

玩儿转数据结构

liuyubobobo

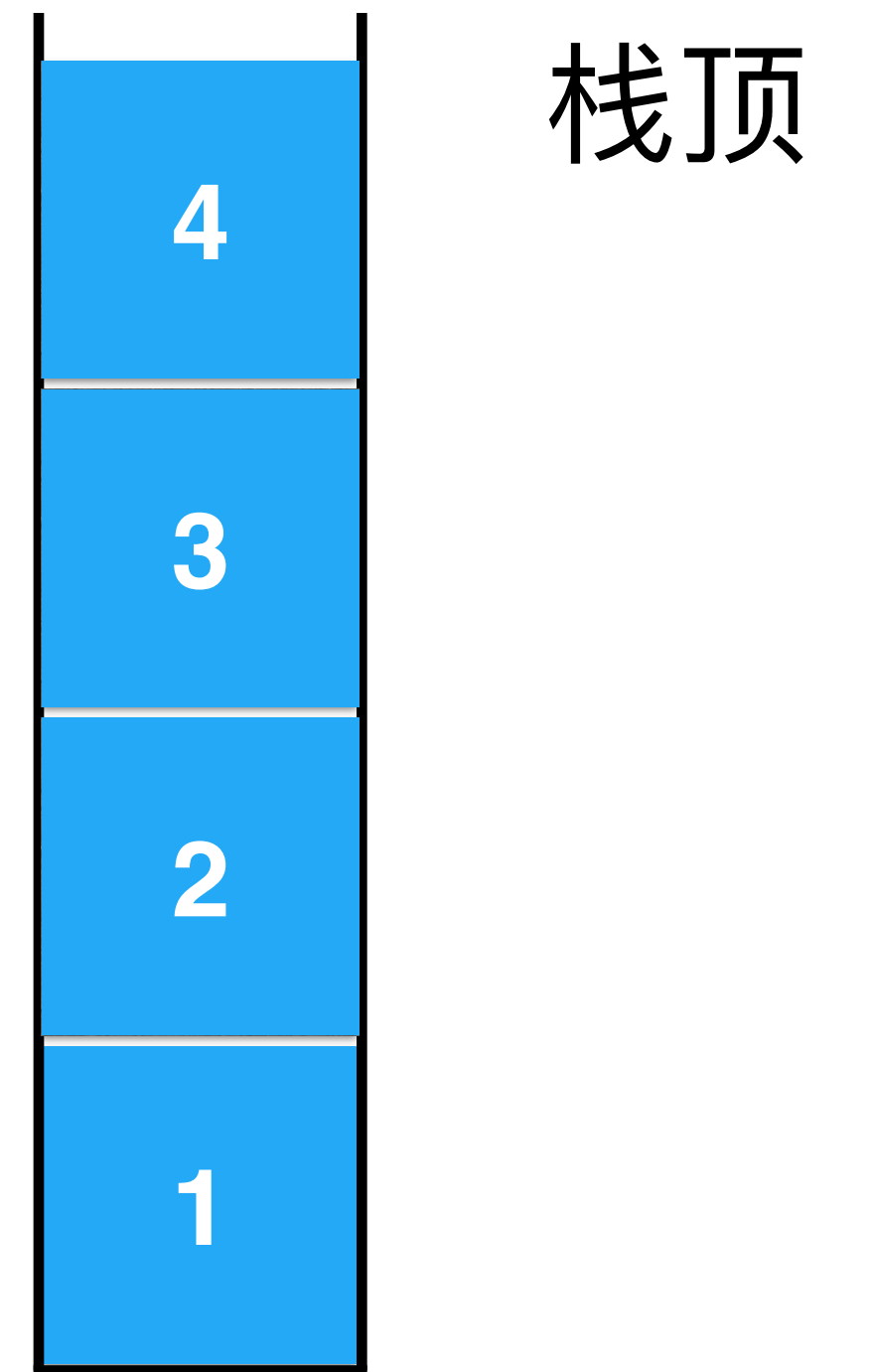
栈和队列

栈 Stack

- 栈也是一种线性结构
- 相比数组，栈对应的操作是数组的子集
- 只能从一端添加元素，也只能从一端取出元素
- 这一端称为栈顶

栈 Stack

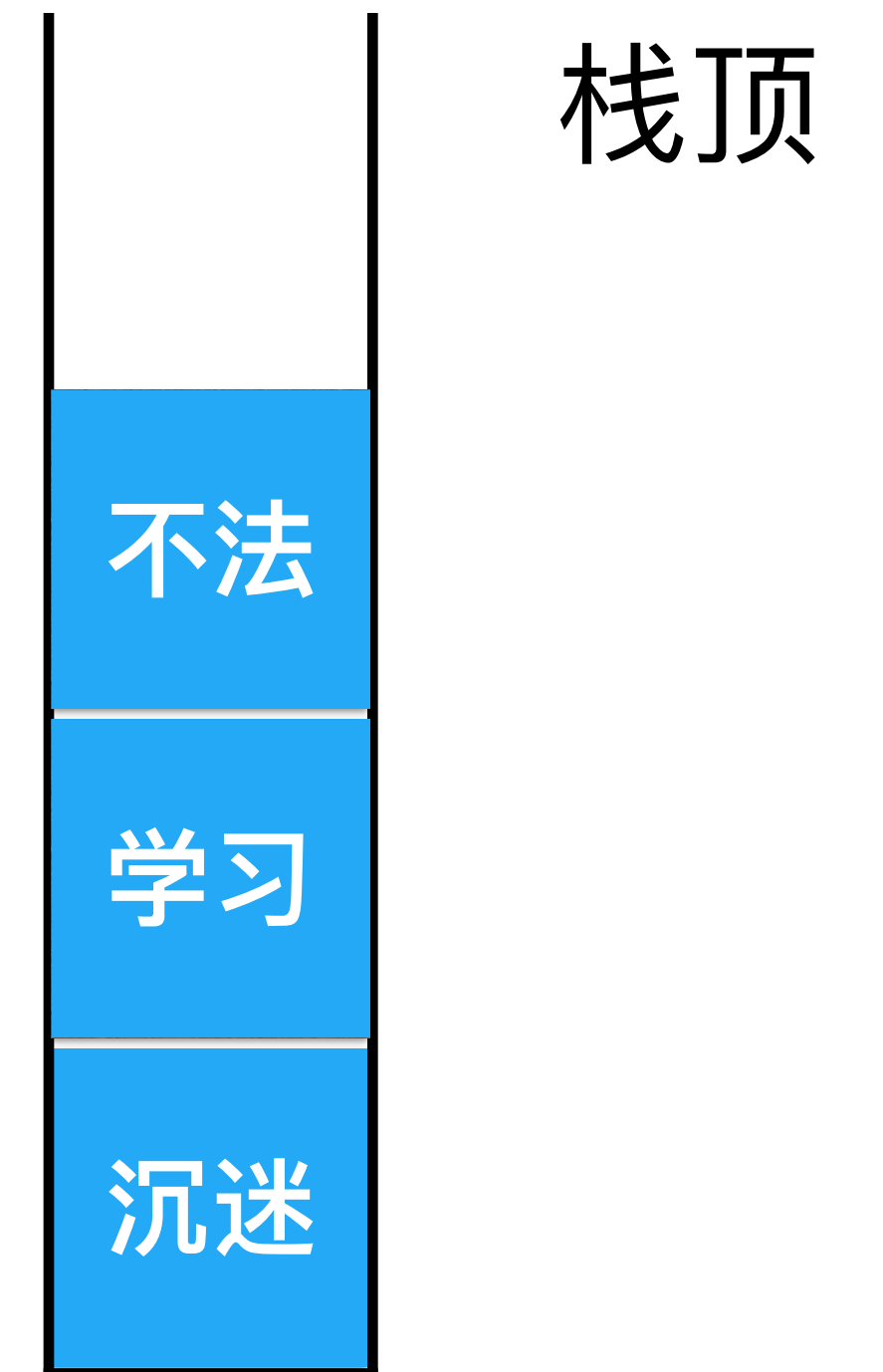
- 栈是一种后进先出的数据结构
- Last In First Out (LIFO)
- 在计算机的世界里，栈拥有着不可思议的作用



栈的应用

- 无处不在的Undo操作（撤销）

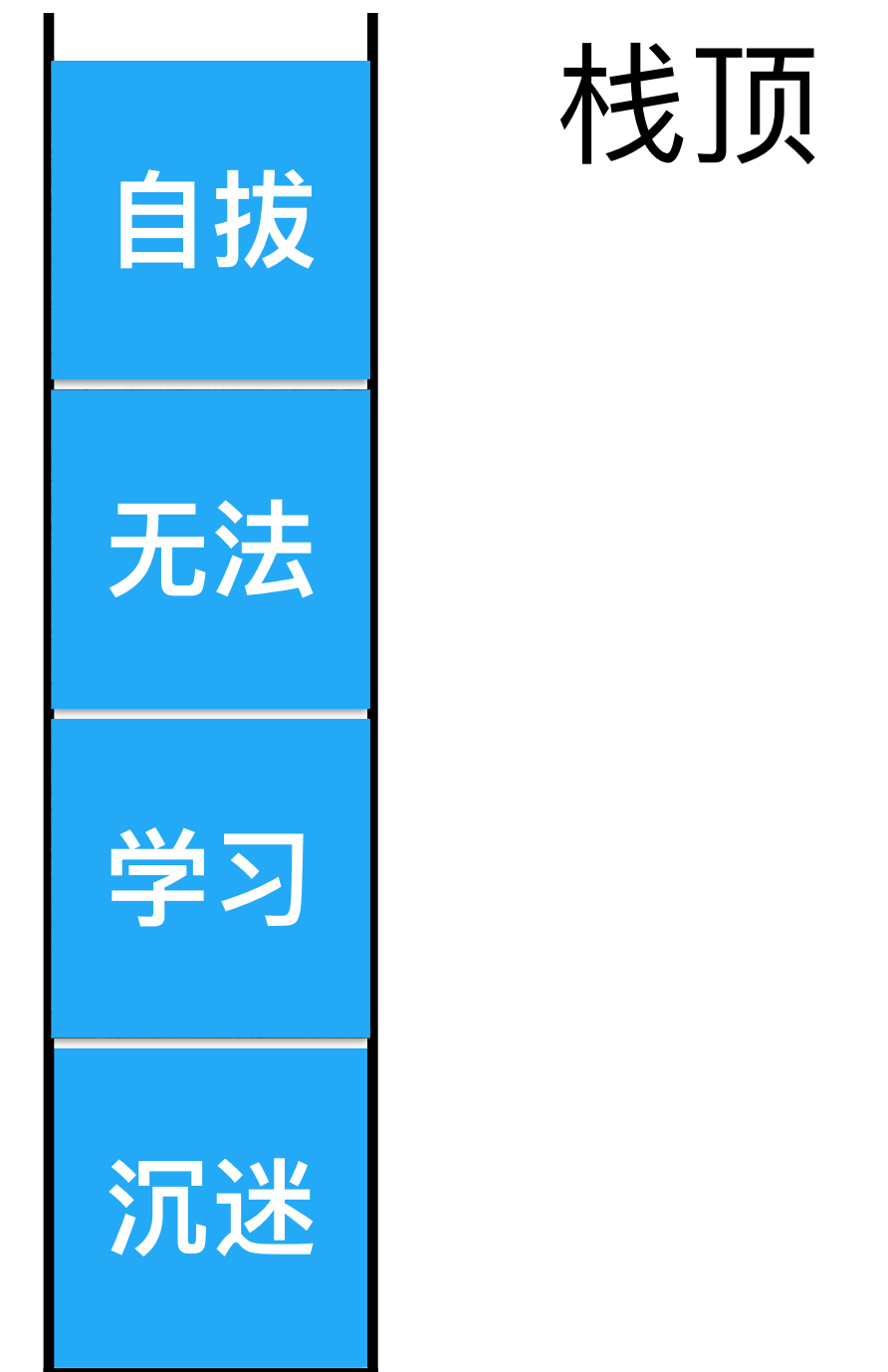
沉迷 学习 不法



栈的应用

- 无处不在的Undo操作（撤销）

沉迷 学习 无法 自拔



栈的应用

- 程序调用的系统栈

→ func A() {
1 ...
2 B()
3 ...
}
|
func B() {
1 ...
2 C()
3 ...
}
|
func C() {
1 ...
2 ...
3 ...
}

栈顶

栈的应用

- 程序调用的系统栈

→ func A() {
1 ...
2 B()
3 ...
}

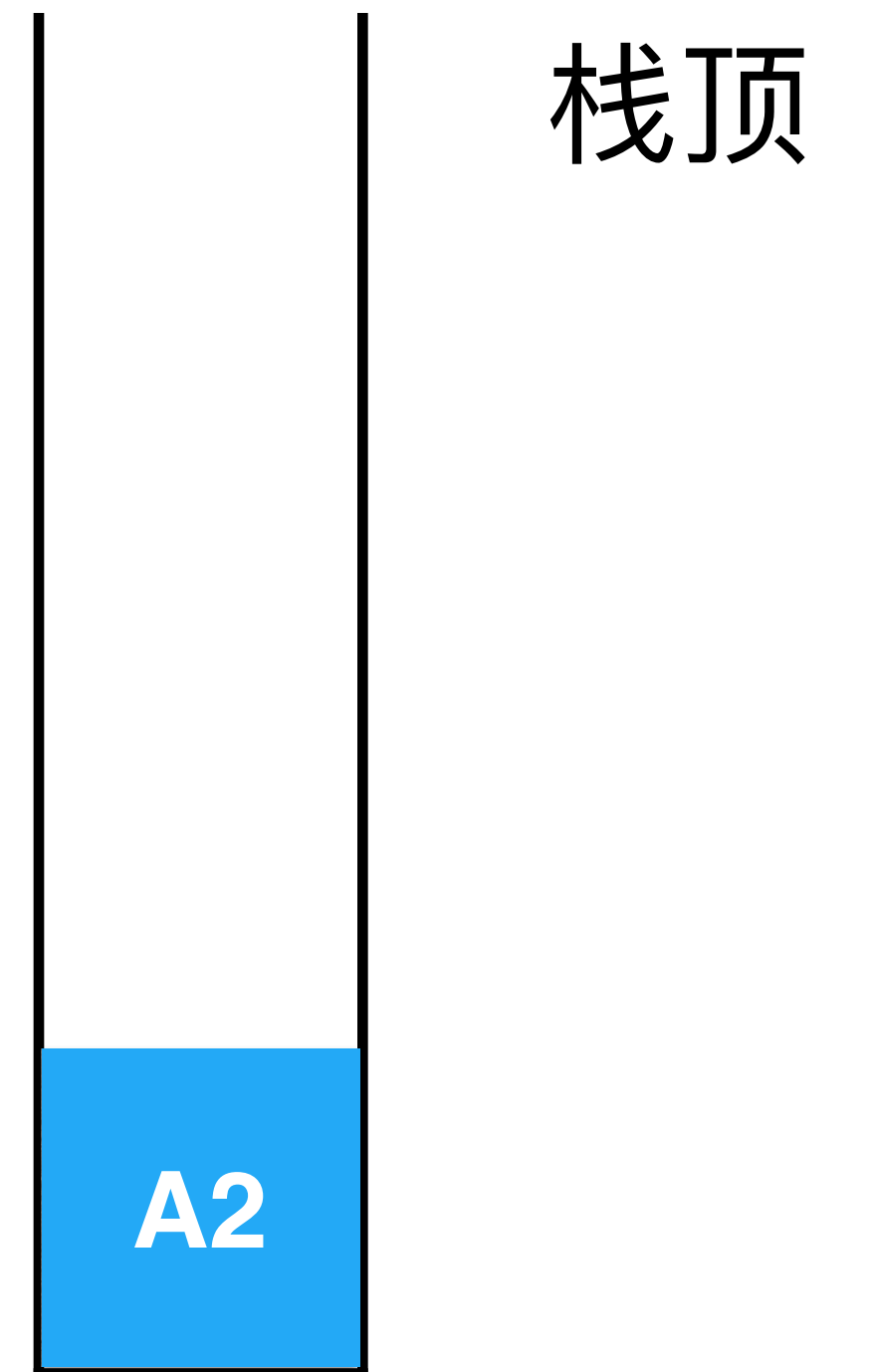
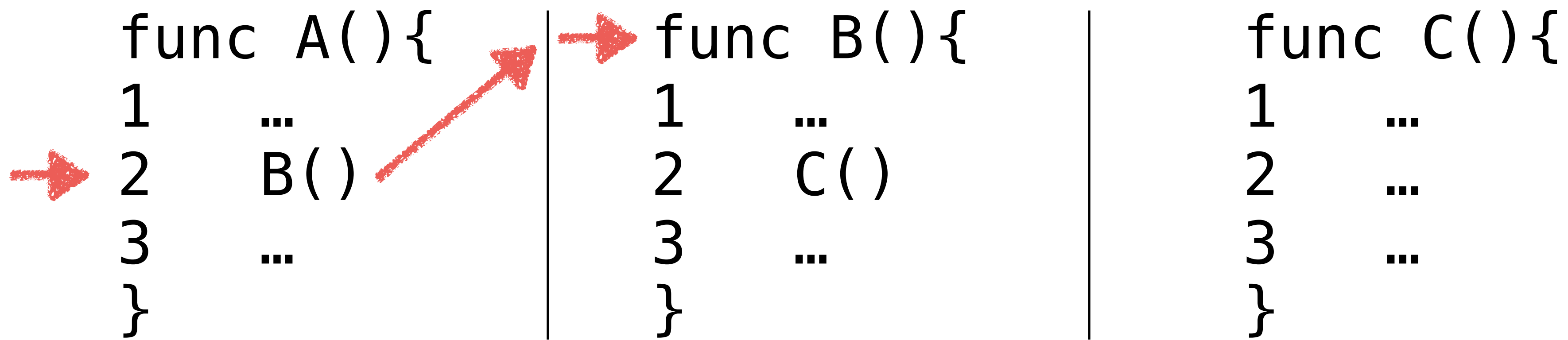
func B() {
1 ...
2 C()
3 ...
}

