计算机组成原理
(2014级)

计算机组成原理课程组

（刘旭东、肖利民、牛建伟、栾钟治）

Tel： 82316285

Mail：liuxd@buaa.edu.cn

liuxd@act.buaa.edu.cn

第七讲：高速缓冲存储器

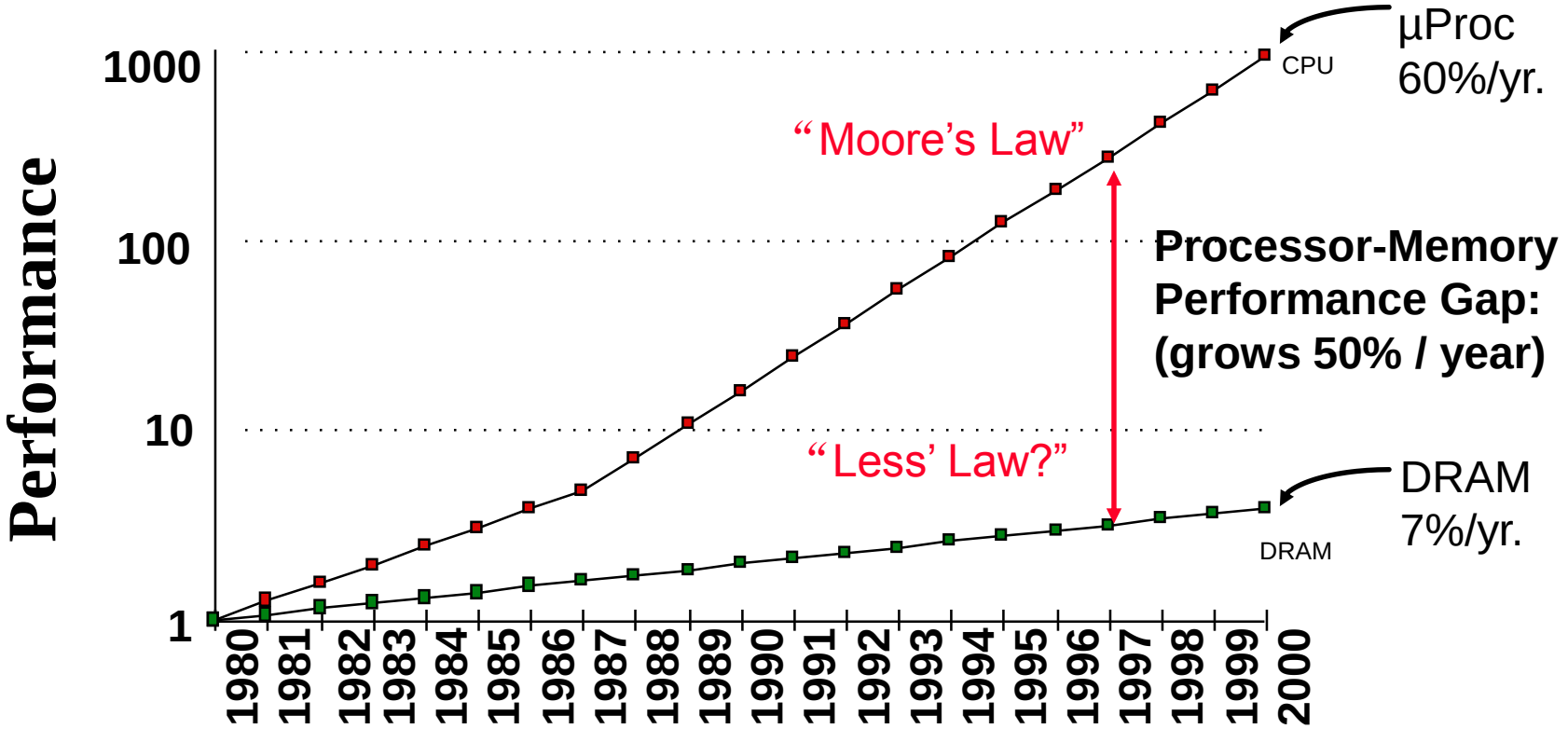
一. Cache的原理
   1. 程序访问的局部性原理
   2. Cache的结构与工作原理

二. Cache的映射机制
   1. 全相联映射
   2. 组相联映射
   3. 直接映射

三. Cache的替换策略

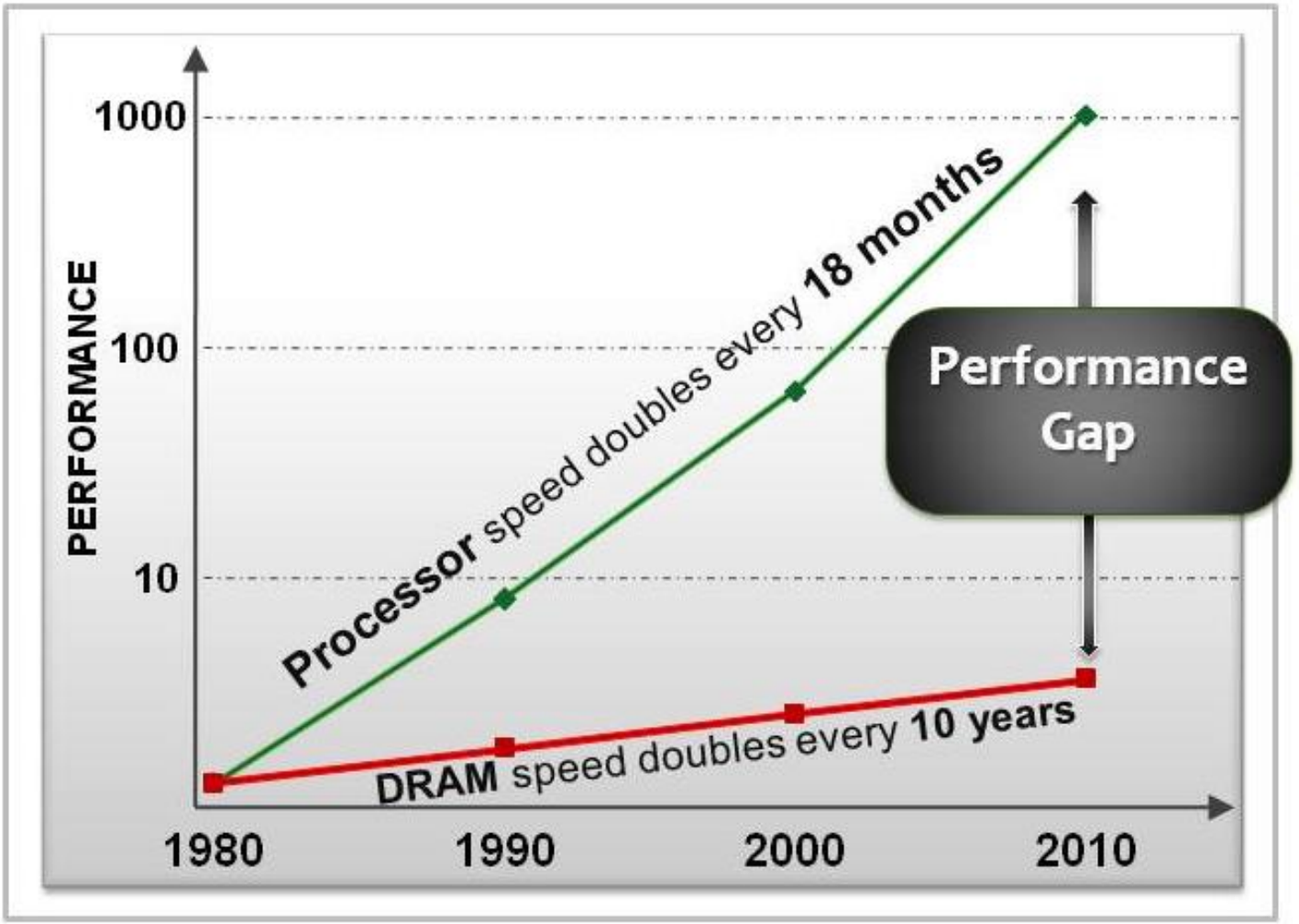
四. Cache性能分析

处理器—DRAM存储器的性能差距



Hennessy, J.L.; Patterson, D.A. *Computer Organization and Design*, 2nd ed. San Francisco: Morgan Kaufmann Publishers, 1997.

处理器—DRAM存储器的性能差距



Source: acm.org & MoSys

1.1 存储访问的局部性原理

❖程序示例

```
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a

```
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;
}
```

程序b

存储空间（M=3，N=4）

地址	内容
a00 地址	a00
+4	a01
+8	a02
+12	a03
+16	a10
+20	a11
+24	a12
+28	a13
+32	a20
+36	a21
+40	a22
+44	a23

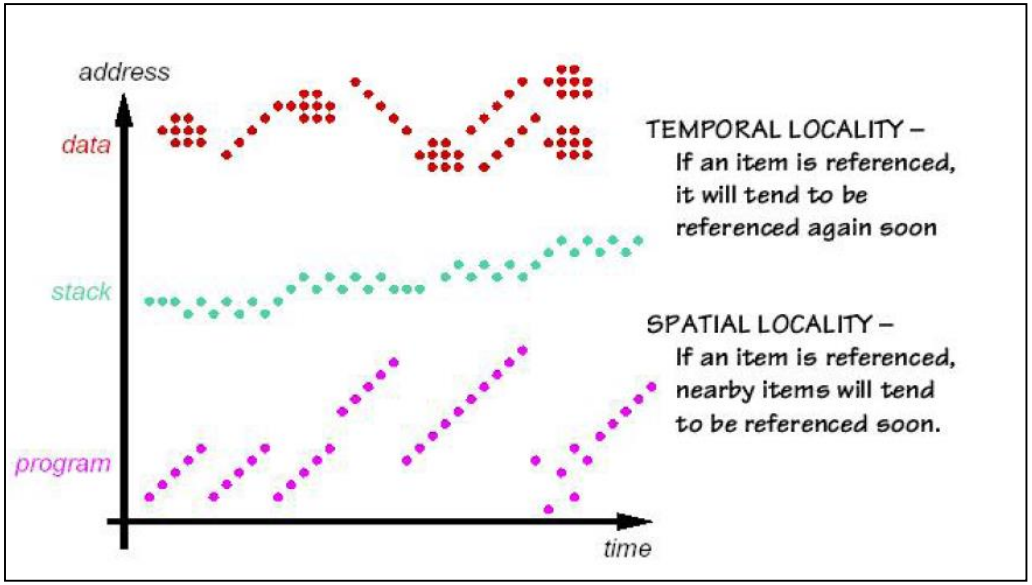
❖ 访问内存特点分析

- 程序a：sum被连读多次访问，数组a的访问具有空间连续性；
- 程序b ： sum被连读多次访问，数组a的访问不具备空间连续性；

1.1 存储访问的局部性原理

- 局部性原理（**principle of locality**）：大量典型程序的运行情况分析结果表明，无论是存取指令或存取数据所访问的存储单元都趋于聚集在一个较小的连续存储区域中。
  - **时间局部性(temporal locality)：** 刚被访问过的存储单元可能不久又将被访问；
  - **空间局部性(spatial locality)：** 岗被访问过的存储单元的临近单位可能不久被访问。

- 局部性的原因
  - 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。



1.1 存储访问的局部性原理

以下程序A和B中，哪一个对数组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:

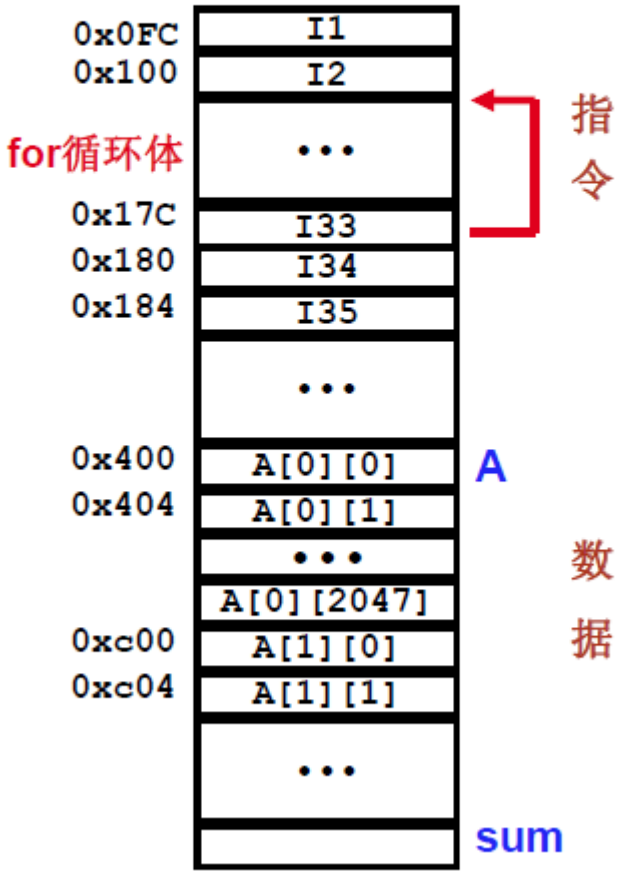
```
int sumarrayrows(int A[M][N])
{
  int i, j, sum=0;
     for  (i=0; i<M, i++)
           for (j=0; j<N, j++)
             sum+=A[i][j];
     return sum;
}
```

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;
}
```

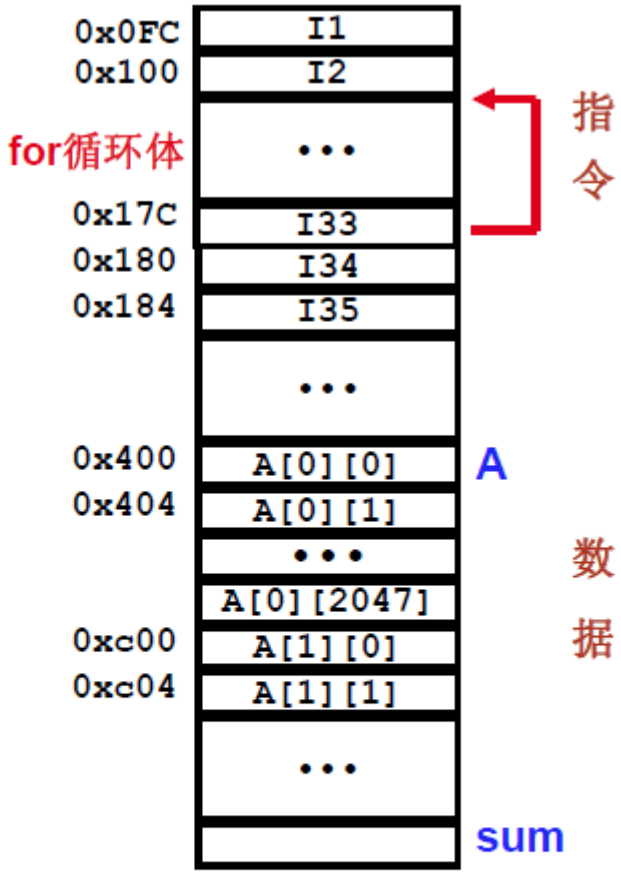
假定数组在存储器中按行优先顺序存放

M=N=2048时主存的布局:



1.1 存储访问的局部性原理

```
程序段A:
        ……
    int i, j, sum=0;
    for  (i=0; i<2048, i++)
          for (j=0; j<2048, j++)
            sum+=A[i][j];
    return sum;
        ……
```



