可持久化线段树

主讲人：邓哲也

可持久化线段树的引入

问题：有一个长度为 n 的序列，a[1]，a[2]，…，a[n]。

现在执行 m 次操作，每次可以执行以下两种操作之一：

1 i v：将数列中的某个数 a[i] 修改为 v。

2 k l r：　　询问第 k 次操作后，一个下标区间 [l，r] 中所有数的和。

样例：	输出：
4	1234
1000 200 30 4	1254
5	1654
1 3 50	
1 2 600	
2 0 1 4	
2 1 1 4	
2 2 1 4	

可持久化线段树的引入

和以往不同，现在询问是对某一次的操作后的序列做询问。

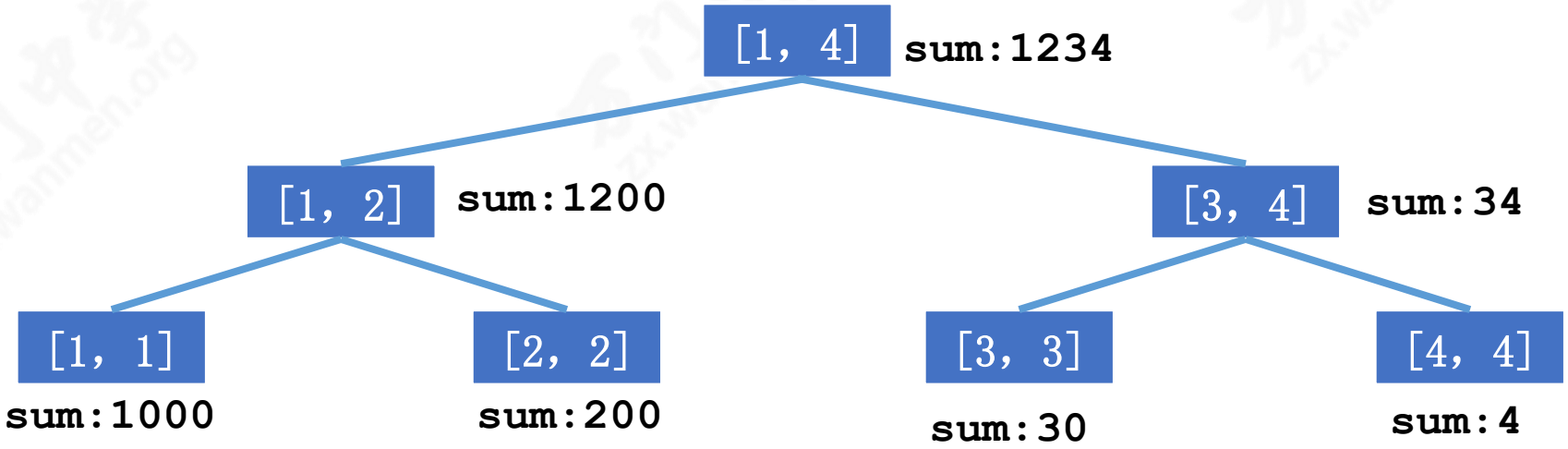
如果用线段树维护区间和，那我们需要存下每次修改完后的线段树。

一个线段树需要 O(n) 的空间，显然不能存下 n 颗线段树。

但是我们可以从这里找找突破口。

线段树单点修改

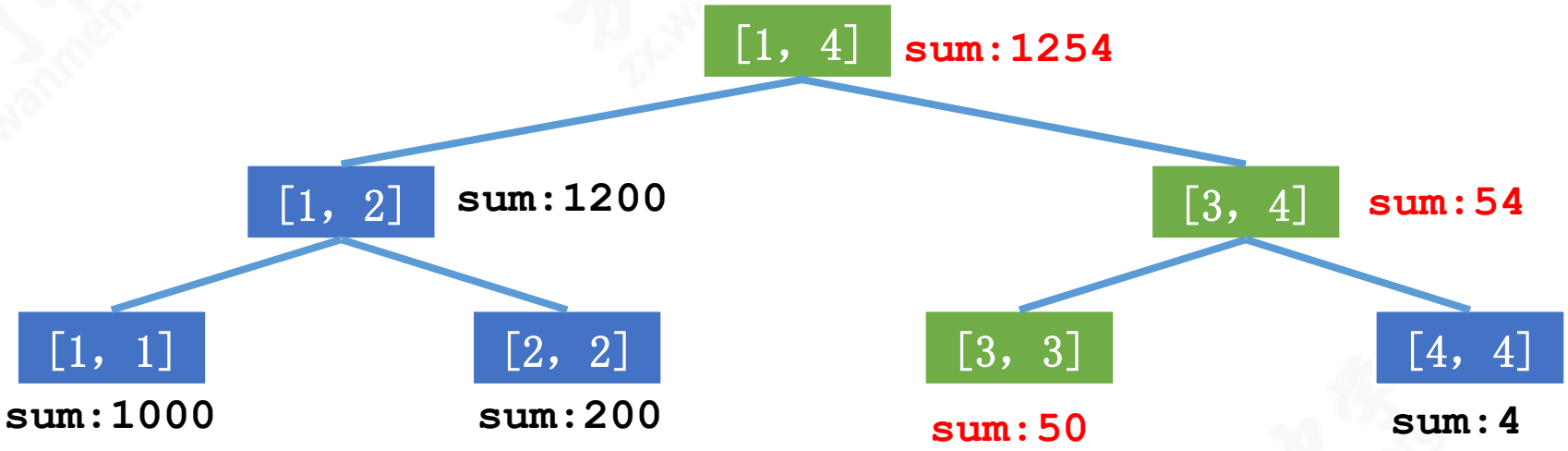
来看这颗线段树的初始状态，每个节点记录了区间的 sum。



线段树单点修改

第一次操作：把第 3 个位置改为 50

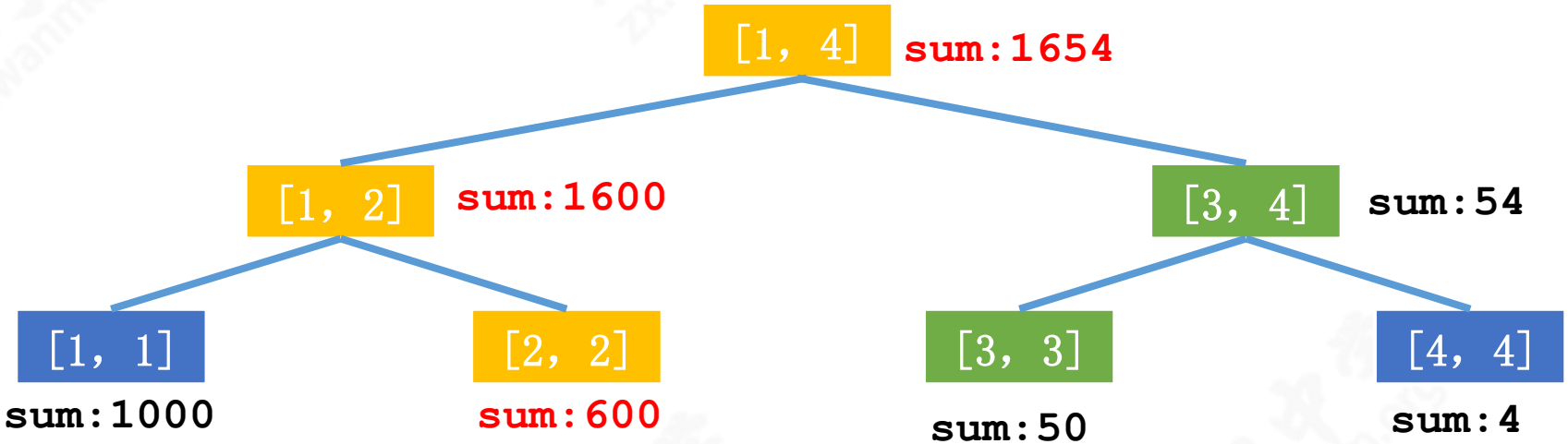
