

算法与数据结构体系课程

liuyubobobo

链表与递归

慕课网 算法与数据结构体系课程
liuyubobobo 是不是很酷
公众号

从Leetcode上一个问题开始

慕课网 算法与数据结构体系课程
liuyubobobo
公众号【是不是很酷】

203. 删除链表中的元素

在链表中删除值为val的所有节点

- 如 $1 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \text{NULL}$, 要求删除值为6的节点
- 返回 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \text{NULL}$

实践：解决203，不使用虚拟头结点

慕课网 计算机数据结构体系课程
liuyubobobo（是不是很酷）
公众号

实践：测试leetcode上的链表程序

慕课网 算法和数据结构体系课程
liuyubobobo 是不是很酷
公众号

实践：解决203，使用虚拟头结点

慕课网 计算机数据结构体系课程
liuyubobobo 是不是很酷
公众号

递归与递归的宏观语意

慕课网 算法与数据结构体系课程
liuyubobobo【是不是很酷】
公众号

递归

- 本质上，将原来的问题，转化为更小的同一问题
- 举例：数组求和

$$\text{Sum}(\text{arr}[0...n-1]) = \text{arr}[0] + \text{Sum}(\text{arr}[1...n-1]) \quad \leftarrow \text{更小的同一问题}$$

$$\text{Sum}(\text{arr}[1...n-1]) = \text{arr}[1] + \text{Sum}(\text{arr}[2...n-1]) \quad \leftarrow \text{更小的同一问题}$$

.....

$$\text{Sum}(\text{arr}[n-1...n-1]) = \text{arr}[n-1] + \text{Sum}([]) \quad \leftarrow \text{最基本的问题}$$

实践：和递归数组求和

慕课网 算法和数据结构体系课程
liuyubobobo（是不是很酷）
公众号

递归

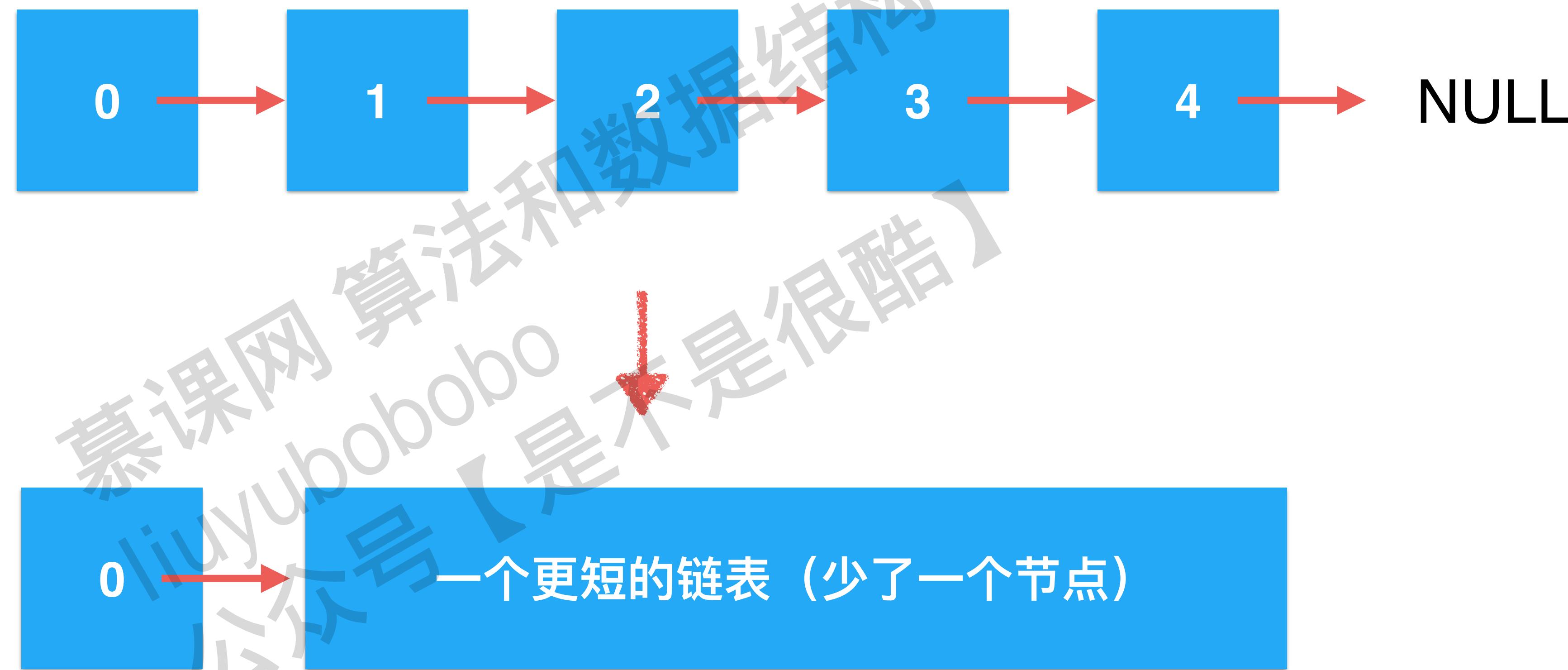
- 注意递归函数的“宏观”语意
 - 递归函数就是一个函数，完成一个功能

```
// 计算 arr[l...n] 范围里的数字和
public static int sum(int[] arr, int l){
    if(l == arr.length)
        return 0;
    return arr[l] + sum(arr, l + 1);
}
```

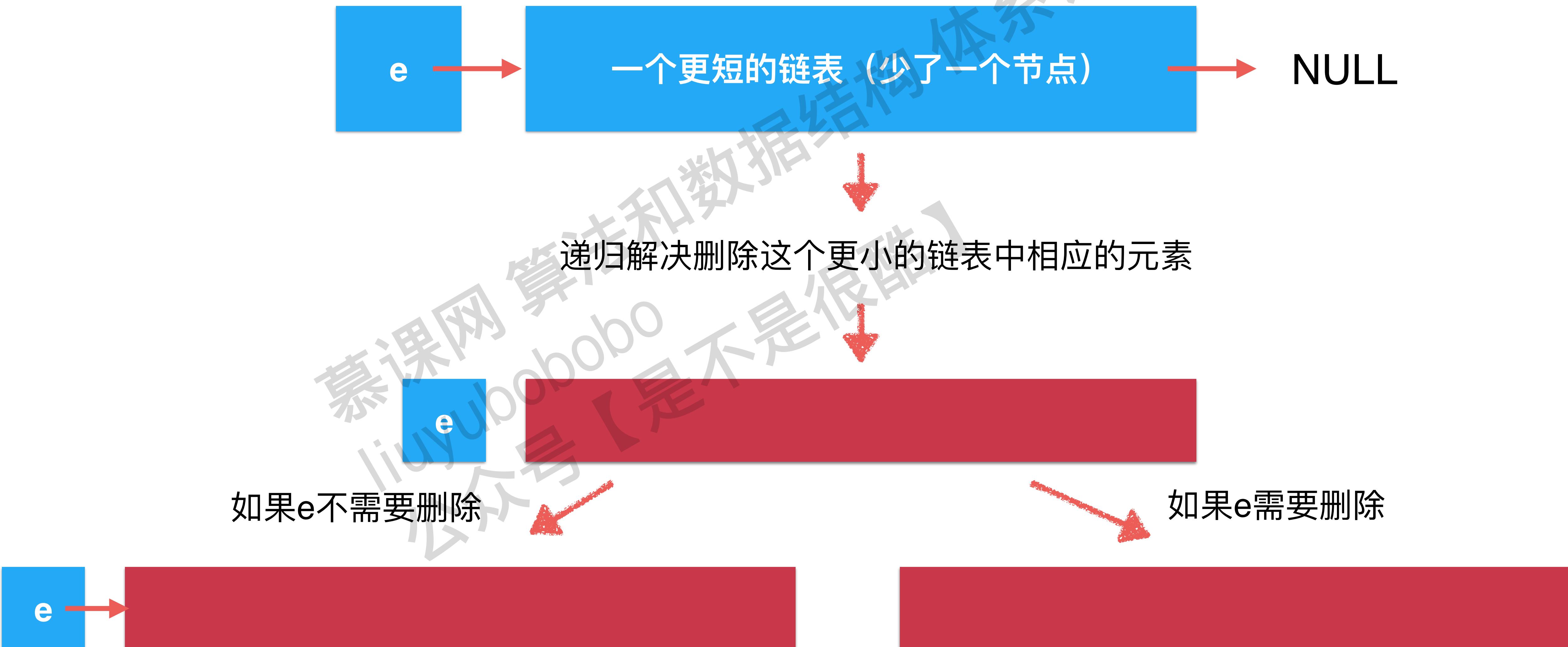
链表和递归

慕课网 算法与数据结构体系课程
liuyubobobo 是不是很酷
公众号

链表天然的递归性



解决链表中删除元素的问题



实践：Leetcode 203 使用递归思路求解

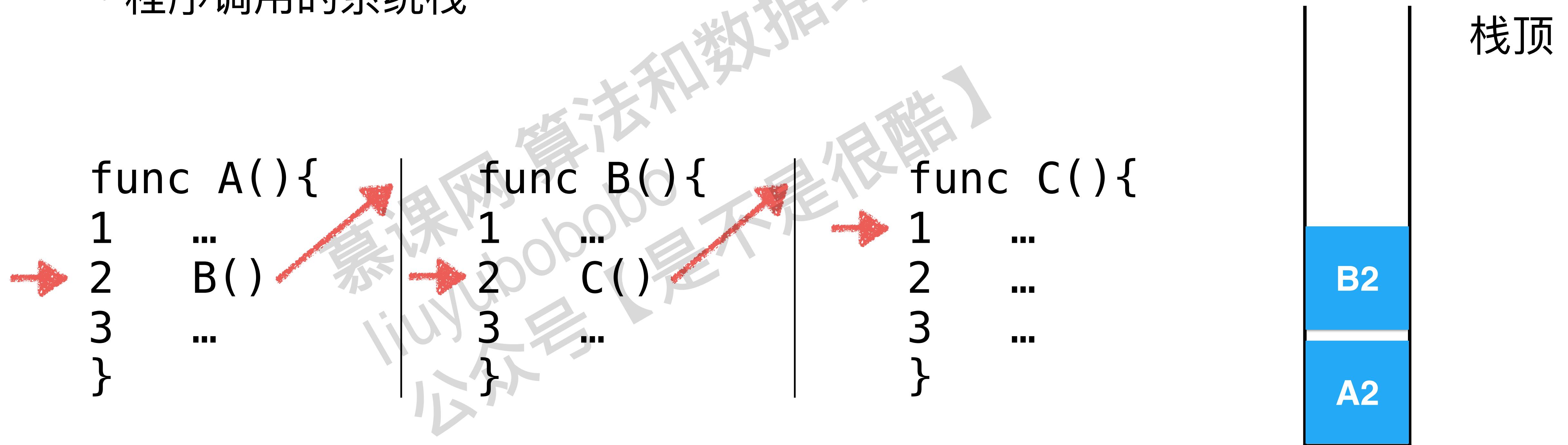
慕课网 算法与数据结构体系课程
liuyubobobo【是不是很酷】
公众号

递归函数的“微观”解读

慕课网 算法与数据结构体系课程
liuyubobobo【是不是很酷】
公众号

栈的应用

- 程序调用的系统栈



递归函数的“微观”解读

```
public static int sum(int[] arr, int l){  
    if(l == arr.length)  
        return 0;  
    return arr[l] + sum(arr, l + 1);  
}
```

递归函数的“微观”解读

- 递归函数的调用，本质就是函数调用
- 只不过调用的函数是自己而已

```
public static int sum(int[] arr, int l){  
    if(l == arr.length)  
        return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    → if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

递归函数的“微观”解读

