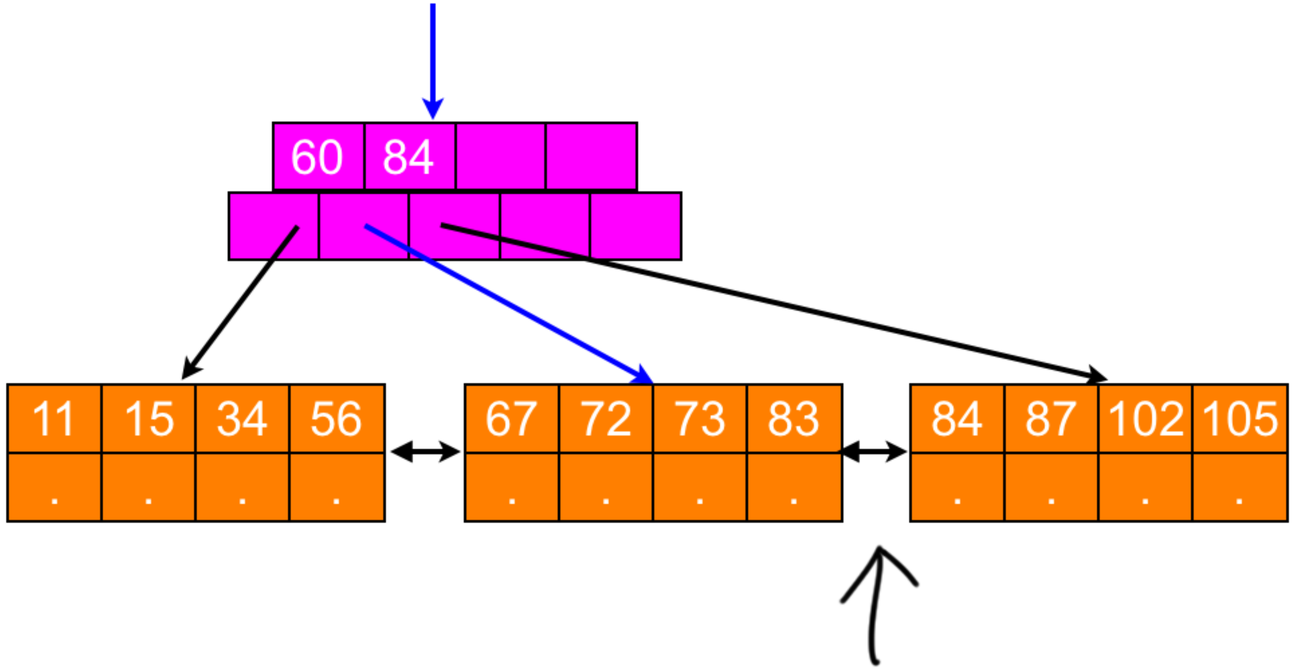
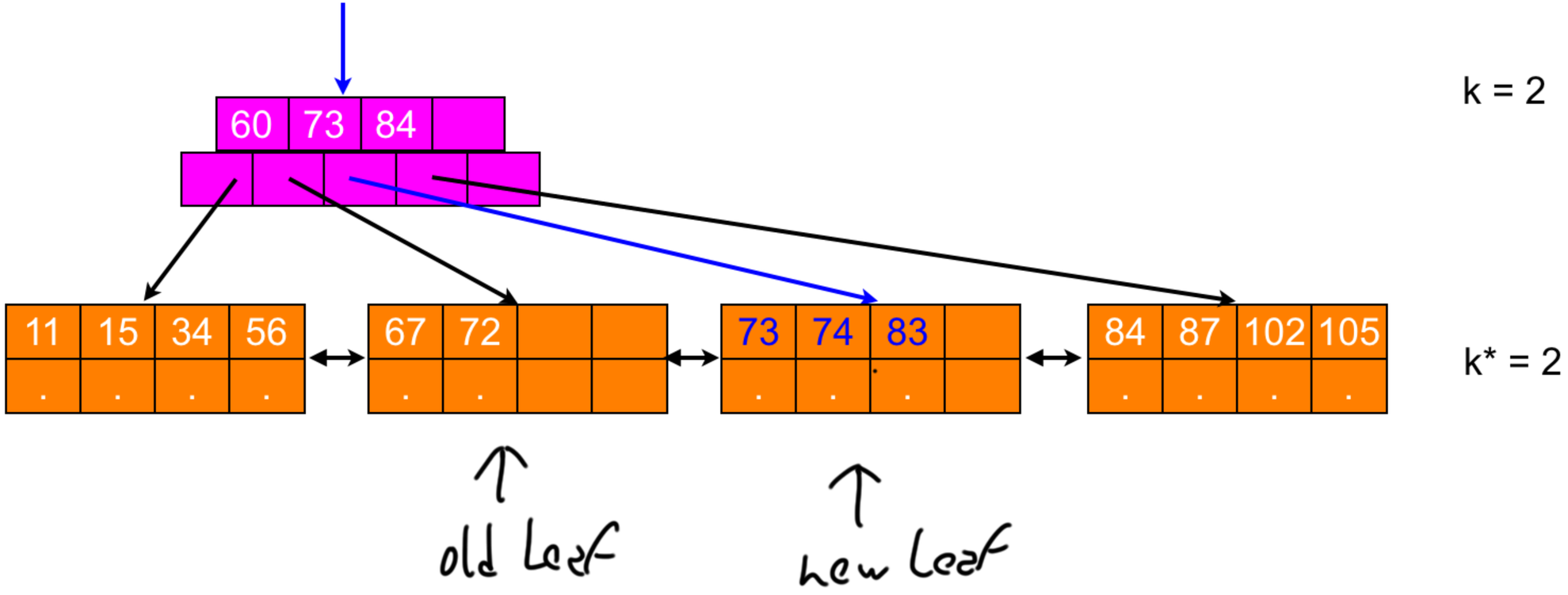
# insert(74)




