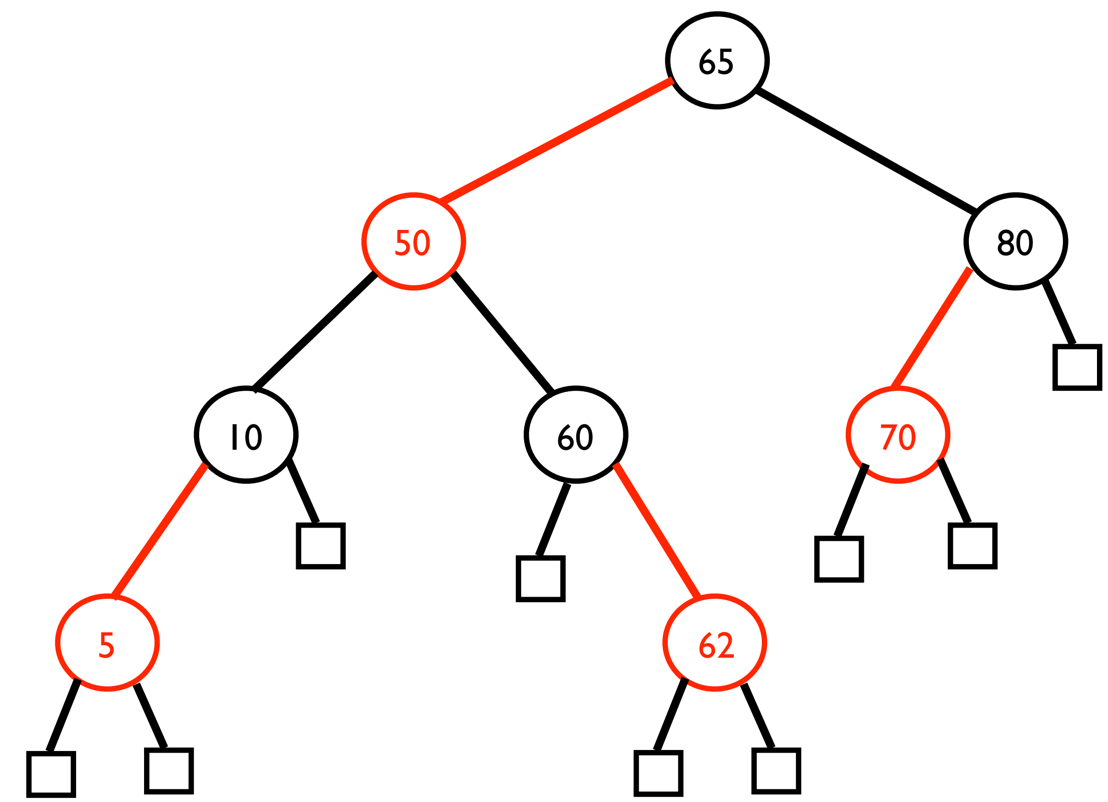
Data Structure:
Red-Black Tree

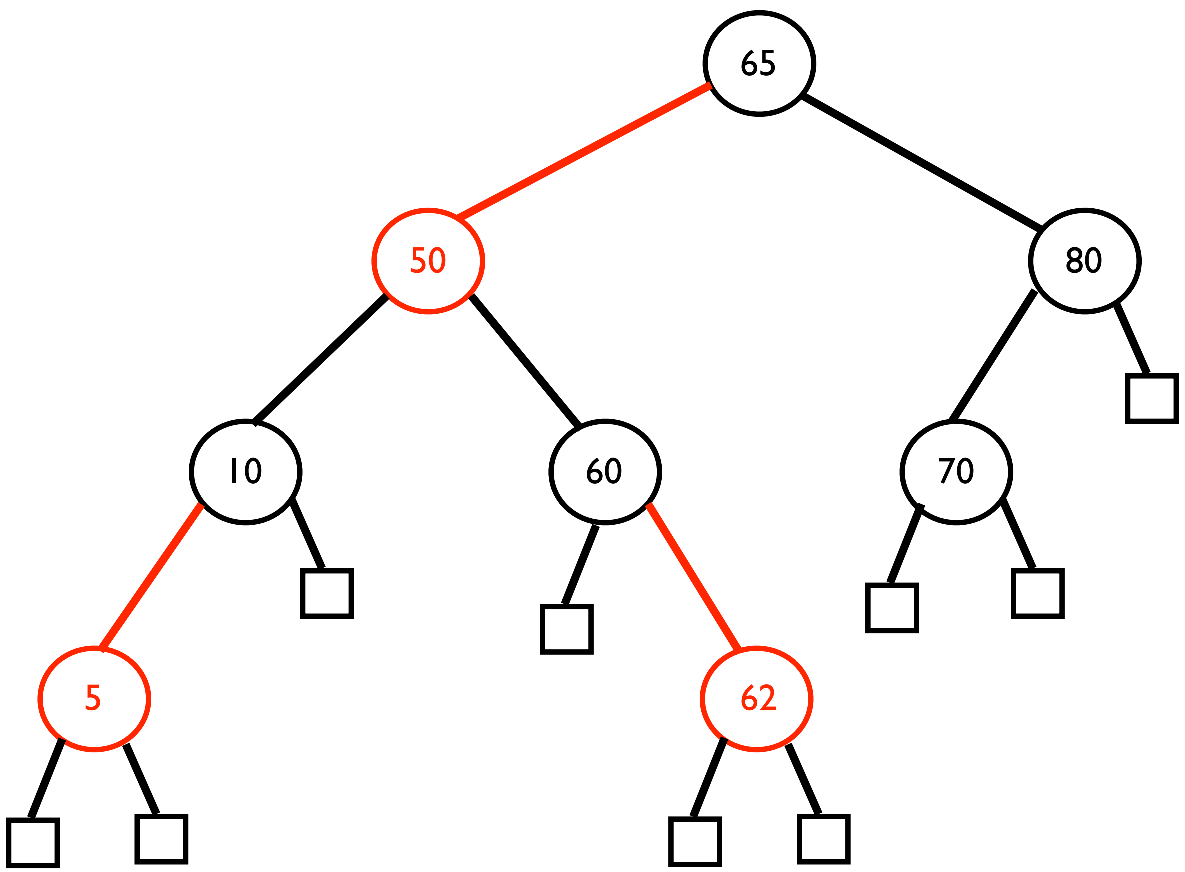
Red-Black Tree

- Red-black tree is a binary search tree in which every node is colored either red or black
- all properties are based on the extended binary search tree; each null pointer is replaced with an external node
- a pointer to a black child (including the external node) is black; a pointer to a red child is red

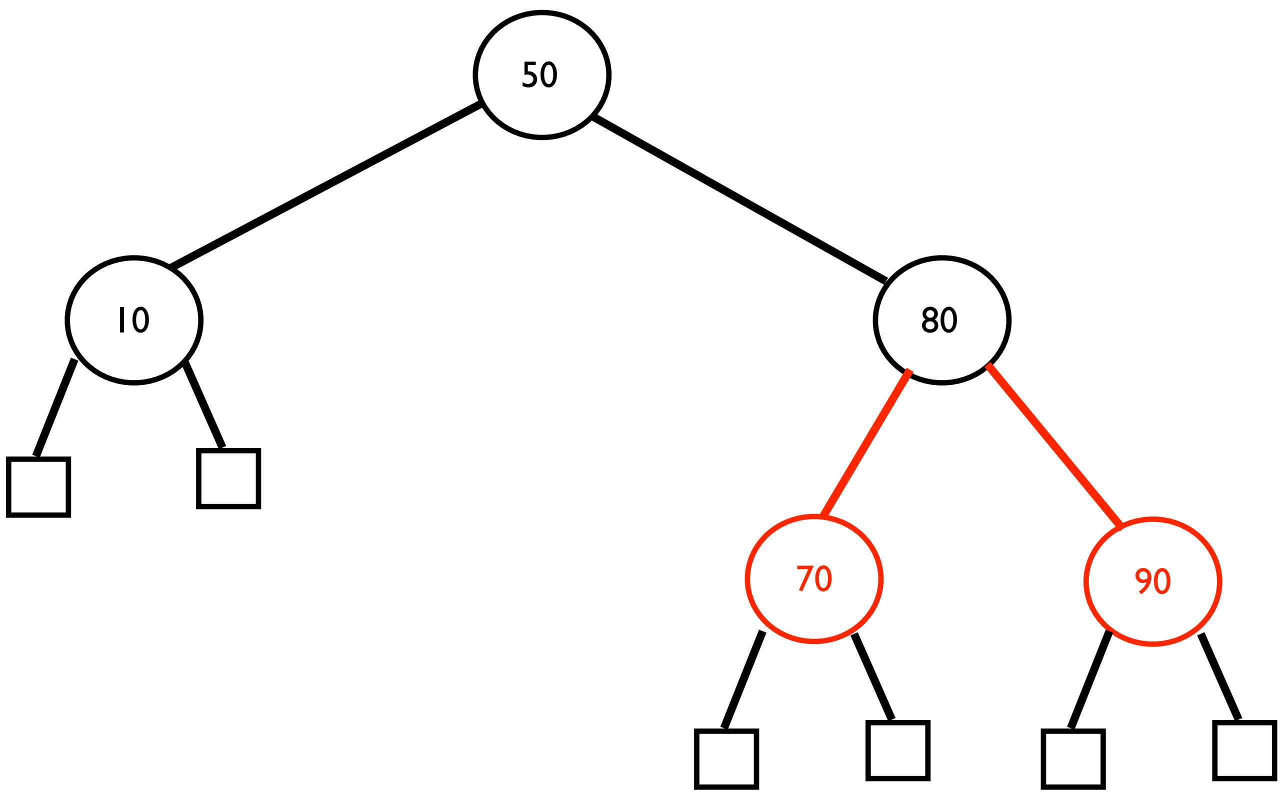


- properties of colored nodes
  - root and all external nodes are black
  - no consecutive red node is on the root-to-external node path
  - all root-to-external node paths have the same number of black nodes

Red-Black Tree



Red Black Tree?



Red Black Tree?

Red-Black Tree

- rank (black height) of a node is the number of black pointers on any path from the node to any external node
    - the rank of an external node is 0

- Lemma 1

  Let the length of a root-to-external-node path be the number of pointers on the path. If P and Q are two root-to-external-node paths in a red-black tree,

  $$length(P) \leq 2\ length(Q)$$

  Proof: When r is the rank of the root, each root-to-external-node path has between r (all black pointers) and 2r (red pointers in every other pointers) pointers

Red-Black Tree

- Lemma 2

  Let h be the height of a red-black tree (excluding the external nodes). Let n be the number of internal nodes in the tree and r be the rank of the root

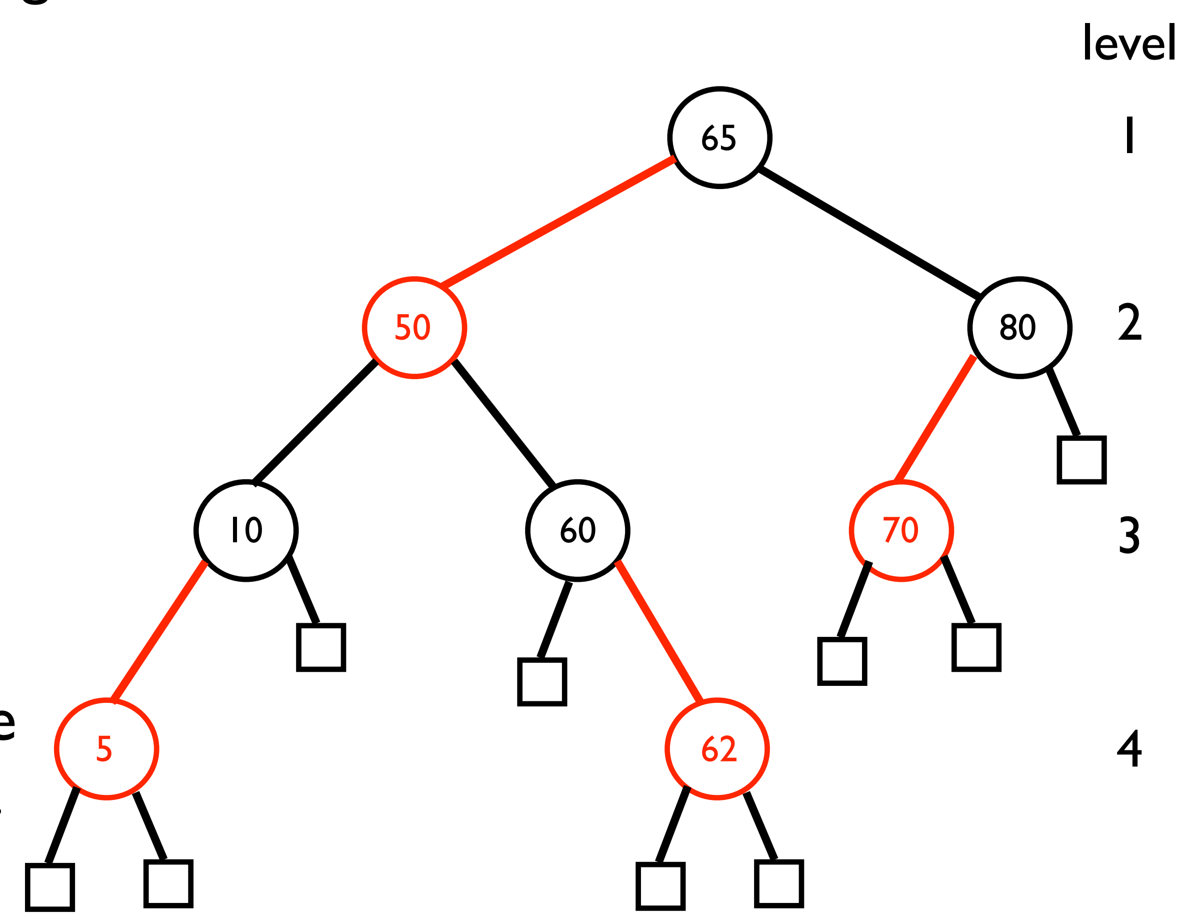
  - $h \le 2r$

    Proof: from Lemma 1, all root-to-external-node path has $\le$ 2r

  - $n \ge 2^r - 1$

    Proof: since the rank of the root is r, there are no external nodes at levels 1 through r. Thus, there are $2^r - 1$ internal nodes