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]都只被访问一次，所以时间局部性差！

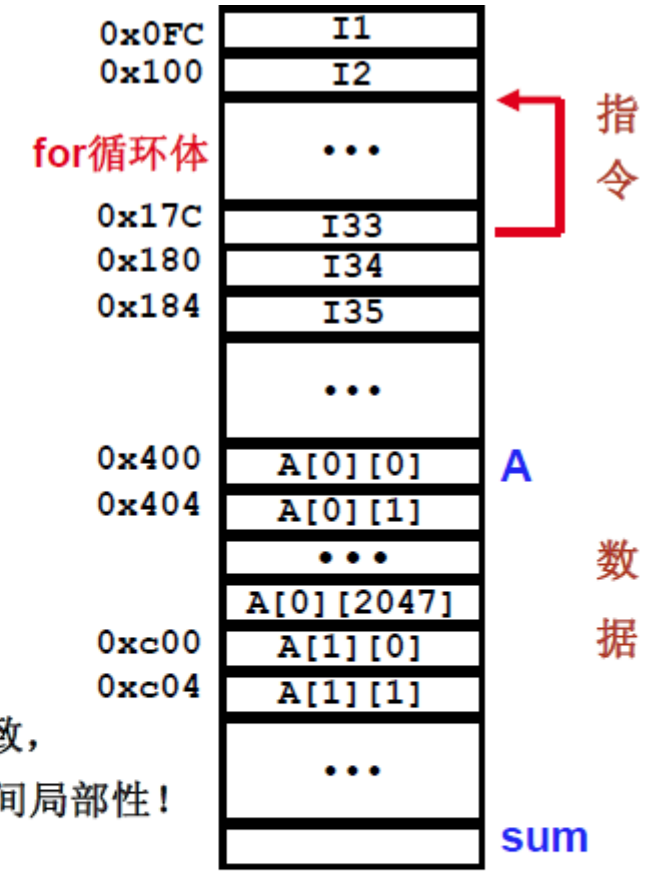
（2）变量sum：单个变量不考虑空间局部性；
每次循环都要访问sum，所以其时间局部性较好！

（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！

1.1 存储访问的局部性原理

```
程序段B:
        ……
    int i, j, sum=0;
     for  (j=0; j<2048, j++)
          for (i=0; i<2048, i++)
               sum+=A[i][j];
      return sum;
}
```



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!!

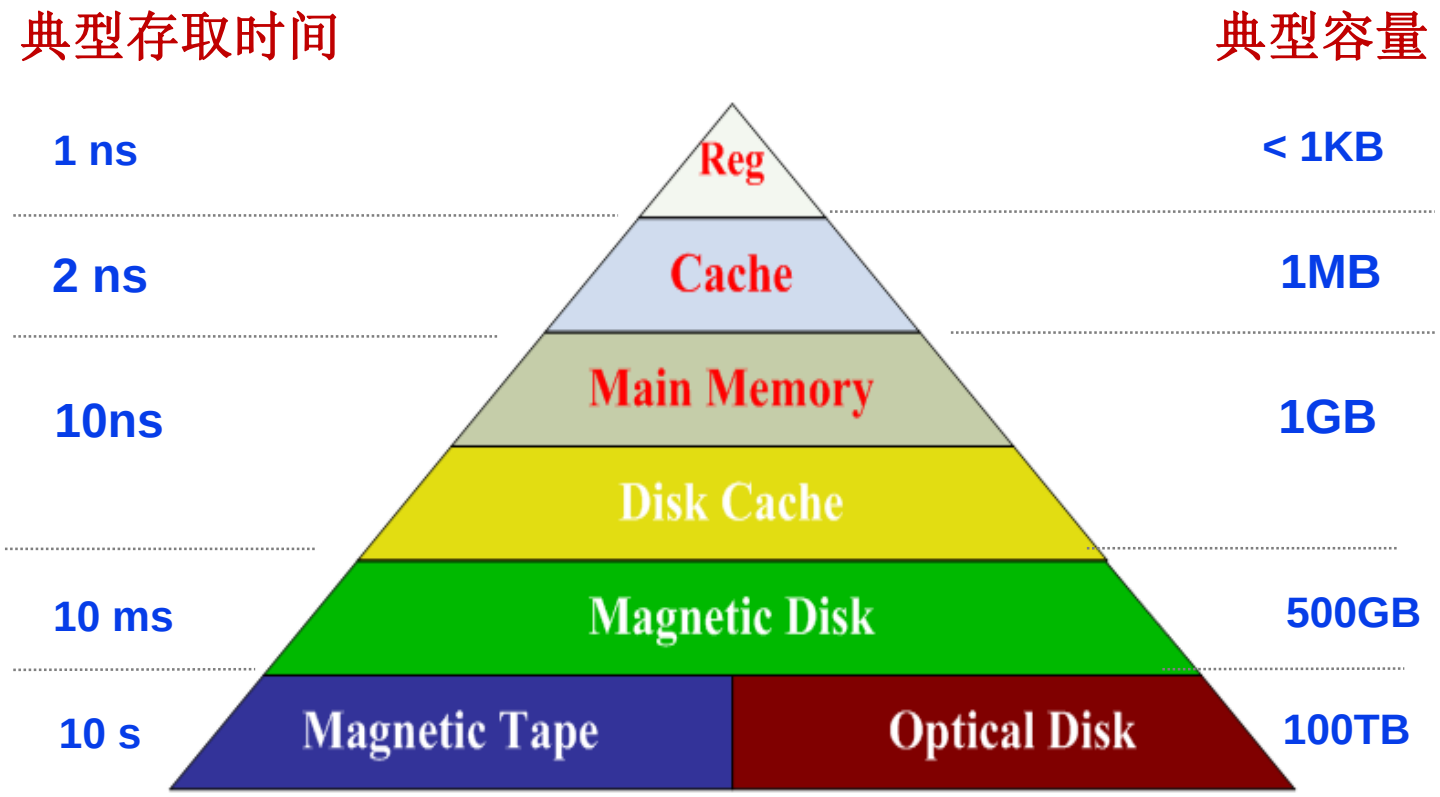
不同类型存储器的性能价格差异

存储器技术	典型存取时间	价格（$/GB，2004）
SRAM	0.5 ~ 5ns	$4,000~$10,000
DRAM	50 ~ 70ns	$100~$200
磁盘	$5*10^6$ ~ $20*10^6$ns	$0.5~$2

- **高速缓冲存储器产生的前提**
  - 单级存储系统中，主存的存储速度与CPU的速度不匹配，造成CPU资源的浪费；
  - 程序运行时访问内存存在明显的局部性特征；
  - 存在比主存普遍采用的DRAM速度更快的存储单元电路；

在CPU和主存间设置一容量较小的高速缓存，其中总是存放最活跃（被频繁访问）的程序块和数据，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。这个高速缓存就是Cache！

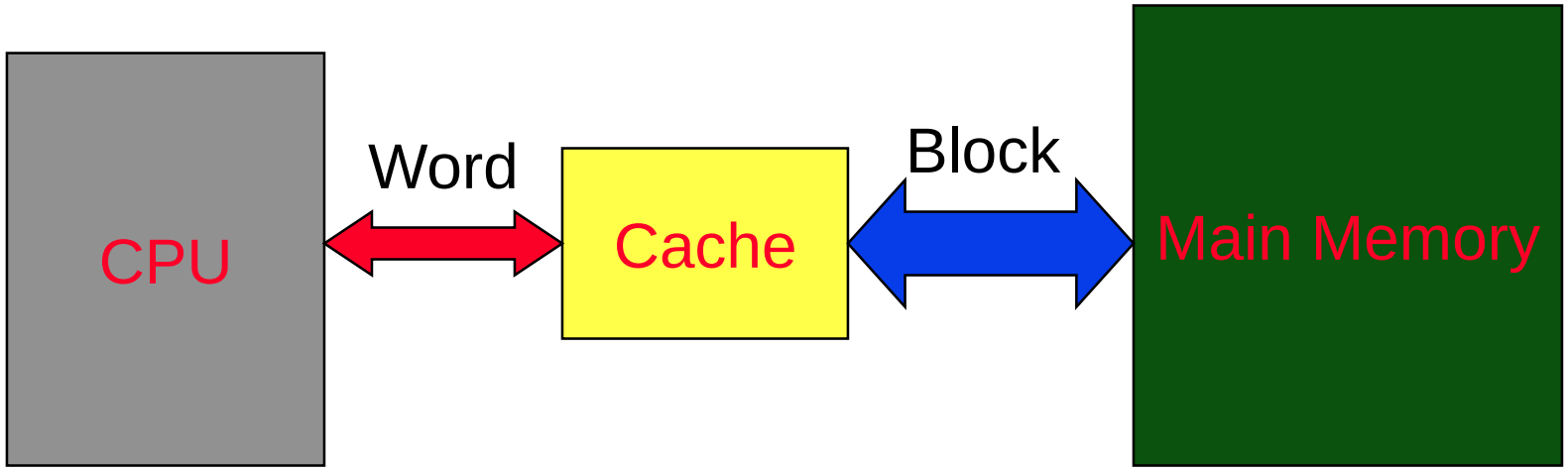
存储系统的层次结构



- 速度越快，成本越高，容量越小
- 工作过程：
  1）**CPU**运行时，需要的操作数首先来自寄存器
  2）需从（向）存储器中取**(**存**)** 数据时，先访问**cache**
  3）如操作数不在**cache**，则访问**RAM**
  4）如操作数不在**RAM**，则访问硬盘，操作数从硬盘→**RAM** →**cache**

1.2 高速缓冲存储器(Cache)的原理

- **Cache：CPU和主存间的一容量较小的高速缓存，其中总是存放最活跃（被频繁访问）的程序块和数据，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与主存之间按照数据块（Block）为单位进行数据交换。**



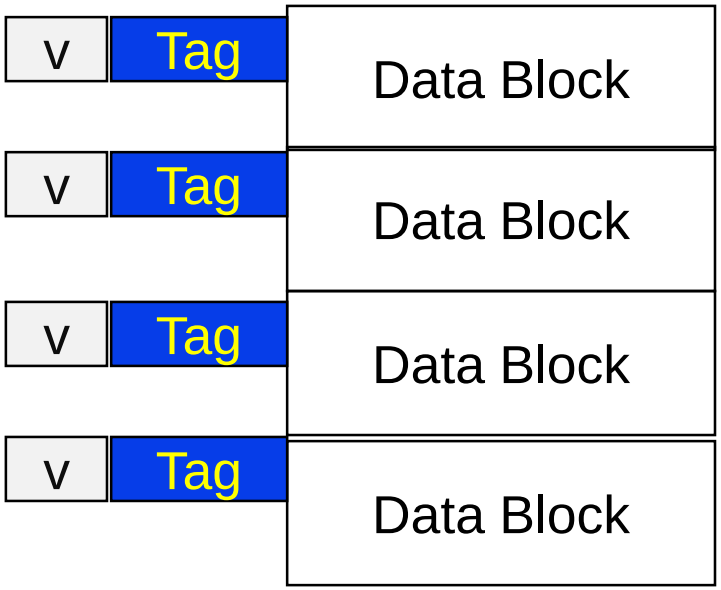
1.2 高速缓冲存储器(Cache)的原理

❖Cache要解决的问题

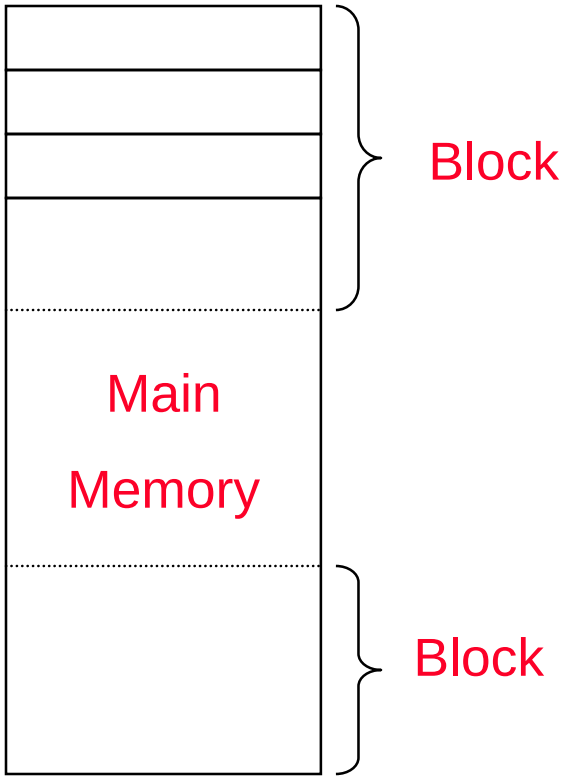
- ➢提供快速访问的能力；
- ➢与主存交换数据的能力；
- ➢由于CPU总是以主存地址访问存储器，所以Cache应具备判断CPU当前要访问的内容是否在Cache中的能力，并具有根据主存地址在Cache中访问相应单元的能力；
- ➢具备在Cache容量不够的前提下替换Cache中的内容的决策能力。

1.2 高速缓冲存储器(Cache)的原理

- **Cache的基本结构**
  - 存储机构：保存数据，存取数据，一般采用SRAM构成。以Block（若干字）为单位；
  - 地址机构：地址比较机制，地址转换机制，地址标标记（Tag），一个Block具有一个Tag；
  - 替换机构：记录Block的使用情况，替换策略，有效位（v）记录对应数据块中的数据是否有效。



