

解：平均访存时间 = 命中时间+缺失率×缺失代价

平均访存时间$_{直接}$ =1×0.35+(0.021×65)=1.715ns

平均访存时间$_{组相联}$=1×0.35×1.35 +(0.019×65) = 1.708ns

CPU时间＝IC×[CPIexe＋访存次数/指令数×缺失率×缺失代价]×时钟周期时间

＝IC×[CPIexe×时钟周期时间＋访存次数/指令数×缺失率×缺失代价×时钟周期时间]

1.4 缓存（Cache）的性能分析

$$CPU时间_{直接} = IC[1.6 \times 0.35 + (1.4 \times 0.021 \times 65)] = 2.47 \times IC$$

$$CPU时间_{组相联} = IC[1.6 \times 0.35 \times 1.35 + (1.4 \times 0.019 \times 65)$$

$$= 2.49 \times IC$$

1.5 6种基本的缓存（Cache）优化

- 平均访存时间＝命中时间＋缺失率×缺失代价

可以从三个方面改进Cache的性能：

- 降低缺失率
- 减少缺失代价
- 减少Cache命中时间

1.5 6种基本的缓存（Cache）优化

- 三种类型的缺失（3C）
  - 强制性缺失(Compulsory miss)

    当第一次访问一个块时，该块不在Cache中，需从下一级存储器中调入Cache，这就是强制性缺失。（冷启动缺失，首次访问缺失）
  - 容量缺失(Capacity miss)

