# 玩儿转数据结构

liuyubobobo

# 链表

# 线性数据结构

- 动态数组
- 栈
- 队列

底层依托静态数组；

靠resize解决固定容量问题

- 链表

真正的动态数据结构

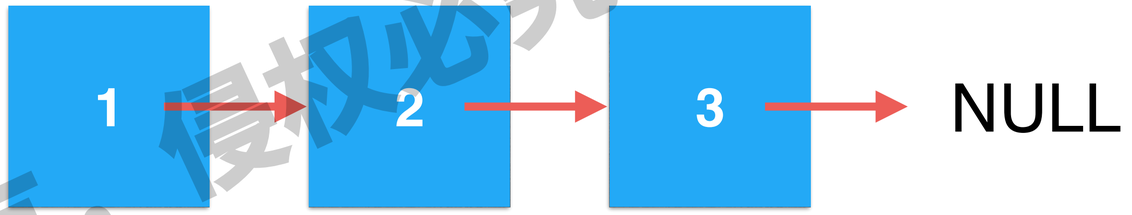
# 为什么链表很重要

- 链表　　真正的动态数据结构
- 最简单的动态数据结构
- 更深入的理解引用（或者指针）
- 更深入的理解递归
- 辅助组成其他数据结构

# 链表 Linked List

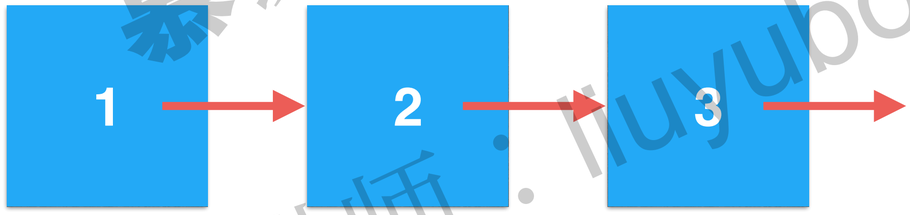
- 数据存储在“节点”（Node）中

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



# 链表 Linked List

- 数据存储在“节点”（Node）中




- 优点：真正的动态，不需要处理固定容量的问题
- 缺点：丧失了随机访问的能力

# 数组和链表的对比

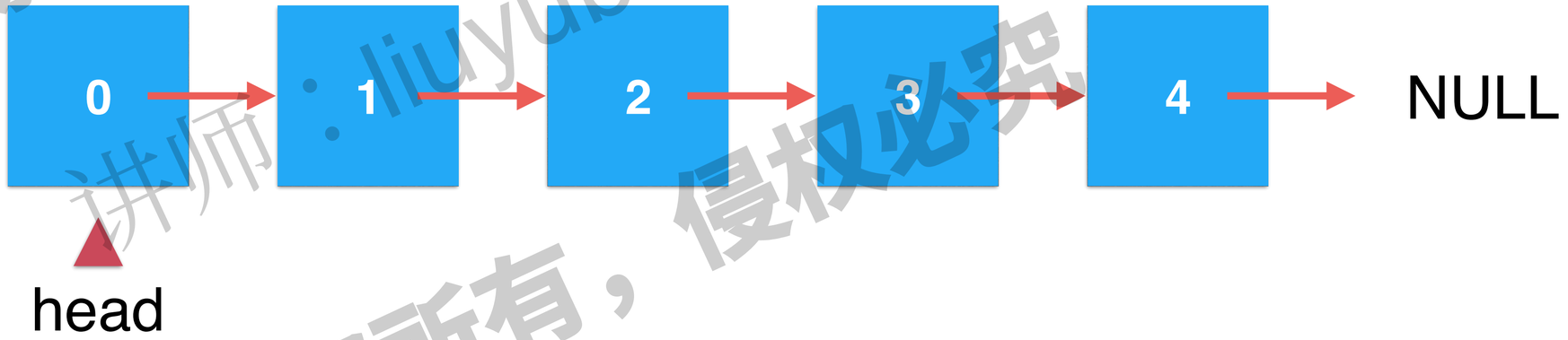
- 数组最好用于索引有语意的情况。scores[2]
- 最大的优点：支持快速查询

- 链表不适合用于索引有语意的情况。
- 最大的优点：动态

# 实践：链表基础

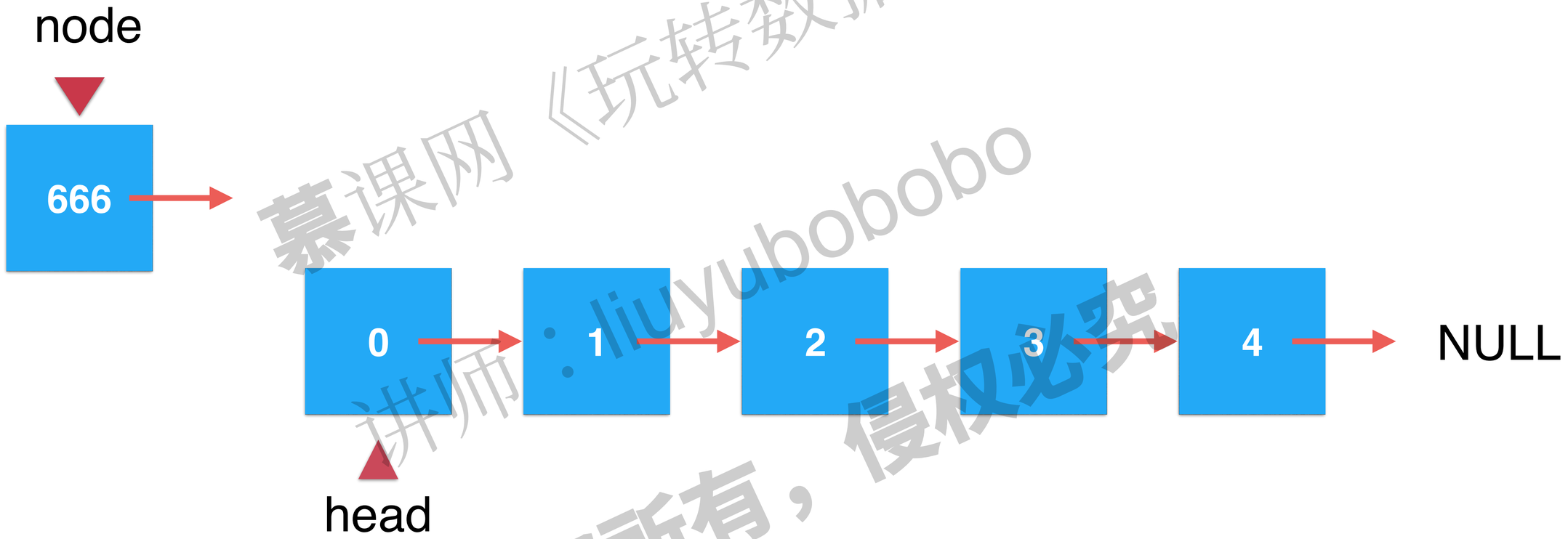
# 向链表中添加元素

# 链表 Linked List



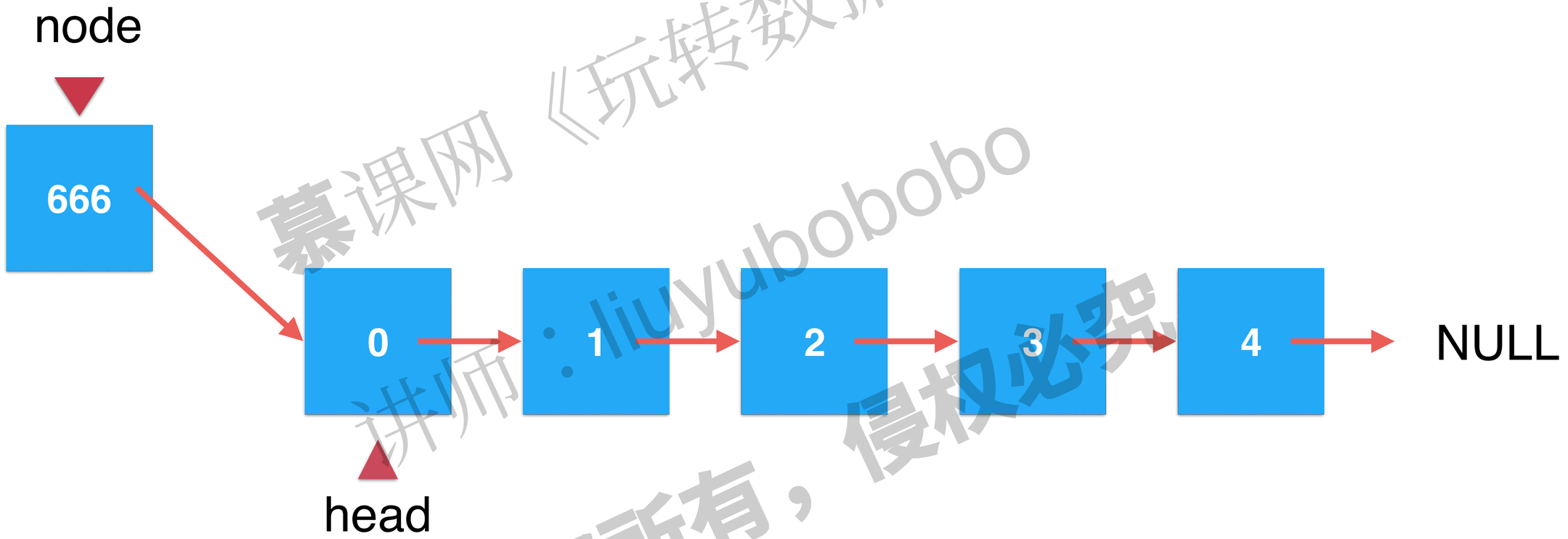
# 实践：链表基础

# 在链表头添加元素




```
node.next = head
```

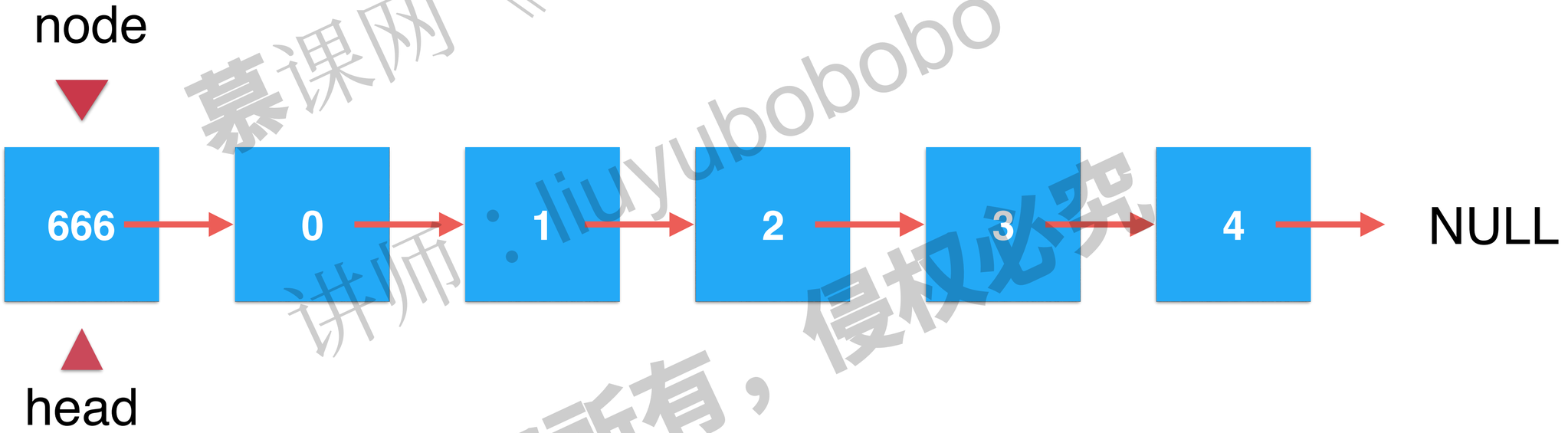
# 在链表头添加元素




```
node.next = head

head = node
```

