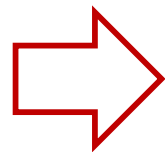


章节导学

实现双向链表



实现置换算法

先进先出算法

最近最少使用算法

最不经常使用算法

章节导学

先进先出算法(FIFO)

- ◆ 把高速缓存看做是一个先进先出的队列
- ◆ 优先替换最先进入队列的字块

章节导学

最近最少使用算法 (LRU)

- ◆ 优先淘汰一段时间内没有使用的字块
- ◆ 有多种实现方法，一般使用双向链表
- ◆ 把当前访问节点置于链表前面（保证链表头部节点是最近使用的）

章节导学

最不经常使用算法(LFU)

- ◆ 优先淘汰最不经常使用的字块
- ◆ 需要额外的空间记录字块的使用频率

章节导学

实现双向链表



实现Node类



实现DoubleLinkedList类

- ◆ pop()方法
- ◆ append(node)方法
- ◆ append_front(node)方法
- ◆ remove(node)方法

实现置换算法



实现Cache类

- ◆ get(key)方法: 返回value或-1
- ◆ put(key, value)方法



双向链表的原理与实践

单向链表：



每一个节点都有下一个节点的地址或引用

双向链表：



每一个节点都有上一个节点和下一个节点的地址或引用

双向链表的原理与实践

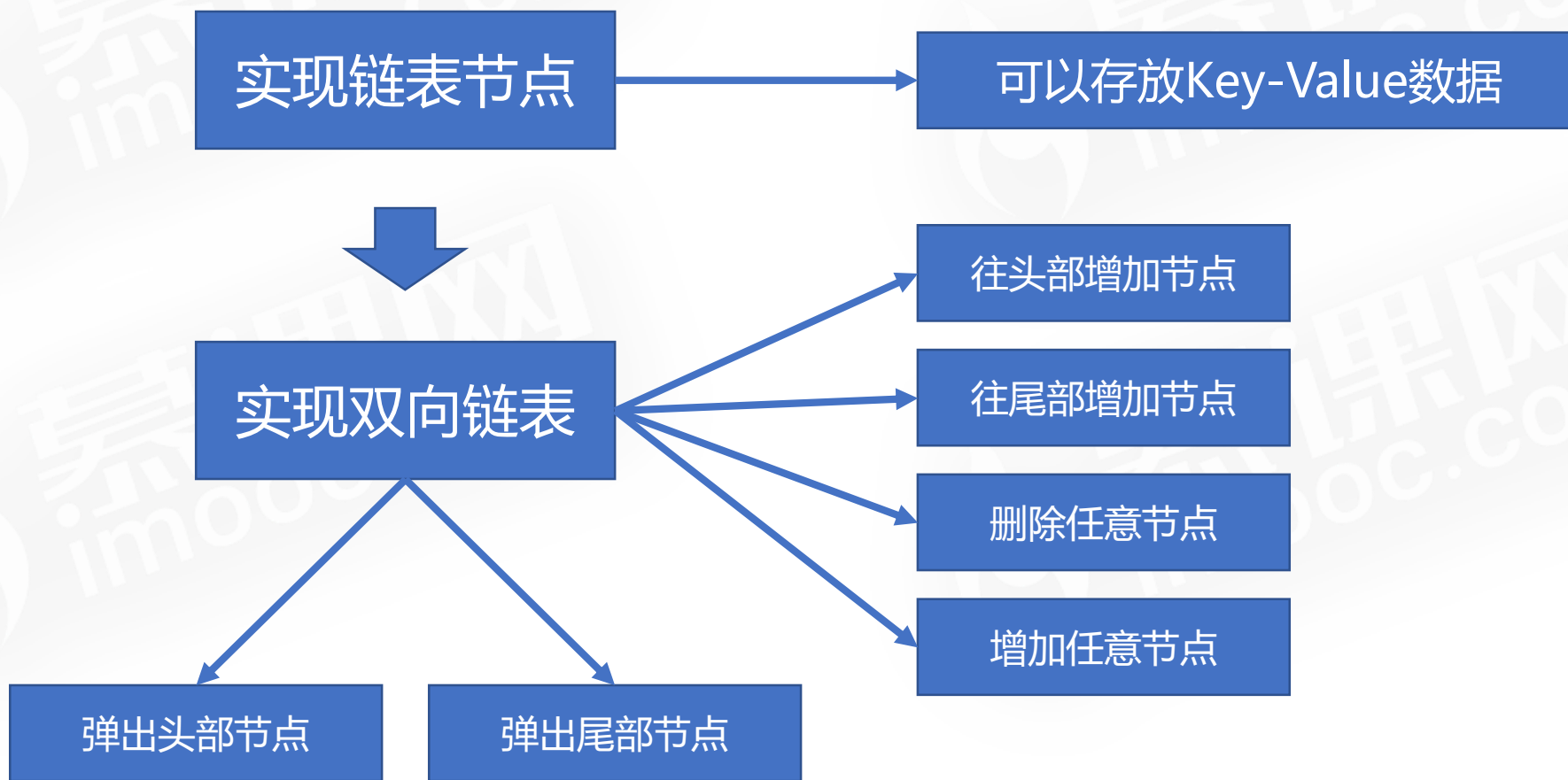
双向链表：



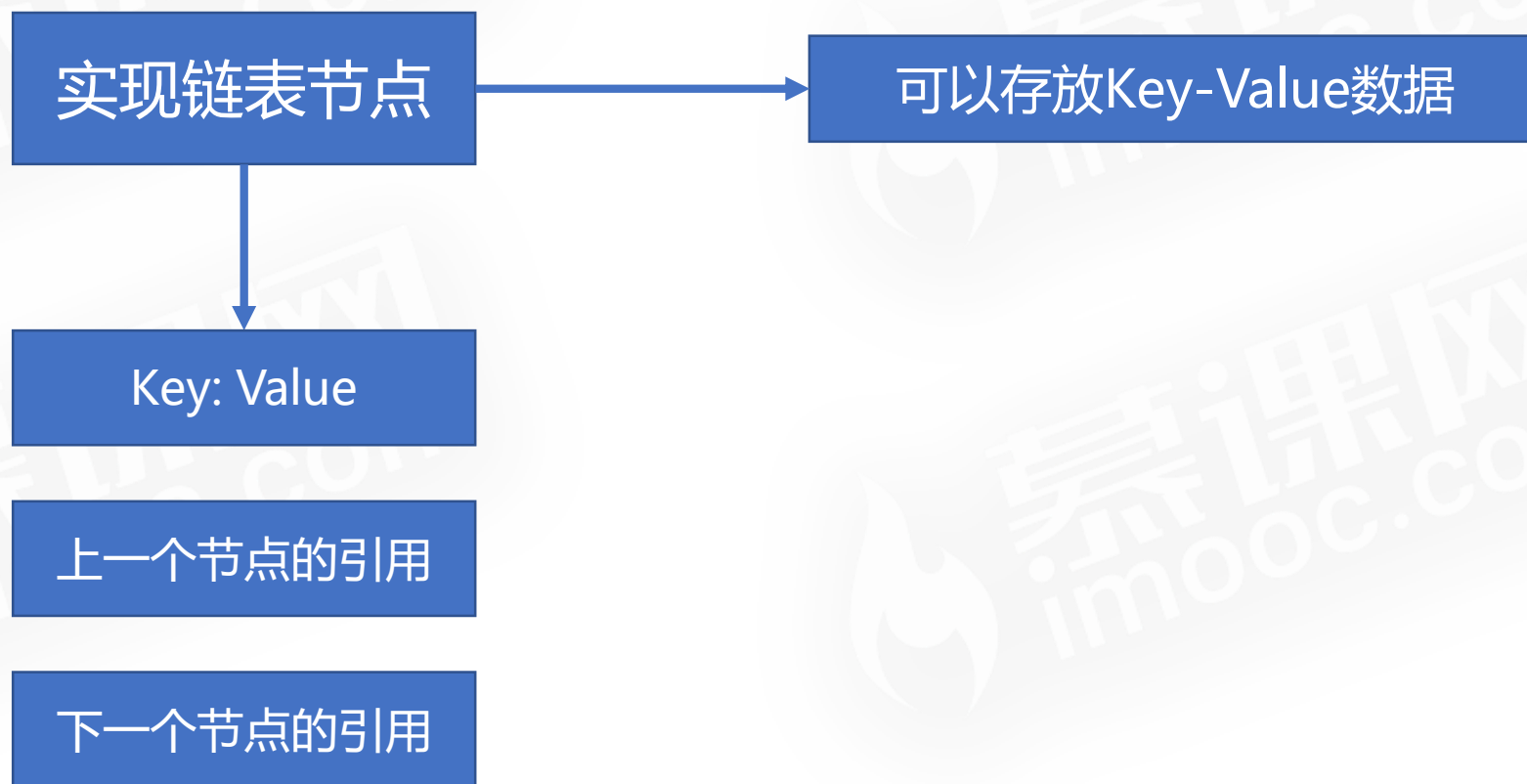