  - $h \le 2 \log_2(n+1)$



- Since the height of a red-black tree is at most $2 \log_2(n+1)$, search, insert, and delete can be done in O(log n)

insertion

- if a new node is colored in black, we will have an extra black node on paths

  → always require recoloring

- if a new node is colored in red, we might have two consecutive red nodes

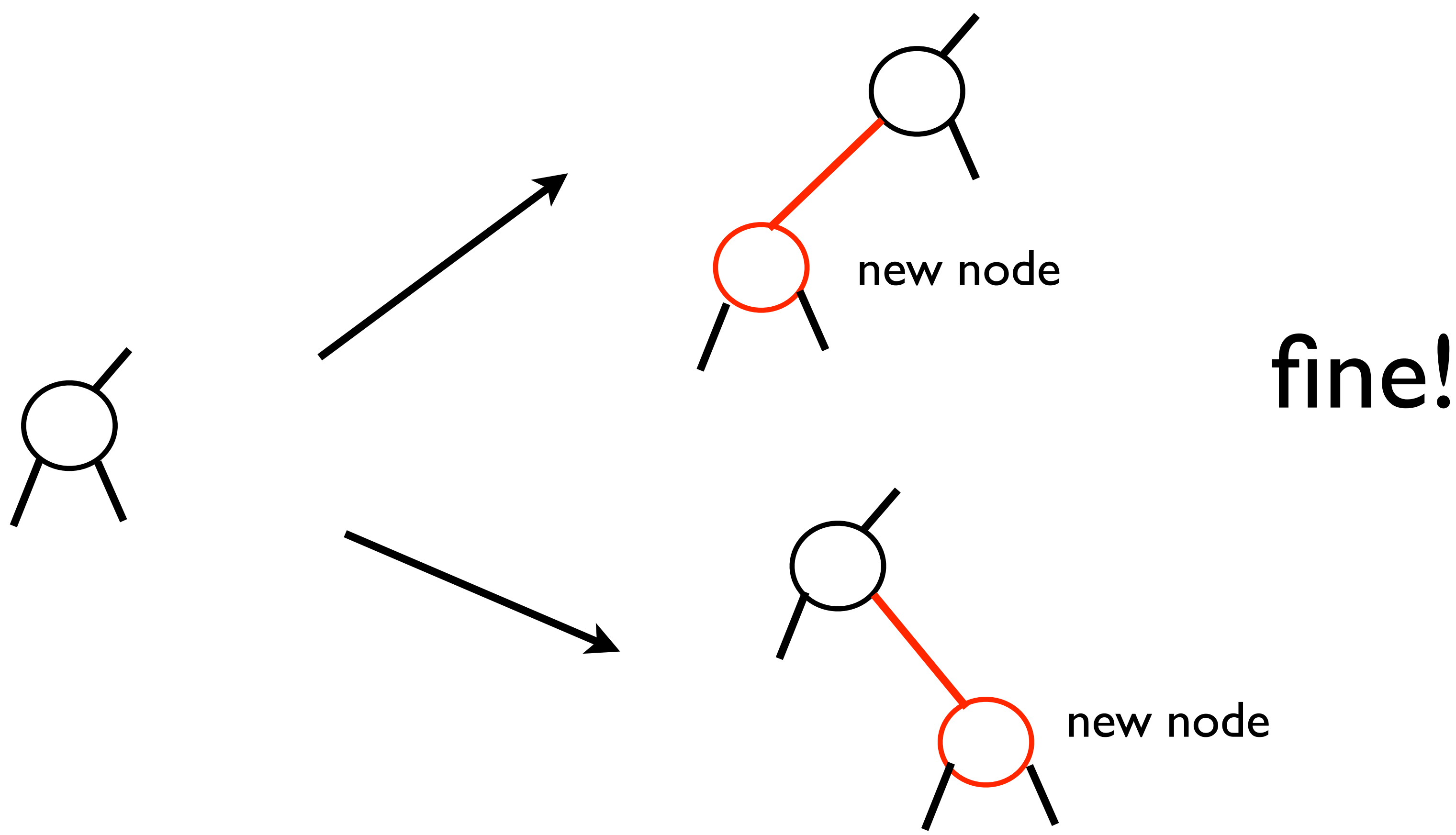
  → may or may not need recoloring

insertion

- insert
  - first, make a normal insert into a binary search tree
  - color it with red
  - fix the tree to meet the red-black properties

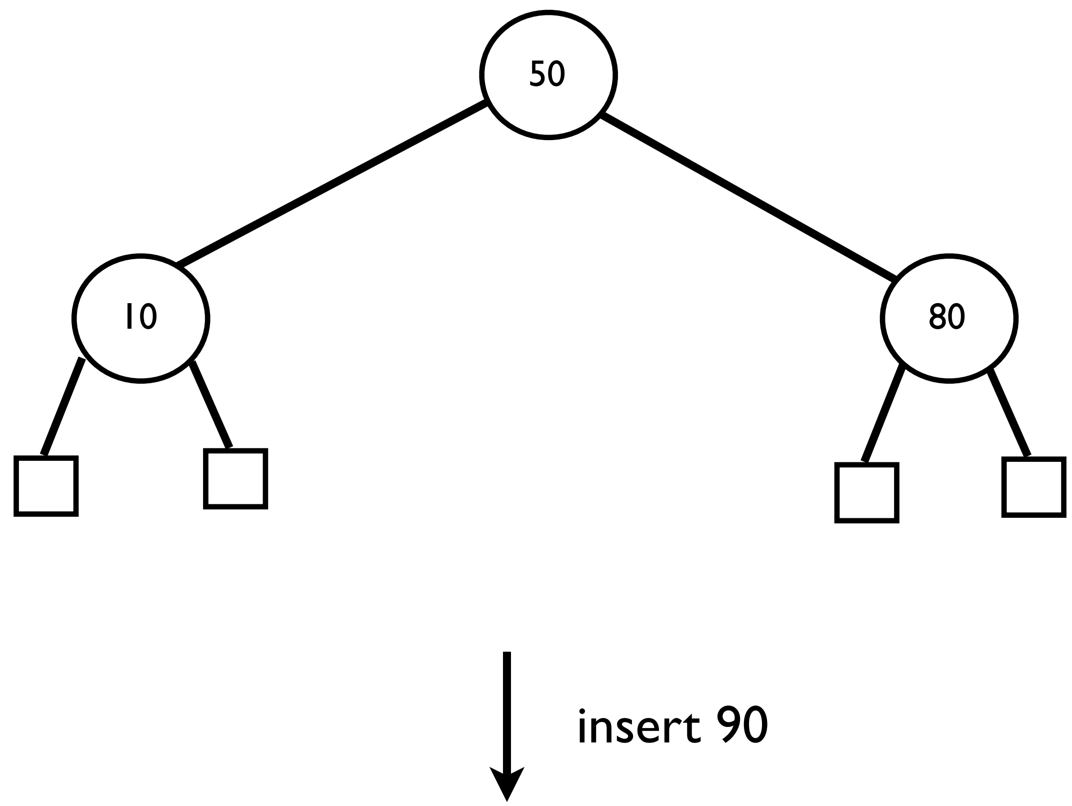
insertion

- when the parent of a new node is black

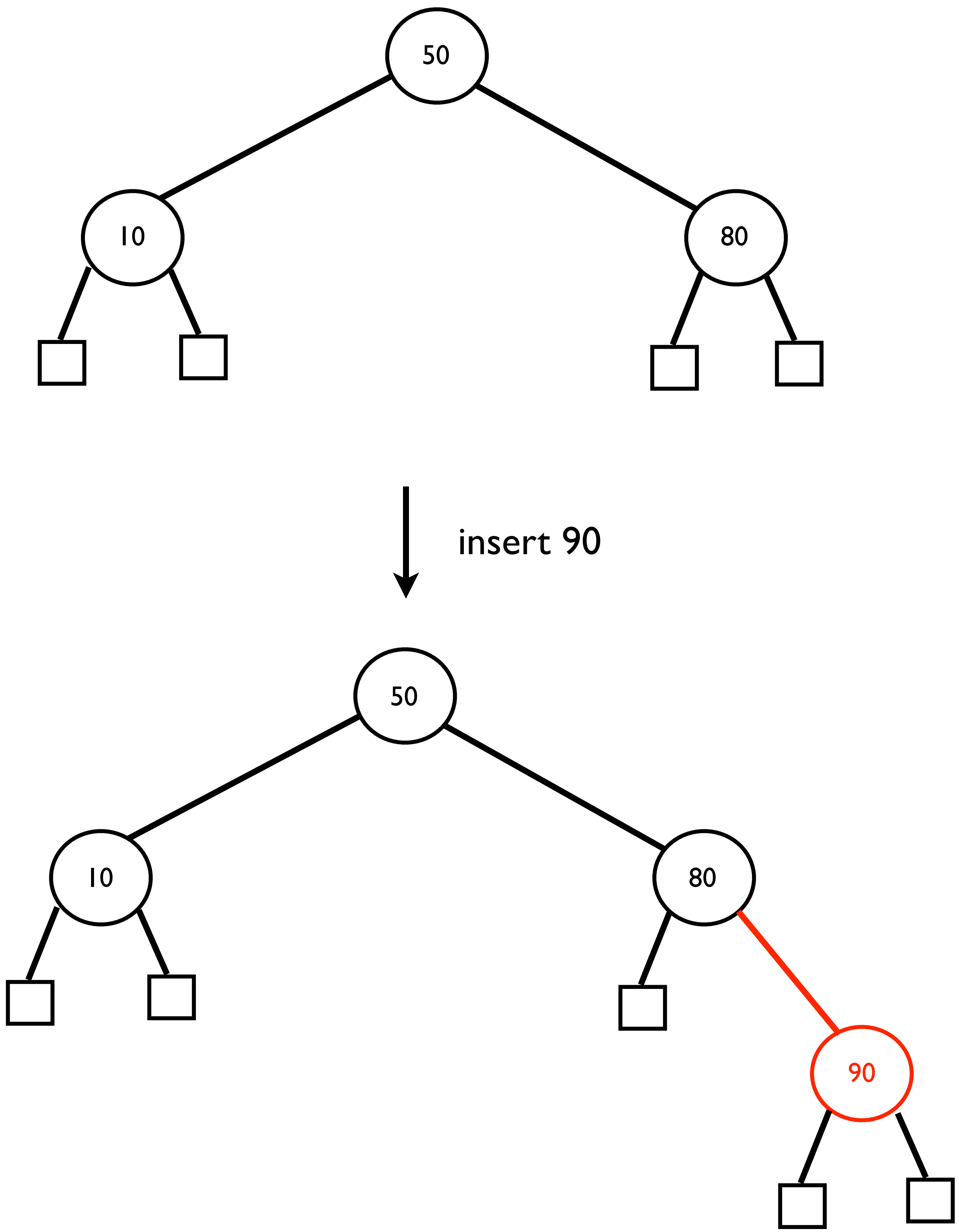


- properties of colored nodes
  - root and all external nodes are black
  - no consecutive red node is on the root-to-external node path
  - all root-to-external node paths have the same number of black nodes