k = 2

k* = 2

# insert(74) -> Leaf Split

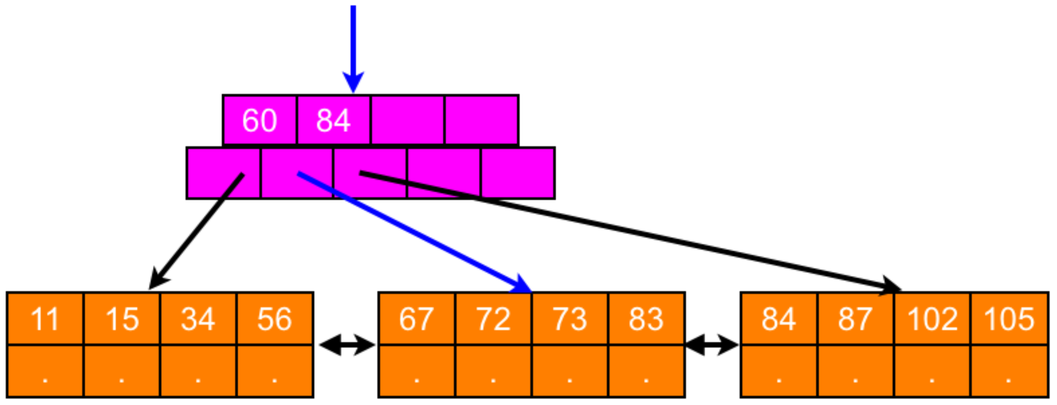



k = 2

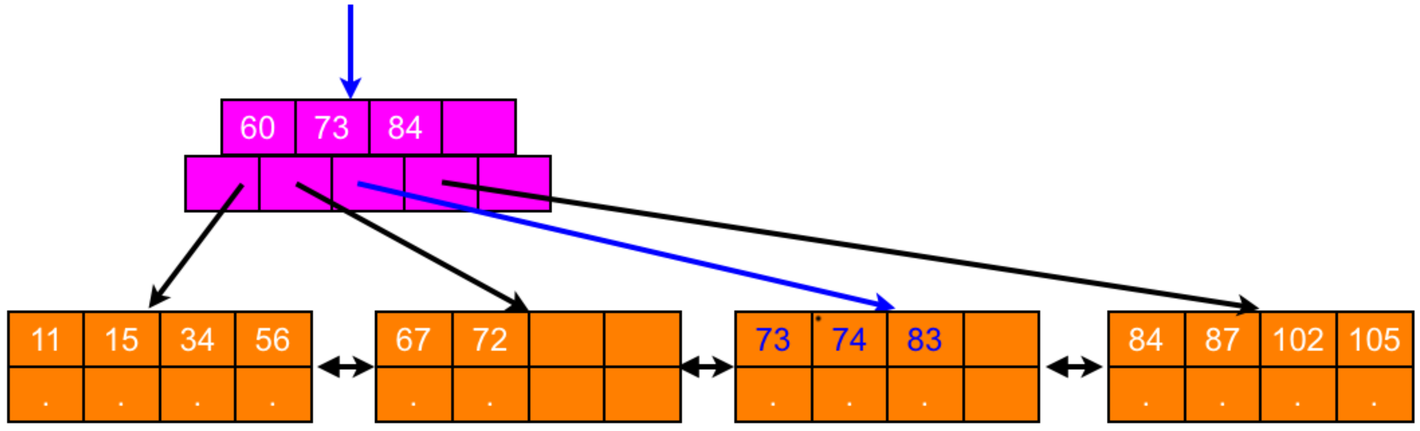
k* = 2

# Summary: insert(74) -> Leaf Split

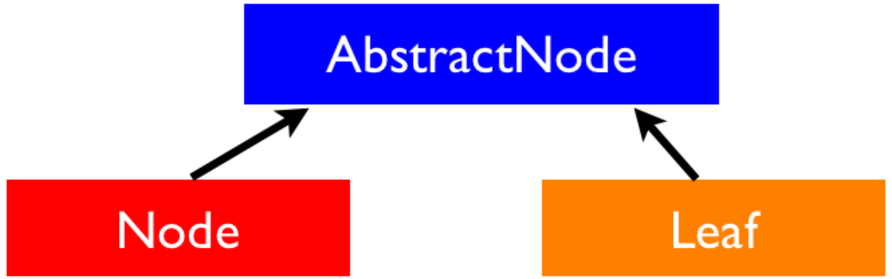
before:




after:



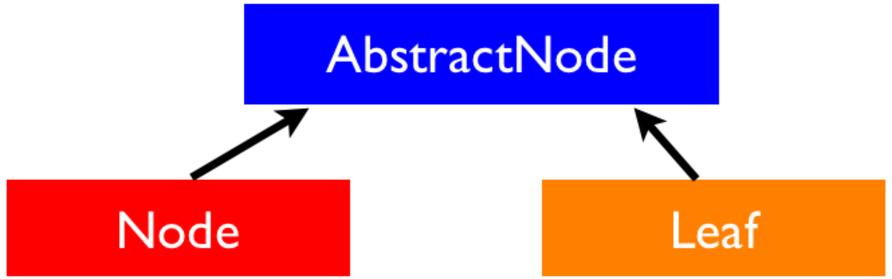
# BTree.insert(key, value)




**Node.insert(key, value):**

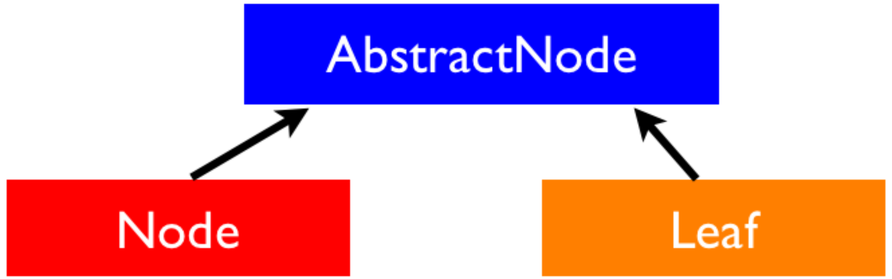
```
AbstractNode subtree = this.choose_subtree(key);                    //find subtree to follow
(newChildNode, newChildPivot) = subtree.insert(key, value);         //route insert-operation to subtree
```