```
arr = [6, 10]
```

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

res = 10

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

x = 0

res = 10

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

res = 10

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

x = 10

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

res = 10

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

x = 10

res = 16

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

res = 10

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

x = 10

res = 16

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

res = 10

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

递归函数的“微观”解读

arr = [6, 10]

调用sum(arr, 0)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;  
}
```

x = 10

res = 16

调用sum(arr, 1)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

x = 0

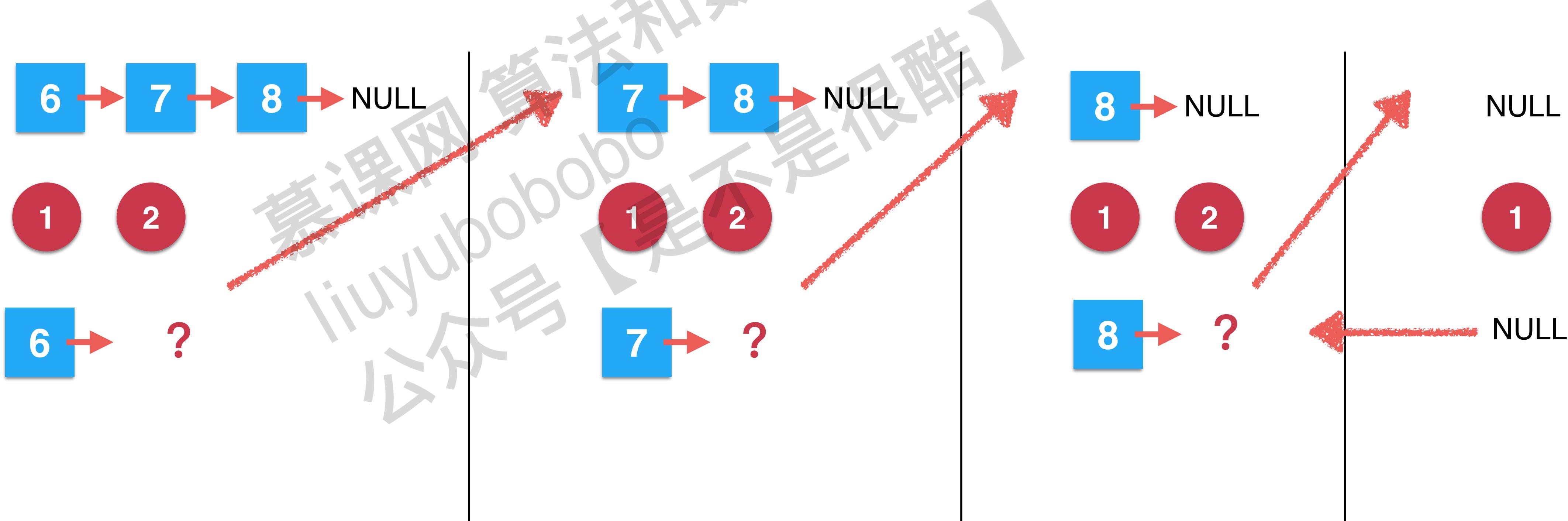
res = 10

调用sum(arr, 2)

```
int sum(int[] arr, int l){  
    if(l == n) return 0;  
    int x = sum(arr, l + 1);  
    int res = arr[l] + x;  
    return res;
```

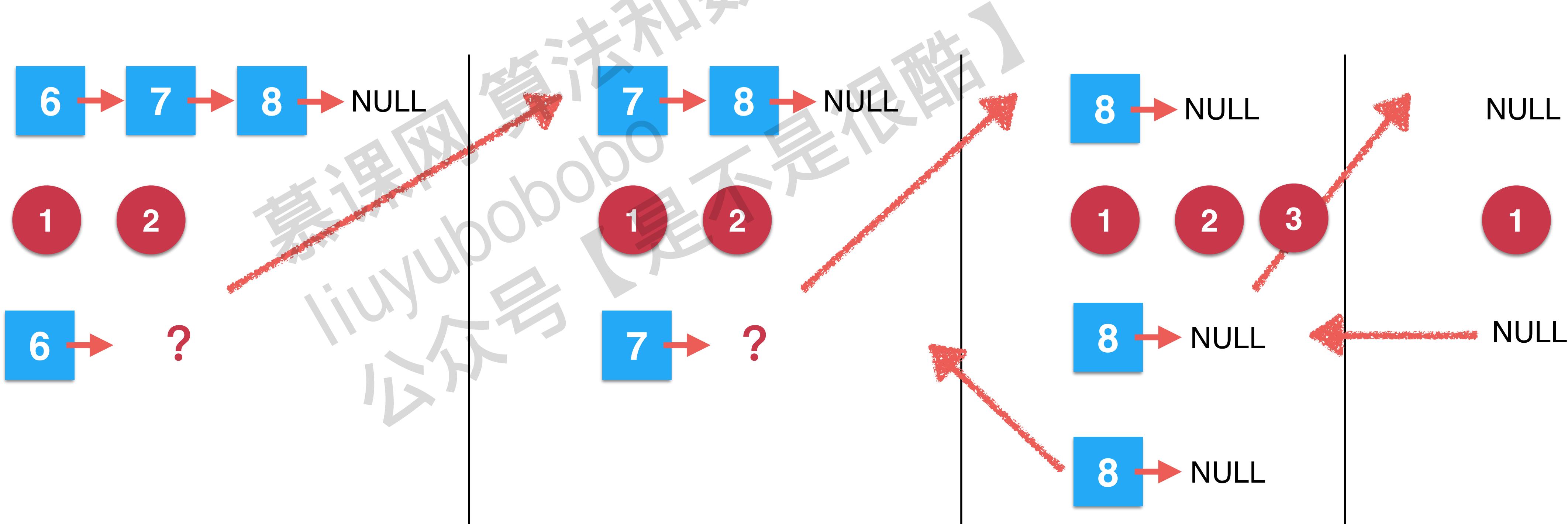
```
public ListNode removeElements(ListNode head, int val) {  
  
    1  if(head == null)  
        return null;  
  
    2  head.next = removeElements(head.next, val);  
    3  return head.val == val ? head.next : head;  
}
```

模拟调用, 对 6->7->8->null 删除7



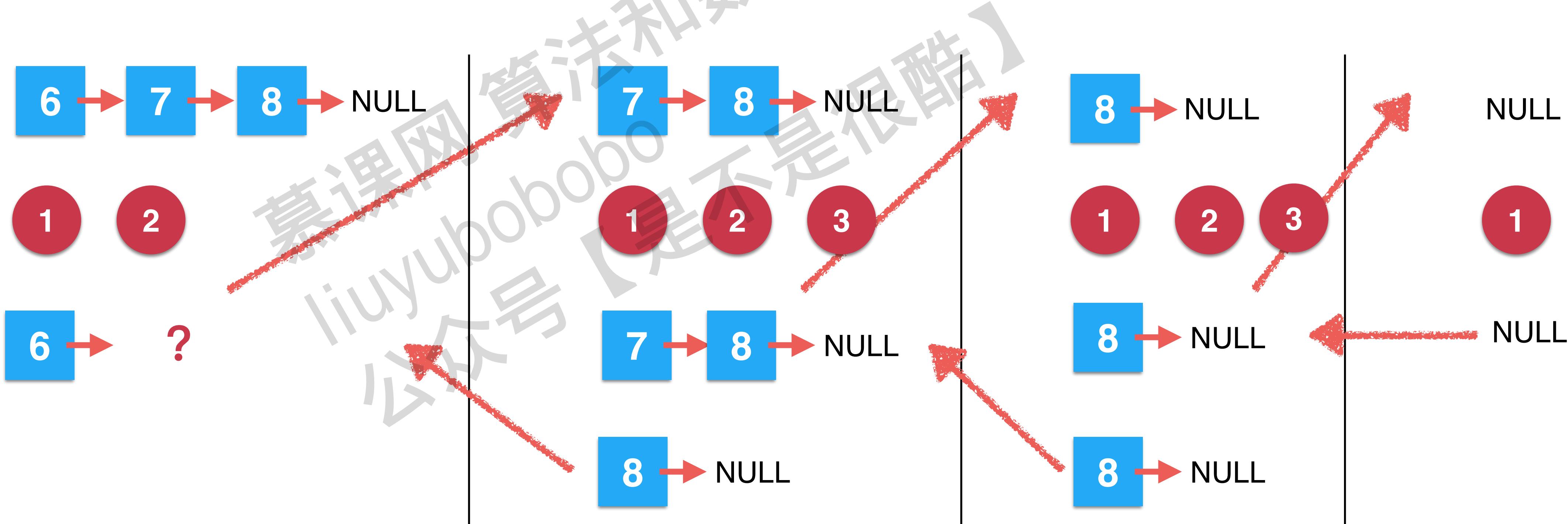
```
public ListNode removeElements(ListNode head, int val) {  
  
    1  if(head == null)  
        return null;  
  
    2  head.next = removeElements(head.next, val);  
    3  return head.val == val ? head.next : head;  
}
```

模拟调用, 对 6->7->8->null 删除7



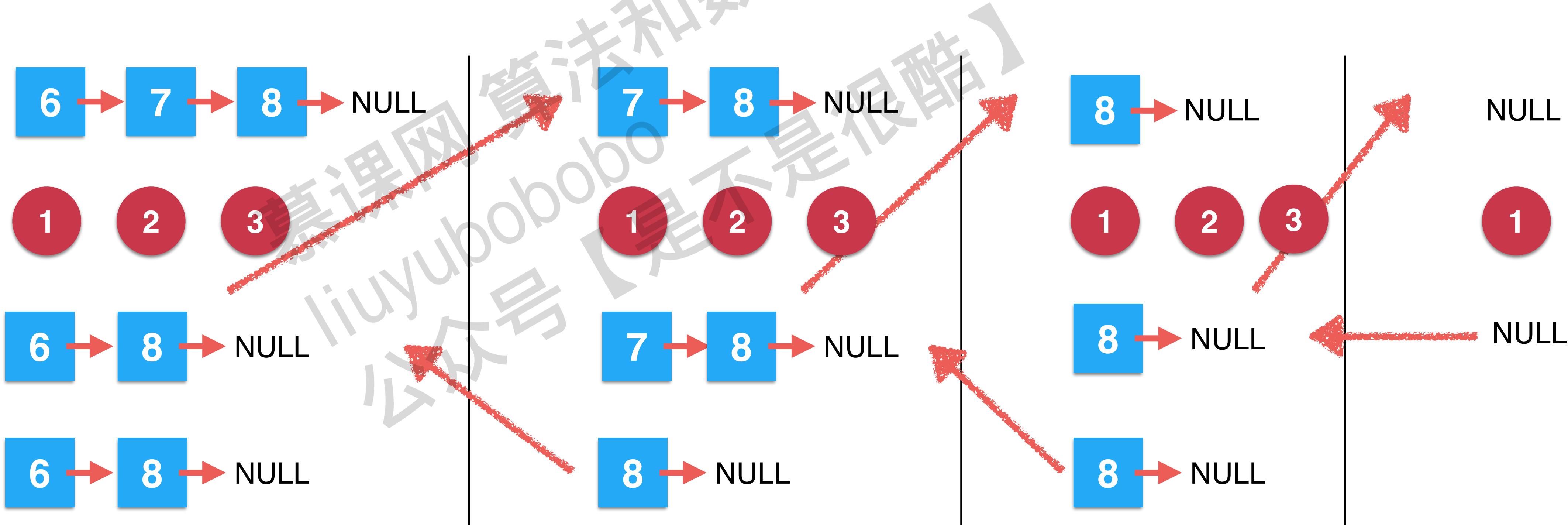
```
public ListNode removeElements(ListNode head, int val) {  
  
    1  if(head == null)  
        return null;  
  
    2  head.next = removeElements(head.next, val);  
    3  return head.val == val ? head.next : head;  
}
```

