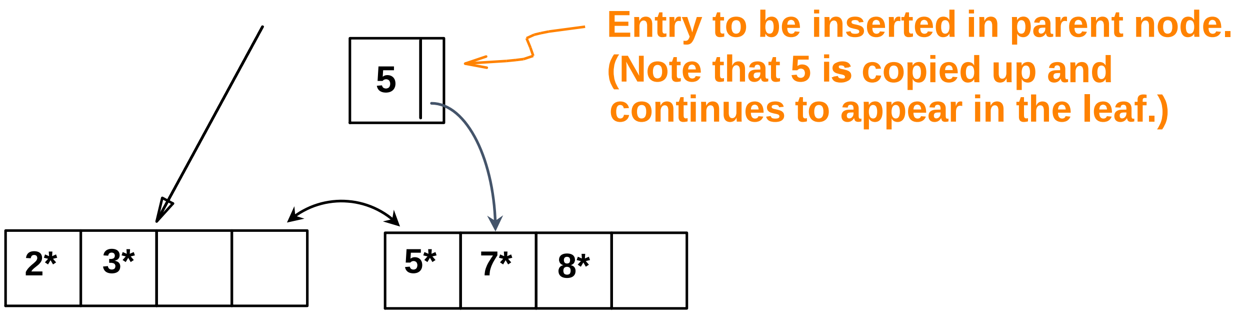
B-tree examples

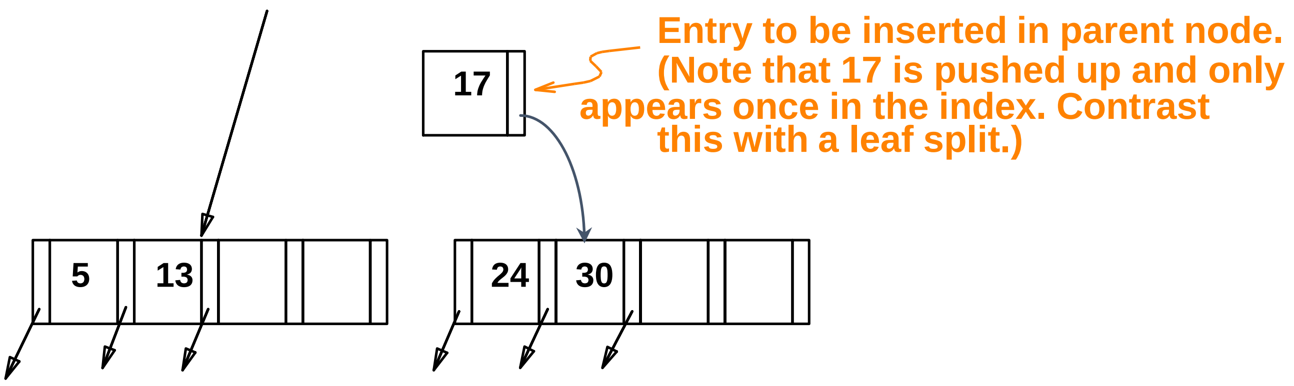
Inserting a Data Entry into a B+ Tree

- Find correct leaf *L.*
- Put data entry onto *L*.
  - If *L* has enough space, *done*!
  - Else, must *split* *L (into L and a new node L2)*
    - Redistribute entries evenly, **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To split index node, redistribute entries evenly, but **push up** middle key. (Contrast with leaf splits.)
- Splits "grow" tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top.*

