

# 极客时间算法训练营

## 第一课

### 数组、链表

李煜东

《算法竞赛进阶指南》作者



# 目录

1. 数组原理讲解、实战应用
2. 设计变长数组
3. 链表原理讲解、实战应用



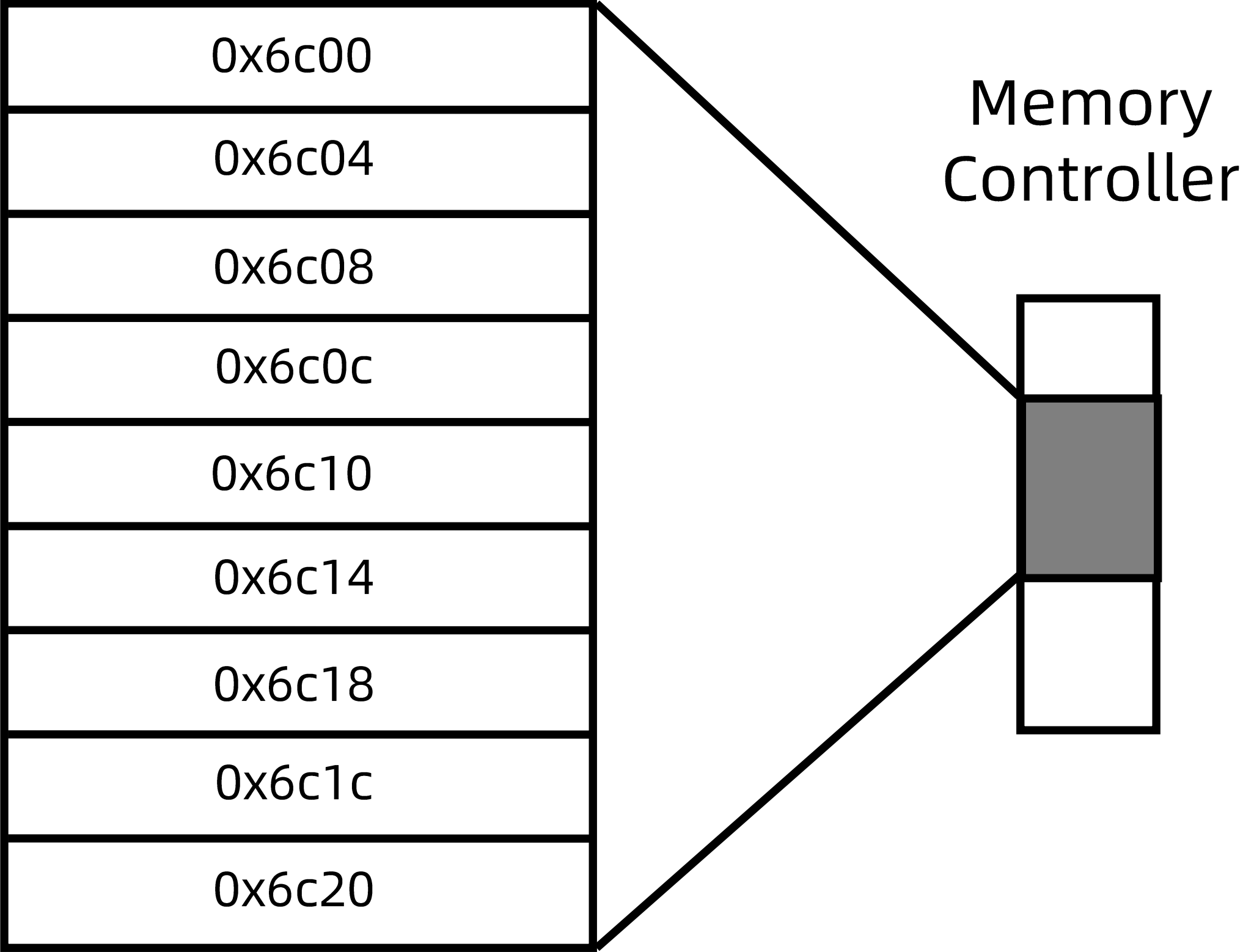
# 数组

# 数组 (array)

- C++: `int a[100];`
- Java: `int[] a = new int[100];`
- Python: `a = []`
  
- 数组的基本特点：支持随机访问
- 数组的关键：索引与寻址
  
- C++: `a[i], *(a+i)`
- Java, Python: `a[i]`
  
- 数组在内存中是一段连续的存储空间

# 数组 (array)

0	123
1	234
2	345
3	456
4	567
5	678
6	789
7	890
8	901



# 数组 - 插入元素

Inserting

0	A
1	B
2	C
3	E
4	F
5	G
6	

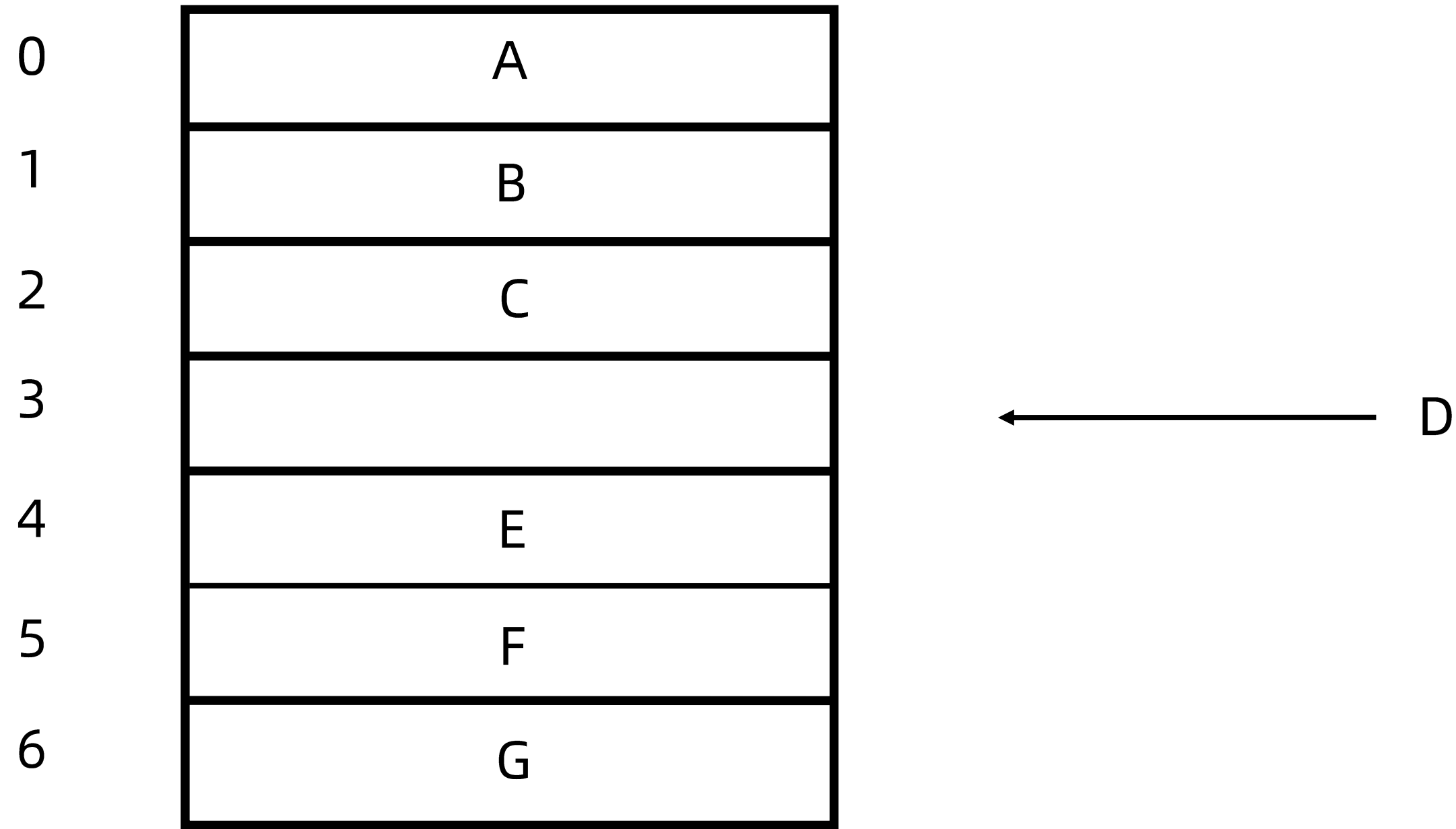
# 数组 - 插入元素

Inserting

0	A	
1	B	
2	C	
3	E	D
4	F	
5	G	
6		

# 数组 - 插入元素

Inserting





