第7章 存储系统（Memory Hierarchy Design）

Ideally one would desire an indefinitely large memory capacity such that any particular… word would be immediately available… We are… forced to recognize the possibility of constructing a hierarchy of memories each of which has greater capacity than the preceding but which is less quickly accessible.

A. W. Burks, H. H. Goldstine,
and J. von Neumann,
Preliminary Discussion of the Logical Design of an Electronic Computing Instrument (1946).

第7章 存储系统（Memory Hierarchy Design）

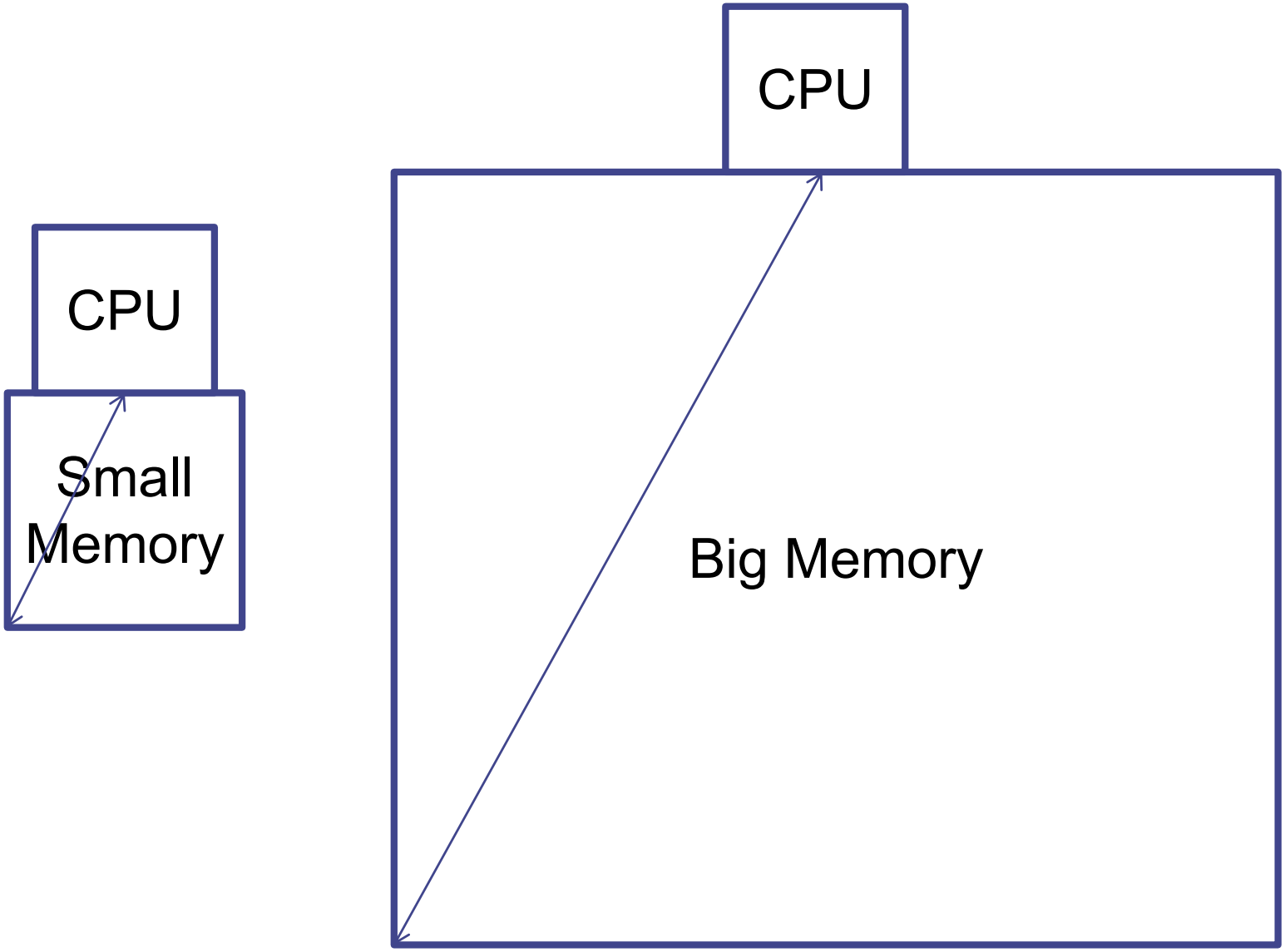
7.1 存储系统的基本知识
7.2 Cache基本知识
7.3 降低Cache不命中率
7.4 减少Cache不命中开销
7.5 减少命中时间
7.7 虚拟存储器
7.8 实例：AMD Opteron的存储器层次结构

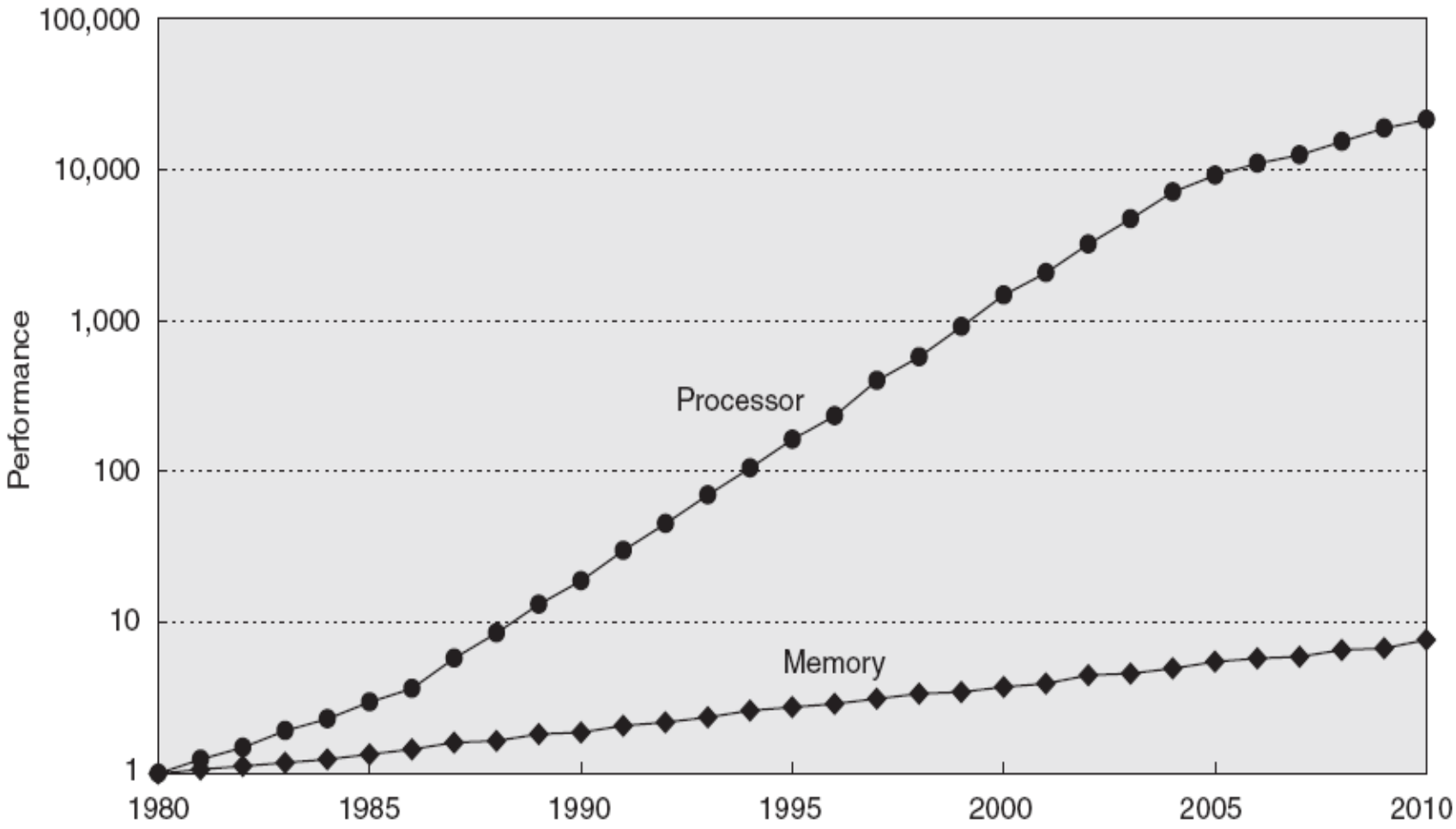
7.1 存储系统的基本知识

7.1.1 存储系统的层次结构

- 计算机系统结构设计中**关键的问题之一**：
  - 如何以合理的价格，设计容量和速度都满足计算机系统要求的存储系统？
- 理想的存储器
  - 容量大
  - 速度快
  - 价格低

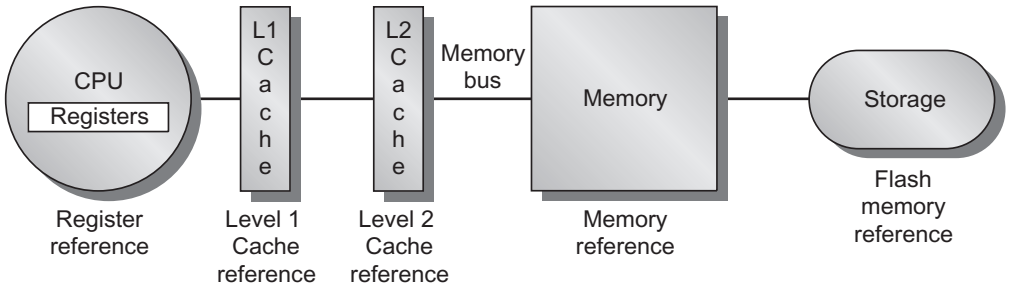
Physical Size Affects Latency



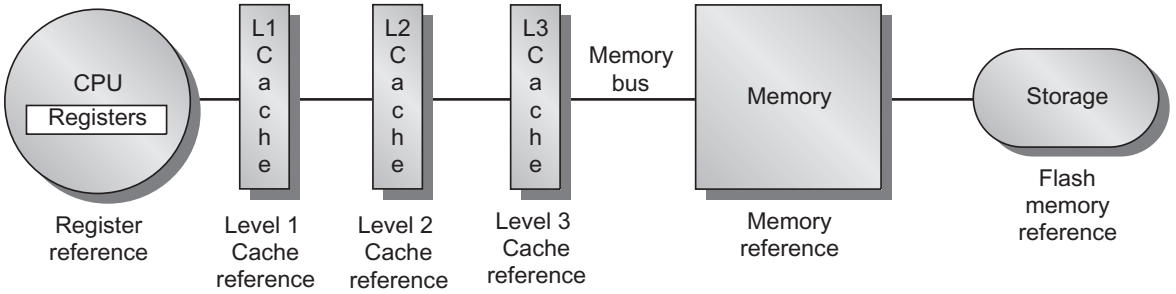


1980年以来存储器和CPU性能随时间而提高的情况

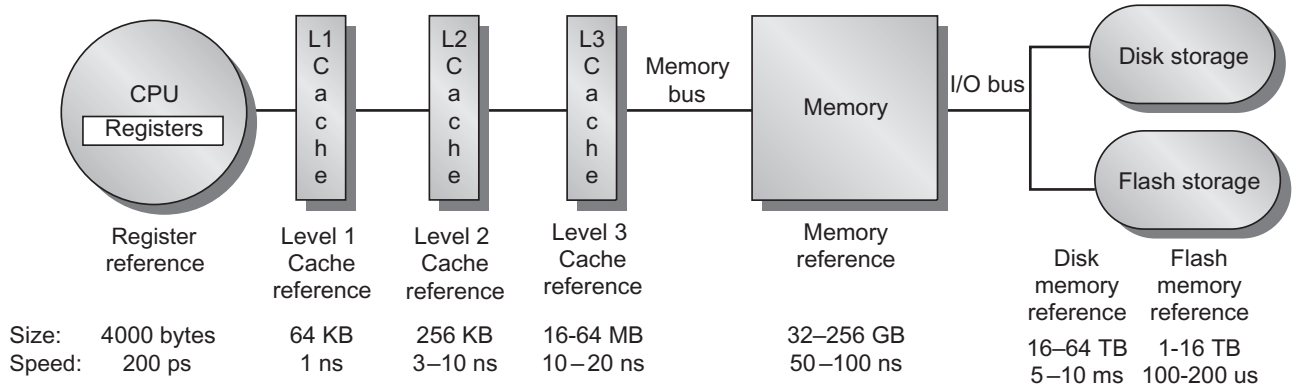
（以1980年时的性能作为基准）



(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

如何达成目标？

采用多种存储器技术，构成多级存储层次结构。

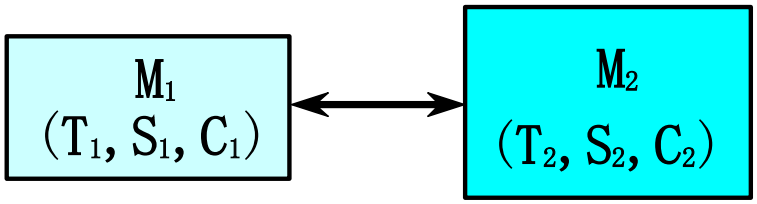
- **程序访问**的局部性原理：对于绝大多数程序来说，程序所访问的指令和数据在地址上不是均匀分布的，而是相对簇聚的。
- 程序访问的局部性包含两个方面
  - 时间局部性：程序马上将要用到的信息很可能就是现在正在使用的信息。
  - 空间局部性：程序马上将要用到的信息很可能与现在正在使用的信息在存储空间上是相邻的。

- 假设第i个存储器M_i的访问时间为T_i，容量为S_i，平均每位价格为C_i，则
  - **访问时间：** $T_1 < T_2 < \ldots < T_n$
  - **容量：** $S_1 < S_2 < \ldots < S_n$
  - **平均每位价格：** $C_1 > C_2 > \ldots > C_n$
- 整个存储系统要达到的目标：从CPU来看，该存储系统的速度接近于M_1的，而容量和每位价格都接近于M_n的。
  - **存储器越靠近CPU，则CPU对它的访问频度越高，而且最好大多数的访问都能在M_1完成。**

7.1.2 存储层次的性能参数

下面仅考虑由M_1和M_2构成的两级存储层次：

- M_1的参数：S_1，T_1，C_1
- M_2的参数：S_2，T_2，C_2



1. 存储容量S

   - 一般来说，整个存储系统的容量即是第二级存储器M_2的容量，即$S=S_2$。

2. 每位价格C

$$C=\frac{C_1S_1+C_2S_2}{S_1+S_2}$$

当$S_1<<S_2$时，$C\approx C_2$

1. 命中率H 和不命中率F

- 命中率：CPU访问存储系统时，在M_1中找到所需信息的概率。

$$H=\frac{N_1}{N_1+N_2}$$

  - N_1 —— 访问M_1的次数
  - N_2 —— 访问M_2的次数

- 不命中率：F＝1－H

4. 平均访问时间T_A

$$T_A = HT_1 + (1-H)(T_1 + T_M)$$
$$= T_1 + (1-H)T_M$$
$$\text{或 } T_A = T_1 + FT_M$$

分两种情况来考虑CPU的一次访存：

- 当命中时，访问时间即为T_1（命中时间）
- 当不命中时，情况比较复杂。

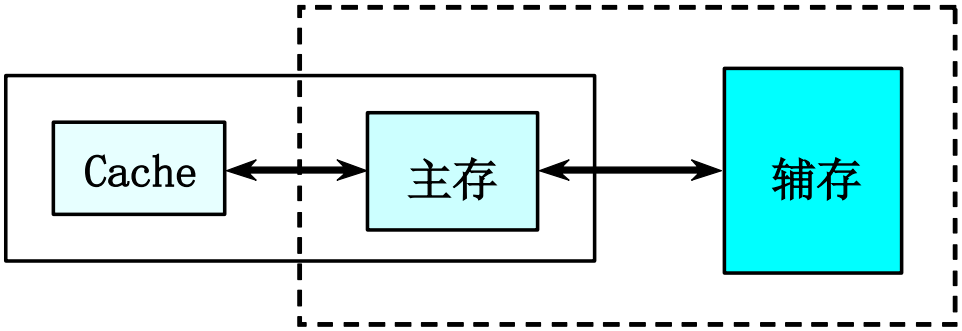
  不命中时的访问时间为：$T_2 + T_B + T_1 = T_1 + T_M$

  $$T_M = T_2 + T_B$$

  - 不命中开销T_M：从向M_2发出访问请求到把整个数据块调入M_1中所需的时间。
  - 传送一个信息块所需的时间为T_B。

7.1.3 三级存储系统

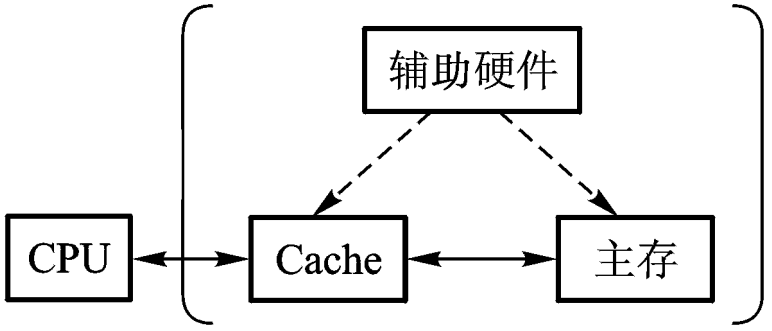
- 三级存储系统
  - **Cache（高速缓冲存储器）**
  - **主存储器**
  - **磁盘存储器（辅存）**
- 可以看成是由“Cache—主存”层次和“主存—辅存”层次构成的系统。



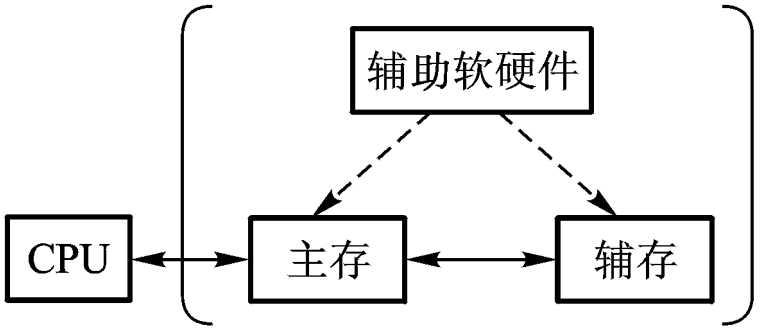
从主存的角度

“Cache—主存”层次：
弥补主存速度的不足

“主存—辅存”层次：
弥补主存容量的不足



(a) “Cachc-主存”层次



(b) “主存-辅存”层次

两种存储层次

“Cache－主存”与“主存－辅存”层次的区别

比较项目 \ 存储层次	“Cache －主存”层次	“主存－辅存”层次
目　　的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理实现	主要由专用硬件实现	主要由软件实现
访问速度的比值(第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程

7.1.4 存储层次的四个问题

Four questions about any level of the hierarchy

Q1: Where can a block be placed in the upper level?

(*Block placement*)

Q2: How is a block found if it is in the upper level?

(*Block identification*)

Q3: Which block should be replaced on a miss?

(*Block replacement*)

Q4: What happens on a write?

(*Write strategy*)

7.1.4 存储层次的四个问题

1. 当把一个块调入高一层(靠近CPU)存储器时，可以放在哪些位置上?

   （映象规则）

2. 当所要访问的块在高一层存储器中时，如何找到该块?

   （查找算法）

3. 当发生不命中时，应替换哪一块？

   （替换算法）

4. 当进行写访问时，应进行哪些操作?

