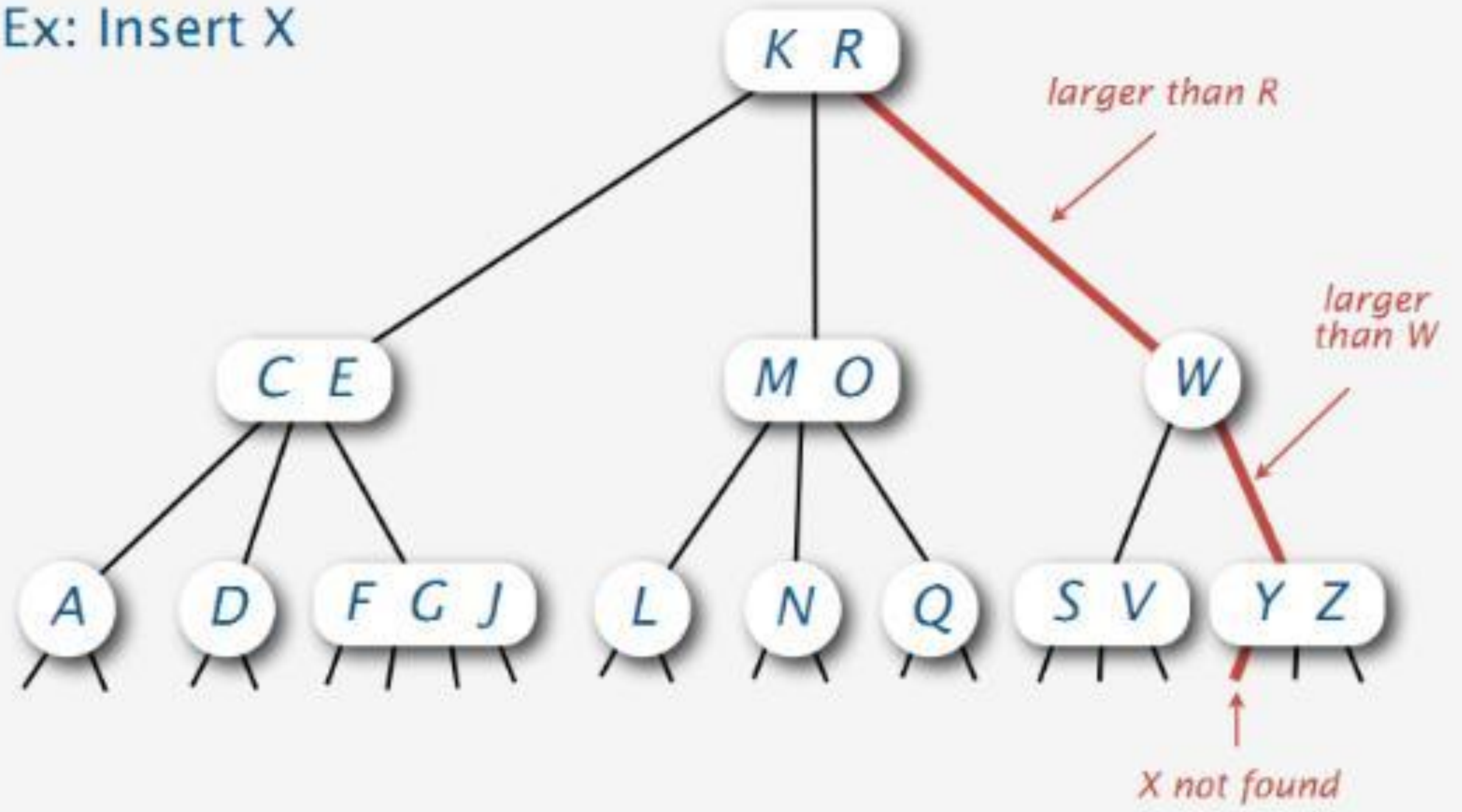
A 2–3–4 tree is a self-balancing data structure that is commonly used to implement dictionaries. The numbers mean a tree where every node with children (internal node) has either two, three, or four child nodes