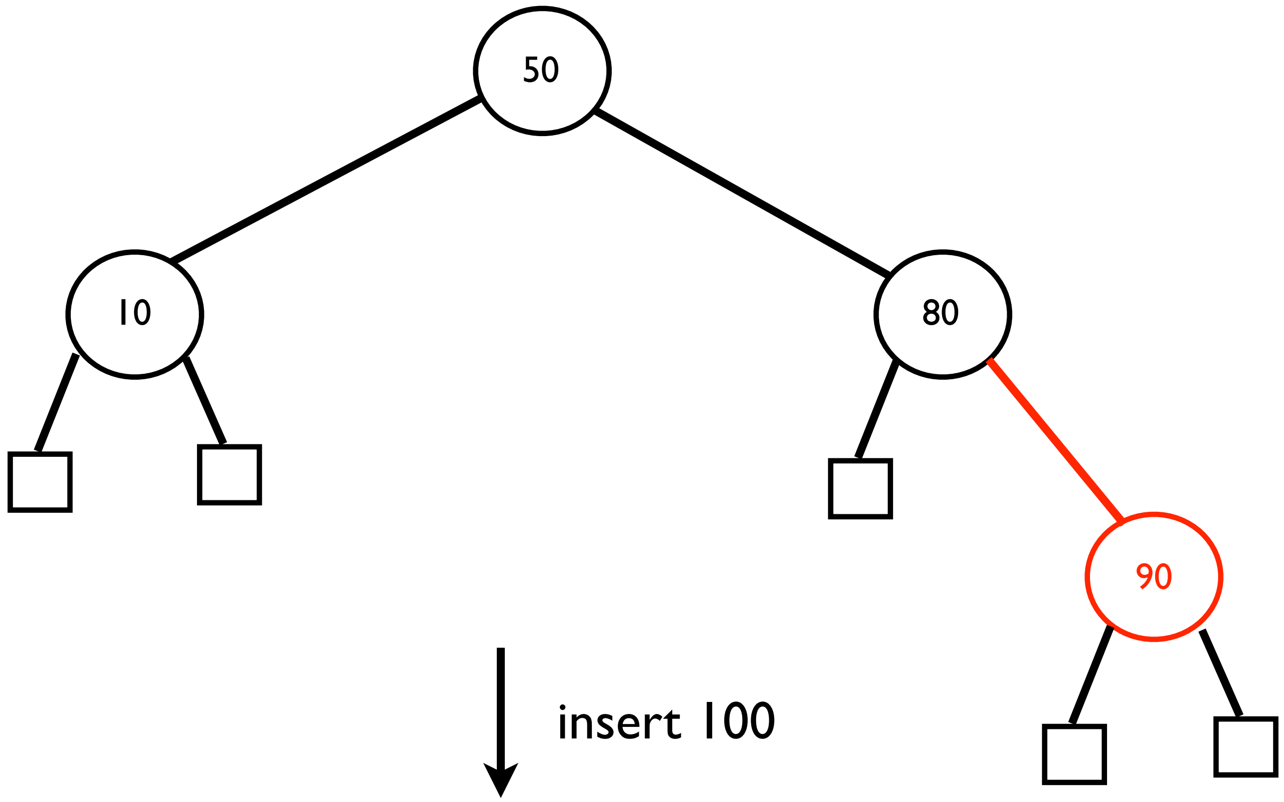
insertion



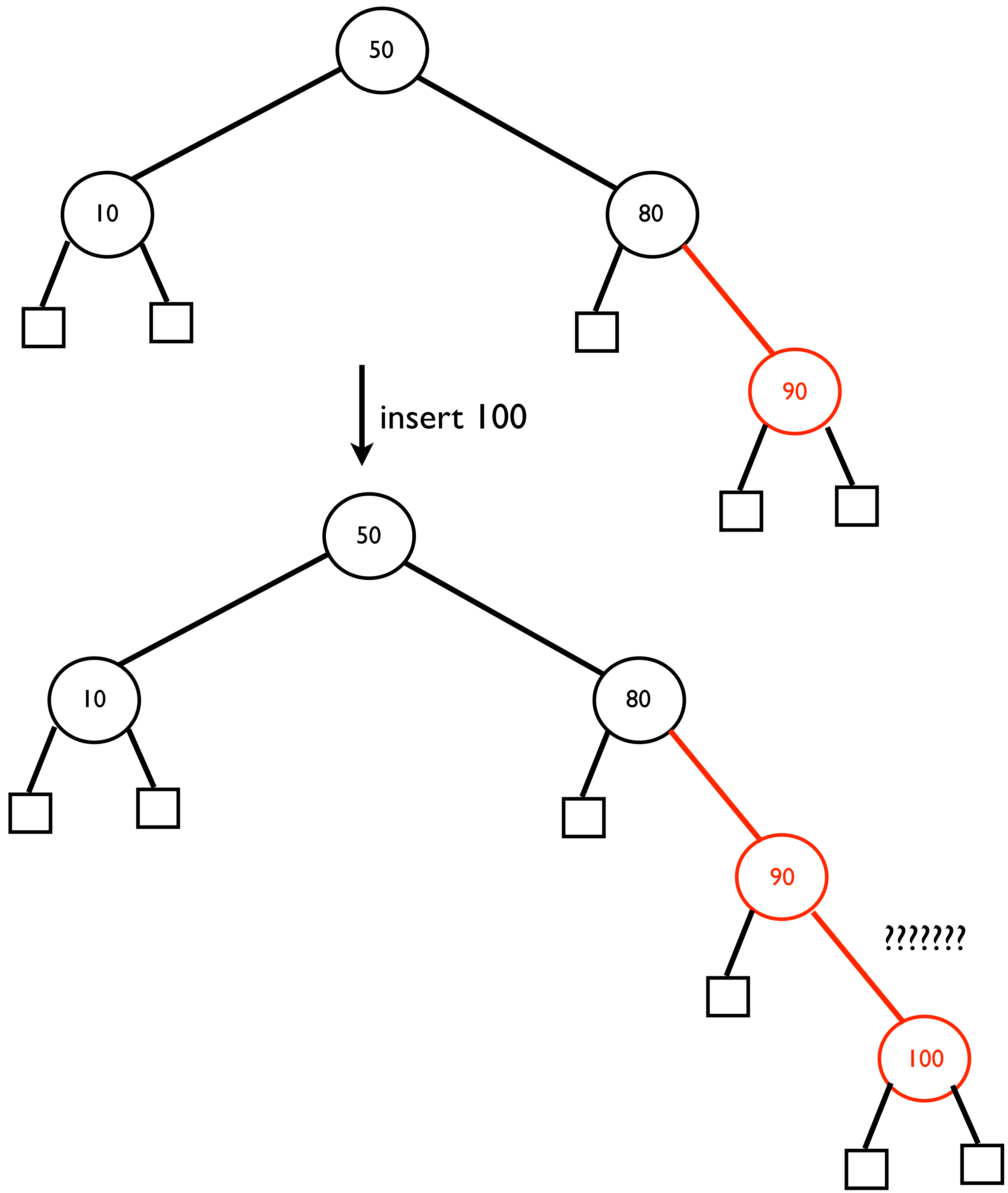
insertion



insertion

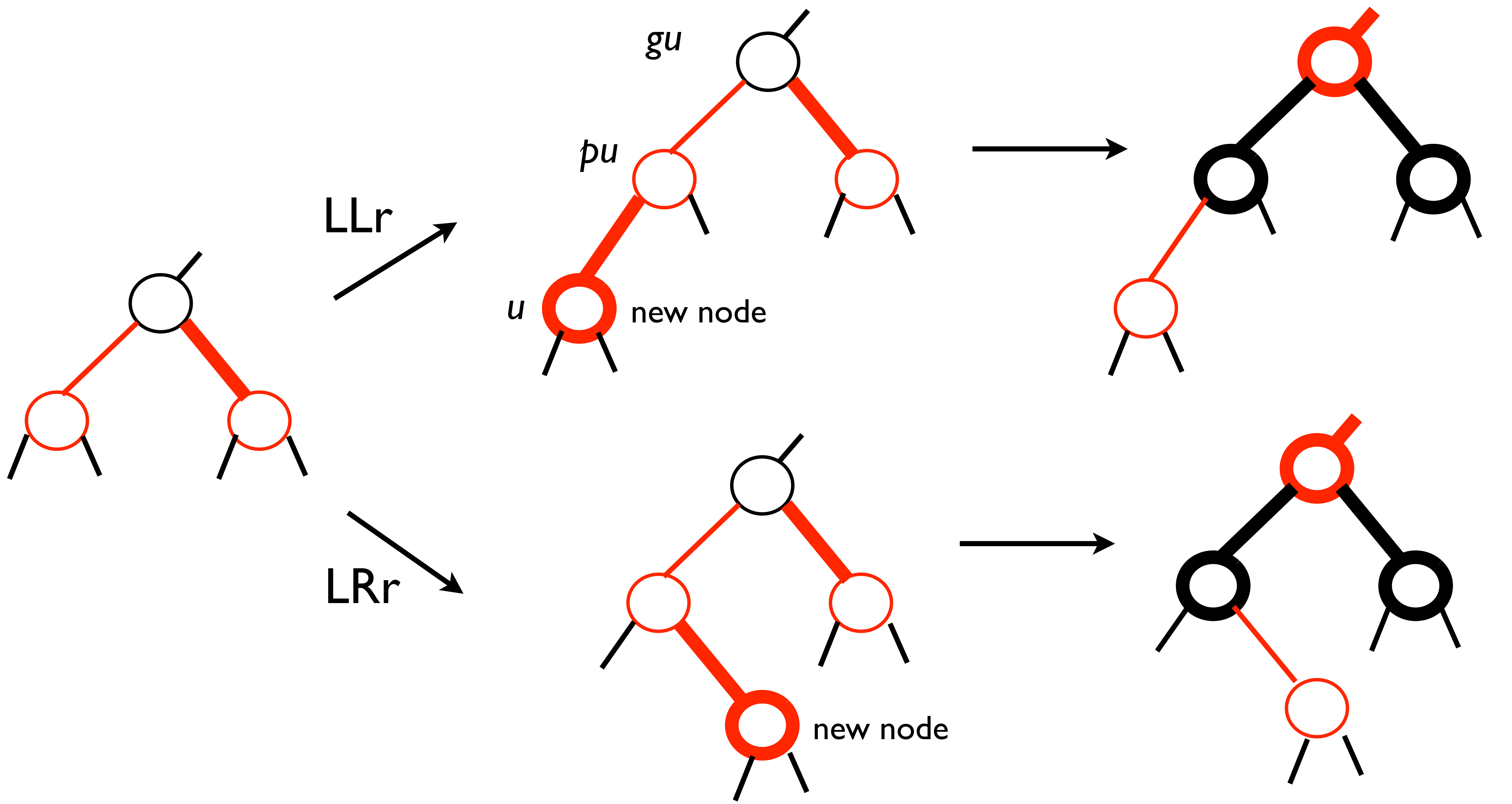


insertion



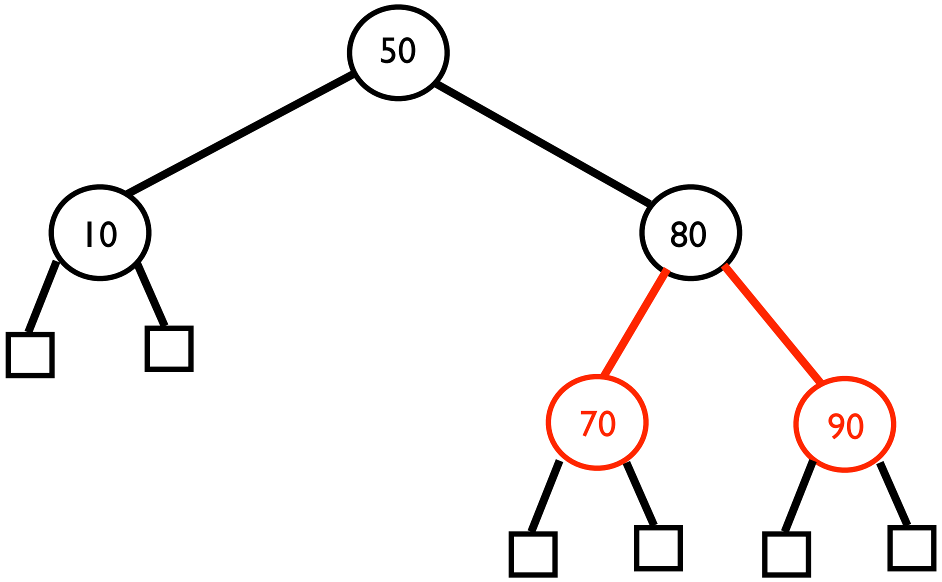
insertion

- when the parent of a new node is red



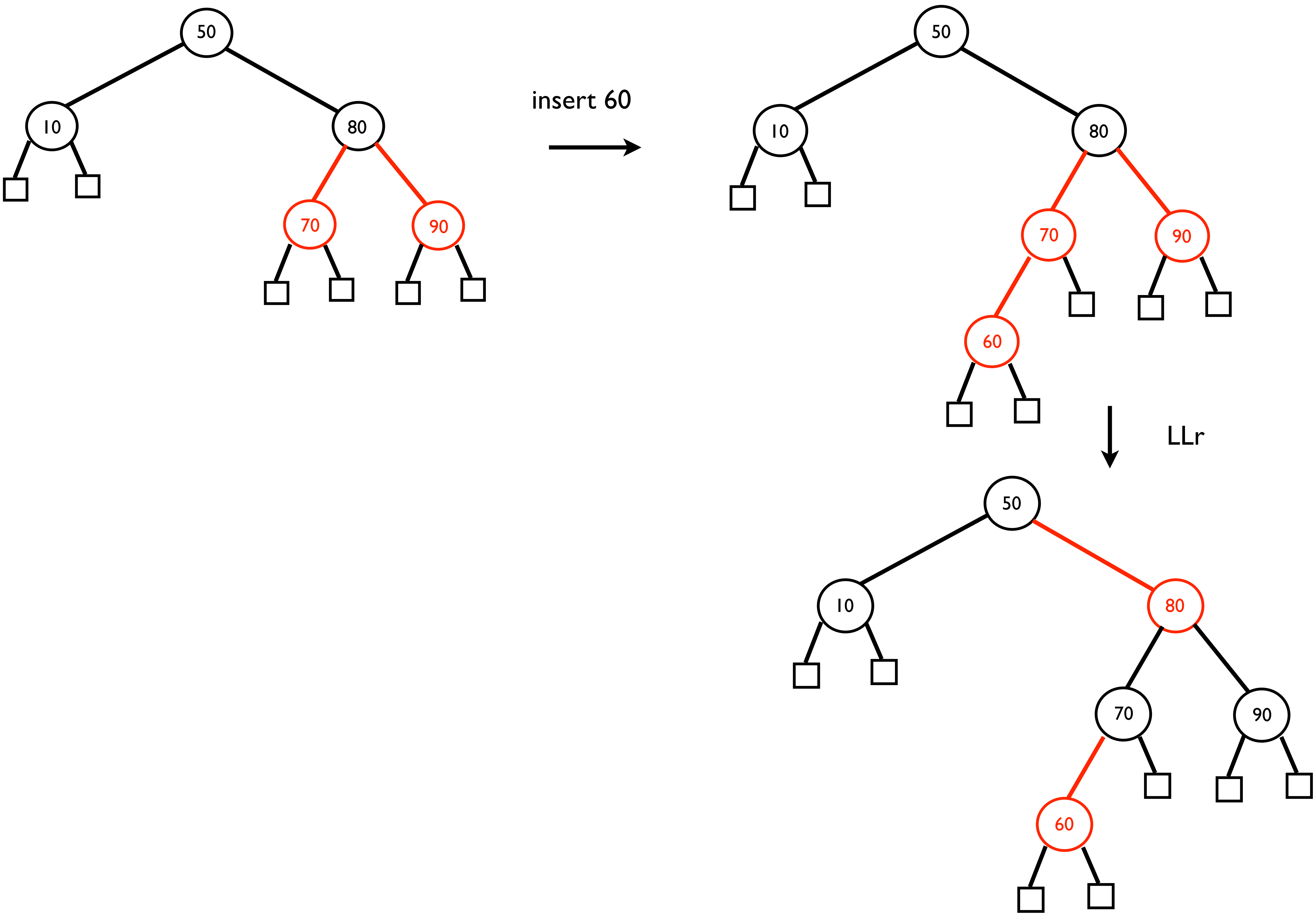
- Rank (black height) is fine
- if *gu* is the root, the number of black nodes on all root-toexternal-node paths increases by 1
- if changing of the color of *gu* to red causes an imbalance, *gu* becomes the new node *u*

