Algorithms

Balanced Trees

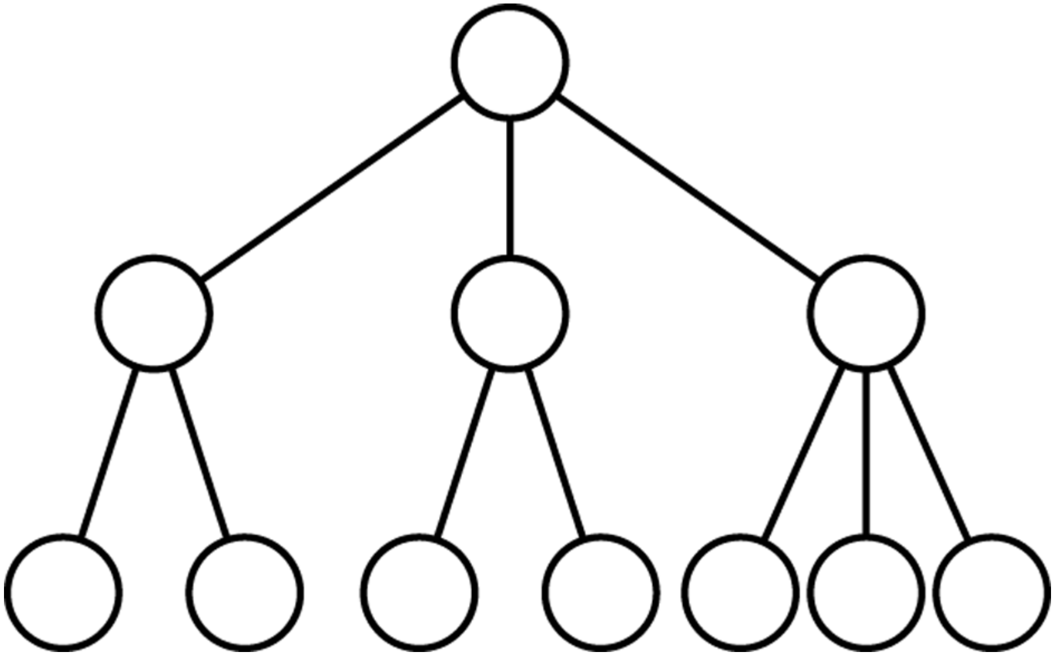
2 - 3 Tree

2 – 3 – 4 Tree

2-3 Trees

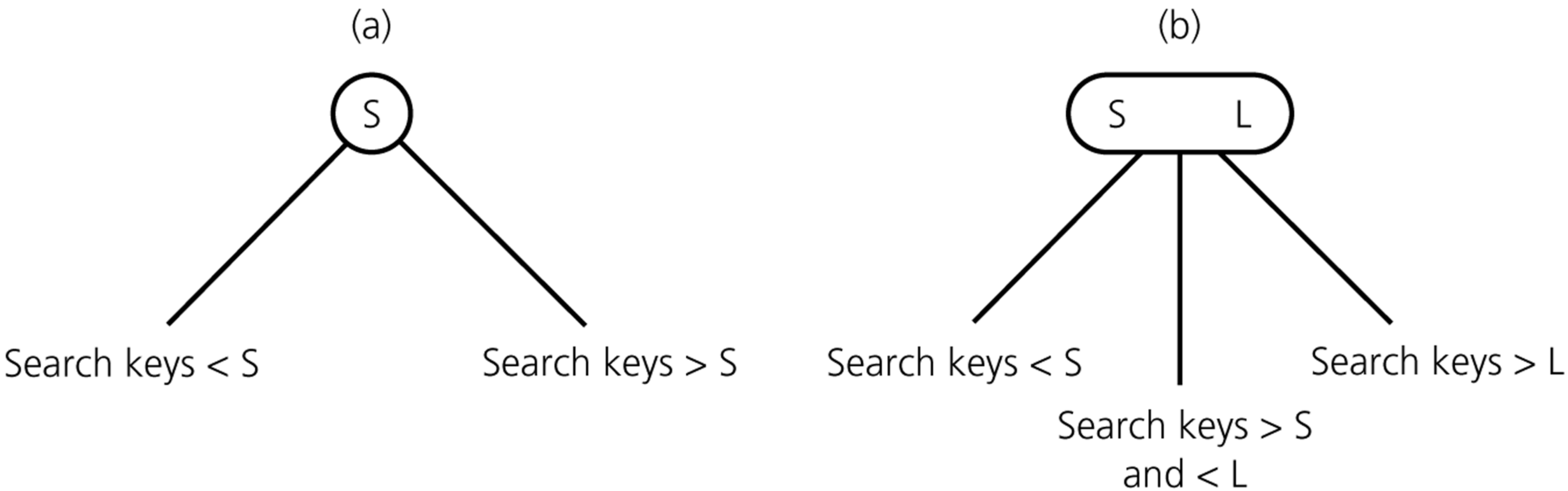
Features

- *each internal node has either 2 or 3 children*
- *all leaves are at the same level*



2-3 Trees with Ordered Nodes

2-node *3-node*



- *leaf node can be either a 2-node or a 3-node*

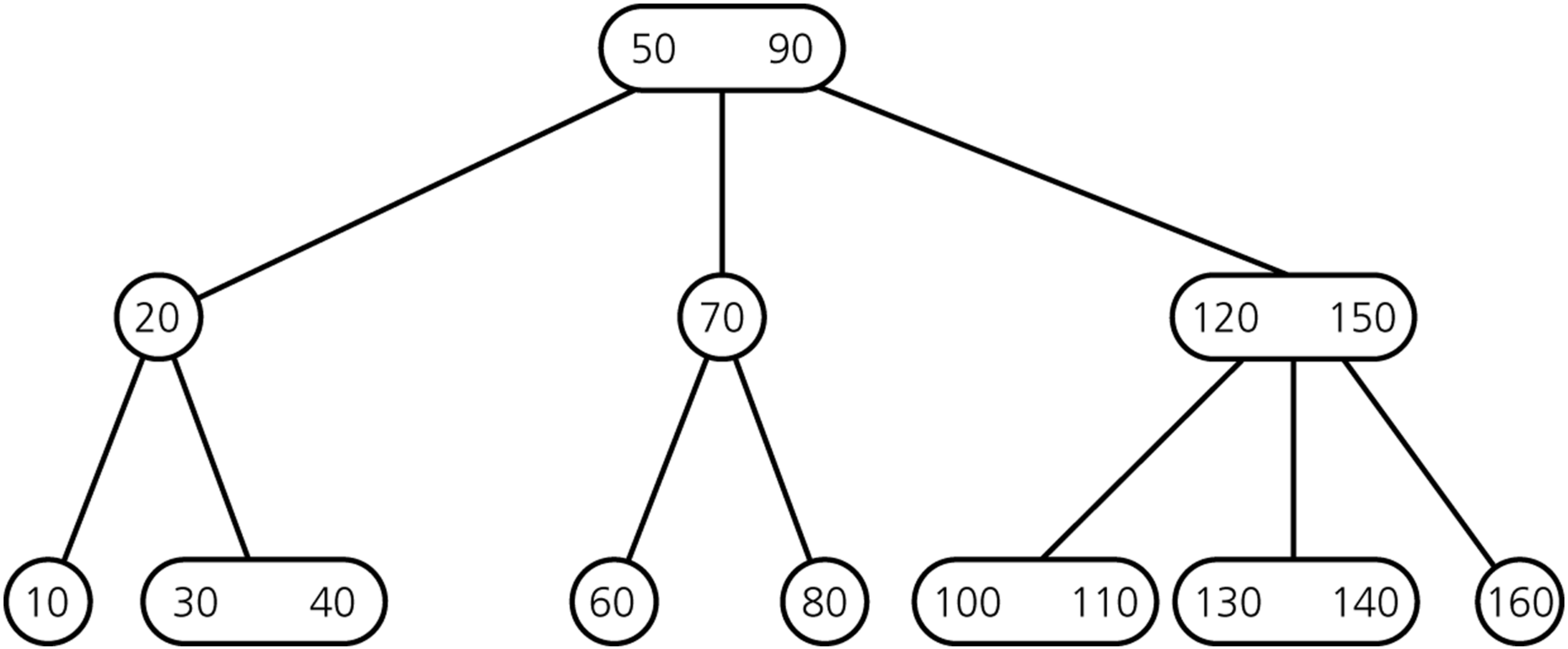
Why 2-3 tree

- Faster searching?
  - Actually, no. 2-3 tree is about as fast as an "equally balanced" binary tree, because you sometimes have to make 2 comparisons to get past a 3-node
- Easier to keep balanced?
  - Yes, definitely.
  - Insertion can split 3-nodes into 2-nodes, or promote 2-nodes to 3-nodes to keep tree approximately balanced!

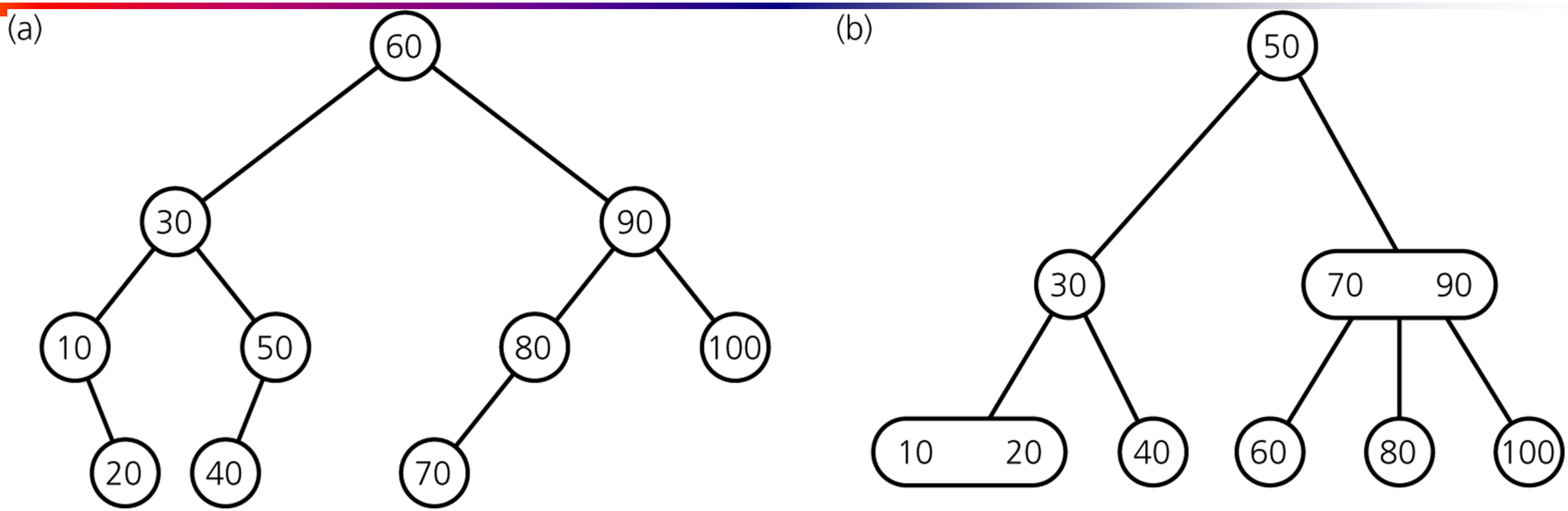
Why is this better?

- Intuitively, you unbalance a binary tree when you add height to one path significantly more than other possible paths.
- With the 2-3 insert algorithm, you can only add height to the tree when you create a new root, and this adds one unit of height to all paths simultaneously.
- Hence, the average path length of the tree stays close to log N.

Example of 2-3 Tree



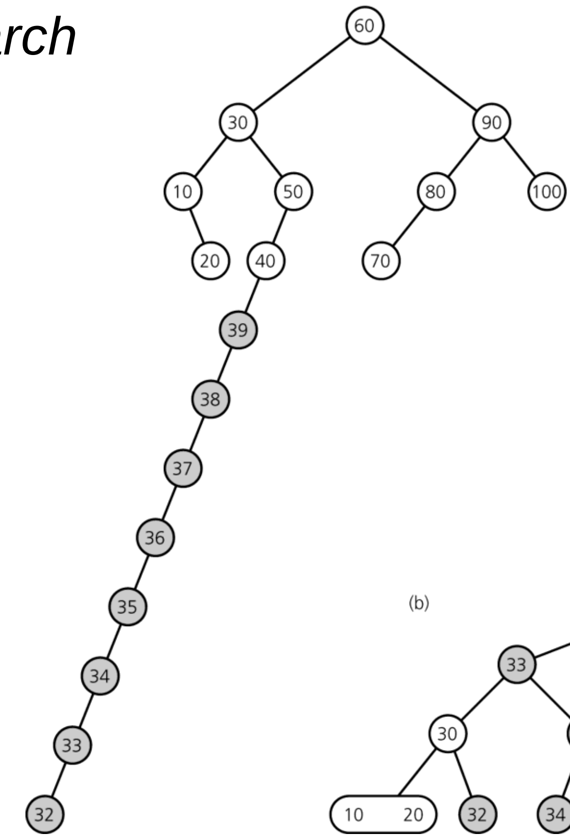
What did we gain?



What is the time efficiency of searching for an item?

