

Importance of Memory

- 在到目前为止的课程中，我们只研究了计算机系统的处理器部分：
 - CPU performance, ISA, Datapath, Pipelined execution.
- 为什么我们关心内存层次结构？
 - In 1980: no cache in microprocessors.
 - In 1995: 2-level caches, e.g., 60% of transistors on the Alpha processor is for the cache system.
 - *cache 缺失造成 150 clock cycles 延迟!!!*
 - 内存的性能提升跟不上处理器的性能提升.

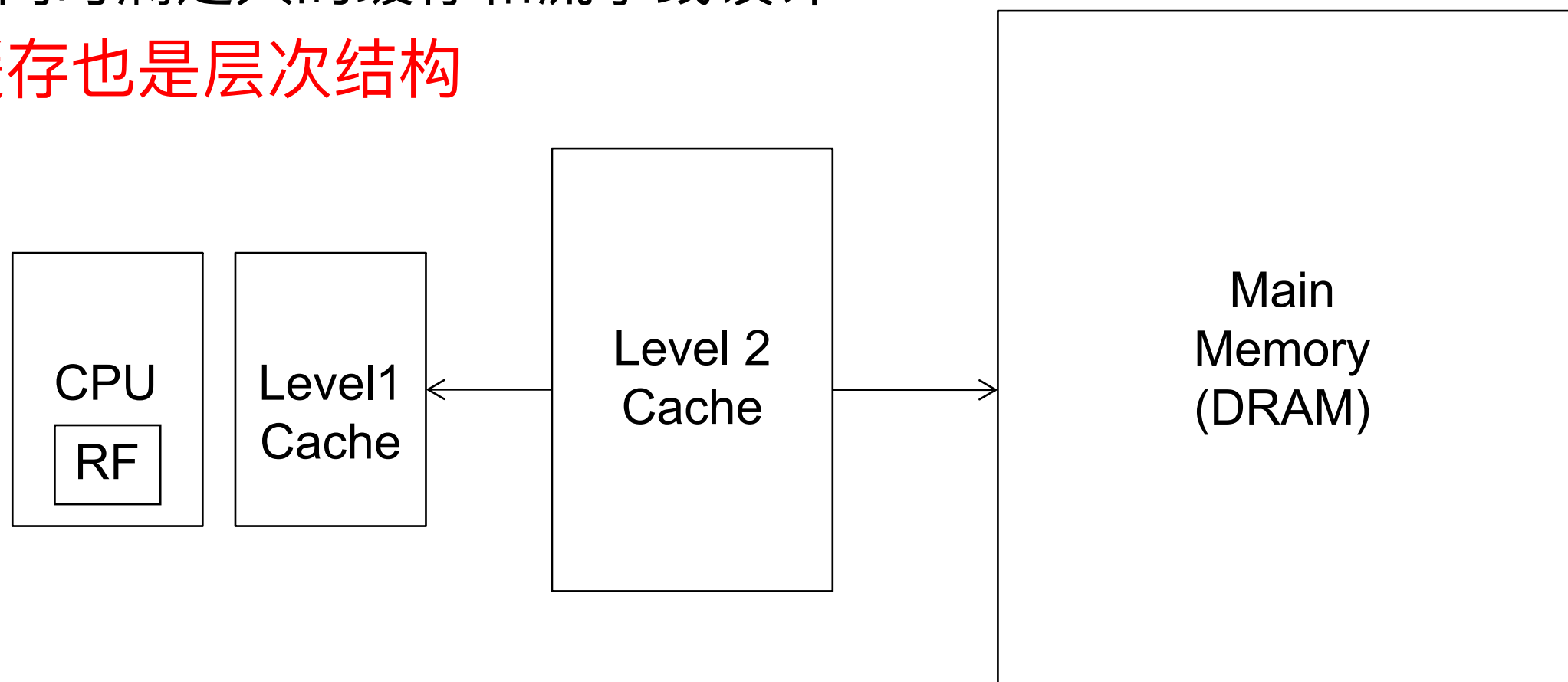
内存性能对计算机的整体性能至关重要.

用书架做类比

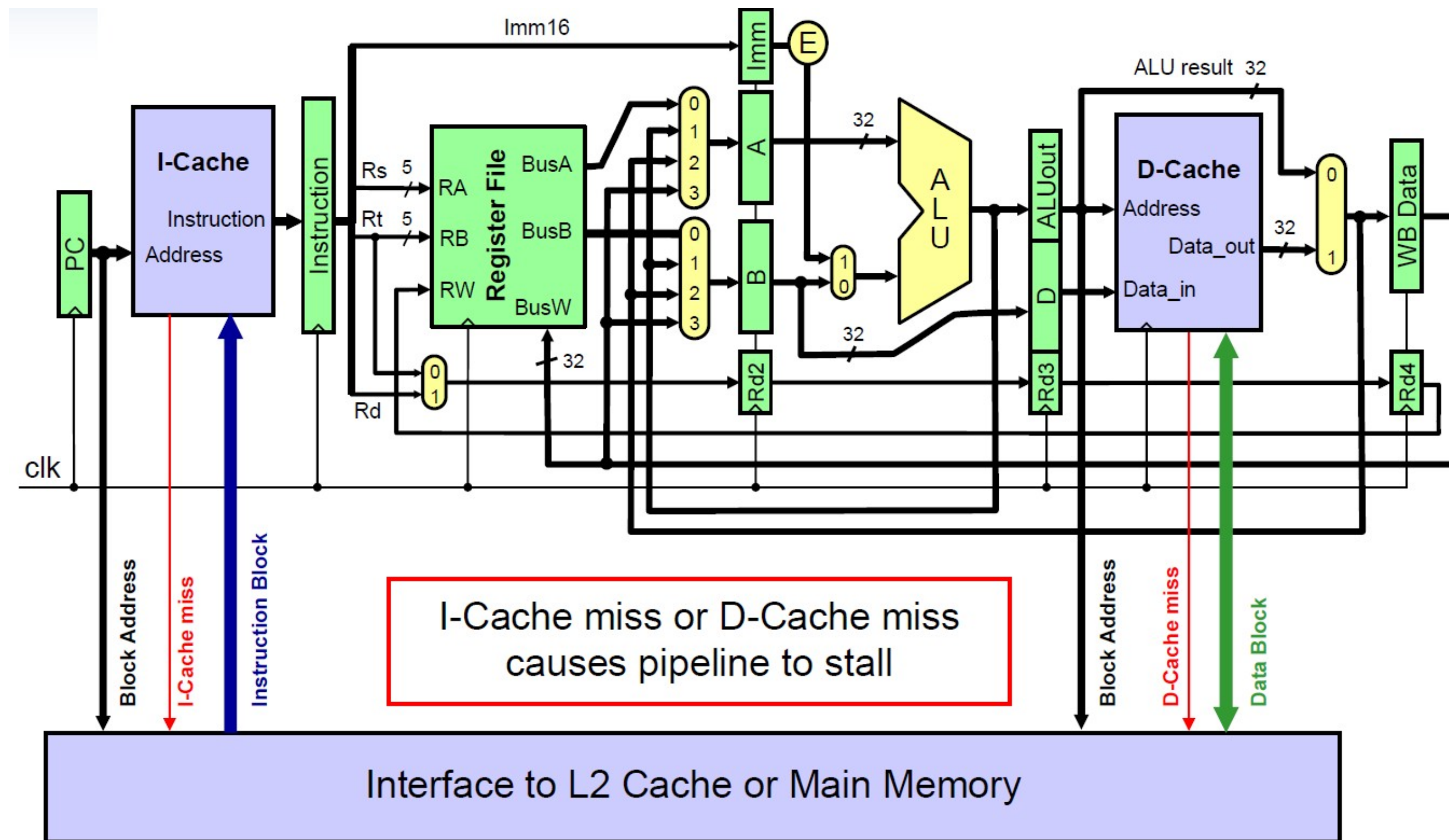
- 在你手中的书-寄存器
 - 桌子上的书-缓存
 - 书架上的书-内存
 - 家中盒子中装的书-闪存
 - 放在储藏室的书-硬盘
-
- 最近使用的书放在桌子上-缓存
 - 计算机组成原理的书，你最近的课程在学习的书
 - 直到桌子放满了
 - 书架上的相邻书籍几乎同时需要
 - 如果书架上的书是按分类整理好的

在流水线中的缓存设计

- 缓存需要紧密地集成到流水线中
 - 理想化, 在1-cycle 内访问, 以使得 load/sw指令的操作不需要阻塞
- 高频率的流水线 → 不能让缓存过大
 - 但, 要同时满足大的缓存和流水线设计
- 想法: 缓存也是层次结构



流水线CPU典型的缓存层次结构



自动：

硬件跨级别管理数据 移动，对程序员透明 ++程序员的生活更轻松

普通程序员不需要了解缓存

你不需要知道缓存有多大，它是如何工作的，就可以编写一个“正确”的程序！

L1: 数据缓存D-cache, 还有一个指令缓存I-cache

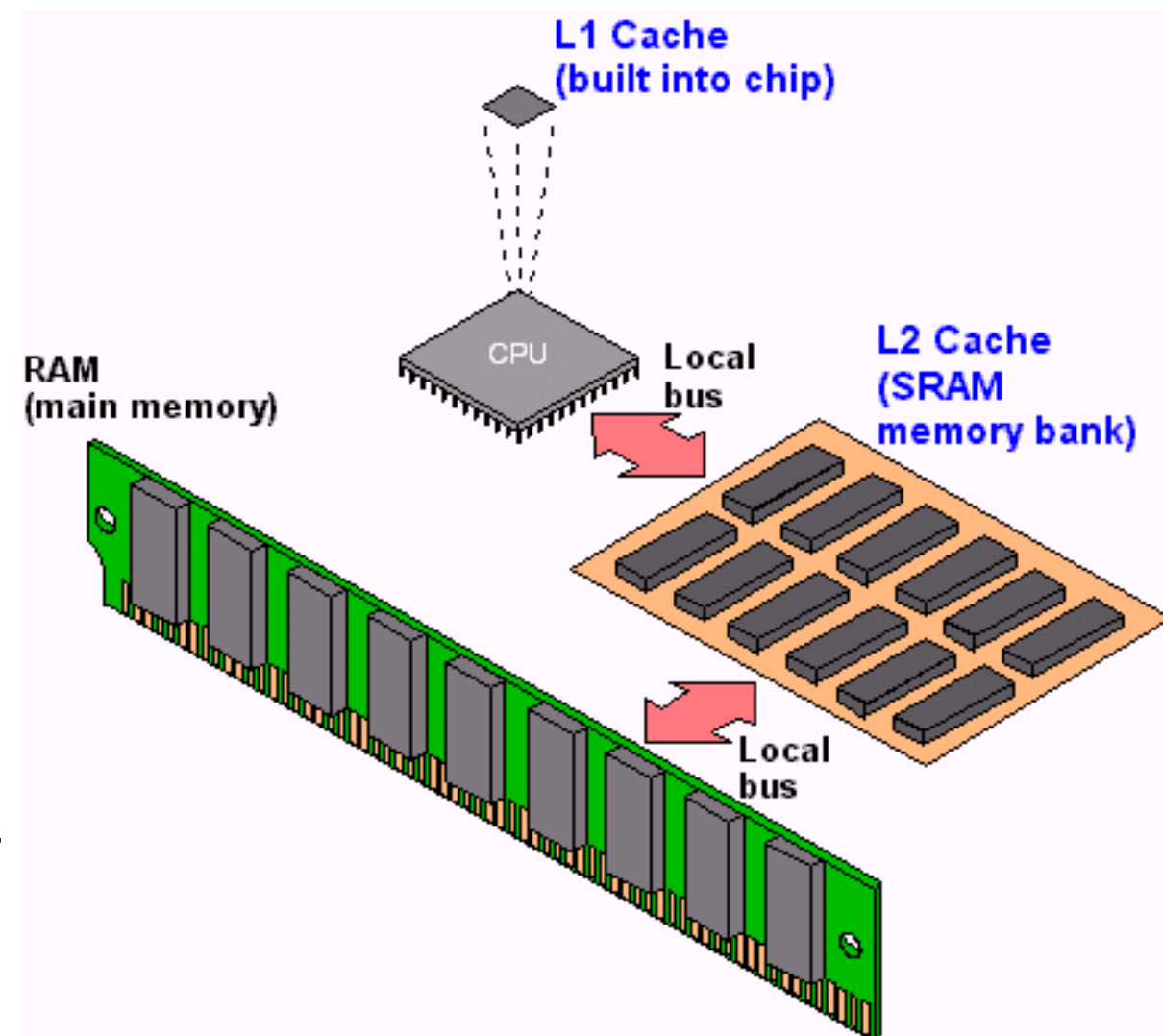
缓存

Cache 缓存:一种更小、更快的存储设备，充当更大、更慢设备中数据子集的暂存区。

内存层次结构的基本思想: 对于每个 k ，第 k 级更快、更小的设备充当 $k+1$ 级更大、更慢设备的缓存。

为什么内存层次结构有效? 与访问 $k+1$ 级数据相比，程序访问 k 级数据的频率更高。因此，级别 $k+1$ 的存储可以更慢，每比特更便宜。

净效应: 一个大型内存池，其成本与靠近底部的廉价存储一样，但它以靠近顶部的快速存储的速度为程序提供数据。我们使用**小快内存**和**大慢内存**的组合来产生大快内存的错觉。



为什么内存层次结构可行

- **10/90 法则 (经验法则)**

10%的静态指令/数据占访问指令/数据的90%

- 指令：内部循环
- 数据：常用的全局变量、内循环栈变量

- **时间局部性(Temporal locality)**

最近访问的指令/数据可能很快会再次被访问

- 指令：内部循环 (下一次迭代)
- 数据：内循环局部变量、全局变量

- **空间局部性(Spatial locality)**

下次访问的存储单元很可能就在刚刚访问的存储单元附近

- 指令：顺序执行
- 数据：数组,结构中的字段, 栈帧中的变量

存储访问的局部性原理

让最近被访问过的信息保留在靠近CPU的存储器中(CACHE)

将刚被访问过的存储单元的邻近单元调到靠近CPU的存储器中(CACHE)

存储访问的局部性原理

在矢量上循环

```
int sumvec(int v[N])
{
    int i, sum = 0;

    for (i = 0; i < N; i++)
        sum += v[i];
    return sum;
}
```

存在寄存器里
(时间局部性)

hopefully in cache
(空间局部性)

Example: 缓存每行
存数组四个元素

v[i]	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7
Access order, [h]it or [m]iss	1 [m]	2 [h]	3 [h]	4 [h]	5 [m]	6 [h]	7 [h]	8 [h]

缓存每行存数组四个元素，i=0，第一次访问时冷启动，缓存缺失，从内存取一块数据（数组4个元素），那么访问2，3，4时命中，以此类推。

存储访问的局部性原理

矩阵循环，按行访问

示例: 每个缓存行包含4个数组元素

注意: 数组在 C 中是 row-major

```
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;  
}
```

stored in registers
(temporal locality)

hopefully in cache
(spatial locality)

- ▶ 数组A: 访问顺序为A[0][0], A[0][1], ..., A[0][3]; A[1][0], A[1][1], ..., A[1][3], 与存放顺序一致, 空间局部性好! 而每个A[i][j]只被访问一次, 故时间局部性差
- ▶ 变量sum: 单个变量不考虑空间局部性; 每次循环都访问sum, 故其时间局部性好!

a[i][j]	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7
i = 0	1 [m]	2 [h]	3 [h]	4 [h]	5 [m]	6 [h]	7 [h]	8 [h]
i = 1	9 [m]	10 [h]	11 [h]	12 [h]	13 [m]	14 [h]	15 [h]	16 [h]
i = 2	17 [m]	18 [h]	19 [h]	20 [h]	21 [m]	22 [h]	23 [h]	24 [h]
i = 3	25 [m]	26 [h]	27 [h]	28 [h]	29 [m]	30 [h]	31 [h]	32 [h]

矩阵循环，按列访问

```
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;  
}
```

stored in registers
(temporal locality)

hopefully in cache
(spatial locality)

访问顺序为A[0][0], A[1][0], ..., A[4][0];
A[0][1], A[1][1], ..., A[4][1];... 与存放顺序不一致，没有空间局部性.

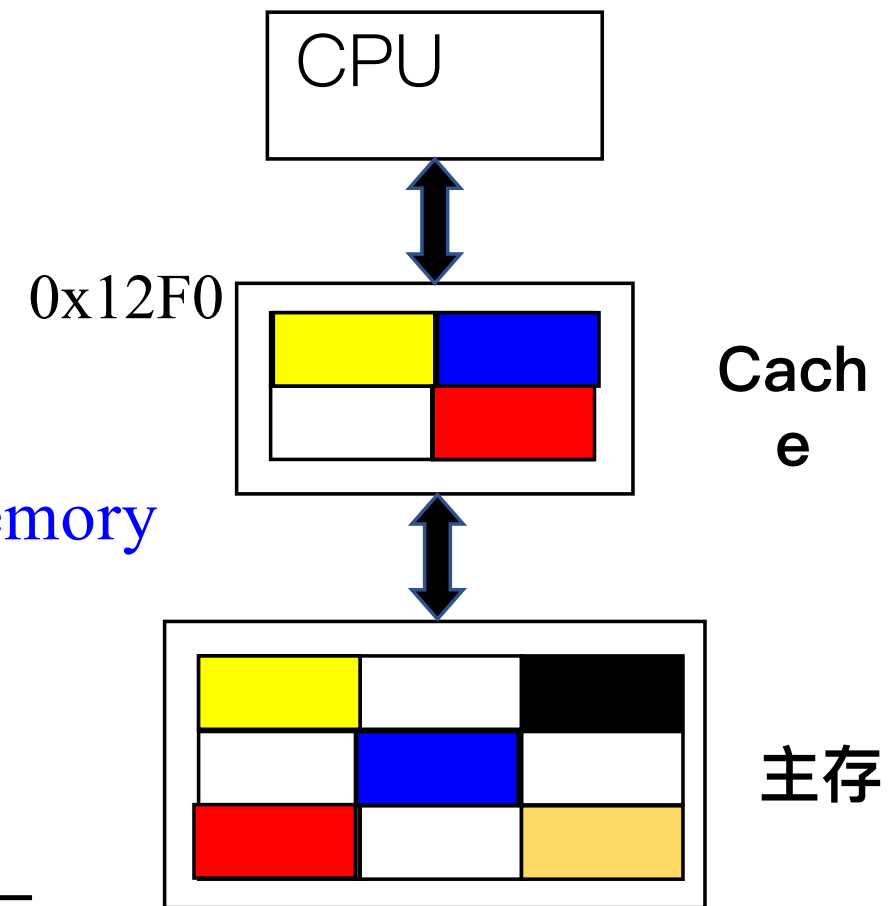
如果缓存比较小，
则全部缺失

a[i][j]	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7
i = 0	1 [m]	5 [m]	9 [m]	13 [m]	17 [m]	21 [m]	25 [m]	29 [m]
i = 1	2 [m]	6 [m]	10 [m]	14 [m]	18 [m]	22 [m]	26 [m]	30 [m]
i = 2	3 [m]	7 [m]	11 [m]	15 [m]	19 [m]	23 [m]	27 [m]	31 [m]
i = 3	4 [m]	8 [m]	12 [m]	16 [m]	20 [m]	24 [m]	28 [m]	32 [m]

Memory Access with Cache

带缓存的内存访问

- Load word instruction: `lw t0,0(t1)`
- `t1` 寄存器内容 `0x12F0`, `Memory[0x12F0] = 99`
- 缓存 : 处理器发地址 `0x12F0` 给缓存
 1. 缓存 是否有内存地址 `0x12F0` 的副本
 - 2a. 有 , 命中 (Hit): 从缓存读出数据 99, 送给处理器
 - 2b. 没有, 缺失(Miss): 缓存把地址 `0x12F0` 送给内存 Memory
 - I. 从内存读出在地址 `0x12F0` 处的数据 99
 - II. 内存把数据送到缓存
 - III. 缓存把地址 `0x12F0` 及它的数据字 99 替换存储在缓存
 - IV. 缓存把数据 99 送给处理器
 2. 处理器把数据 99 加载到寄存器 `t0` 里



CACHE的有关指标

- **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
- 在一个程序执行期间, 设 N_c 是Cache完成存取的总次数, N_m 是主存完成存取的总次数,

那么Cache命中率为:

$$H = N_c / (N_c + N_m),$$

Cache缺失率为: $1 - H$

❖ 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

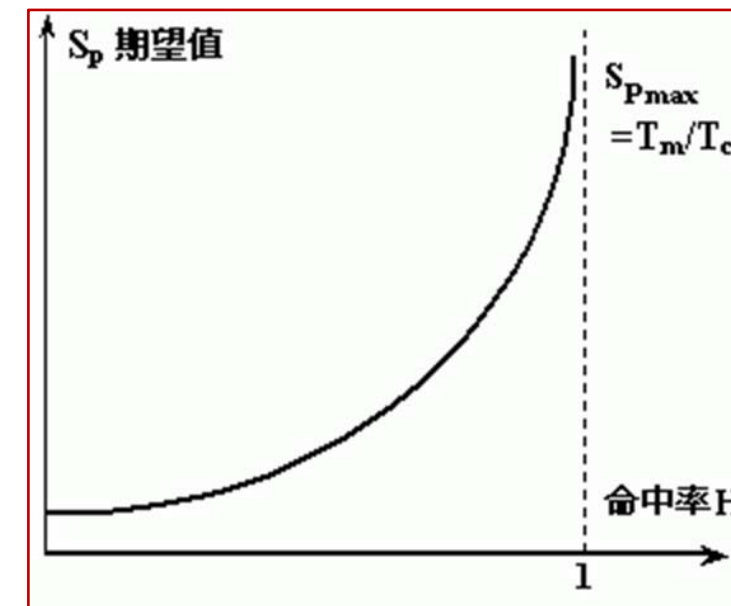
则存储系统的等效访问周期 T 为：

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

■ 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

❖ 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Cache从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)** : 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 (hit)** : CPU要访问的数据在Cache中。
- **缺失 (miss)** : CPU要访问的数据不在Cache中。