insertion

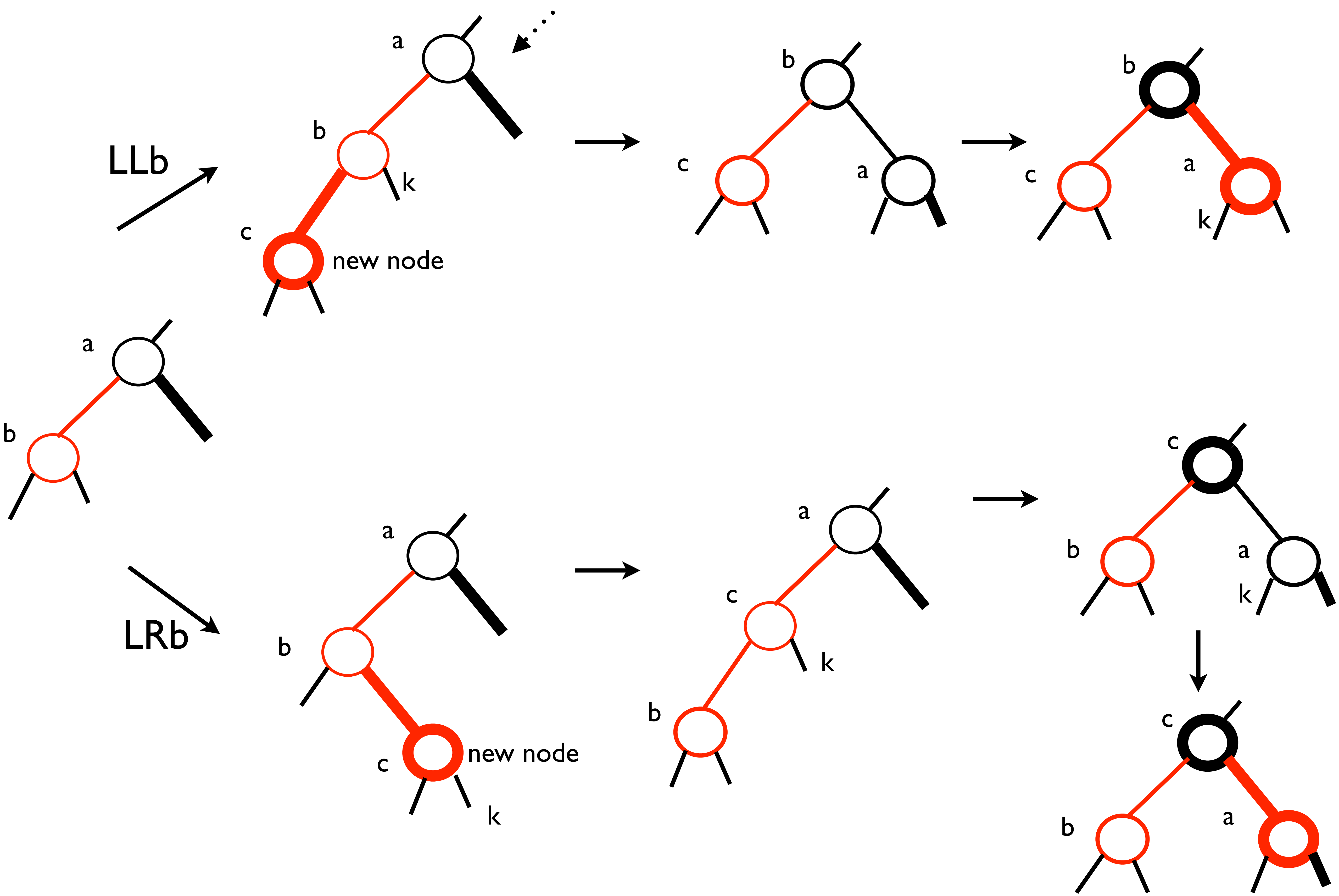


insert 60

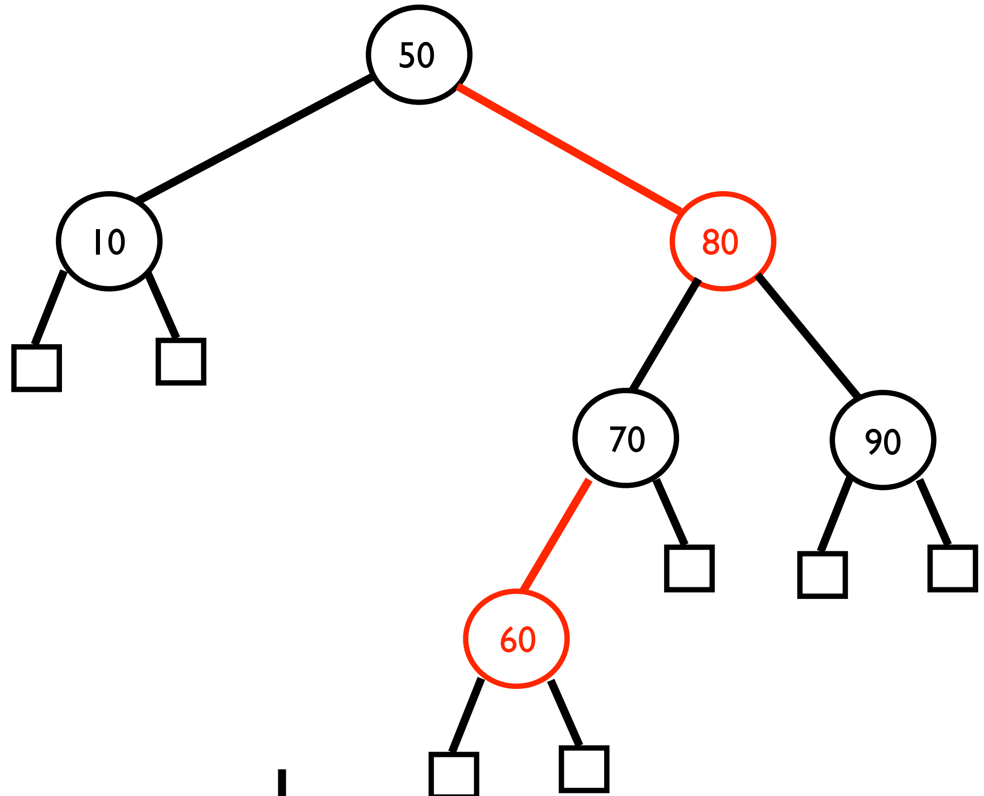
insertion



insertion

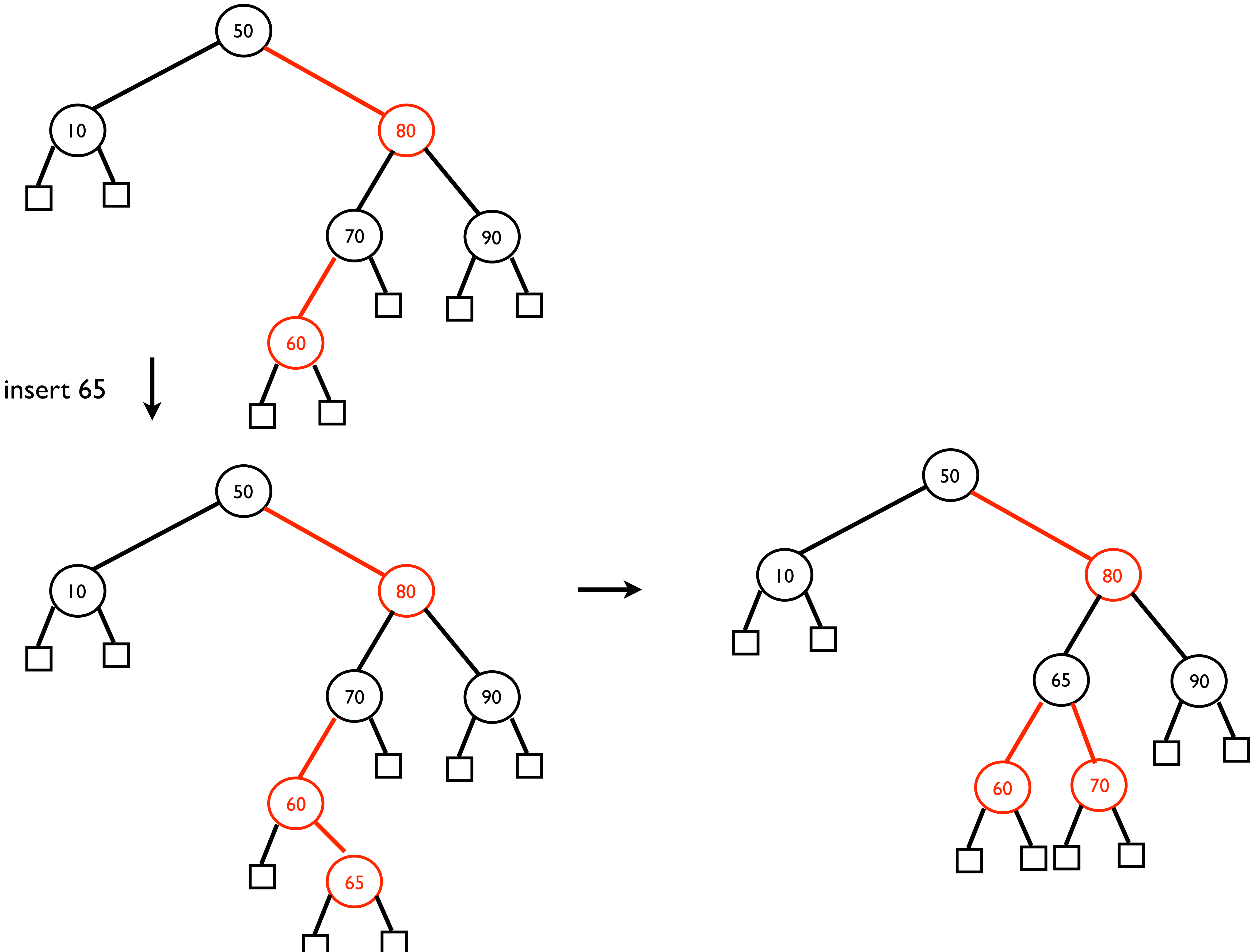


insertion

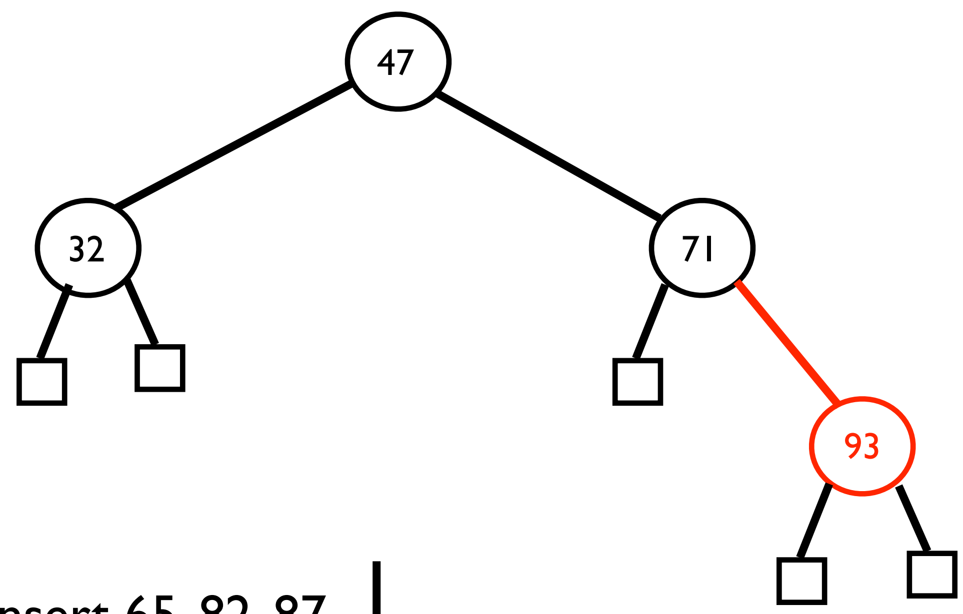


insert 65

insertion

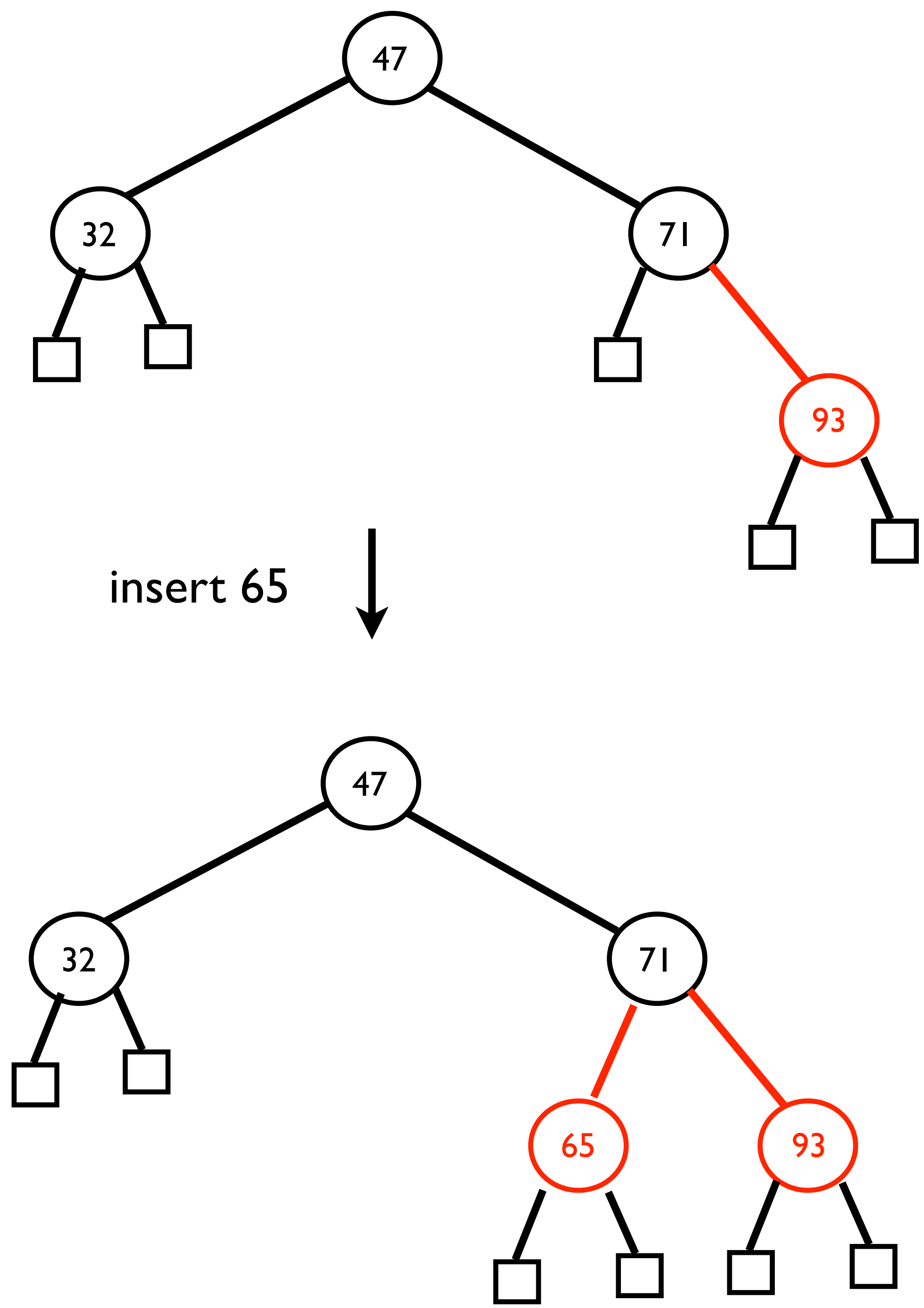


insertion

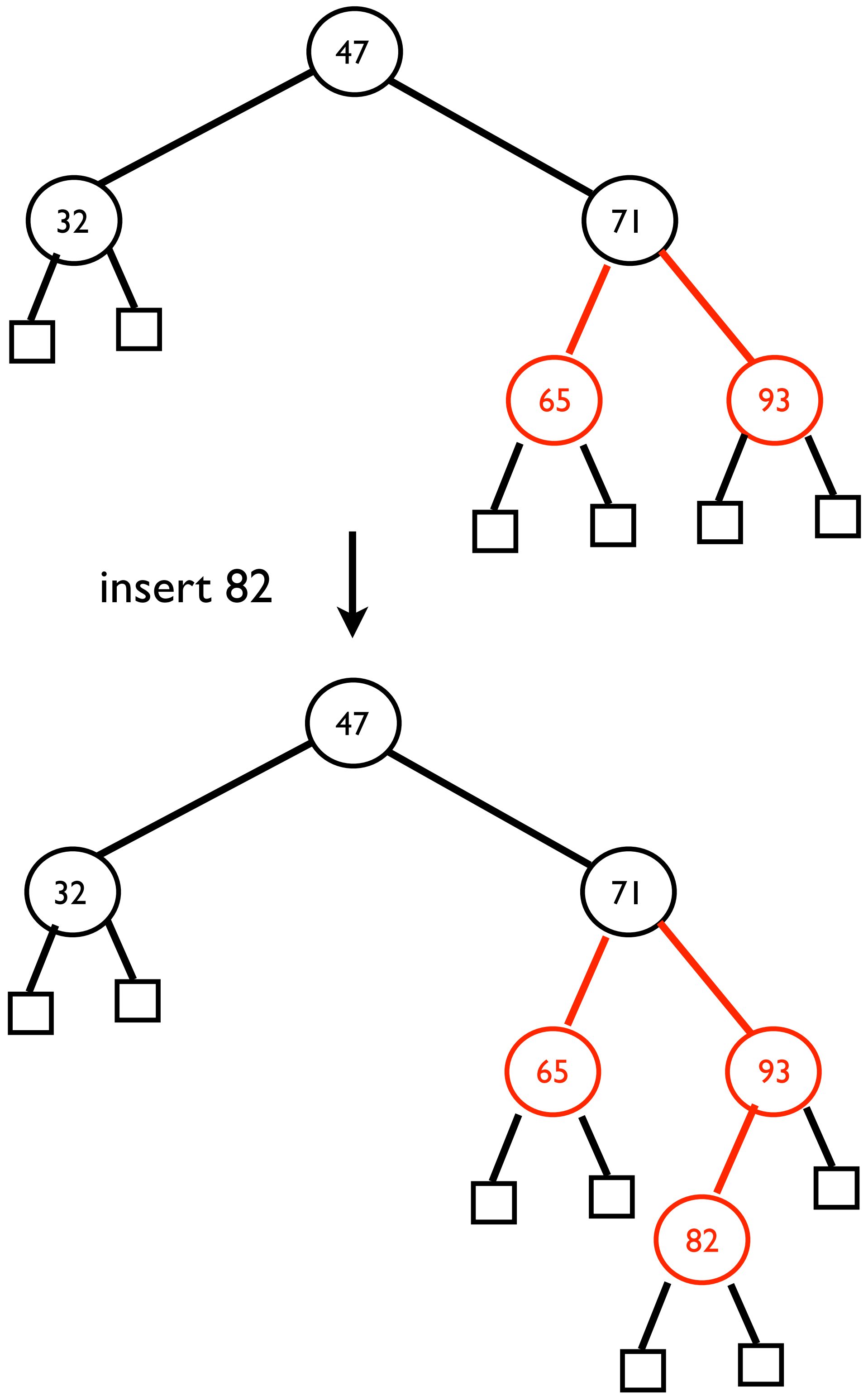


insert 65, 82, 87

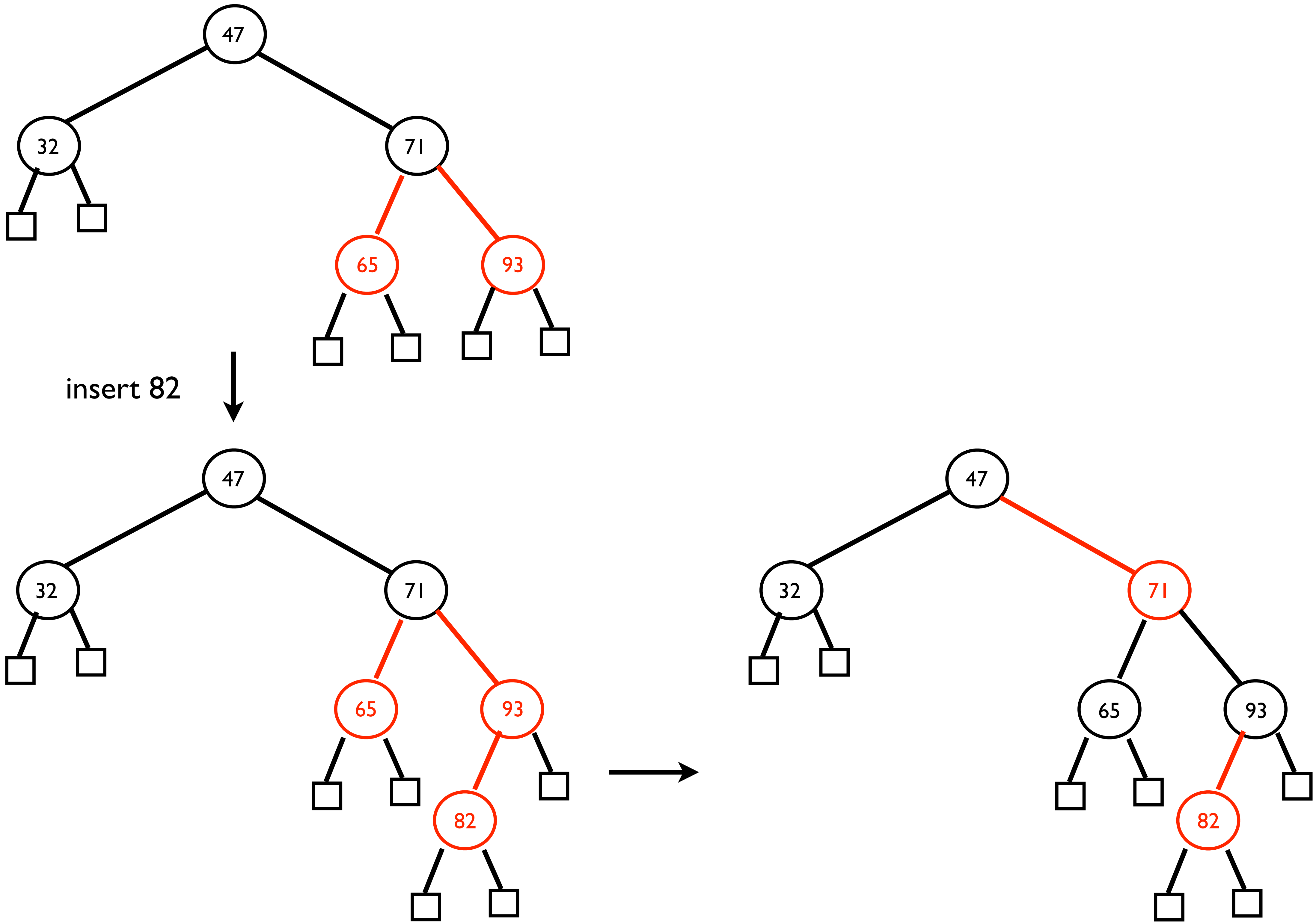