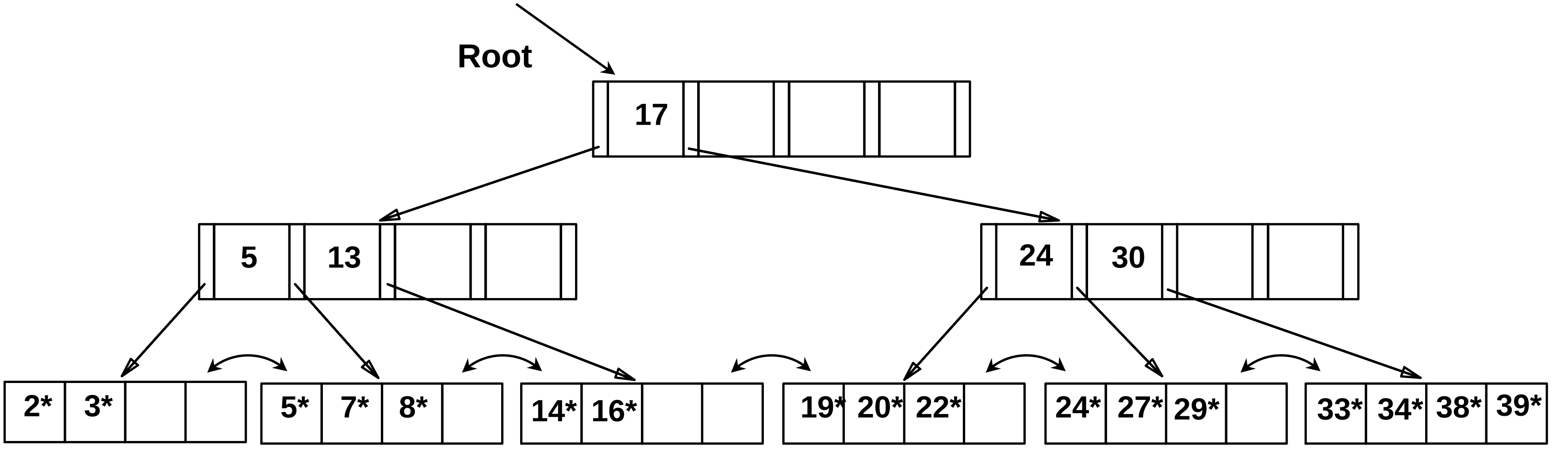
Inserting 8* into Example B+ Tree

- Observe how minimum occupancy is guaranteed in both leaf and index pg splits.
- Note difference between *copy-up* and *push-up;* be sure you understand the reasons for this.





Example B+ Tree After Inserting 8*



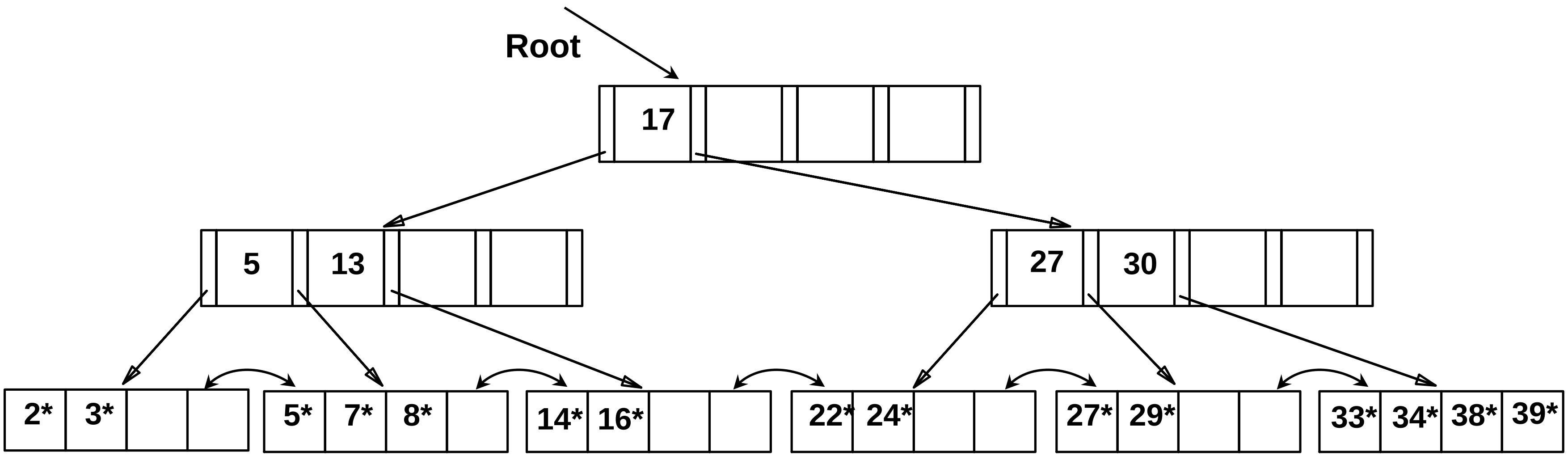
❖ Notice that root was split, leading to increase in height.

❖ In this example, we can avoid split by re-distributing entries; however, this is usually not done in practice.

Deleting a Data Entry from a B+ Tree

- Start at root, find leaf *L* where entry belongs.
- Remove the entry.
  - If L is at least half-full, *done!*
  - If L has only **d-1** entries,
    - Try to re-distribute, borrowing from ***sibling*** *(adjacent node with same parent as L)*.
    - If re-distribution fails, *merge* *L* and sibling.
- If merge occurred, must delete entry (pointing to *L* or sibling) from parent of *L*.
- Merge could propagate to root, decreasing height.

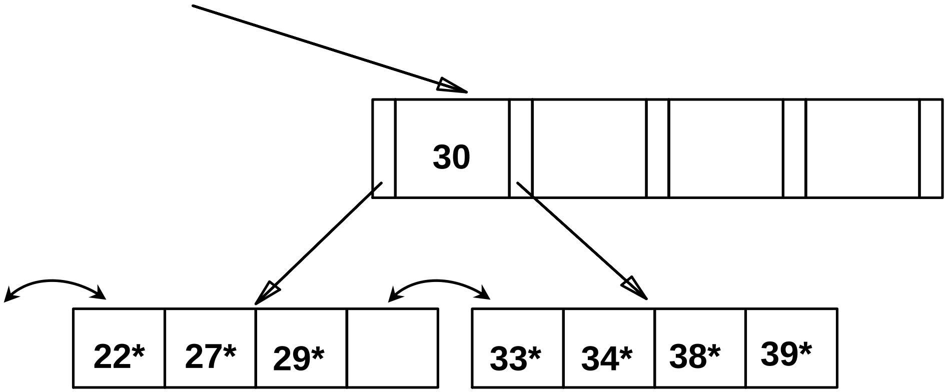
Example Tree After (Inserting 8*, Then) Deleting 19* and 20* ...

