Lecture 11: Cache III

命中率、缺失率、缺失损失

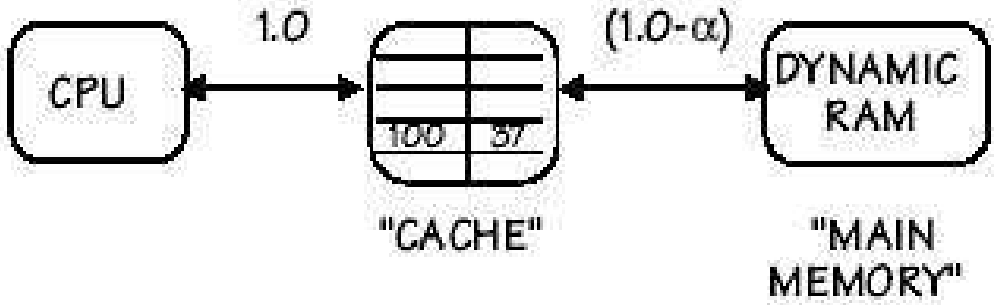
- **Hit: 要访问的信息在Cache中**
  - **Hit Rate(命中率)：在Cache中的概率**
  - **Hit Time (命中时间) ：在Cache中的访问时间，包括：**

    Time to determine hit/miss + Cache access time

    (即： 判断时间 + Cache访问)
- **Miss: 要找的信息不在Cache中**
  - **Miss Rate (缺失率) = 1 - (Hit Rate)**
  - **Miss Penalty (缺失损失)：访问一个主存块所花时间**
- **Hit Time << Miss Penalty （Why?）**

Average access time(平均访问时间)

Program-Transparent Memory Hierarchy



Cache contains TEMPORARY COPIES of selected main memory locations... eg. Mem[100] = 37

GOALS:

1) Improve the average access time

要提高平均访问速度，必须提高命中率！

α HIT RATIO: Fraction of refs found in CACHE.

(1-α) MISS RATIO: Remaining references.

$$t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$$

2) Transparency (compatibility, programming ease)

Challenge:
To make the hit ratio as high as possible.

Cache对程序员是透明，以方便编程！

命中率到底应该有多大?

How high of a hit ratio?

Suppose we can easily build an on-chip static memory with a 4 nS access time, but the fastest dynamic memories that we can buy for main memory have an average access time of 40 nS. How high of a hit rate do we need to sustain an average access time of 5 nS?

$$\alpha = 1 - \frac{t_{ave} - t_c}{t_m} = 1 - \frac{5-4}{40} = 97.5\%$$

WOW, a cache really needs to be good?