模拟调用, 对 6->7->8->null 删除7



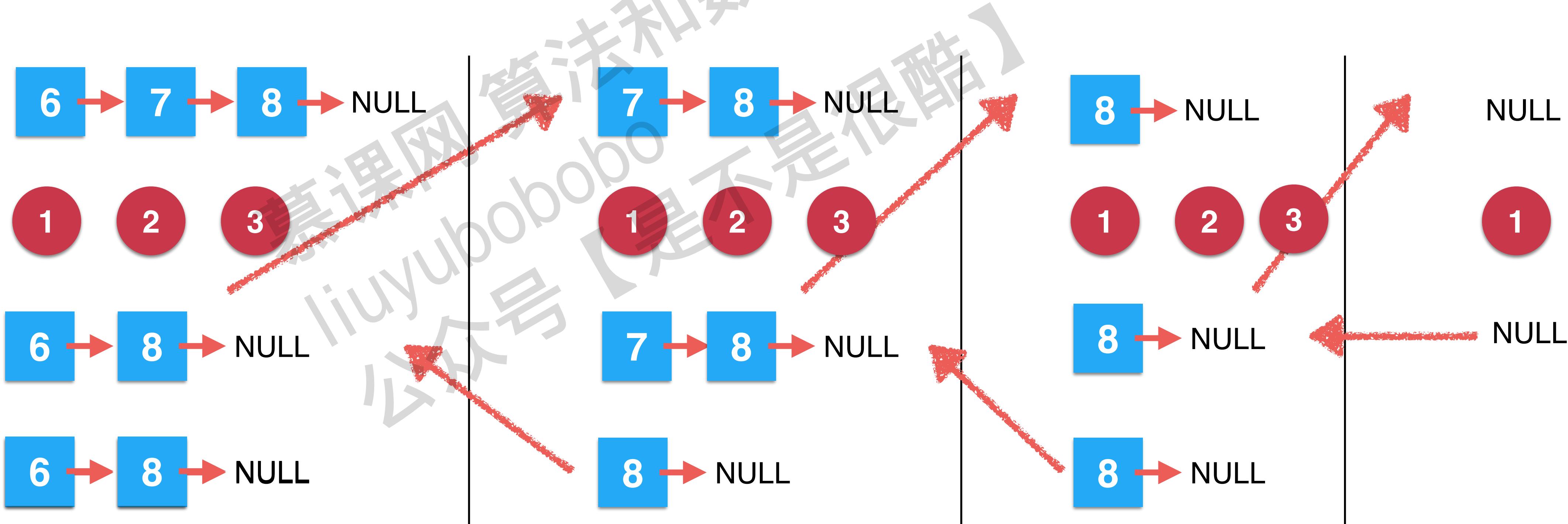
```
public ListNode removeElements(ListNode head, int val) {  
  
    1  if(head == null)  
        return null;  
  
    2  head.next = removeElements(head.next, val);  
    3  return head.val == val ? head.next : head;  
}
```

模拟调用, 对 6->7->8->null 删除7



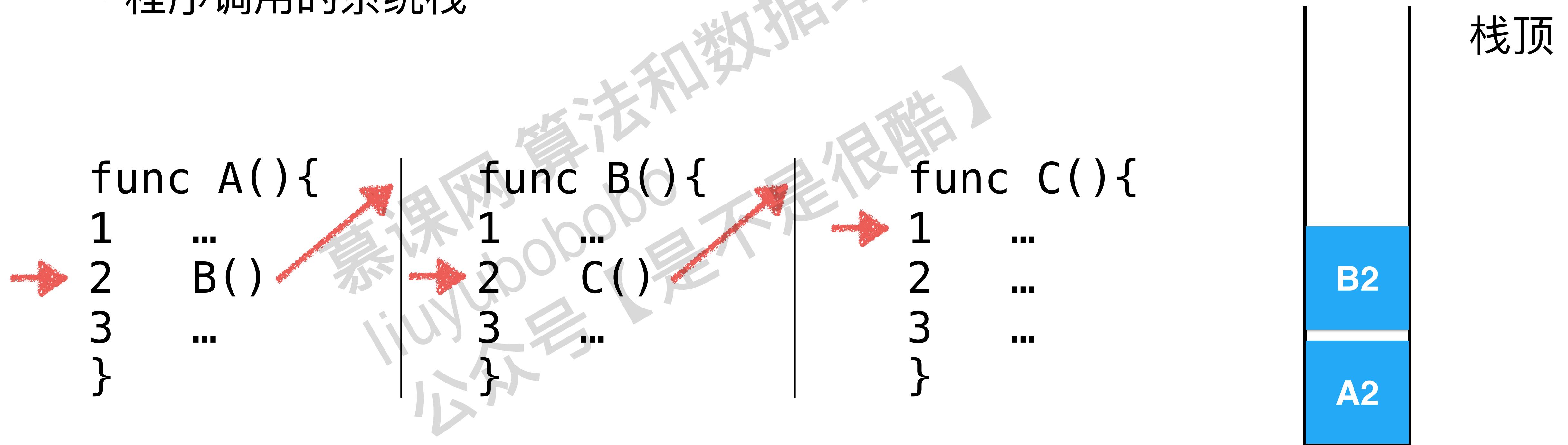
```
public ListNode removeElements(ListNode head, int val) {  
  
    1  if(head == null)  
        return null;  
  
    2  head.next = removeElements(head.next, val);  
    3  return head.val == val ? head.next : head;  
}
```

模拟调用, 对 6->7->8->null 删除7



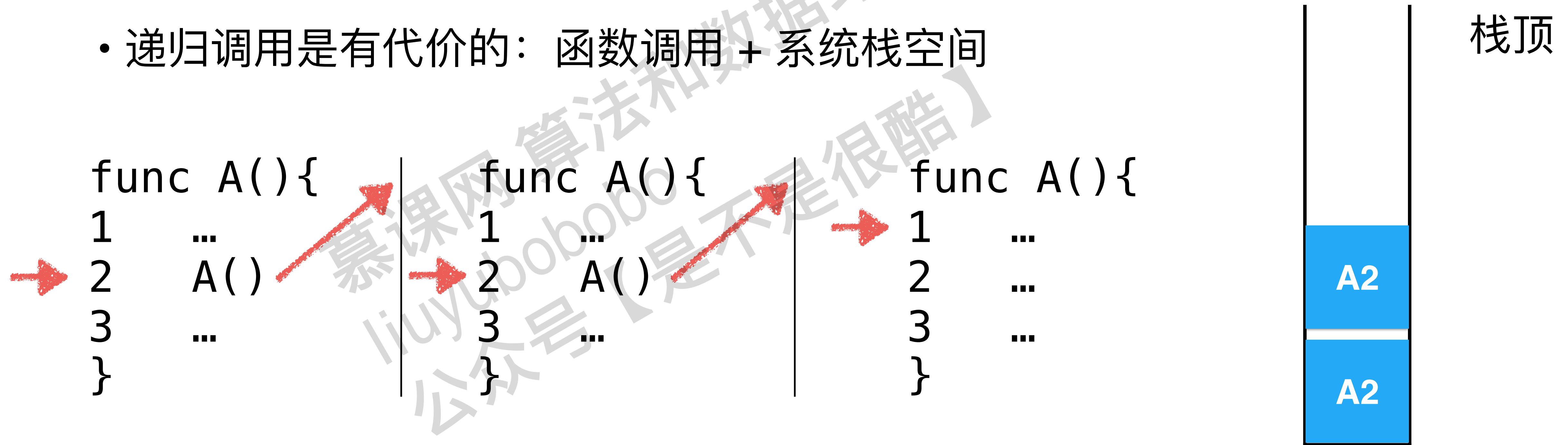
栈的应用

- 程序调用的系统栈



栈的应用

- 程序调用的系统栈
- 递归调用是有代价的：函数调用 + 系统栈空间



调试递归程序

慕课网 算法与数据结构体系课程
liuyubobobo【是不是很酷】
公众号

实践：调试递归程序

慕课网 算法和数据结构体系课程
liuyubobobo（是不是很酷）
公众号

作业：链表的递归实现

慕课网 算法与数据结构体系课程
liuyubobobo 是不是很酷
公众号

作业：链表的递归实现

- 关于递归
- 近乎和链表相关的所有操作，都可以使用递归的形式完成
- 建议同学们对链表的增，删，改，查，进行递归实现

作业解析：链表的递归实现

慕课网 算法和数据结构体系课程
liuyubobobo 是不是很酷
公众号

链表添加元素递归实现的一个常见问题

慕课网 liuyubobobo 是不是很酷

链表添加元素递归实现的一个常见问题

递归函数的“宏观”语意：

在以 head 为头节点的链表中删除值为 val 的节点，并返回结果链表的头结点

```
public ListNode removeElements(ListNode head, int val) {  
    if(head == null)  
        return null;  
  
    head.next = removeElements(head.next, val);  
    return head.val == val ? head.next : head;  
}
```

链表添加元素递归实现的一个常见问题

在以 head 为头节点的链表中删除值为 val 的节点，并返回结果链表的头结点

```
public ListNode removeElements(ListNode head, int val) {  
    if(head == null)  
        return null;  
    head.next = removeElements(head.next, val);  
    return head.val == val ? head.next : head;  
}
```

可不可以不返回头节点？



不可以，否则递归函数的结果不能和前面的链表连接起来

链表添加元素递归实现的一个常见问题

```
public void add(int index, E e){  
    head = add(head, index, e);  
    size ++;  
}
```

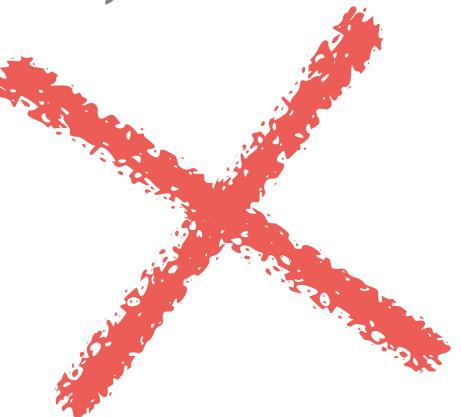
//在以node为头结点的链表的index位置插入元素e，递归算法

```
private Node add(Node node, int index, E e){  
  
    if(index == 0)  
        return new Node(e, node);  
  
    node.next = add(node.next, index - 1, e);  
    return node;  
}
```