层次结构引发的问题

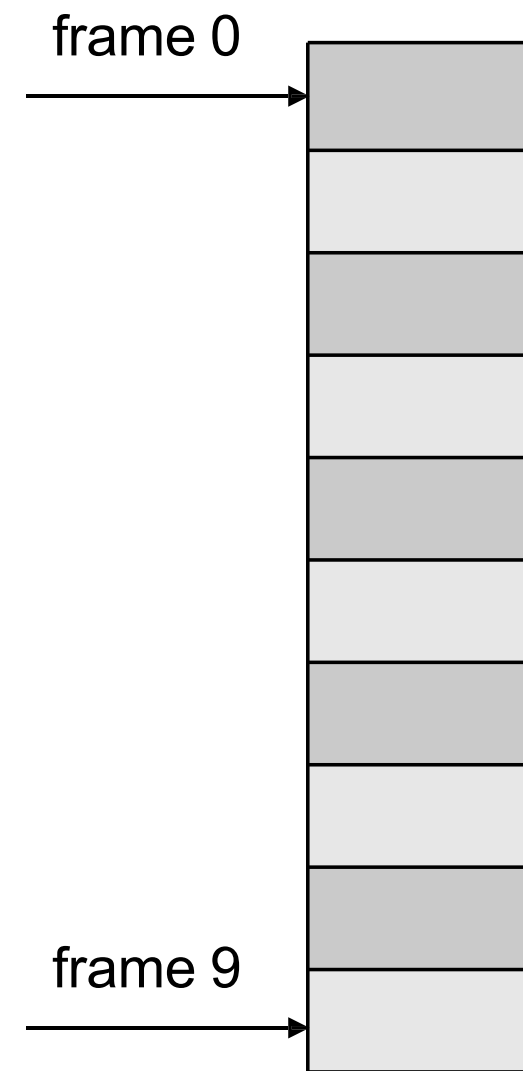
- 逐级信息传送的单位是什么？
 - 字 word, 字块 block, 页表项 page table entry, 页page
- 信息单元放在哪里？
 - 由ISA引导, 受限映射, 通常映射
- 怎样找到存在的信息单元？
 - 取决于映射方式
- 如果缺失会发生什么？
 - 结构危害, 替换算法
- 改变信息单元内容会发生什么？
 - 即., 写会发生什么？

用图书馆类比缓存结构

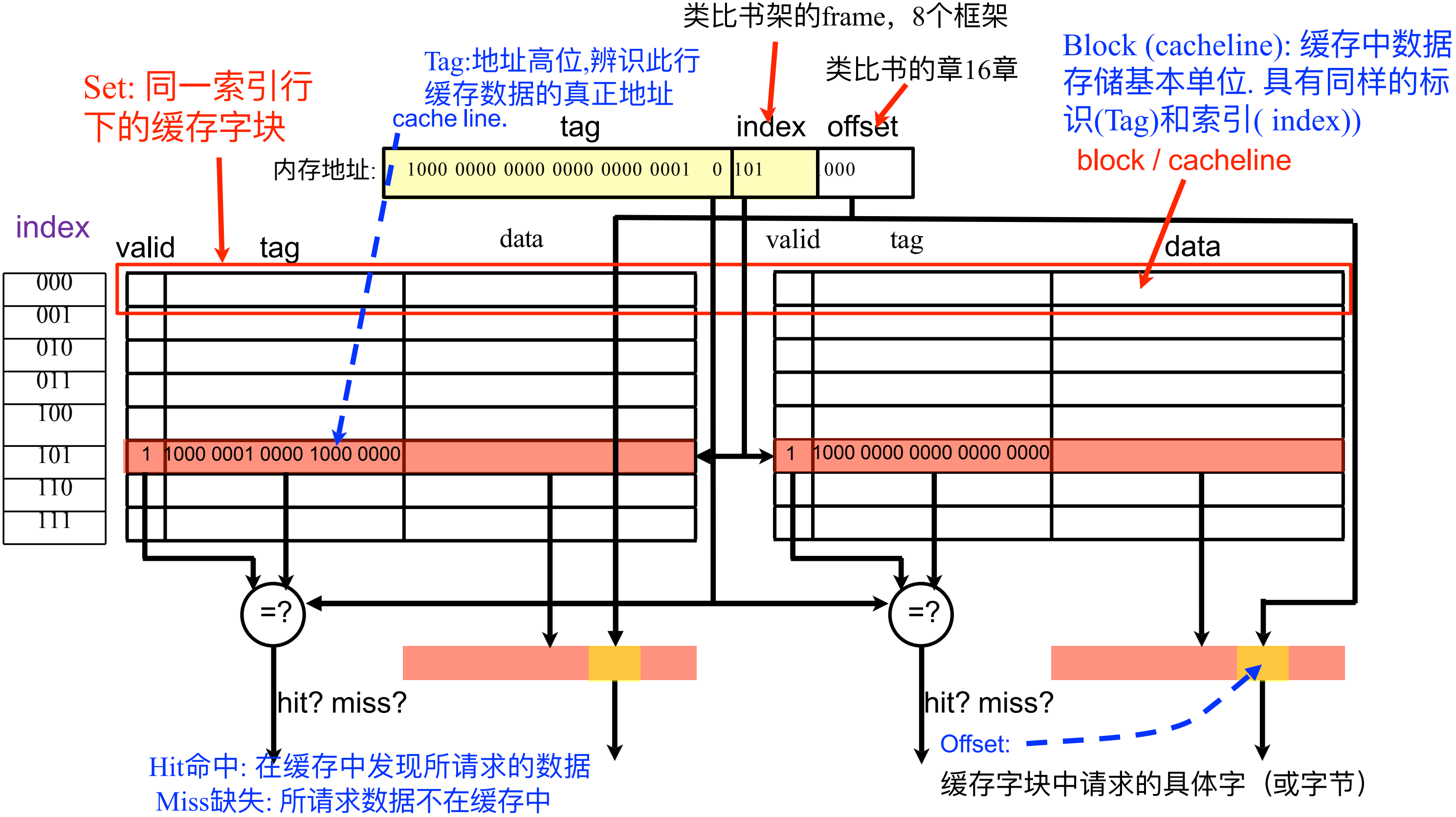
- 用10进制类比
- 假设
 - 图书馆（**类比内存**）有1,000,000 本书（类比缓存中的块**blocks**）
 - 每本书有10章（类比每块有10个字节**bytes**）
 - 每本书的每章有自己的编号（**address**）
 - E.g., 第二本书的第三章用数字 23 表示
 - E.g., 第110本书的第8章用 1108 表示
 - 我的书架（**类比缓存cache**）可以放10本书
 - 书的位置用 框架“**frame**”表示
 - 框架数是书架的“**容量**”

我一次请求(**loads, fetches**)一章或几章

但是我的书架是按照书为单位放置的，每次要拿一本书（不是章）



缓存的结构



缓存逻辑组织

缓存是内存内容的子集

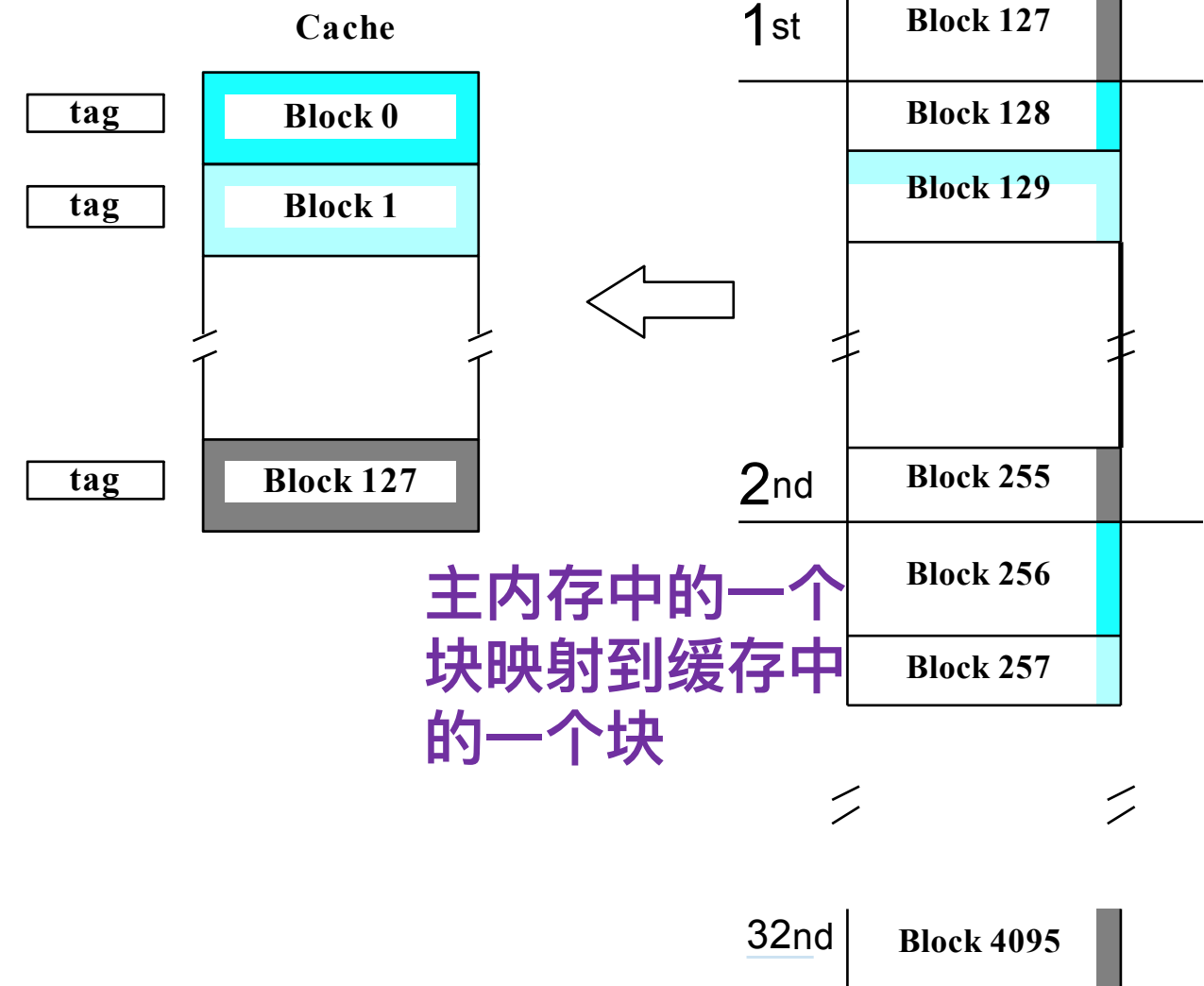
缓存每一行(line)包含

- 标识 tag: 用来辨识是否是我们请求的块
- 状态位: 有效位valid, 修改位dirty等.
- 一块数据

一般将Cache和主存的存储空间都划分为若干大小相同的块（主存中称为：块Block、Cache中称为：行line）

- **缓存容量=块的数量乘以块的大小**

Cache是小容量、高速缓冲存储器，由SRAM组成。

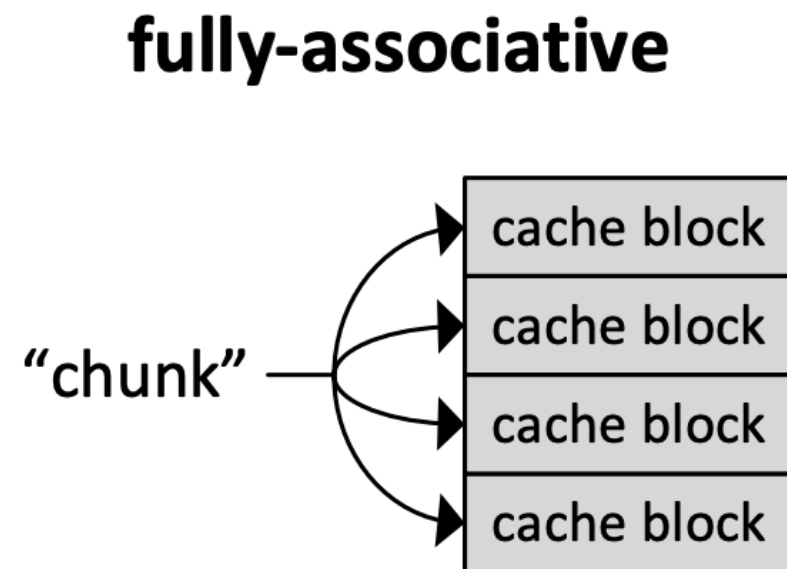


缓存的逻辑组织

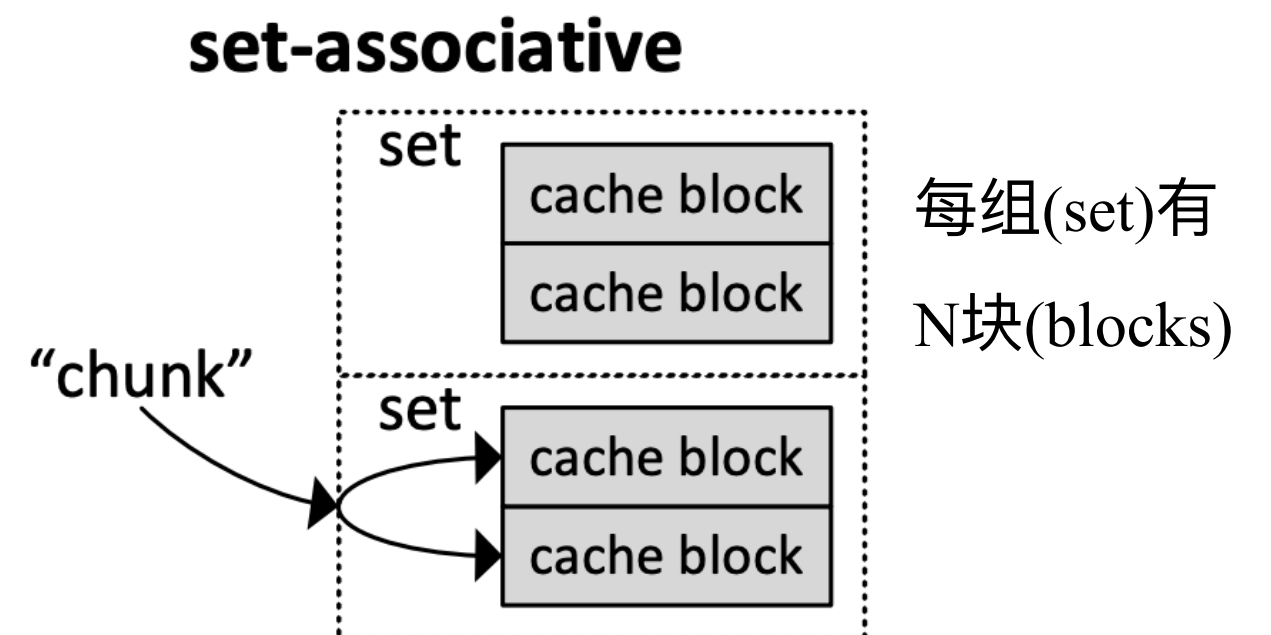
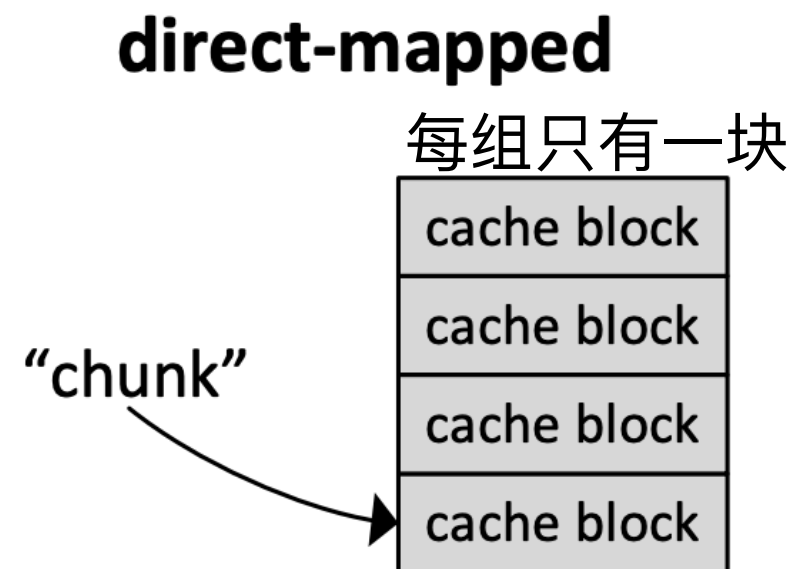
- 关键问题: 怎样把主存的块(chunk) 地址映射到缓存的行 (块) ?
 - 缓存的哪个位置可以让内存的某个块放进来?

Block (line):块 (行) :

