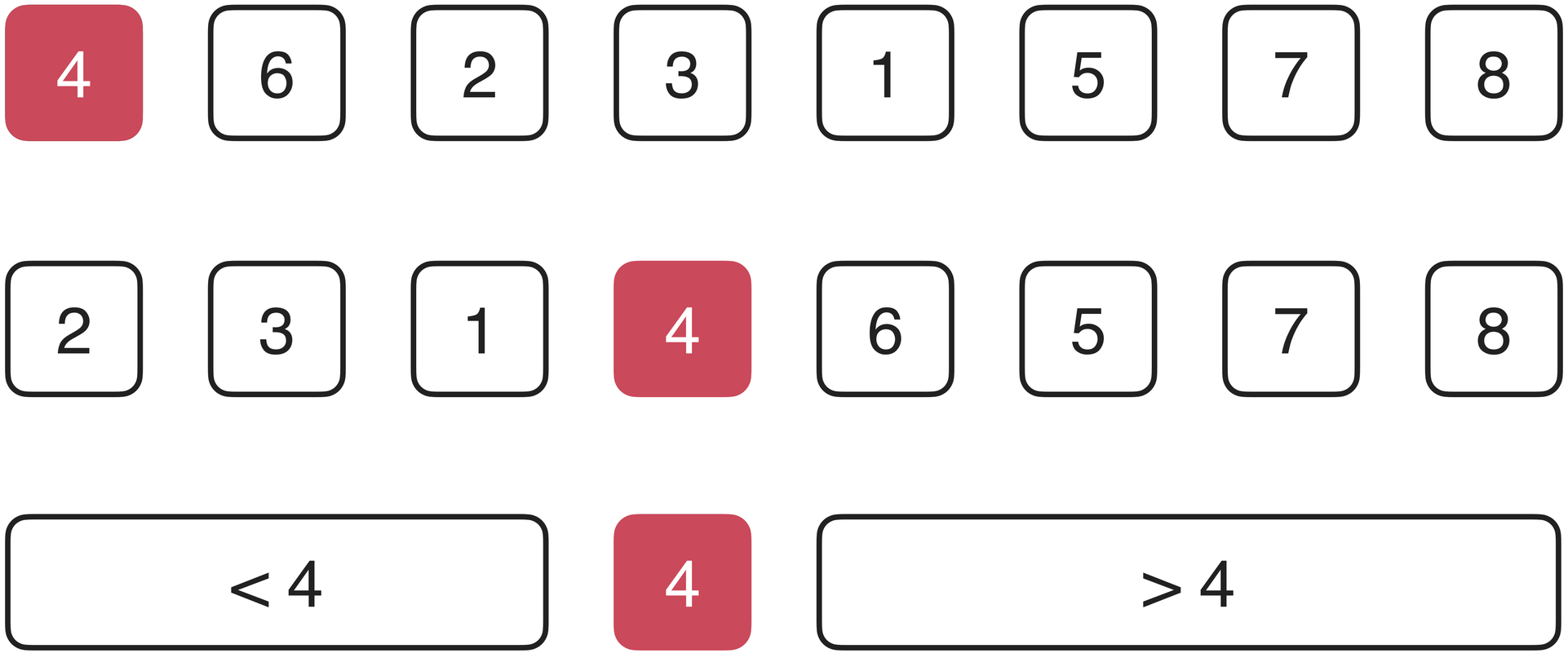
算法与数据结构体系课程

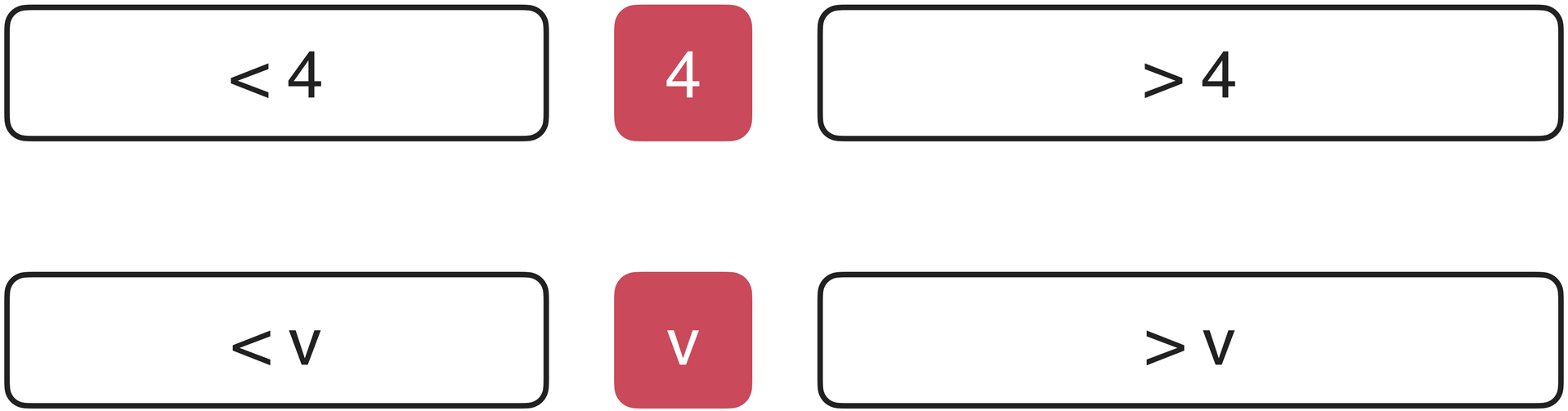
liuyubobobo

快速排序法

快速排序 Quick Sort

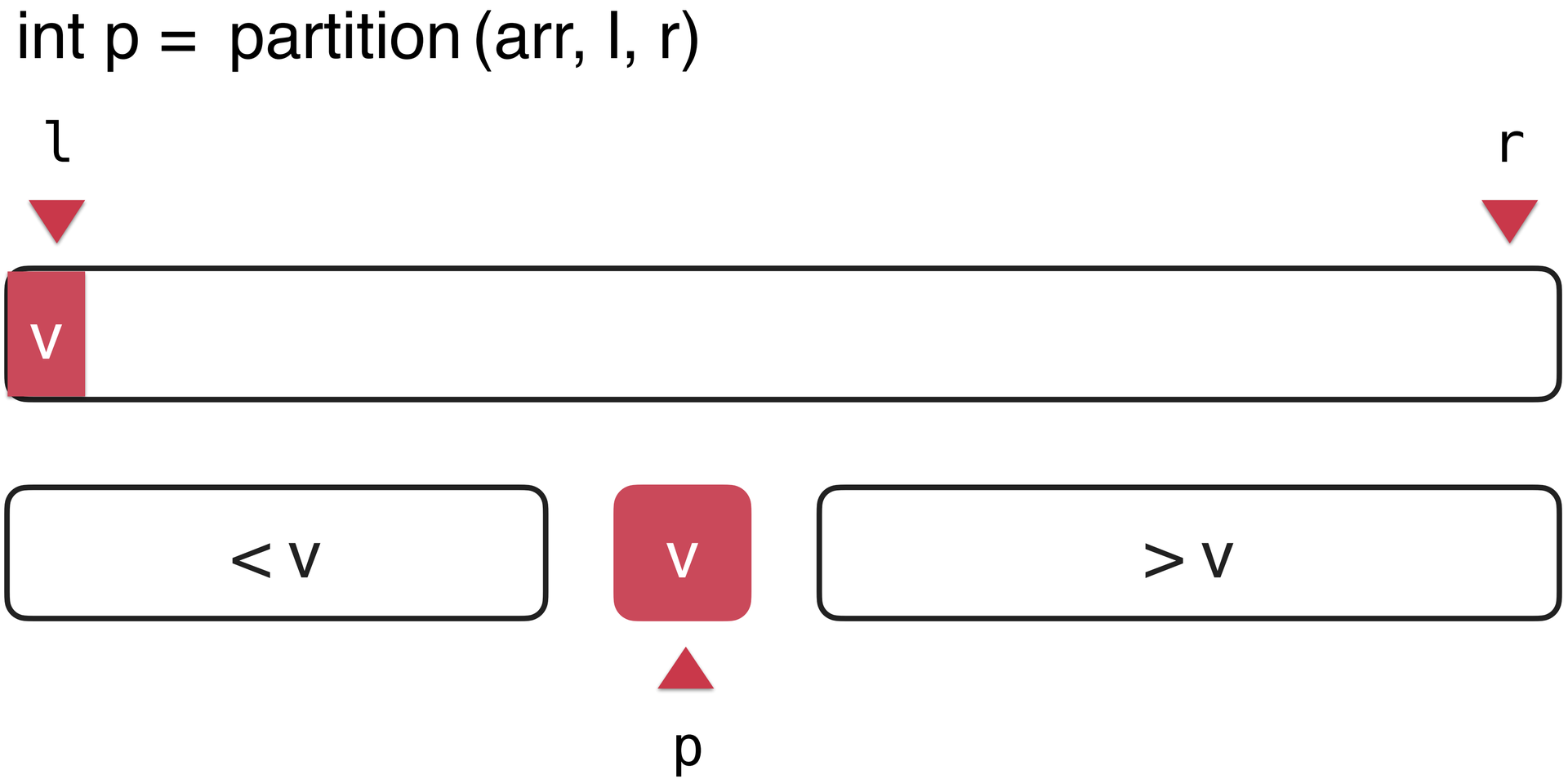


快速排序 Quick Sort



partition

快速排序 Quick Sort



快速排序法

int p = partition(arr, l, r)

注意递归函数的“宏观”语意

`QuickSort(arr, l, r)`

对 arr 的 [l, r] 部分排序

```
QuickSort(arr, l, r){

    if(l >= r) return;

    int p = partition(arr, l, r);

    // 对 arr[l, p - 1] 进行排序
    QuickSort(arr, l, p - 1);

    // 对 arr[p + 1, r] 进行排序
    QuickSort(arr, p + 1, r);

}
```

快速排序法

```
QuickSort(arr, l, r){

    if(l >= r) return;

    int p = partition(arr, l, r);

    // 对 arr[l, p - 1] 进行排序
    QuickSort(arr, l, p - 1);

    // 对 arr[p + 1, r] 进行排序
    QuickSort(arr, p + 1, r);

}
```

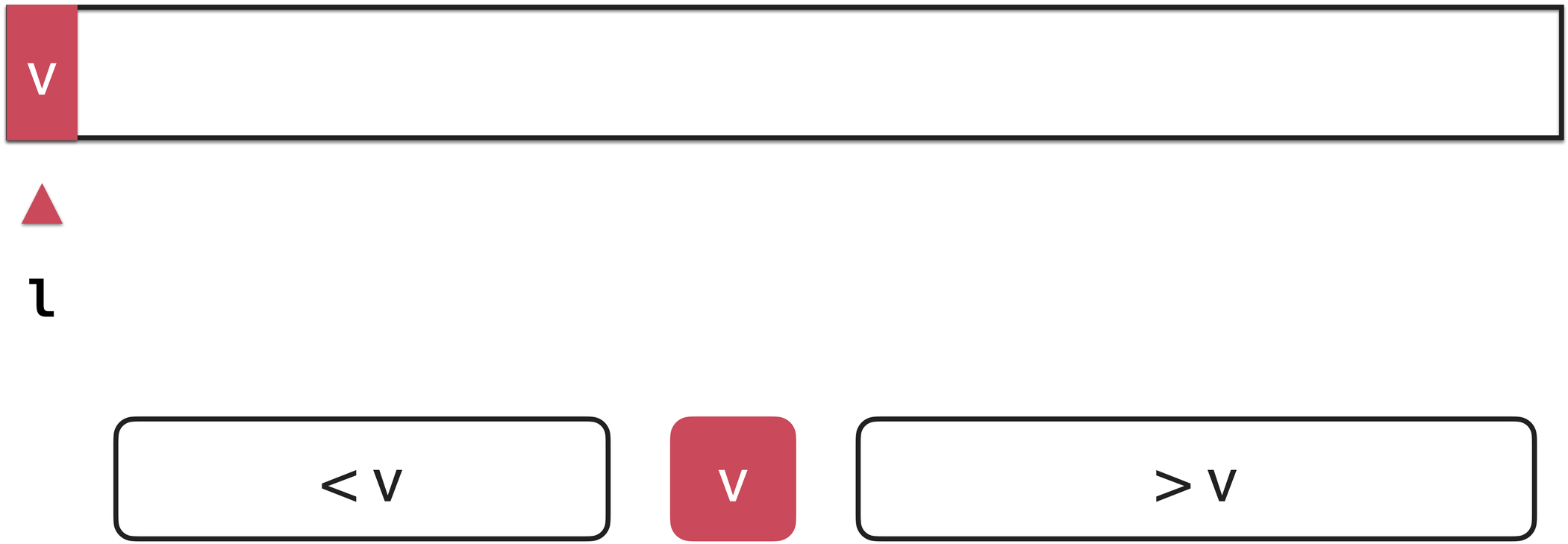
求解最基本问题

如何 partiton?

