CPTS 223 Advanced Data Structure C/C++

Tree: Splay Tree

Overview

- Tree data structure
- Binary search trees
  - Support O(lg(n)) operations
  - Balanced trees
- STL set and map classes
- B-trees for accessing secondary storage
- Applications of Tree

Splay tree

- Observation:
  - Height imbalance is a problem only if & when the nodes in the deeper parts of the tree are accessed
  - Recently accessed nodes are more likely to be accessed again
- Idea:
  - Bring the recently accessed nodes to be the root
  - Amortize the overall search operation to O(log2 N) on average

not stop splaying until reaching the root

Splay tree

- Strategy:
  - After a node is accessed (search/insert), push it to the root via AVL rotations
  - Guarantees that any M consecutive operations on an empty tree will take at most O(M log2 N) time
  - Amortized cost per operation is O(log2 N)
  - Worst case: O(N) when the tree is skewed and the target node is the deeper part
  - Does not require maintaining height or balance information

not stop splaying until reaching the root

Divided by M

Three types of cases

- Worst-case:
  - O(f(N)) → for each single operation
  - if we have M operations:
  - M * O(f(N)) = O(M * f(N)) = O(M * log(N))
- Average-case for 1000 cases:
  - O(f1(N) + f2(N) + … + f1000(N) / 1000)
- Amortized analysis for any sequence of M operations:
  - O(f_1(N) + f_2(N) + … + f_M(N)) = O(M * log(N))
  - for each operation: O(log(N))

Splaying: zig-zag

- Node X is right-child of parent, which is left-child of grandparent (or vice-versa)
- Perform double rotation (left, right)

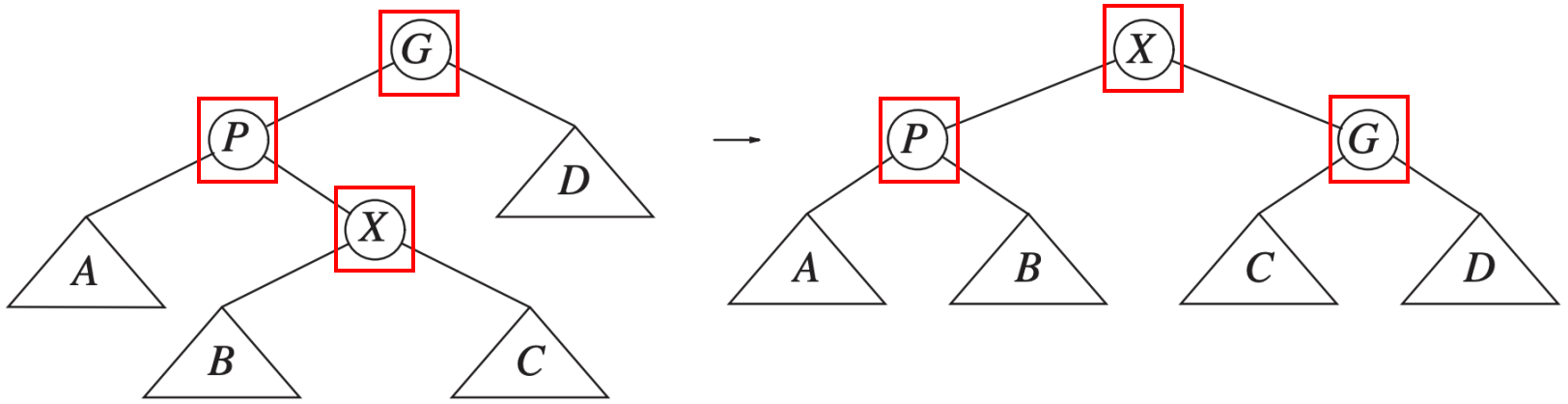


Figure 4.48 Zig-zag

Splaying: zig-zag

- Node X is right-child of parent, which is left-child of grandparent (or vice-versa)
- Perform double rotation (left, right)

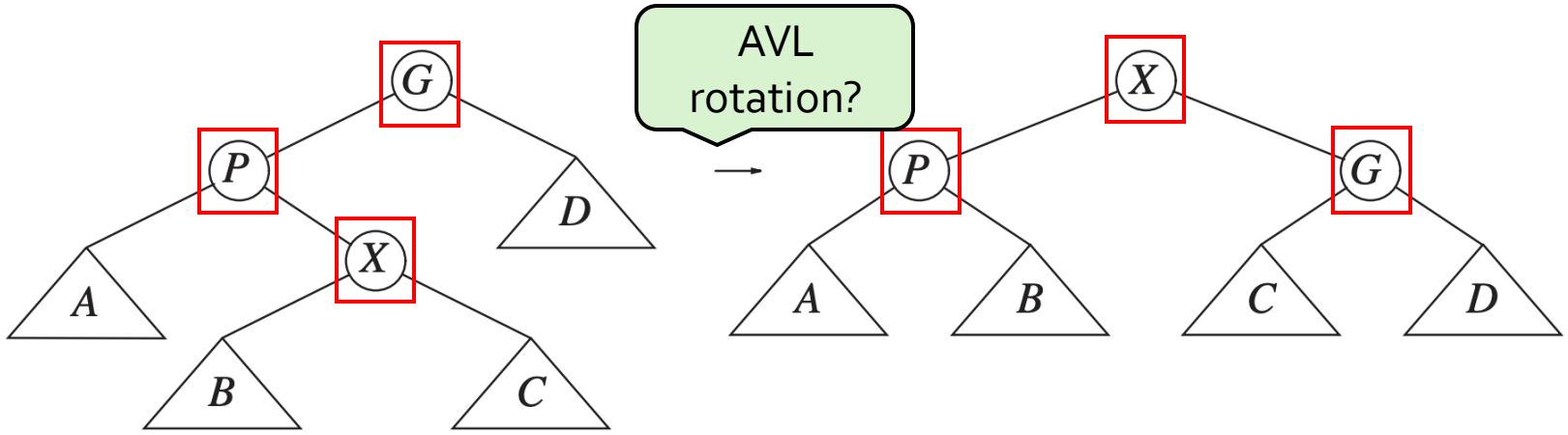


Figure 4.48 Zig-zag

Splaying: zig-zag

- No
vic
- Per

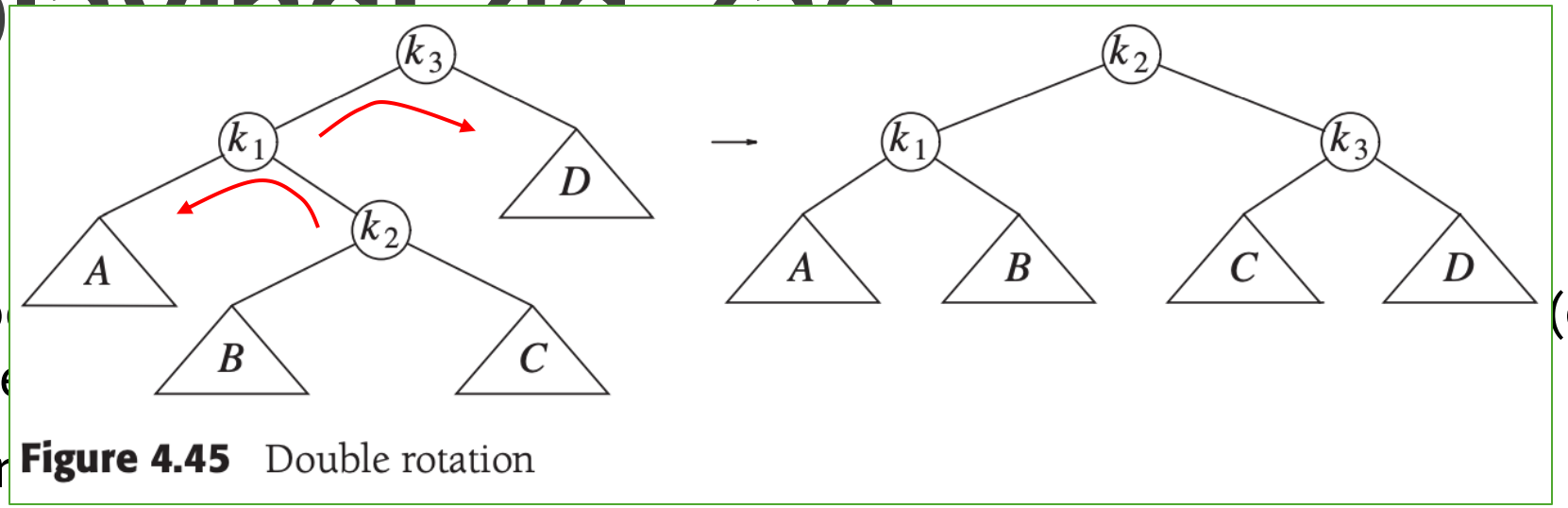


Figure 4.45 Double rotation

(or

AVL rotation?

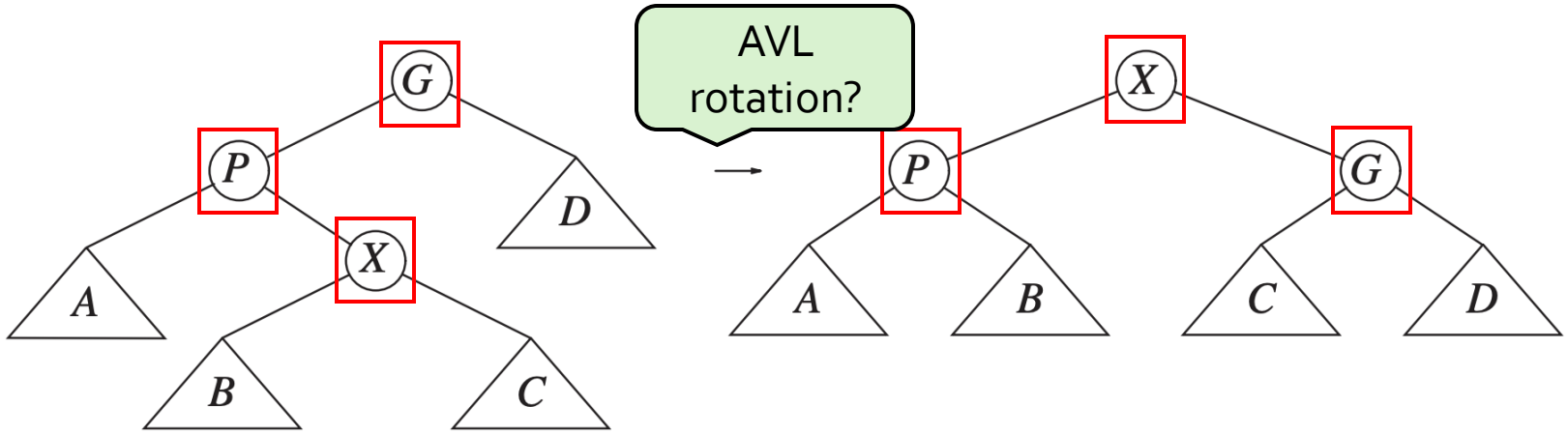


Figure 4.48 Zig-zag

Splaying: zig-zig

- Node X is left-child of parent, which is left-child of grandparent (or right-right)
- Perform double rotation (right-right)

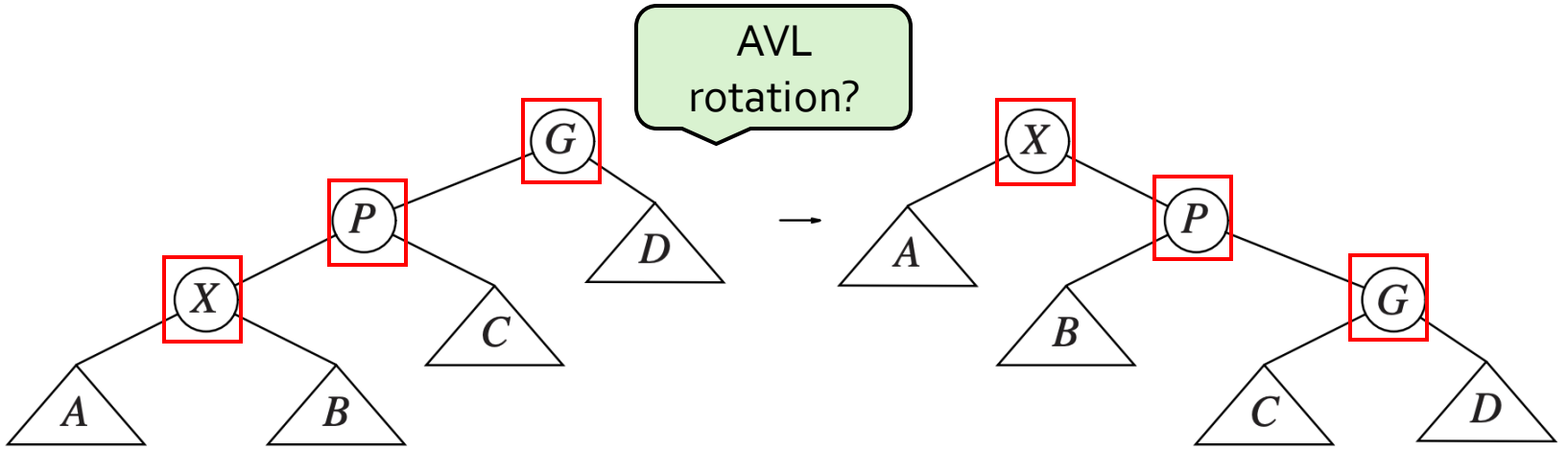


Figure 4.49 Zig-zig

Splaying: zig-zig

- Nod
  right
- Perf

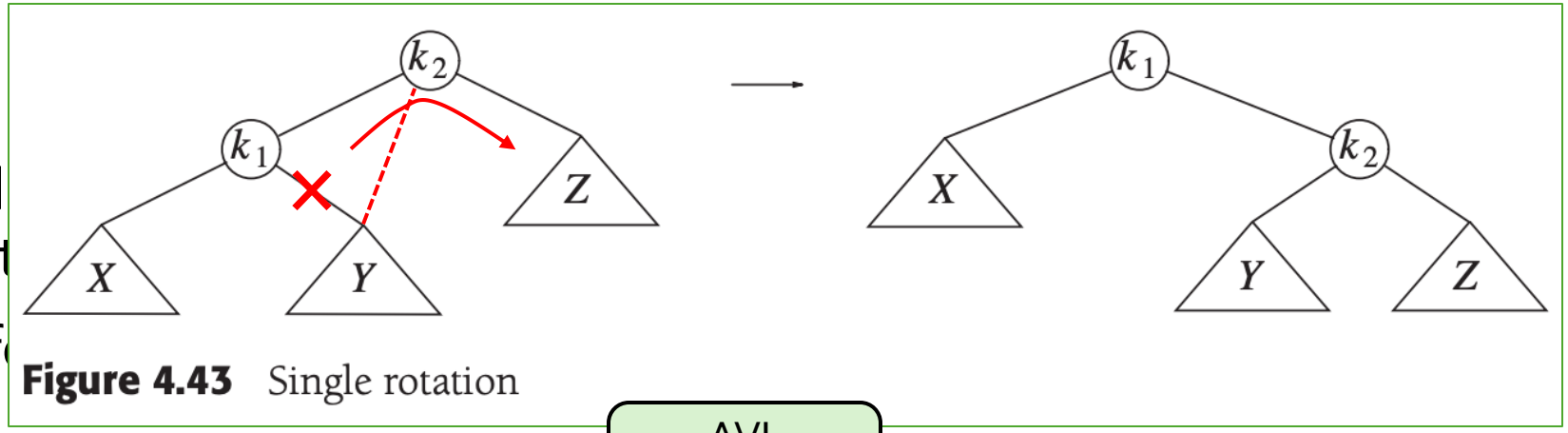


Figure 4.43 Single rotation

AVL rotation?

