Advanced Data Structures

Splay Tree

- A perfectly balanced BST might not be the best BST for a particular data set if the accesses aren't uniform.

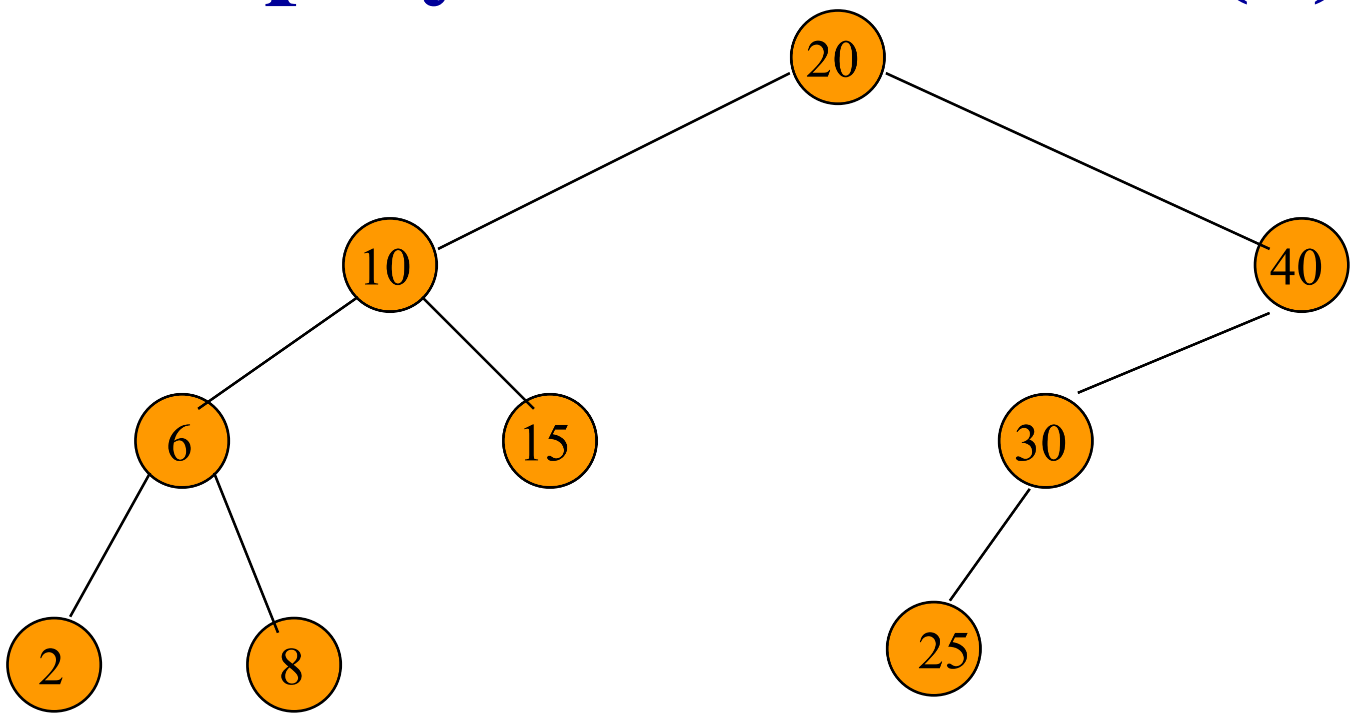
Splay Trees

- Binary search trees.
- Search, insert, delete, and split have amortized complexity O(log n) & actual complexity O(n).
- Actual and amortized complexity of join is O(1).
- Priority queue and double-ended priority queue versions outperform heaps, deaps, etc. over a sequence of operations.
- Two varieties.
  - Bottom up.
  - Top down.

Bottom-Up Splay Trees

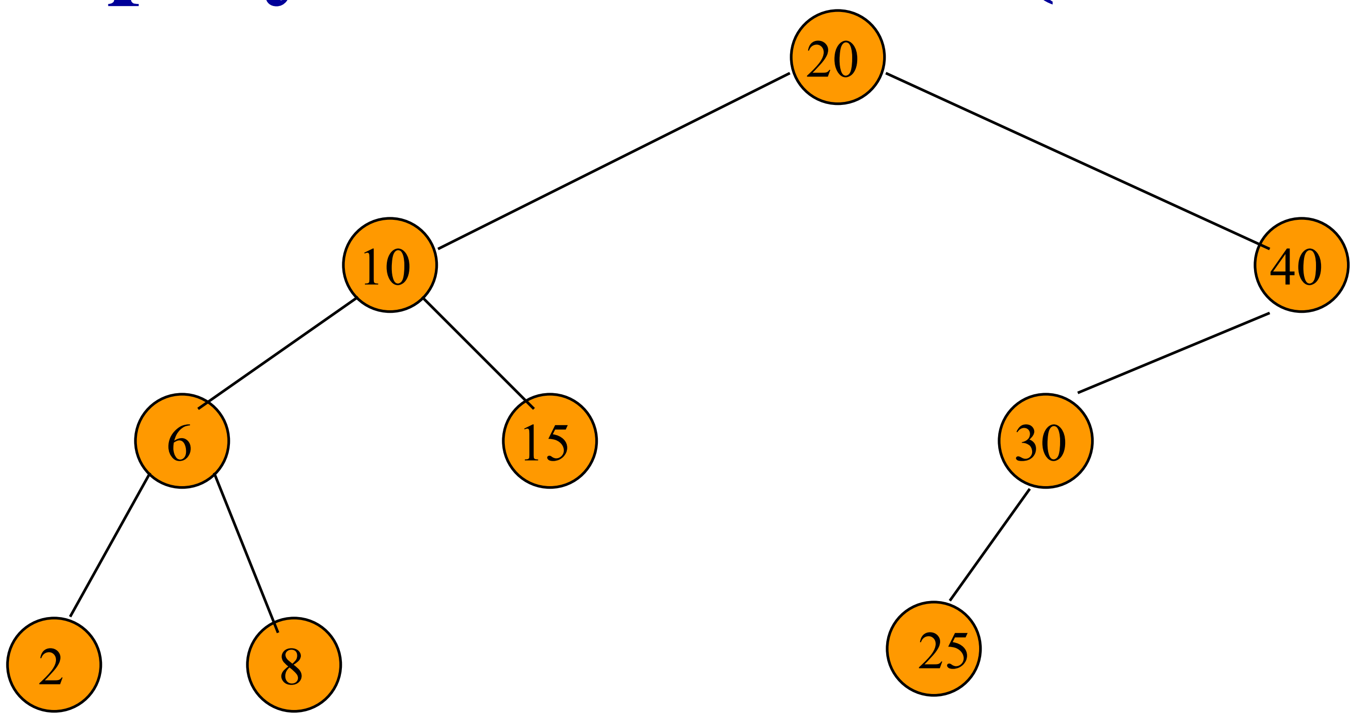
- Search, insert, delete, and join are done as in an unbalanced binary search tree.
- Search, insert, and delete are followed by a splay operation that begins at a splay node.
- When the splay operation completes, the splay node has become the tree root.
- Join requires no splay (or, a null splay is done).
- For the split operation, the splay is done in the middle (rather than end) of the operation.

