

# 极客时间算法训练营

## 第一课

### 数组、链表、栈、队列

李煜东

《算法竞赛进阶指南》作者



# 目录

1. 数组、链表原理讲解、实战应用
2. 栈、队列及其常见变形、实战应用



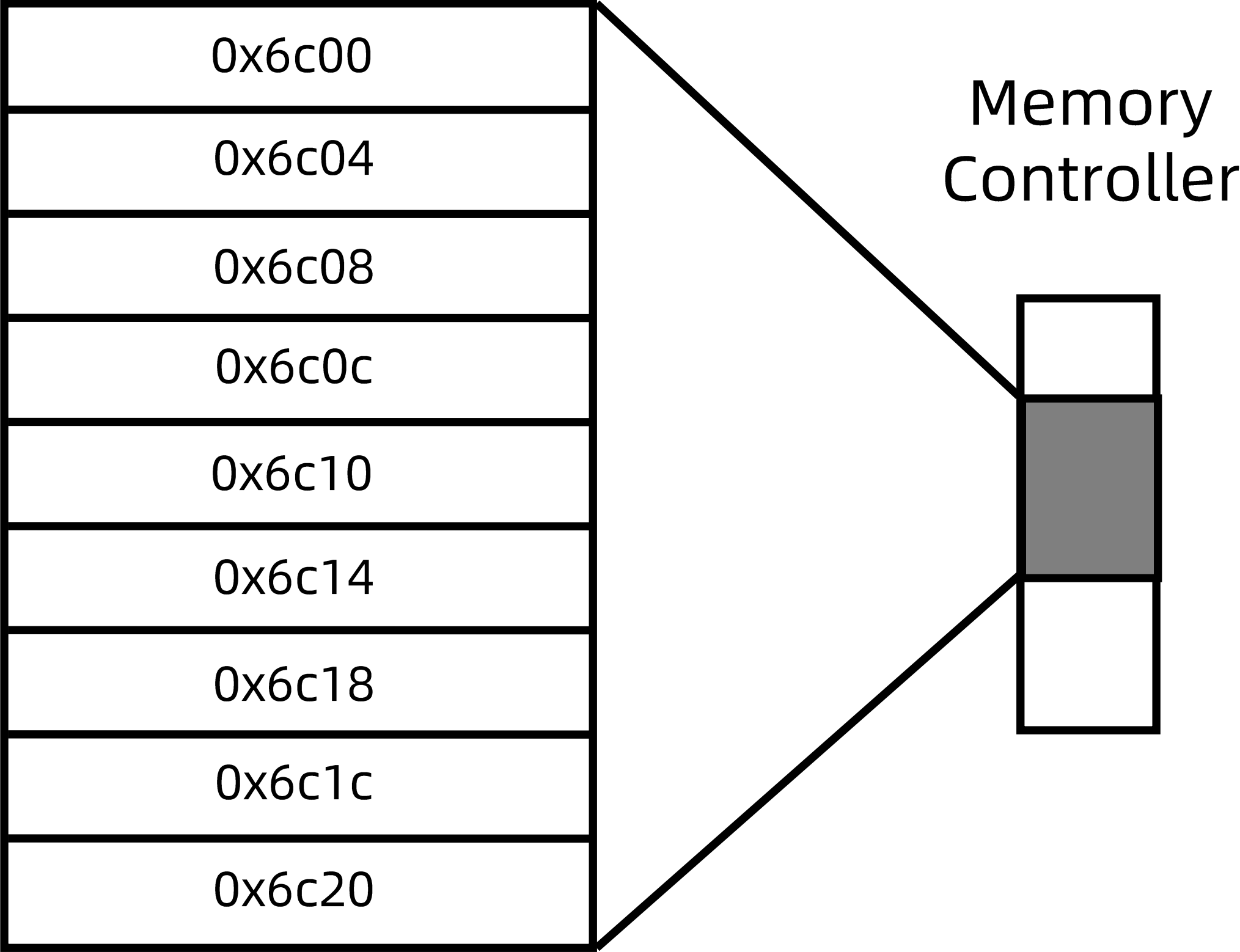
# 数组、链表原理讲解、实战应用

# 数组 (array)

- C++, Java: `int a[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	

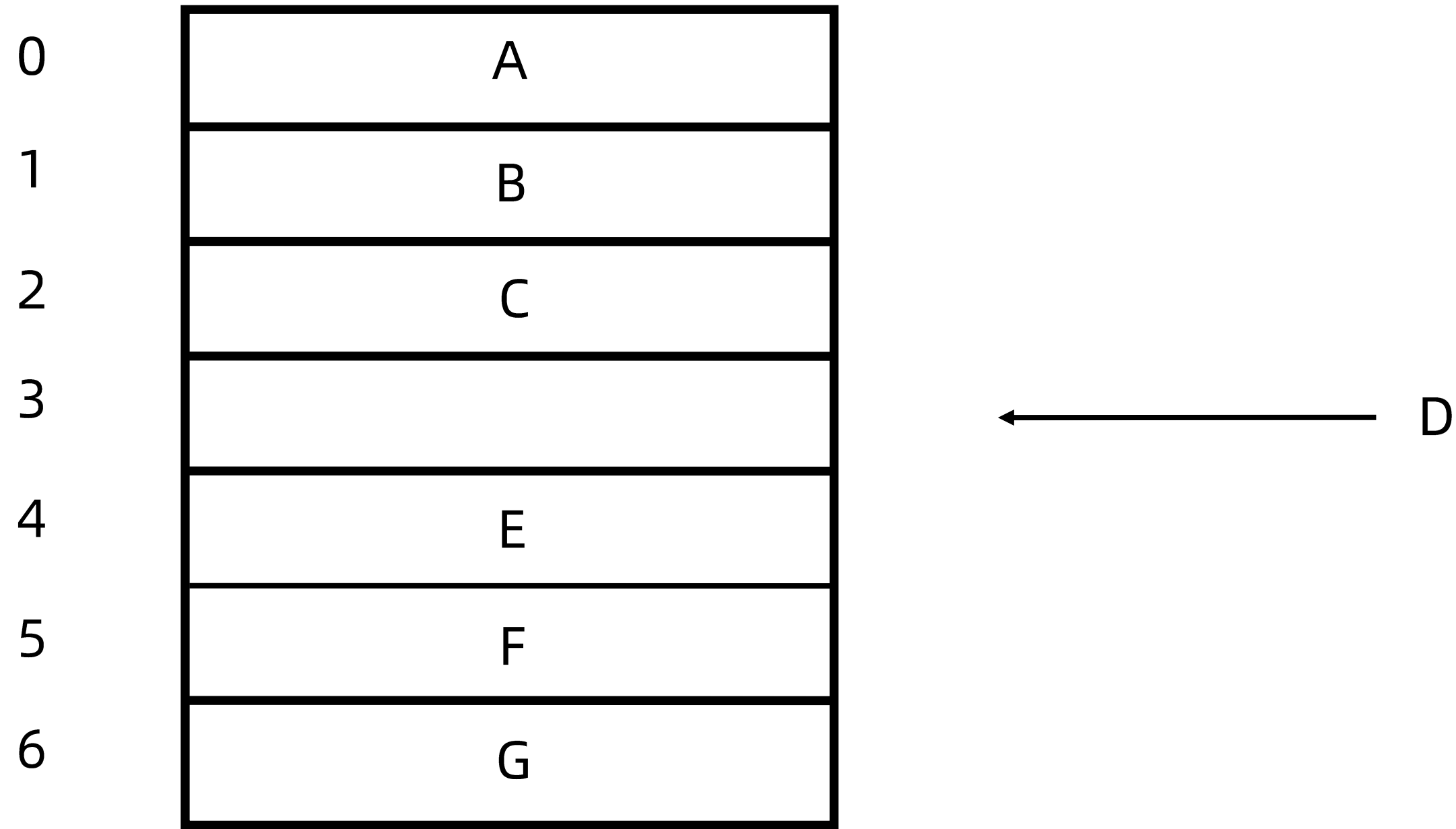
