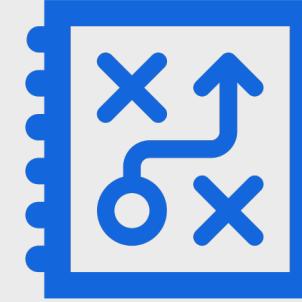


2. 数据的表示与存储

2.9 高速缓存Cache



2. 数据的表示与存储

2.9 高速缓存

01 局部性概念

02 Cache与主存之间的映射方式

03 Cache中主存块的替换算法

04 Cache写操作策略



2. 数据的表示与存储

2.9.1 局部性原理

时间局部性(temporal locality): 不久还会使用

空间局部性(spatial locality): 一会儿还会访问

```
2 #define N 8
3 int sumvec(int v[N])
4 {
5     int i, sum = 0;
6     for(i = 0; i < N; i++)
7         sum += v[i];
8     return sum;
9 }
```

地址	0	4	8	12	16	20	24	28
内容	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
访问顺序	1	2	3	4	5	6	7	8



2. 数据的表示与存储

2.9.1 局部性原理

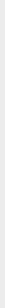
```
11 int sumarrayrows(int a[M][N])
12 {
13     int i, j, sum = 0; 0,1
14     for(i = 0; i < M; i++)
15         for(j = 0; j < N; j++)
16             sum += a[i][j];
17     return sum;
18 }
```

0 → 1 2
↓ ↓ ↓

```
19 int sumarraycols(int a[M][N])
20 {
21     int i, j, sum = 0;
22     for(j = 0; j < N; j++)
23         for(i = 0; i < M; i++)
24             sum += a[i][j];
25     return sum;
26 }
```

地址	0	4	8	12	16	20
内容	a ₀₀	a ₀₁	a ₀₂	a ₁₀	a ₁₁	a ₁₂
访问顺序	1	2	3	4	5	6

地址	0	4	8	12	16	20
内容	a ₀₀	a ₀₁	a ₀₂	a ₁₀	a ₁₁	a ₁₂
访问顺序	1	3	5	2	4	6



2. 数据的表示与存储

2.9.1 局部性原理

```
1 sumvec:  
2     sub    sp, sp, #16  
3     str    x0, [sp, #8]  
4     str    wzr, [sp]  
5     str    wzr, [sp, #4]  
6     b      .LBB0_1  
7 .LBB0_1:  
8     ldr    w8, [sp, #4]  
9     subs   w8, w8, #8  
10    b.ge   .LBB0_4  
11    b      .LBB0_2  
12 .LBB0_2:  
13    ldr    x8, [sp, #8]  
14    ldrsw  x9, [sp, #4]  
15    ldr    w9, [x8, x9, lsl #2]  
16    ldr    w8, [sp]  
17    add    w8, w8, w9  
18    str    w8, [sp]  
19    b      .LBB0_3  
20 .LBB0_3:  
21    ldr    w8, [sp, #4]  
22    add    w8, w8, #1  
23    str    w8, [sp, #4]  
24    b      .LBB0_1
```



2. 数据的表示与存储

2.9.1 局部性小结

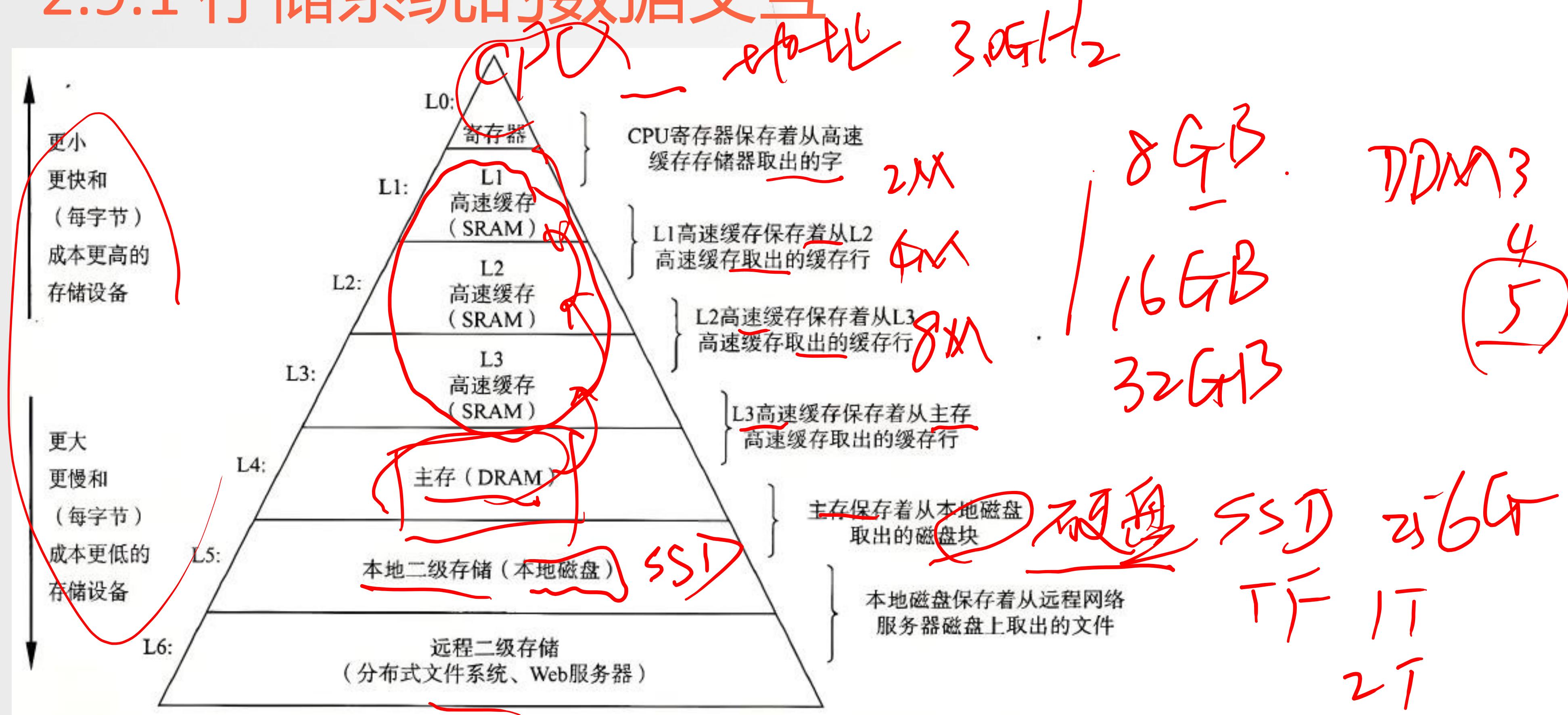
- 1、重复引用相同变量的程序有良好的时间局部性。
- 2、连续内存访问时，步长越小，空间局部性越好。
- 3、对于指令来说，循环有好的时间和空间局部性。
- 4、循环体越小，循环次数越多，局部性就会越好。

sum
0, 1, 2, 3
0, 4, 8, 12
0, 20, 30.



