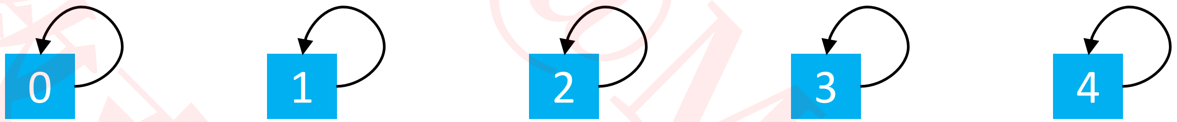
Quick Union – Union

- Quick Union 的 union(v1, v2)：让 v1 的根节点指向 v2 的根节点

0	1	2	3	4
0	1	2	3	4

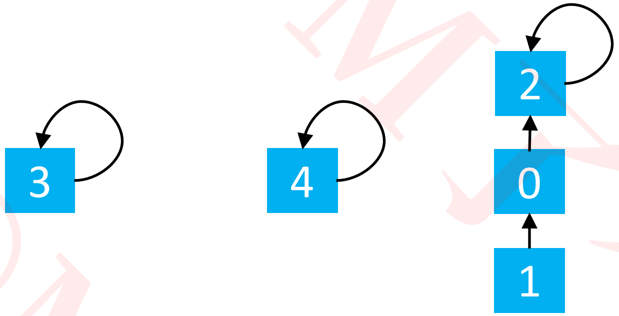


union(1, 0)

0	1	2	3	4
0	0	2	3	4

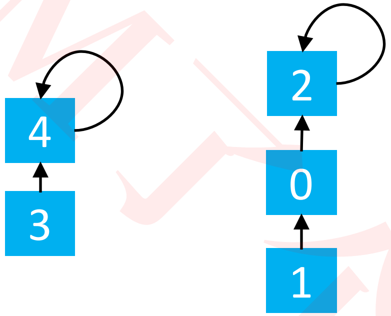
union(1, 2)

0	1	2	3	4
2	0	2	3	4



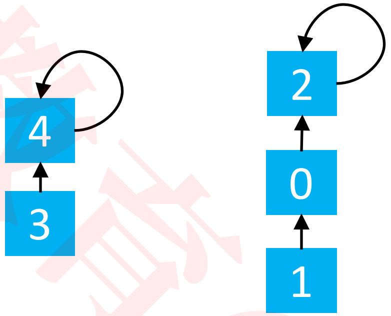
union(3, 4)

0	1	2	3	4
2	0	2	4	4



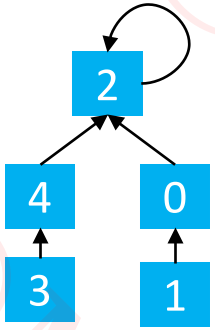
Quick Union – Union

0	1	2	3	4
2	0	2	4	4



union(3, 1)

0	1	2	3	4
2	0	2	4	2



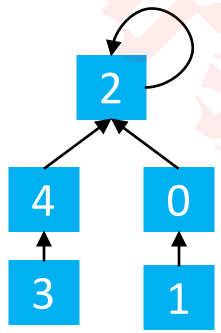
Quick Union – Union

```
public void union(int v1, int v2) {
    int p1 = find(v1);
    int p2 = find(v2);
    if (p1 == p2) return;

    parents[p1] = p2;
}
```

- 时间复杂度：O(logn)

Quick Union – Find

0	1	2	3	4
2	0	2	4	2

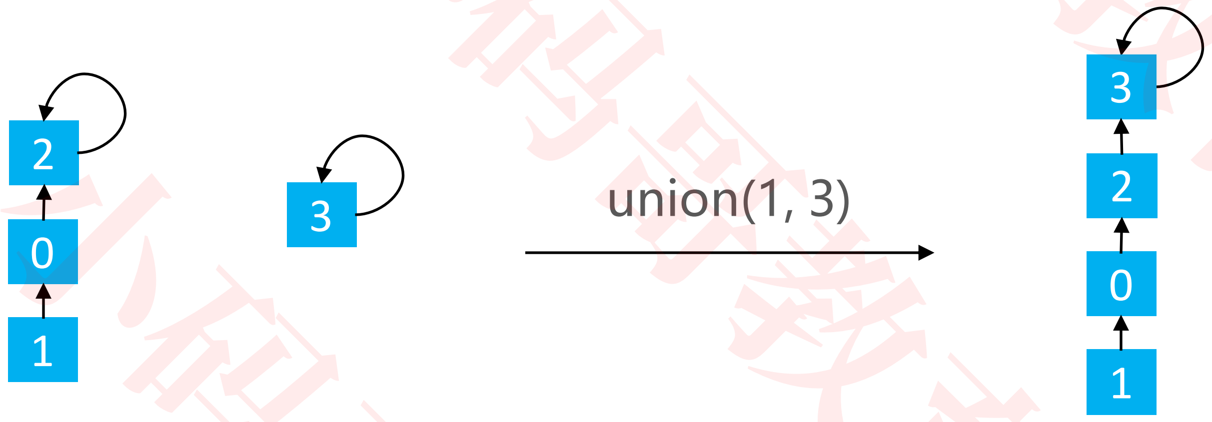



```java
public int find(int v) {
    rangeCheck(v);
    while (v != parents[v]) {
        v = parents[v];
    }
    return v;
}
```

