4.3 高速缓冲存储器

- 一、概述
  - 1、为什么用Cache
  - 2、Cache的工作原理
    - 主存和缓存的编址
    - 命中与未命中
    - Cache的命中率
    - Cache-主存系统的效率
  - 3、Cache的基本结构
  - 4、Cache的读写操作
  - 5、Cache的改进
- 二、Cache-主存的地址映射
- 三、替换算法

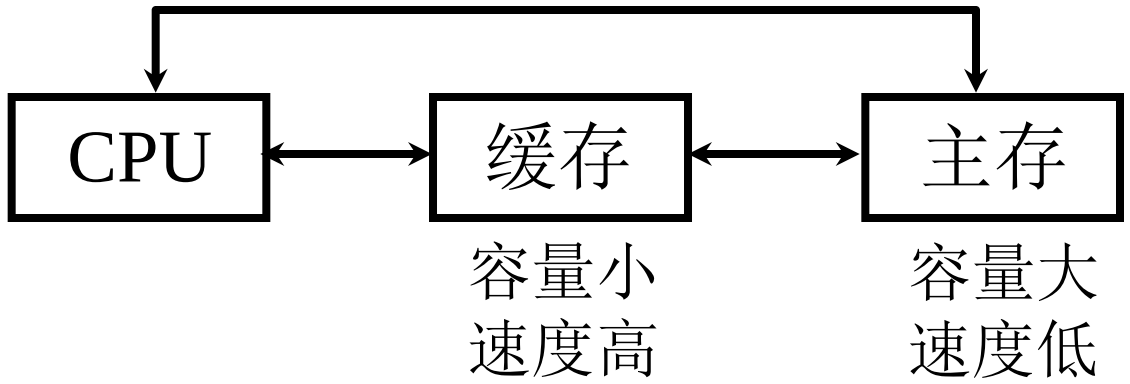
4.3 高速缓冲存储器

一、概述

1. 问题的提出

避免 CPU “空等” 现象

CPU 和主存（DRAM）的速度差异



程序访问的局部性原理

时间局部性：当前使用的指令和数据，在不久的将来 也会用到

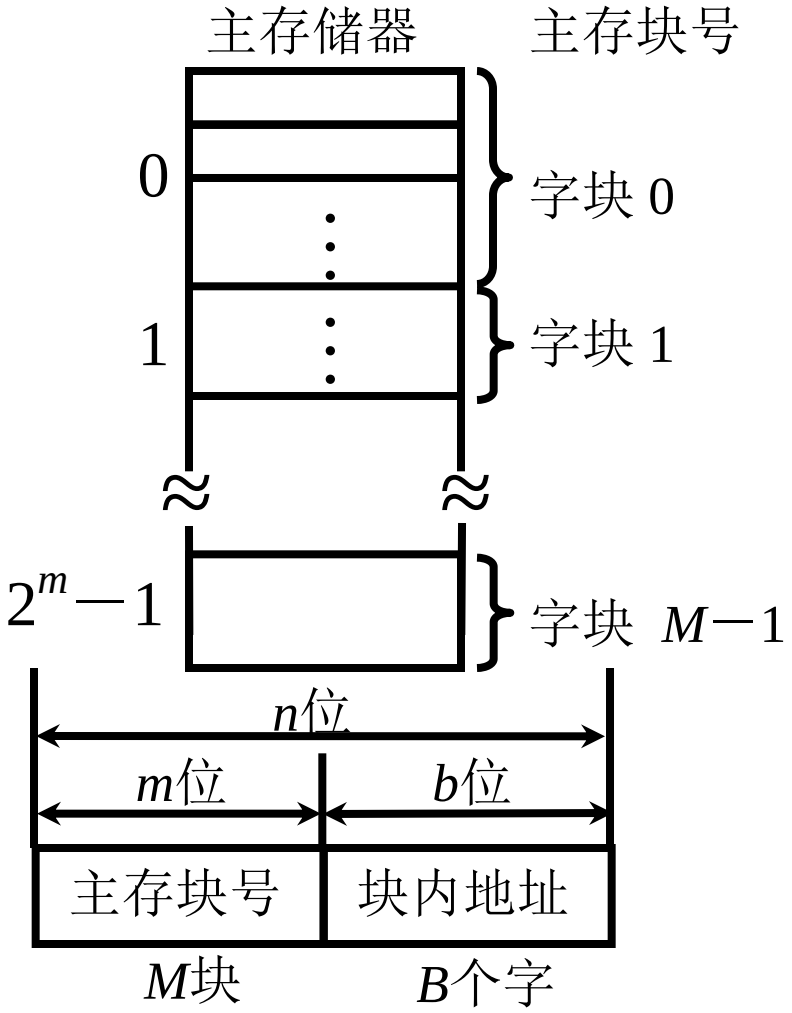
空间局部性：当前使用的指令和数据，相邻的 指令和数据可能会用到

