

# 第十七讲

---

## 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

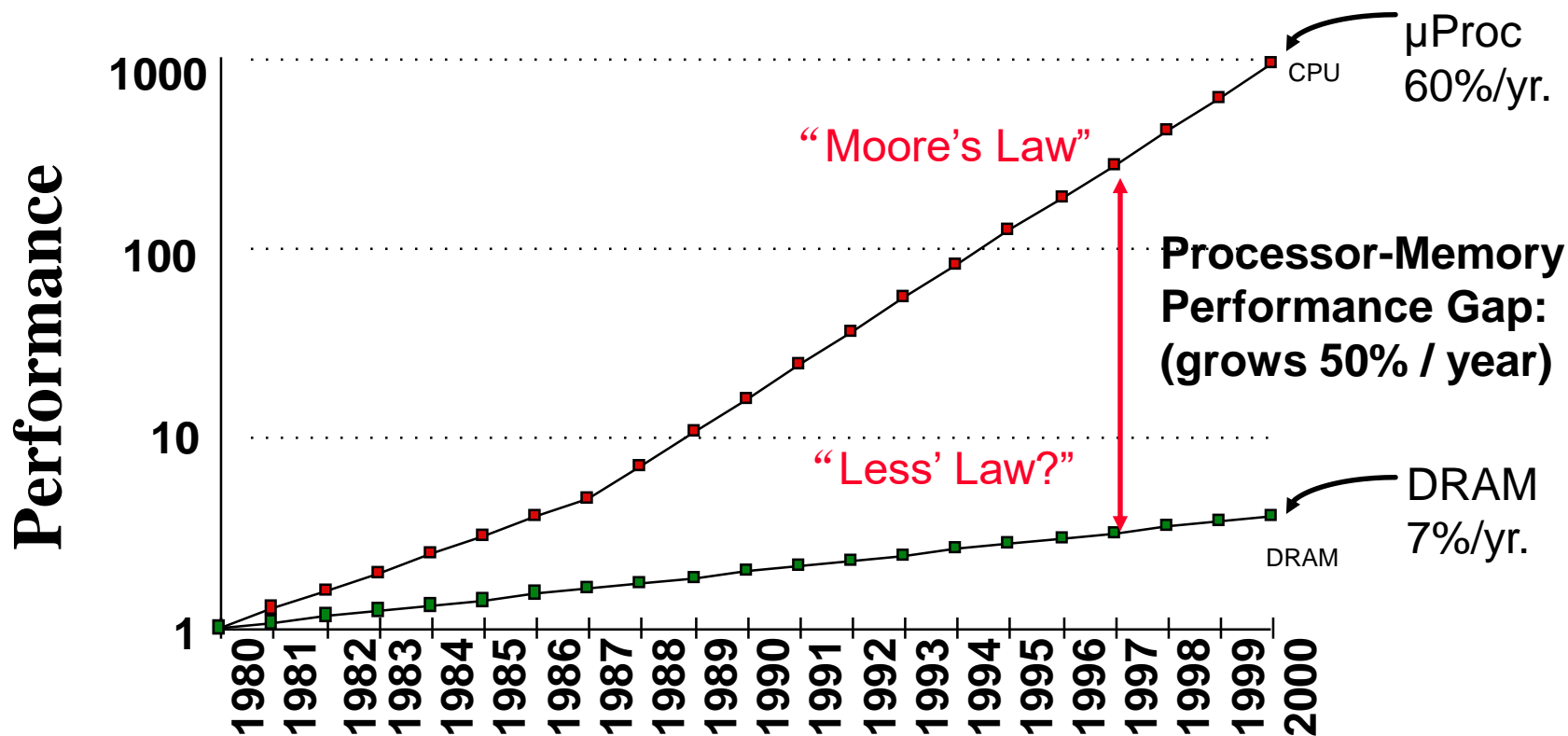
### 二. Cache的映射机制

1. 全相联映射
2. 组相联映射
3. 直接映射

### 三. Cache的替换策略

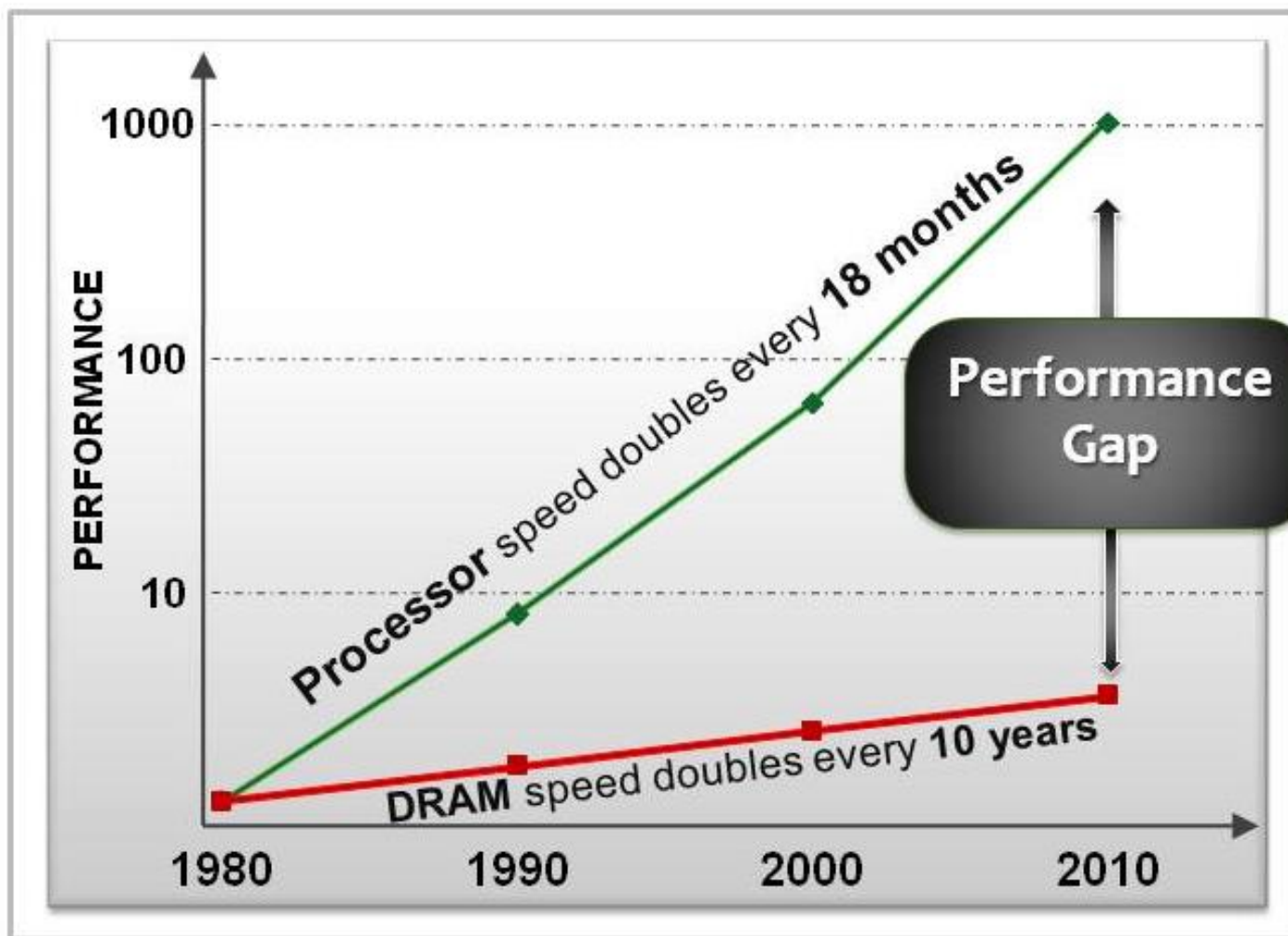
### 四. Cache性能分析

## 1.1 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.

## 1.1 处理器—DRAM存储器的性能差距



Source: acm.org & MoSys

©MoSys, Inc. 2010. All Rights Reserved.

# 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。

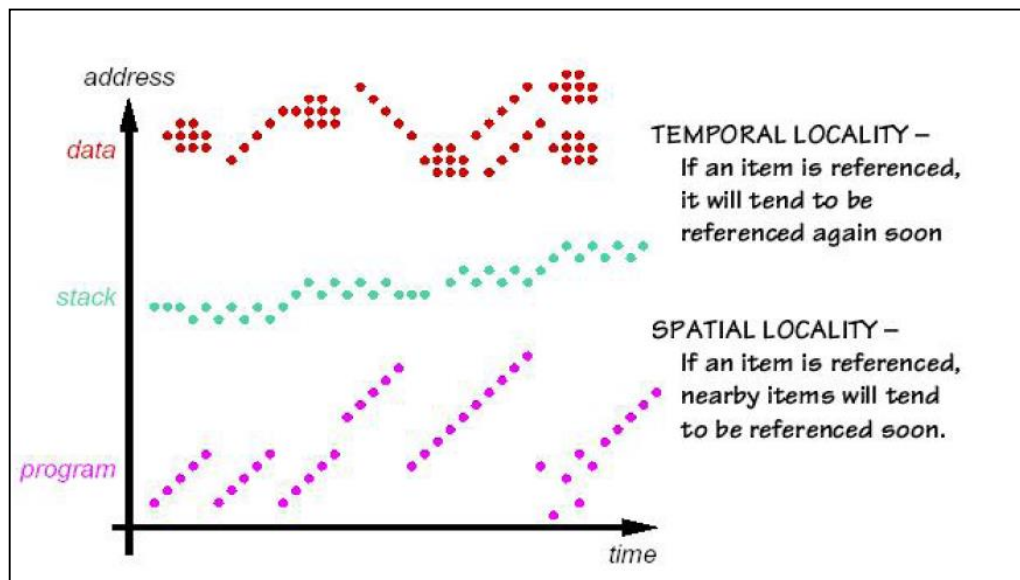
➤ **时间局部性(temporal locality)**：刚被访问过的存储单元可能不久又将被访问；

➤ **空间局部性(spatial locality)**：刚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 不同类型存储器的性能价格差异

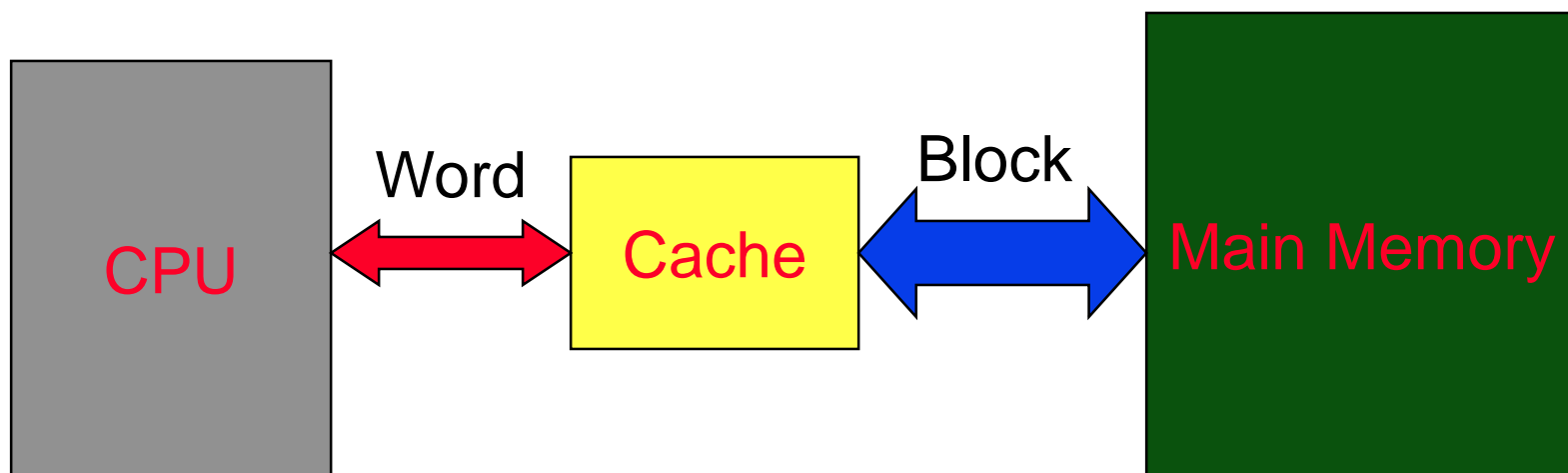
存储器技术	典型存取时间	价格（\$/GB，2004）
SRAM	0.5 ~ 5ns	\$4,000~\$10,000
DRAM	50 ~ 70ns	\$100~\$200
磁盘	$5 \times 10^6 \sim 20 \times 10^6$ ns	\$0.5~\$2

