

数据结构与算法实战

第三讲 线性结构

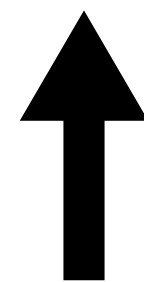
3-1 顺序表

3.1.1 顺序表的操作

- 顺序表：一维数组
- 查找操作：X、Kth、Max (Min)
- 插入操作：在第K位置插入元素
- 删除操作：删除第K位置的元素

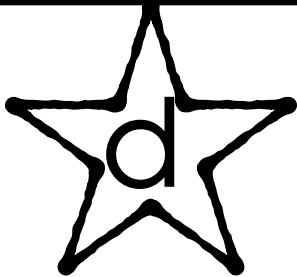
3.1.1 顺序表的操作：插入

0	1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h		



x

3.1.1 顺序表的操作：删除

0	1	2	3	4	5	6	7	8	9
a	b	c		e	f	g	h		

3.1.2 顺序表的操作实现

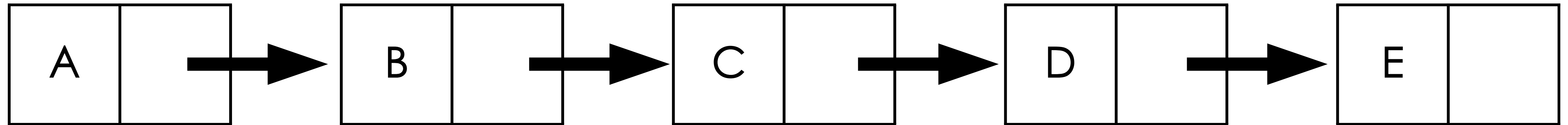
- C语言：数组
- C++：vector
- Java：ArrayList
- Python: List ?

数据结构与算法实战

第三讲 线性结构

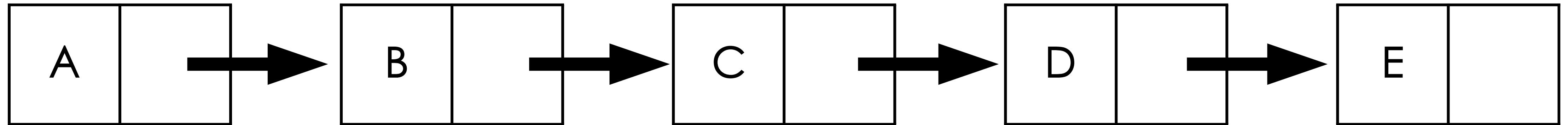
3-2 链表

3.2 链表



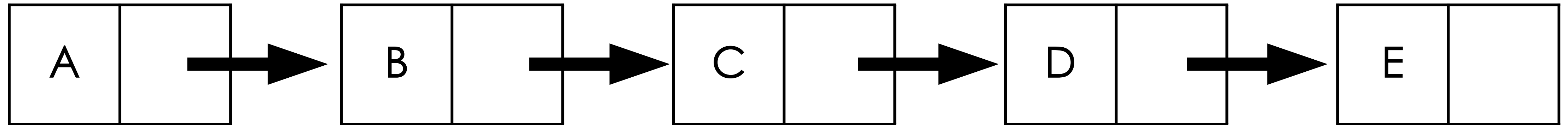
- 为什么要用链表?
- 查找操作: 找 X、Kth
- 插入操作: 在第K位置插入元素
- 删除操作: 删除第K位置的元素
- 性能对比: 顺序表 V.S. 链表

3.2.1 链表求表长、查找



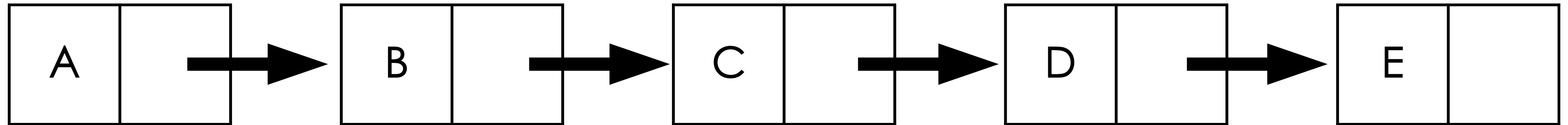
- 求表长（遍历）
- 查找：X Kth

3.2.2 链表插入、删除



- 插入操作： 在第K位置插入元素

3.2.2 链表插入、删除



- 删除操作：删除第K位置的元素

3.2.3 链表的操作实现

- C语言： 结构体+指针
- C++： list
- Java： LinkedList
- Python: List ?

