链表

@M了个J

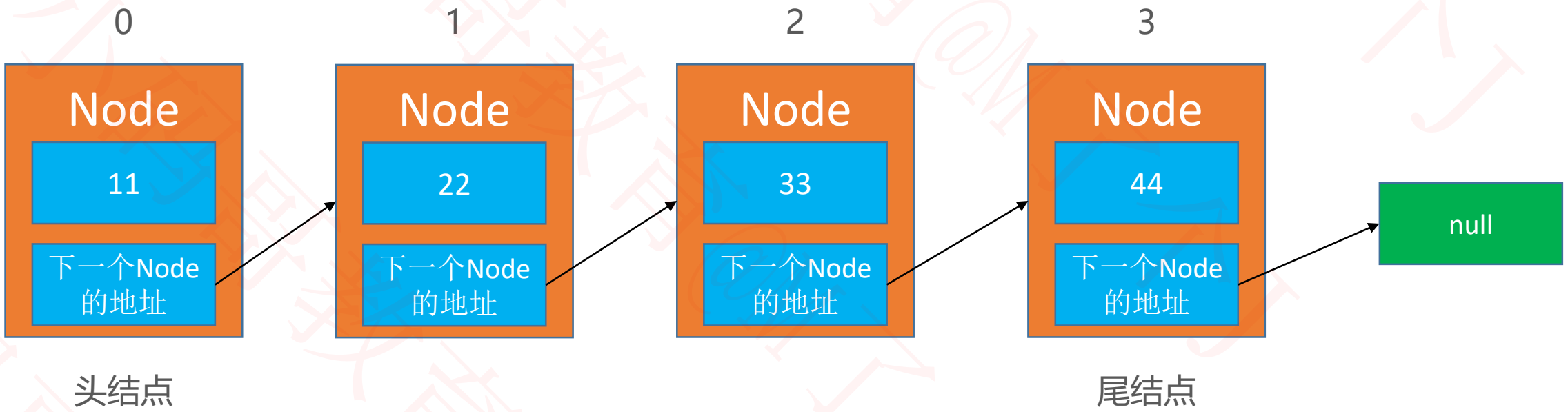
https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

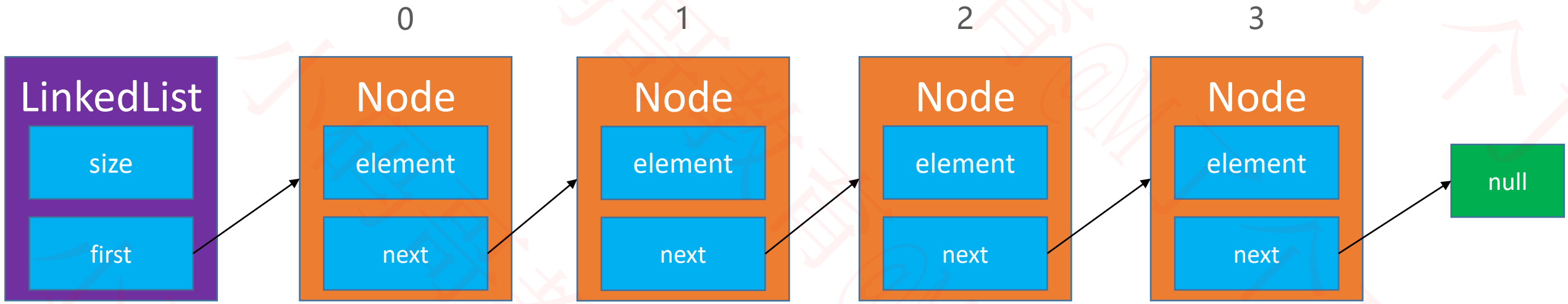
链表（Linked List）

- 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



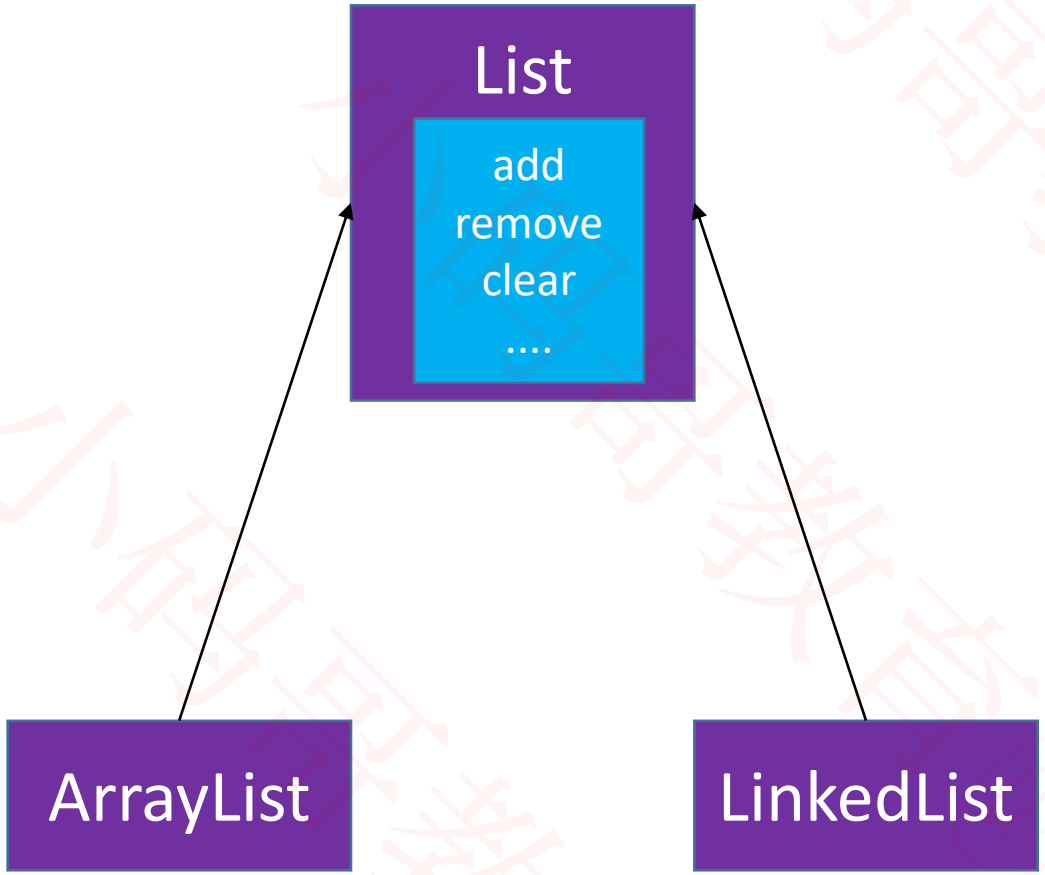
小码哥教育 SEEMYGO

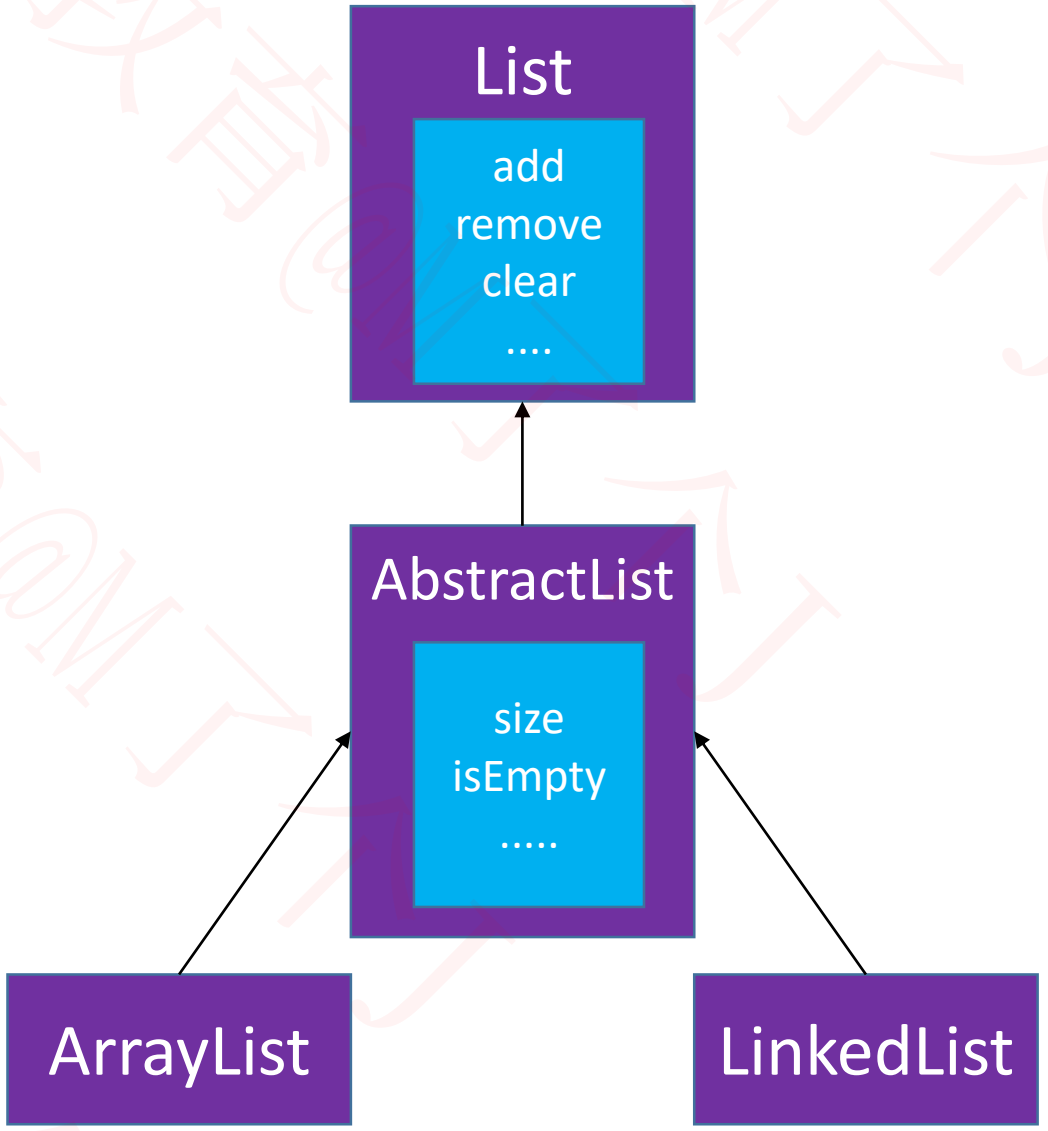
链表的设计



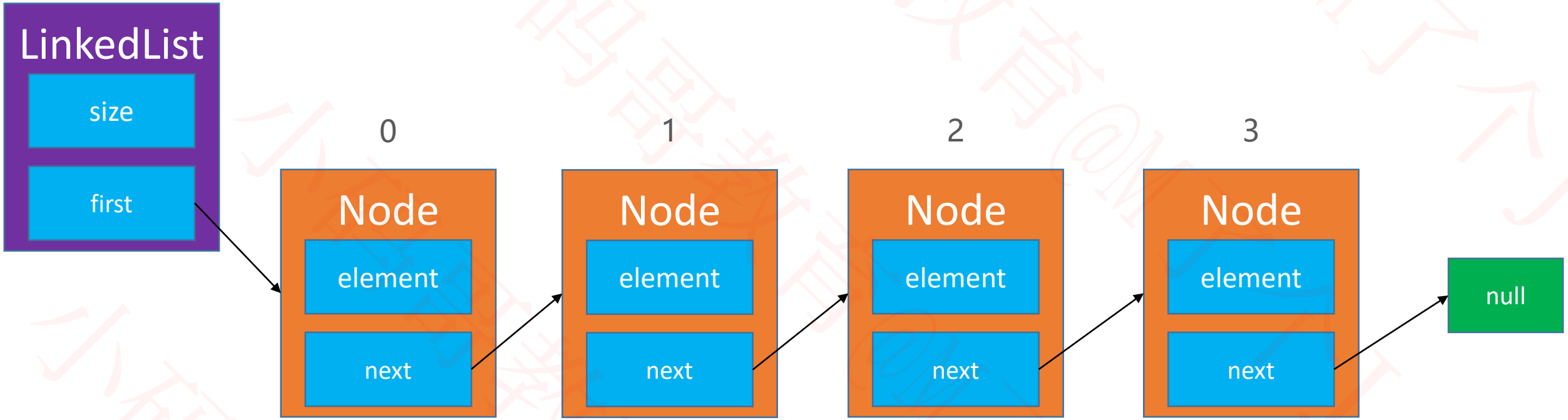
接口设计

- 链表的大部分接口和动态数组是一致的





清空元素 – clear()

