

链表

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

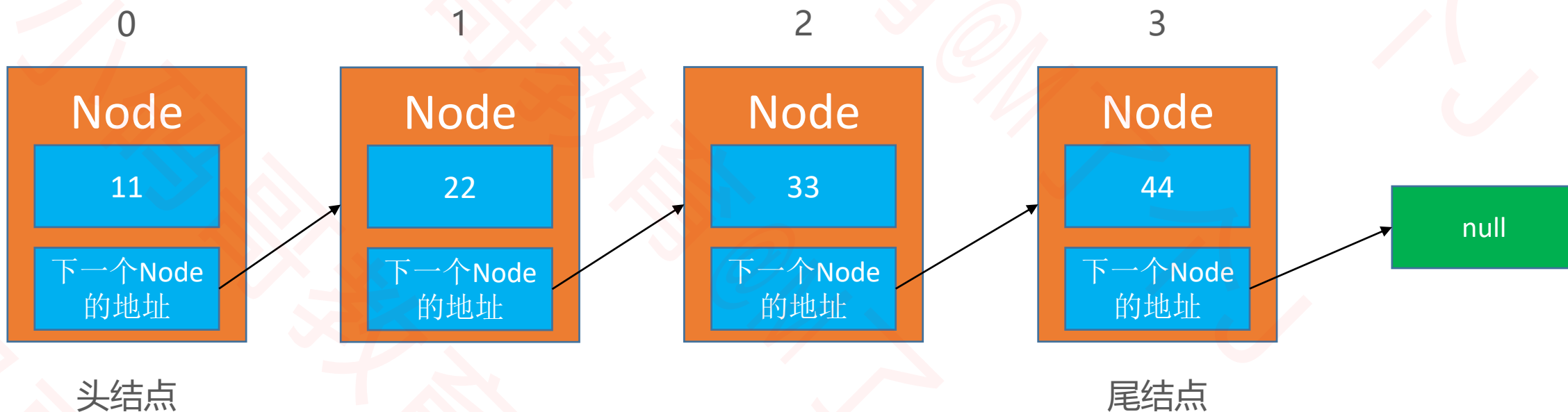
码拉松



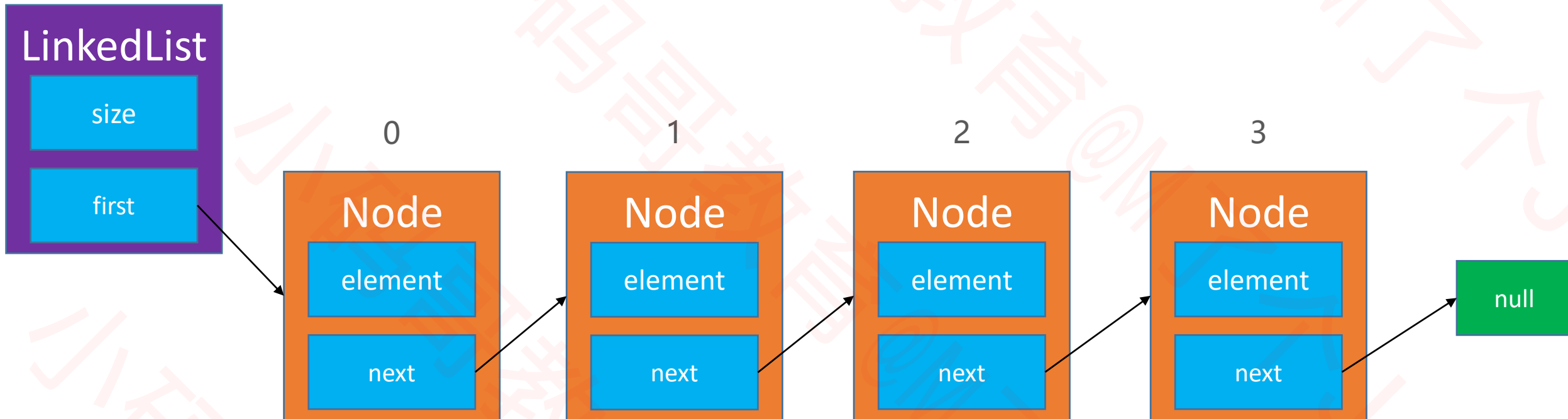
实力IT教育 www.520it.com

链表 (Linked List)

- 动态数组有个明显的缺点
 - 可能会造成内存空间的大量浪费
 - 能否用到多少就申请多少内存?
 - 链表可以办到这一点
- **链表**是一种**链式存储**的线性表，所有元素的内存地址不一定是连续的