# 数组 - 插入元素

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

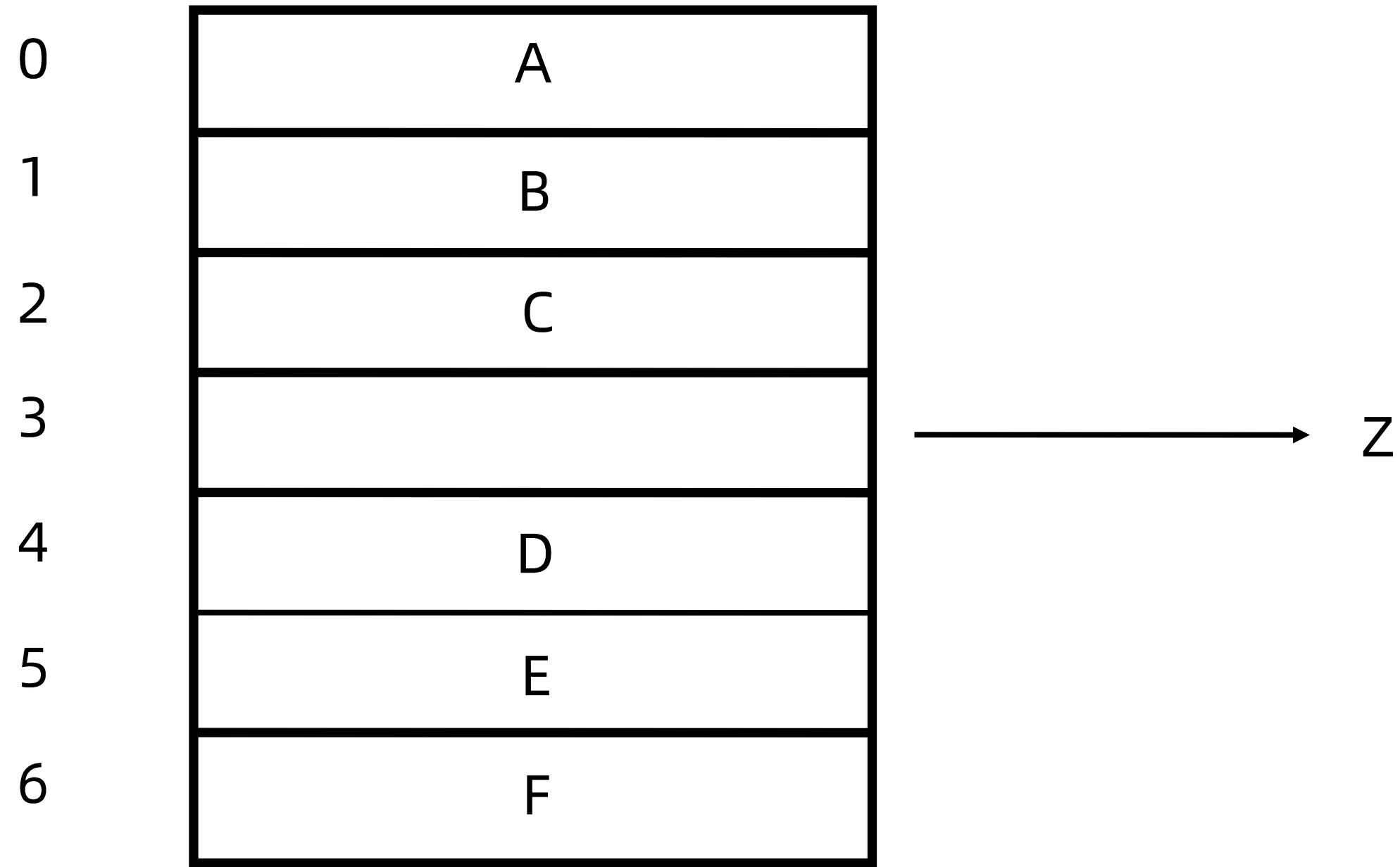
# 数组 - 删除元素

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)  $O(n)$



# 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?

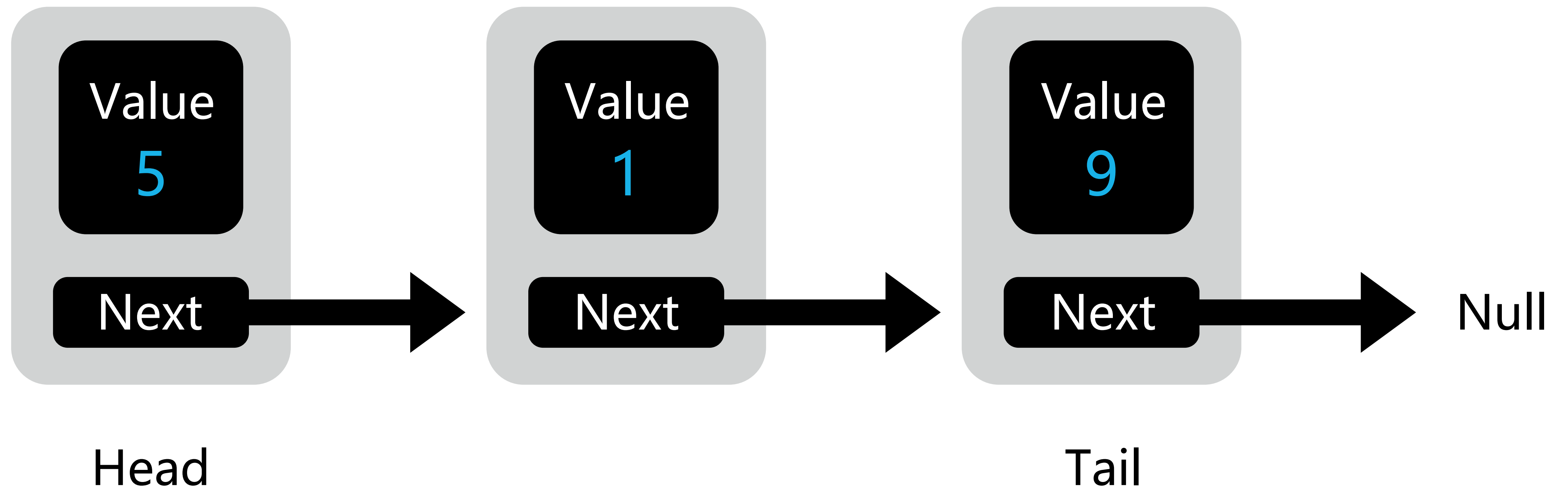
