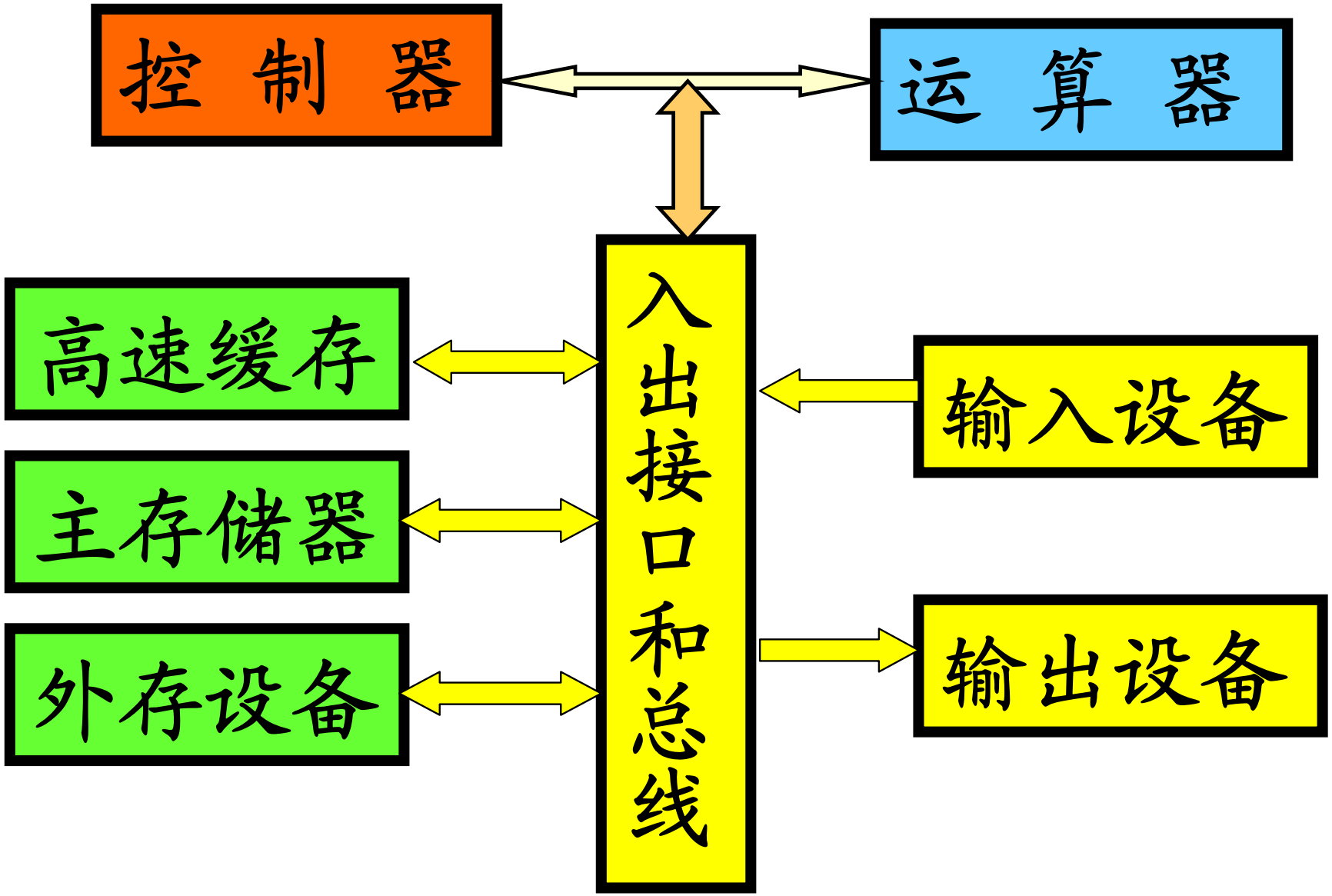
存储器层次结构

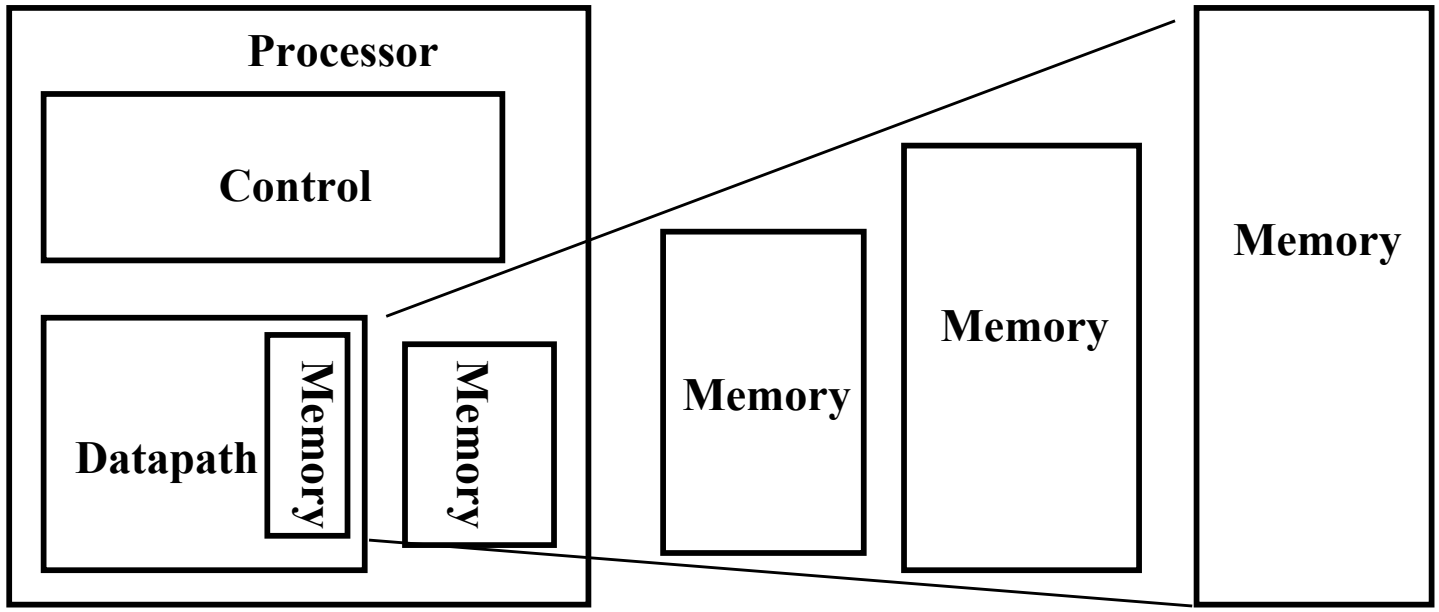
Part2：Cache映射、影响Cache命中率的因素

计算机硬件系统

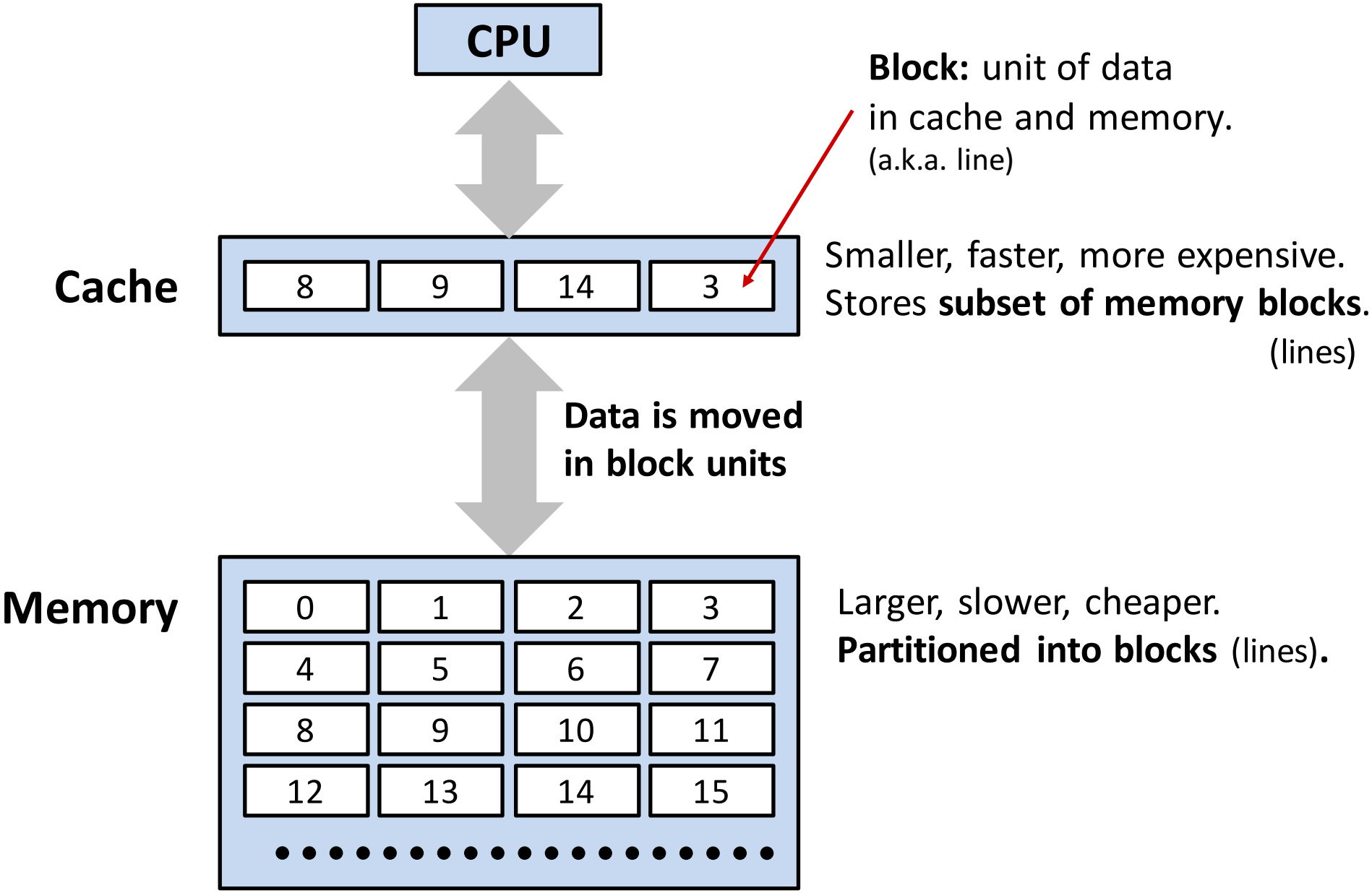


现代计算机的层次存储系统

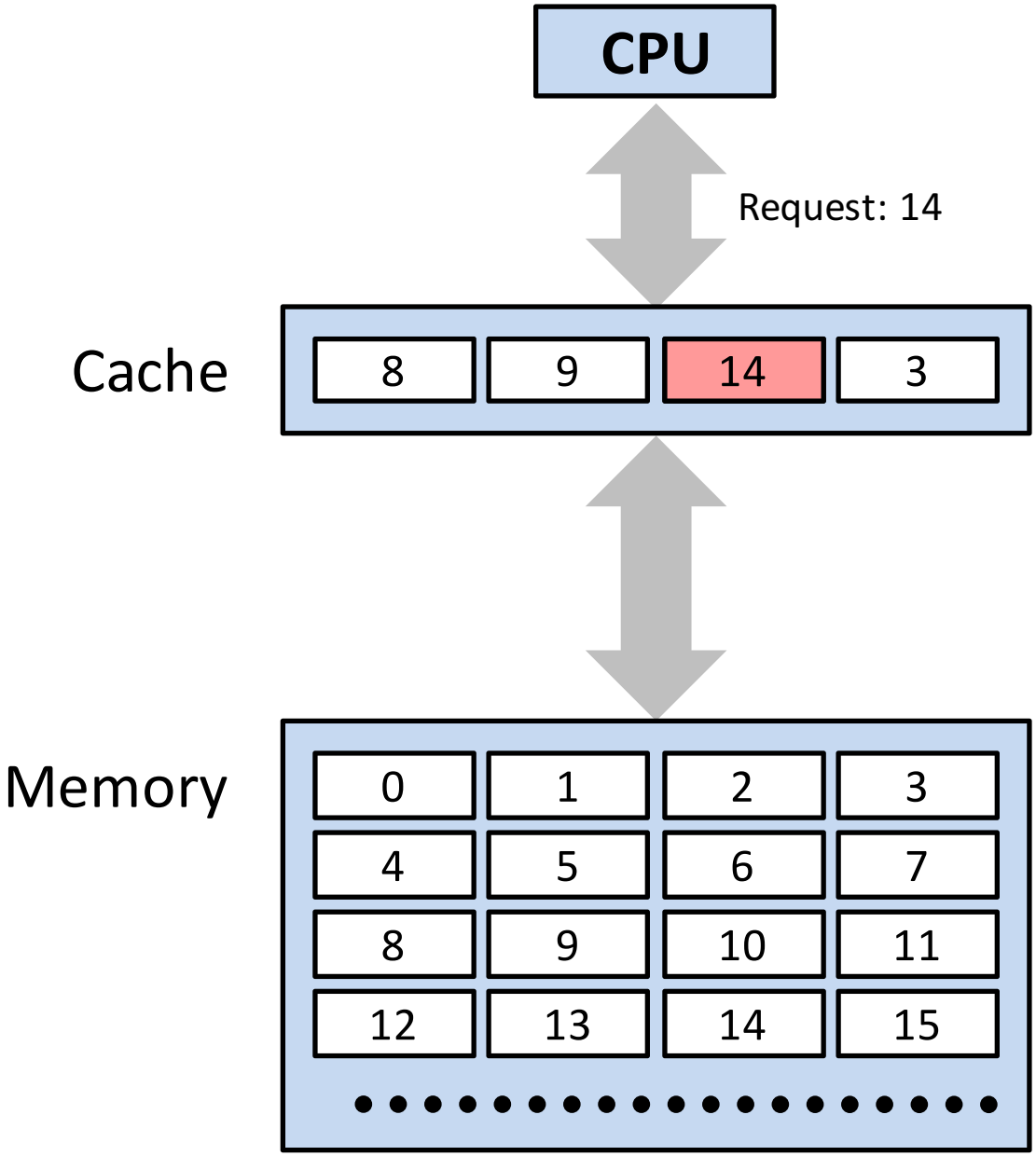
- 利用程序的局部性原理:
  - 以最低廉的价格提供尽可能大的存储空间
  - 以最快速的技术实现高速存储访问



Cache原理


CPU
Block: unit of data
in cache and memory.
(a.k.a. line)
Cache
8
9
14
3
Smaller, faster, more expensive.
Stores subset of memory blocks.
(lines)
Data is moved
in block units
Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper.
Partitioned into blocks (lines).

