(2,4) Trees

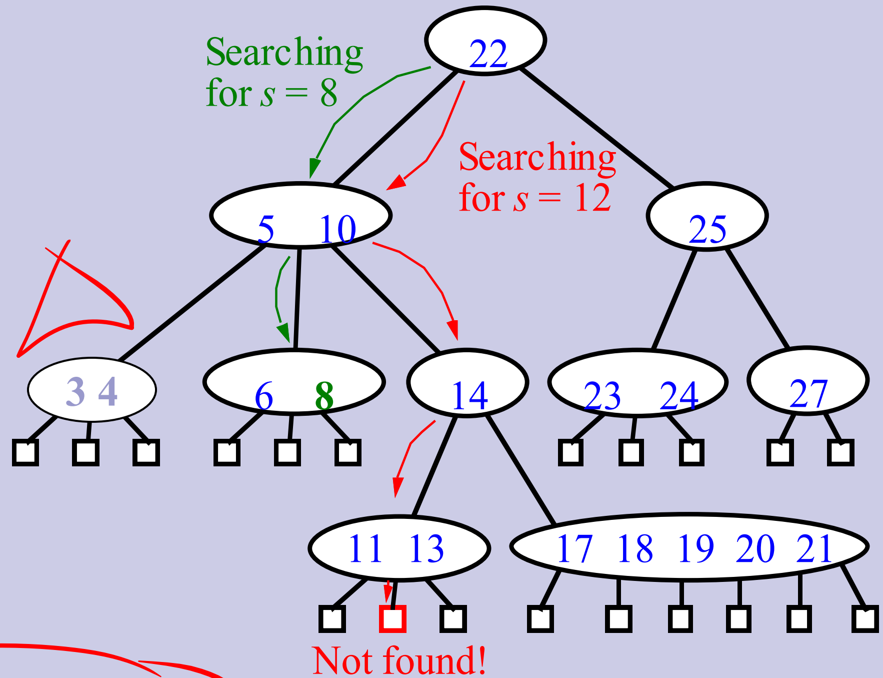
- What are they?
  - They are search Trees (but not binary search trees)
  - They are also known as 2-4, 2-3-4 trees

Multi-way Search Trees

- Each internal node of a multi-way search tree T:
  - has at least two children
  - stores a collection of items of the form (k, x), where k is a key and x is an element
  - contains d - 1 items, where d is the number of children
  - Has pointers to d children
- Children of each internal node are "between" items
- all keys in the subtree rooted at the child fall between keys of those items.

Multi-way Searching

- Similar to binary searching
  - If search key $\mathbf{s<k_1}$ search the leftmost child
  - If $\mathbf{s>k_{d-1}}$, search the rightmost child
- That's it in a binary tree; what about if **d>2**?
  - Find two keys $\mathbf{k_{i-1}}$ and $\mathbf{k_i}$ between which s falls, and search the child $\mathbf{v_i}$.
- What would an in-order traversal look like?

