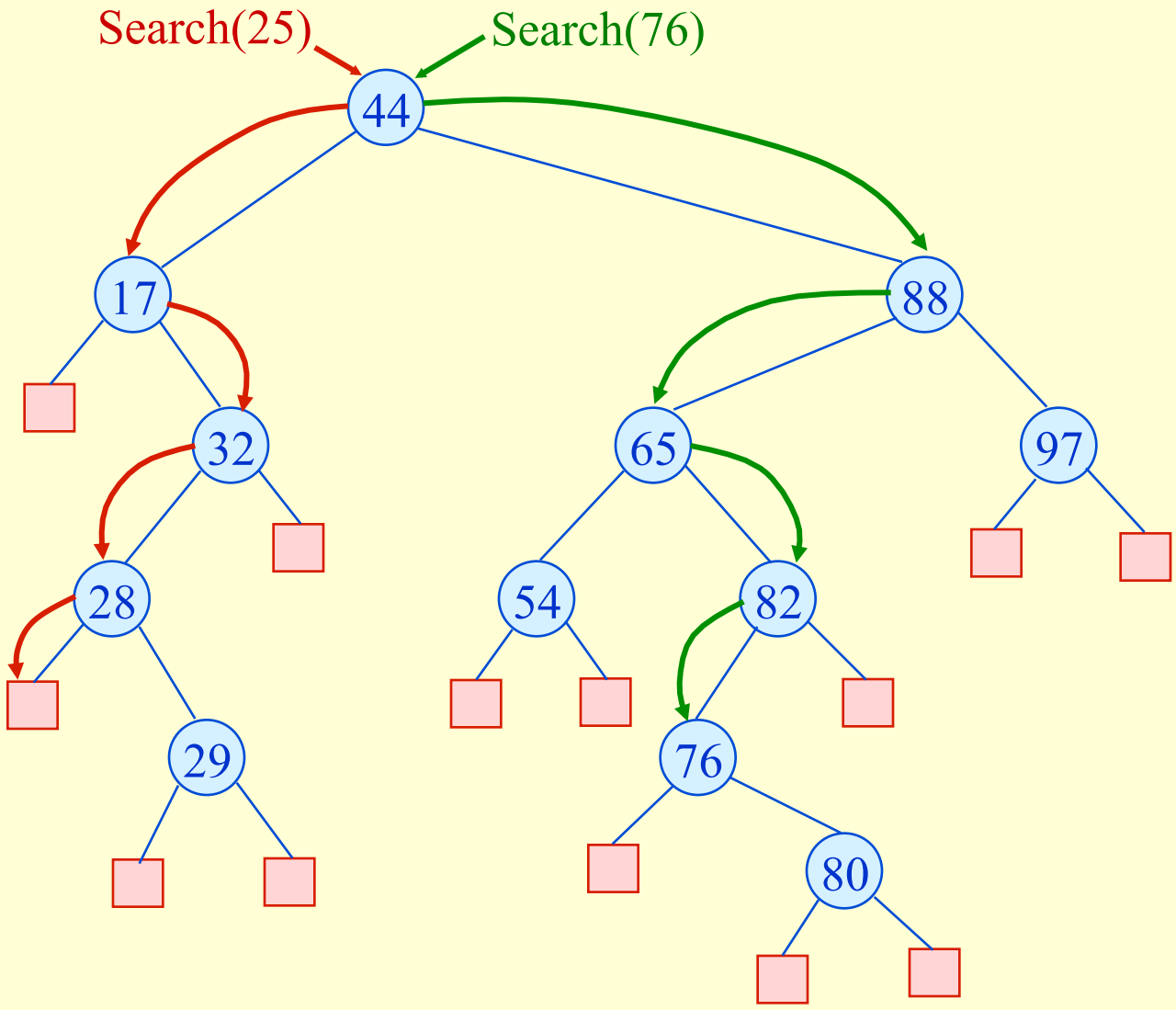
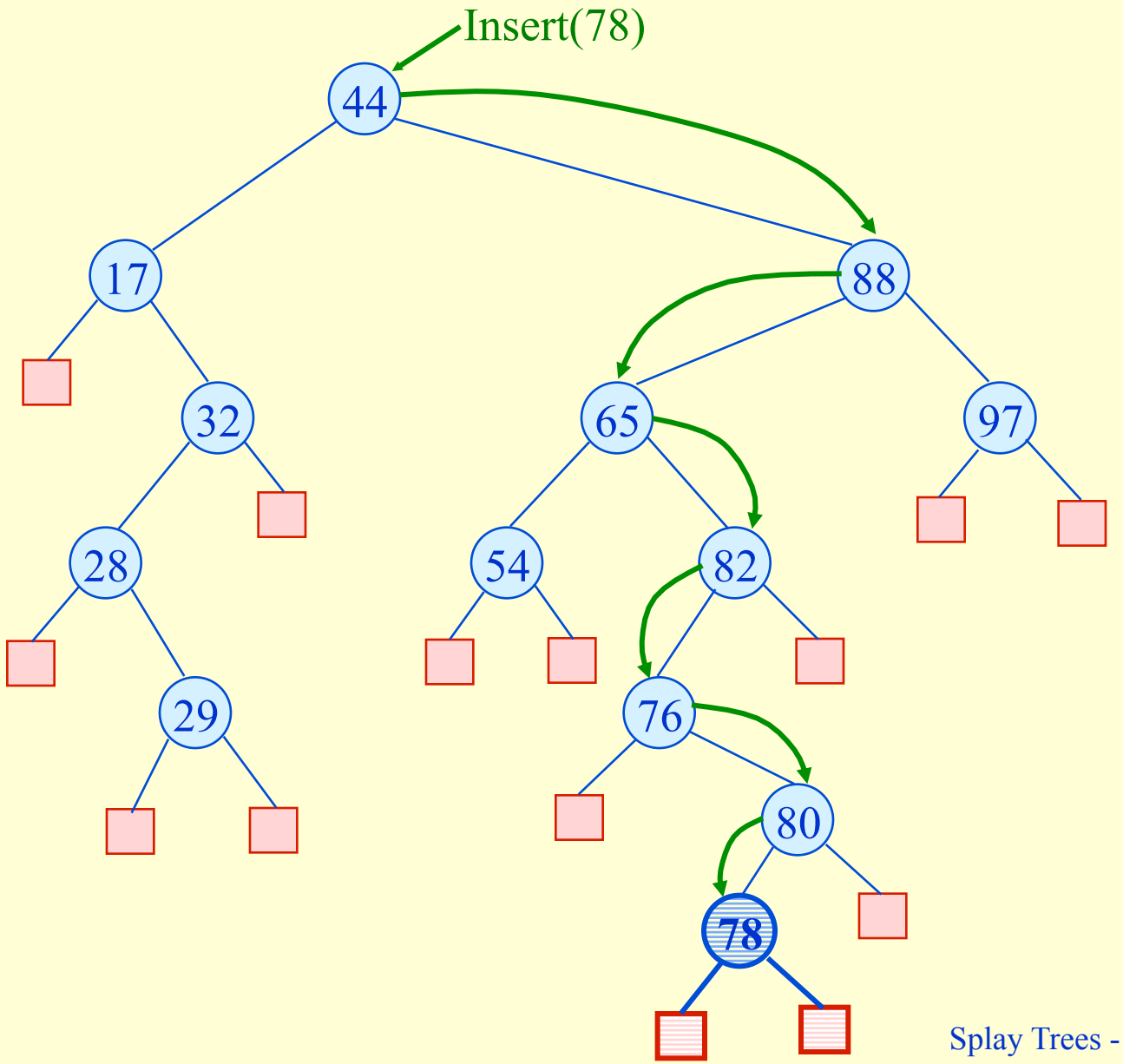
Splay Trees

- In balanced tree schemes, explicit rules are followed to ensure balance.
- In splay trees, there are no such rules.
- Search, insert, and delete operations are like in binary search trees, except at the end of each operation a special step called **splaying** is done.
- Splaying ensures that all operations take O(lg n) amortized time.
- First, a quick review of BST operations…