每一个节点都有上一个节点和下一个节点的地址或引用

- ◆ 可以快速找到一个节点的下一个节点
- ◆ 可以快速找到一个节点的上一个节点
- ◆ 可以快速去掉链表中的某一个节点

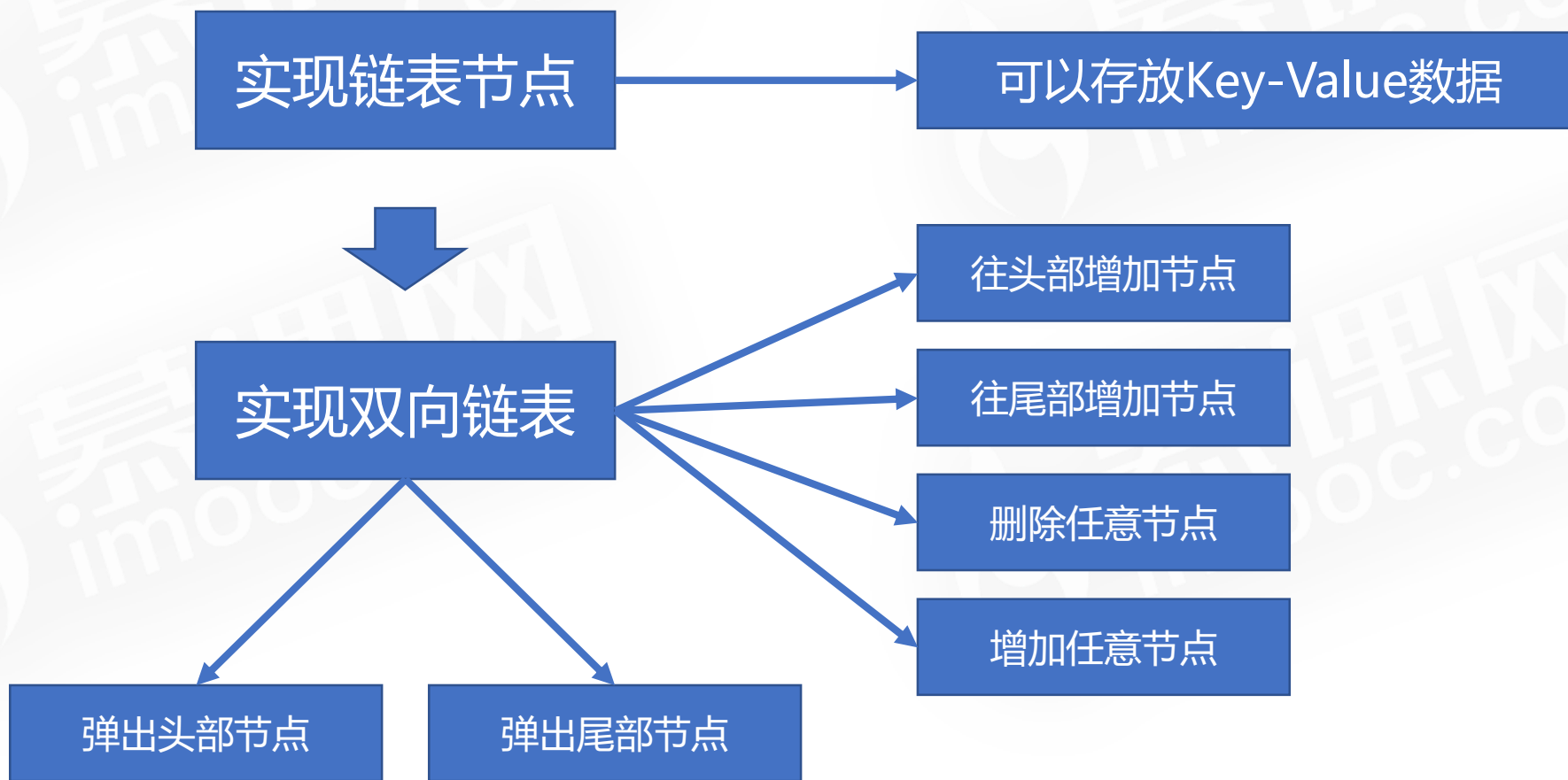
双向链表的原理与实践



双向链表的原理与实践

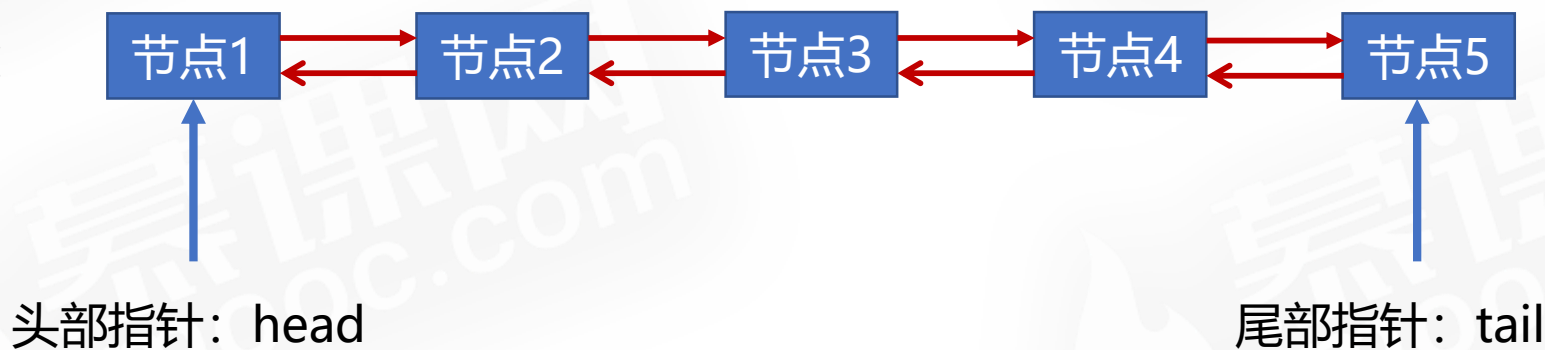


双向链表的原理与实践

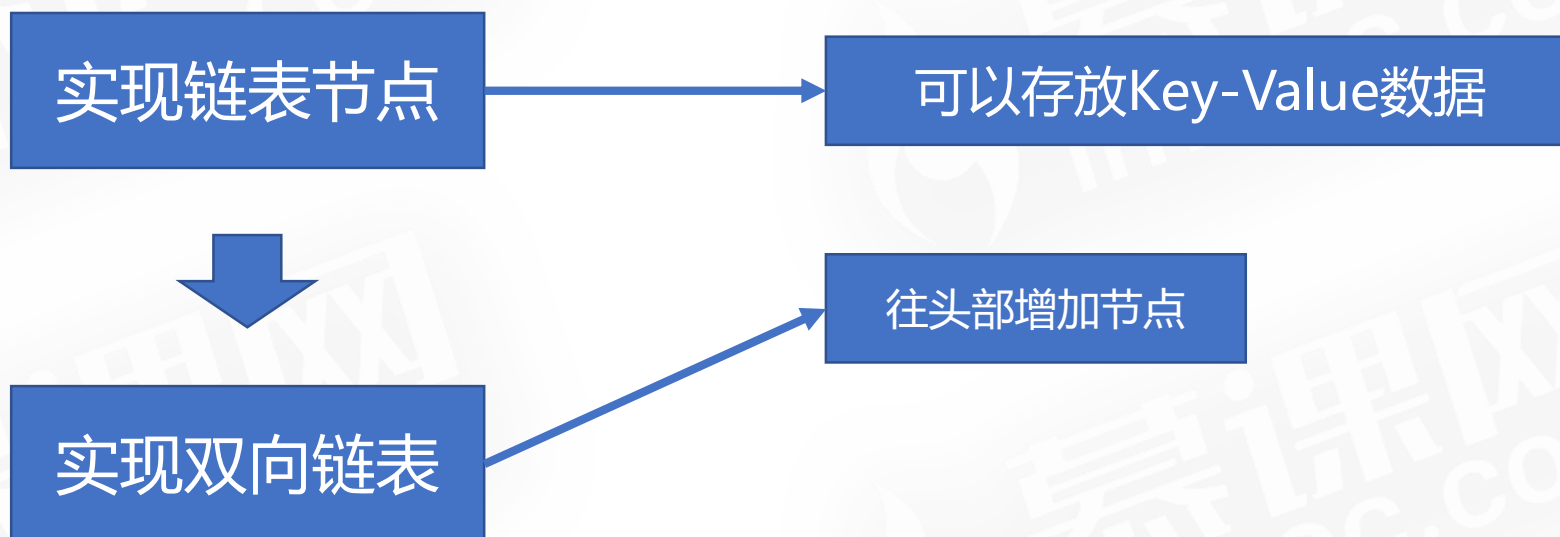


双向链表的原理与实践

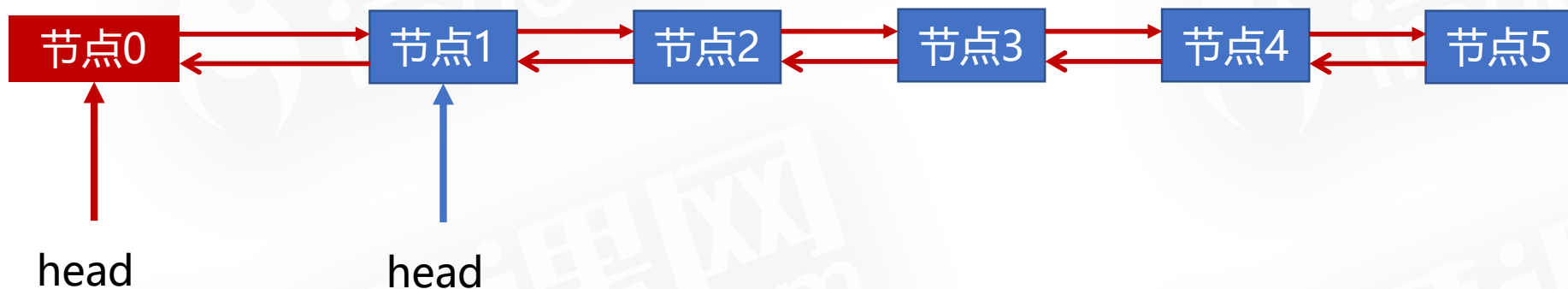
双向链表:



双向链表的原理与实践

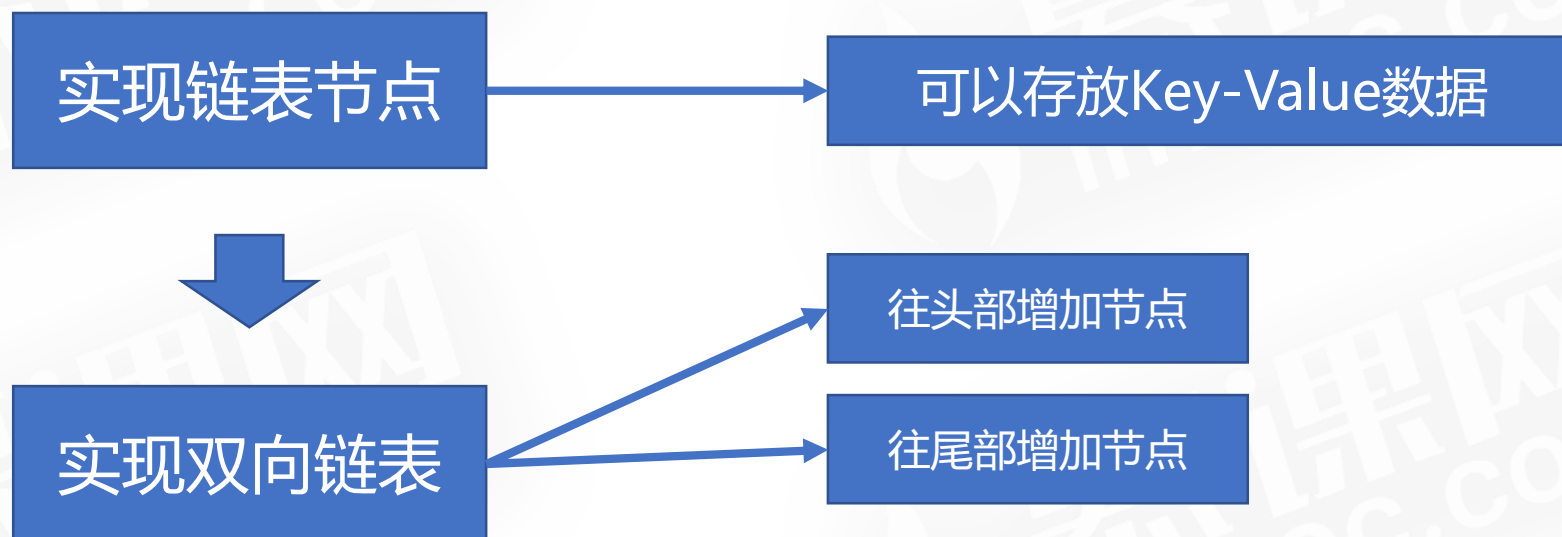


双向链表的原理与实践

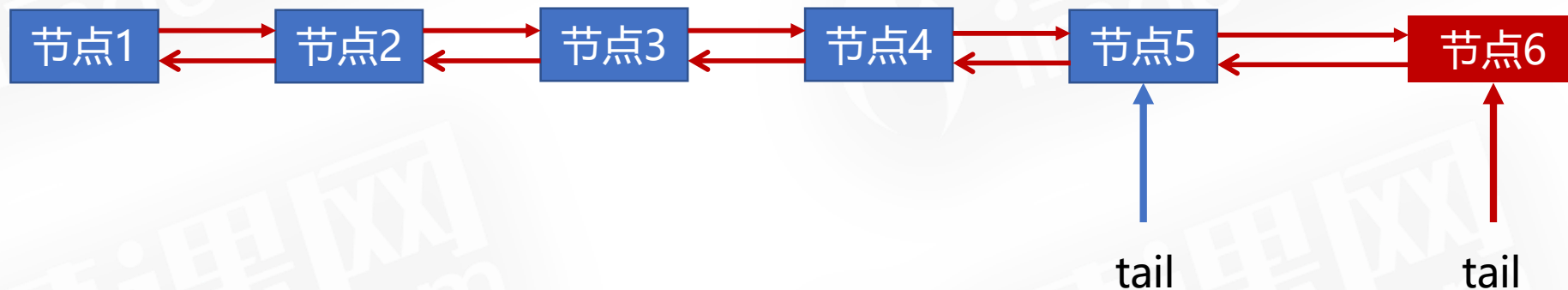


往头部增加节点

双向链表的原理与实践

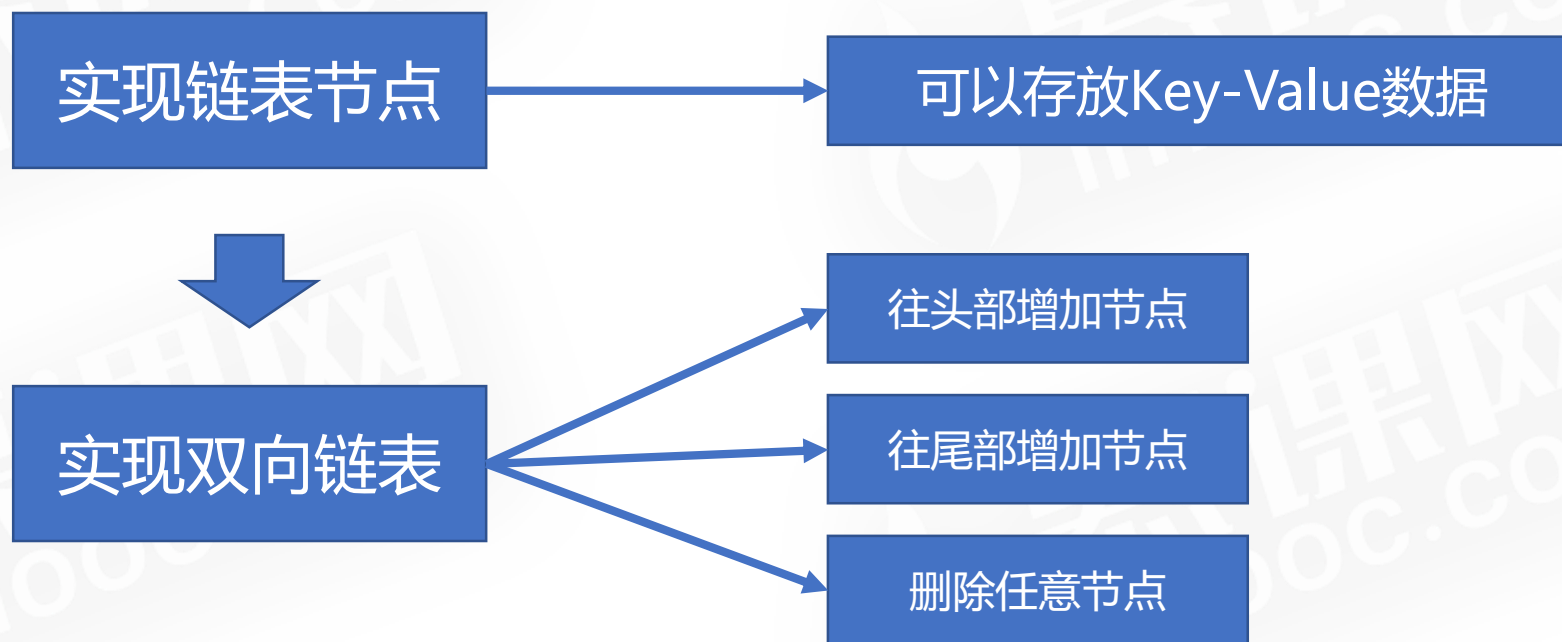


双向链表的原理与实践