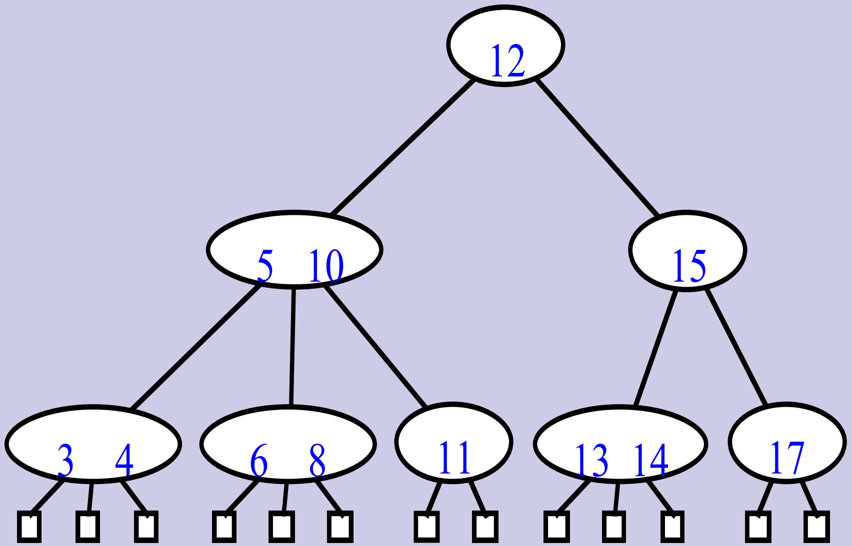


k_1 k_2 . . . k_{d-1}

v_1 v_2 v_3 v_d

(2,4) Trees

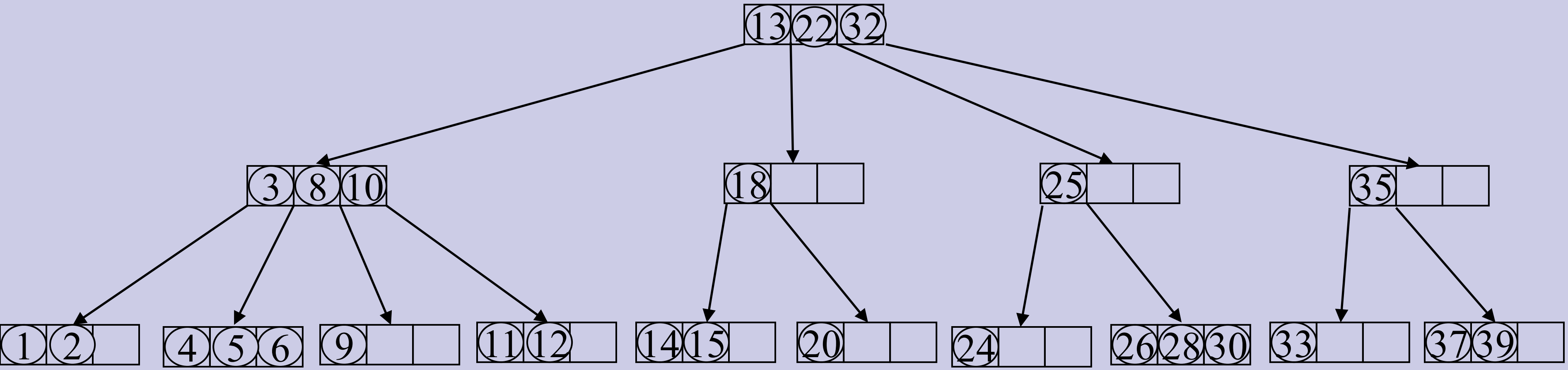
- Properties:
  - At most 4 children
  - All leaf nodes are at the same level.
  - Height *h* of (2,4) tree is at least $\log_4 n$ and atmost $\log_2 n$
- How is the last fact useful in searching?



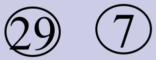
Insertion

21 23 40 29 7

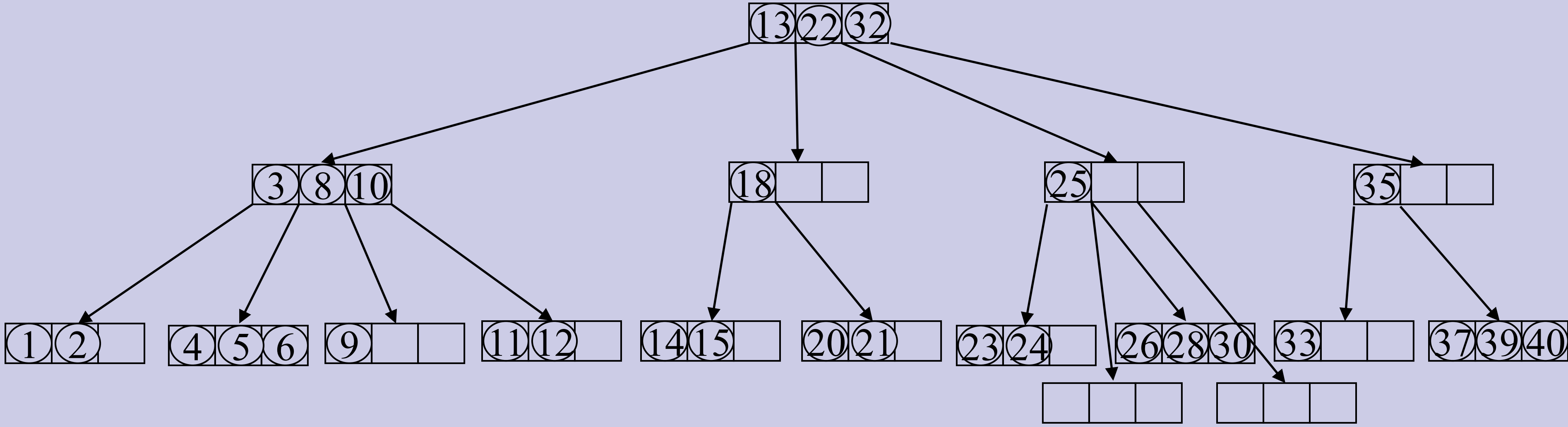
- No problem if the node has empty space



Insertion(2)