    如果程序执行时所需的块不能全部调入Cache中，则当某些块被替换后，若又重新被访问，就会发生缺失。这种不命中称为容量缺失。

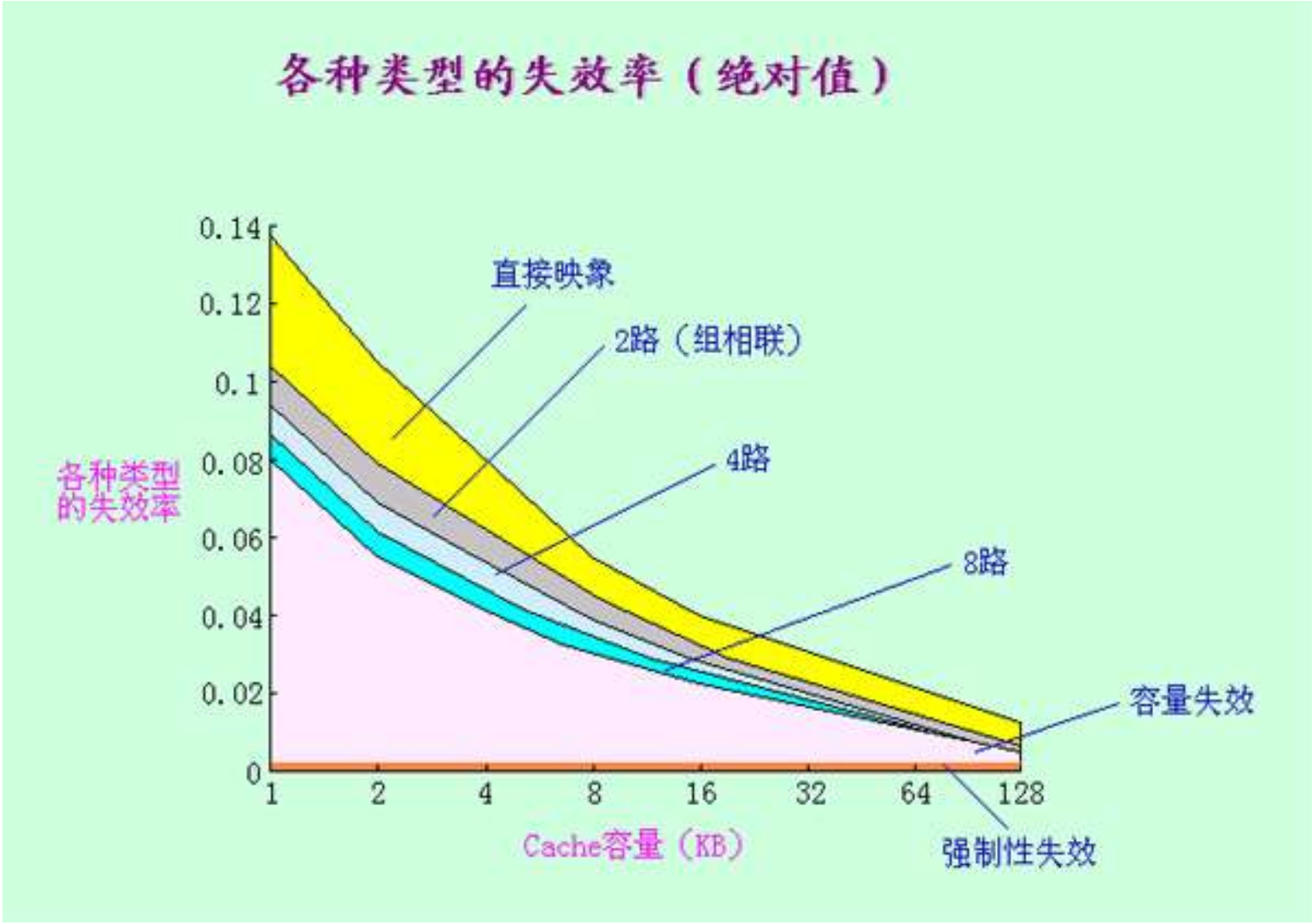
1.5 6种基本的缓存（Cache）优化

- 冲突缺失

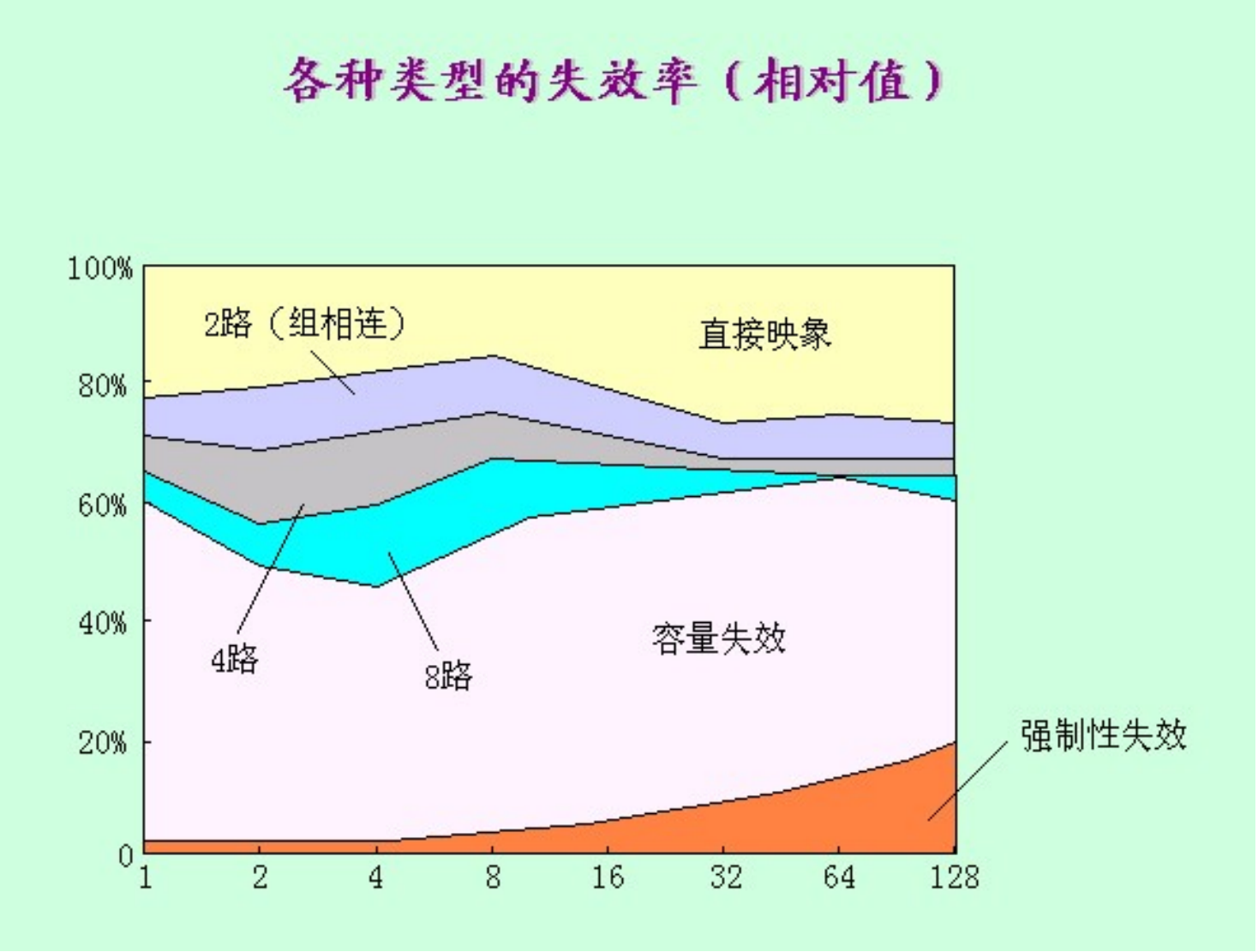
  在组相联或直接映像Cache中，若太多的块映像到同一组(块)中，则会出现该组中某个块被别的块替换(即使别的组或块有空闲位置)，然后又被重新访问的情况。这就是发生了冲突缺失（碰撞缺失）

■ 三种缺失所占的比例

1.5 6种基本的缓存（Cache）优化

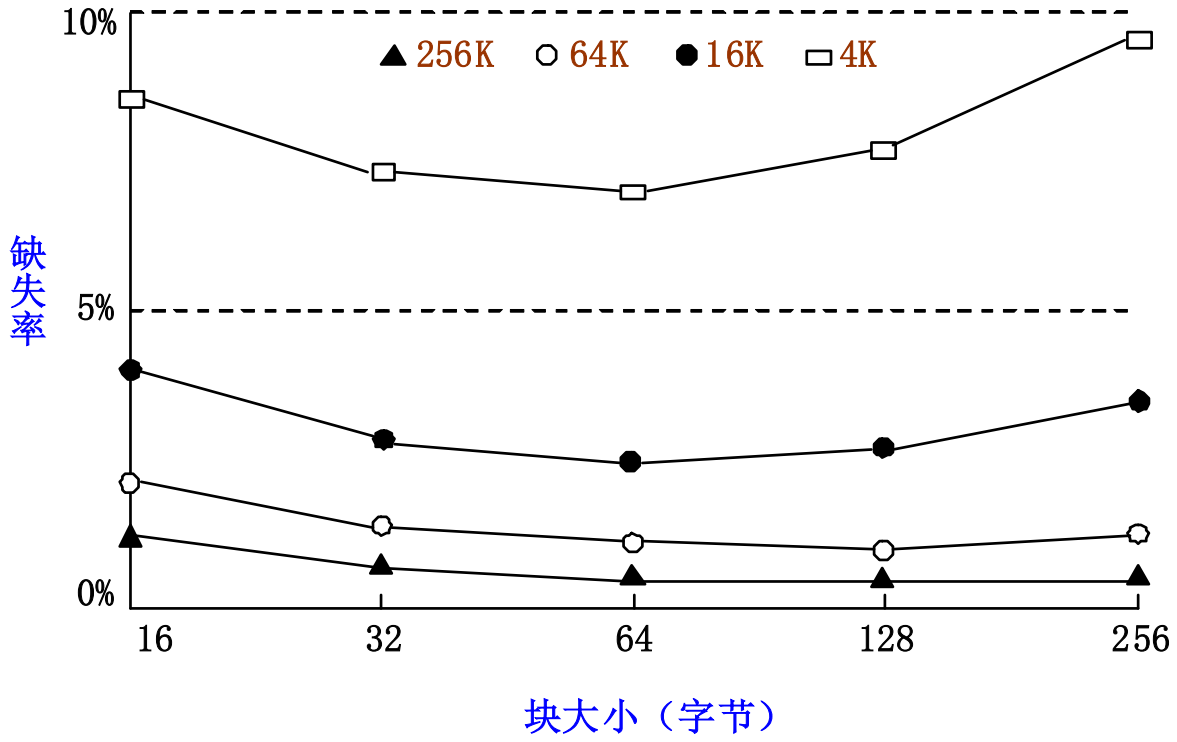


1.5 6种基本的缓存（Cache）优化



1.5 6种基本的缓存（Cache）优化

■ 方法1：增加Cache块大小以降低缺失率



缺失率随块大小变化的曲线

1.5 6种基本的缓存（Cache）优化

不同块大小情况下Cache的缺失率

块大小（字节）	Cache容量（字节）				
	1K	4K	16K	64K	256K
16	15.05%	8.57%	3.94%	2.04%	1.09%
32	13.34%	7.24%	2.87%	1.35%	0.70%
64	13.76%	7.00%	2.64%	1.06%	0.51%
128	16.64%	7.78%	2.77%	1.02%	0.49%
256	22.01%	9.51%	3.29%	1.15%	0.49%

1.5 6种基本的缓存（Cache）优化

- 对于给定的**Cache**容量，当块大小增加时，缺失率开始是下降，后来反而上升了。

  原因：

  - 一方面它减少了强制性缺失；
  - 另一方面，由于增加块大小会减少Cache中块的数目，所以有可能会增加冲突缺失。
- **Cache**容量越大，使缺失率达到最低的块大小就越大。
- 增加块大小会增加缺失代价