# 变长数组 (resizable array)

- 一个简易的实现方法
- 初始：空数组，分配常数空间
- 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  $(1-2*0.25)n=0.5n$  次
- 思考：若释放空间的阈值设定为 50%，会发生什么情况？

# 实战

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

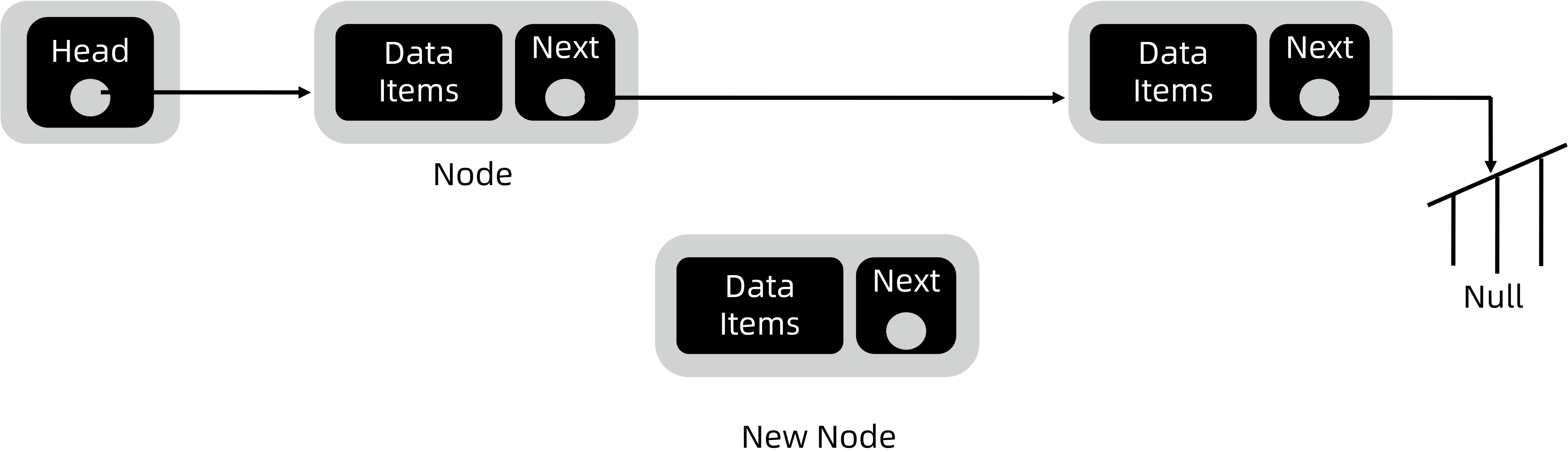
# 单链表 (linked list)



