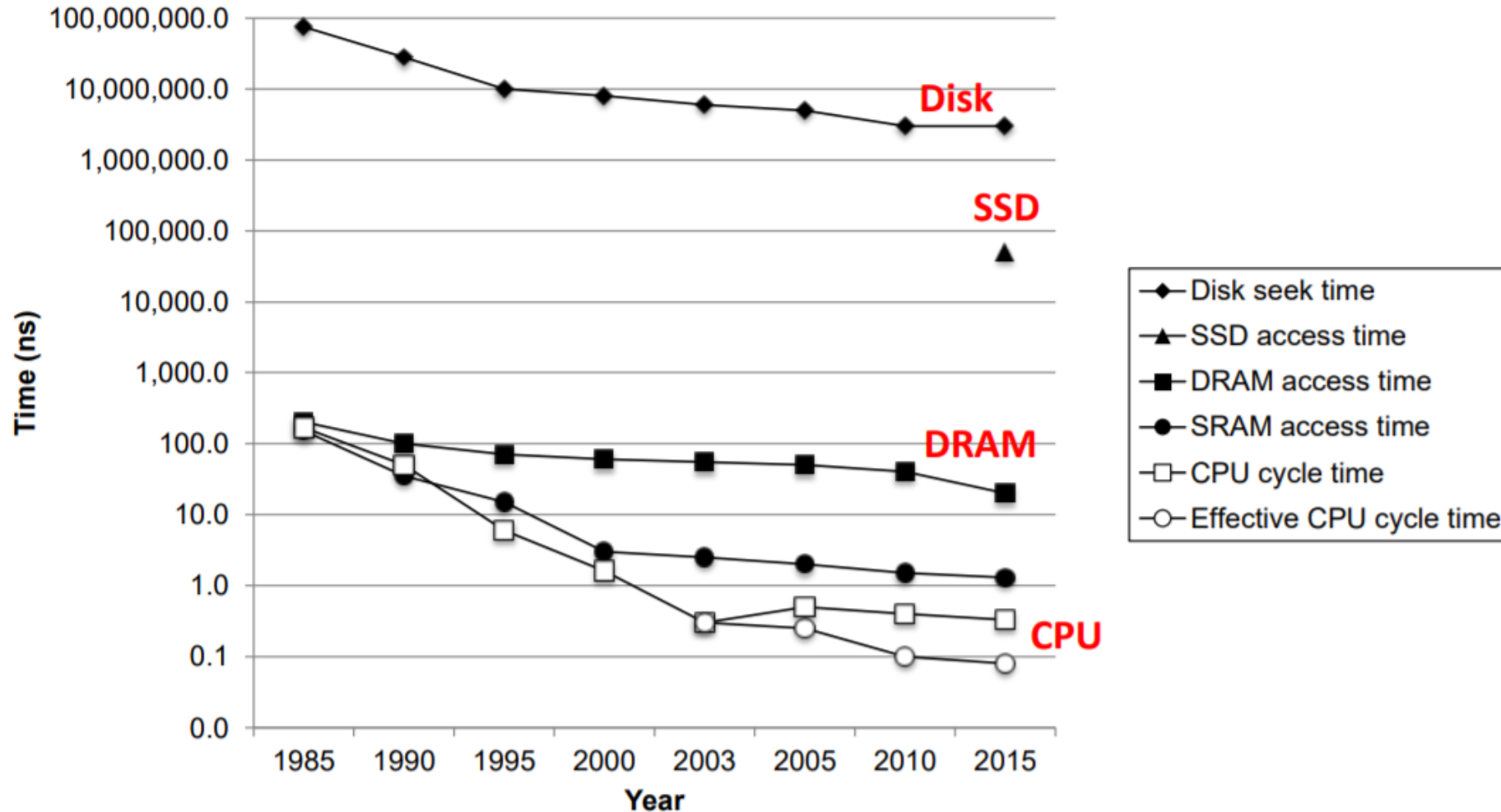


Cache Memories

熊婧

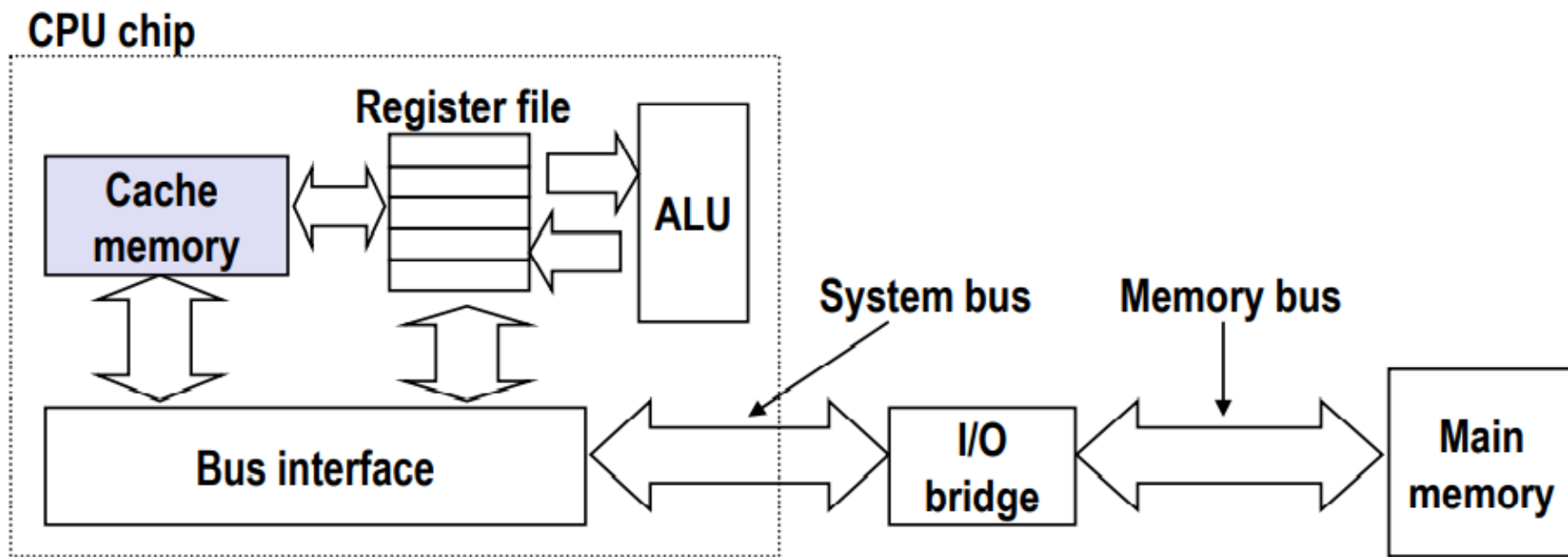
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