1.5 6种基本的缓存（Cache）优化

- 方法2：增加Cache的容量以降低缺失率
  - 增加**Cache**的容量以减少容量缺失
  - 缺点：
    - 增加成本和功耗
    - 可能增加命中时间
  - 这种方法在片外**Cache**中用得比较多

1.5 6种基本的缓存（Cache）优化

- 方法3：提高相联度以降低缺失率
  - 采用相联度超过8的方法实际意义不大
  - **2:1 Cache**经验规则
    容量为**N** 的直接映象 **Cache**与容量为**N/2**的两路组相联**Cache**具有大体相同的缺失率
  - 提高相联度会增加命中时间

1.5 6种基本的缓存（Cache）优化

例：假定提高相联度会按下列比例增加处理器时钟周期时间：

时钟周期2路 ＝1.36×时钟周期1路

时钟周期4路 ＝1.44×时钟周期1路

时钟周期8路 ＝1.52×时钟周期1路

假定命中时间为1个时钟，缺失代价为25个时钟周期，而且假设不必将缺失代价取整。使用表B－4中的缺失率，试问当Cache为多大时，以下不等式成立？

$$\text{平均访存时间}_{8\text{路}} < \text{平均访存时间}_{4\text{路}}$$

$$\text{平均访存时间}_{4\text{路}} < \text{平均访存时间}_{2\text{路}}$$

$$\text{平均访存时间}_{2\text{路}} < \text{平均访存时间}_{1\text{路}}$$

1.5 6种基本的缓存（Cache）优化

解：在各种相联度的情况下，平均访存时间分别为：

$$\text{平均访存时间}_{8路} = \text{命中时间}_{8路} + \text{缺失率}_{8路} \times \text{缺失代价}_{8路}$$
$$= 1.52 + \text{缺失率}_{8路} \times 25$$

$$\text{平均访存时间}_{4路} = 1.44 + \text{缺失率}_{4路} \times 25$$

$$\text{平均访存时间}_{2路} = 1.36 + \text{缺失率}_{2路} \times 25$$

$$\text{平均访存时间}_{1路} = 1.00 + \text{缺失率}_{1路} \times 25$$

在每种情况下的缺失代价相同，都是25个时钟周期。把相应的缺失率代入上式，即可得平均访存时间。

例如，4KB的直接映象Cache的平均访存时间为：

$$\text{平均访存时间}_{1路} = 1.00 + (0.098 \times 25) = 3.44$$

容量为512KB的8路组相联Cache的平均访存时间为：

$$\text{平均访存时间}_{8路} = 1.52 + (0.006 \times 25) = 1.66$$

1.5 6种基本的缓存（Cache）优化

Cache大小（KB）	相联度			
	1路	2路	4路	8路
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

1.5 6种基本的缓存（Cache）优化

- 方法4：采用多级Cache以降低缺失代价
  - 把**Cache**做得更快？还是更大？

    答案：二者兼顾，再增加一级**Cache**
- **第一级Cache(L1)小而快**
- **第二级Cache(L2)容量大**
  - 性能分析：

$$\text{平均访问时间} = \text{命中时间}_{L1} + \text{缺失率}_{L1} \times \text{缺失代价}_{L1}$$

$$= \text{命中时间}_{L1} + \text{缺失率}_{L1} \times (\text{命中时间}_{L2} + \text{缺失率}_{L2} \times \text{缺失代价}_{L2})$$

1.5 6种基本的缓存（Cache）优化

- 局部缺失率与全局缺失率

局部缺失率＝该级**Cache**的缺失次数/到达该级**Cache**的访问次数

全局缺失率＝该级**Cache**的缺失次数/**CPU**发出的访存的总次数

$全局缺失率_{L2}＝缺失率_{L1}×缺失率_{L2}$

评价第二级**Cache**时，一般使用全局缺失率这个指标。它指出了在**CPU**发出的访存中，究竟有多大比例是穿过各级**Cache**，最终到达存储器的。

每条指令的平均存储器停顿时间 = 每条指令的缺失数$_{L1}$ ×命中时间$_{L2}$+每条指令的缺失数$_{L2}$ ×缺失代价$_{L2}$

1.5 6种基本的缓存（Cache）优化

例 假设在1000次访存中，第一级Cache缺失40次， 第二级Cache缺失20次。试问：在这种情况下，该Cache系统的局部缺失率和全局缺失率各是多少？假定L2 cache到存储器的缺失代价为200个时钟周期， L2 cache的命中时间为10个时钟周期， L1 cache的命中时间为1个时钟周期，每条指令共有1.5次存储器引用。每条指令的存储器平均访问时间和平均停顿周期为多少？

1.5 6种基本的缓存（Cache）优化

解：

第一级Cache的缺失率（全局和局部）是 40/1000，即4%；

第二级Cache的局部缺失率是20/40，即50%；

第二级Cache的全局缺失率是20/1000，即2%。

存储器平均访问时间=命中时间$_{L1}$＋缺失率$_{L1}$×(命中时间$_{L2}$＋缺失率$_{L2}$×缺失代价$_{L2}$)

=1+4%×（10+50%×200）=5.4个时钟周期

每条指令的平均存储器停顿时间=每条指令的缺失数$_{L1}$×命中时间$_{L2}$+每条指令的缺失数$_{L2}$×缺失代价$_{L2}$

=（60/1000）×10+（30/1000）×200 =6.6个时钟周期

1.5 6种基本的缓存（Cache）优化

- 对于第二级**Cache**，有以下结论：
  - 在第二级**Cache**比第一级 **Cache**大得多的情况下，两级**Cache**的全局缺失率与第二级**Cache**相同的单级**Cache**的缺失率非常接近。
  - 局部缺失率不是衡量第二级**Cache**的一个好指标，因此，在评价第二级**Cache**时，应用全局缺失率这个指标。

1.5 6种基本的缓存（Cache）优化

- 第二级**Cache**不会影响**CPU**的时钟频率，因此其设计有更大的考虑空间。两个问题：
  - 能否降低**CPI**中的平均访存时间部分？
  - 成本是多少？
  - 容量
- 第二级**Cache**的容量一般比第一级的大许多。
- 第二级**Cache**可采用较高的相联度