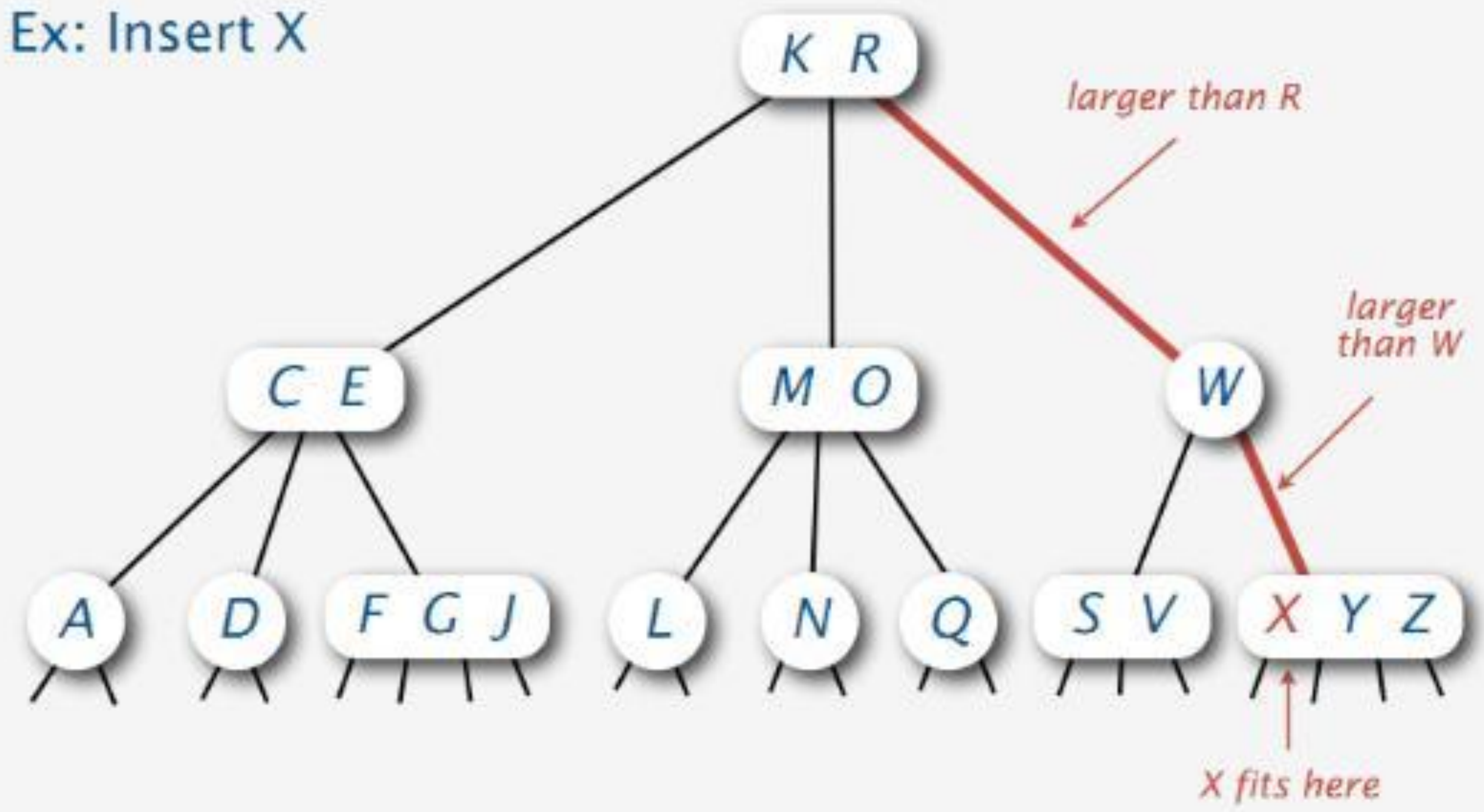
Properties

- Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively.
- All leaves are at the same depth (the bottom level).
- All data is kept in sorted order.