2. 数据的表示与存储

2.9.1 存储系统的数据交互

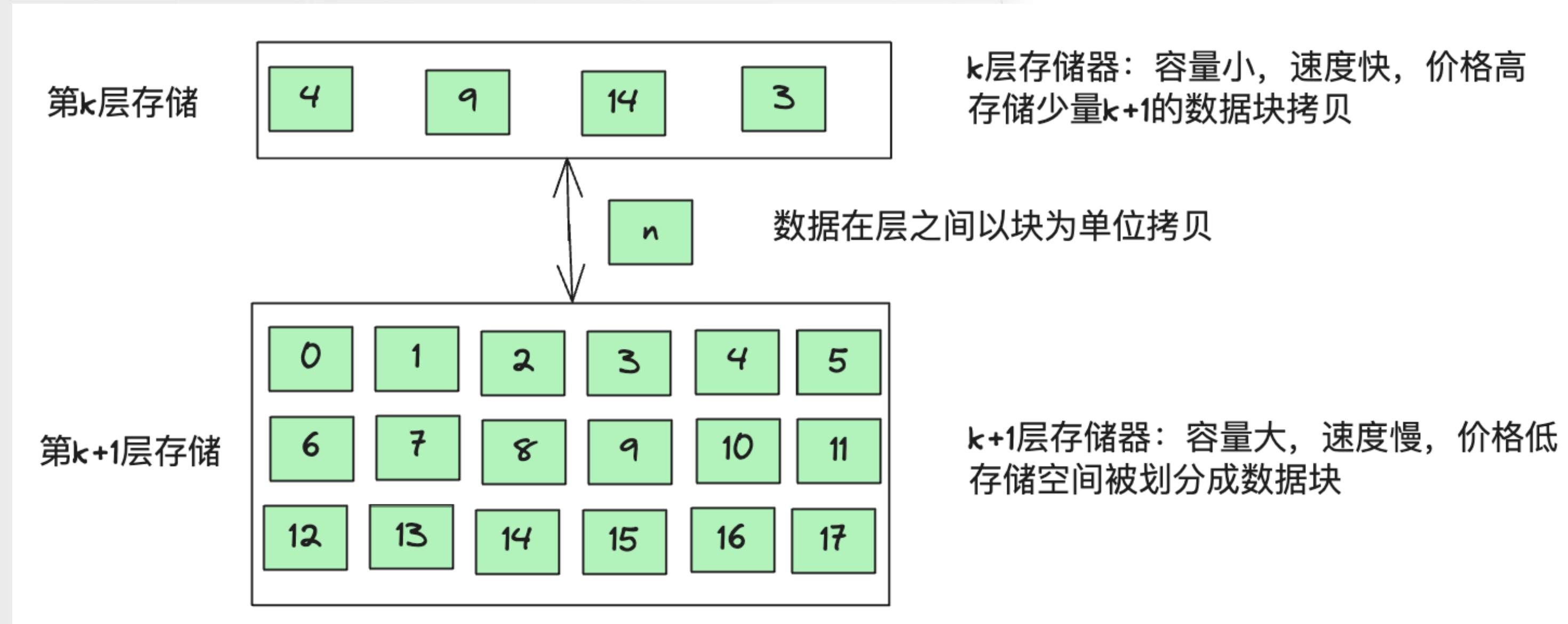


疑问：现代计算机系统里增加了很多Cache，表示计算机的总存储容量增加了，对吗？



2. 数据的表示与存储

2.9.1 缓存原理



1. 缓存命中cache hit
2. 缓存不命中cache miss, 将发生替换 (随机, LRU, LFU)
3. 空的缓存 (cold cache) 冷不命中。
4. 放置策略(placement policy): 直接映射, 全相联, 组相联



2. 数据的表示与存储

2.9.1 几个概念

1. 命中率 h : CPU要访问的信息在Cache中的占比
2. 平均访问时间 t_a : $ht_c + (1-h)t_m$, 其中 t_c 为命中访问时间, t_m 为未命中访问时间
3. Cache-主存系统效率 e : t_c/t_a

例1: 假设CPU执行某段程序时, 命中Cache2000次, 未命中50次, Cache的存储周期为50ns, 主存的存储周期为200ns, 试求:

- 1) 命中率
- 2) 平均访问时间
- 3) Cache-主存系统效率

$$2000/2050 = h = 97\%$$

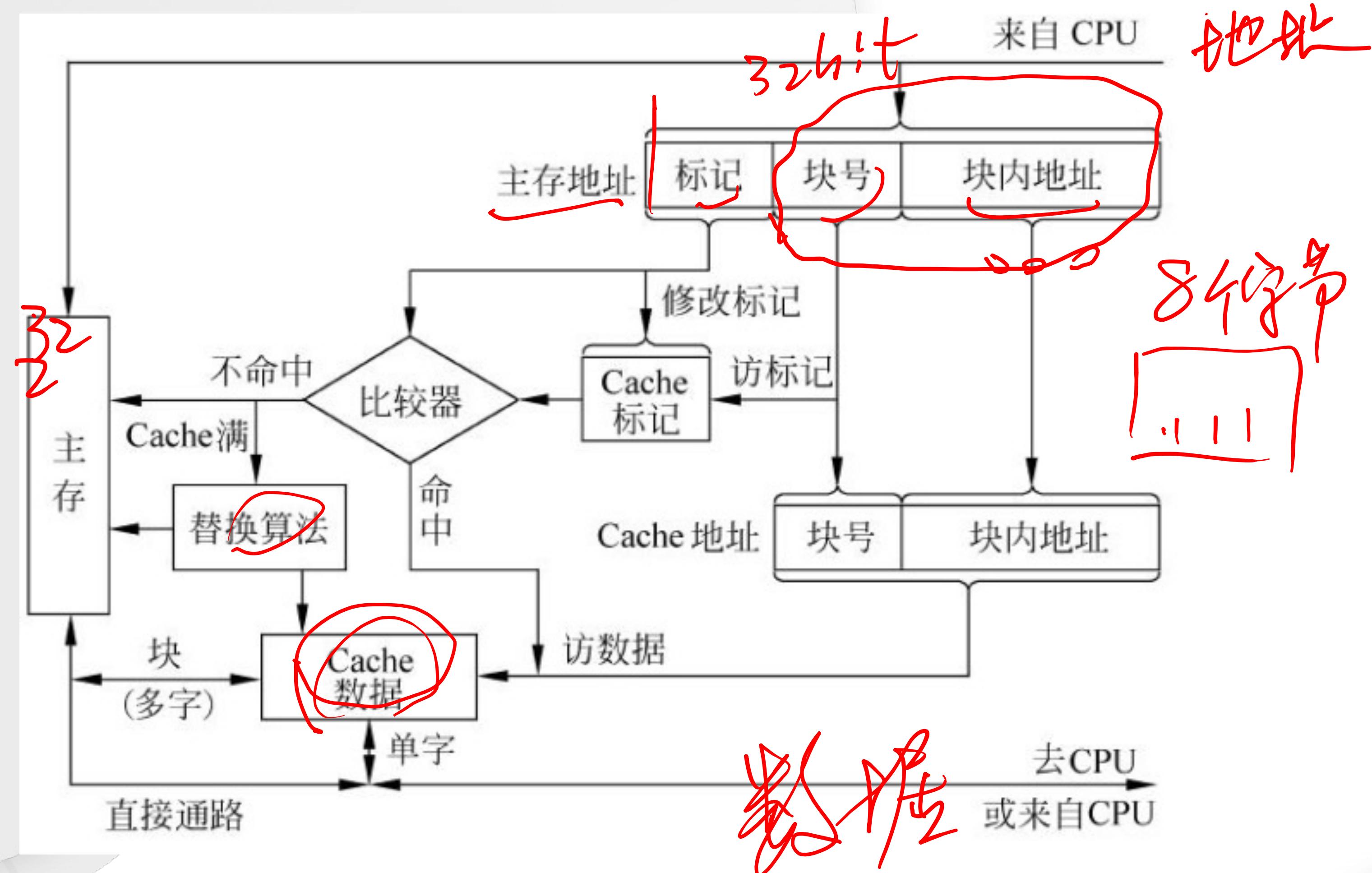
$$0.97 \times 50\text{ns} + (1-0.97) \times 200\text{ns} = 54.5\text{ns}$$

