Data Structures

Lecture by Ccucumber12
2024/03/02

課程內容

- 什麼是資料結構？
- Queue
- Stack
- 例題討論
- Linked-list

什麼是資料結構？

儲存資料的方式

https://www.chinaeducationaltours.com/media/image/2021-03/16154487455131.jpg

儲存資料的方式

https://images.pexels.com/photos/1370295/pexels-photo-1370295.jpeg

儲存資料的方式

- 快速從資料中找到特定資訊
  - 麻將:一眼看出有什麼牌型
  - 圖書館:快速找到書籍位置
- 如何提取資訊？
- 如何維持資料？

電腦中的資料結構

- 程式儲存資料的各種方式
- 協助演算法存取資訊
- 協助節省時間與空間

常見的資料結構

- Array
- Queue, Stack, Deque - today!
- Linked-list - today!
- Set, Map - STL
- Heap - week3
- Binary Indexed Tree - hand 10
- Disjoint Set - hand 12
- ……

Queue

狀況情境

- 在餐廳櫃台處理訂單
- 先到的訂單先處理

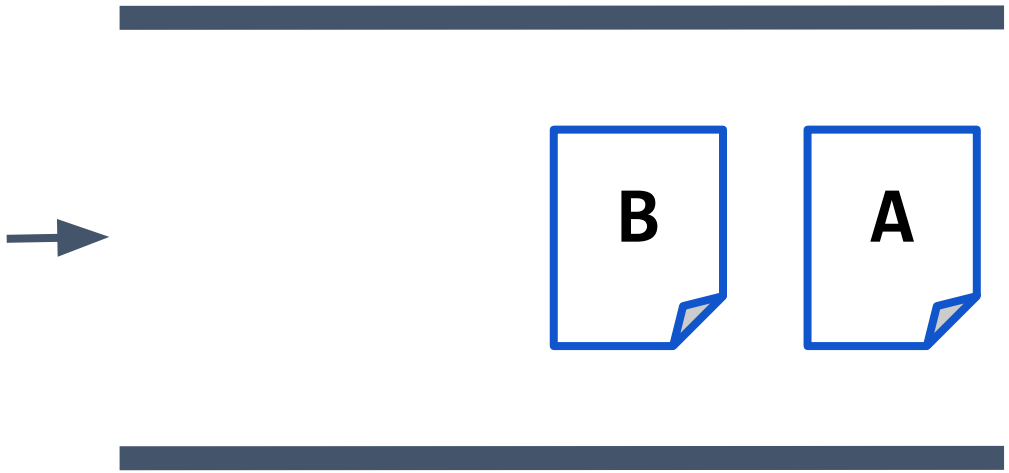
https://www.youtube.com/watch?app=desktop&v=QL6MMXKzkCc

狀況情境

- 加入訂單 A

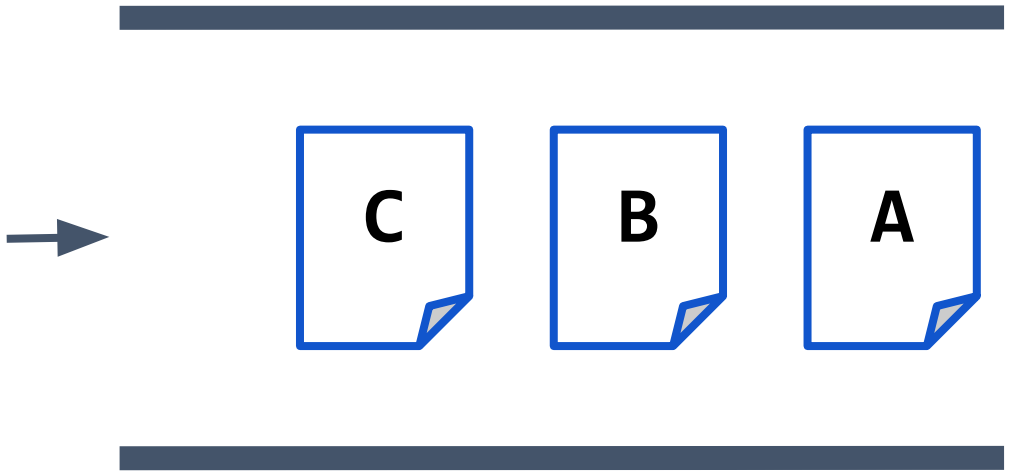
狀況情境

- 加入訂單 B



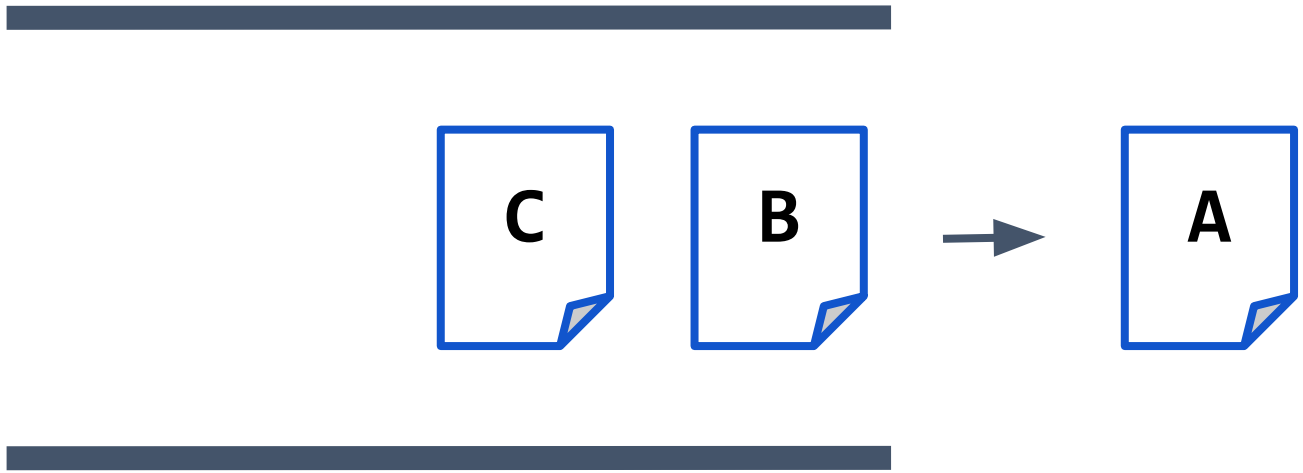
狀況情境

- 加入訂單 C



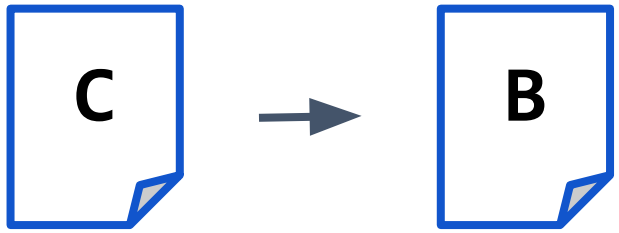
狀況情境

- 處理訂單:把 A 拿出來



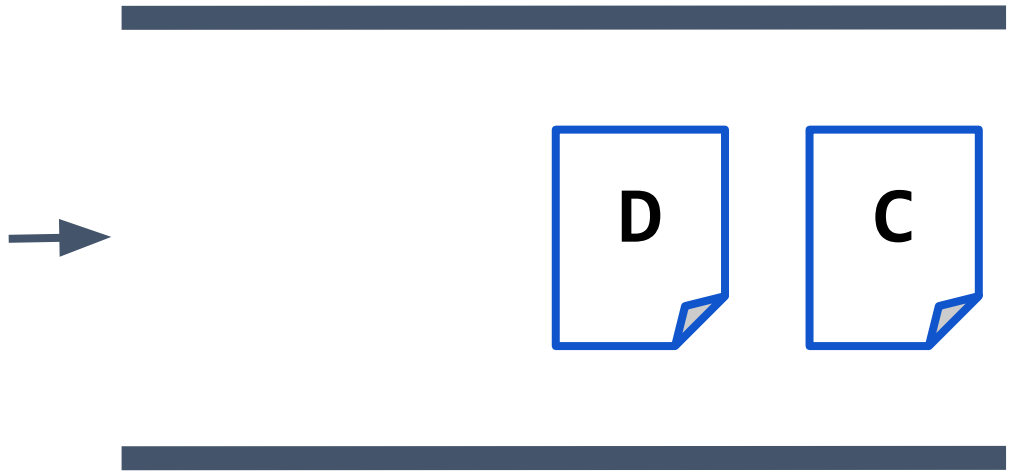
狀況情境

- 處理訂單:把 B 拿出來



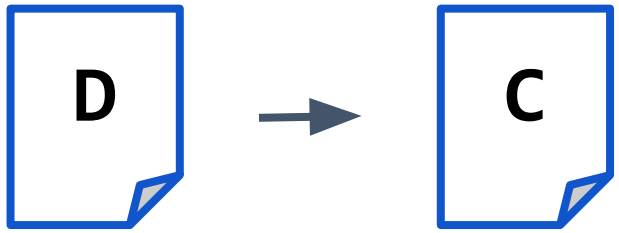
狀況情境

- 加入訂單 D



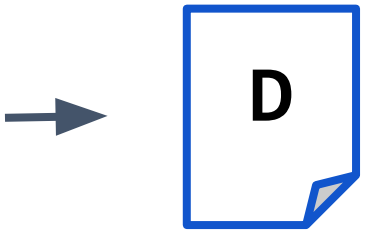
狀況情境

- 處理訂單:把 C 拿出來



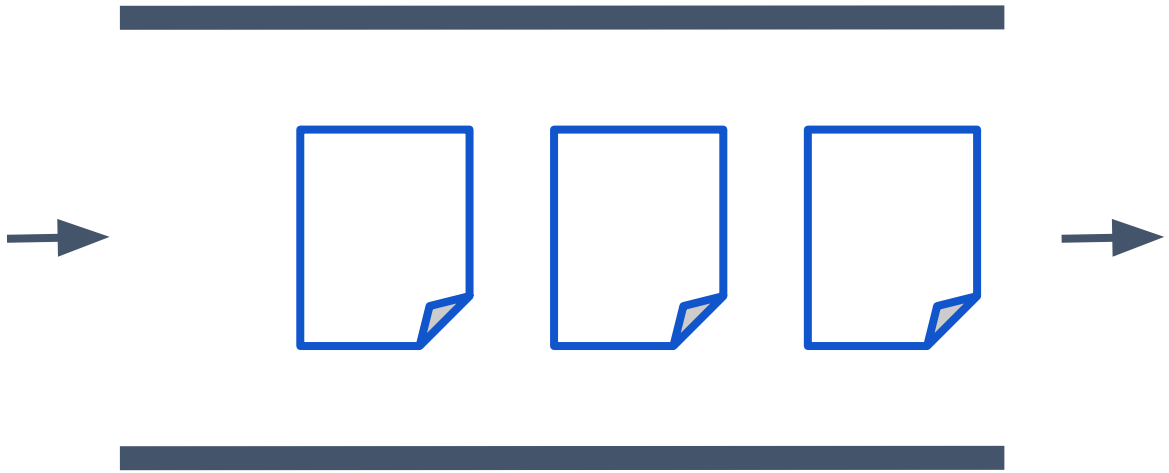
狀況情境

- 處理訂單:把 D 拿出來



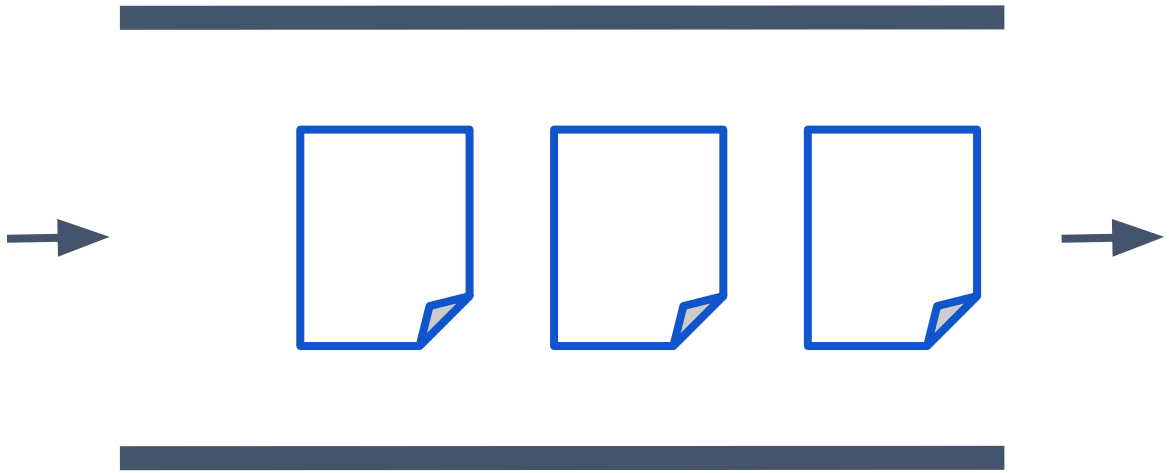
支援操作

- 加入訂單
- 拿出最早進入的訂單



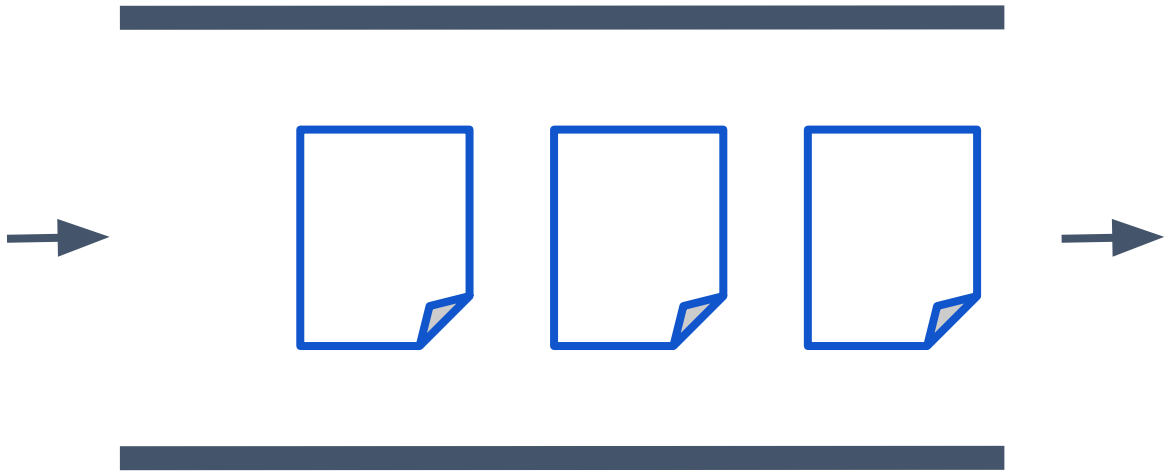
支援操作

- 加入訂單 → 資料
- 拿出最早進入的訂單 → 資料



支援操作

- 加入~~訂單~~ → 資料
- 拿出最早進入的~~訂單~~ → 資料
- **先進先出！**



Queue（佇列）

Queue 的功能

- 存取排在 queue 最前端的資料
- 刪除排在 queue 最前端的資料
- 新增資料到 queue 的最後端

Queue 的特性

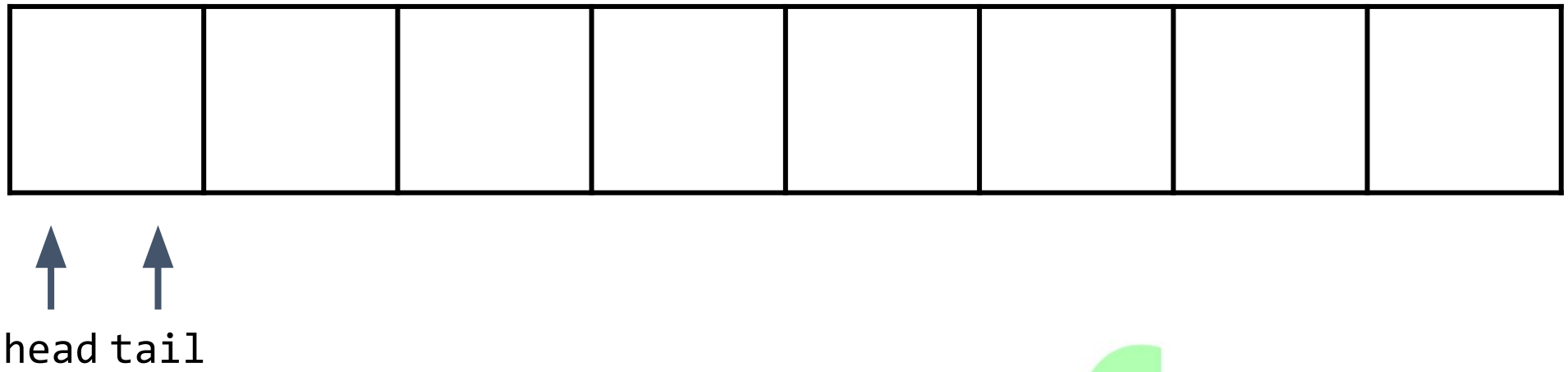
- 只能從最前端存取、刪除資料
- 只能從最後端新增資料
- 先進先出 (First In First Out, FIFO)

Queue 的實作

- front() 回傳 queue 最前端的值
- pop() 刪除 queue 最前端的資料
- push() 將新資料加入 queue 的最後端
- size() 回傳 queue 的大小

