2-3 trees

CMSC 420

Maintain the pros, kill the cons

- Is there any way we can maintain mission-critical logarithmic worst-case complexity for search, while simultaneously shedding the storage cost and rotation overhead of AVL Trees?

Maintain the pros, kill the cons

- Is there any way we can maintain mission-critical logarithmic worst-case complexity for search, while simultaneously shedding the storage cost and rotation overhead of AVL Trees?
- Yup! Three solutions:
  - 2-3 (*"two-three"*) trees (usually implemented via Red-Black Binary Search Trees (RBBSTs))
  - B-Trees (short, fat)
  - SkipLists

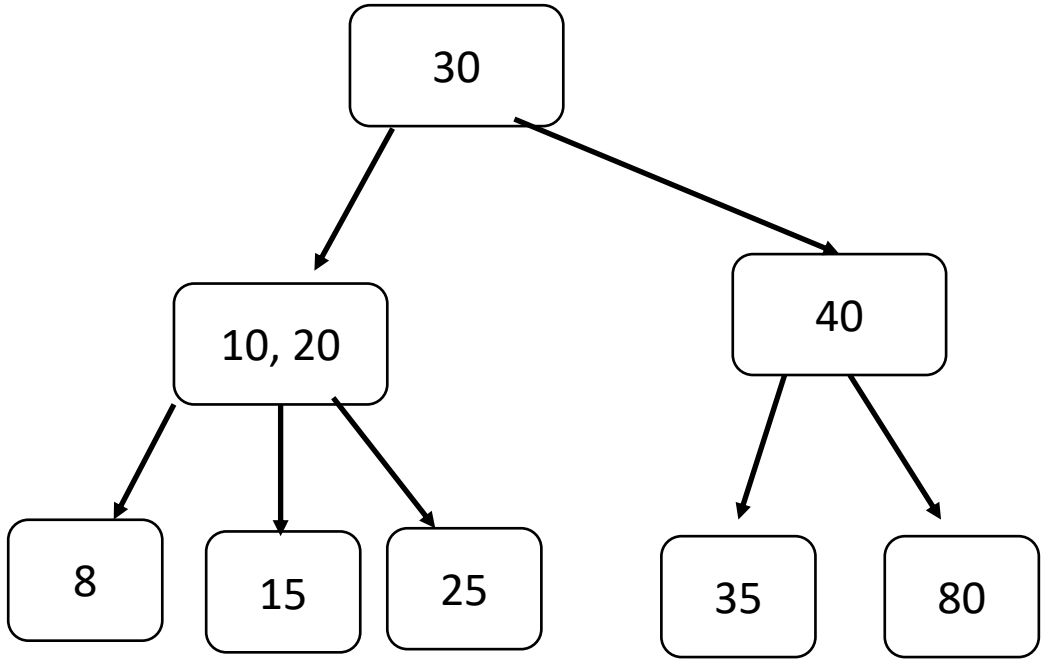
Maintain the pros, kill the cons

- Today, we examine the simplest B-tree, known as a 2-3 tree
- When done, we will discuss an interesting alternative implementation of it as an RBBST.

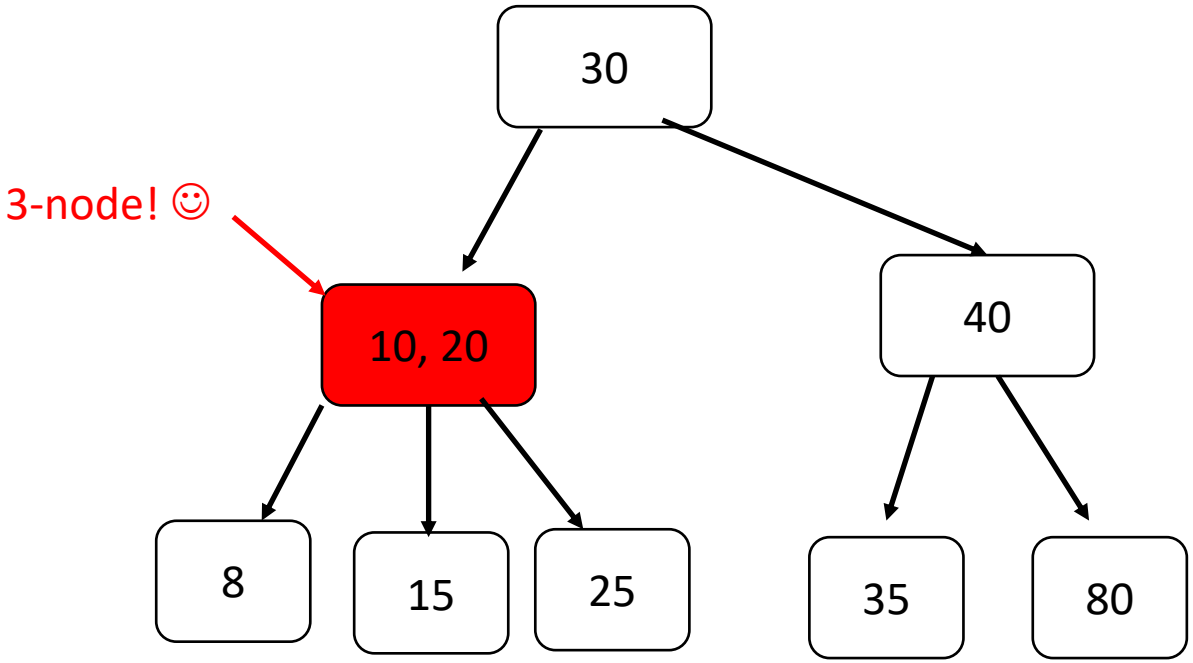
2-3 Trees

- In addition to their other properties, 2-3 trees (and B-trees) will have **perfect balance** all the time!
- Core idea: 2 children per node are ok, more are better (for height).
- 2-3 trees will have two different nodes inside them!
  a) A **BST-like node** with just one key, and two pointers to subtrees **(a 2- node)**
  b) An expanded node with two keys and three pointers to subtrees **(a 3-node)**
     - Leftmost subtree contains keys < Key_1
     - Center subtree contains keys in $[Key_1, Key_2)$
     - Rightmost subtree contains keys $> Key_2$

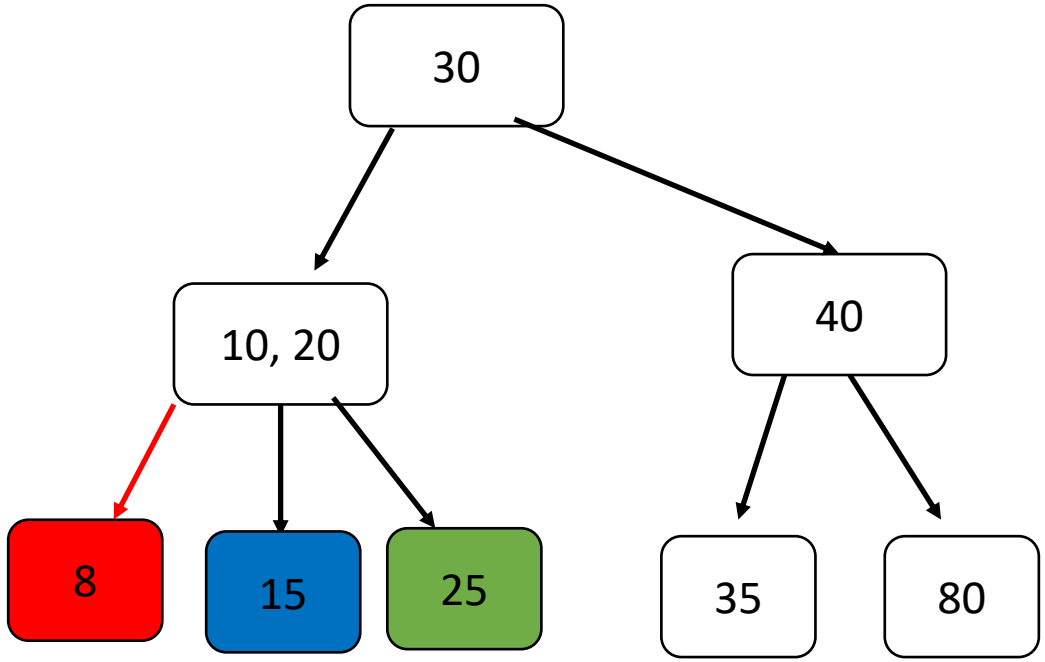
2-3 Tree example



2-3 Tree example



Search



- Searching a 2-3-Tree is done in exactly the same way as a BST.
- Go left, between, or right!

Insertion

- Let’s recall insertion in AVL trees…
- ***Might*** trigger rotations to re-balance the tree according to the AVL condition.
- Tree **not** guaranteed to be perfect.
  - Only exception: Insertion places node in the only empty position at the leaves.

Insertion

- Insertion on both of these trees has two phases…
  1. A search phase, where we go down to search for an empty position to put the key in (if it's not already in there), and
  2. A rotation phase where we go up to rebalance subtrees.

Insertion

- Insertion on both of these trees has two phases…
  1. A search phase, where we go down to search for an empty position to put the key in (if it's not already in there), and
  2. A rotation phase where we go up to rebalance subtrees.
- 2-3 trees never rotate ***nodes***. Never. Ever. Instead…

Insertion

- Insertion on both of these trees has two phases…
  1. A search phase, where we go down to search for an empty position to put the key in (if it's not already in there), and
  2. A rotation phase where we go up to rebalance subtrees.
- 2-3 trees never rotate ***nodes***. Never. Ever. Instead…
  - They rotate **keys**…. *(we will disallow this a few lectures down the road)*

Insertion

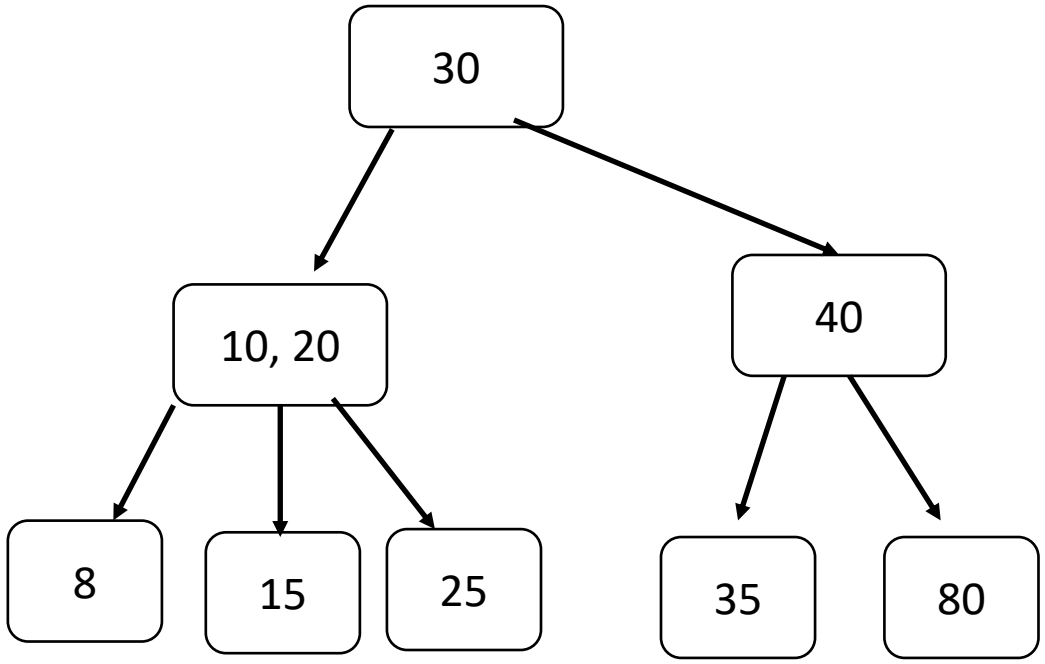
- Insertion on both of these trees has two phases…
  1. A search phase, where we go down to search for an empty position to put the key in (if it's not already in there), and
  2. A rotation phase where we go up to rebalance subtrees.
- 2-3 trees never rotate ***nodes***. Never. Ever. Instead…
  - They rotate **keys**…. *we will disallow this a few lectures down the road)*
  - They also sometimes make nodes fatter or slimmer…

Insertion

- Insertion on both of these trees has two phases…
  1. A search phase, where we go down to search for an empty position to put the key in (if it's not already in there), and
  2. A rotation phase where we go up to rebalance subtrees.
- 2-3 trees never rotate ***nodes***. Never. Ever. Instead…
  - They rotate **keys**…. *(we will disallow this a few lectures down the road)*
  - They also sometimes make nodes fatter or slimmer…
  - Also, they sometimes make new nodes appear or old ones disappear

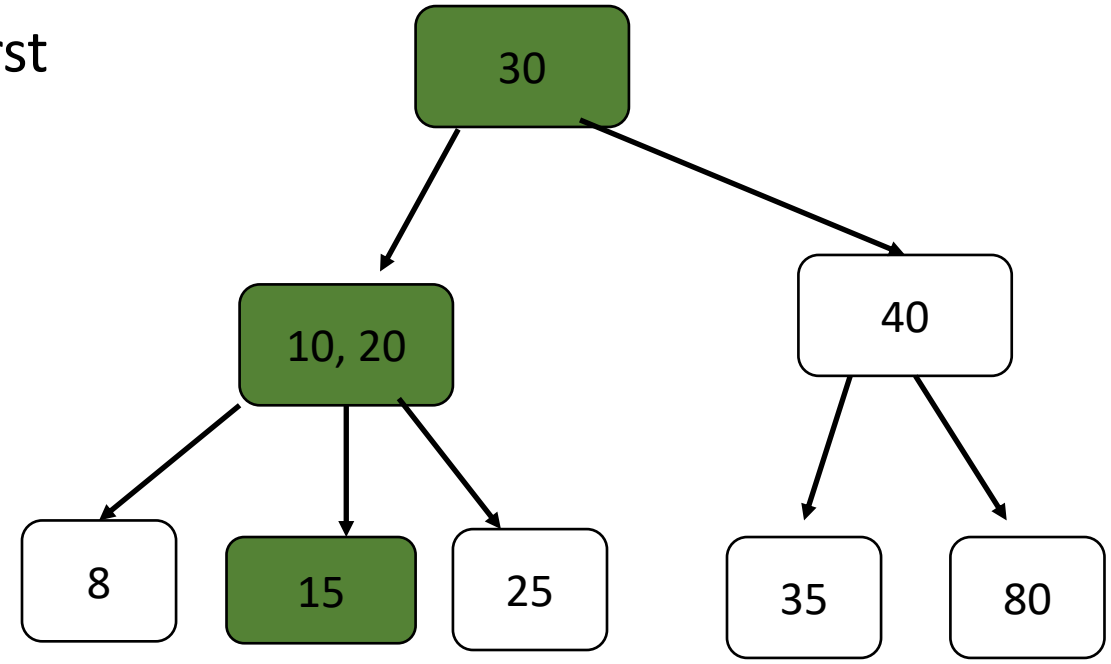
Insertion

- Insert 18



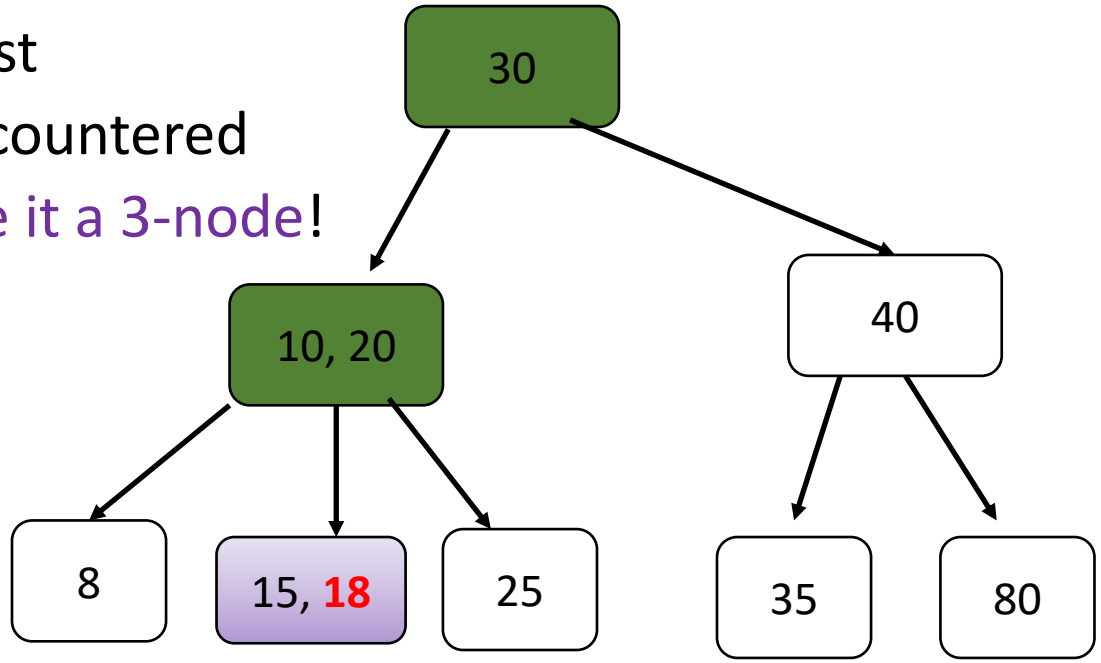
Insertion

- Insert 18
  1. We search first



Insertion

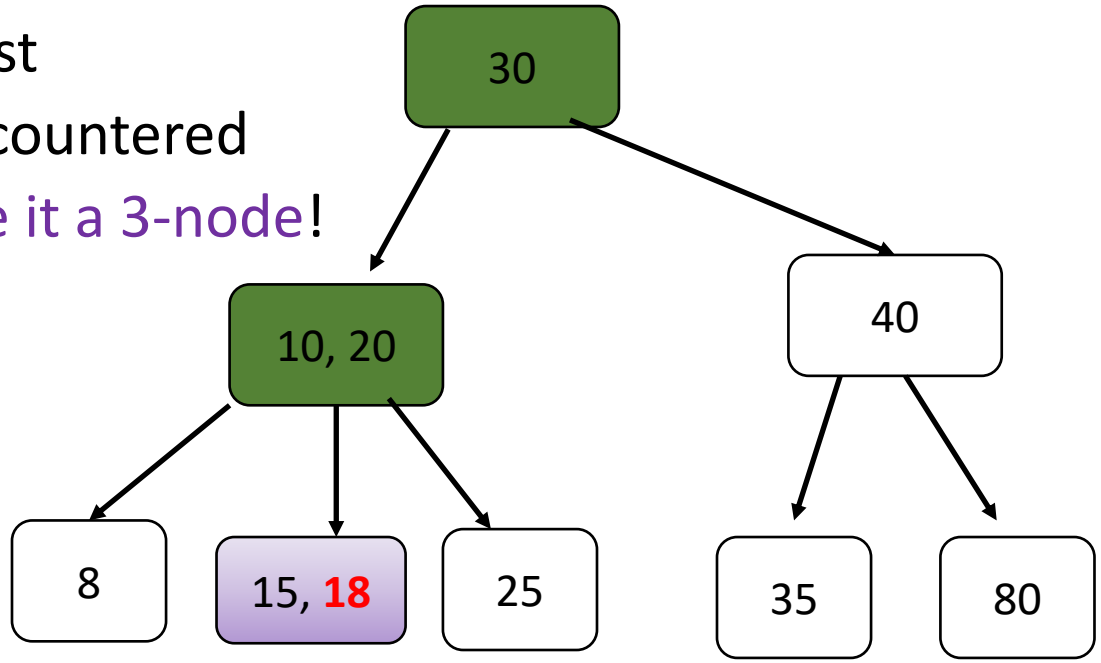
- Insert 18
  1. We search first
  2. Last node encountered
     is 2-node: make it a 3-node!



Insertion

- Insert 18
  1. We search first
  2. Last node encountered

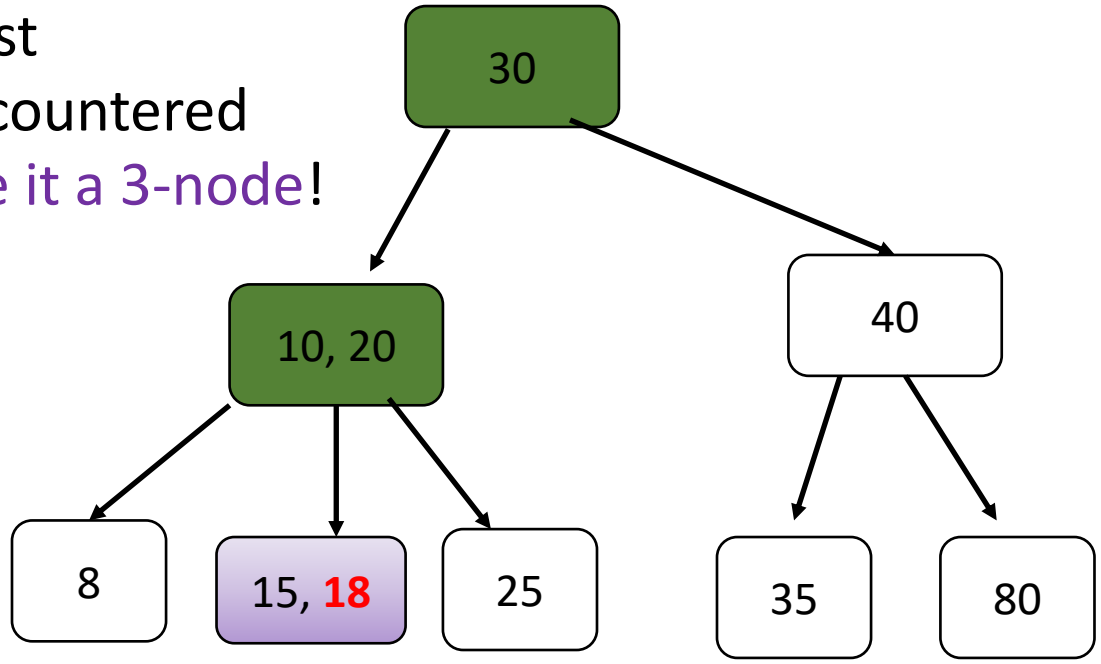
     is 2-node: make it a 3-node!



- Done! ☺

Insertion

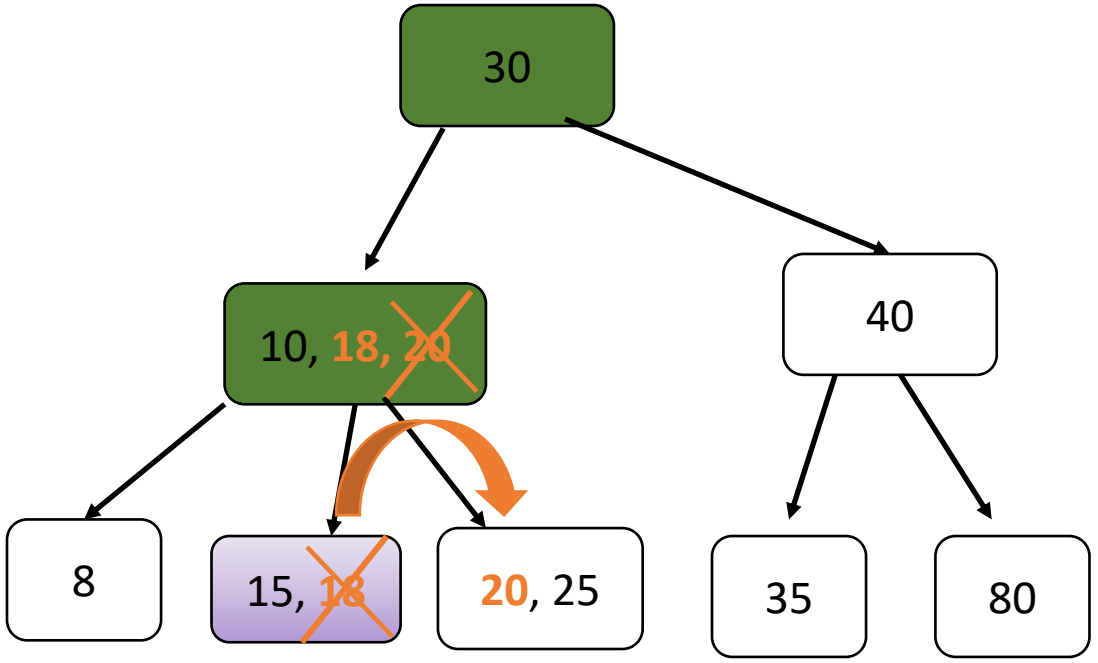
- Insert 18
  1. We search first
  2. Last node encountered is 2-node: make it a 3-node!



- Done! ☺

- But what if I now wanted to insert 17? I don't have space for it!

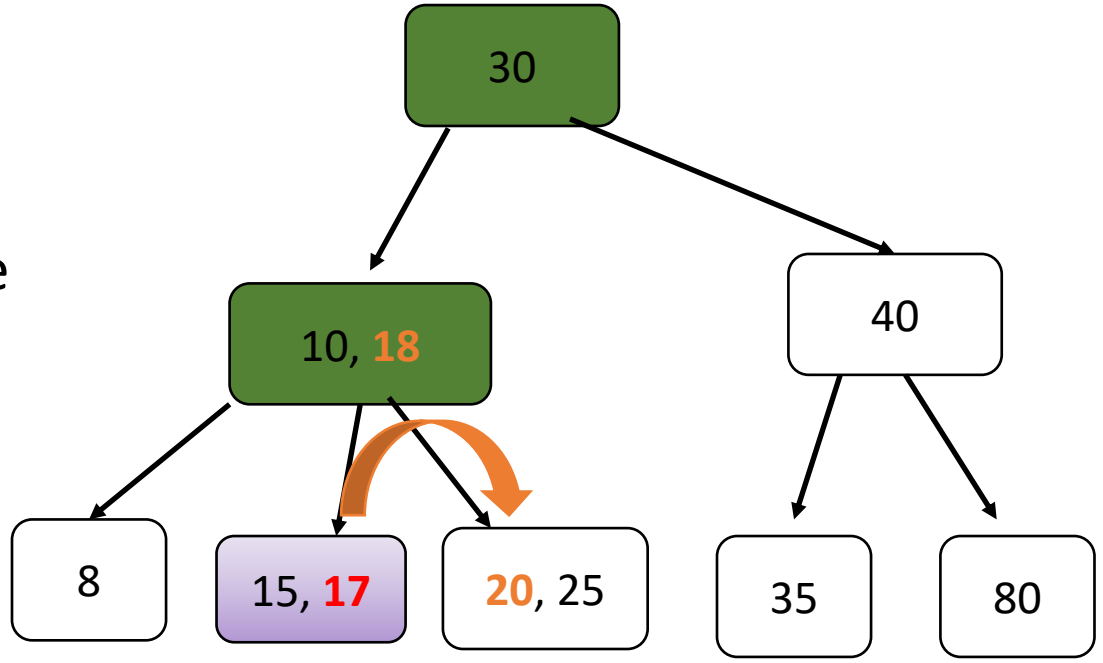
Insertion: Key Rotations



- But what if I now wanted to insert 17? I don't have space for it!
  - Solution: **Rotate 18 and 20 to the right to make space!**
  - Could also rotate 15 and 10 to the left instead.

Insertion: Key Rotations

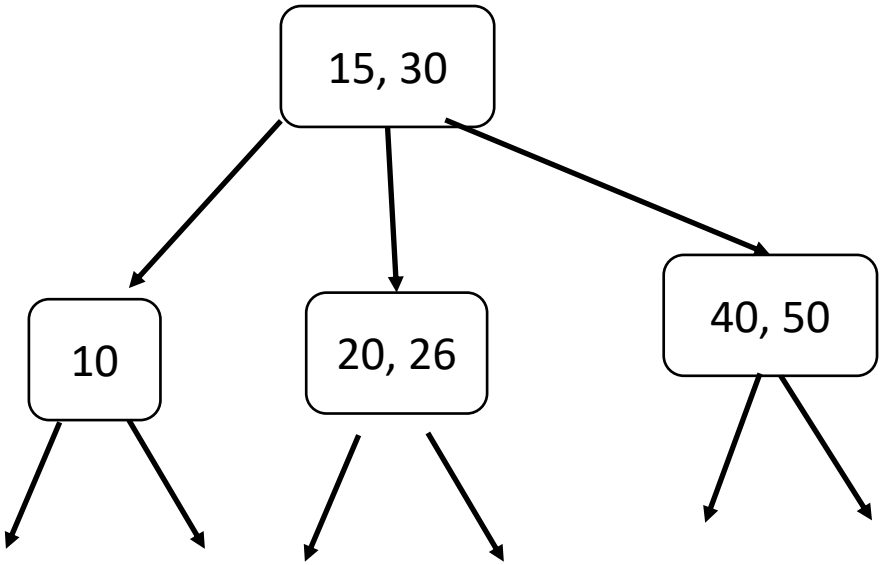
Now the **node** has space for **17,** so we can insert it ☺



- But what if I now wanted to insert 17? I don't have space for it!
  - Solution: **Rotate 18 and 20 to the right to make space!**
  - Could also rotate 15 and 10 to the left instead.