早期计算机系统的存储器层次结构只有三层：CPU寄存器、DRAM主存储器和磁盘存储。

cache



CPU和主存之间插入了一个小的SRAM高速缓存存储器
——L1高速缓存（4个时钟周期）
L2高速缓存（10个时钟周期）
L3高速缓存（50个时钟周期）

cache的真实结构

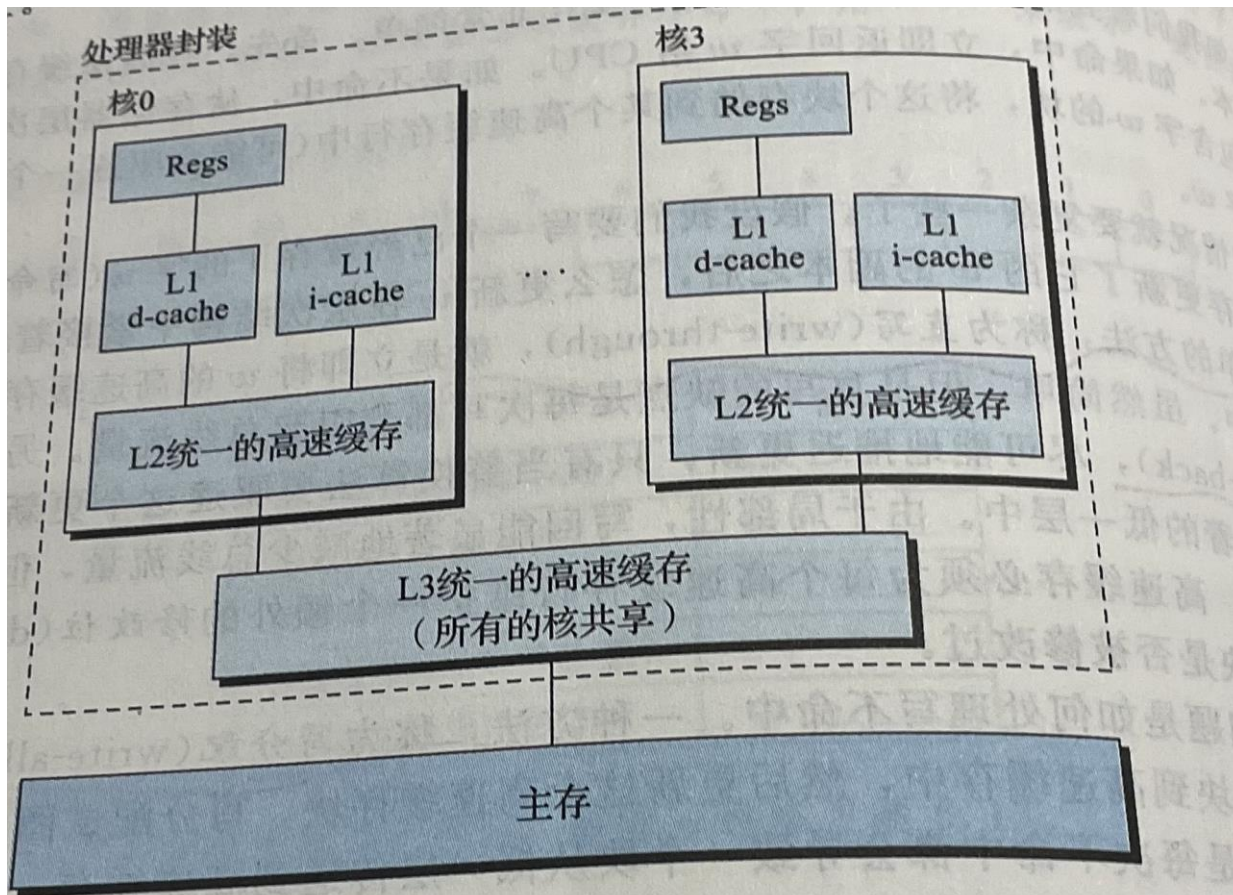


图 6-38 Intel Core i7 的高速缓存层次结构

Core i7 高速缓存的基本特性。

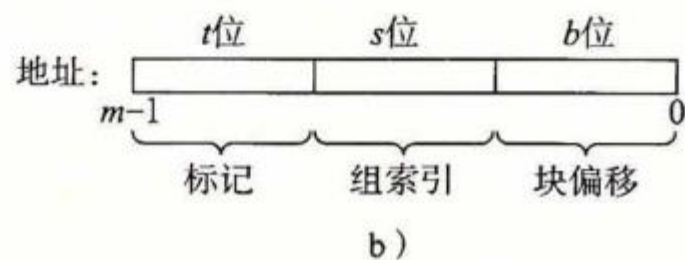
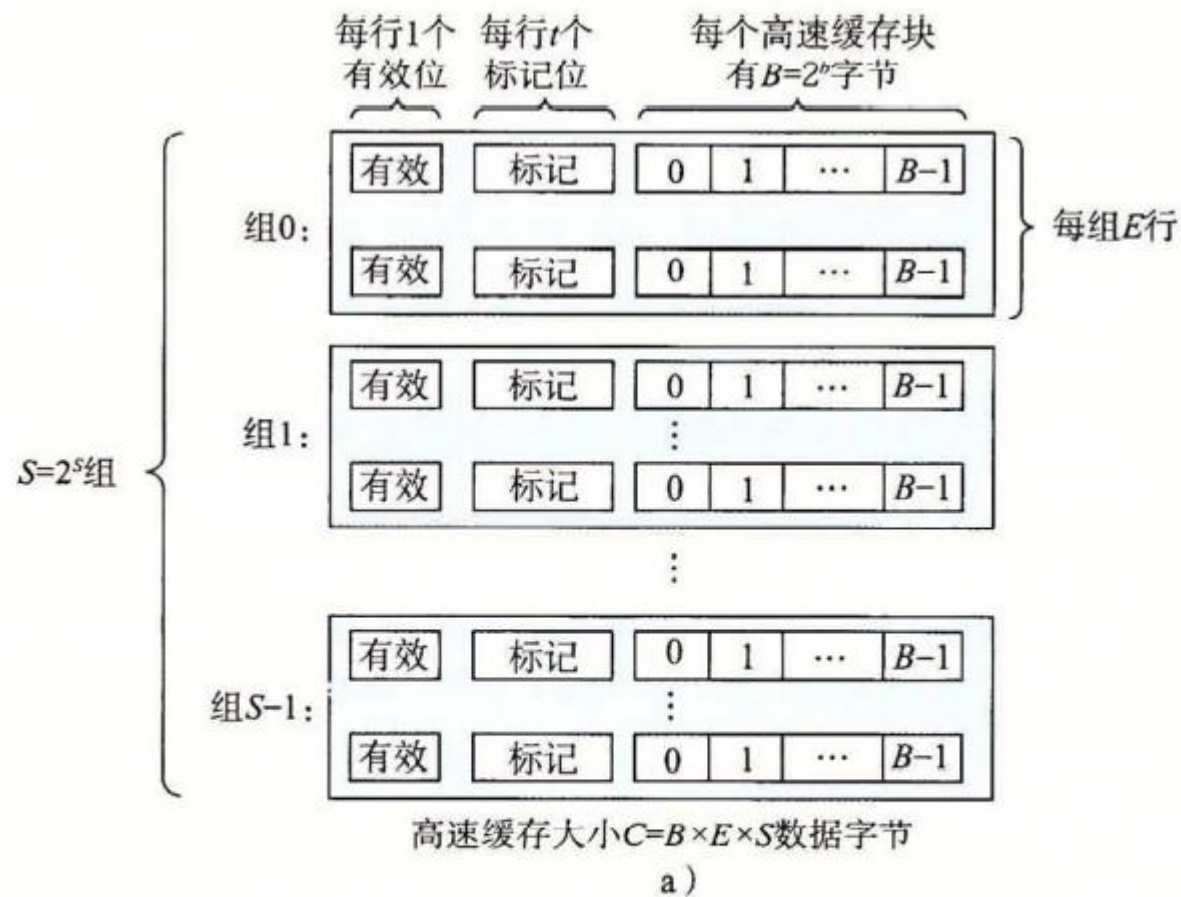
- 只保存指令的高速缓存称为i-cache，只保存程序数据的高速缓存称为d-cache，既包含指令又包含数据的高速缓存称为统一的高速缓存。
- 多核处理器

cache

Pipeline Depth	8
Processor Width	4
Instruction Window	128
DCache	32KB, 8-way, 64B, 2 cycles
ICache	32KB, 4-way, 64B, 2 cycles
L2Cache	256KB, 8-way, 64B, 10 cycles
L3Cache (LLC)	2MB, 16-way, 64B, 30 cycles
Memory latency	200 cycles

Table 1. The single-core processor.

Cache Organization



区分:

M: 内存地址的最大数量

m: 地址位数

$$M = 2^m$$

S: 组数

s: 组索引位数

$$S = 2^s$$

B: 数据块的字节数

b: 块偏移位数

$$B = 2^b$$

t: 标记位 $t = m - (b + s)$

C: 高速缓存的容量或者大小

工作流程:

组选择:

组索引位被解释为无符号整数判断组号


行匹配:

匹配 t 标记位判断是行号, 且要设置了有效位

字抽取:

通过偏移位判断在数据块中的字偏移

cache

 **练习题 6.9** 下表给出了几个不同的高速缓存的参数。确定每个高速缓存的高速缓存组数(S)、标记位数(t)、组索引位数(s)以及块偏移位数(b)。

高速缓存	m	C	B	E	S	t	s	b
1.	32	1024	4	1				
2.	32	1024	8	4				
3.	32	1024	32	32				

6.9 这个解答是对图 6-26 中各种高速缓存参数定义的直接应用。不那么令人兴奋，但是在能真正理解高速缓存如何工作之前，你需要理解高速缓存的结构是如何导致这样划分地址位的。

高速缓存	m	C	B	E	S	t	s	b
1.	32	1024	4	1	256	22	8	2
2.	32	1024	8	4	32	24	5	3
3.	32	1024	32	32	1	27	0	5

cache classification

- Direct Mapped Cache ($E = 1$)
- Set Associative Cache ($1 < E < C/B$)
- Fully Associative Cache ($S = 1, E = C/B$)

直接映射高速缓存

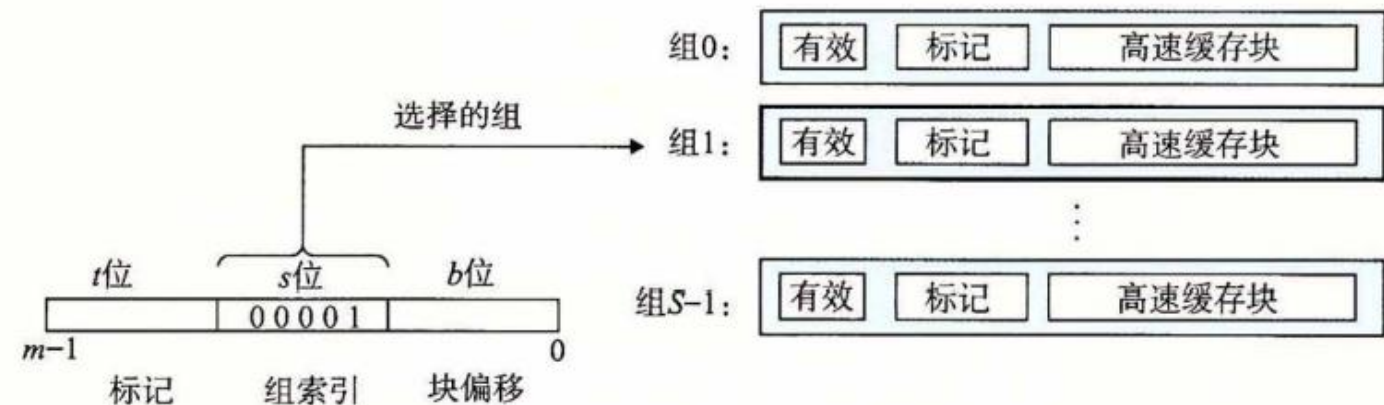


图 6-28 直接映射高速缓存中的组选择

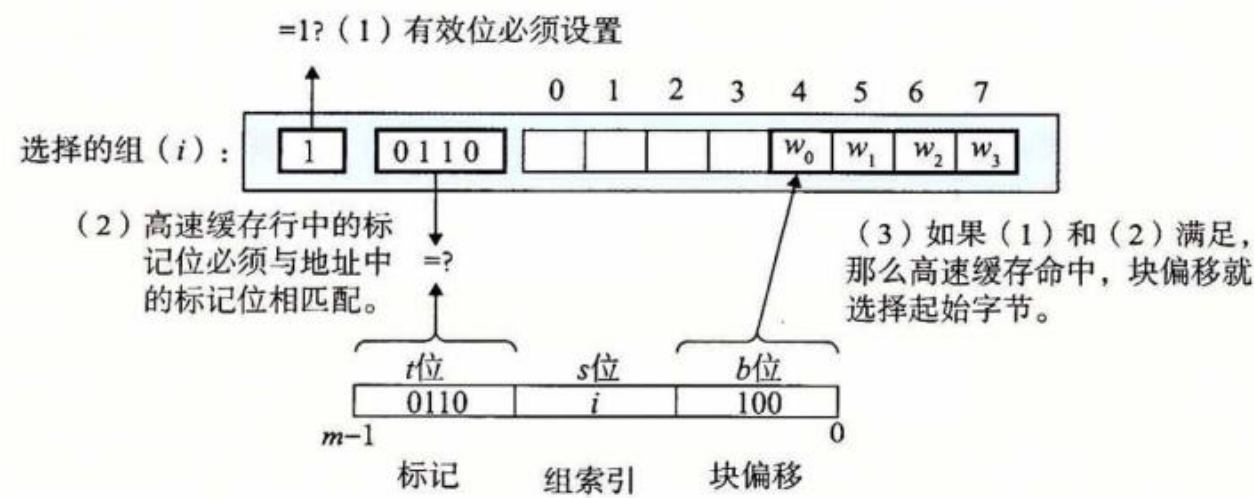


图 6-29 直接映射高速缓存中的行匹配和字选择。在高速缓存块中, w_0 表示字 w 的低位字节, w_1 是下一个字节, 依此类推

工作流程:

组选择:

组索引位被解释为无符号整数判断组号

行匹配:

匹配t标记位判断是行号, 且要设置了有效位

字抽取:

通过偏移位判断在数据块中的字偏移

替换策略:

用新取出的行替换当前行

直接映射高速缓存

地址 (十进制)	地址位			块号 (十进制)
	标记位 ($t=1$)	索引位 ($s=2$)	偏移位 ($b=1$)	
0	0	00	0	0
1	0	00	1	0
2	0	01	0	1
3	0	01	1	1
4	0	10	0	2
5	0	10	1	2
6	0	11	0	3
7	0	11	1	3
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

- 标记位+索引位唯一标识块
- 多个块映射到同一个高速缓存组（cache miss），并被标记位唯一标识

cache miss

- cold miss
- conflict miss(thrash)
- capacity miss

抖动

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

浮点数4个字节，假设一个块16个字节（一个块足够容纳4个浮点数），高速缓存由两个组组成。

高速缓存反复加载和驱逐相同的高速缓存块的组。

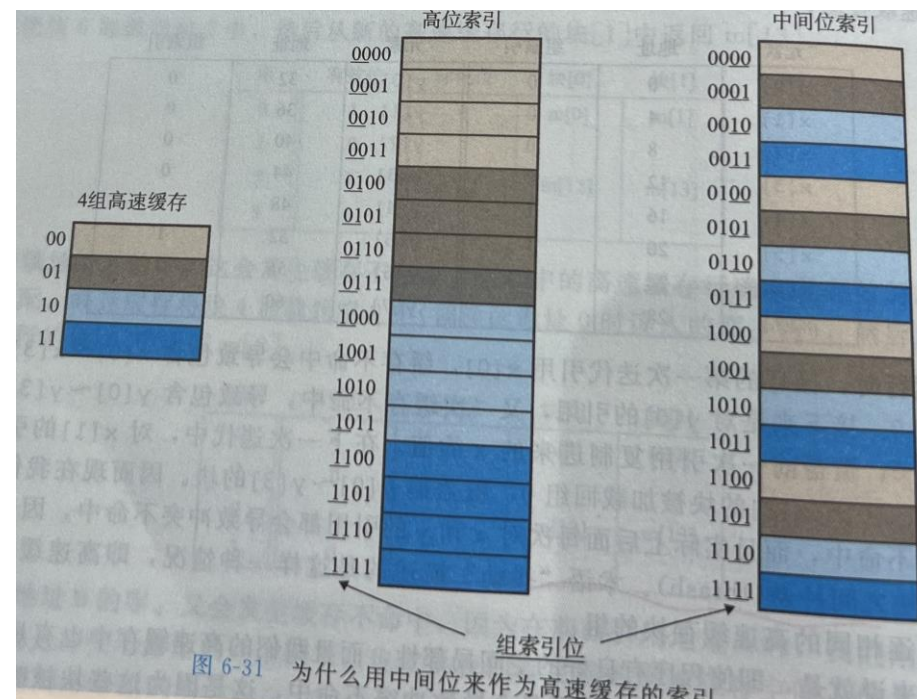
元素	地址	组索引	元素	地址	组索引
x[0]	0	0	y[0]	32	0
x[1]	4	0	y[1]	36	0
x[2]	8	0	y[2]	40	0
x[3]	12	0	y[3]	44	0
x[4]	16	1	y[4]	48	1
x[5]	20	1	y[5]	52	1
x[6]	24	1	y[6]	56	1
x[7]	28	1	y[7]	60	1

抖动

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

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

- float x[8] --> float x[12]
字节填充
- Q:为什么用中间位做索引?



全相联高速缓存

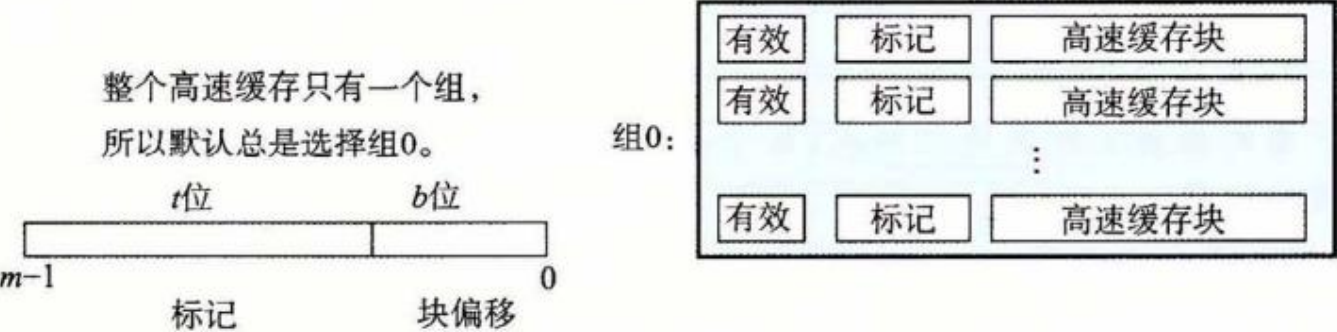


图 6-36 全相联高速缓存中的组选择。注意没有组索引位

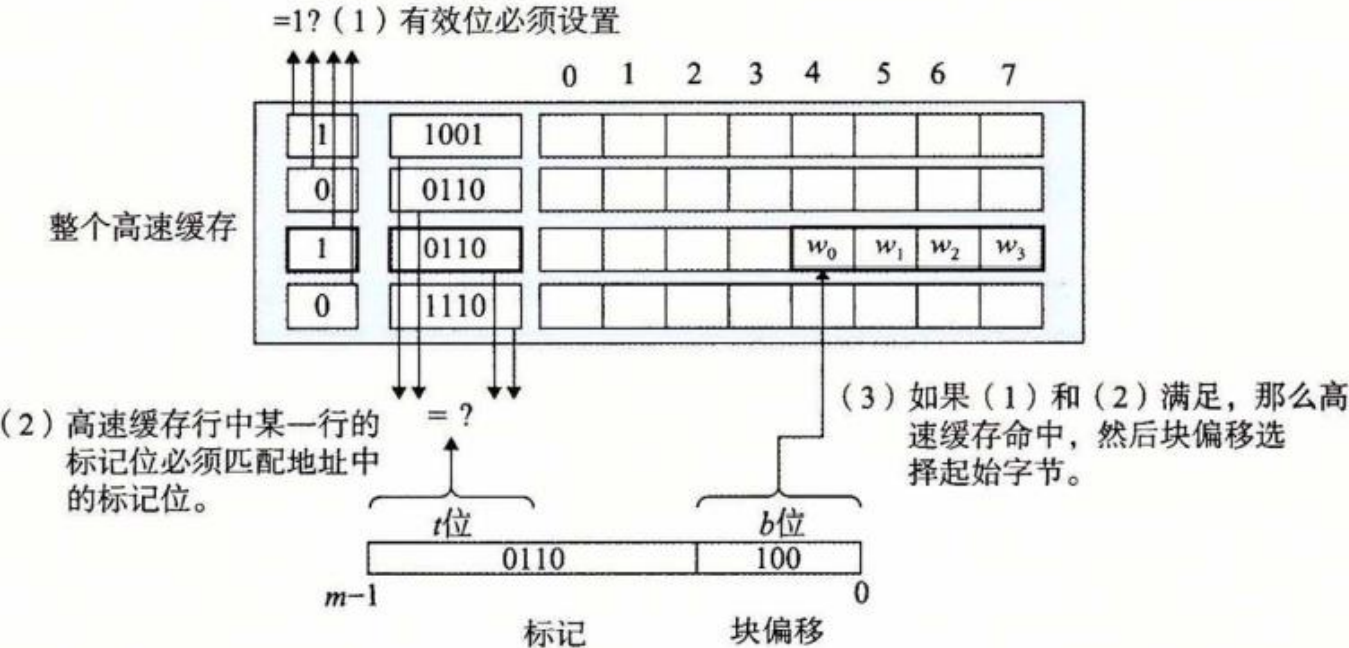


图 6-37 全相联高速缓存中的行匹配和字选择

- 一个包含所有高速缓存行的组
- 不需要组索引位，默认为0组
- 因为高速缓存电路必须并行搜索许多相匹配的标记，构造一个又大又快的相联高速缓存很困难并且昂贵。因此，全相联高速缓存只适合做小的高速缓存，例如虚拟内存系统中的快表TLB