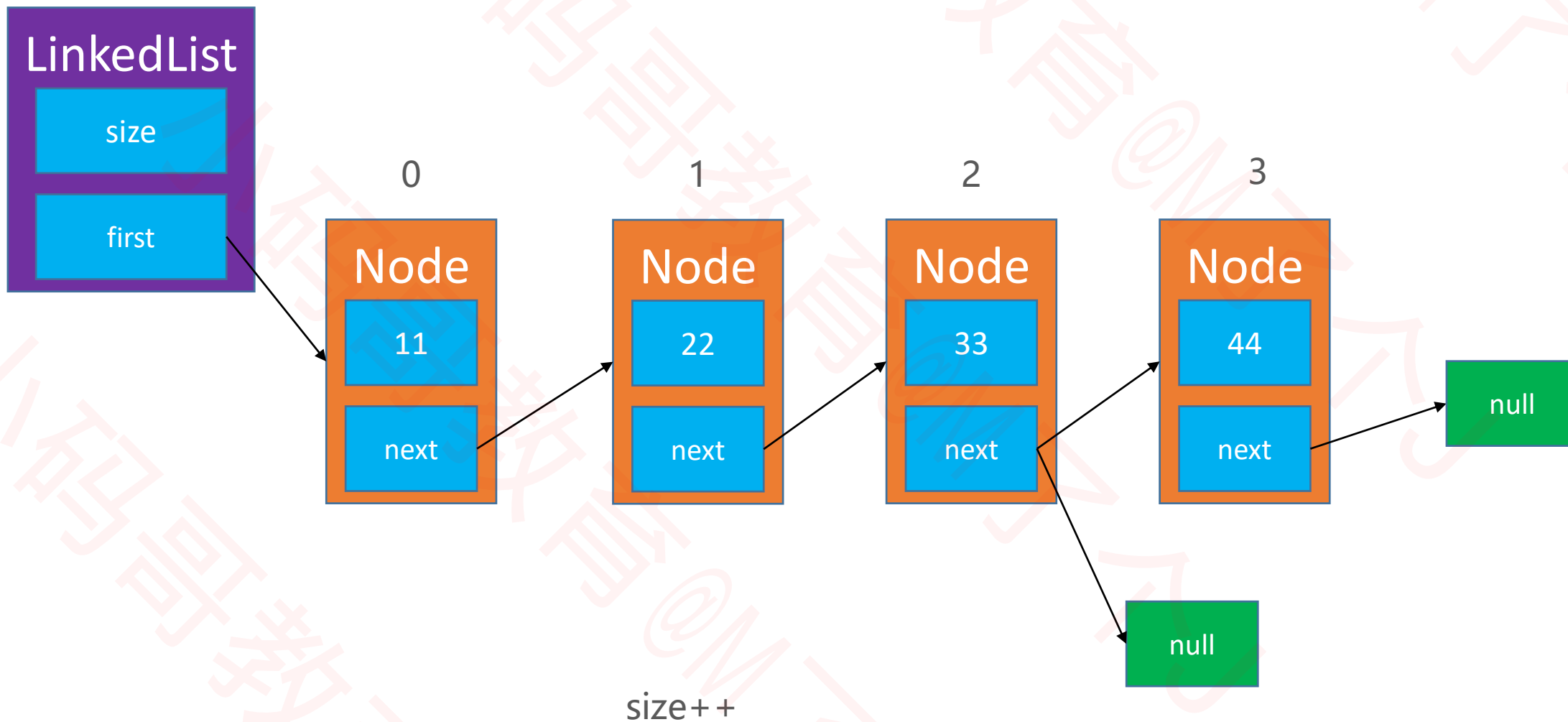


链表的设计

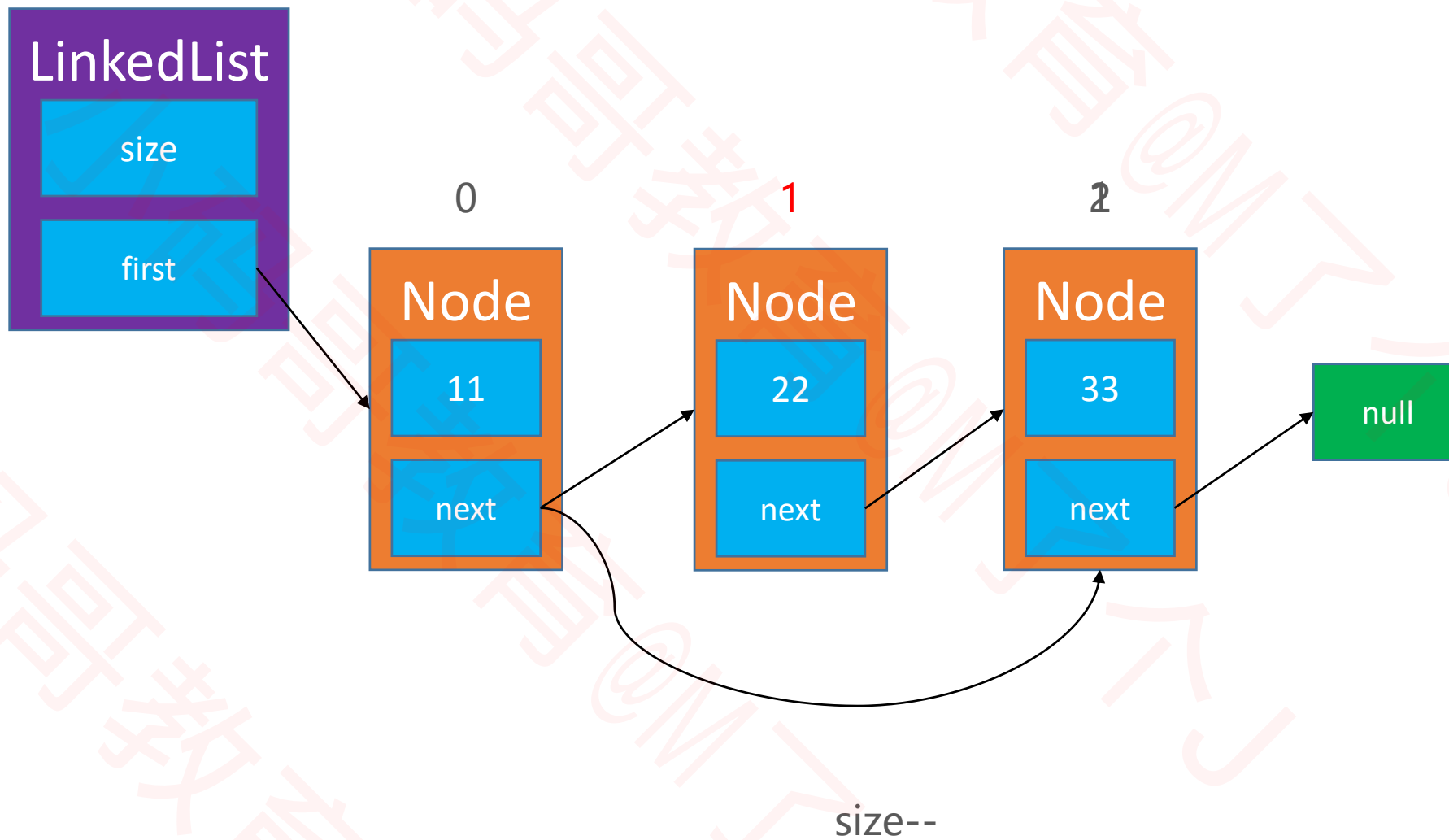


- 链表的大部分接口和动态数组是一致的

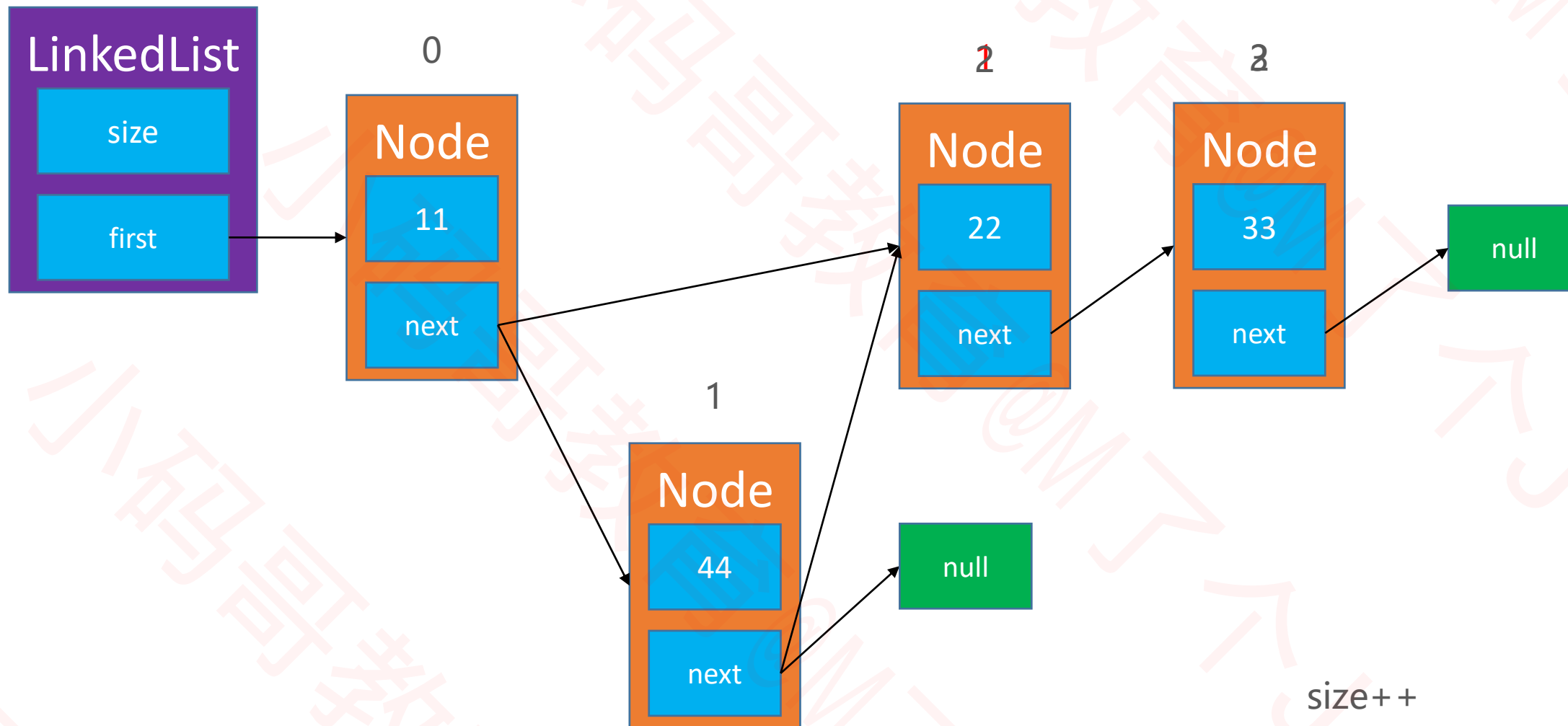
添加元素 - add(E element)