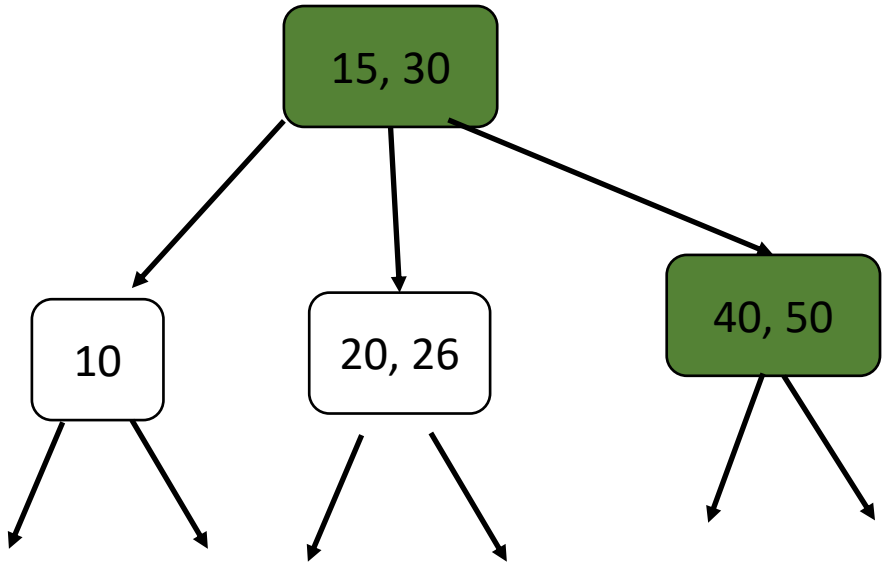
Insertion: Key Rotations

- Let’s now insert 43 in this tree…



Insertion: Key Rotations

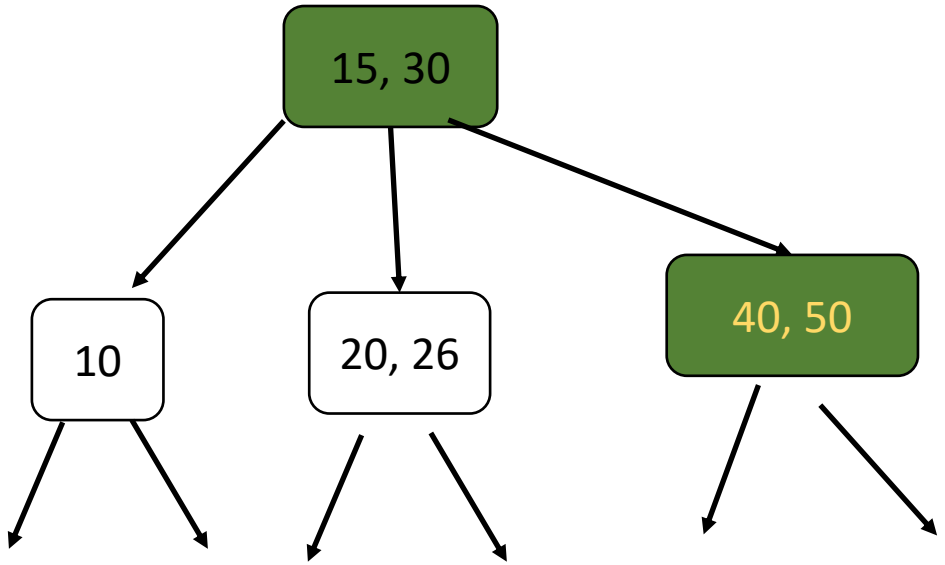
- Let's now insert 43 in this tree…



Searching leads us to the right child…

Insertion: Key Rotations

- Let's now insert 43 in this tree…

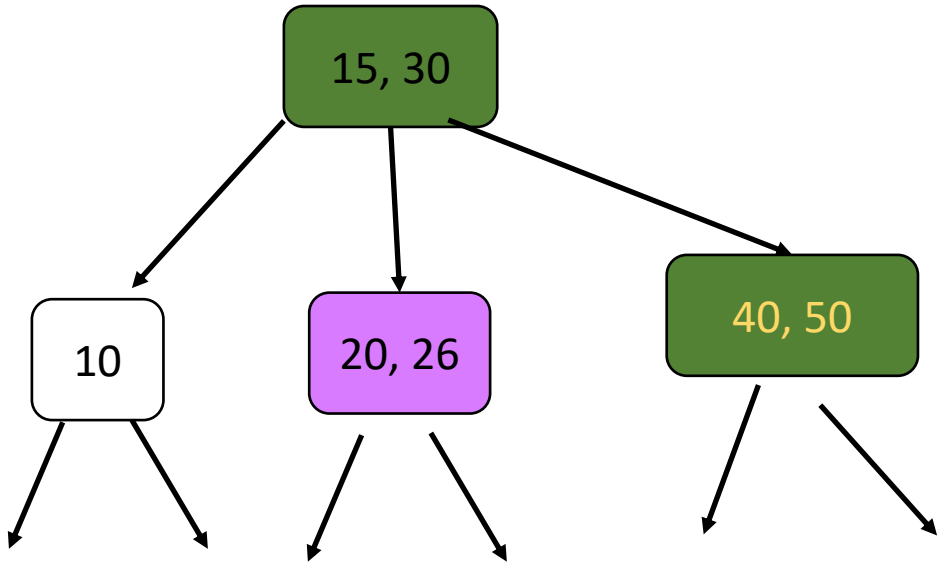


Searching leads us to the right child…

But there's no space ☹

Insertion: Key Rotations

- Let's now insert 43 in this tree…



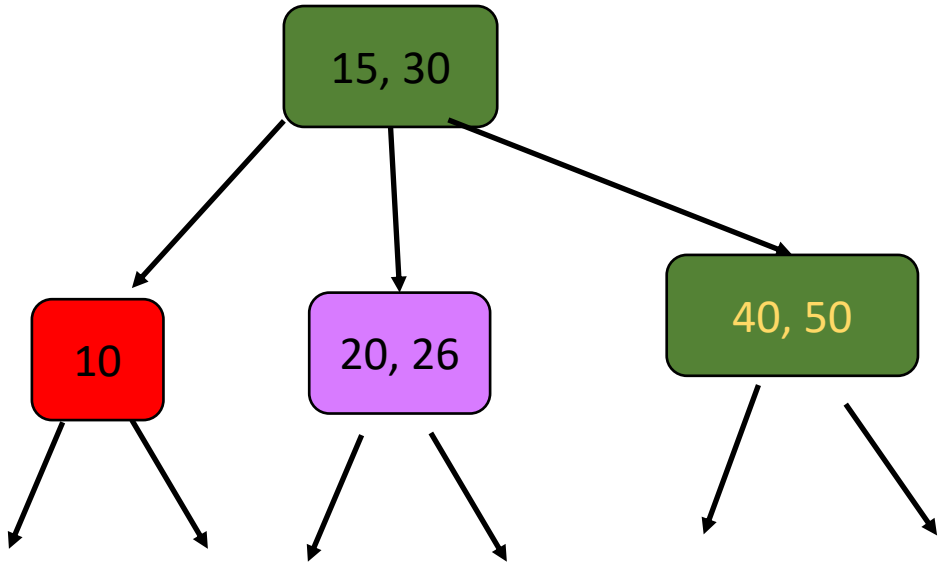
Searching leads us to the right child…

But there's no space ☹

My sibling's also out of space! ☹

Insertion: Key Rotations

- Let's now insert 43 in this tree…



Searching leads us to the right child…

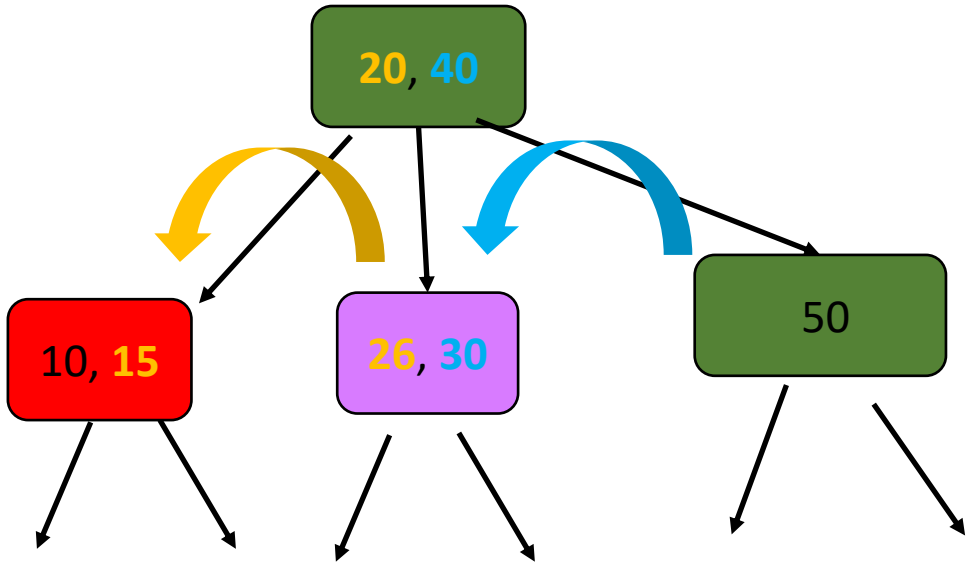
But there's no space ☹

My sibling's also out of space! ☹

But my "second" sibling does!

Insertion: Key Rotations

- Let's now insert 43 in this tree…



Searching leads us to the right child…

But there's no space ☹

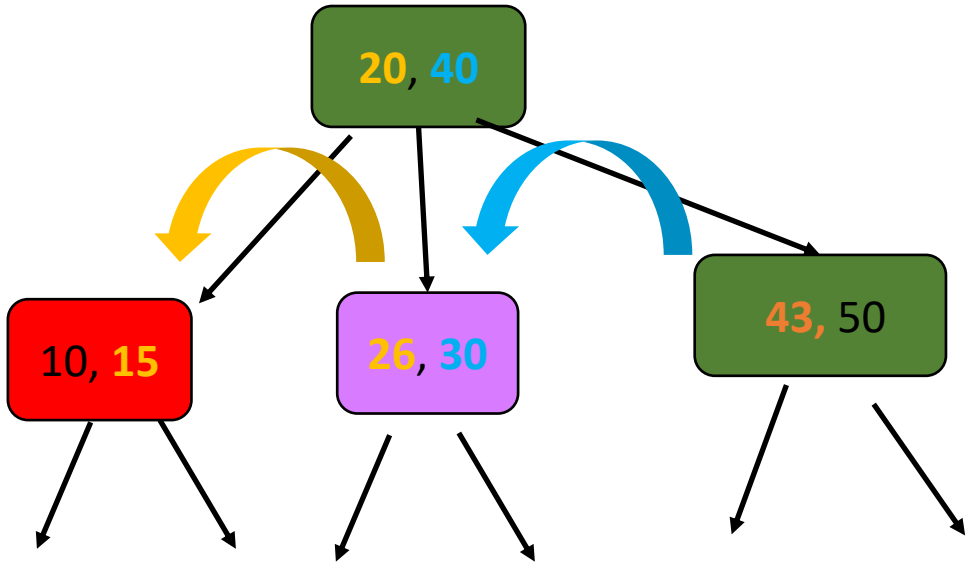
My sibling's also out of space! ☹

But my "second" sibling does!

Rotate twice!

Insertion: Key Rotations

- Let's now insert 43 in this tree…



Searching leads us to the right child…

But there's no space ☹

My sibling's also out of space! ☹

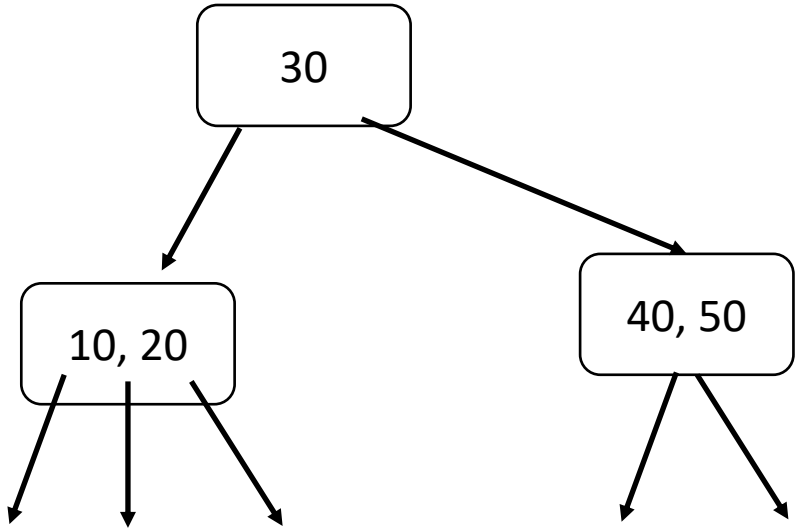
But my "second" sibling does!

Rotate twice!

And now I have space to insert 43! ☺

Insertion

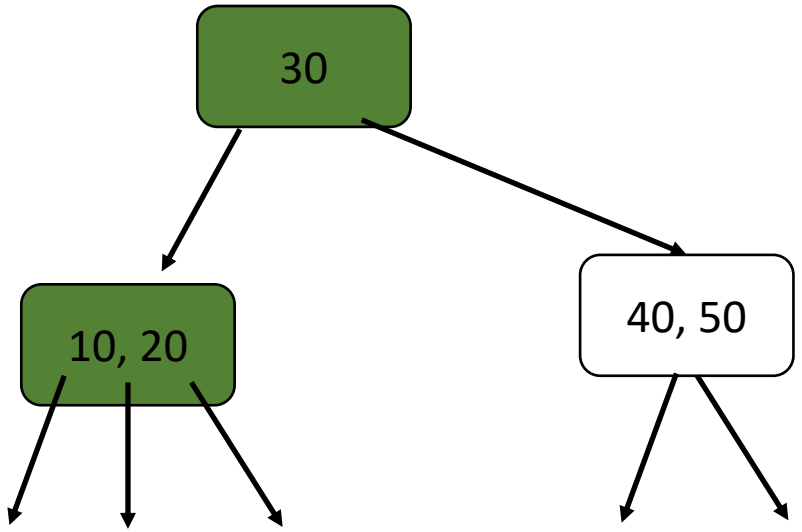
- Suppose we want to Insert **15** in this 2-3 tree…



Insertion

- Suppose we want to Insert **15** in this 2-3 tree…

Searching leads us to the left child of the root…

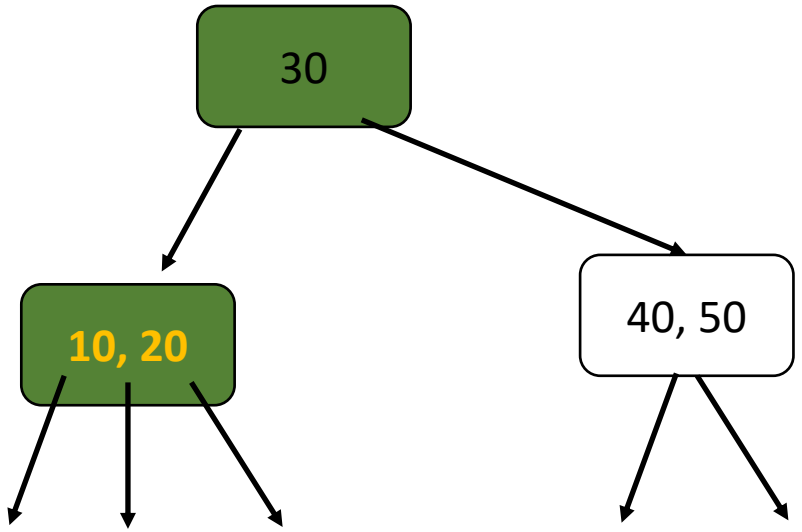


Insertion

- Suppose we want to Insert **15** in this 2-3 tree…

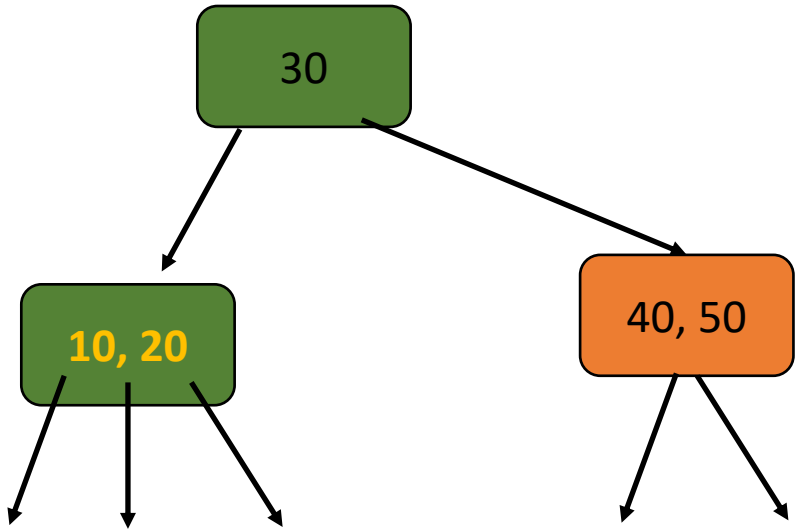
Searching leads us to the left child of the root…

Which doesn't have space ☹



Insertion

- Suppose we want to Insert **15** in this 2-3 tree…



Searching leads us to the left child of the root…

Which doesn't have space ☹

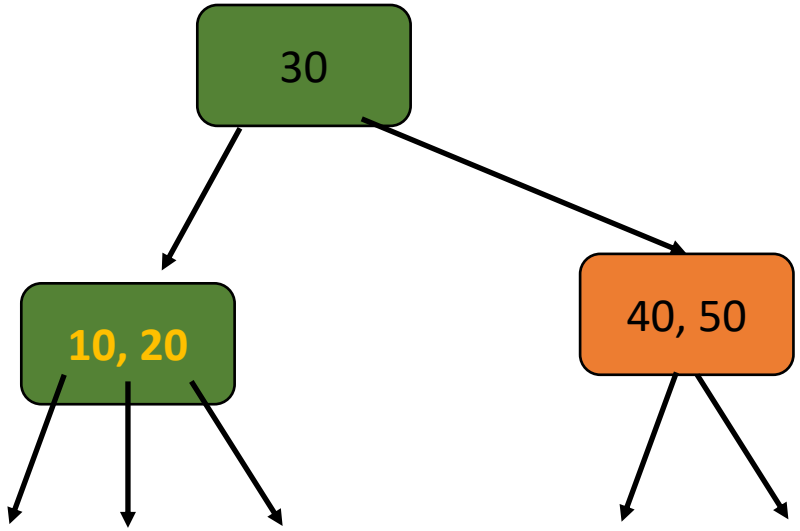
There's only one sibling, and it doesn't have space either ☹

Insertion

- Suppose we want to Insert **15** in this 2-3 tree…

Searching leads us to the left child of the root…

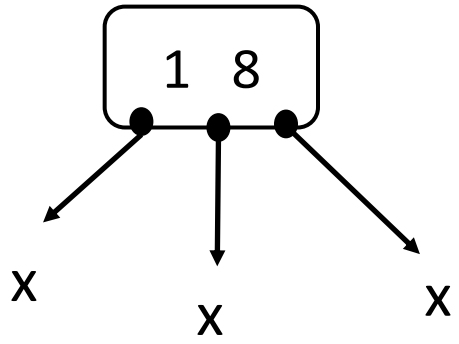
Which doesn't have space ☹



There's only one sibling, and it doesn't have space either ☹

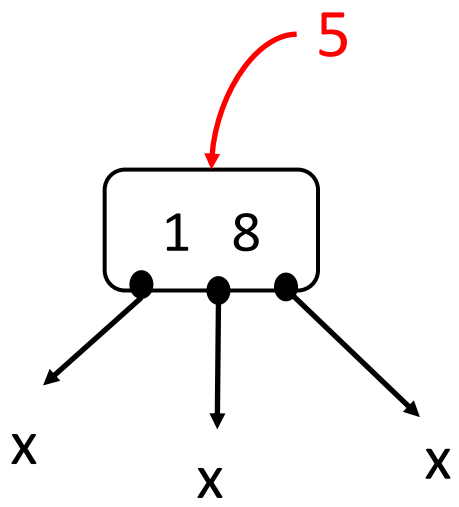
Insertion : Can’t rotate key

- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



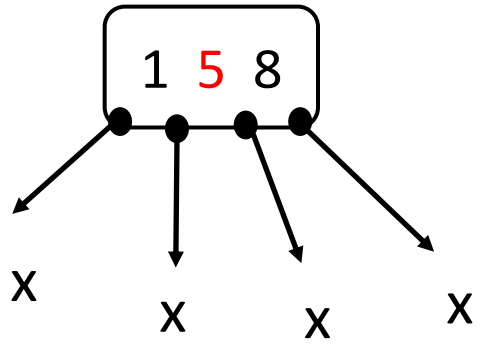
Insertion : Can’t rotate key

- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



Insertion : Can’t rotate key

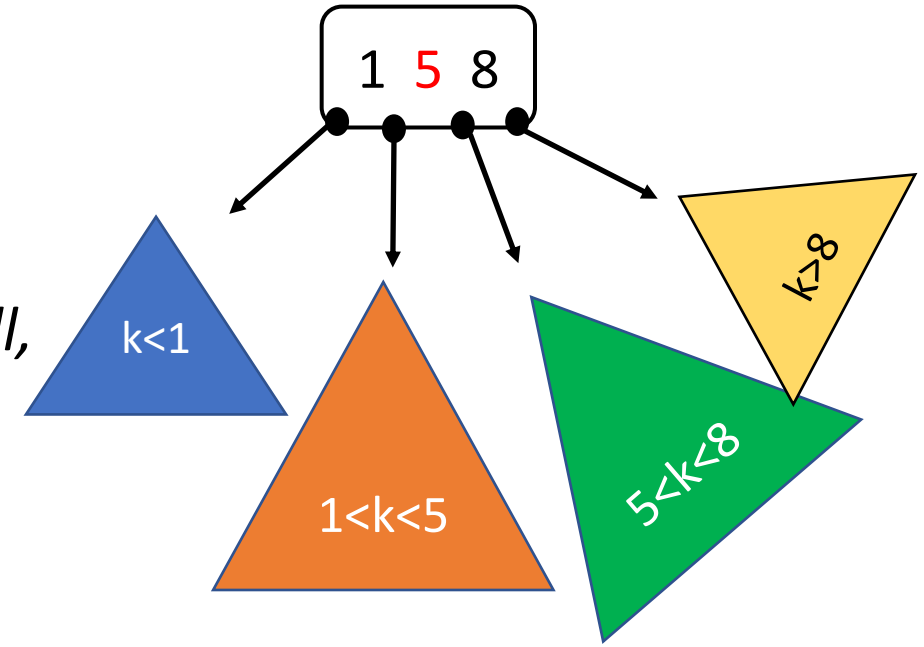
- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



Temporarily assume we can expand the 3-node to a 4-node!

Insertion : Can't rotate key

- To understand what to do in this case, consider inserting 5 in this "stub" 2-3 tree.

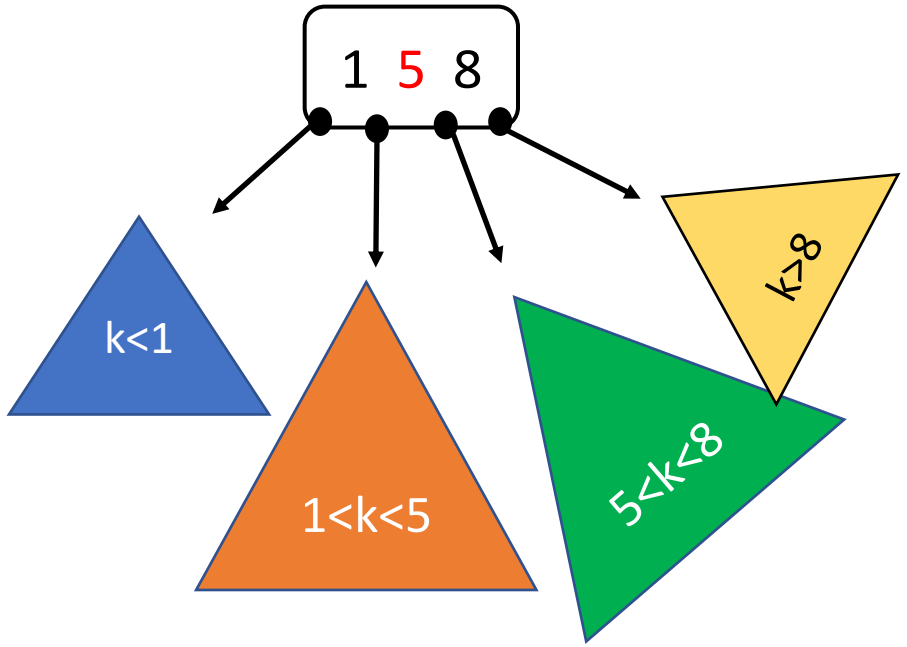


(if the links were non-null, those would be the subtrees / key ranges they would point to)

Temporarily assume we can expand the 3-node to a 4-node!

Insertion : Can’t rotate key

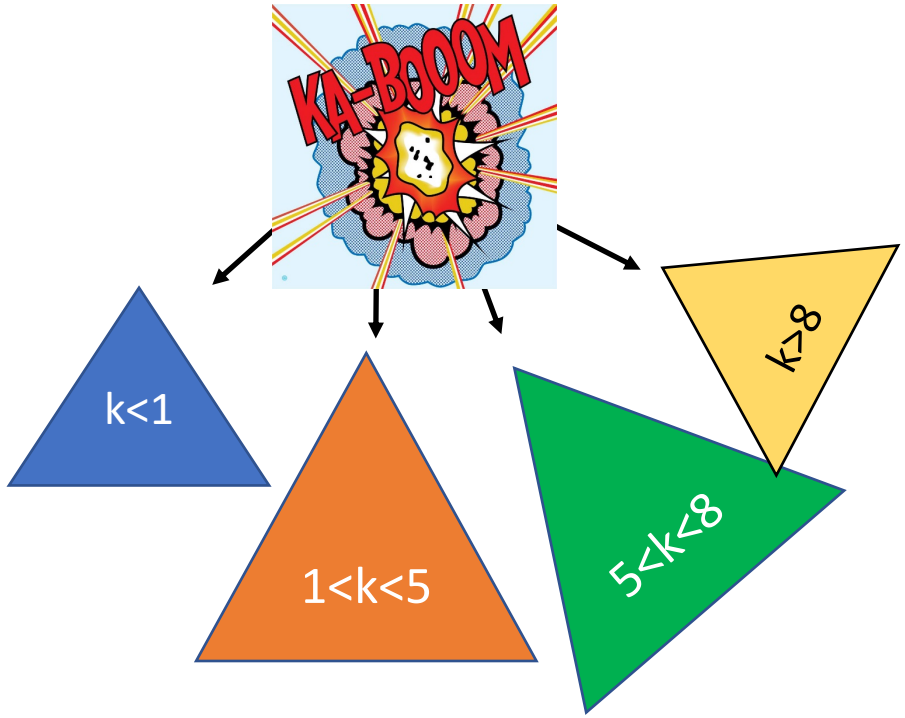
- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



Of course, this is **unacceptable**. Our solution:

Insertion : Can’t rotate key

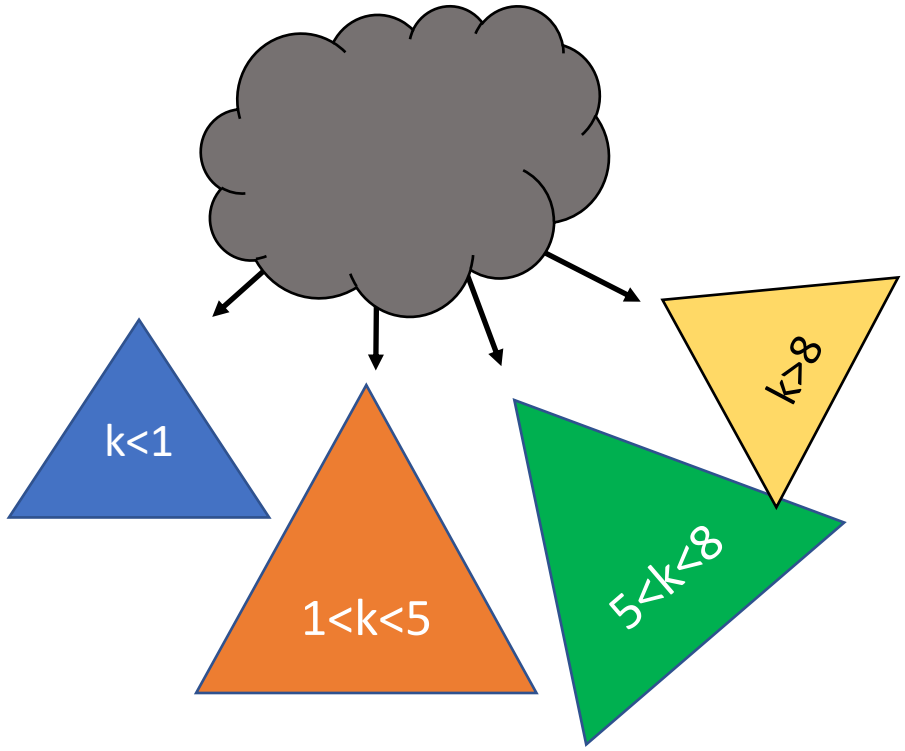
- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



Of course, this is **unacceptable**. Our solution: **Blow the node apart!**

Insertion : Can’t rotate key

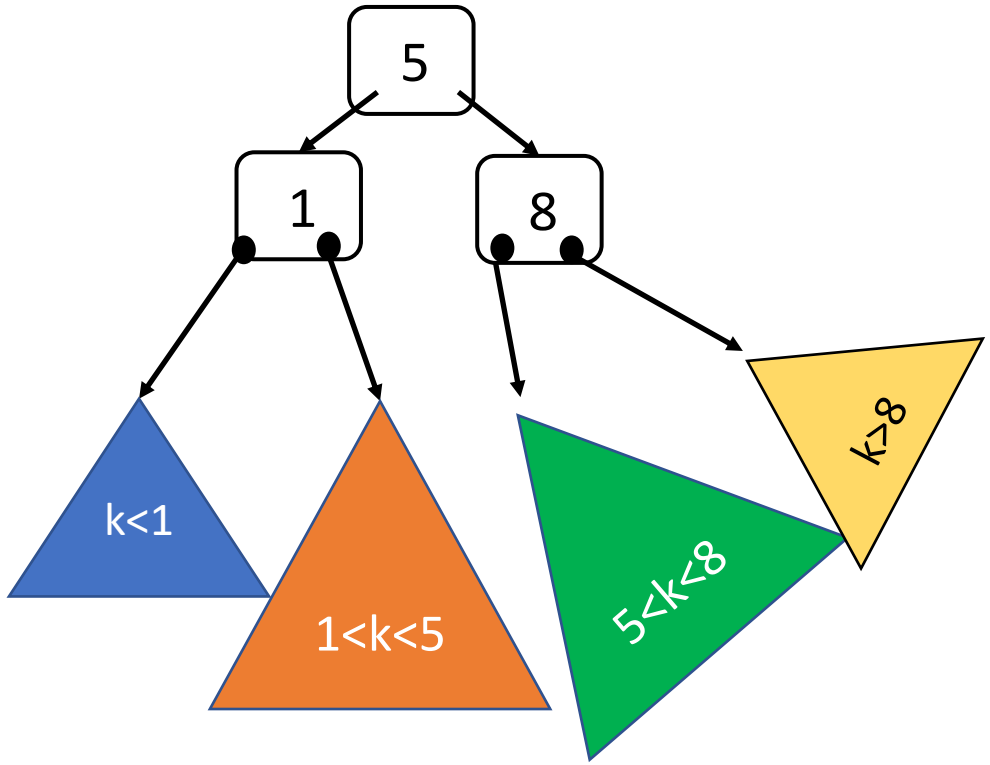
- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.