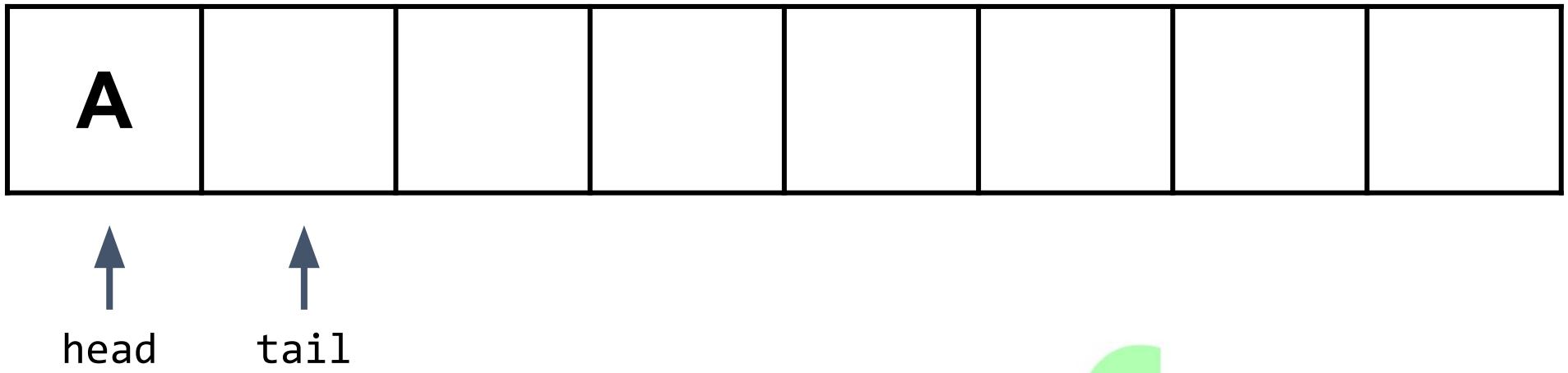
Queue 的實作

- 考慮用陣列儲存資料
- 紀錄目前的最前端 / 最後端



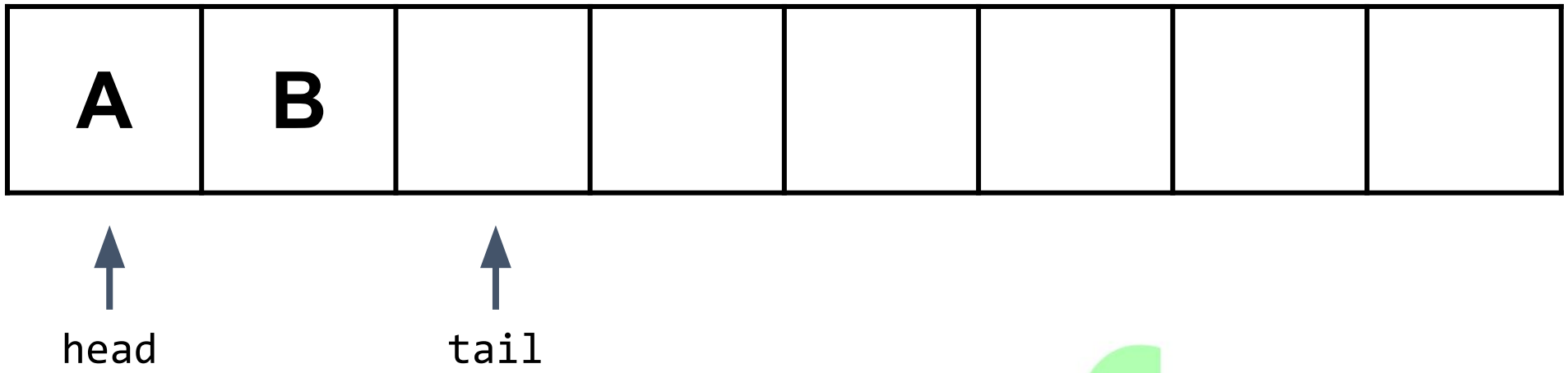
Queue 的實作

- push(A)



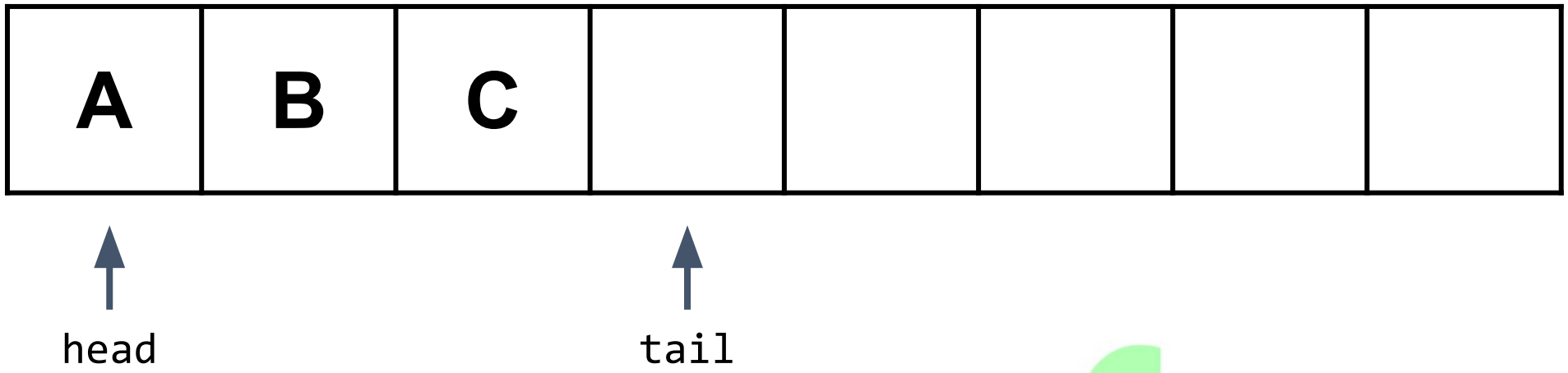
Queue 的實作

- push(B)



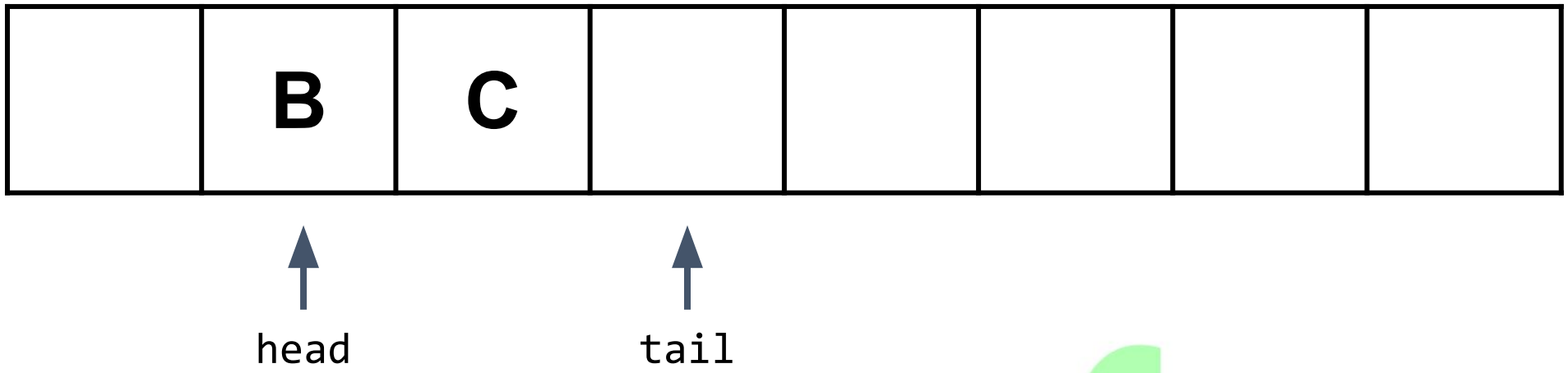
Queue 的實作

- push(C)



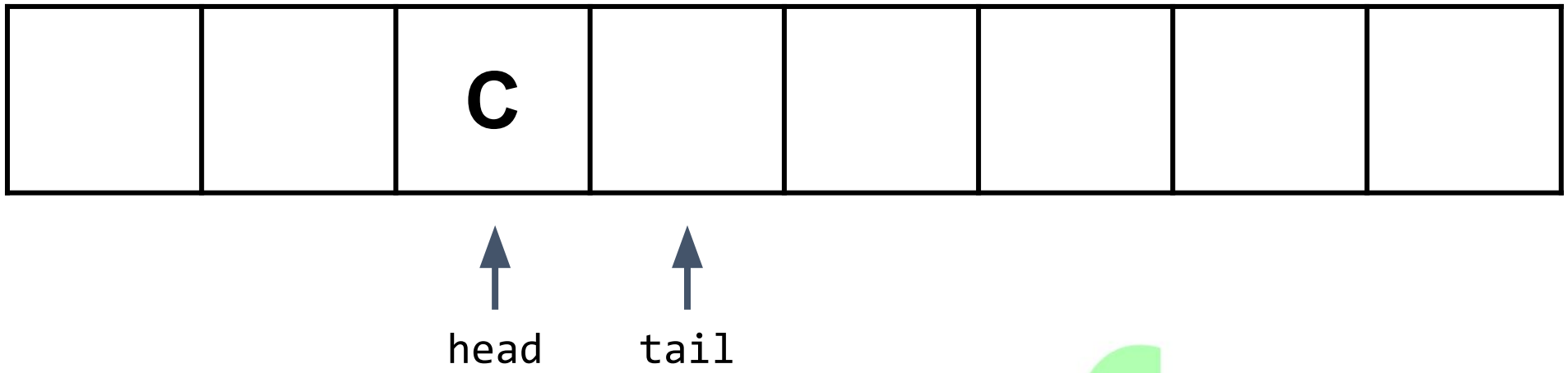
Queue 的實作

- pop()



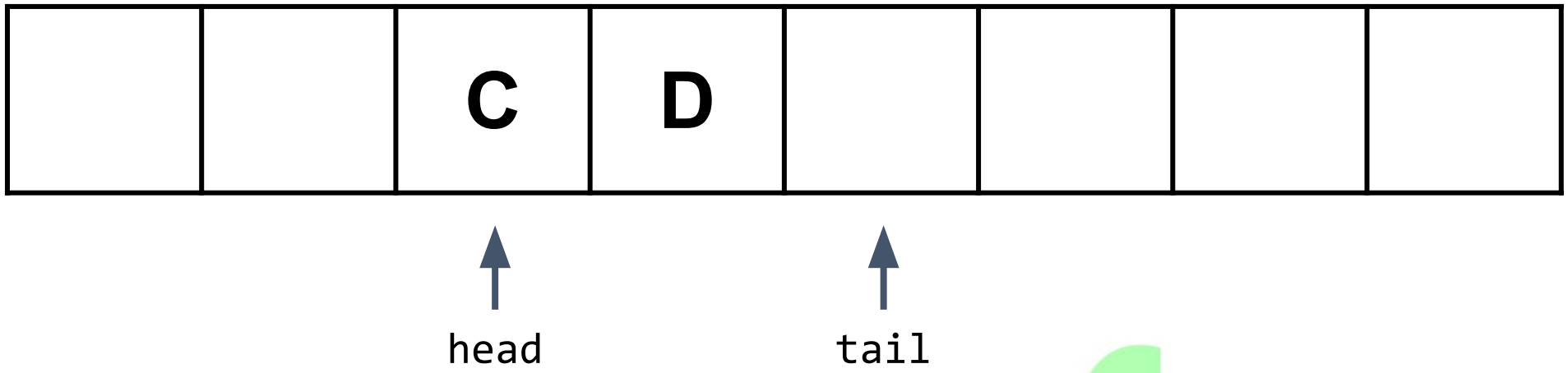
Queue 的實作

- pop()



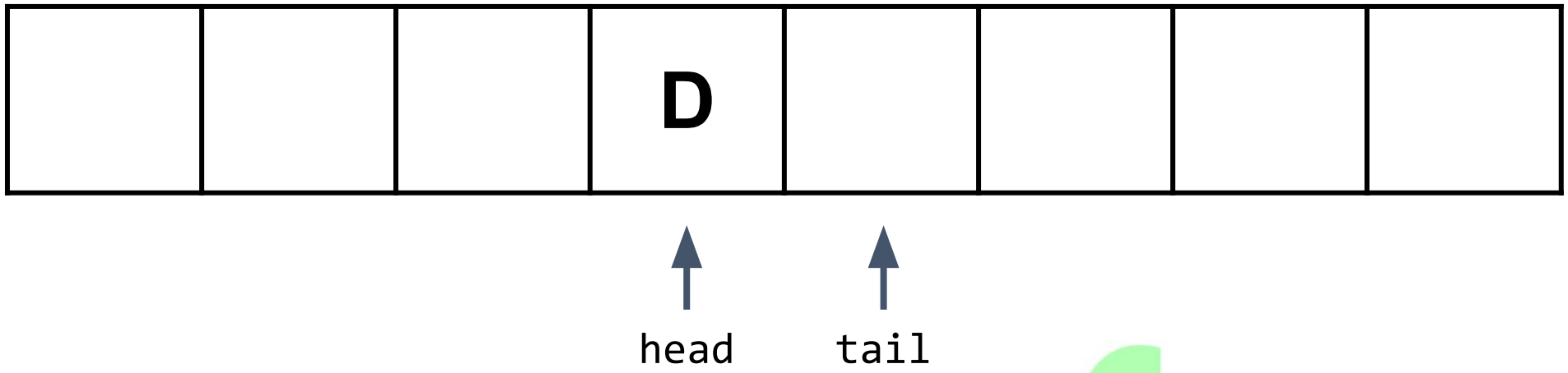
Queue 的實作

- push(D)



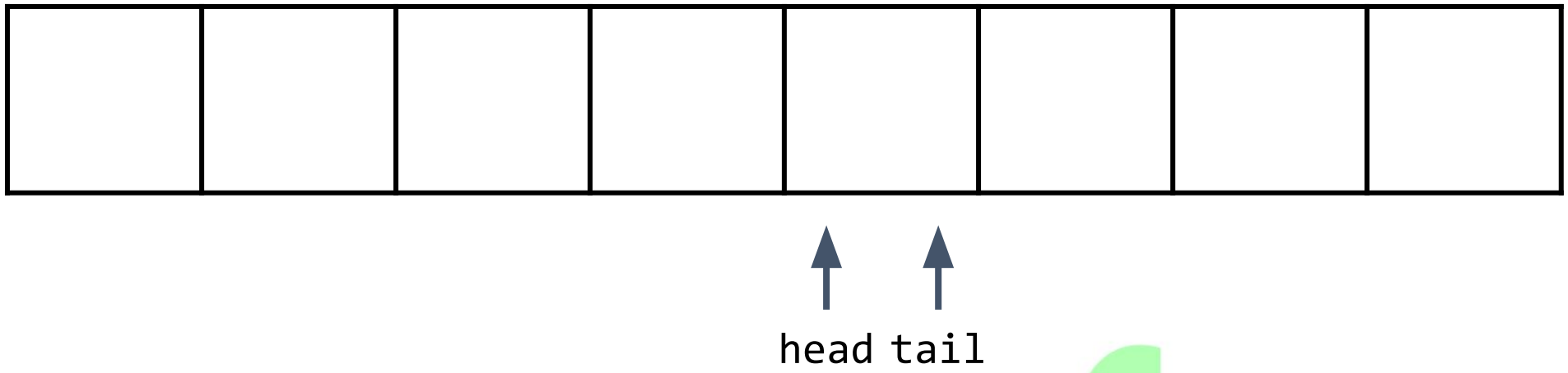
Queue 的實作

- pop()



Queue 的實作

- pop()



Queue 的實作

- 用變數 head, tail 記錄開頭、結尾位置
- push 時 tail++
- pop 時 head++

Queue 的實作

- 陣列要開多大？

Queue 的實作

- 陣列要開多大？
- ~~題目說 N 多大就開多大~~

Queue 的實作

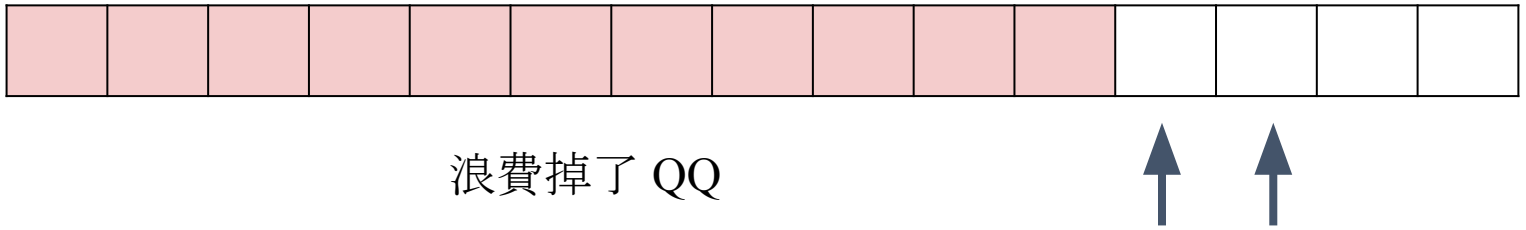
- 陣列要開多大？
- ~~題目說 N 多大就開多大~~
- 如果保證：操作數量 N > queue 大小上限 M
- 能不能開 O(M) 就好了？

Queue 的實作

- 考慮以下操作
  - push(A)
  - pop()
  - push(A)
  - pop()
  - push(A)
  - pop()
  - ...

