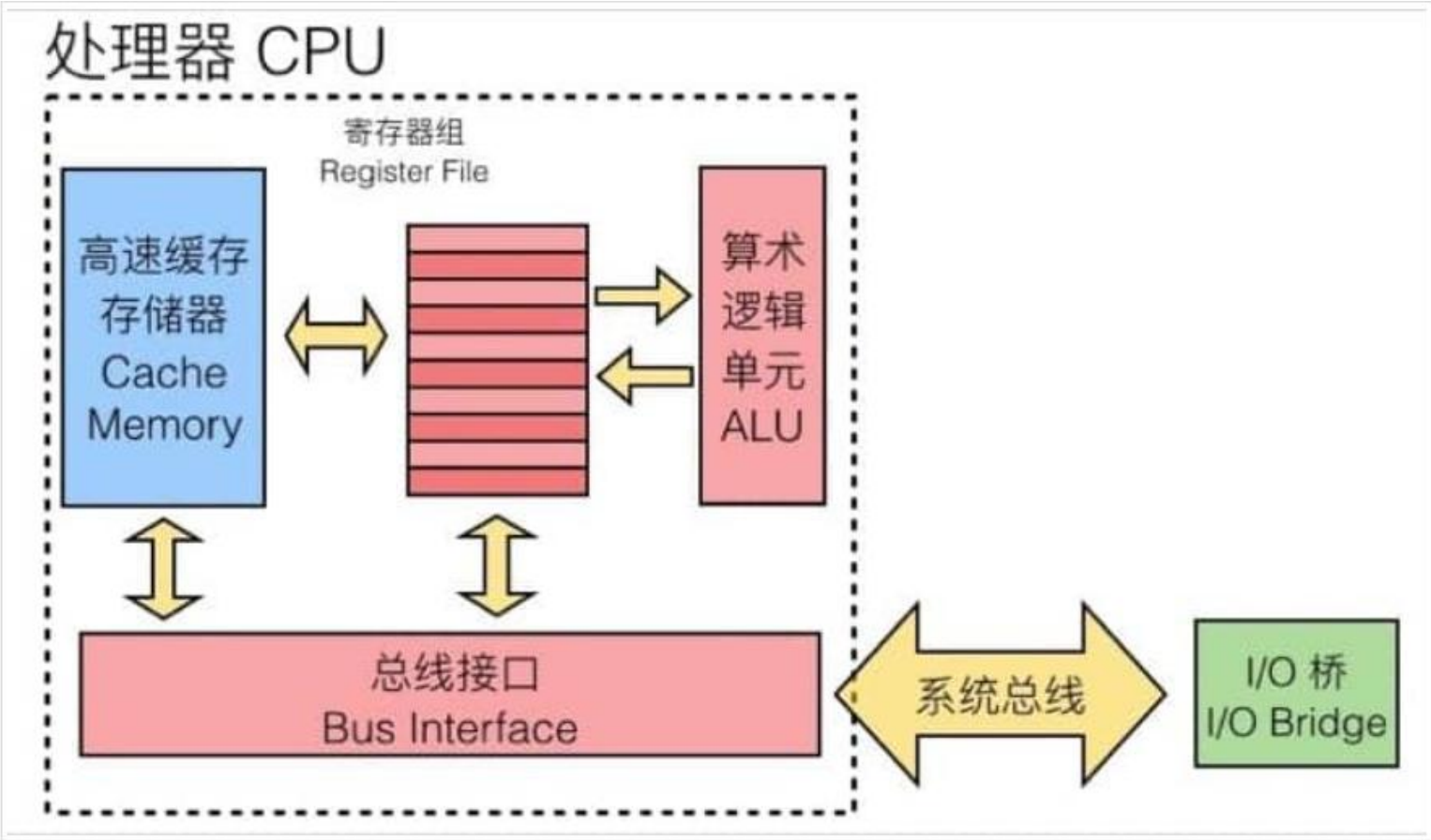