1.5 6种基本的缓存（Cache）优化

例： 给出有关第二级Cache的以下数据：

⑴直接映射的命中时间$_{L2}$＝10个时钟周期

⑵两路组相联使命中时间增加0.1个时钟周期

⑶ 对于直接映象，局部缺失率$_{L2}$＝25%

⑷ 对于两路组相联，局部缺失率$_{L2}$＝20%

⑸缺失代价$_{L2}$＝200个时钟周期

试问第二级Cache的相联度对缺失代价的影响如何？

1.5 6种基本的缓存（Cache）优化

解：

对于一个直接映射的第二级Cache来说，第一级Cache的缺失代价为：

缺失代价$_{\text{直接映象，L1}}$

＝10＋25%×200＝60个时钟周期

对于两路组相联第二级Cache来说，命中时间增加了0.1个时钟周期，故第一级Cache的缺失代价为：

缺失代价$_{\text{两路组相联，L1}}$

＝10.1＋20%×200＝50.1个时钟周期

故对于第二级Cache来说，两路组相联优于直接映象。

1.5 6种基本的缓存（Cache）优化

- 多级包容性

需要考虑的另一个问题：

第一级**Cache**中的数据是否总是同时存在于第 二级**Cache**中。

- 块大小

为减少平均访存时间，可以让容量较小的第一 级**Cache**采用较小的块，而让容量较大的第二级**Cache**采用较大的块。

1.5 6种基本的缓存（Cache）优化

- 方法5：使读取缺失的优先级高于写入缺失，以降低缺失代价
  - **Cache**中的写缓冲器导致对存储器访问的复杂化：写缓冲器进行的写入操作是滞后进行的，所以该缓冲器也被称为后行写数缓冲器。

例： 考虑以下指令序列：

SW R3, 512（R0） ；**M[512] ← R3** （**Cache**索引为**0**）

LW R1, 1024（R0） ；**R1←M[1024]** （**Cache**索引为**0**）

LW R2, 512（R0） ；**R2←M[512]** （**Cache**索引为**0**）

写直达，直接映像，将地址**512**和**1024**映射到同一块，写缓冲器为四个字。**R2**中的值是否总等于**R3**中的值？

1.5 6种基本的缓存（Cache）优化

- 解决问题的方法(读失效的处理)
  - 推迟对读失效的处理
  - 检查写缓冲器中的内容
- 在写回法**Cache**中，如采用写缓冲器，也可以采用类似方法（检查写缓冲器中的内容）

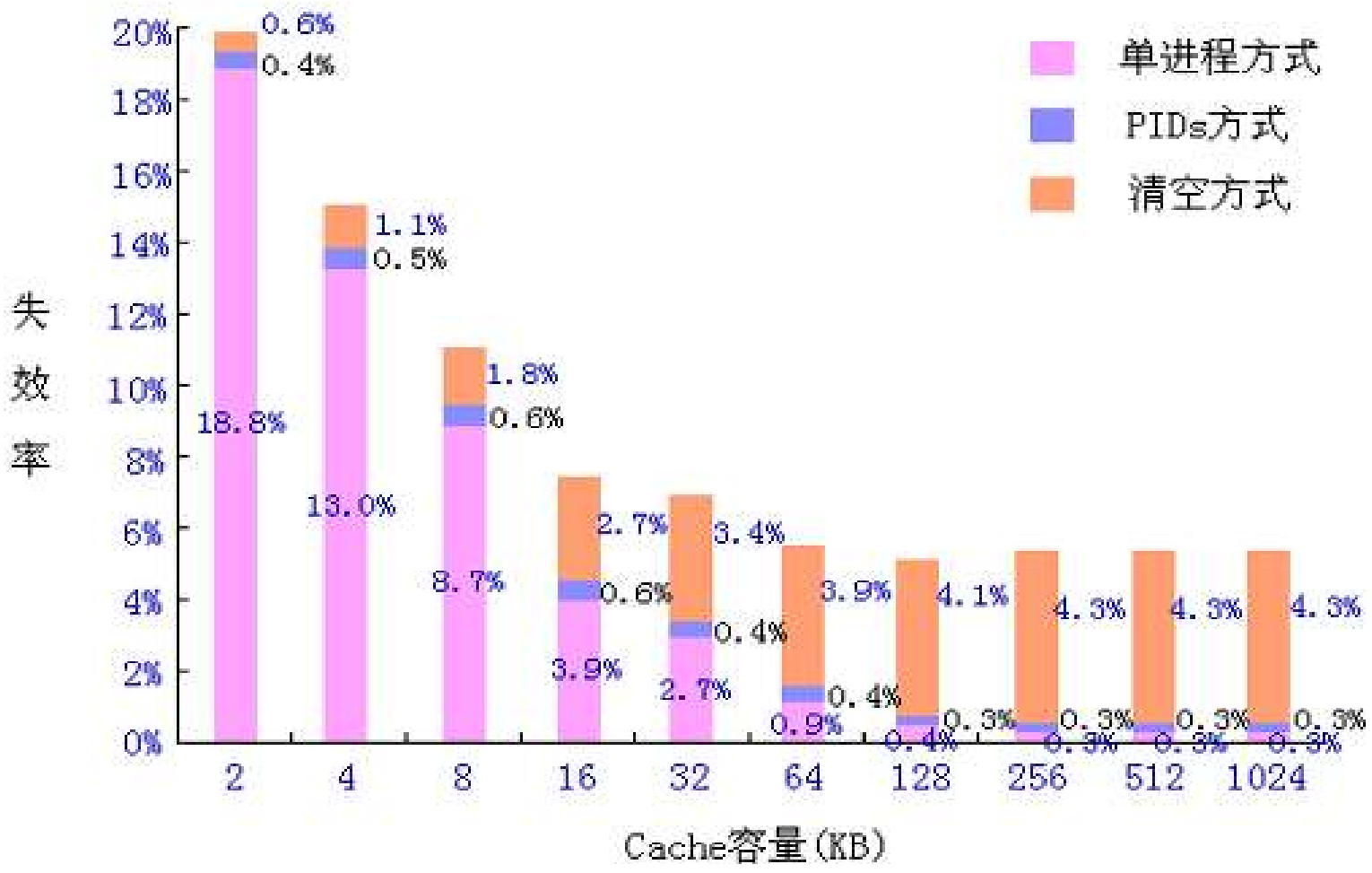
1.5 6种基本的缓存（Cache）优化

- 方法6：避免在索引cache期间进行地址转换，以缩短命中时间

  虚拟**Cache**：访问**Cache**的索引以及**Cache**中的标识都是虚拟地址

  - ➢ 并非都采用虚拟**Cache**
  - 原因一：要提供保护
  - 原因二：每次切换进程时，虚拟地址会指向不同的物理地址，需要对**Cache**进行刷新。
  - ✓ 解决方法：在**Cache**地址标识中增加**PID**字段(进程标识符)
  - ✓ 三种情况下失效率的比较：单进程，**PIDs**，清空（**purge**）

三种方式下，虚拟Cache的失效率



1.5 6种基本的缓存（Cache）优化

- 原因三：为同一的物理地址使用不同的虚拟地址。这些重复的地址称为同义地址或别名地址。

  解决方法：

✓ 别名消去：用硬件来保证每一个**Cache**块对应唯一物理地址

✓ 页着色：**Sun**公司的**UNIX**系统要求所有别名地址的最后**18**位相同，而直接映象**Cache**的索引来自于这最后**18**位(**Cache**容量不超过2^{18}字节)，这使得所有别名映象到同一**Cache**块位置。

