B-trees

Robert Horvick
SOFTWARE ENGINEER
@bubbafat www.roberthorvick.com

Overview

B-tree Overview

- Minimal degree
- Height

Searching

Adding

- Splitting Nodes

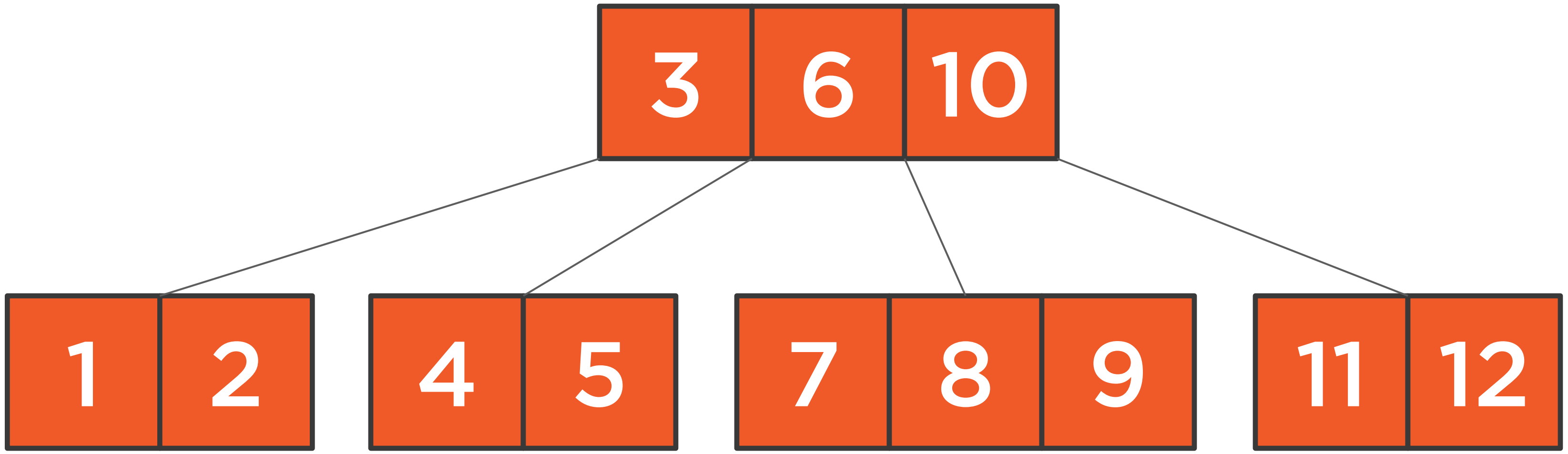
Removing

- Push down
- Rotation

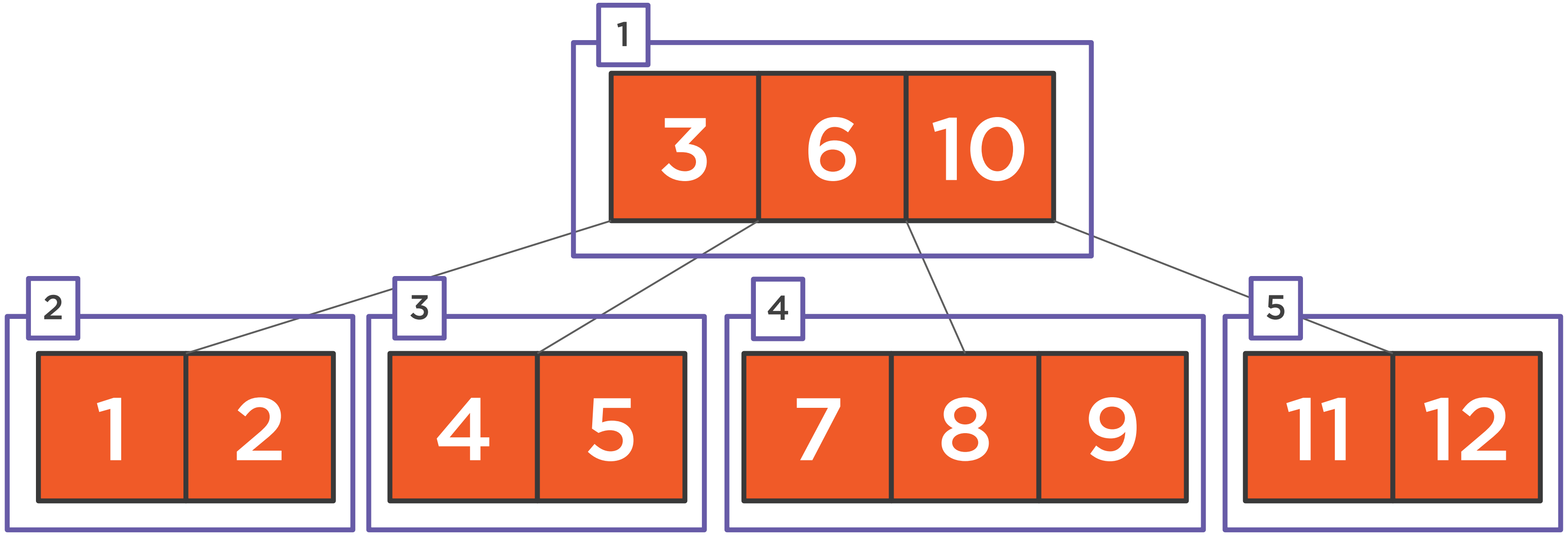
B-tree

A sorted, balanced, tree structure typically used to access data on slow mediums such as disk or tape drives.

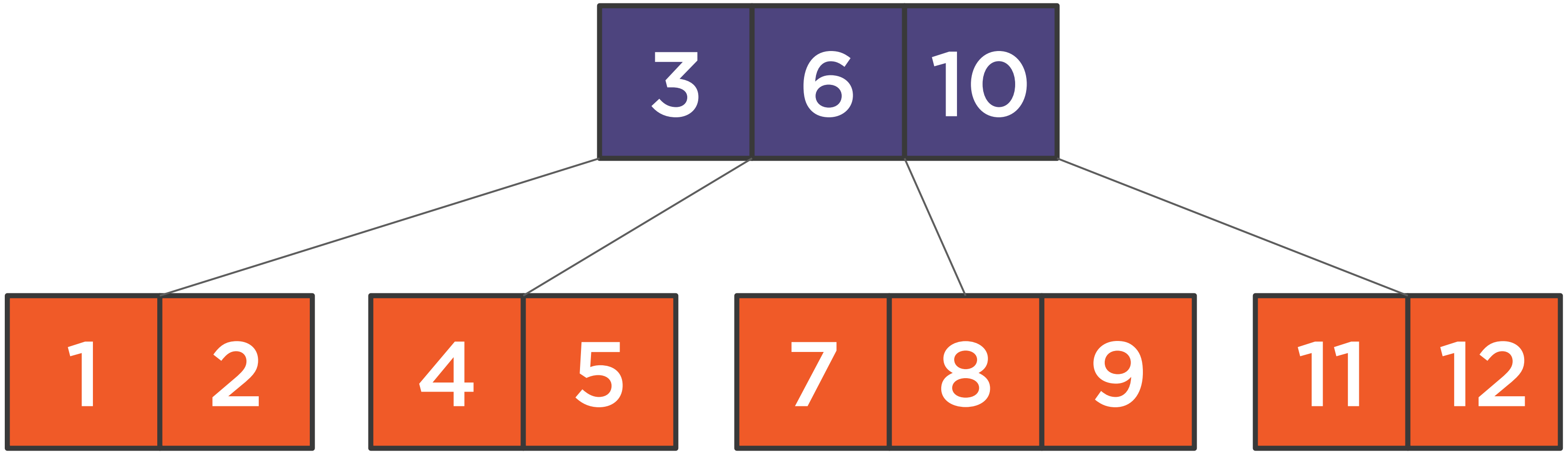
B-tree Properties



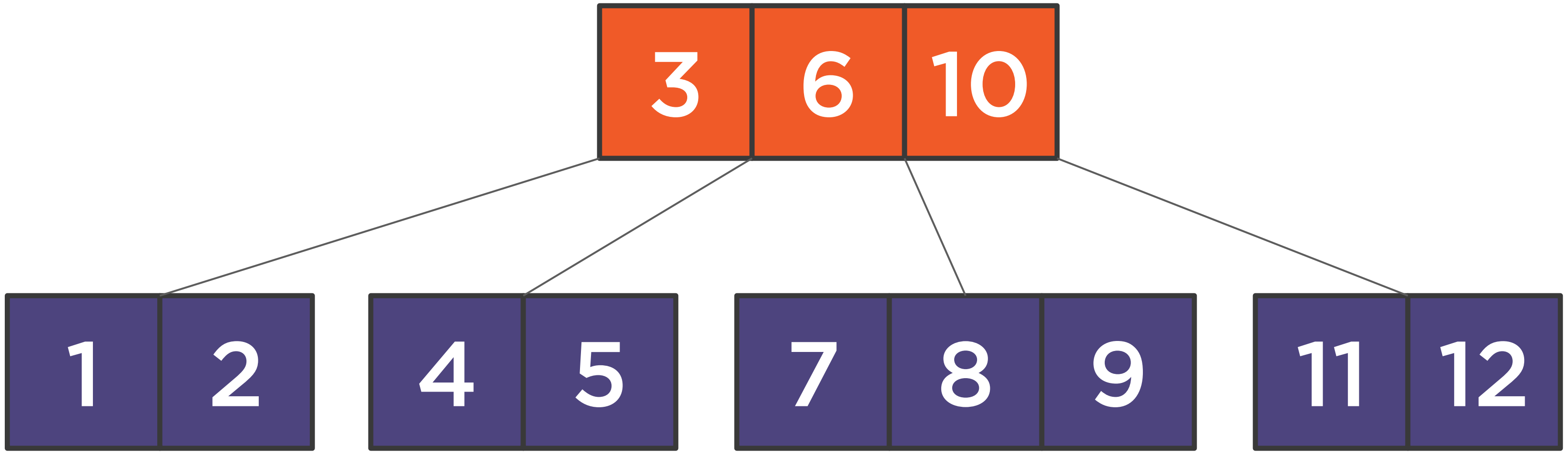
B-tree Properties



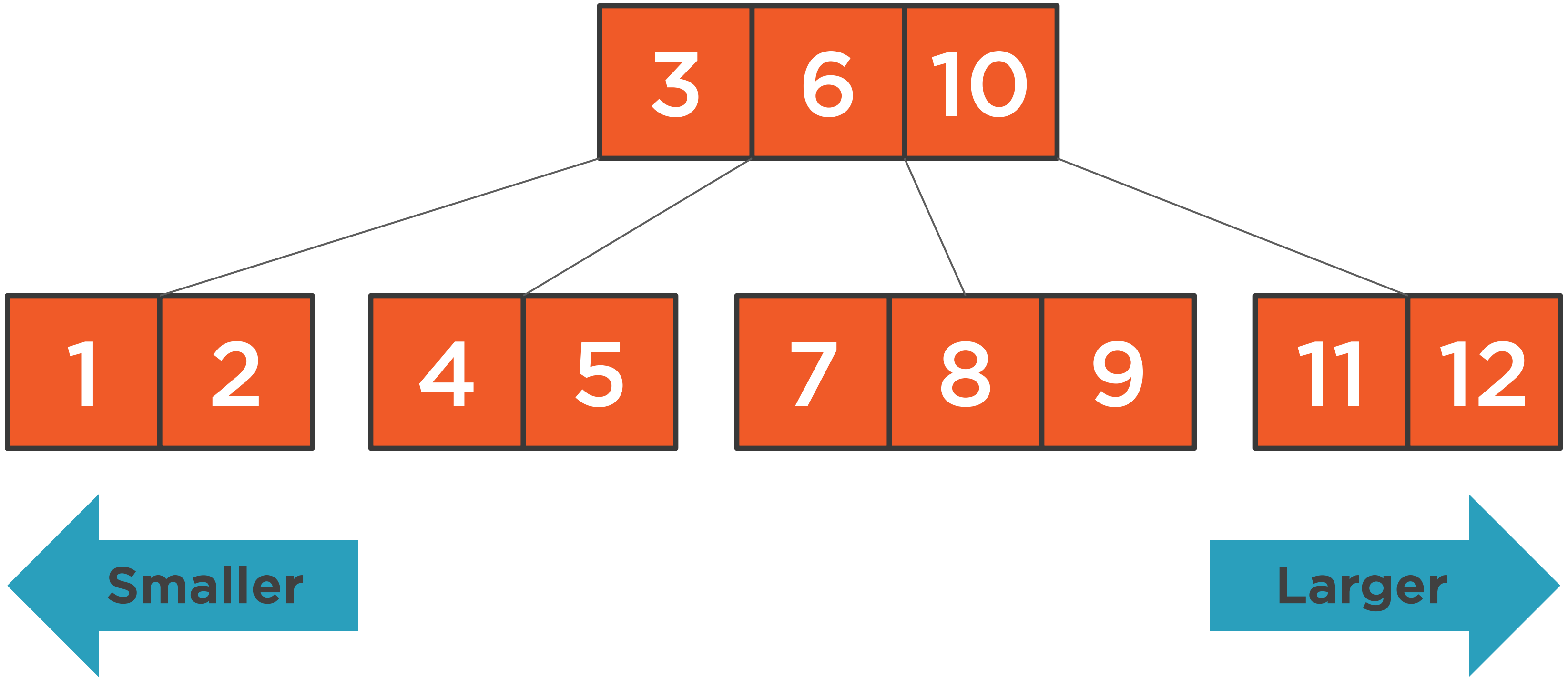
B-tree Properties



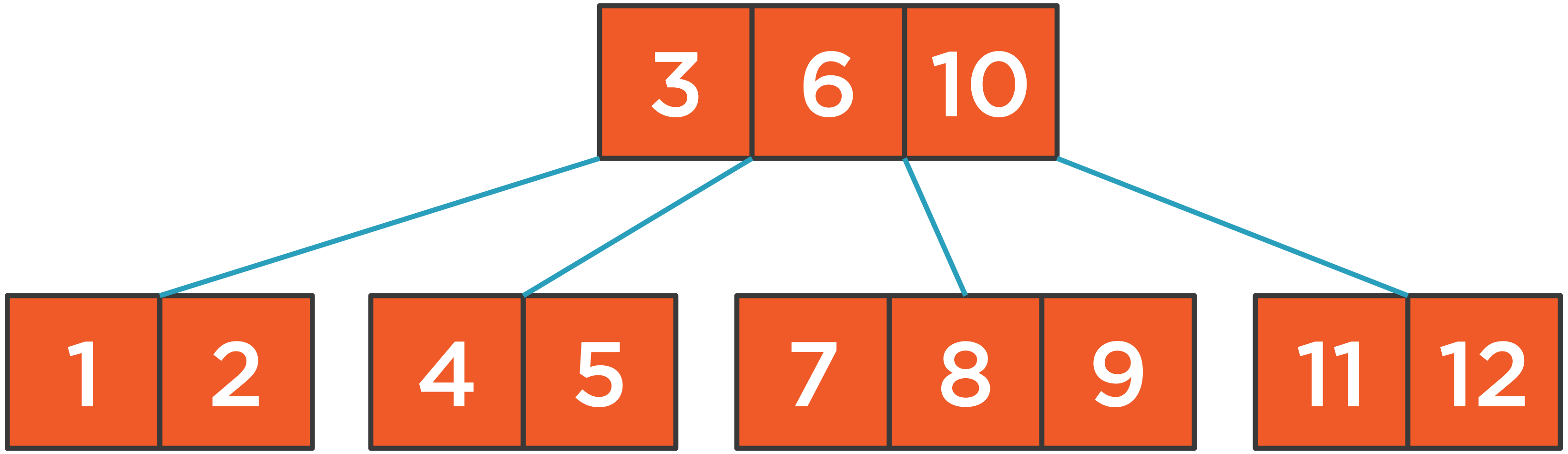
B-tree Properties



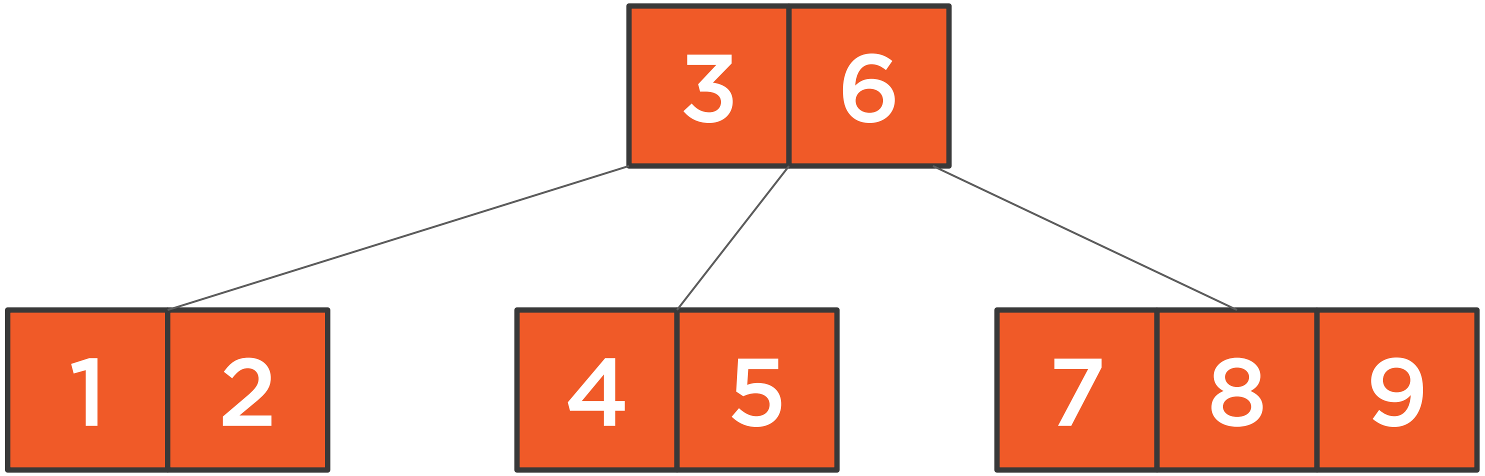
B-tree Properties



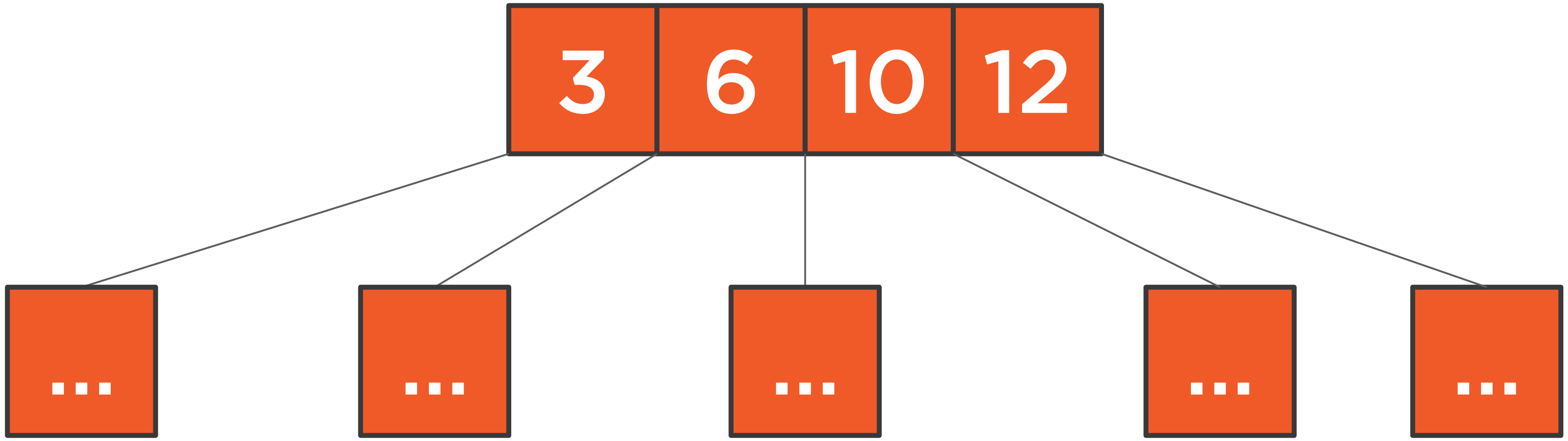
B-tree Properties



B-tree Properties



B-tree Properties



Nodes will always have one more child than values

Minimal Degree

The minimum number of children that every non-root node must have. Represented as the variable “T”.

Minimal Degree

T	**Each non-root must contain at least T (minimal degree) children**
T-1	**Every non-root node will have at least T-1 values**
2T	**Each node in the B-tree will contain a maximum of 2*T children**
2T-1	**Every node will have at most 2*T-1 values**

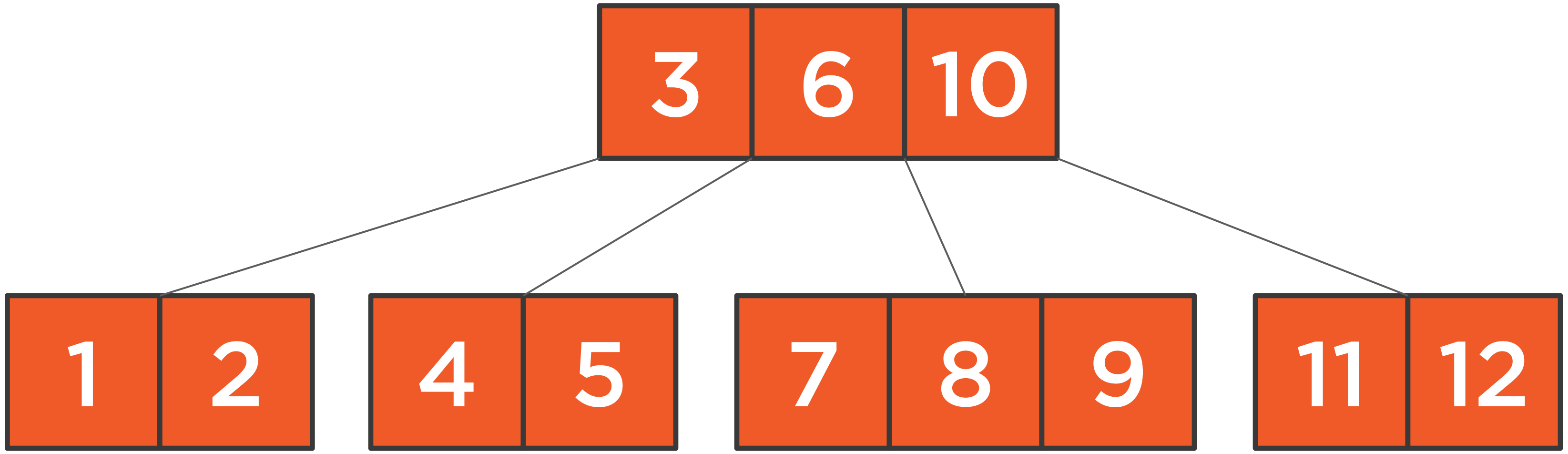
Minimal Node

A non-root node that has T-1 values and T children.

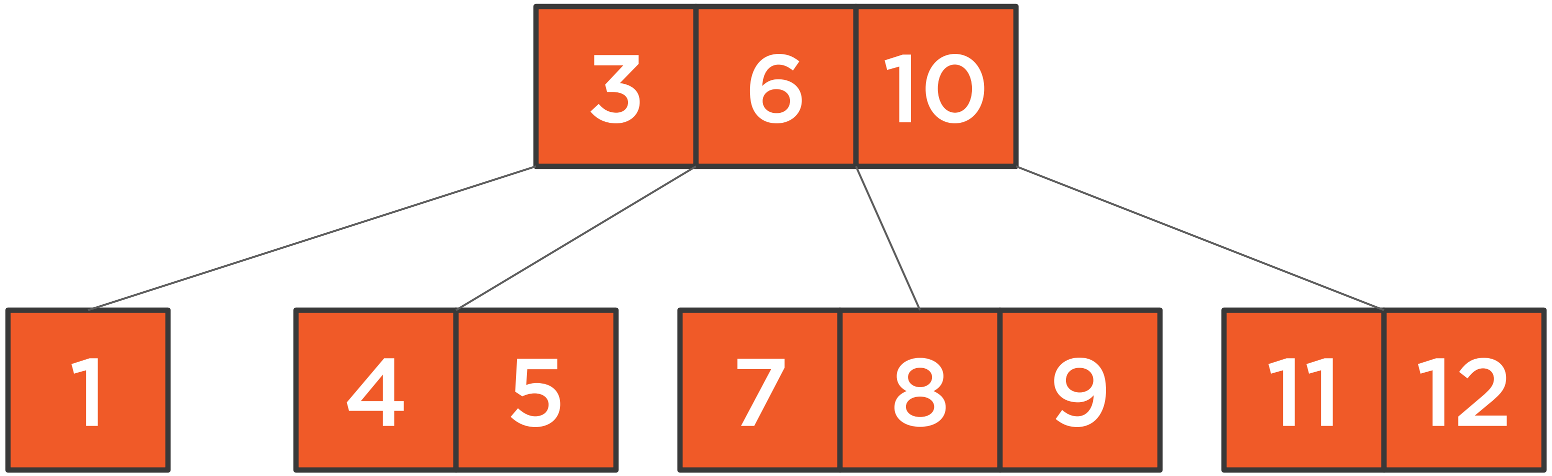
Full Node

A node that has 2*T-1 values and 2*T children.

