计算机体系结构

Topic IV: Memory Hierarchy

第4章　存储层次结构设计

- **存储层次结构**
- **Cache基本知识**
- **基本的Cache优化方法**
- **高级的Cache优化方法**
- **存储器技术与优化**
- **虚拟存储器 - 基本原理**

存储层次结构

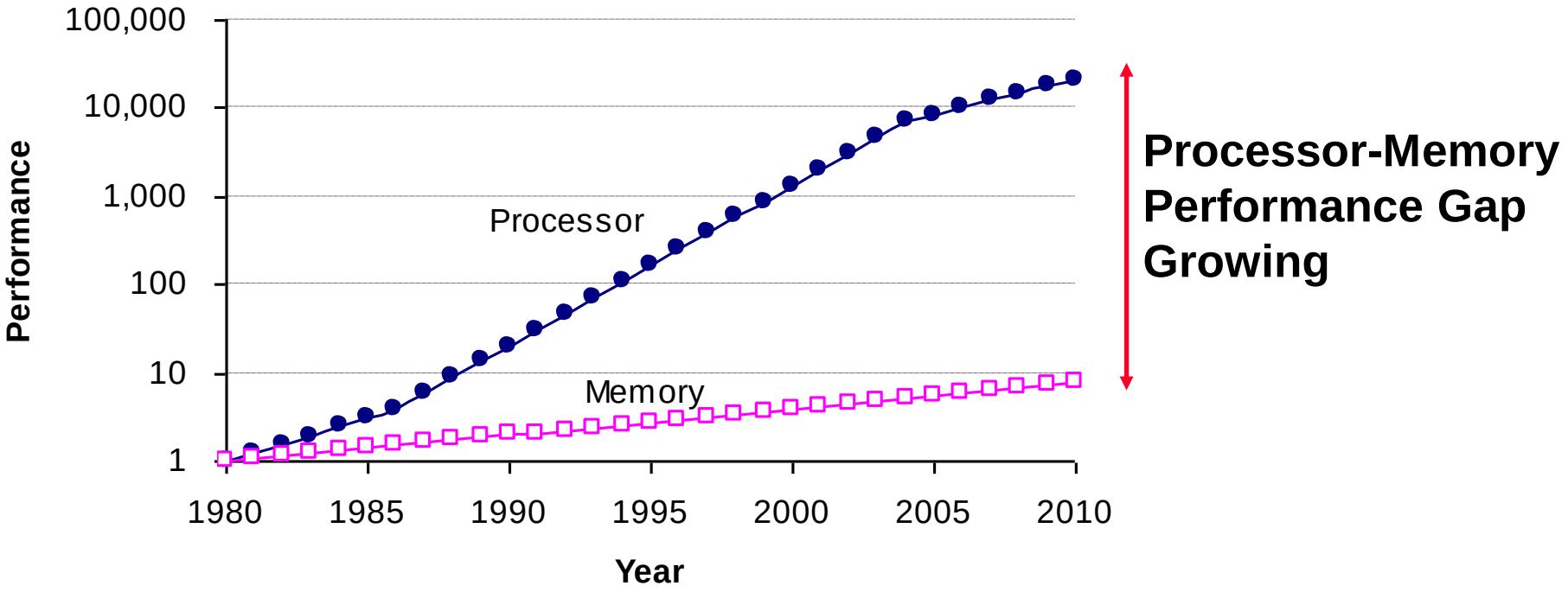
- **存储系统设计是计算机体系结构设计的关键问题之一**
  - 价格，容量，速度的权衡
- **用户对存储器的“容量，价格和速度”要求是相互矛盾的**
  - 速度越快，每位价格就高
  - 容量越大，每位价格就低
  - 容量越大，速度就越慢
  - 目前主存一般由DRAM构成
- **Microprocessor与Memory之间的性能差异越来越大**
  - CPU性能提高大约60%/year
  - DRAM 性能提高大约 9%/year

Trends in DRAM

Year Produced	Chip Size	DRAM Type	Bus Cycle	Latency New Request
1980	64 Kbit	Asynchronous DRAM		250 ns
1983	256 Kbit	Asynchronous DRAM		190 ns
1986	1 Mbit	Asynchronous DRAM		140 ns
1989	4 Mbit	Asynchronous DRAM		110 ns
1992	16 Mbit	Asynchronous DRAM		90 ns
1996	64 Mbit	SDRAM	10 ns	70 ns
1998	128 Mbit	SDRAM	7.5 ns	60 ns
2000	256 Mbit	DDR SDRAM	6 ns	55 ns
2002	512 Mbit	DDR SDRAM	5 ns	55 ns
2004	1 Gbit	DDR2 SDRAM	3 ns	50 ns
2006	2 Gbit	DDR2 SDRAM	2 ns	45 ns
2010	4 Gbit	DDR3 SDRAM	1.5 ns	37 ns
2012	8 Gbit	DDR3 SDRAM	1 ns	31 ns

微处理器与DRAM 的性能差异

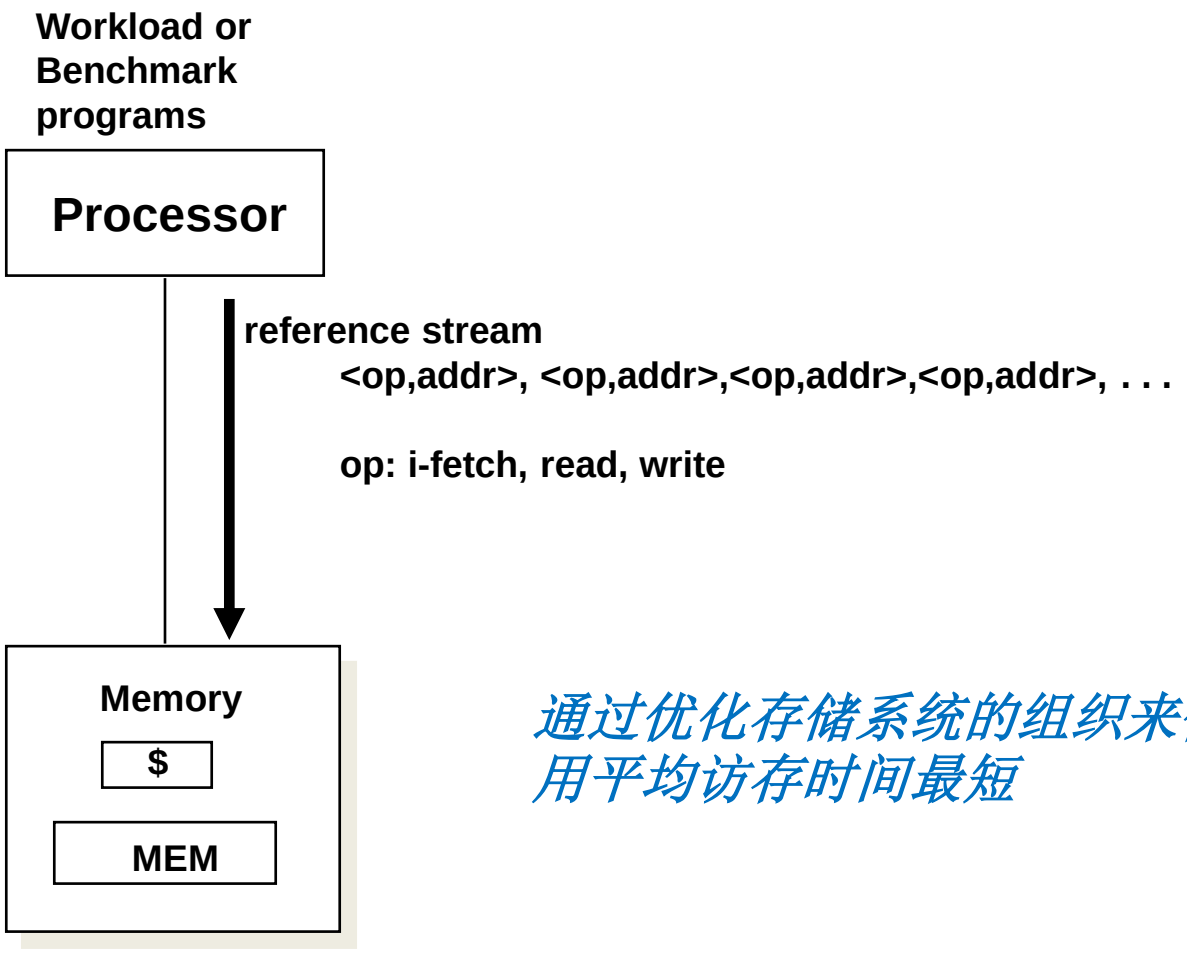
Processor-DRAM Memory Gap (latency)



Microprocessor-DRAM性能差异

- **利用caches缓解微处理器与存储器性能上的差异**
- **Microprocessor-DRAM 性能差异**
  - time of a full cache miss in instructions executed
  - 1st Alpha : 340 ns/5.0 ns = 68 clks x 2 or 136 instructions
  - 2nd Alpha :266 ns/3.3 ns = 80 clks x 4 or 320 instructions
  - 3rd Alpha :180 ns/1.7 ns =108 clks x 6 or 648 instructions

