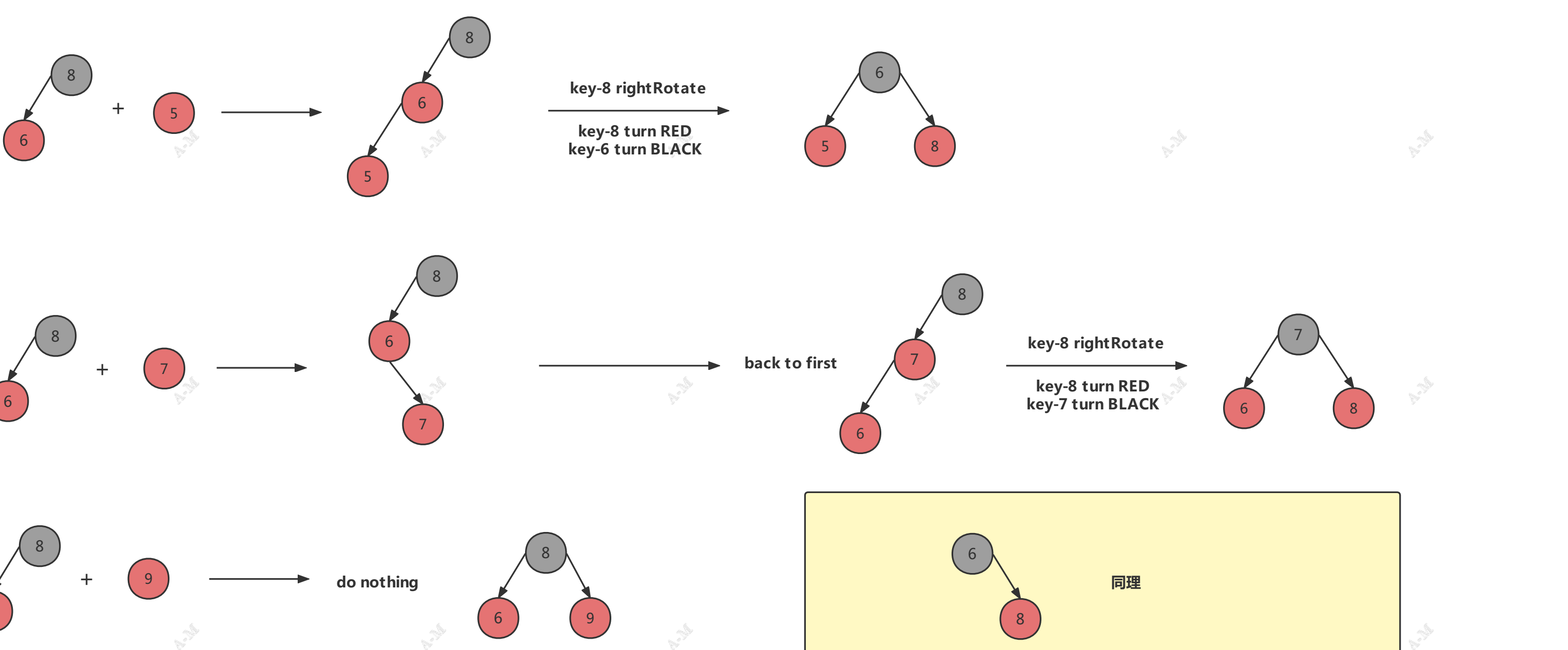


RED-BLACK Tree put 原则：先 put，再调整

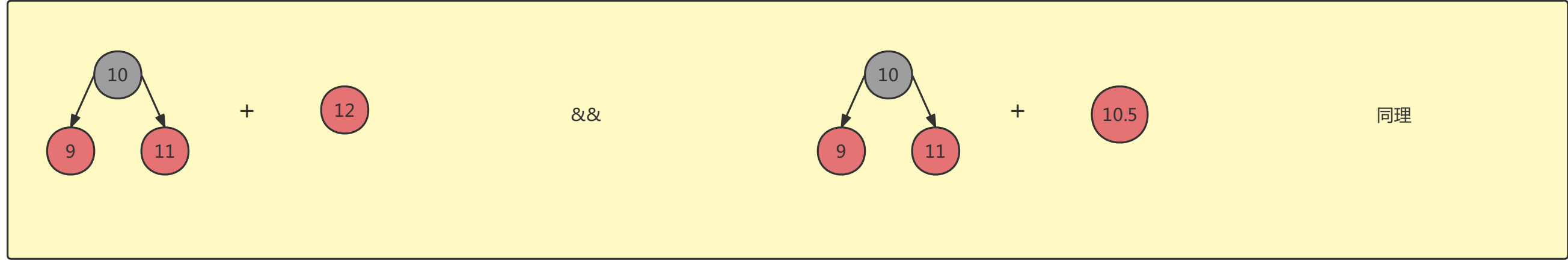
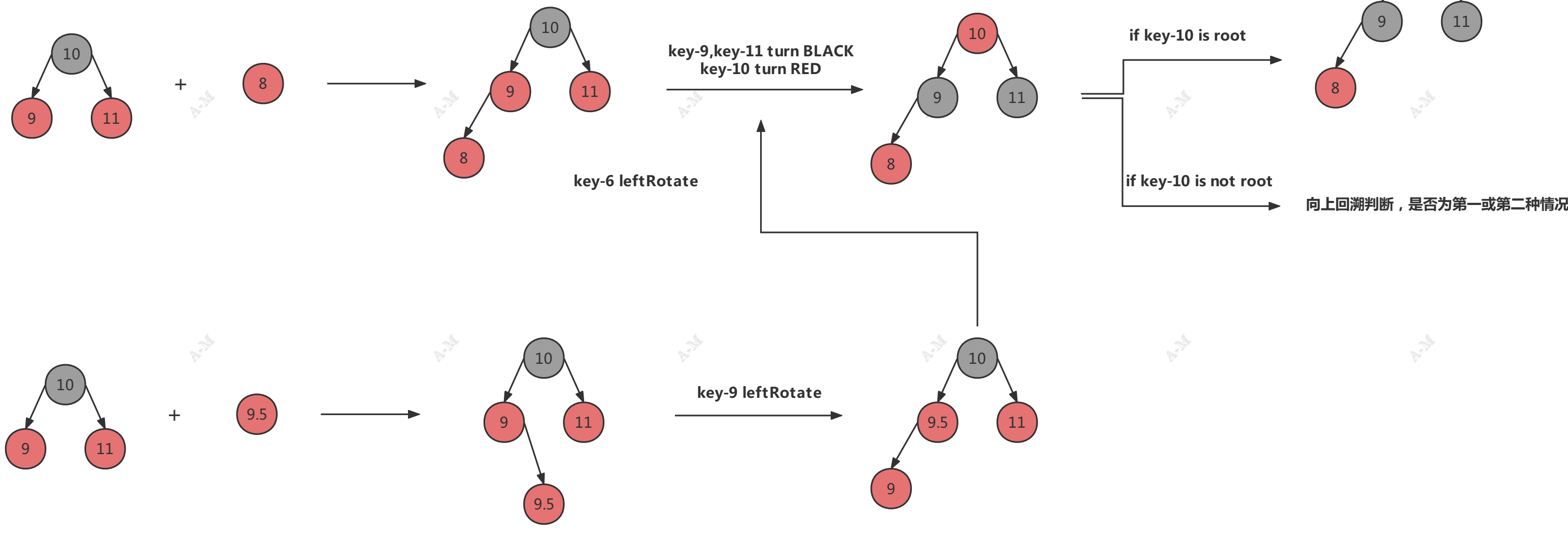