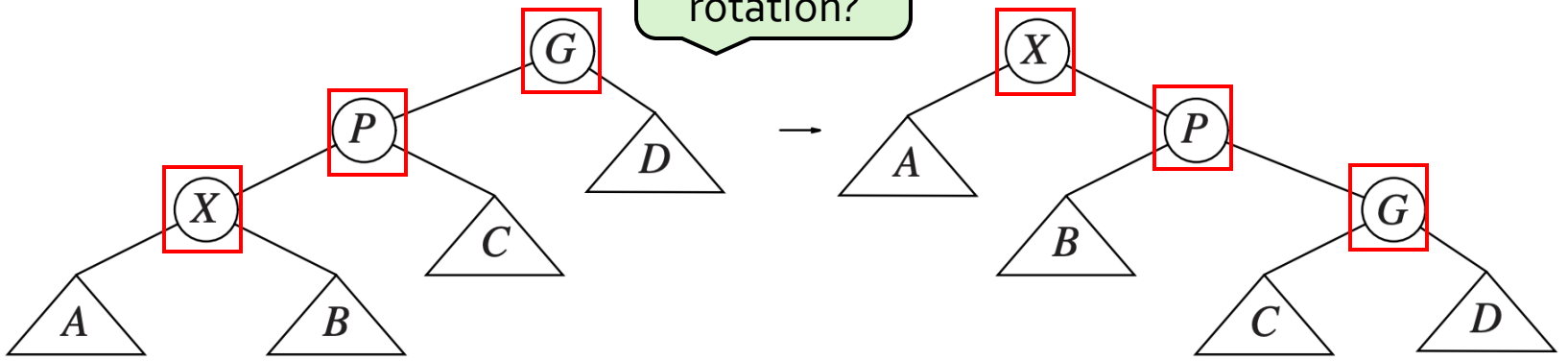
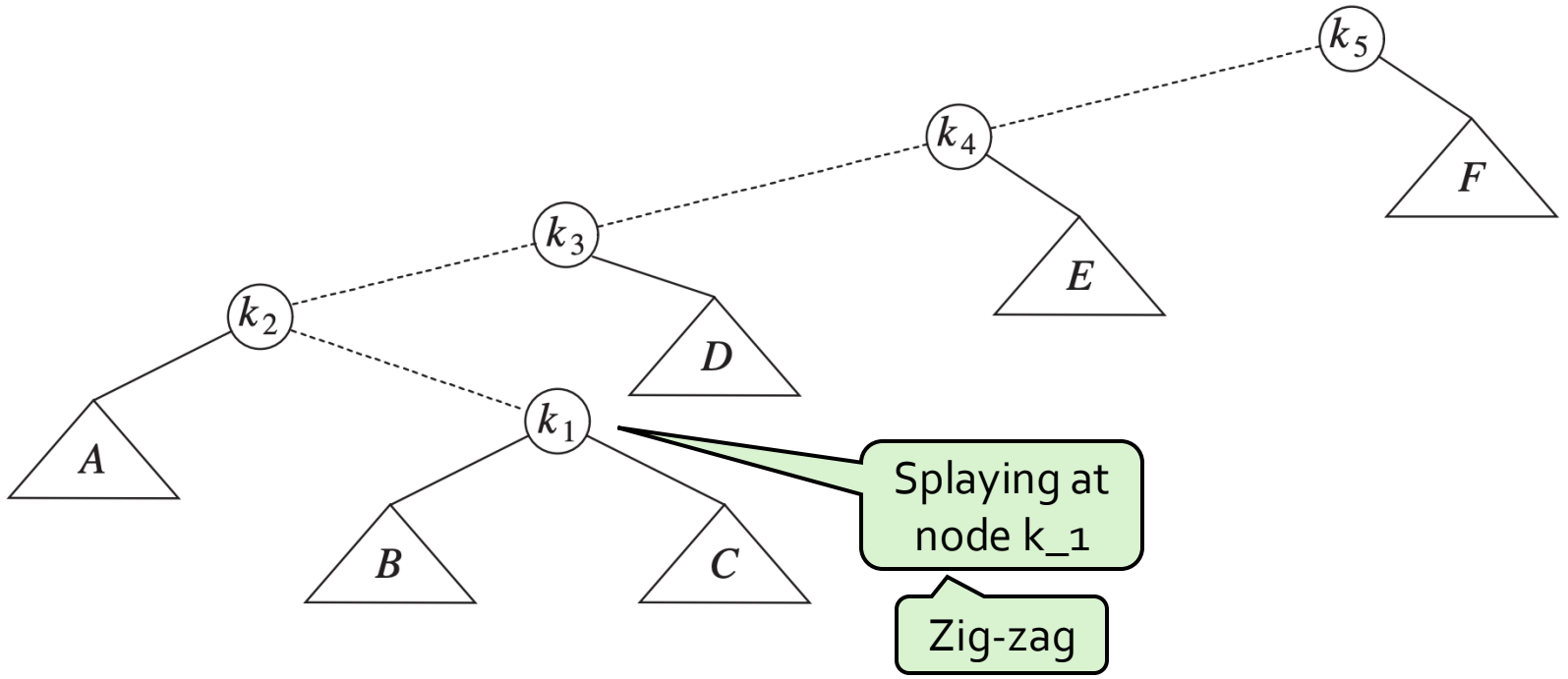


Figure 4.49 Zig-zig

Splay tree



Sp

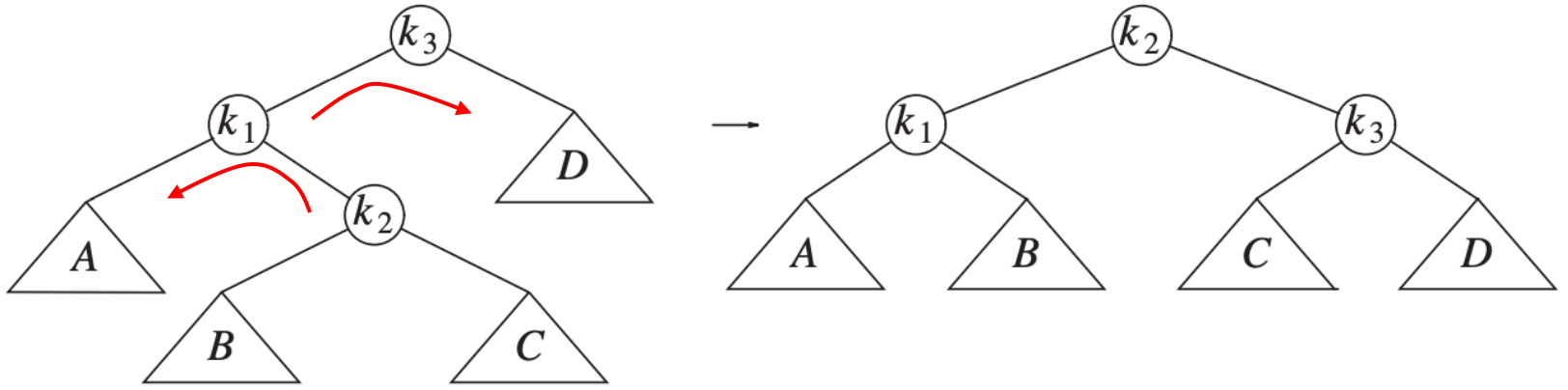
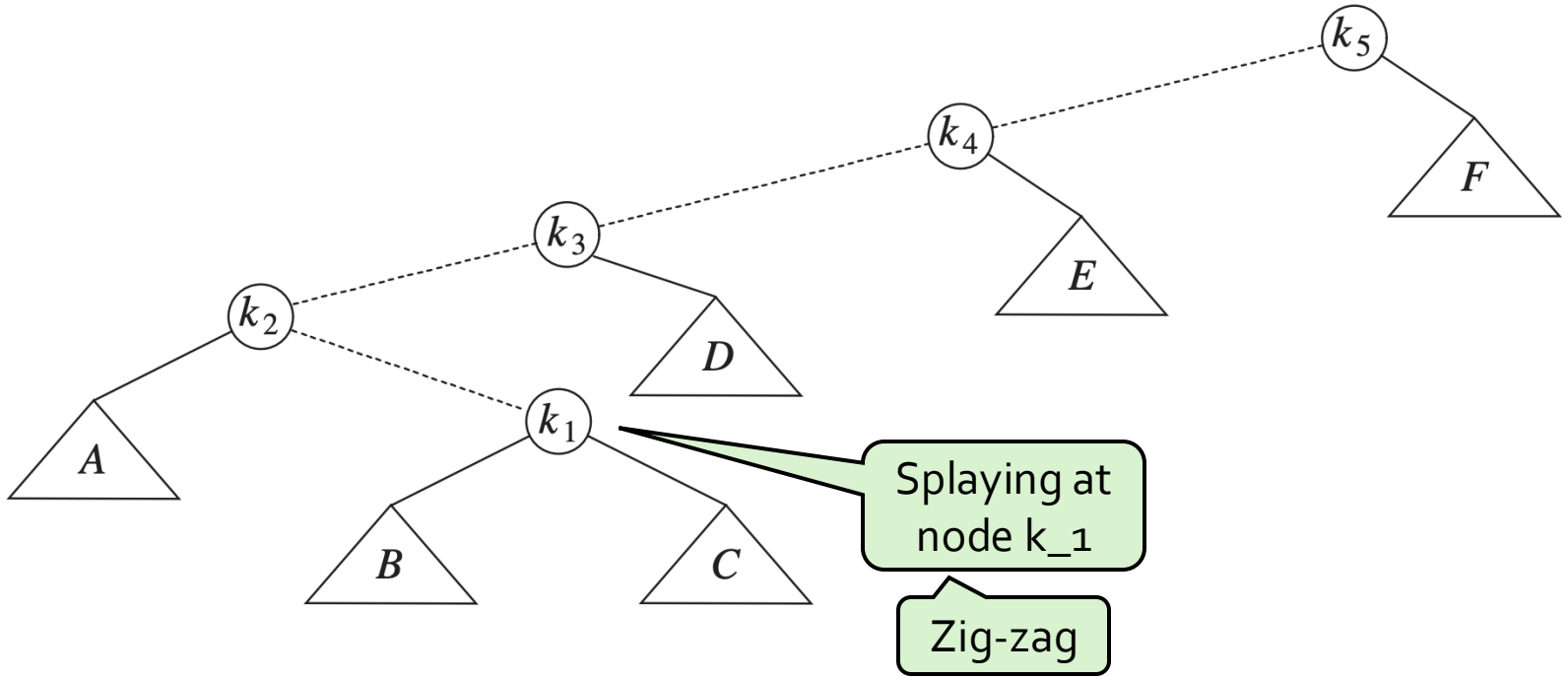


