第四章　存储器-层次结构设计

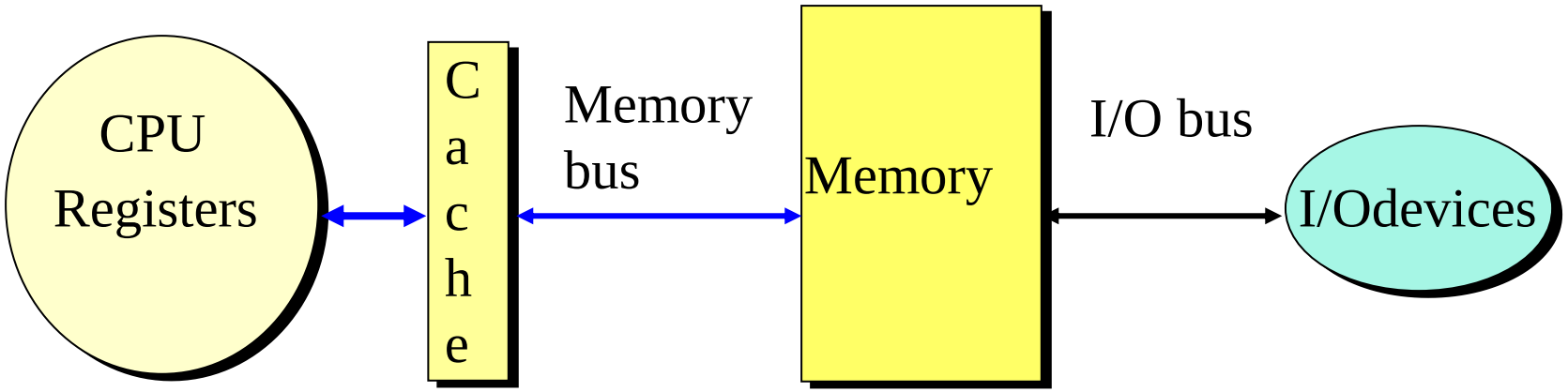
（教材第二章和附录B）

第4章 存储器-层次结构设计

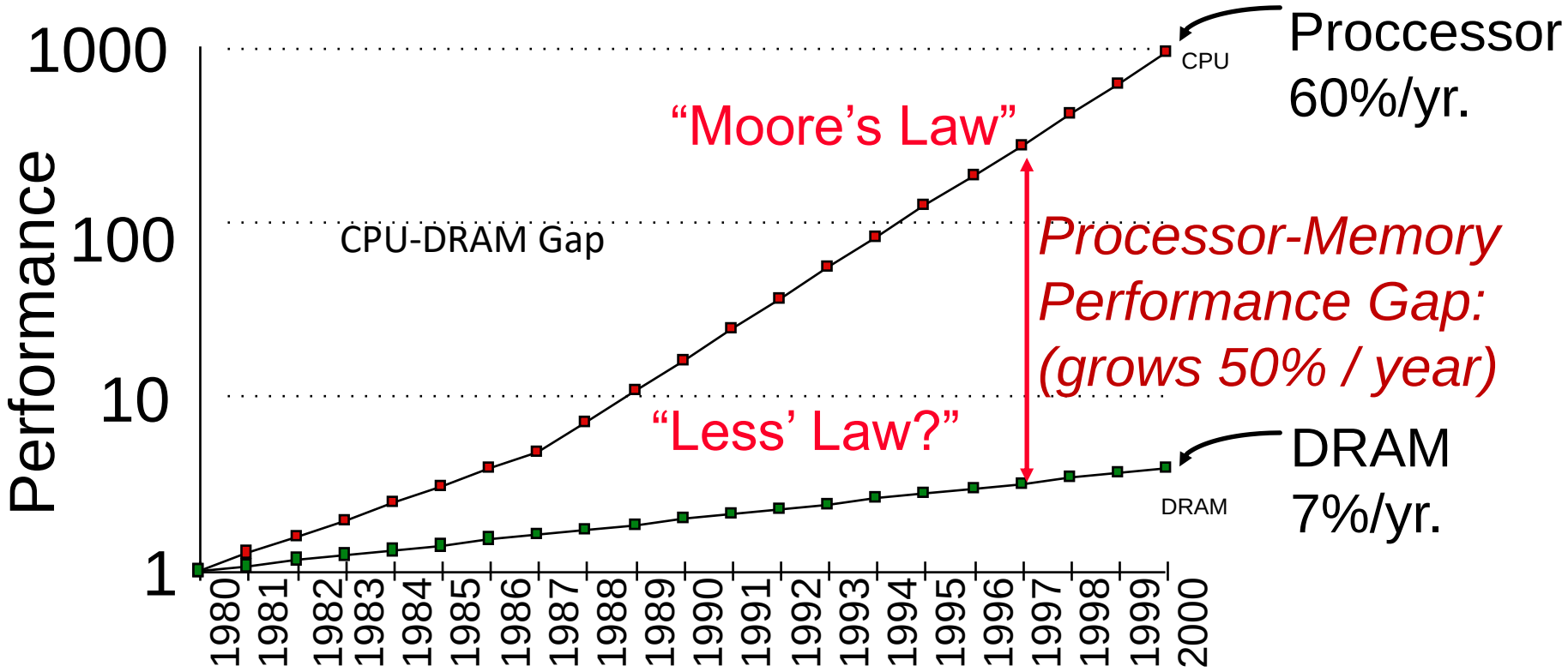
- 4.1 引言
- 4.2 Cache 性能
- 4.3 Cache优化技术
- 4.4 主存储器技术
- 4.5 虚拟存储器
- 4.6 谬误和易犯的错误
- 4.7 小结

4.1 引言

- 存储器存在什么问题？
  - 处理器-存储器之间的性能差距



微处理器-存储器间的性能差距



- **1980：微处理器没有Cache**
- **1995： 芯片集成2级cache（1989 Intel 首次集成cache在芯片上）**

加快存储器的速度

❖ 存储器硬件的部件特征：

- 靠近CPU:更小的存储器更快也更贵
- 离开CPU:更大的存储器更慢也更便宜

❖ 目标

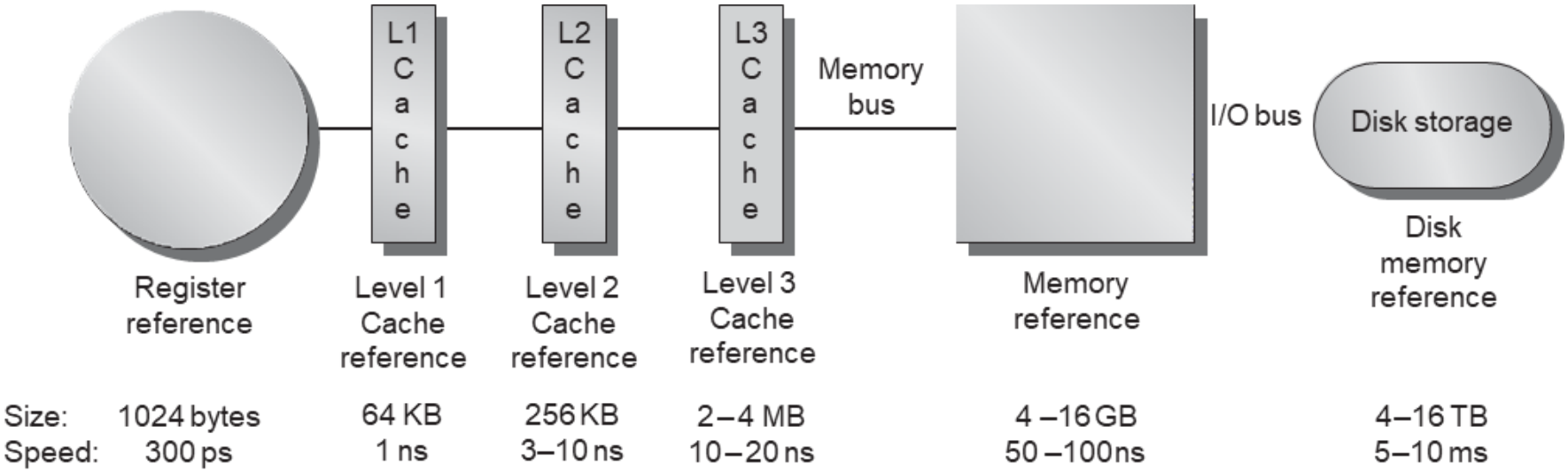
- 具有最小存储器的速度和最大存储器的容量
- 价格接近最便宜级别的存储器

加快存储器速度的方法

- 利用局部性原理：
- 多数程序不会一次访问所有的代码或数据
  - 时间局部性： 如果一项被访问，则该项趋向于近期又被访问
    - 使最近访问过的数据项尽量靠近处理器
  - 空间局部性：如果一项被访问，附近的项趋向于近期被访问
    - 把最近访问过的连续字（块）移向处理器
- 方法：采用层次结构存储系统
  - 基于存储器的各层（级）具有不同的速度与大小
  - 靠CPU越近的级：速度越快，容量越小，价格越贵