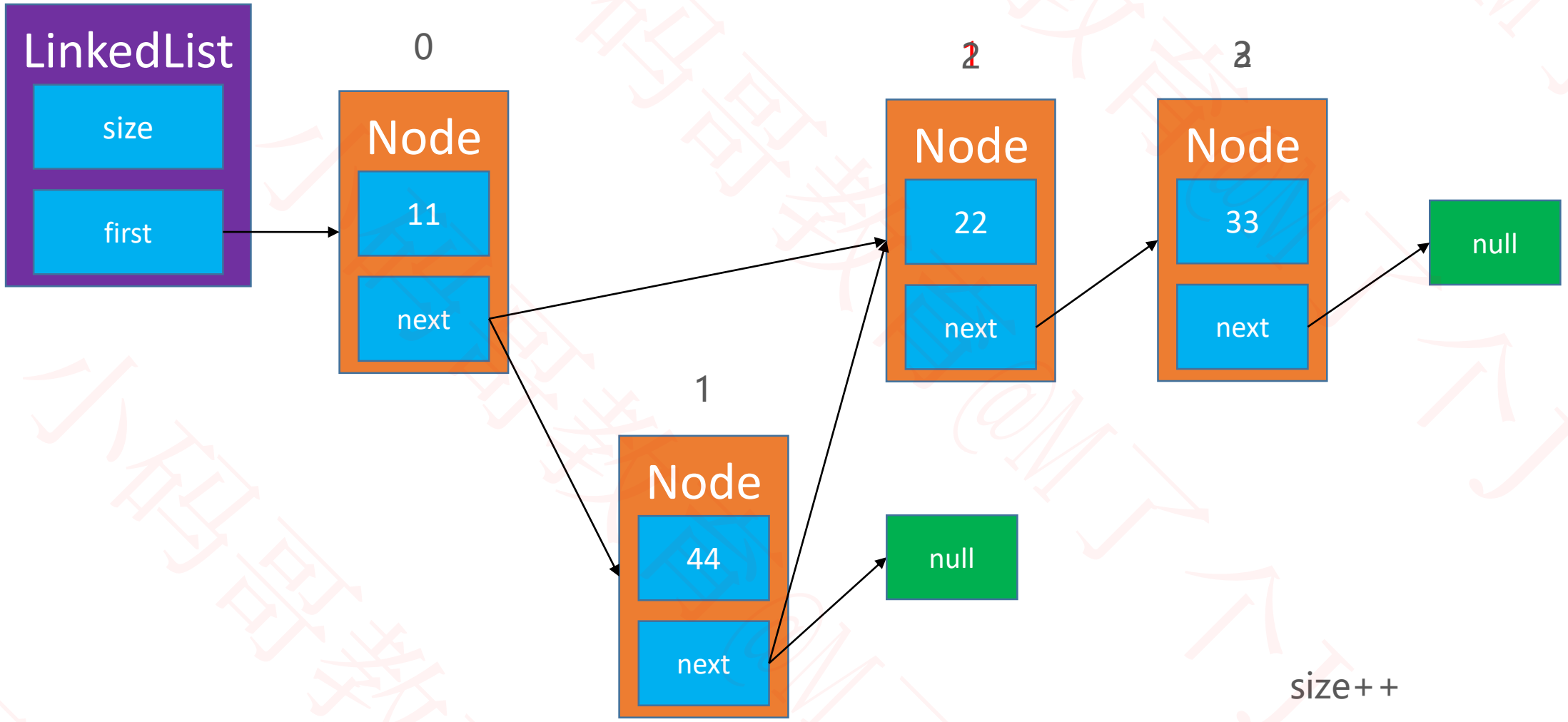


size = 0

first = null

思考：next需要设置为null么？

添加元素 - add(int index, E element)



小码哥教育 SEEMYGO

node方法用于获取index位置的节点

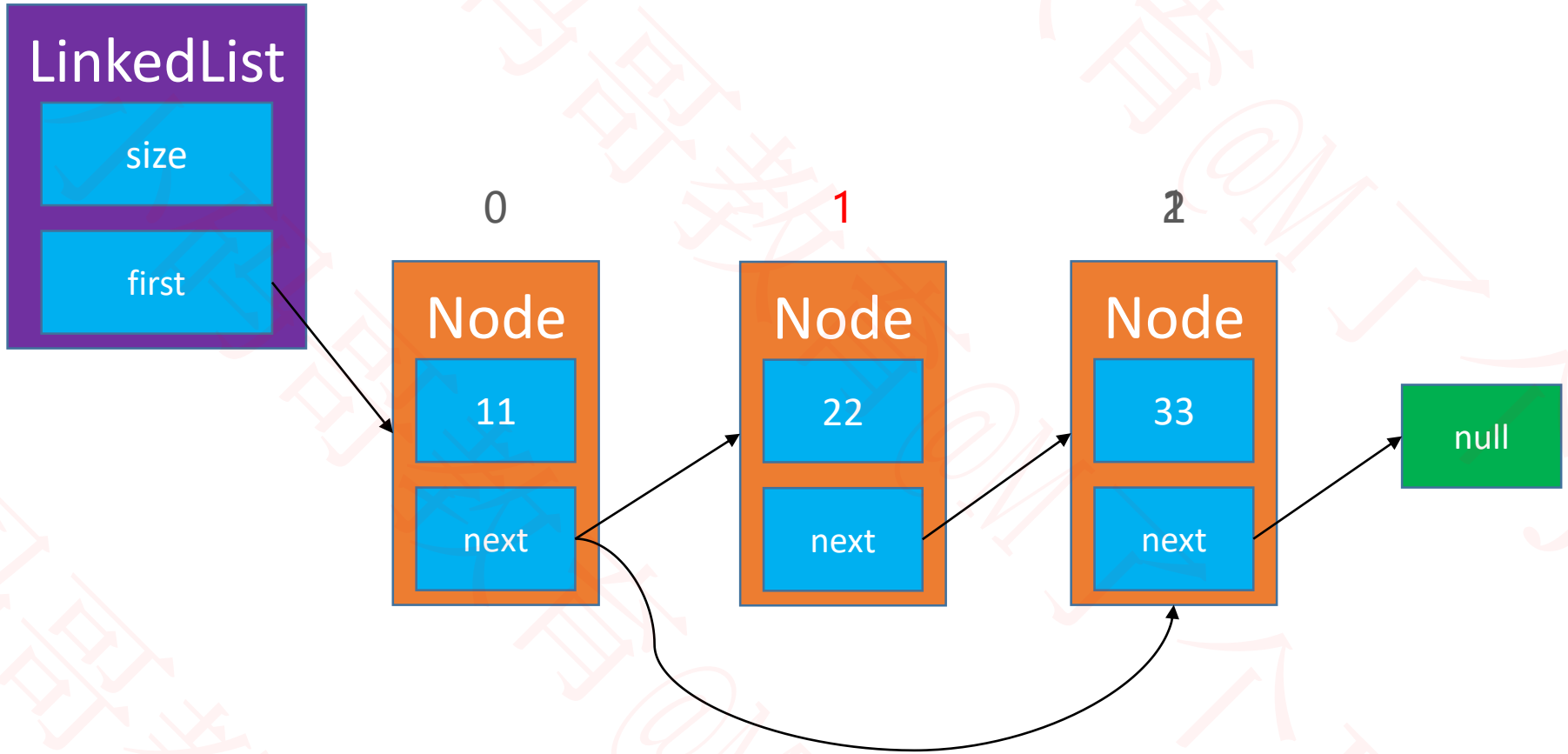
```
private Node<E> node(int index) {
    rangeCheck(index);

    Node<E> node = first;
    for (int i = 0; i < index; i++) {
        node = node.next;
    }
    return node;
}
```

添加元素 – 注意0位置

```
public void add(int index, E element) {
    rangeCheck(index);

    if (index == 0) {
        first = new Node<>(element, first);
    } else {
        Node<E> prev = node(index - 1);
        prev.next = new Node<>(element, prev.next);
    }

    size++;
}
```

在编写链表过程中，要注意边界测试，比如index为0、size – 1、size时

删除元素 - remove(int index)



size--

小码哥教育 SEEMYGO

删除元素 – 注意0位置

```
public E remove(int index) {
    rangeCheck(index);

    Node<E> node = first;
    if (index == 0) {
        first = first.next;
    } else {
        Node<E> prev = node(index - 1);
        node = prev.next;
        prev.next = node.next;
    }

    size--;

    return node.element;
}
```

推荐一个神奇的网站

- https://visualgo.net/zh

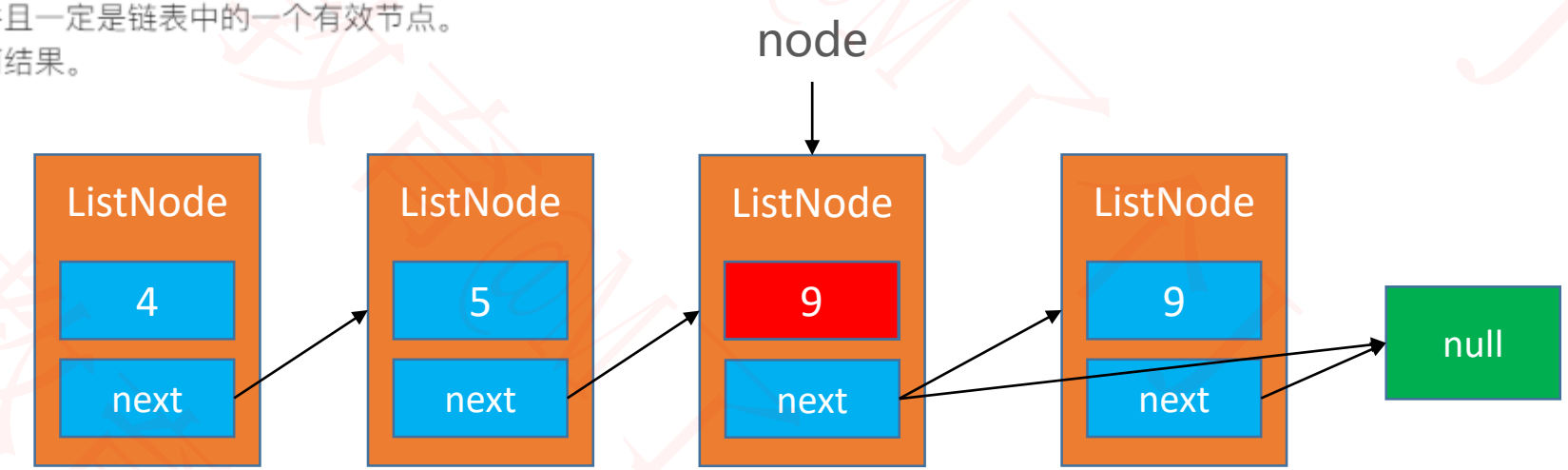
练习 – 删除链表中的节点

- 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

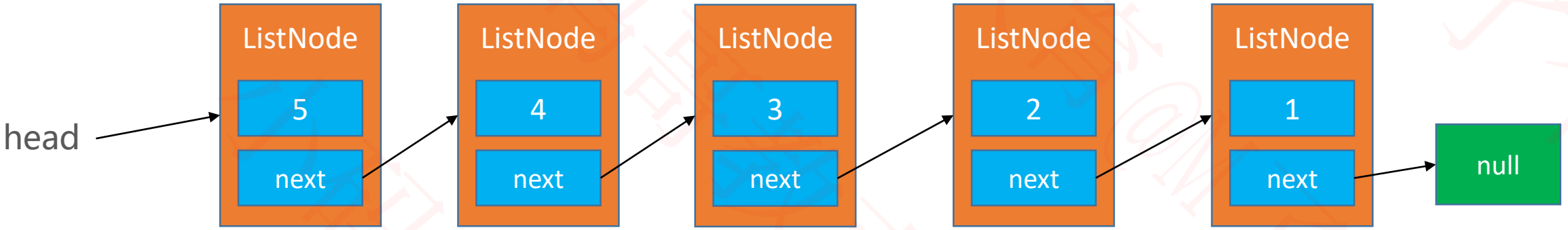
- https://leetcode-cn.com/problems/reverse-linked-list/