BST: Search



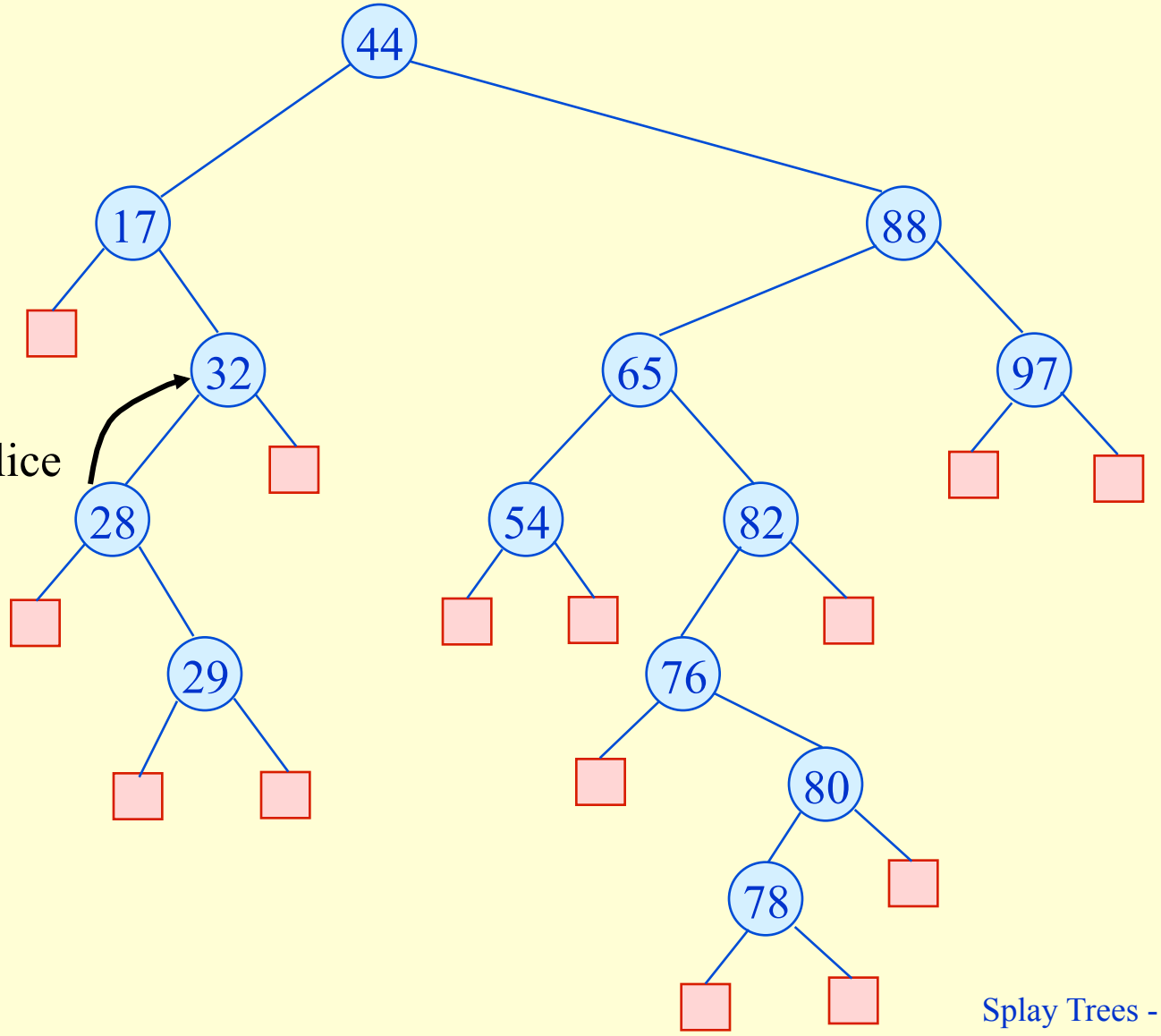
BST: Insert



BST: Delete

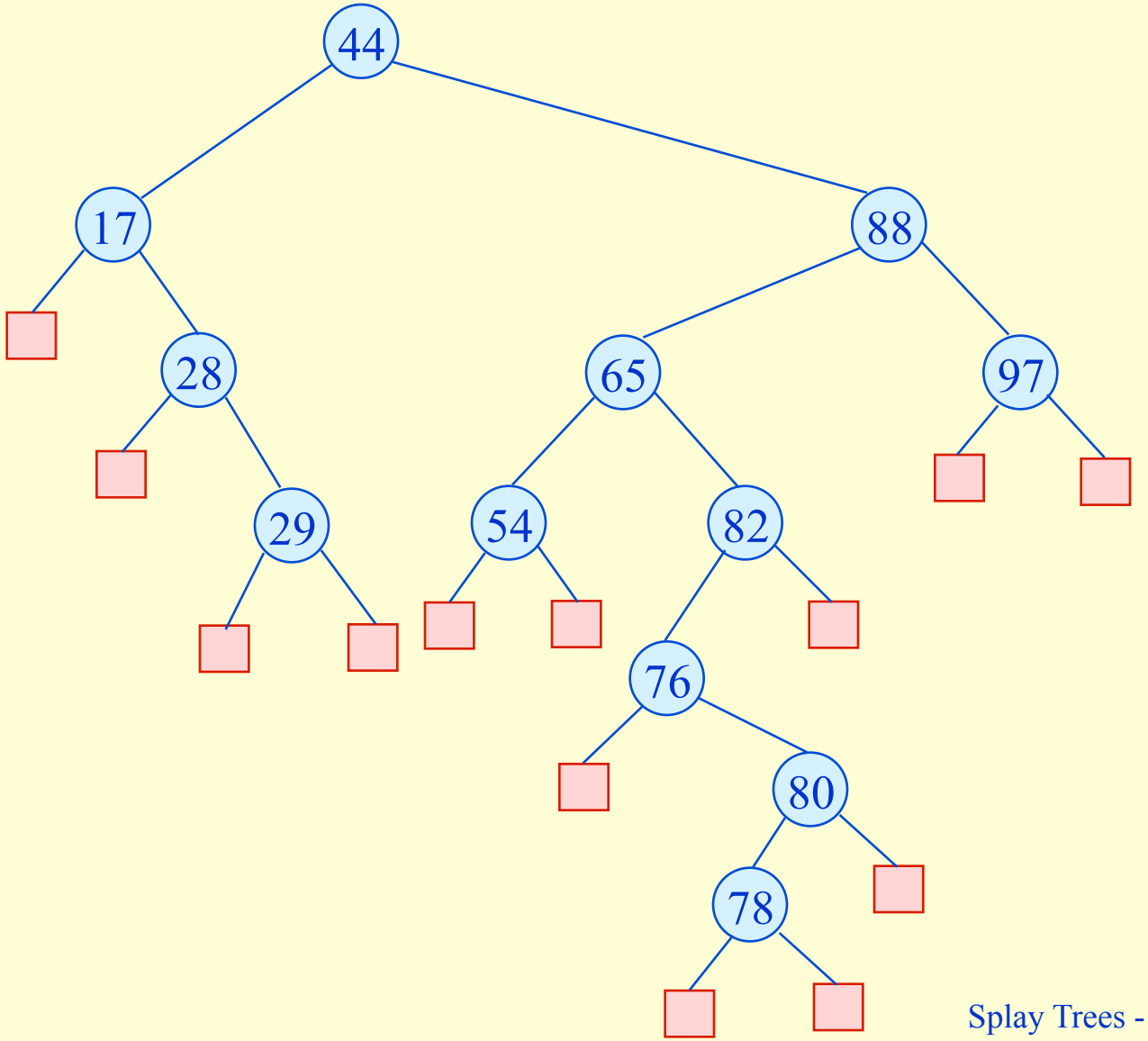
Delete(32)

Has only one child: just splice out 32.