现代计算机系统中的存储器层次结构

- 利用局部性原则：
  - 用最便宜的技术提供给用户尽可能大的存储器
  - 用最快的技术提供访问速度



Cache是什么

- 小而快的存储器，用于改善对慢速存储器的平均访问时间。
- 在计算机结构中，几乎每个存储部件都是一个cache！
  - 寄存器是变量的“cache”– —软件管理
  - 一级 cache 是二级 cache 的“cache”
  - 二级 cache是主存的“cache”
  - 主存是磁盘（虚拟存储器）的 “cache”

Cache组织的基本单位---块（block，line）

- Caching 是一个在处理器、操作系统、文件系统和应用程序中使用的一般概念。
- Cache：按块（block，line）进行管理的。Cache被分割成大小相同的块。对Cache的访问是以块为基本单位。

4个问题：针对存储器层次结构设计者

对于存储器层次结构设计者，存在4个问题：

- 问题1：当把一个块（行）从主存调入Cache时，可以放到Cache的哪些位置上?（映象规则） *(块放置)*
  - 直接映像 Direct Mapped，全相联 Fully Associative, 组相联 Set Associative
- 问题2：如何判断所要访问的块命中Cache？ *(块识别)*
  - 标识/块（Tag/Block）
- 问题3：当读取块不在Cache中，发生失效时，应替换哪一块？*(块替换)*
  - Random, LRU,FIFO
- 问题4：对块进行写时，采用什么策略？ *(写策略)*
  - 写命中时：写回Write Back ，或 写直达Write Through
  - 写未命中时：写分配Write Allocate，或写不分配Write Non-allocate

问题1：映像规则

- 直接映像 **Direct mapped**
  - 块只能放在Cache中唯一的位置
- 全相联 **Fully associative**

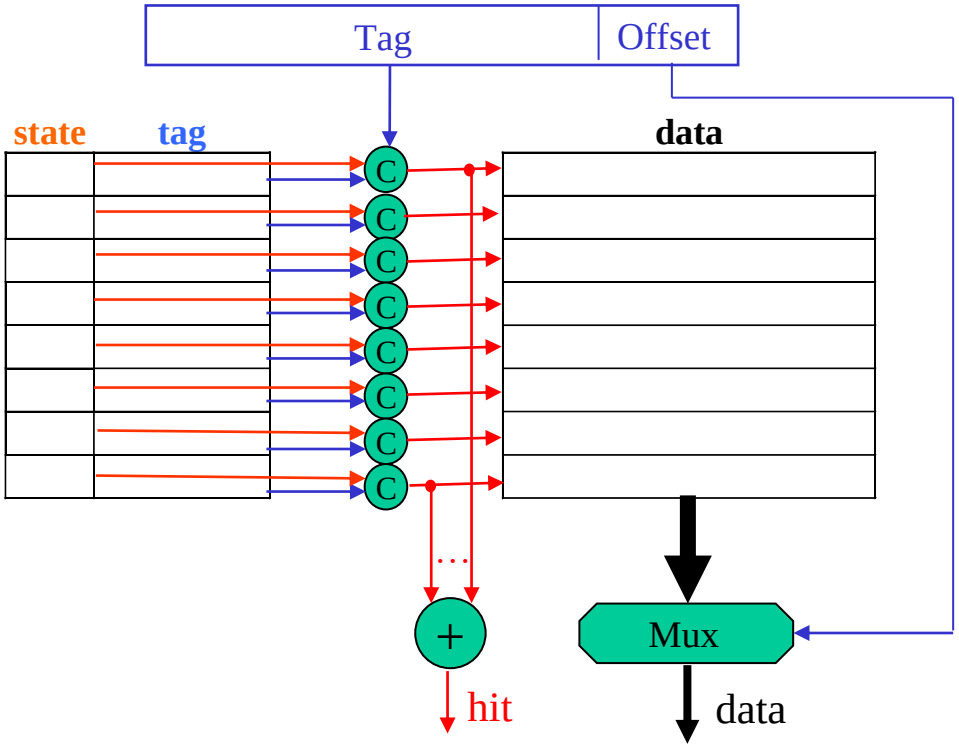
  块可以放在Cache中的任意位置
- 组相联 **Set associative**
  - 块能够放在Cache一组中任意一块位置