# 数组 - 插入元素

Inserting

0	A
1	B
2	C
3	D
4	E
5	F
6	G

# 数组 - 删除元素


Deleting

0	A
1	B
2	C
3	Z
4	D
5	E
6	F

# 数组 - 删除元素

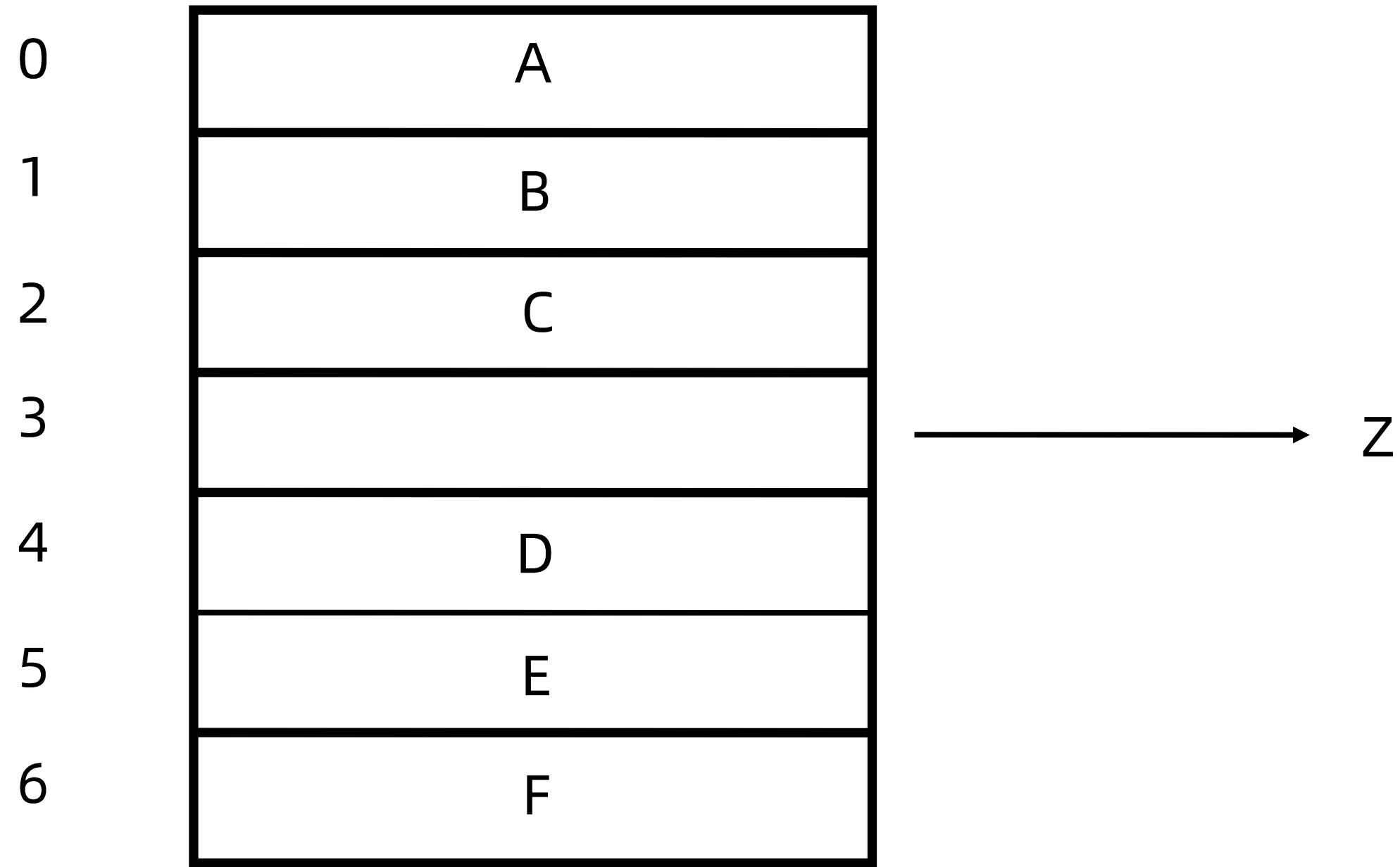
Deleting

0	A
1	B
2	C
3	Z
4	D
5	E
6	F



# 数组 - 删除元素

Deleting



# 数组 - 删除元素

Deleting

0	A	
1	B	
2	C	
3	D	Z
4	E	
5	F	
6		

# 时间复杂度

- Lookup  $O(1)$
- Insert  $O(n)$
- Delete  $O(n)$
- Append (push back)  $O(1)$
- Prepend (push front)  $O(n)$



# 实战

- 去重
- <https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/>
- 移动零
- <https://leetcode-cn.com/problems/move-zeroes/>
- 合并有序数组
- <https://leetcode-cn.com/problems/merge-sorted-array/>