组相联高速缓存

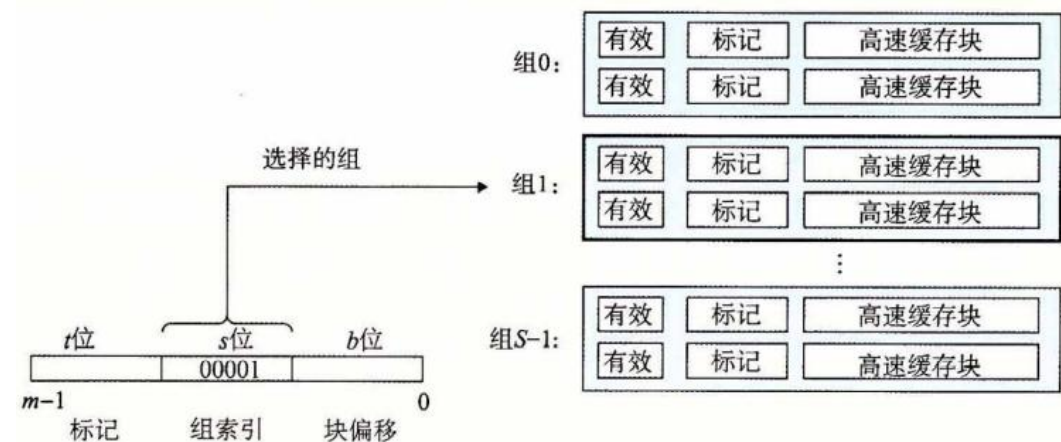


图 6-33 组相联高速缓存中的组选择

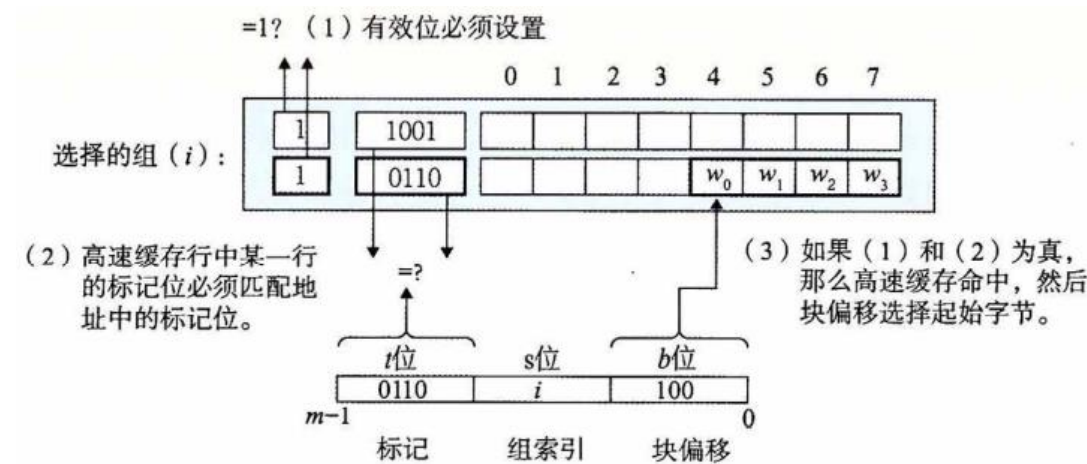


图 6-34 组相联高速缓存中的行匹配和字选择

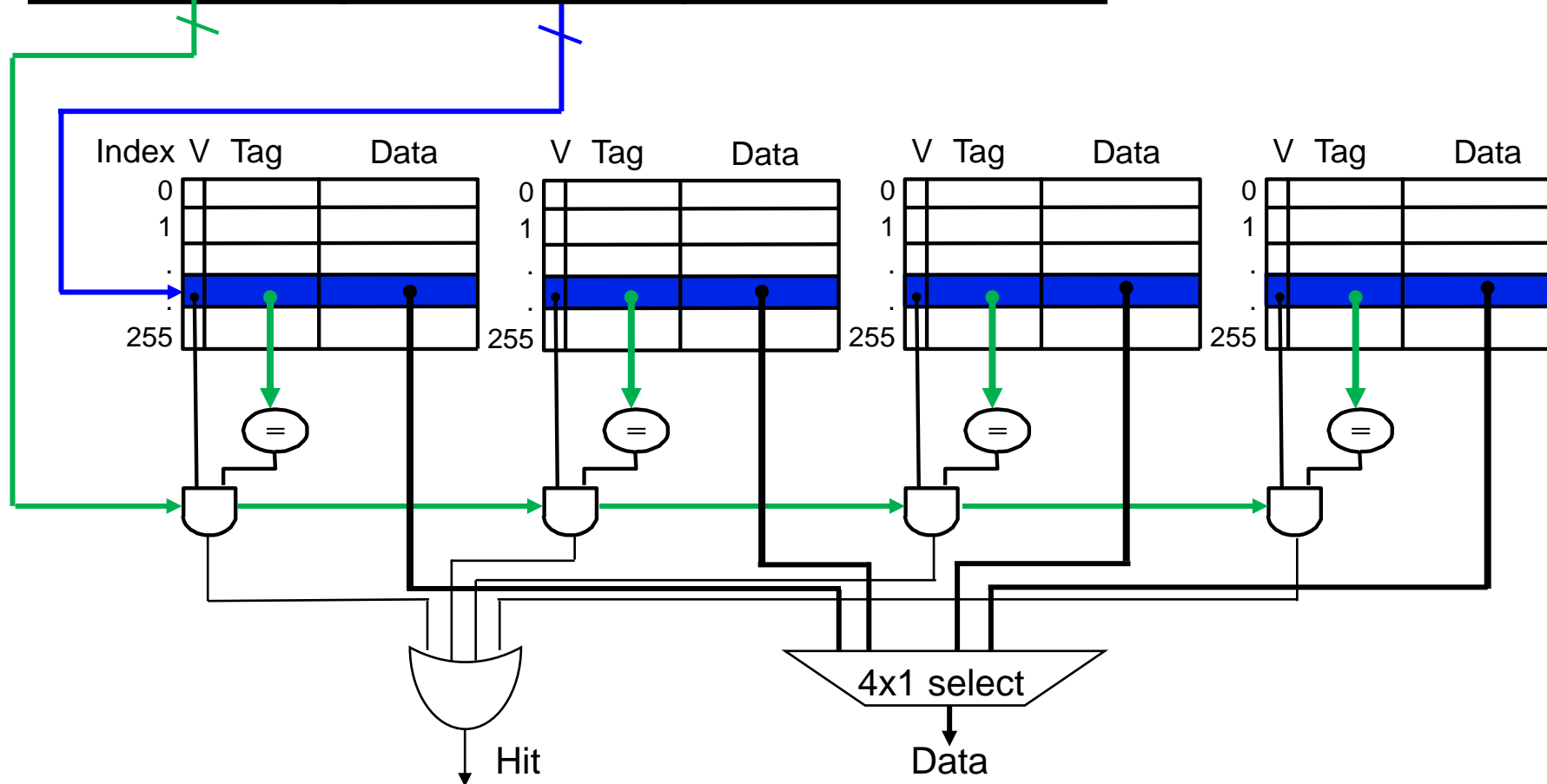
- 行匹配时必须检查多个行的标记位和有效位
- 行替换策略:
 - 随机替换策略
 - LFU (最不常使用)
 - LRU (最近最少使用)

Set-Associative Cache

MIPS Example: 4-Way SA

- One word/block, Cache size = 1K words
- $2^8 = 256$ sets each with four ways (each with one block)

Address of word:

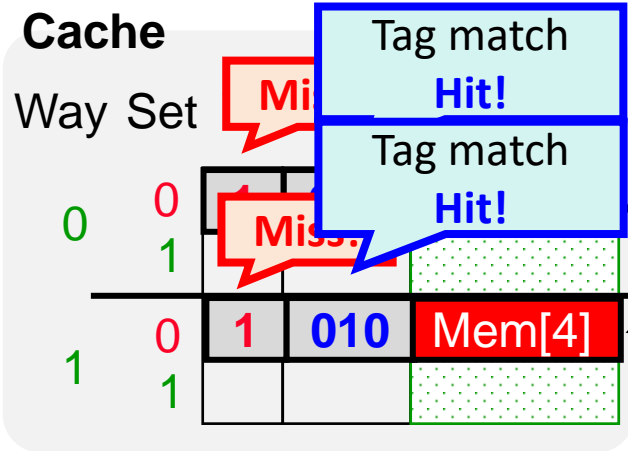


Set-Associative Cache

A Simple Example

➤ Same example w/ direct mapped

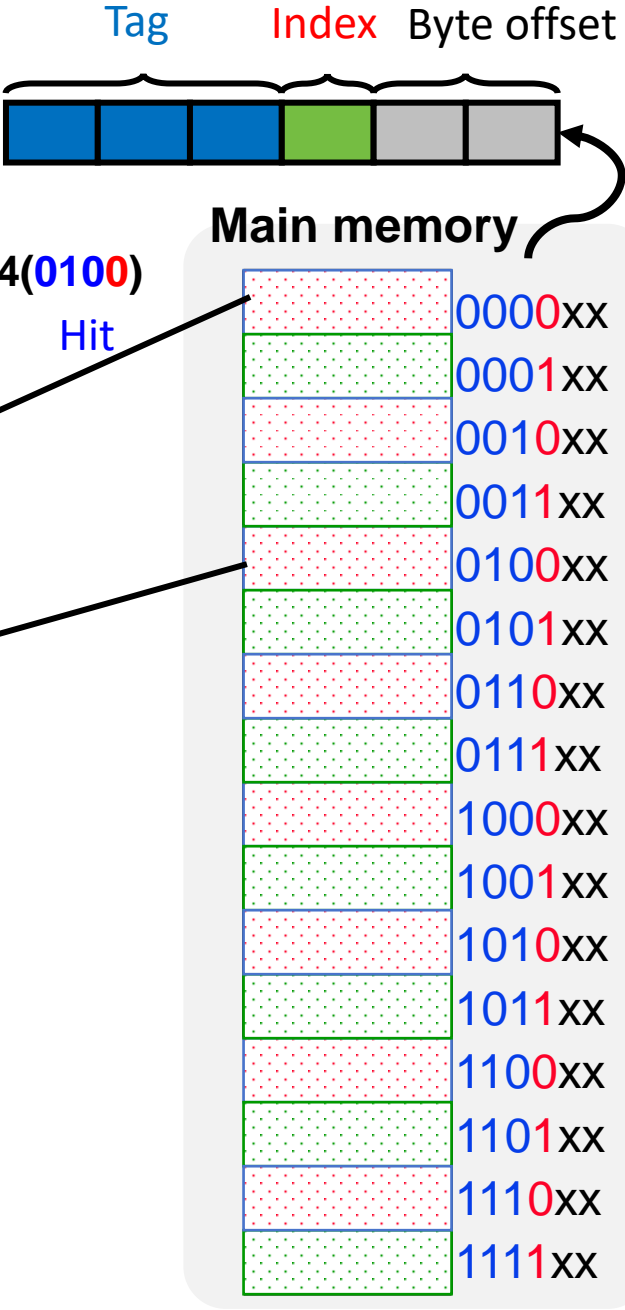
0(0000) 4(0100) 0(0000) 4(0100) 0(0000) 4(0100) 0(0000) 4(0100)
Miss Miss Hit Hit Hit Hit Hit Hit



→ Start with an empty cache, all blocks initially marked as not valid

8 request, 2 misses

Reduces 6 misses than one-word direct mapped



Cache Write

- 写命中:

直写write through: 在高速缓存更新了w的副本后立即将w写回到紧接着的低一层中。

写回write back: 当替换算法要替换这个数据块时再写回; 需要维护一个额外的修改位。

- 写不命中:

写分配write allocate: 加载相应的低一层的块到高速缓存中, 然后更新这个高速缓存块。

非写分配not-write-allocate: 避开高速缓存直接把这个字写到低一层中

- write back+write allocate write through+not-write-allocate

Cache Write

Inclusion Policy

- Inclusive multilevel cache:
 - Inner cache can only hold lines also present in outer cache
 - External coherence snoop access need only check outer cache
 - *Exclusive* multilevel caches:
 - Inner cache may hold lines not in outer cache
 - Swap lines between inner/outer caches on miss
 - Used in AMD Athlon with 64KB primary and 256KB secondary cache
- L1&L2:inclusive
 - L2&L3:exclusive

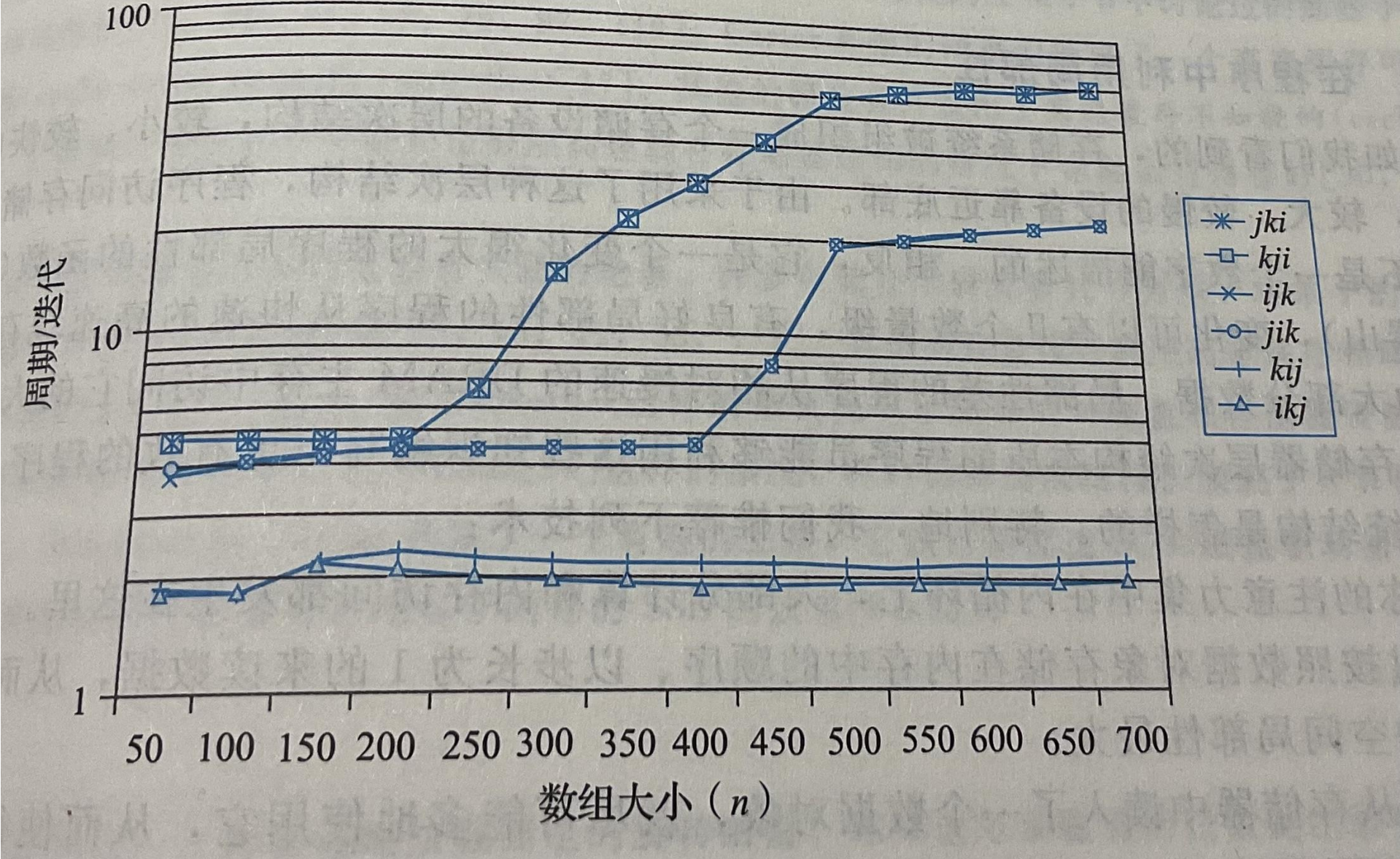
高速缓存性能

- 不命中率(miss rate): 在一个或一部分程序执行期间, 内存引用不命中的比率 计算公式: 不命中数量/引用数量。
- 命中率(hit rate): 命中的内存引用比率。等于 $1 - \text{miss rate}$ 。
- 命中时间(hit time): 从高速缓存传送一个字到CPU所需的时间, 包括组选择、行匹配、字选择。对L1高速缓存来说, 数量级是几个时钟周期。
- 不命中处罚(miss penalty): 由于不命中所需要的额外的时间。L1不命中需要从L2得到服务处罚, 一般10个周期; 从L3处得到服务处罚, 50个周期; 从主存得到服务处罚, 200个周期。

Q: 已知L1访问时间为2个周期, L2服务处罚是10个周期, 那么L1的一个不命中在L2中命中一共要花多少个时钟周期?

- 书上用矩阵乘法的例子展示了较高的不命中率比加载和存储对运行时间的影响都要大, 我们应该尝试写局部性好的程序 (例如步长为1的引用模式, 重新排列循环)

高速缓存性能



性能影响因素

- 高速缓存大小：较大的高速缓存提高命中率但可能增加命中时间
- 块大小：较大的块能利用可能存在的空间局部性提高命中率，但也意味着高速缓存行数越少，也可能增加不命中处罚
- 相联度(E)：较高的相联度降低由于冲突不命中产生抖动的可能，但会增加成本、命中时间和不命中处罚
- 写策略：直写容易实现，不命中开销小；写回减少传送的数量