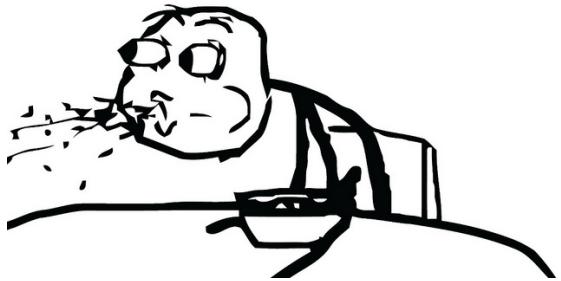
Once the dust settles….

Insertion : Can’t rotate key

- To understand what to do in this case, consider inserting 5 in this “stub” 2-3 tree.

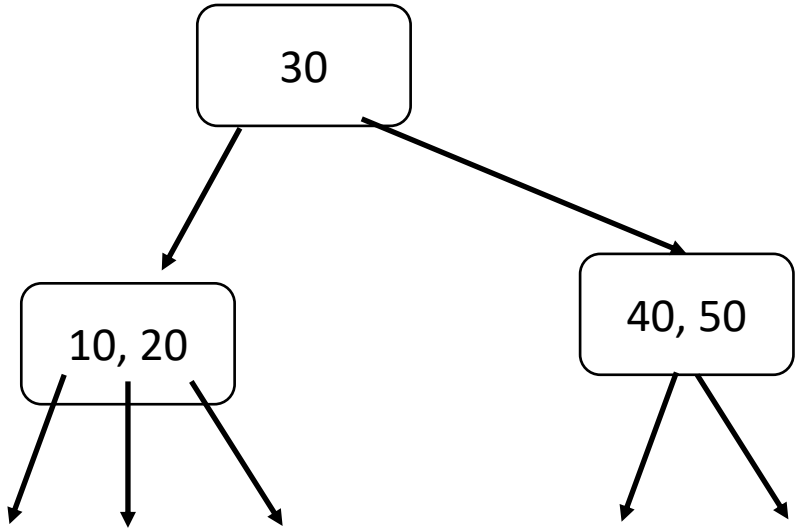


We are left with three 2-nodes that point to the correct subtrees!



Insertion: Can’t rotate key

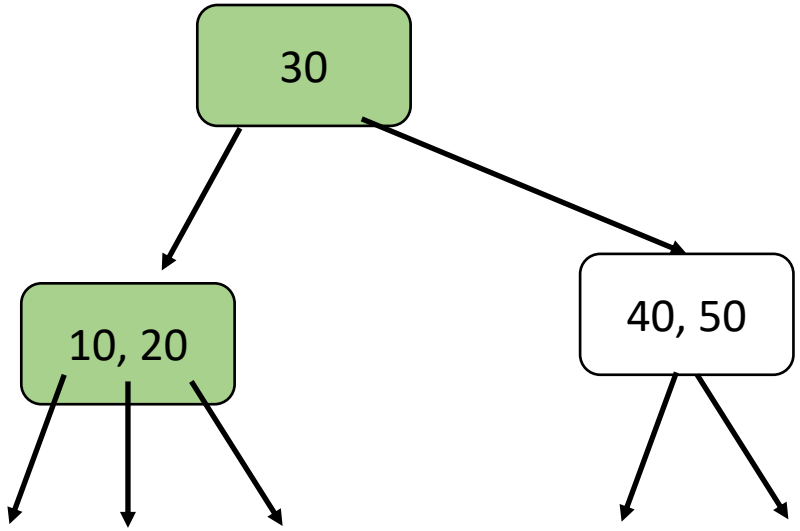
- Task: Insert 15



Insertion: Can’t rotate key

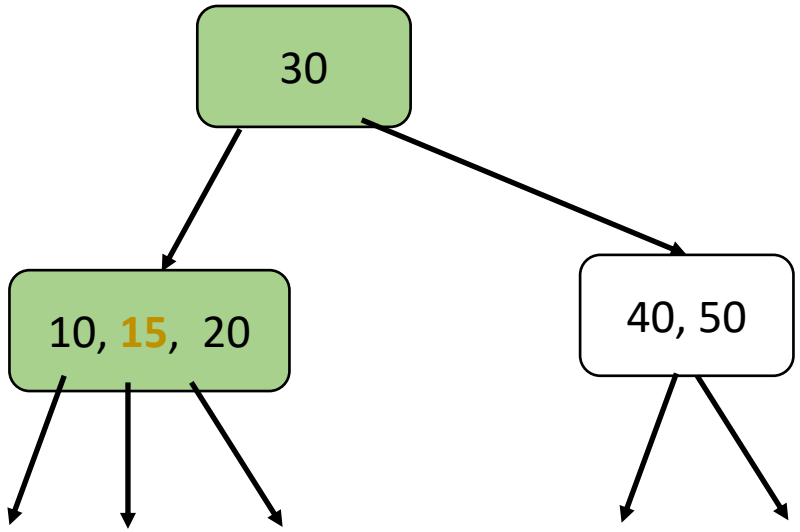
- Task: Insert 15

1. Search



Insertion: Can’t rotate key

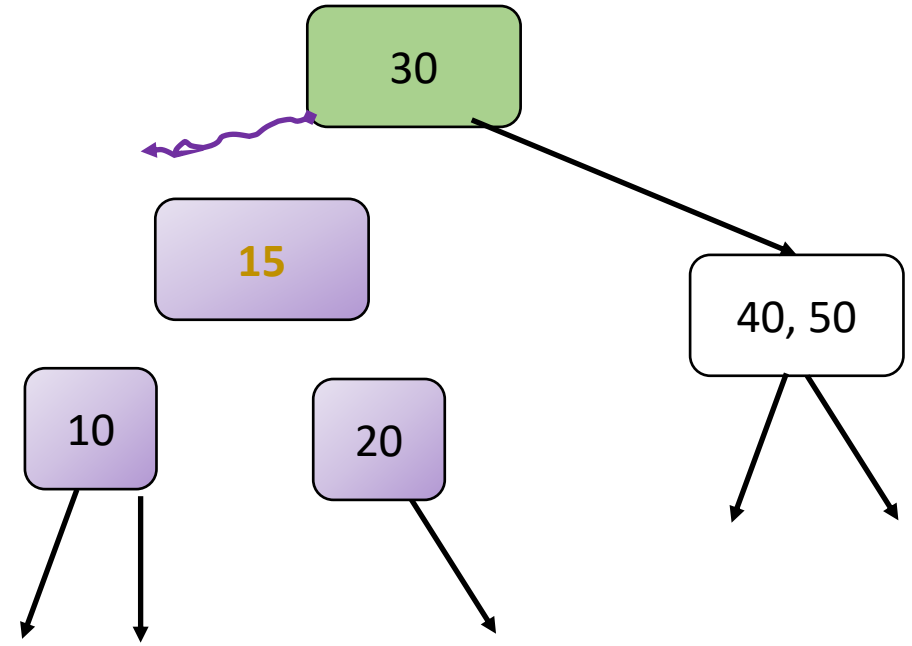
- Task: Insert 15



1. Search
2. Pretend

Insertion: Can’t rotate key

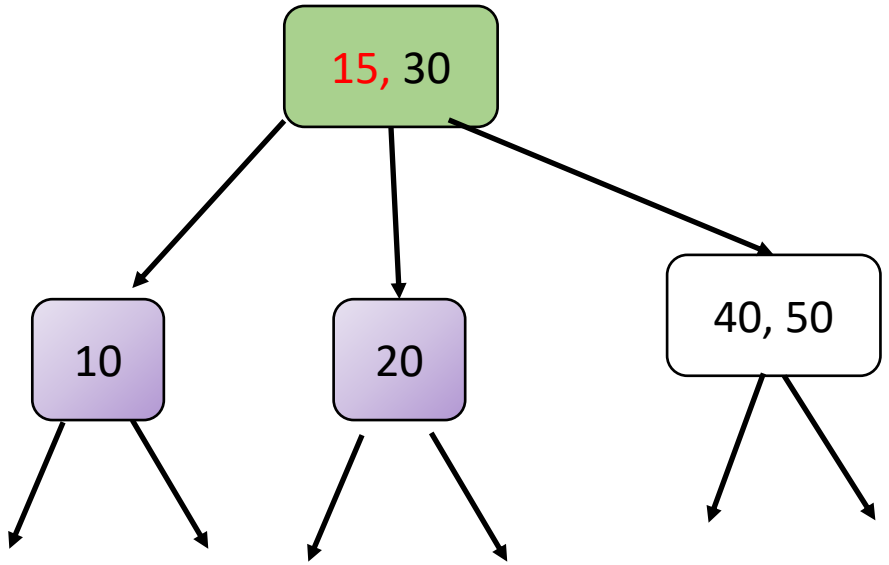
- Task: Insert 15



1. Search
2. Pretend
3. Kaboom

Insertion: Can’t rotate key

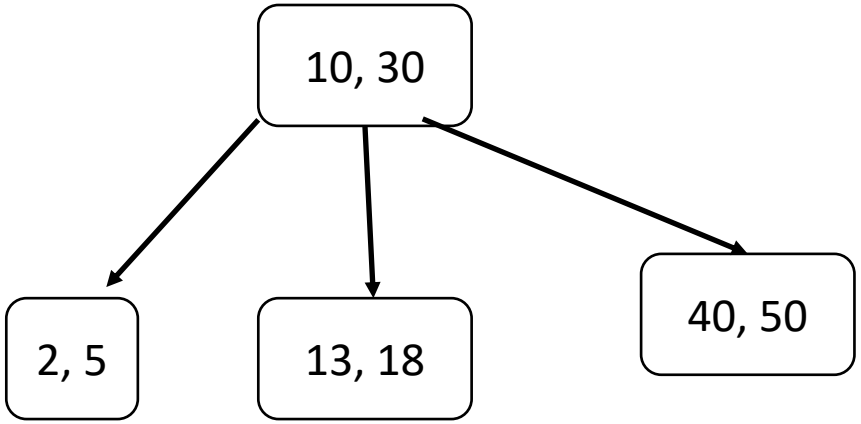
- Task: Insert 15



1. Search
2. Pretend
3. Kaboom
4. Merge with 2-node root, distributing children accordingly!

A more complex example

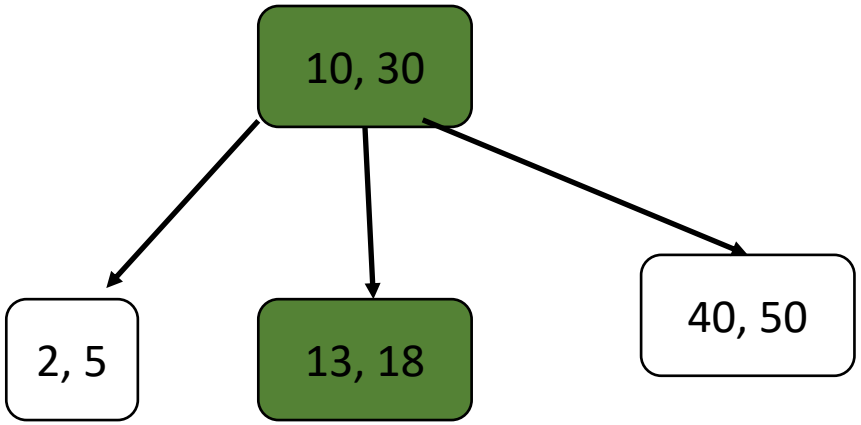
- Suppose that we have the following 2-3 tree and we want to insert **15**:



A more complex example

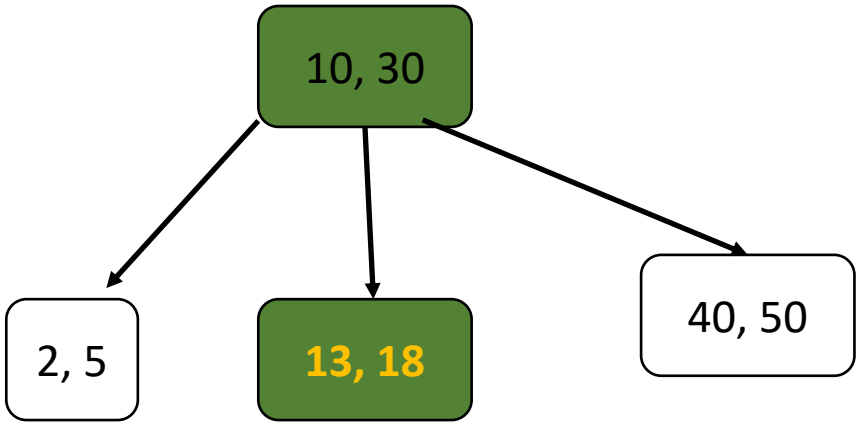
- Suppose that we have the following 2-3 tree and we want to insert **15**:

1. **Search**



A more complex example

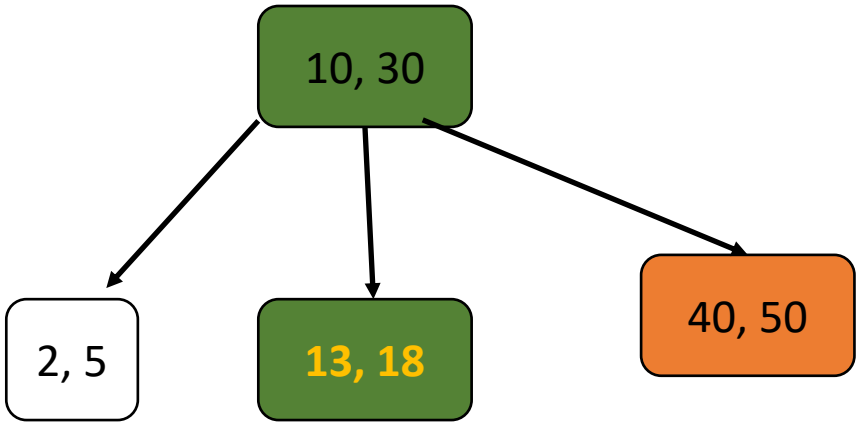
- Suppose that we have the following 2-3 tree and we want to insert **15**:



1. **Search**
2. **Investigate whether overflow occurs (it does in this case)**

A more complex example

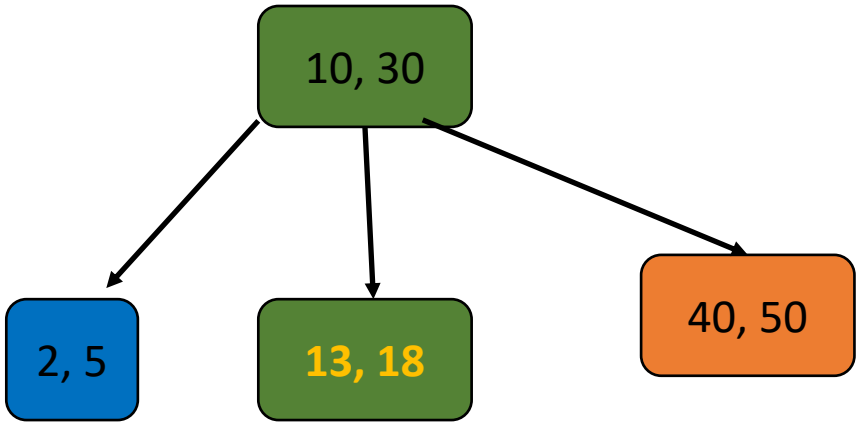
- Suppose that we have the following 2-3 tree and we want to insert **15**:



1. Search
2. Investigate whether overflow occurs (it does in this case)
3. Examine right sibling (full)

A more complex example

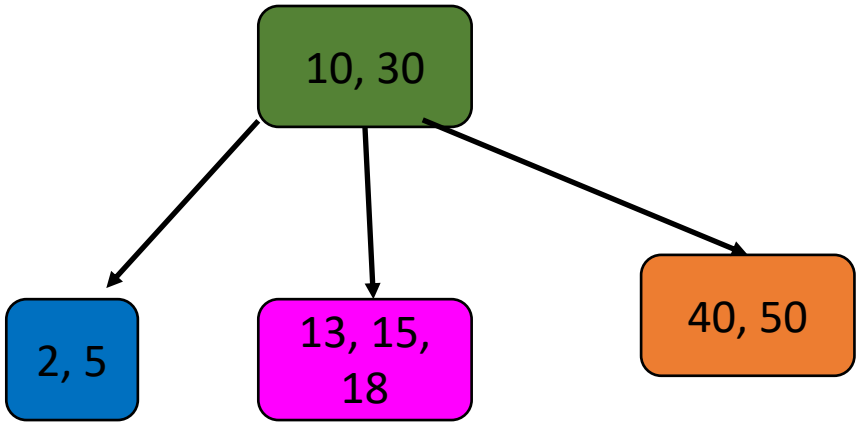
- Suppose that we have the following 2-3 tree and we want to insert **15**:



1. Search
2. Investigate whether overflow occurs (it does in this case)
3. Examine right sibling (full)
4. Examine left sibling (full)

A more complex example

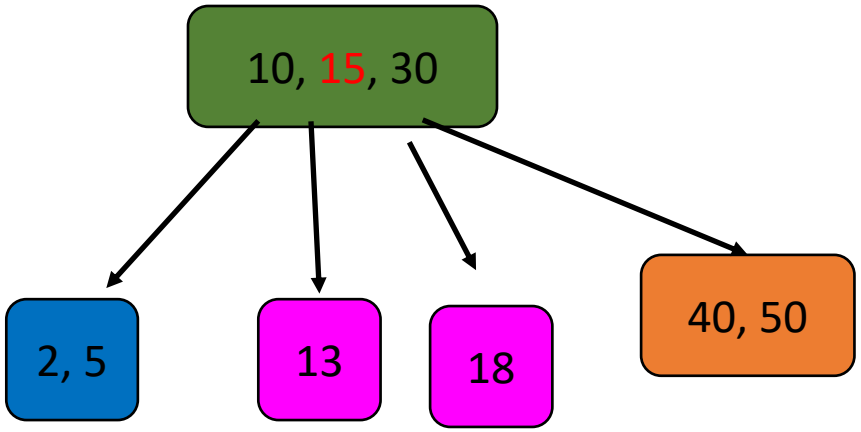
- Suppose that we have the following 2-3 tree and we want to insert **15**:



1. Search
2. Investigate whether overflow occurs (it does in this case)
3. Examine right sibling (full)
4. Examine left sibling (full)
5. Temporarily pretend a 4-node is possible

A more complex example

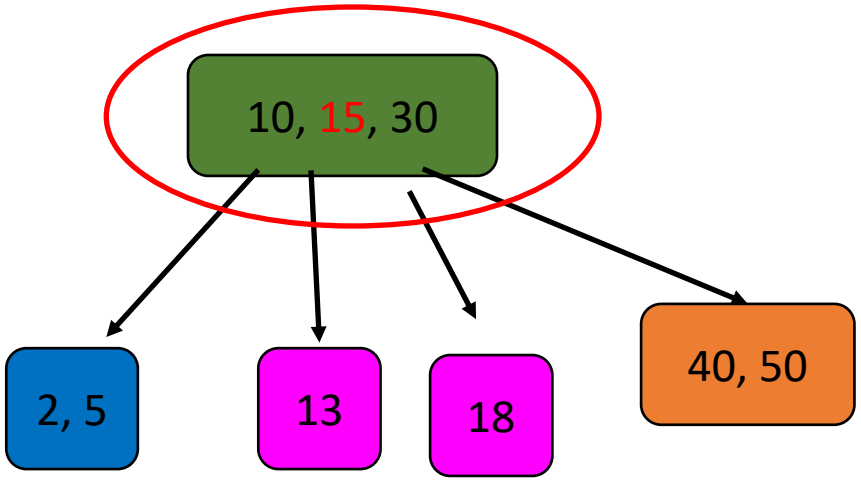
- Suppose that we have the following 2-3 tree and we want to insert **15**:



1. Search
2. Investigate whether overflow occurs (it does in this case)
3. Examine right sibling (full)
4. Examine left sibling (full)
5. Temporarily pretend a 4-node is possible
6. Split it, elevating middle key (15) to parent.

A more complex example

- Suppose that we have the following 2-3 tree and we want to insert **15**:

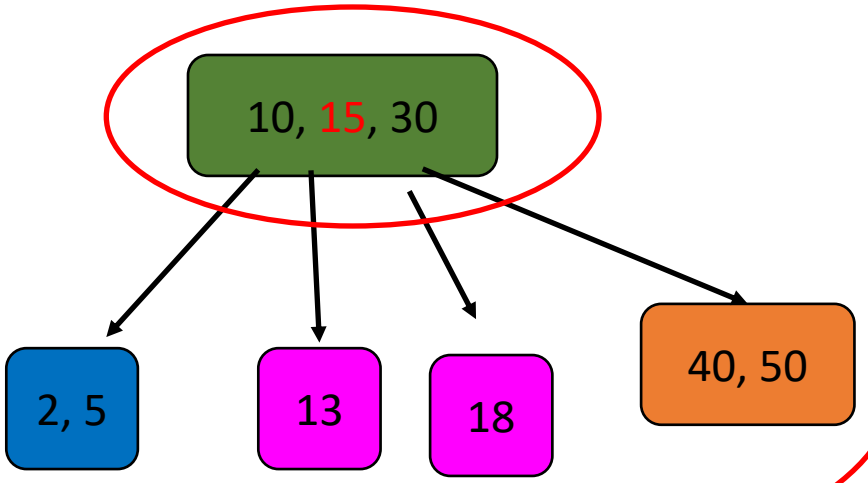


But now we have an overflow at the parent ☹

1. **Search**
2. **Investigate whether overflow occurs (it does in this case)**
3. **Examine right sibling (full)**
4. **Examine left sibling (full)**
5. **Temporarily pretend a 4-node is possible**
6. **Split it, elevating middle key (15) to parent.**

A more complex example

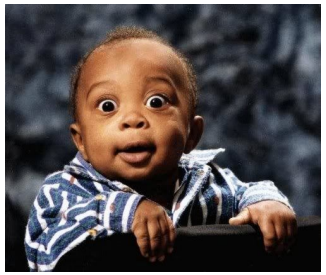
- Suppose that we have the following 2-3 tree and we want to insert **15**:



But now we have an overflow at the parent ☹

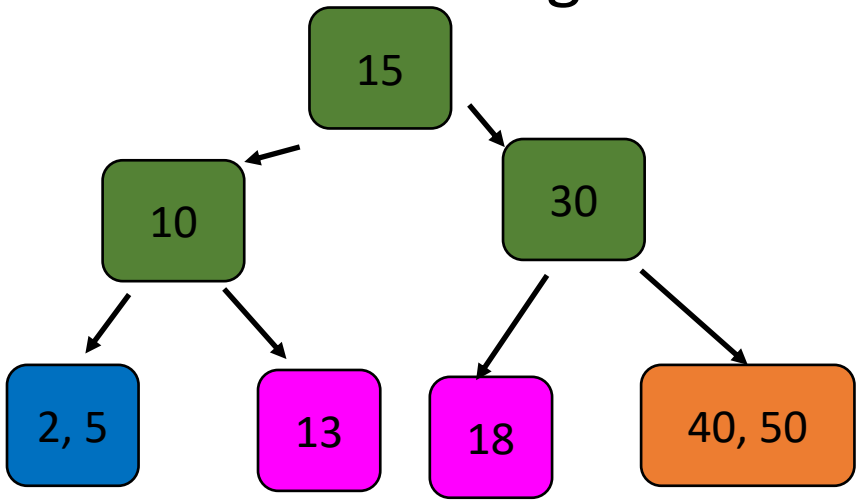
1. **Search**
2. **Investigate whether overflow occurs (it does in this case)**
3. **Examine right sibling (full)**
4. **Examine left sibling (full)**
5. **Temporarily pretend a 4-node is possible**
6. **Split it, elevating middle key (15) to parent.**

The solution: **Recursively apply the checks 2, 3 and 4 for the parent node, to somehow deal with the overflow!**



A more complex example

- Suppose that we have the following 2-3 tree and we want to insert **15**:



But now we have an overflow at the parent ☹

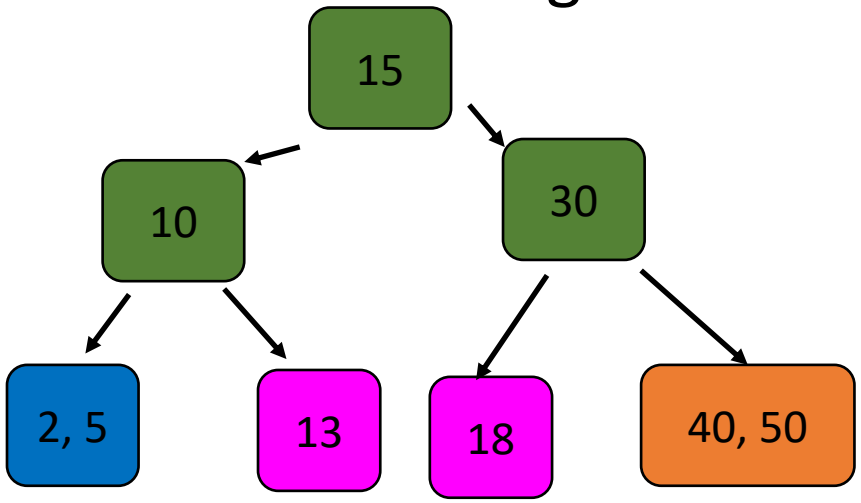
1. **Search**
2. **Investigate whether overflow occurs (it does in this case)**
3. **Examine right sibling (full)**
4. **Examine left sibling (full)**
5. **Temporarily pretend a 4-node is possible**
6. **Split it, elevating middle key (15) to parent.**

The solution: **Recursively apply the checks 2, 3 and 4 for the parent node, to somehow deal with the overflow!**

In this case, a **node split is the only possibility**, since we have no siblings at the root!

A more complex example

- Suppose that we have the following 2-3 tree and we want to insert **15**:



But now we have an overflow at the parent ☹

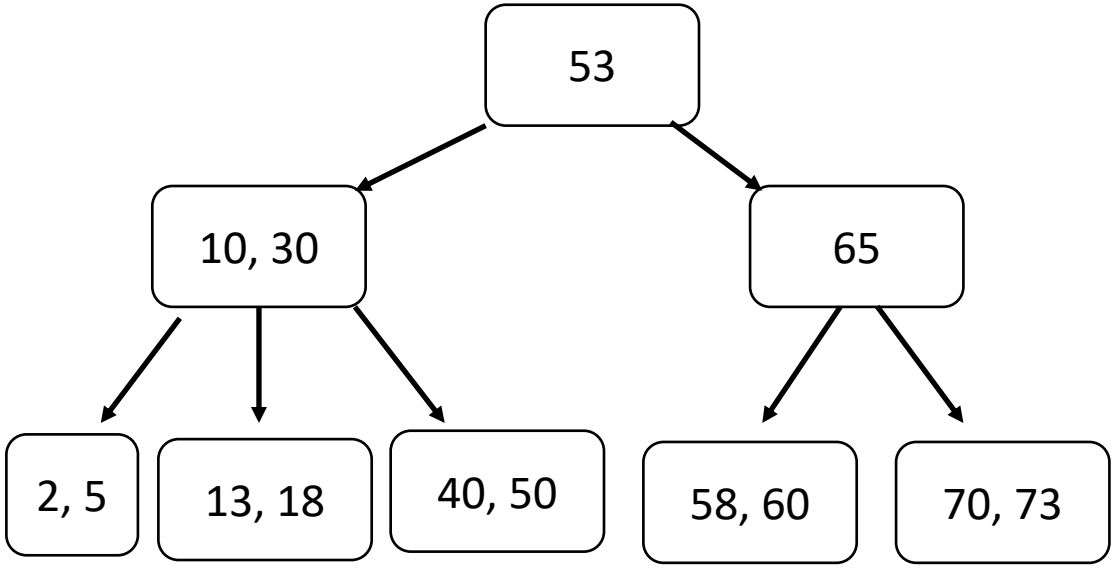
1. **Search**
2. **Investigate whether overflow occurs (it does in this case)**
3. **Examine right sibling (full)**
4. **Examine left sibling (full)**
5. **Temporarily pretend a 4-node is possible**
6. **Split it, elevating middle key (15) to parent.**

The solution: **Recursively apply the checks 2, 3 and 4 for the parent node, to somehow deal with the overflow!**

In this case, a **node split is the only possibility**, since we have no siblings at the root!