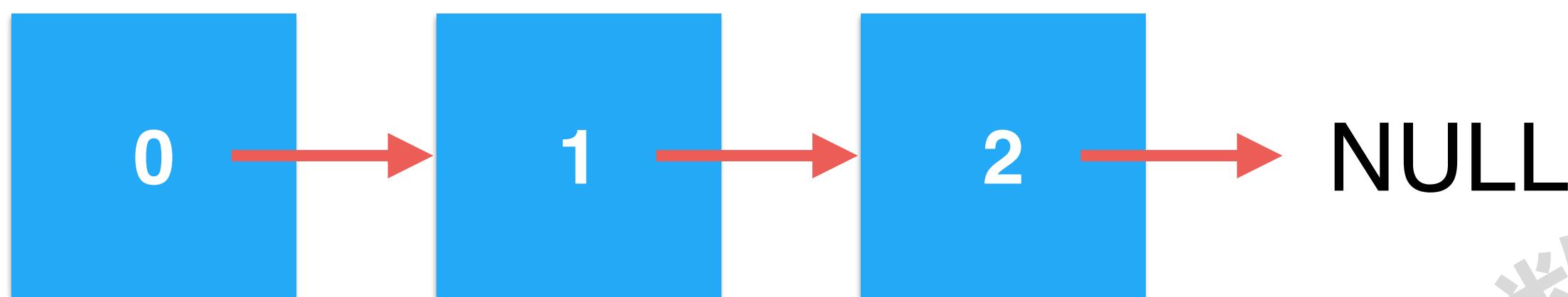
```
public void add(int index, E e){  
    add(head, index, e);  
    size ++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){  
  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题



↑
node

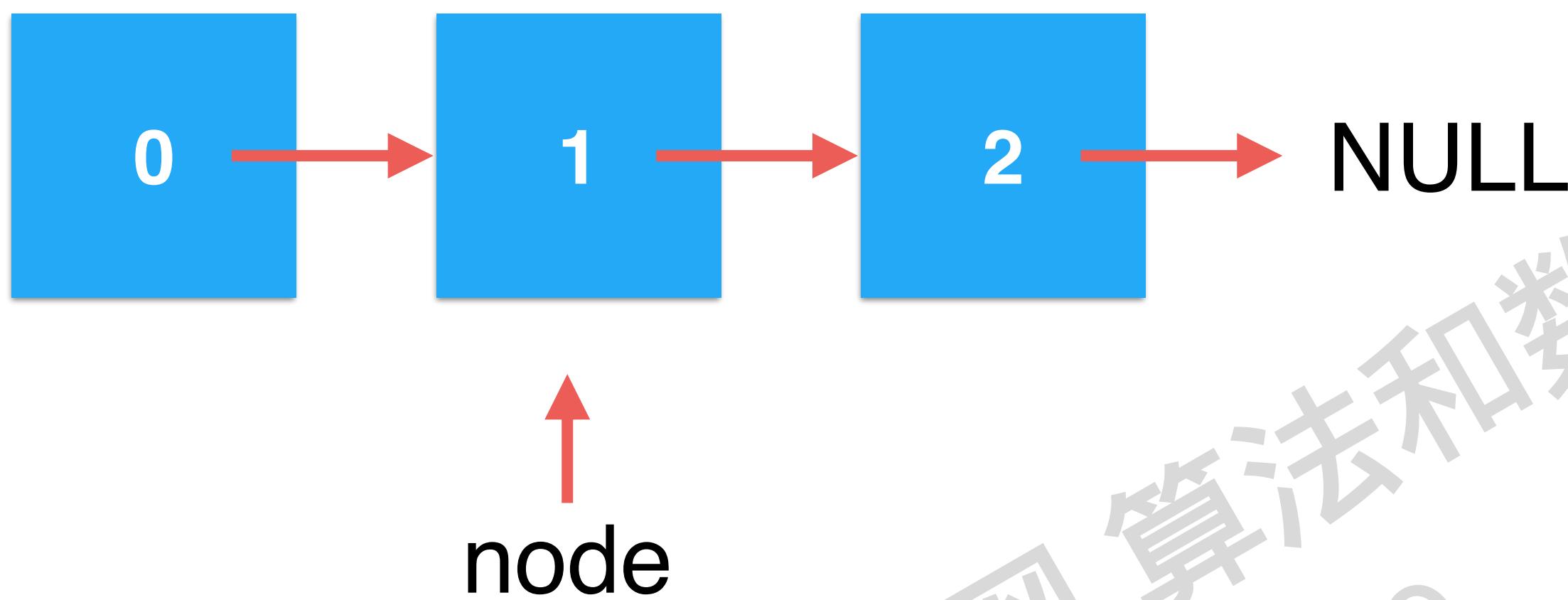
```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){  
  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题



```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

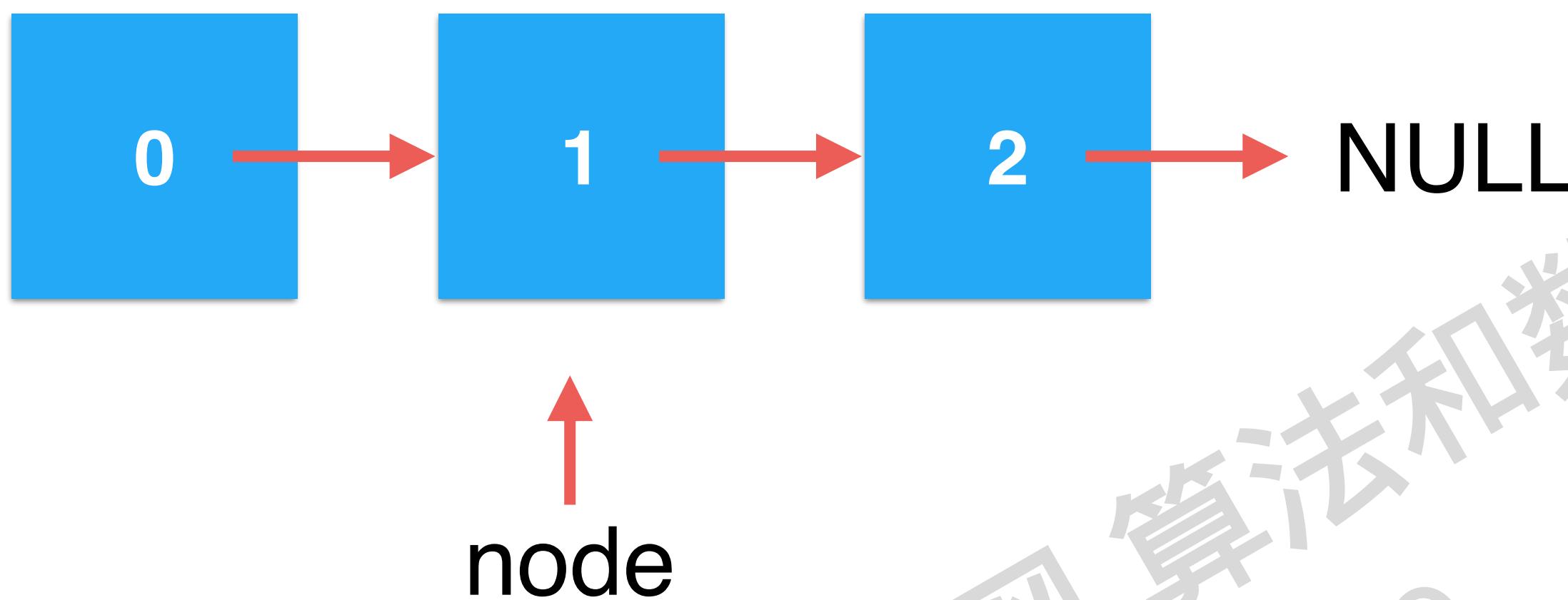
//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){
```

```
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题



```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

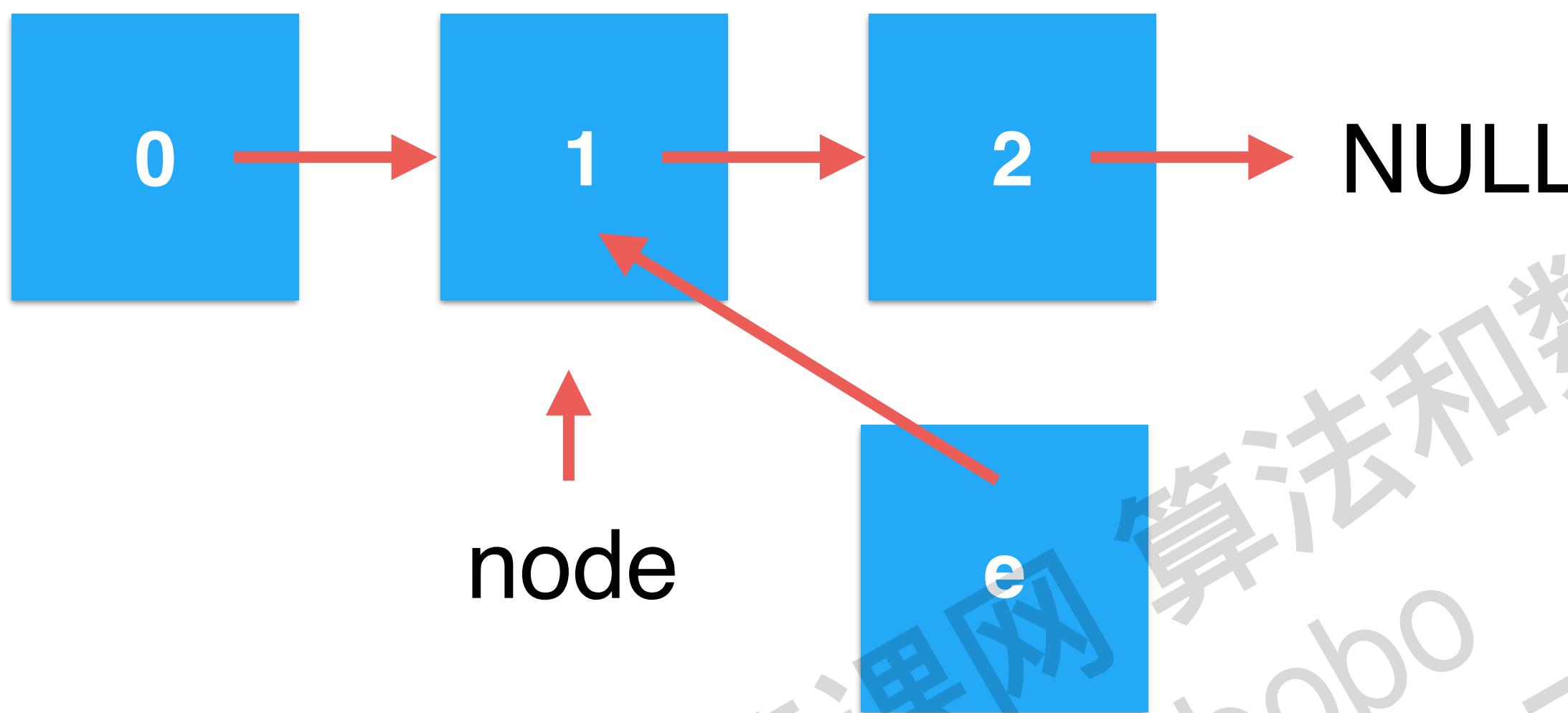
//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){
```

```
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题



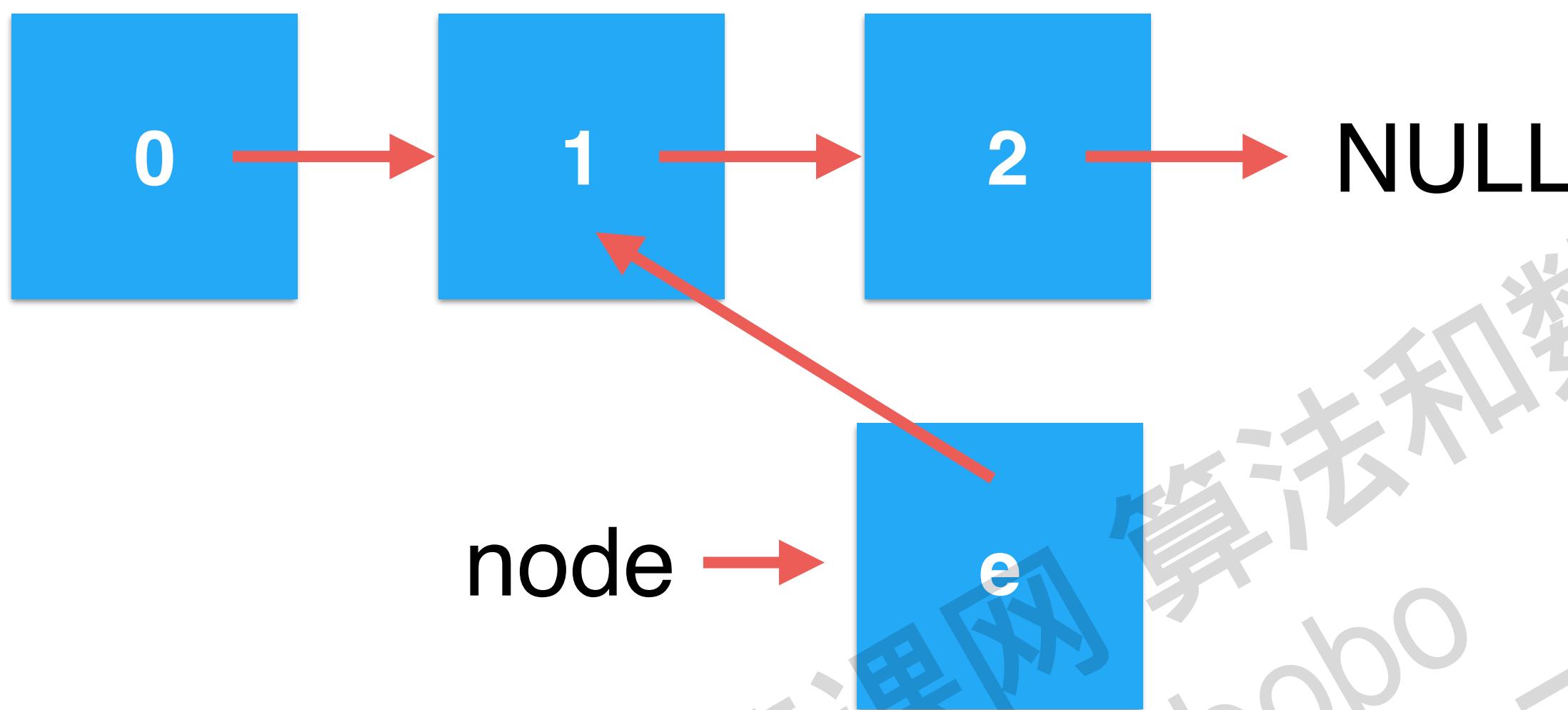
```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
    add(node.next, index - 1, e);  
}
```