示例:

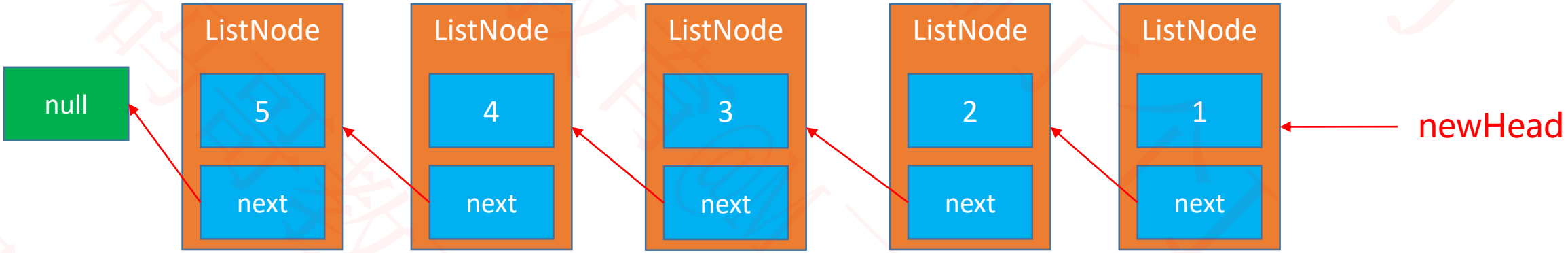
```
输入：1->2->3->4->5->NULL
输出：5->4->3->2->1->NULL
```

- 请分别用递归、迭代（非递归）两种方式实现

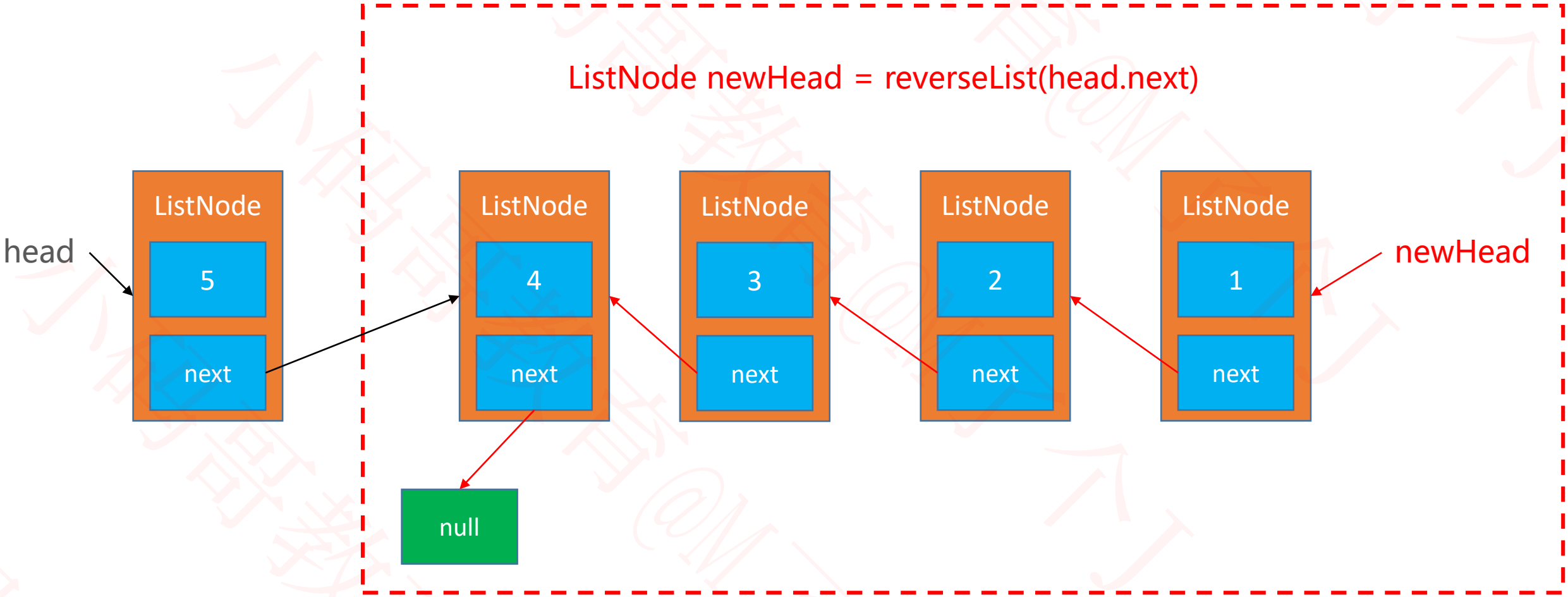
练习 – 反转一个链表 – 递归



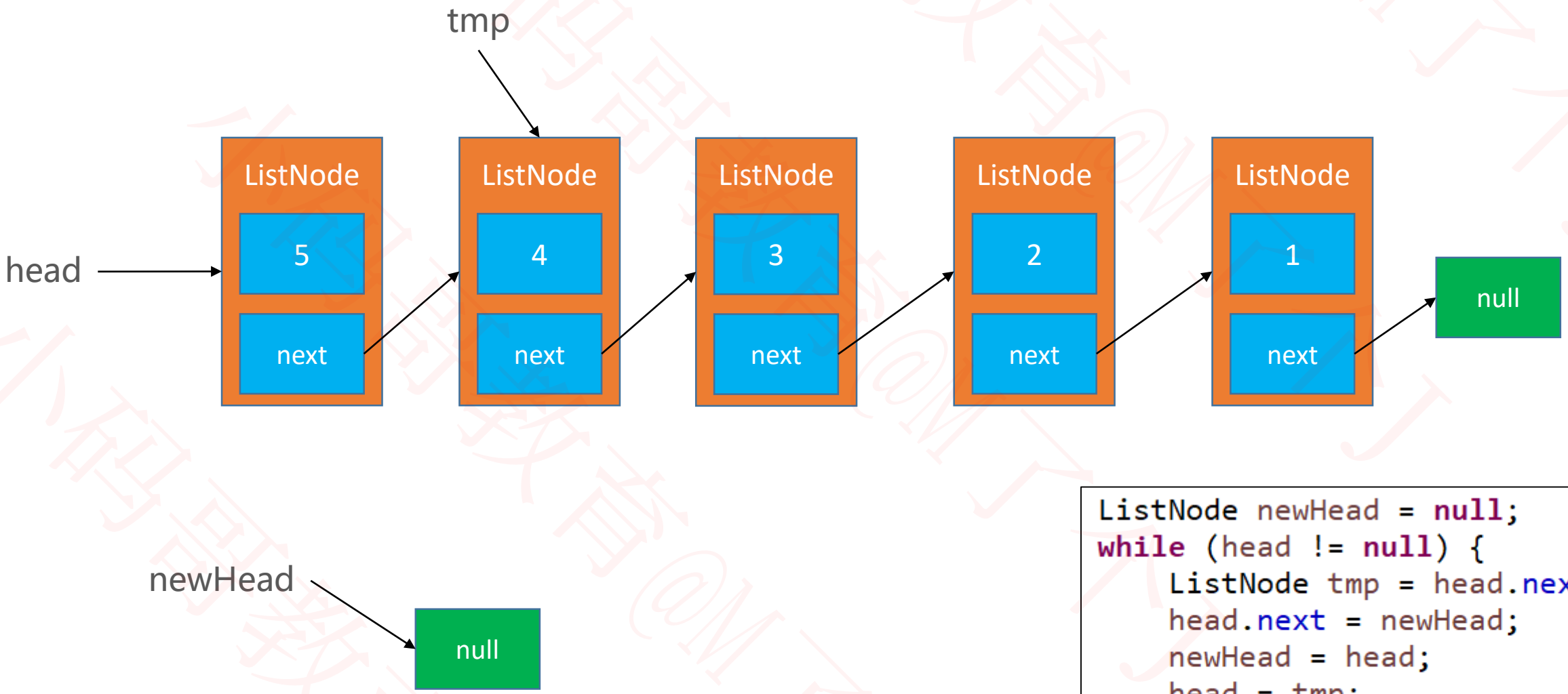
ListNode newHead = reverseList(head)



练习 – 反转一个链表 – 递归


ListNode newHead = reverseList(head.next)
head
ListNode
5
next
ListNode
4
next
ListNode
3
next
ListNode
2
next
ListNode
1
next
newHead
null

练习 – 反转一个链表 – 非递归



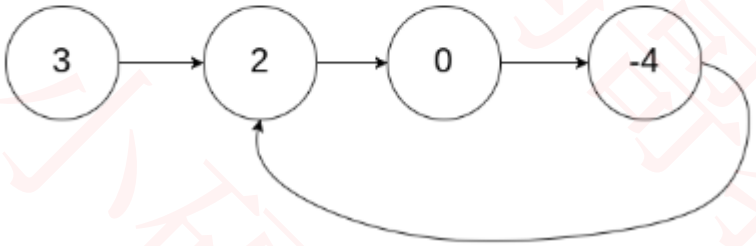

```
ListNode newHead = null;
while (head != null) {
    ListNode tmp = head.next;
    head.next = newHead;
    newHead = head;
    head = tmp;
}
```