数据结构与算法实战

第三讲 线性结构

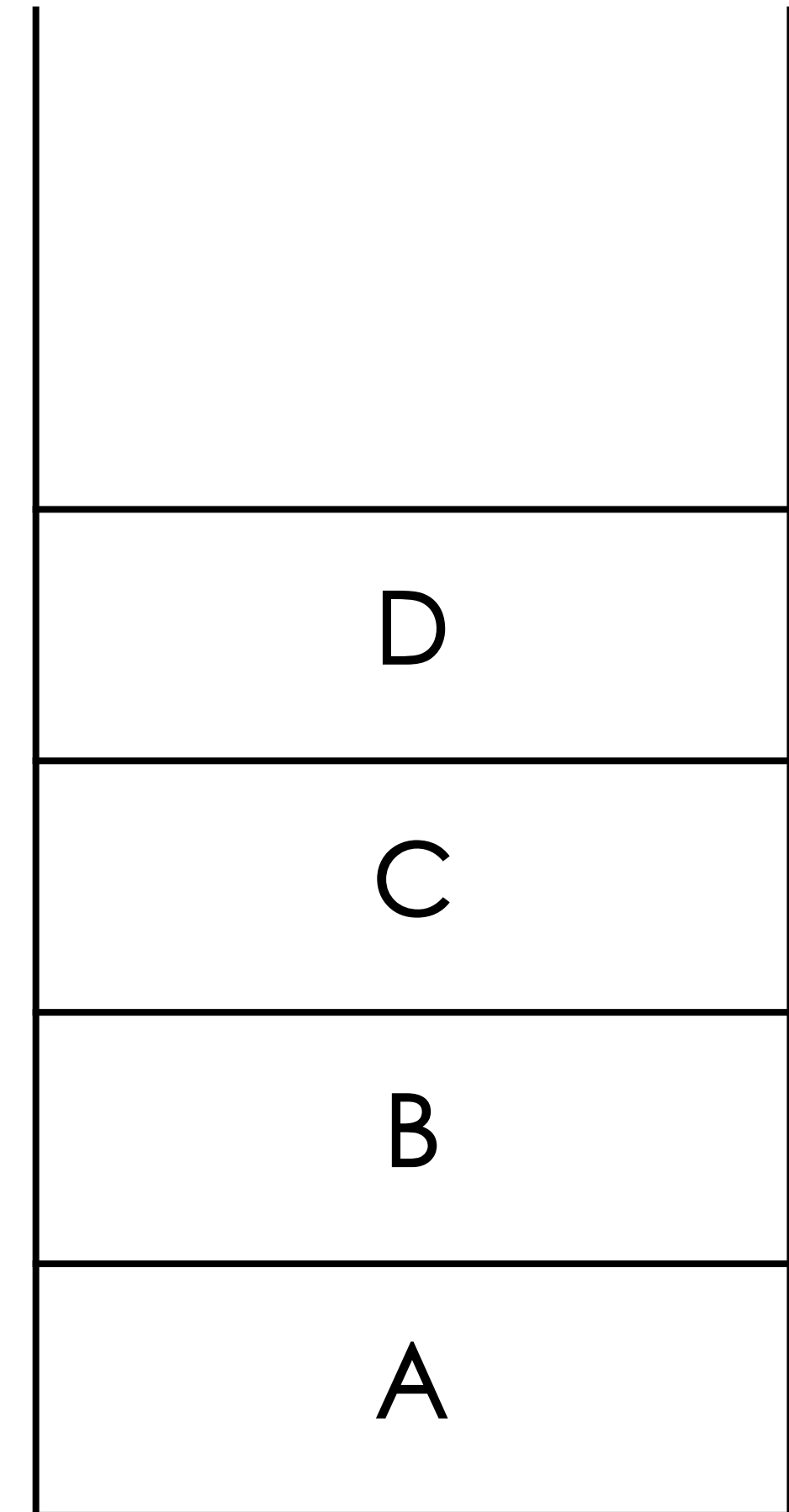
3-3 栈

3.3 栈

- 长啥样：一个桶