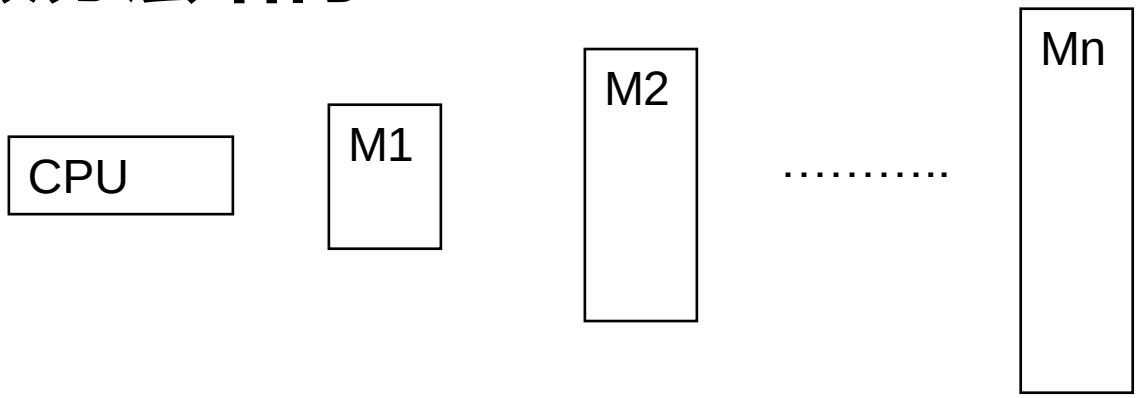
存储系统的设计目标



通过优化存储系统的组织来使得针对典型应用平均访存时间最短

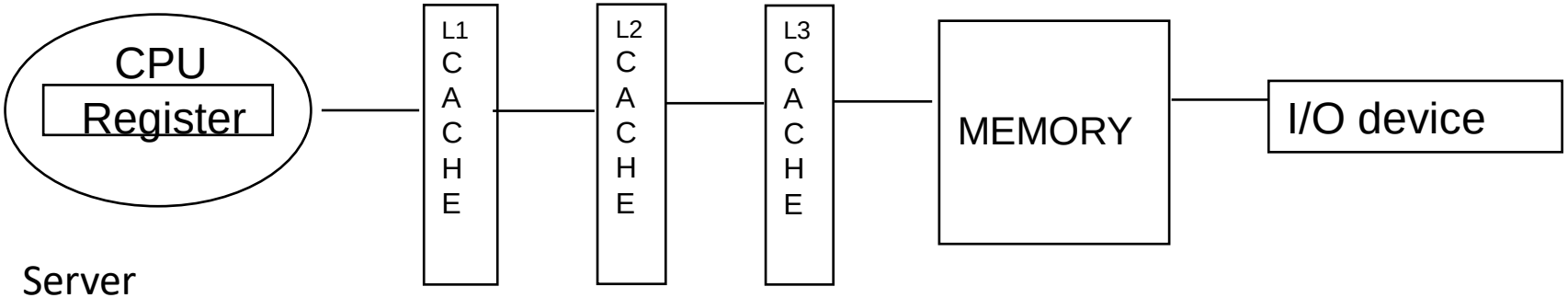
基本解决方法：多级层次结构

- **多级分层结构**



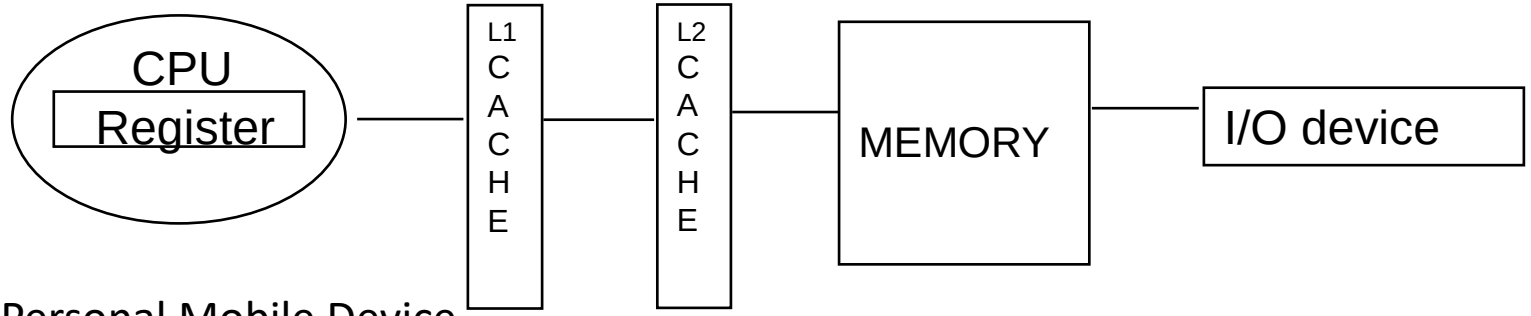
  - M1 速度最快，容量最小，每位价格最高
  - Mn速度最慢，容量最大，每位价格最低
- **并行**
- **存储系统接近M1的速度，容量和价格接近Mn**

计算机系统的多级存储层次



Server

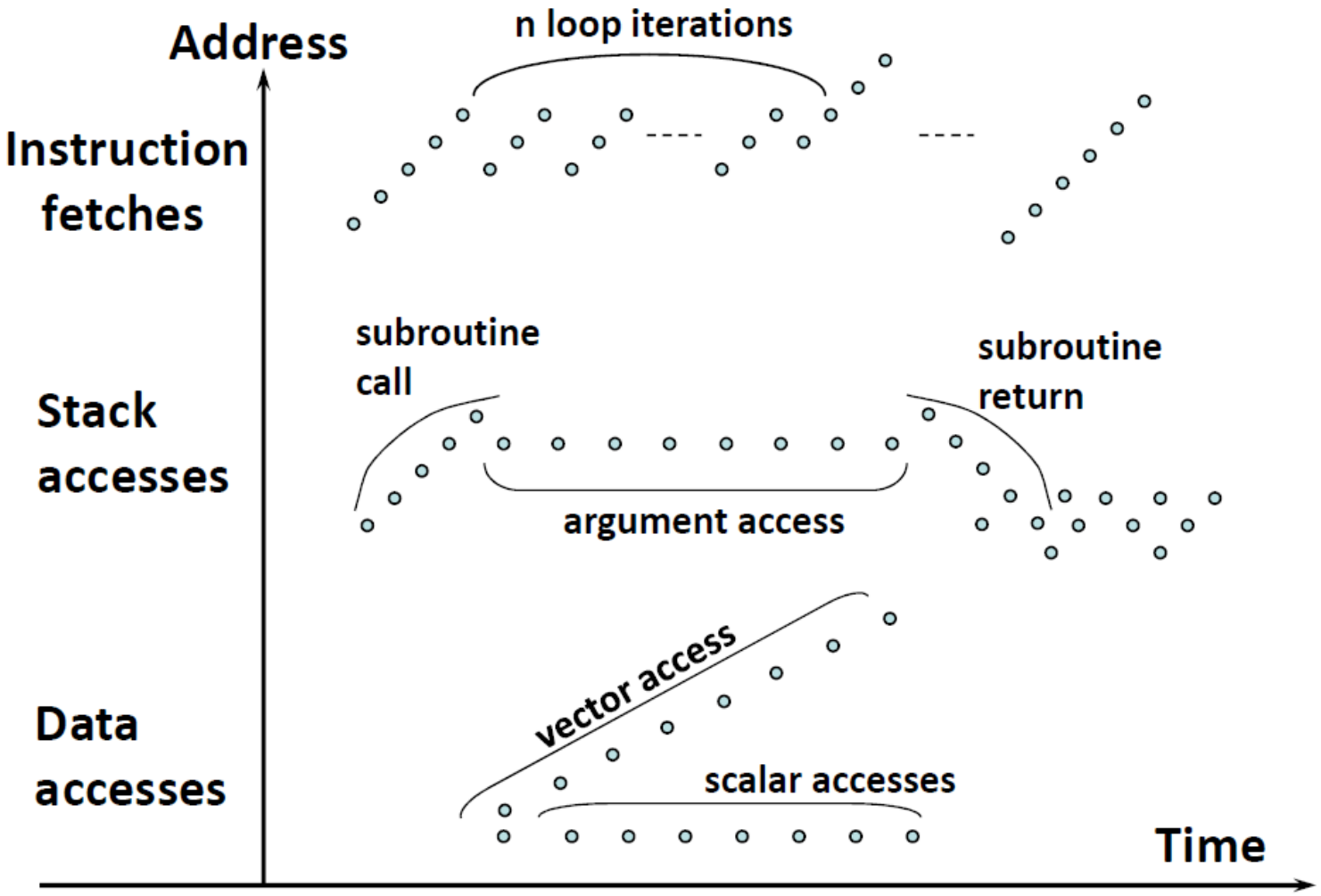
300ps	1ns	3-10ns	10-20ns	50-100ns	5-10ms
1000B	64KB	256K	2-4MB	4-16GB	4-16TB



Personal Mobile Device

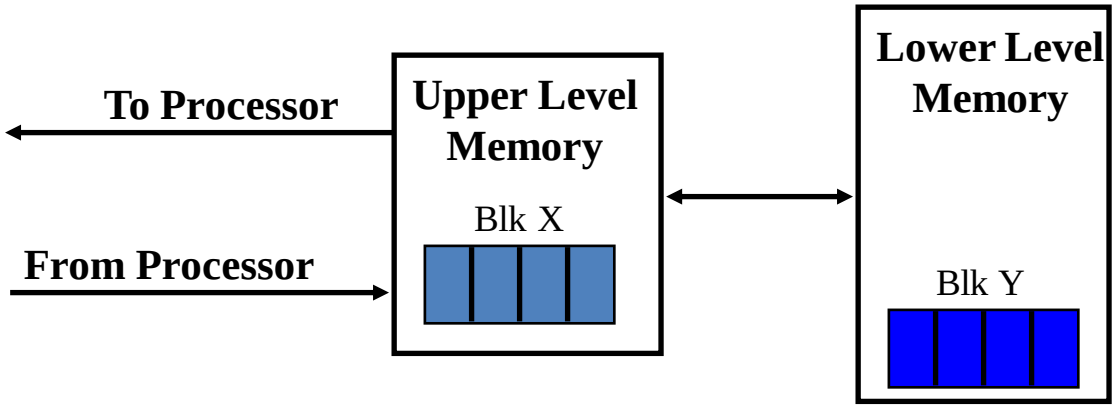
500ps	2ns	10-20ns	50-100ns	25-50μs
500B	64KB	256K	256-512MB	4-8GB

典型的存储器访问模式


Address
n loop iterations
Instruction fetches
subroutine call
subroutine return
Stack accesses
argument access
vector access
scalar accesses
Data accesses
Time

存储层次工作原理： Locality!