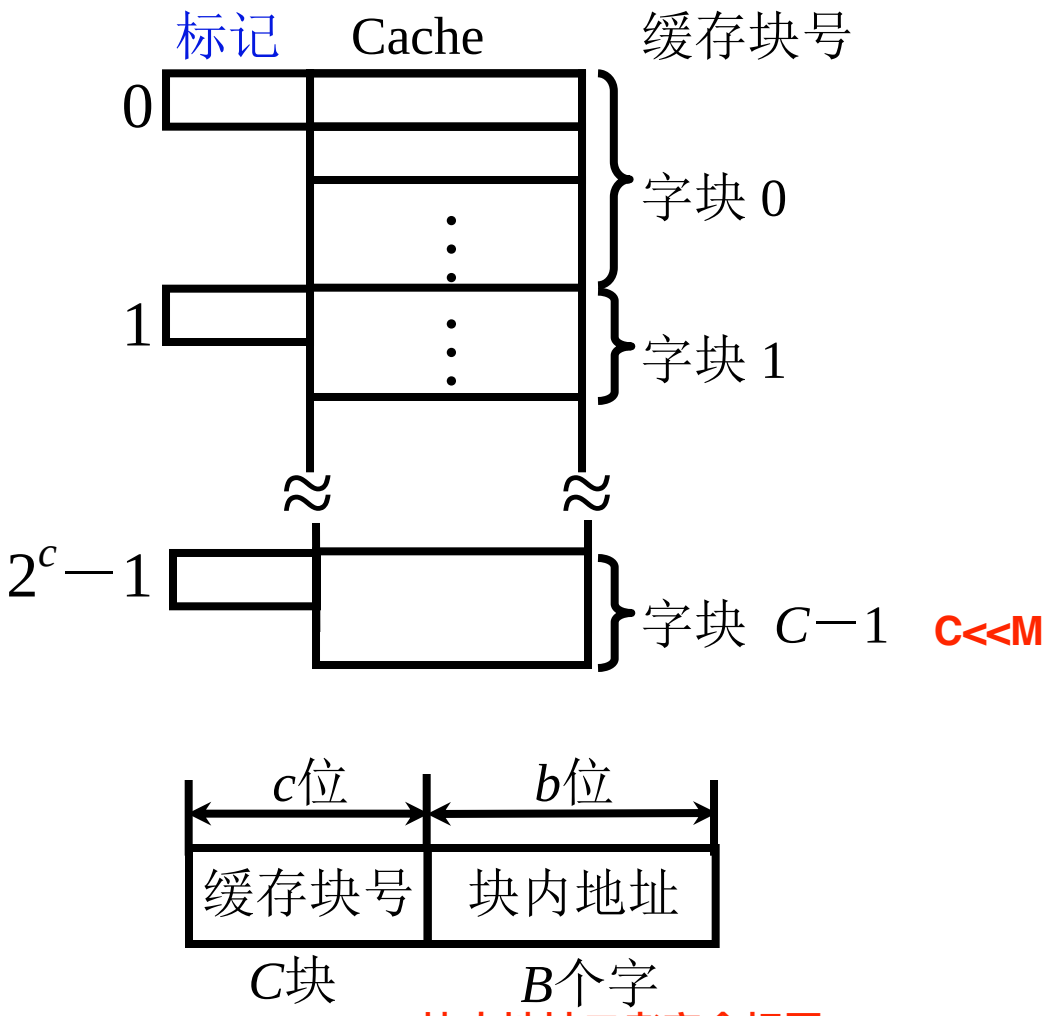
2. Cache 的工作原理

(1) 主存和缓存的编址

4.3

标记主存块和Cache块的对应关系　　将主存块号写入Cache中的标记





C<<M

块内地址二者完全相同

主存和缓存按块存储　　块的大小相同　　B 为块长

(2) 命中与未命中

4.3

缓存共有 C 块

主存共有 M 块　$M >> C$

命中　主存块 调入 缓存

主存块与缓存块 建立 了对应关系　某一个主存块装入了对应的缓存块

用 标记记录 与某缓存块建立了对应关系的 主存块号

未命中　主存块 未调入 缓存

主存块与缓存块 未建立 对应关系

(3) Cache 的命中率

4.3

CPU 欲访问的信息在 Cache 中的 比率

命中率 与 Cache 的 容量 与 块长 有关

容量越大，命中率可能越高　　块太小，没有充分利用程序局部性原理