### ❖ 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。

## 1.2 高速缓冲存储器(Cache)的原理

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**！

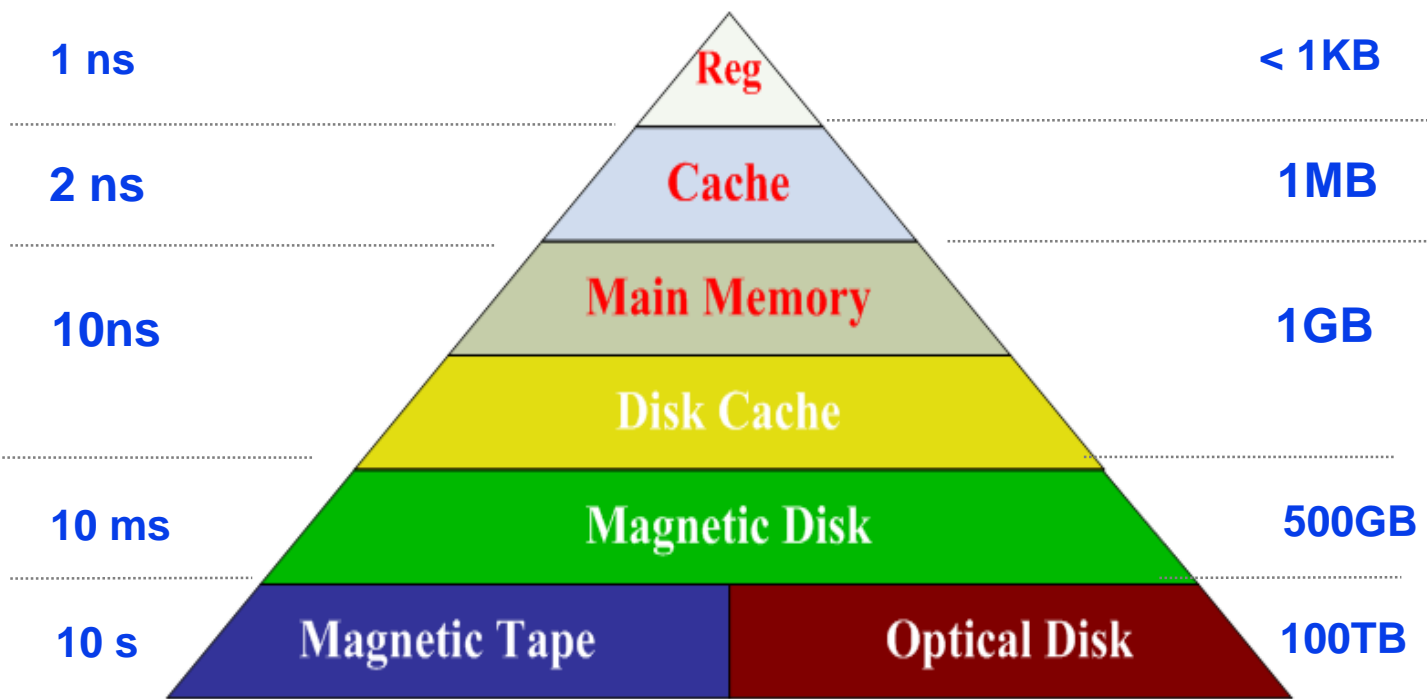




## 1.2 高速缓冲存储器(Cache)的原理 —— 存储层次结构

典型存取时间

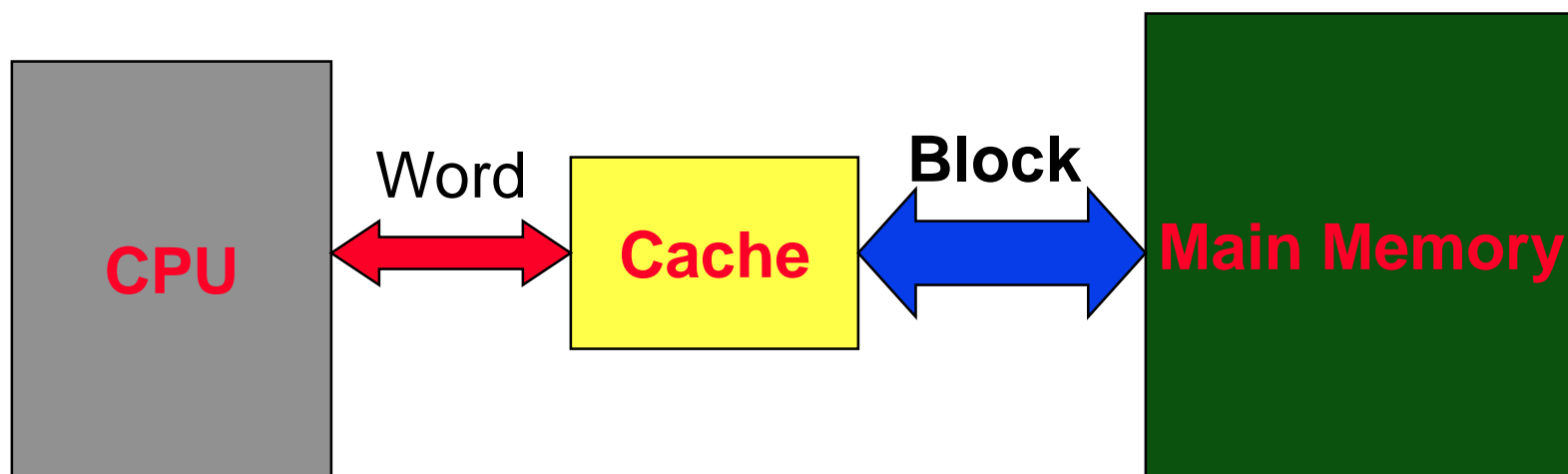
典型容量



- 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**DRAM**
  - 4) 如操作数不在**DRAM**，则访问硬盘，操作数从硬盘→**DRAM** →**cache**

## 1.2 高速缓冲存储器(Cache)的原理

- ❖ Cache与主存之间以数据块（Block）为单位进行数据交换
- ❖ 每个数据块包含若干个字或字节
- ❖ Cache块大小和主存块大小相等



## 1.2 高速缓冲存储器(Cache)的原理

---

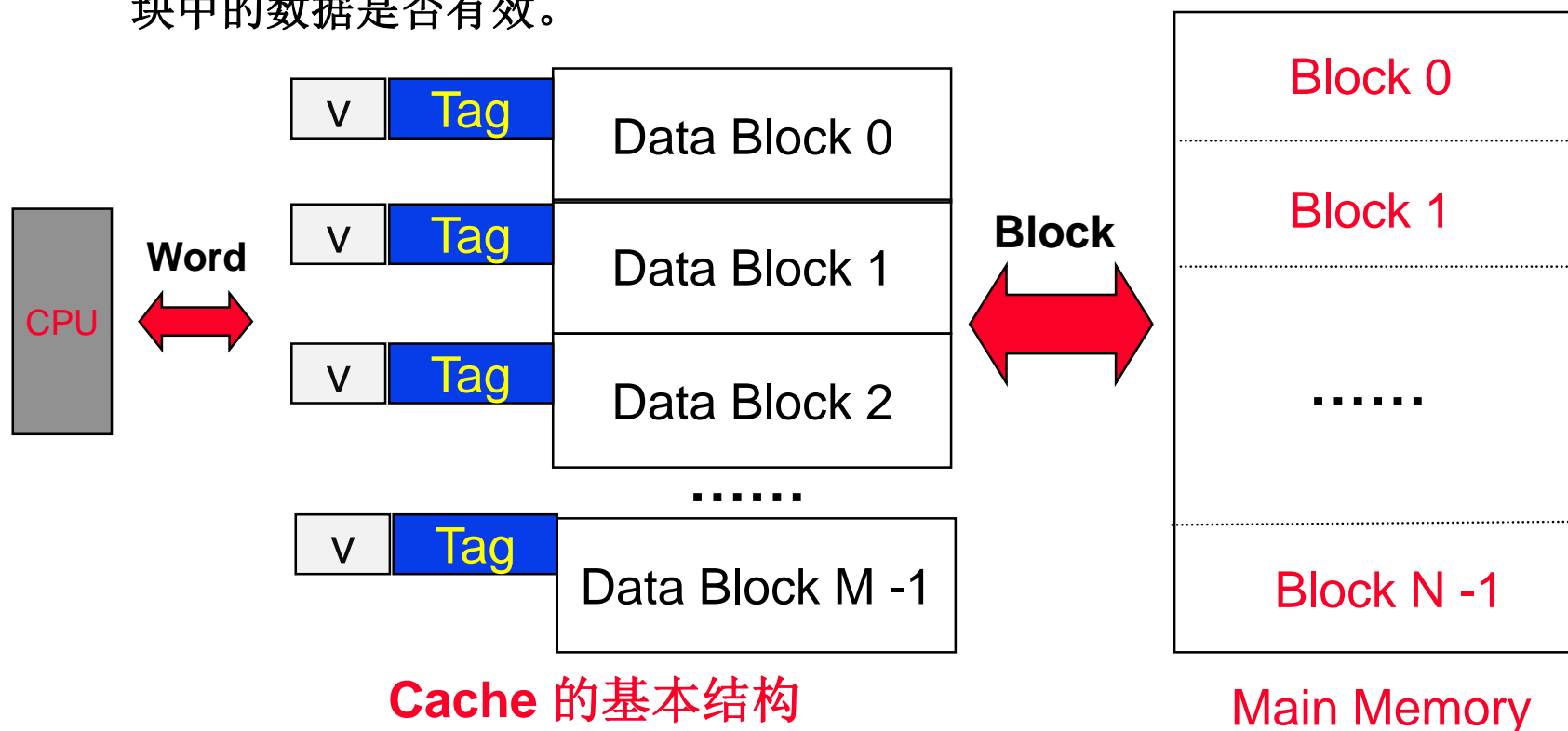
### ❖ Cache要解决的问题

- 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。以数据块Block（若干字）为单位，Cache块大小与主存块大小相同；
- **地址机构**：地址比较机制，地址转换机制，地址标记（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。



## 1.2 高速缓冲存储器(Cache)的原理

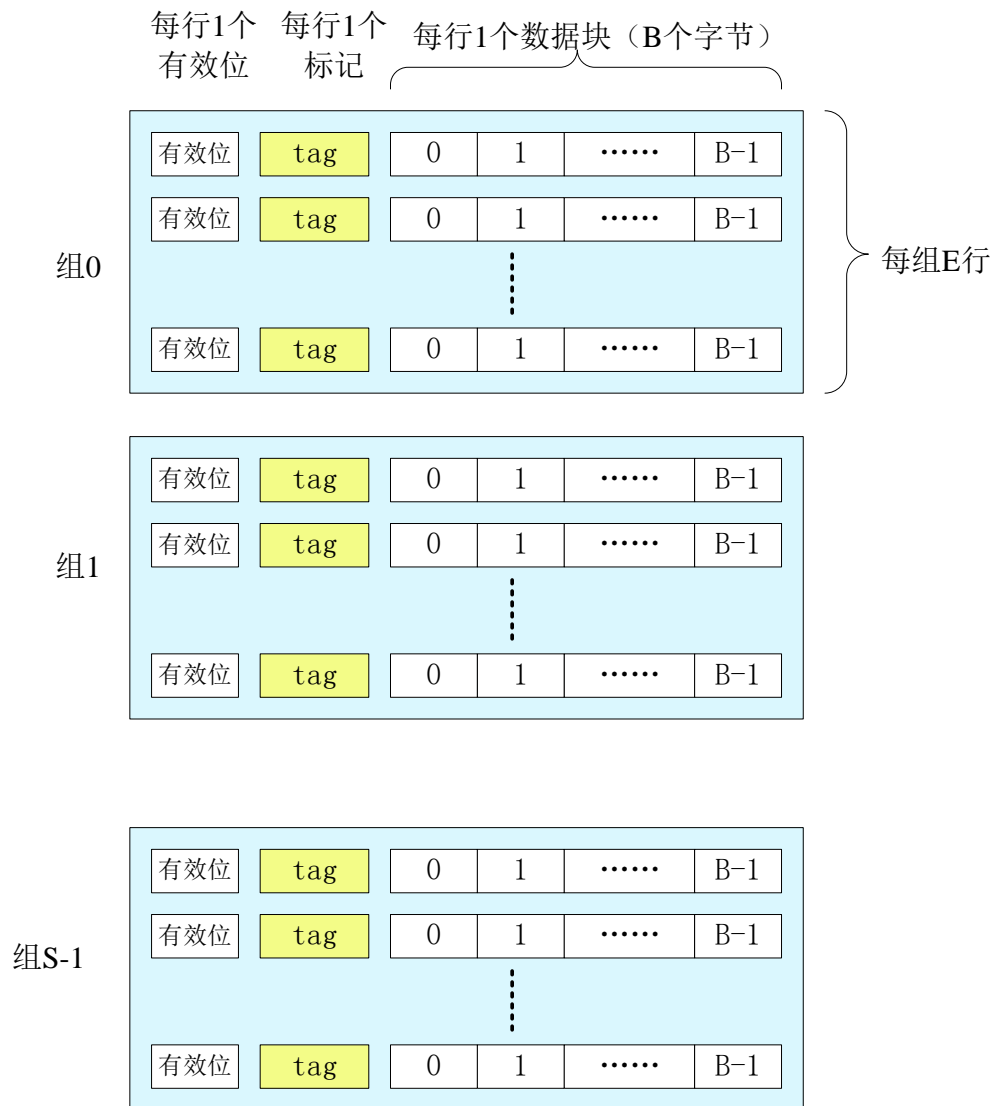
### ❖ Cache的有关术语

- **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，取决于主存一次读写操作所能完成的数据字节数，也表明主存与Cache之间局部总线的宽度。
- **标记 (tag)**：每一Cache数据块有一个标记字段，用来保存该Cache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

