Insertion

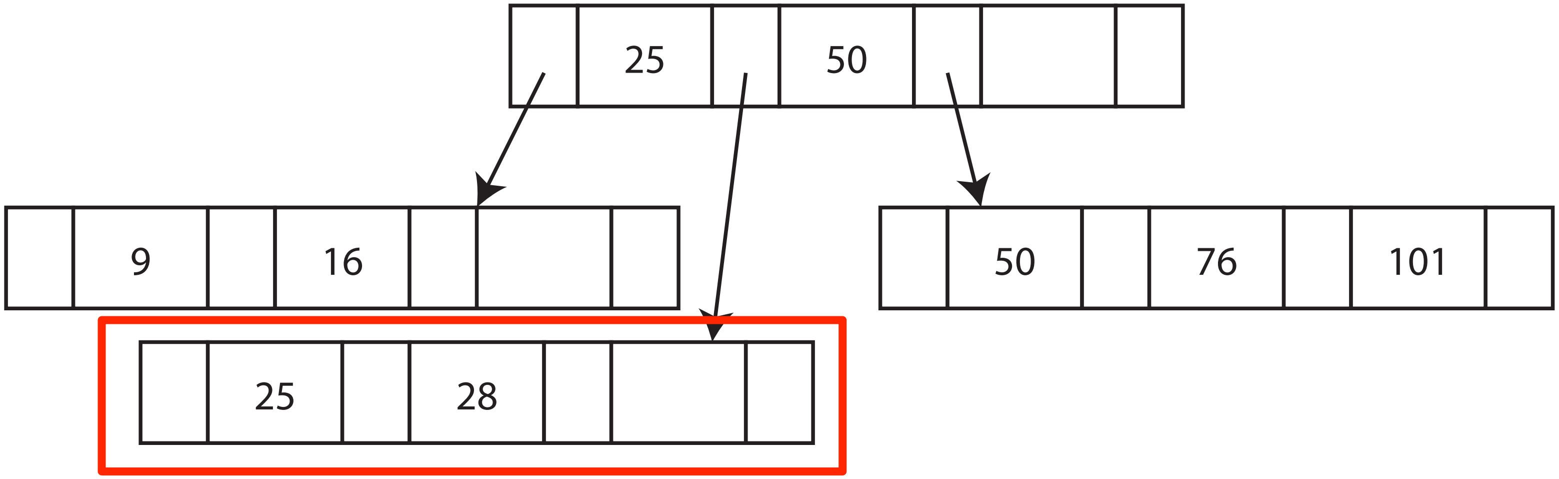
- A little complicated
  - There is no tree rebalancing
  - However, node split is sometimes required, and keys must be kept in order
- In essence:
  - Find the leaf node to should receive the new key/value pair
  - Try to insert into the leaf node
- Three distinct cases
  1. Target (leaf) node has space for one more key
  2. Target (leaf) node is full, but parent (interior) node has space for one more key
  3. Neither target nor parent have space.

```
insert_into_node(target_node, newkey, value)
```