# 变长数组 (resizable array)

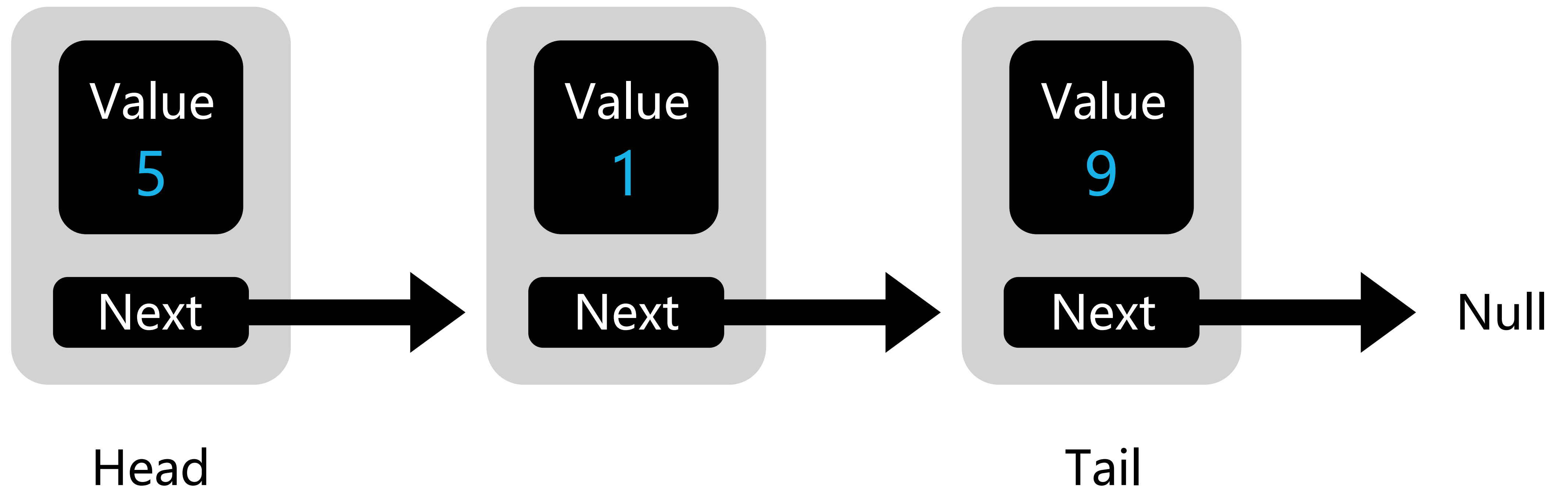
- C++: vector
- Java: ArrayList
- Python: list
- 如何实现一个变长数组?
  - 支持索引与随机访问
  - 分配多长的连续空间?
  - 空间不够用了怎么办?
  - 空间剩余很多如何回收?

# 变长数组 (resizable array)

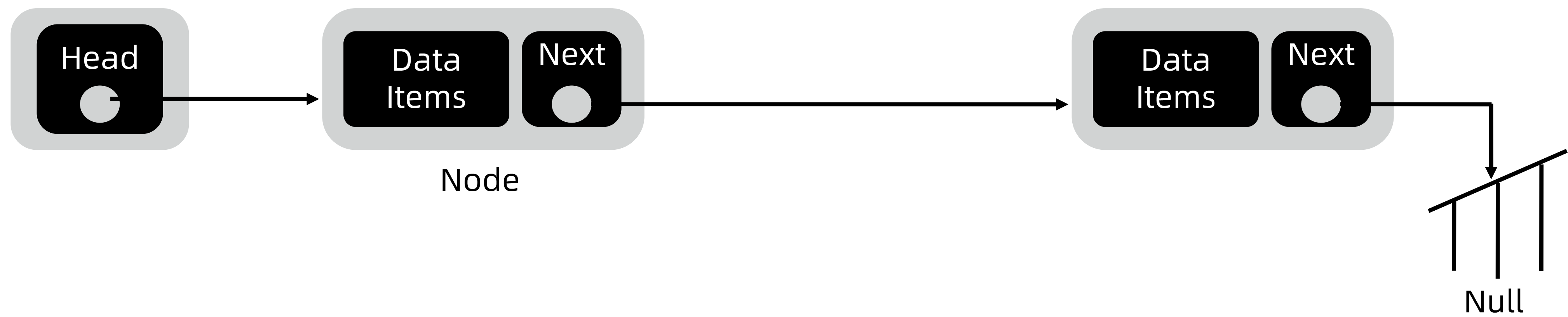
- 一个简易的实现方法
- 初始：空数组，分配常数空间，记录实际长度 (size) 和容量 (capacity)
- Push back: 若空间不够，重新申请 2 倍大小的连续空间，拷贝到新空间，释放旧空间
- Pop back: 若空间利用率 (size/capacity) 不到 25%，释放一半的空间
- 均摊  $O(1)$
- 在空数组中连续插入  $n$  个元素，总插入/拷贝次数为  $n + n/2 + n/4 + \dots < 2n$
- 一次扩容到下一次释放，至少需要再删除  $n - 2n * 0.25 = 0.5n$  次
- 思考：若释放空间的阈值设定为 50%，会发生什么情况？