Cache 的基本结构



1.2 高速缓冲存储器(Cache)的原理

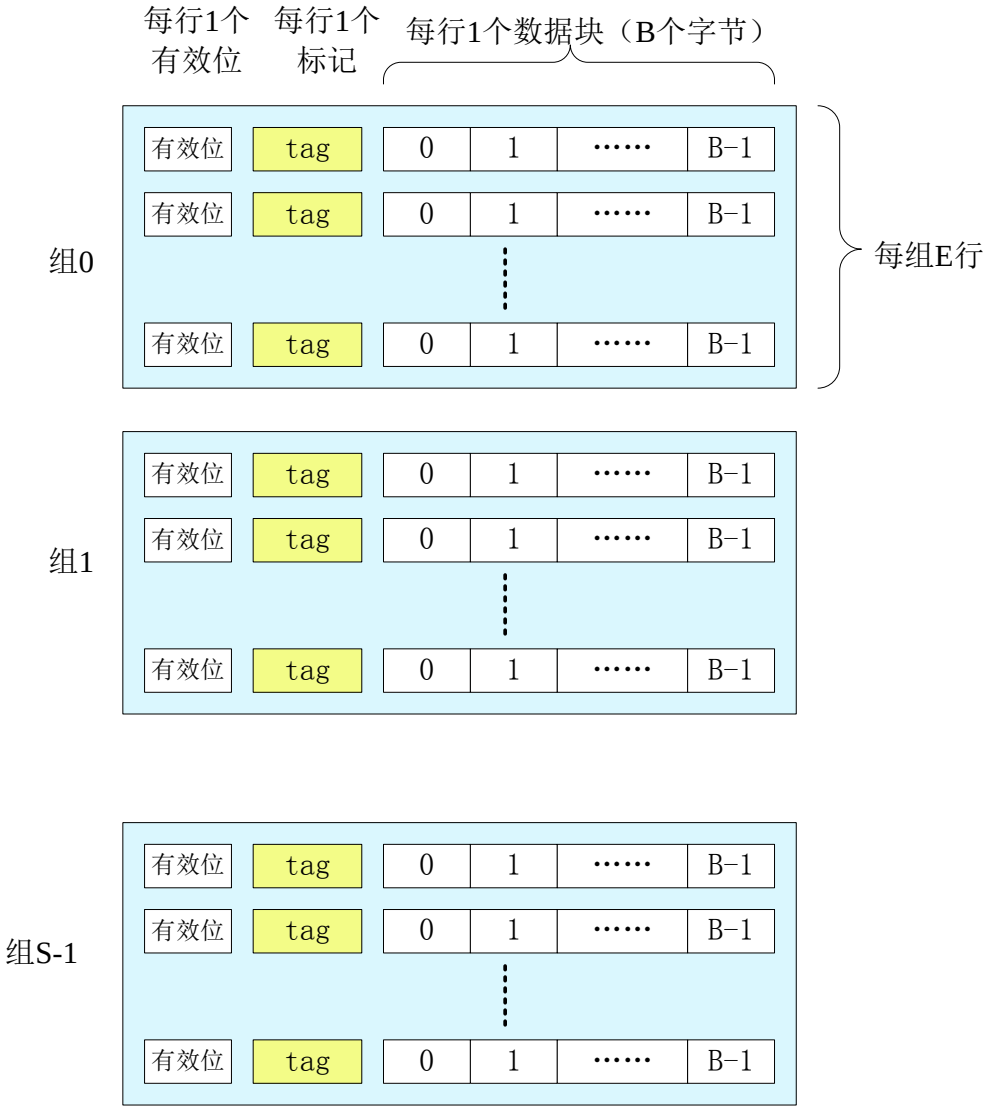
❖ Cache的有关术语

- 数据块（block）：Cache与主存的基本划分单位，也是主存与Cache一次交换数据的最小单位，由多个字节（字）组成，取决与主存一次读写操作所能完成的数据字节数。也表明主存于Cache之间局部总线的宽度。
- 标记（tag）：Cache每一数据块有一个标记字段，用来保存该数据块对应的主存数据块的地址信息。
- 有效位（valid bit）：Cache中每一Block有一个有效位，用于指示相应数据块中是否包含有效数据。
- 行（line)：Cache中 一个block及其 tag、valid bit构成1行。
- 组（set）：若干块(Block)构成一个组，地址比较一般能在组内各块间同时进行。
- 路（way）：Cache相关联的等级，每一路具有独立的地址比较机构，各路地址比较能同时进行（一般与组结合），路数即指一组内的块数。
- 命中率（hit rate）：目标数据在Cache中的存储访问的比例。
- 缺失率（miss rate）：目标数据不在Cache中的存储访问的比例。

1.2 高速缓冲存储器(Cache)的原理

❖Cache结构示意

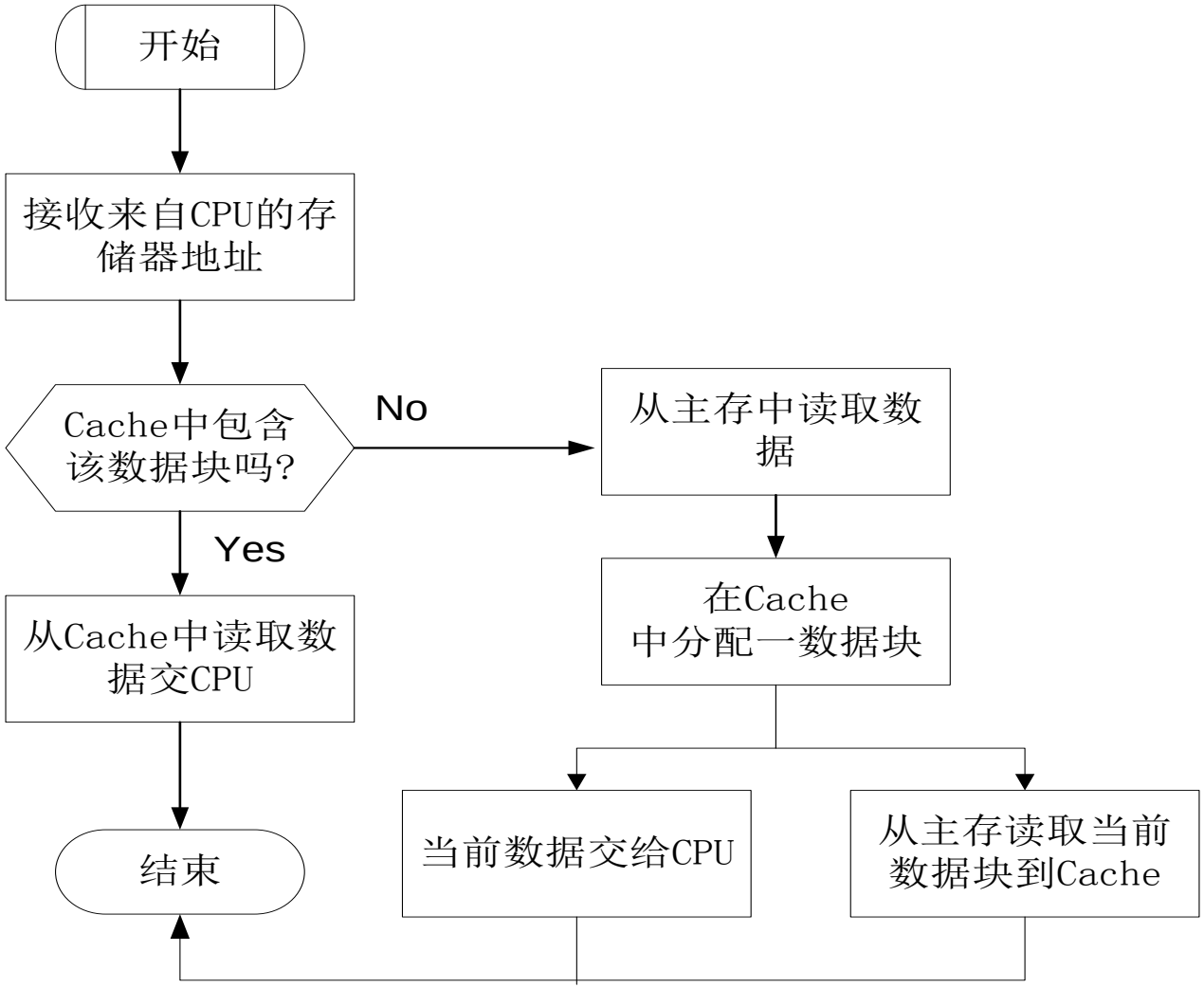
- ➢分S组
- ➢每组E行
- ➢每数据块包含B个字节



Cache容量（字节数）：B ×E ×S

1.2 高速缓冲存储器(Cache)的原理

❖Cache的读操作过程



第七讲：高速缓冲存储器

一．Cache的原理

1. 程序访问的局部性原理
2. Cache的结构与工作原理

二．Cache的映射机制

1. 全相联映射
2. 组相联映射
3. 直接映射

三．Cache的替换策略

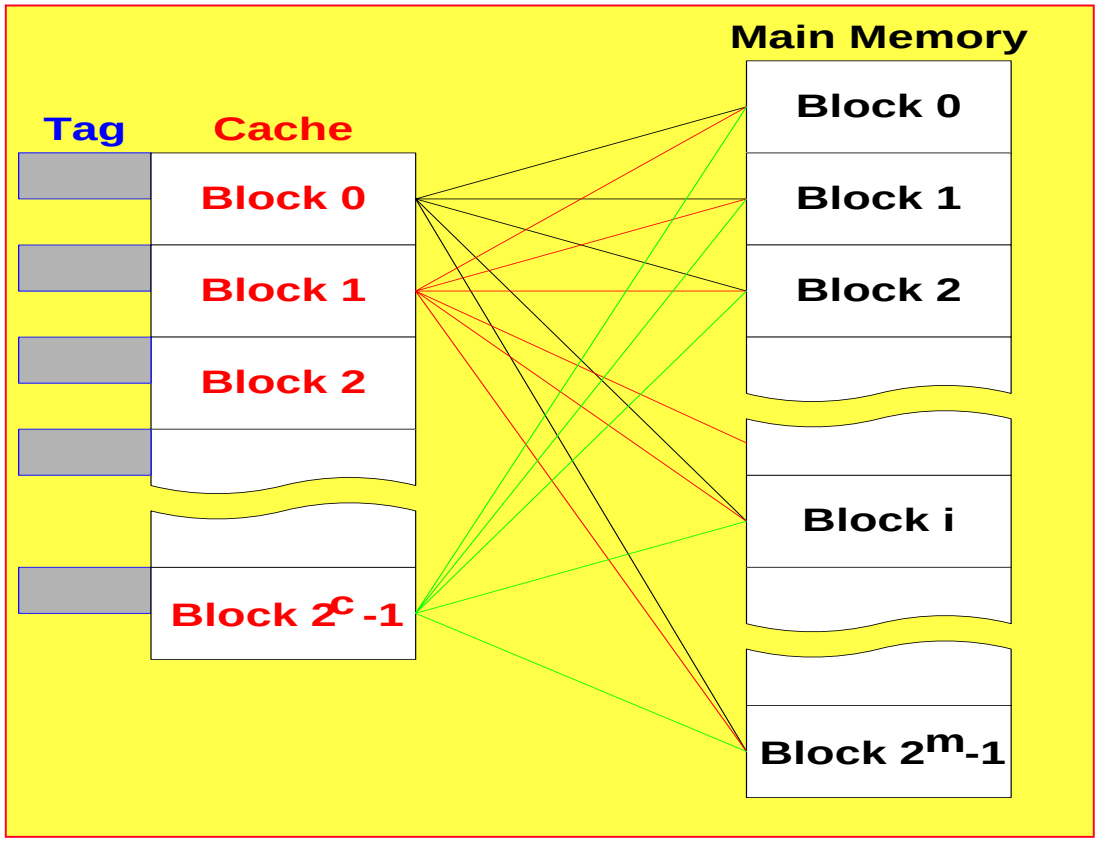
四．Cache性能分析

Cache与主存之间的映射

- 什么是Cache的映射
  - 把访问的局部主存区域取到Cache中时，该放到Cache的何处？
  - Cache的块比主存块少，多个主存块映射到一个Cache块中
- 如何进行映射
  - 把主存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）或项（Entry）或块（Block）
  - 将主存块和Cache块按照以下三种方式进行映射
    - 全相联（Full Associate）：每个主存块映射到Cache的任意块中
    - 组相联（Set Associate）：每个主存块映射到Cache的固定组中的任意块中
    - 直接（Direct）：每个主存块映射到Cache的固定块中

2.1 Cache与主存之间的映射—全相联

- 全相联（Full Associative）
  - 主存分为若干Block，Cache按同样大小分成若干Block，Cache中的Block数目显然比主存的Block数少得多。
  - 主存中的某一Block可以映射到Cache中的任意一Blcok。



2.1 Cache与主存之间的映射—全相联

❖全相联映射的地址

➢主存的地址格式：

Block Number (tag)	Offset

➢**Cache的Tag内容：主存中与该Cache数据块对应的数据块的块地址。**

❖举例

- **主存容量16M Bytes，Cache容量64K Bytes，块大小16字节。**
- **Cache分多少块？**
- **Tag需要多少位？**

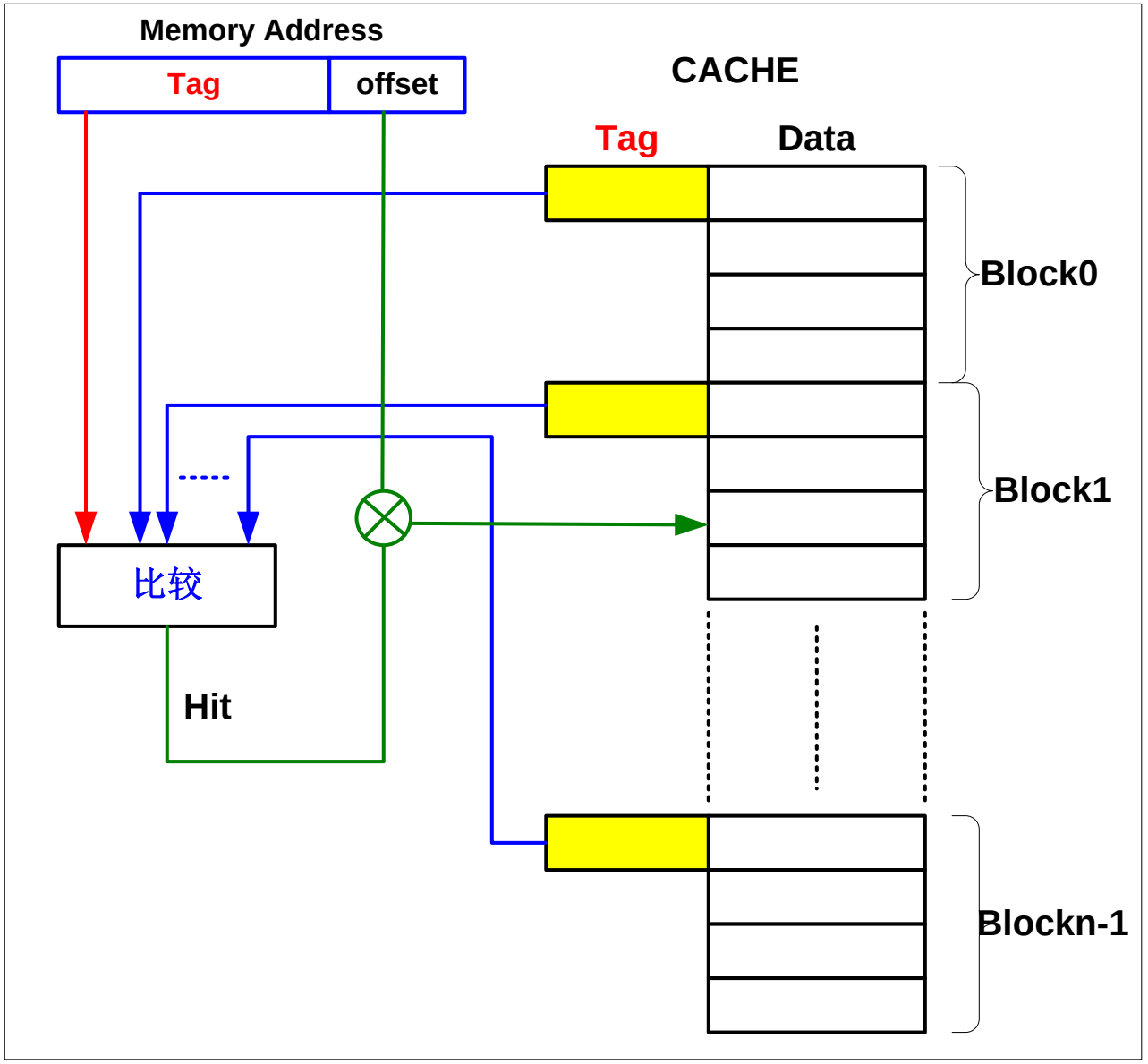
解：

- **主存块数：16M÷16 = 2^{20} Blocks**
- **主存地址：24位，其中高20位为块地址，低 4 位为块内地址**
- **Cache块数：64K÷16 ＝ 2^{12} Blocks**
- **Cache的Tag应该为 20 位。**