把原问题转化成更小的问题

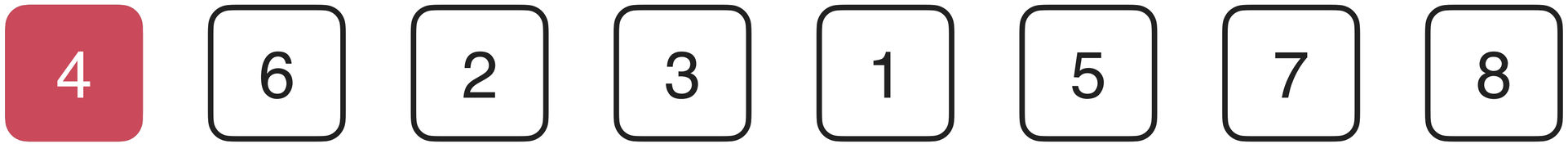
Partition

Partition



最基础的 partition

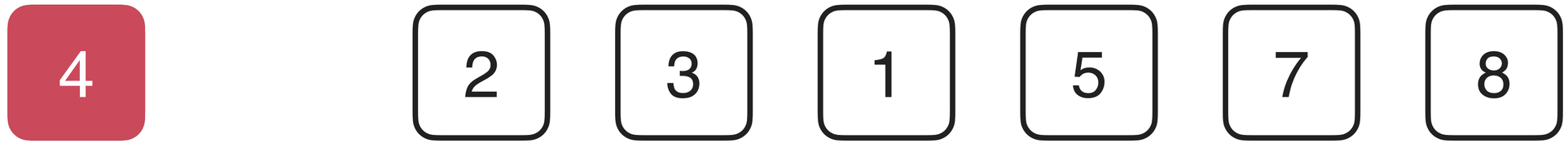
Partition



left :

right :

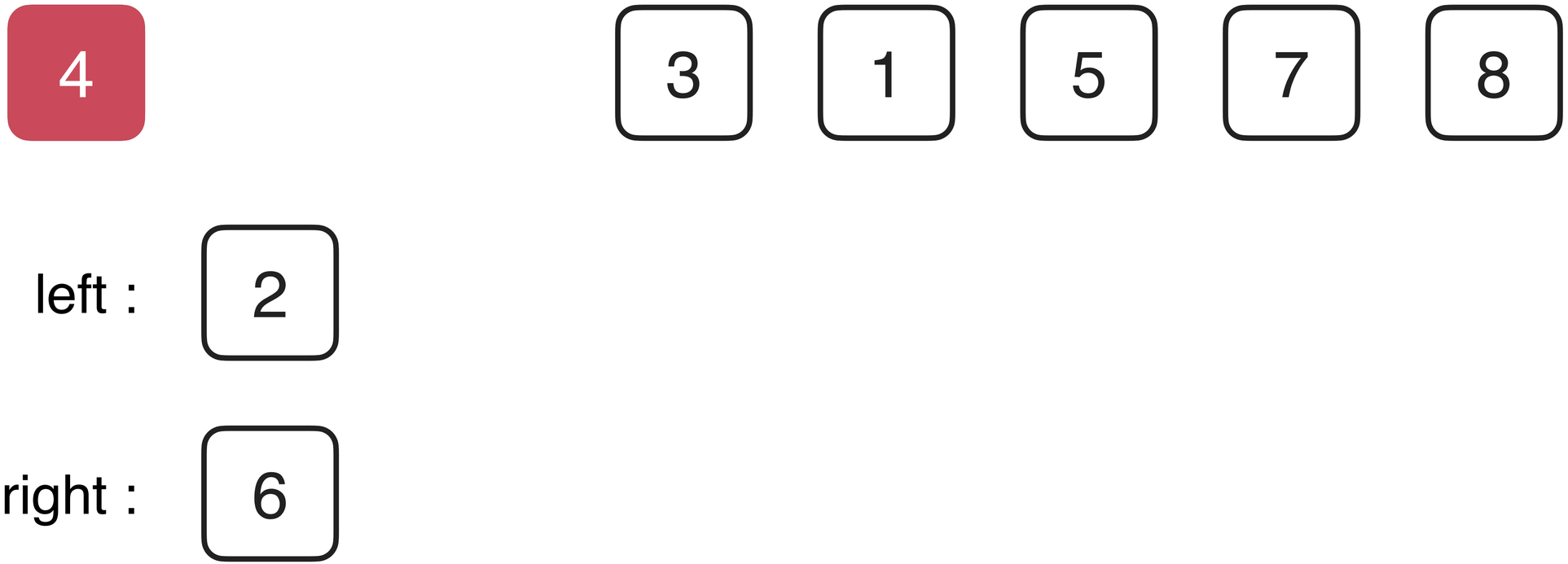
Partition



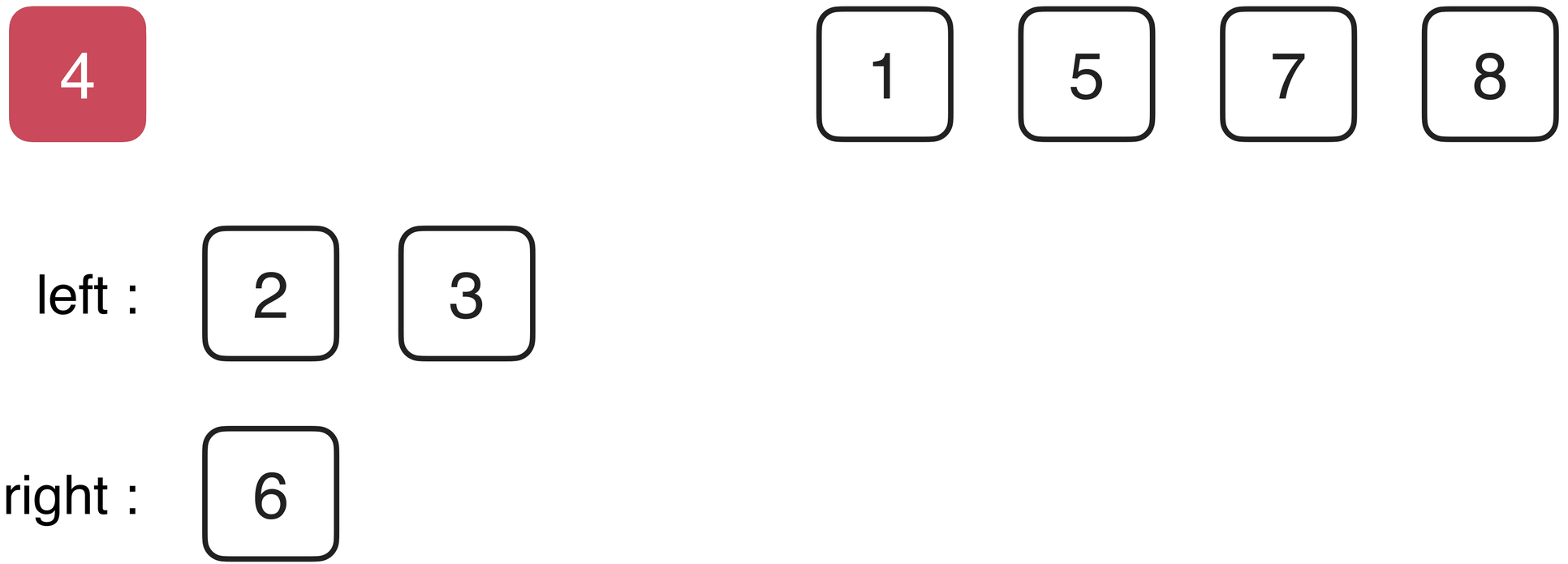
left :

right :

