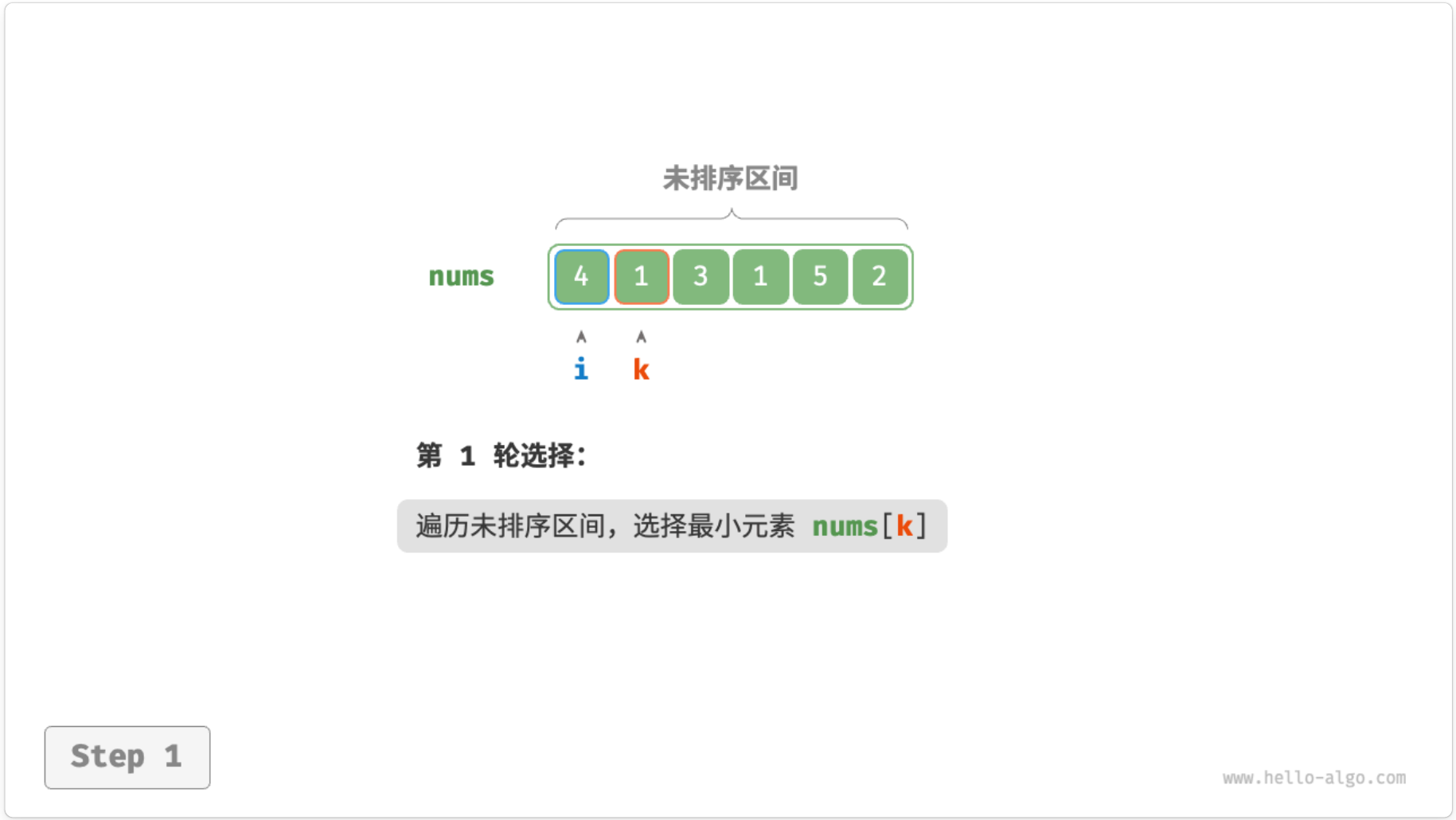
11.2 选择排序

选择排序（selection sort）的工作原理非常简单：开启一个循环，每轮从未排序区间选择最小的元素，将其放到已排序区间的末尾。

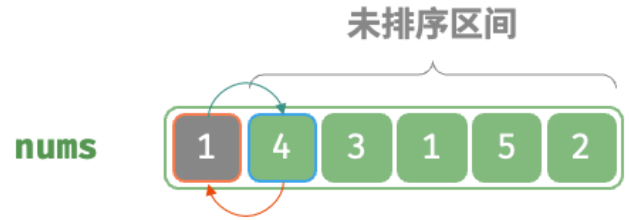
设数组的长度为 n ，选择排序的算法流程如图 11-2 所示。

1. 初始状态下，所有元素未排序，即未排序（索引）区间为 $[0, n-1]$ 。
2. 选取区间 $[0, n-1]$ 中的最小元素，将其与索引 0 处的元素交换。完成后，数组前 1 个元素已排序。
3. 选取区间 $[1, n-1]$ 中的最小元素，将其与索引 1 处的元素交换。完成后，数组前 2 个元素已排序。
4. 以此类推。经过 $n-1$ 轮选择与交换后，数组前 $n-1$ 个元素已排序。
5. 仅剩的一个元素必定是最大元素，无须排序，因此数组排序完成。

<1>


未排序区间
nums
4
1
3
1
5
2
i
k
第 1 轮选择：
遍历未排序区间，选择最小元素 nums[k]
Step 1
www.hello-algo.com