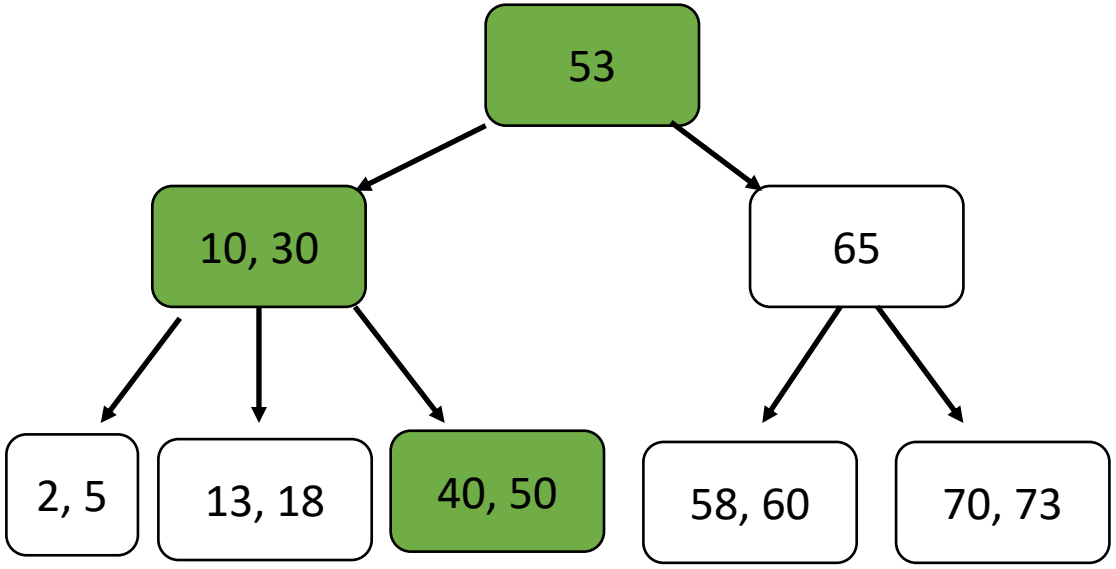
And this is an example of how 2-3 trees increase their height ☺

Mixing up splittings and rotations!



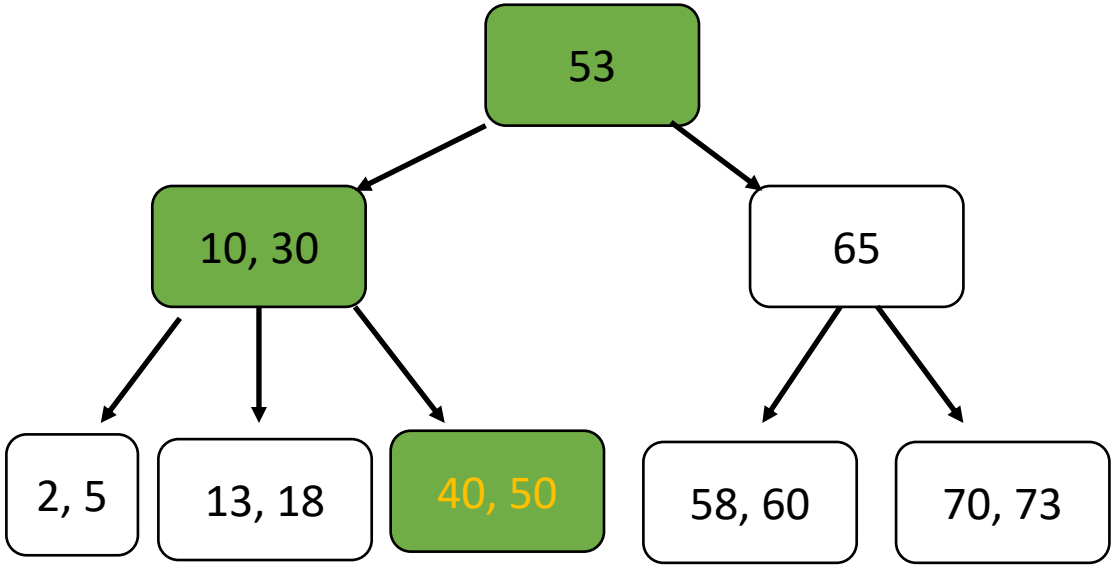
Task: Insert **32**

Mixing up splittings and rotations!



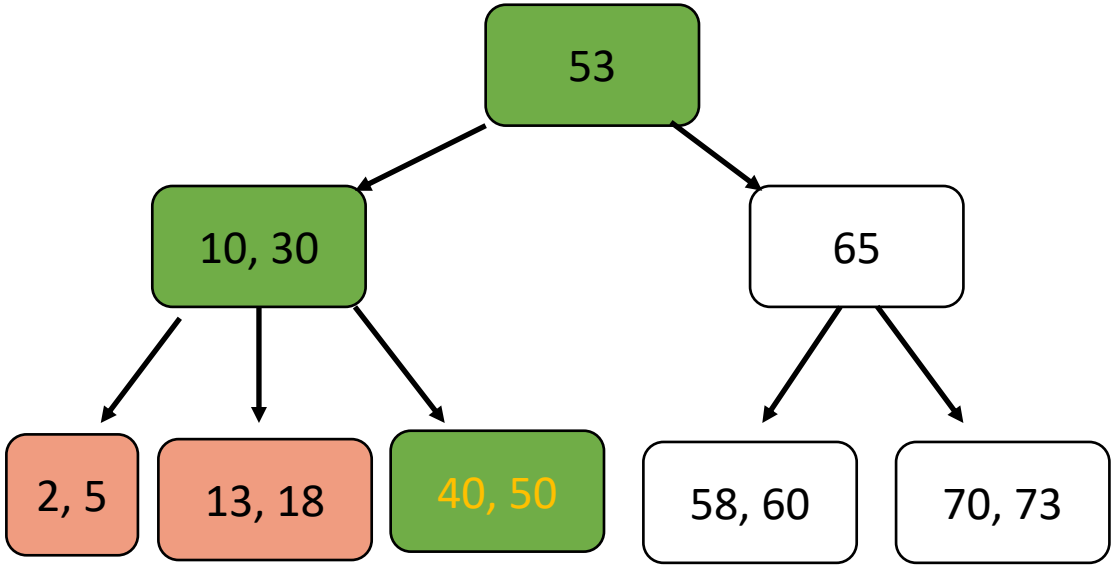
Task: Insert **32**

Mixing up splittings and rotations!



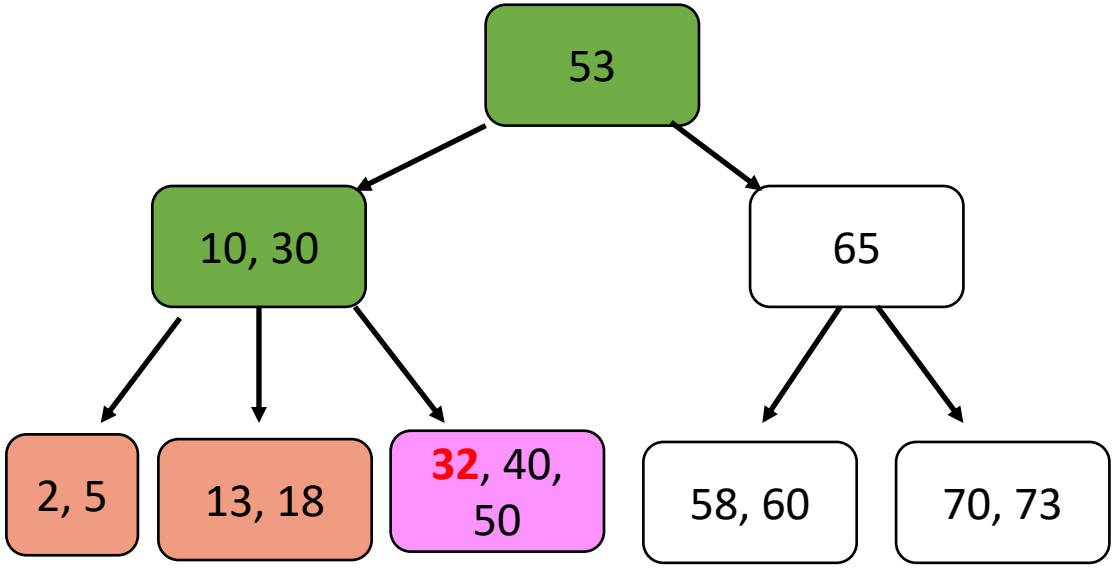
Task: Insert **32**

Mixing up splittings and rotations!



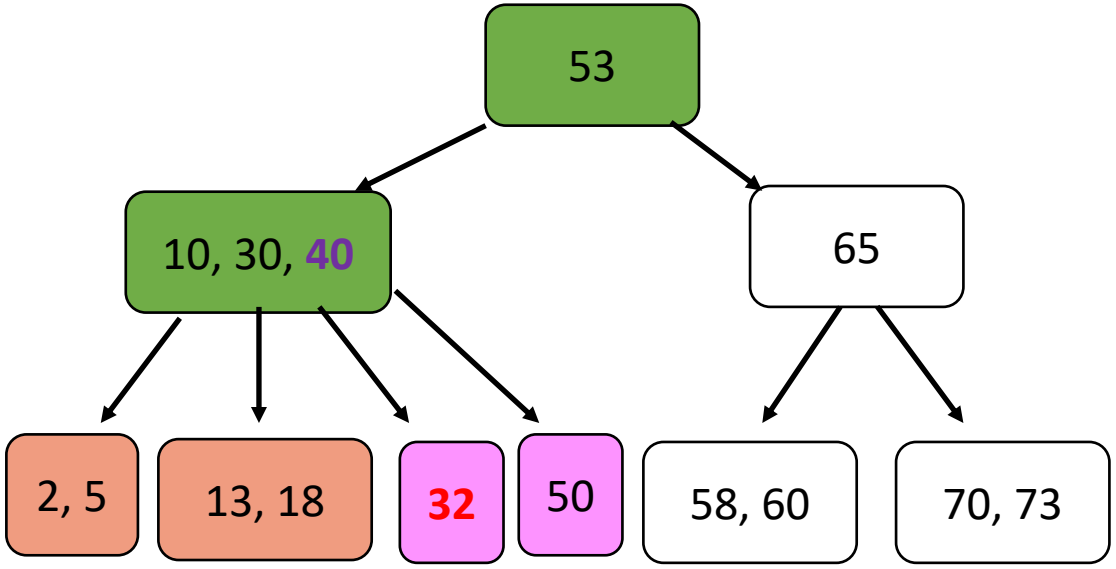
Task: Insert **32**

Mixing up splittings and rotations!



Task: Insert **32**

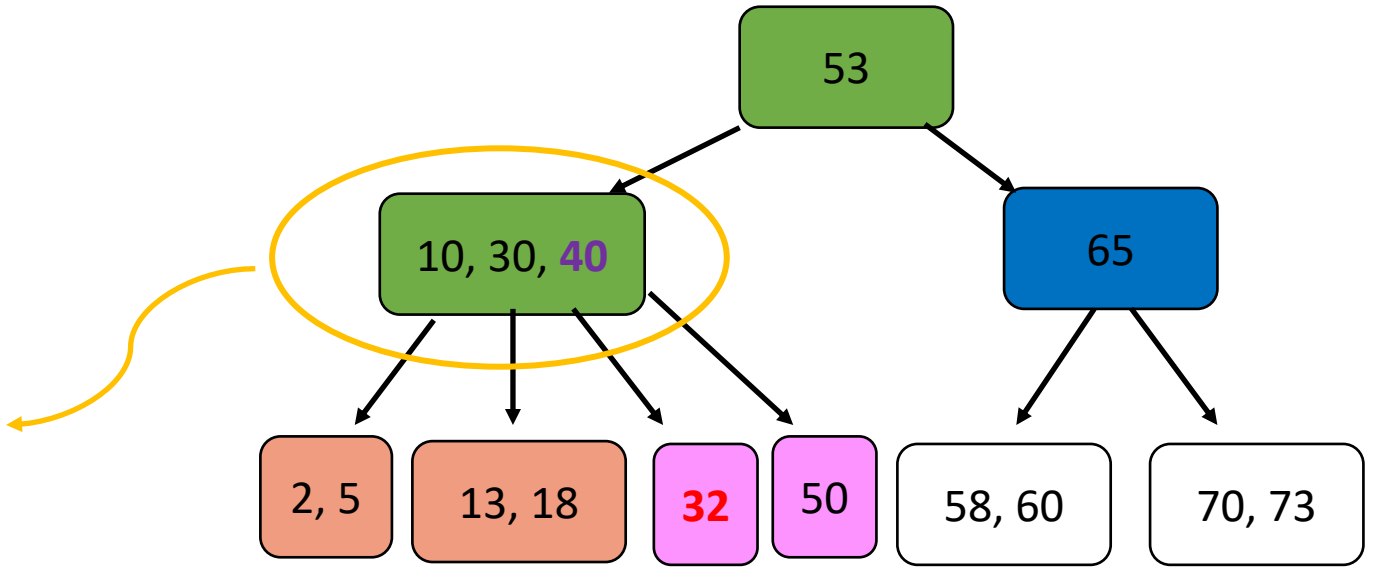
Mixing up splittings and rotations!



Task: Insert **32**

Note: It is **always** the **middle key** that should be elevated (**40** in this case). Otherwise, the BST property is not maintained!

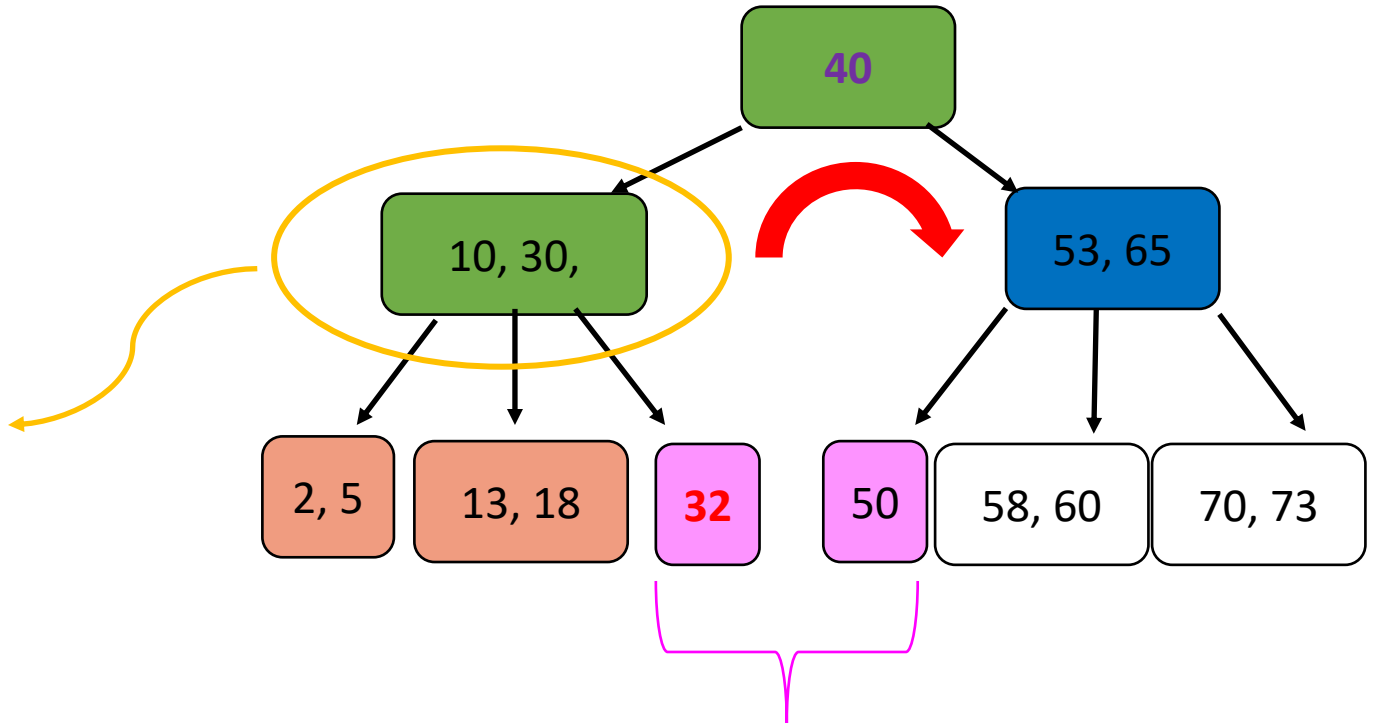
Mixing up splittings and rotations!



Task: Insert **32**

Note that this node, however, has a sibling with space to rotate a key to!

Mixing up splittings and rotations!



Task: Insert **32**

Note that this node, however, has a sibling with space to rotate a key to!

Don't forget to re-distribute subtrees! When rotating keys **above** the leaf level, there are **non-null** subtrees that need to be re-distributed!

Take-home message #1

- 2-3 trees are built from the bottom-up, not the top-down
  - Remember structural induction on trees in 250?
- In the worst case, node splittings will occur all the way to the root, and those are expensive.
  - They also claim memory from the heap.
  - But with $\log_3 n$ height, maybe this cost can be acceptable.

Deletion

- Insertions in 2-3 trees might transform a 2-node into a 3-node, or blow up a temporary 4-node into 3 2-nodes.
- Deletions, on the other hand, have the potential of not just making 3-nodes into 2-nodes, but making 2-nodes into …

Deletion

- Insertions in 2-3 trees might transform a 2-node into a 3-node, or blow up a temporary 4-node into 3 2-nodes.
- Deletions, on the other hand, have the potential of not just making 3-nodes into 2-nodes, but making 2-nodes into … ?

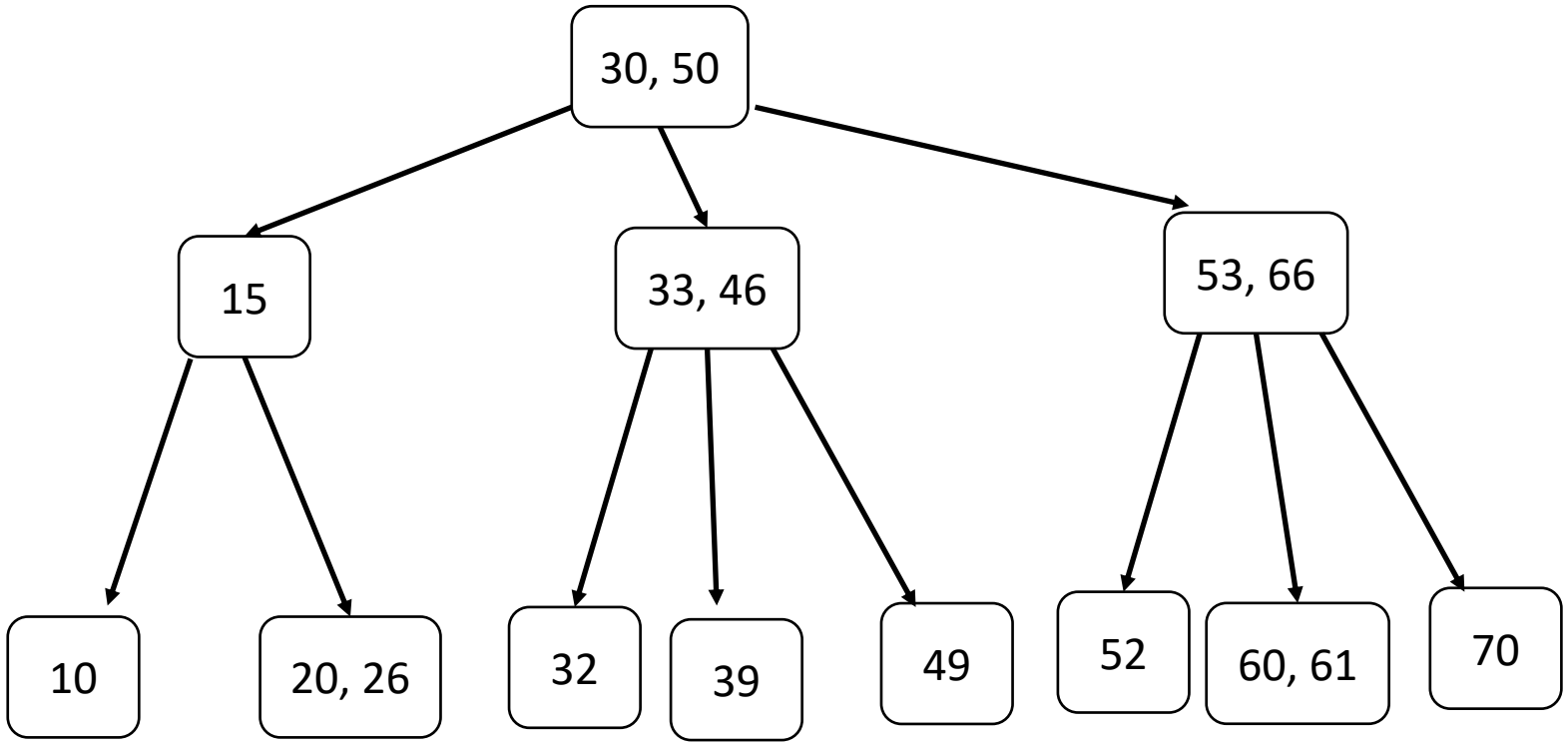
Deletion

- Insertions in 2-3 trees might transform a 2-node into a 3-node, or blow up a temporary 4-node into 3 2-nodes.
- Deletions, on the other hand, have the potential of not just making 3-nodes into 2-nodes, but making 2-nodes into … ?
- We'll see what the result of that will be soon.

Deletion

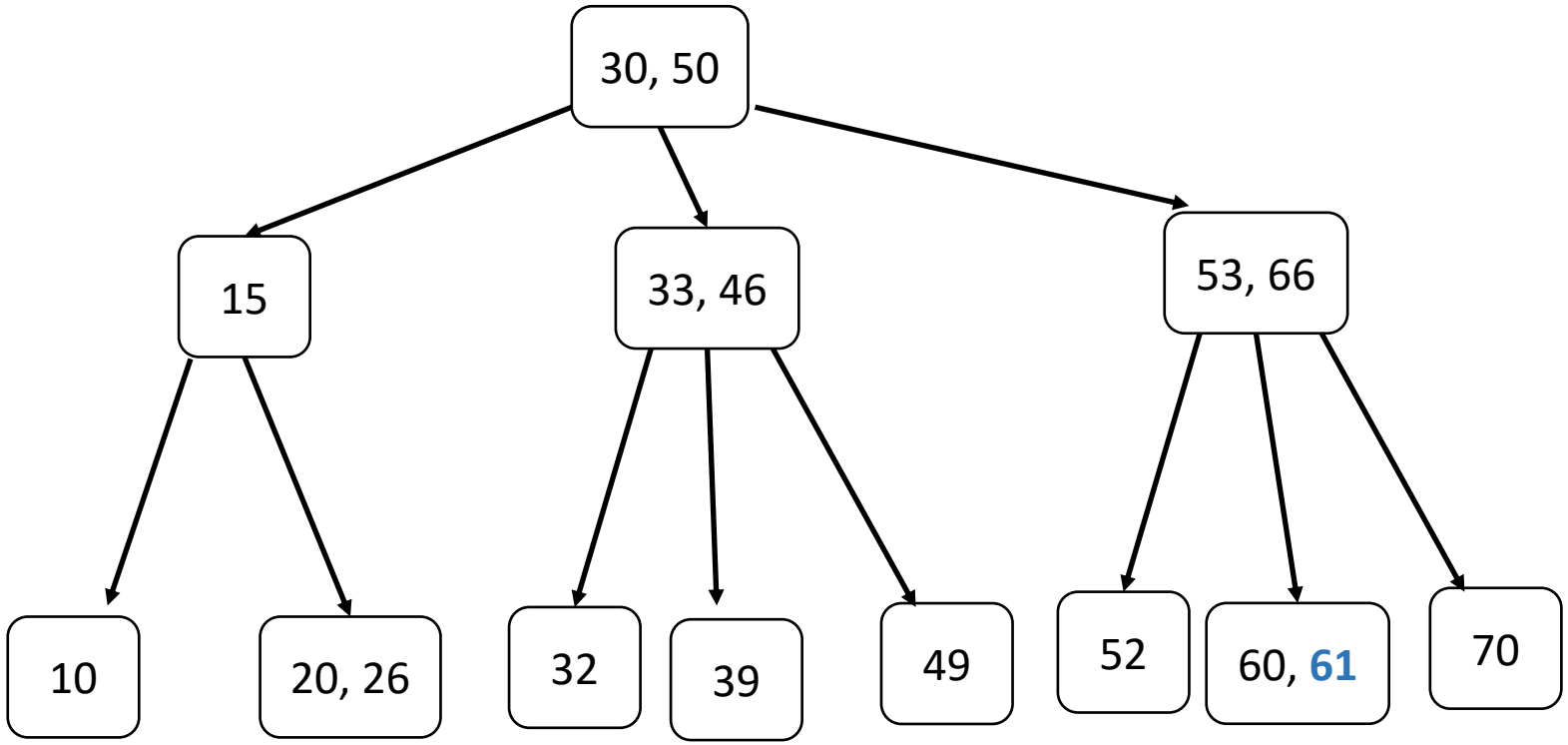
- Insertions in 2-3 trees might transform a 2-node into a 3-node, or blow up a temporary 4-node into 3 2-nodes.
- Deletions, on the other hand, have the potential of not just making 3-nodes into 2-nodes, but making 2-nodes into … ?
- We'll see what the result of that will be soon.
- Key point: Since deleting an inner node always ends up being a deletion at the leaf level, we only care about leaf deletions.