func C() {
1 ...
2 ...
3 ...
}

栈顶

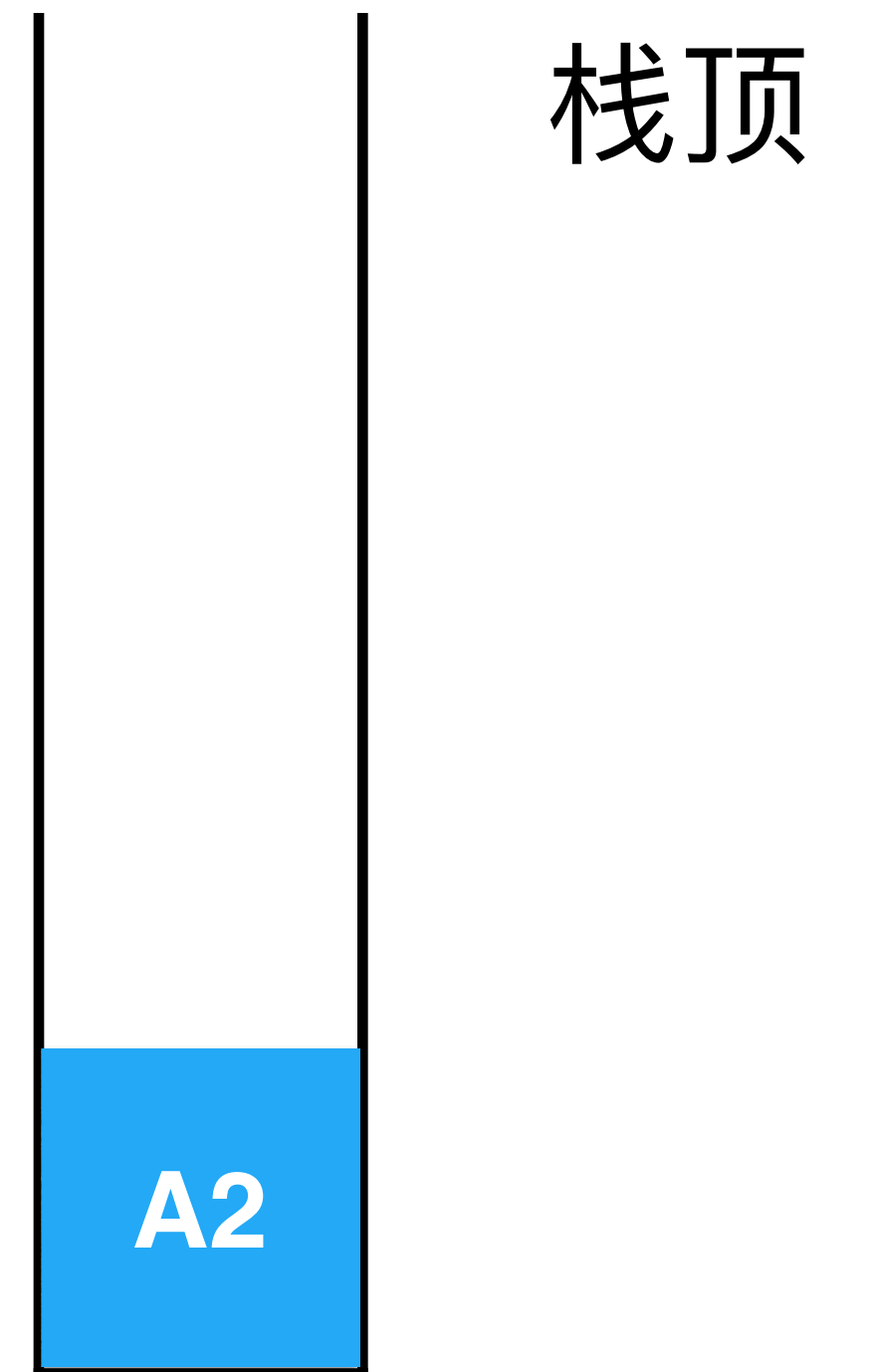
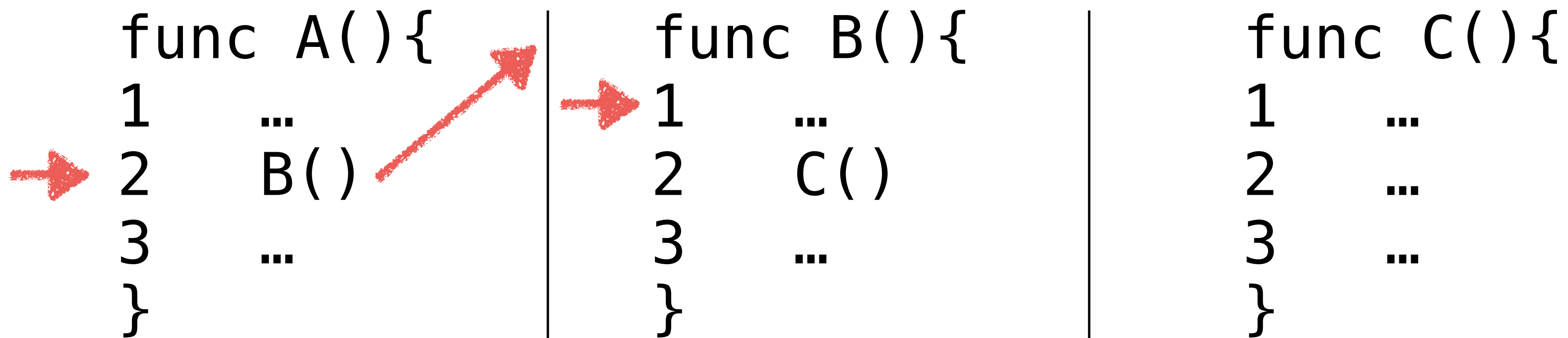
栈的应用

- 程序调用的系统栈



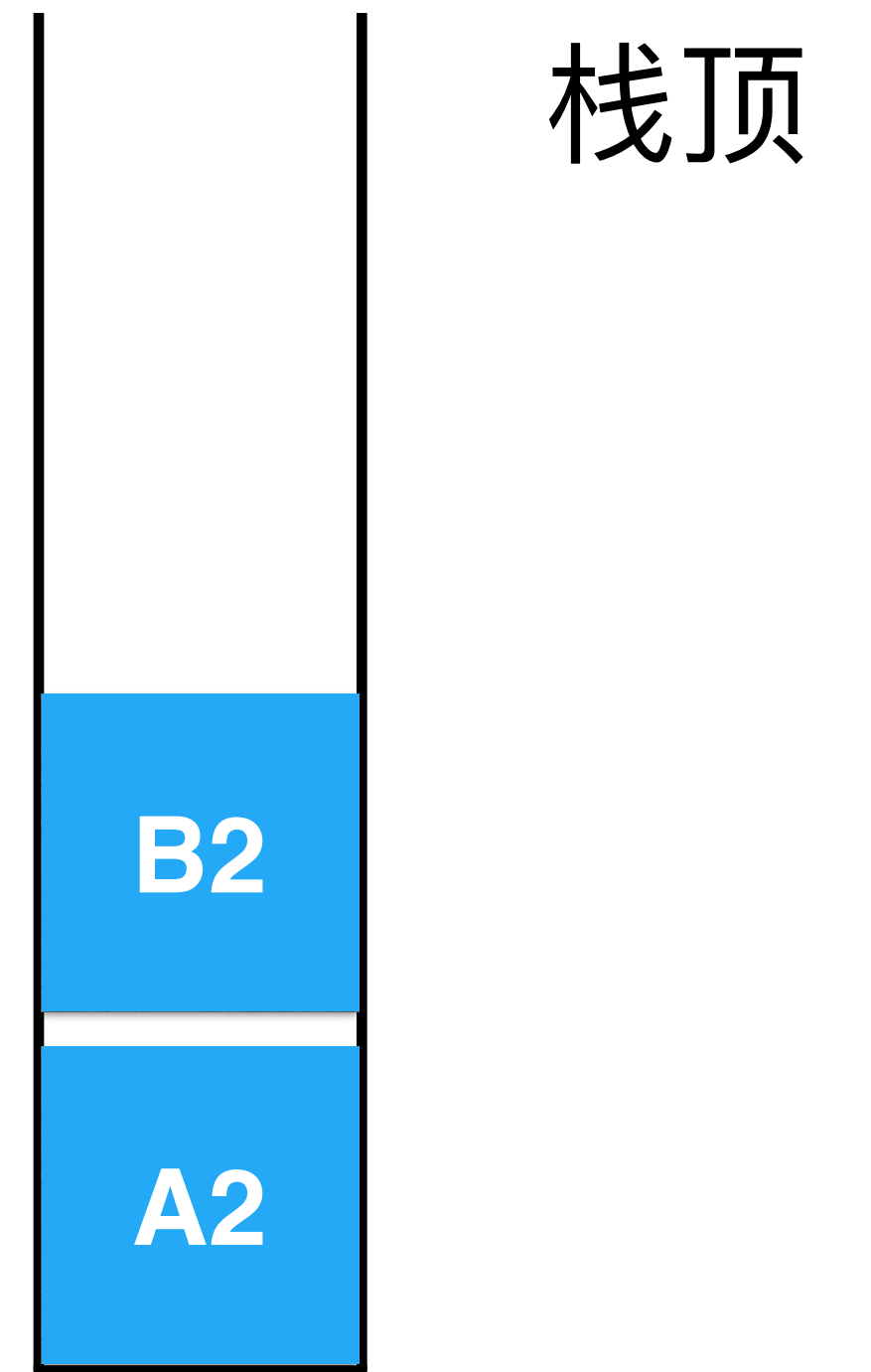
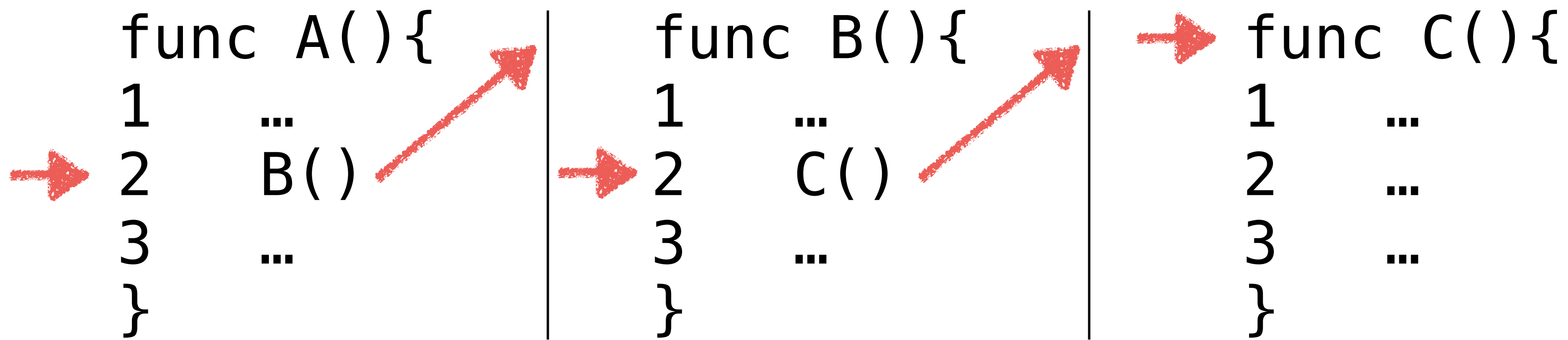
栈的应用

