超星学习通 QR Code

- 作业会放到里面发布

缓存 (Cache)

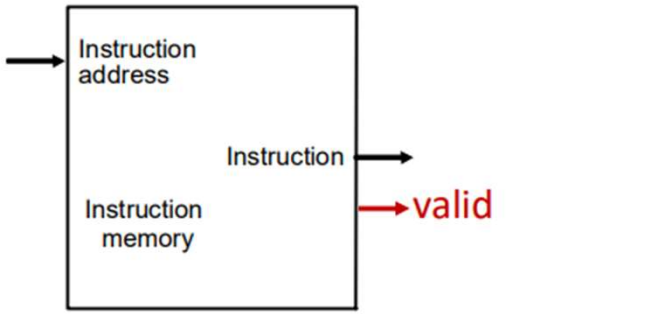
COA: 计算机系统与结构
Sep. 2, 2025

在计算机领域，缓存是一种通用概念：任何能够“记住”经常重复计算结果的结构，都可以称作缓存。通过存储这些结果，我们就可以避免每次都从零开始重新计算，从而节省时间和资源。例如：网页缓存（web cache）。

计算机体系结构中的缓存（Cache）

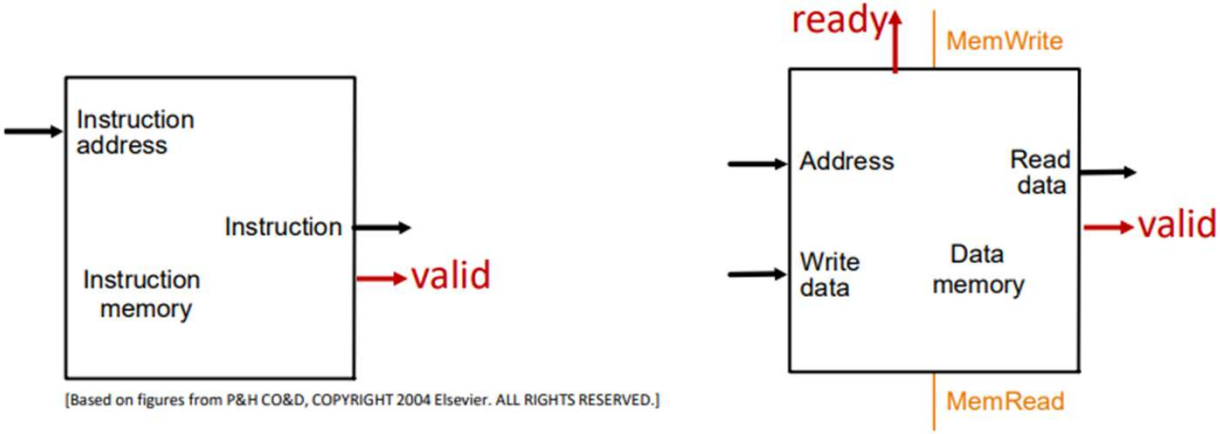
- 缓存是一个“不可见的”、自动管理的内存层次结构
- CPU 首先查找缓存中的数据
- 缓存的核心功能：
  - 缓存会在一块小而快的存储器中，保存一些频繁访问的 DRAM 数据副本
- 程序的预期：
  - 当你在代码中写 x = M[A] 时，你只关心能不能读到数据，而不需要知道它是从 L1 缓存、L2 缓存还是 DRAM 中取出来的。
- 程序一般不需要为缓存做特别处理，但是
  - 多核处理器访问同一块内存；
  - DMA（直接内存访问）绕过缓存直接修改数据；

缓存类型：指令缓存（I-Cache）



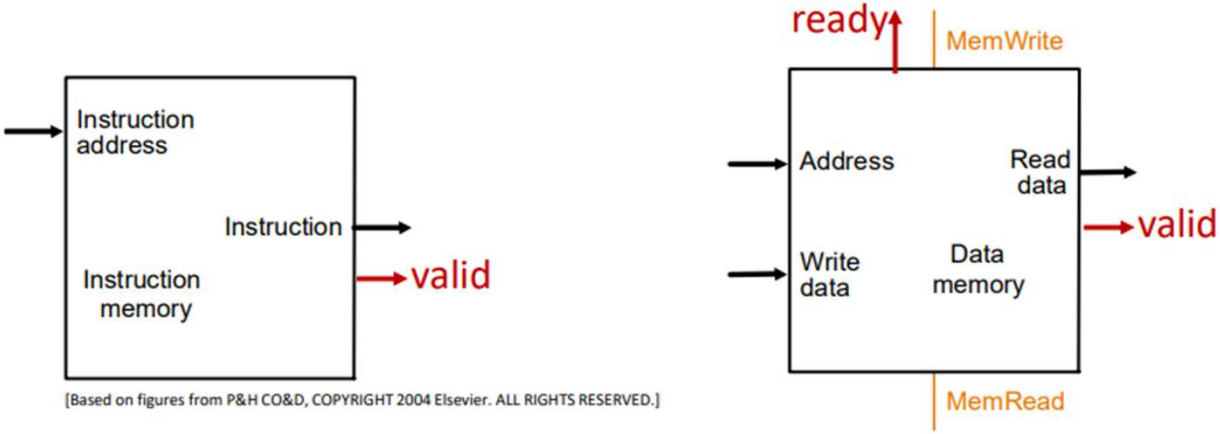
- 指令缓存（Instruction Cache，I-Cache）
  - 专门用来缓存程序指令的小型高速存储器。
  - 当 CPU 需要执行一条指令时，它会先从 I-Cache 中查找对应的指令，而不是直接去主存取。
  - 左边的箭头：CPU → 指令缓存，CPU 把想要取的指令地址发给指令缓存。
  - 右边的箭头 ：指令缓存 → CPU，如果缓存命中，指令缓存会把对应的指令内容返回给 CPU。
  - “valid” 标签：与右边箭头绑定，表示返回的指令是否有效：
    - valid = 1 → 指令已经准备好；
    - valid = 0 → 指令缓存还在等待数据（比如需要去更低级缓存或内存取）。

缓存类型：数据缓存（D-Cache）



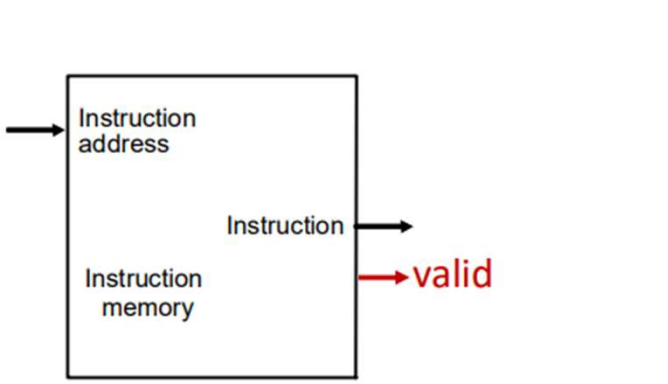
- 数据缓存（Data Cache，D-Cache）
  - 专门用来缓存程序运行时访问的数据的小型高速存储器。
  - 当 CPU 需要读取或写入变量、数组、对象等数据时，会先从 D-Cache 查找。
  - 只存放数据，不存放指令。既有读取（load）操作，又有写入（store）操作。
  - 左边 → Address：CPU 提供要访问的**数据地址**。
  - 左下角 → Write data：如果 CPU 要写数据，就把要写入的内容传给缓存。
  - 上/下方控制箭头 → MemWrite/Read：告诉缓存“我要写/读数据”。
  - 右边 → Read data：如果缓存命中且是读操作，把对应的数据返回给 CPU。
  - “valid” 标签：表示**返回的数据是否准备好**。
  - “ready” 标签：表示缓存是否准备好接收新的读写请求：

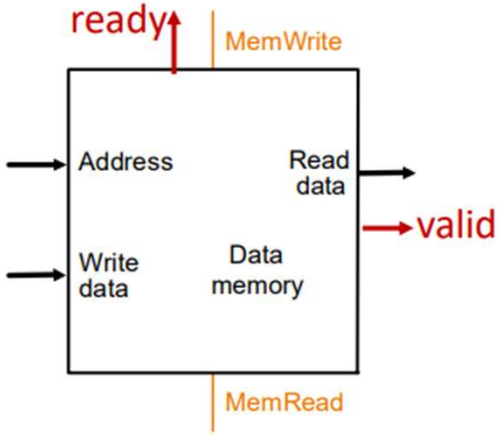
为什么要分开设计 I-Cache 和 D-Cache



- 现代 CPU 通常采用 Harvard 架构（哈佛架构）：
  - I-Cache 和 D-Cache 分开设计，独立访问。
  - 优点：
    - 并行性更高：取指令和取数据可以同时进行。
    - 优化更灵活：指令缓存只读优化，数据缓存读写优化。
    - 减少冲突：避免指令和数据互相抢缓存空间。
- 如果把两者放在一起（统一缓存，Von Neumann 架构），当 CPU 同时需要取指令和读数据时，发生排队，会降低性能。