# 在链表头添加元素



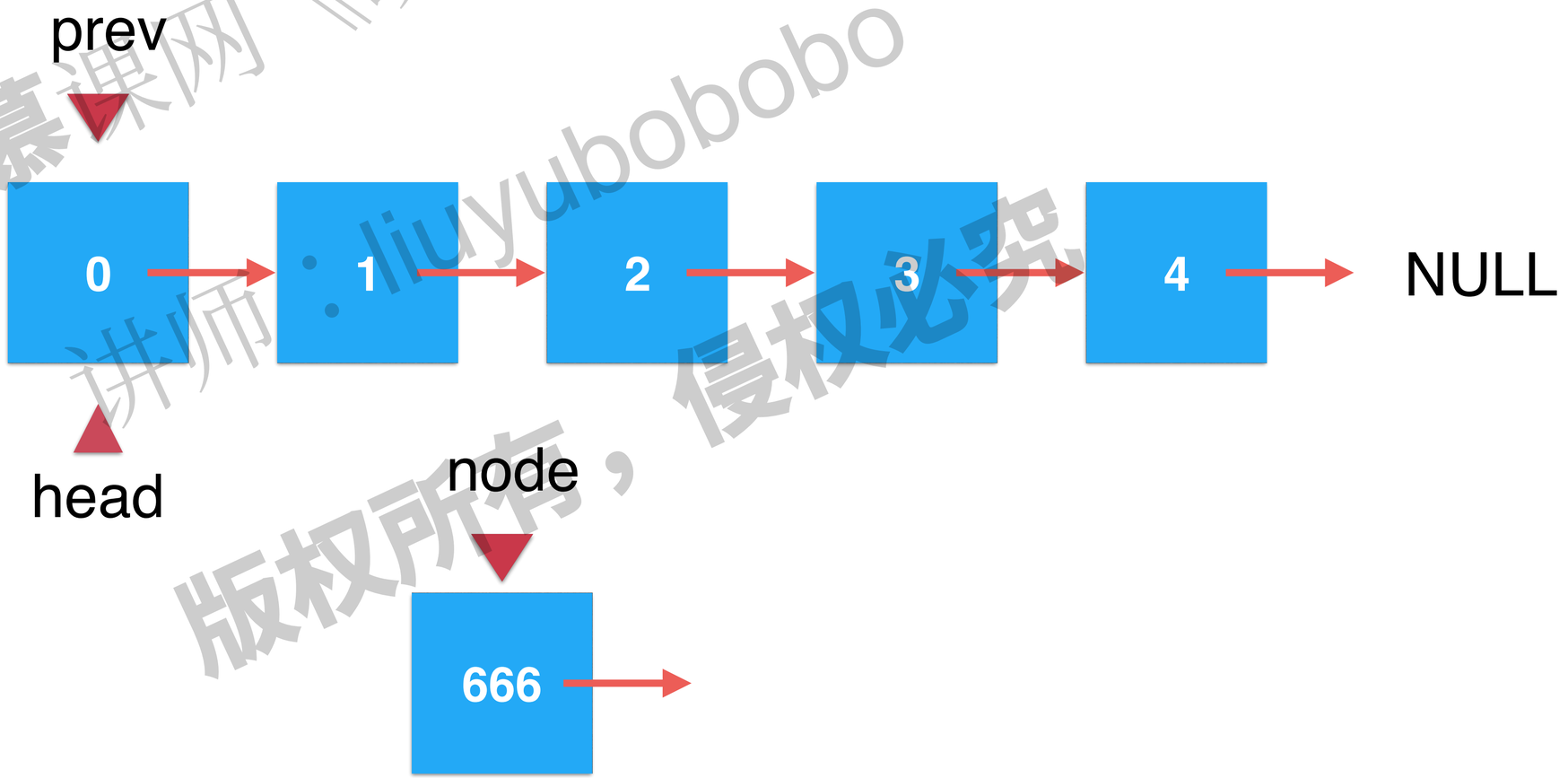

```
node.next = head

head = node
```

# 实践：在链表头添加元素

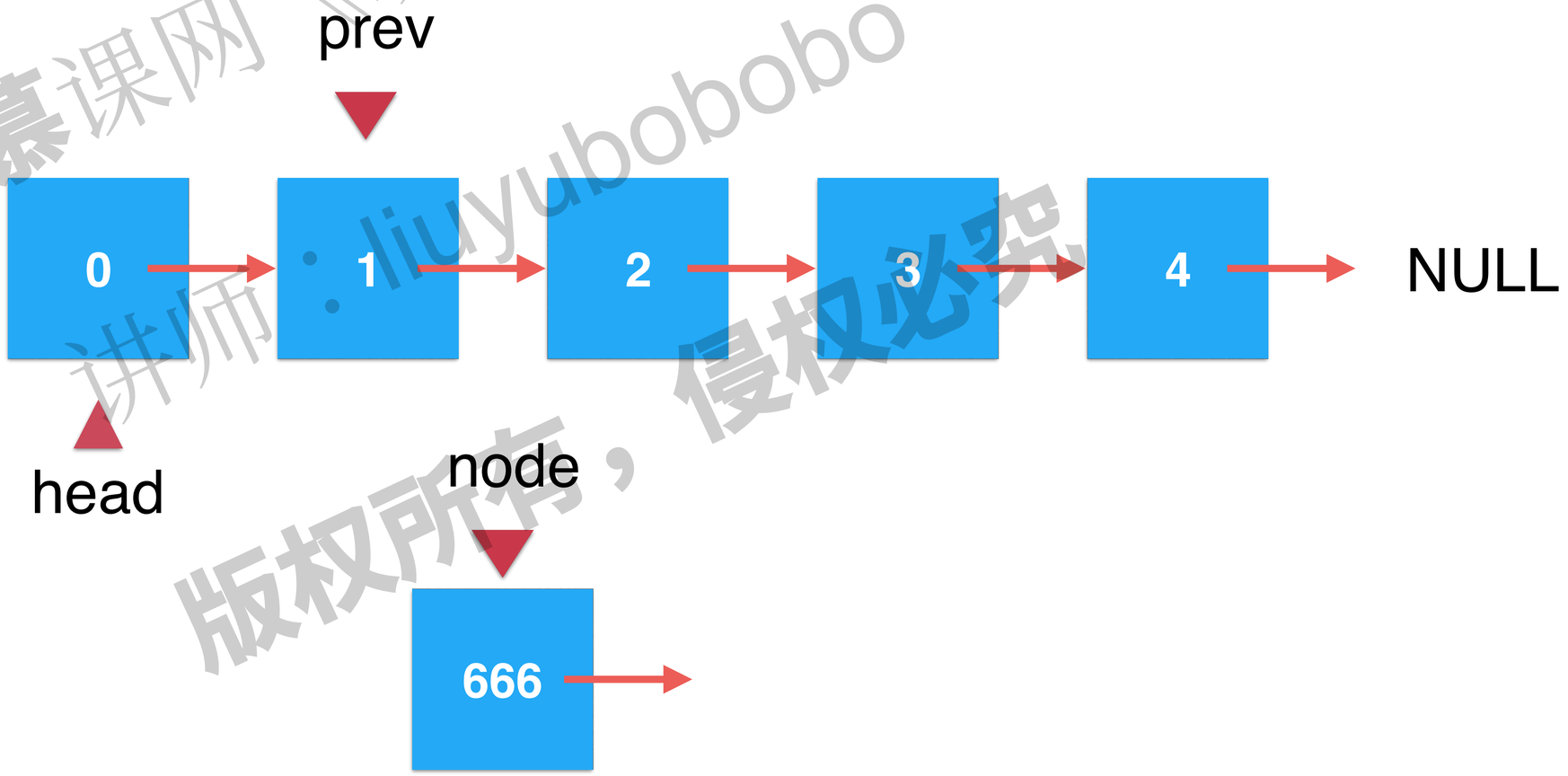
# 在链表中间添加元素

- 在索引为2的地方添加元素666



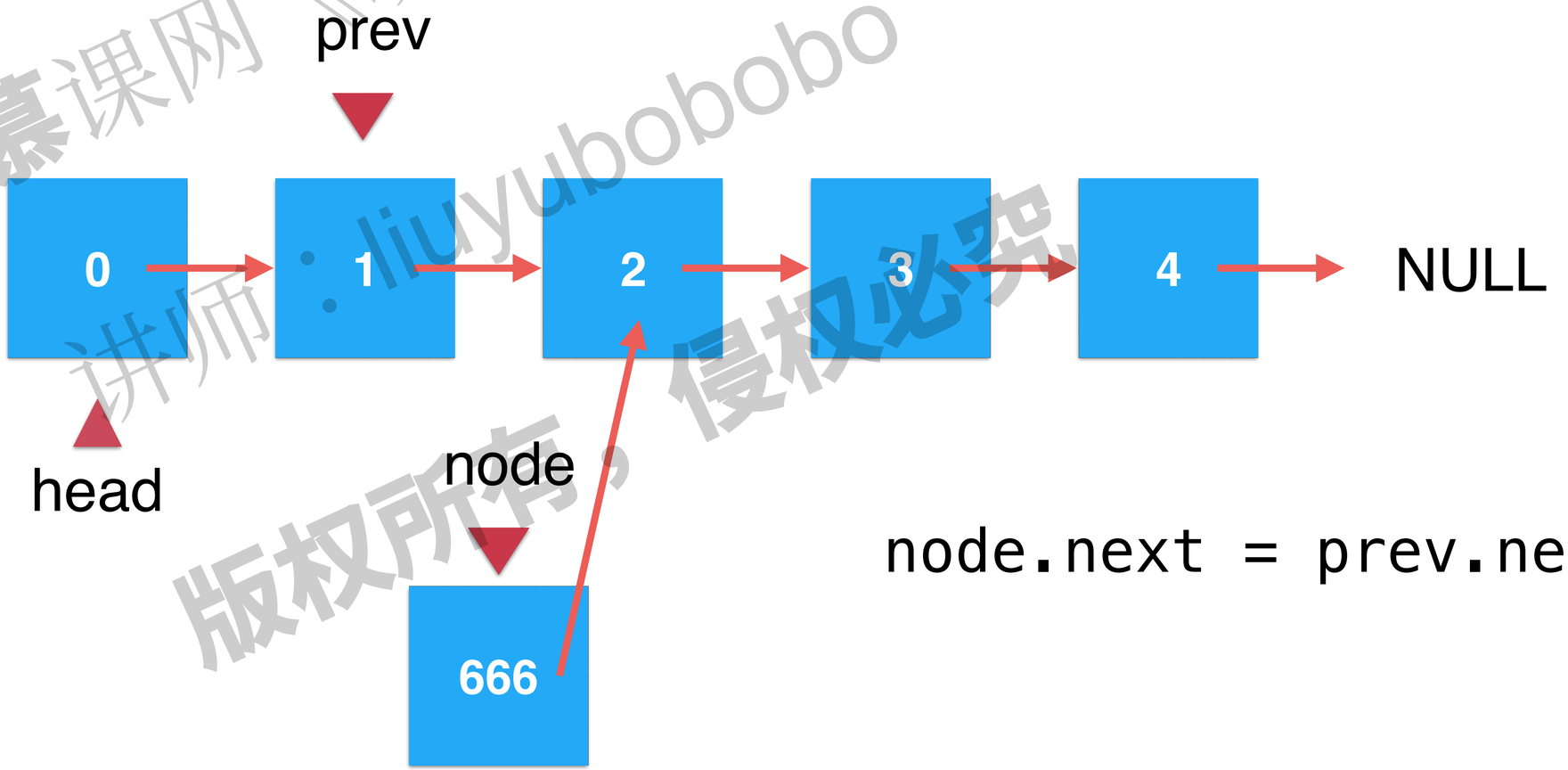
# 在链表中间添加元素

- 在索引为2的地方添加元素666