insertion



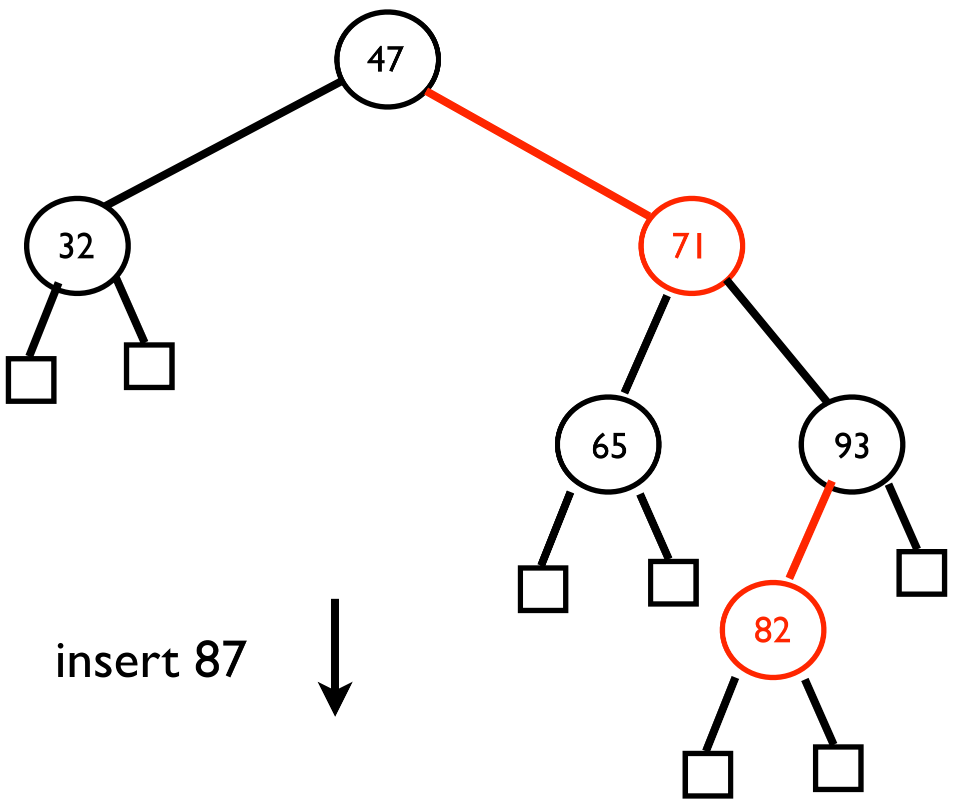
insertion



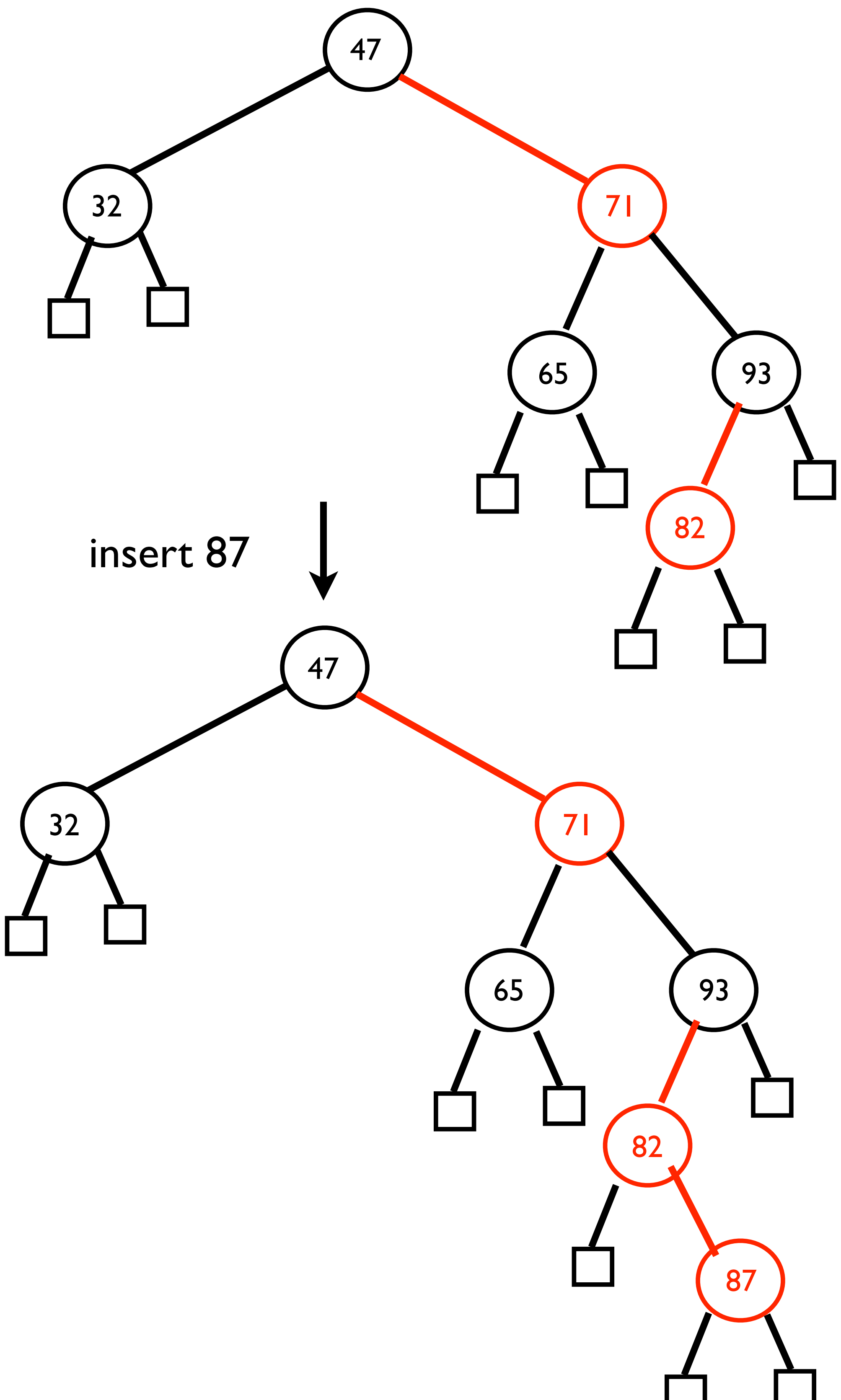
insertion



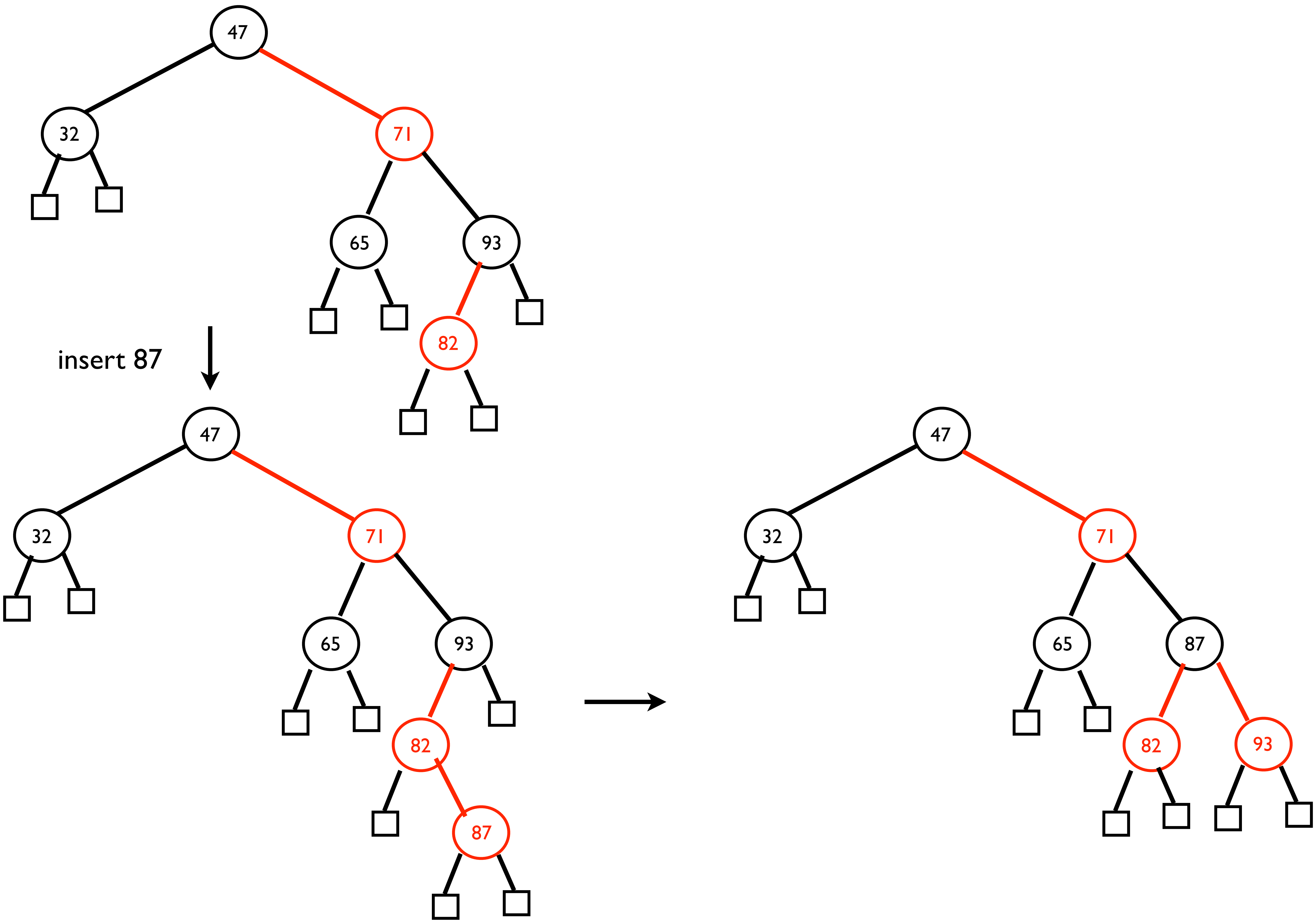
insertion



insertion



insertion



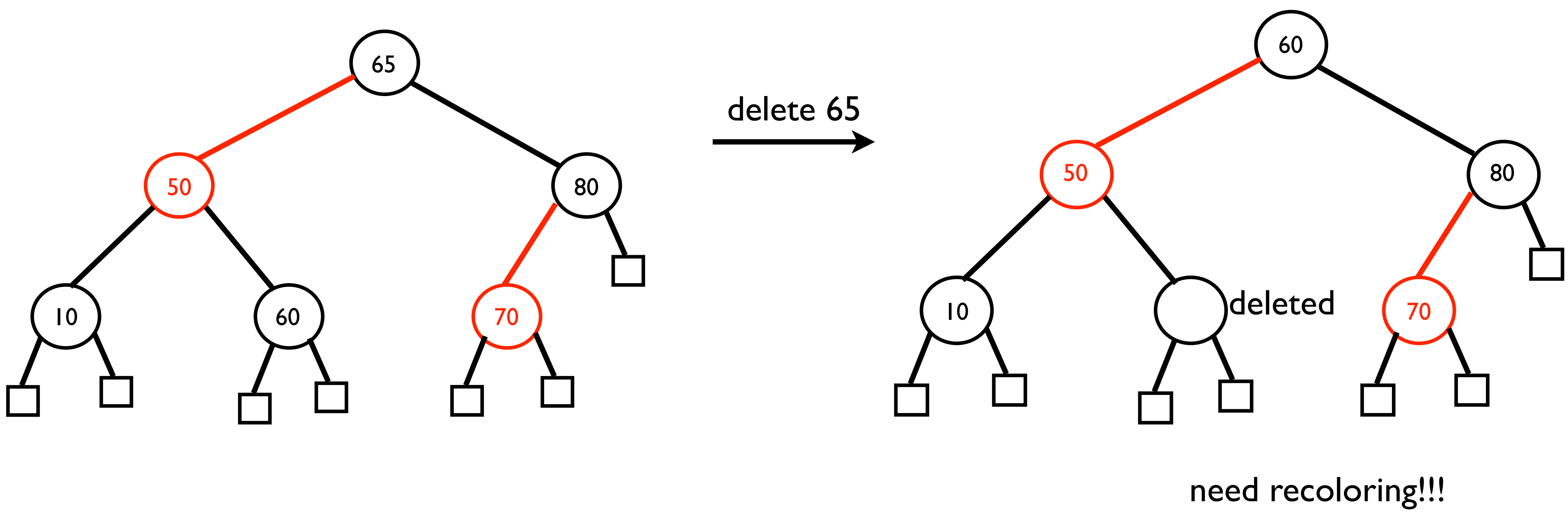
deletion

- we can delete a node with one or less external node in binary search tree

  → delete a node without any child or with one child in the binary search tree