# 链表

# 单链表 (linked list)

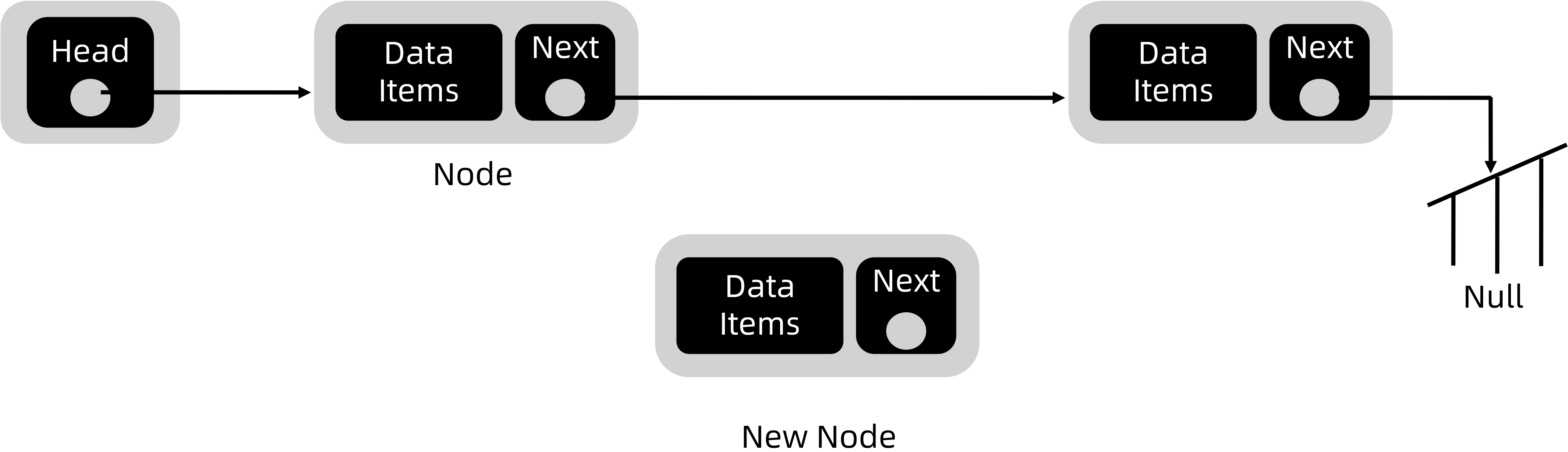


# 单链表 - 插入

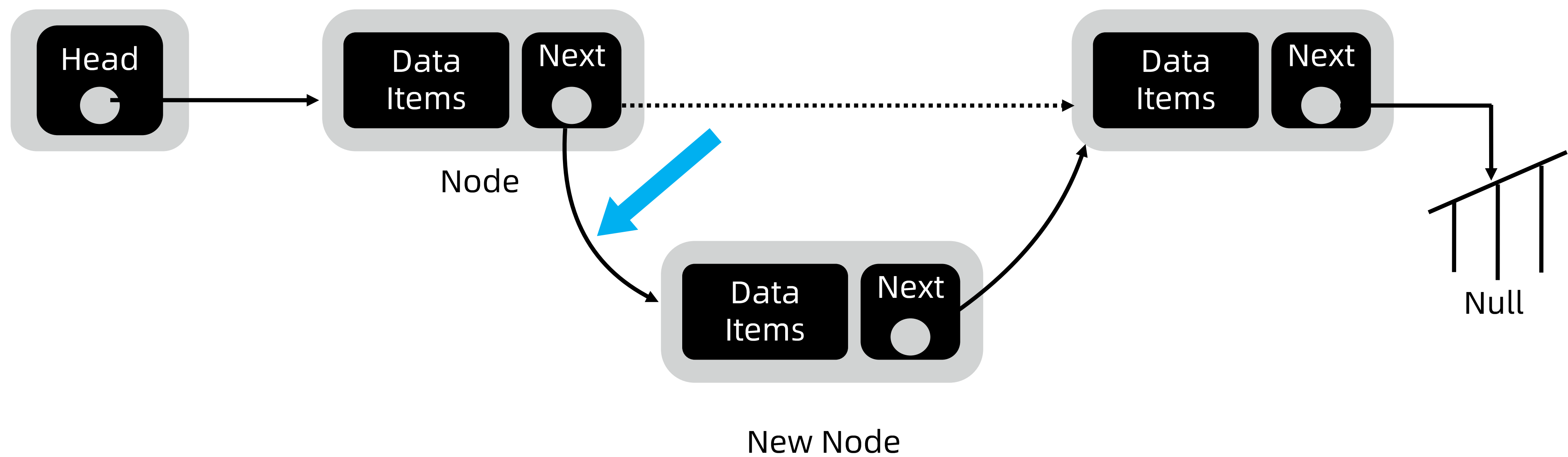




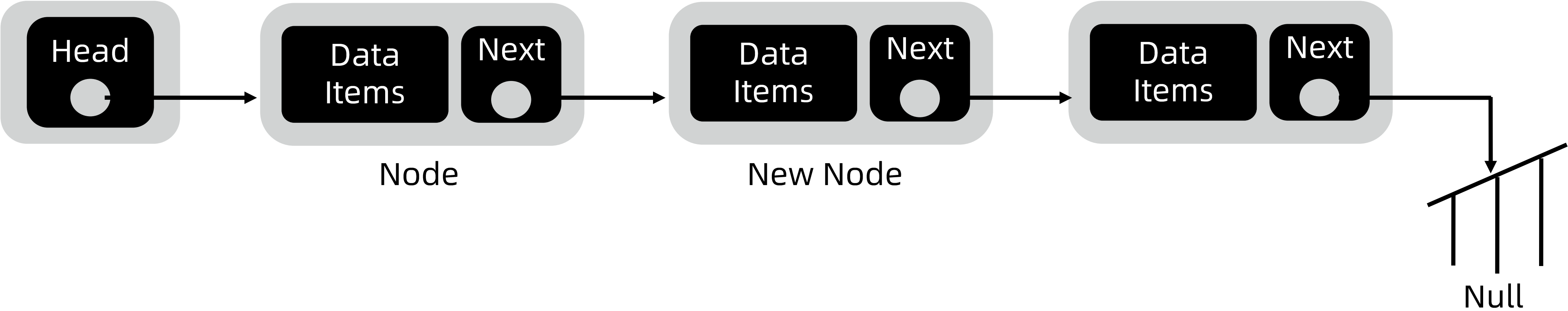
# 单链表 - 插入



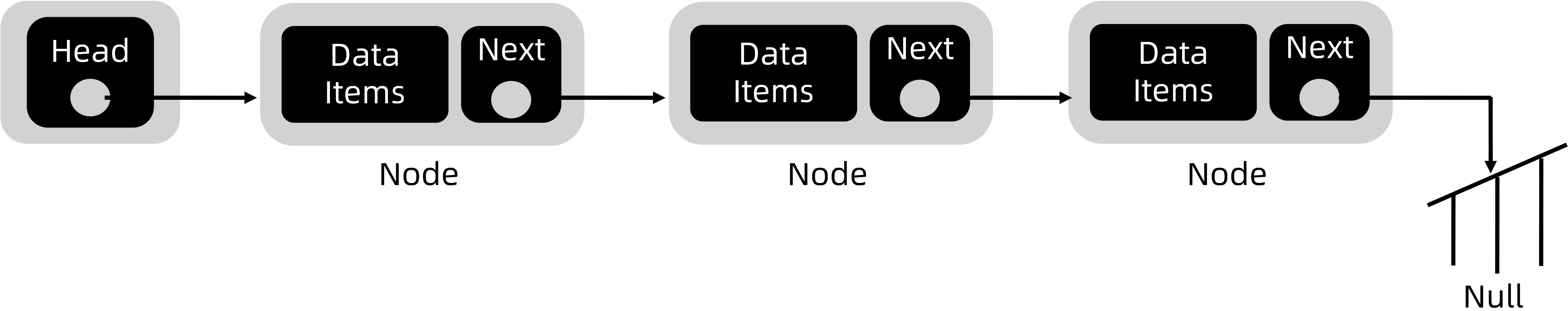
# 单链表 - 插入



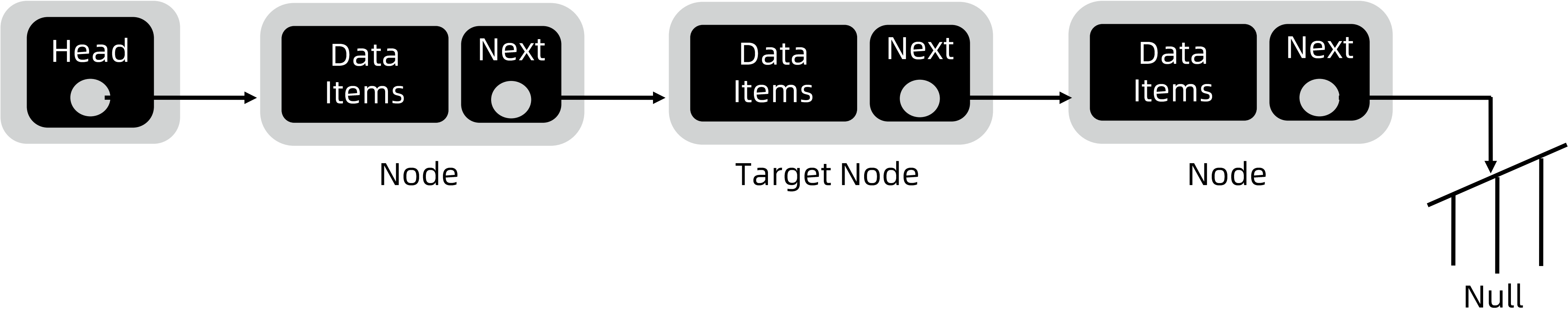
# 单链表 - 插入



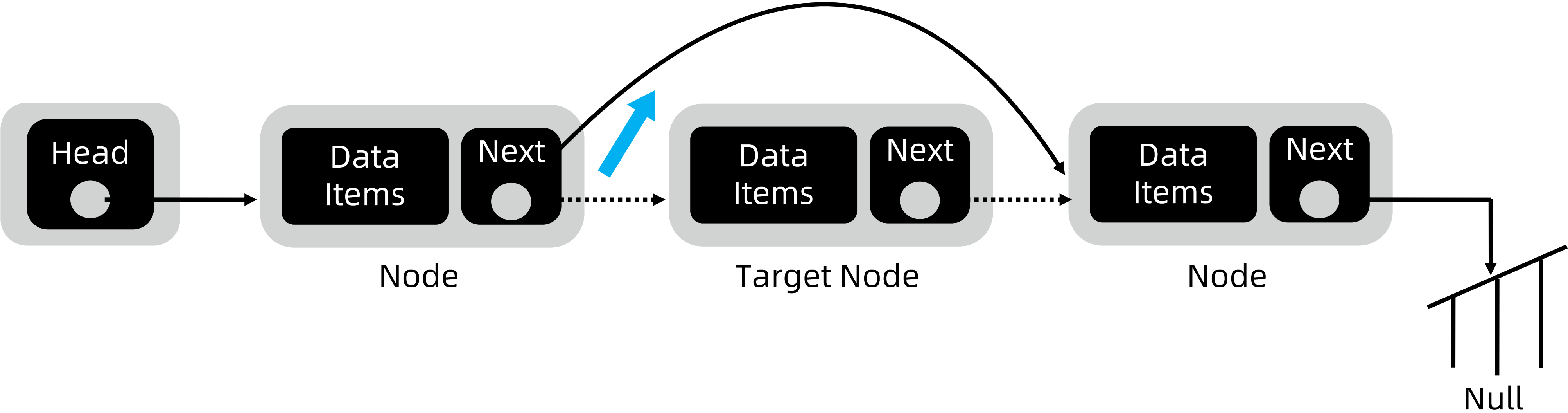