Figure 4.45 Double rotation



Sp

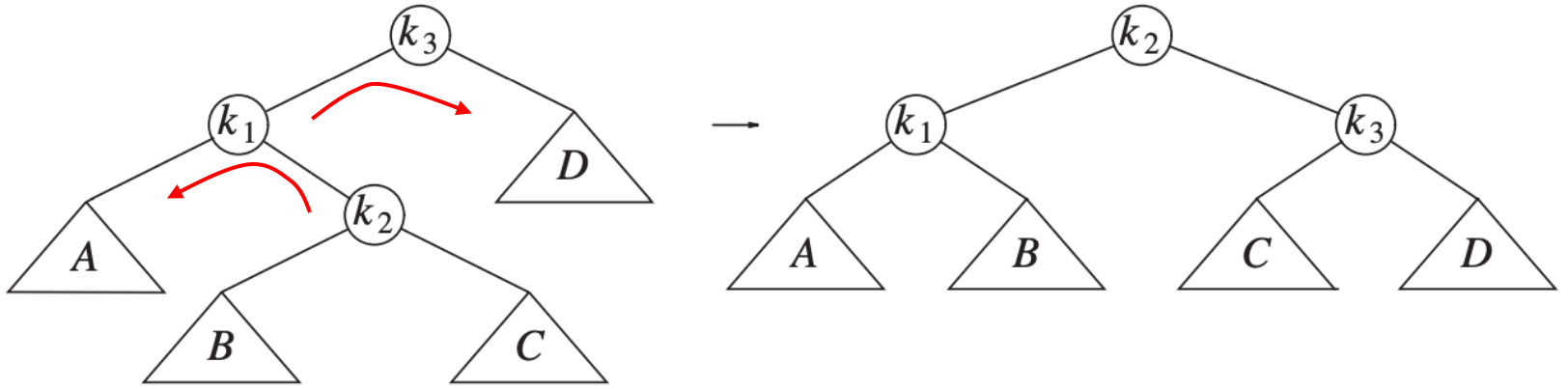
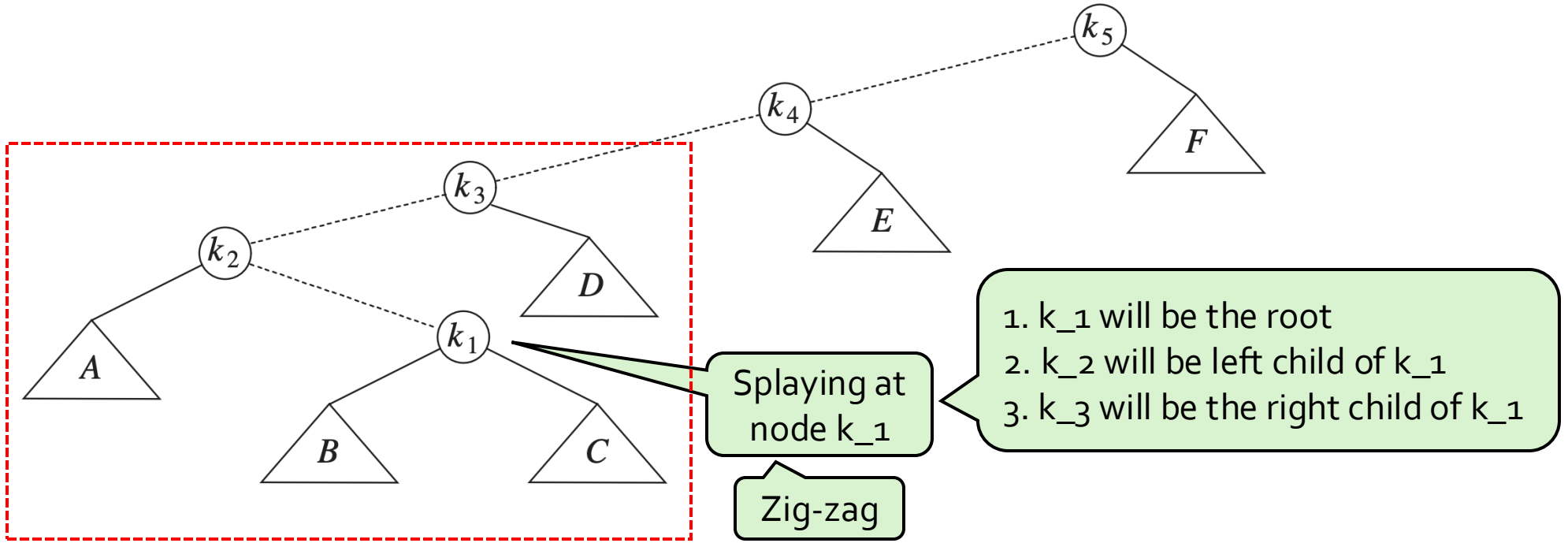
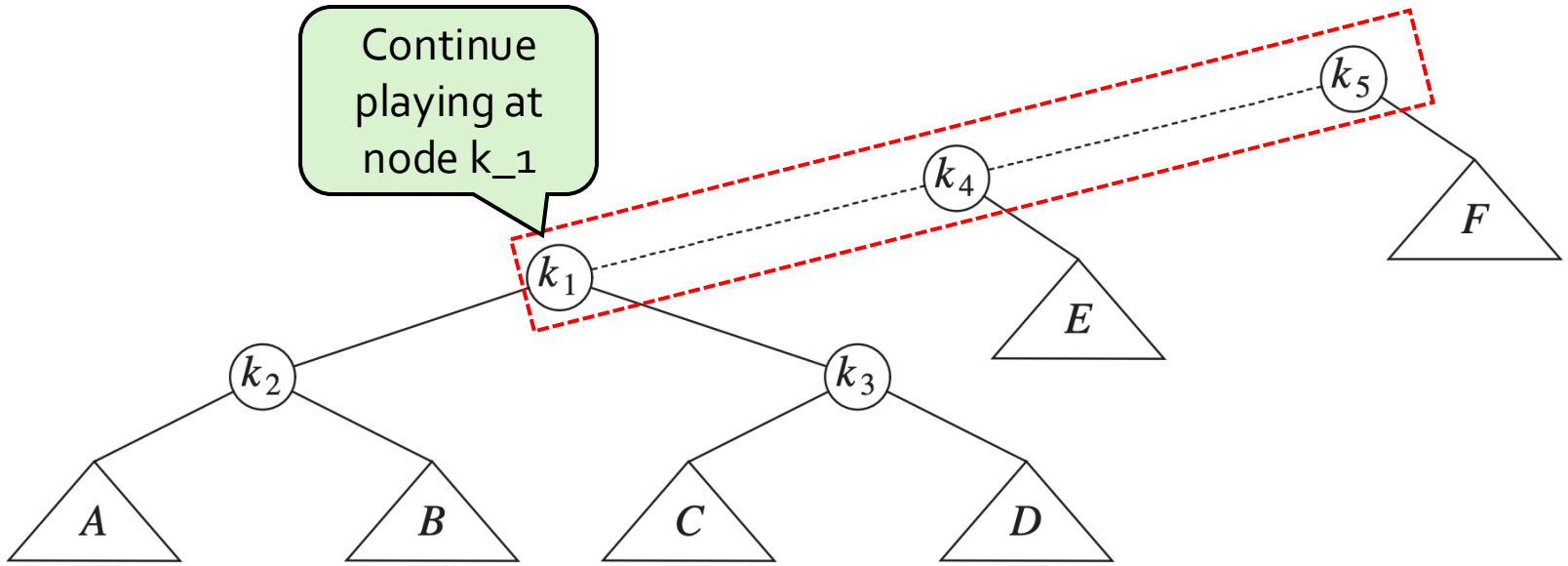


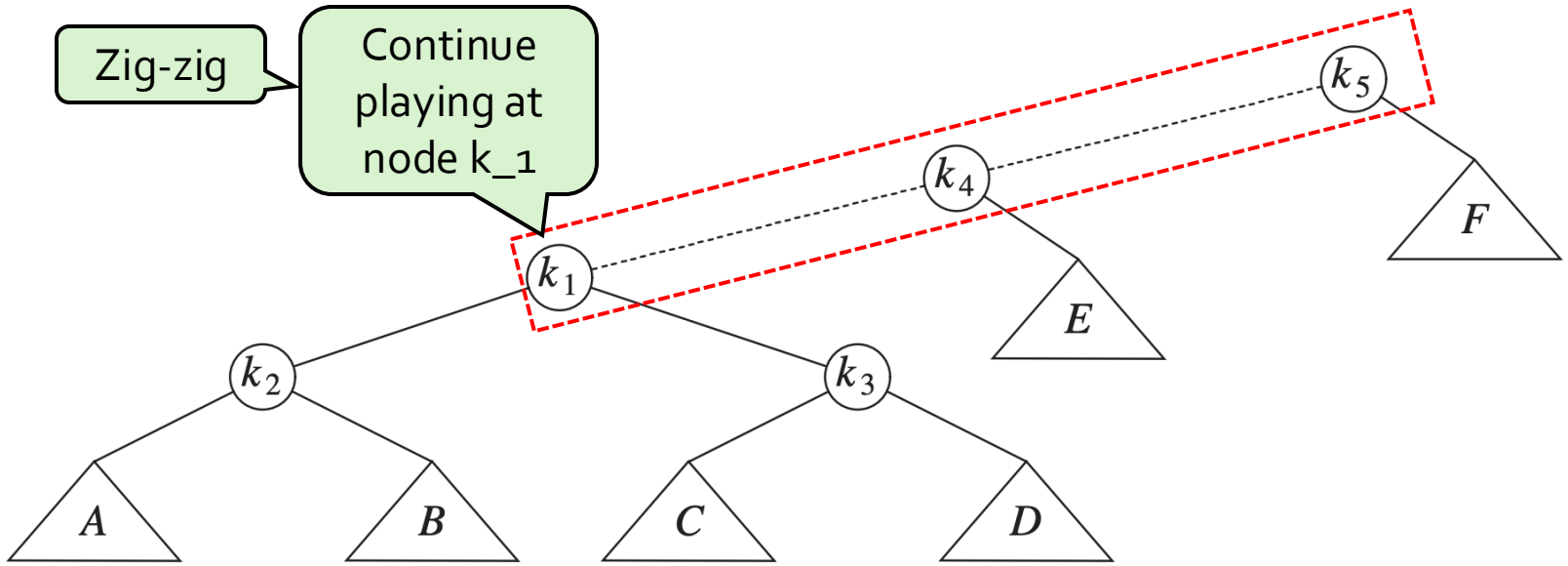
Figure 4.45 Double rotation



Splay tree



Splay tree



Sp

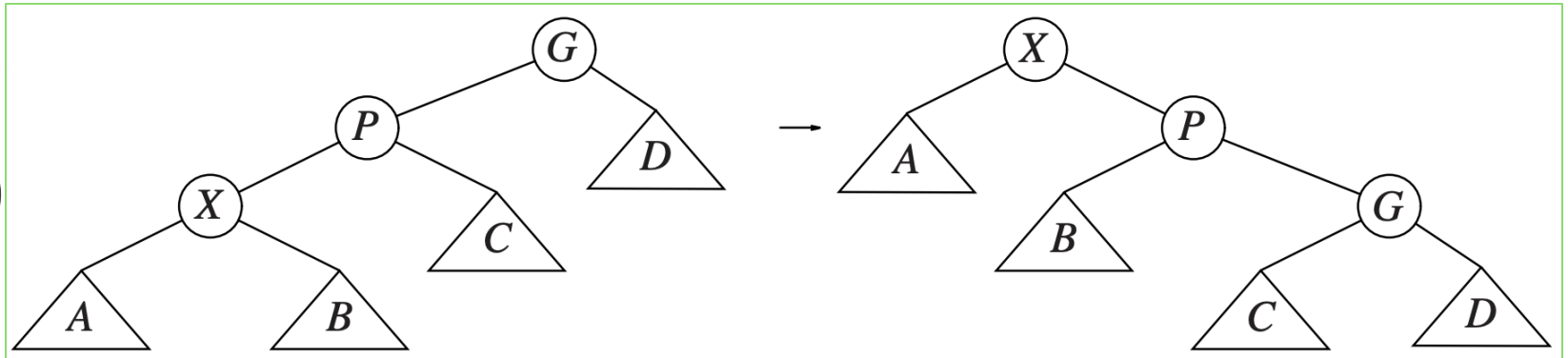
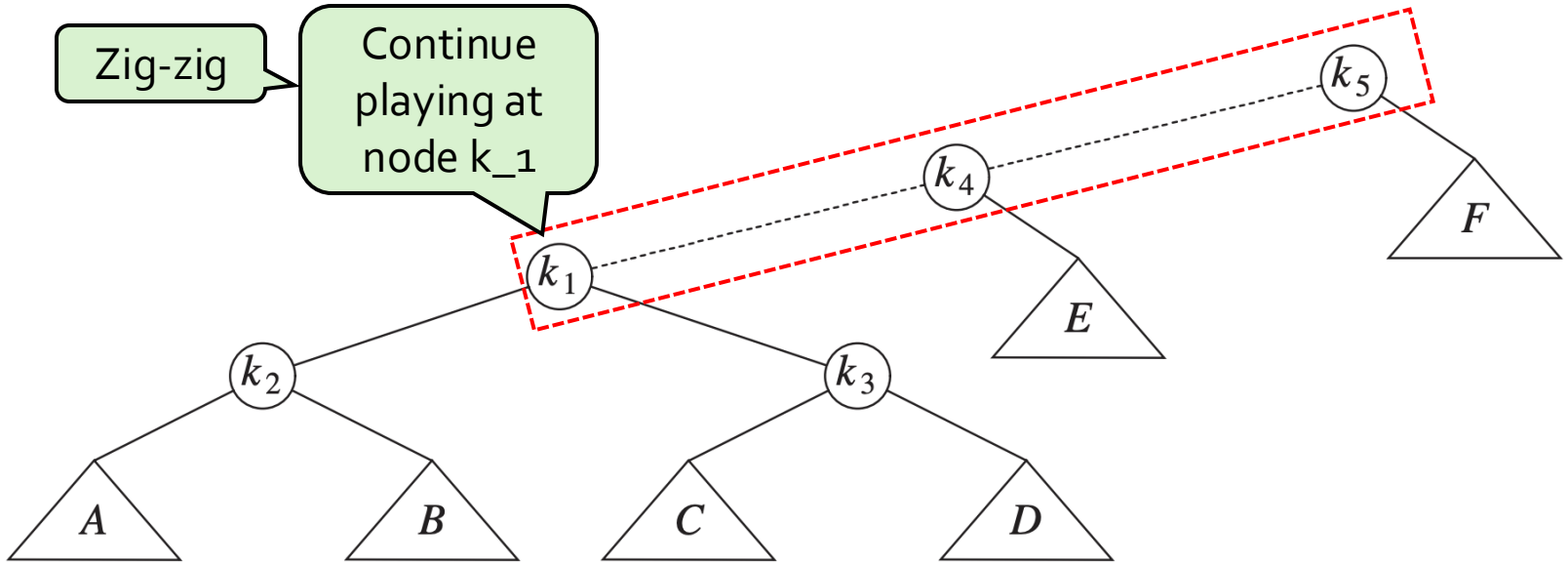