Queue 的實作

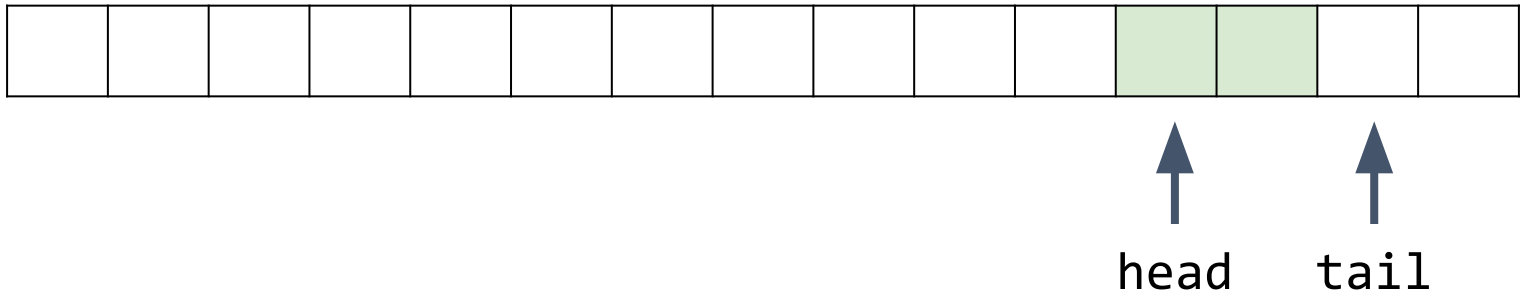
- head, tail 被加了 N 次
- queue 大小不超過 1



- push(A)
- pop()
- push(A)
- pop()
- push(A)
- pop()
- …

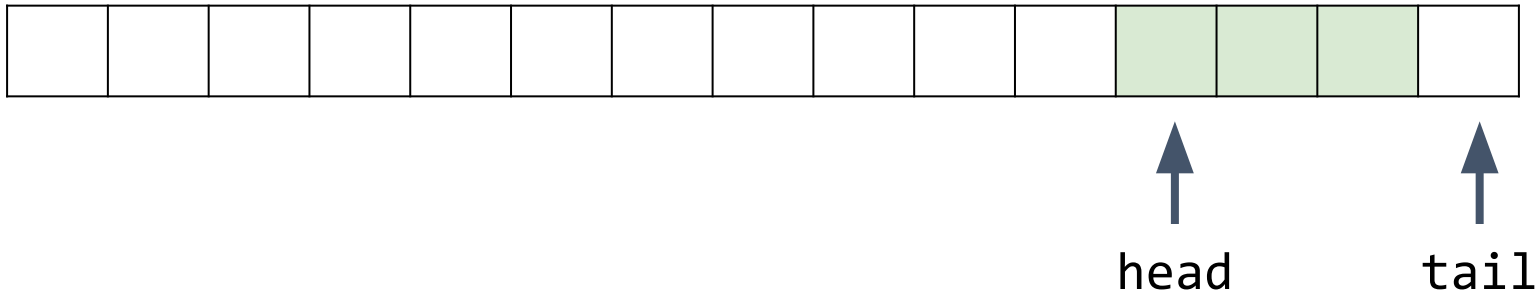
Circular Queue

- 碰到陣列尾巴再繞回來



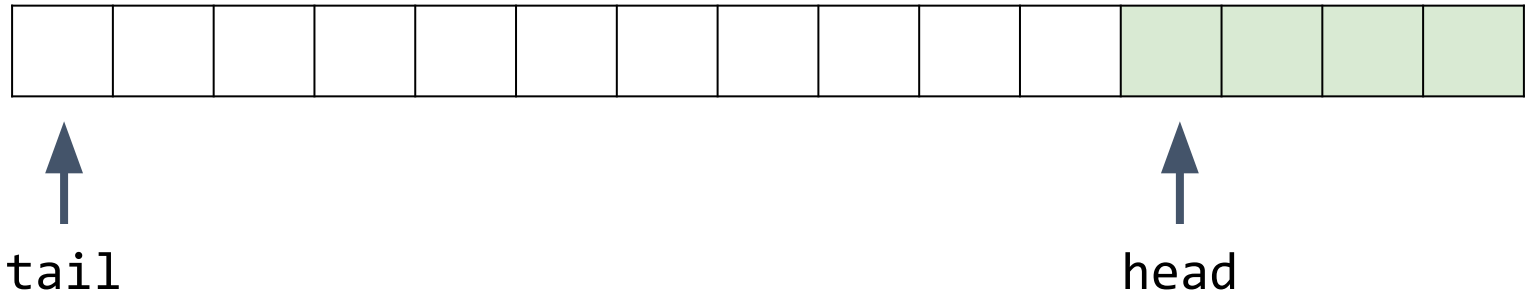
Circular Queue

- 碰到陣列尾巴再繞回來



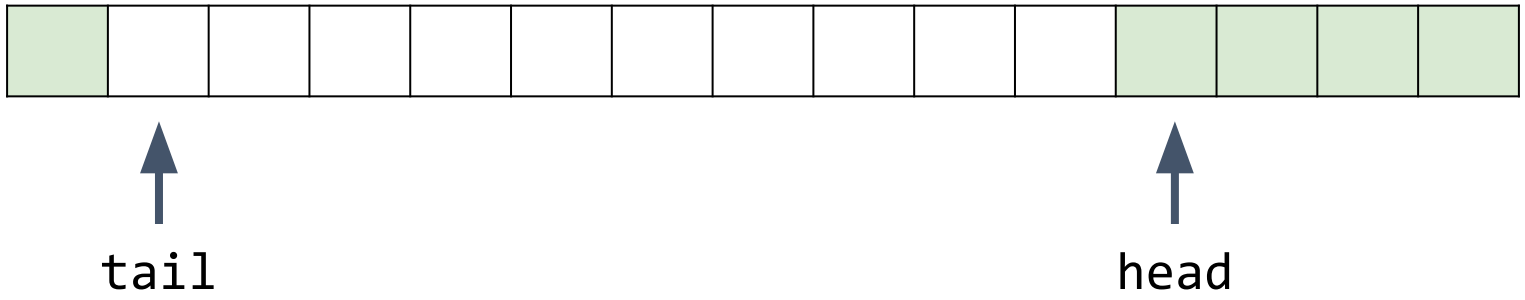
Circular Queue

- 碰到陣列尾巴再繞回來



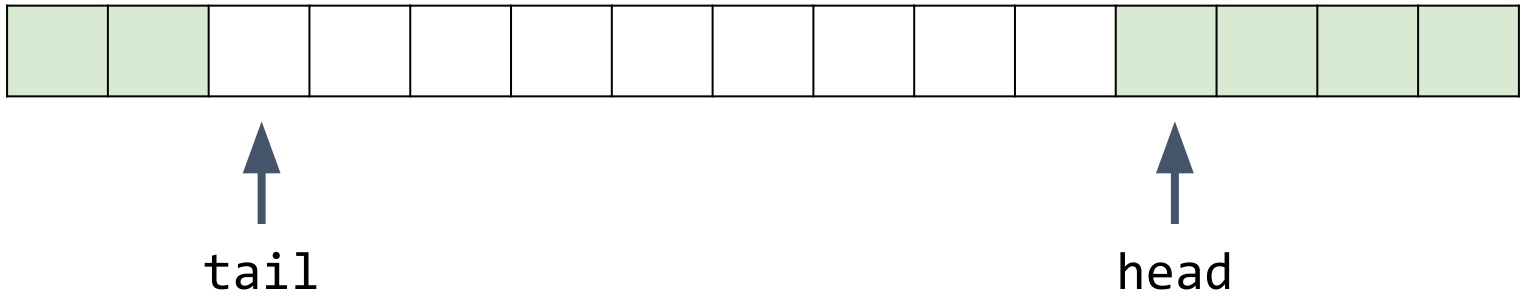
Circular Queue

- 碰到陣列尾巴再繞回來



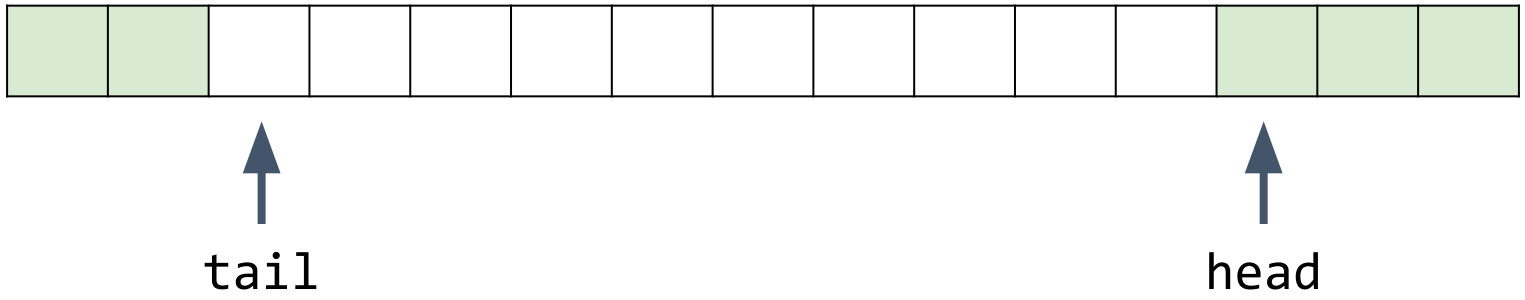
Circular Queue

- 碰到陣列尾巴再繞回來



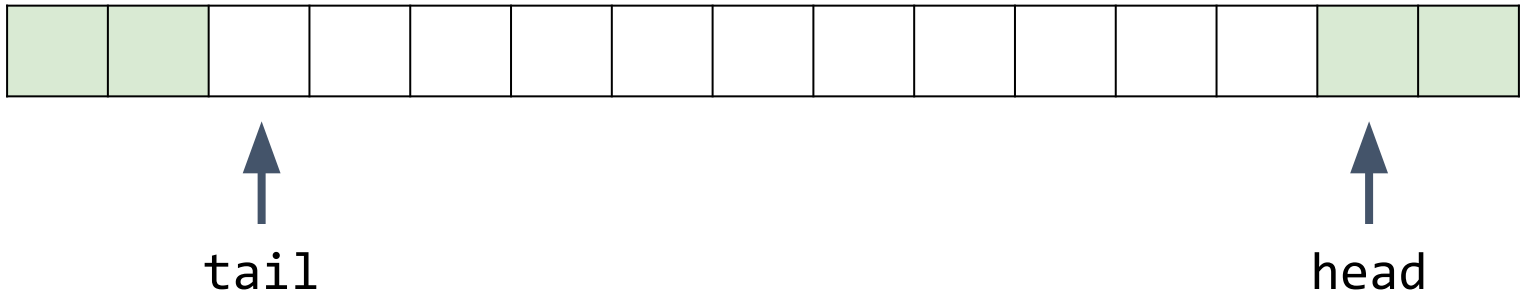
Circular Queue

- 碰到陣列尾巴再繞回來



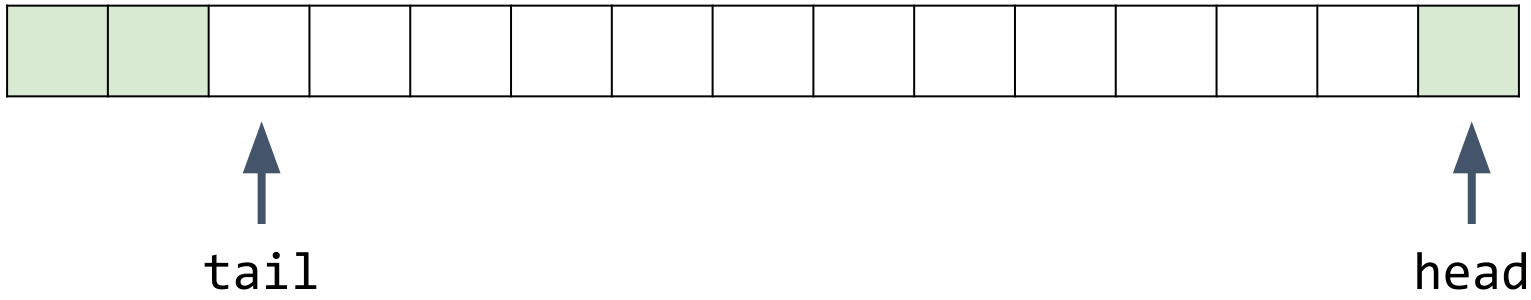
Circular Queue

- 碰到陣列尾巴再繞回來



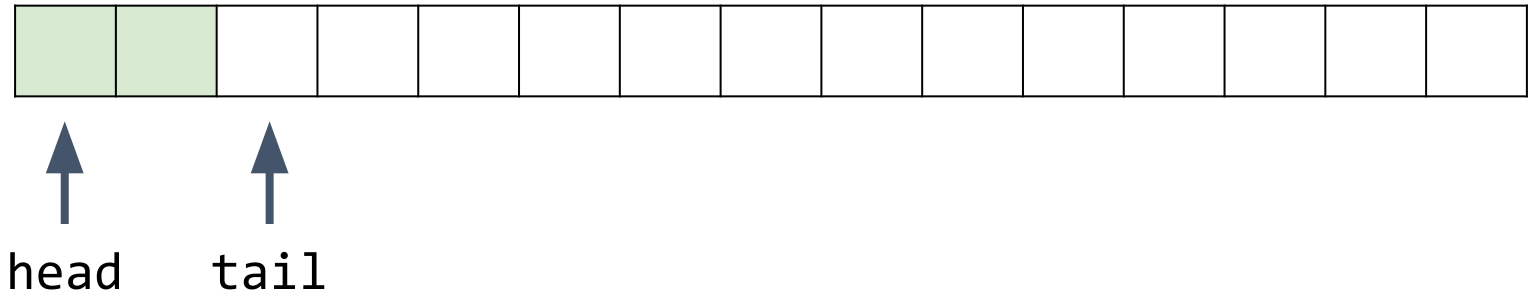
Circular Queue

- 碰到陣列尾巴再繞回來



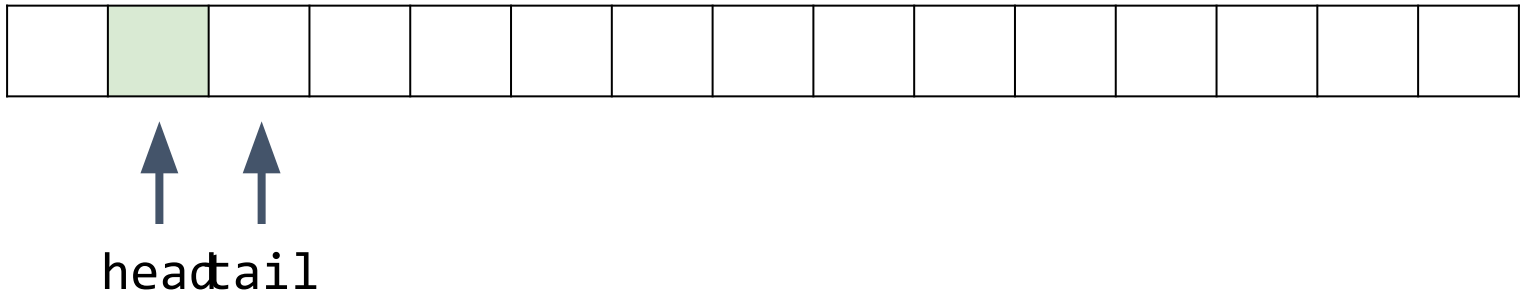
Circular Queue

- 碰到陣列尾巴再繞回來



Circular Queue

- 碰到陣列尾巴再繞回來



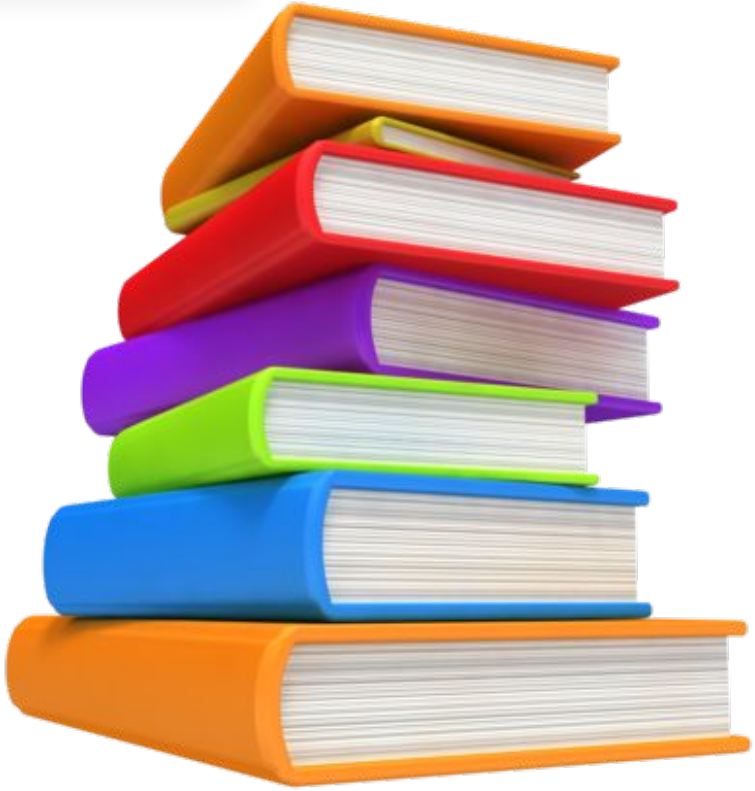

```cpp
struct Queue {
    int arr[MAXN], head, tail;
    Queue() : head(0), tail(0) {}
    int front() {
        return arr[head];
    }
    void pop() {
        head = (head + 1) % MAXN;
    }
    void push(int val) {
        arr[tail] = val;
        tail = (tail + 1) % MAXN;
    }
    int size() {
        return (tail - head + MAXN) % MAXN;
    }
};
```