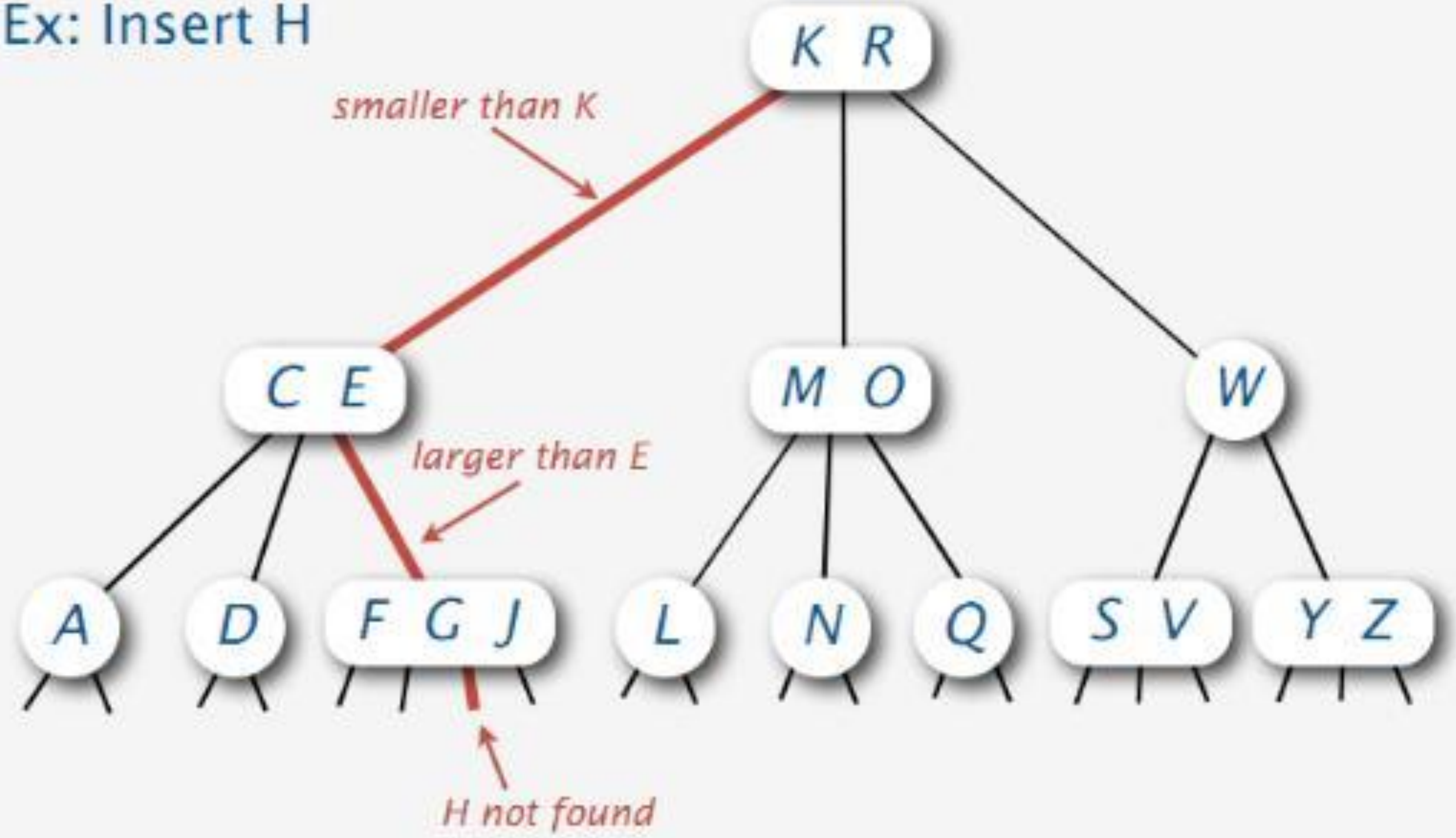
Ex: Insert X

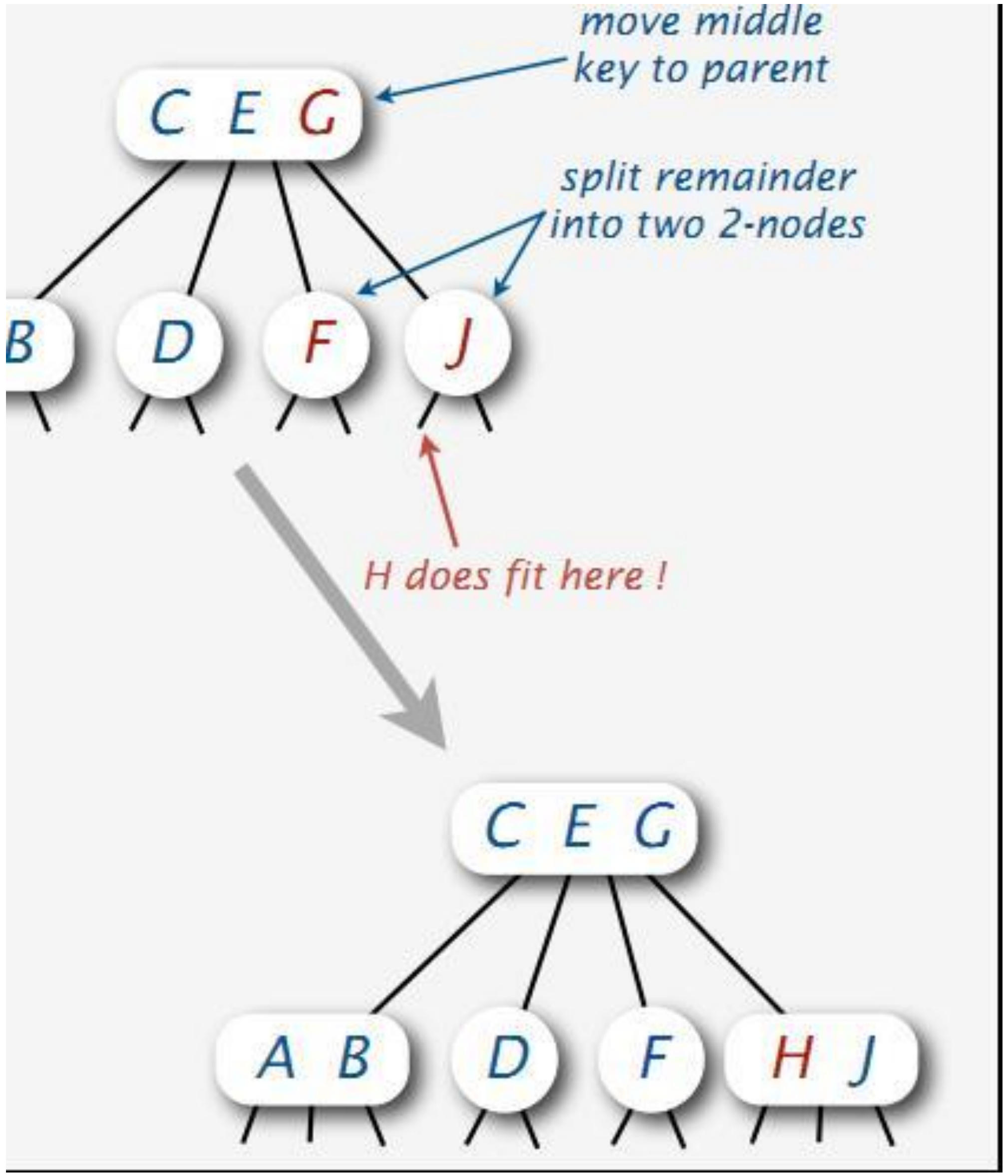

K R
C E
M O
W
A
D
F G J
L
N
Q
S V
Y Z
larger than R
larger than W
X not found

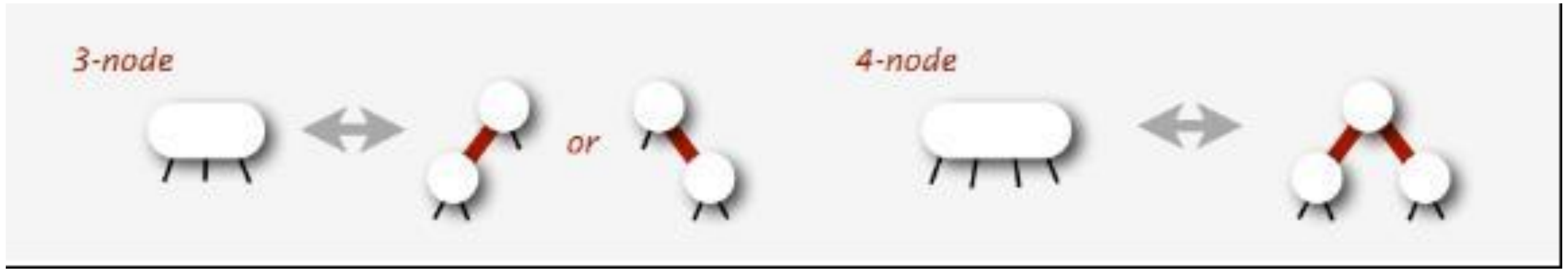
## Ex: Insert X


K R
larger than R
C E
M O
W
larger than W
A
D
F G J
L
N
Q
S V
X Y Z
X fits here