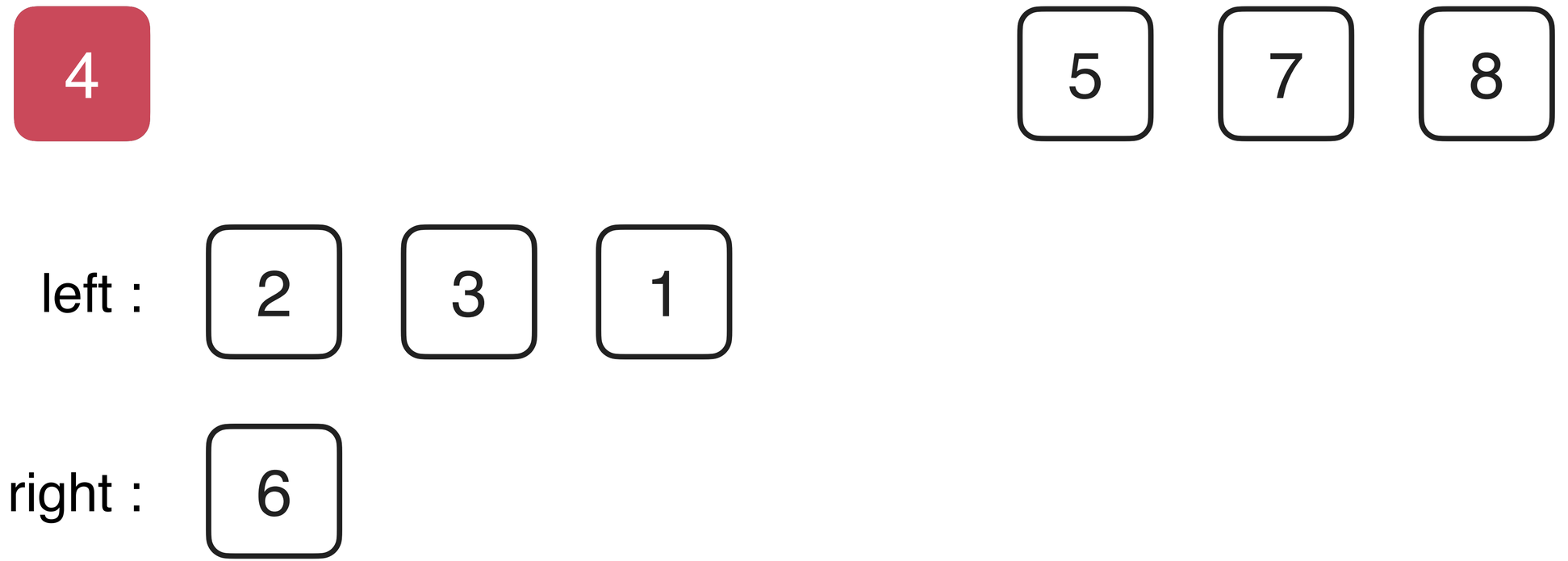
Partition



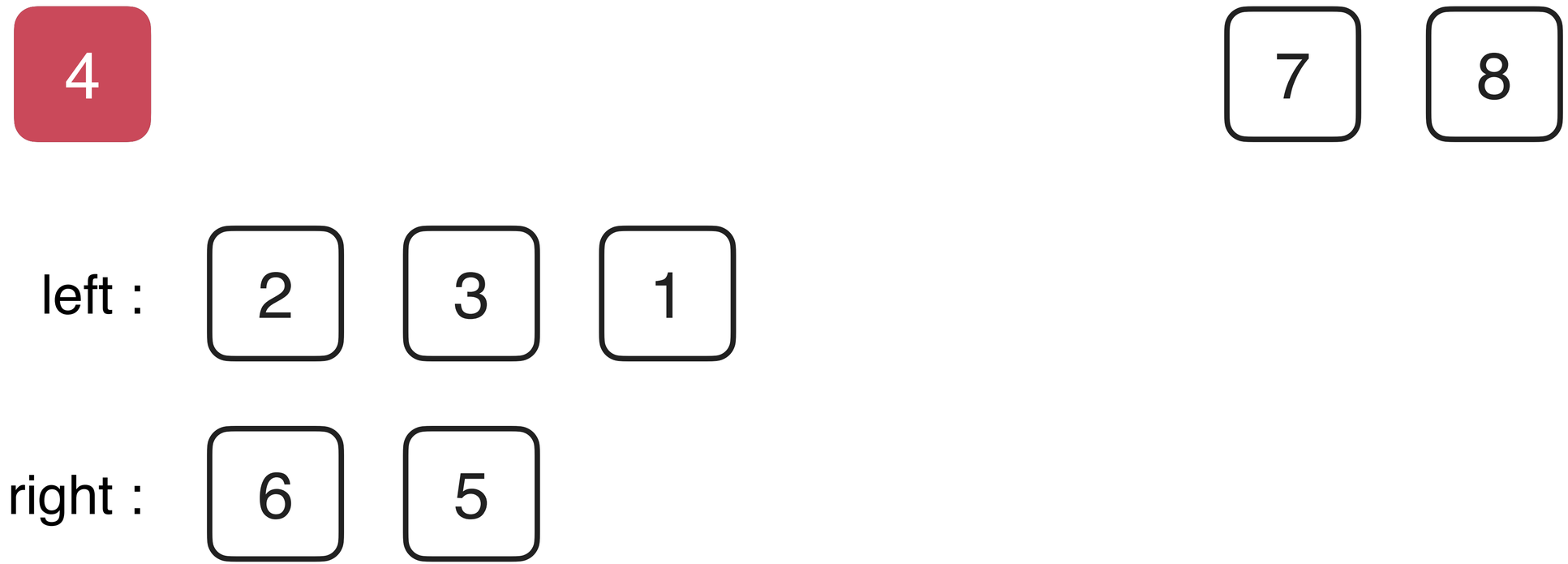
Partition



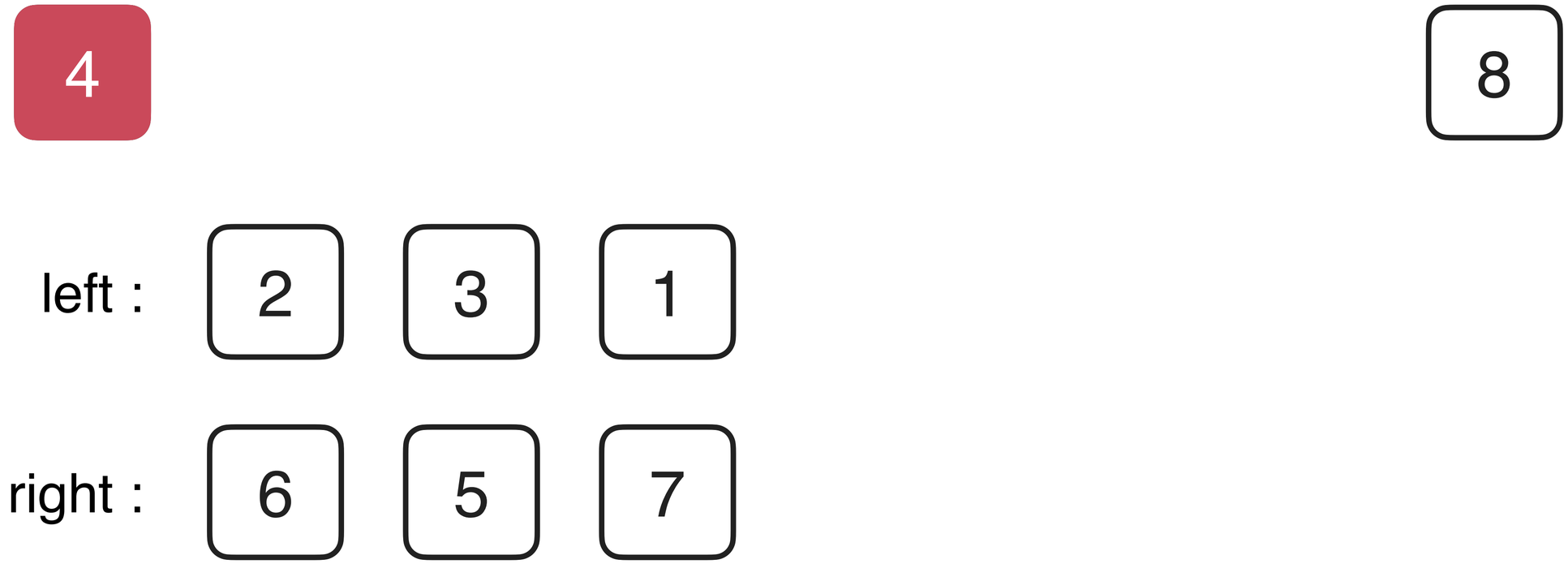
Partition



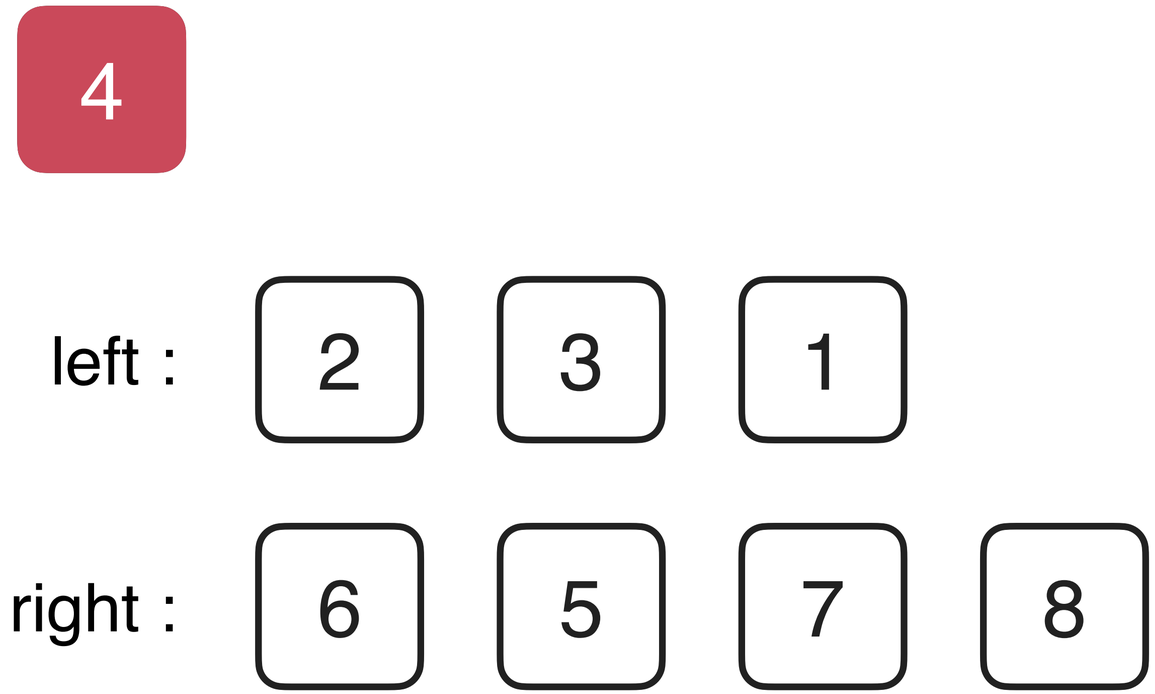
Partition



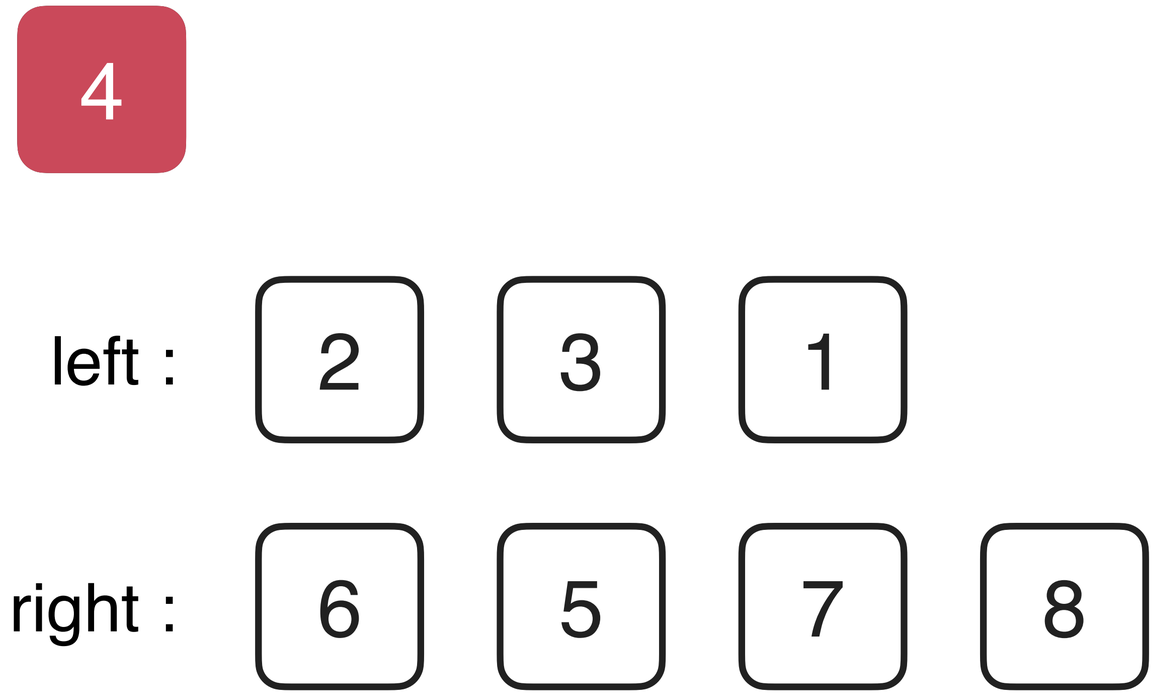
Partition



Partition



Partition



Partition

使用了额外空间，非原地排序

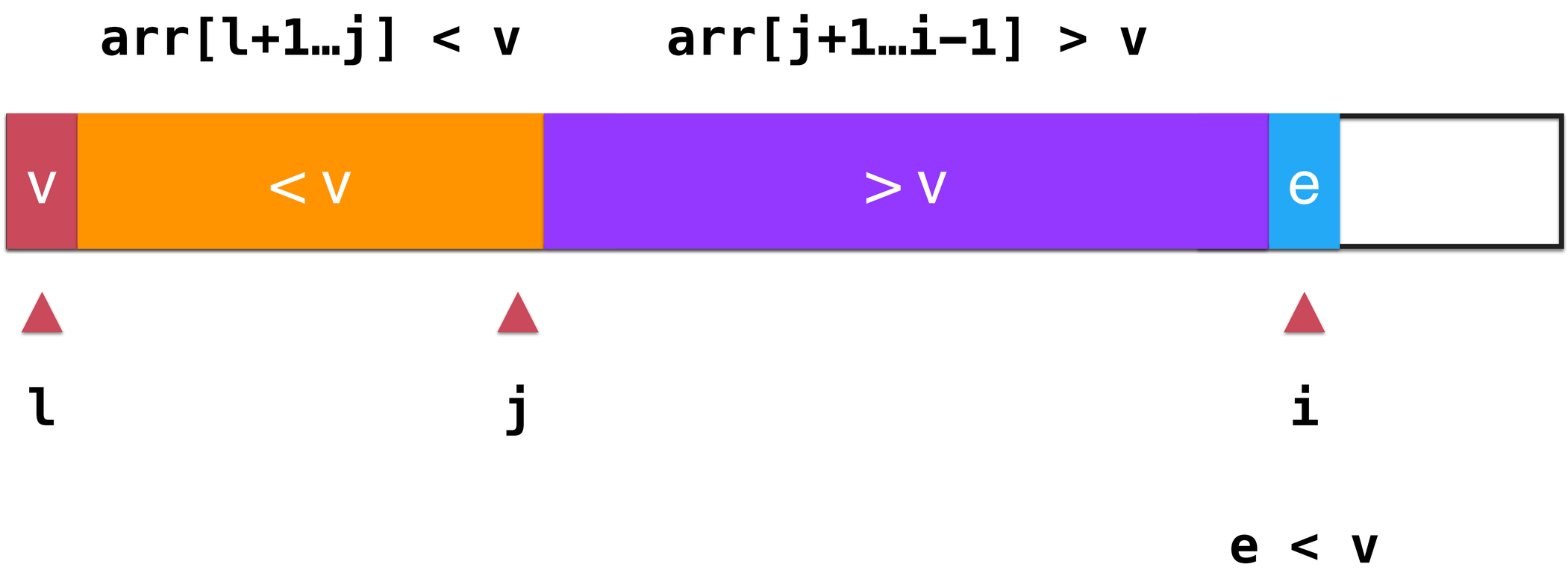
如何原地进行 partition?