# 单链表 - 插入

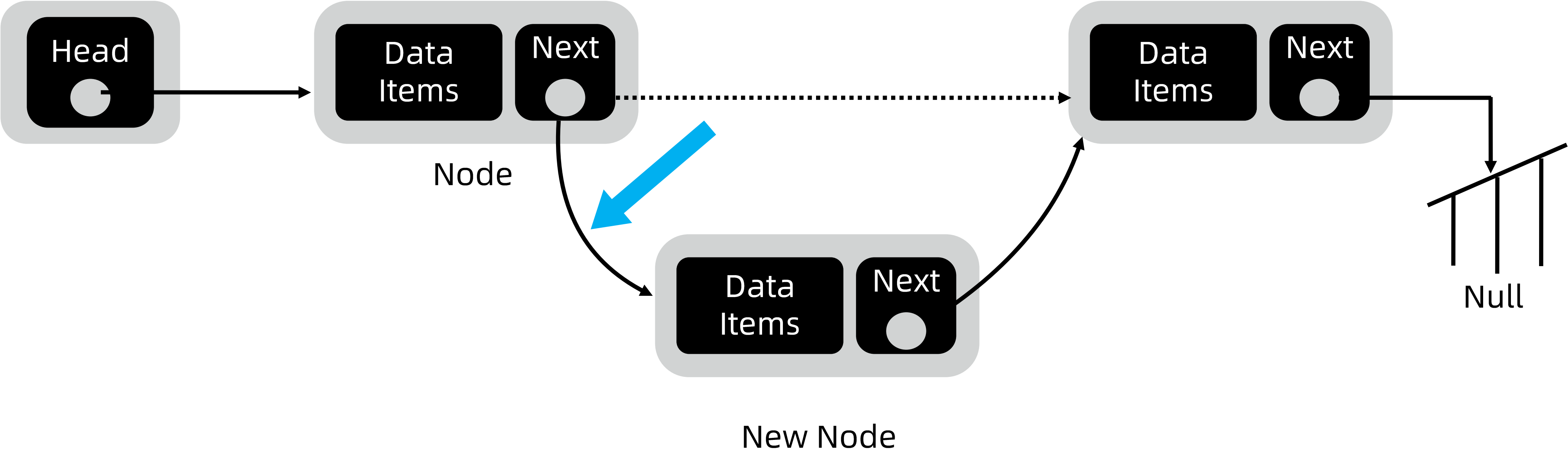


# 单链表 - 插入

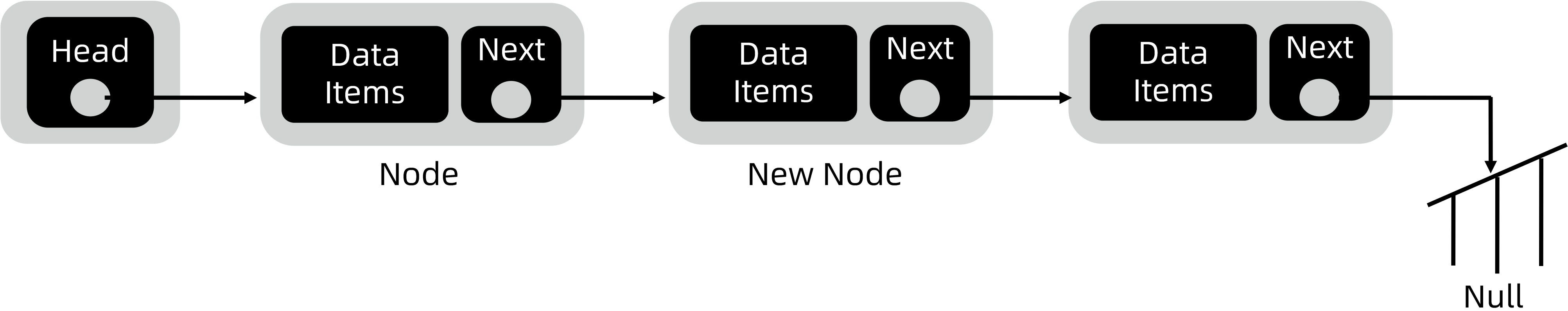




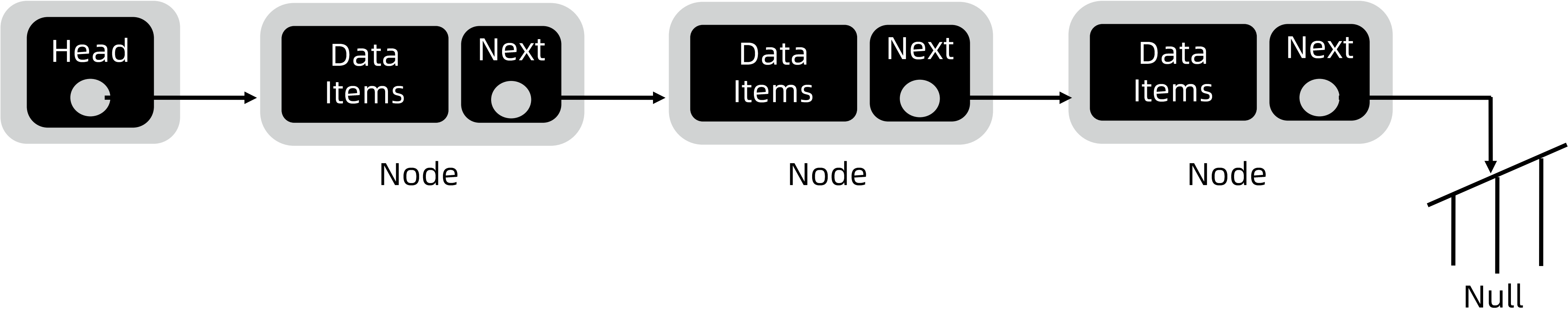
# 单链表 - 插入



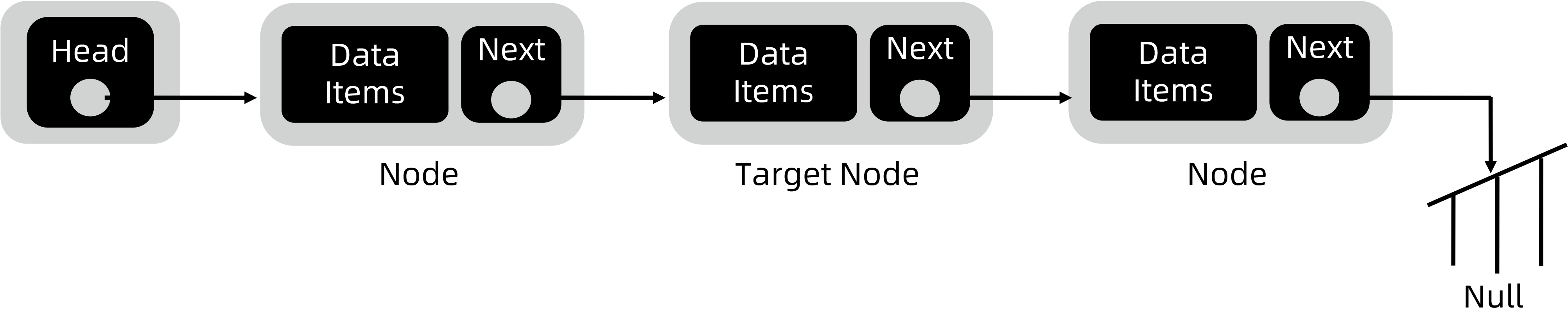
# 单链表 - 插入



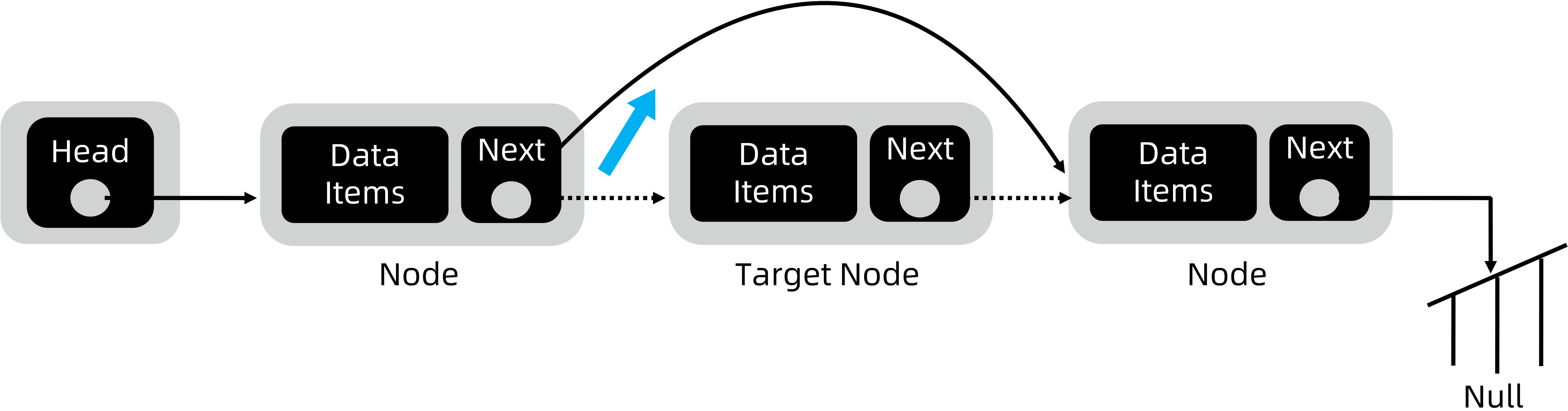