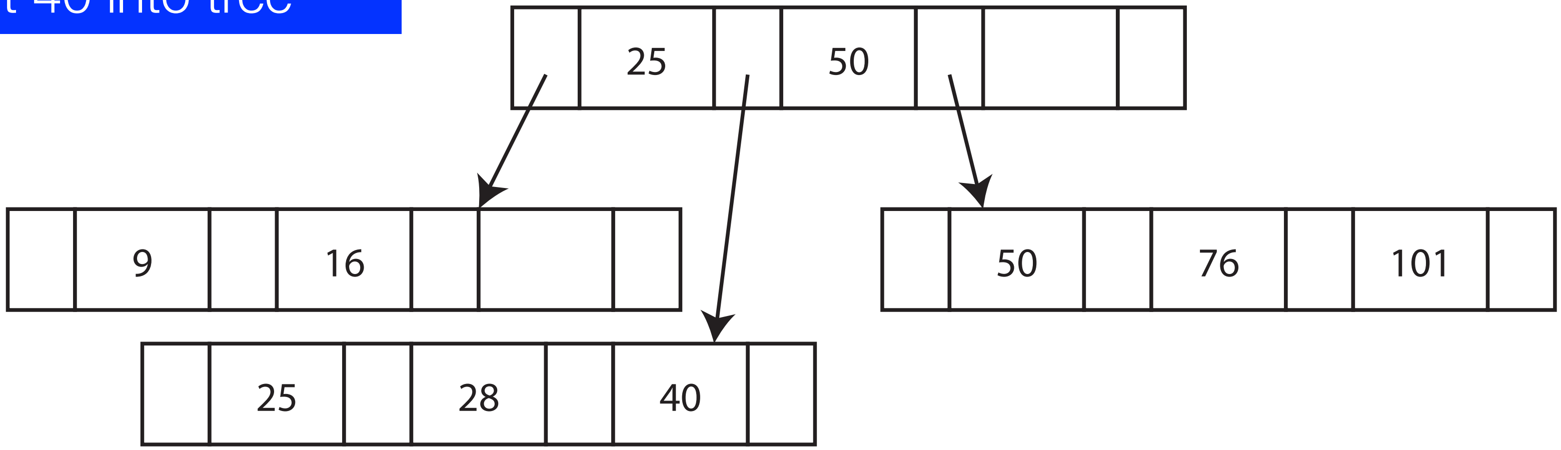
Insertion: case 1

- Target leaf has space for one more key/value pair
  - We'll ignore the value for the example, concentrate on the key
  - Also: similar structure of interior and leaf nodes will be exploited.
  - We will use nodes with degree three (i.e., three key/value pairs in leafs; three keys in interior node, and these have four pointers to blocks.
- We place the key/value into the target such the result has keys in order.

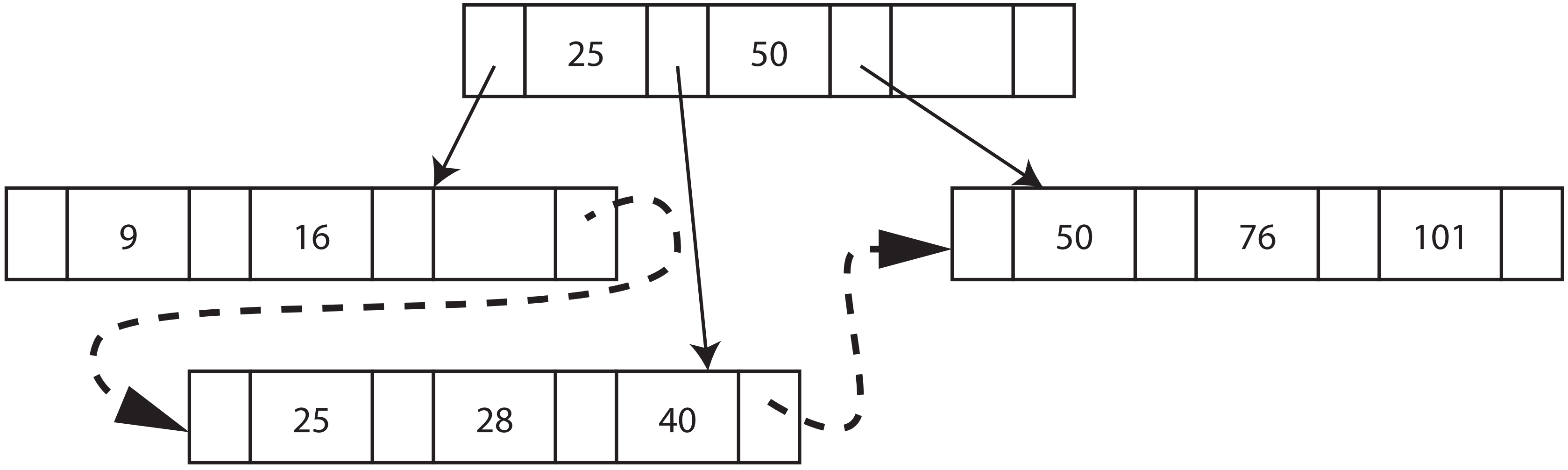
Insertion (case 1)