Stack

Stack 的特性

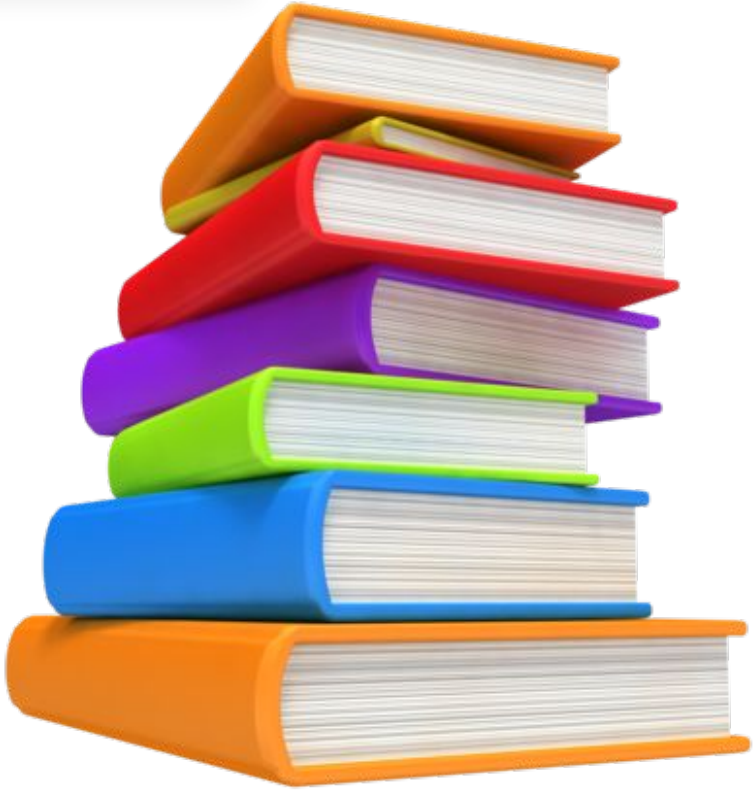
- 只能從最前端存取、刪除資料
- 只能從最**前端**新增資料
- 先進後出 (First In Last Out, FILO)

Stack(堆疊)



Stack(堆疊)

- 要怎麼拿到紫色的那本書？



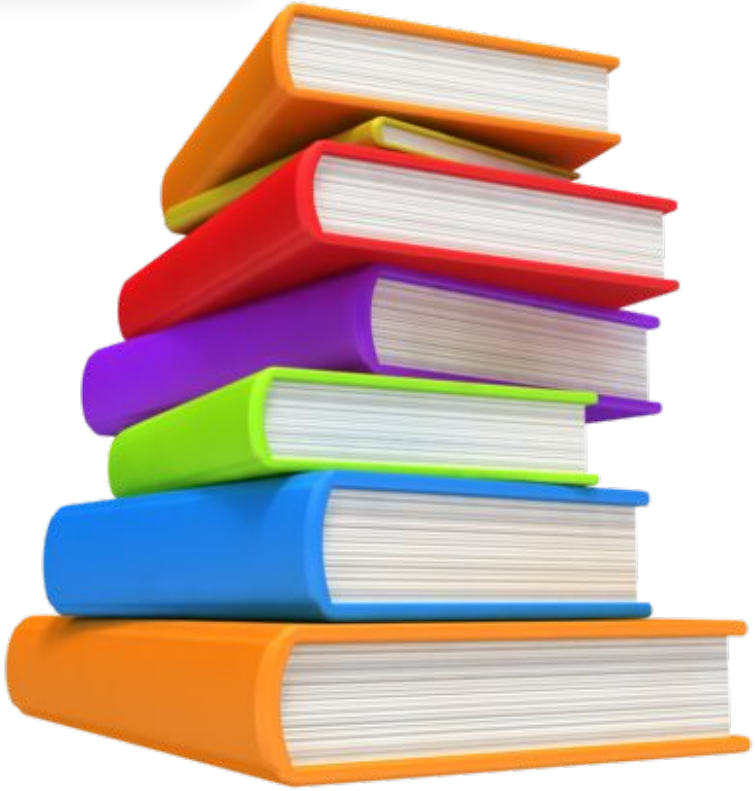
Stack(堆疊)

- 要怎麼拿到紫色的那本書？

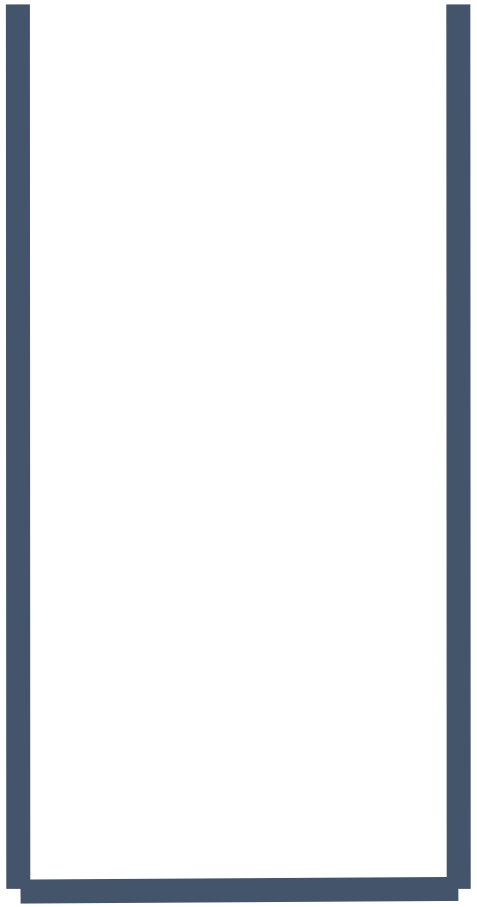
依序把橘、黃、紅的書拿起來

拿到紫色的書

依序將紅、黃、橘的書放回去

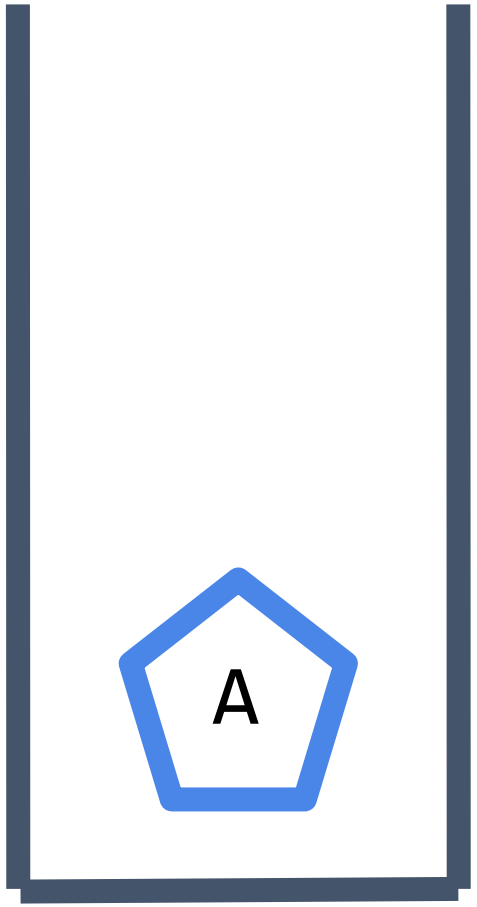


Stack(堆疊)



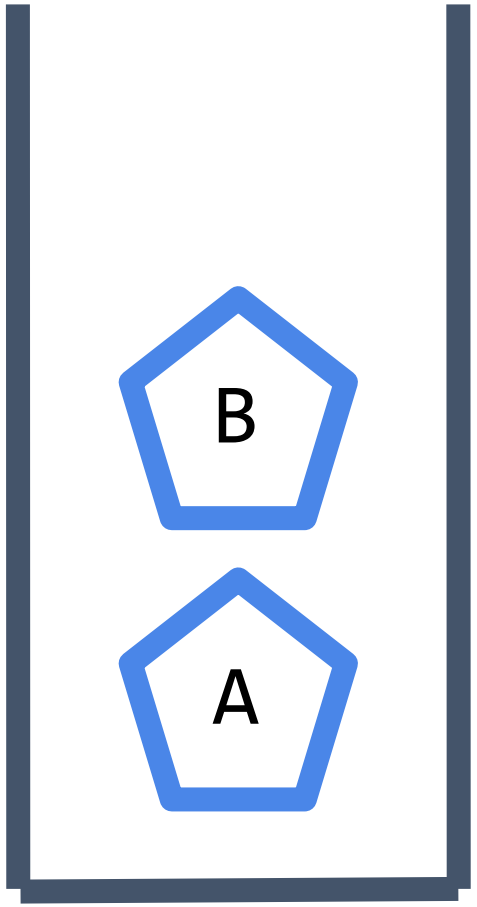
Empty

Stack(堆疊)



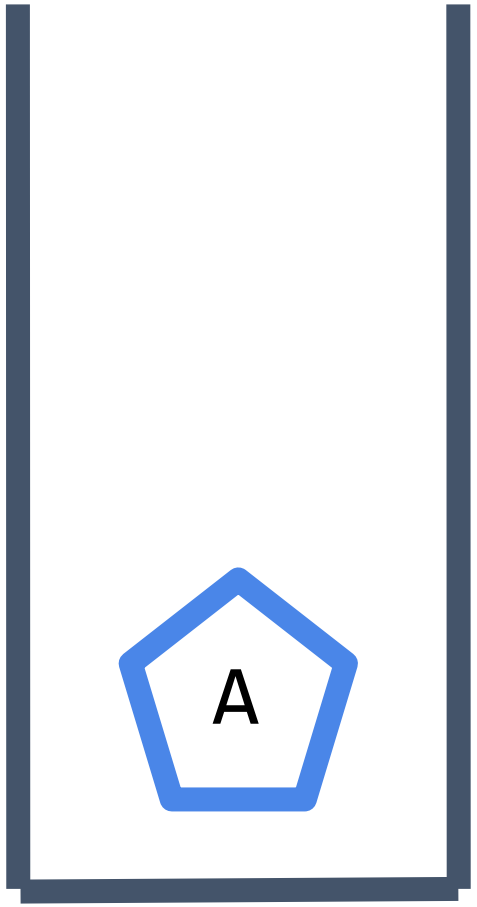
加入資料 A

Stack(堆疊)



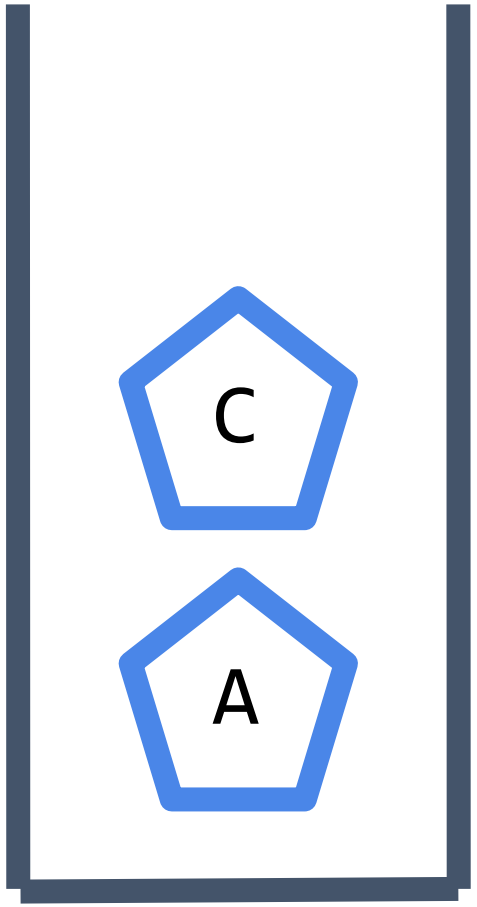
加入資料 B

Stack(堆疊)



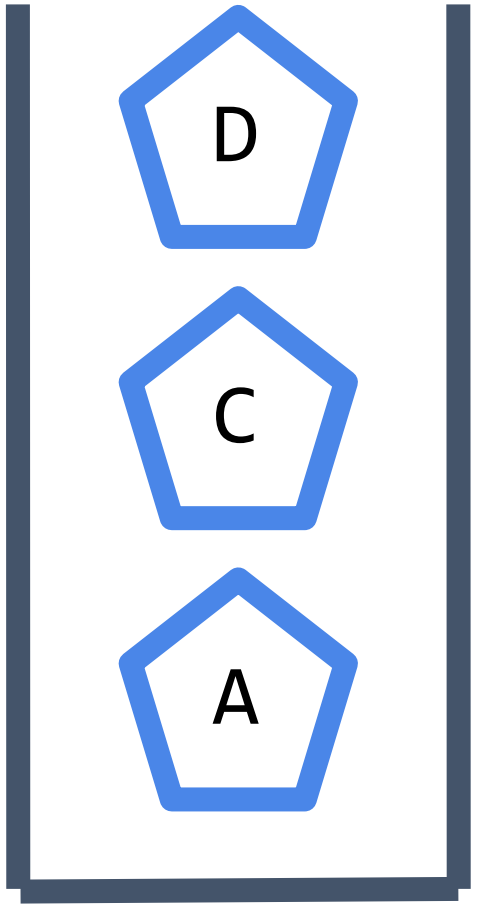
刪除最頂端資料

Stack(堆疊)



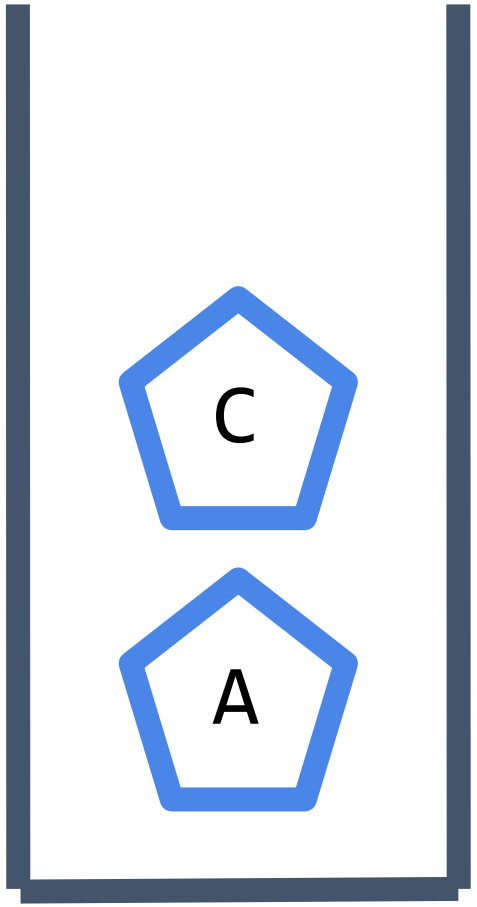
加入資料 C

Stack(堆疊)



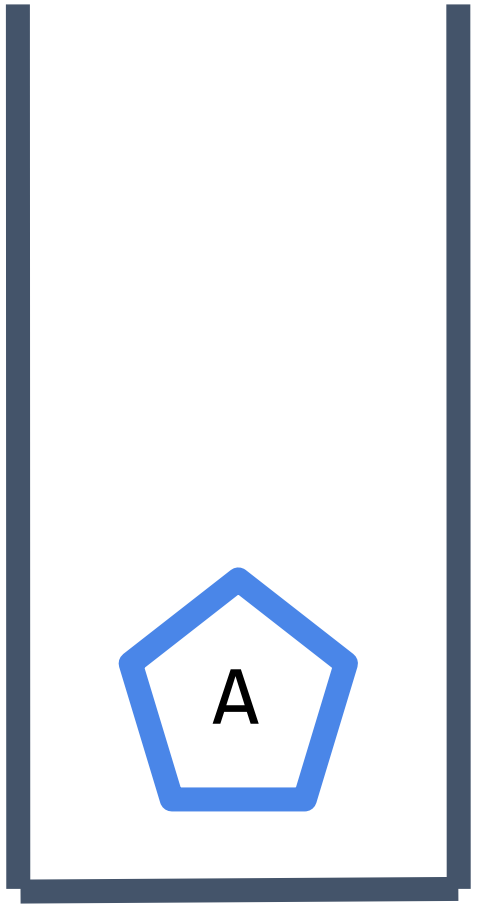
加入資料 D

Stack(堆疊)



刪除最頂端資料

Stack(堆疊)