往尾部增加节点

双向链表的原理与实践



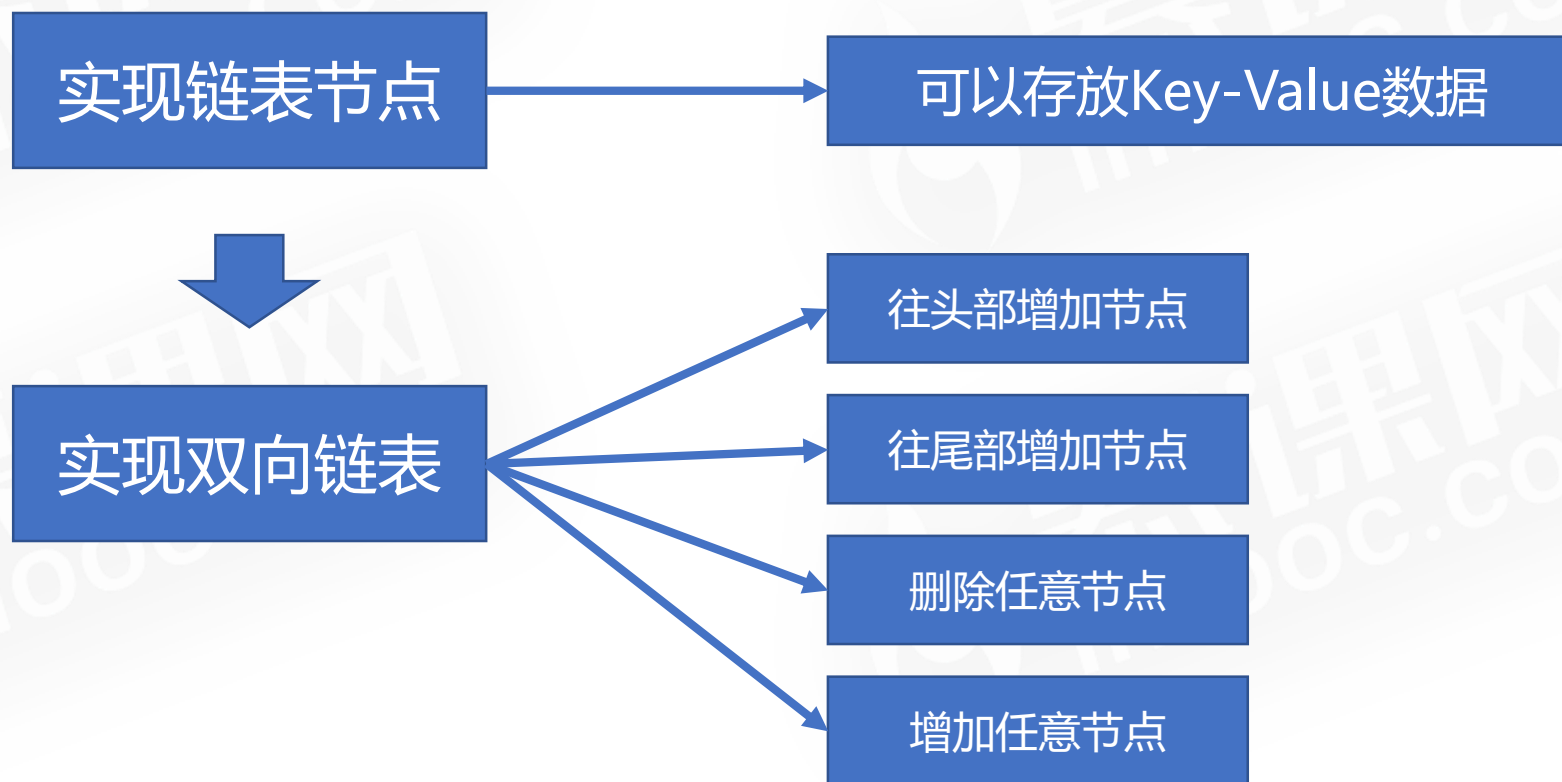
双向链表的原理与实践



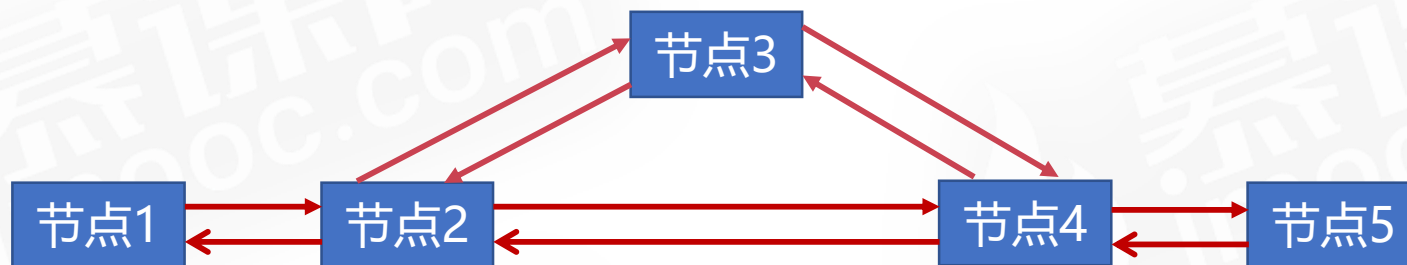
- ◆ 删除节点
- ◆ 改变上一个节点的引用
- ◆ 改变下一个节点的引用

删除任意节点

双向链表的原理与实践