   （写策略）

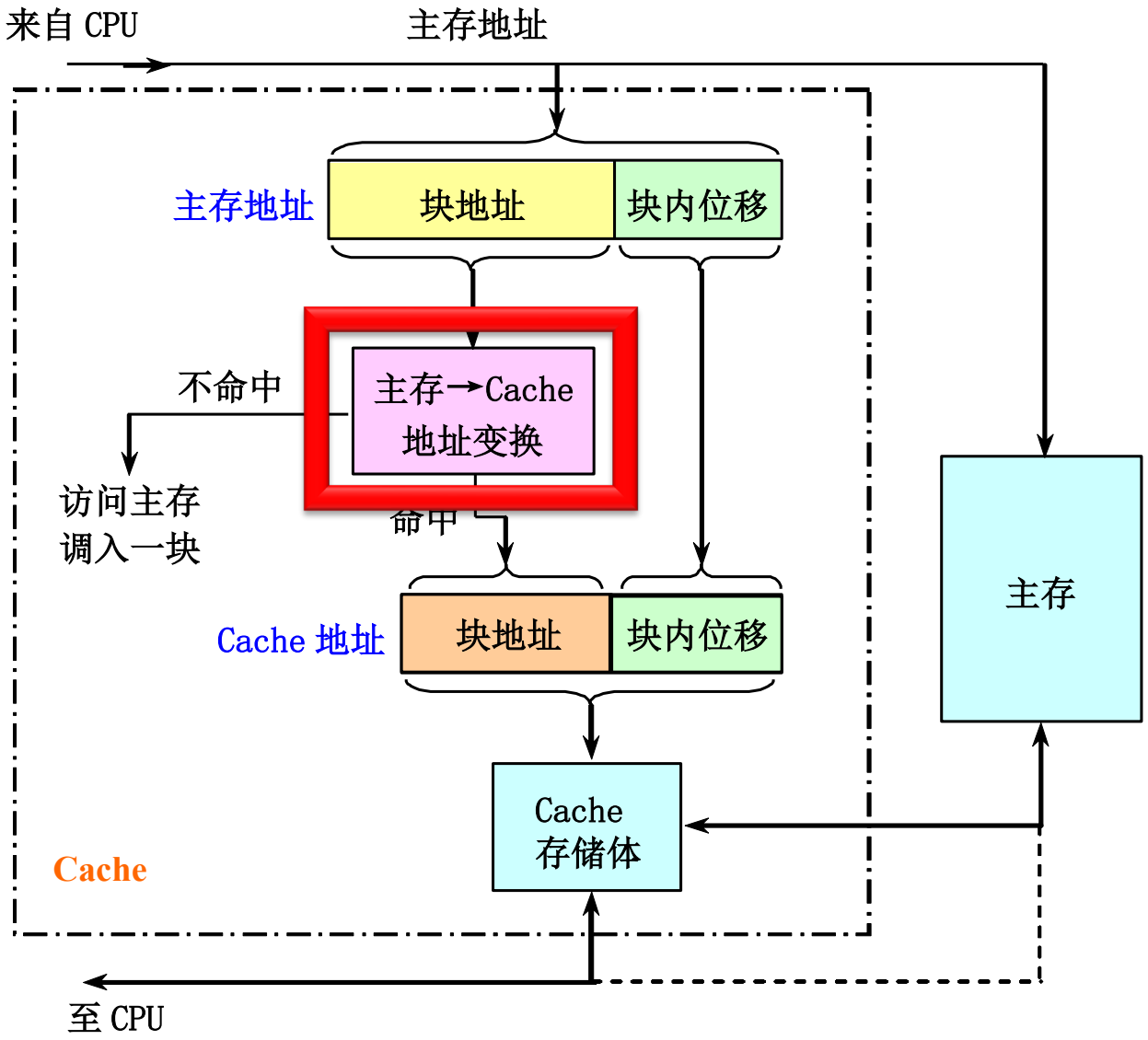
7.2 Cache基本知识

7.2.1 基本结构和原理

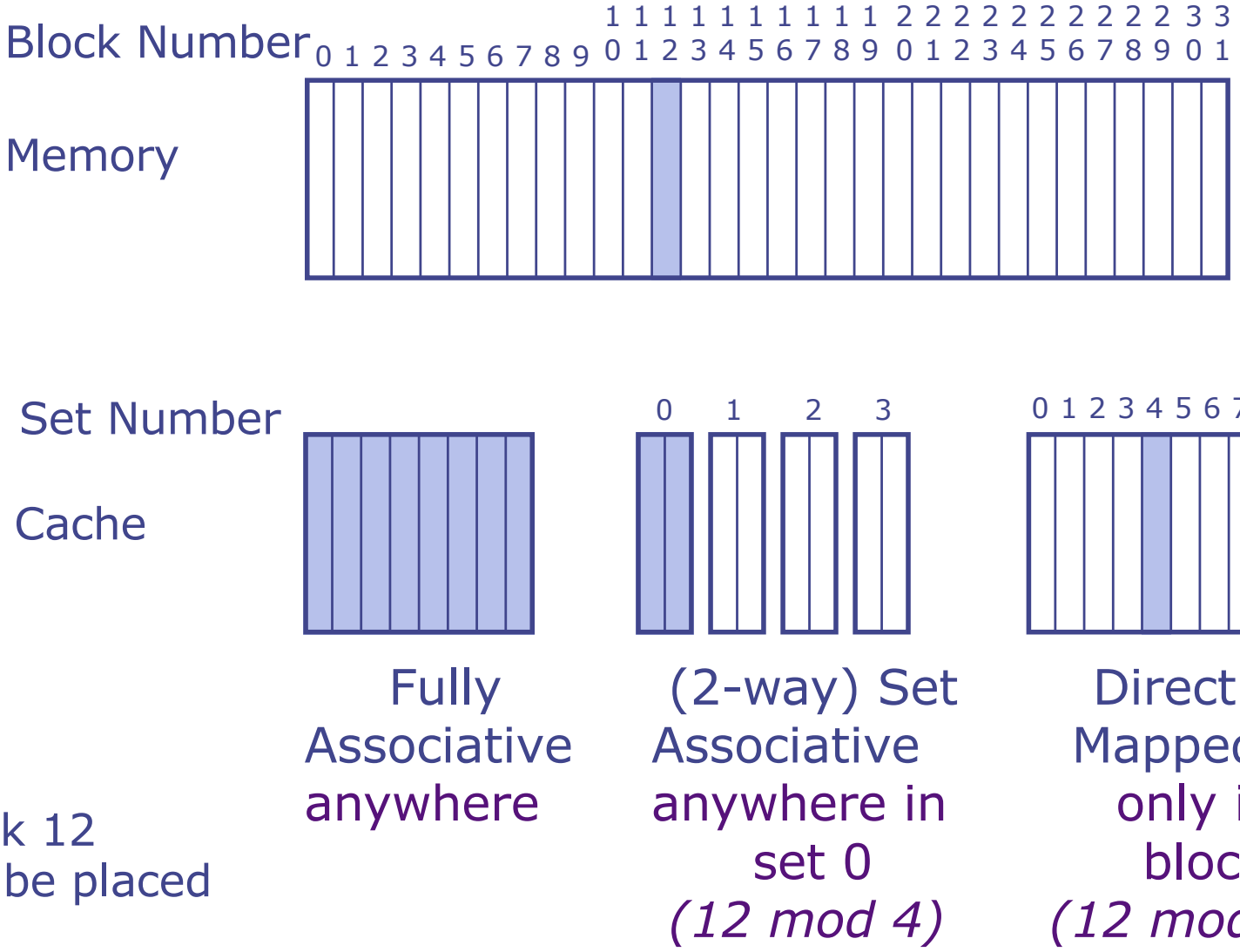
Cache是按块进行管理的。Cache和主存均被分割成大小相同的块。以块为单位调入Cache。

- 主存块地址（块号）用于查找该块在Cache中的位置。
- 块内位移用于确定所访问的数据在该块中的位置。

3. Cache的基本工作原理示意图



Placement Policy



7.2.2 映象规则

1. 全相联映象

- 全相联：主存中的任一块可以被放置到Cache中的任意一个位置。
- 对比：阅览室位置 —— 随便坐
- 特点：空间利用率最高，冲突概率最低，实现最复杂。

2. 直接映象

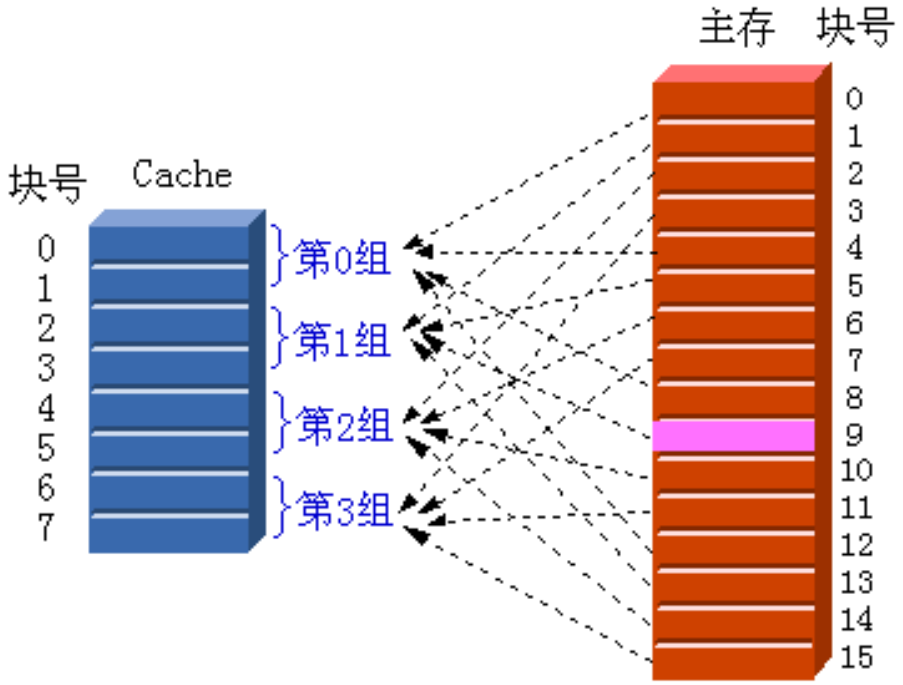
- 直接映象：主存中的每一块只能被放置到Cache中唯一的一个位置。（循环分配）
- 对比：阅览室位置 —— 只有一个位置可以坐
- 特点：空间利用率最低，冲突概率最高，实现最简单。
- 对于主存的第 i 块，若它映象到Cache的第 j 块，则：

$$j = i \bmod (M)$$ （M为Cache的块数）

- 设$M=2^m$，则当表示为二进制数时，j实际上就是i的低m位：

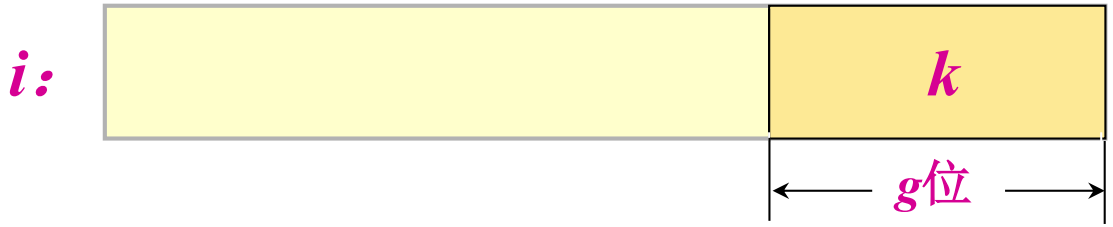
3. 组相联映象

- 组相联：主存中的每一块可以被放置到Cache中唯一的一个组中的任何一个位置。(直接映像到组，组内全相联)
- 组相联是直接映象和全相联的一种折衷



- 组的选择
  - 若主存第i 块映象到第k 组，则：

    $k = i \bmod (G)$　（G为Cache的组数）
  - 设$G=2^g$，则当表示为二进制数时，k 实际上就是i的低 g 位：



- 低g位以及直接映象中的低m位通常称为**索引（Index）**。

回到虚拟cache

- *n* 路组相联：每组中有n个块（$n=M/G$）。

  n 称为**相联度**。

  相联度越高，Cache空间的利用率就越高，块冲突概率就越低，不命中率也就越低。

	n（路数）	*G*（组数）
全相联	***M***	**1**
直接映象	**1**	***M***
组相联	***1<n<M***	***1<G<M***

- 绝大多数计算机的Cache：$n \leqslant 4$

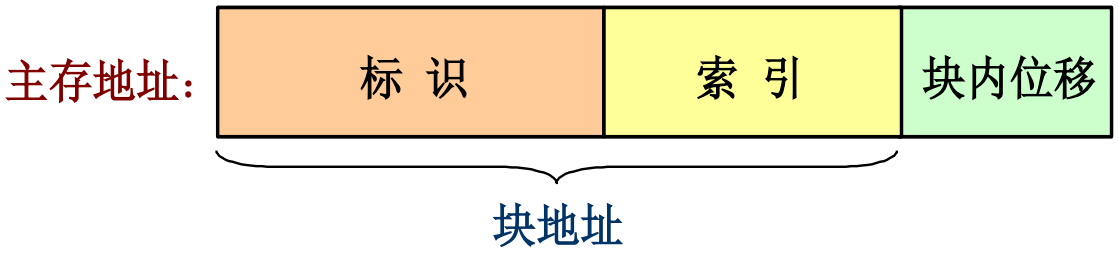
  Q：相联度是否越大越好？

7.2.3 查找算法

- 当CPU访问Cache时，如何确定Cache中是否有所要访问的块？
- 若有的话，如何确定其位置？

1. 通过查找目录表来实现

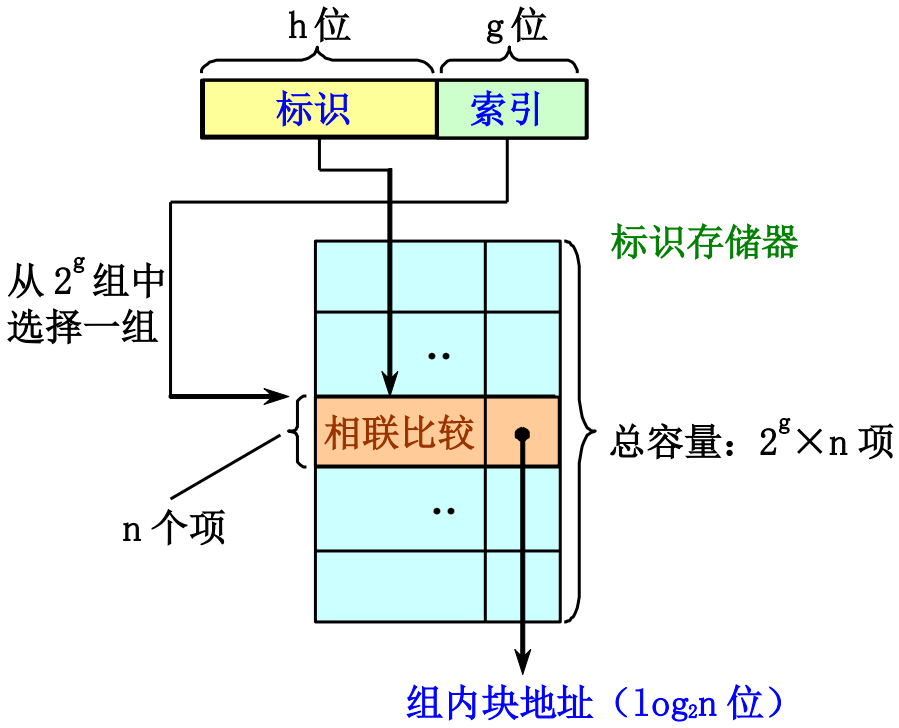
- 目录表的结构
  - 主存块的块地址的高位部分，称为**标识（Tag）**。
  - 每个主存块能唯一地由其标识来确定



- 只需查找**候选位置**所对应的目录表项

2. 并行查找的实现方法

- 相联存储器
  - 目录由2^g个相联存储区构成，每个相联存储区的大小为$n\times(h+\log_2 n)$位。
  - 根据所查找到的组内块地址，从Cache存储体中读出的多个信息字中选一个，发送给CPU。



7.2.5 替换算法

1. 替换算法

- 所要解决的问题：当新调入一块，而Cache又已被占满时，替换哪一块？
  - **直接映象Cache中的替换很简单**
    因为只有一个块，别无选择。
  - **在组相联和全相联Cache中，则有多个块供选择。**
- 主要的替换算法有三种
  - **随机法**
    优点：实现简单
  - **先进先出法FIFO**

- 最近最少使用法LRU
  - 选择近期最少被访问的块作为被替换的块。

    （实现比较困难）
  - 实际上：选择最久没有被访问过的块作为被替换的块。
  - 优点：命中率较高
- LRU和随机法分别因其不命中率低和实现简单而被广泛采用。
  - 模拟数据表明，对于容量很大的Cache，LRU和随机法的命中率差别不大。

7.2.6 写策略

1. “写”在所有访存操作中所占的比例

- 统计结果表明，对于一组给定的程序：
  - load指令：26%
  - store指令：9%
- “写”在所有访存操作中所占的比例：

  9%/(100%＋26%＋9%)≈7%
- “写”在访问数据Cache操作中所占的比例：

  9%/(26%＋9%)≈25%

2. “写”操作必须在确认是命中后才可进行

3. **“写”访问有可能导致Cache和主存内容的不一致**

4. 两种写策略 ——→ 何时更新?

写策略是区分不同Cache设计方案的一个重要标志。

- 写直达法（也称为存直达法）
  - 执行“写”操作时，不仅写入Cache，而且也写入下一级存储器。
- 写回法（也称为拷回法）
  - 执行“写”操作时，只写入Cache。仅当Cache中相应的块被替换时，才写回主存。（设置“修改位”）

5. 两种写策略的比较

- 写回法的优点：速度快，所需的下一级存储器带宽较低。
- 写直达法的优点：易于实现，一致性好。

6. 采用写直达法时，若在进行“写”操作的过程中CPU必须等待，直到“写”操作结束，则称CPU写停顿。

- 减少写停顿的一种常用的优化技术：

采用写缓冲器

7. “写”操作时的调块

- 按写分配(写时取)

  写不命中时，先把所写单元所在的块调入Cache，再行写入。

- 不按写分配(绕写法)

  写不命中时，直接写入下一级存储器而不调块。

8. 写策略与调块

- 写回法 —— 按写分配
- 写直达法 —— 不按写分配

7.2.7 Cache的性能分析

1. 不命中率
   - 与硬件速度无关
   - 容易产生一些误导
2. 平均访存时间

平均访存时间 = 命中时间+不命中率×不命中开销

3. 程序执行时间

CPU时间＝（CPU执行周期数+存储器停顿周期数）× 时钟周期时间

其中：

- 存储器停顿时钟周期数＝“读”的次数×读不命中率×读不命中开销＋“写”的次数×写不命中率×写不命中开销
- 存储器停顿时钟周期数＝访存次数×不命中率×不命中开销

CPU时间＝（CPU执行周期数+访存次数×不命中率×不命中开销）

× 时钟周期时间

$$CPU\text{时间} = IC \times \left(CPI_{execution} + \frac{\text{访存次数}}{\text{指令数}} \times \text{不命中率} \times \text{不命中开销} \right) \times \text{时钟周期时间}$$

$$= IC \times （CPI\text{execution} + \text{每条指令的平均访存次数} \times \text{不命中率} \times \text{不命中开销}） \times \text{时钟周期时间}$$

例7.1 用一个和Alpha AXP类似的机器作为第一个例子。假设Cache失效开销为50个时钟周期，当不考虑存储器停顿时，所有指令的执行时间都是2.0个时钟周期，访问Cache失效率为2%，平均每条指令访存1.33次。试分析Cache对性能的影响。

解：

$$CPU\text{时间} = IC \times \left(CPI_{exe} + \frac{\text{存储器停顿周期数}}{\text{指令数}}\right) \times \text{时钟周期时间}$$

考虑Cache的失效后，性能为：

$CPU时间_{有cache}$＝IC×（2.0＋1.33×2 %×50）×时钟周期时间

＝IC×3.33×时钟周期时间

实际*CPI* ： 3.33

3.33/2.0 = 1.67(倍)

CPU时间也增加为原来的1.67倍。

但若不采用Cache,则：

CPI＝2.0＋50×1.33＝68.5

4. Cache失效对于一个CPI较小而时钟频率较高的CPU来说，影响是双重的：

- $CPI_{execution}$越低，固定周期数的Cache失效开销的相对影响就越大。
- 在计算CPI时，失效开销的单位是时钟周期数。因此，即使两台计算机的存储层次完全相同，时钟频率较高的CPU的失效开销较大，其CPI中存储器停顿这部分也就较大。

因此Cache对于低CPI、高时钟频率的CPU来说更加重要。

例7.2 考虑两种不同组织结构的Cache：直接映象Cache和2路组相联Cache，假设：

(1)理想Cache（命中率为100%）情况下的CPI为2.0，时钟周期为2ns，平均每条指令访存1.3次。

(2)两种Cache容量均为64KB，块大小都是32字节。

(3)组相联Cache中，由于多路选择器的存在而使CPU的时钟周期增加到原来的1.10倍。

(4) 这两种结构Cache的失效开销都是70 ns。（在实际应用中，应取整为整数个时钟周期）

(5) 命中时间为1个时钟周期，64 KB直接映象Cache的失效率为1.4%，相同容量的2路组相联Cache的失效率为1.0%。

试问它们对CPU的性能有何影响？先求平均访存时间，然后再计算CPU性能。

解：　平均访存时间为：

平均访存时间＝命中时间＋失效率×失效开销

因此，两种结构的平均访存时间分别是：

平均访存时间$_{1路}$＝2.0＋（0.014×70）＝2.98 ns

平均访存时间$_{2路}$＝2.0×1.10＋（0.010×70）＝2.90 ns

2路组相联Cache的平均访存时间比较低。

CPU 时间＝*IC*×(*CPI*exe＋每条指令的平均存储器停顿周期数)×时钟周期时间

＝*IC* ×(*CPI*exe×时钟周期时间＋每条指令的平均存储器停顿时间)

因此：

$$\text{CPU时间}_{1路} = IC\times(2.0\times2+(1.3\times0.014\times70)) = 5.27\times IC$$

$$\text{CPU时间}_{2路} = IC\times(2.0\times2\times1.10+(1.3\times0.010\times70)) = 5.31\times IC$$

$$\frac{\text{CPU时间}_{2路}}{\text{CPU时间}_{1路}} = \frac{5.31\times IC}{5.27\times IC} = 1.01$$

直接映象Cache的平均性能好一些。

改进Cache的性能

1. 平均访存时间＝命中时间＋不命中率×不命中开销
2. 可以从三个方面改进Cache的性能：
   - 降低不命中率
   - 减少不命中开销
   - 减少Cache命中时间