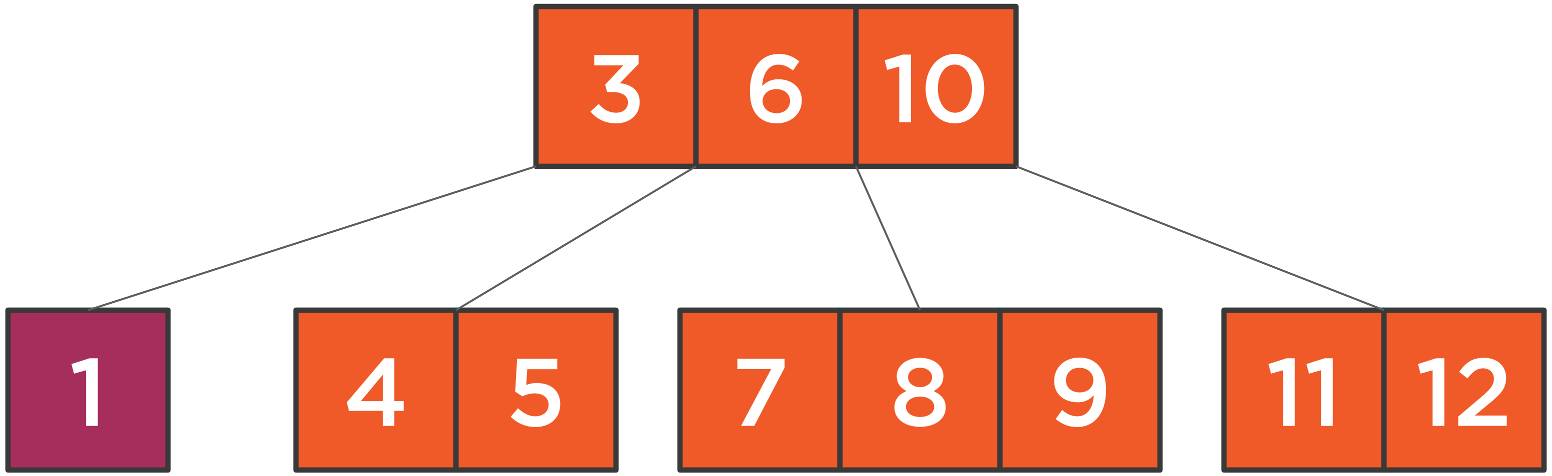
T = 3 (Valid)



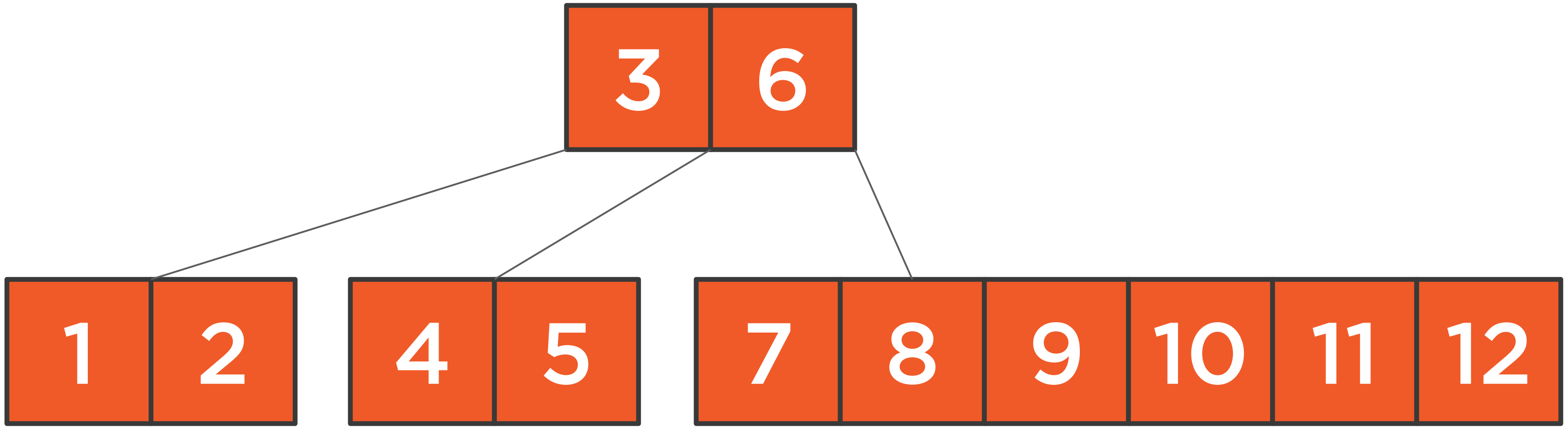
T = 3 (Invalid)



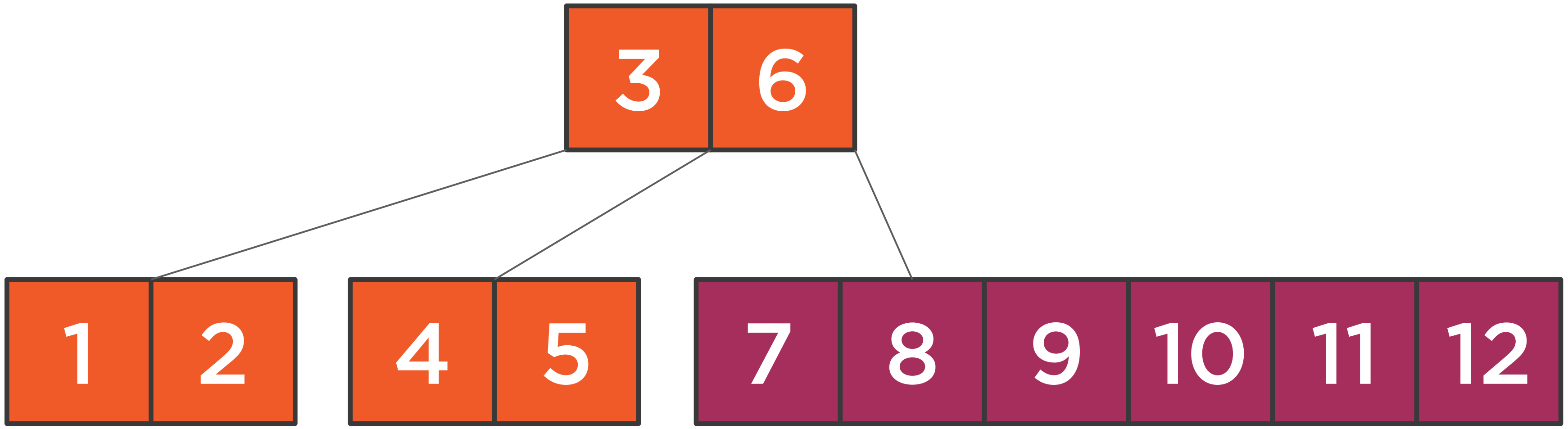
T = 3 (Invalid)



T = 3 (Invalid)



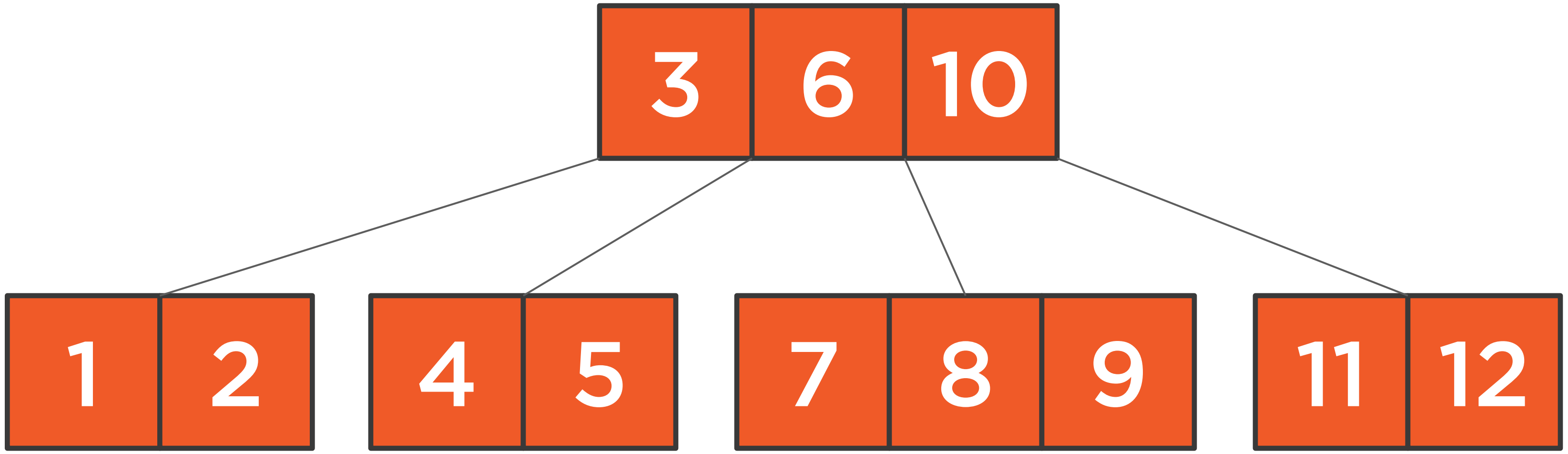
T = 3 (Invalid)



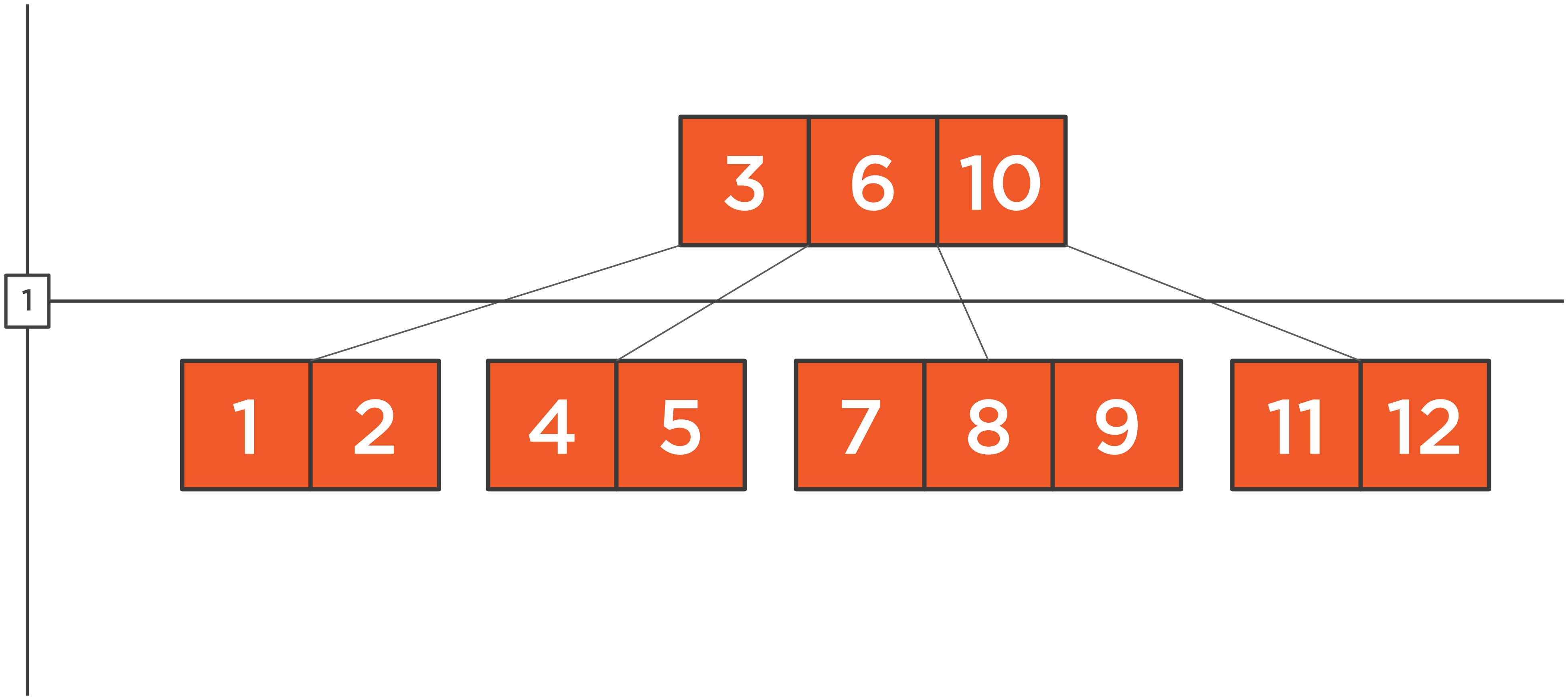
Height

The number of edges between the root and the leaf nodes. All B-tree leaf nodes must have the same height.

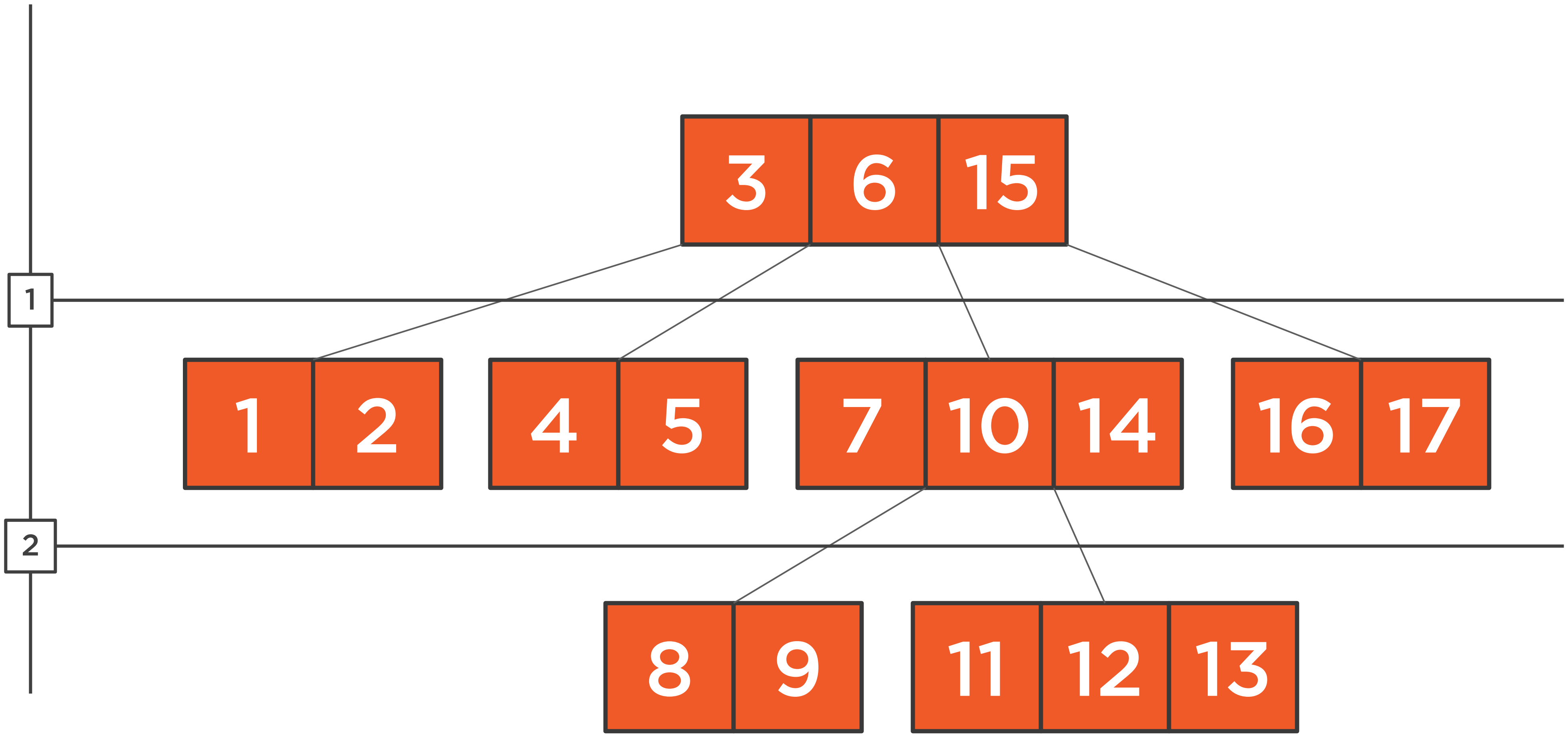
Height (Valid)



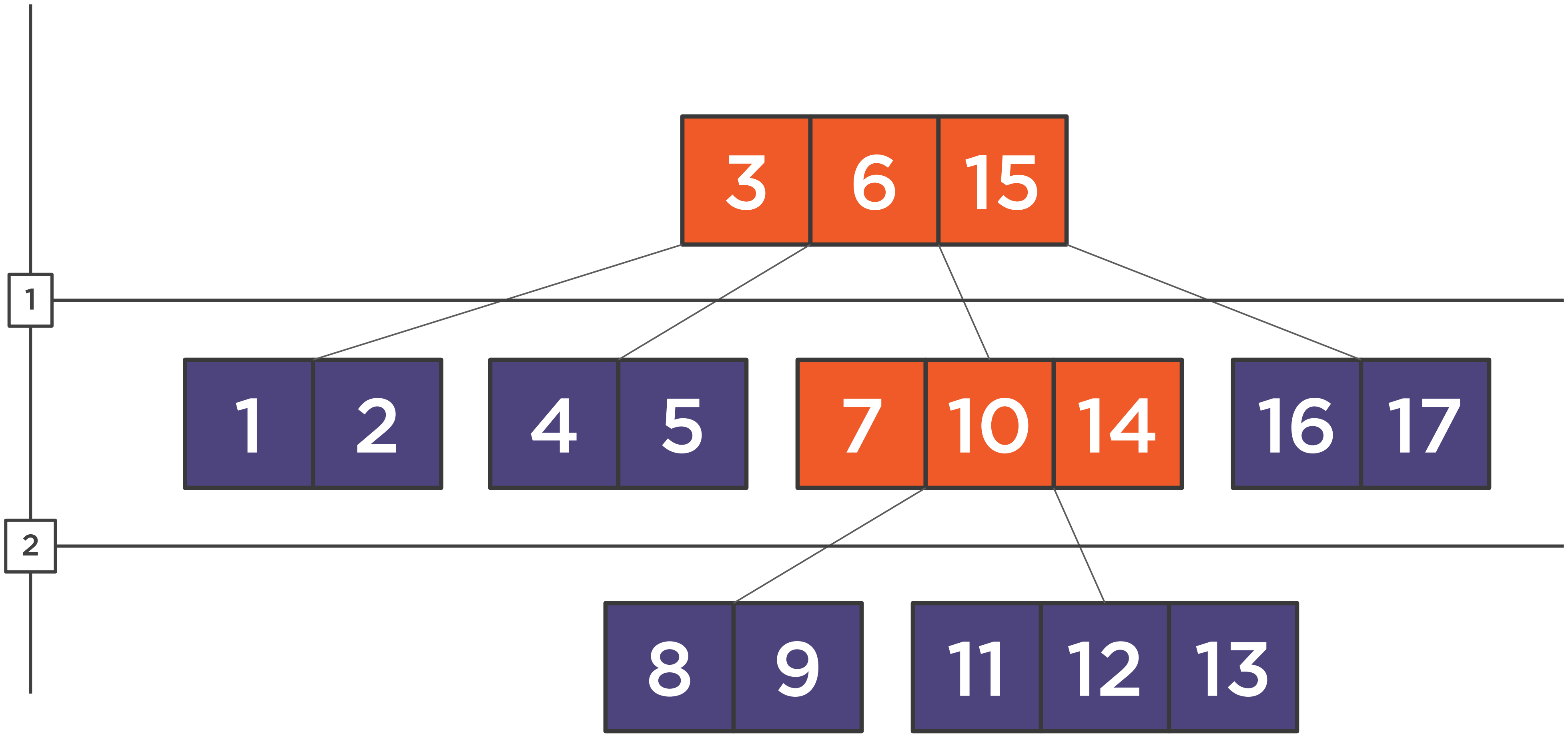
Height (Valid)



Height (Invalid)

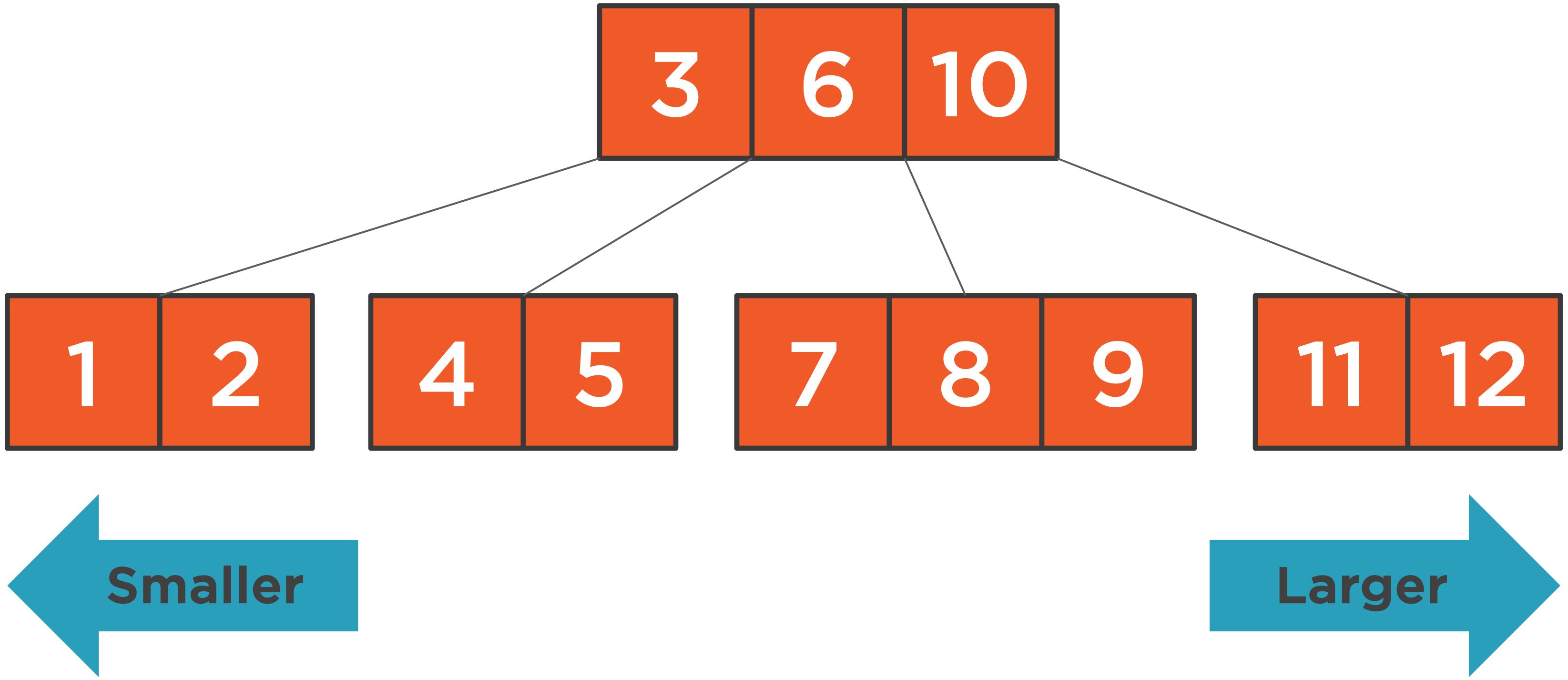


Height (Invalid)