# 在链表中间添加元素

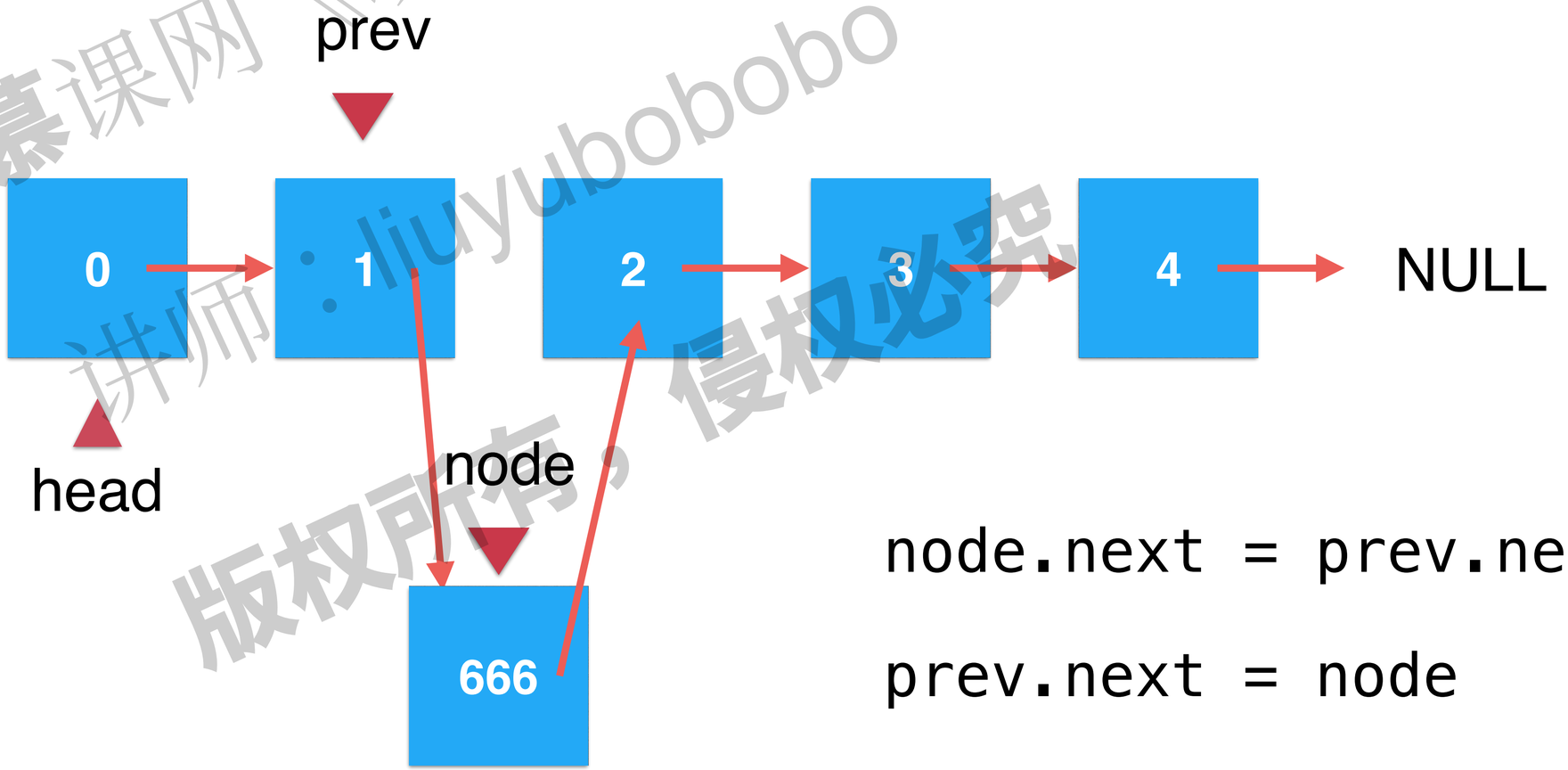
- 在索引为2的地方添加元素666




```
node.next = prev.next
```

# 在链表中间添加元素

- 在索引为2的地方添加元素666

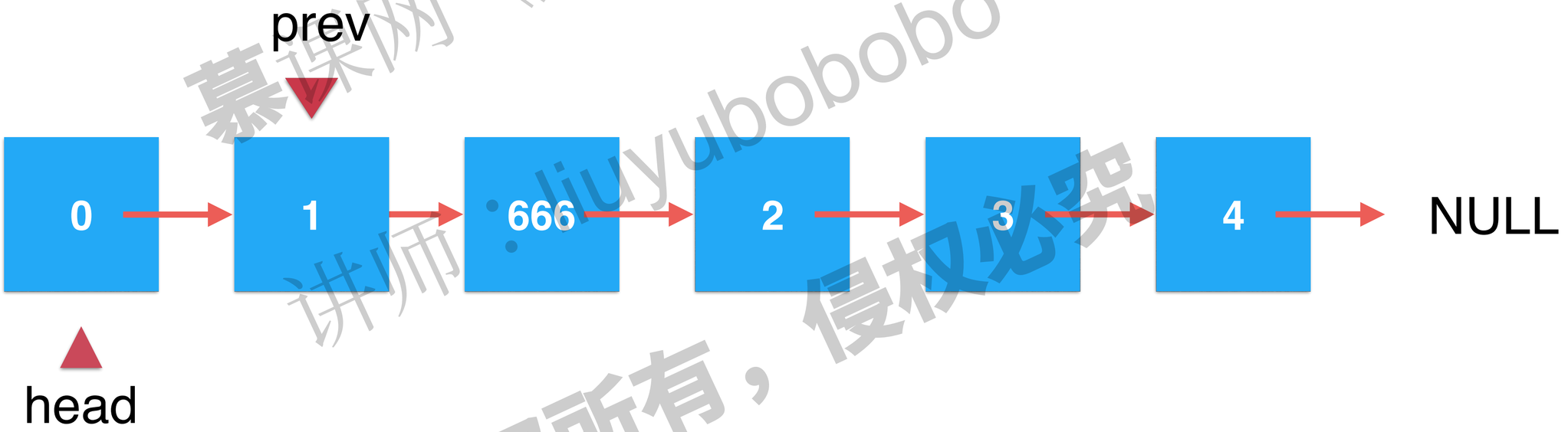



```
node.next = prev.next

prev.next = node
```