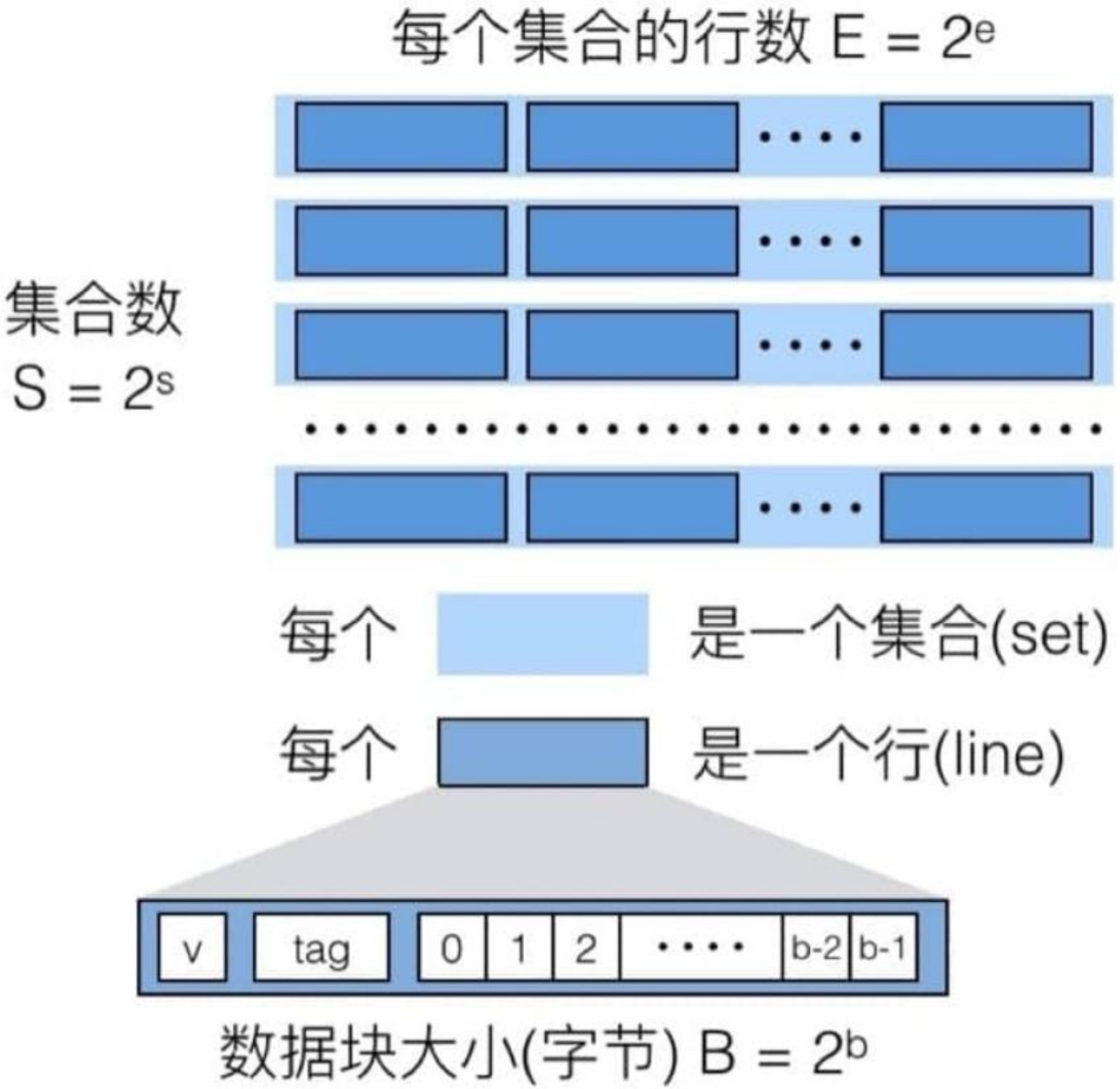
# 高速缓存 存储器
# Cache Memory