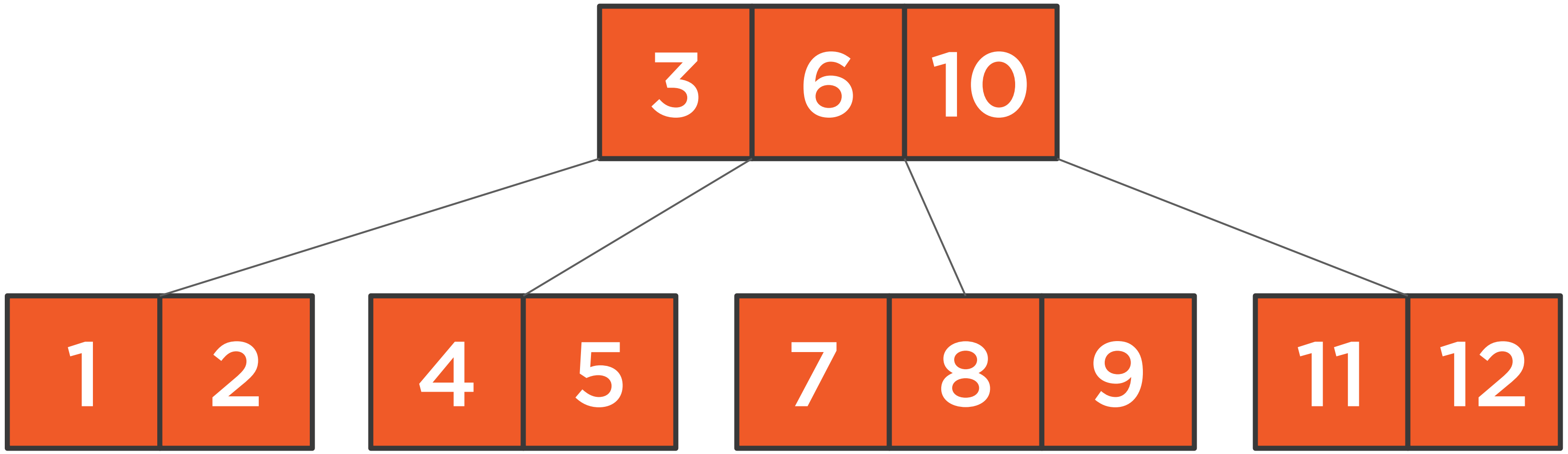


Searching

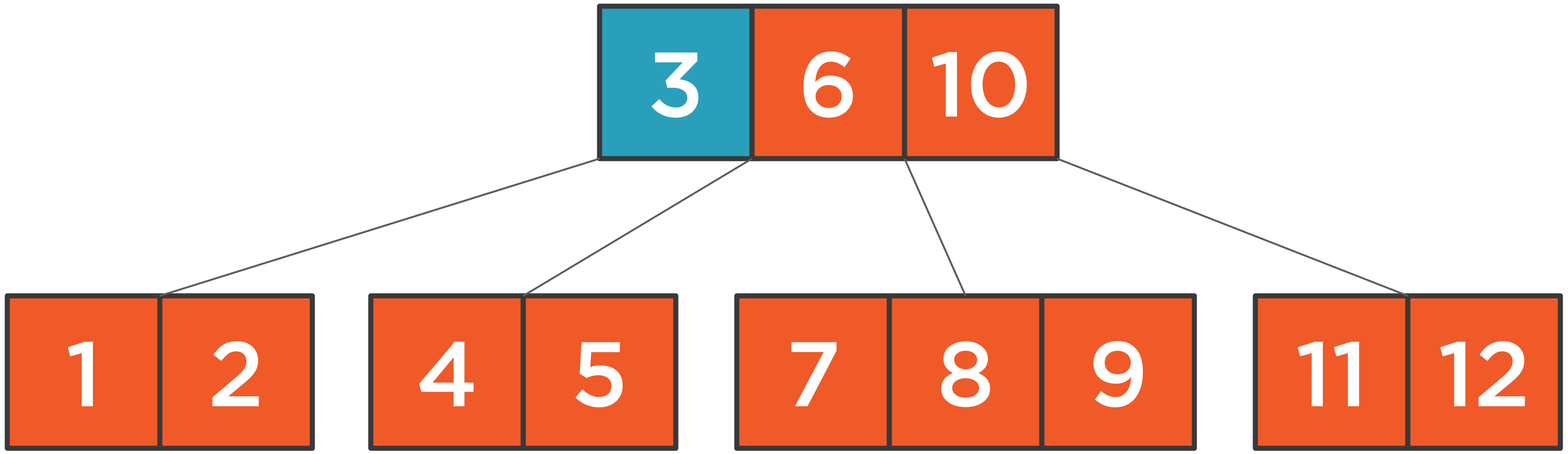
Searching



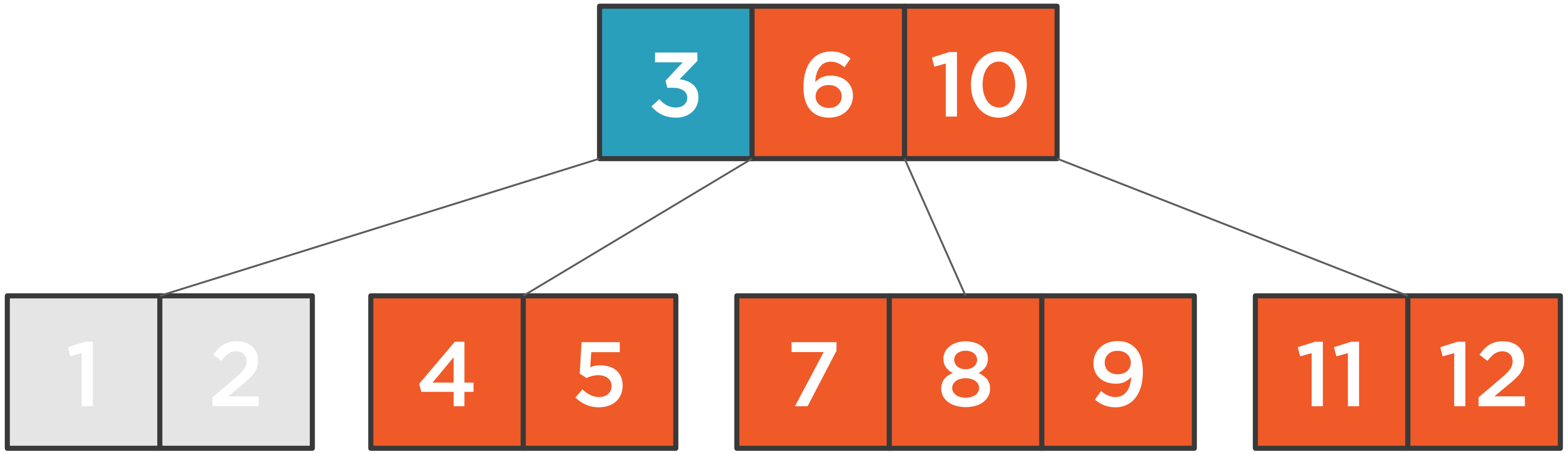
Searching



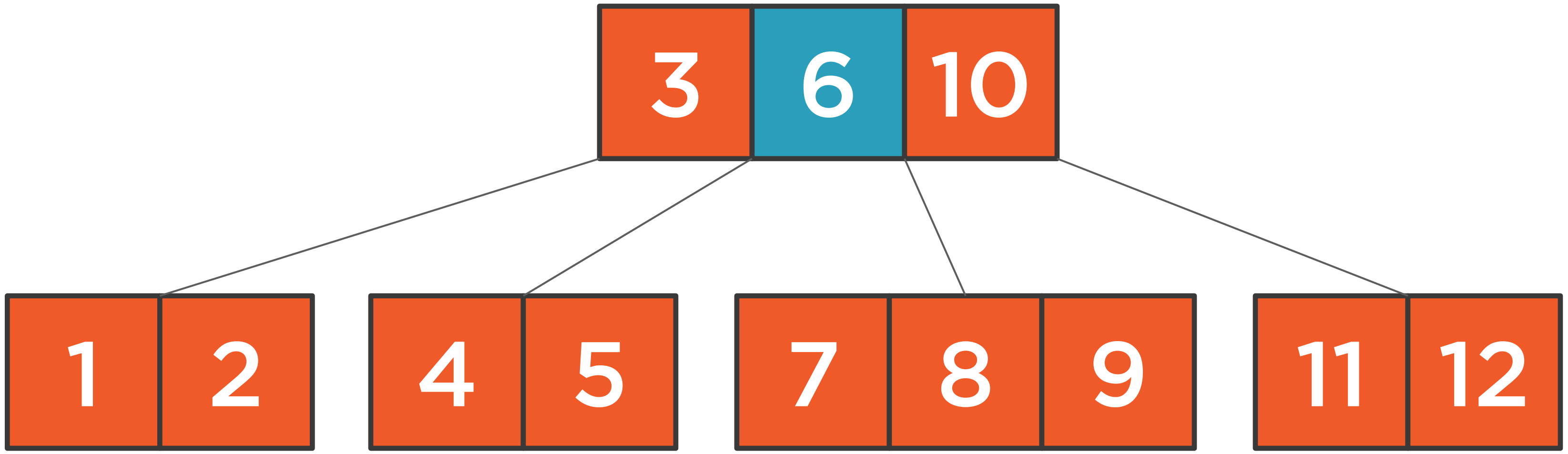
Searching



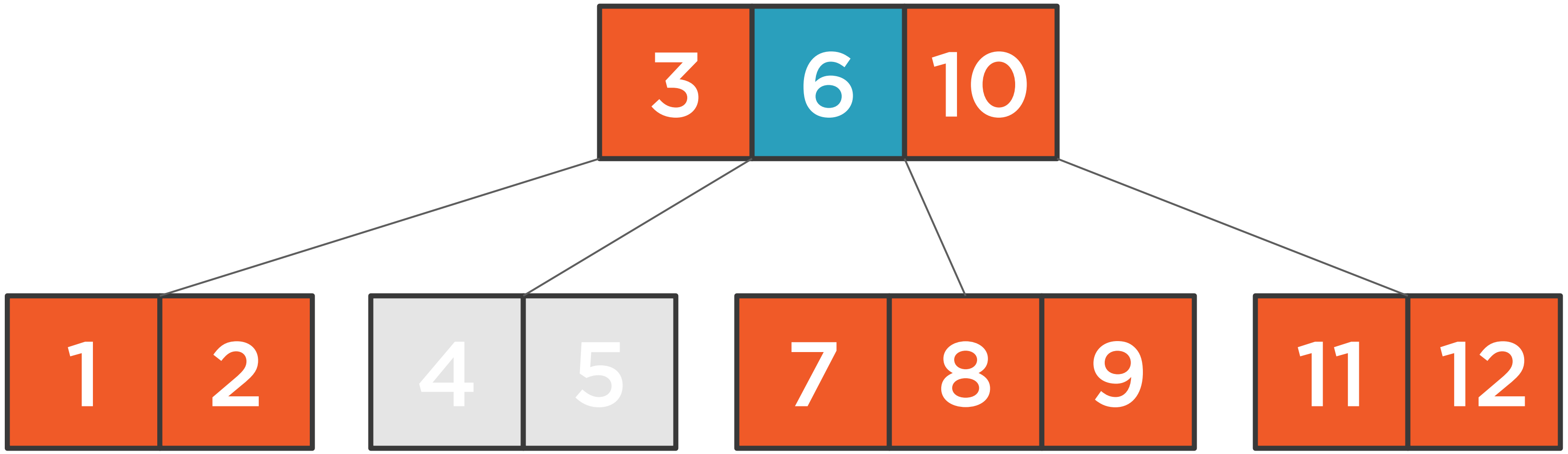
Searching



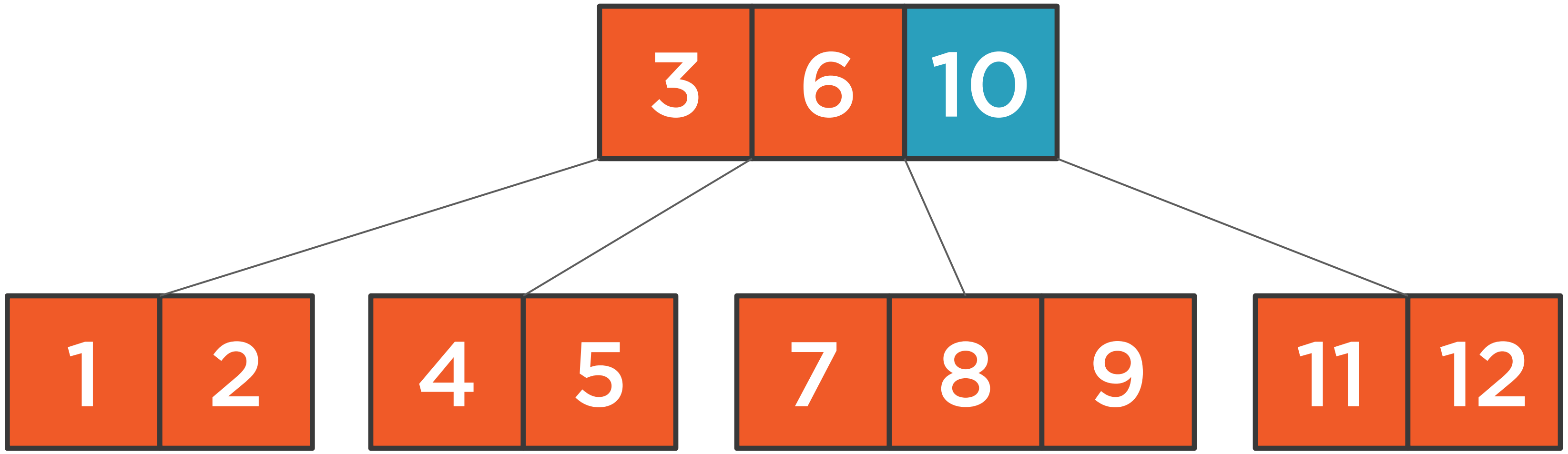
Searching



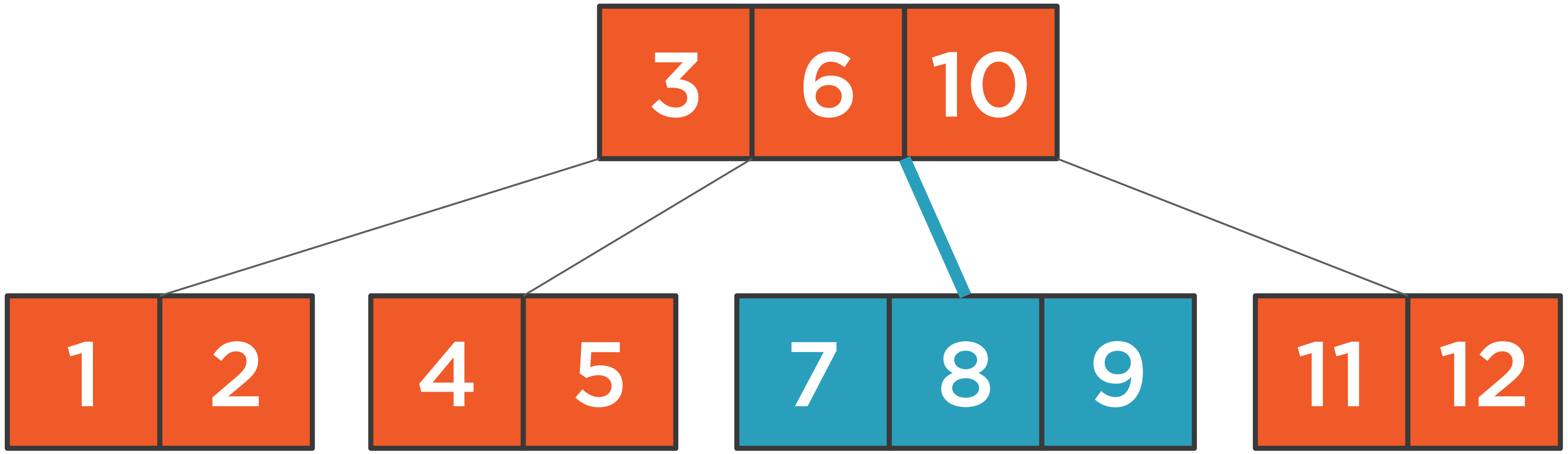
Searching



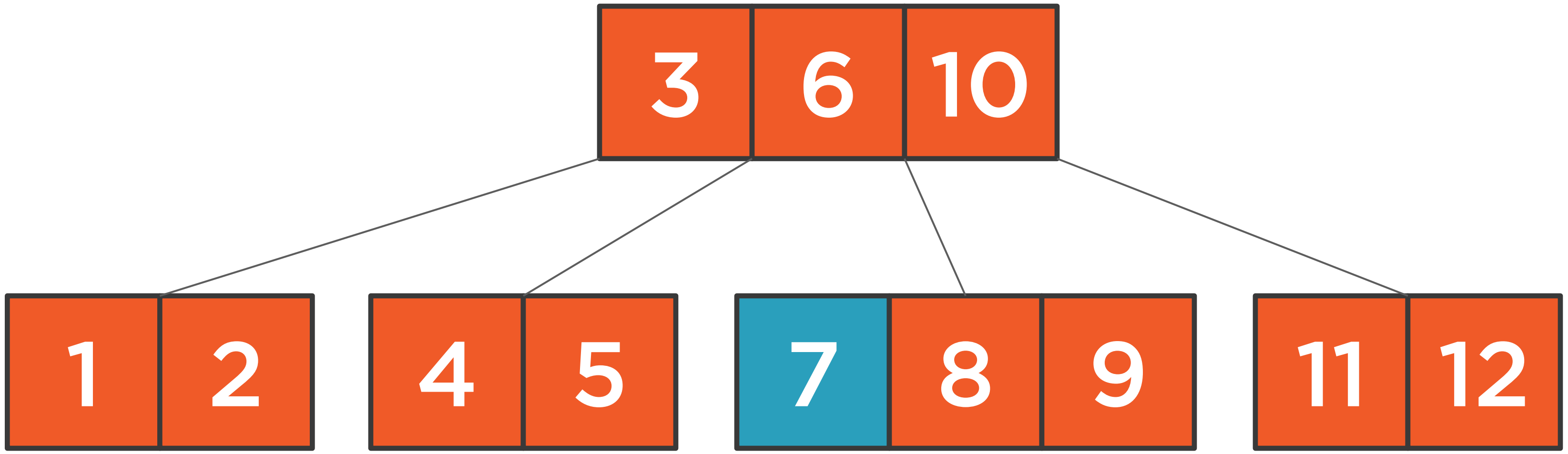
Searching



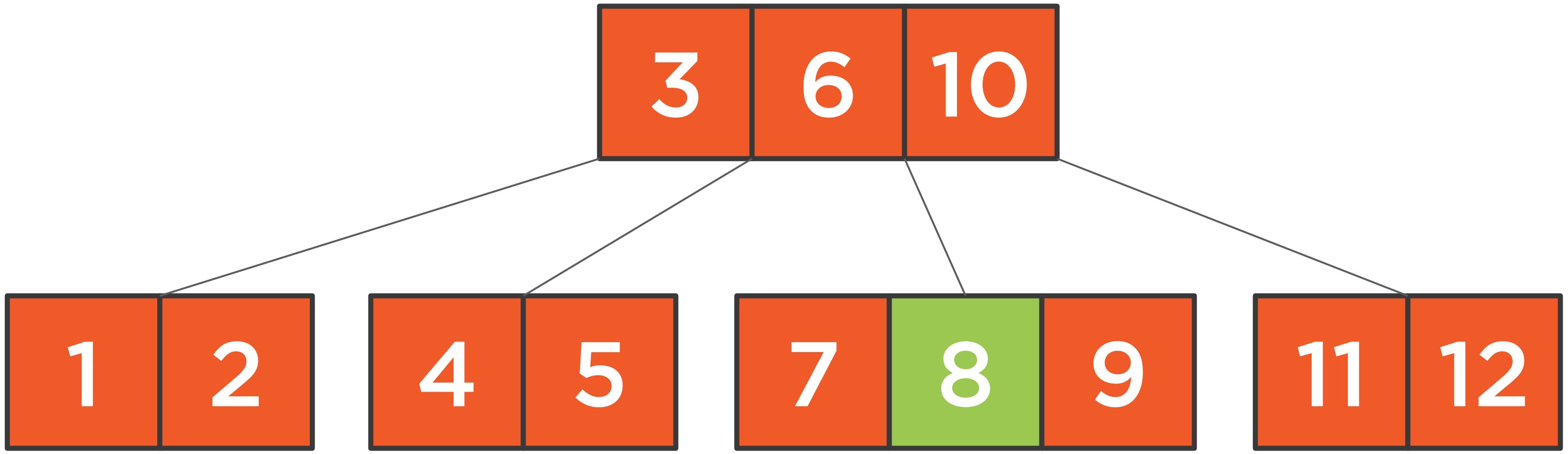
Searching



Searching



Searching



```
bool search(node, valueToFind)

  foreach value in node
    if valueToFind == value
      return true

    if valueToFind < value
      return search(<left child>, valueToFind)
  end

  return search(<last child>, valueToFind)
end
```

Searching

Compare each value in the current node with the sought value, and then recursively check every child's value.

Searching node values can be performed using a binary search

Adding Values

Add Rules

Values can only be added to leaf nodes

Full nodes are split before the algorithm enters them (ensuring leaf nodes are never full before adding an item)

```
add(T value) {
    if Empty
        Root = new BTreeNode(value)
    else {
        if Root.Full
            SplitRootNode(Root)
        InsertNonFull(Root, value)
    }

    Count++
}
```

Adding Data to the B-tree (Root Node)

If the root node is full then split the root node. Add the value to the non-full root node.

```
InsertNonFull(BTreeNode node, T value) {
    if node.Leaf
        AddValueToLeaf (node, value)
    else {
        BTreeNode child = node.ChildForValue(value)

        if (child.Full)
            SplitChildNode(node)

        InsertNonFull(child, value)
    }
}
```

Adding Data to the Tree (Non-full Node)

If the non-full node is a leaf node then add the value, otherwise find the appropriate child, splitting it if full, and then insert the node into the non-full child node.

