玩儿转数据结构

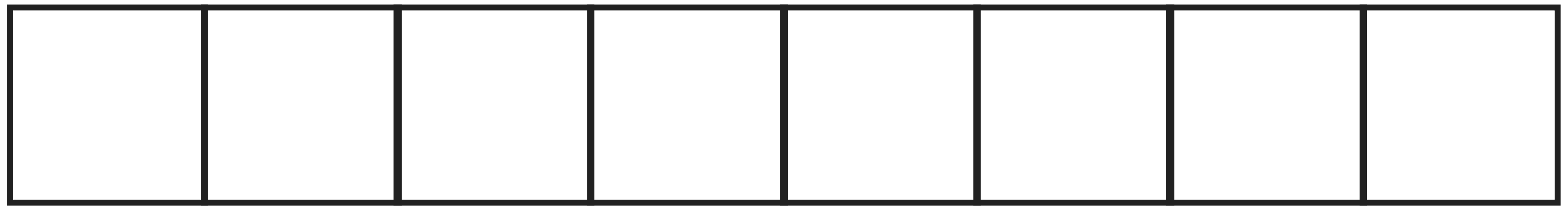
liuyubobobo

不要小瞧数组

数组基础

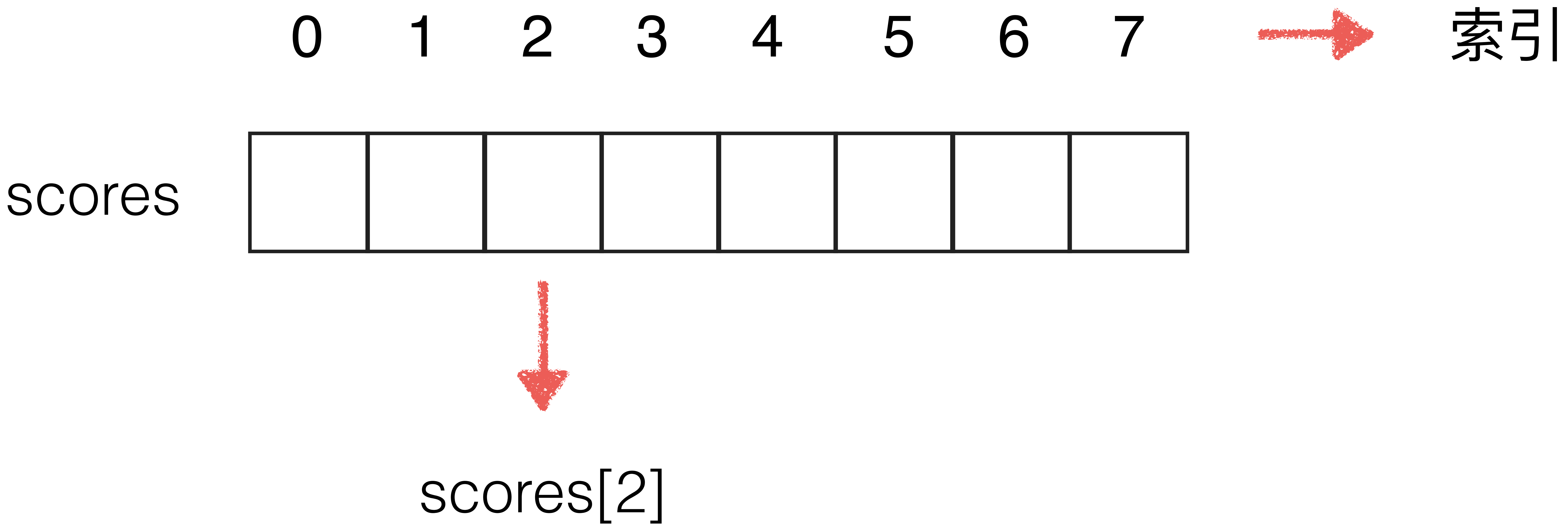
- 把数据码成一排进行存放

arr



数组基础

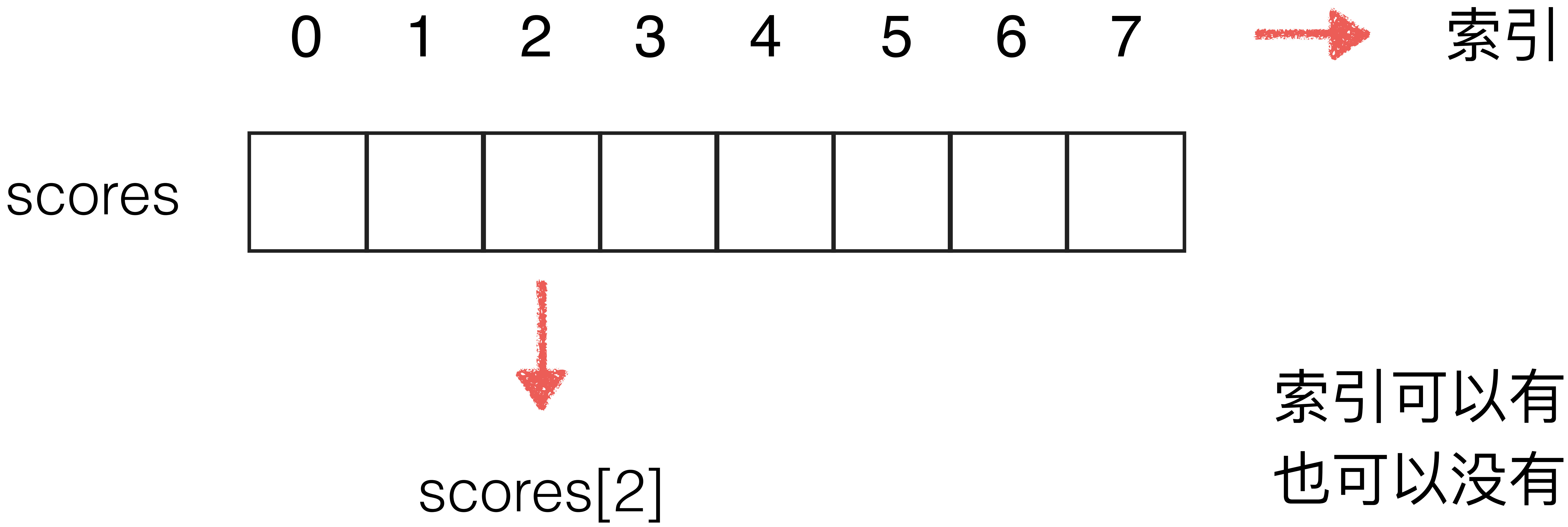
- 把数据码成一排进行存放



二次封装属于我们自己的数组

数组基础

- 把数据码成一排进行存放



索引可以有语意；
也可以没有语意

数组基础

- 数组最大的优点：快速查询。scores[2]
- 数组最好应用于“索引有语意”的情况。
- 但并非所有有语意的索引都适用于数组

身份证号：110103198512166666

数组基础

- 但并非所有有语意的索引都适用于数组

  身份证号：110103198512166666

- 数组也可以处理“索引没有语意”的情况。

- 我们在这一章，主要处理“索引没有语意”的情况数组的使用。

数组基础

- 我们在这一章，主要处理“索引没有语意”的情况数组的使用。

	0	1	2	3	4	5	6	7
scores	100	99	66					

- 索引没有语意，如何表示没有元素？
- 如何添加元素？如何删除元素？
- ……

制作属于我们自己的数组类

- 索引没有语意，如何表示没有元素?
- 如何添加元素？如何删除元素?
- ……