问题1：映像规则

- *直接映像也可以称为1-way set associative*
- *全相联也称为m-way set-associative（设一个cache有m块）*

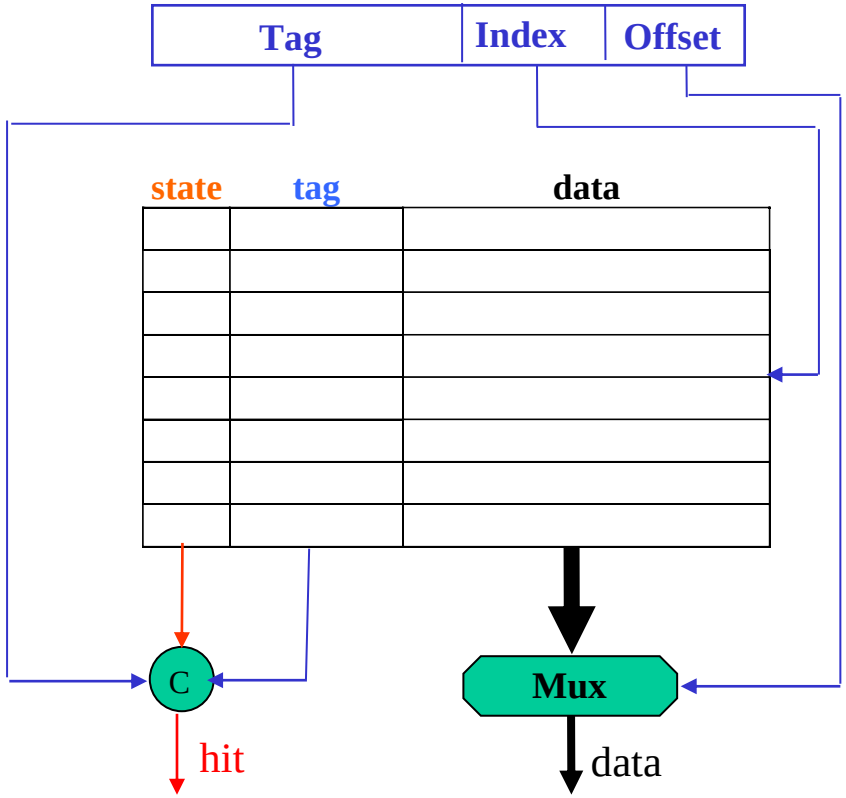
全相联

- 命中率高
- 硬件复杂、延迟大



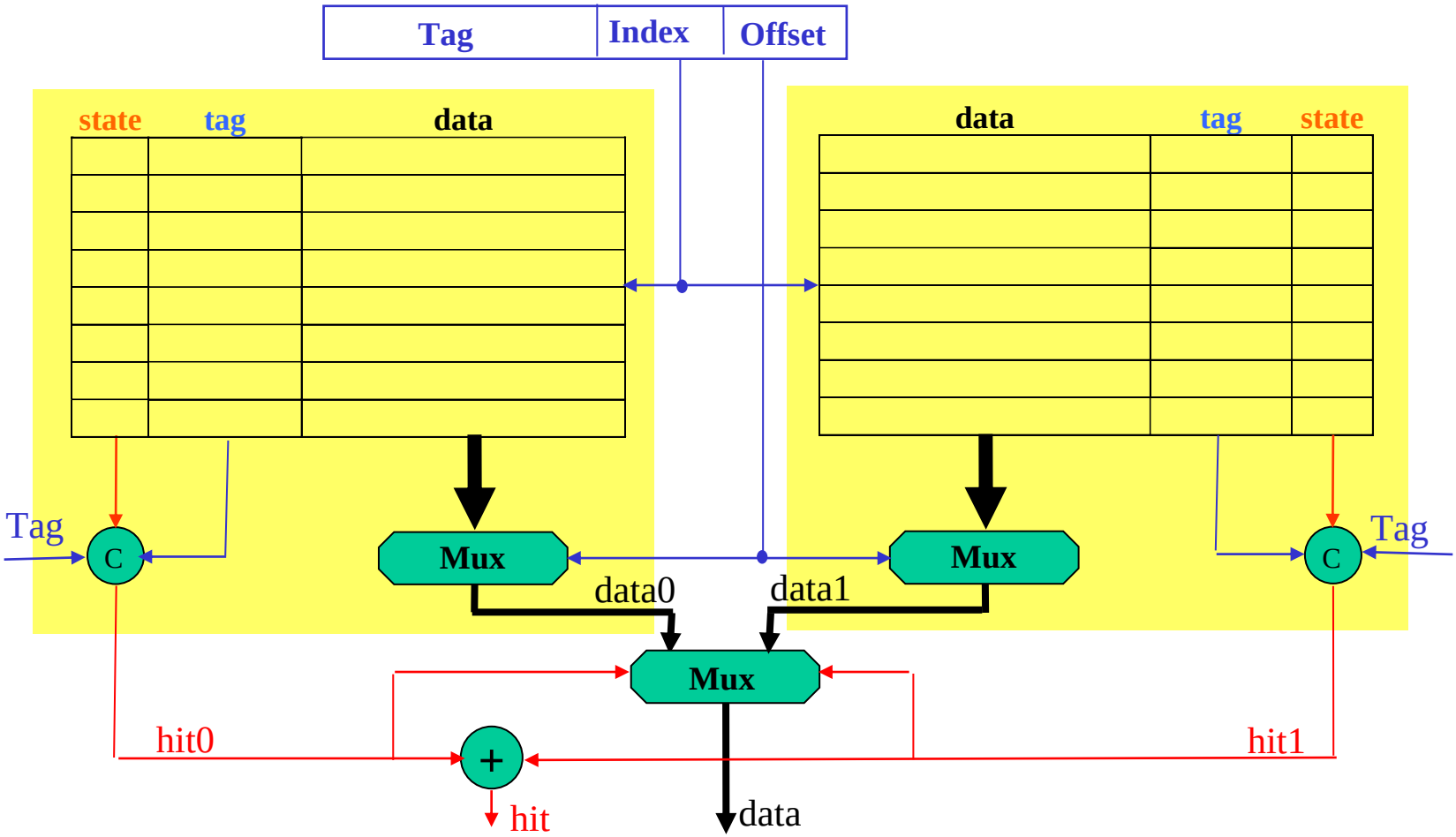
直接相联

- 硬件简单、延迟最小
- 命中率低



组相联

- 硬件复杂度介于全相联和直接相联之间



问题2：块识别

- Cache的每个块都有一个标识 tag ：存放CPU访问数据所在块的主存物理地址中的高位部分（主存多块映射到cache一块）
- 当CPU访问cache时，将比较主存地址高位与cache tag-----如果两者相同，表示cache命中即数据在cache中
- 通常，每个cache块还有一位有效位 valid bit ：表示cache块的内容是否有效

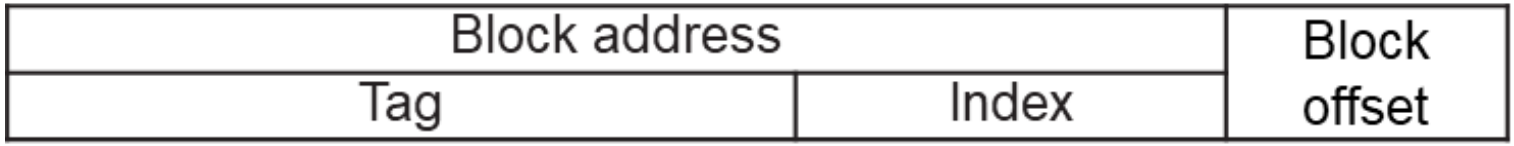


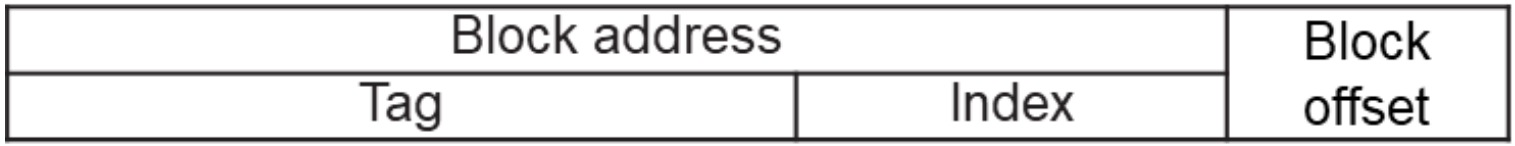
Figure B.3 The three portions of an address in a set associative or direct-mapped

组相联或直接映射缓存中地址的 3 个组成部分。标记用于检查组中的所有块, 索引用于选择该组。块偏移是块中所需数据的地址。全相联缓存没有索引字段

物理地址的格式

- 索引 Index 字段选择
  - 在直接映像Cache中，选择块
  - 在组相联Cache中，选择某一组中的块
  - 索引位数： log2(#块数) ——直接映像caches，前例Cache有8块，索引3位

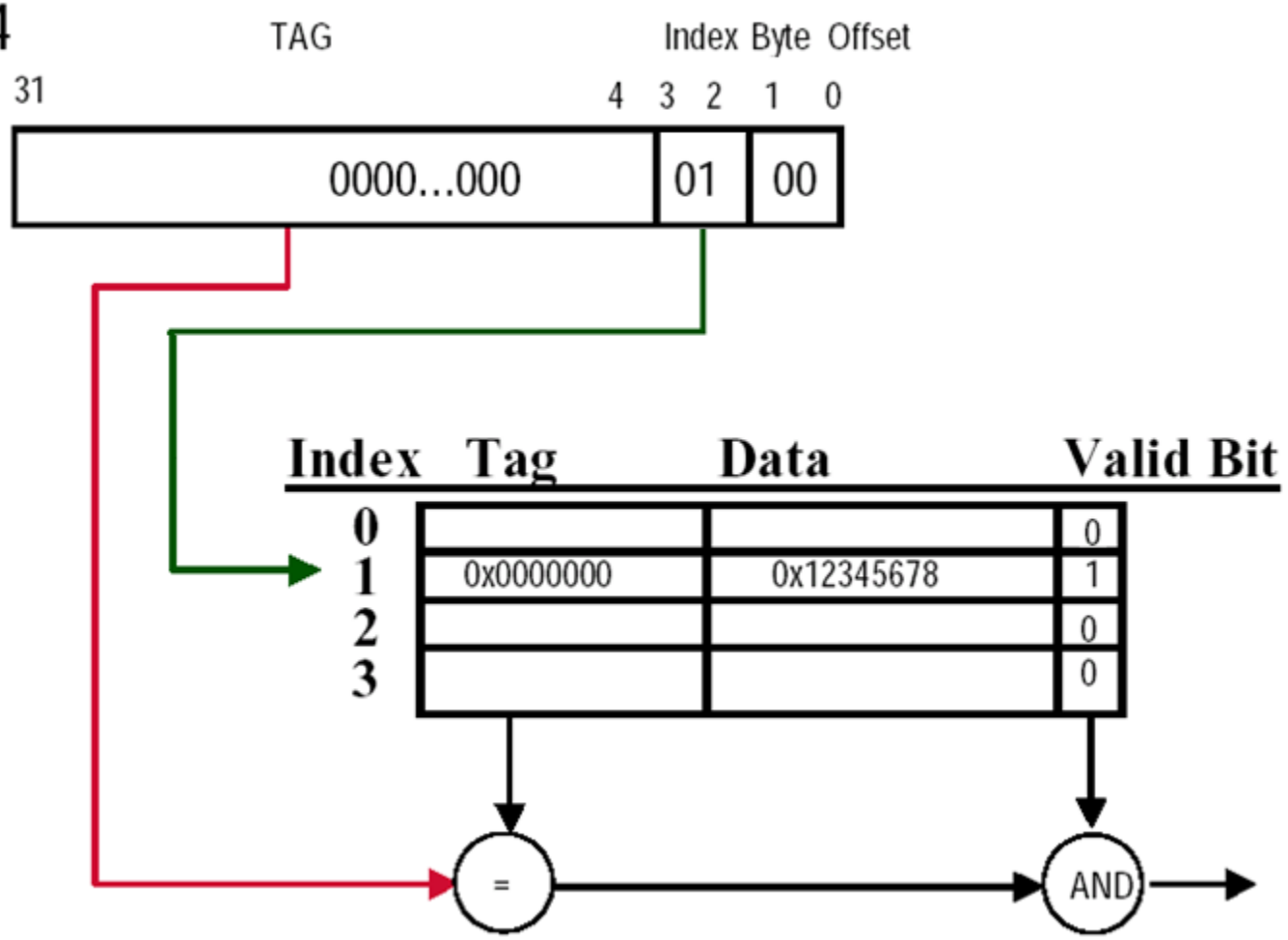
    log2(#组数) ——组相联caches，前例Cache有4组，索引2位
- 字节位移量 Block Offset 字段选择
  - 块中的某个字节
  - 位数： log2(块字节数)。如一块64B，则字节位移量需要6位
- 标识Tag 用于查找在Cache或一组中的匹配块
  - 位数：物理地址位数 – 索引位数 – 位移量位数