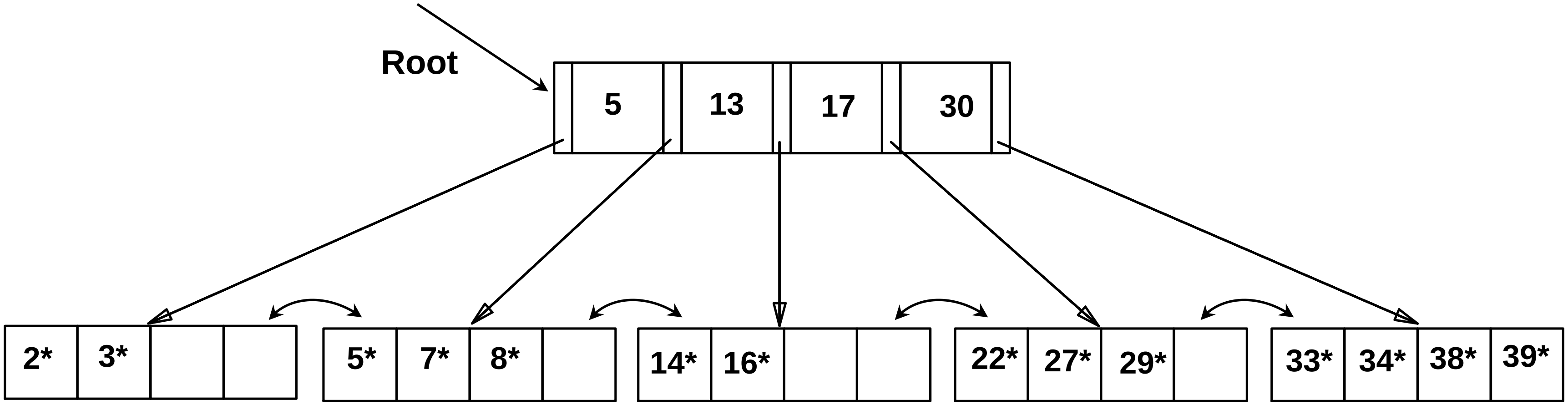


- Deleting 19* is easy.
- Deleting 20* is done with re-distribution. Notice how middle key is *copied up*.

... And Then Deleting 24*

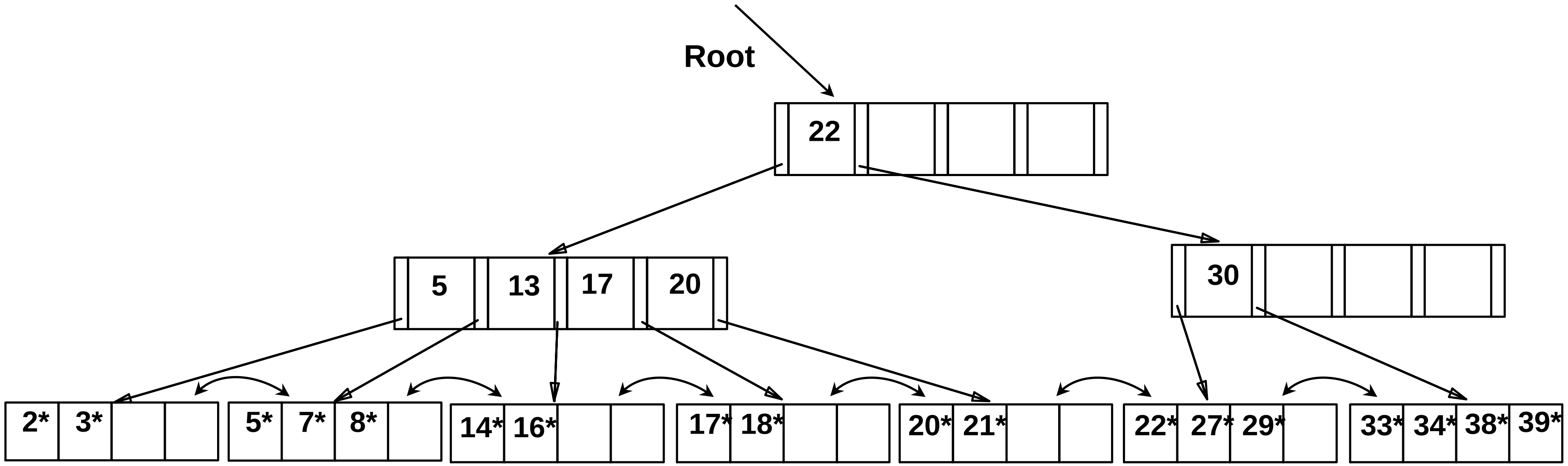
- Must merge.
- Observe *`toss'* of index entry (on right), and *`pull down'* of index entry (below).





