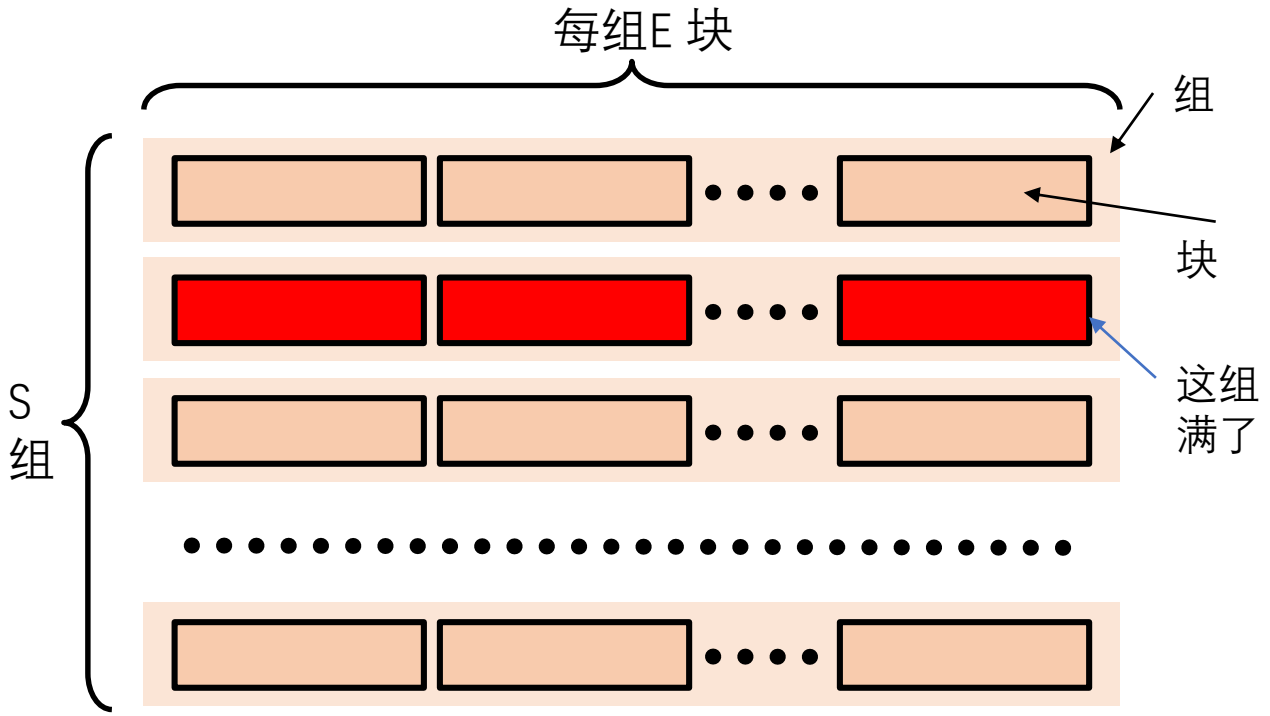