Figure 4.49 Zig-zig



Sp

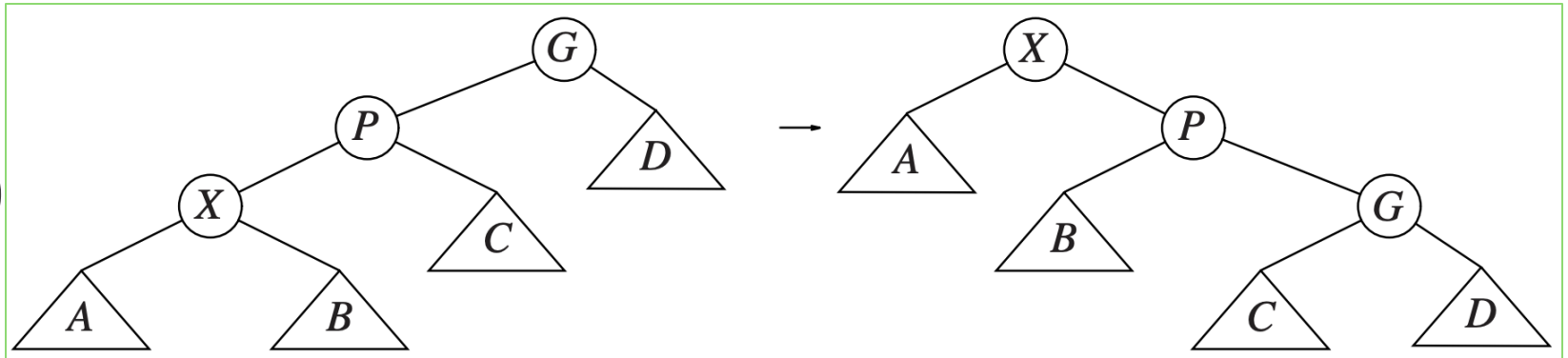
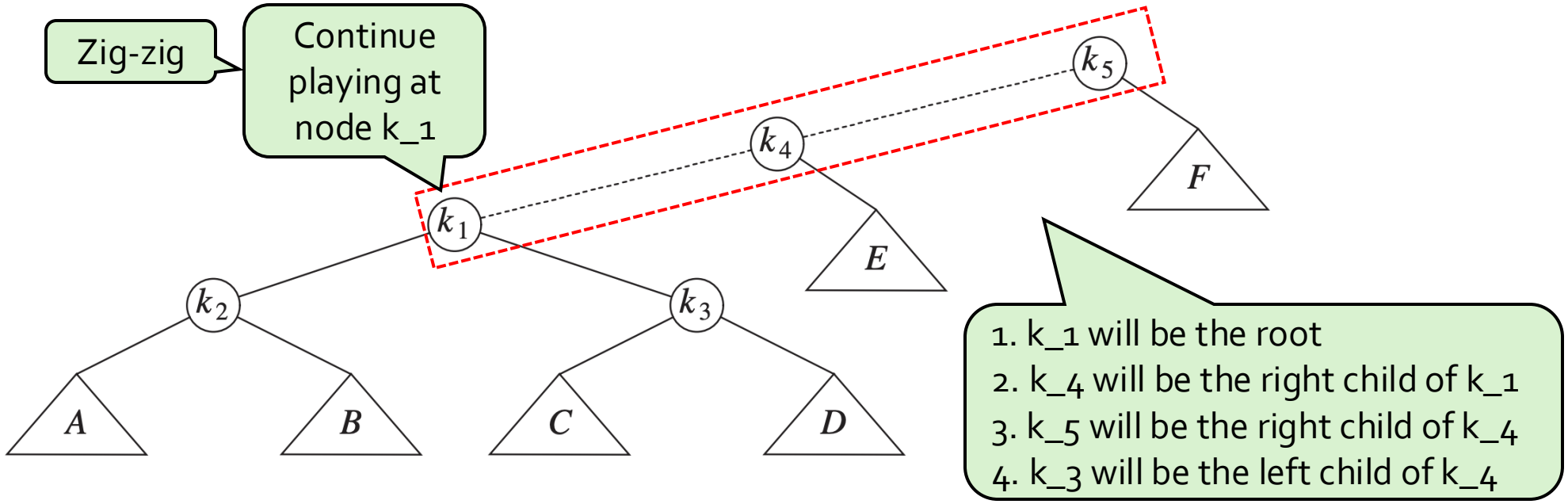
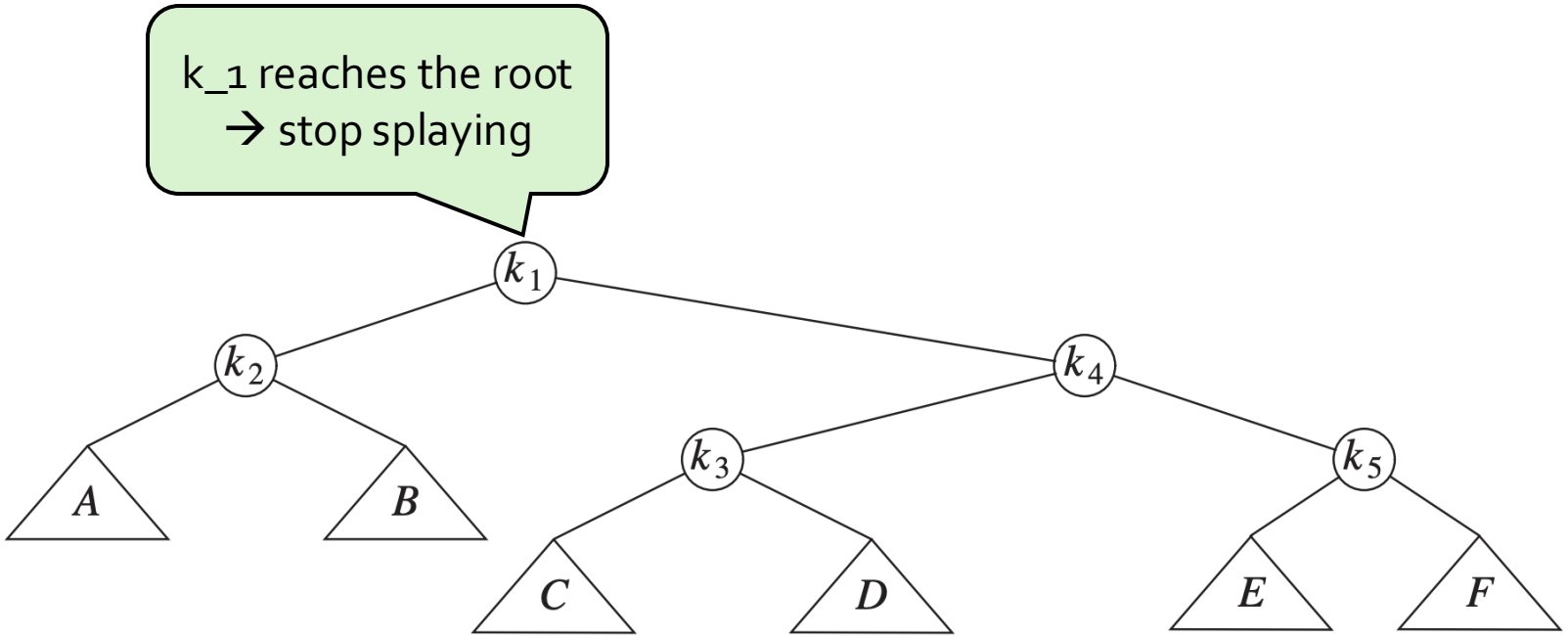