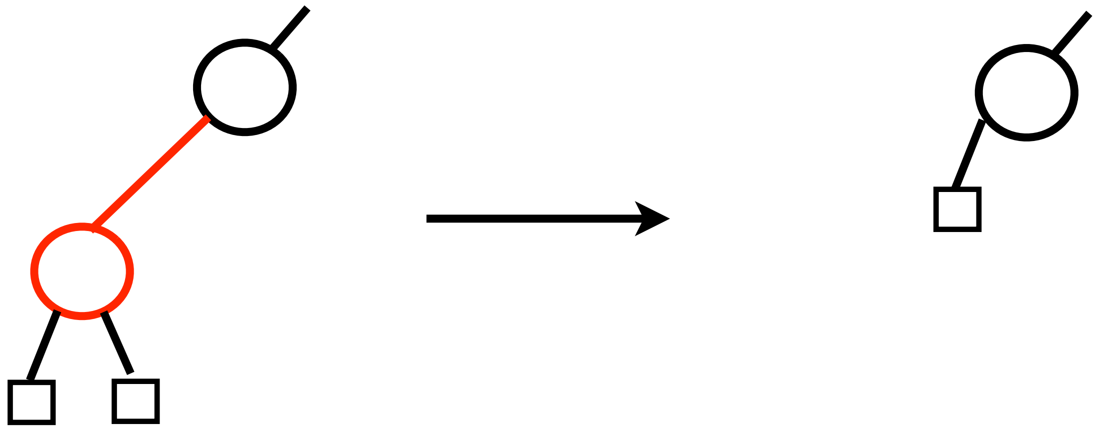
- when the node with two internal nodes is deleted, find the node of its predecessor or successor and delete that node

  →delete a node with both children in the binary search tree



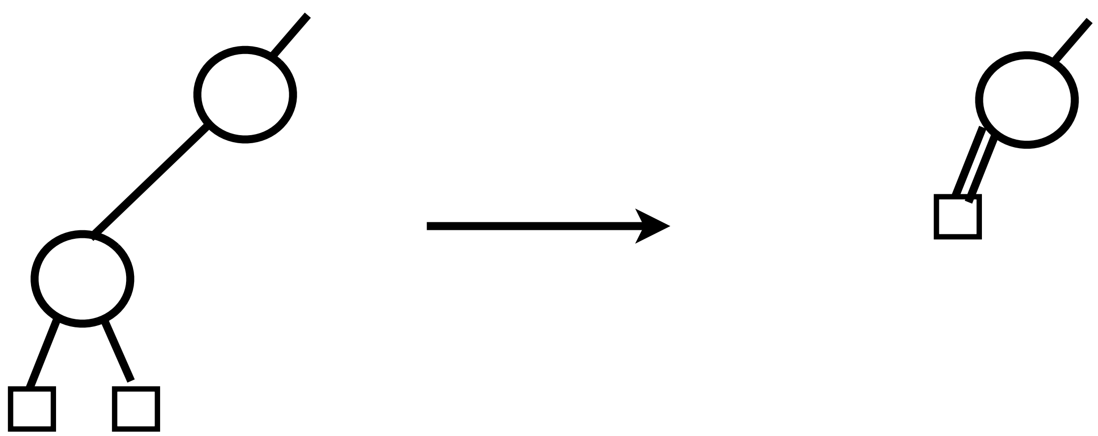
deletion

- if a red node is deleted, no rebalancing is needed since the rank is not changed (property #3)