Splay Node – search(k)



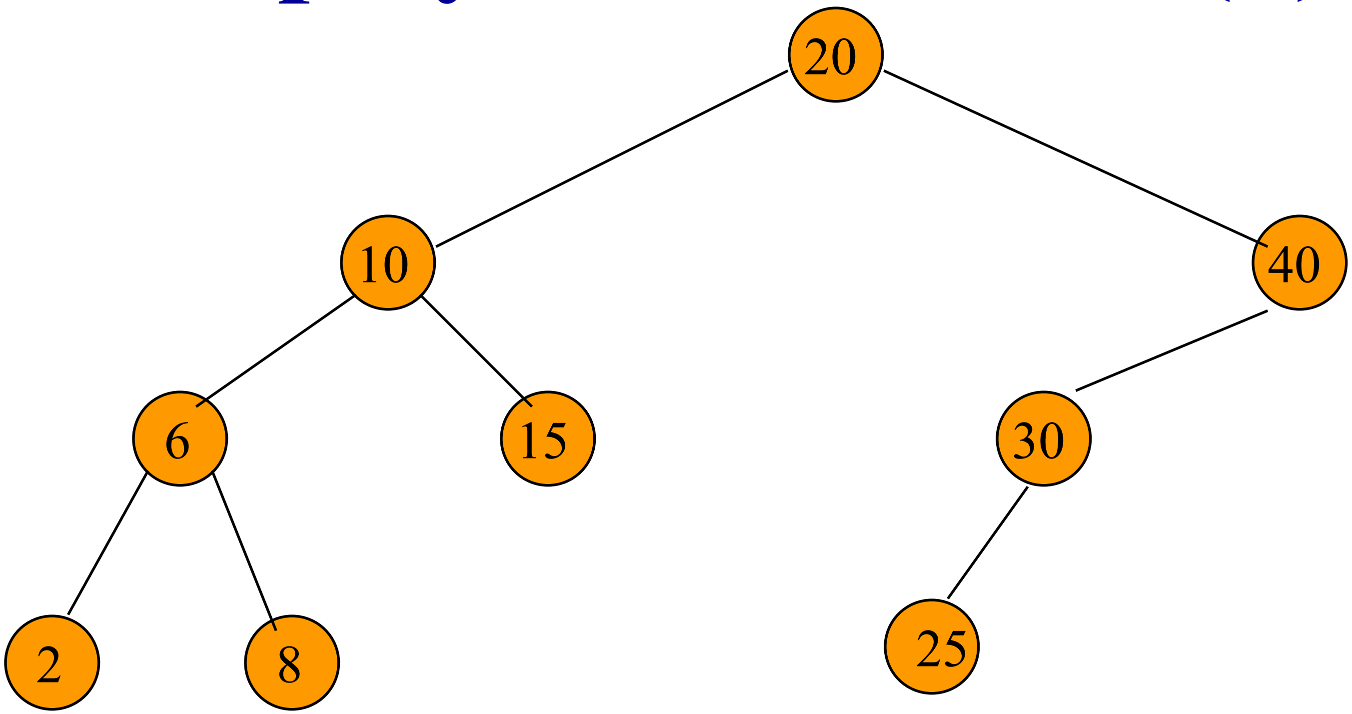
- **If there is a pair whose key is k, the node containing this pair is the splay node.**
- **Otherwise, the parent of the external node where the search terminates is the splay node.**

Splay Node – insert(newPair)



- **If there is already a pair whose key is newPair.key, the node containing this pair is the splay node.**
- **Otherwise, the newly inserted node is the splay node.**

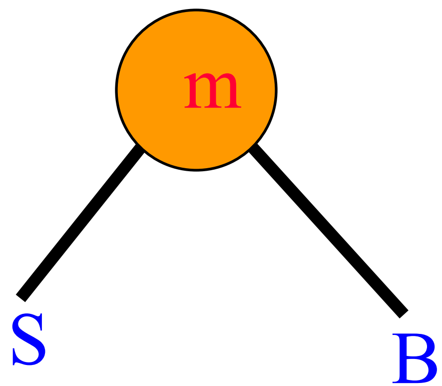
Splay Node – delete(k)



- **If there is a pair whose key is k, the parent of the node that is physically deleted from the tree is the splay node.**
- **Otherwise, the parent of the external node where the search terminates is the splay node.**

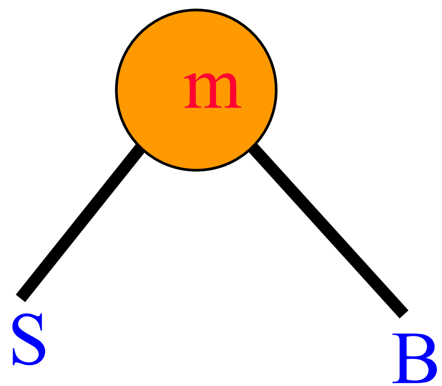
Splay Node – split(k)

- **Use the unbalanced binary search tree insert algorithm to insert a new pair whose key is k.**
- **The splay node is as for the splay tree insert algorithm.**
- **Following the splay, the left subtree of the root is S, and the right subtree is B.**



Splay Node – split(k)

- **Use the unbalanced binary search tree insert algorithm to insert a new pair whose key is k.**
- **The splay node is as for the splay tree insert algorithm.**
- **Following the splay, the left subtree of the root is S, and the right subtree is B.**



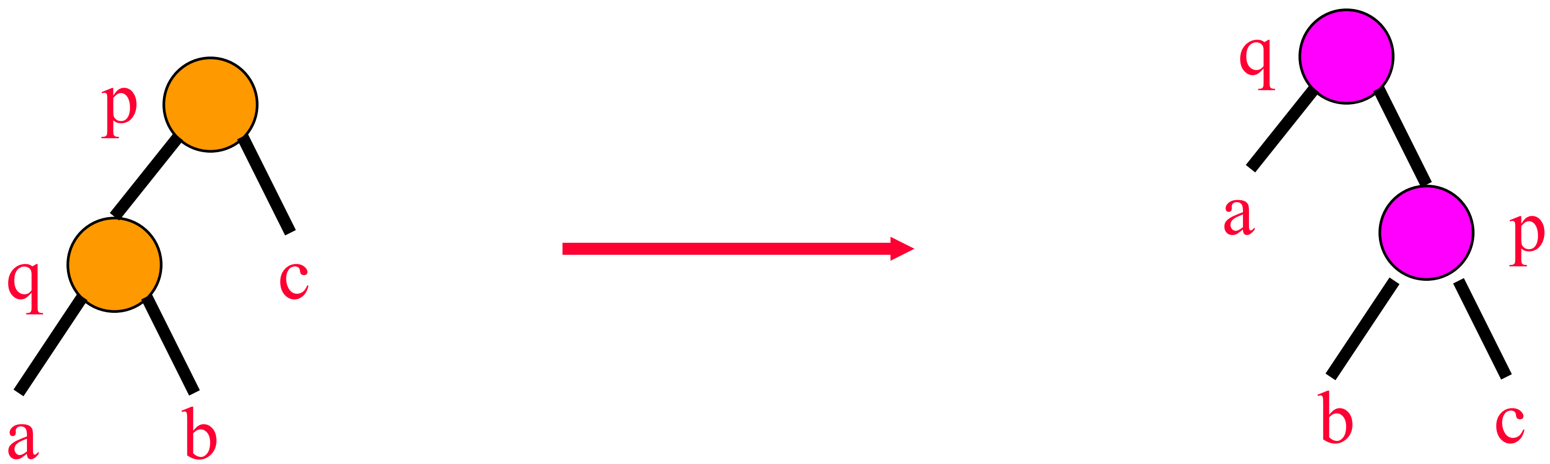
- m is set to null if it is the newly inserted pair.

Splay

- Let q be the splay node.
- q is moved up the tree using a series of splay steps.
- In a splay step, the node q moves up the tree by 0, 1, or 2 levels.
- Every splay step, except possibly the last one, moves q two levels up.

