计算机系统结构

第五课：存储层次 I

王韬

wangtao@pku.edu.cn

http://ceca.pku.edu.cn/wangtao

2018

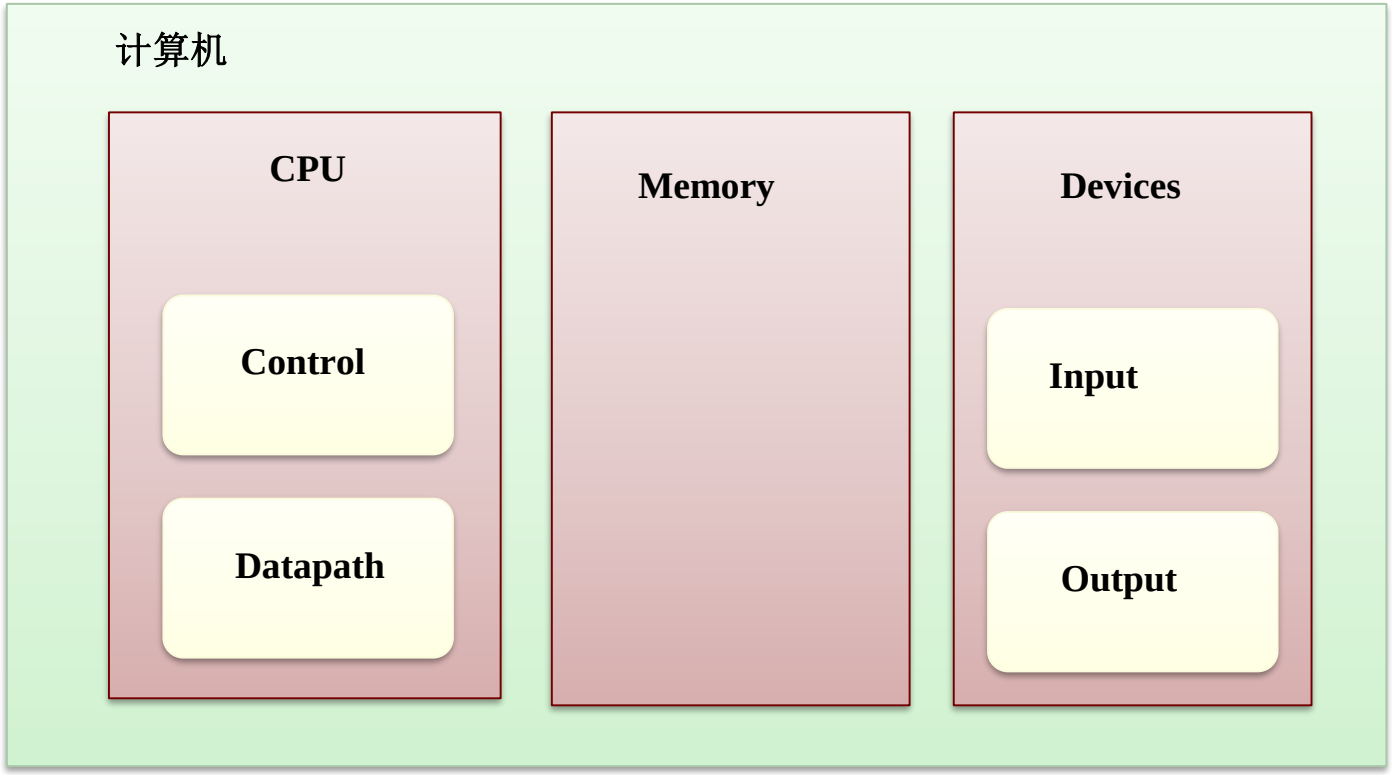
第五课：存储层次 I

- 存储层次概论
- 高速缓存基础
- 提高高速缓存性能

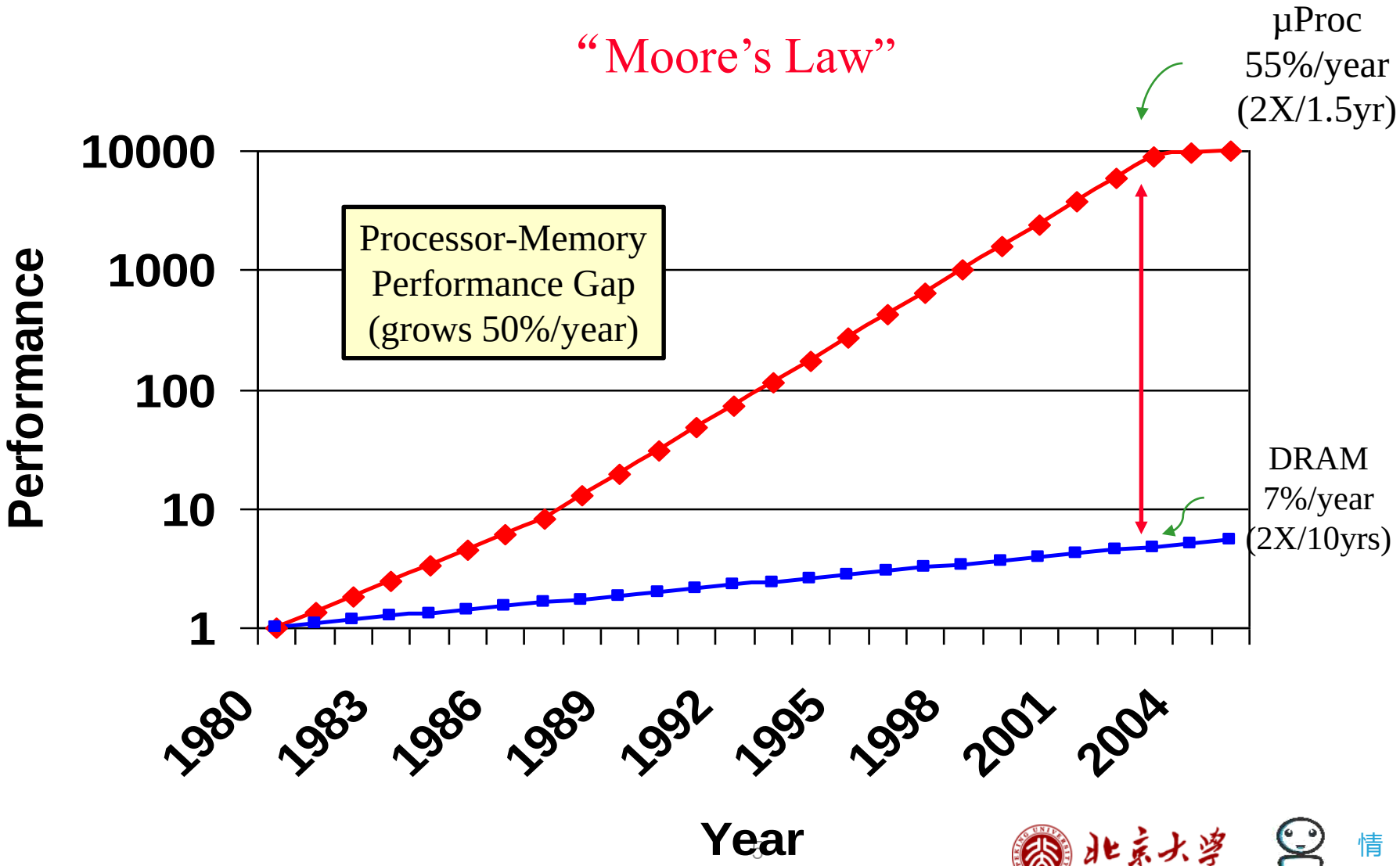
第五课：存储层次 I

- 存储层次概论
- 高速缓存基础
- 提高高速缓存性能

复习：计算机中的主要部分

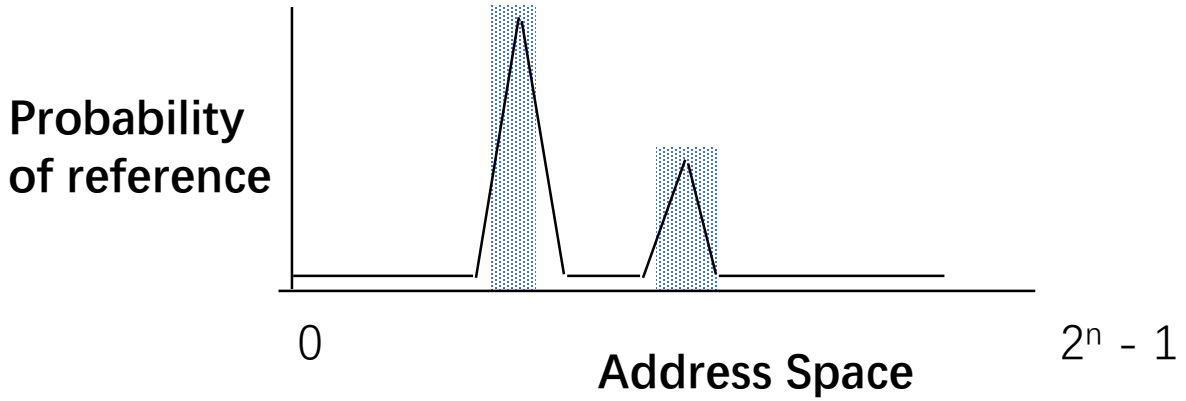


处理器与内存之间的性能差距



存储层次的目标

- 期望：又大，又快
- 事实：大的存储慢，快的存储小
- 我们如何能够提供一种存储，让人们感觉在大多数情况下足够大、便宜，但是又快呢？
- 局部性原理的背后 – 在任何一个时刻，程序只访问一小块地址空间