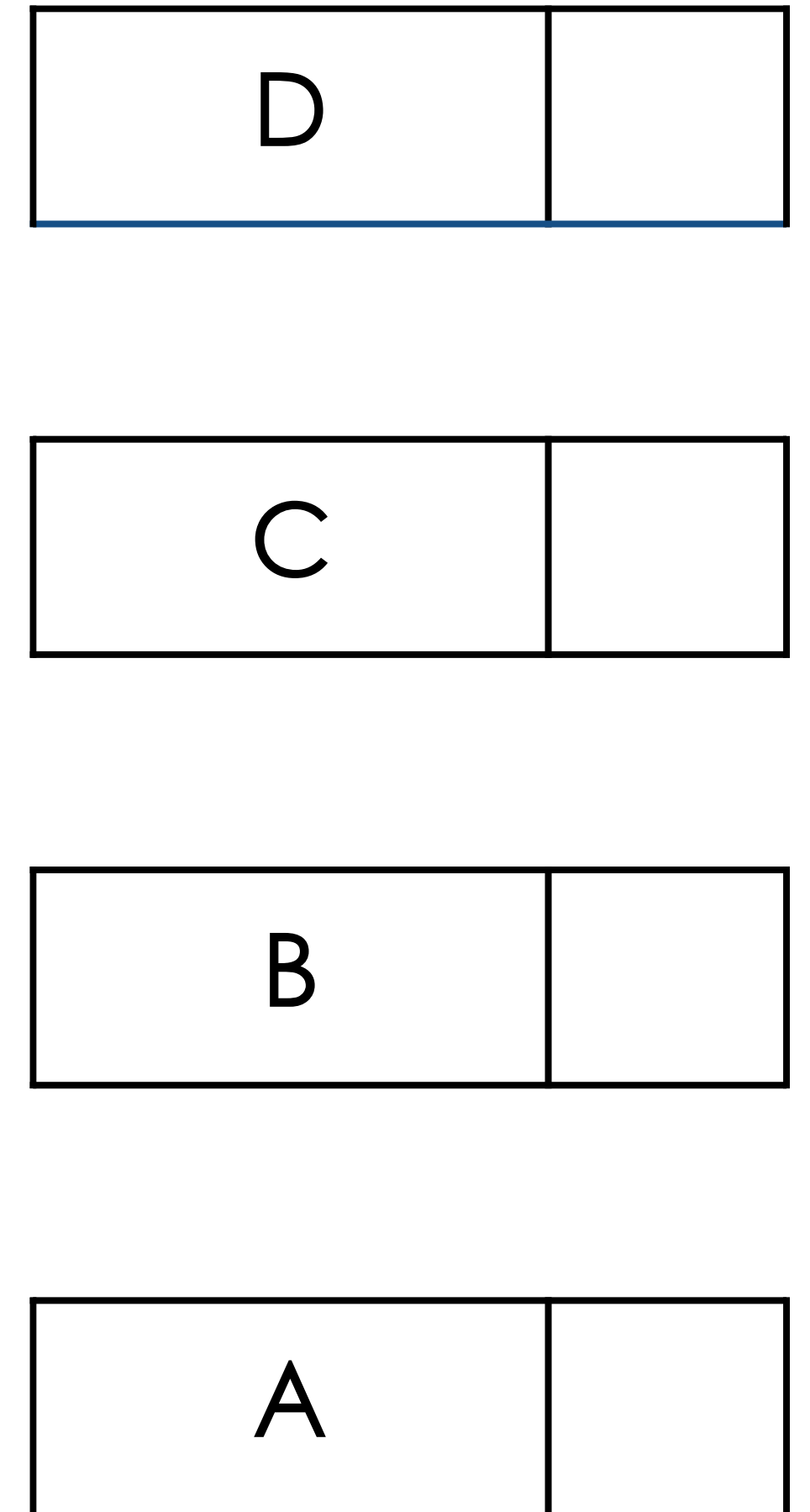
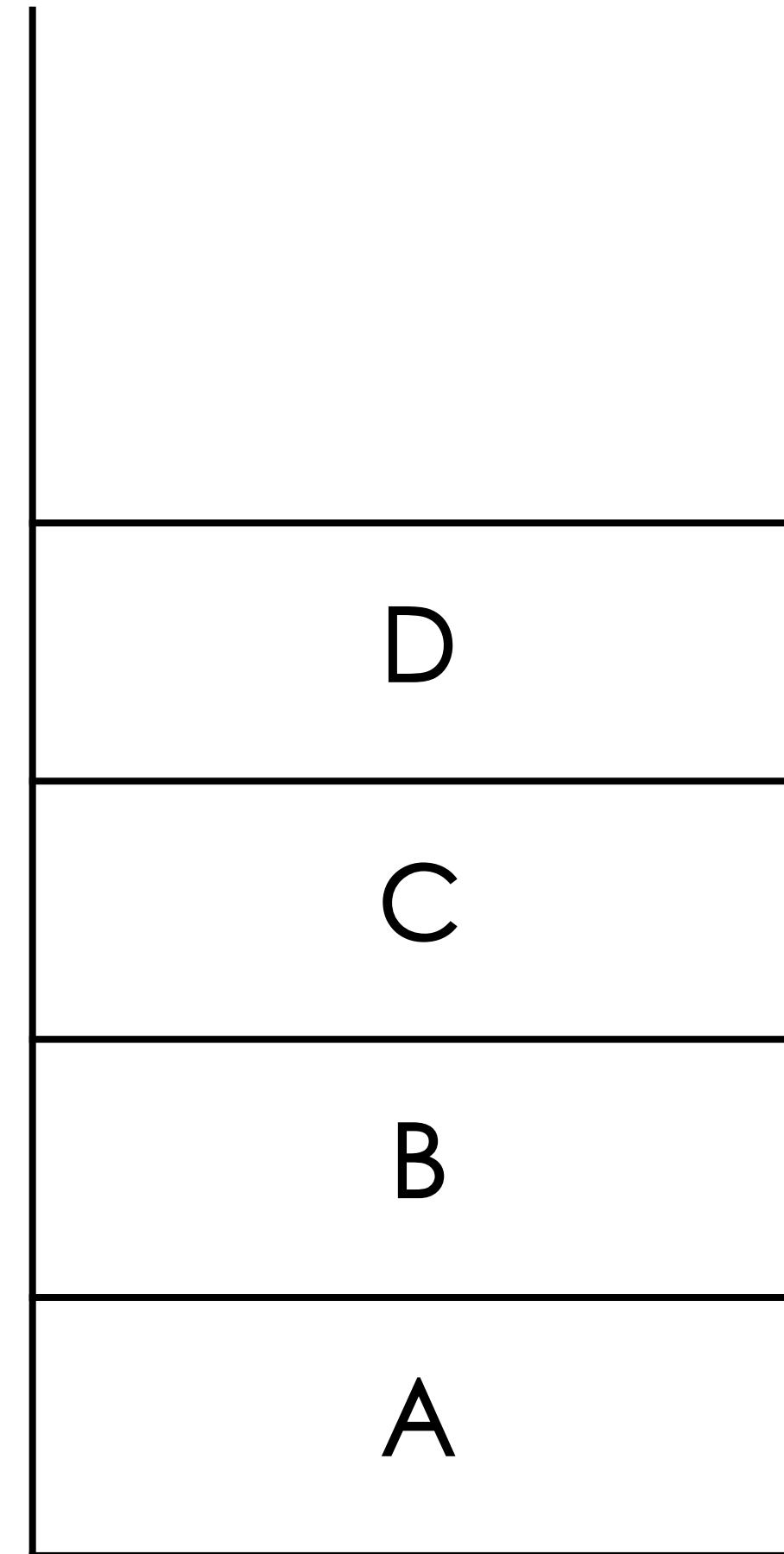
操作（比一般的线性表简单）：