Splay Step

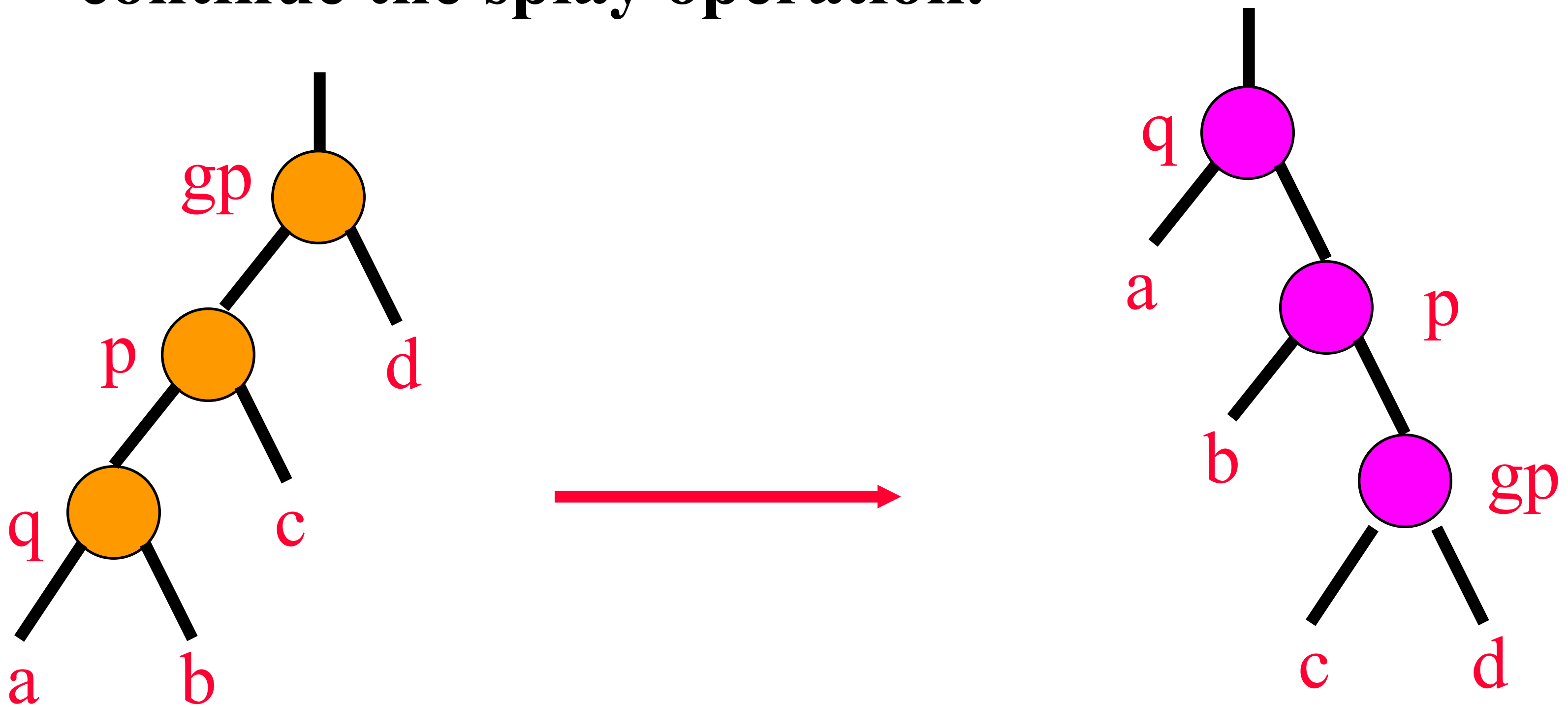
- **If q = null or q is the root, do nothing (splay is over).**
- **If q is at level 2, do a one-level move and terminate the splay operation.**



- q right child of p is symmetric.

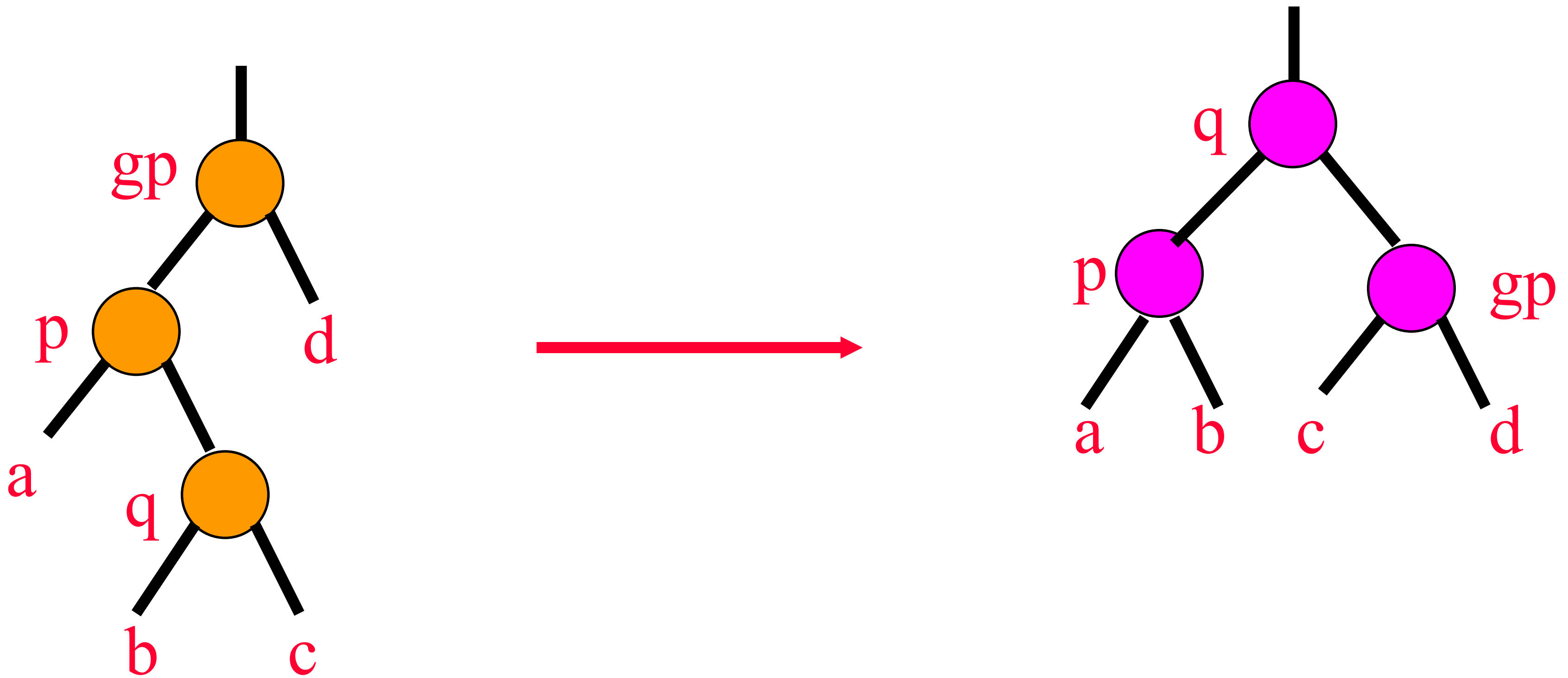
Splay Step

- **If q is at a level > 2, do a two-level move and continue the splay operation.**



- q right child of right child of gp is symmetric.

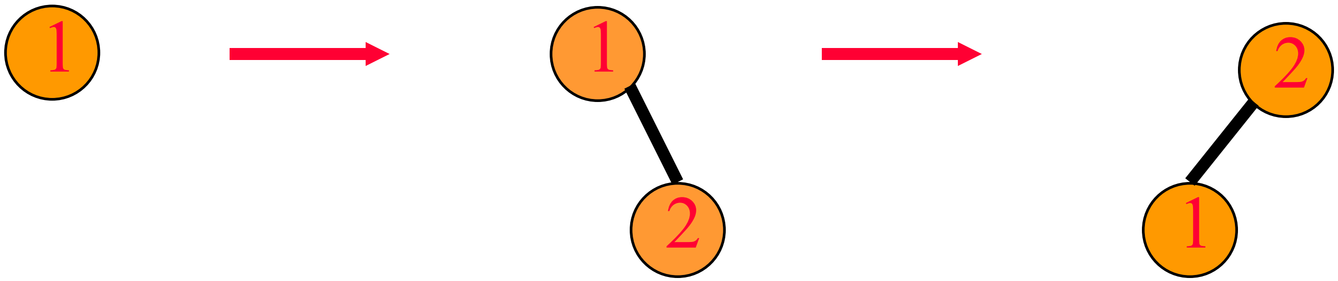
2-Level Move (case 2)



- q left child of right child of gp is symmetric.