- 缓存中的存储单位, 内存在逻辑上被划分为固定大小块 (块), 缓存只能容纳有限数量的块



所有的缓存块在一组内

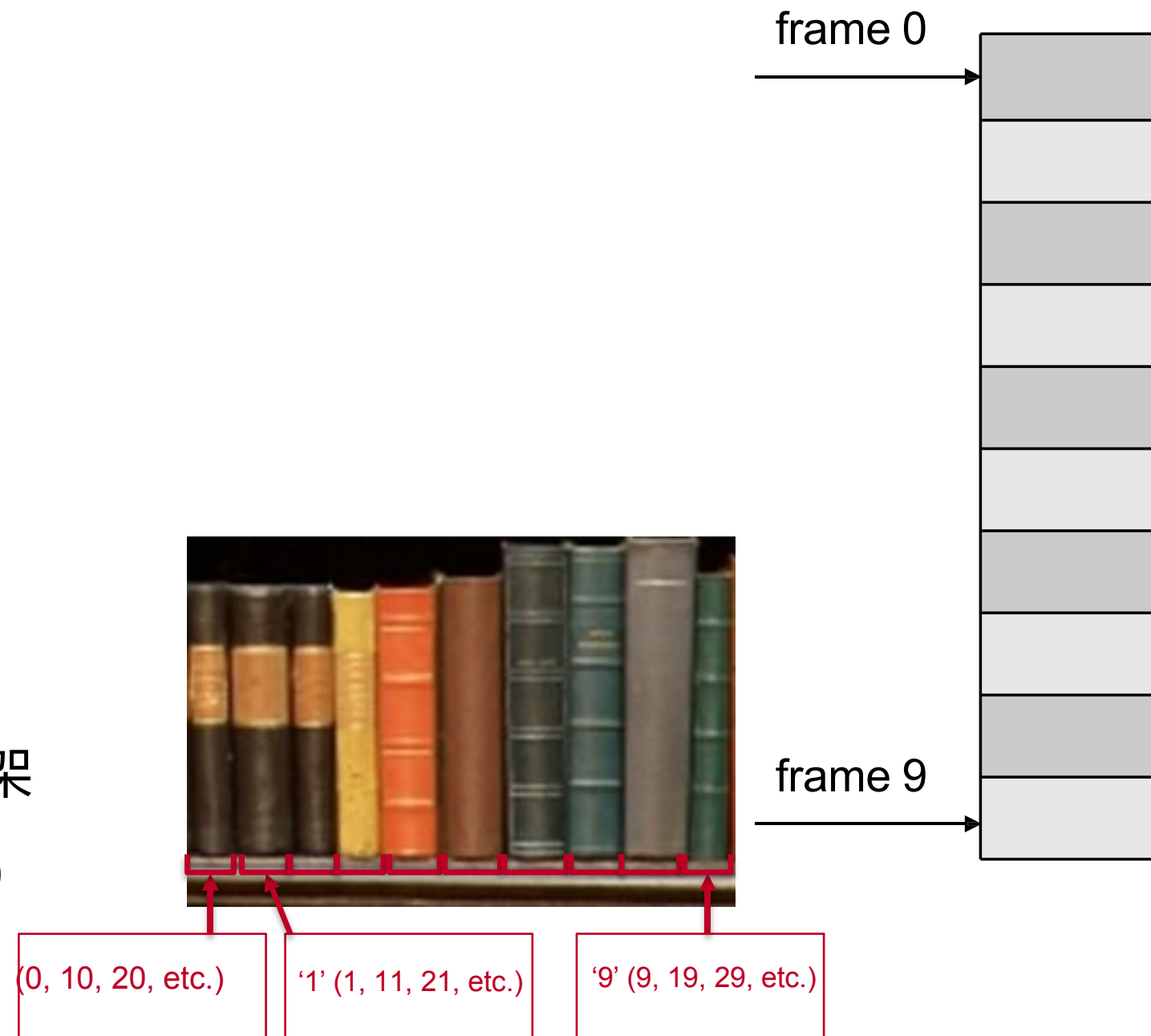


怎样找到数据？

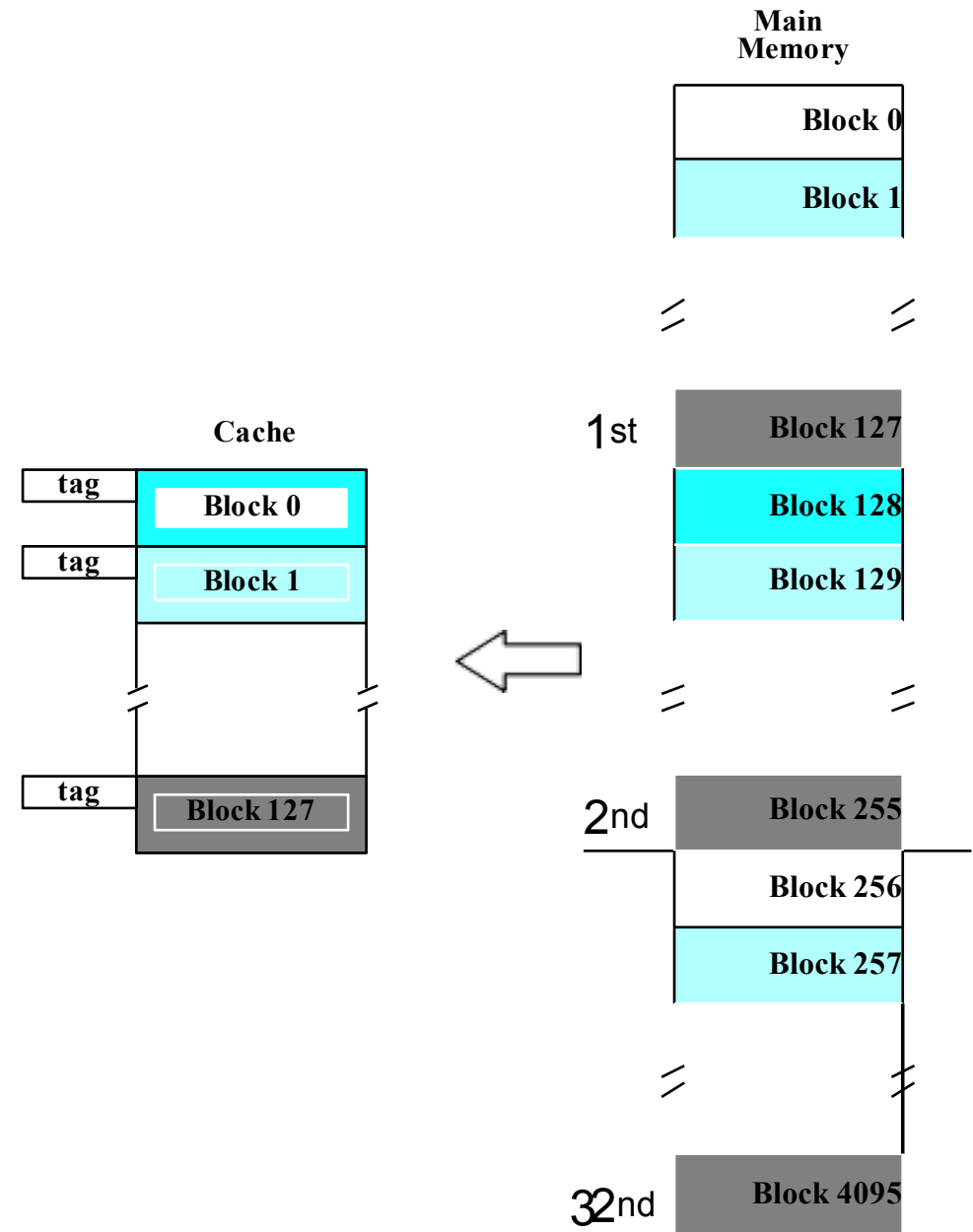
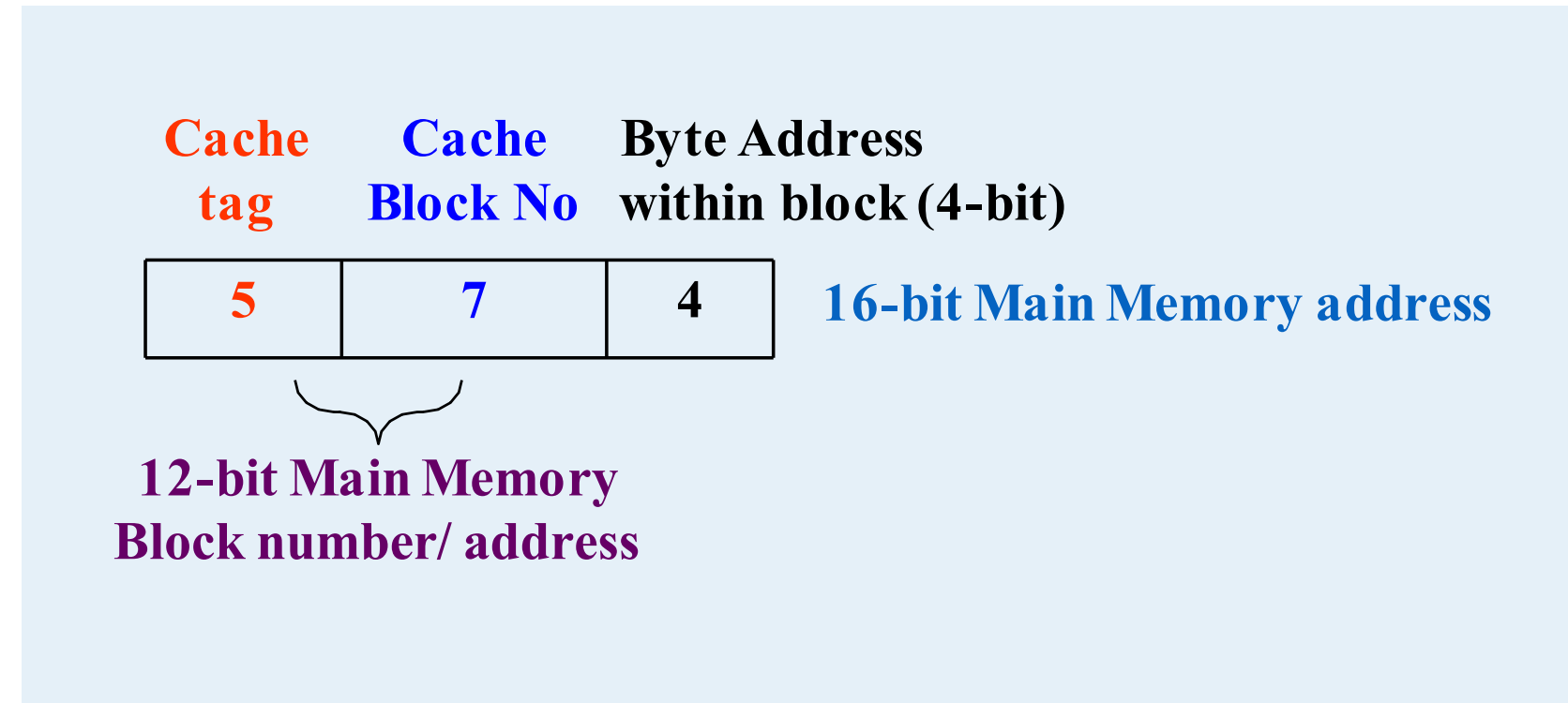
- 缓存划分为S组(S sets)
 - 每个内存地址映射到一组(set)
 - 缓存按照一组(set)中有多少块 (blocks) 分成:
 - 直接映射 (Direct mapped) : 每组只有一块
 - N-way 组相联(set associative): 每组有N块(blocks)
 - 全相联(Fully associative): 所有的缓存块在一组内
 - 下面来了解每种结构的缓存
-

最不灵活的结构: Direct-mapped

- 直接映射 (**direct-mapped**)
- 编号 X 的书映射到 $X \bmod 10$ 这个框架
 - Book 0 in frame 0
 - Book 1 in frame 1
 - Book 9 in frame 9
 - Book 10 in frame 0
 - Etc.
- 如果你想把第3本书和第23本书同时放在书架上, 会发生什么? 你不能! 必须更换 (**驱逐**) 一个以腾出空间给另一个。



Direct Mapping 直接映射



- ▶ $2^4 = 16$ bytes in a block (每块可以存16个字节数据)
- ▶ $2^7 = 128$ Cache blocks (缓存可以存128块数据)
- ▶ $2^{(7+5)} = 4096$ main memory blocks (主存分成4096块)
- ▶ 主存中每一个块(block) j 只能映射到某一固定的Cache块中 ($j \bmod 128$) (图中相同颜色)
- ▶ 如果缓存tag与目标地址匹配, 则命中 hit

Direct Mapping 直接映射

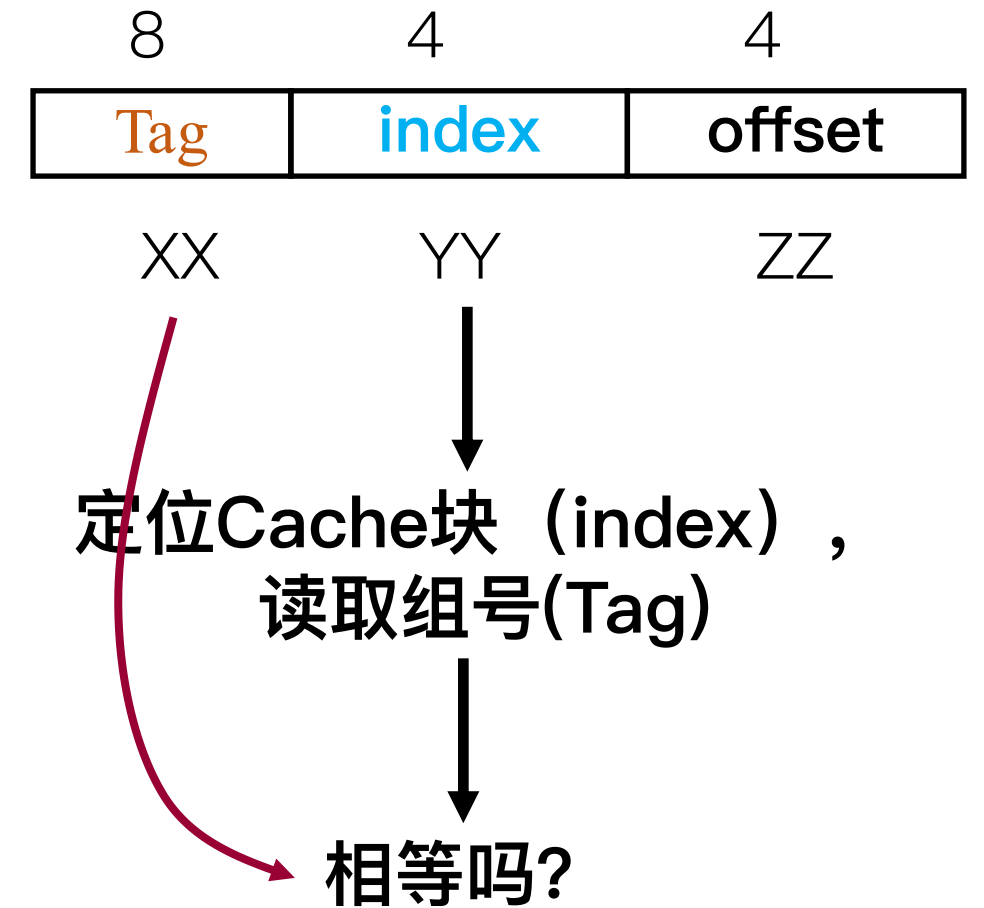
如何由内存地址（字节地址）检测是否在缓存中，它有一个有效的映射？

内存地址分为3个字段

- ▶ 主内存set组号决定了块在缓存中的位置(index)
- ▶ **Tag** 用于识别哪个块在缓存中（因为许多 MM 块可以映射到缓存中的相同位置）
- ▶ 地址中的最后位(offset)用来选择块中的目标字

Example: 给定地址(**t**,**i**,**o**) (16-bit)

1. 由 **i** 定位缓存位置，内存中的**t**与这行缓存中的Tag比较
2. 如果不等, 缓存没有命中 cache miss! 在当前缓存行从内存取出块进行替换 (**t**,**i**) (12-bit)



据此判断内存目标块
是否在Cache中

直接映射Cache 举例

- One word/block, cache size = 1K words

Cache每块存一个字

(4 bytes) ,

块数为1 K的 cache

内存地址32位,

Cache索引1k用10位

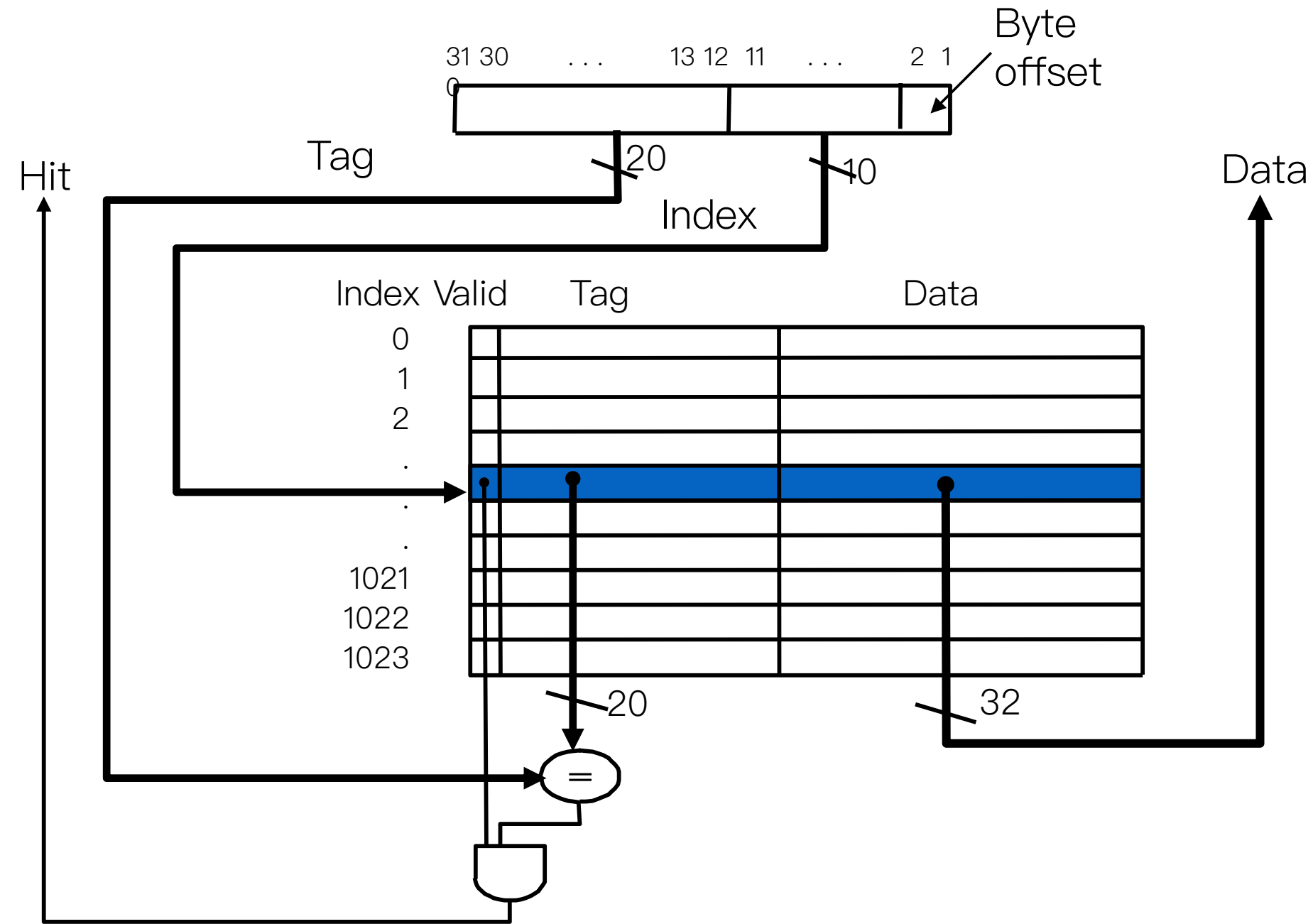
地址, 一个字, 2位

地址, 低位地址

$10+2=12$

$32-12=20$ 为内存组号,

存于Cache的Tag。



What kind of locality are we taking advantage of?

Direct Mapping Address Usage

- If 80386 used Direct Cache Mapping :

以字节编址32位的内存MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words

内存以块划分: Number of blocks in MM = $2^{32} / 2^4 = 2^{28}$

缓存的行数: Number of Cache Block Frames = $2^{16} / 2^4 = 2^{12} = 4096$

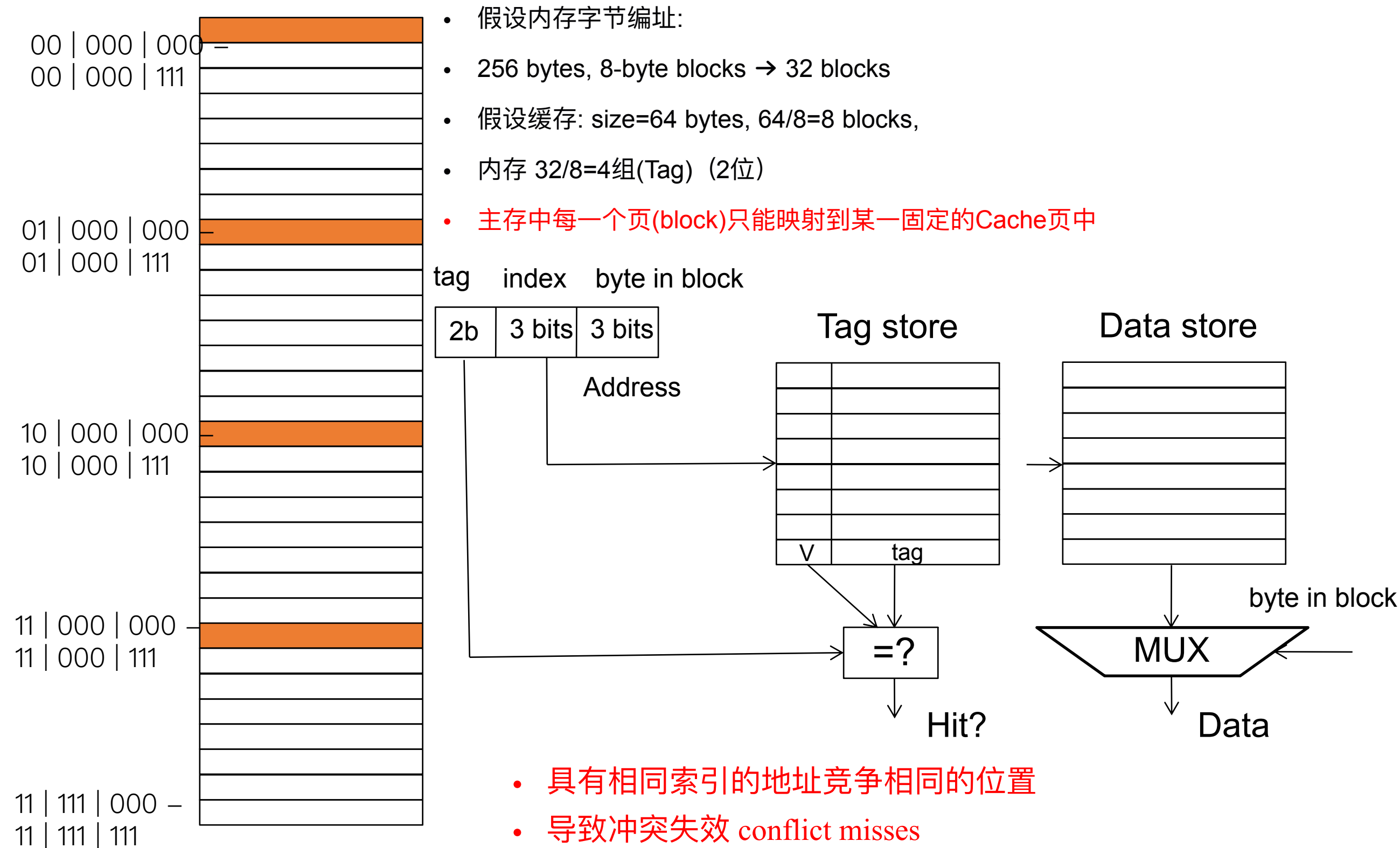
缓存对行索引所需位数: Number of "colors" => 12 Block field bits

在缓存的每一行标识位: $32-12-4=16$

– 16 Tag Field Bits

Tag	CBLK	Word	Byte
16	12	2	2
Block ID=28			

直接映射缓存：放置和访问



- 具有相同索引的地址竞争相同的位置
- 导致冲突失效 conflict misses

Caching: A Simple First Example

Cache

Index Valid Tag Data

n: Cache块号的位数2

00			
01			
10			
11			

组号: Cache的Tag存内存相应块的高位地址

Q1: Is it there?