# 单链表 - 删除



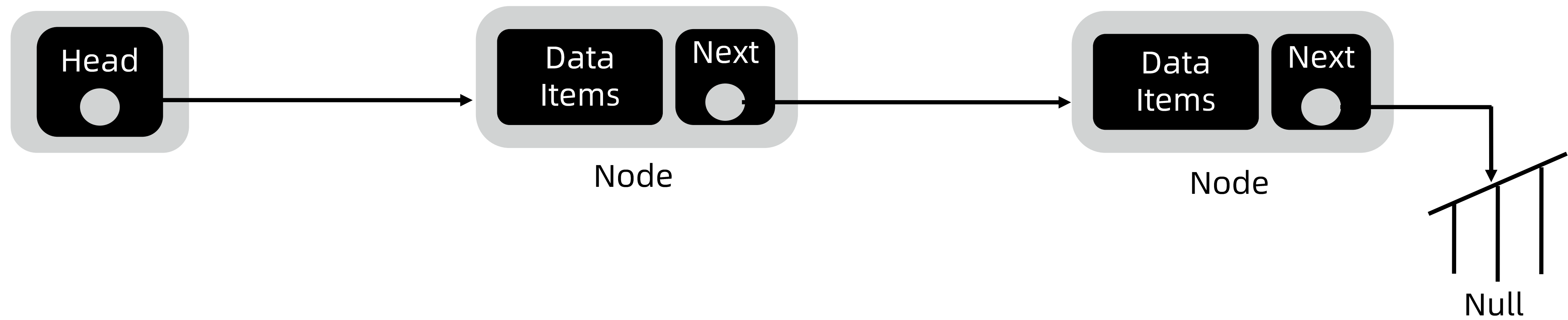
# 单链表 - 删除



# 单链表 - 删除

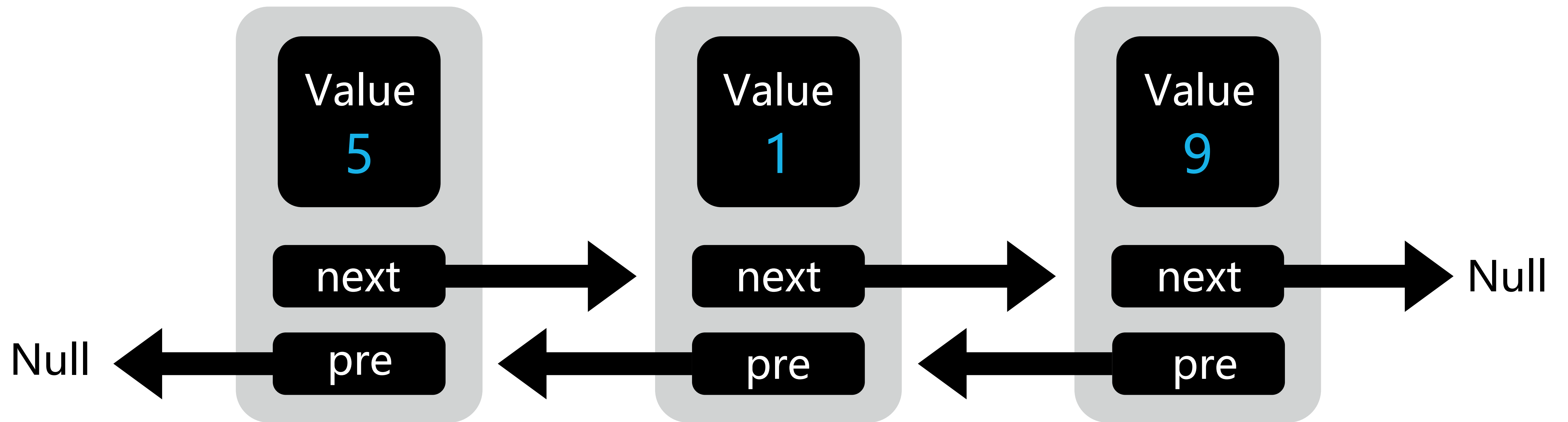


# 单链表 - 删除





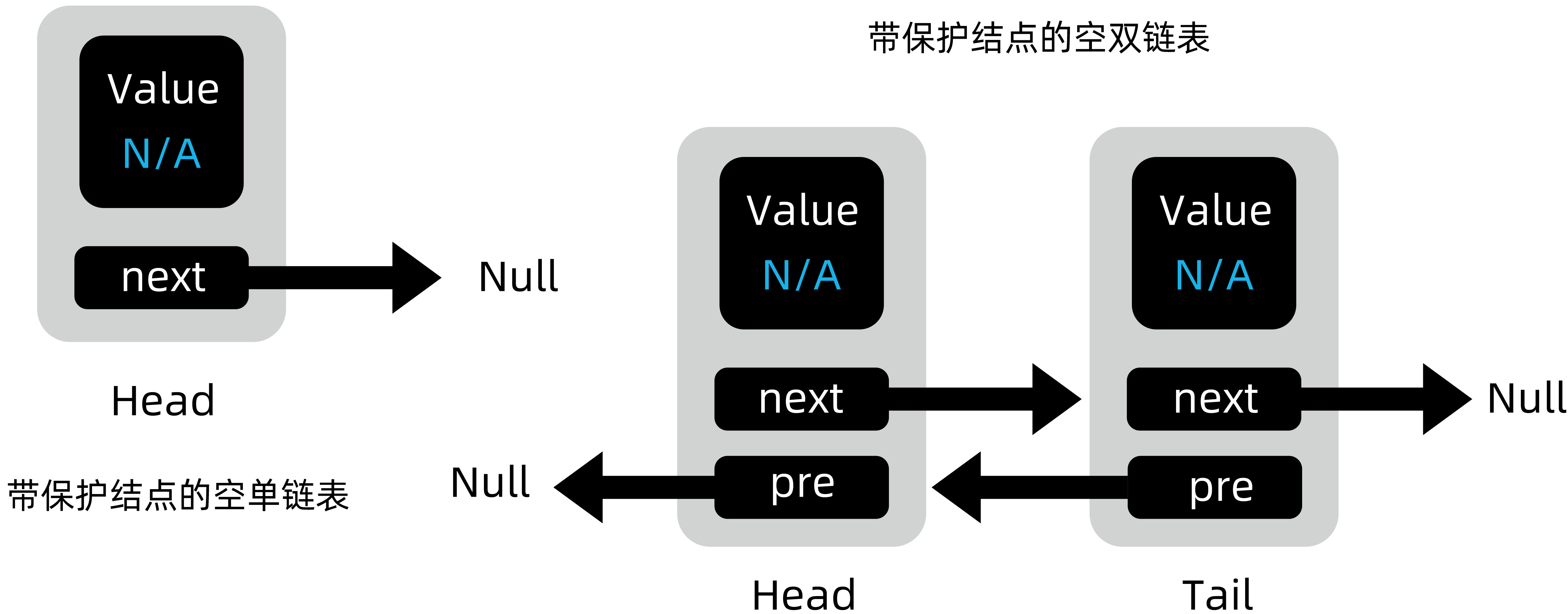
# 双链表 (double linked list)