2-Node



3-Node



4-Node



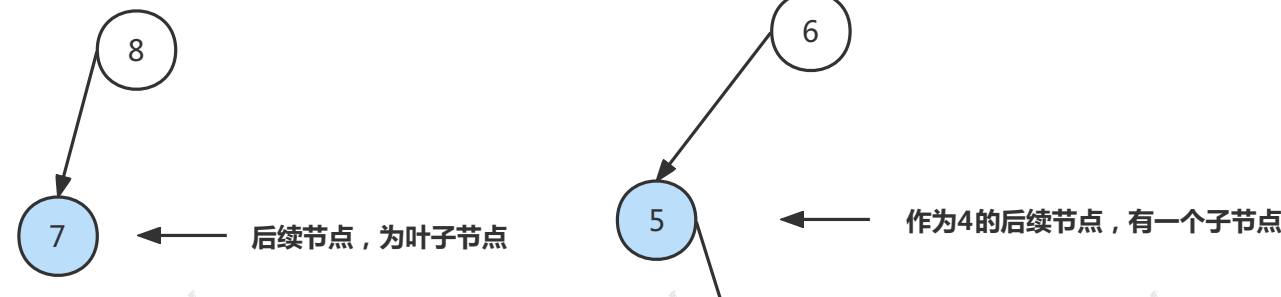
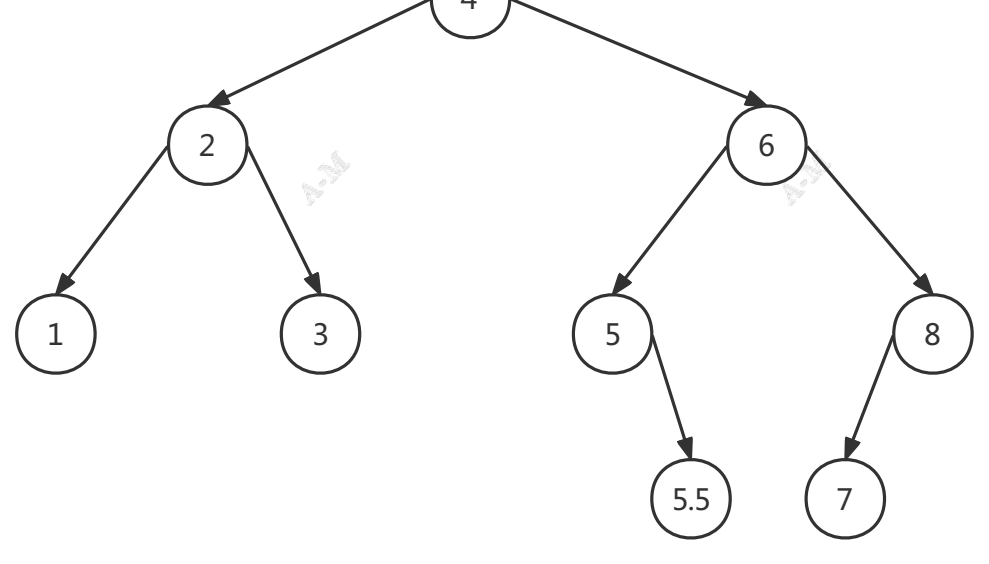
RED-BLACK Tree remove 原则：先 remove，再调整