复习：局部性原理

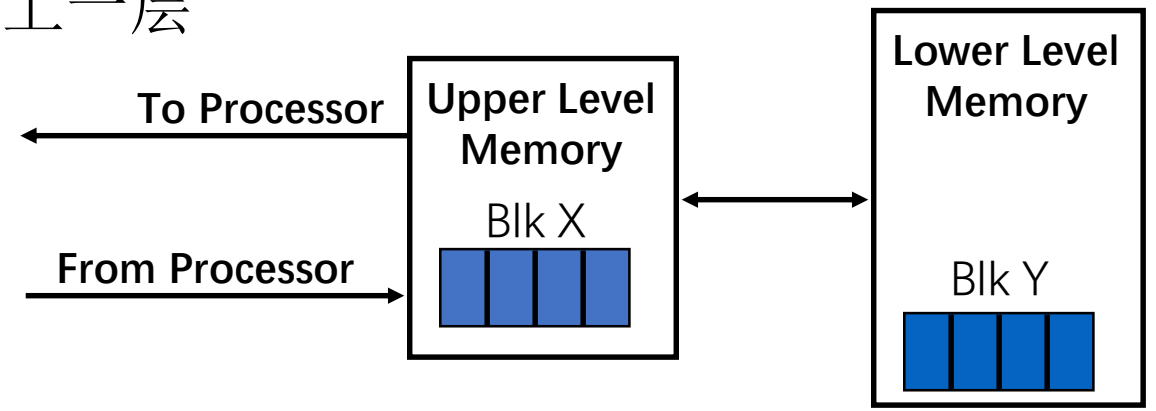
- 时间局部性
  - 近期被访问的位置，很可能会再次被访问
- 空间局部性
  - 近期被访问的位置的“邻居位置”，也很有可能会被访问

分层次处理问题

- 较上层存储（离处理器较近）
  - 更快、更小，更贵，离处理器更近
- 时间局部性

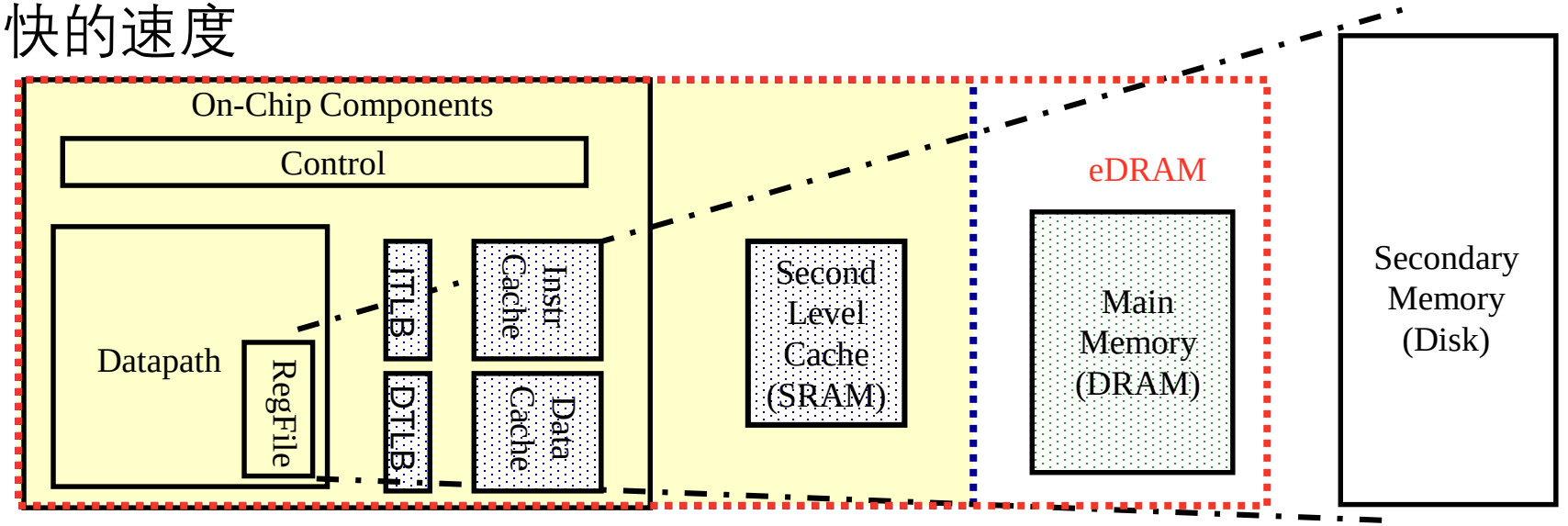
  => 将最近访问过的数据保持在离处理器较近的位置
- 空间局部性

  => 将含有连续数据的块同时搬移到上一层



一个典型的存储结构

- 通过利用局部性原理
  - 对用户展现尽可能大的存储空间（利用最便宜的技术）
  - 同时提供最快的速度



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

存储层次所用的技术

- 用**SRAM**实现缓存以提高速度
  - 低密度、高功耗、昂贵、快速
  - 静态：只要不断电，内容就会一直存在
- 用**DRAM**实现内存来提高容量
  - 高密度、低功耗、便宜、慢
  - 动态：必须定时“刷新”（~ 8 ms）
    - 1% to 2% of the active cycles of the DRAM
  - 分两部分输入地址（行和列）
    - RAS or Row Access Strobe：行地址解码
    - CAS or Column Access Strobe：列地址解码

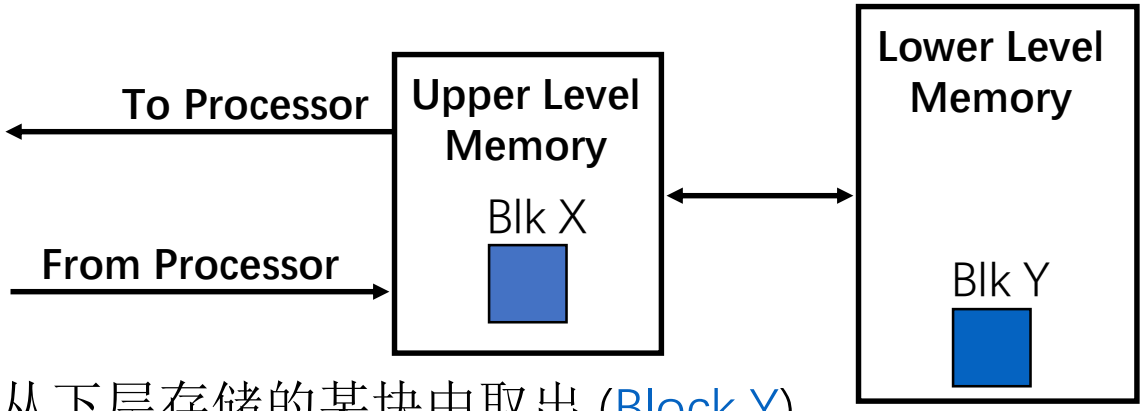
存储性能衡量维度

- 延迟：访问一个word的时间
  - *访问时间 Access time*: 从请求发出，到请求完成之间的时间
  - *周期时间 Cycle time*: 两个请求之间间隔的最短时间
  - 通常周期时间 > 访问时间
- 带宽：单位时间内可以访问多大的数据量
  - 数据通道的宽度 * 每秒能够访问的次数

- *容量：* DRAM to SRAM *4 to 8*
- *造价/周期时间：* SRAM to DRAM *8 to 16*

命中与失效

- 命中 Hit: 数据在某层（Upper Level）存储的某块中能够被找到 (Block X)
  - 命中率: 能够命中的比率
  - 命中时间: 访问这层（Upper Level）存储的时间－RAM访问时间 +判断是否命中的时间