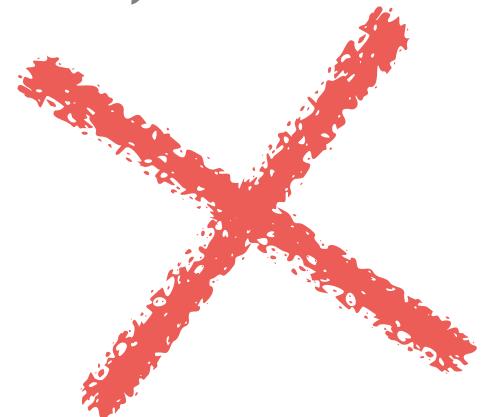
链表添加元素递归实现的一个常见问题



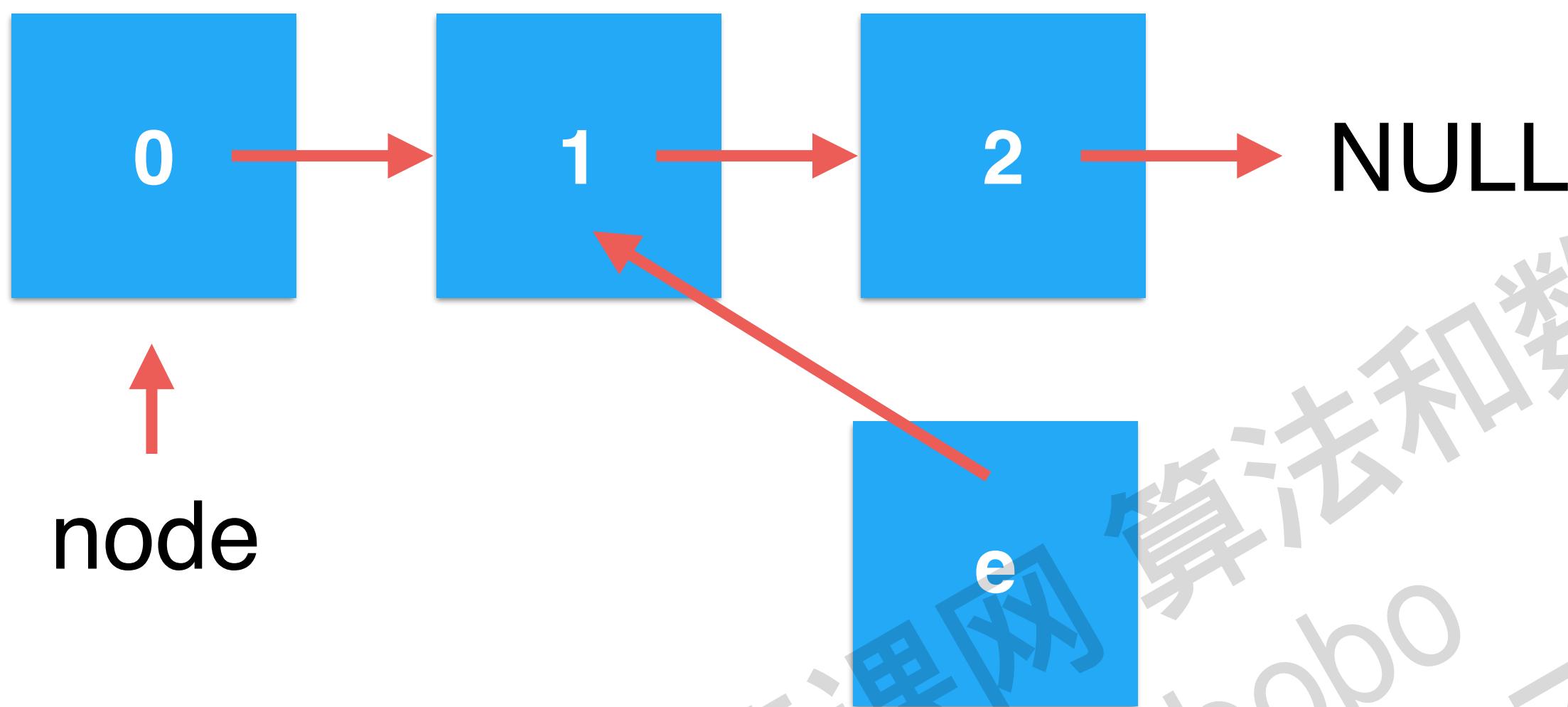
```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
    add(node.next, index - 1, e);  
}
```



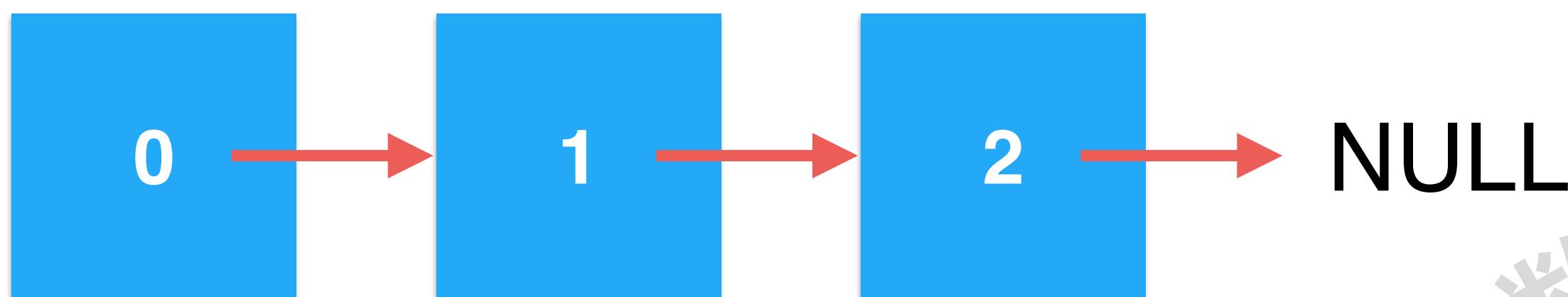
链表添加元素递归实现的一个常见问题



```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}  
  
//在以node为头结点的链表的index位置插入元素e，递归算法  
private void add(Node node, int index, E e){  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题

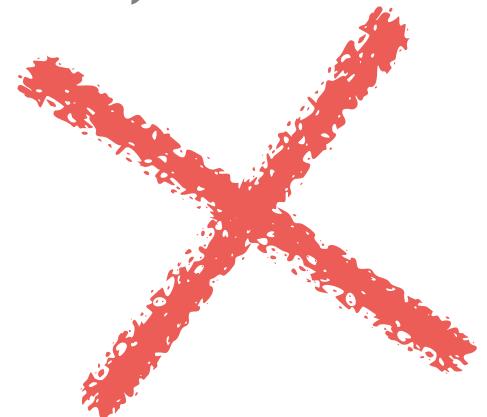


↑
node

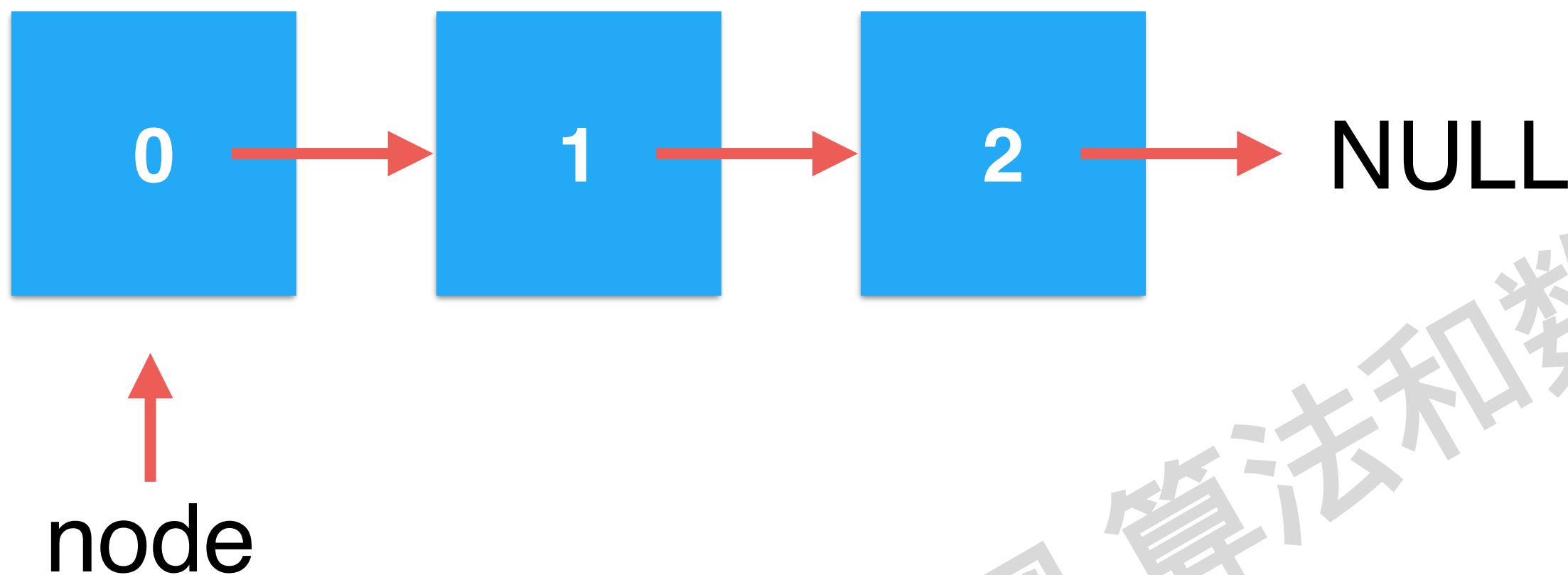
```
public void add(int index, E e){  
    add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private void add(Node node, int index, E e){  
  
    if(index == 0){  
        node = new Node(e, node);  
        return;  
    }  
  
    add(node.next, index - 1, e);  
}
```



链表添加元素递归实现的一个常见问题



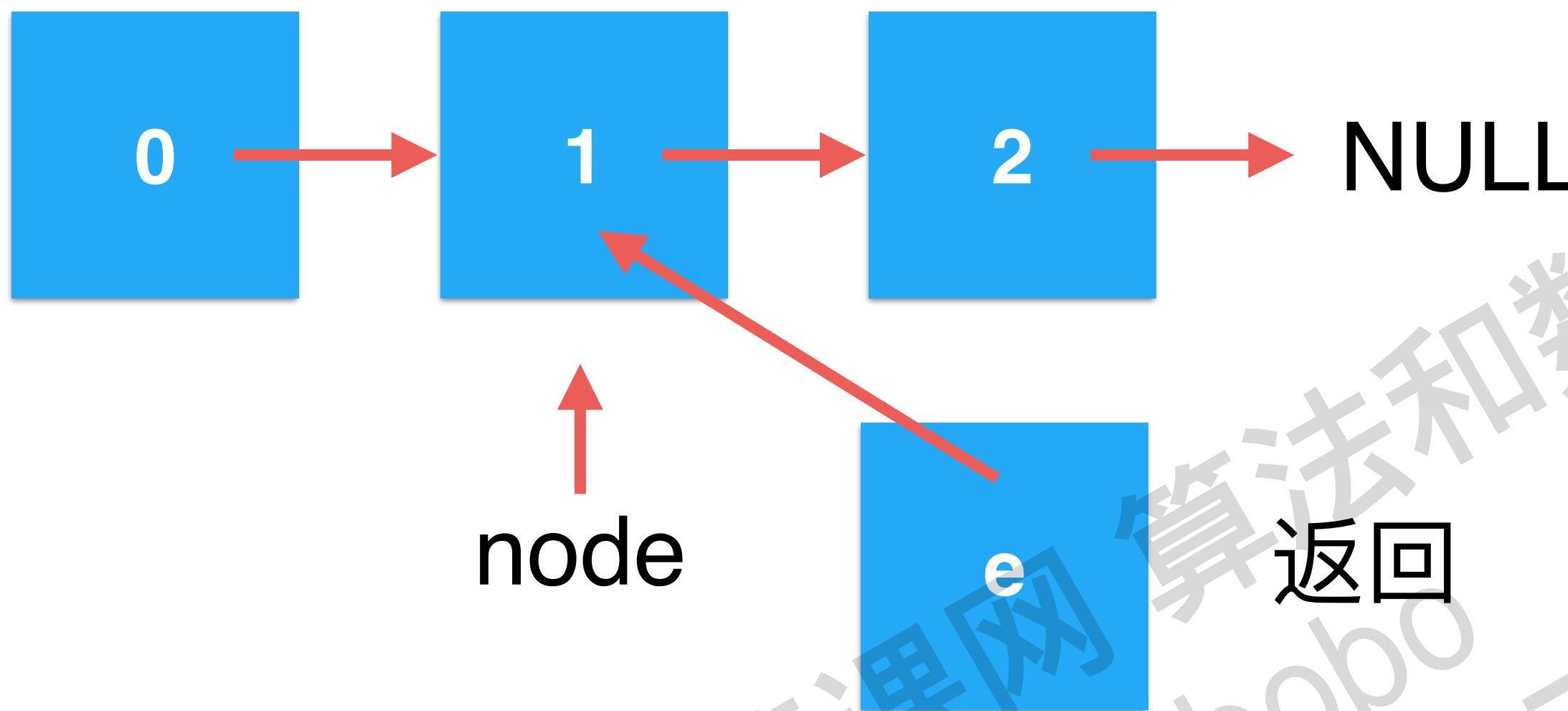
```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法
private Node add(Node node, int index, E e){

```
if(index == 0)  
    return new Node(e, node);
```

```
node.next = add(node.next, index - 1, e);  
return node;
```