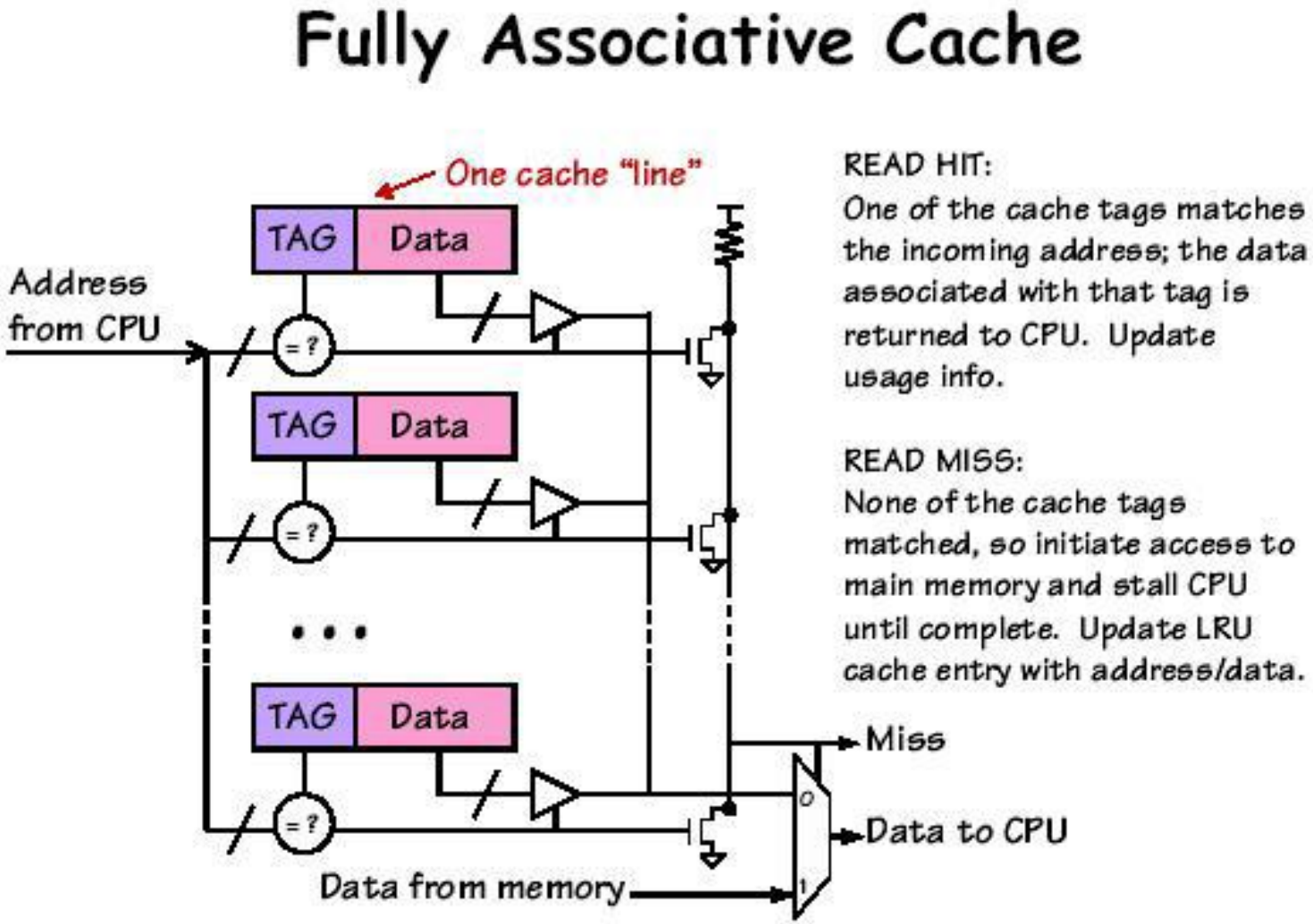
2.1 Cache与主存之间的映射—全相联

全相联Cache组织

- 同时与所有Tag进行比较，需N个比较器；
- 数据访问与比较并行执行。



2.1 Cache与主存之间的映射—全相联



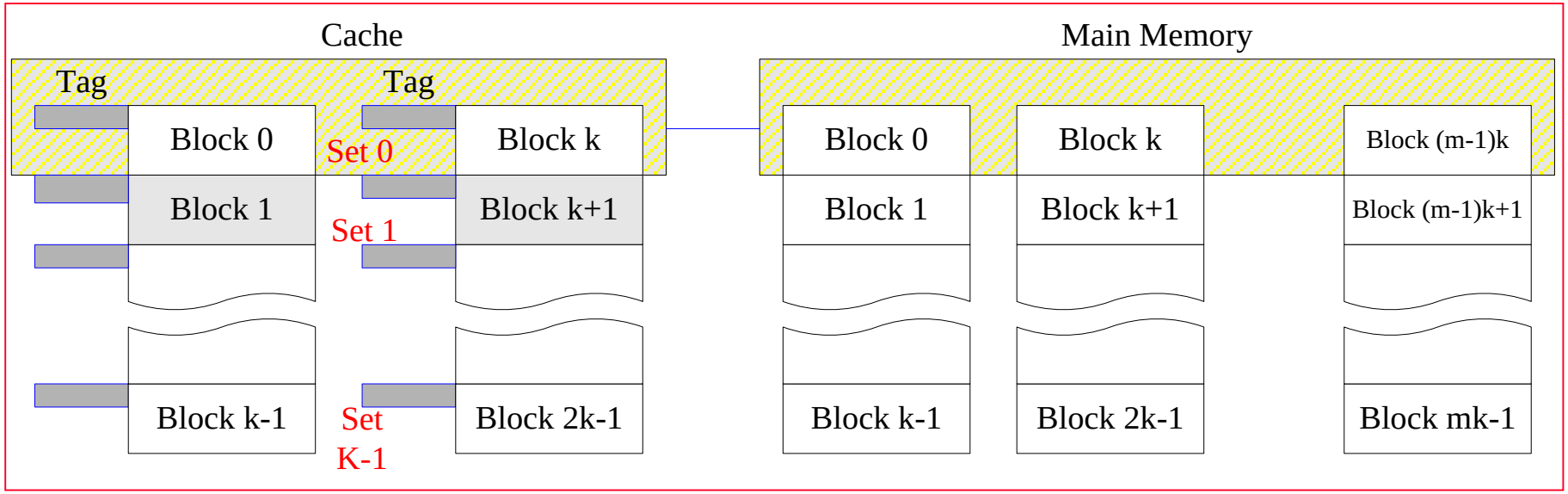
2.2 Cache与主存之间的映射—组相联

❖组相联（Set Associative）

- 映射关系：Cache 分成 K 组，每组分成 L 块；主存的块 J 以下列原则映射到 Cache 的组 I 中的任何一块。

  I = J mod K

- 实际上主存与Cache都分成 K 组，主存每一组内的块数与Cache一组内的块数不一致, 主存组M内的某一块只能映射到Cache组M内,但可以是组M内的任意一块。



2.2 Cache与主存之间的映射—组相联

❖组相联映射

- ➢主存的地址格式：

Tag	Set #	Offset

- ➢Tag的内容：主存中与该Cache数据块对应的数据块的组内块地址。

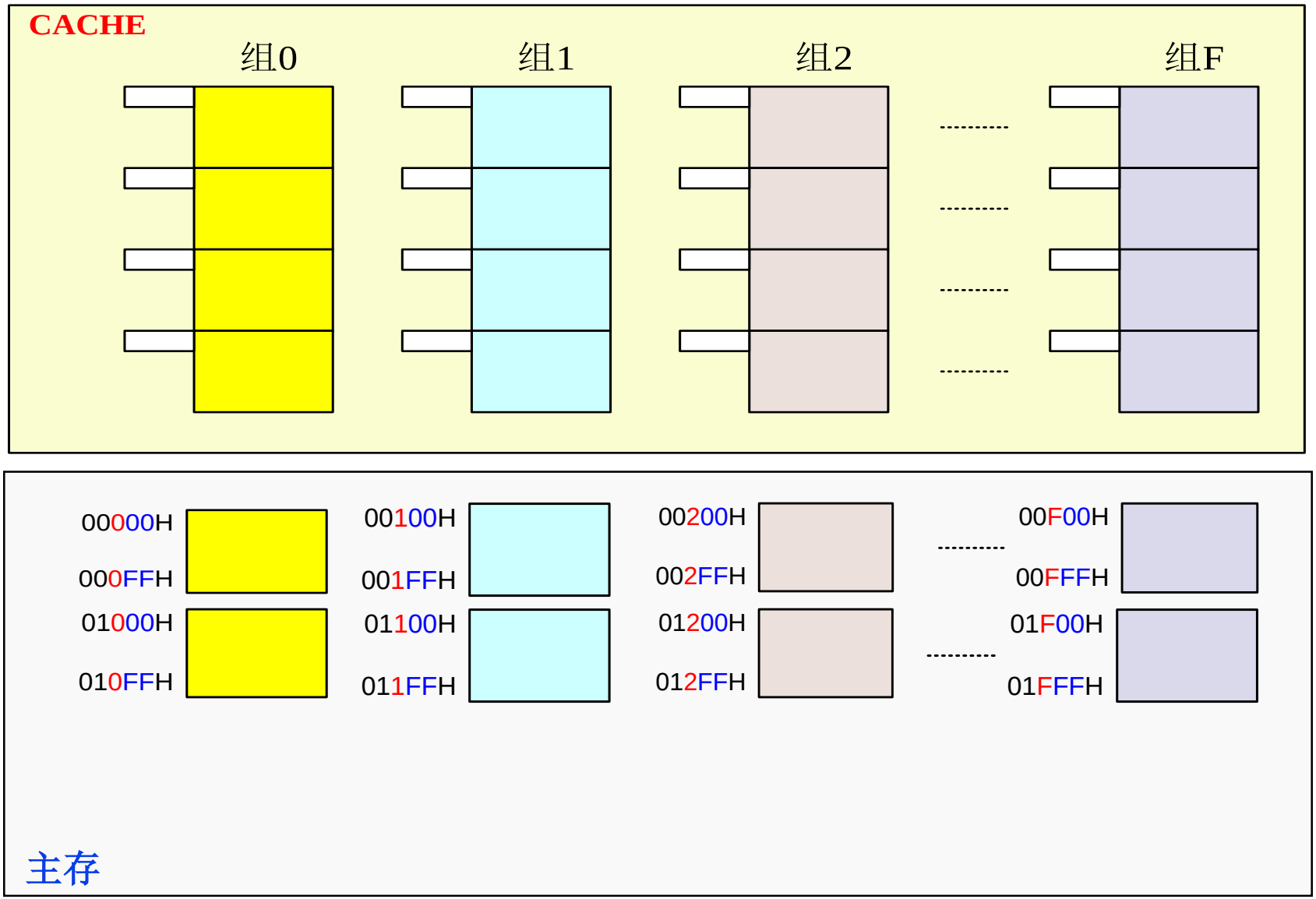
❖举例

- 主存容量1M 字节，4路组相联（每组包含4个Block）Cache容量16K字节，Block大小256 字节
- Cache分多少组？每组包含多少块？
- Cache的Tag需要多少位？

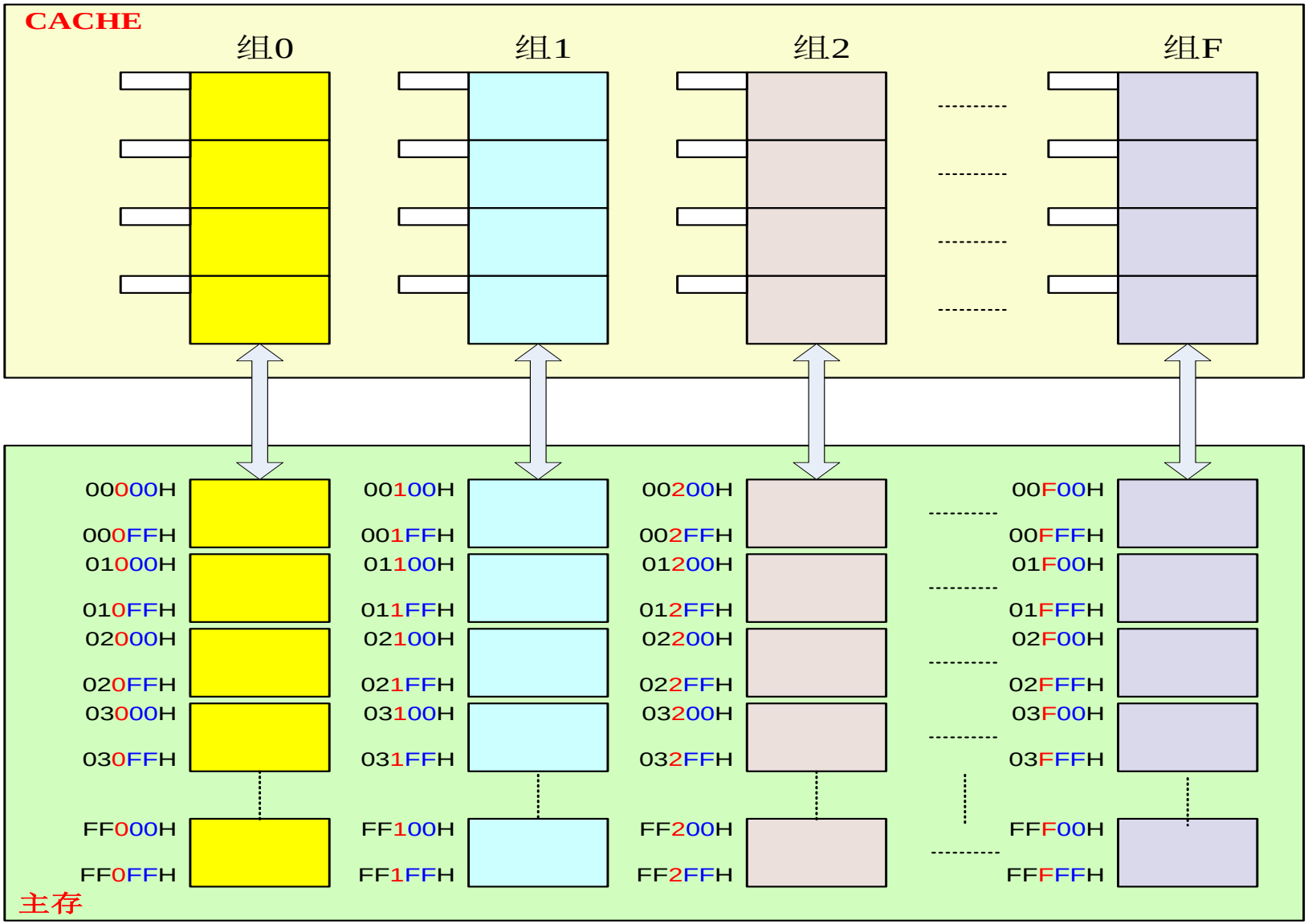
解：

- Cache 组数＝ $2^{14} \div (2^8 \times 2^2) = 2^4 = 16$ 组
- 主存每组块数＝ $2^{20} \div (2^8 \times 2^4) = 2^8 = 256$ 块/组
- 主存地址：20 位，其中高8 位为组内块地址，中间4 位为组地址，低 8位为块内地址
- Cache的Tag应该为 8 位。