BST: Delete

Delete(32)

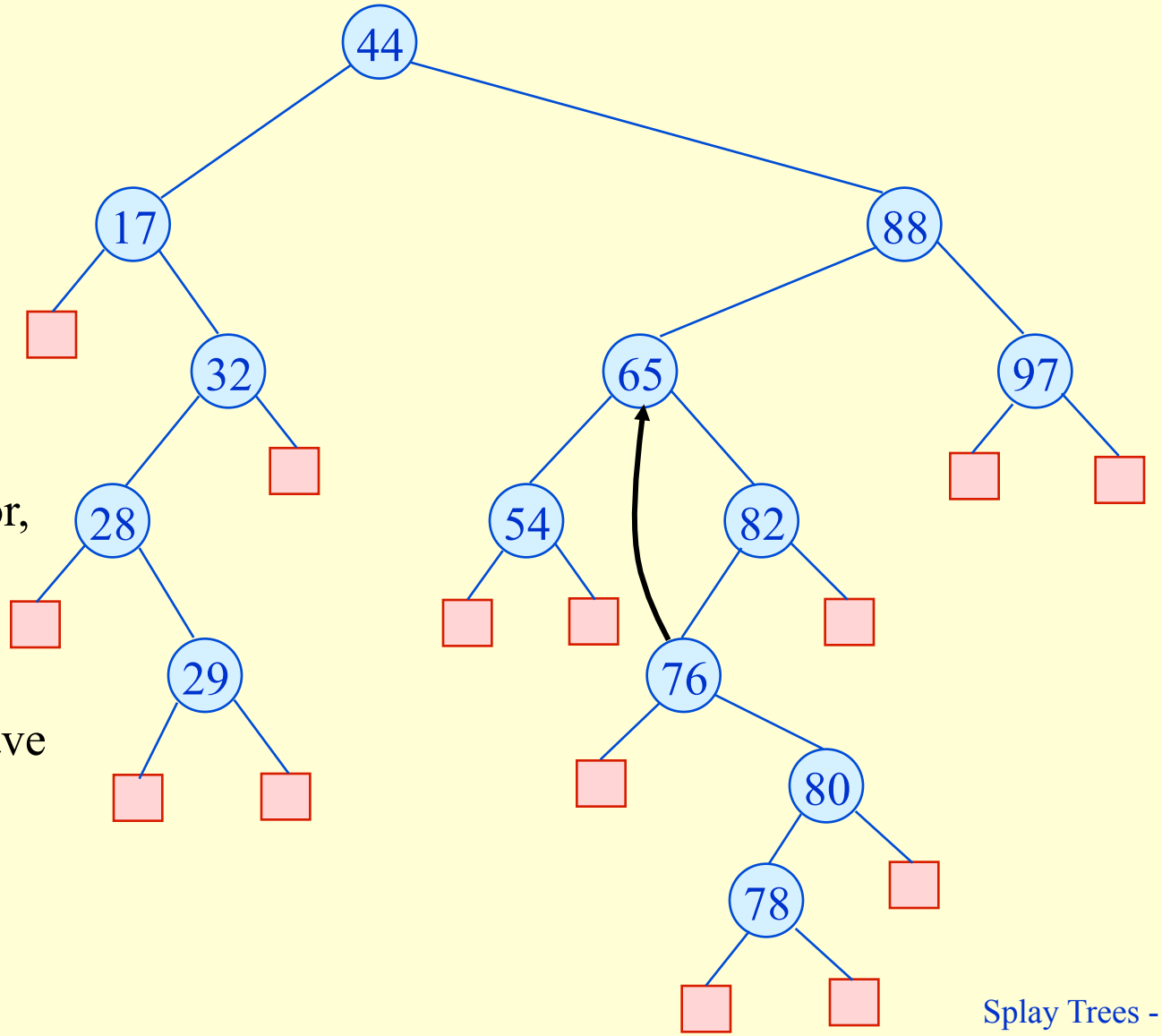


BST: Delete

Delete(65)

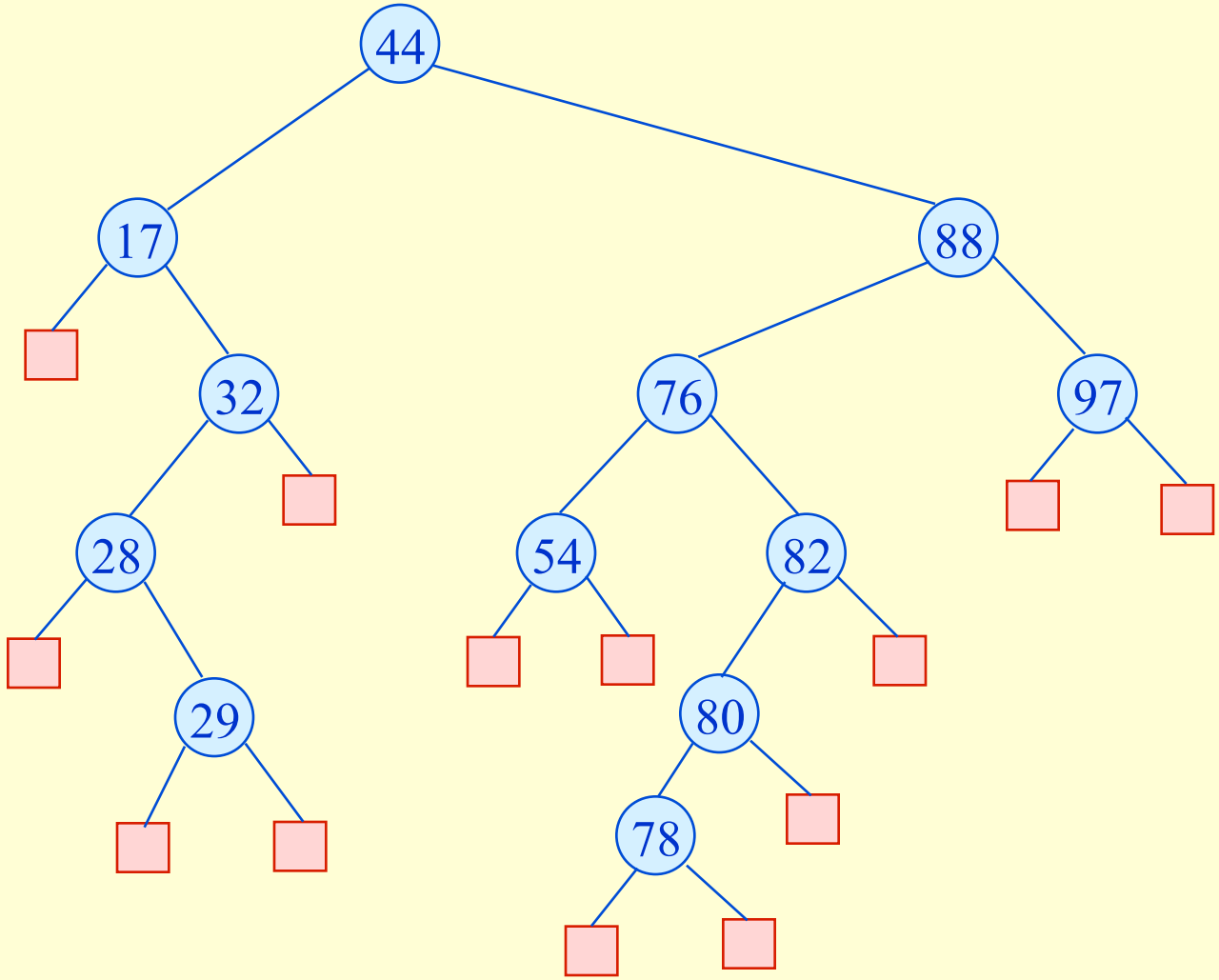
Has two children:
Replace 65 by successor, 76, and splice out successor.

Note: Successor can have at most one child.



