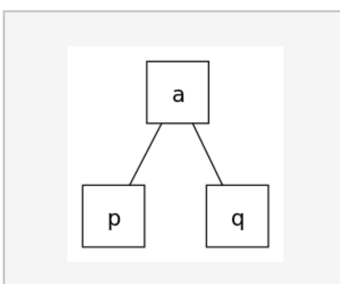
CMSC 132: Object-Oriented Programming II

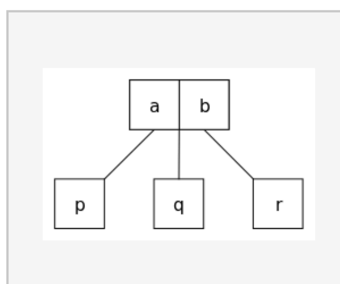
2-3-4 Tree

2-3-4 Tree

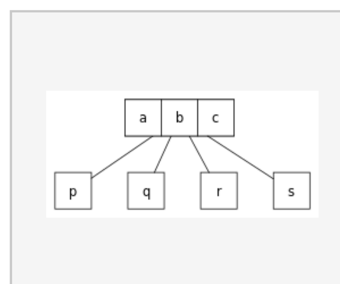
- Self-balancing tree
- every internal node has either two, three, or four child nodes.
  - a 2-node has one data element, and if internal has two child nodes;
  - a 3-node has two data elements, and if internal has three child nodes;
  - a 4-node has three data elements, and if internal has four child nodes.



2-node



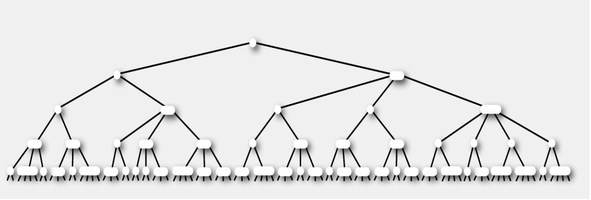
3-node



4-node

2-3-4 Tree Properties

- Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively.
- All leaves are at the same depth (the bottom level).
- All data is kept in sorted order.



- Tree height.
  - Worst case: lg N [all 2-nodes]
  - Best case: log4 N = 1/2 lg N [all 4-nodes]
  - Between 10 and 20 for 1 million nodes.
  - Between 15 and 30 for 1 billion nodes.
- Guaranteed logarithmic performance for both search and insert.

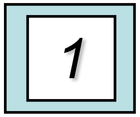
2-3-4 Tree Insertion

1. If the current node is a 4-node:
   - Remove and save the middle value to get a 3-node.
   - Split the remaining 3-node up into a pair of 2-nodes (the now missing middle value is handled in the next step).
   - If this is the root node (which thus has no parent):
     - the middle value becomes the new root 2-node and the tree height increases by 1. Ascend into the root.
   - Otherwise, push the middle value up into the parent node. Ascend into the parent node.
2. Find the child whose interval contains the value to be inserted.
3. If that child is a leaf, insert the value into the child node and finish.
   - Otherwise, descend into the child and repeat from step 1

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

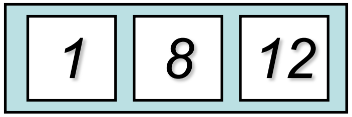
Insert 1



2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

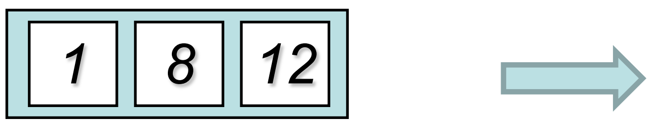
Insert 12,8



2-3-4 Tree Example: Insertion

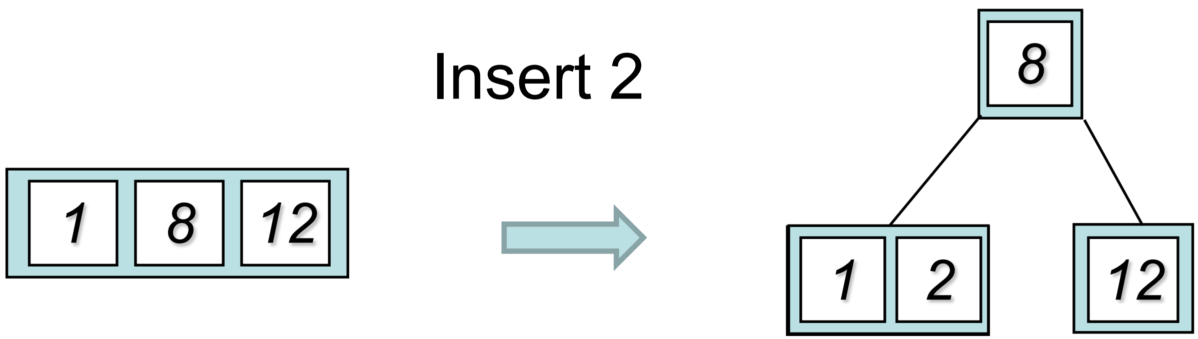
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

Insert 2



2-3-4 Tree Example: Insertion

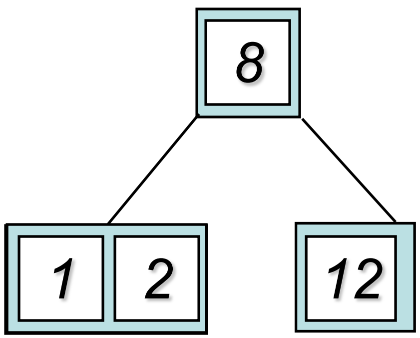
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



4-node splits

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



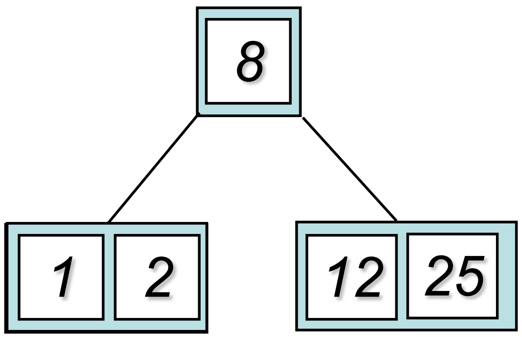
Insert 25

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

2-3-4 Tree Example: Insertion

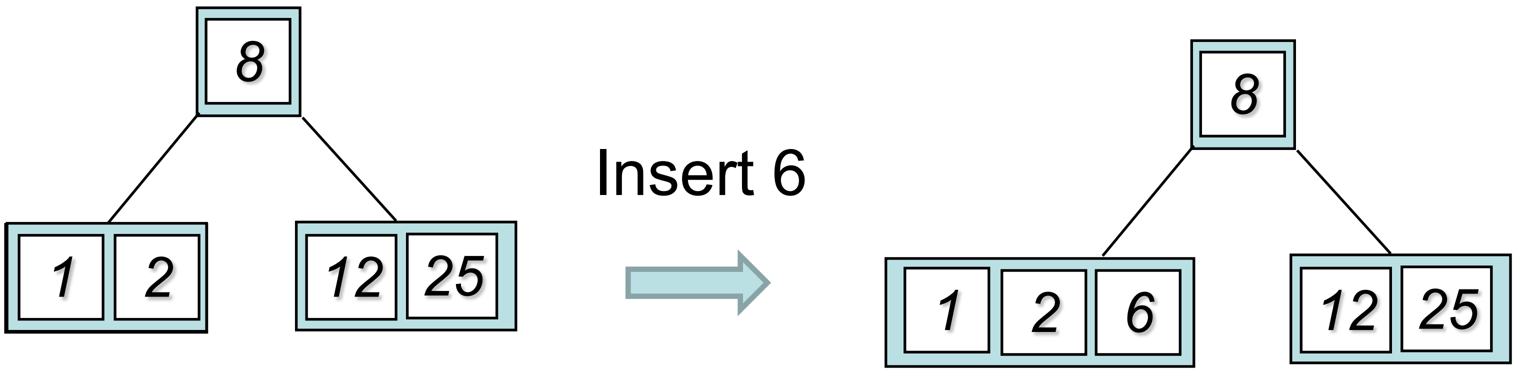
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Insert 6

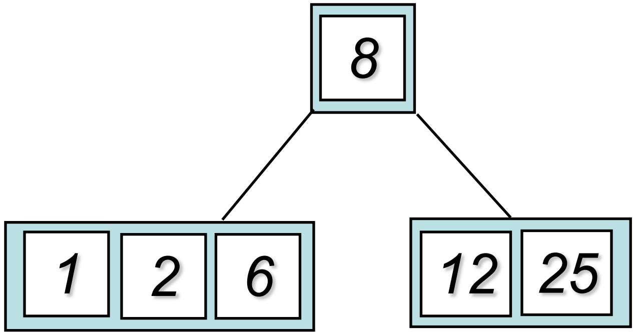
2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



2-3-4 Tree Example: Insertion

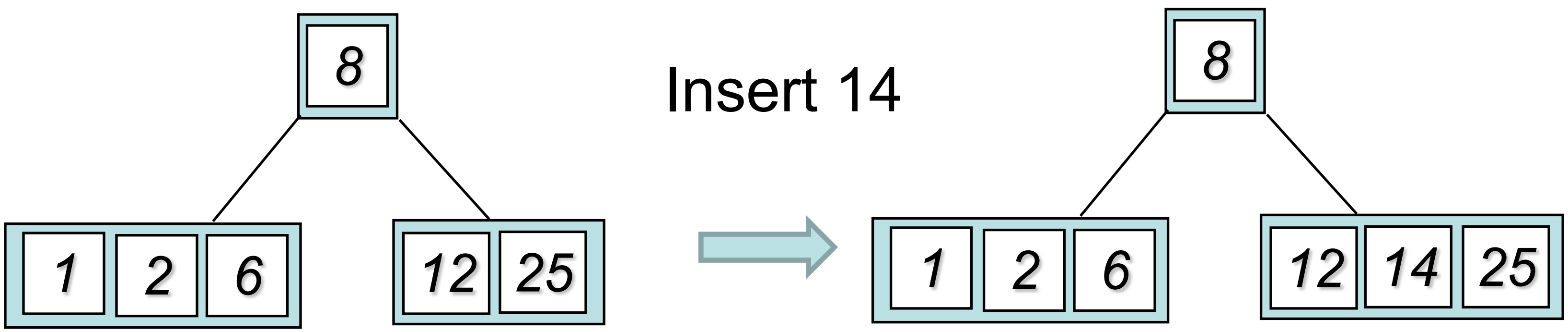
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Insert 14

2-3-4 Tree Example: Insertion

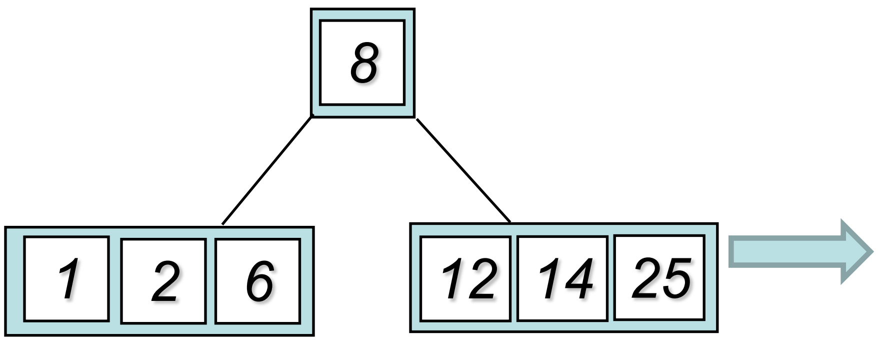
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



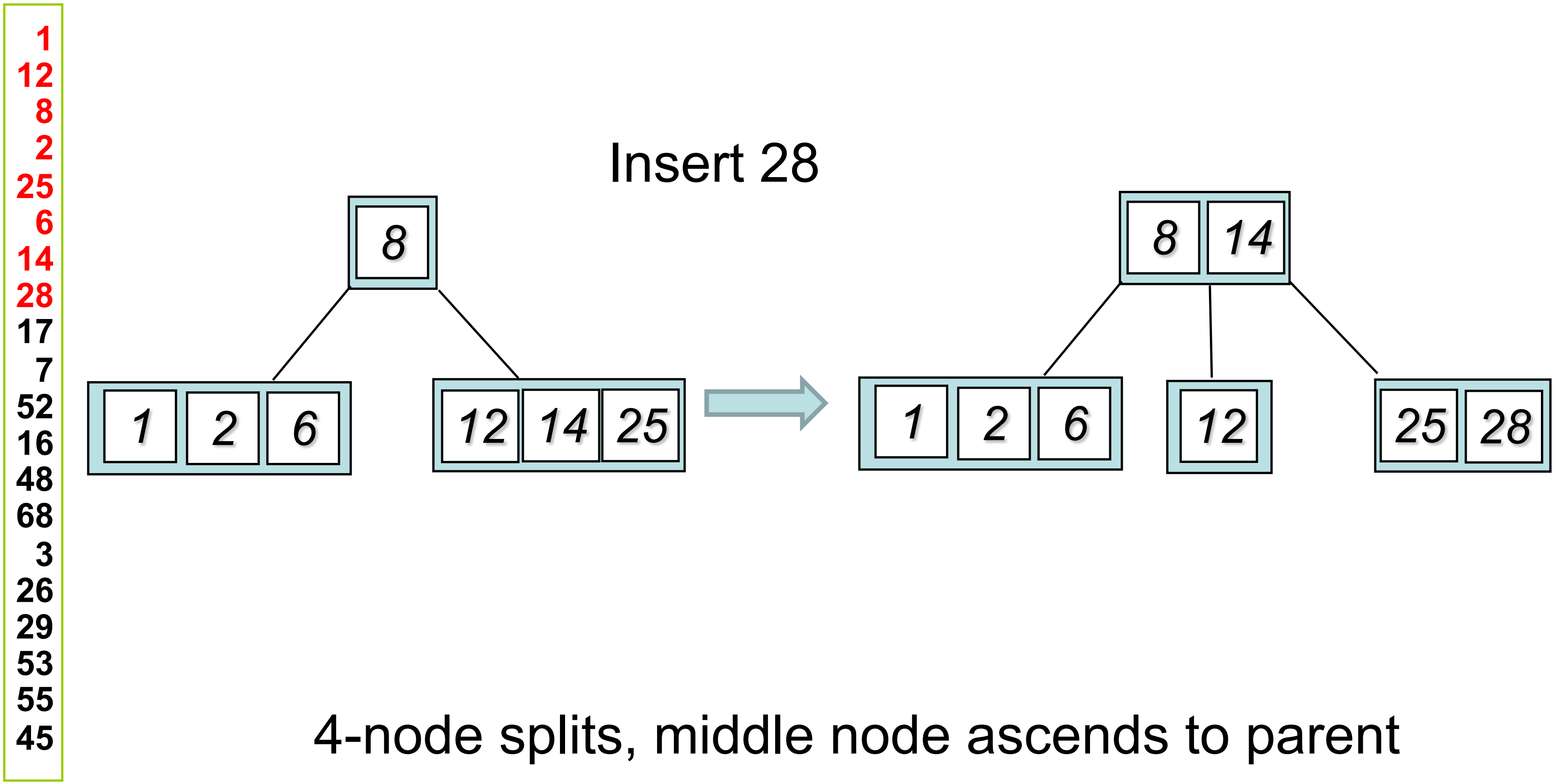
2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