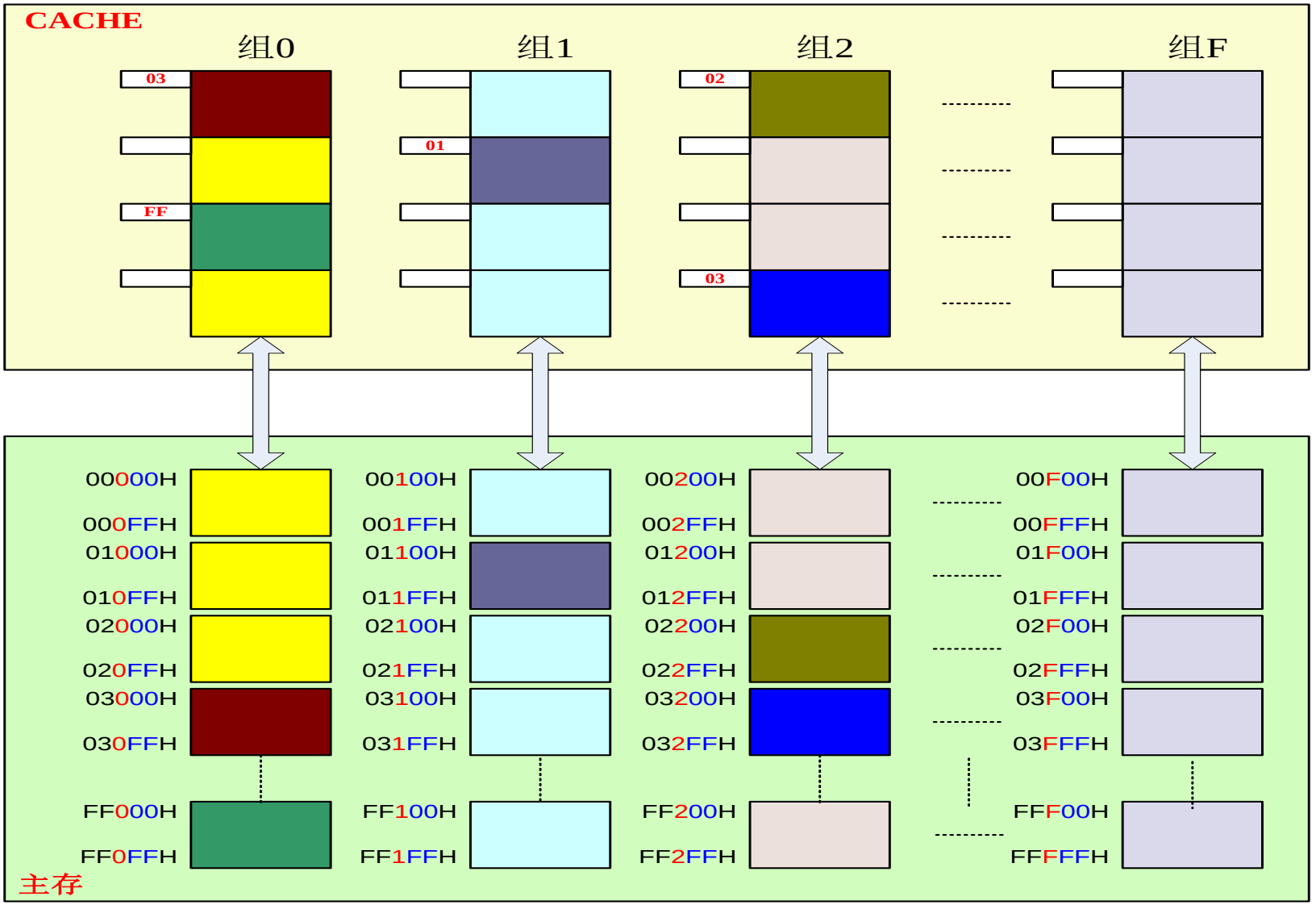
2.2 Cache与主存之间的映射—组相联



2.2 Cache与主存之间的映射—组相联



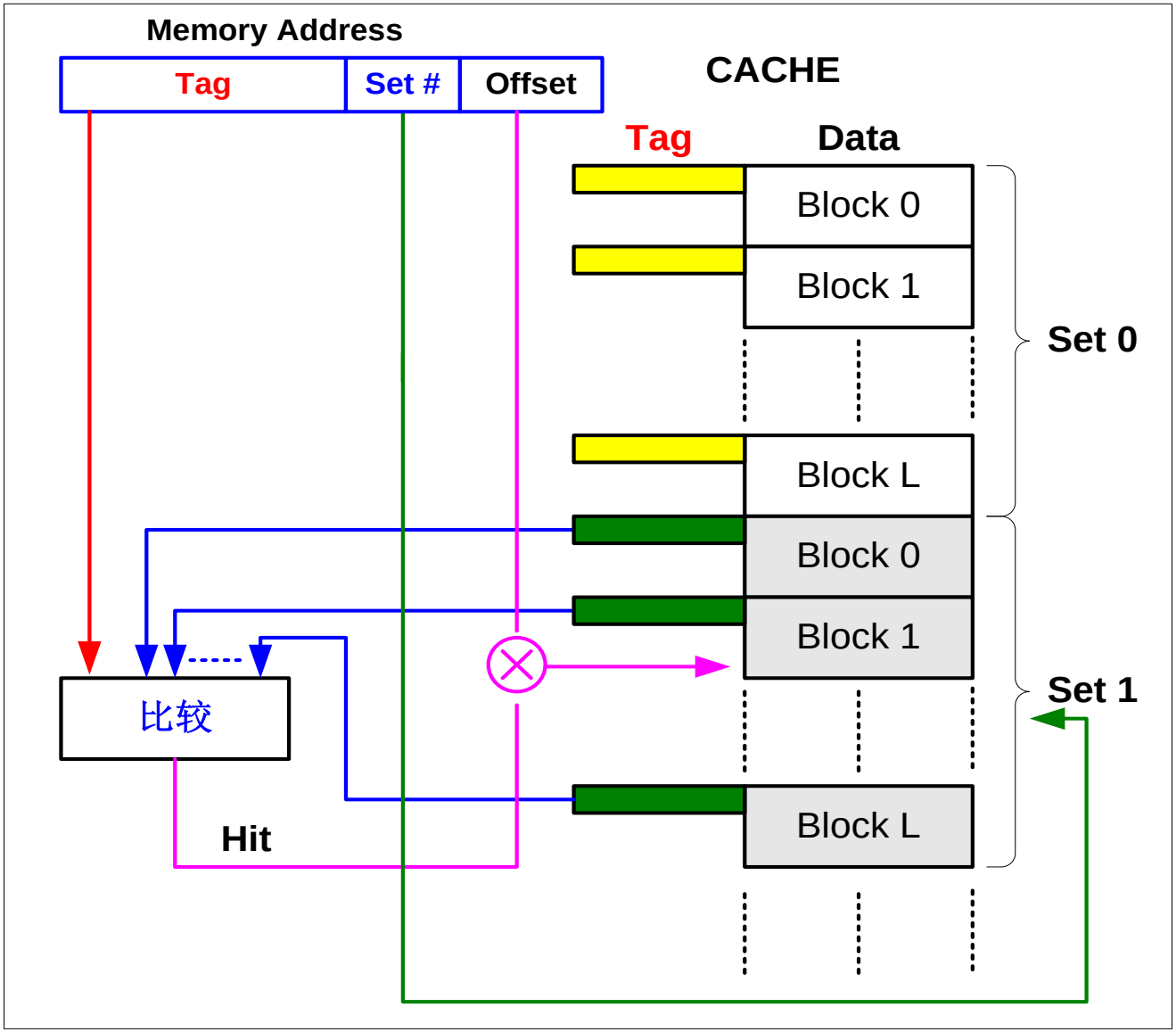
2.2 Cache与主存之间的映射—组相联



2.2 Cache与主存之间的映射—组相联

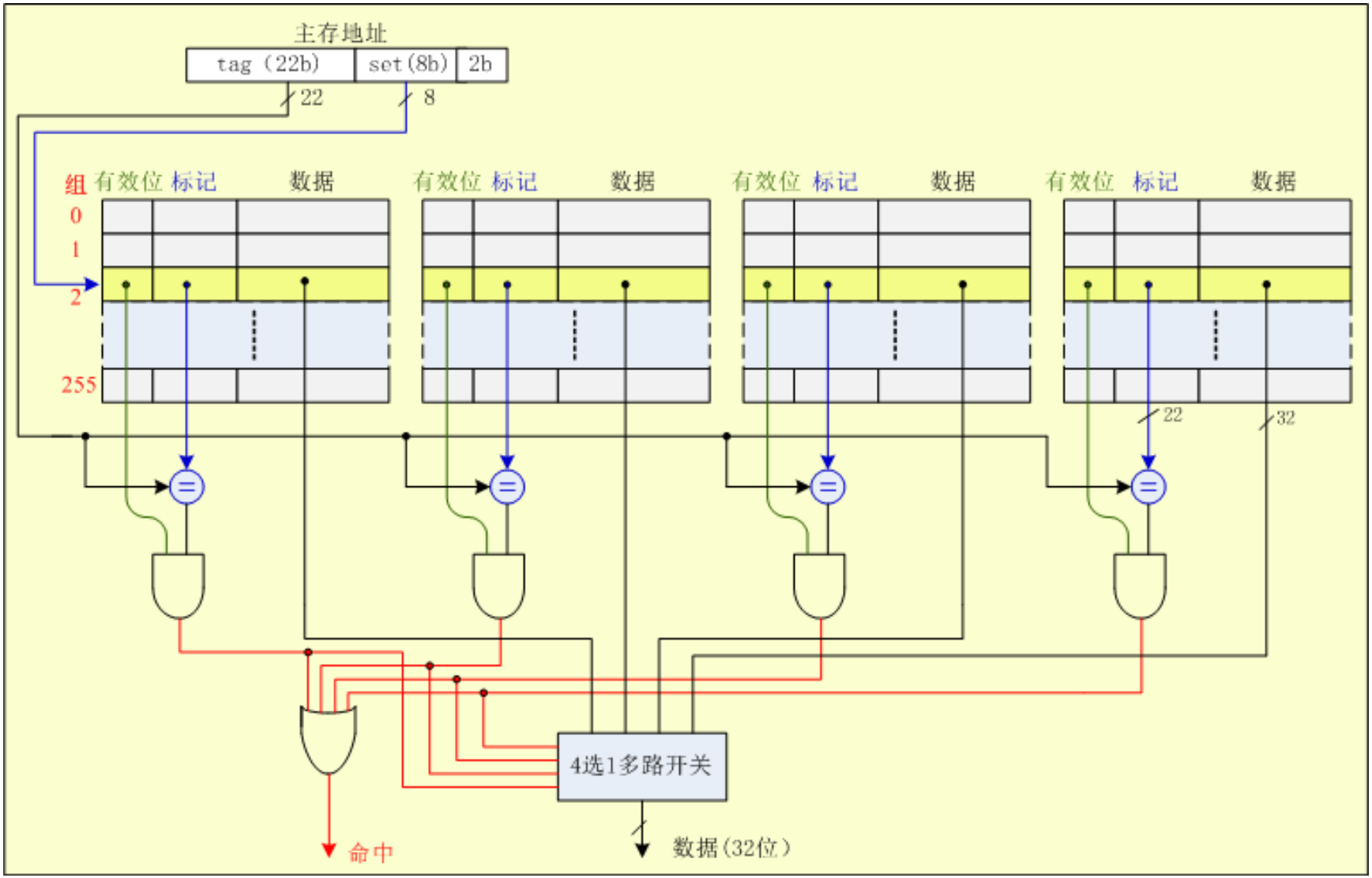
组相联Cache组织

- 同时与一个组内的所有Tag进行比较，N路组相联Cache则需N个比较器；
- 数据访问与比较并行执行。



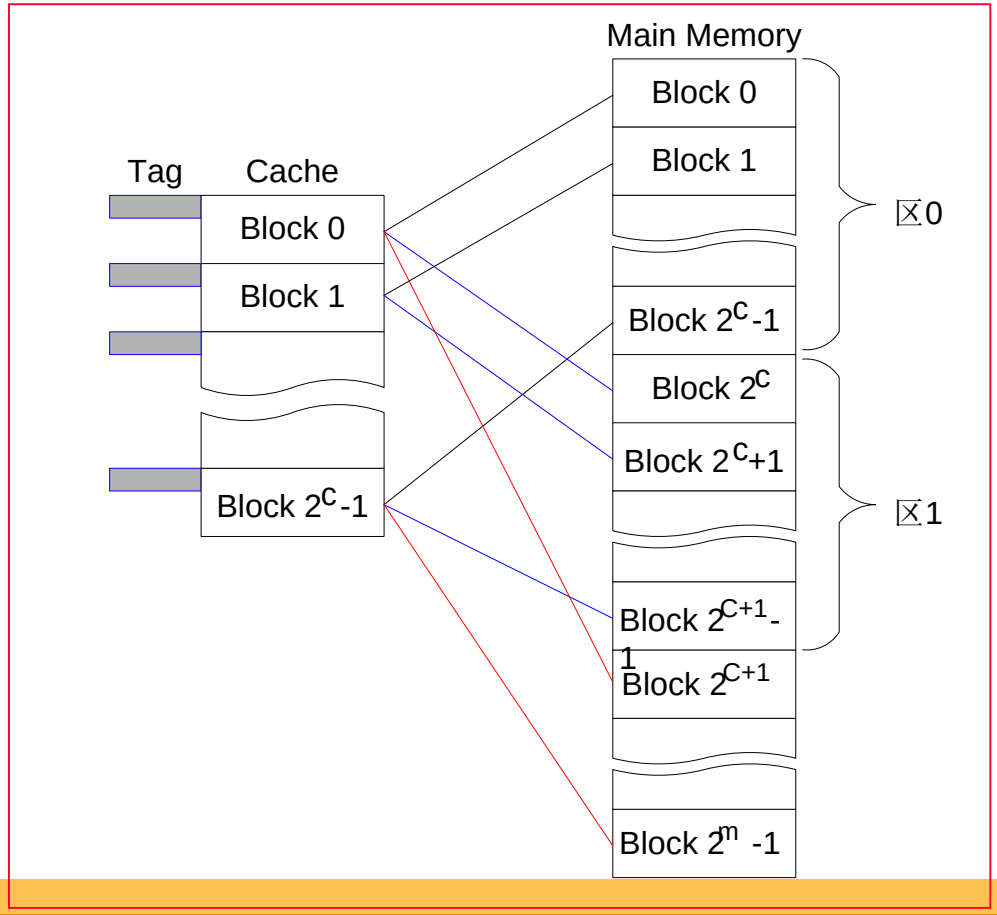
2.2 Cache与主存之间的映射—组相联

- Cache示例：Cache容量4KB，4路组相联，数据块大小4B，主存地址32位。



2.3 Cache与主存之间的映射—直接映射

- 直接（Direct）
  - 主存中的某一块 J 映射到Cache中的固定块 K， K = J Mod M， 其中M是Cache包含的块数。
  - 实际上是将主存按Cache的大小分区，一个区内的各块分别与Cache的对应各块映射。