BST: Delete

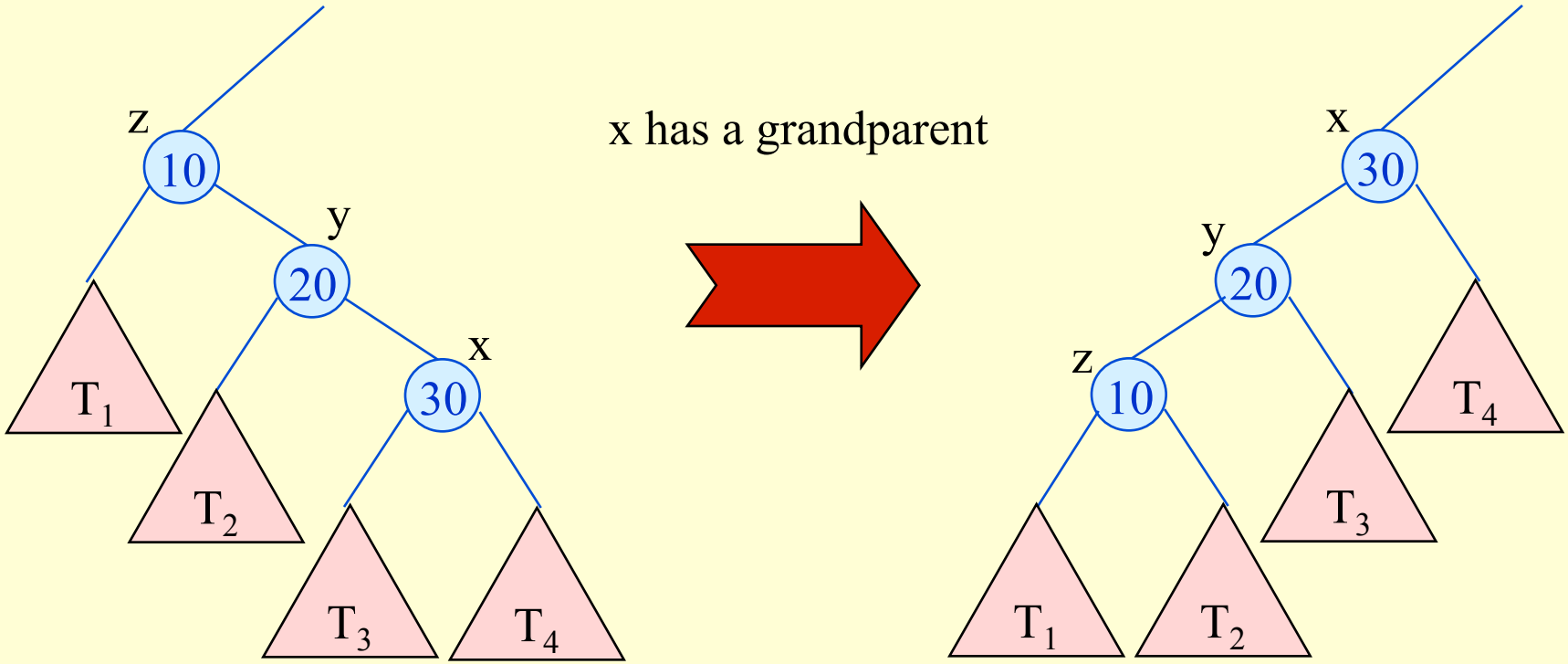
Delete(65)



Splaying

- In splay trees, after performing an ordinary BST Search, Insert, or Delete, a **splay operation** is performed on some node x (as described later).
- The splay operation moves x to the root of the tree.
- The splay operation consists of sub-operations called zig-zig, zig-zag, and zig.

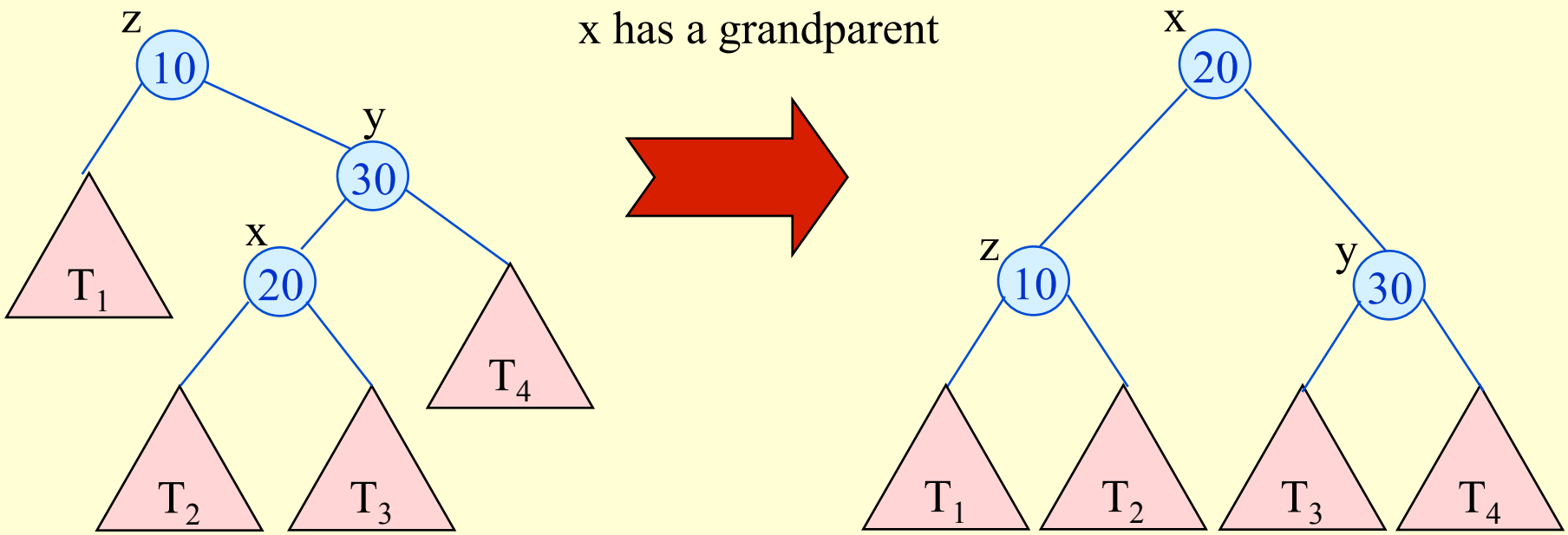
Zig-Zig



(Symmetric case too)

Note: x's depth decreases by two.

Zig-Zag



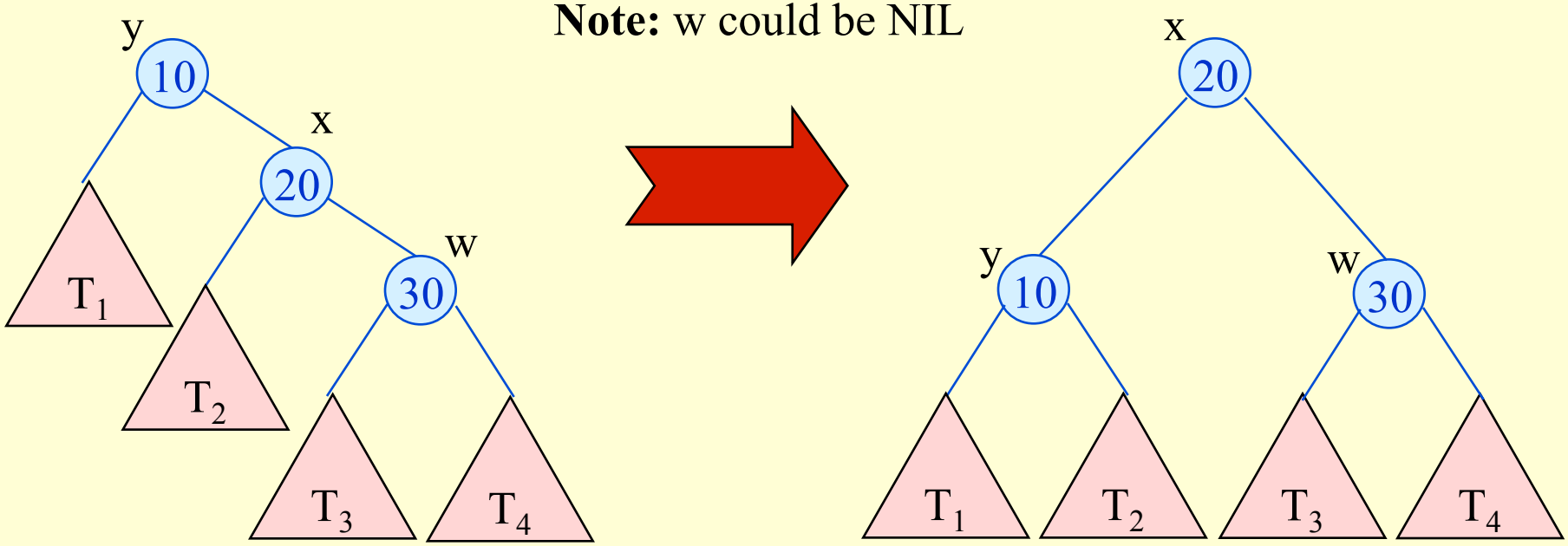
(Symmetric case too)

Note: x's depth decreases by two.