$$e = \frac{50}{54.5} = 91.7\%$$



2. 数据的表示与存储

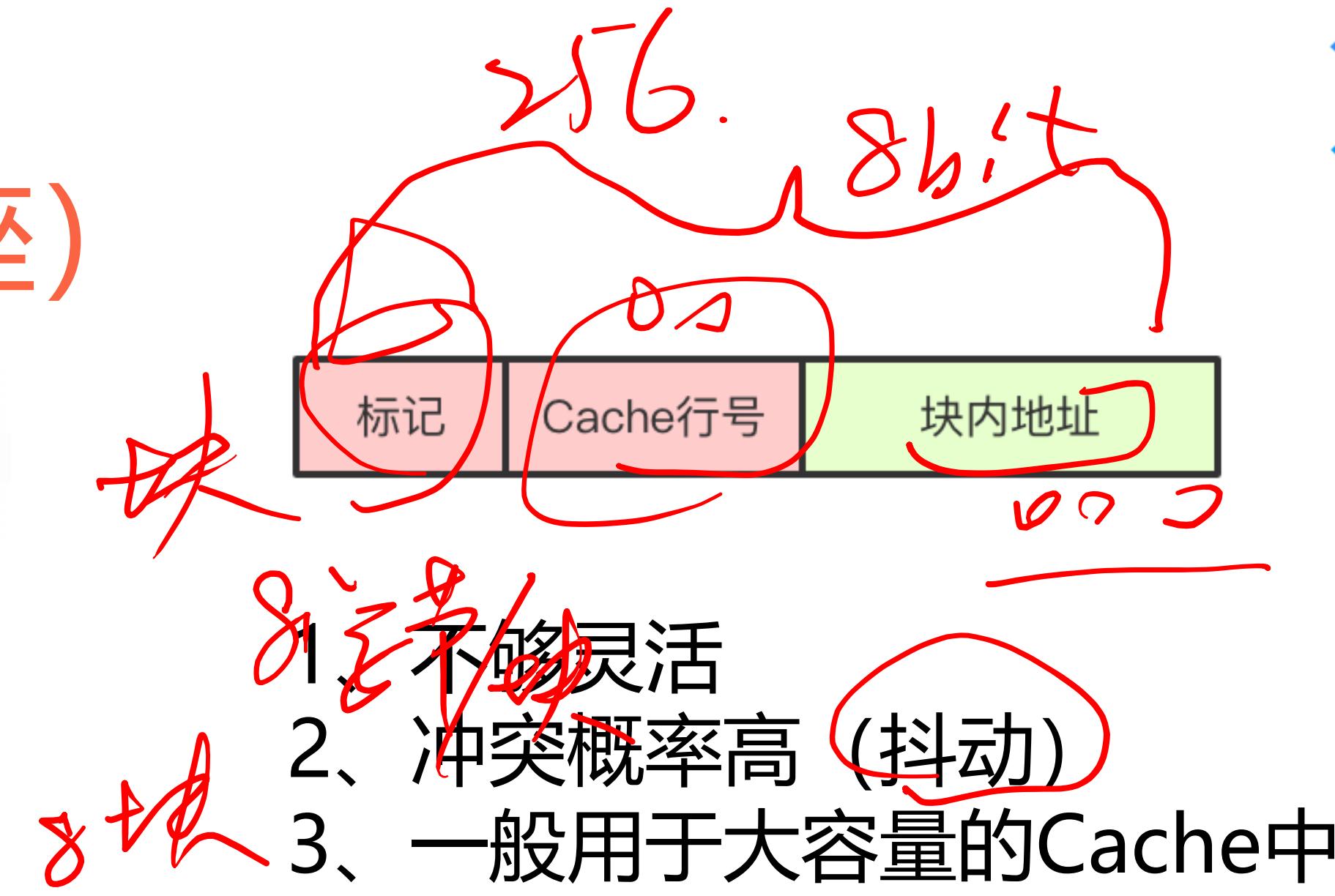
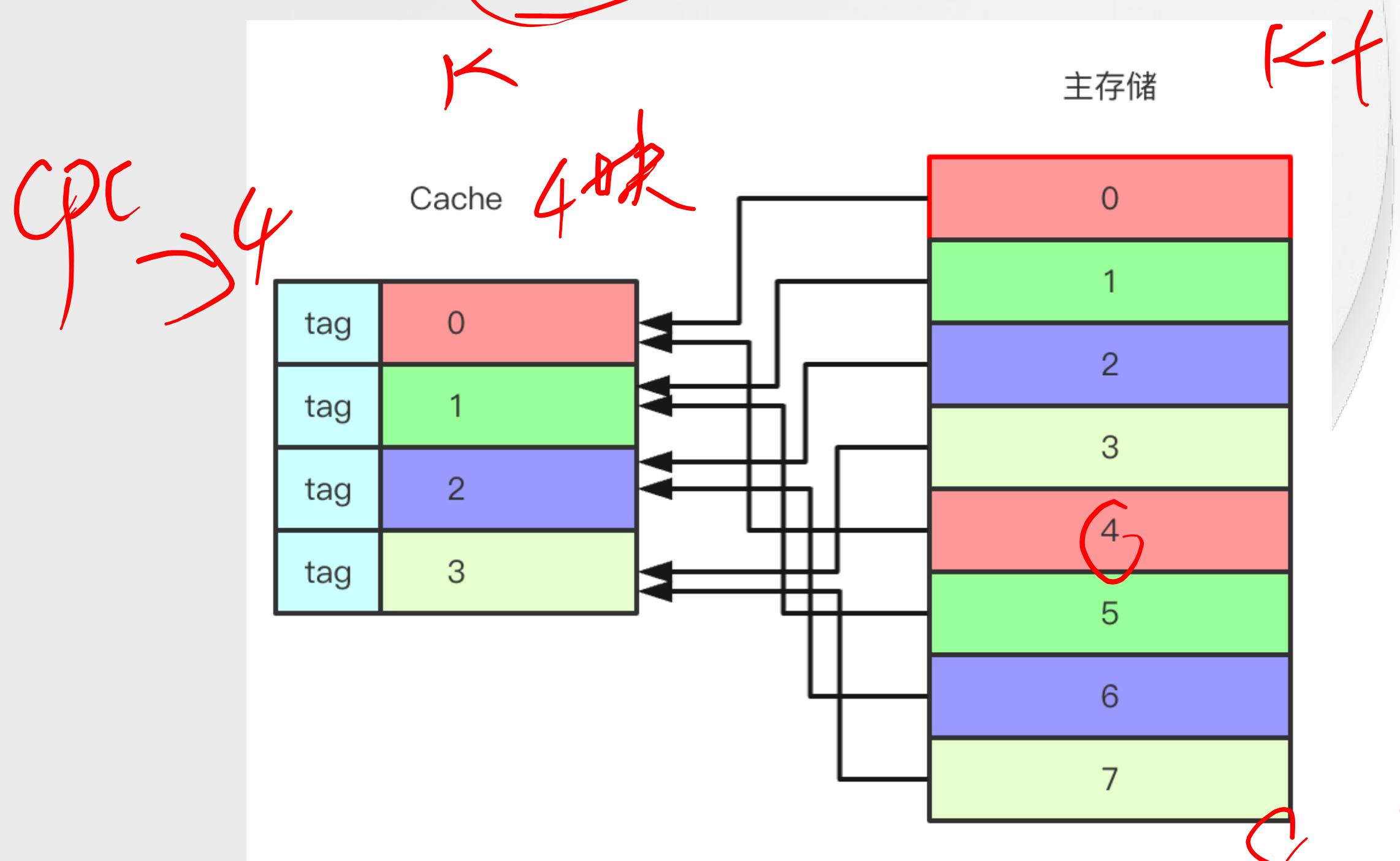
2.9.2 Cache映射地址处理过程





2. 数据的表示与存储

2.9.2 直接映射 (对号入座)