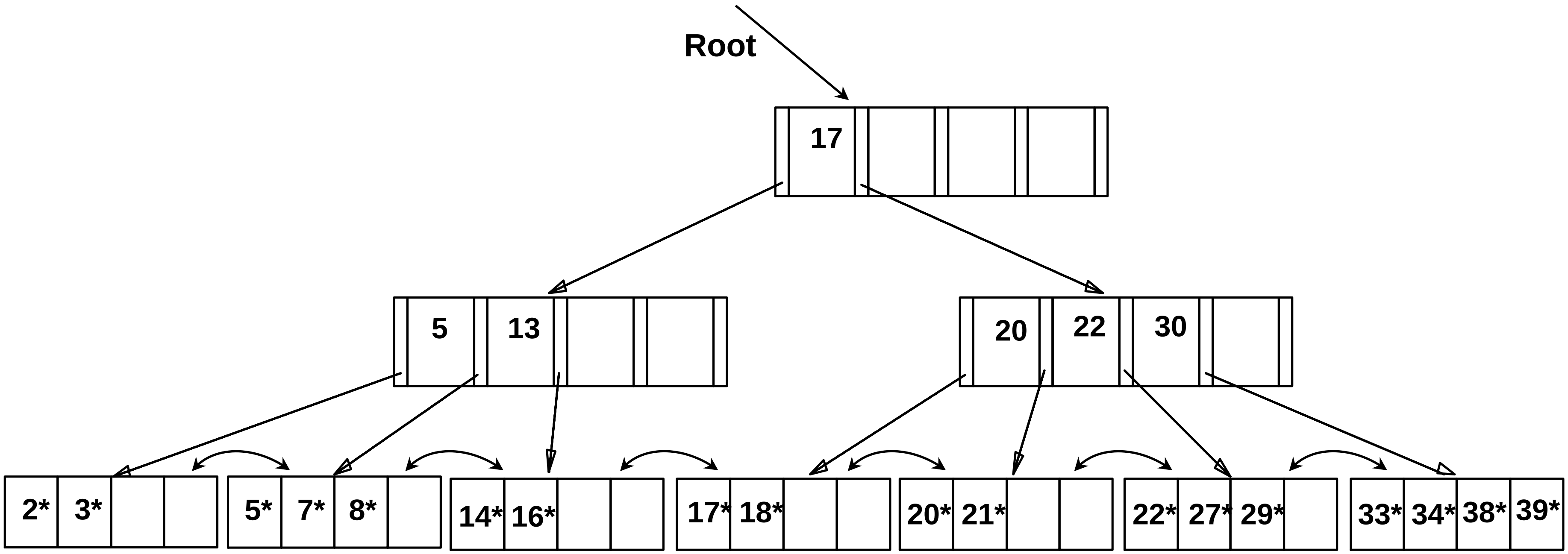
Example of Non-leaf Re-distribution

- Tree is shown below *during deletion* of 24*. (What could be a possible initial tree?)
- In contrast to previous example, can re-distribute entry from left child of root to right child.