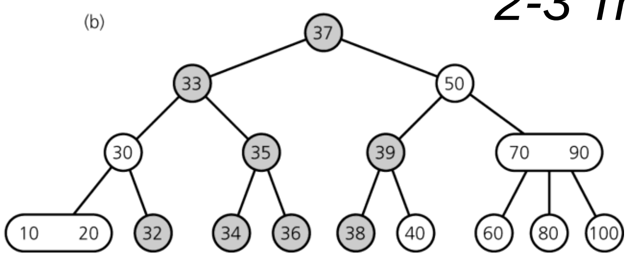
Gain: Ease of Keeping the Tree Balanced

Binary Search Tree



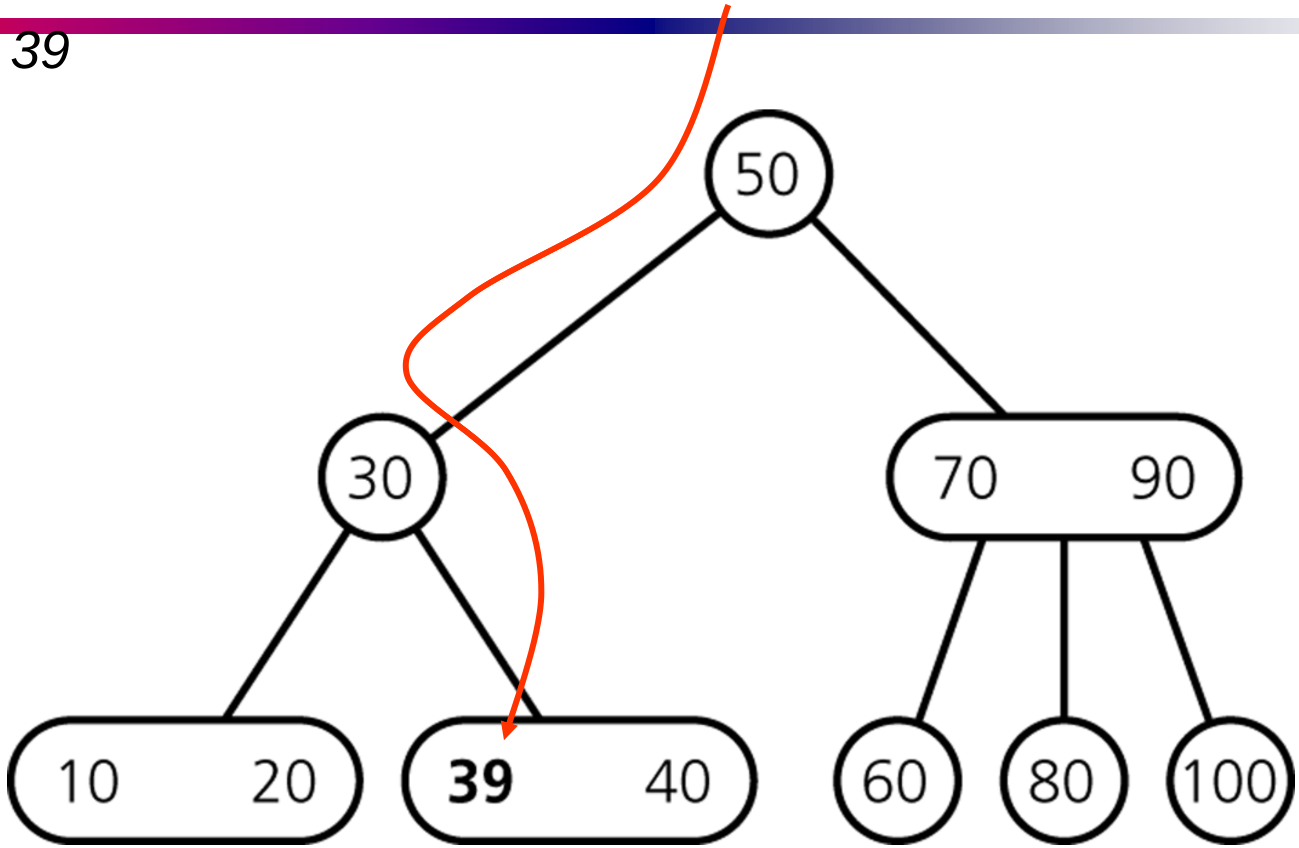
both trees after inserting items 39, 38, ... 32

2-3 Tree



Inserting Items

Insert 39



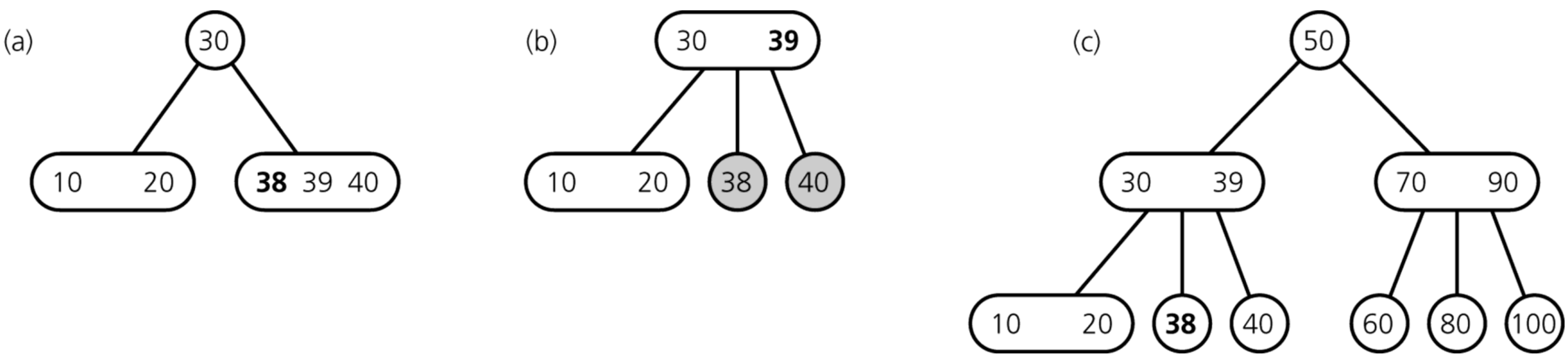
Inserting Items

Insert 38

insert in leaf

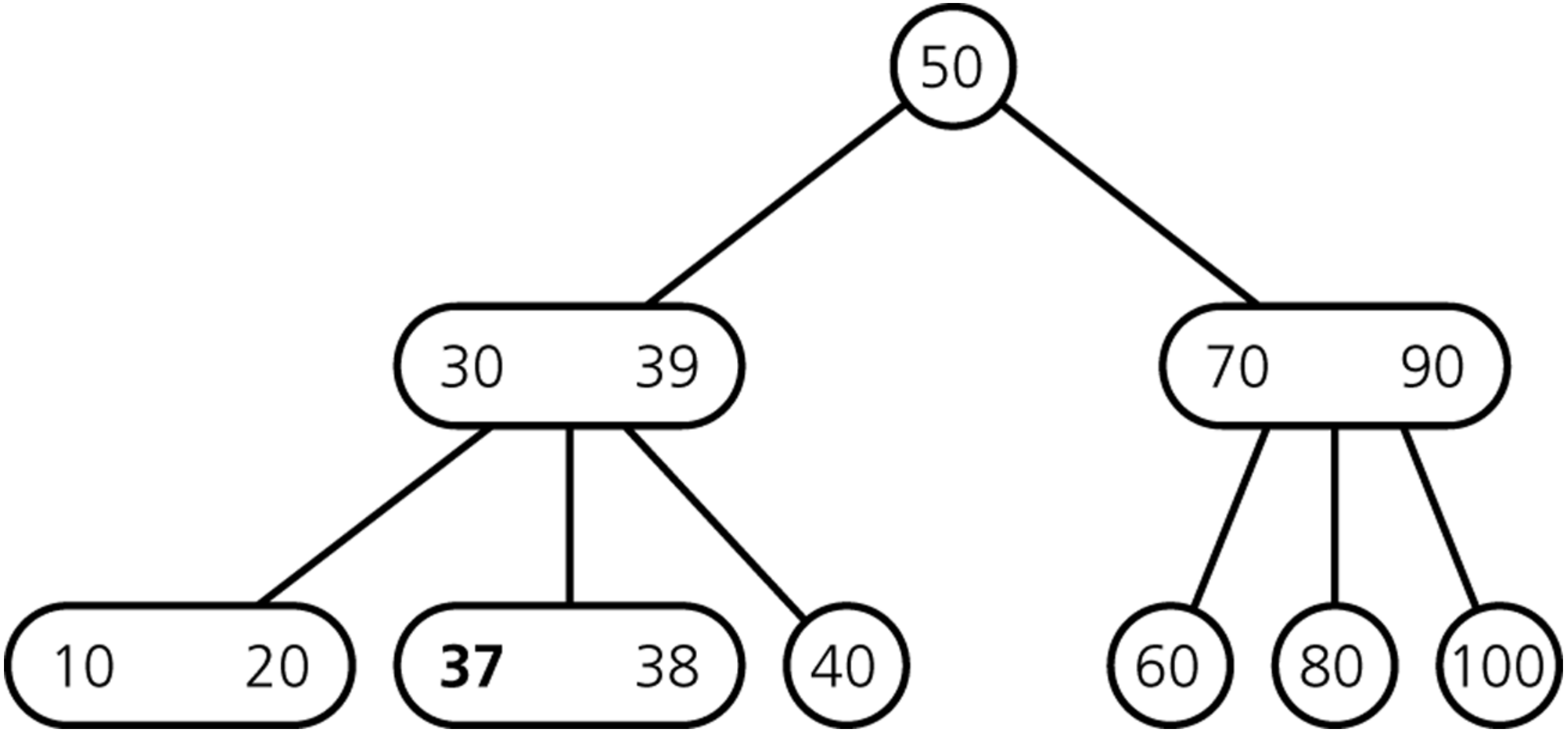
divide leaf and move middle value up to parent

result



Inserting Items

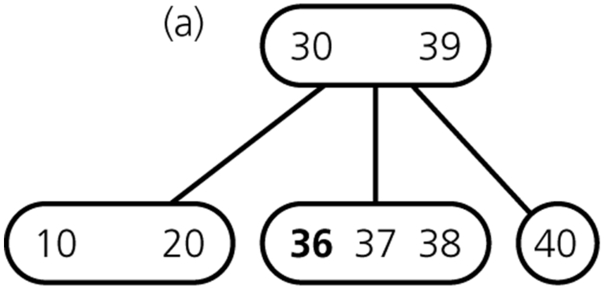
Insert 37



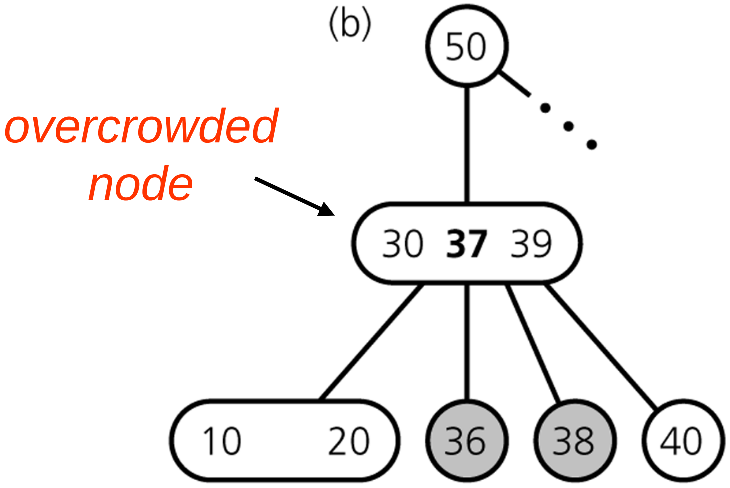
Inserting Items

Insert 36

insert in leaf



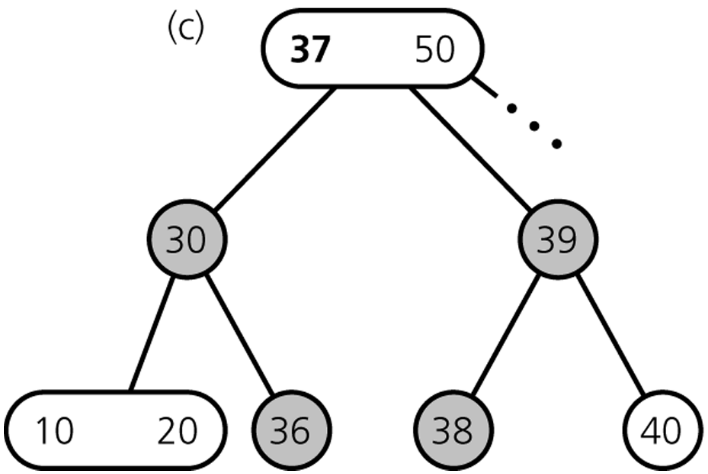
divide leaf and move middle value up to parent



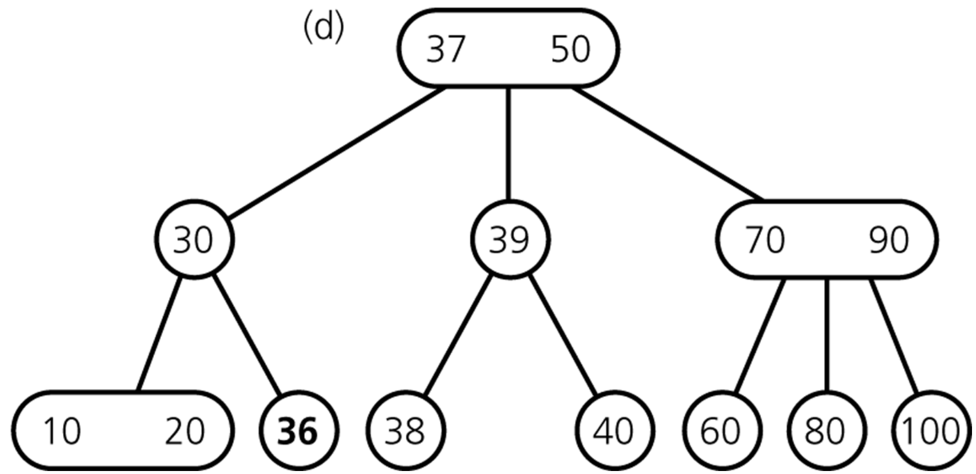
Inserting Items

... still inserting 36

divide overcrowded node,
move middle value up to parent,
attach children to smallest and largest