Figure 4.49 Zig-zig

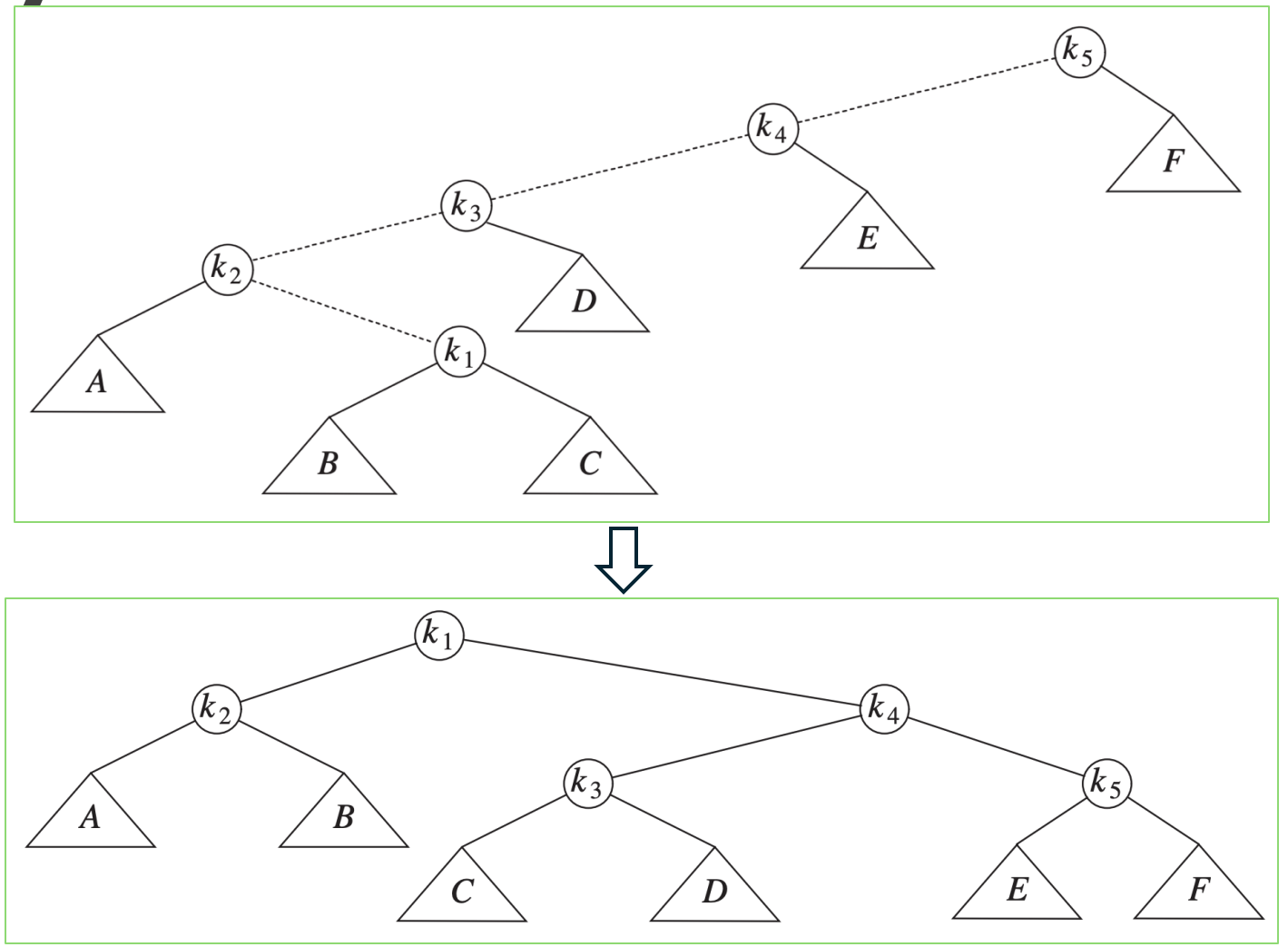


Splay tree



WSU

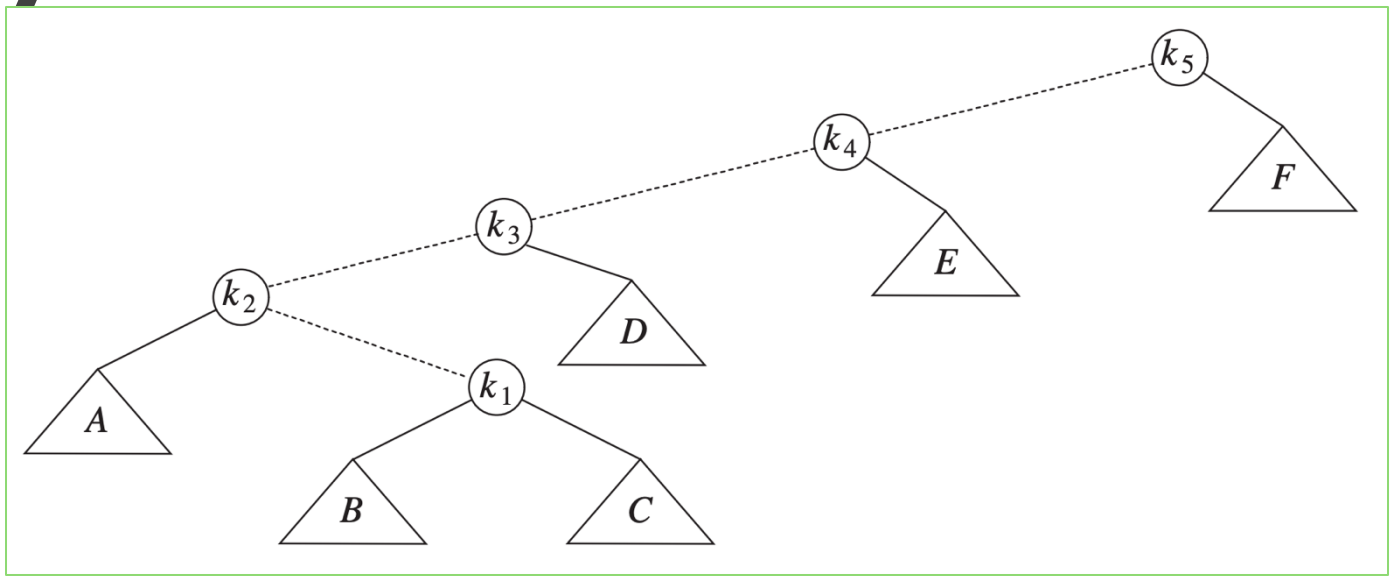
Splay tree



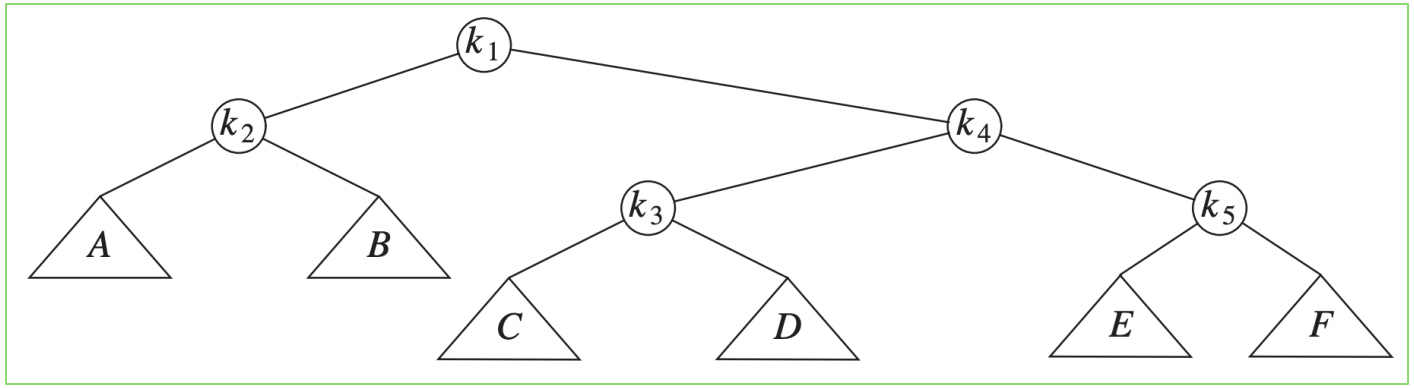
Splay tree

height = 5

Assume A-F are nodes



height = 3



Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

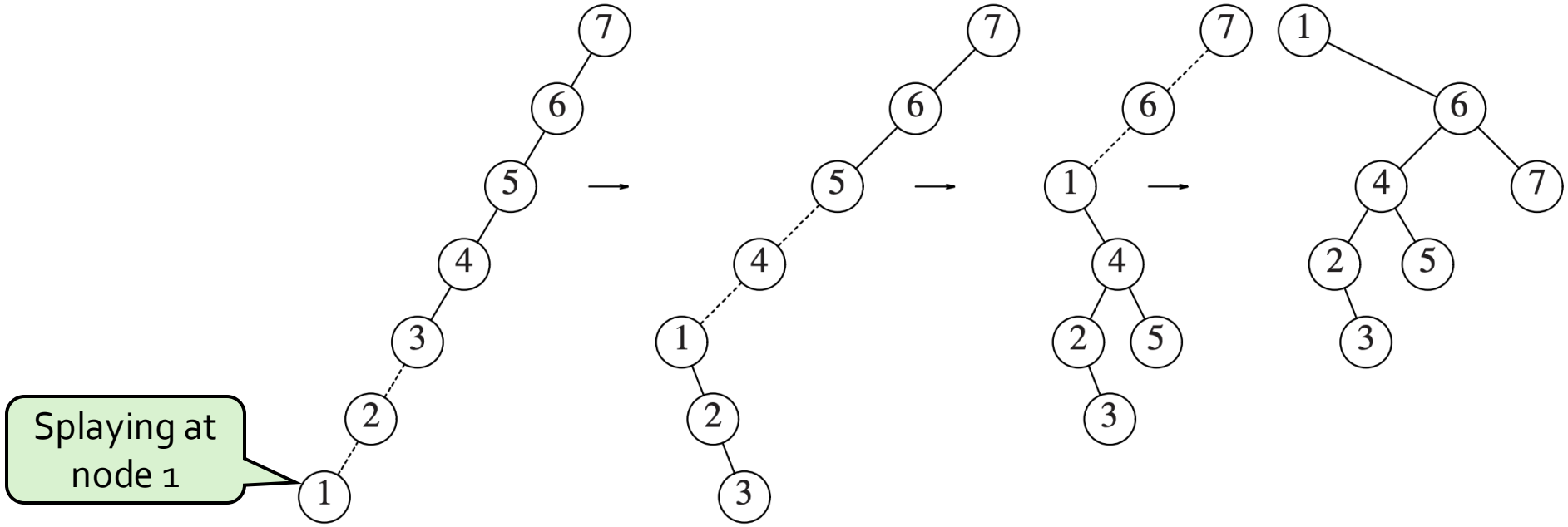


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

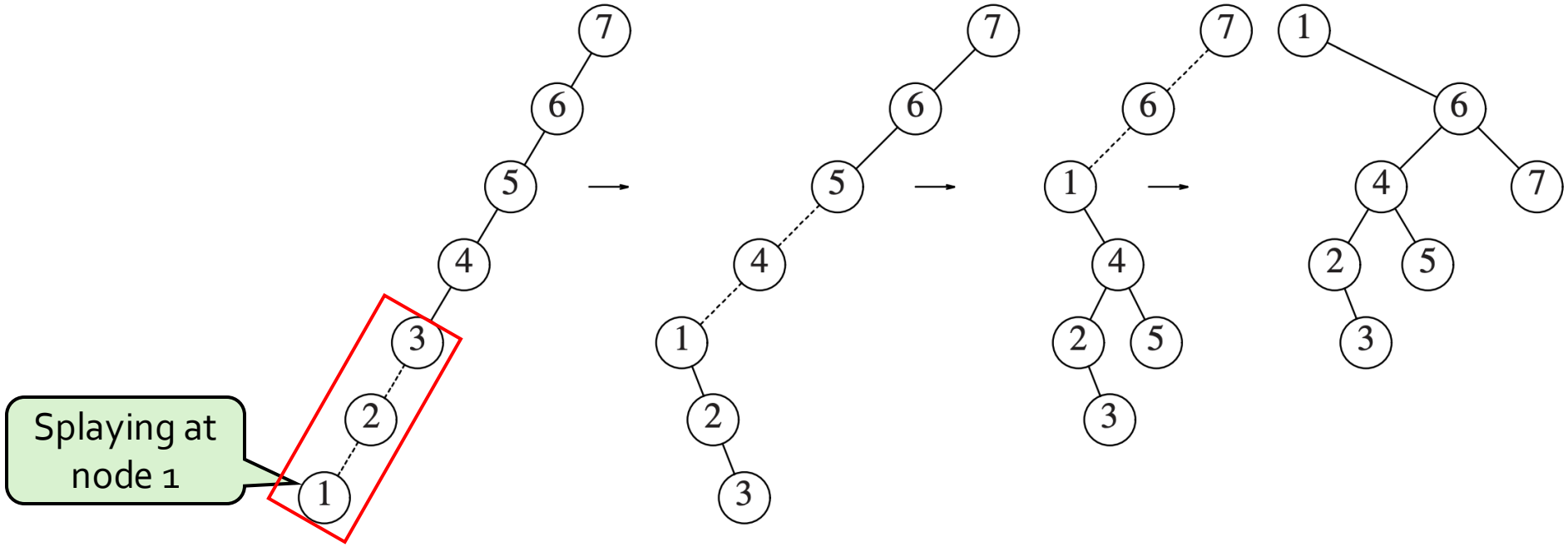


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

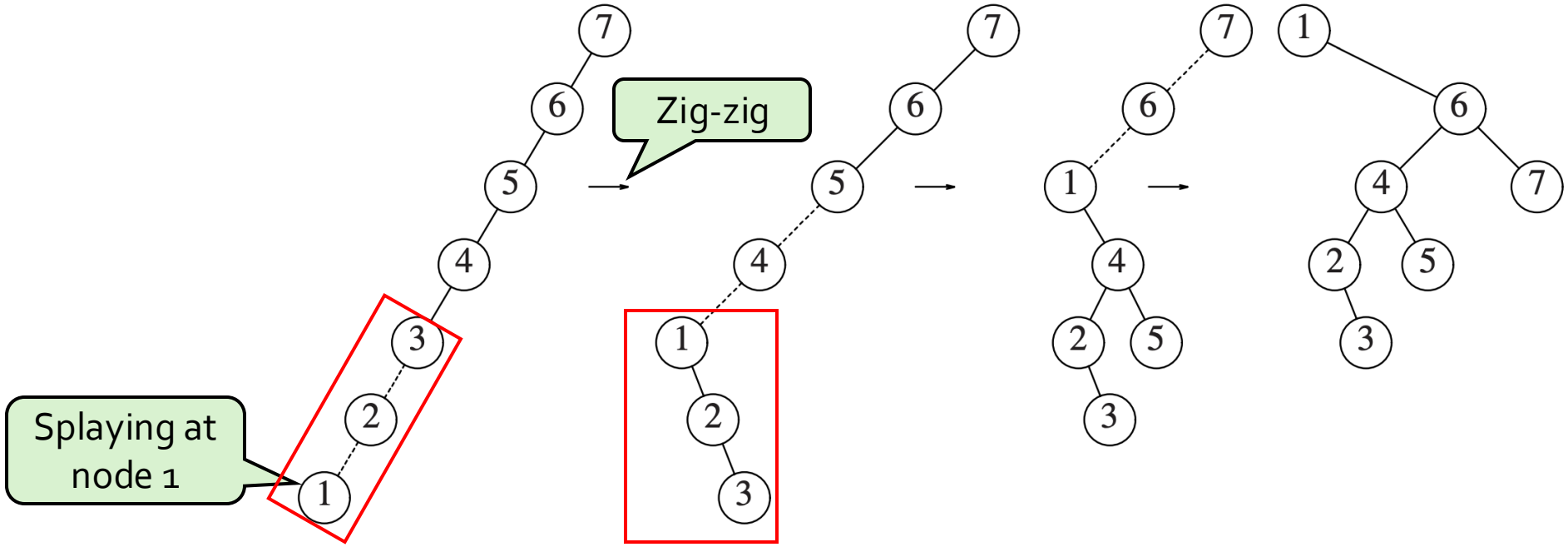


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

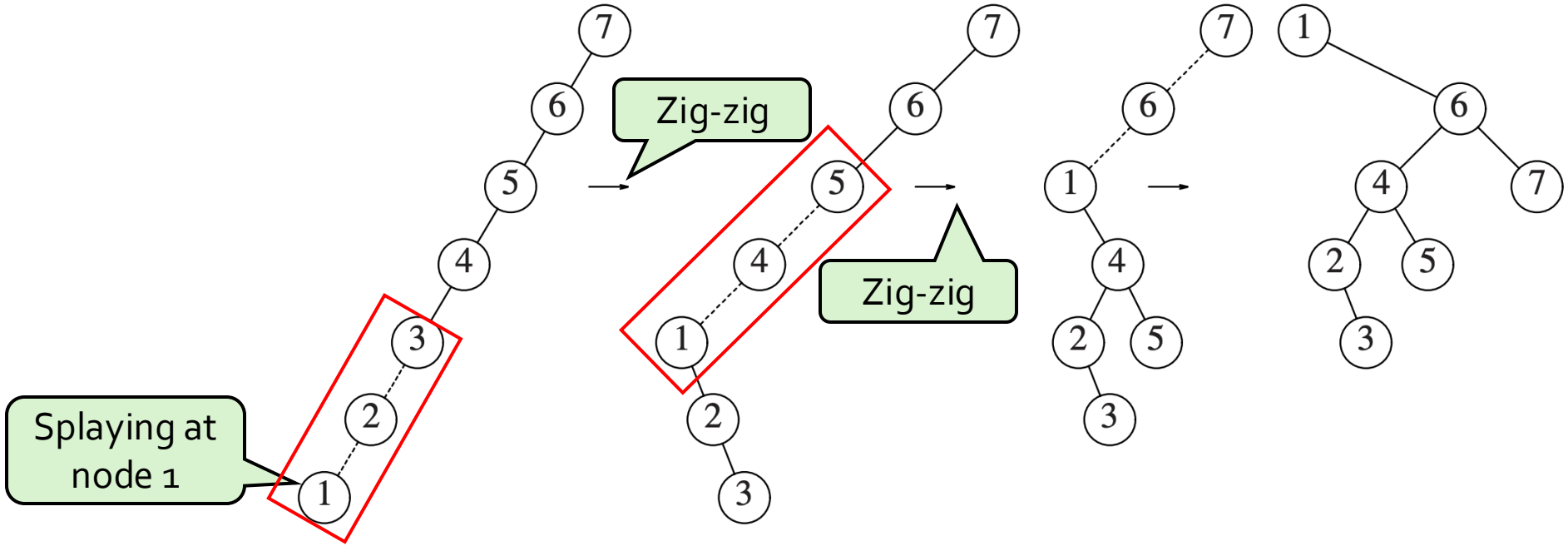


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

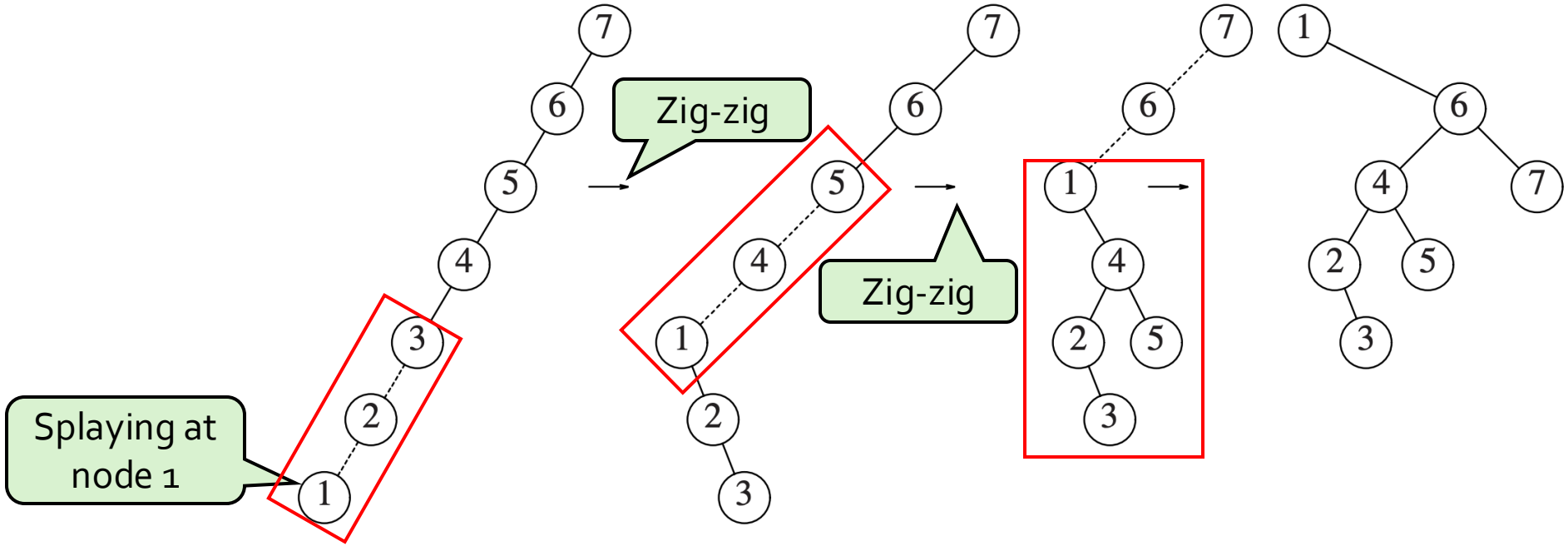