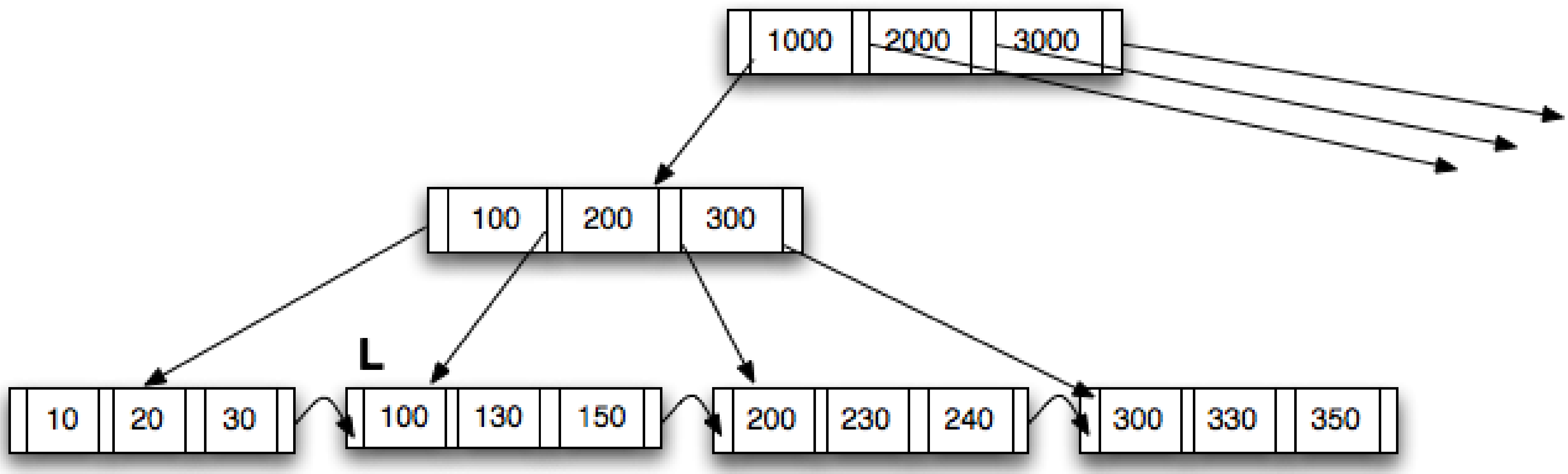
After Re-distribution

- Intuitively, entries are re-distributed by *`pushing through'* the splitting entry in the parent node.
- It suffices to re-distribute index entry with key 20; we've re-distributed 17 as well for illustration.



Another B+Tree Insertion Example

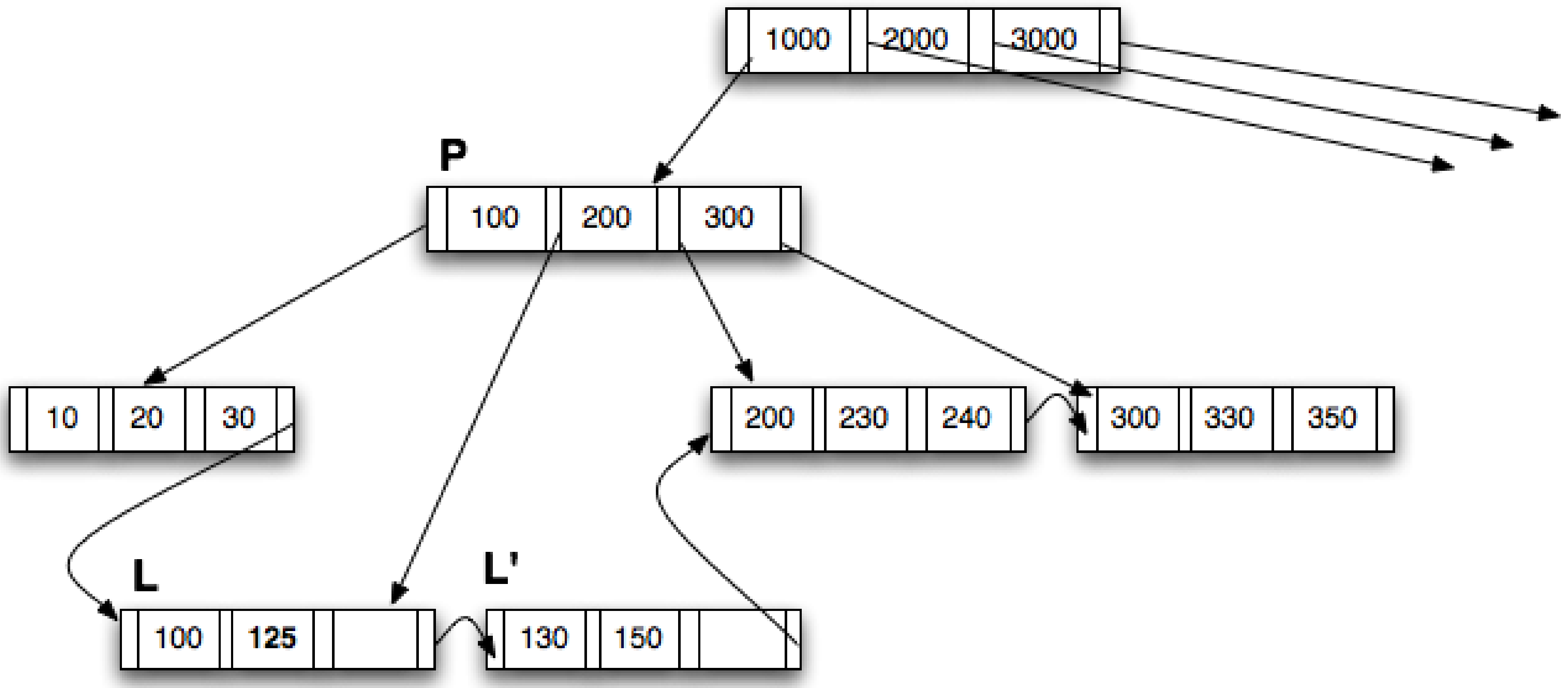
INITIAL TREE



Next slides show the insertion of (125) into this tree

Another Example: INSERT (125)