7.3 降低Cache不命中率

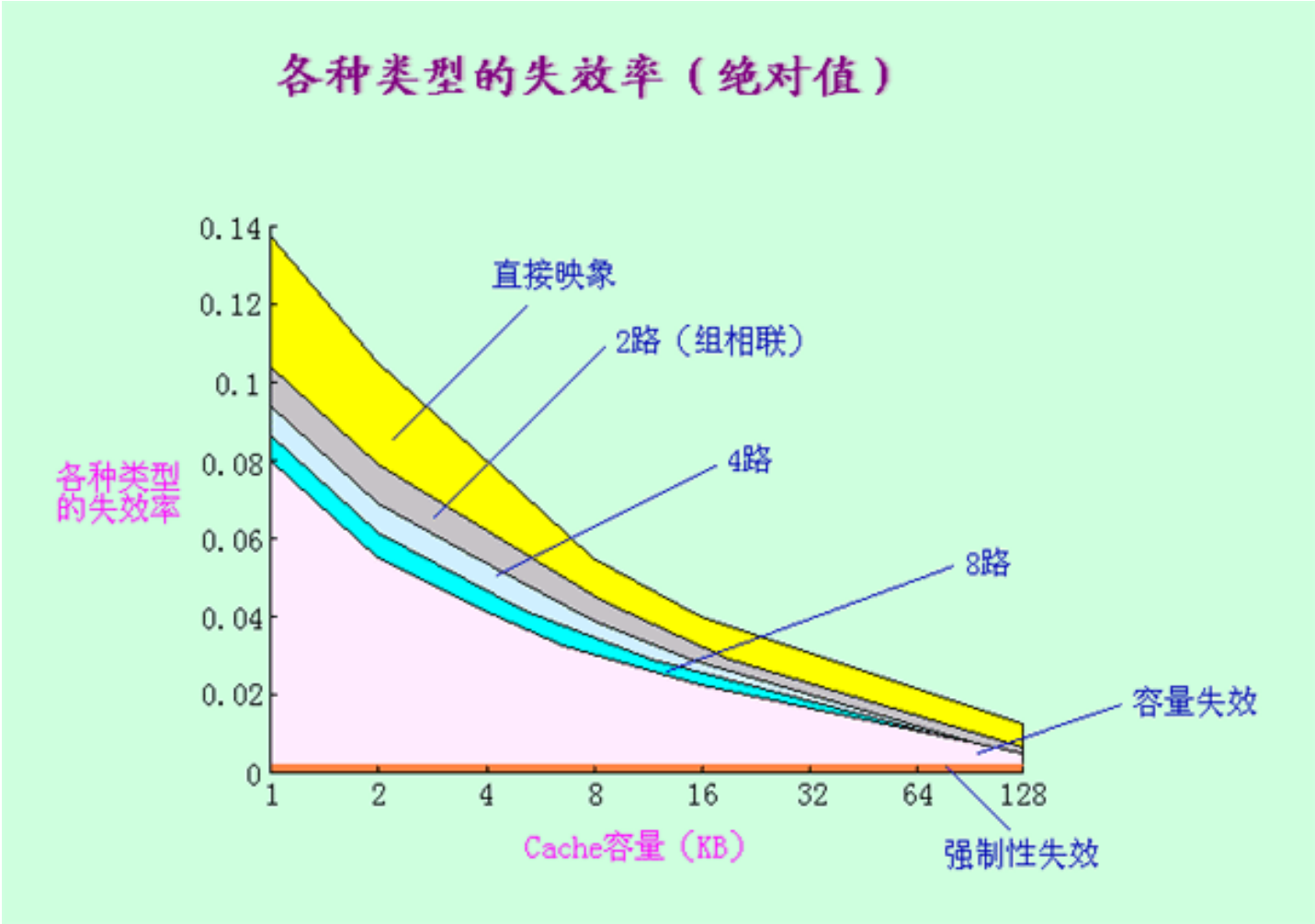
7.3.1 三种类型的不命中

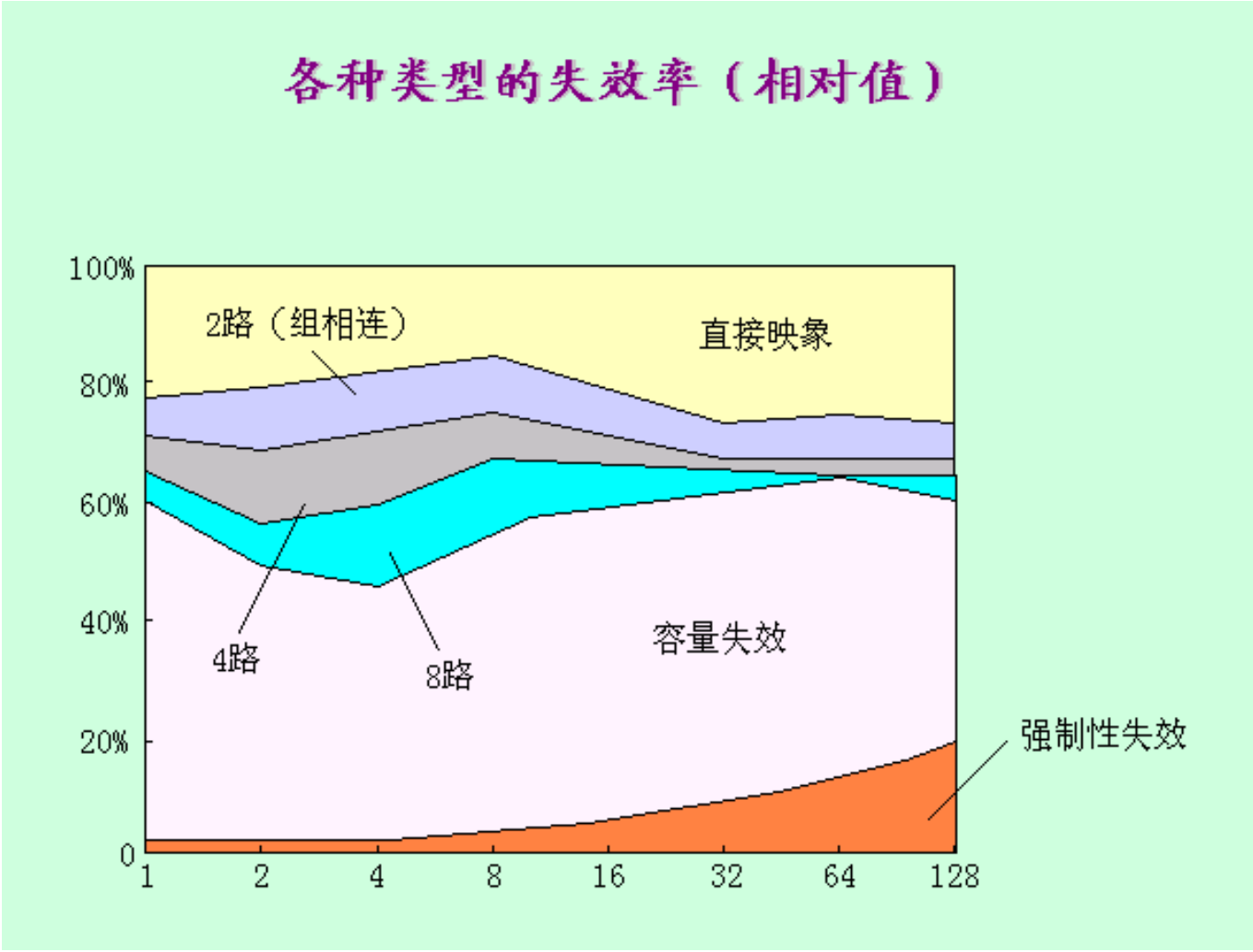
1. 三种类型的不命中(3C)

- 强制性不命中(Compulsory miss)
  - 当第一次访问一个块时，该块不在Cache中，需从下一级存储器中调入Cache，这就是强制性不命中。(冷启动不命中，首次访问不命中)
- 容量不命中(Capacity miss)
  - 如果程序执行时所需的块不能全部调入Cache中，则当某些块被替换后，若又重新被访问，就会发生不命中。这种不命中称为容量不命中。

- 冲突不命中(Conflict miss)
  - 在组相联或直接映象Cache中，若太多的块映象到同一组(块)中，则会出现该组中某个块被别的块替换(即使别的组或块有空闲位置)，然后又被重新访问的情况。这就是发生了冲突不命中。

    (碰撞不命中，干扰不命中)


各种类型的失效率（绝对值）
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
各种类型的失效率
直接映象
2路（组相联）
4路
8路
容量失效
强制性失效
1
2
4
8
16
32
64
128
Cache容量（KB）


各种类型的失效率（相对值）
100%
80%
60%
40%
20%
0
1
2
4
8
16
32
64
128
2路（组相连）
直接映象
4路
8路
容量失效
强制性失效

- 可以看出：
  - 相联度越高，冲突失效就越少；
  - 强制性失效和容量失效不受相联度的影响；
  - 强制性失效不受Cache容量的影响，但容量失效却随着容量的增加而减少；
  - 表中的数据符合2:1的Cache经验规则，即大小为N的直接映象Cache的失效率约等于大小为N/2的2路组相联Cache的失效率。

- 减少三种失效的方法
  - 强制性失效：增加块大小，预取

    （本身很少）
  - 容量失效：增加容量

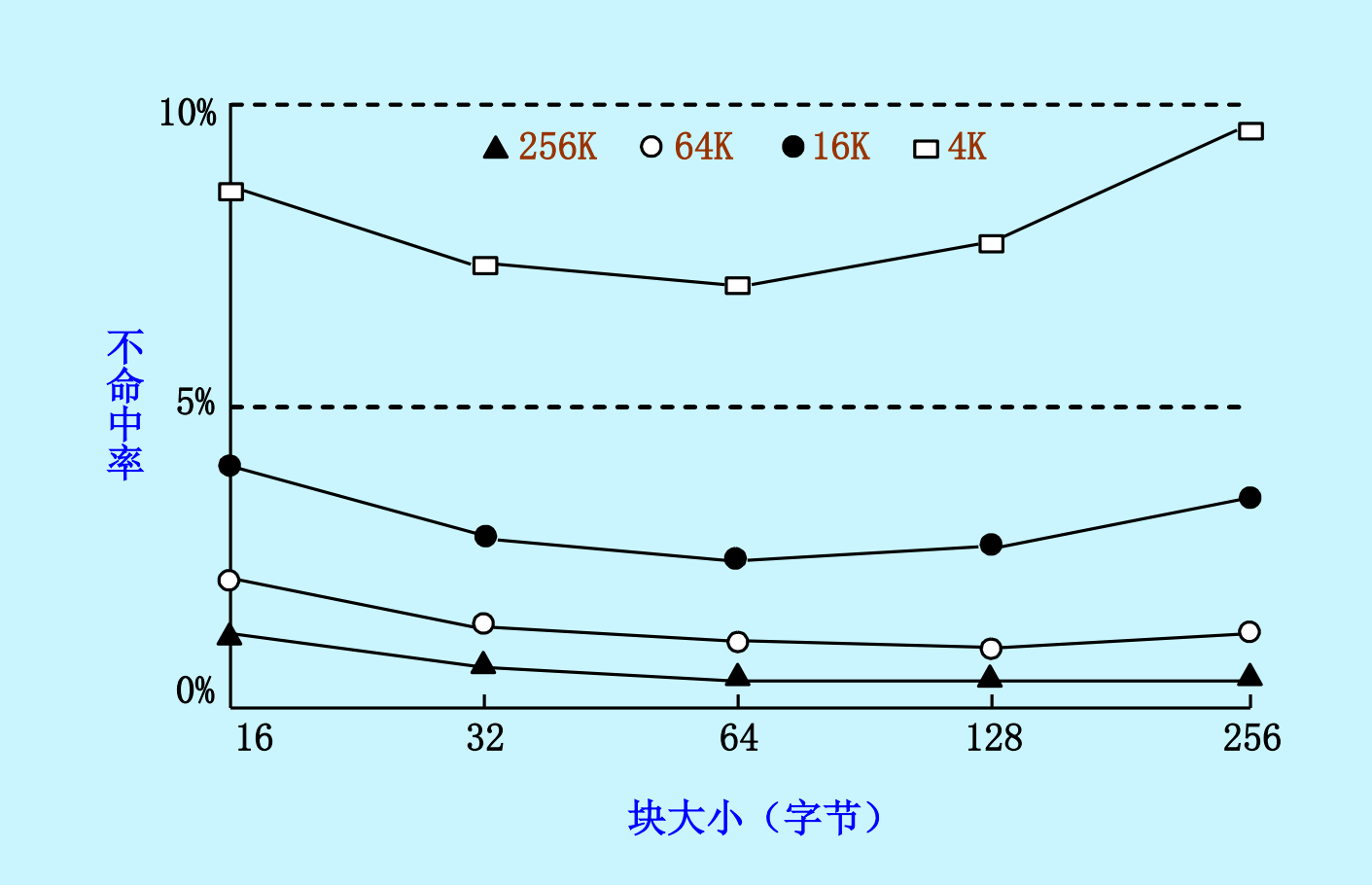
    （抖动现象）
  - 冲突失效：提高相联度

    （理想情况：全相联）
- 许多降低失效率的方法会增加命中时间或失效开销

7.3.2 增加Cache块大小

1. 不命中率与块大小的关系
   - 对于给定的Cache容量，当块大小增加时，不命中率开始是下降，后来反而上升了。

     原因：
     - 一方面它减少了强制性不命中；
     - 另一方面，由于增加块大小会减少Cache中块的数目，所以有可能会增加冲突不命中。
   - Cache容量越大，使不命中率达到最低的块大小就越大。
2. 增加块大小会增加不命中开销



不命中率随块大小变化的曲线

- 各种块大小情况下Cache的不命中率

块大小（字节）	Cache容量（字节）				
	1K	4K	16K	64K	256K
16	15.05%	8.57%	3.94%	2.04%	1.09%
32	13.34%	7.24%	2.87%	1.35%	0.70%
64	13.76%	7.00%	2.64%	1.06%	0.51%
128	16.64%	7.78%	2.77%	1.02%	0.49%
256	22.01%	9.51%	3.29%	1.15%	0.49%

7.3.3 增加Cache的容量

1. 最直接的方法是增加Cache的容量
   - 缺点：
     - 增加成本
     - 可能增加命中时间
   - 这种方法在片外Cache中用得比较多

7.3.4 提高相联度

1. 采用相联度超过8的方案的实际意义不大。
2. 2:1 Cache经验规则

   容量为N的直接映象Cache的不命中率和容量为N/2的两路组相联Cache的不命中率差不多相同。
3. 提高相联度是以增加命中时间为代价。

7.3.5 伪相联 Cache (列相联 Cache)

1. 多路组相联的低不命中率和直接映象的命中速度

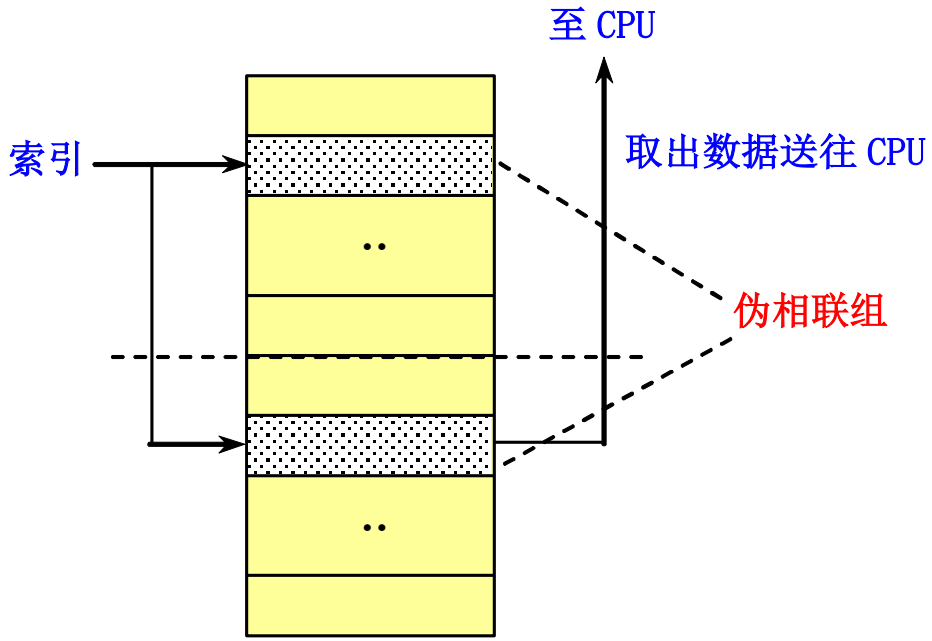
	优　点	缺　点
直接映象	命中时间小	不命中率高
组相联	不命中率低	命中时间大

2. 伪相联Cache的优点

- 命中时间小
- 不命中率低

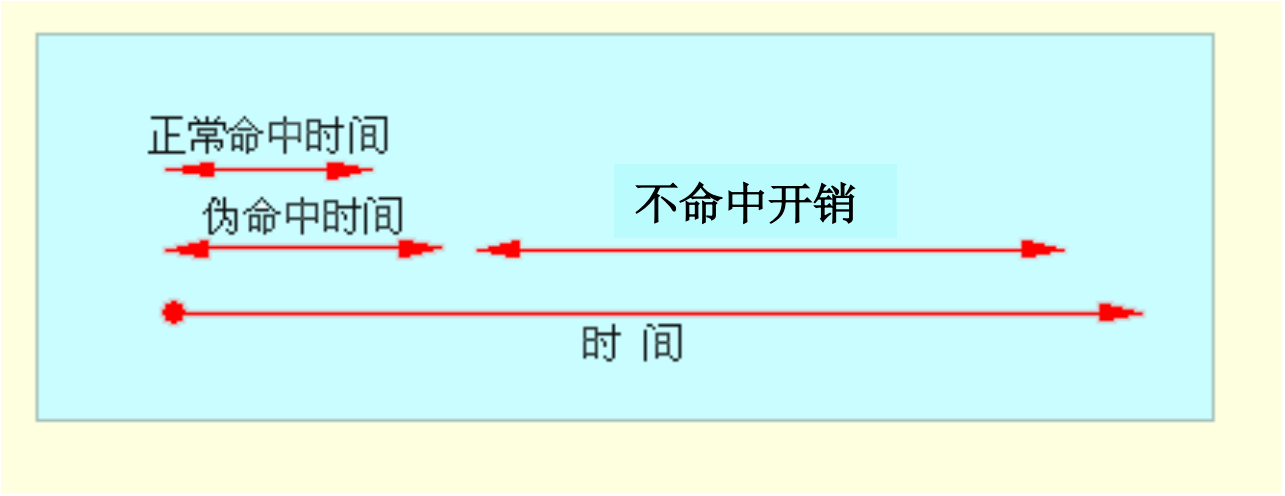
3. 基本思想及工作原理

在逻辑上把直接映象Cache的空间上下平分为两个区。对于任何一次访问，伪相联Cache先按直接映象Cache的方式去处理。若命中，则其访问过程与直接映象Cache的情况一样。若不命中，则再到另一区相应的位置去查找。若找到，则发生了伪命中，否则就只好访问下一级存储器。



4. 快速命中与慢速命中

要保证绝大多数命中都是快速命中。how?



5. 缺点：

多种命中时间→ CPU 流水线复杂化

7.3.6 硬件预取

1. 指令和数据都可以预取
2. 预取内容既可放入Cache，也可放在外缓冲器中。

   例如：指令流缓冲器
3. 指令预取通常由Cache之外的硬件完成

The Intel Pentium 4 can prefetch data into the second-level cache from up to eight streams from eight different 4 KB pages.