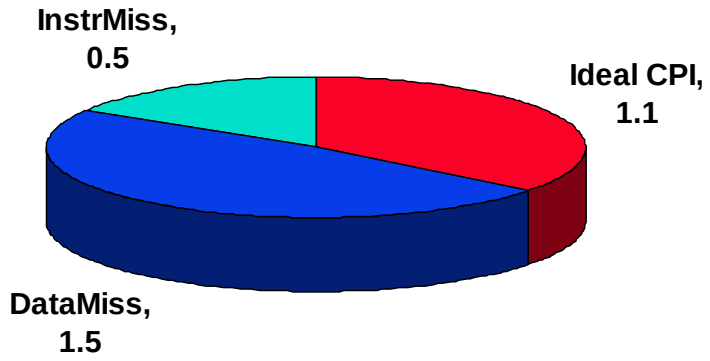


- 失效 Miss: 数据需要从下层存储的某块中取出 (Block Y)
  - 失效率 = 1 – 命中率
  - 失效损失 Miss Penalty: 替换某层（Upper Level）存储某块的时间 + 将此块发送给处理器的时间
  - 命中时间 << 失效损失

存储性能对于系统性能的影响

- 假设一个处理器有如下特性
  - ideal CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control

  并且有10%的数据失效率，失效损失是50个周期，求当前的CPI：
- CPI = ideal CPI + average stalls per instruction
  = 1.1 (cycle) + (0.30 (datamemops/instr)
  x 0.10 (miss/datamemop)
  x 50 (cycle/miss))
  = 1.1 (cycle) + 1.5 (cycle) = 2.6 (cycle)
- 处理器58%的时间用来等待存储（失效）！
  - (2.6-1.1)/2.6
- 1%的指令失效会再给CPI增加0.5!
  - 0.01 x 50



又快、空间又大

- DRAM慢，但是便宜，且密度高
  - 为用户提供很好的“大内存”
- SRAM快，但是贵、密度低
  - 为用户提供快速的访问
- 两种不同的局部性
  - 时间局部性：近期被访问的位置，很可能会再次被访问
  - 空间局部性：近期被访问的位置的“邻居位置”，也很有可能会被访问
- 通过利用局部性原理
  - 对用户展现尽可能大的存储空间
  - 同时提供最快的速度

管理存储层次

- 寄存器 ↔ 内存
  - 通过编译器 (程序员?)
- 高速缓存 ↔ 内存
  - 通过缓存控制器硬件
- 内存 ↔ 磁盘
  - 通过操作系统（虚拟内存）
  - 通过程序员 (文件)

第五课：存储层次 I

- 存储层次概论
- 高速缓存基础
- 提高高速缓存性能

高速缓存 Cache

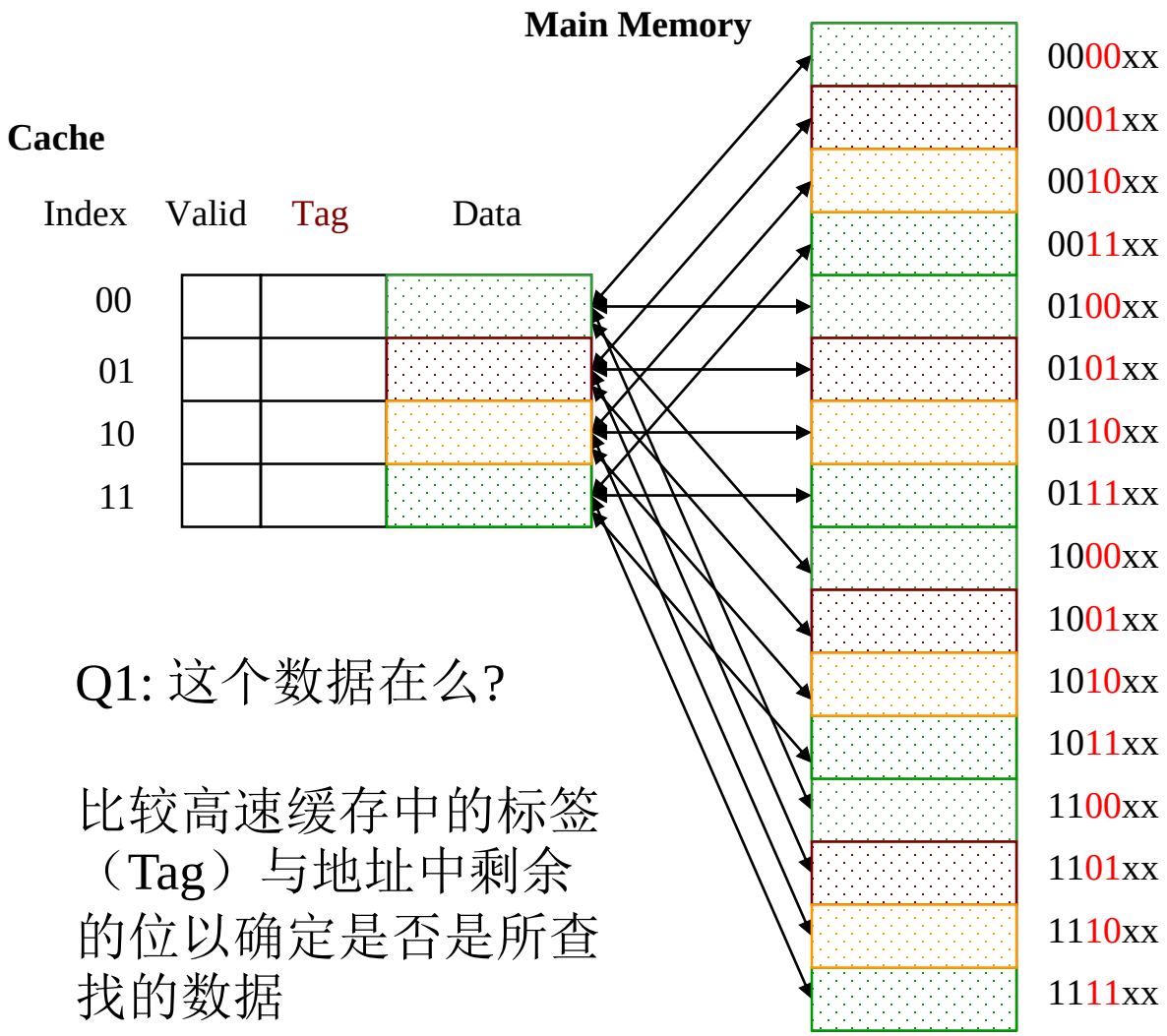
- 快而小（SRAM），离处理器近

- 两个问题 (硬件中):
  - Q1: 如何能够知道一份数据是否在高速缓存中？
  - Q2: 如果在，怎么能够找到它（寻址）？

- 直接映射
  - 对于任一个在较低层的数据，在上一层中都能找到其唯一可能的位置 – 于是较低层很多的数据必须共享上一层的同一位置（为什么？）
  - 地址映射方式:

(block address) modulo (# of blocks in the cache)

直接映射举例：假设一个缓存块为一个Word



地址最低两位没有被使用：
定义一个32位word中的字节位置

Q1: 这个数据在么?