- **应用程序局部性原理: 给用户**
  - 一个采用低成本技术达到的存储容量.（容量大，价格低）
  - 一个采用高速存储技术达到的访问速度.（速度快）
- **Temporal Locality (时间局部性):**
  - =>保持最近访问的数据项最接近微处理器
- **Spatial Locality (空间局部性):**
  - 以由地址连续的若干个字构成的块为单位，从低层复制到上一层

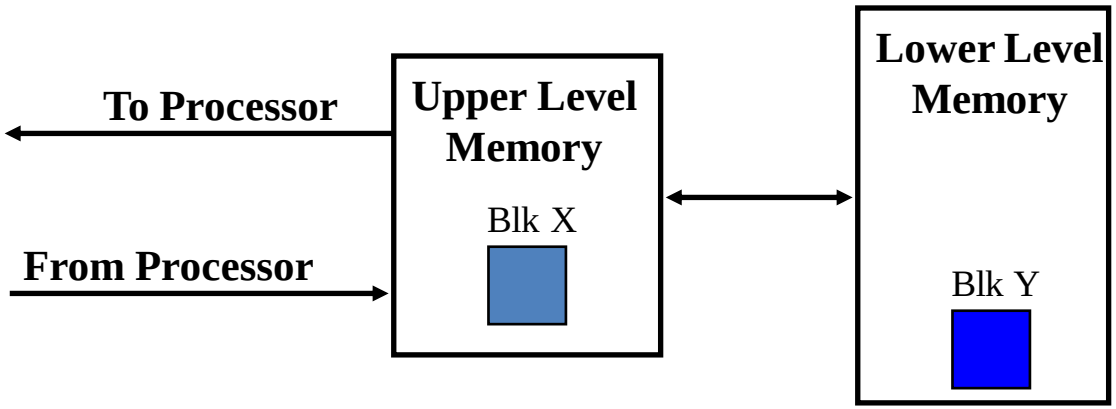


存储层次结构涉及的基本概念

- **Block**
  - Block （块）：不同层次的Block大小可能不同
  - 命中（hit）和命中率 (hit rate)
  - 失效 (miss) 和失效率 (miss rate)
- **镜像和一致性问题 (inclusion and consistency)**
  - 高层存储器是较低层存储器的一个镜像
  - 高层存储器内容的修改必须反映到低层存储器中
    - 数据一致性问题
- **寻址：不管如何组织，我们必须知道如何访问数据**
- **要求：不同层次上块大小可以不同**
  - 在L0 cache 可能以Double, Words, Halfwords, 或bytes
  - 在L1cache仅以cache line 或 slot为单位访问
  - 在更低层.....
  - 因此总是存在地址映射问题
  - 物理地址格式　Block Address + Block Offset

存储层次的性能参数(1/2)

- **假设采用二级存储：M1和M2**
  - M1和M2的容量 (size)、价格(cost)、访问时间(time to access)分别为：
  - S1 、C1、TA1　　S2、C2、TA2



存储层次的性能参数 (2/2)

- **存储层次的平均每位价格C**
  - C=(C1*S1+C2*S2)/(S1+S2)
- **命中(Hit): 访问的块在存储系统的较高层次上**
  - 若一组程序对存储器的访问，其中N1次在M1中找到所需数据，N2次在M2中找到数据 则
  - Hit Rate（命中率）: 存储器访问在较高层命中的比例 H= N1/(N1+N2)
  - Hit Time（命中时间）:访问较高层的时间，TA1
- **失效(Miss):访问的块不在存储系统的较高层次上**
  - Miss Rate （失效）= 1 - (Hit Rate) = 1 – H = N2/(N1+N2)
  - 当在M1中没有命中时：一般必须从M2中将所访问的数据所在块搬到M1中，然后CPU才能在M1中访问。
  - 设传送一个块的时间为TB,即不命中时的访问时间为：TA2+TB+TA1 = TA1+TM
  - TM 通常称为失效开销
- **平均访存时间:**
  - 平均访存时间 TA = H*TA1+(1-H)*(TA1+TM) = TA1+(1-H)*TM

常见的存储层次的组织

- **Registers <-> Memory**
  - 由编译器完成调度
- **cache <-> memory**
  - 由硬件完成调度
- **memory <-> disks**
  - 由硬件和操作系统（虚拟管理）
  - 由程序员完成调度

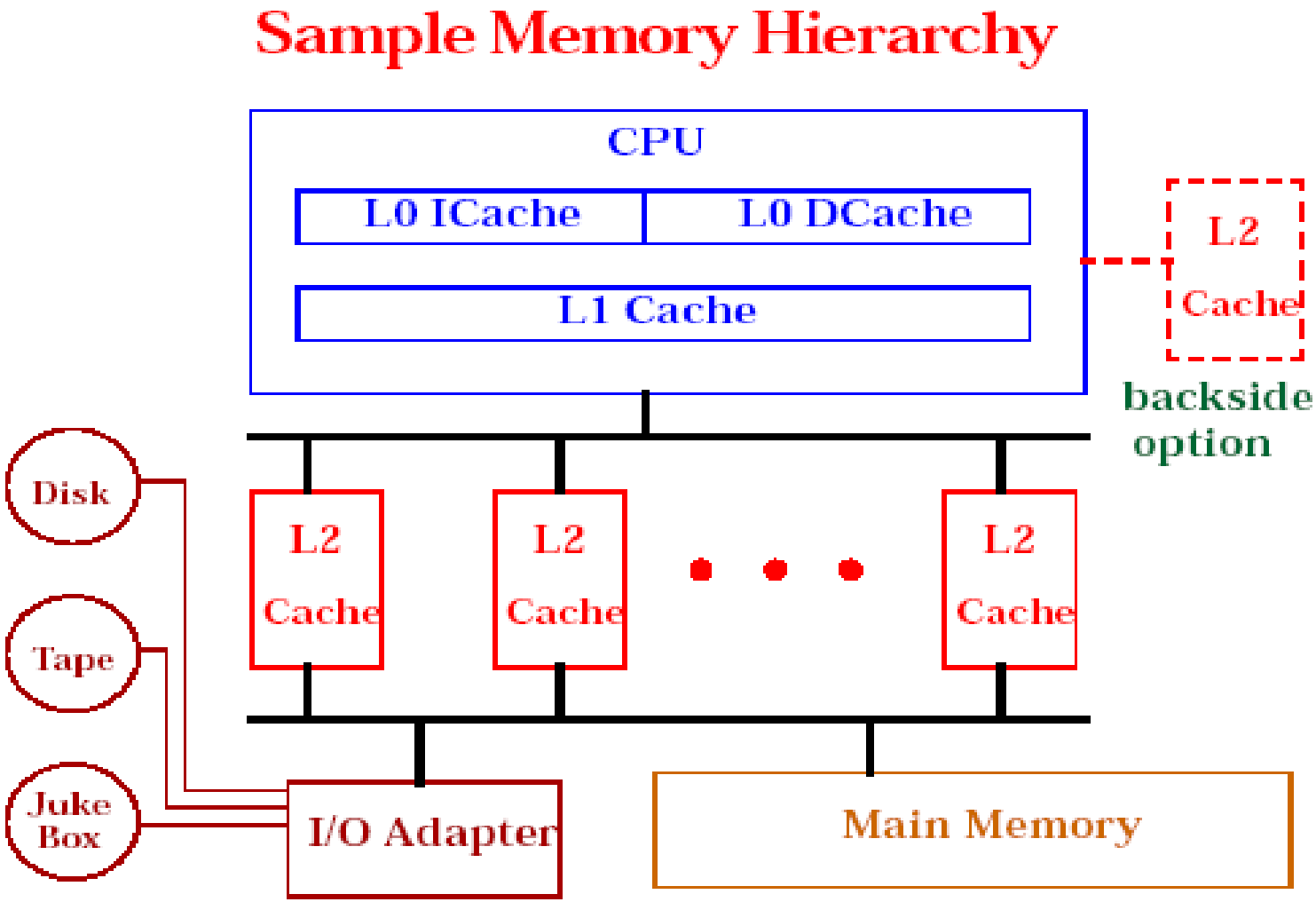
Cache Memory

- **小而快（SRAM）的存储技术**
  - 存储正在访问的部分指令和数据
- **用于减少平均访存时间**
  - 通过保持最近访问的数据在处理器附近来挖掘时间局部性
  - 通过以块为单位在不同层次移动数据来挖掘空间局部性
- **主要目标：**
  - 提高访存速度
  - 降低存储系统成本

Cache 无处不在

- **体系结构中，Cache无处不在**
- **寄存器：Cache on variables**
- **一、二级Cache： Cache on memory**
- **Memory：Cache on hard disks**
  - 存储最近执行的程序和数据
  - Hard disks可以视为主存的扩展（VM）
- **分支目标缓存(branch target buffer)及分支预测缓存(branch prediction buffer)**
  - 缓存分支目标及预测信息

Cache基本知识


Sample Memory Hierarchy
CPU
L0 ICache
L0 DCache
L1 Cache
L2
Cache
backside
option
Disk
Tape
Juke
Box
L2
Cache
L2
Cache
L2
Cache
I/O Adapter
Main Memory

Q1：映象(mapping)规则

- **当要把一个块从主存调入Cache时，如何放置问题**
- **三种方式**
  - 全相联方式 (Fully Associative)：即所调入的块可以放在cache中的任何位置
  - 直接映象方式 (Direct Mapped)：主存中每一块只能存放在cache中的唯一位置。 一般，主存块地址i 与cache中块地址j 的关系为：

    j = i mod （M），M为cache中的块数
  - 组相联映象(Set Associative)：主存中每一块可以被放置在Cache中唯一的一个组(set)中的任意一个位置，组由若干块构成，若一组由n块构成，我们称N路组相联
    - 组间直接映象
    - 组内全相联
    - 若cache中有G组，则主存中的第i 块的组号K为 K = i mod (G),

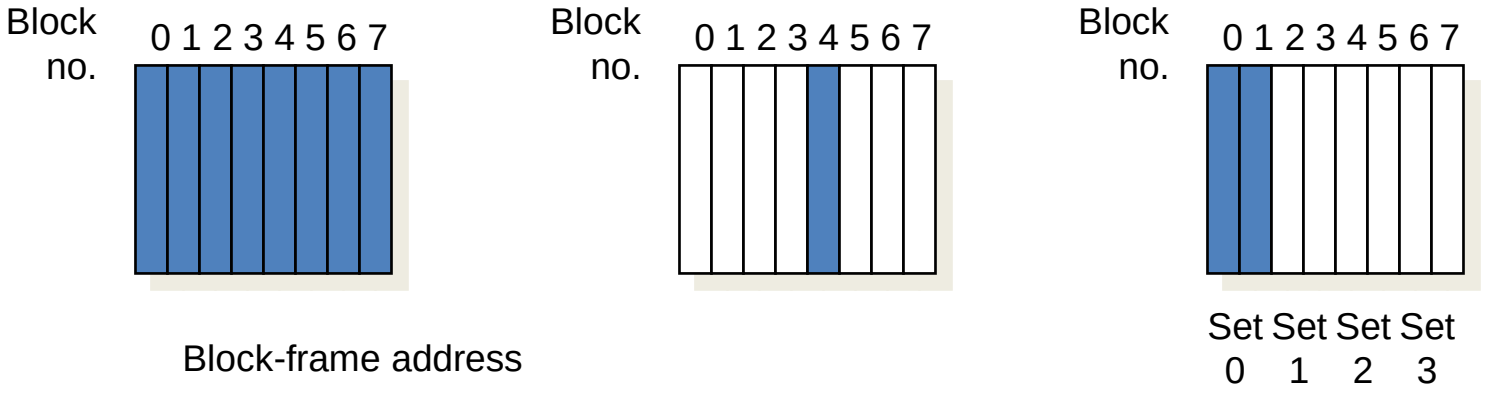
Q1: Where can a block be placed in the upper level?