预取应利用存储器的空闲带宽，不能影响对正常不命中的处理，否则可能会降低性能。

7.4 减少Cache不命中开销

7.4.1 采用两级Cache

1. 应把Cache做得更快？还是更大？

答案：二者兼顾，再增加一级Cache

- 第一级Cache(L1)小而快
- 第二级Cache(L2)容量大

2. 性能分析

平均访存时间 ＝ 命中时间$_{L1}$＋不命中率$_{L1}$×不命中开销$_{L1}$

不命中开销$_{L1}$ ＝ 命中时间$_{L2}$＋不命中率$_{L2}$×不命中开销$_{L2}$

平均访存时间 ＝ 命中时间$_{L1}$＋不命中率$_{L1}$×

（命中时间$_{L2}$＋不命中率$_{L2}$×不命中开销$_{L2}$）

3. 局部不命中率与全局不命中率

- 局部不命中率＝该级Cache的不命中次数/到达该级Cache的访问次数

  例如：上述式子中的不命中率L2

- 全局不命中率＝该级Cache的不命中次数/CPU发出的访存的总次数

- 全局不命中率$_{L2}$＝不命中率$_{L1}$×不命中率$_{L2}$

  评价第二级Cache时，应使用全局不命中率这个指标。它指出了在CPU发出的访存中，究竟有多大比例是穿过各级Cache，最终到达存储器的。

4. 采用两级Cache时，每条指令的平均访存停顿时间：

每条指令的平均访存停顿时间

＝每条指令的平均不命中次数$_{L1}$×命中时间$_{L2}$＋

每条指令的平均不命中次数$_{L2}$×不命中开销$_{L2}$

5. 第二级Cache的参数

- 容量

  第二级Cache的容量一般比第一级的大许多。

  大容量意味着第二级Cache可能消除容量不命中，只剩下一些强制性不命中和冲突不命中。

- 相联度

  第二级Cache可采用较高的相联度或伪相联方法。

- 块大小
  - 第二级Cache可采用较大的块

    如 64、128、256字节
  - 为减少平均访存时间，可以让容量较小的第一级Cache采用较小的块，而让容量较大的第二级Cache采用较大的块。
  - 多级包容性

    需要考虑的另一个问题：

    第一级Cache中的数据是否总是同时存在于第二级Cache中。

7.4.2 让读不命中优先于写

1. Cache中的写缓冲器导致对存储器访问的复杂化

在读不命中时，所读单元的最新值有可能还在写缓冲器中，尚未写入主存。

2. 解决问题的方法(读不命中的处理)

- 推迟对读不命中的处理，直至写缓冲器清空

（缺点：读不命中的开销增加）

- 检查写缓冲器中的内容

3. 在写回法Cache中，也可采用写缓冲器(读不命中的处理)。

7.4.3 写缓冲合并

写直达Cache

依靠写缓冲来减少对下一级存储器写操作的时间。

- 如果写缓冲器为空，就把数据和相应地址写入该缓冲器。

  从CPU的角度来看，该写操作就算是完成了。

- 把这次的写入地址与写缓冲器中已有的所有地址进行比较，看是否有匹配的项。如果有地址匹配而对应的位置又是空闲的，就把这次要写入的数据与该项合并。这就叫写缓冲合并。

写地址	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

（a）不采用写合并

写地址	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

（b）采用了写合并

7.4.4 请求字处理技术

1. 请求字

从下一级存储器调入Cache的块中，只有一个字是立即需要的。这个字称为请求字。

2. 应尽早把请求字发送给CPU

- 尽早重启动：调块时，从块的起始位置开始读起。一旦请求字到达，就立即发送给CPU，让CPU继续执行。
- 请求字优先：调块时，从请求字所在的位置读起。这样，第一个读出的字便是请求字。将之立即发送给CPU。

7.4.5 非阻塞Cache技术

1. 非阻塞Cache：Cache不命中时仍允许CPU进行其它的命中访问。即允许“不命中下命中”。
2. 进一步提高性能：
   - “多重不命中下命中”
   - “不命中下不命中”

（存储器必须能够处理多个不命中）

7.5 减少命中时间

命中时间直接影响到处理器的时钟频率。在当今的许多计算机中，往往是Cache的访问时间限制了处理器的时钟频率。

7.5.1 容量小、结构简单的Cache

1. 硬件越简单，速度就越快；
2. 应使Cache足够小，以便可以与CPU一起放在同一块芯片上。

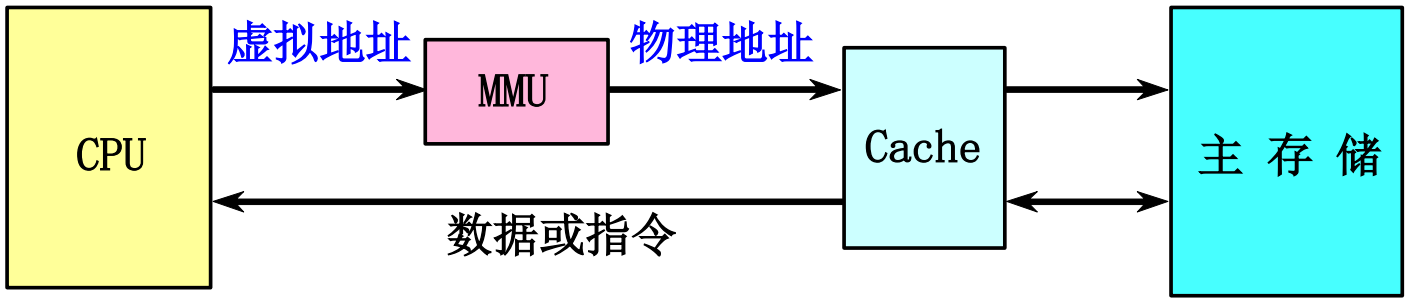
某些设计采用了一种折衷方案：

把Cache的标识放在片内，而把Cache的数据存储器放在片外。

7.5.2 虚拟Cache

1. 物理Cache

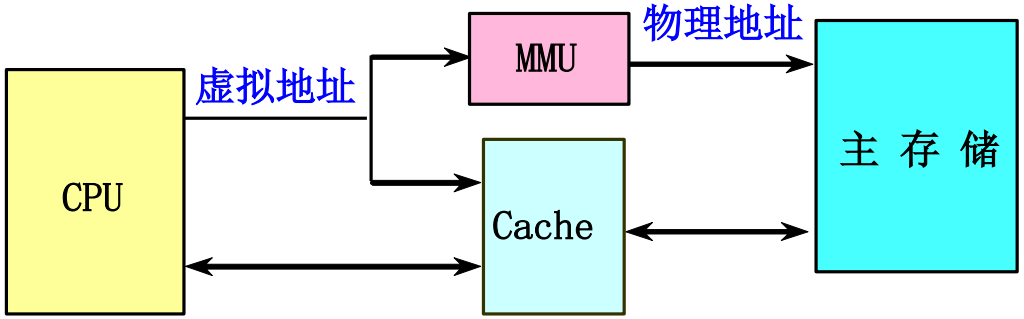
- 使用物理地址进行访问的传统Cache。
- 标识存储器中存放的是物理地址，进行地址检测也是用物理地址。
- 缺点：地址转换和访问Cache串行进行，访问速度很慢。



物理Cache存储系统

2. 虚拟Cache

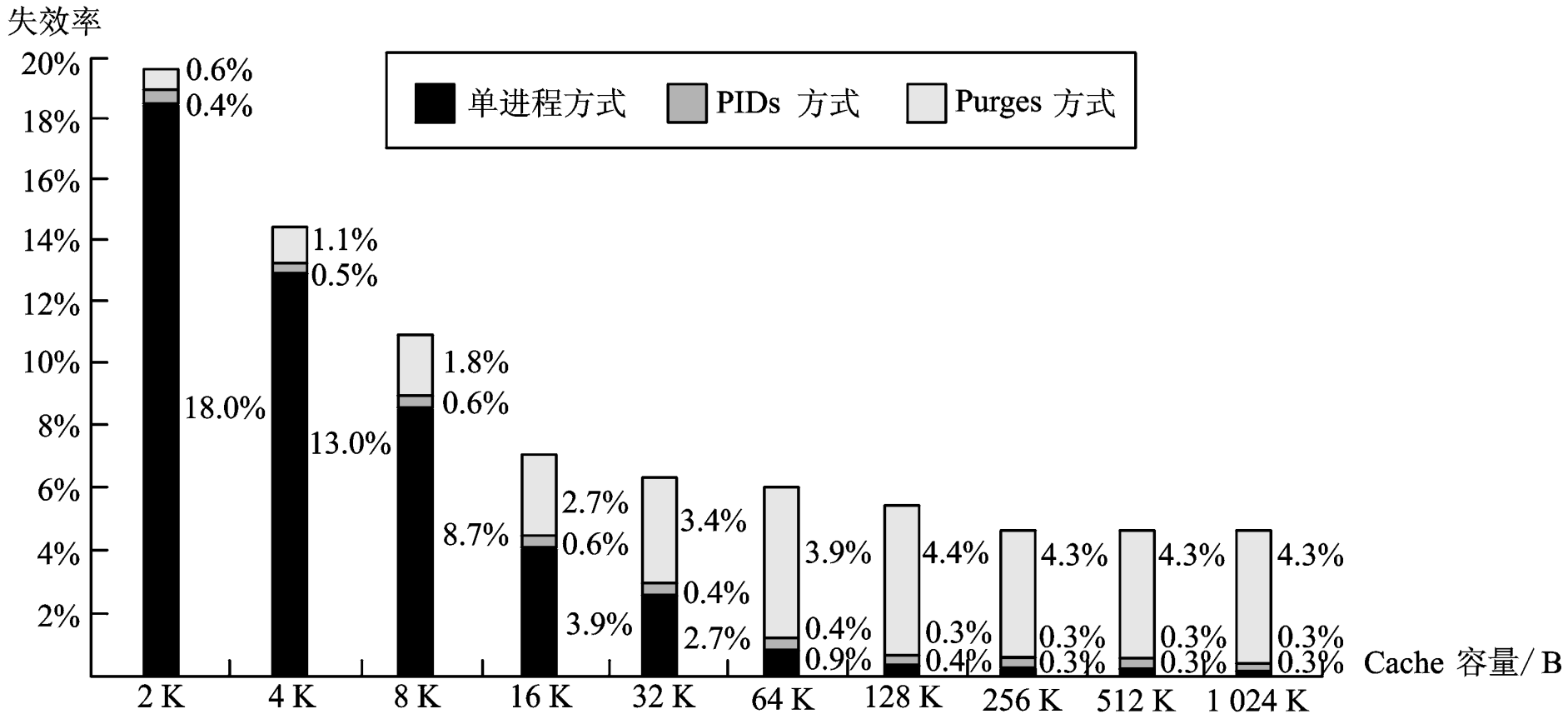
- 可以直接用虚拟地址进行访问的Cache。标识存储器中存放的是虚拟地址，进行地址检测用的也是虚拟地址。
- 优点：
  - **在命中时不需要地址转换，省去了地址转换的时间。即使不命中，地址转换和访问Cache也是并行进行的，其速度比物理Cache快很多。**



3. 并非都采用虚拟Cache(为什么?)

- 虚拟Cache的清空问题
  - 解决方法：在地址标识中增加PID字段

    (进程标识符)
  - 三种情况下不命中率的比较
    - 单进程，PIDs，清空(Purges)
    - PIDs与单进程相比：+0.3%～+0.6%
    - PIDs与清空相比：　-0.6%～-4.3%

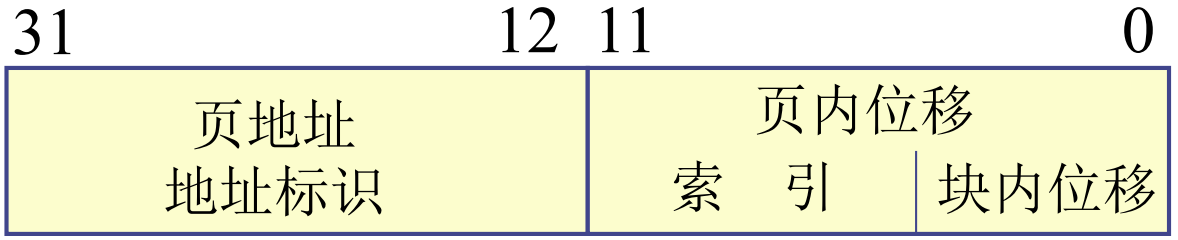

失效率
20%
18%
16%
14%
12%
10%
8%
6%
4%
2%
单进程方式
PIDs 方式
Purges 方式
0.6%
0.4%
18.0%
1.1%
0.5%
13.0%
1.8%
0.6%
8.7%
2.7%
0.6%
3.9%
3.4%
0.4%
2.7%
3.9%
0.4%
0.9%
4.4%
0.3%
0.4%
4.3%
0.3%
0.3%
4.3%
0.3%
0.3%
4.3%
0.3%
0.3%
2 K
4 K
8 K
16 K
32 K
64 K
128 K
256 K
512 K
1 024 K
Cache 容量/ B

4. 虚拟索引＋物理标识
   - 优点：兼得虚拟Cache和物理Cache的好处
   - 局限性：Cache容量受到限制

(页内位移)

Cache容量≤页大小×相联度

5. 举例：IBM3033的Cache
   - 页大小＝4KB　　相联度＝16



回顾查找算法

Cache容量＝16×4KB＝64KB

7.5.3 Cache访问流水化

1. 对第一级Cache的访问按流水方式组织
2. 访问Cache需要多个时钟周期才可以完成

   例如

   - Pentium访问指令Cache需要一个时钟周期
   - Pentium Pro到Pentium III需要两个时钟周期
   - Pentium 4 则需要4个时钟周期

7.5.4 Trace Cache

1. 开发指令级并行性所遇到的一个挑战是：

当要每个时钟周期流出超过4条指令时，要提供足够多条彼此互不相关的指令是很困难的。

2. 一个解决方法：采用Trace Cache

存放CPU所执行的动态指令序列

包含了由分支预测展开的指令，该分支预测是否正确需要在取到该指令时进行确认。

3. 优缺点

- 地址映象机制复杂。
- 相同的指令序列有可能被当作条件分支的不同选择而重复存放。
- 能够提高指令Cache的空间利用率。

7.5.5 Cache优化技术总结

- “+”号：表示改进了相应指标。
- “—”号：表示它使该指标变差。
- 空格栏：表示它对该指标无影响。
- 复杂性：0表示最容易，3表示最复杂。

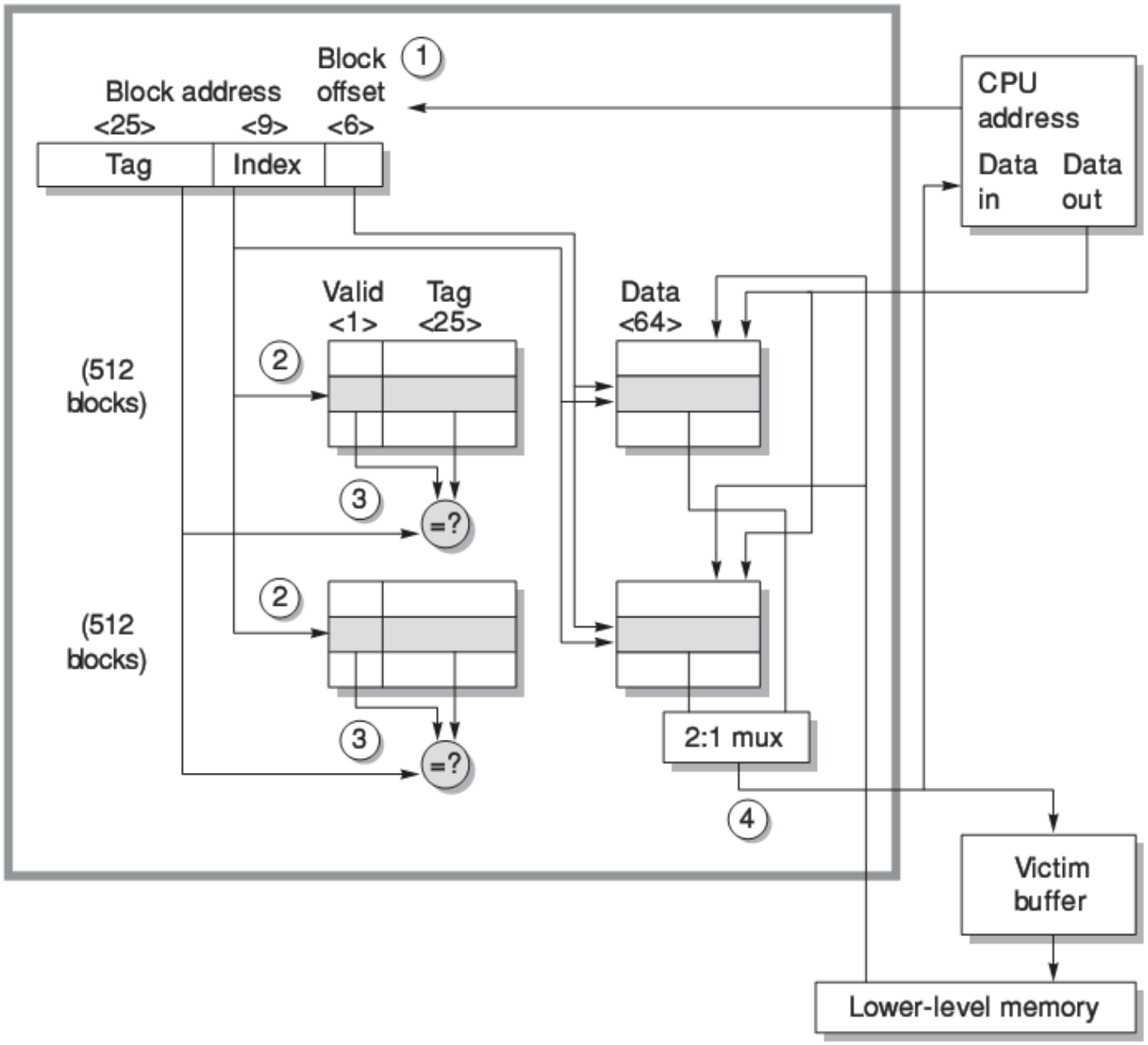
Cache优化技术总结

优化技术	不命中率	不命中开销	命中时间	硬件复杂度	说 明
增加块大小	+	–		0	实现容易；Pentium 4 的第二级Cache采用了128字节的块
增加Cache容量	+			1	被广泛采用，特别是第二级Cache
提高相联度	+		–	1	被广泛采用
牺牲Cache	+			2	AMD Athlon采用了8个项的Victim Cache
伪相联Cache	+			2	MIPS R10000的第二级Cache采用
硬件预取指令和数据	+			2～3	许多机器预取指令，UltraSPARC III预取数据

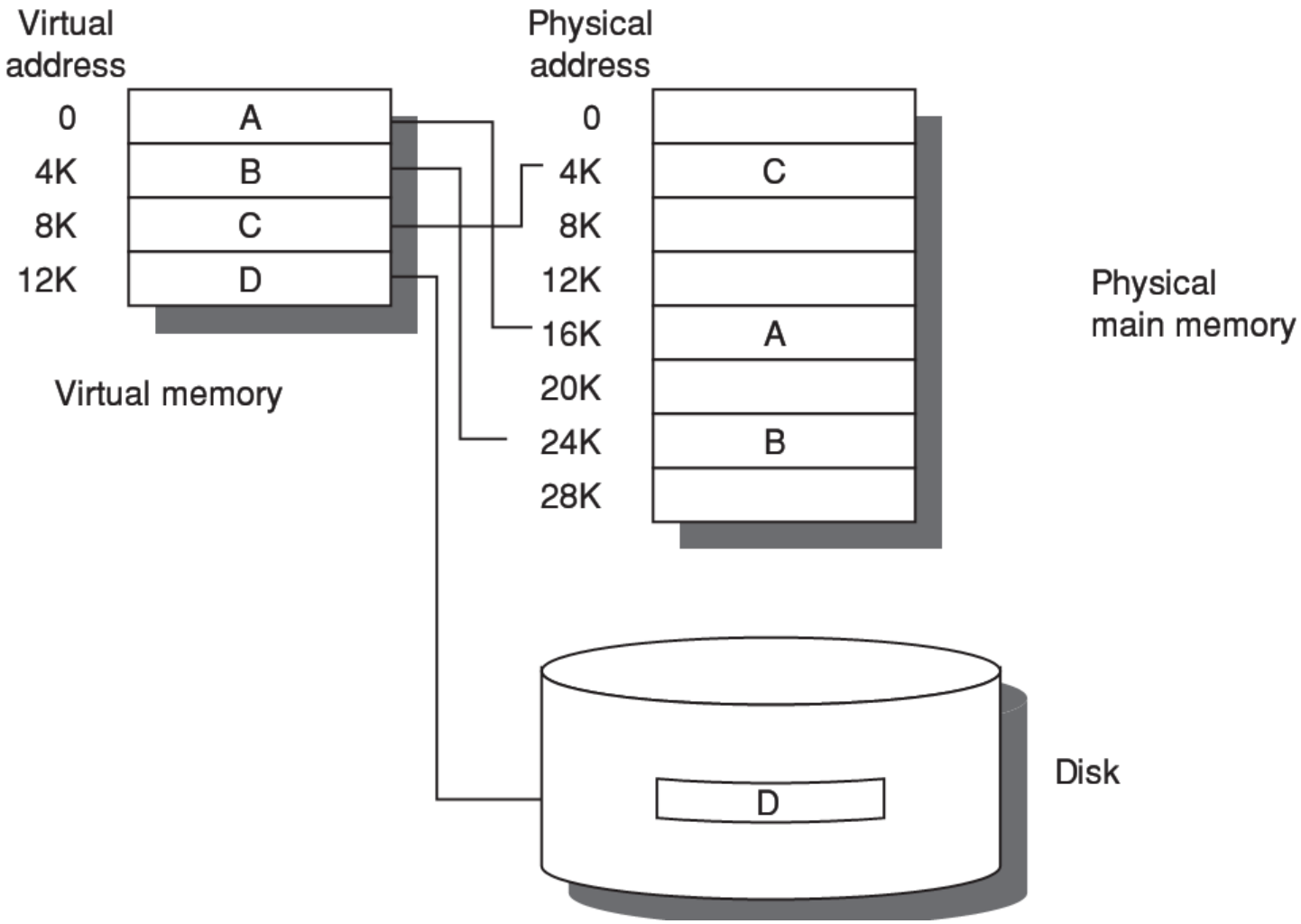
优化技术	不命中率	不命中开销	命中时间	硬件复杂度	说 明
编译器控制的预取	+			3	需同时采用非阻塞Cache；有几种微处理器提供了对这种预取的支持
用编译技术减少Cache不命中次数	+			0	向软件提出了新要求；有些机器提供了编译器选项
使读不命中优先于写		+	–	1	在单处理机上实现容易，被广泛采用
写缓冲归并		+		1	与写直达合用，广泛应用，例如21164，UltraSPARC III
尽早重启动和关键字优先		+		2	被广泛采用
非阻塞Cache		+		3	在支持乱序执行的CPU中使用

优化技术	不命中率	不命中开销	命中时间	硬件复杂度	说 明
两级Cache			+	2	硬件代价大；两级Cache的块大小不同时实现困难；被广泛采用
容量小且结构简单的Cache	–		+	0	实现容易，被广泛采用
对Cache进行索引时不必进行地址变换			+	2	对于小容量Cache来说实现容易，已被Alpha 21164和UltraSPARC III采用
流水化Cache访问			+	1	被广泛采用
Trace Cache			+	3	Pentium 4 采用

The Opteron Data Cache

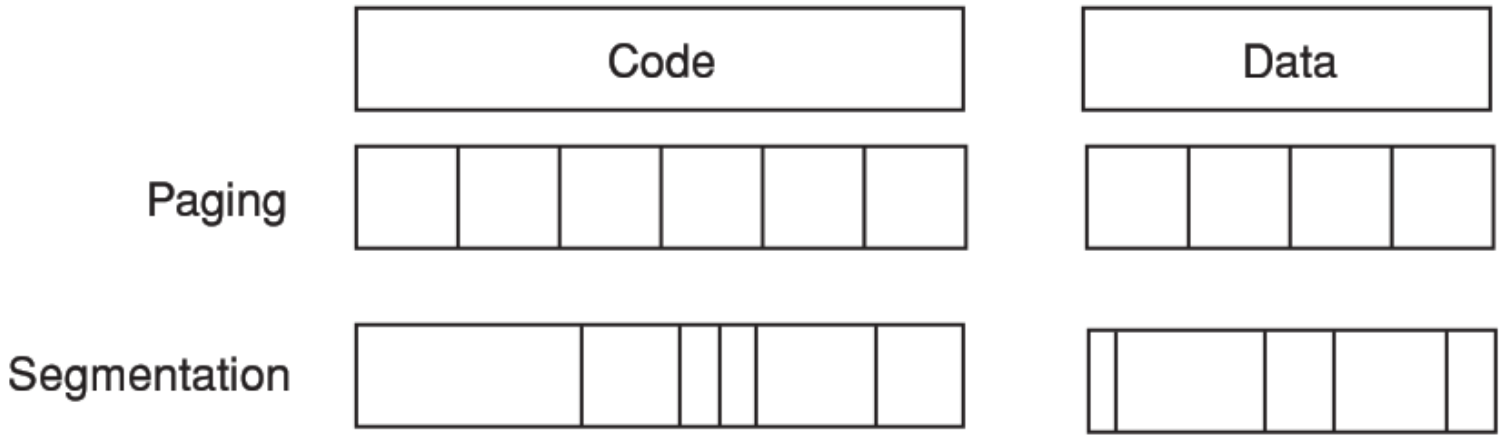


7.7 虚拟存储器


Virtual address
0
4K
8K
12K
A
B
C
D
Virtual memory
Physical address
0
4K
8K
12K
16K
20K
24K
28K
C
A
B
Physical main memory
D
Disk

7.7.1 虚拟存储器的基本原理

1. 发明虚拟存储器的目的
2. 虚拟存储器可以分为两类：页式和段式
   - 页式虚拟存储器把空间划分为大小相同的块。（页面）
   - 段式虚拟存储器则把空间划分为可变长的块。（段）
   - 页面是对空间的机械划分，而段则往往是按程序的逻辑意义进行划分。



Example of how paging and segmentation divide a program.