直接映像Cache例子（1-word Blocks）

- 设Cache有4块，每块1个字即4个字节

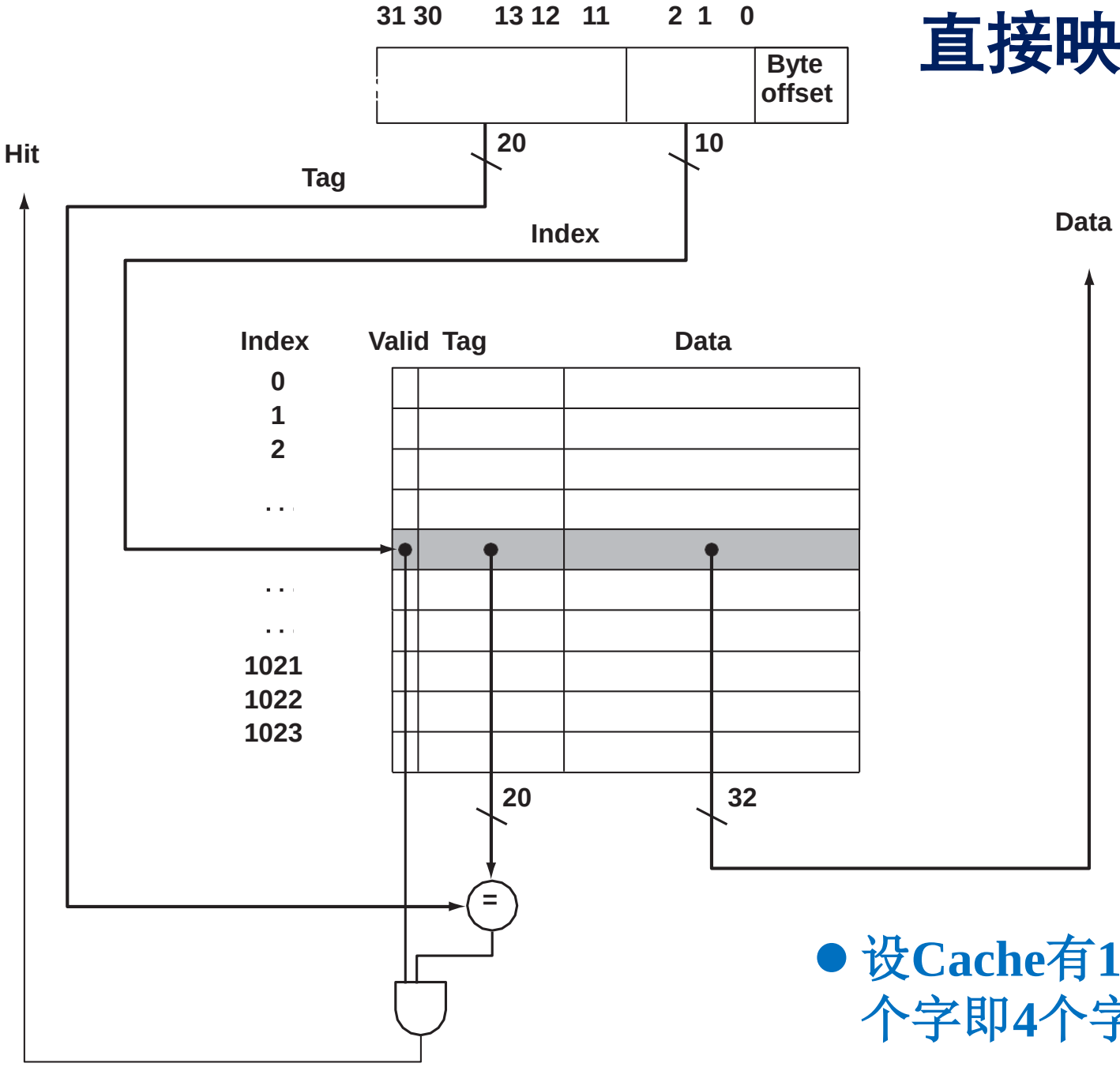
LOAD R1, 0x04



MEMORY

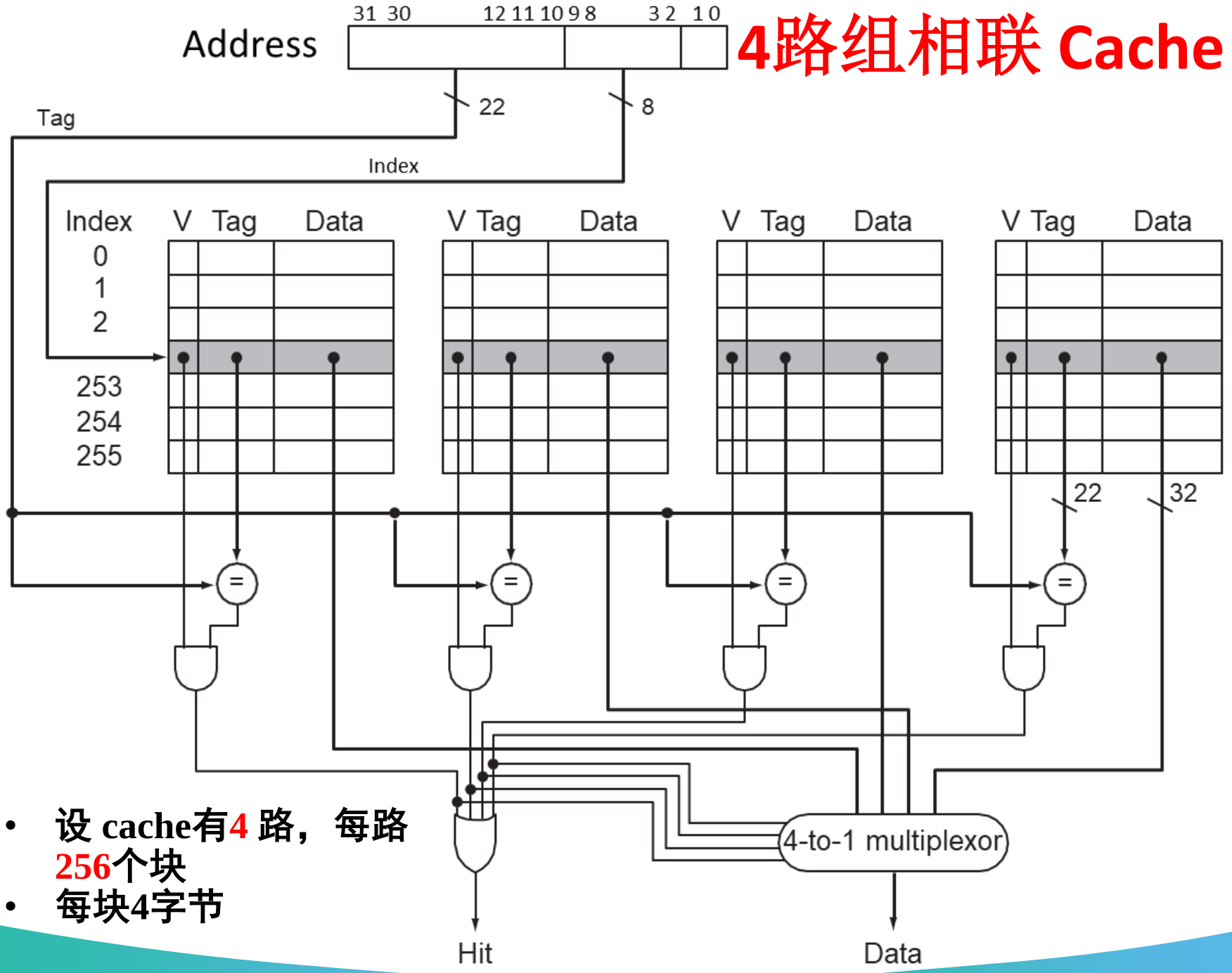
Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