一般每块可取 4 ~ 8 个字

CPU和主存完整完成一次数据调度的时间（指令发出到数据呈现）

块长取一个存取周期内从主存调出的信息长度

CRAY_1　16体交叉　块长取 16 个存储字

IBM 370/168　4体交叉　块长取 4 个存储字

（64位×4 = 256位）

(4) Cache –主存系统的效率

4.3

效率 e 与 命中率 有关

$$e = \frac{\text{访问 Cache 的时间}}{\text{平均访问时间}} \times 100\%$$

有些从Cache中取（命中），有时不命中

设 Cache 命中率 为 h，访问 Cache 的时间为 t_c，
访问 主存 的时间为 t_m

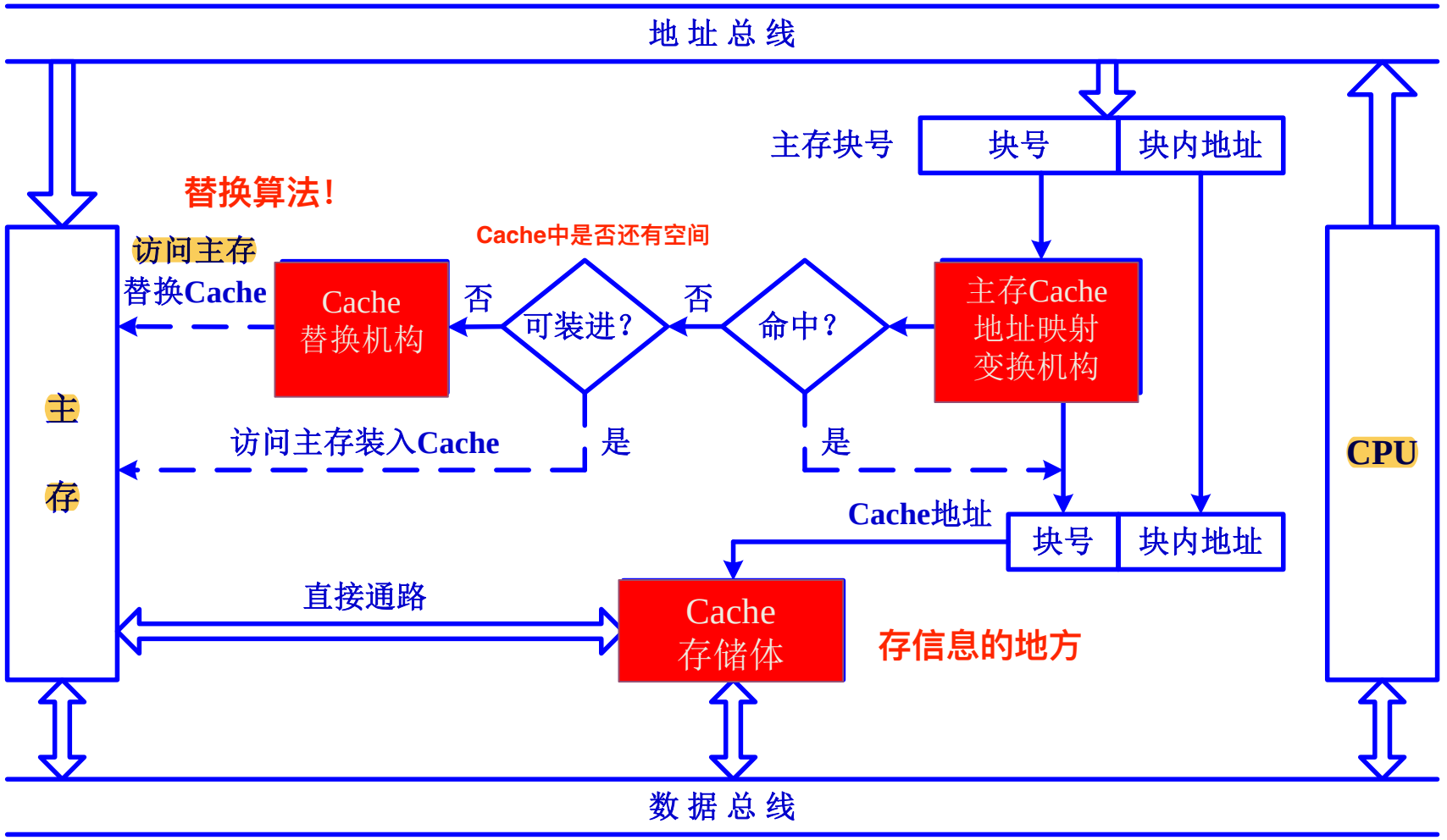
则 $$e = \frac{t_c}{h \times t_c + (1-h) \times t_m} \times 100\%$$