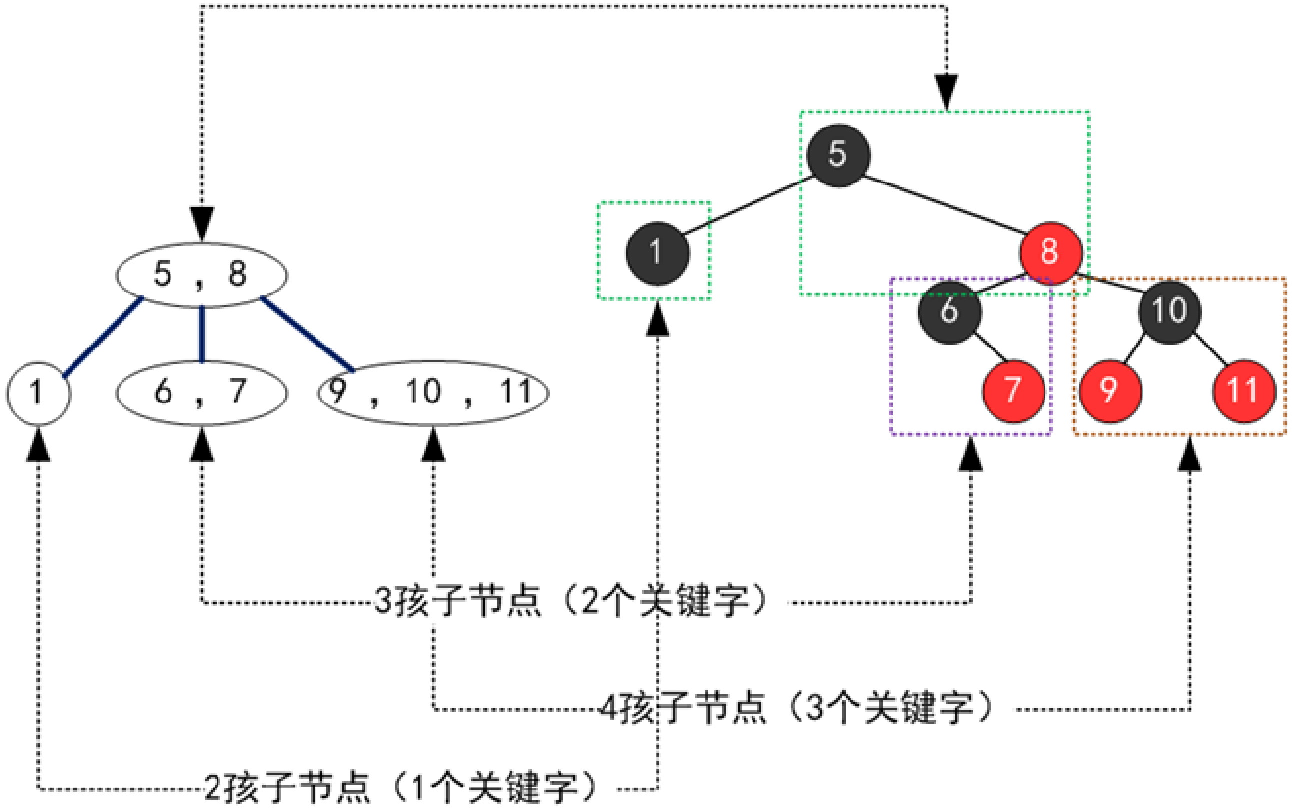
Ex: Insert H

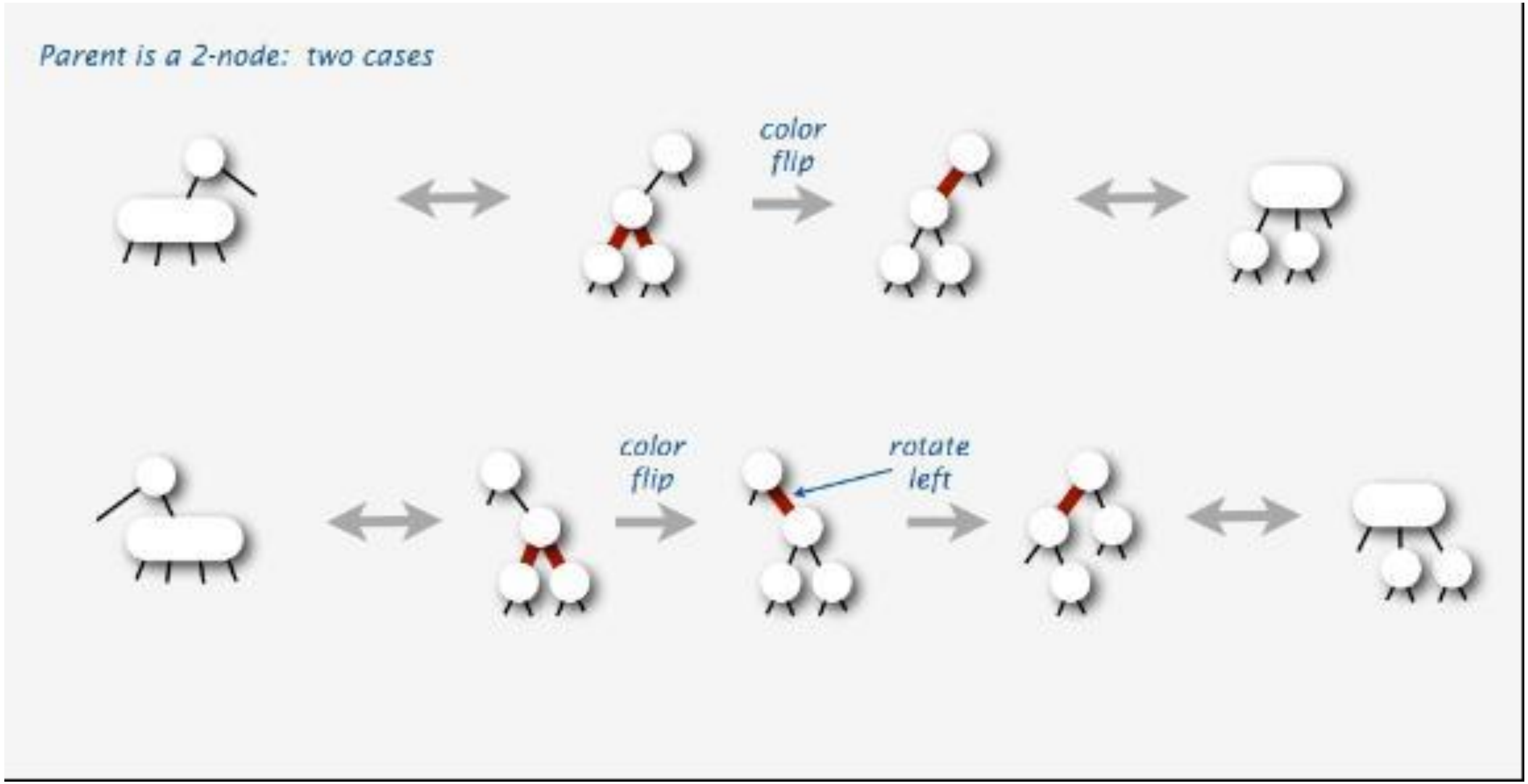

K R
smaller than K
C E
M O
W
larger than E
A
D
F G J
L
N
Q
S V
Y Z
H not found


move middle key to parent
C E G
split remainder into two 2-nodes
B
D
F
J
H does fit here !
C E G
A B
D
F
H J