Insert 28



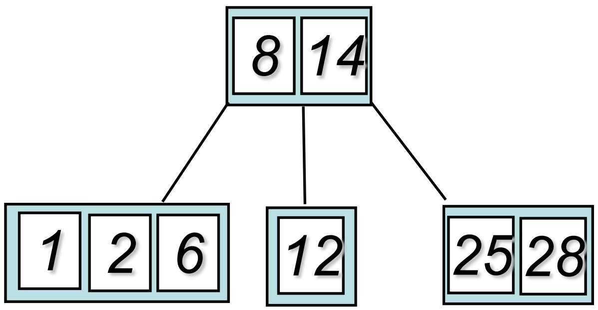
2-3-4 Tree Example: Insertion



4-node splits, middle node ascends to parent

2-3-4 Tree Example: Insertion

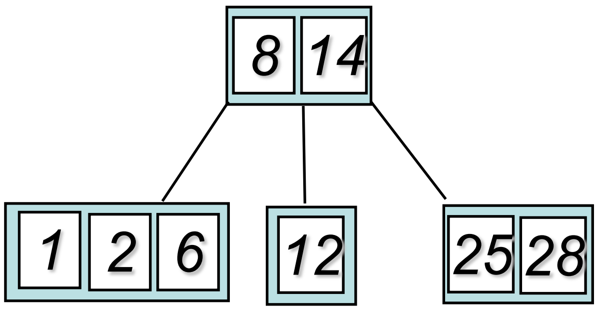
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



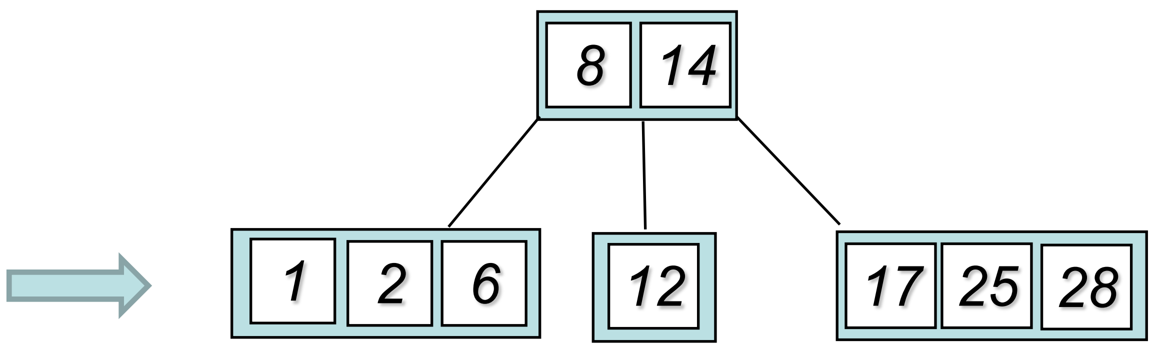
Insert 17

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

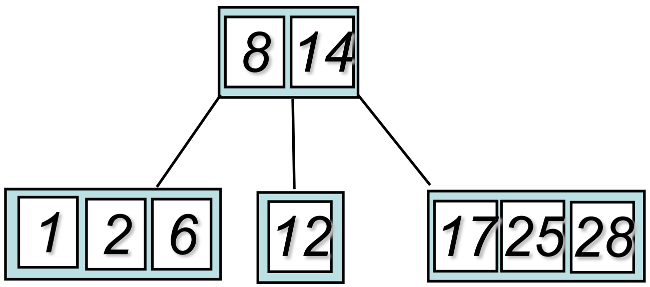


Insert 17



2-3-4 Tree Example: Insertion

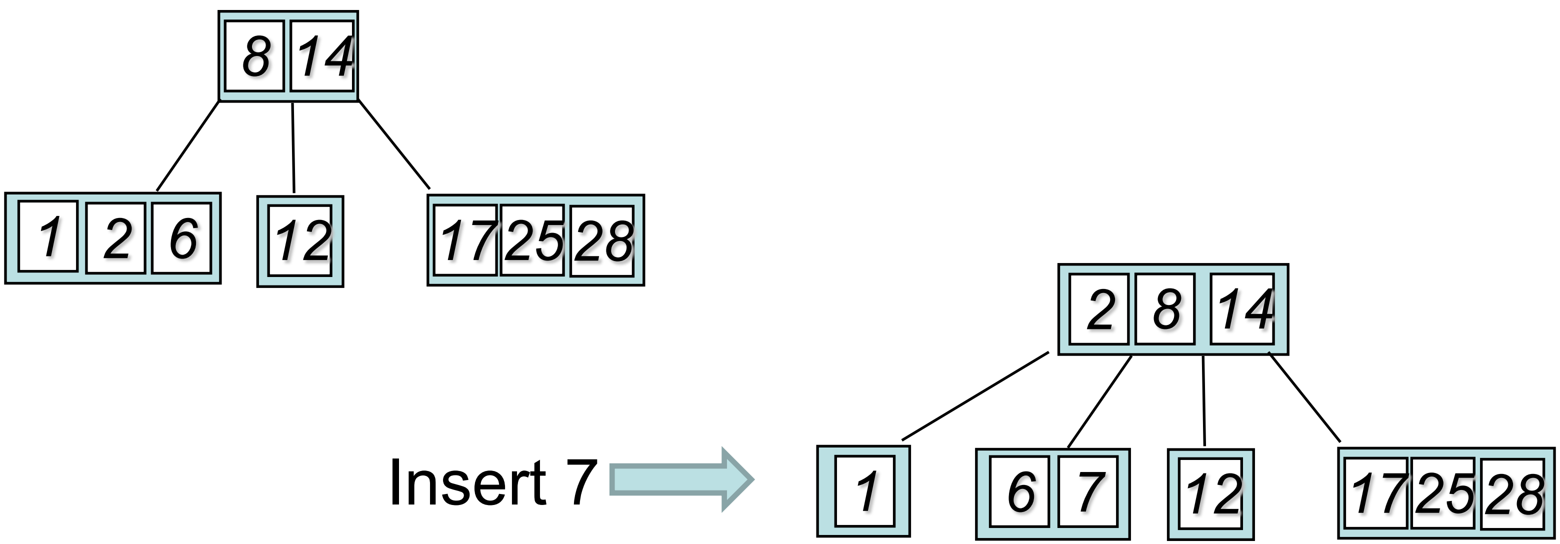
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Insert 7

2-3-4 Tree Example: Insertion

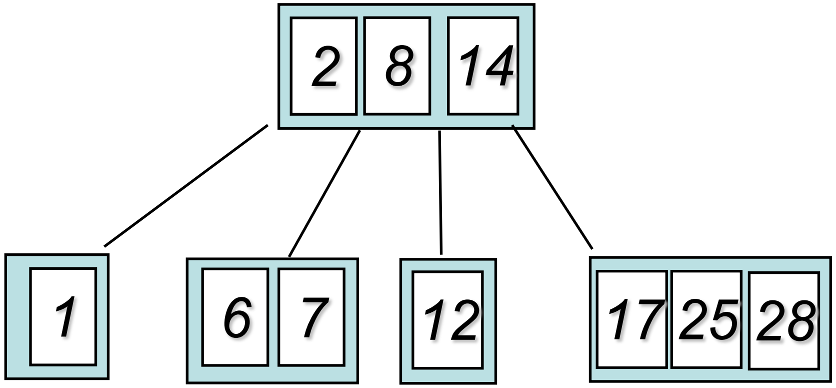
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



4-node splits, middle node ascends to parent

2-3-4 Tree Example: Insertion

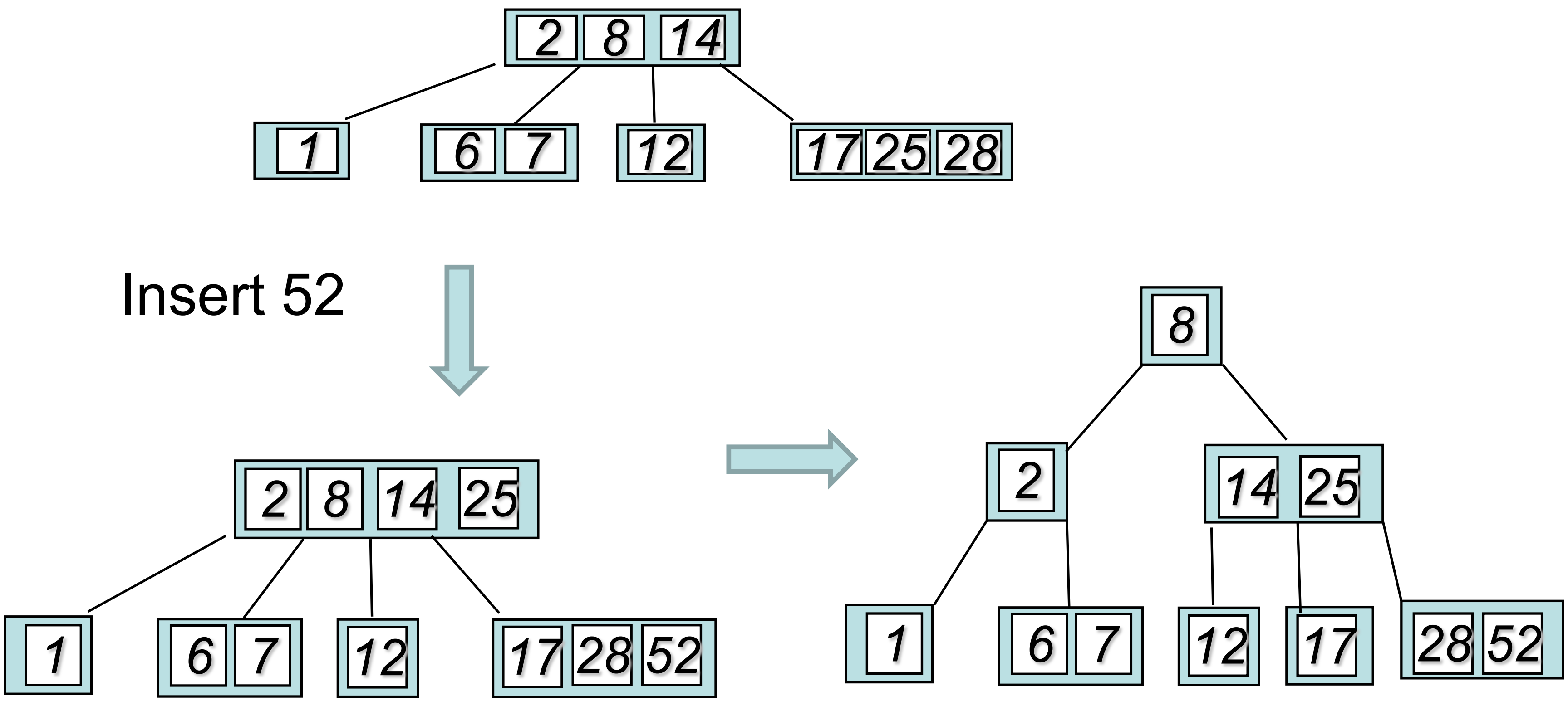
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Insert 52

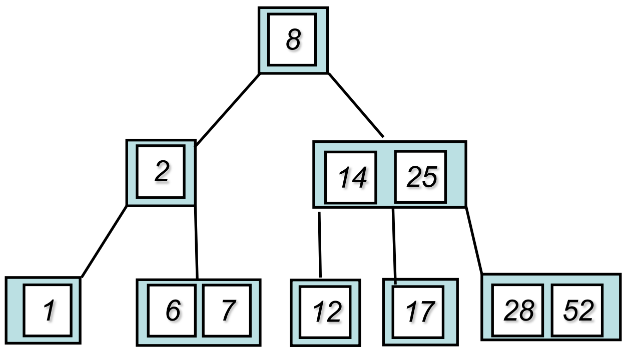
2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