性能影响因素

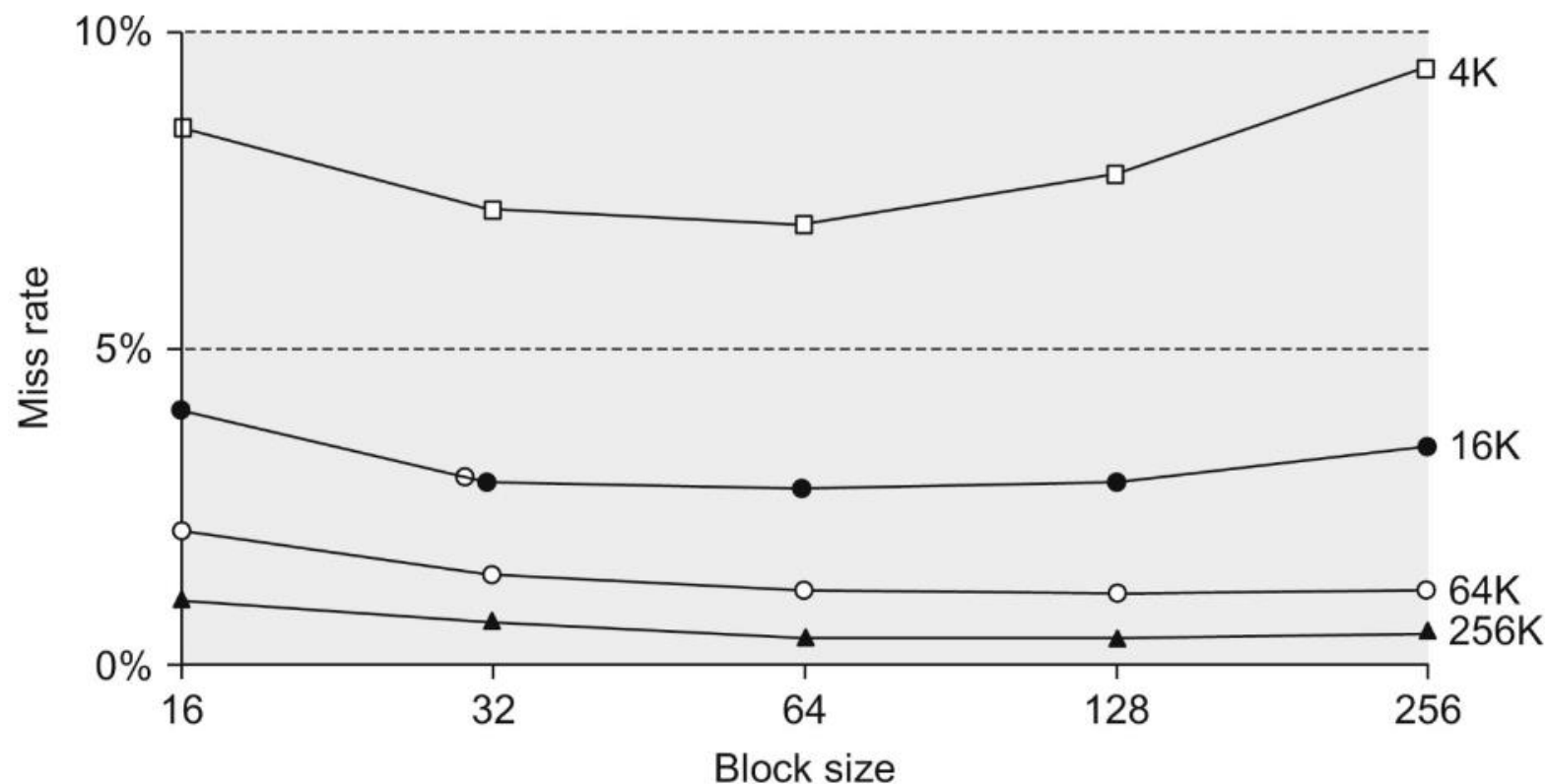


Figure B.10 Miss rate versus block size for five different-sized caches.

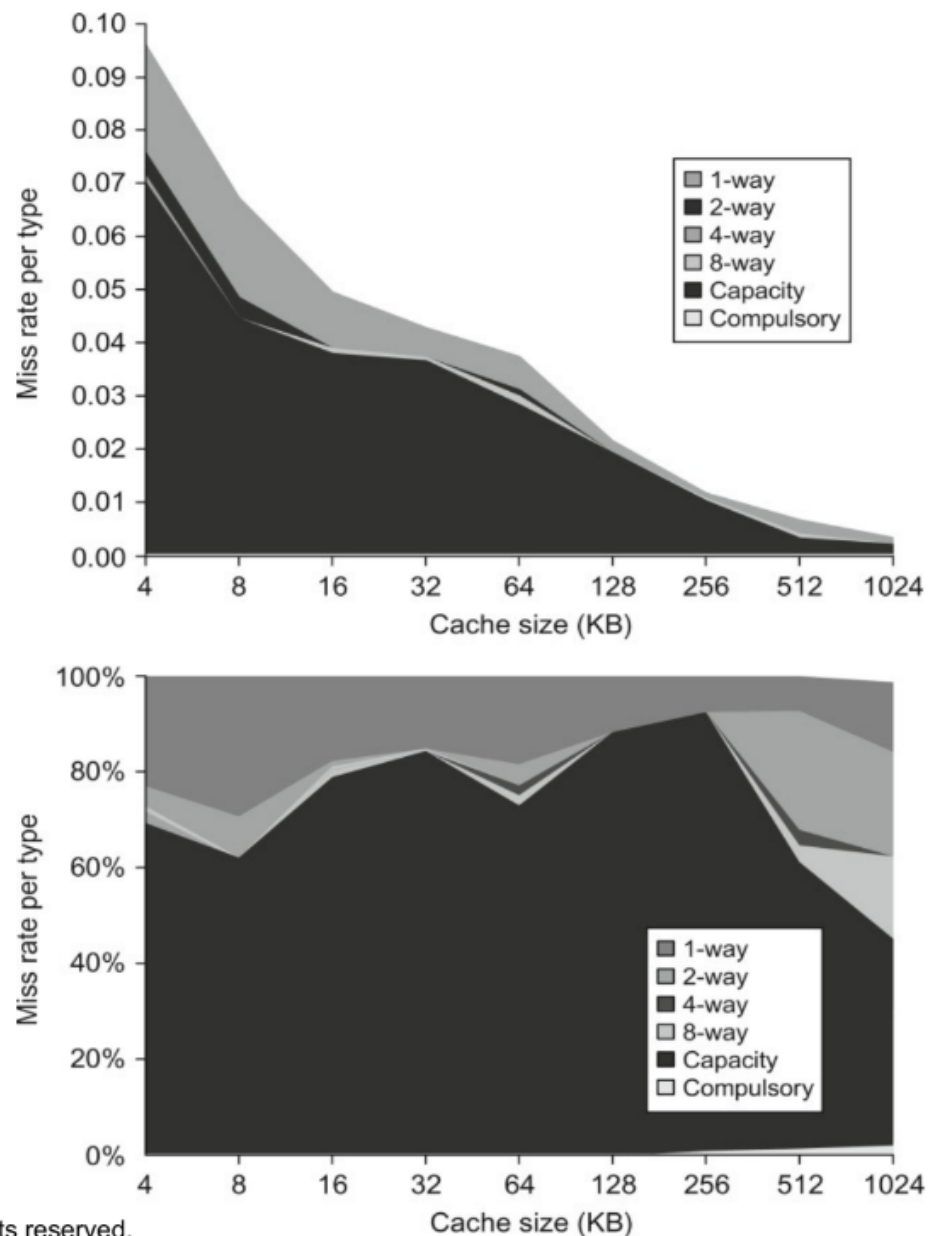
Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92 on a DECstation 5000 (Gee et al. 1993).

性能影响因素

- 高速缓存大小：较大的高速缓存提高命中率但可能增加命中时间
- 块大小：较大的块能利用可能存在的空间局部性提高命中率，但也意味着高速缓存行数越少，也可能增加不命中处罚
- 相联度(E)：较高的相联度降低由于冲突不命中产生抖动的可能，但会增加成本、命中时间和不命中处罚
- 写策略：直写容易实现，不命中开销小；写回减少传送的数量

性能影响因素

Figure B.9 Total miss rate (top) and distribution of miss rate (bottom) for each size cache according to the three C's for the data in Figure B.8. The top diagram shows the actual data cache miss rates, while the bottom diagram shows the percentage in each category. (*Space allows the graphs to show one extra cache size than can fit in Figure B.8.*)



性能影响因素

- 高速缓存大小：较大的高速缓存提高命中率但可能增加命中时间
- 块大小：较大的块能利用可能存在的空间局部性提高命中率，但也意味着高速缓存行数越少，也可能增加不命中处罚
- 相联度(E)：较高的相联度降低由于冲突不命中产生抖动的可能，但会增加成本、命中时间和不命中处罚
- 写策略：直写容易实现，不命中开销小；写回减少传送的数量