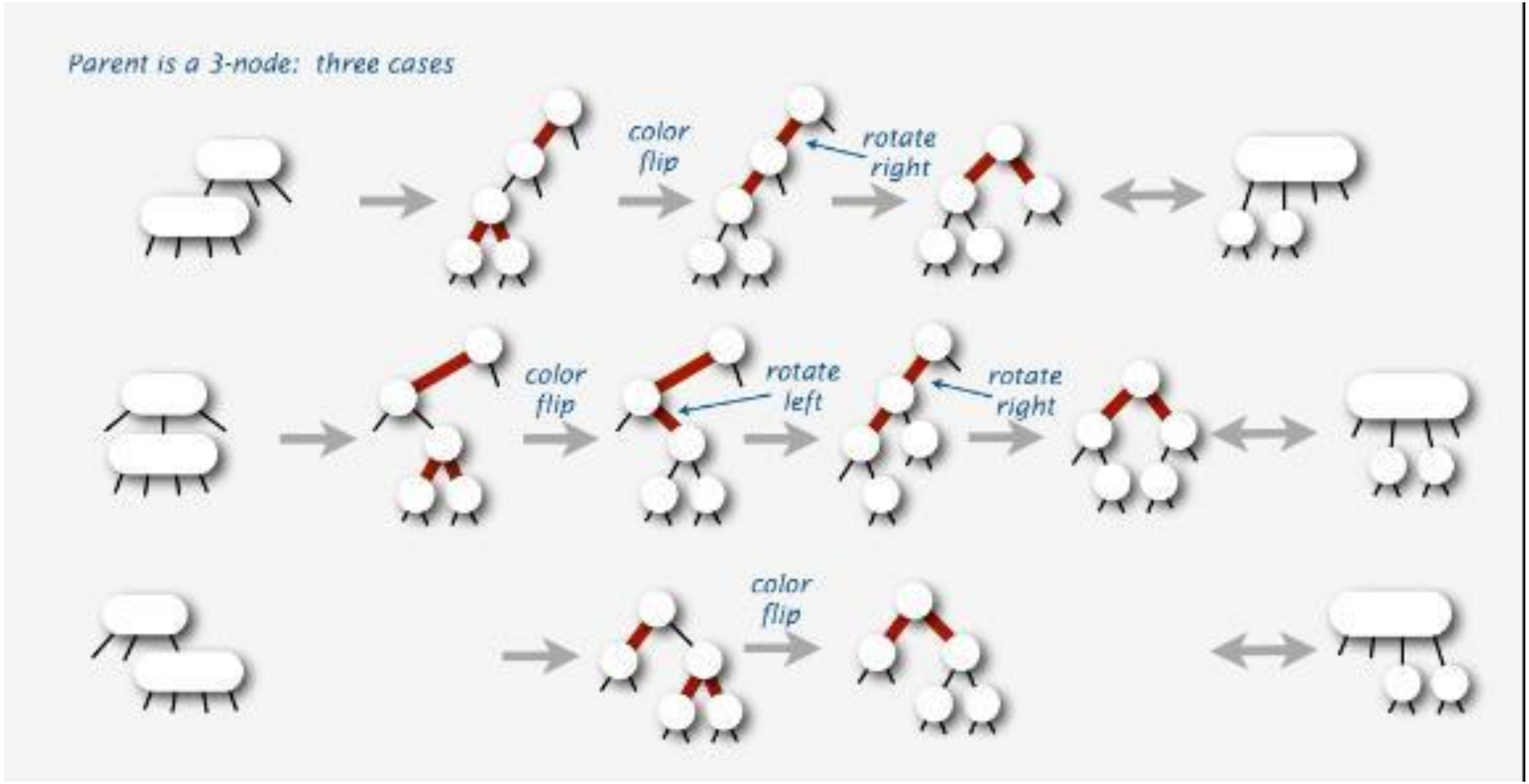


## 红黑树与2-3-4树的等价性


5 , 8
1
6 , 7
9 , 10 , 11
5
1
8
6
10
7
9
11
3孩子节点（2个关键字）
4孩子节点（3个关键字）
2孩子节点（1个关键字）


Parent is a 2-node: two cases
color flip
color flip
rotate left

插入操作的等价性


Parent is a 3-node: three cases
color flip
rotate right
color flip
rotate left
rotate right
color flip