Deletion



Deletion

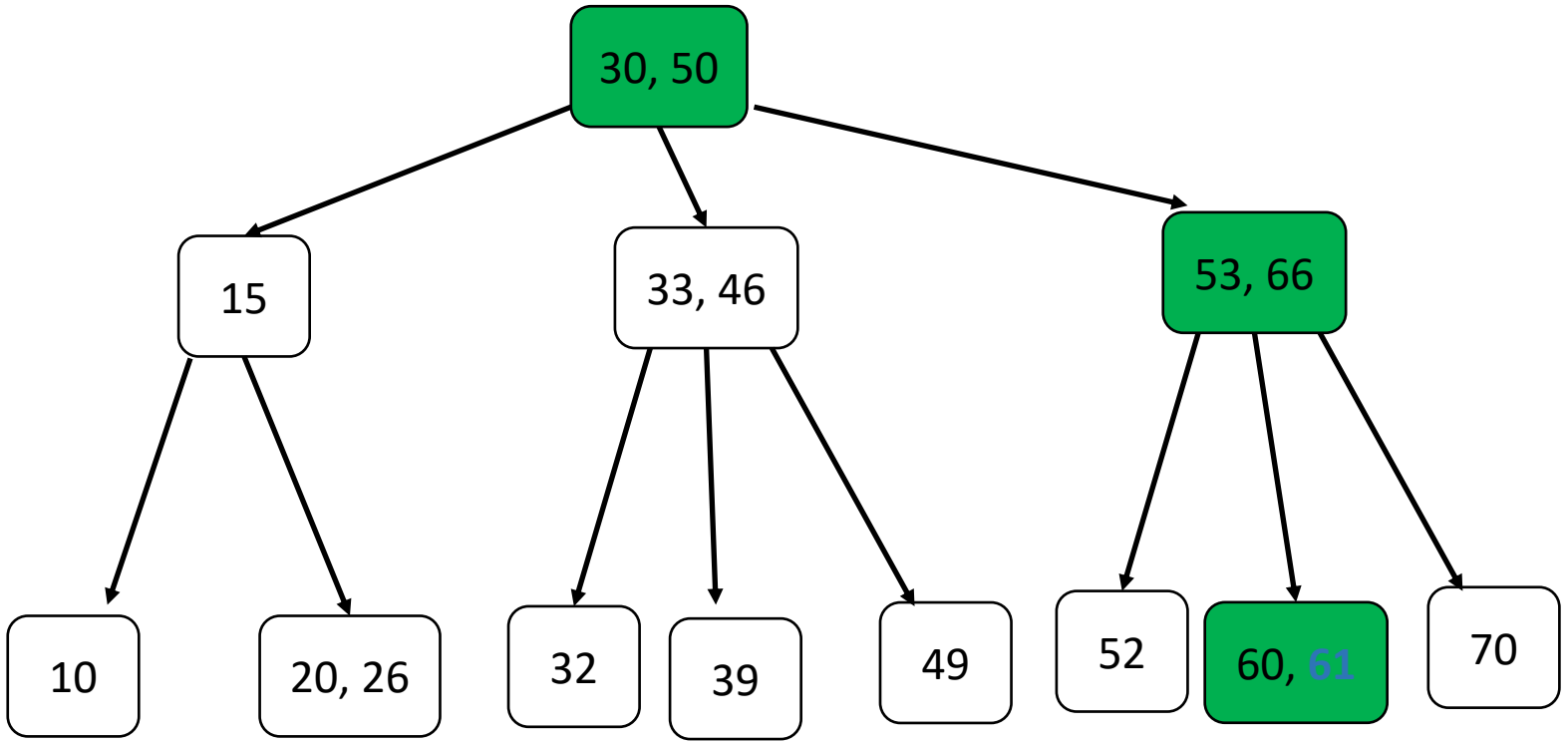
Delete 61



Deletion

Delete 61

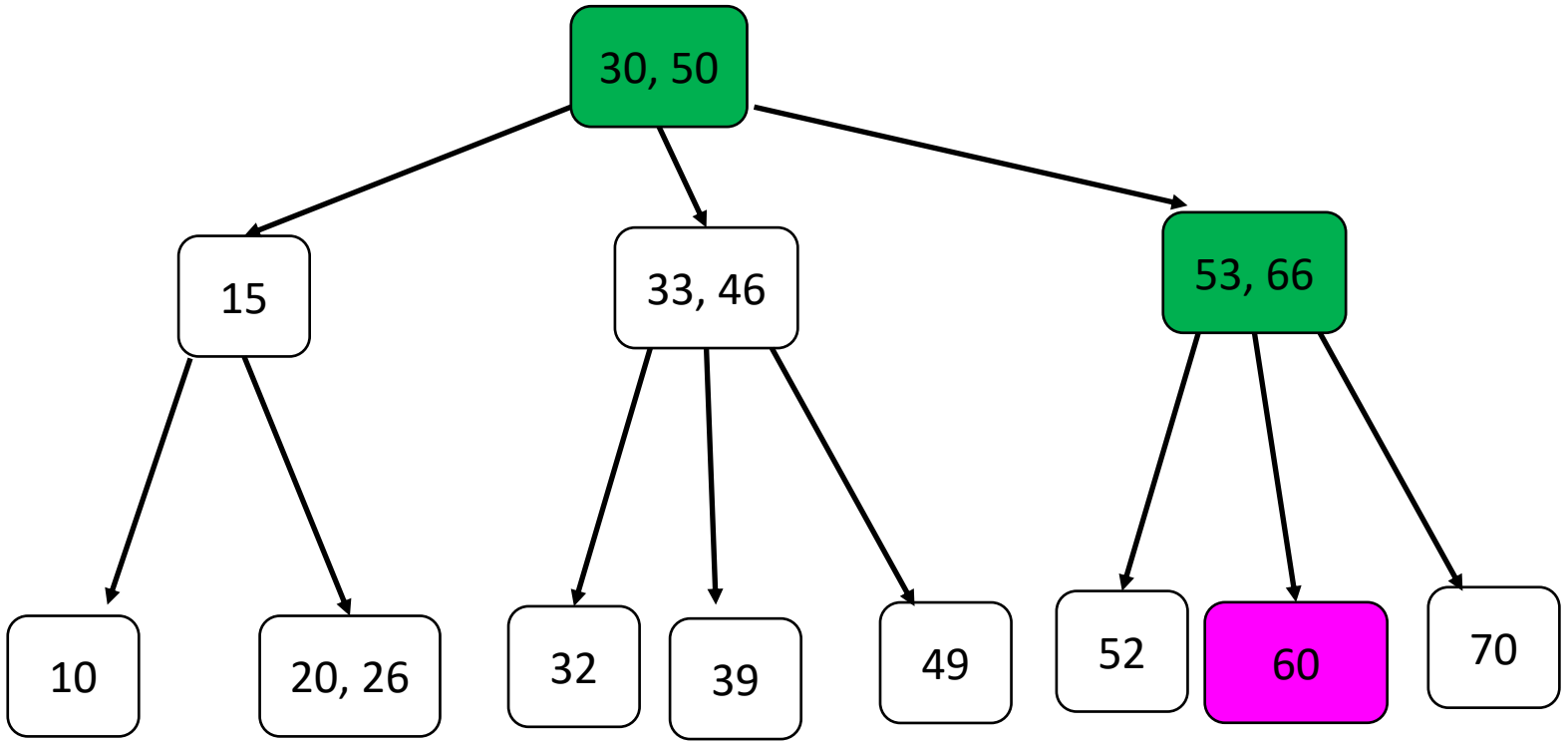
1. Search



Deletion

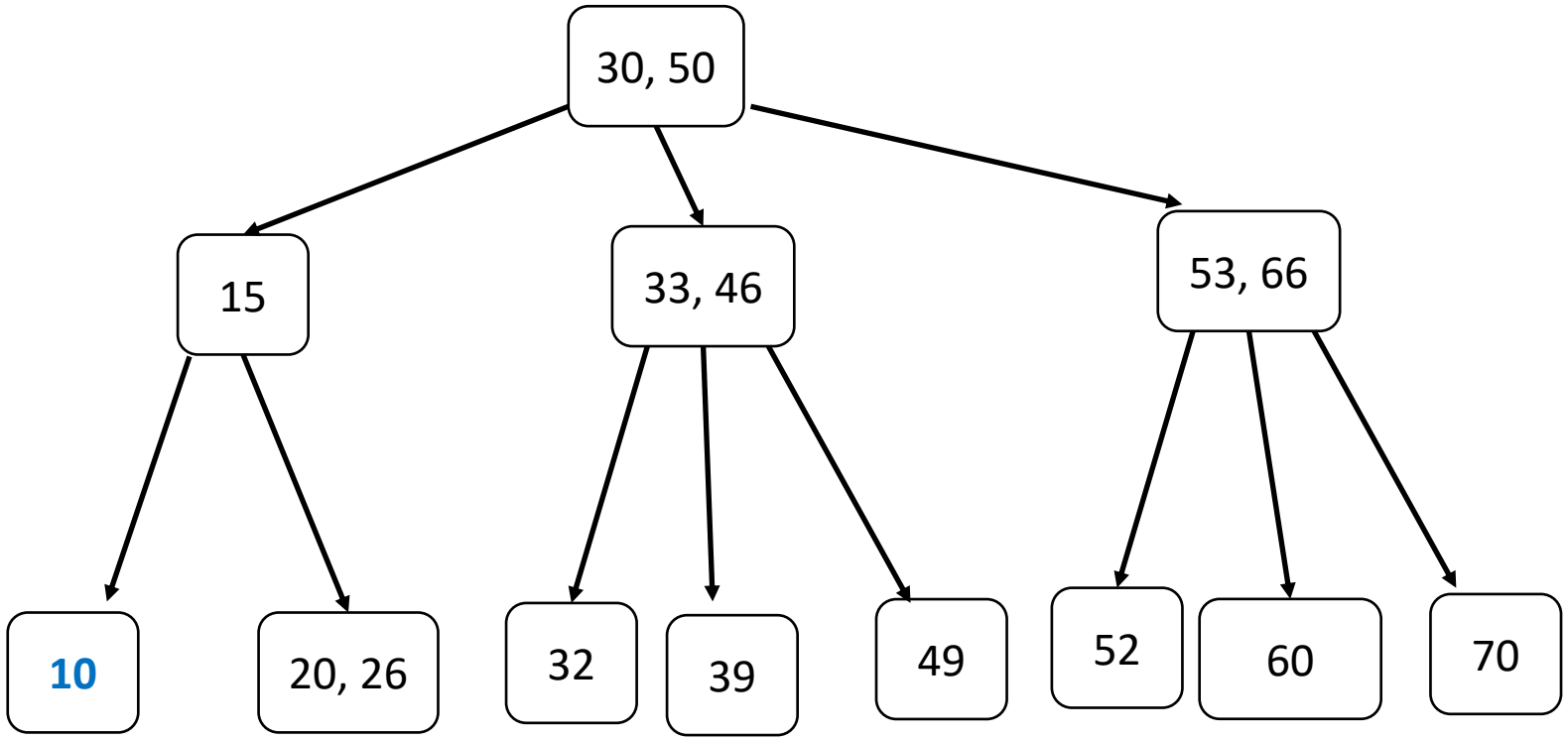
Delete 61

1. **Search**
2. **Since node that contains 61 is a 3-node, we just remove 61, make it a 2-node and we're done! ☺**



Deletion

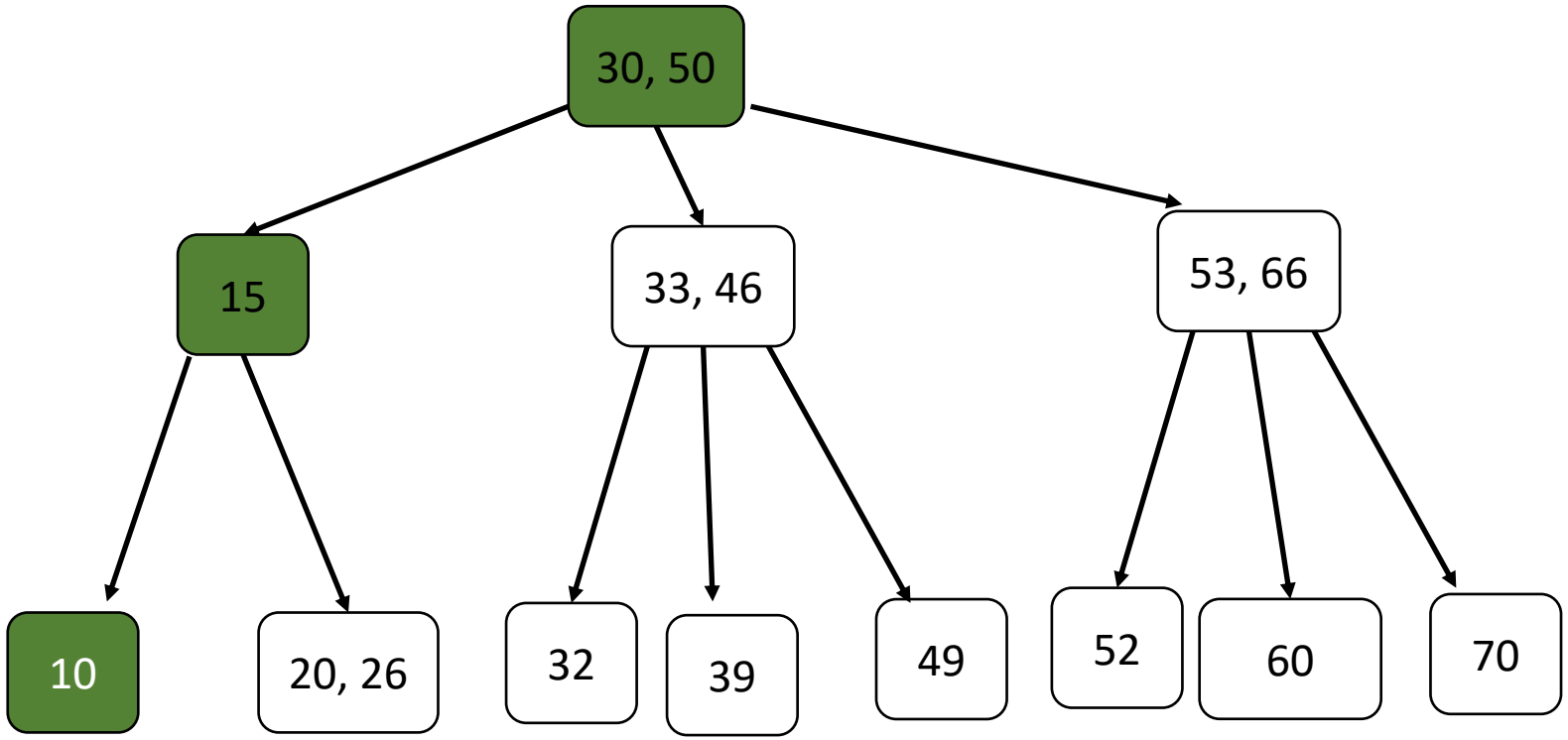
Delete 10



Deletion

Delete 10

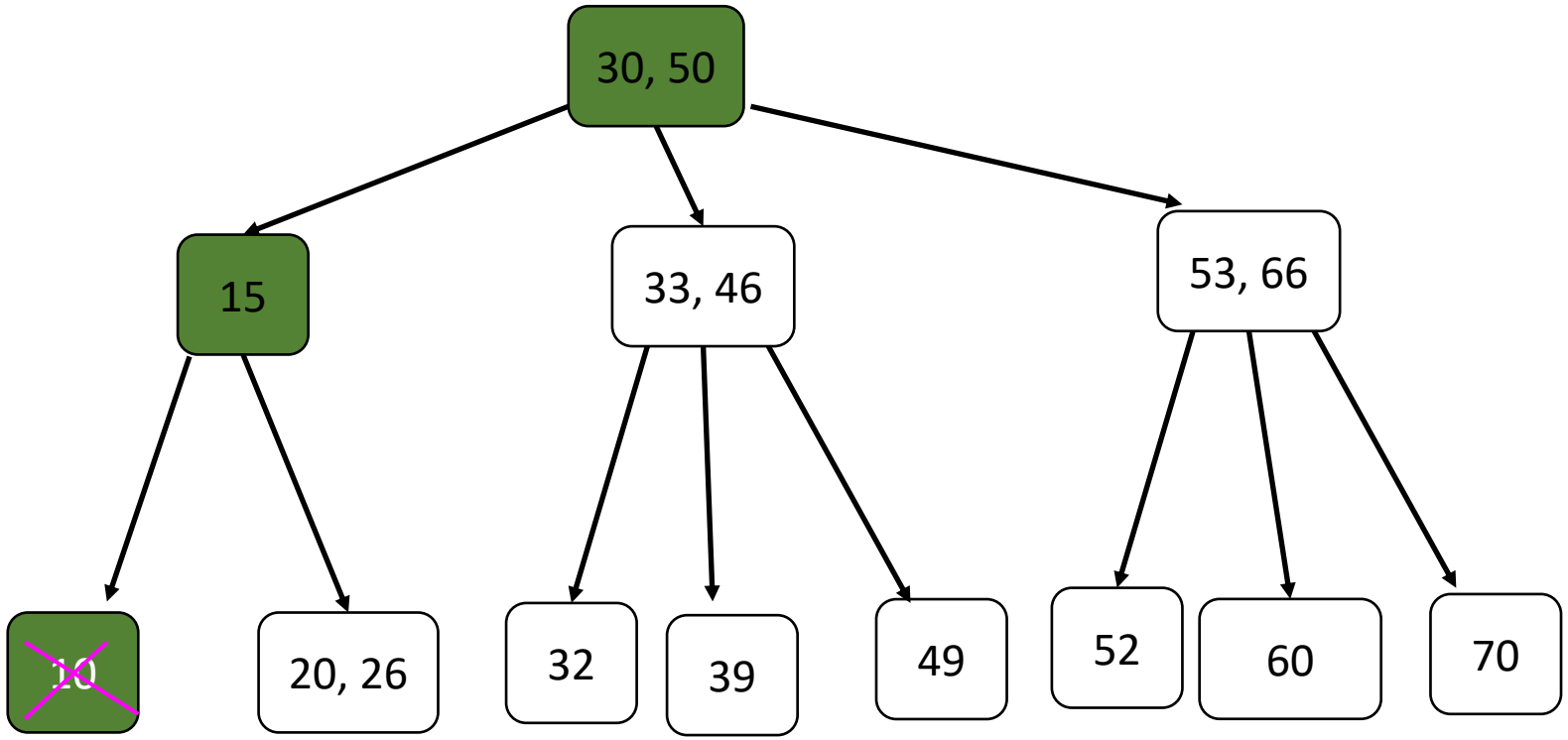
1. Search



Deletion

Delete 10

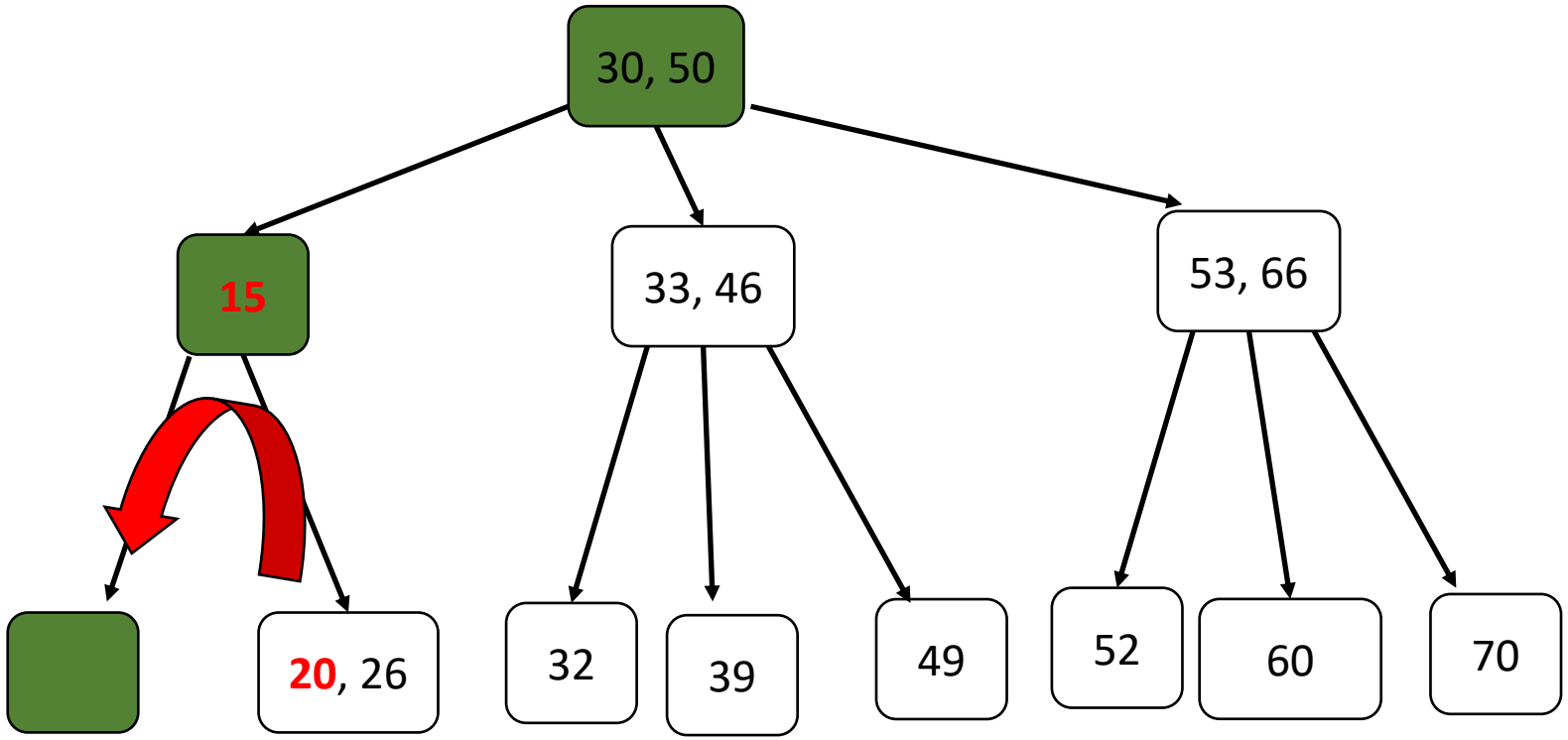
1. **Search**
2. Deleting 10 leads to a node underflow… ☹



Deletion

Delete 10

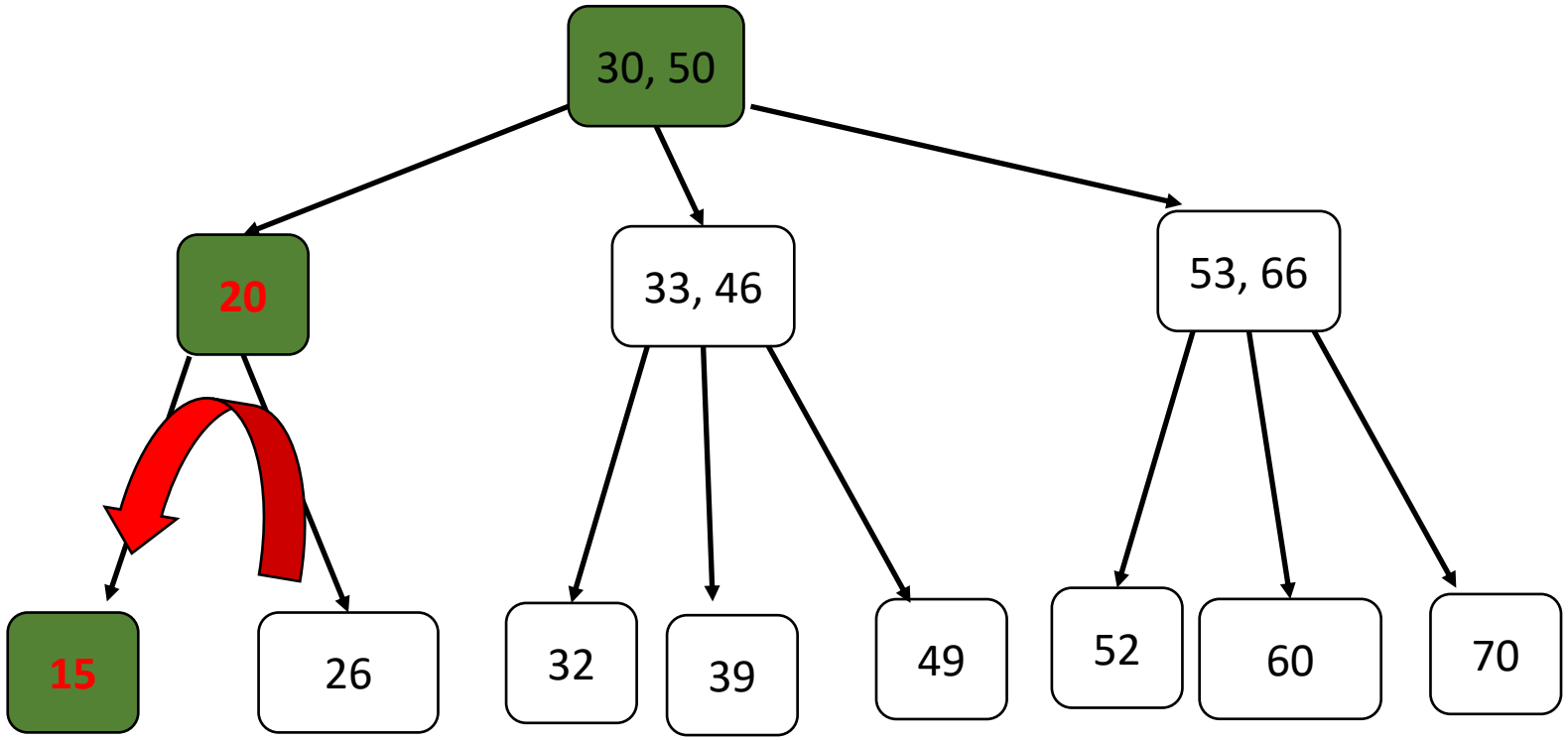
1. **Search**
2. Deleting 10 leads to a node underflow… ☹
3. The solution: rotate a key from our sibling! ☺



Deletion

Delete 10

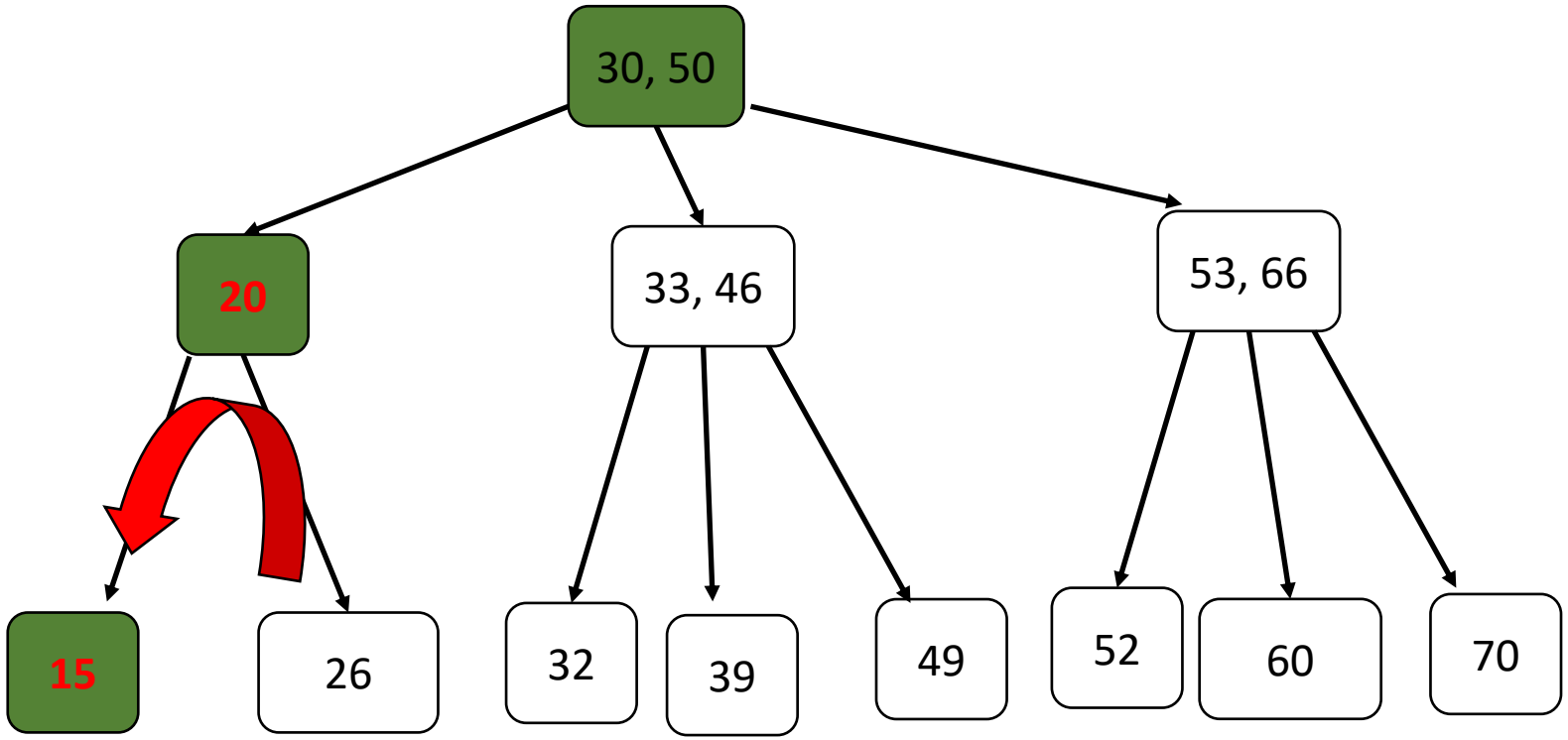
1. **Search**
2. Deleting 10 leads to a node underflow… ☹
3. The solution: rotate a key from our sibling! ☺



Deletion

Delete 10

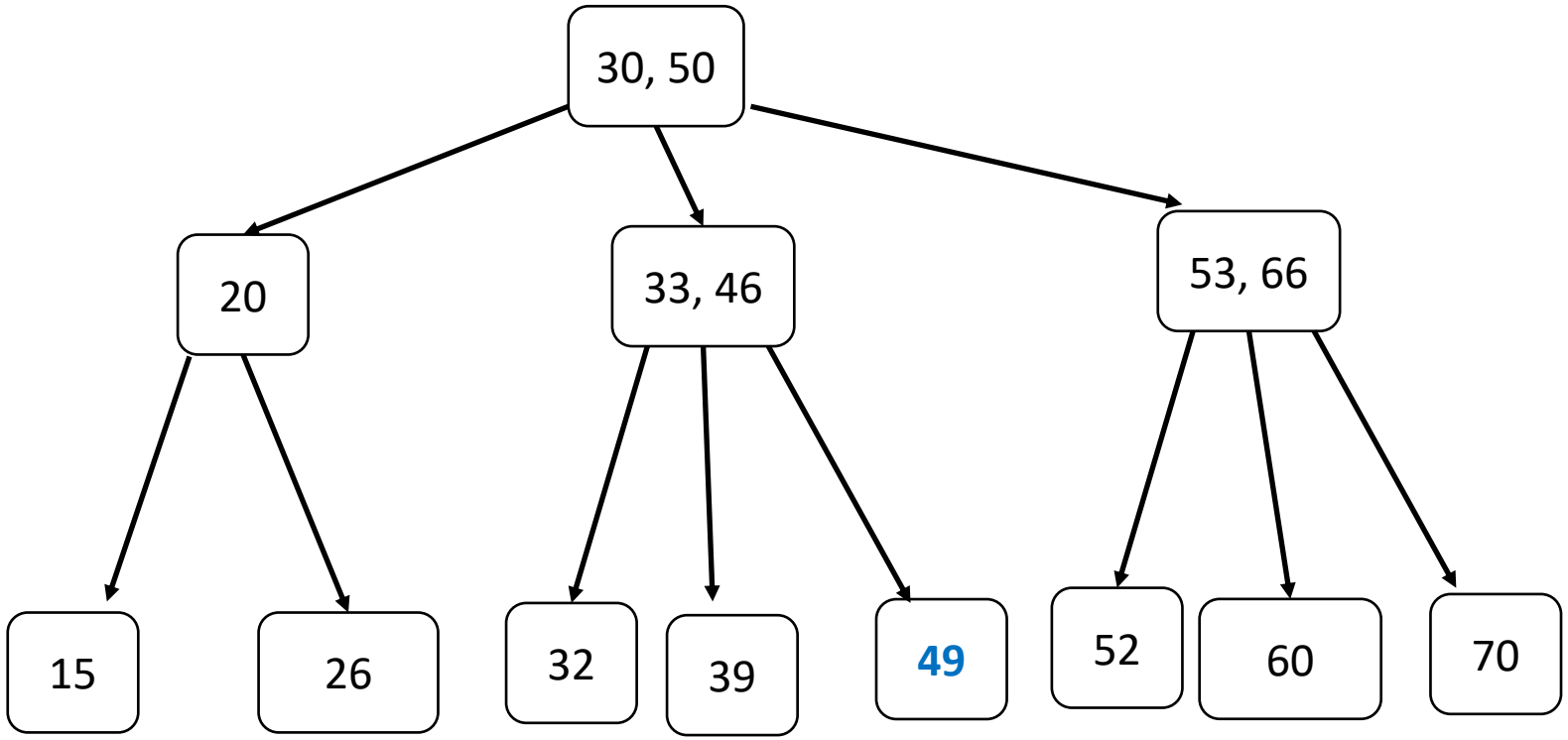
1. **Search**
2. Deleting 10 leads to a node underflow… ☹
3. The solution: rotate a key from our sibling! ☺



These operations are called **“key rotations”**, to disambiguate them from the **node rotations** that occur in AVL and Splay trees.