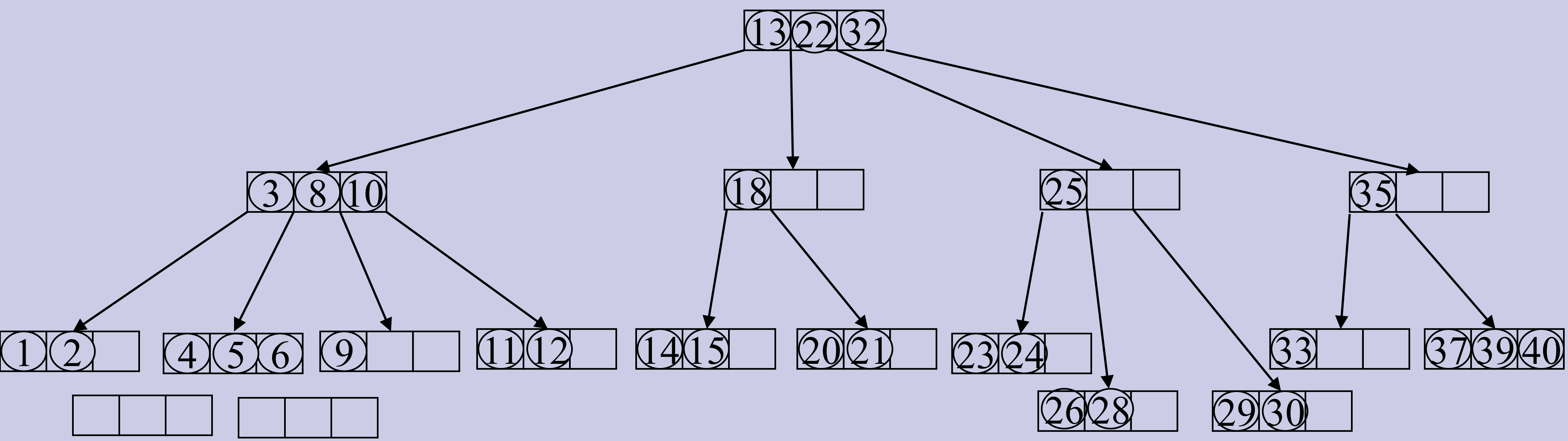


- Nodes get split if there is insufficient space.



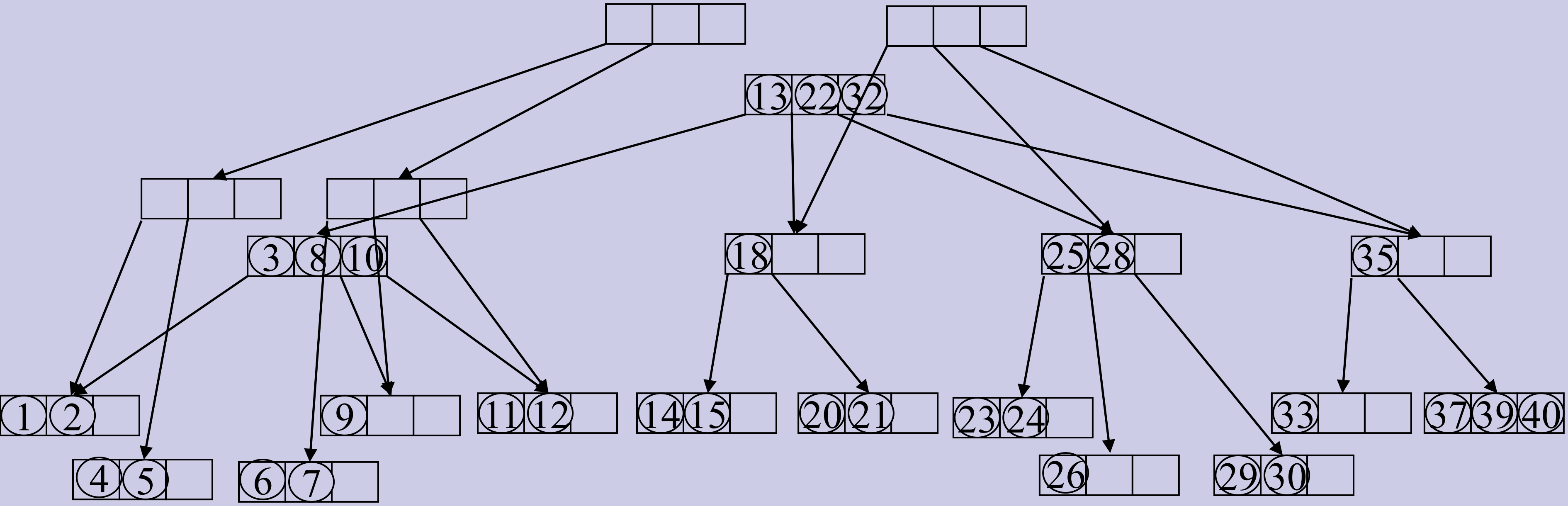
Insertion(3)