连昊




| 类型 | 缓存什么 | 被缓存在何处 | 延迟（周期数） | 由谁管理 |
|---|---|---|---|---|
| L1高速缓存 | 64字节块 | 芯片上的L1高速缓存 | 4 | 硬件 |
| L2高速缓存 | 64字节块 | 芯片上的L2高速缓存 | 10 | 硬件 |
| L3高速缓存 | 64字节块 | 芯片上的L3高速缓存 | 50 | 硬件 |

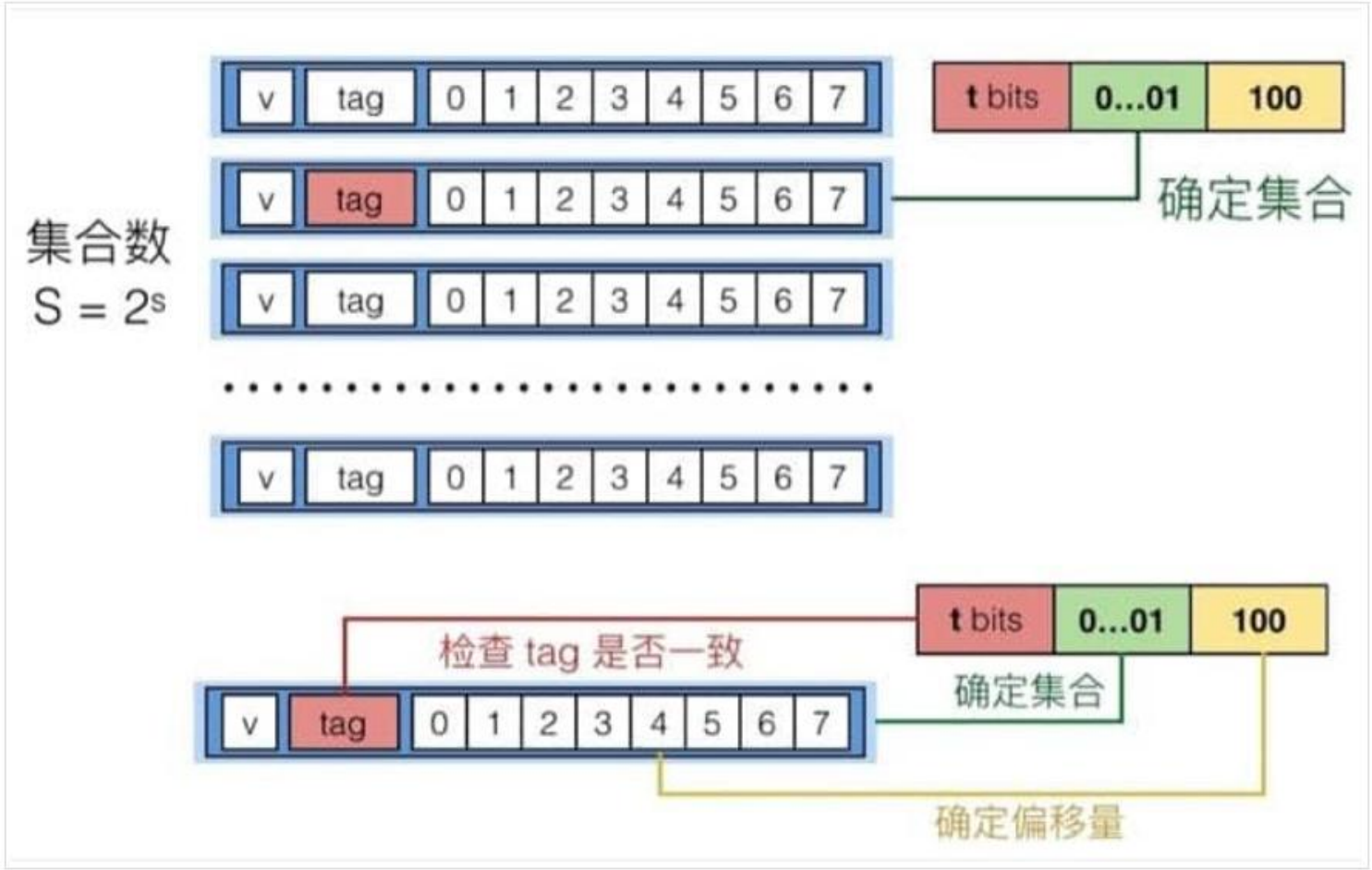



- 高速缓存的大小（容量）
- C=S×E×B