刪除最頂端資料

Stack 的功能

- 存取排在 stack 最頂端的資料
- 刪除排在 stack 最頂端的資料
- 新增資料到 stack 的最頂端

Stack 的特性

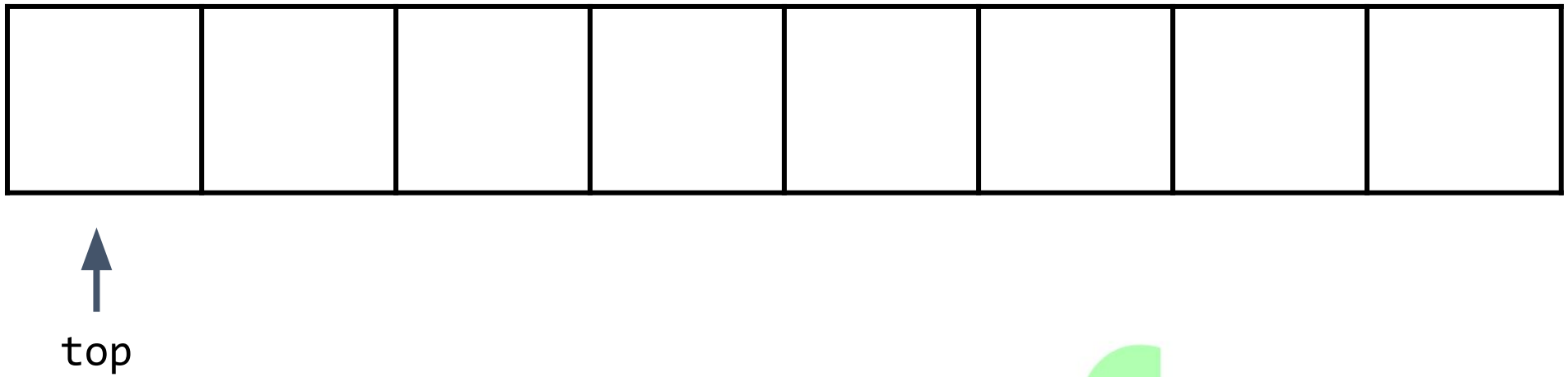
- 只能從最頂端存取、刪除、新增資料
- 先進後出(First In Last Out, FILO)
- 後進先出(Last In First Out, LIFO)

Stack 的實作

- top() 回傳 stack 最頂端的資料
- pop() 刪除 stack 最頂端的資料
- push() 將一個新的資料加入 stack
- size() 回傳 stack 的大小

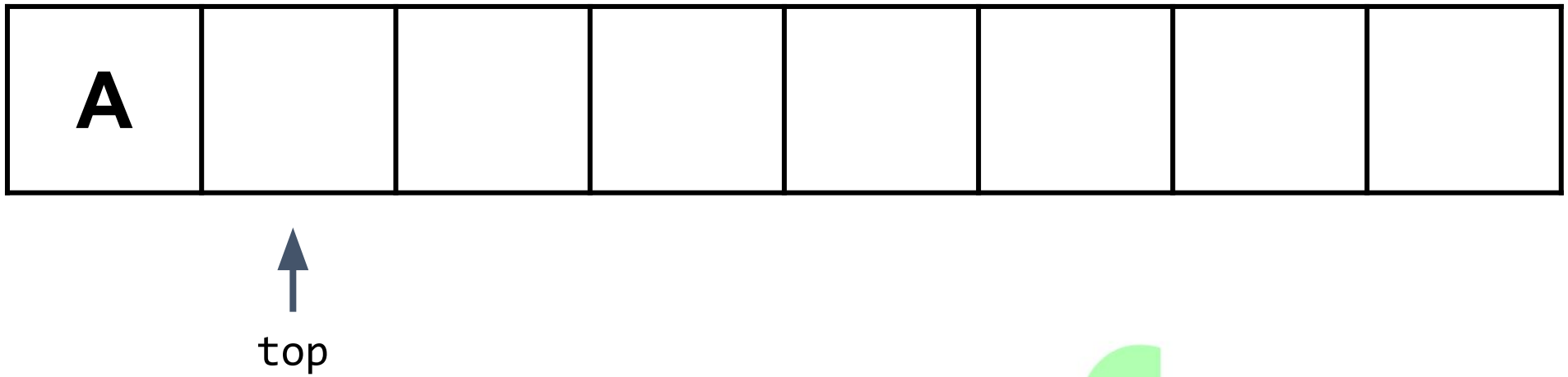
Stack 的實作

- 考慮用陣列儲存資料
- 紀錄目前的最頂端



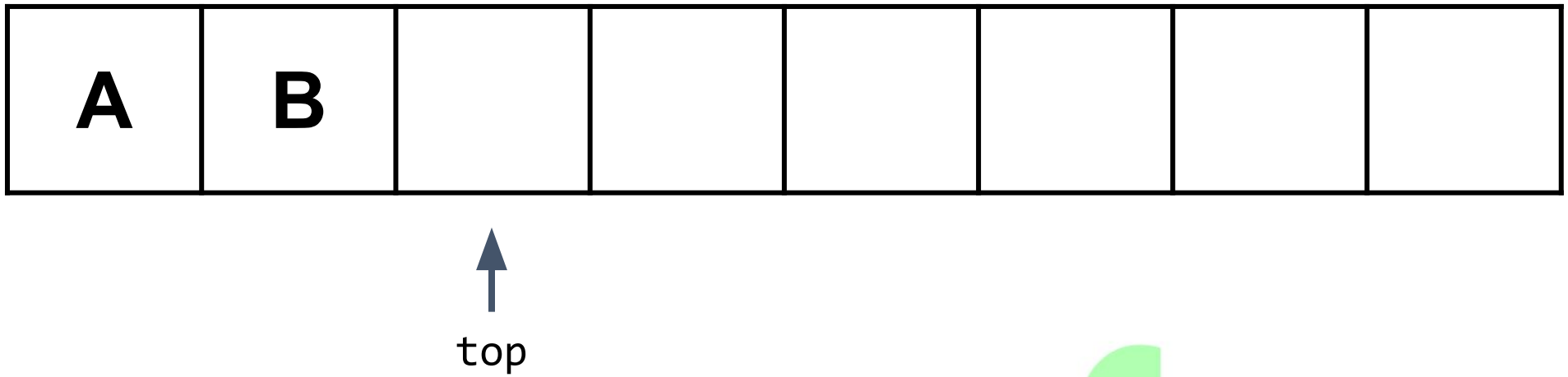
Stack 的實作

- push(A)



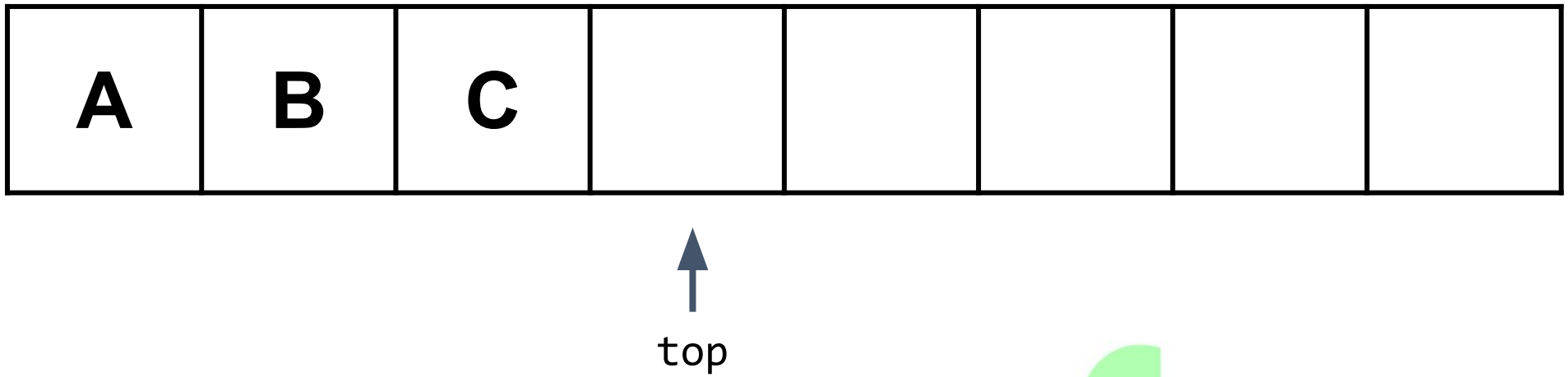
Stack 的實作

- push(B)



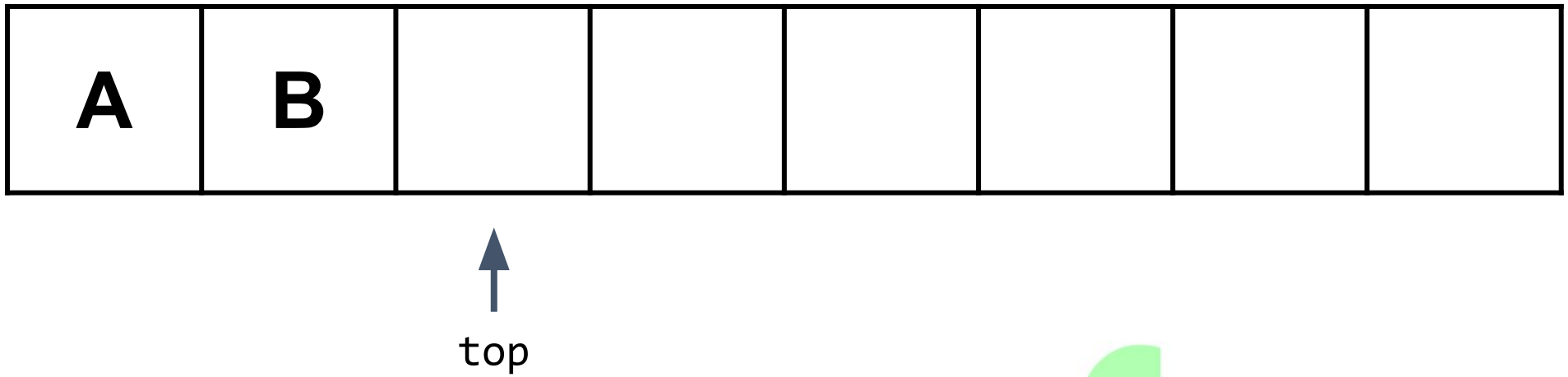
Stack 的實作

- push(C)



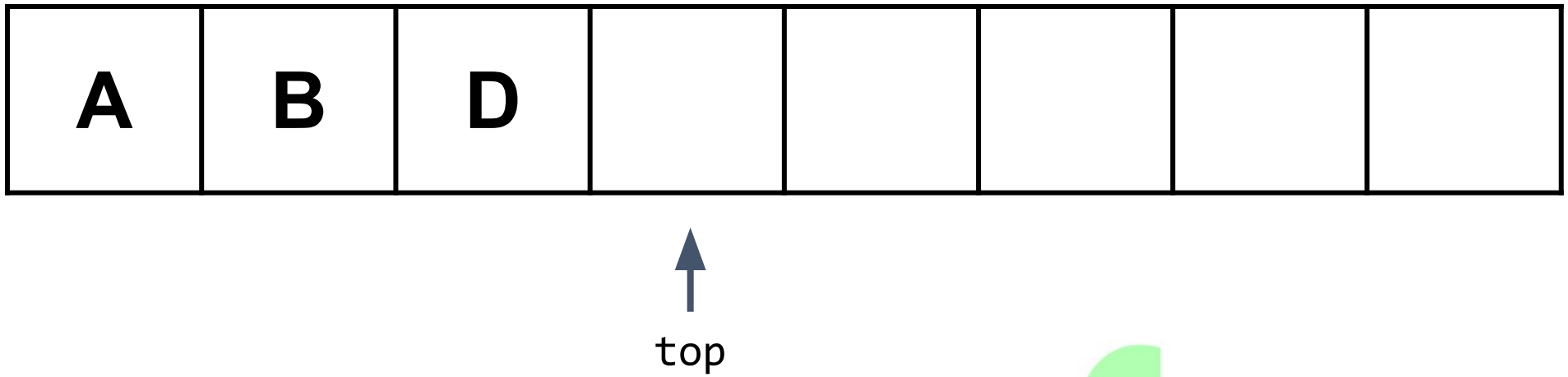
Stack 的實作

- pop()



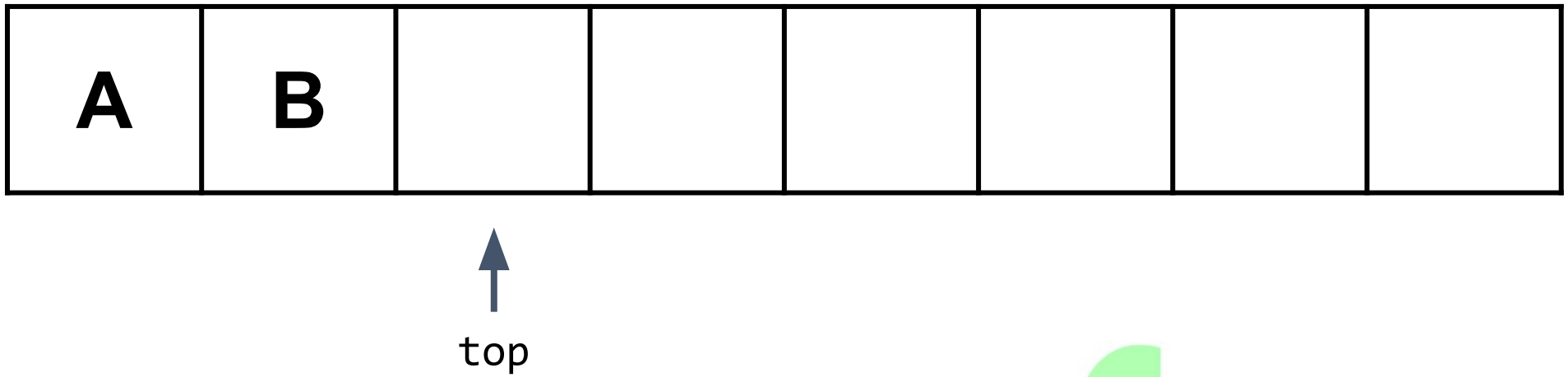
Stack 的實作

- push(D)



Stack 的實作

- pop()

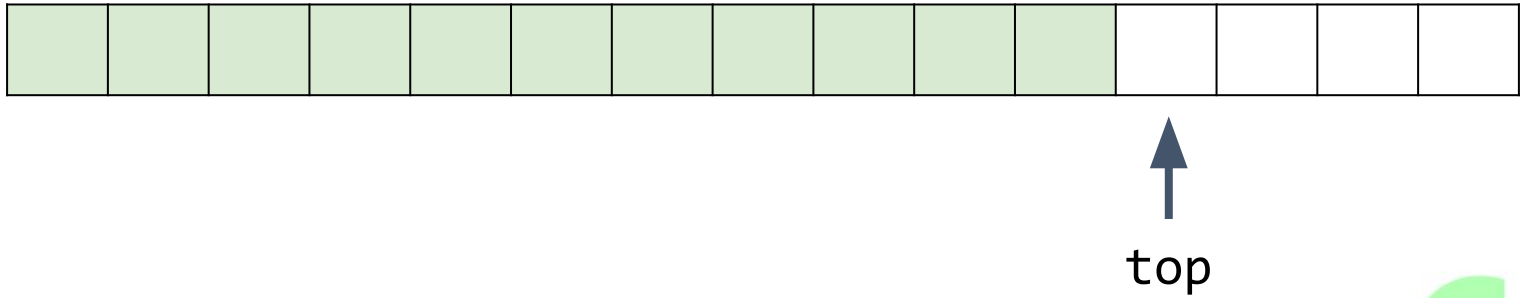


Stack 的實作

- 用變數 top 記錄頂端位置
- push 時 top++
- pop 時 top--

Stack 的實作

- 陣列要開多大？
- 題目說 N 多大就開多大
- 左側不會有浪費的空間




```cpp
struct Stack {
    int arr[MAXN], head;
    Stack() : head(0) {}
    int top() {
        return arr[head-1];
    }
    void pop() {
        head--;
    }
    void push(int val) {
        arr[head++] = val;
    }
    int size() {
        return head;
    }
};
```

到底要用 Stack 還是 Queue？

到底要用 Stack 還是 Queue？

Deque

Deque（雙端佇列）

- 有些人喜歡念成「de-queue」
- 有些人喜歡念成「D-E-queue」

Deque（雙端佇列）

我都念「deck」 /dɛk/

A deque ("double-ended queue") is a linear list for which all insertions and deletions (and usually all accesses) are made at the ends of the list. A deque is therefore more general than a stack or a queue; **it has some properties in common with a deck of cards, and it is pronounced the same way.**

The Art of Computer Programming, volume 1, section 2.2.1

Deque 的功能

- 存取、刪除排在 deque 最前端的資料
- 存取、刪除排在 deque 最後端的資料
- 新增資料到 deque 的最前端、最後端

Deque 的實作

- front(), back() 詢問前端 / 後端
- pop_front(), pop_back() 刪除前端 / 後端
- push_front(), push_back() 加入前端 / 後端
- size() 詢問大小

Deque 的實作

- 用變數 head, tail 記錄開頭、結尾位置
- 依照 前後 / 插入刪除 調整 head, tail

例題討論

例題一、括弧匹配

括弧匹配

```
#include <iostream>
using namespace std ;

const int MAXN = 100;

struct Queue {
    int arr[MAXN], head, tail;
    Queue() : head(0), tail(0) {}
    int front() {
        return arr[head];
    }
    void pop() {
        head = (head + 1) % MAXN;
    }
    void push(int val) {
        arr[tail] = val;
        tail = (tail + 1) % MAXN;
```

括弧匹配

給定一個僅包含 '('、')' 的字串，問其是否為合法括弧字串。

範例：

"()(()())" 是一個合法括弧字串

"()((()()" 不是一個合法括弧字串

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 每個 **左括弧** 都能夠往右邊找到 **右括弧**
  - 不會有多餘的右括弧沒有配對到