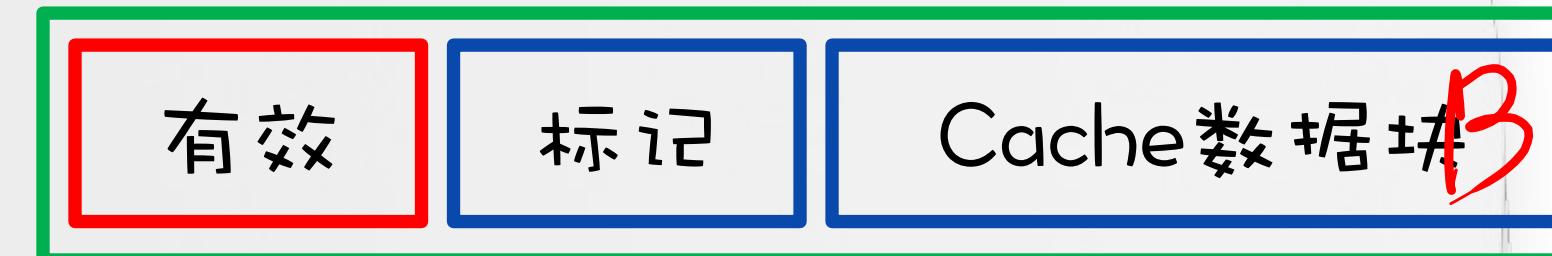




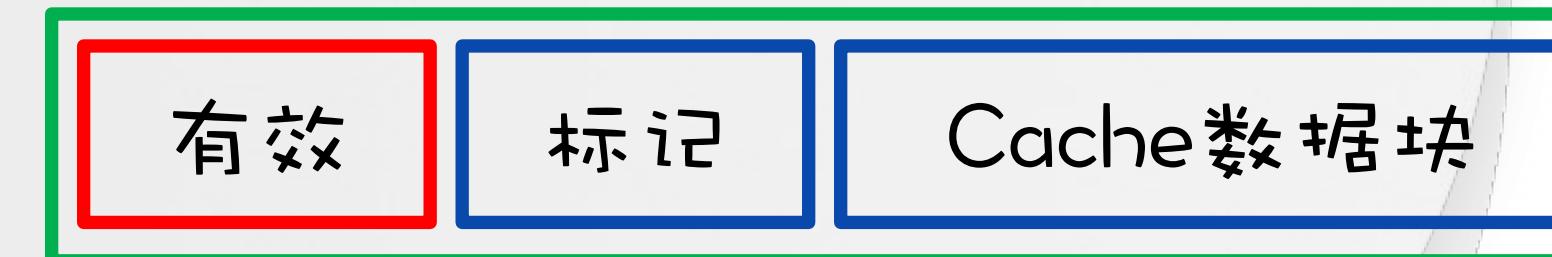
2. 数据的表示与存储

2.9.2 直接映射组选择

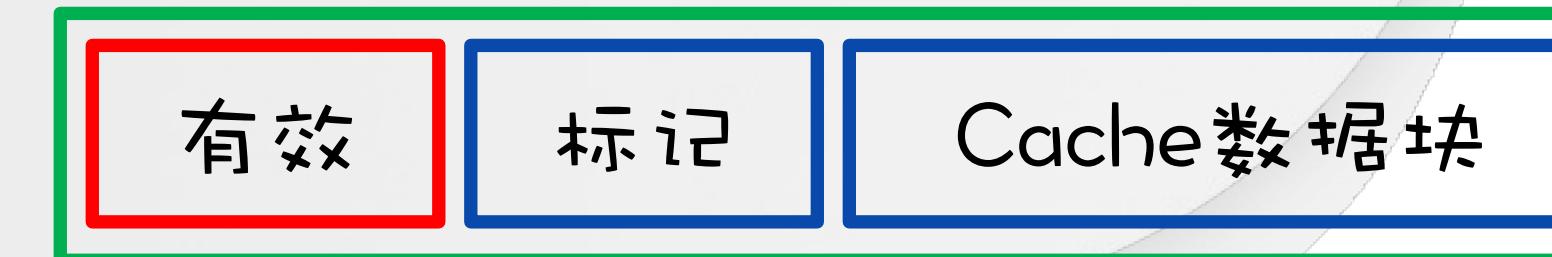
行0：



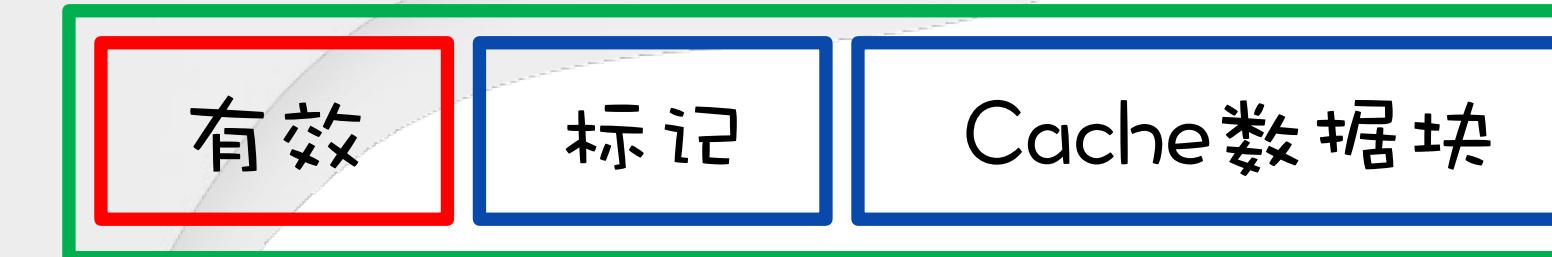
行1：



行2：



行S-1：



每行1
个有
效位

每行t
个有
效位

每块有
B字节

Size
位



256B

访存地址：

M=8

m	Cache	块大小B	t	s	b
8	16B	2B	4	3	1
32	1024B	4B	22	8	2
32	2048B	8B	21	8	3

求三种地址位分配：t, s, b

$$1024 = 2^{10} / 2$$

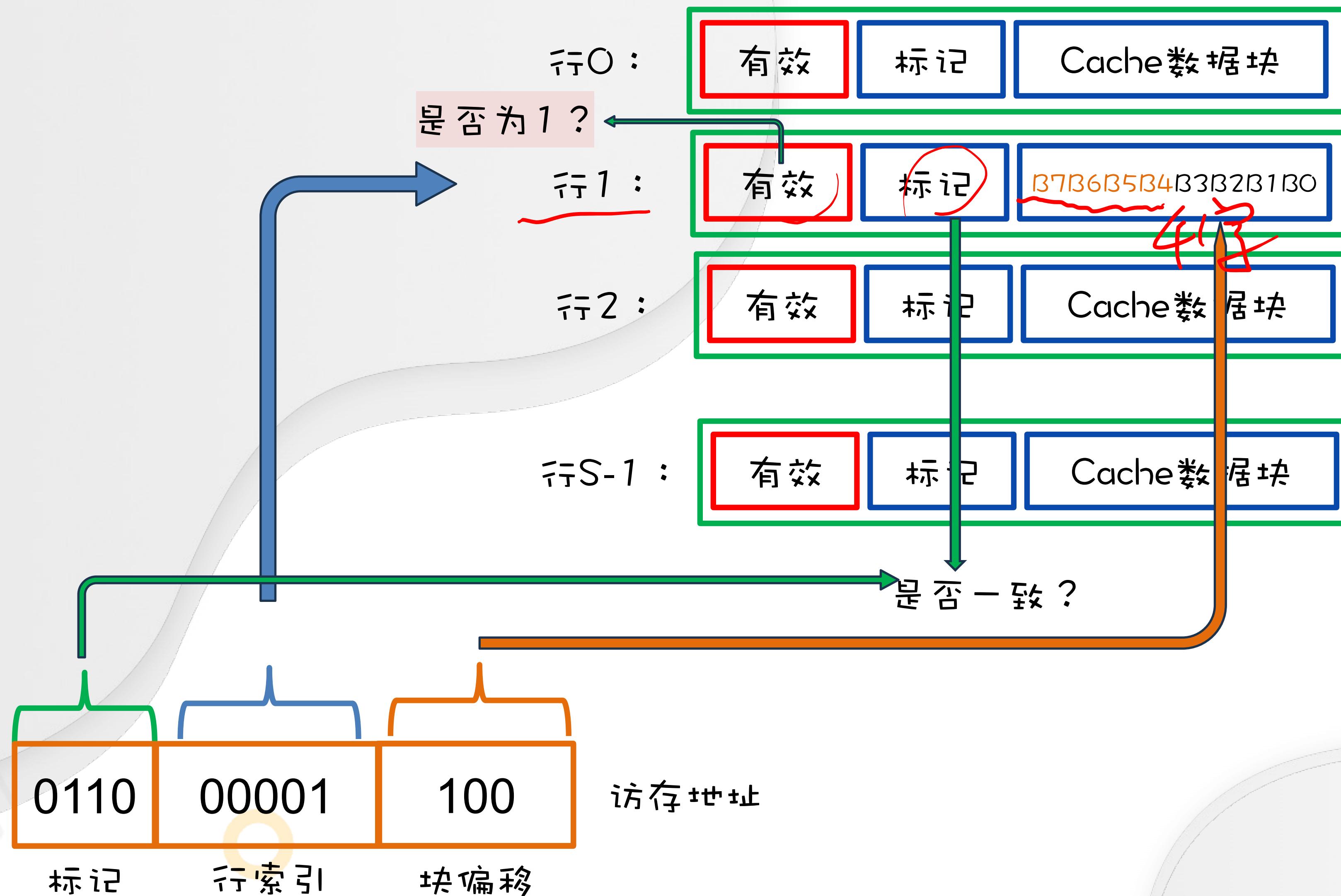
$$2048 = 2^{11} / 2$$



2. 数据的表示与存储

2.9.2 直接映射组选择

8.18



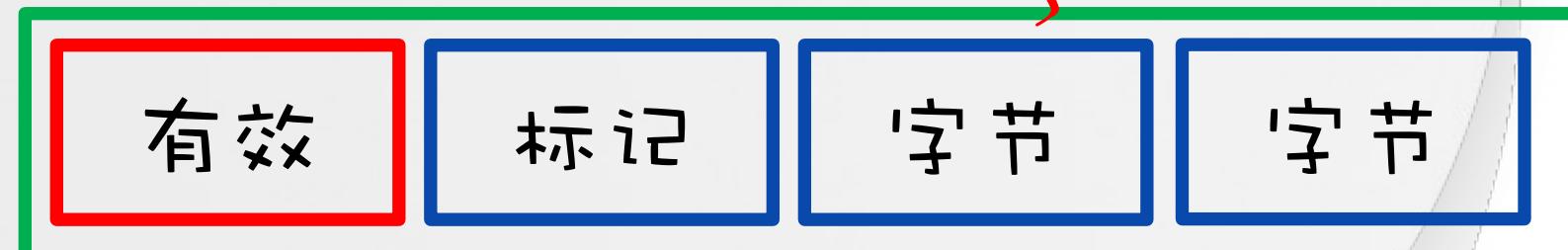


2. 数据的表示与存储

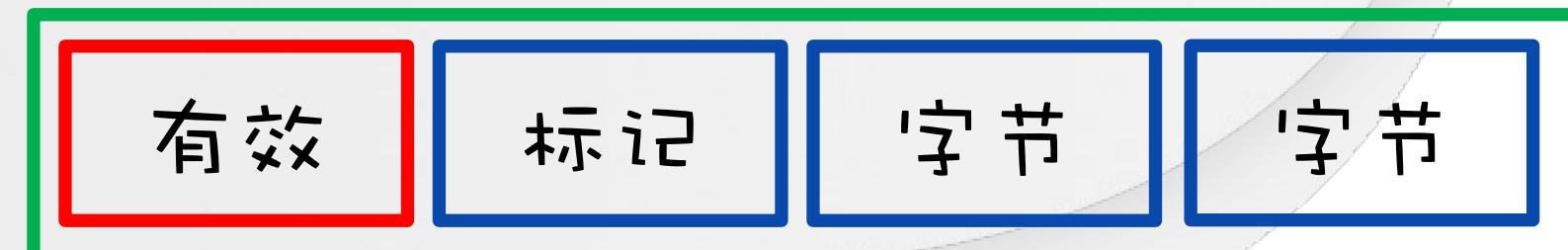
2.9.2 直接映射举例

$$\text{Cache}(S, E, B) = (4, \cancel{1}, 2)$$

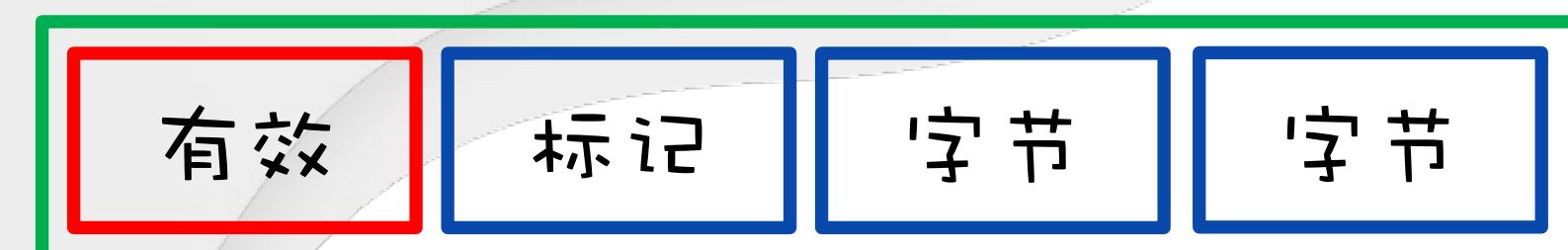
行0：



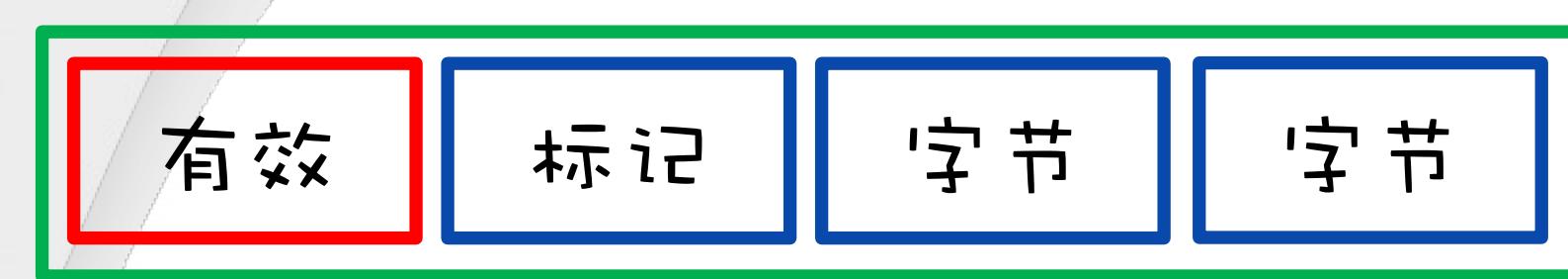
行1：



行2：



行3：





2. 数据的表示与存储

2.9.2 直接映射举例

Memory(B , m) = (2, 4) $\frac{4}{2}$, 16字节

地址	标记	索引	偏移	块号
0	00	00	0	0
1	00	00	1	0
2	01	01	0	1
3	01	01	1	1
4	10	10	0	2
5	10	10	1	2
6	11	11	0	3
7	11	11	1	3

地址	标记	索引	偏移	块号
8	1	00	0	4
9	1	00	1	4
10	1	01	0	5
11	1	01	1	5
12	1	10	0	6
13	1	10	1	6
14	1	11	0	7
15	1	11	1	7

84页



2. 数据的表示与存储

2.9.2 直接映射举例

0000	m[0]	block 0
0001	m[1]	