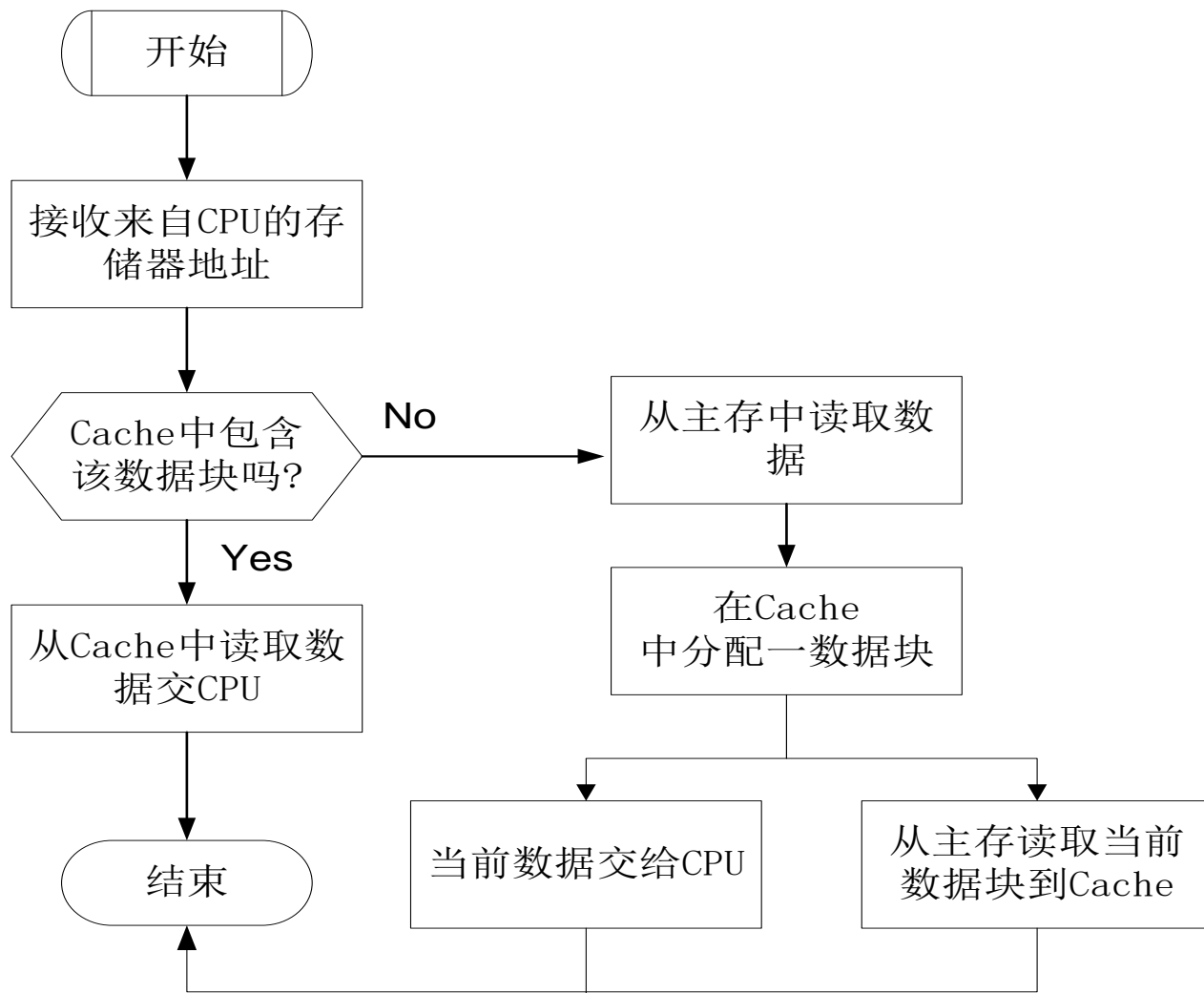
### ❖ Cache结构示意

- 分S组
- 每组E行
- 每数据块包含B个字节



## 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

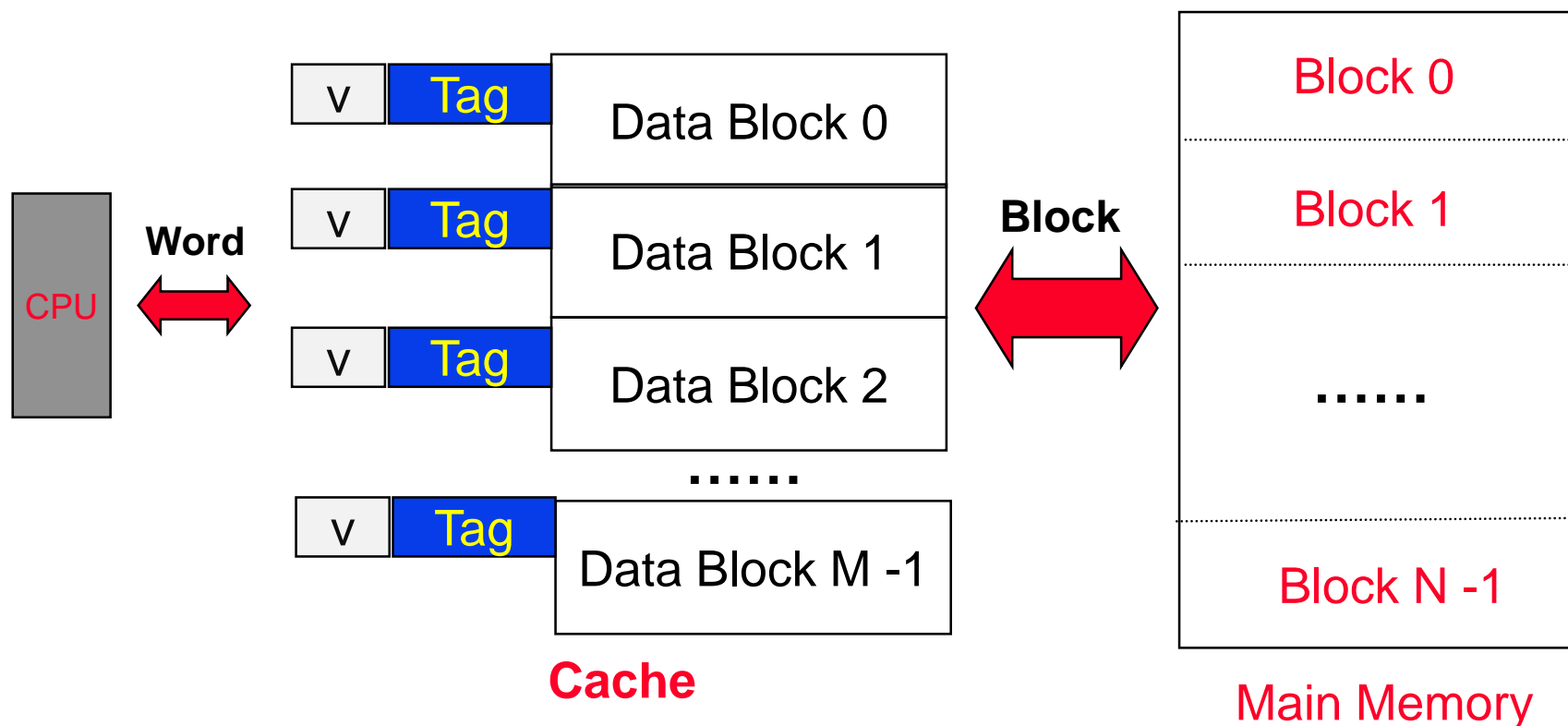
### 四. Cache性能分析



## 2.1 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



## 2.1 Cache与主存之间的映射

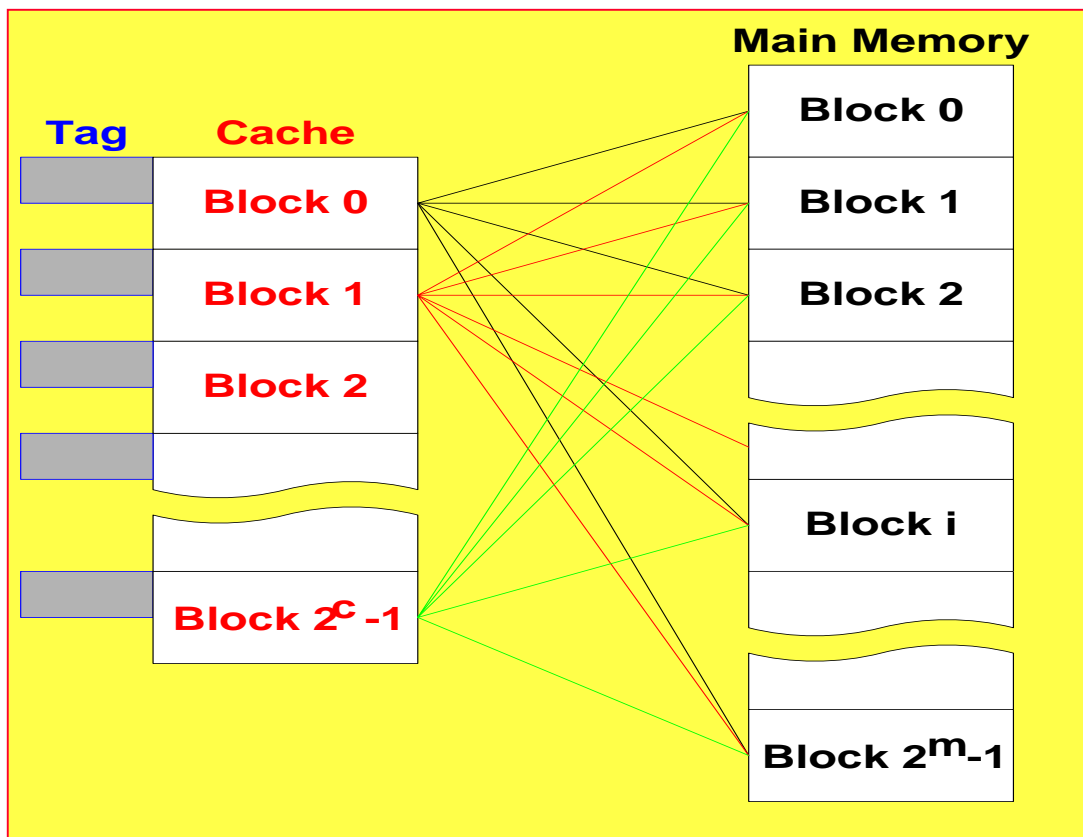
### ❖ 如何进行映射？

- 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**Cache块（Block）**
- **Cache块大小与主存块大小相等**
- 主存块和**Cache块**的映射方式有：
  - 全相联（**Full Associate**）：每个主存块映射到**Cache**的任意块中
  - 直接映射（**Direct**）：每个主存块映射到**Cache**的固定块中
  - 组相联（**Set Associate**）：每个主存块映射到**Cache**的固定组中的任意块中

## 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Block



## 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:** 与该**Cache**块对应的主存块的块地址

### ❖ 全相联Cache举例

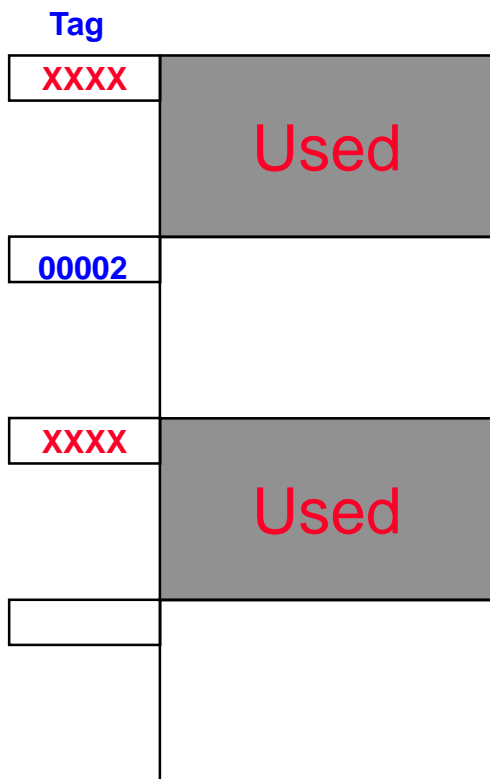
主存容量**16M Bytes**, **Cache**容量**64K Bytes**, 块大小**16字节**。

- 主存和**Cache**各分多少块?
- 主存地址的位数是多少? 格式是什么?
- **Tag**需要多少位?

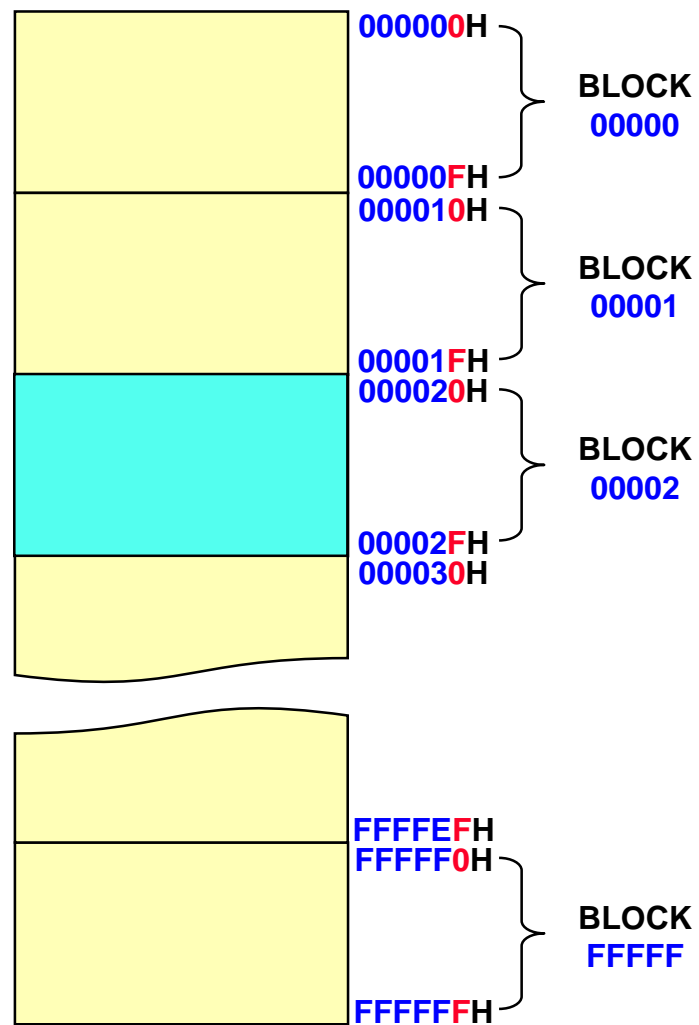
解:

- 主存块数:  $16\text{M} \div 16 = 2^{20} \text{ Blocks}$
- 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:  $64\text{K} \div 16 = 2^{12} \text{ Blocks}$
- **Cache**的**Tag**应该为 **20** 位, 用于存放对应主存块的块地址