插入操作的等价性

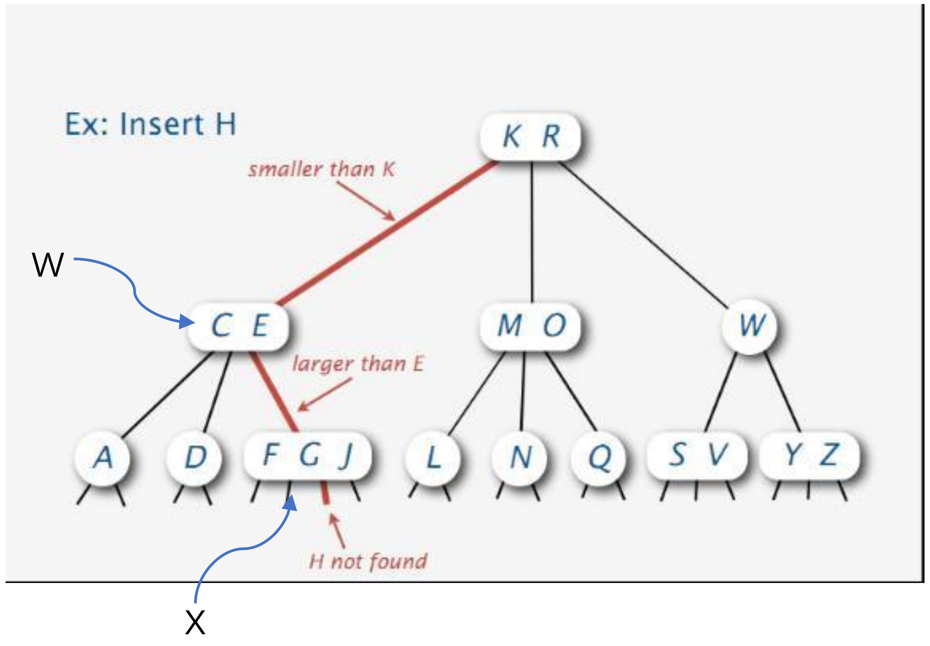
## Exercise 1

The ***join*** operation takes two dynamic sets S’ and S” and an element x such that for any $x' \in S'$and $x'' \in S''$ , we have x’.*key* < x.*key* < x”.*key*. It returns a set $S = S' \cup \{x\} \cup S''$ .

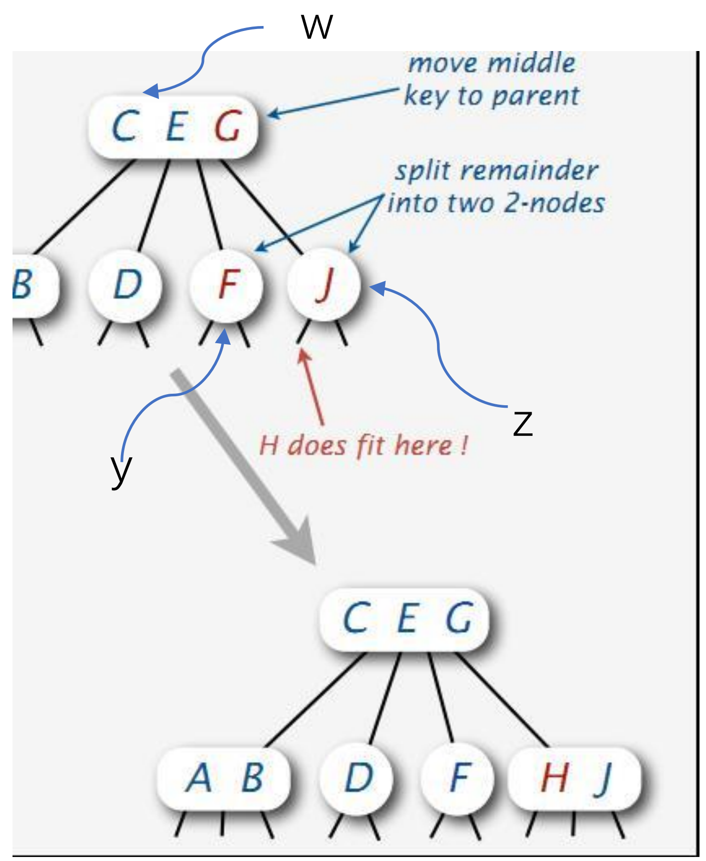
The ***split*** operation is like an “inverse” join: given a dynamic set S and an element $\mathrm{x} \in S$, it creates a set S’ that consists of all elements in S-{x} whose keys are less than x.*key* and a set S” that consists of all elements in S-{x} whose keys are greater than x.*key*.

In this problem, we investigate how to implement these operations on 2-3-4 trees. We assume for convenience that elements consist only of keys and that all key values are distinct.

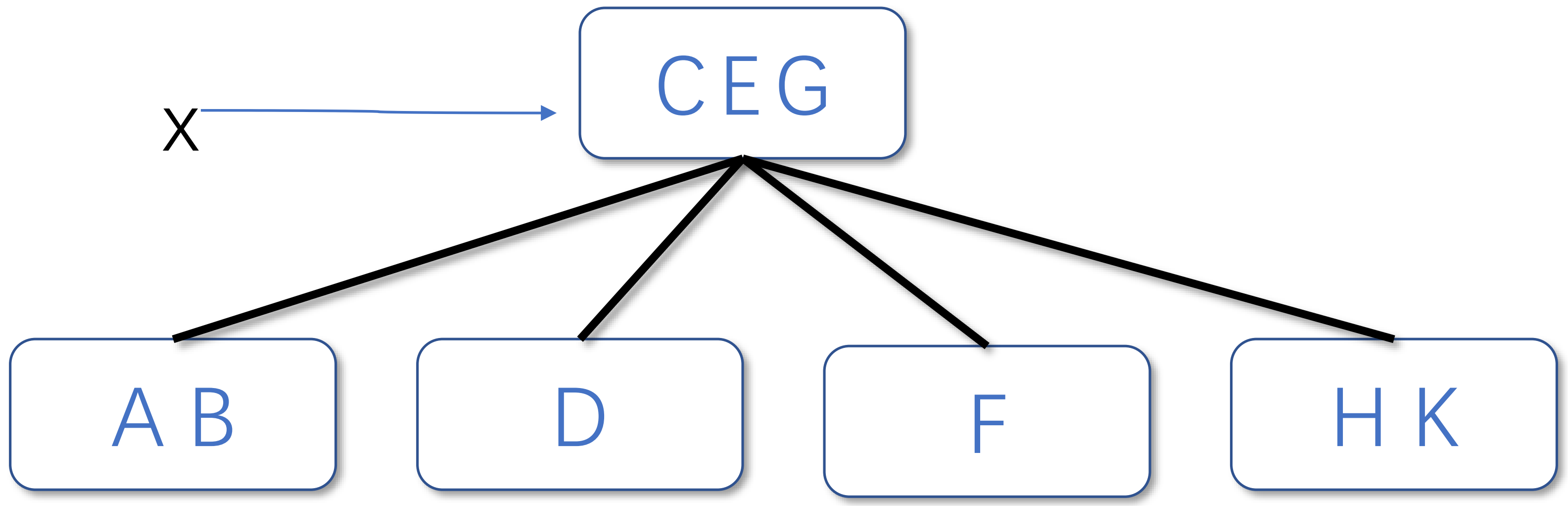
***a.*** Show how to maintain, for every node x of a 2-3-4 tree, the height of the subtree rooted at x as an attribute x.*height*. Make sure that your implementation does not affect the asymptotic running times of searching, insertion, and deletion.