可以发现的是，这颗线段树和上一颗线段树只有 log n 个节点发生了变化。



线段树单点修改

第二次操作：把第 2 个位置改为 600

可以发现的是，这颗线段树和上一颗线段树同样也只有 log n 个节点发生了变化。



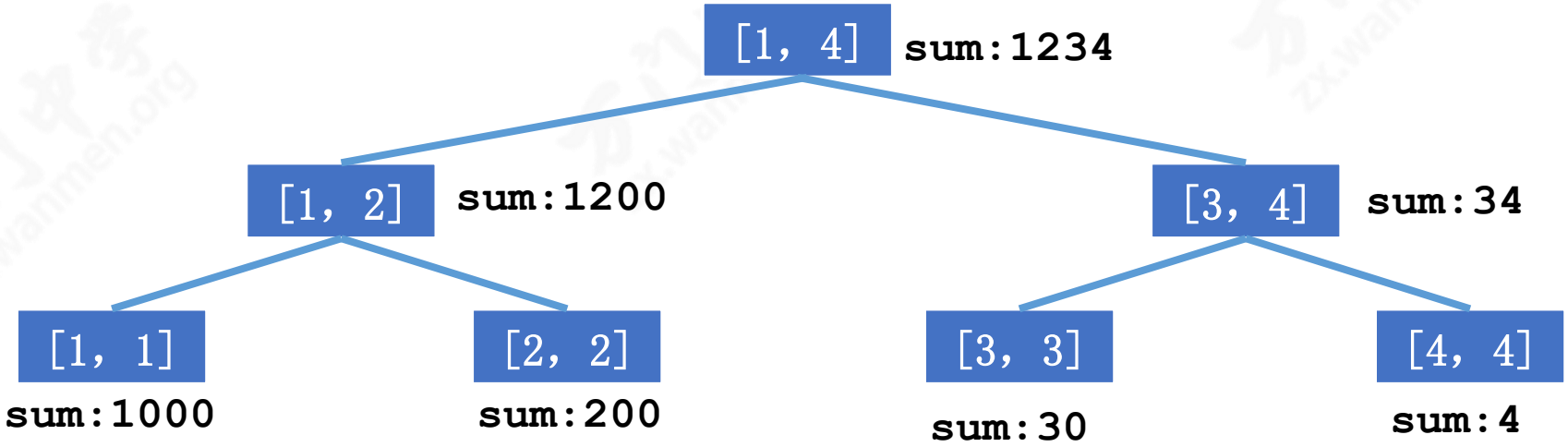
可持久化线段树的引入

我们发现，每一次单点修改，发生改变的节点只有 log n 个，就是从根节点到那个点对应的叶节点的路径上的所有节点。

那么对于每次修改，我们只要存下来这新的 log n 个节点，尽可能的利用上一颗树的信息，就可以存下一颗新的线段树！

可持久化线段树单点修改