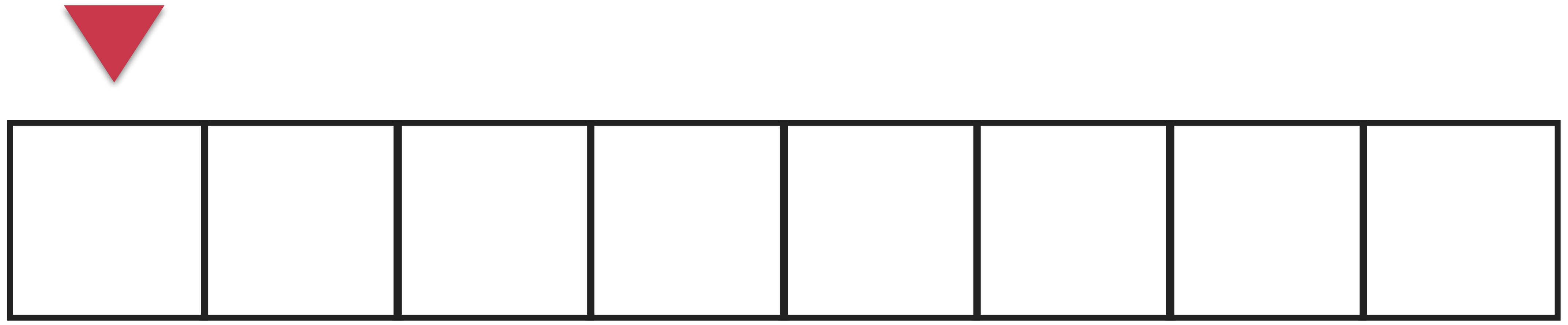
- 基于java的数组，二次封装属于我们自己的数组类

制作属于我们自己的数组类

class Array

size

data



capacity

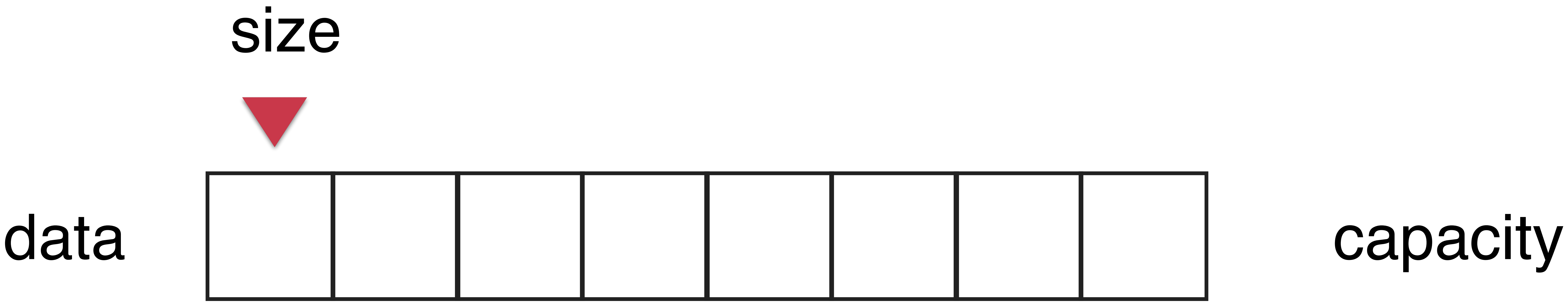
增 删 改 查

实践：二测封装属于我们自己的数组

向数组添加元素

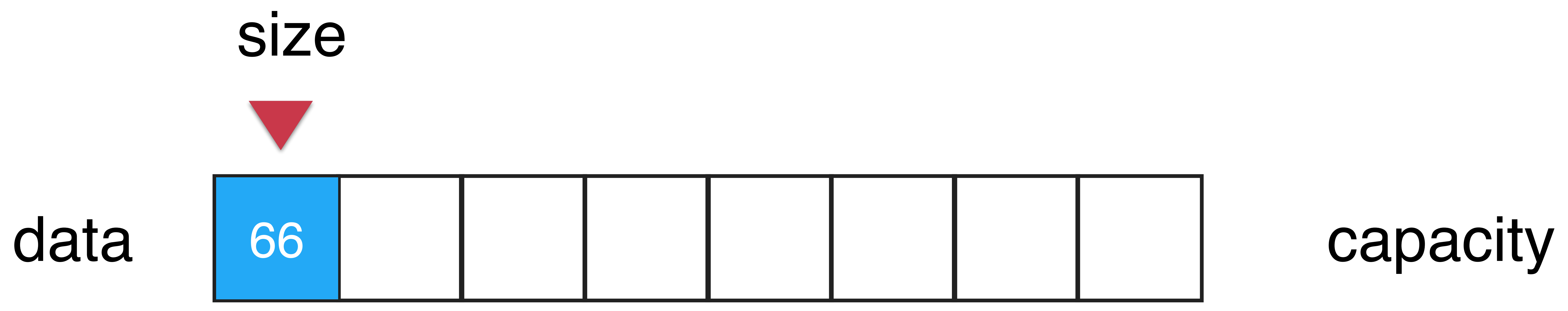
向数组中添加元素

向数组末添加元素



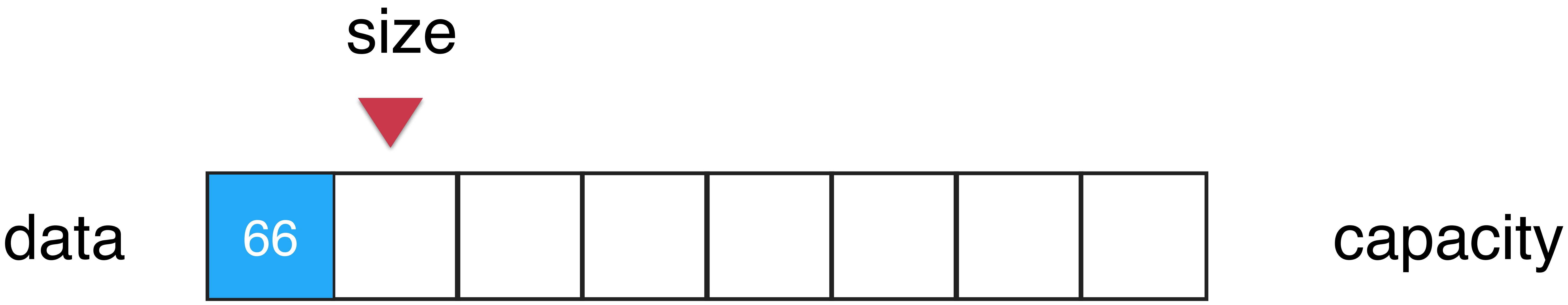
向数组中添加元素

向数组末添加元素



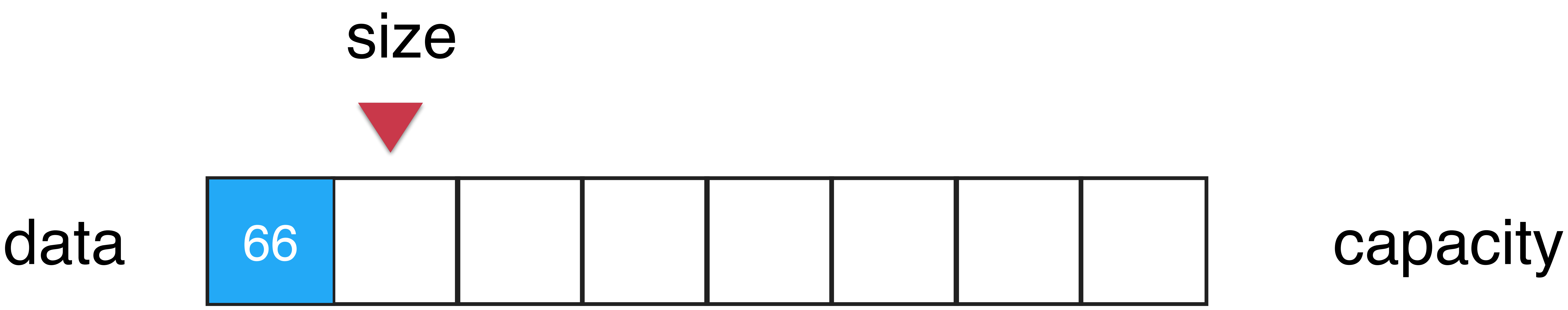
向数组中添加元素

向数组末添加元素



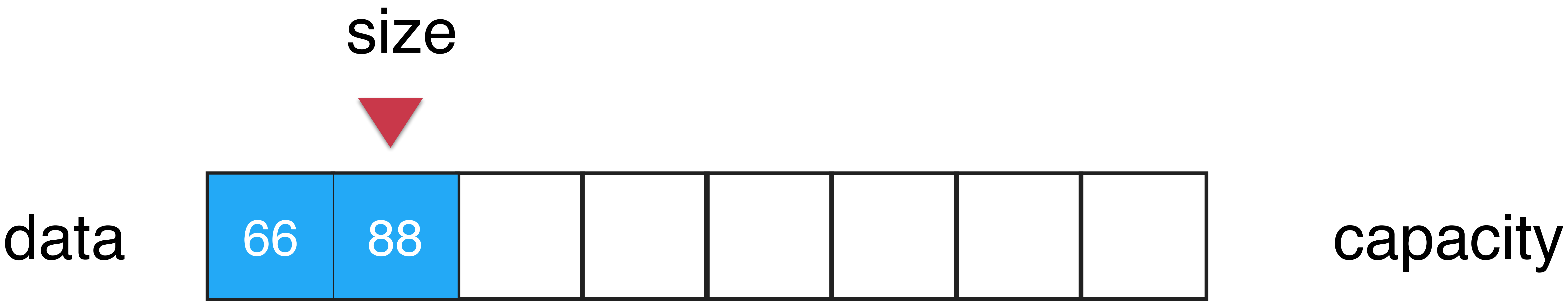
向数组中添加元素

向数组末添加元素



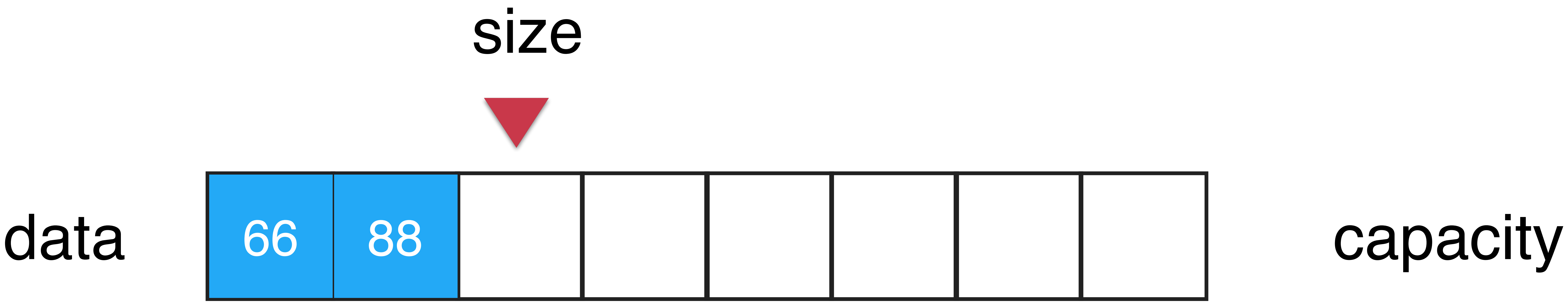
向数组中添加元素

向数组末添加元素



向数组中添加元素

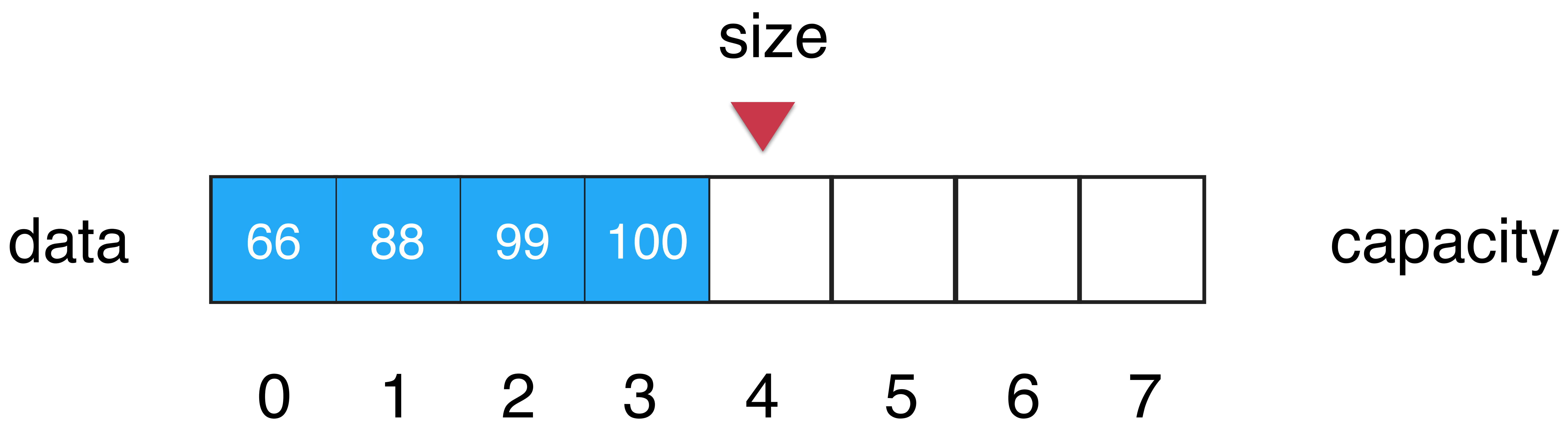
向数组末添加元素



实践：向数组末尾添加元素

向数组中添加元素

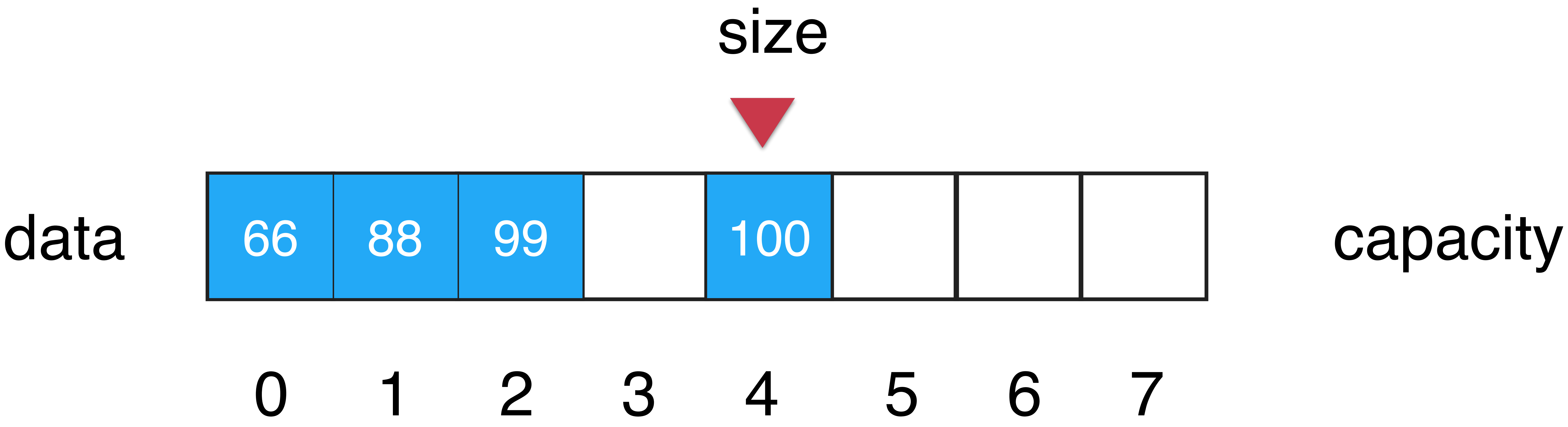
向指定位置添加元素



把77插入到索引为1的位置