Zig

x has no grandparent (so, y is the root)

Note: w could be NIL



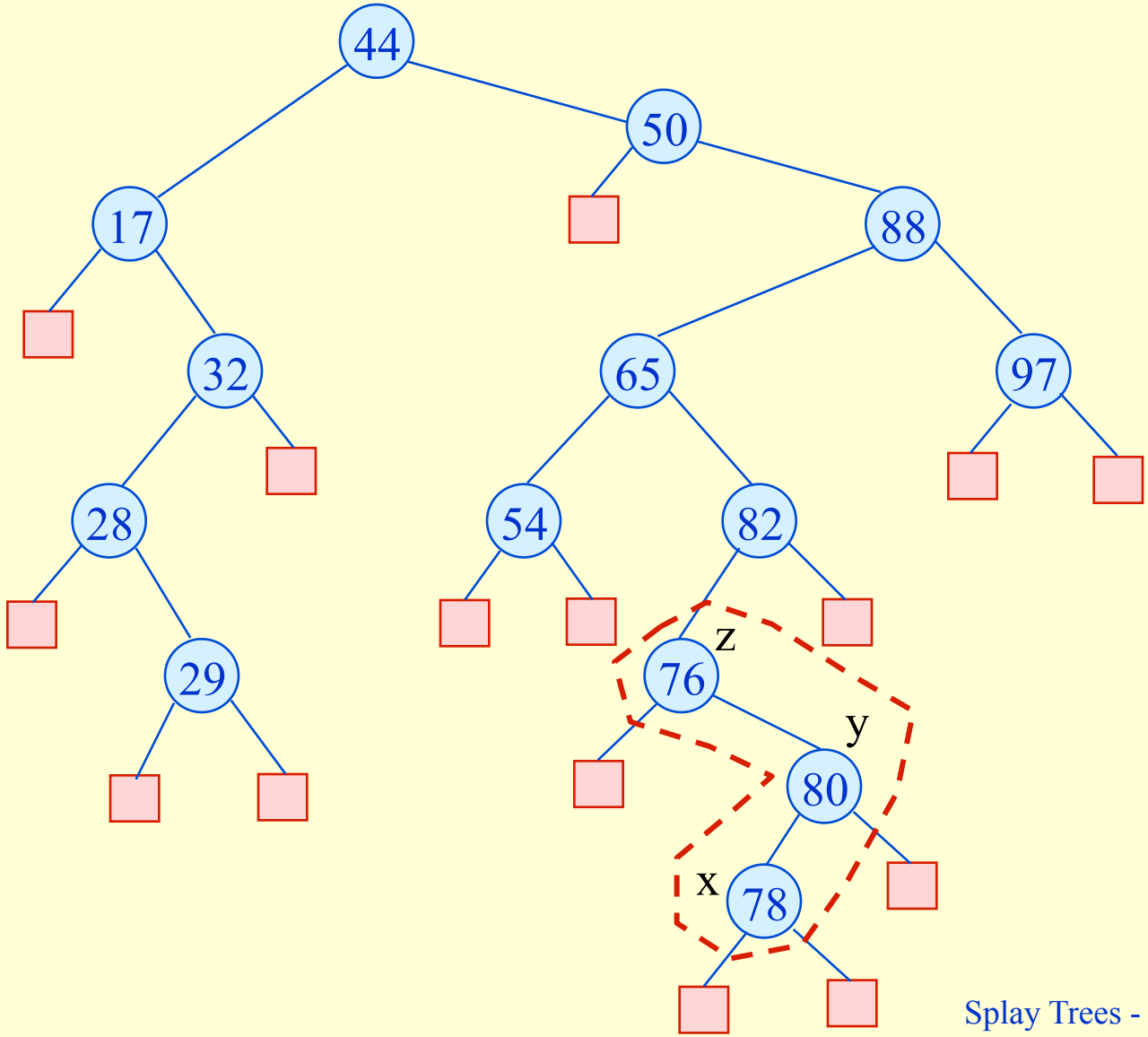
(Symmetric case too)

Note: x's depth decreases by one.

Complete Example

Splay(78)

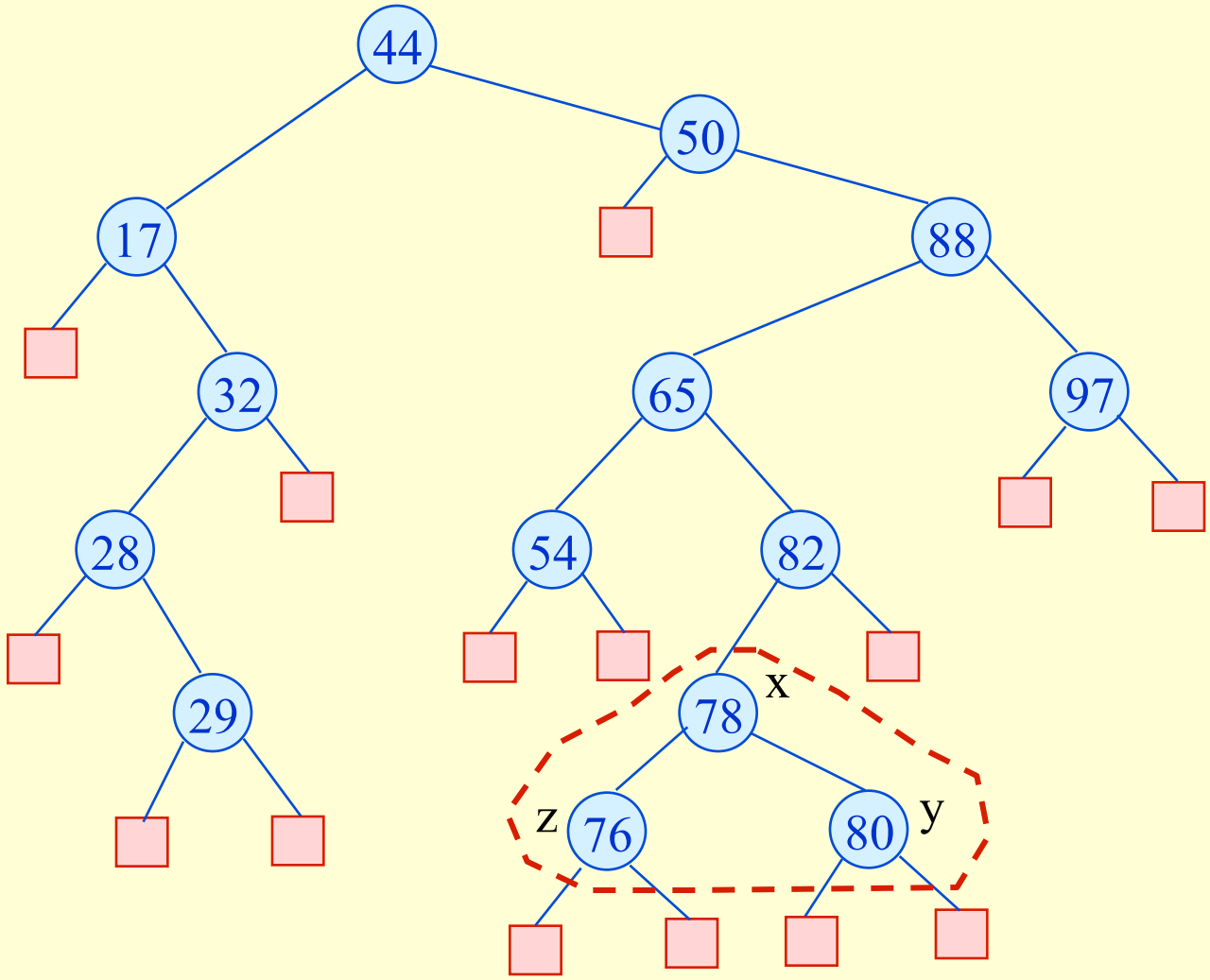
zig-zag



Complete Example

Splay(78)