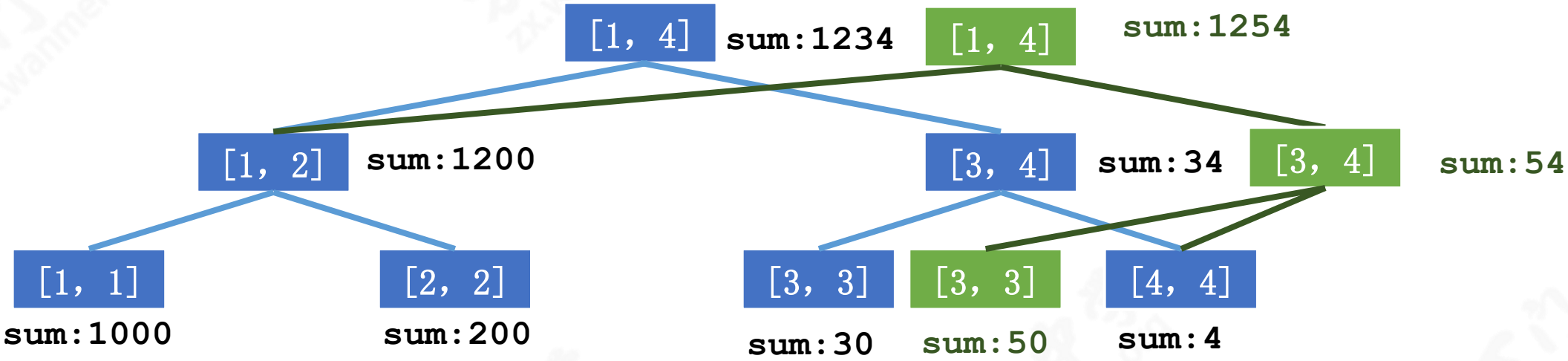
来看这颗线段树的初始状态，每个节点记录了区间的 sum。



可持久化线段树单点修改

第一次操作：把第 3 个位置改为 50。

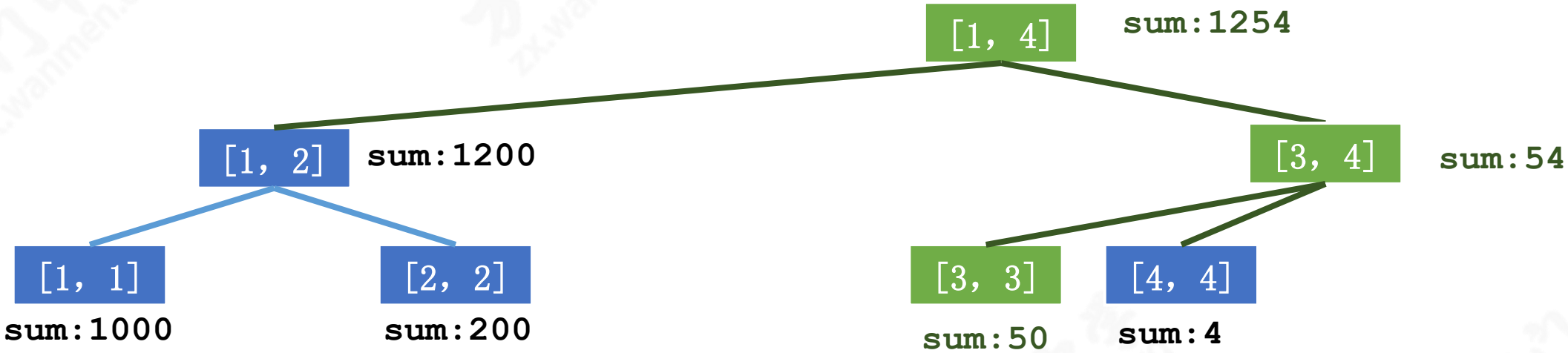
我们只要新建 log n 个新点，并且尽可能的利用上一颗树的信息。



可持久化线段树单点修改

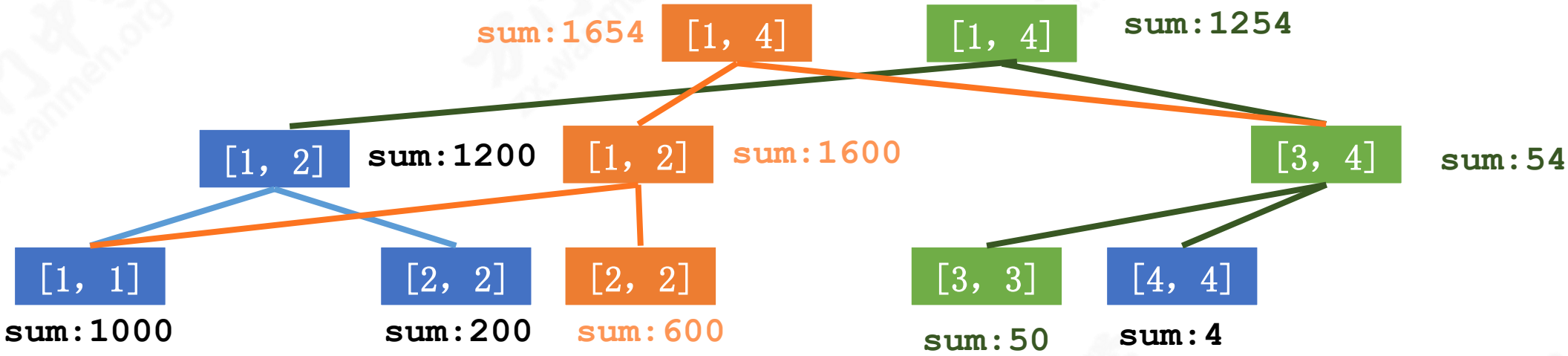
这就是第一次操作后的线段树。

绿色的是新节点，蓝色的是沿用上一颗树中的节点。



可持久化线段树单点修改