- One key is promoted to parent and inserted in there



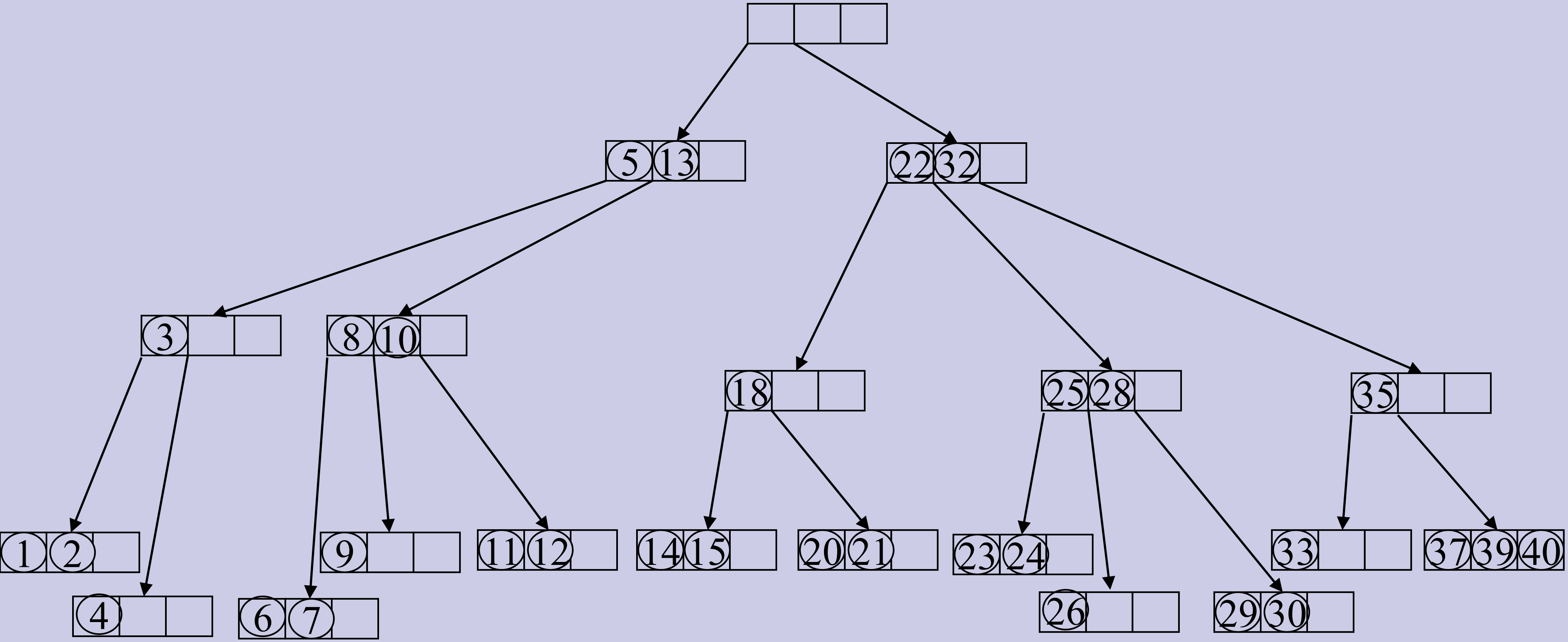
Insertion(4)

- If parent node does not have sufficient space then it is split.
- In this manner splits can cascade.



Insertion(5)

- Eventually we may have to create a new root.
- This increases the height of the tree

