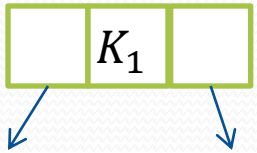
2-3 Search Trees

2-3 Search Trees

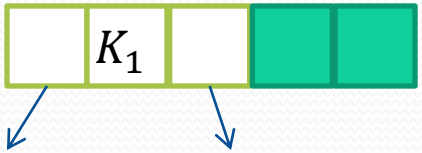
- Each node can contain 1 or 2 keys
- Each node has 2 or 3 children, hence 2-3 trees.
- The keys in the nodes are ordered from ***small to large***.
- All leaves are at the same (bottom most) level, meaning we always add at the bottom.

2-3 Search Tree Node

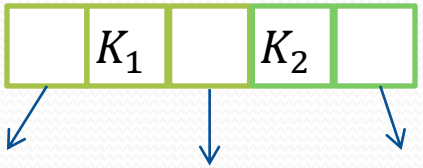
```
struct Node23
{
  Node23 *left, *middle, *right;
  Key key1, key2;
};
```



2-node (not showing empty)



2-node (showing empty)



3-node

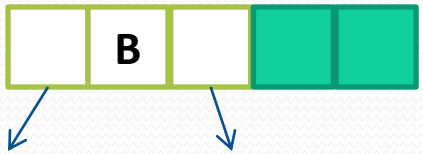
$K_1 < K_2$

Properties of 2-3 Search Trees

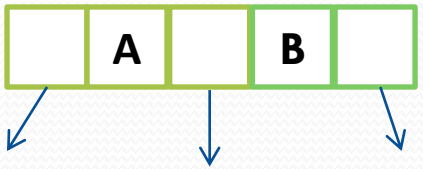
- 2-3 search trees guarantee to be balanced at all times.
- Searches are **O(lgN)**.
- The tree grows at the ***root***.
- Balance is maintained during insertion
  - Splitting nodes

Insertion

- If the node you insert is a 2-node, simply grow the node to a 3-node

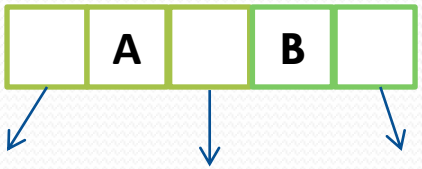


Inserting **'A'**

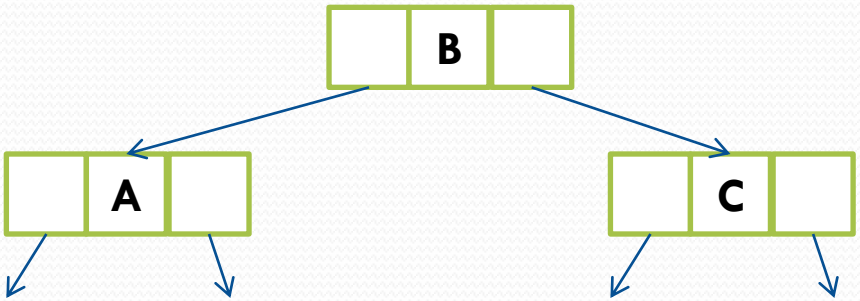


Insertion

- If the node you insert is a 3-node, we cannot grow the node more
  - We split it!