1.5 6种基本的缓存（Cache）优化

- 使用页内位移(这部分虚拟地址和物理地址相同)进行Cache索引，同时进行虚实地址转换，标志匹配使用物理地址。

- 优点：兼得虚拟Cache和物理Cache的好处（Cache索引和虚实地址转换同时进行）
- 局限性：Cache容量受到限制

直接映象： Cache容量≤页大小

组相联映象：Cache容量≤页大小×相联度

Cache优化技术总结

优化技术	命中时间	缺失代价	缺失率	硬件复杂度	说 明
增加块大小		–	+	0	Pentium 4 的第二级Cache采用了128 B的块
增加Cache容量	–		+	1	广泛采用，特别是第二级Cache
提高相联度	–		+	1	广泛采用
多级Cache		+		2	硬件代价大；两级Cache的块大小不同时实现困难；广泛采用
使读缺失优先于写缺失		+		1	广泛采用
对Cache进行索引时不必进行地址变换	+			1	

2. 10种高级的缓存优化方法

- 10种高级优化方法可分为5类
  - 缩短命中时间
  - 增加缓存带宽
  - 降低缺失代价
  - 降低缺失率
  - 通过并行降低缺失代价或缺失率

2. 10种高级的缓存优化方法

- 方法1：容量小、结构简单的第一级Cache，以缩短命中时间、降低功耗
  - 硬件越简单，速度就越快
  - 小容量Cache可以减少命中时间和降低功耗
  - Cache结构简单，采用较低级别的相联度，如直接映射Cache ，也可以减少命中时间和降低功耗。

2. 10种高级的缓存优化方法

- 方法2：采用路预测（Way Prediction）以缩短命中时间

➢ 可以减少冲突缺失，又能保持直接映射Cache命中速度。

➢ 在一个Cache中的每个块中都添加块预测位。根据这些位选定要在下一次缓存访问中优先尝试哪些块。

- 如果预测正确，则缓存访问延迟就等于这一快速命中时间；
- 如果预测错误，则尝试其他块，改变路预测器，延迟会增加一个时钟周期。
- 模拟表明，两路组相联缓存，预测准确度超过90%；四路组相联缓存，预测准确度超过80%

2. 10种高级的缓存优化方法

- 方法3：实现缓存访问的流水化，以提高缓存带宽

  - 对第一级**Cache**的访问按流水方式组织，使第一级缓存的实际延迟可以分散到多个时钟周期，从而缩短时钟周期时间，提高带宽，但会增加**Cache**的命中时间。
  - 访问**Cache**需要多个时钟周期才可以完成，例如：
    - Pentium访问指令Cache需要一个时钟周期
    - Pentium Pro到Pentium III需要两个时钟周期
    - Pentium 4到Intel Core i7则需要4个时钟周期

2. 10种高级的缓存优化方法

方法4：采用无阻塞Cache，以提高缓存带宽

- 无阻塞**Cache**：Cache缺失期间仍允许**CPU**进行其它的命中访问。即允许“缺失下命中” 。

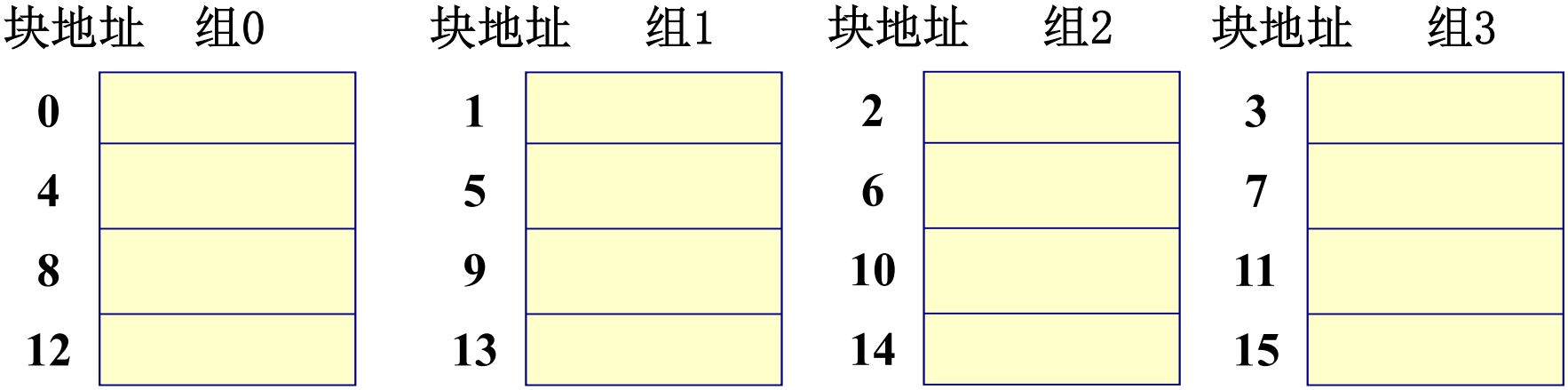
  在缺失期间，不会完全拒绝处理器的请求，从而降低了实际缺失代价。

- 进一步提高性能：“多重缺失下命中”，
  “缺失下缺失”（存储器系统必须能够为多次缺失提供服务）

- 对非阻塞**Cache**进行性能评估时，真正的难度在于一次**Cache**缺失不一定会使处理器停顿。

2. 10种高级的缓存优化方法

- 方法5：采用多组**Cache**，以提高缓存带宽
  - 将**Cache**划分为几个相互独立、支持同时访问的缓存组。
  - 一个简单有效的映射方法：将缓存块地址按顺序分散在这些缓存组中，这种方法称为顺序交错。



2. 10种高级的缓存优化方法

- 方法6：关键字优先和提前重启动以降低缺失代价
  - 请求字(关键字)： 从下一级存储器调入**Cache**的块中，只有一个字是处理器立即需要的。这个字称为请求字。
  - 尽早把请求字发送给CPU，两个策略：
    - 请求字优先：首先从存储器中请求缺失的字，在其到达Cache后立即发给处理器；
    - 提前重启动：调块时，从块的起始位置开始读起。一旦请求字到达，就立即发送给 CPU，让CPU继续执行。
  - 在采用大型Cache块时，这种技术有效果。

2. 10种高级的缓存优化方法

方法7：合并写缓冲区以降低缺失代价

- 依靠写缓冲来减少对下一级存储器写操作的时间。
- 如果写缓冲器为空，就把数据和相应地址写入该缓冲器。从CPU的角度来看，该写操作就算完成了。
- 如果写缓冲器中已经有了待写入的数据，就要把这次的写入地址与写缓冲器中已有的所有地址进行比较，看是否有匹配的项。如果有地址匹配而对应的位置又是空闲的，就把这次要写入的数据与该项合并。这就叫写缓冲合并。
- 如果写缓冲器满且又没有能进行写合并的项，就必须等待。
- 提高了写缓冲器的空间利用率，而且还能减少因写缓冲器满而要进行的等待时间。