链表添加元素递归实现的一个常见问题



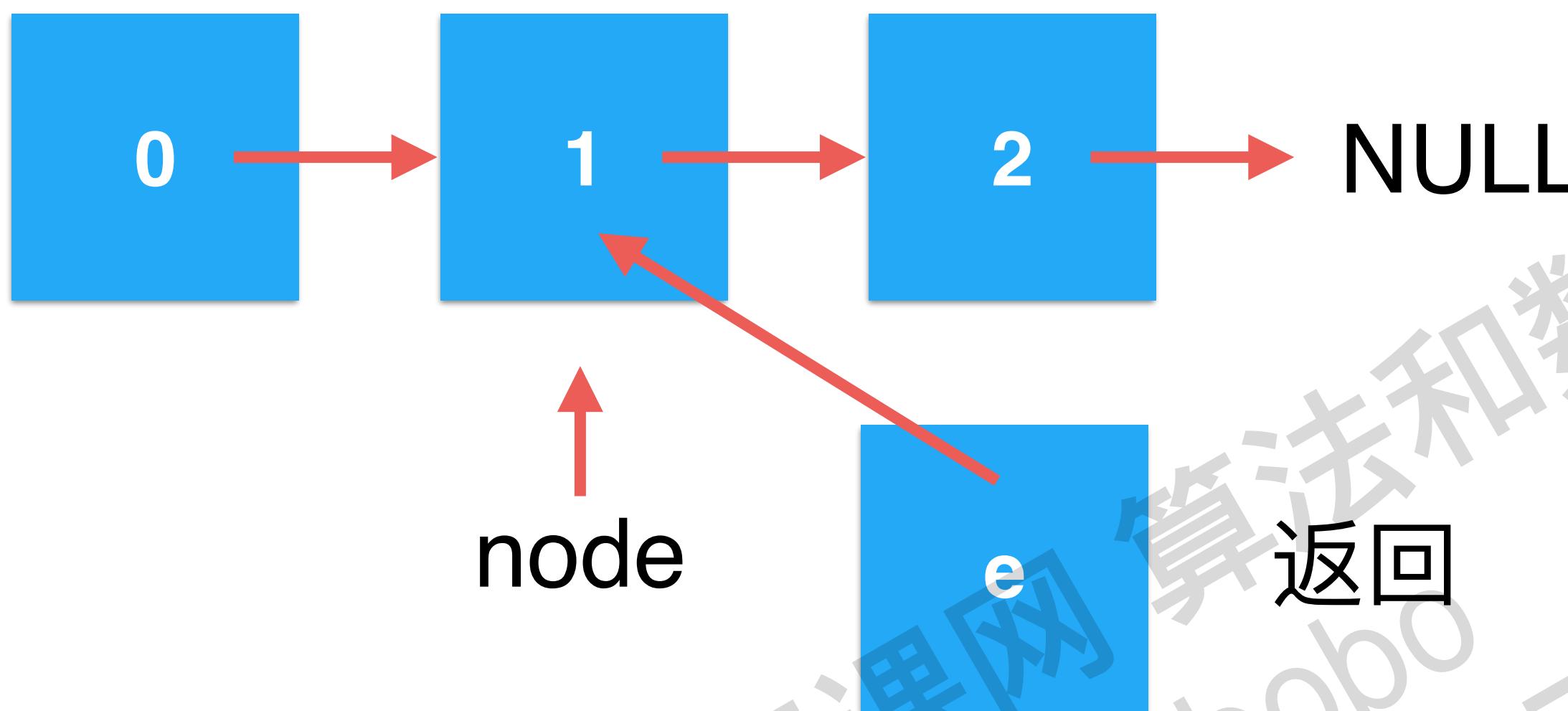
```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法
private Node add(Node node, int index, E e){

```
if(index == 0)  
    return new Node(e, node);
```

```
node.next = add(node.next, index - 1, e);  
return node;
```

链表添加元素递归实现的一个常见问题



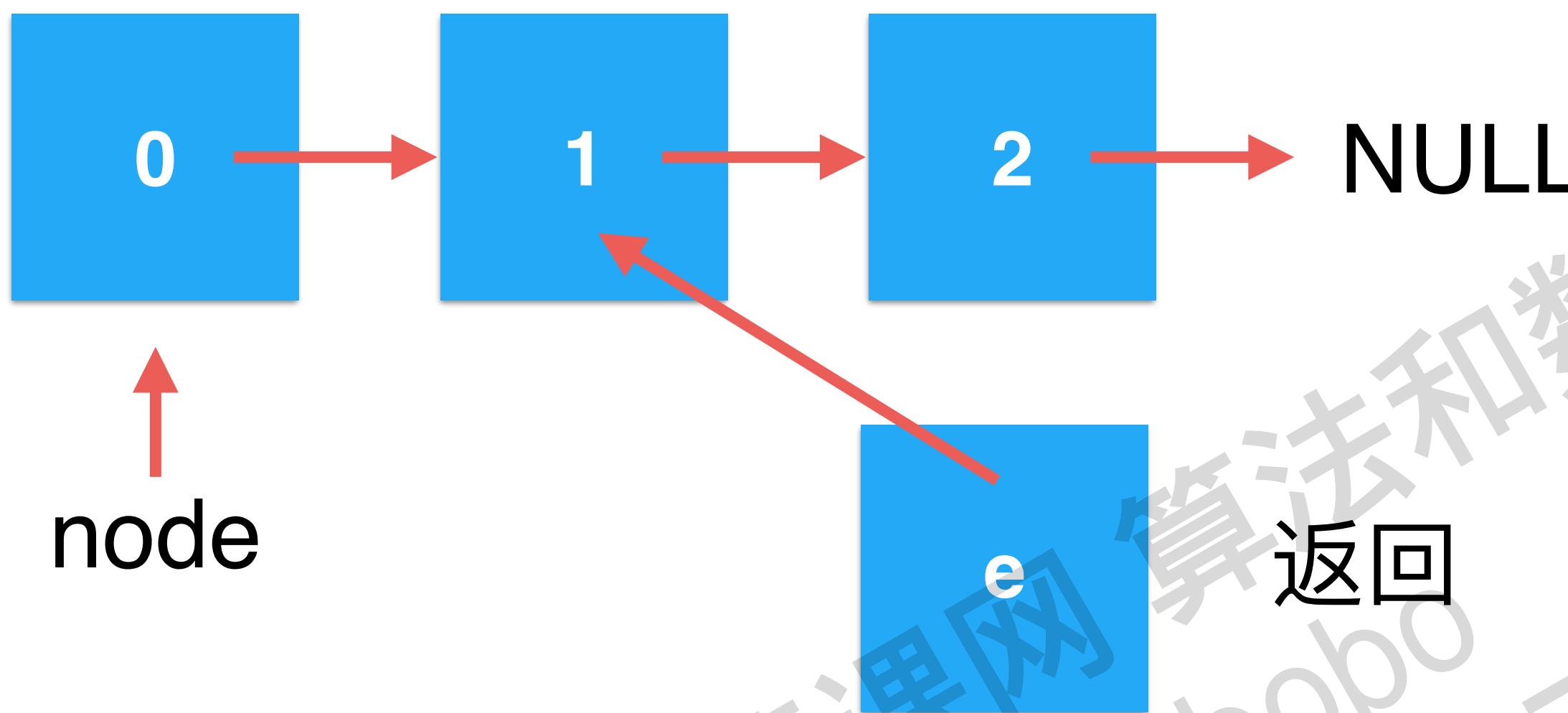
```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法
private Node add(Node node, int index, E e){

```
if(index == 0)  
    return new Node(e, node);
```

```
node.next = add(node.next, index - 1, e);  
return node;
```

链表添加元素递归实现的一个常见问题

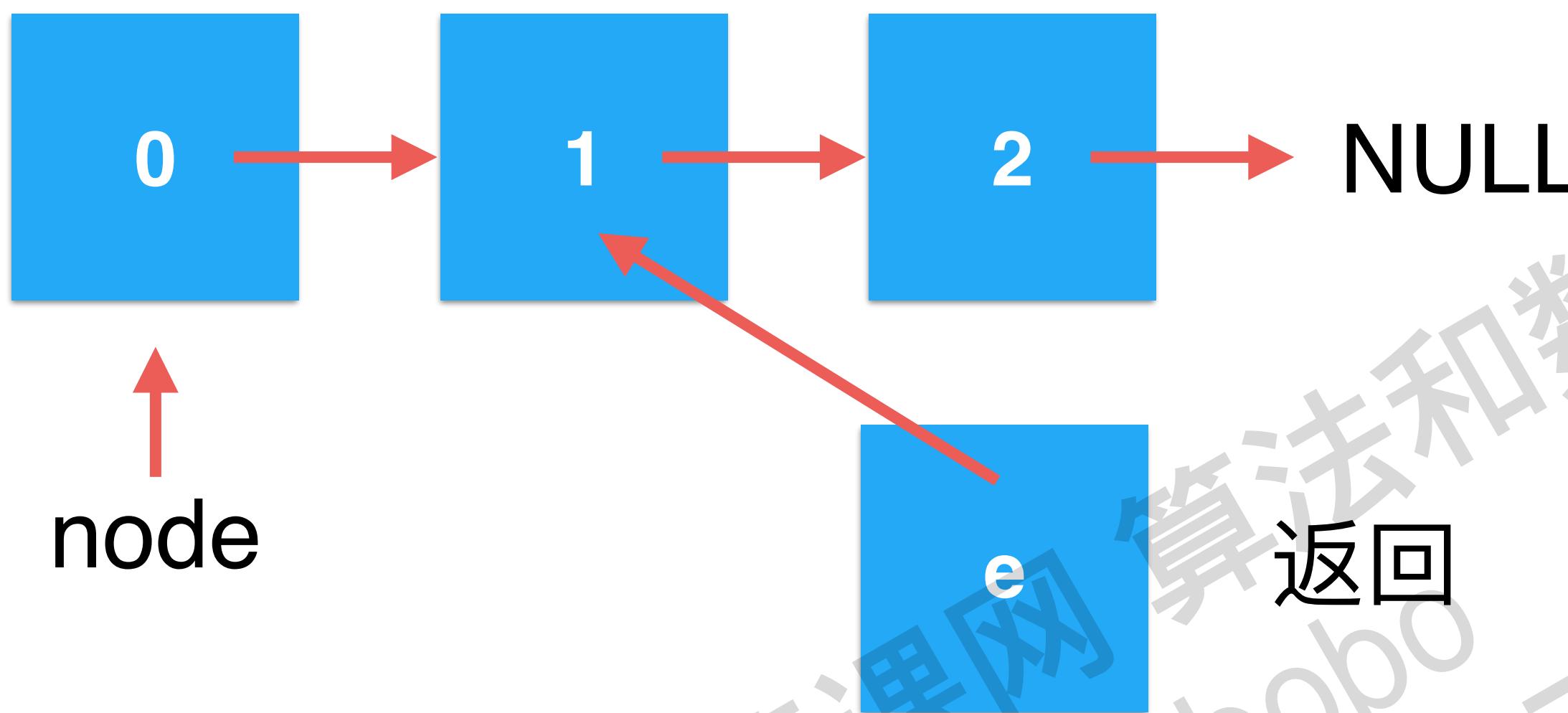


```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private Node add(Node node, int index, E e){  
  
    if(index == 0)  
        return new Node(e, node);  
  
    node.next = add(node.next, index - 1, e);  
    return node;  
}
```