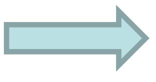


2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

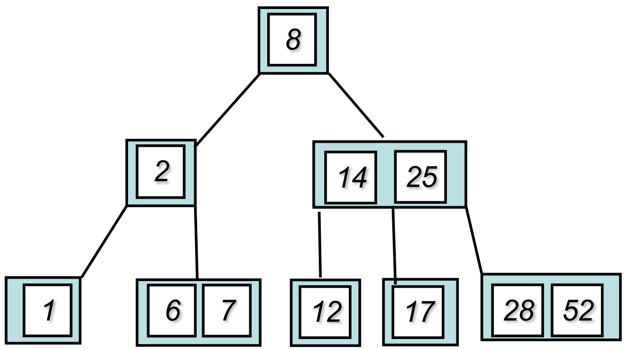


Insert 16

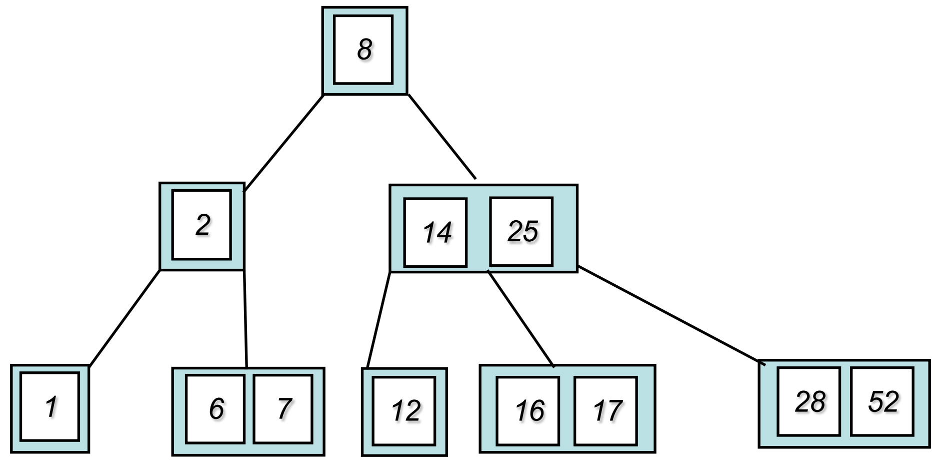


2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

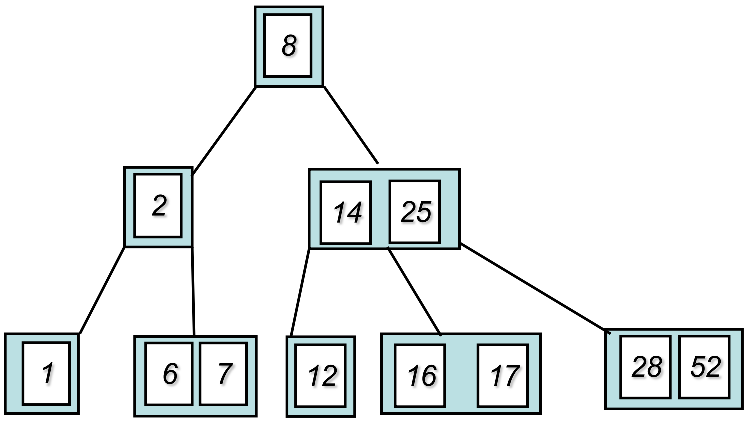


Insert 16



2-3-4 Tree Example: Insertion

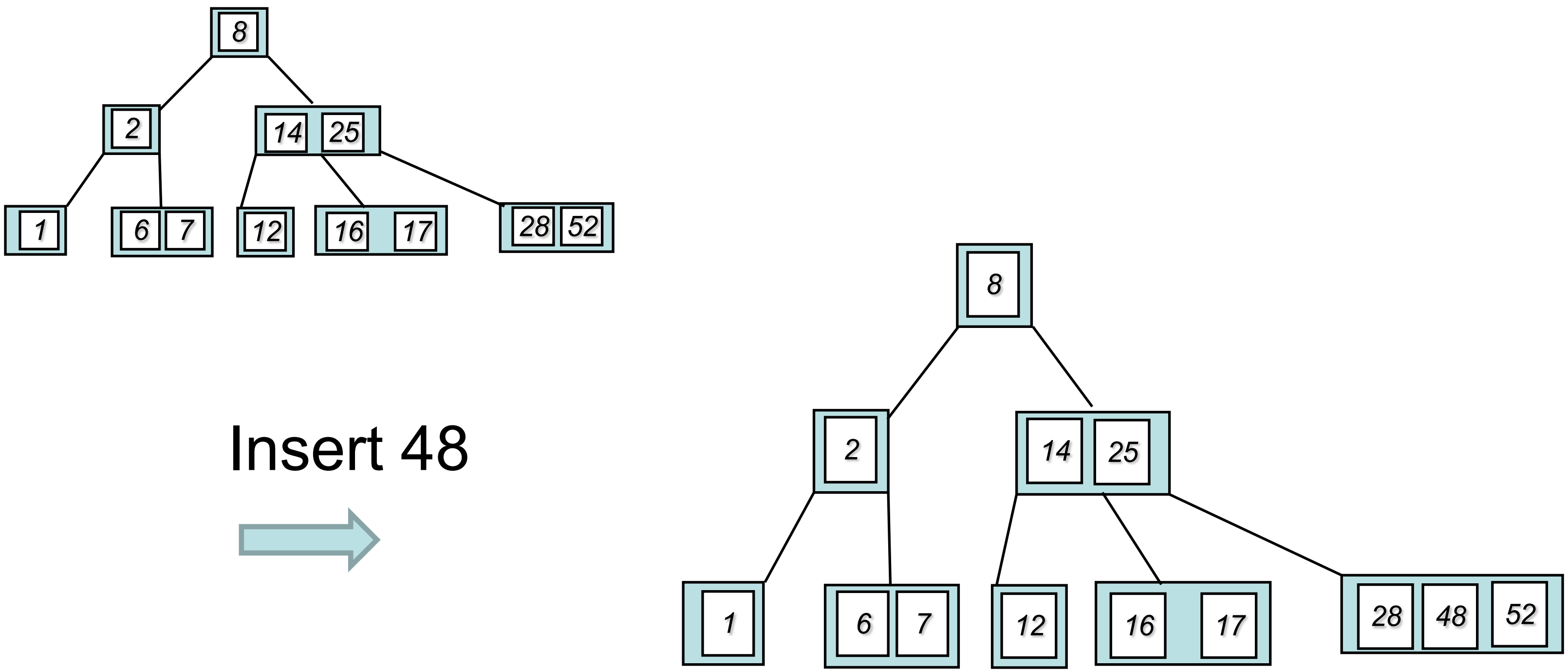
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Insert 48

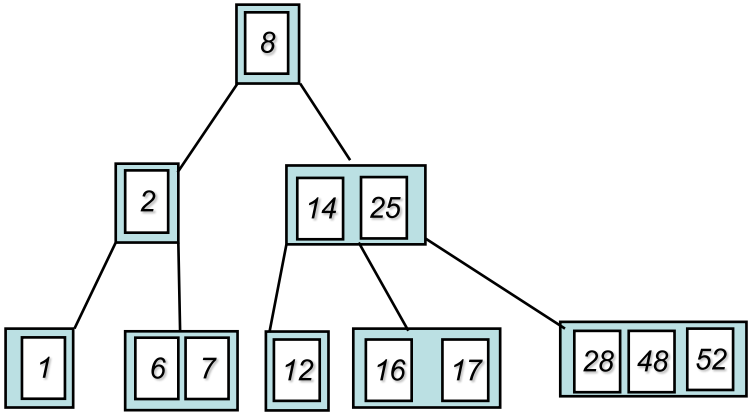
2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



2-3-4 Tree Example: Insertion

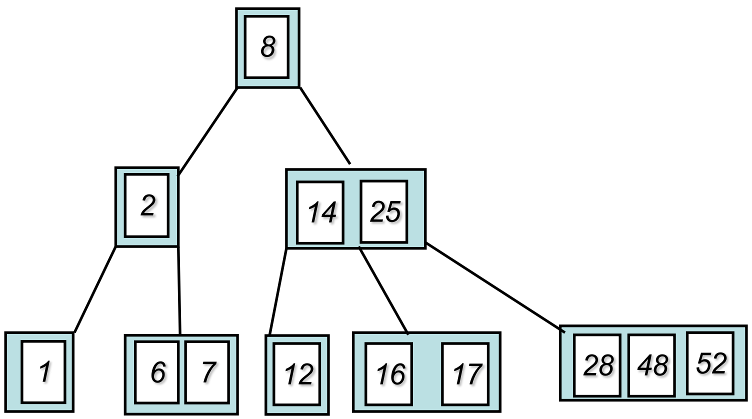
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



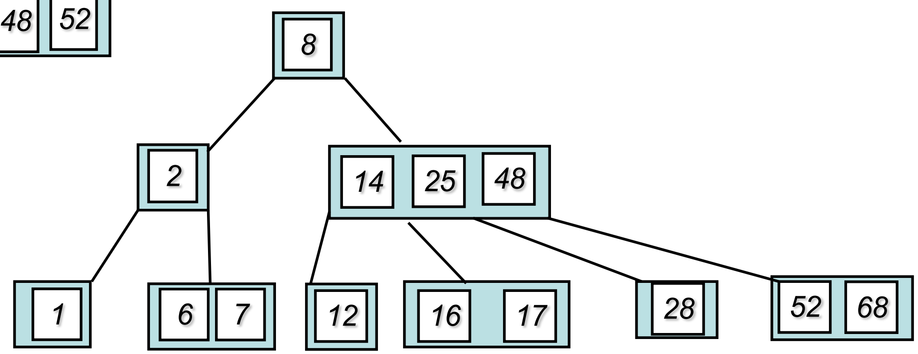
Insert 68

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

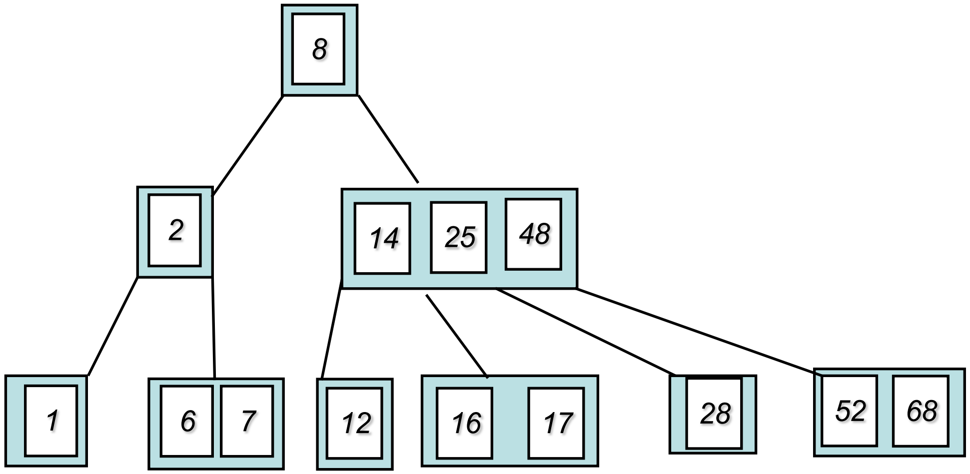


Insert 68



2-3-4 Tree Example: Insertion

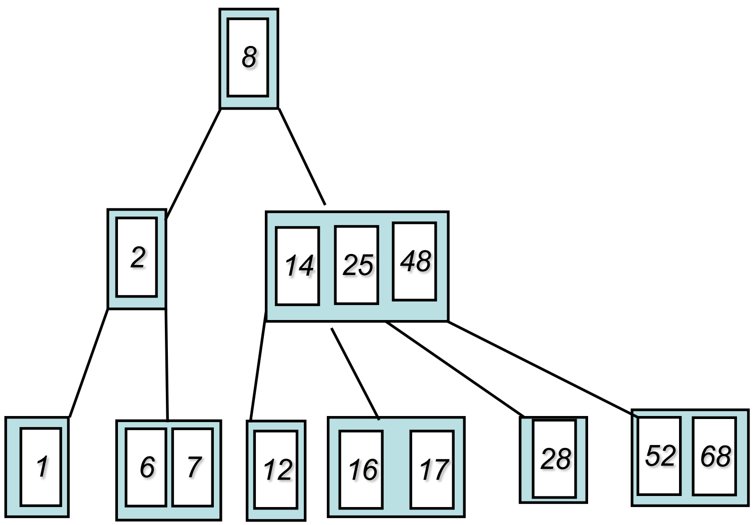
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