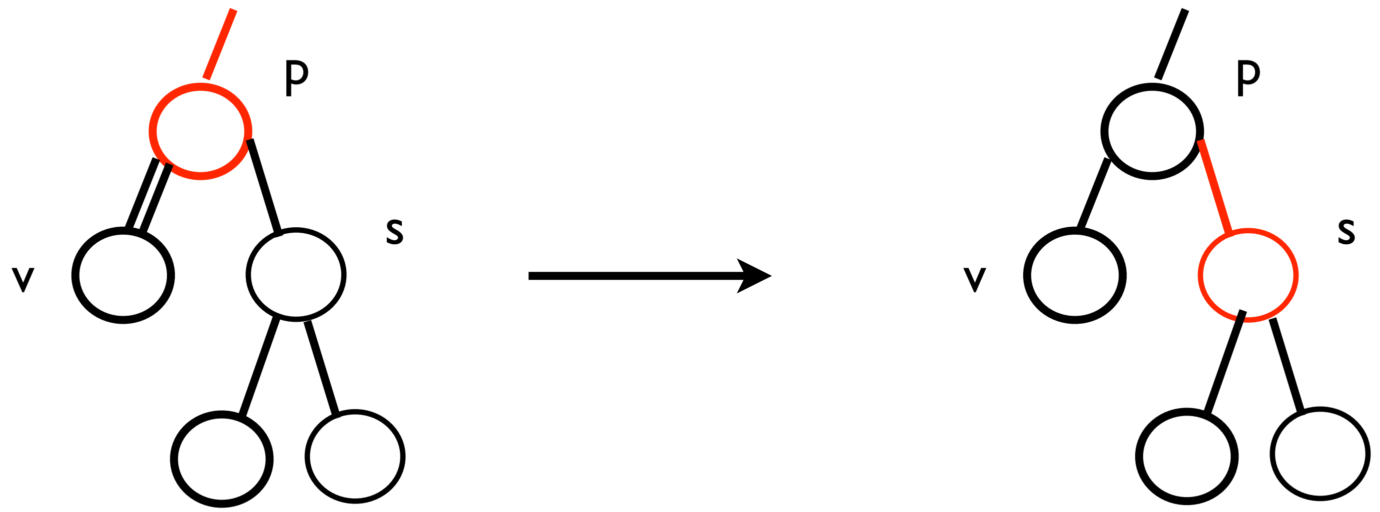
- if a black node is deleted, rebalancing is needed

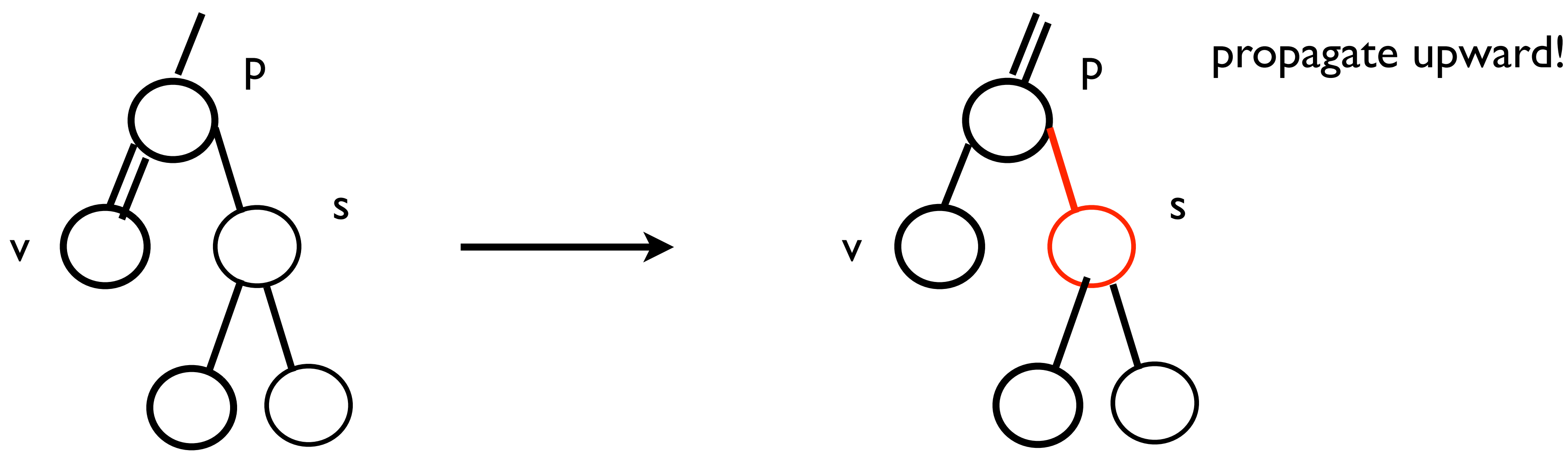
  → color the edge double black



deletion: how to remove the double edge

- black sibling with black children (including any internal node)

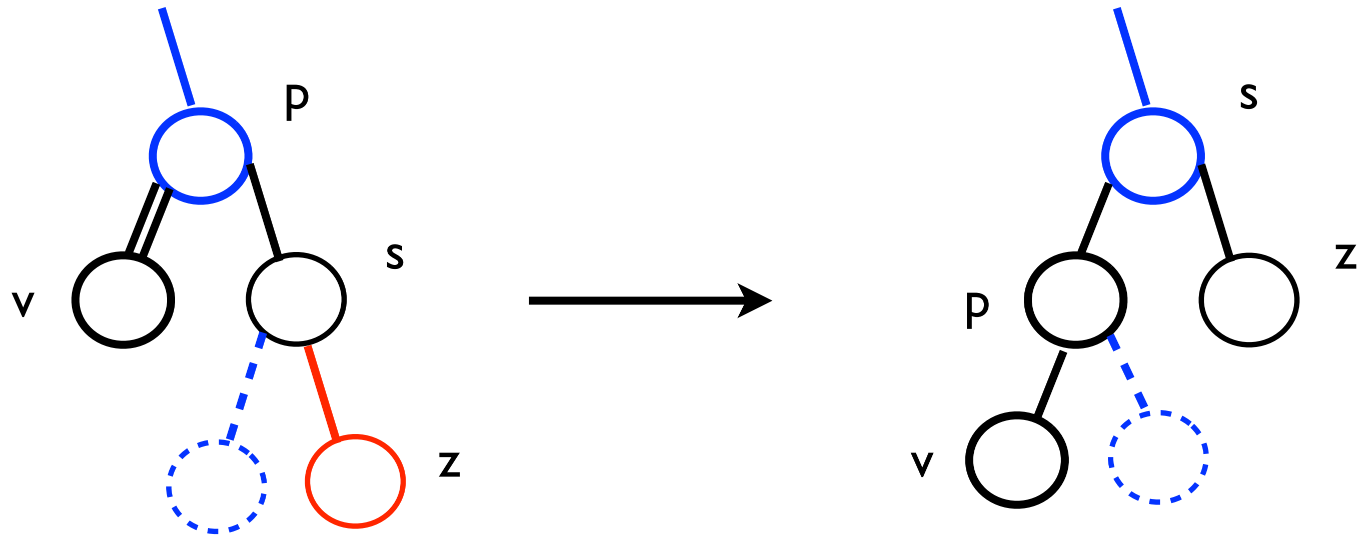


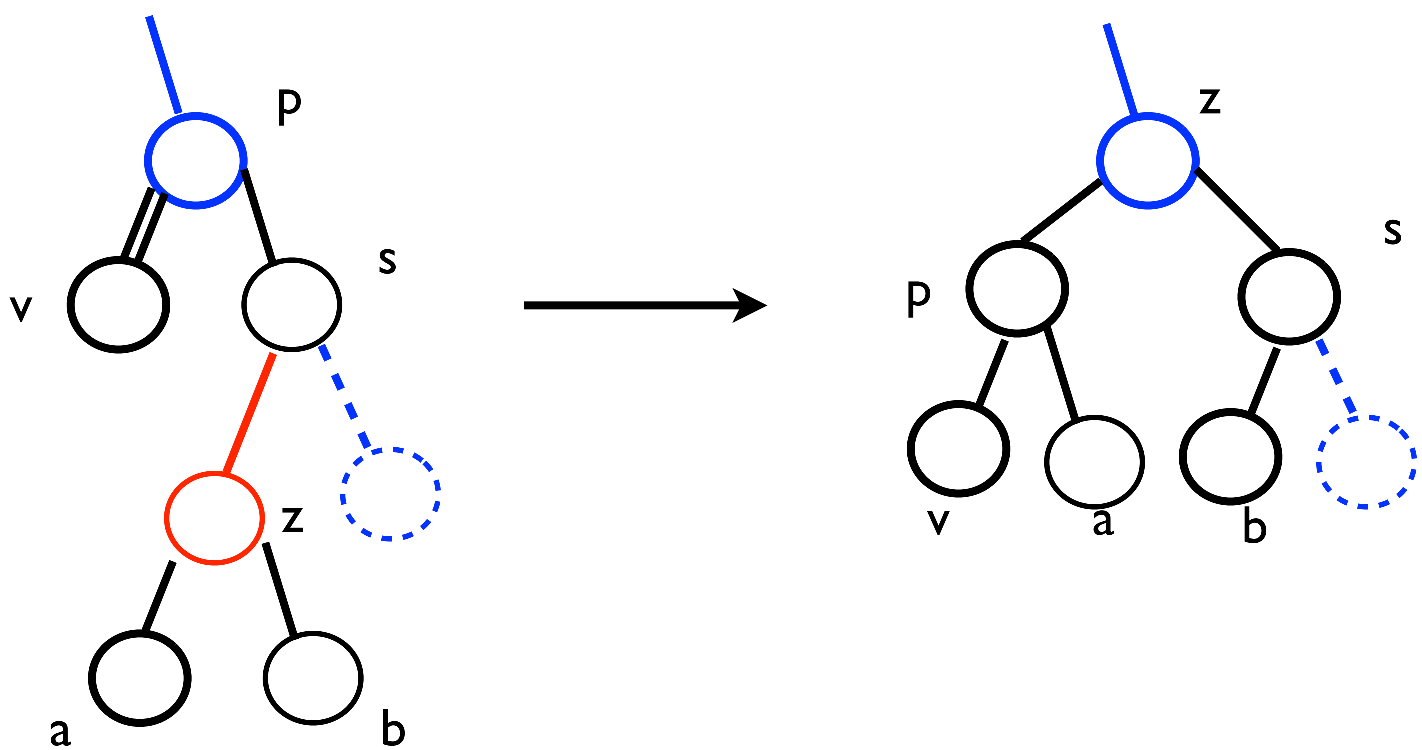


deletion: how to remove the double edge

- a node in blue can be either black or red
- a node in dotted line can exist or not exist

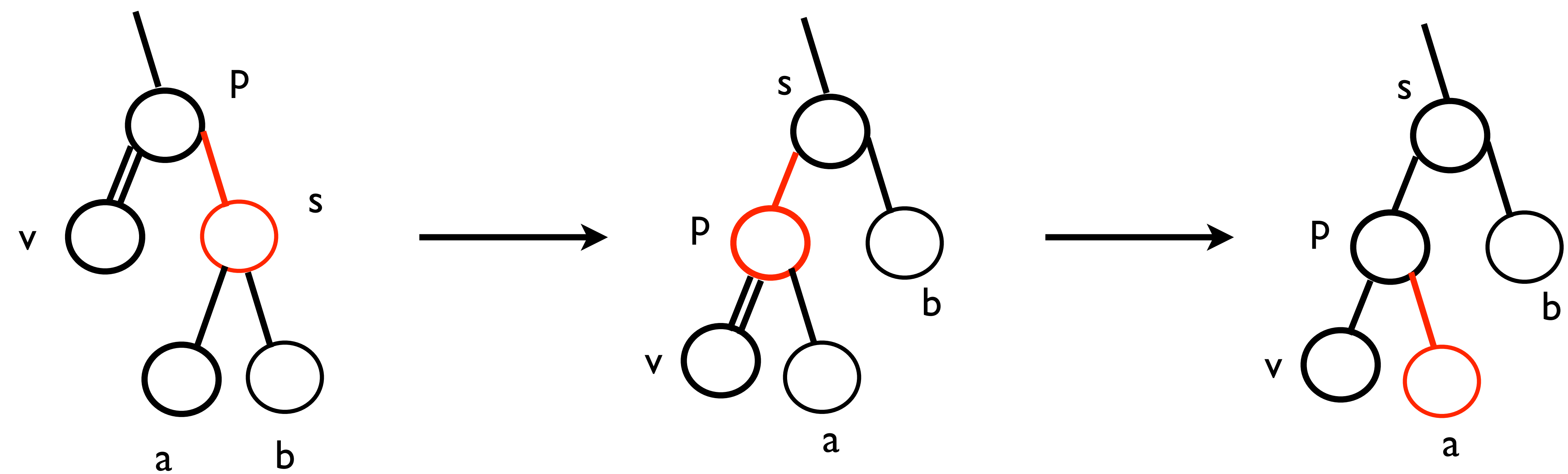
- black sibling with a red child (need rotation)