# 单链表 - 删除



# 单链表 - 删除

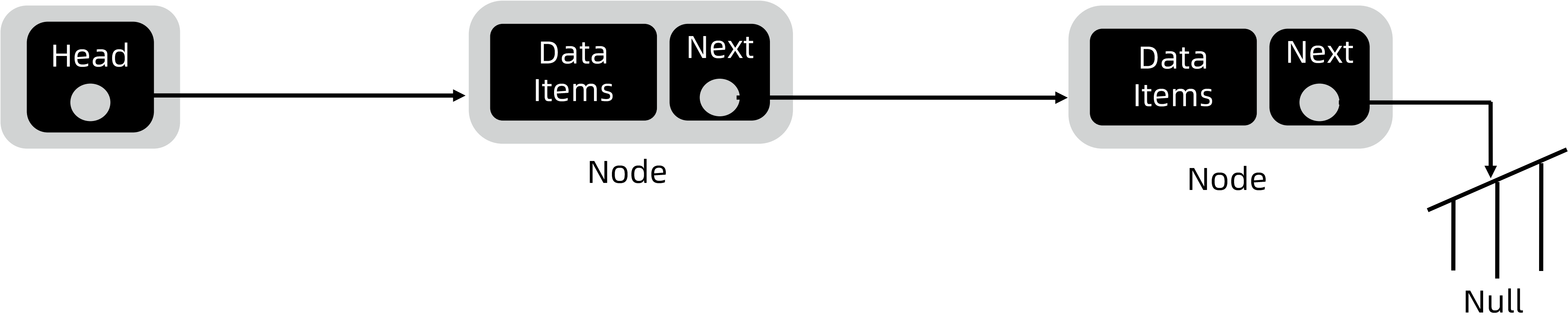


# 单链表 - 删除

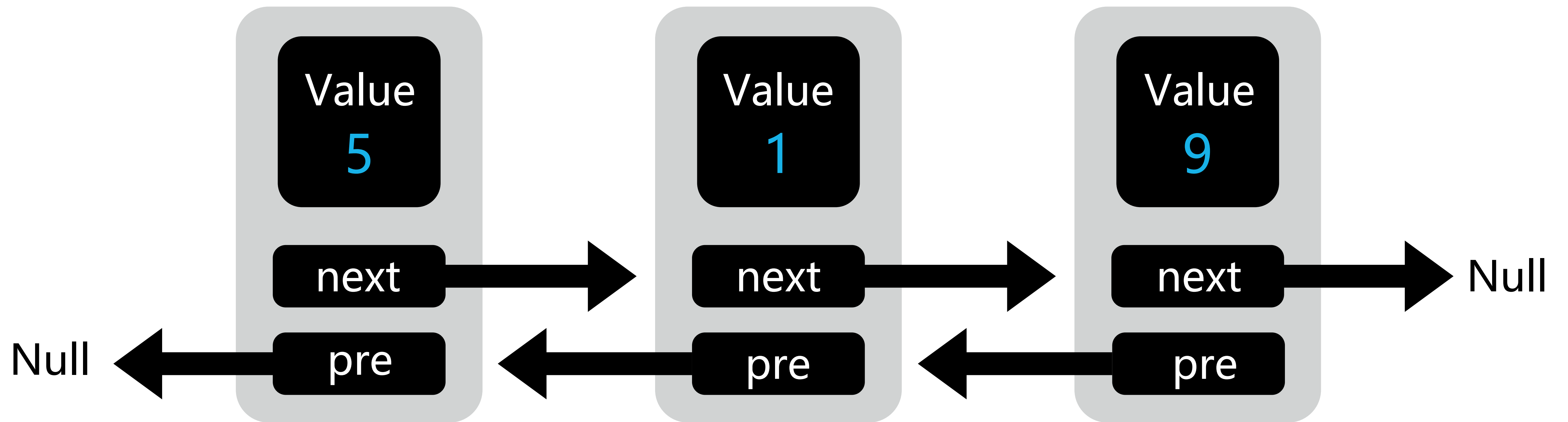




# 单链表 - 删除



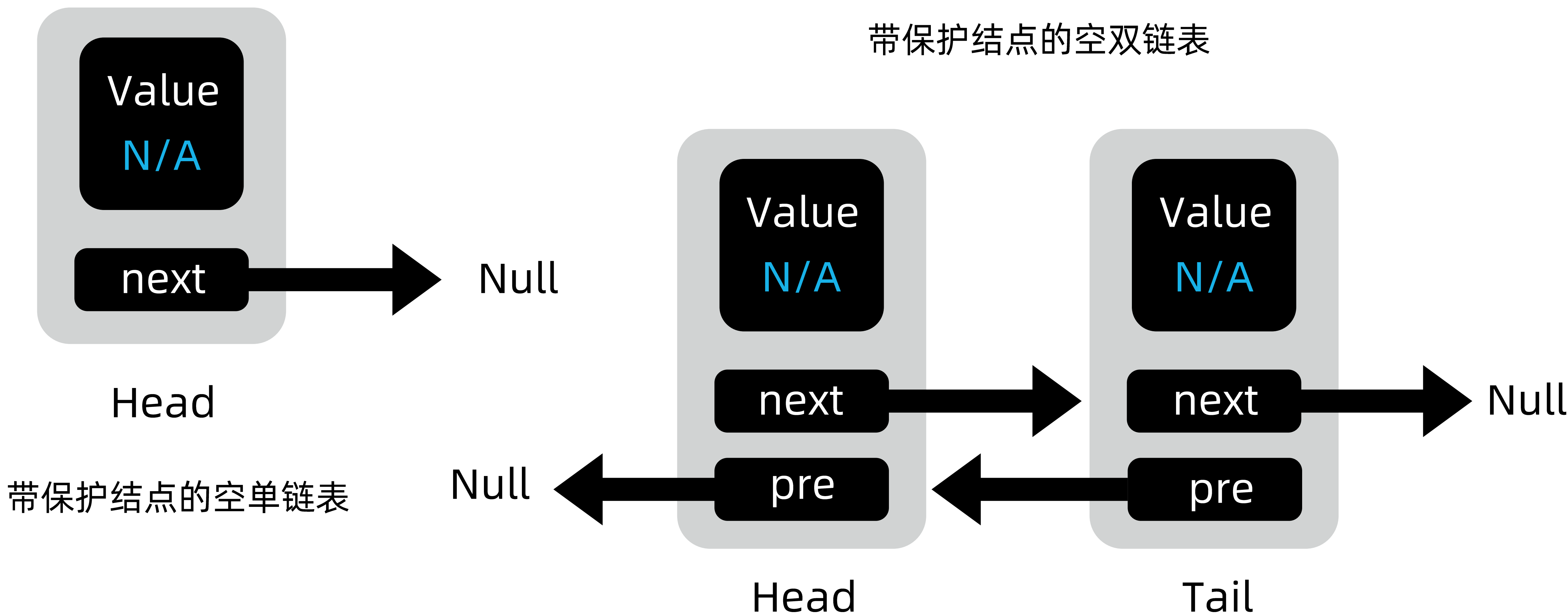
# 双链表 (double linked list)



# 时间复杂度

- Lookup  $O(n)$
- Insert  $O(1)$
- Delete  $O(1)$
- Append (push back)  $O(1)$
- Prepend (push front)  $O(1)$

# 保护结点



# 实战

- 反转链表
- <https://leetcode-cn.com/problems/reverse-linked-list/>
- K个一组翻转链表
- <https://leetcode-cn.com/problems/reverse-nodes-in-k-group/>

# 实战：邻值查找（Hard）

<https://www.acwing.com/problem/content/description/138/>

给定一个长度为  $n$  的序列  $A$ ， $A$  中的数各不相同。

对于  $A$  中的每一个数  $A_i$ ，求  $i$  前面与  $A_i$  相差最小的数，即：

$$\min_{1 \leq j < i} |A_i - A_j|$$

以及令上式取到最小值的位置  $j$ 。

若最小值点不唯一，则选择使  $A_j$  较小的那个。



# 实战：邻值查找

<https://www.acwing.com/problem/content/description/138/>

- 按数值排序，建立有序双链表
- 链表虽然不能随机访问，但可以记录A数组每个下标对应的链表结点
- 倒序考虑每个下标，只需要在链表中查找前驱、后继，然后删除结点