练习 – 判断一个链表是否有环

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

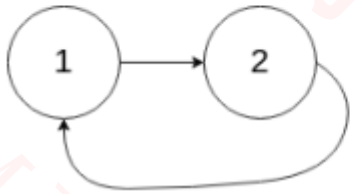
示例 1：

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。

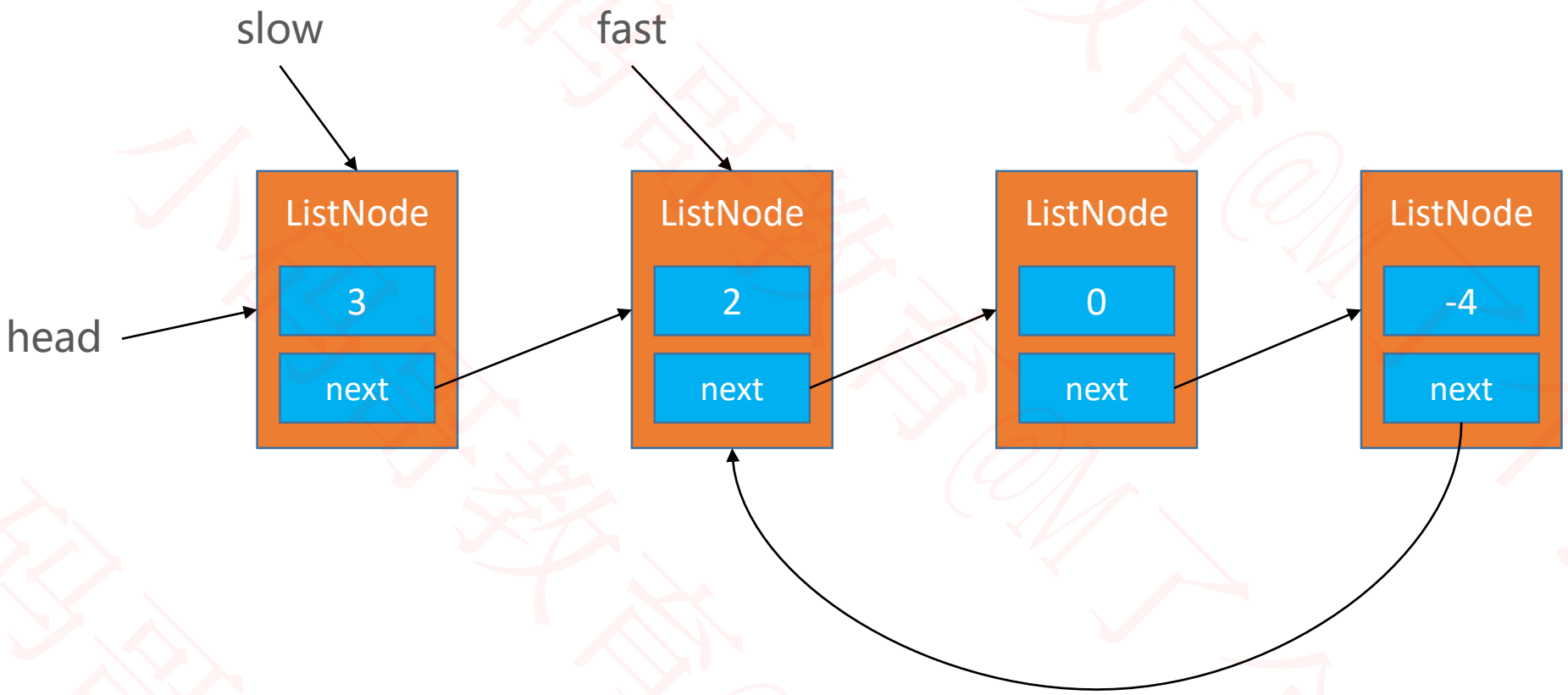


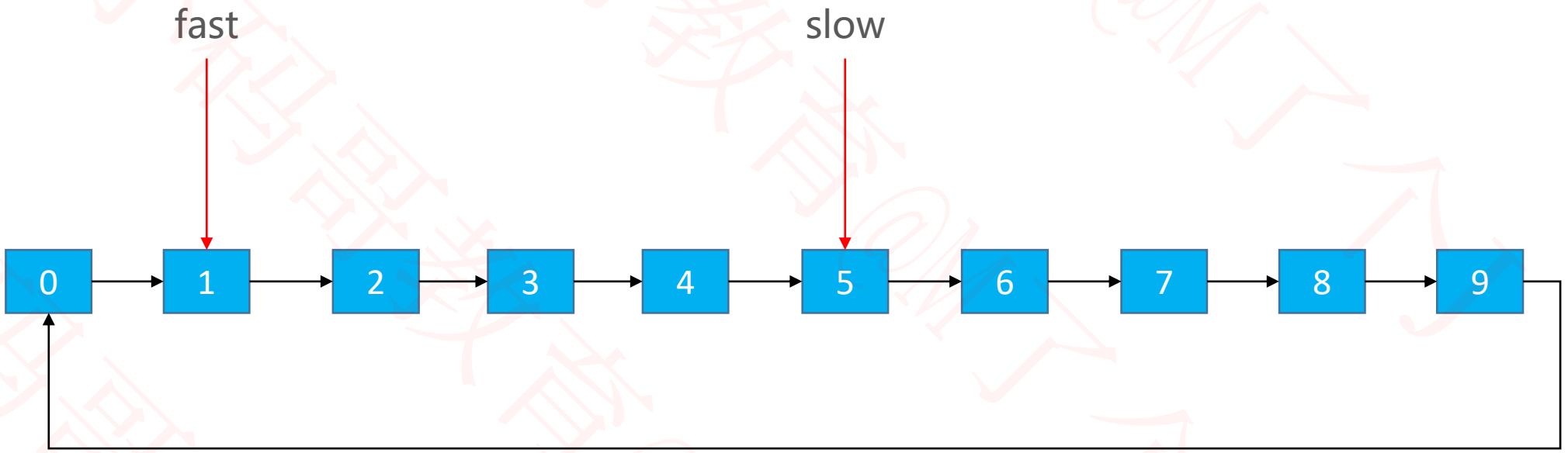
示例 3：

输入： head = [1], pos = -1
输出： false
解释： 链表中没有环。

小码哥教育 SEEMYGO

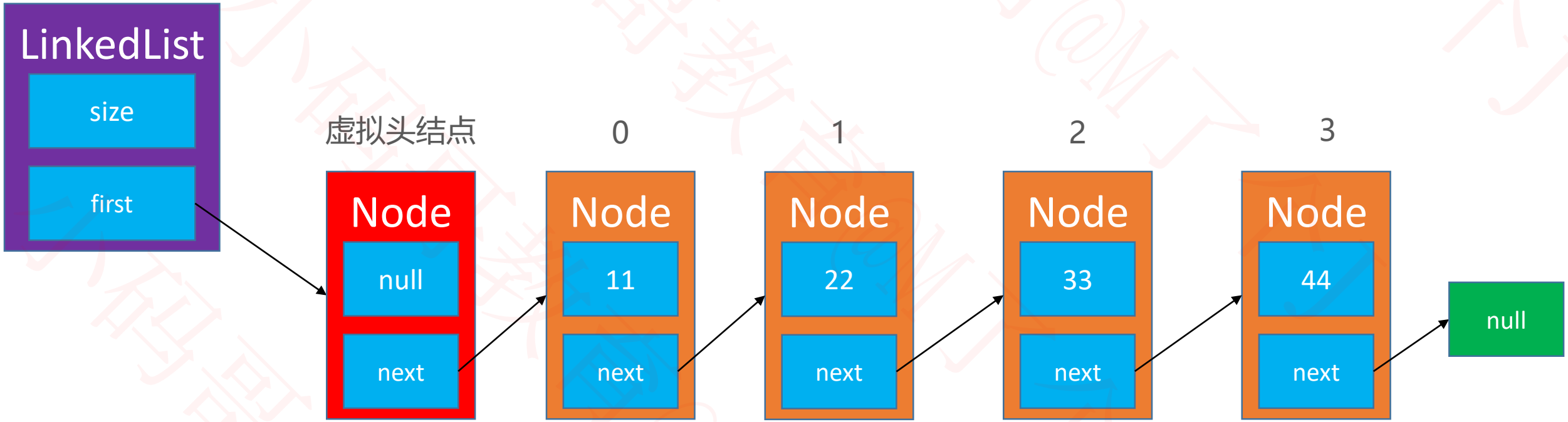
练习 – 判断一个链表是否有环





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）




```
public LinkedList() {
    first = new Node<>(null, null);
}
```

小码哥教育 SEEMYGO

虚拟节点 – node方法

```
private Node<E> node(int index) {
    rangeCheck(index);

    Node<E> node = first.next;
    for (int i = 0; i < index; i++) {
        node = node.next;
    }
    return node;
}
```

小码哥教育 SEEMYGO

虚拟节点 - 添加、删除

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    Node<E> prev = (index == 0) ? first : node(index - 1);
    prev.next = new Node<>(element, prev.next);

    size++;
}
```

```
public E remove(int index) {
    rangeCheck(index);

    Node<E> prev = (index == 0) ? first : node(index - 1);
    Node<E> node = prev.next;
    prev.next = node.next;