- 程序调用的系统栈



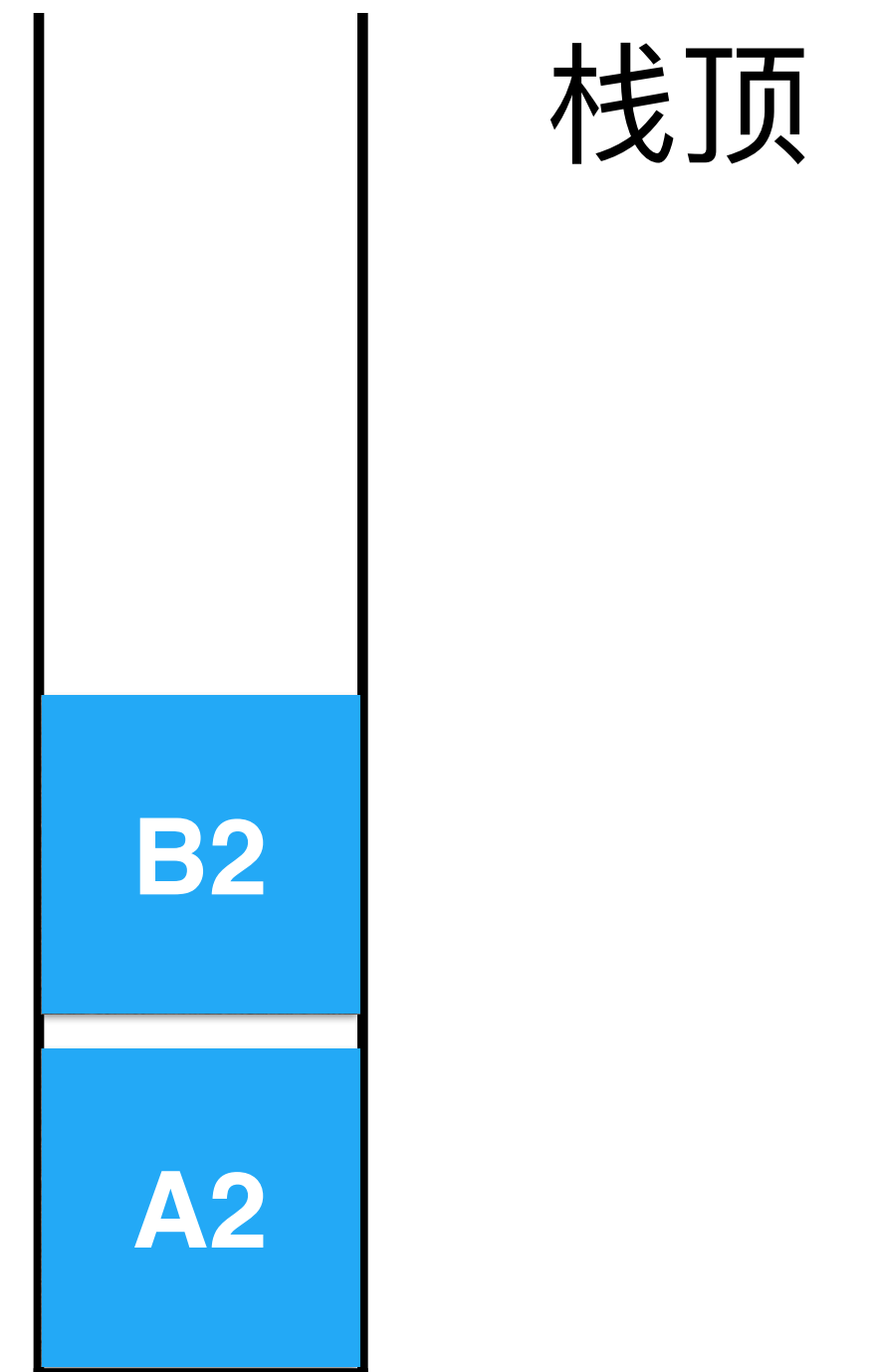
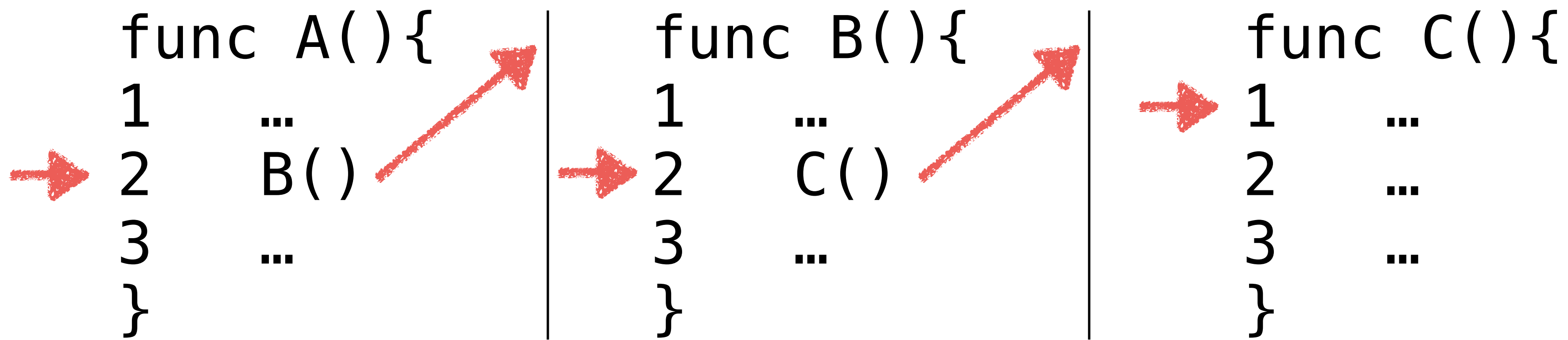
栈的应用

- 程序调用的系统栈



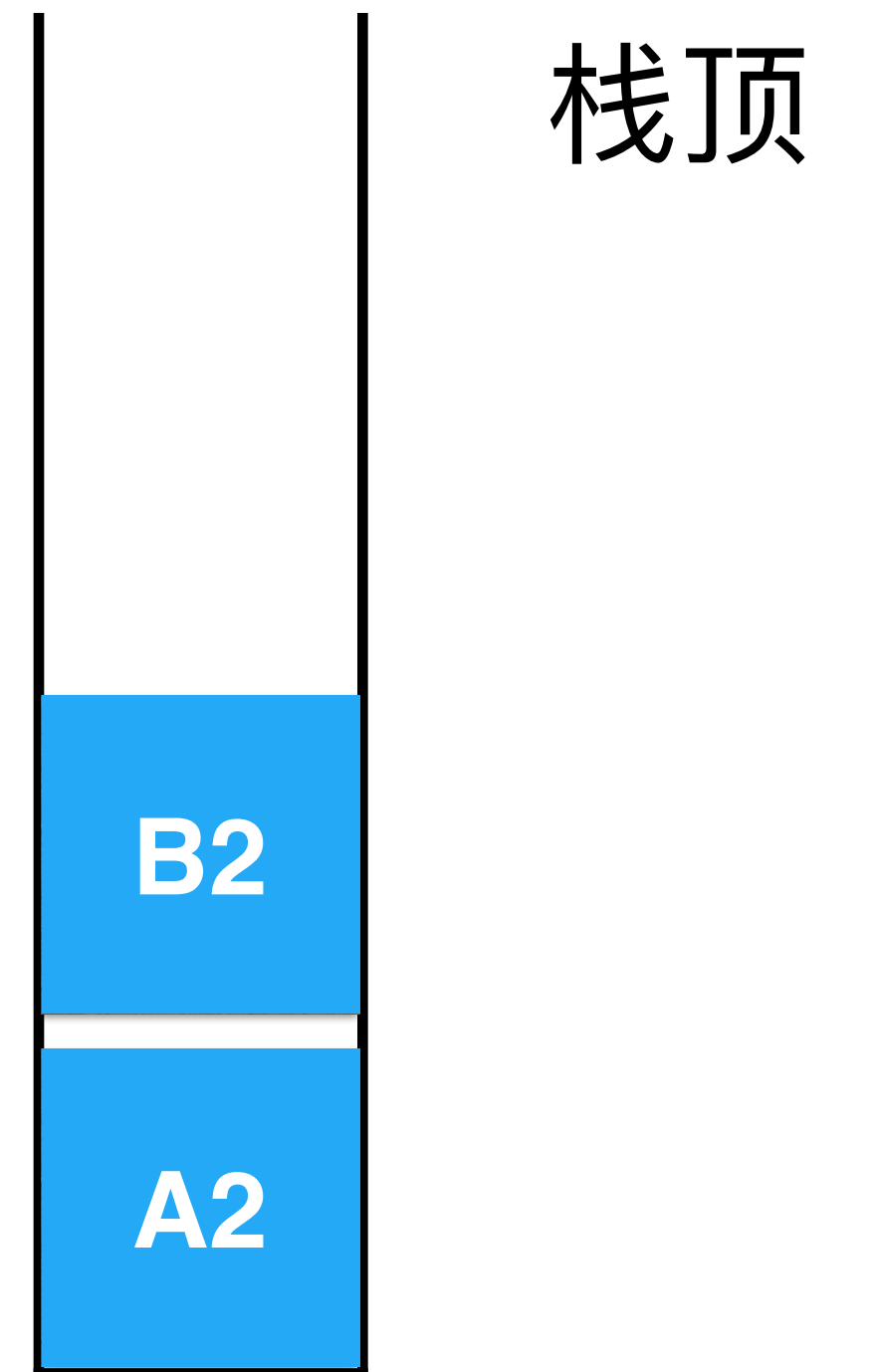
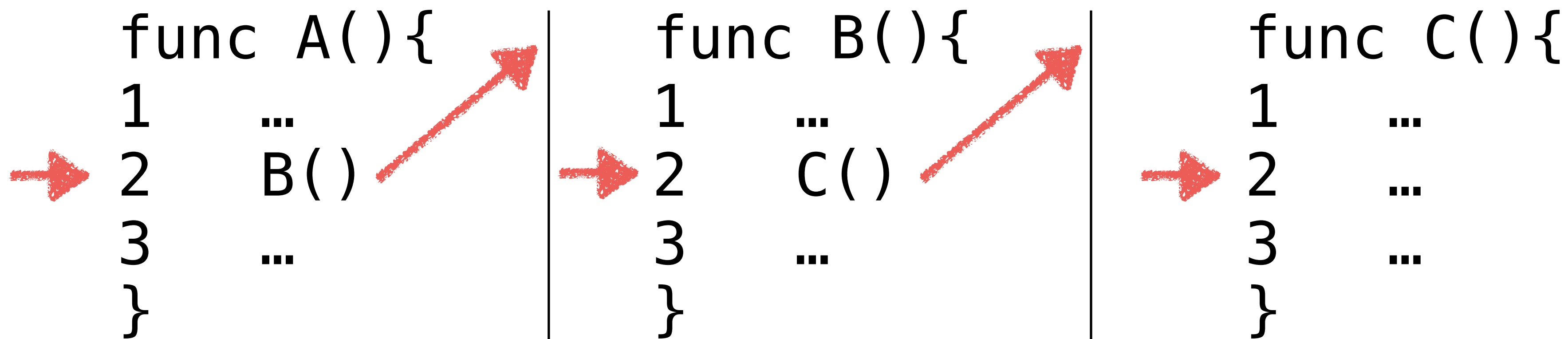
栈的应用

- 程序调用的系统栈



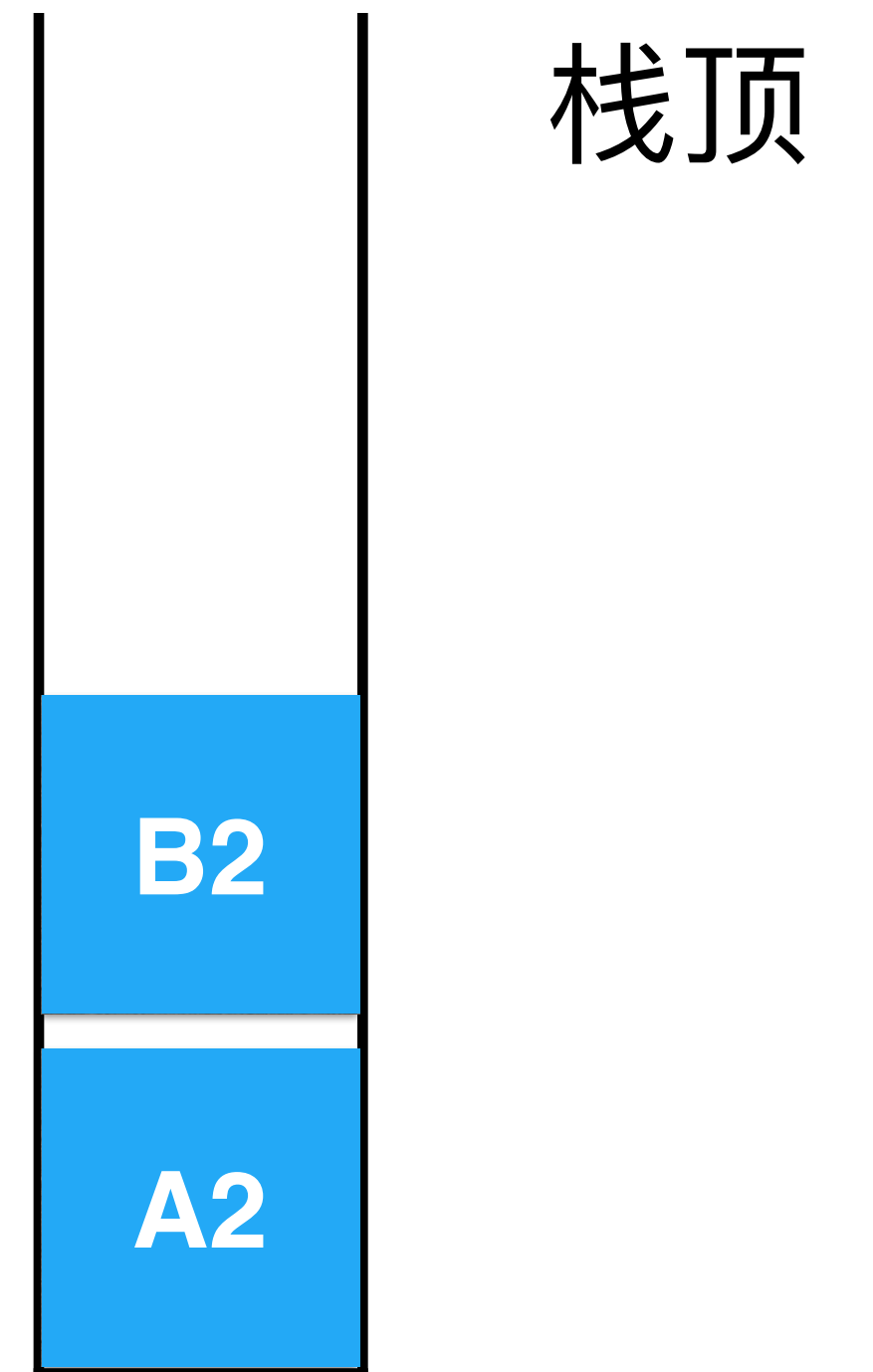
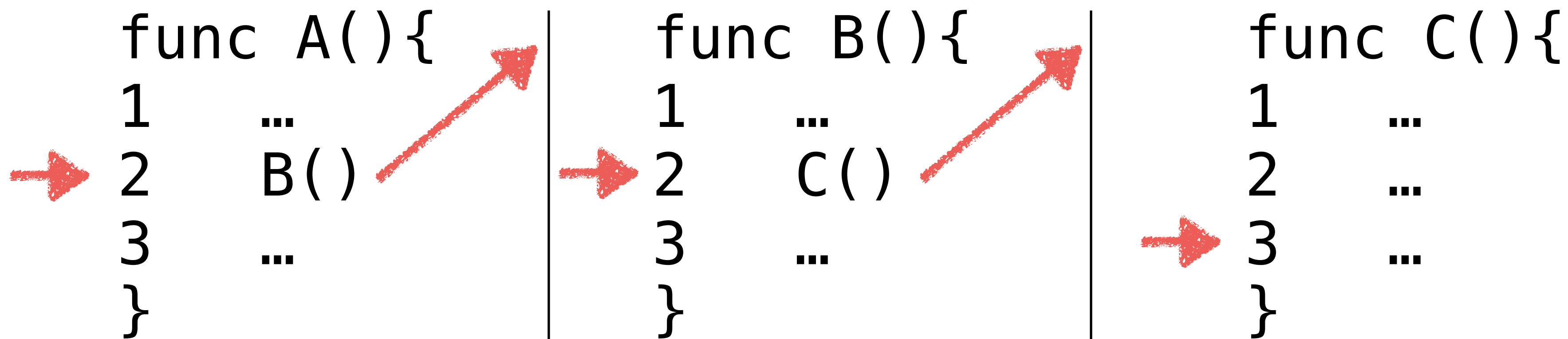
栈的应用