# BTree.insert(key, value)



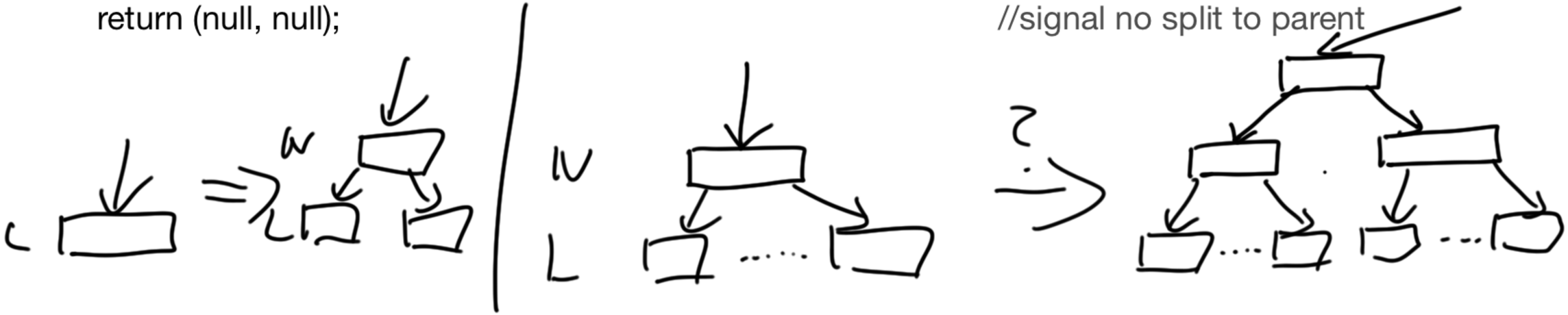

```
Node.insert(key, value):
    AbstractNode subtree = this.choose_subtree(key);                    //find subtree to follow
    (newChildNode, newChildPivot) = subtree.insert(key, value);         //route insert-operation to subtree
    If newChildNode != null:                                            //did a split occur in the child?
        If this.keys ≥ 2k:                                              //check overflow condition of this Node
            (newNode, newNodePivot) = this.split();                     //split this Node
            If newChildPivot < newNodePivot:                            //check into which Node to insert
                this.insertEntry(newChildNode, newChildPivot);          //insert into this (old) node
            Else:                                                       //i.e., newChildPivot ≥ newNodePivot
                newNode.insertEntry(newChildNode, newChildPivot);       //insert into newly created node
            return (newNode, newNodePivot);                             //return split information to parent
        Else:                                                           //no split necessary for this Node
            this.insertEntry(newChildNode, newChildPivot);              //insert new child into this node
            return (null, null);                                        //signal no split to parent
    Else:                                                               //i.e., no split in child
        return (null, null);                                            //signal no split to parent
```