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

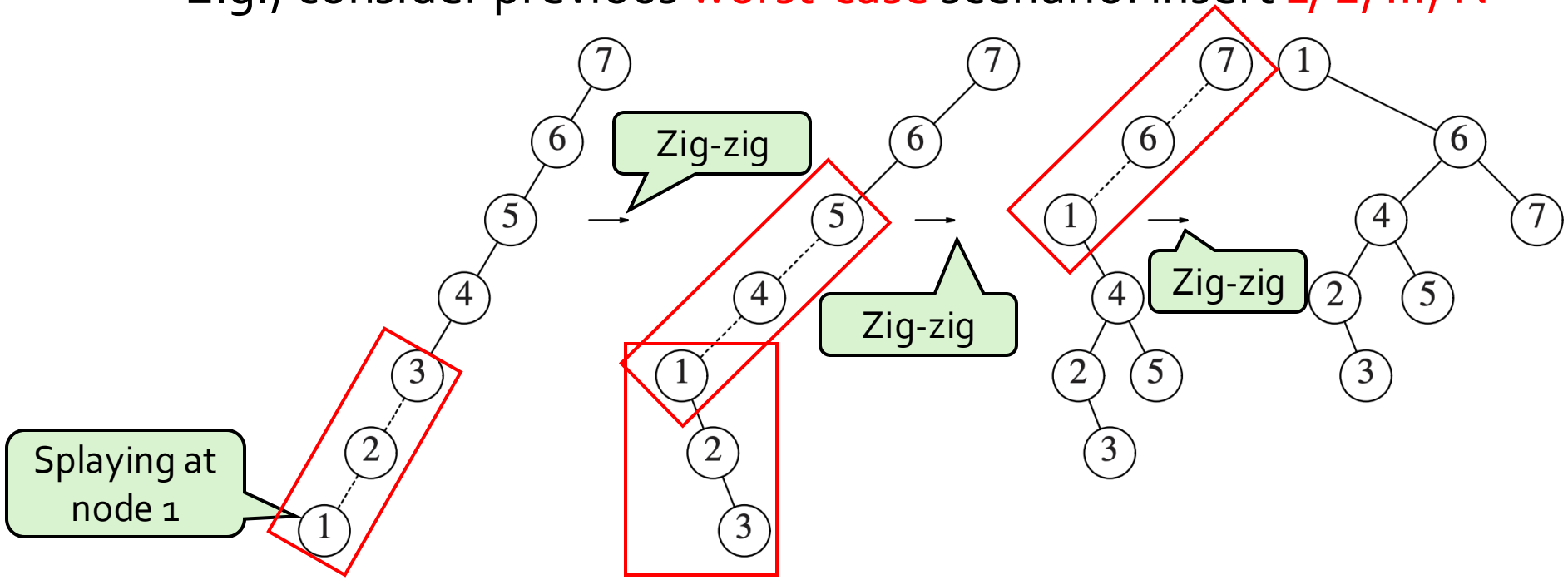


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

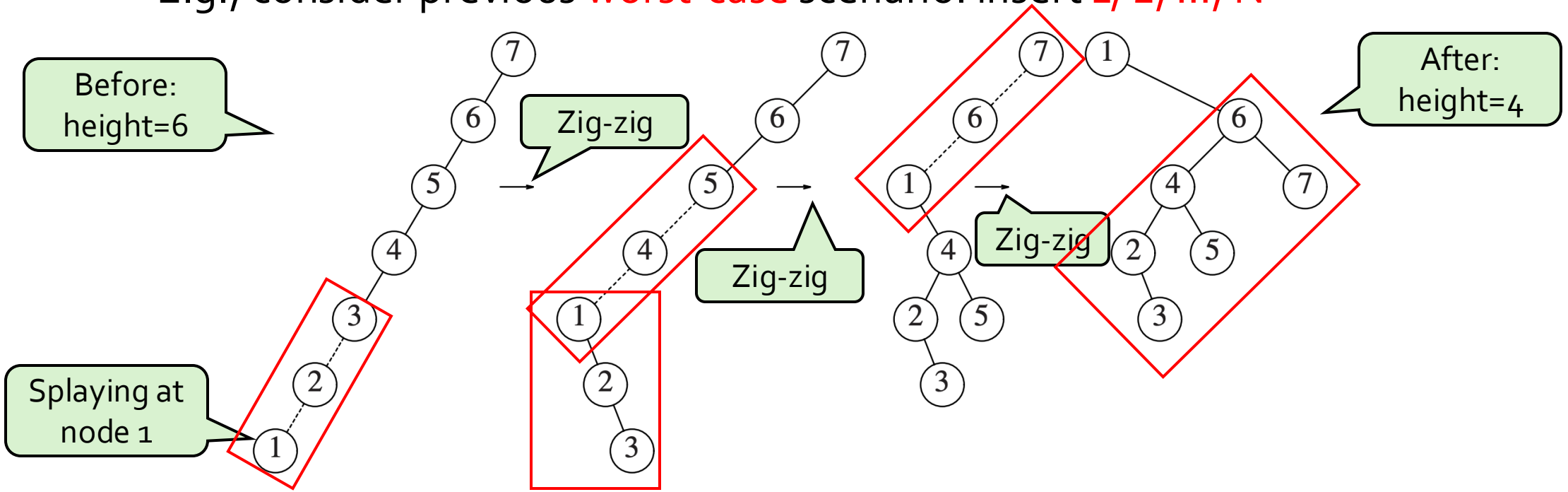


Figure 4.50 Result of splaying at node 1

Splay tree

- E.g., consider previous worst-case scenario: insert 1, 2, …, N

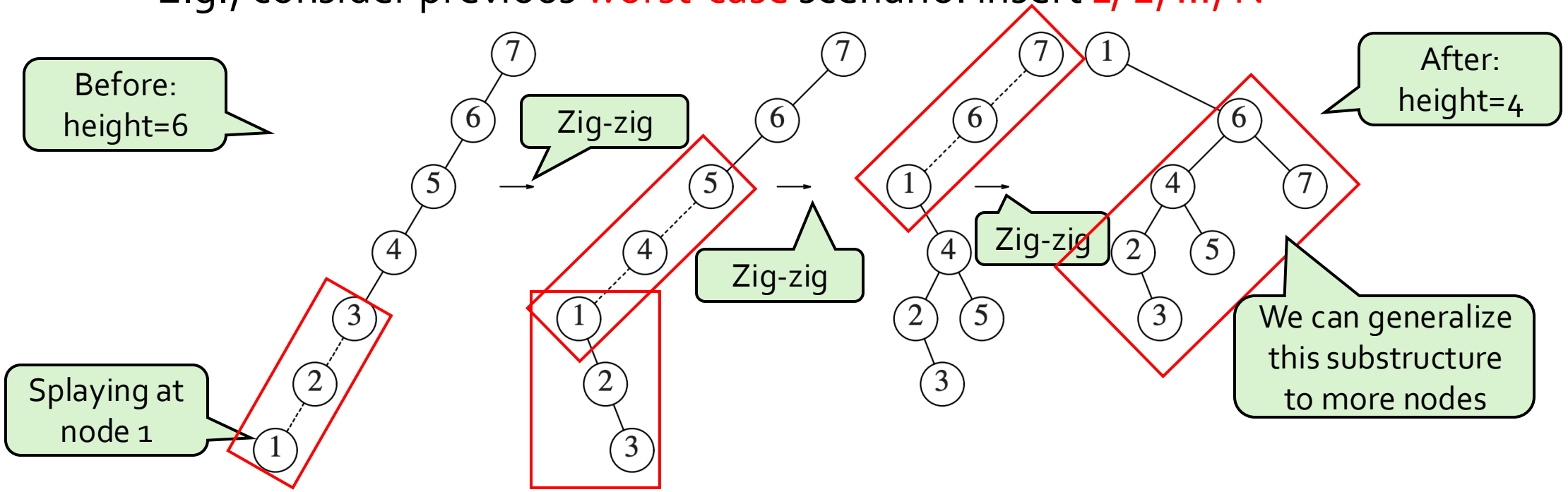


Figure 4.50 Result of splaying at node 1

Splay tree

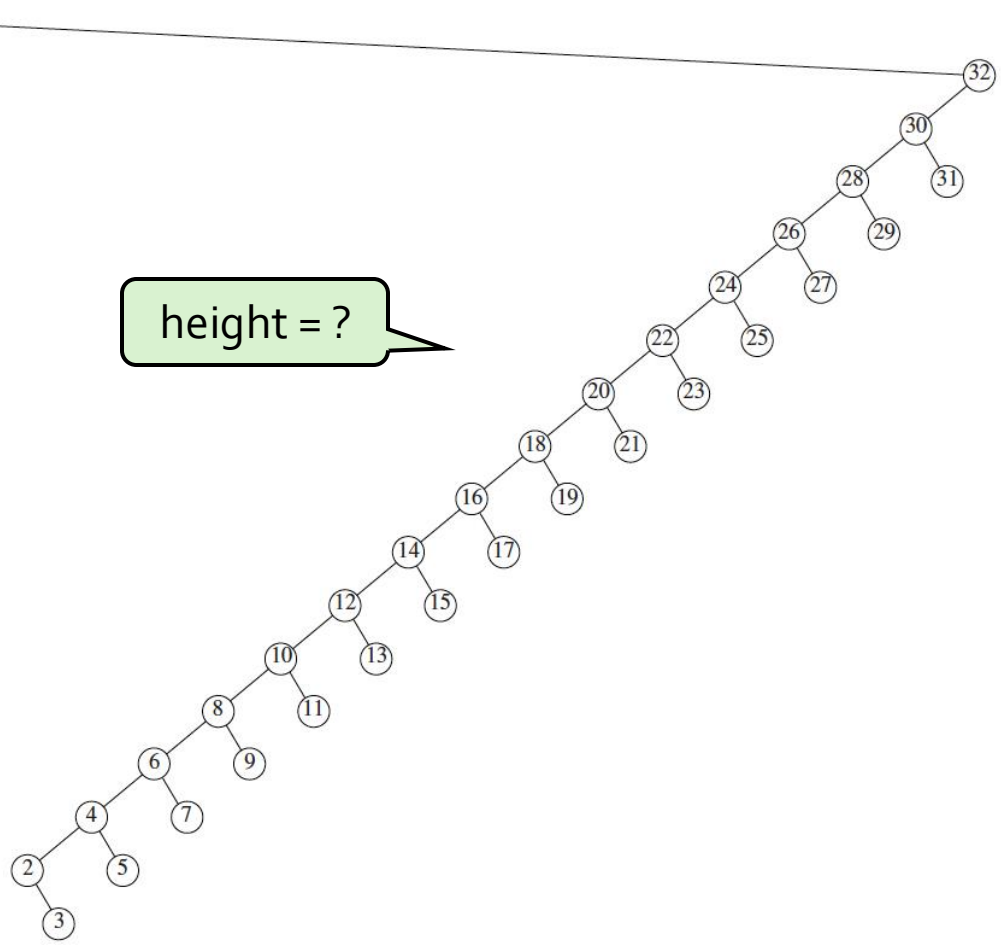


Figure 4.51 Result of splaying at node 1 a tree of all left children

Splay tree

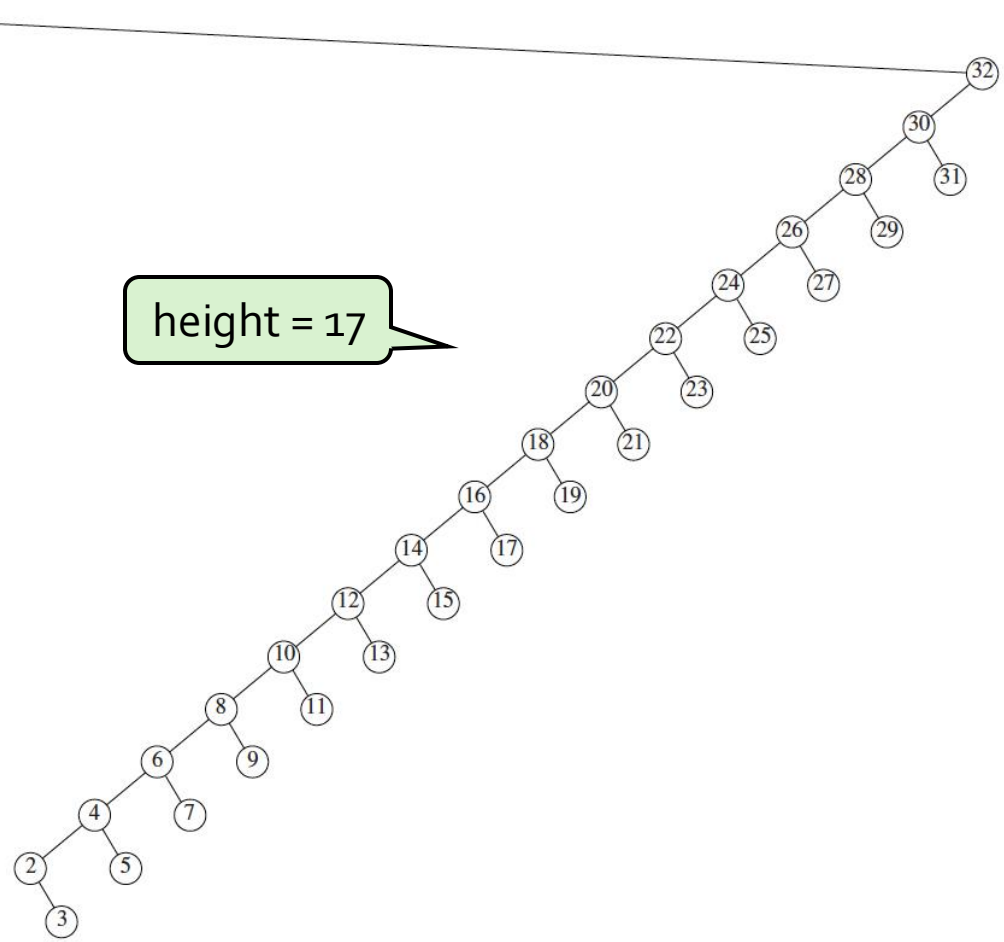


Figure 4.51 Result of splaying at node 1 a tree of all left children

Splay tree

height = 17

Continue splaying nodes from smallest to largest (2, 3, 4, ...)