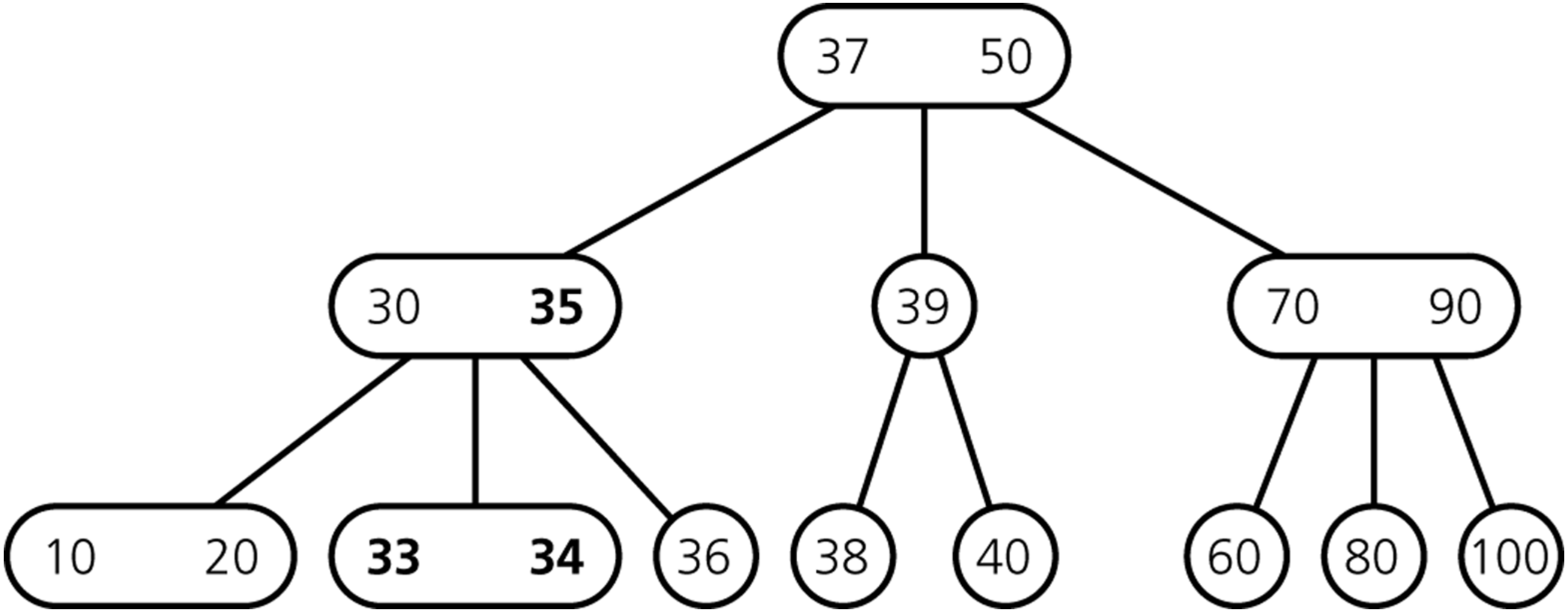


result

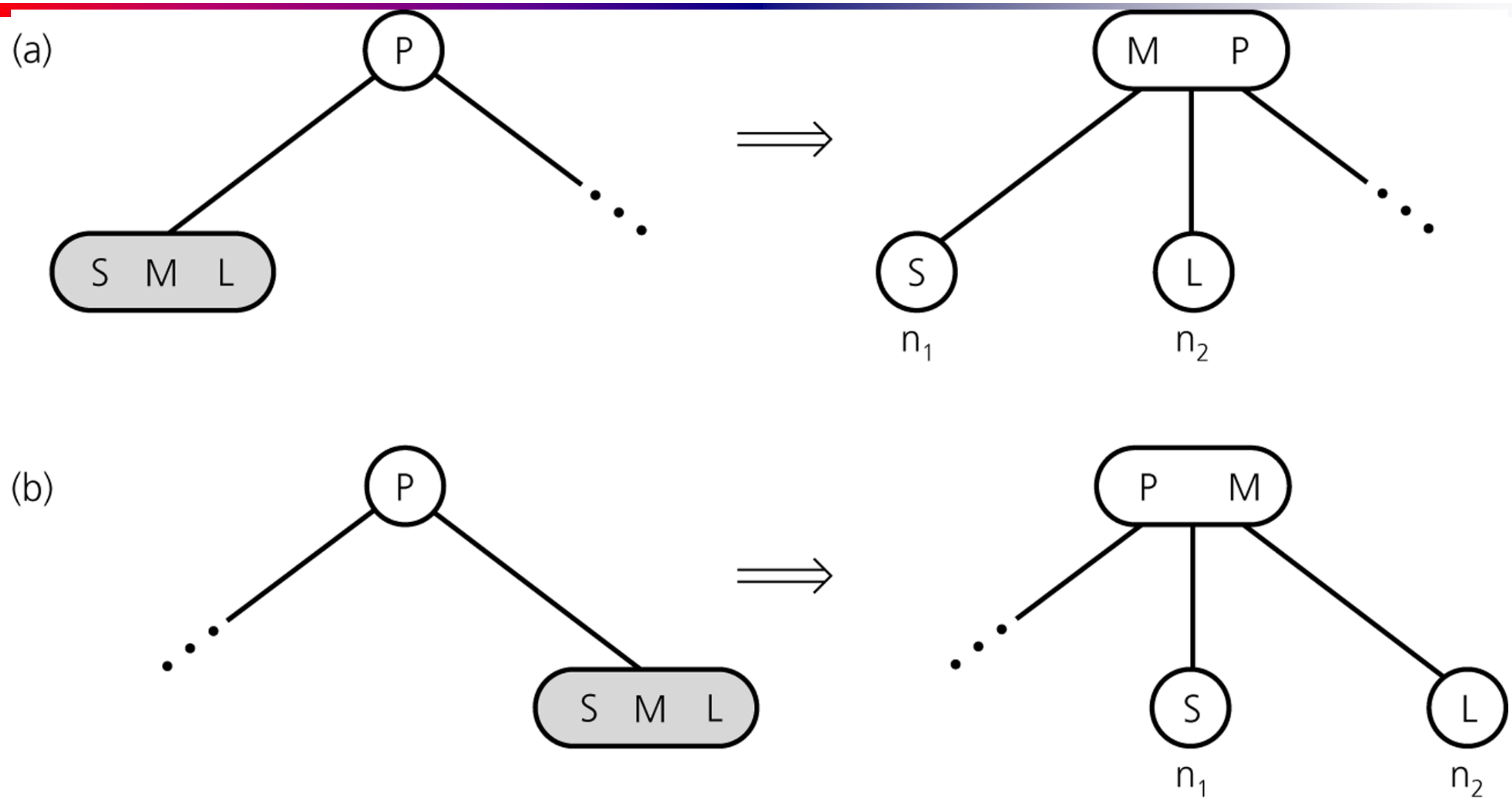


Inserting Items

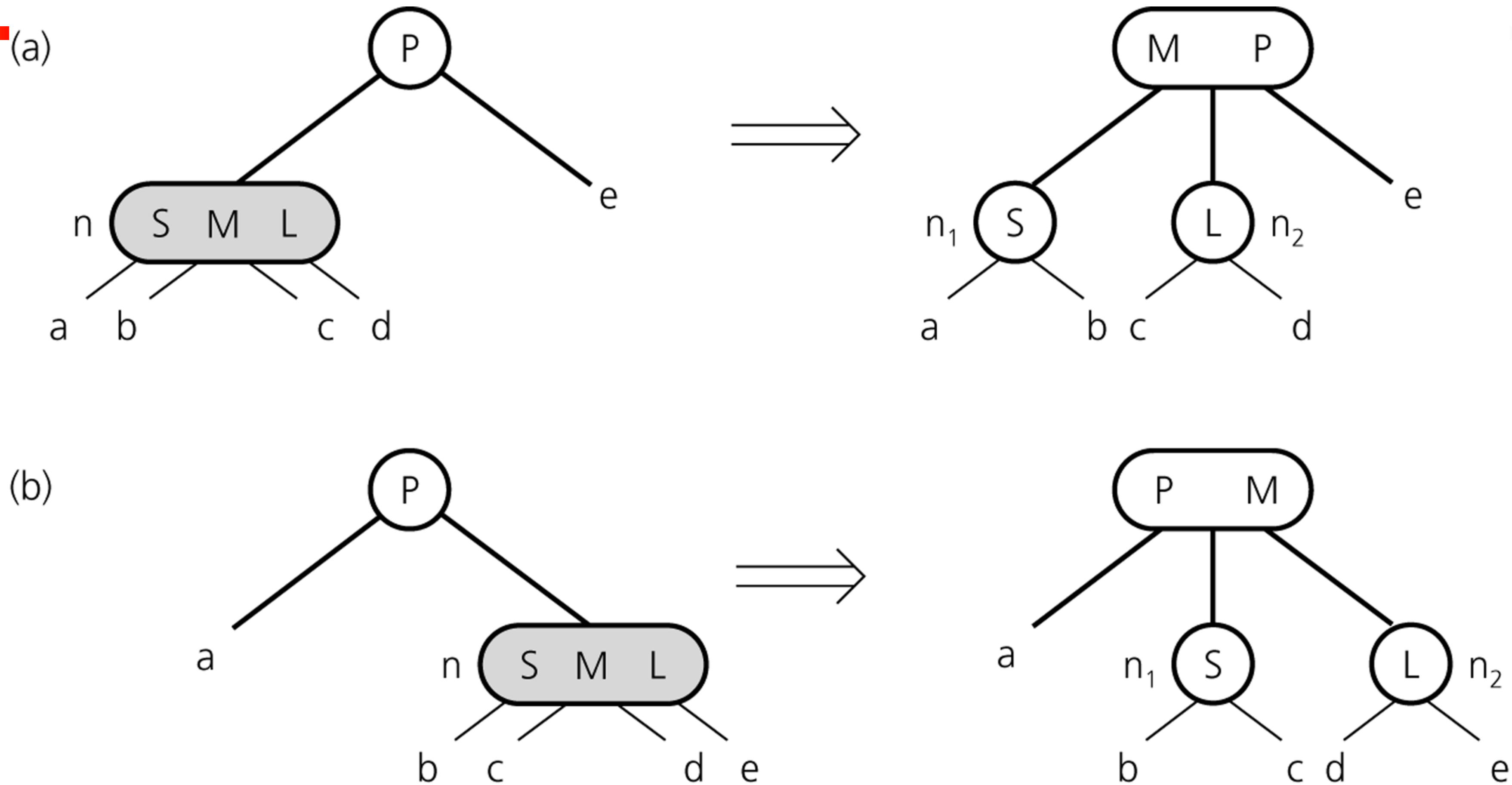
After Insertion of 35, 34, 33



Inserting so far

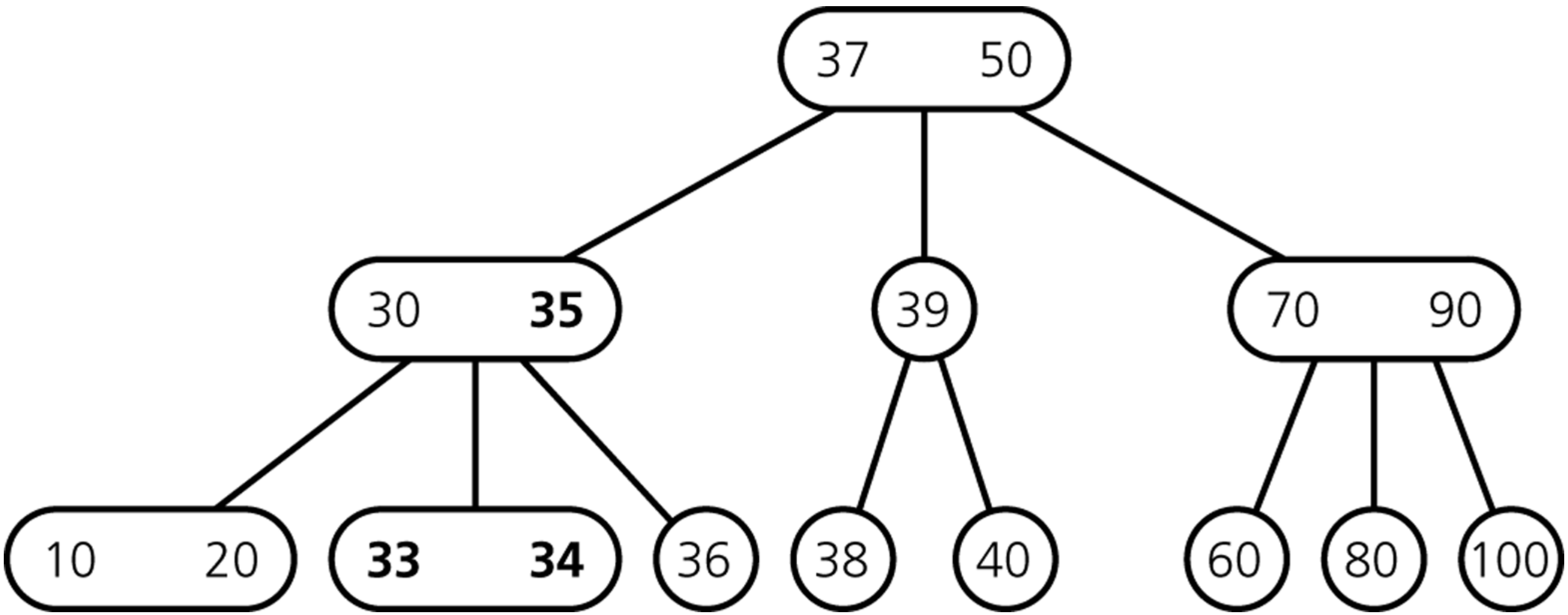


Inserting so far



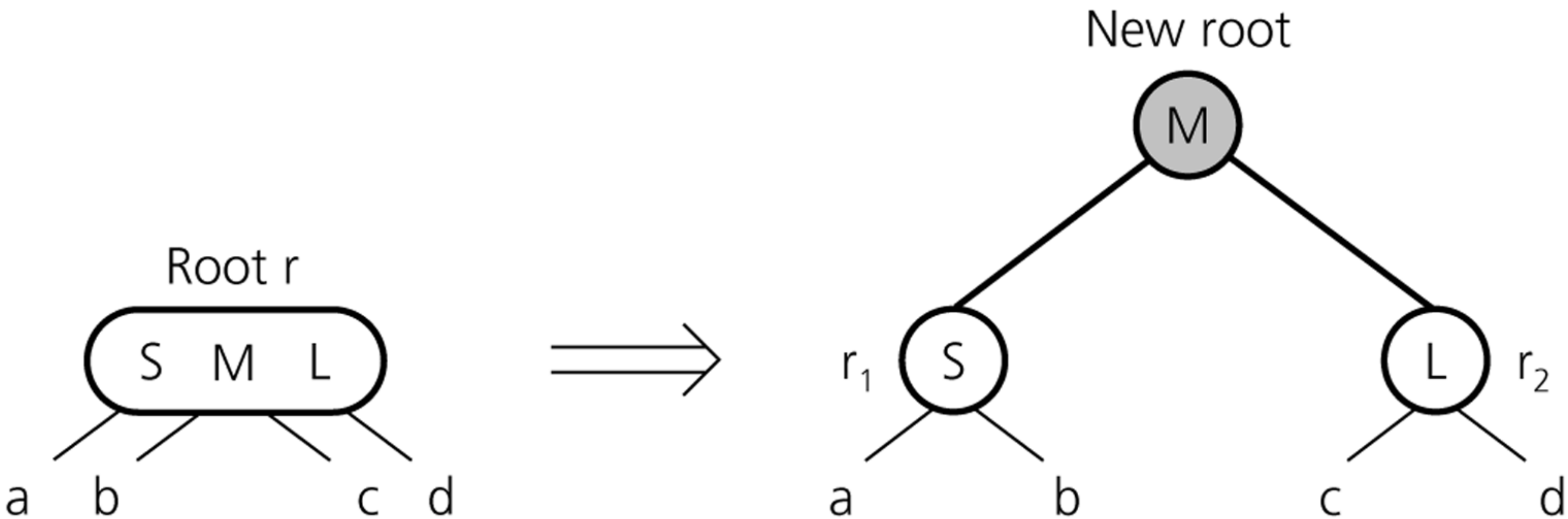
Inserting Items

How do we insert 32?