链表添加元素递归实现的一个常见问题

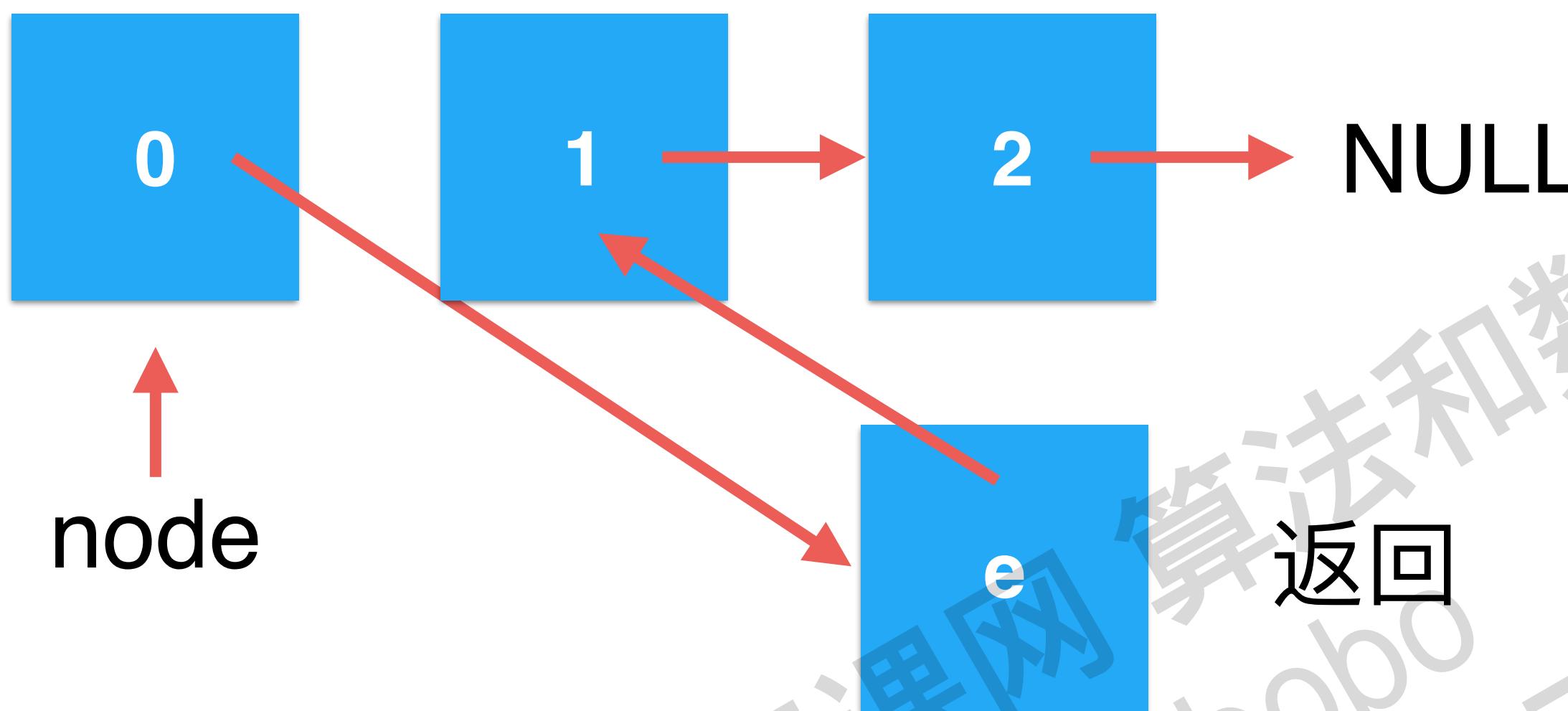


```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private Node add(Node node, int index, E e){  
    if(index == 0)  
        return new Node(e, node);  
  
    node.next = add(node.next, index - 1, e);  
    return node;  
}
```

链表添加元素递归实现的一个常见问题



```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private Node add(Node node, int index, E e){
```

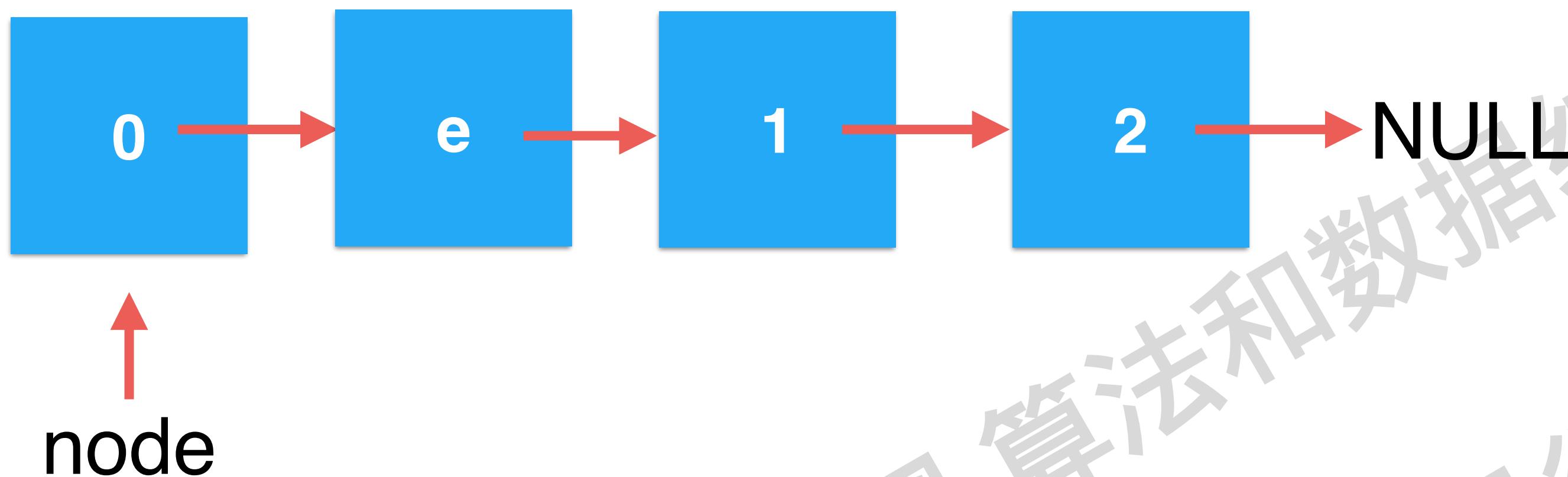
```
if(index == 0)
```

```
    return new Node(e, node);
```

```
node.next = add(node.next, index - 1, e);
```

```
return node;
```

链表添加元素递归实现的一个常见问题



```
public void add(int index, E e){  
    head = add(head, index, e);  
    size++;  
}
```

//在以node为头结点的链表的index位置插入元素e，递归算法

```
private Node add(Node node, int index, E e){  
  
    if(index == 0)  
        return new Node(e, node);  
  
    node.next = add(node.next, index - 1, e);  
    return node;  
}
```

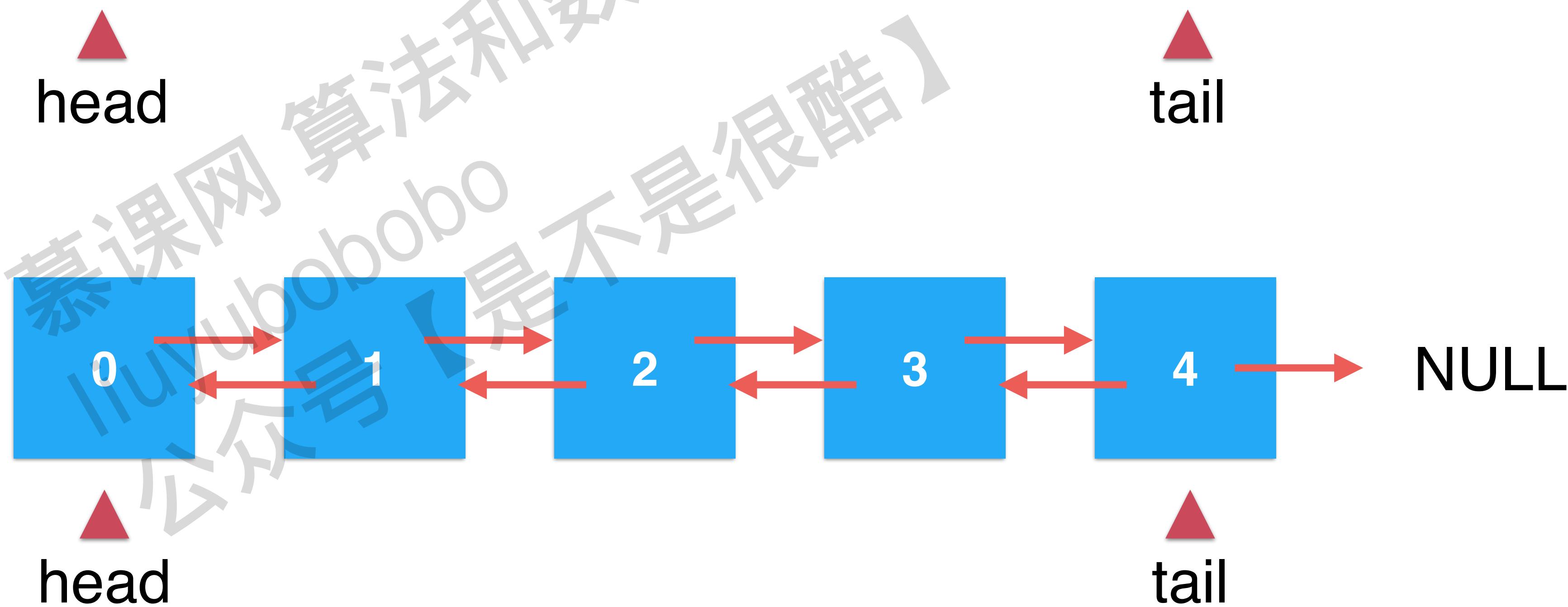
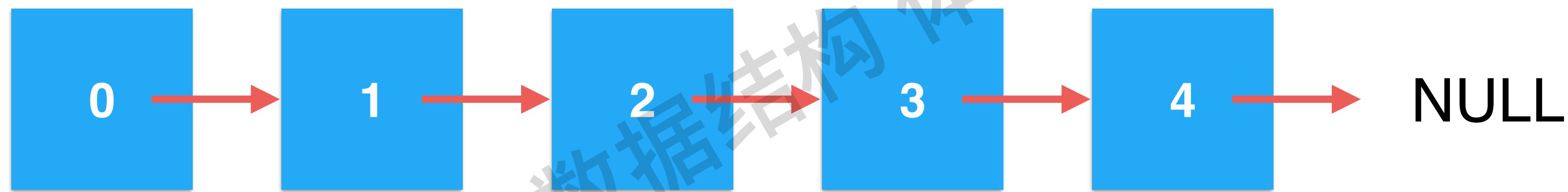
更多和链表相关的话题