- 根据E的不同对高速缓存分类
- E=1: 直接映射高速缓存
- 1<E<C/B: 组相联高速缓存
- E=C/B： 全相联高速缓存

# 数据读取：以直接映射高速缓存为例

三步走：（1）组选择 （2）行匹配 （3）字抽取




- 缓存不命中时，用新取出的行直接替代当前行

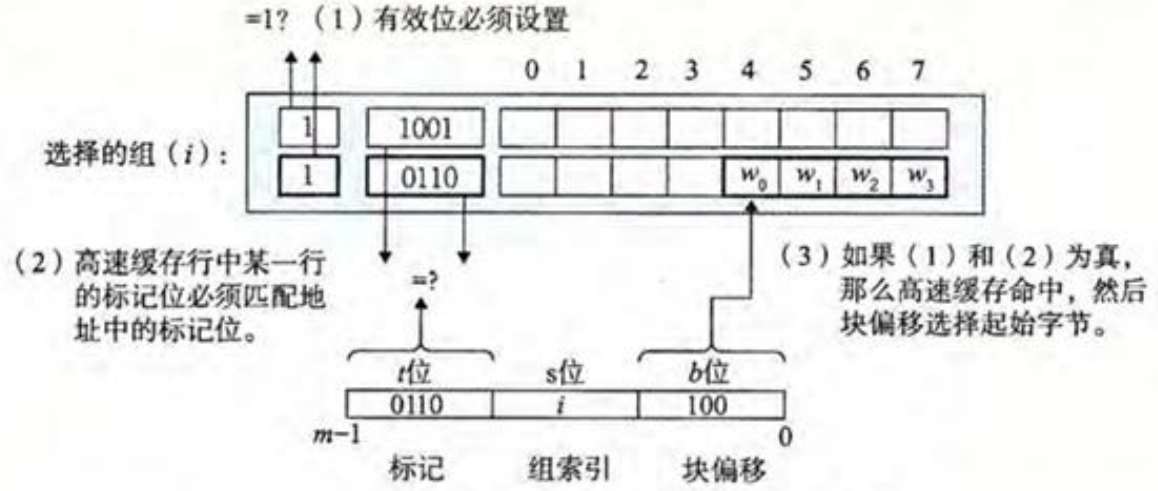



图 6-34 组相联高速缓存中的行匹配和字选择

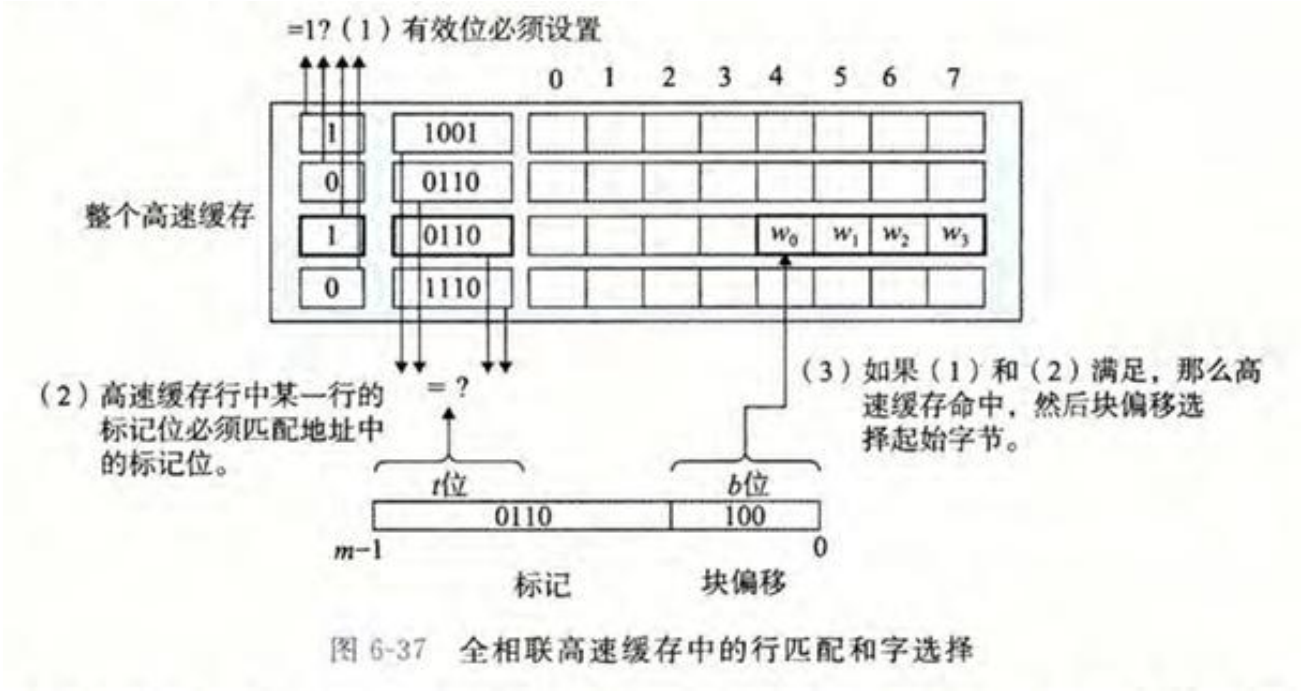