将缓存标记与高 2 位内存地址位进行比较, 以判断内存块是否在缓存中

Main Memory

0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

xx:两个低位定义字中的 (32-b words)字节。
Block=1 Word

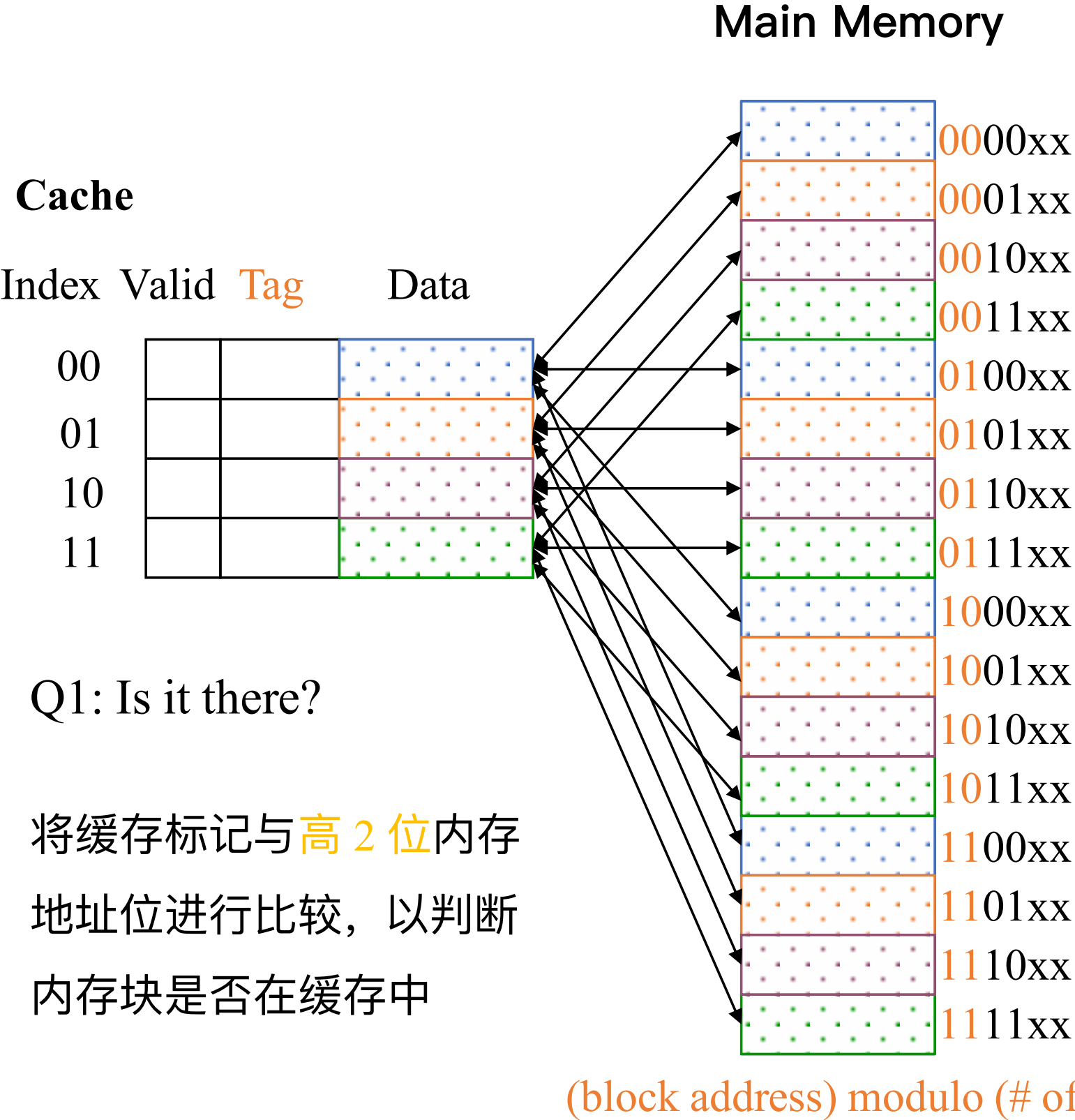
页 (块) 号: Cache的索引为内存相应块的低位地址(2位)

Q2: How do we find it?

使用接下来的 2 个低阶内存地址位 - 索引 - 来确定哪个缓存块 (即, 以缓存中的块数为模)

(block address) modulo (# of blocks in the cache)

Caching: A Simple First Example



xx:两个低位定义字中的 (32-b words)字节。
Block=1 Word

Q2: How do we find it?

使用接下来的 2 个低阶内存地址位 - 索引 - 来确定哪个缓存块 (即, 以缓存中的块数为模)

Cache Field Sizes 举例

- 内存32-bit地址，缓存可以存数据16KB，块大小为4-word，那么直接映射缓存全部位的数量是多少？

- 块大小为4-word， $\text{offset}=2+2=4$ ，每行存16个字节数据，缓存16KB， $16\text{KB} / 16 = 1\text{k}(\text{line}) = 1024 (2^{10}) \text{ blocks}$
- 缓存每行存数据 4×32 字或 128 位，每行有一位有效位，还有标识Tag.

标识位 = $32 - (10 + 2 + 2) = 18 \text{ bits}$ ，加一个有效位 = 19 bits

- 所以缓存存储的所有位是：

$$2^{10} \times (4 \times 32 + 18 + 1) = 2^{10} \times 147 = 147\text{Kbits}$$





- 是需要存储数据空间所需的 $1.15 \times$

Question: 直接映射的命中率

考虑一个4行的 (4-block) 空的缓存, 所有的块初始化有效位为0 (not valid) . 给定主存的字地址“0 1 2 3 4 3 4 15”进行访问, 计算缓存的命中率.

Cache

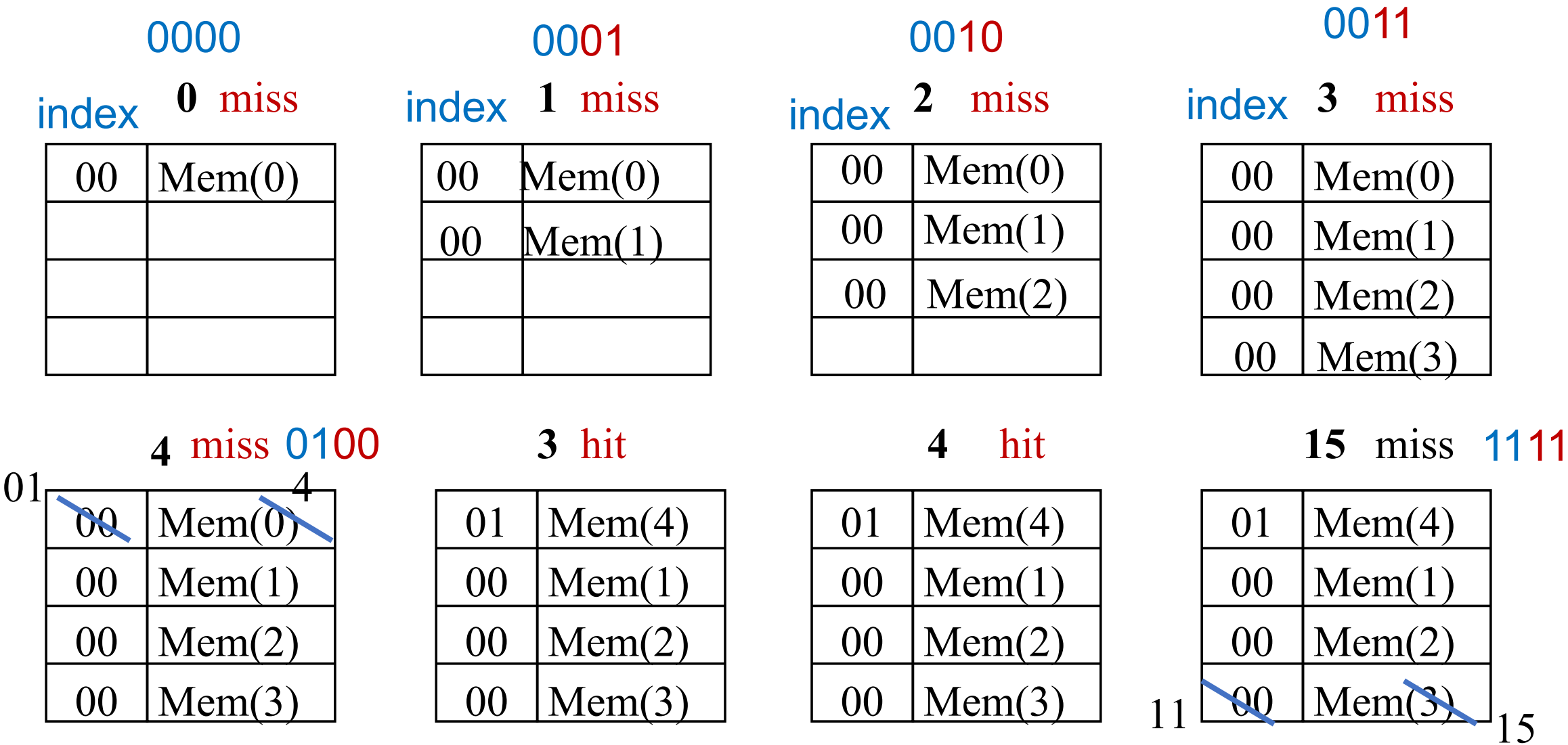
Index Valid Tag Data

00			
01			
10			
11			

Direct Mapped Cache

如果每块存一个字

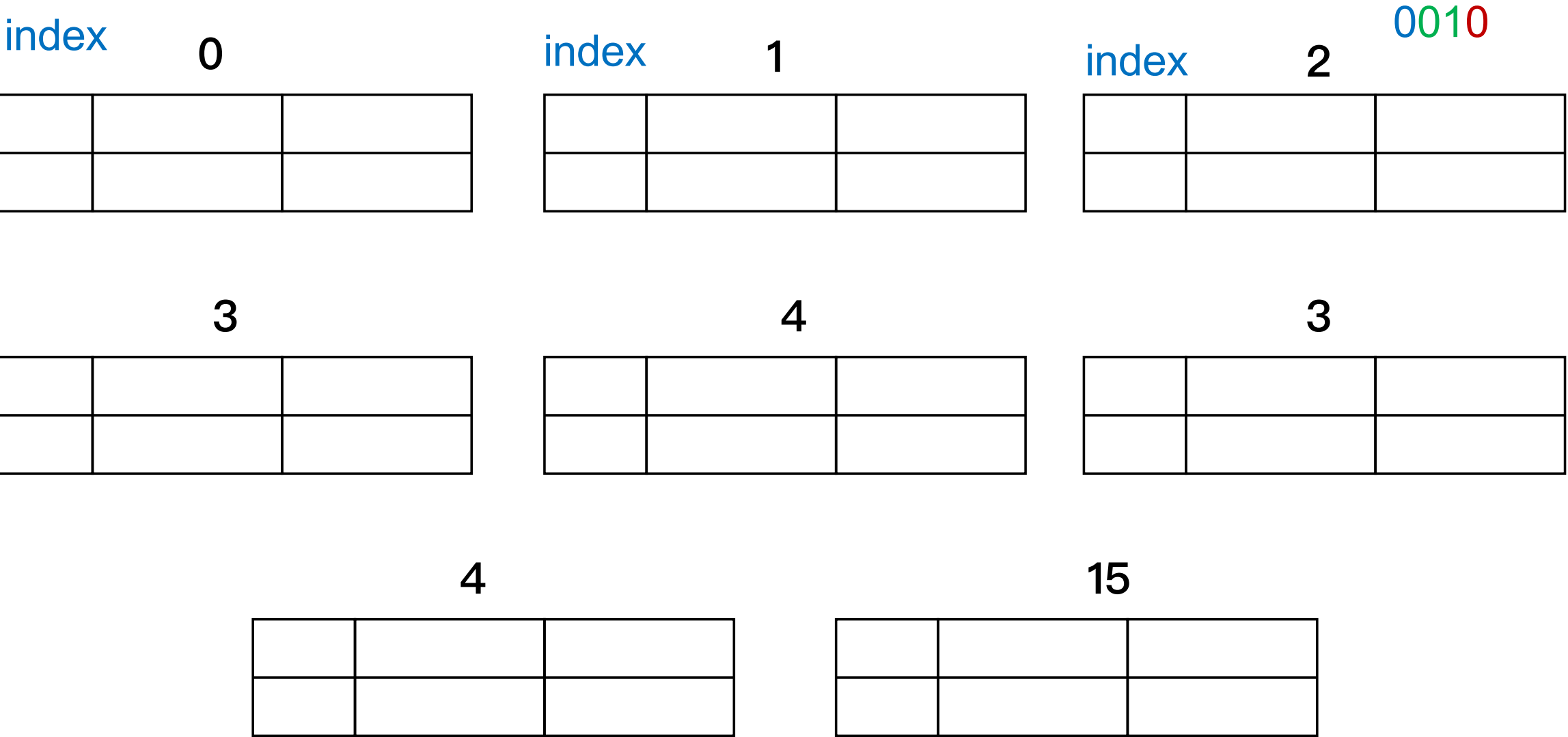
每行 (line) 存
一个字，地址以
字索引， cache
大小4字,4行，索
引2位，
内存16/4=4组



| 8 requests, 6 misses

Taking Advantage of Spatial Locality

如果每块存2个字



Taking Advantage of Spatial Locality

如果每块存**2**个字

0 miss 00**00**

00	Mem(1)	Mem(0)

1 hit 00**01**

00	Mem(1)	Mem(0)

2 miss 00**10**

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit 00**11**

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

01

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

5 4

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss 11**10**

11

00	Mem(3)	Mem(2)
01	Mem(5)	Mem(4)

15 14

| 8 requests, 4 misses

由于空间局部性，更大的 LINE (block) 意味着更少的不命中，但一旦缺失，需要更长的时间，即缺失代价增加

因为缓存总大小不变，缓存变为2行，每行存2个字。

example

- Analyzing C Code
- # define NUM_INTS 4096
- int A[NUM_INTS]; /* A lives at 0x10000*/
- int i, total=0;
- for (i=0; i < NUM_INTS; i+=64) { A[i] = i; } /* Line 1 */
- for (i=0; i<NUM_INTS; i+=64) { total +=A[i]; } /* Line 2 */
- 计算机字节编址，容量 1MiB. 它带有 8KiB 片内缓存，块大小128 个字，直接映射.

计算第一行Line 1数据访问的命中率Line 1:

整数访问相隔 64 words, 每个块可以访问两次. 第一次访问冷启动都是缺失, 但第二次访问命中, 因为每个块128 个字, A[i] 和 A[i+64] 在同一个缓存块里. 50%

计算第二行Line 2数据访问缓存命中率:

循环次数 $4096/64=64$, 缓存一共16行, 缓存每16次循环装满缓存, 又重新装入, 不能利用以前装入的数据, 只能像第一行循环一样, 每次加载一块, 第二次可以利用空间局部性命中, 只有 **50%的命中率**.

另一个极端的示例



°Cache大小= 4 bytes

块大小= 4 bytes

- cache中只能有唯一一个数据块
- °真: 如果正在访问一个信息项, 那么很可能在最近的将来还将访问该信息项
 - 但是, 通常却不会立即再次使用该信息项!!!
 - 那么, 再次访问该信息项时, 仍然很可能会失效
 - 继续向cache中装入数据, 但是在它们被再次使用之前会被强行排出cache
 - cache设计人员最害怕出现的事情: 乒乓现象
- °冲突失效 (Conflict Misses) 是下述原因导致的失效:
 - 不同的存储位置映射到同一cache索引
 - 解决方案1: 增大cache容量
 - 解决方案2: 对同一Cache索引可以有多个信息项

另一个极端的示例

- 考虑一个4行的（4-block)空的缓存,每块一个字，所有的块初始化有效位为0（not valid）. 给定主存的字地址“0 4 0 4 0 4 0 4”进行访问, 计算缓存的命中率。

0	4	0	4																																
<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>								
0	4	0	4																																
<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>									<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>								

Another Reference String Mapping

- 考虑一个4行的 (4-block)空的缓存, 所有的块初始化有效位为0 (not valid) . 给定主存的字地址“0 4 0 4 0 4 0 4”进行访问, 计算缓存的命中率。



| 8 requests, 8 misses

- 由于**冲突缺失 (conflict misses)** 造成的乒乓效应 – 两个内存地址映射到缓存的同一位置块。

Cache与主存之间的映射—直接映像优缺点

❖ 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

5.2 32位内存地址访问如下序列字地址，3，180，43，2，191，88，190，14，181，44，186，253（10进制表示）。

5.2.1. CACHE 直接映射，16行，每行block一个字，分析访问序列的缺失命中，以及CACHE的TAG,Index. 16行索引地址4位，每行存一个字，所给为字地址

Word Address	Binary Address Tag Index	Tag	Index	Hit/Miss	Types of cache misses
3	0000 0011	0	3	M	Compulsory
180	1011 0100	11	4	M	Compulsory
43	0010 1011	2	11	M	Compulsory
2	0000 0010	0	2	M	Compulsory
191	1011 1111	11	15	M	Compulsory
88	0101 1000	5	8	M	Compulsory
190	1011 1110	11	14	M	Compulsory
14	0000 1110	0	14	M	Compulsory
181	1011 0101	11	2	H	
44	0010 1100	2	6	M	Compulsory
186	1011 1010	11	5	M	Compulsory
253	1111 1101	15	6	M	Compulsory

5.2.2 每个block 2个字， 8个block.index 3位，offset 1位

Word Address	Binary Address Tag Index offset	Tag	Index	Hit/Miss	Types of cache misses
3	0000 001 1	0	1	M	Compulsory
180	1011 010 0	11	2	M	Compulsory
43	0010 101 1	2	5	M	Compulsory
2	0000 001 0	0	1	H	
191	1011 111 1	11	7	M	Compulsory
88	0101 100 0	5	4	M	Compulsory
190	1011 111 0	11	7	H	
14	0000 111 0	0	7	M	Compulsory
181	1011 010 1	11	2	H	
44	0010 110 0	2	6	M	Compulsory
186	1011 101 0	11	5	M	Compulsory
253	1111 110 1	15	6	M	Compulsory

练习

如图32-bit 内存地址，块大小 4 words，计算

(A) 内存可以在Cache存最多多少个字，Cache索引地址多少位，组号Tag位需多少位？

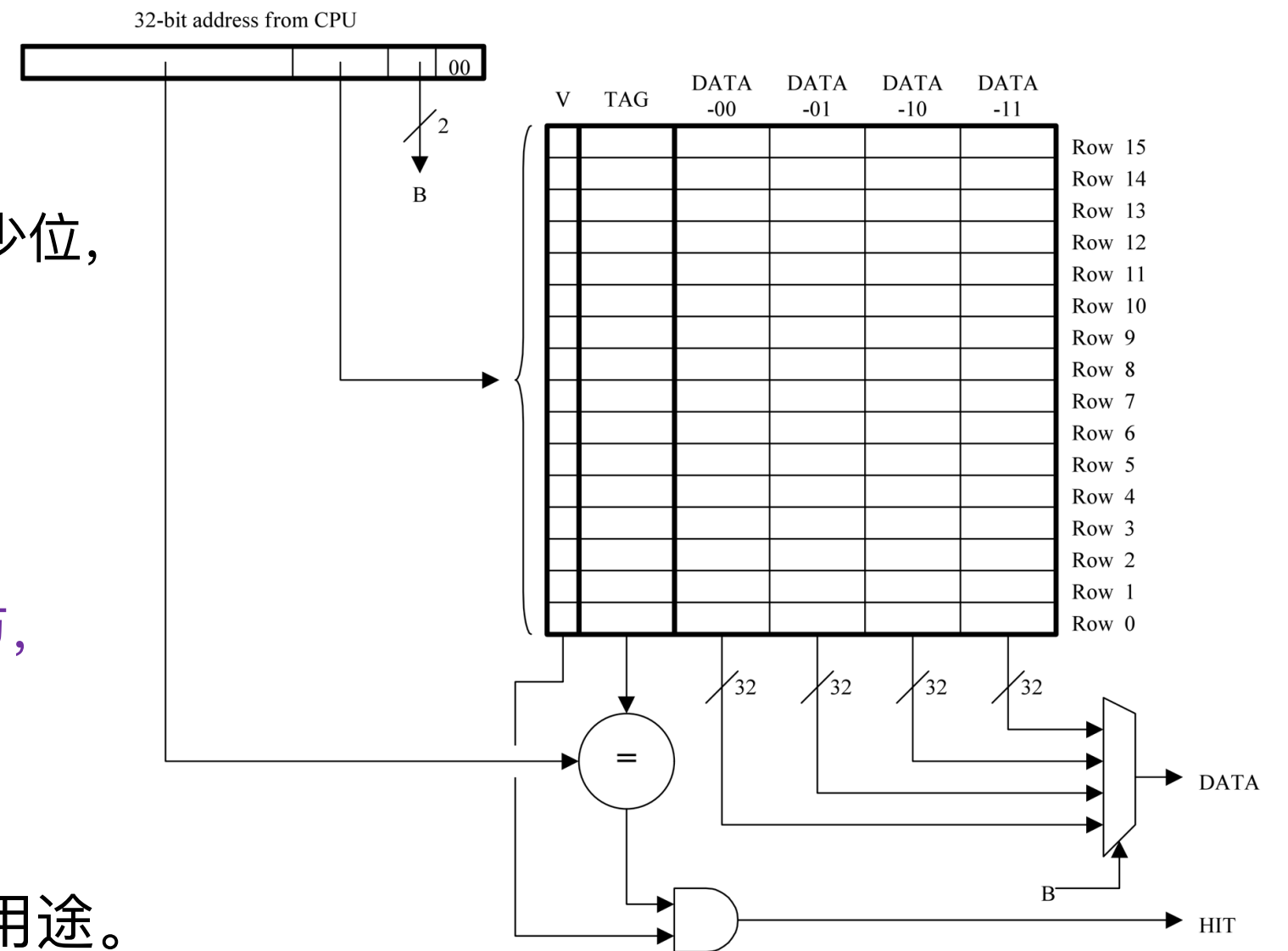
一行4个字，一共16行

一共 $4 \times 16 = 64$ words。

Cache一共16行，4位索引地址，4个字，2位，字（4个字节，2位地址）， $32 - (4 + 2 + 2) = 24$ 位Tag

(B)简要说明与每个高速缓存行关联的一位 V 字段的用途。

V 位表示对应的缓存行包含有效数据。



(C)假设内存位置 0x3328C 的内容存在于缓存中。使用图中的行和列标签，我们可以在哪些缓存位置找到该位置的内容？对于出现数据的缓存行，Tag 的值必须是什么？

0x3328c=0011 0011 0010 1000 1100

- 第8行, block 3 (Data -11)

(D)来自位置 0x12368 和 0x322F68 的数据可以同时存在于缓存中吗？来自位置 0x2536038 和 0x10F4 的数据呢？解释

位置 0x12368 和 0x322f68 映射到同一个高速缓存行 6，因此不能同时在高速缓存中。位置 0x2536038 和 0x000010f4 可以同时存在缓存中，因为它们映射到不同的行，（分别为 3 和 F）

(E) 当访问导致缓存未命中时，需要从内存中取出多少字来填充适当的缓存位置并满足请求？

在高速缓存未命中时，需要提取 4 个连续字来填充该行。

直接映射的问题

- 假设字块大小 8B blocks
- CPU按下列字节地址访问:
 - 2, 67, 2, 67, 2, 67, 2, 67, ...
- 命中率? 缺失率?
- 怎样能更好地利用缓存的空间?