0010	m[2]	block 1
0011	m[3]	
0100	m[4]	block 2
0101	m[5]	
0110	m[6]	block 3
0111	m[7]	
1000	m[8]	block 4
1001	m[9]	
1010	m[10]	block 5
1011	m[11]	
1100	m[12]	block 6
1101	m[13]	
1110	m[14]	block 7
1111	m[15]	

行	有效	标记	字节1	字节0
0	/	t	m9	m8
1	/			
2	/		m13	m12

内存m0

命中

8.

9.

10.

11.

12.

13.

14.



2. 数据的表示与存储

2.9.2 Cache冲突不命中举例

Cache(S, E, B) = (2, 1, 16)

```
1 float dotprod(float x[8], float y[8])
2 {
3     float sum = 0.0;
4     int i;
5     for (i = 0; i < 8; i++)
6     {
7         sum += x[i] * y[i];
8     }
9     return sum;
10 }
```

元素	地址	行号	元素	地址	行号
x[0]	0	0	y[0]	32	0
x[1]	4	0	y[1]	36	0
x[2]	8	0	y[2]	40	0
x[3]	12	0	y[3]	44	0
x[4]	16	1	y[4]	48	1
x[5]	20	1	y[5]	52	1
x[6]	24	1	y[6]	56	1
x[7]	28	1	y[7]	60	1

行	有效	块
0	1	M_{0,1} M_{1,2} M_{2,3} M_{3,1}
1		

抖动



2. 数据的表示与存储

2.9.2 Cache冲突不命中举例

```
1 float dotprod(float x[12], float y[8])
2 {
3     float sum = 0.0;
4     int i;
5     for (i = 0; i < 8; i++)
6     {
7         sum += x[i] * y[i];
8     }
9     return sum;
10 }
```

行	有效	块
0		$y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7$
1		$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$