Insert 40 into tree



Implied: leafs as singly-linked list



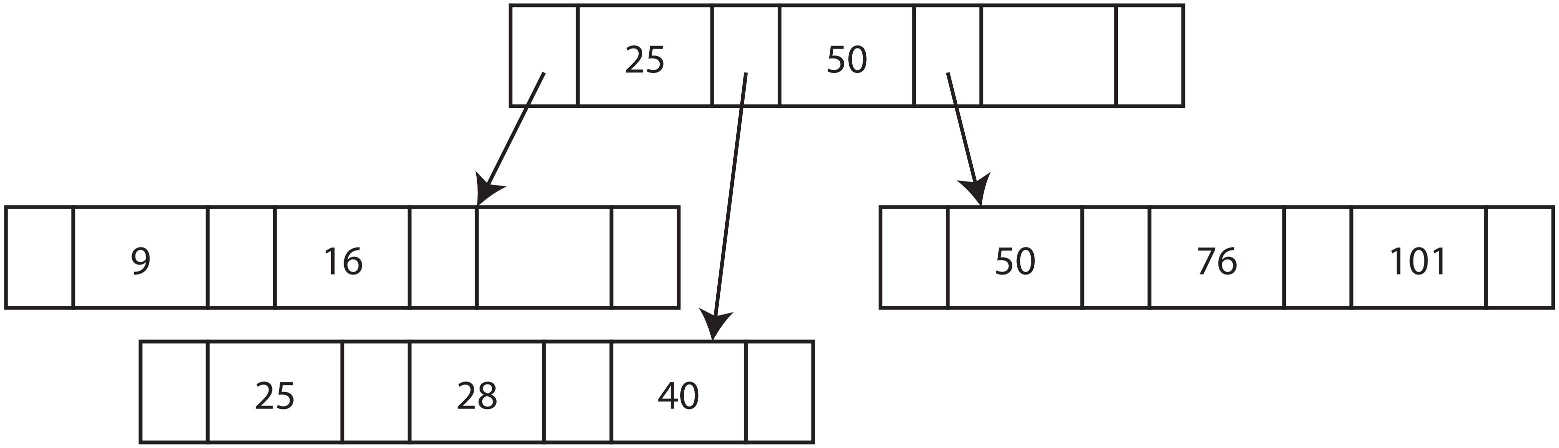
Insertion: case 2

- Target leaf is full, but parent node has space for at least one more key
  - Again, the result must be that all keys within nodes are in search-key order
  - We must be able to traverse the leaf nodes such that all keys are in order
- We place the key/value into the target such the result has keys in order.