    size--;
    return node.element;
}
```

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

索引	元素
0	11
1	22
2	33
3	44
4	55
5	66

elements[4]

随机访问速度非常快

小码哥教育 SEEMYGO

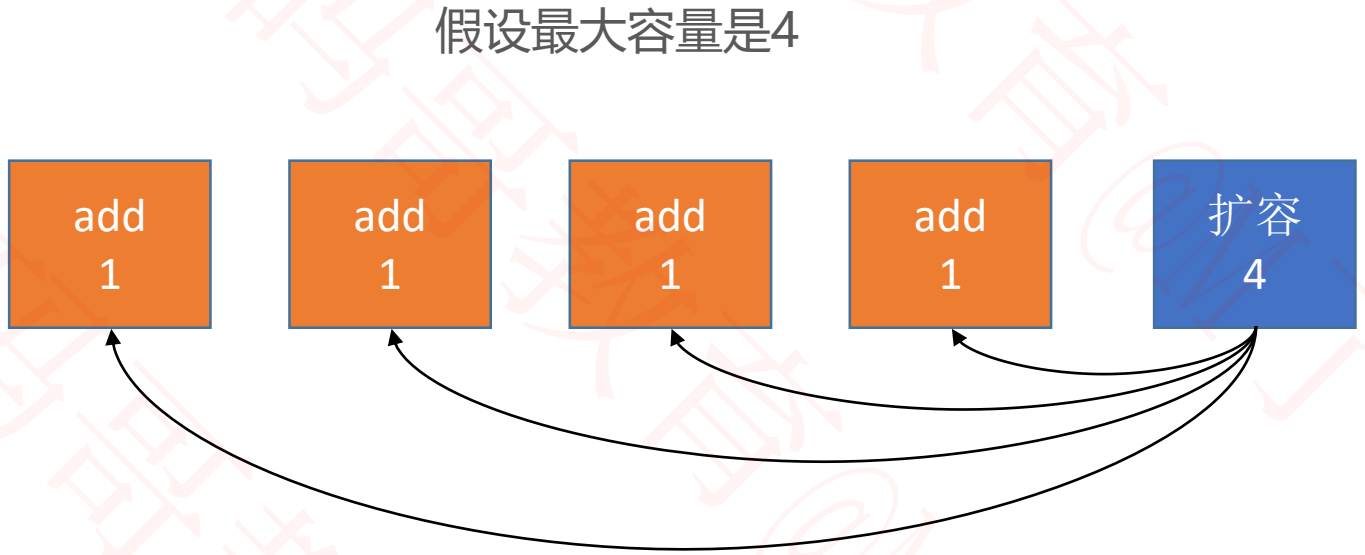
动态数组、链表复杂度分析

	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)



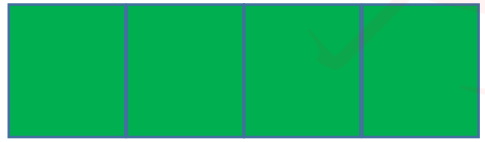
相当于每次add的操作次数是2，也就是O(1)复杂度

- 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

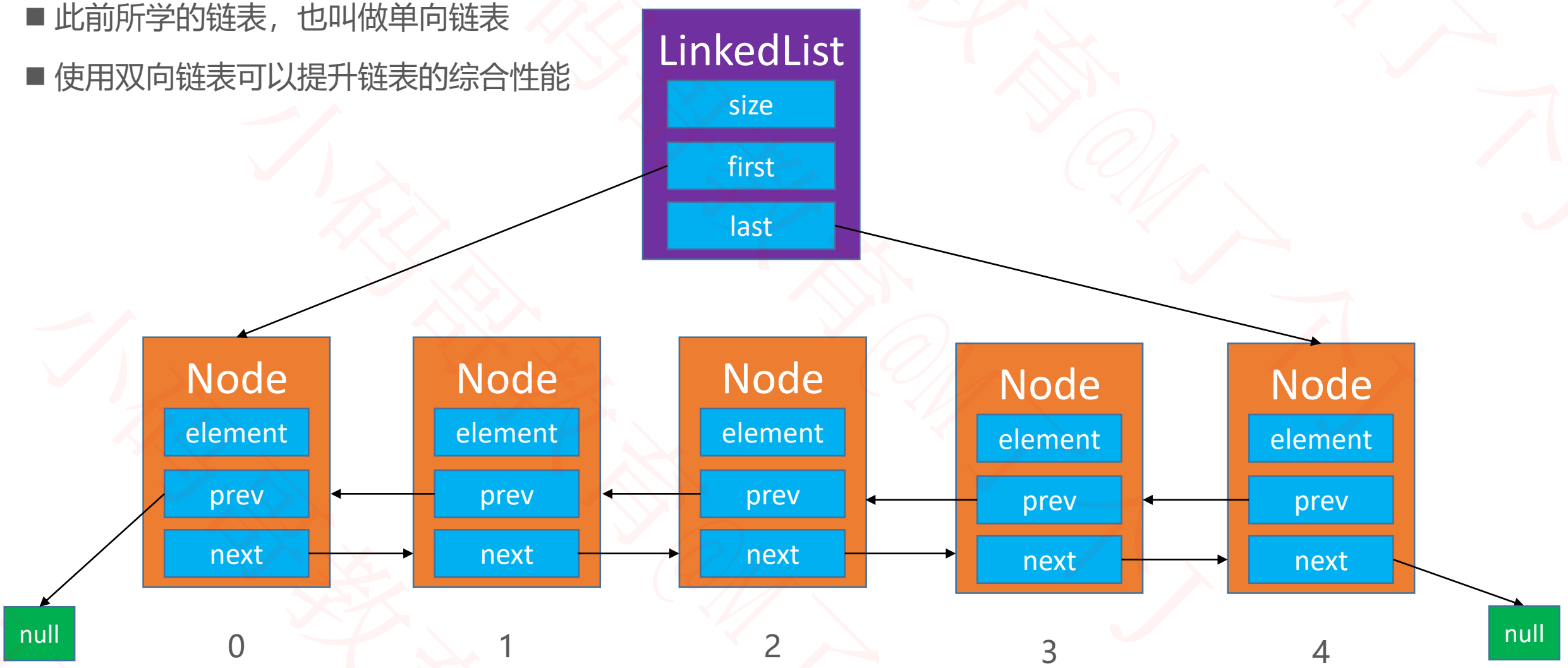
- 如果内存使用比较紧张，动态数组有比较多的剩余空间，可以考虑进行缩容操作
  - 比如剩余空间占总容量的一半时，就进行缩容

- 如果扩容倍数、缩容时机设计不得当，有可能会导致复杂度震荡

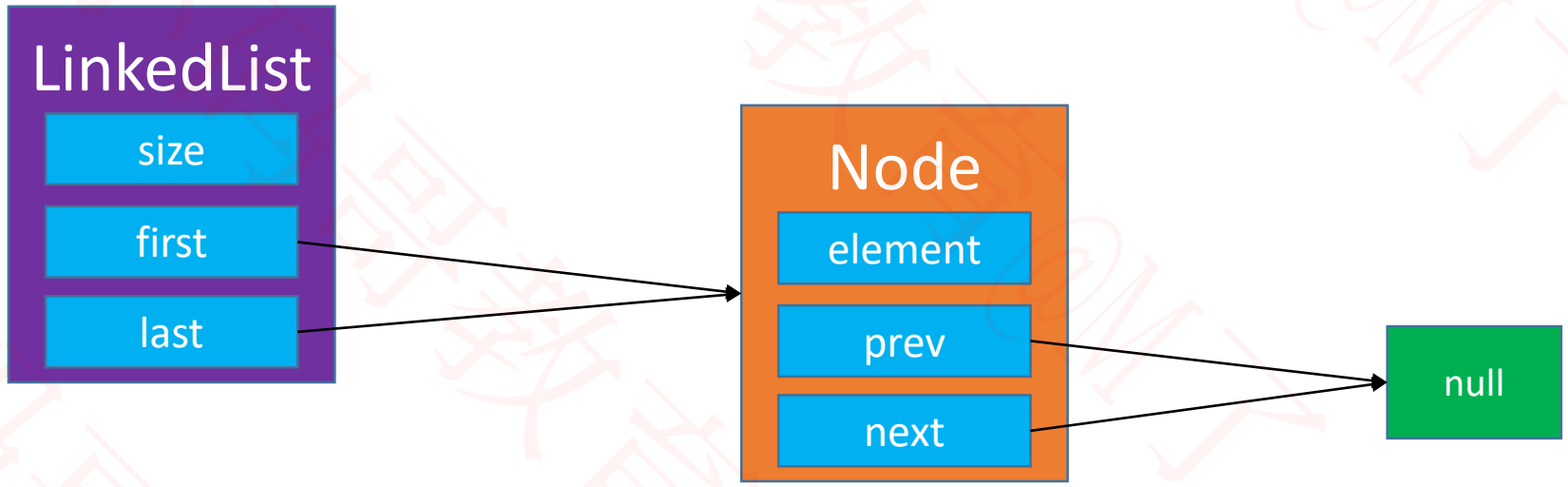


双向链表

- 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综合性能



双向链表 – 只有一个元素

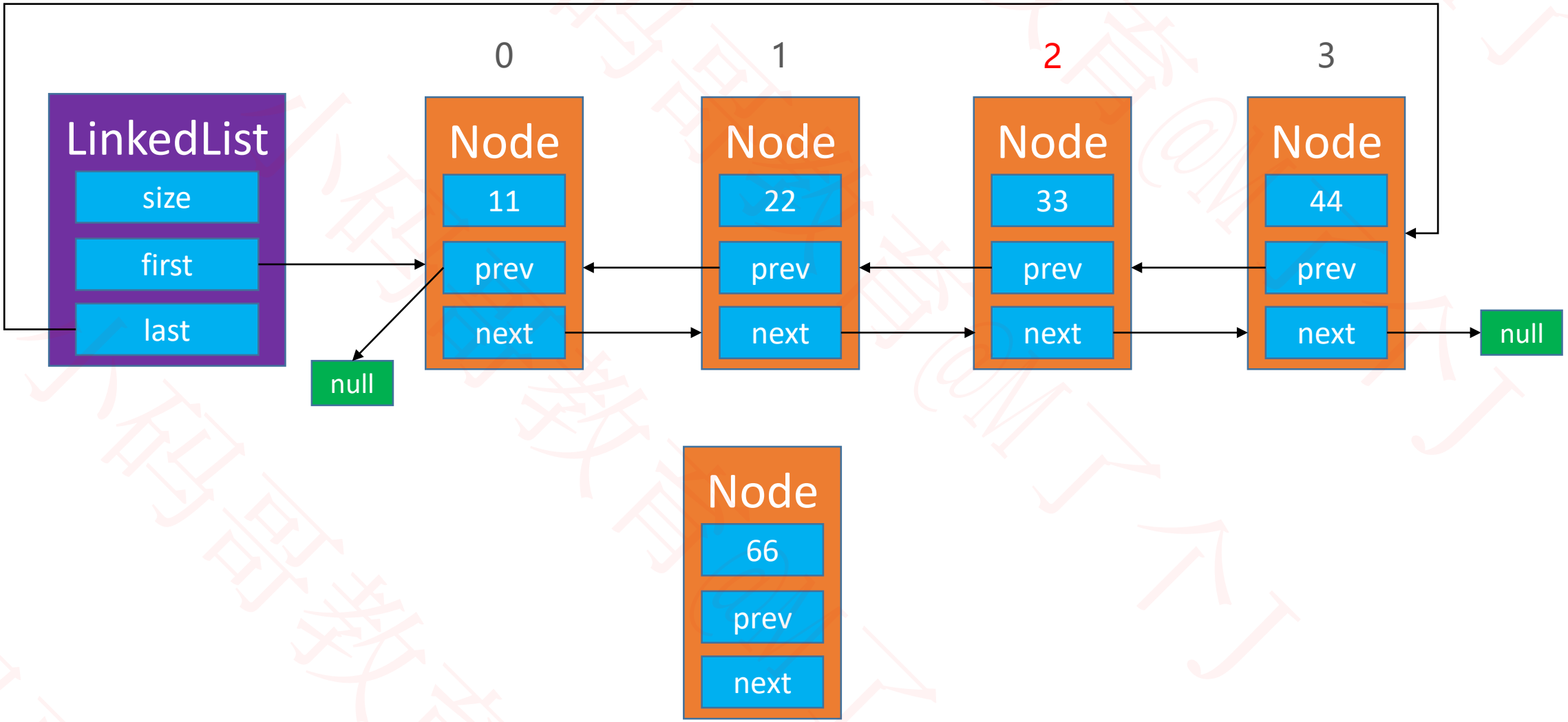


小码哥教育 SEEMYGO

双向链表 – node方法

```
private Node<E> node(int index) {
    rangeCheck(index);

    if (index < (size >> 1)) {
        Node<E> node = first;
        for (int i = 0; i < index; i++) {
            node = node.next;
        }
        return node;
    } else {
        Node<E> node = last;
        for (int i = size - 1; i > index; i--) {
            node = node.prev;
        }
        return node;
    }
}
```

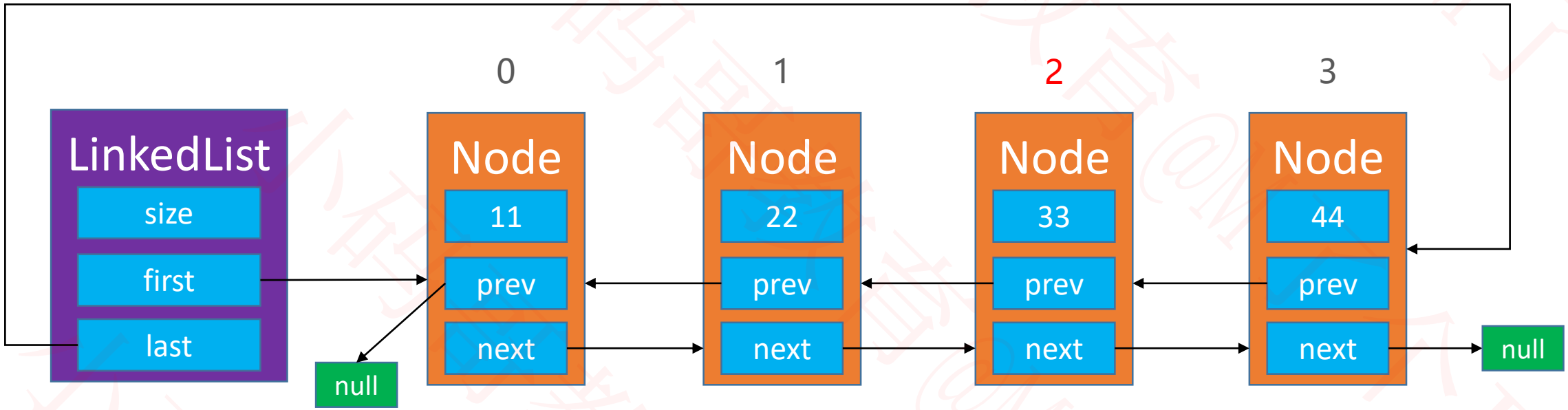
双向链表 – add(int index, E element)



双向链表 – add(int index, E element)

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    if (index == size) {
        Node<E> oldLast = last;
        last = new Node<>(oldLast, element, null);
        if (oldLast == null) {
            first = last;
        } else {
            oldLast.next = last;
        }
    } else {
        Node<E> next = node(index);
        Node<E> prev = next.prev;
        Node<E> current = new Node<>(prev, element, next);
        next.prev = current;

        if (prev == null) {
            first = current;
        } else {
            prev.next = current;
        }
    }
    size++;
}
```