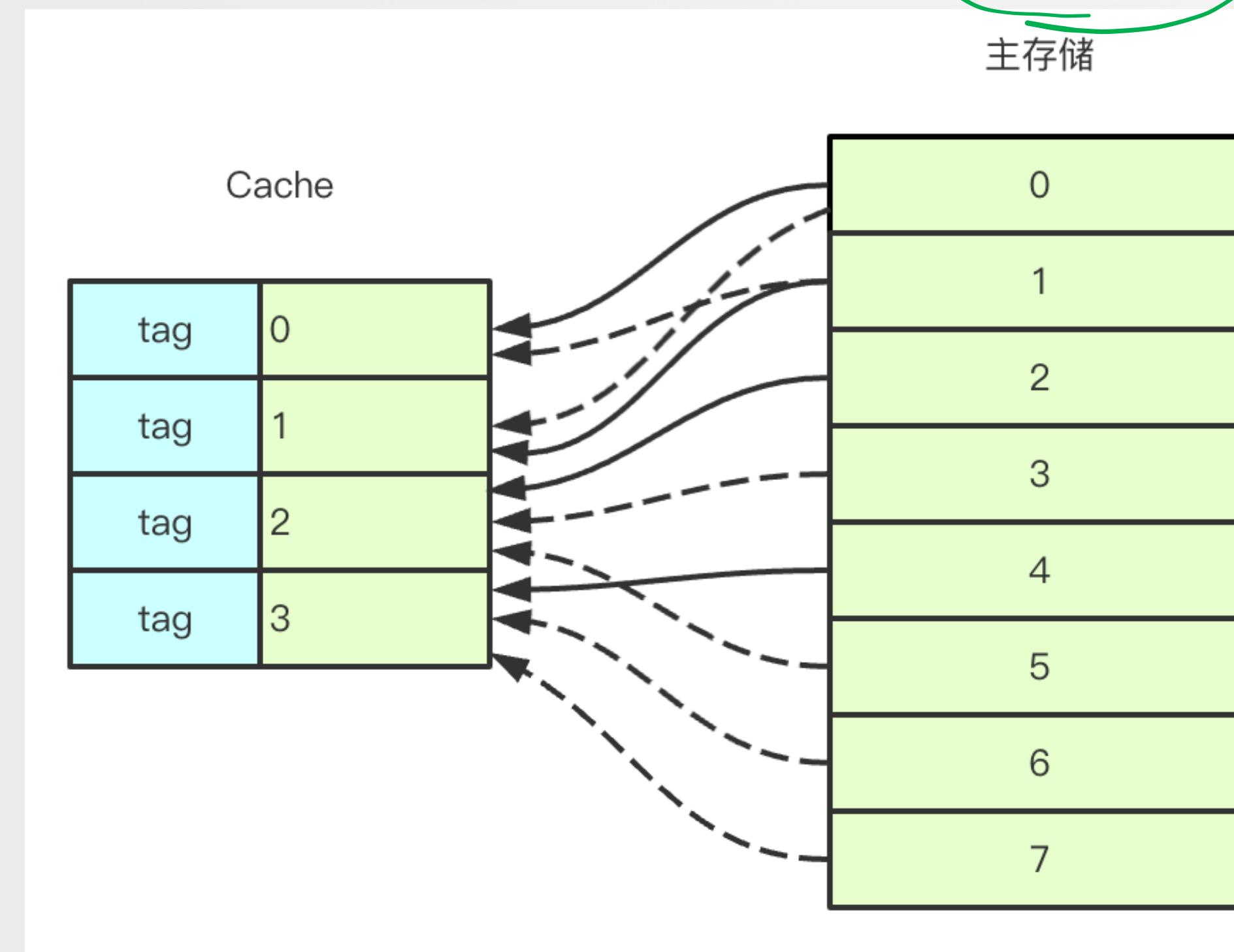
行 有效 块
 $y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7$
 $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$

元素	地址	行号	元素	地址	行号
x[0]	0	0		32	0
x[1]	4	0		36	0
x[2]	8	0		40	0
x[3]	12	0		44	0
x[4]	16	1	y[0]	48	1
x[5]	20	1	y[1]	52	1
x[6]	24	1	y[2]	56	1
x[7]	28	1	y[3]	60	1
			y[4]	64	0
			y[5]	68	0
			y[6]	72	0
			y[7]	76	0



2. 数据的表示与存储

2.9.2 全相联映射 (随便坐)