Deletion

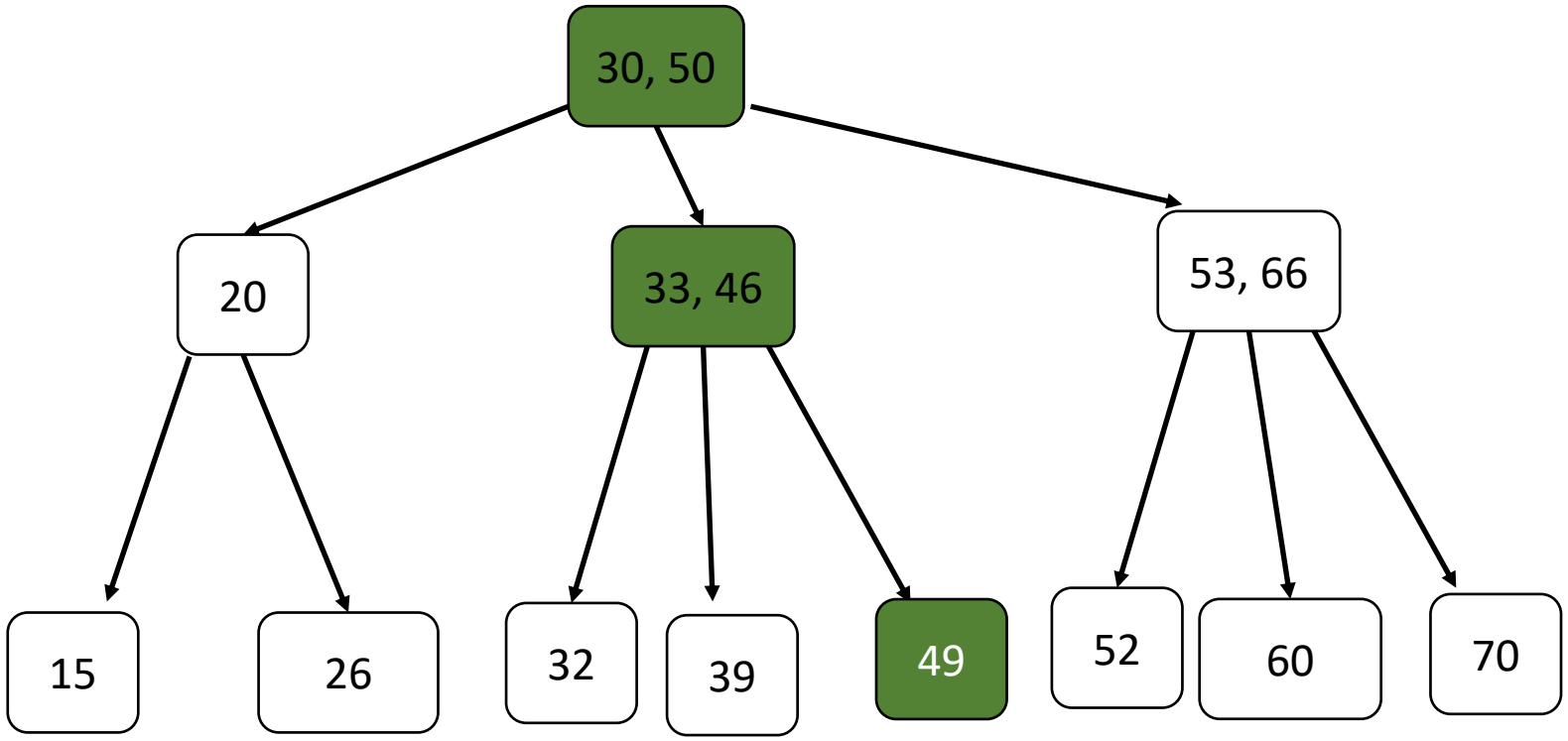
Delete 49



Deletion

Delete 49

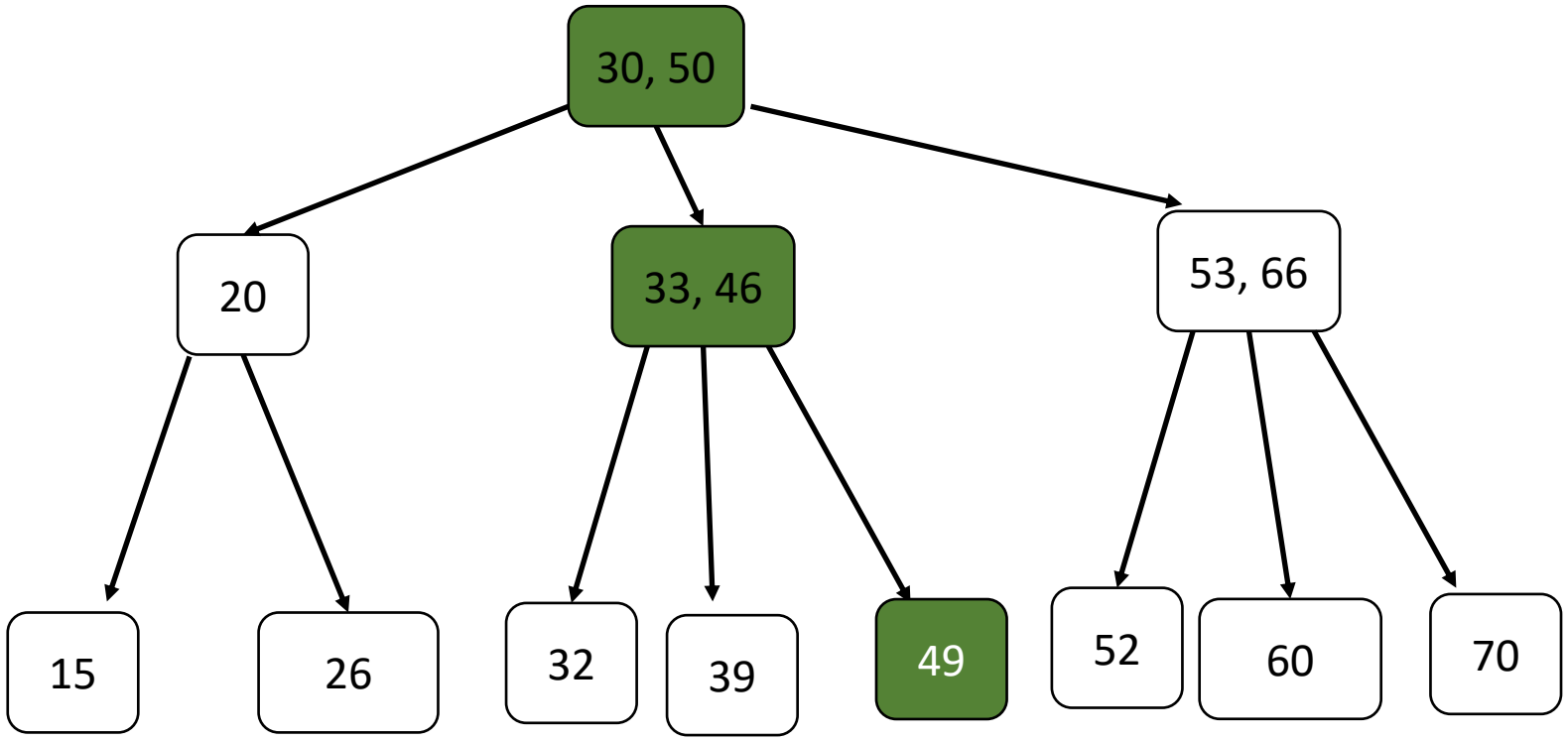
1. Search



Deletion

Delete 49

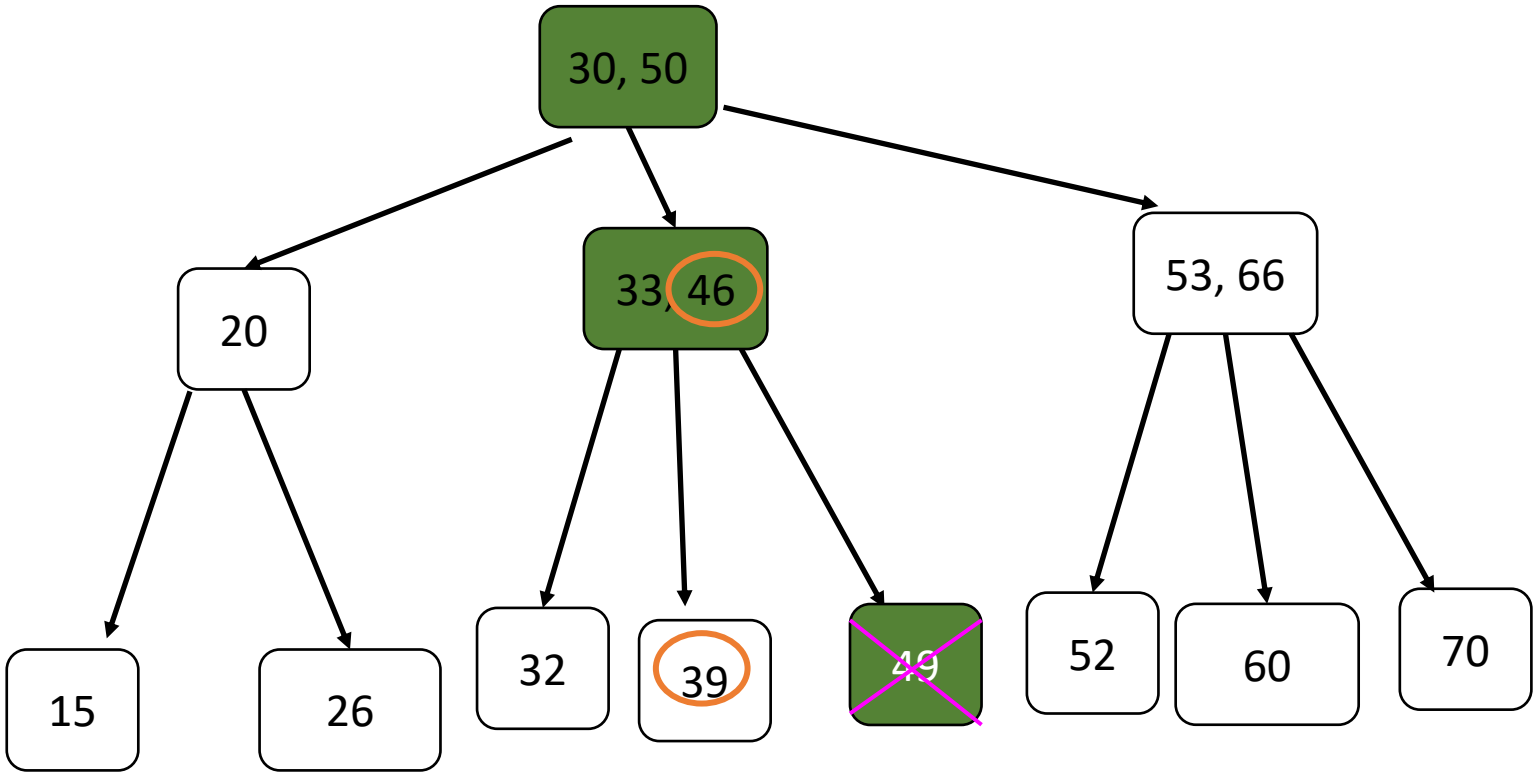
1. **Search**
2. **49 is contained in a 2-node, and both our siblings are 2-nodes as well… ☹**



Deletion

Delete 49

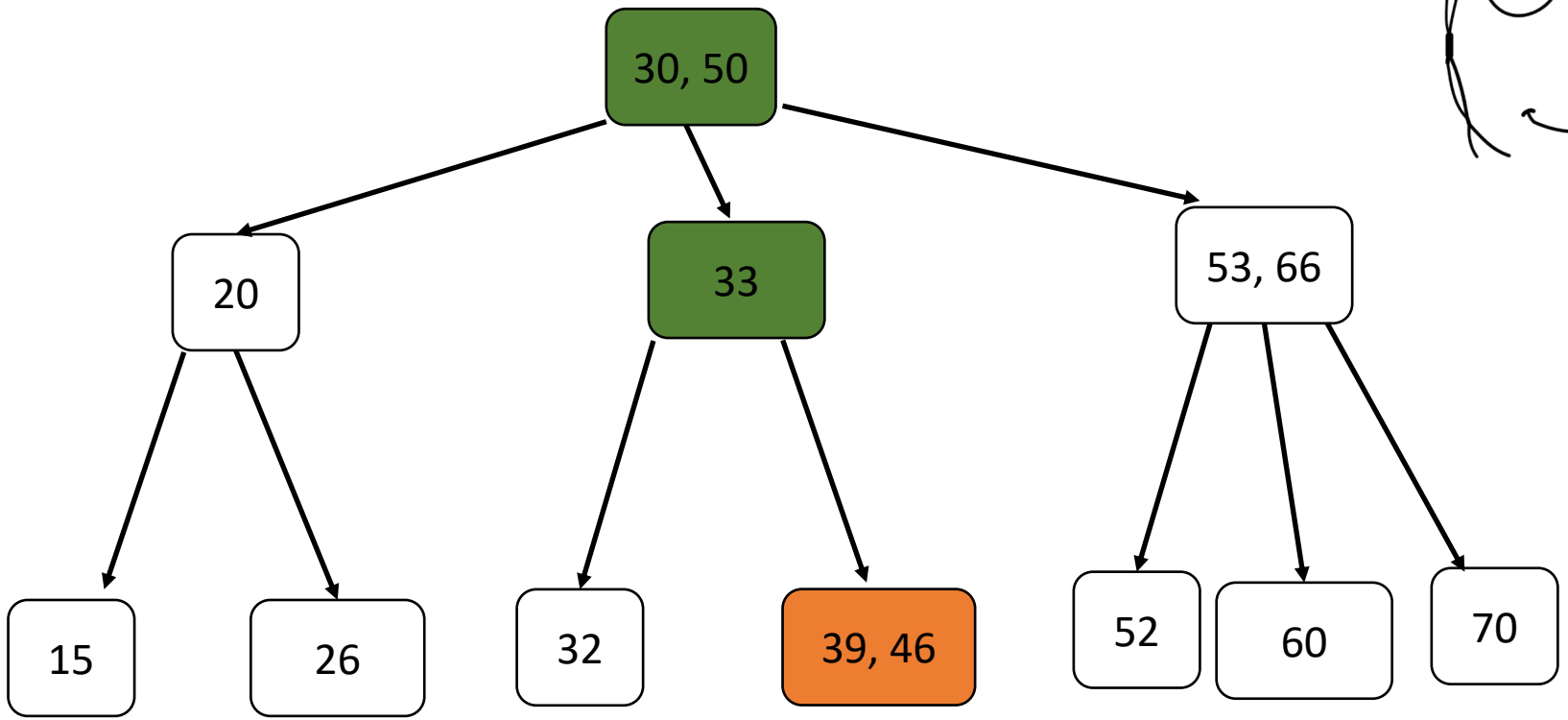
1. **Search**
2. **49 is contained in a 2-node, and both our siblings are 2-nodes as well… ☹**
3. **Solution: Merge parent separator with relevant sibling**



Deletion

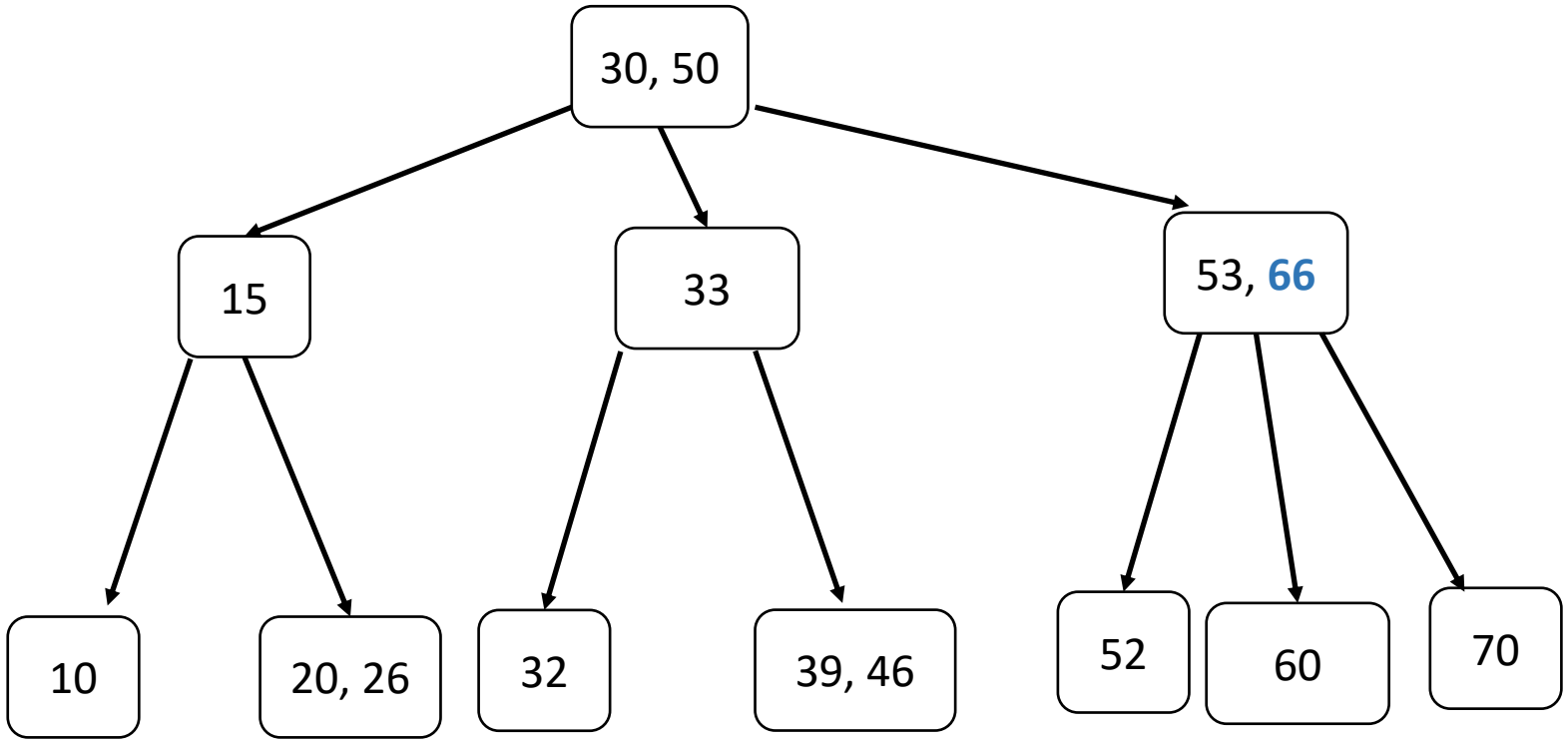
Delete 49

1. **Search**
2. **49 is contained in a 2-node, and both our siblings are 2-nodes as well… ☹**
3. **Solution: Merge parent separator with relevant sibling!**