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 每個 **左括弧** 都能夠往右邊找到 **右括弧**
  - 不會有多餘的右括弧沒有配對到
- 從左到右掃過去
  - 對於每個 **左括弧** 希望未來可以遇到 **右括弧** 配對抵銷

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 每個 **左括弧** 都能夠往右邊找到 **右括弧**
  - 不會有多餘的右括弧沒有配對到
- 從左到右掃過去
  - 對於每個 **左括弧** 希望未來可以遇到 **右括弧** 配對抵銷
- 把左括弧儲存到 stack 上面
  - 遇到 '(' 就 push
  - 遇到 ')' 就 pop

括弧匹配

- 左括弧匹配失敗
  - 到最後還有左括弧剩下
  - 到最後 stack 還沒空
- 有多餘的右括弧
  - 遇到時沒有左括弧可以匹配
  - pop 的時候 stack 空了

括弧匹配

- 把左括弧想成 +1
- 把右括弧想成 -1

(	(	)	(	(	)	)	)
+1	+1	-1	+1	+1	-1	-1	-1

括弧匹配

- stack 大小 = 前綴和(從左邊開始到現在的總和)

(	(	)	(	(	)	)	)
+1	+1	-1	+1	+1	-1	-1	-1

括弧匹配

- stack 大小 = 前綴和(從左邊開始到現在的總和)
- 非法字串
  - 左括弧匹配失敗:到最後總和為正
  - 有多於的右括弧:過程中出現負數

(	(	)	(	(	)	)	)
+1	+1	-1	+1	+1	-1	-1	-1

記錄匹配對象

```
#include <iostream>
using namespace std ;

const int MAXN = 100;

struct Queue {
    int arr[MAXN], head, tail;
    Queue() : head(0), tail(0) {}
    int front() {
        return arr[head];
    }
    void pop() {
        head = (head + 1) % MAXN;
    }
    void push(int val) {
        arr[tail] = val;
        tail = (tail + 1) % MAXN;
```

記錄匹配對象

給定一個僅包含 '('、')' 長度 N 的字串
輸出 N/2 個數對代表 N/2 組括弧匹配。

範例:

"()(()())"

"(1, 2), (3, 8), (4, 5), (6, 7)"

記錄匹配對象

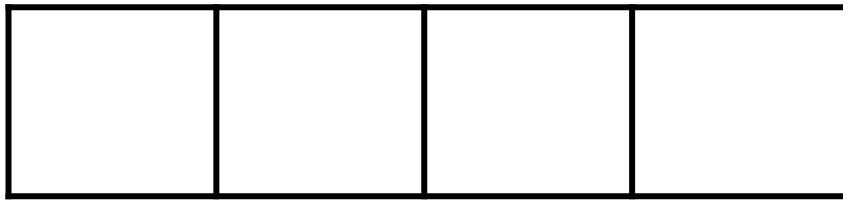
- 要怎麼知道誰跟誰配對？

記錄匹配對象

- 要怎麼知道誰跟誰配對？
- 在 stack pop 的時候 top 就是匹配對象
- 把 index 儲存到 stack 裡面

記錄匹配對象

(	)	(	(	)	(	)	)



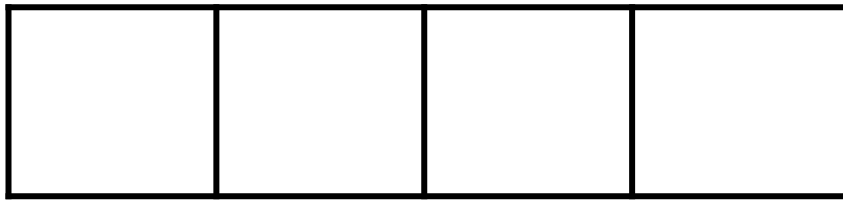
記錄匹配對象

(	)	(	(	)	(	)	)

1			

記錄匹配對象

(	)	(	(	)	(	)	)



(1, 2)

記錄匹配對象

(	)	(	(	)	(	)	)

3			

記錄匹配對象

(	)	(	(	)	(	)	)

3	4		

記錄匹配對象

(	)	(	(	)	(	)	)

3			

(4, 5)

記錄匹配對象

(	)	(	(	)	(	)	)

3	6		

記錄匹配對象

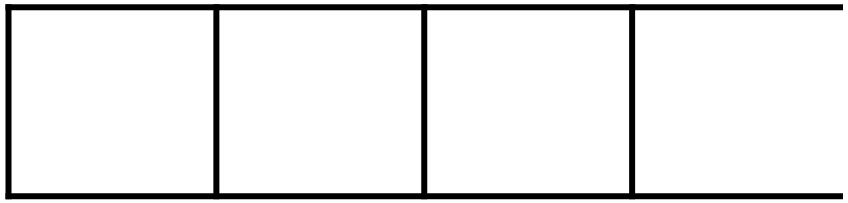
(	)	(	(	)	(	)	)

3			

(6, 7)

記錄匹配對象

(	)	(	(	)	(	)	)



(3, 8)

例題二、單調隊列

題目敘述

給 N 個數字，對於每個數字找到他右邊第一個比他小的人

N <= 10^5

範例：

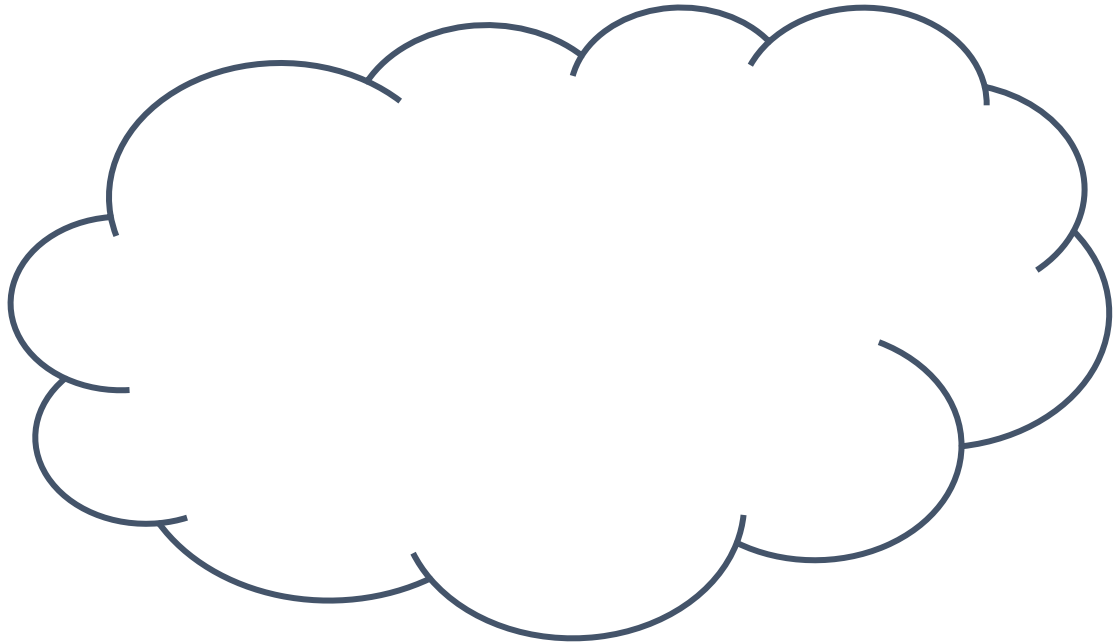
```
      3 7 5 6 2 5 4 3
ans> 5 3 5 5 - 7 8 -
```