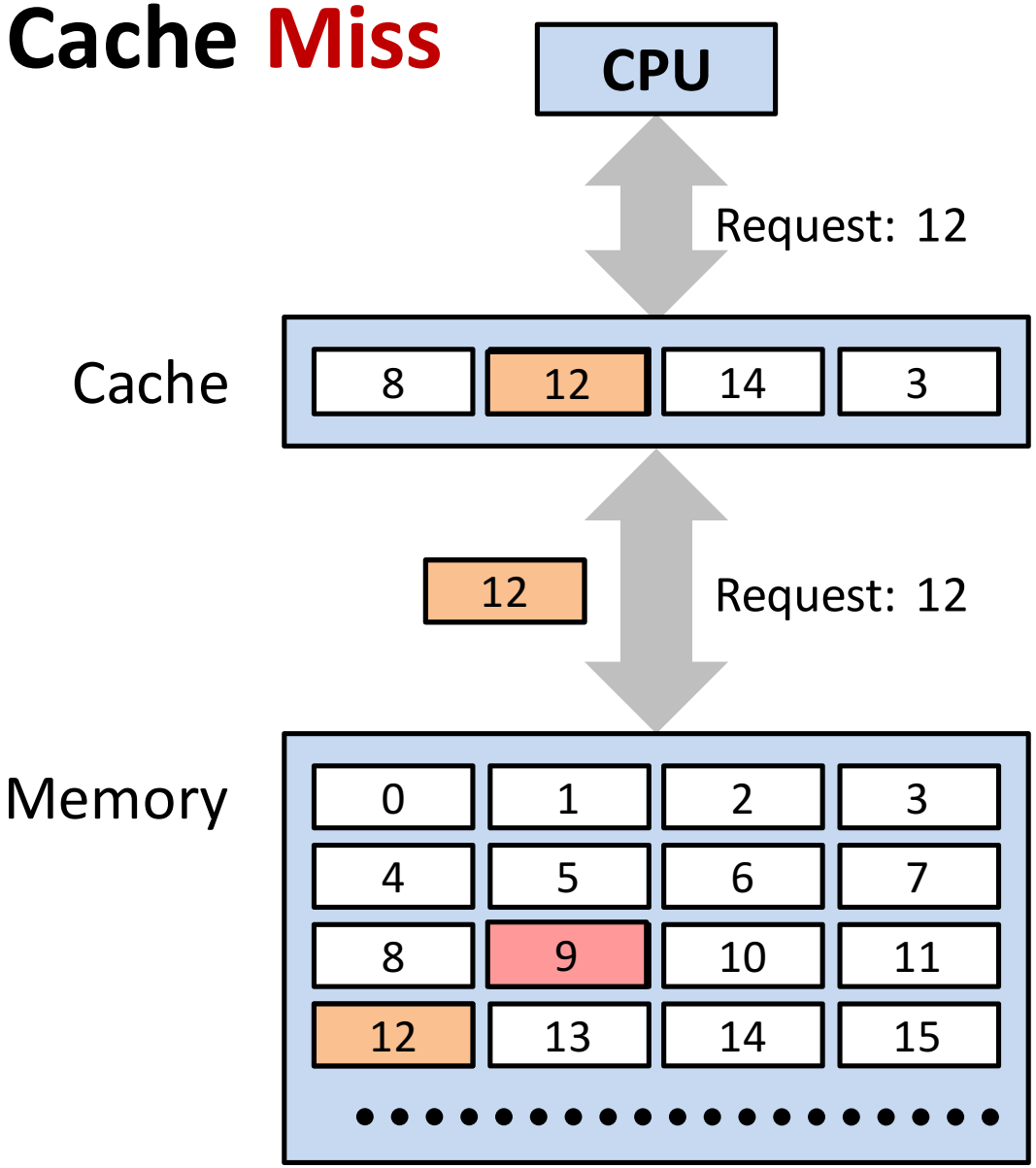
Cache Hit



1. Request** data in block **b.

2. Cache hit:
Block b is in cache.

Cache Miss



1. Request** data in block **b.

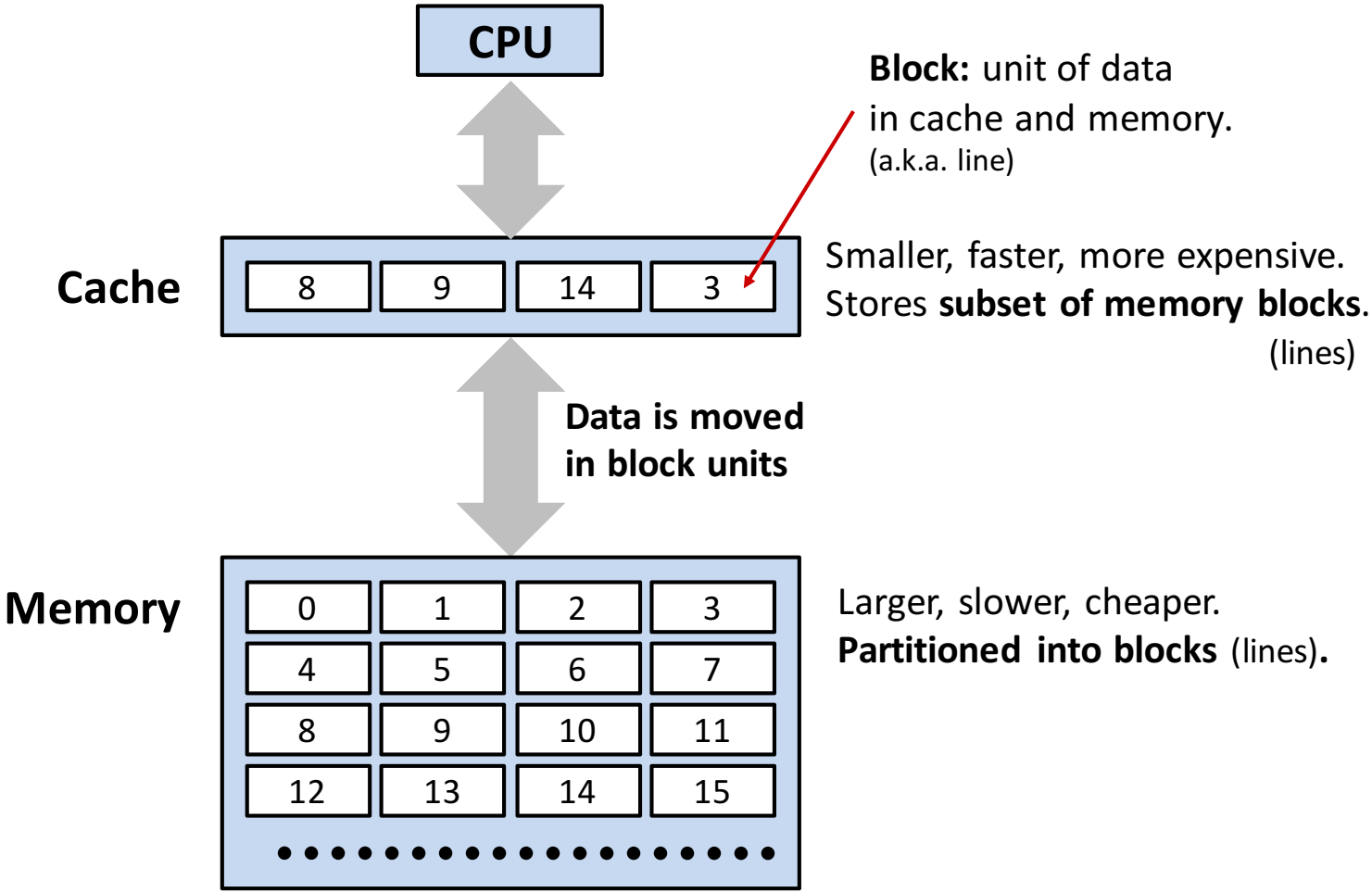
2. Cache miss:
block is not in cache

3. Cache eviction:
Evict a block to make room, maybe store to memory.

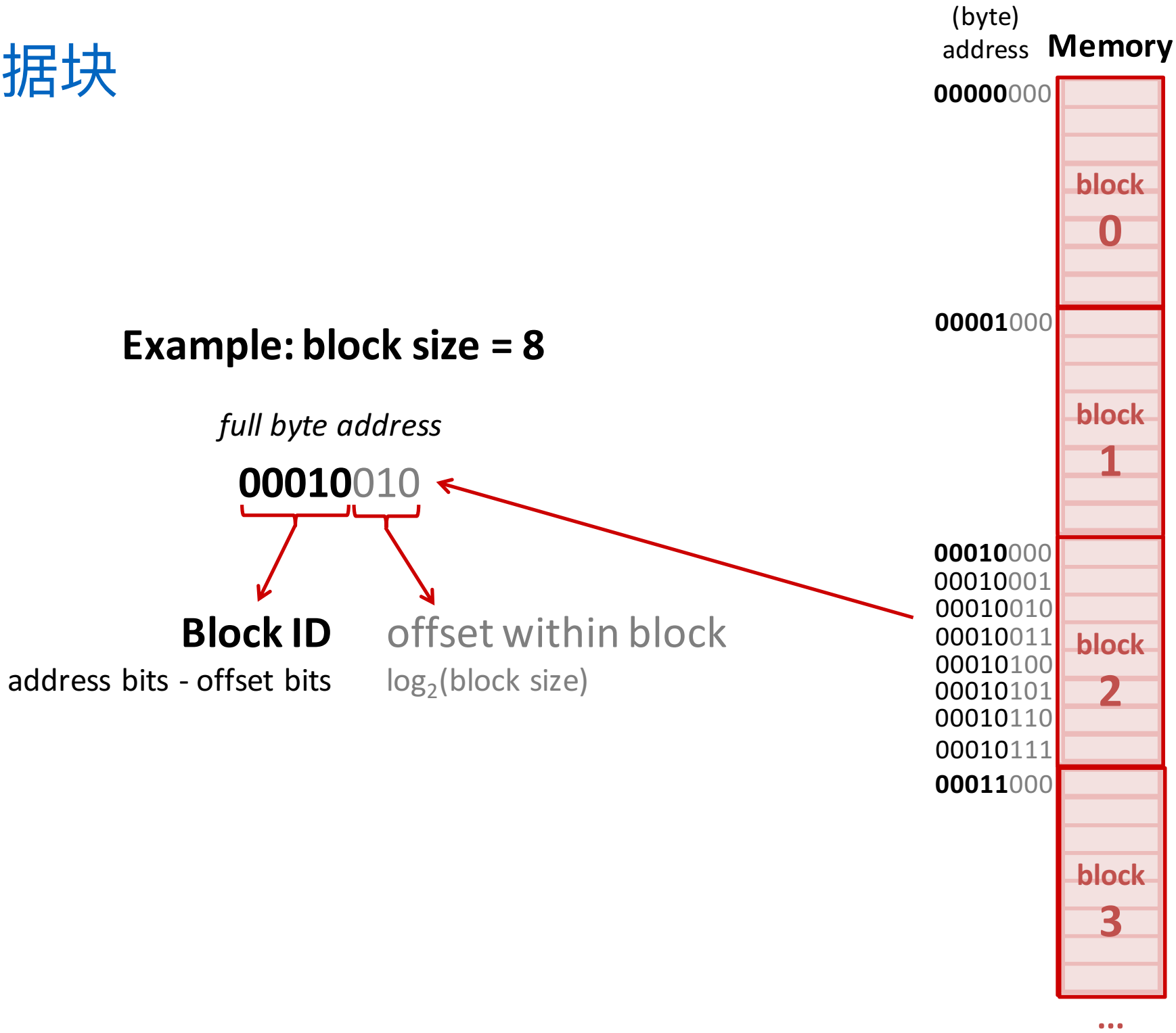
4. Cache fill:
Fetch block from memory, store in cache.

Placement Policy:
where to put block in cache

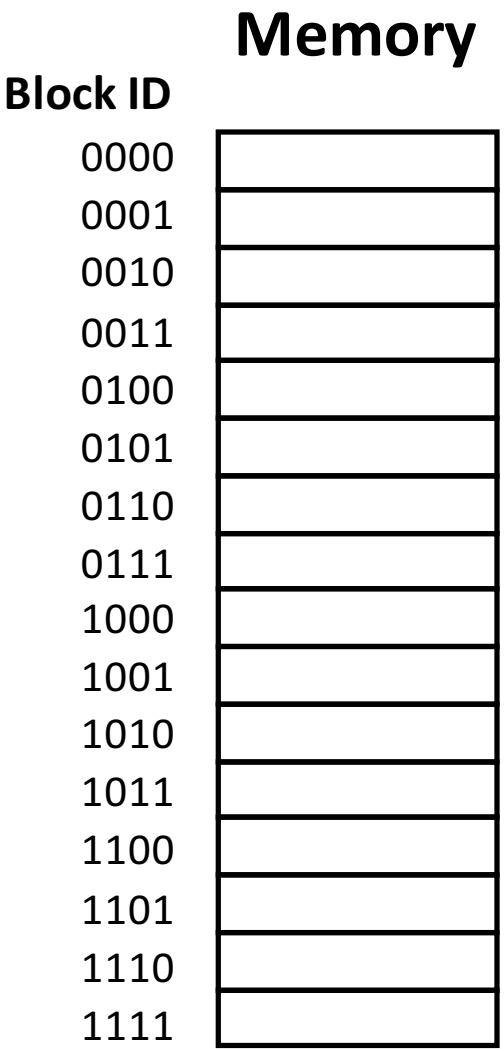
***Re*placement Policy:**
which block to evict


CPU
Block: unit of data
in cache and memory.
(a.k.a. line)
Cache
8
9
14
3
Smaller, faster, more expensive.
Stores subset of memory blocks.
(lines)
Data is moved
in block units
Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper.
Partitioned into blocks (lines).

数据块

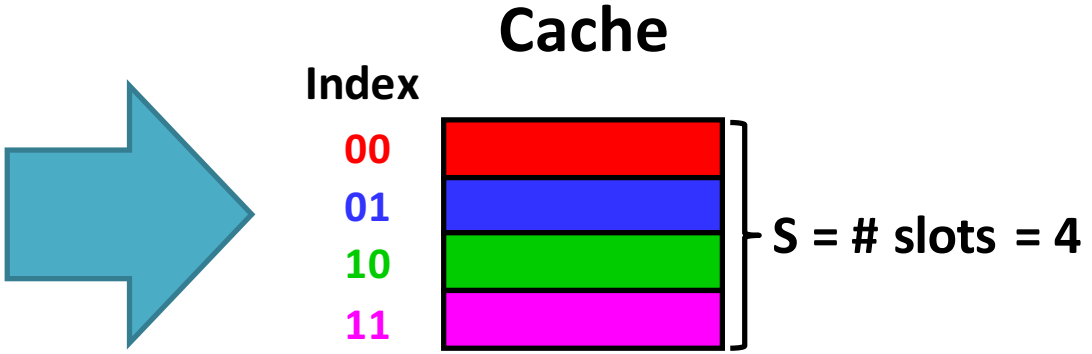

(byte)
address
Memory
00000000
block
0
00001000
block
1
00010000
00010001
00010010
00010011
00010100
00010101
00010110
00010111
block
2
00011000
block
3
...
Example: block size = 8
full byte address
00010010
Block ID
address bits - offset bits
offset within block
log2(block size)

Cache映射



Large, fixed number of block slots.

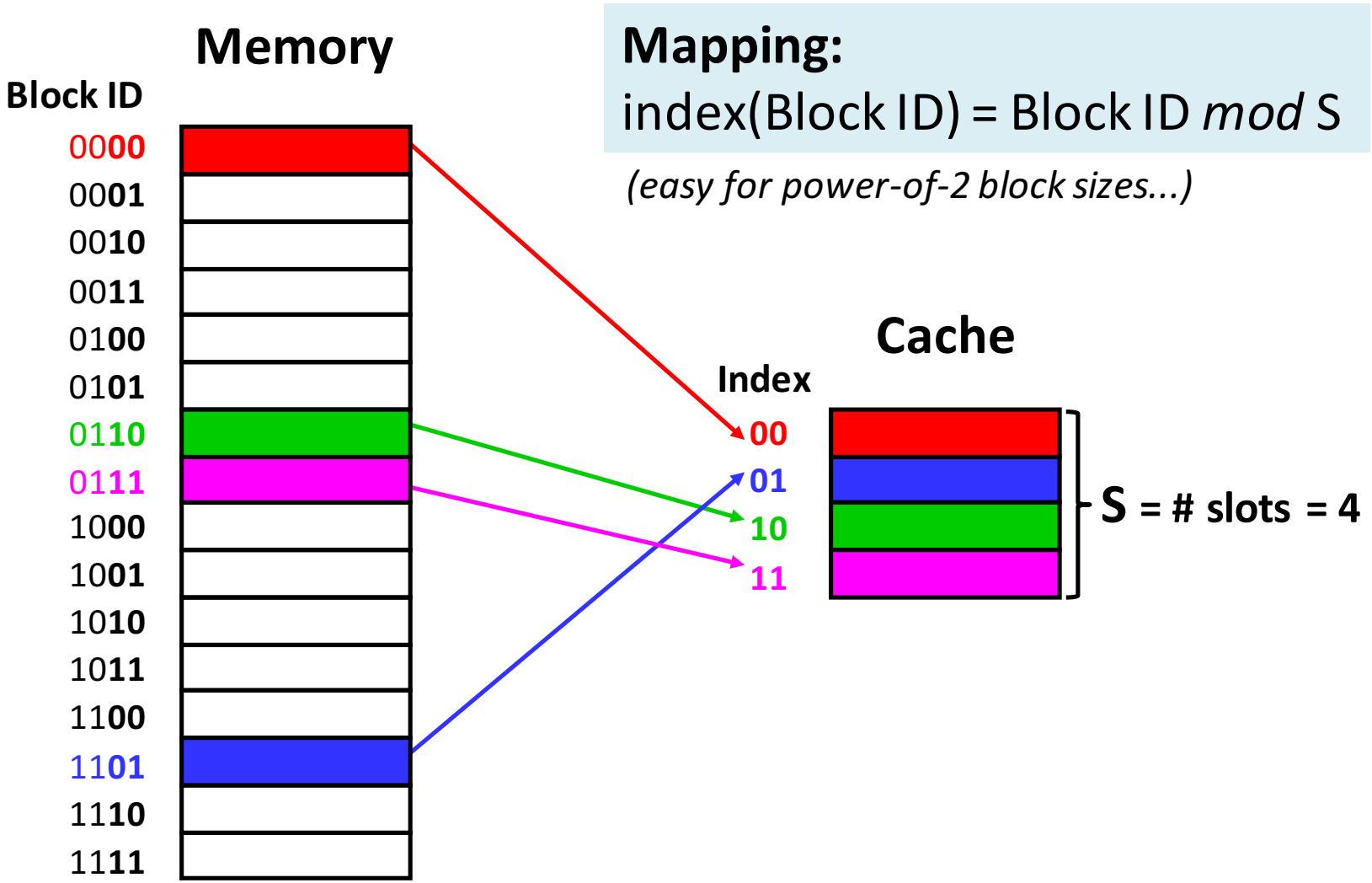
Mapping:
index(Block ID) = **???**



Small, fixed number of block slots.