直接映射Cache



- 设Cache有1024块，每块1个字即4个字节

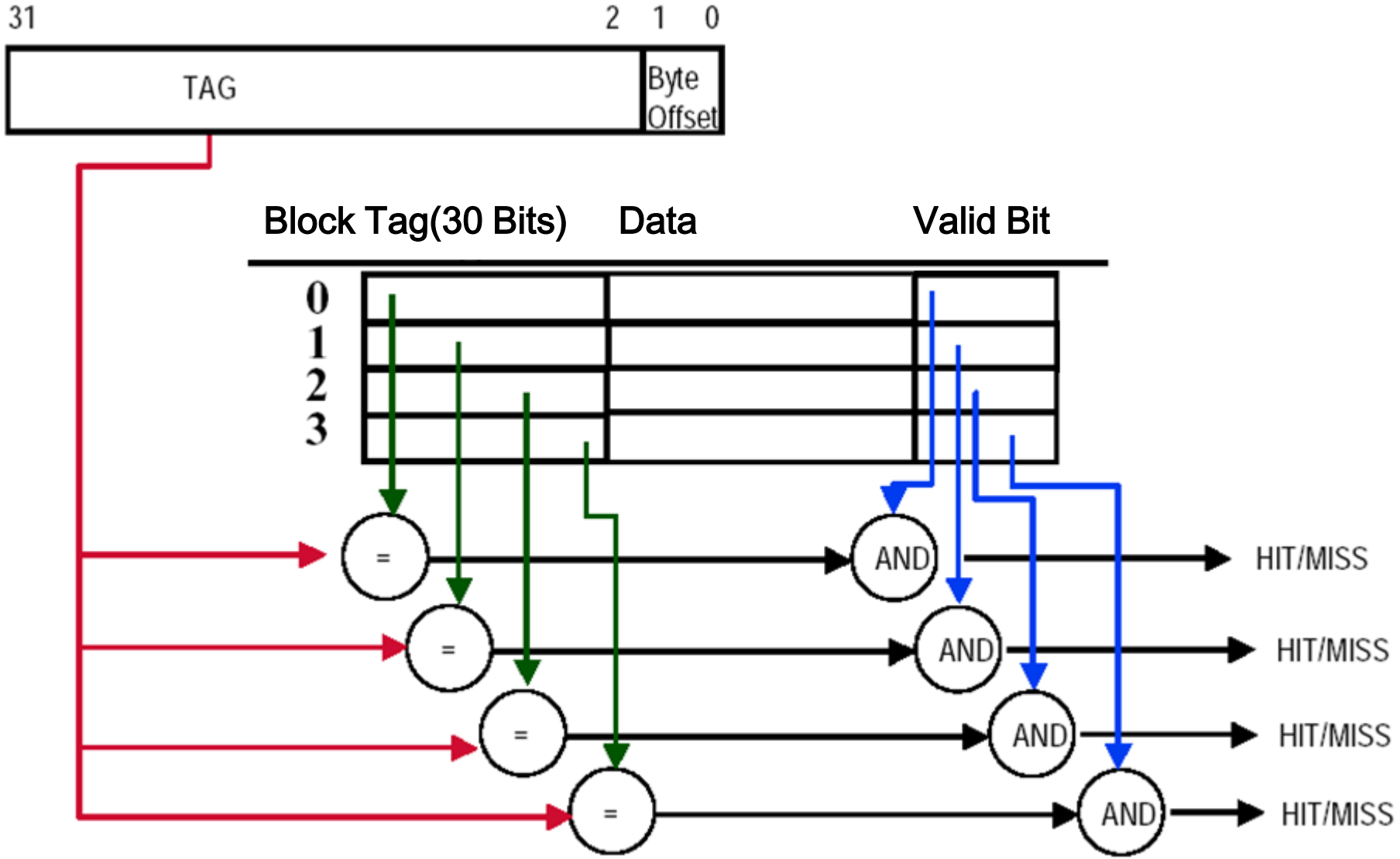
4路组相联 Cache



- 设 cache有4 路，每路256个块
- 每块4字节

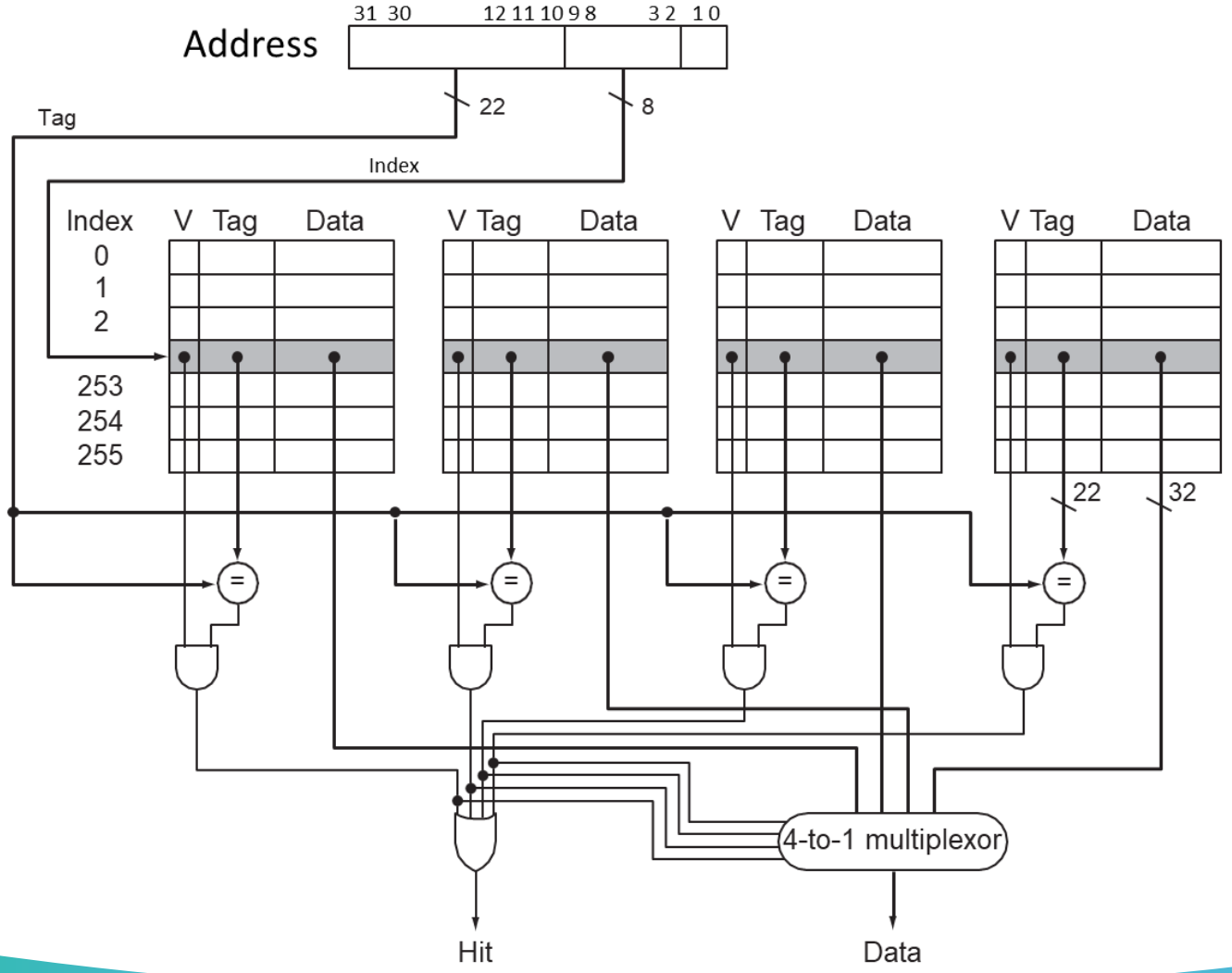
全相联Cache例子（1-word Blocks）

- 设Cache有4块，每块1个字



问题3：块替换

- 直接映像 cache，仅有唯一的一块能够被替换
- 对于组相联和全相联 caches，则有N块替换位置（ N是相联度）



块替换策略

- 3种替换策略
  - 随机替换 —— 随机选择被替换的一块
    - 硬件容易实现，需要随机数产生器
    - 均匀使用一组中的各块
    - 可能替换即将被访问的那一块
  - 最近最少使用Least-recently used (LRU)——选择一组中最近最少被访问的块作为被替换块
    - 假定最近被访问的块很可能会一再访问
    - Cache中需要额外的位记录访问历史
  - 先进先出(FIFO)——*选择一组中最先进入cache的一块*

Figure B.4

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Figure B.4 Data cache misses per 1000 instructions comparing least recently used, random, and first in, first out replacement for several sizes and associativities.