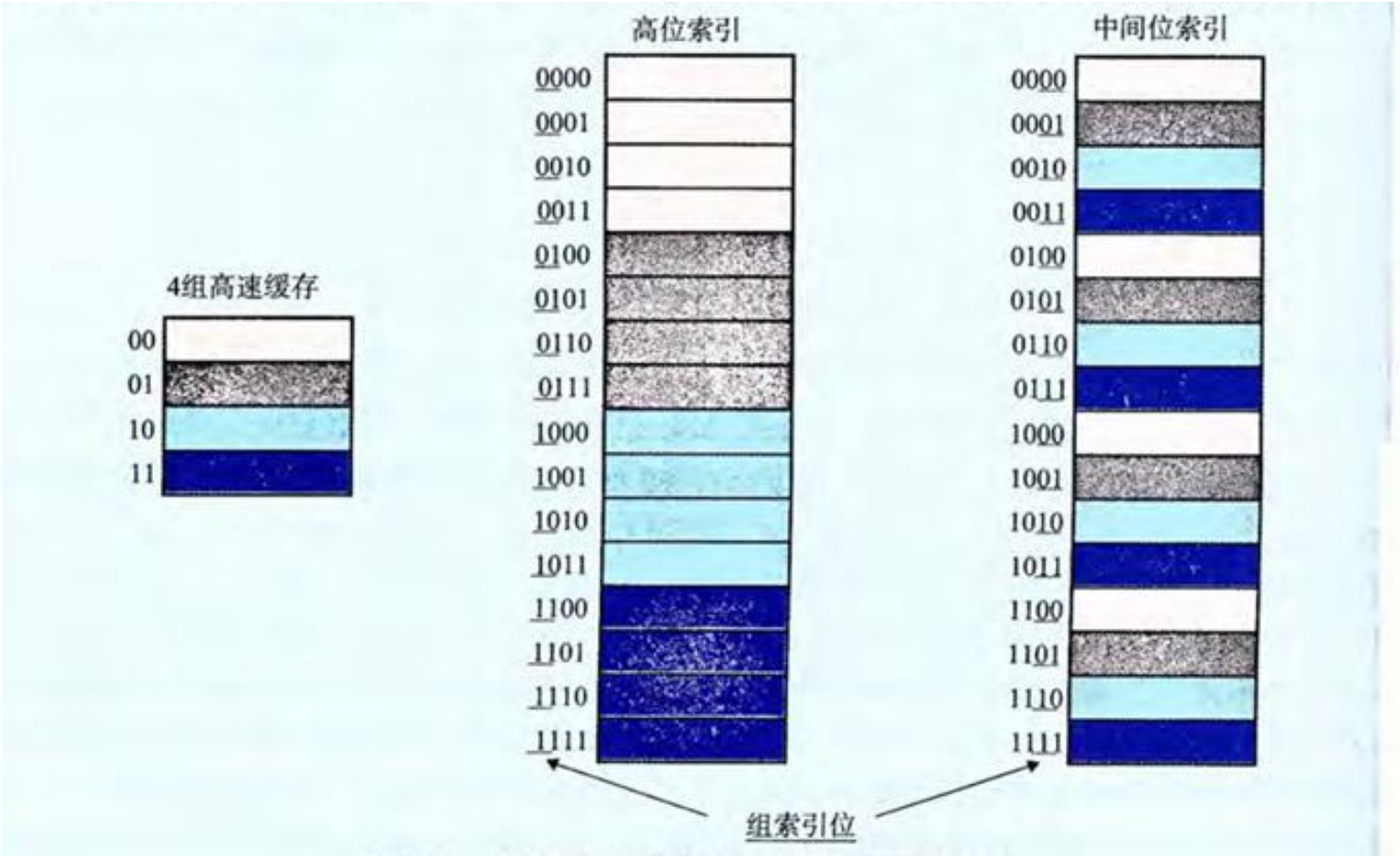

图 6-37 全相联高速缓存中的行匹配和字选择

- 组相联高速缓存
- 搜索组中的每一行进行匹配
- 缓存不命中时，随机选择要替换的行，或者采用局部性友好的优化策略

- 全相联高速缓存
- 地址中没有组索引位
- 搜索组中的每一行进行匹配

# 思考题

数据读取时，我们按照组(set）、标记（tag）、块偏移（block）的顺序进行寻址。为什么我们编写地址时按照的是标记（tag）、组(set）、块偏移（block）的顺序?