8 sets, 8 frames

- 2= 0 000 010B
- 67=1 000 011

两个地址都映射到set0,发生冲突失效

缺失率100%

	way 0		
	valid	tag	data
set 0			M[0]-M[7] M[60]-M[67]
set 1			
set 2			
set 3			
set 4			
set 5			
set 6			
set 7			

2-Way Set-Associativity

	way 0			way 1		
	valid	tag	data	valid	tag	data
set 0			M[0]-M[7]			M[60]-M[67]
set 1						
set 2						
set 3						

假设缓存容量不变，每个set有2路，2-way，即同一个set有两个位置可以放置块

- 2 = 00 **00** **01**0B

变成4 sets, 2 ways/set 8 frames不变

- 67 = 10 **00** **01**1

同样的访问序列 2, 67, 2, 67, 2, 67, 2, 67, ...

两者还是都映射到set0，因为有两个位置，

当访问67时，不用驱逐0出去，只有第一次访问缺失，后面都命中

Reducing Cache Miss Rates

通过灵活放置块来减少Cache的缺失

在直接映射缓存中，一个内存块只能映射到一个缓存块block位置，
在另一个极端，可以允许将内存块映射到任何缓存块位置

– fully associative cache 全相联映射

检索项多、硬件实现复杂，Cache利用率高)

– n-way set associative 组相联映射

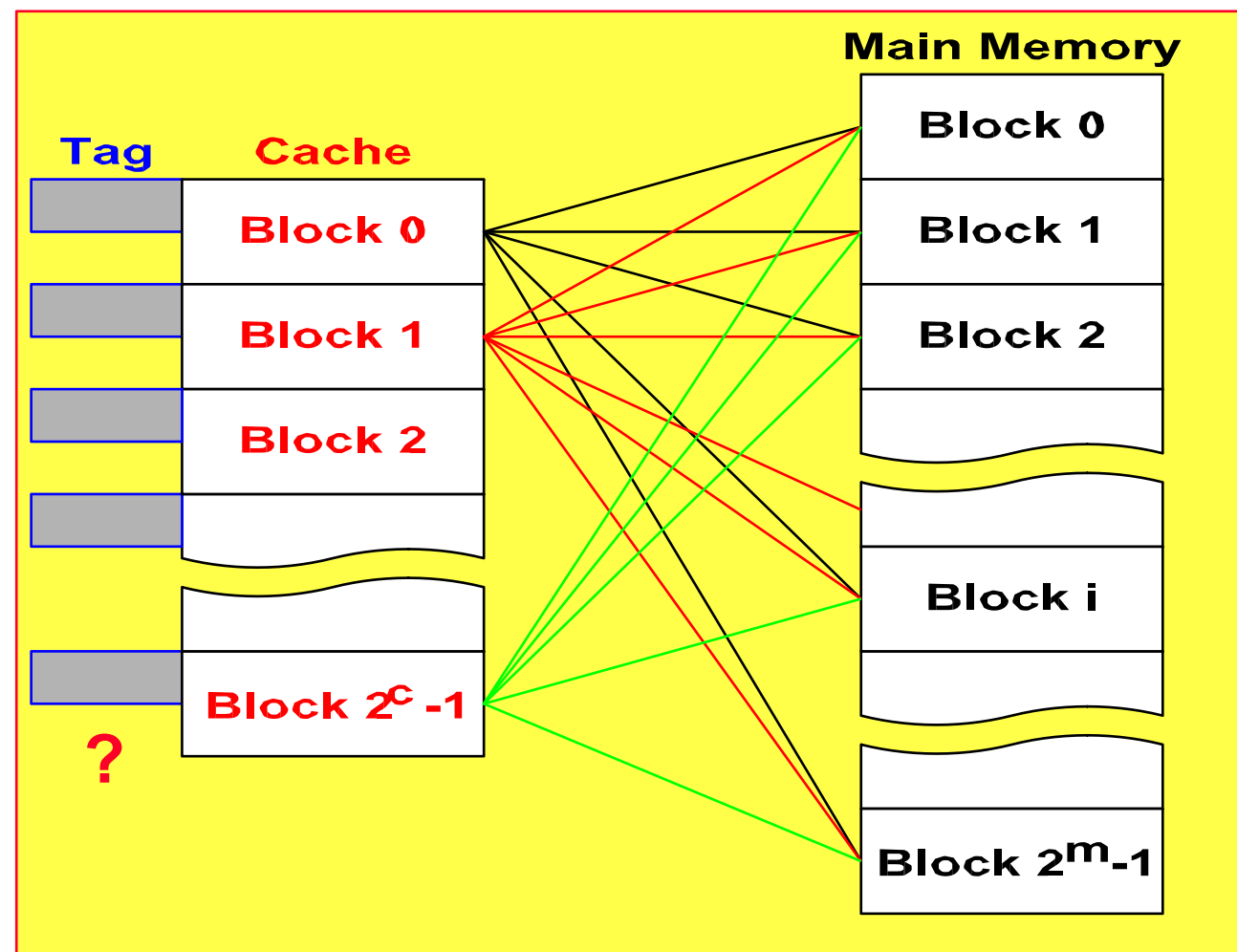
是一种直接映射和全相联映射的折衷方案。主存和Cache 都分组，
主存中组内的页数与Cache的分组数相同。

$(\text{block address}) \bmod (\# \text{ sets in the cache})$

Cache与主存之间的映射—全相联

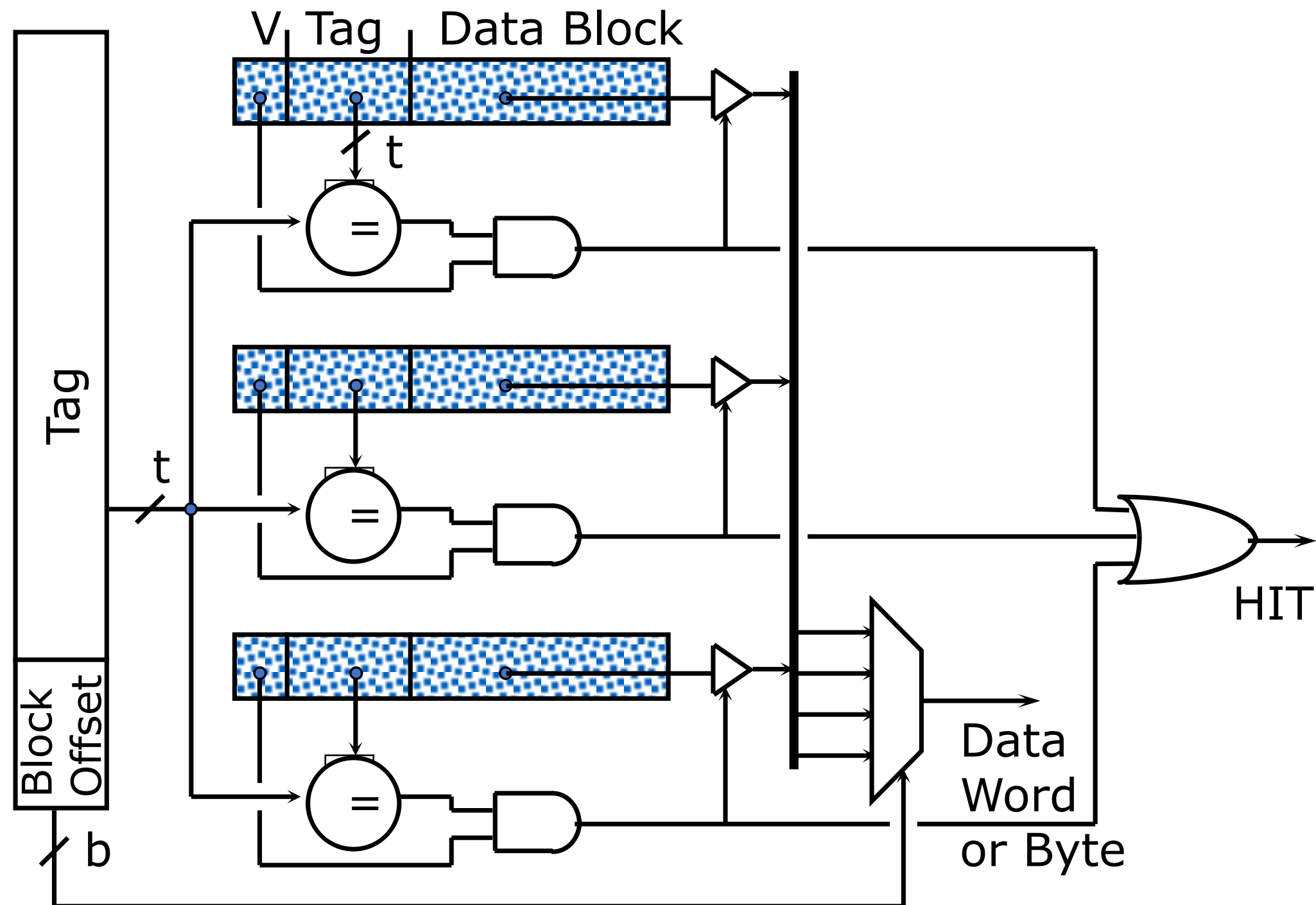
❖ 全相联映射 (Associative Mapping)

- 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。



Fully Associative Cache

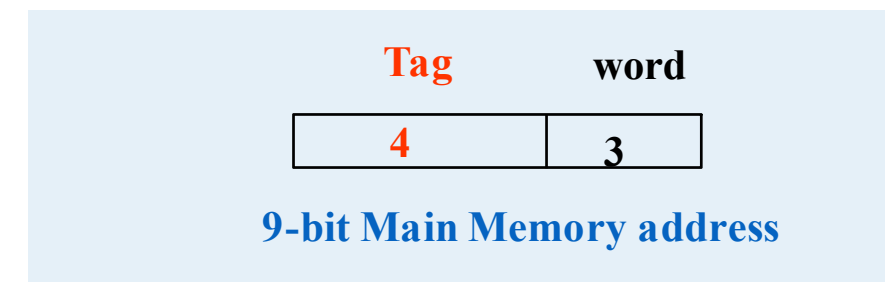
在此示例中，所有 Tag 标签条目必须与地址标签并行比较（通过硬件）



Fully Associative Hit Logic

全相联命中逻辑

- 举例:
 - 全相联, MM = 128 words (2^7), Cache Size = 32 (2^5) words,
Block Size = (2^3) words
- 内存中的块数 $MM = 2^7 / 2^3 = 2^4$
Block ID = 4 bits (Tag)
- 缓存行数 = $2^5 / 2^3 = 2^2 = 4$ (line)
 - 每行存 4 位 Tags + 1 valid bit
 - 需要 4 个 5 位比较器



Fully Associative 不能规模太大

- 如果 80386 采用全相联:

- MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words

- 内存中的块数 $MM = 2^{32} / 2^4 = 2^{28}$

$A == B$ Block ID = 28 bits (Tag)

- 缓存行数 = $2^{16} / 2^4 = 2^{12} = 4096$

- 存储 4096 个 28-bits 的 Tags + 1 valid bit

- 需要 4096 个 29 bits 的比较器

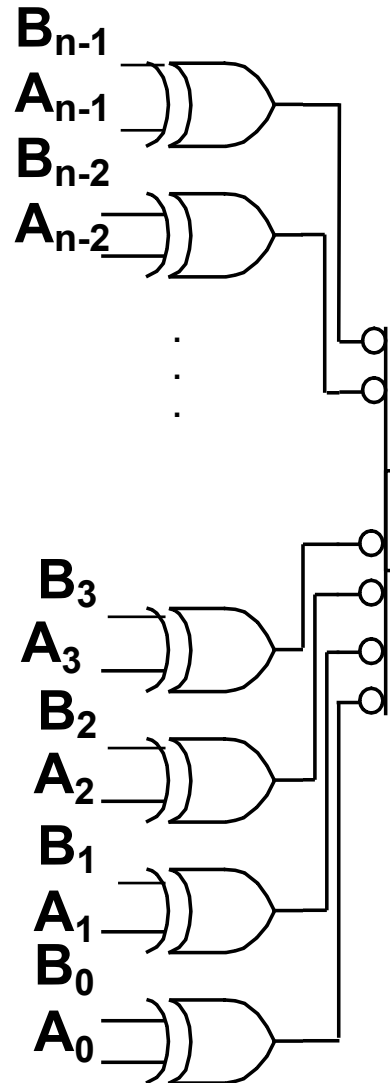
标签位/缓存行 = 29 bits

数据位/缓存行 = 4×32 bits

$29/128 = 23\%$

Tags Are Expensive!

缓存中的23% 存储用于标签



Cache与主存之间的映射—全相联

❖ 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

组相联映射主存地址格式

- 是一种直接映射和全相联映射的折衷方案。主存和Cache 都分组，主存中组内的页数(block)与Cache的分组数(set)相同。
- Tag的内容：主存中与该Cache数据块对应的在内存中的块地址组号。

块组地址(tag)	组内地址set	块内偏移offset
-----------	---------	------------

该映射方式将所有Cache分为K行 (set) ，每行有R块(way)，
则有以下关系：

$$i = j \bmod K$$

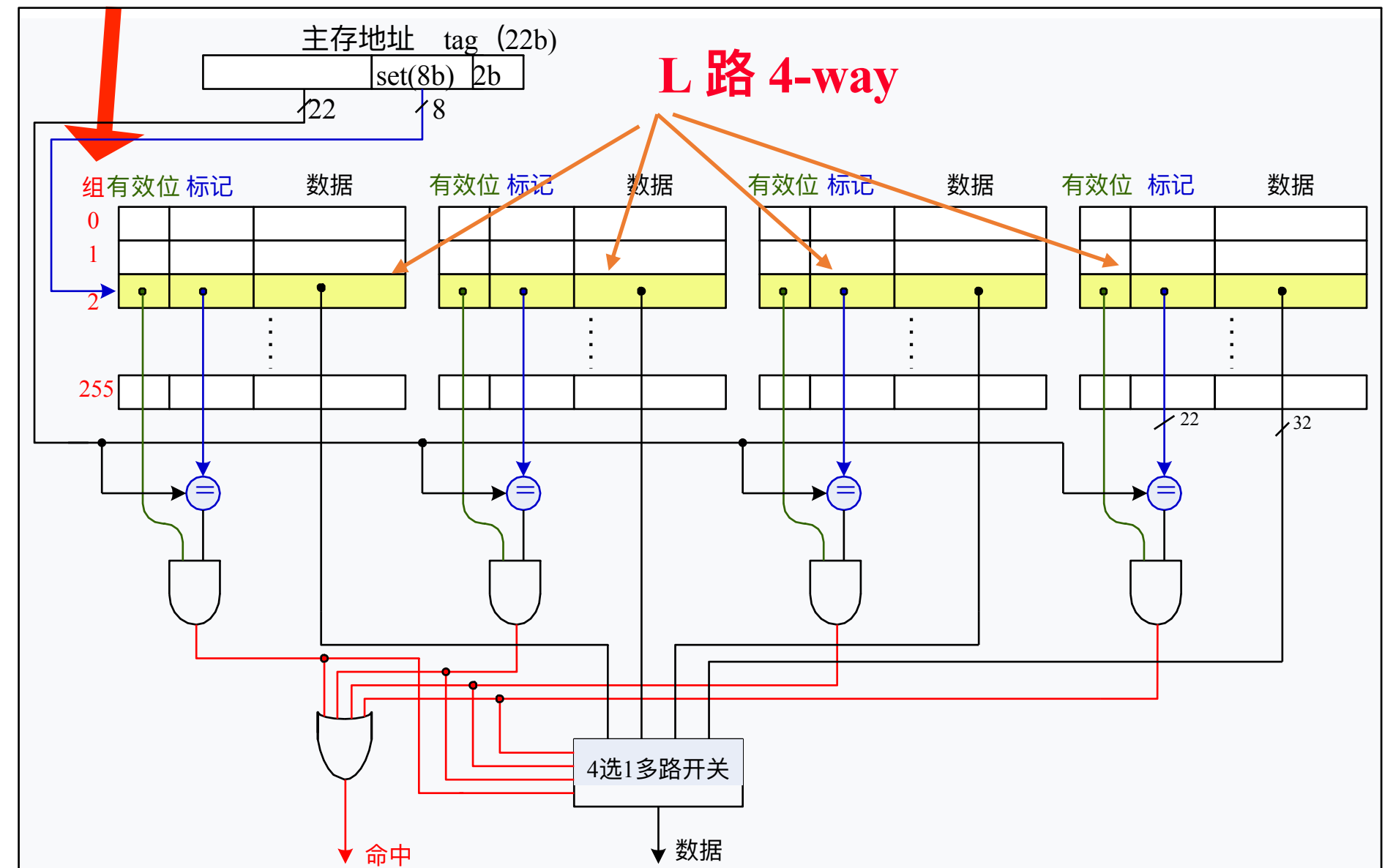
其中i,j的含义与直接映射中的含义一致。上述表达式意思即为某一主存J块按模K将其映射到缓存的第i组(set)内。
 $i = \text{Cache set地址}$

Cache与主存之间的映射 — 组相联 (Set Associative Mapping)

■ Cache结构示例：Cache容量**4KB**，**4路**组相联，数据块大小**4B**，主存地址32位。

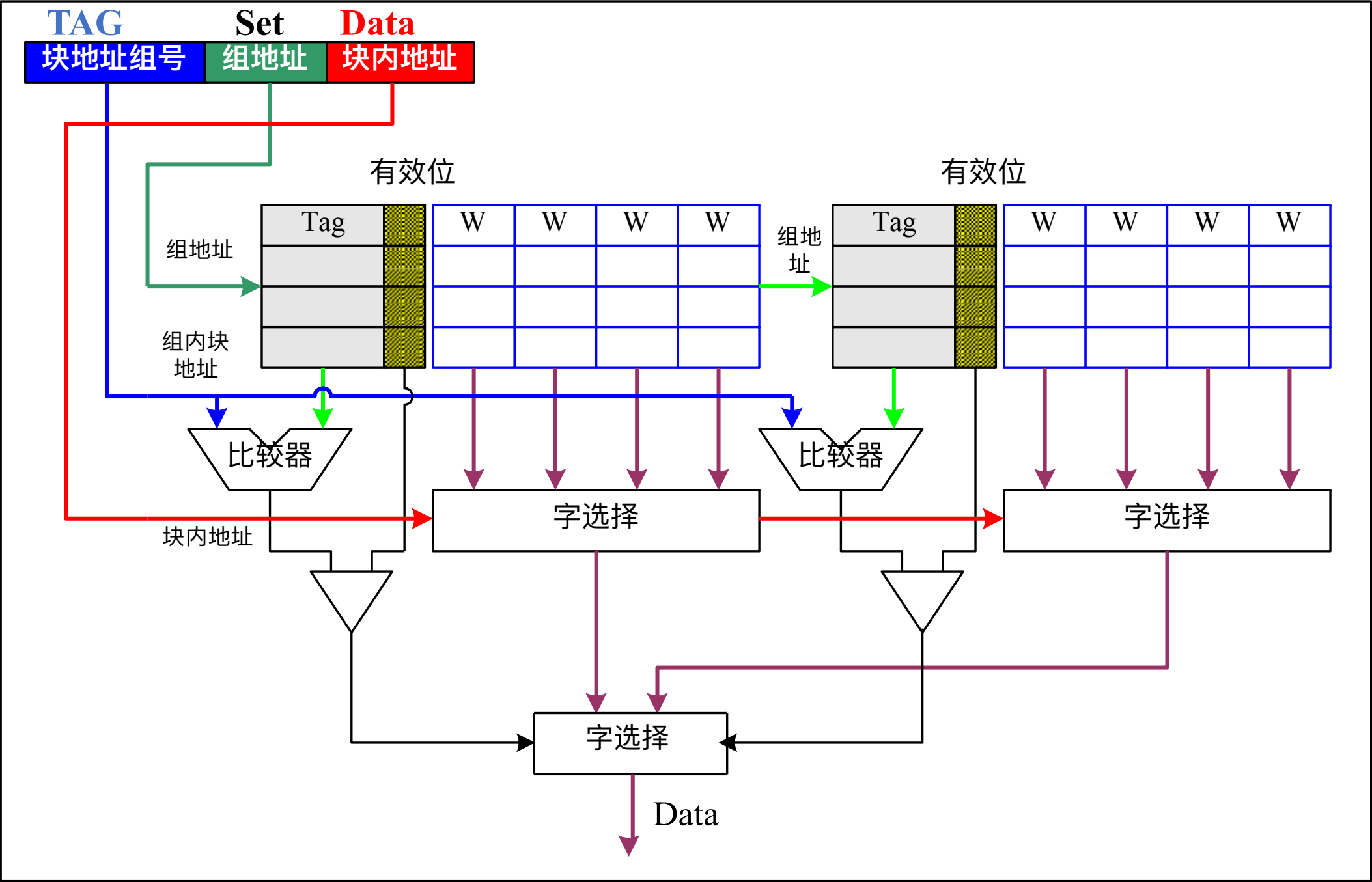
- Cache行数 = $2^{12} \div (2^2 \times 2^2) = 256$ 行(set)
- Cache容量4KB，4路组相联，每一路大小**1KB**。
- 主存地址：32 位，其中高22位为块组地址，2-10为cache行地址8位(set or index),低2位为数据offset.
- Cache的Tag应该为22 位， $22 = 32 - (8 + 2) = 22$ 。