# 在链表中间添加元素

- 在索引为2的地方添加元素666



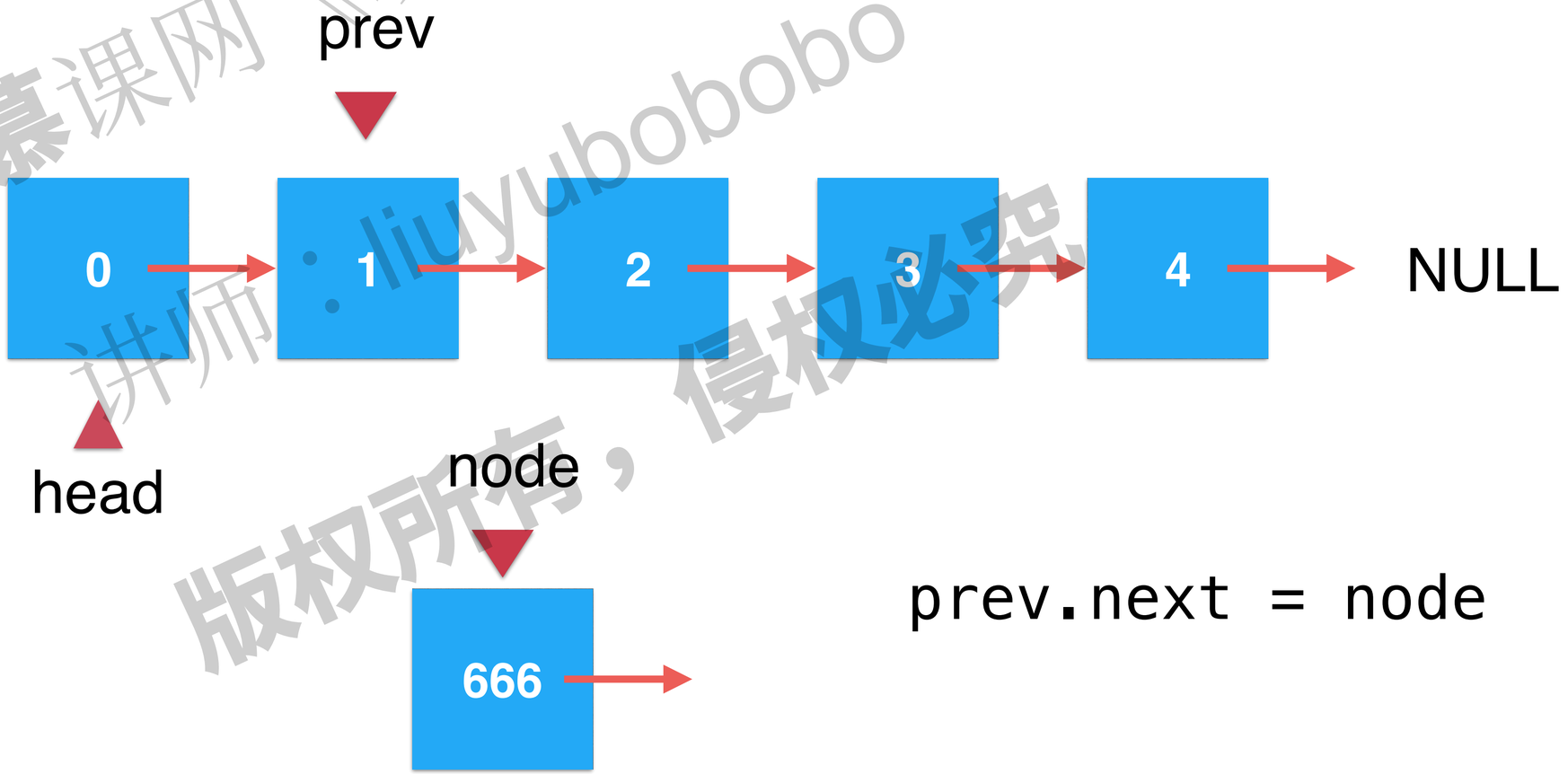

- 关键：找到要添加的节点的前一个节点

```
node.next = prev.next

prev.next = node
```

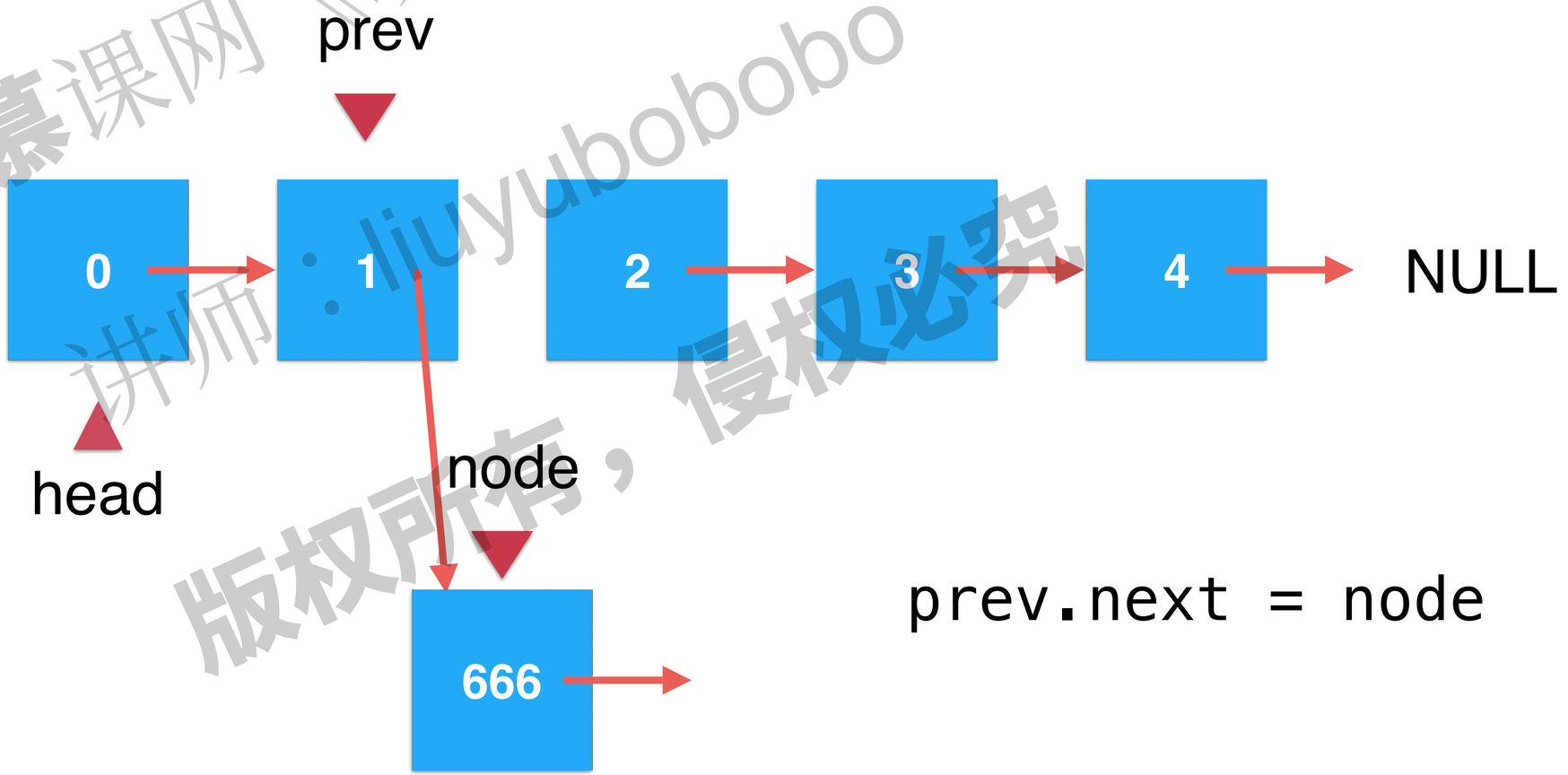
# 在链表中间添加元素

- 在索引为2的地方添加元素666



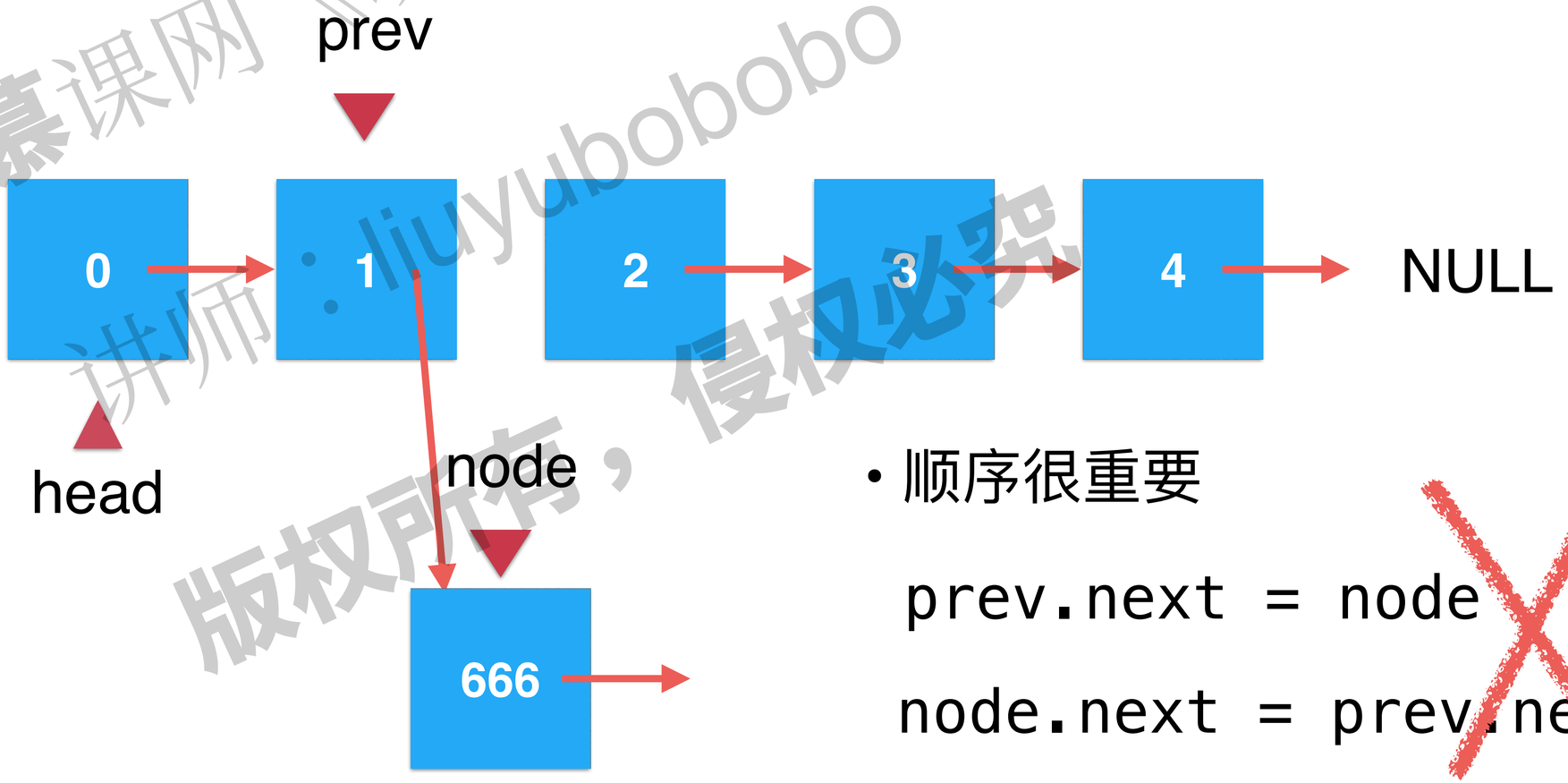
# 在链表中间添加元素

- 在索引为2的地方添加元素666



# 在链表中间添加元素

- 在索引为2的地方添加元素666