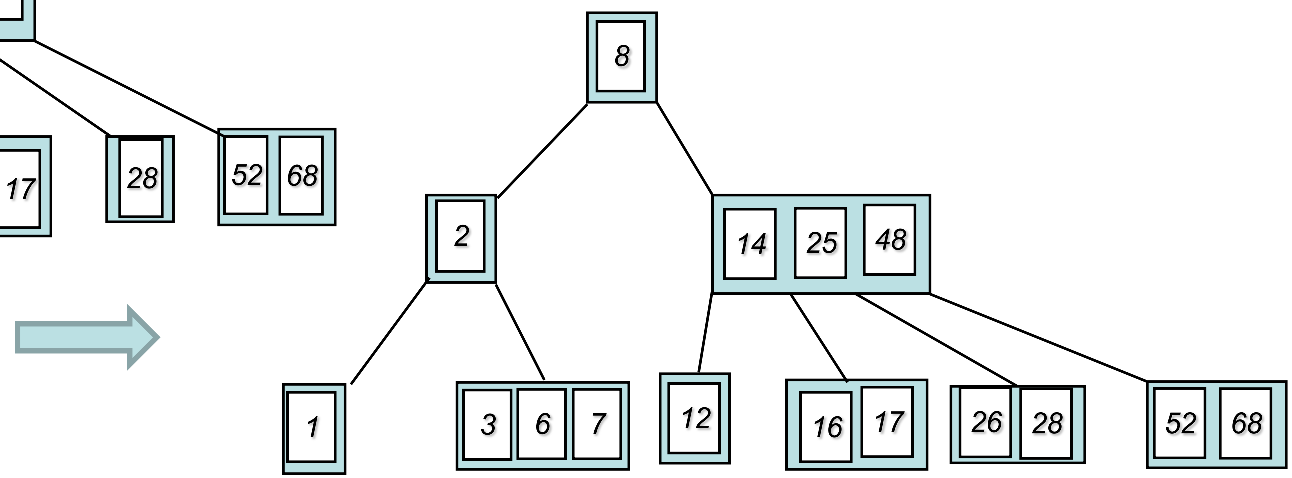
Insert 3, 26

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

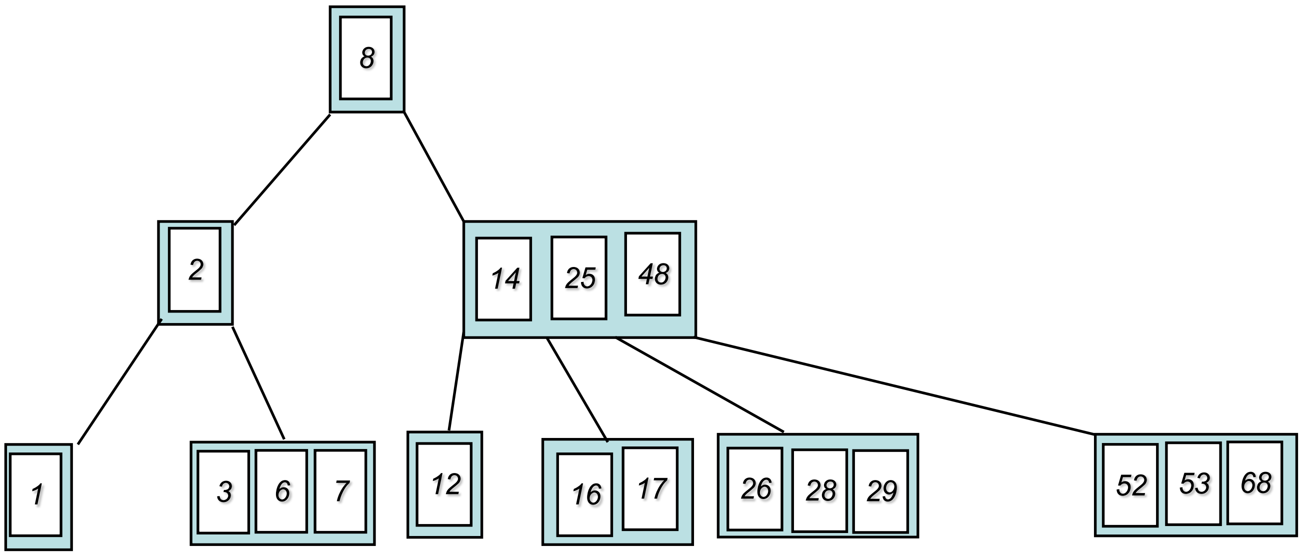


Insert 3, 26



2-3-4 Tree Example: Insertion

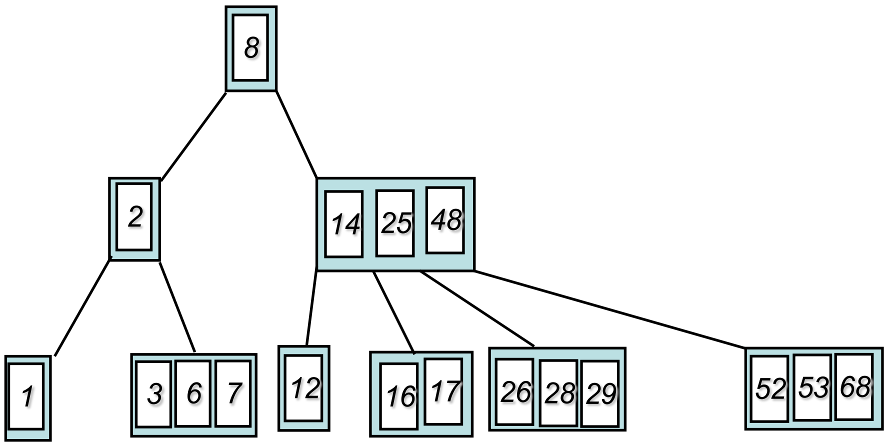
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



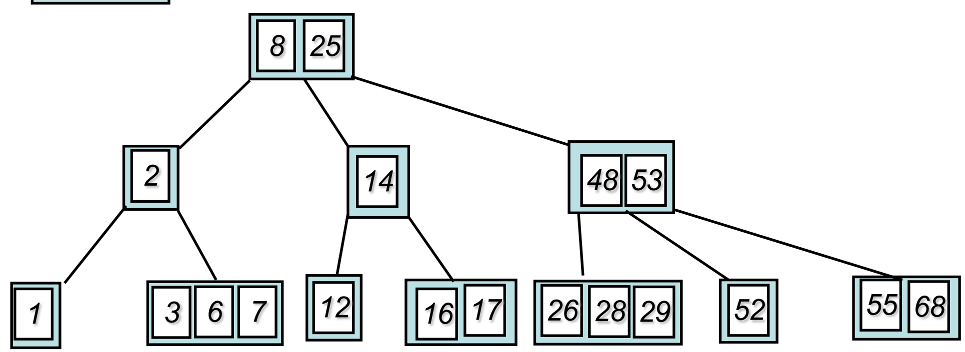
Insert 55

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

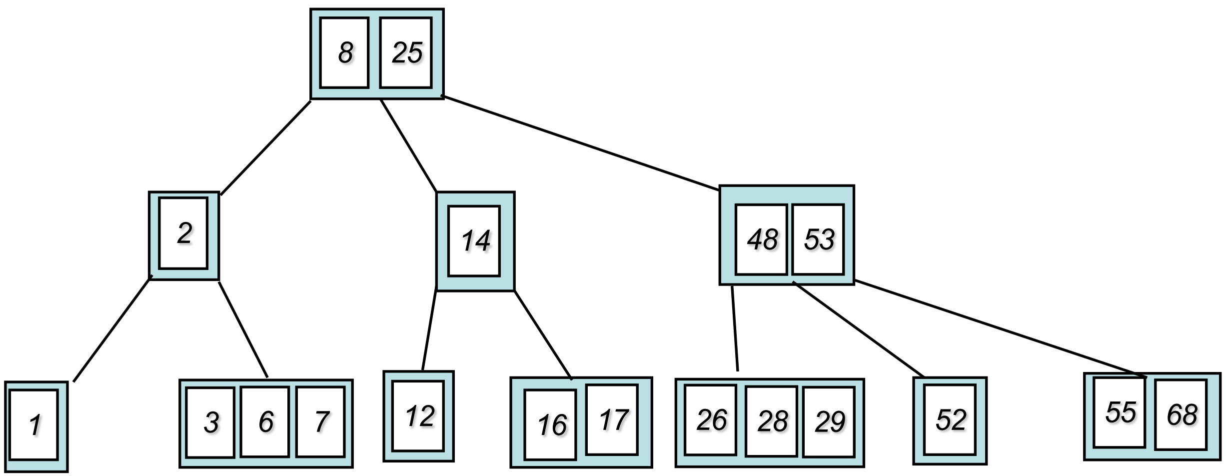


Insert 55



2-3-4 Tree Example: Insertion

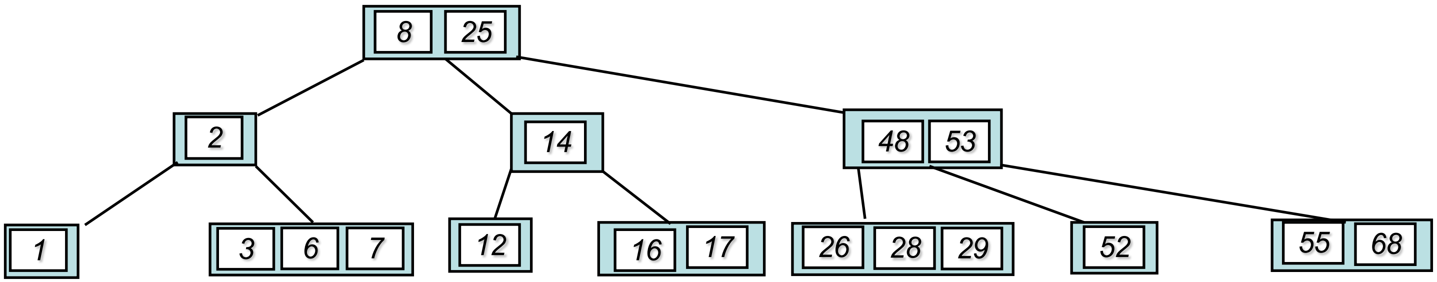
1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



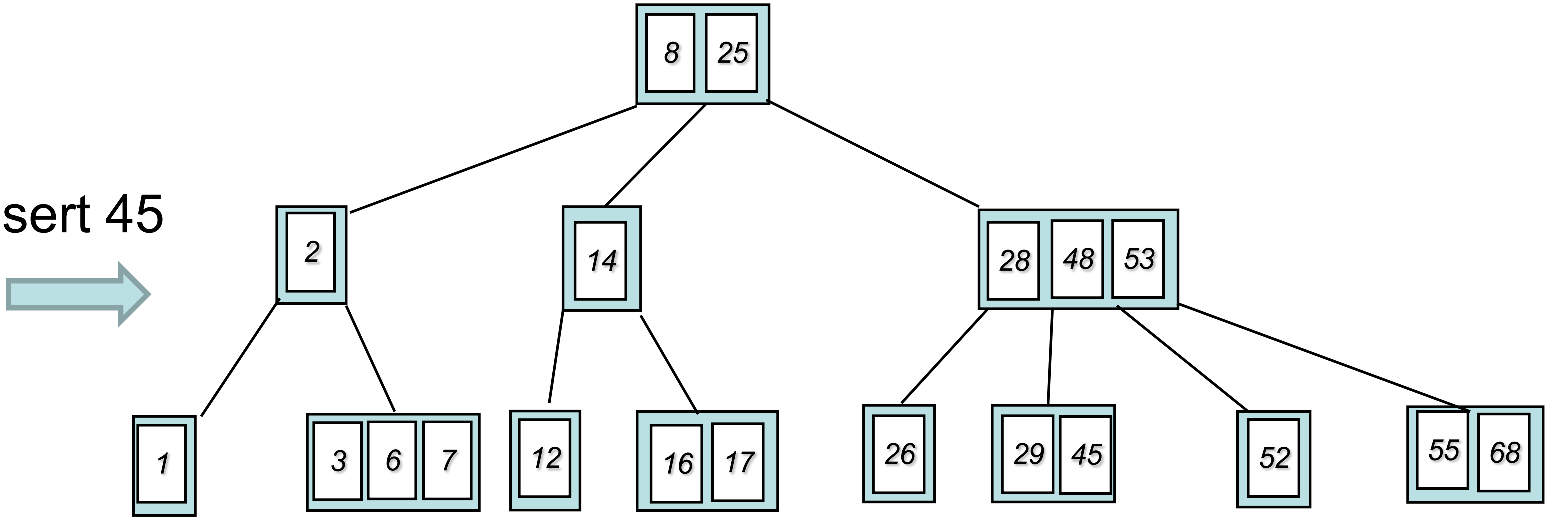
Insert 45

2-3-4 Tree Example: Insertion

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

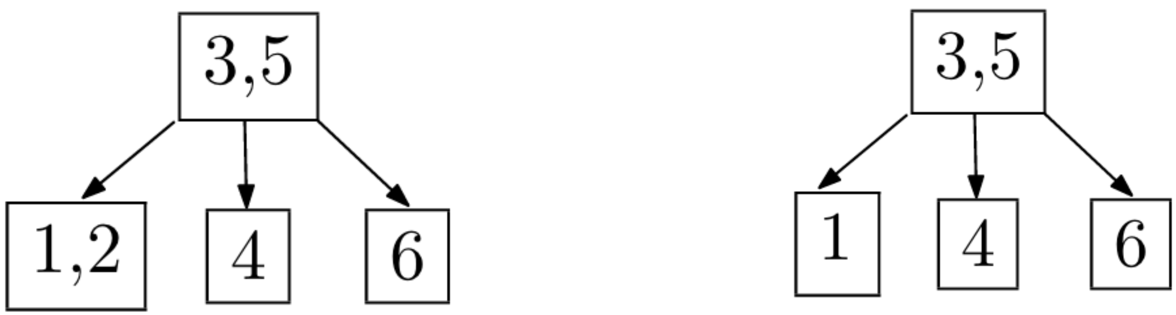


Insert 45



2-3-4 Tree: Delete

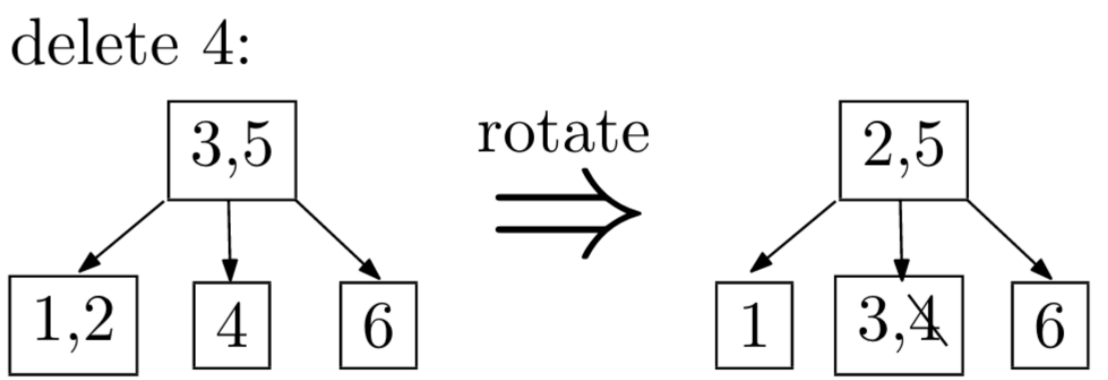
- Leaf:
  - Just delete the key
  - Make sure that a leaf is not empty after deleting a key



Delete 2

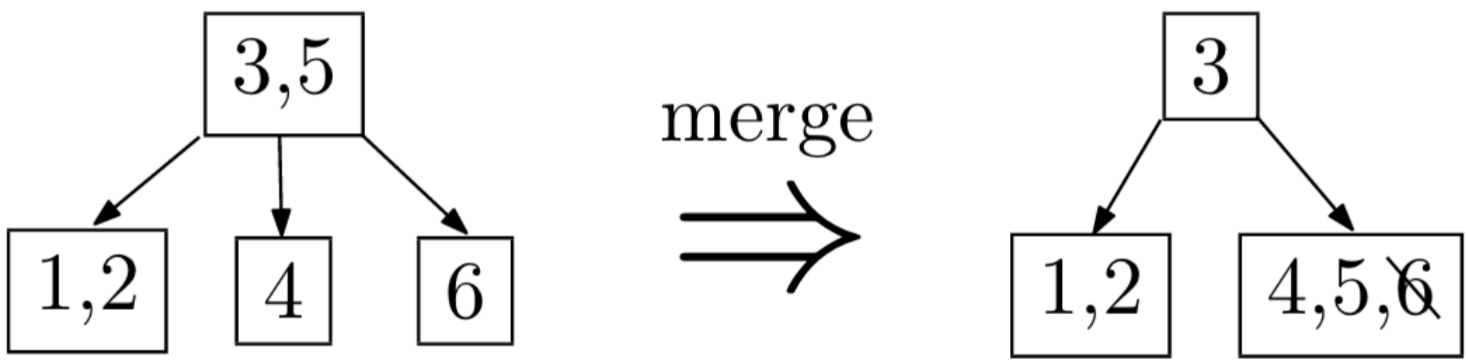
2-3-4 Tree: Delete

- Leaf:
  - When key deletion would create an empty leaf, borrow a key from leaf 's immediate siblings (i.e. to the left and then right).