<2>



第 1 轮选择：

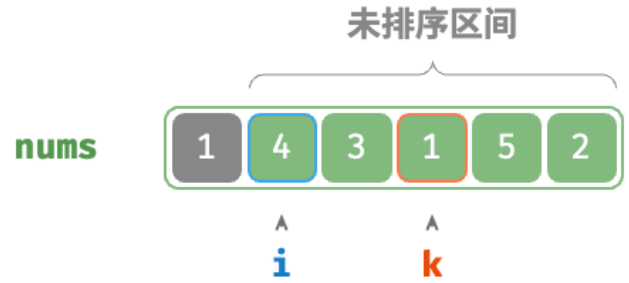
遍历未排序区间，选择最小元素 `nums[k]`

交换最小元素 `nums[k]` 和未排序区间的首个元素 `nums[i]`

Step 2

www.hello-algo.com

<3>



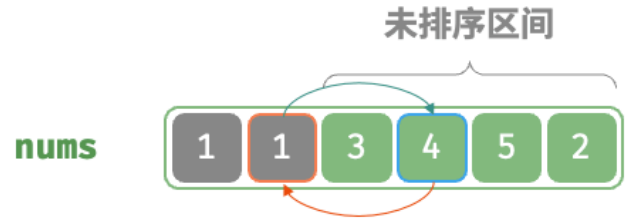
第 2 轮选择：

遍历未排序区间，选择最小元素 `nums[k]`

Step 3

www.hello-algo.com

<4>



第 2 轮选择：

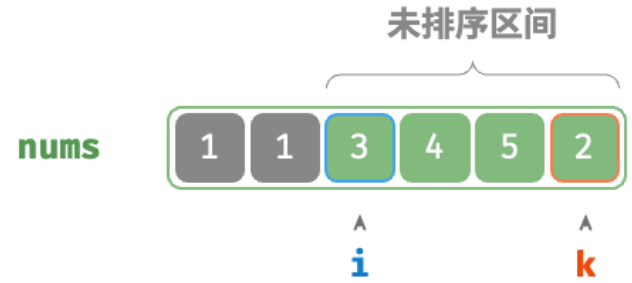
遍历未排序区间，选择最小元素 `nums[k]`

交换最小元素 `nums[k]` 和未排序区间的首个元素 `nums[i]`

Step 4

www.hello-algo.com

<5>



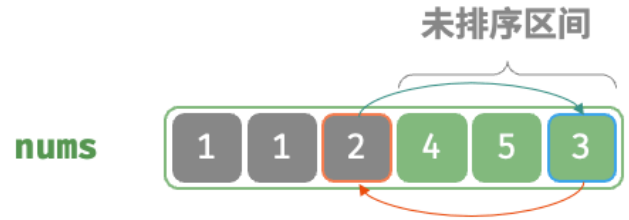
第 3 轮选择：

遍历未排序区间，选择最小元素 `nums[k]`

Step 5

www.hello-algo.com

<6>



第 3 轮选择：

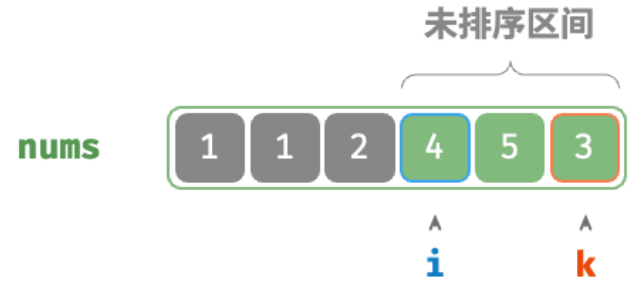
遍历未排序区间，选择最小元素 `nums[k]`

交换最小元素 `nums[k]` 和未排序区间的首个元素 `nums[i]`

Step 6

www.hello-algo.com

<7>



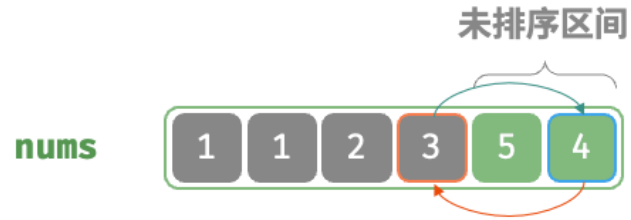
第 4 轮选择：

遍历未排序区间，选择最小元素 `nums[k]`