## 关键点

- “索引”的灵活性——按下标/按值
- 不同“索引”的数据结构之间建立“映射”关系
- 倒序考虑问题

# 实战

环形链表

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

环形链表 II

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

# 实战

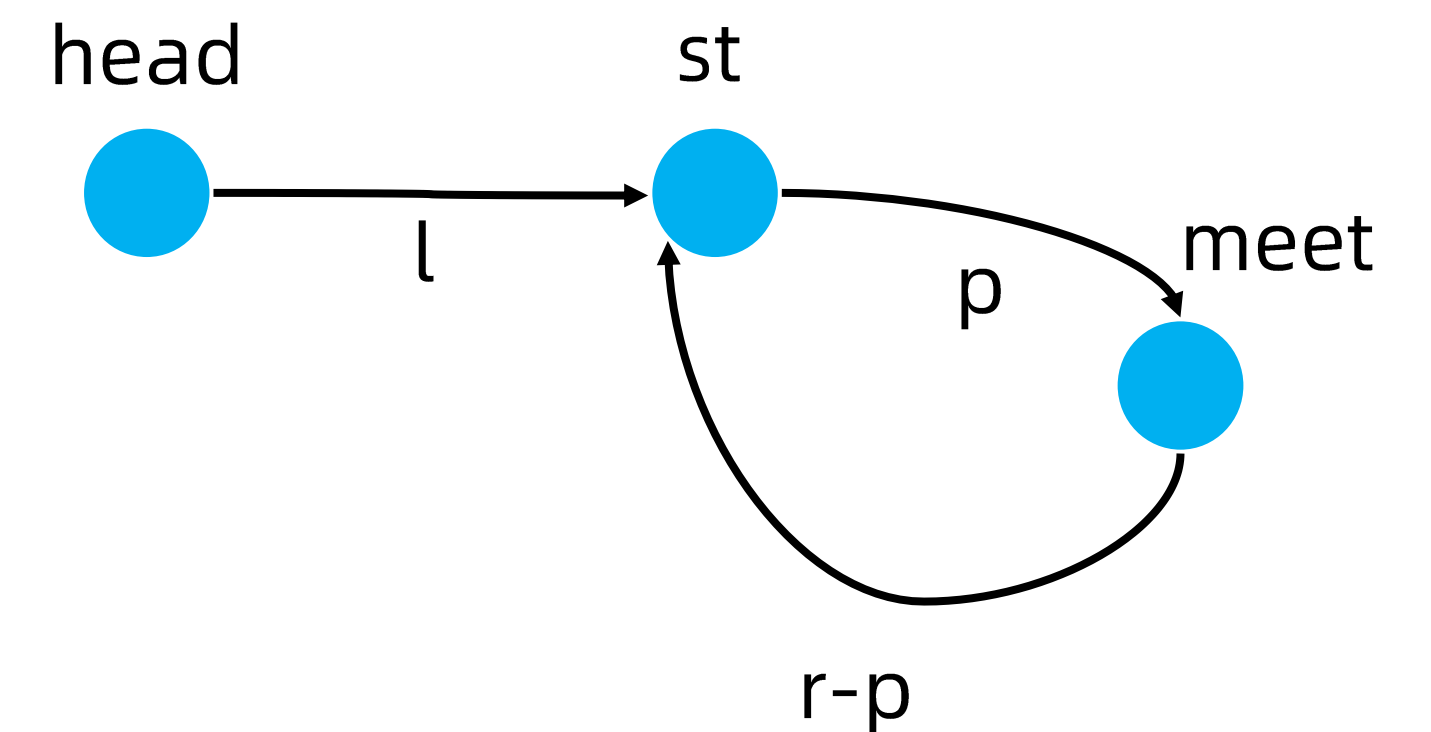
## 环形链表

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

- 快慢指针法,  $O(\text{length})$  时间,  $O(1)$  空间
- 有环必定发生套圈（快慢指针相遇），无环不会发生套圈（快指针到达null）

## 环形链表 II

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



# 实战

## 环形链表

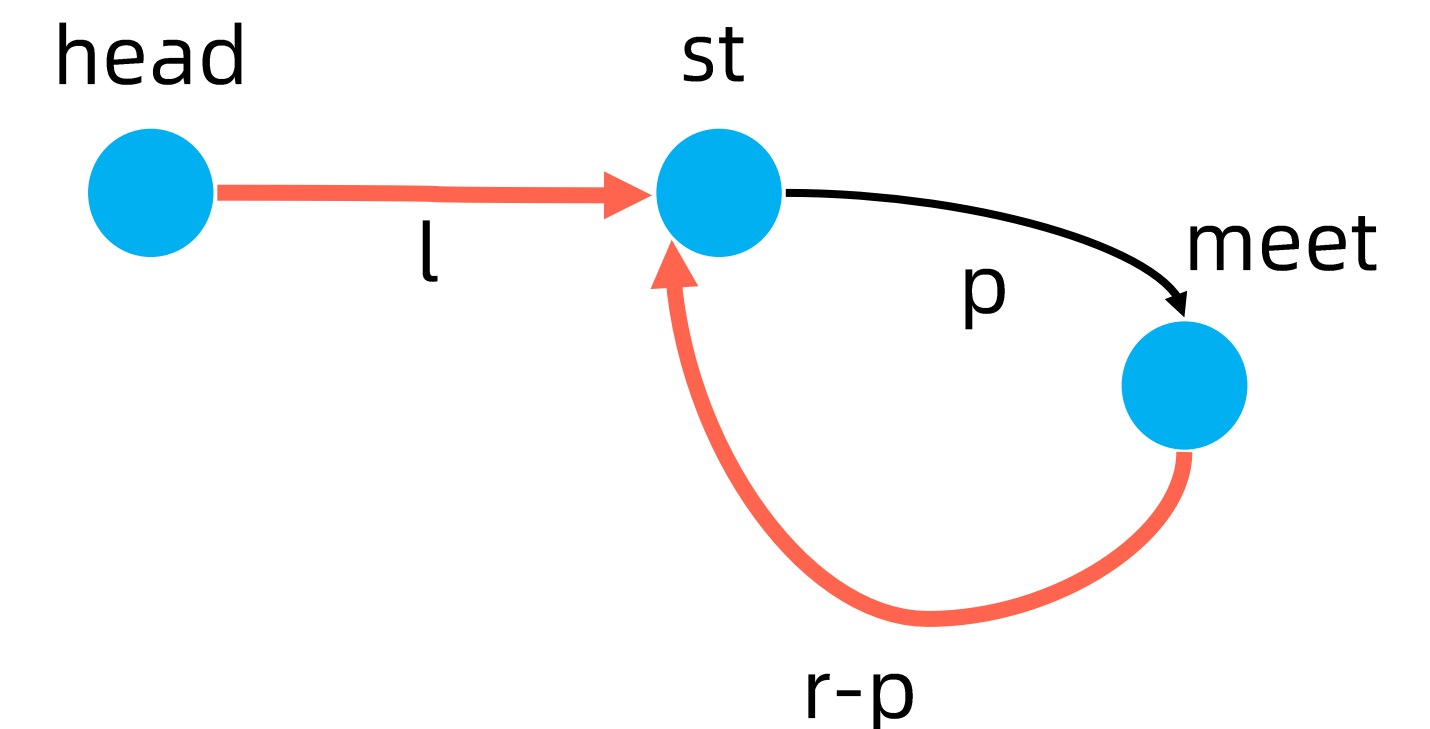
<https://leetcode-cn.com/problems/linked-list-cycle/>

- 快慢指针法， $O(\text{length})$  时间， $O(1)$  空间
- 有环必定发生套圈（快慢指针相遇），无环不会发生套圈（快指针到达null）

## 环形链表 II

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

- 相遇时，有  $2(l + p) = l + p + k * r$ ，其中  $k$  为整数（套的圈数）
- 即  $l = k * r - p = (k - 1) * r + (r - p)$
- 含义：从 head 走到 st，等于从 meet 走到 st，然后再绕几圈
- 此时开始让慢指针与 head 同时移动，必定在环的起始点相遇



# Homework

合并两个有序链表

<https://leetcode-cn.com/problems/merge-two-sorted-lists/>

加一

<https://leetcode-cn.com/problems/plus-one/>



# THANKS

 极客时间 | 训练营