Adding Values

Adding Values

3

Adding Values

3	5

Adding Values

Adding Values

1	3	4	5

Adding Values

Adding Values

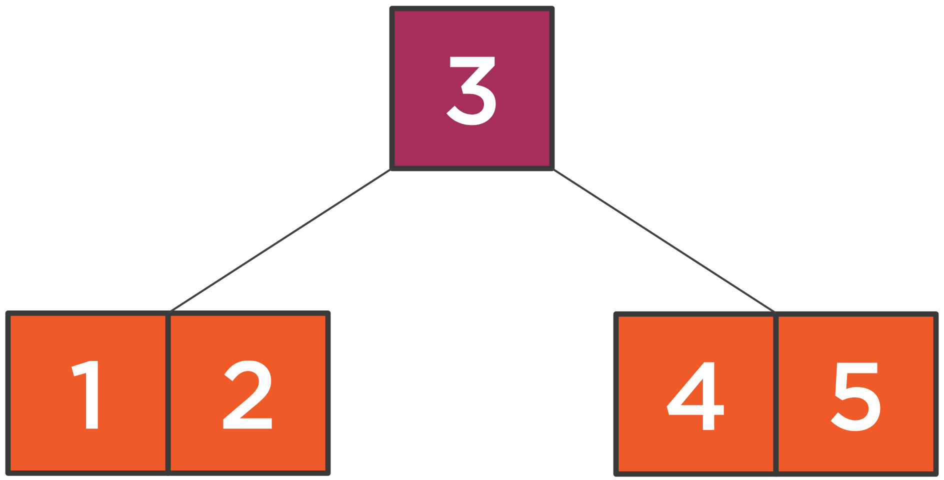
Adding Values

Adding Values

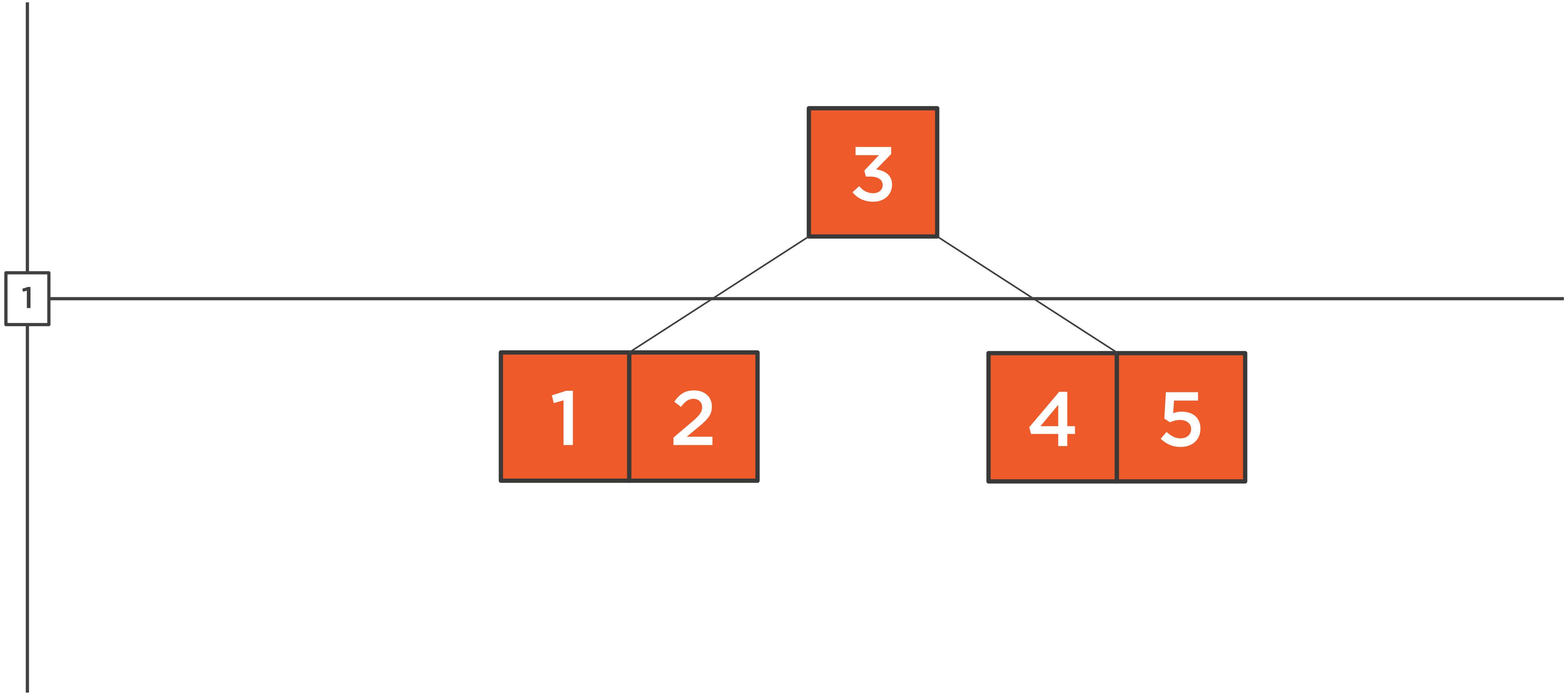
Adding Values

Adding Values

Adding Values

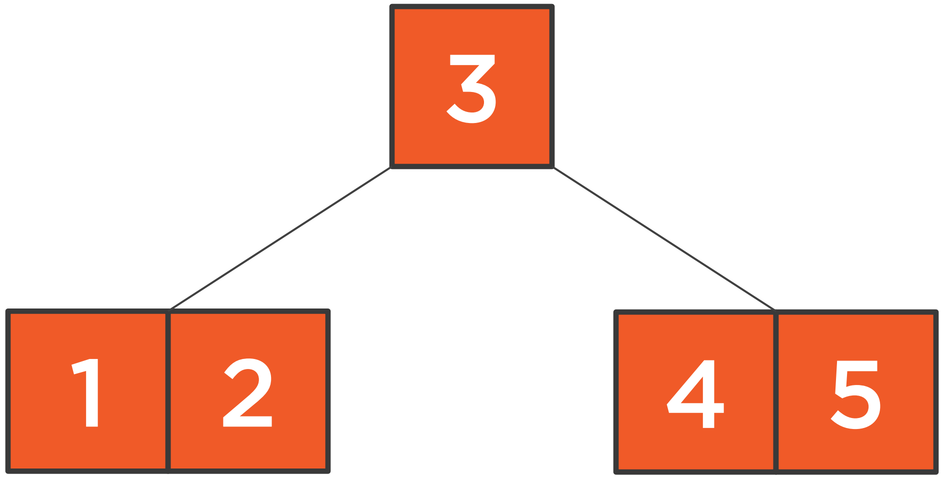


Adding Values

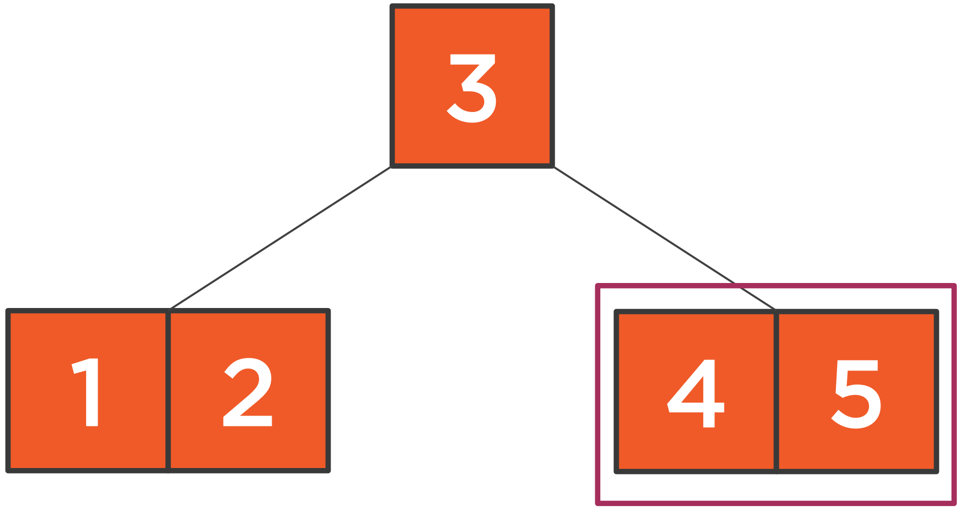


Splitting the root is the only way that the B-tree height increases

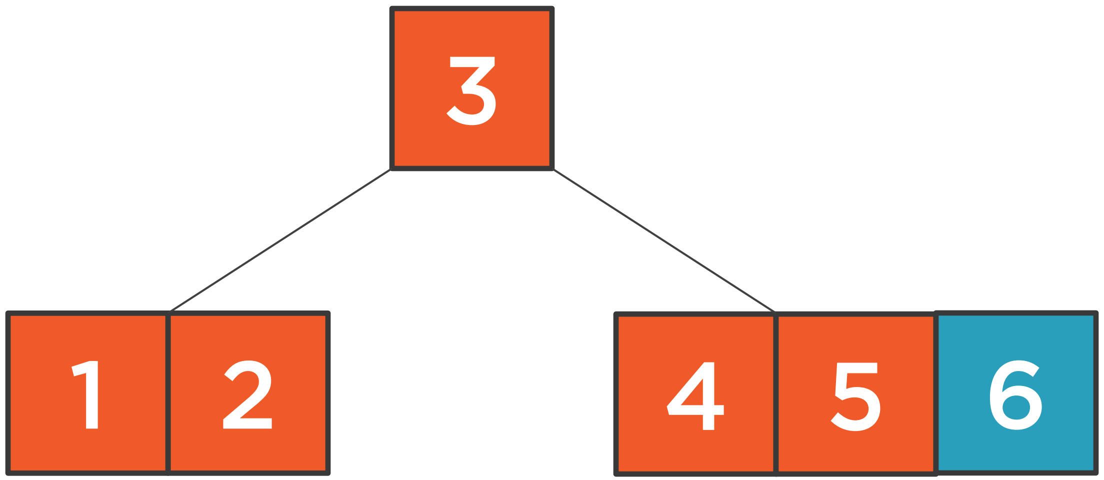
Adding Values



Adding Values



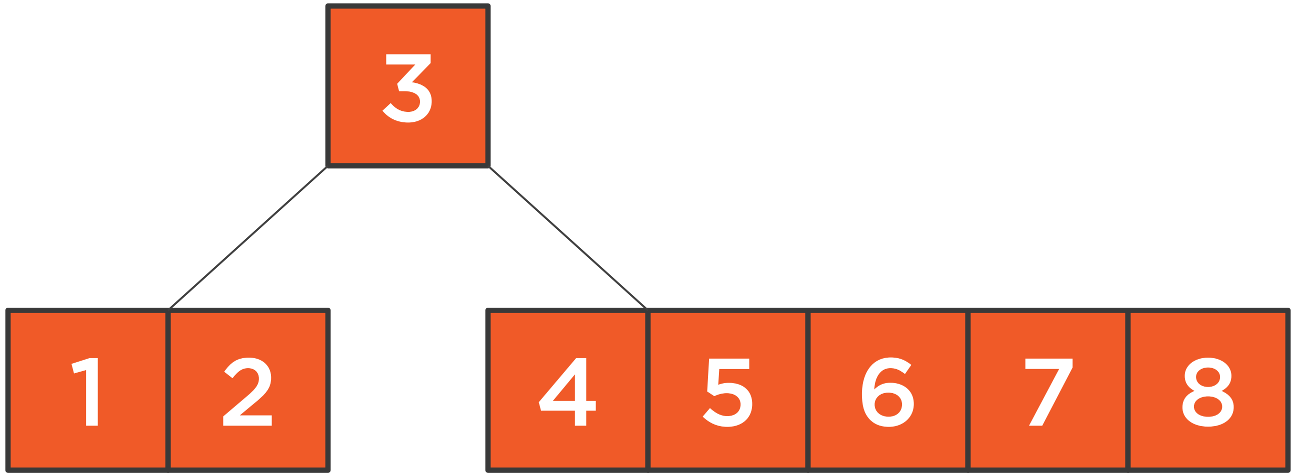
Adding a Value to a Leaf Node (T=3)



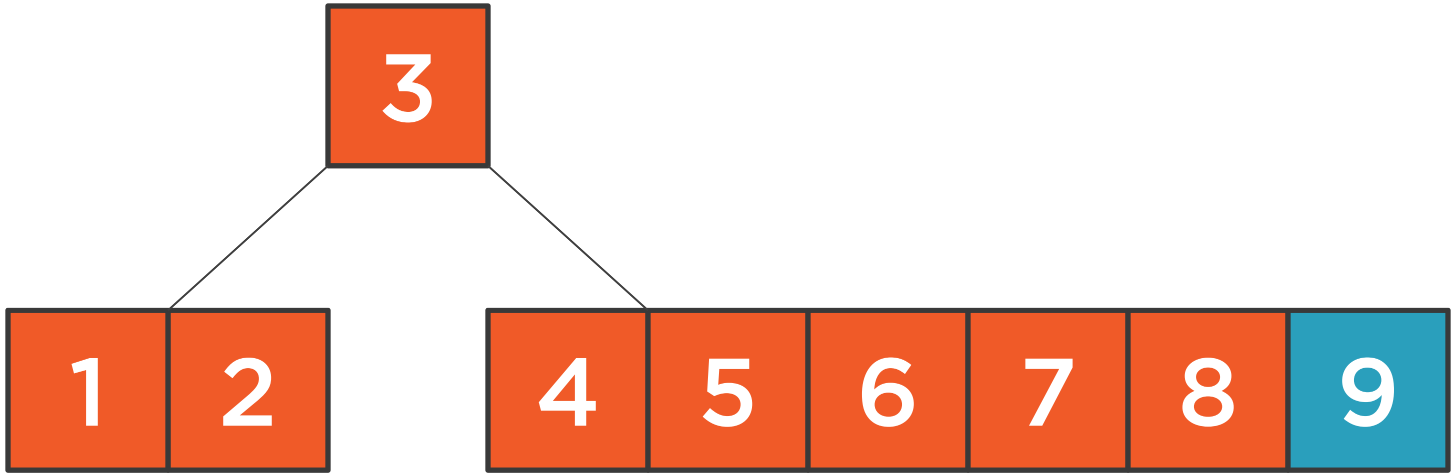
Root Node Splitting

Splitting two child nodes out of the root node to prevent the root node from exceeding the maximum number of values allowed by the B-tree degree.

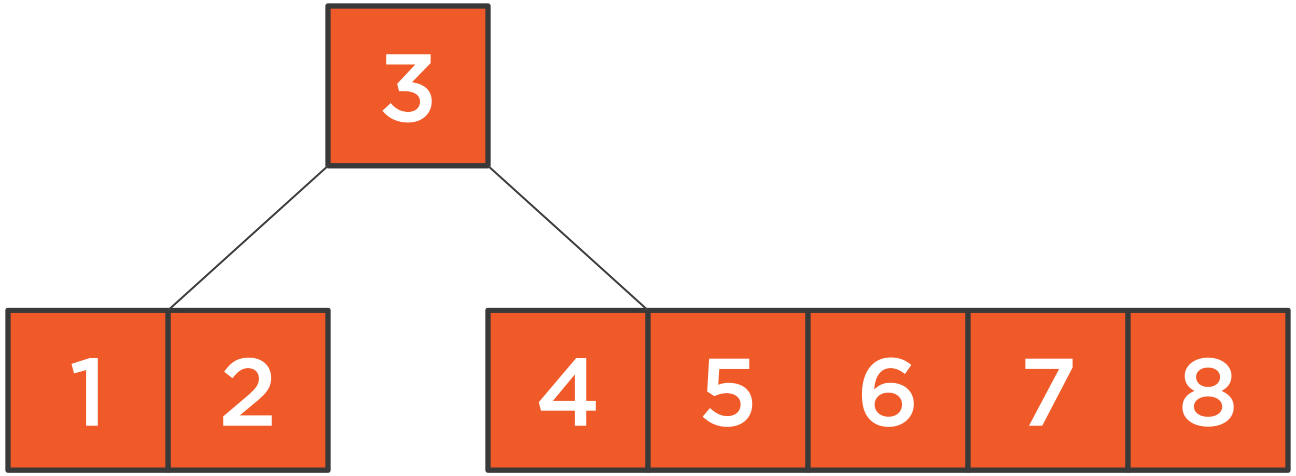
Adding a Value to a Full Leaf (T=3)



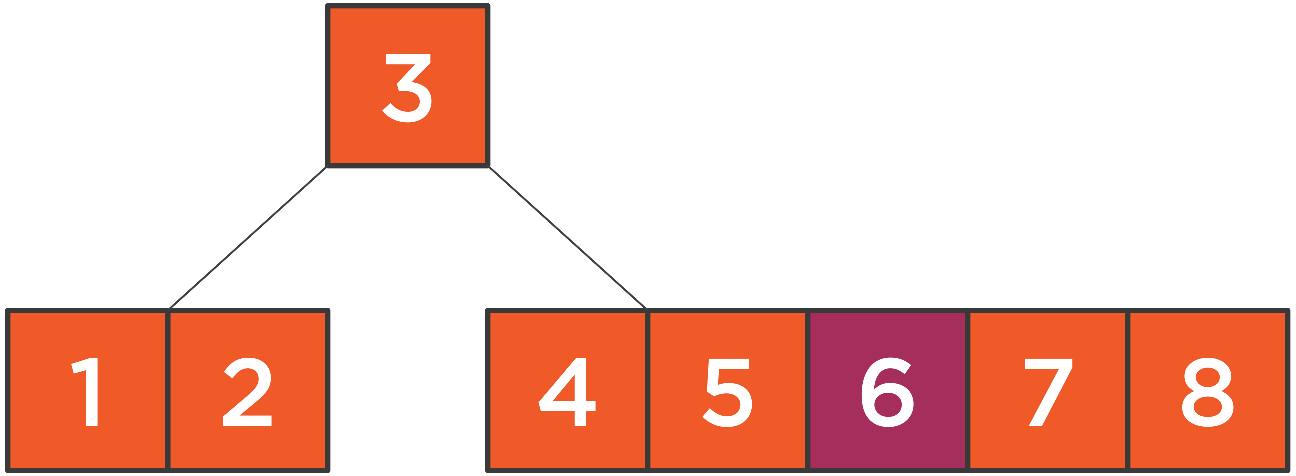
Adding a Value to a Full Leaf (T=3)



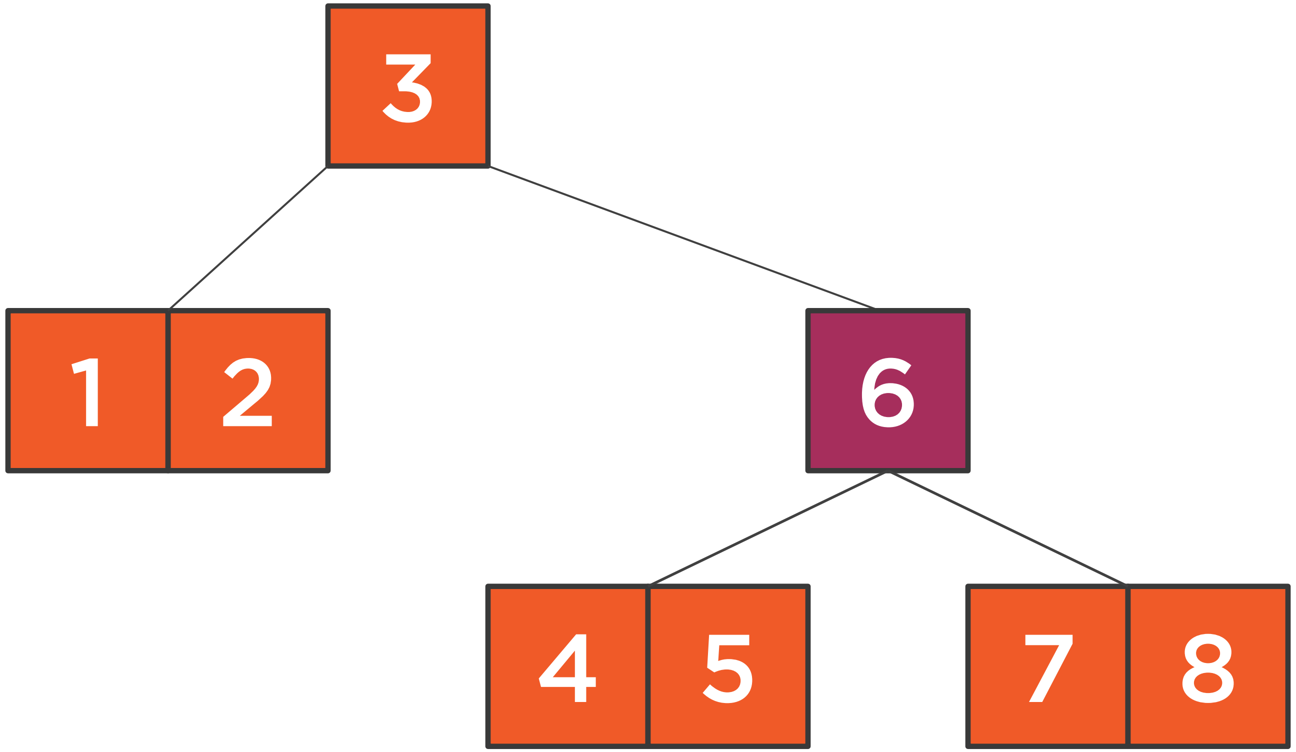
Adding a Value to a Full Leaf (T=3)



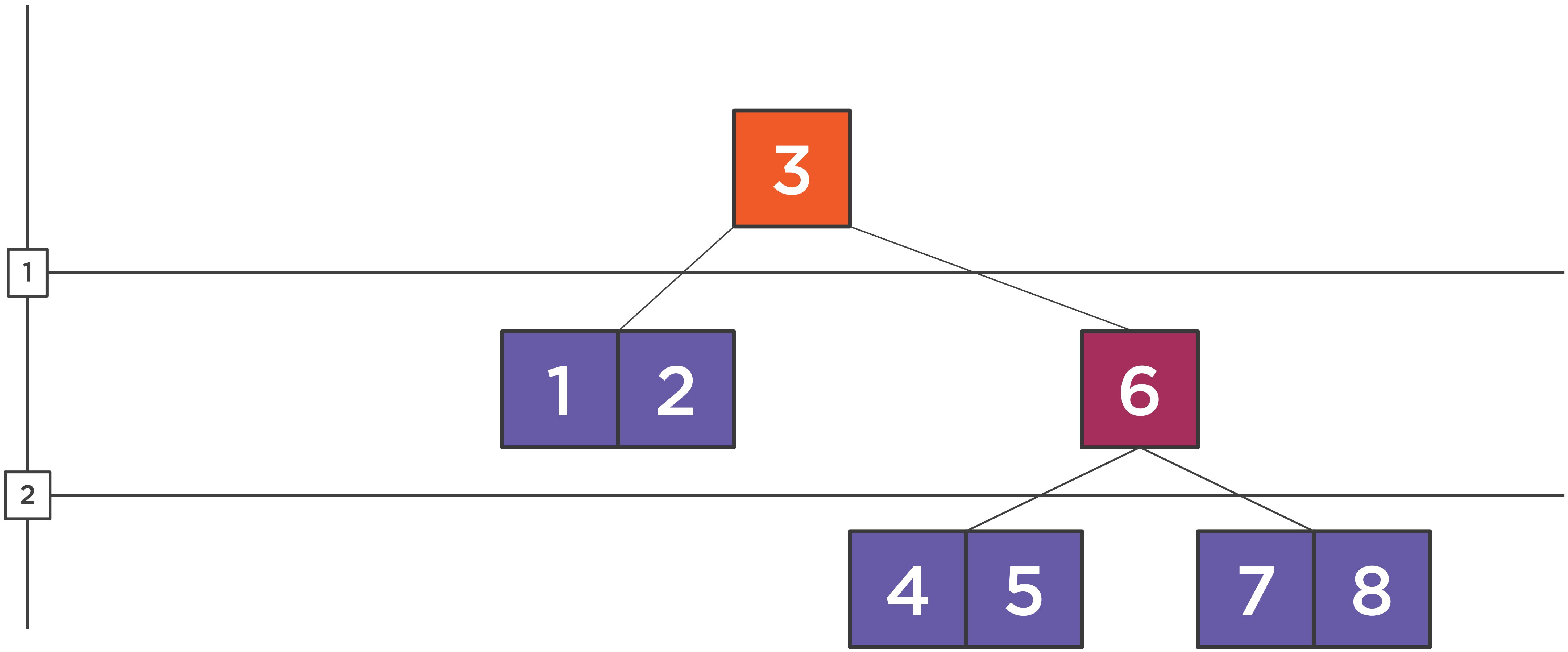
Adding a Value to a Full Leaf (T=3)



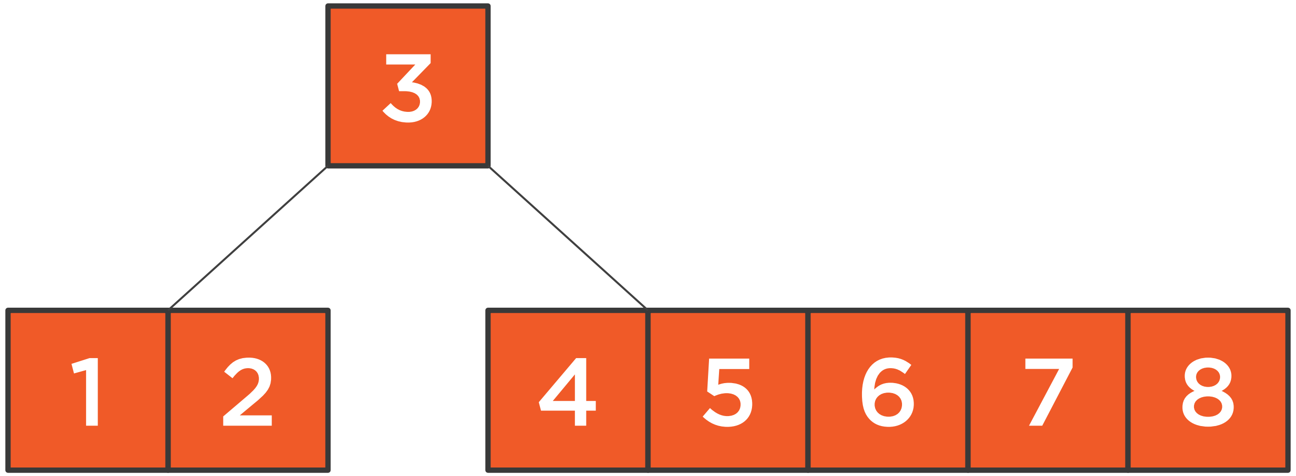
Adding a Value to a Full Leaf (T=3)



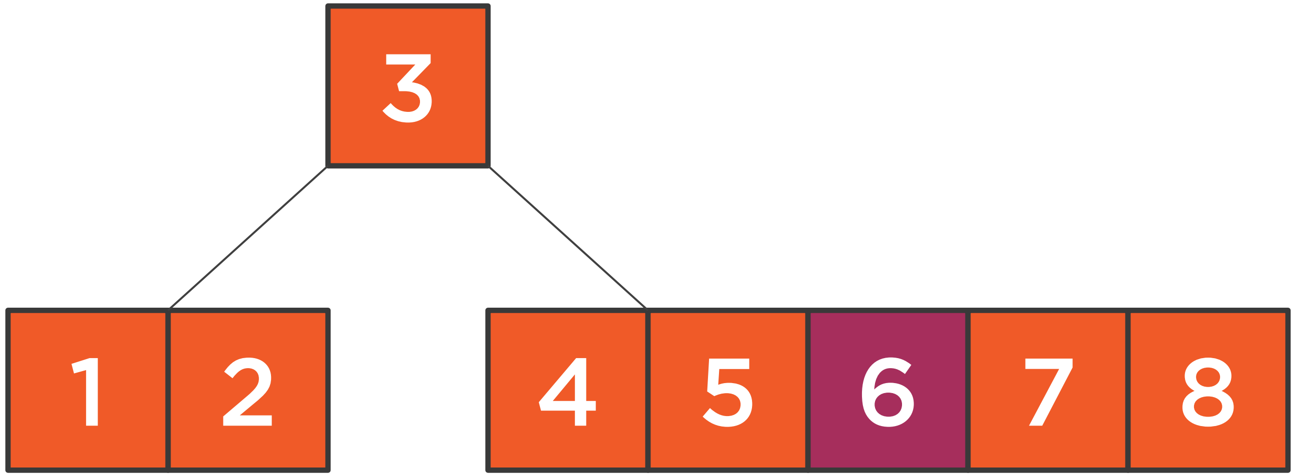
Adding a Value to a Full Leaf (T=3)



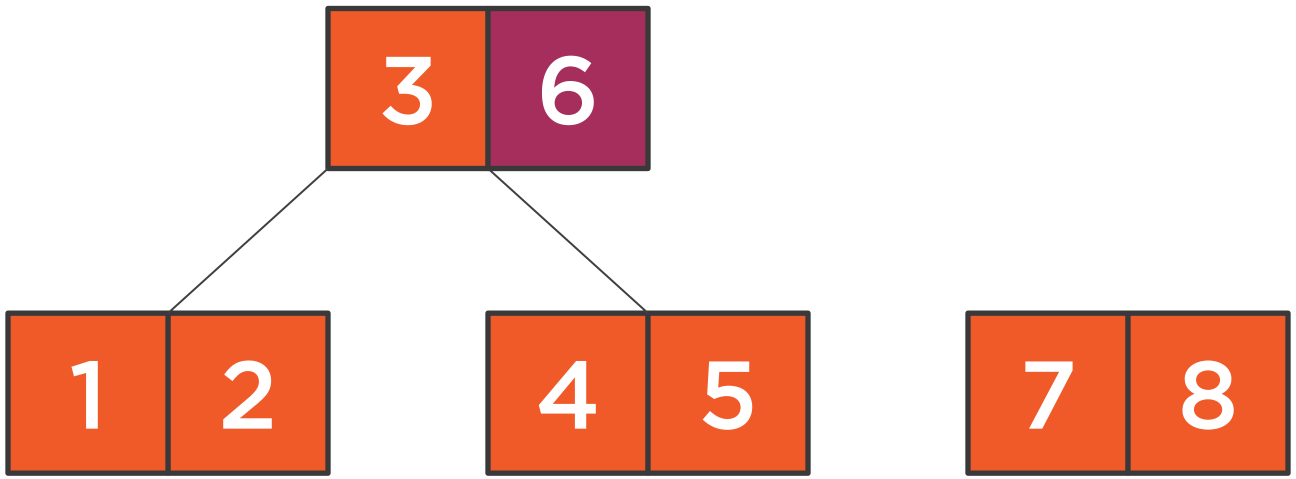
Adding a Value to a Full Leaf (T=3)