2-3-4 Tree: Delete

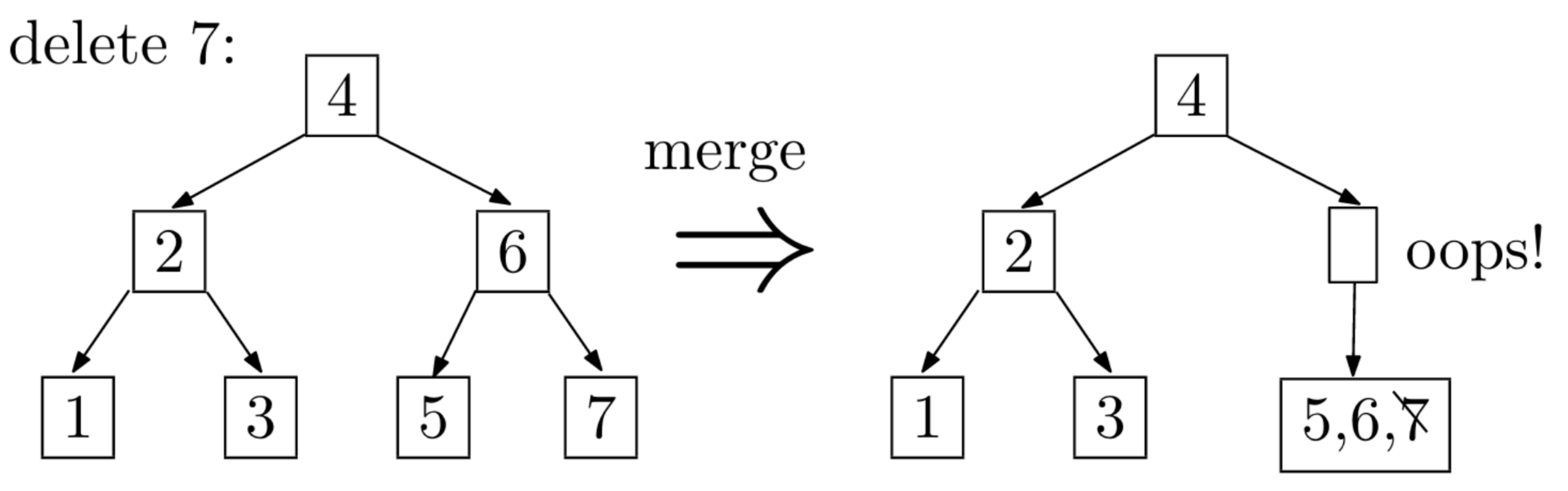
- Leaf:
  - If siblings are 2-nodes (no immediate sibling from which to borrow a key), steal a key from our parent by doing the opposite of a split.



Delete 6

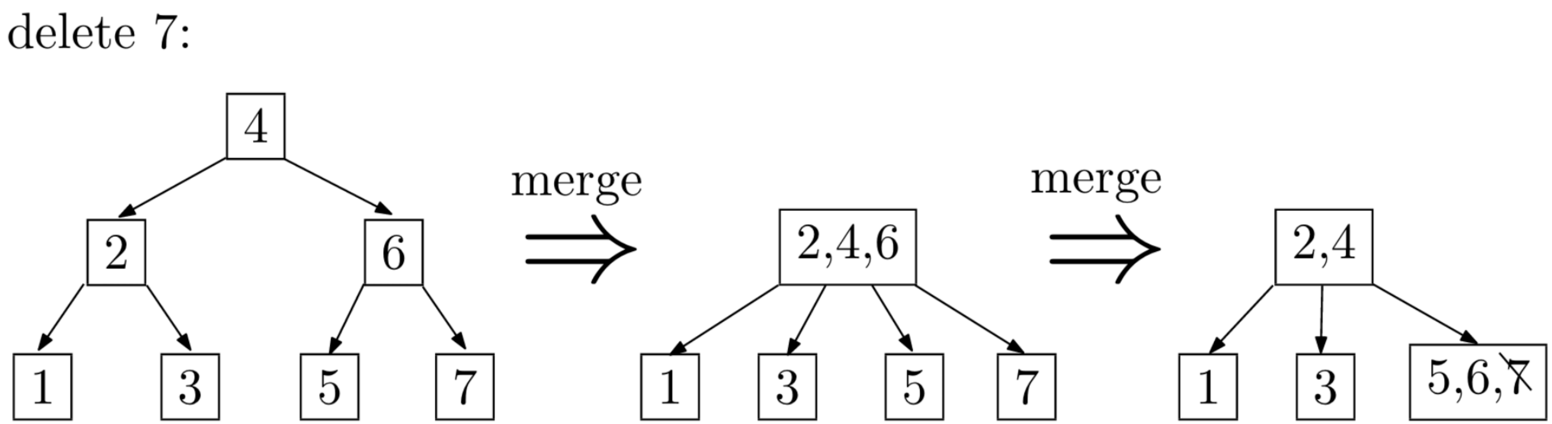
2-3-4 Tree: Delete

- What if parent is a 2-node (one key)?



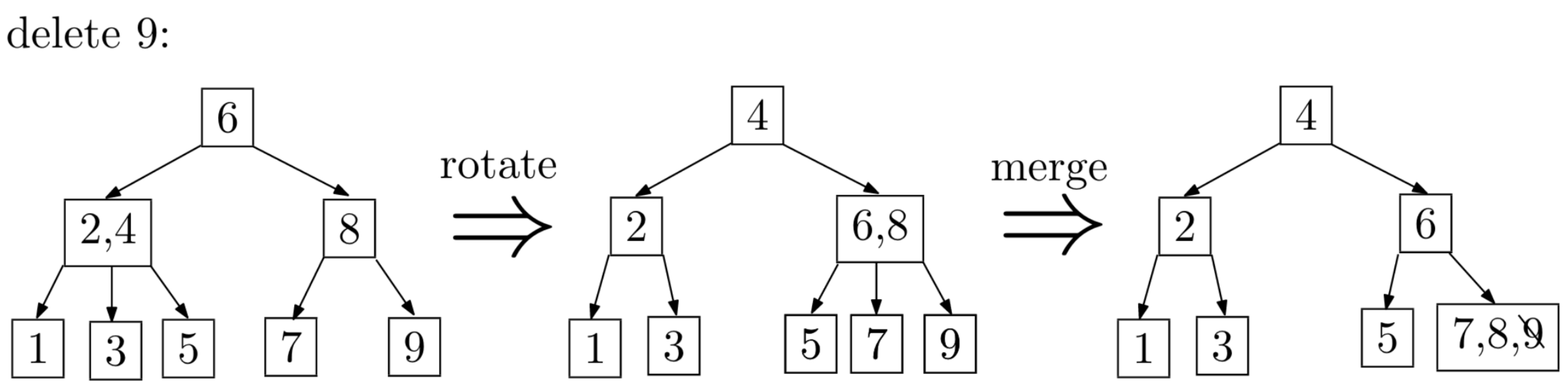
2-3-4 Tree: Delete

- What if parent is a 2-node (one key)?
  - Steal from siblings (parent's)
  - Merge



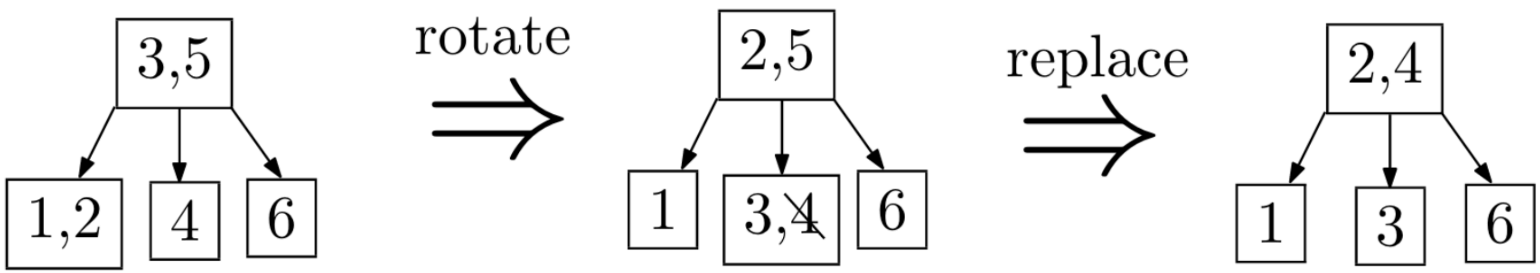
2-3-4 Tree: Delete

- What if parent is a 2-node (one key)?
  - Steal from siblings (parent's)
  - Merge



2-3-4 Tree: Delete

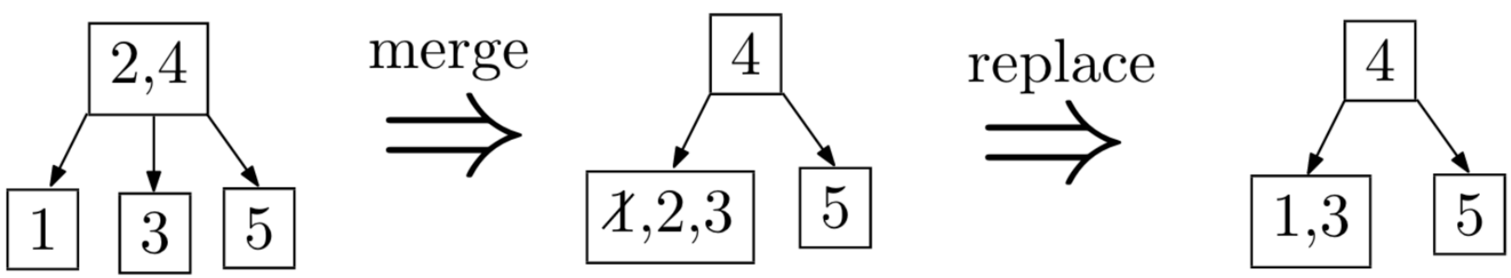
- Internal Node:
  - Delete the predecessor, and swap it with the node to be deleted.



Delete 5: first delete 4, then swap 4 for 5.

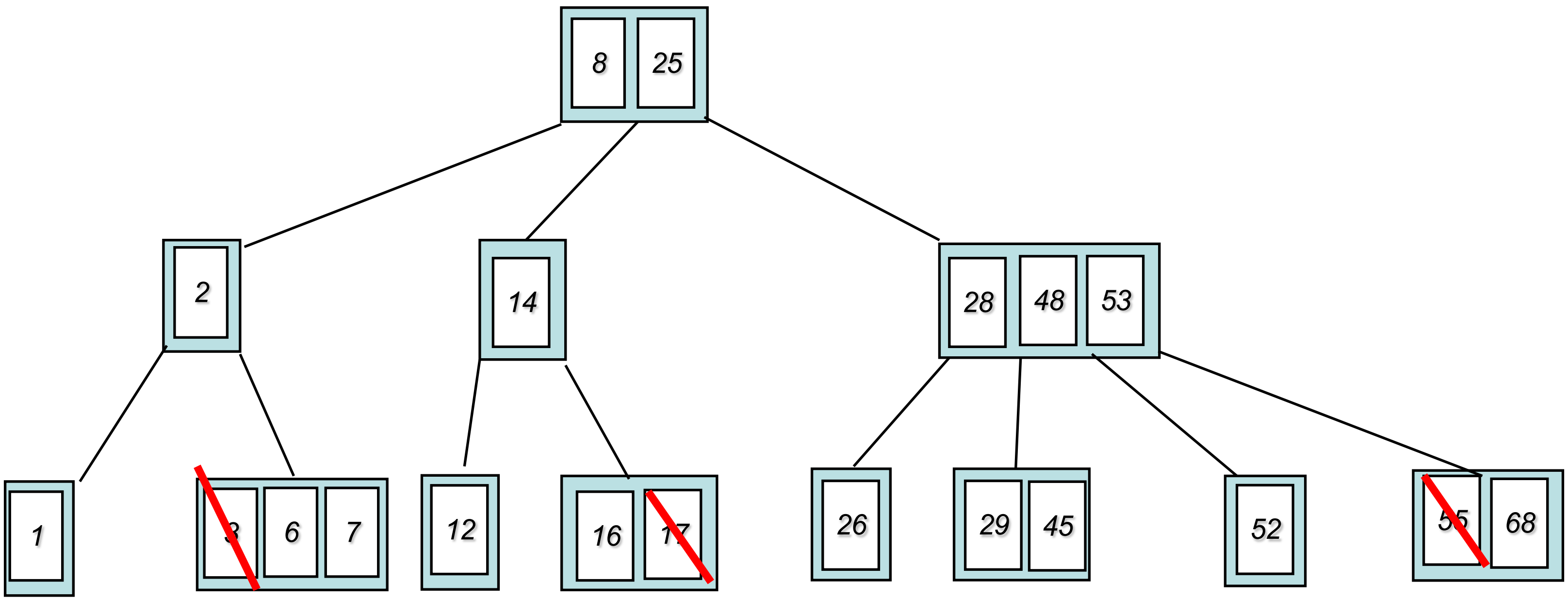
2-3-4 Tree: Delete

- Internal Node:
  - Delete the predecessor, and swap it with the node to be deleted.
  - **Key to delete may move.**