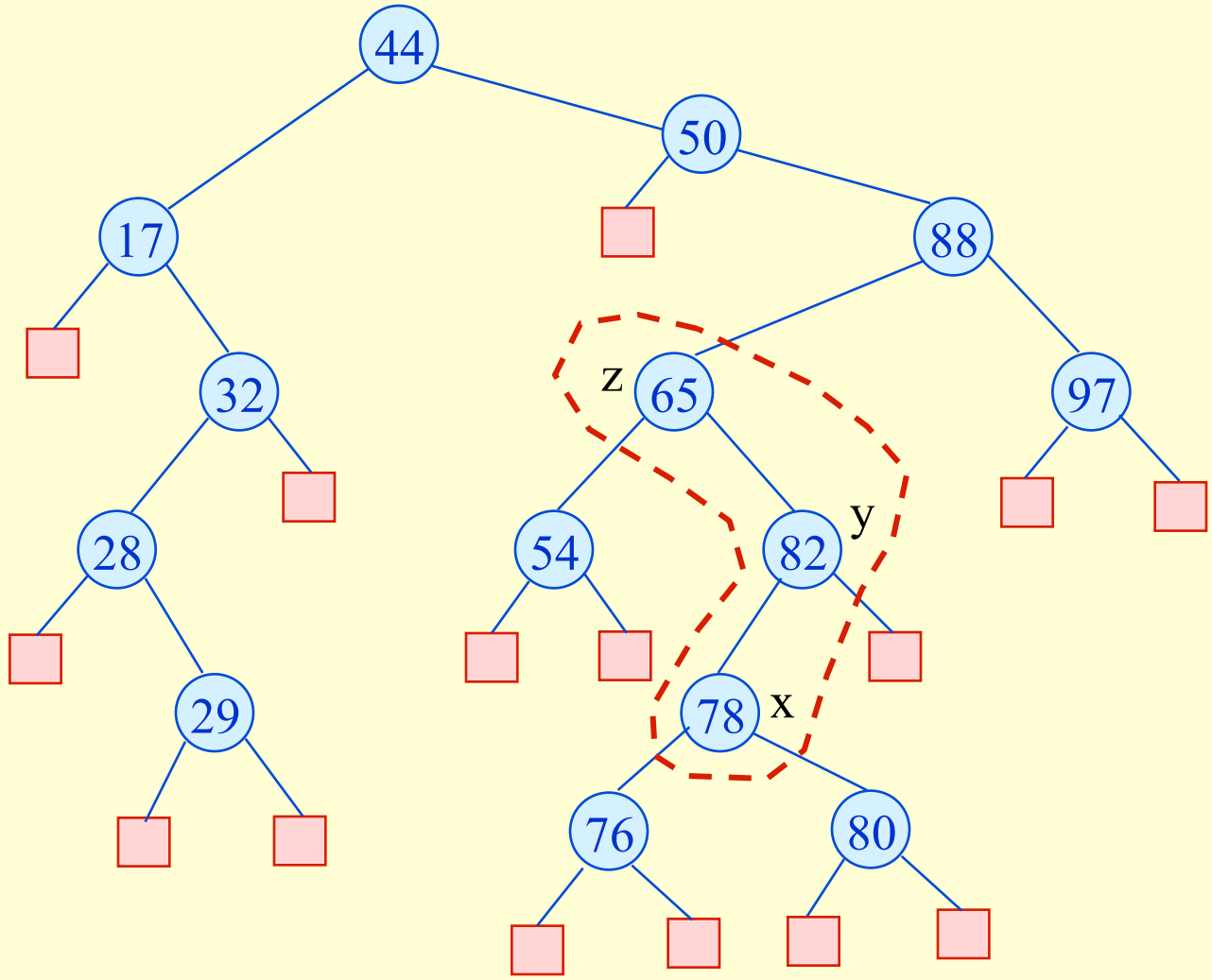
zig-zag



Complete Example

Splay(78)

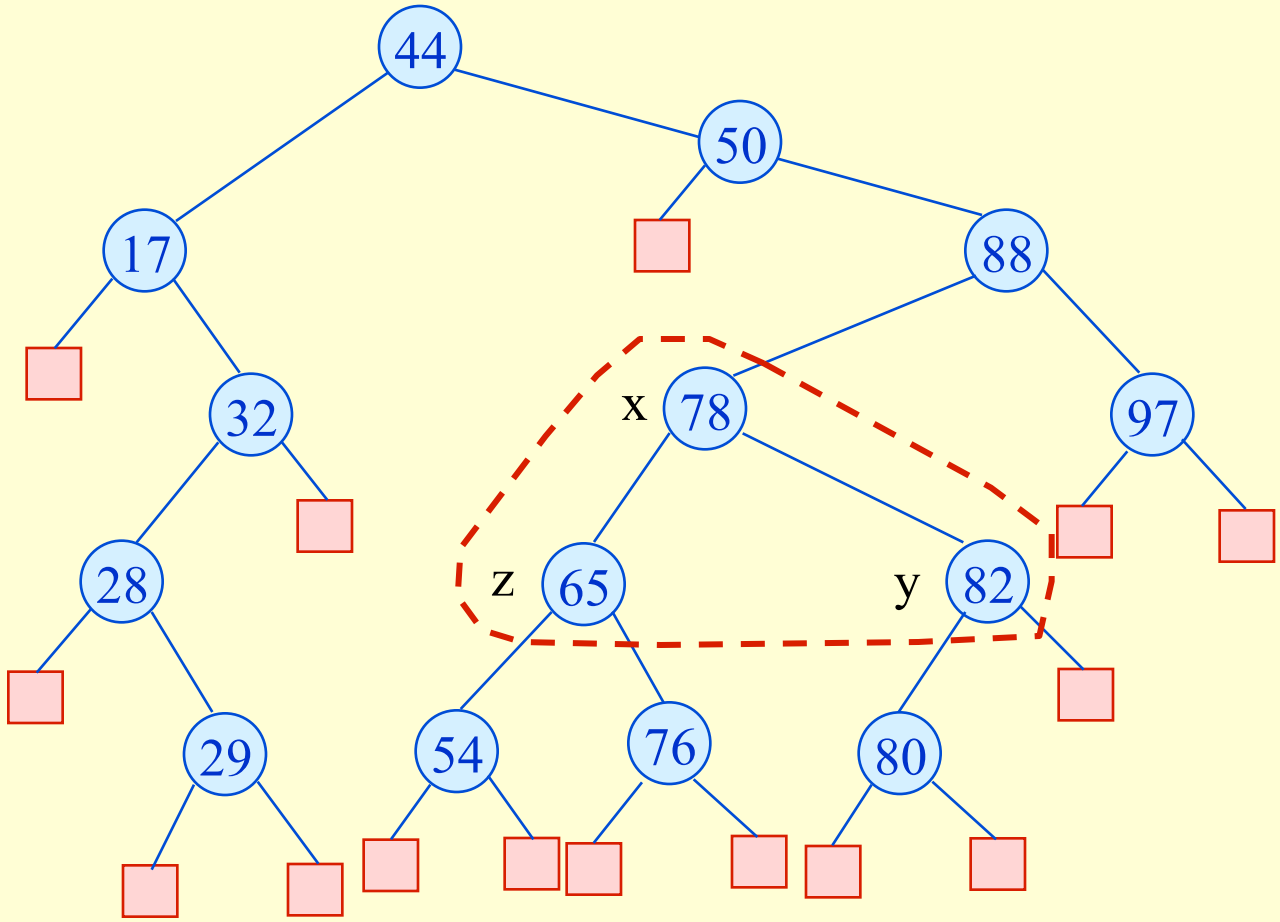
zig-zag



Complete Example

Splay(78)