3. Cache和虚拟存储器的参数取值范围

参数	第一级Cache	虚拟存储器
块（页）大小	16-128字节	4096-65,536字节
命中时间	1-3个时钟周期	100-200个时钟周期
不命中开销	8-200个时钟周期	1,000,000-10,000,000个时钟周期
（访问时间）	（6-160个时钟周期）	（800,000-8,000,000个时钟周期）
（传输时间）	（2-40个时钟周期）	（200,000-2,000,000个时钟周期）
不命中率	0.1-10%	0.00001-0.001%
地址映象	25-45位物理地址到14-20位Cache地址	32-64位虚拟地址到25-45位物理地址

Four questions about any level of the hierarchy（Cache）

Q1: Where can a block be placed in the upper level?

(*Block placement*)　　三种映像规则

Q2: How is a block found if it is in the upper level?

(*Block identification*)　目录表

Q3: Which block should be replaced on a miss?

(*Block replacement*)　LRU

Q4: What happens on a write?

(*Write strategy*)　　写回/写直达

Four questions about any level of the hierarchy（VM）

Q1: Where can a block be placed in the upper level?

(*Block placement*)　　Full

Q2: How is a block found if it is in the upper level?

(*Block identification*)　Page Table

Q3: Which block should be replaced on a miss?

(*Block replacement*)　LRU（how to implement?）

Q4: What happens on a write?

(*Write strategy*)　　Write Back

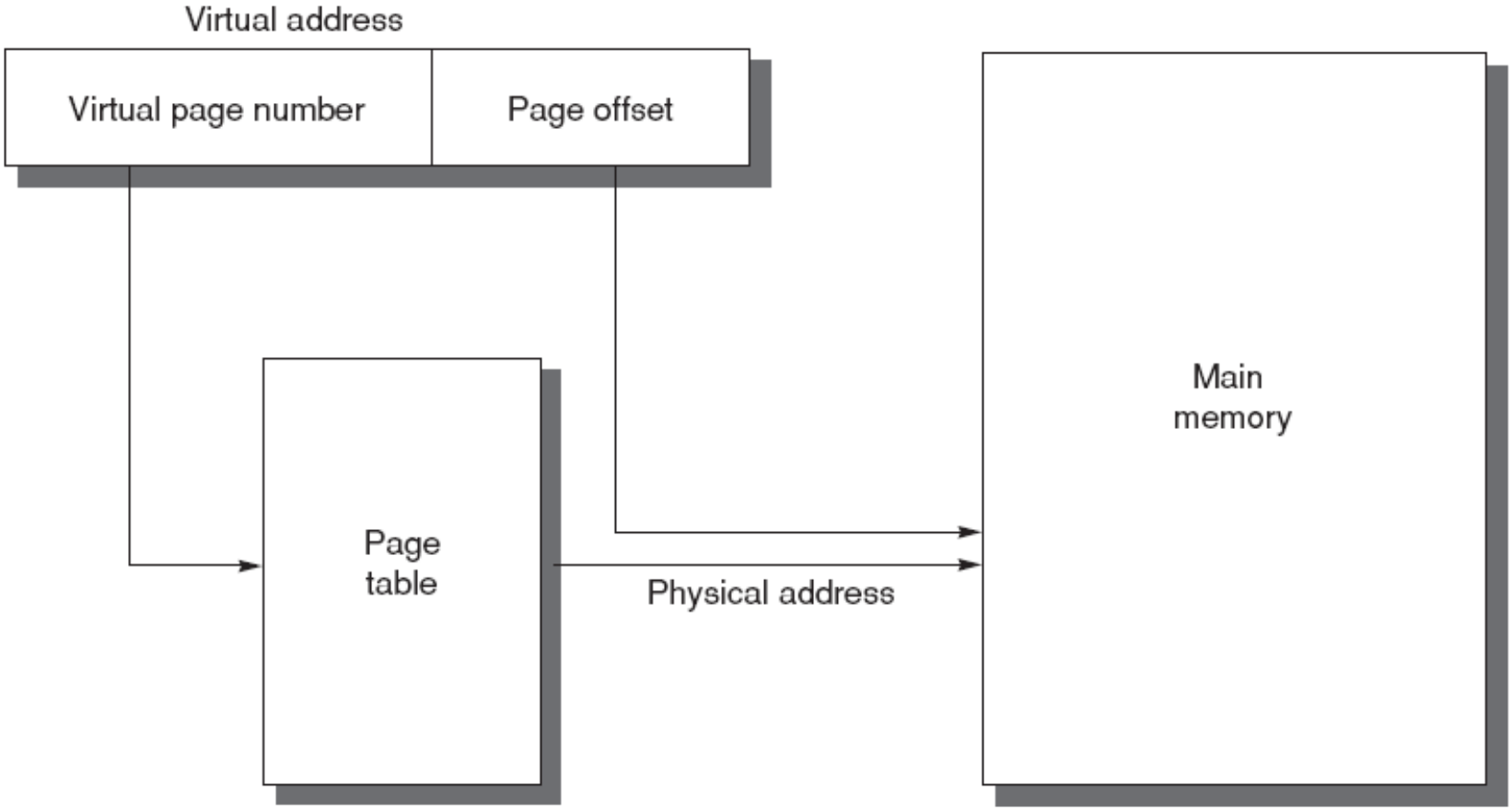


Figure C.22 The mapping of a virtual address to a physical address via a page table.

7.7.2 快速地址转换技术

1. 地址变换缓冲器TLB

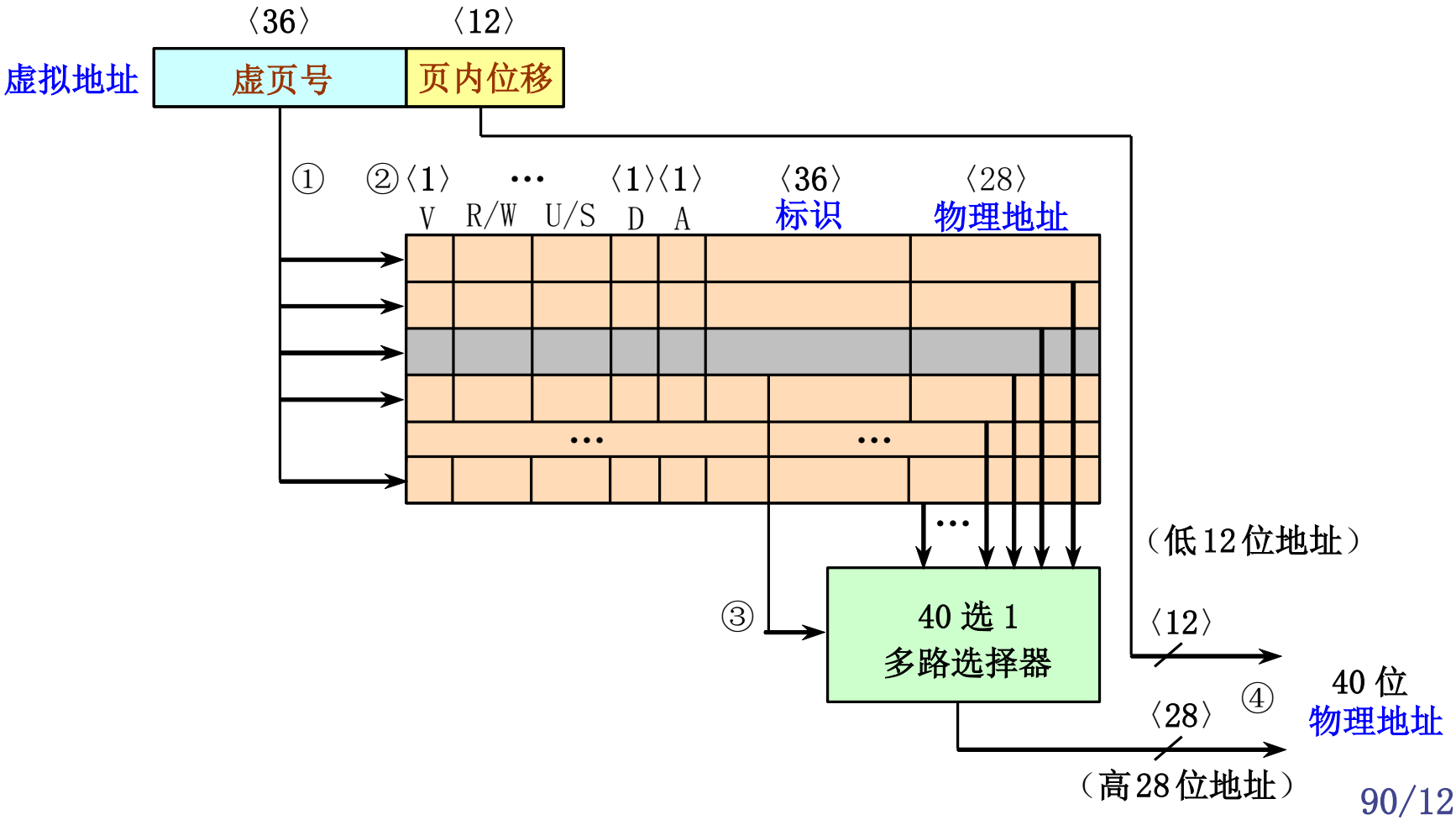
- TLB是一个专用的高速缓冲器，用于存放近期经常使用的页表项；
- TLB中的内容是页表部分内容的一个副本；
- TLB也利用了局部性原理。

2. TLB中的项由两部分构成：标识和数据

- 标识中存放的是虚地址的一部分。
- 数据部分中存放的则是物理页帧号、有效位、存储保护信息、使用位、修改位等。

3. AMD Opteron的数据TLB的组织结构

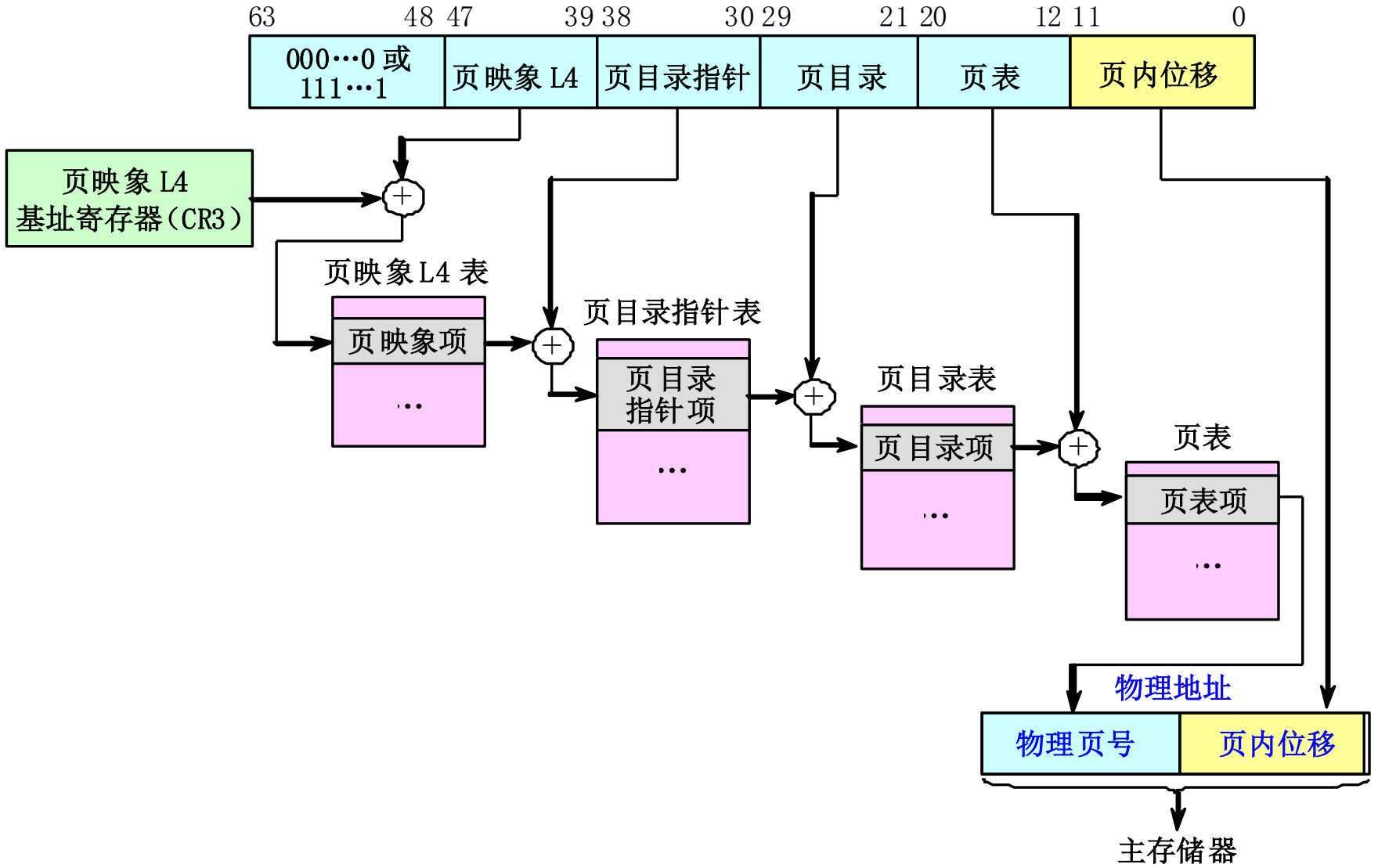
- 采用全相联映象



7.7.3 页式虚拟存储器实例：64位Opteron的存储管理

1. Opteron的页面大小：4KB，2MB和4MB。
2. AMD64系统结构

   虚拟地址： 64位　物理地址：52位
3. 采用多级分层页表结构来映射地址空间，以便使页表大小合适。
   - 分级的级数取决于虚拟地址空间的大小
   - Opteron的48位虚拟地址的4级转换
   - 每个分级页表的偏移量分别来自4个9位的字段


63
48 47
39 38
30 29
21 20
12 11
0
000…0 或
111…1
页映象 L4
页目录指针
页目录
页表
页内位移
页映象 L4
基址寄存器(CR3)
页映象 L4 表
页映象项
…
页目录指针表
页目录
指针项
…
页目录表
页目录项
…
页表
页表项
…
物理地址
物理页号
页内位移
主存储器

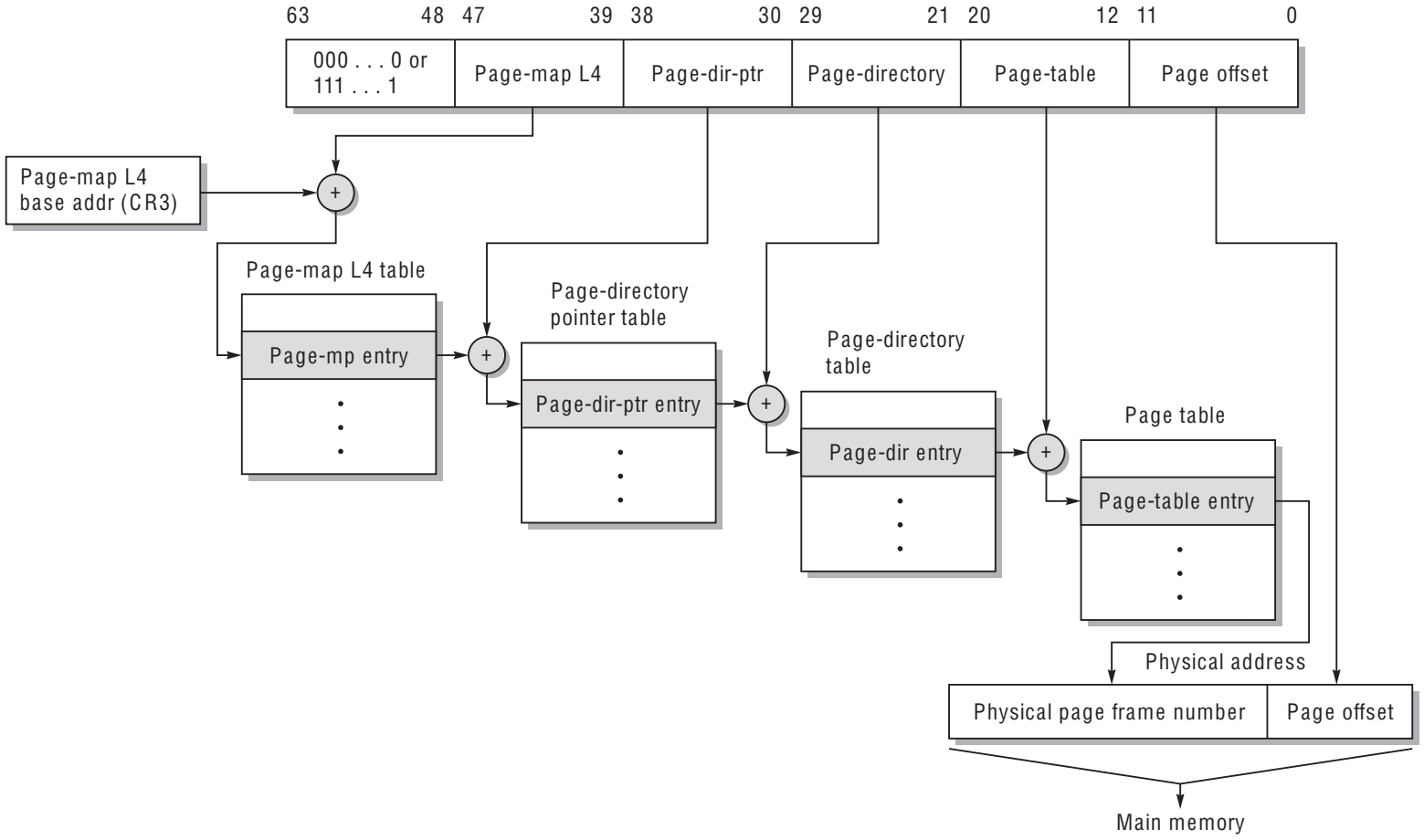
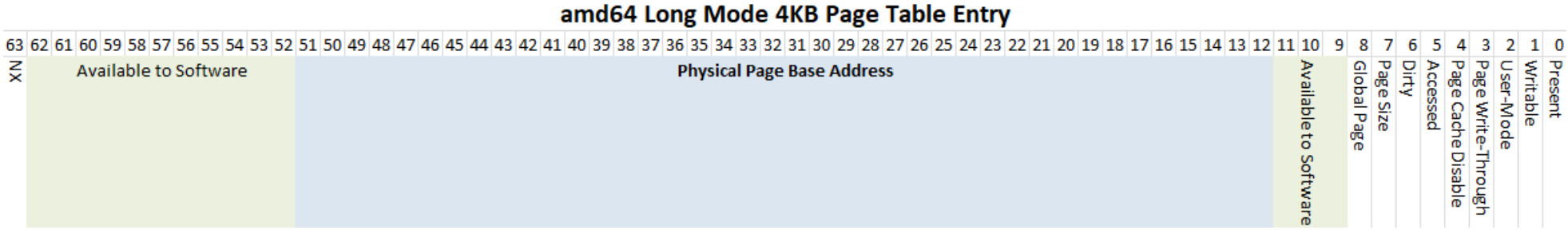


Figure C.26 The mapping of an Opteron virtual address. The Opteron virtual memory implementation with four page table levels supports an effective physical address size of 40 bits. Each page table has 512 entries, so each level field is 9 bits wide. The AMD64 architecture document allows the virtual address size to grow from the current 48 bits to 64 bits, and the physical address size to grow from the current 40 bits to 52 bits.