每千条指令的数据缓存缺失，比较了几种不同大小和相联度的最近最少使用、随机和先进先出替换方式

问题4：写策略

- 当数据写进cache时，也会写入主存吗？
  - 如果数据写cache的同时也写主存，cache被称为写直达（*write-through cache*）
    - Cache中的数据可以随时丢弃——主存中有最新的数据
    - Cache 控制位：只需要一位 *valid* bit
    - 主存（或其他处理器）总是有最新的数据
  - 如果数据写cache时不写主存，cache被称为写回（*write-back cache*）
    - 不能丢弃cache中的数据——可能需要写回到主存（替换时）
    - Cache 控制位：需要 *valid位* 和*dirty* 位
    - 带宽较小，因为对同一块的多次写仅需要对主存写一次
- 写直达优点：保持数据一致性，读缺失不会导致替换时的写操作，实现简单。
- 写回优点：写cache的速度更快，主存带宽更低，减少缺失次数。

写直达cache：写停顿和写缓冲

- 写停顿-CPU必须等待写操作完成
- 写缓冲
  - 一个小缓冲区：存放等待写入主存的几个值。
  - 为了避免等待，很多CPU都使用一个写缓冲。
  - 在写操作集中时，这个缓冲很有作用。
  - 它不能完全消除停顿。例如：如果写的数据量大于缓冲区， 就可能产生停顿。

写命中时采取的策略

- **写穿透（Write Through）**
  - 写Cache的同时写内存
  - 内存里的数据永远是最新的，Cache替换时直接扔掉
  - Cache块管理简单，只需有效位
- **写回（Write-back）**
  - 只写Cache不写内存
  - 替换时要把Cache块写回内存
  - Cache块状态复杂一些，需要有效位和脏位
- **写回/写穿透的使用**
  - L1到L2用写穿透的多，L2较快
  - L2到内存用写回的多，内存太慢了
  - 龙芯2号两级都采用写回策略。

写缺失

- 写缺失

  对cache进行写时，如果要写的块不在cache，有两种策略选择：

  - 写分配（Write allocate）
    - 写失效时，把所写单元所在的块调入Cache，然后再进行写命中操作。写的值在Cache中。这与读失效类似。
  - 不按写分配 Write around (no write allocate)
    - 写失效时，直接将值写入下一级存储器而不将相应的块调入Cache。
    - 写的值不在cache中（要写的块本来也不在cache中）。

  - 通常，写回caches采用写分配；
    写直达caches采用不按写分配。

例子

例：假设有一个全相联映射有多个项的Cache，采用写回策略，刚开始时Cache为空。有下面5个存储器操作（方括号是地址）：

1 write Mem[100];
2 write Mem[100];
3 Read Mem[200];
4 write Mem[200];
5 write Mem[100];

按写分配　不按写分配
命中次数　缺失次数

例子

例：假设有一个全相联映射有多个项的Cache，采用写回策略，刚开始时Cache为空。有下面5个存储器操作（方括号是地址）：

1 write Mem[100];
2 write Mem[100];
3 Read Mem[200];
4 write Mem[200];
5 write Mem[100];