比较高速缓存中的标签（Tag）与地址中剩余的位以确定是否是所查找的数据

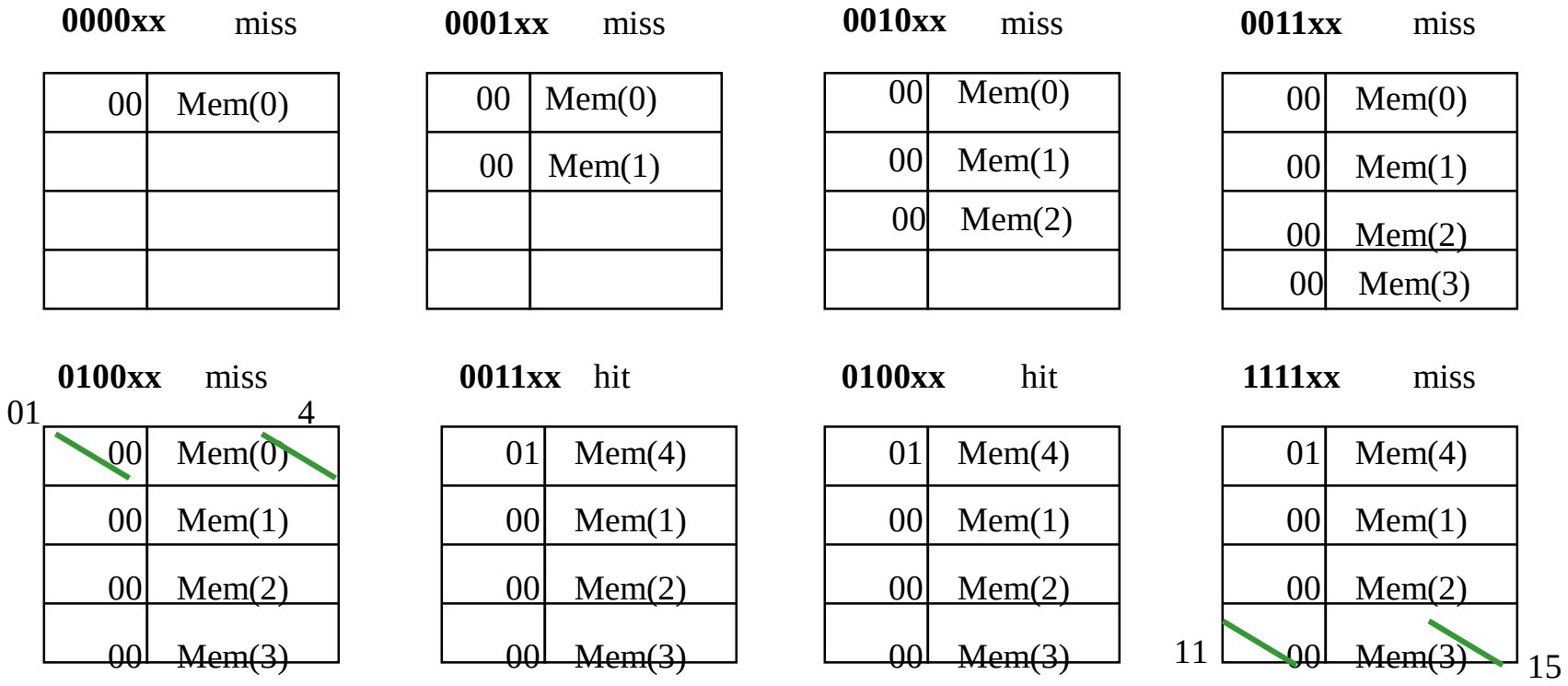
Q2: 如何找到数据？

用地址中接下来的两个低位作为索引值来查找高速缓存块 (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

直接映射举例2：设开始时缓存中“无内容”

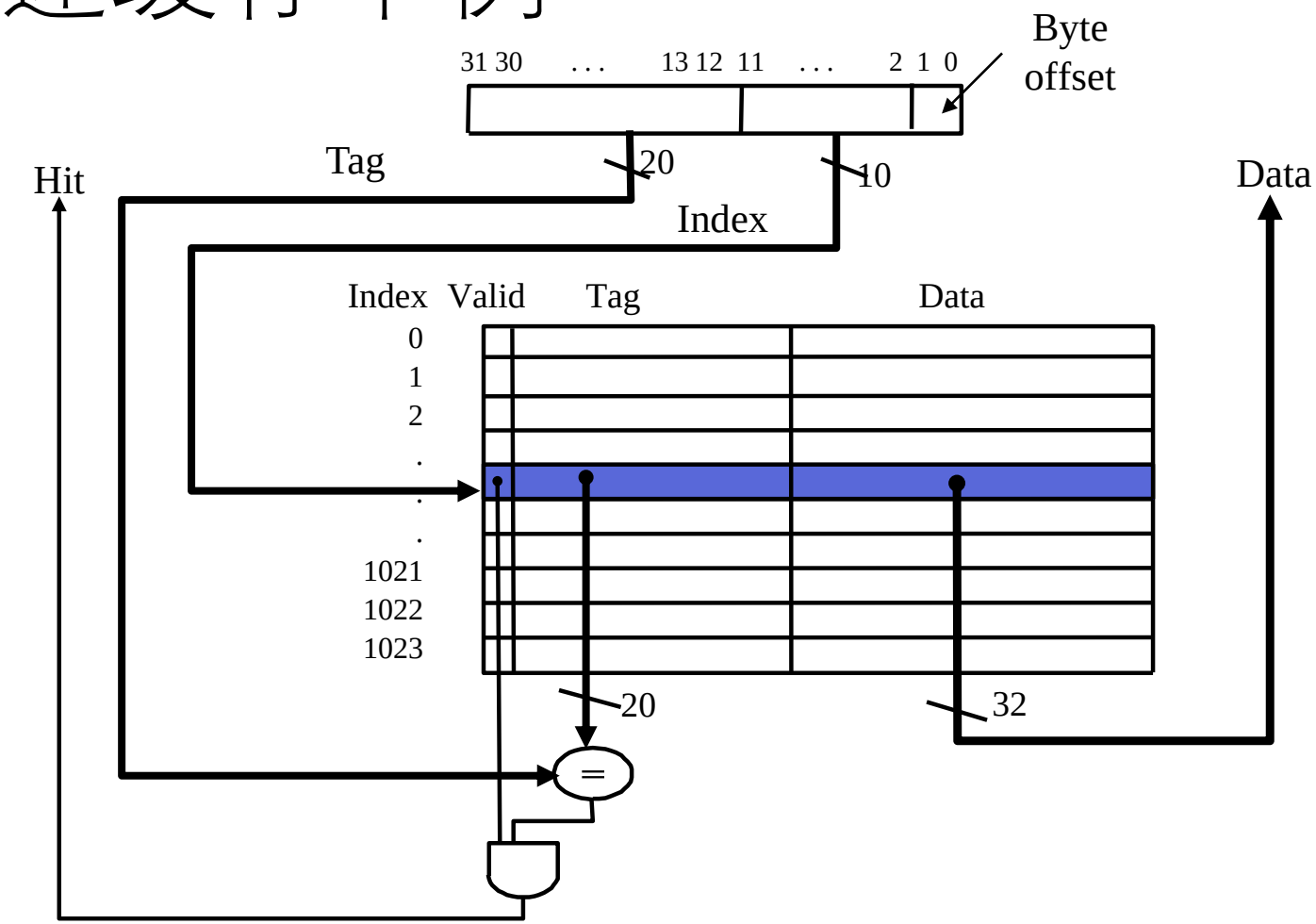
- 考虑访问Word地址（非Byte地址）：0 1 2 3 4 3 4 15



8 requests, 6 misses

MIPS直接映射高速缓存举例

- 每个缓存块为一个word，缓存容量 = 1K words



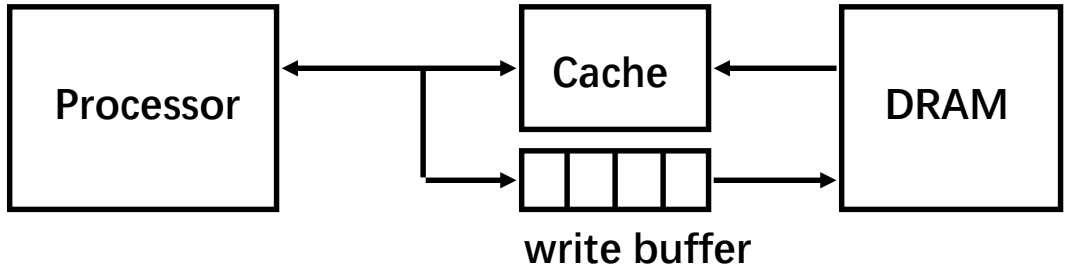
这种情况下，我们能够利用到哪种局部性？

高速缓存命中时的处理

- 读命中 (I$ and D$)
  - 完成任务！
- 写命中 (D$ only)
  - 允许高速缓存与内存存在不一致的情况（write-back）
    - 仅将数据写入到高速缓存块中；当这个缓存块下次被“弹出”的时候，写回（write-back）到下一层次中
    - 需要为每个缓存块标志一个脏（dirty）位，表明这个块是否与下一层次中的内容不一致
  - 要求高速缓存与内存一致的情况（write-through）
    - 总是将数据同时写入到高速缓存块与下一层次中
    - 注意：下一层次的写速度比较慢，所以性能较低，可以采用写缓冲buffer来缓解此问题，只有在缓冲满的时候才需要暂停（stall）

Write-Through缓存的写缓冲

- 处理器与内存中间的写缓冲
  - 处理器：将数据同时写入缓存和写缓冲
  - 内存控制器：将写缓冲的内容写入到内存中
- 写缓冲就是一个FIFO
  - 典型的条目数：4



高速缓存失效时的处理

- 读失效 (I$ and D$)
  - 暂停整个处理器流水线，从存储层次下一层取出相应数据块，将其装入缓存（可能会涉及到块替换），并把需要读取的word发给处理器，最后继续流水线的执行
- 写失效 (D$ only)
  - Write allocate: 暂停整个处理器流水线，从存储层次下一层取出相应数据块，将其装入缓存（可能会涉及到块替换），将处理器所需要写的word写到缓存中，最后继续流水线的执行
  - No-write allocate方法:不写入缓存，直接写入内存