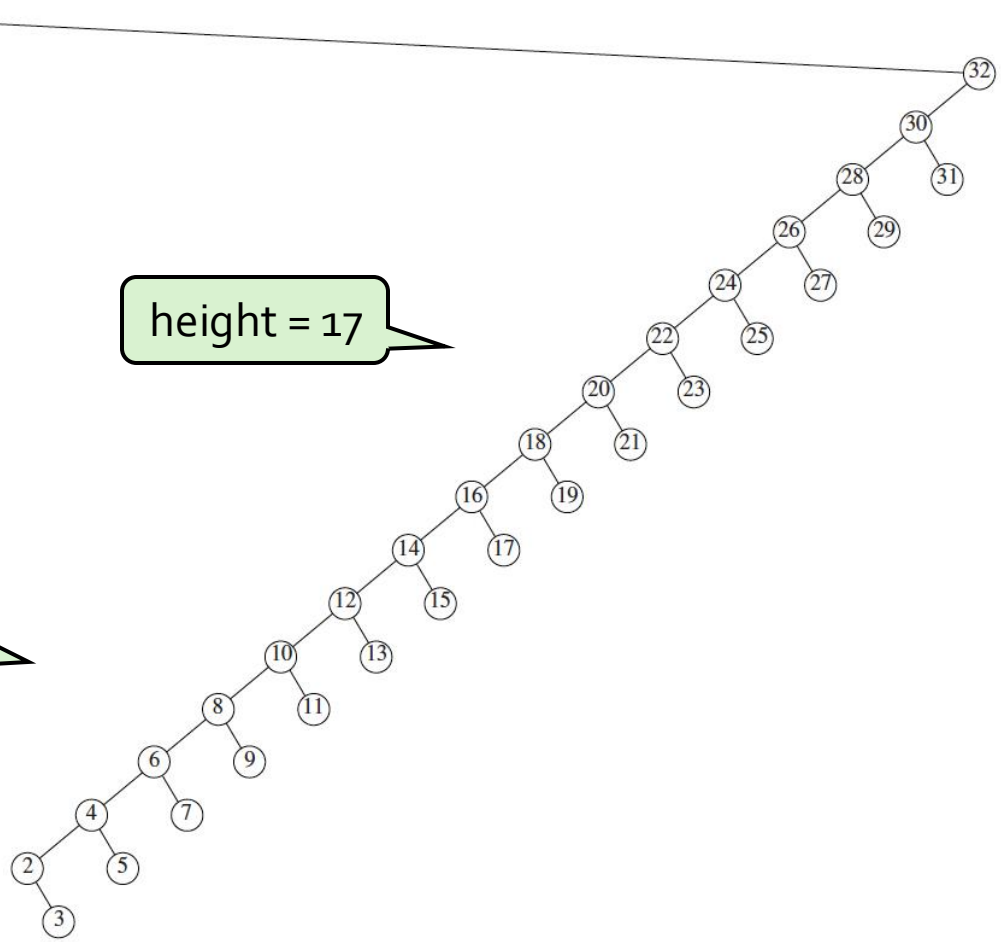


Figure 4.51 Result of splaying at node 1 a tree of all left children

Splay tree

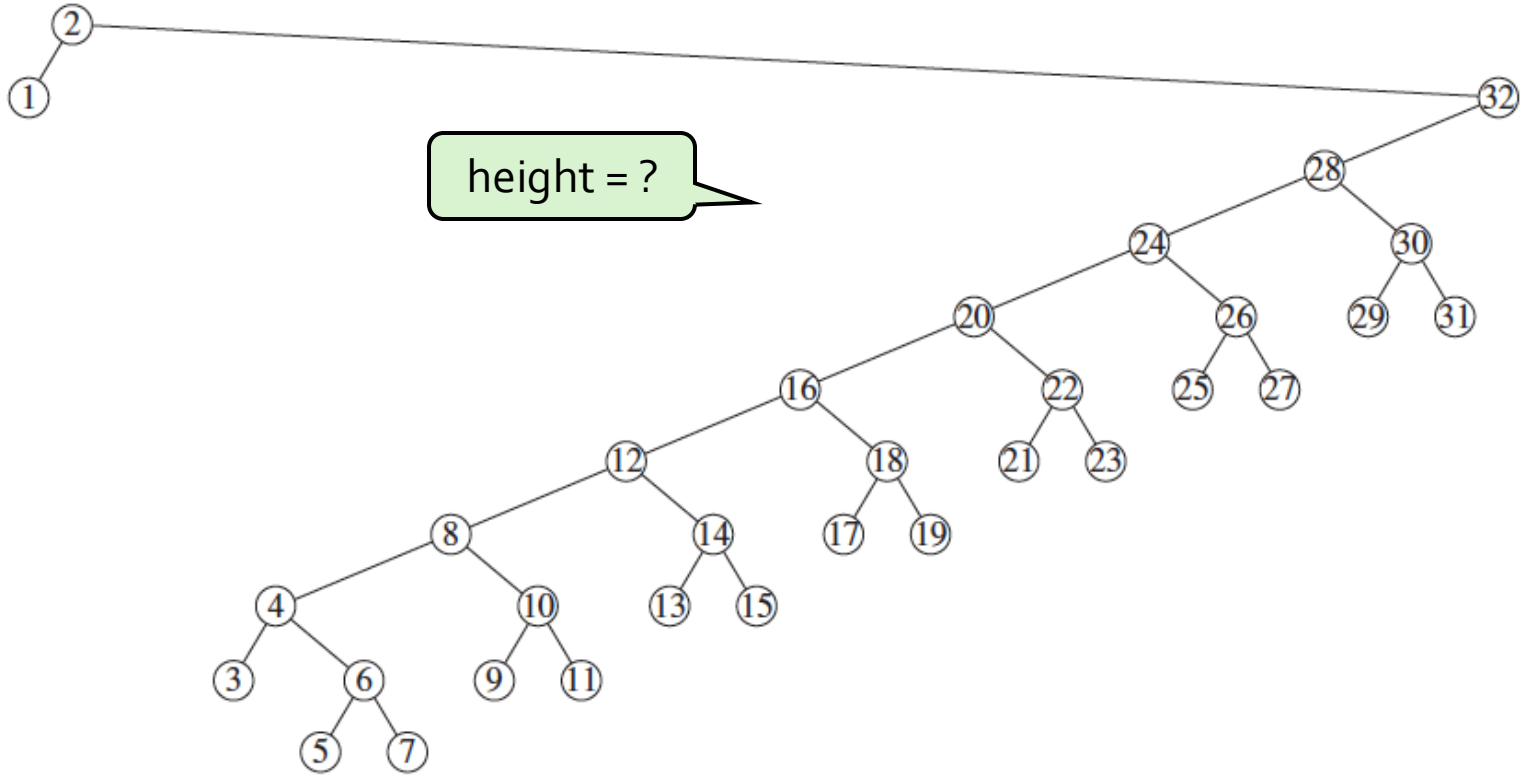


Figure 4.52 Result of splaying the previous tree at node 2

Splay tree

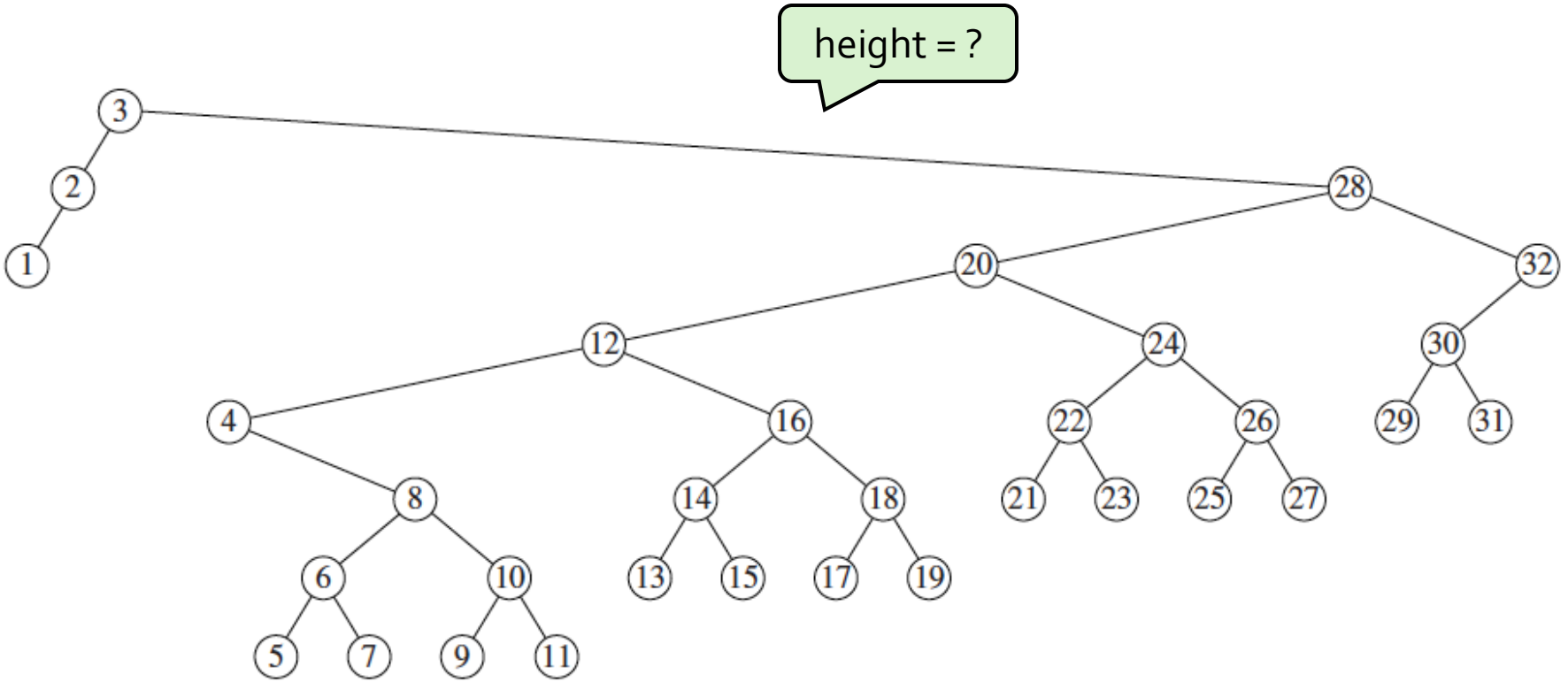


Figure 4.53 Result of splaying the previous tree at node 3

Splay tree

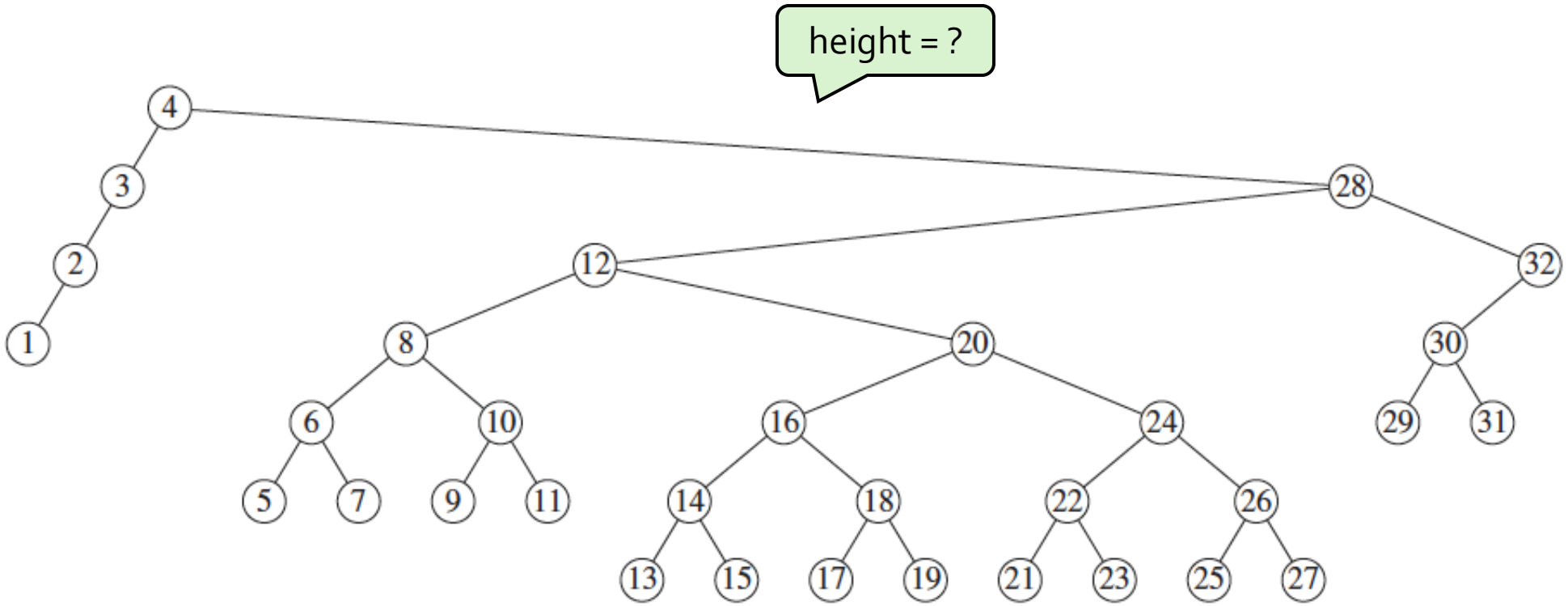


Figure 4.54 Result of splaying the previous tree at node 4

Splay tree

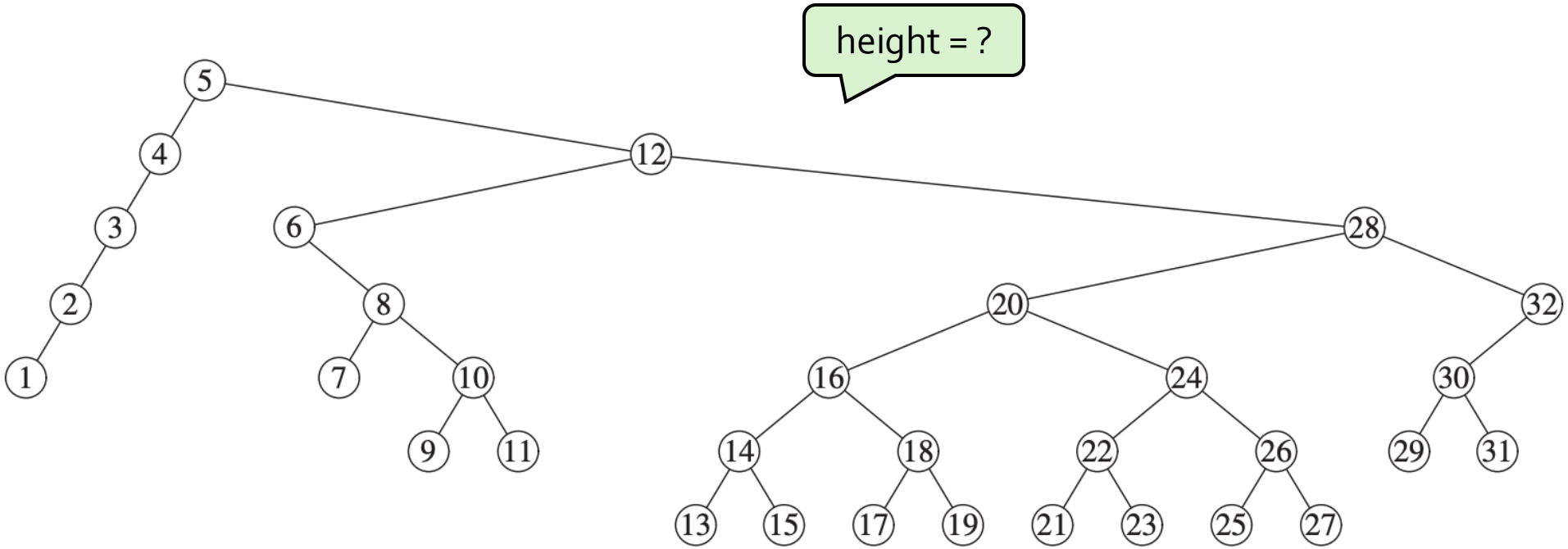


Figure 4.55 Result of splaying the previous tree at node 5

Splay tree

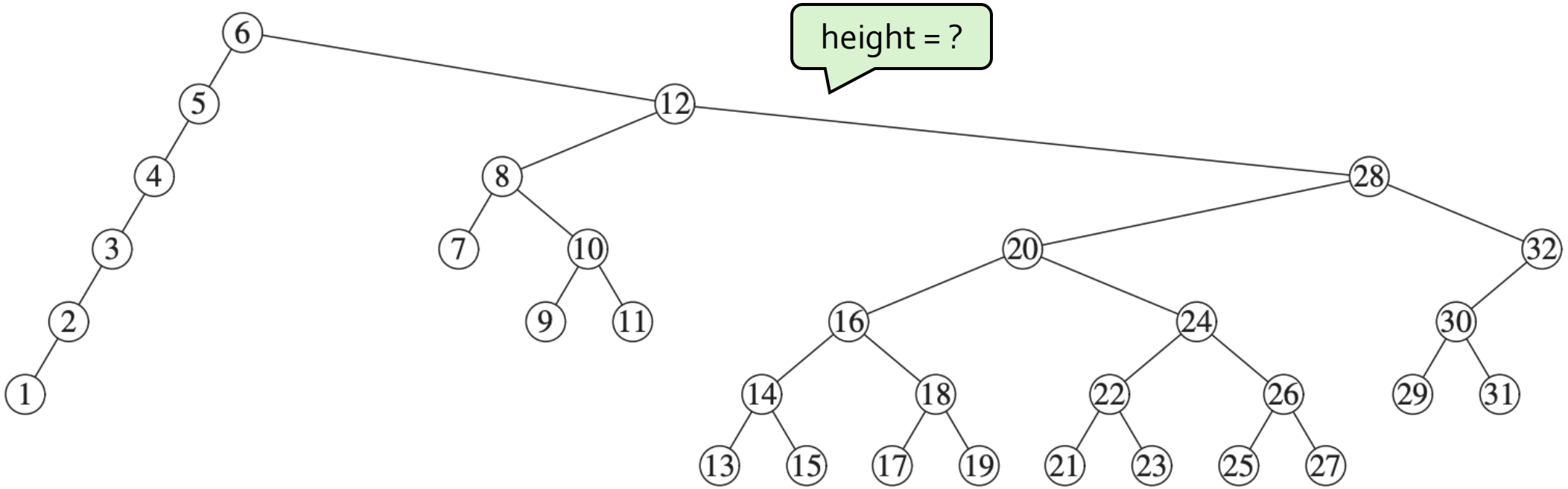


Figure 4.56 Result of splaying the previous tree at node 6

Splay tree