Deletion

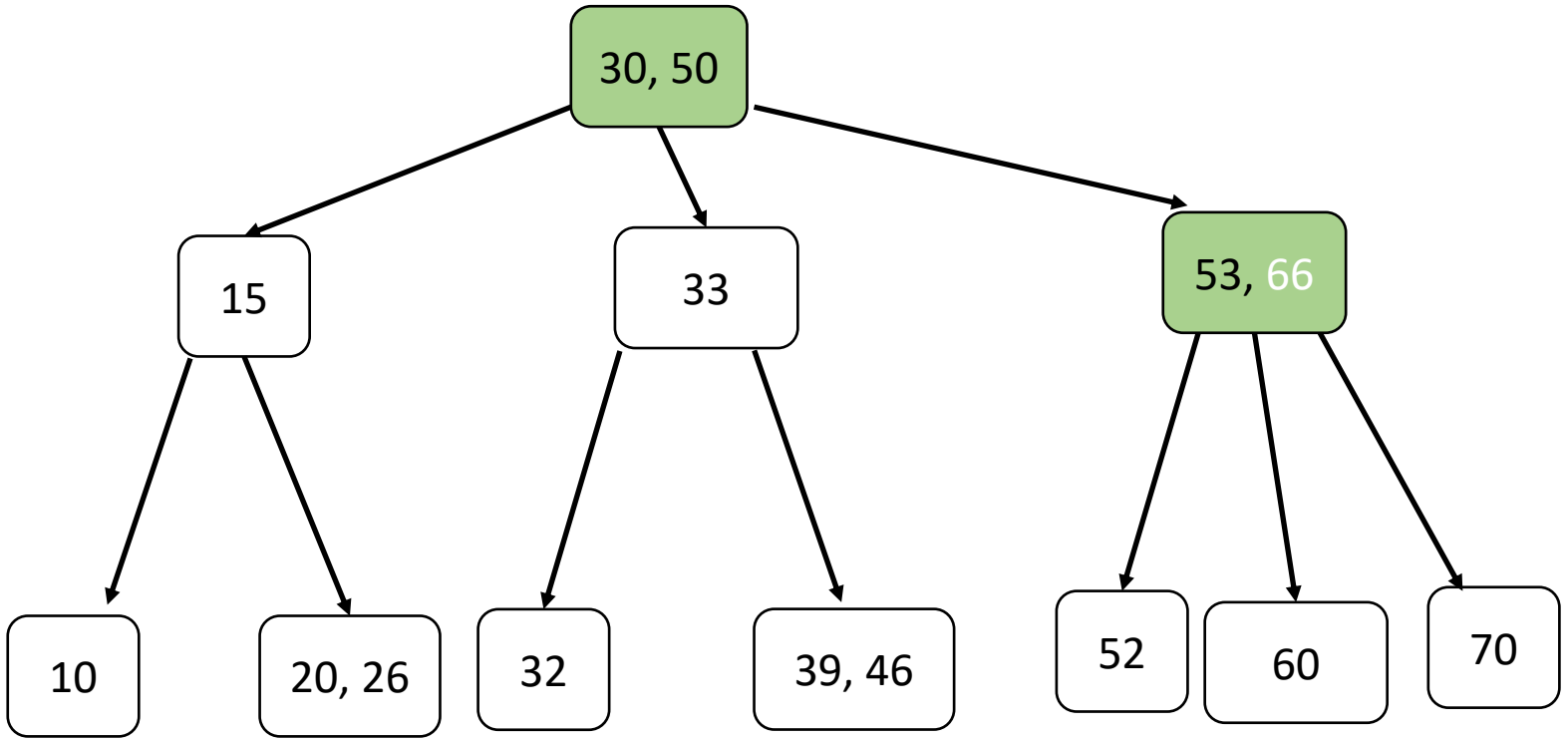
Delete 66



Deletion

Delete 66

1. Search

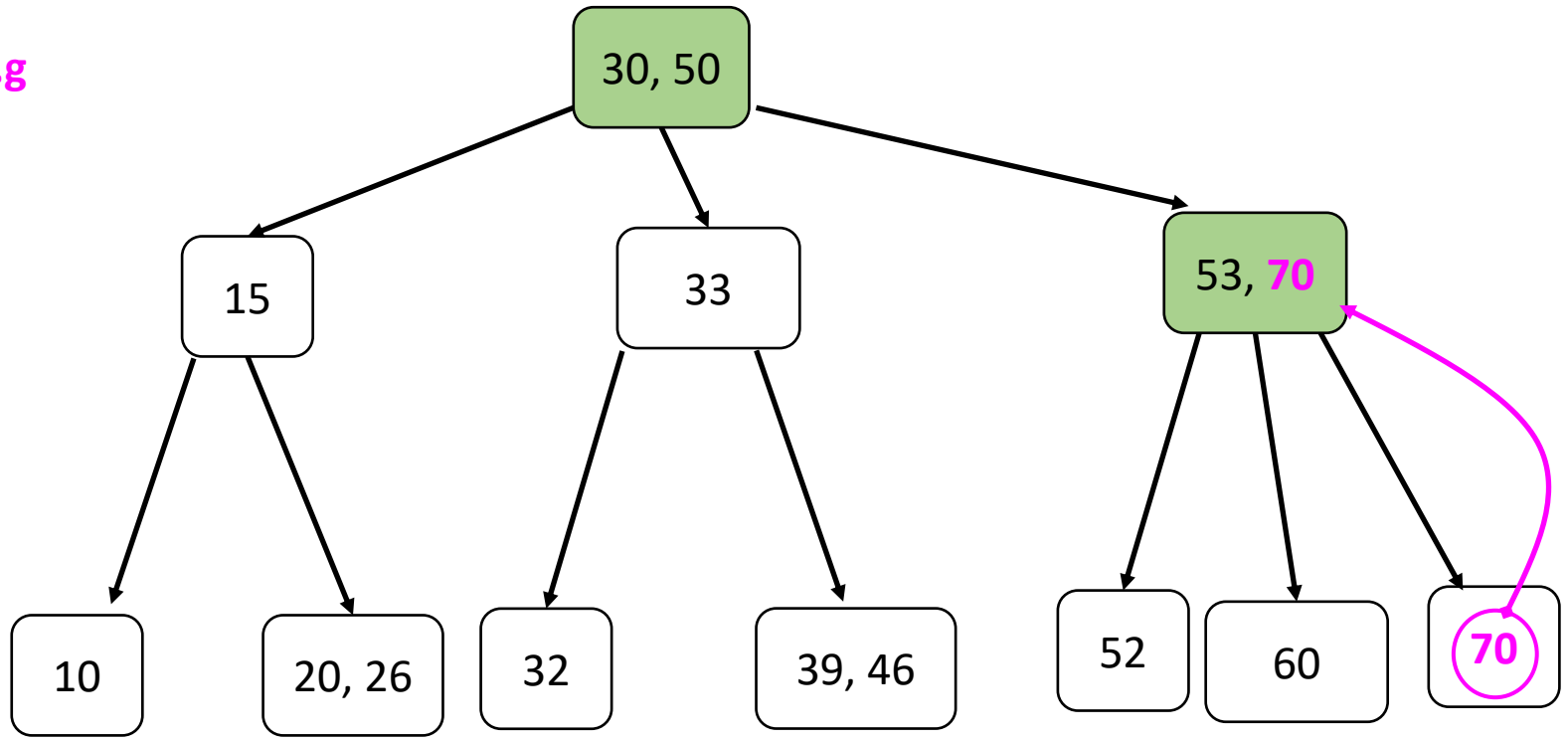


Deletion

Delete 66

1. Search

2. Have to replace key (e.g with inorder successor…)



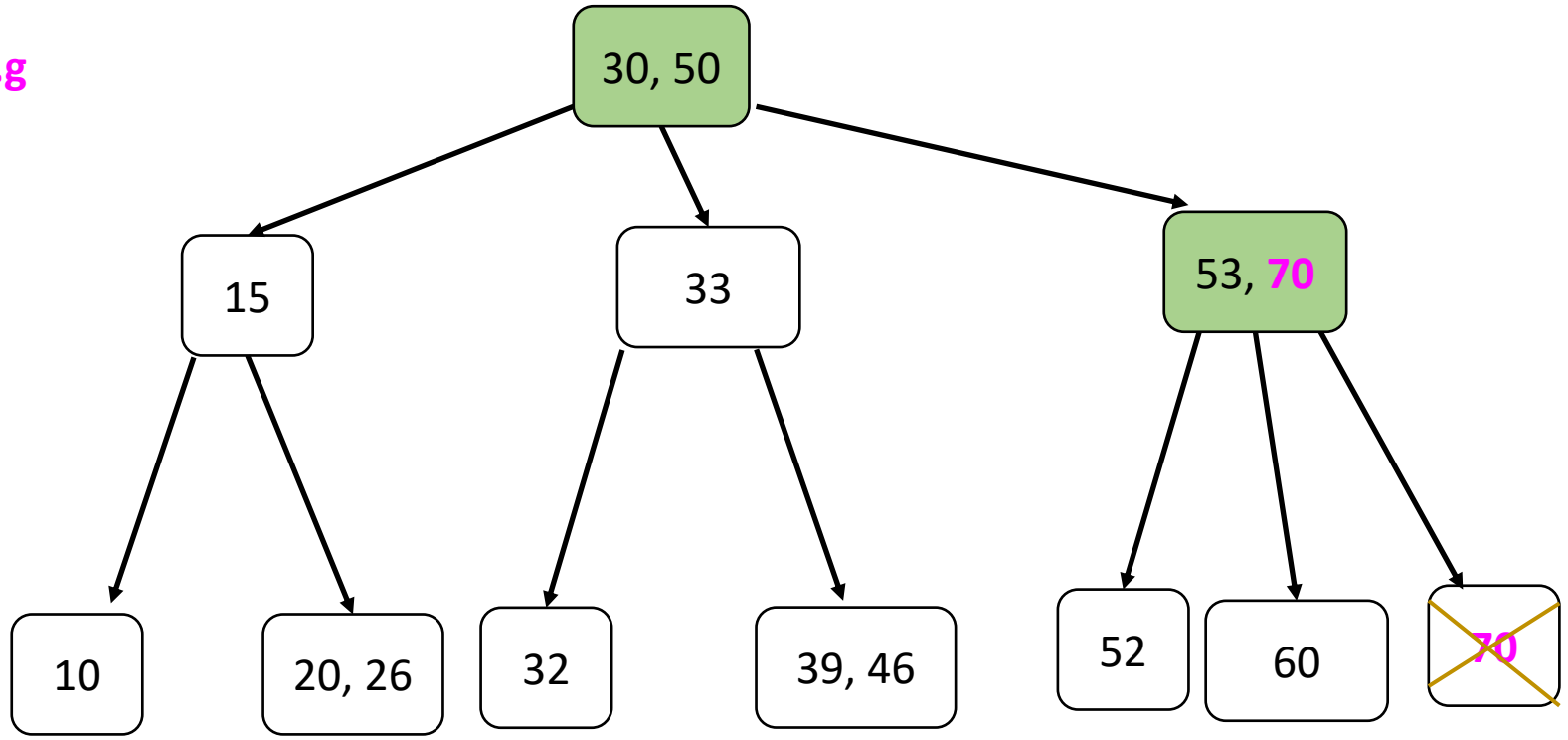
Deletion

Delete 66

1. Search

2. Have to replace key (e.g with inorder successor…)

3. Have to delete the inorder succesor



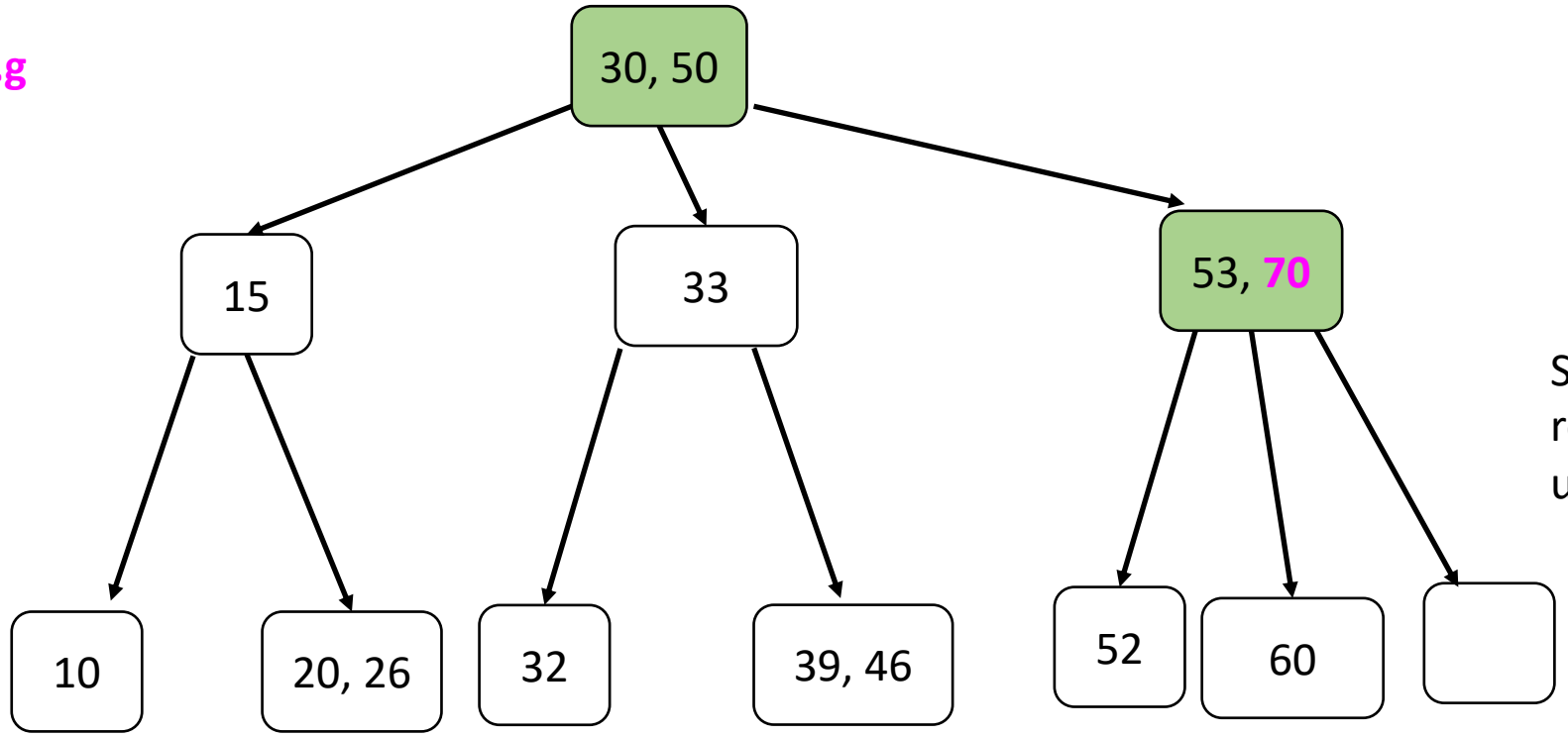
Deletion

Delete 66

1. Search

2. Have to replace key (e.g with inorder successor…)

3. Have to delete the inorder succesor



Siblings can't rotate keys to us…. ☹

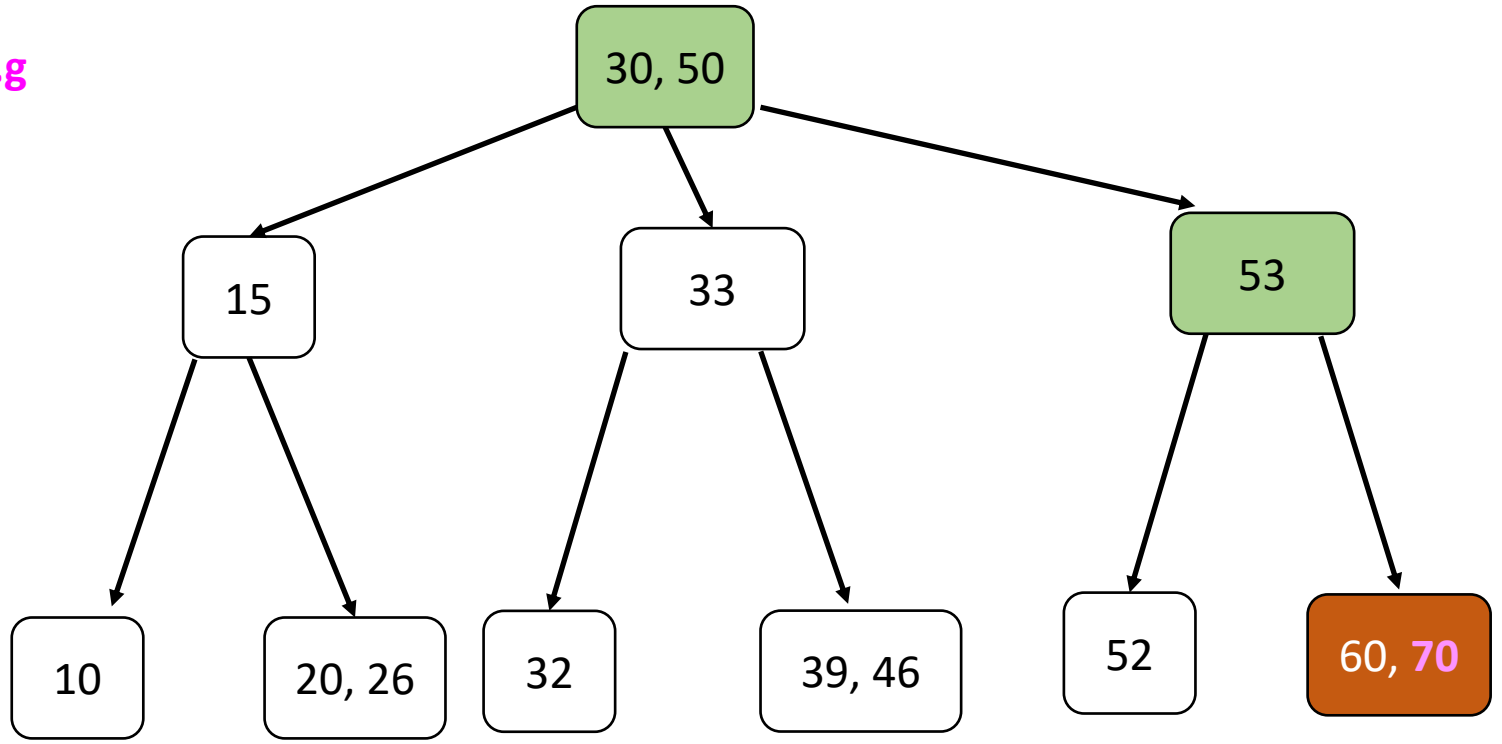
Deletion

Delete 66

1. Search

2. Have to replace key (e.g with inorder successor…)

3. Have to delete the inorder succesor



So we will have to merge our parent and sibling!

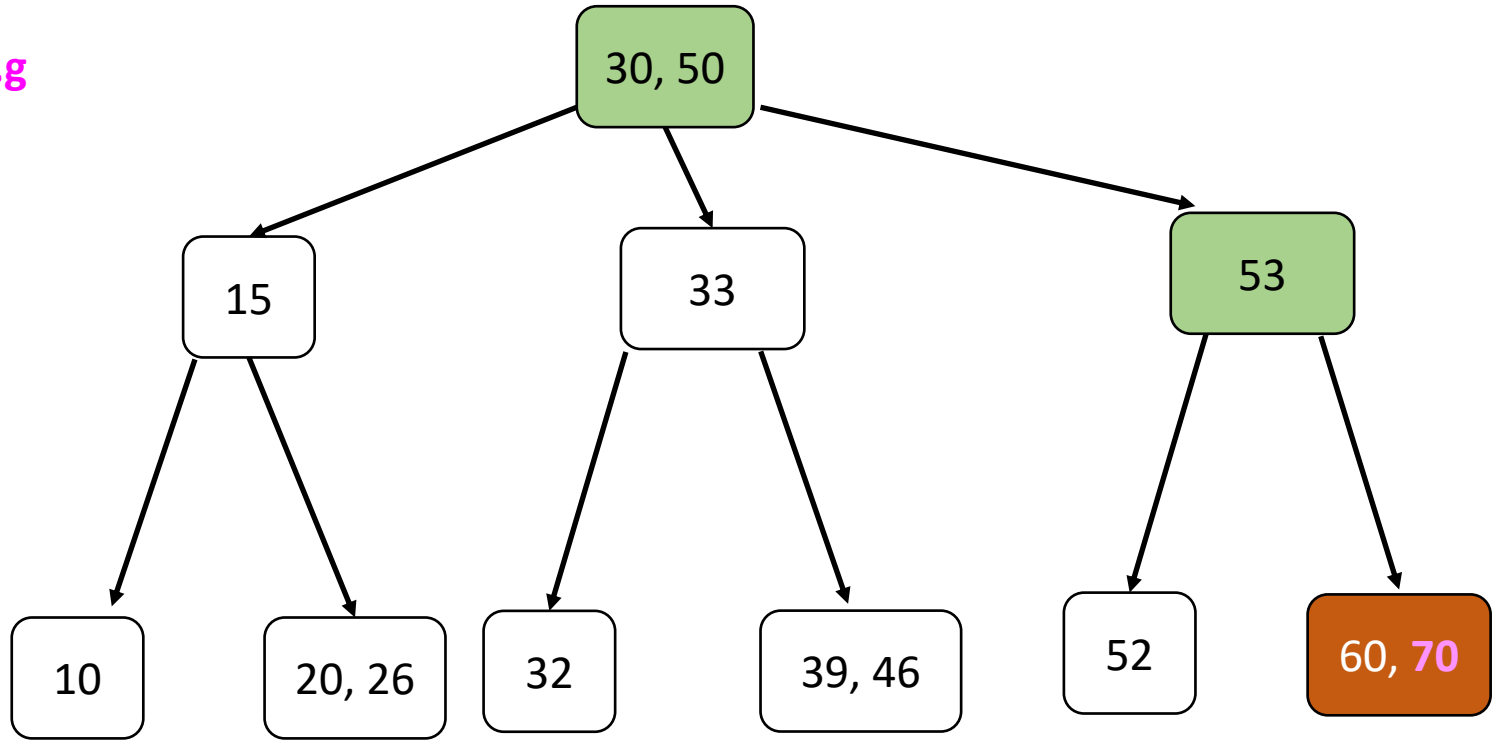
Deletion

Delete 66

1. Search

2. Have to replace key (e.g with inorder successor…)

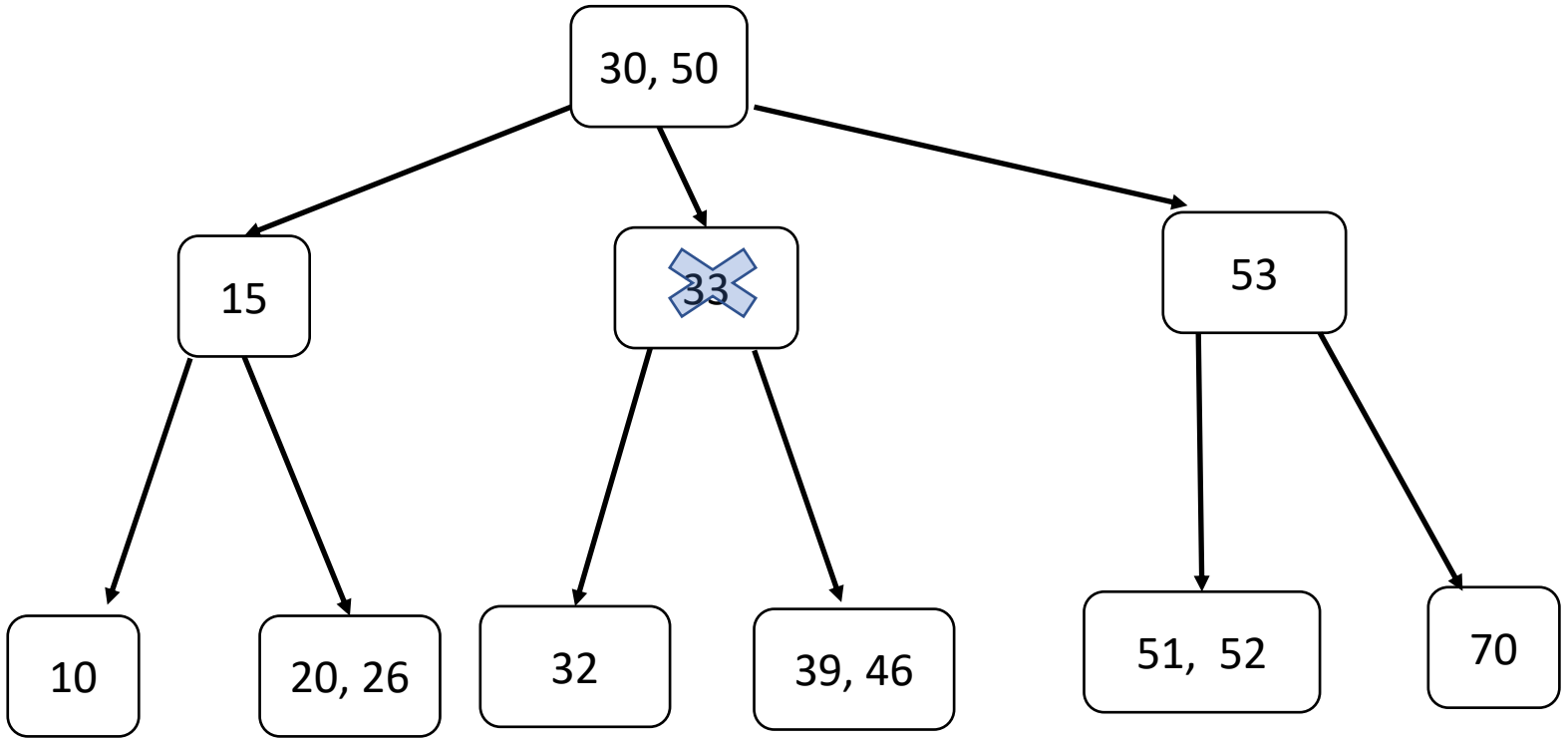
3. Have to delete the inorder succesor



So we will have to merge our parent and sibling!