2. 10种高级的缓存优化方法

写地址	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

（a）不采用写合并

写地址	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

（b）采用了写合并

2. 10种高级的缓存优化方法

方法8：采用编译器优化以降低缺失率

- 基本思想：在编译时，对程序中的指令和数据进行重新组织，以降低Cache失效率。
- 内外循环交换：

举例：

```
/* 修改前 */
for (j＝0 ;j<100 ;j＝j＋1)
    for (i＝0 ;i<5000 ;i＝i＋1)
        x[i][j]＝2*x[i][j];

/* 修改后 */
for (i＝0 ;i<5000 ;i＝i＋1)
    for (j＝0 ;j< 100 ;j＝j＋1)
        x[i][j]＝2*x[i][j];
```

2. 10种高级的缓存优化方法

- 分块：把对数组的整行或整列访问改为按块进行。

举例：

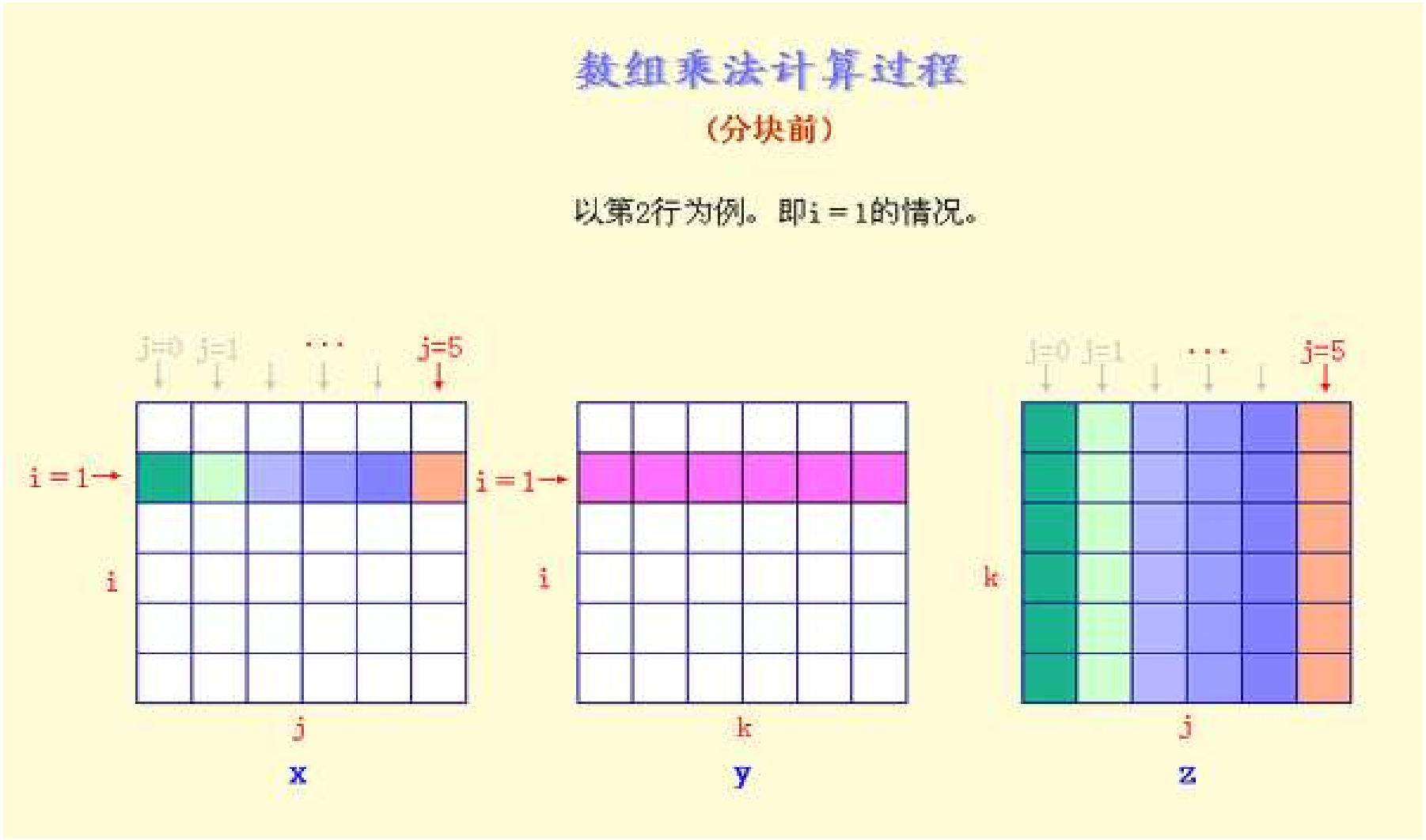
```
        /* 修改前 */
    for (i＝0; i < N; i＝i＋1)
      for (j＝0; j < N; j＝j＋1) {
        r＝0;
        for (k＝0; k < N; k＝k＋1) {
          r＝r＋y[i][k]*z[k][j];
          }
        x[i][j]＝r;
      }
```

2. 10种高级的缓存优化方法

- 分块：把对数组的整行或整列访问改为按块进行。

举例：

```
        /* 修改前 */
    for (i＝0; i < N; i＝i＋1)
      for (j＝0; j < N; j＝j＋1) {
        r＝0;
        for (k＝0; k < N; k＝k＋1) {
          r＝r＋y[i][k]*z[k][j];
          }
        x[i][j]＝r;
      }
```