序

二叉树删除：寻找前驱和后继节点顶替（即覆盖指定删除的节点的值，真正删除的是前驱或后继节点）

删除有三种情况：

- 指定删除的节点为叶子节点，直接删除
  - 指定删除的节点只有一个子节点，用子节点代替，并删除子节点
  - 指定删除的节点有两个子节点，寻找前驱或后继节点代替，并删除前驱或后继节点
- 以后继节点为例，后继节点有且只有两种形式，如下所示：



首先必须明白哪些情况再删除后进行调整

故删除节点的问题，红黑树与二叉树一致，最终只会成为删除一个叶子节点或者删除一个只有一个子节点的节点的问题。  
所以对应 2-3-4 树来讲，真正删除的永远是最底层的节点

1. 找到指定删除的节点和真正要被删除的节点，用真正要被删除的节点的值覆盖指定删除的节点

在此，假设我们选取后继节点作为真正要被删除的节点，这是以下讨论的基本条件！！  
若无后继节点，则另讨论

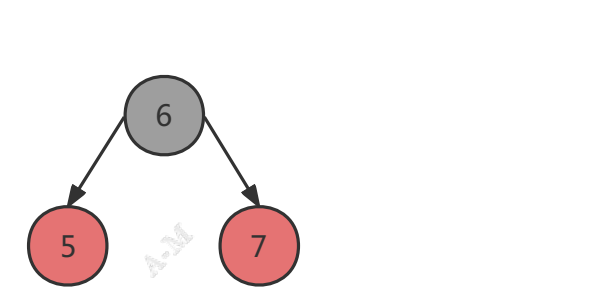
2. 删除并调整

首先必须明白哪些情况再删除后进行调整

1 若删除的是 4-Node 中的一个



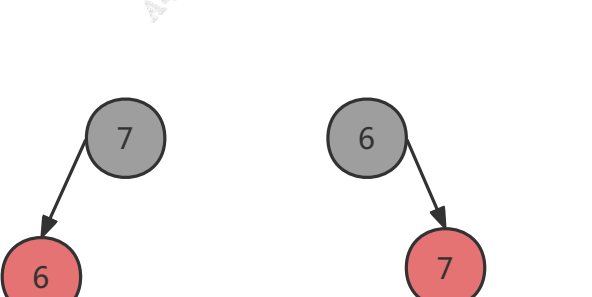
结合上述，这种情况下，被删除的只能是 5 或 7，那么这种情况删除的是红色叶子节点，不影响黑高，直接删除即可



2 若删除的是 3-Node 中的一个



这种情况下，被删除的只能是 6：  
左边情况：6 是红色的叶子节点，直接删除即可，同上面 4-Node 情况  
右边情况：6 是带有子节点的黑色节点，用其子节点覆盖它，并将其染成黑色