最小值, h=0, tc/tm;
最大值, h=1, 1.

分母说明了CPU对Cache和主存是并行访问的
不是先访问Cache后看命中与否后再访问主存

3. Cache 的基本结构

4.3

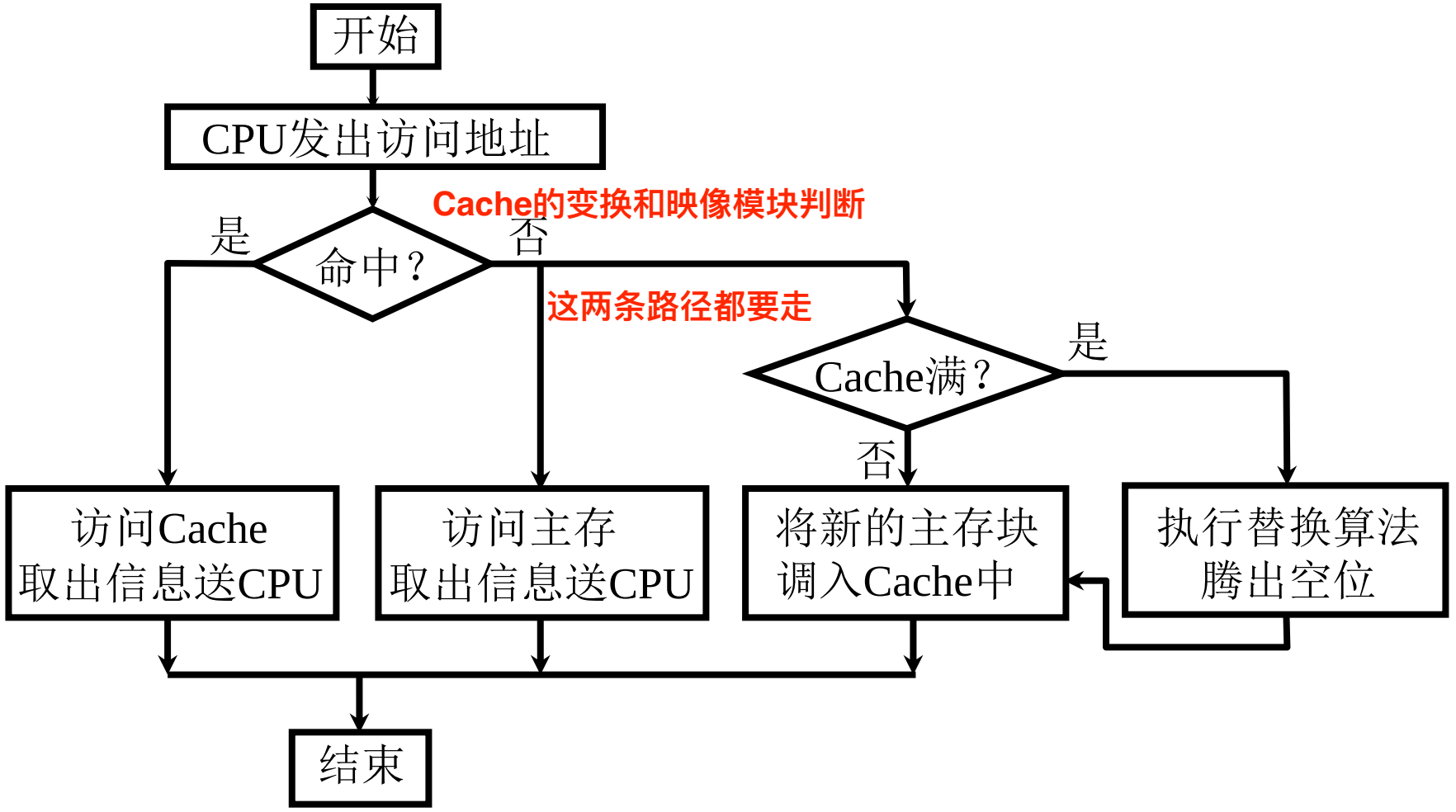


变换：主存块号转换为Cache块号；
主存地址转换为Cache地址

4. Cache 的 读写 操作

4.3

读 读操作不对Cache块中的信息进行修改，Cache信息和主存内信息一致



4. Cache 的 读写 操作

4.3

写　Cache 和主存的一致性

- 写直达法（Write – through）（写通过）

  写操作时数据既写入Cache又写入主存　时刻保存Cache与主存内容一致

  写操作时间就是访问主存的时间，Cache块退出时，