- push
- pop
- top



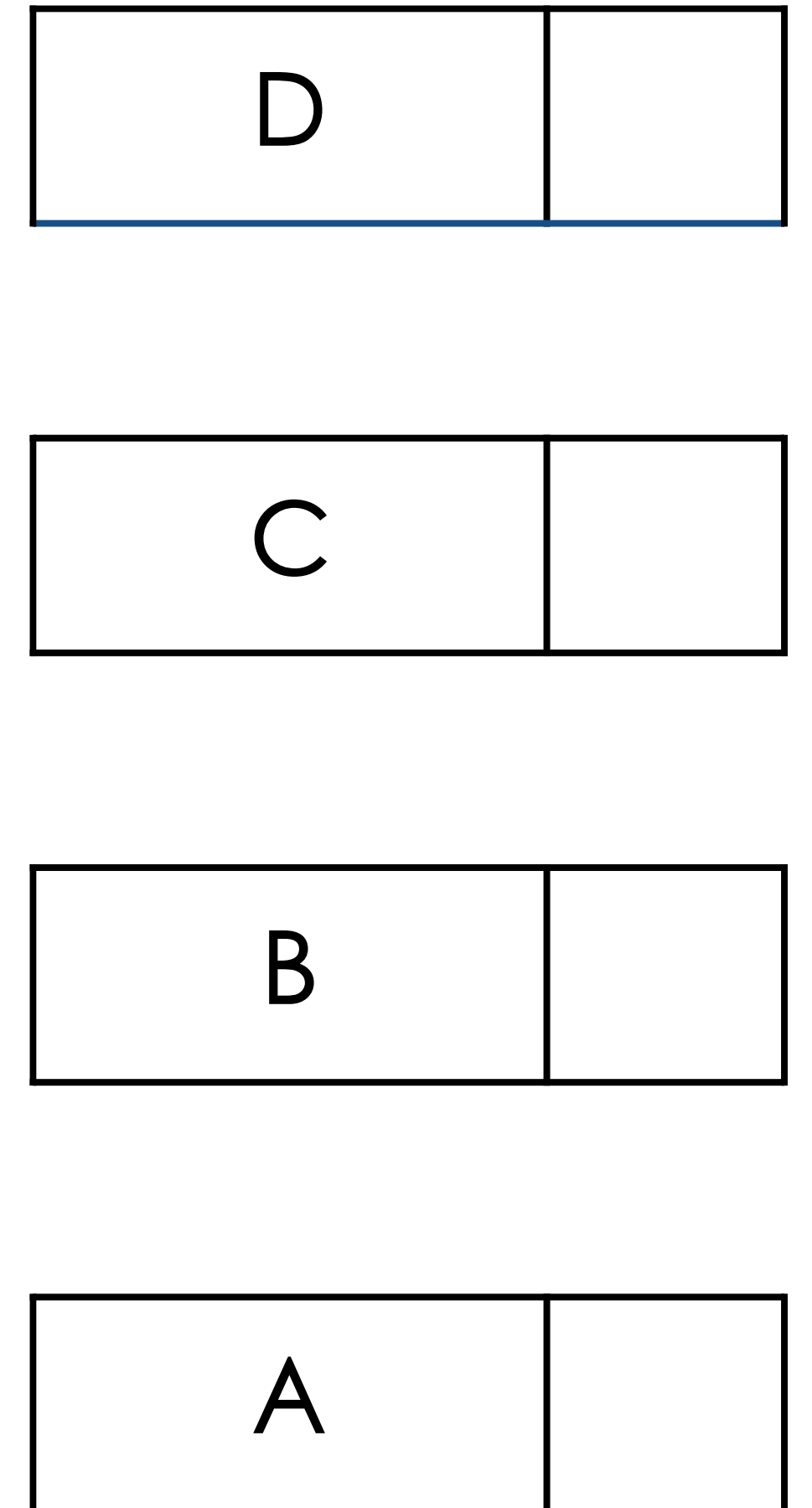
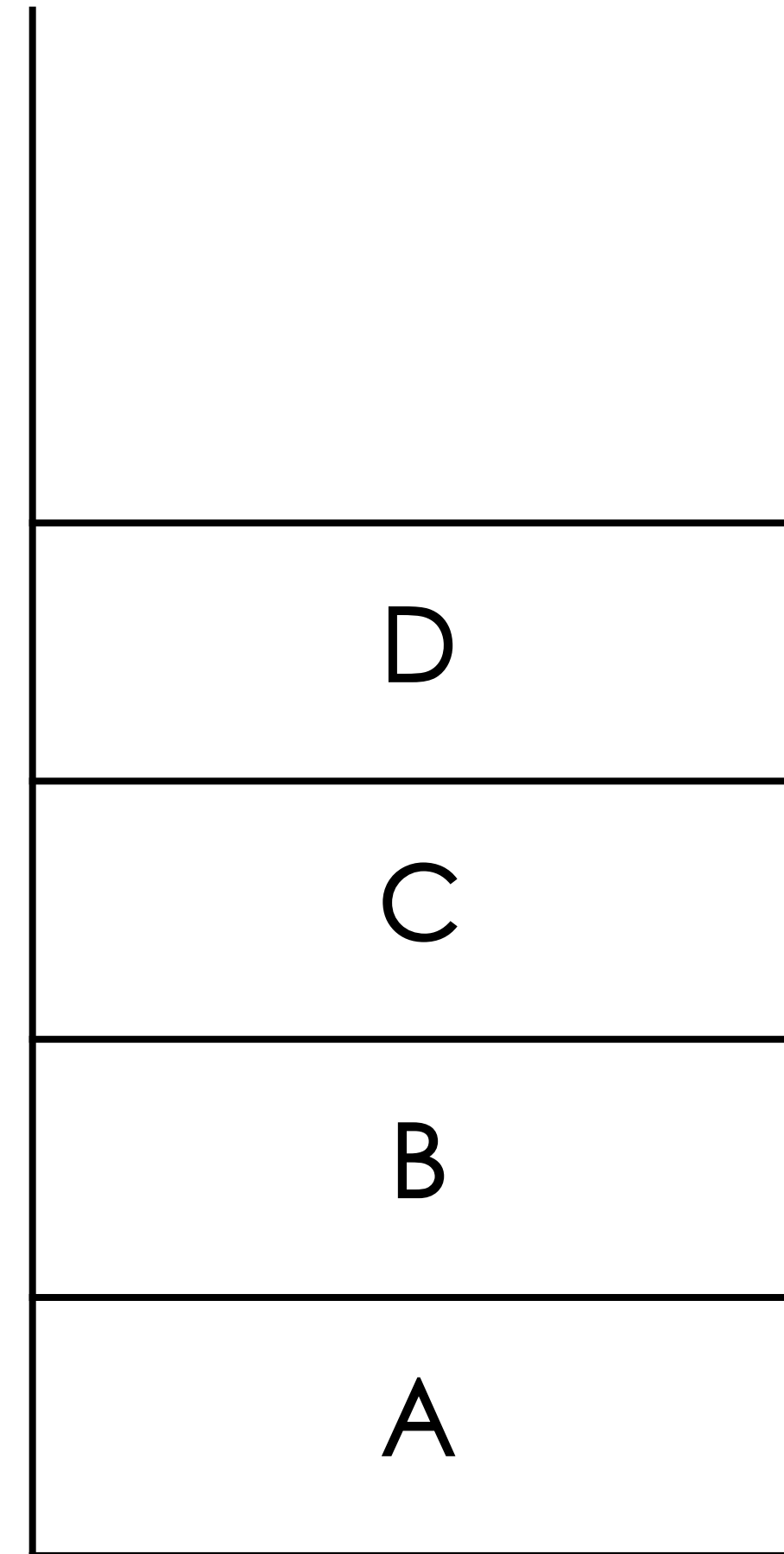
3.3.1 顺序栈和链式栈

- 初始化：空数组 / 空链表
- push：数组?插入 / 链表?插入
- pop：数组?删除 / 链表?删除
- top：看数组? / 链表?



3.3.1 顺序栈和链式栈

- 初始化：空数组 / 空链表
- push：数组尾插入 / 链表头插入
- pop：数组尾删除 / 链表头删除
- top：看数组尾 / 链表头



3.3.2 栈的实现

- C: 数组 / 链表
- C++: stack
- Java: Stack
- Python: list

3.3.3 实例：后缀式求值

- 中缀式： $3 + 5 * 2$
- 后缀式（逆波兰式）： $3\ 5\ 2\ *\ +$
- 前缀式（波兰式）： $+ 3\ * 5\ 2$
- 哪种适合人类计算？
- 哪种适合计算机求值？