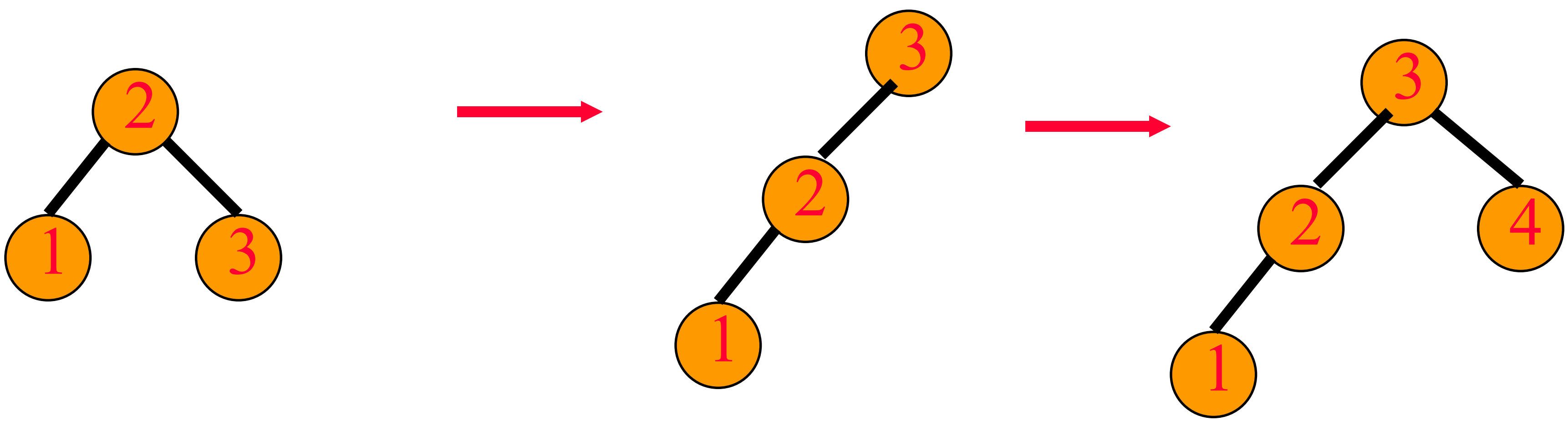
Per Operation Actual Complexity

- Start with an empty splay tree and insert pairs with keys 1, 2, 3, …, in this order.



Per Operation Actual Complexity

- **Start with an empty splay tree and insert pairs with keys 1, 2, 3, …, in this order.**



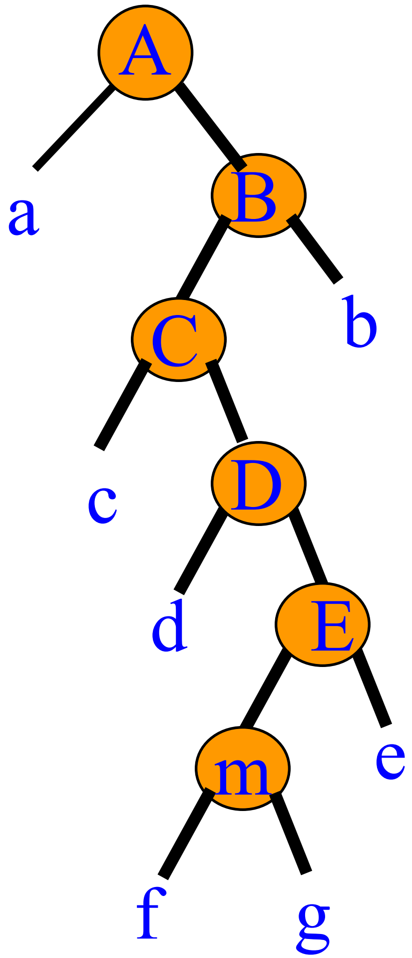
Per Operation Actual Complexity

- **Worst-case height = n.**
- **Actual complexity of search, insert, delete, and split is O(n).**

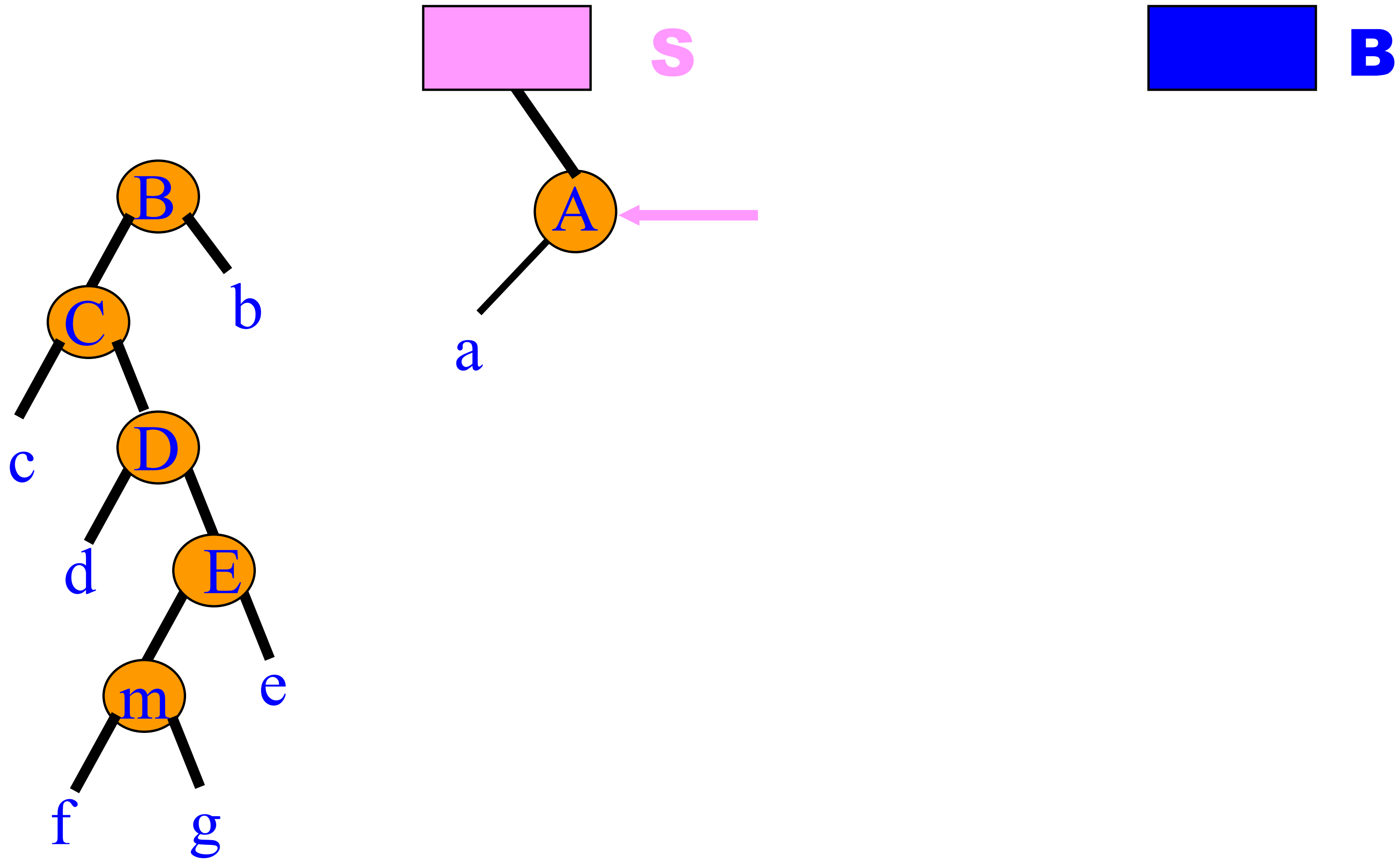
Top-Down Splay Trees

- **On the way down the tree, split the tree into the binary search trees S (small elements) and B (big elements).**
  - **Similar to split operation in an unbalanced binary search tree.**
  - **However, a rotation is done whenever an LL or RR move is made.**
  - **Move down 2 levels at a time, except (possibly) in the end when a one level move is made.**
- **When the splay node is reached, S, B, and the subtree rooted at the splay node are combined into a single binary search tree.**