# BTree.insert(key, value)



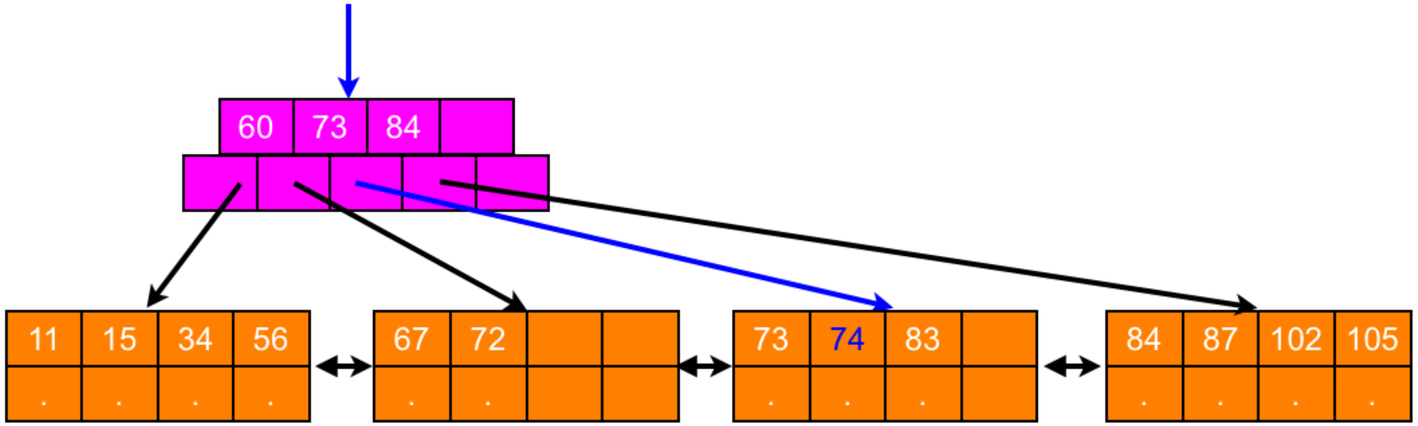

**Leaf.insert (key, value):**

```
If this.keys ≥ 2k*:                                  //check overflow condition
    (newLeaf, newLeafPivot) = split();               //split this Leaf
    If key < newLeafPivot:                           //check into which Leaf to insert
        this.insertEntry(key, value);                //insert into this (old) Leaf
    Else:                                            //i.e., key ≥ newLeafPivot
        newLeaf.insertEntry(key, value);             //insert into newly created Leaf
    return (newLeaf, newLeafPivot);                  //return split information to parent
Else:                                                //no split necessary for this Leaf
    this.insert(key, value);                         //insert into newly created Leaf
    return (null, null);                             //signal no split to parent
```

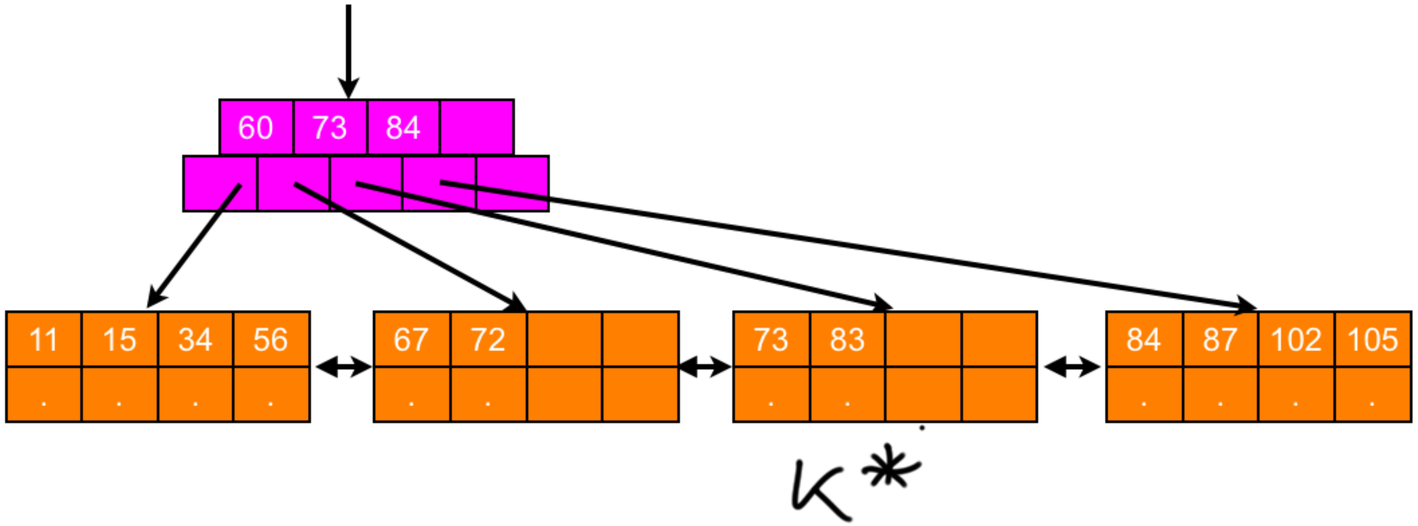


# delete(74)

before:

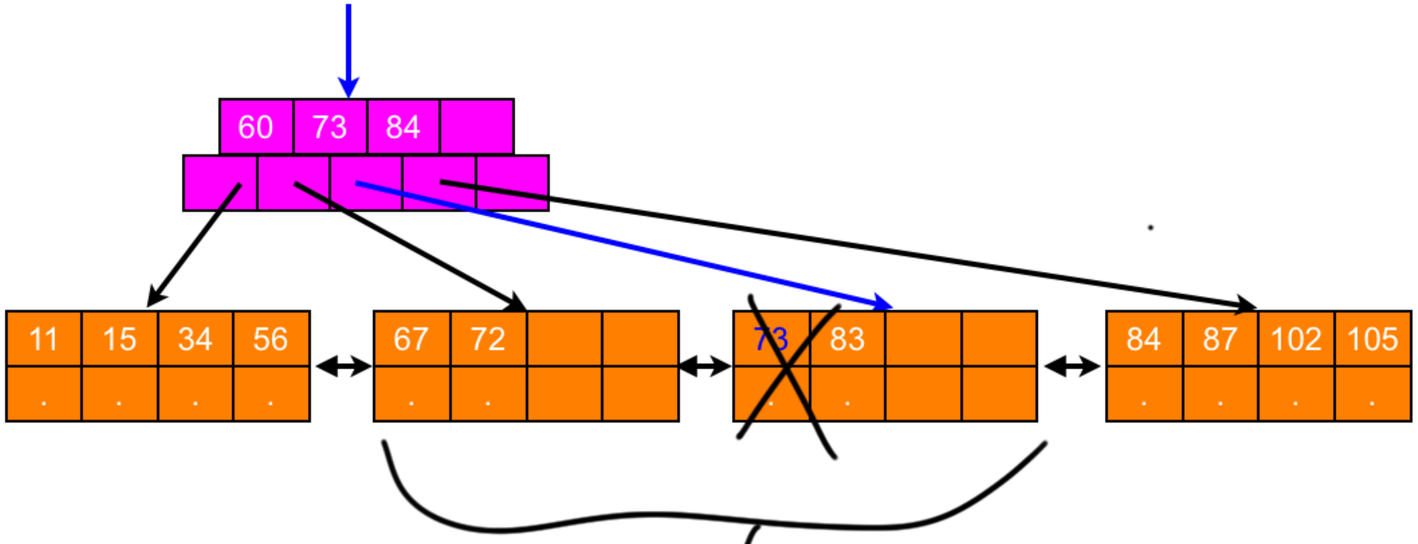



after:

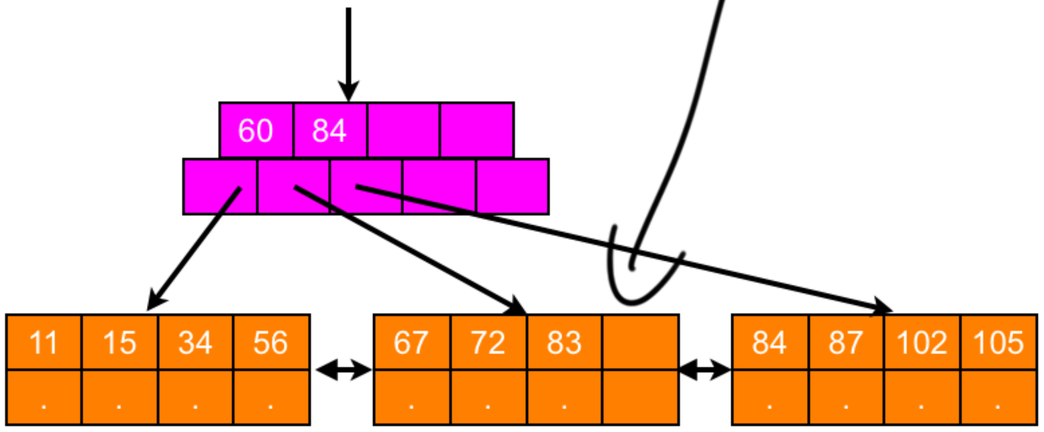


# delete(73) -> merge()

before:




after:




$3 \in [2;4]$

# delete() and merge()

delete():

=inverse of insert()

first: find_key()

then: delete entry from leave

merge():

=inverse of split()

first: merge two nodes into one

then: remove one pivot from parent