看看命中率对平均访问时间的影响

- 设H是命中率，则平均访问时间$T = HT_C + (1 - H)(T_C + T_M)$
  $= T_C + (1 - H)T_M$

- 例1. 若H=0.85, Tc=1ns, T_M=20ns，则T为多少？
  答： T = 4ns

- 例2. 若命中率H提高到0.95，则结果又如何？
  答： T = 2ns

- 例3. 若命中率为0.99呢？
  答： T = 1.2ns

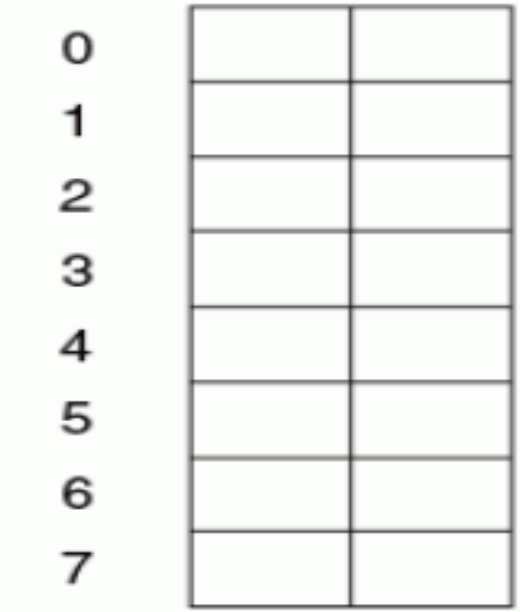
访存速度与命中率的关系非常大！

高速缓存的失靶率和关联度

- 三种映射方式
  - 直接映射：唯一映射（只有一个可能的位置）
  - 全相联映射：任意映射（每个位置都可能）
  - N-路组相联映射：N-路映射（有N个可能的位置）
- 什么叫关联度？
  - 一个主存块映射到Cache中时，可能存放的位置个数
    - 直接映射：关联度最低，为1
    - 全相联映射：关联度最高，为Cache行数
    - N-路组相联映射：关联度居中，为N
- 关联度和miss rate有什么关系呢？和命中时间的关系呢？
  - 直观上，你的结论是什么？（Cache大小和块大小一定时）
    - 缺失率：直接映射最高，全相联映射最低
    - 命中时间：直接映射最小，全相联映射最大
  - 用例子来说明

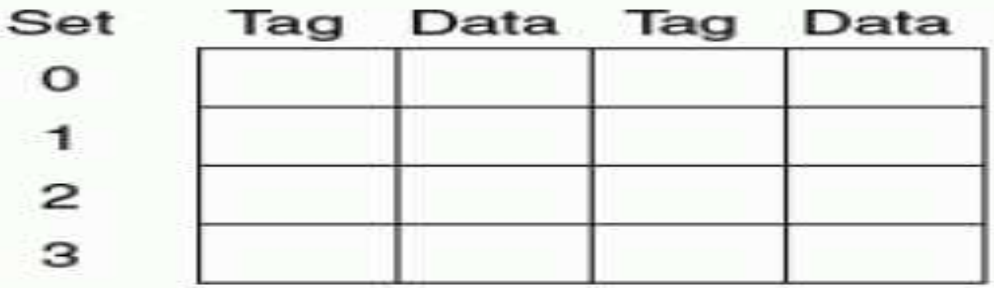
关联度示例

One-way set associative (direct mapped)



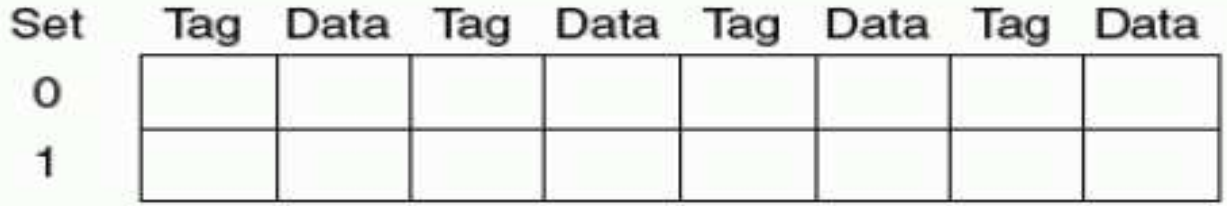
关联度为多少？ 1

Two-way set associative



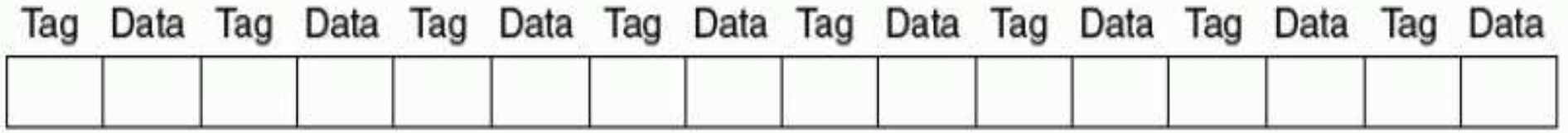
关联度为多少？ 2

Four-way set associative



关联度为多少？ 4

Eight-way set associative (fully associative)