- 减少冲突不命中的发生
- 更充分的利用高速缓存的各个块

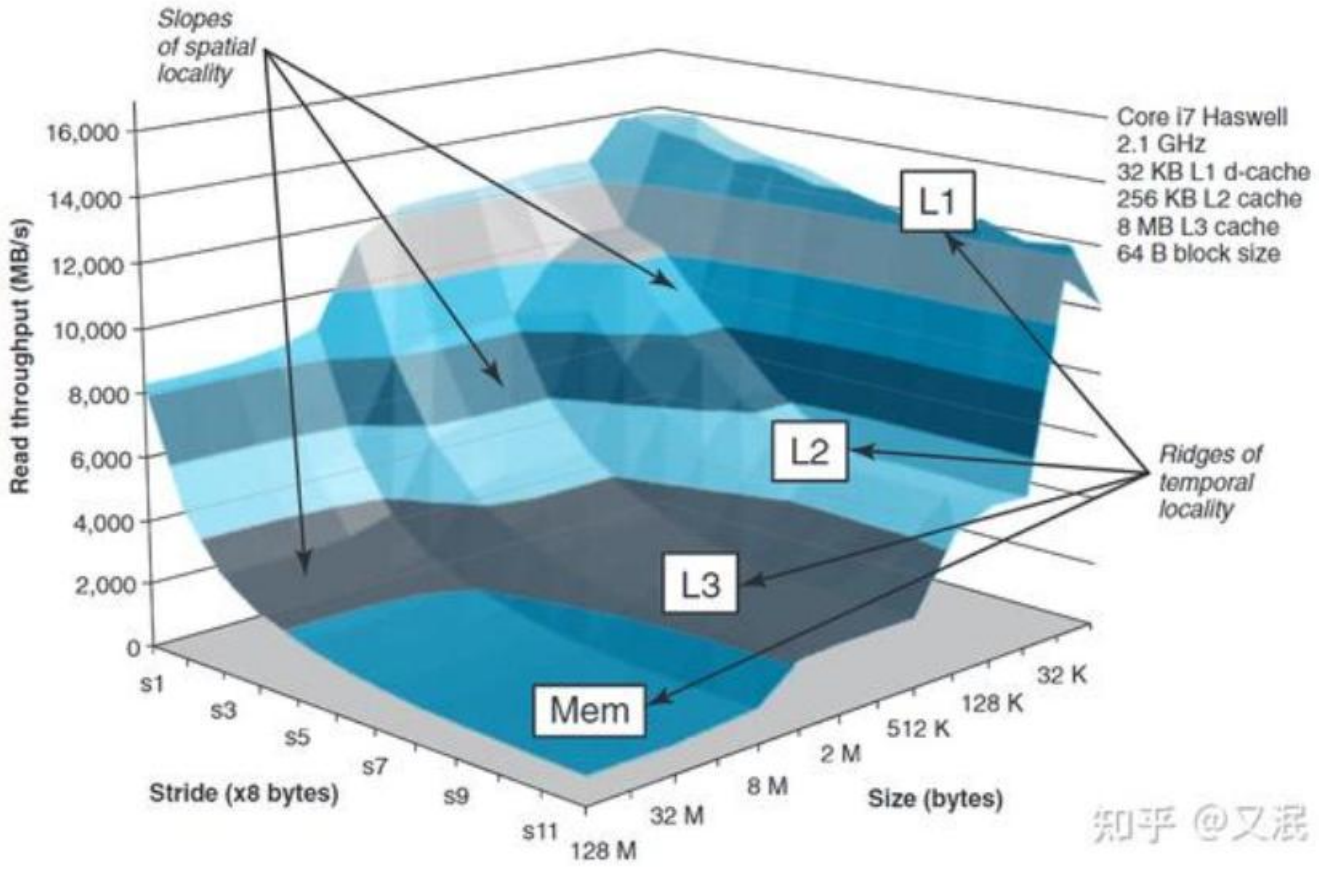
# 写入

- 发生写命中（write hit）时：
- 直写（Write-through）：命中后更新缓存，同时写入到内存中
- 写回（Write-back）：命中后更新缓存，但直到这个缓存需要被置换出去，才写入到内存中
- 写回更好的利用了局部性，能显著地减少总线流量，但需要额外维护一个修改位(dirty bit)，表明其是否被修改过