- 程序调用的系统栈



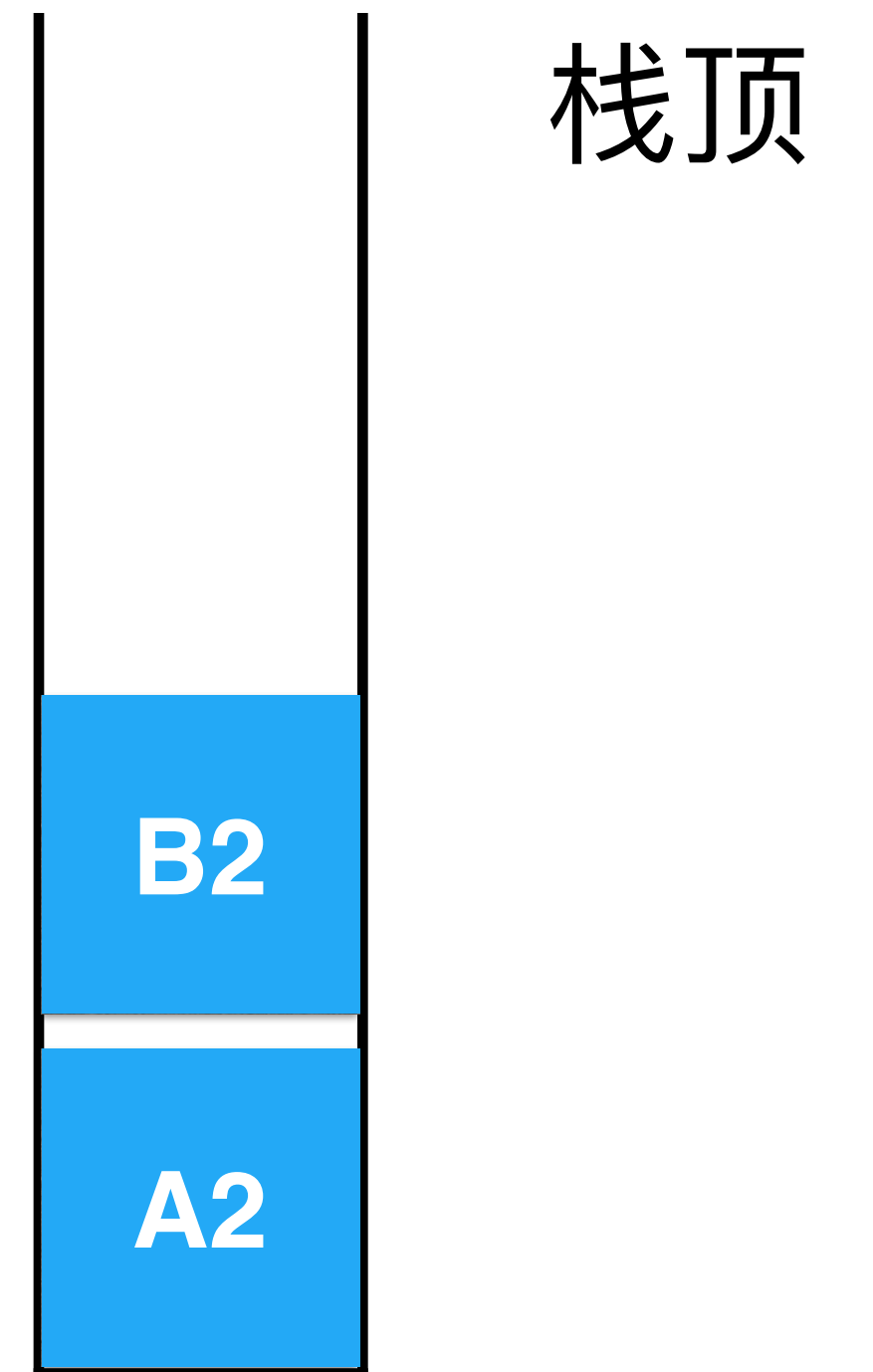
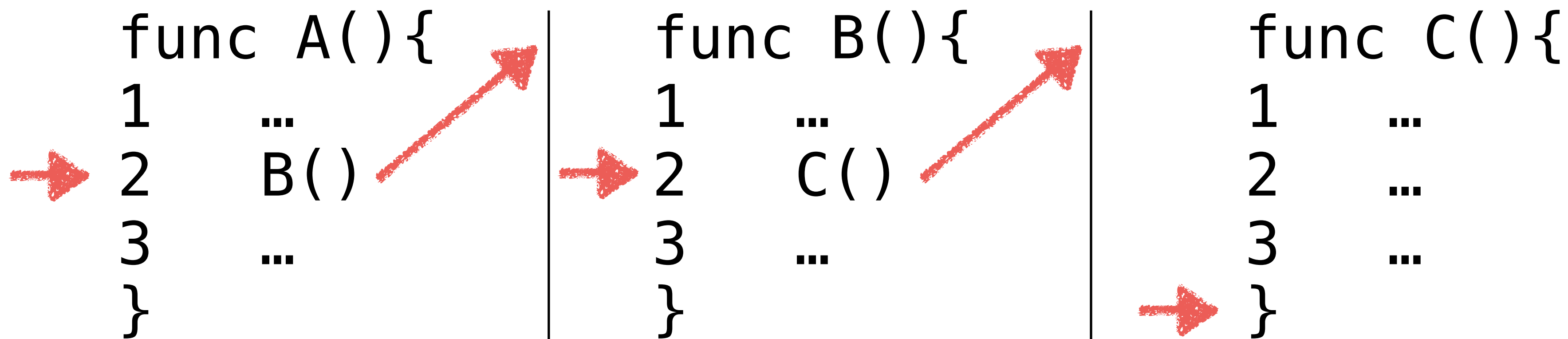
栈的应用

- 程序调用的系统栈



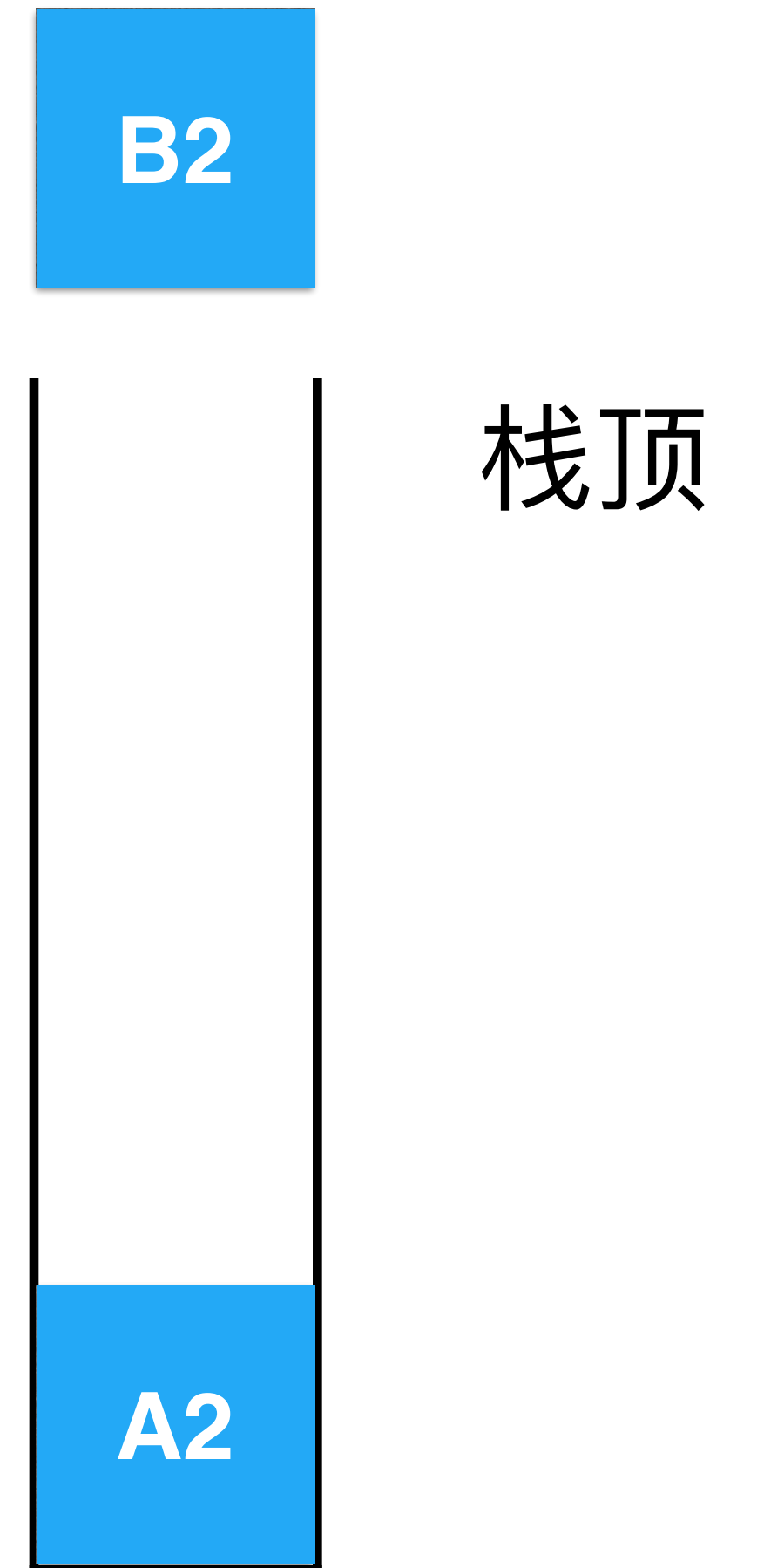
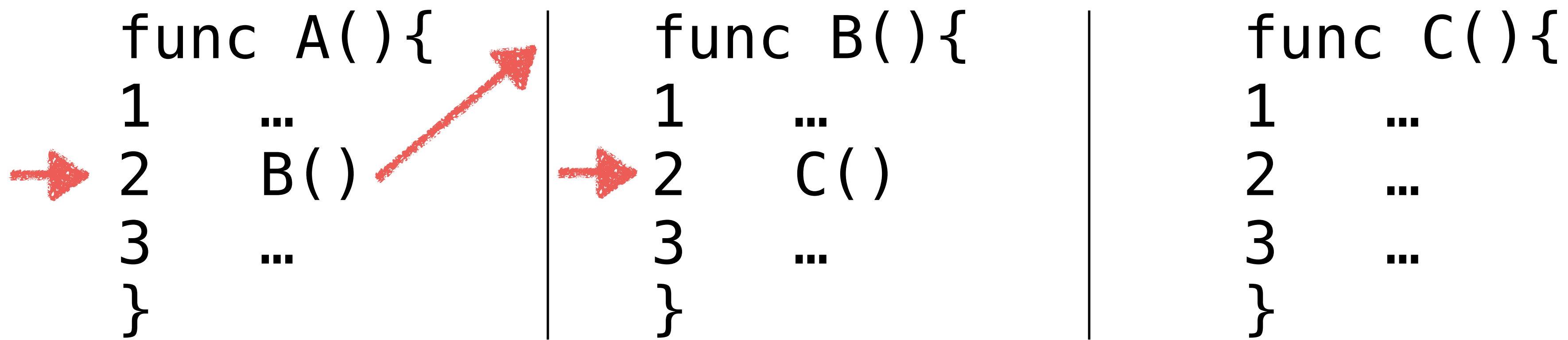
栈的应用

- 程序调用的系统栈



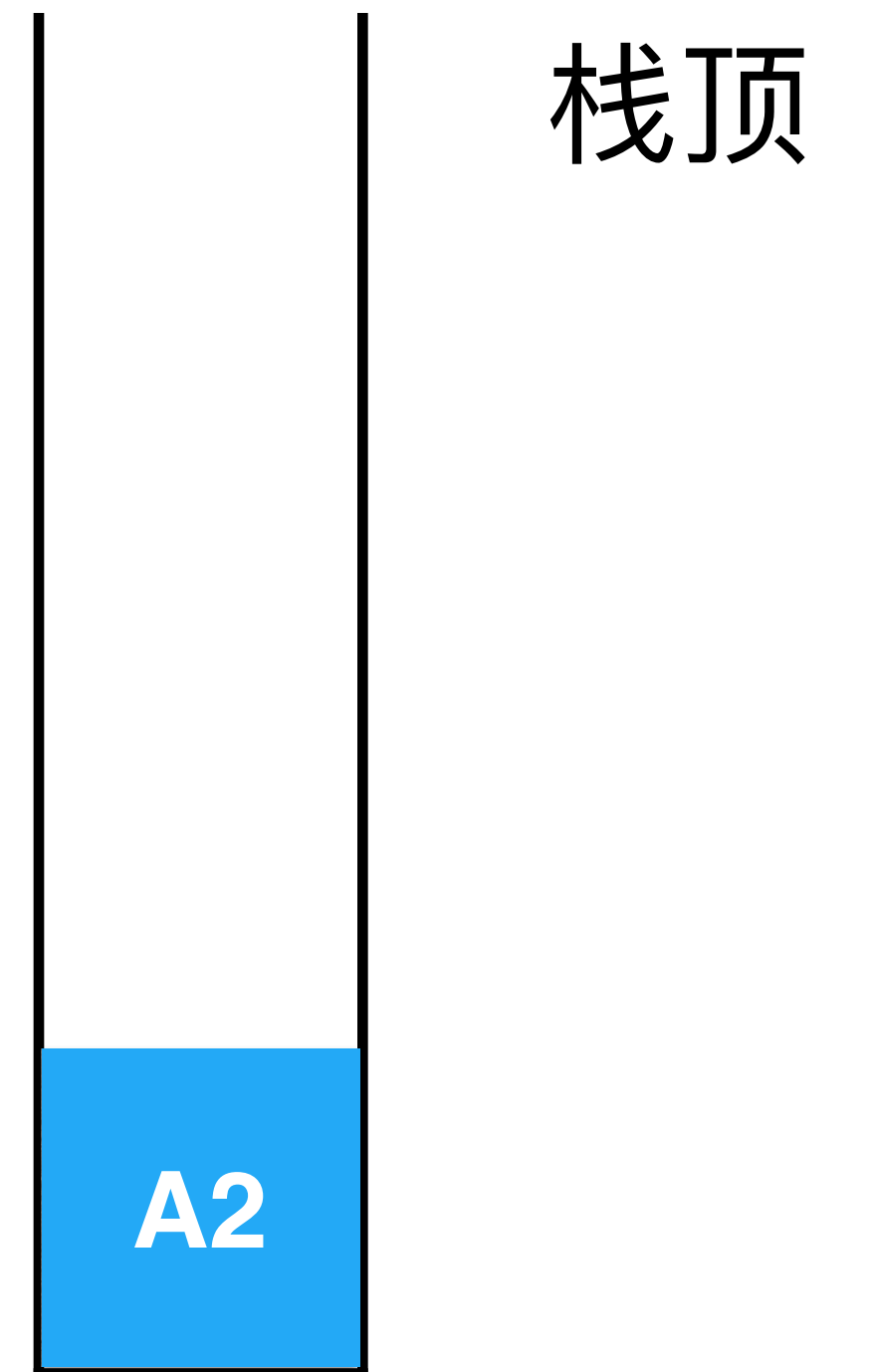
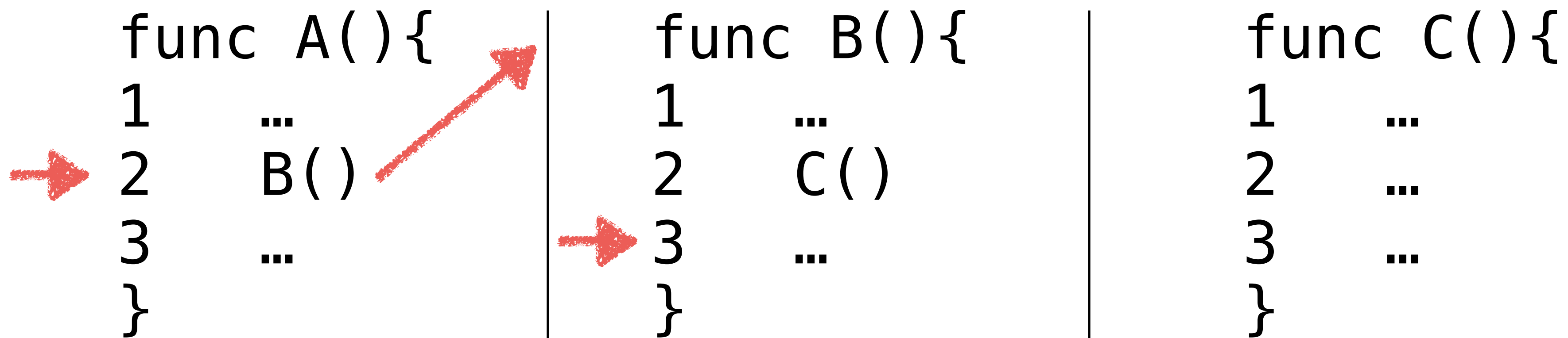
栈的应用