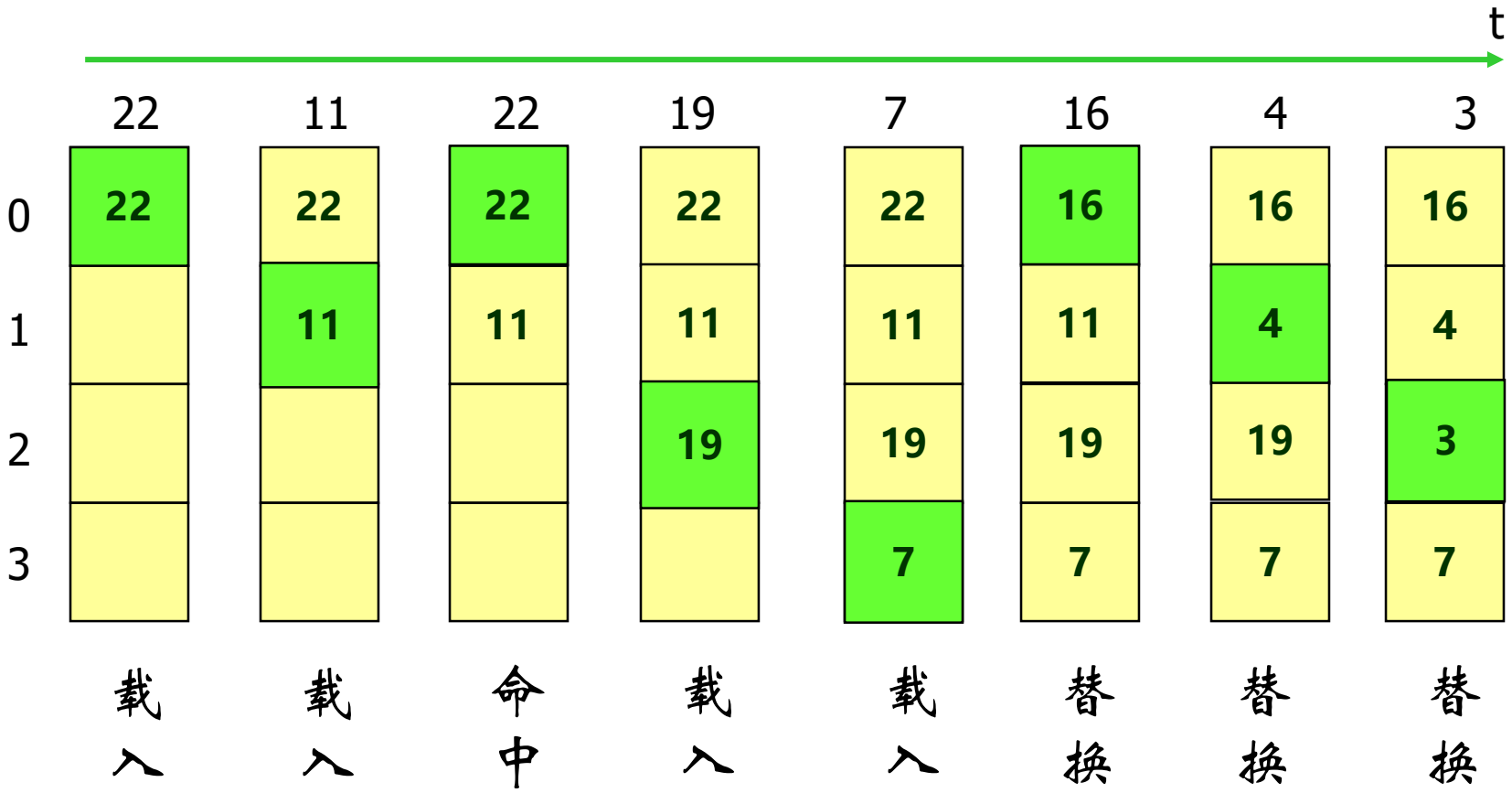
Cache 的替换策略