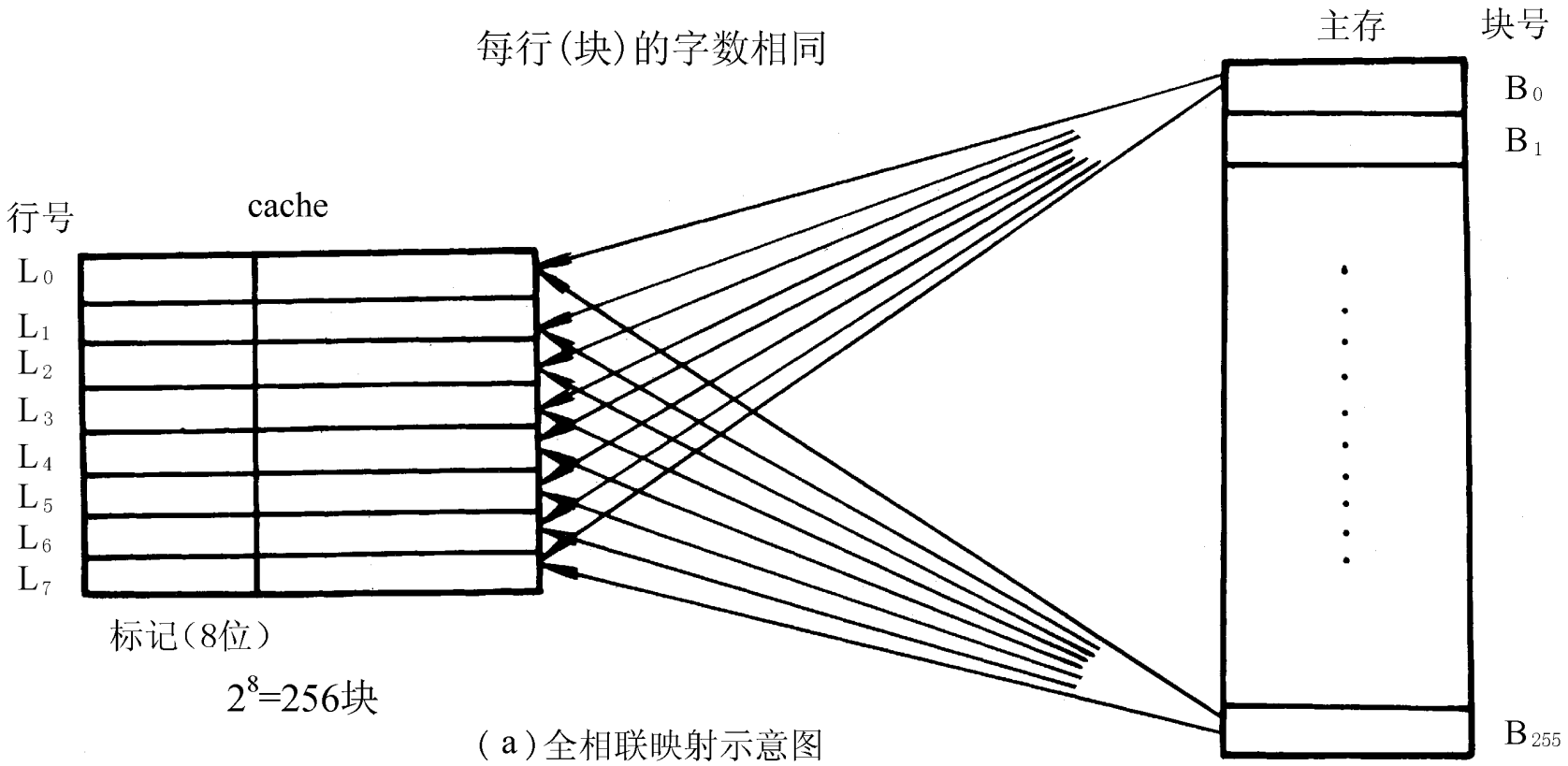
直接映射方式

- 一个主存块只能拷贝到cache的一个特定行位置上去



全相联映射方式

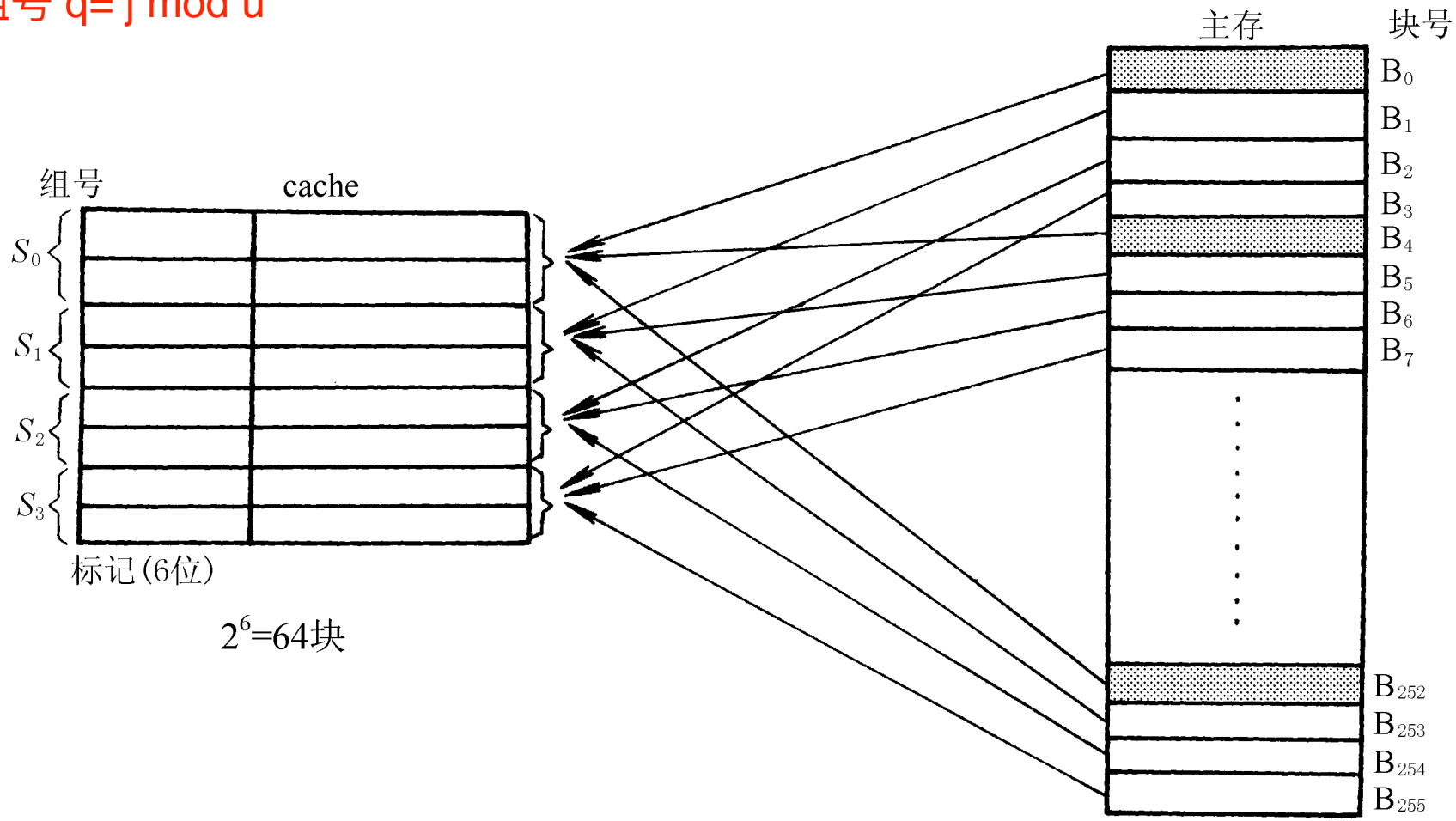
全相联映射方式基本思想：一个主存块可以装入cache任意一行



(a)全相联映射示意图

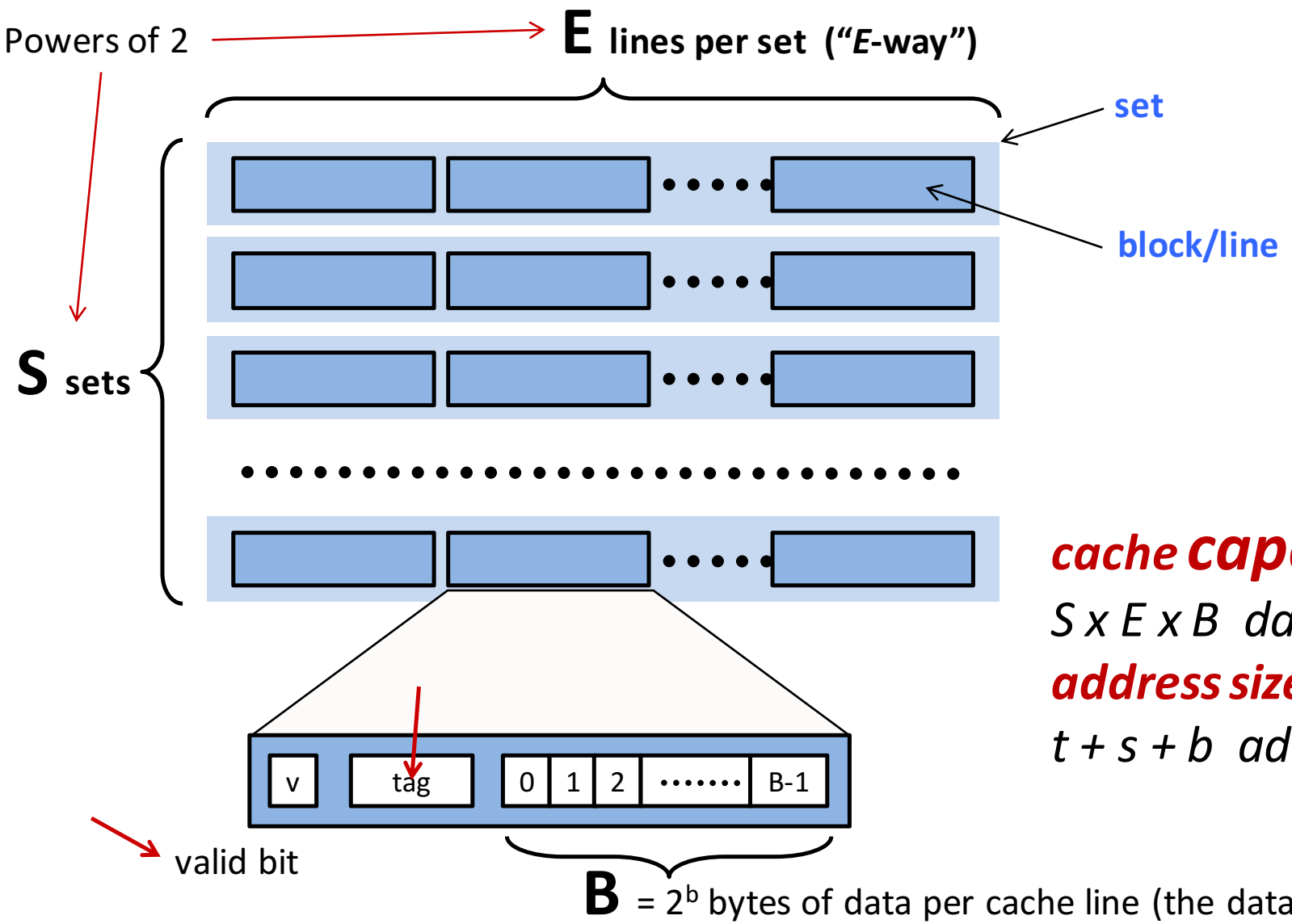
组相联映射方式

- 将cache分组，组间采用直接映射方式，组内采用全相联的映射方式
- Cache分u组，组内容量v行；映射方法：
- 组号 q= j mod u



(a) 组相联映射示意图(4组)

一般化的Cache组织形式



cache capacity:

S x E x B data bytes

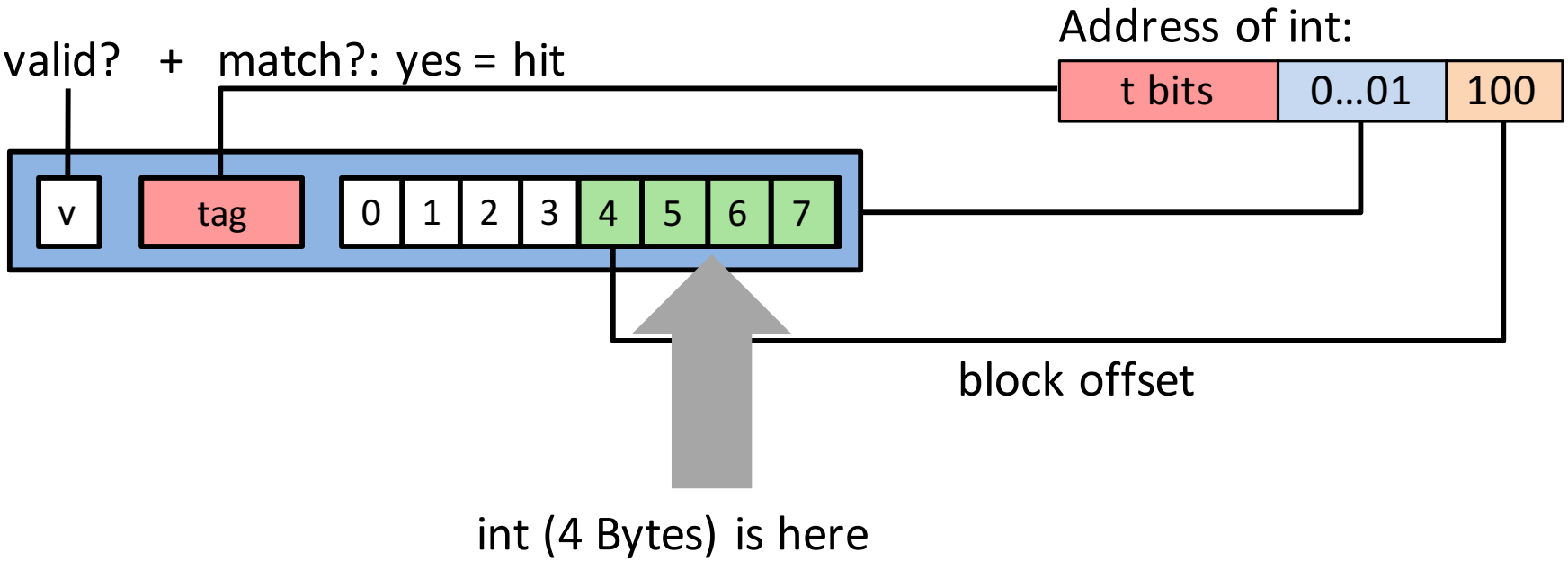
address size:

t + s + b address bits

读Cache：直接映射

This cache:

- Block size: 8 bytes
- Associativity: 1 block per set (direct mapped)



读Cache：直接映射

12-bit address

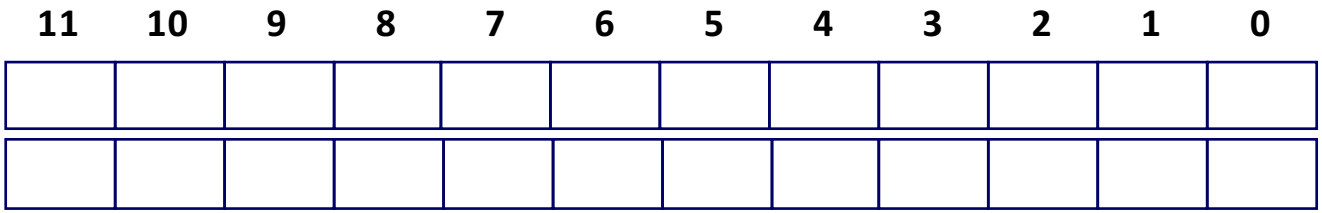
16 lines, 4-byte block size

Direct mapped

0x354

0xA20

Offset bits? Index bits? Tag bits?



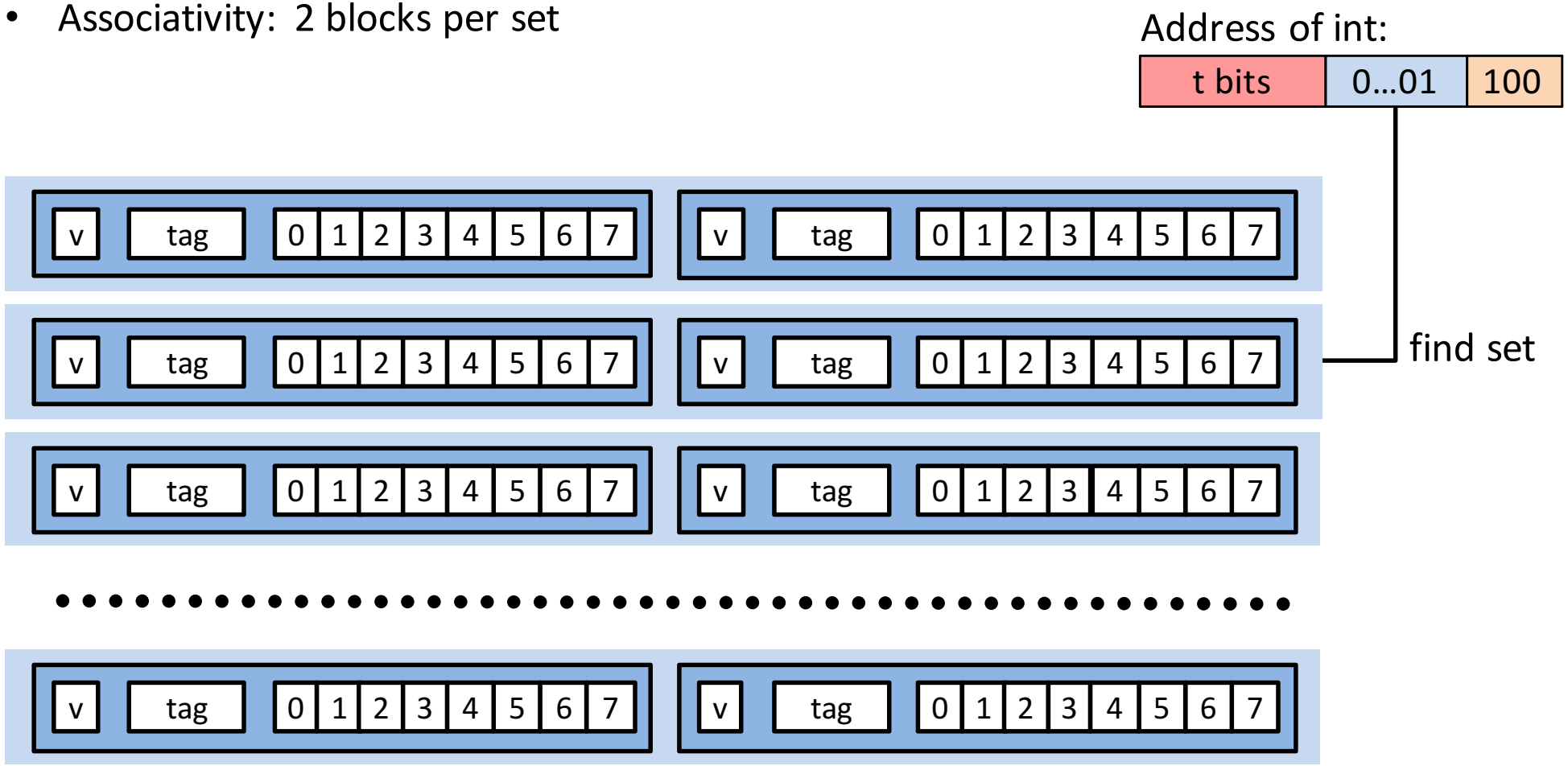
Index	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Index	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