Partition

Partition

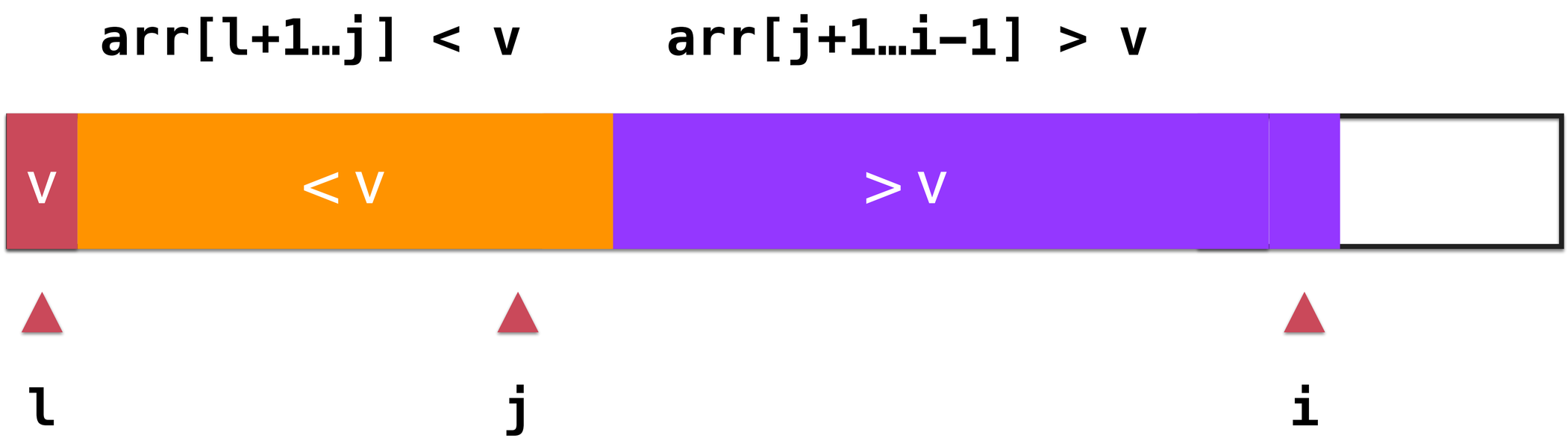
Partition

Partition

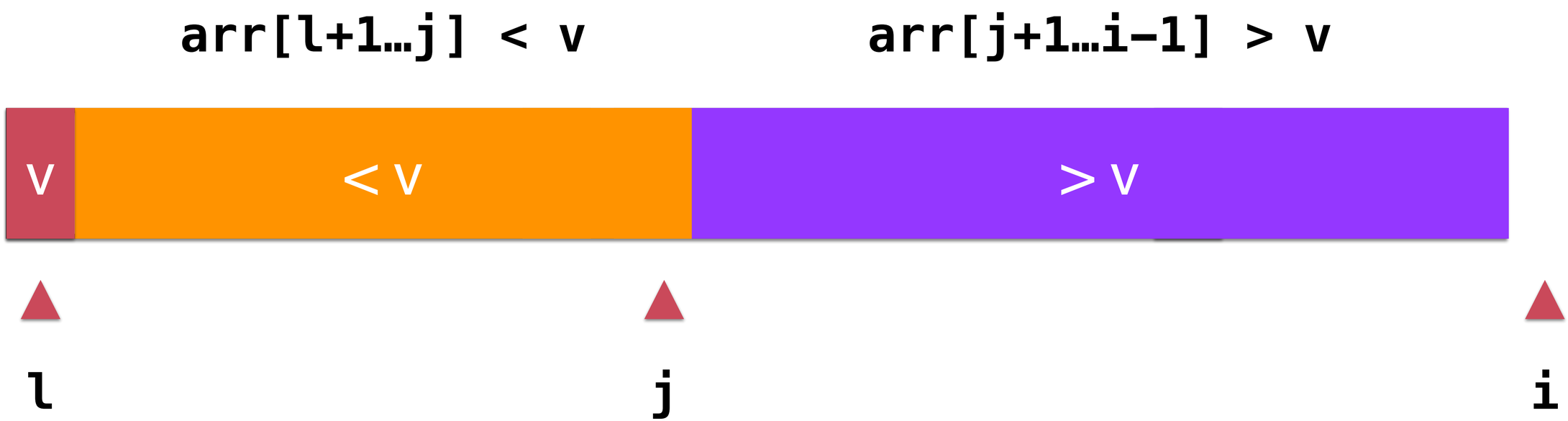


Partition

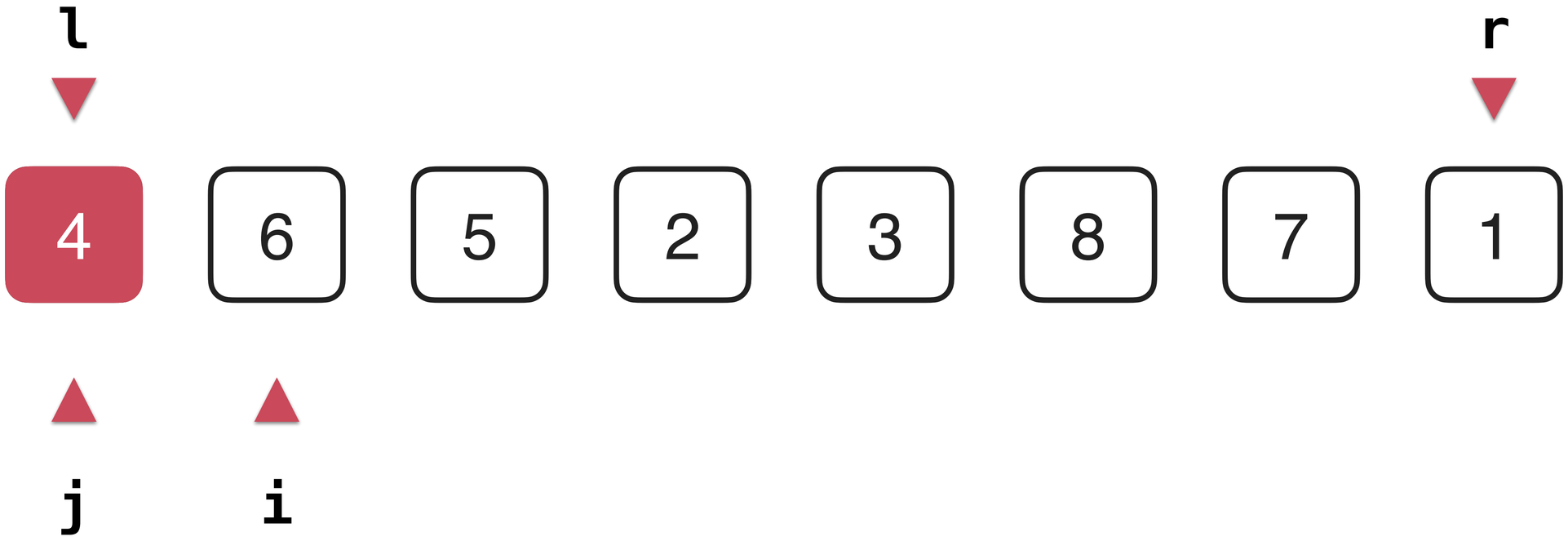
Partition



Partition

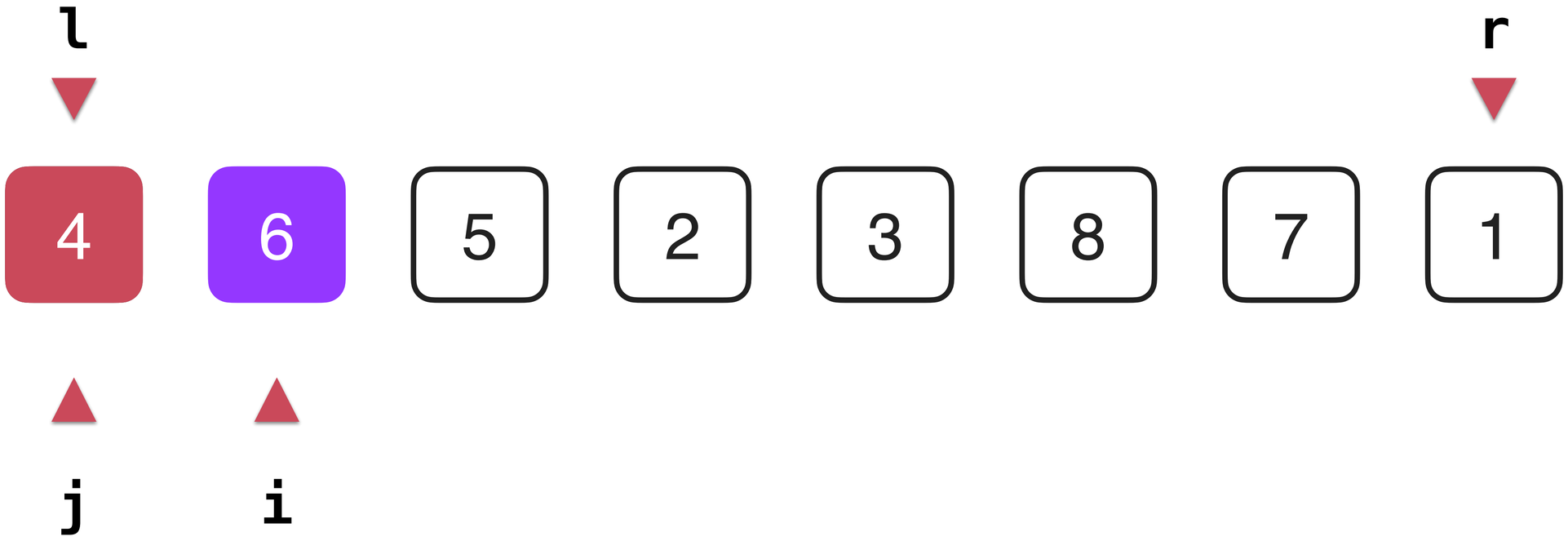


Partition



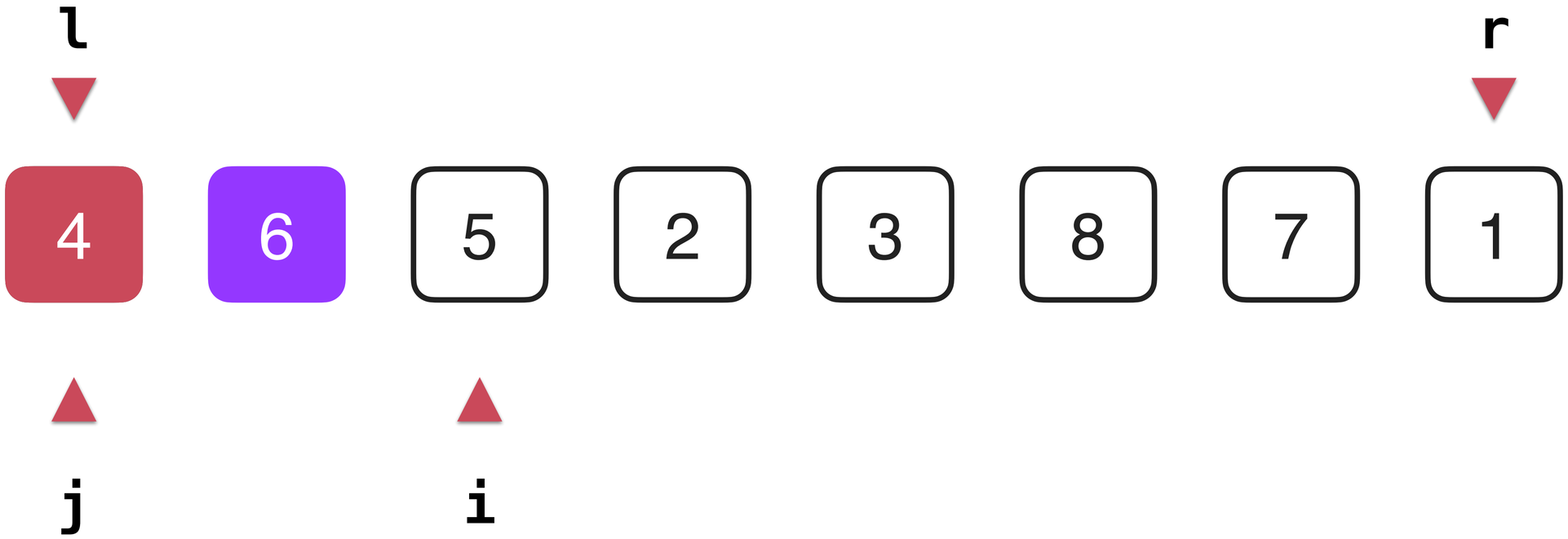
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



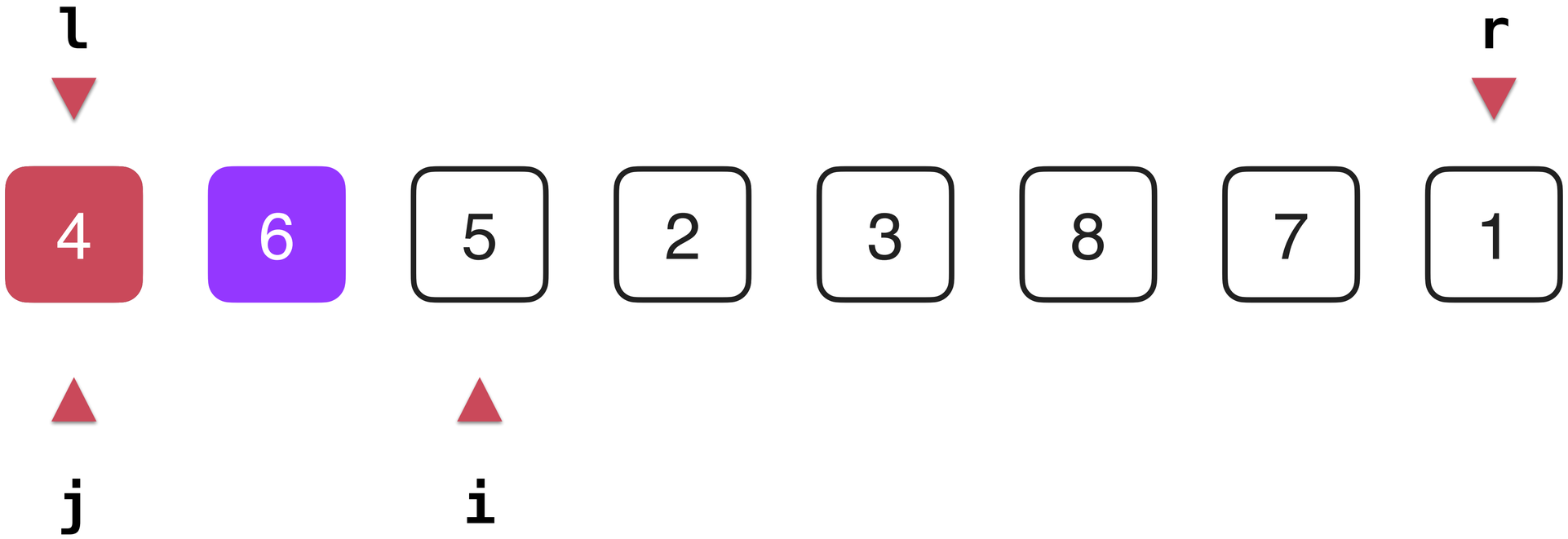
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



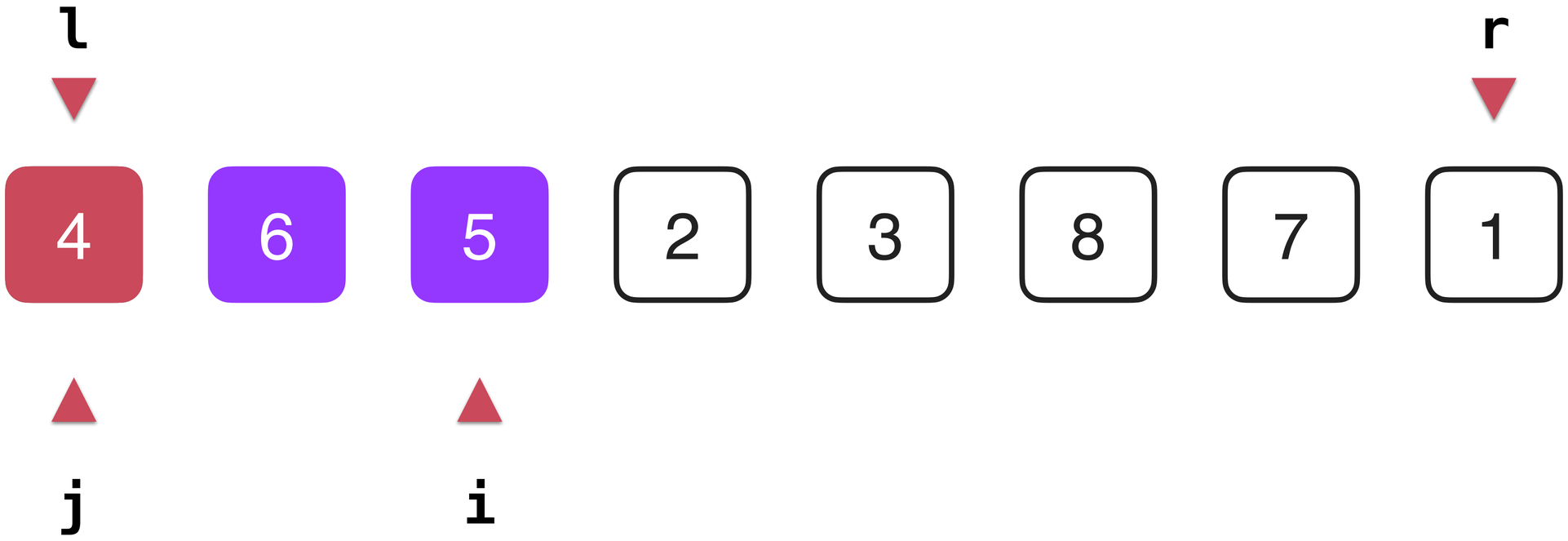
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