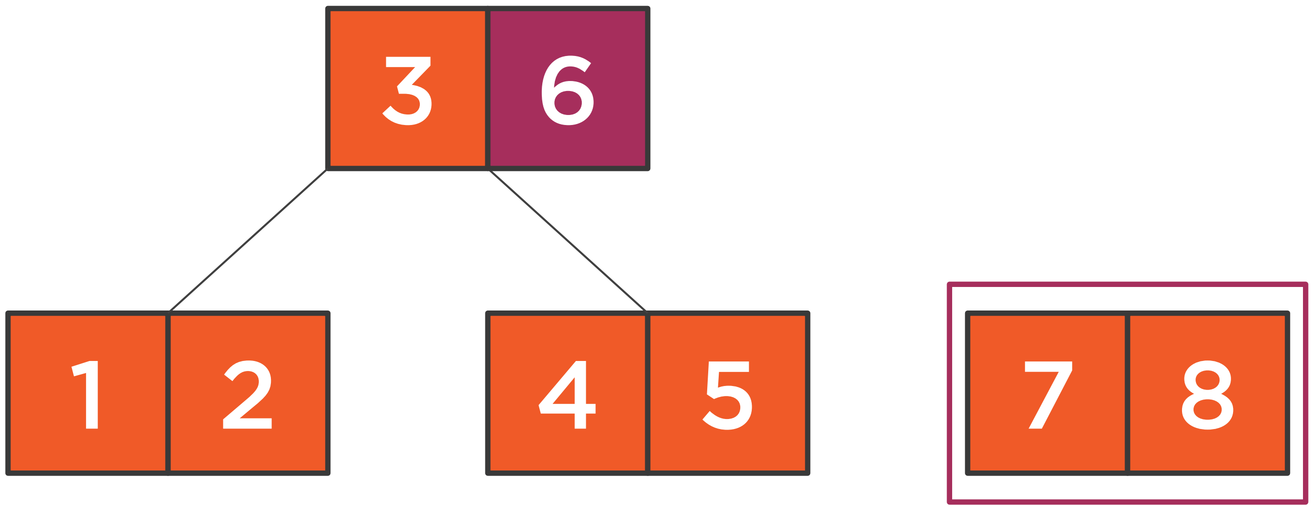
Adding a Value to a Full Leaf (T=3)



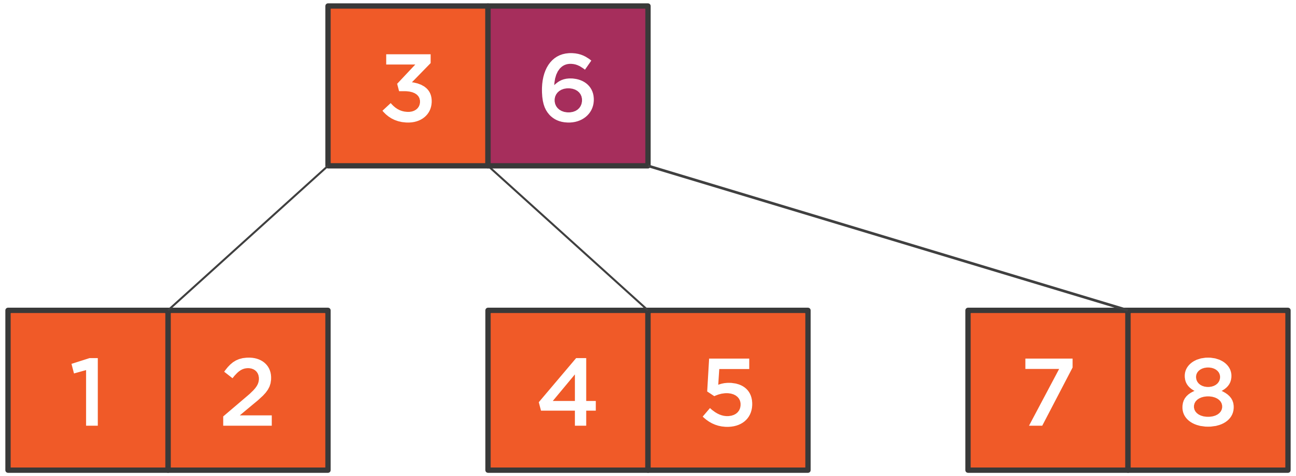
Adding a Value to a Full Leaf (T=3)



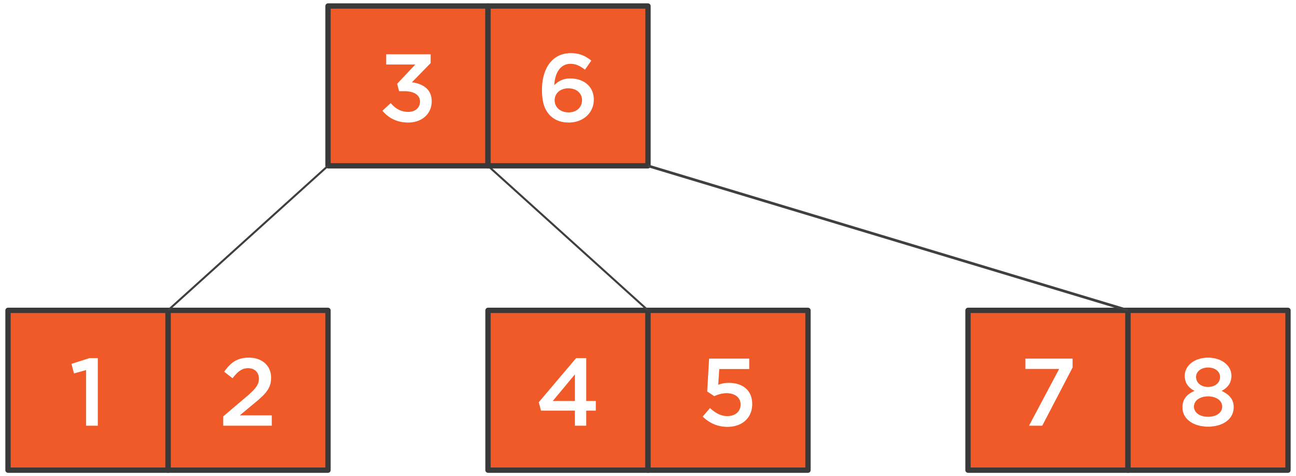
Adding a Value to a Full Leaf (T=3)



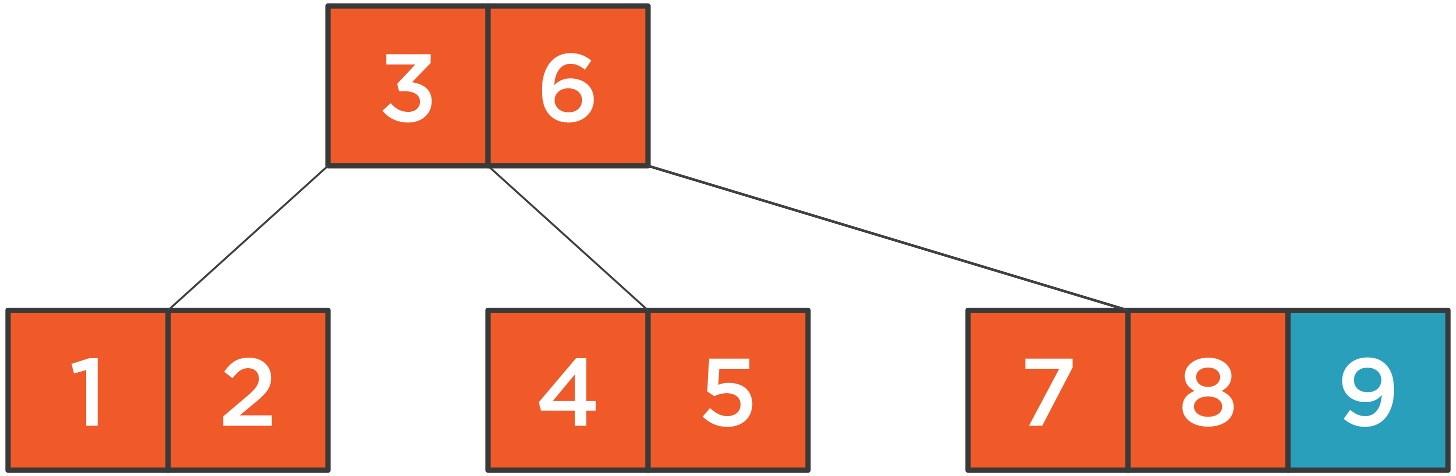
Adding a Value to a Full Leaf (T=3)



Adding a Value to a Full Leaf (T=3)



Adding a Value to a Full Leaf (T=3)



Node Splitting

Splitting a child value in half by pulling the middle value up to the parent.

Removing Values

Remove Rules

Values can only be removed from leaf nodes

Ensure that all nodes visited during the remove process have T values before entering them

Balancing Operations

Splitting Nodes

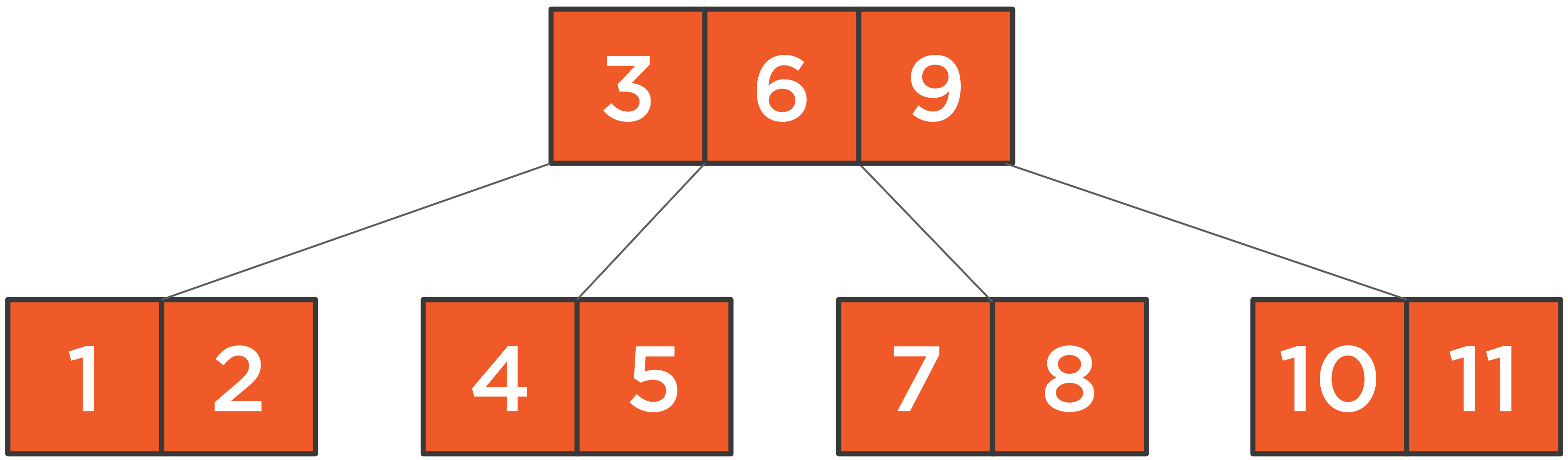
Pushing Down

Rotation

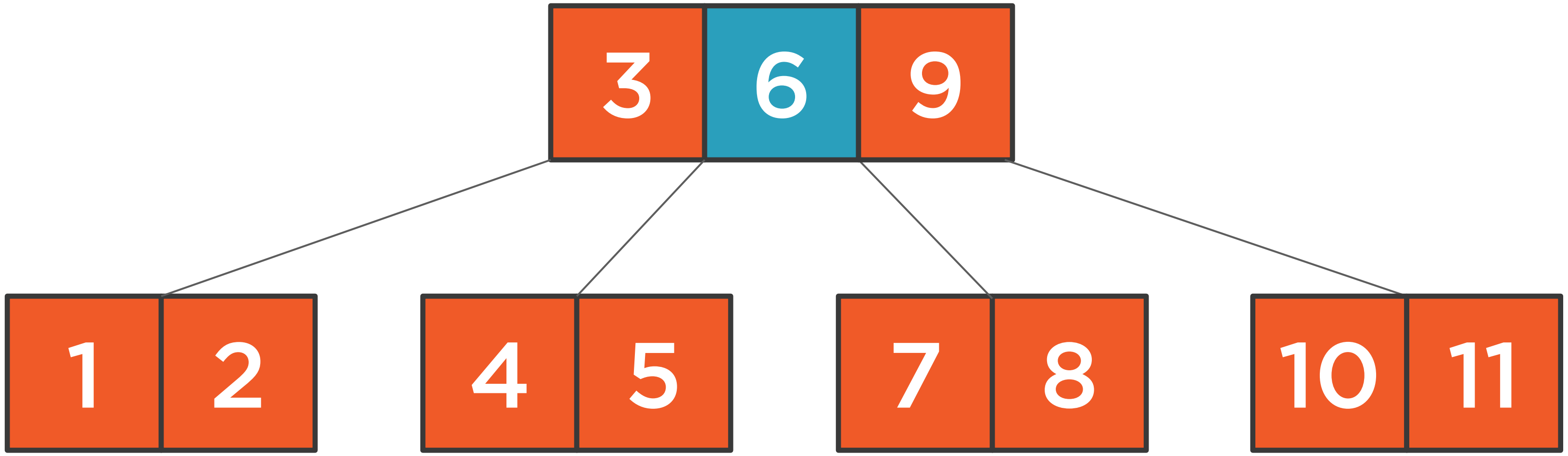
Pushing Down

Merging two child nodes by pushing a parent value between them.