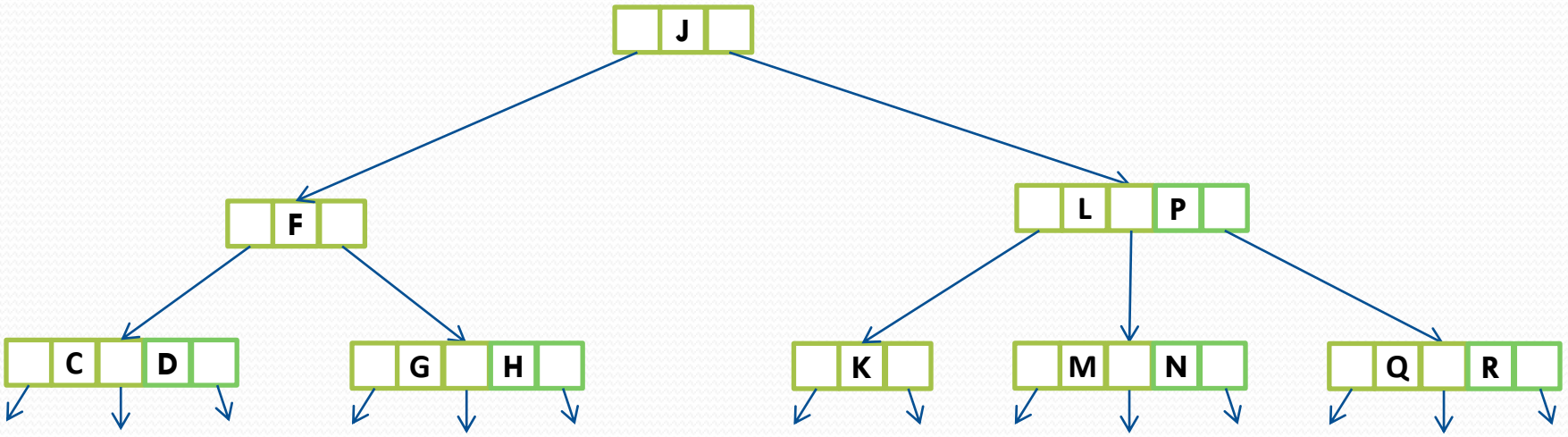
Inserting **'C'**



Insertion

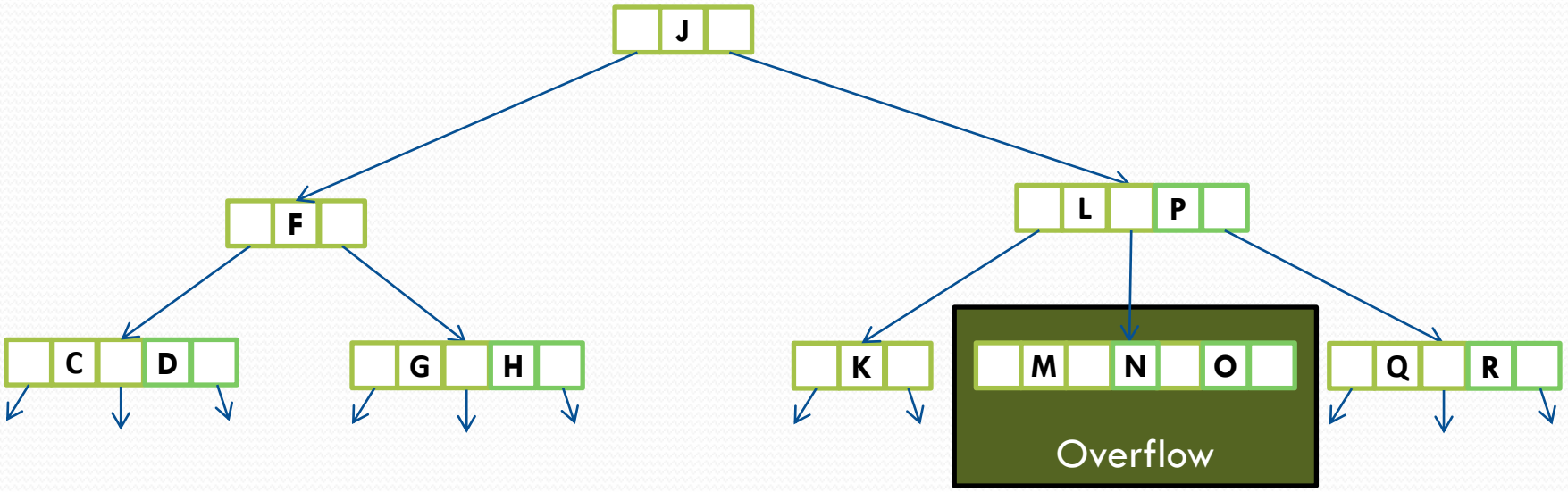
- Splitting this way is called “bottom-up” balancing
  - Insert the node at the bottom-most level at correct location.
  - If the node is a 3-node, split it and pass the middle key to the parent.
    - If the parent is also a 3-node, split the parent and pass the middle key up
      - Etc…
  - Eventually, the root will also be a 3-node and splitting it will *grow* the tree one level.