3.3.3 实例：后缀式求值

- 中缀式： $3 + 5 * (2 - 1)$
- 后缀式（逆波兰式）： $3\ 5\ 2\ 1\ -\ *\ +$

3.3.3 实例：后缀式求值

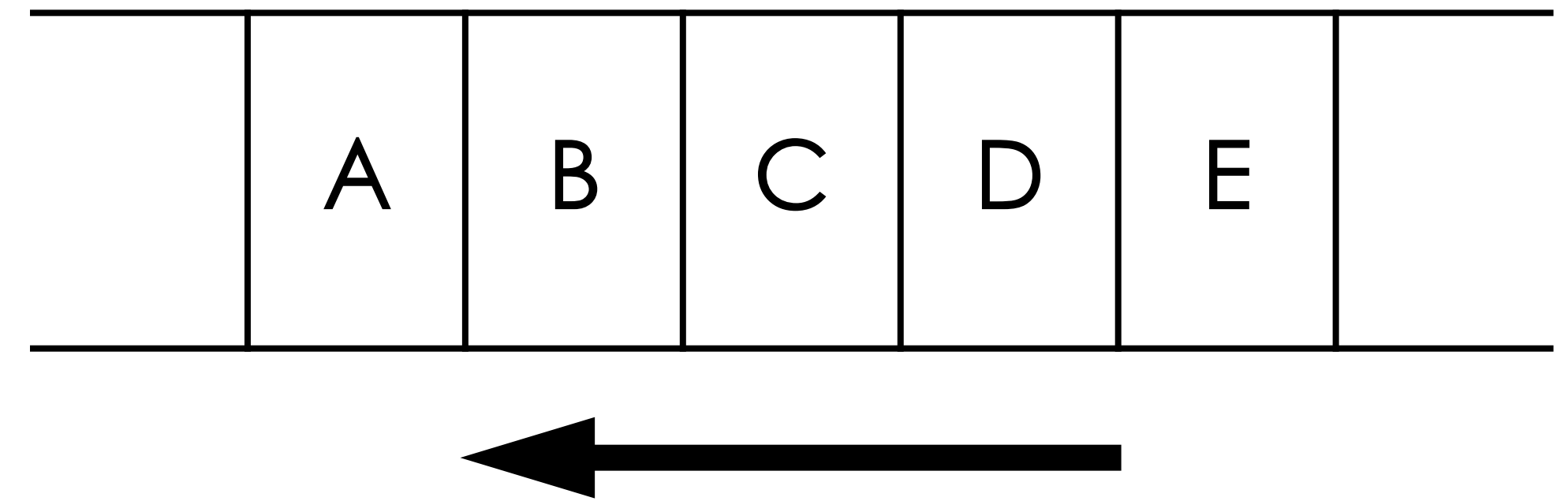
- 后缀式（逆波兰式）：3 5 2 1 - * + 思路？

3.4 队列

- 长啥样：一个桶、没底、横过来

操作（比一般的线性表简单）：

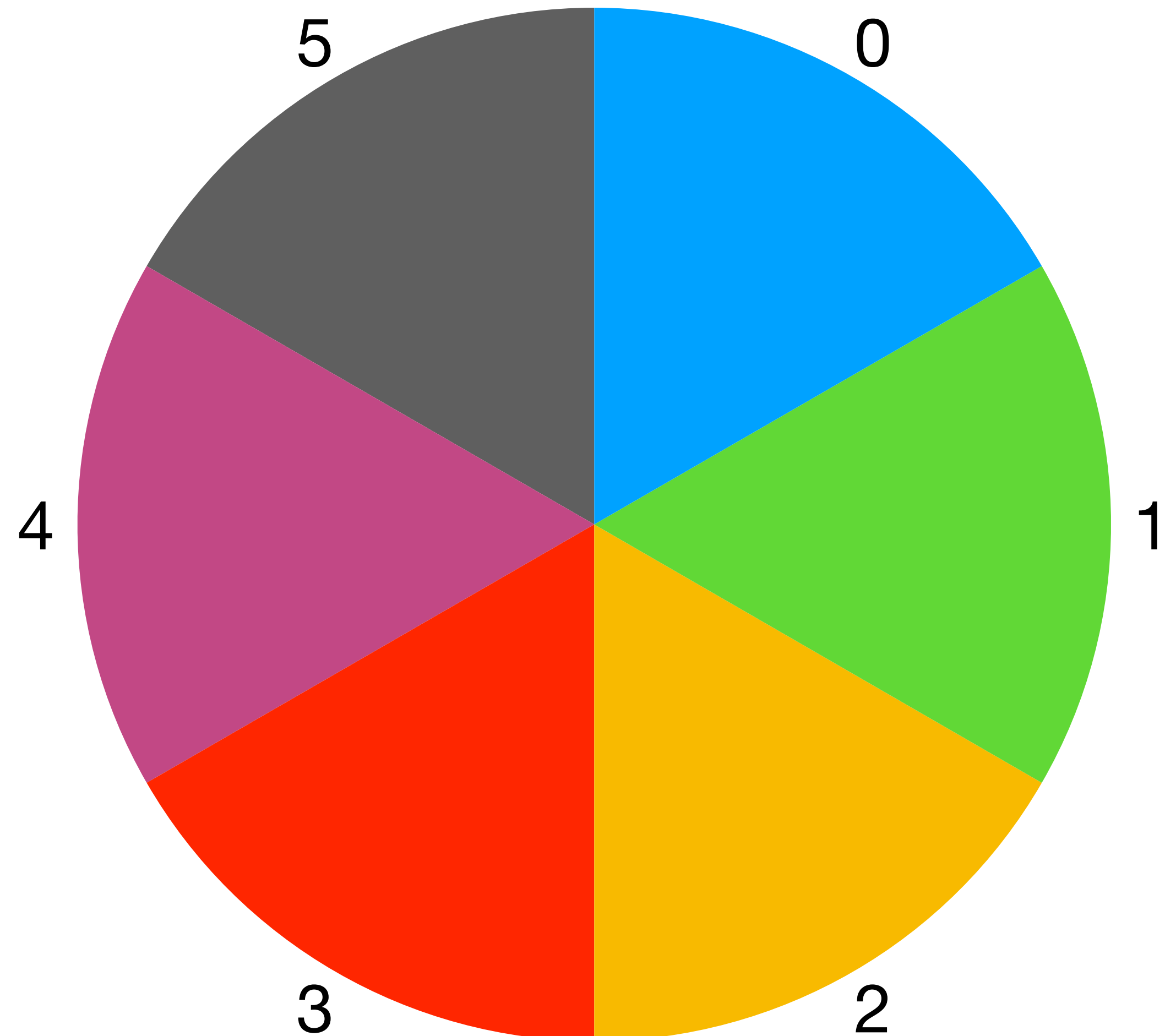
- enQueue
- deQueue
- Front



3.4.1 循环队列和链式队列

循环队列：

- 为什么用循环队列？
- 如何初始化？
- 如何判断队空？
- 如何判断队满？



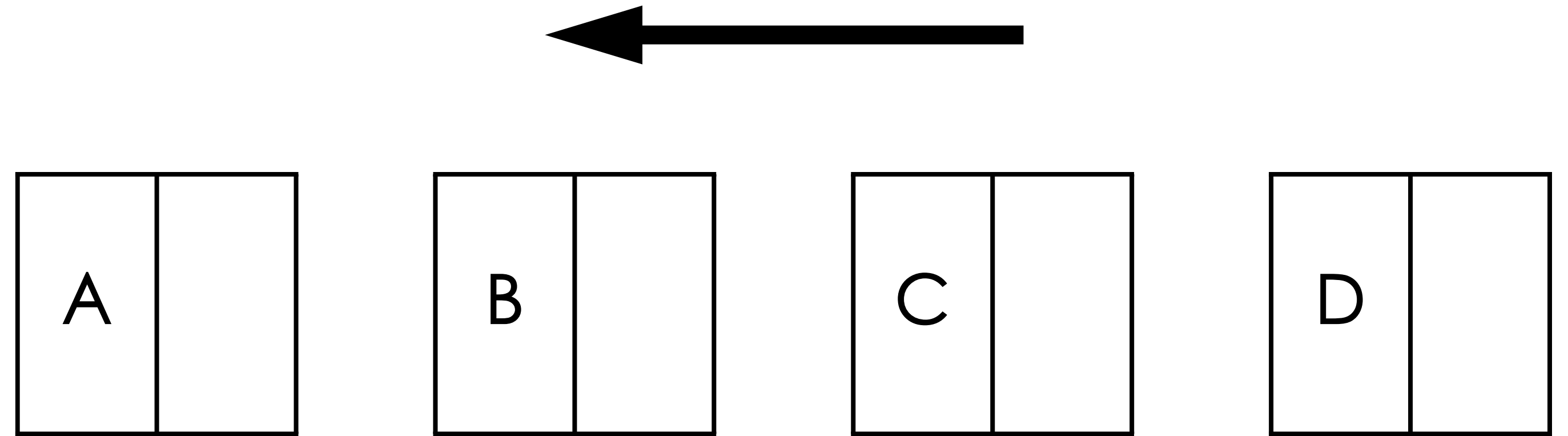
3.4.1 循环队列和链式队列



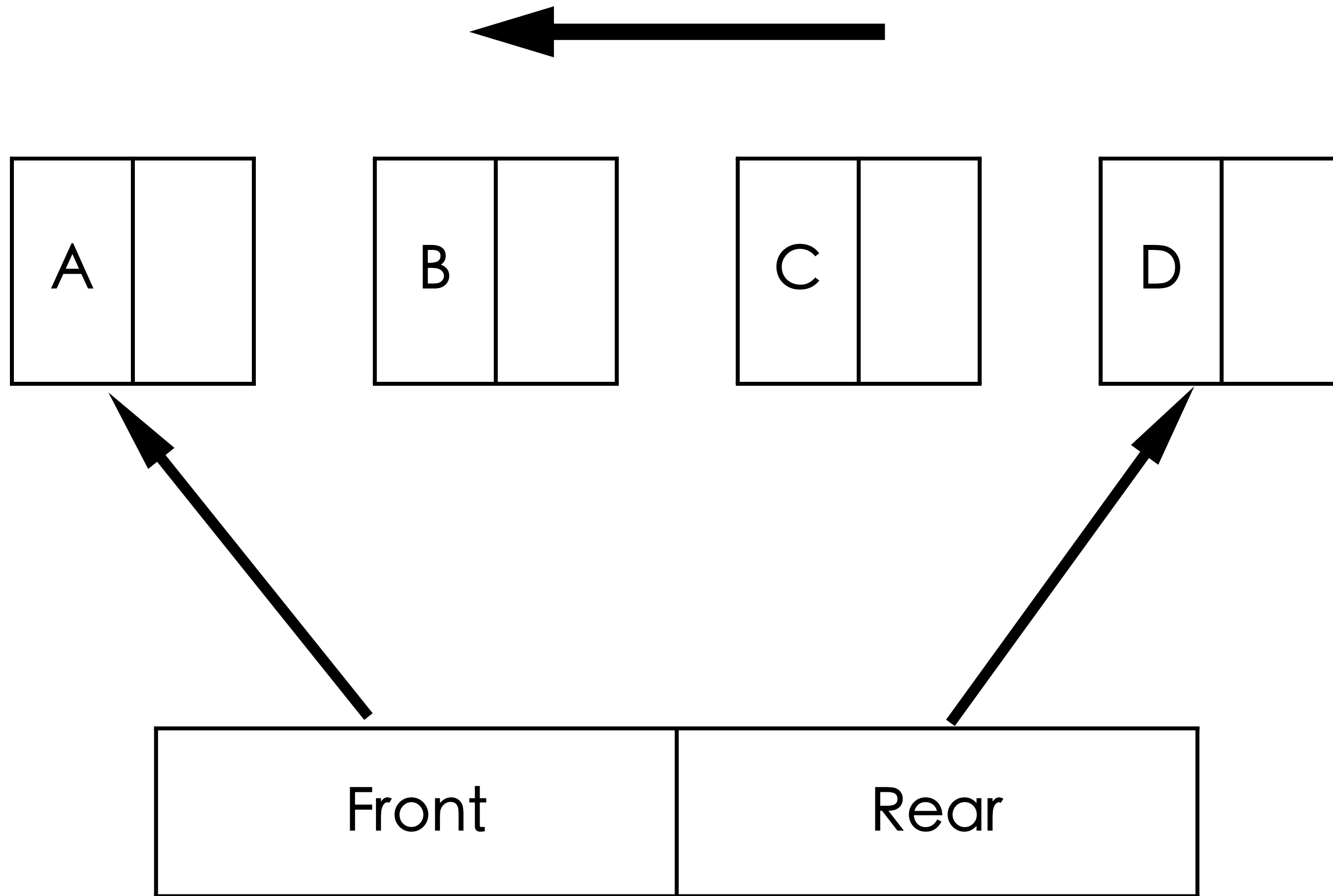
3.4.1 循环队列和链式队列