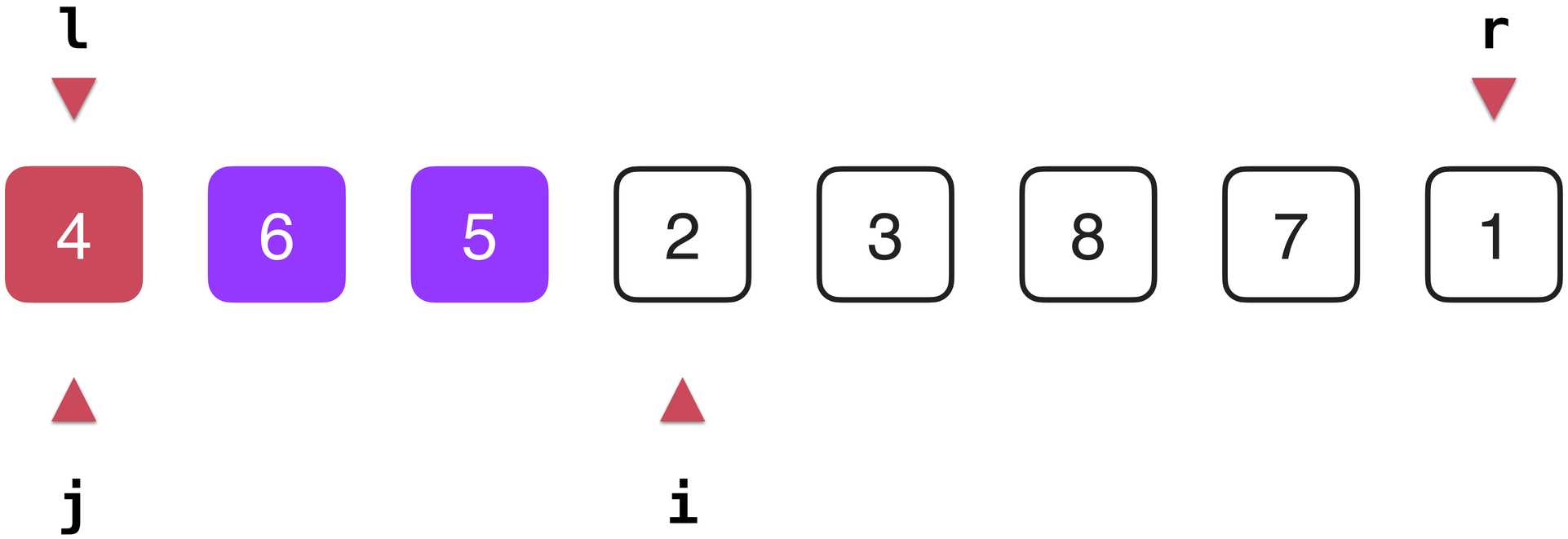
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



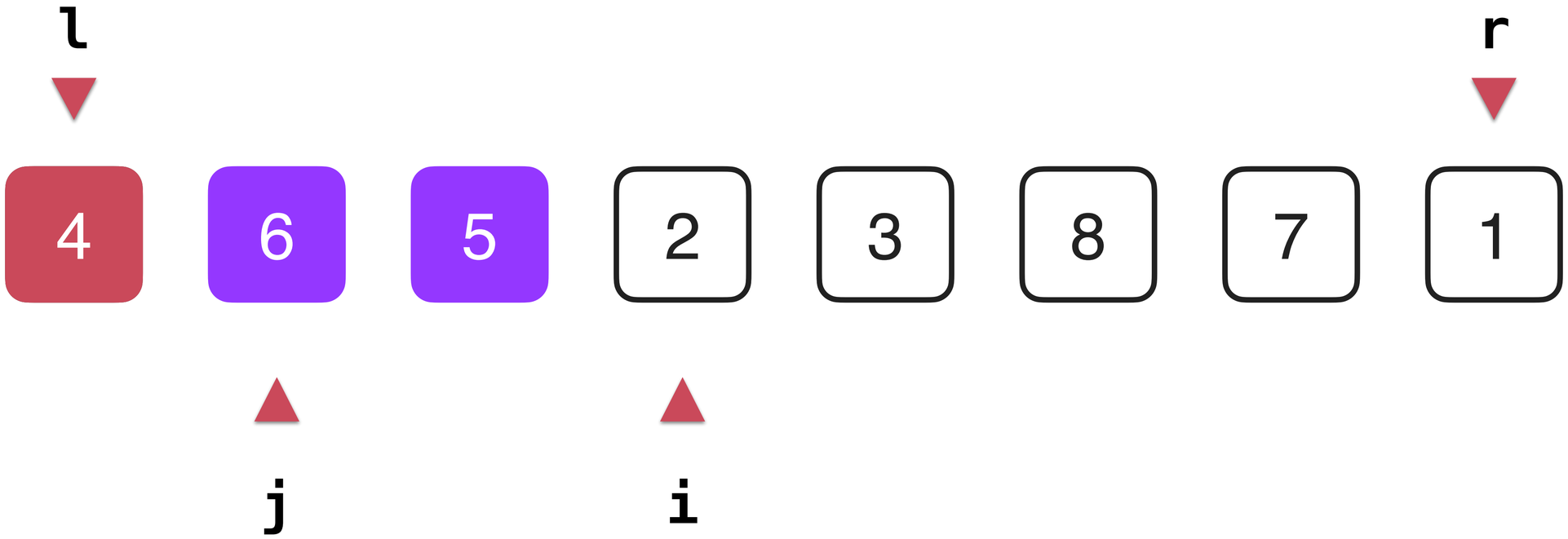
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



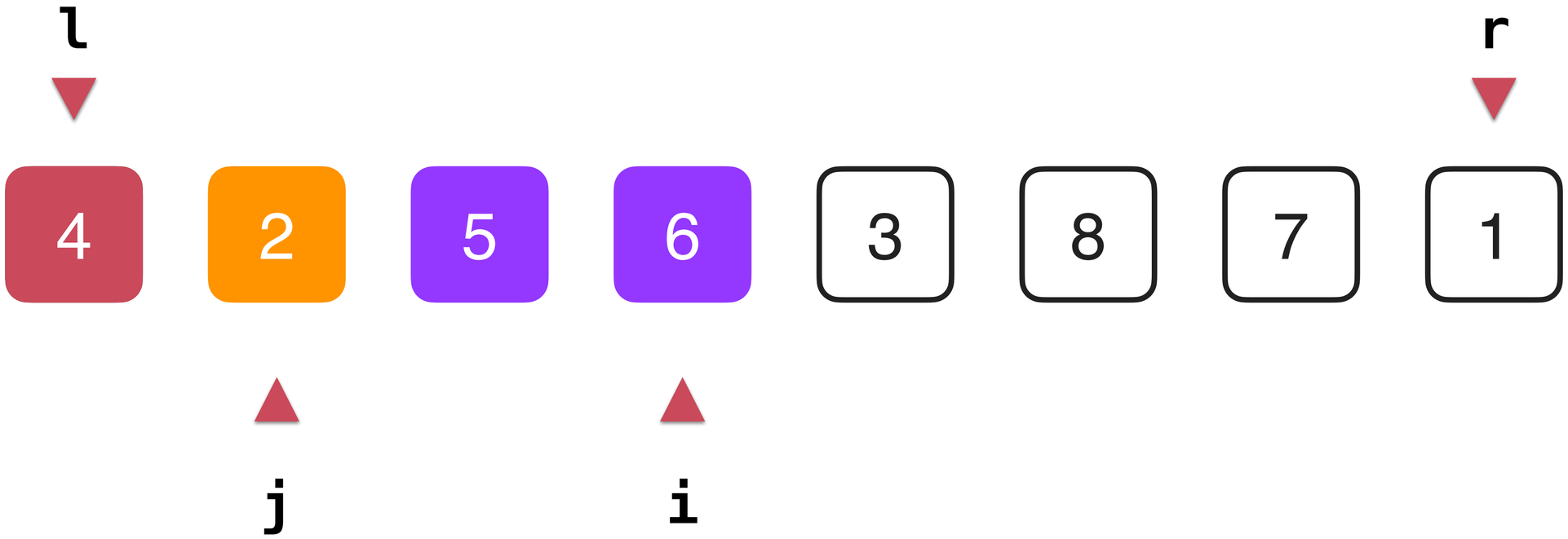
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



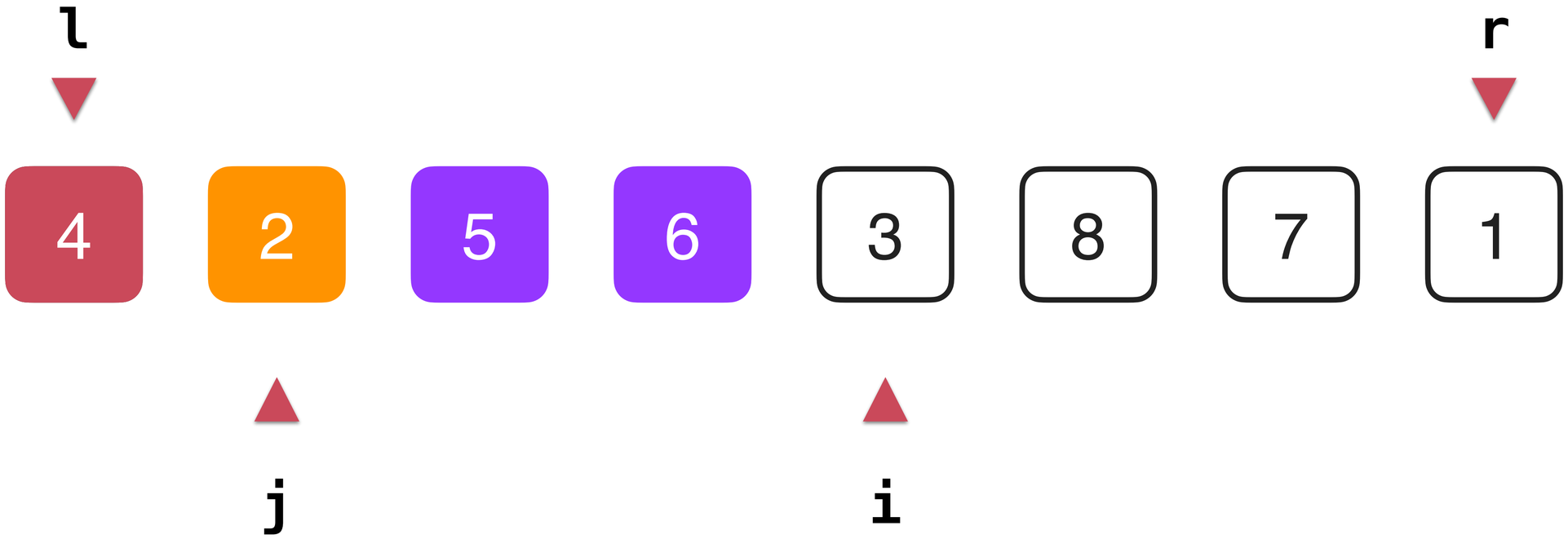
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



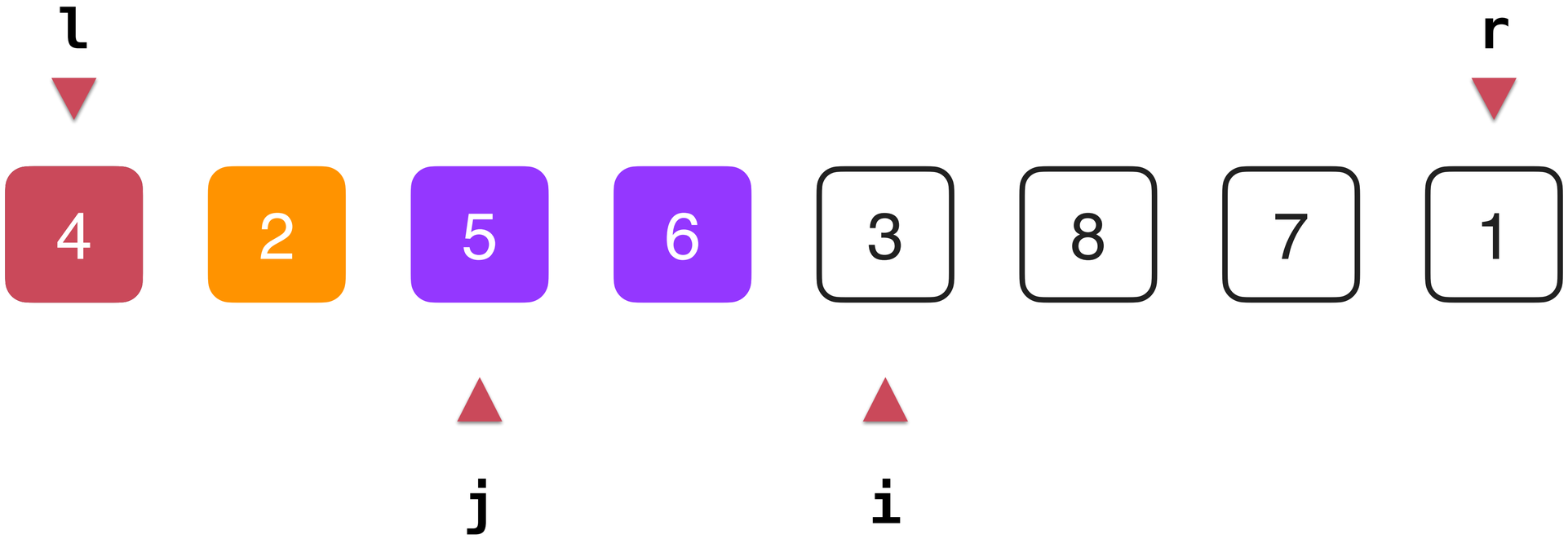
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