最坏情况下失效次数：$2N^3+N^2$

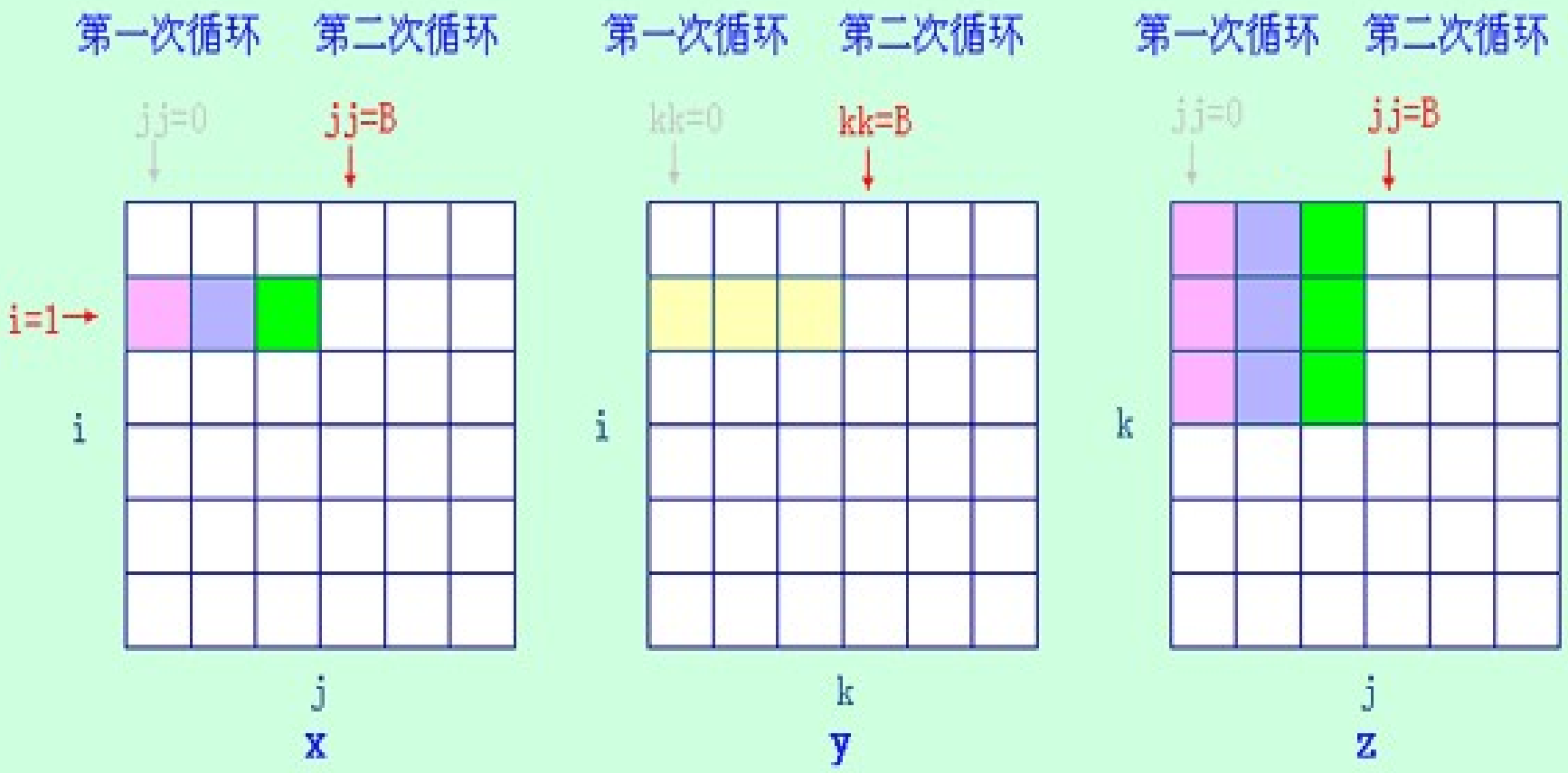
2. 10种高级的缓存优化方法

```
/* 修改后 */
for (jj＝0; jj < N; jj＝jj＋B)
for (kk＝0; kk < N; kk＝kk＋B)
for (i＝0; i < N; i＝i＋1)
    for (j＝jj; j < min(jj＋B－1,N); j＝j＋1)
    {
        r＝0;
        for (k＝kk; k < min(kk＋B－1,N); k＝k＋1)
            r＝r＋y[i][k]*z[k][j];
        x[i][j]＝x[i][j]＋r;
}
```

失效次数： $2N^3/B + N^2$

数组乘法计算过程

（分块后）



2. 10种高级的缓存优化方法

- 方法9：对指令和数据进行硬件预取，以降低缺失代价或缺失率
  - 依据：空间局部性
  - 指令和数据都可以预取
  - 预取内容既可放入Cache，也可放在外部缓冲器（速度快于主存储器)中
  - 指令预取经常在cache外部的硬件中完成。通常，处理器在一次缺失时提取两个块：被请求块和下一个相邻块。被请求块放入指令cache中，预取块放入指令流缓冲区中。

2. 10种高级的缓存优化方法

- Palacharla和Kessler的研究结果

  一个处理器具有两个**64KB**四路组相联**Cache**（分别用于缓存指令和数据），**8**个流缓冲器能捕获其所有缺失的**50%～70%**

- 预取应建立在存储器的空闲频带

2. 10种高级的缓存优化方法

- 方法10：用编译器控制预取，以降低缺失代价或缺失率

  - 由编译器加入预取指令，在数据被用到之前发出预取请求。
  - 两种预取类型
    - 寄存器预取：把数据载入到寄存器中
    - Cache预取： 只将数据载入到Cache中
  - 在预取数据的同时，处理器应能继续执行，只有这样，预取才有意义。数据 Cache通常是非阻塞性的。
  - 循环是预取优化的重要对象；
  - 发出预取指令会导致指令开销，编译器要确保这些开销不会超过所得到的好处。

2. 10种高级的缓存优化方法

例：对于下面的程序，判断哪些访问可能导致数据Cache缺失。然后，加入预取指令以减少缺失。最后，计算所执行的预取指令的条数以及通过预取避免的缺失次数。假定：

(1) 一个容量为**8KB**、块大小为**16**字节的直接映象**Cache**，它采用写回法并且按写分配。

(2) a、**b**分别为**3×100(3**行**100**列**)**和**101×3** 的双精度浮点数组，每个元素都是**8**个字节。当程序开始执行时，这些数据都不在**Cache**内。

```
for (i＝0 ; i＜3 ; i＝i＋1 )
    for (j＝0 ; j＜100 ; j＝j＋1 )
        a[i][j]＝b[j][0]×b[j＋1][0];
```

2. 10种高级的缓存优化方法

计算过程

$$
\begin{array}{ll}
i=0 & \left[\begin{array}{lll} b_{00}*b_{10} & b_{10}*b_{20} \cdots & b_{99,0}*b_{100,0} \end{array}\right. \\
i=1 & \left[\begin{array}{lll} b_{00}*b_{10} & b_{10}*b_{20} \cdots & b_{99,0}*b_{100,0} \end{array}\right. \\
i=2 & \left[\begin{array}{lll} b_{00}*b_{10} & b_{10}*b_{20} \cdots & b_{99,0}*b_{100,0} \end{array}\right.
\end{array}
$$

a 数组

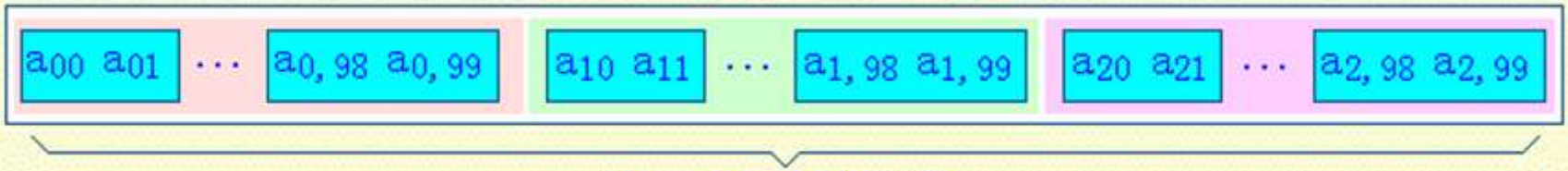
$$
\left[\begin{array}{lll}
b_{00} & b_{01} & b_{02} \\
b_{10} & b_{11} & b_{12} \\
b_{20} & b_{21} & b_{22} \\
\vdots & \vdots & \vdots \\
b_{99,0} & b_{99,1} & b_{99,2} \\
b_{100,0} & b_{100,1} & b_{100,2}
\end{array}\right.
$$

b 数组

2. 10种高级的缓存优化方法

对数组a的访问受益于空间局部性

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,98} & a_{0,99} \\ a_{10} & a_{11} & \cdots & a_{1,98} & a_{1,99} \\ a_{20} & a_{21} & \cdots & a_{2,98} & a_{2,99} \end{pmatrix}$$