链式队列：

- 为什么用链式队列？
- 如何初始化？
- 如何判断队空？
- 如何判断队满？



3.4.1 循环队列和链式队列



3.4.2 队列的实现

- C: 数组 / 链表
- C++: queue
- Java: Queue ArrayDeque
- Python: list

3.4.3 实例： 银行排队

[◀ 返回](#)

7-3 简单模拟单队列排队 (20 分)

用程序简单模拟一个单队列多窗口的排队模式：

设某银行有一个固定能容纳N个顾客的等候区，顾客想进银行，若等候区有空则可进，否则被拒绝进入。

每当银行柜员叫号时，等候区中最先进入的顾客离开等候区前往柜台办理业务，若叫号时等候区无人，则此次叫号作废。

输入格式：

第一行输入一个不大于20的正整数 `N`，表示银行等候区能容纳的人数，

接下来用若干行表示依时间顺序先后发生的“`顾客想进银行`”或“`叫号`”事件，格式分别是：

- `顾客想进银行`，用 `In <id>` 表示，其中 `<id>` 是顾客编号，为不大于100000的正整数；
- `叫号`，用 `Calling` 表示。

最后一行是一个 `#` 符号，表示输入结束。

注意：

1. 题目输入保证每个顾客的编号是独一无二的，即：不会出现想进银行的顾客与已经在等候区的顾客编号相同的情况。
2. 保证后一个事件一定在前一个事件完成之后才发生，即：不需要考虑事件之间的“同步”问题。

输出格式：

对于输入的每个事件，按同样顺序在一行内输出事件的结果，格式分别是：

- `顾客想进银行`，若顾客进入，则输出 `<id> joined. Total:<t>` 其中 `<id>` 是该顾客的编号，`<t>` 是顾客进入后，等候区的人数
- `顾客想进银行`，若因等候区满而被拒绝，则输出 `<id> rejected.` 其中 `<id>` 是该顾客的编号
- `叫号`，若有顾客前往柜台，则输出 `<id> called. Total:<t>` 其中 `<id>` 是该顾客的编号，`<t>` 是顾客去柜台后，等候区的人数
- `叫号`，等候区无人，则输出 `No one!`