双向链表 – remove(int index)



小码哥教育 SEEMYGO

双向链表 – remove(int index)

```
public E remove(int index) {
    rangeCheck(index);

    Node<E> node = node(index);
    Node<E> prev = node.prev;
    Node<E> next = node.next;

    if (prev == null) {
        first = next;
    } else {
        prev.next = next;
    }

    if (next == null) {
        last = prev;
    } else {
        next.prev = prev;
    }

    size--;

    return node.element;
}
```

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

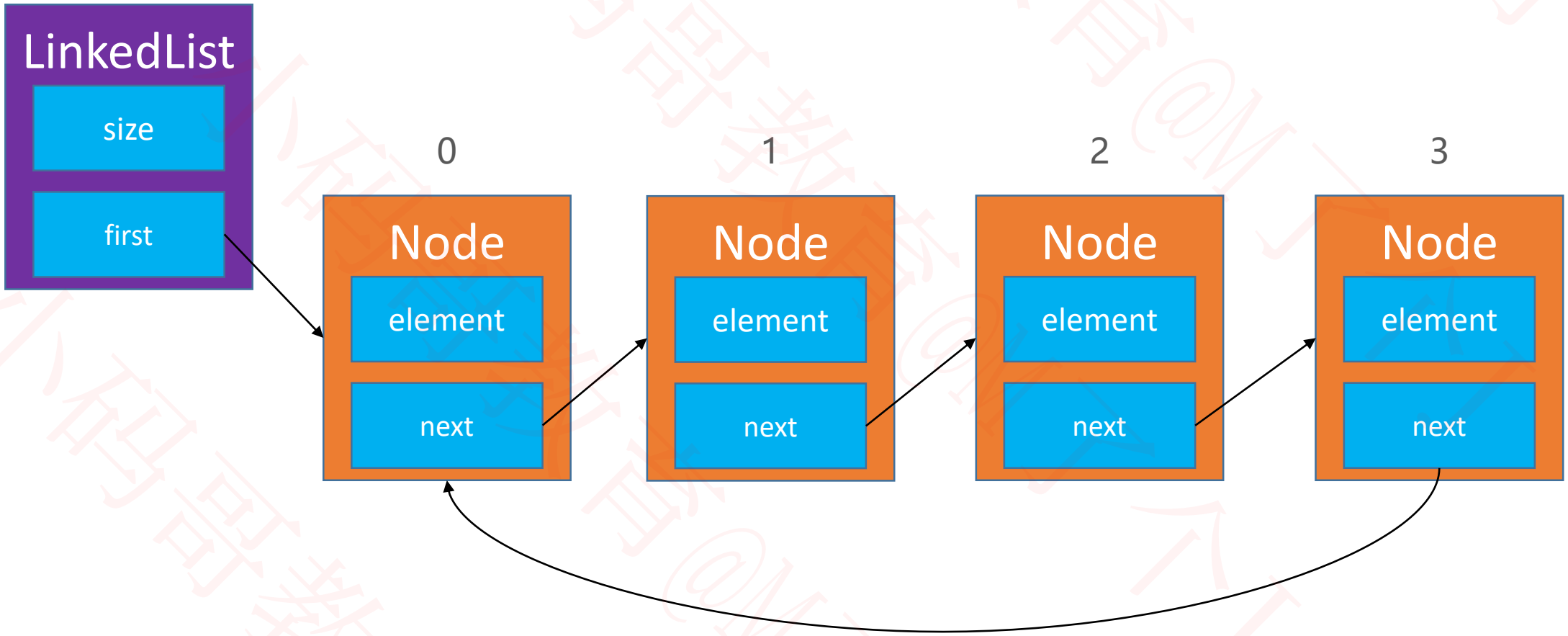
- 动态数组：开辟、销毁内存空间的次数相对较少，但可能造成内存空间浪费（可以通过缩容解决）
- 双向链表：开辟、销毁内存空间的次数相对较多，但不会造成内存空间的浪费

- 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
- 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

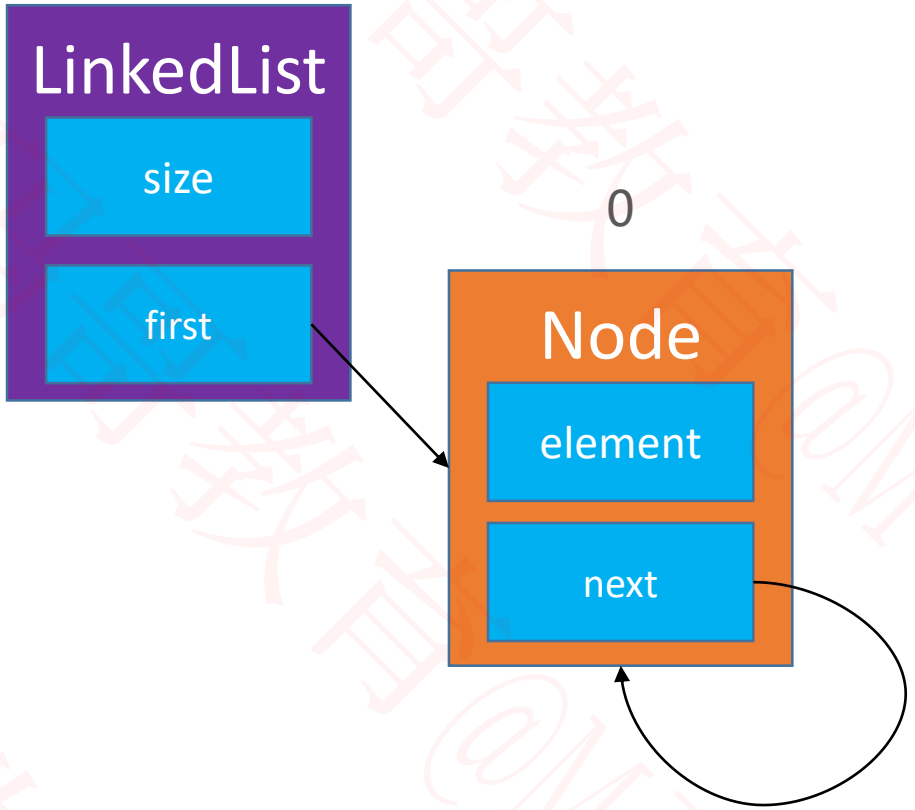
- 有了双向链表，单向链表是否就没有任何用处了？
  - 并非如此，在哈希表的设计中就用到了单链表
  - 至于原因，在哈希表章节中会讲到