缓存接口：简单抽象





- **缓存接口：**
  - CPU 给出地址、读/写命令等请求
  - 缓存会在极短且固定的延迟后返回结果或完成更新
- **但有时候会遇到例外，当缓存未命中或者需要额外时间处理时：**
  - 数据最终会变得 valid/ready（数据总能在更低层级找到，缓存处理任务总会完成）
  - 但是在此期间，CPU 的流水线**必须等待**
  - 这种等待现象被称为 **流水线停顿（stall）**

缓存的核心问题

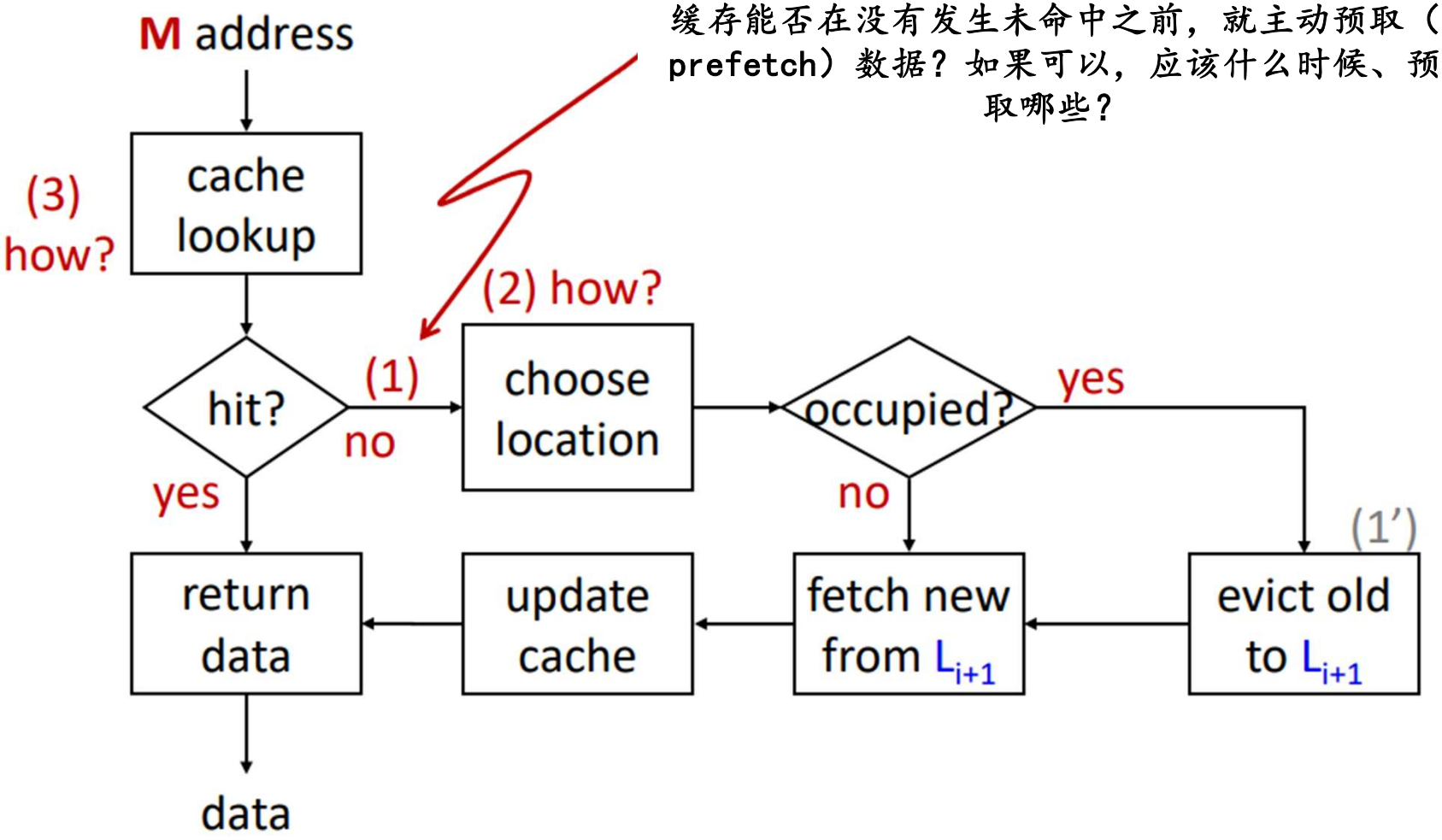
- 缓假设系统有 M = 2^m 字节的主存储器，我们需要在只有 C 字节的小型高速存储器（缓存）中，保存最常用数据的副本，且满足 C << M（缓存容量远小于主存）。

- 缓存设计的三大基本问题（相互关联）
  1. 何时缓存（When）
     - 什么时候应该把某个内存位置的数据副本放到缓存里？
  2. 缓存位置（Where）
     - 在有限的高速缓存空间里，应该把这个副本放在哪一块？
  3. 如何找到（How）
     - 当再次访问该数据时，如何快速在缓存中找到对应的副本？

- 必须既高效又快速，否则缓存的存在就失去了意义。

答案（1）：按需驱动（demand-driven）

缓存能否在没有发生未命中之前，就主动预取（prefetch）数据？如果可以，应该什么时候、预取哪些？



缓存的基本参数

- M = 2^m ：地址空间大小（以字节为单位）
  - 例如：2^{32}（4GB）、2^{64}（16EB）
- G = 2^g ：缓存访问粒度（以字节为单位）
  - 例如：4 字节、8 字节
- C ：缓存容量（以字节为单位）
  - 例如：L1 缓存 16KB，L2 缓存 1MB
- B = 2^b ：缓存块大小（以字节计）
  - 示例：L1 缓存常见 16B，L2 缓存常见 ≥64B
- a ：缓存的相联度（associativity）
  - 示例：1、2、4、，，“C/B”
  - 直接映射、全相联映射和组相联映射

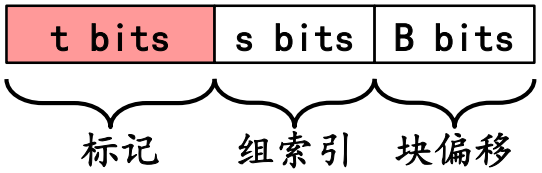
内存块与缓存寻址

- 内存会被逻辑上划分为固定大小的块:内存块（memory block）

- 每个内存块会映射到缓存中的某个位置，这个位置由内存地址中的索引位（index bits）决定
  - 用于定位标记（tag）和数据（data）

内存块与缓存寻址

CPU 想访问某个数据时，它会给出一个内存地址：



- 假设我们有一个8 位的内存地址，地址会被拆成三部分：

地址部分	作用	类比
标记位 Tag	确认缓存里存的数据是不是我们要的	书名
索引位 Index	告诉我们数据在哪个缓存槽位	书架号
块内偏移 Offset	定位块内的具体字节	页码

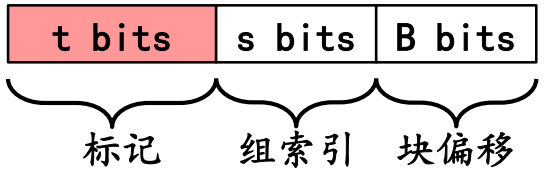
- 地址总长：8 位，标记位 Tag：2位，索引位 Index：3位，块内偏移 Offset：3位
- 举个例子，如果地址是 101 011 00：
  - 前 2 位是 Tag → 确认是不是我们想要的那本书
  - 中间 3 位是 Index → 去找第几个缓存槽
  - 后 3 位是 Offset → 这个块里的第几个字节

内存块与缓存寻址

- 内存会被逻辑上划分为固定大小的块:内存块（memory block）

- 每个内存块会映射到缓存中的某个位置，这个位置由内存地址中的索引位（index bits）决定
  - 用于定位标记（tag）和数据（data）

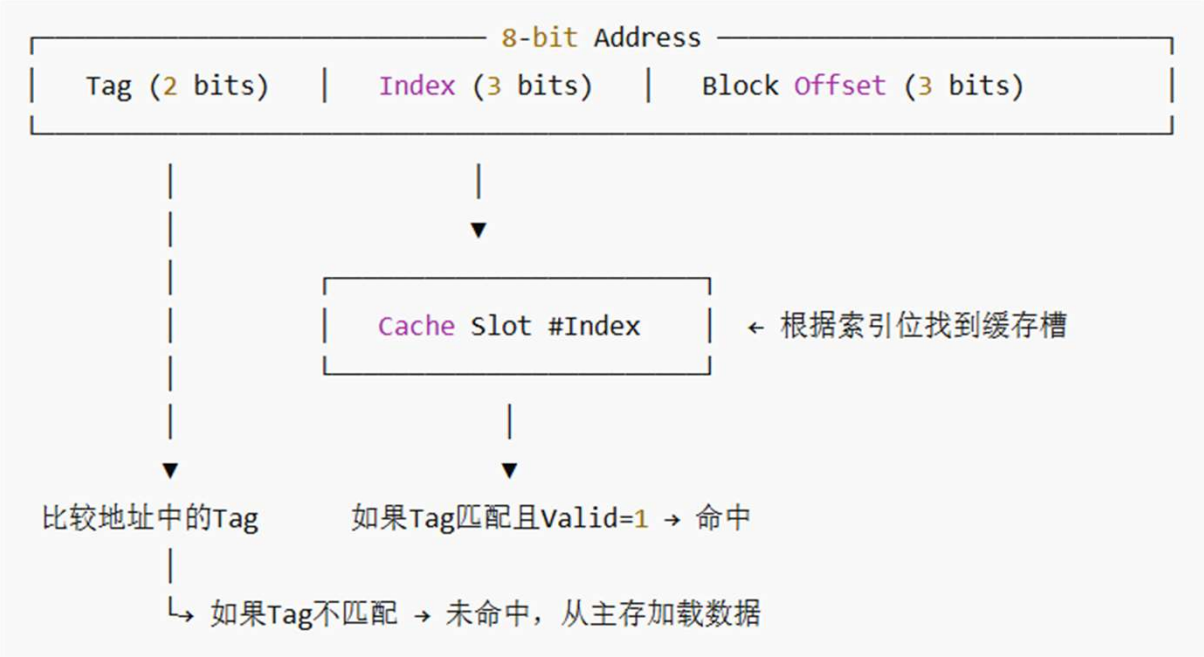
CPU 想访问某个数据时，它会给出一个内存地址:



- 缓存访问流程:
  - 当 CPU 给出一个内存地址，缓存访问分三步走:
    1. 根据索引位定位缓存槽位
       - 就像先找到哪一排书架。
    2. 检查有效位（valid bit）
       - 确认这个槽里是否有最新数据。
    3. 比较标记位
       - 如果缓存里存的 Tag 和地址中的 Tag 相同 → 缓存命中（Cache Hit），直接取数据;
       - 如果不一样 → 缓存未命中（Cache Miss），需要去主存加载数据。

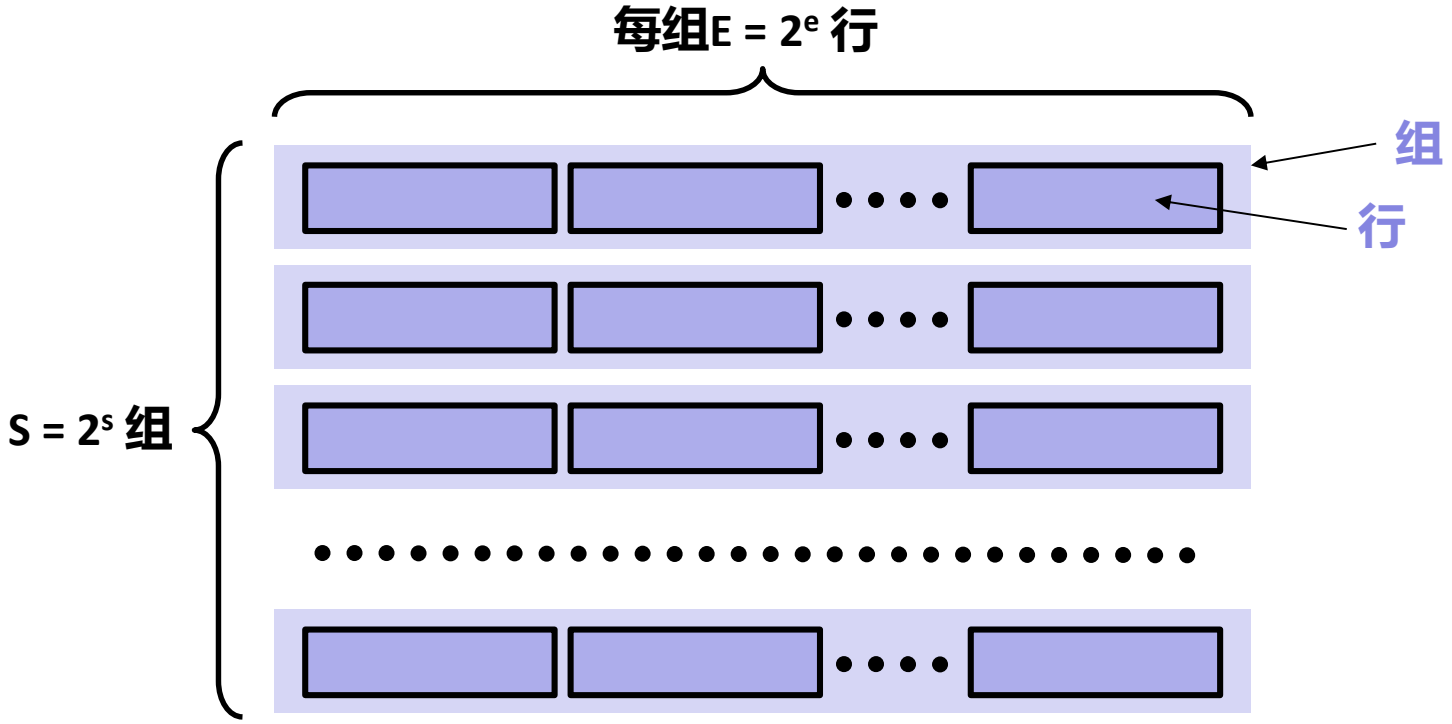
内存块与缓存寻址

- 内存会被逻辑上划分为固定大小的块:内存块（memory block）
- 每个内存块会映射到缓存中的某个位置，这个位置由内存地址中的索引位（index bits）决定
  - 用于定位标记（tag）和数据（data）
- 缓存访问流程:

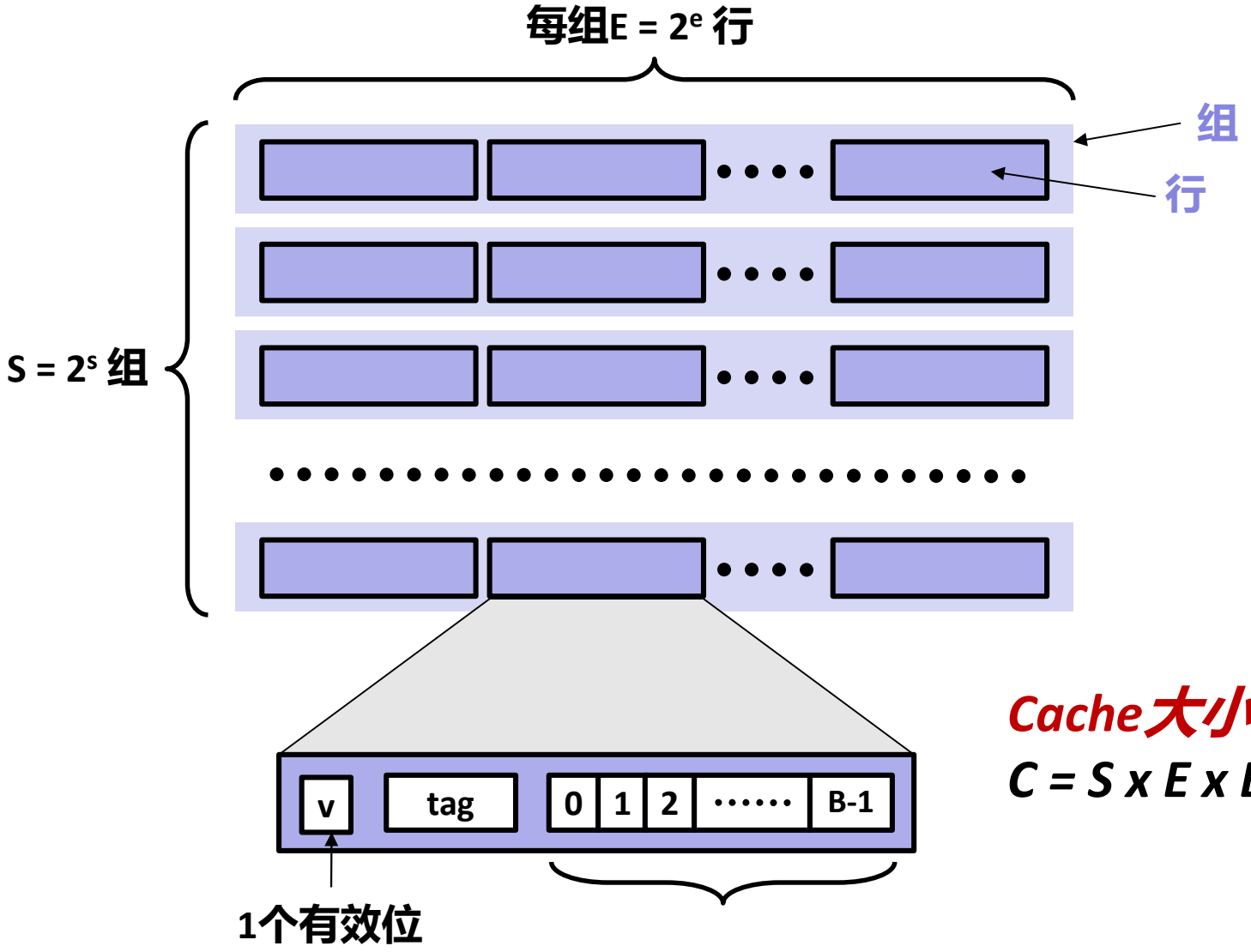


1. 根据索引位定位缓存槽
2. 地址中的 Tag 和 缓存中对应槽位的 Tag 比较，确认是不是我们要的数据
3. 根据块内偏移找到具体字节

一般Cache组织结构(S, E, B)