慕课网 算法与数据结构体系课程
liuyubobobo 是不是很酷
公众号

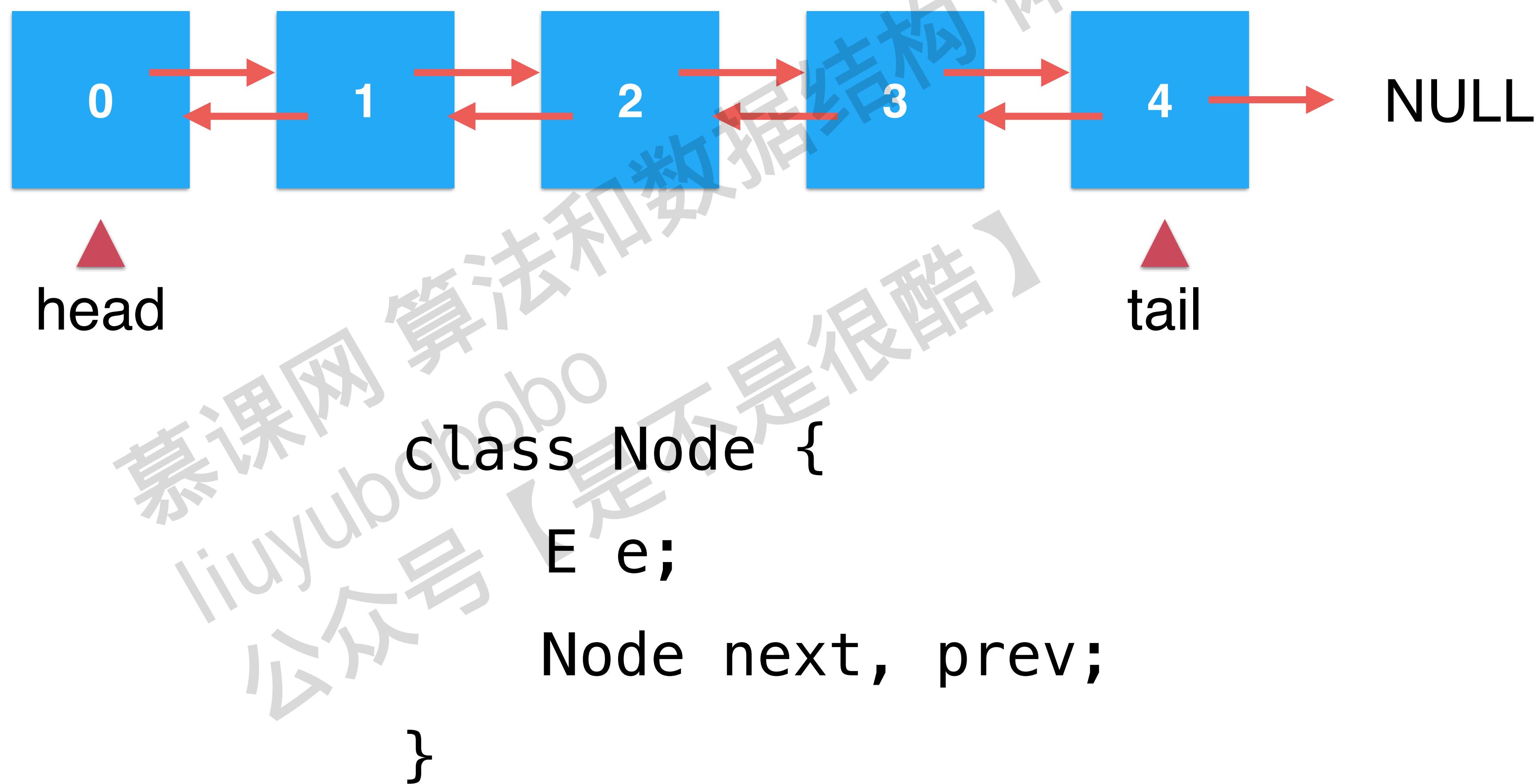
斯坦福文档说明

慕课网 liuyubobobo 云众号【是不是很酷】

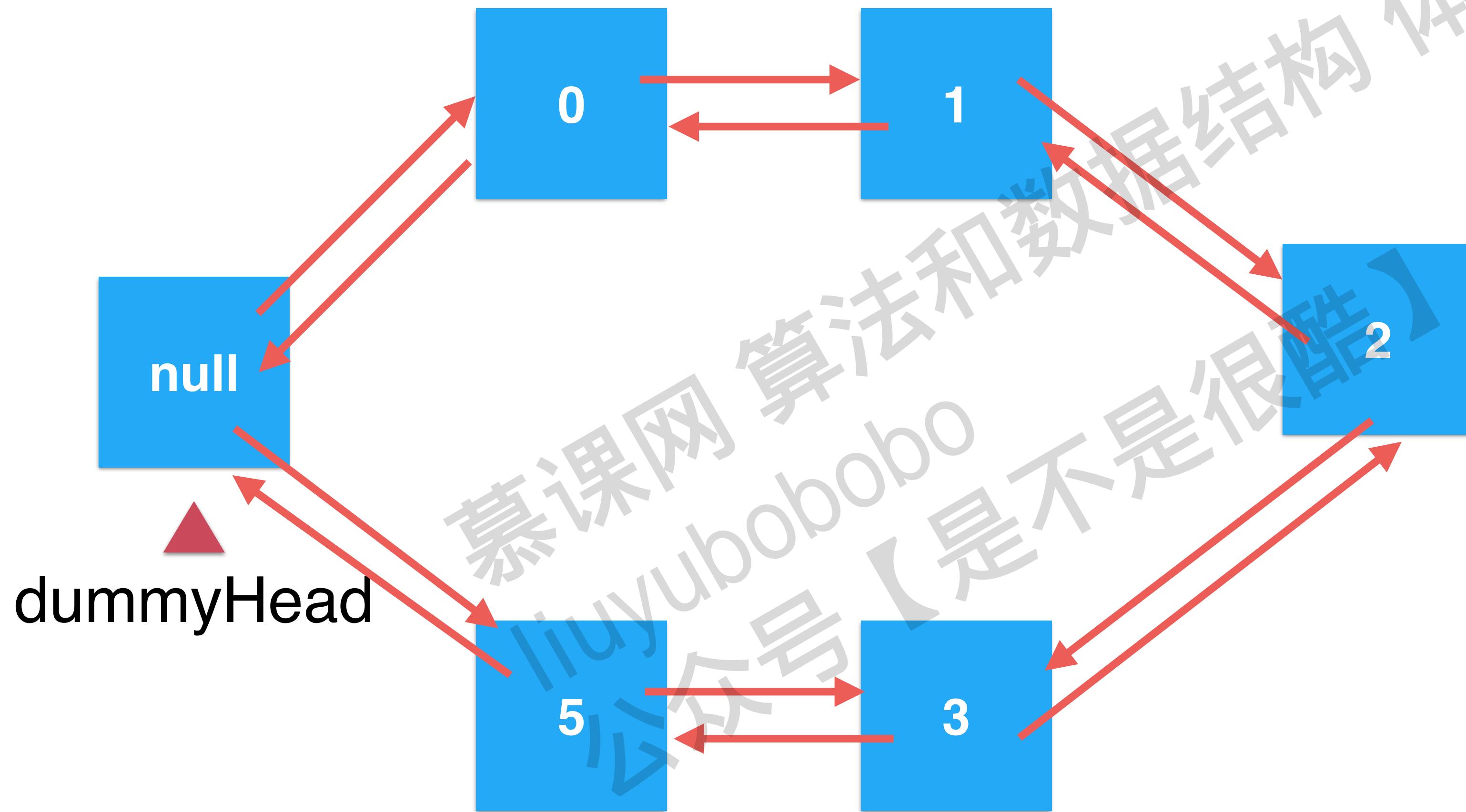
双链表



双链表

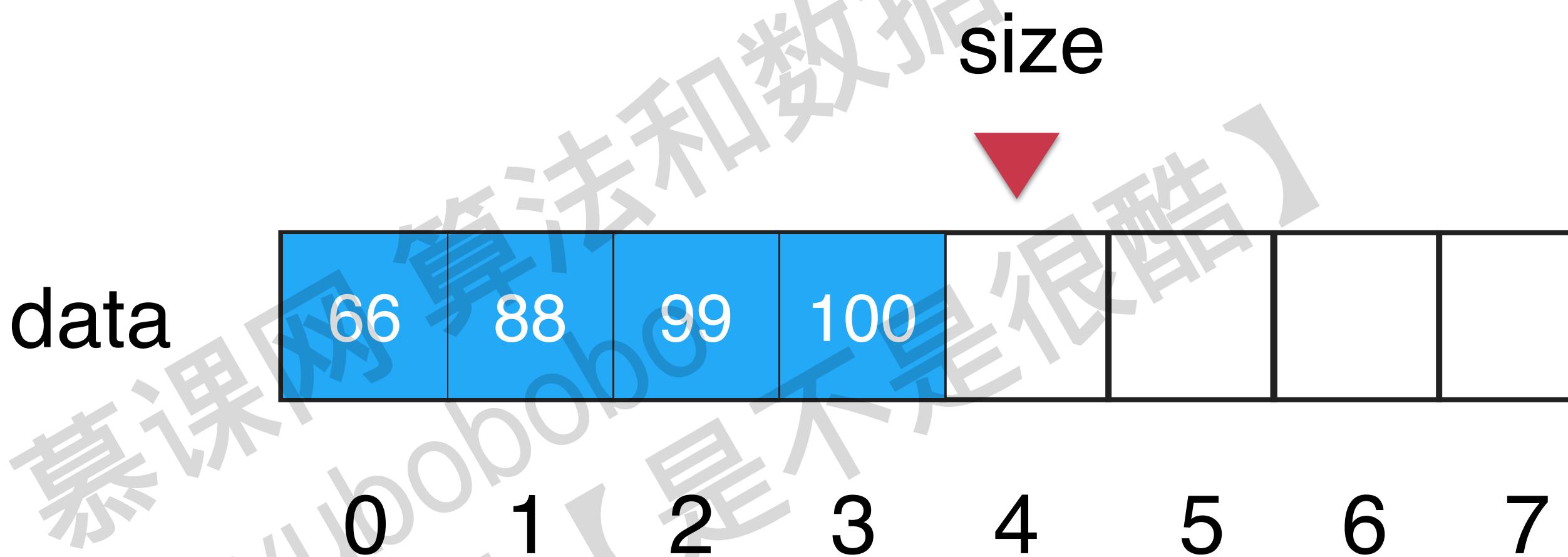


循环链表

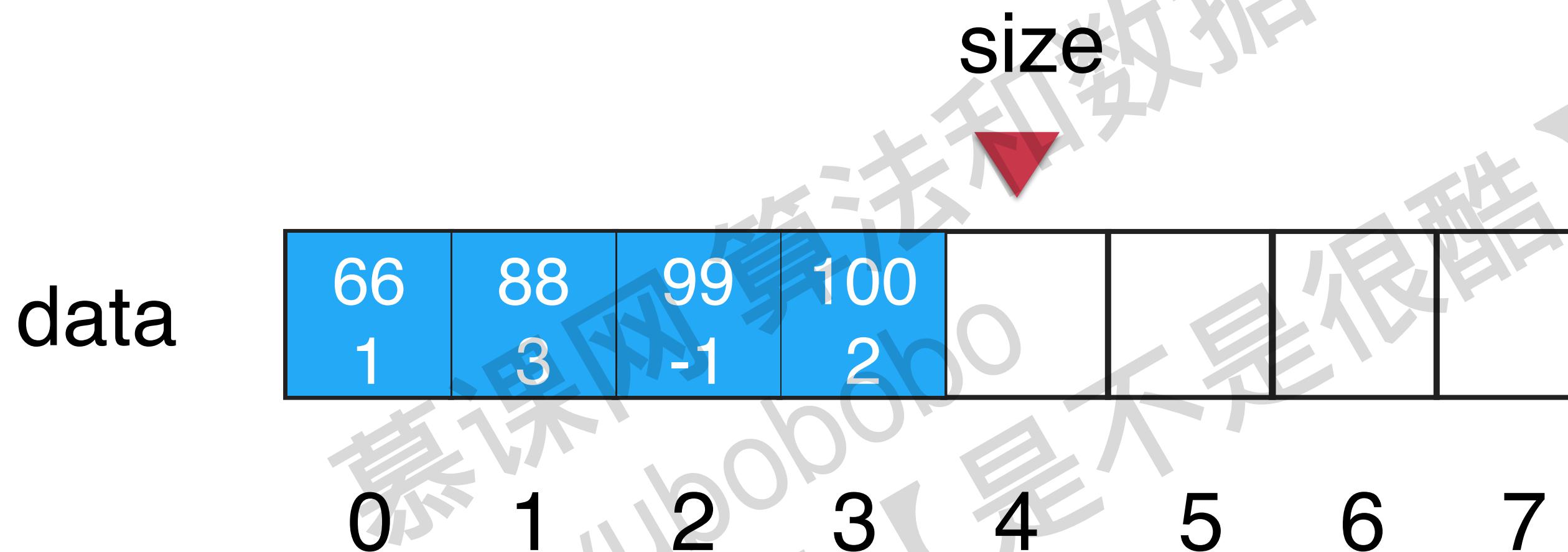


```
class Node {  
    E e;  
    Node next, prev;  
}
```

数组链表



数组链表



```
class Node {  
    E e;  
    int next;  
}
```

链表

慕课网 算法和数据结构体系课程
liuyubobobo
公众号【是不是很酷】

其他

欢迎大家关注我的个人公众号：是不是很酷



算法与数据结构体系课程

liuyubobobo