- **Block 12 placed in 8 block cache:**
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

Set associative: block 12 can go anywhere in set 0 (12 mod 4)



Q1的讨论

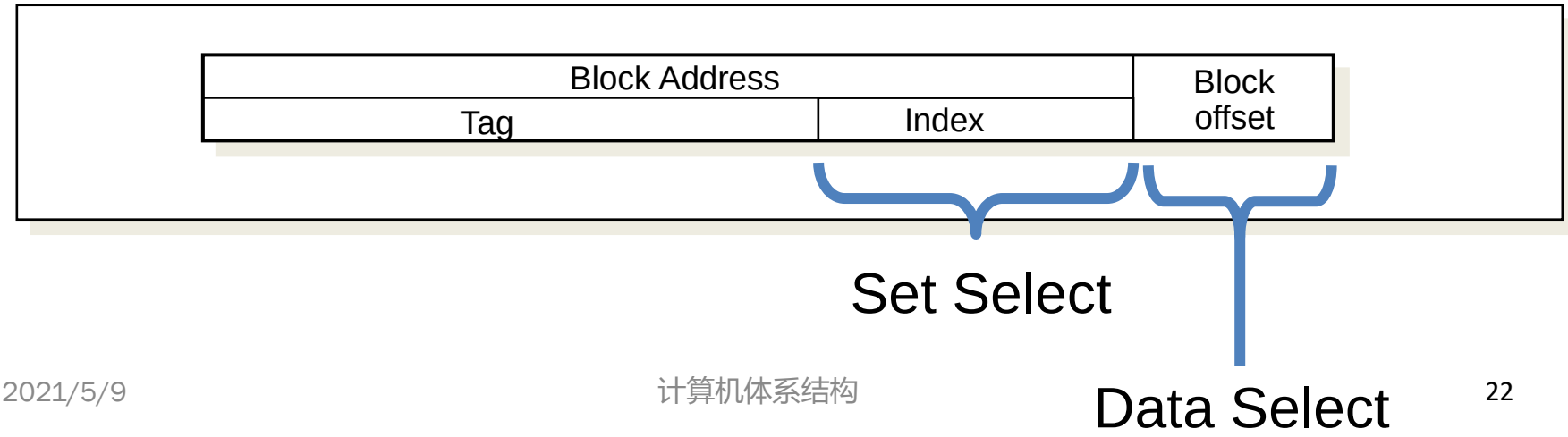
- **N-Way组相联：如果每组由N个块构成，cache的块数为M，则cache的组数G为M/N**
- **不同相联度下的路数和组数**

	路数	组数
全相联	**M**	**1**
直接相联	**1**	**M**
其他组相联	**1 < N <M**	**1 < G < M**

  - 相联度越高，cache空间利用率就越高，块冲突概率就越小，失效率就越低
  - N值越大，失效率就越低，但Cache的实现就越复杂，代价越大
  - 现代大多数计算机都采用直接映象，两路或四路组相联。

Q2(1/2): 查找方法

- **在CACHE中每一block都带有tag域（标记域），标记分为两类**
  - Address Tags：标记所访问的单元在哪一块中，这样物理地址就分为三部分： Address Tags ## Block index## block Offset
  - 全相联映象时，没有Block Index
  - 显然 Address tag越短，查找所需代价就越小
  - Status Tags：标记该块的状态，如Valid, Dirty等



Q2（2/2)查找方法

- **原则：所有可能的标记并行查找，cache的速度至关重要，即并行查找**
- **并行查找的方法**
  - 用相联存储器实现，按内容检索
  - 用单体多字存储器和比较器实现
- **显然相联度 N越大，实现查找的机制就越复杂，代价就越高**
- **无论直接映象还是组相联，查找时，只需比较 tag，index无需参加比较**

Tag和数据阵列并行访问的逻辑结构

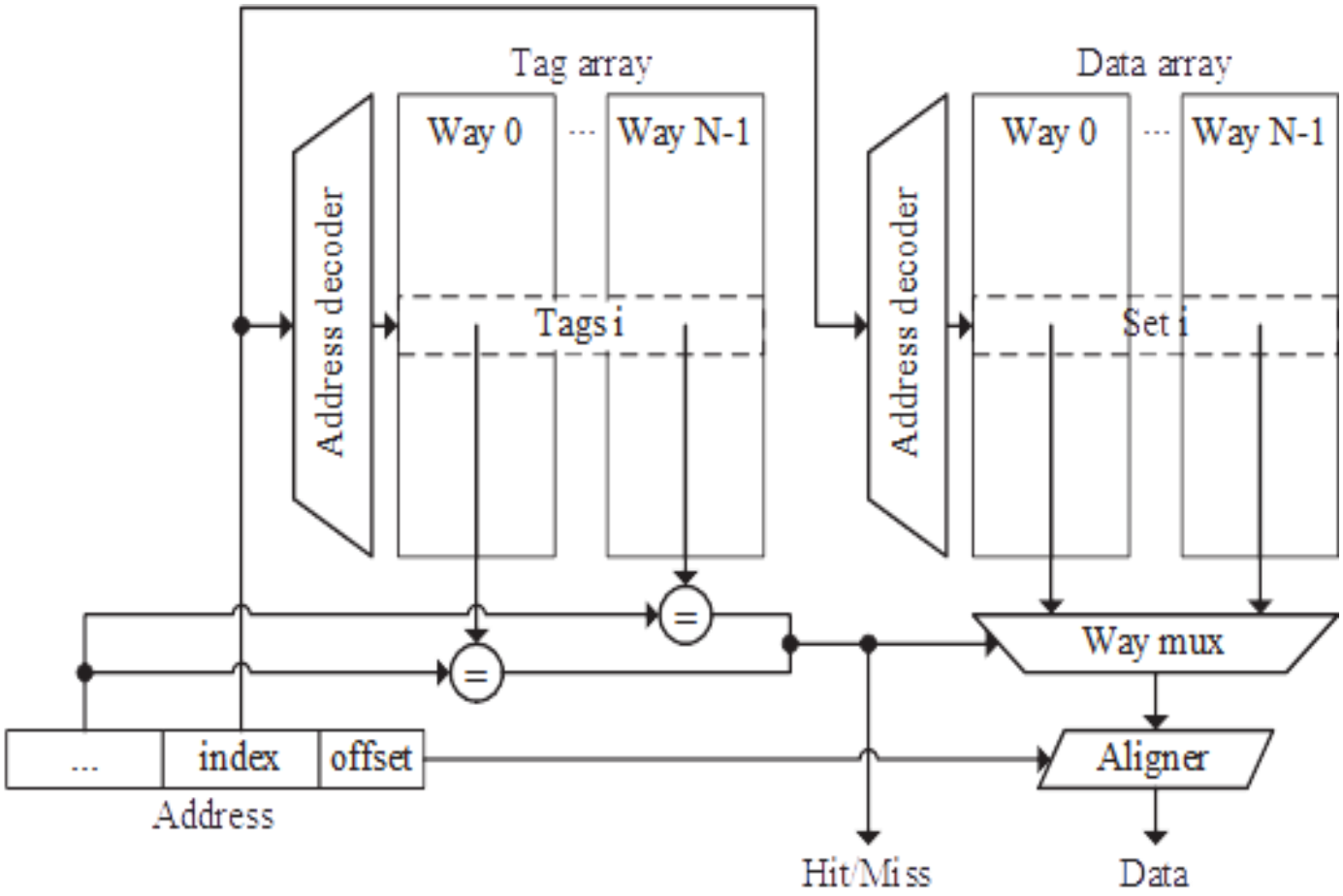


FIGURE 2.1: High-level logical cache organization.

Tag 和数据阵列并行访问的流水线模式

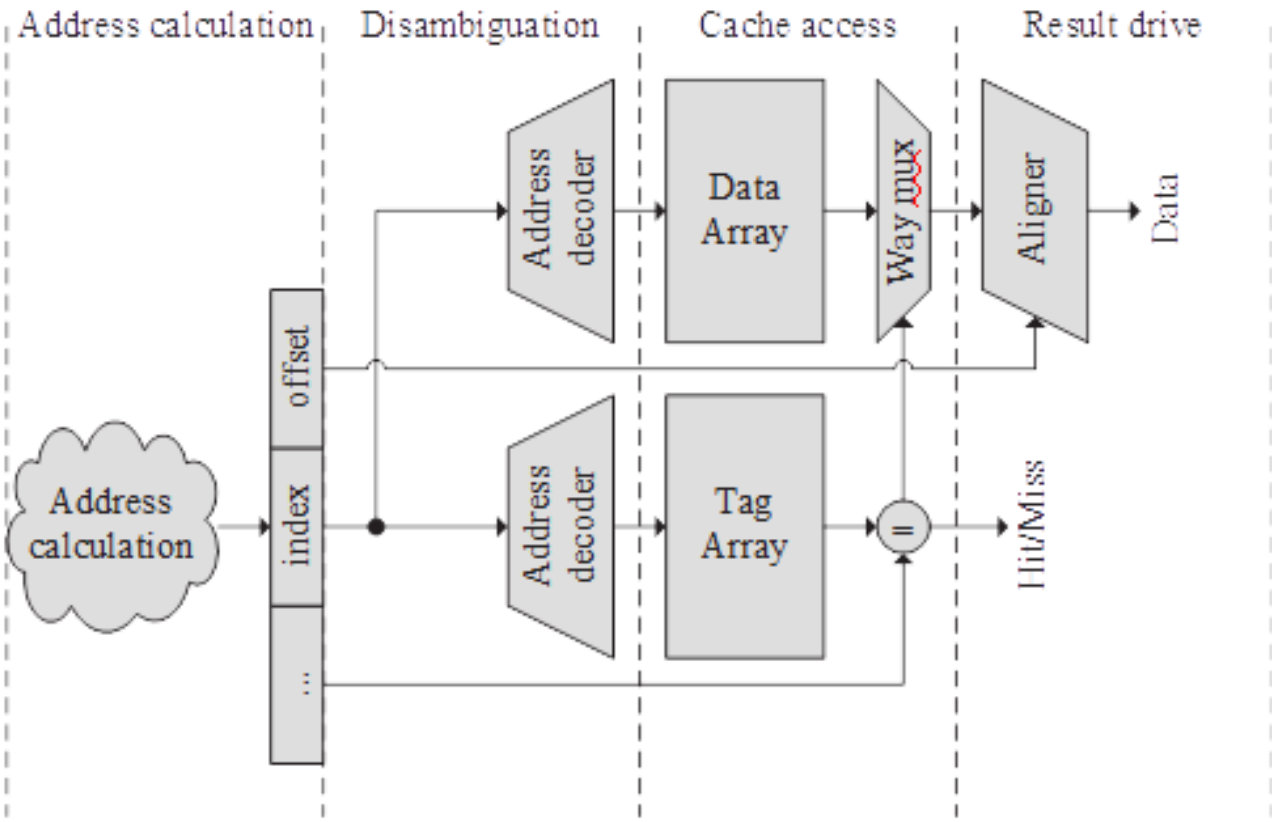


FIGURE 2.2: Parallel tag and data array access pipeline.

