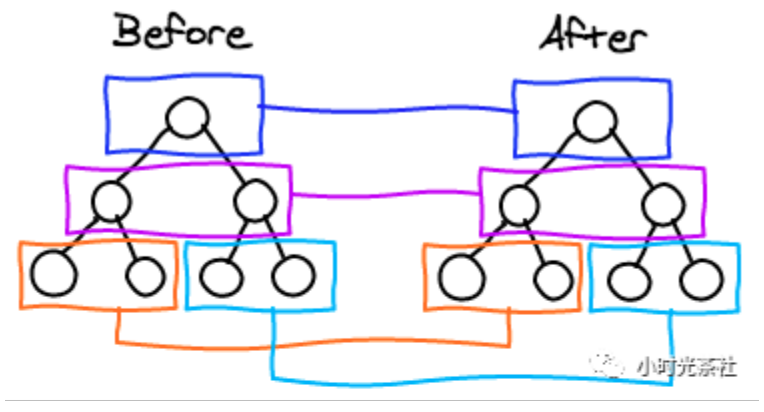


为了实现高效的DOM操作，一套高效的虚拟DOM diff算法显得很有必要。

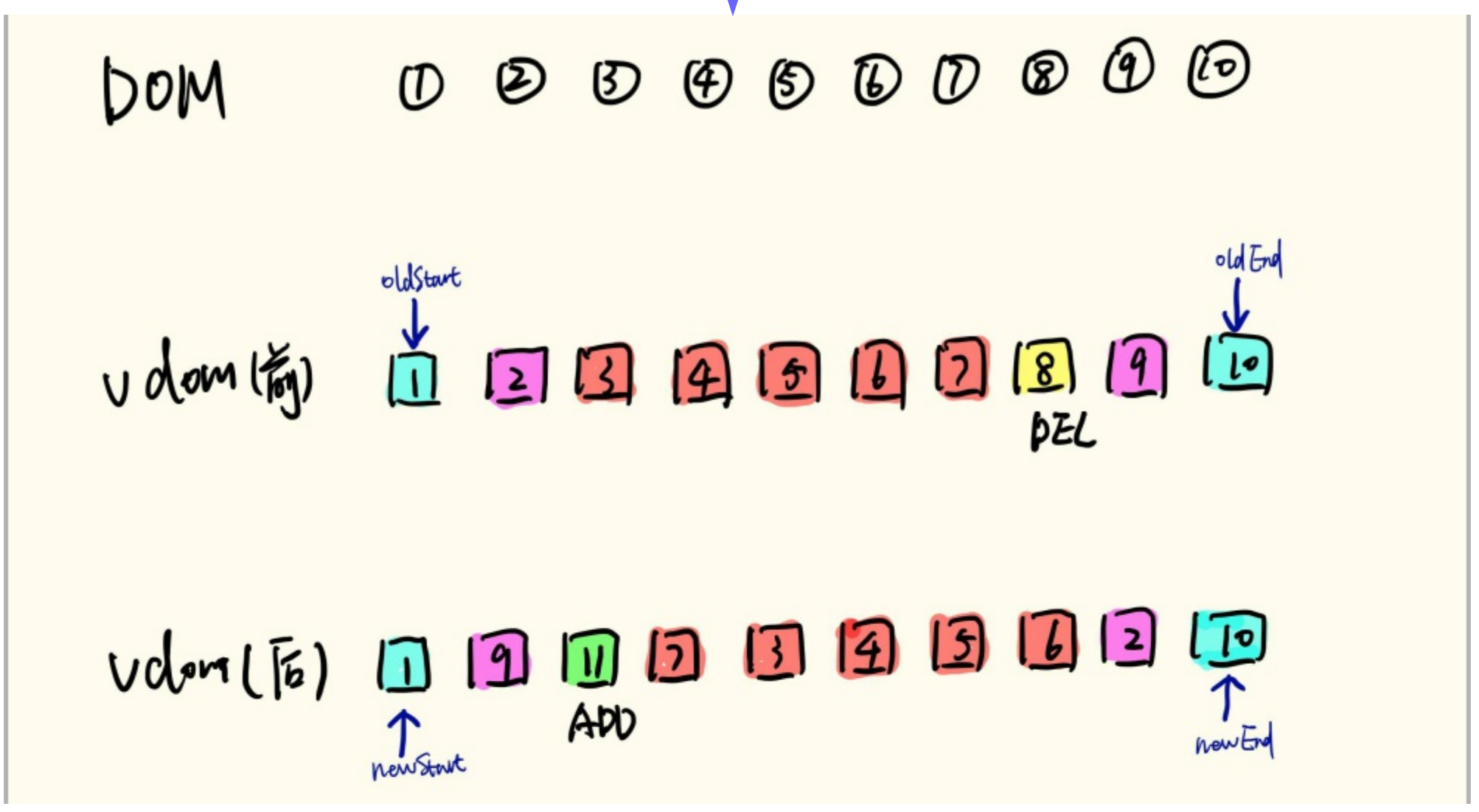
Vue的diff算法是基于snabbdom改造过来的

这是一张很经典的图，出自《React’ s diff algorithm》，Vue的diff算法也同样，

即仅在同级的vnode间做diff，递归地进行同级vnode的diff，最终实现整个DOM树的更新。

那同级vnode diff的细节又是怎样的呢？正是本文所要讲的。

下面先用一个简单的方法来完成diff



第一次比较: 头部相同、尾部相同的节点：如1、10 , 不用动,

新 DOM : 1 2 3 4 5 6 7 8 9 10

old vdom 标记 1 和 10 为已处理 new vdom也标记为已经处理

第二次比较: 头尾相同的节点：如2、9（处理完头部相同、尾部相同节点之后）

new 中 9 的位置是 => 2

在old中 找到9这个值的位置 => 9

在真实 DOM中 把第9个元素直接移动到第二个元素的前面

新 DOM : 1 9 3 4 5 6 7 8 2 10

old vdom 标记 2 和 9 为已处理

第三次比较: 新增的节点：11, 并放在dom列表第三个元素前面

新 DOM : 1 9 11 3 4 5 6 7 8 2 10 ;

第四次比较: 找到7, 把它放入dom中第4个位置

新 DOM : 1 9 11 7 3 4 5 6 8 2 10

newvdom: 1 9 11 7 3 4 5 6 2 10

标记 oldvdom的 7 为已处理

第五次比较:

3 4 5 6 位置一样, 节点也一样, 标记为已处理, 此时 newvdom已处理完

新 DOM : 1 9 11 7 3 4 5 6 8 2 10

newvdom: 1 9 11 7 3 4 5 6 2 10

在 oldvdom中 最后剩下 8 没有被标记为已处理, 删除 dom中的 8

vue也采用了类似的标记处理, 但vue做了优化

1.同类型的的节点不做移动, 原地复用, 只更新其内容

2. 定义两对指针, 逐渐缩小范围

Vue不断对vnode进行处理同时移动指针

直到其中任意一对起点和终点相遇。

处理过的节点Vue会在oldVdom和newVdom中同时将它标记为已处理