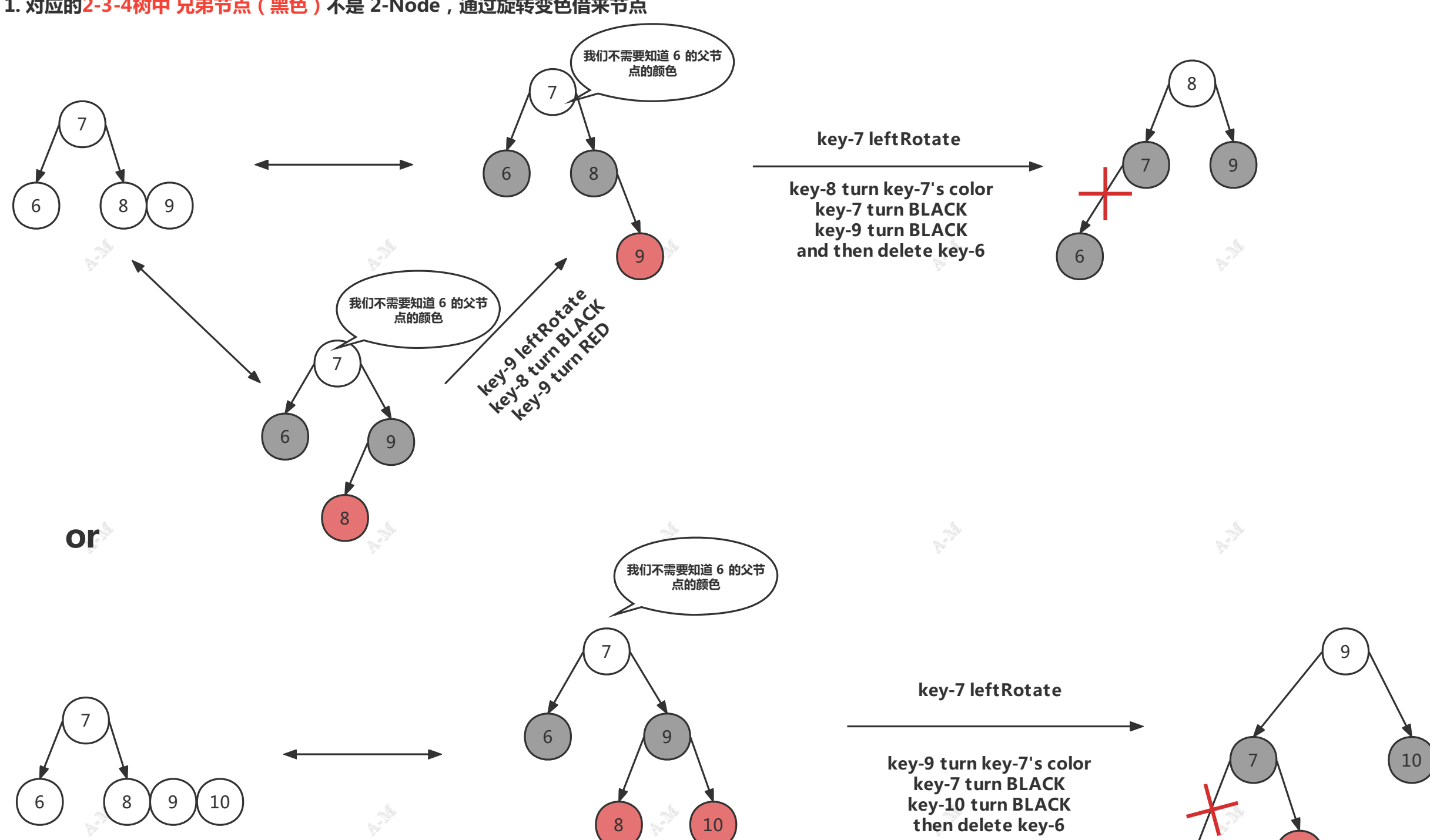
3 若删除的是 2-Node

排除以上并不复杂两种情况，删除后真正需要调整的只有删除一个 2-Node 的情况

被删除的节点必然是黑色叶子节点。若删除的目标是 6，且该节点是父节点的左节点：分以下几种情况

在红黑树对应的 2-3-4 树中，兄弟节点必然是黑色，如果在红黑树兄弟节点不为红，要通过旋转来将其兄弟节点成为黑色节点

1. 对应的 2-3-4 树中 兄弟节点（黑色）不是 2-Node，通过旋转变色借来节点



2. 对应的 2-3-4 树中 兄弟节点是 2-Node



这种情况，删除 6 节点，会导致黑高条件被打破，我们采取的措施是：将它的兄弟节点变为红色，向上回溯：  
1) 如果父节点为根节点，退出回溯  
2) 如果父节点不为根节点，将父节点染黑，如果父节点的兄弟节点为黑节点，将其染红。  
3) 当前节点赋值父节点，向上回溯 ——> 只能是 情况 1 和 情况 2

以下为该情况的演示

删除 2 节点