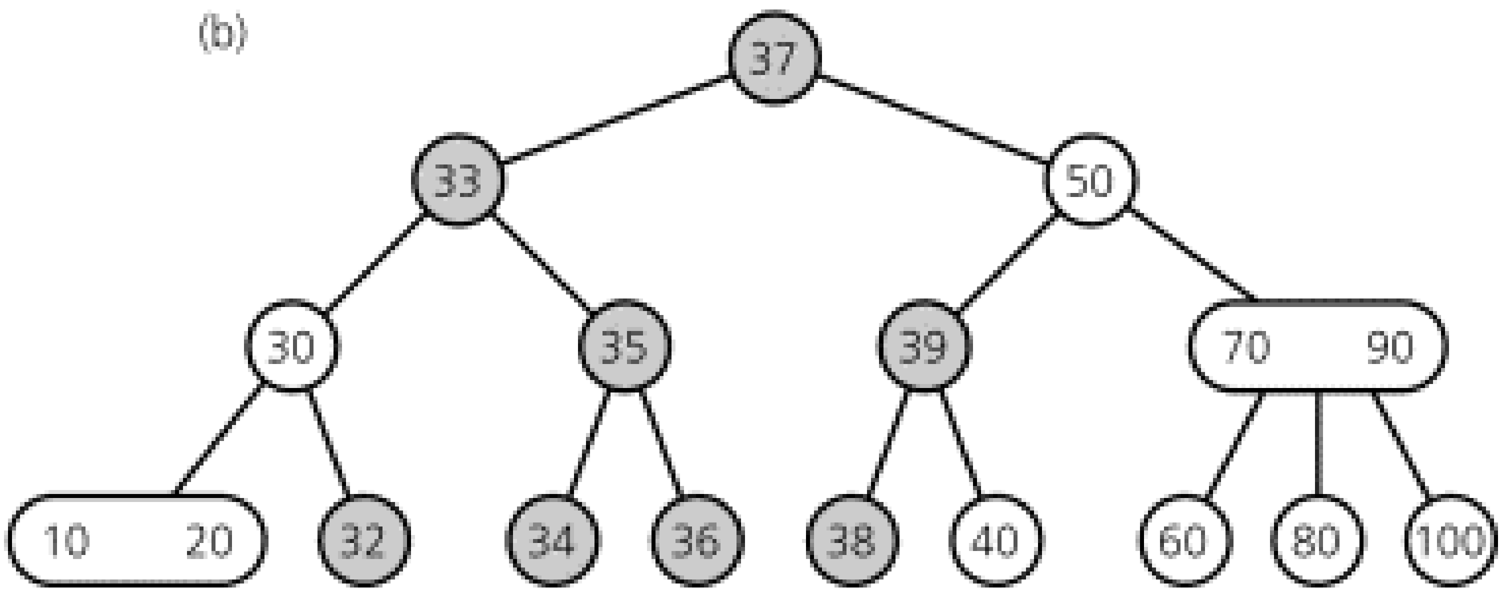
Inserting Items

→ *creating a new root if necessary*
→ *tree grows at the root*



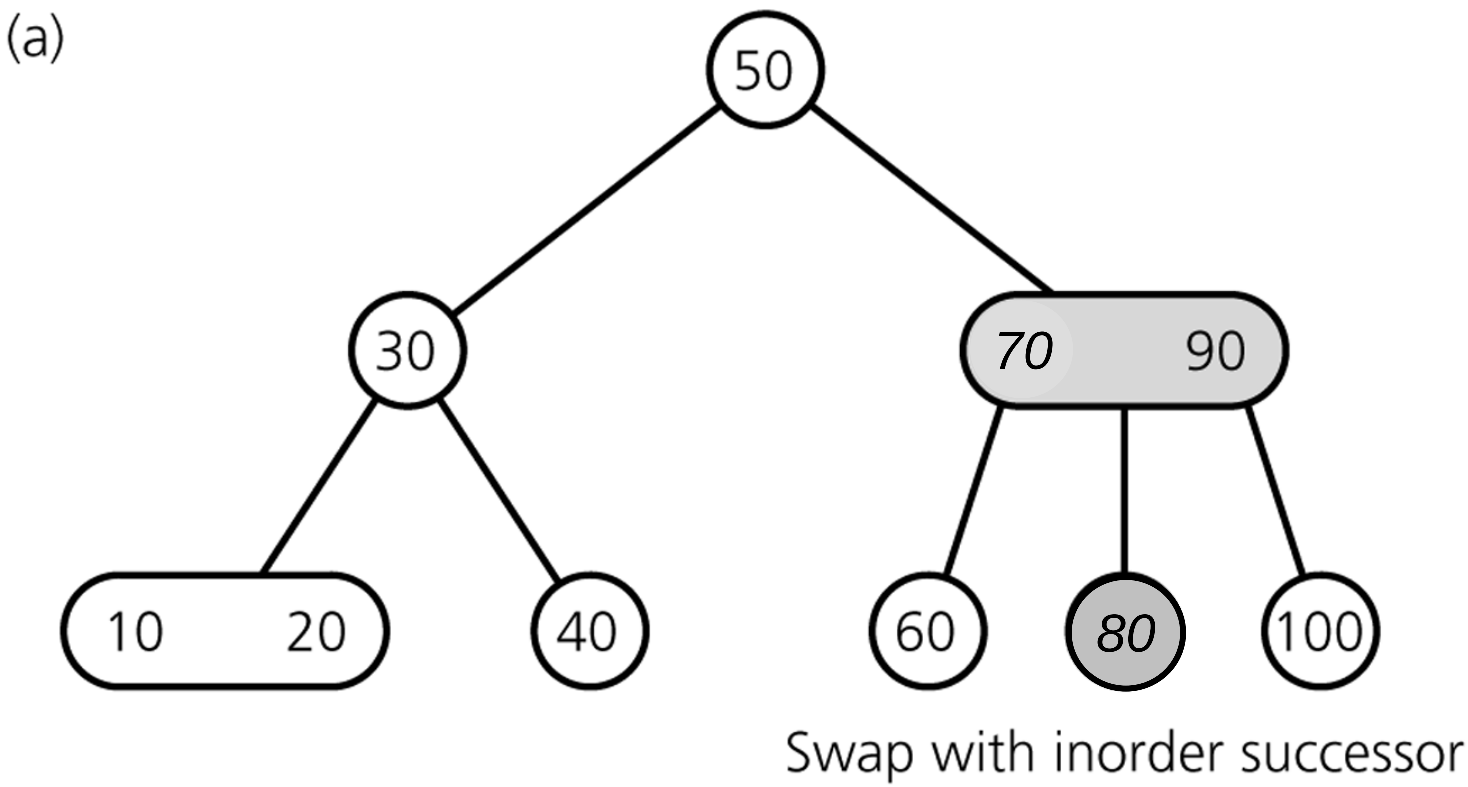
Inserting Items

Final Result



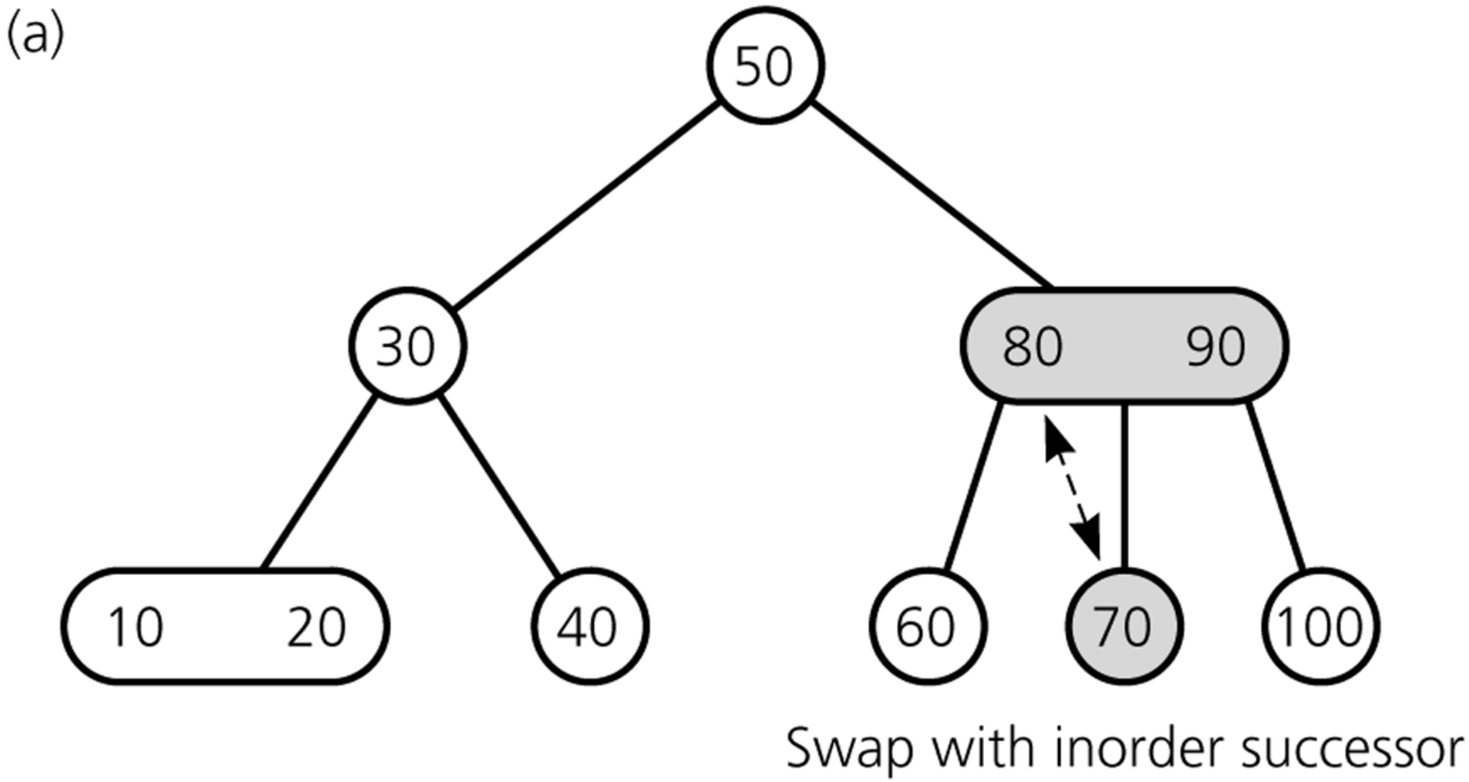
Deleting Items

Delete 70



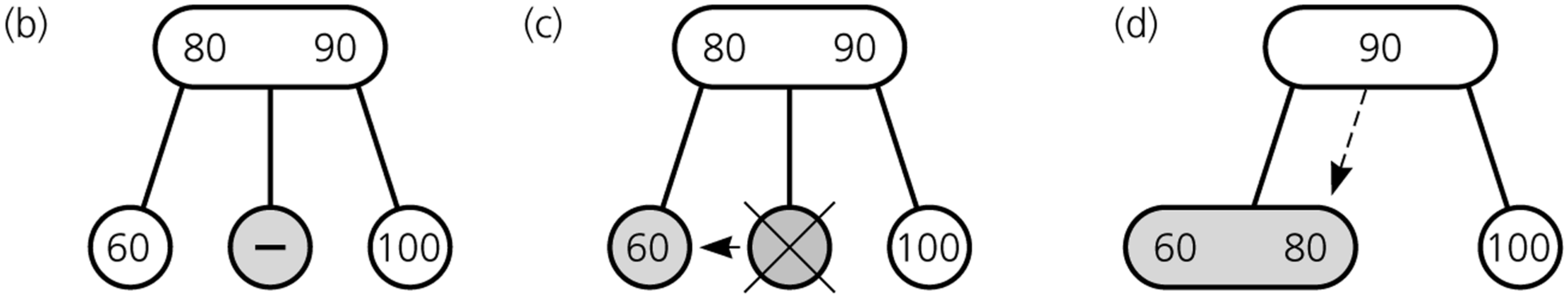
Deleting Items

Deleting 70: swap 70 with inorder successor (80)



Deleting Items

Deleting 70: ... get rid of 70



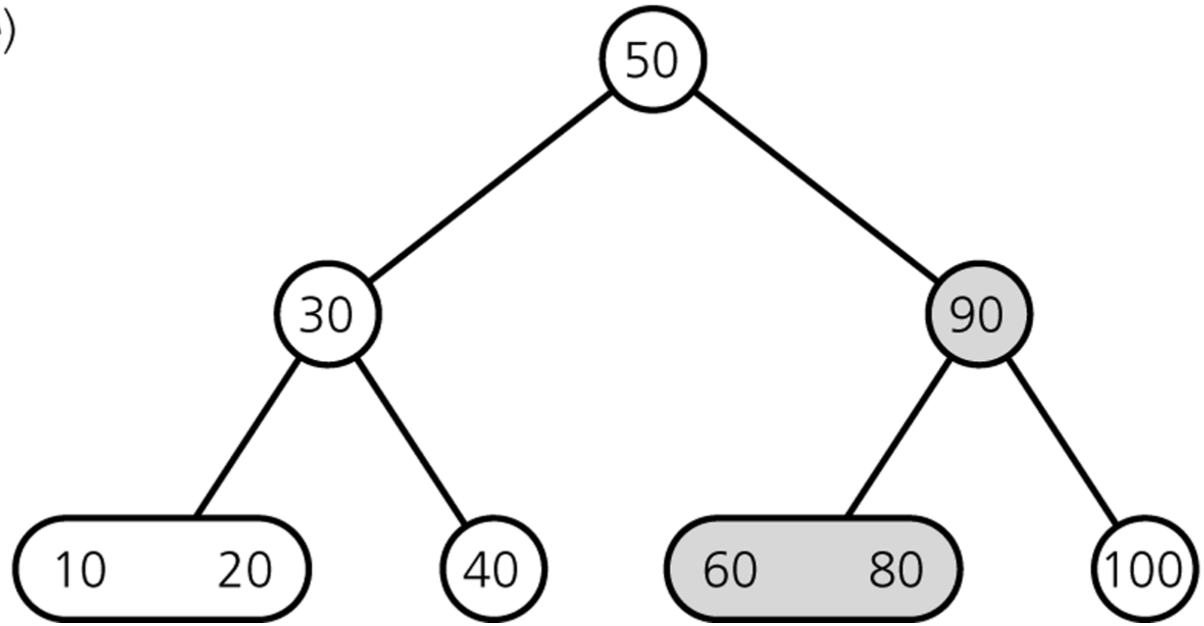
Delete value from leaf

Merge nodes by deleting empty leaf and moving 80 down

Deleting Items

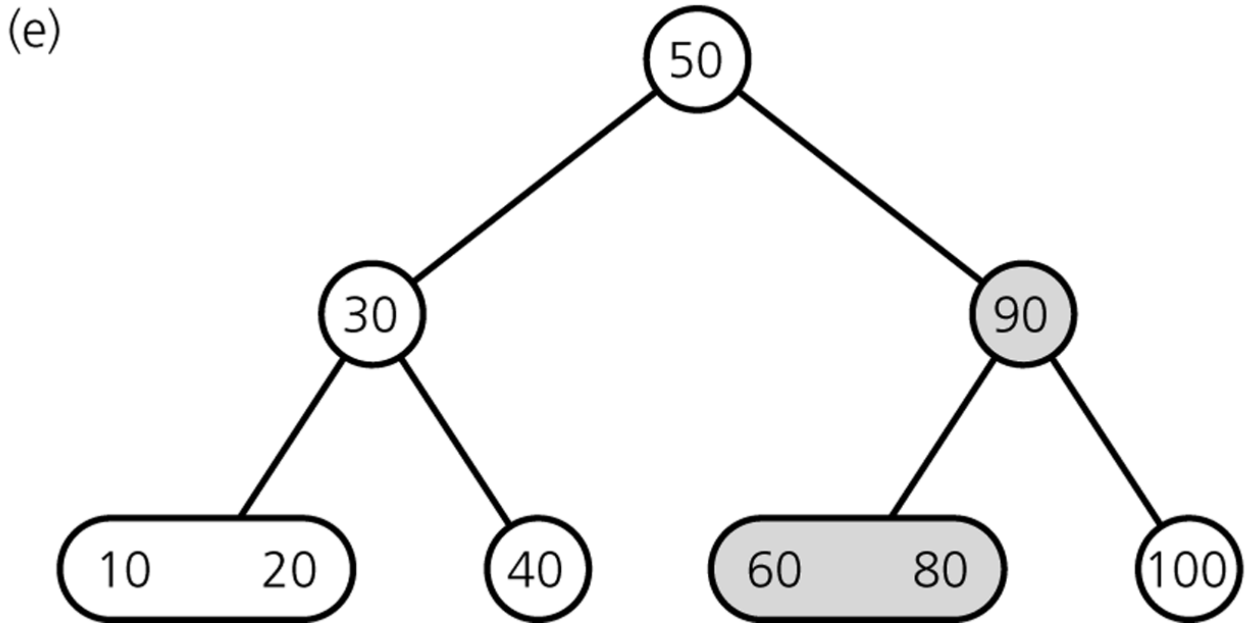
Result

(e)