Step 7

www.hello-algo.com

<8>



第 4 轮选择：

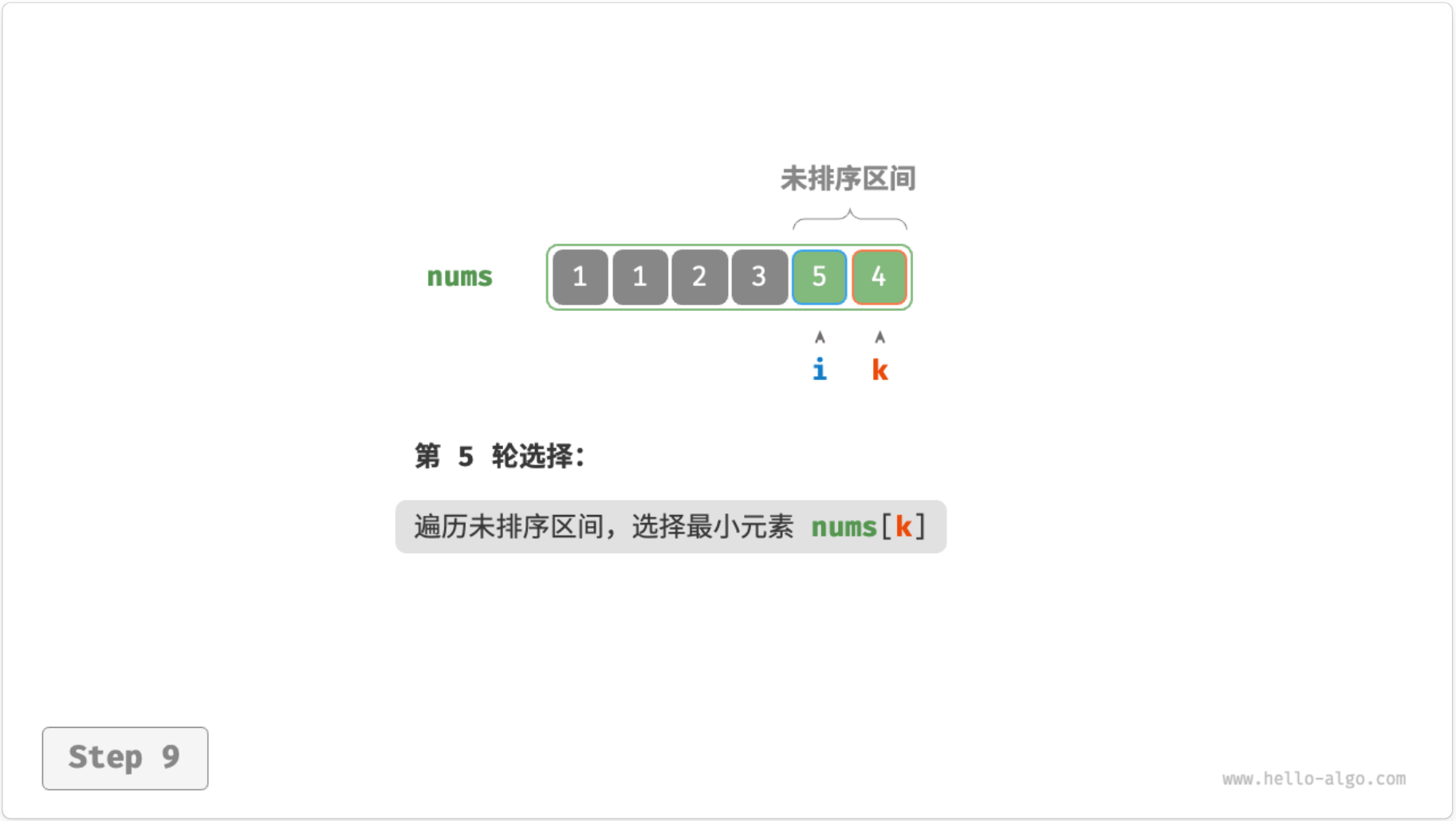
遍历未排序区间，选择最小元素 `nums[k]`

交换最小元素 `nums[k]` 和未排序区间的首个元素 `nums[i]`

Step 8

www.hello-algo.com

<9>


未排序区间
nums
1
1
2
3
5
4
i
k
第 5 轮选择：
遍历未排序区间，选择最小元素 nums[k]
Step 9
www.hello-algo.com

<10>


未排序区间
nums
1
1
2
3
4
5
第 5 轮选择：
遍历未排序区间，选择最小元素 nums[k]
交换最小元素 nums[k] 和未排序区间的首个元素 nums[i]
Step 10
www.hello-algo.com

<11>

图 11-2 选择排序步骤

在代码中，我们用 k 来记录未排序区间内的最小元素：

Python

selection_sort.py

```python
def selection_sort(nums: list[int]):
    """选择排序"""
    n = len(nums)
    # 外循环：未排序区间为 [i, n-1]
    for i in range(n - 1):
        # 内循环：找到未排序区间内的最小元素
        k = i
        for j in range(i + 1, n):
            if nums[j] < nums[k]:
                k = j  # 记录最小元素的索引
        # 将该最小元素与未排序区间的首个元素交换