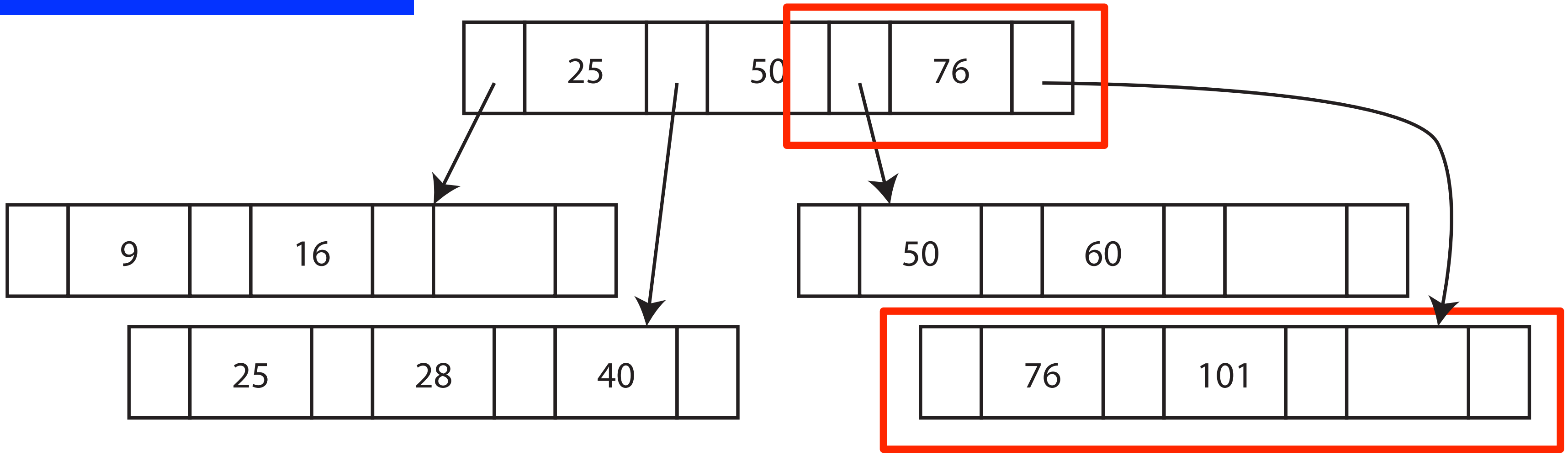
Insertion: case 1

- Target leaf is full, but parent node has available space for one more key
  - A new leaf node will be created.
  - Key/value pairs from the target leaf and the new leaf – including the one to be inserted – are distributed amongst the two nodes.
  - New leaf node must now have a pointer to it from the parent node (i.e., we must add it).
  - That last step might involve shifting entries around in the parent node. At the very least a new key value will be placed into the parent node.
- We place the key/value into the target such the result has keys in order.

Insertion (case 2)



Insert 60 into tree



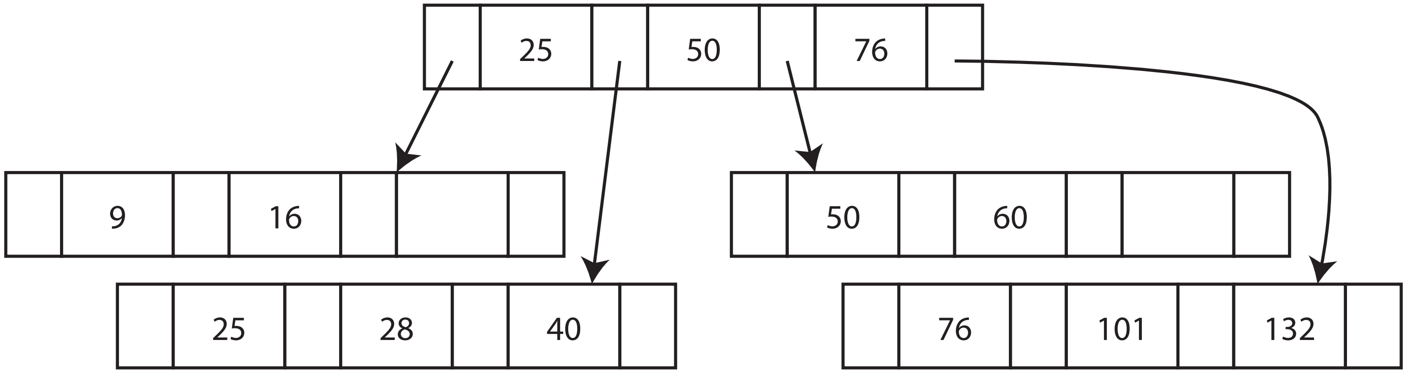
Insertion: case 3

- Target leaf is full, and parent node is full
  - Must recursively attempt an insert into the ancestors of the target leaf.
  - Interior nodes may split as a result.
  - In fact, even the root node may split!
- Procedure:
  - Create new leaf node, and insert it into the linked list of leafs in a position after the target leaf.
  - Distribute entries amongst these two leafs (as in case 2)
  - Then we follow the "insert/split" algorithm