- 程序调用的系统栈



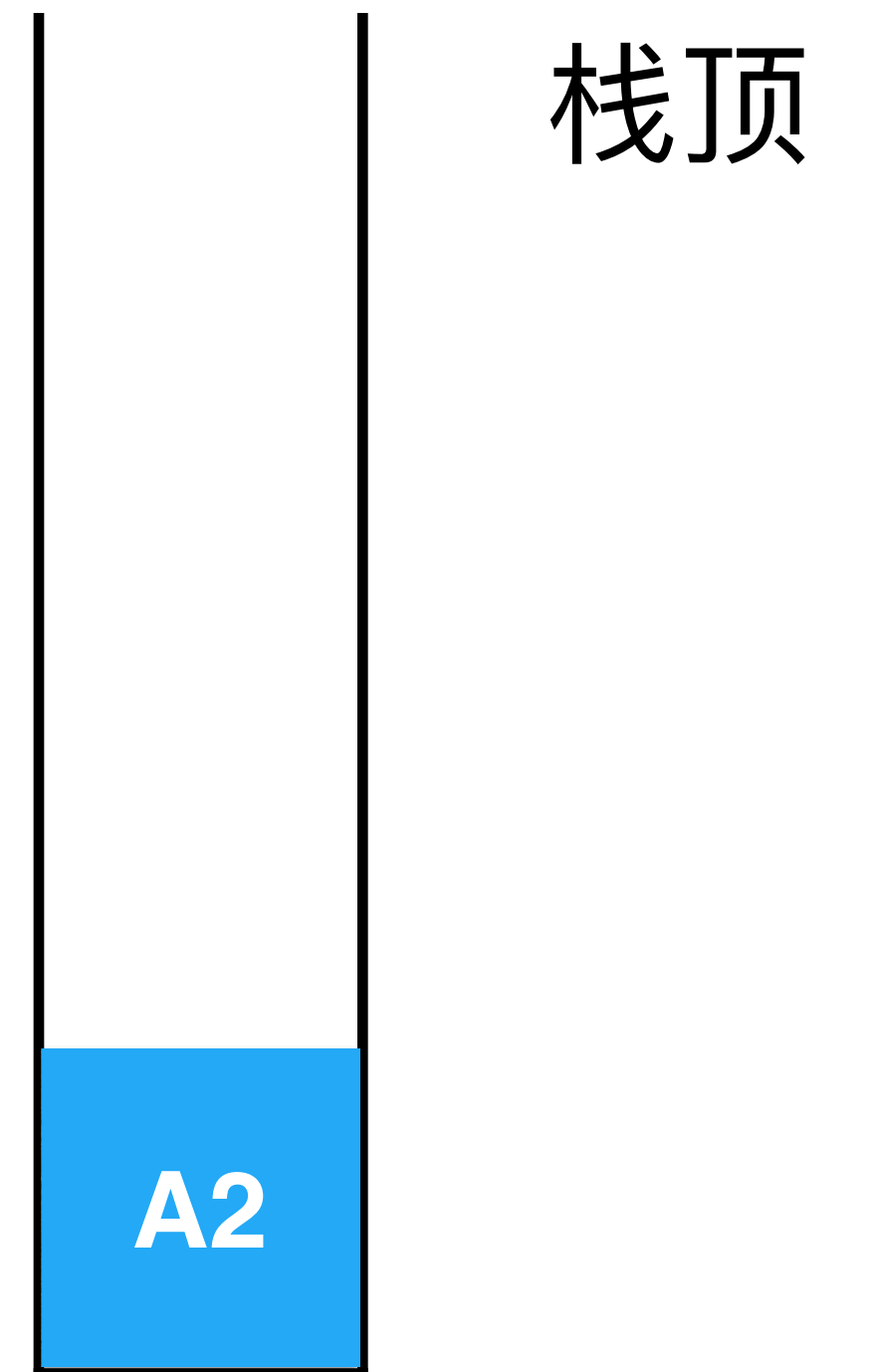
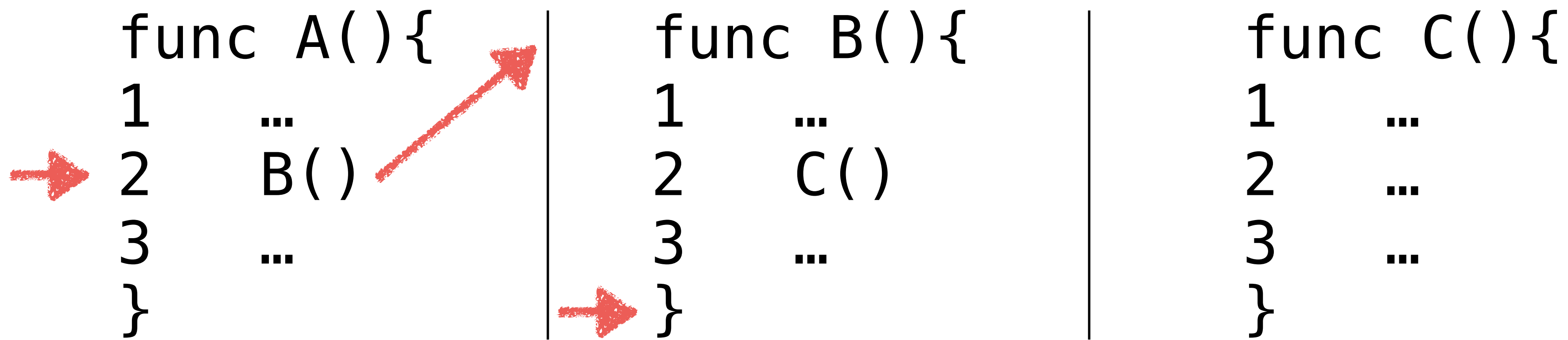
栈的应用

- 程序调用的系统栈



栈的应用

- 程序调用的系统栈



栈的应用

- 程序调用的系统栈

→ func A() {
1 ...
2 B()
3 ...
}

func B() {
1 ...
2 C()
3 ...
}


func C() {
1 ...
2 ...
3 ...
}

A2

栈顶

栈的应用

- 程序调用的系统栈




func A() {	func B() {	func C() {
1 ...	1 ...	1 ...
2 B()	2 C()	2 ...
3 ...	3 ...	3 ...
}	}	}

栈顶

栈的应用

- 程序调用的系统栈

	<pre>func A() { 1 ... 2 B() 3 ... }</pre>		<pre>func B() { 1 ... 2 C() 3 ... }</pre>		<pre>func C() { 1 ... 2 ... 3 ... }</pre>
--	--	--	--	--	--

栈顶

栈的实现

栈的实现

Stack<E>


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

栈的实现

Stack<E>

- `void push(E)`
 - `E pop()`
 - `E peek()`
 - `int getSize()`
 - `boolean isEmpty()`
- 从用户的角度看，支持这些操作就好
 - 具体底层实现，用户不关心
 - 实际底层有多种实现方式

栈的实现

Interface Stack<E>  ArrayStack<E>

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

实践：栈的实现

栈的复杂度分析

ArrayStack<E>

- `void push(E)` $O(1)$ 均摊
- `E pop()` $O(1)$ 均摊
- `E peek()` $O(1)$
- `int getSize()` $O(1)$
- `boolean isEmpty()` $O(1)$

栈的应用

栈的应用

- undo 操作 - 编辑器
- 系统调用栈 - 操作系统
- 括号匹配 - 编译器

Leetcode



LeetCode

- leetcode.com
- leetcode-cn.com

20. Valid Parentheses



20. Valid Parentheses

给定一个字符串，只包含 (, [, {,),], }, 判定字符串中的括号匹配是否合法。

- 如 "()", "()[]{}" 是合法的
- 如 "[", "([)]" 是非合法的

20. Valid Parentheses


{ [()] }

Stack

A horizontal rectangular box with a black border, representing a stack data structure. It is currently empty.

20. Valid Parentheses

{ [()] }



Stack

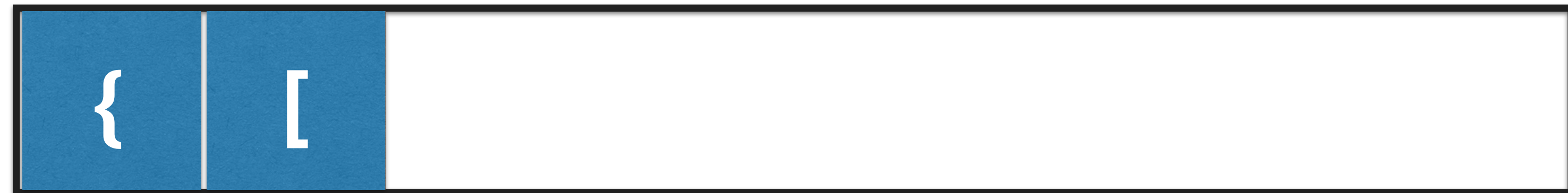


20. Valid Parentheses

{ [()] }



Stack



20. Valid Parentheses

{ [()] }

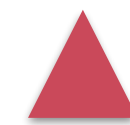


Stack

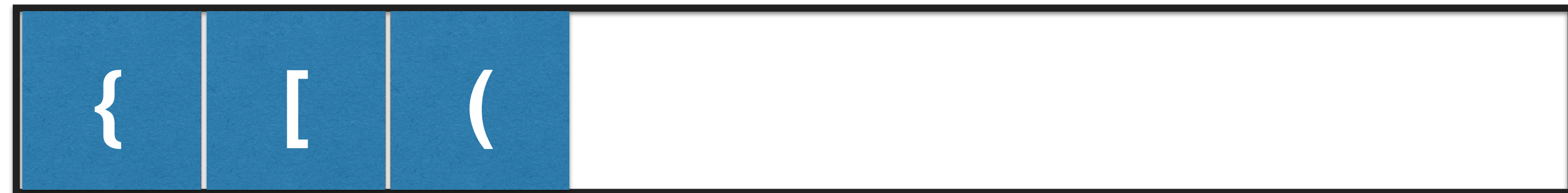


20. Valid Parentheses

{ [()] }



Stack



20. Valid Parentheses

{ [()] }



Stack

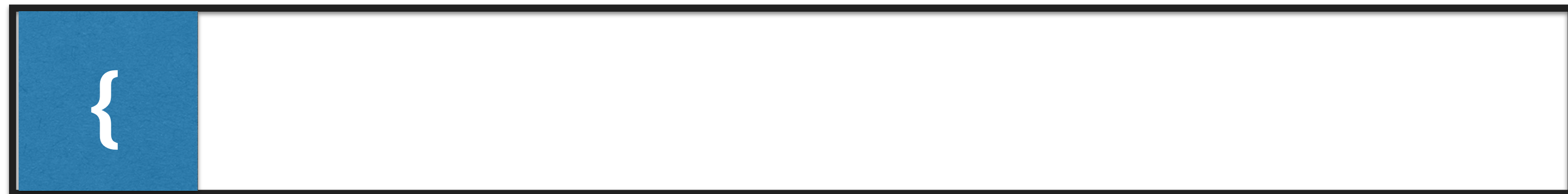


20. Valid Parentheses

{ [()] }



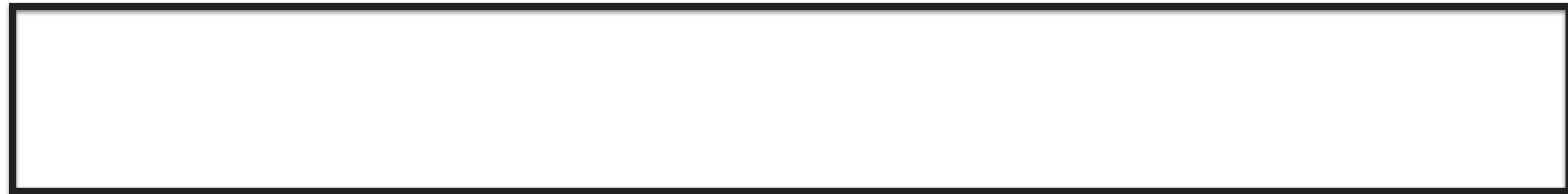
Stack



20. Valid Parentheses


{ [}]

Stack



20. Valid Parentheses

{ [}]



Stack



20. Valid Parentheses

{ [}]

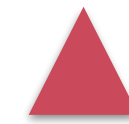


Stack

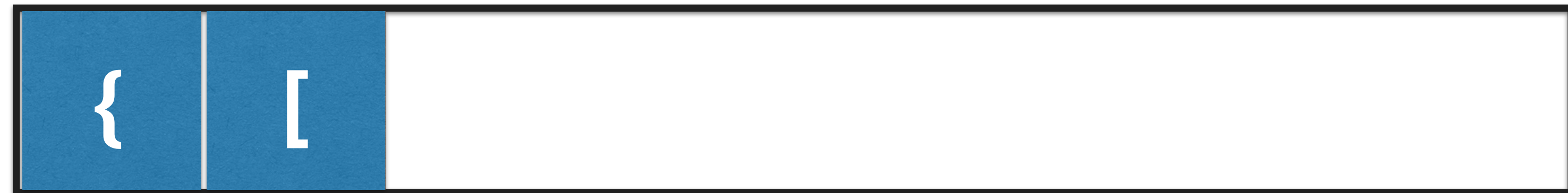


20. Valid Parentheses

{ [}]