單調隊列

- 從左到右掃過去
- 紀錄「還沒有答案」的集合
- 對於每個數字
  - 如果有人比較小 > 記錄他的答案並丟掉
  - 把目前數字塞進集合

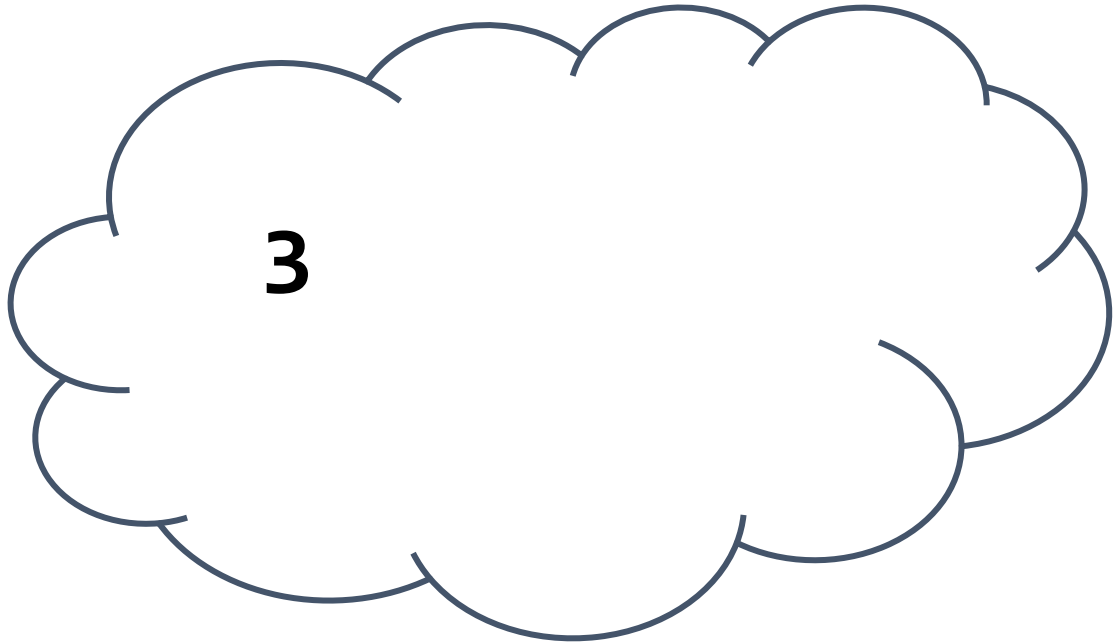
單調隊列

3	7	5	6	2	5	4	3



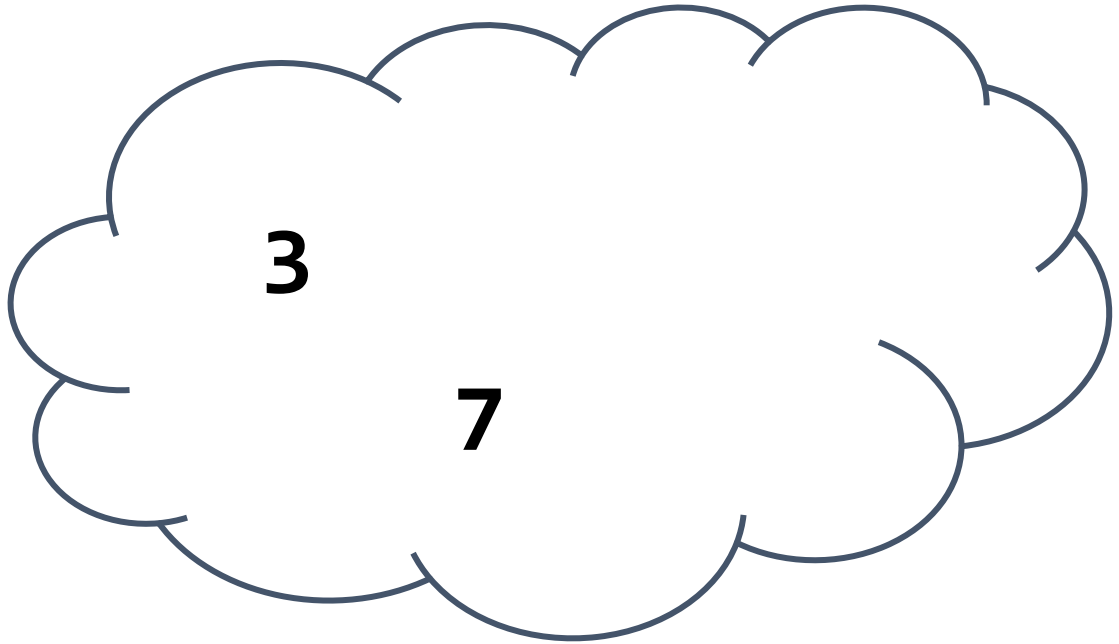
單調隊列

3	7	5	6	2	5	4	3



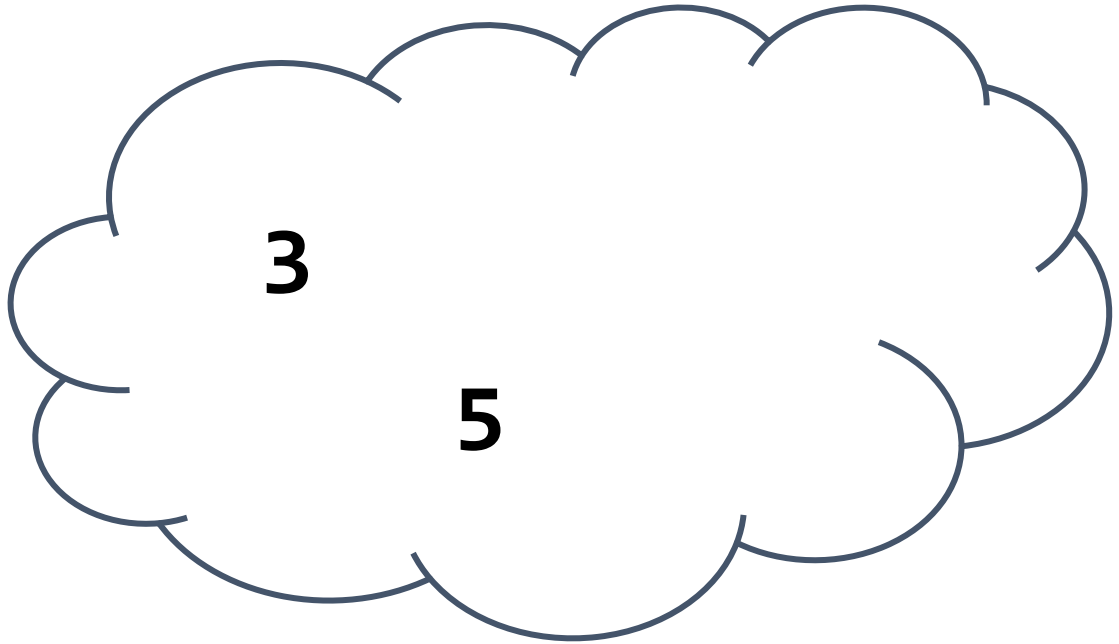
單調隊列

3	7	5	6	2	5	4	3



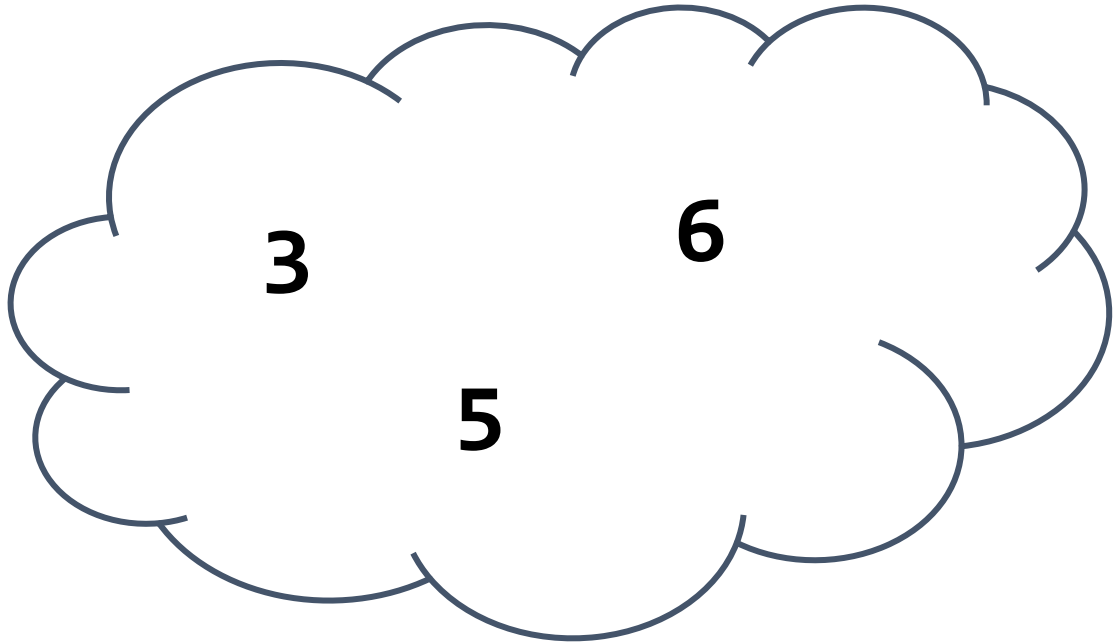
單調隊列

3	7	5	6	2	5	4	3
	3						



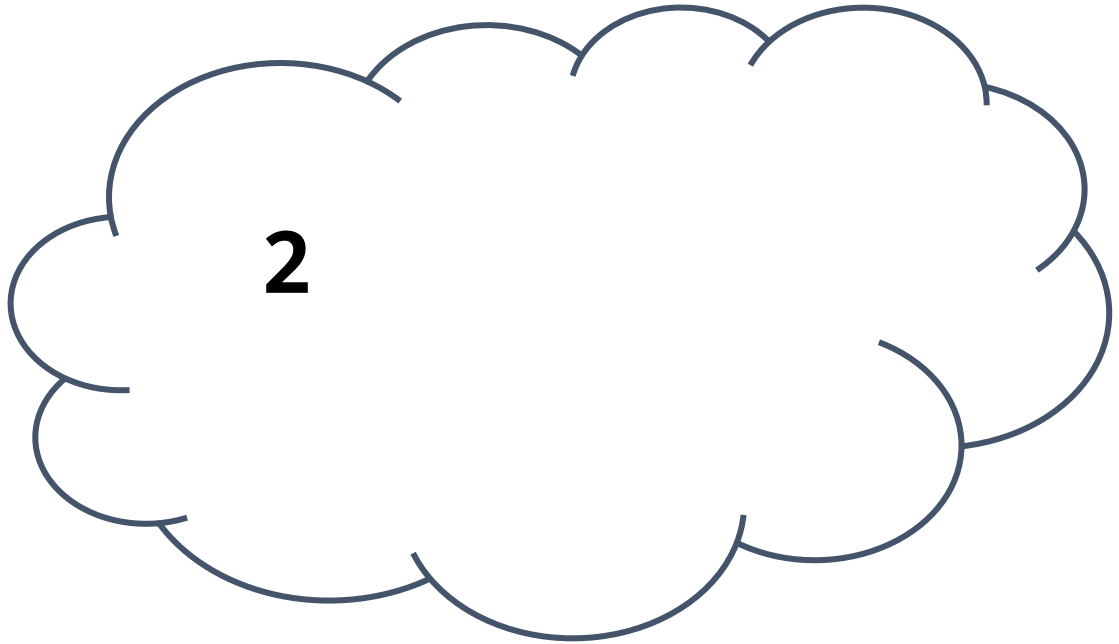
單調隊列

3	7	5	6	2	5	4	3
	3						



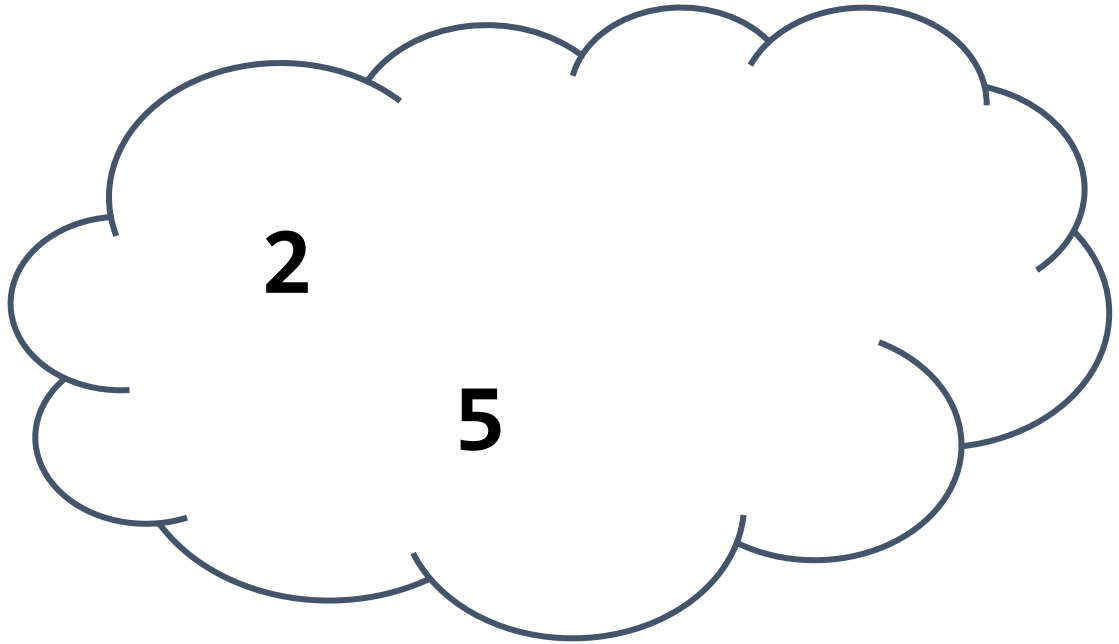
單調隊列

3	7	5	6	2	5	4	3
5	3	5	5				



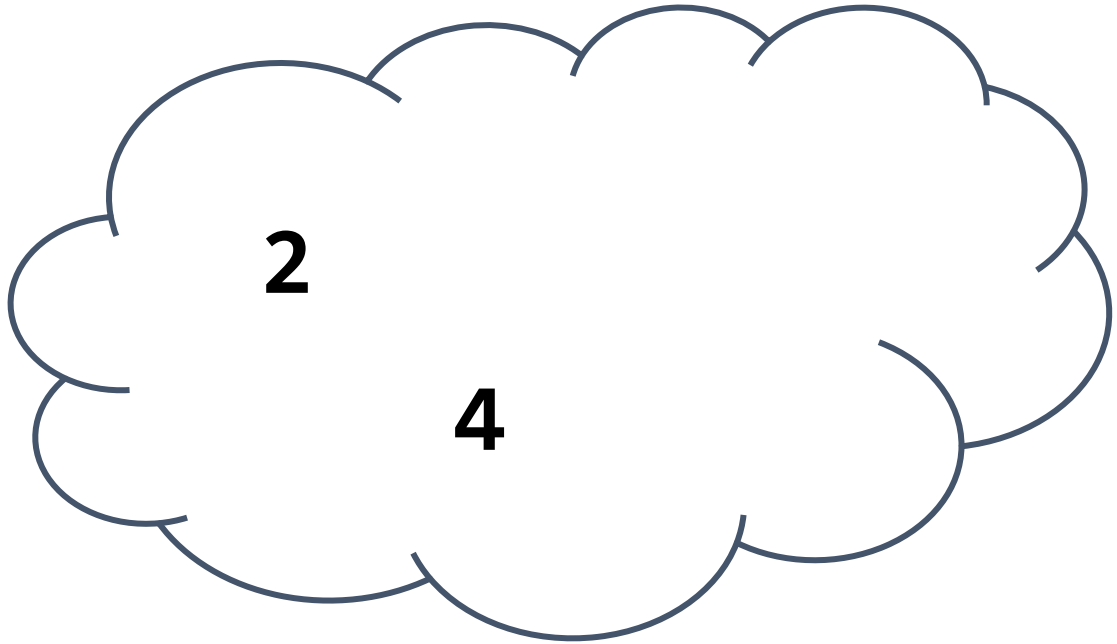
單調隊列

3	7	5	6	2	5	4	3
5	3	5	5				



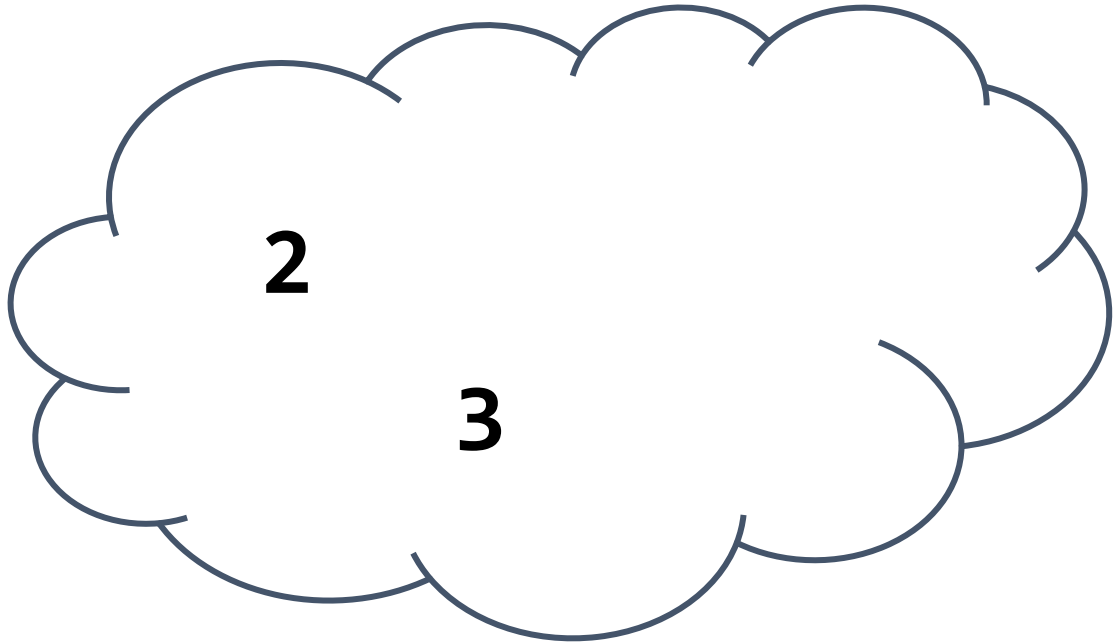
單調隊列

3	7	5	6	2	5	4	3
5	3	5	5		7		



單調隊列

3	7	5	6	2	5	4	3
5	3	5	5		7	8	



單調隊列

- 剔除比自己大的人 > 進入集合時自己必定最大
- 「還沒有答案」的集合中 越後面進來的人越大

單調隊列

- 剔除比自己大的人 > 進入集合時自己必定最大
- 「還沒有答案」的集合中 越後面進來的人越大
- 改成使用 stack 儲存
  - top 是目前最大的
  - 從頂端 push 進去

```
while(stk.size() > 0 && value[stk.top()] > value[idx]) {
    ans[stk.top()] = idx;
    stk.pop();
}
stk.push(idx);
```

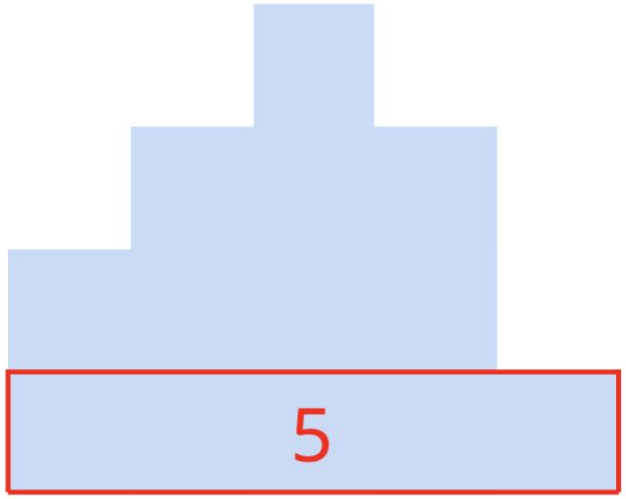
例題三、長條圖最大矩形

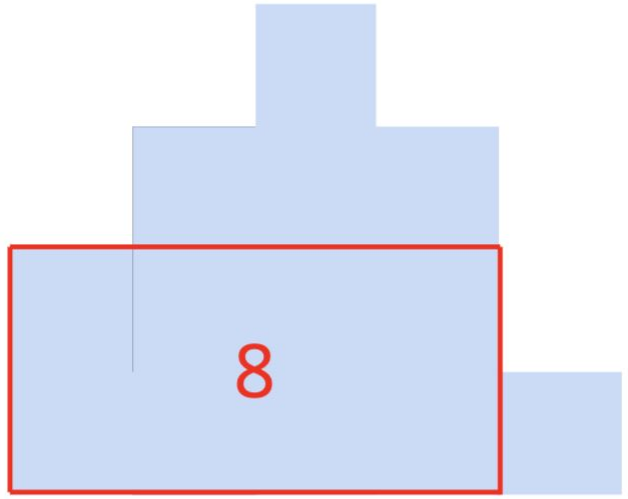
題目敘述

給你一張長條圖每個位置的高度，問你能畫出的最大矩形面積。（N <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



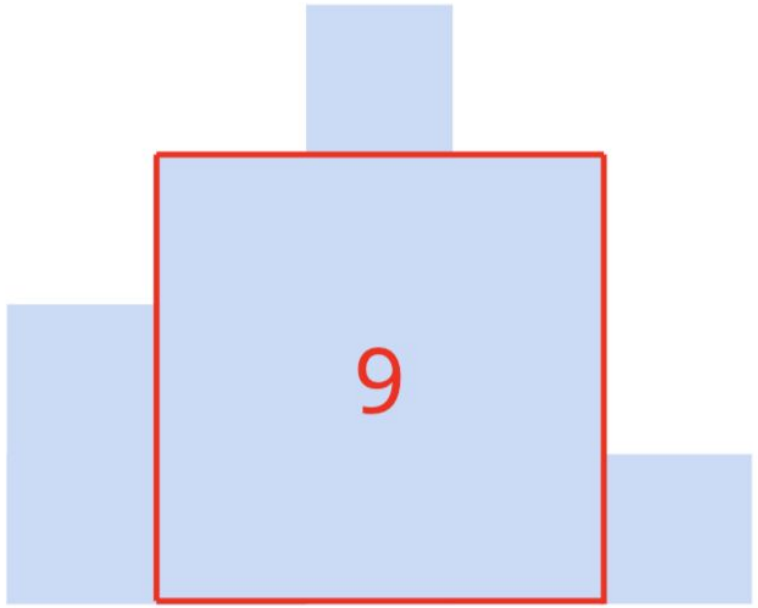


題目敘述

給你一張長條圖每個位置的高度，問你能畫出的最大矩形面積。（N <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



Hint

- 不知道從何下手的時候，可以先從複雜度較差的解開始想！

直覺的做法

- 枚舉每段區間，然後看高度最高可以是多少
- 複雜度:O(N^3)
  - 區間總共有 O(N^2) 個
  - 高度至多只能到最矮的那個 => O(N) 掃一遍區間找最小值

直覺的做法

- 枚舉每段區間，然後看高度最高可以是多少
- 複雜度:O(N^3)
  - 區間總共有 O(N^2) 個
  - **高度至多只能到最矮的那個** => O(N) 掃一遍區間找最小值

再想多一點...

- 「一塊區間的高度至多只能到最矮的那個」

再想多一點...

- 「一塊區間的高度至多只能到最矮的那個」
- 重點不是區間，是「最矮的那個」
  - 從枚舉區間，變成枚舉每個 bar 的高度

再想多一點...

- 「一塊區間的高度至多只能到最矮的那個」
- 重點不是區間，是「最矮的那個」
  - 從枚舉區間，變成枚舉每個 bar 的高度
- 如果我是最低的，那往左往右至多可以延伸多少？
  - 分別找到左右兩邊第一個比我小的！

問題轉換

- 給定序列，對每一項分別找到左右離他最近且比他小的值。（N <= 10^5、值域 <= 10^9 ）

問題轉換

- 給定序列，對每一項分別找到左右離他最近且比他小的值。（N <= 10^5、值域 <= 10^9 ）

問題轉換

- 給定序列，對每一項分別找到左右離他最近且比他小的值。（N <= 10^5、值域 <= 10^9 ）
- 左右邊各做一次！
- O(N) + O(N)

步驟整理

1. 要找最大矩形，可以「枚舉每個值作為最小值」向外延伸
2. 將向左、向右拆開成兩個問題
3. 用單調對列解「找到左 / 右邊第一個比我小的值」
4. 合併左右答案，枚舉計算所有位置的答案

延伸問題

給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）

3	7	5	6	2	5	4	3

延伸問題

給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）

3	7	5	6	2	5	4	3

7

延伸問題

給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）

3	7	5	6	2	5	4	3

7

延伸問題

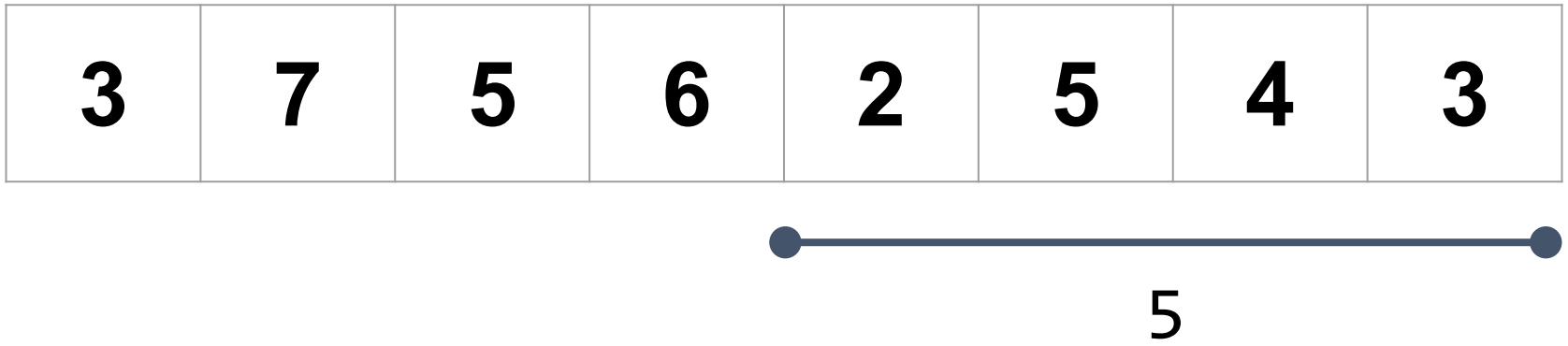
給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）