  不需要对主存执行写操作，更新策略比较容易实现　对内存同一个存储单元多次写入，浪费资源

- 写回法（Write – back）

  写操作时只把数据写入 Cache 而不写入主存

  当 Cache 数据被替换出去时才写回主存　无法保证Cache和主存内容实时一致

  写操作时间就是访问 Cache 的时间，

  Cache块退出时，被替换的块需写回主存，增加了　并行计算机，Cache副本不一致

  Cache 的复杂性

5. Cache 的改进

4.3

(1) 增加 Cache 的级数

片载（片内）Cache　CPU内部就直接做Cache　多核CPU，每个核都有自己的Cache

片外 Cache　在CPU外面，主板上

(2) 统一缓存和分立缓存

指令 Cache　数据 Cache　MIPS就是分开的（流水线机器）

与指令执行的控制方式有关　是否流水

Pentium	8K 指令 Cache	8K 数据 Cache
PowerPC620	32K 指令 Cache	32K 数据 Cache

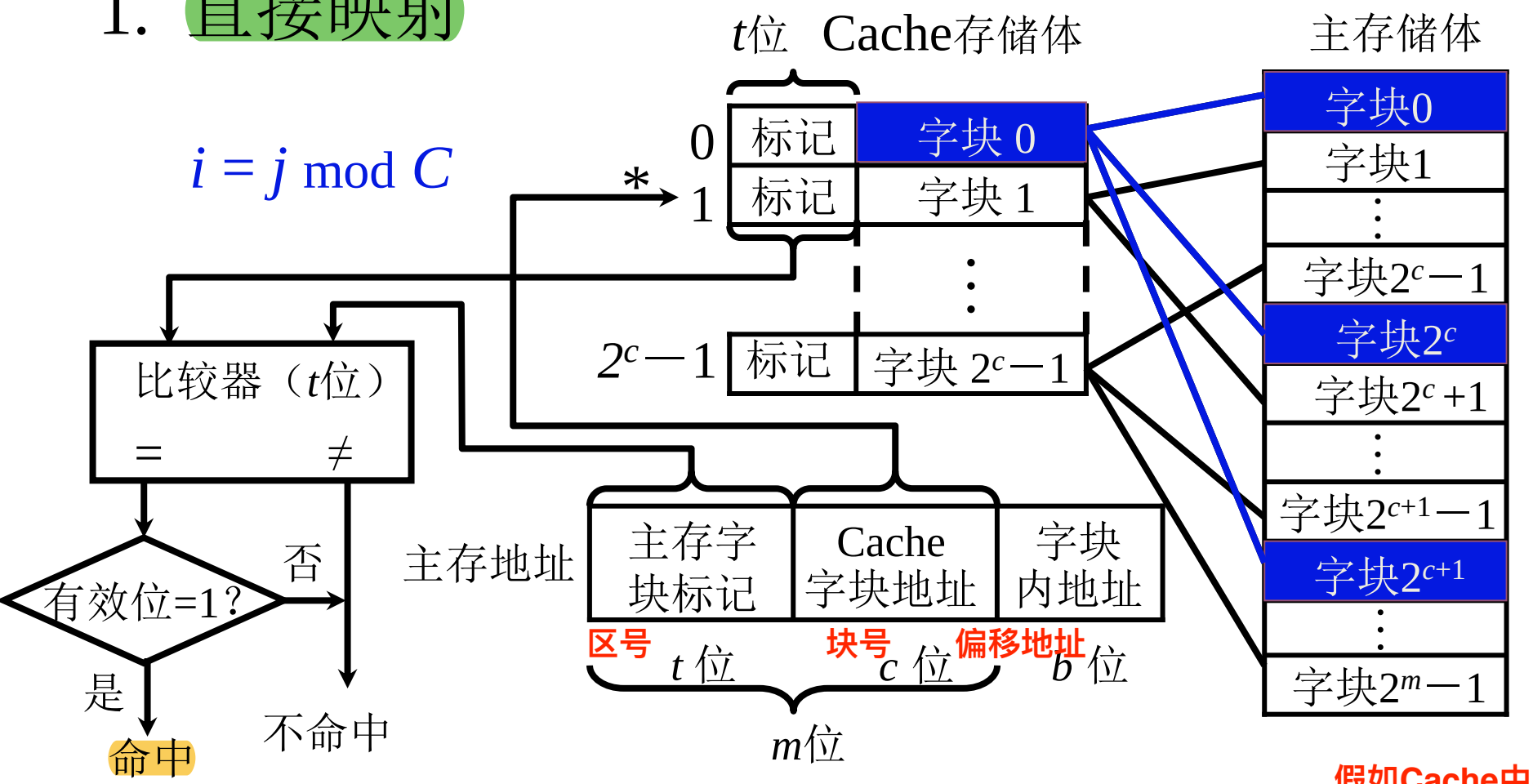
4.3 高速缓冲存储器

- 一、概述
- 二、Cache-主存的地址映射
  - 1、直接映射
  - 2、全相联映射
  - 3、组相联映射
- 三、替换算法

二、Cache – 主存的地址映射

4.3

1. 直接映射



Cache利用率很低，
Cache调入时冲突率很大

每个缓存块 i 可以和 若干 个 主存块 对应

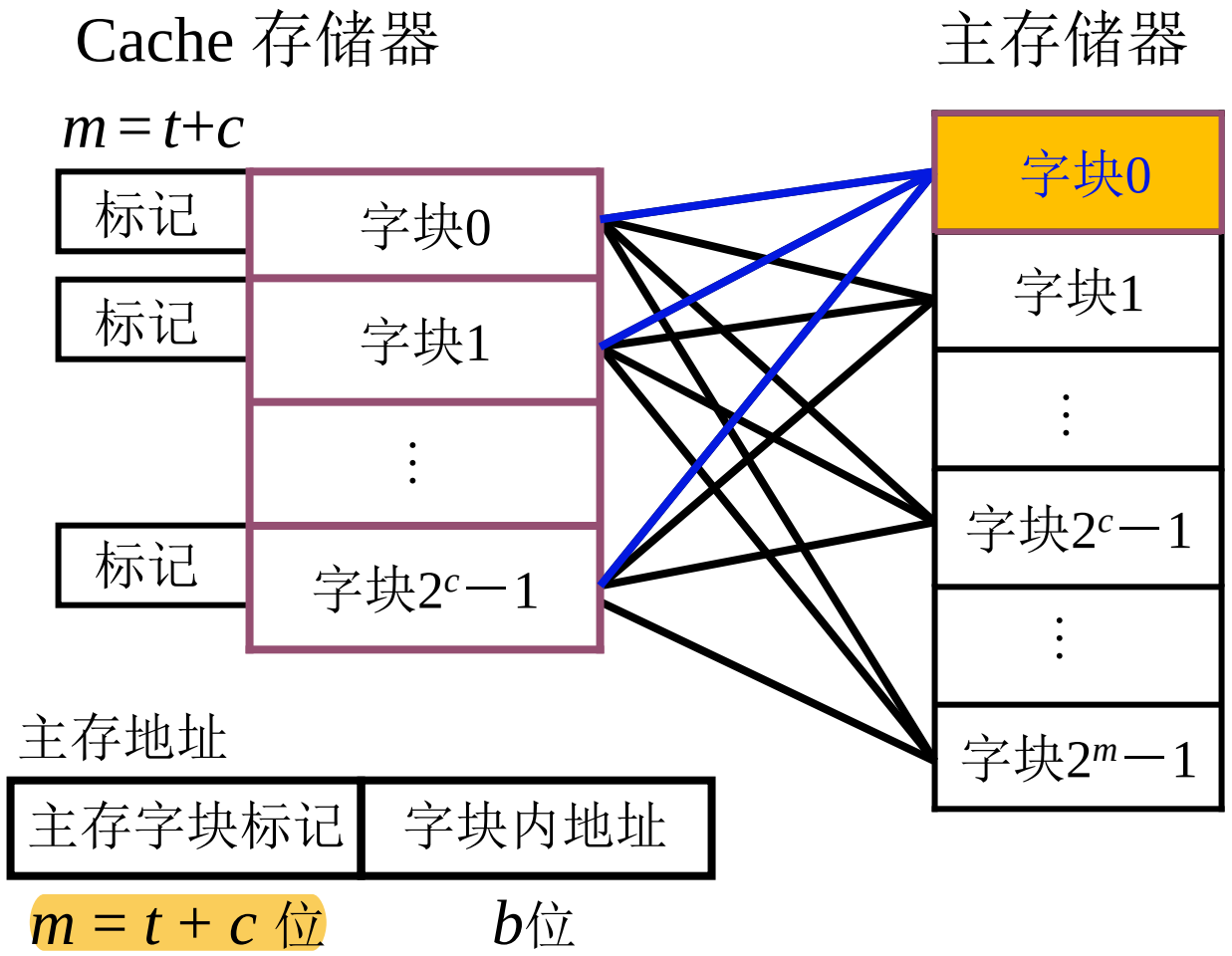
每个主存块 j 只能和 一 个 缓存块 对应

假如Cache中字块0已经被主存的字块0占用，
其他的Cache块均为空，
如果想调用主存的2^c字块，
必须存储在Cache字块0，其他的都不能用
浪费