关联度为多少？ 8

BACK

例子：Cache缺失和关联度

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Cache块号=主存块号 mod 4　　缺失5次

例子：Cache缺失和关联度

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

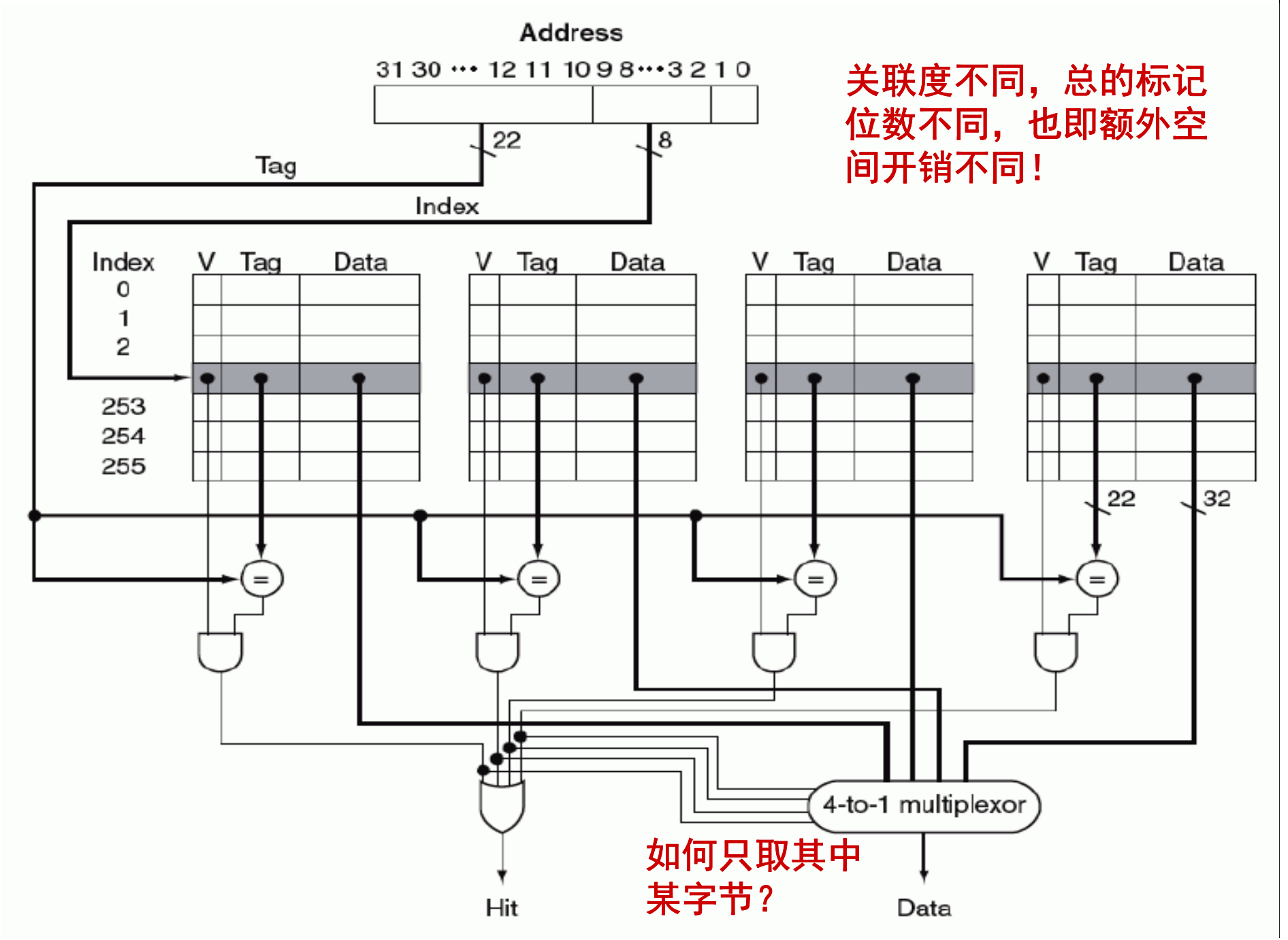
Cache组号 = 主存块号 mod 2　　缺失4次

例子：Cache缺失和关联度

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

缺失3次

关联度越高，缺失次数越少！


Address
31 30 ··· 12 11 10 9 8···3 2 1 0
22
8
Tag
Index
Index
V
Tag
Data
0
1
2
253
254
255
22
32
=
4-to-1 multiplexor
Hit
Data
关联度不同，总的标记位数不同，也即额外空间开销不同！
如何只取其中某字节？

标记位大小与关联度

问题：若主存地址32位，块大小为16字节，Cache总大小为4K行，问：标记位的总位数是多少？

直接映射方式下：相当于每组1行，共4K组，标志占32-4-12=16位
总位数占 4Kx16=64K位

关联度增加一倍(2-way)：每组2行，共2K组，标志占32-4-11=17位
总位数占4Kx17=68K位

关联度增加2倍(4-way)：每组4行，共1K组，标志占32-4-10=18位