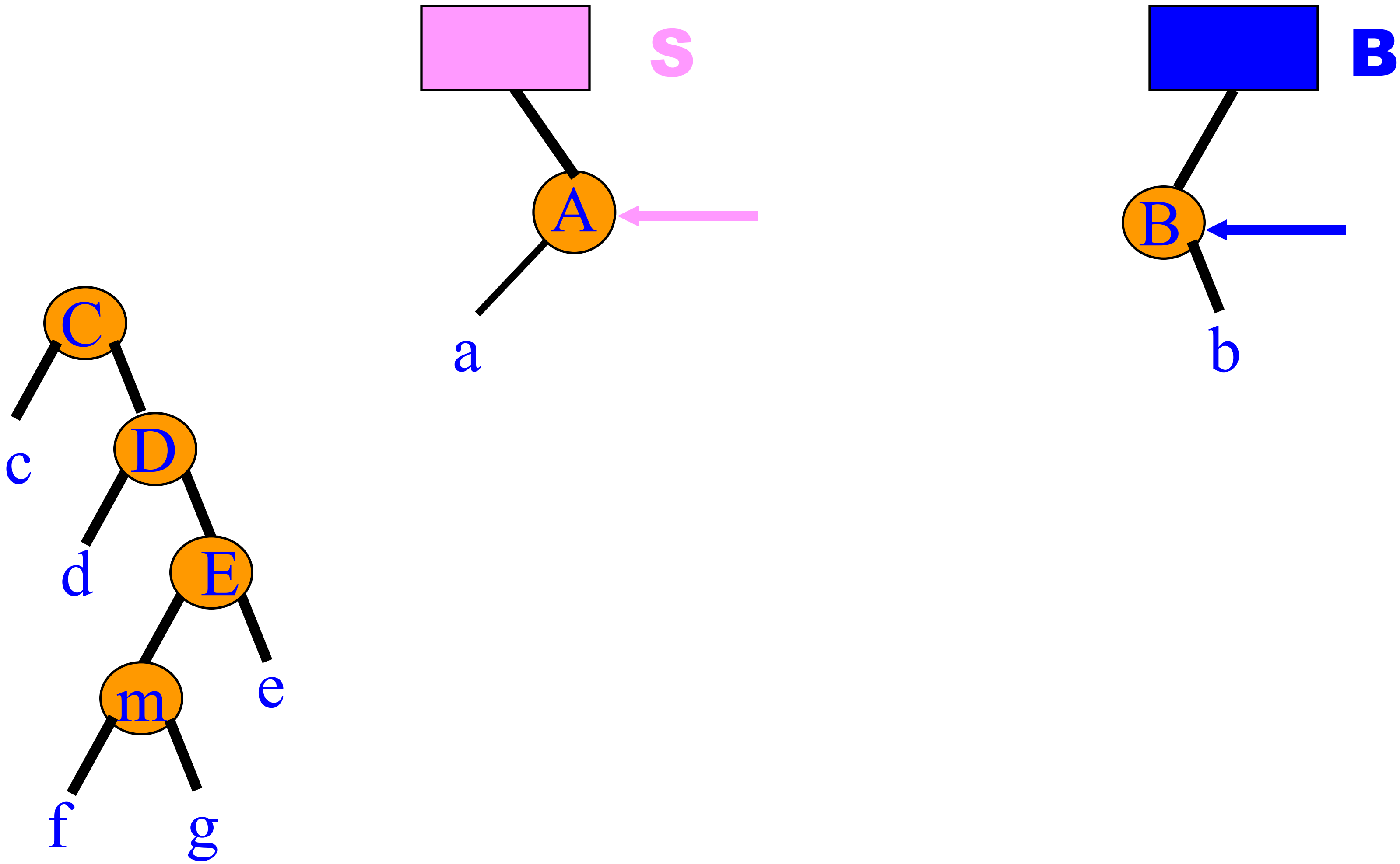
Split A Binary Search Tree



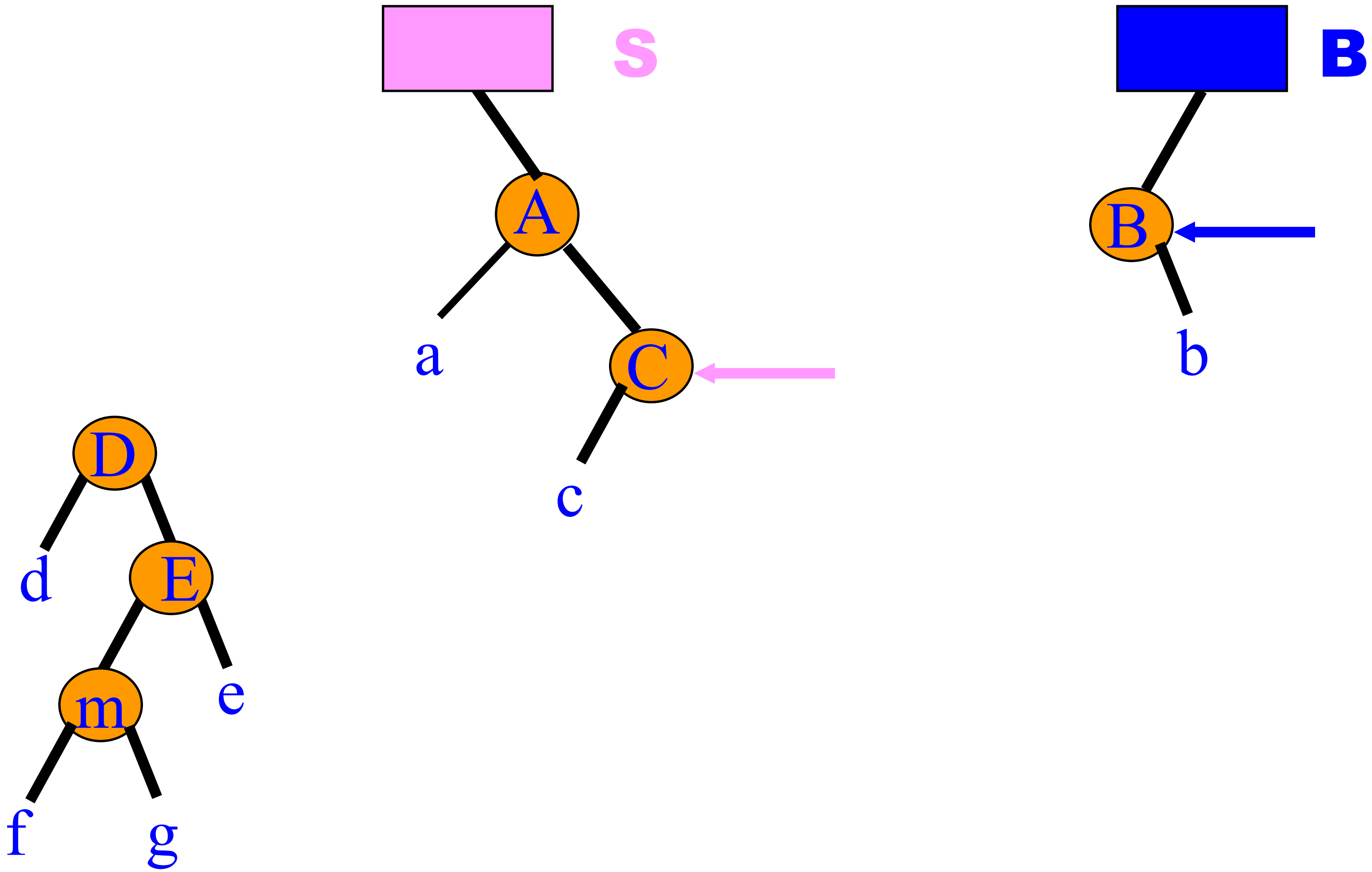
Split A Binary Search Tree



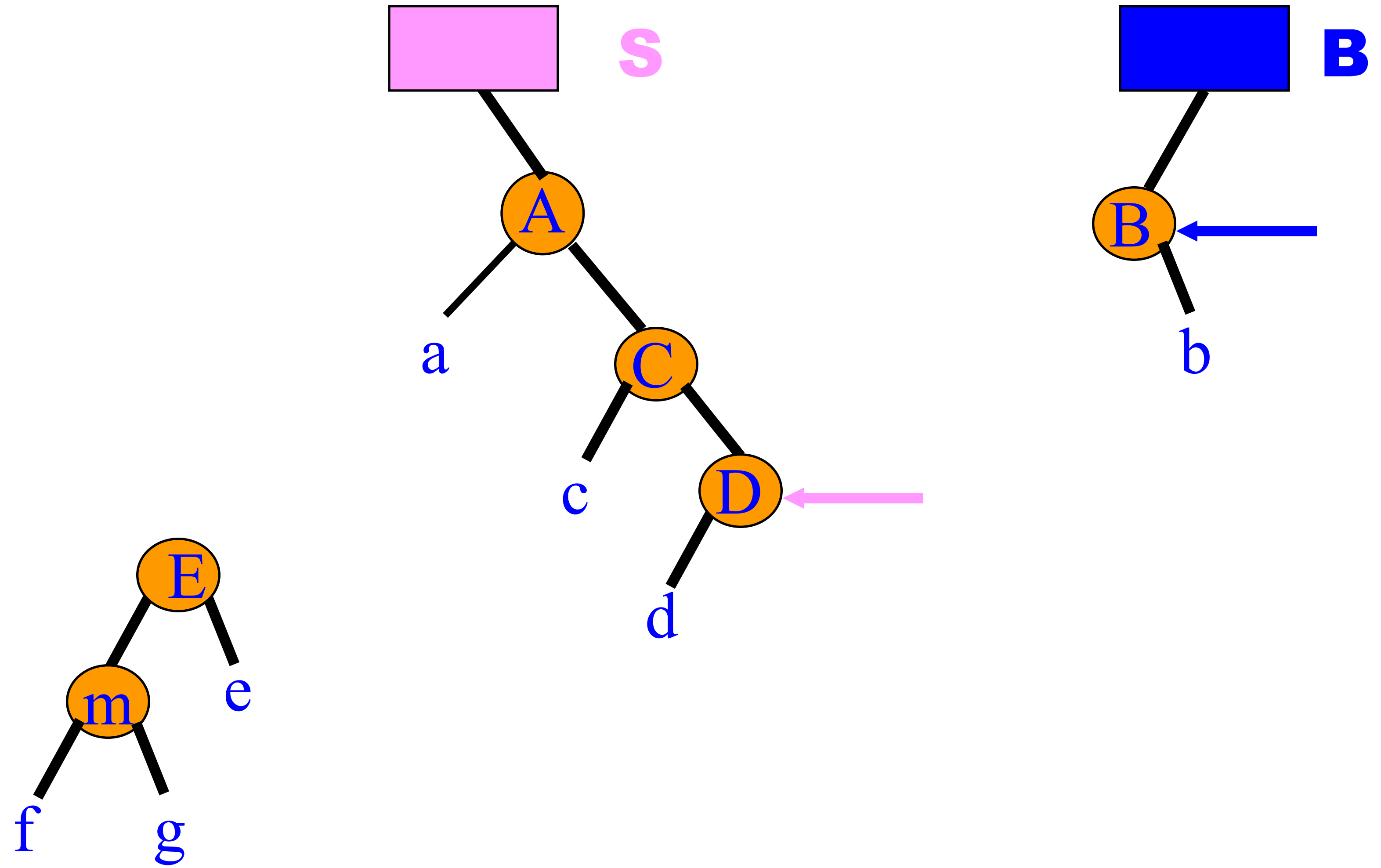