# 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

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

# 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

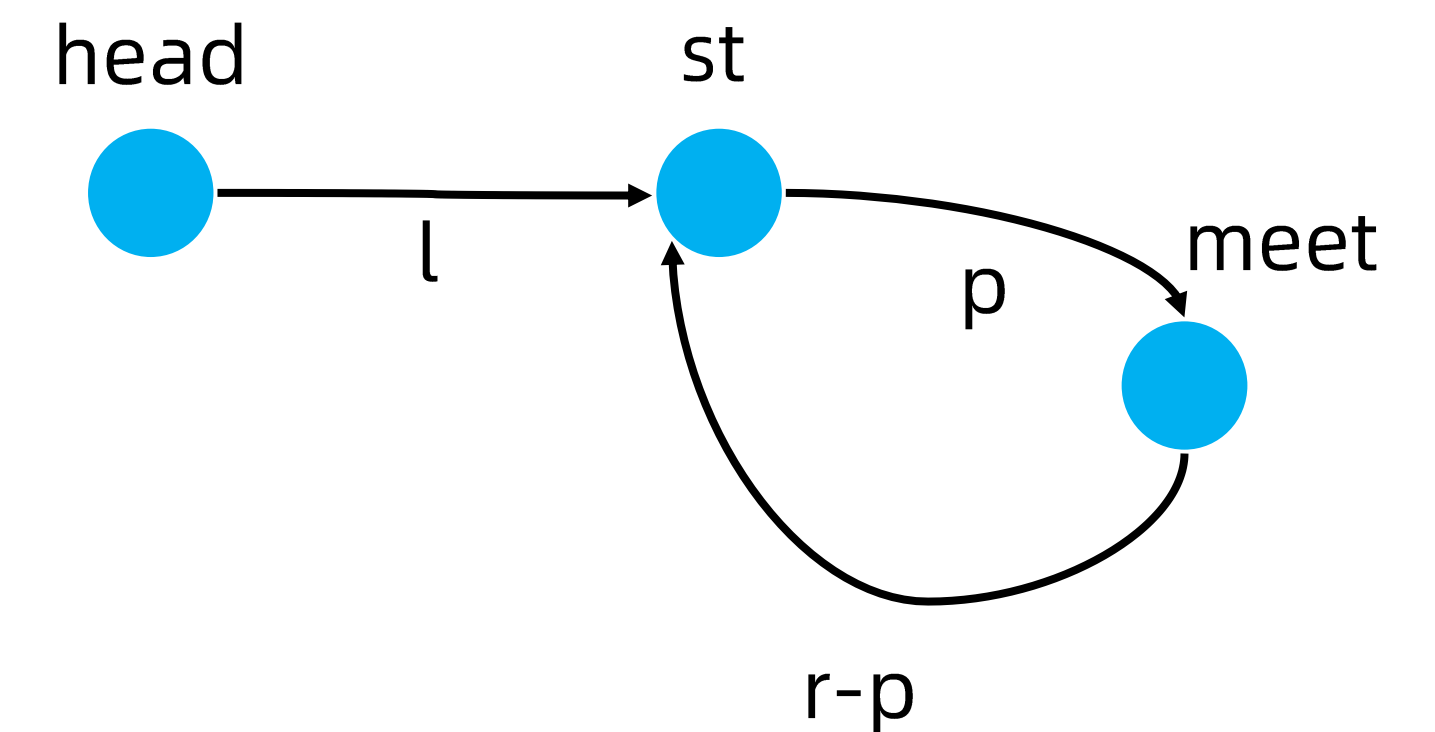
## 环形链表

<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

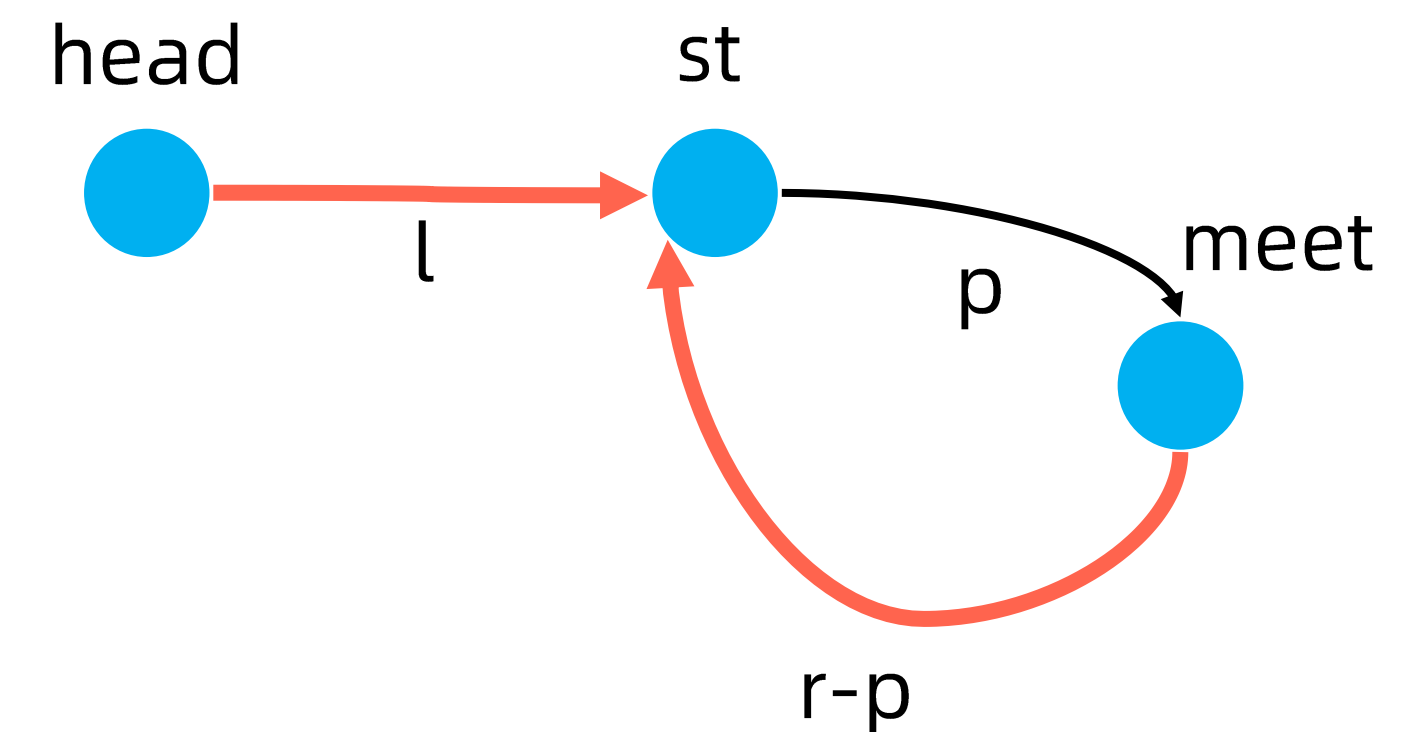