读Cache：组相联映射

This cache:

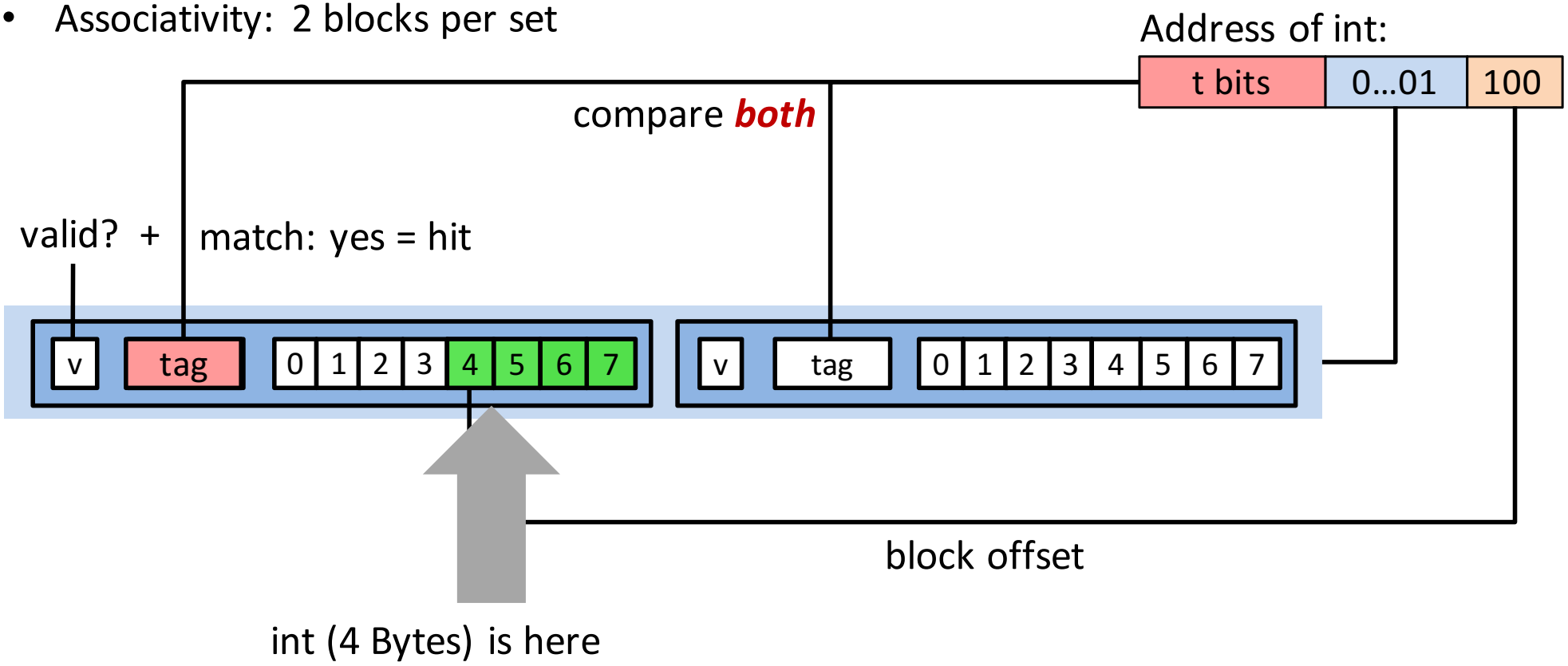
- Block size: 8 bytes
- Associativity: 2 blocks per set



读Cache：组相联映射

This cache:

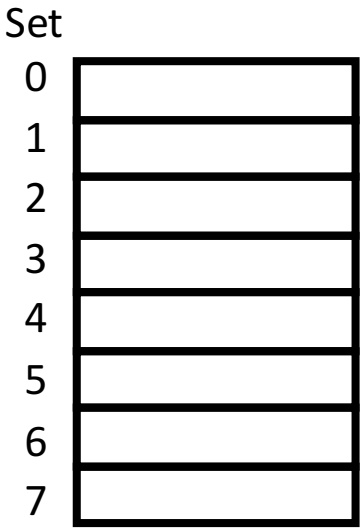
- Block size: 8 bytes
- Associativity: 2 blocks per set



8块cache配置成不同映射方式

1-way

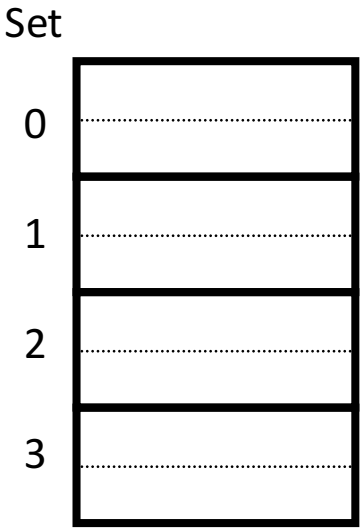
8 sets,
1 block each



direct mapped

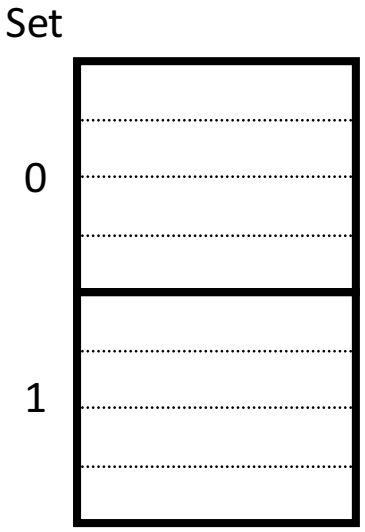
2-way

4 sets,
2 blocks each



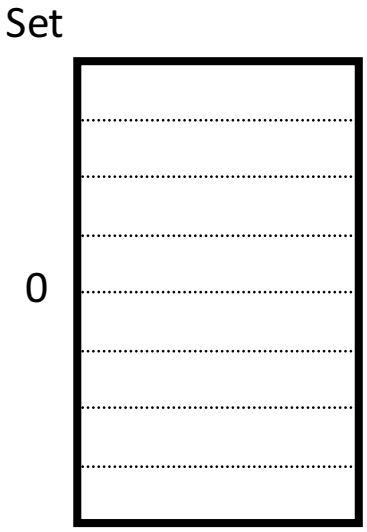
4-way

2 sets,
4 blocks each



8-way

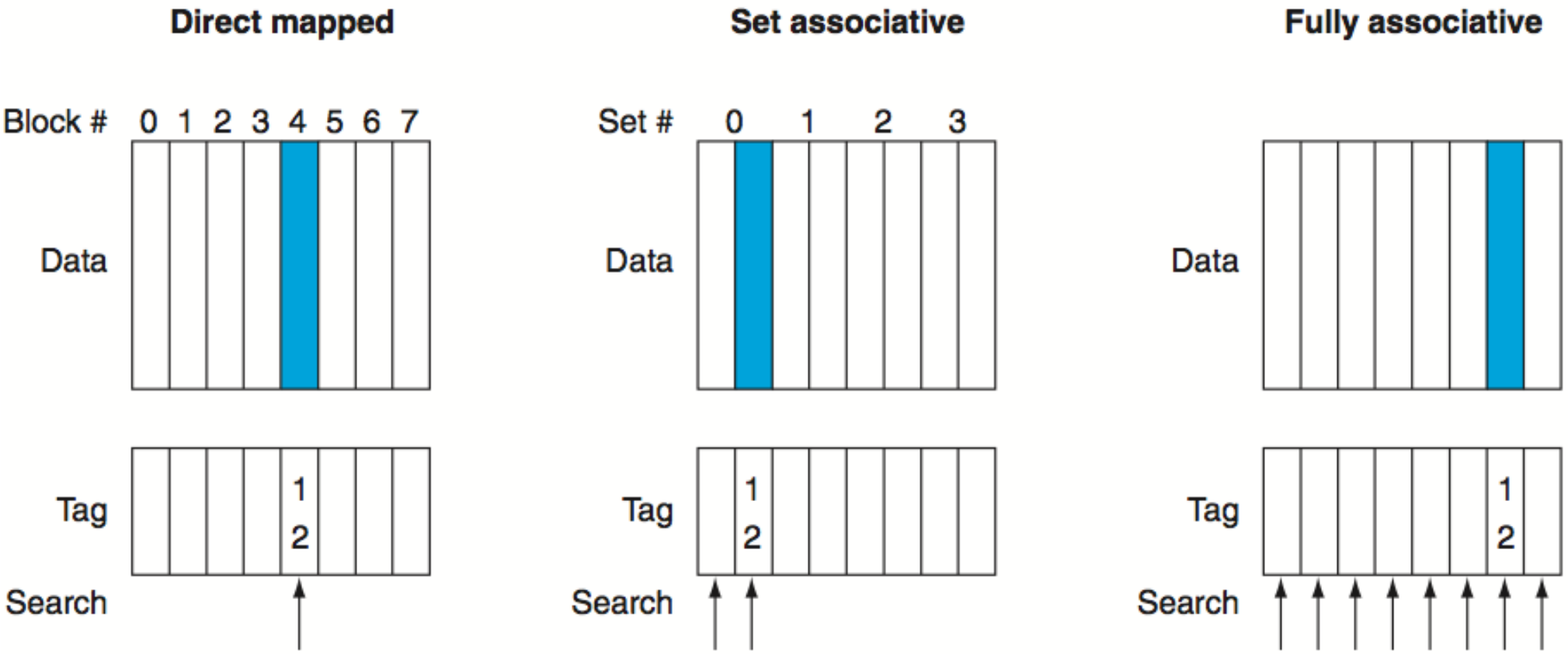
1 set,
8 blocks



fully associative

三种映射方式比较

主存块号为12的块在cache中的位置



三种映射方式比较

- 直接映射
  - 主存中的一块只能映射到Cache中唯一的一个位置 定位时,不需要判断,只需替换
- 全相联映射
  - 主存中的一块可以映射到Cache中任何一个位置
- N路组相联映射
  - 主存中的一块可以选择映射到Cache中N个位置
- 全相联映射和N路组相联映射的失效处理
  - 从主存中取出新块
  - 为了腾出Cache空间,需要替换出一个Cache块
  - 不唯一,则需要判断应替出哪块

思考题

- 某计算机的 Cache 共有 16 行,采用 2 路组相连映射方式。每个主存块大小为 32字节,按字节编址。主存 129 号单元所在主存块应转入到 Cache 组号是( )
- A. 0
- B. 2
- C. 4
- D. 6

思考题

- 某计算机的 Cache 共有 16 行,采用 2 路组相连映射方式。每个主存块大小为 32字节,按字节编址。主存 129 号单元所在主存块应转入到 Cache 组号是( )
- A. 0
- B. 2
- C. 4
- D. 6

提高cache的性能

- 平均访问时间 =命中时间 x 命中率

  + 失效损失 x 缺失率

- 提高命中率
- 缩短缺失时的访问时间
- 提高Cache本身的速度

Cache命中率

- 增加Cache的目的
  - 在性能上使主存的平均读出时间尽可能接近cache的读出时间，即在所有的存储器访问中由cache满足CPU需要的部分应占很高的比例。
- 命中率
  - 在一个程序执行期间，设Nc表示cache完成存取的总次数，Nm表示主存完成存取的总次数，h定义为命中率，则有

$$h=Nc/(Nc+Nm)$$

  - 若tc表示命中时的cache访问时间，tm表示未命中时的主存访问时间，1-h表示未命中率，则cache/主存系统的平均访问时间ta为：

$$ta=h*tc+(1-h)tm$$

命中率对平均访问时间的影响

- 平均访问时间ta，cache访问时间tc，主存访问时间tm，命中率h

$$ta=h*tc+(1-h)tm$$

- 例1. 若h=0.85, tc=1ns, tm=20ns，则ta为多少?

- 例2. 若命中率h提高到0.95，则结果又如何?

- 例3. 若命中率h为0.99呢?

命中率对平均访问时间的影响

- 平均访问时间ta，cache访问时间tc，主存访问时间tm，命中率h

$$ta=h*tc+(1-h)tm$$

- 例1. 若h=0.85, tc=1ns, tm=20ns，则ta为多少?
- 答： ta = 4ns

这里计算失误，三道题目答案分别为3.85 1.95 1.19
可以四舍五入，但是第一题很明显不对

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

- 例3. 若命中率h为0.99呢?
- 答: ta = 1.2ns

命中率对平均访问时间的影响

- 平均访问时间ta，cache访问时间tc，主存访问时间tm，命中率h

$$ta=h*tc+(1-h)tm$$

- 例1. 若h=0.85, tc=1ns, tm=20ns，则ta为多少?
- 答： ta = 4ns

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

- 例3. 若命中率h为0.99呢?
- 答: ta = 1.2ns

访存速度与命中率的关系非常大!

Cache缺失的四大原因

- 必然缺失(Compulsory Miss)
  - 开机或者是进程切换
  - 首次访问数据块
- 容量缺失(Capacity Miss)
  - 活动数据集超出了Cache的大小
- 冲突缺失(Conflict Miss)
  - 多个内存块映射到同一Cache块
  - 某一Cache组块已满,但空闲的Cache块在其他组
- 无效缺失
  - 其他进程修改了主存数据

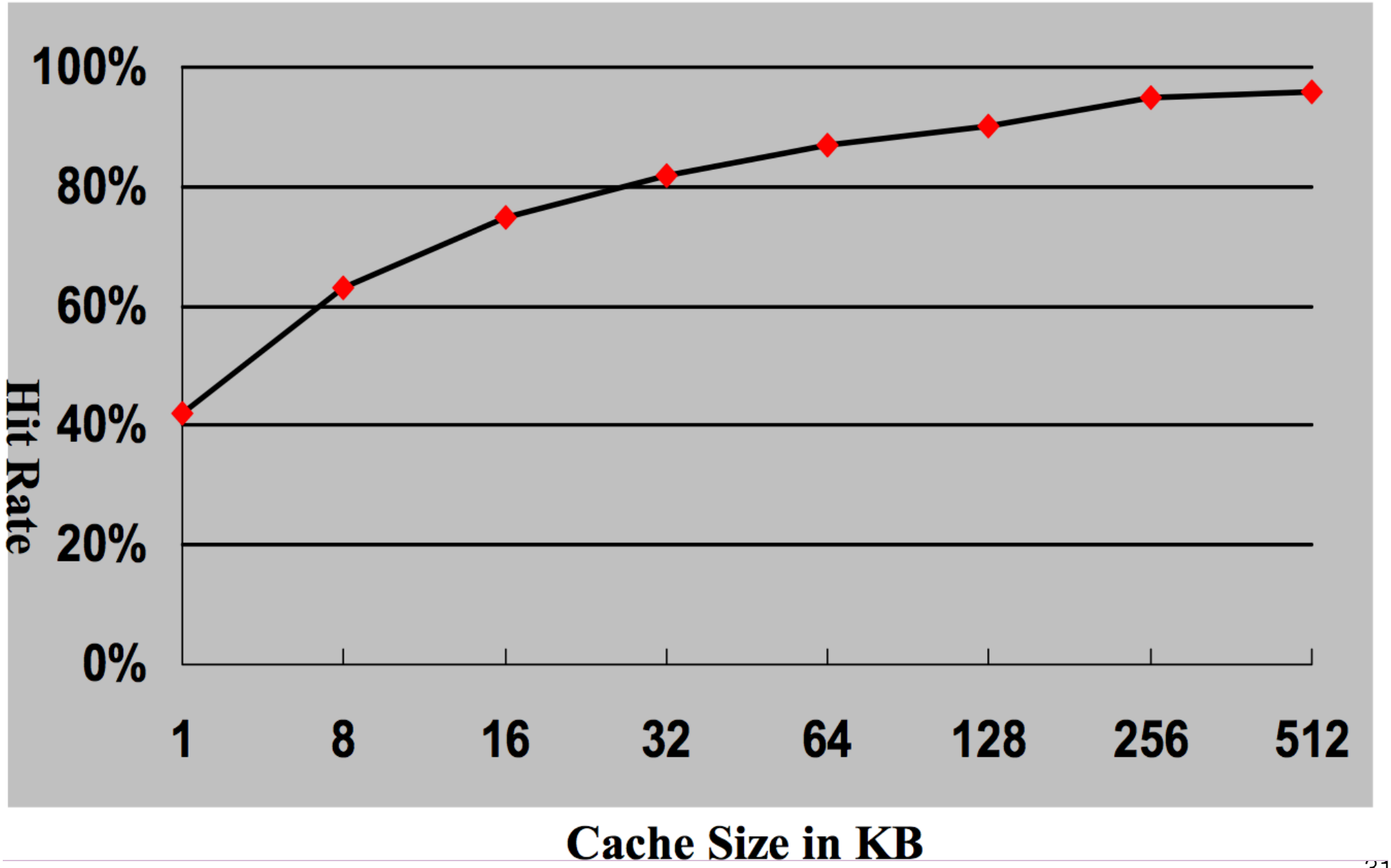
处理缺失的方法

- 必然缺失
  - 预取
- 容量缺失
  - 出现在Cache容量太小的时候
  - 增加Cache容量,可缓解缺失现象
- 冲突缺失
  - 两块不同的内存块映射到相同的Cache块
  - 对直接映射的Cache,这个问题尤其突出
    - 增加Cache容量有助于缓解冲突
    - 增加相联的组数有助于缓解冲突