总位数占4Kx18=72K位

全相联时：整个为1组，每组4K行，标志占32-4=28位
总位数占4Kx28=112K位

关联度越高，总的标记位数越多，额外空间开销越大！

The Need to Replace! (何时需要替换？)

- **Direct Mapped Cache:**
  - 映射唯一，毫无选择，无需考虑替换
- **N-way Set Associative Cache:**
  - 每个主存数据有N个Cache行可选择，需考虑替换
- **Fully Associative Cache:**
  - 每个主存数据可存放到Cache任意行中，需考虑替换

结论：若Cache miss in a N-way Set Associative or Fully Associative Cache，则可能需要替换。其过程为：

- 从主存取出一个新块
- 选择一个有映射关系的空Cache行
- 对应的Cache行已被占满而需要调入新的主存块时，必须考虑从cache行中调出一个主存块

替换(Replacement)算法

- 问题举例：

  组相联映射时，假定第0组的两行分别被主存第0和8块占满，此时若需调入主存第16块，根据映射关系，它只能放到Cache第一组，因此，第一组中必须调出一块，那么调出哪一块呢？

  这就是淘汰策略问题，也称替换算法。

- 常用替换算法有：
  - 先进先出FIFO （first-in-first-out）
  - 最近最少用LRU （ least-recently used）
  - 最不经常用LFU （ least-frequently used）
  - 随机替换算法（Random）