不按写分配和按写分配
命中次数 缺失次数

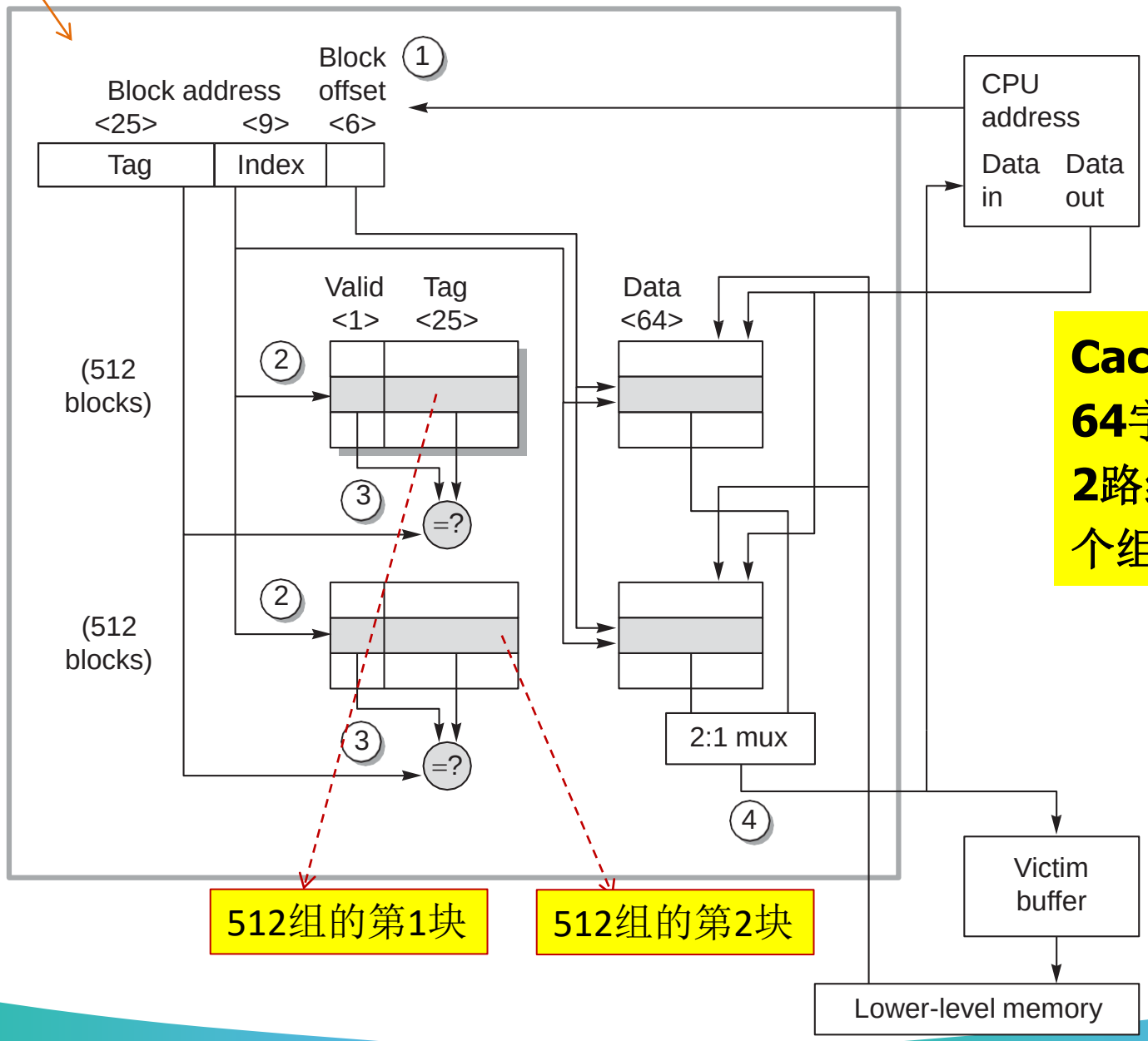
Answer：

for no-write allocate misses: 1,2,3,5
hit : 4

for write allocate misses: 1,3
hit : 2,4,5

物理地址40位

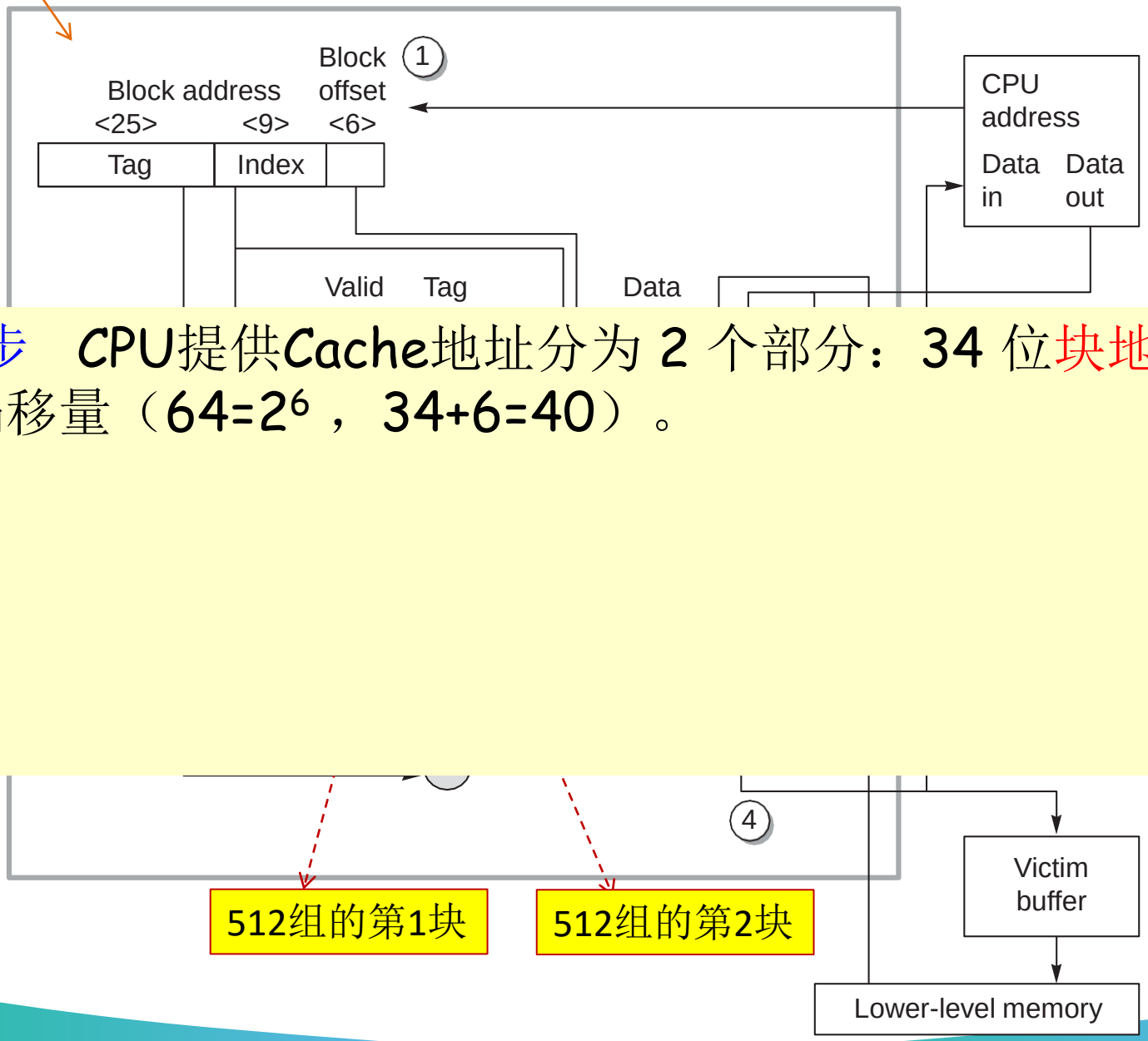
Figure B.5： AMD Opteron数据 cache



Cache64KB，块64字节(1K个块)，2路组相联（512个组）

物理地址40位

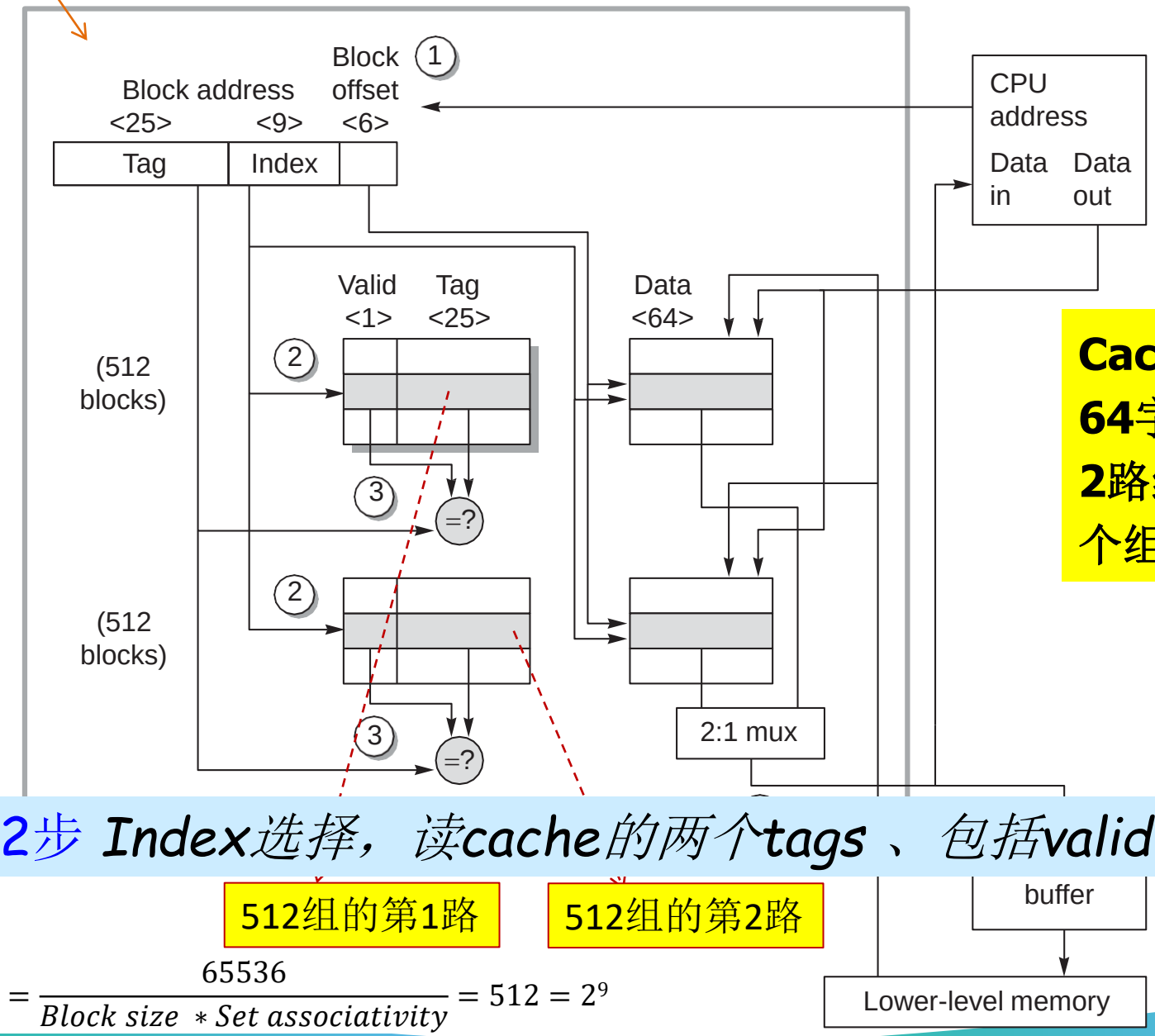
Figure B.5： AMD Opteron数据 cache



第1步　CPU提供Cache地址分为 2 个部分：34 位块地址， 6位块偏移量（64=2^6，34+6=40）。

物理地址40位

Figure B.5： AMD Opteron数据 cache



Cache64KB，块64字节(1K个块)，2路组相联（512个组）

第2步 Index选择，读cache的两个tags 、包括valid位和数据。