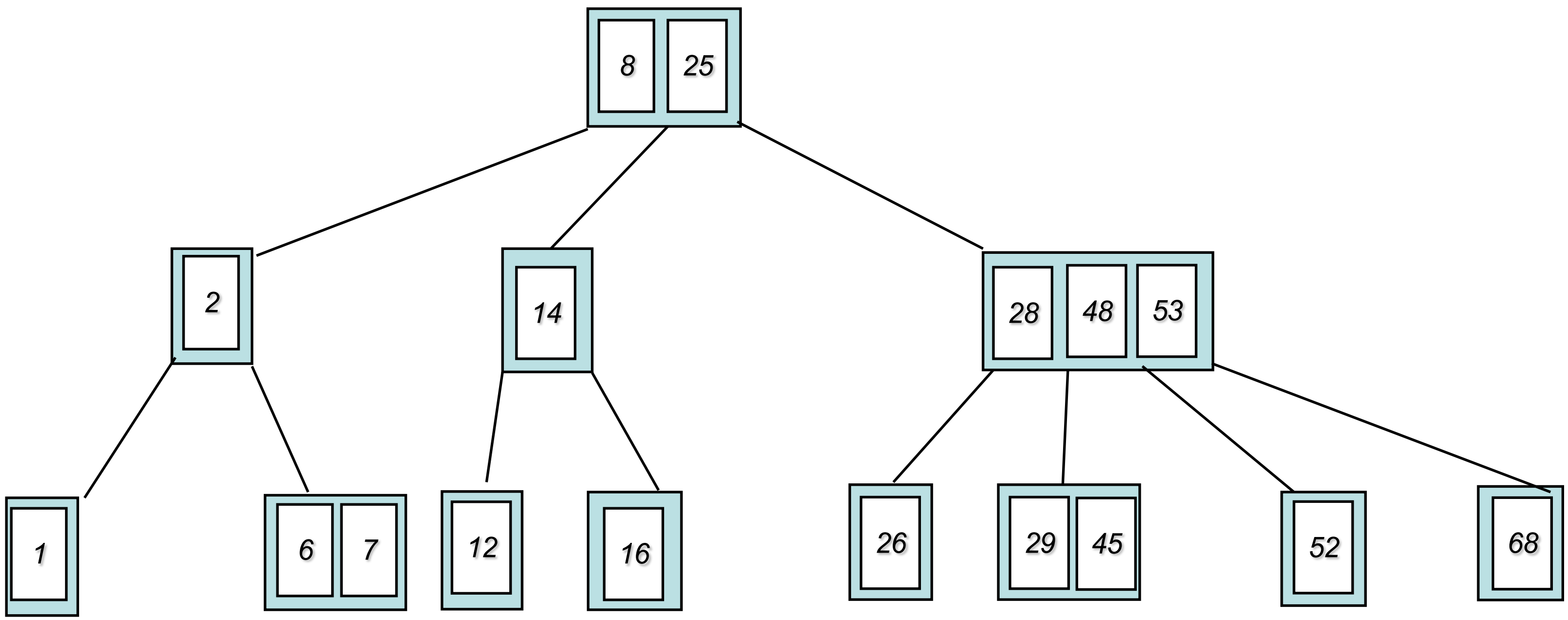
Delete 2: first delete 1, then swap 1 for 2.

2-3-4 Tree Example: Delete



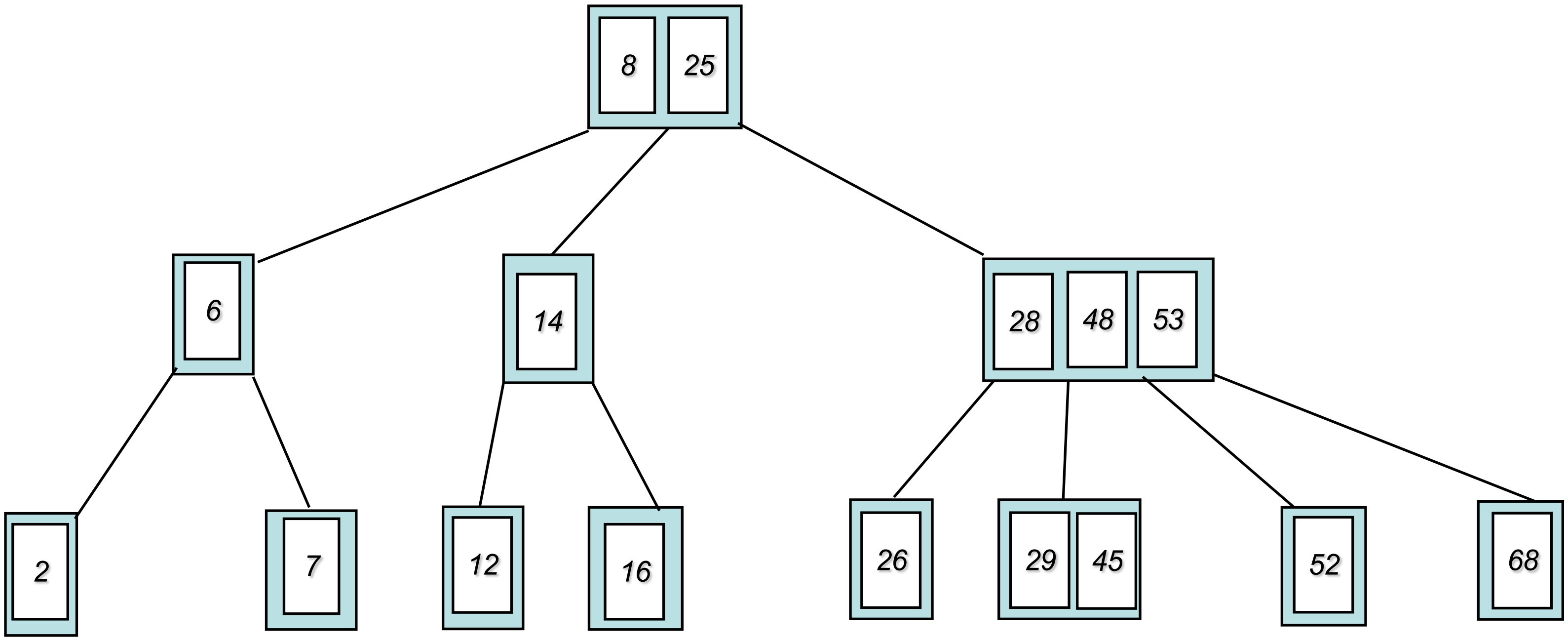
Delete 3,17,55

2-3-4 Tree Example: Delete



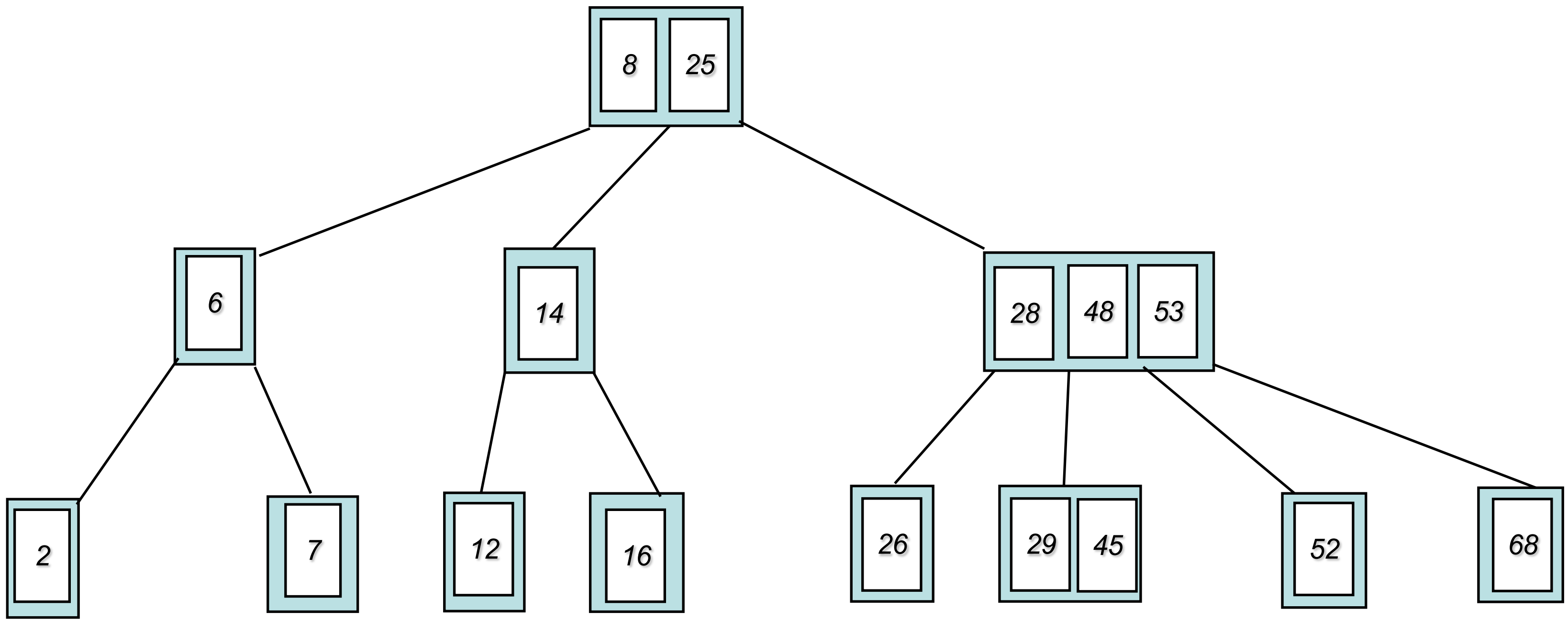
Delete 1: borrow from siblings (rotate)