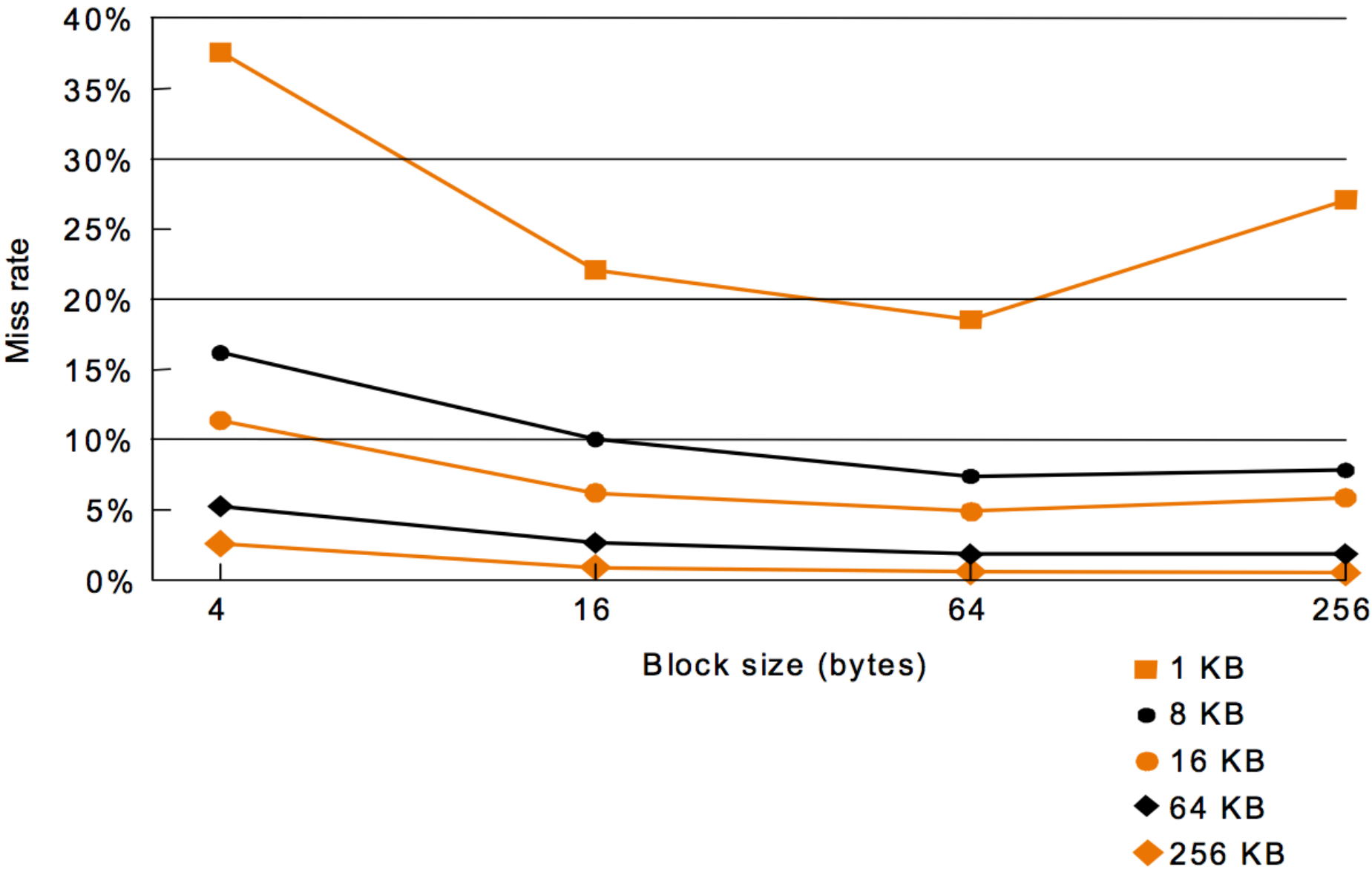
影响cache缺失率的因素

- CACHE 的容量大，命中率高
- CACHE 与主存储器每次交换信息的单位量(Cache Line Size)适中
- CACHE 不同的组织方式，多路组相联更好
- CACHE 的多级组织可提高命中率
- CACHE 装满后的替换策略

Cache命中率与容量的关系



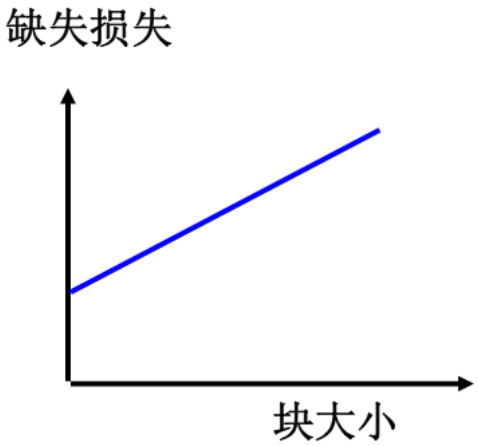
Cache缺失率和块大小的关系

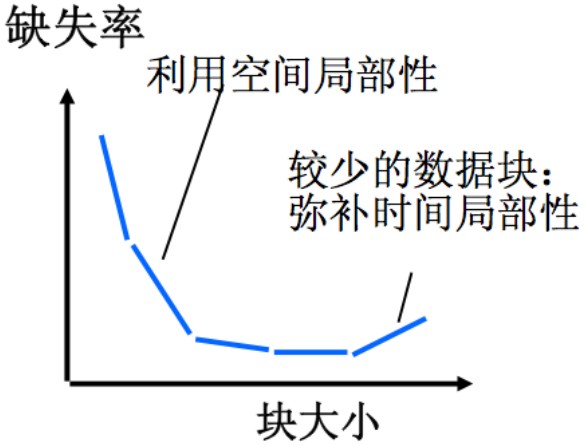


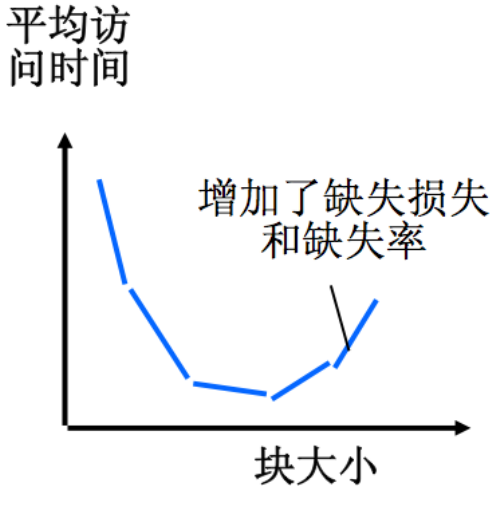
块大小的权衡

- 数据块较大可以更好地利用空间局部性
  - 数据块大意味着缺失损失的增大
    - 需要花费更长的时间来装入数据块
  - 若块大小相对Cache总容量来说太大的话，命中率将降低
    - Cache块数太少

平均访问时间 = 命中时间 × 命中率 + 失效损失 × 缺失率







关联度对cache缺失率的影响

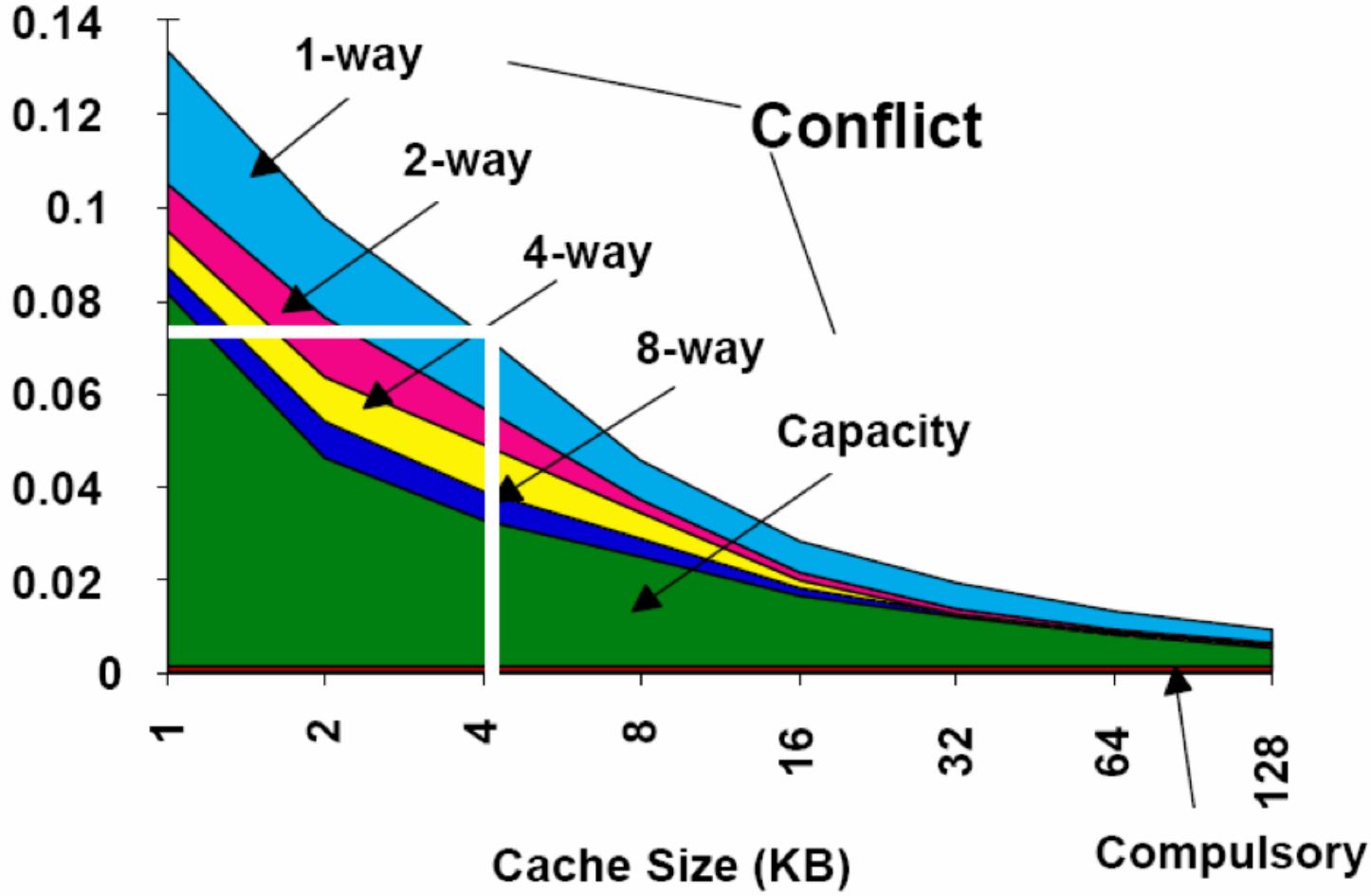
- 关联度：一个主存块映射到Cache中时，可能存放的位置个数
  - 直接映射：关联度最低，为1
  - 全相联映射：关联度最高，为Cache行数
  - N-路组相联映射：关联度居中，为N
- 关联度和缺失率有什么关系呢？和命中时间的关系呢？
  - 直观上，你的结论是什么？（Cache大小和块大小一定时）

关联度对cache缺失率的影响

- 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
- 关联度和缺失率有什么关系呢？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）
  - 关联度越高（低），缺失率越低（高）；
  - 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

组织方式对cache缺失率的影响

- 经验总结:容量为N、采用直接映射方式Cache的缺失率和容量为N/2、采用2路组相联映射方式Cache的缺失率相当



缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

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

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

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]			

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

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]			

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

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]	

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

直接映射方式

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

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]	

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

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]			

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

2路组相联映射方式

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

全相联映射方式

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]			

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

全相联映射方式

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

全相联映射方式

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]		

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

全相联映射方式

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]	

缺失率与关联度的关系

- 例：4行cache，块大小为1个字，需要按顺序访问主存中的块：0，8，0，6，8

全相联映射方式

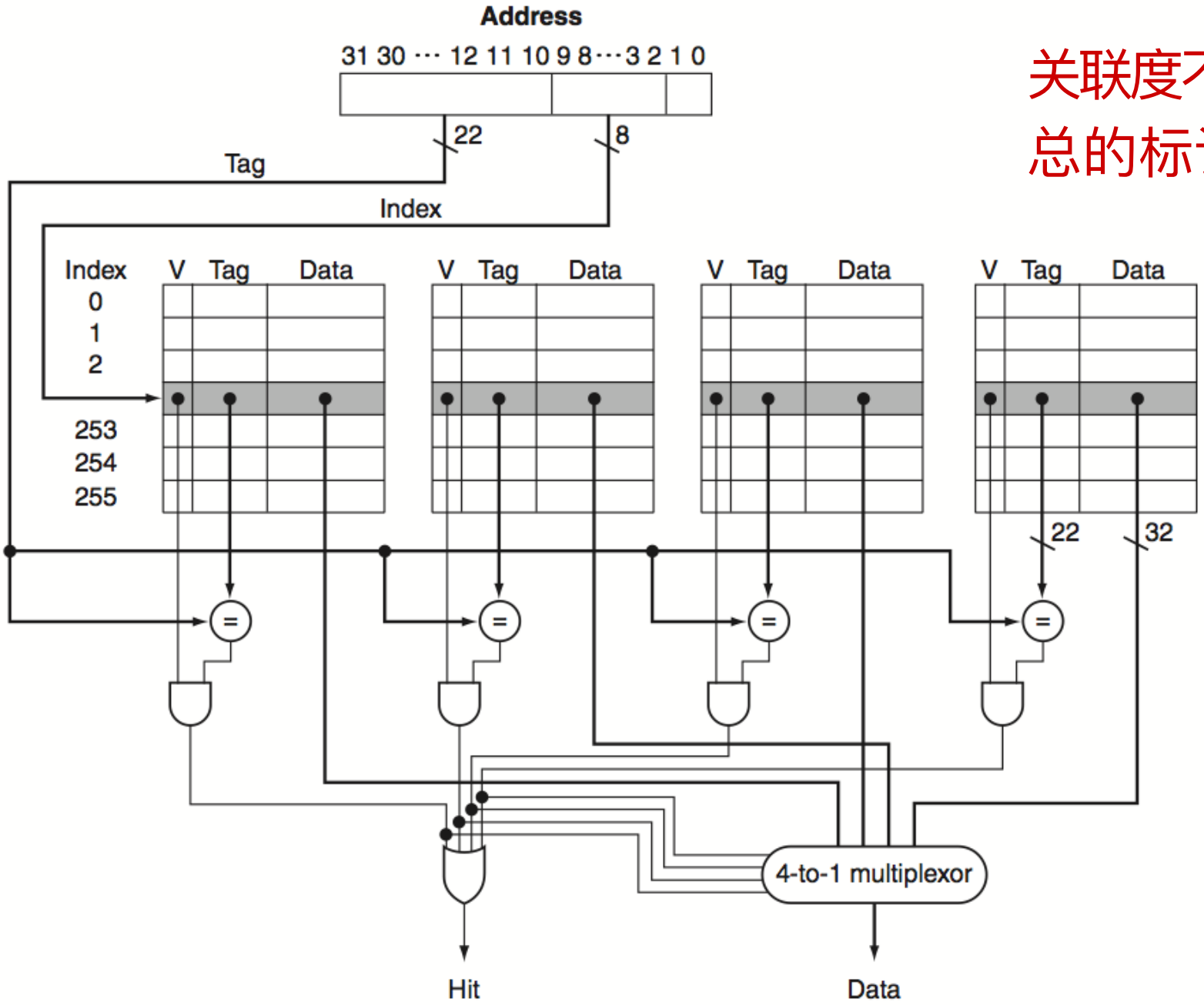
Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

缺失率与关联度的关系

- 使用内置FastMATH处理器相似的cache结构，关联度从一路到八路，采用SPEC2000基准测试程序测出的cache缺失率

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

缺失率与关联度的关系



关联度不同，比较电路、总的标记位数不同

提纲

- 高速缓冲存储器Cache
  - Cache内容装入和替换策略
  - 改写主存储器的策略
  - 多级缓冲存储器
- 虚拟存储器
  - 虚拟存储器基本概念
  - 虚拟地址到物理地址的转换
  - 虚拟存储器的页式管理

使用Cache需要解决的问题

- 地址之间的映射关系:
  - 如何从主存地址得到Cache地址?
- Cache内容装入和替换策略
  - 如何提高Cache的命中率?
- 数据之间一致性:
  - Cache中的内容是否已经是主存对应地址的内容?