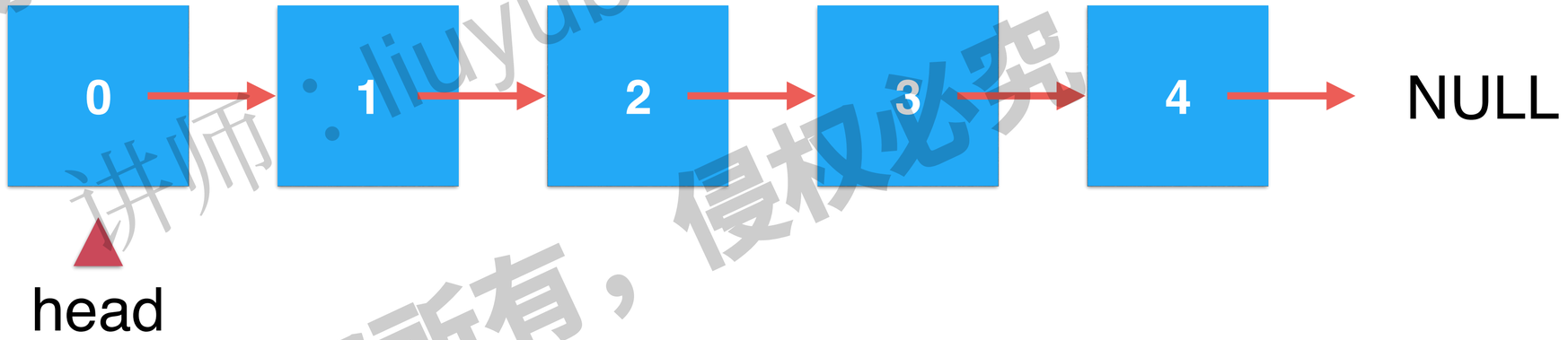

- 顺序很重要

```
prev.next = node
node.next = prev.next
```

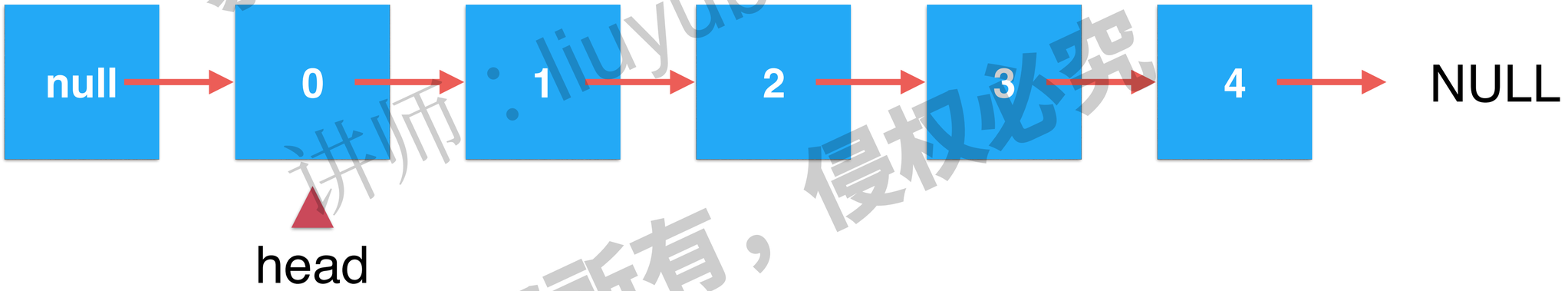
# 实践：在链表中间添加元素

# 为链表设立虚拟头结点

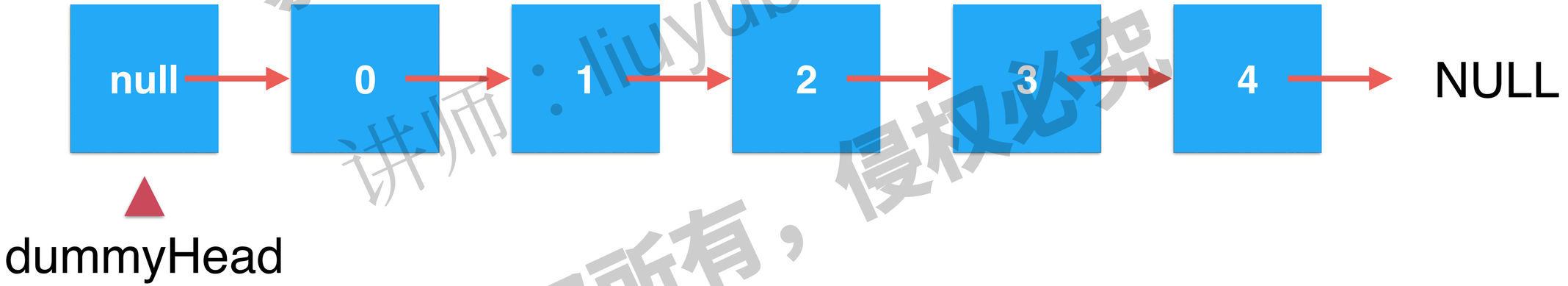
# 为链表设立虚拟头结点



# 为链表设立虚拟头结点



# 为链表设立虚拟头结点



# 实践：为链表设立虚拟头结点

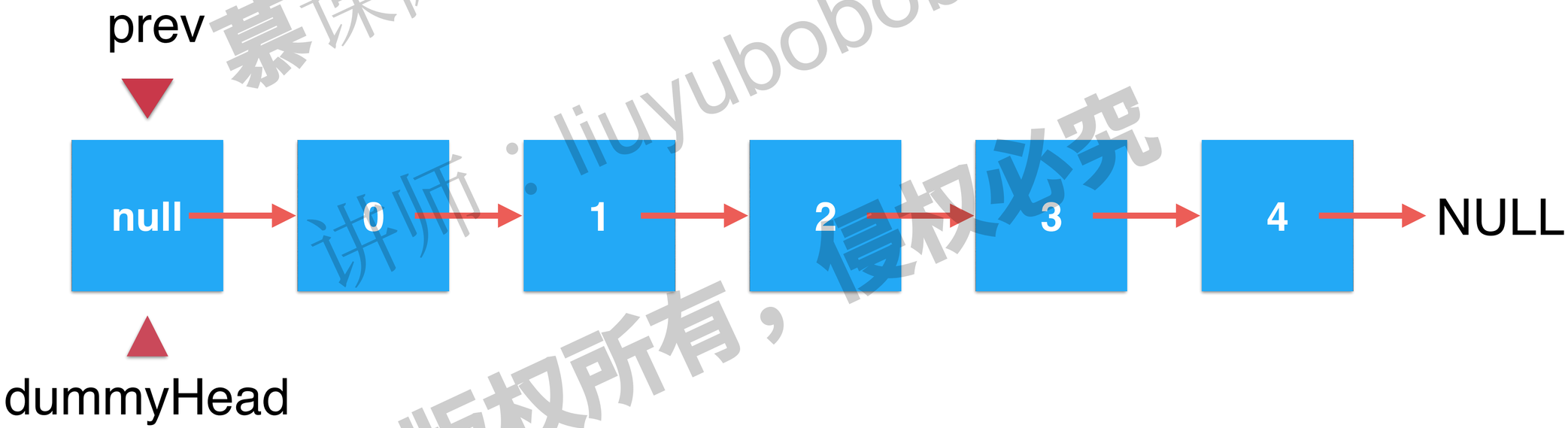
# 链表元素的查询，更新与遍历

# 实践：链表元素的遍历，查询与更新

# 链表元素的删除

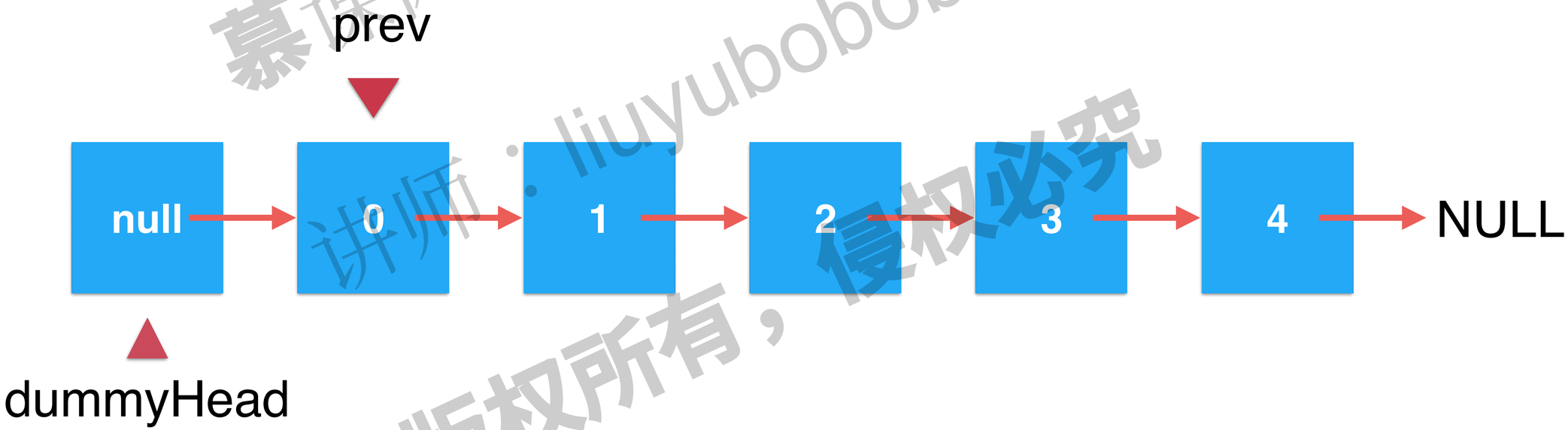