Example:



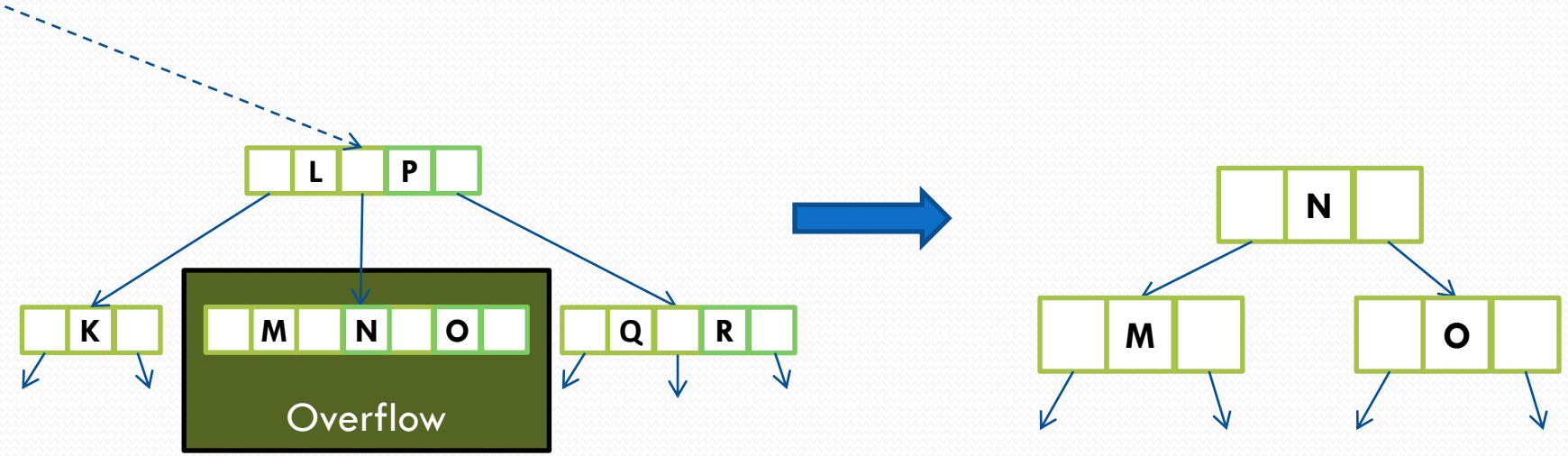
- Inserting **'O'** causes the 3-node **M,N** to overflow to **M,N,O**
- Split the node into two 2-nodes, **M and O.**
- Pass up the middle **N**

Example:



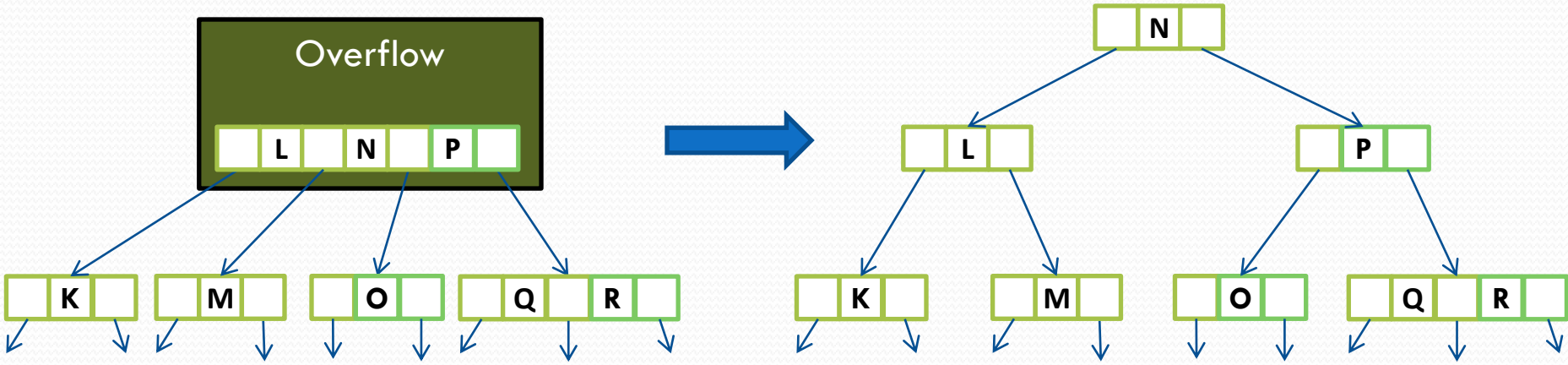
- Inserting **'O'** causes the 3-node **M,N** to overflow to **M,N,O**
- Split the node into two 2-nodes, **M and O.**
- Pass up the middle **N**