替换策略

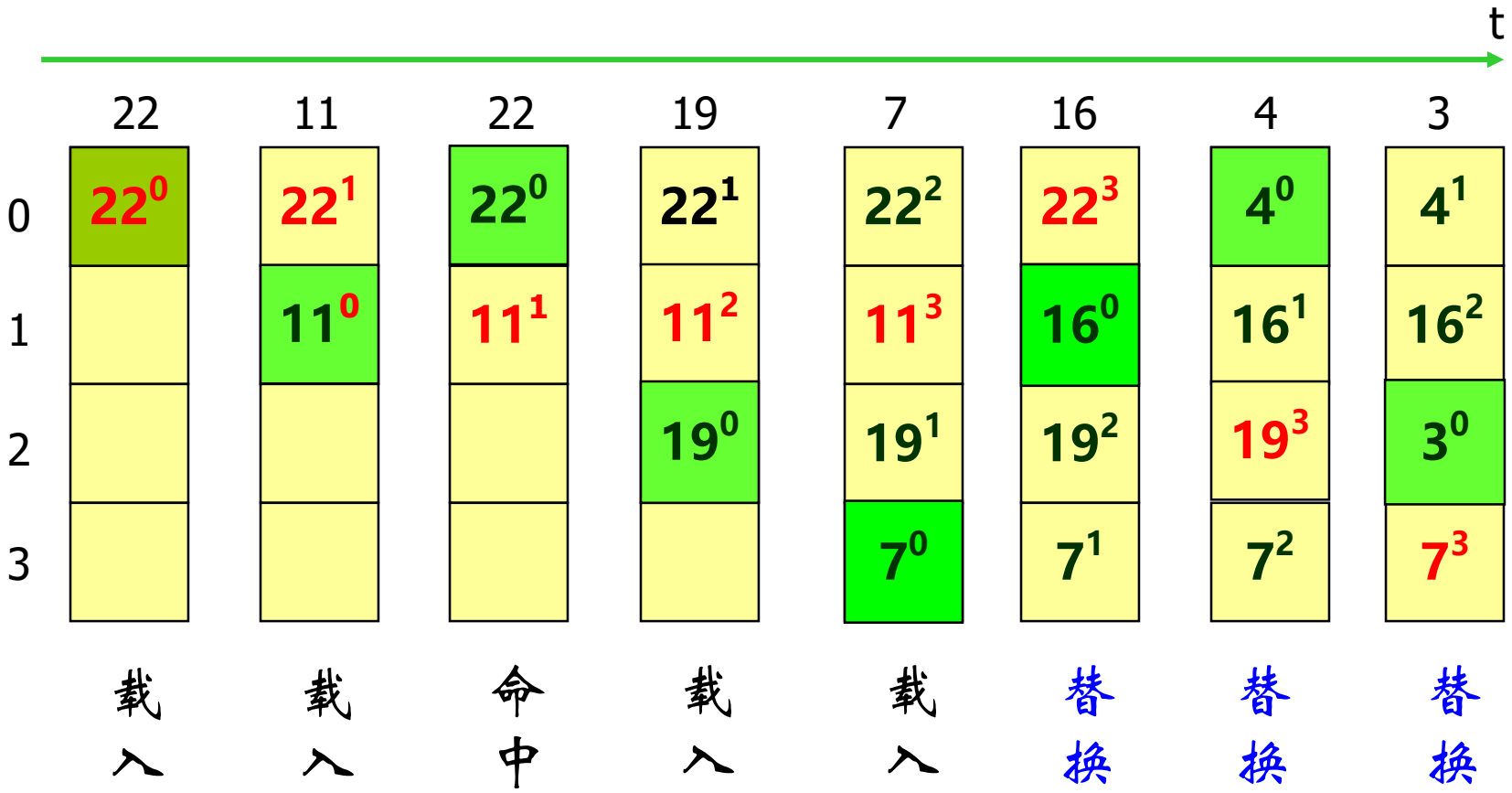
- 组相联映射cache，一个组（set）有E块，当E块都满时，替换哪一块？
  - 随机替换法
  - 先进先出法 FIFO
  - 近期最少使用法--- LRU
  - 伪LRU算法
  - 近期不常使用法---NMRU