Stack



栈顶元素反映了在嵌套的层次关系中，**最近的**需要匹配的元素

实践： Leetcode 20

更多和Leetcode相关的问题

实践： 测试自己的Solution

实践： 想要测试我们自己的Stack类？

展示：更多Leetcode上stack相关的问题

学习方法讨论

- 不要完美主义。掌握好“度”。
- 学习本着自己的目标去。
- 对于这个课程，大家的首要目标，是了解各个数据结构的底层实现原理

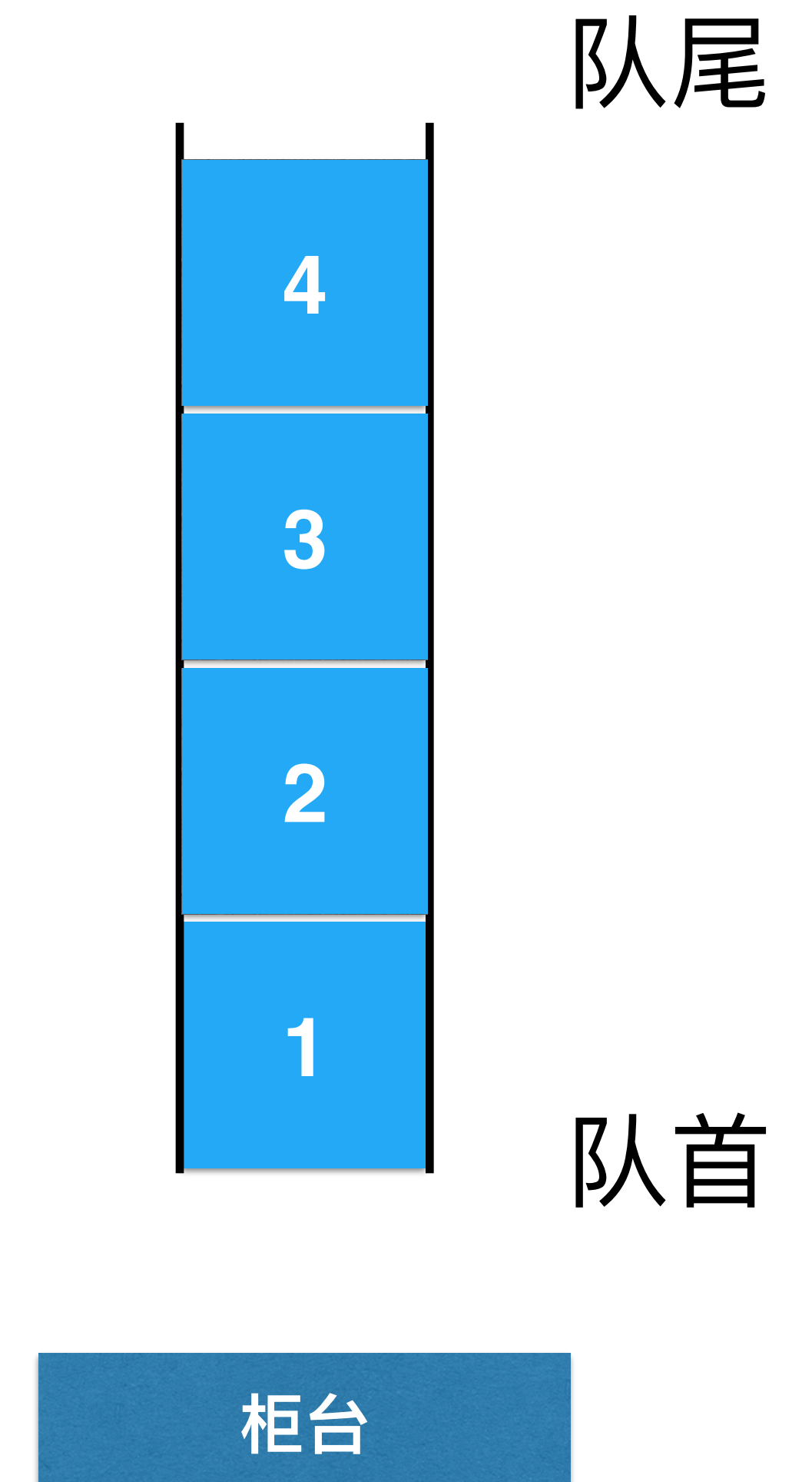
队列

队列 Queue

- 队列也是一种线性结构
- 相比数组，队列对应的操作是数组的子集
- 只能从一端（队尾）添加元素，只能从另一端（队首）取出元素

队列 Queue

- 队列是一种先进先出的数据结构（先到先得）
- First In First Out (FIFO)




队列的实现

Queue<E>

- `void enqueue(E)`
- `E dequeue()`
- `E getFront()`
- `int getSize()`
- `boolean isEmpty()`

队列的实现

Interface Queue<E>  ArrayQueue<E>

- void enqueue(E) implement
- E dequeue()
- E getFront()
- int getSize()
- boolean isEmpty()

实践：数组队列的实现

数组队列的复杂度分析

ArrayQueue<E>

- void enqueue(E) $O(1)$ 均摊
- E dequeue() $O(n)$
- E getFront() $O(1)$
- int getSize() $O(1)$
- boolean isEmpty() $O(1)$