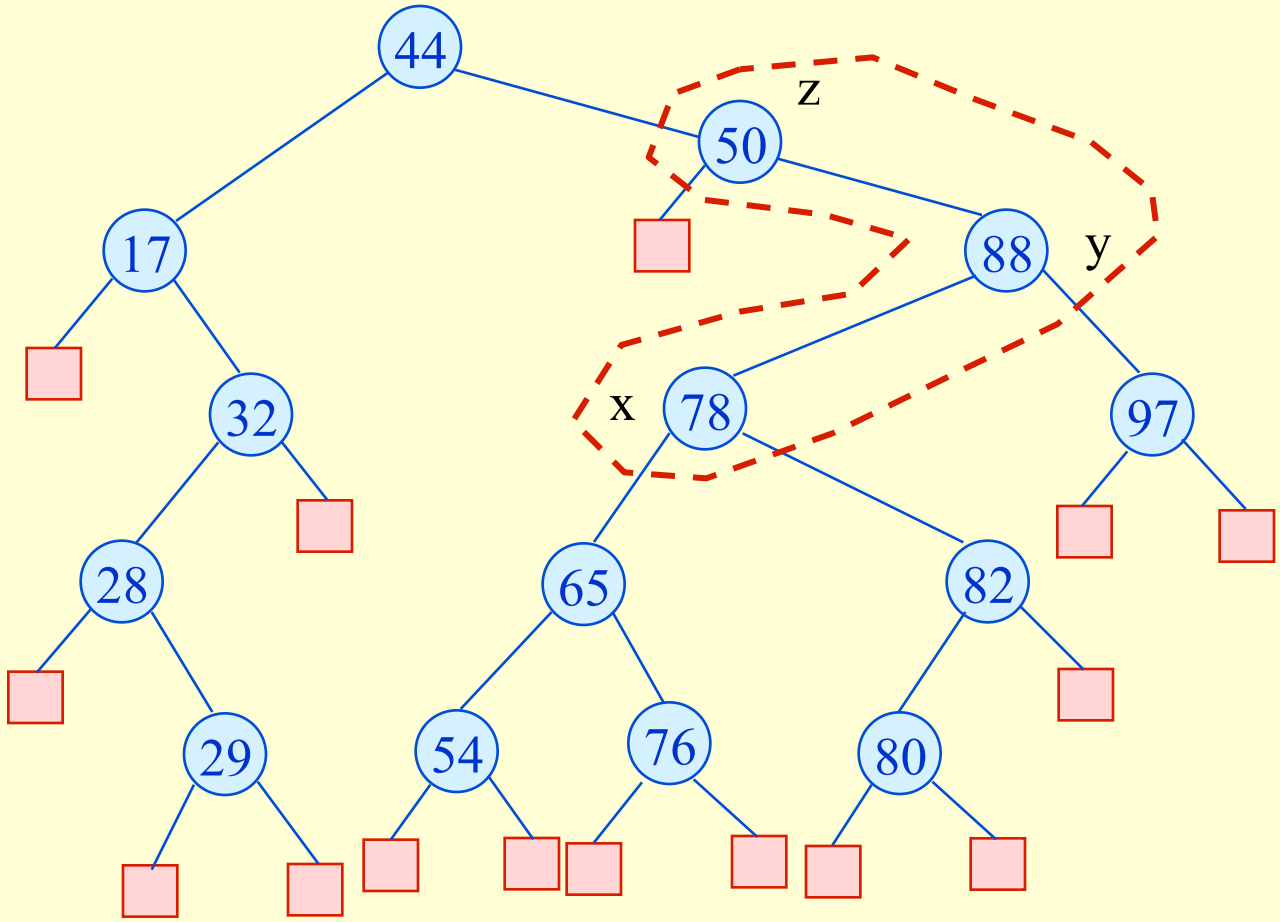
zig-zag



Complete Example

Splay(78)

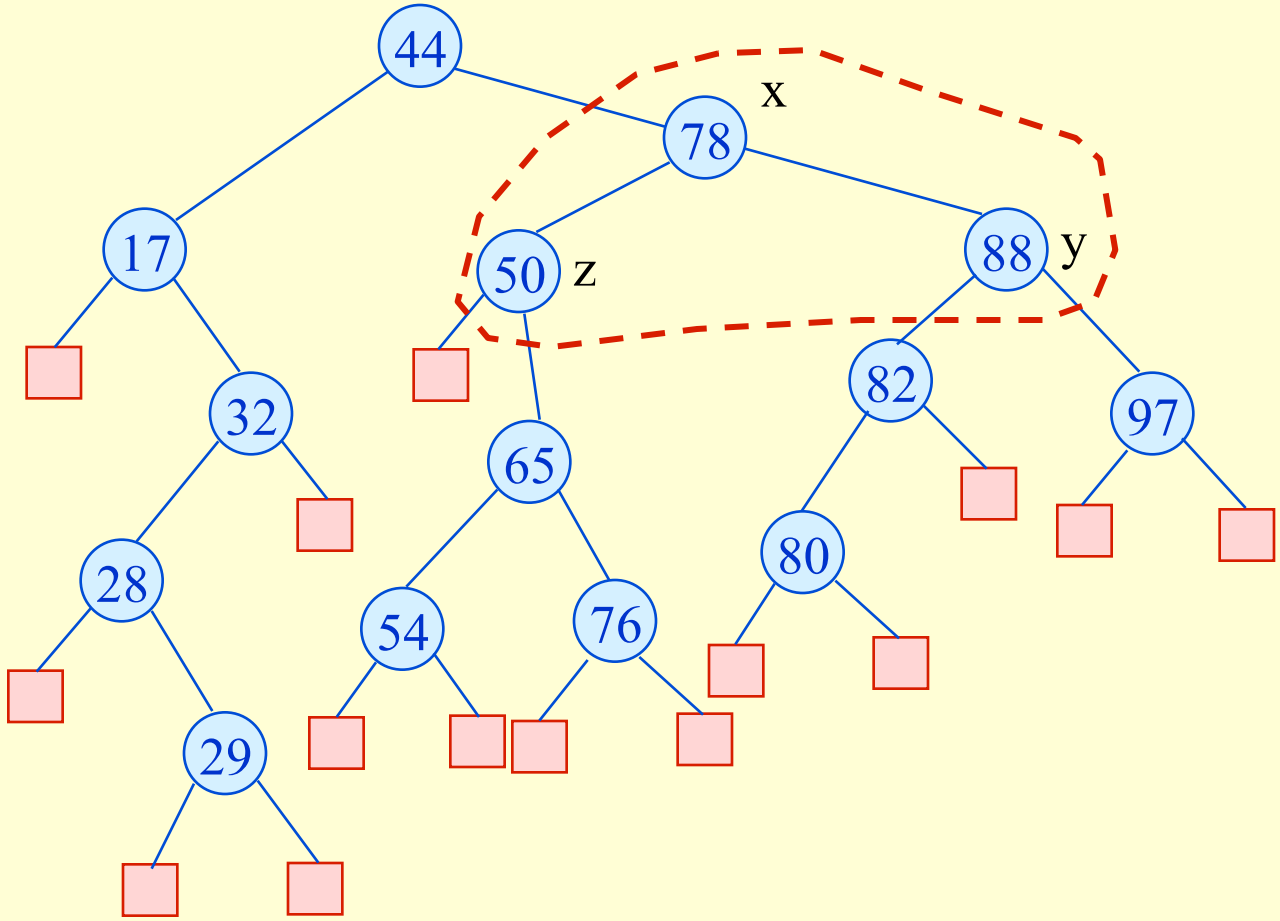
zig-zag



Complete Example

Splay(78)