Tag和数据阵列串行访问的逻辑结构

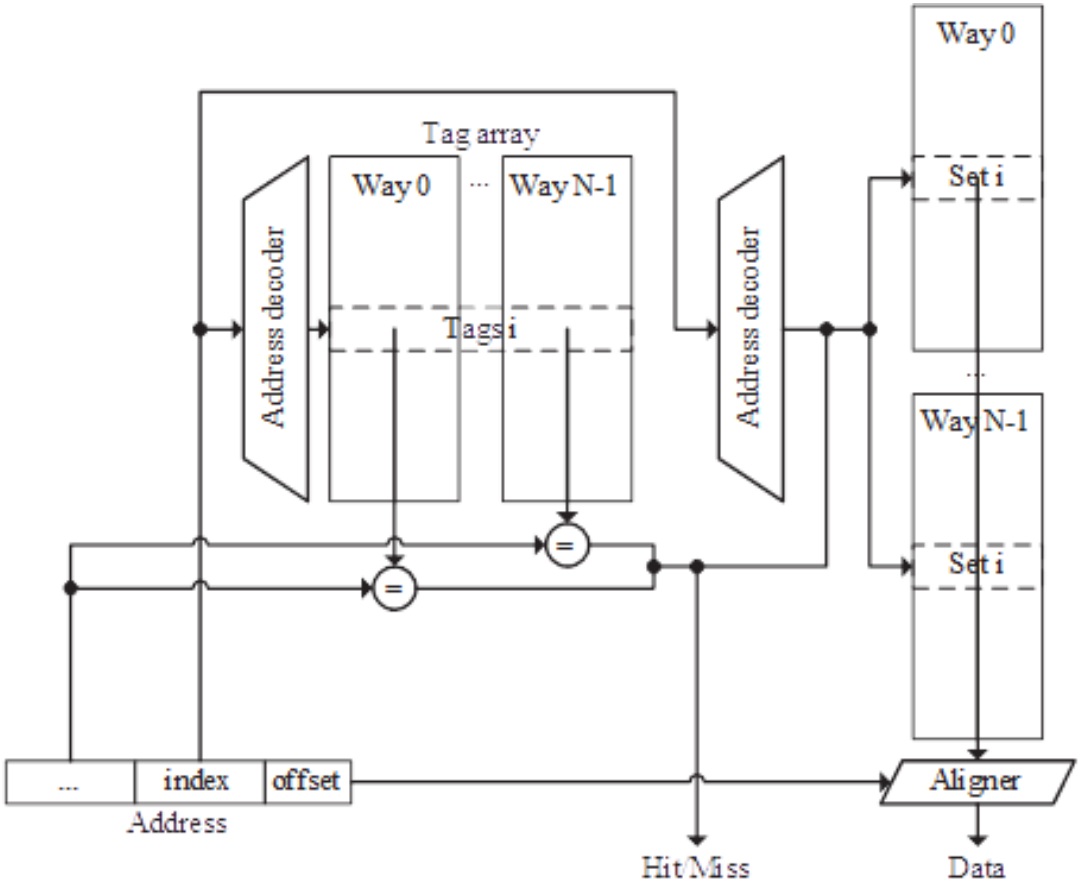


FIGURE 2.3: High-level logical cache organization with serial tag and data array access.

Tag 和数据阵列串行访问的流水线模式

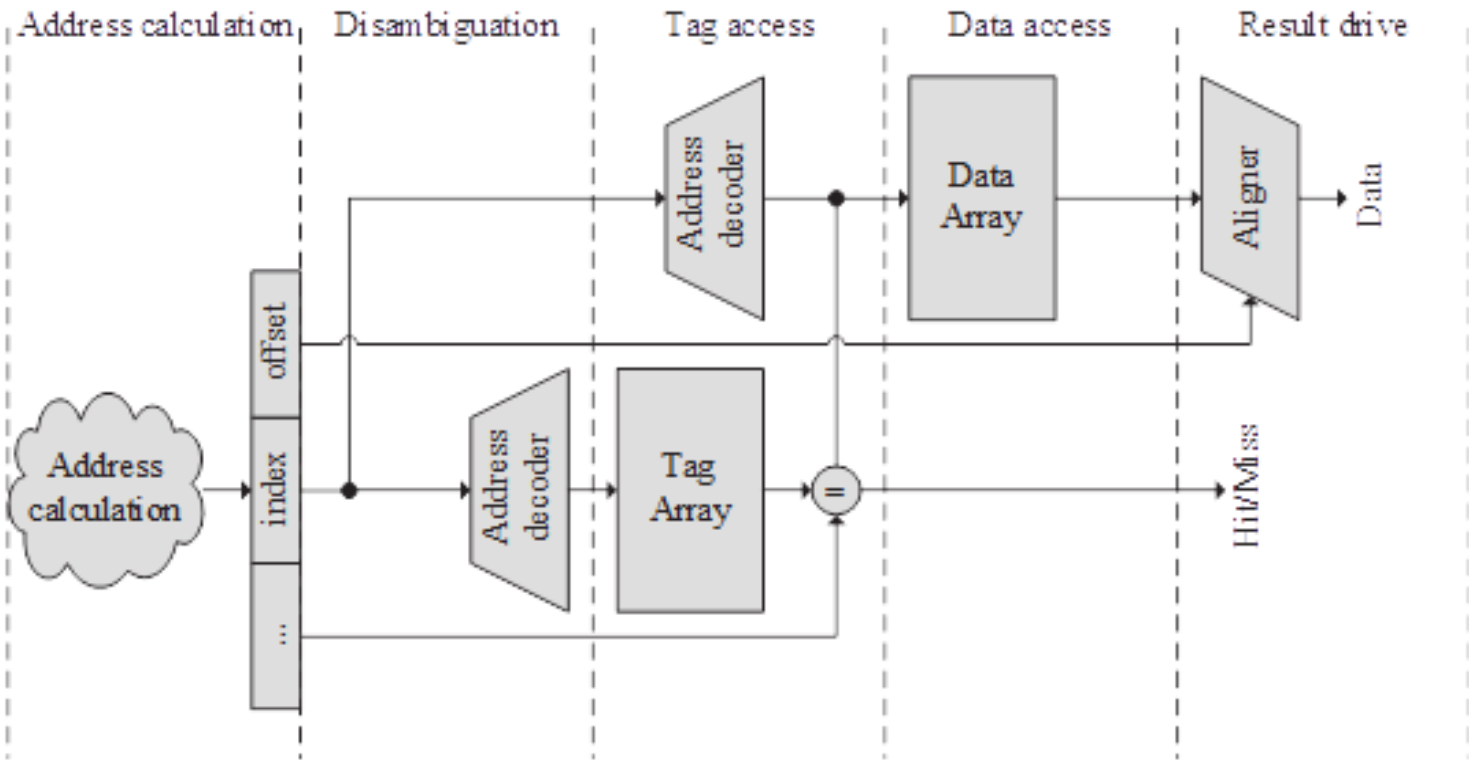
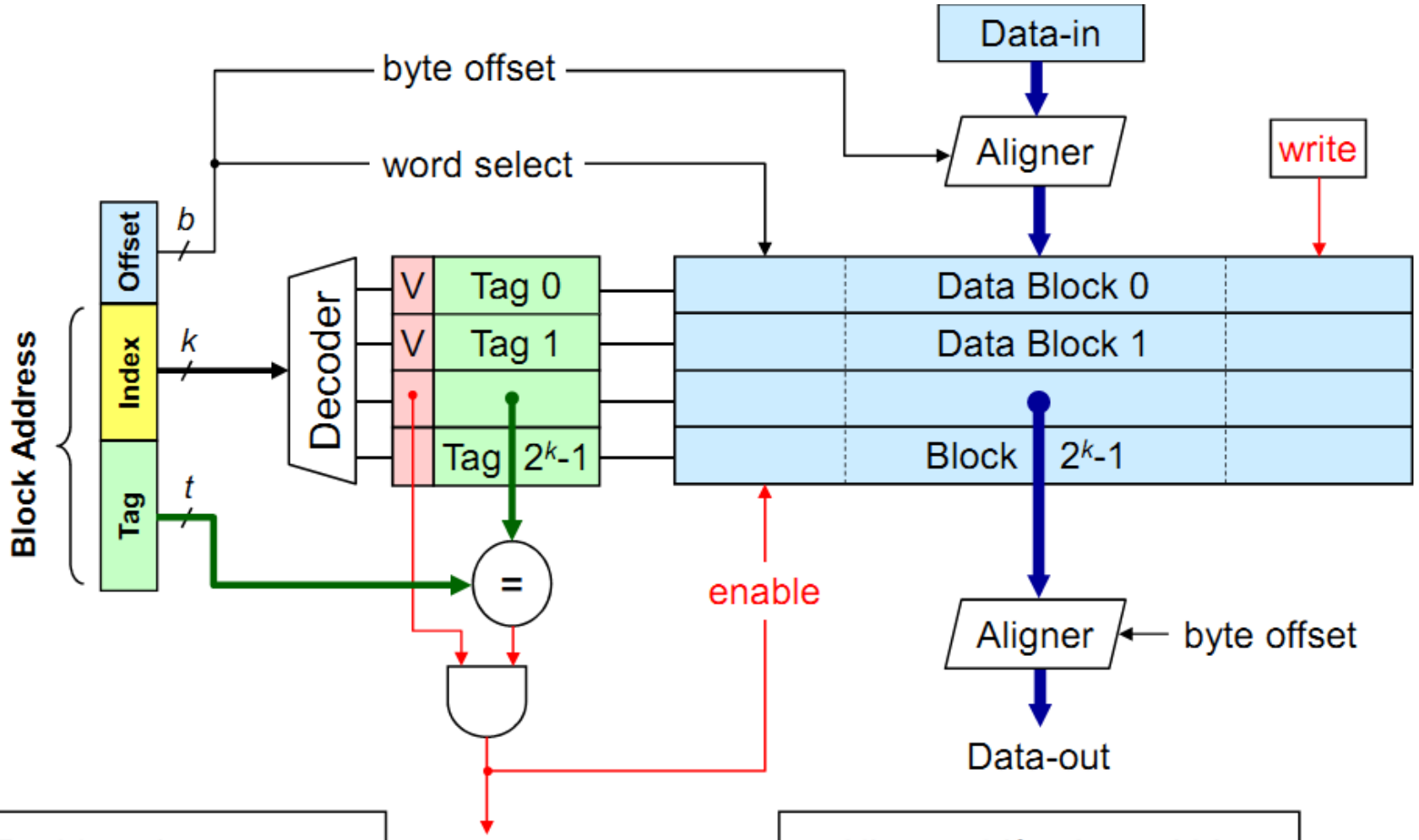
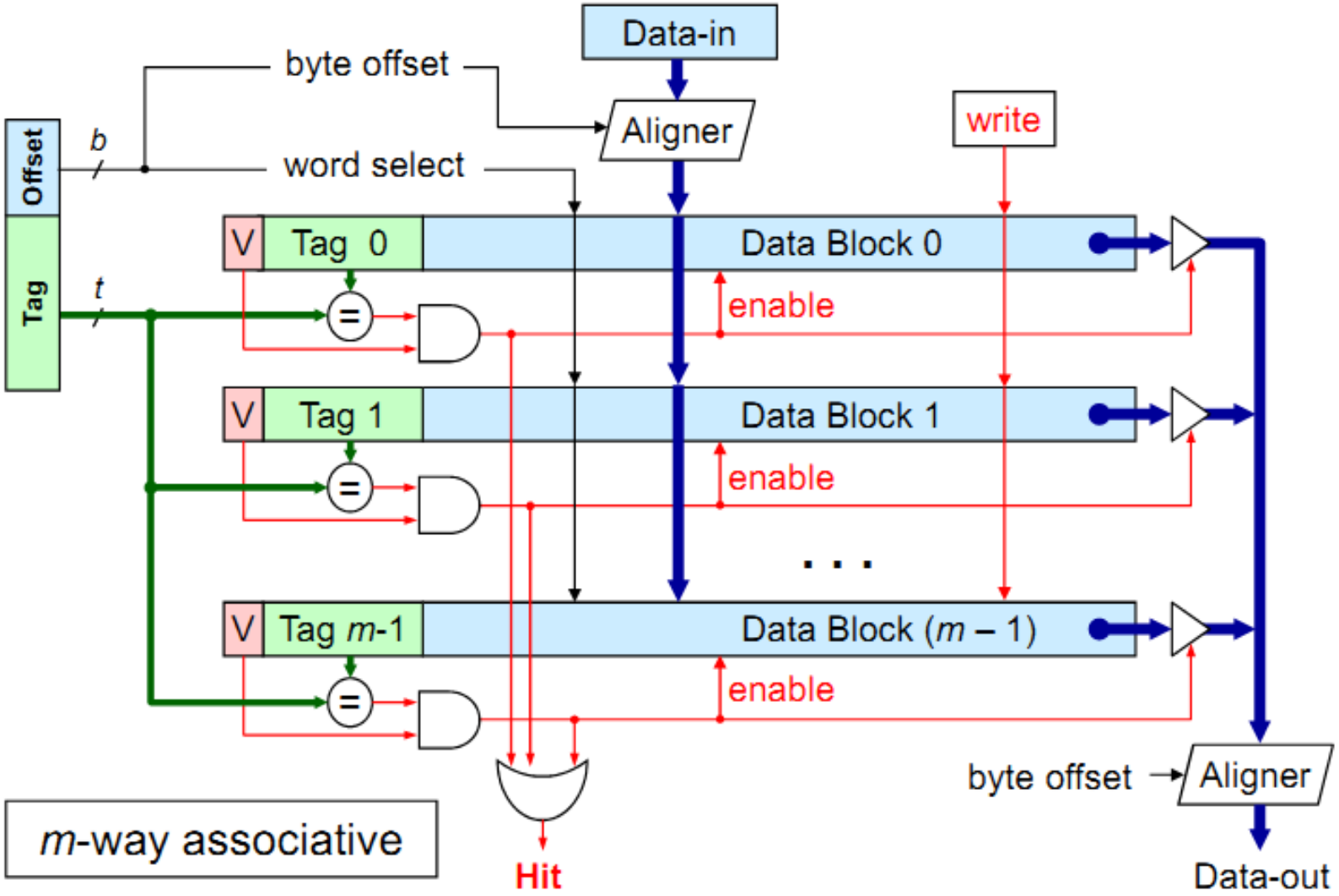