小码哥教育 SEEMYGO

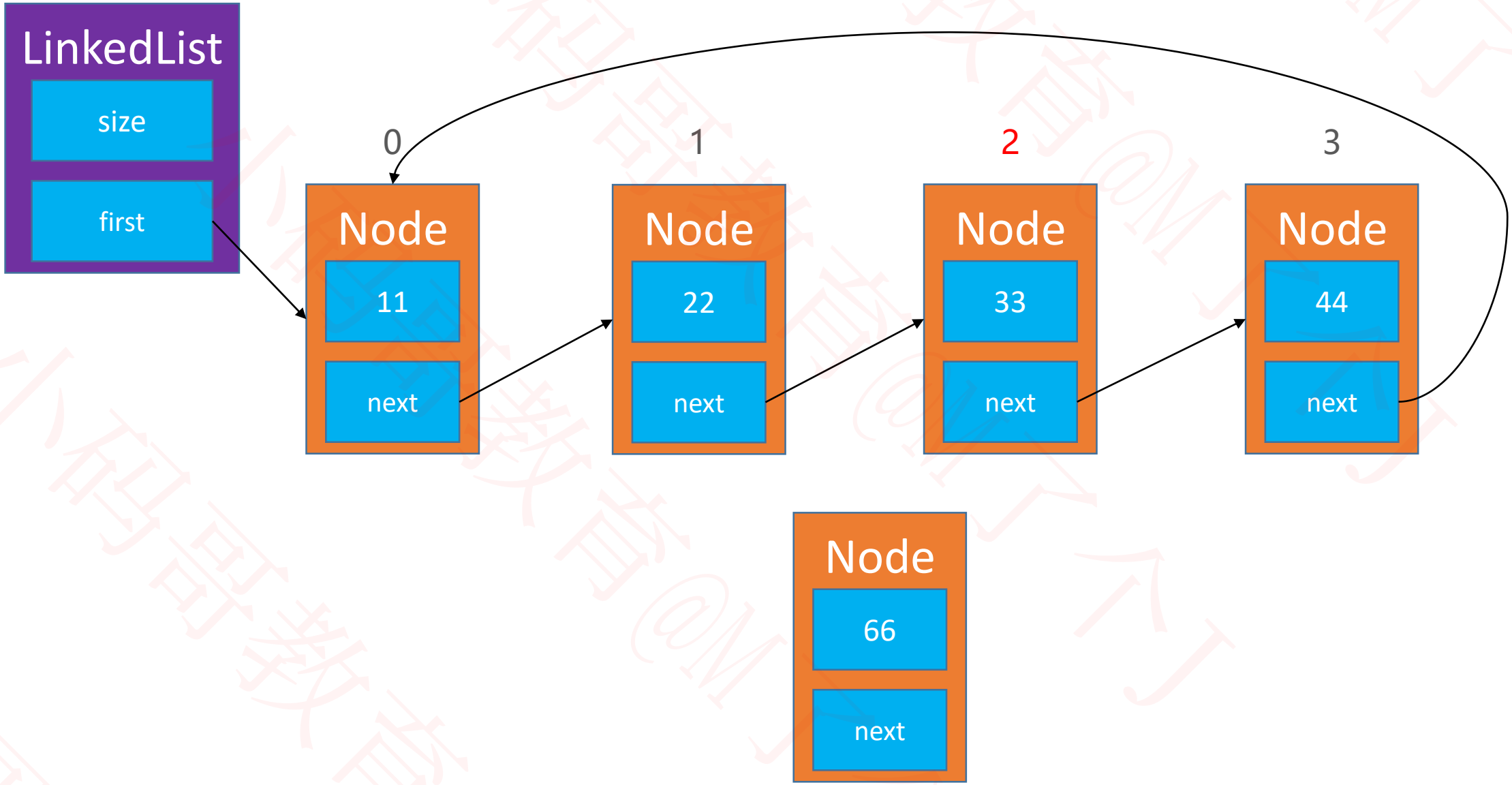
单向循环链表



单向循环链表 – 只有1个节点



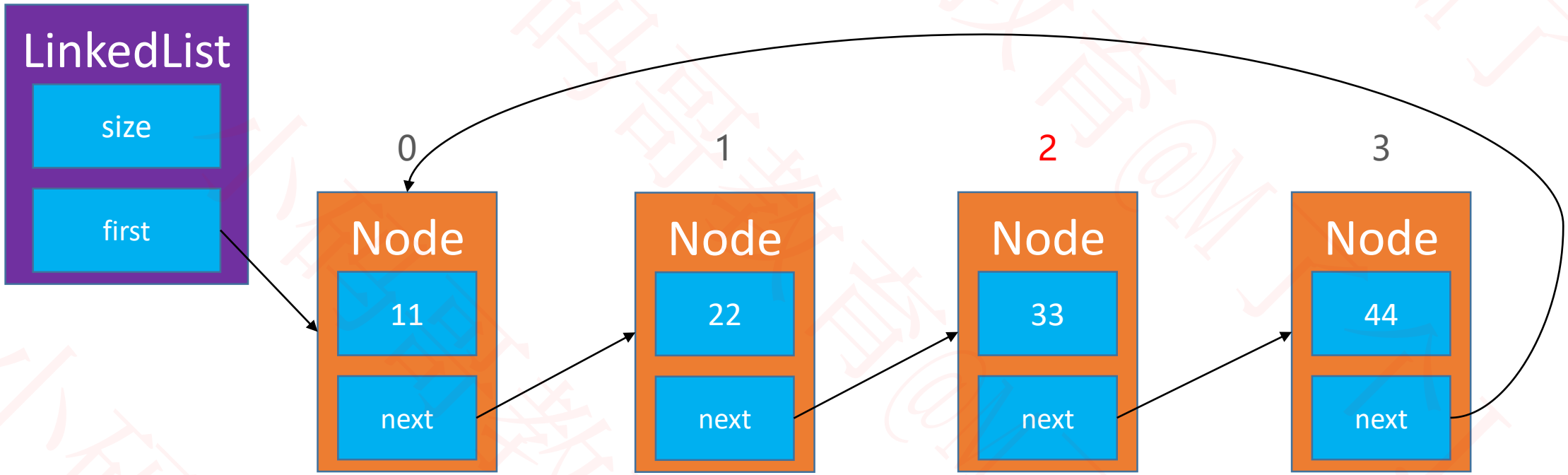
单向循环链表 – add(int index, E element)



单向循环链表 – add(int index, E element)

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    if (index == 0) {
        first = new Node<>(element, first);
        Node<E> last = (size == 0) ? first : node(size - 1);
        last.next = first;
    } else {
        Node<E> prev = node(index - 1);
        prev.next = new Node<>(element, prev.next);
    }
    size++;
}
```

单向循环链表 – remove(int index)



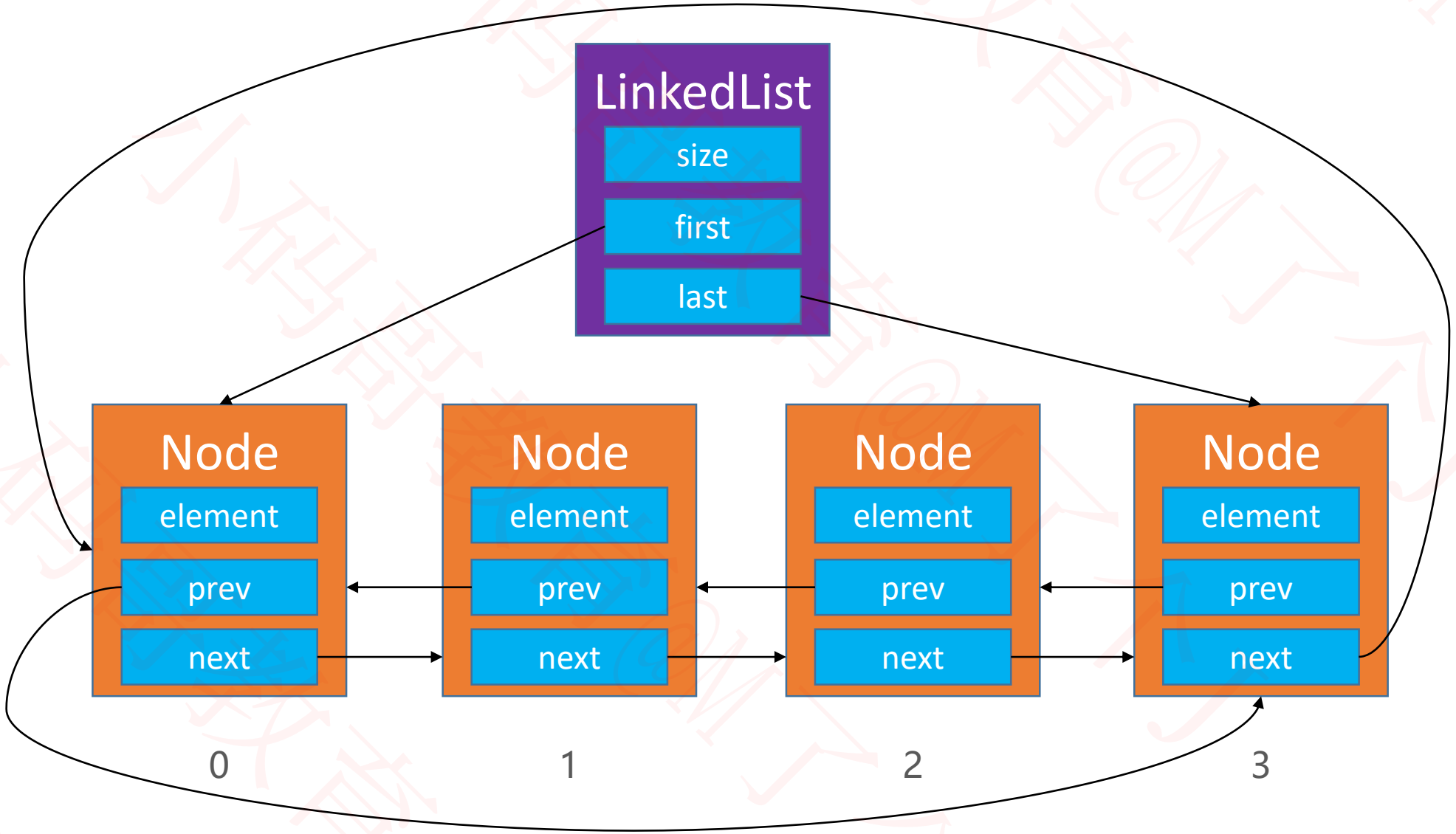
小码哥教育 SEEMYGO

单向循环链表 – remove(int index)

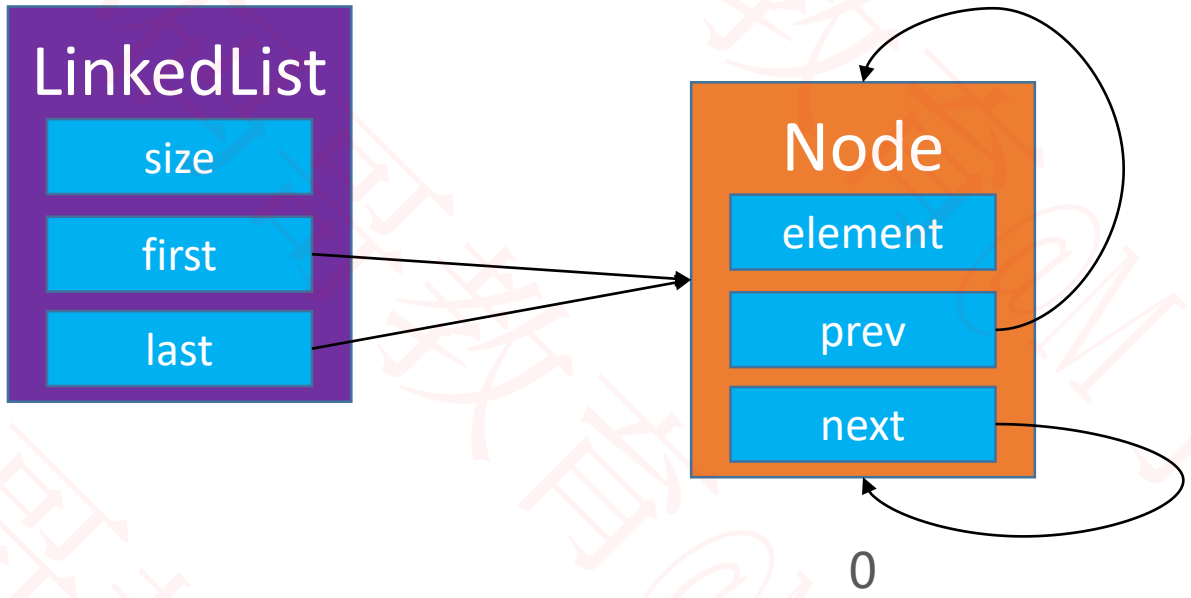
```
public E remove(int index) {
    rangeCheck(index);

    Node<E> node = first;
    if (index == 0) {
        if (size == 1) {
            first = null;
        } else {
            Node<E> last = node(size - 1);
            first = first.next;
            last.next = first;
        }
    } else {
        Node<E> previous = node(index - 1);
        node = previous.next;
        previous.next = node.next;
    }

    size--;