共3×100＝300个元素

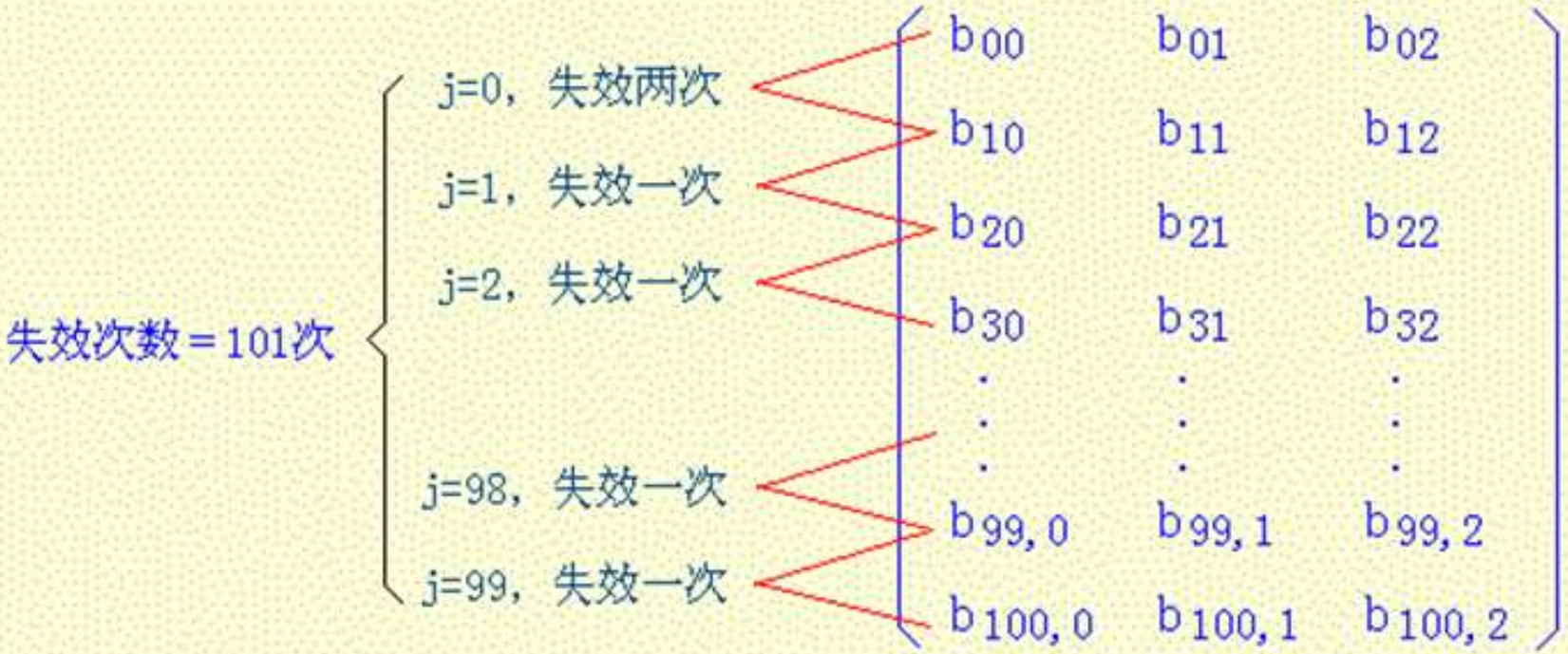
故失效次数＝300/2＝150

2. 10种高级的缓存优化方法

2. 数组b两次受益于时间局部性

(1) 对于i的每次循环，都访问同样的元素

(2) 对于j的每次循环，都使用一次上一次循环用过的b元素



2. 10种高级的缓存优化方法

```
for (j＝0， j＜100； j＝j＋1) {
    prefetch (b[j＋7][0]);
     /* 预取7次循环后所需的b(j ,0 ) */
    prefetch (a[0][j＋7]);
     /* 预取7次循环后所需的a(0,j ) */
    a[0][j]＝b[j ][0] * b [j＋1][0] }
for (i＝1; i＜3; i＝i＋1)
     for (j＝0; j＜100; j＝j＋1) {
          prefetch(a[i][j＋7]);
          /* 预取7次循环后所需的a(i , j ) */
          a[i][j]＝b[j][0] * b[j＋1][0];}
```

总的缺失次数＝(7+4)+4+4=19次

2. 10种高级的缓存优化方法

例：在以下条件下，计算上例中所节约的时间：

(1) 忽略指令**Cache**缺失，并假设数据**Cache**无冲突缺失和容量缺失。

(2) 假设预取可以被重叠或与**Cache**缺失重叠执行，从而能以最大的存储带宽传送数据。

(3) 不考虑**Cache**缺失时，修改前的循环每**7**个时钟周期循环一次。修改后的程序中，第一个预取循环每**9**个时钟周期循环一次，而第二个预取循环每**8**个时钟周期循环一次(包括外层**for**循环的开销)。

(4) 一次缺失需**100**个时钟周期。

2. 10种高级的缓存优化方法

解：

修改前：

循环时间 ＝300×7 ＝2100

总缺失开销＝251×100＝25100

总运行时间＝2100＋25100＝27200

修改后：

循环时间＝100×9＋200×8＝2500

总缺失时间＝19×100＝1900

总运行时间＝2500＋1900＝4400

加速比＝27200/4400＝6.2

10种Cache高级优化技术总结

优化技术	命中时间	带宽	缺失代价	缺失率	功耗	硬件复杂度
小而简单的cache	+			–	+	0
路预测Cache	+				+	1
流水化cache访问	–	+				1
非阻塞Cache		+	+			3
分组Cache		+			+	1
关键字优先和提前重启动			+			2
合并写缓冲区			+			1
以编译器技术减少缓存缺失				+		0
指令和数据的硬件预取			+	+	–	2/3
编译器控制的预取			+	+		3