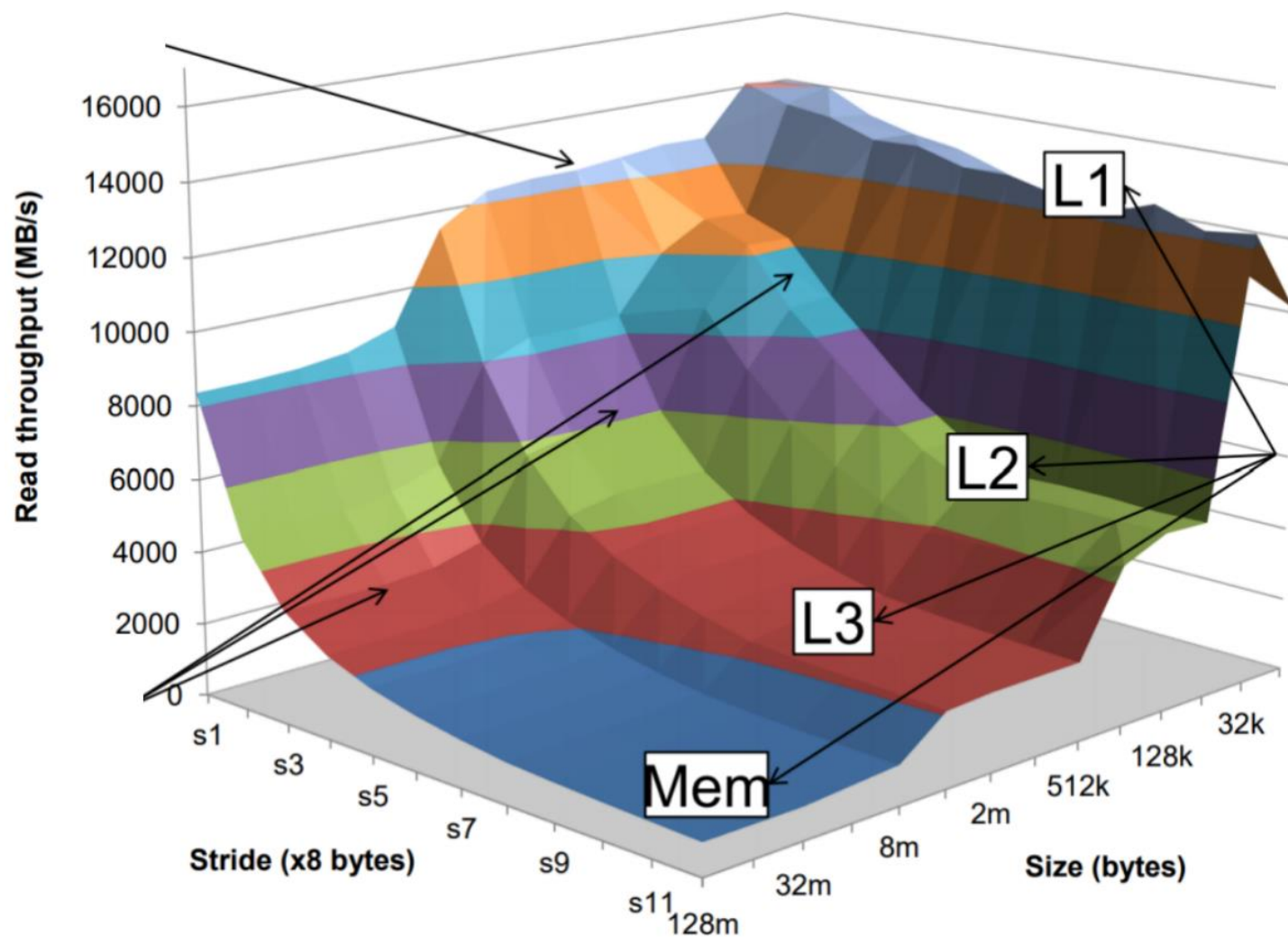
存储器山

```
1 long data[MAXELEMS]; /* The global array we use */
2
3 /* test - Iterate over first "elems" elements of array "data" with
4 * stride of "stride", using 4 x 4 loop unrolling.
5 */
6 int test(int elems, int stride)
7 {
8     long i, sx2 = stride*2, sx3 = stride*3, sx4 = stride*4;
9     long acc0 = 0, acc1 = 0, acc2 = 0, acc3 = 0;
10    long length = elems;
11    long limit = length - sx4;
12
13    /* Combine 4 elements at a time */
14    for (i = 0; i < limit; i += sx4) {
15        acc0 = acc0 + data[i];
16        acc1 = acc1 + data[i+stride];
17        acc2 = acc2 + data[i+sx2];
18        acc3 = acc3 + data[i+sx3];
19    }
20
21    /* Finish any remaining elements */
22    for (; i < length; i+=stride) {
23        acc0 = acc0 + data[i];
24    }
25    return ((acc0 + acc1) + (acc2 + acc3));
26 }
27
28 /* run - Run test(elems, stride) and return read throughput (MB/s).
29 * "size" is in bytes, "stride" is in array elements, and Mhz is
30 * CPU clock frequency in Mhz.
31 */
32 double run(int size, int stride, double Mhz)
33 {
34     double cycles;
35     int elems = size / sizeof(double);
36
37     test(elems, stride); /* Warm up the cache */
38     cycles = fcyc2(test, elems, stride, 0); /* Call test(elems, stride) */
39     return (size / stride) / (cycles / Mhz); /* Convert cycles to MB/s */
40 }
```

code/mem/mountain/mountain.c

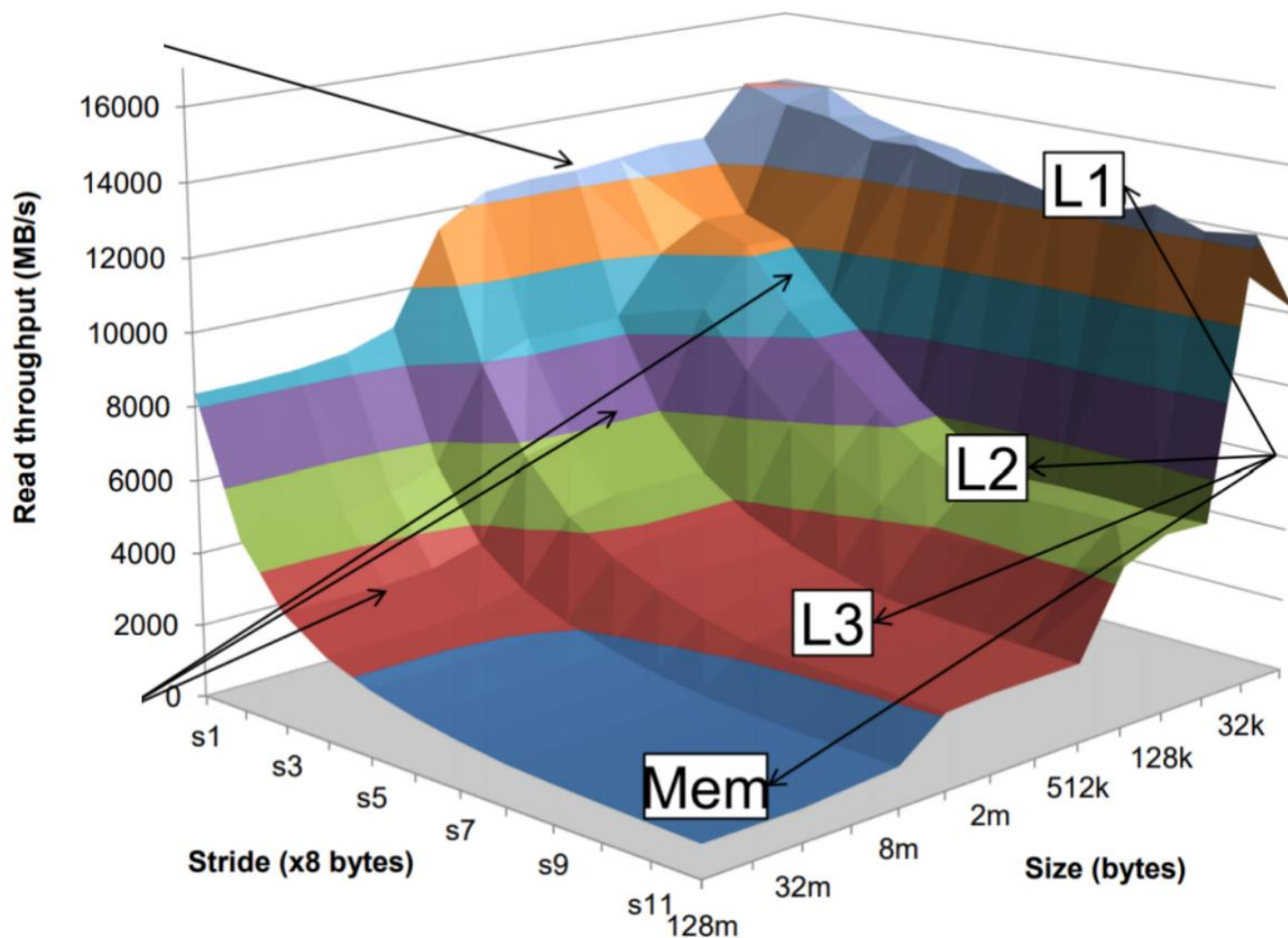
-40 测量和计算读吞吐量的函数。我们可以通过以不同的 size(对应于时间局部性)和 stride(对应于空间局部性)的值来调用 run 函数,产生某台计算机的存储器山

存储器山



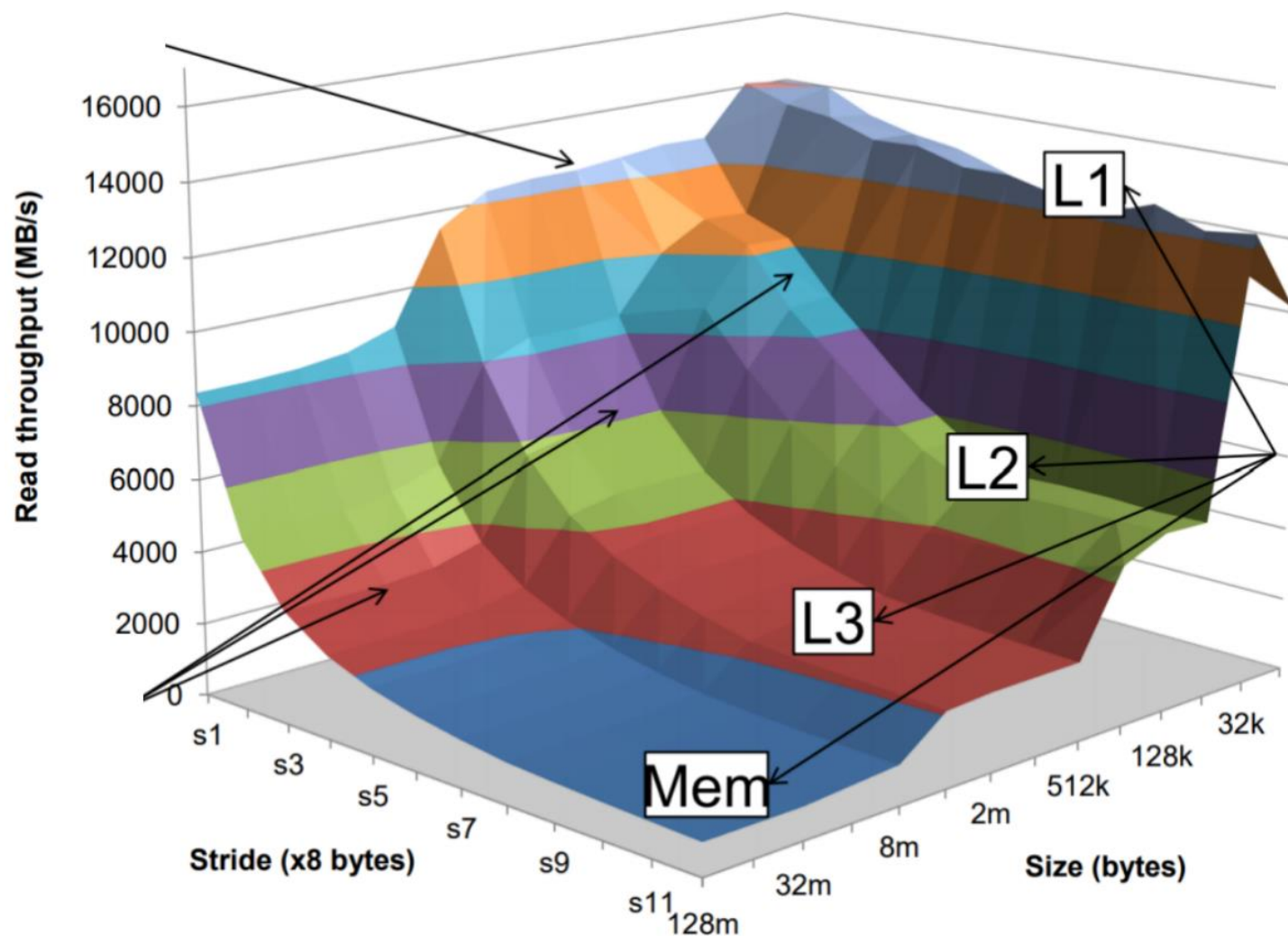
- 读吞吐量（读带宽）单位MB/s
- Size越小，得到的工作集越小，时间局部性越好
- Stride越小，空间局部性越好