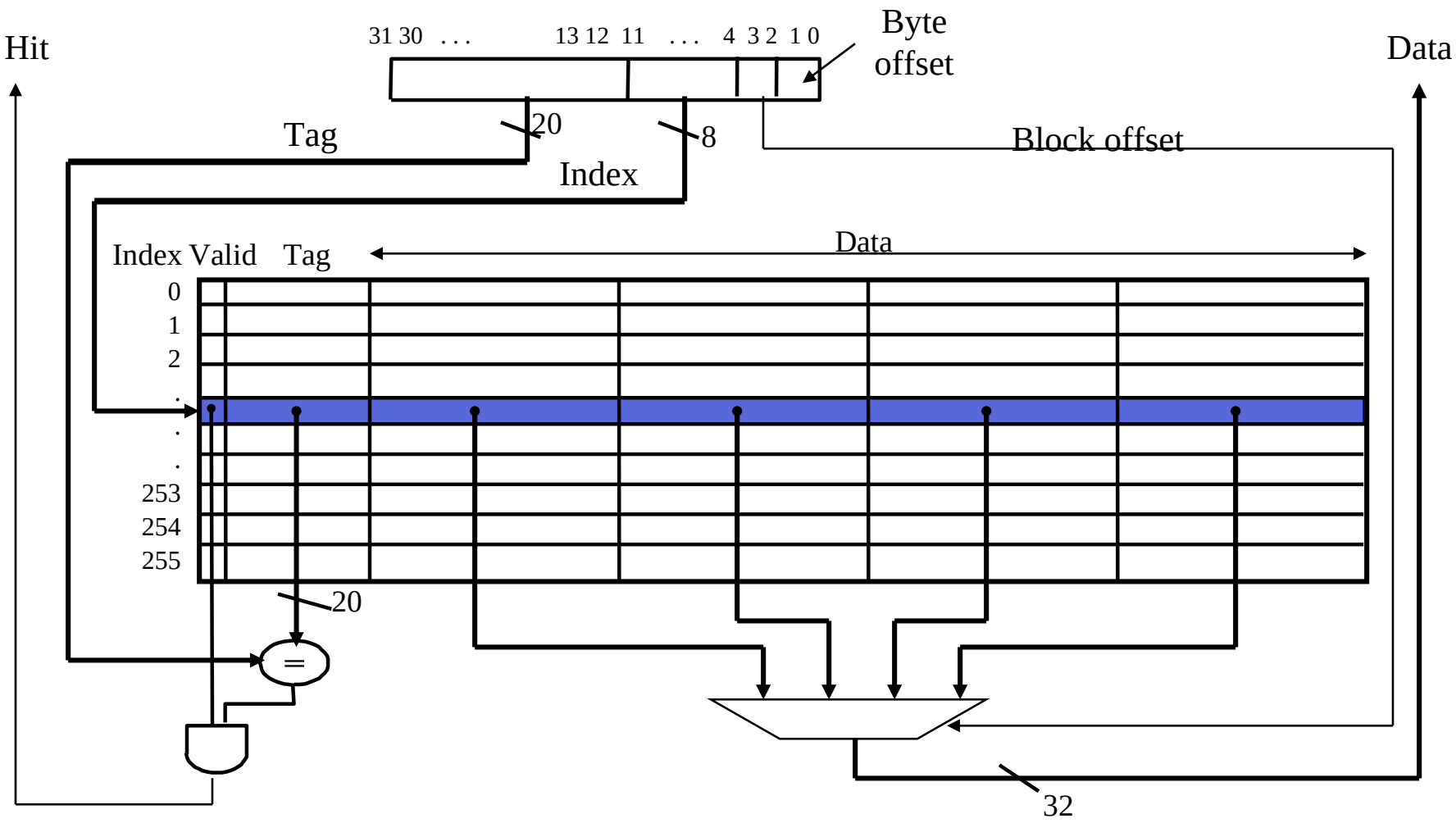
第五课：存储层次 I

- 存储层次概论
- 高速缓存基础
- 提高高速缓存性能

更大的缓存块（示例）

- 每个块有4个word，缓存容量 = 1K words

这种情况下，我们能够利用到哪种局部性？



利用空间局部性

- 让一个块含有2个word：0 1 2 3 4 3 4 15

0000xx miss

00	Mem(1)	Mem(0)

0001xx hit

00	Mem(1)	Mem(0)

0010xx miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

0011xx hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

0100xx miss

01 5 4

~~00~~	~~Mem(1)~~	~~Mem(0)~~
00	Mem(3)	Mem(2)