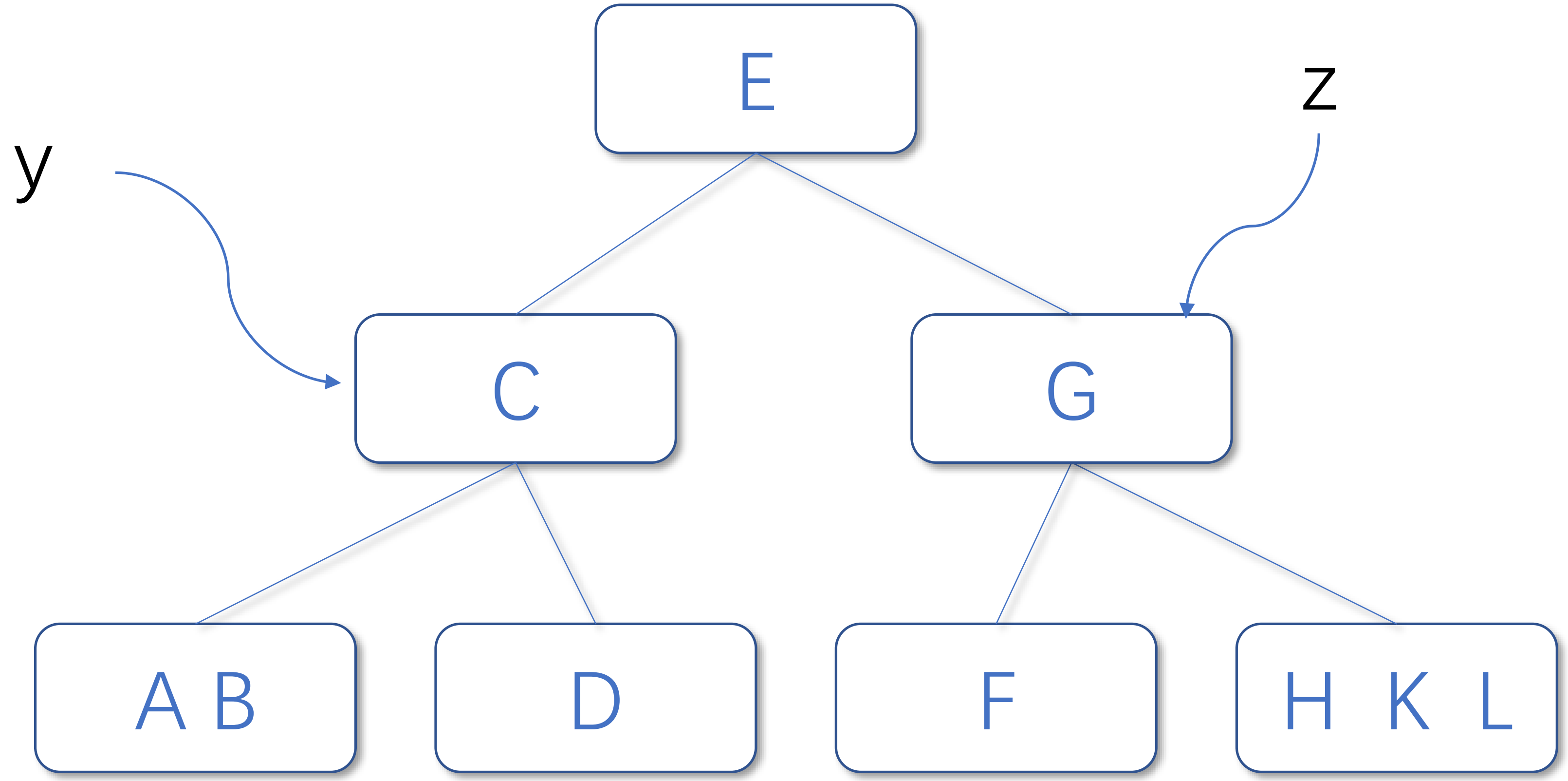


Suppose node x is split into nodes y and z, and the median of x is merged into node w. The height of w remains unchanged unless x was the root (in which case w.height = x.height + 1). The height of y and z will unchange.