存储器山



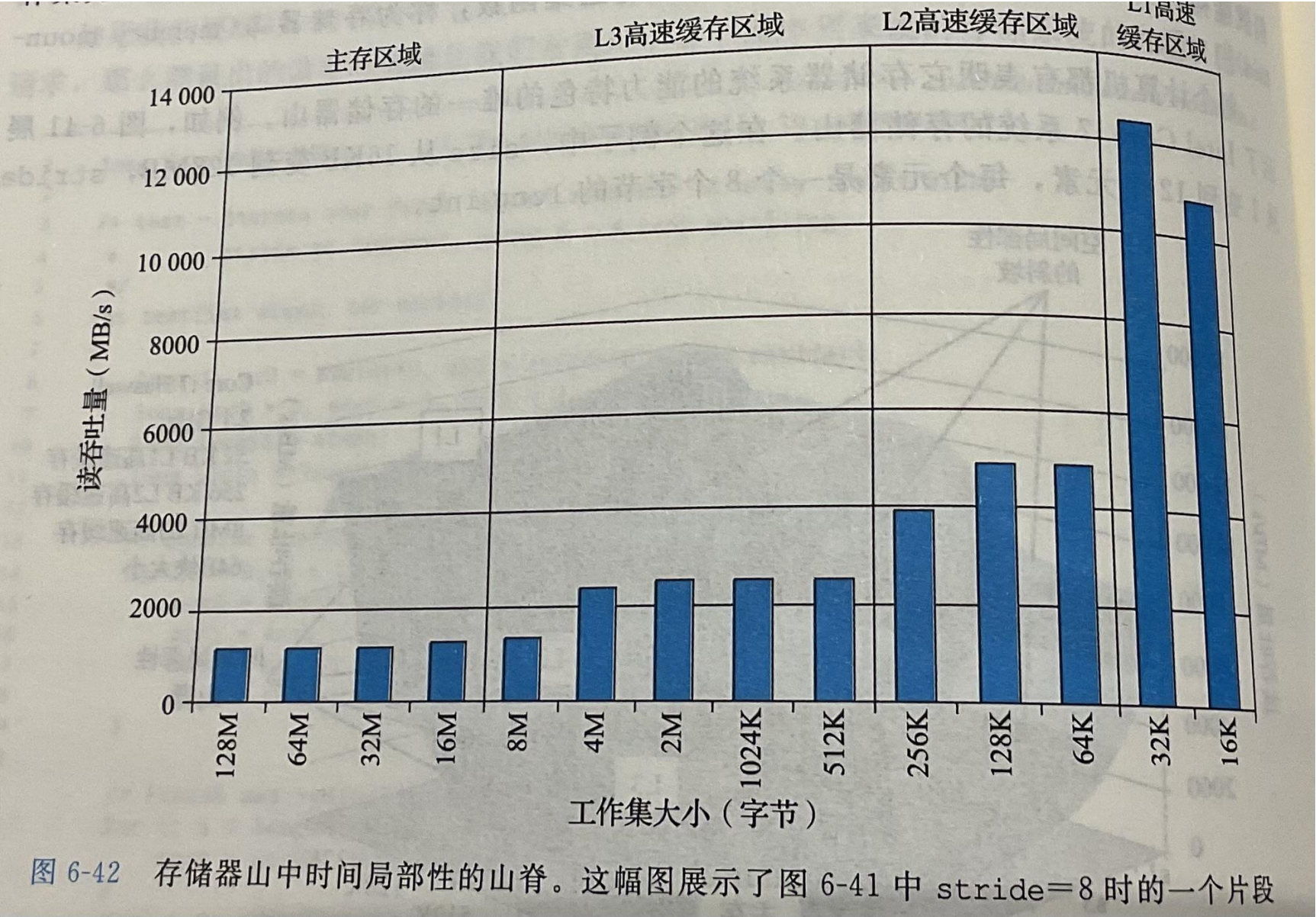
- 四条山脊的读吞吐量差别非常大
- 在L2、L3和主存山脊上，随着步长的增加有空间局部性的斜坡。我们观察到，即使工作集太大，不能全都装进任意一个高速缓存时，空间局部性仍能弥补时间局部性。

存储器山

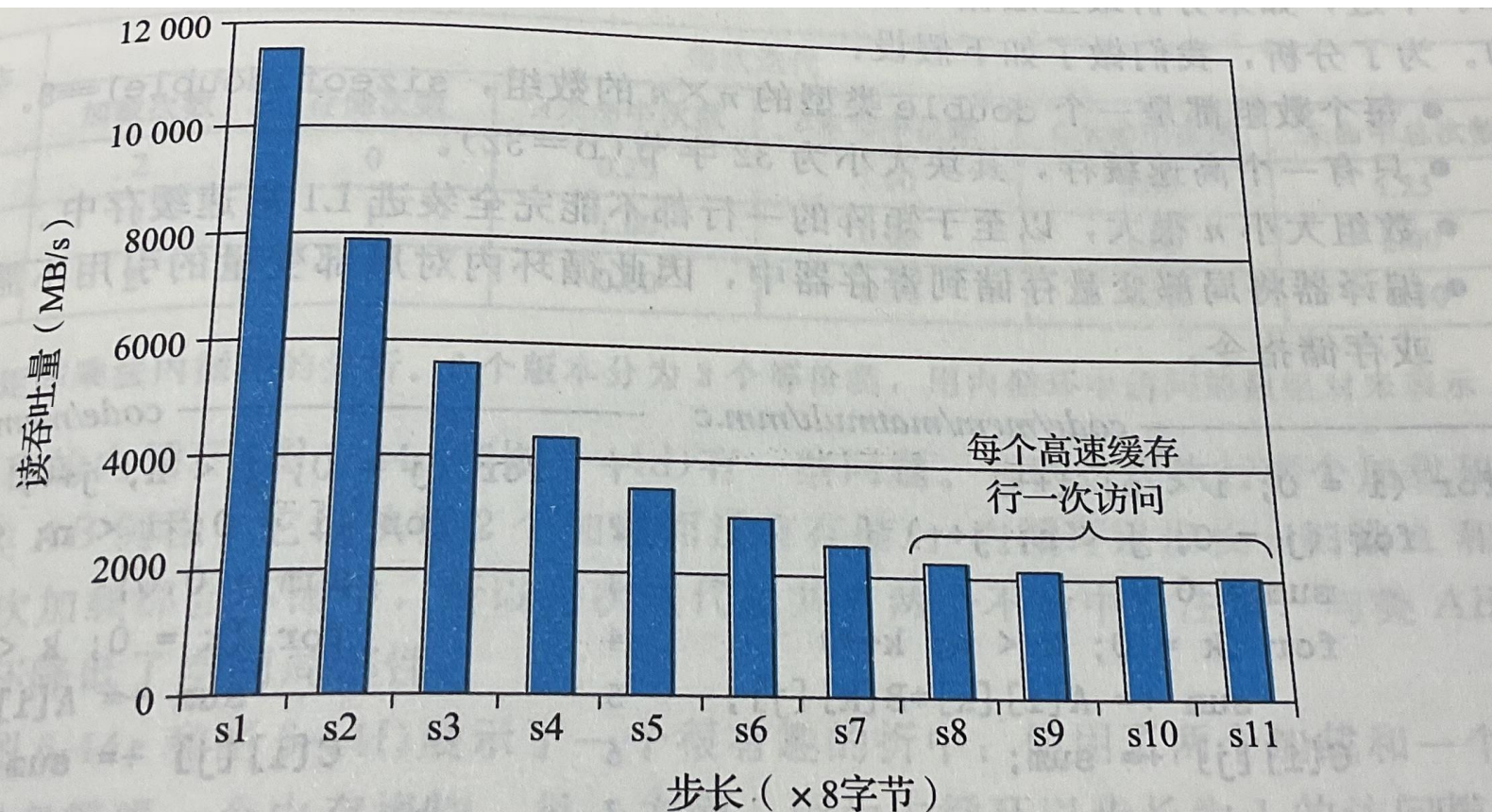


- 垂直于步长轴的平坦的山脊线，读吞吐量保持不变为12GB/s，即使工作集超出了L1和L2的大小，因为硬件预取机制——自动识别顺序的、步长为1的引用模式，试图在一些块被访问以前将它们存储到高速缓存中。

存储器山



存储器山



一个空间局部性的斜坡。这幅图展示了图 6-41 中大小=4MB 时的一个片段

- 4M的工作集能够放在L3高速缓存中，但是对L2高速缓存来说太大了。
- 随着步长的增加，L2不命中与L2命中的比值增加。
- 一旦步长达到8个字（64个字节，所有高速缓存的数据块大小都是64B），每个读请求在L2都不会被命中，都要重新从L3读取），由L3传送高速缓存块到L2的速率决定。

Thank you for listening!