0011xx hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

0100xx hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

1111xx miss

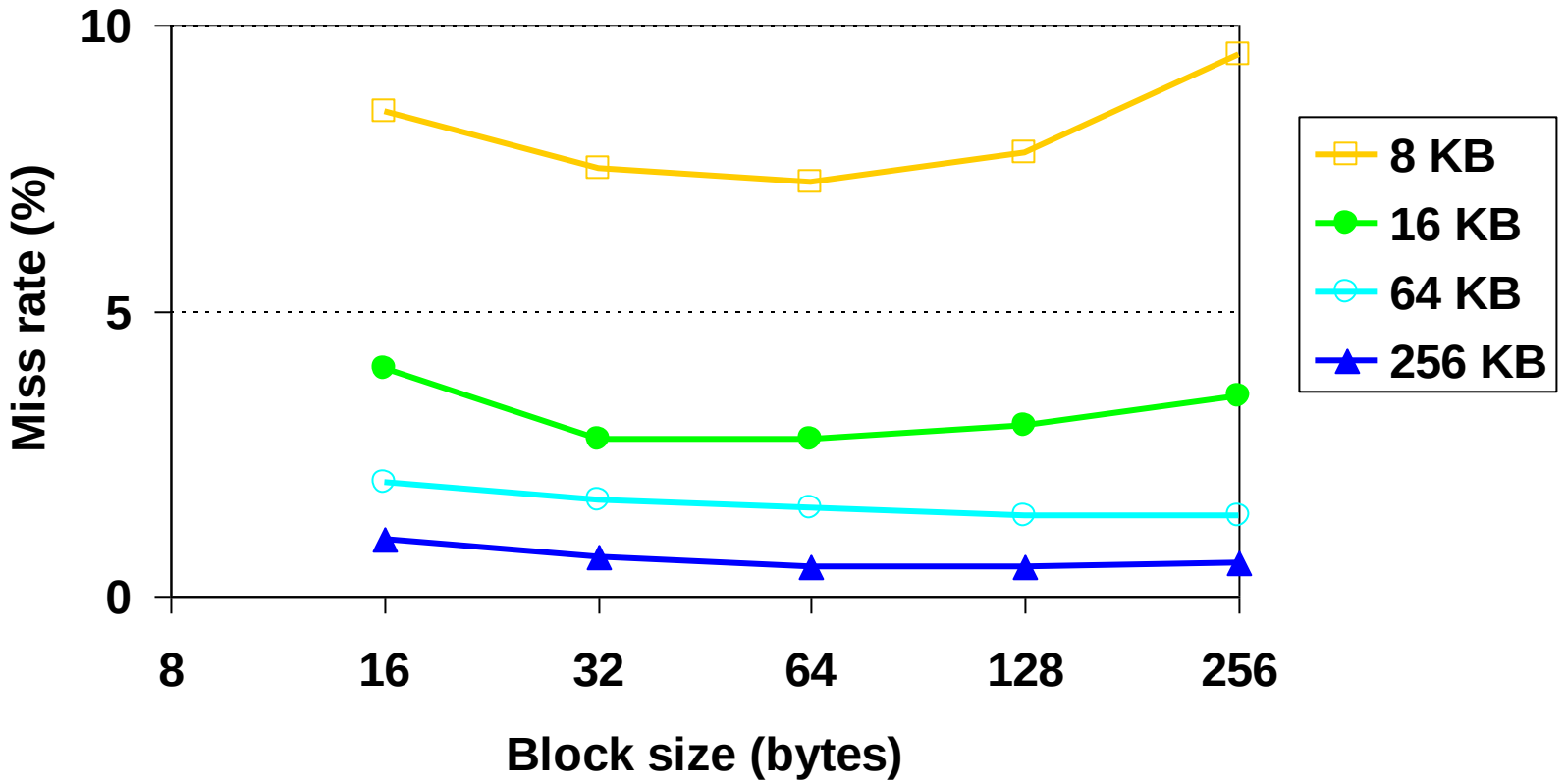
11 15 14

~~01~~	~~Mem(5)~~	~~Mem(4)~~
~~00~~	~~Mem(3)~~	~~Mem(2)~~

8 requests, 4 misses

与第19页的情况比较？

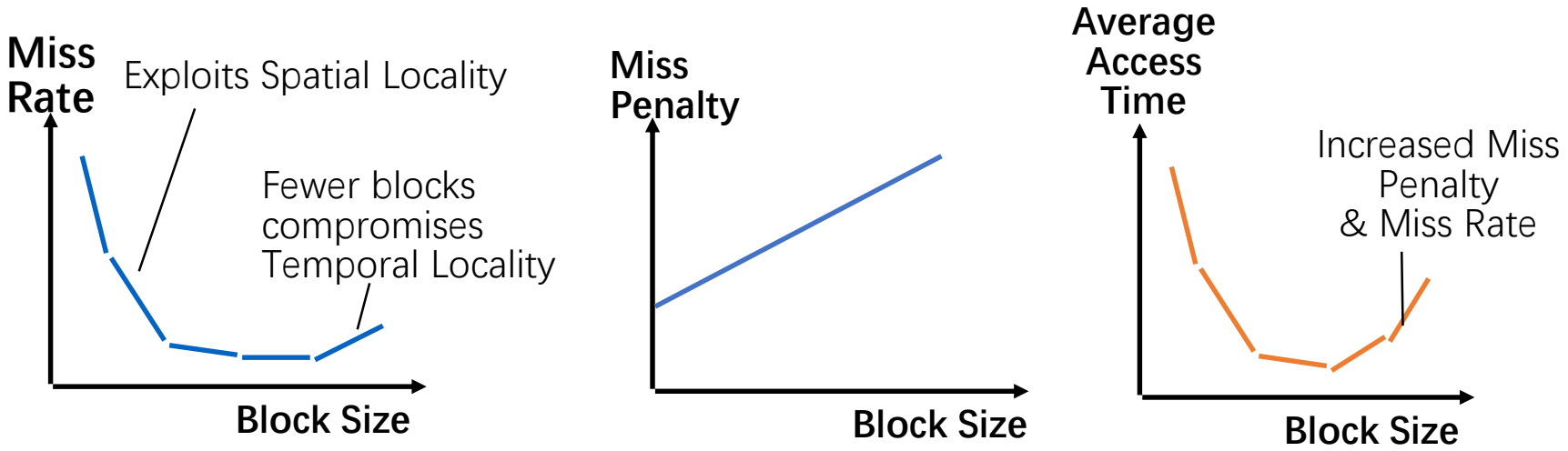
失效率 v.s. 缓存块大小 v.s. 缓存容量



- 当缓存块大小相对整个缓存容量太大时，由于块数目减少，增大了冲突失效，整体失效率会提高

缓存块大小的权衡

- 大的缓存块，意味着大的失效损失
  - 读取块中第一个word的延时+传输整个块剩下部分的时间
- 大的缓存块，能够更好地利用空间局部性，但是
  - 当缓存块大小相对整个缓存容量太大时，整体失效率会提高



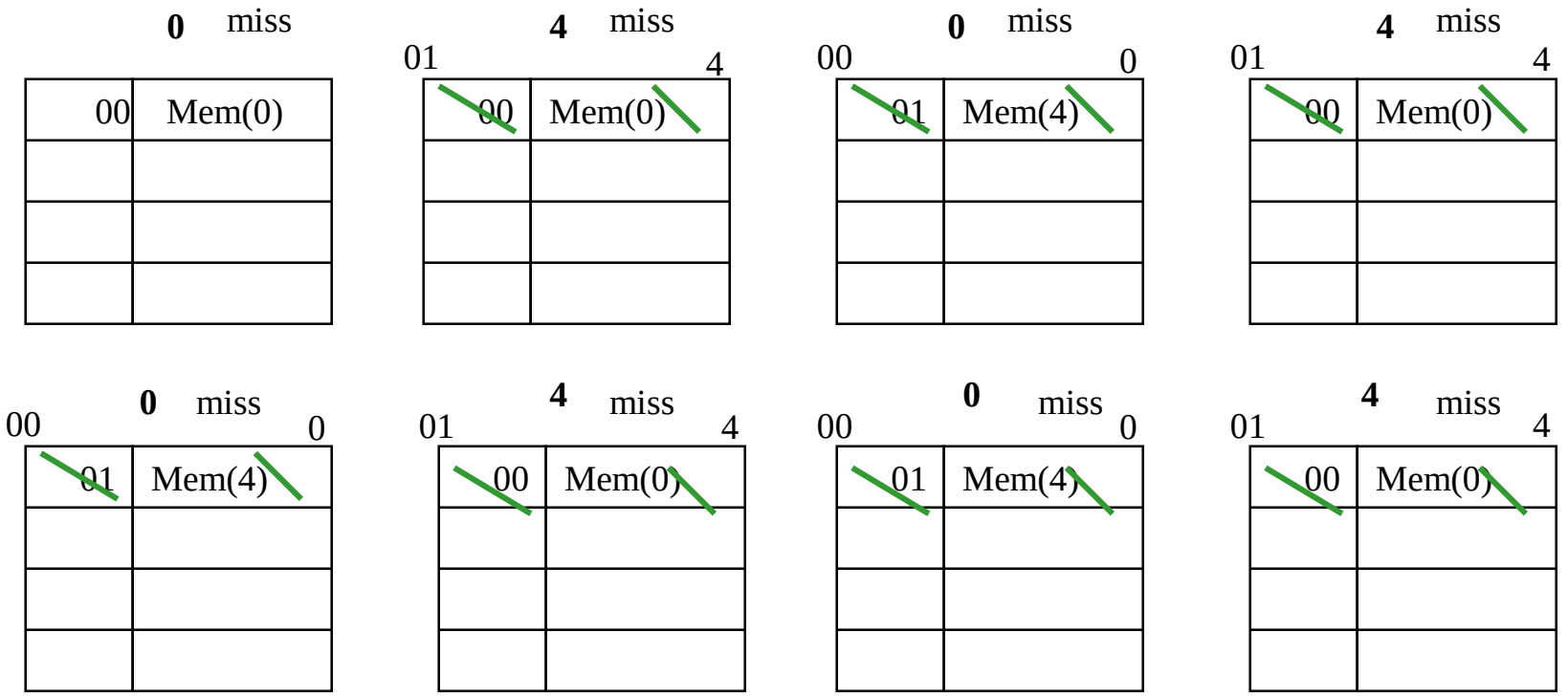
- 通常情况下，平均内存访问时间 = Hit Time + Miss Penalty x Miss Rate

每个块含有多个word时的一些考虑事项

- 与单word块处理相同，一次失效时，要将整个块都从内存中读入高速缓存
- 当块大小增加时，失效损失增大；一组优化方法：
  - Requested word first – 先传输一个块中需要访问的word
  - Early restart – 一个块中需要访问的word一准备好，就恢复处理器执行
- 非阻塞型高速缓存 – 高速缓存处理前期失效时，允许处理器继续访问高速缓存的其它块