K 行 256-set



Cache与主存之间的映射 — 组相联

❖ Cache举例： 2路组相联Cache的地址机构

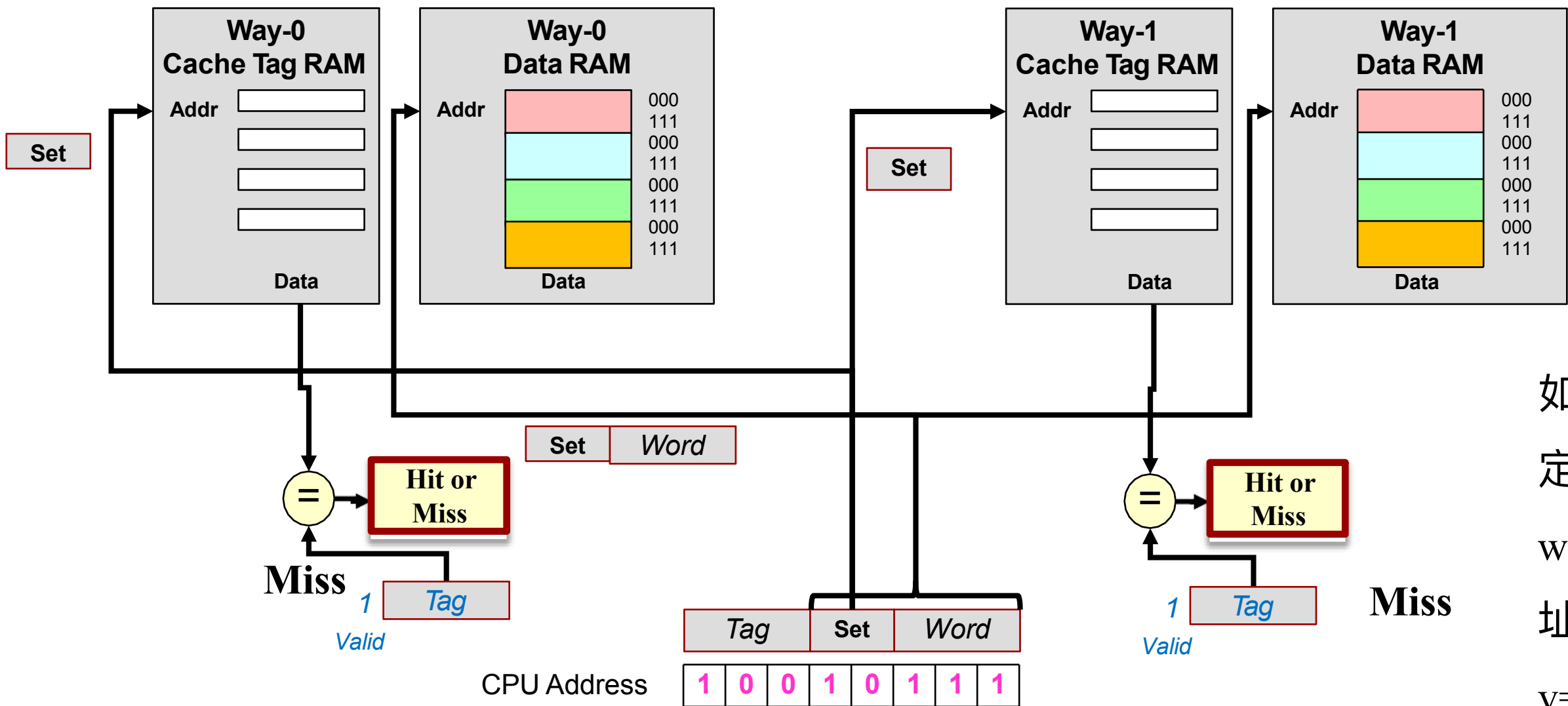


Set-Associative Datapath

Cache 分成4行 (set) , 每个set分成2路
主存块J可以映射到Cache的I行 (set) 内, 但可以是2-way内的任意一路。

MM = 256 words=2^8 words
Block size=8 words
N=8 Total Cache Blocks 4 Sets with 2-ways each

	V	Tag	Cache	Word
Set 0	1	1 0 1	Data Way 0	000 111
	0	1 0 1	Data Way 1	000 111
Set 1	1	0 0 1	Data Way 0	000 111
	1	1 1 0	Data Way 1	000 111
Set 2	1	1 1 1	Data Way 0	000 111
	0	1 0 1	Data Way 1	000 111
Set 3	0	1 0 1	Data Way 0	000 111
	1	0 1 0	Data Way 1	000 111



如图, 根据CPU地址, set=10, 定位于set2, 蓝色部分, way0,v=1,Tag=111, 与内存地址的100不等, Miss, way1的v=0, Miss

K-Way Set Associative Mapping

- 如果 80386 采用 K-Way 组相联映射:
 - MM = 4GB (2^{32}), Cache Size = 64KB (2^{16}), Block Size = ($16=2^4$) bytes = 4 words
- 主存中的块数 MM = $2^{32} / 2^4 = 2^{28}$
- 缓存行数 = $2^{16} / 2^4 = 2^{12} = 4096$
- Set Associativity/Ways (K) = 2 Blocks/Set ,每行2-way
 - 缓存分成2-way,则 set 行数为 $\Rightarrow 2^{12}/2 = 2^{11}$ 行索引|Sets $\Rightarrow 11$ bits
- $2^{28} / 2^{11} = 2^{17} = 128K$ (内存组数)
 - 17 Tag Field Bits

Tag	Set	Word	Byte
17	11	2	2
Block ID=28			

Tag RAM Organizations

- 80386 2-Way Set-Associative Cache Organization

Q2: Is it there?

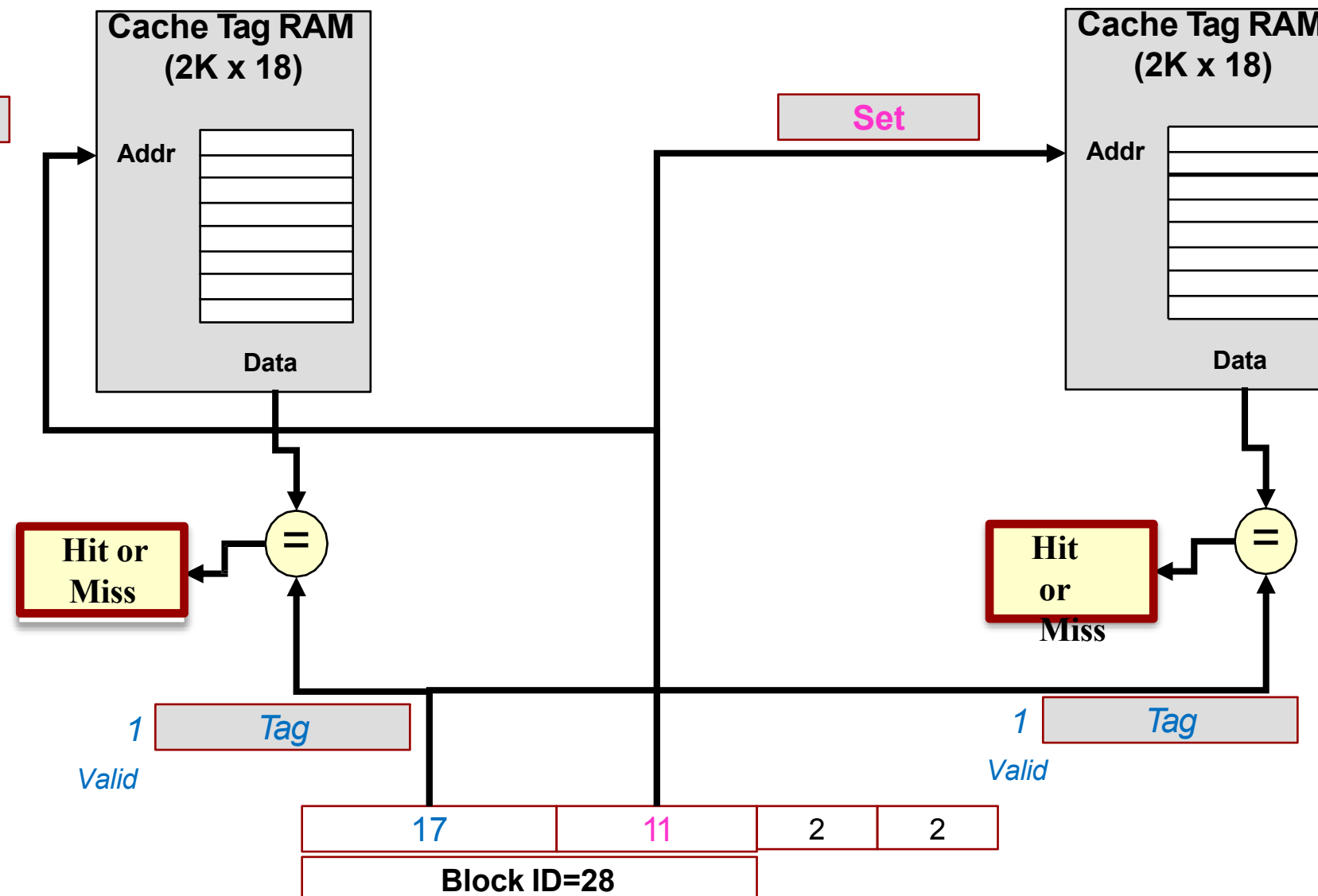
将映射到缓存set处的所有way的标记

(tags) 与内存高 17 位地址位加有效位

进行比较，以判断内存块是否在缓存中。

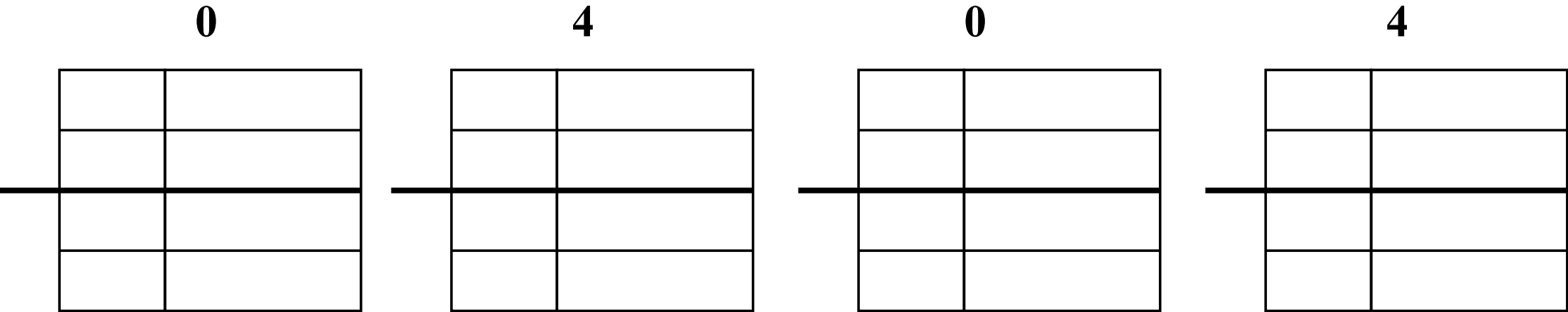
Q1: How do we find it?

使用内存地址粉色的11位index- 索引 - 来确定哪个缓存块.



Another Reference String Mapping

- 考虑一个2-way,2 set 的 (4-block)空的缓存, 所有的块初始化有效位为0 (not valid) .
给定主存的字地址“0 4 0 4 0 4 0 4”进行访问, 计算缓存的命中率。



Another Reference String Mapping

- 考虑一个2-way,2 set 的 (4-block)空的缓存,所有的块初始化有效位为0 (not valid) .
给定主存的字地址“0 4 0 4 0 4 0 4”进行访问,计算缓存的命中率。

Block=1word, 字地址0, 0000=0, 映射到set=0,tag=000,

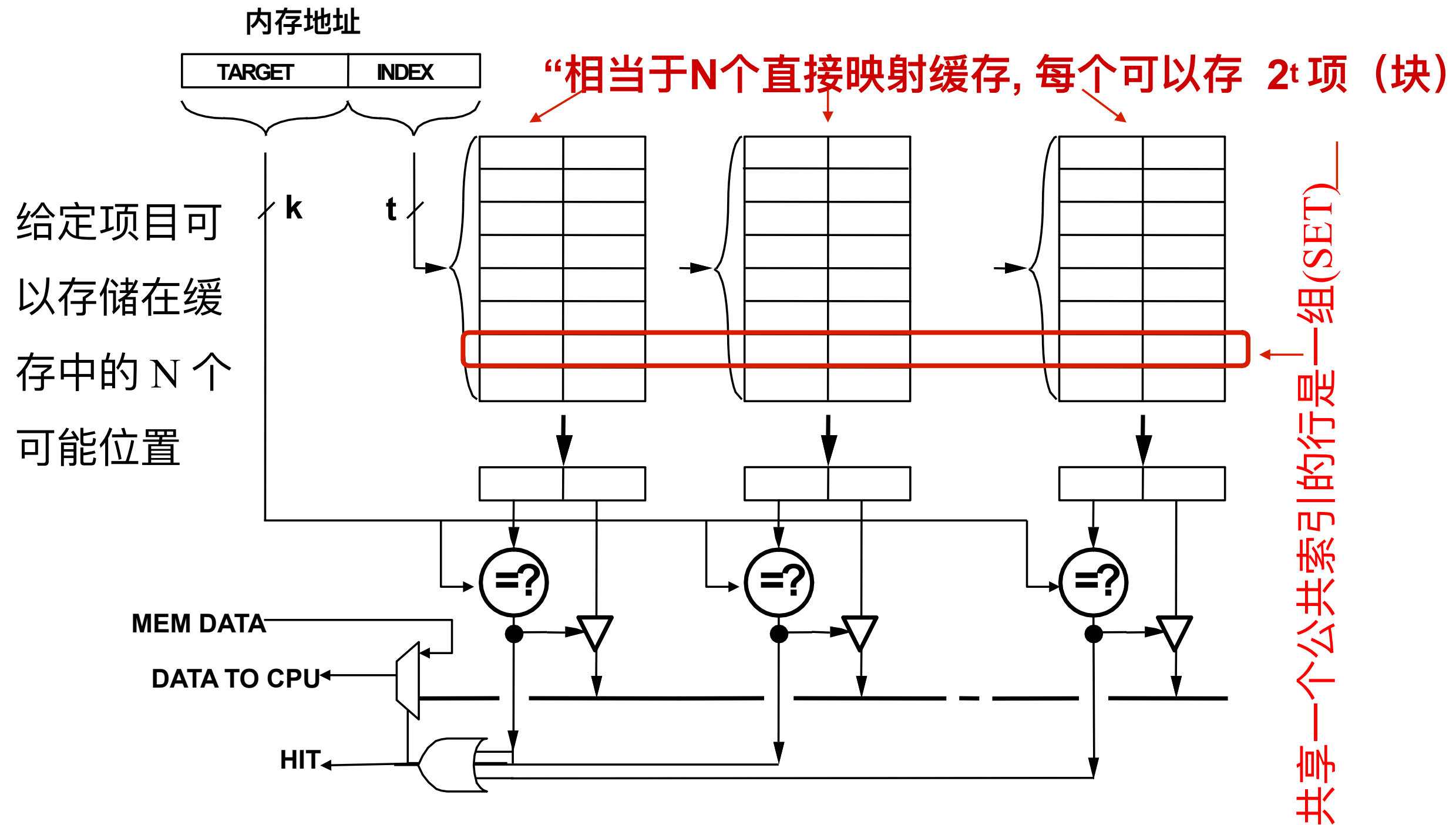
字地址4, 0100, 映射到set=0,tag=010

0 miss		4 miss		0 hit		4 hit	
000	Mem(0)	000	Mem(0)	000	Mem(0)	000	Mem(0)
		010	Mem(4)	010	Mem(4)	010	Mem(4)

| 8 requests, 2 misses

- 解决了由于冲突未命中导致的直接映射缓存中的乒乓效应,
因为现在映射到同一缓存行 (set) ,有两个位置可以存储!

N-Way Set-Associative Cache



Cache和主存之间的映射方式

► 高速缓存的缺失率和关联度

三种映射方式

- ◆ 直接映射：唯一映射(只有一个可能的位置)
- ◆ 全相联映射：任意映射(每个位置都可能)
- ◆ N-路组相联映射：N-路映射(有N个可能的位置)

什么是关联度？

主存块映射到Cache时，可能存放的位置个数

For a cache with 8 entries, 缓存大小不变, 用不同的映射方式

关联度示例

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

关联度为多少? 1

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

关联度为多少? 2

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

关联度为多少? 4

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

关联度为多少? 8

缺失率与组关联性的对比

Assume:有三个小缓存，每个缓存由四个单字块组成。
一个采用直接映射，一个采用2-way组相联映射，第三个采用全相联映射。

Question:给定以下块地址序列：0,8,0,6,8，求每个缓存组织的未命中数。

直接映射, Block=1word, 字地址0, 0000=0, 映射到set=0,tag=00,
字地址8, 1000, 映射到set=0,tag=10
字地址6, 0110, 映射到set=10,tag=01

Answer: for direct-mapped
5 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0	Set 1	Set 2	Set 3
0	Miss	M[0]			
8	Miss	M[8]			
0	Miss	M[0]			
6	Miss	M[0]		M[6]	
8	Miss	M[8]		M[6]	

2-way组相联映射

000	Mem(0)

Block=1word, 字地址0, 0000=0, 映射到set=0,tag=000,
字地址8, 1000, 映射到set=0,tag=100
字地址6, 0110, 映射到set=0,tag=011

给定以下块地址序列： 0,8,0,6,8

4 misses

0 刚访问，替
换最久的 8

Memory block	Hit or miss	Contents after each reference			
		Set 0		Set 1	
		Way 0	Way1	Way 0	Way 1
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[6]		
8	Miss	M[8]	M[6]		

全相联映射

Block=1 word, 可以映射到任意Block,

字地址0, 0000, 映射到Way0,tag=0000,

字地址8, 1000, 映射到Way1,tag=1000

字地址6, 0110, 映射到Way2,tag=0110

3 misses

Memory block	Hit or miss	Contents after each reference			
		Only one set			
		Way 0	Way 1	Way 2	Way 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[8]	M[6]	
8	Hit	M[0]	M[8]	M[6]	

关联性降低了多少未命中率?