3	7	5	6	2	5	4	3

6

延伸問題

給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）



延伸問題

給定長度為 N 的序列，問每個長度 K 連續區間的區間最大值。
（N, K <= 10^6）

- 維護「可能成為答案」的集合 > 單調遞增隊列
- 在左邊的人有機會過期 > deque

3	7	5	6	2	5	4	3

Linked-list

Linked-list 的概念

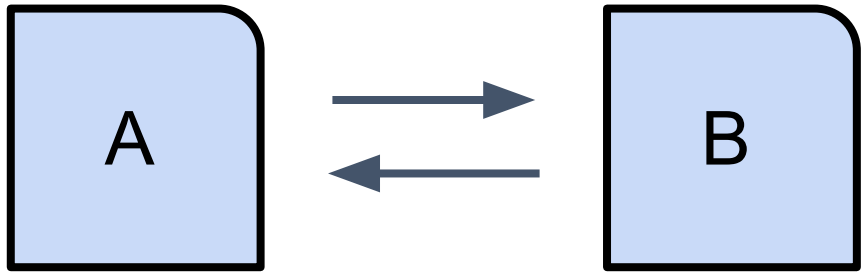
- 對於每個資料紀錄前後資料的位置
- 可以 O(1) 加入、刪除特定資料
- 不支援 random-access
  - 不能 O(1) 存取第 i 個資料

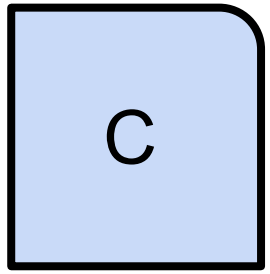
Linked-list 的概念

- 對於每個資料紀錄前後資料的位置
- 可以 O(1) 加入、刪除特定資料
- 不支援 random-access
  - 不能 O(1) 存取第 i 個資料
- 這跟陣列不一樣的地方在哪？

加入資料

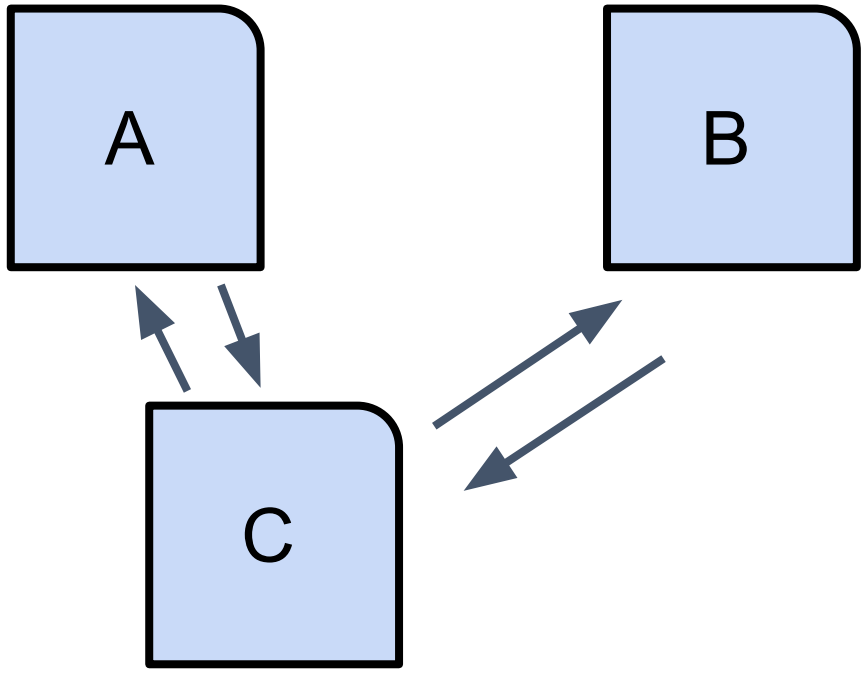
- 假設我們想將資料 C 插入在資料 A、B 之間





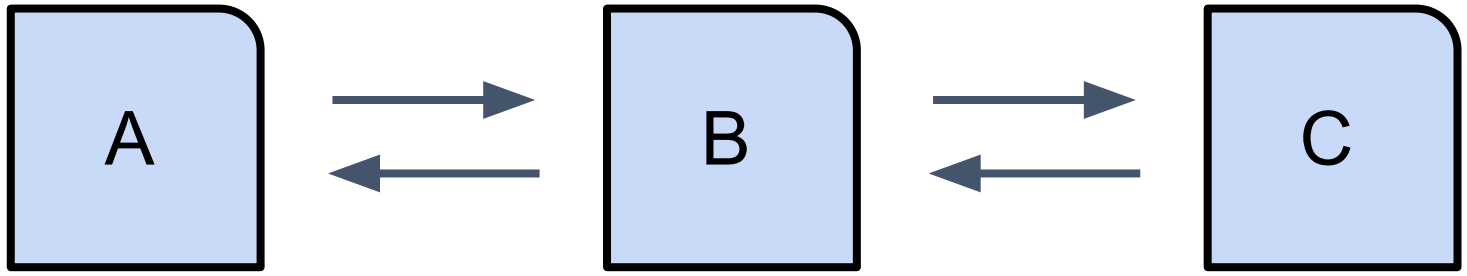
加入資料

- 改變他們指向前後的那些箭頭！



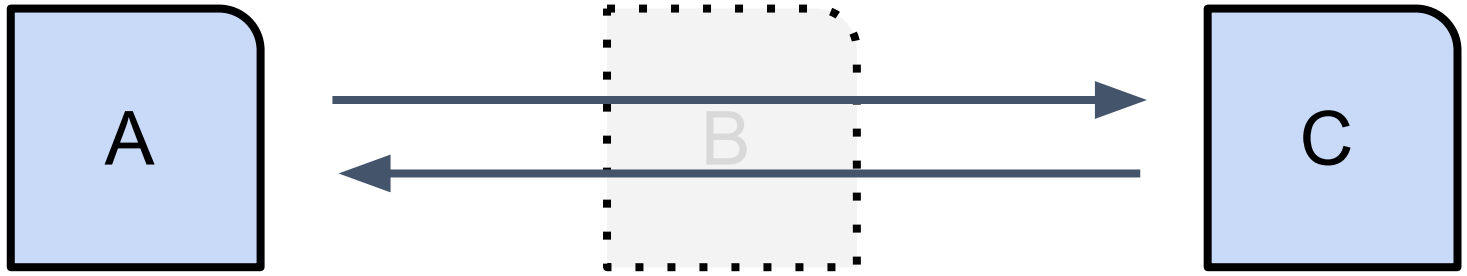
刪除資料

- 假設我們想將資料 B 從資料 A、C 之間刪除



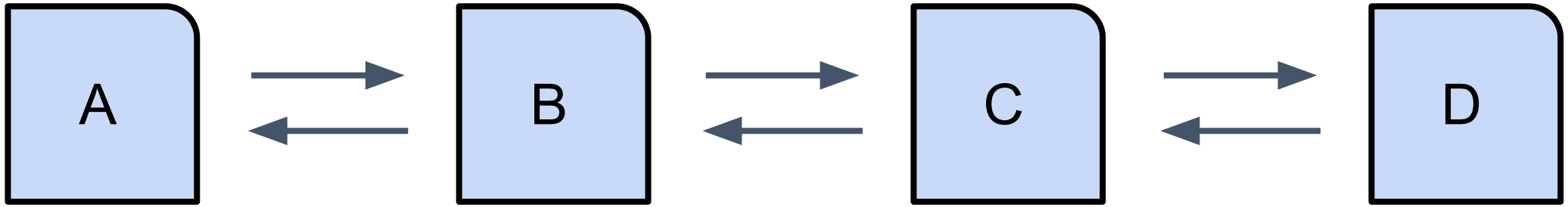
刪除資料

- 假設我們想將資料 B 從資料 A、C 之間刪除



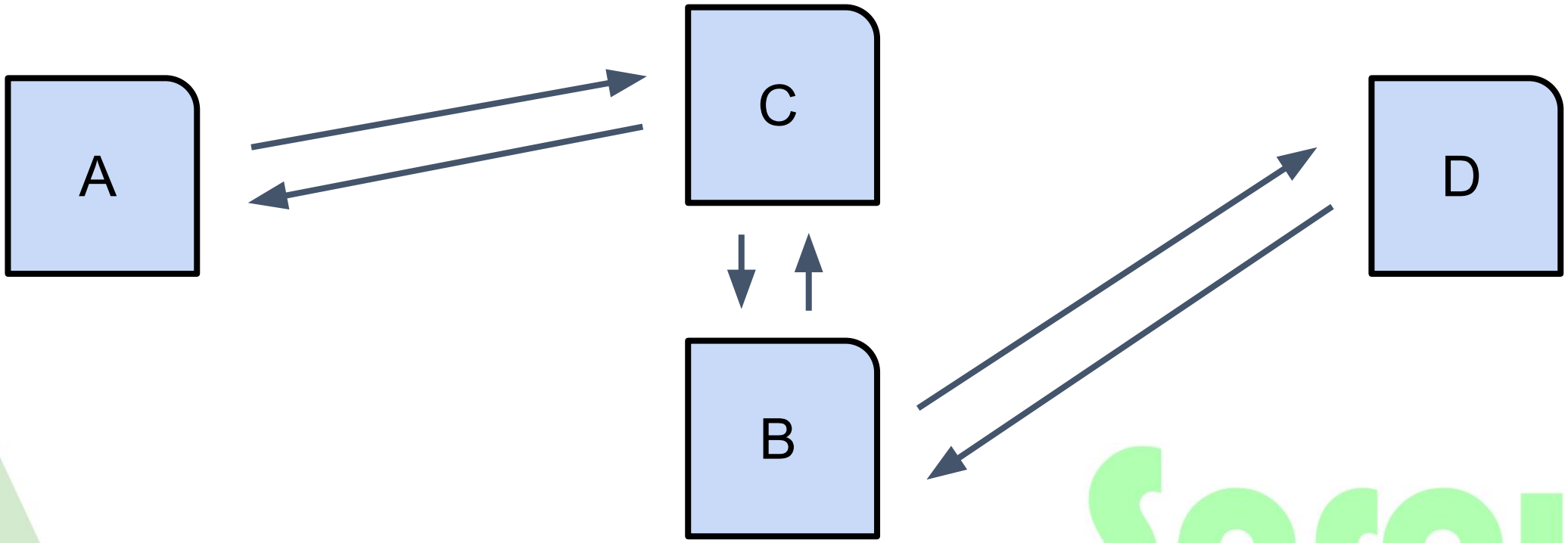
交換資料

- 假設我們想將把 B，C 換位置



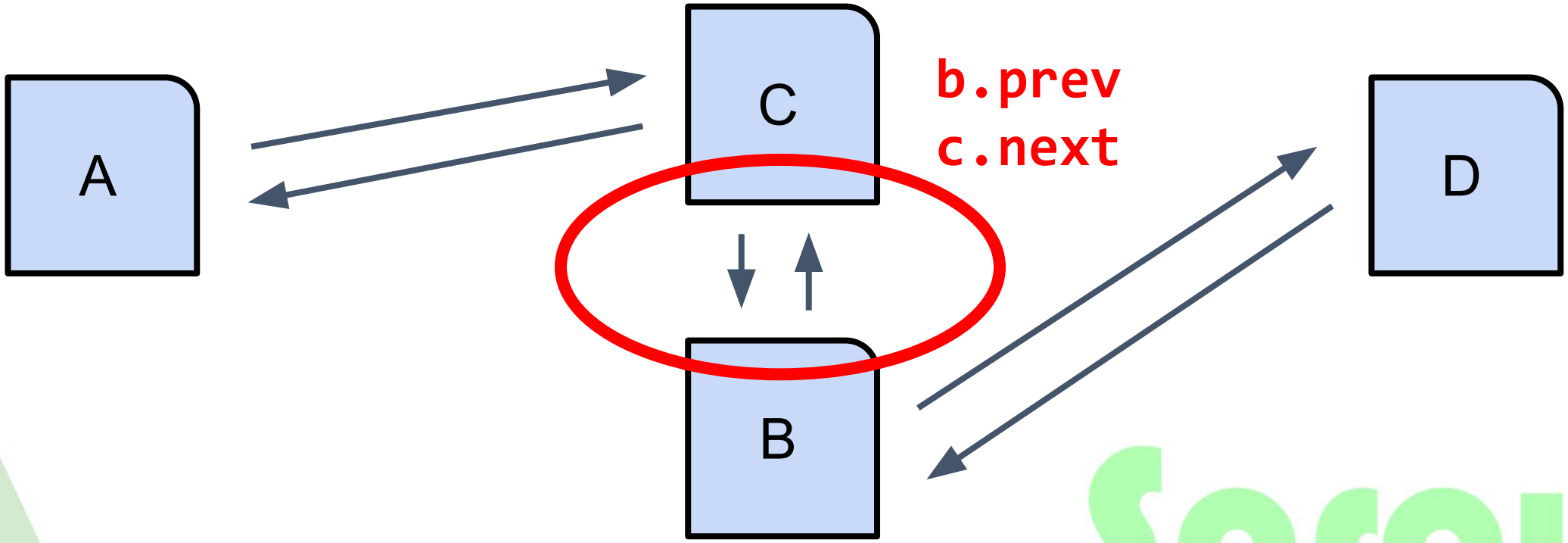
交換資料

- 假設我們想將把 B, C 換位置



交換資料

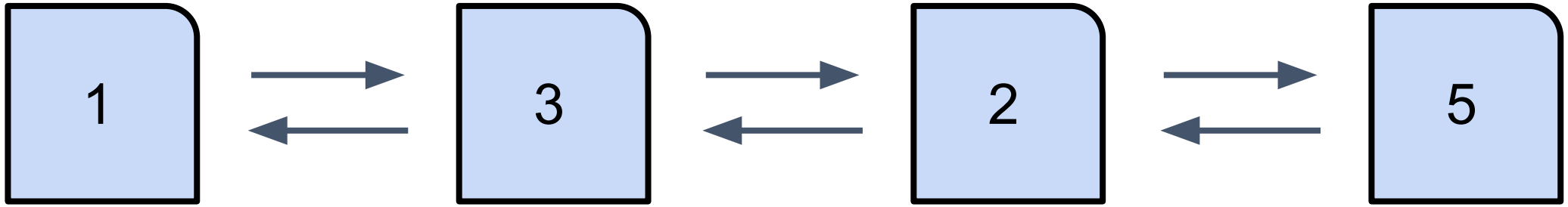
- 假設我們想將把 B, C 換位置



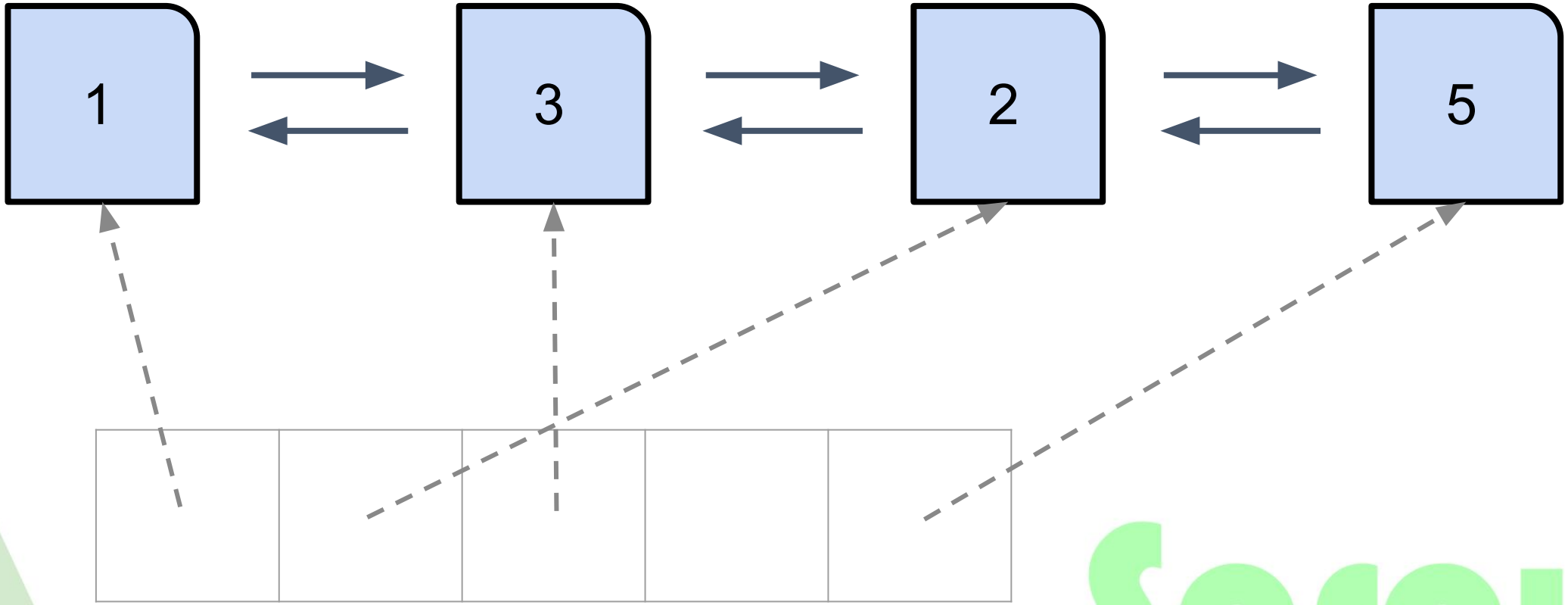
陸行鳥大賽車 prob.21

- 一開始 1~N 依序排好，支援以下操作
- 把一個人淘汰
- 把兩個相鄰的人前後交換

陸行鳥大賽車 prob.21



陸行鳥大賽車 prob.21



謝謝大家！

Sprout