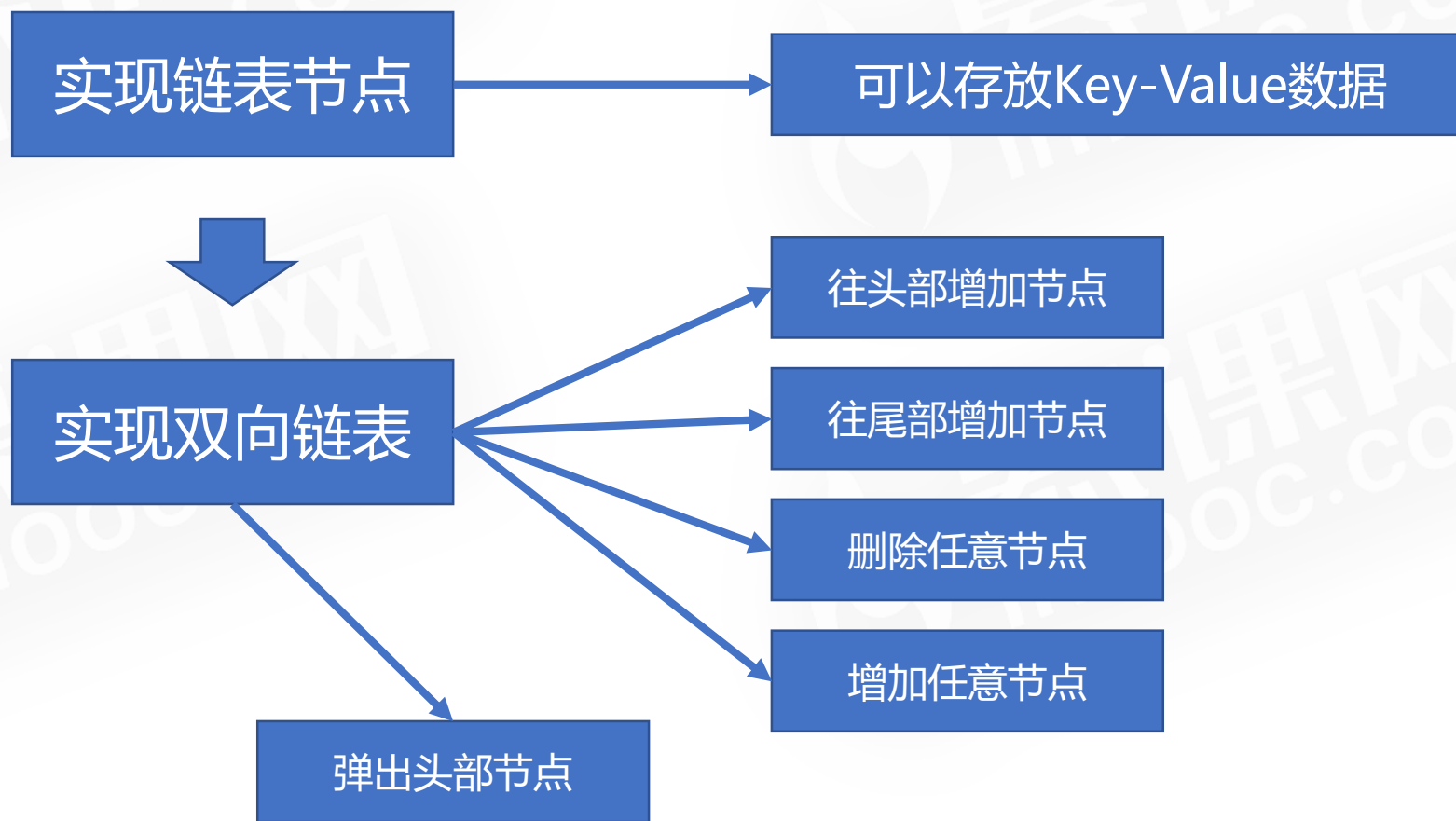
双向链表的原理与实践



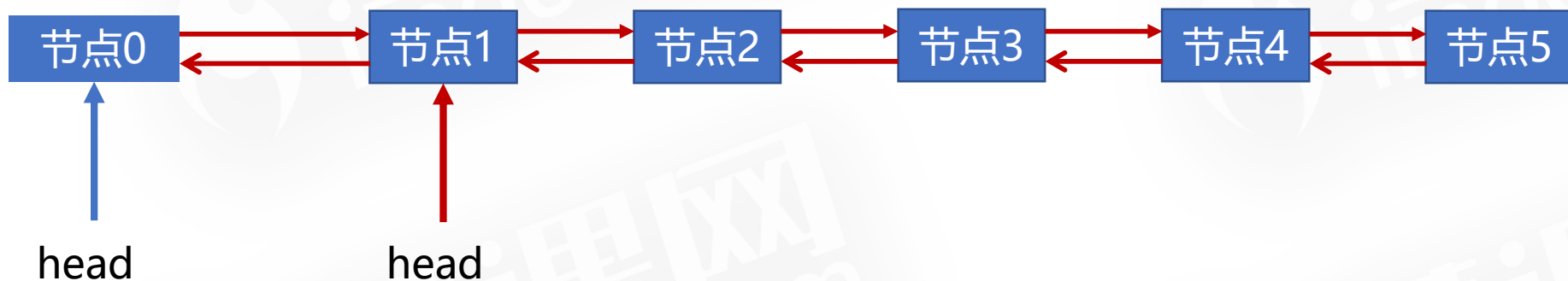
- ◆ 新建节点
- ◆ 改变上一个节点的引用
- ◆ 改变下一个节点的引用

增加任意节点

双向链表的原理与实践

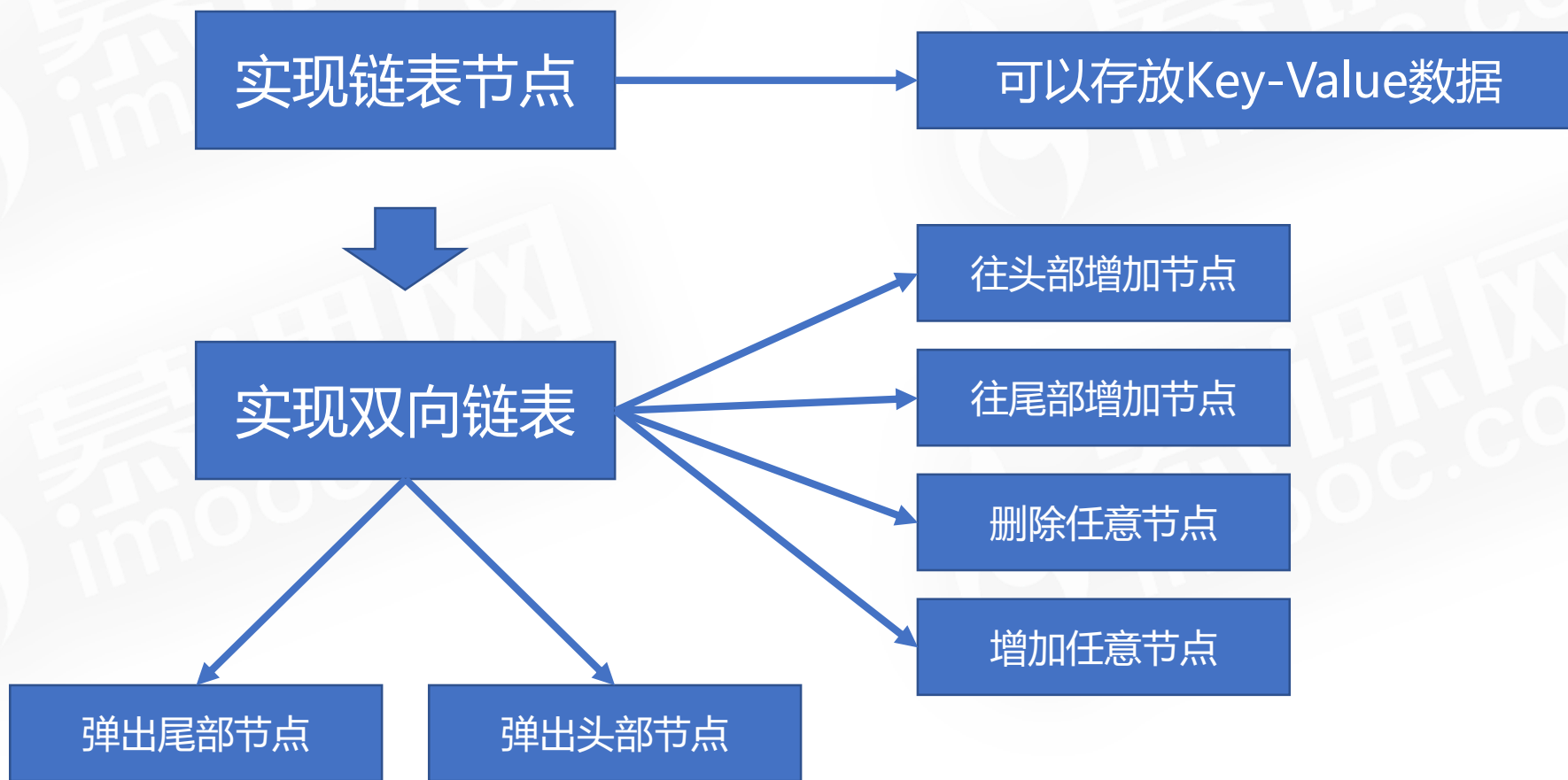