- 发生写不命中时：
- 写分配（Write-allocate）：将第一层的块载入到缓存中，并更新缓存（能更好的利用空间局部性）
- 非写分配（No-write-allocate）：直接写入到内存中，不载入到缓存

- 通常情况下：
- 直写+非写分配
- 写回+写分配

# 存储器山

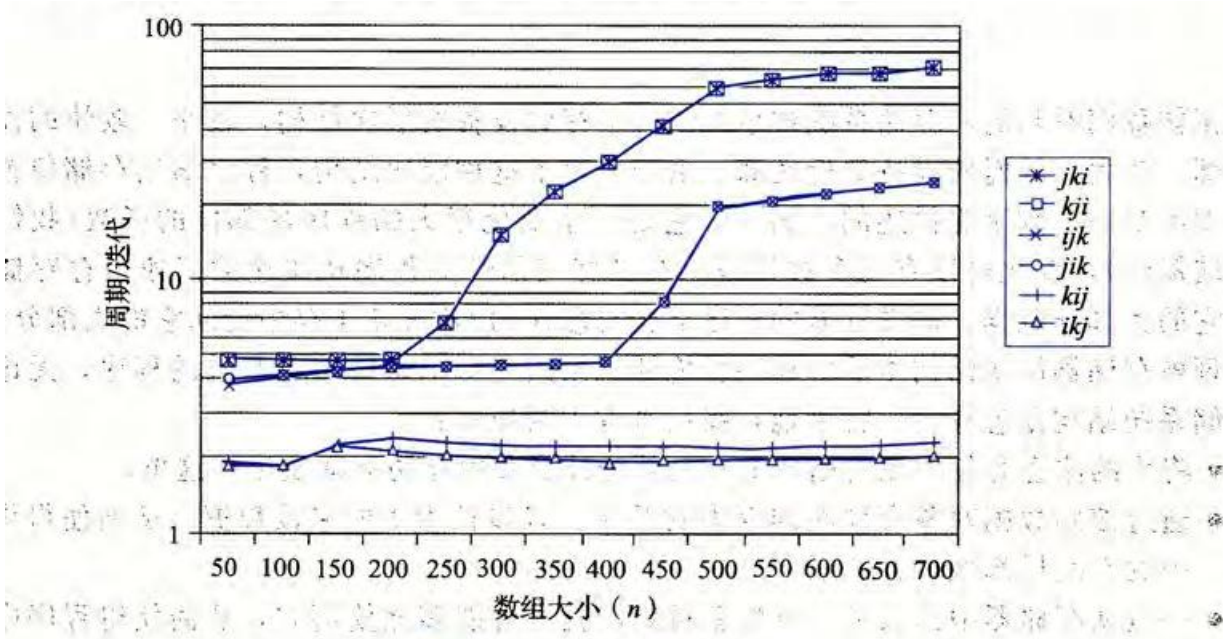



- 工作集大小：时间局部性
- 步长：空间局部性

- 硬件预取机制：对于步长为1的情况，读吞吐量在L3时依然能保持相对稳定

# 程序优化：编写高速缓存友好代码

| 矩阵乘法版本（类） | 每次迭代 | | | | | |
|---|---|---|---|---|---|---|
| | 加载次数 | 存储次数 | *A*未命中次数 | *B*未命中次数 | *C*未命中次数 | 未命中总次数 |
| *ijk* & *jik* (*AB*) | 2 | 0 | 0.25 | 1.00 | 0.00 | 1.25 |
| *jki* & *kji* (*AC*) | 2 | 1 | 1.00 | 0.00 | 1.00 | 2.00 |
| *kij* & *ikj* (*BC*) | 2 | 1 | 0.00 | 0.25 | 0.25 | 0.50 |




- 让常见情况运行的更快：主要优化核心函数的内循环
- 尽量减少循环内部的缓存不命中次数：反复引用局部变量、尽量利用步长为1的引用
- 分块技术

感谢聆听！