<https://leetcode-cn.com/problems/linked-list-cycle/>

- 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

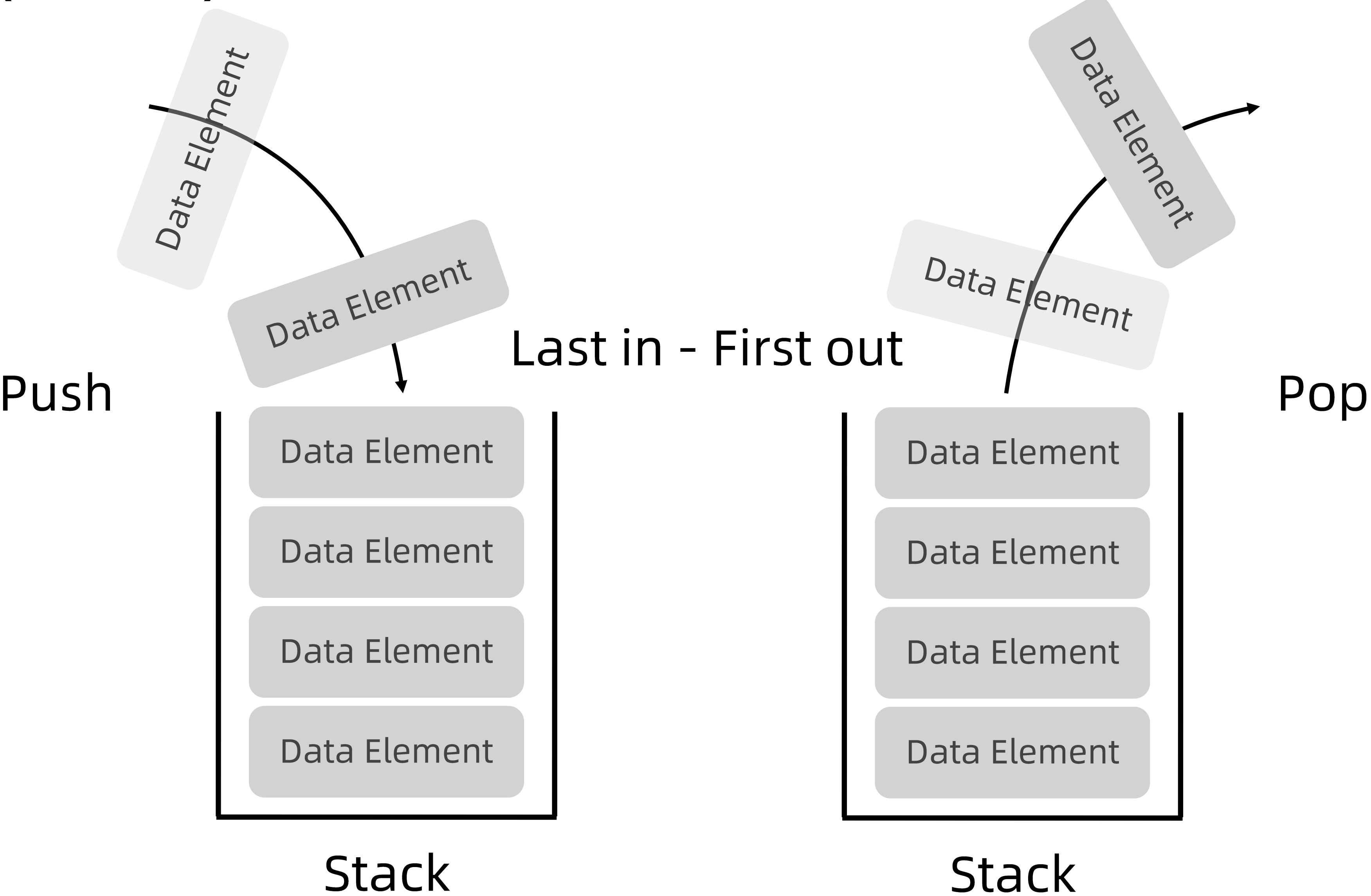
<https://leetcode-cn.com/problems/linked-list-cycle-ii/>

- 相遇时，有  $2(l + p) = l + p + k * r$ ，其中  $k$  为整数（套的圈数）
- 即  $l = r - p \pmod r$
- 含义：从 head 走到 st，与从 meet 走到 st，在模环长同余系中相等
- 此时开始让慢指针与 head 同时移动，必定在环的起始点相遇