## 2.1 Cache与主存之间的映射—全相联



Cache

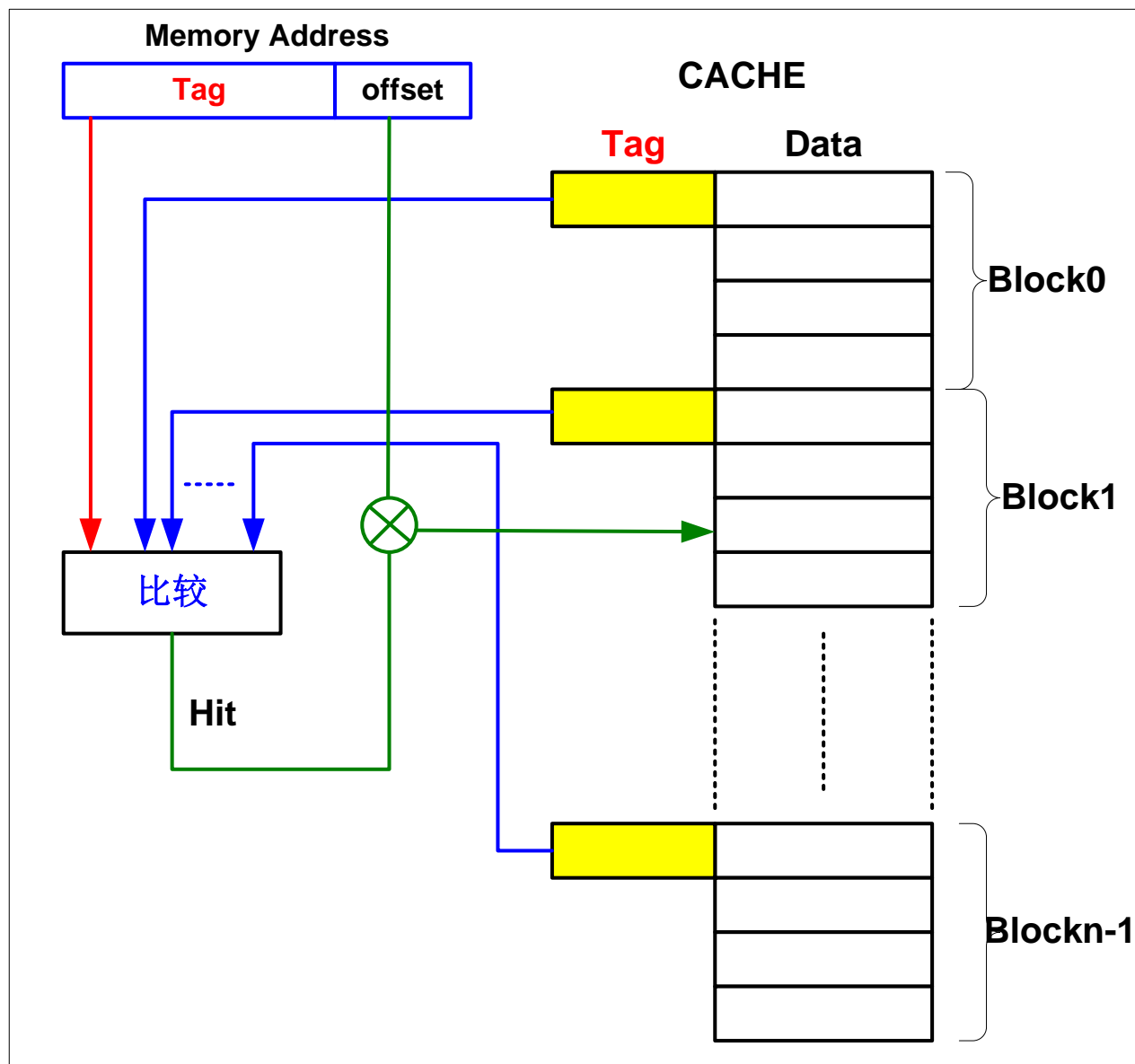


内存

## 2.1 Cache与主存之间的映射—全相联

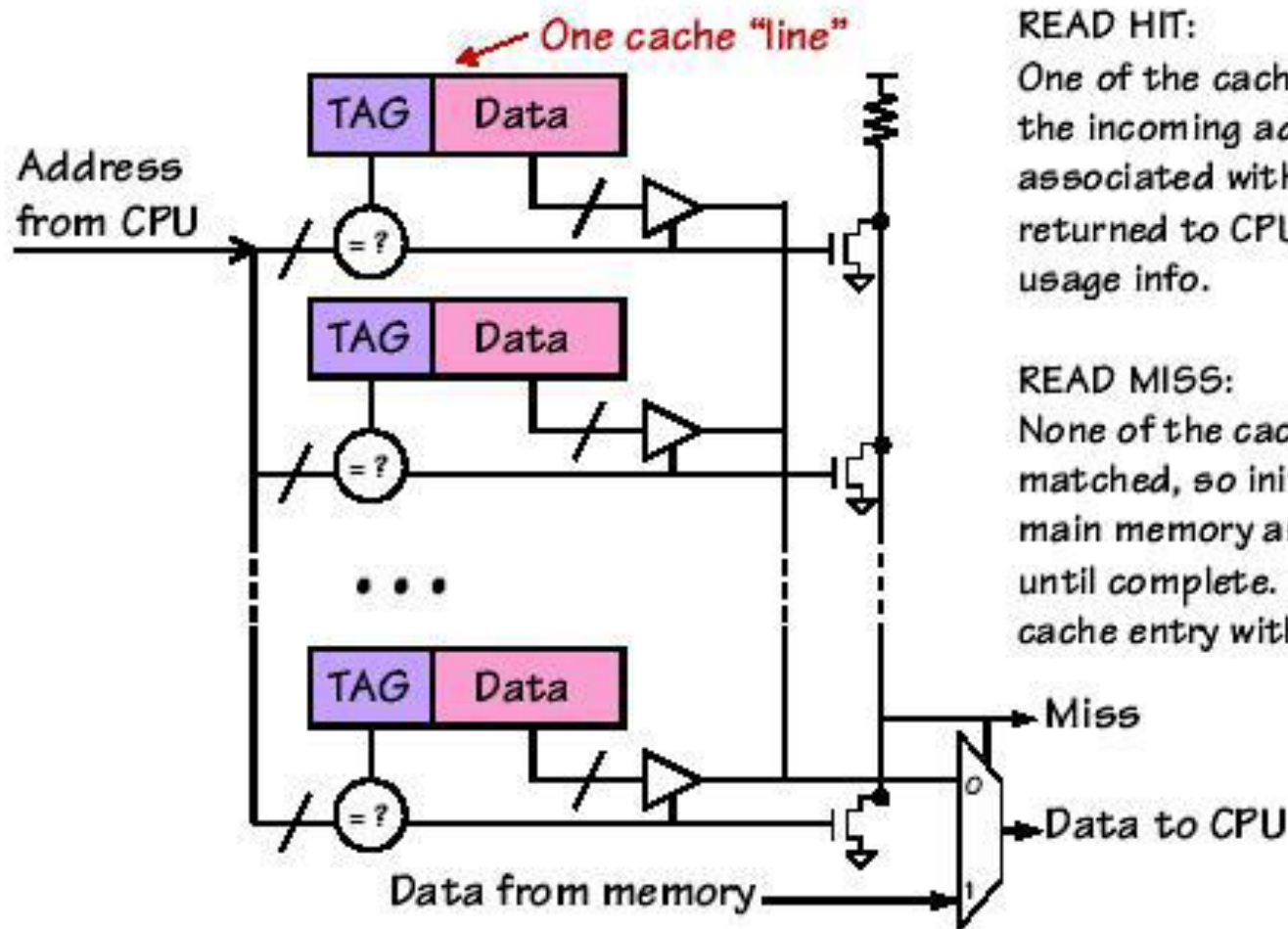
### 全相联Cache组织

- 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

### Fully Associative Cache



READ HIT:

One of the cache tags matches the incoming address; the data associated with that tag is returned to CPU. Update usage info.

READ MISS:

None of the cache tags matched, so initiate access to main memory and stall CPU until complete. Update LRU cache entry with address/data.

## 2.1 Cache与主存之间的映射—全相联

### 全相联映射

#### ❖ 优点:

对Cache的使用可以有最大的灵活性:

- 如Cache空闲, 能确保新块直接写入
- 如Cache已满, 也可方便地选择一个Cache块来替换

#### ❖ 缺点:

在执行Cache读写操作时, 主存地址中的块地址要与Cache中所有Tag都比较后, 才能知晓是否命中

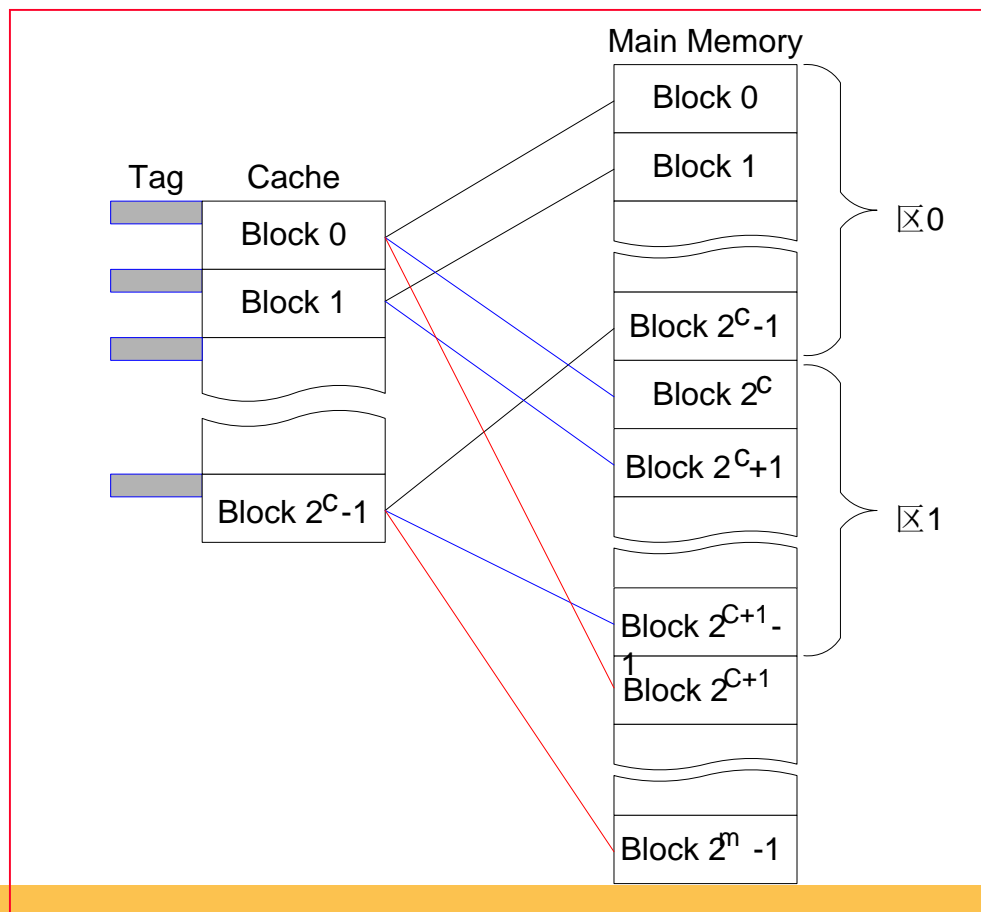
由于实现这一比较操作的电路过多过于复杂, 实现成本太高而难以实用, 因此仅在Cache容量很小时采用



## 2.2 Cache与主存之间的映射—直接映射

### ❖ 直接映射 (Direct Mapping)

- 主存中的某一块  $J$  映射到Cache中的固定块  $K$ ,  $K = J \bmod M$ , 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。



## 2.2 Cache与主存之间的映射—直接映射

### ❖ 直接映射

- 主存的地址格式:
- |           |              |        |
|-----------|--------------|--------|
| 区地址 (Tag) | 区内块地址(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。

### ❖ 直接映射Cache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?
- 主存地址有多少位? 格式是什么?
- Tag应该多少位?

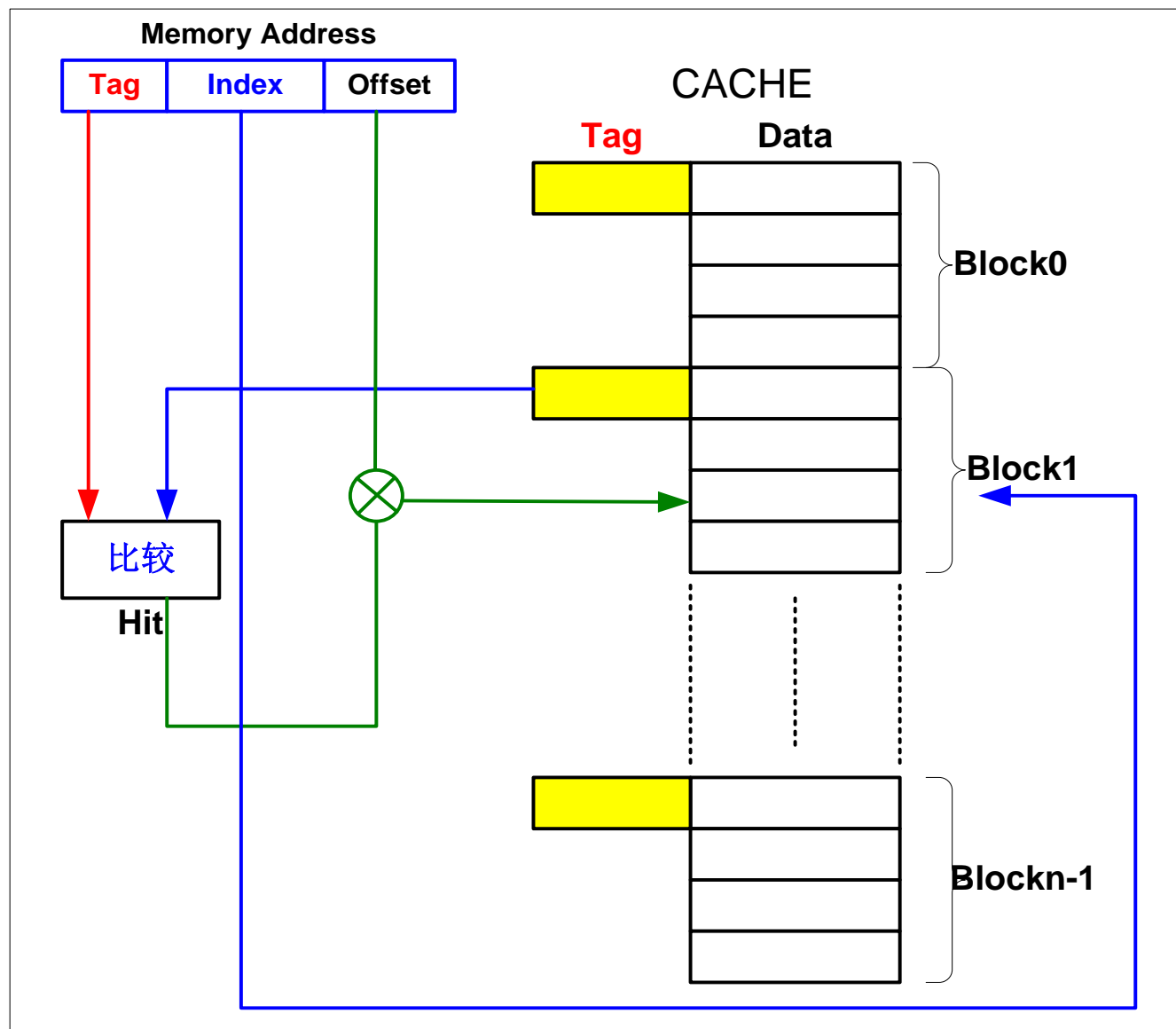
解:

- Cache:  $64K \div 16 = 2^{12}$  Blocks
- 主存块数:  $16M \div 16 = 2^{20}$  Blocks
- 主存分区:  $2^{20} \div 2^{12} = 2^8$ , 可分成 $2^8$ 个区, 每个区 $2^{12}$ 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, 用于存放对应主存块的区地址

## 2.2 Cache与主存之间的映射—直接映射

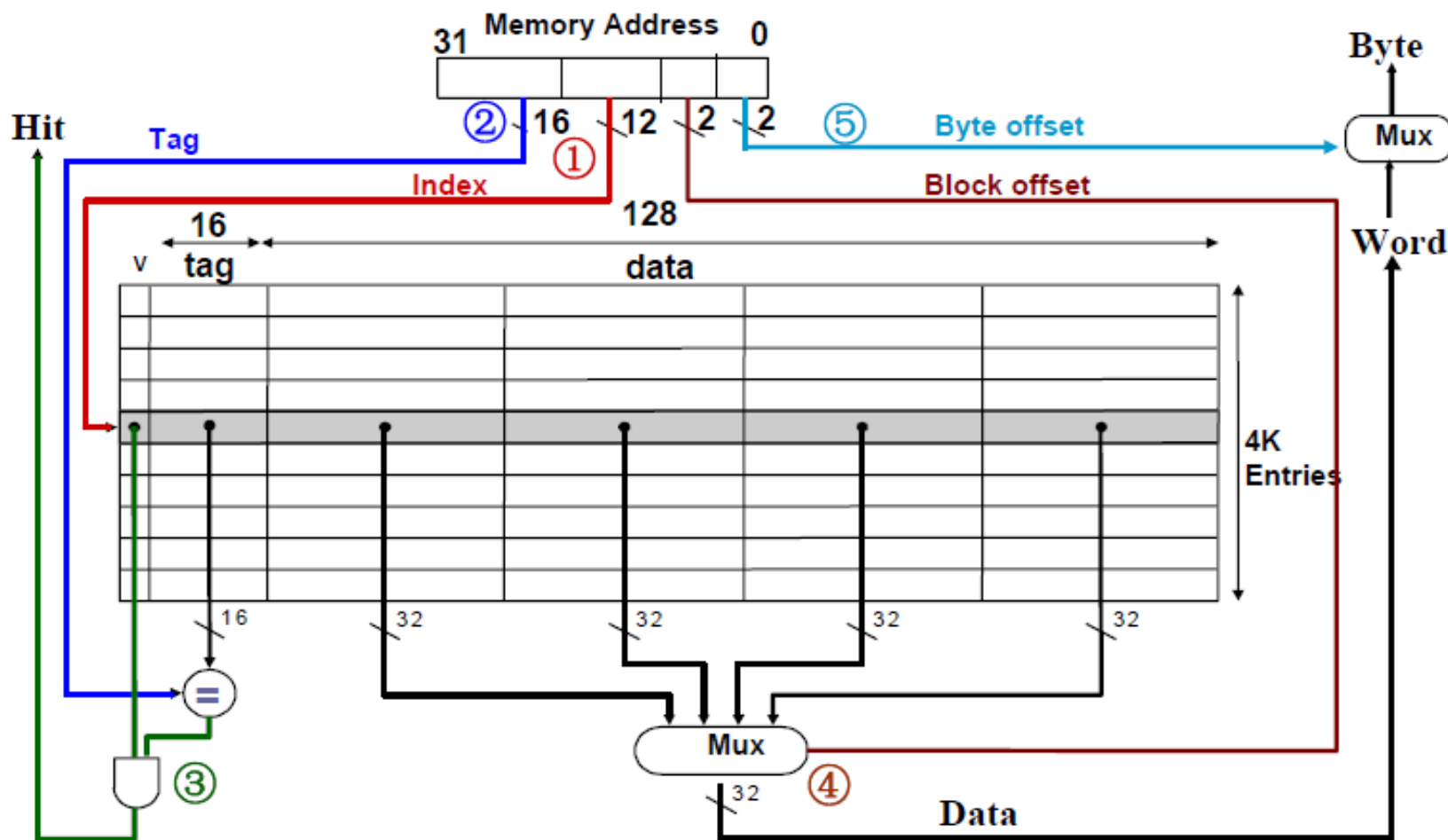
直接映射Cache组织

- 只需与一个Tag进行比较。



## 2.2 Cache与主存之间的映射—直接映射

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



## 2.2 Cache与主存之间的映射—直接映射

### 直接映射

#### ❖ 优点:

实现简单，只需利用主存地址中的区地址，与块地址对应的Cache块中Tag 进行1次比较，即可确定是否命中

#### ❖ 缺点:

映射关系不灵活，因每个主存块只能固定地对应某个确定的Cache块，会出现Cache有很多空闲，但新块不能直接写入而需要替换的现象，Cache空间的利用不充分

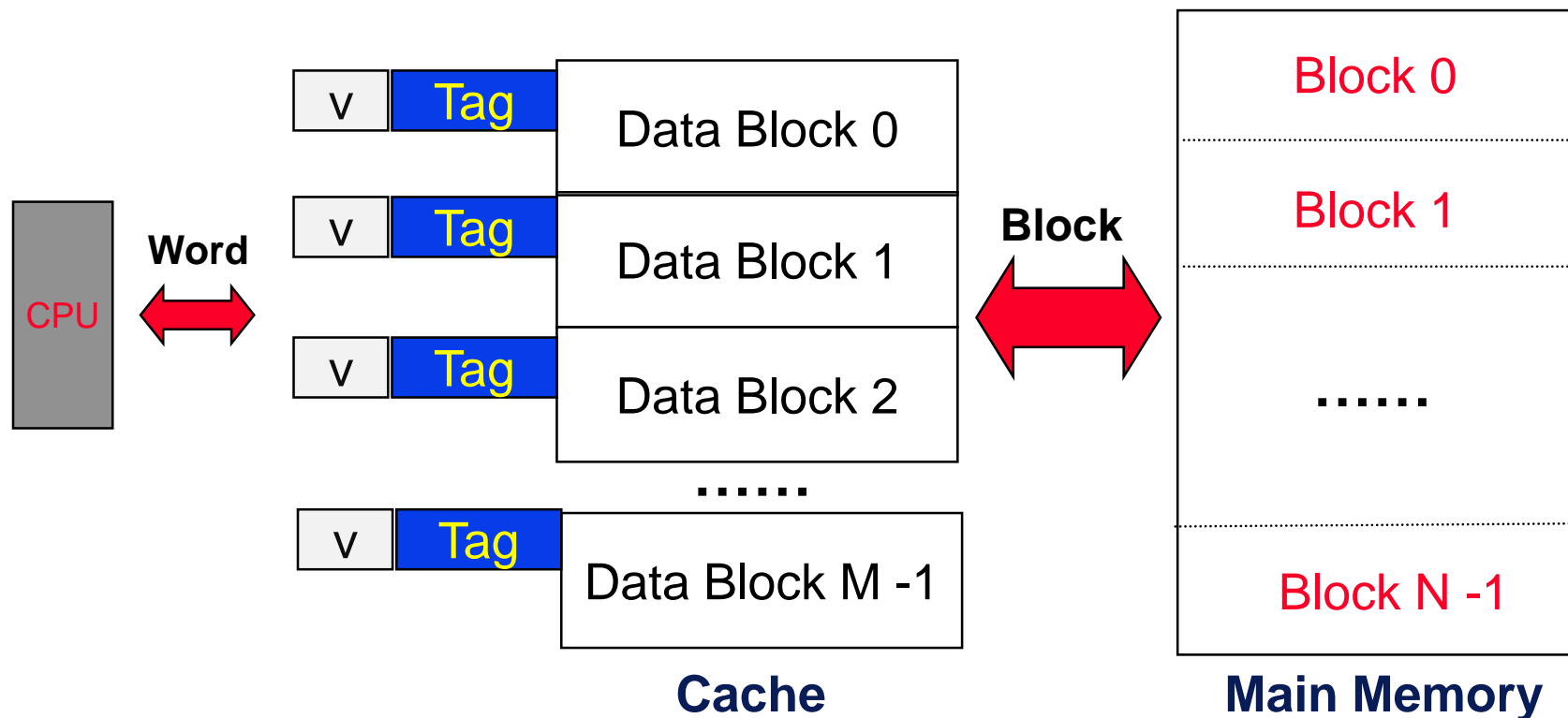
# 第十八讲

---

## 上一讲简要回顾

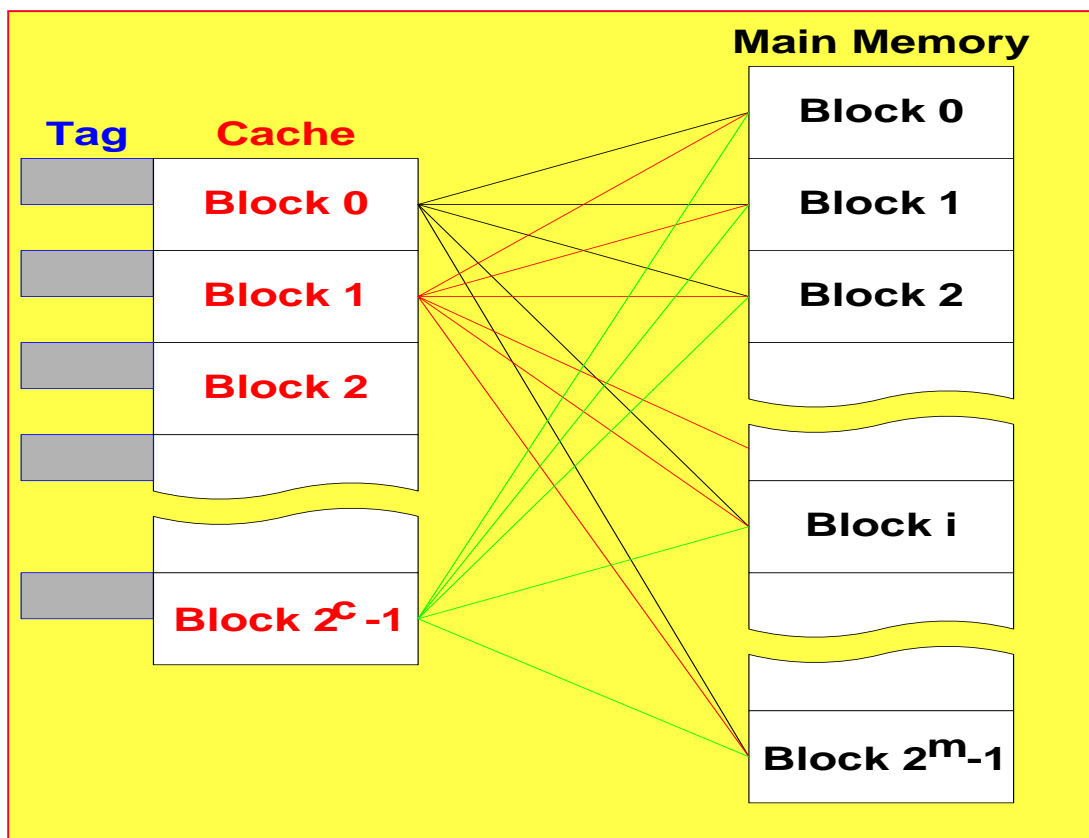
### ❖ 高速缓冲存储器Cache

- Cache产生的原因和前提（存储墙、局部性、SRAM）
- Cache需解决问题、结构、工作过程（存储、地址、替换机构）
- Cache相关的一些术语和指标（块、valid、tag、行、组/路）
- 主存块与Cache块的映射方式（全相联、直接映射）