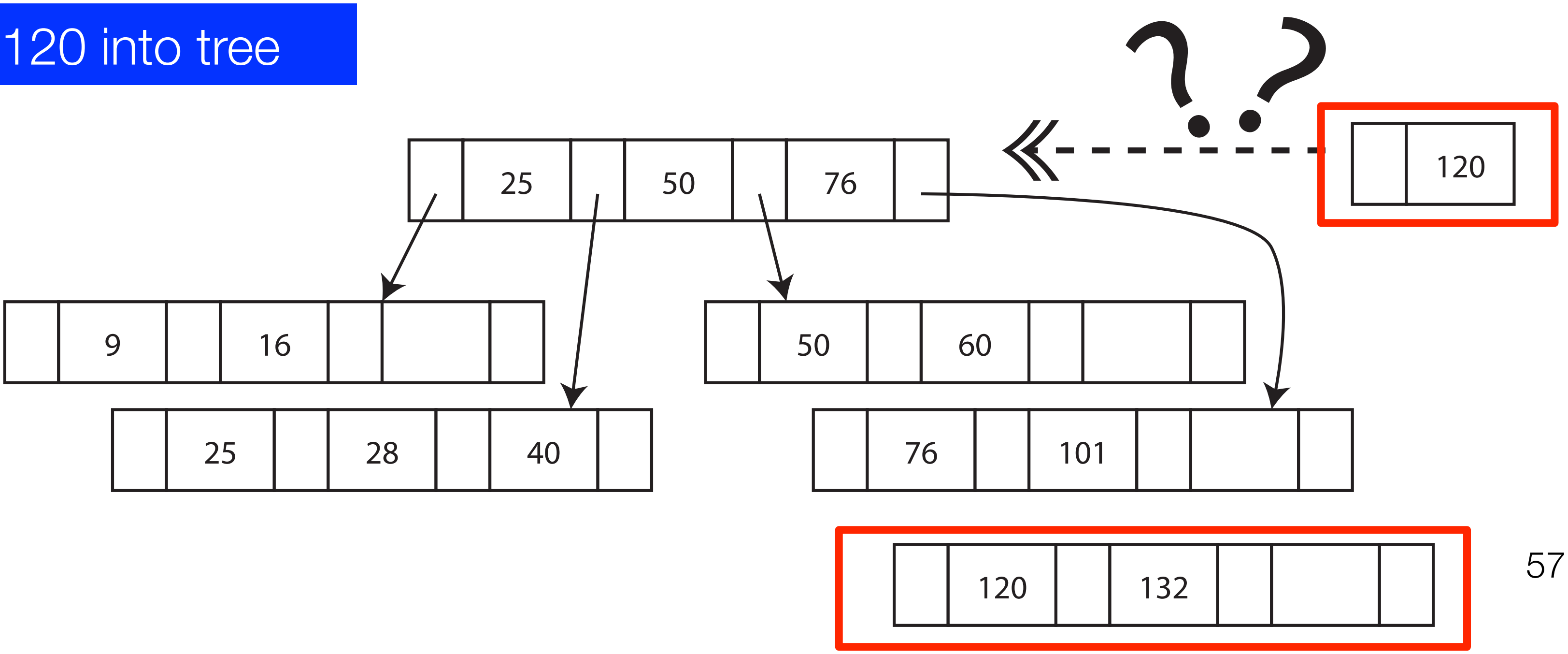
Case 3: insert/split step

```
target_parent_node = target_leaf_node.parent()

all_keys = target_parent_node.keys() UNION {k}

new_interior_node = new node()

i = floor((all_keys.size() + 1) / 2)

middle_key = all_keys[i]

target_parent_node.keys() ← all_keys[0 : i-1]  # Unlike python, include i-1!
new_interior_node.keys() ← all_keys[i : n-1]   # Unlike python, include n-1!

if (target_parent_node == root {
    target_grandparent_node = new node()
} else {
    target_grandparent_node = target.parent_node.parent()
}

# now make a recursive call to insert routine
#
insert_into_node(target_grandparent_node, middle_key, &new_interior_node)
```