何时需要替换

- 直接映射
  - 映射唯一，毫无选择，将现有数据块从cache中踢出，为新的数据块留出存放空间；不需要考虑替换策略；
- N路组相联映射
  - 每个主存数据有N个Cache行可选择，需考虑替换策略；
- 全相联映射
  - 每个主存数据可存放到Cache任意行中，需考虑替换策略；
- 结论：若N路组相联或全相联映射cache缺失，则需要替换，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

Cache的替换算法

- 先进先出算法(FIFO)
  - 将最早调入Cache的字块替换出去。容易实现,开销小。
- 最近最少使用算法(LRU)
  - 需要计算字块的使用次数,开销大,但平均命中率比FIFO要高。
- 最不经常用（LFU）
- 随机替换(RAND)

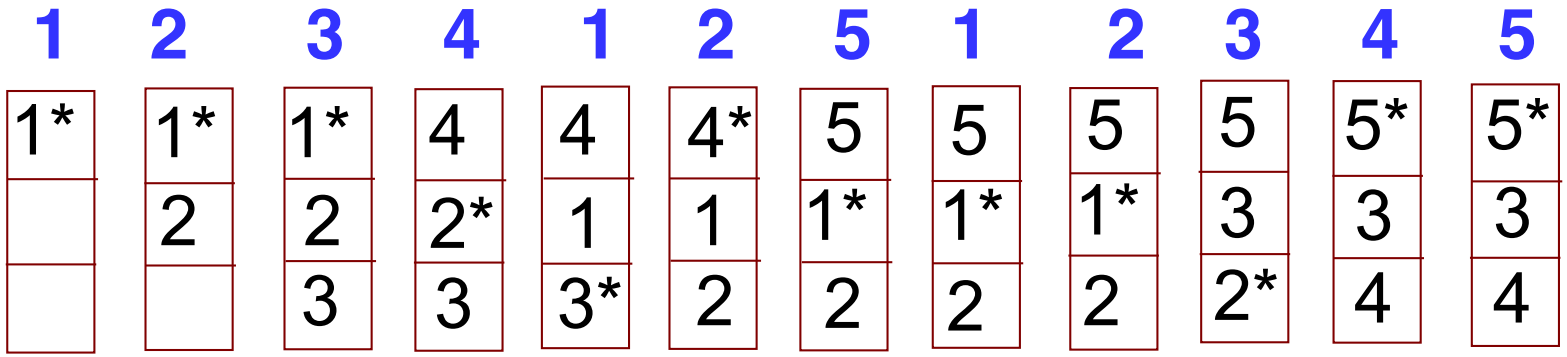
替换算法-先进先出（FIFO）

- 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。

替换算法-先进先出（FIFO）

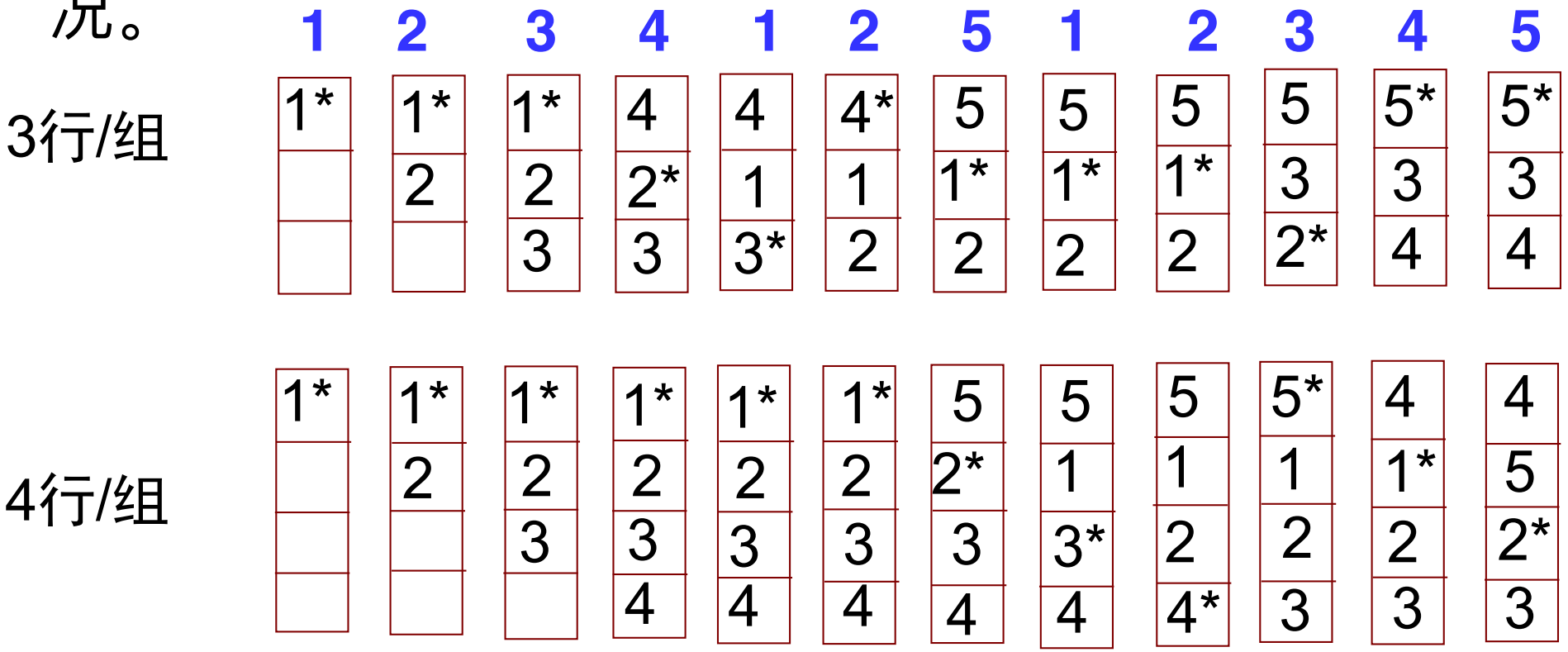
- 总是把最先进入的那一块淘汰掉。
- 例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组的情况。

3行/组



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。



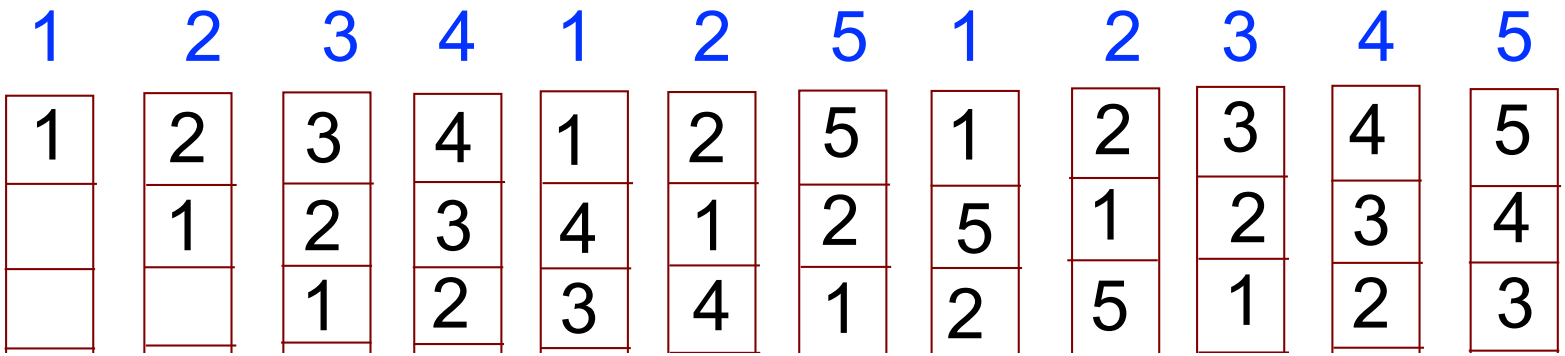
替换算法-最近最少用(LRU)

- 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(LRU)

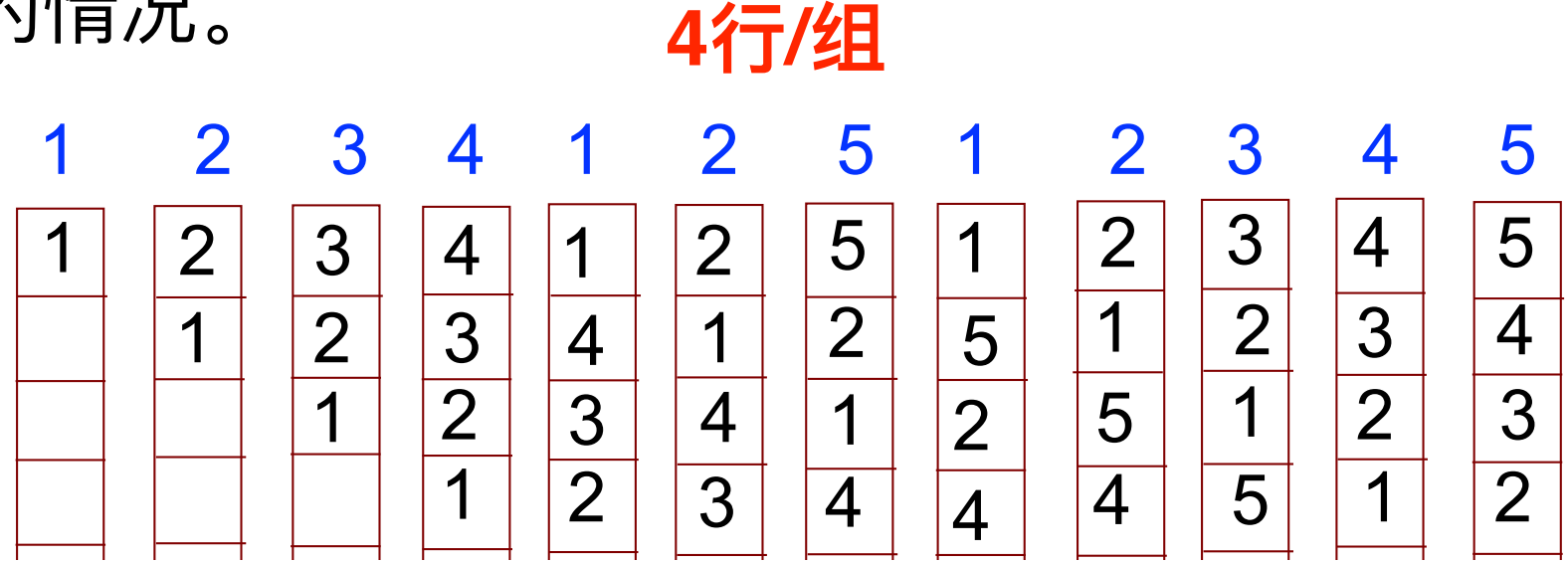
- 总是把最近最少用的那一块淘汰掉。
- 例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、4行/组、5行/组的情况。

3行/组



替换算法-最近最少用(LRU)

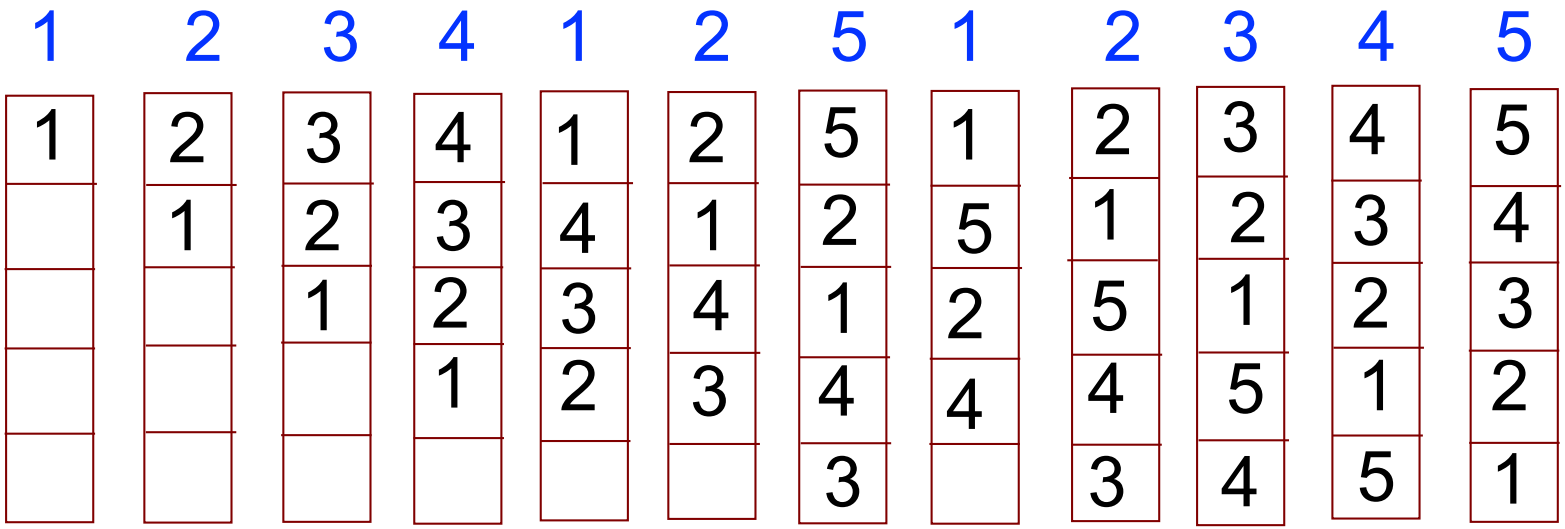
- 总是把最近最少用的那一块淘汰掉。
- 例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、4行/组、5行/组的情况。



替换算法-最近最少用(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。

5行/组



替换算法-最近最少用(LRU)

- 它的命中率随组的增大而提高。
- 当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。

替换算法-最近最少用(LRU)

- 计数器变化规则：
  - 计数值越小则说明越被常用。
  - 命中时，被访问行的计数器置0，比余计数器加1。
  - 未命中且该组未满时，新行计数器置为0，其余全加1。
  - 未命中且该组已满时，计数值为3（最大）的那一行中的主存块被淘汰，新行计数器置为0，其余加1。

1	2	3	4	1	2	5	1	2	3	4	5

0	1	1	1	2	1	3	1	0	1	1	1	2	1	0	1	1	1	2	1	3	1	0	5
		0	2	1	2	2	2	3	2	0	2	1	2	2	2	0	2	1	2	2	2	3	4
				0	3	1	3	2	3	3	3	0	5	1	5	2	5	3	5	0	4	2	3
						0	4	1	4	2	4	3	4	3	4	3	4	0	3	1	3	1	2

其他替换算法

- 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！

替换算法

- 例子：设cache有1、2、3、4共4个块，a、b、c、d等为主存中的块,访问顺序一次如下：a、b、c、d、b、b、c、c、d、d、a ,下次若要再访问e块。
  - 问，采用LFU和LRU算法替换结果是不是相同?

替换算法

	LFU（最不经常使用）					LRU（近期最少使用）				
	说明	1块	2块	3块	4块	说明	1块	2块	3块	4块
a	a进入	1	0	0	0	a进入	0	1	1	1
b	b进入	1	1	0	0	b进入	1	0	2	2
c	c进入	1	1	1	0	c进入	2	1	0	3
d	d进入	1	1	1	1	d进入	3	2	1	0
b	命中	1	2	1	1	命中	4	0	2	1
b	命中	1	3	1	1	命中	5	0	3	2
c	命中	1	3	2	1	命中	6	1	0	3
c	命中	1	3	3	1	命中	7	2	0	4
d	命中	1	3	3	2	命中	8	3	1	0
d	命中	1	3	3	3	命中	9	4	2	0
a	命中	2	3	3	3	命中	0	5	3	1
e	替换a	1	0	0	0	替换b	1	0	4	2

使用Cache需要解决的问题

- 地址之间的映射关系:
  - 如何从主存地址得到Cache地址?
- Cache内容装入和替换策略
  - 如何提高Cache的命中率?
- 数据之间一致性:
  - Cache中的内容是否已经是主存对应地址的内容?

改写主存储器的策略

Write-hit policy

Write-through:

Write-back: needs a ***dirty bit***

Write-miss policy

Write-allocate:

No-write-allocate:

Typical caches:

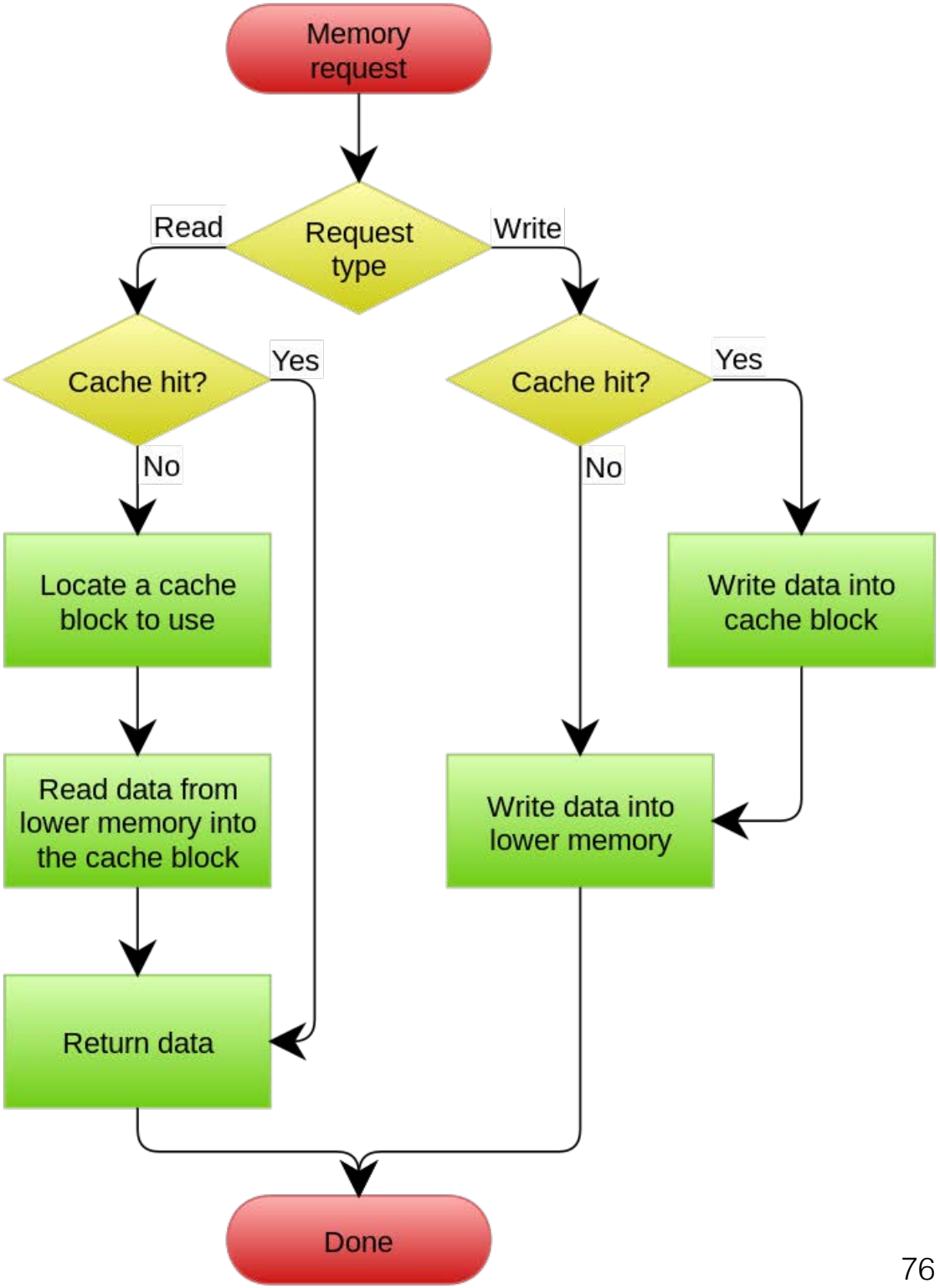
Write-back + Write-allocate, usually

Write-through + No-write-allocate, occasionally

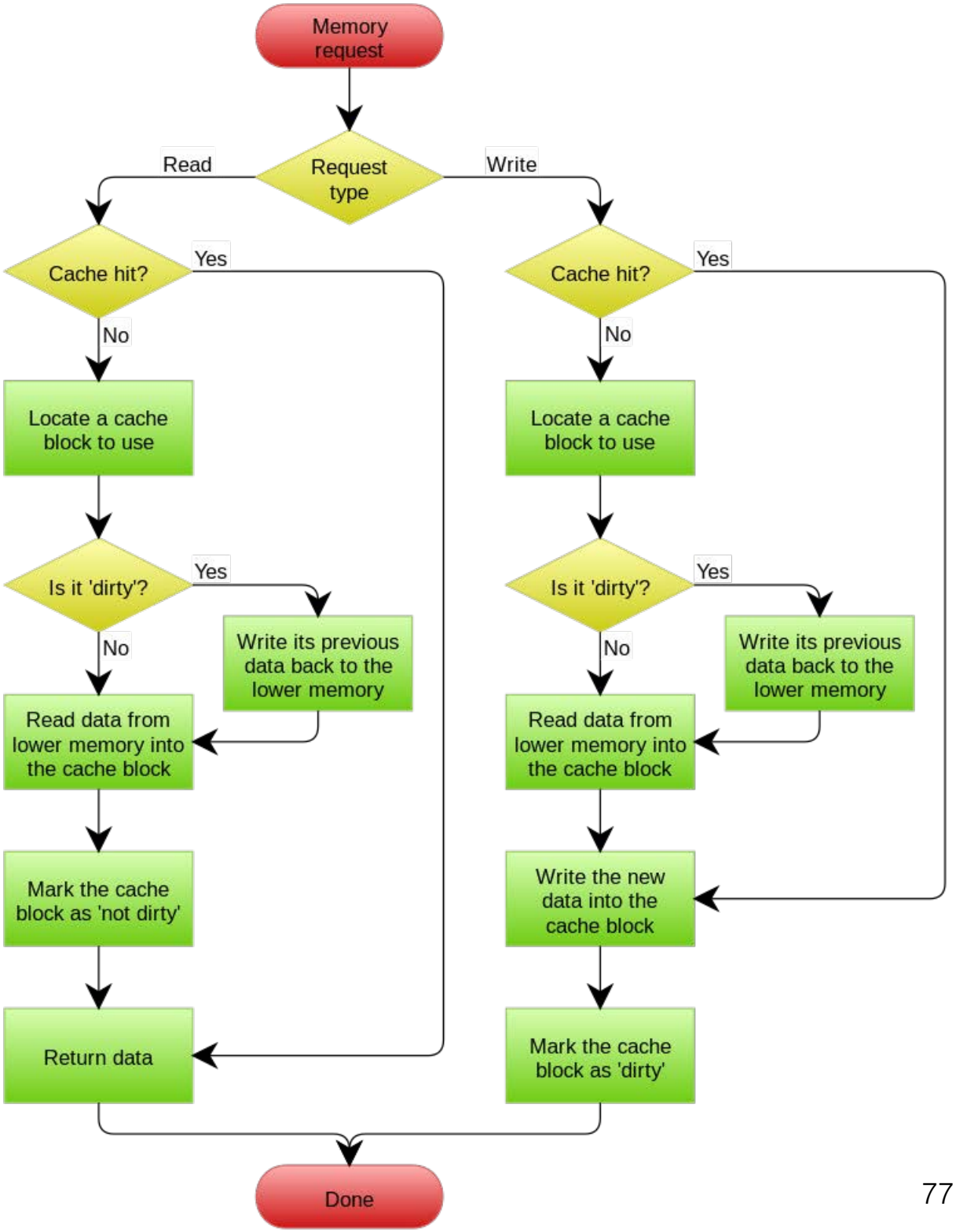
改写主存储器的策略

- 若 CPU 改写了cache一单元内容后且尚未改变主存相应单元内容,则出现数据不一致性。有两种解决办法:
  - 接下来直接改写主存单元内容(Write Through)
    - 简便易行, 但可能带来系统运行效率不高的问题。需要写入缓冲存储器。
  - 拖后改写主存单元内容(Write Back)
    - 一直拖到有另外的设备要读该内容过时的主存单元时,则首先停止这一读操作,接下来改写主存内容,之后再启动 已停下来的读操作。
    - 控制复杂些,但可以提供更高的系统运行效率。

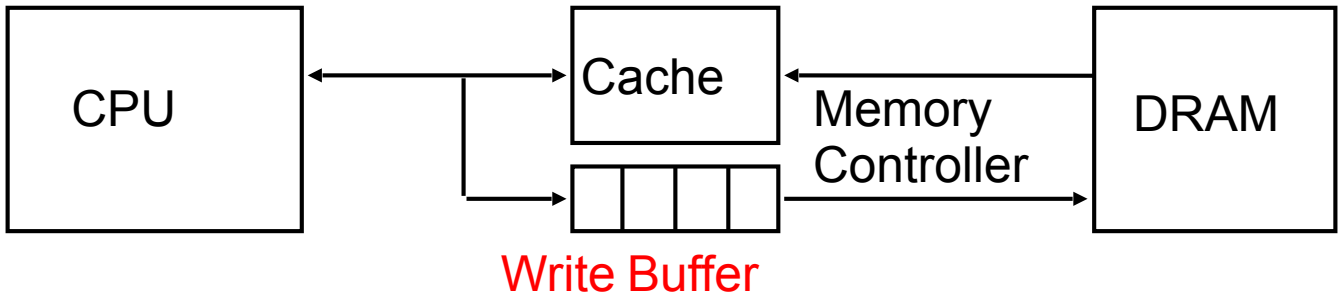
write-through
+
Non-write-allocate



write-back
+
write-allocate

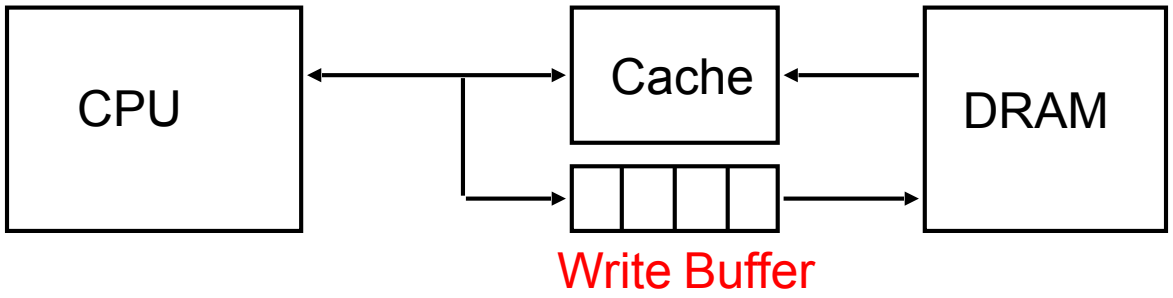


Write Through中的Write Buffer



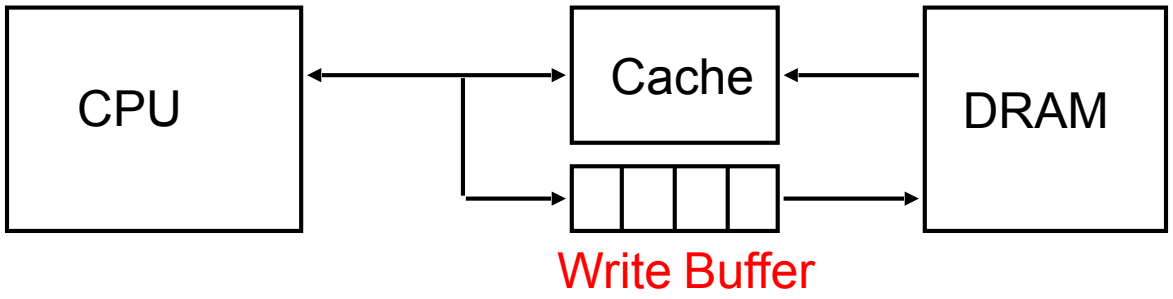
- 在 Cache 和 Memory之间加一个Write Buffer
  - CPU: 同时写数据到Cache和Write Buffer
  - Memory controller（存控）: 将缓冲内容写主存
- Write buffer (写缓冲) 是一个FIFO队列
  - 一般有4项
  - 在存数频率<<DRAM写（周期）频率情况下，效果好
- 最棘手的问题
  - Store frequency > 1 / DRAM write cycle(频繁写)时，使Write buffer 饱和(溢出)，会发生阻塞

写缓冲饱和

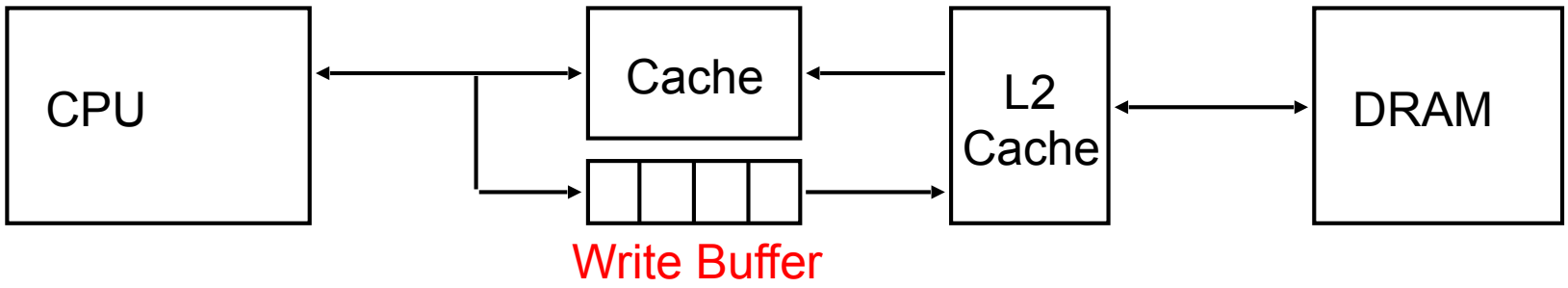


- 发生写缓冲饱和的可能性
  - CPU时钟周期 < DRAM写周期 (客观上如此)
  - 存数频率 >> 1/ DRAM写周期 (发生频繁写)
- 如何解决写缓冲饱和?

写缓冲饱和



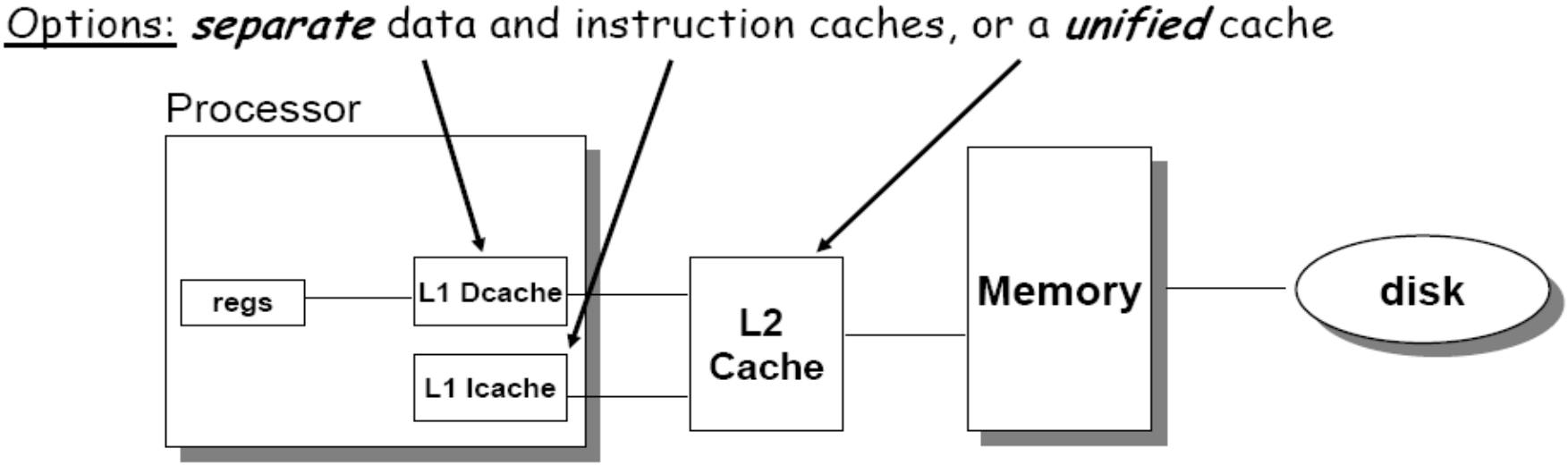
- 发生写缓冲饱和的可能性
  - CPU时钟周期 < DRAM写周期 (客观上如此)
  - 存数频率 >> 1/ DRAM写周期 (发生频繁写)
- 如何解决写缓冲饱和?
  - 加一个二级Cache
  - 使用Write Back方式的Cache



多级Cache

- 采用两级或更多级cache来提高命中率
  - 增加Cache层次
  - 减少访存次数，当L1未命中时，可以从L2中获取信息
- 将Cache分解为指令Cache和数据Cache
  - 指令流水的现实要求
  - 根据具体情况,选用不同的组织方式、容量

多级Cache



Inclusive vs. Exclusive

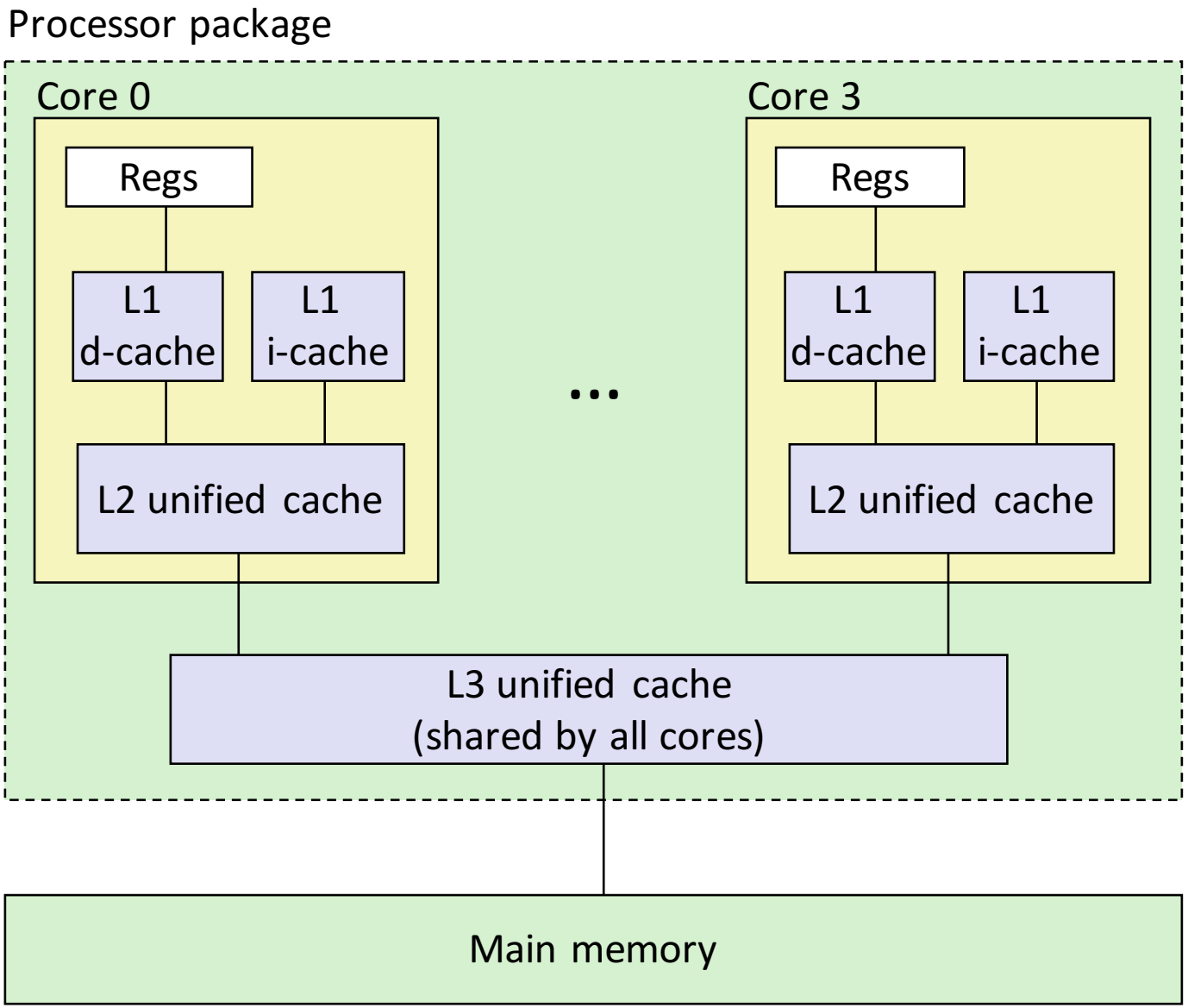
- 在处理器中设置独立的数据Cache和指令Cache
- 在处理器外设置第二级Cache,甚至是第三级Cache
- 分立Cache特别适用于如Pentium和PowerPC的超标量机器中。这些机器采用指令并行执行和指令预取技术。这样，减少了在指令译码处理和指令执行阶段对Cache的竞争。流水线方式下这点尤为重要。

多级Cache的性能

- 采用L2 Cache的系统，其缺失损失的计算如下：
  - 若L2 Cache包含所请求信息，则缺失损失为L2 Cache的访问时间
  - 否则，要访问主存，并取到L1 Cache和L2 Cache（缺失损失更大）

典型的多级高速缓冲存储器

Typical laptop/desktop processor
(always changing)



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

slower, but more likely to hit

多级Cache的性能

- 例子：某处理器在无cache缺失时CPI为1，时钟频率为5GHz。假定访问一次主存的时间（包括所有的缺失处理）为100ns，平均每条指令在L1 Cache中的缺失率为2%。若增加一个L2 Cache，其访问时间为5ns，而且容量足够大到使全局缺失率减为0.5%，问处理器执行指令的速度提高了多少?