2-3-4 Tree Example: Delete



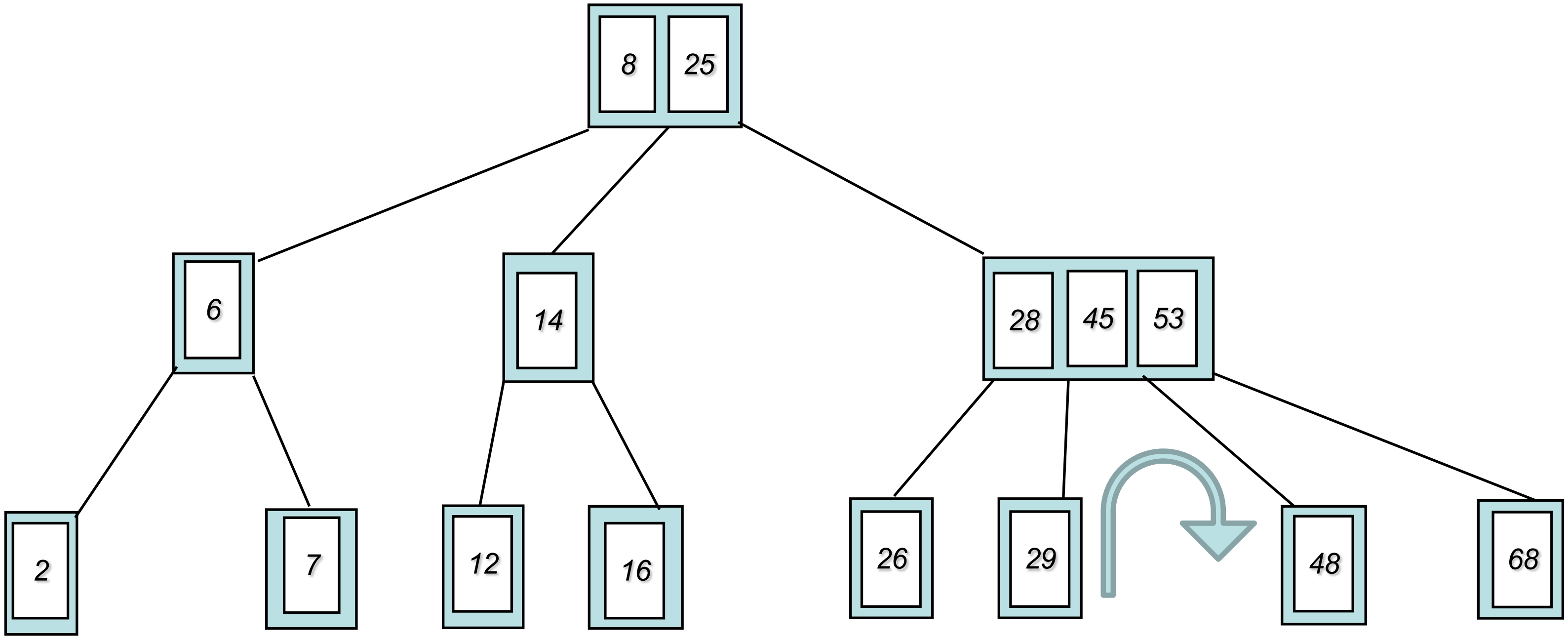
Delete 1

2-3-4 Tree Example: Delete



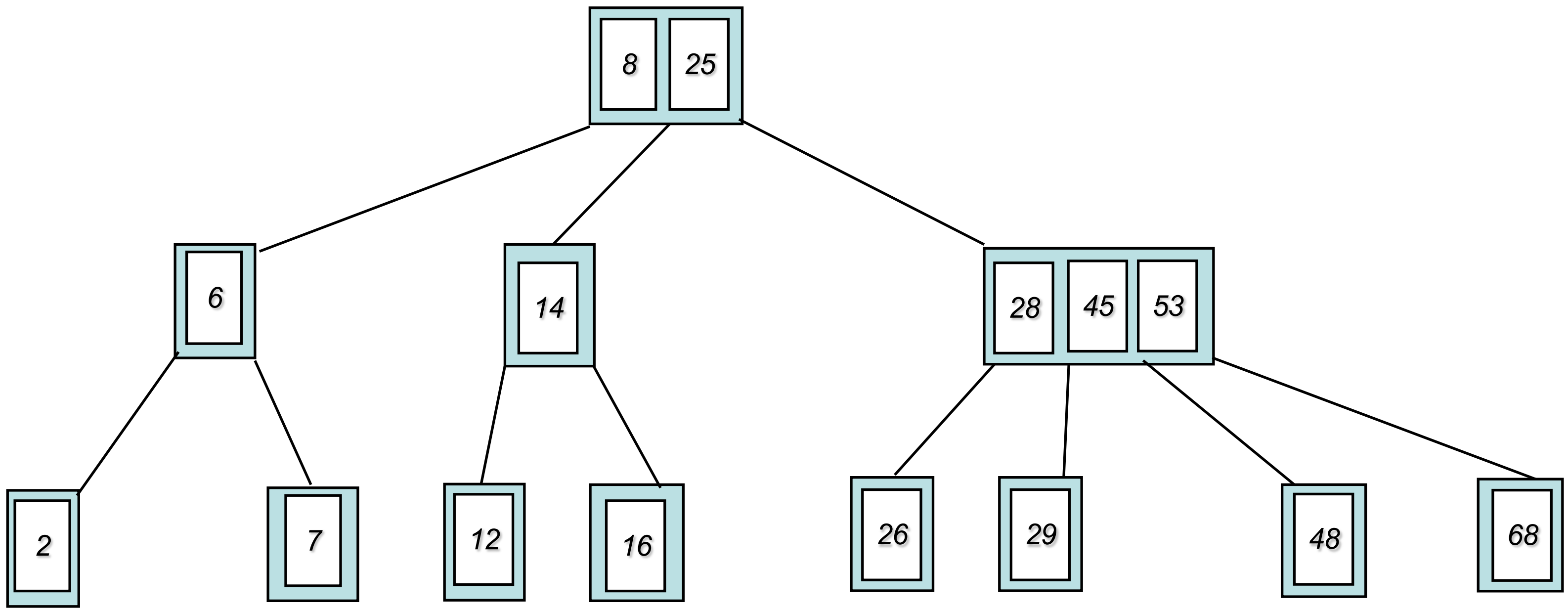
Delete 52: borrow from sibling

2-3-4 Tree Example: Delete



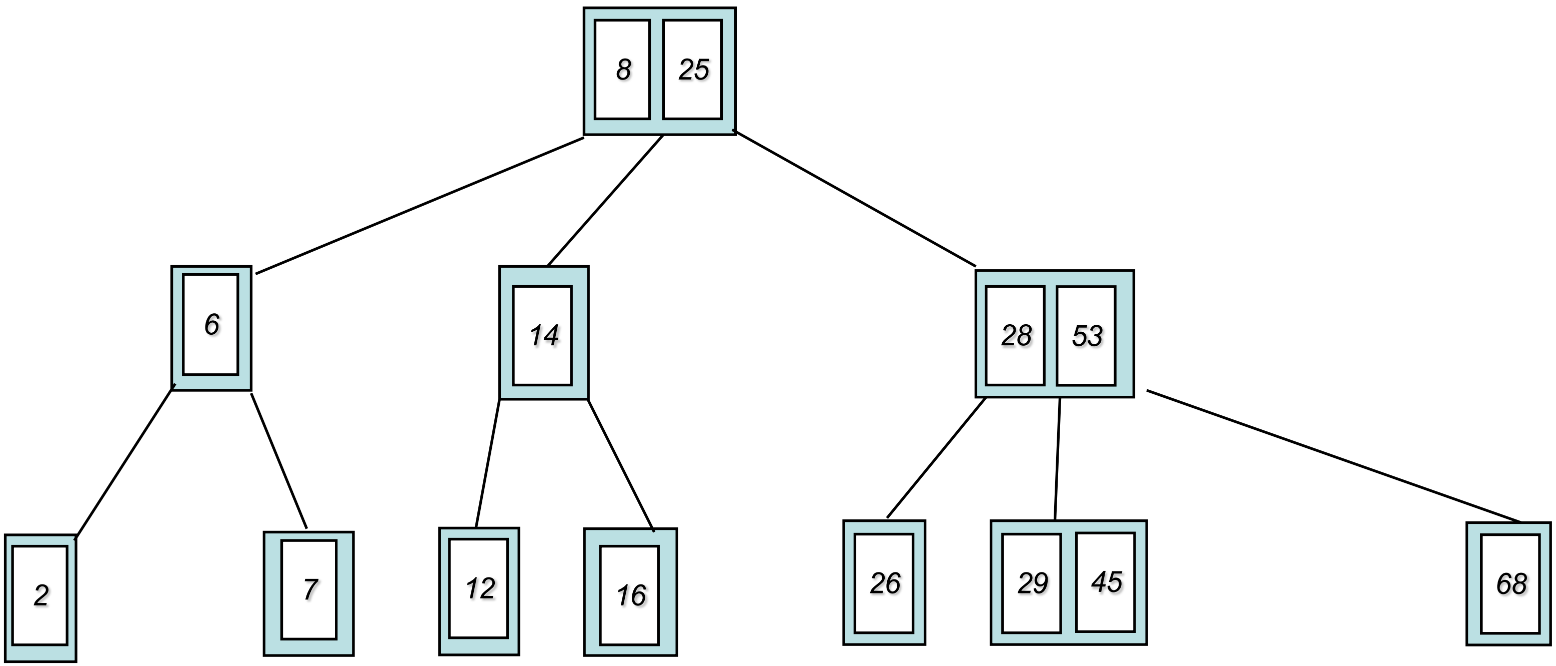
Delete 52: borrow from sibling

2-3-4 Tree Example: Delete



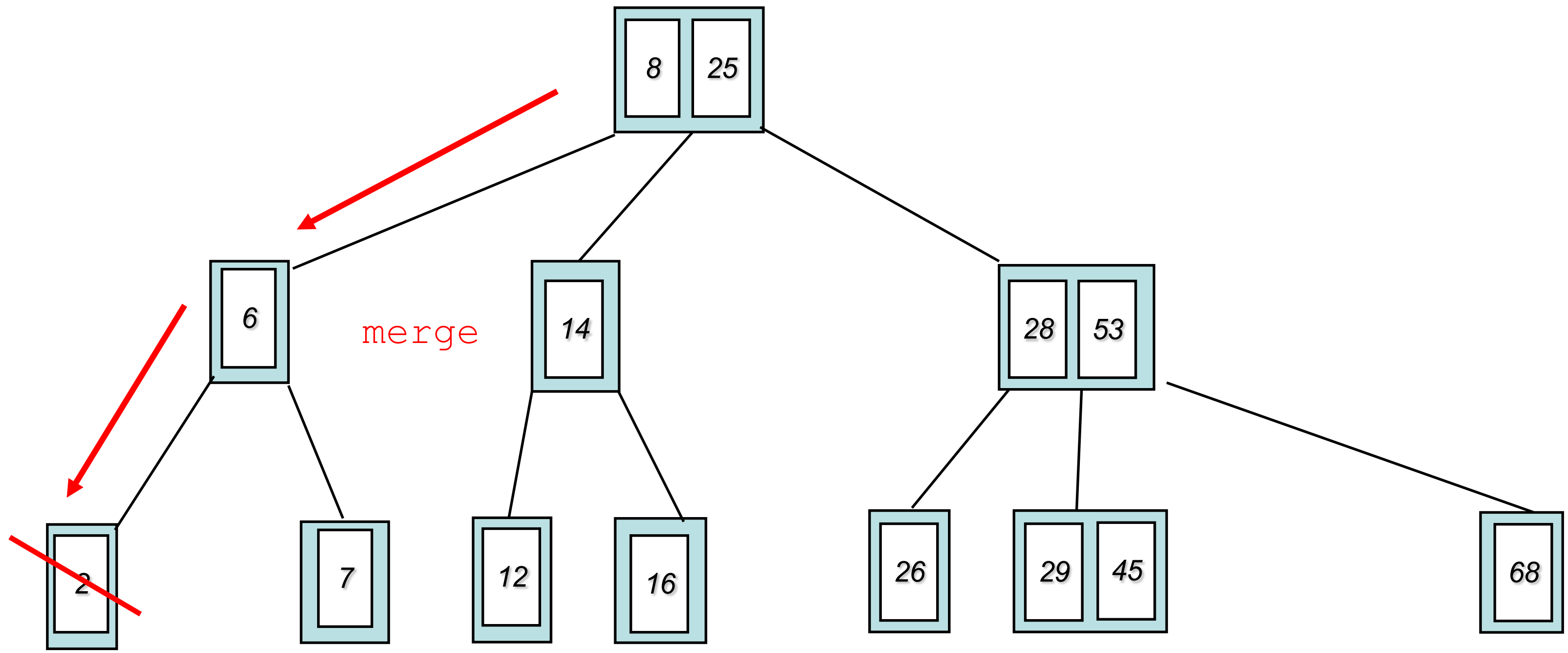
Delete 48: borrow from parent

2-3-4 Tree Example: Delete



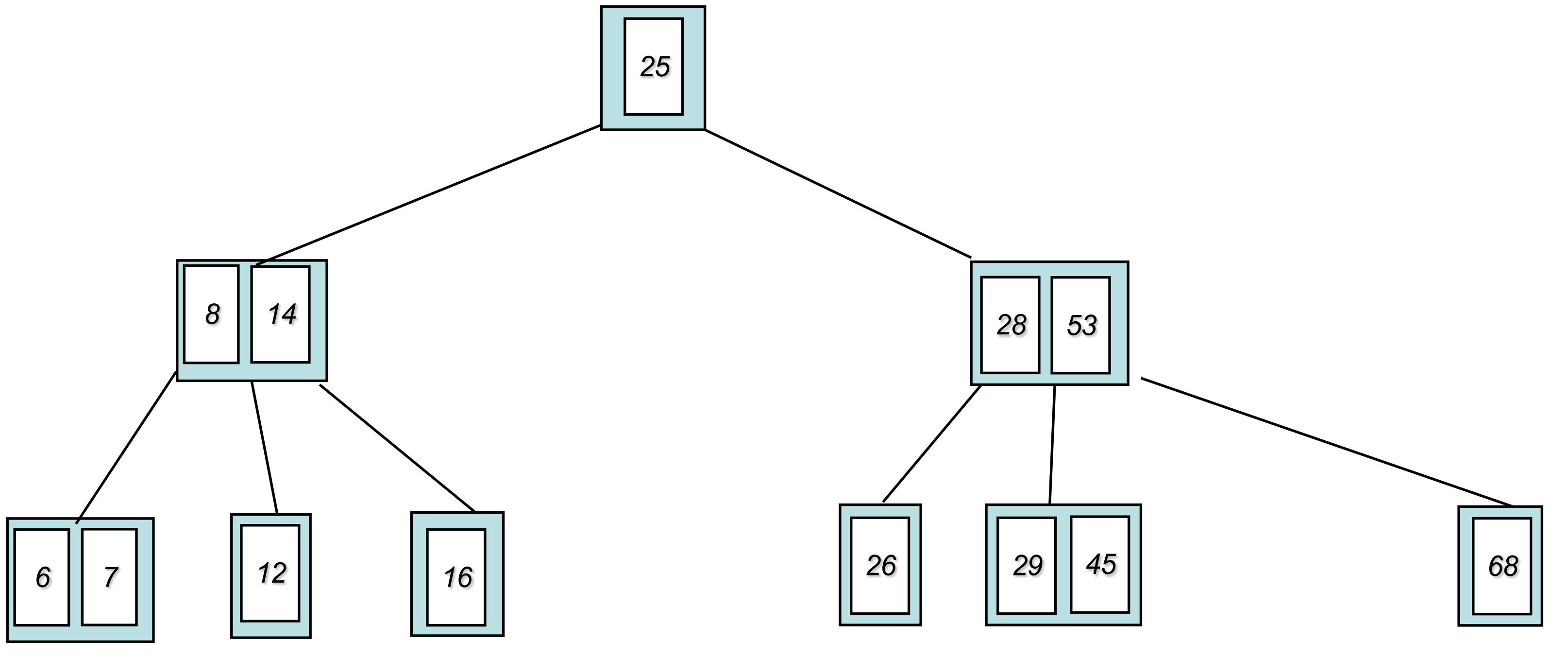
Delete 48: borrow from parent

2-3-4 Tree Example: Delete



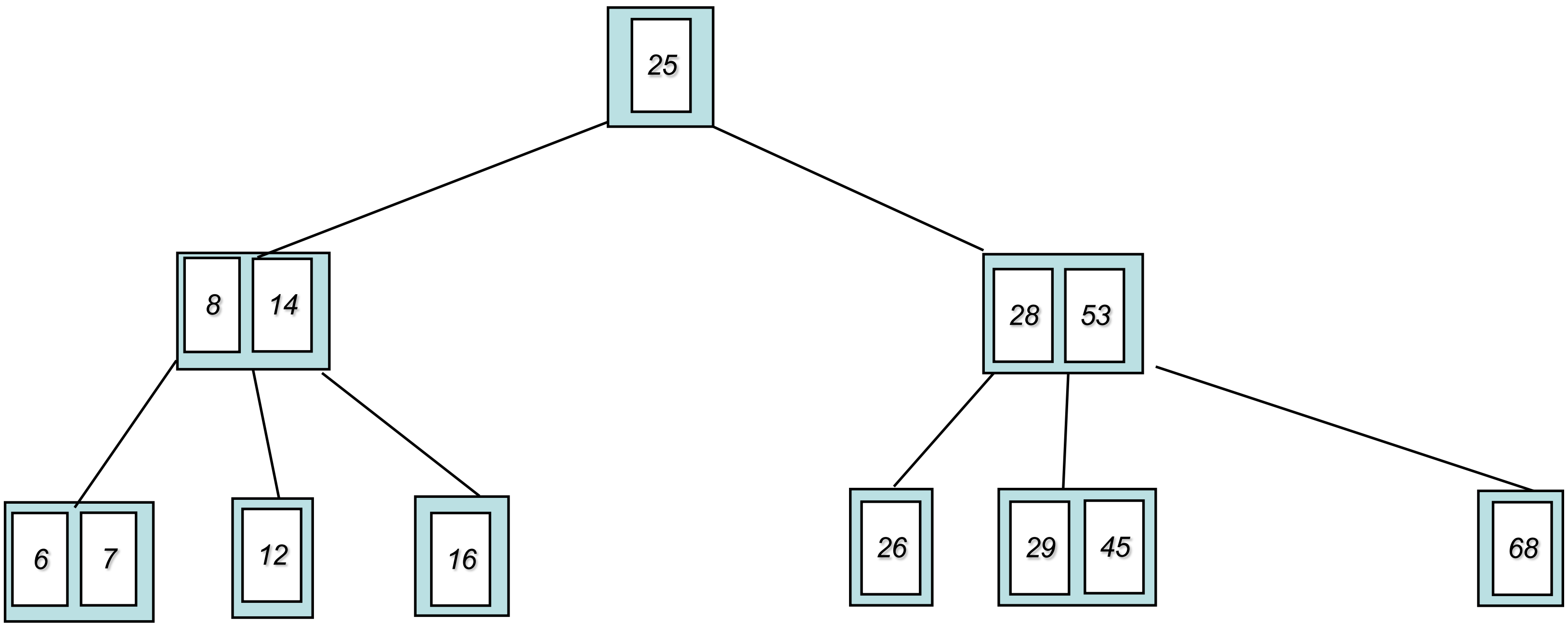
Delete 2: borrow from parent, and parent