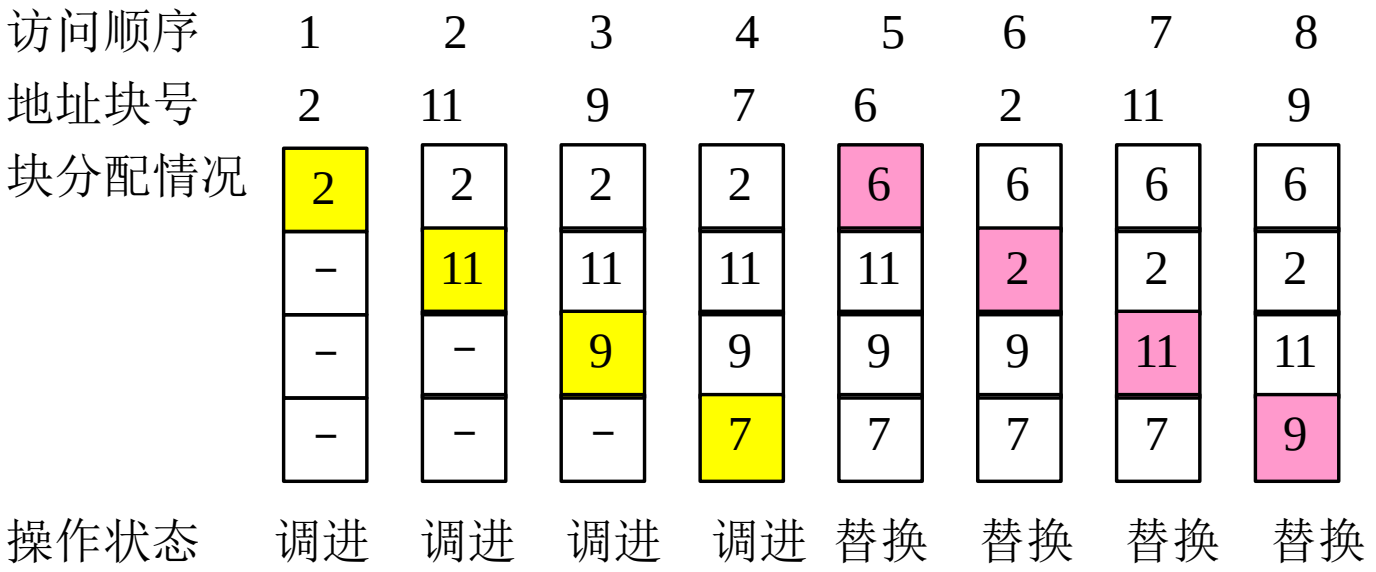
Cache先进先出替换策略(FIFO)



Cache近期最久未使用算法(LRU)