Split A Binary Search Tree



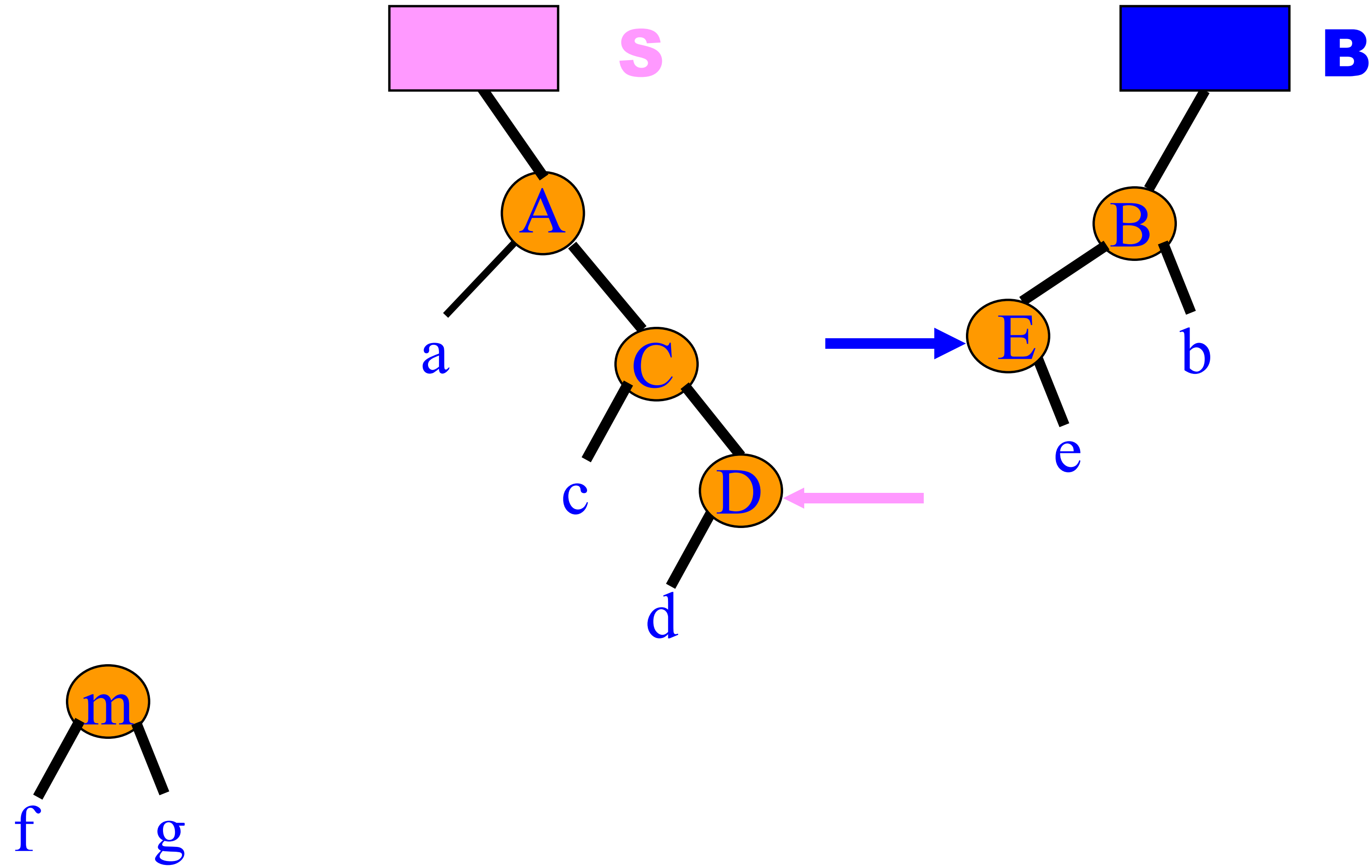
Split A Binary Search Tree



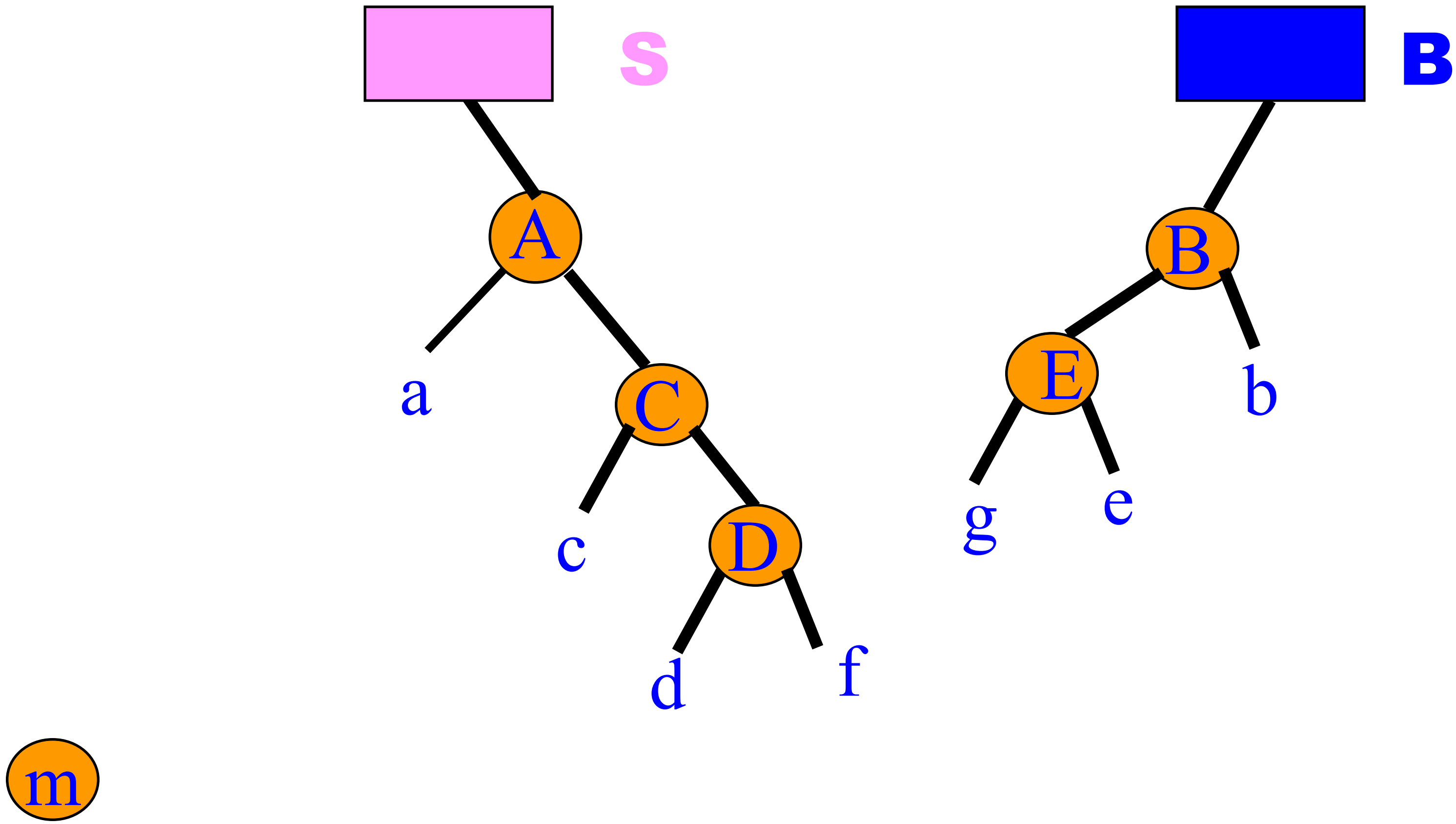
Split A Binary Search Tree