## 上一讲简要回顾 —— 全相联

❖ 全相联 (Full Associative) : 每个主存块可映射到Cache的任意块中



❖ 优点：使用灵活：

- 如Cache空闲，能确保新块直接写入
- 如Cache已满，也可方便选择一个块来替换

❖ 缺点：主存地址中的块地址要与Cache中所有Tag都比较，才能知晓是否不命中。实现电路多，成本高，仅用于Cache容量很小时

❖ 主存的地址格式：

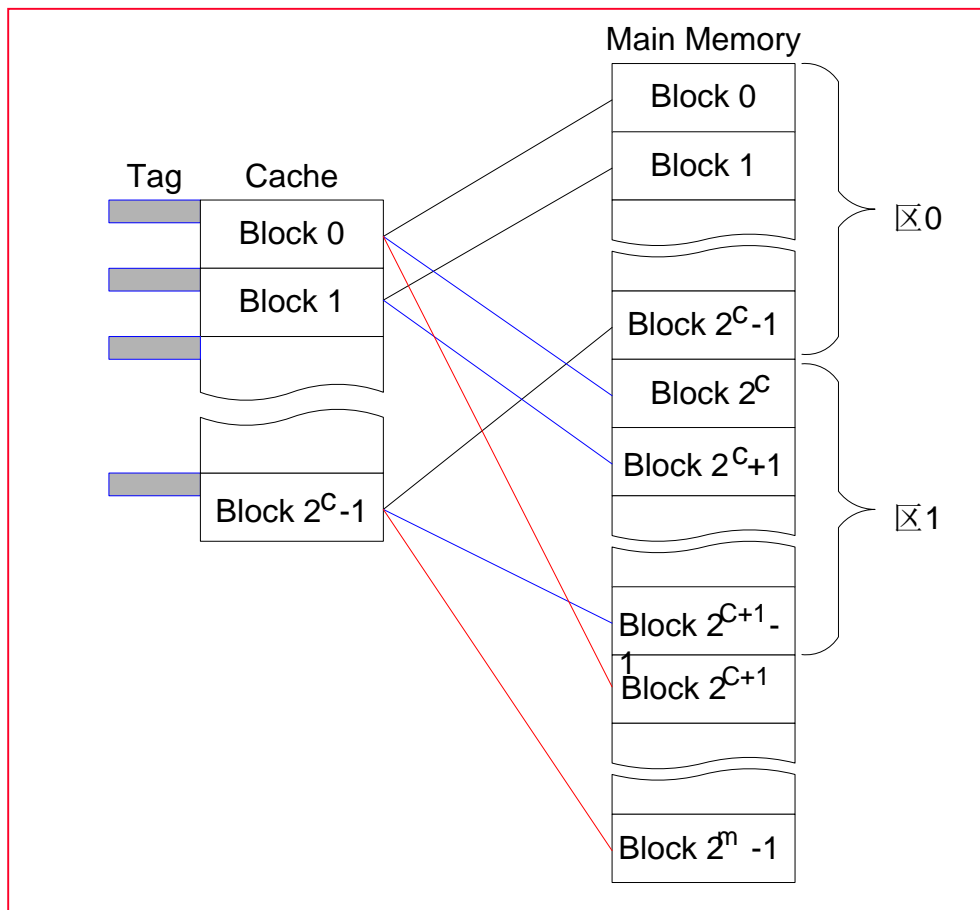
Block Number (Tag)

Offset



## 上一讲简要回顾 —— 直接映射

❖ 直接映射（Direct Mapping）：每个主存块映射到Cache的固定块中



❖ 优点：实现简单，只需利用主存地址中的区地址，与块地址对应的Cache块中Tag 进行1次比较，即可确定是否命中

❖ 缺点：映射关系不灵活，会出现Cache有很多空闲，但新块不能直接写入而需替换的现象，Cache空间利用不充分

❖ 主存的地址格式：

区地址 (Tag)

区内块地址(Index)

Offset

## 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. 直接映射
3. 组相联映射

### 三. Cache的替换策略

### 四. Cache性能分析

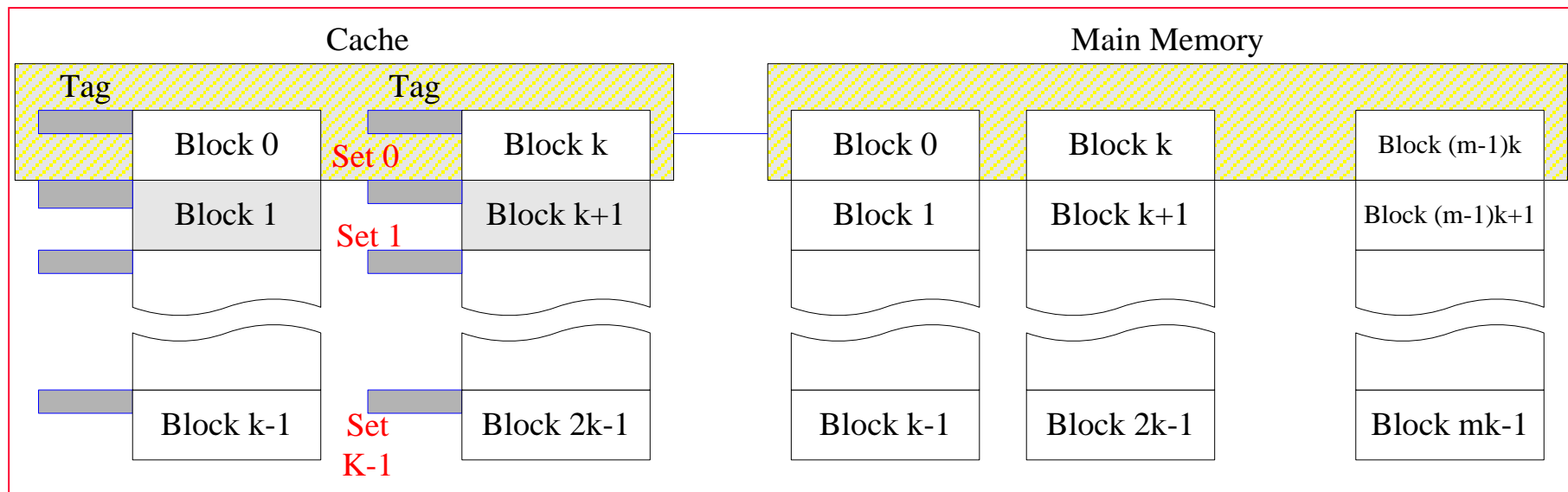
## 2.3 Cache与主存之间的映射—组相联

### ❖ 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。

$$I = J \bmod K$$

- 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。



## 2.3 Cache与主存之间的映射—组相联

### ❖ 组相联映射

➤ 主存的地址格式：

组内块地址(Tag)	组地址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。

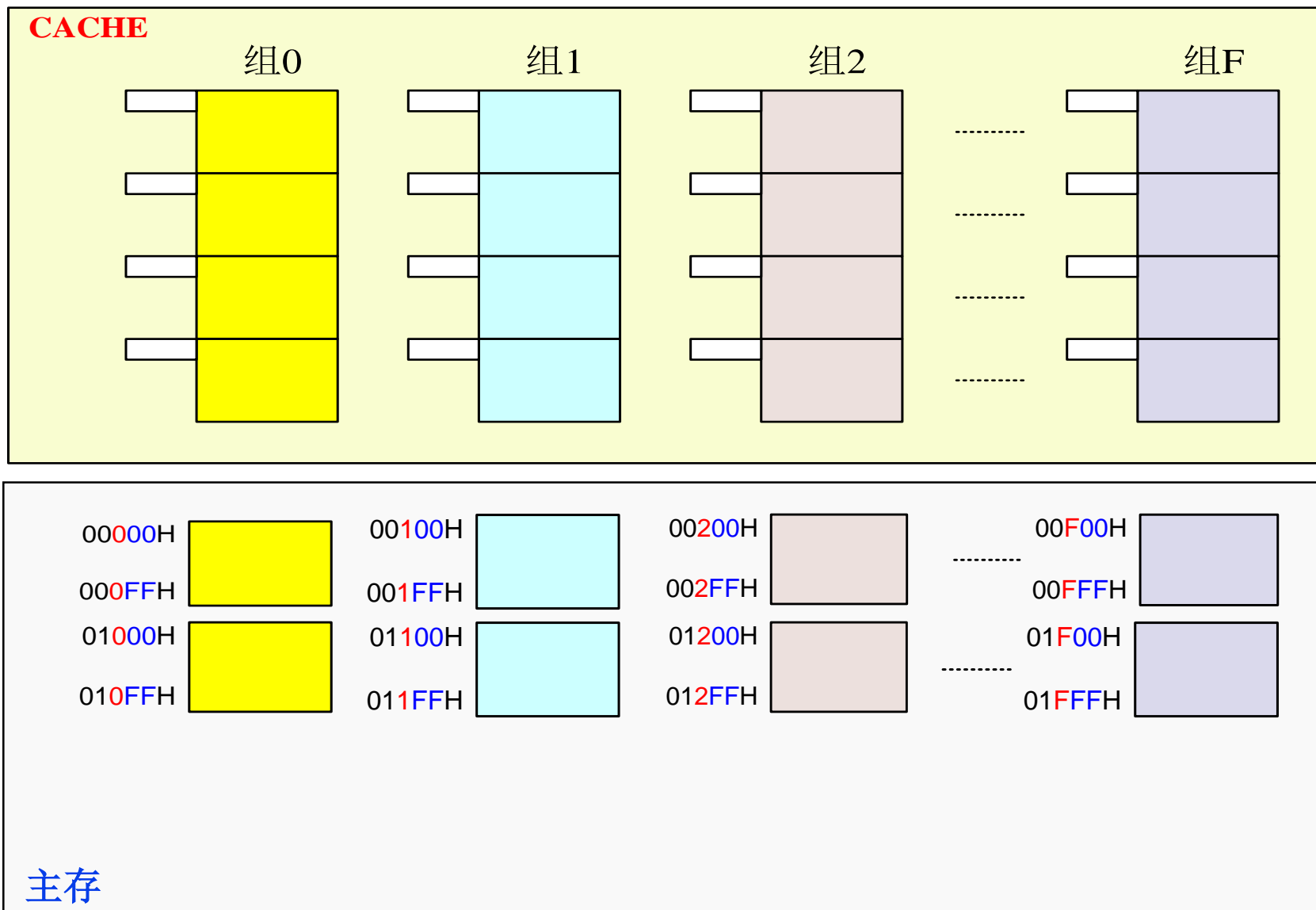
### ❖ 举例

- 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？
- 主存分多少组？每组包含多少块？
- Cache的Tag需要多少位？存放什么内容？