2-3-4 Tree Example: Delete



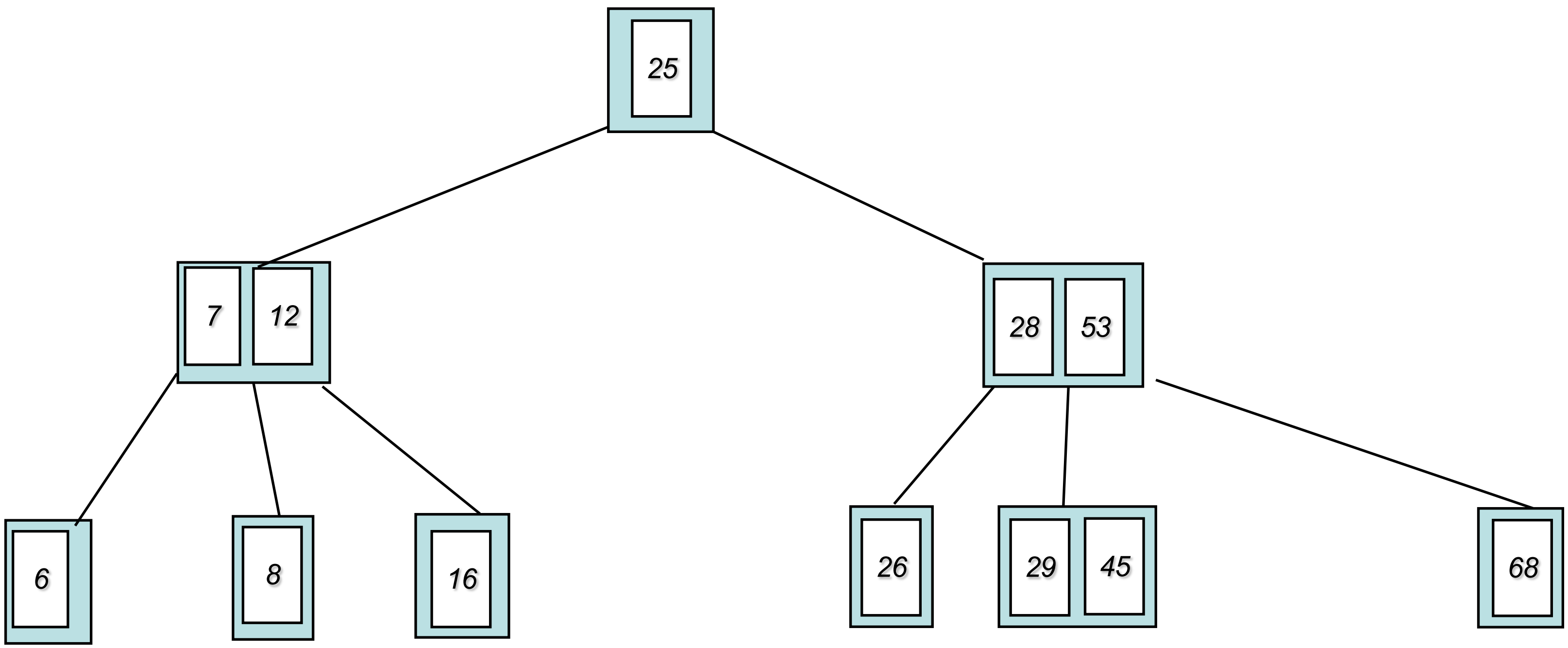
Delete 2: borrow from parent, and parent

2-3-4 Tree Example: Delete



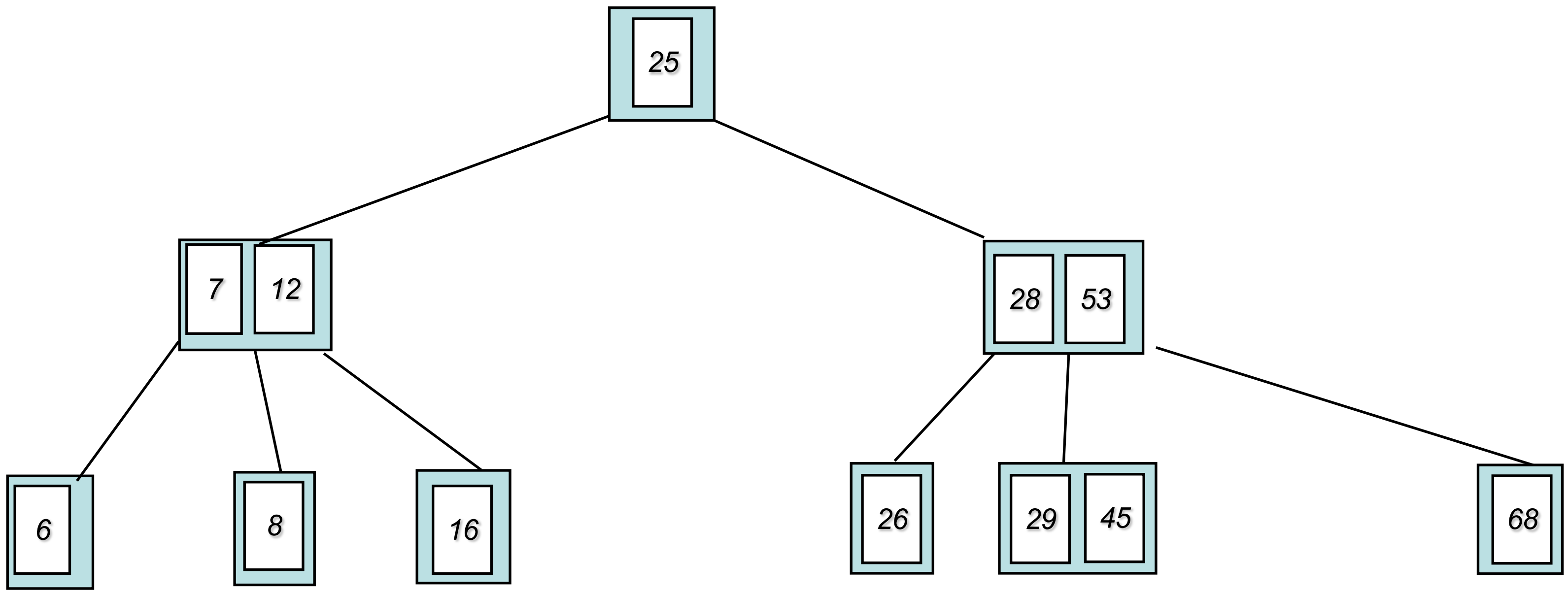
Delete 14: delete 12, swap 12 for 14

2-3-4 Tree Example: Delete



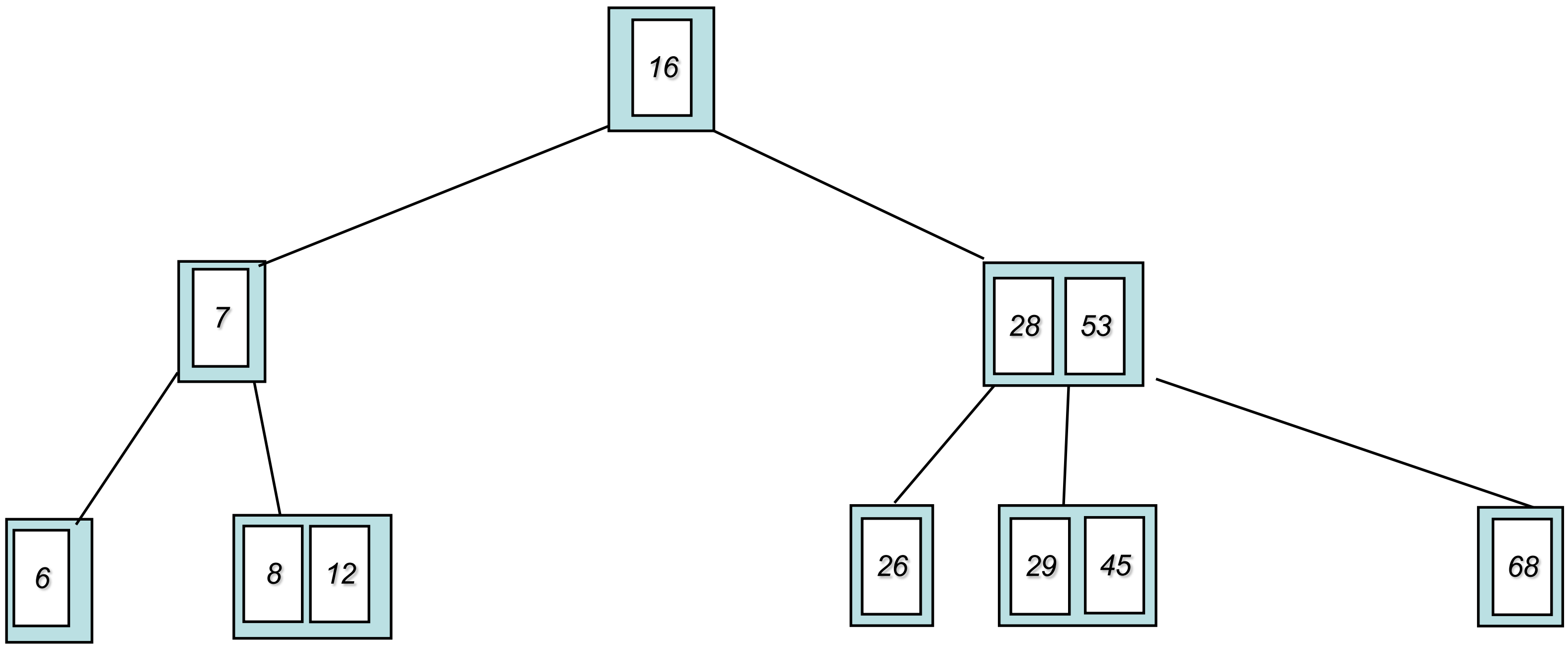
Delete 14: delete 12, swap 12 for 14

2-3-4 Tree Example: Delete



Delete 25: delete 16, swap 16 for 25

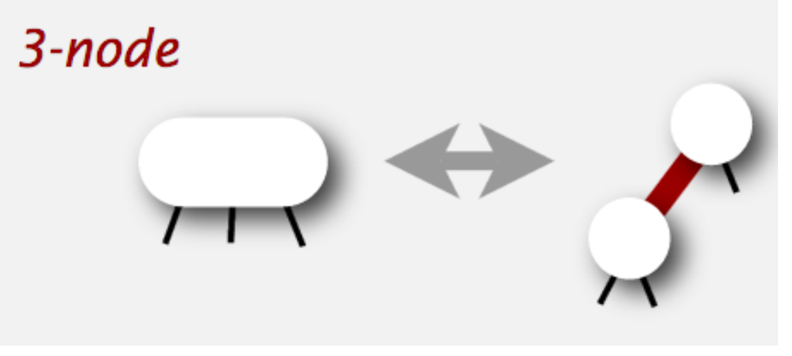
2-3-4 Tree Example: Delete

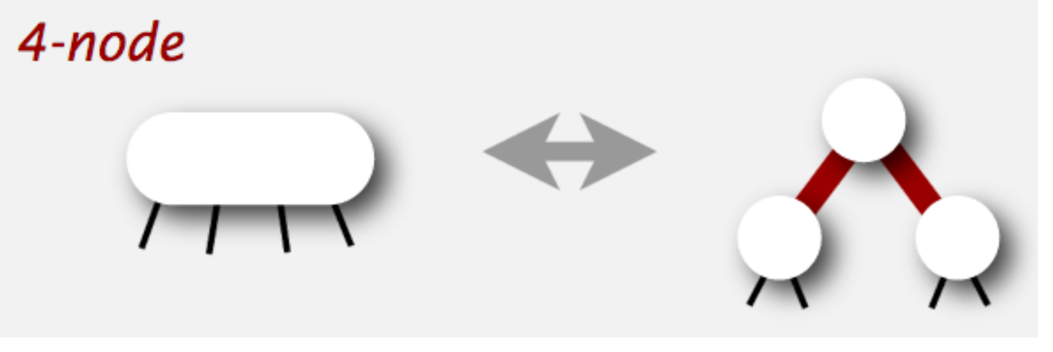


Delete 25: delete 16, swap 16 for 25

Represent 2-3-4 tree as a BST

- Use "internal" red edges for 3- and 4- nodes.
- Require that 3-nodes be left-leaning.





Represent 2-3-4 tree as a BST

- Elementary BST search works
- Easy-to-maintain 1-1 correspondence with 2-3-4 trees
- Trees therefore have perfect black-link balance