删除元素 - remove(int index)



添加元素 - add(int index, E element)



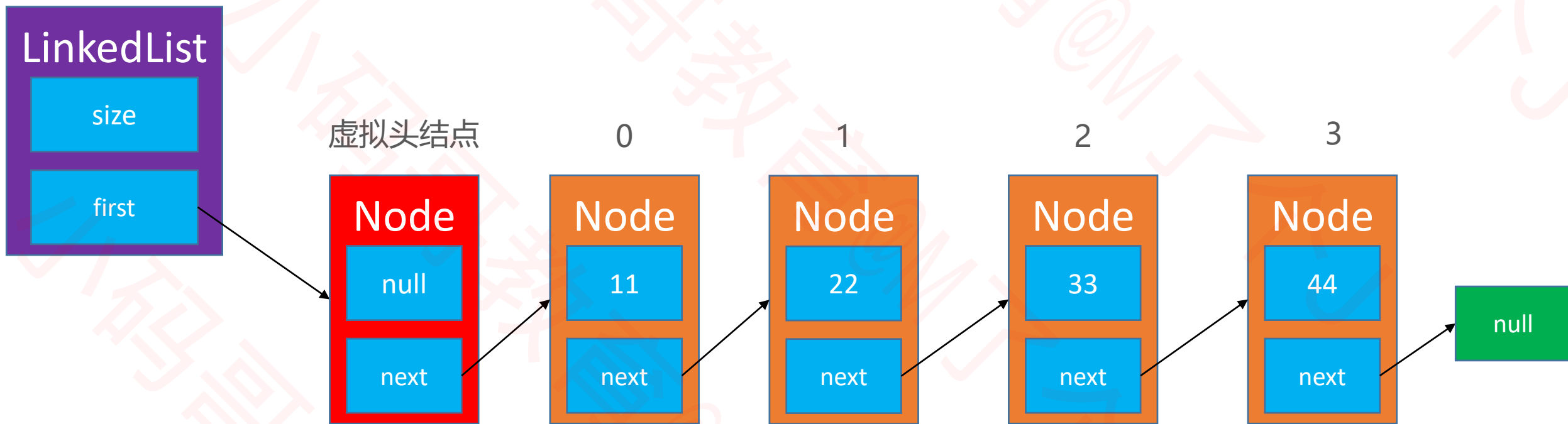
推荐一个神奇的网站

■ <https://visualgo.net/zh>



虚拟头结点

- 有时候为了让代码更加精简，统一所有节点的处理逻辑，可以在最前面增加一个虚拟的头结点（不存储数据）



练习 - 删除链表中的节点

■ <https://leetcode-cn.com/problems/delete-node-in-a-linked-list/>

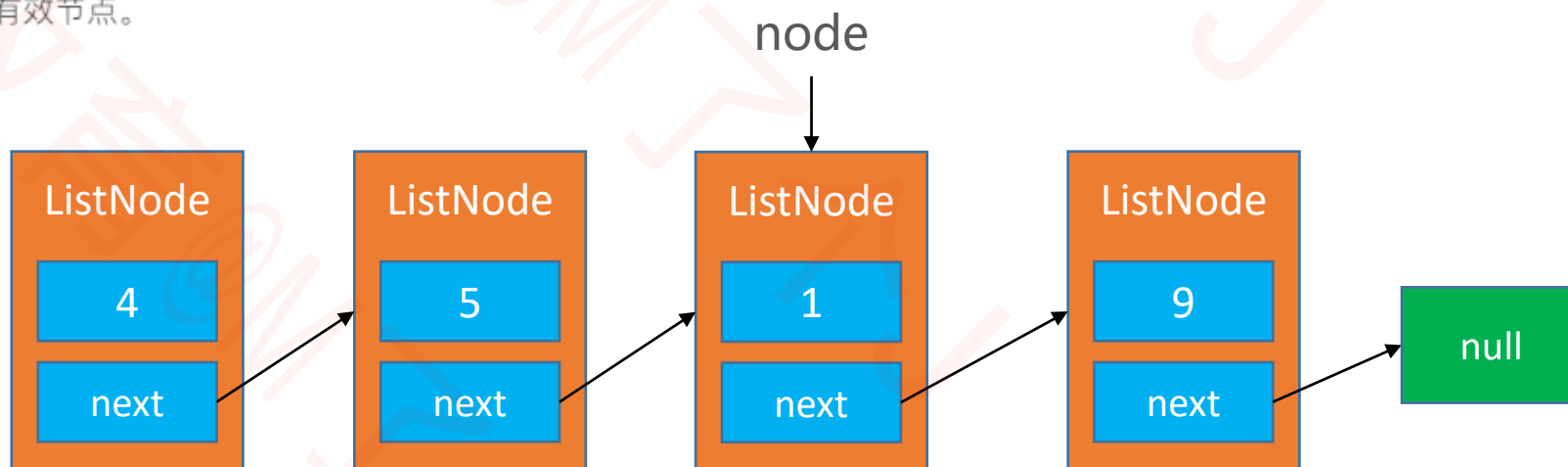
输入: head = [4,5,1,9], node = 1

输出: [4,5,9]

解释: 给定你链表中值为 1 的第三个节点, 那么在调用了你的函数之后, 该链表应变为 4 -> 5 -> 9.

说明:

- 链表至少包含两个节点。
- 链表中所有节点的值都是唯一的。
- 给定的节点为非末尾节点并且一定是链表中的一个有效节点。
- 不要从你的函数中返回任何结果。



练习 - 反转一个链表

- <https://leetcode-cn.com/problems/reverse-linked-list/>

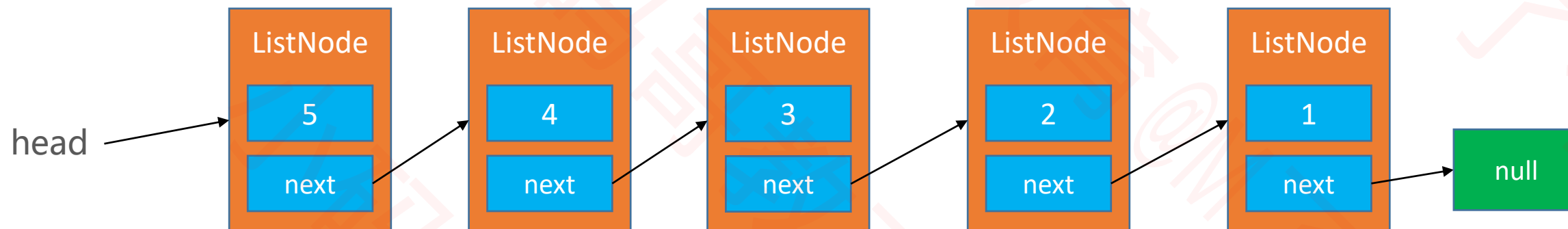
示例:

输入: 1->2->3->4->5->NULL

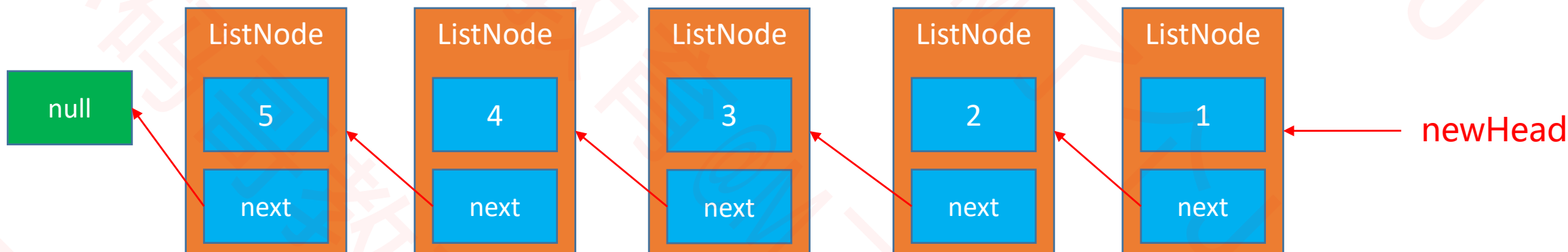
输出: 5->4->3->2->1->NULL

- 请分别用递归、迭代（非递归）两种方式实现

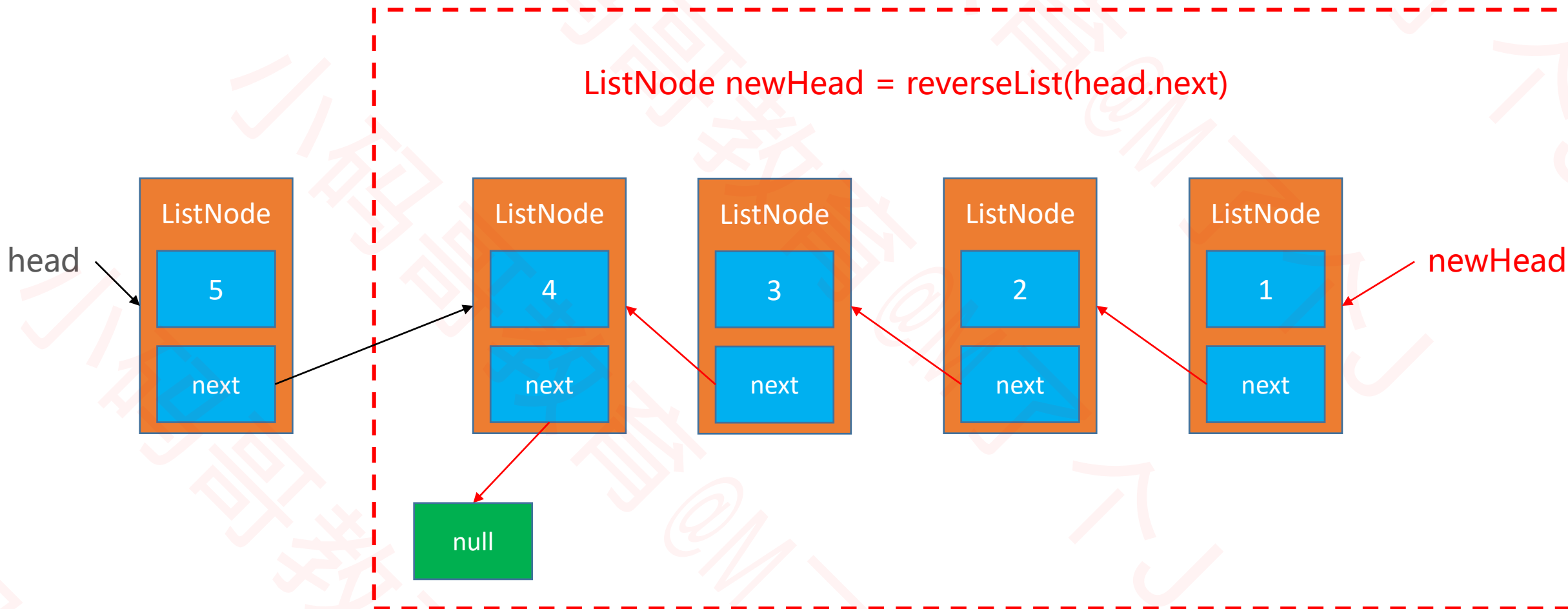
练习 - 反转一个链表 - 递归



`ListNode newHead = reverseList(head)`

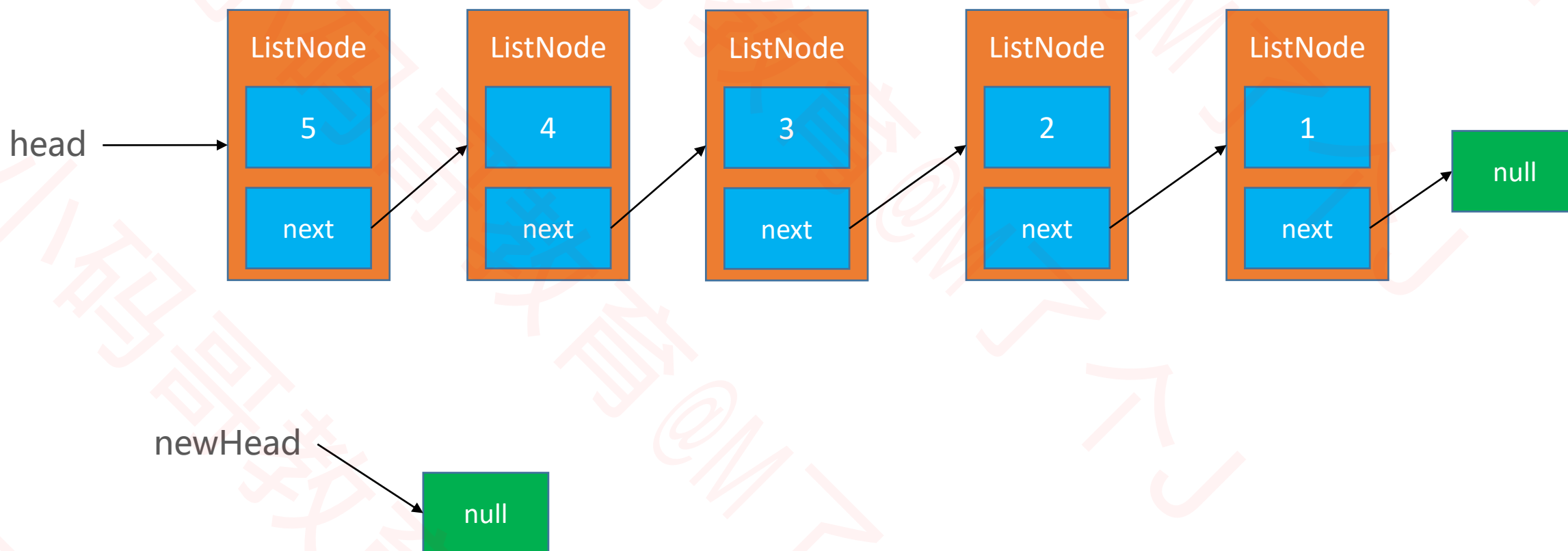


练习 - 反转一个链表 - 递归



练习 - 反转一个链表 - 非递归

tmp = head.next



练习 - 判断一个链表是否有环

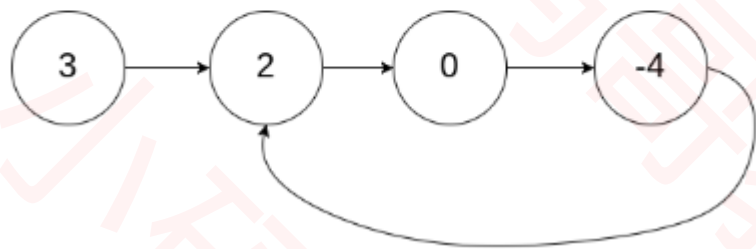
■ <https://leetcode-cn.com/problems/linked-list-cycle/>

示例 1:

输入: head = [3,2,0,-4], pos = 1

输出: true

解释: 链表中有一个环, 其尾部连接到第二个节点。

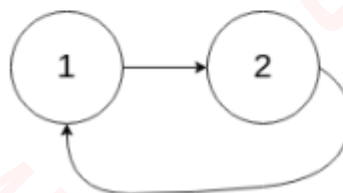


示例 2:

输入: head = [1,2], pos = 0

输出: true

解释: 链表中有一个环, 其尾部连接到第一个节点。



示例 3:

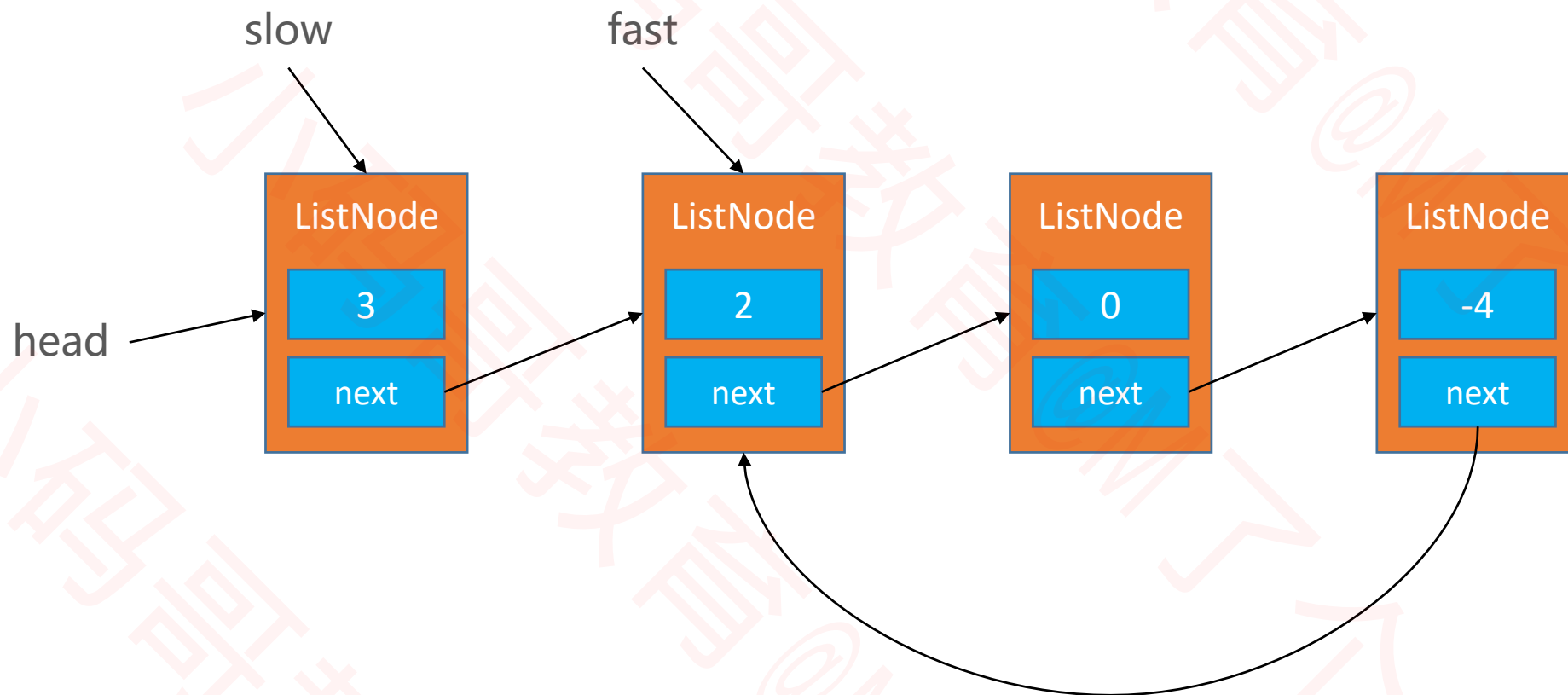
输入: head = [1], pos = -1

输出: false

解释: 链表中没有环。



练习 - 判断一个链表是否有环



■ 移除链表元素

□ <https://leetcode-cn.com/problems/remove-linked-list-elements/>

■ 删除排序链表中的重复元素

□ <https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list/>

■ 链表的中间结点

□ <https://leetcode-cn.com/problems/middle-of-the-linked-list/solution/>

复杂度分析

- 最好情况复杂度
- 最坏情况复杂度
- 平均情况复杂度

动态数组、链表复杂度分析

	动态数组			链表		
	最好	最坏	平均	最好	最坏	平均
add(int index, E element)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
remove(int index)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
set(int index, E element)	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$
get(int index)	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$