一般Cache组织结构(S, E, B)



Cache大小:

C = S x E x B 字节

Cache 的三大核心参数：S、E、B

- 1. S：组的数量（Sets）
  - 缓存被分成了 S 个组。
  - 每个组可以看作是一排“书架”。
  - CPU 地址中的**索引位（index bits）**会告诉我们，数据在哪一个组中。
  - 例如，如果 S = 8，那么缓存就有 8 个组，索引位就需要 3 位（因为 $2^3 = 8$）。
- 2. E：每组的行数（Lines per Set）
  - 每个组内部有 E 行（也称 E 个缓存槽位）。
  - E 决定了同一组里能存放多少不同的数据块。
    - 如果 E = 1 → 每组只有一行 → 直接映射缓存（Direct-Mapped Cache）。
    - 如果 E 很大，甚至能容纳所有数据块 → 全相联缓存（Fully Associative Cache）。
    - 如果 1 < E < 全部 → 组相联缓存（Set-Associative Cache）。
- 3. B：每行的数据块大小（Block Size）
  - 每一行缓存存放的数据量是 B 字节。
  - CPU 地址中的块内偏移（block offset）用来找到块内的具体字节。
  - 例如，如果 B = 8，那么每行缓存存 8 个字节，块内偏移就需要 3 位（因为 $2^3 = 8$）。

Cache 的三大核心参数：S、E、B

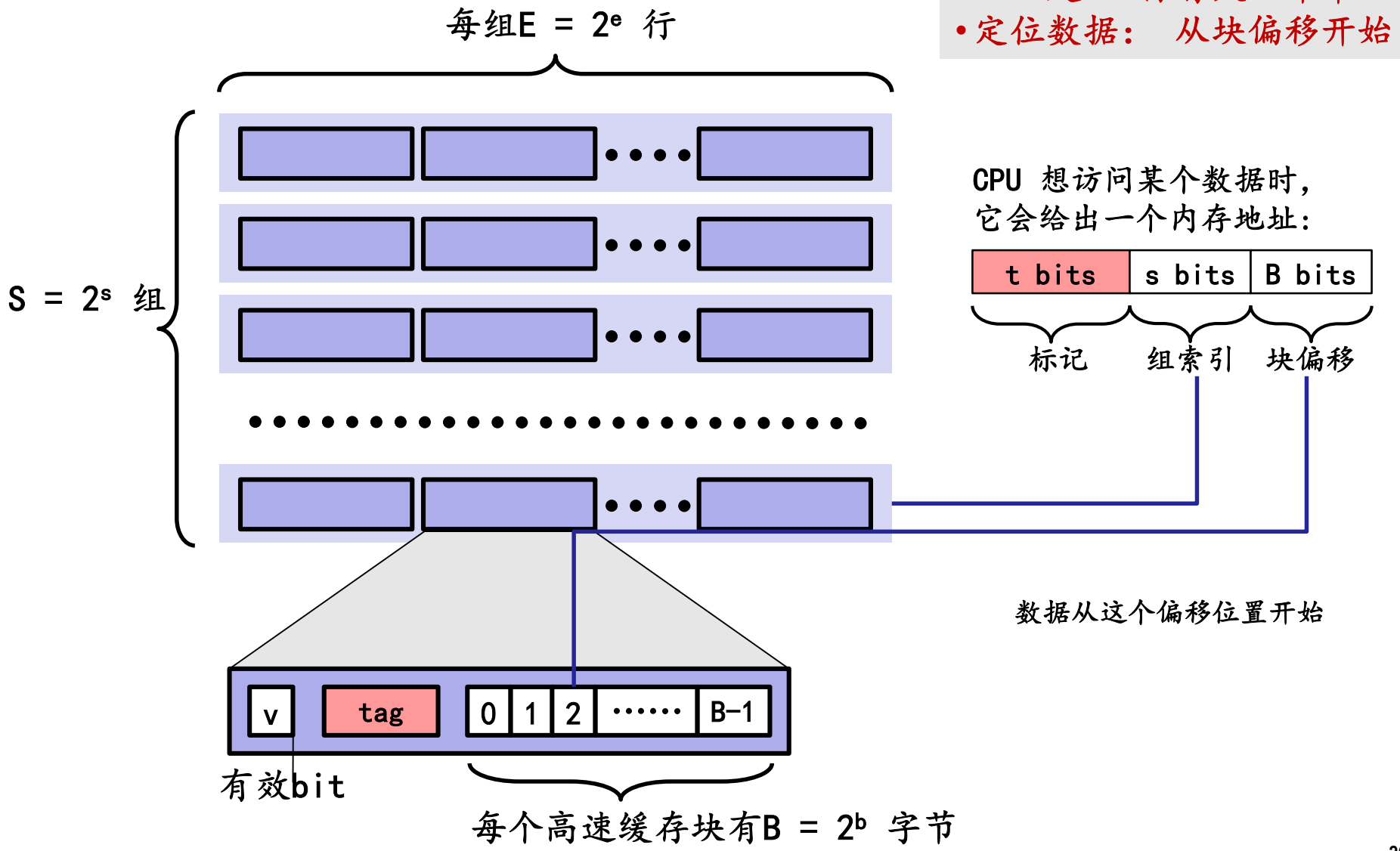
符号	含义	解释
S	组的总数（Number of Sets）	缓存被分成多少组
s	组索引位数（Index Bits）	用于定位缓存组的比特位数
E	每组的行数（Lines per Set）	每组里有多少缓存行（Cache Lines）
e	行索引位数（Line Index Bits）	如果 E>1，用多少比特区分组内行
B	每行数据块大小（Block Size in Bytes）	每行缓存能存多少字节
b	块内偏移位数（Block Offset Bits）	用于确定块内具体字节的比特数

Cache 的三大核心参数：S、E、B

- 每一行缓存包含三个关键部分：
  - 有效位 v
    - 只有当 v=1 时，表示这一行的数据是有效的。
    - 如果 v=0，说明这一行的数据是垃圾，需要去主存取数据。
  - 标记位 tag
    - 标记这行缓存对应的是主存中的哪一个数据块。
    - 当我们找到组后，要通过比较 Tag 判断数据是否匹配。
  - 数据块（B 字节）
    - 真正存储的数据。

读取Cache：可视化

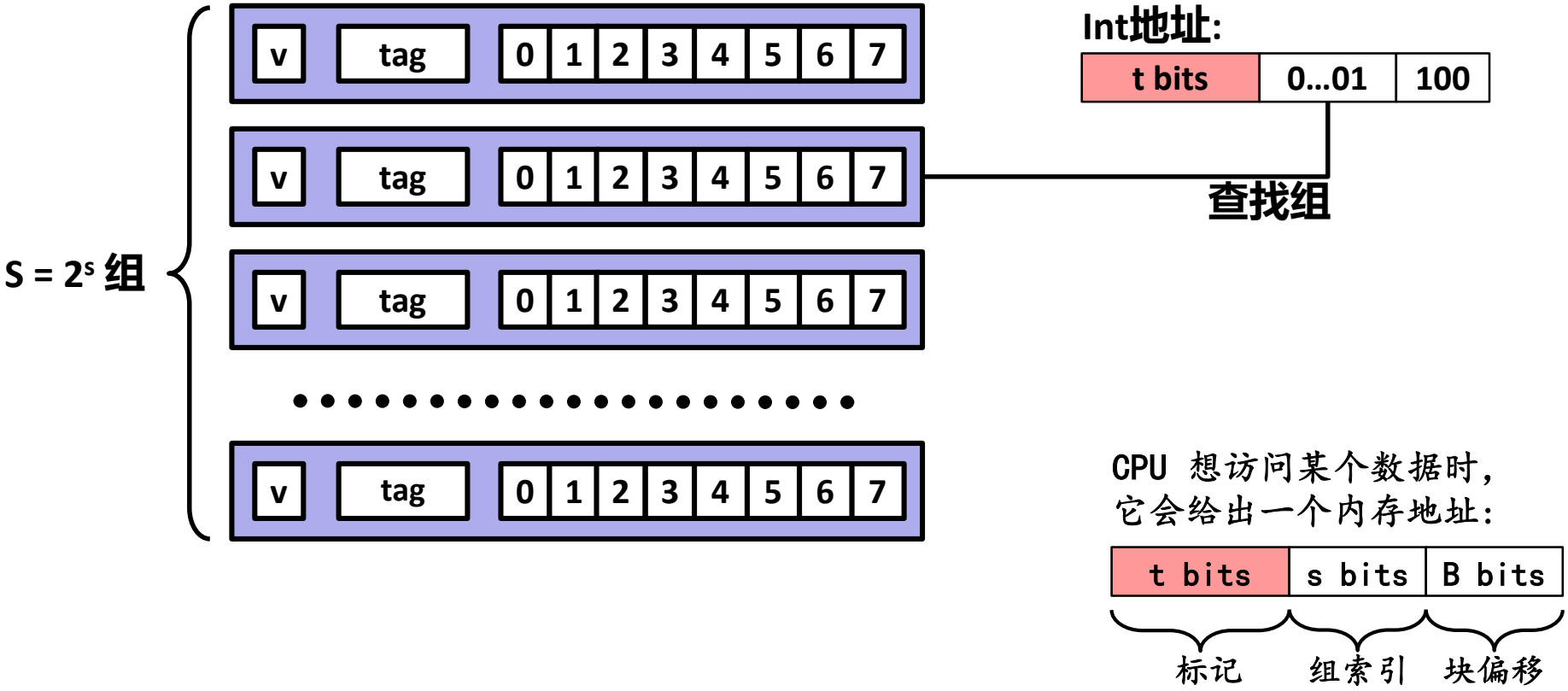
- 定位组
- 定位行
  - 是 + 行有效：命中
- 定位数据：从块偏移开始