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

用于 SPEC2000 基准测试的 Intrinsuty FastMATH 处理器的数据缓存未命中率从1-way到8-way。

- 64KB 数据缓存大小 16-word block

标签(Tag)大小与组 (set)关联性

Assume

Cache size is 4K Block

Block size is 4 words Physical address is 32bits

Question

找出不同关联性的set总数和Tag位总数

Answer

Offset size (Byte) = $16 = 2^4$

内存块数 block = $2^{32} \div 2^4 = 2^{28}$

缓存块数 = 2^{12}

对于直接映射, 缓存行地址(set) = 12 bits

bits of Tag = $(28-12) \times 4K = 16 \times 4K = 64 \text{ Kbits}$

4 bits 为块内地址(offset)

28 bits for 内存块地址

12 bits 缓存块数

缓存块数= 2^{12}
内存28块

For two-way 组相联

set = $2^{12} \div 2 = 2^{11}$ Bits of
index = $12-1=11$ bits tag= $32- (11+2+2) =17$
Bits of Tag = $(28-11) \times 2 \times 2K=17 \times 2 \times 2K=68$ Kbits

For four-way组相联

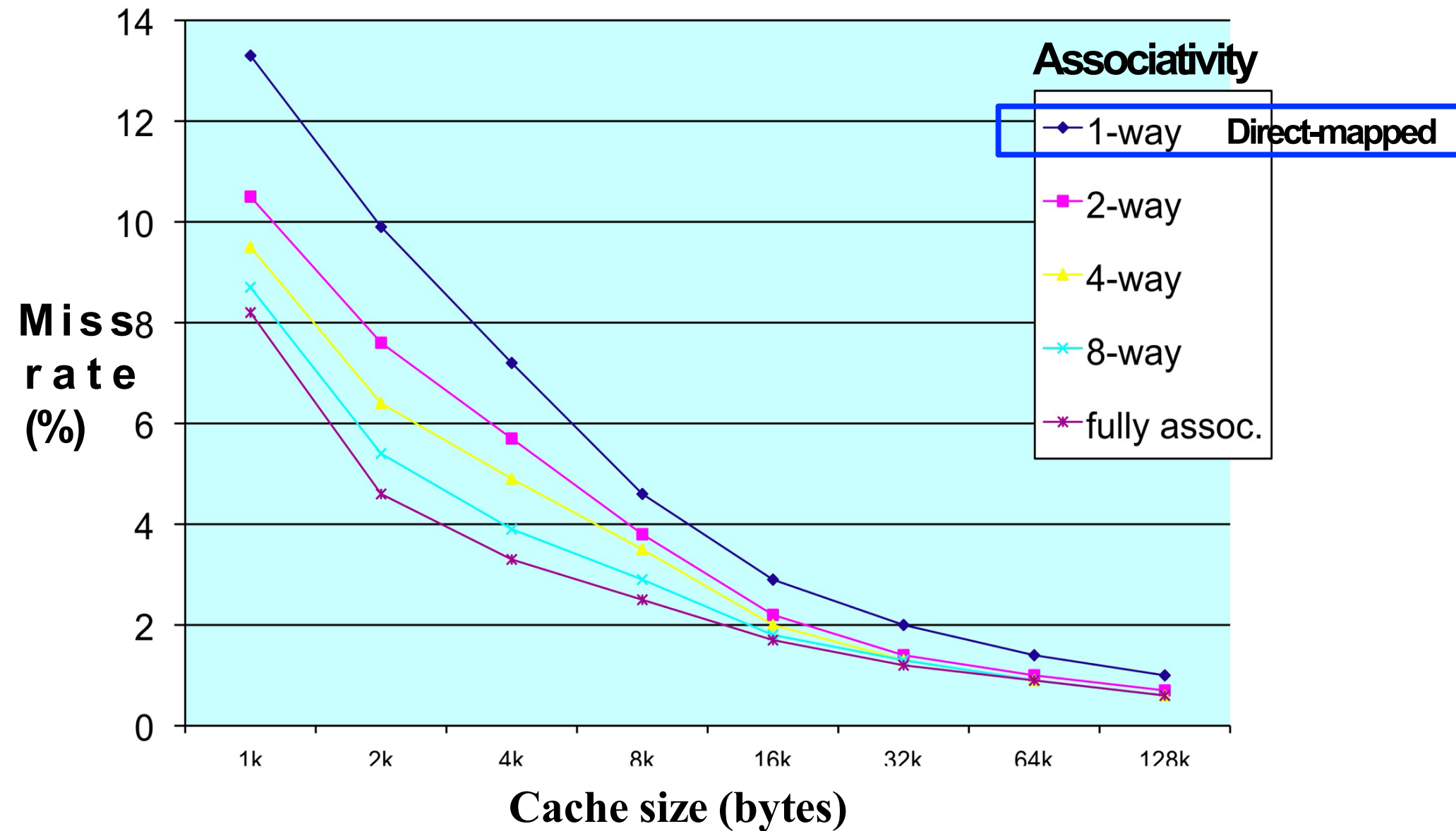
Number of cache set = $2^{12} \div 4 = 2^{10}$ Bits of index = $12-2=10$ bits
tag= $32- (10+2+2) =18$
Bits of Tag = $(28-10) \times 4 \times 1K=18 \times 4 \times 1K=72$ Kbits

For 全相联

Number of cache set = $2^{12} \div 2^{12} = 2^0$ Bits of index = $12-12=0$ bits
Bits of Tag = $(28-0) \times 4K \times 1=128$ Kbits

	Direct	2-way	4-way	Fully
Index(bit)	12	11	10	0
Tag(bit)	16	17	18	28

Associativity vs. Miss Rate



- 8路（几乎）与全关联的一样有效
- 经验法则：N-byte M-way set assoc \approx N/2-byte 2M-way set assoc。

例题：标准的32位字节编址MIPS机器 4 GiB RAM内存, 有一个 4-way 组相联缓存，每个缓存块可以存 32 字节数据, 容量 16 KiB. 多少位用于标识、索引和偏移 tag, index, and offset?

以字节编址，每个block放32B， $16\text{KiB}/32\text{B}=512$ Blocks, $512\text{B}/4=128$ sets(7位index)

32B的offset为5， $32-(7+5)=20$ ，tag为20位。

Tag	Index	Offset
20	7	5

如果改为8ways, How many bits are used for the tag, index, and offset?

$512\text{B}/8=64$ sets,则cache的index为6位，其他不变， $32-(6+5)=21$

Tag	Index	Offset
21	6	5

- 缓存一块存 8B , 缓存容量 64B. 你不知道关联度! 给定以字编址 (4B) 的程序访问的缺失情况, 确定关联度是多少?
- 0 (MISS), 1 (HIT), 2 (MISS), 15 (MISS), 17 (MISS), 0 (HIT), 32 (MISS), 1 (MISS)
 - Direct- -mapped
 - 2- -way
 - 4- -way
 - Fully associative (8- -way)
 - Cannot be determined from the sequence above

Answer: 直接映射64B/8B=8行,offset=1+2=4, 内存以字地址访问,index3位, offset 3位 2=0010XX(index=001B,tag=0) 15=1111XX(index=111B,tag=0) 17=10001XX(index=000B,tag=1) 32=0010 0000 000 0XX (index=000B,tag=0x20)

如果直接映射, 如下表, 与给定的情况不符

Block address	Tag	Cache index	Hit/miss	Cache content after access	
				0	1
0	0	0	miss	Mem[0]	Mem[1]
1	0	0	hit		
2	0	1	miss	Mem[2]	Mem[3]
15	0	7	miss	Mem[15]	Mem[16]
17	1	0	miss	Mem[17]	Mem[18]
0	0	0	miss	Mem[0]	Mem[1]
32	0x20	0	miss	Mem[32]	Mem[33]
1	0	0	miss		

如果2-way, 如下表, 与给定的情况一致

Block address	Cache index	Hit/miss	Cache content after access			
			way 0		way1	
0	0	miss	Mem[0]	Mem[1]		
1	0	hit				
2	1	miss	Mem[2]	Mem[3]		
15	3	miss	Mem[15]	Mem[16]		
17	0	miss	Mem[0]	Mem[1]	Mem[17]	Mem[18]
0	0	hit				
32	0	miss	Mem[32]	Mem[33]	Mem[17]	Mem[18]
1		miss				

Answer: 2-way组相联

64B/8B=8, 8/2=4行,offset=1+2=3, 内存以字地址访问,index2位, offset 3位

2=0010XX(index=01B,tag=0)

15=1111XX(index=11B,tag=1)

17=10001XX(index=00B,tag=10)

32=0010 0000 000 0XX (index=00B,tag=0x40)

CACHE的其它问题 —— Cache容量的计算

■ 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。

■ 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，那么Cache总共有多少位？

- Cache每数据块大小： $4 \times 32 = 128 \text{ bits} = 2^4 \text{ Bytes}$ ；（块内地址）
- Cache块数： $16\text{KB} \div 2^4 \text{ B} = 2^{10} \text{ 块}$ ；（区内块地址）
- tag位数： $32 - 10 - 4 = 18 \text{ bits}$ （区地址）
- 有效位： 1 bit
- Cache实际总容量： $2^{10} \times (128 + 18 + 1) = 147\text{Kbit} \approx 18.4\text{KByte}$

Cache失效原因 3C's

(1) 强制性失效(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，这就是强制性失效。（冷启动失效，首次访问失效。）

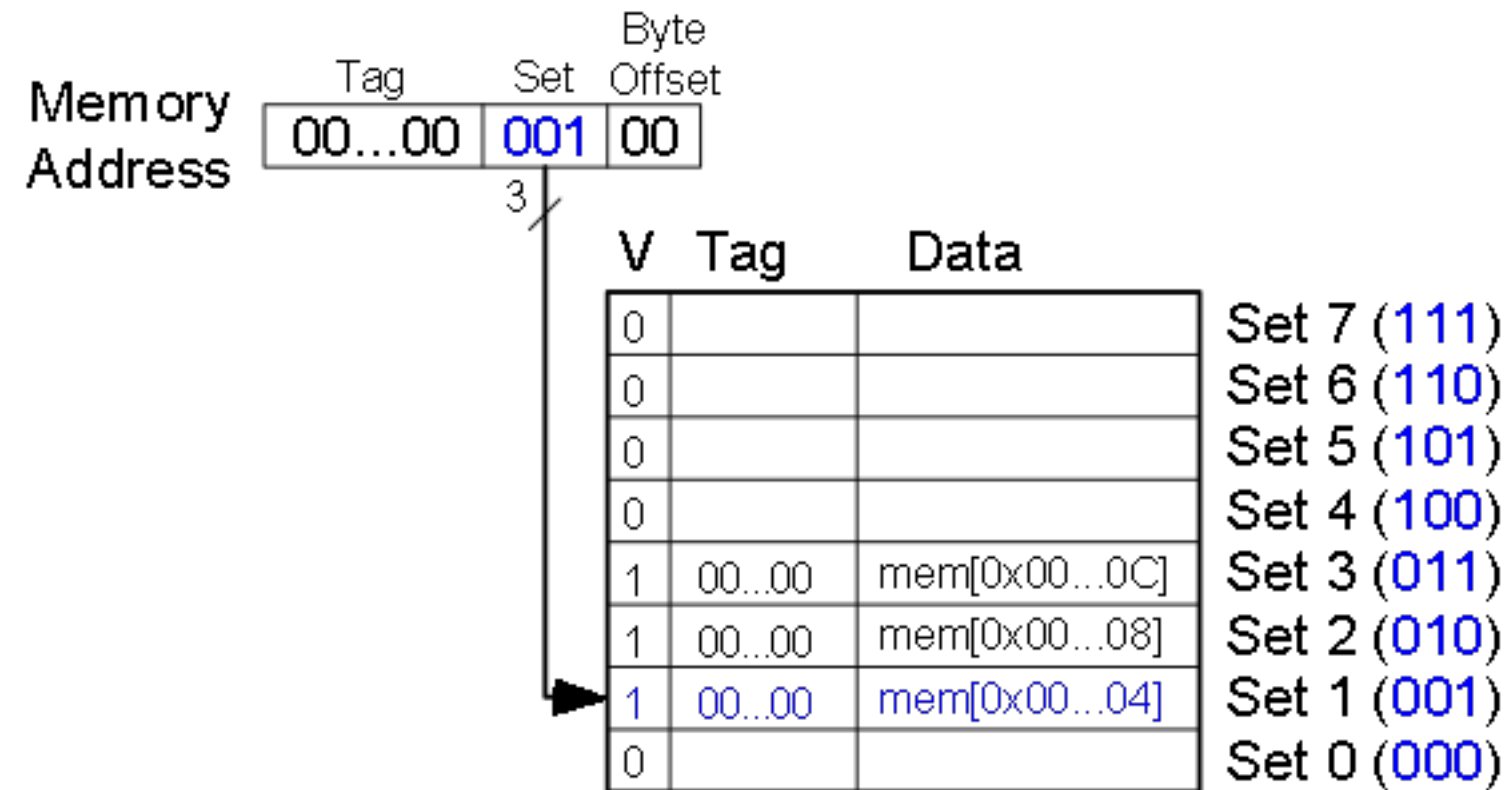
(2) 容量失效(Capacity miss)

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，就会发生失效。这种失效称为容量失效。

(3) 冲突失效(Conflict miss) 提高相联度

在组相联或直接映象Cache中，若太多的块映象到同一组(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。这就是发生了冲突失效。(碰撞失效，干扰失效)

强制性失效(Compulsory miss)



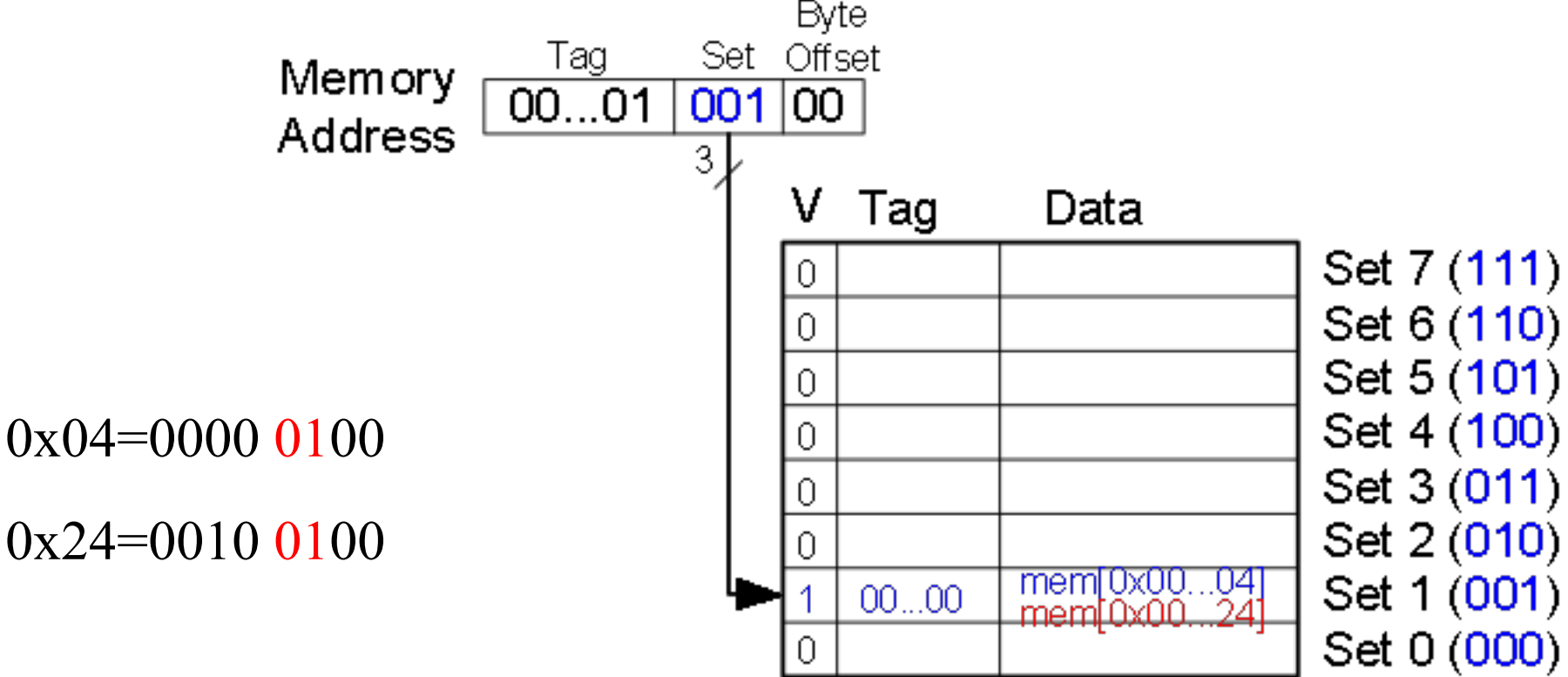
MIPS assembly code

```
    addi $t0, $0, 5
loop: beq  $t0, $0, done
      lw   $t1, 0x4($0)
      lw   $t2, 0xC($0)
      lw   $t3, 0x8($0)
      addi $t0, $t0, -1
      j    loop
done:
```

$$\text{Miss Rate} = 3/15 = 20\%$$

第一次访问缓存，块都不在缓存中，3次缺失，放入缓存后，后面的访问都在缓存中，一共循环5次，每次访问数据内存3次，一共15次
Compulsory Misses

Direct Mapped Cache: Conflict 冲突失效



```
# MIPS assembly code
loop:    addi $t0, $0, 5
        beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 10/10
= 100%

Conflict Misses

Example 内存32位字节编址访问下表所示的缓存，哪些位用来索引缓存行？

由于每块存8个字节数据，因此offset位有3位，所以4th和5th位作为行缓存索引。

按照下表地址进行内存访问，确认命中（H）还是缺失（M）并分类缺失类型, 替换用R表示.

3C’s of Caches。

Cache 4行,index=2bits
Offset=3bits,store 8 bytes

CPU Cache	Index Number	Offset							
		7	6	5	4	3	2	1	0
	0								
	1								
	2								
	3								

	Adress	Binary address	Index	Tag	M/R		Types of cache misses
1	0x00000004	00100	0	0	M	M[0]-M[7]	Compulsory
2	0x00000005	00101	0	0	H		H
3	0x00000068	1101000	1	3	M	M[68]-M[6F]	Compulsory
4	0x000000C8	11001000	1	6	M/R	M[C8]-M[CF]	Compulsory
5	0x00000068	1101000	1	3	M/R	M[68]-M[6F]	Conflict
6	0x000000DD	11011101	3	6	M	M[D8]-M[DF]	Compulsory
7	0x00000045	1000101	0	2	M/R	M[40]-M[47]	Compulsory
8	0x00000004	00100	0	0	M/R	M[0]-M[7]	Capacity
9	0x000000C9	11001001	1	6	M/R	M[C8]-M[CF]	Capacity

组相联冲突造成缺失

2-way 缓存Size 64KB, 64B block, 32-bit address.
Cache performance for the following code?

```
int  a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
{
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
}
```



缓存set=512

	address	Adress	tag	index	?
load a[0]	0x20000	0010 0000 0000 00 000000	0x4	0	compulsory misse
load b[0]	0x30000	0011 0000 0000 00 000000	0x6	0	compulsory miss
load c[0]	0x10000	0001 0000 0000 00 000000	0x2	0	compulsory miss, evict 0x4
load a[1]	0x20004	0010 0000 0000 00 000100	0x4	0	conflict miss, evict 0x6
load b[1]	0x30004	0011 0000 0000 00 000100	0x6	0	conflict miss, evict 0x2
load c[1]	0x10004	0001 0000 0000 00 000100	0x2	0	conflict miss, evict 0x4

100% miss rate due to conflict miss!

关联度增加? !

组相联冲突造成缺失

4-way 缓存Size 64KB, 64B block, 32-bit address.
Cache performance for the following code?