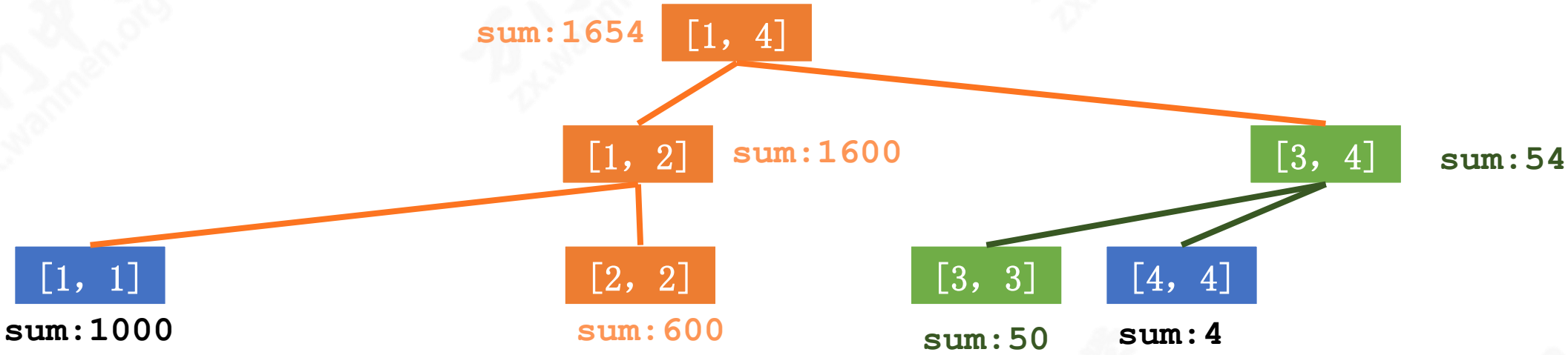
第二次操作：把第 2 个位置改为 600



可持久化线段树单点修改

这就是第二次操作后的线段树。

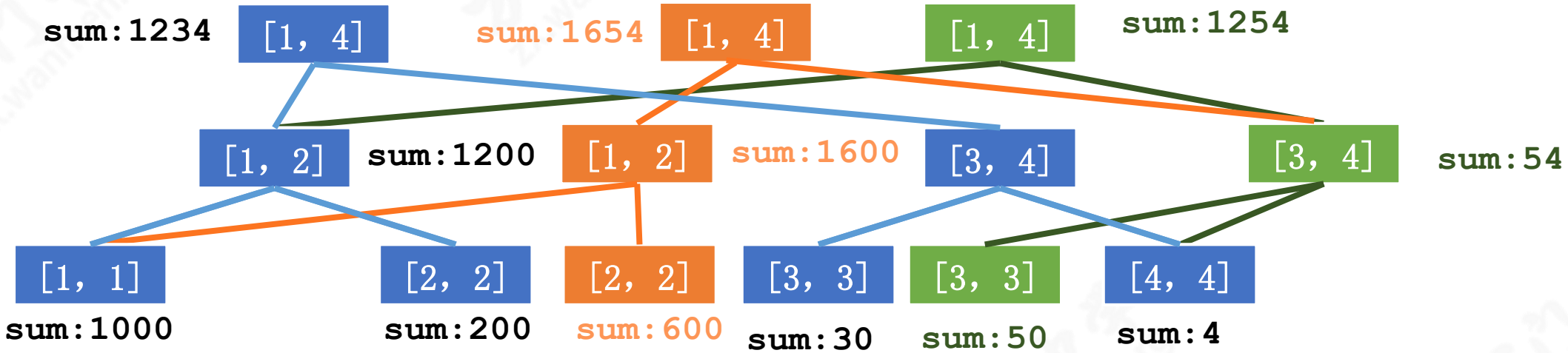
橘黄色的点是新节点，其他的都是上一颗树中的节点。



可持久化线段树单点修改

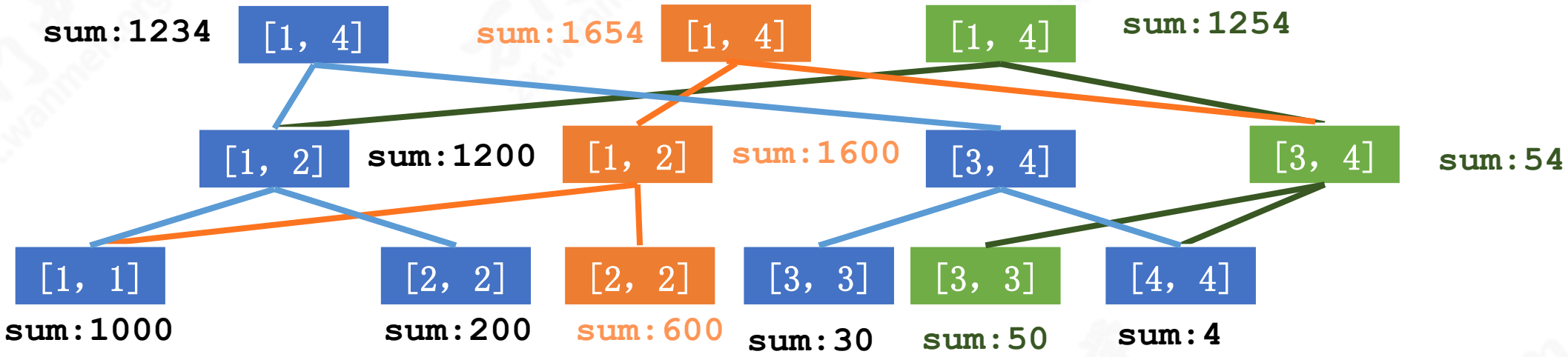
把三棵树都列出来：

0 - 蓝色， 1 - 绿色， 2 - 橘黄色。



可持久化线段树单点修改

记录下每一颗树的根节点，就可以直接在对应的树上做查询了。



可持久化线段树单点修改

由这个例子我们可以发现，每次修改都会多 O(log n) 个点。

n 次操作后，n 颗线段树的空间复杂度就是 O(n log n)！

实现的时候，不能再用 2x 和 2x+1 来表示子节点了。

因为每次都要新加点，所以我们只要在每次新开一个节点的时候，给它一个全新的标号就行了。

可持久化线段树实现

我们用 lc[x]， rc[x] 来表示 x 的左孩子和右孩子。

```
void update(int x){
        sum[x] = sum[lc[x]] + sum[rc[x]];
}
```