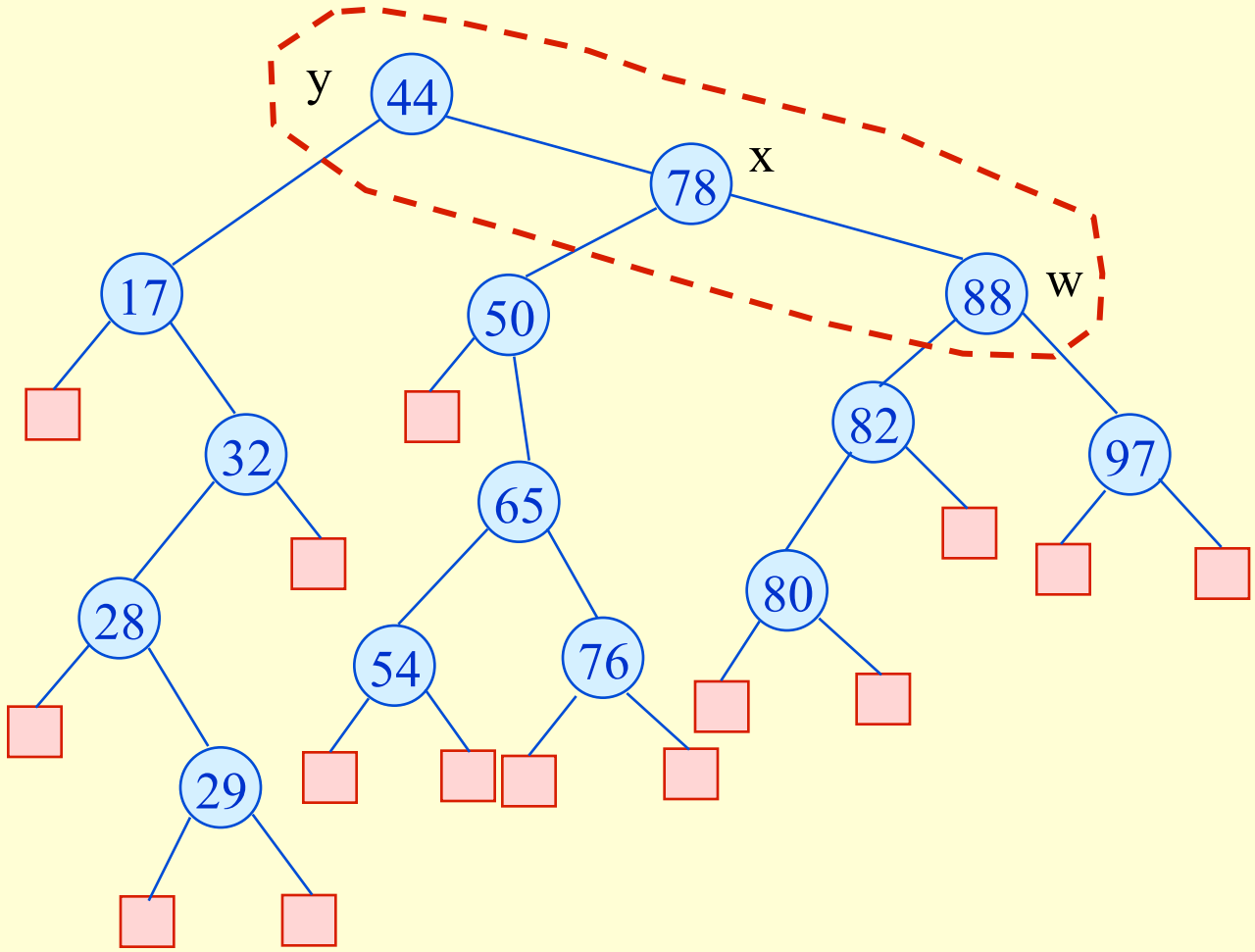
zig-zag



Complete Example

Splay(78)

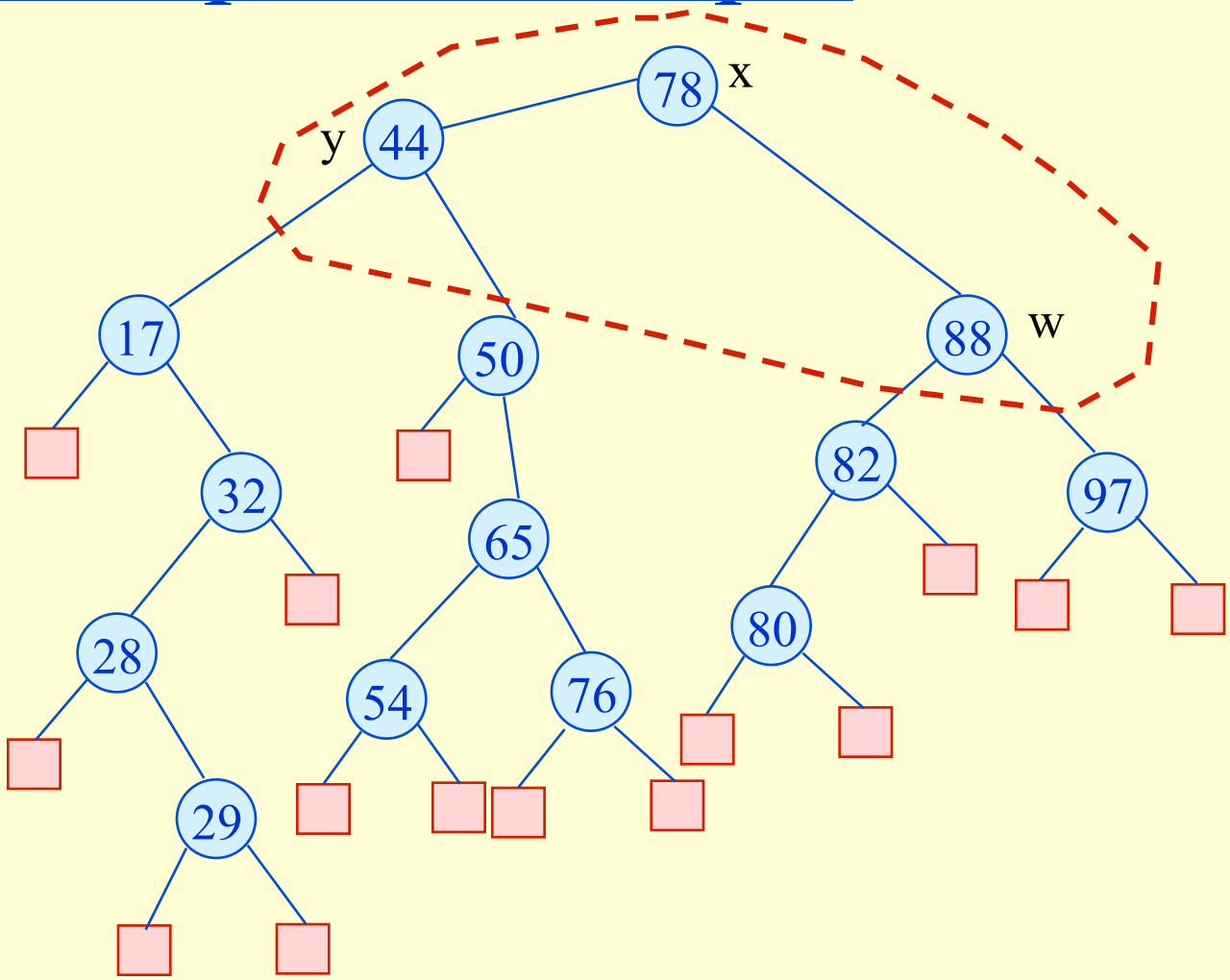
zig



Complete Example

Splay(78)

zig



Result of splaying

- The result is a binary tree, with the left subtree having all keys less than the root, and the right subtree having keys greater than the root.
- Also, the final tree is "more balanced" than the original.
- However, if an operation near the root is done, the tree can become less balanced.

When to Splay

- **Search:**
  - Successful: Splay node where key was found.
  - Unsuccessful: Splay last-visited internal node (i.e., last node with a key).
- **Insert:**
  - Splay newly added node.
- **Delete:**
  - Splay parent of removed node (which is either the node with the deleted key or its successor).
- **Note:** All operations run in O(h) time, for a tree of height h.