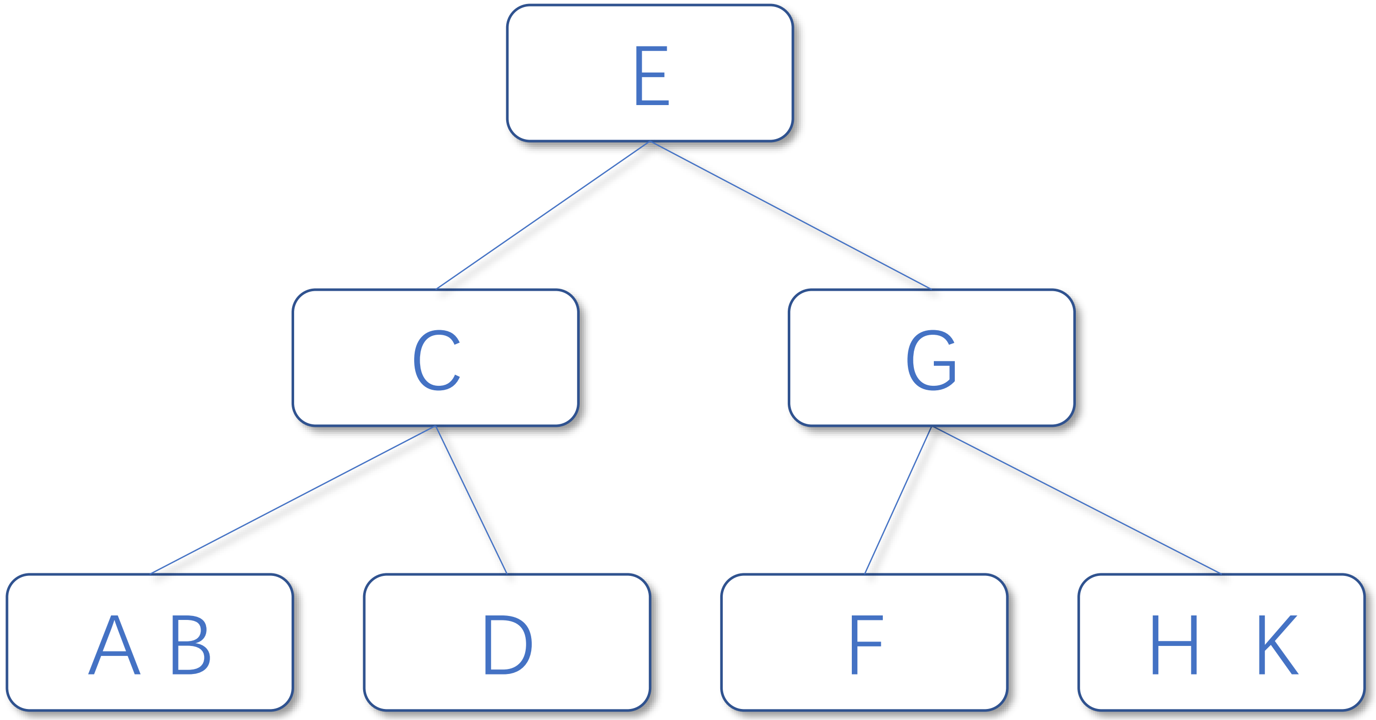


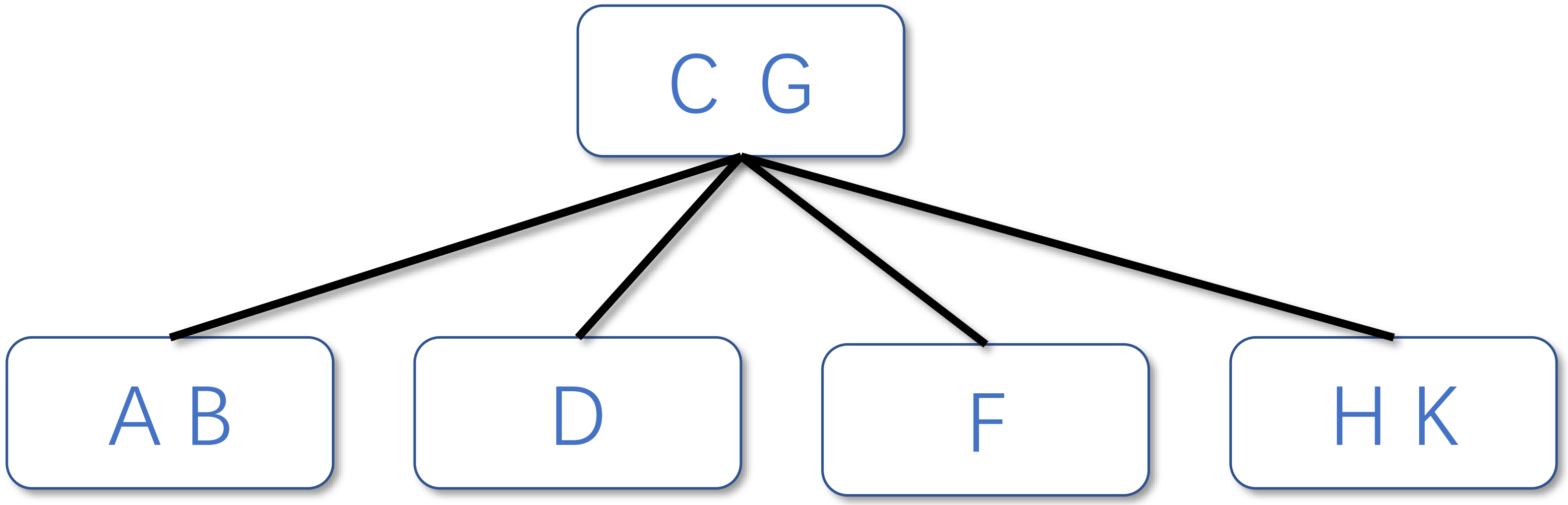
Suppose node x is split into nodes y and z, and the median of x is merged into node w. The height of w remains unchanged unless x was the root (in which case w.height = x.height + 1). The height of y and z will unchange.


x
C E G
A B
D
F
H K


E
z
y
C
G
A B
D
F
H K L

For deletion the situation is even simple. The only time the height changes is when the root has a single node and it is merged with its subtree nodes. In this case, we update the height of the new node to be the (old) height of the root minus 1.


E
C
G
A B
D
F
H K


C G
A B
D
F
H K

***b.*** Show how to implement the join operation. Given two 2-3-4 trees T’ and T” and a key k, the join operation should run in $O(1 + |h' - h"|)$ time, where h’ and h” are the heights of T’ and T”, respectively.

Without loss of generality, assume h’ ⩾ h”. We essentially wish to merge T” into T’ at a node of height h” using node x. To do this, find the node at height h’’+1 on the right spine of T’. Add x as a key to this node, and T” as the additional child. If it should happen that the node was already full, perform a split operation.