4. Opteron的页表采用64位的项

- 前12位留给将来使用
- 最后的12位是保护和使用信息。



- 不同级的页表中有所不同，但大都包含以下基本字段：
  - 存在位：说明该页面在存储器中。
  - 读/写位：说明该页面是只读还是可读写。
  - 用户/管理位：说明用户是否能访问此页或只能由上面的3个特权级所访问。
  - 修改位：说明该页面已被修改过。
  - 访问位：说明自上次该位被清0后到现在，该页面是否被读或写过。
  - 页面大小：说明最后一级页面是4KB还是4MB；如果是4MB，则Opteron仅使用三级页表而非四级。

- **非执行位：在有些页面中用来阻止代码的执行。**
- **页级Cache使能：说明该页面能否进入Cache。**
- **页级写直达：说明该页是允许对数据Cache进行写回还是写直达。**

5. Opteron通常在TLB不命中时要遍历所有四级页表，故有3个位置可以进行保护限制的检查。
   - 仅遵从底层的PTE，而在其他级上只需确认有效位是有效的即可。
6. 在保护方面，如何避免用户进行非法的地址转换？
   - 由于页表本身已被保护，用户程序无法对它们进行写操作。

- 操作系统通过控制页表项来控制哪些物理地址可以被访问，哪些不能访问。
- 多个进程共享存储器是通过使各自的地址空间中的一个页表项指向同一个物理页面来实现的。

7. Opteron使用4个TLB以减少地址转换时间
   - 两个用于访问指令，两个用于访问数据。
8. 和多级Cache类似，Opteron通过采用两个更大的第二级TLB来减少TLB不命中。
   - 一个用于访问指令
   - 另一个用于访问数据

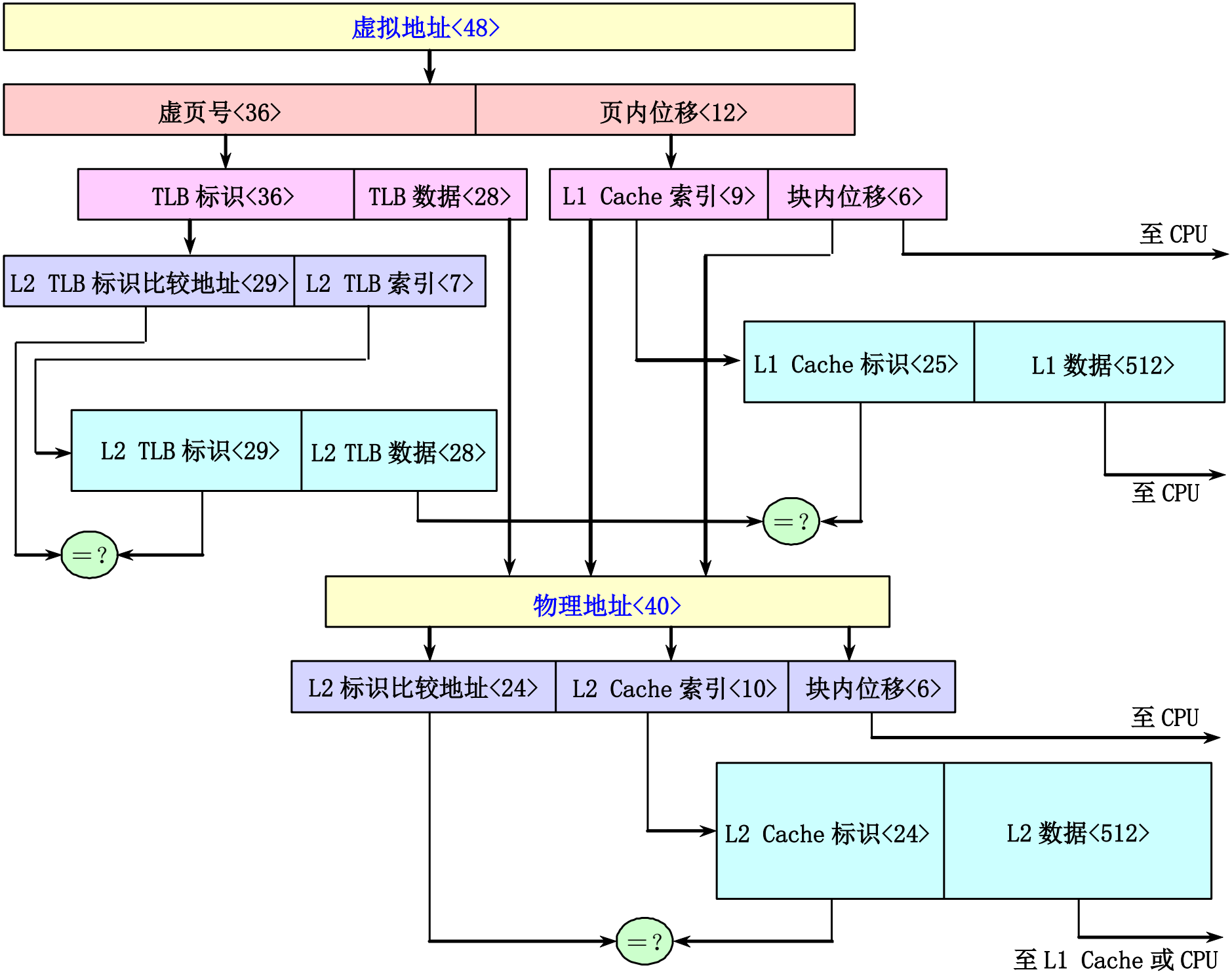
- Opteron中第一级和第二级指令、数据TLB的参数

参数	描述
块大小	1个 PTE（8字节）
L1命中时间	1个时钟周期
L2命中时间	7个时钟周期
L1 TLB大小	指令和数据TLB都是40个PTE，其中32个用于4KB大小的页面，8个用于2MB或4MB页面。
L2 TLB大小	指令和数据TLB都是512个PTE，用于4KB页面
块选择	LRU
L1映象规则	全相联
L2映象规则	4路组相联

7.8 实例：AMD Opteron的存储器层次结构

- 一个乱序执行处理器
  - **每个时钟周期最多可以取出3条80x86指令，并将之转换成类RISC操作，然后以每个时钟周期3个操作的速率流出。**
- 有11个并行的执行部件
- 在2006年，其12级整数流水线使得该处理器的最高时钟频率达到了2.8GHz。
- 虚地址：48位　　物理地址：40位

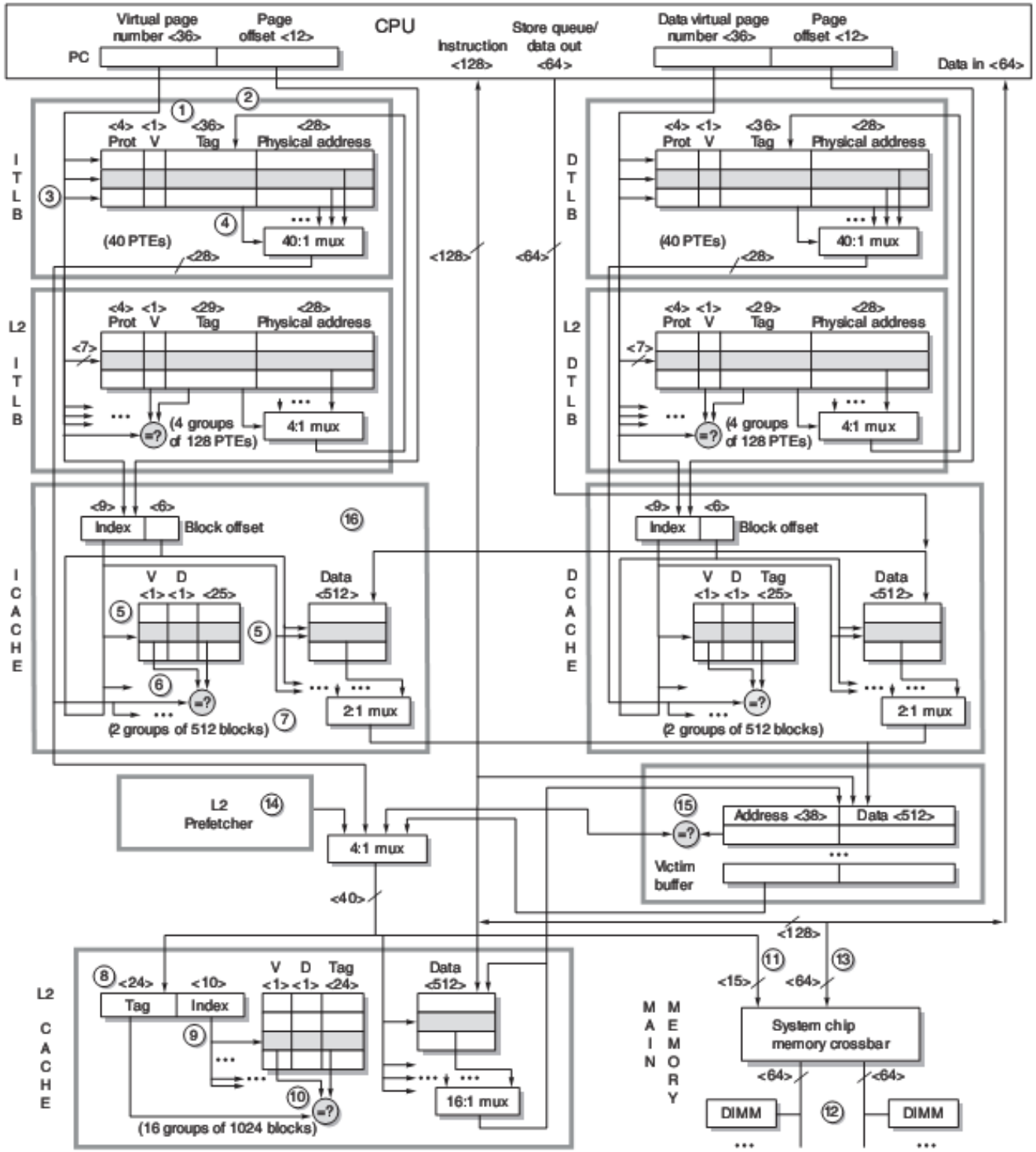
通过两级TLB实现的从虚拟地址到物理地址的转换以及对两级数据Cache的访问情况


虚拟地址<48>
虚页号<36>
页内位移<12>
TLB 标识<36>
TLB 数据<28>
L1 Cache 索引<9>
块内位移<6>
至 CPU
L2 TLB 标识比较地址<29>
L2 TLB 索引<7>
L1 Cache 标识<25>
L1 数据<512>
L2 TLB 标识<29>
L2 TLB 数据<28>
至 CPU
=?
=?
物理地址<40>
L2 标识比较地址<24>
L2 Cache 索引<10>
块内位移<6>
至 CPU
L2 Cache 标识<24>
L2 数据<512>
=?
至 L1 Cache 或 CPU

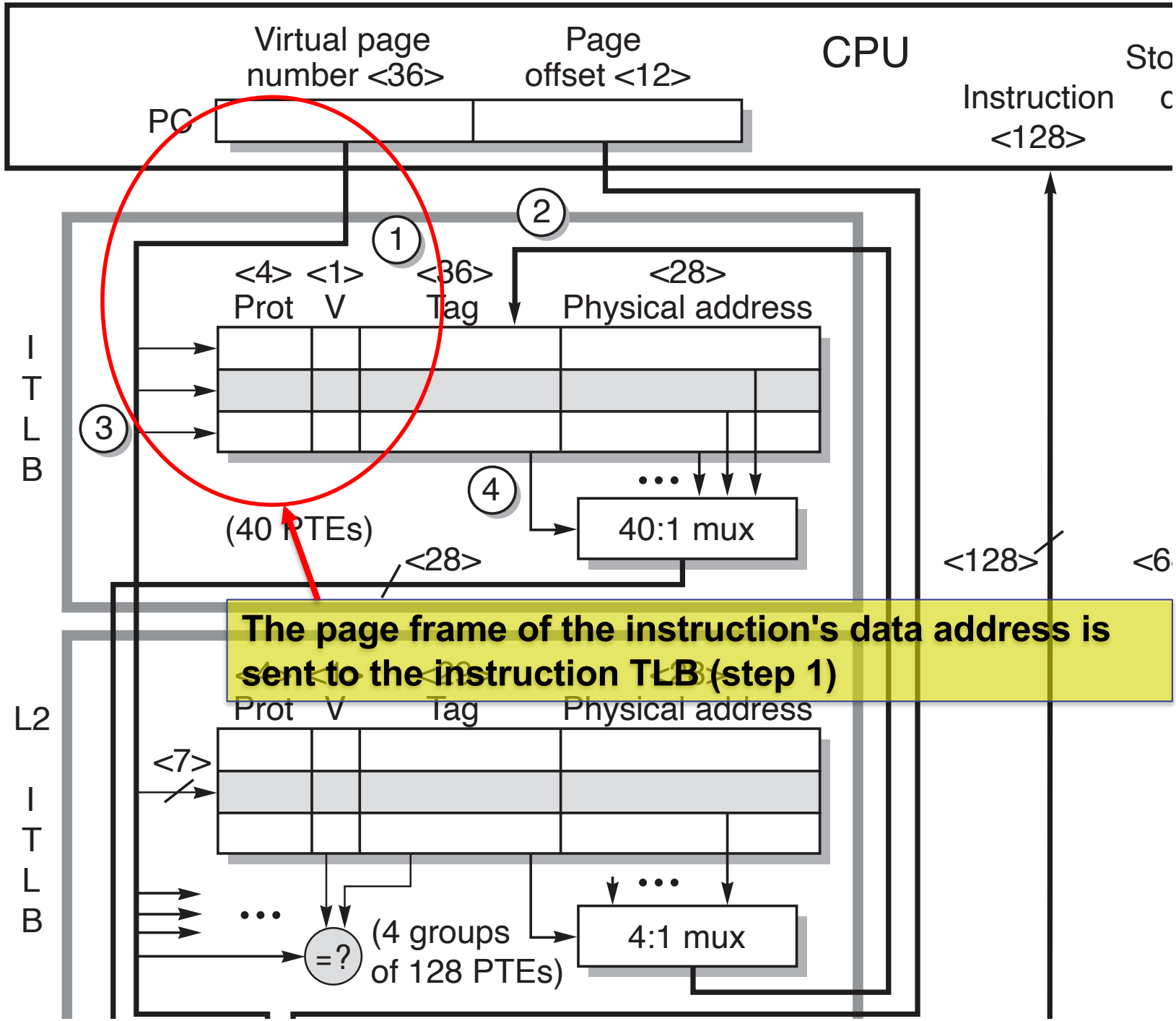
- The L1 TLB is fully associative with 40 entries.
- The L2 TLB is 4-way set associative with 512 entries.
- The L1 cache is 2-way set associative with 64-byte blocks and 64 KB capacity.
- The L2 cache is 16-way set associative with 64-byte blocks and 1 MB capacity.

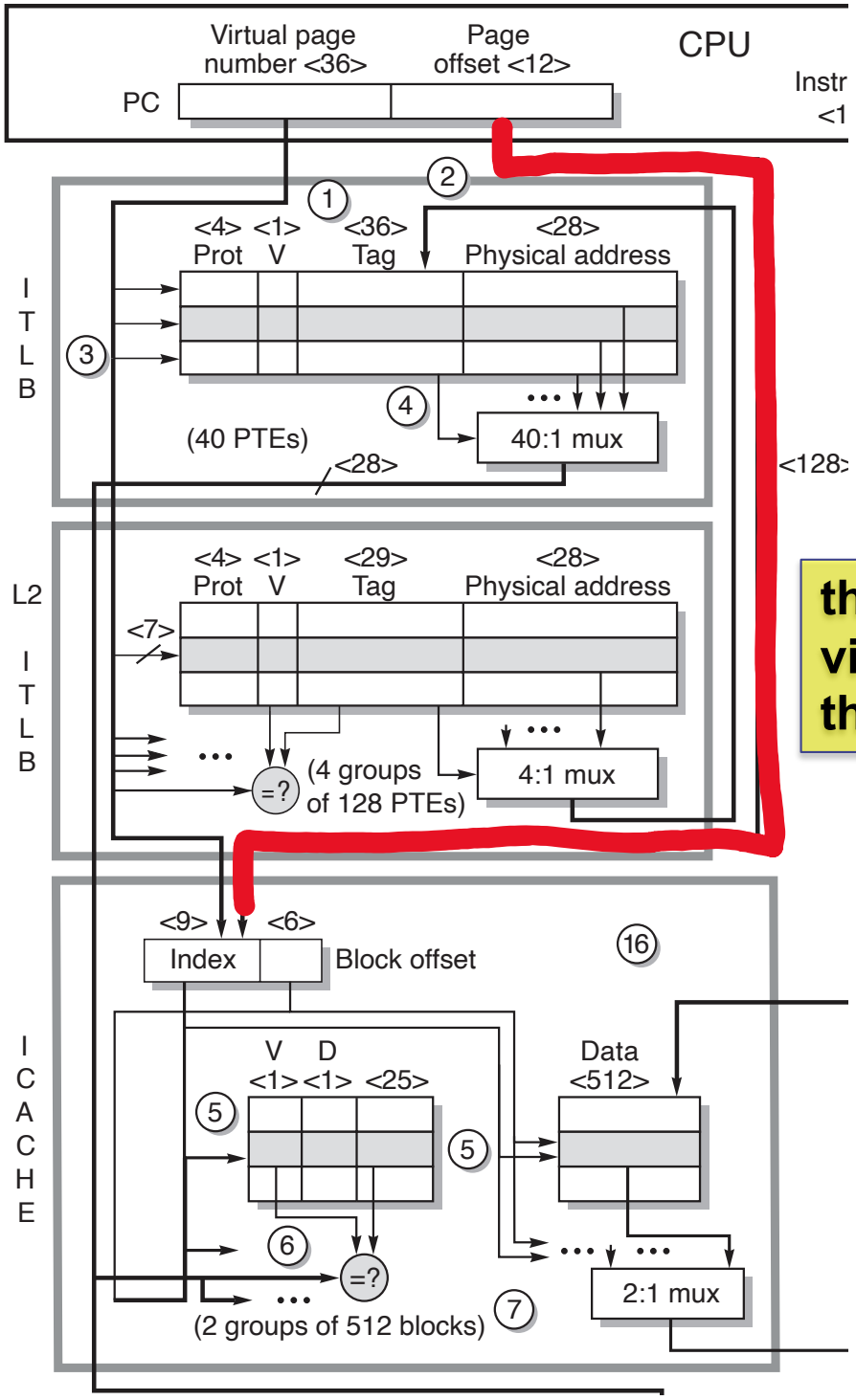
- L1指令Cache的索引位数：

$$2^{Index} = \frac{Cache容量}{\text{Cache块大小} \times \text{组相联度}} = \frac{64\text{KB}}{64\text{B} \times 2} = 2^9$$

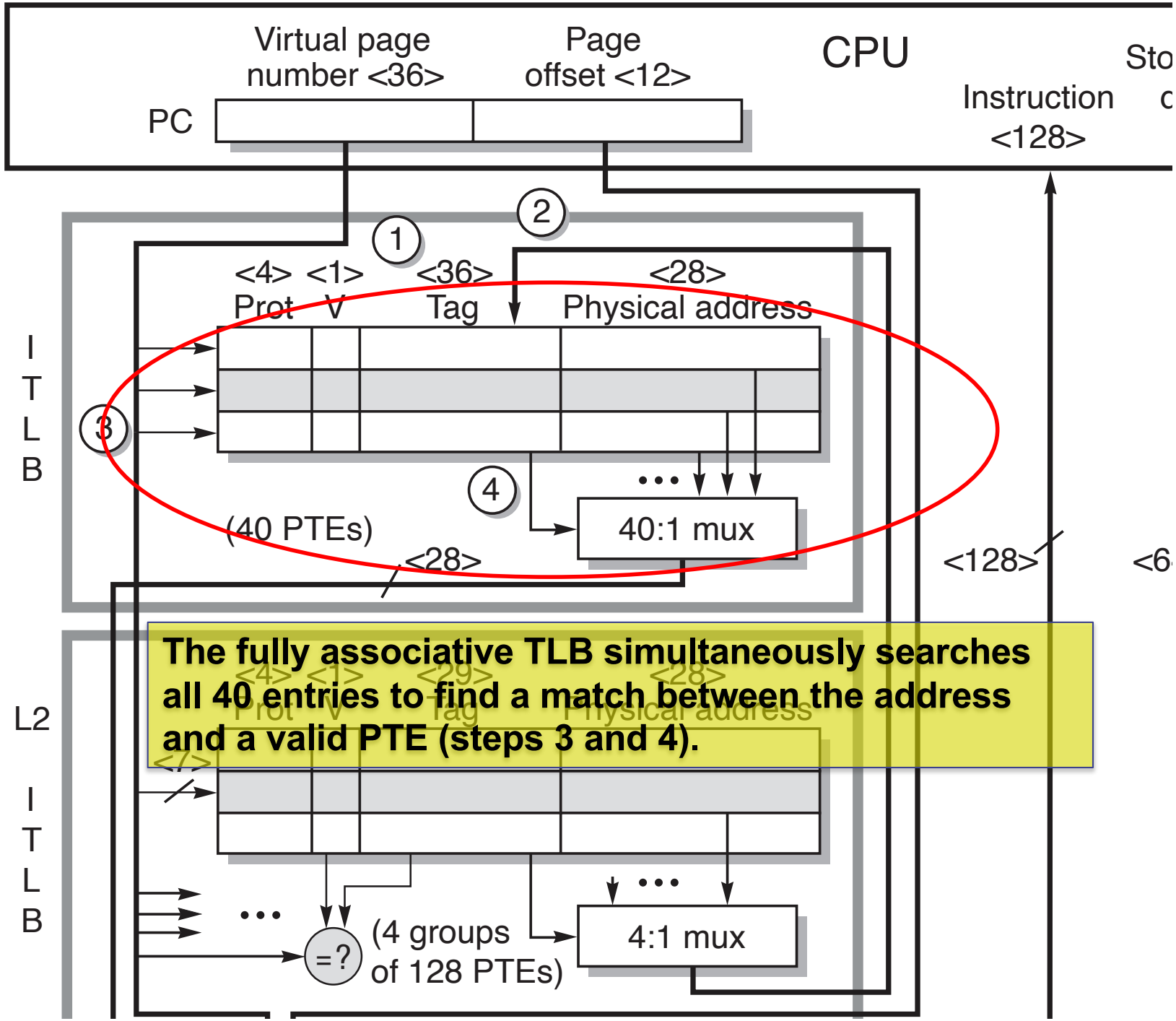

CPU
Virtual page number <36>
Page offset <12>
PC
Instruction <128>
Store queue/ data out <64>
Data virtual page number <36>
Page offset <12>
Data in <64>
ITLB
DTLB
<4> Prot
<1> V
<36> Tag
<28> Physical address
(40 PTEs)
40:1 mux
L2 ITLB
L2 DTLB
<29> Tag
<7>
(4 groups of 128 PTEs)
4:1 mux
ICACHE
DCACHE
<9> Index
<6> Block offset
V <1>
D <1>
Tag <25>
Data <512>
(2 groups of 512 blocks)
2:1 mux
L2 Prefetcher
Victim buffer
Address <38>
<40>
<128>
L2 CACHE
<24> Tag
<10> Index
(16 groups of 1024 blocks)
16:1 mux
MAIN MEMORY
<15>
<64>
System chip memory crossbar
DIMM