2.3 Cache与主存之间的映射—直接映射

❖直接映射

- ➢主存的地址格式：

Tag	Index	Offset

- ➢Cache的Tag内容：主存中与该Cache数据块对应的数据块的区地址。

❖举例

- 主存容量16M字节，Cache容量64K字节，Block大小16 Bytes
- Cache包含多少块？
- Tag应该多少位？

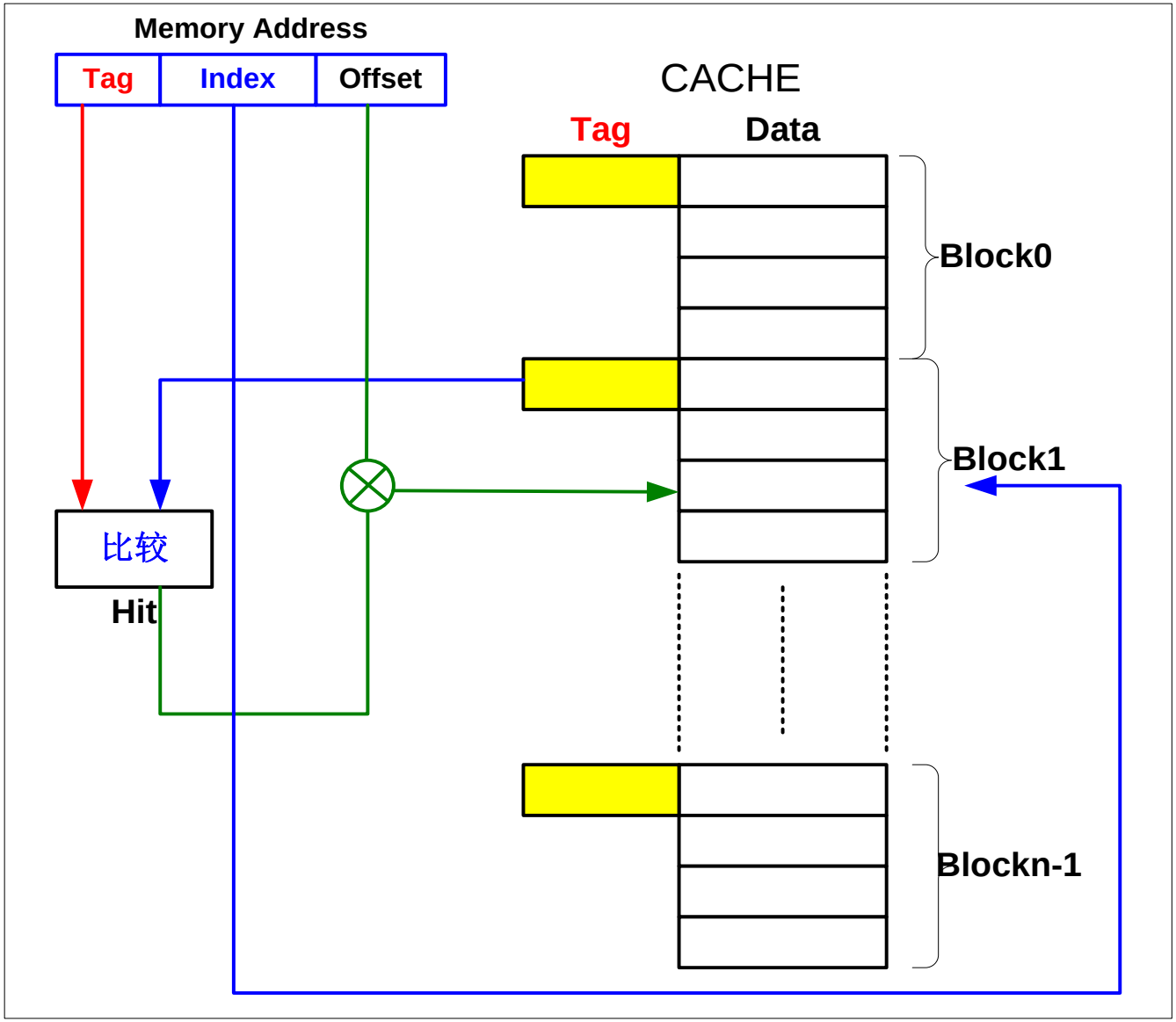
解：

- 主存: 2^{20} Blocks, 分成2^{8}个区，每个区2^{12}Blocks
- Cache: 2^{12} Blocks
- 主存地址：24位，其中高 8 位区地址，中间12位为区内块地址，低4位为块内地址
- Cache的Tag应该为 8位。

2.3 Cache与主存之间的映射—直接映射

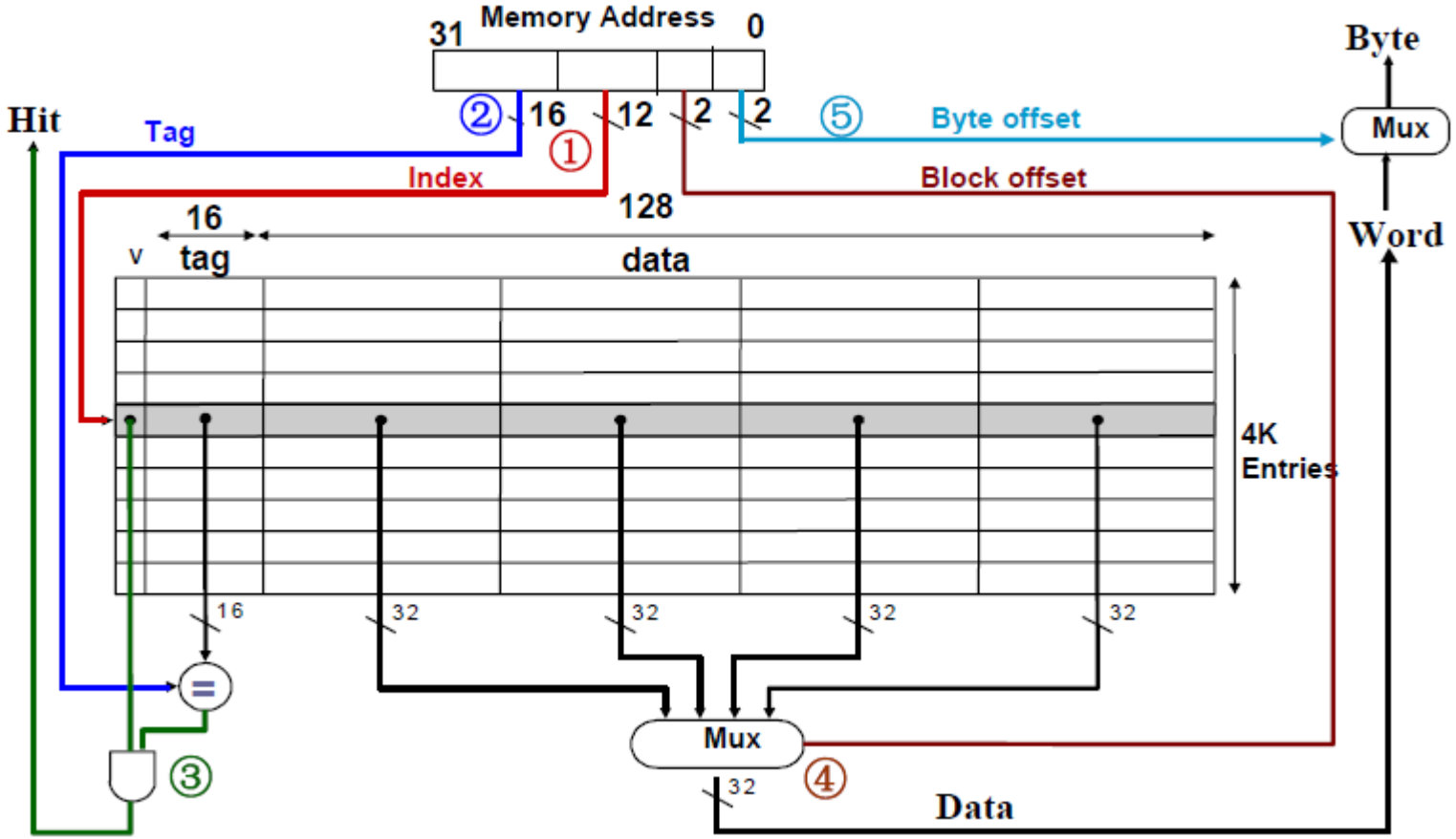
直接映射Cache组织

- 只需与一个Tag进行比较。



2.3 Cache与主存之间的映射—直接映射

示例：假定主存和Cache之间采用直接映射方式，块大小为16B。Cache的数据区容量为64KB，主存地址为32位，按字节编址。



第七讲：高速缓冲存储器

一．Cache的原理

1. 程序访问的局部性原理
2. Cache的结构与工作原理

二．Cache的映射机制

1. 全相联映射
2. 组相联映射
3. 直接映射

三．**Cache的替换策略**

四．Cache性能分析

Cache的缺失处理

■缺失损失

- ➢CPU访问Cache缺失时，CPU必须等待数据装入Cache后才能访问Cache，这期间的时间损失称为缺失损失。
- ➢取出块的时间：第一个字的延迟时间（存储器访问）+ 块的剩余部分的传送时间。
- ➢Cache的存储组织对缺失损失具有很大的影响。

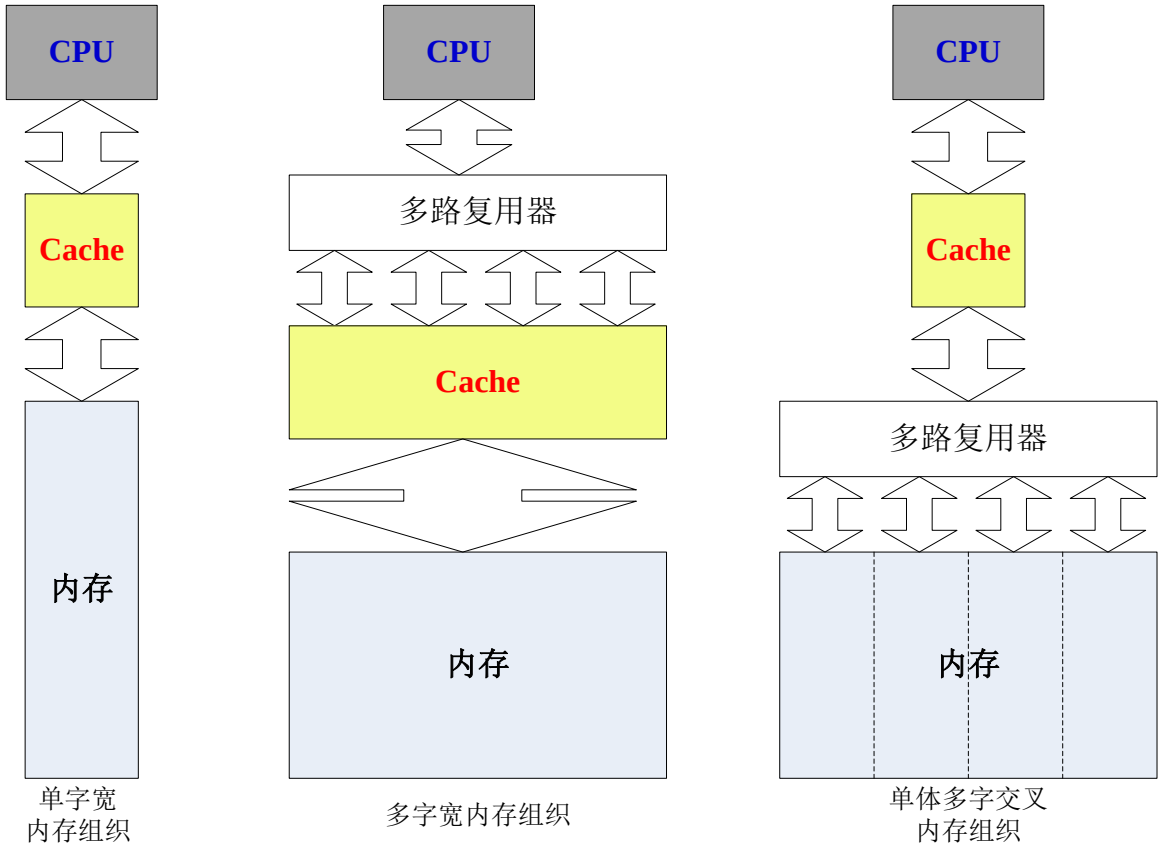
Cache的缺失处理

■缺失损失示例

假定：存储总线时钟周期为T；发送地址需1个T，访问DRAM需要15T，传输一个字需要1个T。Cache块大小为4字。

计算三种不同存储组织的缺失损失。

- 单字宽的缺失损失
  1+4*15+4*1=65T
- 2字宽的缺失损失
  1+2*15+2*1=33T
- 4字宽的缺失损失
  1+1*15+1*1=17T
- 4字交叉的缺失损失
  1+1*15+4*1=20T

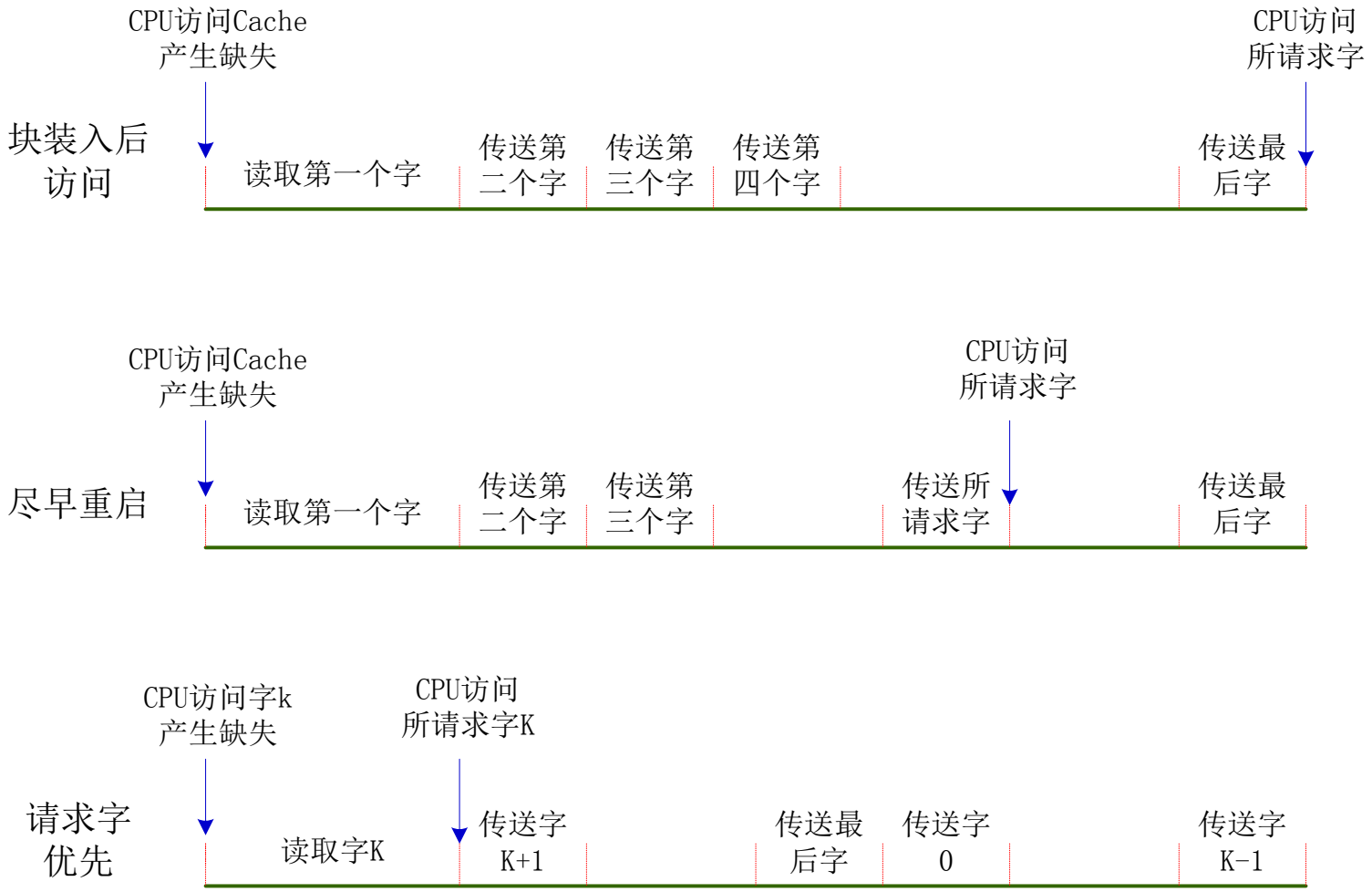


Cache的缺失处理

- 缺失处理（以读操作为例，写操作比较复杂）
  - 缺失数据块中各字按顺序全部装入Cache后，再从Cache中访问所请求的字（也是引起缺失的字）；
  - 尽早重启（ **early restart**）：缺失数据块中各字按顺序装入Cache，一旦所请求的字装入Cache，CPU立即访问该字，控制机构再继续传送剩余数据到cache ；
  - 请求字优先（**requested word first**）：所请求的字先装入Cache，CPU立即访问该字，控制机构再按照先从所请求字的下一个地址、再到块的起始地址的顺序继续传送剩余数据到cache 。