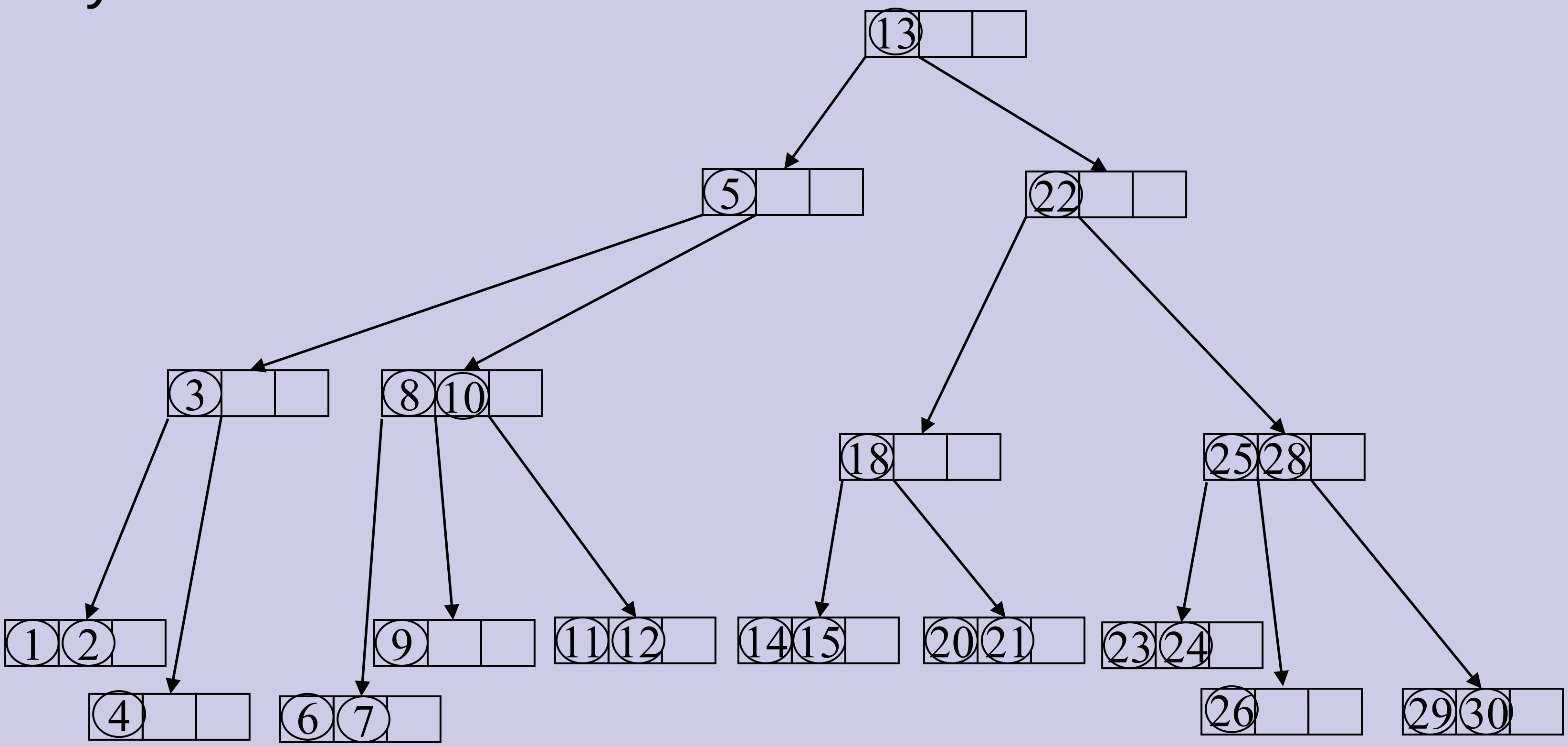


Time for Search and Insertion

- A search visits O(log N) nodes
- An insertion requires O(log N) node splits
- Each node split takes constant time
- Hence, operations Search and Insert each take time O(log N)