FIGURE 2.4: Serial tag and data array access pipeline.

直接映像Cache查找过程

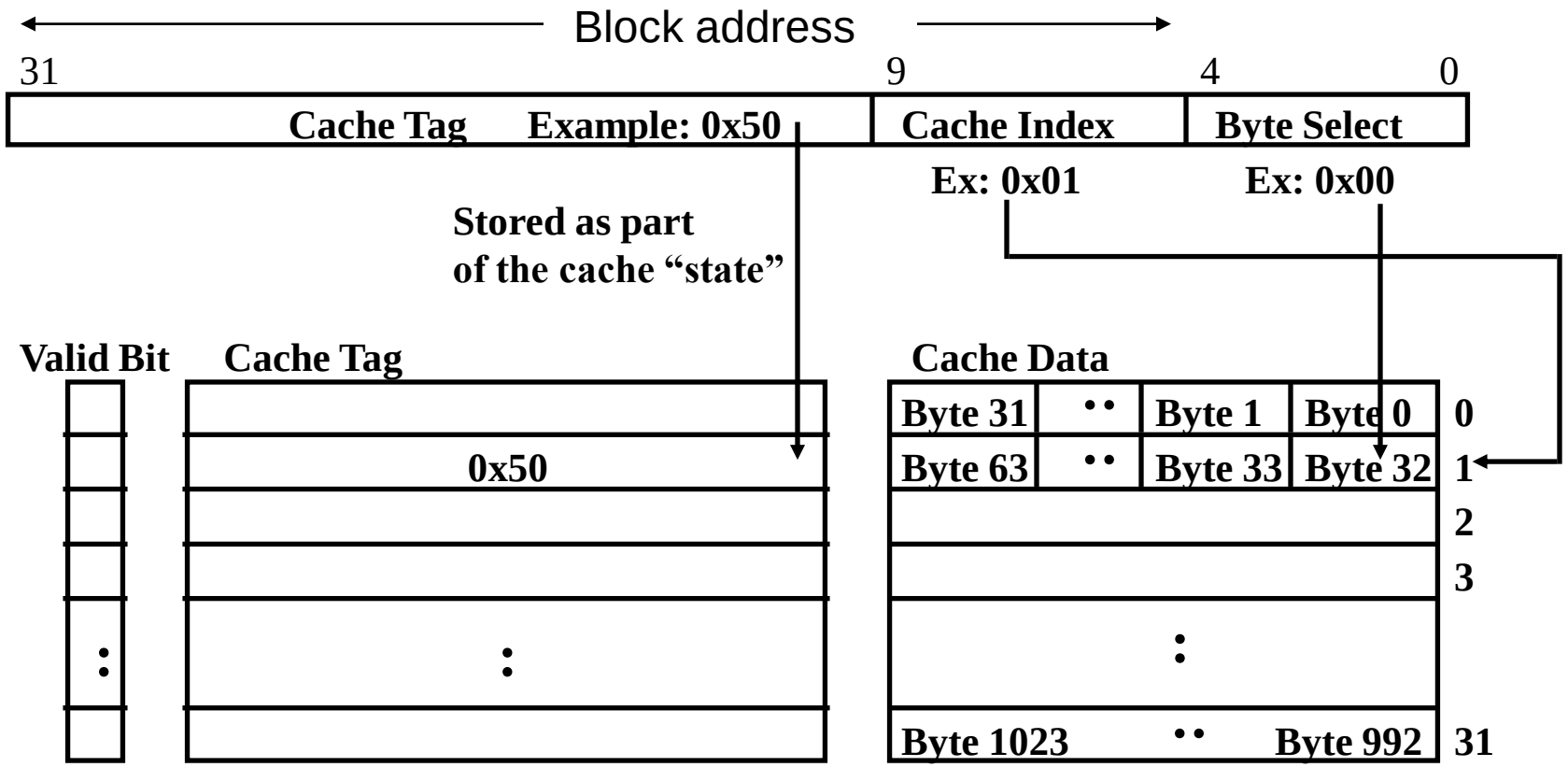


全相联Cache查找过程



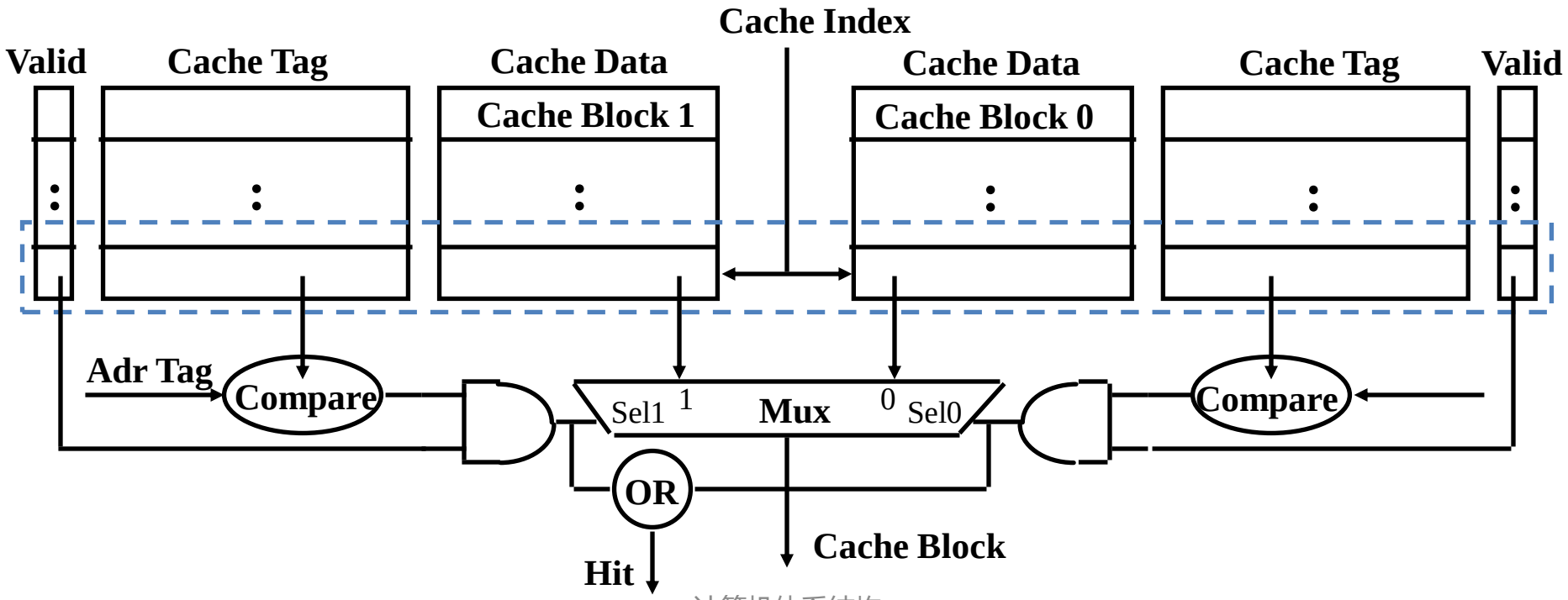
Example: 1 KB Direct Mapped Cache with 32 B Blocks