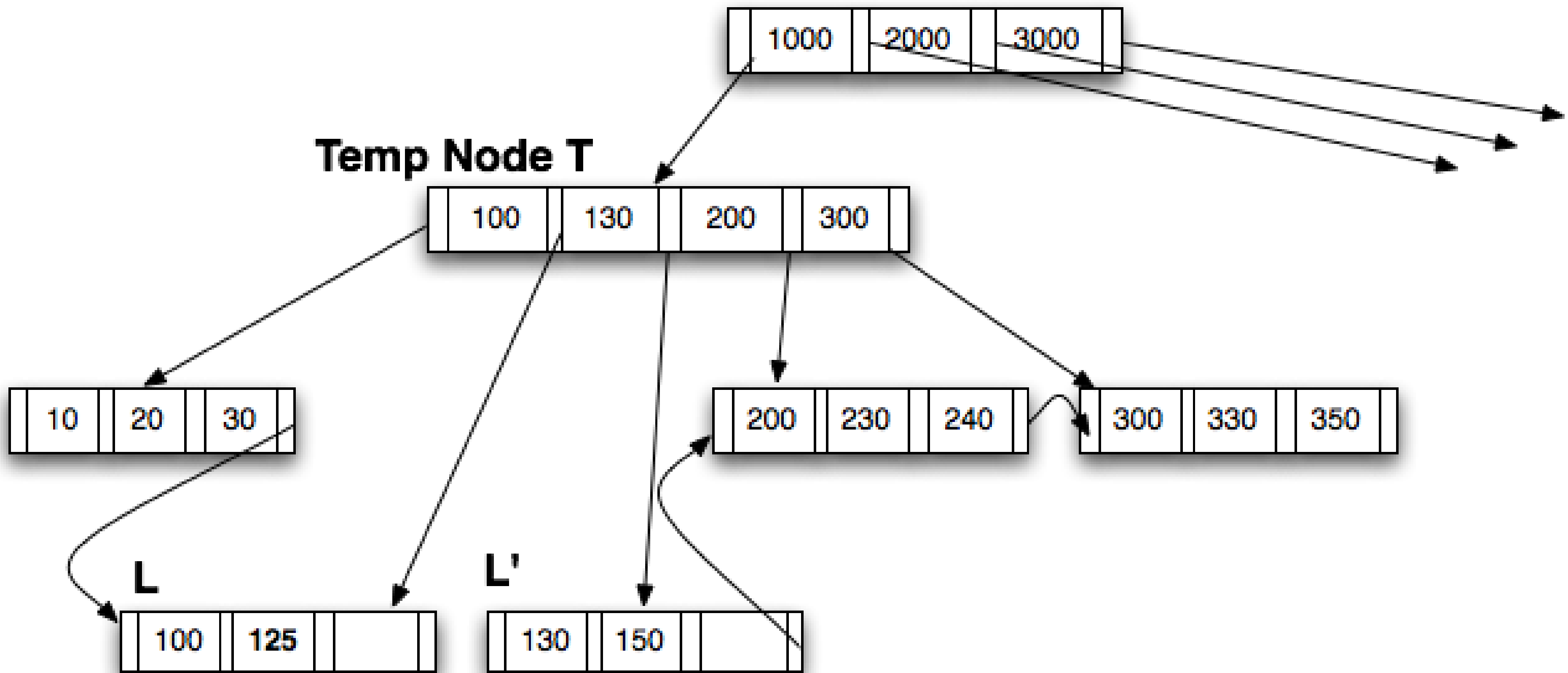
Step 1: Split L to create L'