双向链表的原理与实践

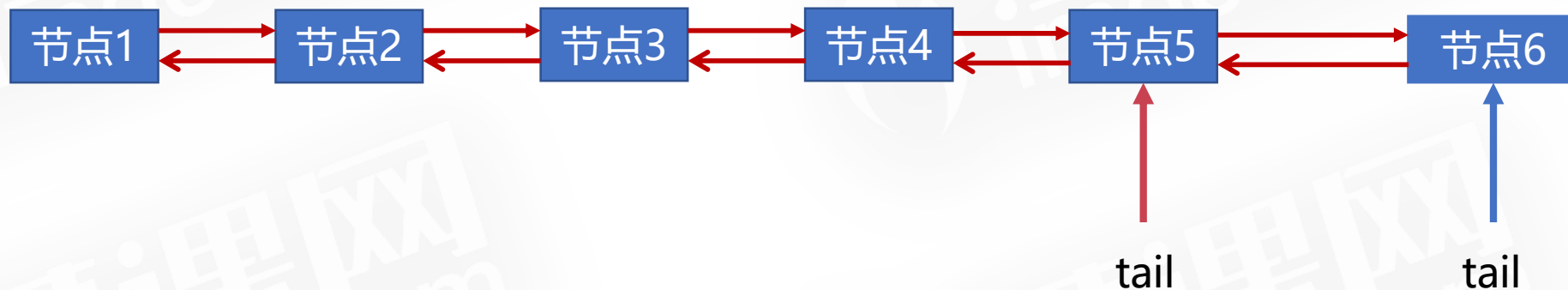


弹出头部节点

双向链表的原理与实践

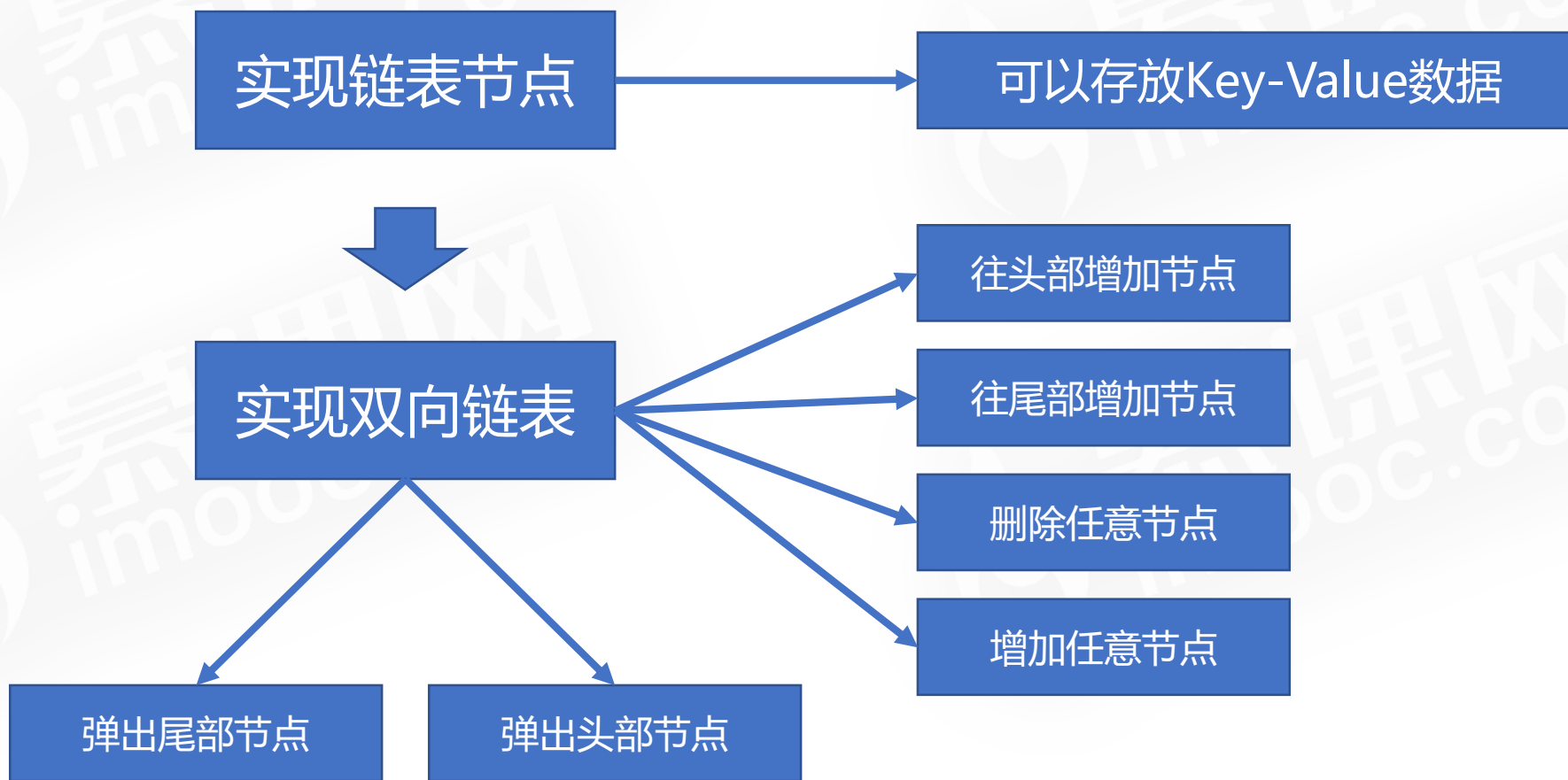


双向链表的原理与实践



弹出尾部节点

双向链表的原理与实践





实践FIFO缓存置换算法

FIFO:

先进先出算法



淘汰缓存时，把最先进入链表的节点淘汰



实践LRU缓存置换算法

最近最少使用算法 (LRU)

(1) 1

(2) 2、 1

(4) 4、 2、 1

(7) 7、 4、 2、 1

(5) 5、 7、 4、 2 [1]

(4) 4、 5、 7、 2

(6) 6、 4、 5、 7 [2]

(1) 1、 6、 4、 5 [7]

(6) 6、 1、 4、 5

(7) 7、 6、 1、 4 [5]

(4) 4、 7、 6、 1

(1) 1、 4、 7、 6

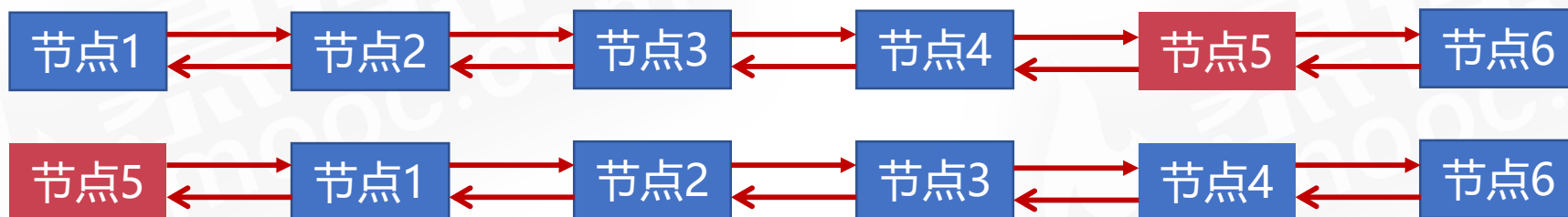
假设缓存4个字块， () 表示使用的字块， []表示淘汰的字块

实践LRU缓存置换算法

最近最少使用算法 (LRU)



◆ 每次使用，把使用的节点放到链表最前面



淘汰缓存时，把链表尾部的节点淘汰

慕课网
imooc.com

慕课网
imooc.com

慕课网
imooc.com

慕课网
imooc.com

实践LFU缓存置换算法

LFU:

最不经常使用算法

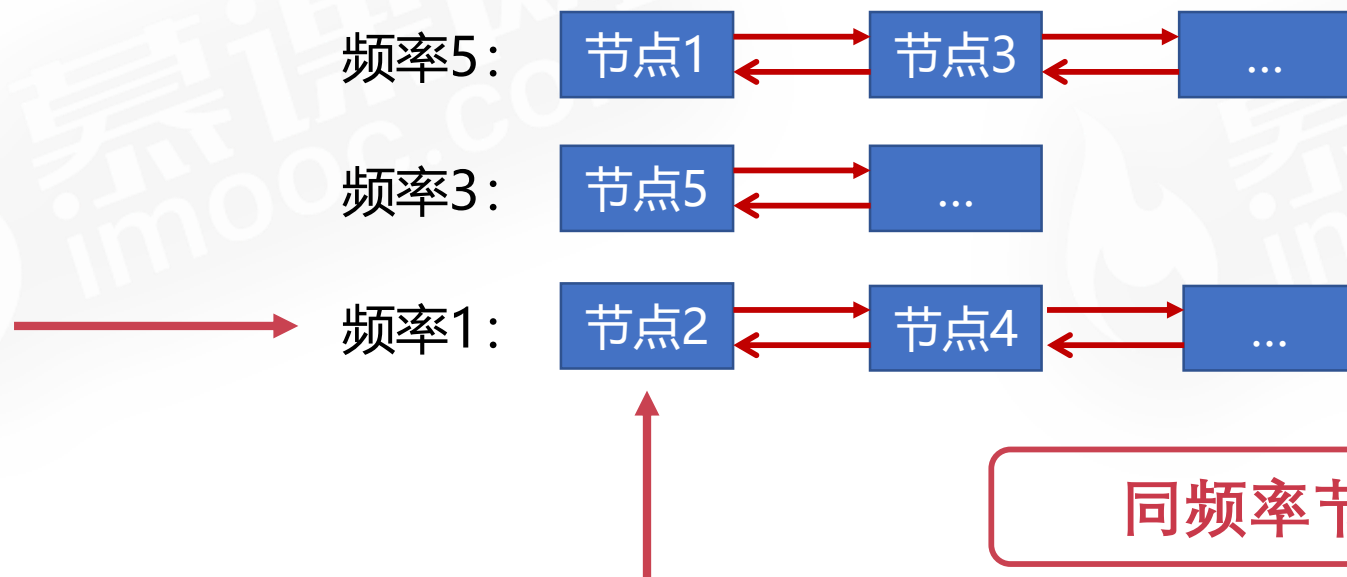
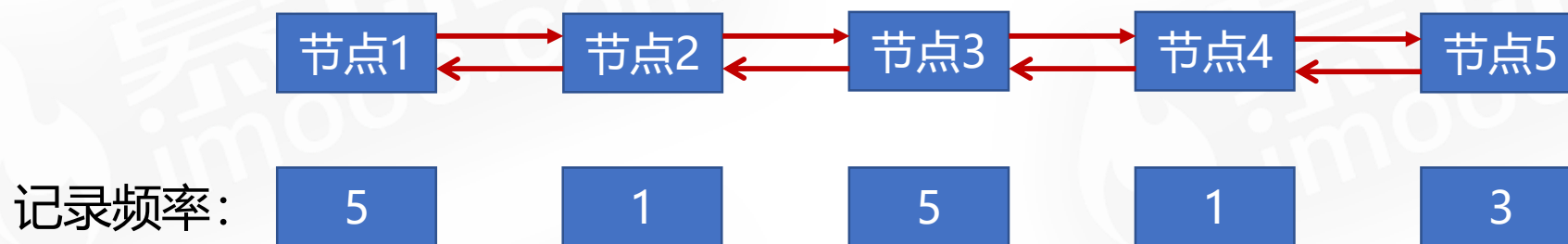
可能存在相同频率的情况，这时应该淘汰哪个节点呢？



淘汰缓存时，把使用频率最小的淘汰



实践LFU缓存置换算法



同频率节点按FIFO算法淘汰