Cache的缺失处理

- 几种缺失处理方式



Cache块的替换

- 替换块的选择
  - 直接映射Cache：访问缺失时，被请求数据所在的块只能进入Cache的一个位置，占用该位置的数据块必须被替换掉；
  - 组相联Cache：访问缺失时，被请求数据所在块可以进入Cache某一组的任何位置，因此应在Cache对应组内选择一个数据块进行替换；
  - 全相联Cache：访问缺失时，被请求数据所在块可以进入Cache的任何位置，因此应在Cache中选择一个数据块进行替换。

Cache的替换策略

替换策略

- 最近最少使用法（LRU，Least-Recently Used）：记录每一个数据块的相对使用情况，最近没有被使用的块被替换。
- 先进先出法（FIFO，First-In-First-Out）：最先装入数据的块被替换；
- 最小使用频率法 （LFU，Least-Frequently Used）：记录每一个数据块的使用频率，使用次数最少的被替换。
- 随机法（RAND，Random）：随机选择一个数据块进行替换。

Cache的替换策略

- **替换算法的实现**
  - 一般由硬件电路实现，因此应以简单、便于硬件实现为宜。
- **LRU的实现（计数器法）**
  - 缓存的每一块都设置一个计数器；
  - 被调入或者被替换的块， 其计数器清 0，而其它的计数器则加 1；
  - 访问命中时，所有块的计数值与命中块的计数值进行比较，如果计数值小于命中块的计数值， 则该块的计数值加 1；如果块的计数值大于命中块的计数值，则数值不变。最后将命中块的计数器清为0。
  - 需要替换时，则选择计数值最大的块被替换。
- **FIFO的实现（计数器法）**
  - 例如Solar－16/65机Cache采用组相联方式，每组4块，每块都设定一个两位的计数器，当某块被装入或被替换时该块的计数器清为0，而同组的其它各块的计数器均加1，当需要替换时就选择计数值最大的块被替换掉。

Cache举例

某计算机主存容量 16MB，Cache 容量 16KB，采用 4 路组相联（每组 4 块）映射方式和 LRU 替换策略，每个数据块 16 字节。假设 Cache 中第 8 组（组地址为 8）的 4 个数据块的装入情况及 Tag 内容如下表（装入位为 1 表示数据块已装入）。

装入位	Tag
1	F40H
1	430H
0	218H
1	030H

（1） Cache 分多少组？（2 分）

（2） 给出主存的地址格式；（2 分）

（3） 若 CPU 要依次读取主存地址 430082H、2F8086H、03008AH、F40088H、063081H 单元中的数，则读取哪几个地址单元的数会产生 Cache 失效？5 个数读取结束时上表中装入位和 Tag 内容各是多少。

第七讲：高速缓冲存储器

一．Cache的原理

1. 程序访问的局部性原理
2. Cache的结构与工作原理

二．Cache的映射机制

1. 全相联映射
2. 组相联映射
3. 直接映射

三．Cache的替换策略

四．Cache性能分析

Cache的容量

- **Cache的容量**
  - 不作特殊申明时，**Cache**的容量指**Cache**数据块的容量；
  - **Cache**实际总的存储容量实际上还包含**tag**和**valid bit**的位数。

Cache的存储布局

- 例：假设一直接映射像**Cache**，有**16KB**数据，块大小为**4**个字（**32**位字），主存地址**32**位，每个数据块包括**1**位有效位，计算实现该**Cache**所需总存储容量？
  - **Cache**每数据块大小：$4\times32 = 128$ bits $= 2^4$ Bytes;
  - **Cache**块数：$16K \div 2^4 = 2^{10}$ 块；
  - **tag**位数：$32 - 10 - 4 = 18$ bits
  - 有效位：**1**位
  - **Cache**实际总容量：$2^{10} \times (128+18+1) = 147K$位 $\approx 18.4KB$

Cache的容量

- 例：假设一4路组相联Cache，64KB数据存储空间大小64KB，块大小为16字节，主存地址32位，主存一个字包含4个字节，Cache采用写回策略，每个数据块包括1位有效位，Cache每个字用1位脏位来表示是否被修改。
  1. 计算实现该Cache所需总存储容量？
- 解答
  - Cache每数据块大小：16×8 = 128 bits = 2^4 Bytes;
  - Cache块数：64K ÷ 2^4 = 2^{12} 块;
  - Cache组数： 2^{12} ÷ 4 = 2^{10} 组
  - tag位数：32 – 10 – 4 = 18 bits
  - 每个Block有效位：1位
  - 每个Block脏位：4位（1个Block包含4个字）
  - Cache实际总容量：2^{12} × (128+18+1+4) = 618496位 ≈ 75.5KB

Cache的性能计算

■存储访问时间

若：T_m为主存储器的访问周期；

T_c为Cache的访问周期；

H为Cache命中率

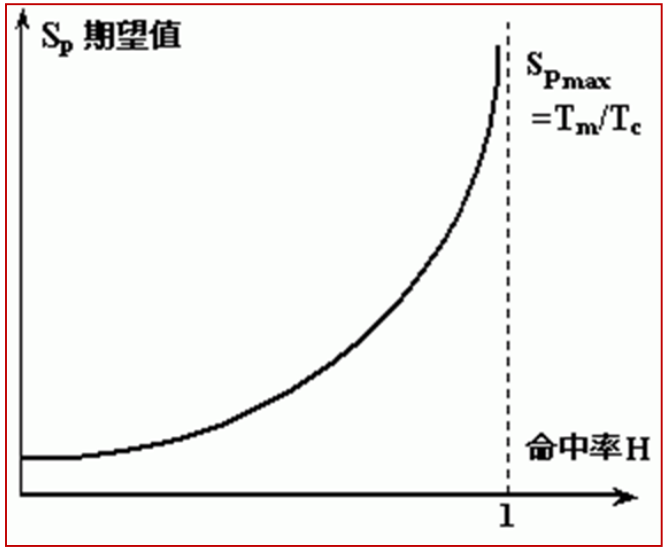
则存储系统的等效访问周期T为：

$$T = T_c \times H + T_m \times (1-H)$$

■加速比SP（Speedup）

存储系统的加速比S_p为：

$$S_p = \frac{T_m}{T} = \frac{T_m}{H \times T_c + (1-H) \times T_m} = \frac{1}{(1-H) + H \times \frac{T_c}{T_m}}$$



加速比与命中率的关系

命中率对存储器性能的影响

- 设平均访问时间T：

$T = HT_C + (1 - H)(T_C + T_M)$

$= T_C + (1 - H)T_M$

- 例1. 若命中率H=0.85, Tc=1 ns, T_M =20ns，则平均访问时间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

How high of a hit ratio?

Suppose we can easily build an on-chip static memory with a 4 nS access time, but the fastest dynamic memories that we can buy for main memory have an average access time of 40 nS. How high of a hit rate do we need to sustain an average access time of 5 nS?

$$\alpha = 1 - \frac{t_{ave} - t_c}{t_m} = 1 - \frac{5-4}{40} = 97.5\%$$

WOW, a cache really needs to be good?

访问速度与命中率的关系非常大！

Cache性能计算举例

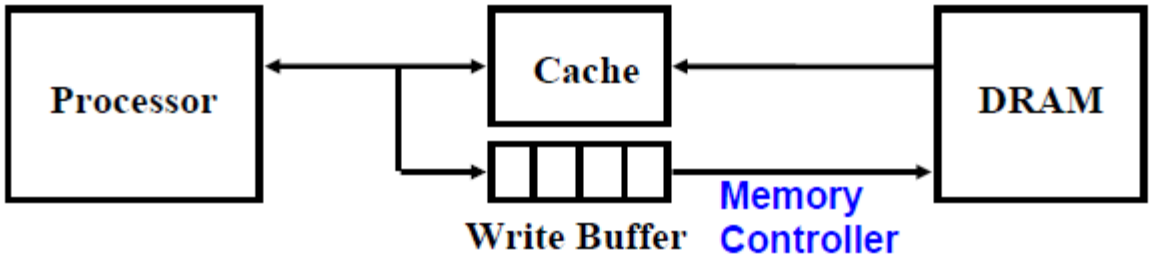
某计算机的存储系统由Cache、主存和用于虚拟存储的磁盘组成。CPU总是从Cache中获取数据。若所访问的字在Cache中，则存取它需要10ns；将所访问的字从主存装入Cache需要40ns；而将它从磁盘装入主存则需要2μs。假定Cache的命中率为0.9，数据不在Cache时主存的命中率为0.6，计算该系统访问一个字的平均存取时间。

Cache与主存的数据一致性

- 处理Cache读比Cache写更容易，指令Cache比数据Cache容易设计
- 对于写命中，有Two options
  - Write Through (通过式写、写直达、直写)
    - 同时写Cache和主存单元
    - What!!! How can this be? Memory is too slow(>100Cycles)?
      10%的存储指令使CPI增加到：1.0+100x10%=11
    - 使用写缓冲（Write Buffer）
  - Write Back (一次性写、写回、回写)
    - 在失靶时一次写回Cache块，每块有个修改位（“dirty bit-脏位”）
    - 大大降低主存带宽需求，控制可能很复杂
- 对于写不命中，有Two options
  - Write Allocate (写分配)
    - 将主存块装入Cache，然后更新相应单元
    - 试图利用空间局部性，但每次都要从主存读一个块
  - Not Write Allocate (非写分配)
    - 直接写主存单元，不装入主存块到Cache

直写Cache可用非写分配或写分配
写回Cache通常用写分配 为什么？

Cache与主存的数据一致性



- 在 Cache 和 Memory之间加一个Write Buffer
  - Processor: 同时写数据到Cache和Write Buffer
  - Memory controller: 将缓冲内容写主存
- Write buffer (写缓冲) 是一个FIFO队列
  - 一般有4项
  - 在存数频率<<DRAM写（周期）频率情况下，效果好
- 最棘手的问题
  - Store frequency > 1 / DRAM write cycle(频繁写)时，使 Write buffer 饱和(溢出)，会发生阻塞

举例：失靶带来的损失到底多大？

设Code Cache的失靶率为2%，Data Cache的失靶率为4%。假定一个处理器在没有任何存储器阻塞时的CPI为2，miss penalty为100个时钟。如果用SPECint2000来衡量，则使用完全没有失靶的完美Cache，处理器的速度会快多少？

分析过程如下：

指令的失靶时钟数为：Ix2%x100=2.0xI

SPECint2000的访存指令(Load和Store)频度为：36%，所以

数据的失靶时钟数为：Ix36%x4%x100=1.44xI

指令和数据总的失靶时钟数为：2xI+1.44xI=3.44I，也即：

平均每条指令要有3.44个时钟处在存储器阻塞状态

因此，因为存储器阻塞而使得CPI数增大到2+3.44=5.44. 故：

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{\text{IxCPIstallxClock cycle}}{\text{IxCPIperfectxClock cycle}} = \frac{5.44}{2}$$

如果Cache不发生失靶，则处理器速度会快2.72倍。

举例：处理器速度提高而存储器不变时的情况

- 例1：假定上例中CPI减为1，时钟宽度不变，则：

  因为存储器阻塞而使得CPI数增大到1+3.44=4.44. 故：

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{\text{IxCPIstallxClock cycle}}{\text{IxCPIperfectxClock cycle}} = \frac{4.44}{1}$$

  由此可知：存储器阻塞所花时间占整个执行时间的比例从：
  3.44 / 5.44=63% 上升到 3.44 / 4.44=77%

  结论：CPI越小，Cache阻塞的影响越大

- 例2：假定上例中时钟频率加倍， CPI不变，则：

  主存速度不太可能改变，故绝对时间不变，所以miss损失为200个时钟。

  每条指令发生的总失靶时钟数为：(2%x200)+36%x(4%x200)=6.88

  故：存储器阻塞使得CPI数增大到2+6.88=8.88

$$\frac{\text{时钟快的机器的性能}}{\text{时钟慢的机器的性能}} = \frac{\text{IxCPIstallofslow xClock cycle}}{\text{IxCPIstalloffast xClock cycle/2}} = \frac{5.44}{8.88/2} = 1.23$$

  由此可知：时钟快的机器的性能只是较慢时钟机器的1.2倍。
  如果没有Cache失靶的话，应该是2倍！

  结论：CPU时钟频率越高，Cache缺失损失就越大

多级cache的性能

- 采用L2 Cache的系统，其缺失损失的计算如下：
  - 若L2 Cache包含所请求信息，则缺失损失为L2 Cache的访问时间
  - 否则，要访问主存，并取到L1 Cache和L2 Cache（缺失损失更大）
- 例子：有一处理器的CPI为1，所有访问能在L1 Cache中命中，时钟频率为5GHz。假定访问一次主存的时间为100ns，包括所有的缺失处理。设平均每条指令在L1 Cache中的缺失率为2%，若增加一个L2 Cache，访问时间为5ns，而且容量大到使L2 Cache缺失率减为0.5%，问处理器速率提高了多少？