2. 全相联映射

主存中的任何一个块可以
放入Cache中的任何一个位置

4.3



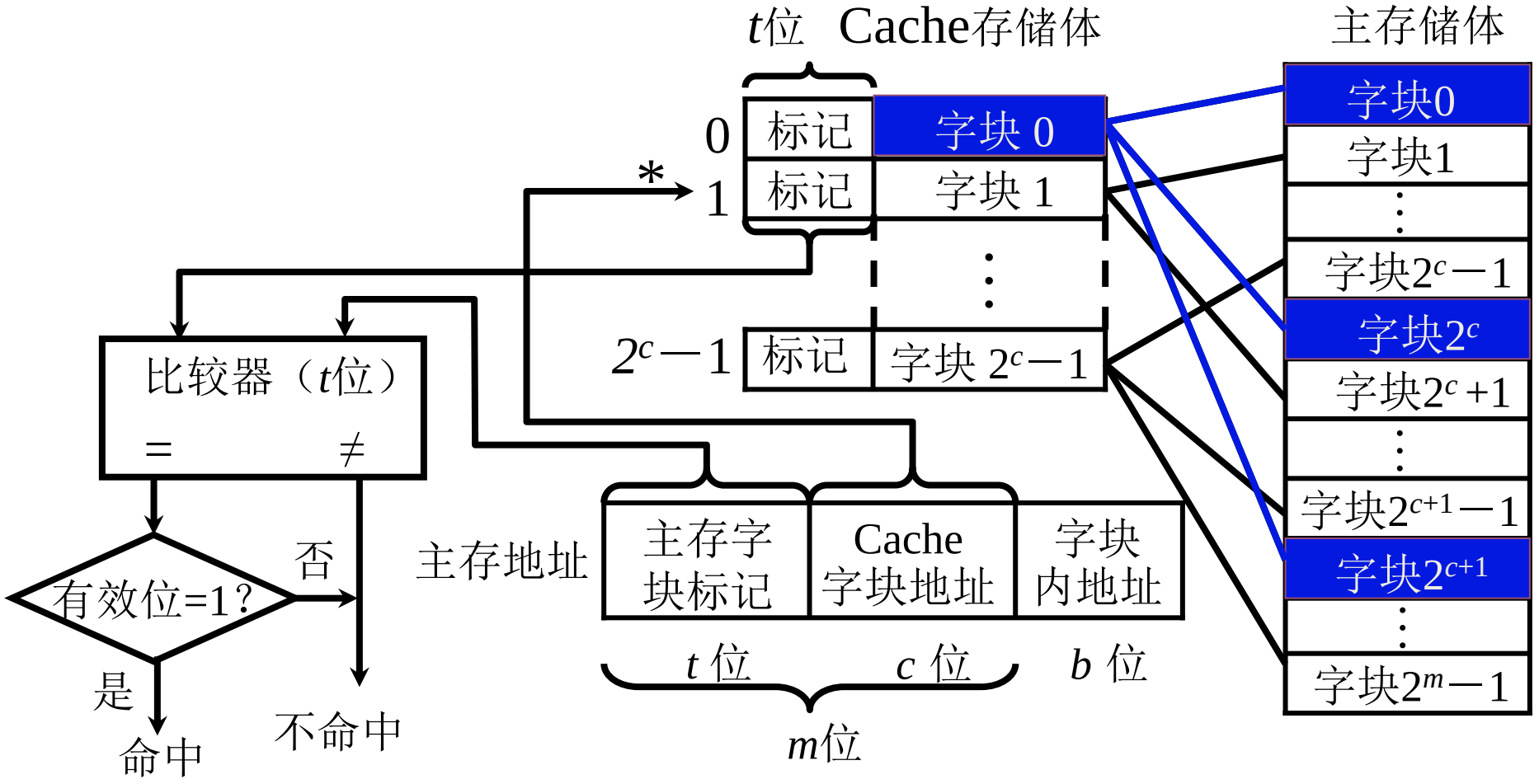
看命中的时候：
需要和Cache中的所有标记比较
速度极慢；参加比较的位数也比较长

主存 中的 任一块 可以映射到 缓存 中的 任一块

二、Cache – 主存的地址映射

4.3

刚刚讲过的直接相联映射



3. 组相联映射

直接相联映射和全相联映射的折中

4.3

路数

组　Cache　共 Q 组，每组内两块（$r = 1$）

组				
0	标记	字块 0	标记	字块 1
1	标记	字块 2	标记	字块 3
	⋮	⋮	⋮	⋮
$2^{c-r}-1$	标记	字块 2^c-2	标记	字块 2^c-1

主存储器

字块0
字块1
⋮
字块$2^{c-r}-1$
字块2^{c-r}
字块$2^{c-r}+1$
⋮
字块2^{c-r+1}
⋮
字块2^m-1

主存中 对应的 第0块，
可以放到Cache中第0组的任何一个位置

靠近Cpu：采用直接相联（高速）；
路数比较少的组相联
距离远：全相联

主存地址

主存字块标记	组地址	字块内地址
$s = t + r$ 位	$q = c-r$ 位	b 位
m 位		

$i = j \bmod Q$

直接映射　全相联映射

某一主存块 j 按模 Q 映射到 缓存 的第 i 组中的 任一块

三、替换算法

4.3

1. 先进先出（FIFO）算法 并不能很好的使用局部性原理

2. 近期最少使用（LRU）算法

小结

成本高

直接 某一 主存块 只能固定 映射到 某一 缓存块 Cache利用率低

全相联 某一 主存块 能 映射到 任一 缓存块 Cache利用率高

组相联 某一 主存块 只能 映射到 某一 缓存 组 中的 任一块