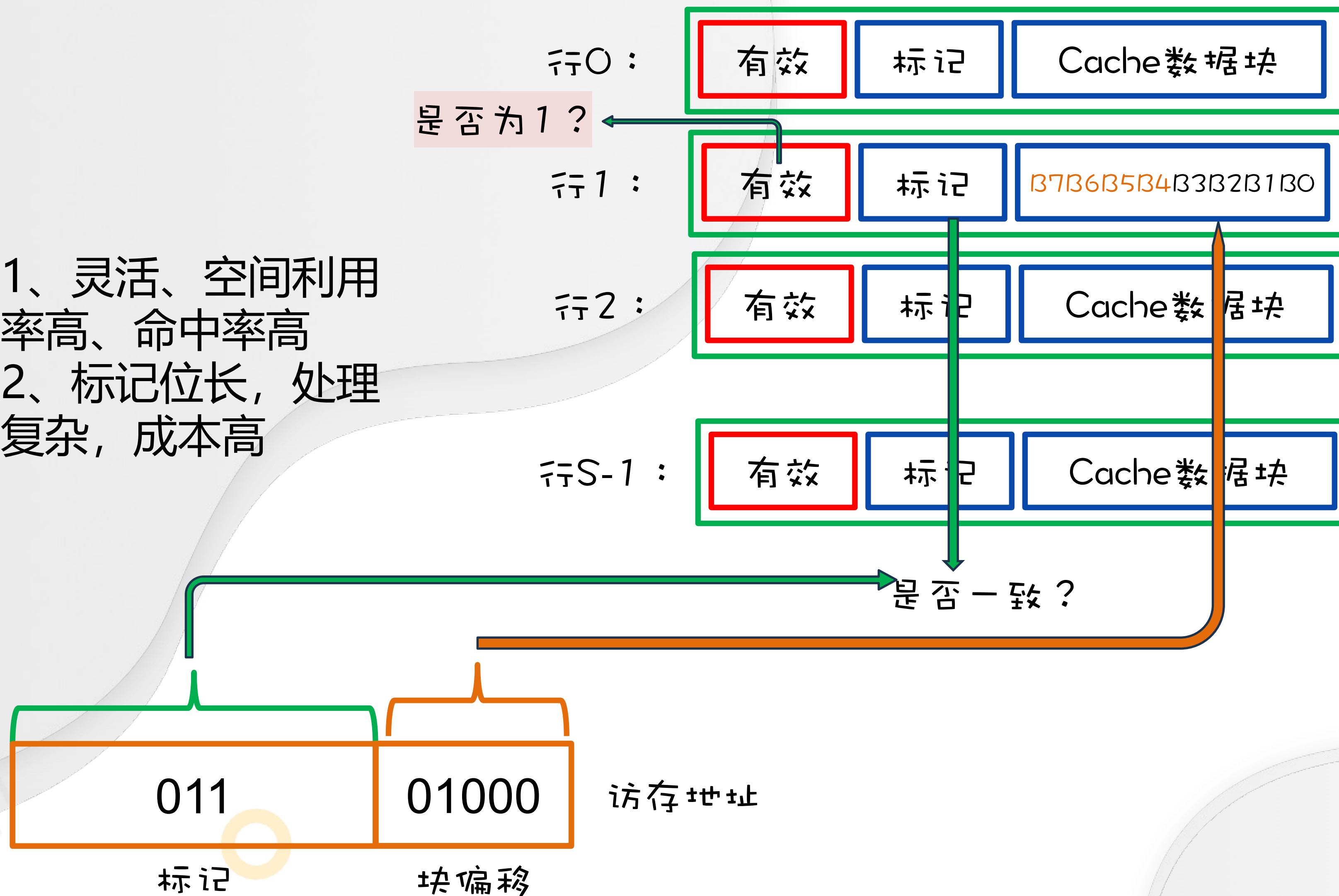




2. 数据的表示与存储

2.9.2 全相联映射

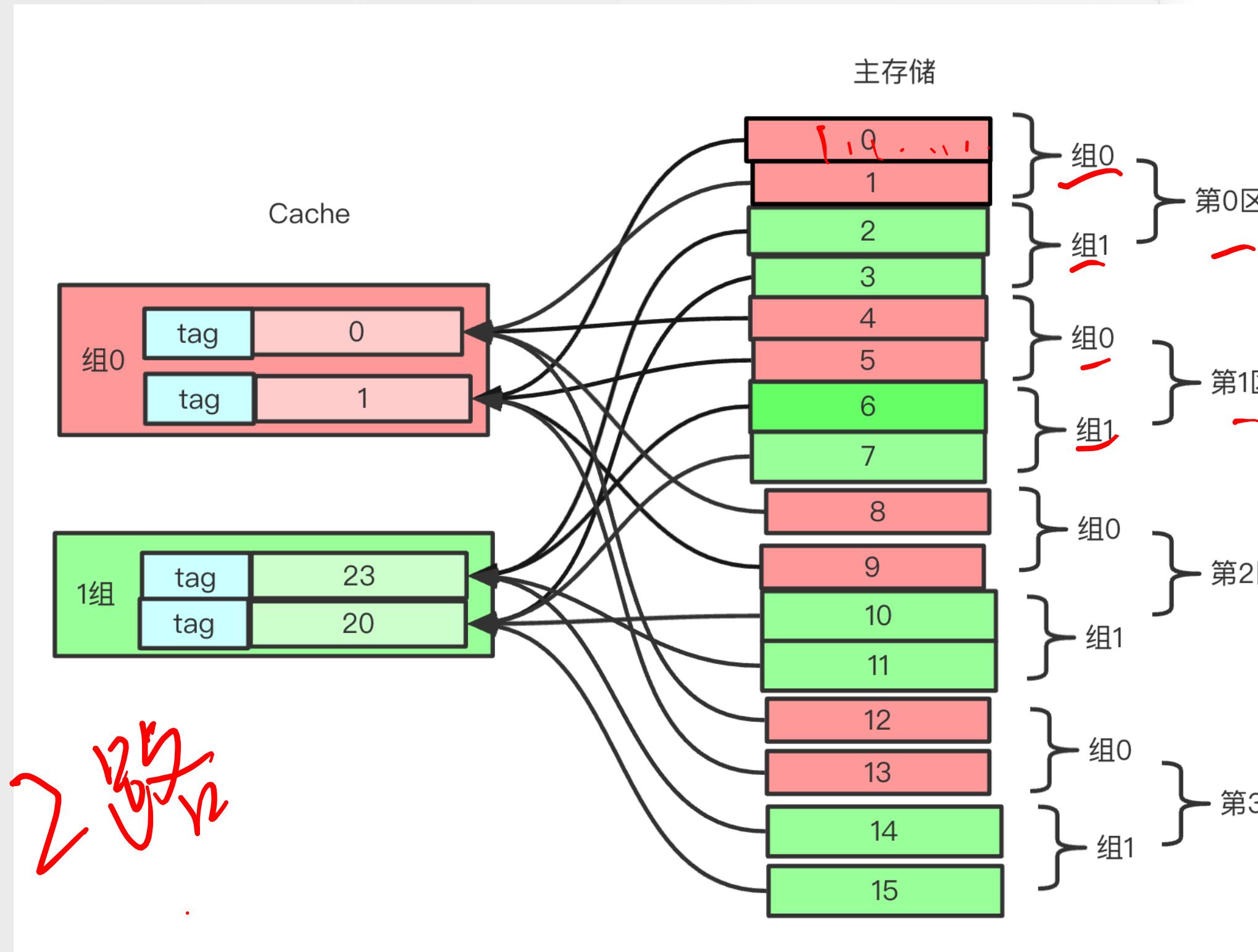
- 1、灵活、空间利用率高、命中率高
- 2、标记位长，处理复杂，成本高





2. 数据的表示与存储

2.9.2 组相联映射 (分组对号入座, 组内随便坐)



- 1、两种方案的折中
- 2、每组n块(行)数称为“n路组相联”



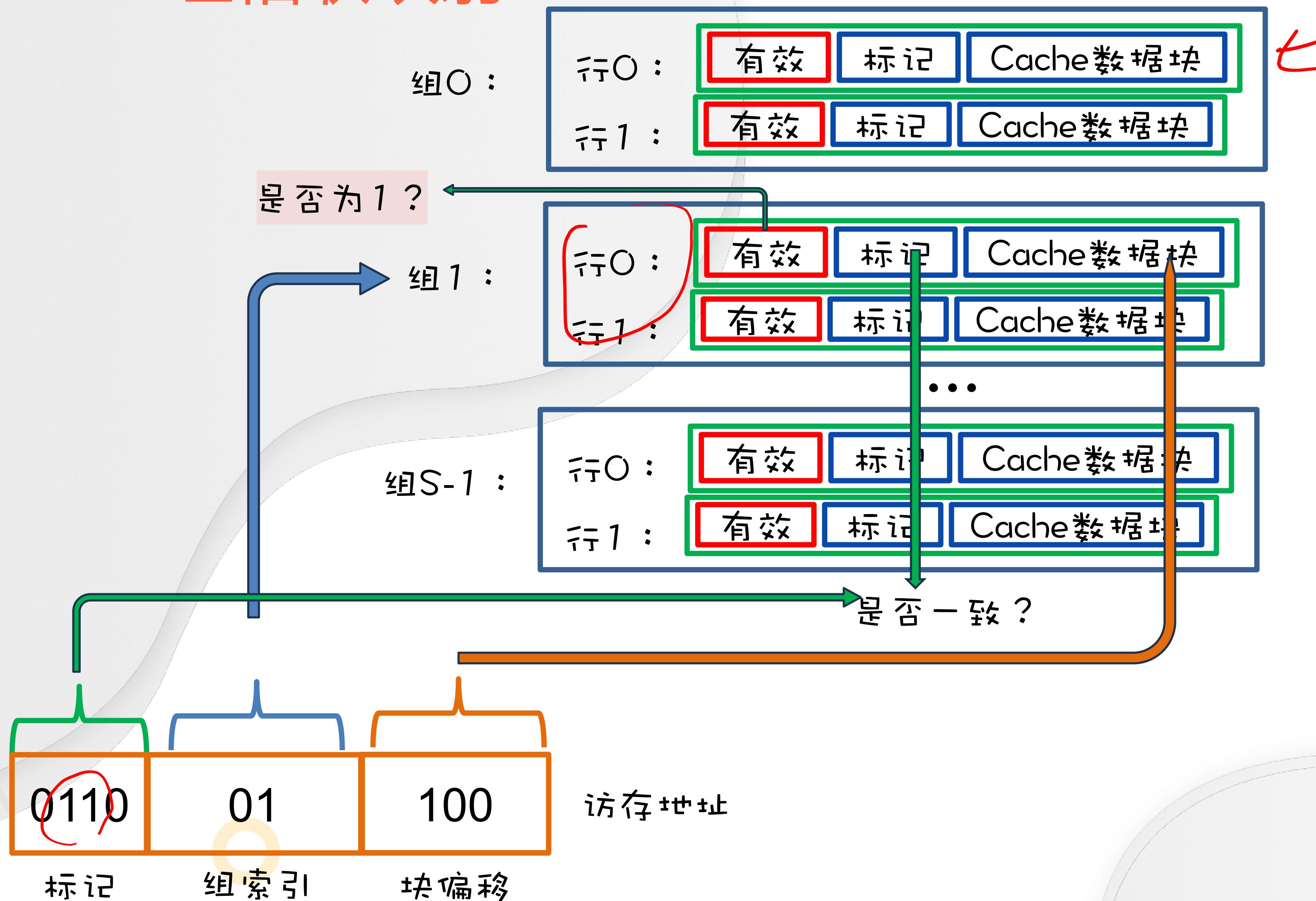
2. 数据的表示与存储

2.9.2 组相联映射地址处理

- 1、组选择 (set selection)
- 2、行匹配 (line matching)
- 3、字抽取 (word extraction)

2. 数据的表示与存储

2.9.2 组相联映射





2. 数据的表示与存储

2.9.2 举例

例2：假设某个计算机的主存大小为256MB，按字节编址，数据Cache有8行，一长64B。

1、若不考虑用于Cache的一致维护性和替换算法控制位，并且采取直接映射方式，则该数据Cache的总容量为多少？

2、若该Cache采用直接映射方式，则主存地址为3200（十进制）的主存块对应的Cache行号是多少？采用二路组相联映射时又是多少？

3、以直播映射方式为例，简述访存过程（设访存的地址为0123456H）。

0000000|101001101000101010110
块 行 块偏
001 第 1 行 . 块偏 22.