Example:



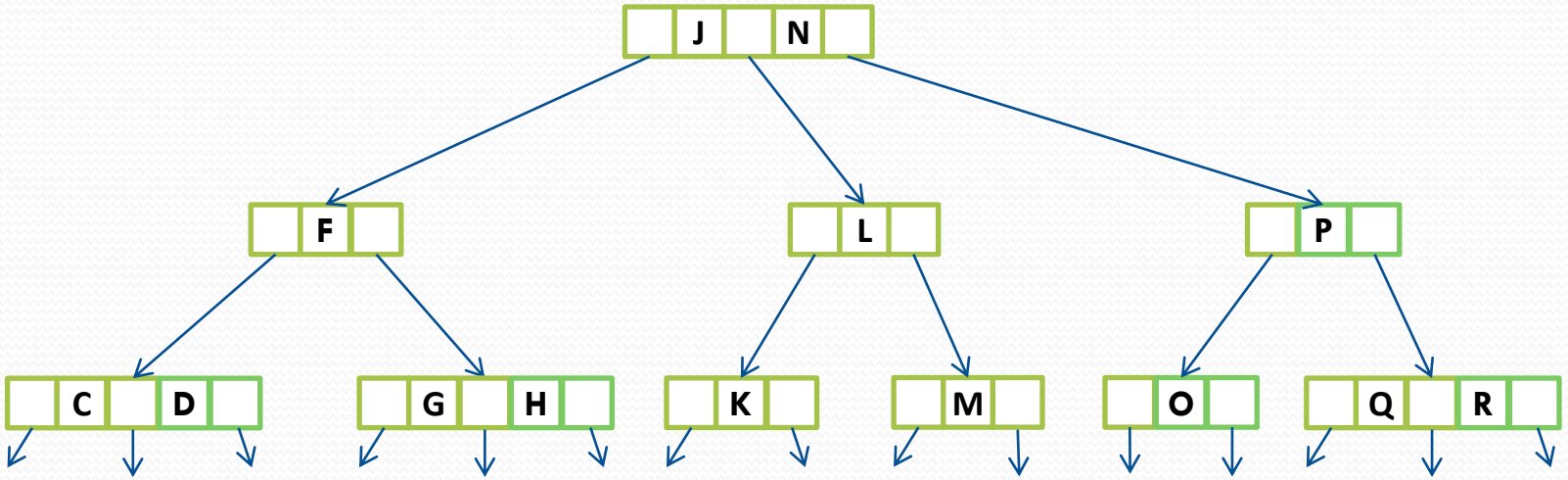
- Passing up **N** causes the 3-node **L,P** to overflow to **L,N,P**
- Split the node into two 2-nodes, **L and P**
- Pass up the middle, **N**

Example:



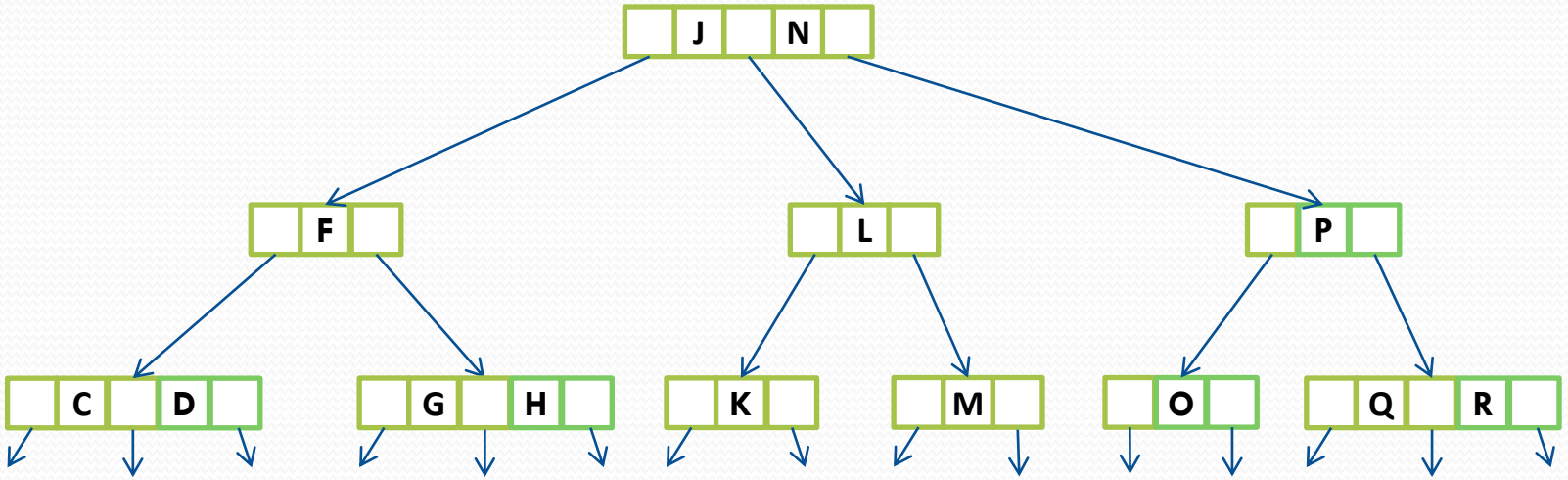
- Passing up **N** causes the 2-node **J** to become a 3-node **J,N**
- **No need to split, tree is balanced**

Example:



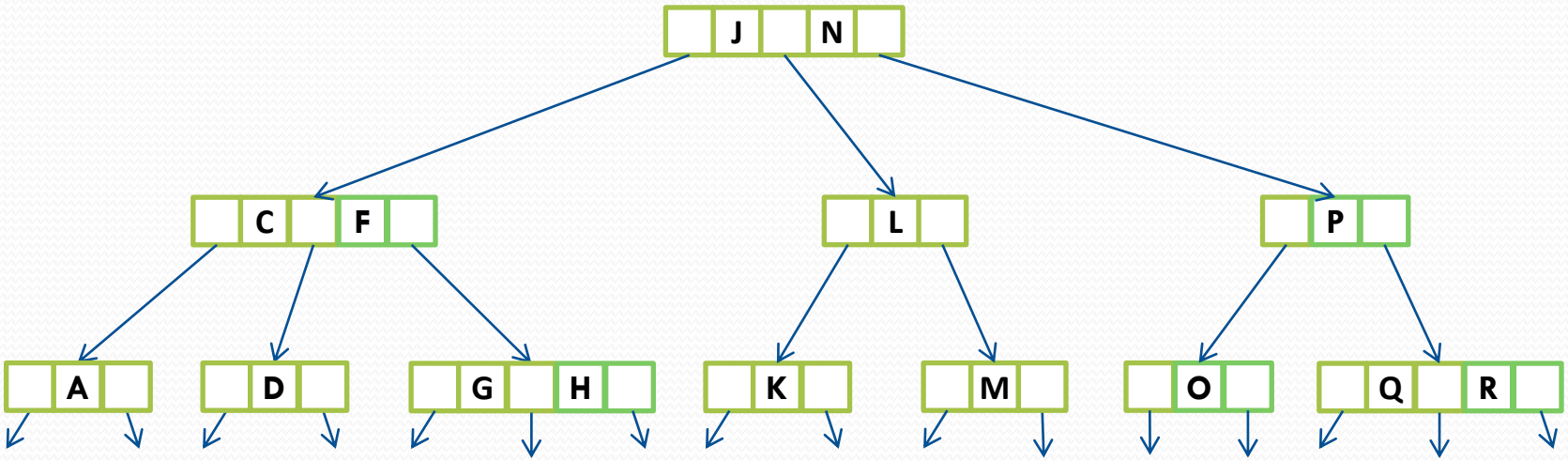
- Passing up **N** causes the 2-node **J** to become a 3-node **J,N**
- **No need to split, tree is balanced**