  解：如果只有一级Cache，则缺失只有一种：

  即：L1缺失(访问主存)，其缺失损失为：100nsx5GHz=500个时钟

  CPI=1+500x2%=11.0

  如果有二级Cache，则有两种缺失：

  即：L1缺失(访问L2Cache)：5nsx5GHz=25个时钟

  L2缺失(访问主存)：500个时钟

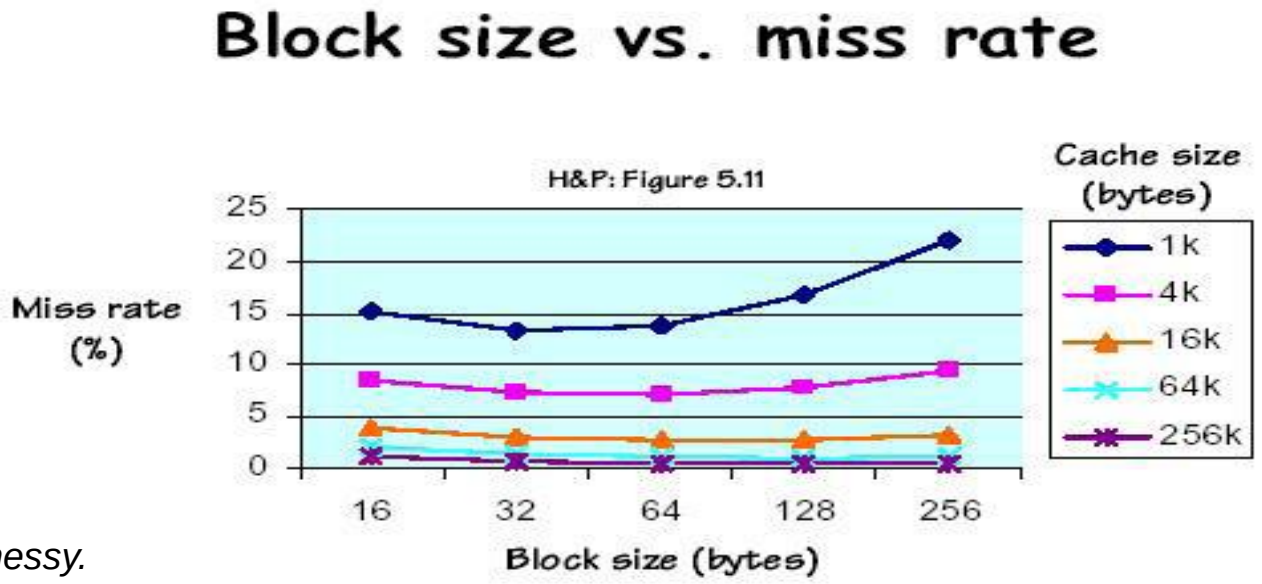
  CPI=1+25x2%+500x0.5%=4.0

  因此，二者的性能比为11.0/4.0=2.8倍

Cache命中率问题

- 块的大小与命中率：比较复杂。
  - 一般而言，增加块大小将降低缺失率（因为空间局部性），但块大小达到一定程度时，缺失率会随块大小的继续增加而上升（因为块数量下降带来块替换的增加）；
  - 单纯增加块大小带来缺失代价（缺失损失）的增大。

块大小与缺失率的关系



D. A. Patterson, J. L. Hennessy. Computer Organization and Design The hardware/software Interface(3th Ediiton). Elsevier Inc. 2007

CACHE举例

❖Pentiun的Cache

- 采用两级Cache结构。CPU内部Cache（Level 1 Cache）包括8K指令Cache和8K数据Cache，32Bytes/Line，采用2路组相联结构和LRU替换策略，数据Cache采用Write Back写策略（可以动态配置为Write-through）；外部Cache (Level 2 Cache)256KB或512KB，32Bytes/Line, 64Bytes/Line,128Bytes/Line，采用2路组相联结构。

❖PowerPc 620 Cache

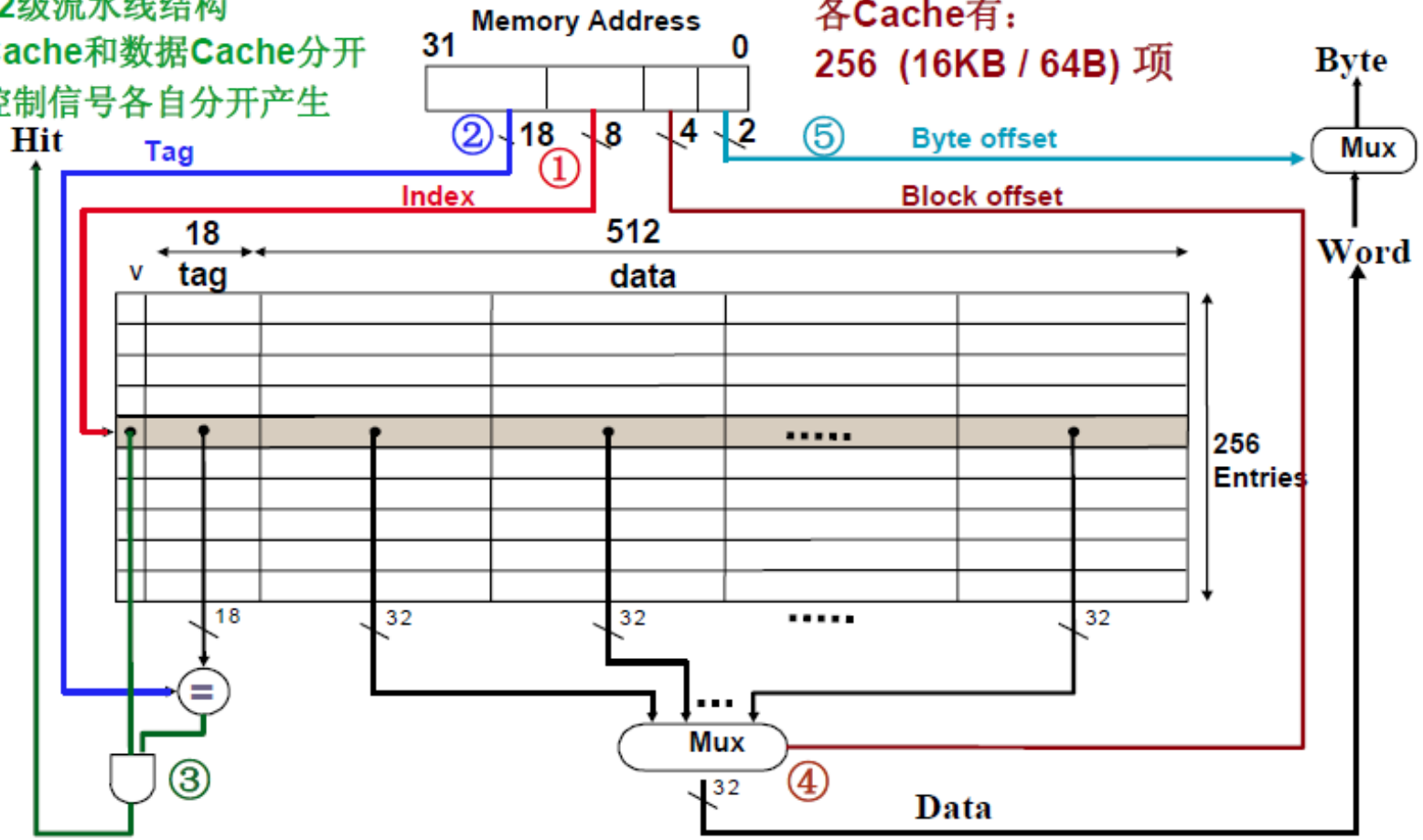
- 采用两级Cache结构。CPU内部Cache（Level 1 Cache）包括32K指令Cache和32K数据Cache，采用8路组相联结构。

Cache实例：内置FastMATH处理器

- FastMATH处理器是MIPS结构的嵌入式微处理器

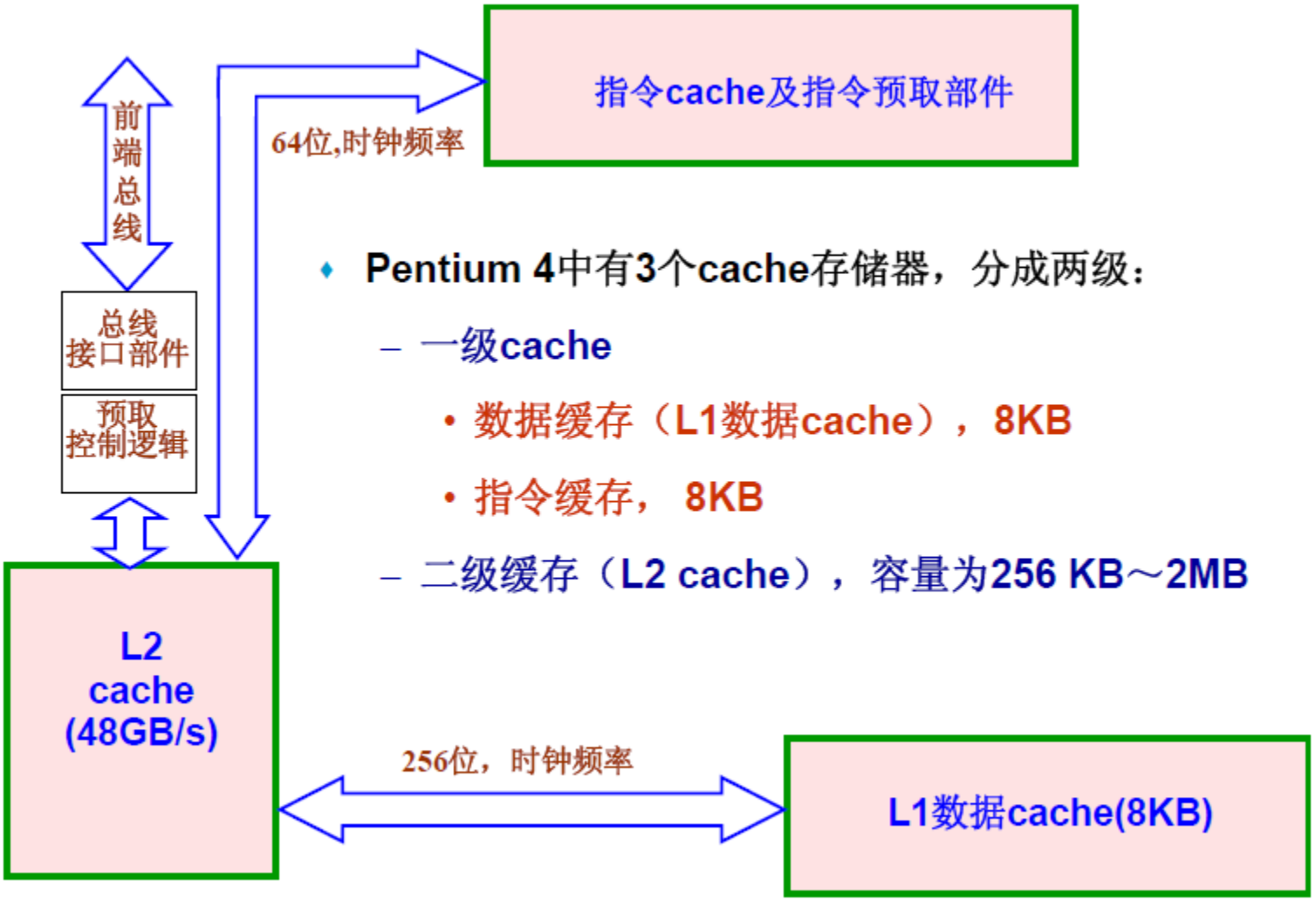
采用12级流水线结构
指令Cache和数据Cache分开
所以控制信号各自分开产生

各Cache有：
256 (16KB / 64B) 项



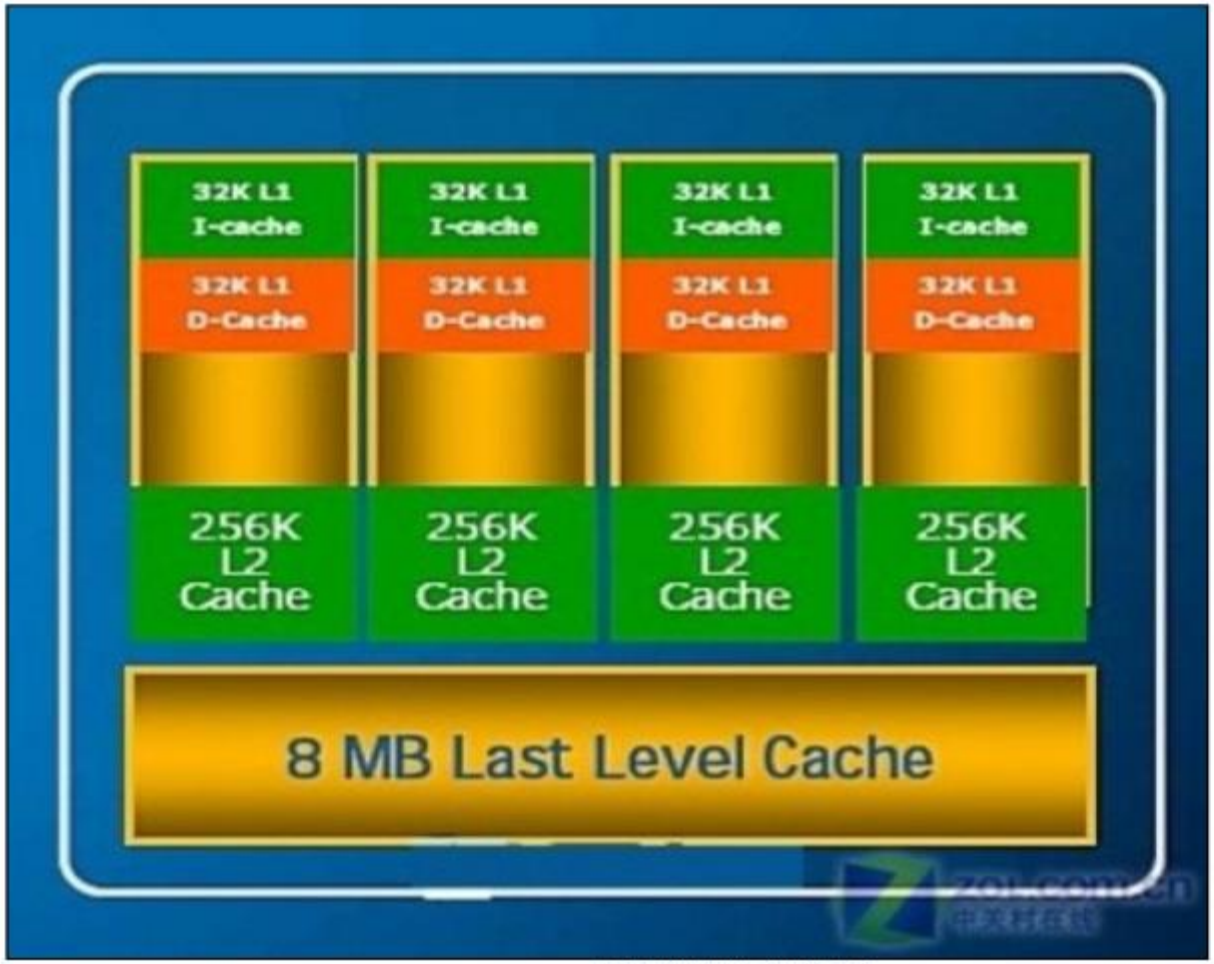
写操作比读操作复杂! 处理器提供了写通过和写回两种方式，由OS决定采用何种策略
SPEC2000int的指令、数据和综合缺失率分别为：0.4%, 11.4%, 3.2%

Pentium 4的Cache示例



- Pentium 4中有3个cache存储器，分成两级：
  - 一级cache
    - 数据缓存（L1数据cache），8KB
    - 指令缓存， 8KB
  - 二级缓存（L2 cache），容量为256 KB～2MB

多核处理器中的多级Cache



Nehalem Core i7处理器缓存结构图

Per core:

-32KB, 4-way L1 $I

-32KB, 8-way L1 $D

-256KB, 8-way L2

Shared

- 8 MB, 16-way L3