向数组中添加元素

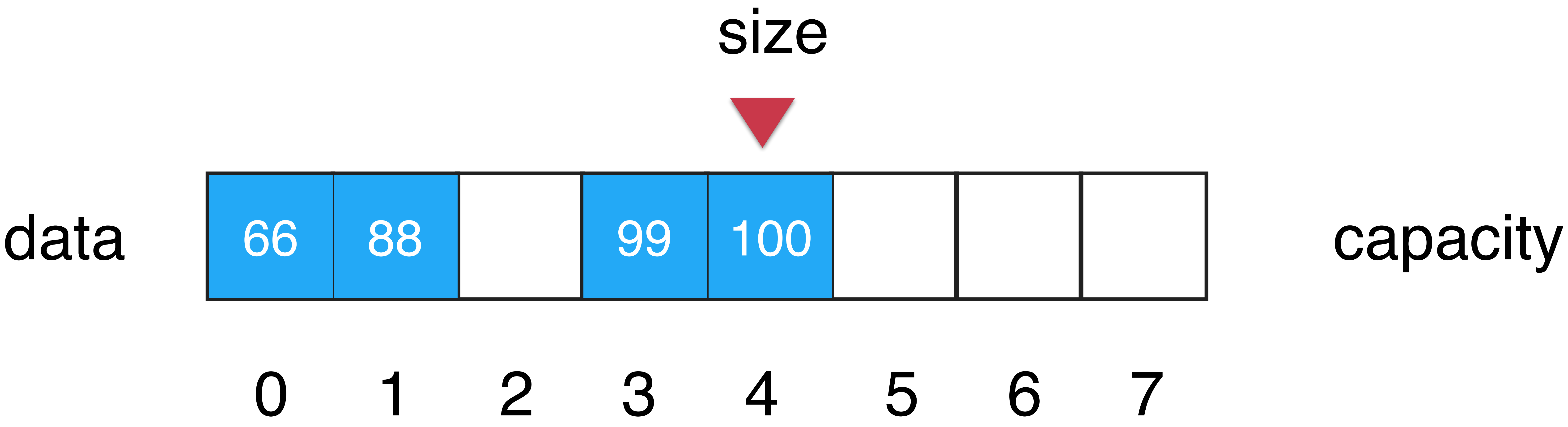
向指定位置添加元素



把77插入到索引为1的位置

向数组中添加元素

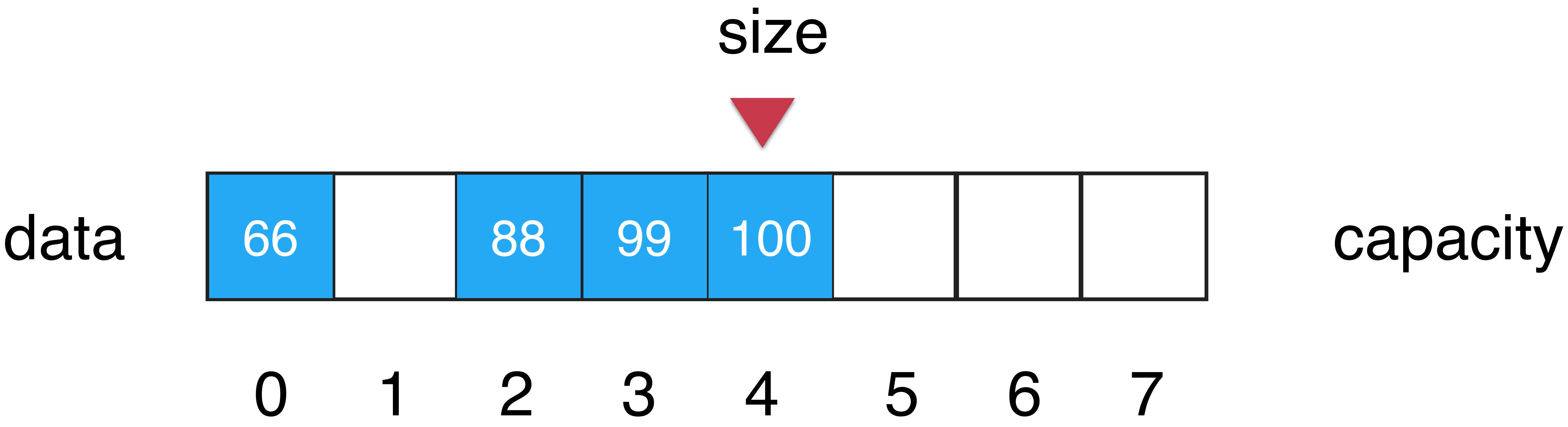
向指定位置添加元素



把77插入到索引为1的位置

向数组中添加元素

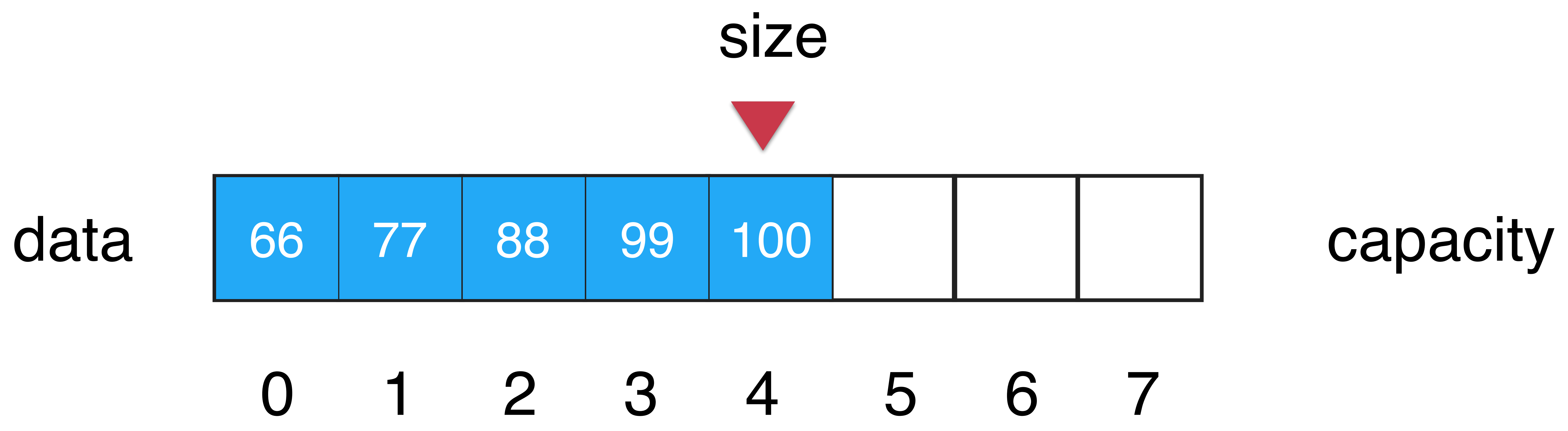
向指定位置添加元素



把77插入到索引为1的位置

向数组中添加元素

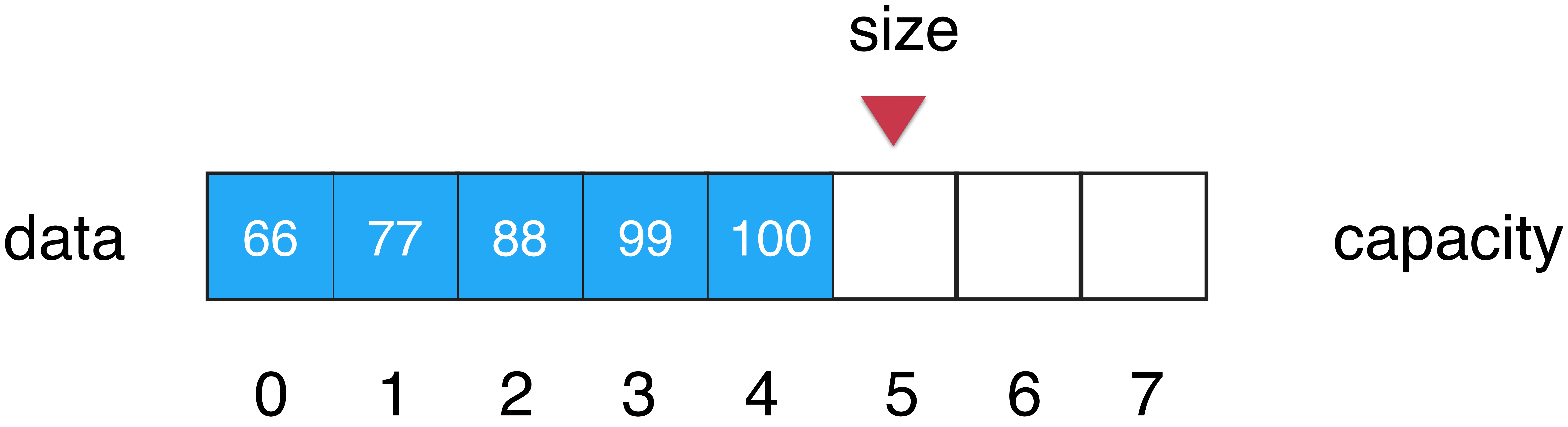
向指定位置添加元素



把77插入到索引为1的位置

向数组中添加元素

向指定位置添加元素



把77插入到索引为1的位置

实践：向数组任意位置添加元素

在数组中查询元素和修改元素

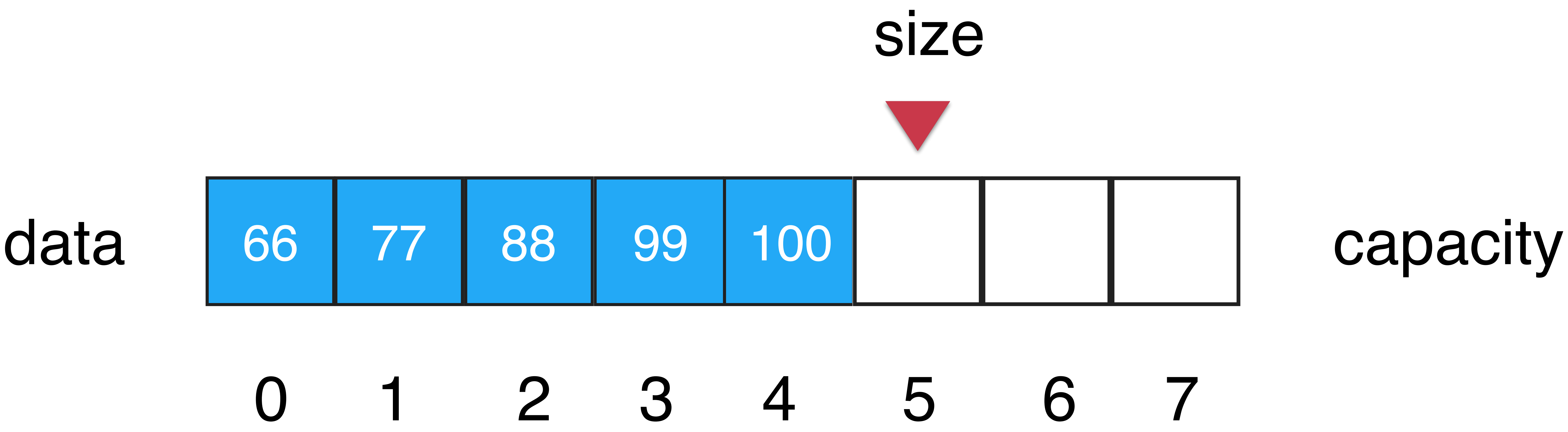
实践：在数组中查询元素和修改元素

数组中的包含，搜索和删除元素

实践：数组中的包含和搜索

从数组中删除元素

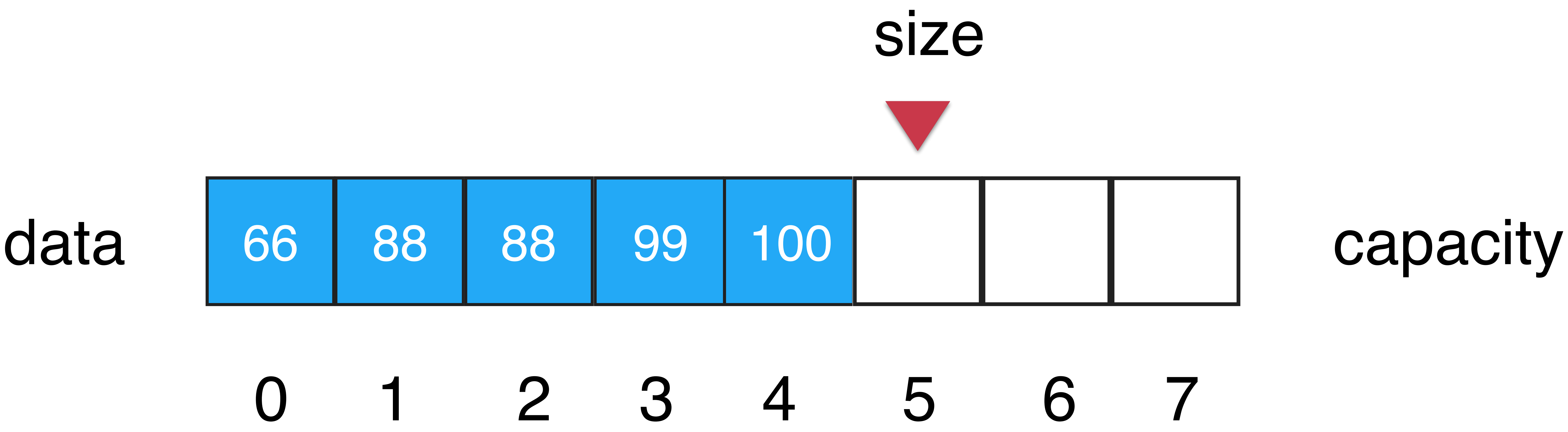
删除指定位置元素



删除索引为1的元素

从数组中删除元素

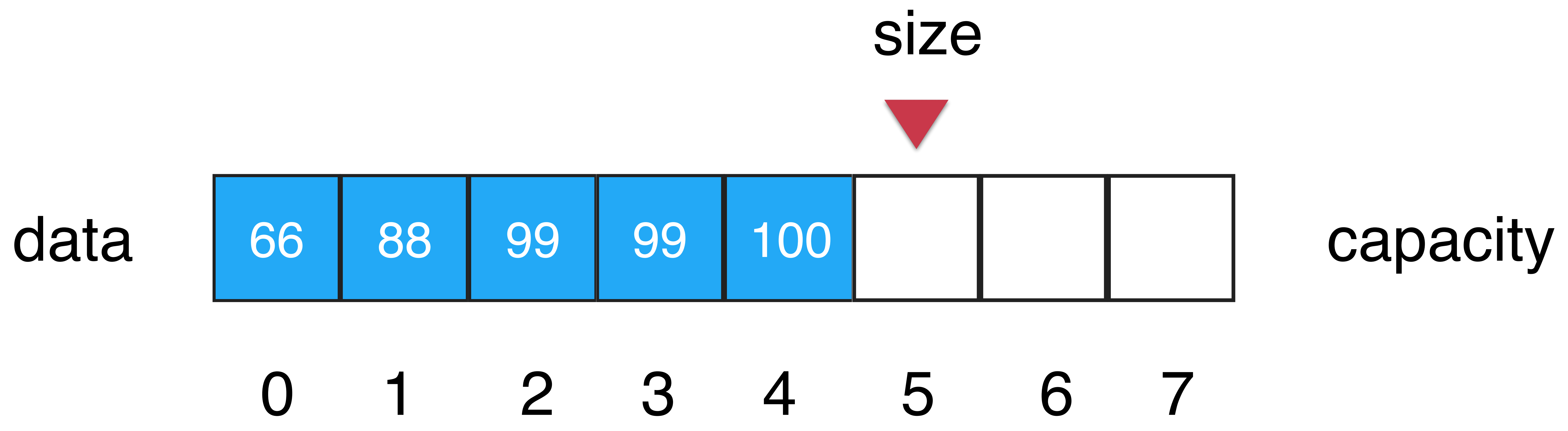
删除指定位置元素



删除索引为1的元素

从数组中删除元素