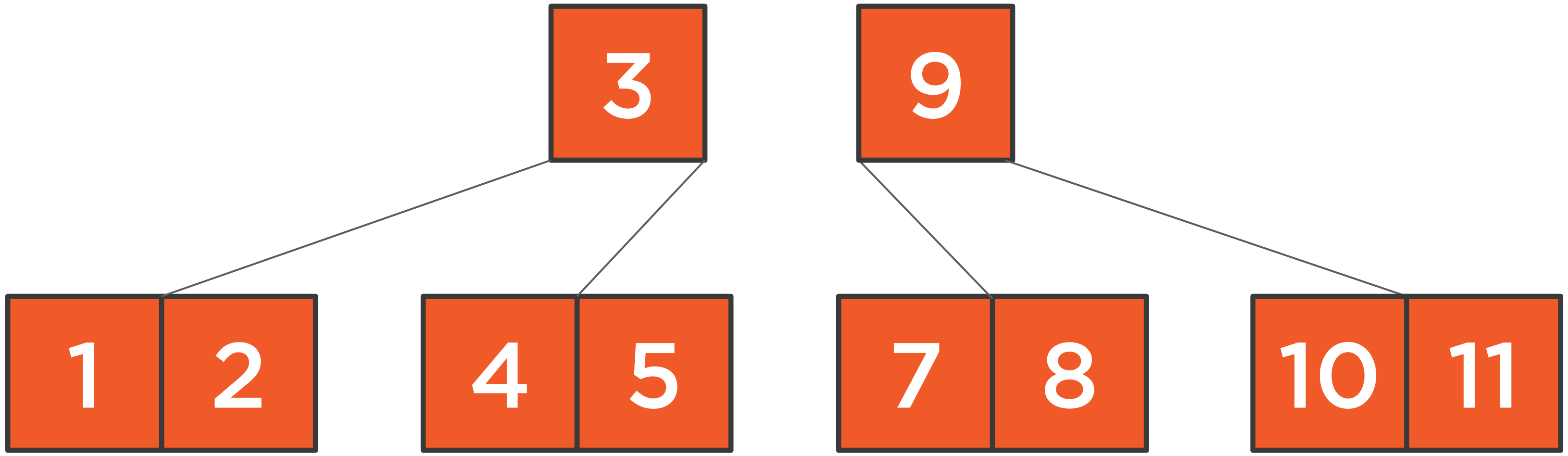
Pushing Down



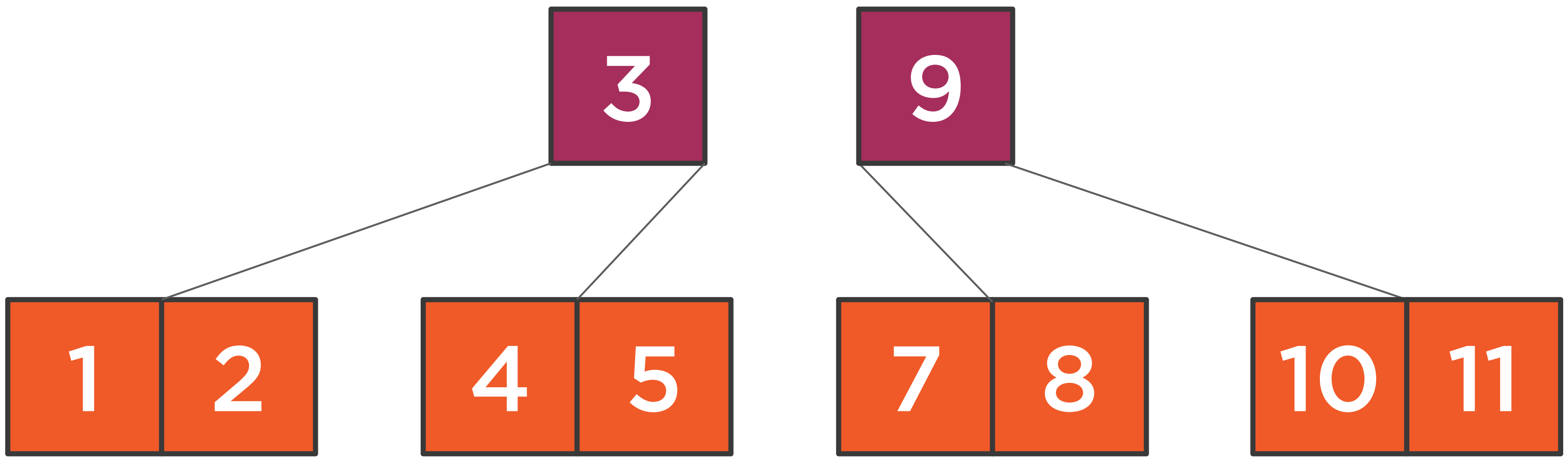
Pushing Down



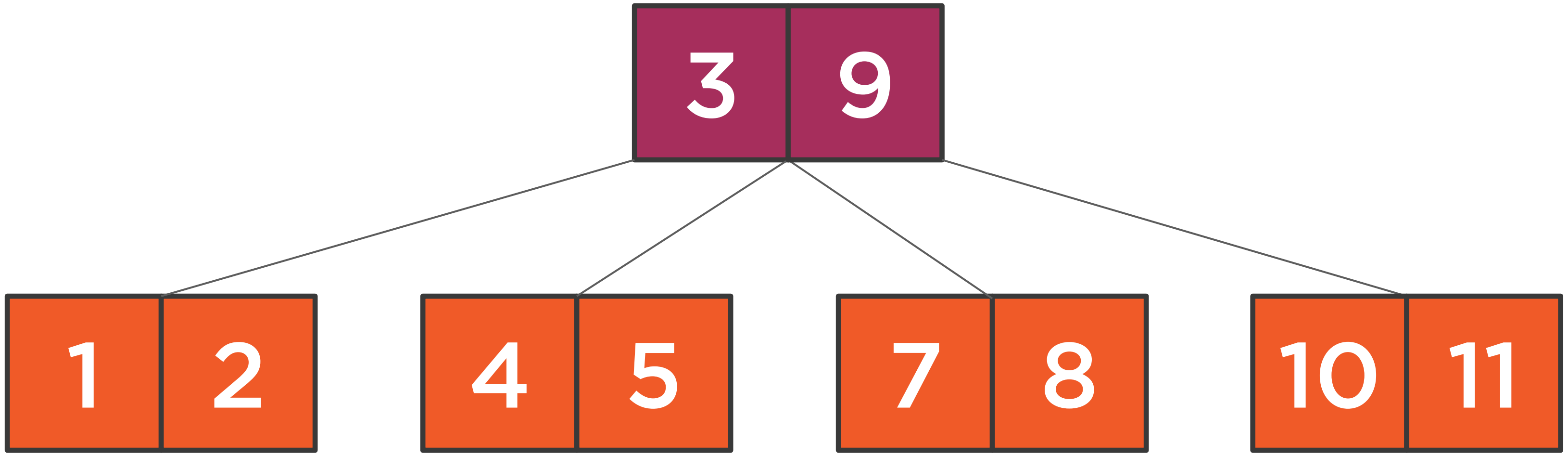
Pushing Down



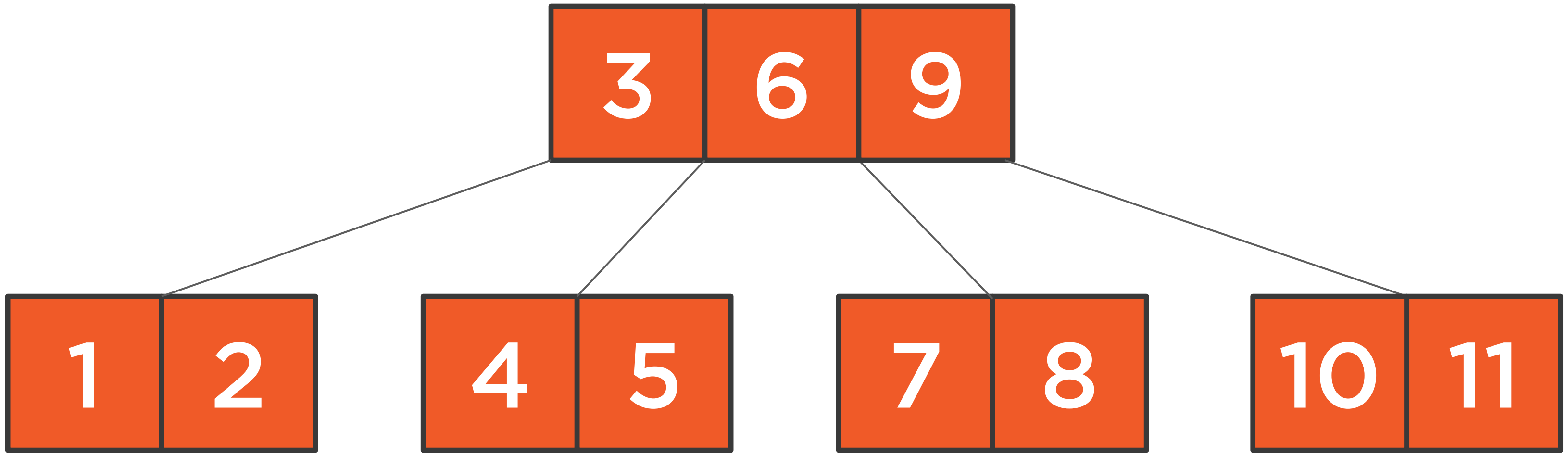
Pushing Down



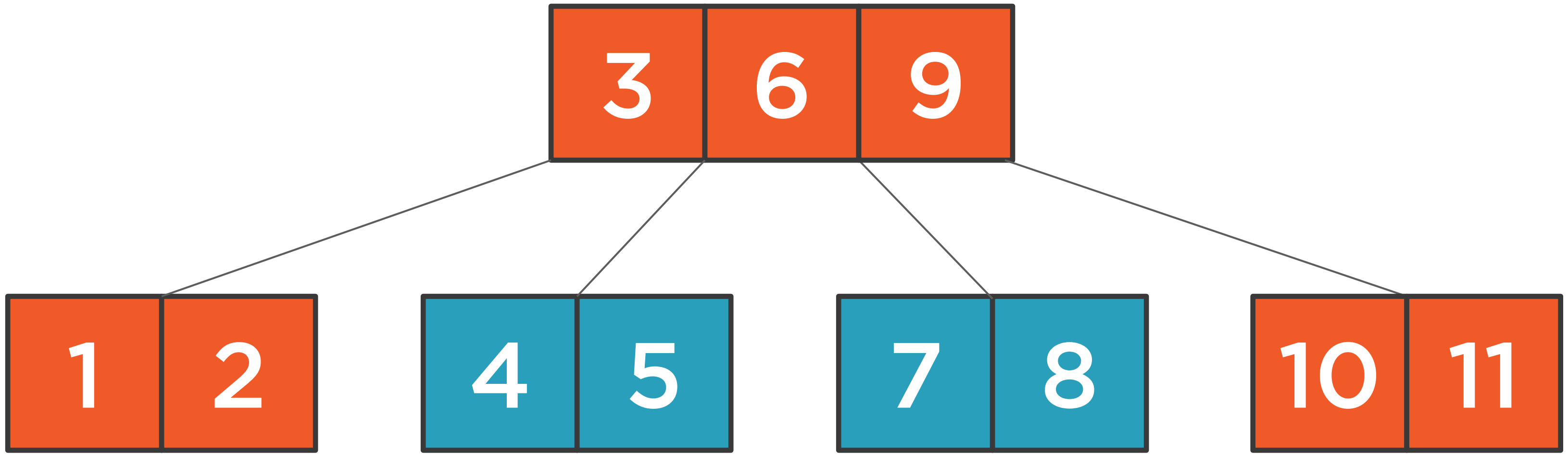
Pushing Down



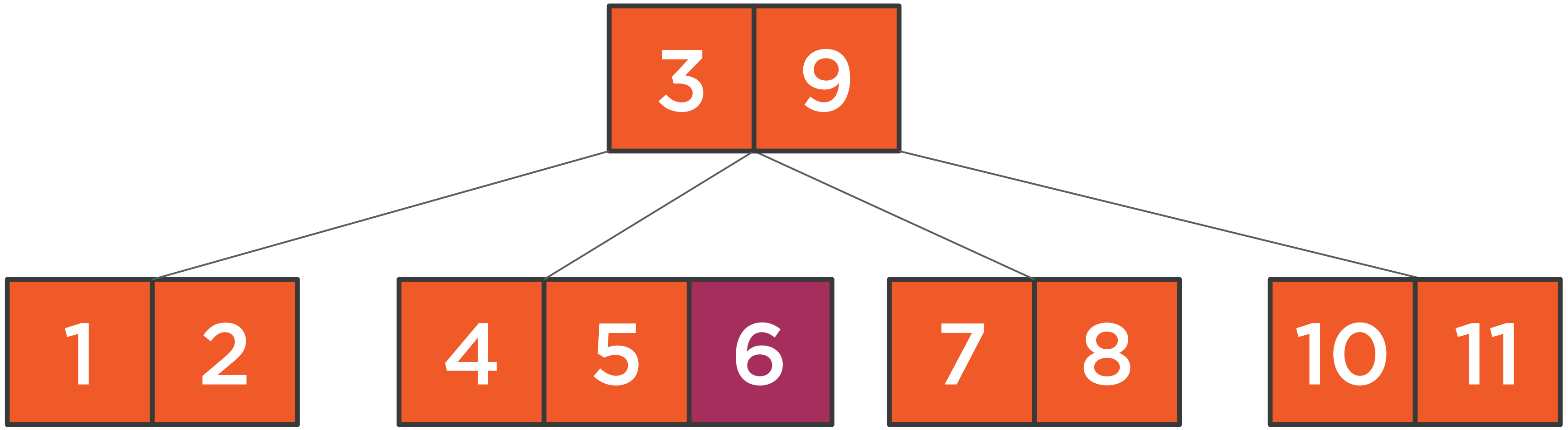
Pushing Down



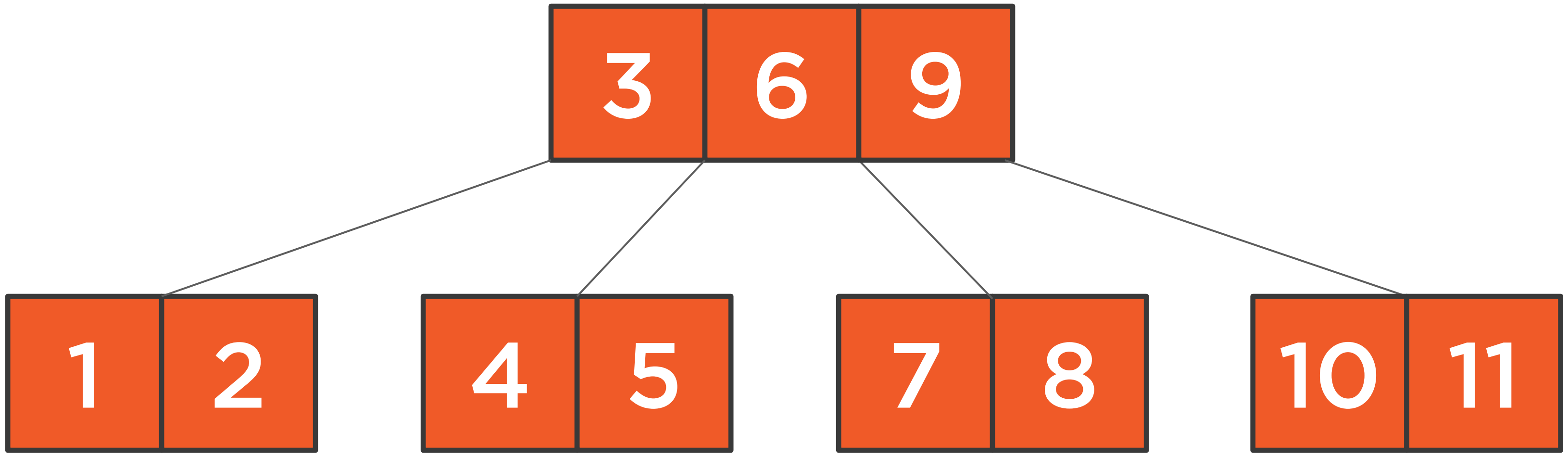
Pushing Down



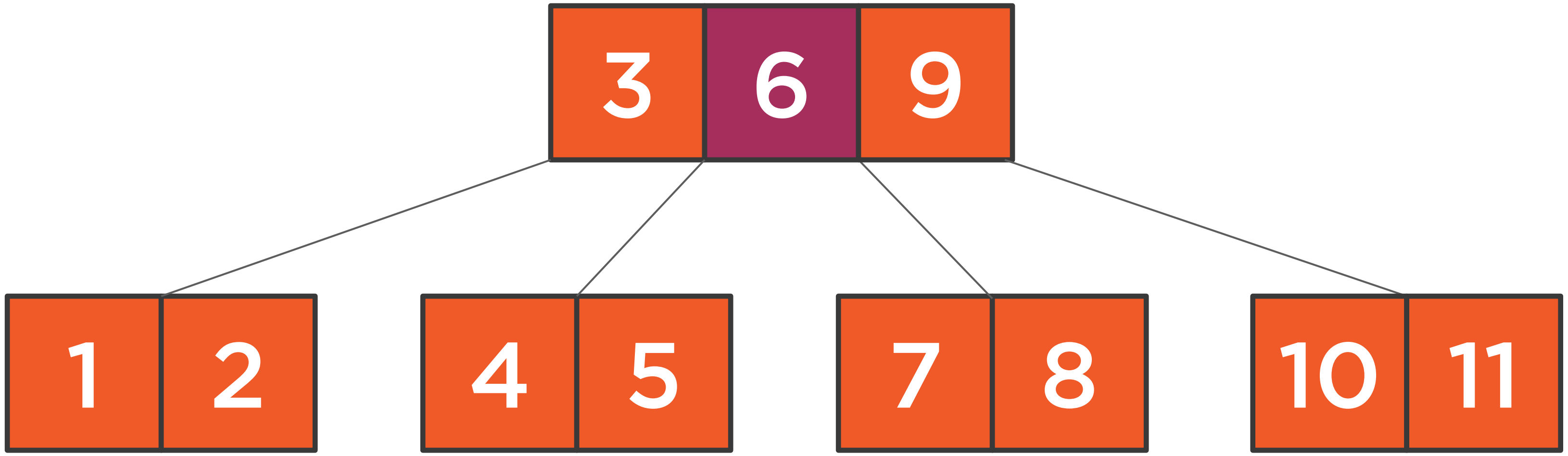
Pushing Down



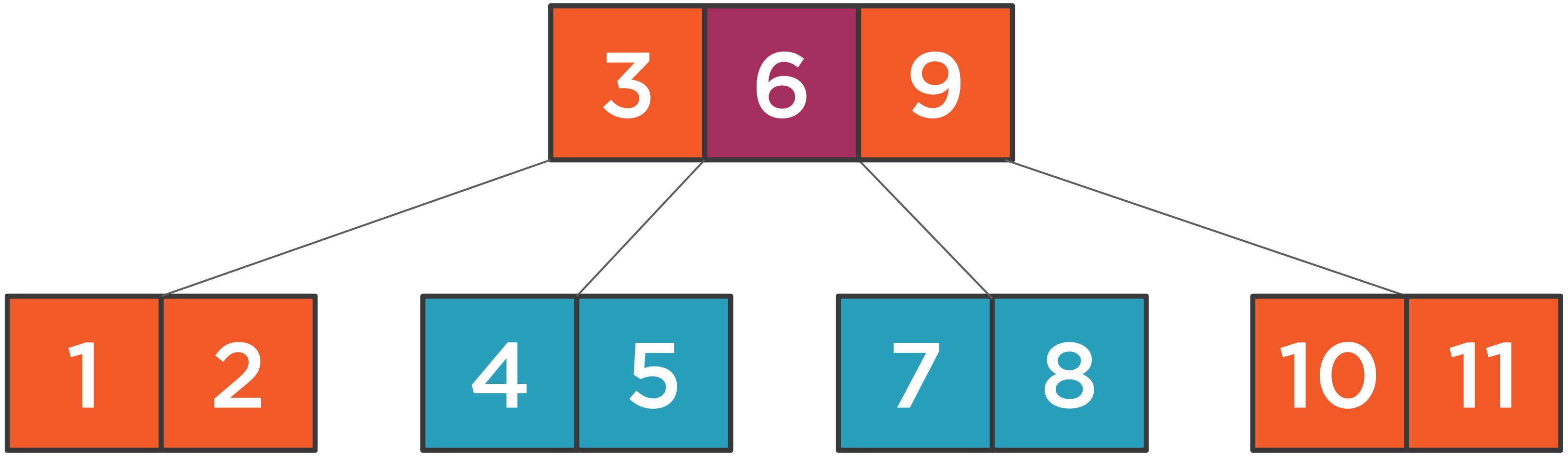
Pushing Down



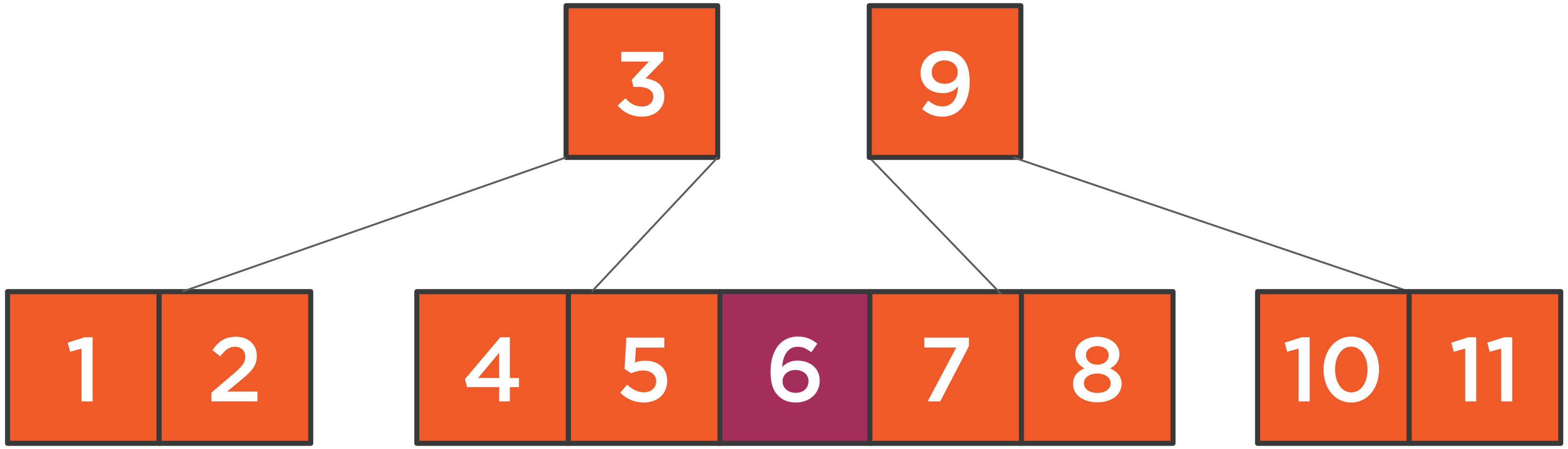
Pushing Down



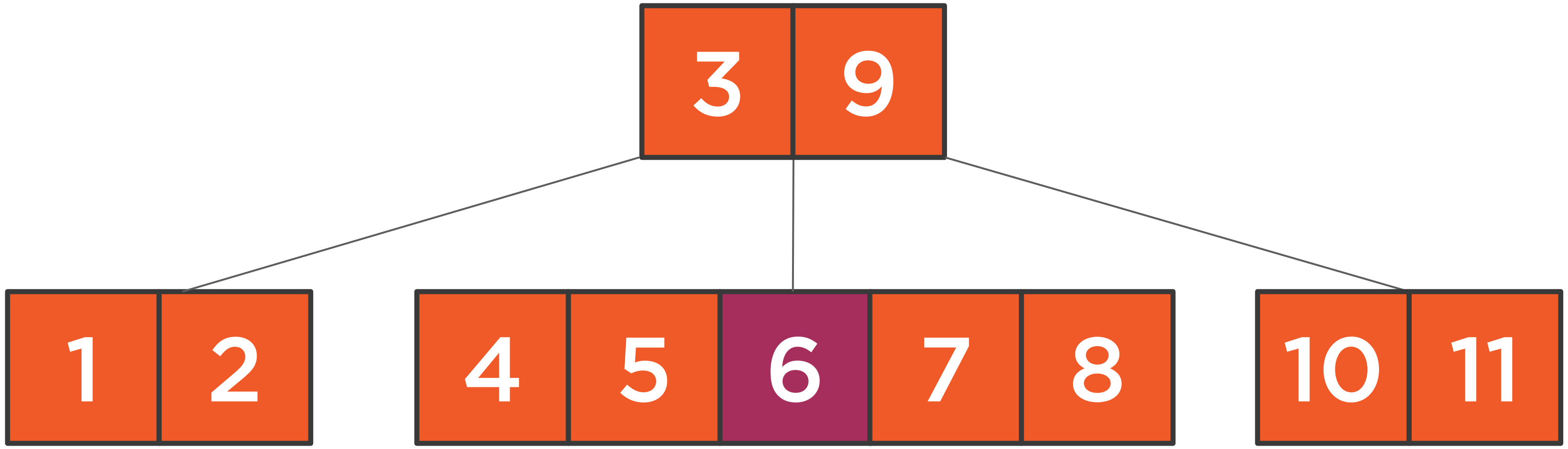
Pushing Down



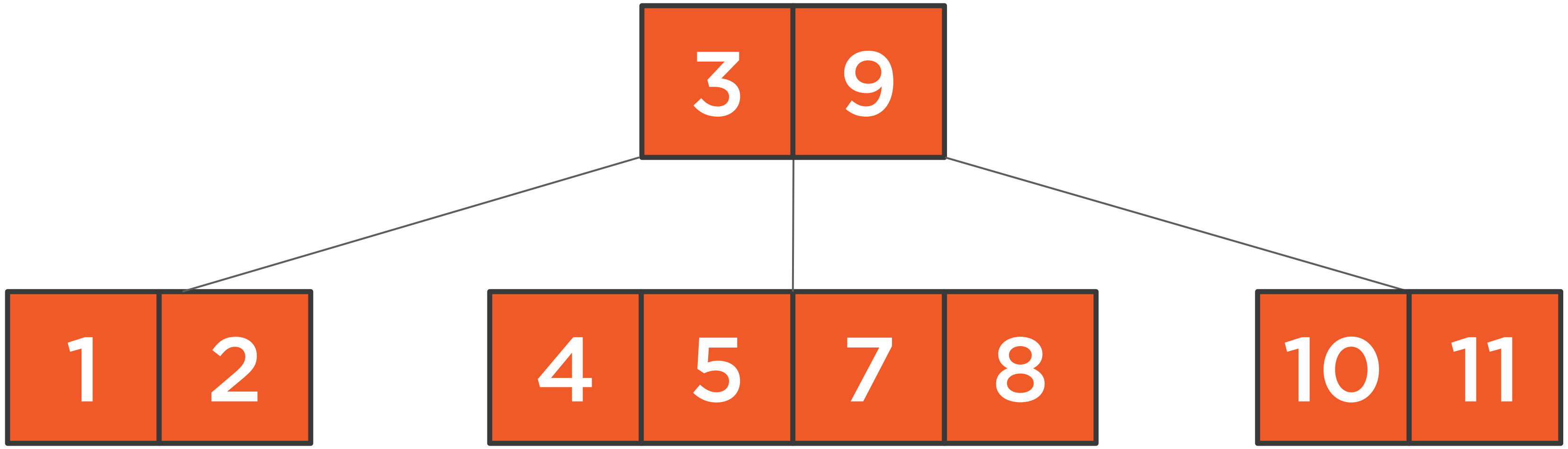
Pushing Down



Pushing Down



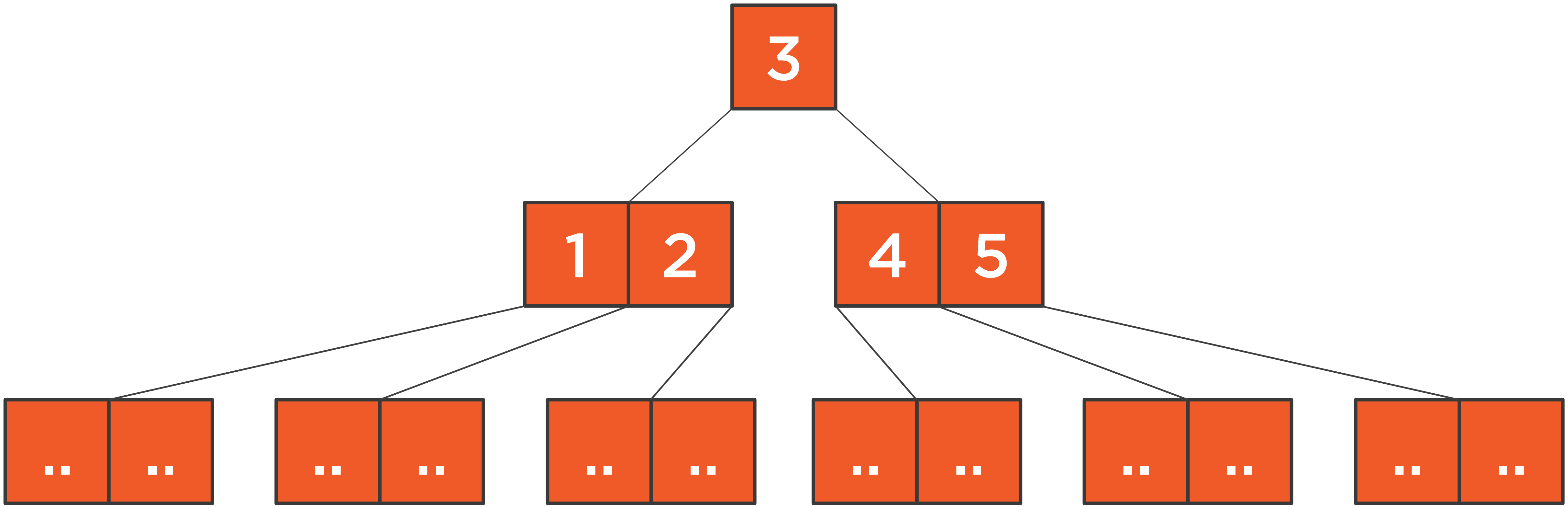
Pushing Down



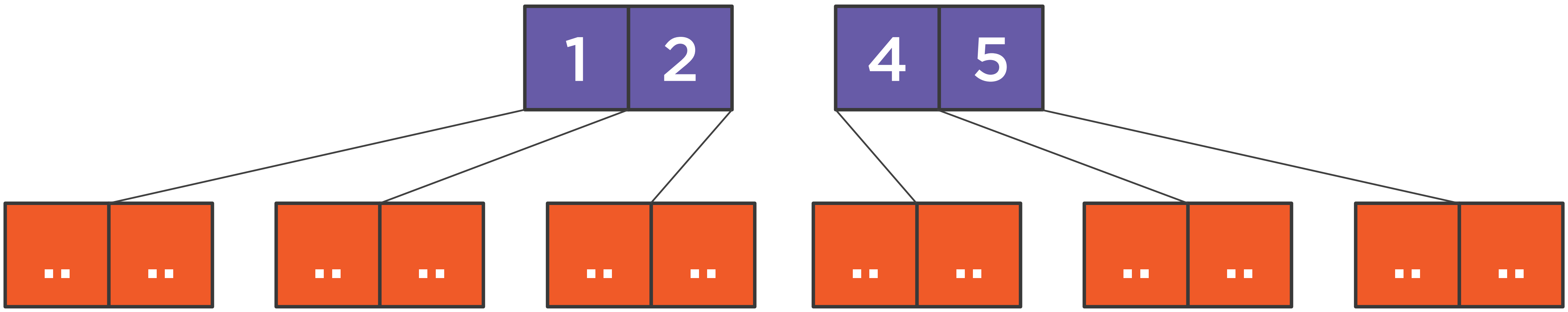
Push Down Minimal Root

When the root has a single value and two minimal children, they can be combined with it to form a full root node.