例子: 直接映射缓存 (E = 1)

直接映射: 每一组只有一行

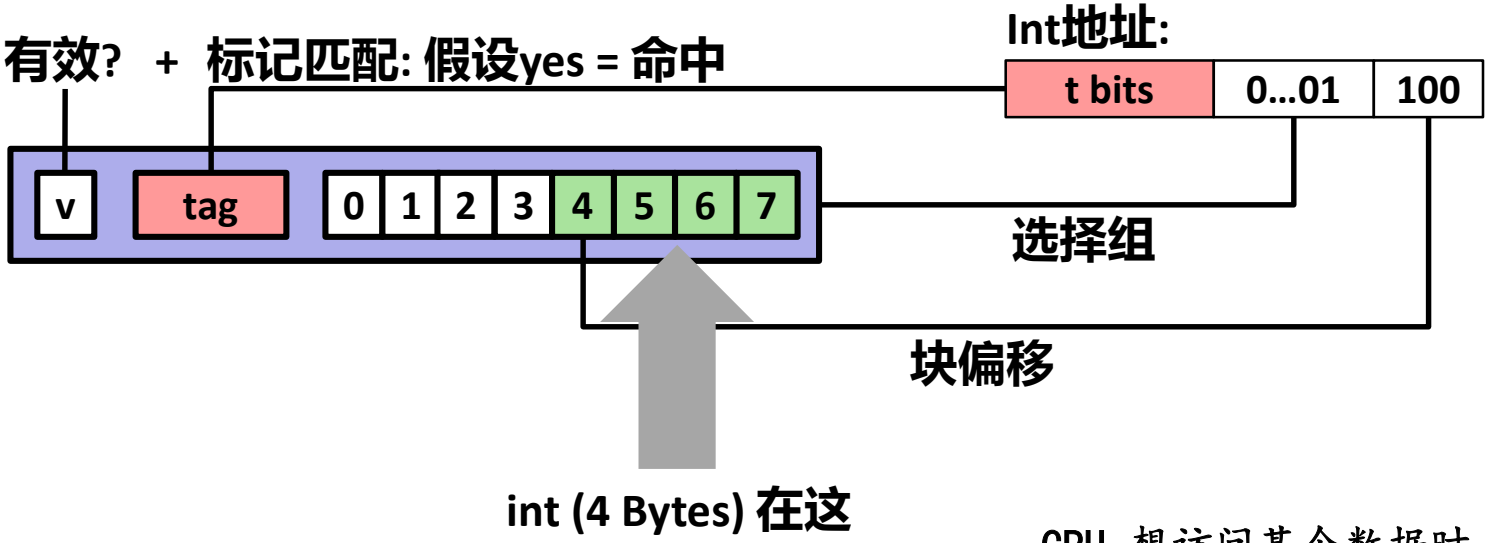
假设: 缓存块大小为8字节:



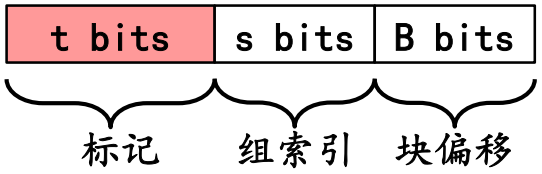
例子: 直接映射高速缓存 (E = 1)

直接映射: 每一组只有一行

假设: 缓存块大小为8字节:



CPU 想访问某个数据时，
它会给出一个内存地址:



如果标记不匹配: 旧的行被驱逐、替换

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1]
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1]
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	0	?	?
Set 1	0	?	?
Set 2	0	?	?
Set 3	0	?	?

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1]
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1]
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	0	?	?
Set 1	0	?	?
Set 2	0	?	?
Set 3	0	?	?

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1]
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1]
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	0	?	?

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1]
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	0	?	?

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	0	?	?

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0]
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	1	0	M[6-7]

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0] 未命中
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	1	0	M[6-7]

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0] 未命中
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0	?	?
Set 2	0	?	?
Set 3	1	0	M[6-7]

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0] 未命中
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0]

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1	0	?	?
Set 2	0	?	?
Set 3	1	0	M[6-7]

直接映射高速缓存模拟

t=1	s=2	b=1
x	xx	x

b:（块内偏移 1 位）
s:（组索引 2 位）
t:（标记 1 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 4 个 set
  - E=1 块/组 ⇒ 每组 1 行（直接映射）

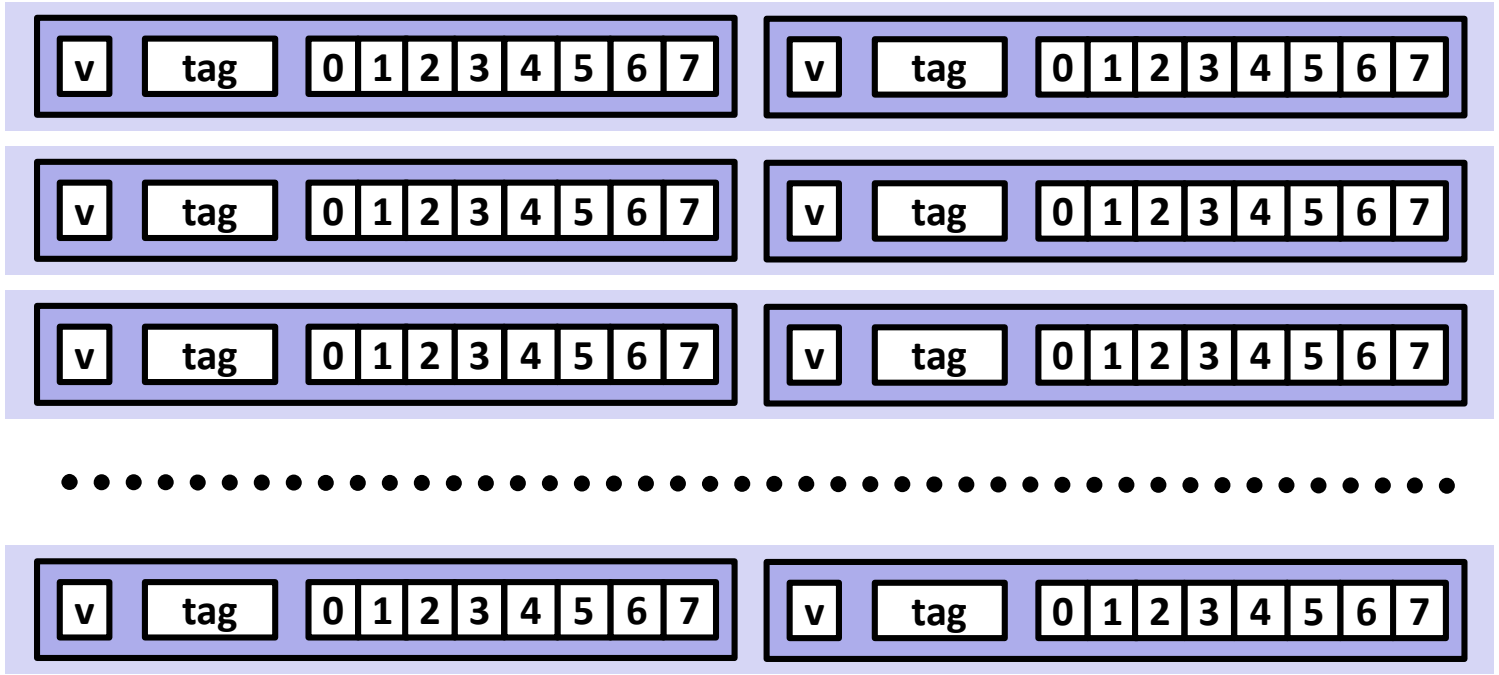
假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中
- 地址1 = 0001_2 → [tag 0 | set 00 | off 1] 命中
- 地址7 = 0111_2 → [tag 0 | set 11 | off 1] 未命中
- 地址8 = 1000_2 → [tag 1 | set 00 | off 0] 未命中
- 地址0 = 0000_2 → [tag 0 | set 00 | off 0] 未命中

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1	0	?	?
Set 2	0	?	?
Set 3	1	0	M[6-7]