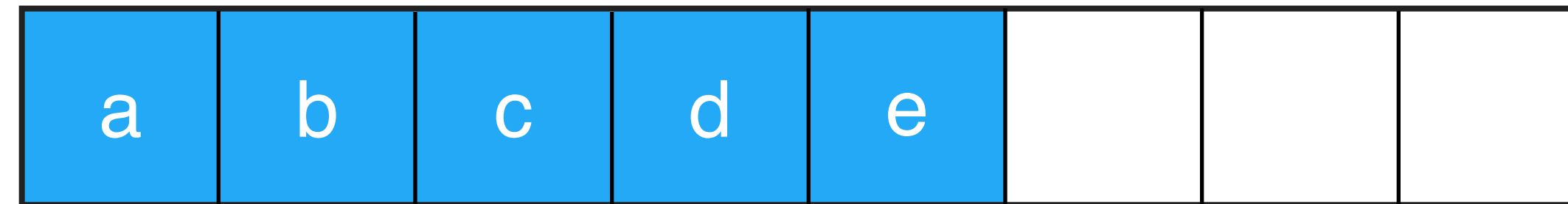
循环队列

数组队列的问题

删除队首元素

data

size

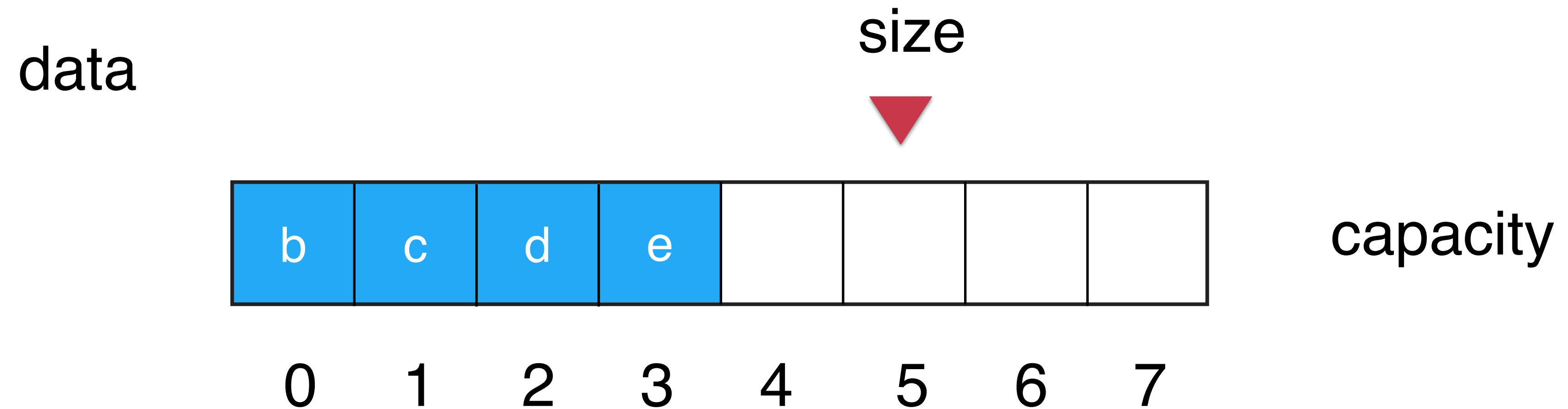


capacity

0 1 2 3 4 5 6 7

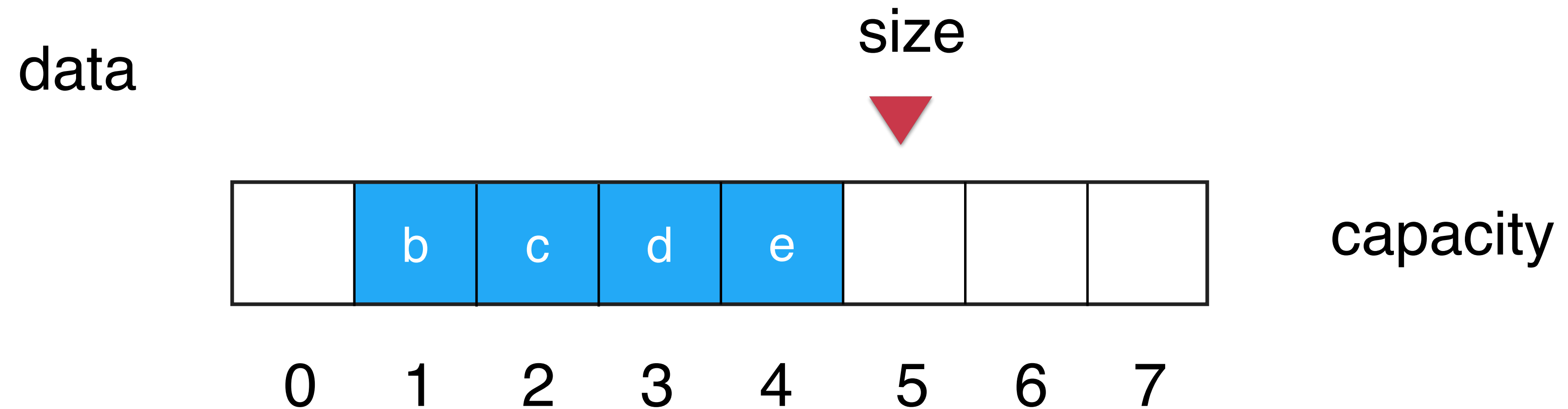
数组队列的问题

删除队首元素



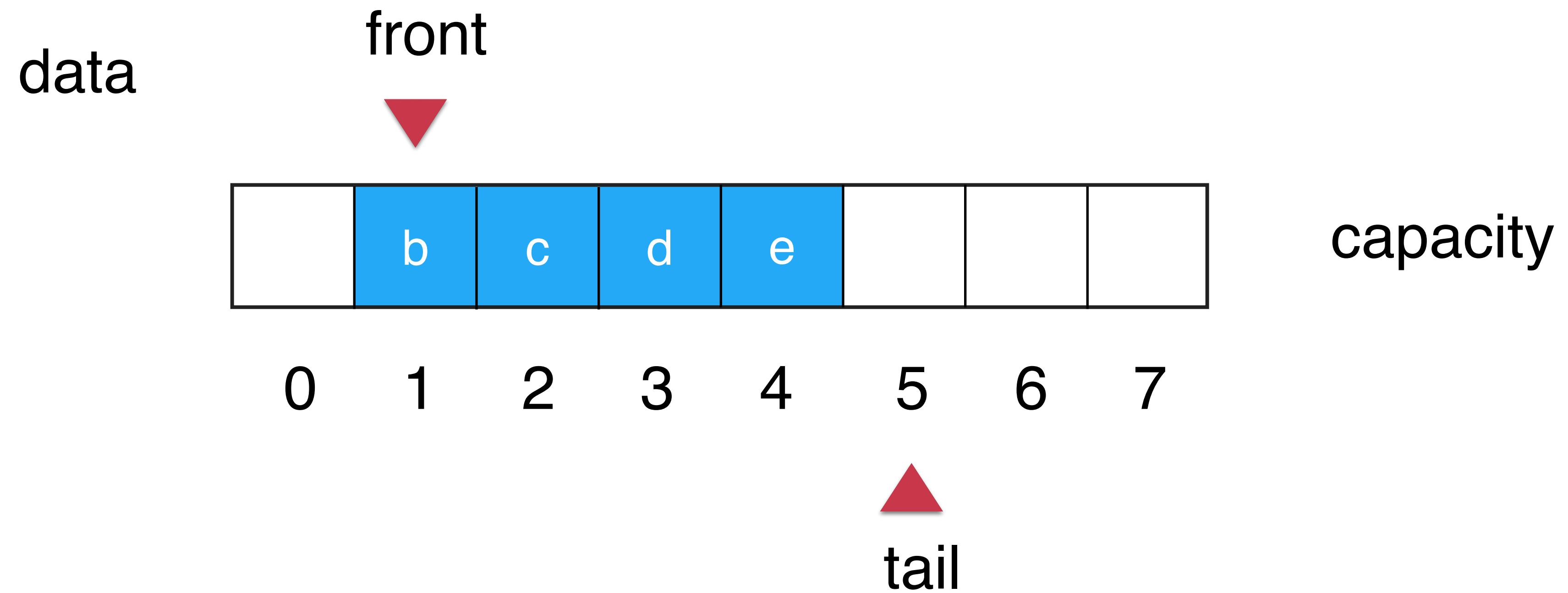
数组队列的问题

删除队首元素



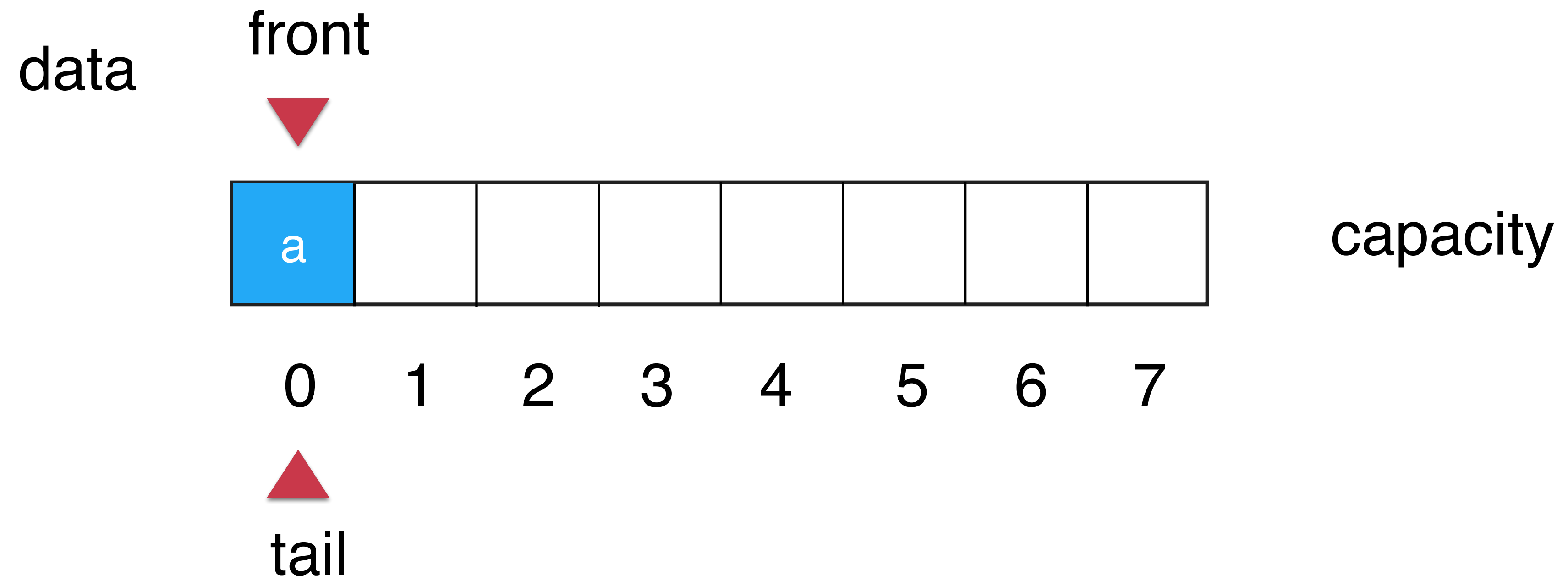
数组队列的问题

删除队首元素



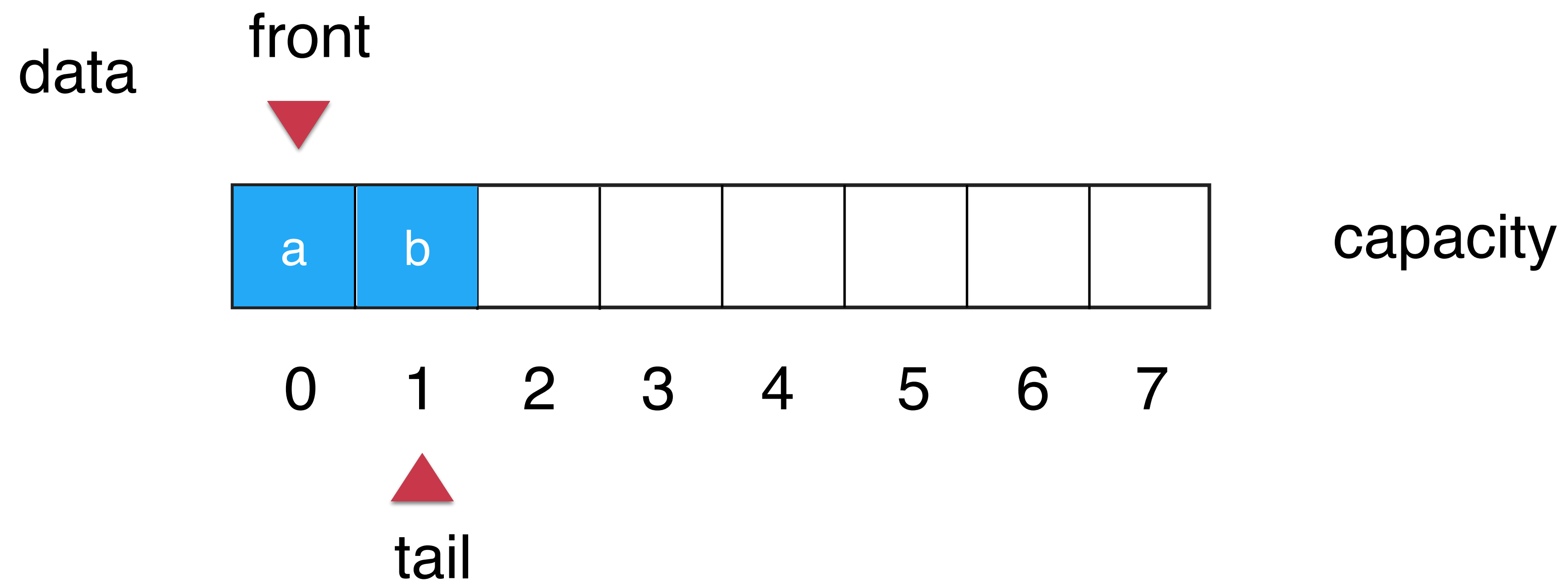
循环队列

front == tail 队列为空



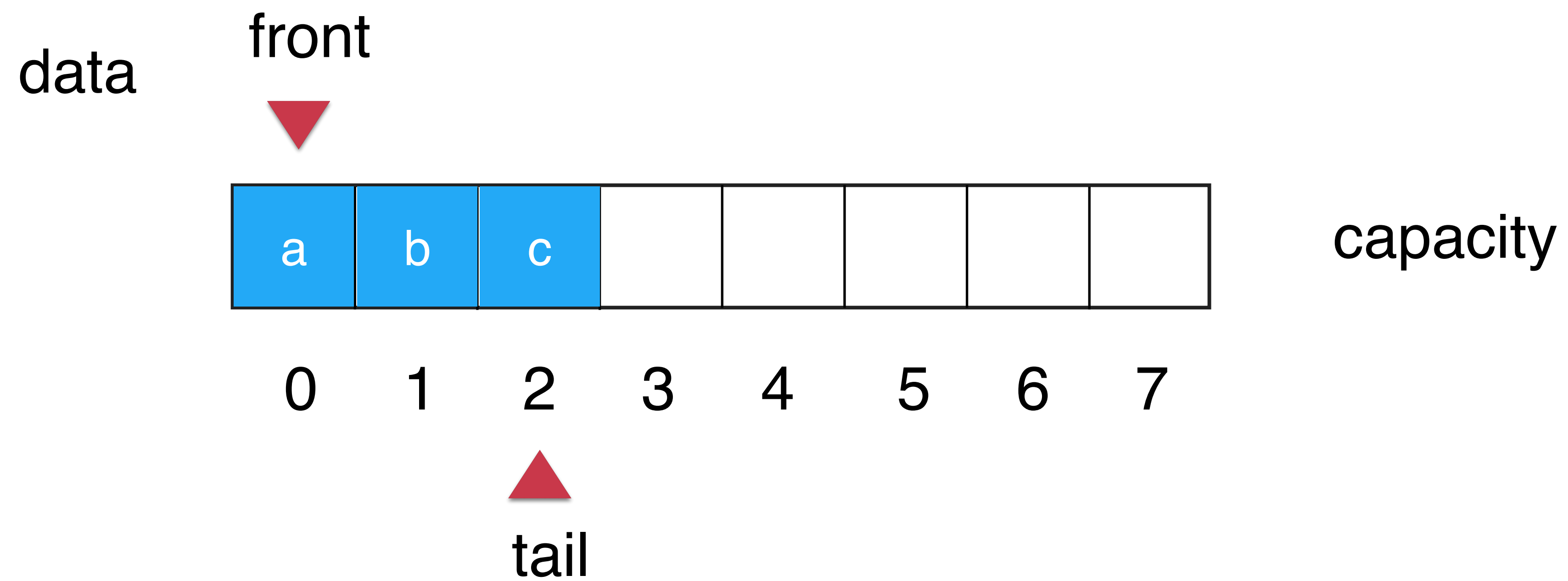
循环队列

front == tail 队列为空



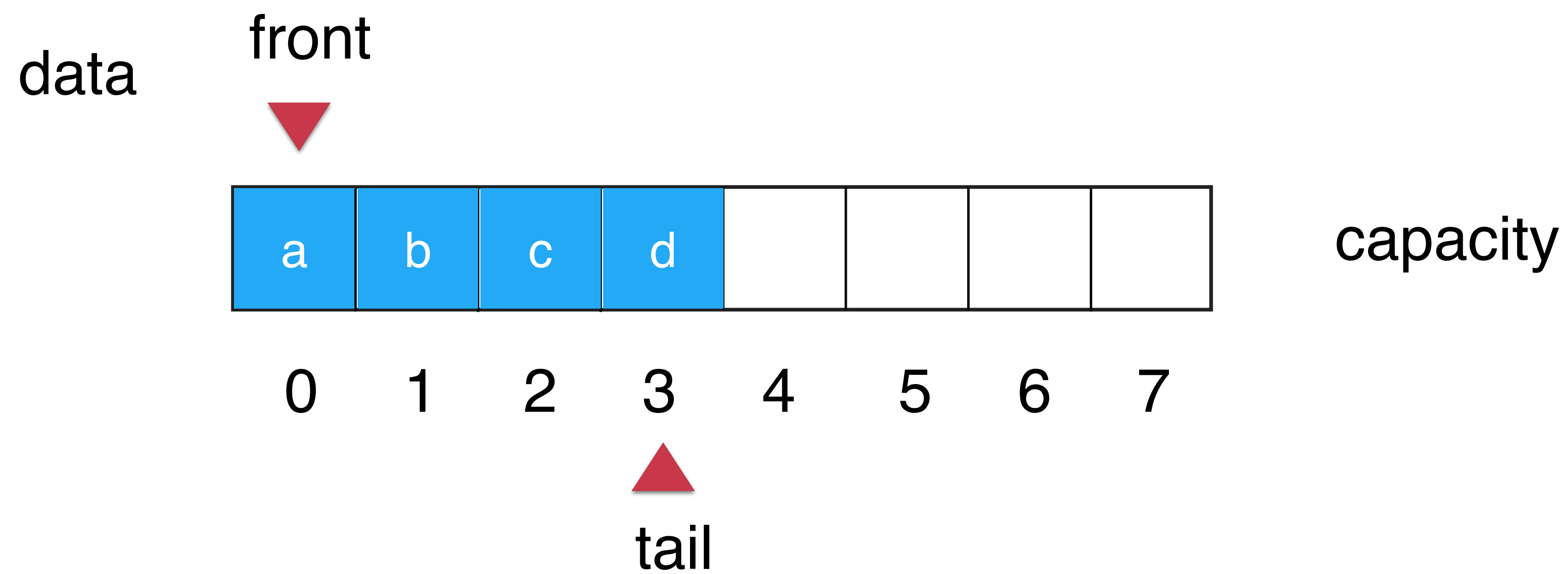
循环队列

front == tail 队列为空



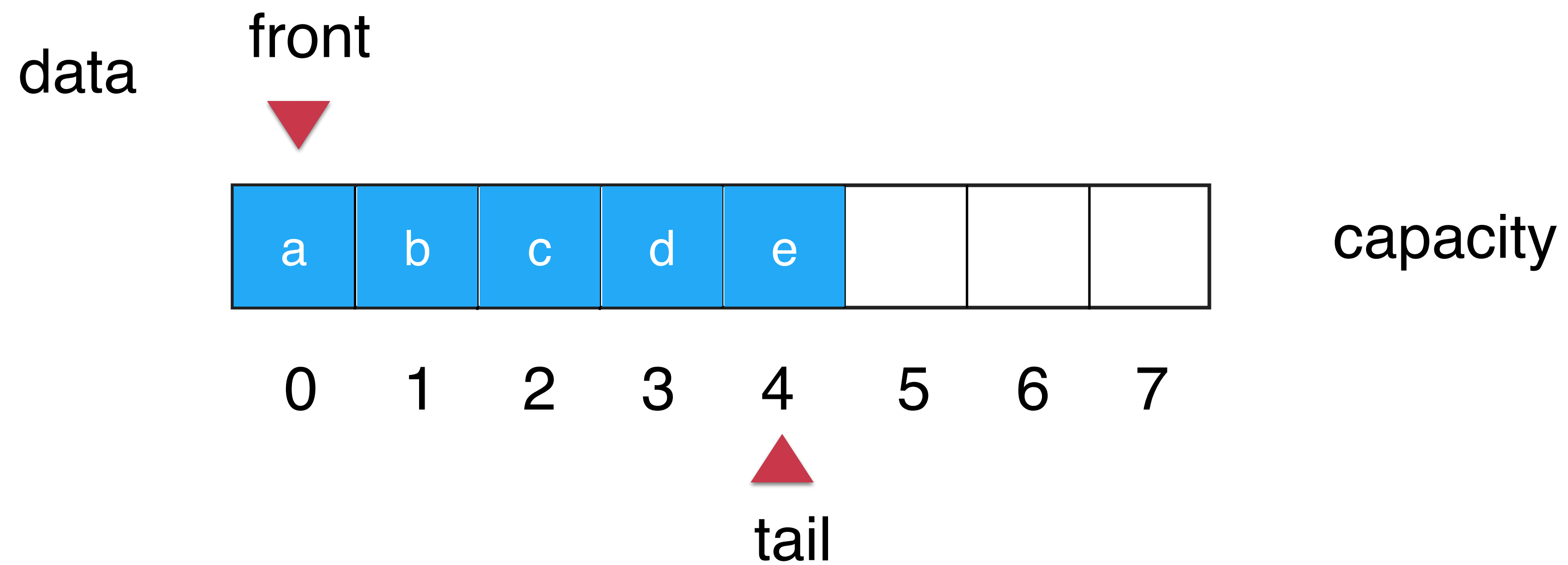
循环队列

front == tail 队列为空



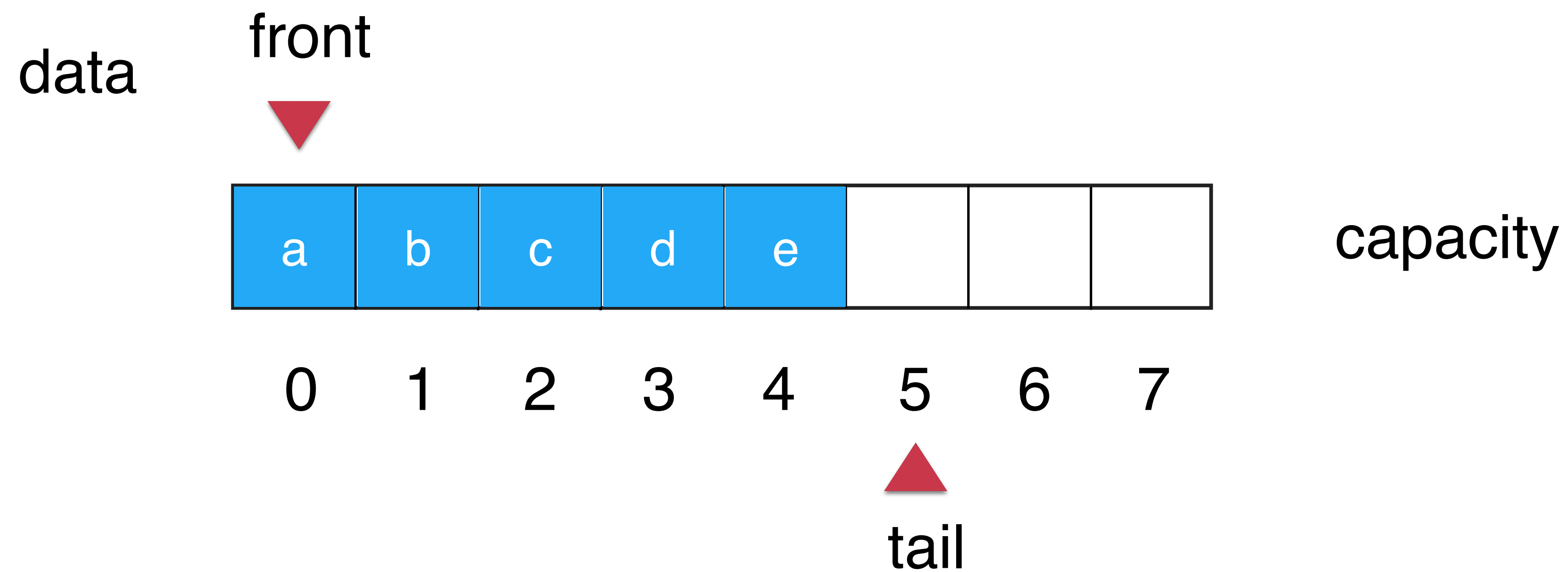
循环队列

front == tail 队列为空



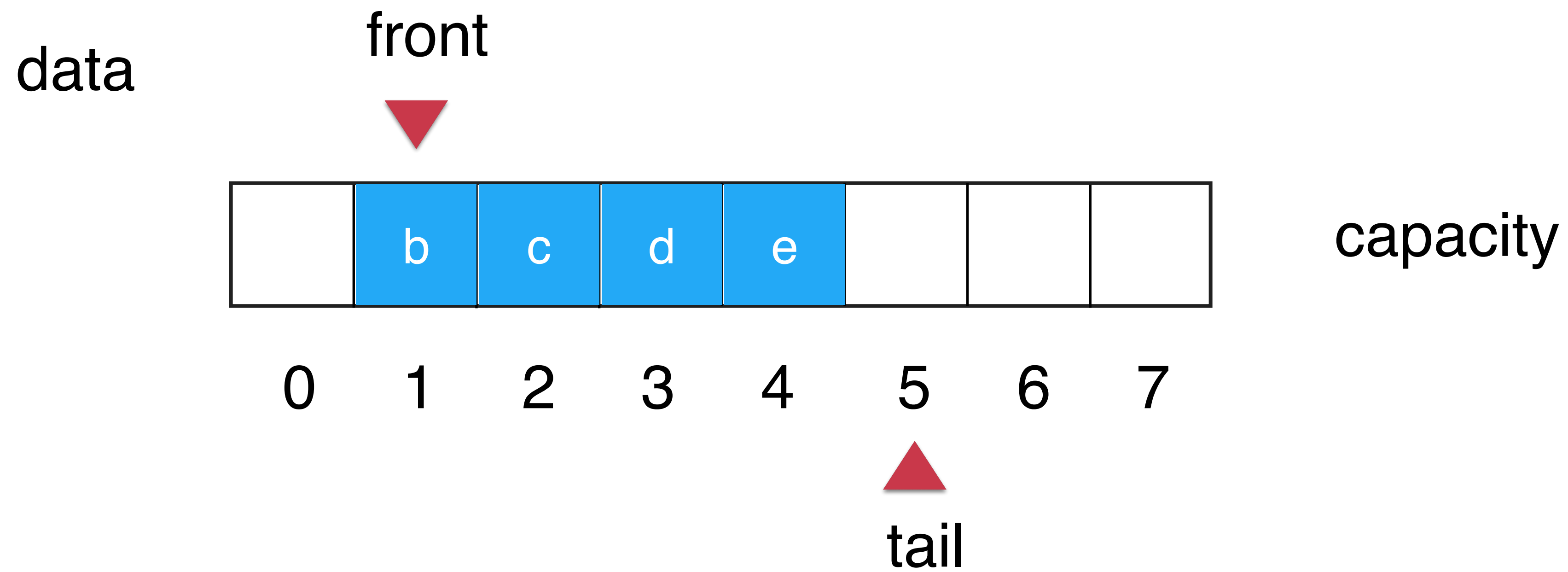
循环队列

front == tail 队列为空



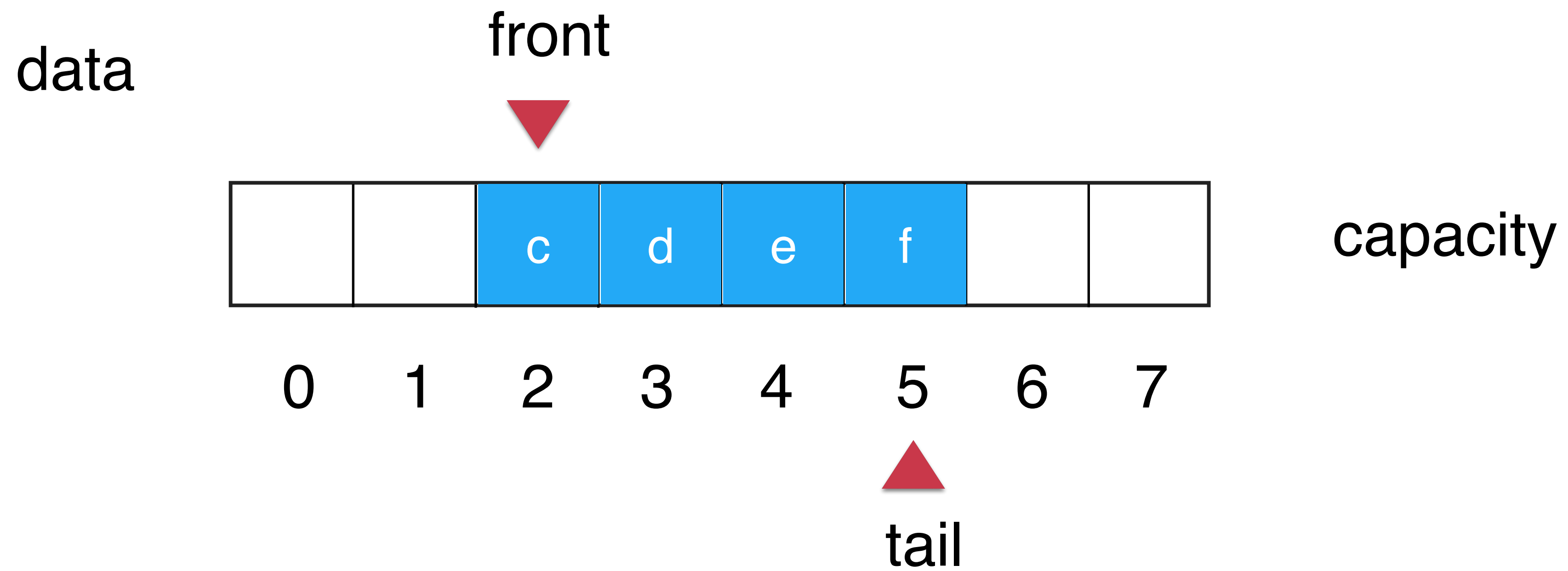
循环队列

front == tail 队列为空