size是数组规模n

动态数组add(E element)复杂度分析

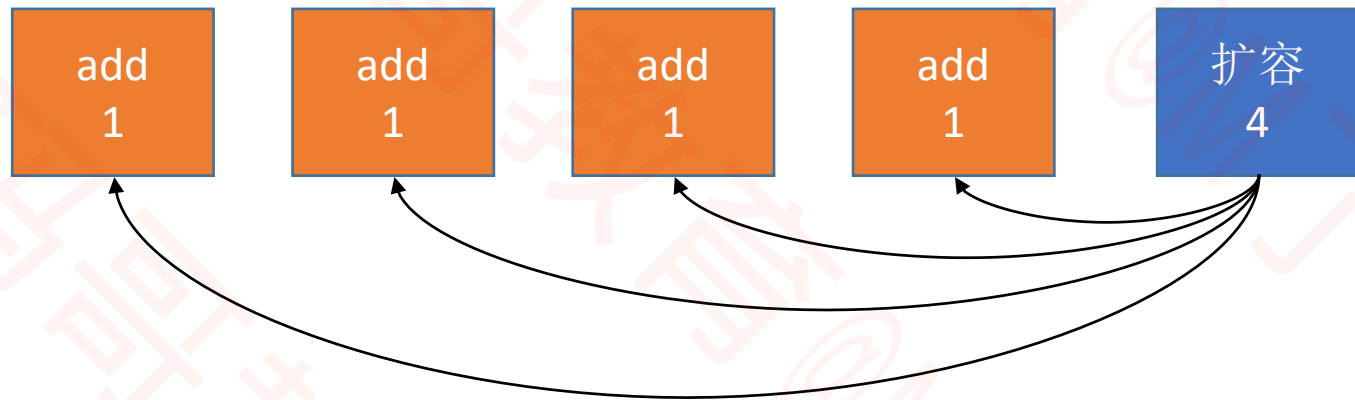
■ 最好: $O(1)$

■ 最坏: $O(n)$

■ 平均: $O(1)$

■ 均摊: $O(1)$

假设最大容量是4



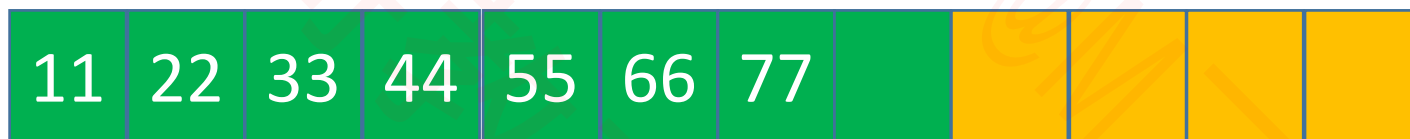
相当于每次add的操作次数是2，也就是 $O(1)$ 复杂度

■ 什么情况下适合使用均摊复杂度

□ 经过连续的多次复杂度比较低的情况后，出现个别复杂度比较高的情况

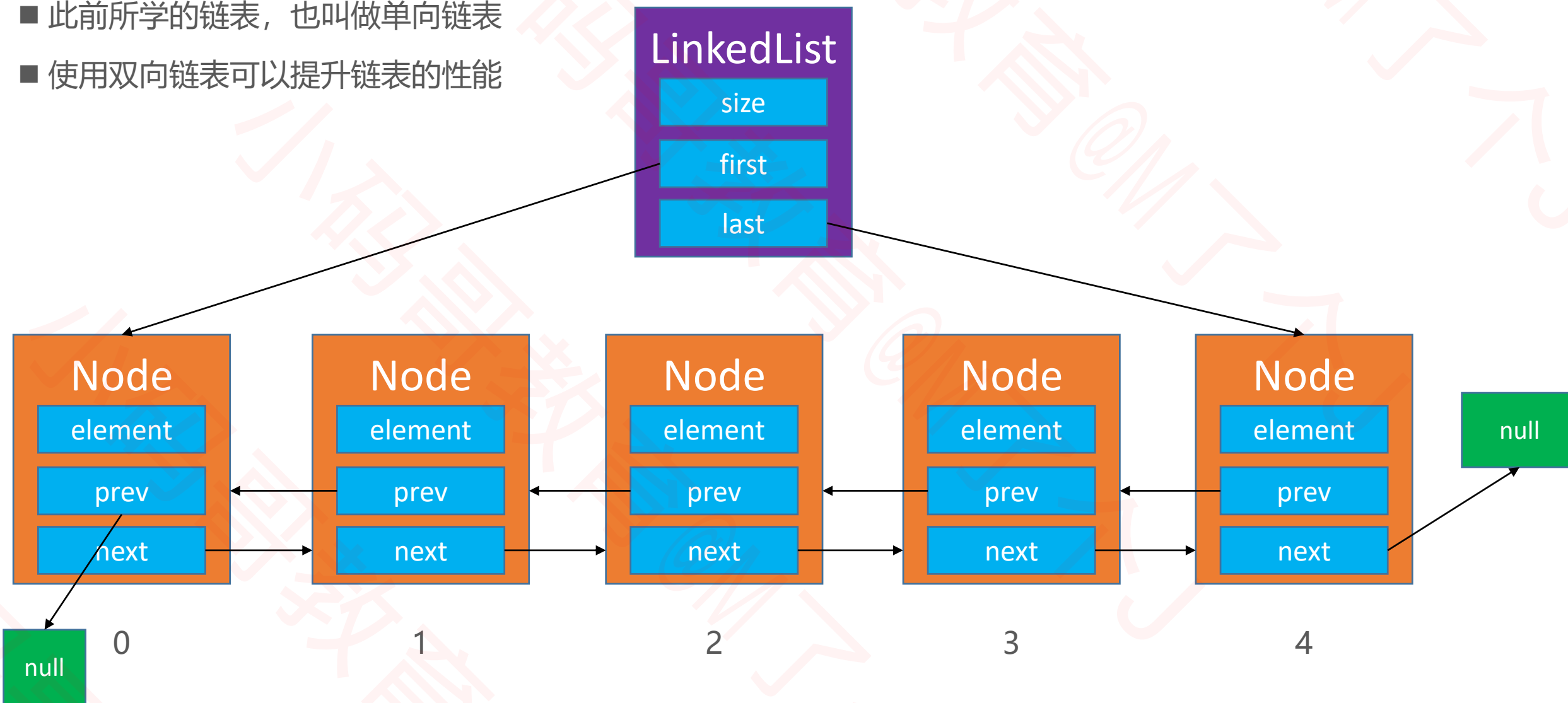
动态数组的缩容

- 如果内存使用比较紧张，动态数组有比较多的剩余空间，可以考虑进行缩容操作
 - 比如剩余空间占总容量的一半时，就进行缩容
- 如果扩容倍数、缩容时机设计不得当，有可能会造成复杂度震荡

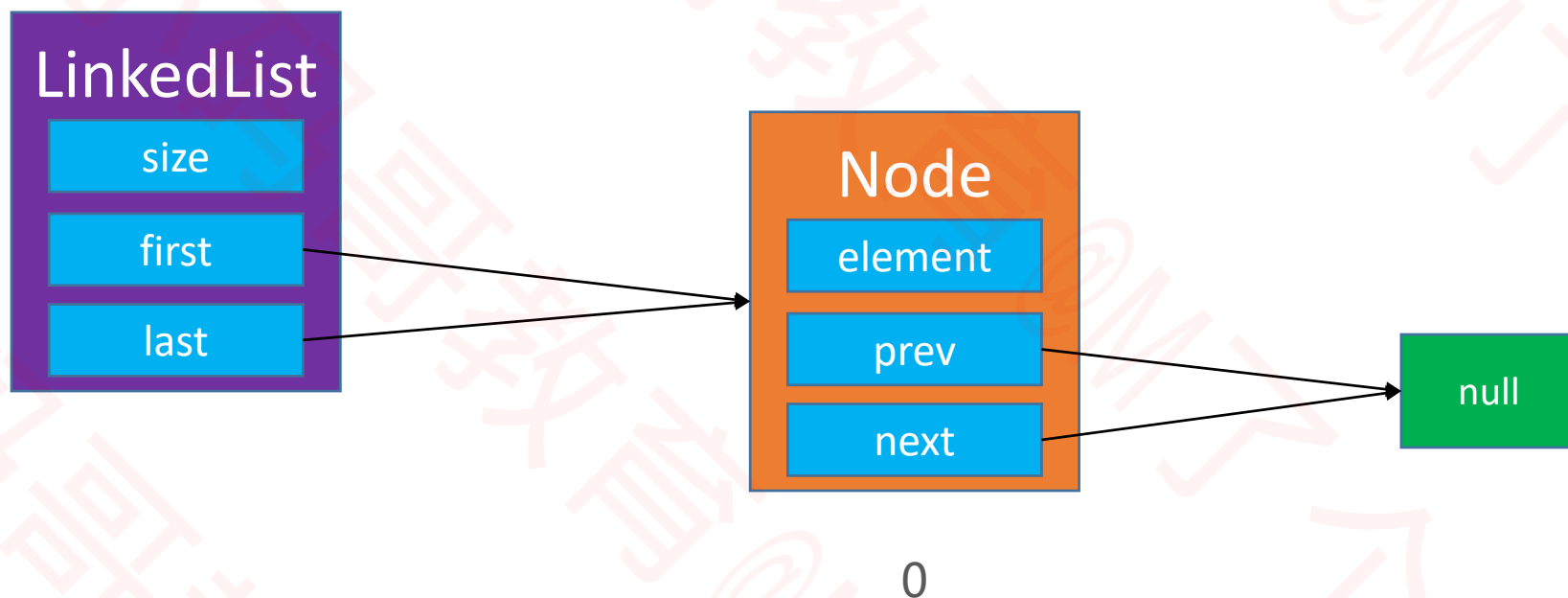


双向链表

- 此前所学的链表，也叫做单向链表
- 使用双向链表可以提升链表的性能



双向链表 - 只有一个元素



双向链表 vs 单向链表

■ 粗略对比一下删除的操作数量

□ 单向链表: $1 + 2 + 3 + \dots + n = \frac{(1+n) * n}{2} = \frac{n}{2} + \frac{n^2}{2}$, 除以n平均一下是 $\frac{1}{2} + \frac{n}{2}$

□ 双向链表: $(1 + 2 + 3 + \dots + \frac{n}{2}) * 2 = \frac{(1 + \frac{n}{2}) * \frac{n}{2}}{2} * 2 = \frac{n}{2} + \frac{n^2}{4}$, 除以n平均一下是 $\frac{1}{2} + \frac{n}{4}$