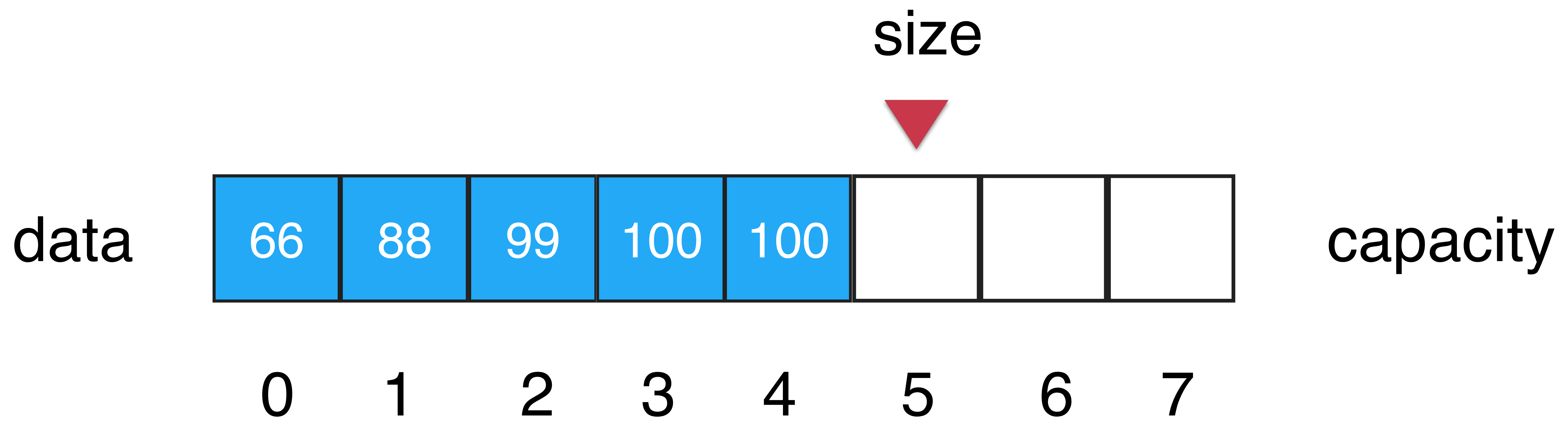
删除指定位置元素



删除索引为1的元素

从数组中删除元素

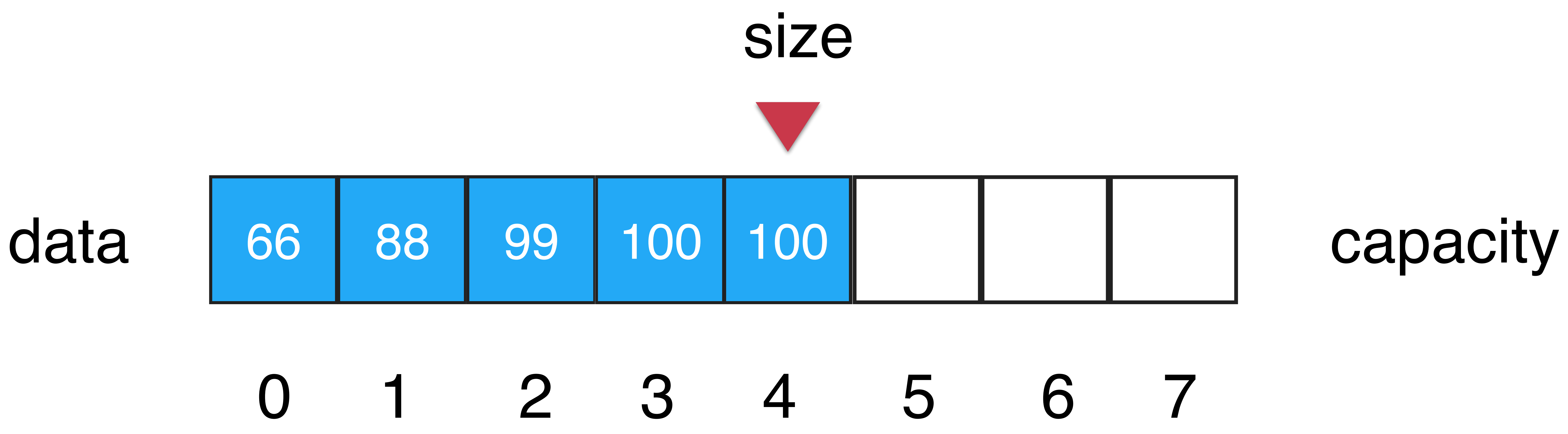
删除指定位置元素



删除索引为1的元素

从数组中删除元素

删除指定位置元素



删除索引为1的元素

实践：从数组中删除元素

使用泛型

使用泛型

- 让我们的数据结构可以放置“任何”数据类型

- 不可以是基本数据类型，只能是类对象

  boolean , byte , char , short , int , long , float , double

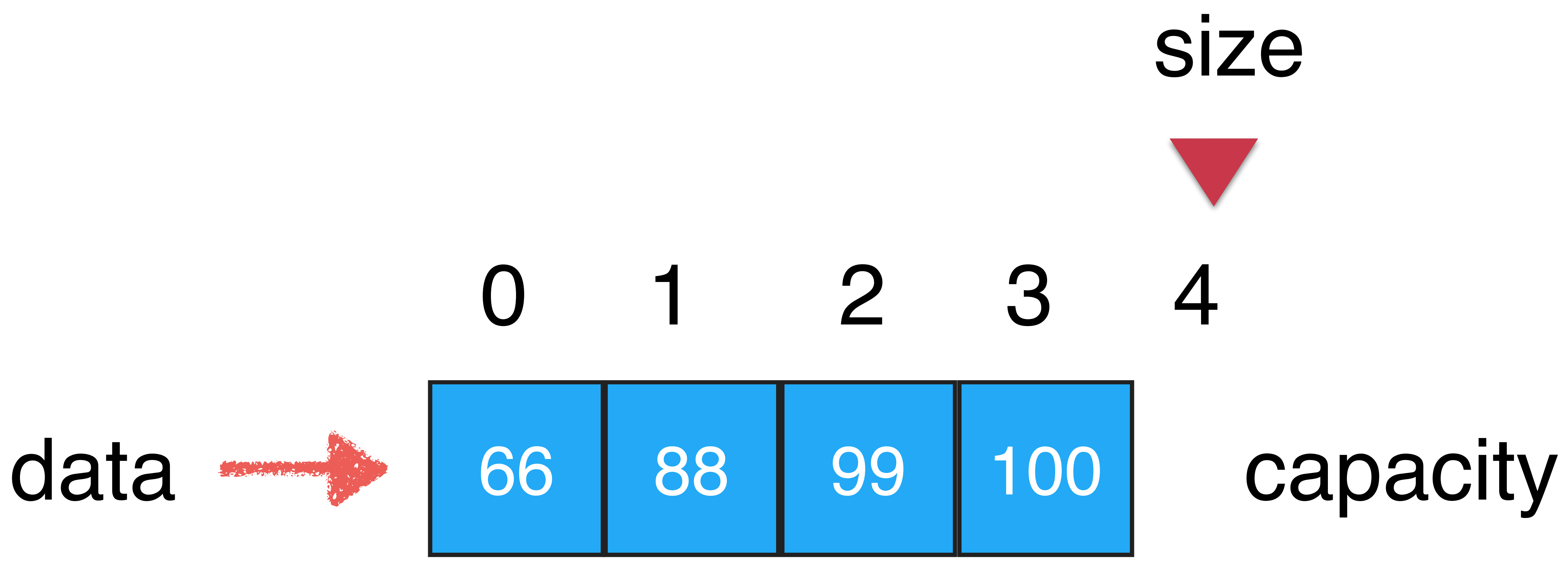
- 每个基本数据类型都有对应的包装类

  Boolean , Byte , Char , Short , Int , Long , Float , Double

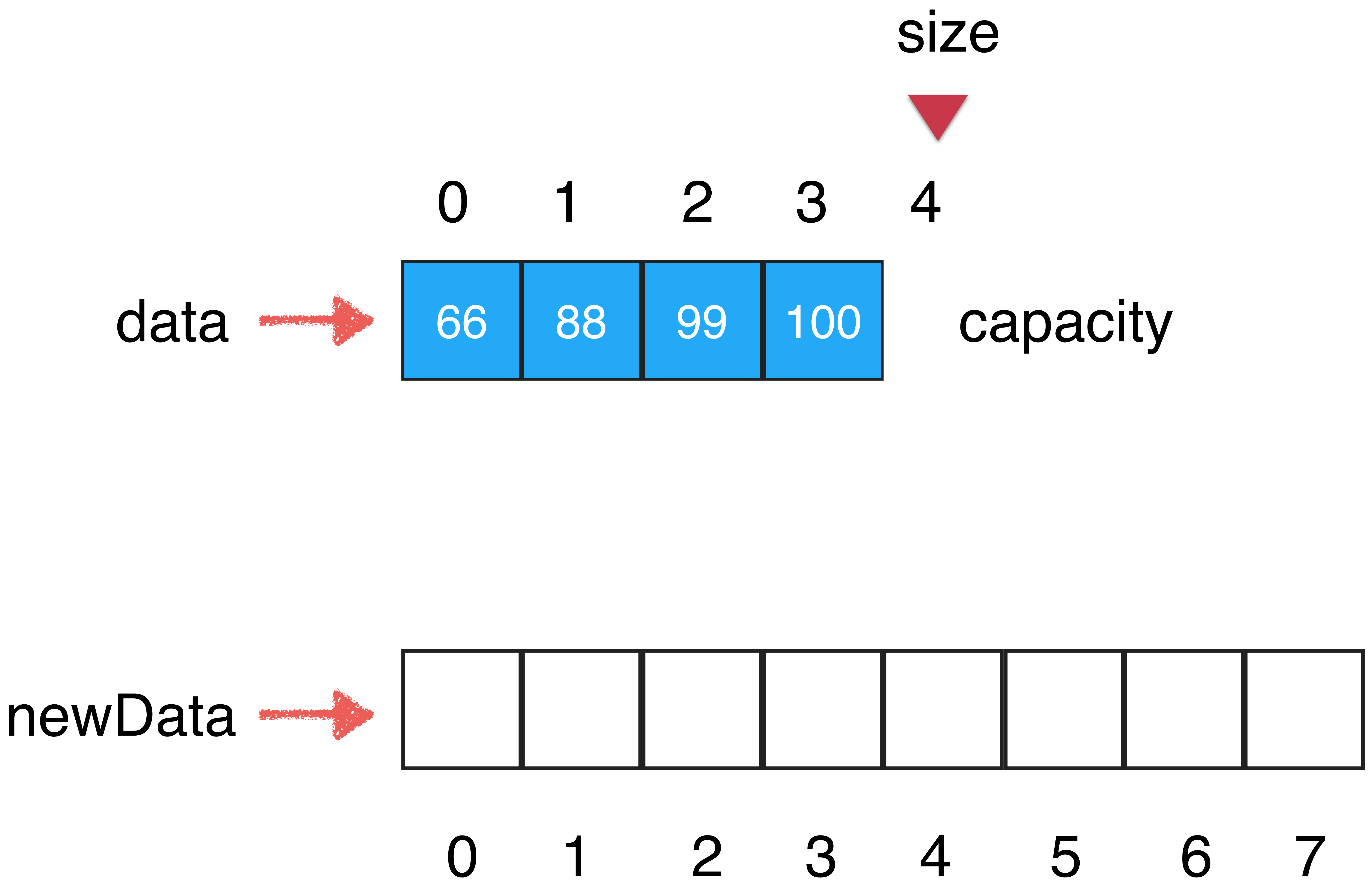
实践：使用泛型

动态数组

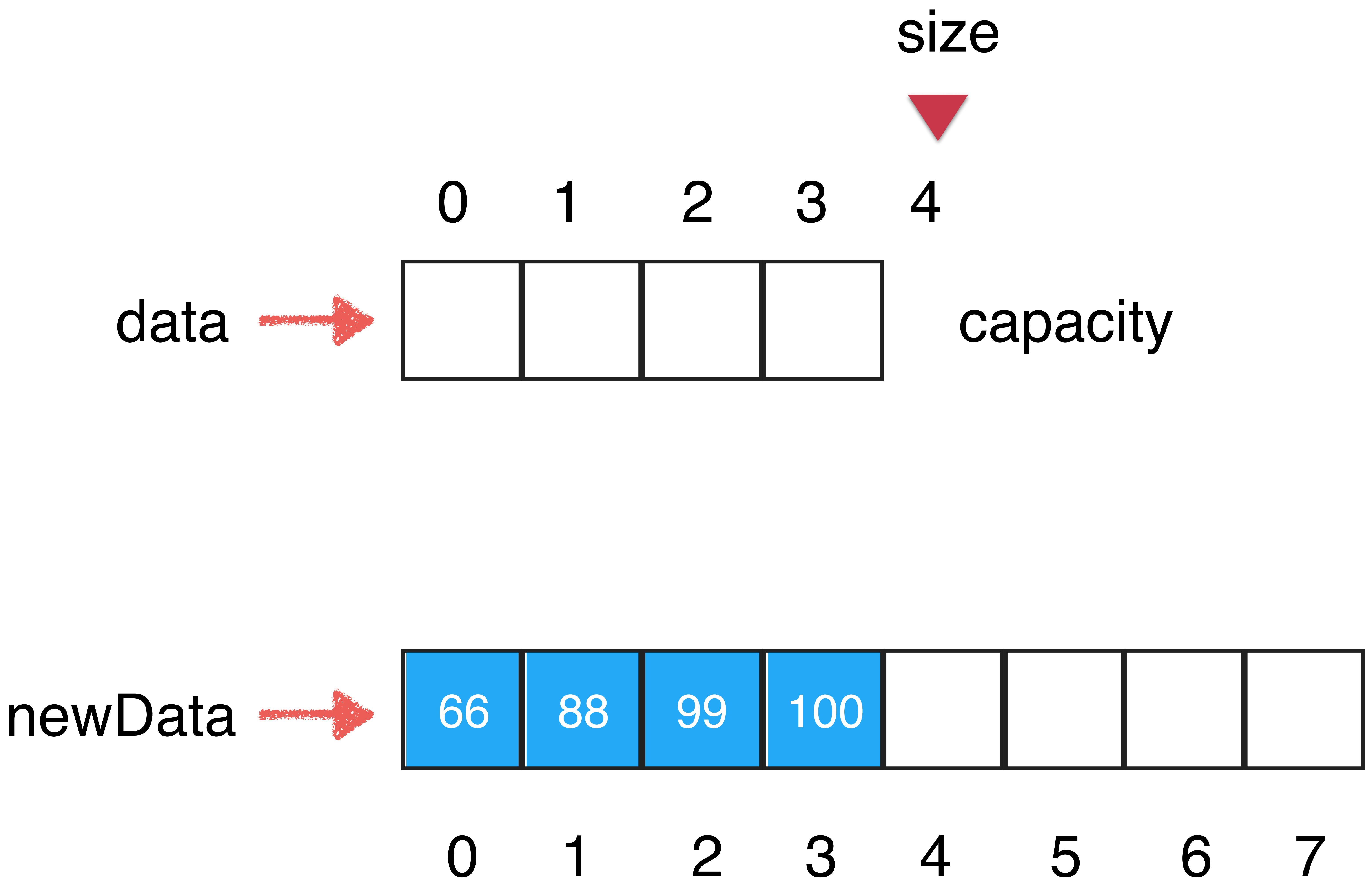
动态数组



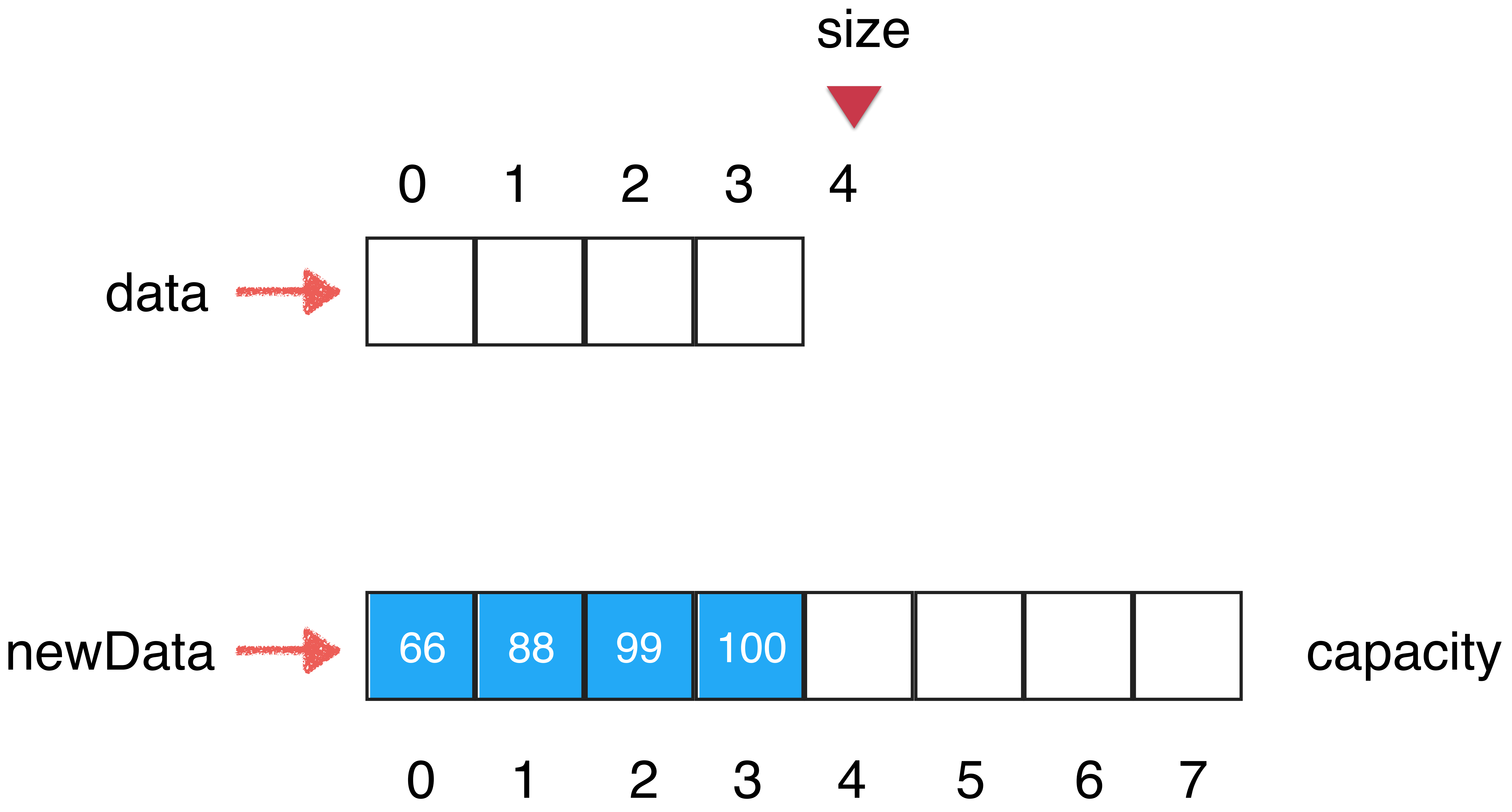
动态数组



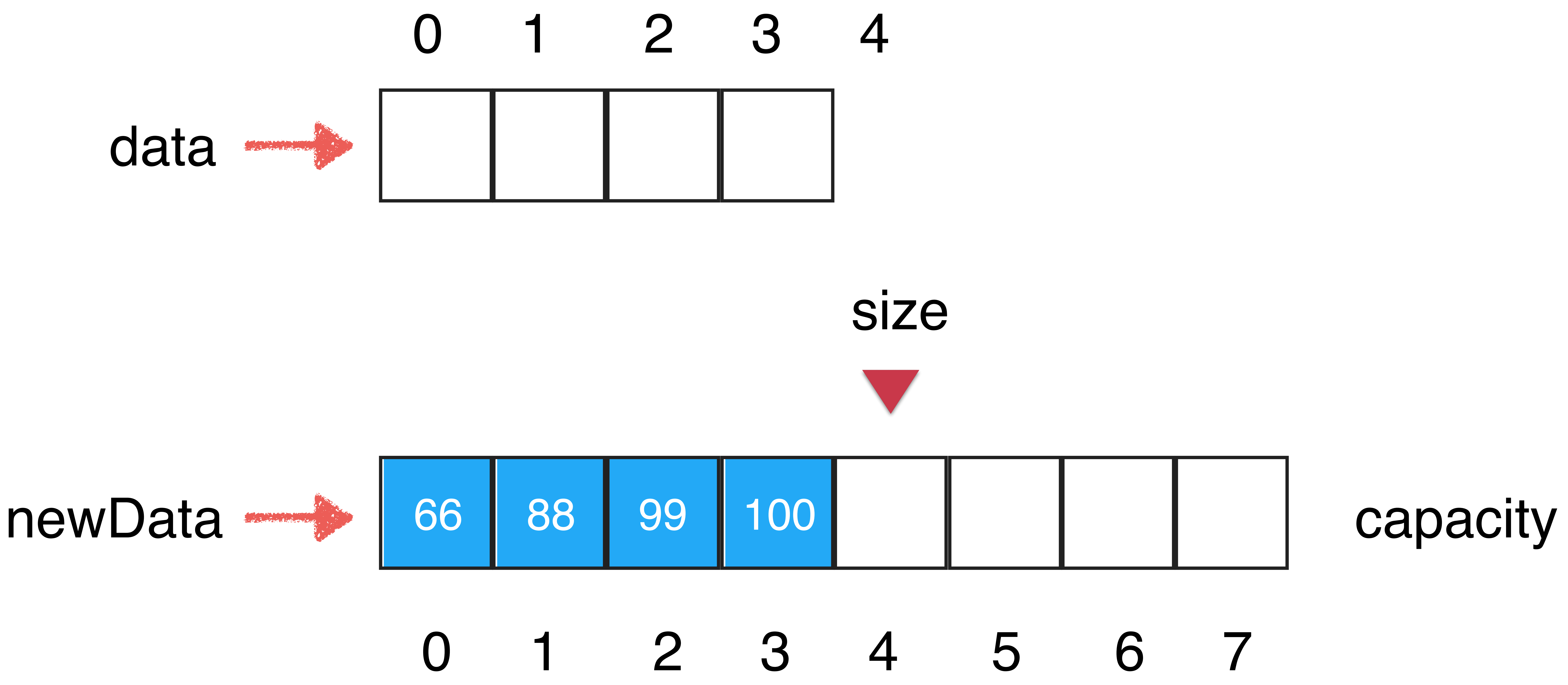
动态数组



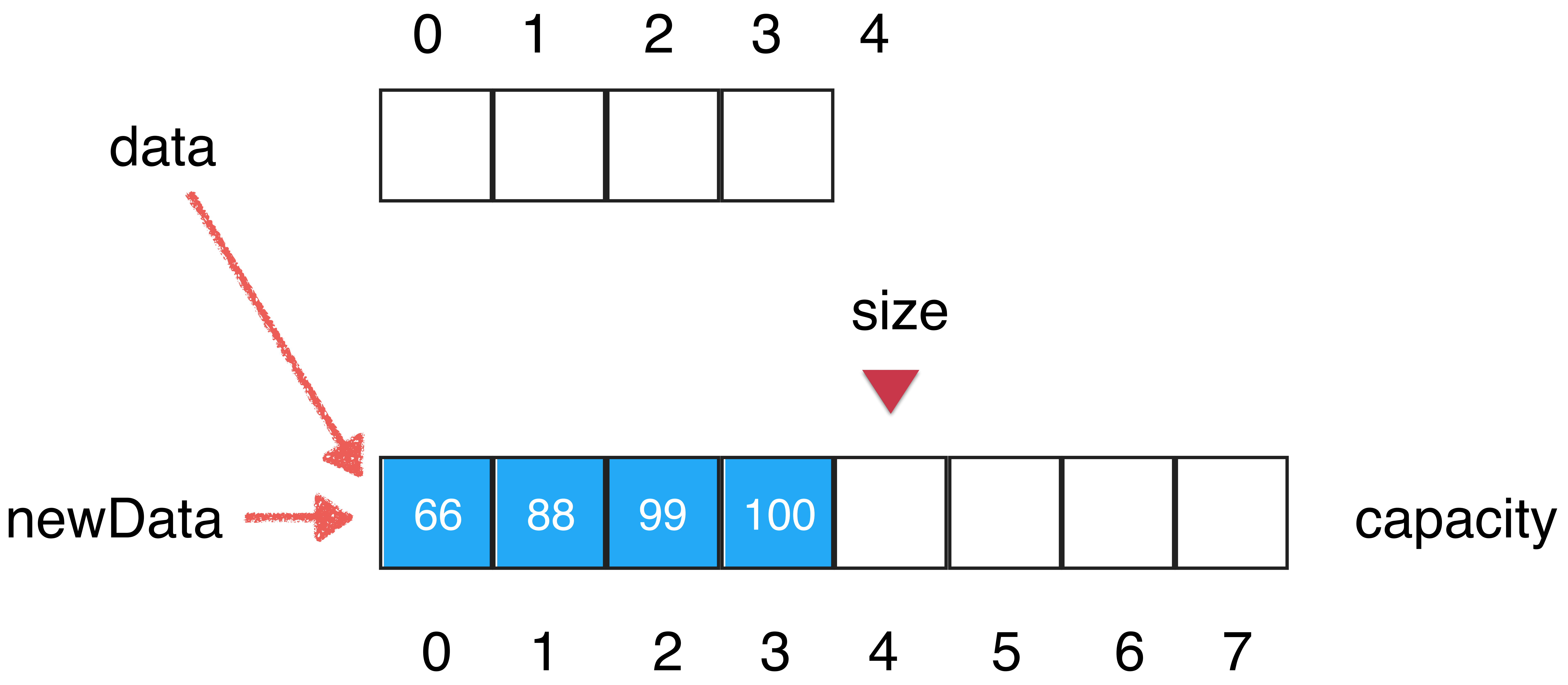