# 链表元素的删除

- 删除索引为2位置的元素



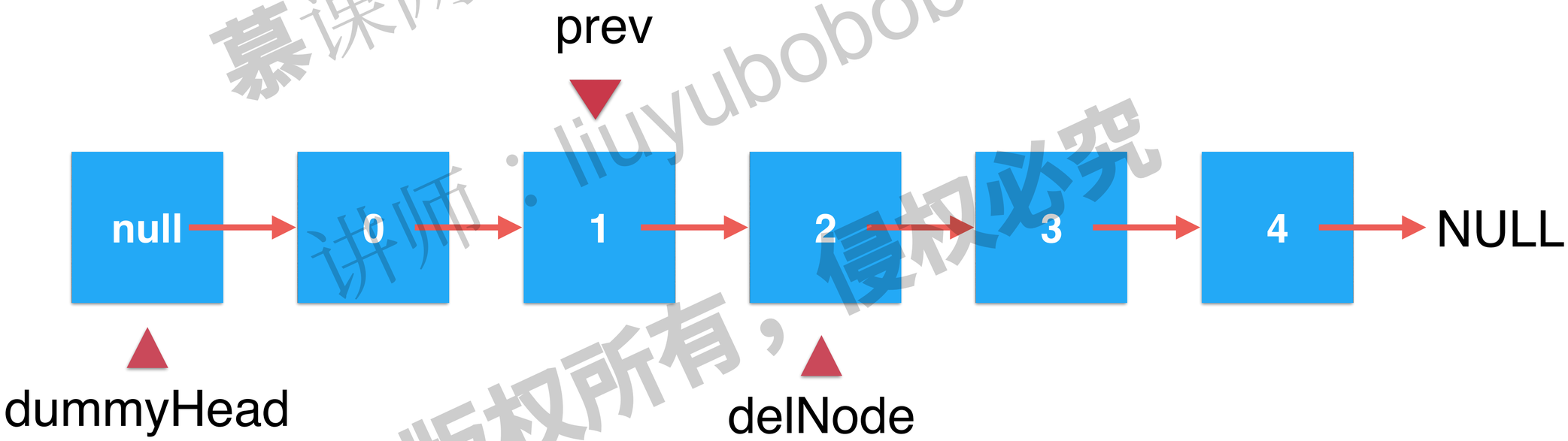
# 链表元素的删除

- 删除索引为2位置的元素



# 链表元素的删除

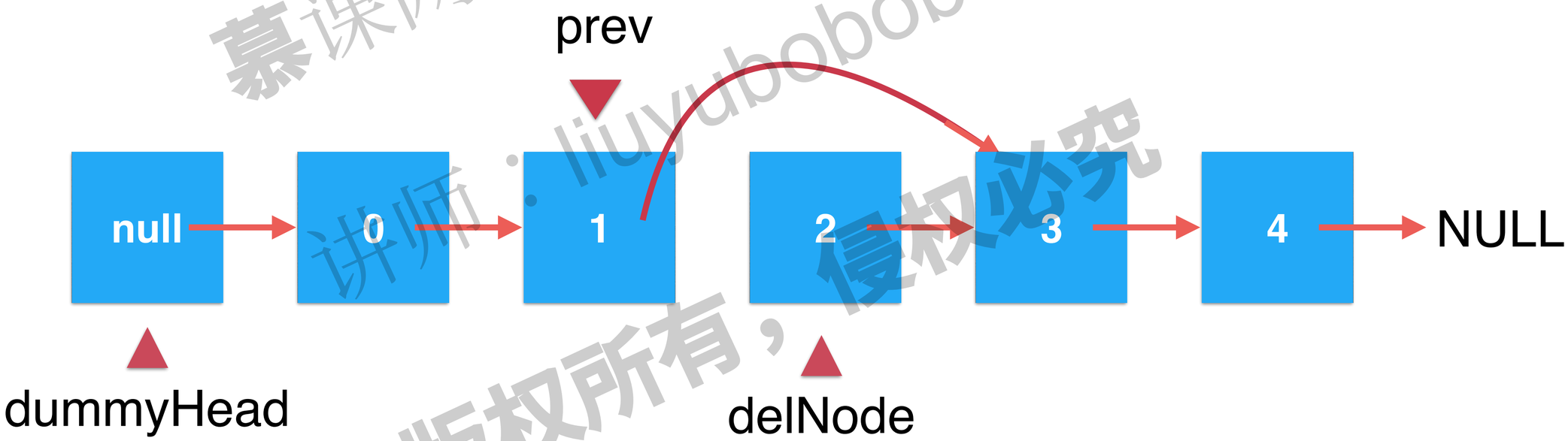
- 删除索引为2位置的元素




```
prev.next = delNode.next
```

# 链表元素的删除

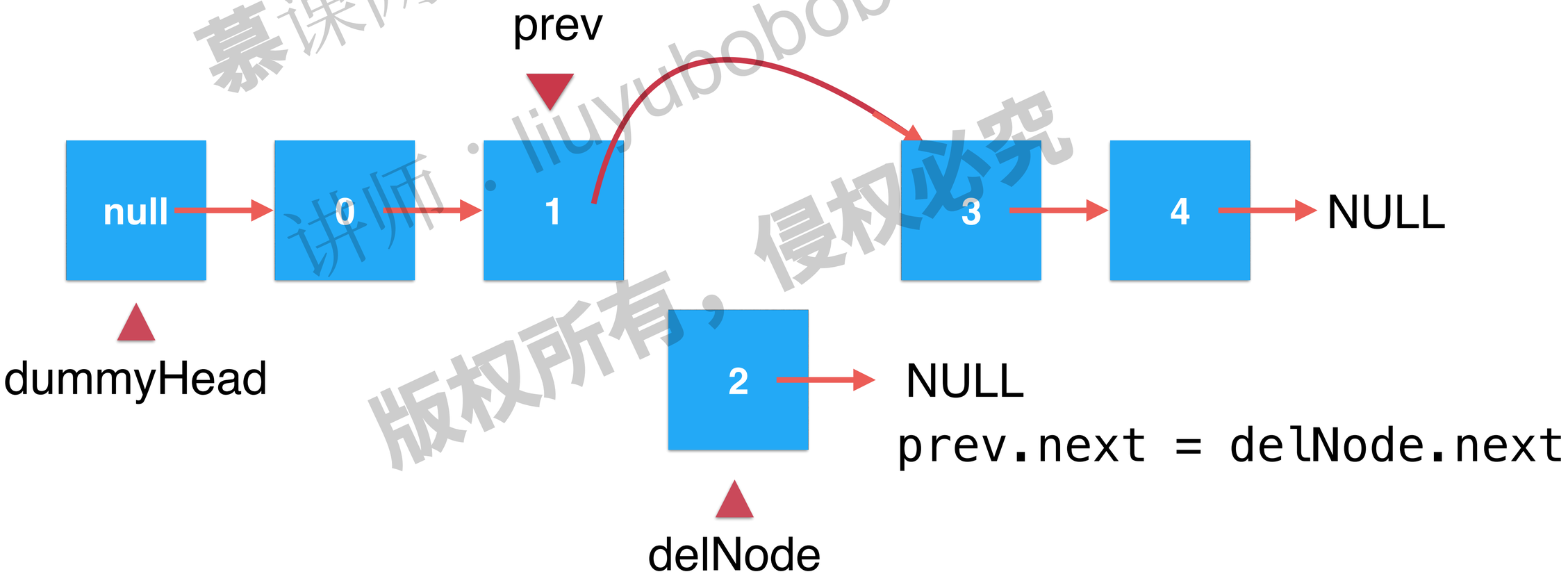
- 删除索引为2位置的元素




```
prev.next = delNode.next
```