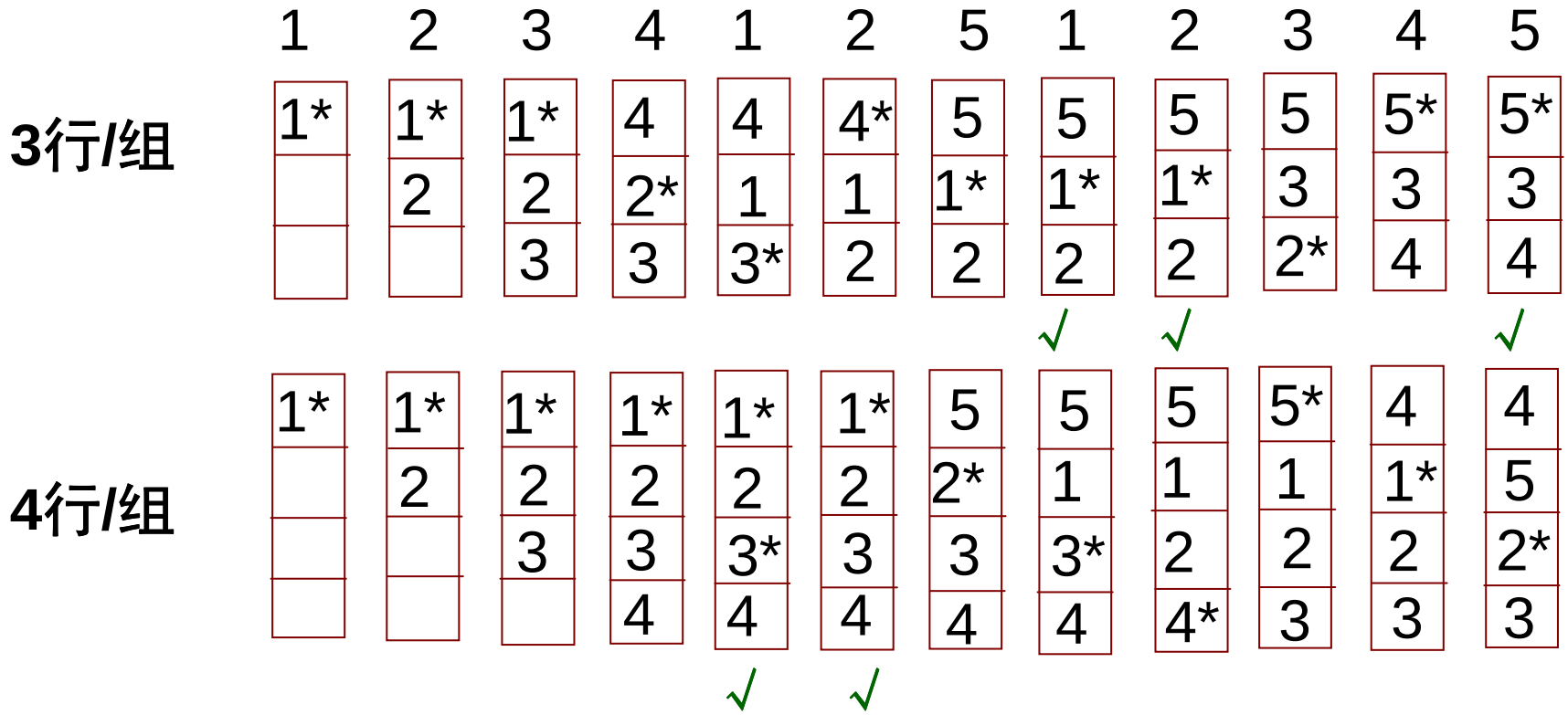
  等等

这里的替换策略和后面的虚拟存储器所用的替换策略类似，将是以后操作系统课程的重要内容，本课程只做简单介绍。有兴趣的同学可以自学。

替换算法-先进先出（FIFO）

- 总是把最先进入的那一块淘汰掉。

  例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组的情况。

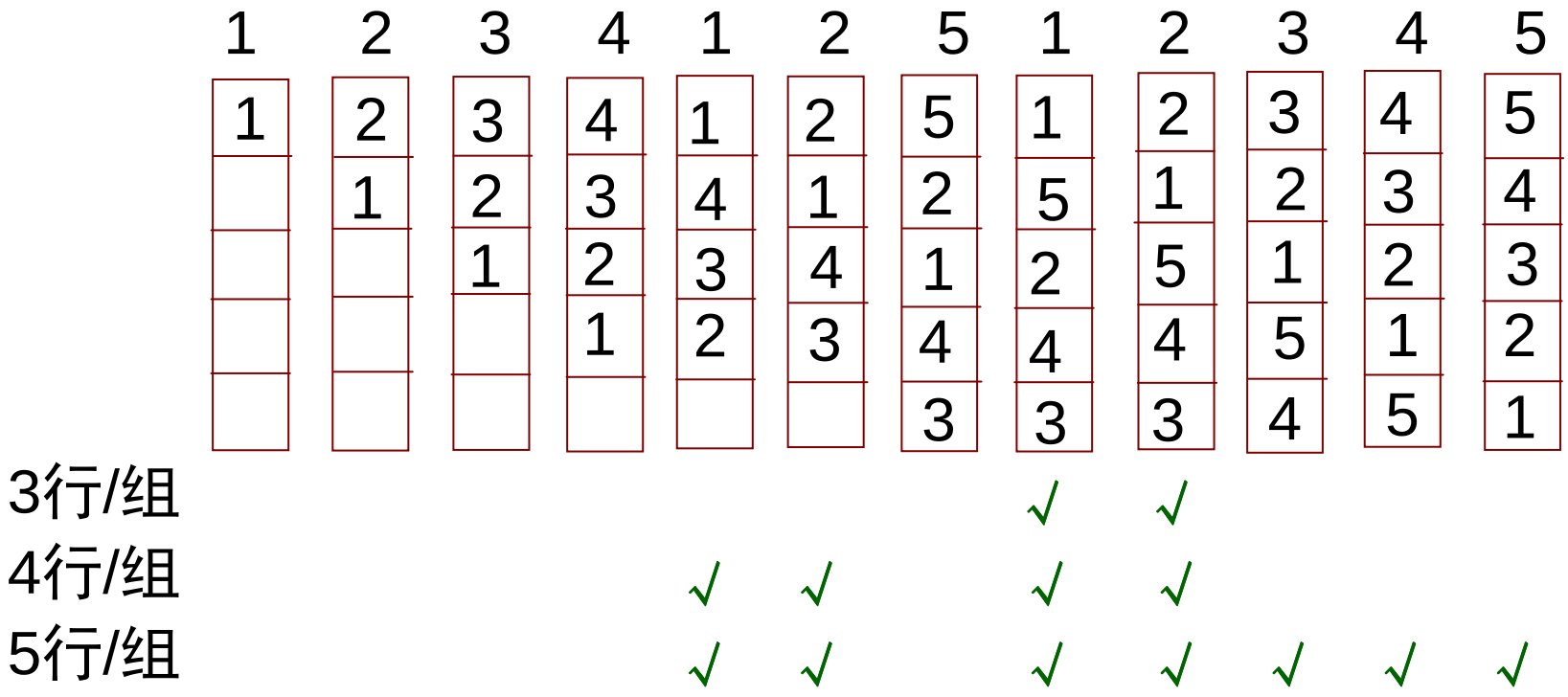


由此可见，FIFO不是一种堆栈算法，即命中率并不随组的增大而提高。

替换算法-最近最少用(LRU)

- 总是把最近最少用的那一块淘汰掉。

  例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组、 5行/组的情况。



替换算法-最近最少用

- 是一种堆栈算法，它的命中率随组的增大而提高。
- 当分块局部化范围(即：某段时间集中访问的存储区)超过了Cache存储容量时，命中率变得很低。极端情况下，假设地址流是1,2,3,4,1 2,3,4,1,……，而Cache每组只有3行，那么，不管是FIFO，还是LRU算法，其命中率都为0。这种现象称为颠簸(Thrashing / PingPong)。
- 该算法具体实现时，并不是通过移动块来实现的，而是通过给每个cache行设定一个计数器，根据计数值来记录这些主存块的使用情况。这个计数值称为**LRU位**。

具体实现

Pseudo-LRU

- One LRU bit for each way.
- When a set is accessed, the bit corresponding to the way containing the desired block is turned on.
- If all the bits associated with a set are turning on, they are reset with the exception of the most recently turned on bit.
- When a block must be replaced, the processor chooses a block from the way whose bit is turned off.
- If more than one choice is available, often choose randomly.

替换算法-其他算法

- 最不经常用（LFU）算法：

  替换掉Cache中引用次数最少的块。LFU也用与每个行相关的计数器来实现。

  （这种算法与LRU有点类似，但不完全相同。）

- 随机算法：

  随机地从候选的cache行中选取一个淘汰，与使用情况无关。

  （模拟试验表明，随机替换算法在性能上只稍逊于基于使用情况的算法。而且代价低！）