动态数组



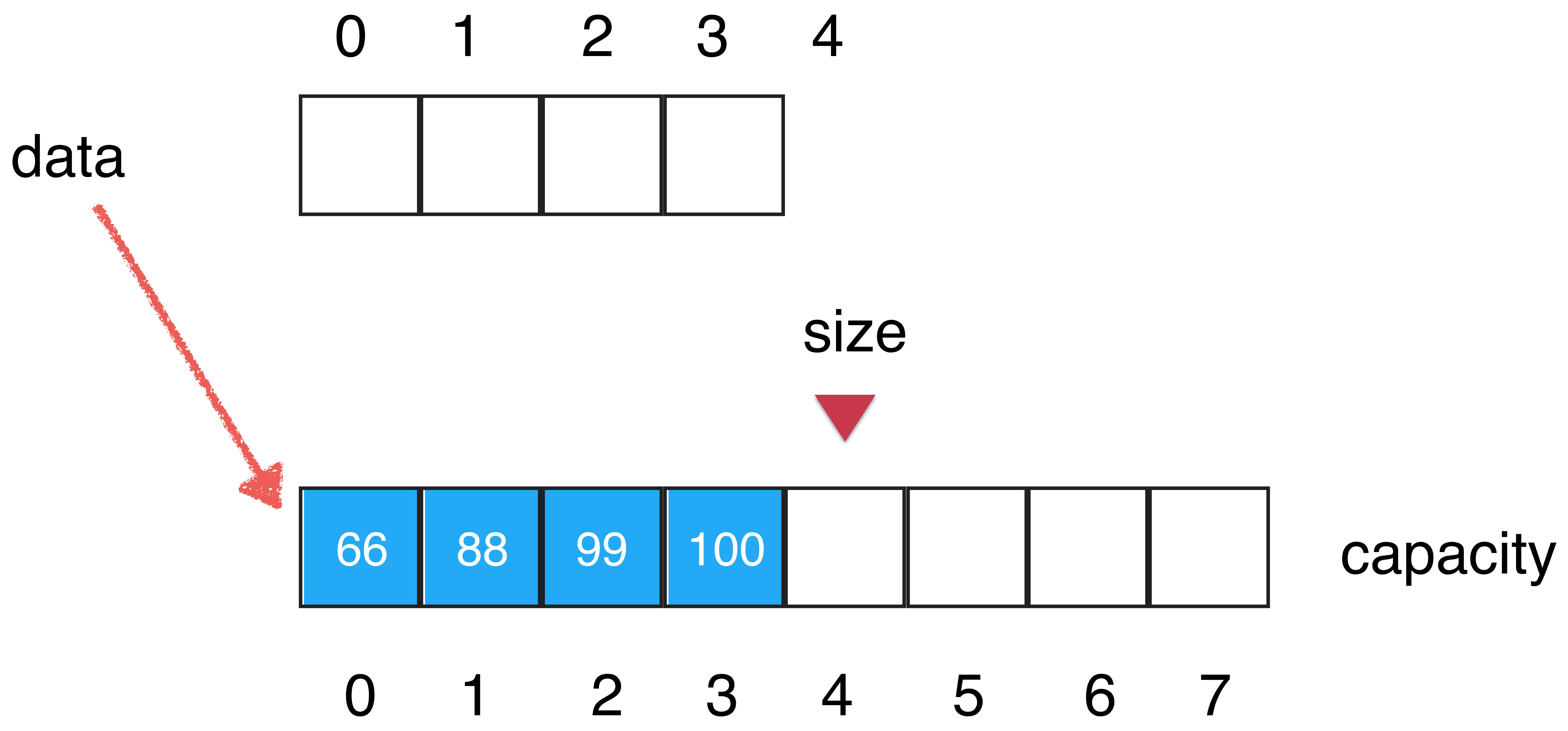
动态数组



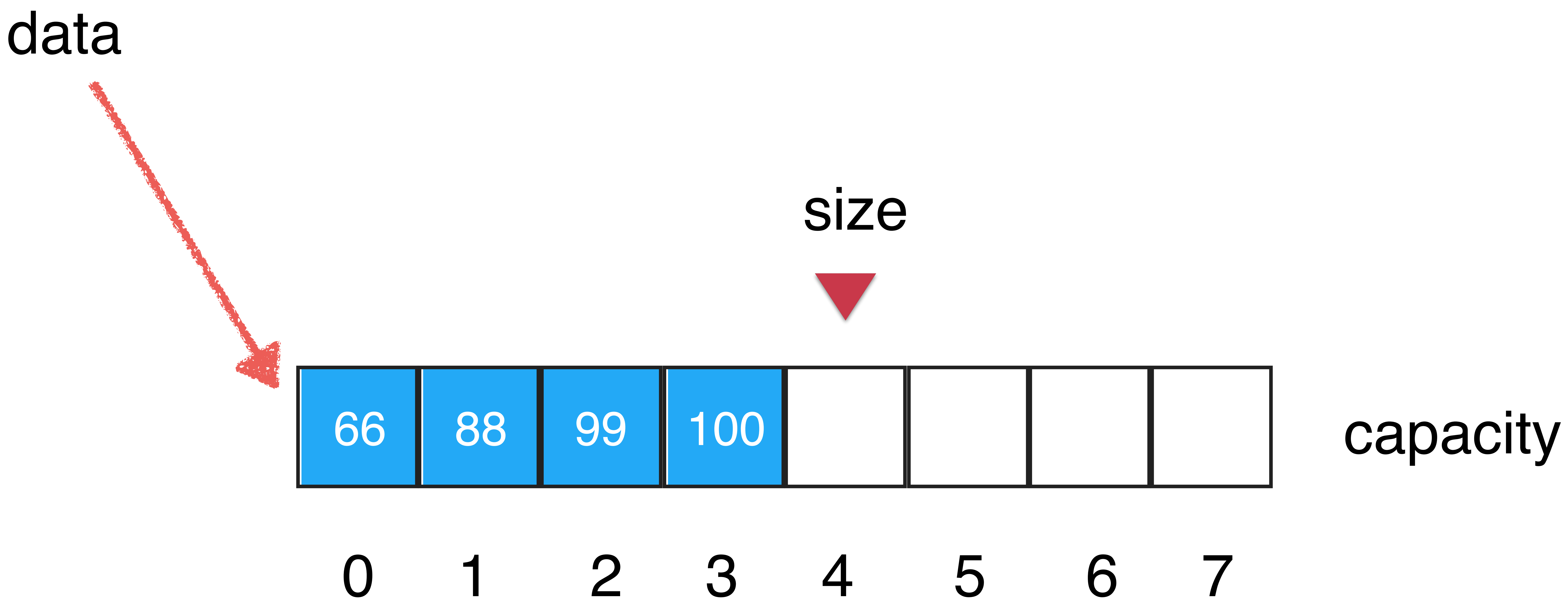
动态数组



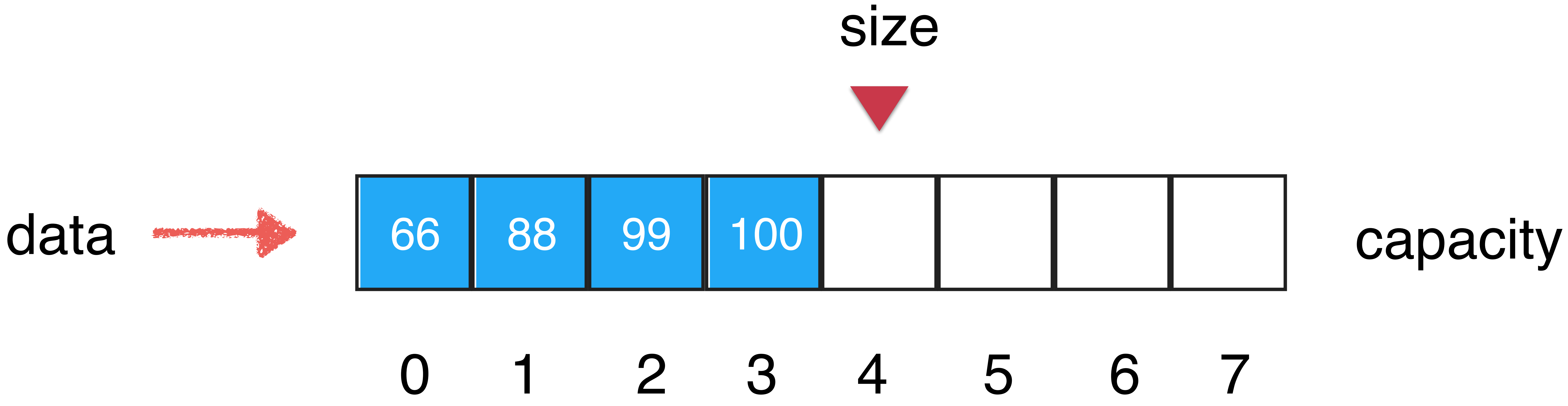
动态数组



动态数组



动态数组



实践：动态数组

简单的时间复杂度分析

简单的时间复杂度分析

- O(1) , O(n) , O(lgn) , O(nlogn) , O(n^2)

- 大O描述的是算法的运行时间和输入数据之间的关系

```
public static int sum(int[] nums){
        int sum = 0;
        for(int num: nums) sum += num;
        return sum;
}
```

O(n)

n是nums中的元素个数

算法和n呈线性关系

简单的时间复杂度分析

```java
public static int sum(int[] nums){
        int sum = 0;
        for(int num: nums) sum += num;
        return sum;
}
```

O(n)

n是nums中的元素个数

算法和n呈线性关系

- 为什么要用大O，叫做O(n)?　　忽略常数。实际时间 T = c1*n + c2