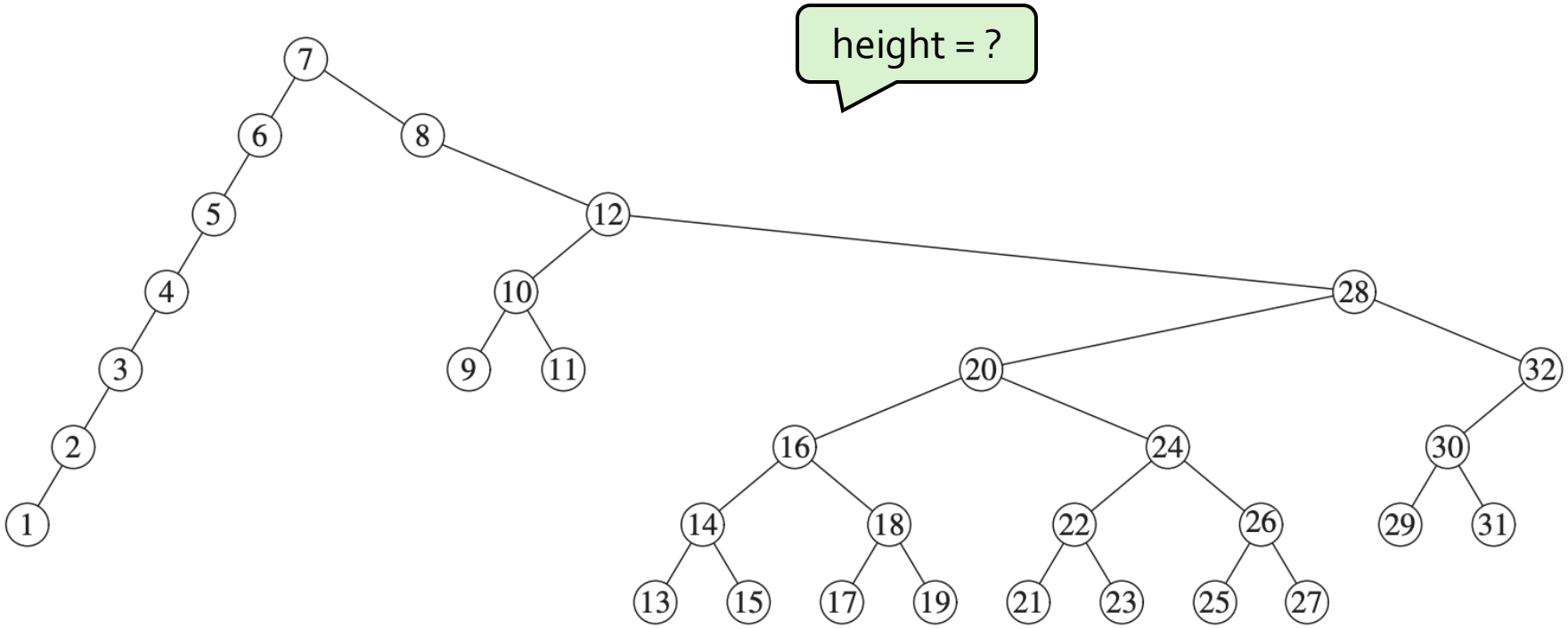


Figure 4.57 Result of splaying the previous tree at node 7

Splay tree

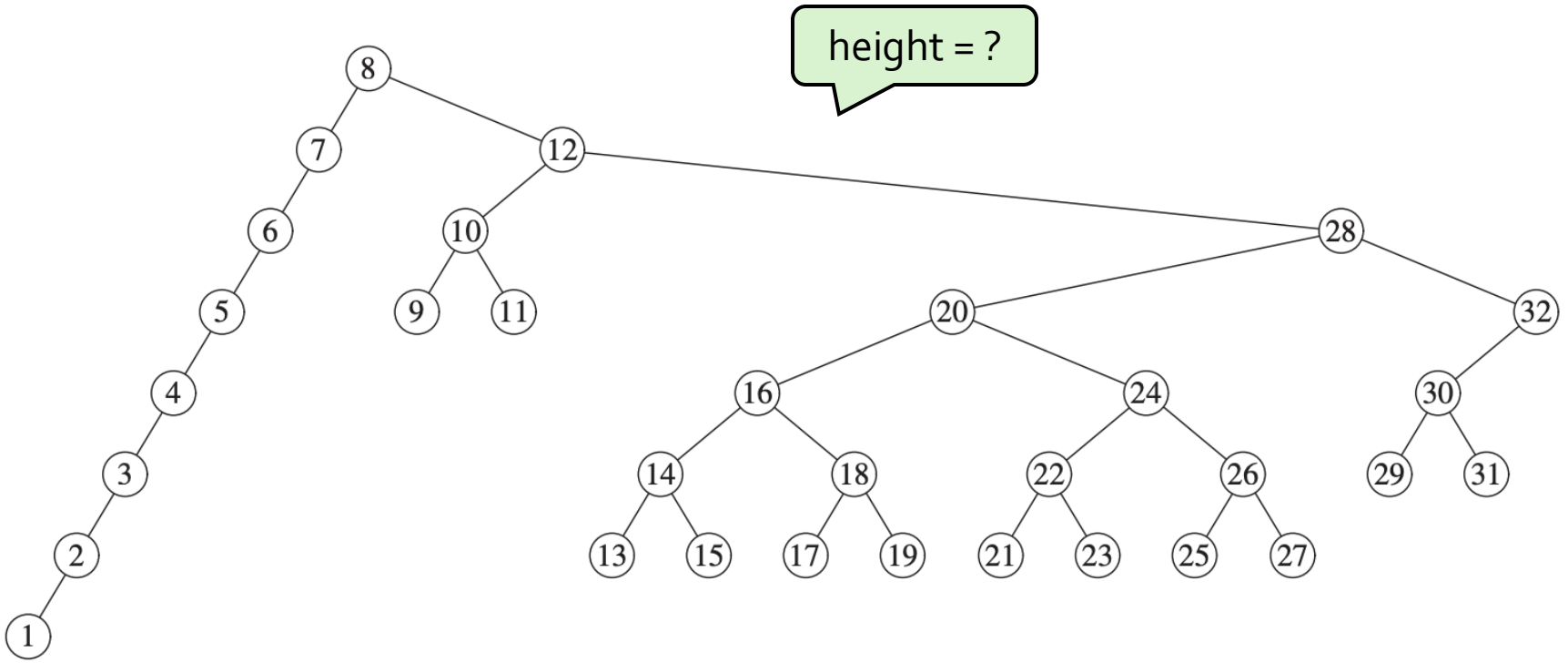


Figure 4.58 Result of splaying the previous tree at node 8

Splay tree

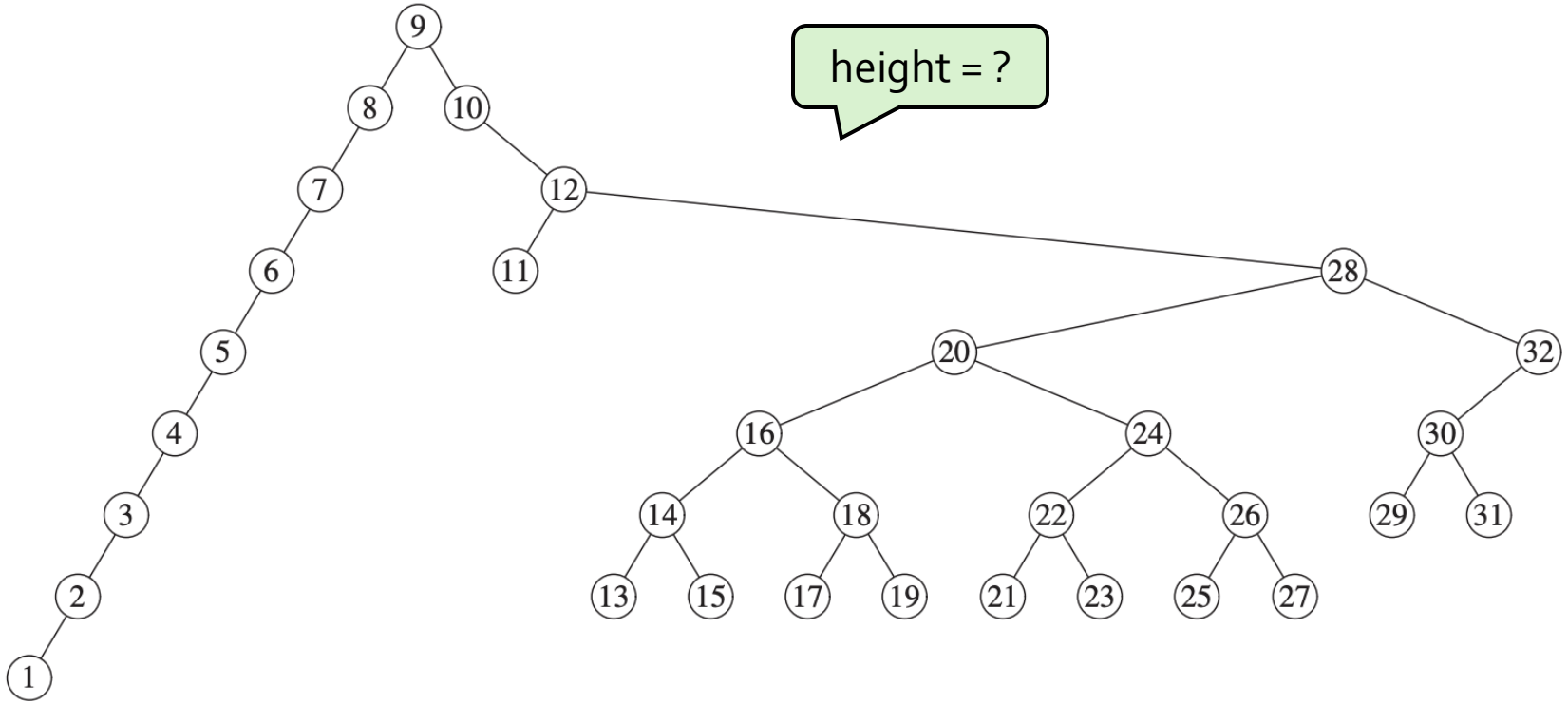


Figure 4.59 Result of splaying the previous tree at node 9

Splay tree: remove

Splay this node (to root)

- Access node to be removed (now at root)
- Remove node leaving two subtrees T_L and T_R
- Access largest element in T_L
  - Now at root

findMax(T_L) and splay this node

- Make T_R right child of root of TL

Splay tree: remove

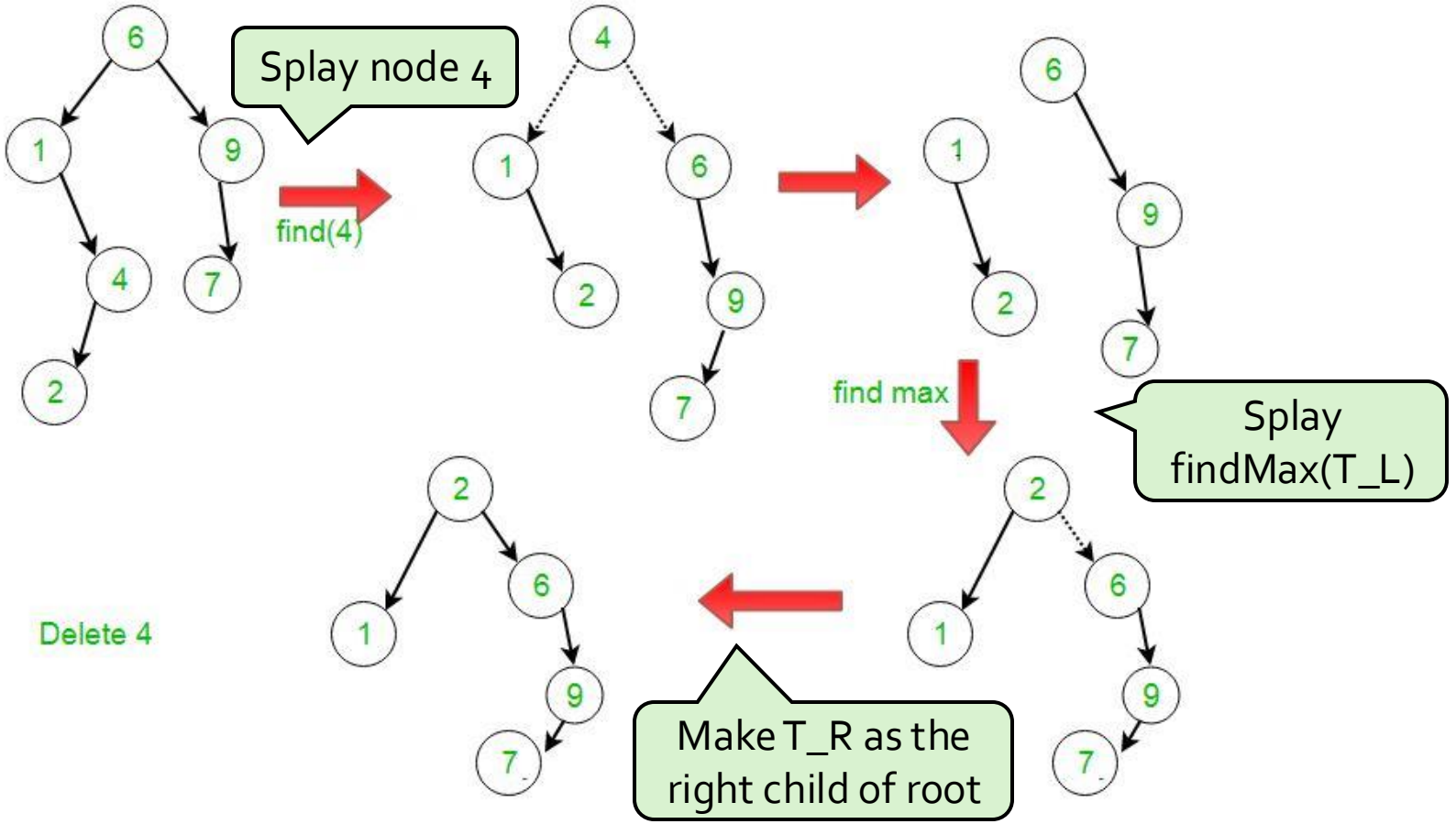


Image credit: https://www.geeksforgeeks.org/deletion-in-splay-tree/