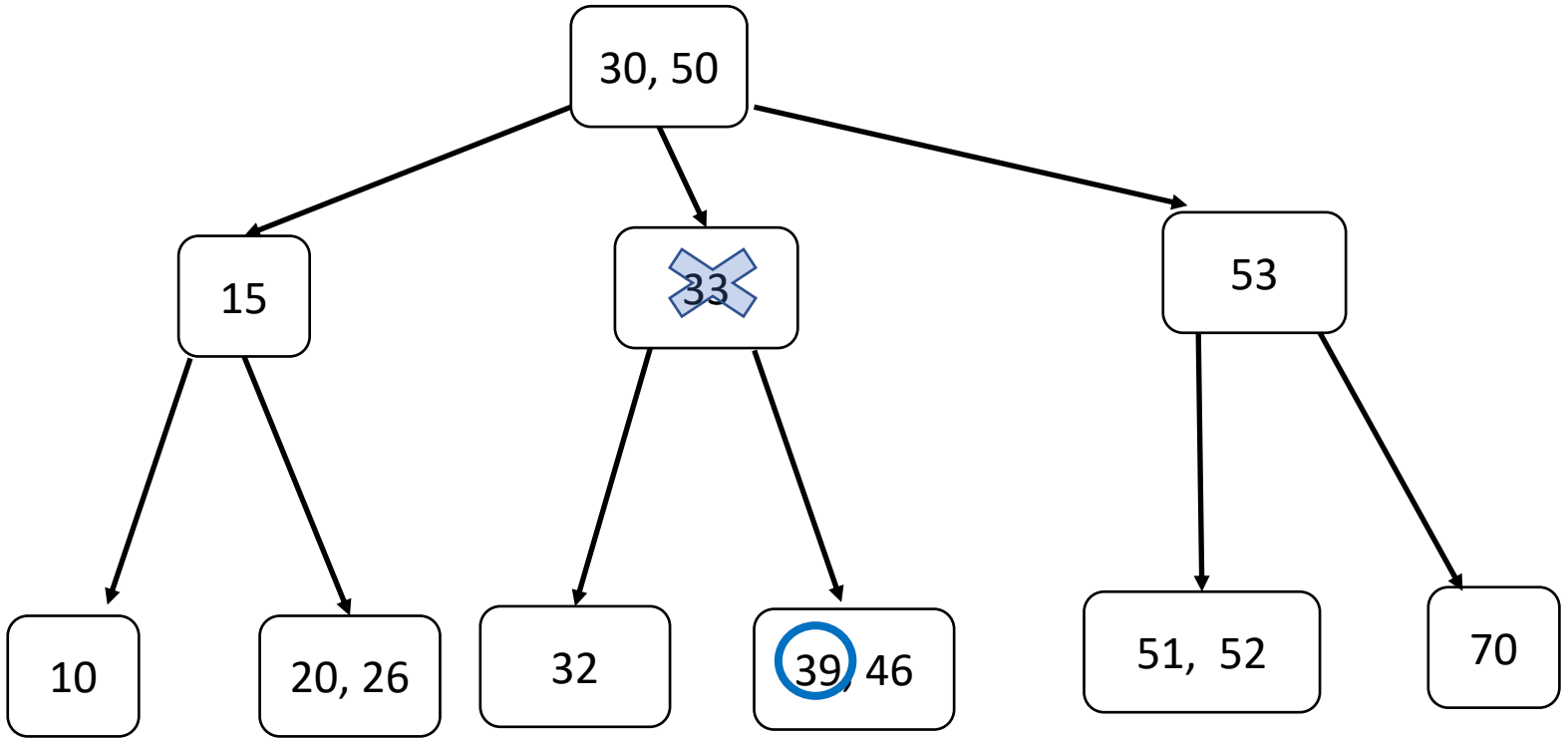
Deletion

Let's delete 33 real quick to make a point…



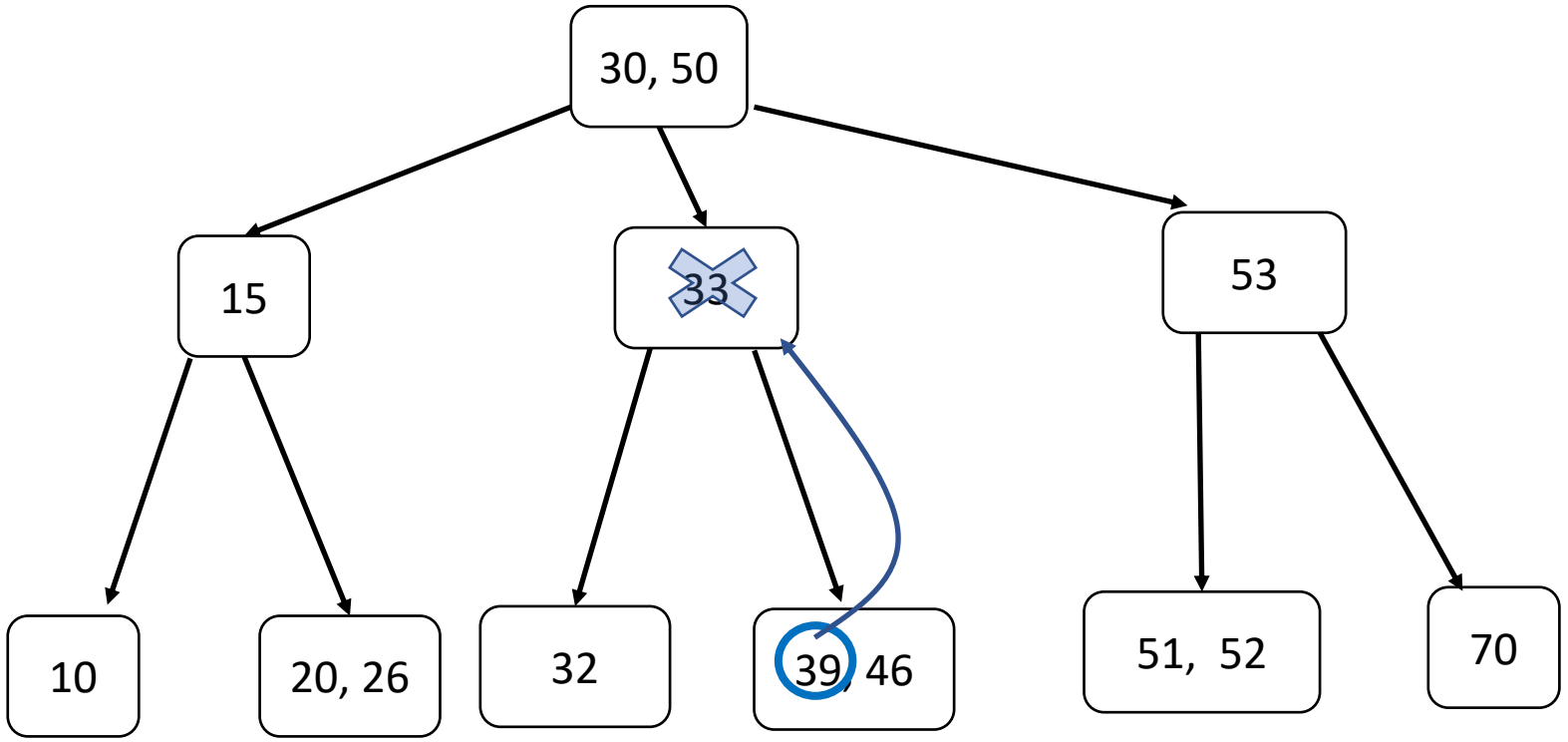
Deletion

Let's delete 33 real quick to make a point…



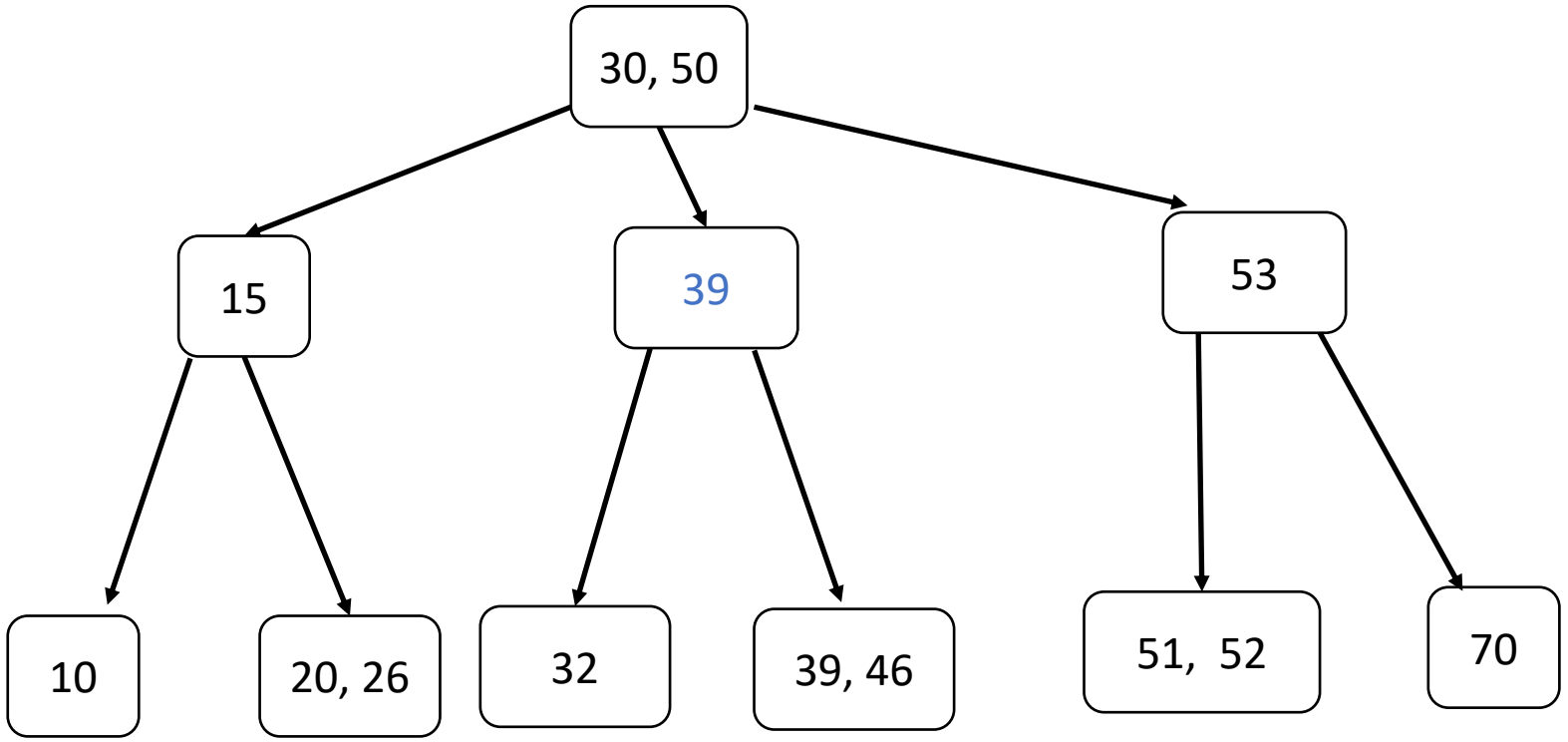
Deletion

Let's delete 33 real quick to make a point…



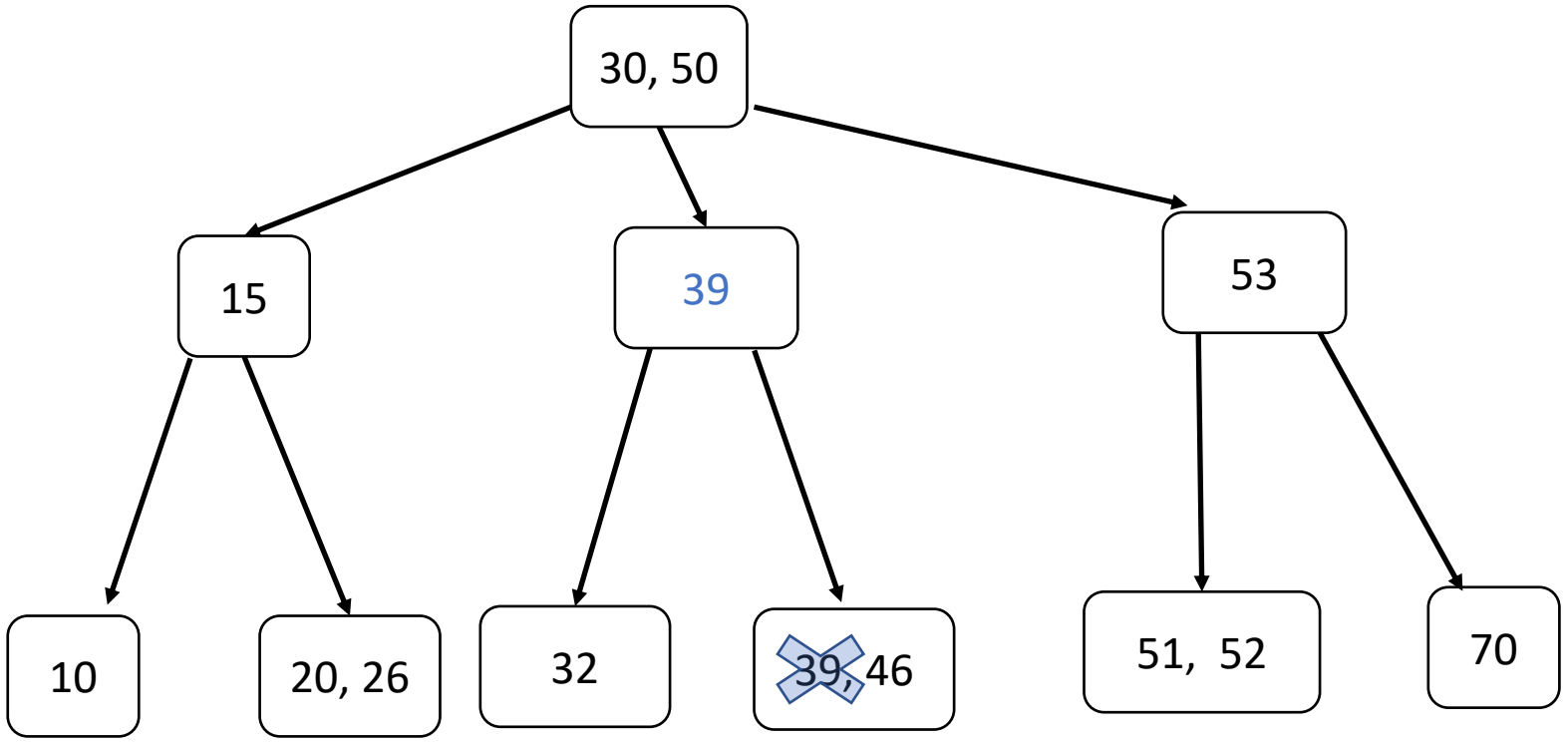
Deletion

Let's delete 33 real quick to make a point…



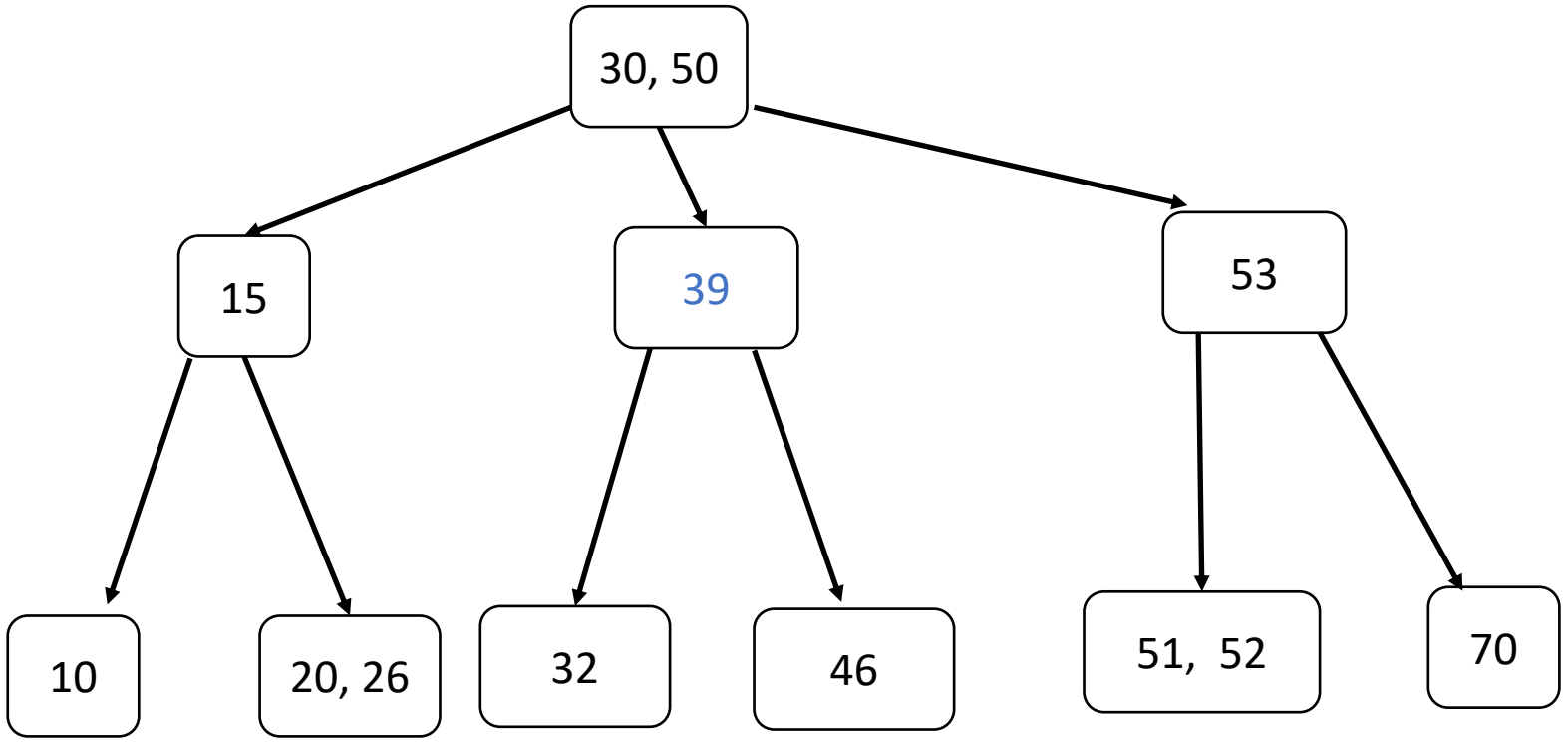
Deletion

Let's delete 33 real quick to make a point…



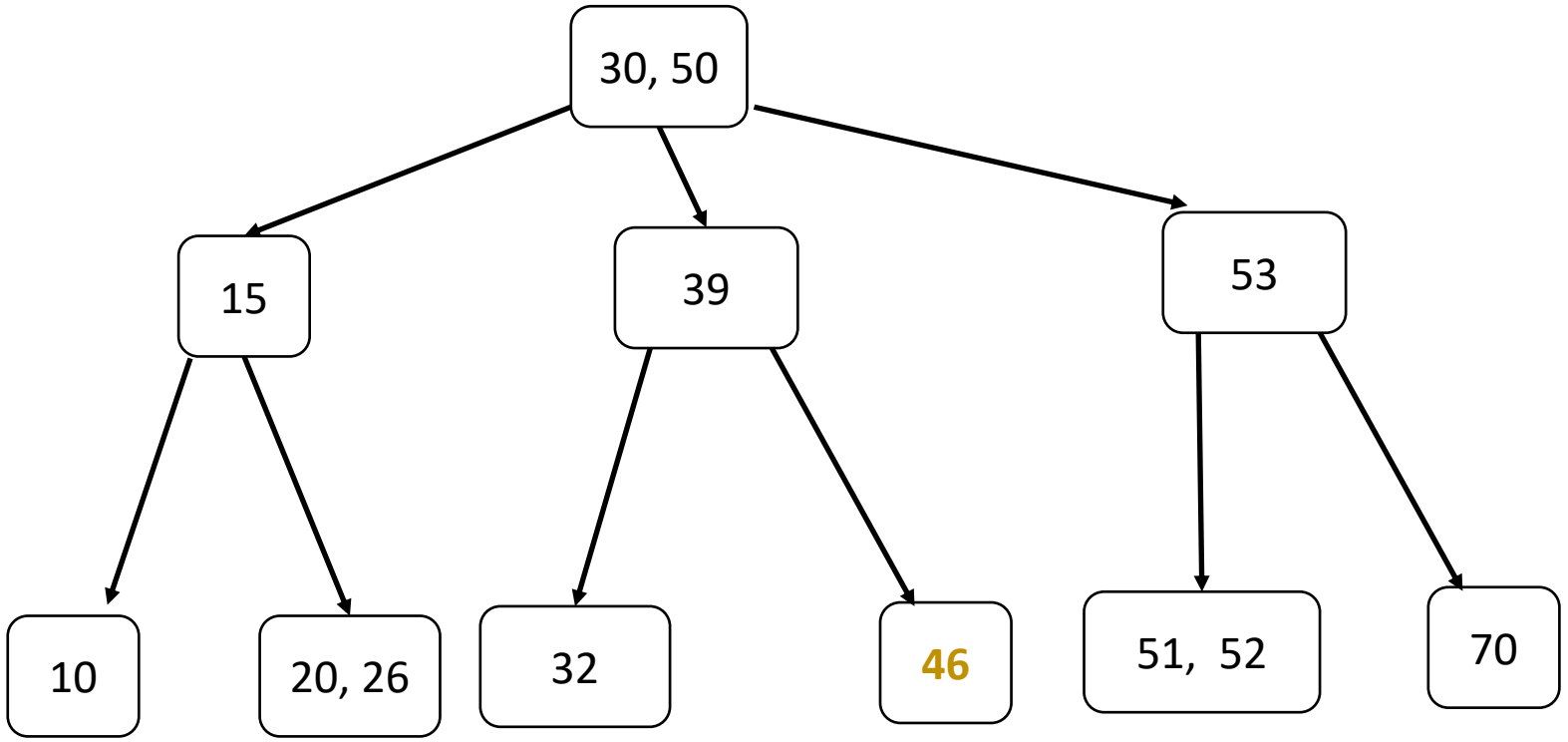
Deletion

Let's delete 33 real quick to make a point…



Deletion

Let's delete 33 real quick to make a point…
And now let's delete 46.

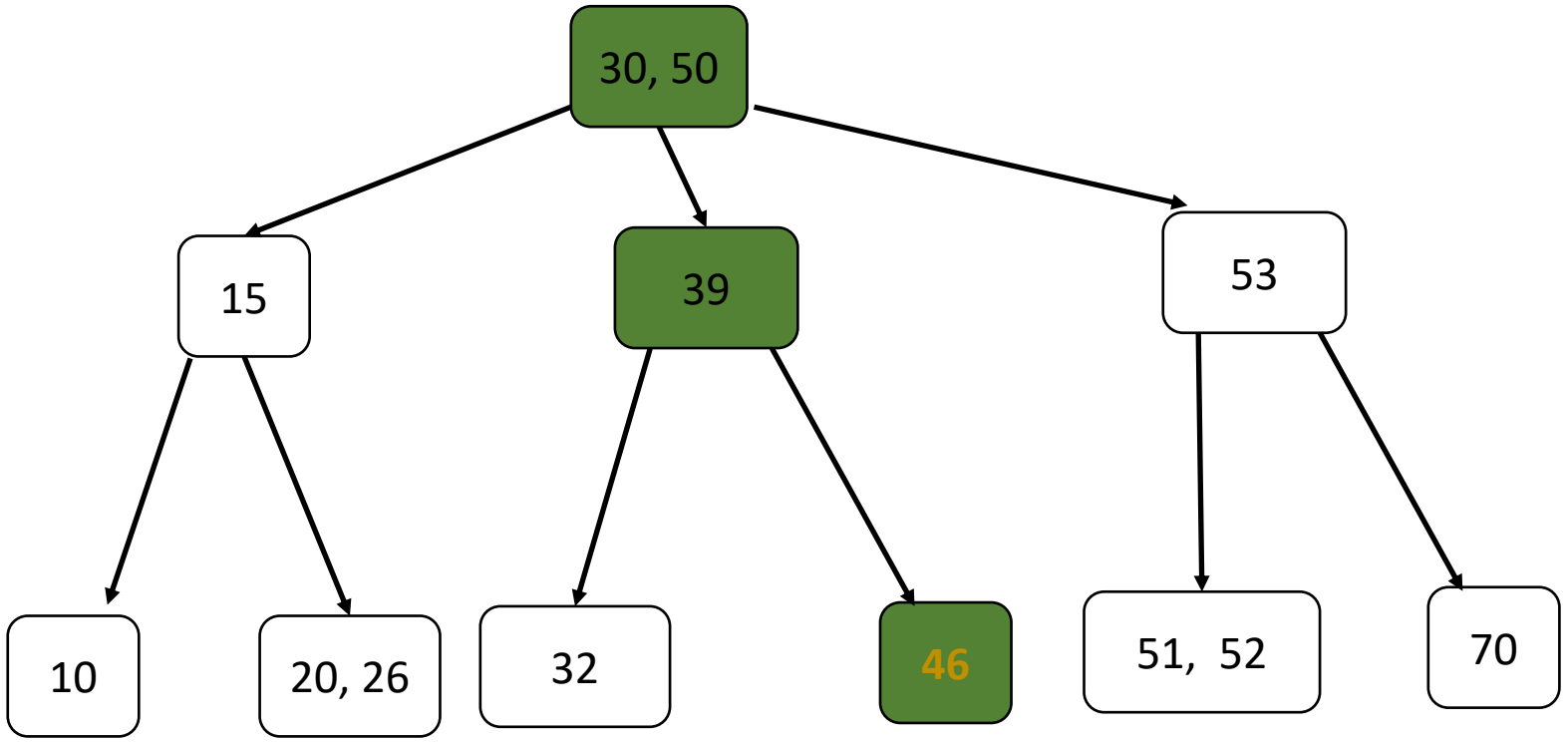


Deletion

Let's delete 33 real quick to make a point…

And now let's delete 46.

1. Search

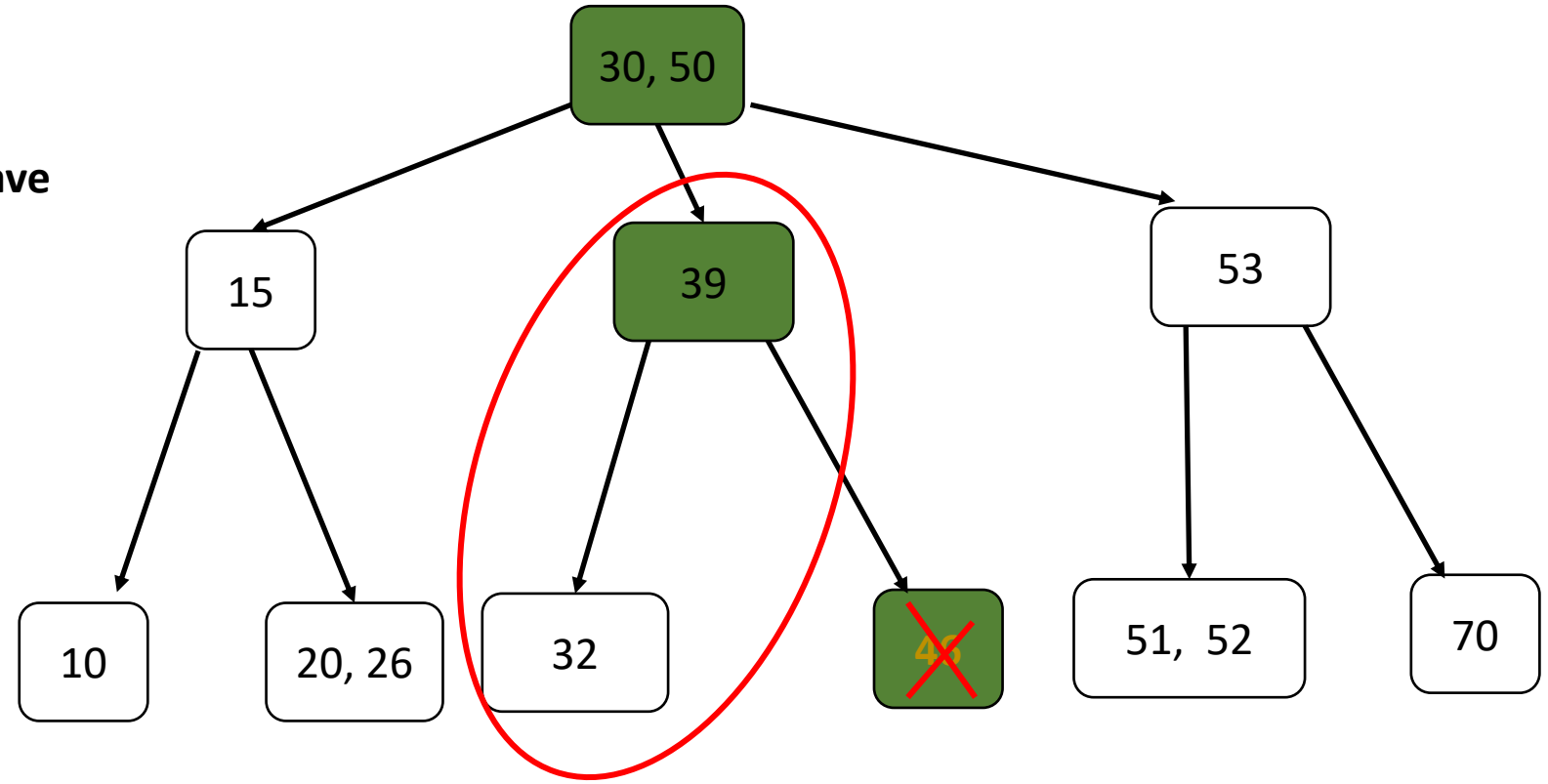


Deletion

Let's delete 33 real quick to make a point…

And now let's delete 46.

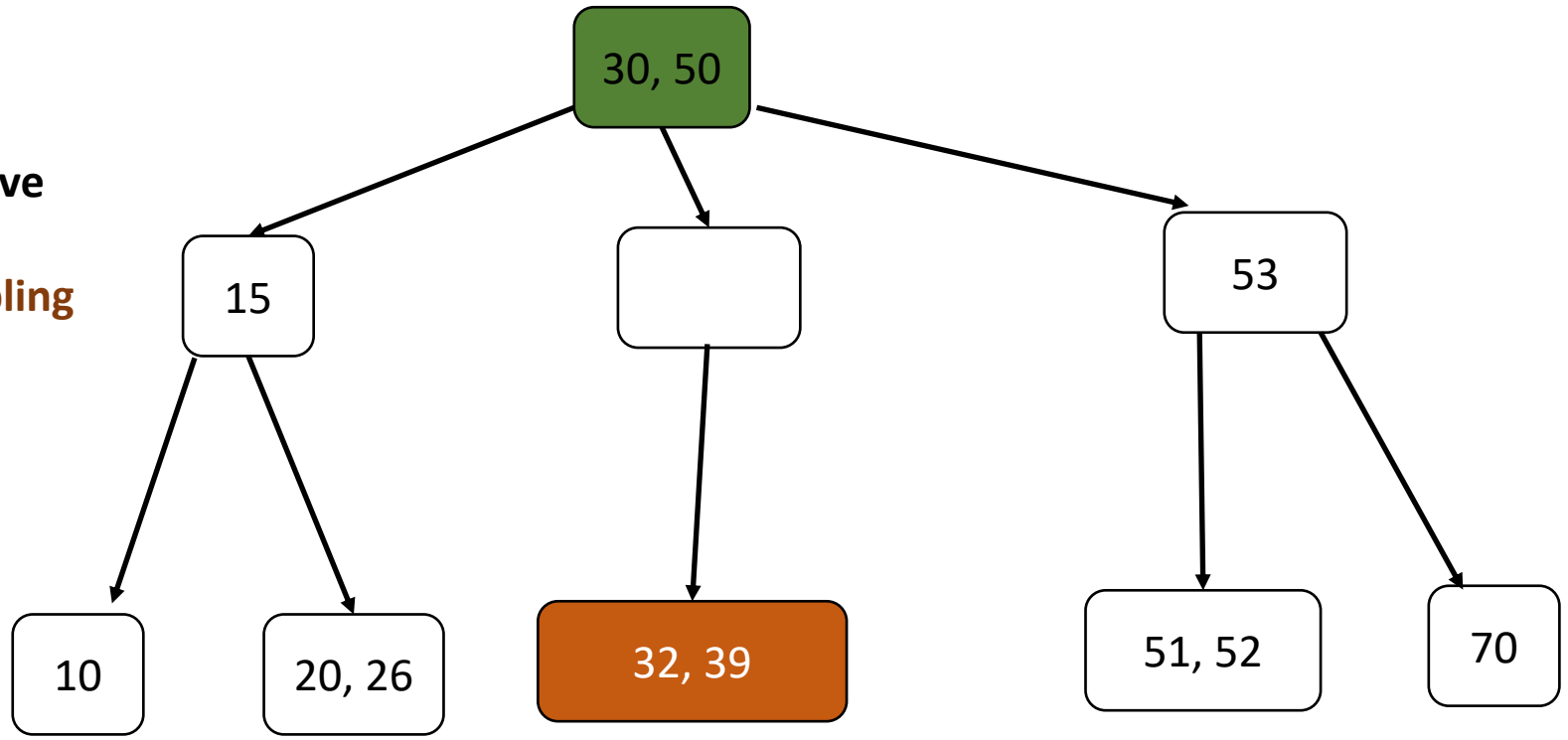
1. **Search**
2. **Sibling and parent have no keys to spare! ☹**



Deletion

Let's delete 33 real quick to make a point…
And now let's delete 46.

1. **Search**
2. **Sibling and parent have no keys to spare! ☹**
3. **Merge parent and sibling into a 3-node…**

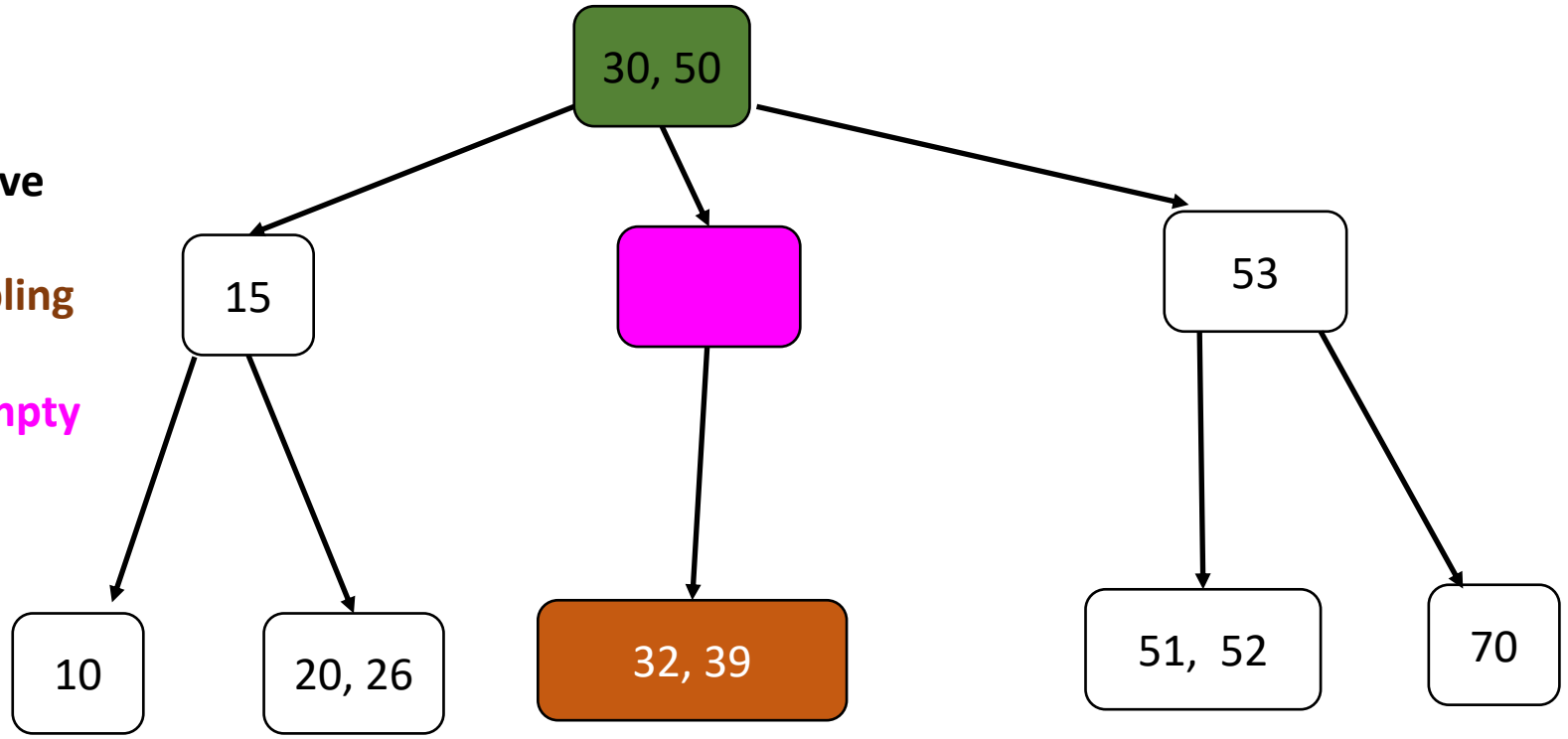


Deletion

Let's delete 33 real quick to make a point…

And now let's delete 46.

1. Search
2. Sibling and parent have no keys to spare! ☹
3. Merge parent and sibling into a 3-node…
4. Deal with the new empty node recursively!



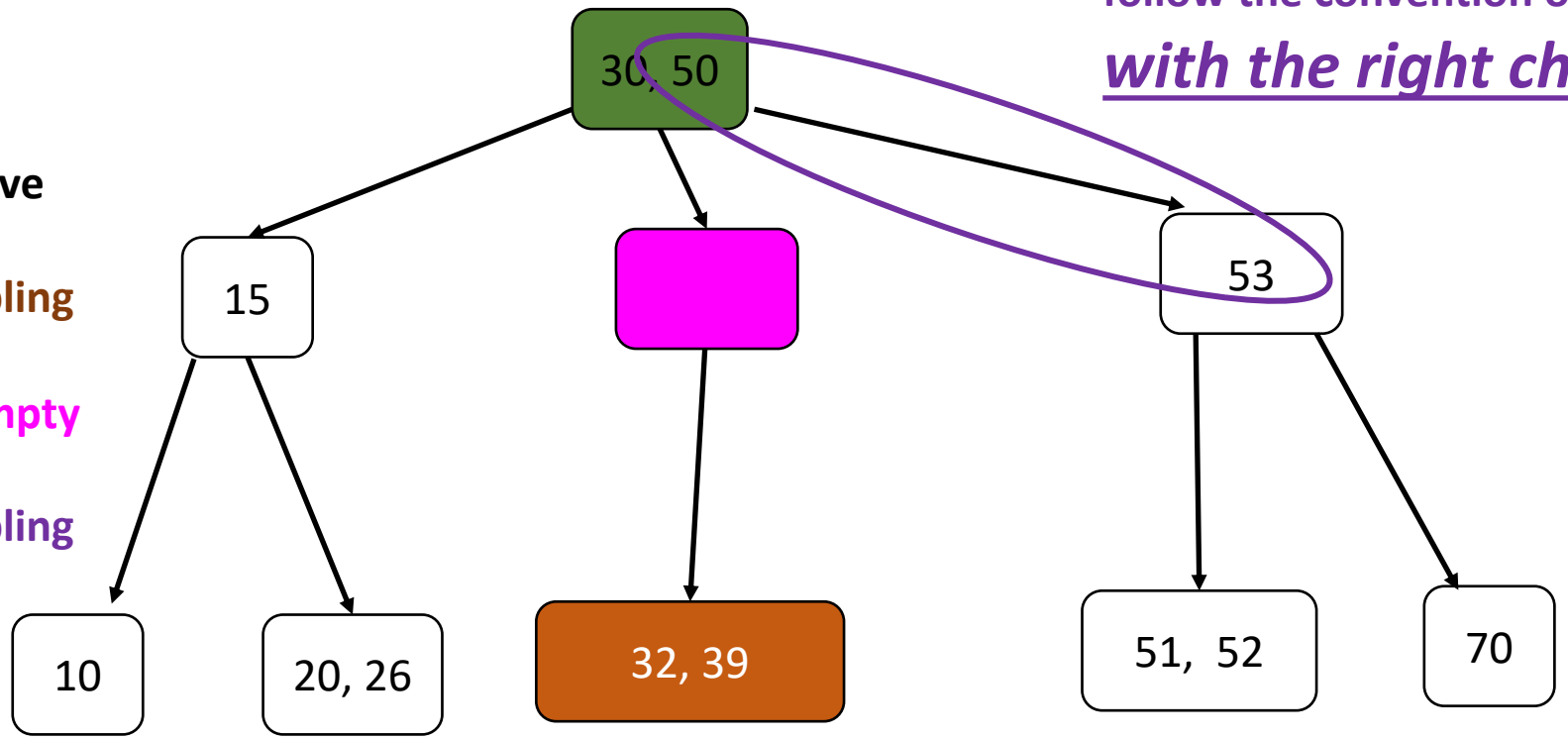
Deletion

Let's delete 33 real quick to make a point…

And now let's delete 46.

1. Search
2. Sibling and parent have no keys to spare! ☹
3. Merge parent and sibling into a 3-node…
4. Deal with the new empty node recursively!
5. Merge parent and sibling into a 3-node…

If we can merge with both siblings, let's follow the convention of ***merging with the right child.***



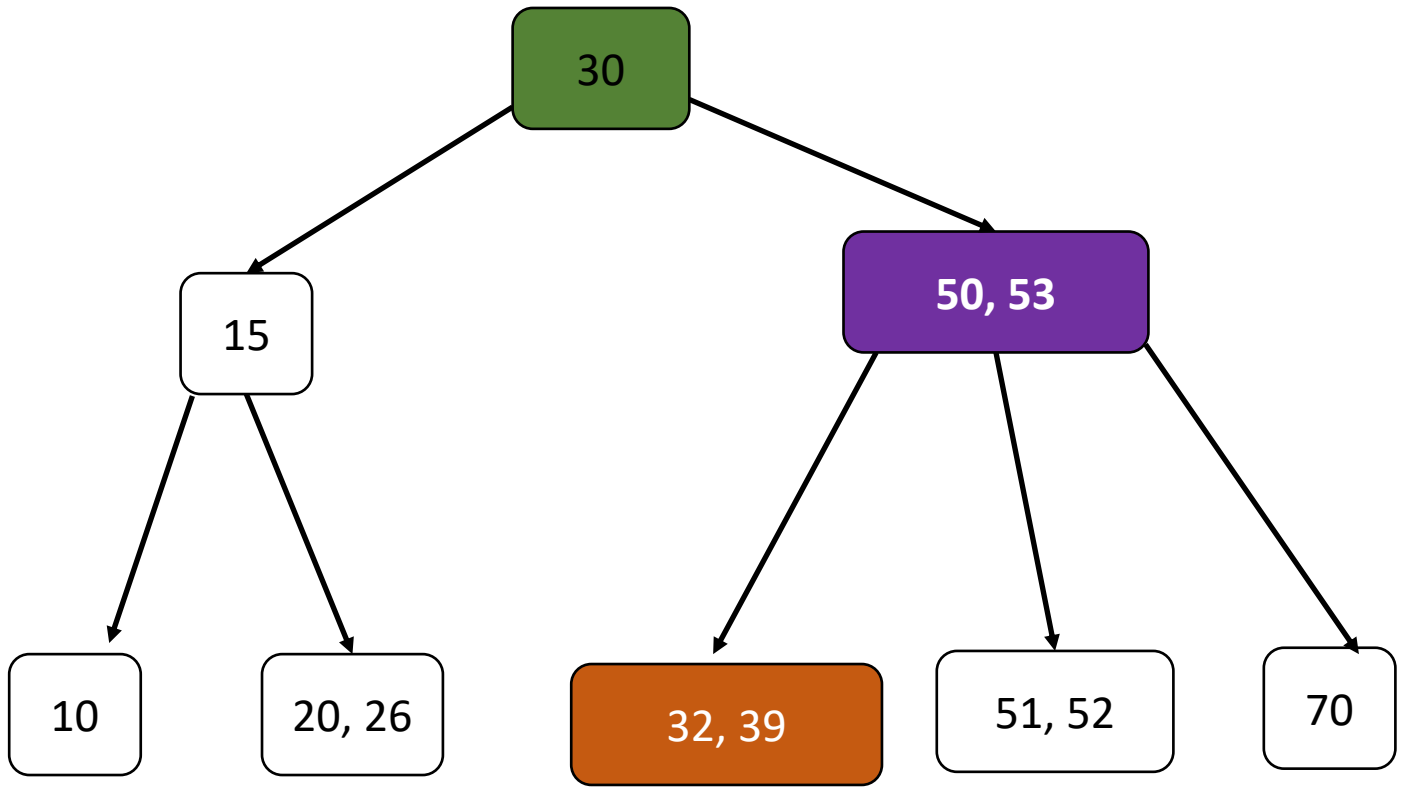
Problem 2 please!

Deletion

Let's delete 33 real quick to make a point…

And now let's delete 46.

1. Search
2. Sibling and parent have no keys to spare! ☹
3. Merge parent and sibling into a 3-node…
4. Deal with the new empty node recursively!
5. Merge parent and sibling into a 3-node…



Take-home message #2

- Deletions in 2- 3 trees can propagate **all the way up to the root**.
- They are **very expensive**.
  - Reason: Need to maintain **very useful property** of perfect balance.
- **It's worth it.** ***Data will be searched much more often than it will be deleted.***
- Mark-and-sweep schemes very popular in AVL trees and 2-3 trees
  - (And B-Trees and Red-Black Trees)

Implementing 2-3 Trees

- It turns out we can implement 2-3 trees as ***binary trees!*** 
- This tree will be known as a Red-**Black** binary search tree (R**B**BST)
- We will talk about it starting next time.

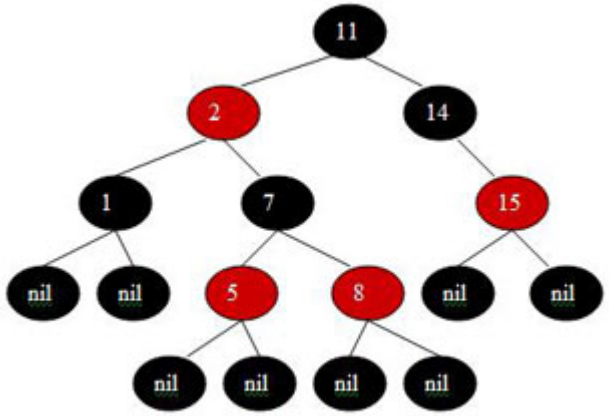


Figure 5. an example of a red-black tree