简单的时间复杂度分析

- 为什么要用大O，叫做O(n)?　　忽略常数。实际时间 $T = c1*n + c2$

$T = 2*n + 2$ 　　O(n)

$T = 2000*n + 10000$ 　　O(n)

$T = 1*n*n + 0$ 　　O(n^2)

$T = 2*n*n + 300n + 10$ 　　O(n^2)

渐进时间复杂度

描述n趋近于无穷的情况

分析动态数组的时间复杂度

- 添加操作　　O(n)

addLast(e)　　$O(1)$

addFirst(e)　　$O(n)$

add(index, e)　　$O(n/2) = O(n)$

$O(n)$　　resize　　$O(n)$

最坏情况

严格计算需要一些概率论知识

分析动态数组的时间复杂度

- 删除操作　　O(n)

removeLast(e)　　O(1)

removeFirst(e)　　O(n)　　O(n)　　resize　　O(n)

remove(index, e)　　O(n/2) = O(n)

分析动态数组的时间复杂度

- 修改操作　　O(1)

  set(index, e)　　O(1)

分析动态数组的时间复杂度

- 查找操作

get(index)	O(1)
contains(e)	O(n)
find(e)	O(n)

分析动态数组的时间复杂度

- 增： O(n)
- 删： O(n)

如果只对最后一个元素操作
依然是O(n)? 因为resize?