- find(0) == 2
- find(1) == 2
- find(3) == 2
- find(2) == 2
- 时间复杂度：O(logn)

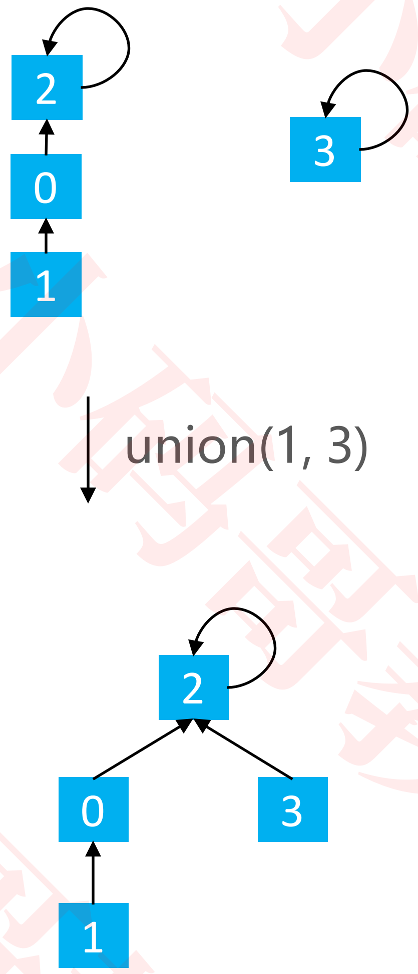
Quick Union – 优化

- 在Union的过程中，可能会出现树不平衡的情况，甚至退化成链表



- 有2种常见的优化方案
- 基于size的优化：元素少的树 嫁接到 元素多的树
- 基于rank的优化：矮的树 嫁接到 高的树

Quick Union – 基于size的优化

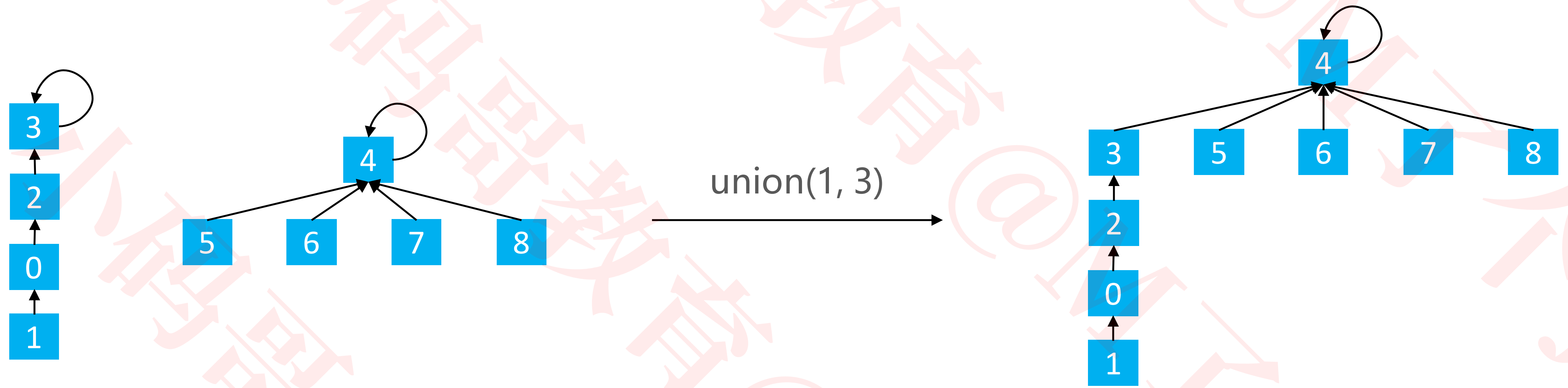



```java
sizes = new int[capacity];
for (int i = 0; i < sizes.length; i++) {
    sizes[i] = 1;
}
```