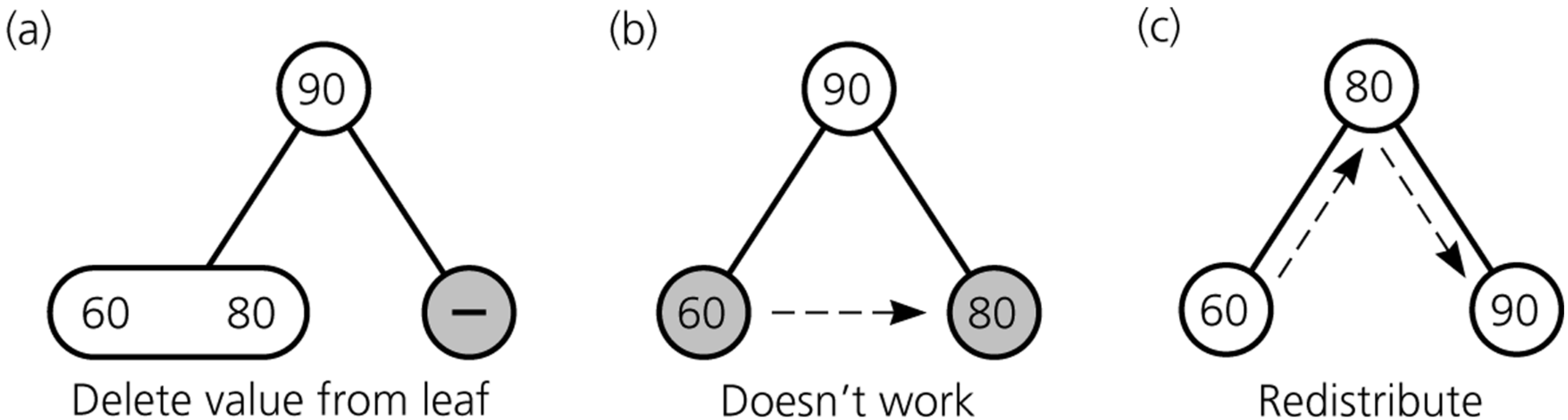
Deleting Items

Delete 100



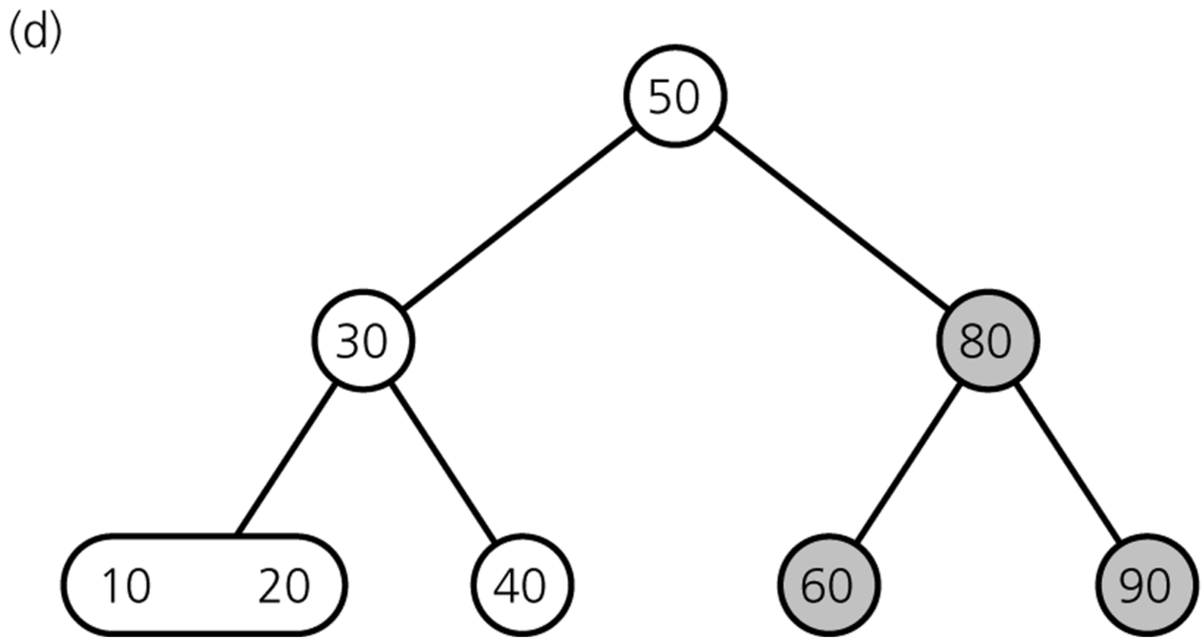
Deleting Items

Deleting 100



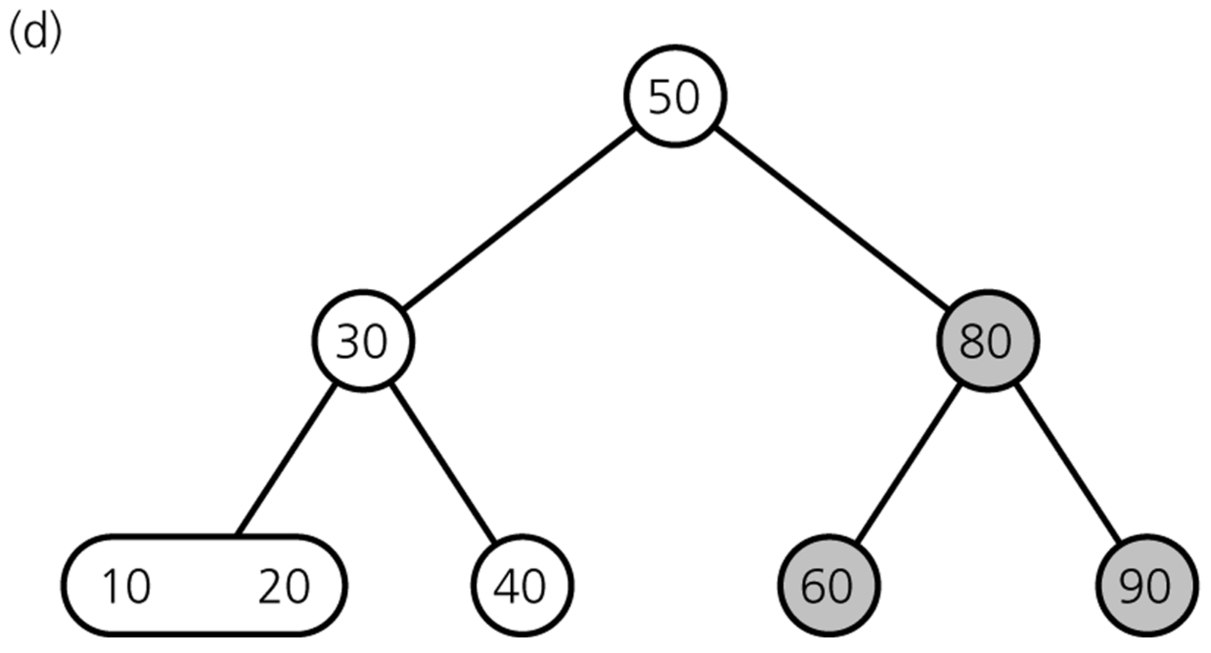
Deleting Items

Result



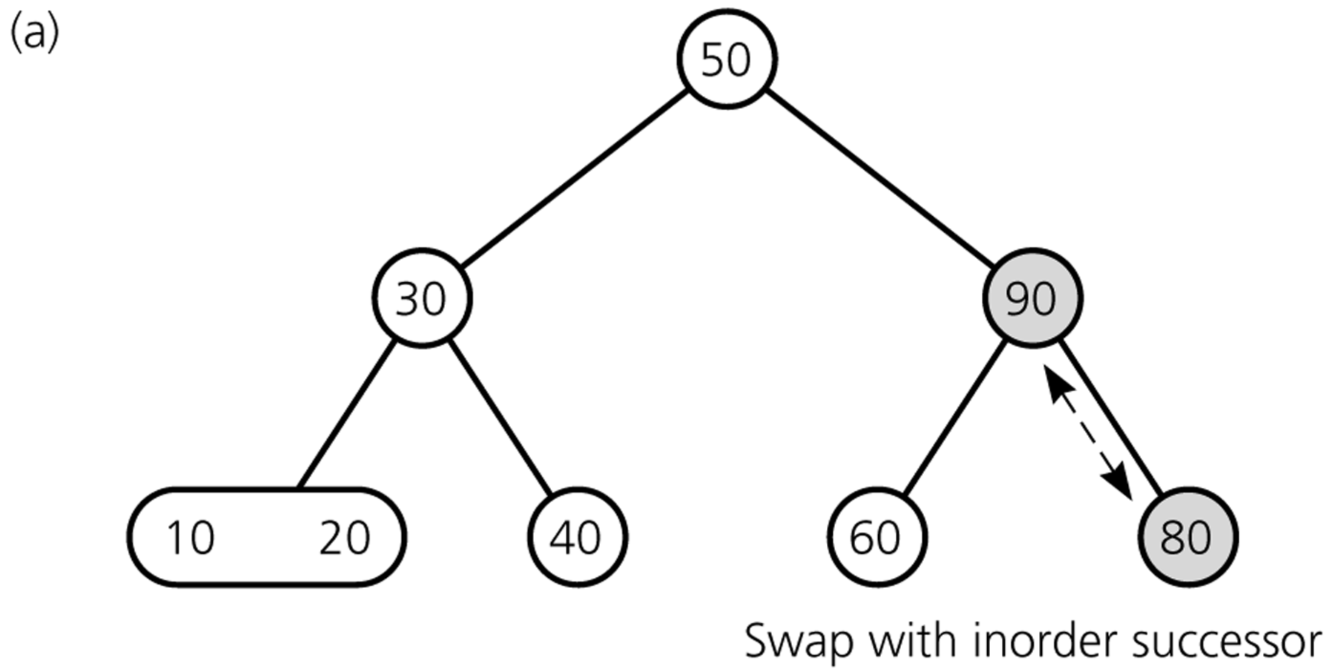
Deleting Items

Delete 80



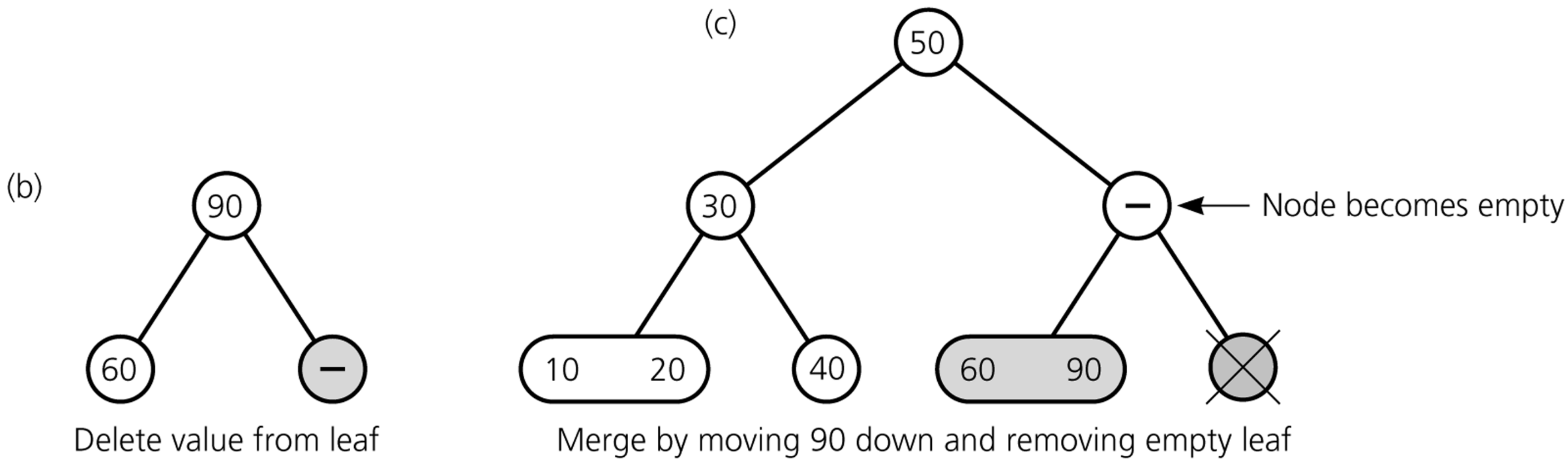
Deleting Items

Deleting 80 ...