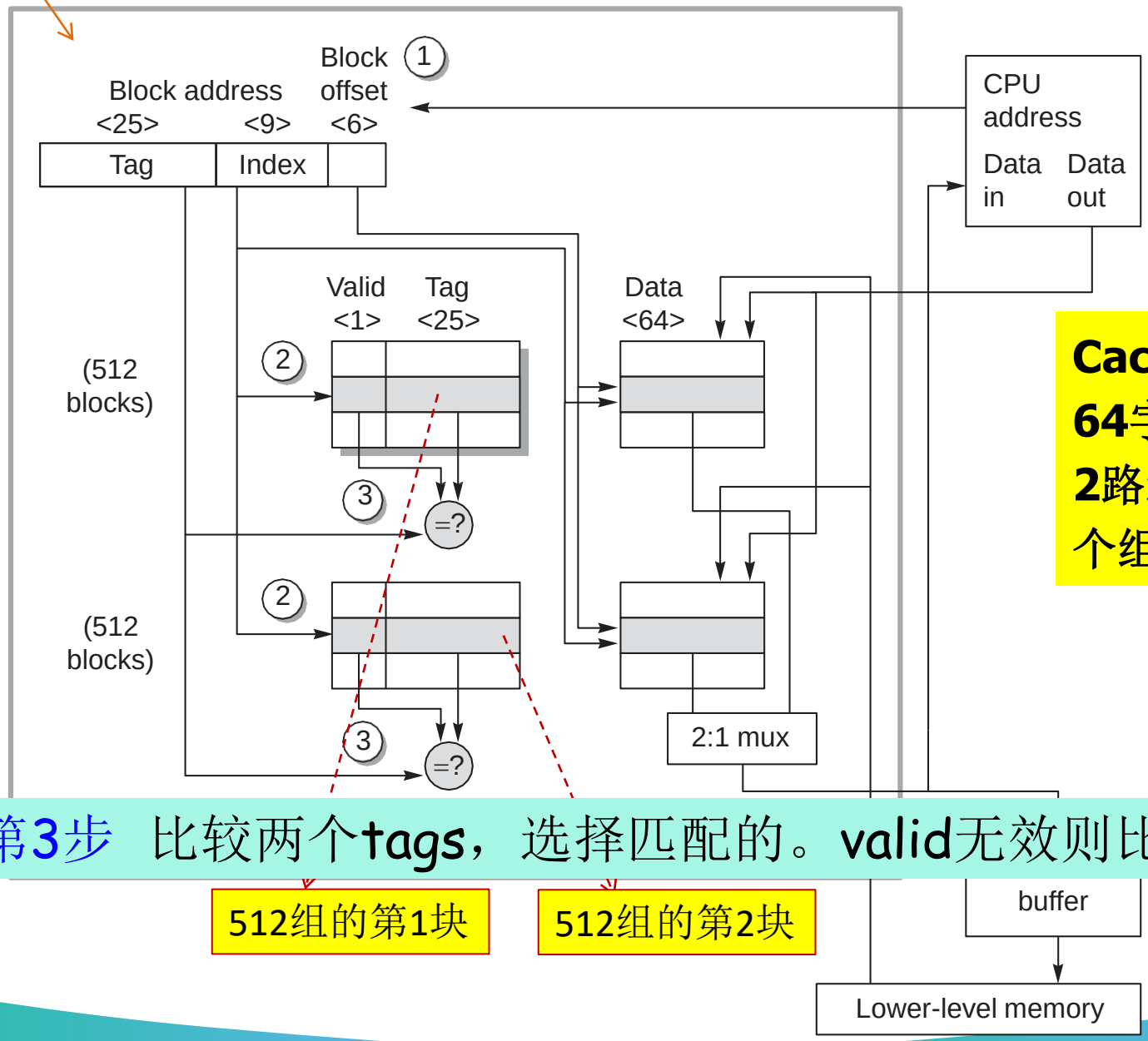
512组的第1路

512组的第2路

$$2^{\text{Index}} = \frac{65536}{Block\ size\ *\ Set\ associativity} = 512 = 2^9$$

物理地址40位

Figure B.5： AMD Opteron数据 cache



Cache64KB，块64字节(1K个块)，2路组相联（512个组）

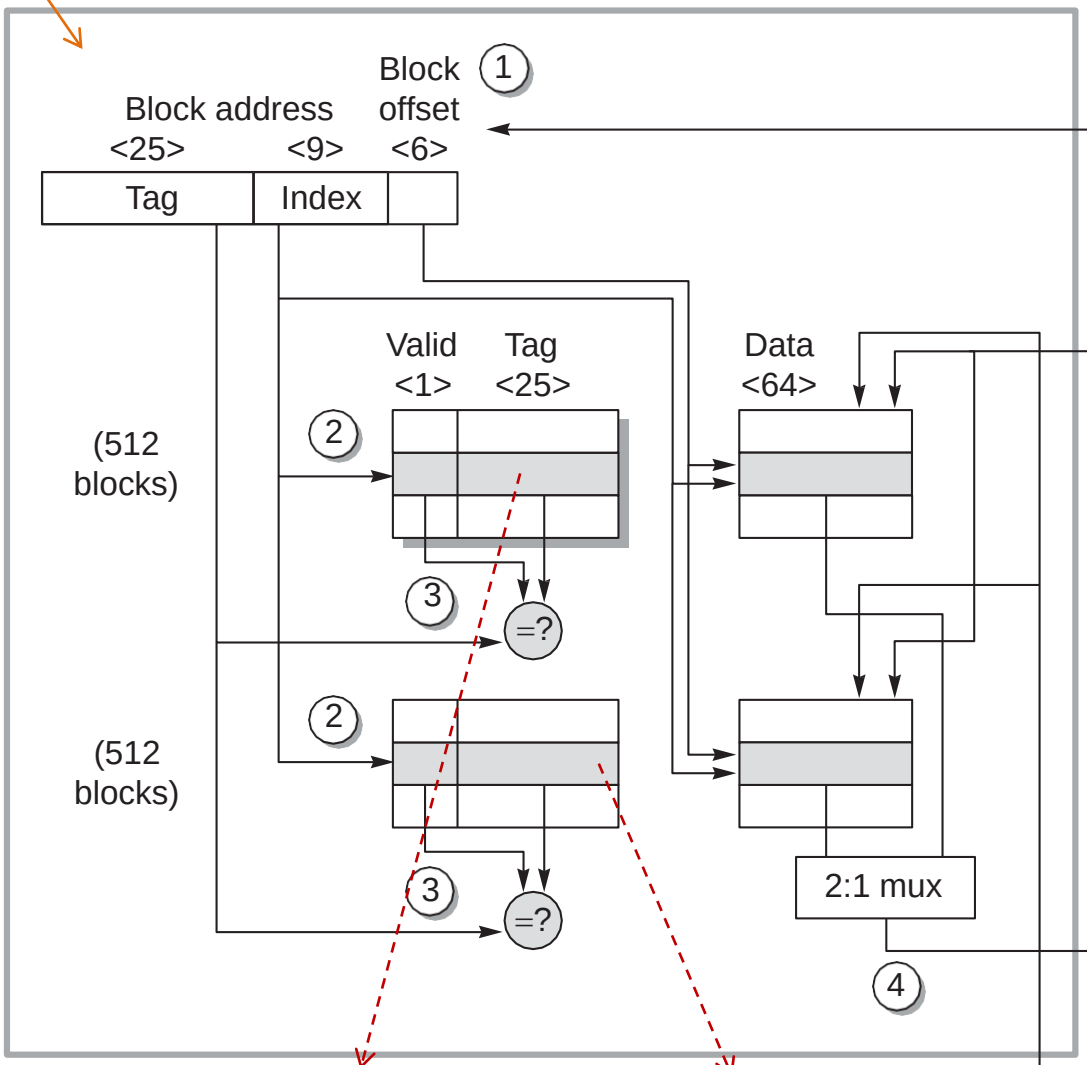
第3步 比较两个tags，选择匹配的。valid无效则比较结果无效。

512组的第1块

512组的第2块

物理地址40位

Figure B.5： AMD Opteron数据 cache



第4步　如果一个 tag匹配，CPU从cache中获得数据；否则从下级存储器读取。

允许2个时钟周期完成这4步操作，如果随后2个时钟周期内的指令要使用这个load结果则将等待。

分离cache与一体caches

- 一体cache
  - 所有存储访问都是对单个 cache进行的。
  - 需要较少的硬件；性能更低
- 分离的指令cache与数据cache
  - 指令与数据存放在不同的 cache中
  - 需要额外的硬件