Insert the lowest value in L' (130) upward into the parent P

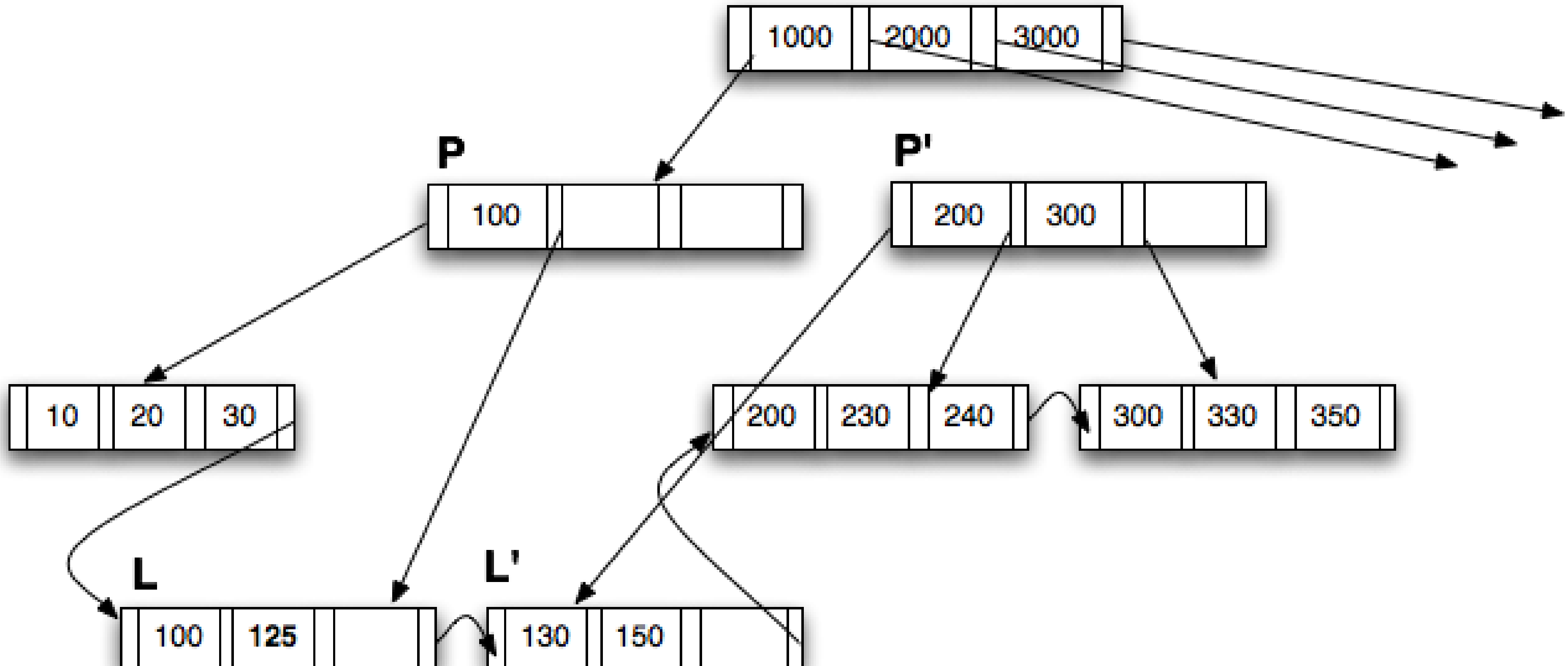
Another Example: INSERT (125)

Step 2: Insert (130) into P by creating a temp node T



Another Example: INSERT (125)

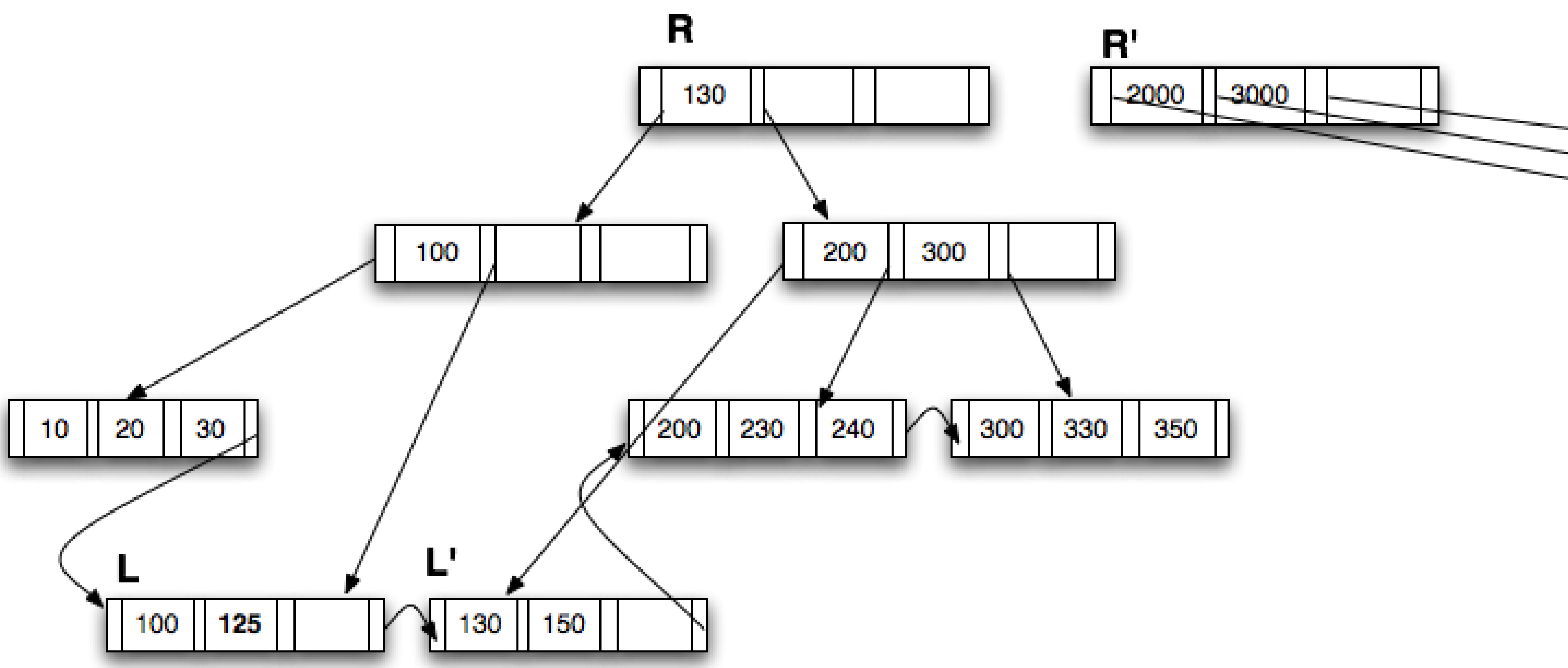
Step 3: Create P'; distribute from T into P and P'