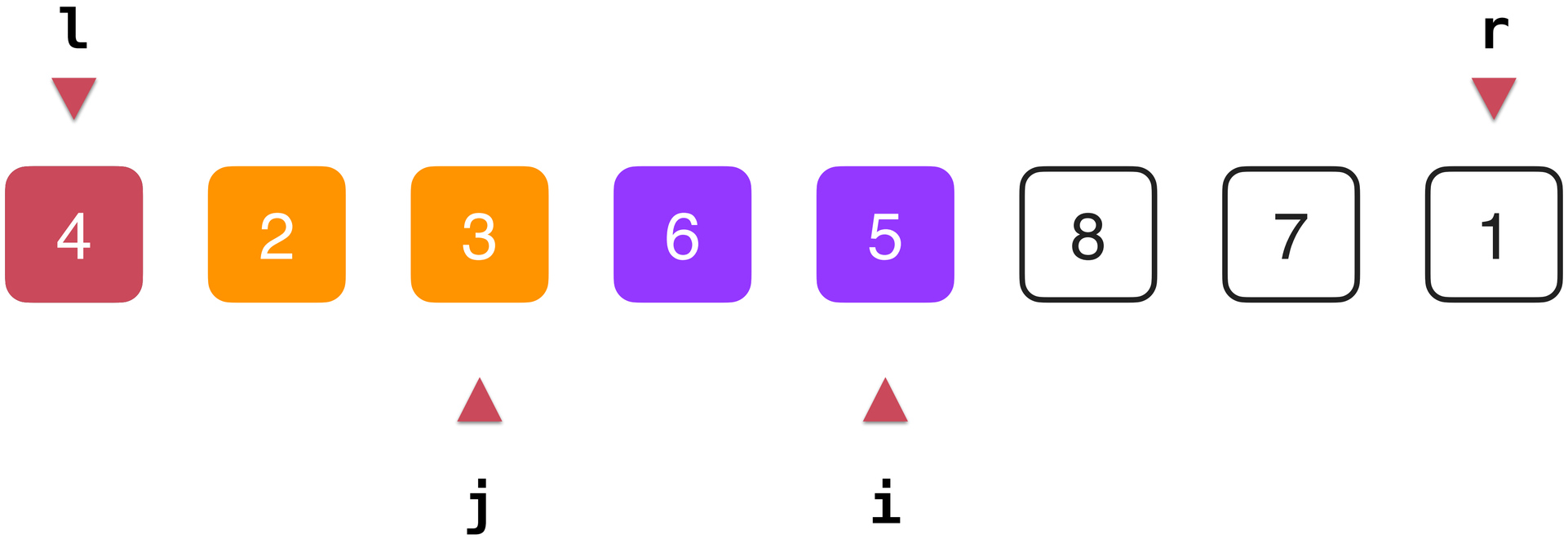
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



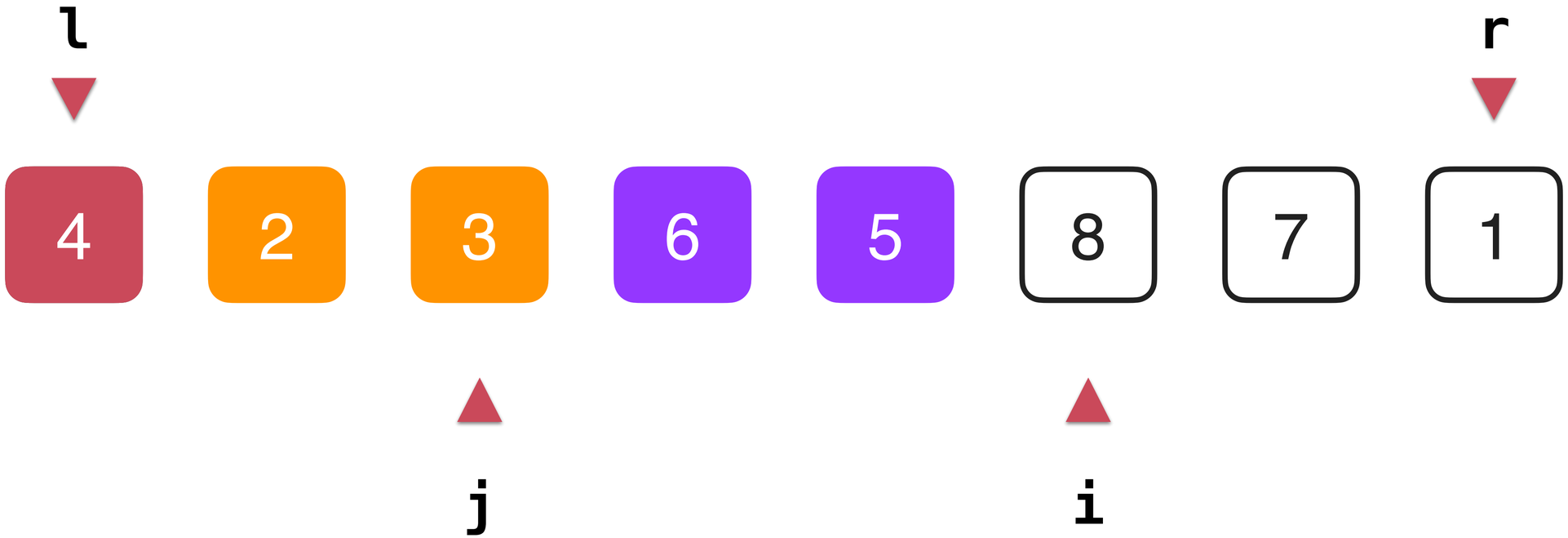
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



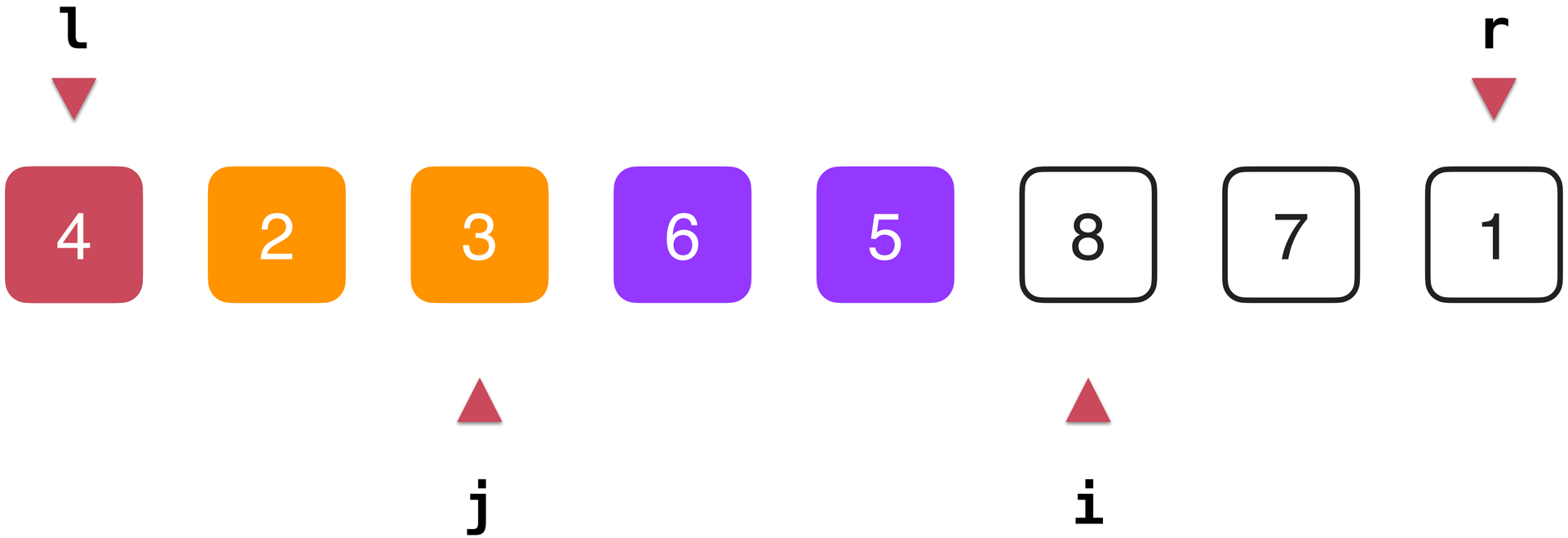
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition

`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition

`arr[l+1…j] < v` `arr[j+1…i-1] > v`

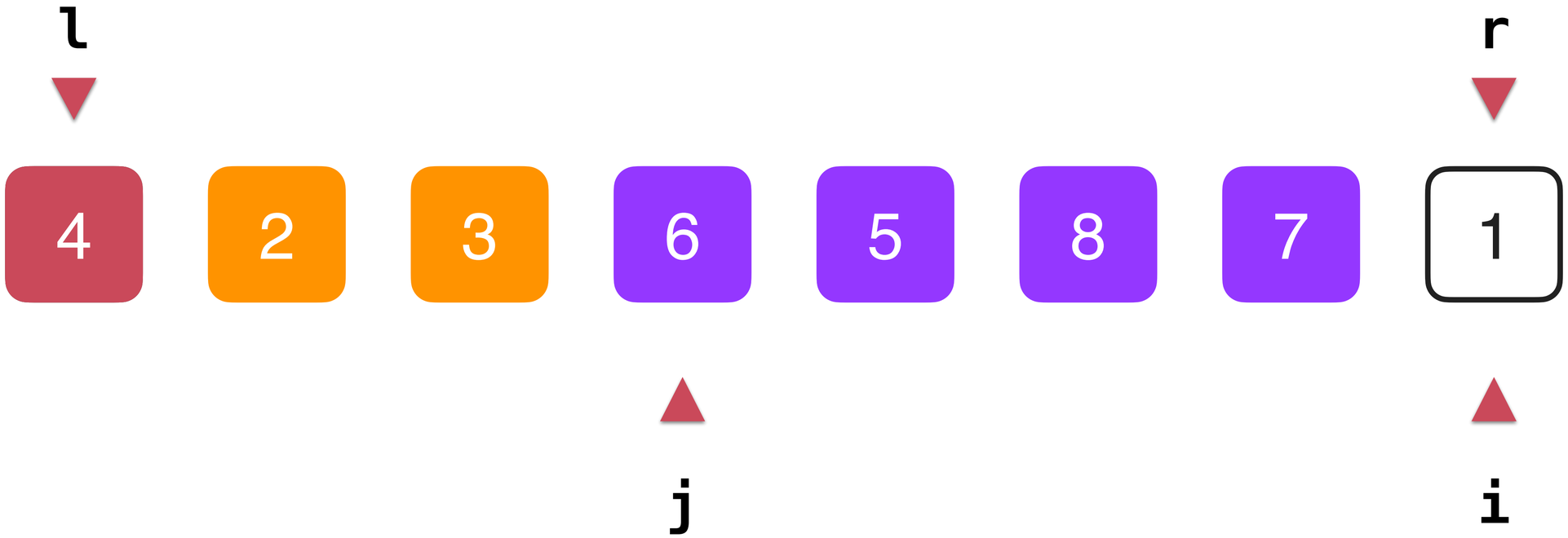
Partition

`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition

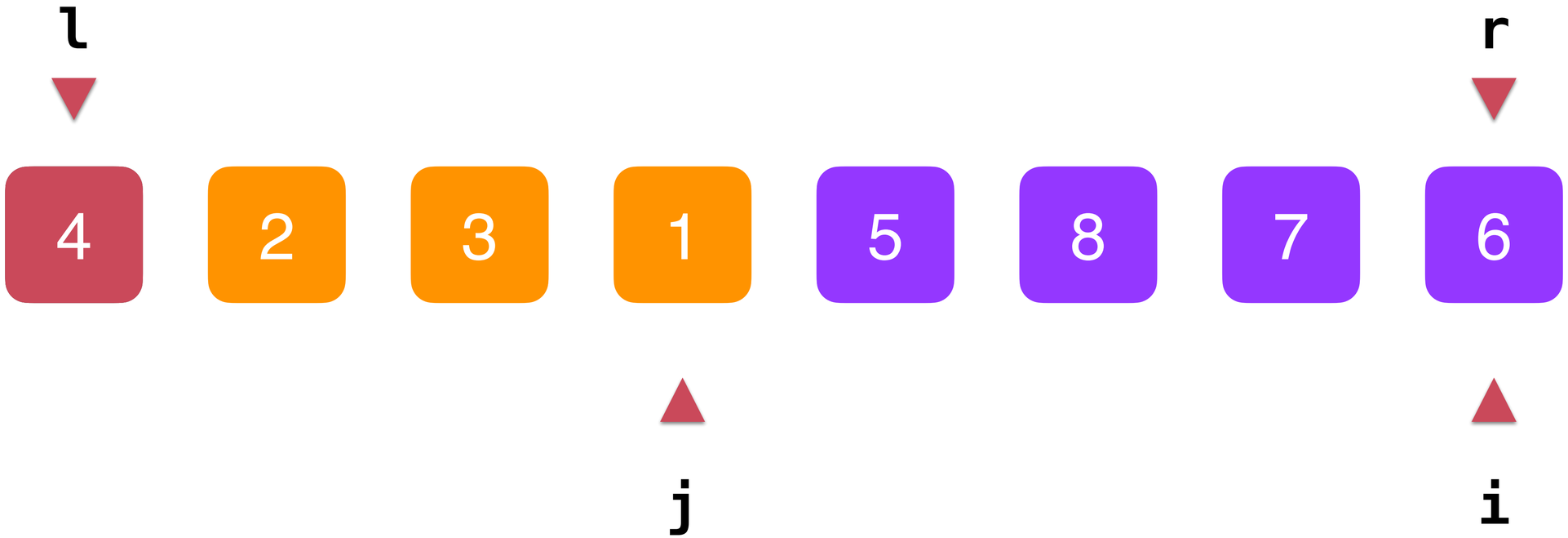
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