    return node.element;
}
```

小码哥教育 SEEMYGO

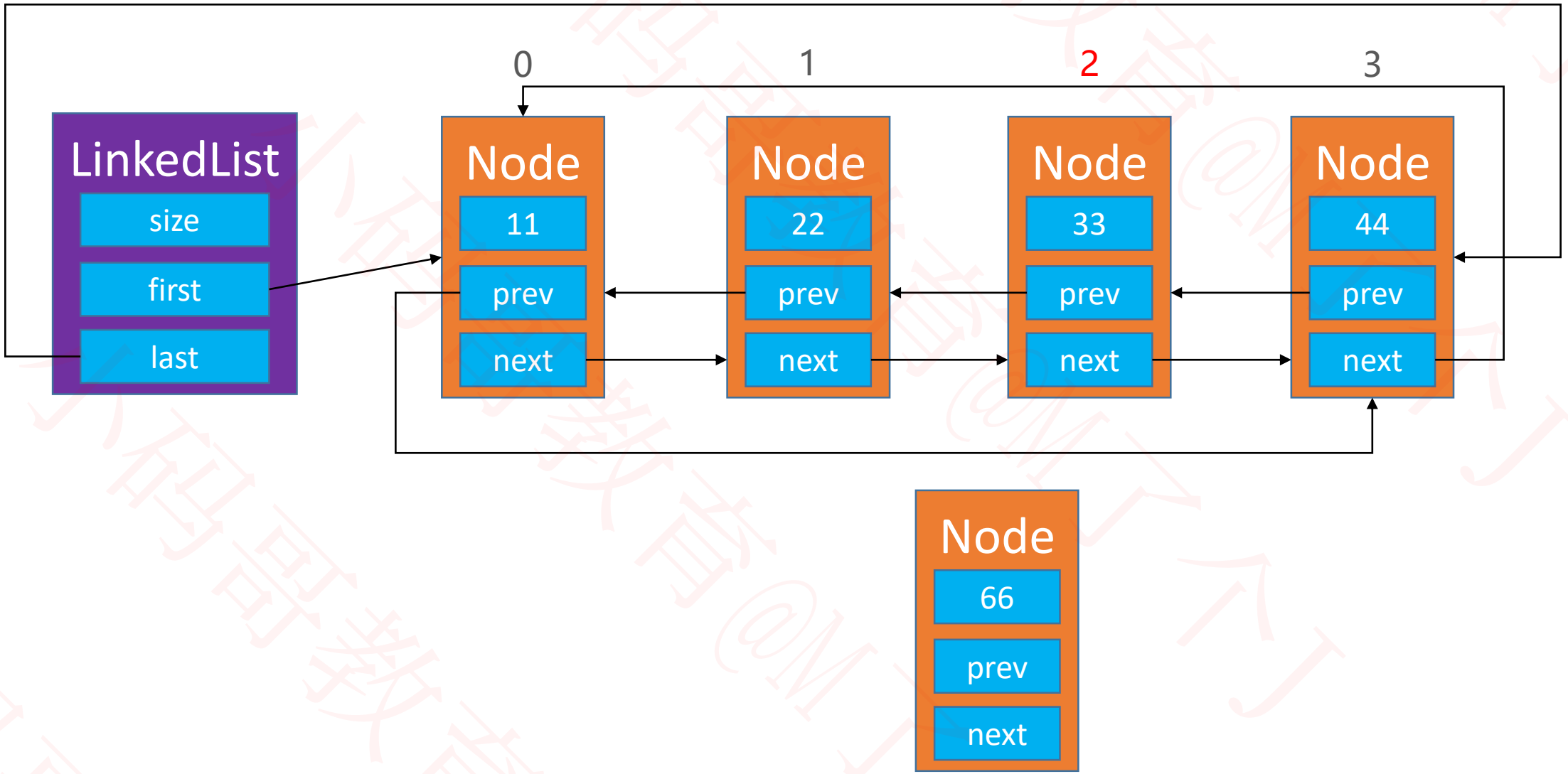
双向循环链表



双向循环链表 – 只有1个节点



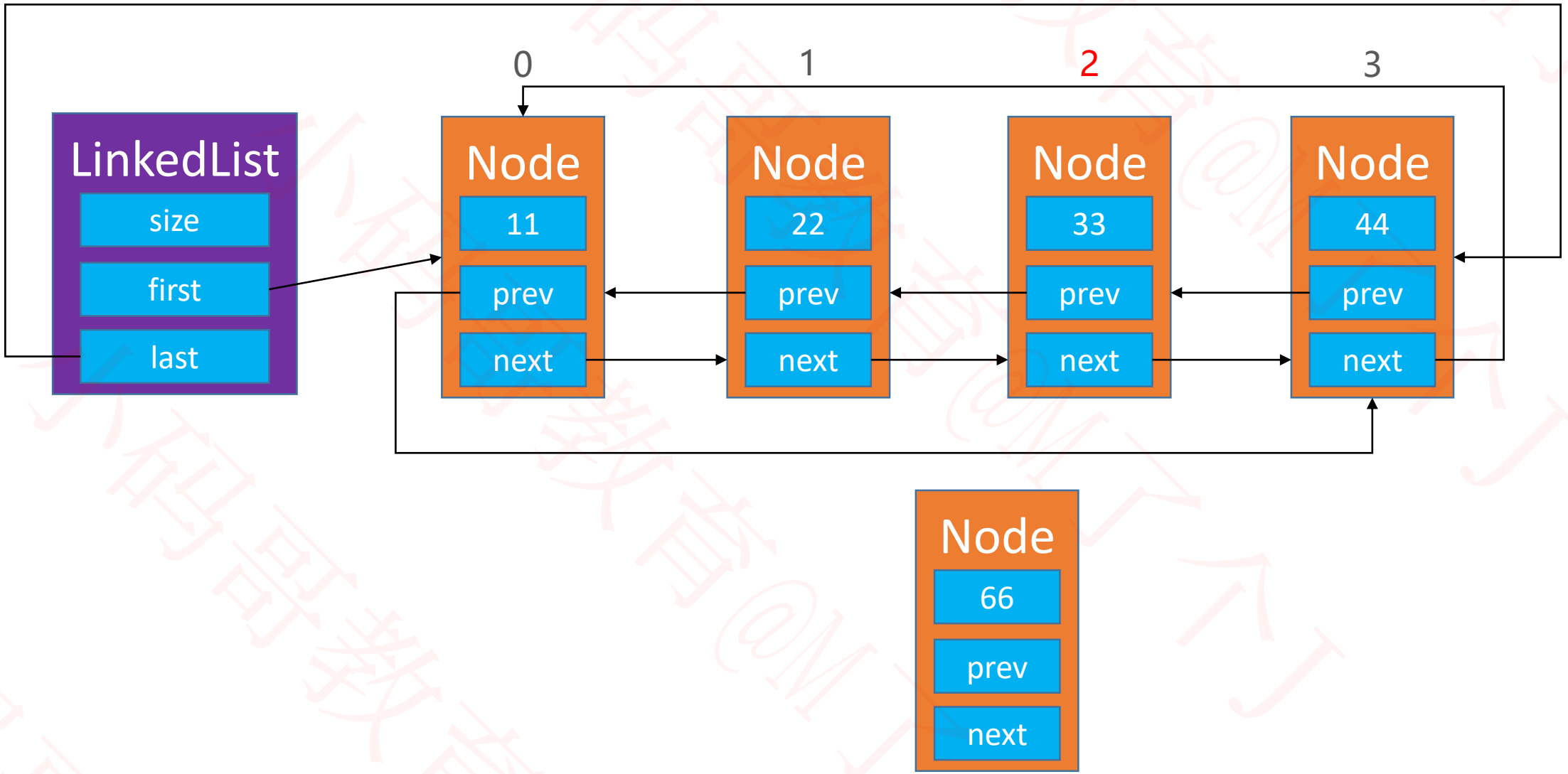
双向循环链表 – add(int index, E element)



双向循环链表 – add(int index, E element)

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    if (index == size) {
        Node<E> oldLast = last;
        last = new Node<>(oldLast, element, first);
        if (oldLast == null) {
            first = last;
            first.next = first;
            first.prev = first;
        } else {
            oldLast.next = last;
            first.prev = last;
        }
    } else {
        Node<E> next = node(index);
        Node<E> prev = next.prev;
        Node<E> current = new Node<>(prev, element, next);
        next.prev = current;
        prev.next = current;

        if (next == first) {
            first = current;
        }
    }
    size++;
}
```

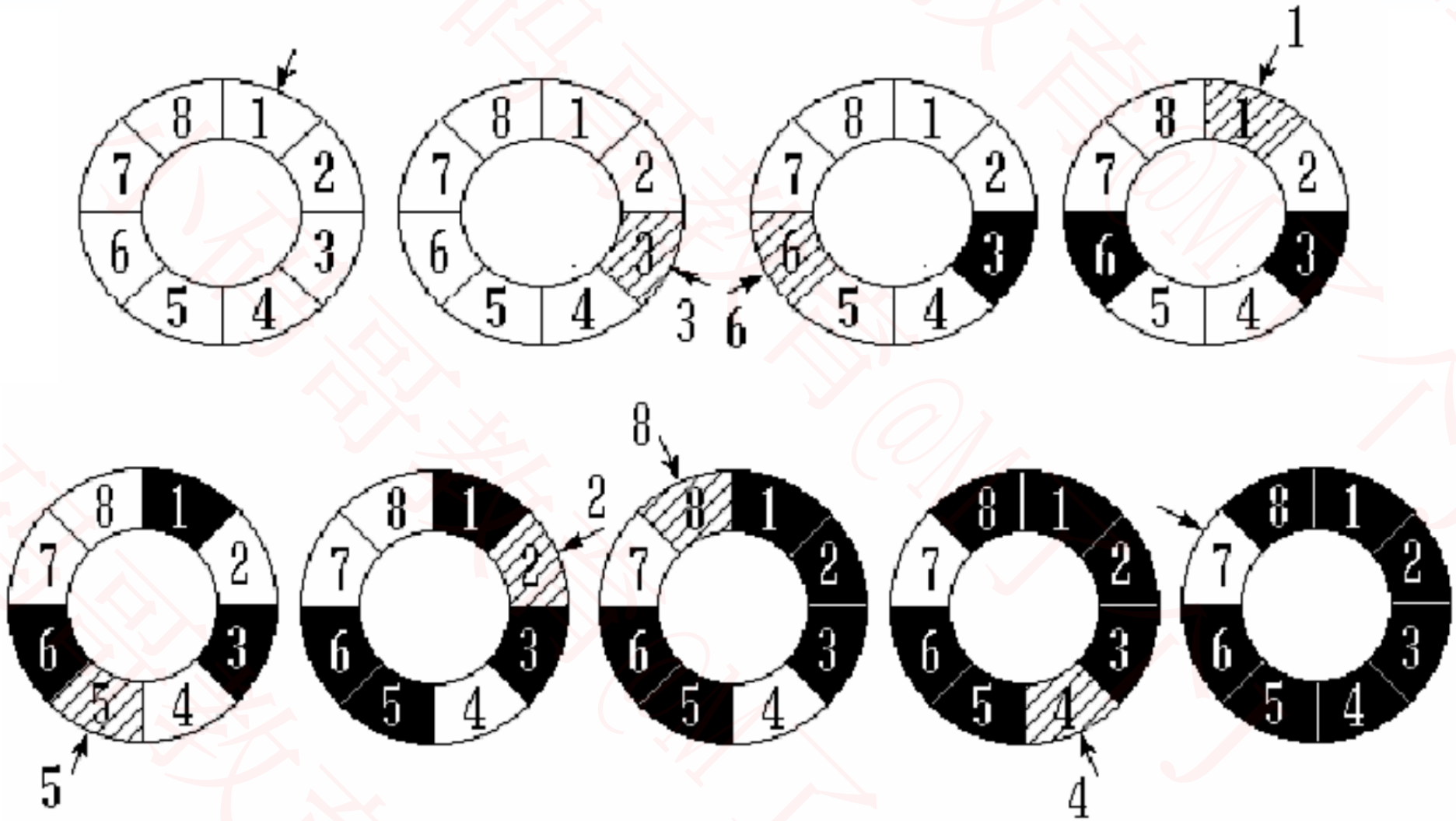
双向循环链表 – remove(int index)



双向循环链表 – remove(int index)

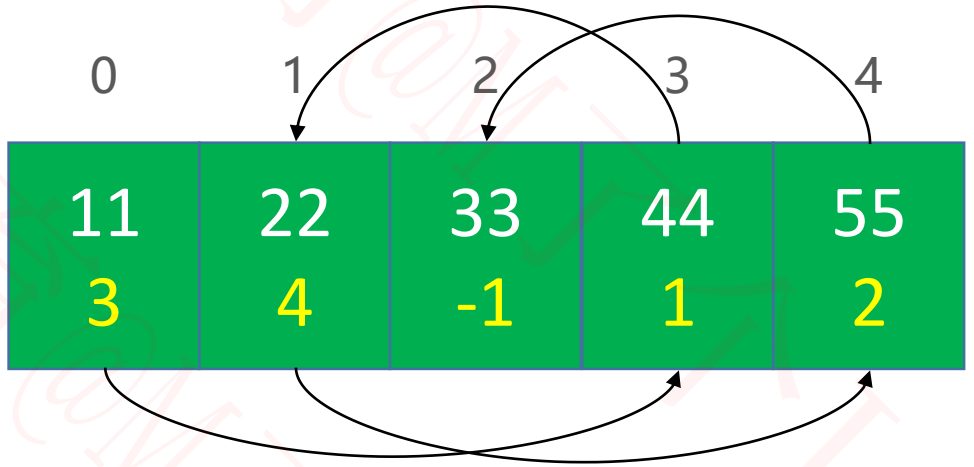
```
public E remove(int index) {
    rangeCheck(index);

    Node<E> node = node(index);
    if (size == 1) {
        first = null;
        last = null;
    } else {
        Node<E> prev = node.prev;
        Node<E> next = node.next;
        prev.next = next;
        next.prev = prev;

        if (node == first) {
            first = next;
        }

        if (node == last) {
            last = prev;
        }
    }
    size--;
    return node.element;
}
```

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



小码哥教育 SEEMYGO

作业

- 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/

作业 – ArrayList能否进一步优化?