- **对于容量为 2^N 字节的cache:**
  - 最高(32-N)位部分 为 Cache Tag
  - 最低M位为字节选择位 (Block Size = 2^M)



Example: Set Associative Cache

- **N-way set associative: 每一个cache索引对应N个cache entries**
  - 这N个cache项并行操作
- **Example: Two-way set associative cache**
  - Cache index 选择cache中的一组
  - 这一组中的两块对应的Tags与输入的地址同时比较
  - 根据比较结果选择数据



Q3：替换算法

- **主存中块数一般比cache中的块多，可能出现该块所对应的一组或一个Cache块已全部被占用的情况，这时需强制腾出其中的某一块，以接纳新调入的块，替换哪一块，这是替换算法要解决的问题：**
  - 直接映象，因为只有一块，别无选择
  - 组相联和全相联有多种选择
- **替换方法**
  - 随机法（Random），随机选择一块替换
    - 优点：简单，易于实现
    - 缺点：没有考虑Cache块的使用历史，反映程序的局部性较差，失效率较高
  - FIFO - 选择最早调入的块
    - 优点：简单
    - 虽然利用了同一组中各块进入Cache的顺序，但还是反映程序局部性不够，因为最先进入的块，很可能是经常使用的块
  - 最近最少使用法（LRU) (Least Recently Used)
    - 优点：较好的利用了程序的局部性，失效率较低
    - 缺点：比较复杂，硬件实现较困难

LRU和Random的比较（失效率）

Size	Associativity								
	Two-way			Four-way			Eight-way		
	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

Data Cache misses per 1000 instructions comparing LRU, Random, FIFO replacement for several sizes and associativities. These data were collected for a block size of 64 bytes for the Alpha architecture using 10 SPEC2000 benchmarks. Five are from SPECint2000(gap, gcc, gzip, mcf and perl) and five are from SPECfp2000(applu, art, equake, lucas and swim)

- **观察结果（失效率）**
  - 相联度高，失效率较低。
  - Cache容量较大，失效率较低。
  - LRU 在Cache容量较小时，失效率较低
  - 随着Cache容量的加大，Random的失效率在降低

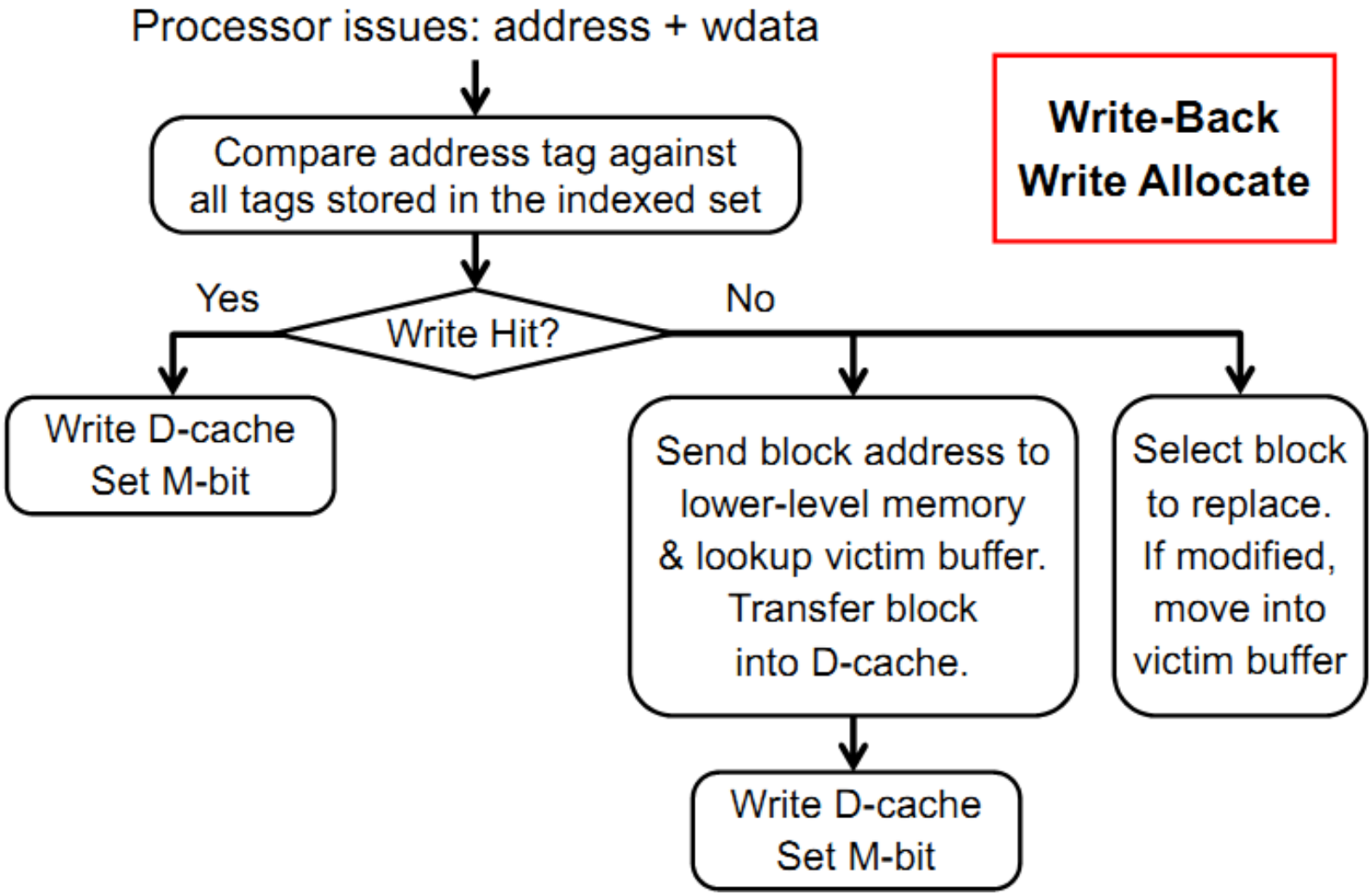
Q4: 写策略

- **程序对存储器读操作占26%， 写操作占9%**
  - 写所占的存储器访问比例　9/(100+26+9) 大约为7%
  - 占访问数据Cache的比例： 9/(26+9) 大约为25%
- **大概率事件优先原则 - 优化Cache的读操作**
- **Amdahl定律：不可忽视“写”的速度**
- **“写”的问题**
  - 读出标识，确认命中后，对Cache写　（串行操作）
  - Cache与主存内容的一致性问题
- **写策略就是要解决: 何时更新主存问题**