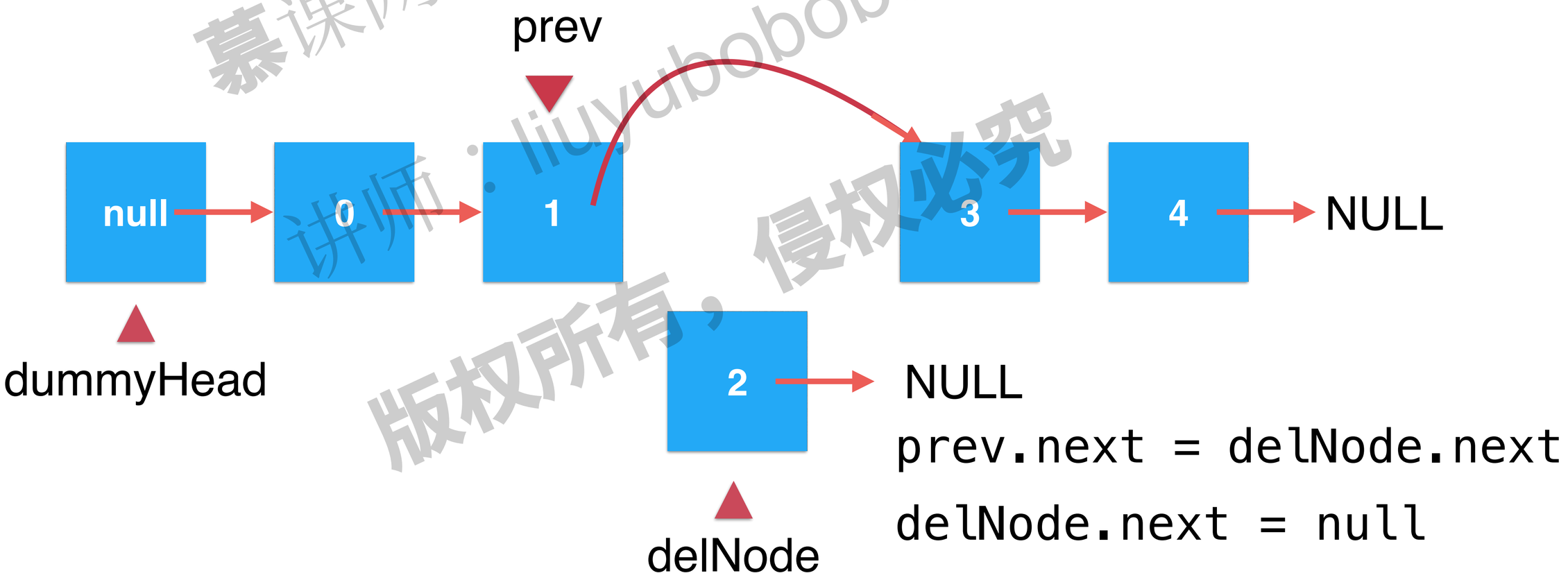
# 链表元素的删除

- 删除索引为2位置的元素



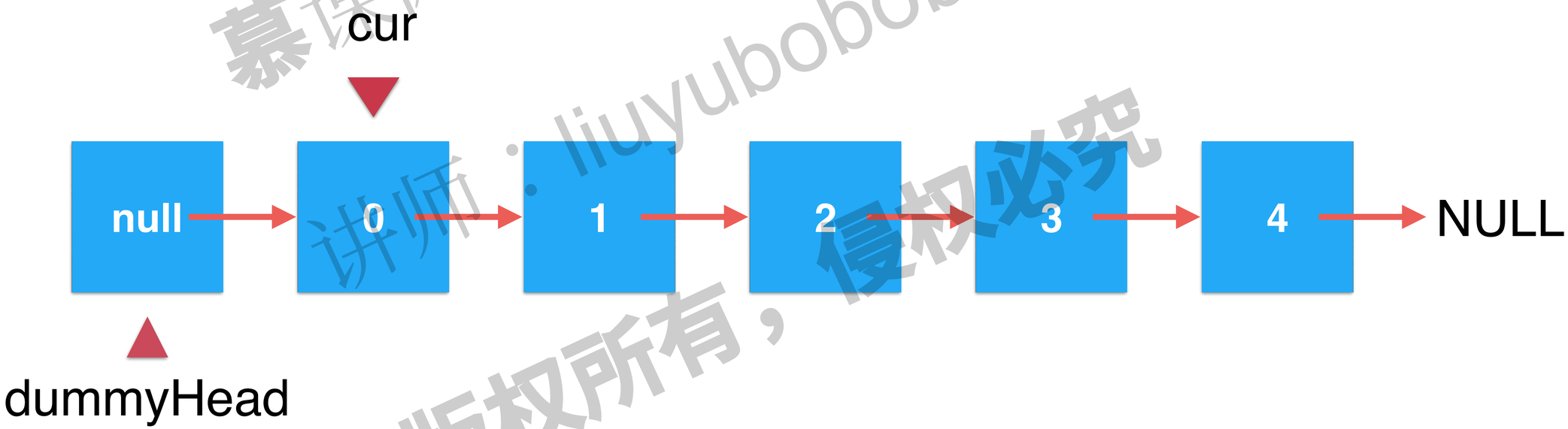
# 链表元素的删除

- 删除索引为2位置的元素



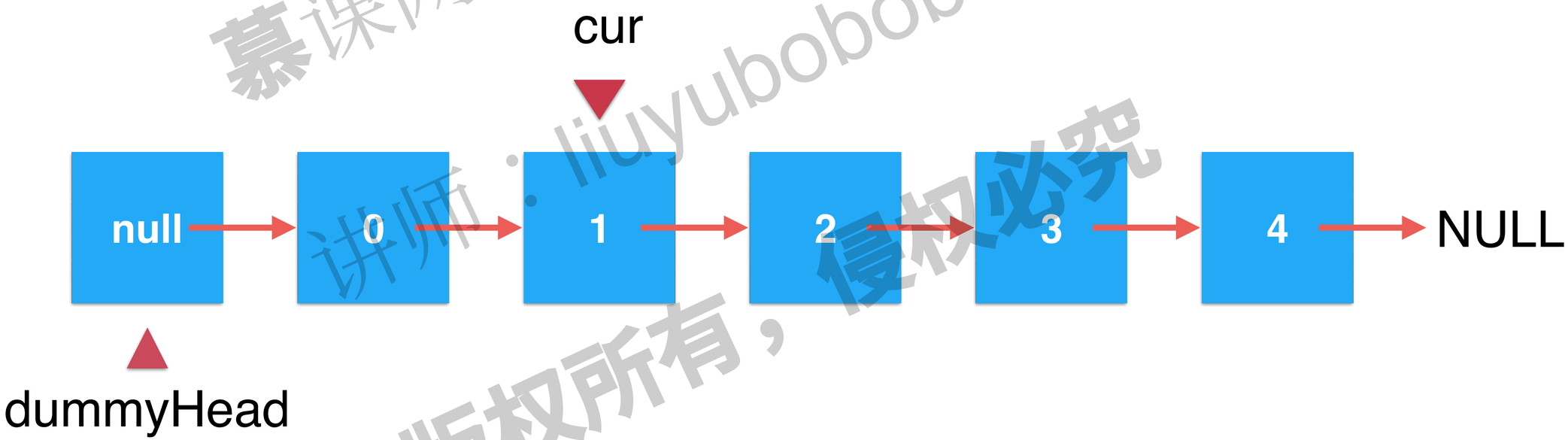
# 链表元素删除常见的错误

- 删除索引为2位置的元素



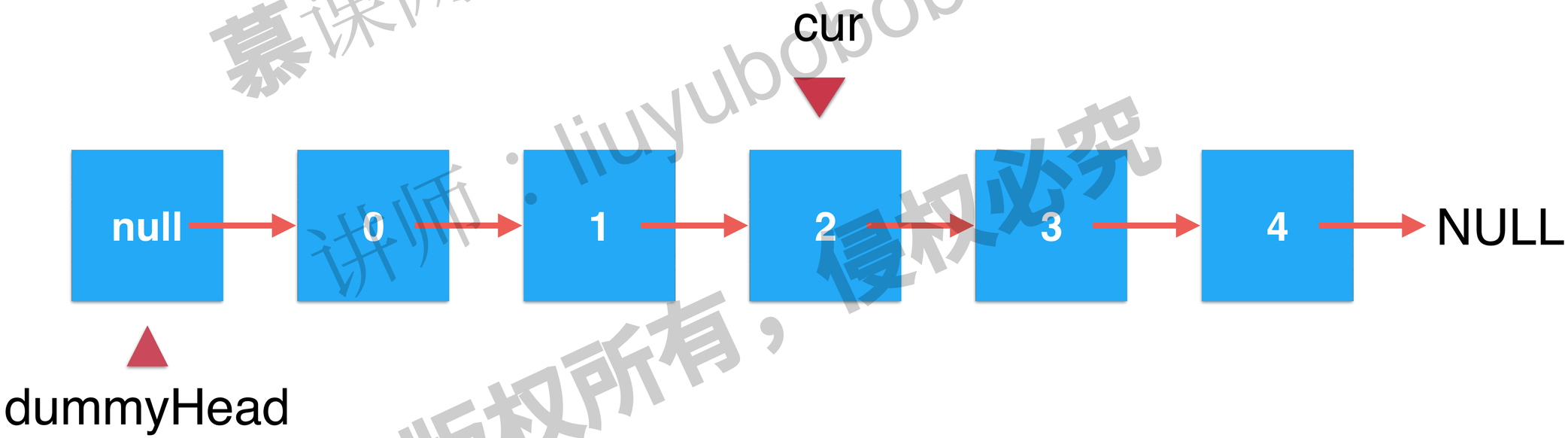
# 链表元素删除常见的错误

- 删除索引为2位置的元素



# 链表元素删除常见的错误

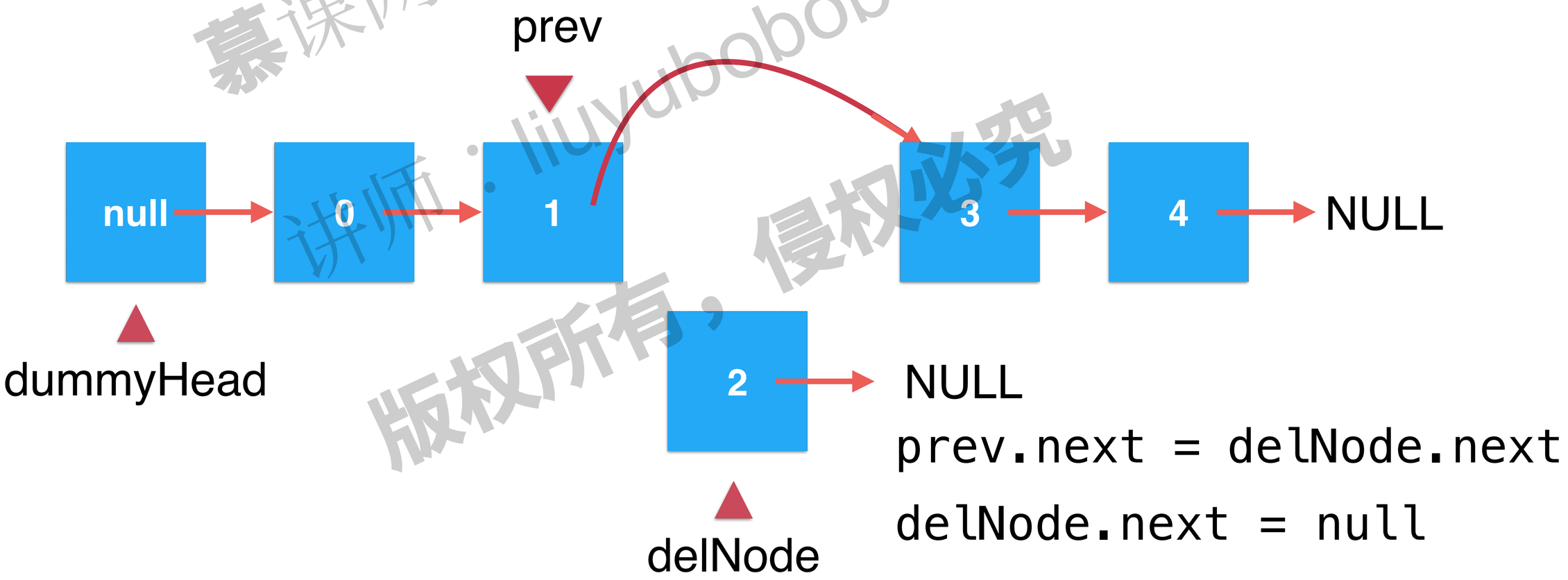
- 删除索引为2位置的元素




```
cur = cur.next
```

# 链表元素的删除

- 删除索引为2位置的元素



# 实践：链表元素的删除

# 链表的时间复杂度分析

- 添加操作 O(n)

  addLast(e) O(n)

  addFirst(e) O(1)

  add(index, e) O(n/2) = O(n)

# 链表的时间复杂度分析

- 删除操作　　O(n)

  removeLast(e)　O(n)

  removeFirst(e)　O(1)

  remove(index, e)　O(n/2) = O(n)

# 链表的时间复杂度分析

- 修改操作　　O(n)

set(index, e)　　O(n)

# 链表的时间复杂度分析

- 查找操作

| | |
|---|---|
| get(index) | O(n) |
| contains(e) | O(n) |
| ~~find(e)~~ | ~~O(n)~~ |

# 链表的时间复杂度分析

- 增： O(n)
- 删： O(n)
- 改： O(n)
- 查： O(n)

如果只对链表头进行操作：O(1)

# 链表的时间复杂度分析

- 增： O(n)
- 删： O(n)
- 改： O(n)
- 查： O(n)

如果只对链表头进行操作：O(1)

# 链表的时间复杂度分析

- 增： O(n)
- 删： O(n)

如果只对链表头进行操作：O(1)

- 改： O(n)

- 查： O(n)

只查链表头的元素：O(1)

# 使用链表实现栈

# 链表的时间复杂度分析

- 增： O(n)
- 删： O(n)

如果只对链表头进行操作：O(1)

- 改： O(n)

- 查： O(n)

只查链表头的元素：O(1)

# 使用链表实现栈


- void push(E)
- E pop()
- E peek()
- int getSize()
- boolean isEmpty()

# 实践：使用链表实现栈

# 使用链表实现队列

# 链表的时间复杂度分析

- 增： O(n)
- 删： O(n)