```
int  a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512;      i++)
{
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
}
```



缓存set=256

	address	Adress	tag	index	?
load a[0]	0x20000	0010 0000 0000 00 000000	0x8	0	compulsory misse
load b[0]	0x30000	0011 0000 0000 00 000000	0xc	0	compulsory miss
load c[0]	0x10000	0001 0000 0000 00 000000	0x4	0	compulsory miss
load a[1]	0x20004	0010 0000 0000 00 000100	0x8	0	hit
load b[1]	0x30004	0011 0000 0000 00 000100	0xc	0	hit
load c[1]	0x10004	0001 0000 0000 00 000100	0x4	0	hit

50% miss rate due to conflict miss!

Cache的缺失损失

■ 缺失损失

- CPU访问Cache缺失时，导致一次流水线阻塞（pipeline Stall，可以理解为停顿），CPU必须等待数据装入Cache后才能访问Cache，这期间的损失时间称为缺失损失。
- 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 Misses

指令Cache缺失的处理步骤

- 将PC - 4送到存储器中
- 通知主存执行读操作，并等待主存访问完成
- 写Cache项，并设置Cache相关标志
- 重新取指令，此时指令在Cache中

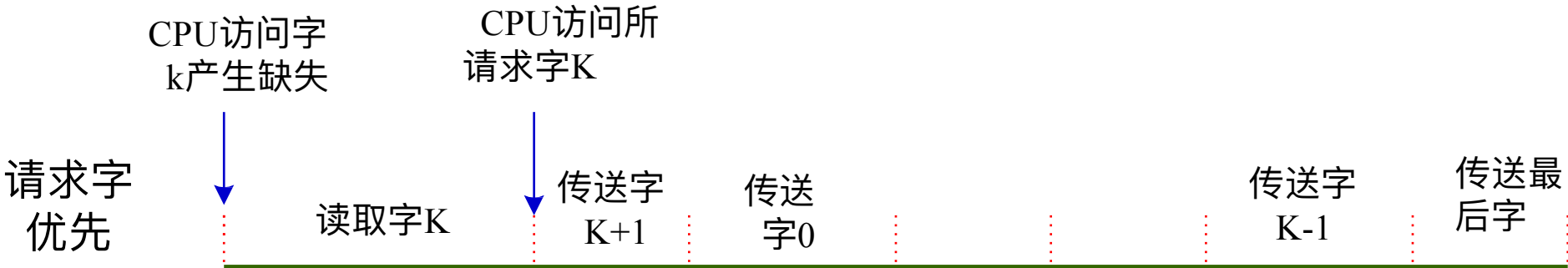
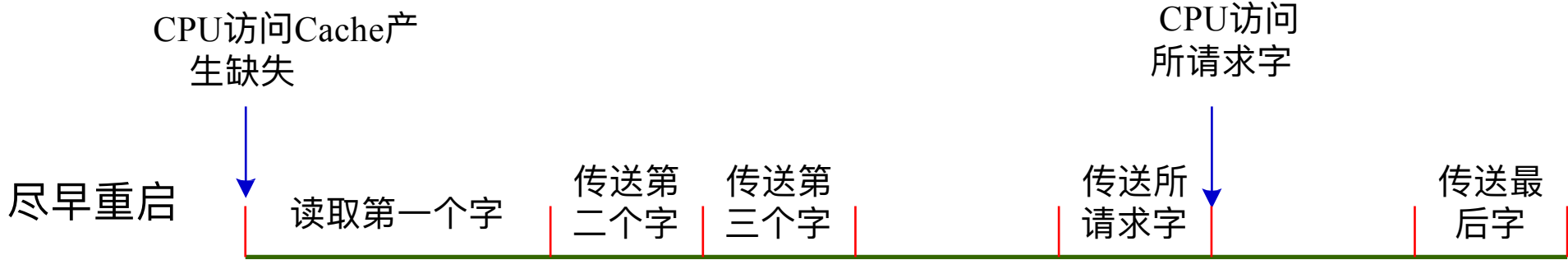
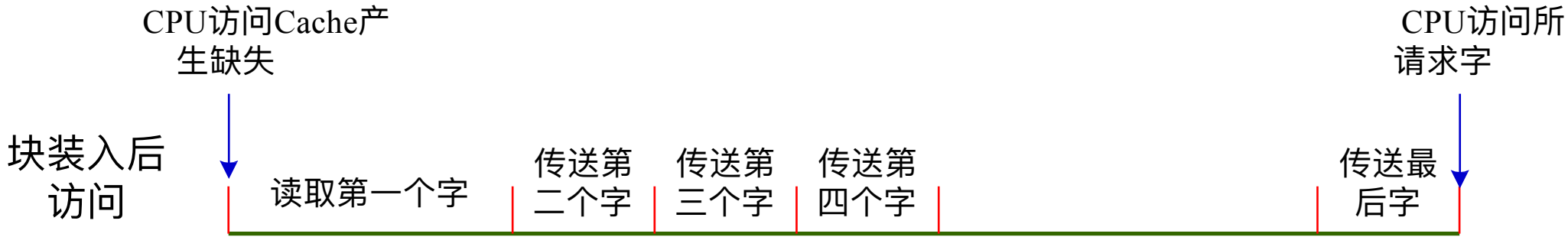
Cache的替换策略——Cache的缺失处理

■ 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的替换策略——Cache的缺失处理

■ 几种缺失处理方式



Cache的替换策略 —— Cache块的替换

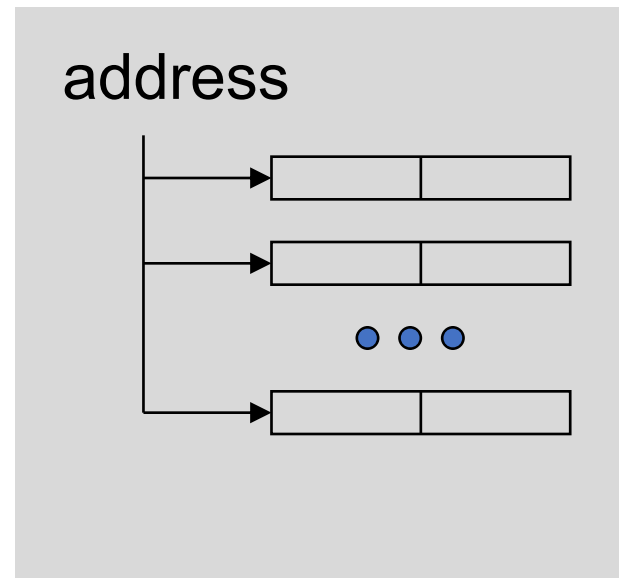
■ 替换块的选择范围

- 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；
- 直接映像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。

缺失时三种结构的处理方式

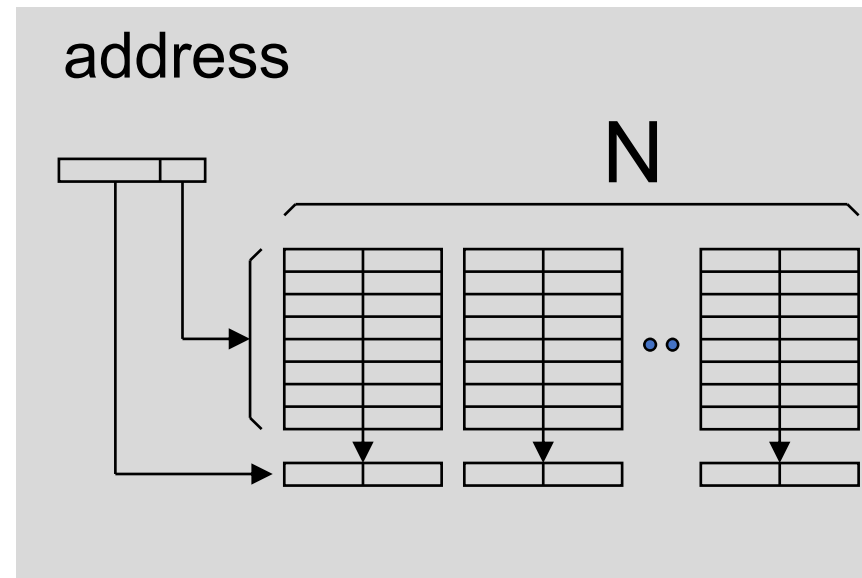
替换策略?

← Fully associative N-way set associative Direct-mapped →



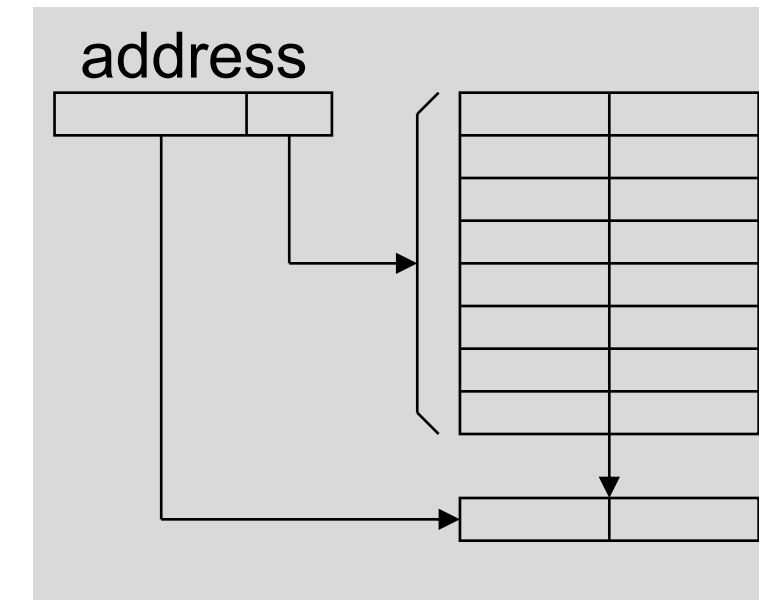
地址A可能映射到缓存任意行中，addr 需与缓存中所有的Tags同时做比较。

全相联从整个缓存中找到一个条目将其逐出



与缓存中N个Tags同时做比较。数据可能存于缓存对应行(set)的N个位置的任意一个。

组相联在某一行 (set) , 从 (way) N项中找到一个条目将其逐出



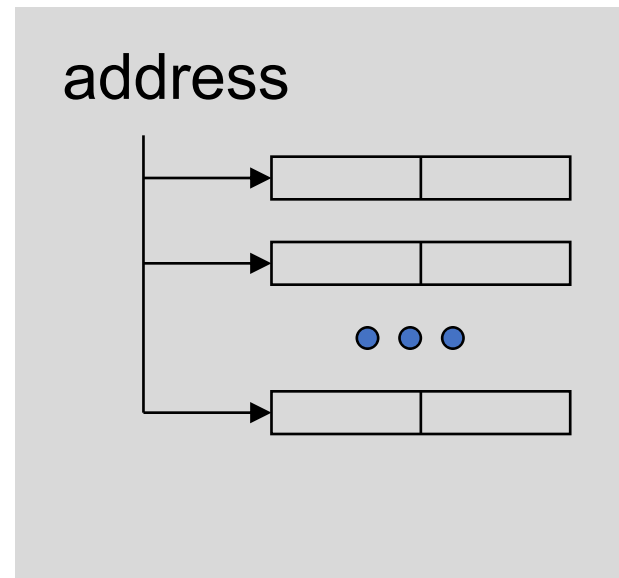
addr 只与一个Tag比较，地址A只能映射到缓存的一个位置。

直接映射只有一个位置可以替换

缺失时三种结构的处理方式

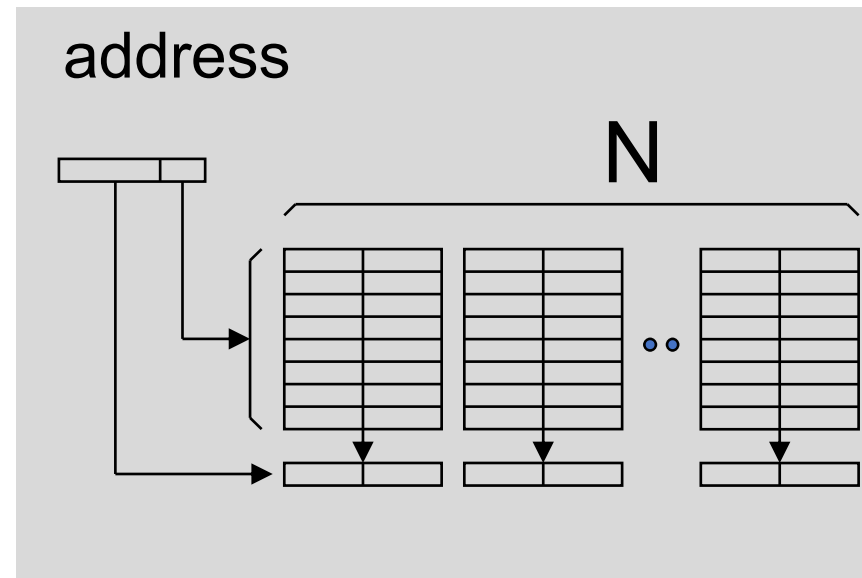
替换策略?

← Fully associative N-way set associative Direct-mapped →



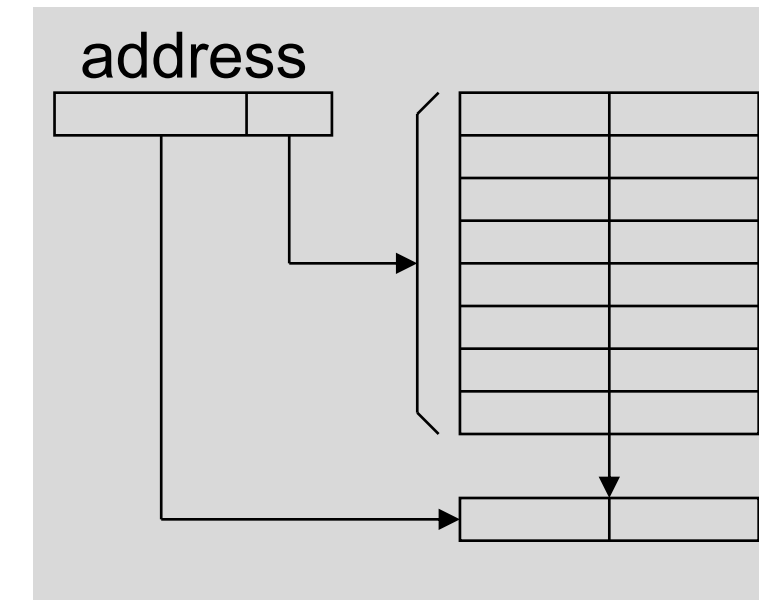
地址A可能映射到缓存任意行中，addr 需与缓存中所有的Tags同时做比较。

全相联从整个缓存中找到一个条目将其逐出



与缓存中N个Tags同时做比较。数据可能存于缓存对应行(set)的N个位置的任意一个。

组相联在某一行 (set) ，从 (way) N项中找到一个条目将其逐出



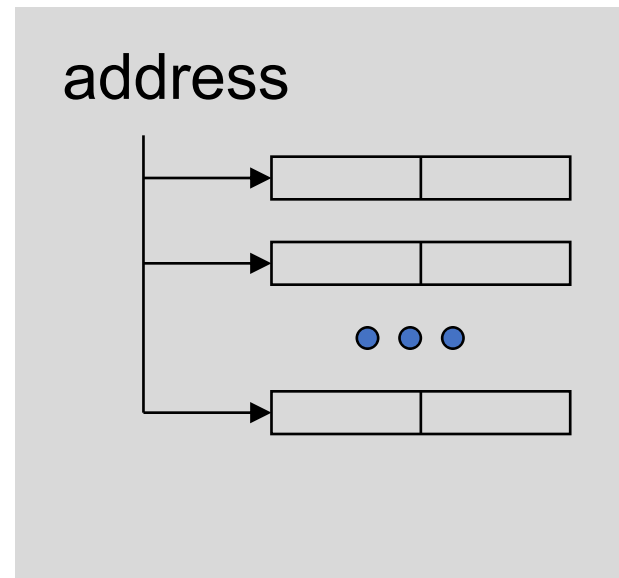
addr 只与一个Tag比较，地址A只能映射到缓存的一个位置。

直接映射只有一个位置可以替换

缺失时三种结构的处理方式

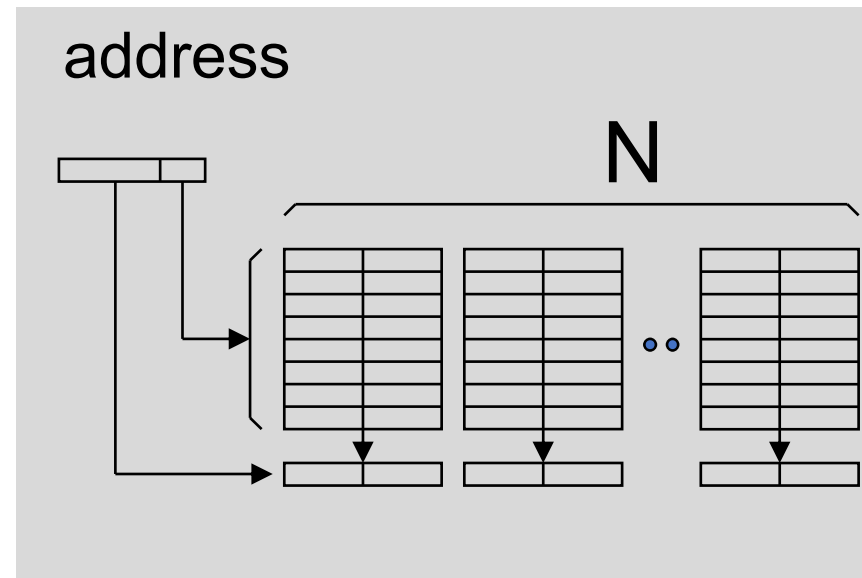
替换策略?

← Fully associative N-way set associative Direct-mapped →



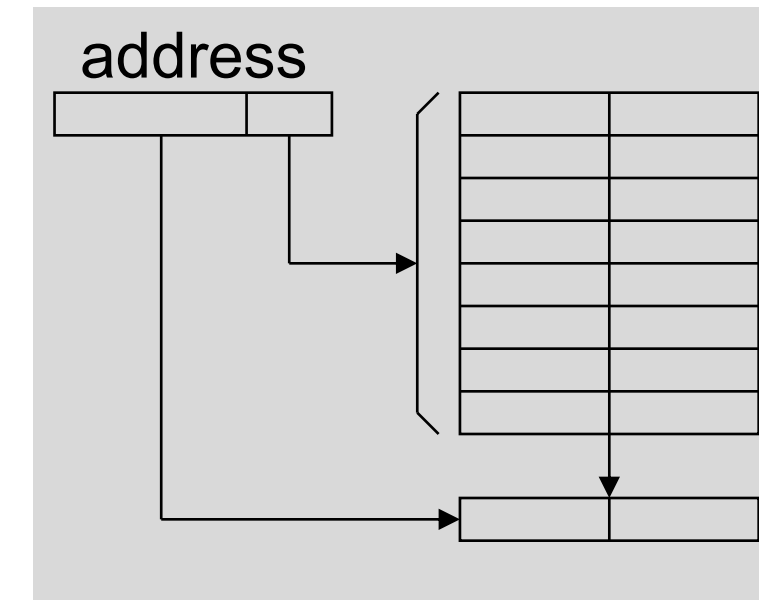
地址A可能映射到缓存任意行中，addr 需与缓存中所有的Tags同时做比较。

全相联从整个缓存中找到一个条目将其逐出



与缓存中N个Tags同时做比较。数据可能存于缓存对应行(set)的N个位置的任意一个。

组相联在某一行 (set) ，从 (way) N项中找到一个条目将其逐出



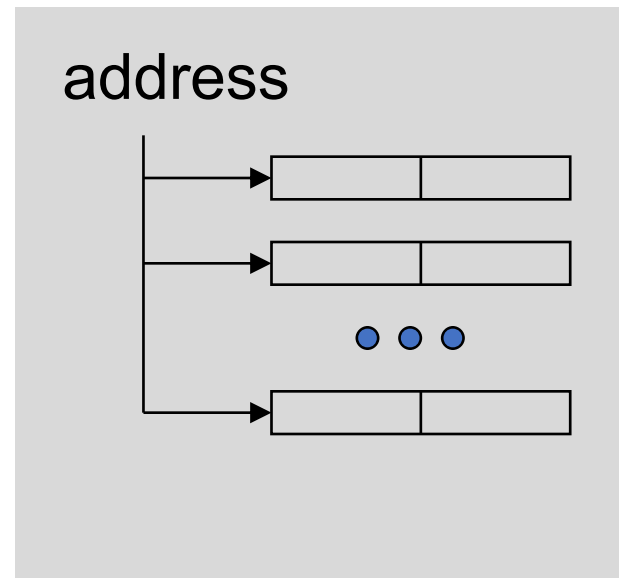
addr 只与一个Tag比较，地址A只能映射到缓存的一个位置。

直接映射只有一个位置可以替换

缺失时三种结构的处理方式

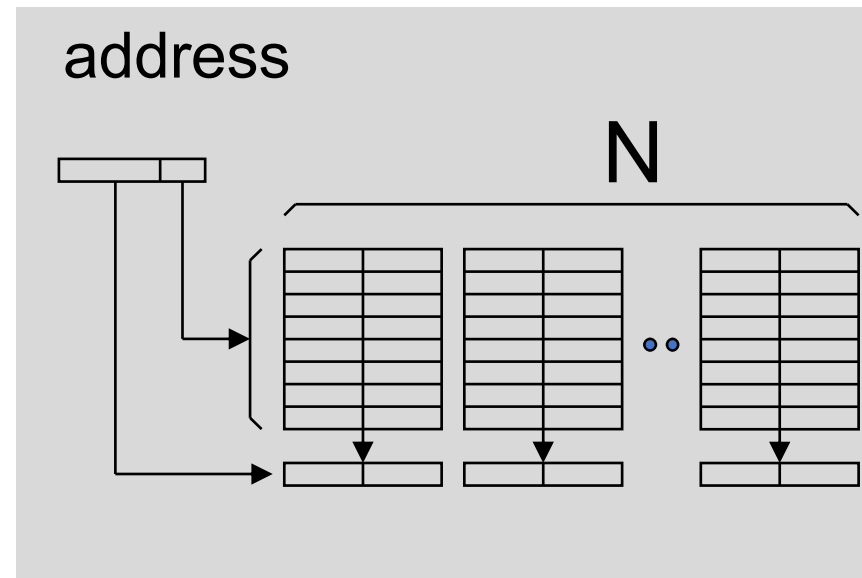
替换策略?

← Fully associative N-way set associative Direct-mapped →



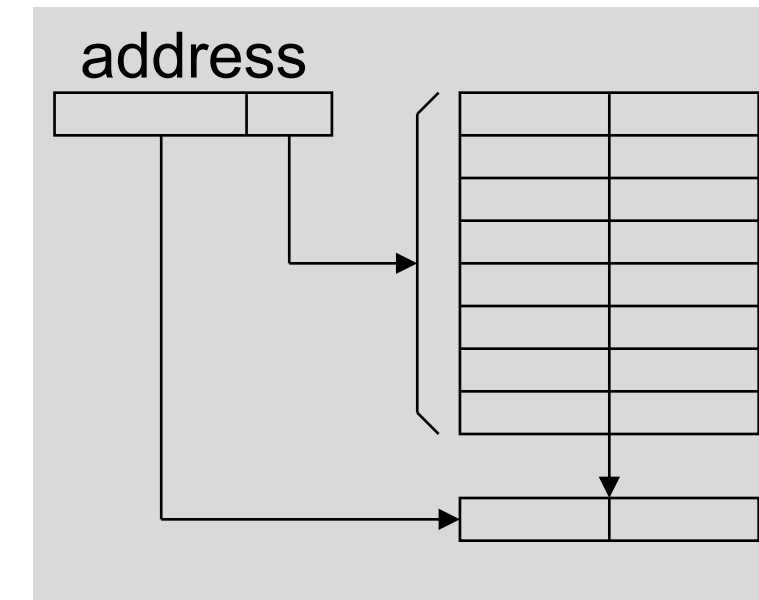
地址A可能映射到缓存任意行中，addr 需与缓存中所有的Tags同时做比较。

全相联从整个缓存中找到一个条目将其逐出



与缓存中N个Tags同时做比较。数据可能存于缓存对应行(set)的N个位置的任意一个。

组相联在某一行 (set) , 从 (way) N项中找到一个条目将其逐出



addr 只与一个Tag比较，地址A只能映射到缓存的一个位置。

直接映射只有一个位置可以替换