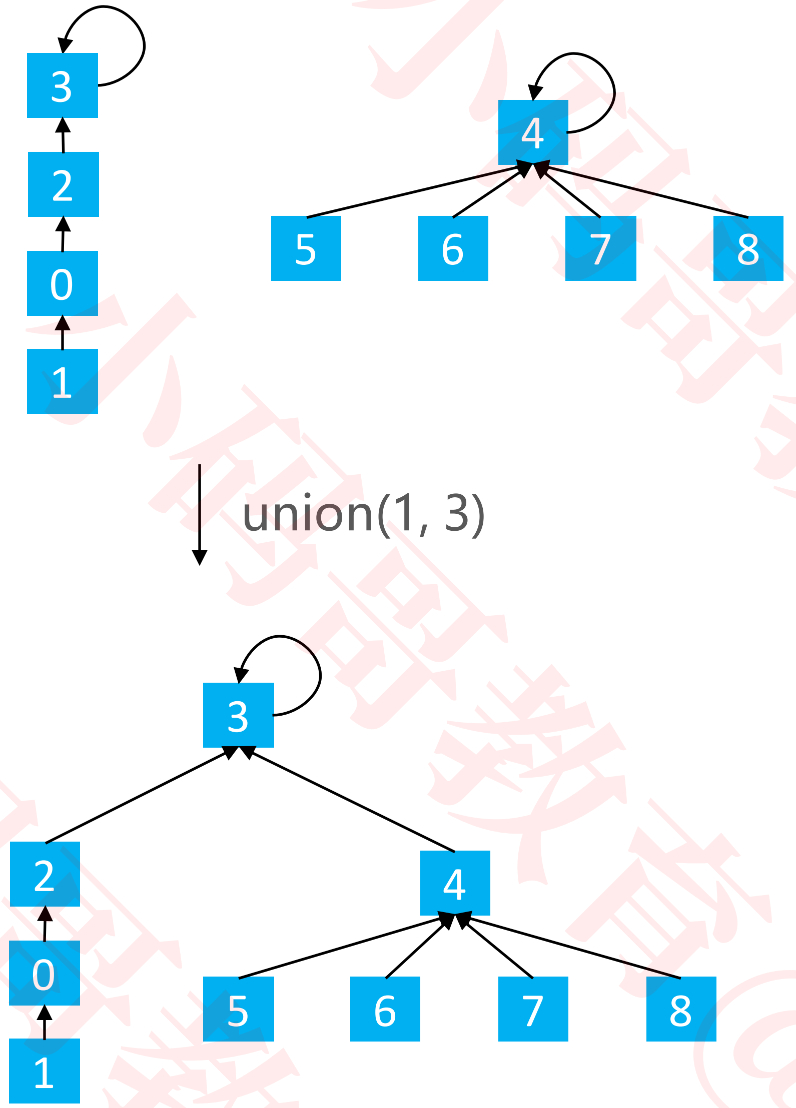
```java
private int[] sizes;
public void union(int v1, int v2) {
    int p1 = find(v1);
    int p2 = find(v2);
    if (p1 == p2) return;

    if (sizes[p1] < sizes[p2]) {
        parents[p1] = p2;
        sizes[p2] += sizes[p1];
    } else {
        parents[p2] = p1;
        sizes[p1] += sizes[p2];
    }
}
```

Quick Union – 基于size的优化

■ 基于size的优化，也可能会存在树不平衡的问题



Quick Union – 基于rank的优化




```java
ranks = new int[capacity];
for (int i = 0; i < ranks.length; i++) {
    ranks[i] = 1;
}
```

```java
private int[] ranks;
public void union(int v1, int v2) {
    int p1 = find(v1);
    int p2 = find(v2);
    if (p1 == p2) return;

    if (ranks[p1] < ranks[p2]) {
        parents[p1] = p2;
    } else if (ranks[p2] < ranks[p1]) {
        parents[p2] = p1;
    } else {
        parents[p1] = p2;
        ranks[p2]++;
    }
}
```