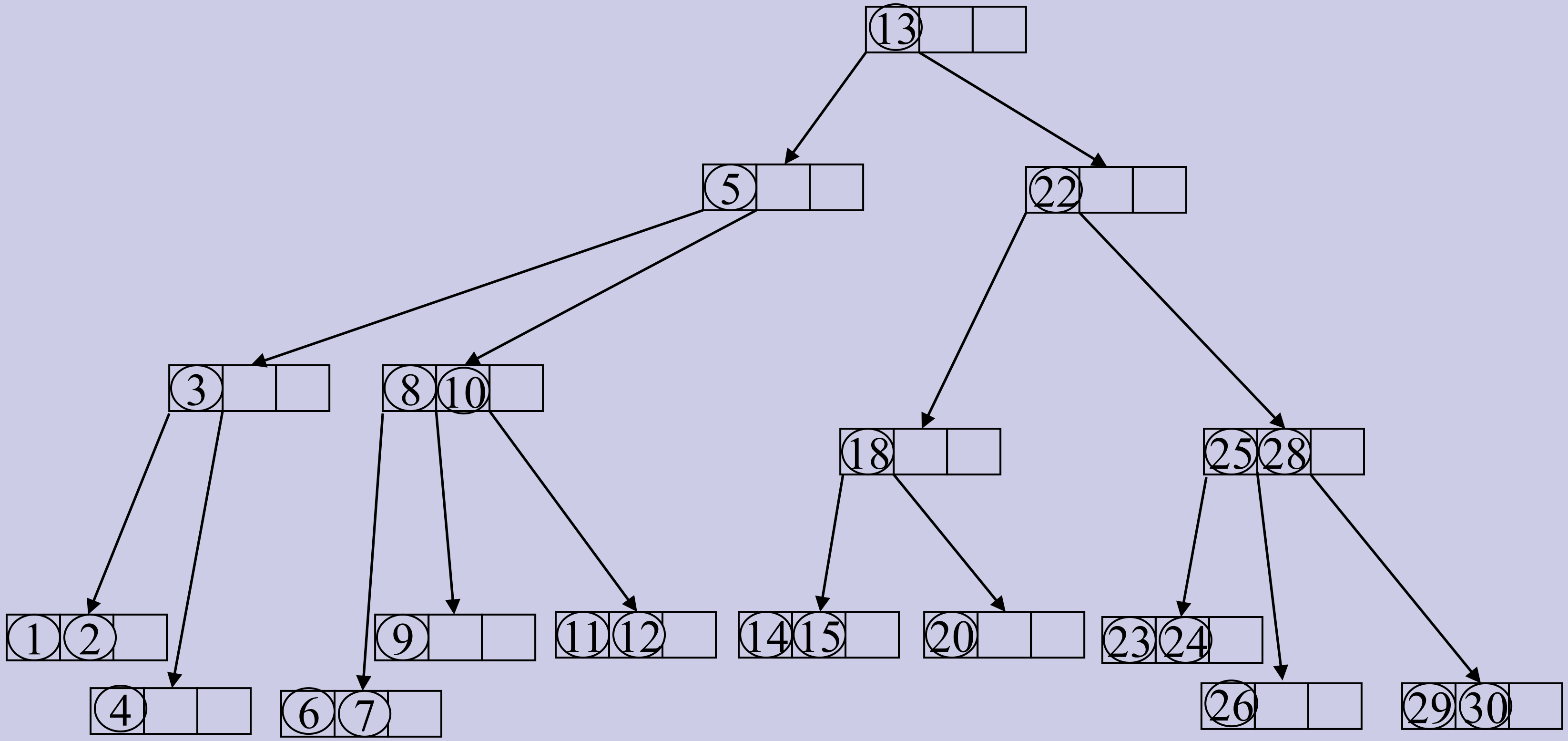
Deletion

- Delete 21.
- No problem if key to be deleted is in a leaf with at least 2 keys



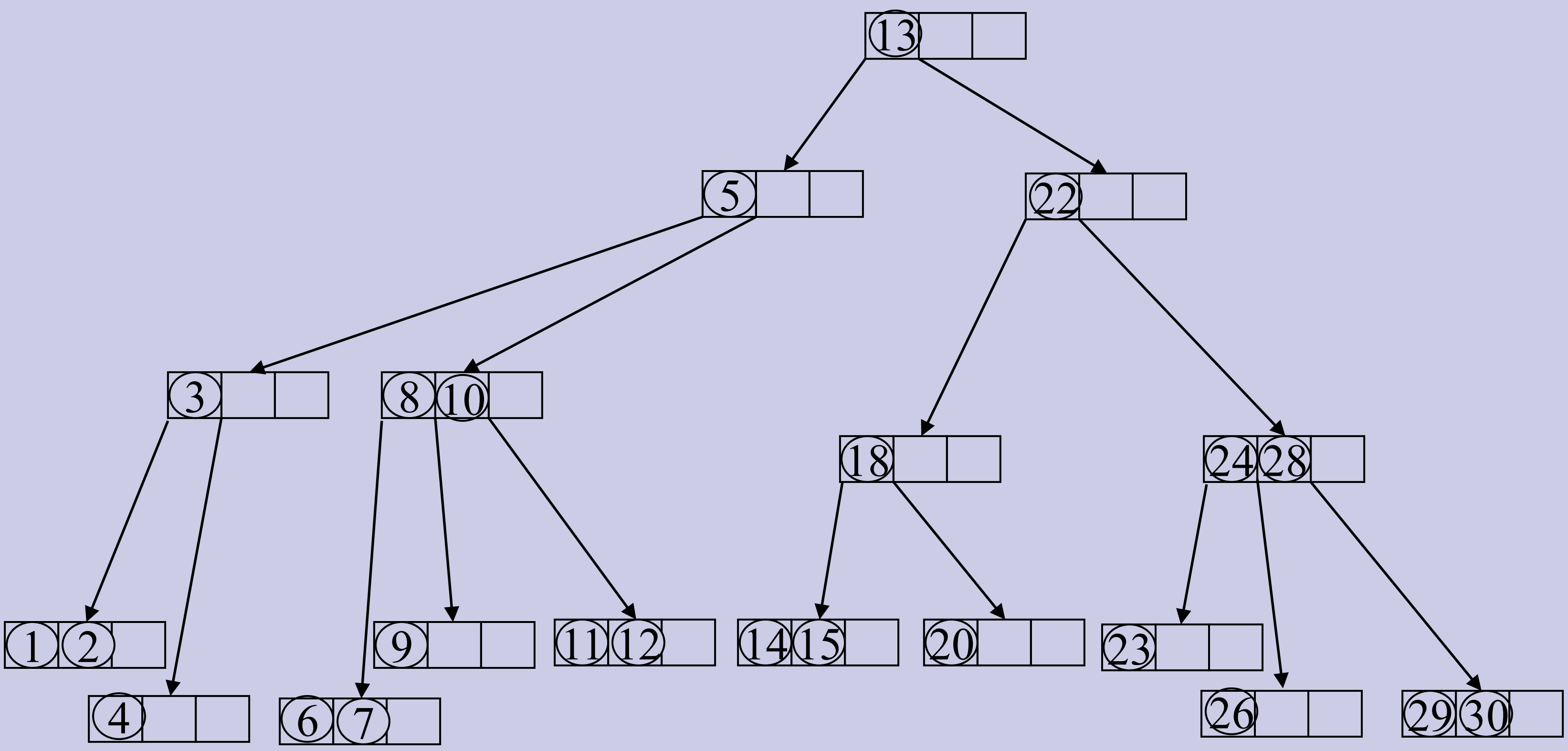
Deletion(2)