New P has only 1 key, but two pointers so it is OKAY.
This follows the last 4 lines of Figure 12.13 (note that "n" = 4)
K'' = 130. Insert upward into the root

Another Example: INSERT (125)

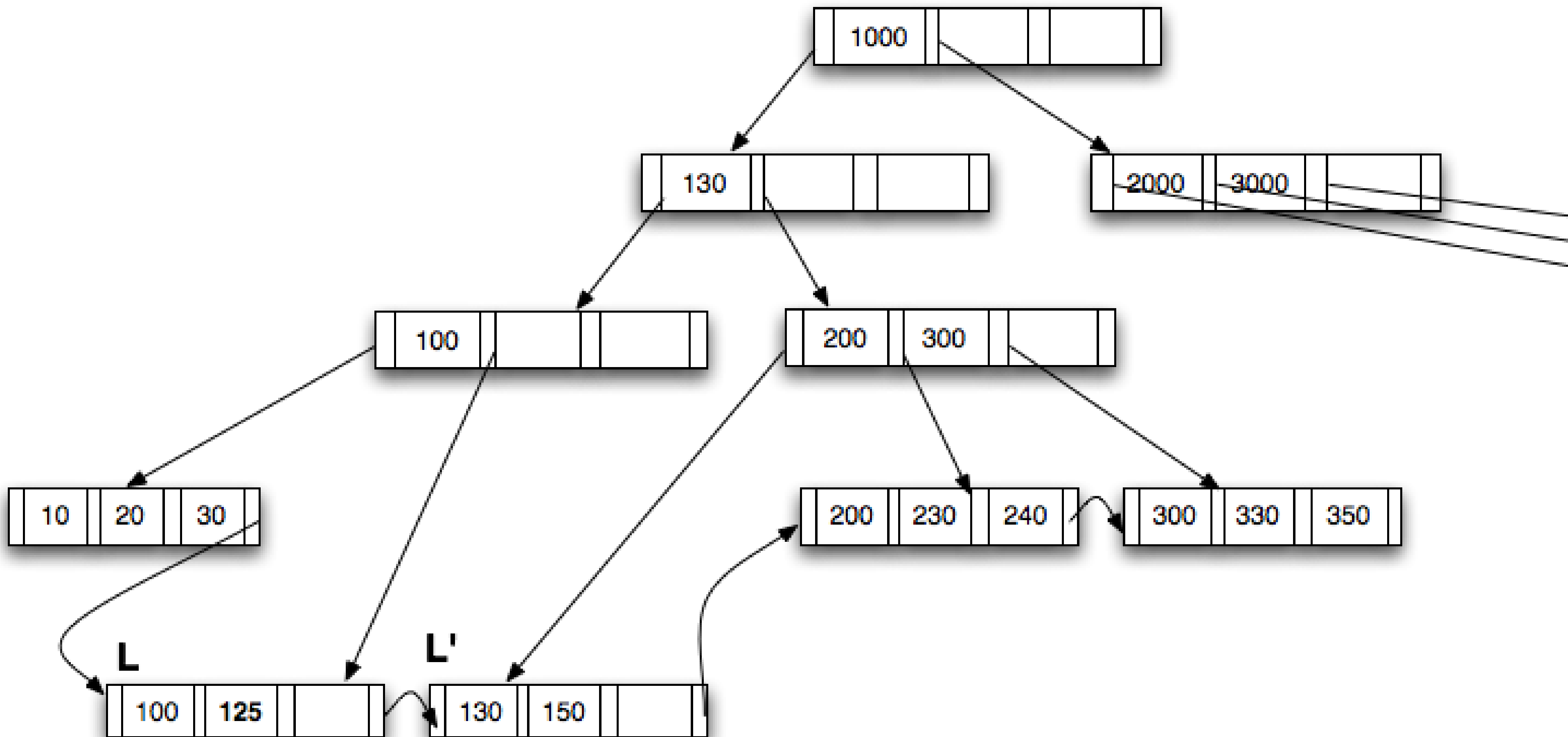
Step 4: Insert (130) into the parent (R); create R'



Once again following the insert_in_parent() procedure, K'' = 1000

Another Example: INSERT (125)

Step 5: Create a new root



From the book

- Exercises 10.1, 10.5, 10.11