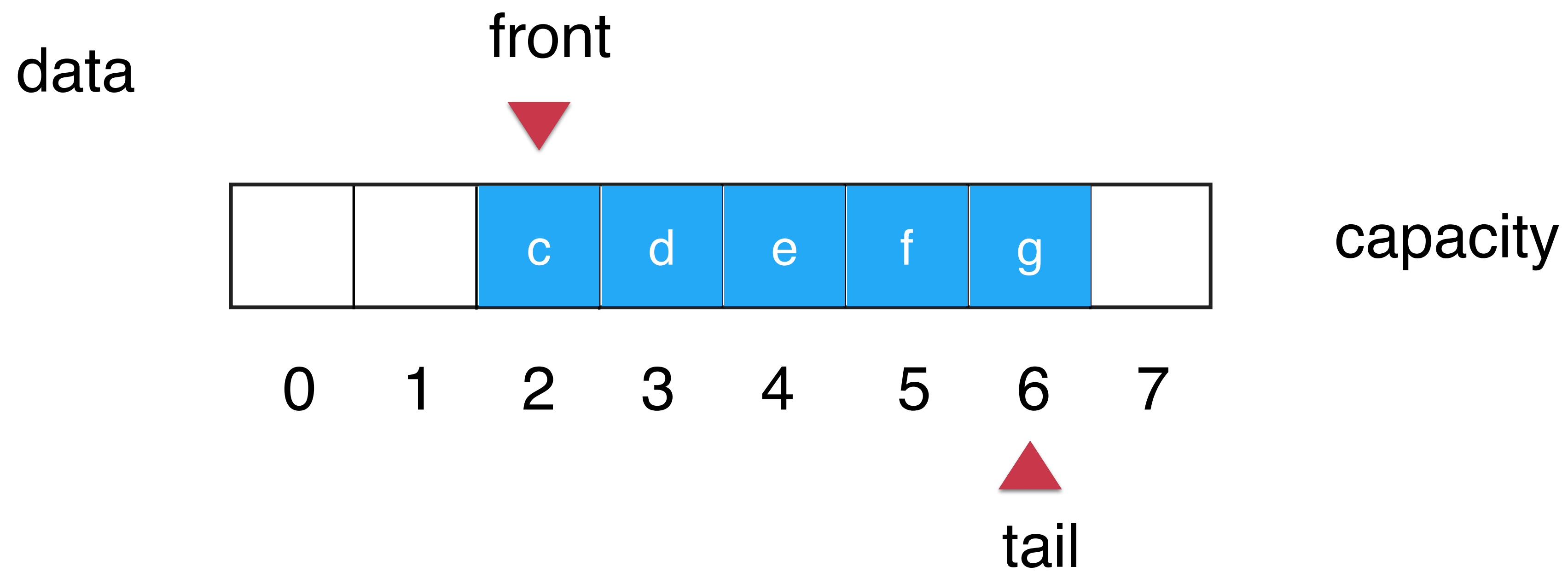
循环队列

front == tail 队列为空



循环队列

front == tail 队列为空



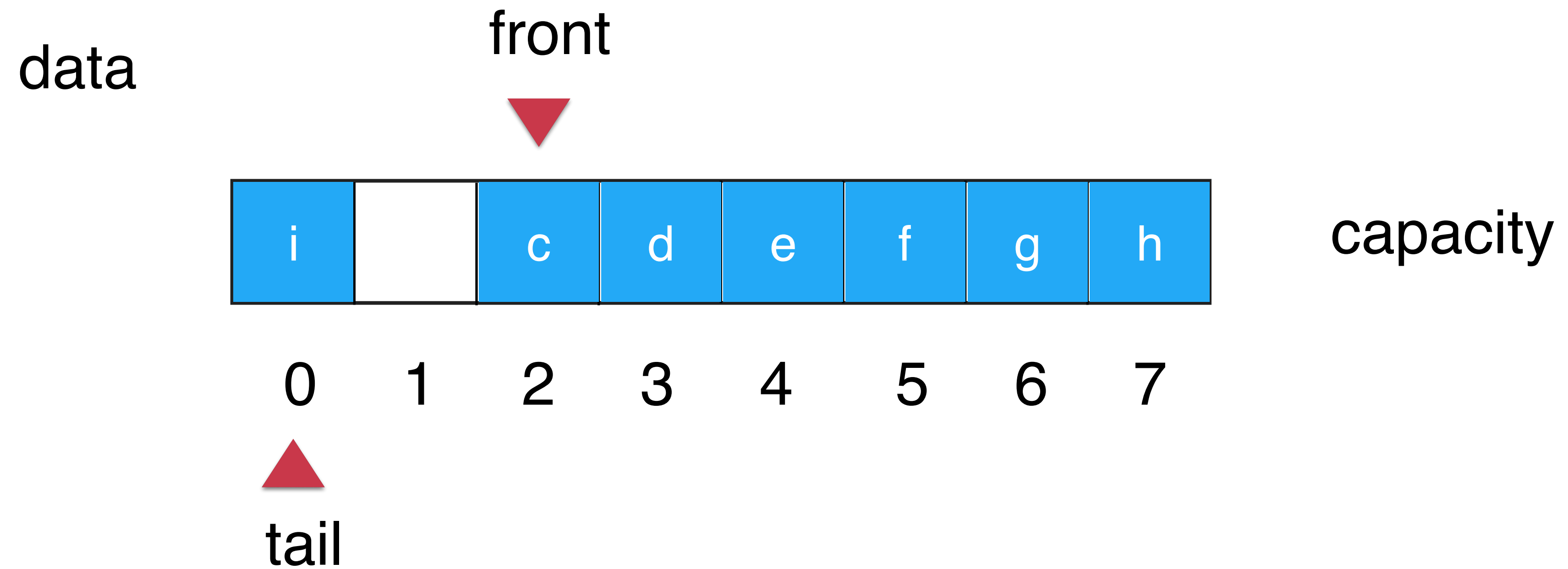
循环队列

front == tail 队列为空



循环队列

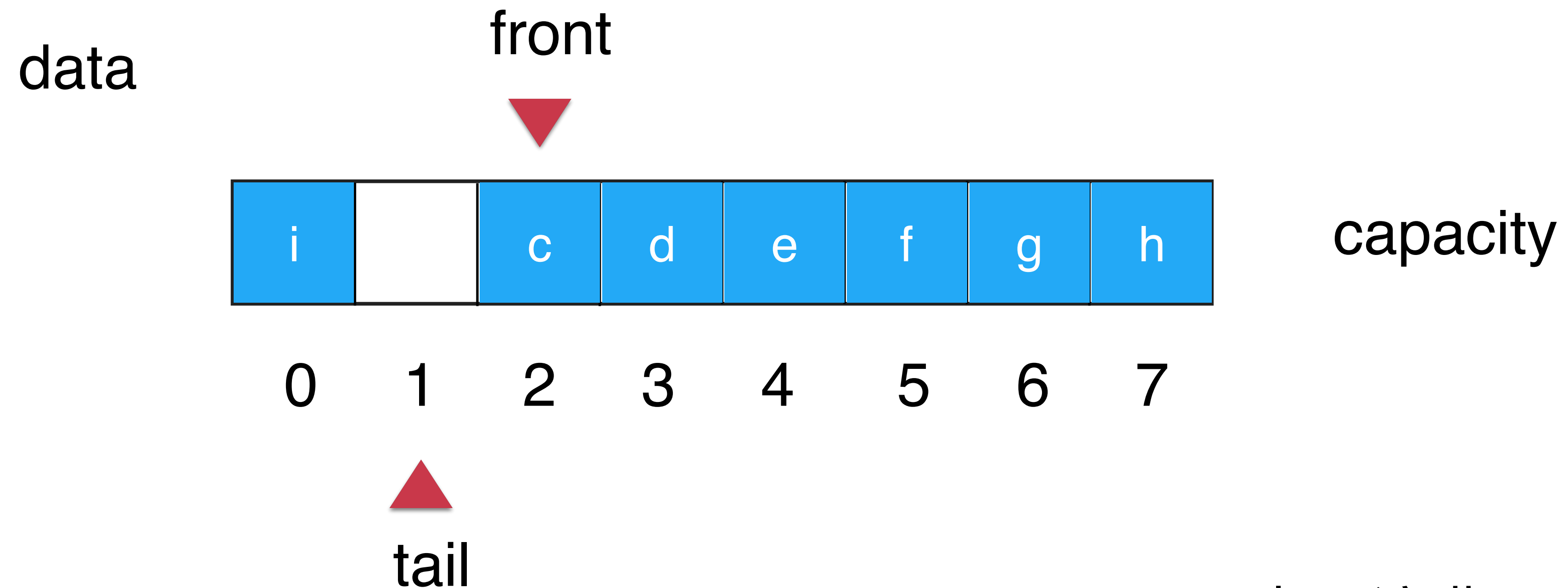
front == tail 队列为空



循环队列

$\text{front} == \text{tail}$ 队列为空

$\text{tail} + 1 == \text{front}$ 队列满

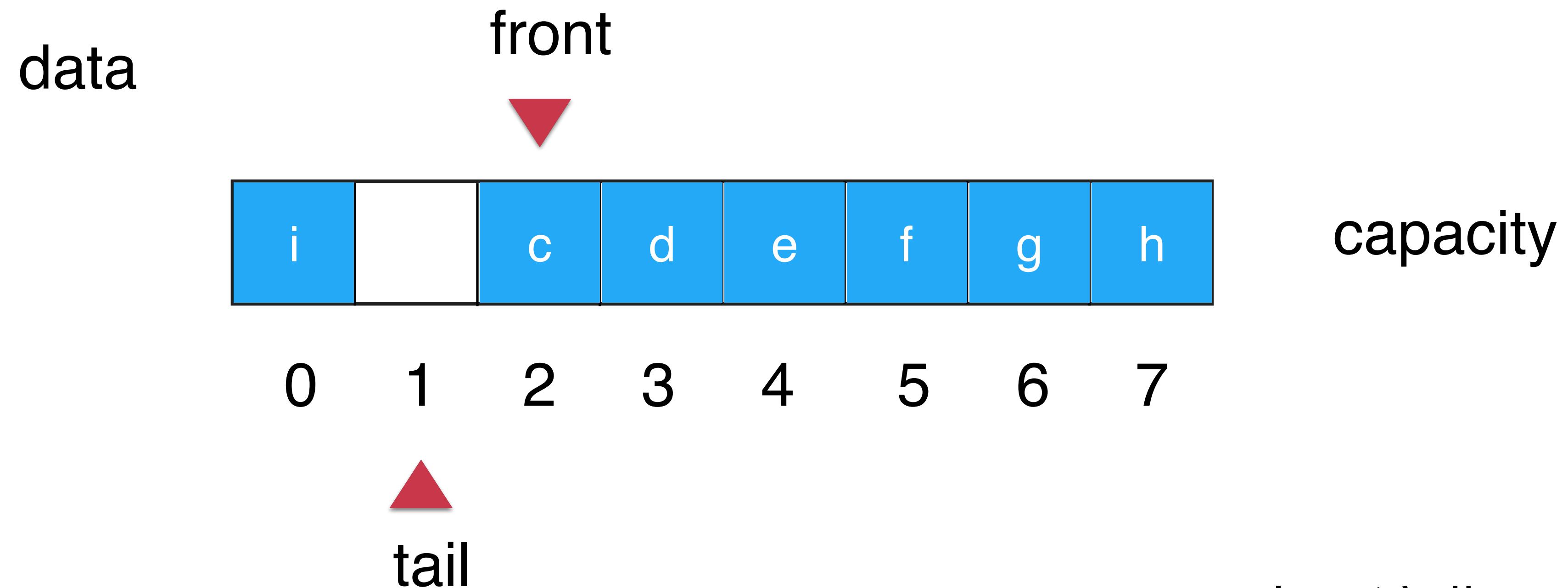


capacity中，浪费一个空间

循环队列

$\text{front} == \text{tail}$ 队列为空

$(\text{tail} + 1) \% c == \text{front}$ 队列满



capacity中，浪费一个空间

实践：循环队列的基础实现

循环队列的实现

实践：循环队列的实现

实践： 循环队列的实现加入resize()

数组队列和循环队列的比较

循环队列的复杂度分析

LoopQueue<E>

- `void enqueue(E)` $O(1)$ 均摊
- `E dequeue()` $O(1)$ 均摊
- `E getFront()` $O(1)$
- `int getSize()` $O(1)$
- `boolean isEmpty()` $O(1)$

实践：数组队列和循环队列的比较

队列的应用

栈和队列

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