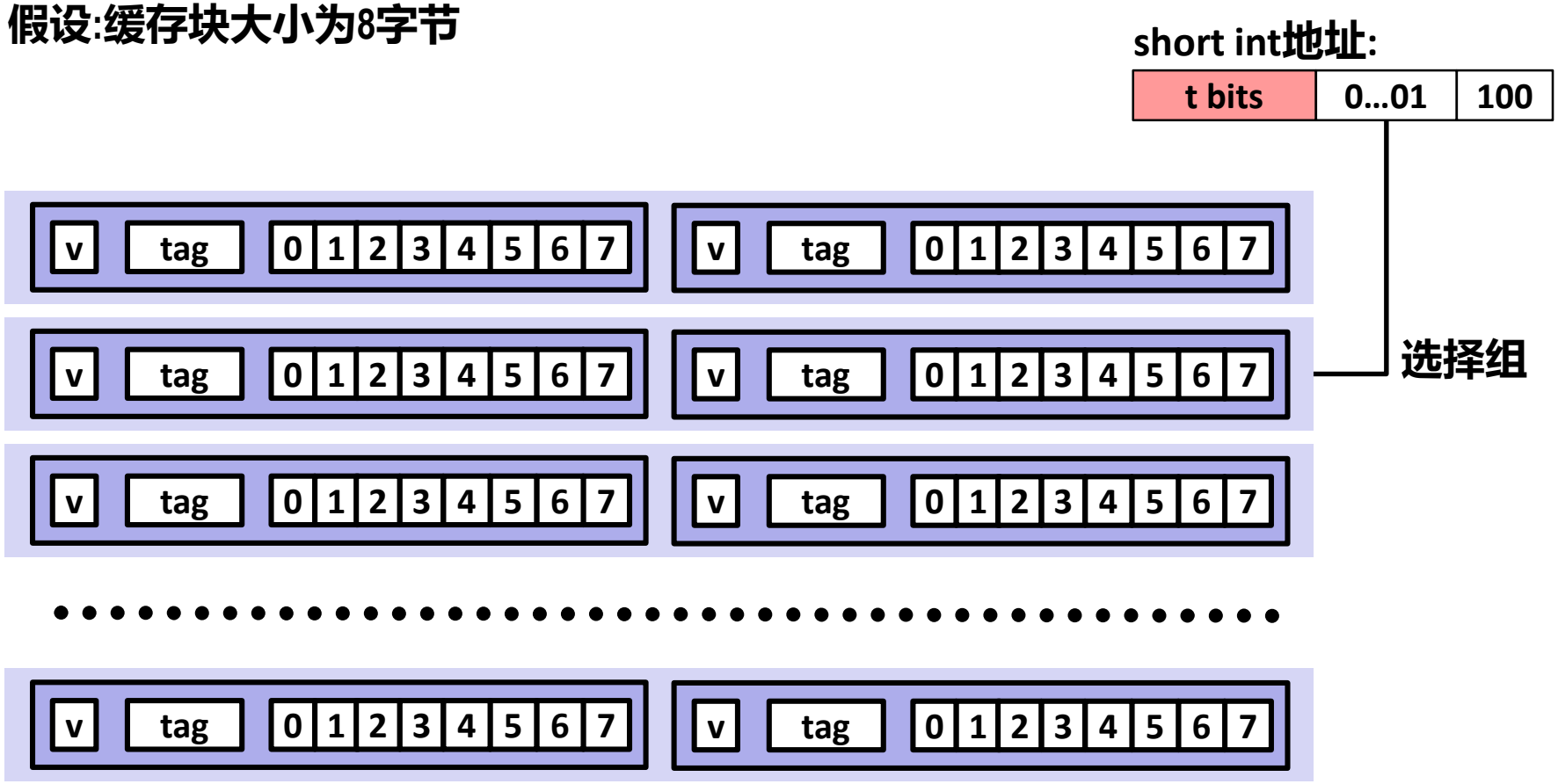
E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



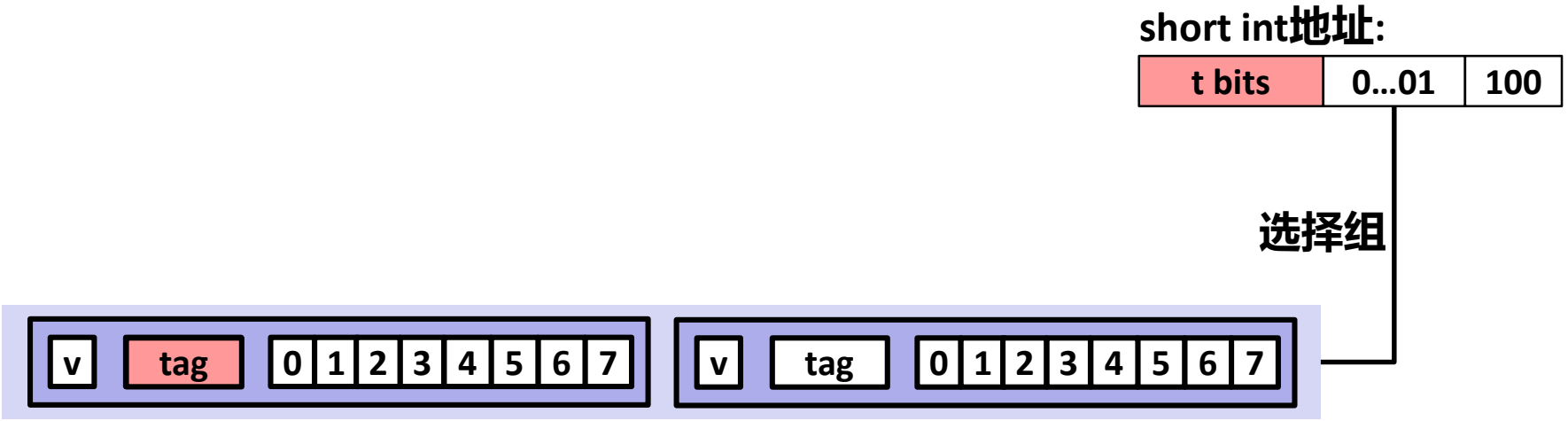
E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



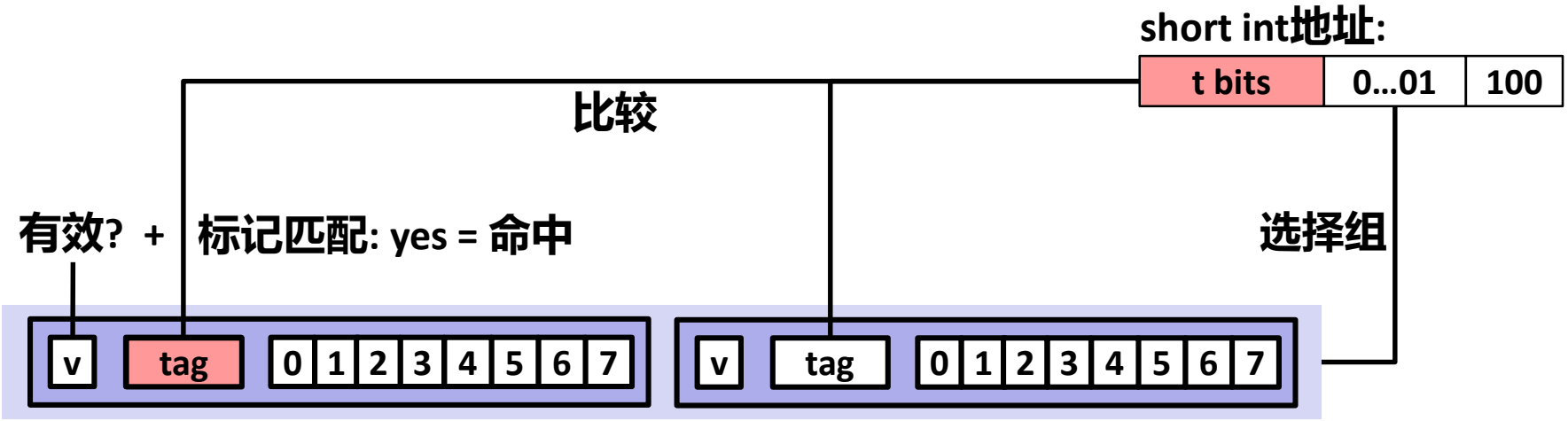
E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