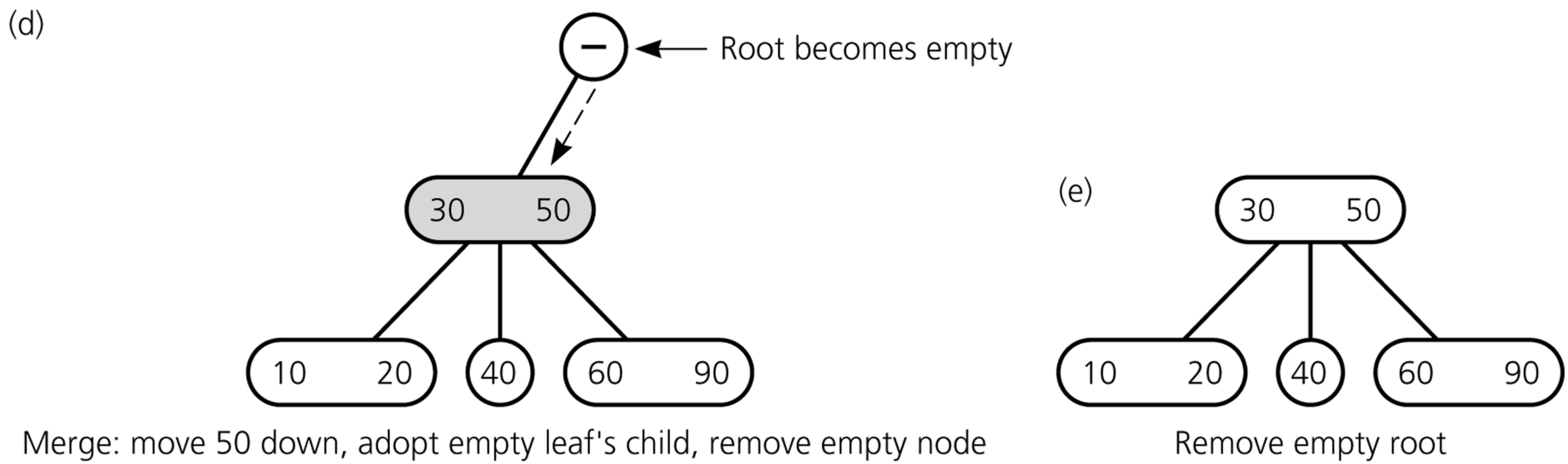
Deleting Items

Deleting 80 ...



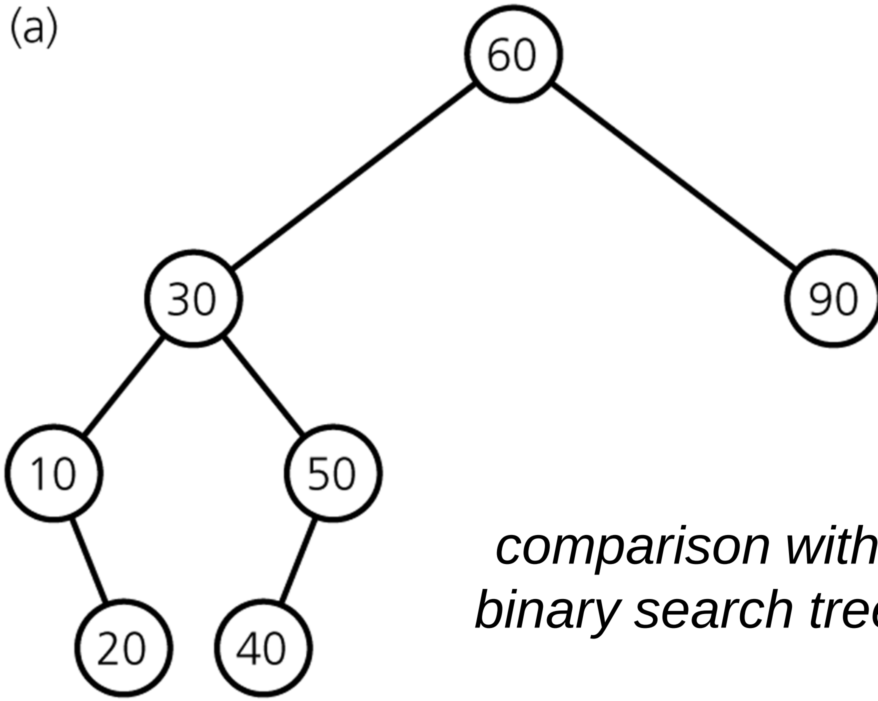
Deleting Items

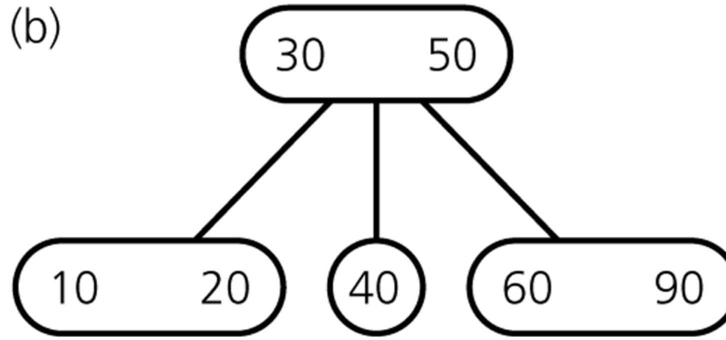
Deleting 80 ...



Deleting Items

Final Result





comparison with binary search tree

Deletion Algorithm I

Deleting item I:

1. *Locate node n, which contains item I*
2. *If node n is not a leaf → swap I with inorder successor*

→ *deletion always begins at a leaf*

3. *If leaf node n contains another item, just delete item I*
   else
      try to redistribute nodes from siblings (see next slide)
      if not possible, merge node (see next slide)

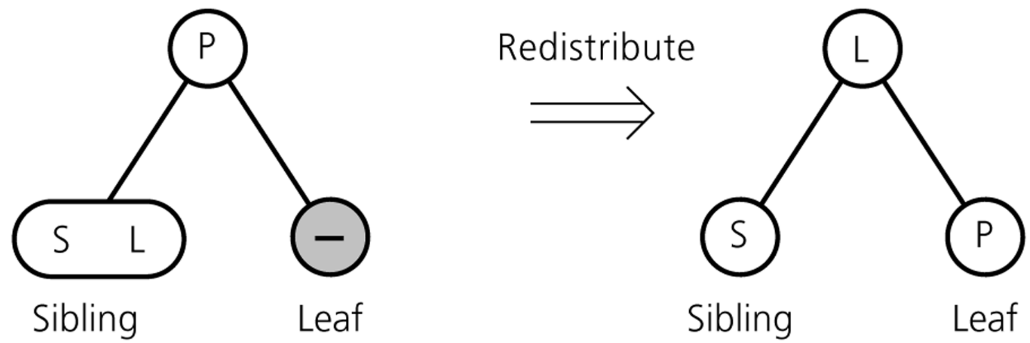
Deletion Algorithm II

Redistribution

A sibling has 2 items:

→ *redistribute item between siblings and parent*

(a)



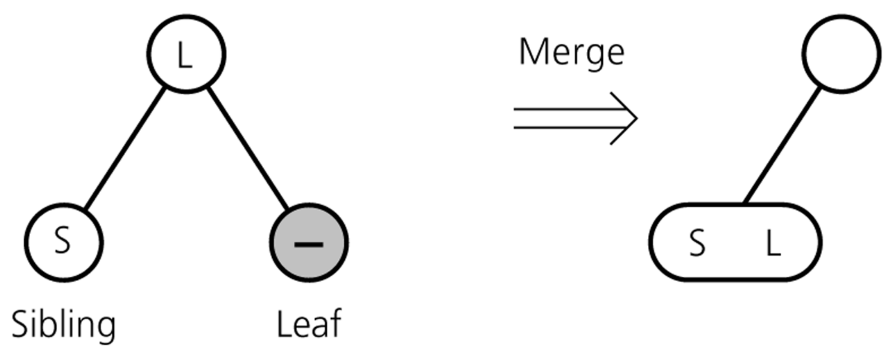
Merging

No sibling has 2 items:

→ *merge node*

→ *move item from parent to sibling*

(b)



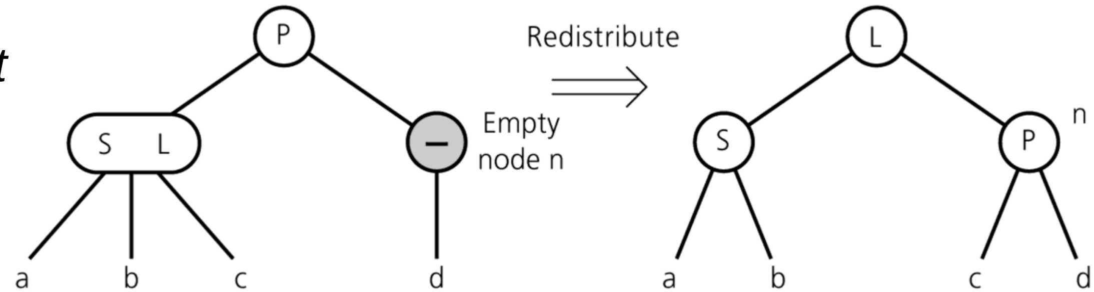
Deletion Algorithm III

Redistribution

Internal node n has no item left

→ *redistribute*

(c)



Merging

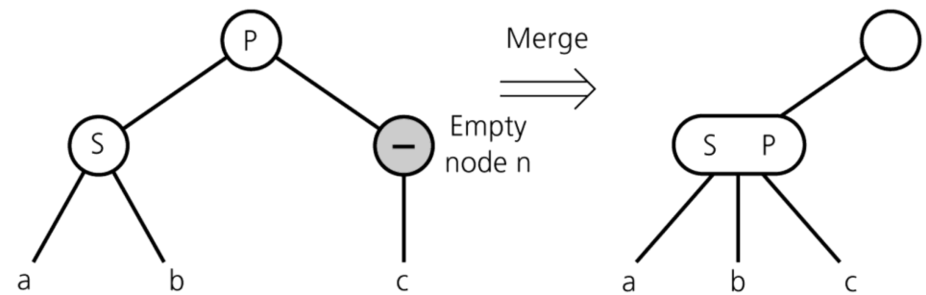
Redistribution not possible:

→ *merge node*

→ *move item from parent to sibling*

→ *adopt child of n*

(d)

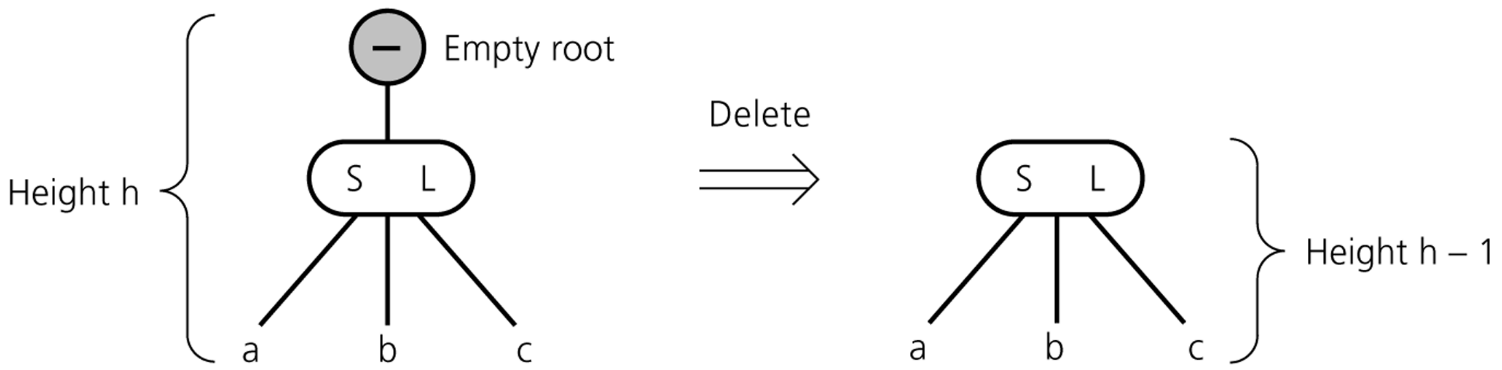


If n's parent ends up without item, apply process recursively

Deletion Algorithm IV

If merging process reaches the root and root is without item
→ delete root

(e)



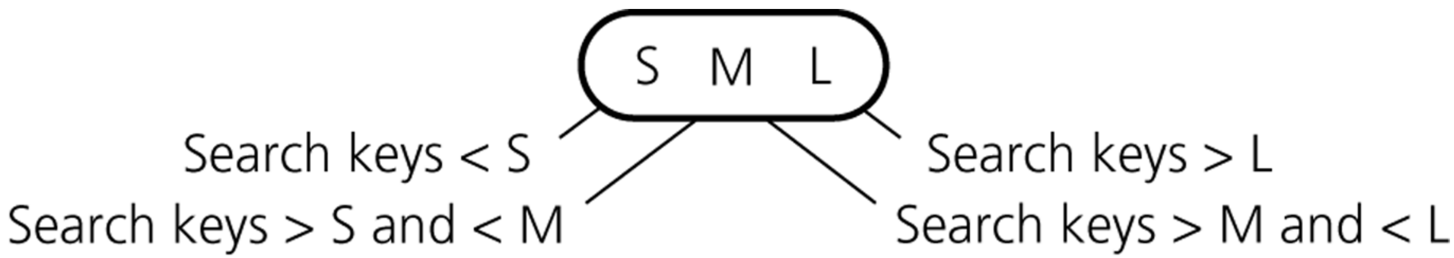
Operations of 2-3 Trees

all operations have time complexity of log n

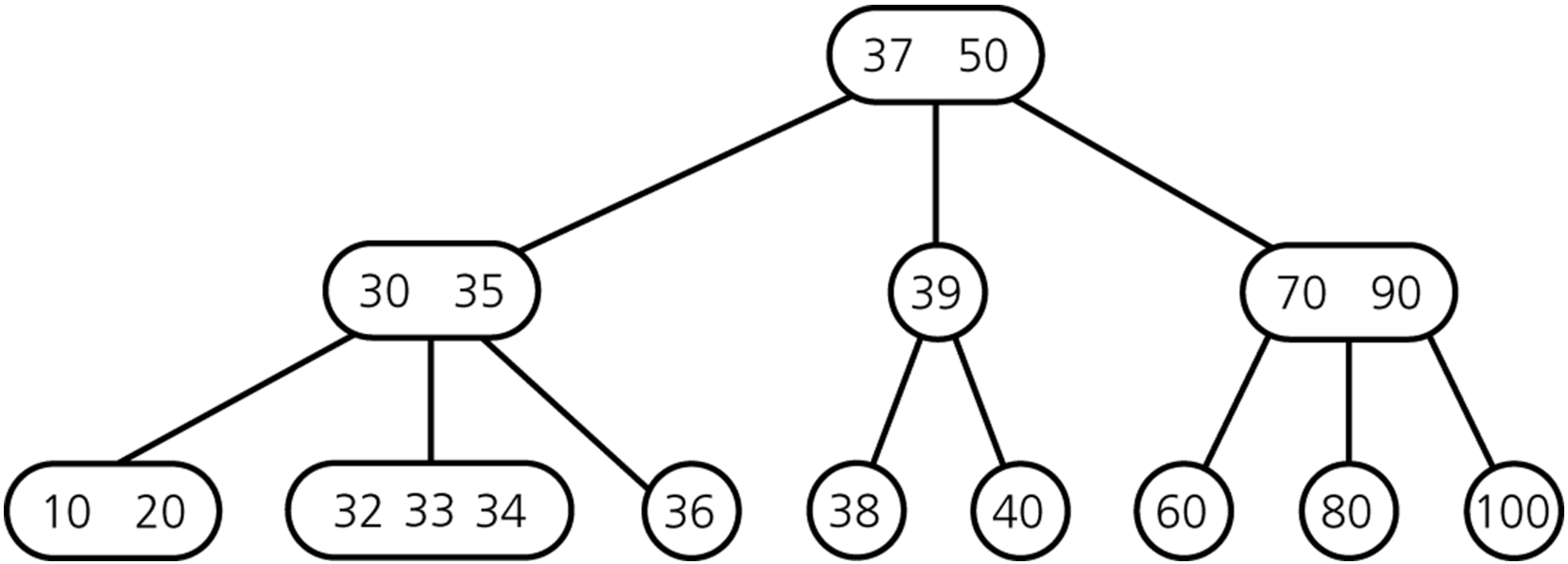
2-3-4 Trees

- *similar to 2-3 trees*
- *4-nodes can have 3 items and 4 children*

4-node



2-3-4 Tree Example



2-3-4 Tree: Insertion

Insertion procedure:

- *similar to insertion in 2-3 trees*
- *items are inserted at the leafs*
- *since a 4-node cannot take another item,*
  4-nodes are split up during insertion process

Strategy

- *on the way from the root down to the leaf:*
  split up all 4-nodes "on the way"

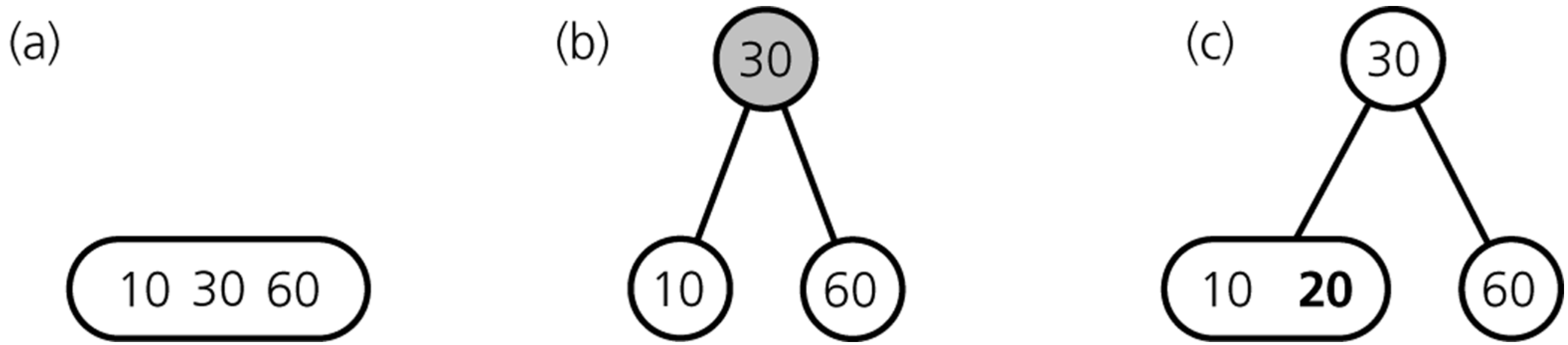
→ *insertion can be done in one pass*
(remember: in 2-3 trees, a reverse pass might be necessary)

2-3-4 Tree: Insertion

Inserting 60, 30, 10, 20, 50, 40, 70, 80, 15, 90, 100

2-3-4 Tree: Insertion

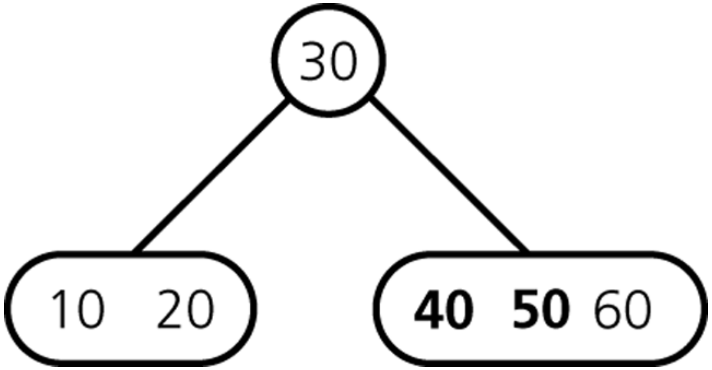
Inserting 60, 30, 10, 20 ...



... 50, 40 ...

2-3-4 Tree: Insertion

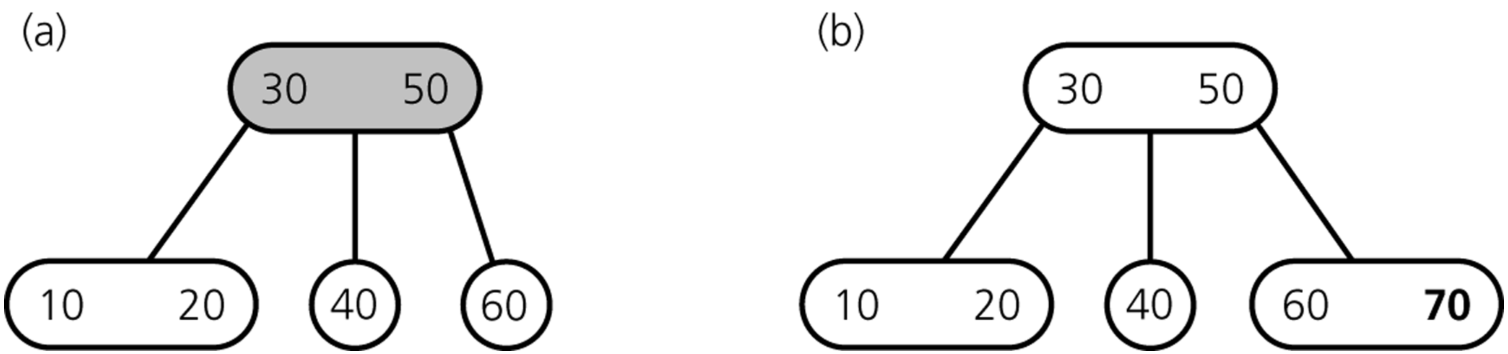
Inserting 50, 40 ...



... 70, ...

2-3-4 Tree: Insertion

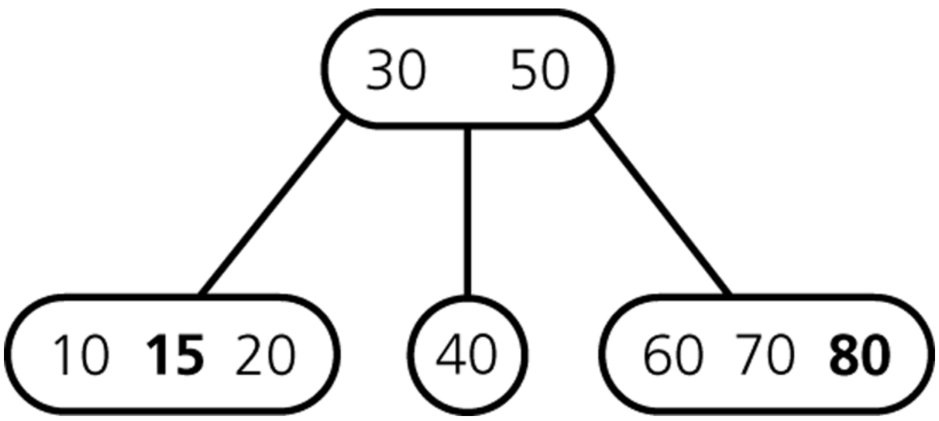
Inserting 70 ...



... 80, 15 ...

2-3-4 Tree: Insertion

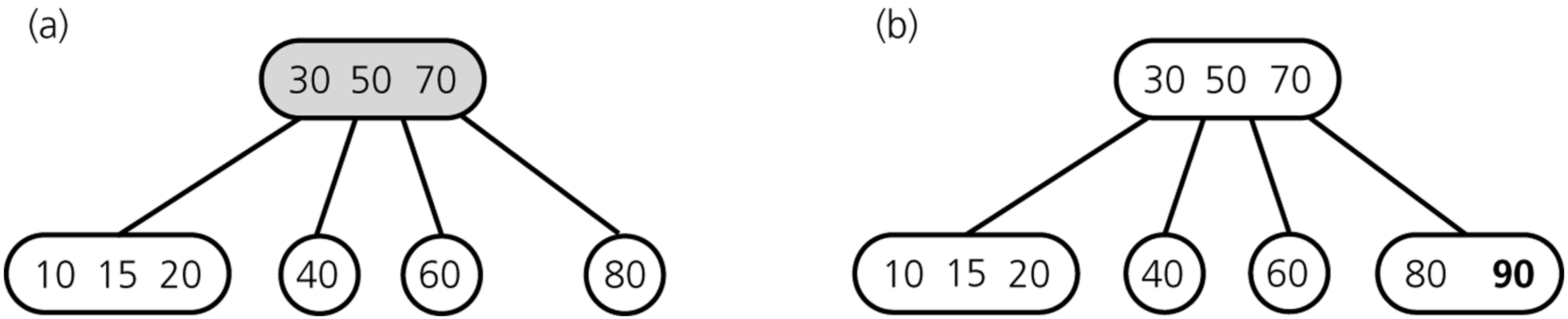
Inserting 80, 15 ...



... 90 ...

2-3-4 Tree: Insertion

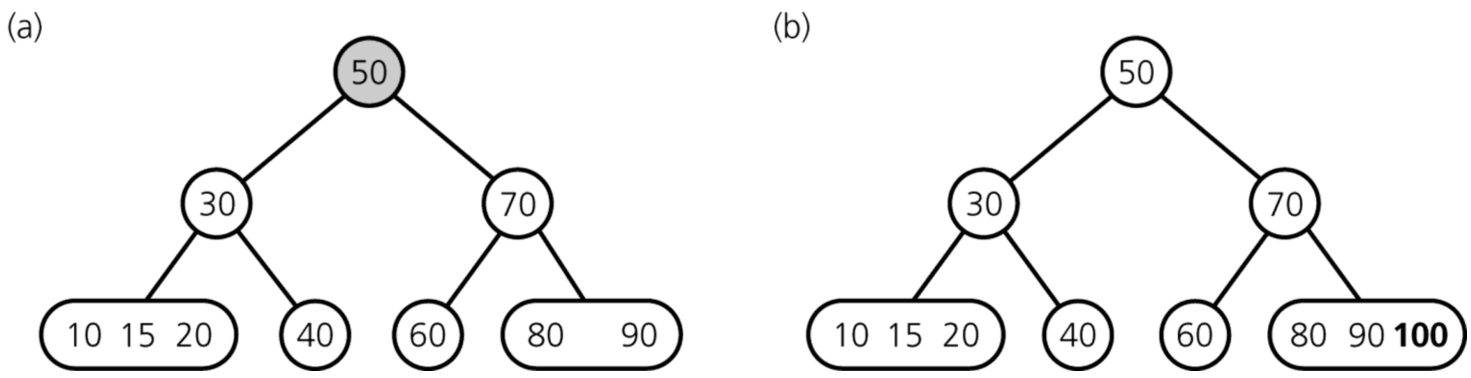
Inserting 90 ...



... 100 ...

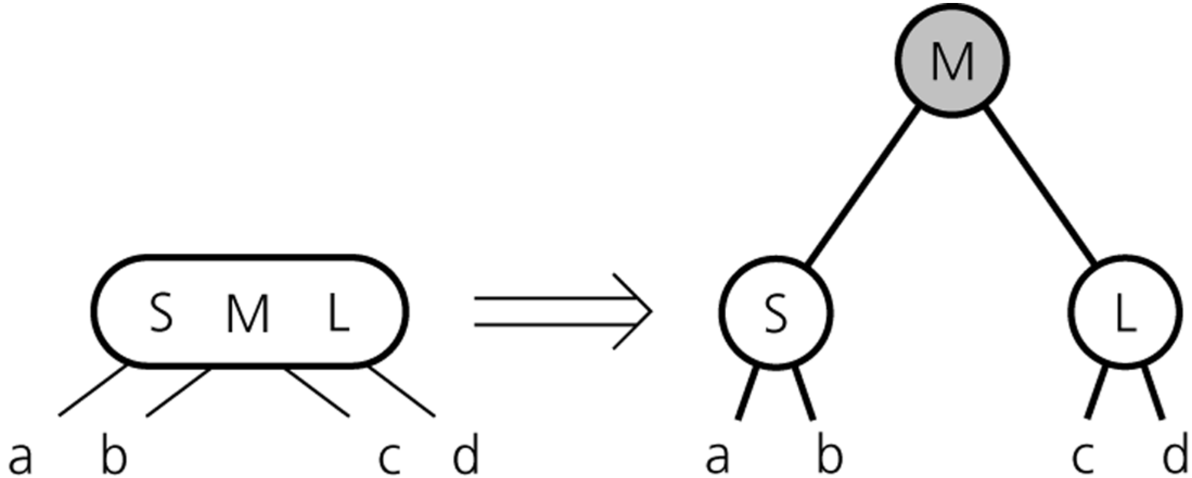
2-3-4 Tree: Insertion

Inserting 100 ...