Exercise: Insert ‘A’ then ‘I’

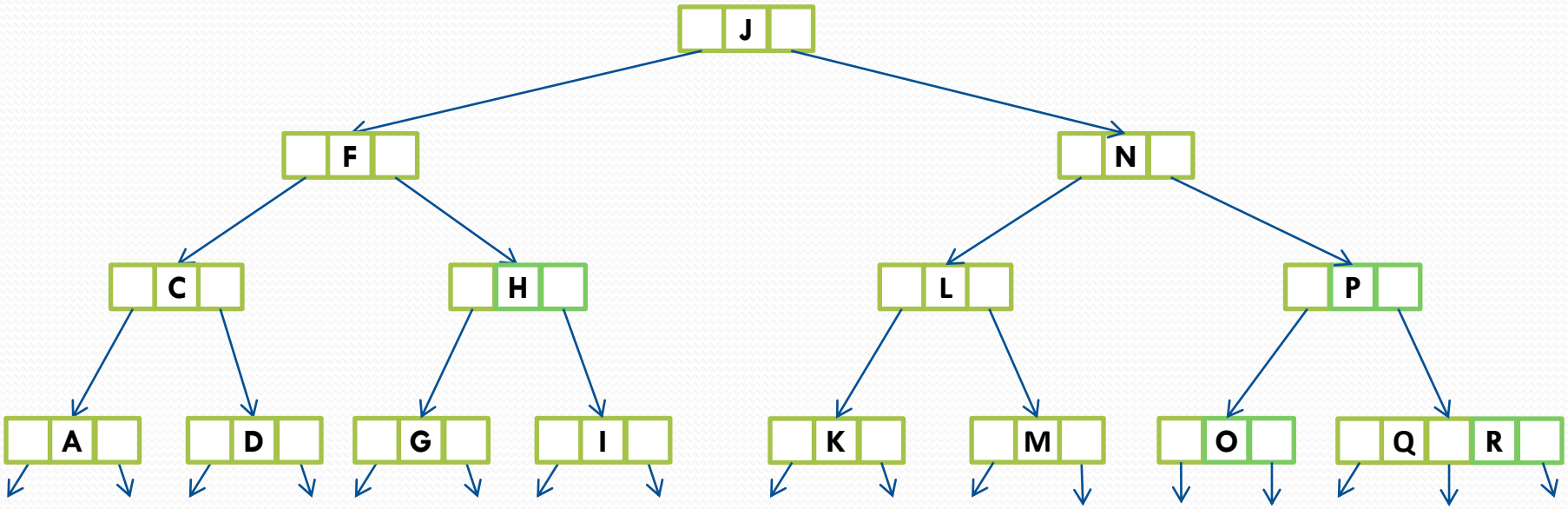


- Copy this 2-3 Tree on a piece of paper.
- **Draw the resulting tree after inserting ‘A’ and ‘I’**

Result of Inserting ‘A’



Result of Inserting ‘I’



Summary

- 2-3 Trees are always balanced
- Nodes are **ALWAYS** inserted at the bottom-most level
- Balance is maintained by splitting full nodes and *passing up* the middle node.
- This makes the tree's height increase by one only when the root is split.