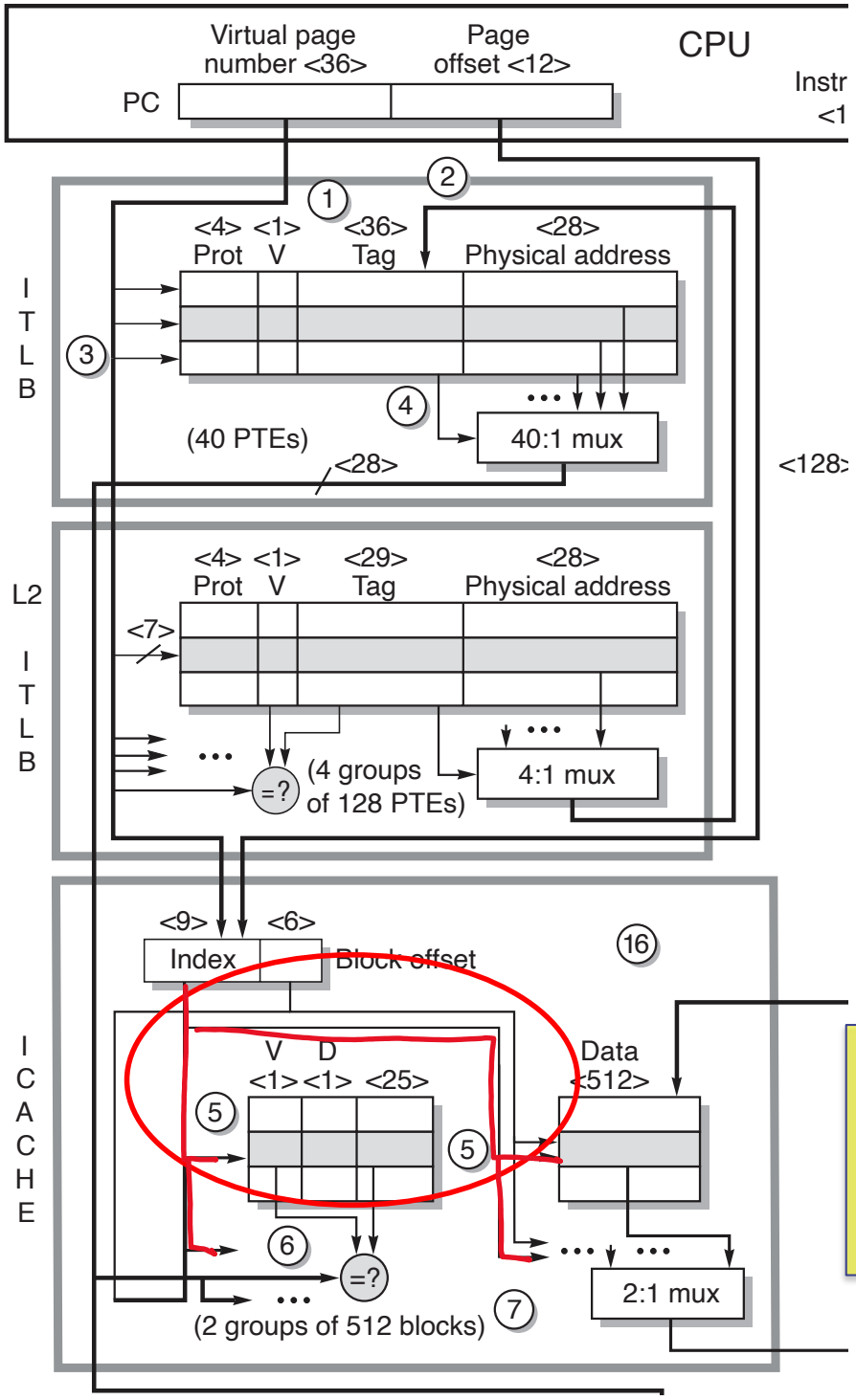
1. The L1 caches are both 64 KB, 2-way set associative with 64-byte blocks and LRU replacement.
2. The L2 cache is 1 MB, 16-way set associative with 64-byte blocks, and pseudo LRU replacement.
3. The data and L2 caches use write back with write allocation.
4. The L1 instruction and data caches are virtually indexed and physically tagged, so every address must be sent to the instruction or data TLB at the same time as it is sent to a cache.
5. Both TLBs are fully associative and have 40 entries, with 32 entries for 4 KB pages and 8 for 2 MB or 4 MB pages.
6. Each TLB has a 4-way set associative L2 TLB behind it, with 512 entities of 4 KB page sizes.
7. Opteron supports 48-bit virtual addresses and 40-bit physical addresses.


Virtual page number <36>
Page offset <12>
CPU
PC
Instruction <128>
1
2
3
4
<4> Prot
<1> V
<36> Tag
<28> Physical address
ITLB
(40 PTEs)
40:1 mux
<28>
<128>
The page frame of the instruction's data address is sent to the instruction TLB (step 1)
L2 ITLB
<7>
=?
(4 groups of 128 PTEs)
4:1 mux

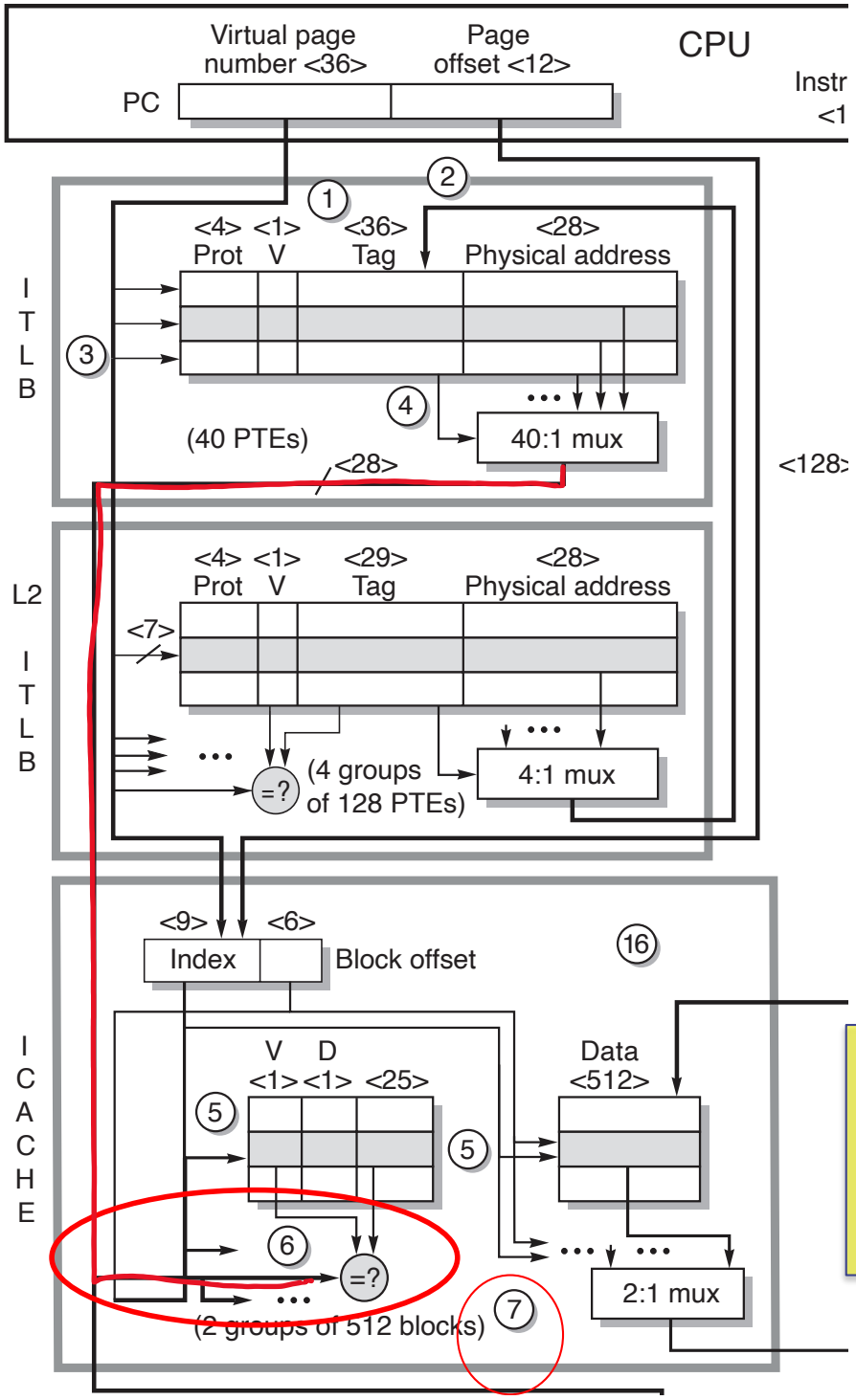


the 9-bit index from the virtual address is sent to the data cache (step 2)


Virtual page number <36>
Page offset <12>
CPU
PC
Instruction <128>
Sto
c
1
2
<4> <1> <36> <28>
Prot V Tag Physical address
I T L B
3
4
(40 PTEs)
40:1 mux
<28>
<128>
<6
The fully associative TLB simultaneously searches all 40 entries to find a match between the address and a valid PTE (steps 3 and 4).
L2
I T L B
<7>
=?
(4 groups of 128 PTEs)
4:1 mux



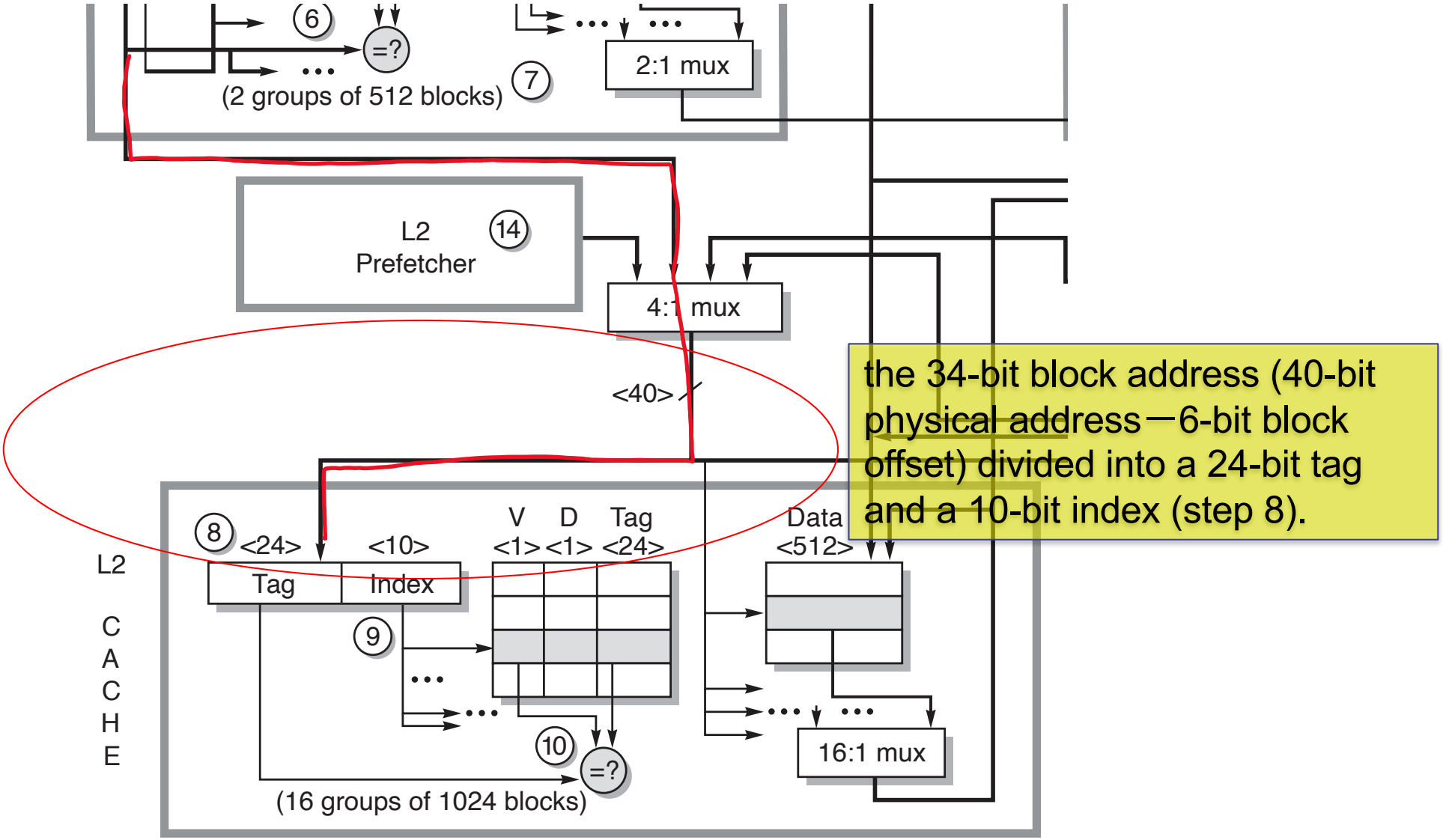
The index field of the address is sent to both groups of the two-way set-associative data cache (step 5).

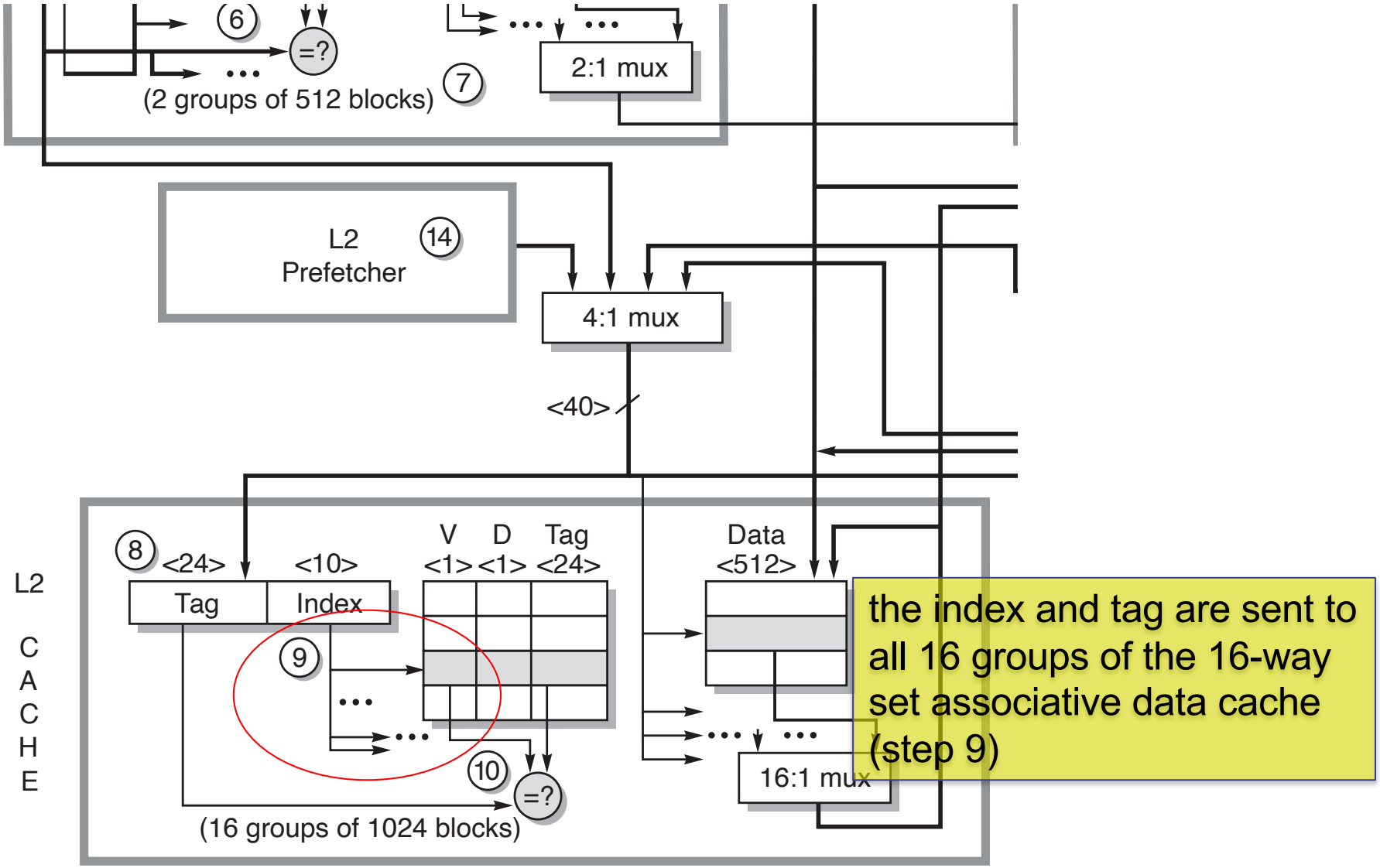


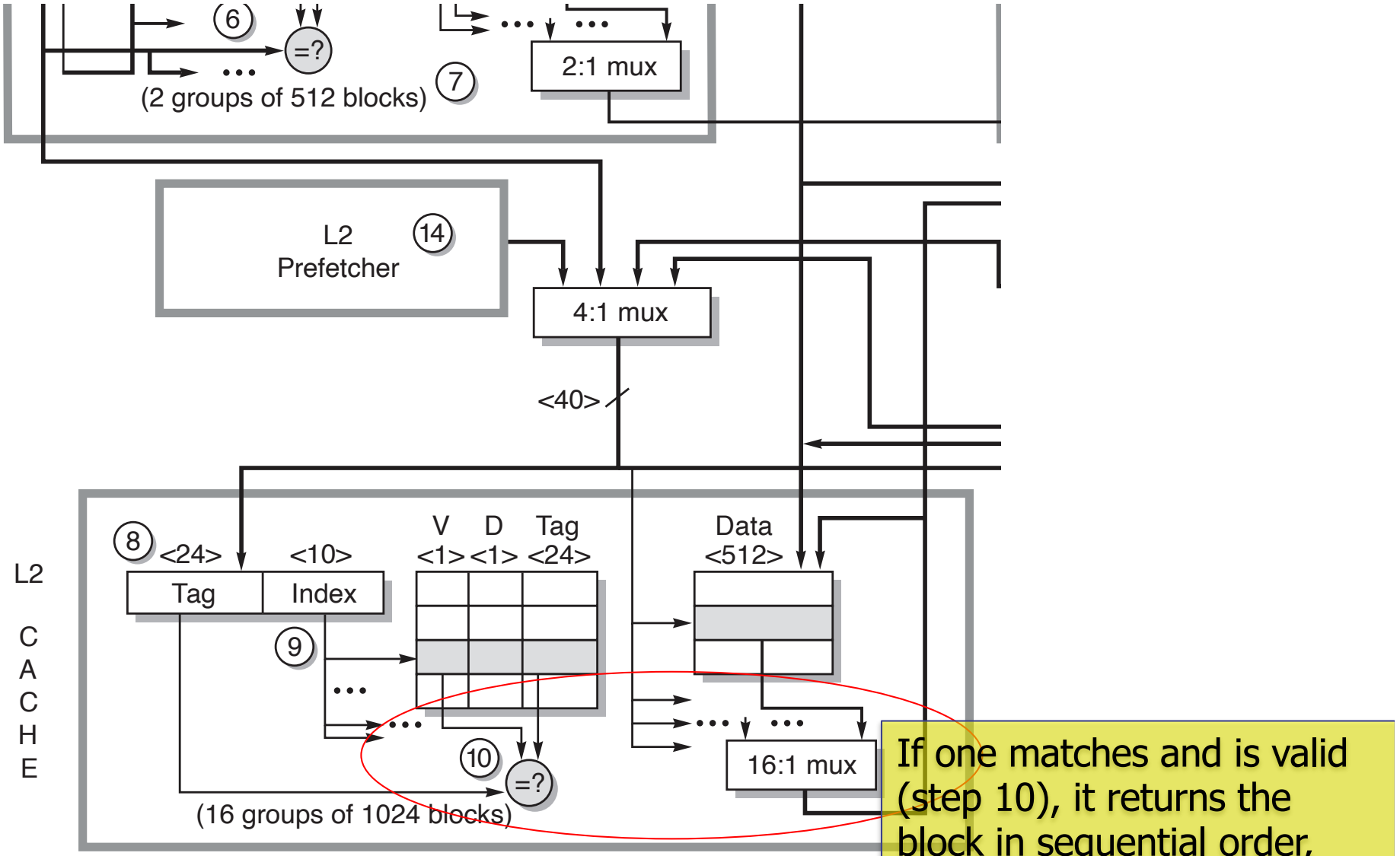
L1 Cache: VIPT(virtually indexed and physically tagged)