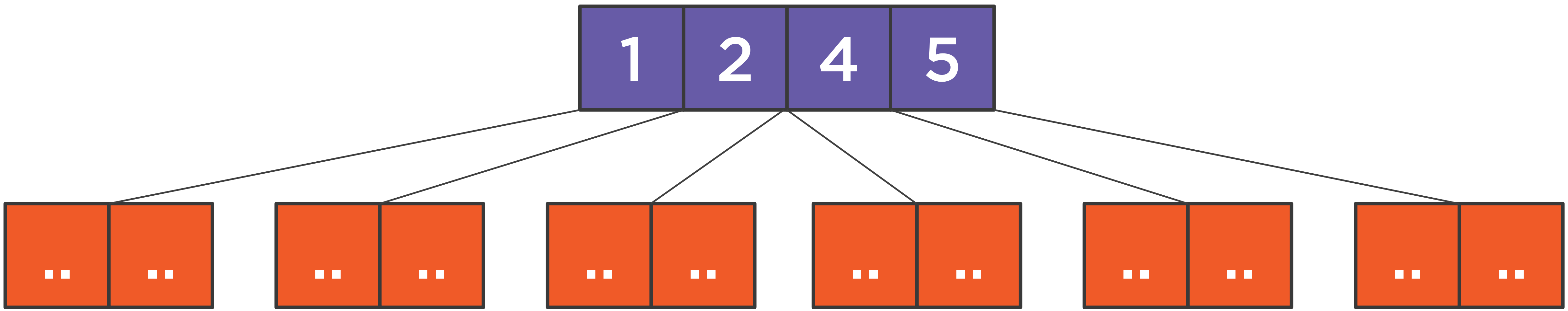
Pushing Down Minimal Root



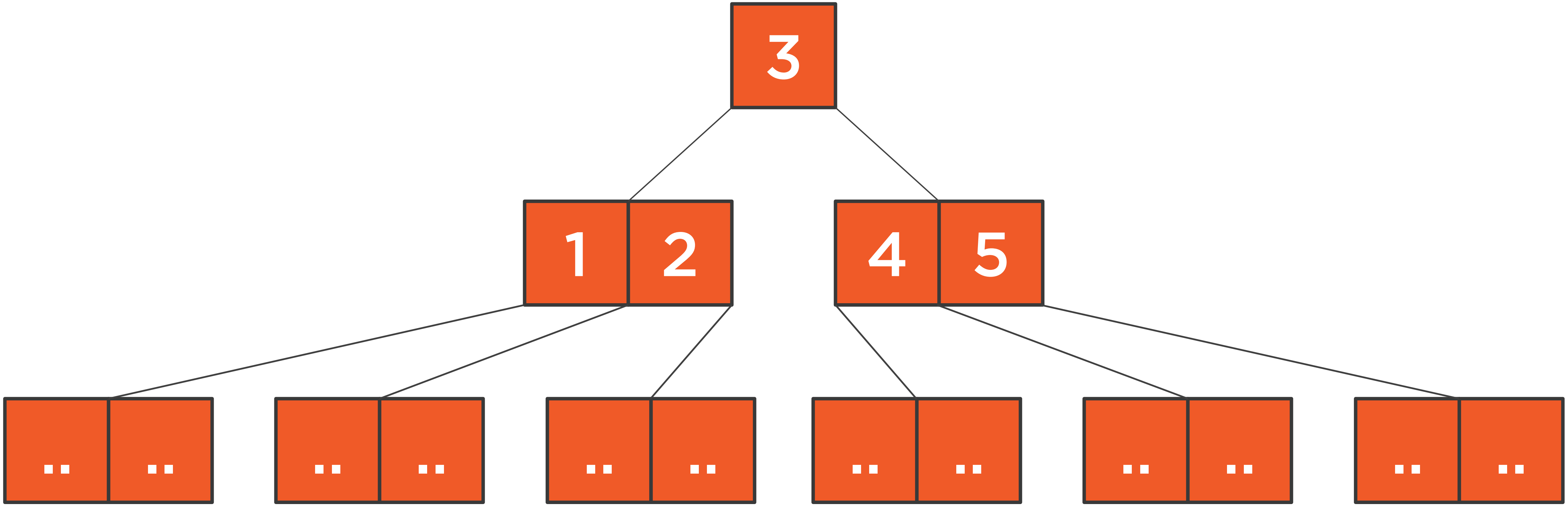
Pushing Down Minimal Root



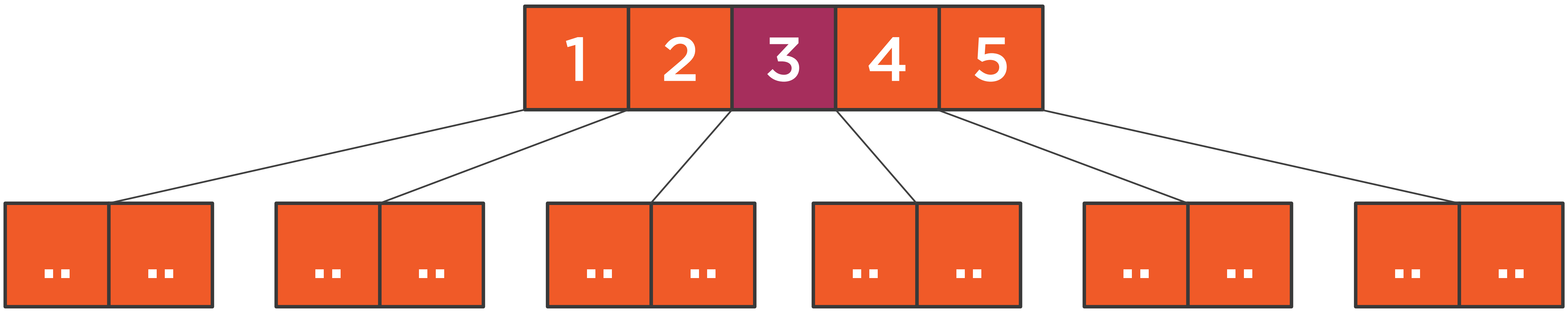
Pushing Down Minimal Root



Pushing Down Minimal Root



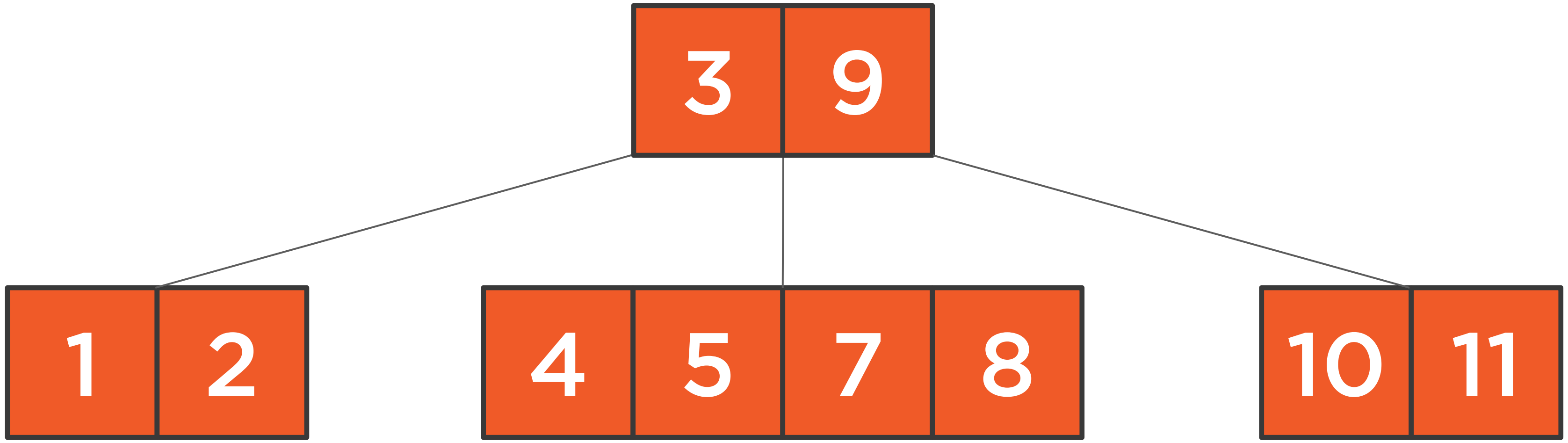
Pushing Down Minimal Root



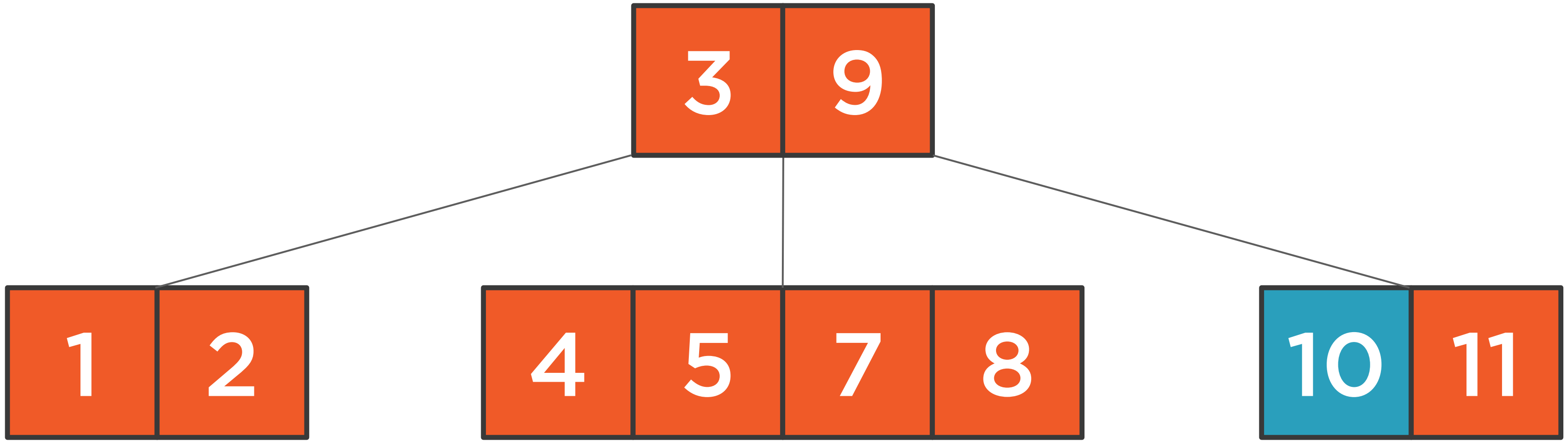
Rotation

Rotating a value from a non-minimal child, to a sibling minimal child

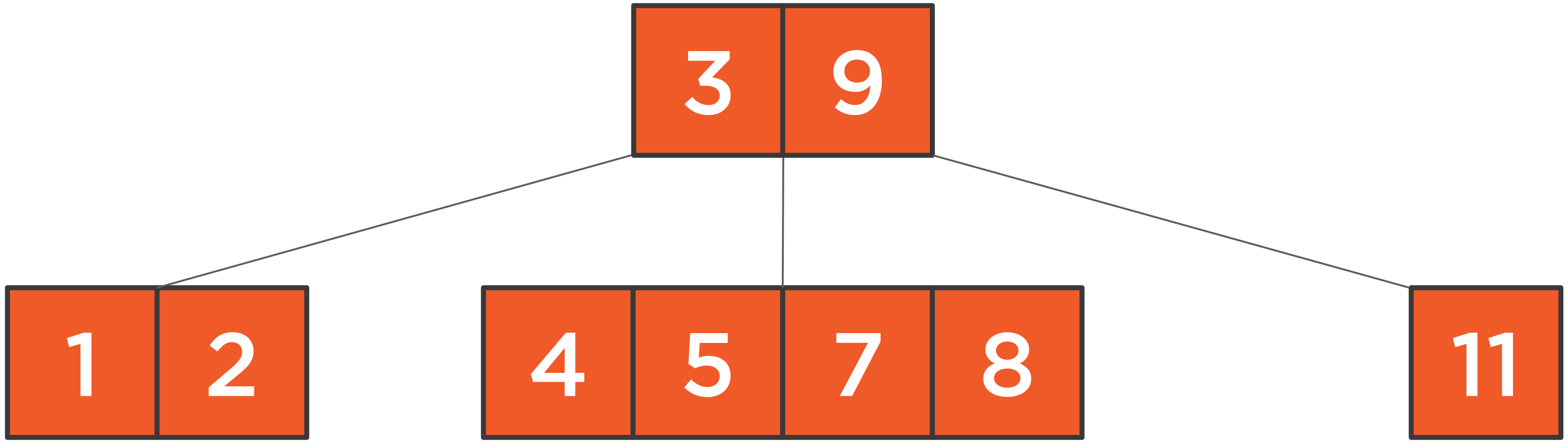
Rotation



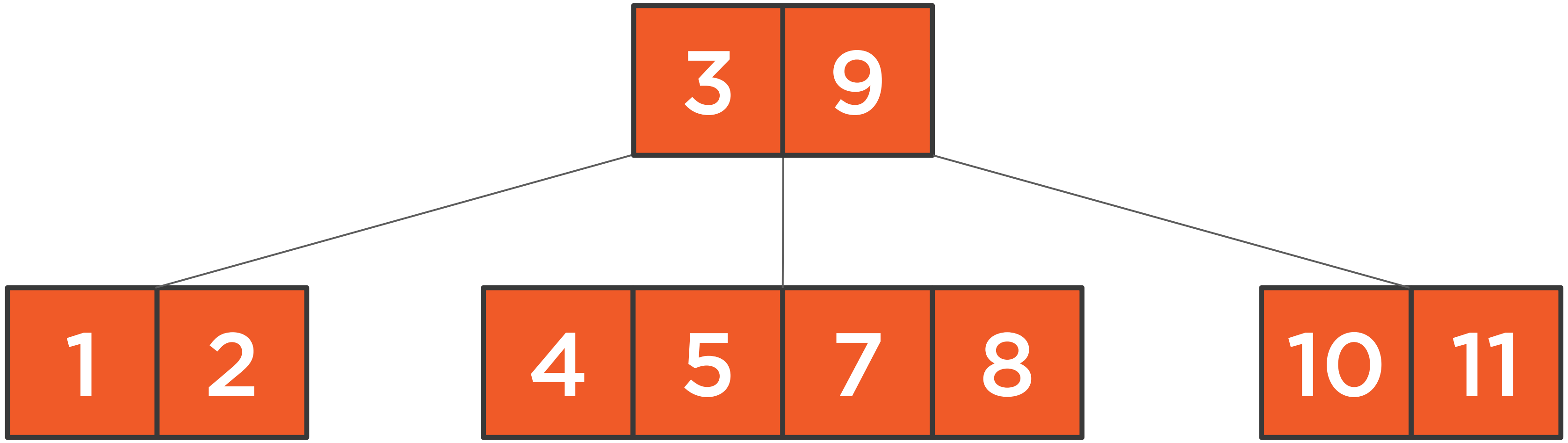
Rotation



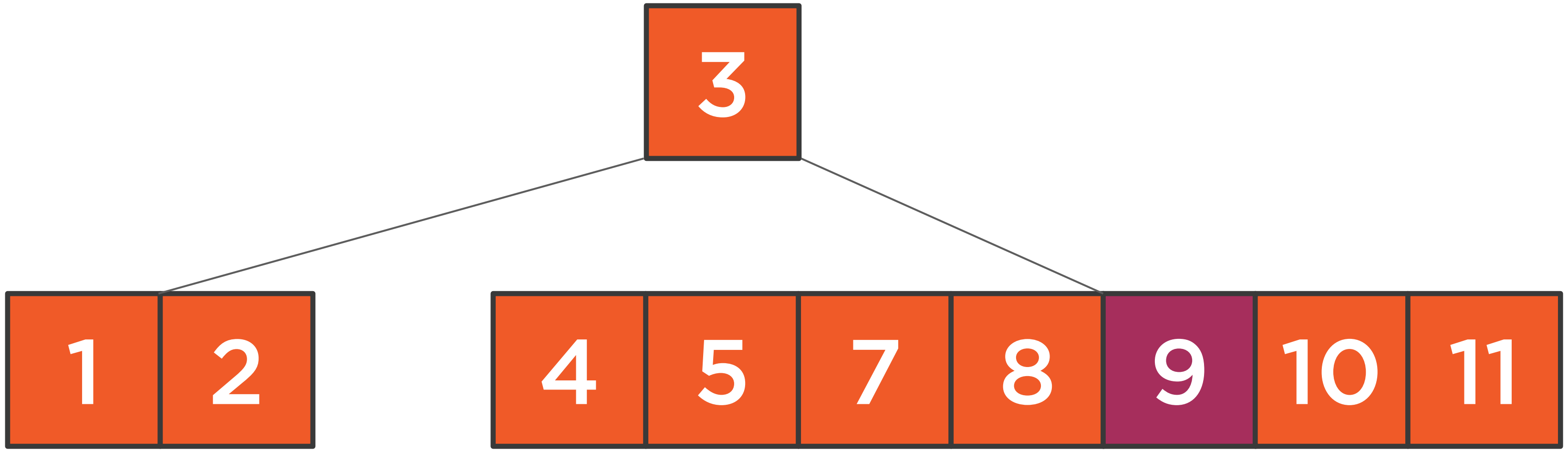
Rotation



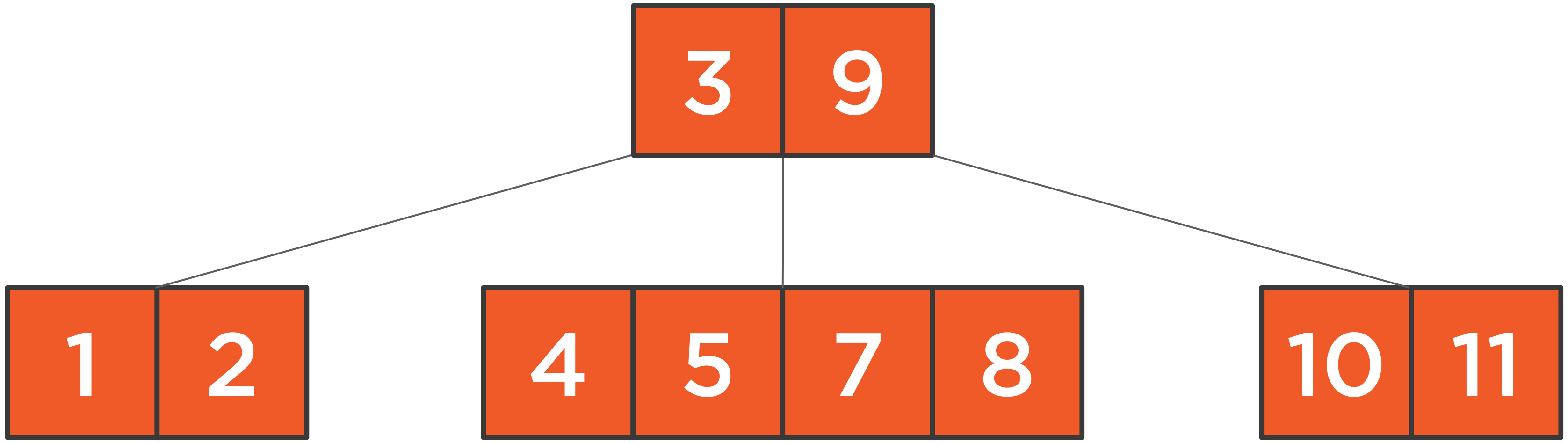
Rotation



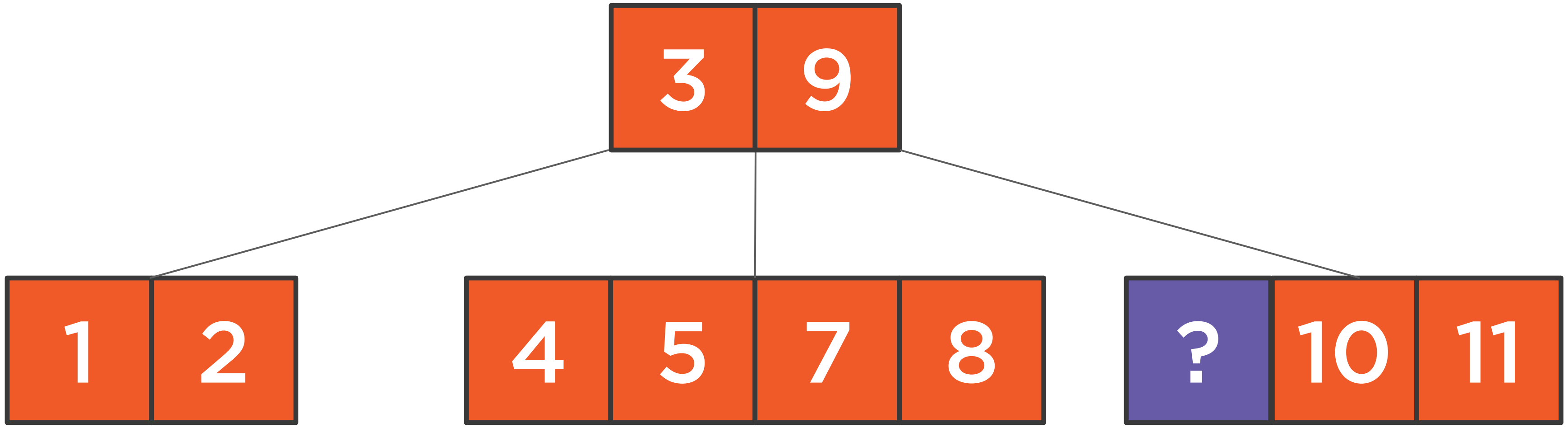
Rotation



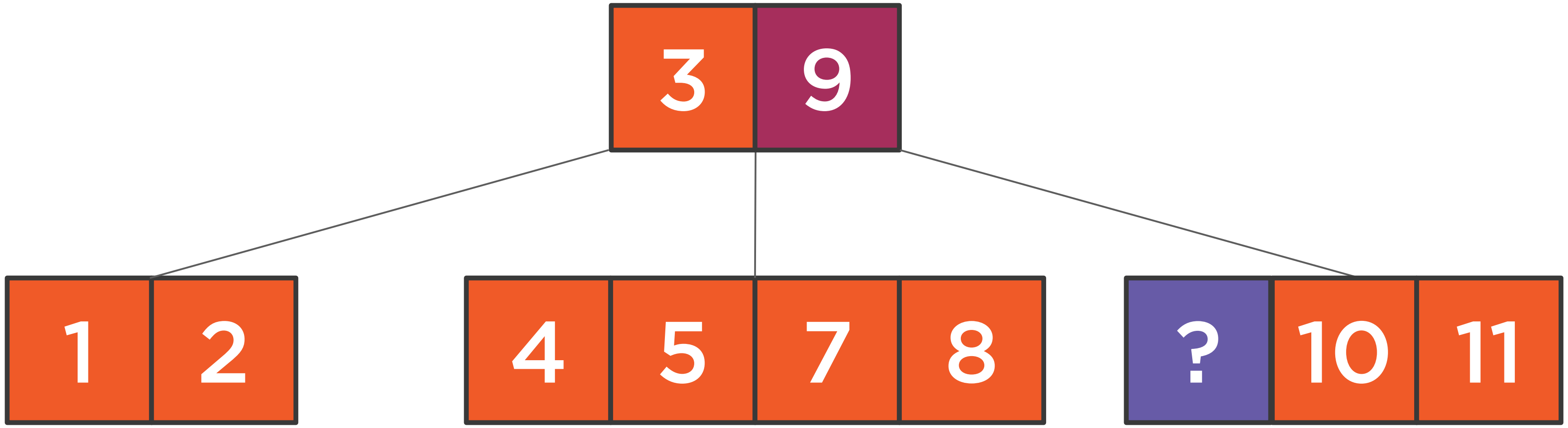
Right Rotation



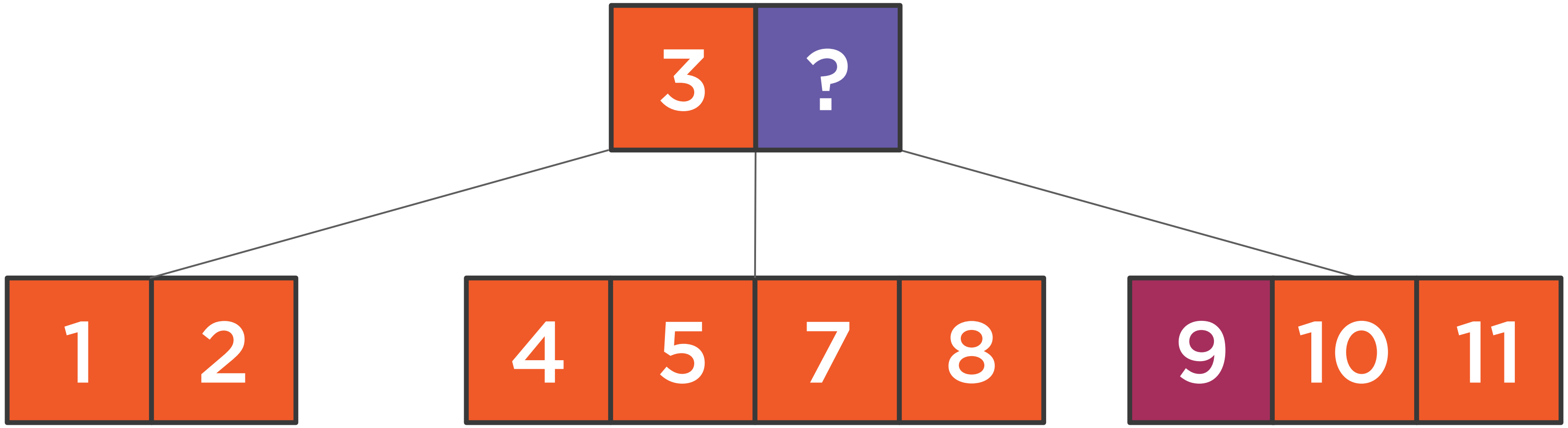
Right Rotation



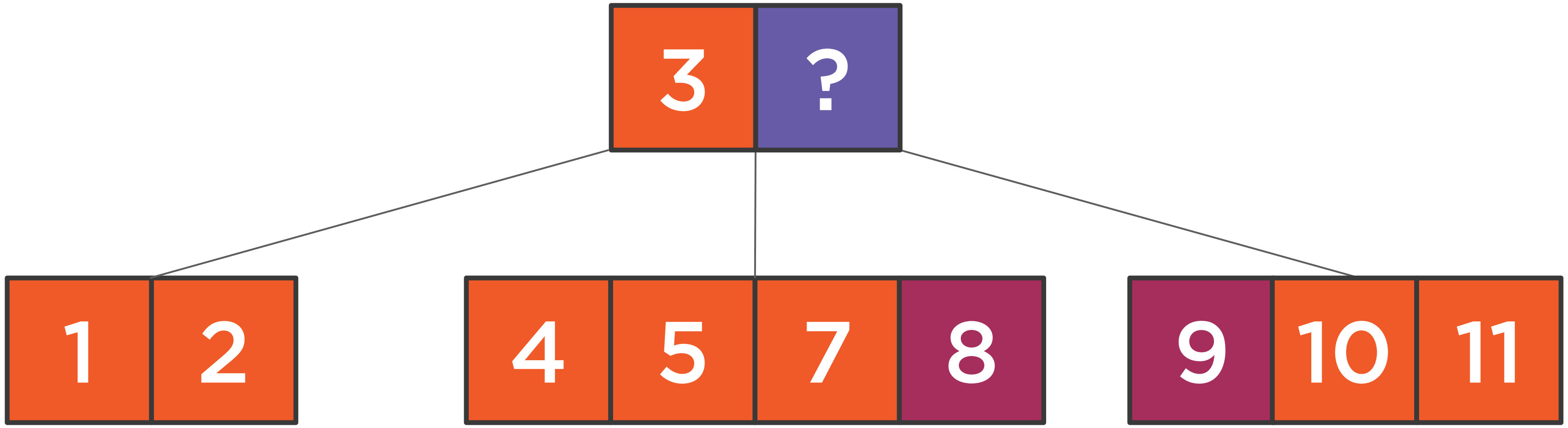
Right Rotation



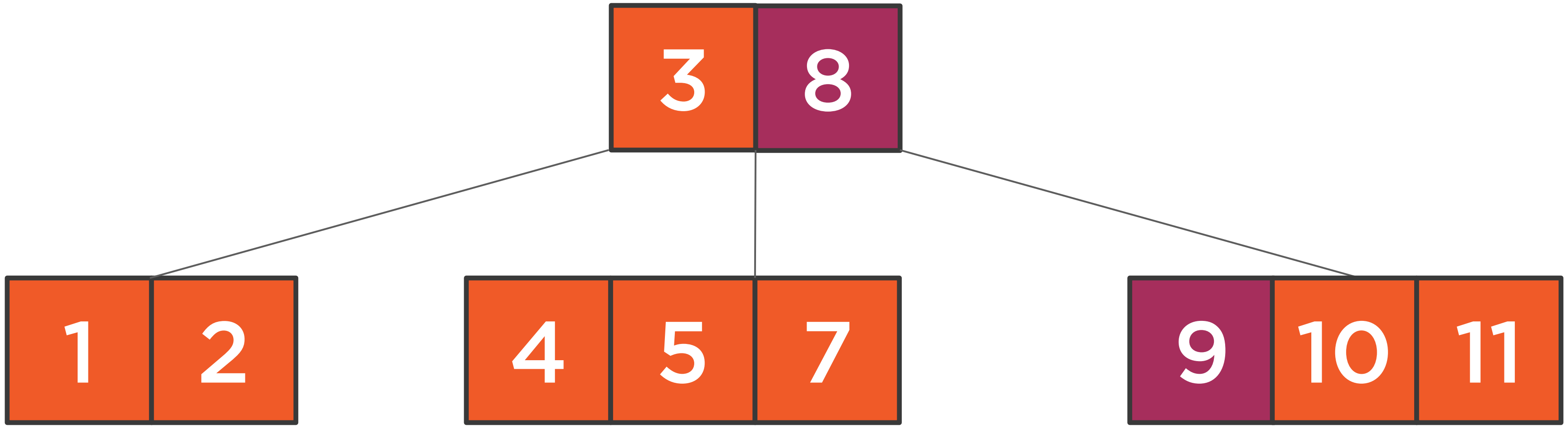
Right Rotation



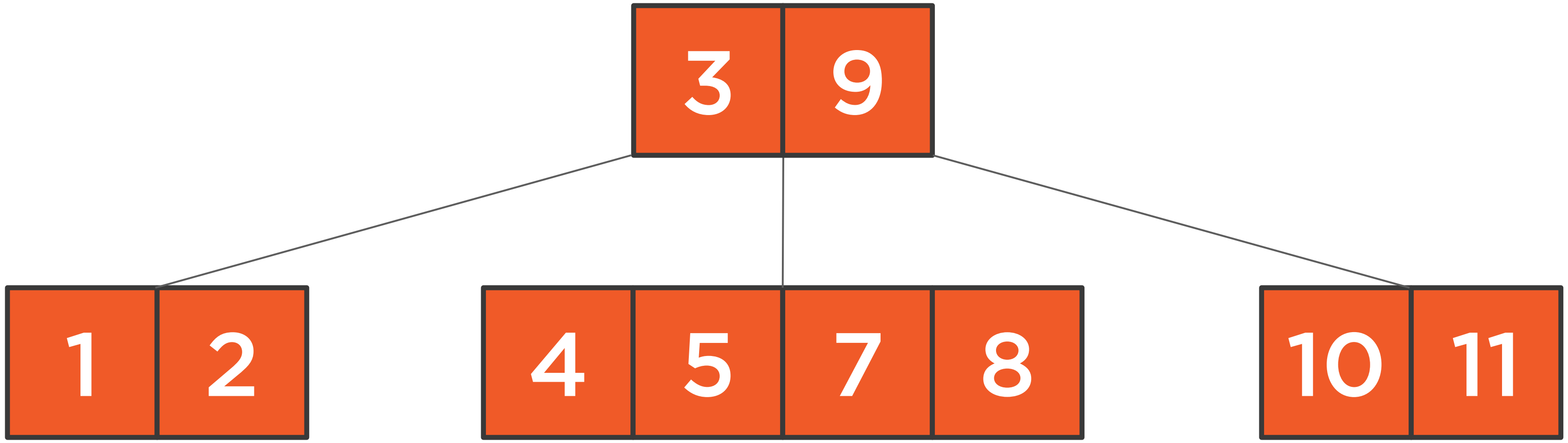
Right Rotation



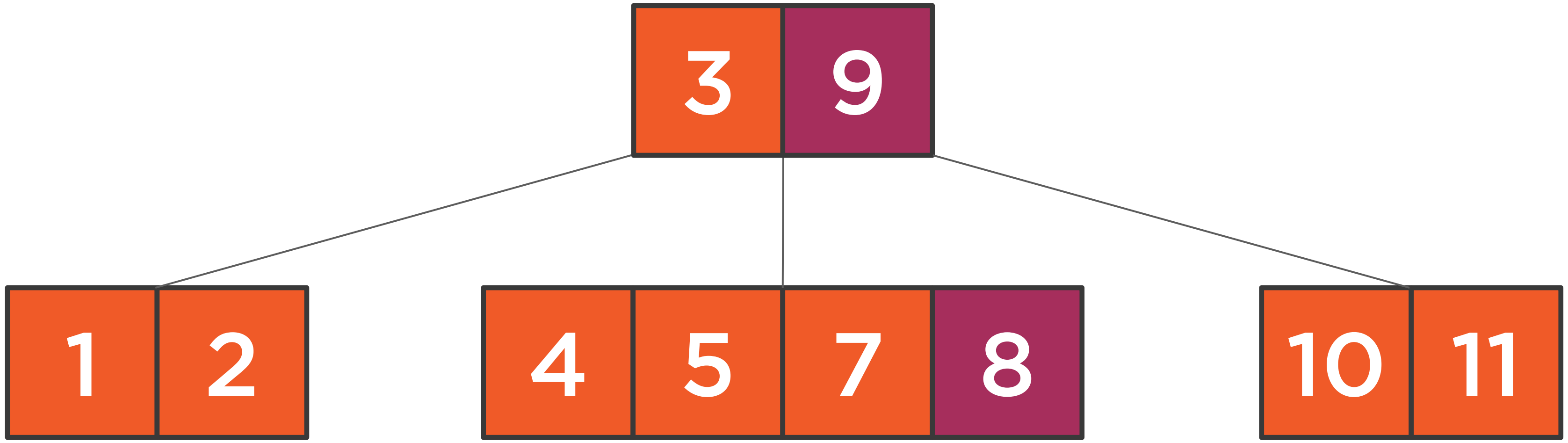
Right Rotation



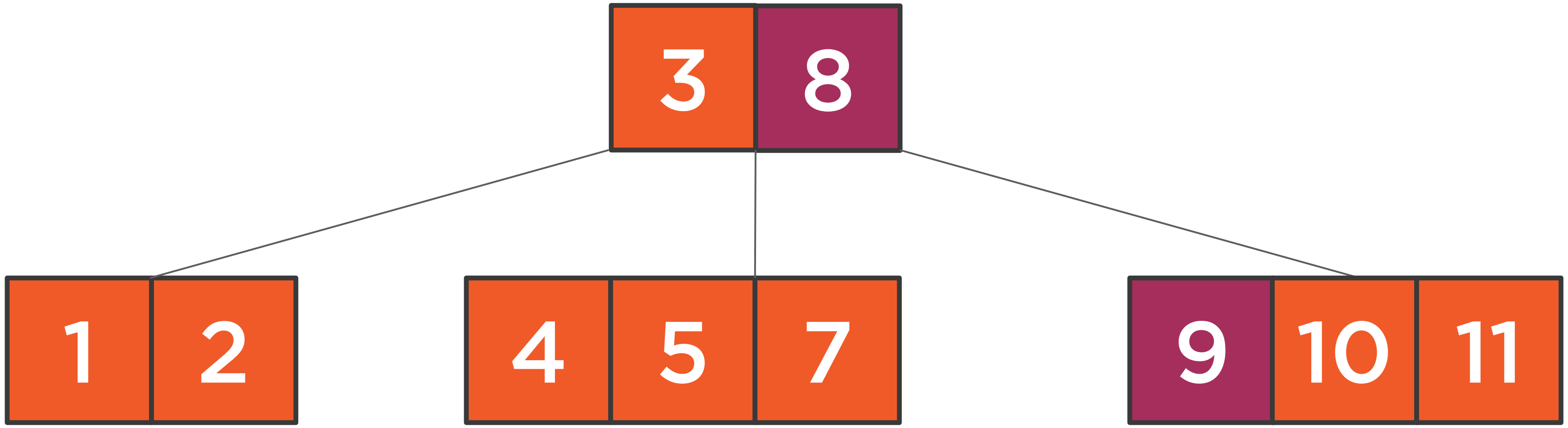