□ 操作数量缩减了近一半

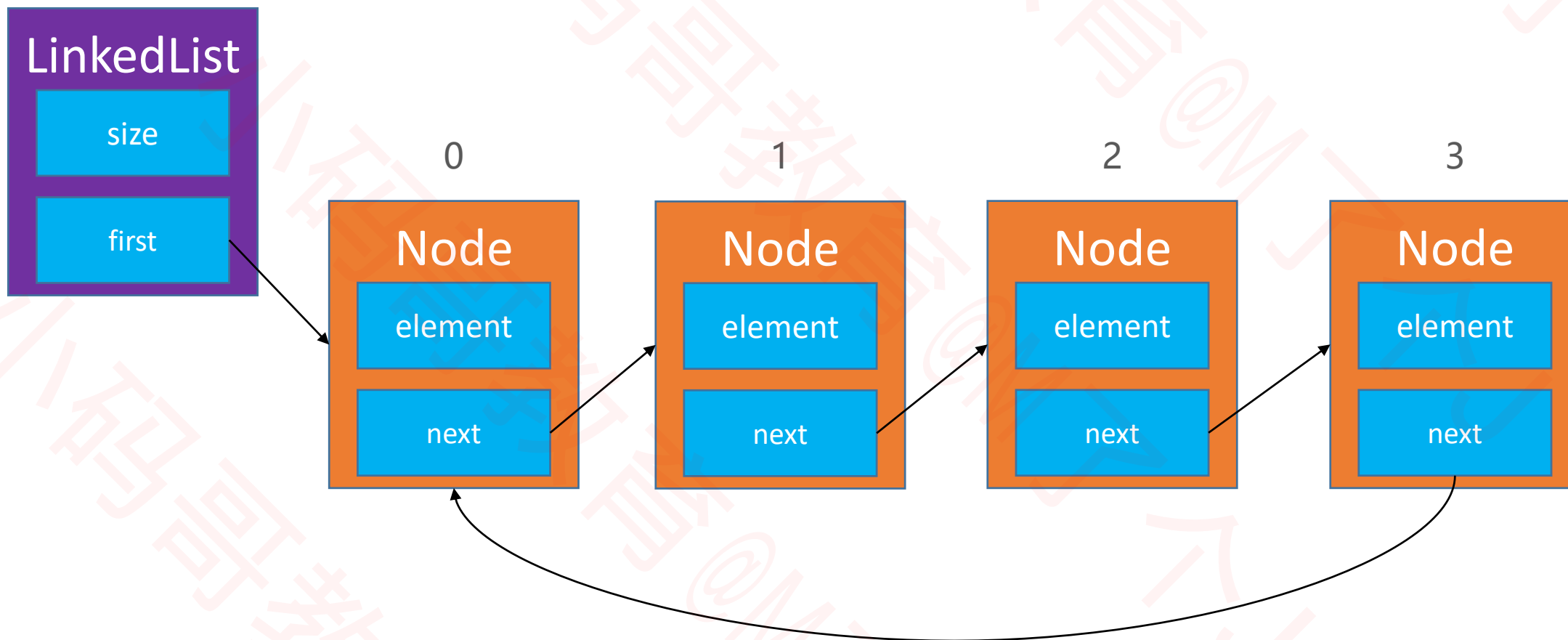
双向链表 vs 动态数组

- 如果有频繁的查询操作（随机访问操作），建议选择使用动态数组
- 如果频繁在尾部进行添加、删除操作，动态数组、双向链表均可选择
- 如果频繁在头部进行添加、删除操作，建议选择使用双向链表
- 如果有频繁的（在任意位置）添加、删除操作，建议选择使用双向链表