Insertion: case 3



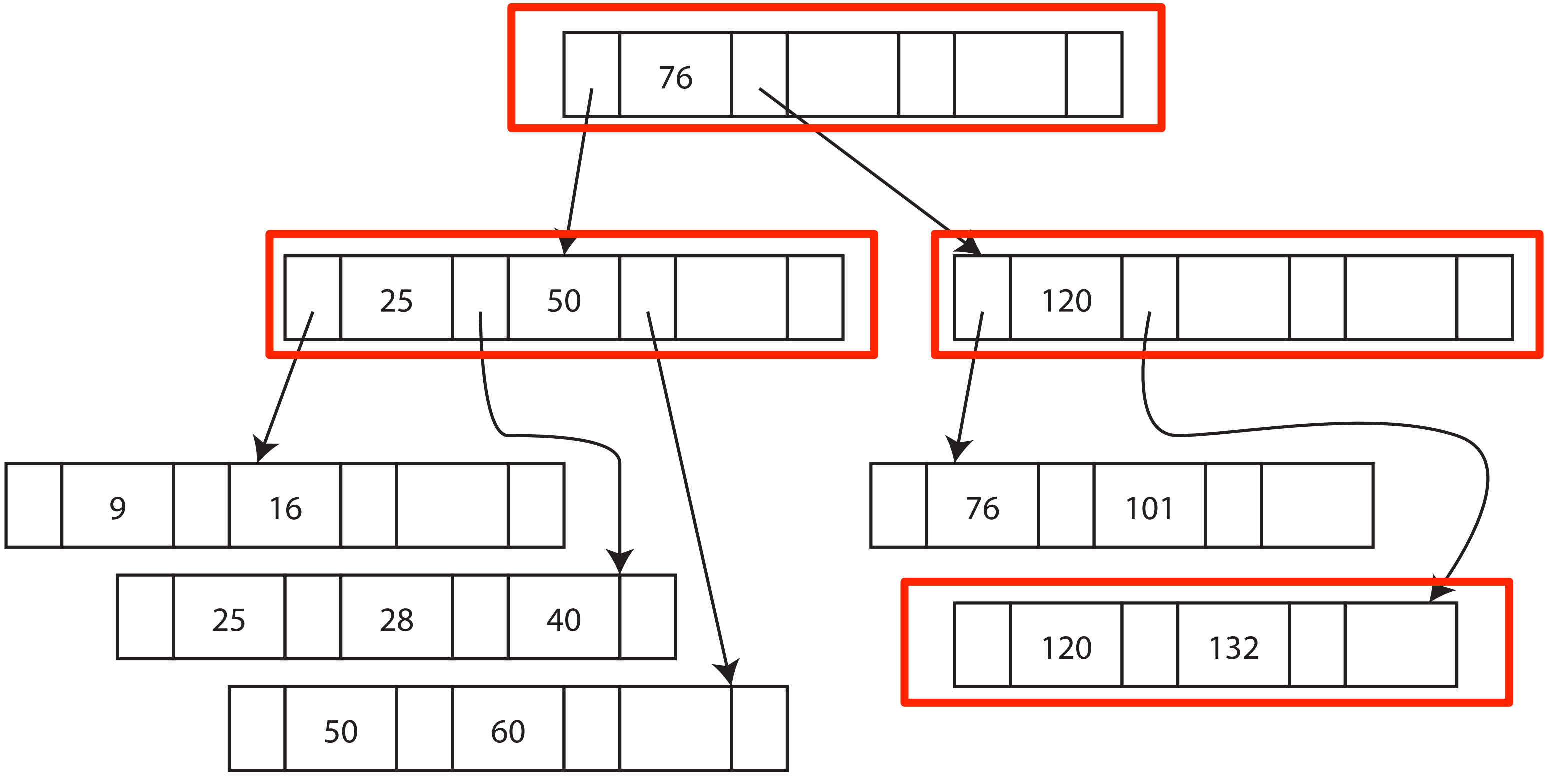
Insert 120 into tree



Insertion: case 3

Interior node keys + new key

25 50 76 120



Let's try this

- Here is a sequence of keys. (We'll ignore values as they are trivial to include but add to length of example)

- Sequence:

  3, 1, 41, 5, 59, 35, 8, 97,
  9, 32, 38, 4, 62, 6, 43, 383, 27,
  95, 20, 44