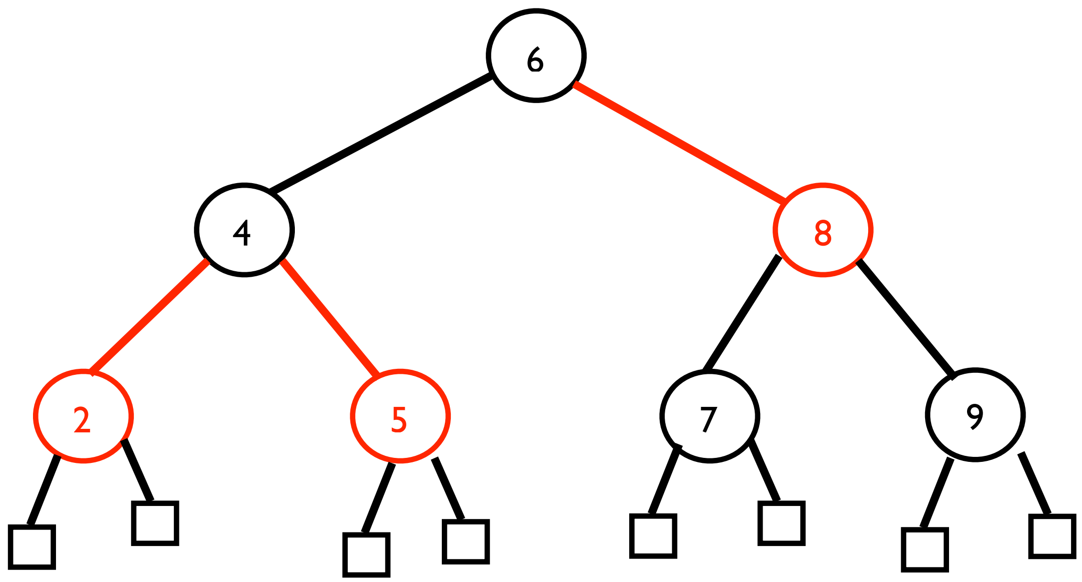


deletion: how to remove the double edge

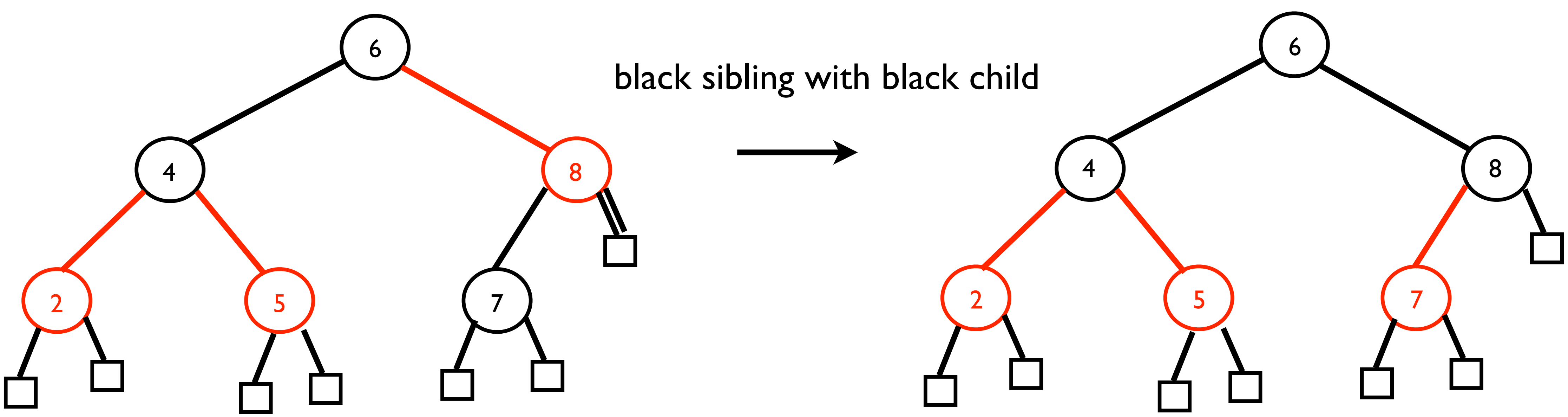
- red sibling
  - → restructure to have a black sibling



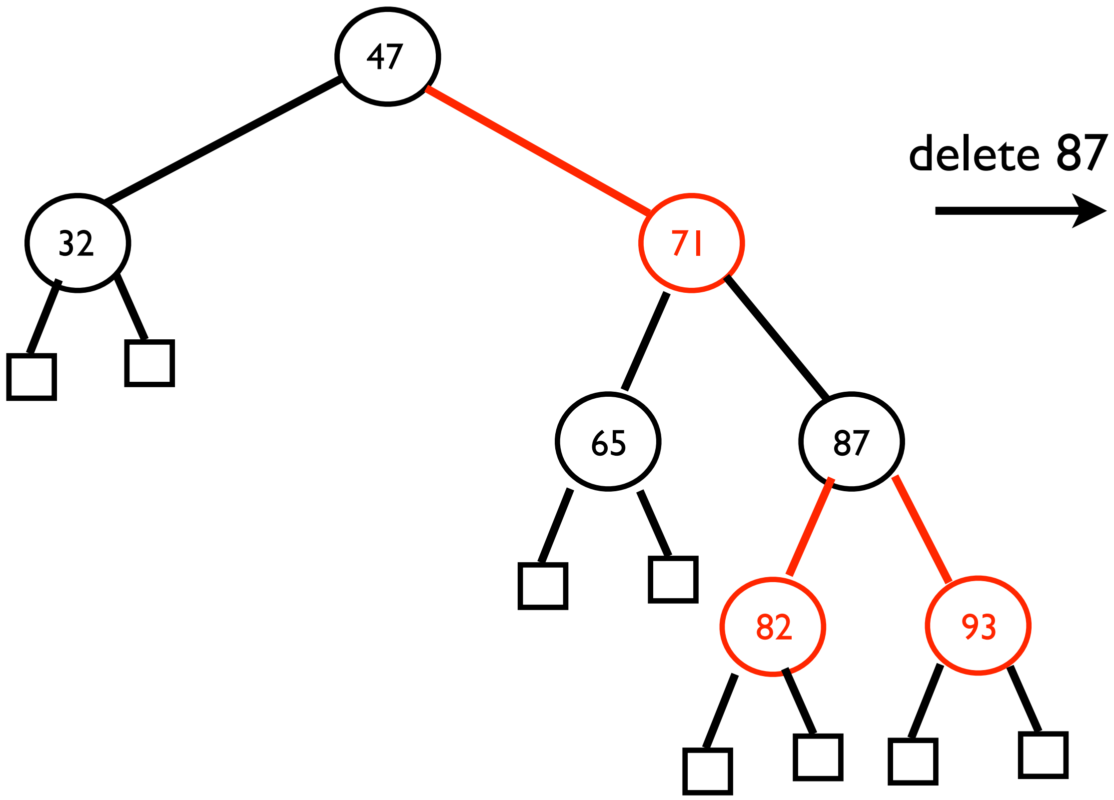
deletion: example



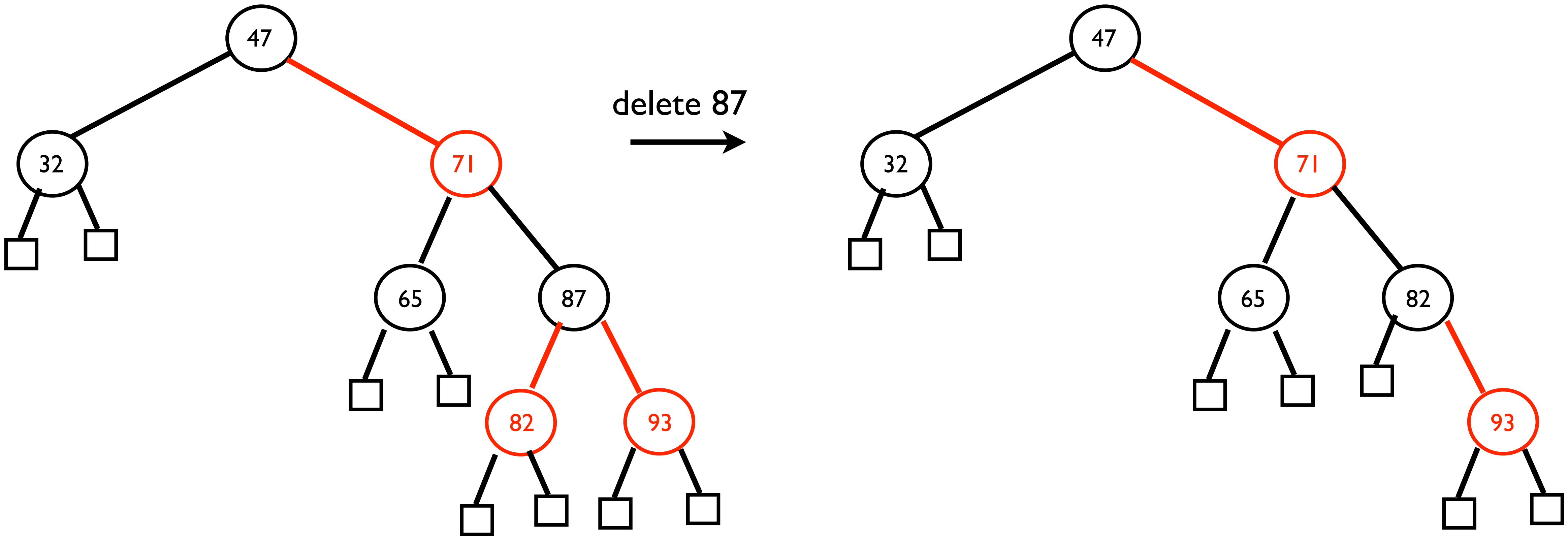
delete 9



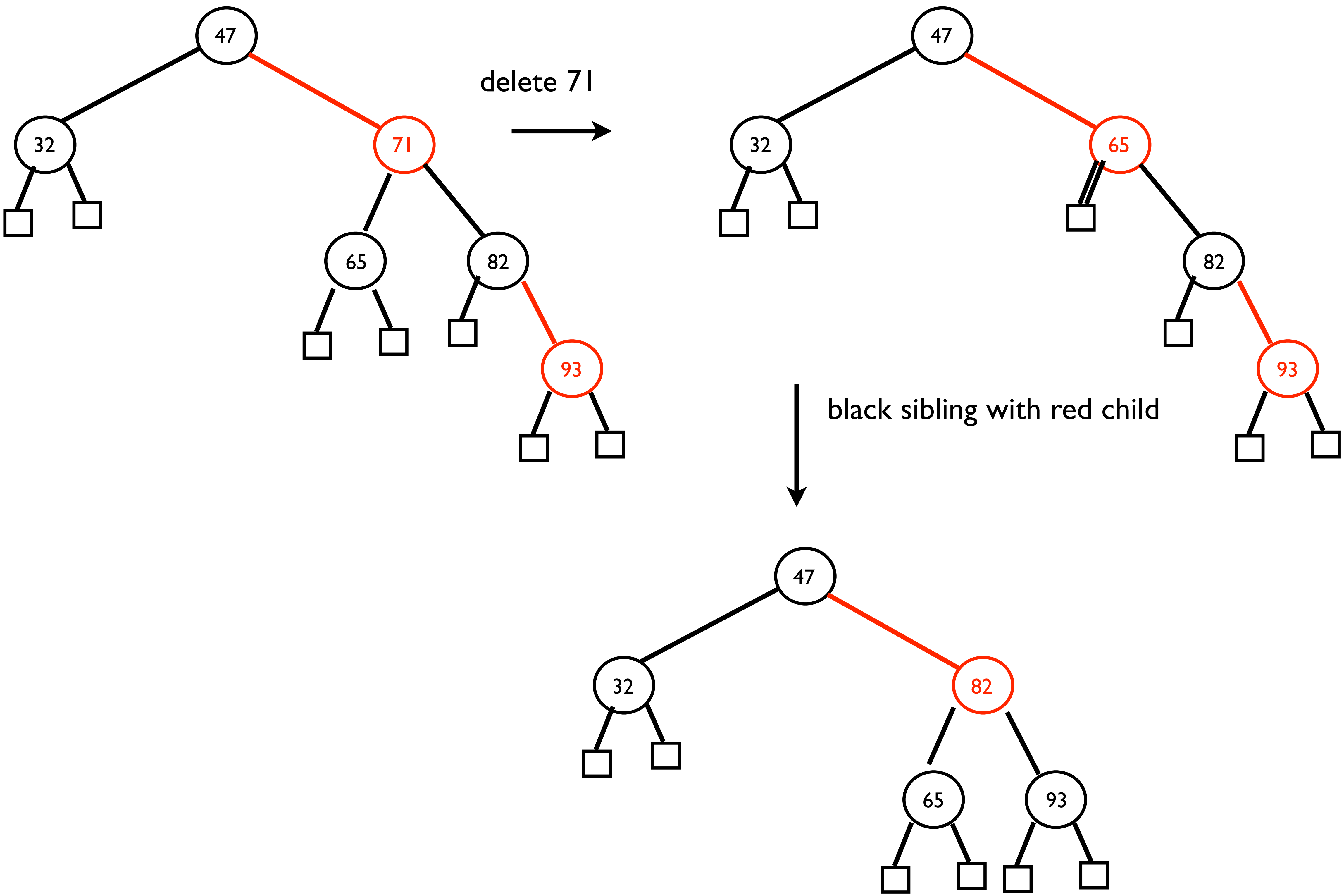
deletion: example



deletion: example



deletion: example



deletion: example