- 改： 已知索引 O(1)；未知索引 O(n)
- 查： 已知索引 O(1)；未知索引 O(n)

均摊复杂度分析和防止复杂度震荡

resize的复杂度分析

- 添加操作 O(n)

addLast(e)	O(1)	O(n)	resize O(n)
addFirst(e)	O(n)	最坏情况	
add(index, e)	O(n/2) = O(n)		

resize的复杂度分析

- 添加操作　　O(n)

addLast(e)　　**O(1)**

addFirst(e)　　O(n)　　　O(n)　　　**resize　O(n)**

add(index, e)　　O(n/2) = O(n)　　　最坏情况

resize的复杂度分析

resize　O(n)

假设当前capacity = 8，并且每一次添加操作都使用addLast

1　1　1　1　1　1　1　1　8 + 1

9次addLast操作，触发resize，总共进行了17次基本操作

resize的复杂度分析

resize　O(n)

9次addLast操作，触发resize，总共进行了17次基本操作

平均，每次addLast操作，进行2次基本操作

假设capacity = n，n+1次addLast，触发resize，总共进行2n+1次基本操作

平均，每次addLast操作，进行2次基本操作

resize的复杂度分析

resize　O(n)

平均，每次addLast操作，进行2次基本操作

这样均摊计算，时间复杂度是O(1)的！

在这个例子里，这样均摊计算，比计算最坏情况有意义。

均摊复杂度 amortized time complexity

resize　O(n)

addLast 的均摊复杂度为O(1)

同理，我们看removeLast操作，均摊复杂度也为O(1)

复杂度震荡

但是，当我们同时看addLast和removeLast操作：

capacity = n

addLast O(n)

removeLast O(n)

复杂度震荡

但是，当我们同时看addLast和removeLast操作：

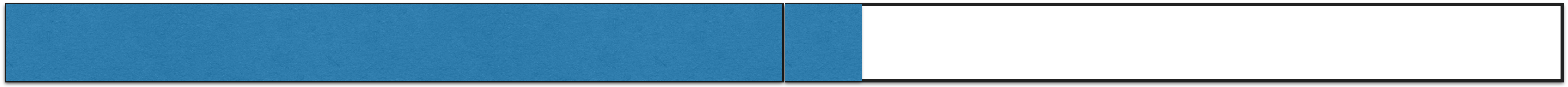
capacity = n

addLast	O(n)
removeLast	O(n)
addLast	O(n)
removeLast	O(n)

复杂度震荡

出现问题的原因：removeLast 时 resize 过于着急（Eager）

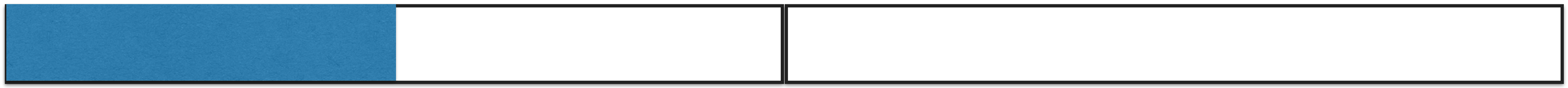
解决方案：Lazy



复杂度震荡

出现问题的原因：removeLast 时 resize 过于着急（Eager）

解决方案：Lazy



当 size == capacity / 4 时，才将capacity减半

实践：防止复杂度震荡

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