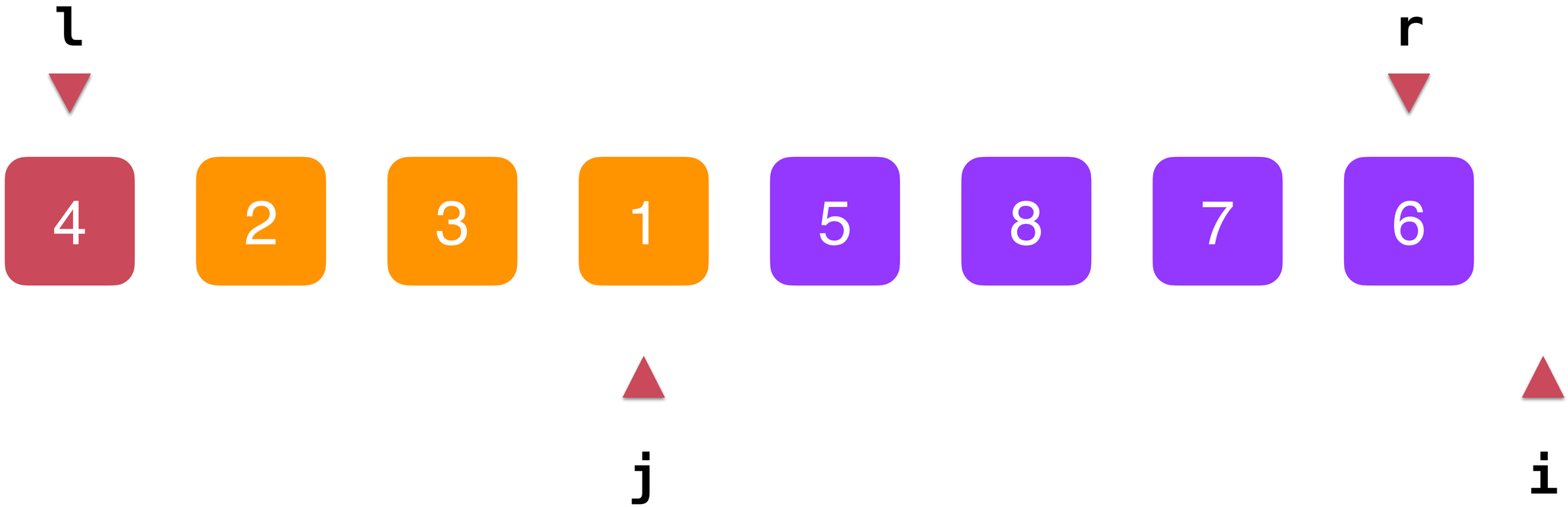
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



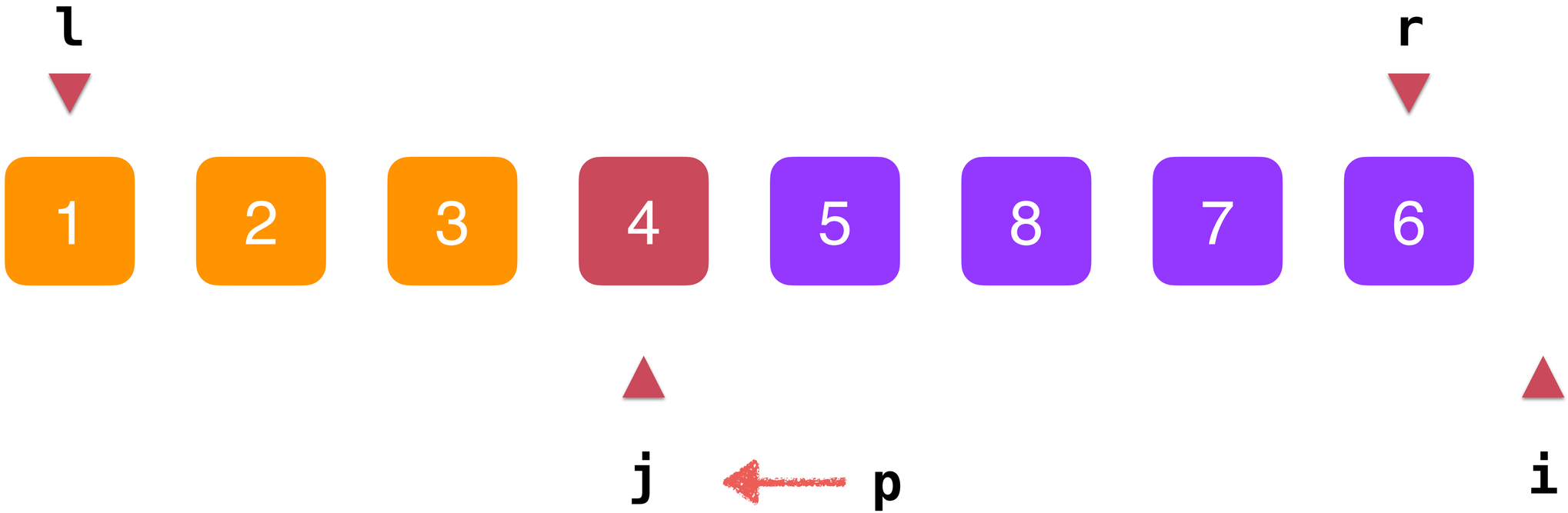
`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