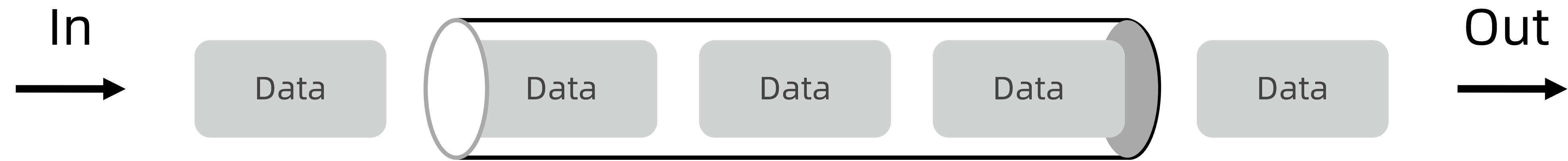


# 栈、队列及其常见变形、实战应用

# 栈 (stack)



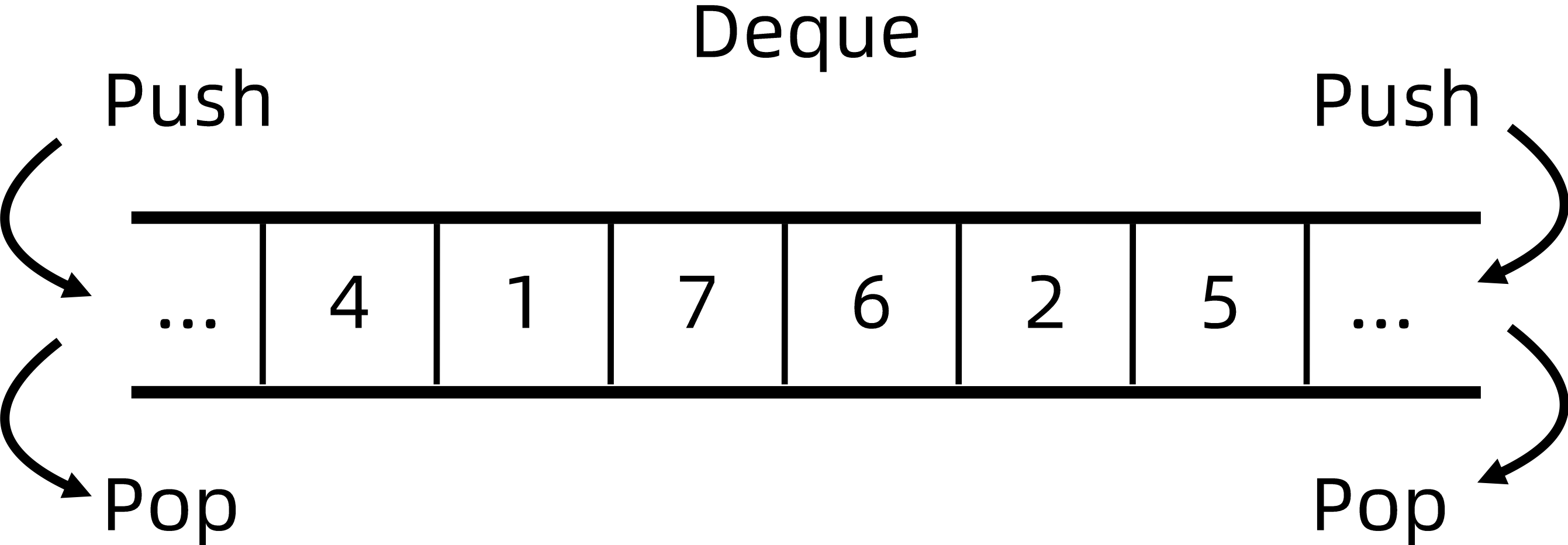
# 队列 (queue)



Last in - Last out

First in - First out

# 双端队列 (deque)



# 优先队列 (priority queue)

一般的队列是以“时间”为顺序的（先进先出）

优先队列按照元素的“优先级”取出

- “优先级”可以是自己定义的一个元素属性

许多数据结构都可以用来实现优先队列，例如二叉堆、二叉平衡树等



# 时间复杂度

## 栈、队列

- Push（入栈、入队）： $O(1)$
- Pop（出栈、出队）： $O(1)$
- Access（访问栈顶、访问队头）： $O(1)$

## 双端队列

- 队头、队尾的插入、删除、访问也都是  $O(1)$

## 优先队列

- 访问最值： $O(1)$
- 插入：一般是  $O(\log N)$ ，一些高级数据结构可以做到  $O(1)$
- 取最值： $O(\log N)$

# Homework

读文档，学习使用自己的语言中所带的栈、队列、双端队列、优先队列 library

C++: stack, queue, deque, priority\_queue

Java

- Stack
- Queue, Deque 可以用 LinkedList 实现
- PriorityQueue

Python

- 栈、队列、双端队列可以用 list 实现
- 优先队列可以用 heapq 库

# 实战

有效的括号

<https://leetcode-cn.com/problems/valid-parentheses/>

- 栈与“括号序列”
- “最近相关性”

<https://leetcode-cn.com/problems/min-stack/>

- 前缀最小值

# 表达式求值

## 前缀表达式

- 形如 “op A B” , 其中op是一个运算符, A,B是另外两个前缀表达式
- 例如:  $* 3 + 1 2$
- 又称波兰式

## 后缀表达式

- 形如 “A B op”
- $1 2 + 3 *$
- 又称逆波兰式

## 中缀表达式

- $3 * (1 + 2)$

# 后缀表达式求值

<https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/>

建立一个用于存数的栈，逐一扫描后缀表达式中的元素。

- 如果遇到一个数，则把该数入栈。
- 如果遇到运算符，就取出栈顶的两个数进行计算，然后把结果入栈。

扫描完成后，栈中恰好剩下一个数，就是该后缀表达式的值。

时间复杂度  $O(n)$

# 中缀表达式求值（选做）

<https://leetcode-cn.com/problems/basic-calculator/>（以及-ii, -iii）

建立一个用于存运算符的栈，逐一扫描中缀表达式中的元素。

- 如果遇到一个数，输出该数。
- 如果遇到左括号，把左括号入栈。
- 如果遇到右括号，不断取出栈顶并输出，直到栈顶为左括号，然后把左括号出栈。
- 如果遇到运算符，只要栈顶符号的优先级  $\geq$  新符号，就不断取出栈顶并输出，最后把新符号进栈。优先级顺序为乘除号  $>$  加减号  $>$  左括号。
- 思考：如何辨别运算符是加减法运算还是正负号？

依次取出并输出栈中的所有剩余符号。

最终输出的序列就是一个与原中缀表达式等价的后缀表达式，再对后缀表达式求值。

时间复杂度  $O(n)$

# 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/>

设计循环双端队列

<https://leetcode-cn.com/problems/design-circular-deque/>



# THANKS

 极客时间 | 训练营