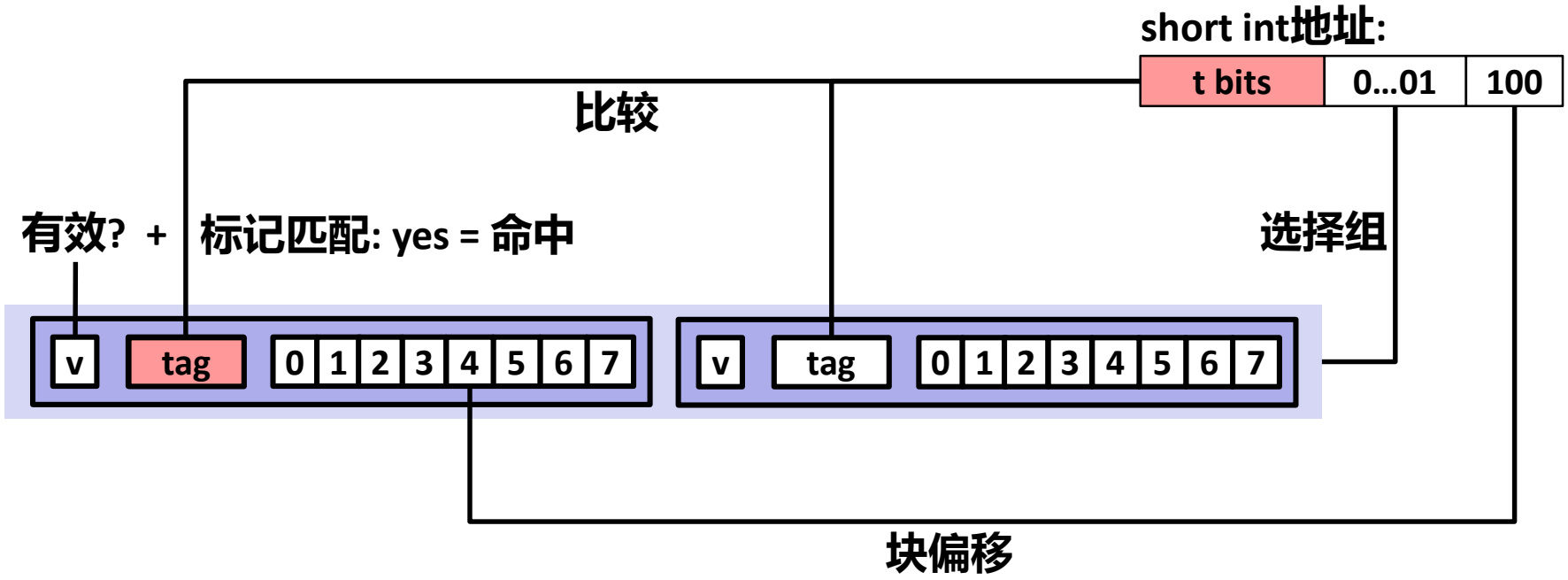
E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



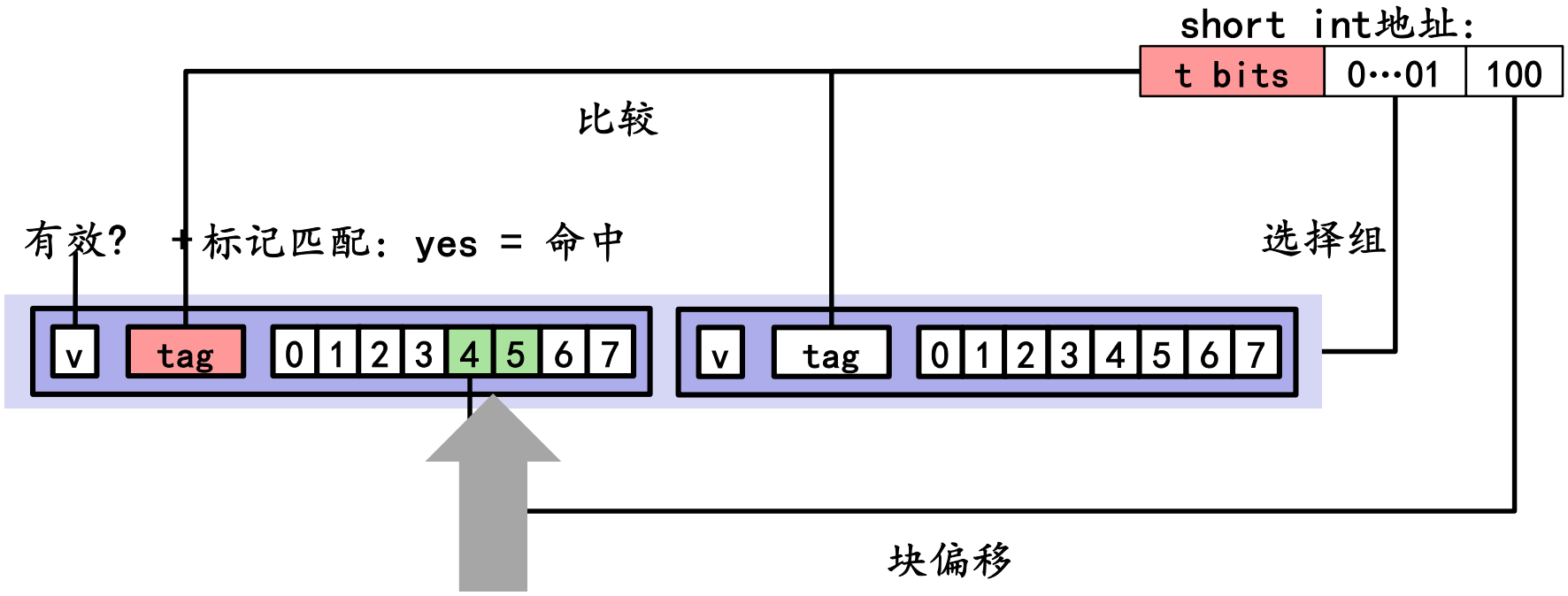
E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



E-路组相联高速缓存 (E = 2)

E = 2: 每组两行
假设:缓存块大小为8字节



如果不匹配:

- 在组中选择1行用于驱逐和替换
- 替换策略: 随机、最近使用(LRU), …

2-路组相联缓存模拟

t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0]
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1]
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1]
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0]
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0]

	v	Tag	Block
Set 0	0	?	?
	0	?	?
Set 1	0	?	?
	0	?	?

2-路组相联缓存模拟

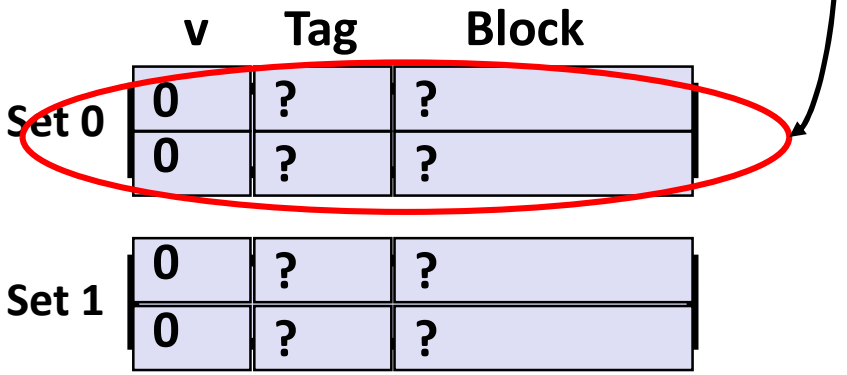
t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)， 2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1]
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1]
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0]
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0]



2-路组相联缓存模拟

t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中

	v	Tag	Block
Set 0	1	00	M[0-1]
	0	?	?
Set 1	0	?	?
	0	?	?

2-路组相联缓存模拟

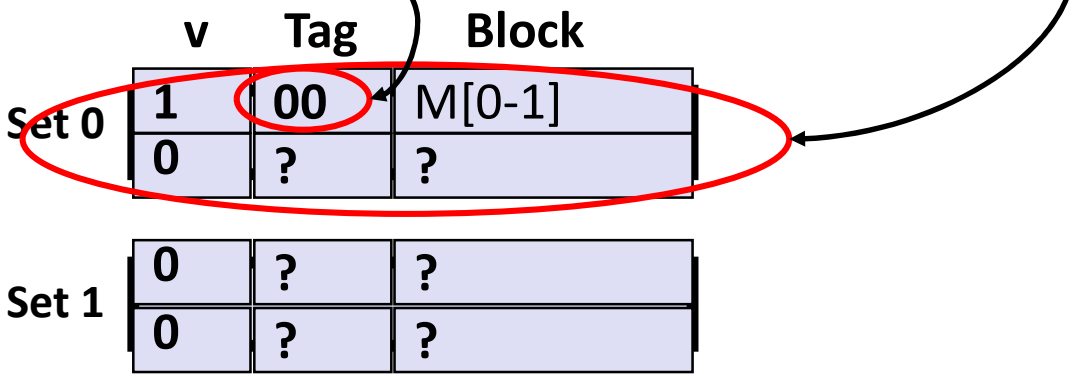
t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中



2-路组相联缓存模拟

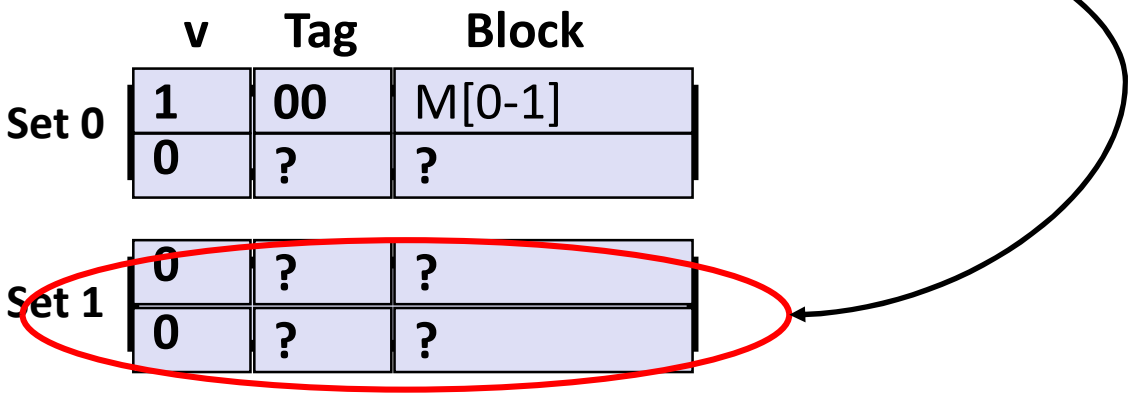
t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中



2-路组相联缓存模拟

t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中

	v	Tag	Block
Set 0	1	00	M[0-1]
	0	?	?
Set 1	1	01	M[6-7]
	0	?	?