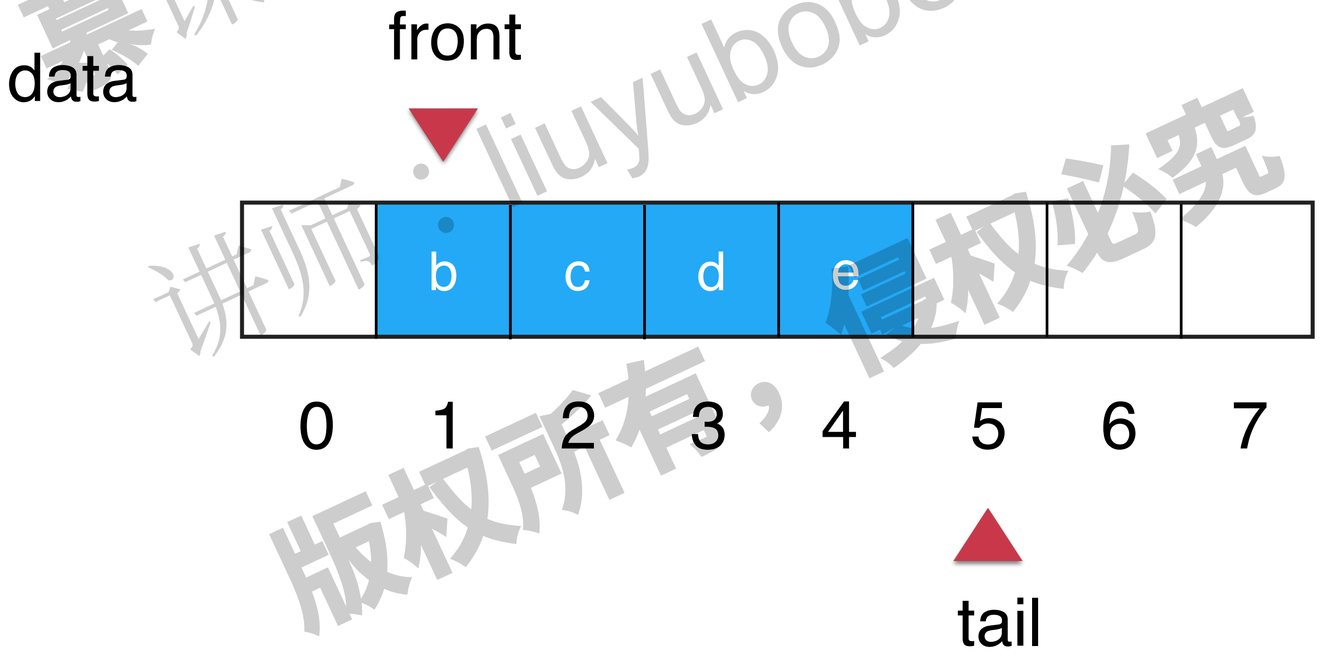
如果只对链表头进行操作：O(1)

- 改： O(n)

- 查： O(n)

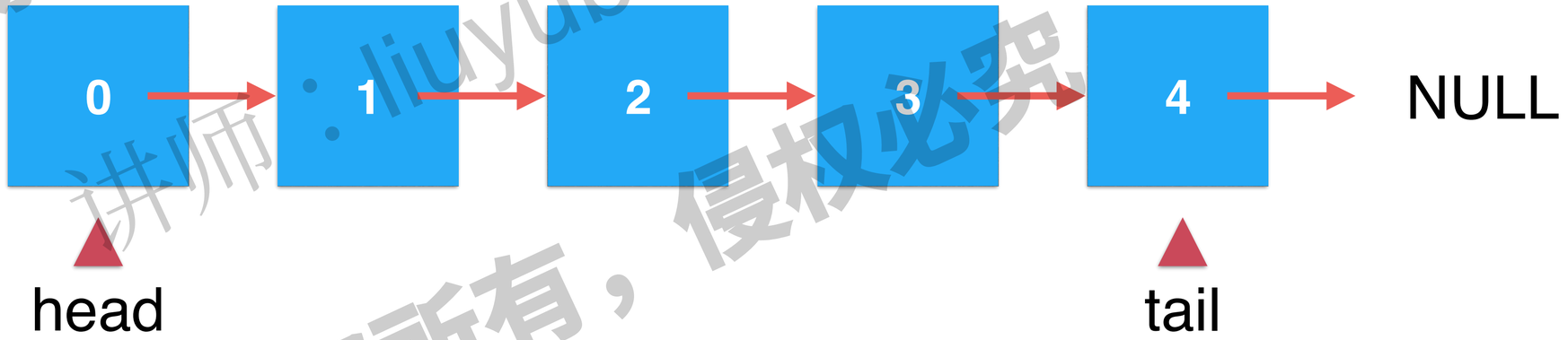
只查链表头的元素：O(1)

# 循环队列



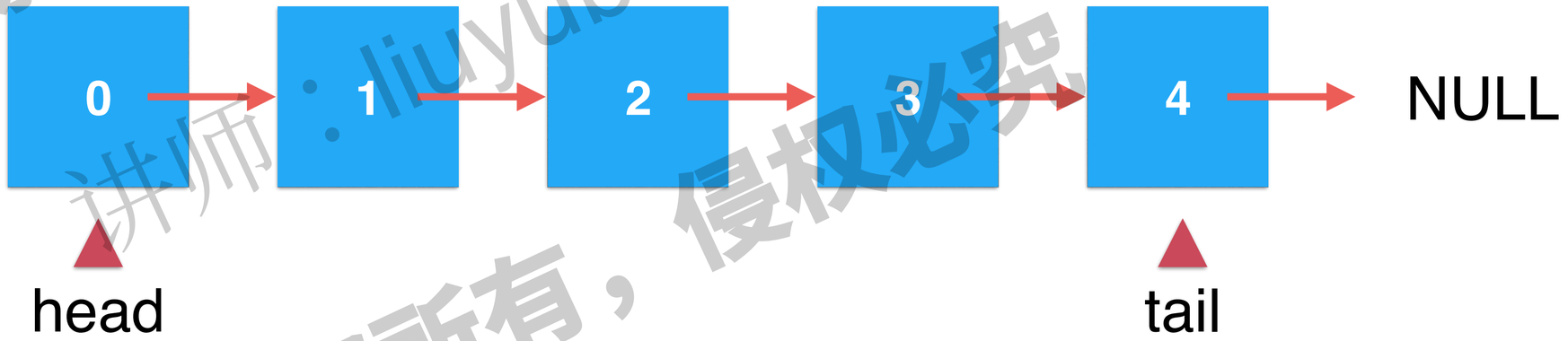
# 改进我们的链表






- 从两端插入元素都是容易的

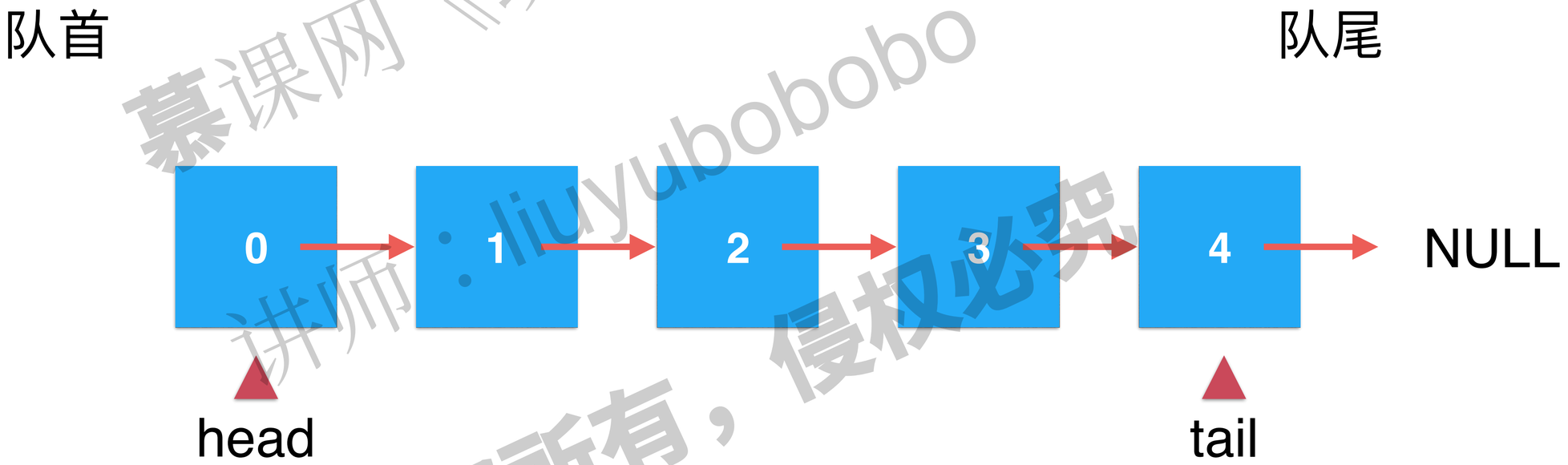
# 改进我们的链表






- 从tail删除元素不容易

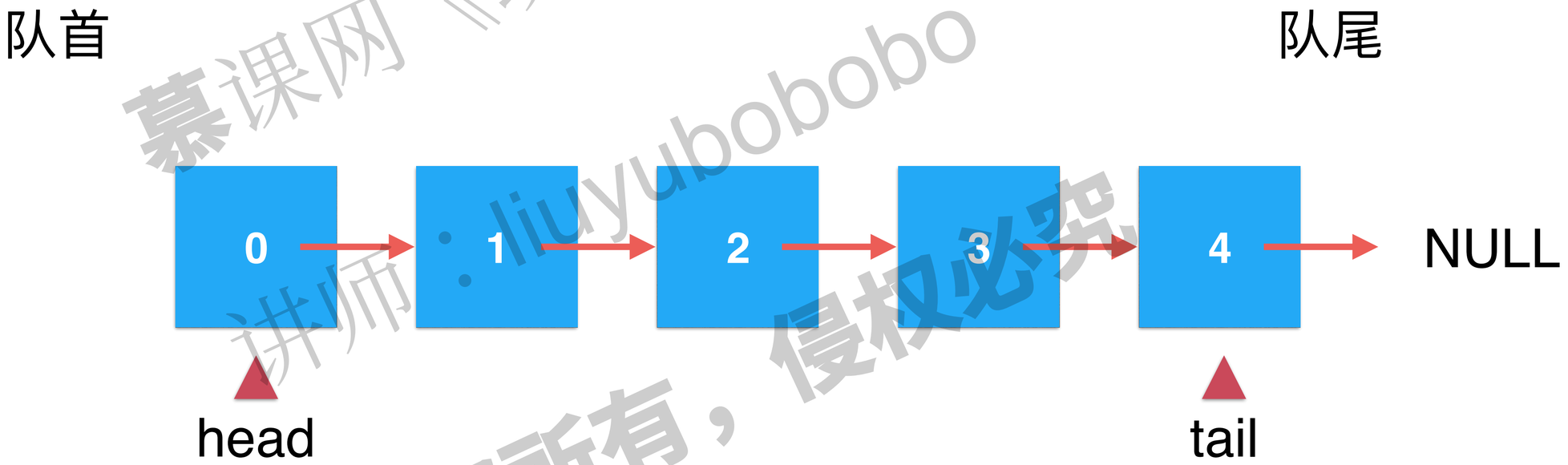
# 改进我们的链表






- 从head端删除元素，从tail端插入元素

# 改进我们的链表






- 由于没有dummyHead，要注意链表为空的情况

# 实践：使用链表实现队列

# 链表

# 其他

欢迎大家关注我的个人公众号：是不是很酷

# 玩儿转数据结构

liuyubobobo