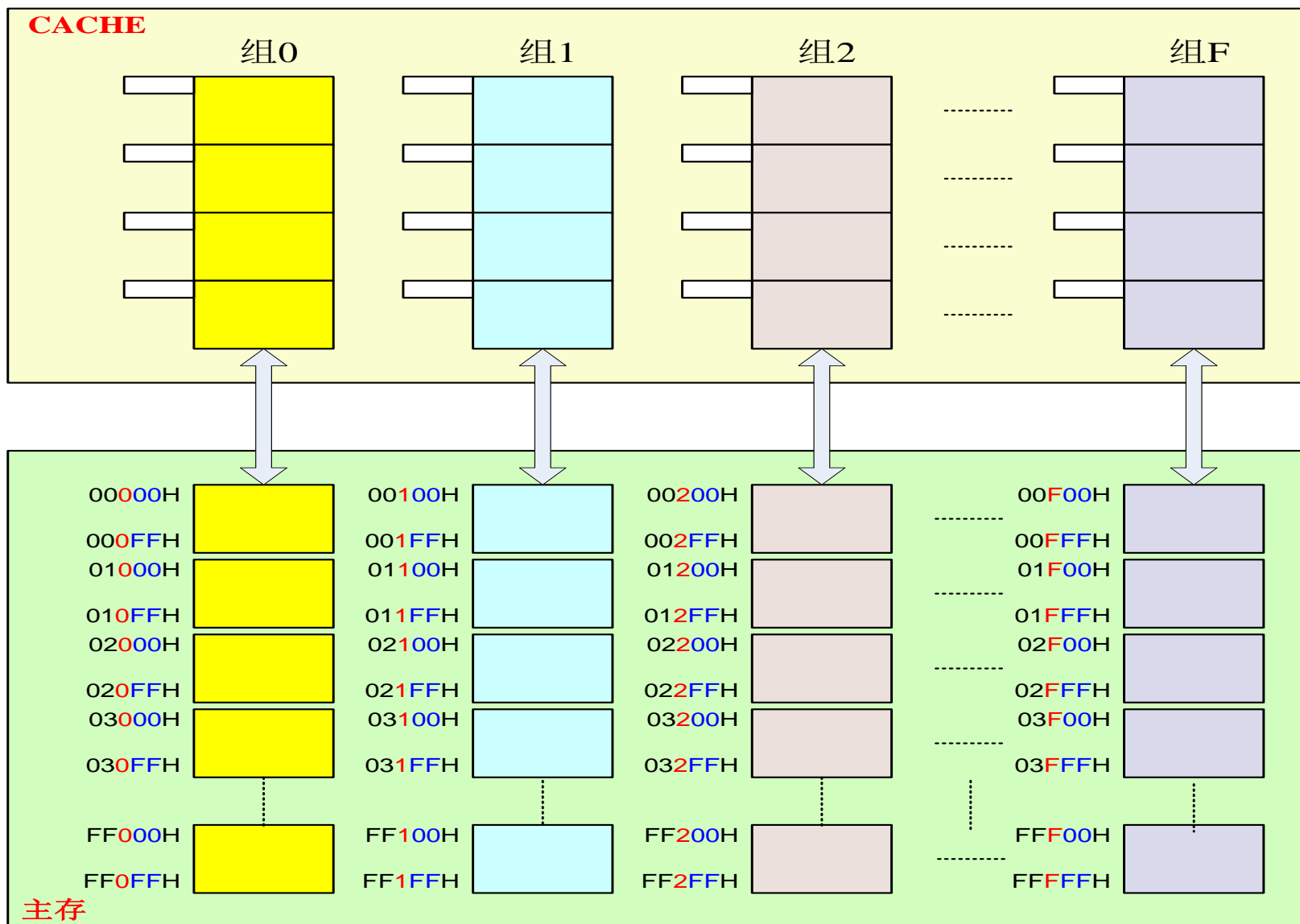
解：

- 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。存放Cache块对应主存块的组内块地址

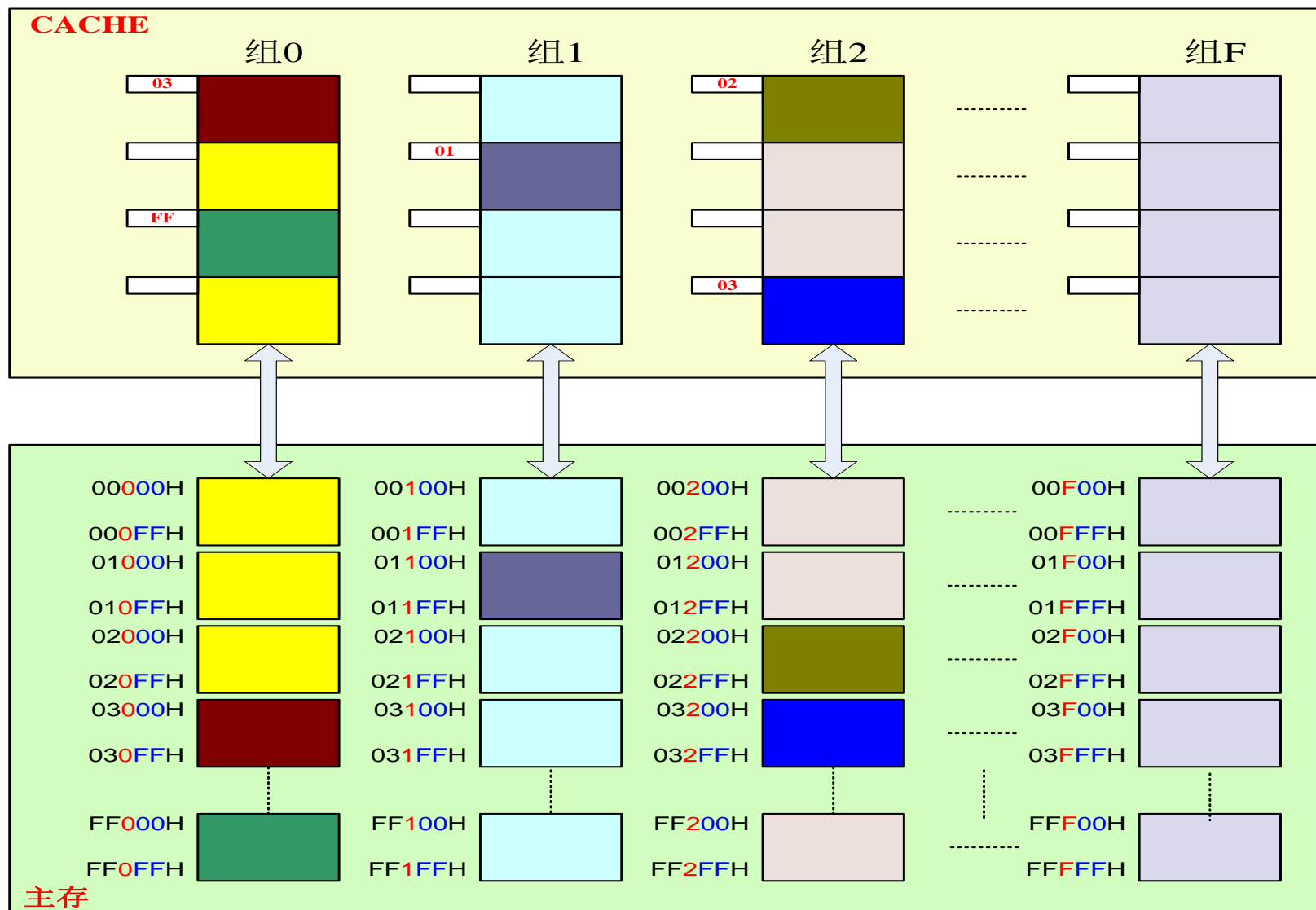
## 2.3 Cache与主存之间的映射—组相联



## 2.3 Cache与主存之间的映射—组相联



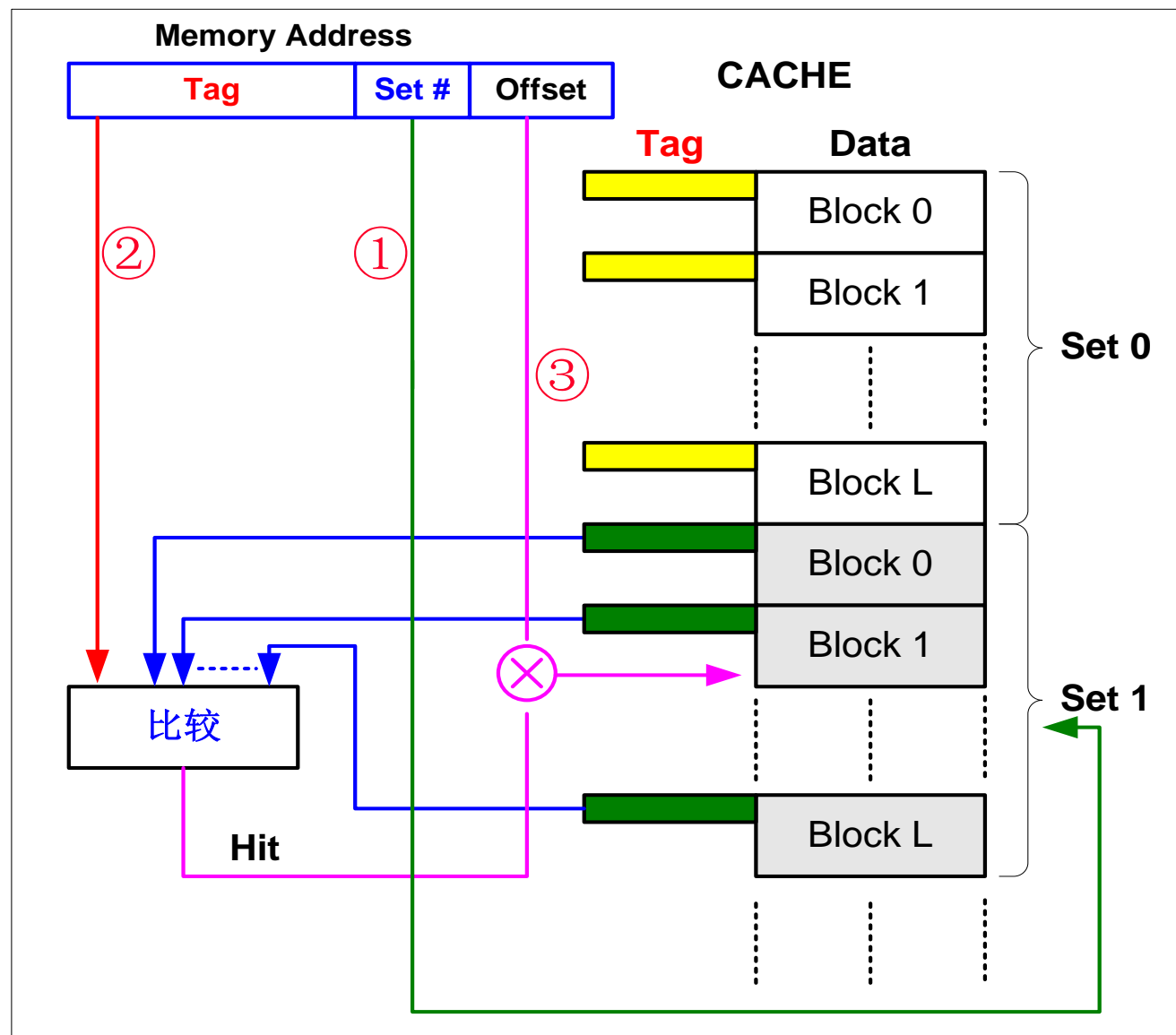
## 2.3 Cache与主存之间的映射—组相联



## 2.3 Cache与主存之间的映射—组相联

组相联Cache组织

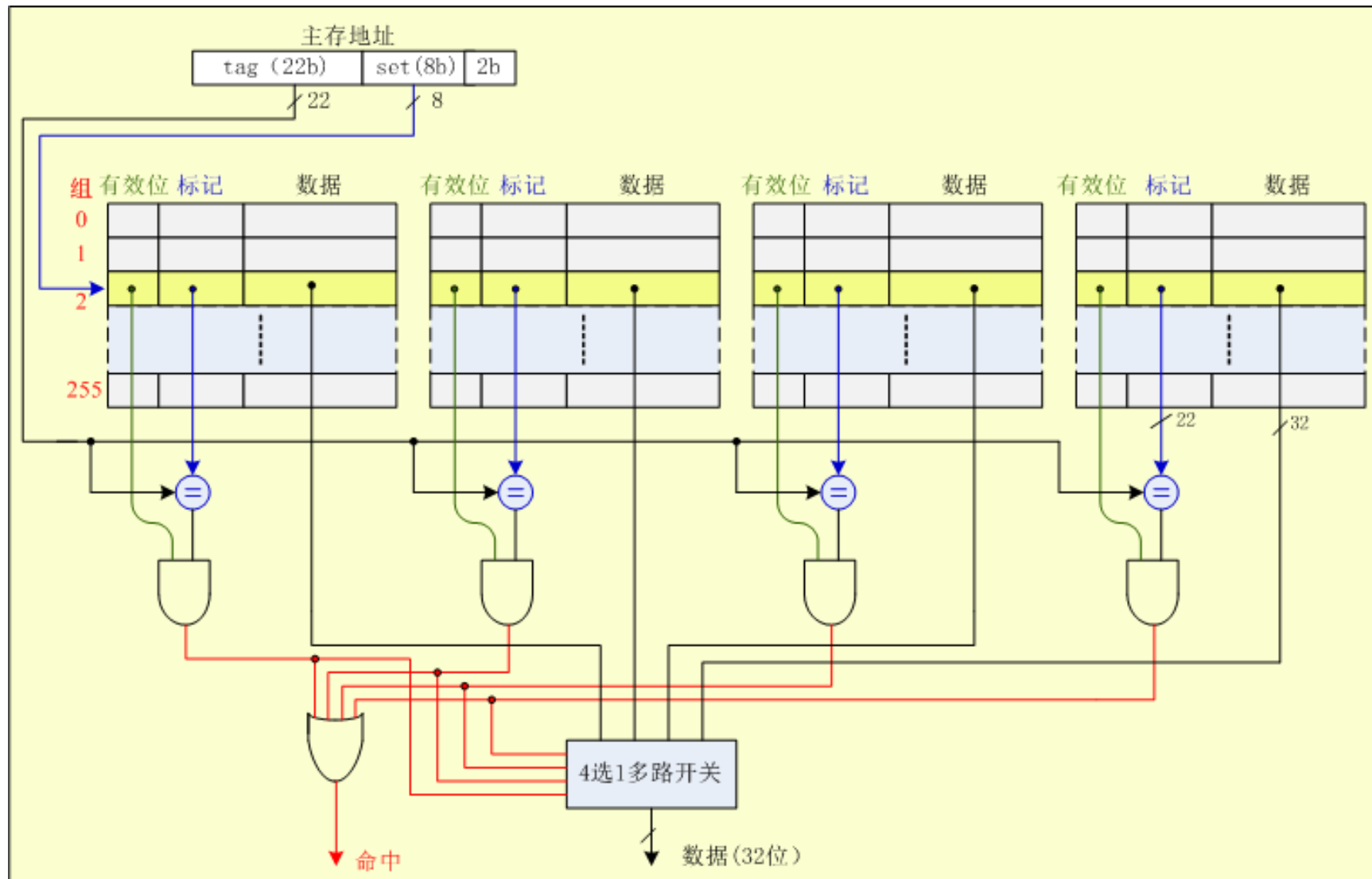
- 同时与一个组内的所有Tag进行比较，N路组相联Cache则需N个比较器；
- 数据访问与比较并行执行。





## 2.3 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 — 组相联

### 组相联映射

- ❖ 组相联映射是直接映射和全相联映射的折衷
- ❖ 主存块与Cache组之间是一一对应关系，好似直接映射方式，简化了实现
- ❖ 主存块与对应组中的Cache块是任意对应关系，好似全相联映射方式，体现了一定的灵活性
- ❖ 组相联映射兼顾了实现成本和灵活性

## 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

## ■ 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

$$1 + 4 \times 15 + 4 \times 1 = 65T$$

### ■ 2字宽的缺失损失

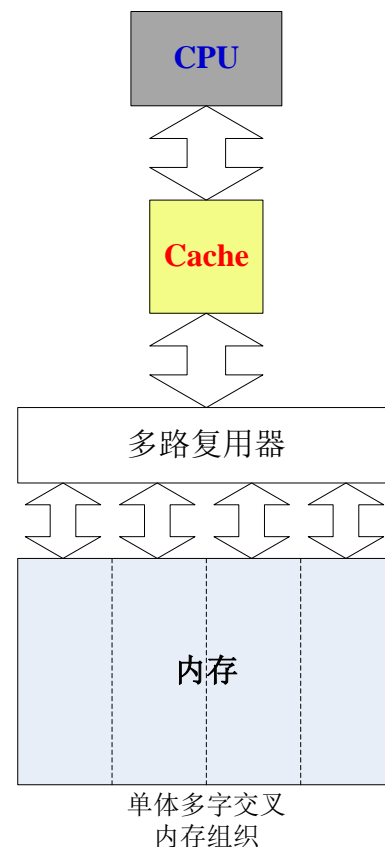
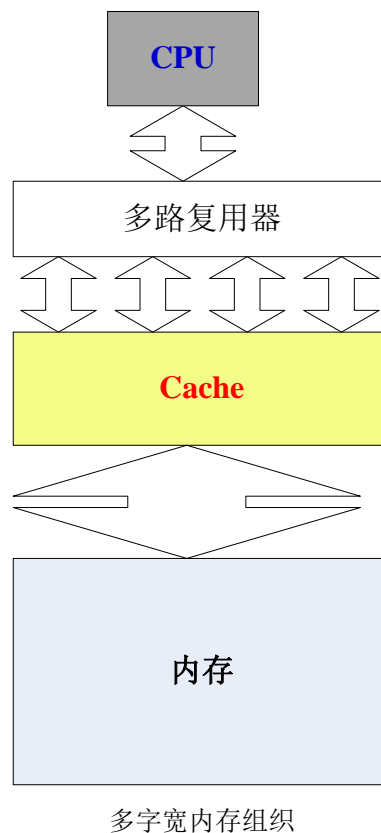
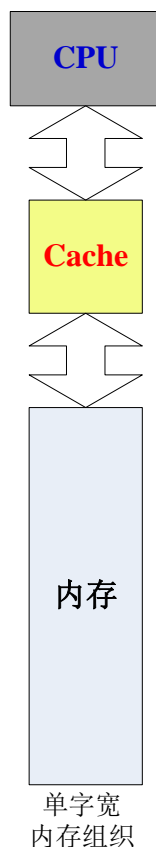
$$1 + 2 \times 15 + 2 \times 1 = 33T$$