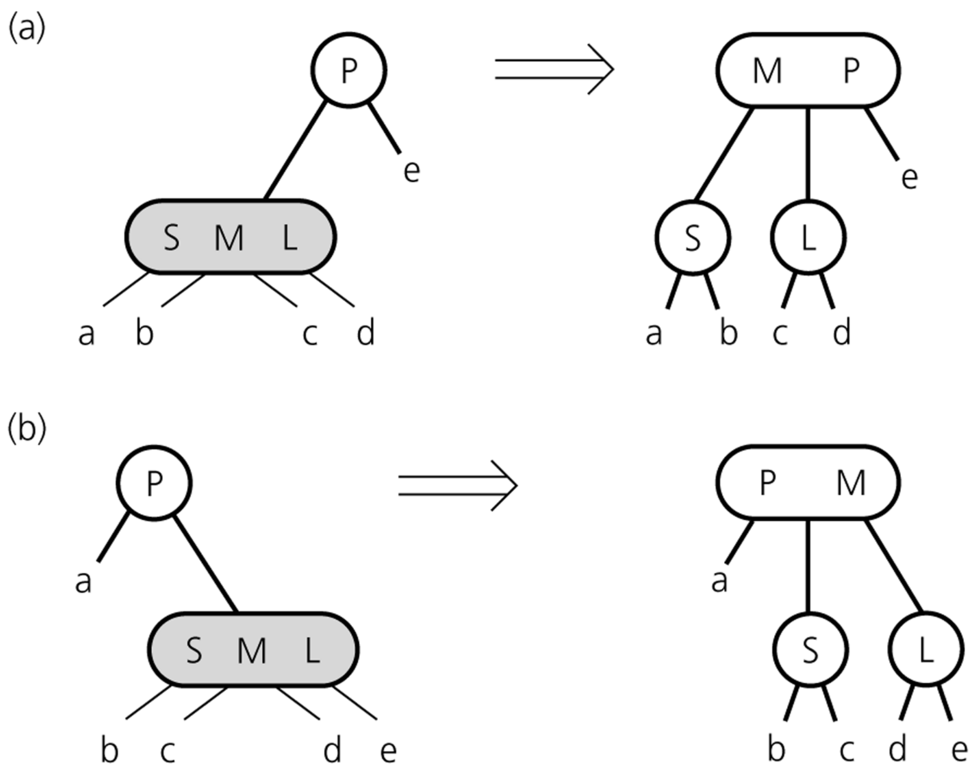
2-3-4 Tree: Insertion Procedure

Splitting 4-nodes during Insertion



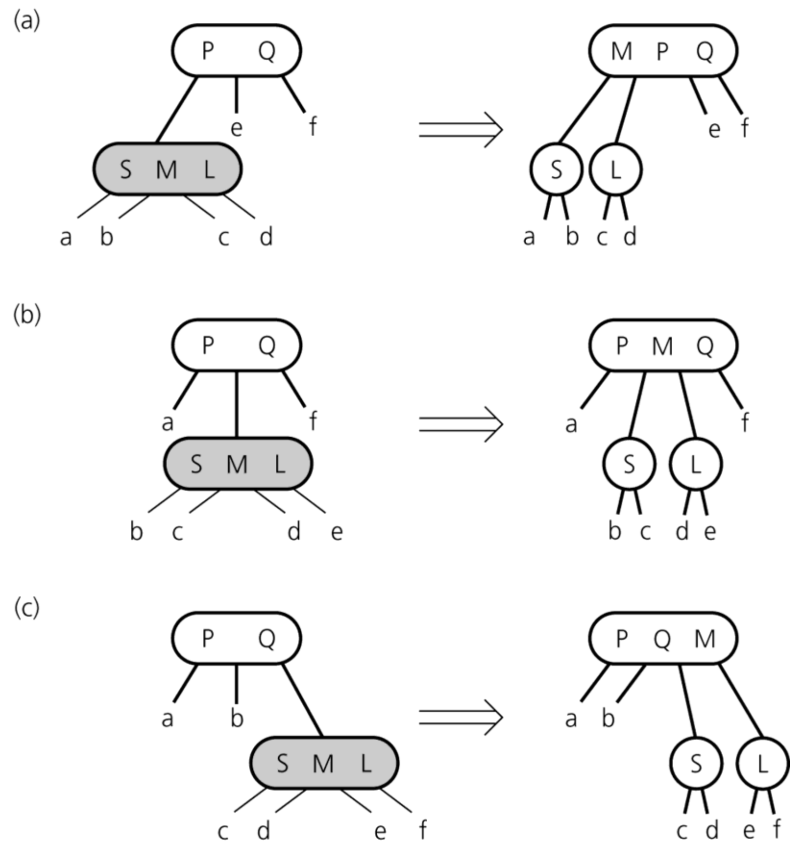
2-3-4 Tree: Insertion Procedure

Splitting a 4-node whose parent is a 2-node during insertion



2-3-4 Tree: Insertion Procedure

Splitting a 4-node whose parent is a 3-node during insertion



2-3-4 Tree: Deletion

Deletion procedure:

- *similar to deletion in 2-3 trees*
- *items are deleted at the leafs*
  → swap item of internal node with inorder successor
- *note: a 2-node leaf creates a problem*

Strategy (different strategies possible)

- *on the way from the root down to the leaf:*
  turn 2-nodes (except root) into 3-nodes

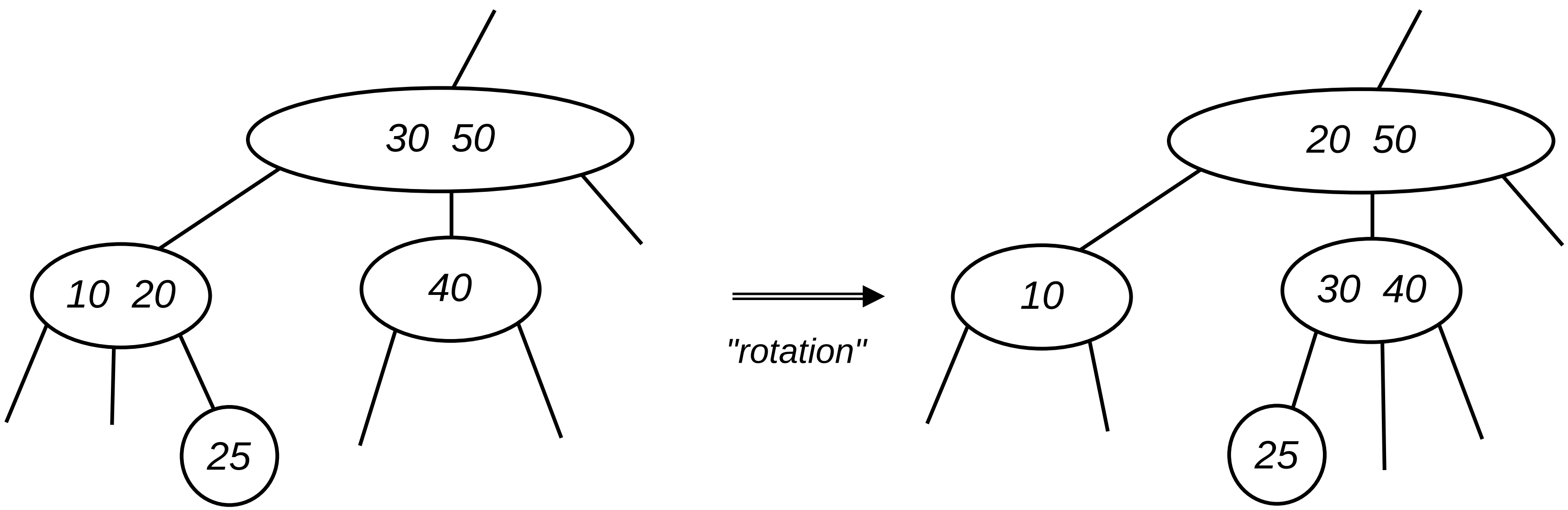
→ deletion can be done in one pass
(remember: in 2-3 trees, a reverse pass might be necessary)

2-3-4 Tree: Deletion

Turning a 2-node into a 3-node ...

Case 1: an adjacent sibling has 2 or 3 items

→ *"steal" item from sibling by rotating items and moving subtree*



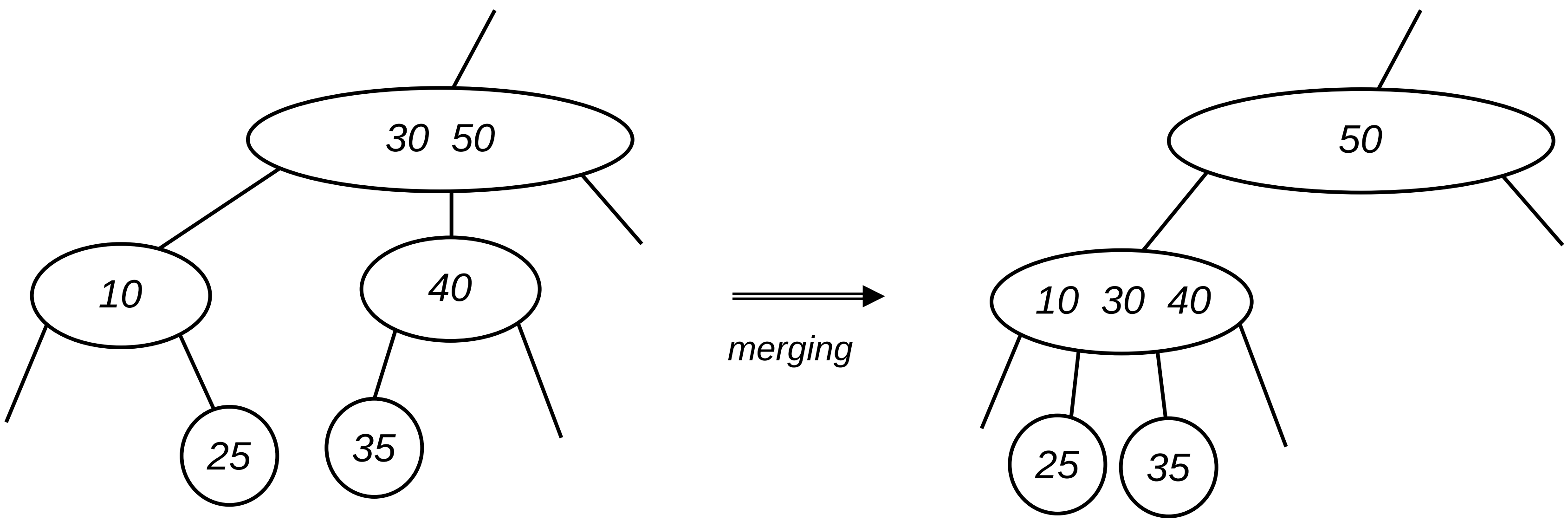
2-3-4 Tree: Deletion

Turning a 2-node into a 3-node ...

Case 2: each adjacent sibling has only one item

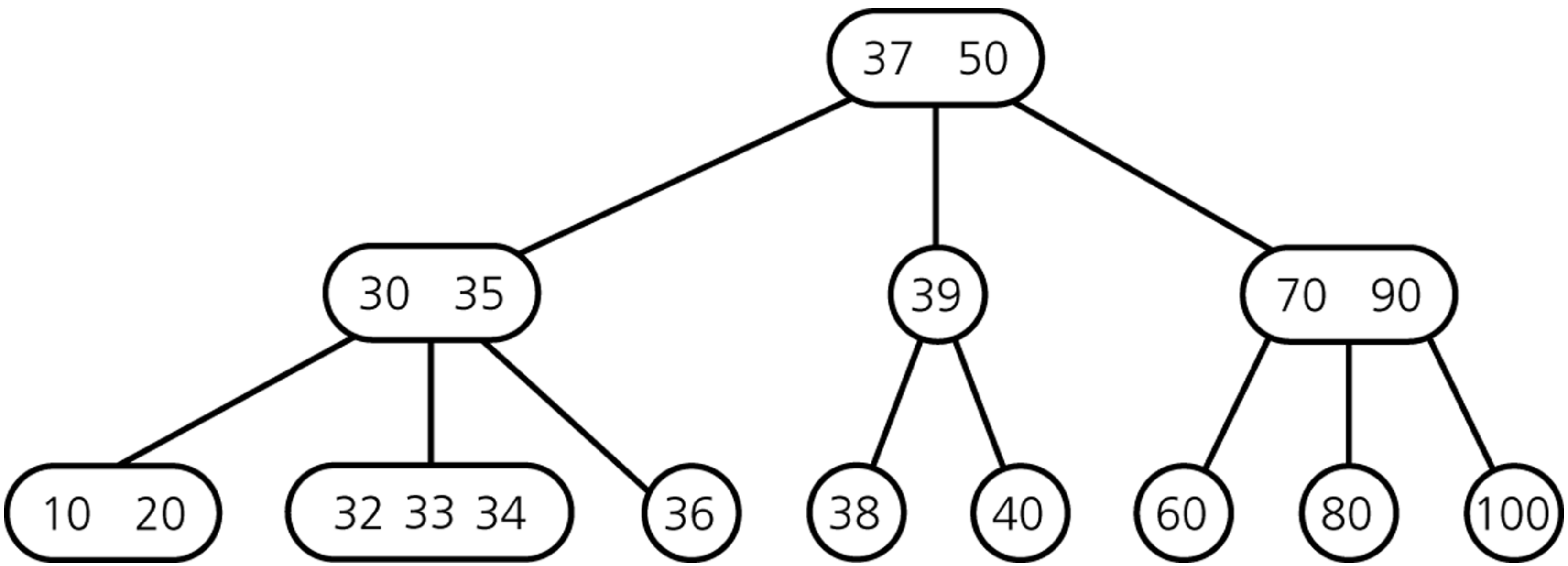
→ *"steal" item from parent and merge node with sibling*

(note: parent has at least two items, unless it is the root)

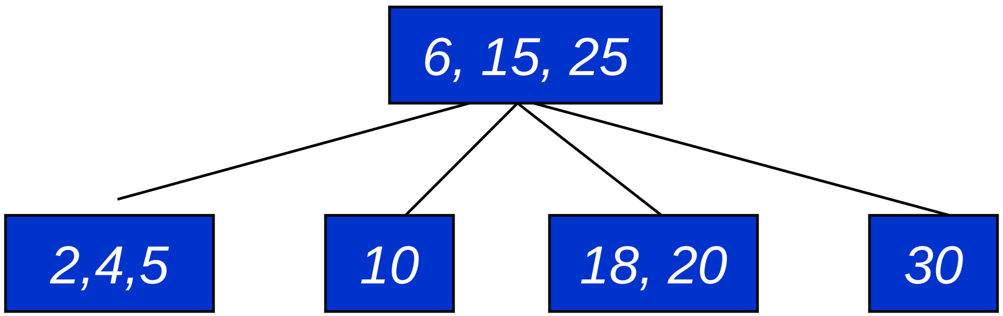


2-3-4 Tree: Deletion Practice

Delete 32, 35, 40, 38, 39, 37, 60

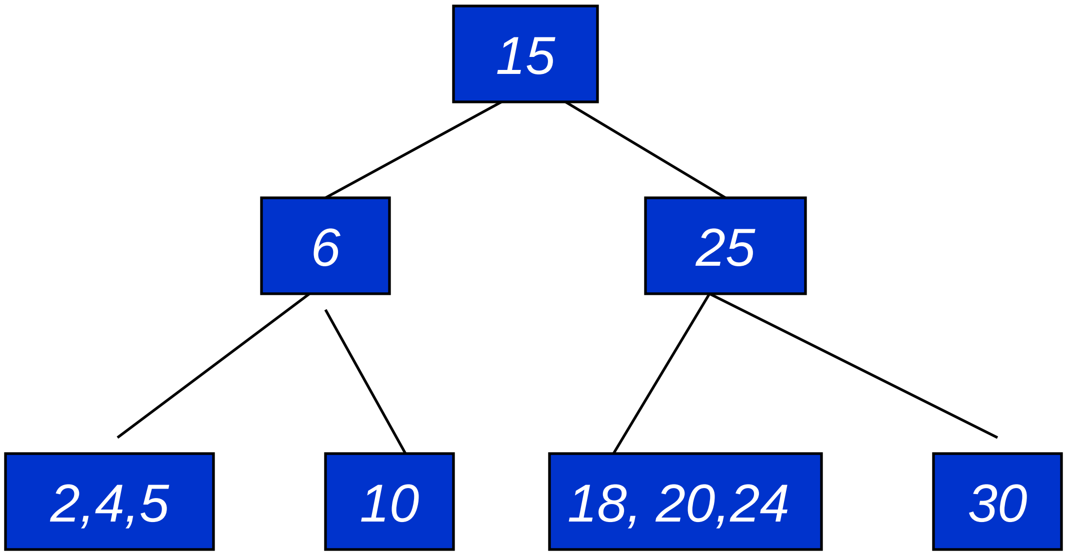


2-3-4 Insert Example



Insert 24, then 19

Insert 24: Split root first



Insert 19, Split leaf (20 up) first