2-3-4 Trees

Basic Properties

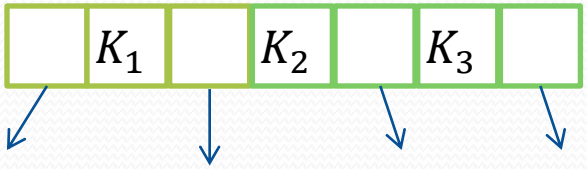
- Similar to 2-3 trees
- Nodes can contain 1, 2, or 3 keys.
- Nodes can has 2, 3, *4* children, hence *2-3-4 tree*.
  - Each can have <u>at most</u> *4* children.
- Similarly to 2-3 trees, 2-3-4 trees are guaranteed to be always balanced.
- Balancing algorithm also relies on Splitting nodes
- Number of splits in the worst-case is **O(lg N)**
  - When is the worst-case?
- Average Number of splits is very few

Balancing Algorithm

- Balancing also occurs on insertion.
- Modifying the algorithms for balancing can produce better efficiency
- Previously, with 2-3 Trees, we have seen ***bottom-up*** balancing.
- We will see ***top-down*** balancing
  - As you go down the tree to insert a node, split any *full* node.
  - **A *full* node is a 4-node.**

2-3-4 Node:

```
struct Node234
{
  Node23 *left, *midleft, *midright, *right;
  Key key1, key2, key3;
};
```

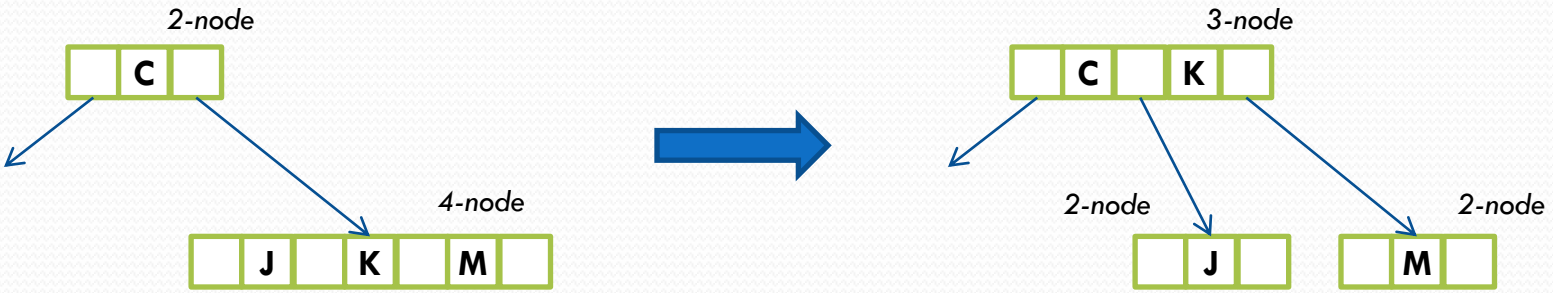


3-node

$$K_1 < K_2 < K_3$$

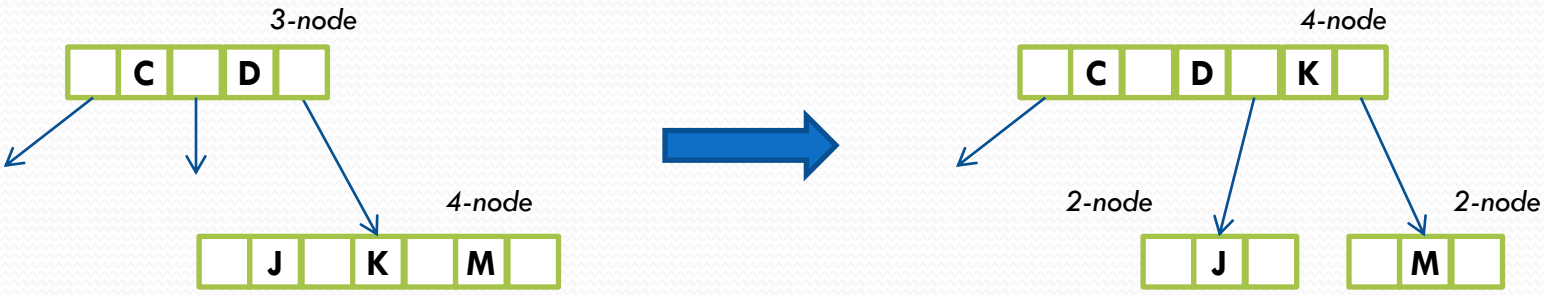
Splitting a 2-3-4 node:

- A 2-node attached to a 4-node becomes a 3-node attached to two 2-nodes



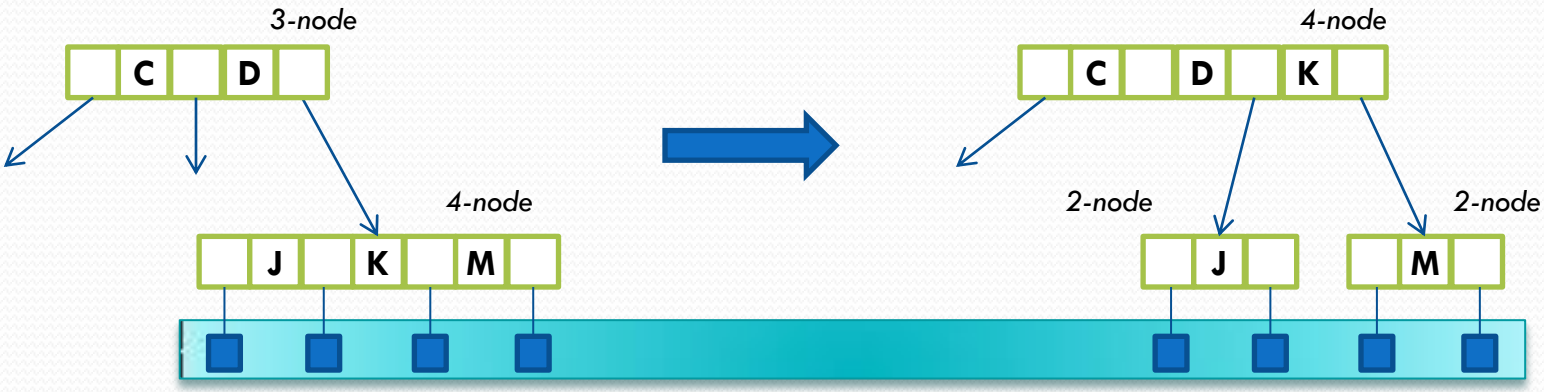
Splitting a 2-3-4 node:

- A 3-node attached to a 4-node becomes a 4-node attached to two 2-nodes:



Advantage of splitting 2-3-4 Trees:

- Splitting a node is “cleaner”.
- Splitting a 4-node into two 2-nodes preserves the number of child links.
- Changes do not have to be propagated. Change remains local to split.



Top-Down Balancing

- ***Split nodes on the way down.***
  - Guarantees that each node we pass through is not a 4-node.
  - When we reach the bottom, we will *not* on a 4-node (think about it)
- This way, we only traverse the tree once, when inserting/balancing.
- After each insertion, check if the root is a 4-node
  - If it is split it directly. This will avoid to do it at next insertion.
  - Splitting the root is the only way to grow the tree.

Exercise: "DAFTPUNKERS"

- Insert the above character sequence in a 2-3-4 Tree using the *top-down* method.