- If key to be deleted is in an internal node then we swap it with its predecessor (which is in a leaf) and then delete it.
- Delete 25



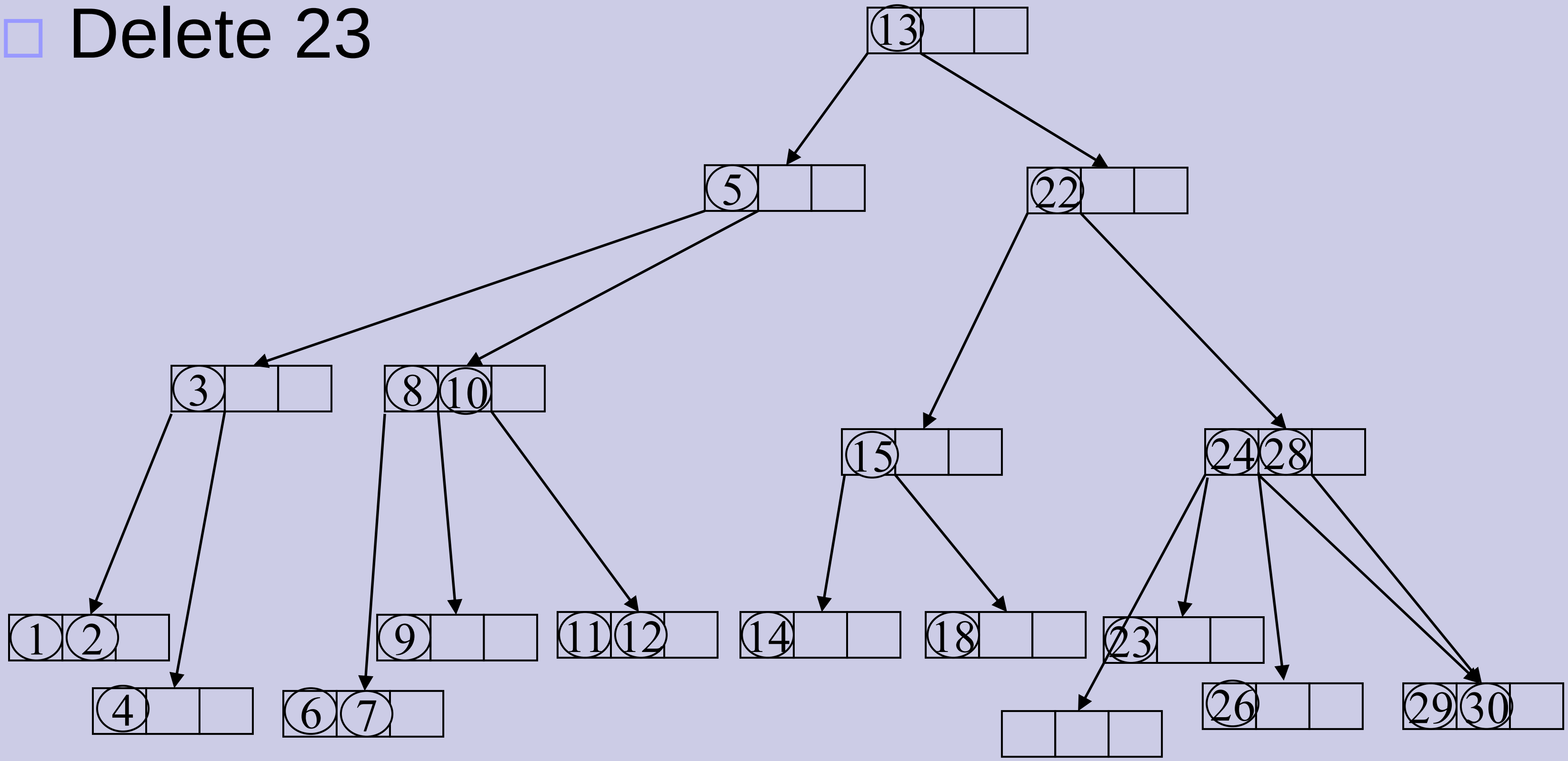
Deletion(3)