2-路组相联缓存模拟

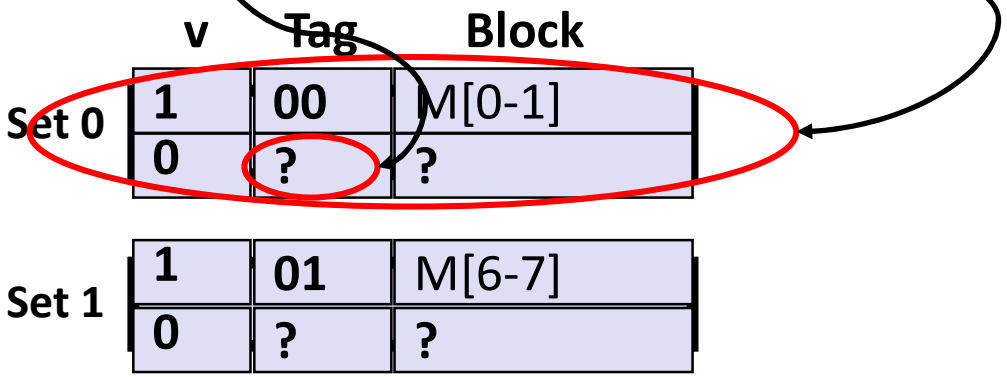
t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中



2-路组相联缓存模拟

t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中

	v	Tag	Block
Set 0	1	00	M[0-1]
	0	10	M[8-9]
Set 1	1	01	M[6-7]
	0	?	?

2-路组相联缓存模拟

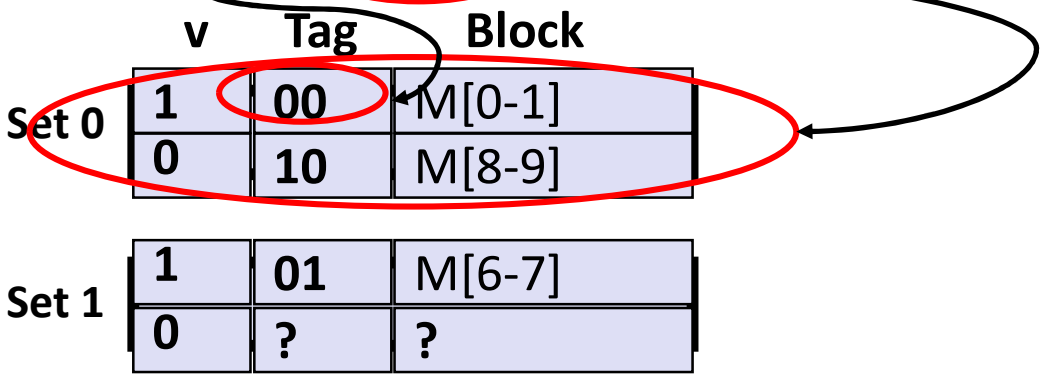
t=2	s=1	b=1
xx	xx	x

b:（块内偏移 1 位）
s:（组索引 1 位）
t:（标记 2 位）

- 假设主存容量M=16 字节 (4-位地址,地址从 0-15)，2 字节/块
- 假设缓存容量为8 字节
  - B=2 字节/块 ⇒ 每行存 2 字节(一次从内存取2字节)
  - S=4 组 ⇒ 有 2 个 set
  - E=2 块/组 ⇒ 每组 2 行

假设CPU开始尝试顺序读取以下内存地址数据：

- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 未命中
- 地址1 = 0001_2 → [tag 00 | set 0 | off 1] 命中
- 地址7 = 0111_2 → [tag 01 | set 1 | off 1] 未命中
- 地址8 = 1000_2 → [tag 10 | set 0 | off 0] 未命中
- 地址0 = 0000_2 → [tag 00 | set 0 | off 0] 命中



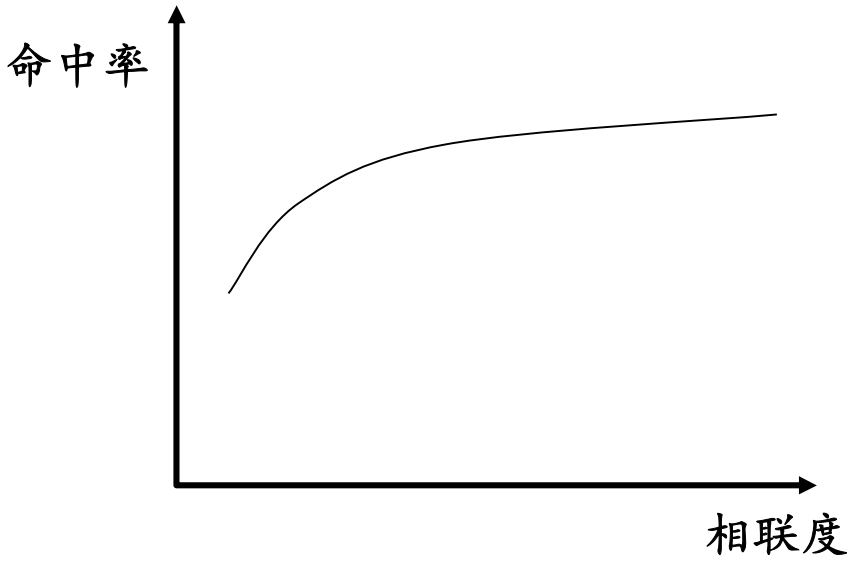
Cache的种类

- **直接映射缓存（Direct-Mapped Cache）：E = 1**
  - 主存中的一个块只能映射到 Cache 中的某一个**特定块**。
  - 类比：就像宿舍床位固定分配，每个人只能睡自己的床位。如果别人偶尔需要住一晚，就必须赶走你 → 容易打架。
- **全相联映射缓存（Fully Associative Cache）：S = 1**
  - 主存中的任何一块数据都可以映射到 Cache 中的**任意一块**。
  - 类比：就像宿舍没有床位固定分配，大家可以随便睡哪张床。虽然灵活，但每次都得走遍整个宿舍找空床 → 成本高。
- **组相联映射缓存（Set-Associative Cache）：E > 1 且 S > 1**
  - 是前两种方法的折中方案。
  - **兼顾两者优点，尽量避免两者缺点**，因此在现代计算机中最常用。
  - 类比：宿舍被分成多个寝室（组），每个寝室里有多张床（行）。学号告诉你住哪个寝室，但寝室里的床可以灵活选择 → 冲突少，灵活高效。

相联度（Associativity）及其权衡

- **相联度的定义：**
  - 指有多少个主存块可以映射到同一个缓存索引（或组）。
- **更高的相联度：**
  - 优点：缓存命中率更高（Higher hit rate）
  - 缺点：
    - 缓存访问时间变慢（包括命中延迟和数据访问延迟）
    - 硬件成本更高（需要更多比较器 Comparator）
- **高相联度的边际收益递减：**
  - 当相联度不断提高时，

命中率提升越来越有限。



缓存的驱逐 / 替换策略

- 缓存未命中（cache miss）时：应该替换同一组中的哪一块？
  - 优先替换无效块（invalid block）。
  - 如果所有块都有效，则遵循替换策略（replacement policy）。
- 常见替换策略：
  - 随机替换（Random）
  - 先进先出（FIFO, First In First Out）
  - 最近最少使用（LRU, Least Recently Used）
  - 非最近使用（NRU, Not Most Recently Used）
  - 最少使用次数（LFU, Least Frequently Used）
  - 混合替换策略（Hybrid replacement policies）
  - 最优替换策略（Optimal replacement policy） —— 理论上最优，但实现困难

补充：缓存替换与页面替换的关系

- 物理内存（DRAM）本质上是磁盘的缓存
  - 通常由操作系统通过虚拟内存子系统进行管理
- 页面替换（Page Replacement）与缓存替换类似
- 页表（Page Table）相当于物理内存的数据“标记存储区”（tag store）
- 两者的主要区别：
  - 访问速度要求不同：缓存访问速度要求比内存更快
  - 块数量不同：缓存中的块数量远少于物理内存
  - 可容忍的替换延迟不同：
    - 磁盘访问慢 → 页面替换可以容忍较长延迟
    - 缓存访问快 → 缓存替换必须非常迅速
  - 硬件与软件的角色不同：
    - 缓存管理主要依赖硬件
    - 页面替换主要由操作系统软件管理