Right Rotation



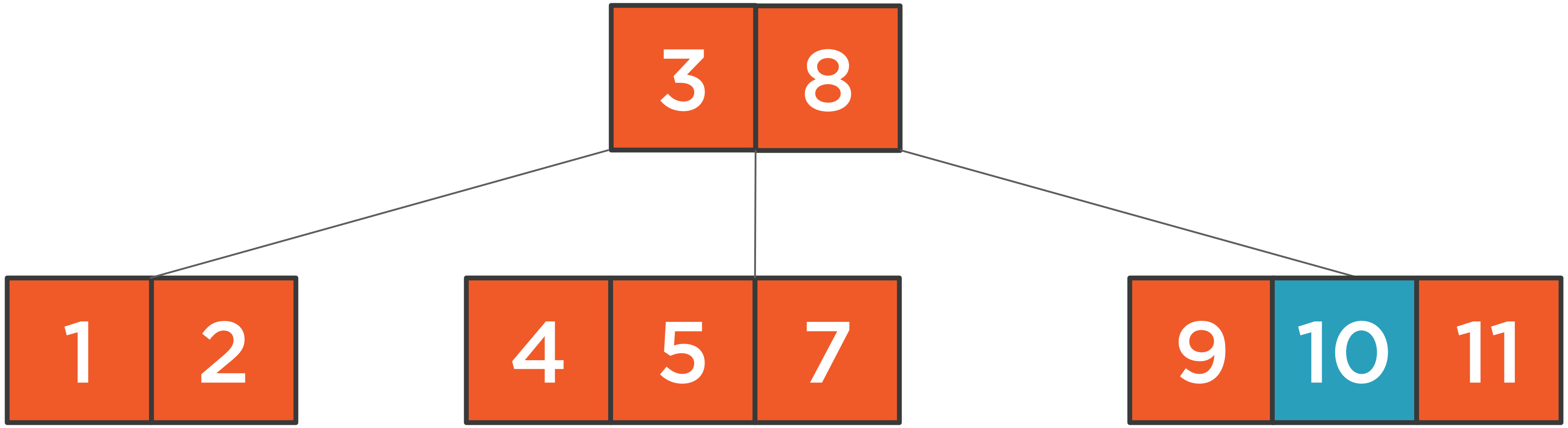
Right Rotation



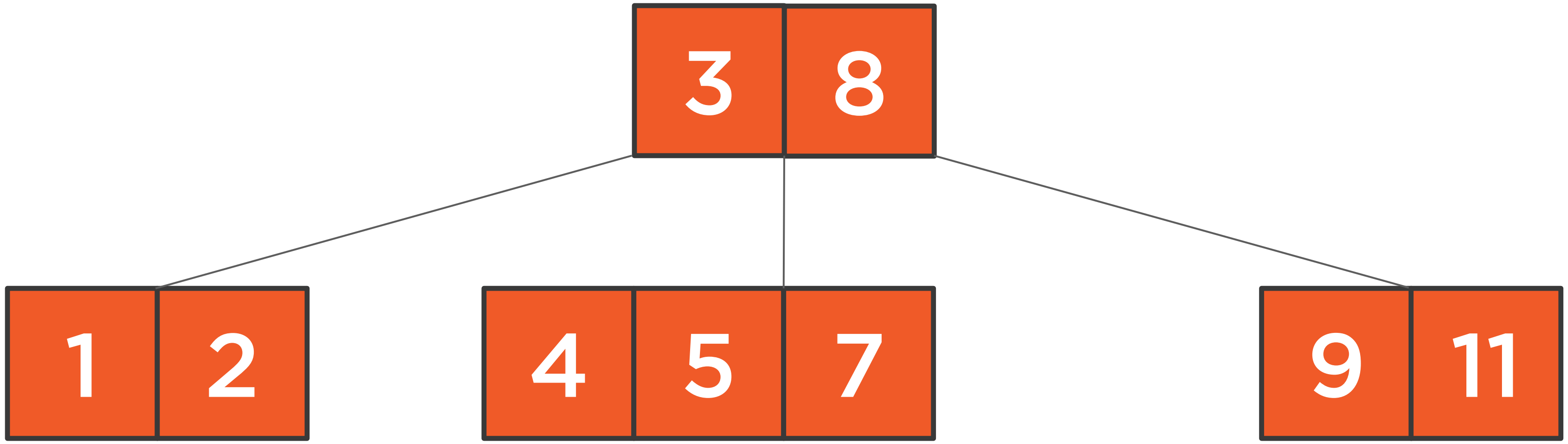
Right Rotation



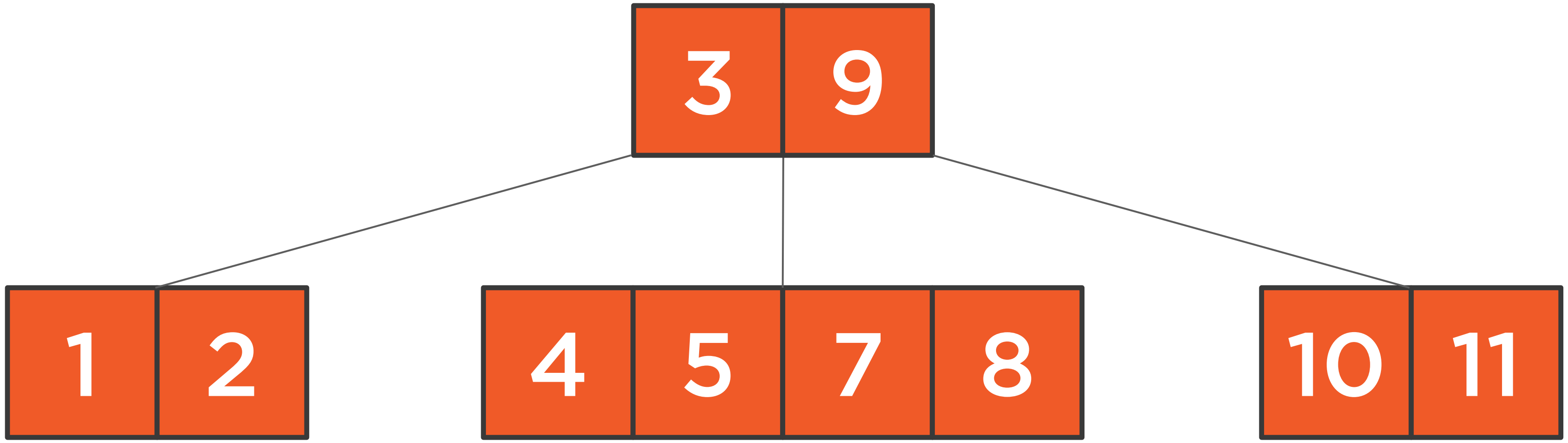
Right Rotation



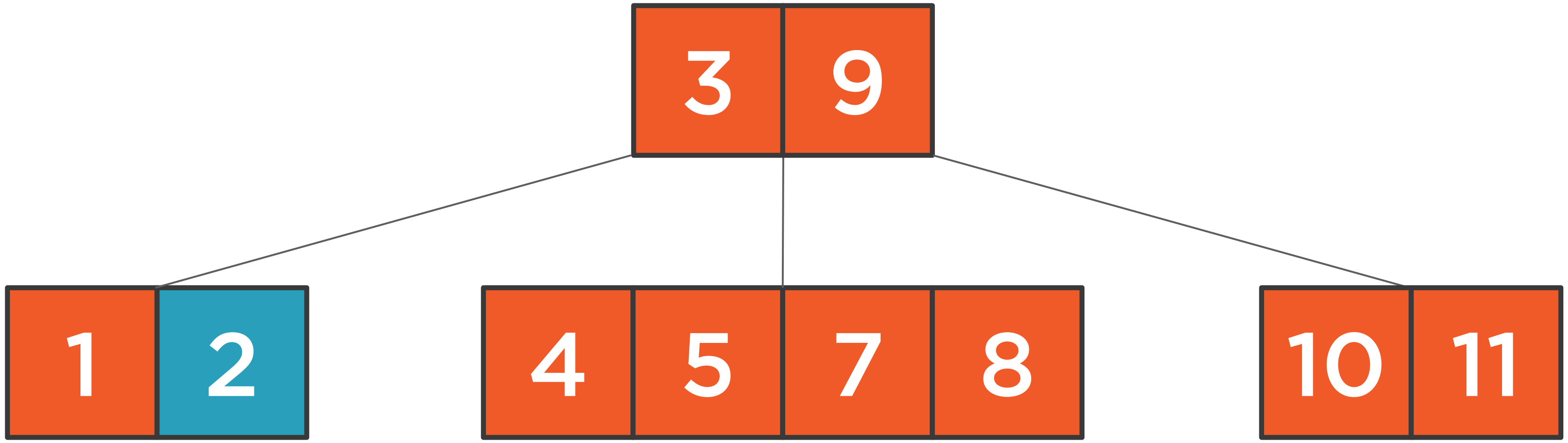
Right Rotation



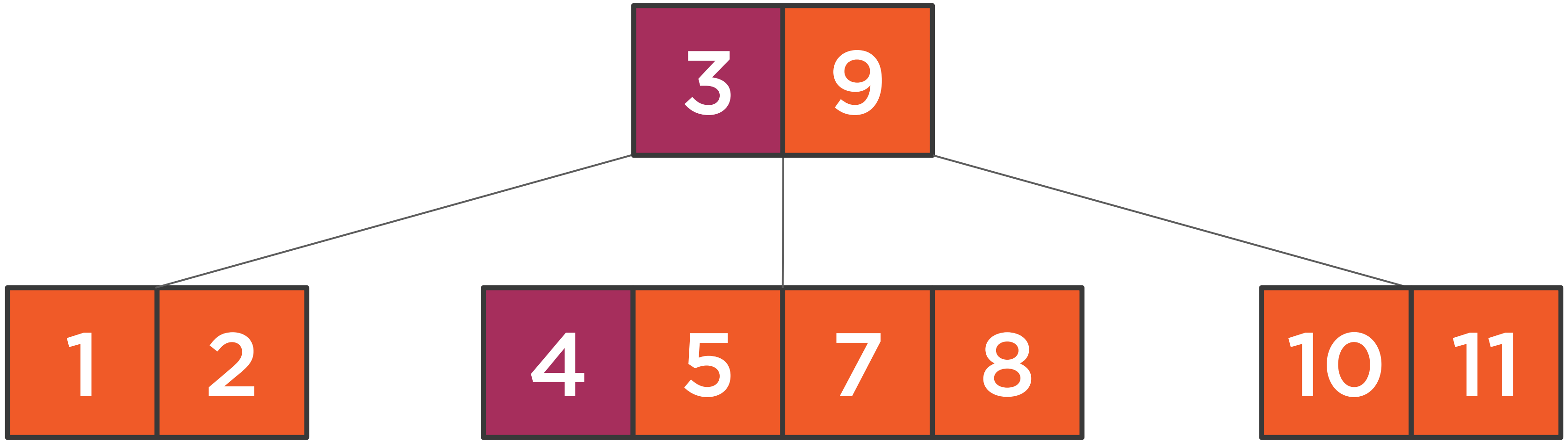
Left Rotation



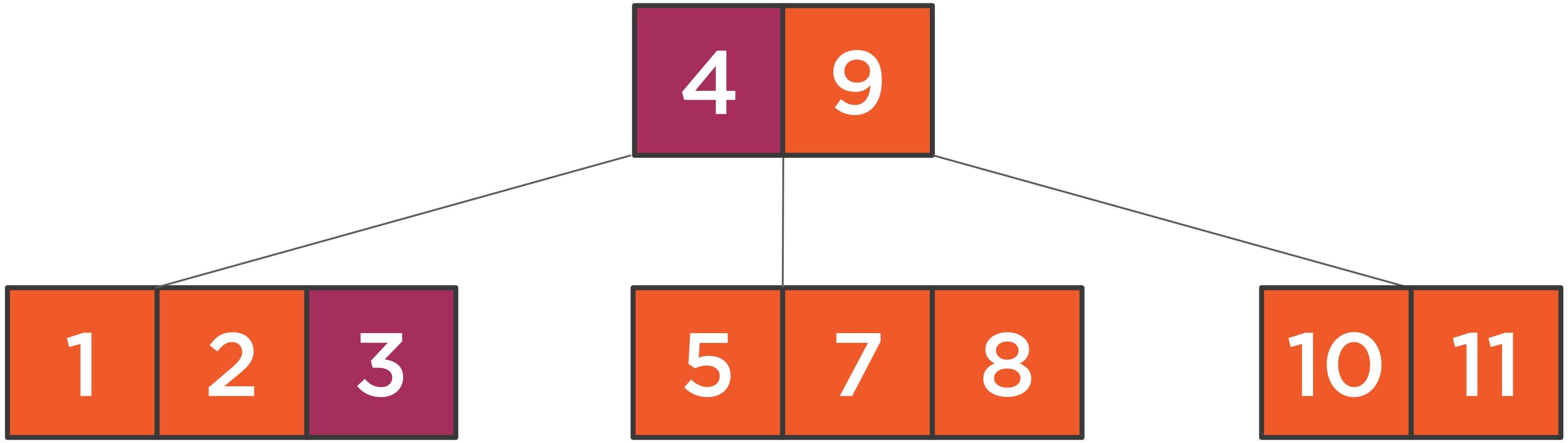
Left Rotation



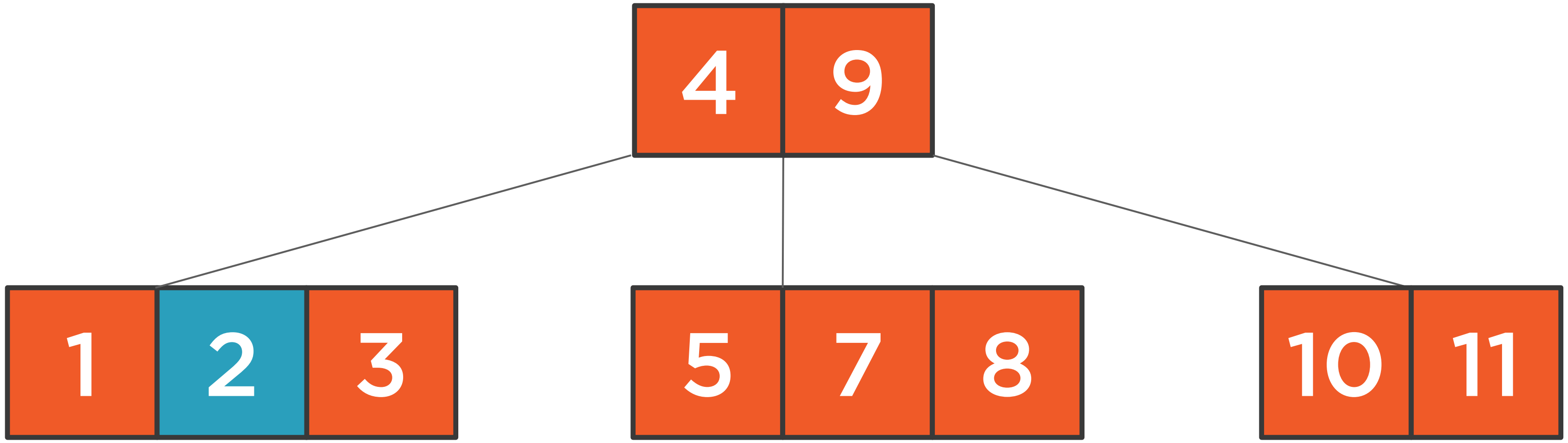
Left Rotation



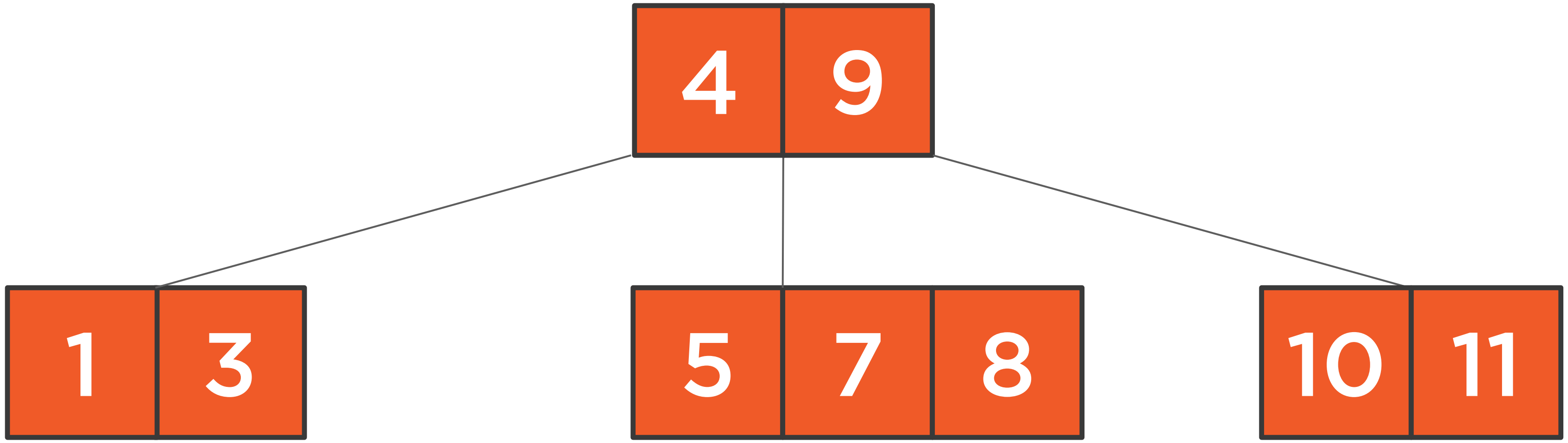
Left Rotation



Left Rotation



Left Rotation



Balancing Operations

Pushing Down
Used when removing a value would leave a child node with too few values, the parent has multiple values, and the sum of the adjacent child and the current child is less than 2*T-1.
Rotation
Used when removing a node and pushing down would not work because the parent has too few values, but a sibling has a value to spare.
Splitting Nodes
Used when adding a node and the destination node would have too many values.

Demo

Sample B-tree Implementation

- Search
- Add
- Remove
- Balancing Operations