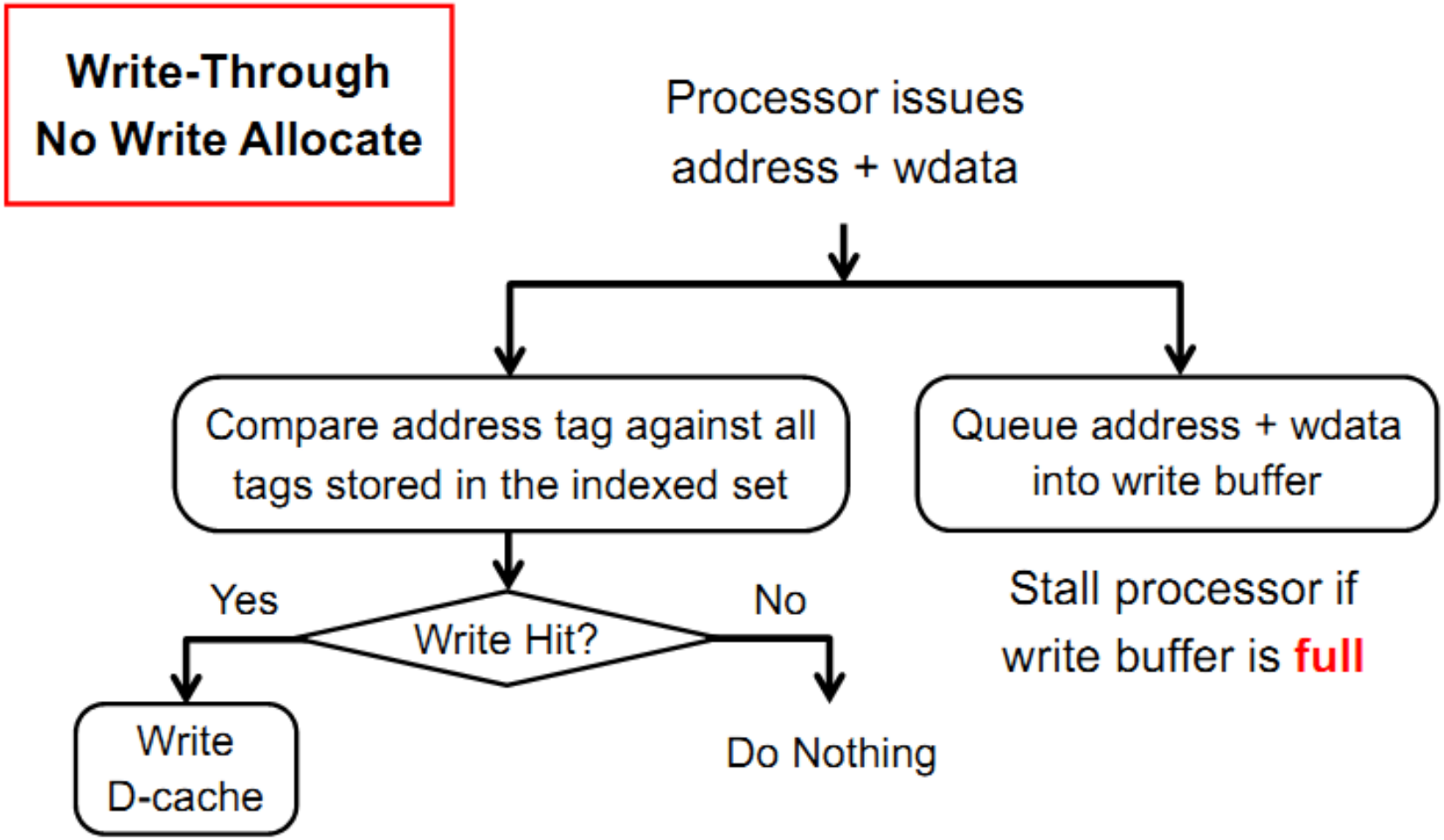
两种写策略

- **写直达法（Write through)**
  - 优点：易于实现，容易保持不同层次间的一致性
  - 缺点：速度较慢
- **写回法 (Write back)**
  - 优点：速度快，减少访存次数
  - 缺点：一致性问题
- **当发生写失效时的两种策略**
  - 按写分配法(Write allocate)：写失效时，先把所写单元所在块调入Cache，然后再进行写入，也称写时取（Fetch on Write)方法
  - 不按写分配法（no-write allocate): 写失效时，直接写入下一级存储器，而不将相应块调入Cache，也称绕写法（Write around)
  - 原则上以上两种方法都可以应用于写直达法和写回法，一般情况下
    - Write Back 用Write allocate
    - Write through 用no-write allocate

Write-back, Write Allocate



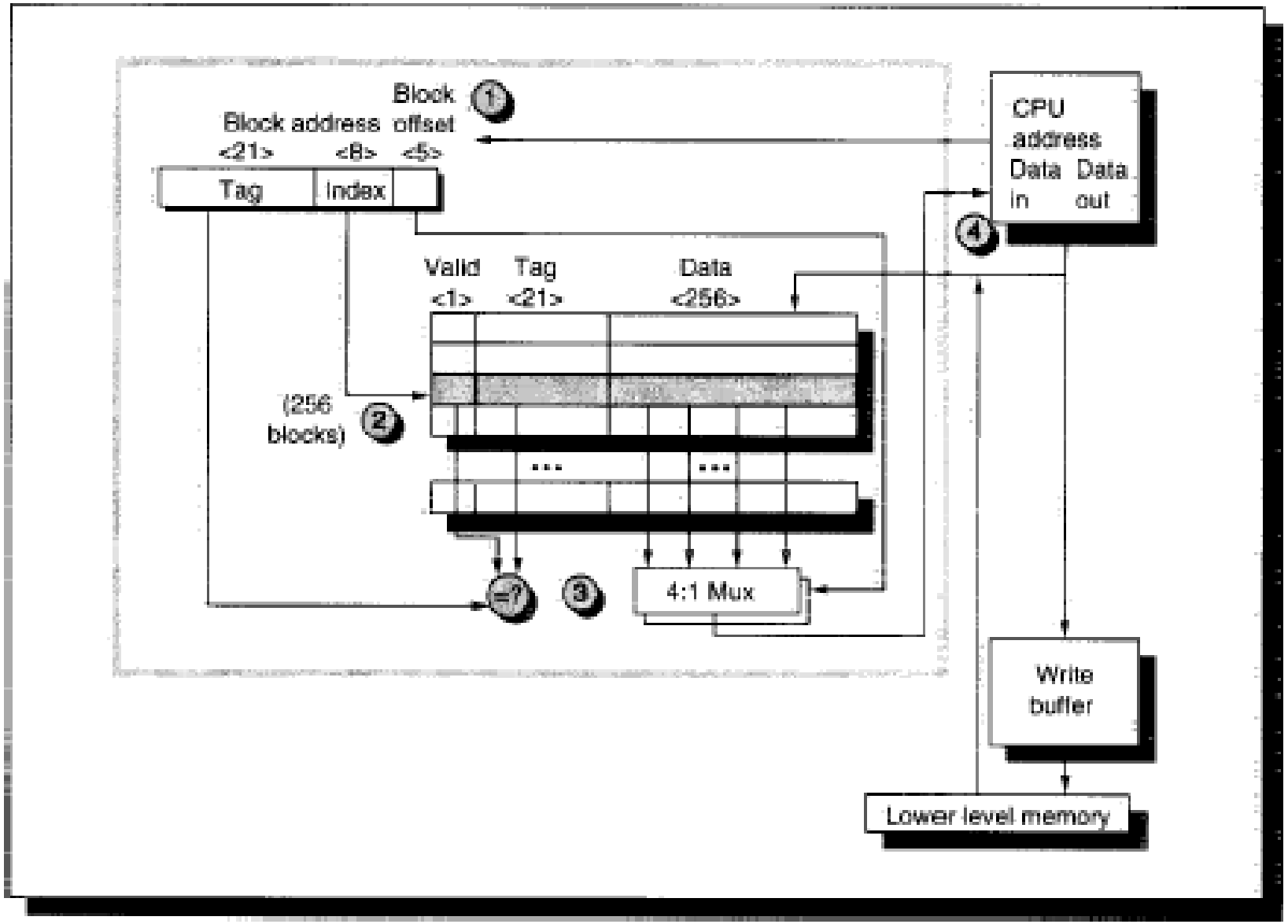
Write-Through, No Write Allocate



Alpha AX 21064 Cache结构（数据Cache)

- **基本技术特性**
  - 容量 8KB，Block为32Bytes，共256个Blocks，每个字为8个字节
  - 直接映象
  - 写直达法，写失效时，no-write allocate 方法
  - 写缓冲：4个blocks
- **21064物理地址34位**
  - 21位tag##8位index ##5位块内偏移　怎么算出来的？
- **Cache命中的步骤**
  - 读命中
  - 写命中
- **Cache失效**
  - Cache向CPU发暂停信号
  - 块传送，21064 Cache与下一级存储器之间数据通路16字节，传送全部32字节需要10个cycles

Alpha AX 21064 Cache结构（数据Cache)



Alpha 21264 Data Cache

- **基本技术特征**
  - Cache size: 64Kbytes
  - Block size: 64-bytes
  - Two-way 组相联
  - Write back, 写失效时， write allocate
- **21264 48-bits 虚拟地址，虚实映射为44-bits的物理地址，也支持43-bits虚拟地址，虚实映射为41-bits的物理地址**
  - 29位tags##9位index##6位 Block offset

怎么算出来的?

Alpha 21264 Data Cache

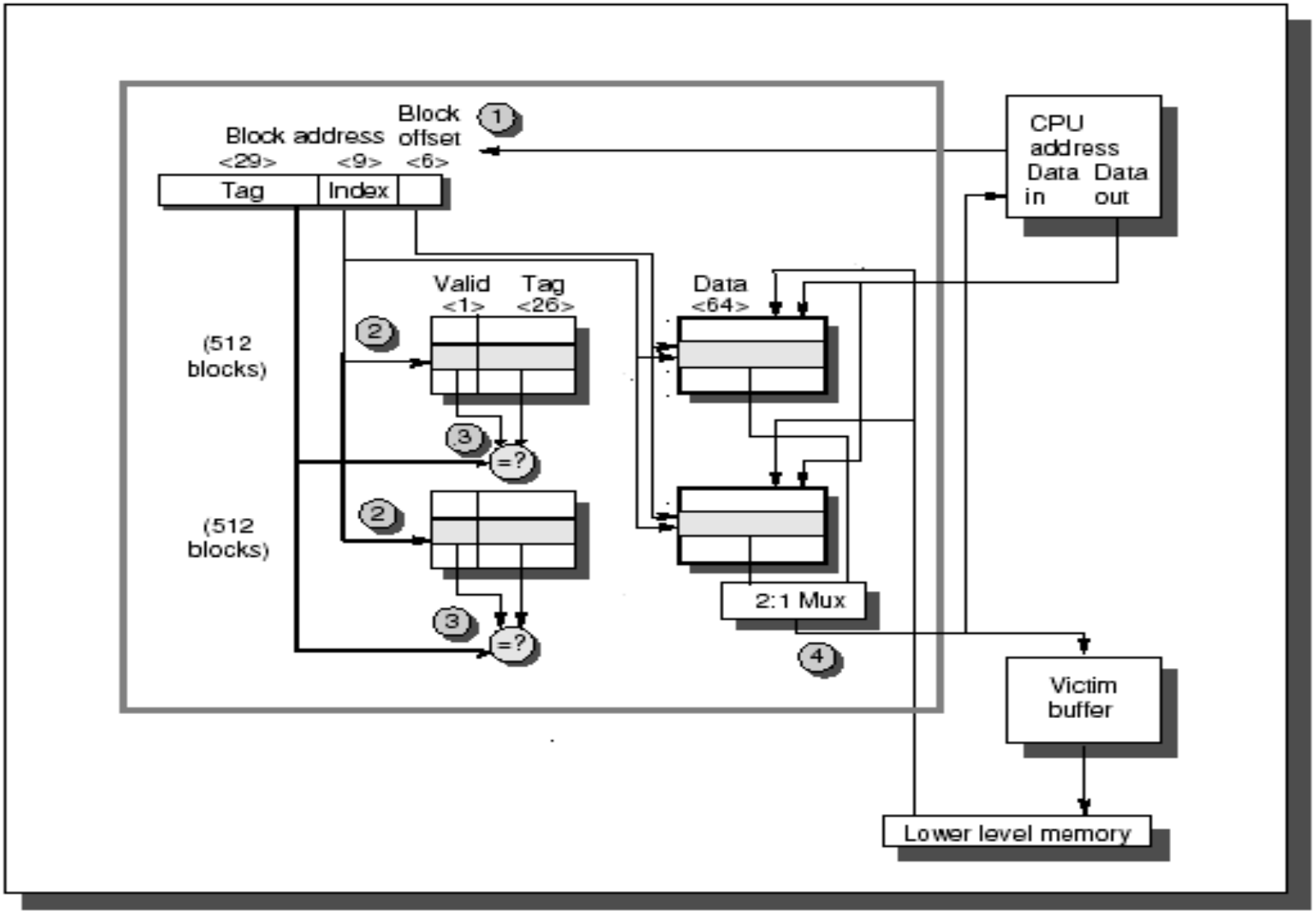


FIGURE 5.7 The organization of the data cache in the Alpha 21264 microprocessor.

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - David Patterson (UCB)
  - Chenxi Zhang (Tongji)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**