`arr[l+1…j] < v` `arr[j+1…i-1] > v`

Partition



`arr[l+1…j] < v` `arr[j+1…i-1] > v`

实现 partition

实践：实现 partition

QuickSort 相关的作业

QuickSort 相关的作业

QuickSort 递归的微观解读

在 QuickSort 的递归函数中添加打印输出，

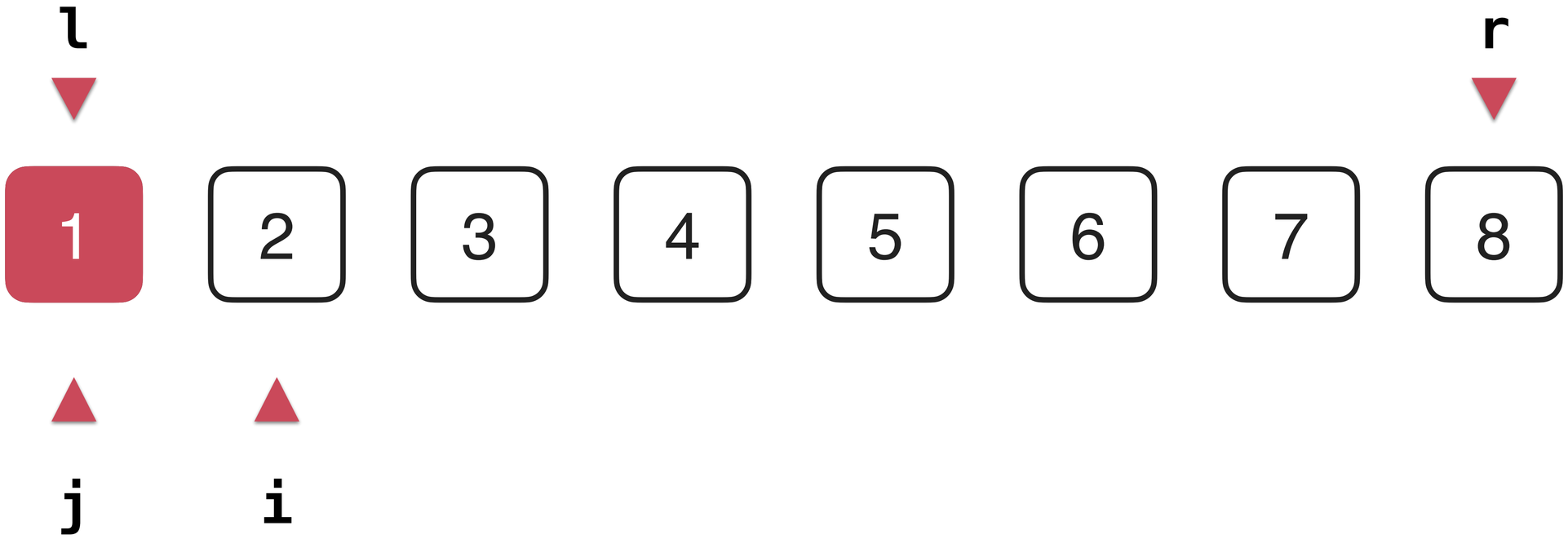
仔细理解 QuickSort 的递归执行过程

使用 InsertionSort 来优化 QuickSort

第一版 QuickSort 的问题

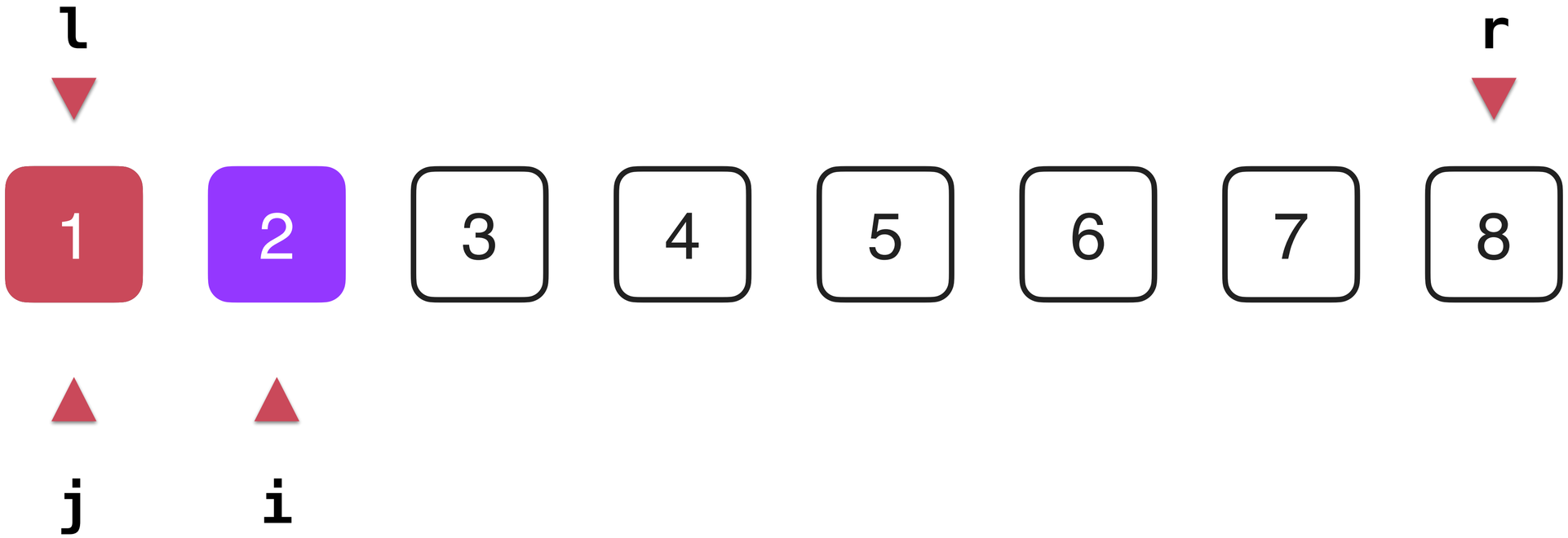
实践：有序数组的问题

第一版 QuickSort 的问题



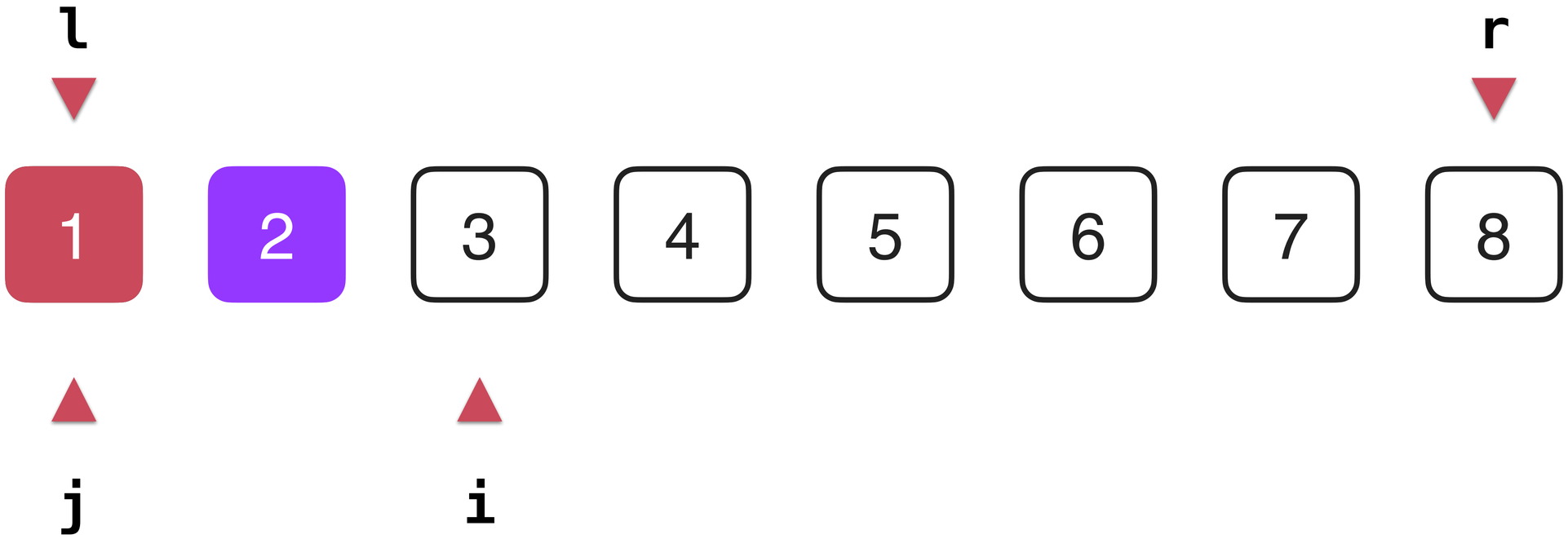
arr[l+1…j] < v　　arr[j+1…i-1] > v

第一版 QuickSort 的问题



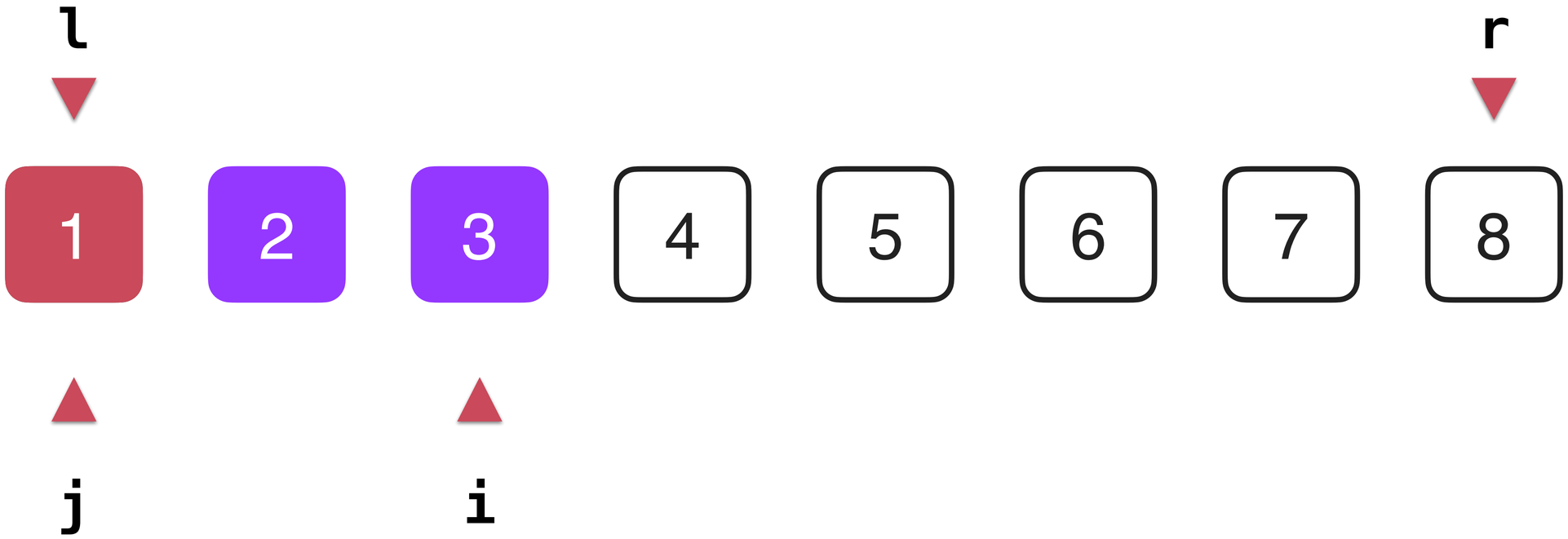
`arr[l+1…j] < v`　　`arr[j+1…i-1] > v`

第一版 QuickSort 的问题



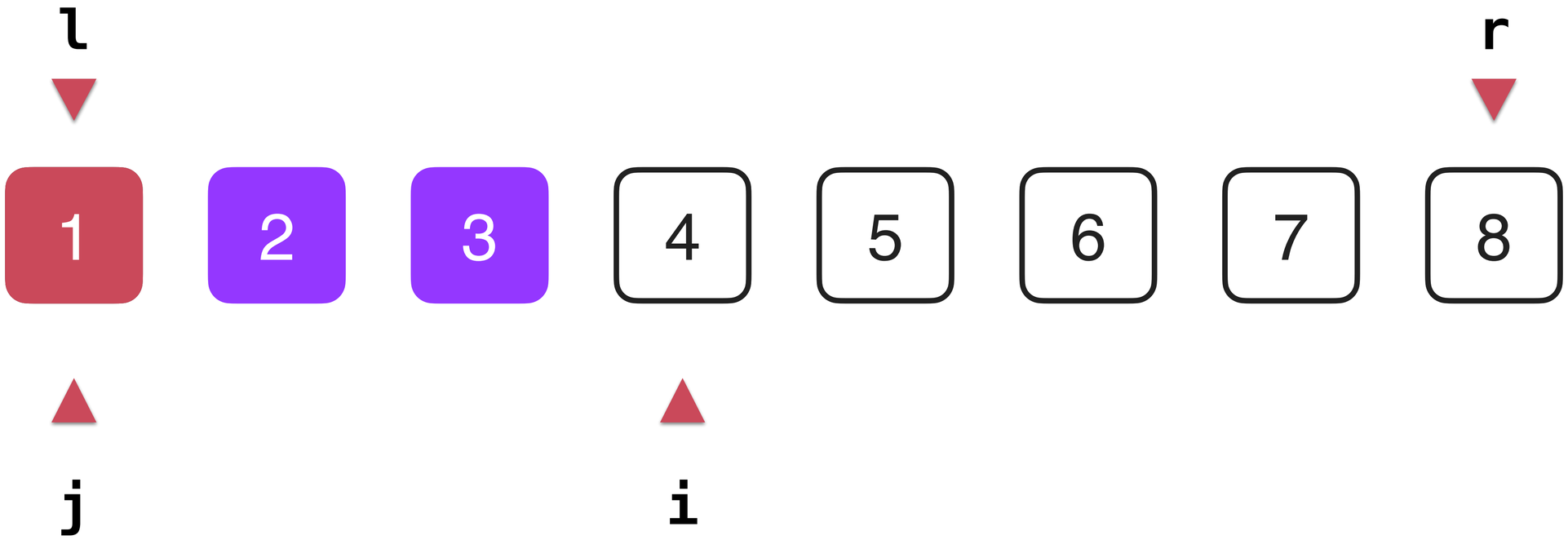
`arr[l+1…j] < v`　　`arr[j+1…i-1] > v`

第一版 QuickSort 的问题



`arr[l+1…j] < v`　　`arr[j+1…i-1] > v`

第一版 QuickSort 的问题

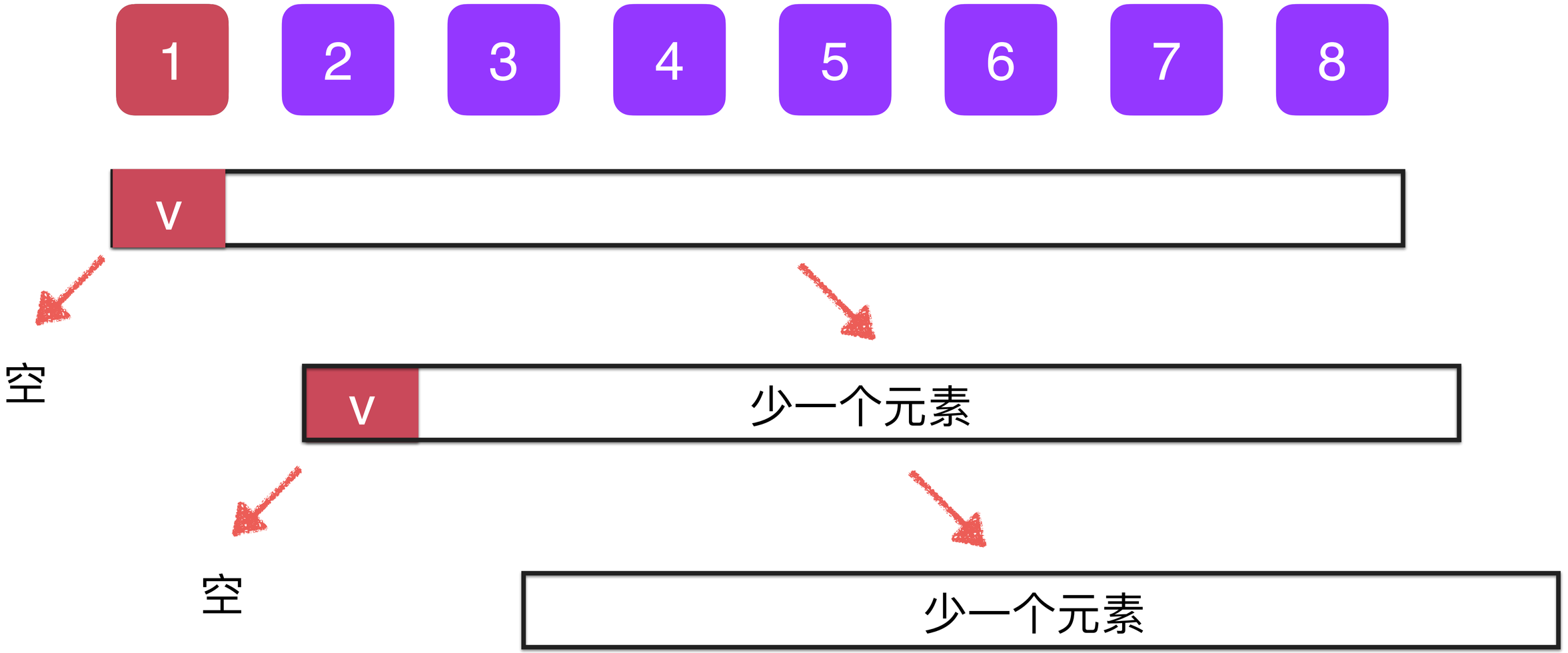


`arr[l+1…j] < v`　　`arr[j+1…i-1] > v`

第一版 QuickSort 的问题

`arr[l+1…j] < v`　　`arr[j+1…i-1] > v`

第一版 QuickSort 的问题

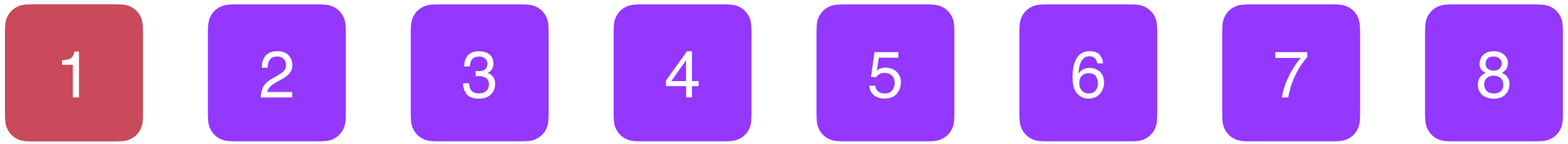


第一版 QuickSort 的问题

时间复杂度：O(n^2)　　递归深度：O(n)

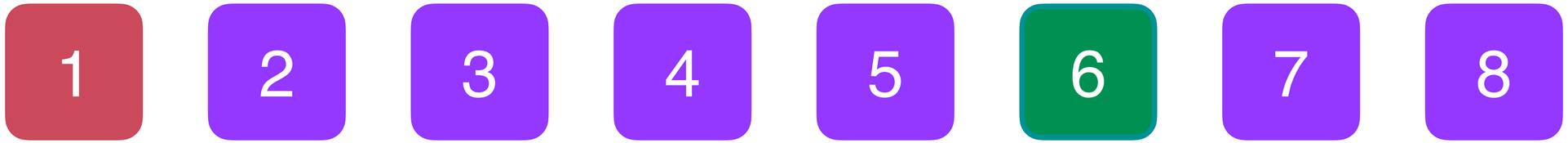
第一版 QuickSort 的问题

如何解决?　　时间复杂度：O(n^2)　　递归深度：O(n)



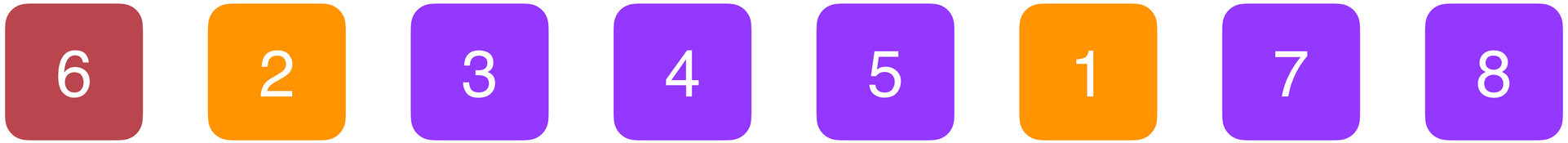
第一版 QuickSort 的问题

如何解决?　　时间复杂度：O(n^2)　　递归深度：O(n)



第一版 QuickSort 的问题

如何解决?　　时间复杂度：O(n^2)　　递归深度：O(n)



更具体的理论分析，后续揭晓

为快速排序添加随机化

为快速排序添加随机化

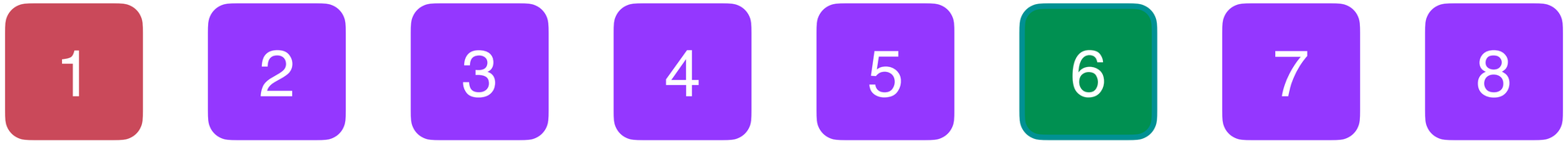
目标：生成一个 [l, r] 区间的随机值

生成一个 [0, r - l] 区间的随机值

l + [0, r - l] 区间的随机值 -> [l, r] 区间的随机值

实践：为快速排序添加随机化

为快速排序添加随机化



每次取中间的值作为标定点?

万一随机每次都取最小值?

$$\frac{1}{n} \times \frac{1}{n-1} \times \frac{1}{n-2} \times \ldots \times \frac{1}{1} = \frac{1}{n!}$$

两个作业：深入玩转快速排序

作业一

```
private static <E extends Comparable<E>> int partition(E[] arr, int l, int r){

    // 生成 [l, r] 之间的随机索引
    int p = l + (new Random()).nextInt(r - l + 1);
    swap(arr, l, p);

    // arr[l+1...j] < v ; arr[j+1...i] >= v
    int j = l;
    for(int i = l + 1; i <= r; i ++)
        if(arr[i].compareTo(arr[l]) < 0){
            j ++;
            swap(arr, i, j);
        }

    swap(arr, l, j);
    return j;
}
```

作业二

```
private static <E extends Comparable<E>> int partition(E[] arr, int l, int r){

    swap(arr, l, (l + r) / 2);

    // arr[l+1...j] < v ; arr[j+1...i] >= v
    int j = l;
    for(int i = l + 1; i <= r; i ++)
        if(arr[i].compareTo(arr[l]) < 0){
            j ++;
            swap(arr, i, j);
        }

    swap(arr, l, j);
    return j;
}
```

ArrayGenerator.generateSpecialArray(n)

其他

欢迎大家关注我的个人公众号：是不是很酷

算法与数据结构体系课程

liuyubobobo