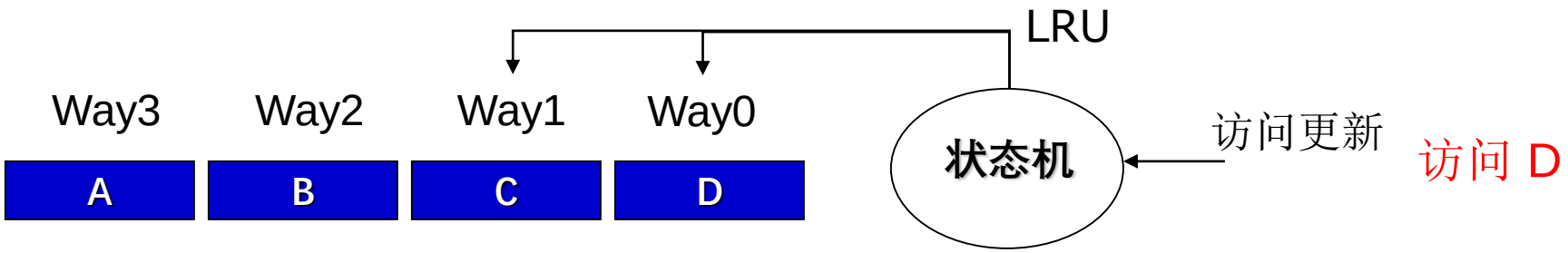


LRU 可能出现的问题：颠簸现象



根据局部性原理，LRU是有效的替换算法，但也可能会导致颠簸发生

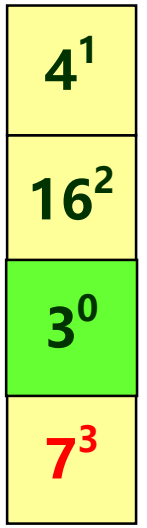
硬件如何实现LRU



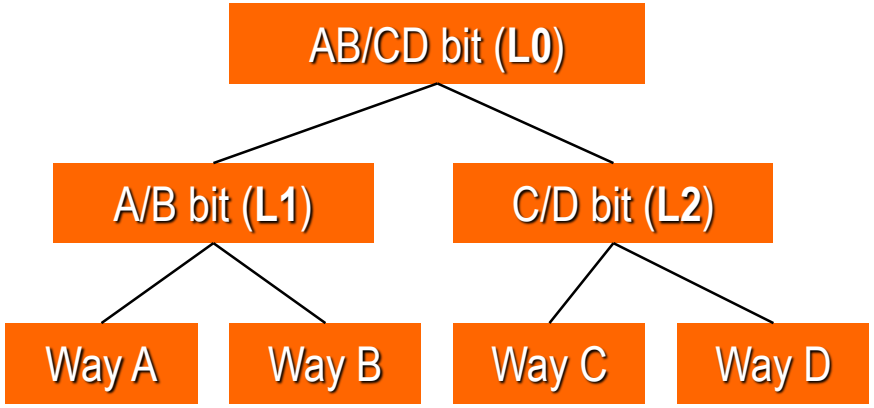
LRU 算法需要额外的硬件位来存储各块的状态，
LRU算法增加了cache 访问时间（ access times ）

近期最少使用算法(LRU)

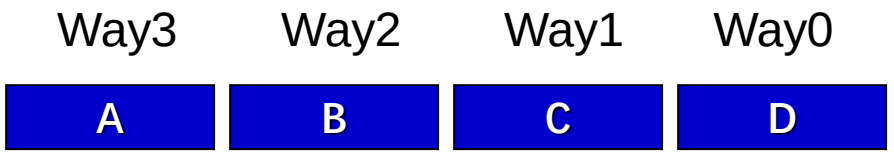
- LRU 的实现
  - 需要付出硬件代价
  - 要记住所有N行被访问的顺序
  - 共 N! 种情况
    - 2-ways →两种情况 AB BA = 2 = 2!
    - 3-ways → 六种情况 ABC ACB BAC BCA CAB CBA = 6 = 3!
    - 4-ways → 24情况 = 4!
- N! 种情况 需要 $O(\log N!)$ 位表达 $\approx O(N \log N)$ 位
- 例如：4路组相联，每组有4行，每一行必须增加两个状态位，总共增加了8个状态位，才能记住它们的访问顺序



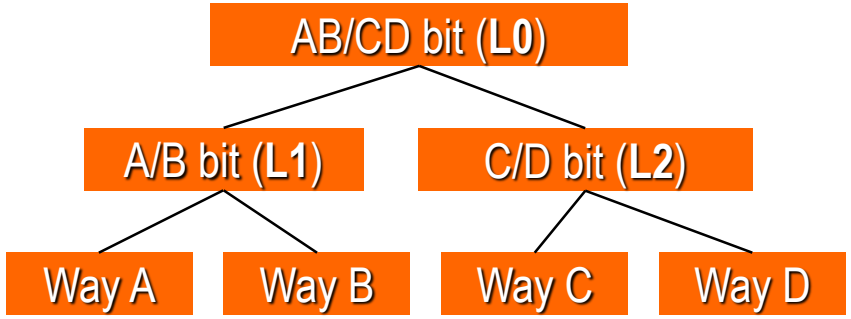
伪 LRU (4-way)



- 基于树状结构 Tree-based
- 硬件代价 O(N): 3 bits for 4-way



Pseudo LRU Algorithm



- Less hardware than LRU
- Faster than LRU

•L2L1L0 = 000, there is a hit in Way B, what is the new updated L2L1L0?

•L2L1L0 = 001, a way needs to be replaced, which way would be chosen?

LRU update algorithm

	CD	AB	AB/CD
Way hit	**L2**	**L1**	**L0**
Way A	---	1	1
Way B	---	0	1
Way C	1	---	0
Way D	0	---	0

Replacement Decision

CD	AB	AB/CD	
L2	**L1**	**L0**	**Way to replace**
X	0	0	Way A
X	1	0	Way B
0	X	1	Way C
1	X	1	Way D

小结

- 组关联映射cache：当一个set中所有的行都满，再将新的主存块映射到该组，就需要替换其中的一行
- 常用替换算法LRU
- 避免付出过多硬件代价的伪LRU算法