2. 数据的表示与存储

2.9.3 Cache什么时候需要替换策略

- 1、全相联时，所有Cache块装满后，再有数据块装入时
- 2、组相联时，组内所有Cache块装满后，再有数据块装入时



2. 数据的表示与存储

2.9.3 Cache替换策略

- 1、随机替换 (Random)
- 2、先进先出 (FIFO) : 最早调入的被换走
- 3、最近最少使用 (Least Recently Used) : 命中或调入计数器清0, 比命中的计数值小的加1, 计算器最大的被换掉, 计数器值 (0 ~ 行数-1)
- 4、最不常用 (Least Frequently Used)



2. 数据的表示与存储

2.9.3 FIFO替换演示

有4个Cache行，程序访问主存块顺序为：1,2,3,4,1,2,5,1,2,3,4,5

待访问主存块	1	2	3	4	1	2	5	1	2	3	4	5
Cache#0	1	1	1	1	1	1	5	5	5	5	4	4
Cache#0		2	2	2	2	2	2	1	1	1	1	5
Cache#0		3	3	3	3	3	3	3	2	2	2	2
Cache#0		3	4	4	4	4	4	4	4	3	3	3
命中？	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
替换？	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓

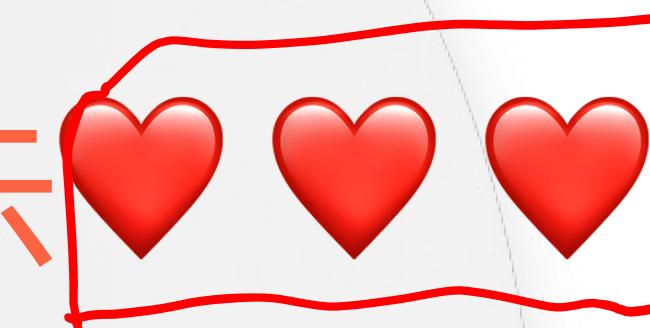
最先装入的被替换

2/12
6/12



2. 数据的表示与存储

2.9.3 LRU替换演示



有4个Cache行，程序访问主存块顺序为：1,2,3,4,1,2,5,1,2,3,4,5

待访问主存块	1	2	3	4	1	2	5	1	2	3	4	5
Cache#0	1/0	1/1	1/2	1/3	1/0	1/1	1/2	1/0	1/1	1/2	1/3	5/0
Cache#0	2/0	2/1	2/2	2/3	2/0	2/1	2/2	2/0	2/1	2/2	2/3	
Cache#0	3/0	3/1	3/2	3/3	3/0	3/1	3/2	3/0	3/1	3/2	3/0	4/1
Cache#0					4/0	4/1	4/2	4/3	4/0	4/1	4/2	3/1
命中？	X	X	X	X	✓	✓	X	✓	✓	X	X	X
替换？	X	X	X	X	X	X	✓	X	X	✓	✓	✓

1、未命中且有空闲行时，新装入的行计数清0，其余非空行加1

2、命中时，比所命中行计数值小的加1，且命中行计算清0

3、未命中，且无空闲，将计数值最大的替换且清0，其他计数值加1

当频繁访问的主存块大于行数时，
有可能发生抖动。

4/12



2. 数据的表示与存储

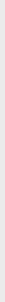
2.9.3 LFU替换演示

有4个Cache行，程序访问主存块顺序为：1,2,3,4,1,2,5,1,2,3,4,5

待访问主存块	1	2	3	4	1	2	5	1	2	3	4	5
Cache#0	1/0	1/0	1/0	1/0	1/1	1/1	1/1	1/2	1/2	1/2	1/2	1/2
Cache#0	2/0	2/0	2/0	2/0	2/1	2/1	2/1	2/2	2/2	2/2	2/2	2/2
Cache#0	3/0	3/0	3/0	3/0	3/0	3/0	5/0	5/0	5/0	5/0	4/0	4/0
Cache#0				4/0	4/0	4/0	4/0	4/0	4/0	3/0	3/0	5/0
命中？	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗
替换？	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✓

- 1、新调入时计数为0
- 2、访问过的计数加1
- 3、满了要替换时，计数最小的被换掉。数值一样时按行号顺序或使用FIFO策略。

当下频繁使用的块，不一定后面就一直会用。



2. 数据的表示与存储

2.9.4 Cache写策略

1、写命中 (Write Hit) :

全写法, 写穿透 (write-through) : 同时写入Cache和主存

回写法 (write-back) : 只写入Cache, 被替换出去时写入主存 (要记录状态, 脏位)

2、写不命中 (Write Miss) :

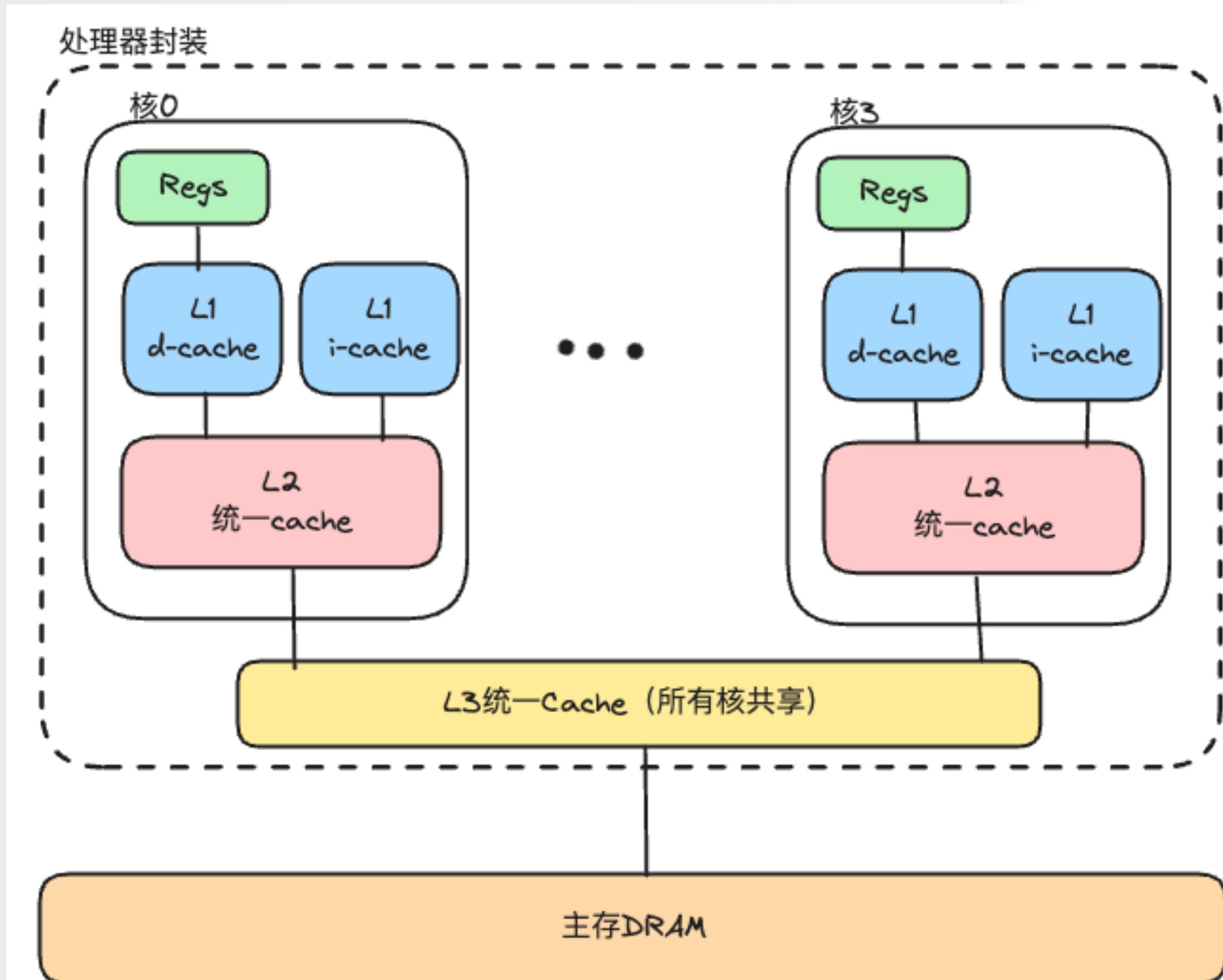
写分配 (write-allocate) : 加载主存块到Cache中, 然后更新

写不分配 (no-write-allocate) : 只写入主存, 不调入Cache。



2. 数据的表示与存储

2.9.4 Intel Core i7 Cache





2. 数据的表示与存储

2.9.4 Intel Core i7 Cache

缓存类型	访问时间 (时钟周期)	缓存容量C	相联度E	块大小B	组数S
L1 i-cache	4	32KB	8	64B	64
L1 d-cache	4	32KB	8	64B	64
L2 cache	10	256KB	8	64B	512
L3 cache	4075	8MB	16	64B	8192



2. 数据的表示与存储

2.9 本节总结

1. 高速缓存Cache利用计算机处理的局部性特性，
将少部分数据从慢速存储体拷贝到高速存储体使
用，从而提升整体处理速度。
2. 组相联映射方式可以兼顾冲突和空间利用效率
3. 当发生冲突要替换时，使用LRU策略效果最佳
4. 缓存层级不高，在整个系统中根据使用分别优化
处理，协同完成计算，提升整体效率。

欢迎参与学习

WELCOME FOR YOUR JOINING

船说：计算机基础