        nums[i], nums[k] = nums[k], nums[i]
```

C++

selection_sort.cpp

```cpp
/* 选择排序 */
void selectionSort(vector<int> &nums) {
    int n = nums.size();
    // 外循环：未排序区间为 [i, n-1]
    for (int i = 0; i < n - 1; i++) {
        // 内循环：找到未排序区间内的最小元素
        int k = i;
        for (int j = i + 1; j < n; j++) {
            if (nums[j] < nums[k])
                k = j; // 记录最小元素的索引
        }
        // 将该最小元素与未排序区间的首个元素交换
        swap(nums[i], nums[k]);
    }
}
```

Java

selection_sort.java

- **时间复杂度为 $O(n^2)$、非自适应排序**：外循环共 $n - 1$ 轮，第一轮的未排序区间长度为 n ，最后一轮的未排序区间长度为 2 ，即各轮外循环分别包含 n、$n - 1$ 、$\dots$、3、2 轮内循环，求和为 $\frac{(n-1)(n+2)}{2}$ 。
- **空间复杂度为 $O(1)$、原地排序**：指针 i 和 j 使用常数大小的额外空间。
- **非稳定排序**：如图 11-3 所示，元素 `nums[i]` 有可能被交换至与其相等的元素的右边，导致两者的相对顺序发生改变。