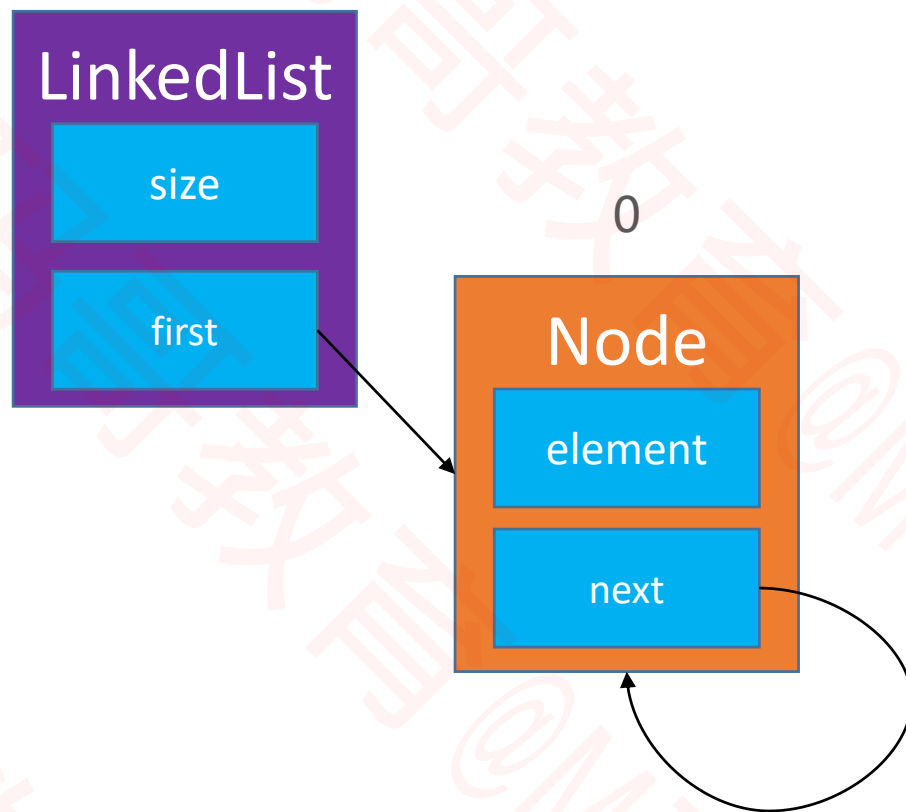
作业 – ArrayList的进一步优化

0	1	2	3	4	5	6	7	8
11	22	33	44	55	66	null	null	null

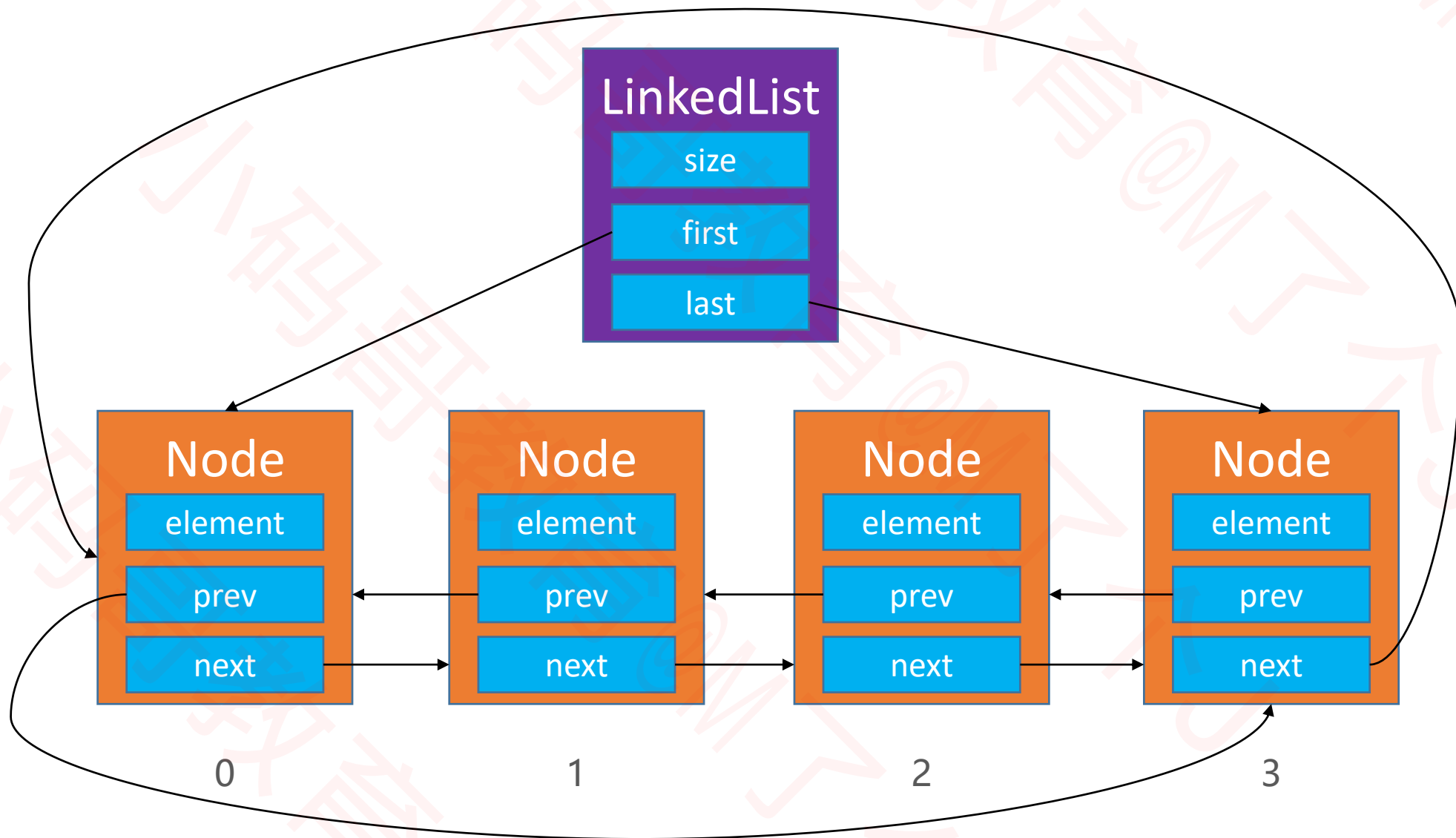
循环链表



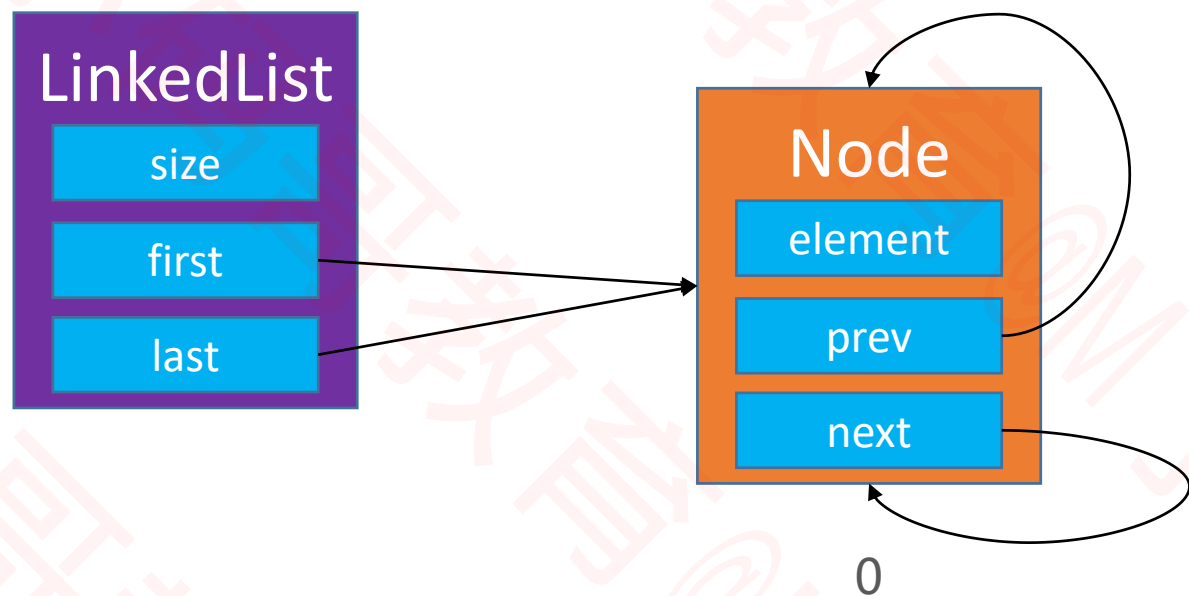
循环链表 – 只有1个节点



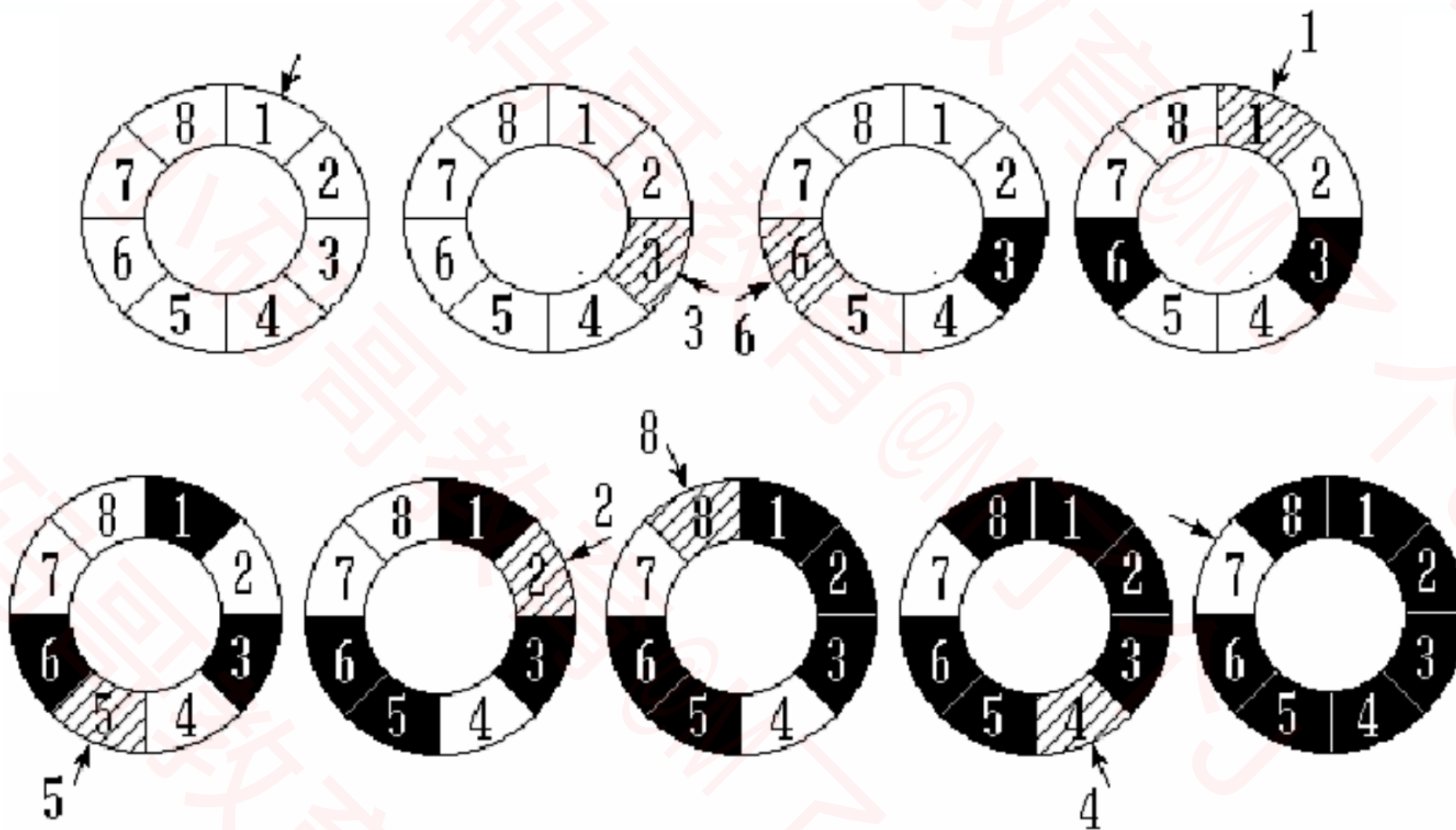
双向循环链表



双向循环链表 - 只有1个节点



练习 – 约瑟夫问题 (Josephus Problem)



静态链表

- 前面所学习的链表，是依赖于指针（引用）实现的
- 有些编程语言是没有指针的，比如早期的BASIC、FORTRAN语言
- 没有指针的情况下，如何实现链表？
- 可以通过数组来模拟链表，称为静态链表
- 数组的每个元素存放2个数据：值、下个元素的索引