高速缓存失效：极端例子

- 考虑访问Word地址（非Byte地址）：0 4 0 4 0 4 0 4



8 requests, 8 misses

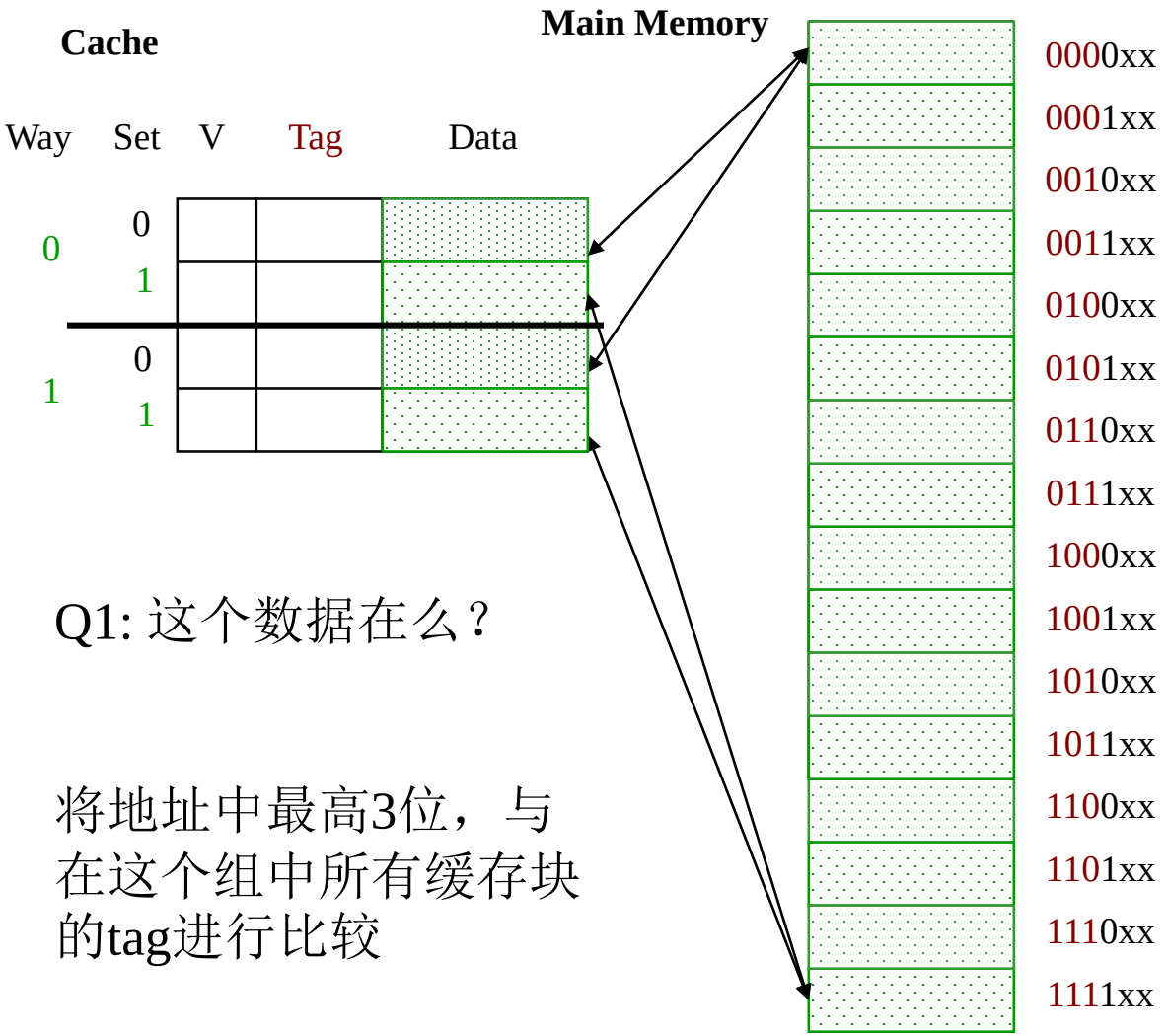
- 由于冲突失效产生的Ping pong效应 – 两个内存位置不断反复被映射到同一高速缓存块

多路组相联高速缓存

- 在直接映射缓存中，一个内存块只能映射到唯一一个缓存块中
- 处于另外一个极端，全相联缓存中，一个内存块可以映射到任意一个缓存块中

- 一个权衡是，将缓存分成若干sets，每个set包含N路（N路组相联）
  - 每个内存块被映射到唯一的组中（通过索引），而在这个组中，可以放到任意一个块中

组相联缓存示例



地址最低两位没有被使用：
定义一个32位word中的字节位置

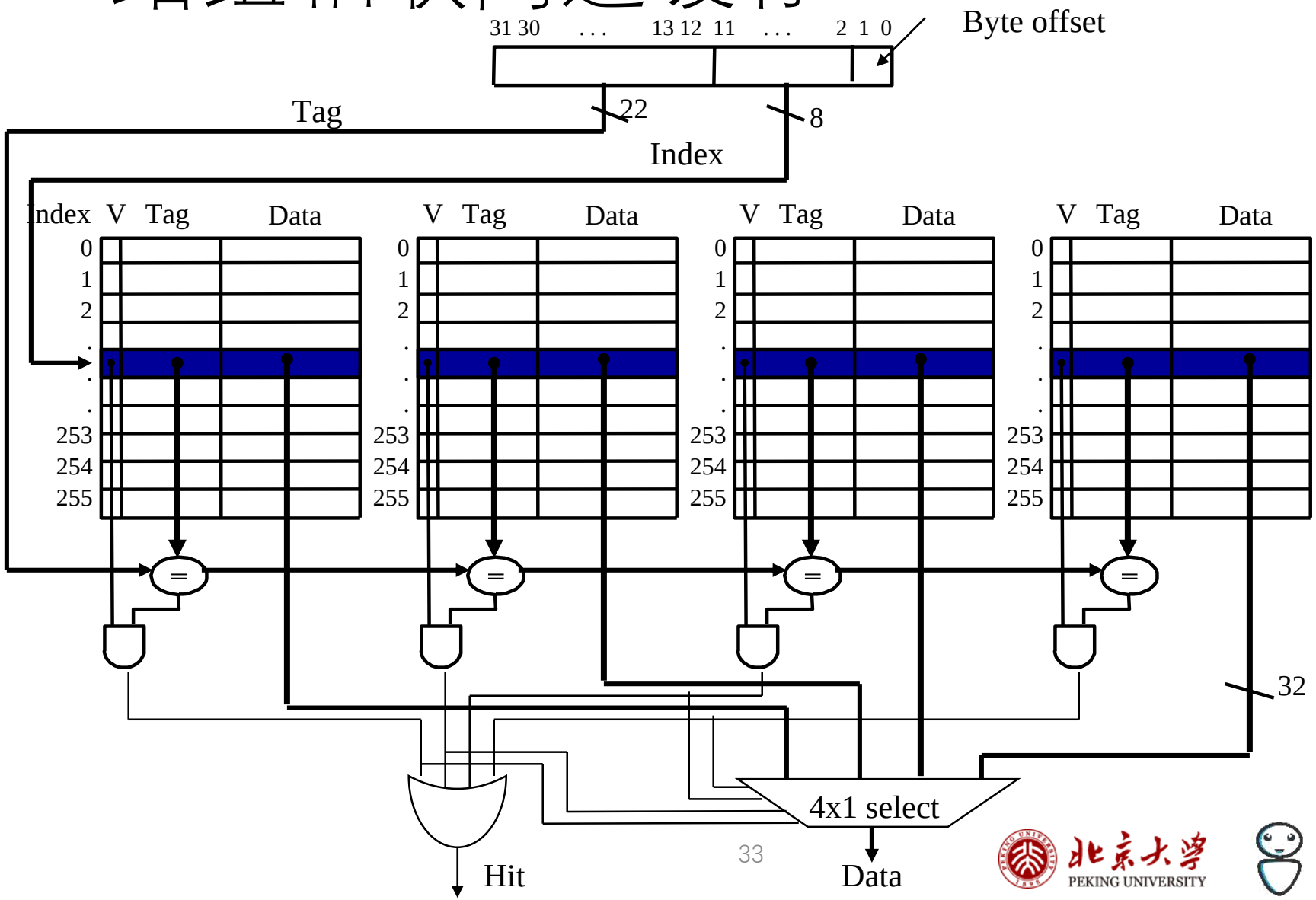
Q1: 这个数据在么？

将地址中最高3位，与在这个组中所有缓存块的tag进行比较

Q2: 如何找到数据？

用地址中接下来的1个低位作为索引值来查找高速缓存的组 (i.e., modulo the number of sets in the cache)

4路组相联高速缓存

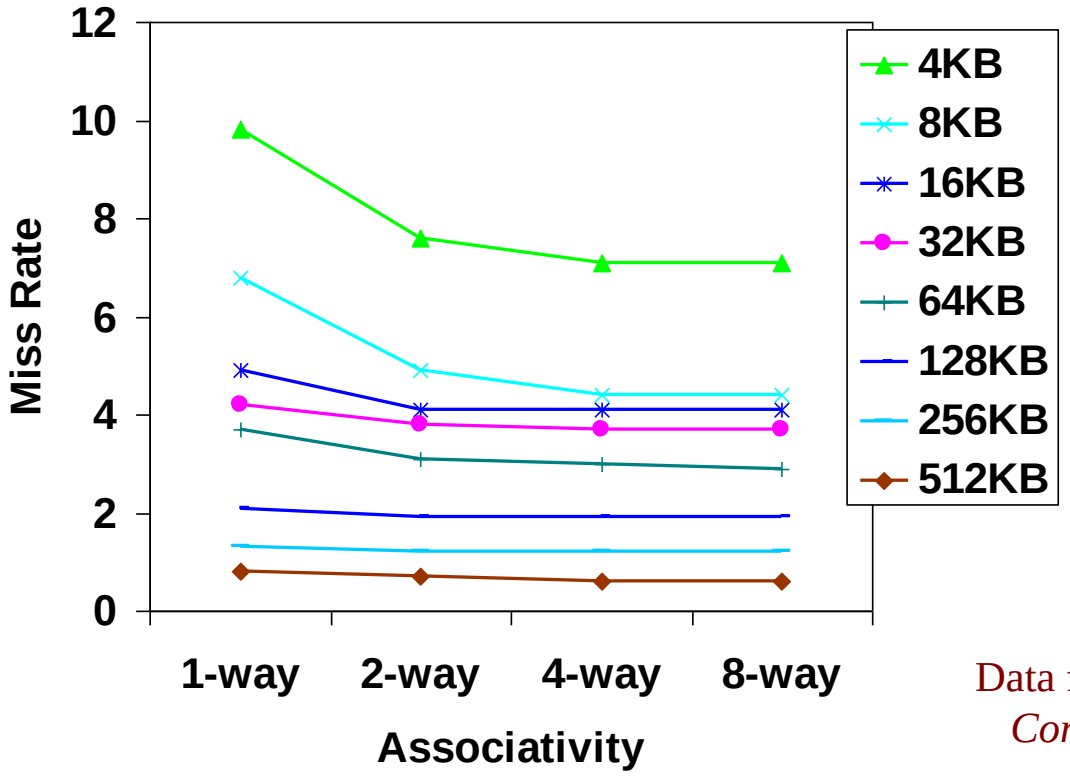


组相联高速缓存的开销

- 产生失效时，如何选择被替换的缓存块？
  - FIFO
  - 随机
  - Least Recently Used (LRU): 选择离现在最长时间没有被用到的缓存块
    - 必须有硬件来记录使用信息
- N路组相联高速缓存的开销
  - N个比较器（延迟、面积）
  - N-to-1选择器MUX
  - 无法像直接映射高速缓存那样在判断是否命中的同时猜测（命中）取数（不知道取哪一路）