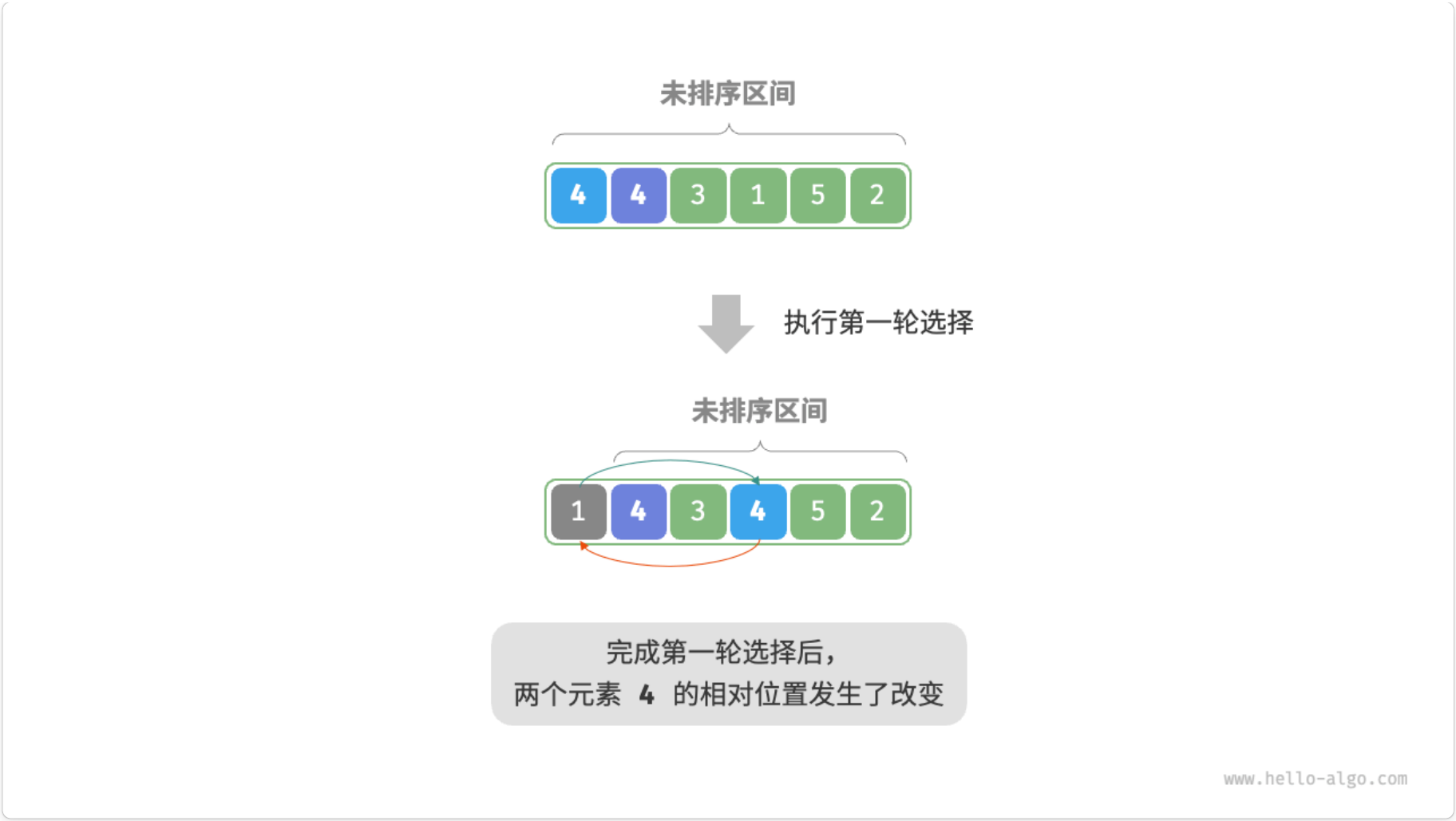


图 11-3 选择排序非稳定示例

上一页 下一页