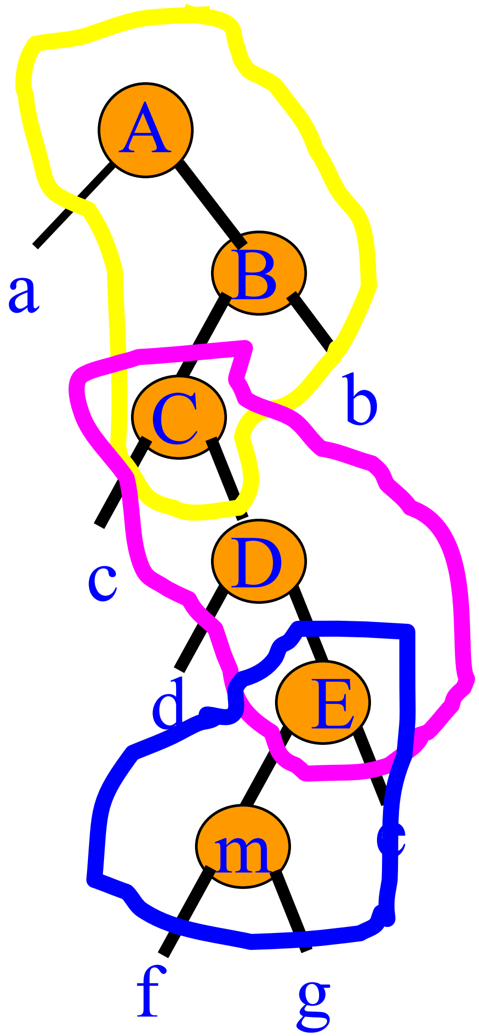
Split A Binary Search Tree



Split A Binary Search Tree

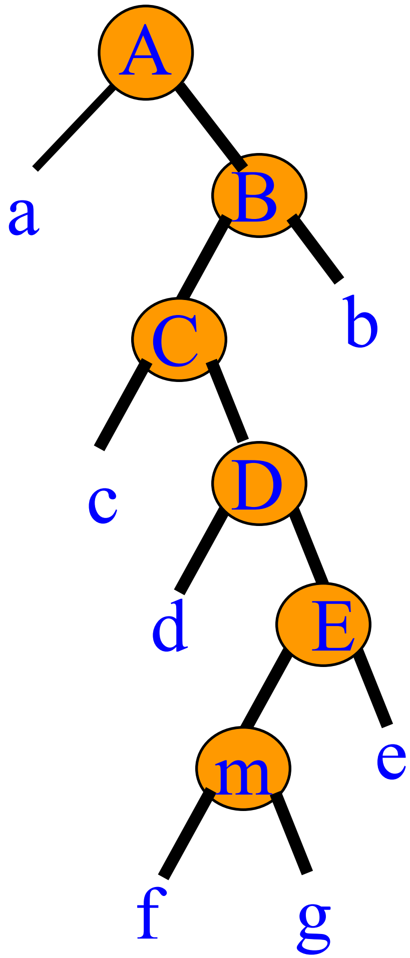


Two-Level Moves

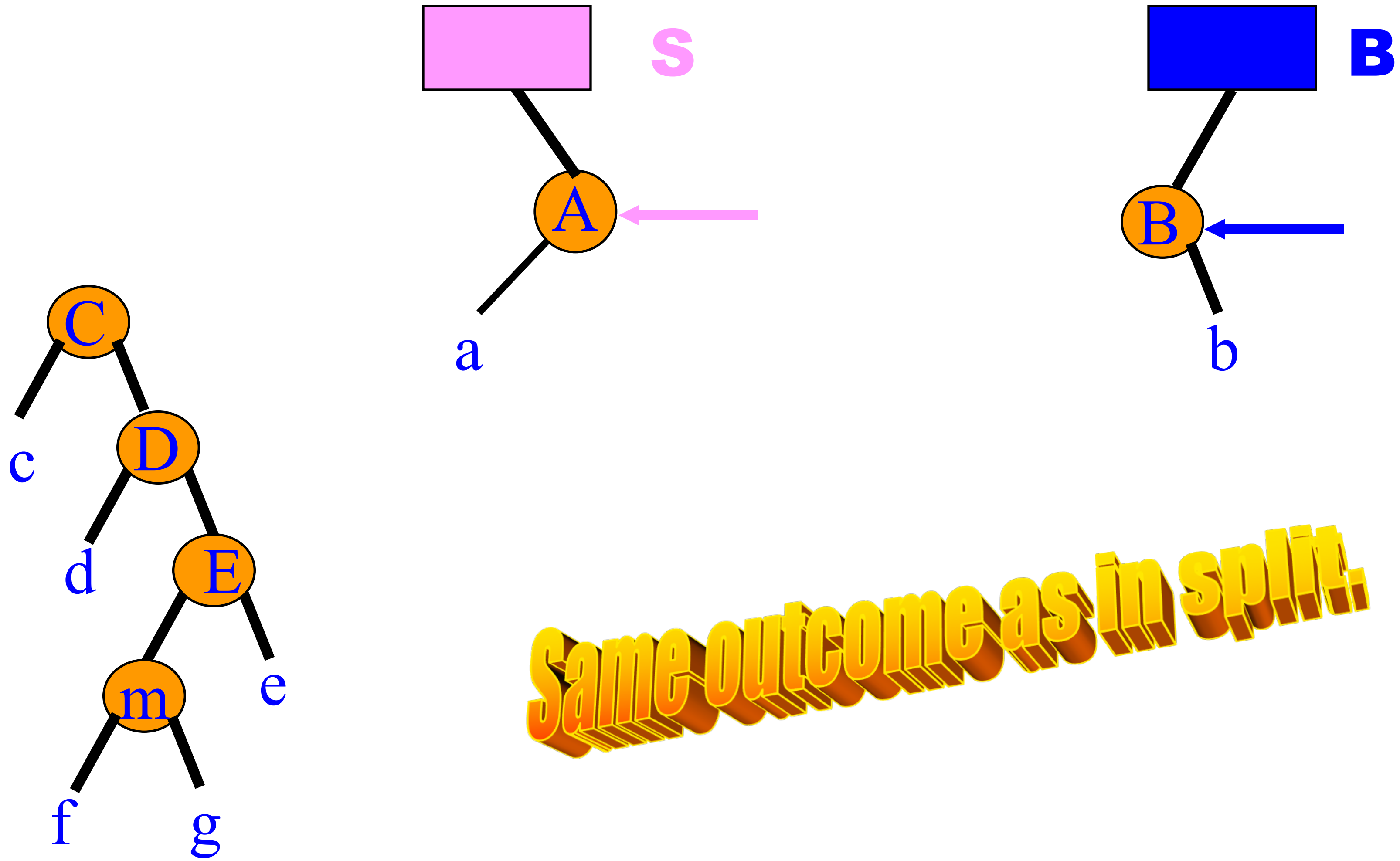