多级Cache的性能

- 例子：某处理器在无cache缺失时CPI为1，时钟频率为5GHz。假定访问一次主存的时间（包括所有的缺失处理）为100ns，平均每条指令在L1 Cache中的缺失率为2%。若增加一个L2 Cache，其访问时间为5ns，而且容量足够大到使全局缺失率减为0.5%，问处理器执行指令的速度提高了多少?
- 解：
  - 如果只有一级Cache，则缺失只有一种。即L1缺失(需访问主存)，其缺失损失为：100nsx5GHz=500个时钟，CPI=1+500x2%=11.0。
  - 如果有二级Cache，则有两种缺失：
    - L1缺失(需访问L2 Cache)：5nsx5GHz=25个时钟
    - L1和L2都缺失(需访问主存)：500个时钟
  - 因此，CPI=1+25x2%+500x0.5%=4.0
  - 二者的性能比为11.0/4.0=2.8倍！

程序设计时如何利用Cache

- 编写具有局部性特点的程序
  - 空间局部性
    - 按顺序访问数据
  - 时间局部性
    - 确保要访问的数据集中在短时间内访问
- 如何做?
  - 选择合适的算法
  - 选择合适的循环方式

例子（cache采用直接映射方式）

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

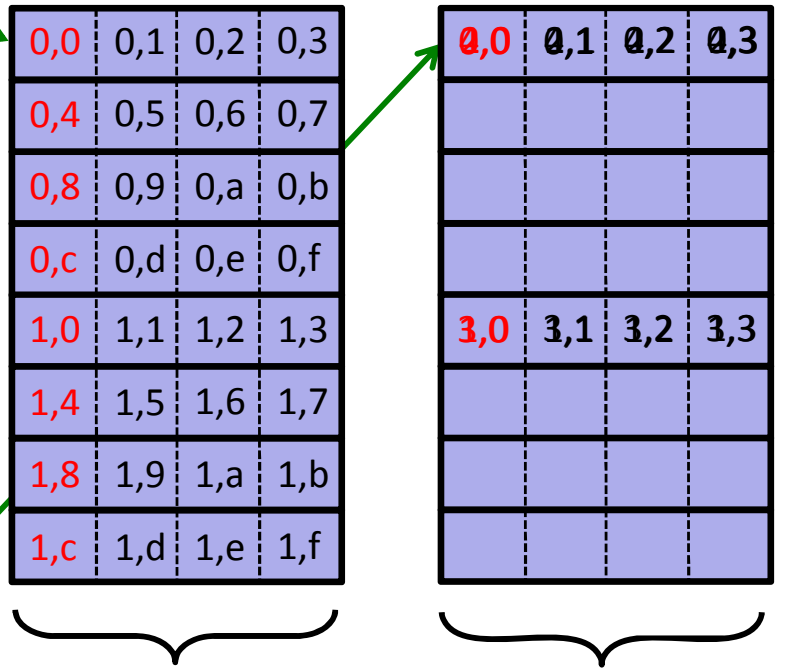
例子（cache采用直接映射方式）

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Assume: cold (empty) cache
3 bits for set, 5 bits for byte

aa….aaxxx xyy yy000



32 B = 4 doubles

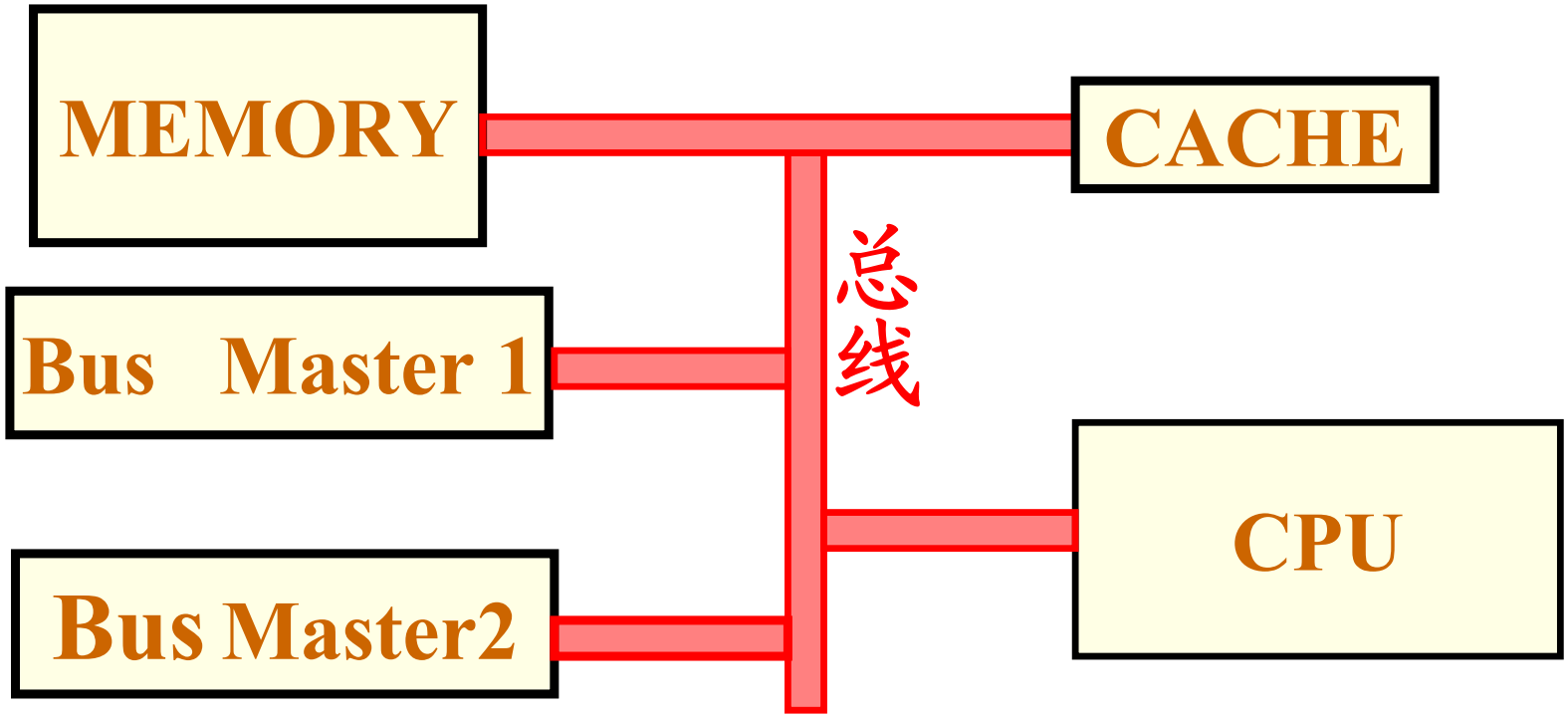
4 misses per row of array
4*16 = 64 misses

32 B = 4 doubles

every access a miss
16*16 = 256 misses

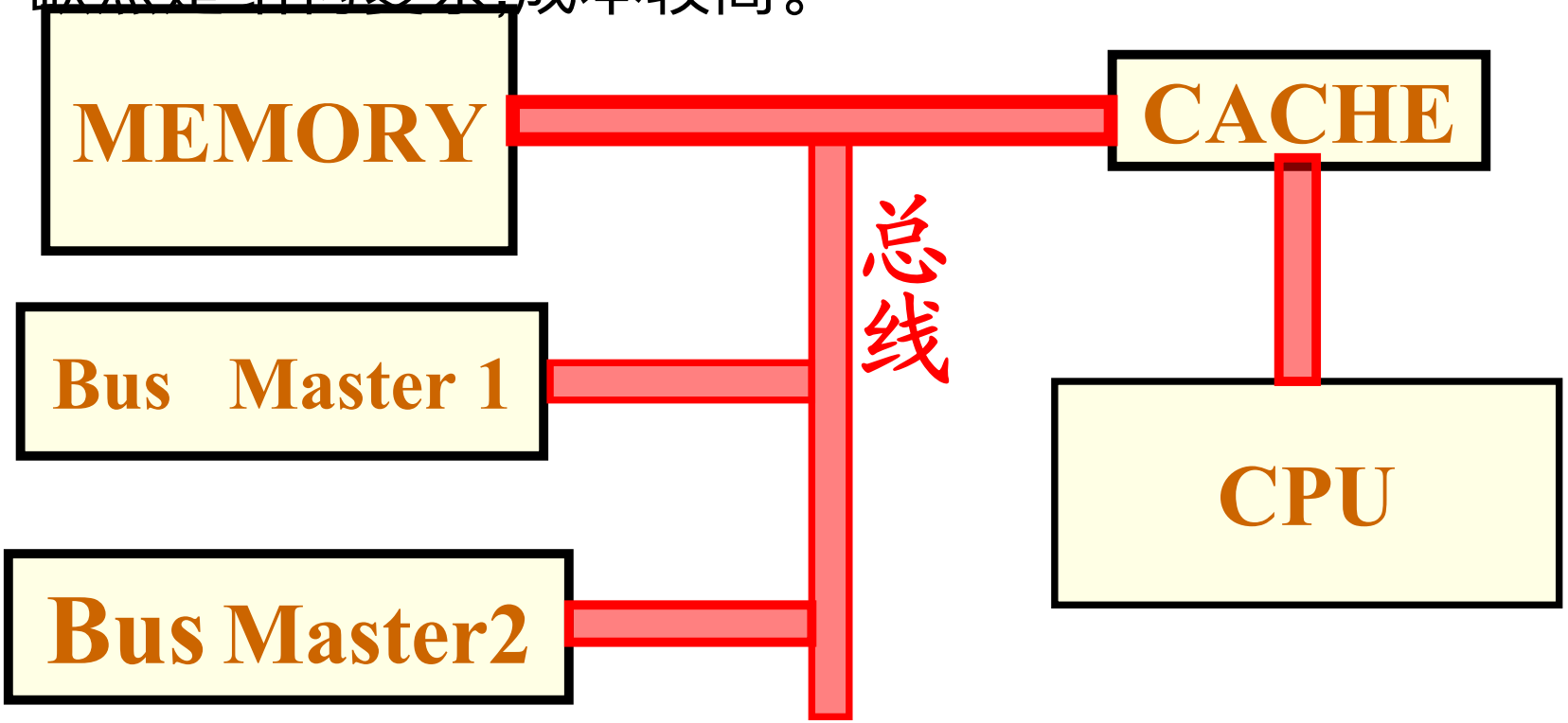
Cache接入系统的体系结构

- 侧接法:像输入输出设备似的连接到总线上
  - 优点是结构简单,成本低,
  - 缺点是不利于降低总线占用率。



Cache接入系统的体系结构

- 隔断法:把原来的总线打断为两段,使CACHE处在两段之间
  - 优点是有利于提高总线利用率,支持总线并发操作
  - 缺点是结构复杂,成本较高。



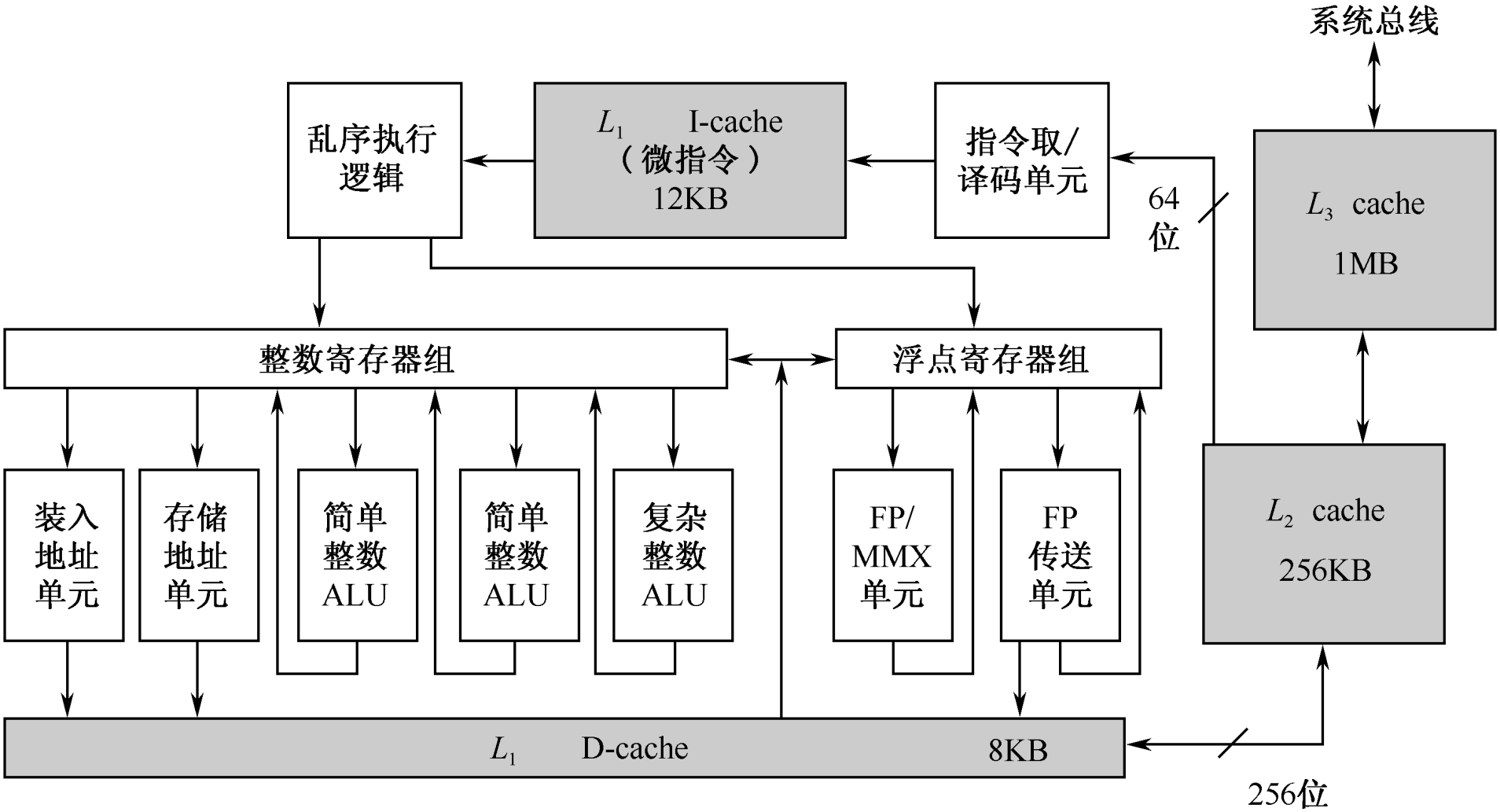
Pentium上的Cache

- Intel的80386及更早的处理器芯片上并没有片上的高速缓冲存储器。
- 80486:引入一个8K字节的片上Cache,其块大小为16字节,采用的是四路组相连映象方式。
- Pentium:芯片上已经有了两个8K字节的片上 cache,一个用作指令缓存,另一个用作数据缓存。它们的块大小都为32字节,采用两路组相连的组织方式。
- Pentium2包含L2 cache，256KB，每行128B，8路组相连结构
- Pentium3增加了一个L3 cache
- Pentium4将L3 cache移到处理器芯片内部

Pentium的数据缓存

- 数据缓存由128组,每组两个cache块组成,从逻辑上可以看成是大小为4K字节的两“路”。
- 每个cache块都有20位的标记位和两位的状态位与其对应,标记位即存放在该cache块中的主存块的地址的高20位,两位状态位可以标记出4个不同的块状态。
- 替换算法采用的是最近最少使用法(LRU),所以对每组cache块还需要有1位的LRU位来表示CPU最近访问的是该组中的哪一块。
- 采用拖后写策略。支持两级Cache。

Pentium 4三级cache布局



Cache小结

- 目标
  - 提高CPU访问存储器系统的平均速度
- 策略
  - 利用一容量较小(降低成本)的高速缓冲存储器
- 组织方式
  - 全相连、直接映射、组相连

Cache小结

- 包含性保证
  - cache中的块和主存中的块进行映射
- 一致性保证
  - 写回主存策略
- 提高命中率
  - 块替换算法

Cache小结

- 局部性原理:
  - 任何时候,程序需要访问的只是相对较小的一些地址空间。
    - 时间局部性
    - 空间局部性
- 三类主要的Cache缺失原因:
  - 必然缺失:如:第一次装入
  - 冲突缺失:增大Cache容量、改进组织方式 避免不断的块冲突
  - 容量缺失:增大Cache容量
- Cache设计
  - 总容量、块大小、组织方式替换算法
  - 写策略(命中时):写直达、拖后写
  - 写策略(不命中时):是否装入到Cache?

设计Cache

- 有关方案
  - cache
  - 容量
  - 块大小
  - 组织方式
  - 替换算法
  - 写策略
- 方案优化
  - 根据用途选择
    - 指令数据平衡
    - 海量数据处理
  - 根据成本优化

本讲提纲

- 高速缓冲存储器Cache
  - Cache内容装入和替换策略
  - 改写主存储器的策略
- 虚拟存储器
  - 虚拟存储器基本概念
  - 虚拟地址到物理地址的转换
  - 虚拟存储器的页式管理