组相联高速缓存的优势

- 做抉择时，需要综合考虑失效损失的代价和实现开销



Data from Hennessy & Patterson, *Computer Architecture*, 2003

最大的提升，在于从直接映射到2路组相联（失效率减少20%左右）

多级高速缓存

- 随着工艺进步，同样芯片面积可容纳更多的缓存
  - 分级处理，在下一级中采用更大的缓存容量
- 考虑例子：
  - CPI_{ideal} = 2, 到内存中有100个cycle的失效损失，36%的内存读写指令，2%的L1指令缓存失效率，4%的L1数据缓存失效率

CPIstalls = 2 + .02x100 + .36*.04*100 = 5.44

  - 再加上一个统一的L2缓存，如果它命中，就只给L1缓存带来25个cycle的失效损失；它本身有0.5%的总体失效率

CPIstalls = 2 + .02×25 + .36×.04×25 + .005×100 + .36×.005×100 = 3.54

多级高速缓存的考虑因素

- 对于L1和L2高速缓存的设计考虑因素是非常不同的
  - 第一级高速缓存，应当专注于尽量减小命中时间，以支持更短的时钟周期
    - 小容量、小缓存块
  - 第二级高速缓存，应当专注于减少失效率，以减小从很慢的内存中访问数据带来的高失效损失
    - 大容量、大缓存块
- L2高速缓存的存在可以显著降低L1高速缓冲的失效损失，所以L1高速缓存可以容忍较高的失效率，而专注于减少命中时间

高速缓存失效的三种原因

- 义务失效/冷失效（冷启动或者进程切换时产生）
  - 第一次访问缓存块时产生，没有什么办法避免
  - 如果要运行几百万条指令，义务失效可以忽略
- 冲突失效
  - 多个内存地址被映射到同一个缓存块中
  - 解决方法1：增大缓存容量（随即增加了缓存块数目）
  - 解决方法2：增加相联度（每个缓存位置存在多个缓存块）
- 容量失效
  - 高速缓存不能容纳程序所需访问的所有内容
  - 解决方法：增加缓存容量

提高高速缓存性能的方法 1/2

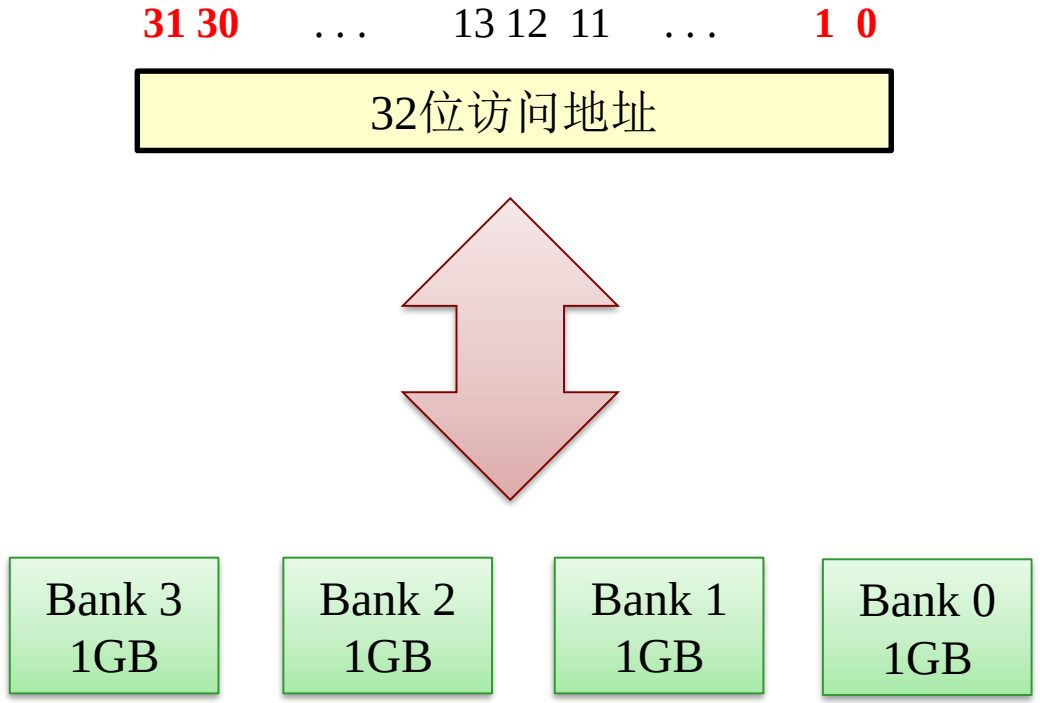
- 减小高速缓存命中时所需的时间
  - 更小的缓存
  - 采用直接映射
  - 更小的缓存块（更快取出所需word）
- 减小失效率
  - 更大的缓存
  - 允许更高的相联度
  - 更大的缓存块（典型值为16到64个字节）
  - victim cache – 小的缓冲区，保存刚刚被弹出的若干块数据

提高高速缓存性能的方法 2/2

- 减少失效损失
  - 更小的缓存块
  - 采用写缓冲，这样下一个读操作就不用等被替换的块真正完成被写操作
  - 当读失效的时候，检测victim cache或者写缓冲 – 看运气怎样
  - 对于大的缓存块，先取回关键word（需要操作的word）
  - 用多级缓存 – L2缓存的时钟周期可以比处理器时钟周期慢很多
  - 更快的下一级存储、更高的内存带宽
    - 更宽的内存总线
    - 交叉访问内存

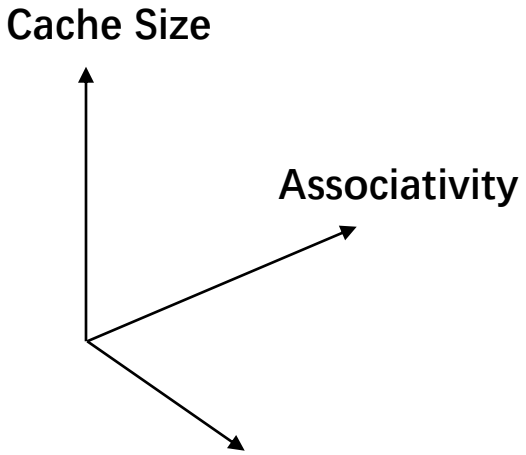
交叉访问存储器

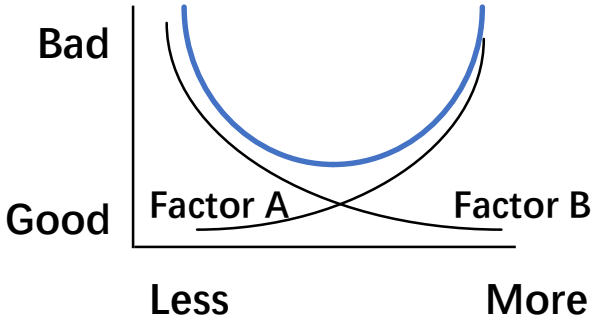
- 通过并行访问，提高总吞吐率
- 两种访问方法
  - 高位交叉访问
  - 低位交叉访问
- 各自优劣？



高速缓存设计空间

- 若干相互影响的维度
  - 缓存容量
  - 块大小
  - 相联度
  - 替换策略
  - write-through vs write-back
  - write allocation
- 需要权衡
  - 取决于数据访问特性
    - 不同应用、不同系统
  - 还取决于造价
- 简单的设计经常会更好





总结

- 存储层次概论
- 高速缓存基础
- 提高高速缓存性能

- 下一课：存储层次 II