### ■ 4字宽的缺失损失

$$1 + 1 \times 15 + 1 \times 1 = 17T$$

### ■ 4字交叉的缺失损失

$$1 + 1 \times 15 + 4 \times 1 = 20T$$



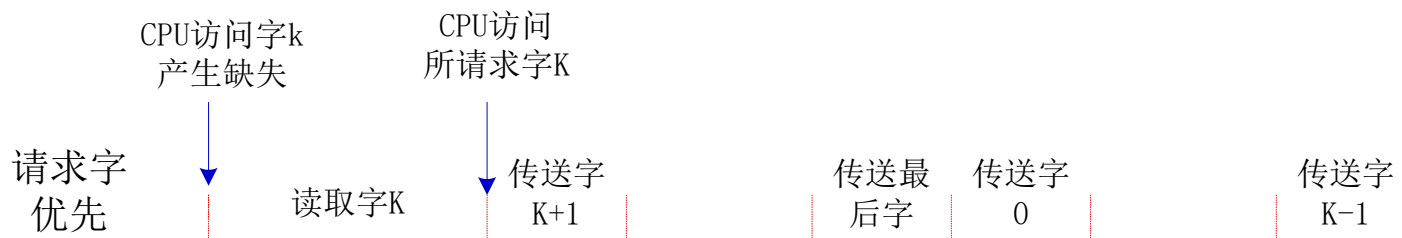
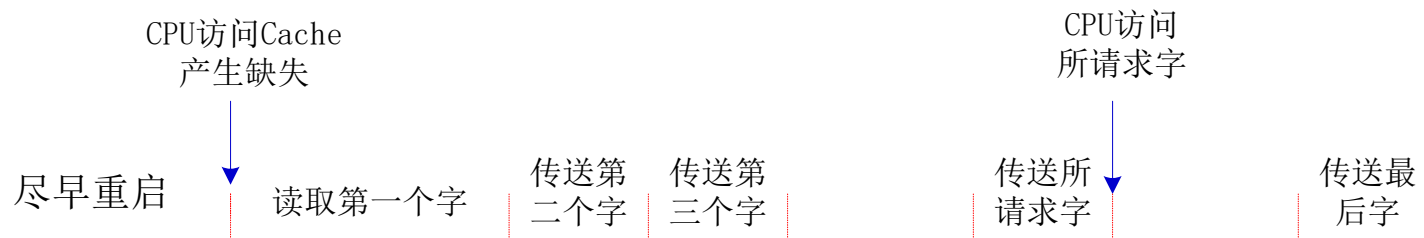
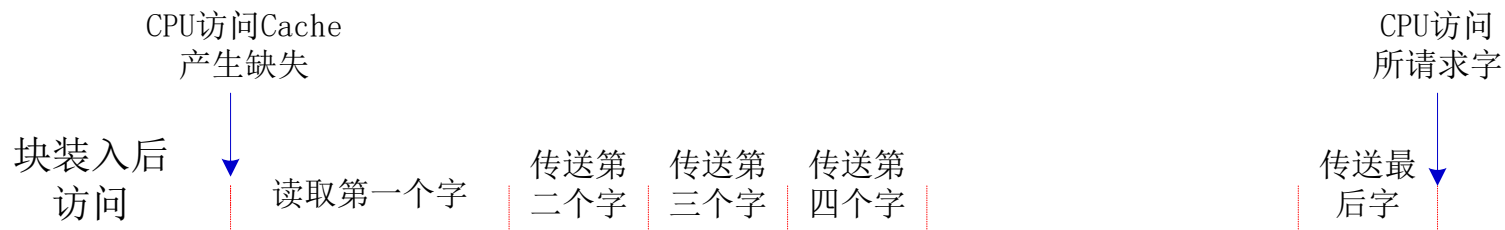
## Cache的缺失处理

### ■ 缺失处理（以读操作为例，写操作比较复杂）

- 块装入后访问：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（也即引起缺失的字）
- 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



## ■ 替换块的选择范围

- 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，可在Cache中任选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

## ■ 替换算法的实现

- 一般由硬件电路实现，因此，算法应以简单、便于硬件实现为宜

# Cache的替换策略

## ■ LRU的实现（计数器法）

➤ 原则：将近期最少使用的块替换出去

➤ 方法：

- Cache的每一块都设置一个计数器；
  - 被调入或被替换的块，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：
    - （1）如果计数值小于命中块的计数值，则该块的计数值加 1；
    - （2）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来替换。

➤ 优点：

- 符合Cache基本原理，并考虑了新进入Cache的块，命中率较高

# Cache的替换策略

## ■ FIFO的实现（计数器法）

➤原则：总是将最先调入Cache的块替换出去

➤方法：

- 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

➤优缺点：

- 实现容易，开销小
- 可能会把一些较早进入Cache、但经常使用的块（如循环程序）替换出去

## 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。

## Cache举例

(1) Cache分组数:  $16\text{KB} \div 16 \div 4 = 256$  组

(2) 内存每组块数:  $16\text{MB} \div 16 \div 256 = 4096$  块

内存地址格式: 组内块地址(12b) + 组地址(8b) + 块内偏移地址(4b)

(3) CPU开始读内存单元 (都是第08组内存块):

- 读**430082H**单元, 命中, 数据在Cache第2块中
- 读**2F8086H**单元, 缺失, 此时, 读内存块, 并装入到Cache第3块, 改写其装入位为1, Tag为**2F8**
- 读**03008AH**单元, 命中, 数据在Cache第4块中
- 读**F40088H**单元, 命中, 数据在Cache第1块中
- 读**063081H**单元, 缺失, 此时, 读内存块, 并装入到Cache中, 但该组Cache已满, 需要选择最近最少使用的一块替换出去, 应选第2块中; 改写其装入位为1, Tag为**063**。

装入位	Tag
1	F40H
1	430H
0	218H
1	030H

装入位	Tag
1	F40H
1	430H
1	2F8H
1	030H

装入位	Tag
1	F40H
1	063H
1	2F8H
1	030H

## 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

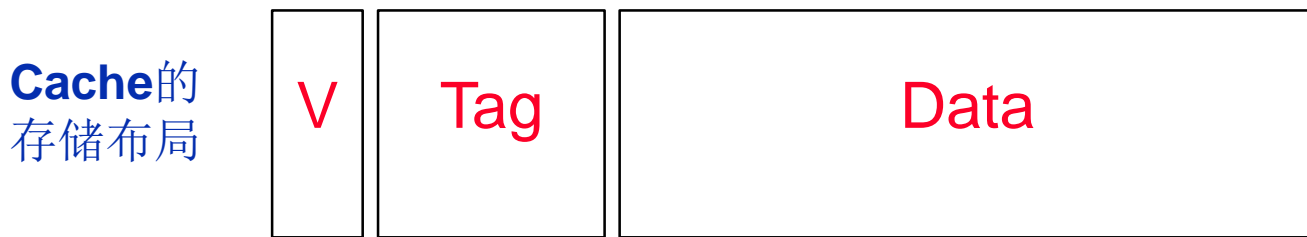
### 三. Cache的替换策略

### 四. Cache性能分析

# Cache的容量

## ■ Cache的容量

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



- 例：假设一个直接映射像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： $4 \times 32 = 128 \text{ bits} = 2^4 \text{ Bytes}$ ；（块内地址）
- Cache块数： $16\text{K} \div 2^4 = 2^{10}$  块；（区内块地址）
- tag位数： $32 - 10 - 4 = 18 \text{ bits}$ ；（区地址）
- 有效位：1位
- Cache实际总容量： $2^{10} \times (128 + 18 + 1) = 147\text{Kb} \approx 18.4\text{KB}$

## Cache的容量

- 例：假设有一个4 路组相联Cache，数据存储空间大小64KB，块大小为16字节，主存地址32位，主存一个字含4个字节，每个数据块对应1位有效位，此外，Cache采用写回策略，Cache每个字用1位脏位来表示是否被修改。
- 计算实现该Cache所需总存储容量？
- 解答
  - Cache每数据块大小： $16 \times 8 = 128 \text{ bits} = 2^4 \text{ Bytes}$ （块内地址）
  - Cache块数： $64\text{KB} \div 2^4 = 2^{12}$  块
  - Cache组数： $2^{12} \div 4 = 2^{10}$  组（组地址）
  - tag位数： $32 - 10 - 4 = 18 \text{ bits}$ （组内块地址）
  - 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： $2^{12} \times (128 + 18 + 1 + 4) = 618496\text{b} \approx 75.5\text{KB}$



## Cache的性能计算

### ■ 存储访问时间： 对于cache和主存组成的两级存储系统

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

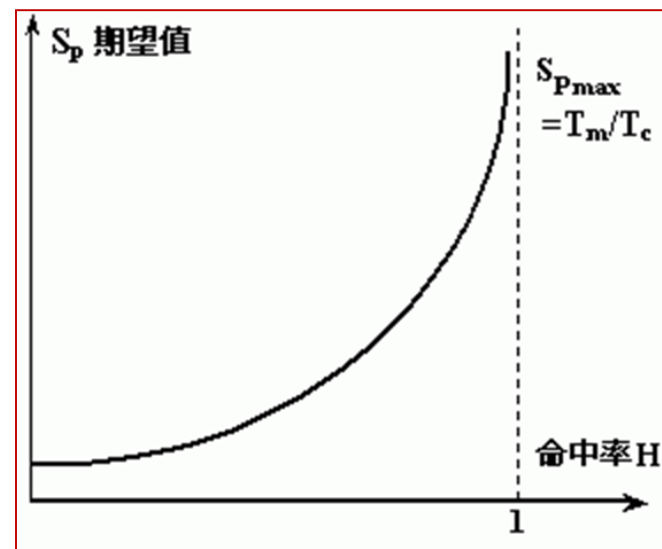
则存储系统的等效访问周期 $T$ 为：

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

### ■ 加速比SP (Speedup)

存储系统的加速比 $S_p$ 为：

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

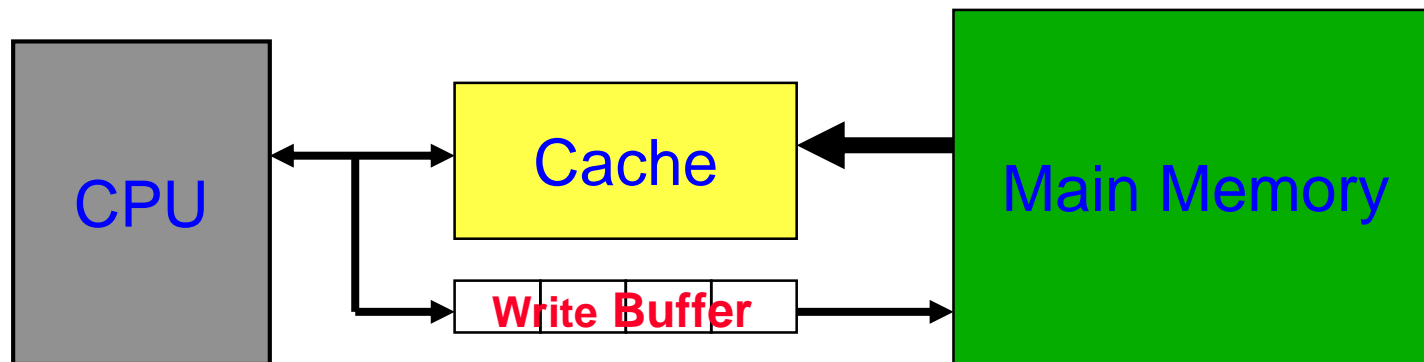


加速比与命中率的关系

# Cache与主存的数据一致性

## ❖ 数据一致性的问题主要由写操作产生

- 写通过（写直达，Write Through）：写Cache的同时写主存，效率较低；
- 写回（Write Back）：写操作只更新Cache中的数据，直到Block替换时才将整个Block写回主存，一般使用“脏位”（dirty bit）来表示Block在替换回主存之前是否被修改过。



Write Through 模式的Cache结构

- 一般Write Buffer 是FIFO

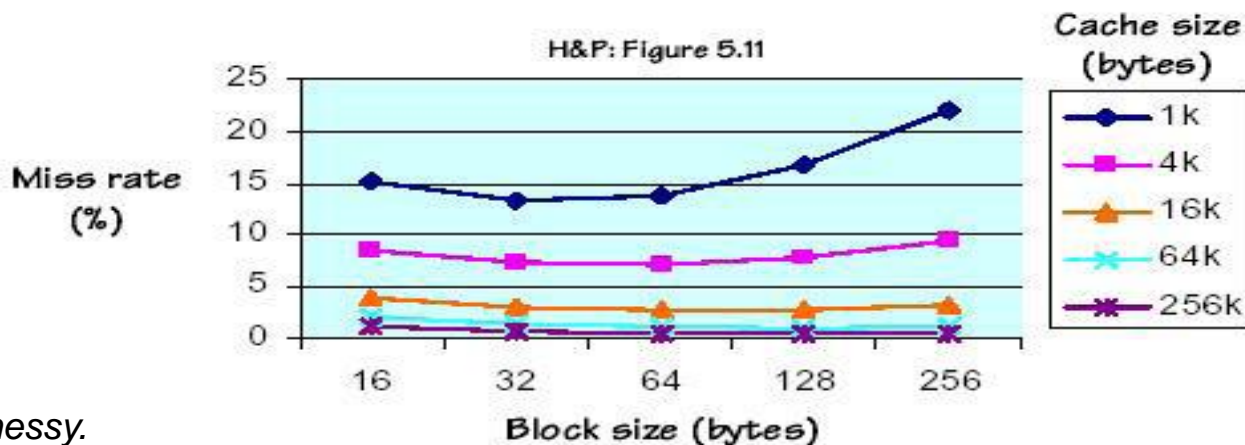
# Cache命中率问题

## ❖ 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Cache总容量限制，块数量会下降，导致块替换增加）；
- 单纯增加块大小带来缺失代价（缺失损失）的增大。

## Block size vs. miss rate

块大小与缺失率的关系



- spatial locality: larger blocks → reduce miss rate
- fixed cache size: larger blocks → fewer lines in cache → higher miss rate, especially in small caches

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

### ❖ PowerPC 620的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。

### ❖ Pentium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。



## Pentium 4的Cache示例