- If after deleting a key a node becomes empty then we borrow a key from its sibling.
- Delete 20



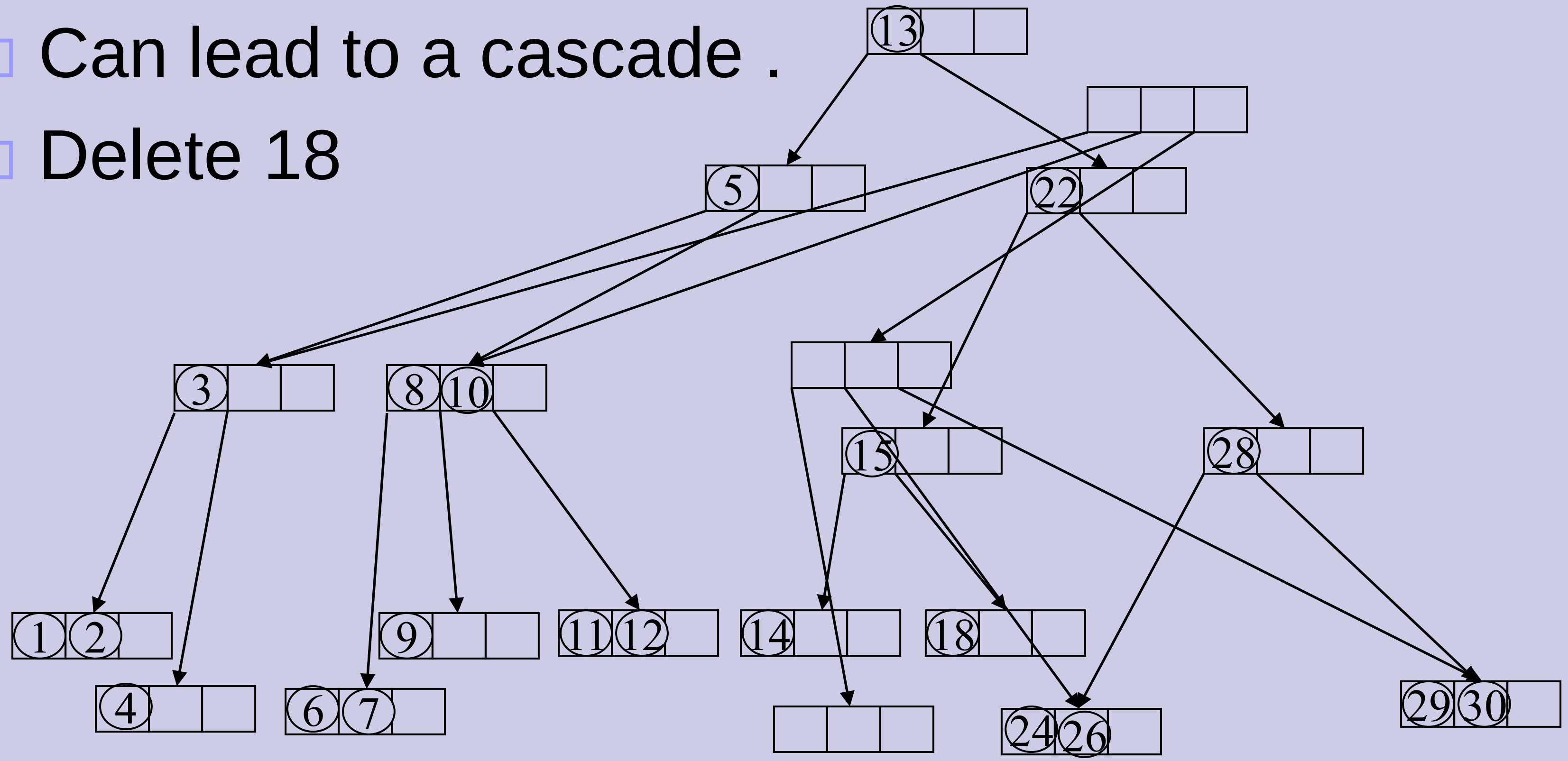
Deletion(4)

- If sibling has only one key then we merge with it.
- The key in the parent node separating these two siblings moves down into the merged node.
- Delete 23



Delete(5)

- Moving a key down from the parent corresponds to deletion in the parent node.
- The procedure is the same as for a leaf node.
- Can lead to a cascade .
- Delete 18



(2,4) Conclusion

- The height of a (2,4) tree is $O(\log n)$.
- Split, transfer, and merge each take $O(1)$.
- Search, insertion and deletion each take $O(\log n)$.
- Why are we doing this?
  - (2,4) trees are fun! Why else would we do it?
  - Well, there’s another reason, too.
  - They’re pretty fundamental to the idea of Red-Black trees as well.