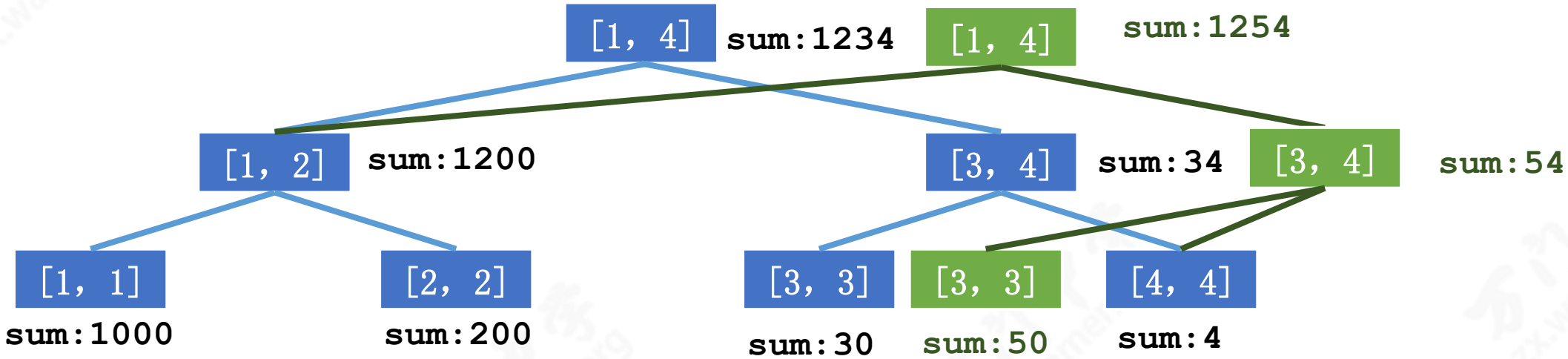
可持久化线段树实现

```
void build(int l, int r, int &x){
    if(!x) x = ++ tot;
    if (l == r){
        sum[x] = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, lc[x]);
    build(mid + 1, r, rc[x]);
    update(x);
}
```

初始化：build(1, n, root[0]);

可持久化线段树单点修改

单点修改的时候，我们需要新建一个树根，同时有一个指针指向上一个树根，一起向下移动，决定每一层复制左节点还是右节点，如图所示，修改[3，3]。



可持久化线段树实现

```
void modify(int p, int v, int l, int r, int &x, int last){
    if(!x) x = ++ tot;
    if (l == r){
        sum[x] = v;
        return;
    }
    int mid = (l + r) >> 1;
    if (p <= mid){
        rc[x] = rc[last];
        modify(p, v, l, mid, lc[x], lc[last]);
    }else{
        lc[x] = lc[last];
        modify(p, v, mid + 1, r, rc[x], rc[last]);
    }
    update(x);
}
```

可持久化线段树实现

```
int query(int A, int B, int l, int r, int x){
    if (!x) return 0;
    if (A <= l && r <= B) return sum[x];
    int mid = (l + r) >> 1, ret = 0;
    if (A <= mid) ret += query(A, B, l, mid, lc[x]);
    if (mid < B)  ret += query(A, B, mid + 1, r, rc[x]);
    return ret;
}
```

可持久化线段树实现

```
int main(){
    scanf("%d", &n);
    for(int i = 1;i <= n;i ++) scanf("%d", &a[i]);
    build(1, n, root[0]);
    scanf("%d", &Q);
    int op, k, x, y;
    for(int i = 1;i <= Q;i ++){
        scanf("%d", &op);
        if (op == 1){
            scanf("%d%d", &x, &y);
            modify(x, y, 1, n, root[i], root[i - 1]);
        } else {
            scanf("%d%d%d", &k, &x, &y);
            printf("%d\n", query(x, y, 1, n, root[k]));
        }
    }
}
```

下节课再见