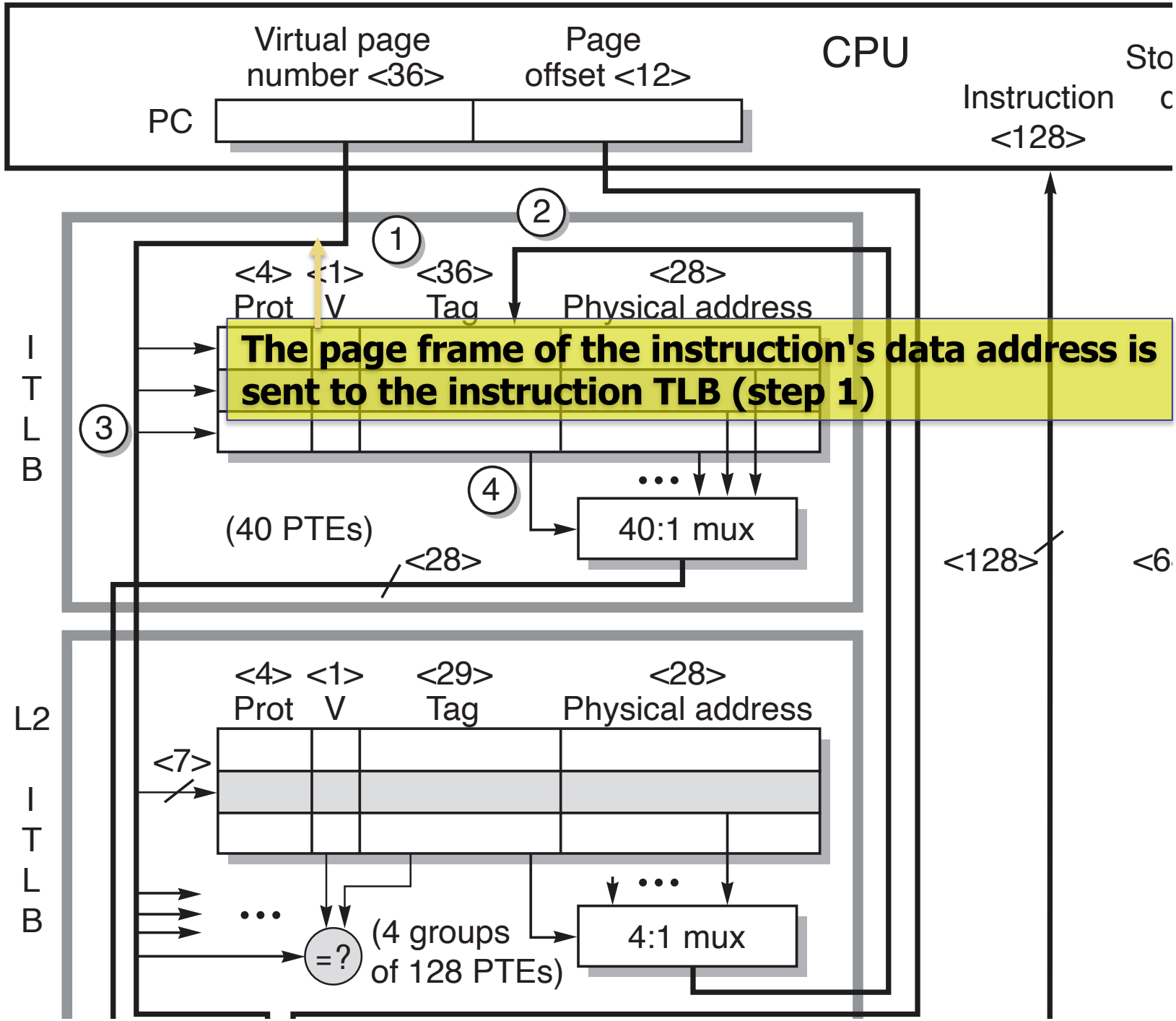
On a miss, the cache controller must check for a synonym (two different virtual addresses that reference the same physical address) **(step 7).**

The tags and valid bits are compared to the physical page frame from the Instruction TLB (step 6).


6
=?
(2 groups of 512 blocks)
7
2:1 mux
L2
Prefetcher
14
4:1 mux
<40>
the 34-bit block address (40-bit physical address−6-bit block offset) divided into a 24-bit tag and a 10-bit index (step 8).
8
<24>
<10>
Tag
Index
V
D
Tag
<1>
<1>
<24>
Data
<512>
L2
C
A
C
H
E
9
10
=?
(16 groups of 1024 blocks)
16:1 mux


6
=?
(2 groups of 512 blocks)
7
2:1 mux
L2
Prefetcher
14
4:1 mux
<40>
L2
CACHE
8
<24>
<10>
Tag
Index
V
D
Tag
<1>
<1>
<24>
9
10
=?
(16 groups of 1024 blocks)
Data
<512>
16:1 mux
the index and tag are sent to all 16 groups of the 16-way set associative data cache (step 9)


6
=?
(2 groups of 512 blocks)
7
2:1 mux
L2
Prefetcher
14
4:1 mux
<40>
L2
CACHE
8
<24>
<10>
Tag
Index
V
D
Tag
<1>
<1>
<24>
Data
<512>
9
10
=?
(16 groups of 1024 blocks)
16:1 mux
If one matches and is valid (step 10), it returns the block in sequential order,



The page frame of the instruction's data address is sent to the instruction TLB (step 1)

第7章 存储系统（Memory Hierarchy）

7.1 存储系统的基本知识
7.2 Cache基本知识
7.3 降低Cache不命中率
7.4 减少Cache不命中开销
7.5 减少命中时间
7.7 虚拟存储器
7.8 实例：AMD Opteron的存储器层次结构

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{Total miss latency} - \text{Overlapped miss latency})$$

$$\text{Average memory access time} = \text{Hit time}_{\text{L1}} + \text{Miss rate}_{\text{L1}} \times (\text{Hit time}_{\text{L2}} + \text{Miss rate}_{\text{L2}} \times \text{Miss penalty}_{\text{L2}})$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}_{\text{L1}}}{\text{Instruction}} \times \text{Hit time}_{\text{L2}} + \frac{\text{Misses}_{\text{L2}}}{\text{Instruction}} \times \text{Miss penalty}_{\text{L2}}$$

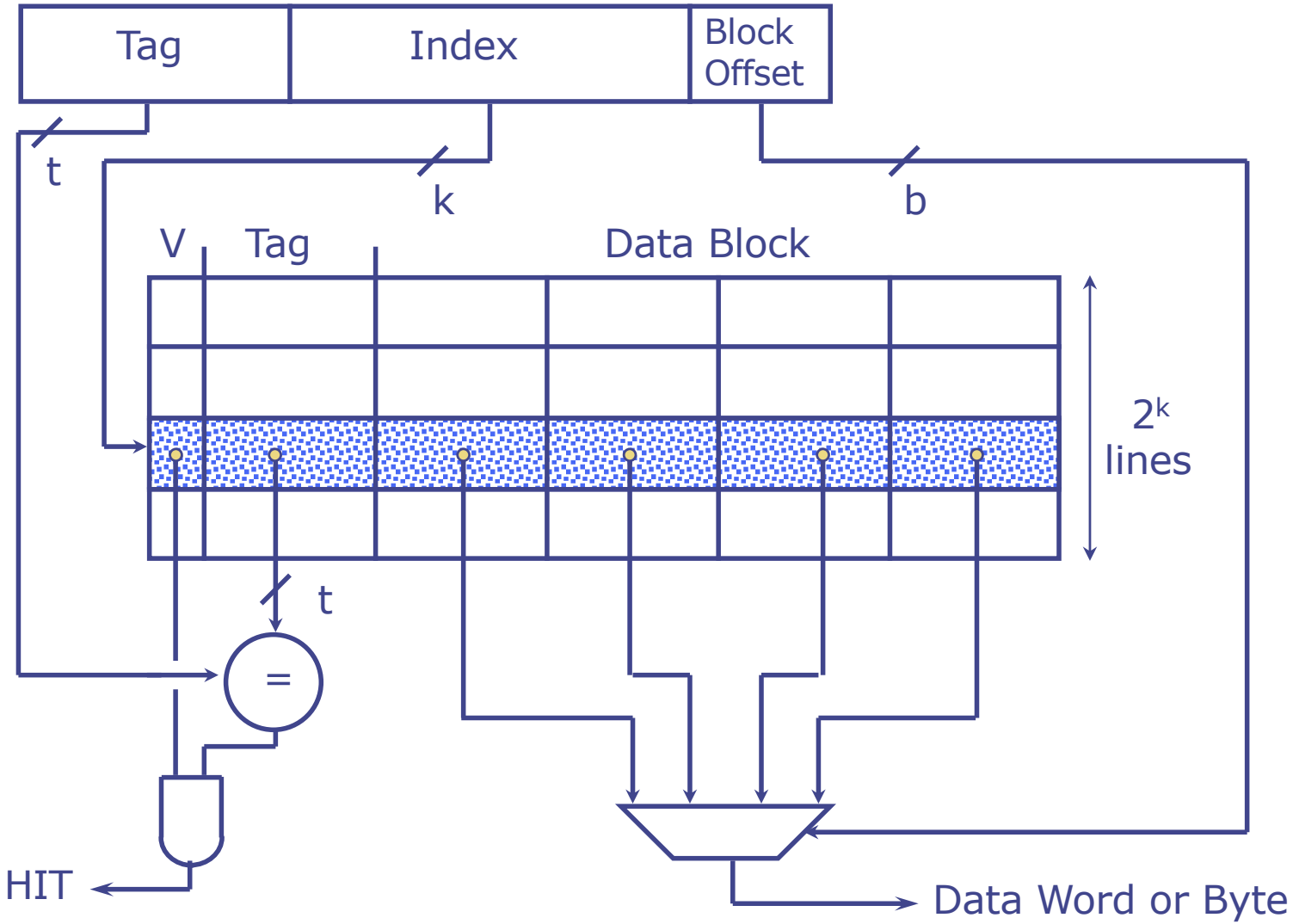
Figure C.7 Summary of performance equations in this appendix. The first equation calculates the cache index size, and the rest help evaluate performance. The final two equations deal with multilevel caches, which are explained early in the next section. They are included here to help make the figure a useful reference.

- Reducing the hit time: small and simple caches, way prediction, and trace caches
- Increasing cache bandwidth: pipelined caches, multibanked caches, and non-blocking caches
- Reducing the miss penalty: critical word first and merging write buffers
- Reducing the miss rate: compiler optimizations
- Reducing the miss penalty or miss rate via parallelism: hardware prefetching and compiler prefetching

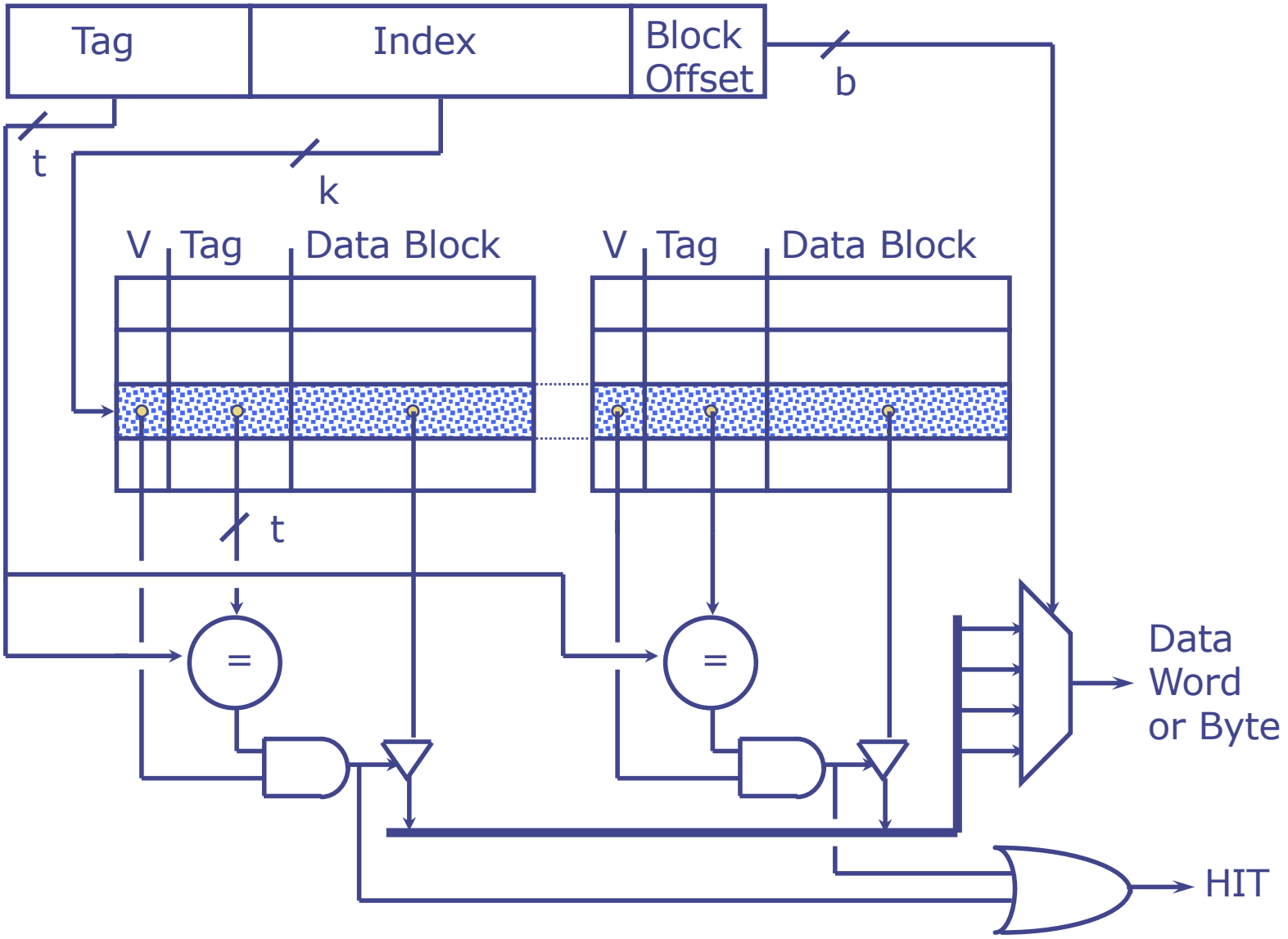
Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4 L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size ≠ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

Technique	Hit time	Band-width	Miss pen-alty	Miss rate	Hardware cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of Opteron and Niagara
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data			+	+	2 instr., 3 data	Many prefetch instructions; Opteron and Pentium 4 prefetch data
Compiler-controlled prefetching			+	+	3	Needs nonblocking cache; possible instruction overhead; in many CPUs

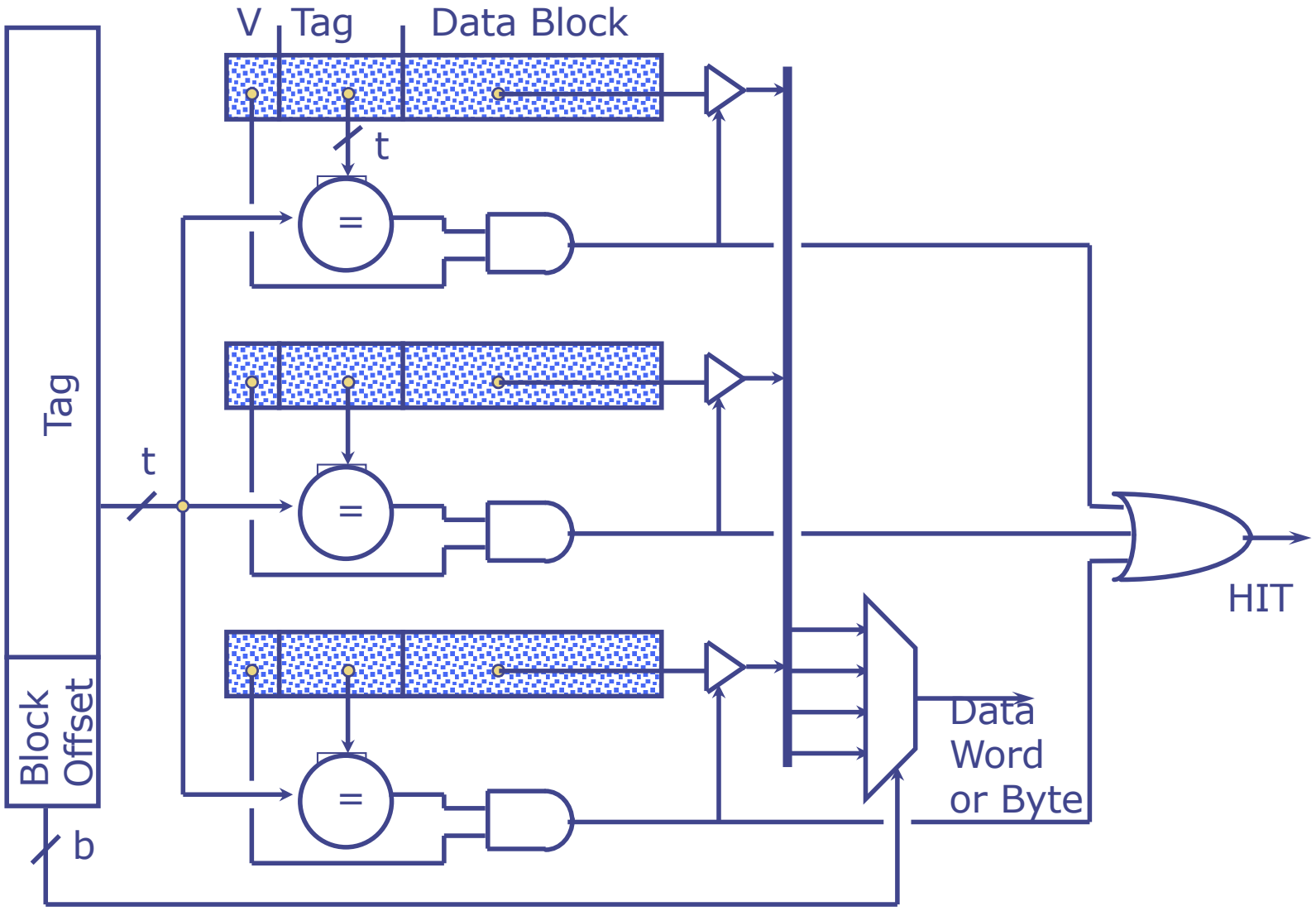
Direct-Mapped Cache



2-Way Set-Associative Cache



Fully Associative Cache



1. L1数据Cache大小为64KB，Cache块大小为64 Byte，采用2路组相联，LRU替换算法，写回策略，Write allocate，即写不命中时，将要写的块从下一级存储器调入Cache，然后再像写命中时一样操作（在使用写回策略时，将数据写入相应Cache块）。
2. $2^{Index} = \frac{Cache容量}{Cache块大小 \times 组相联度} = \frac{64KB}{64B \times 2} = 2^9$，意味着共有512组，每组有两路。
3. **Cache可分为两个group，每个group有4096个64位字。4K*8Byte = 32 KB。每个group包含512 Blocks。**
4. 留给处理器访问的地址长度是40位，因为这是物理地址而不是虚拟地址。
5. Opteron向Cache提供48位虚拟地址，这个地址同时被转换成40位物理地址，用于tag比较。(1)

6. 进入Cache的物理地址被分为来部分：34bit块地址和6位块内偏移。块地址又进一步分为地址标识和Cache索引。
7. 根据Cache索引选择要比较的tag，以确定所需的块是否在Cache中。Index的大小取决于Cache的大小，块大小和组相联度。

$$2^{Index} = \frac{Cache\text{容量}}{\text{Cache块大小} \times \text{组相联度}} = \frac{64\text{KB}}{64\text{B} \times 2} = 2^9$$

因此，索引为9位，tag是34-9=25bit。 (2)

8. 在从Cache中读取两个tags之后，它们和来自处理器的块地址的tag部分进行比较。如果valid位是置位的，那么tag包含的信息是有效的。(3)
9. 假设有一个tag是匹配的，最后的步骤是通知CPU从多路选择器的输出取得正确的数据。(4)