- **Let m be the splay node.**
- RL move from A to C.
- RR move from C to E.
- L move from E to m.

RL Move

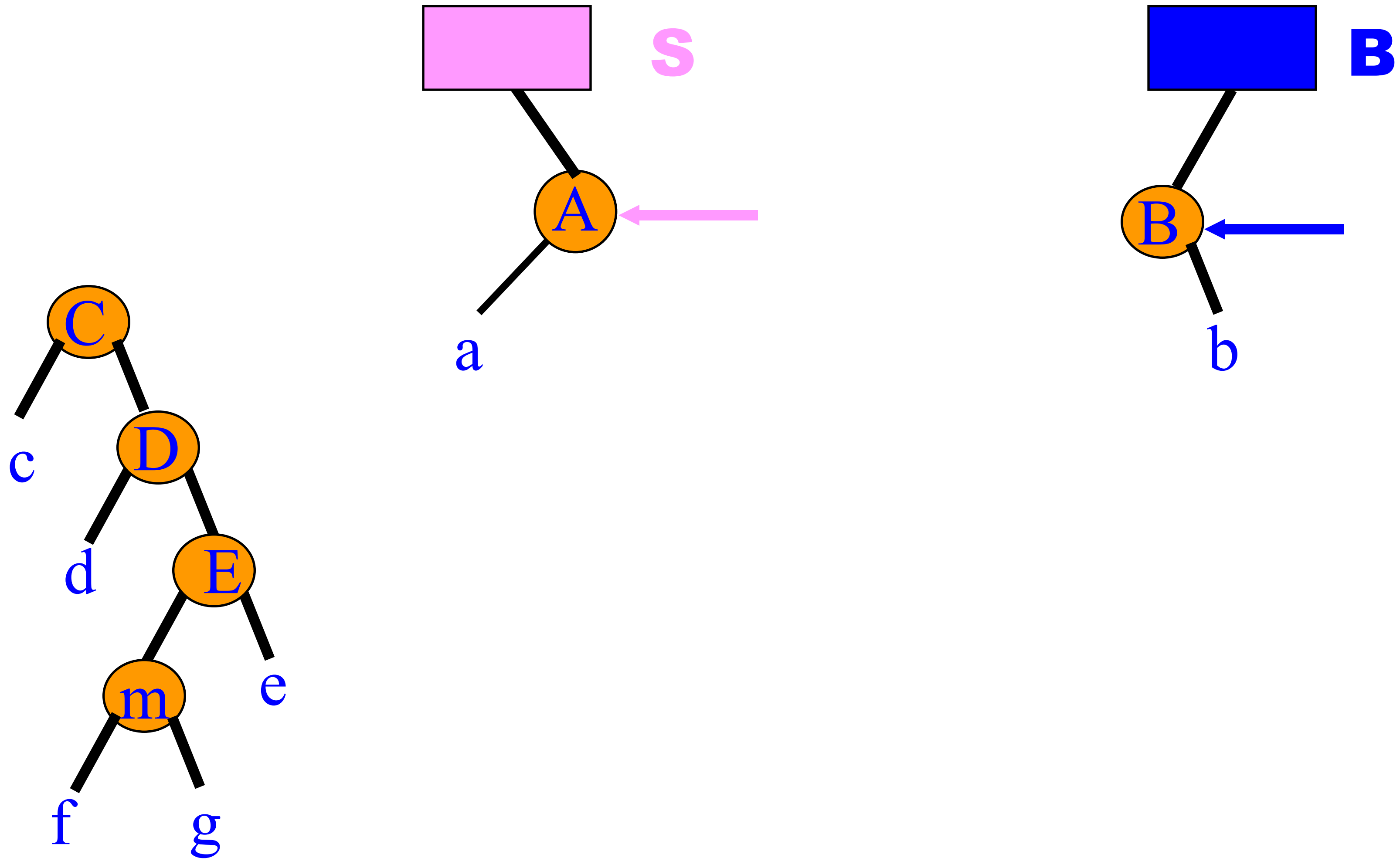


RL Move

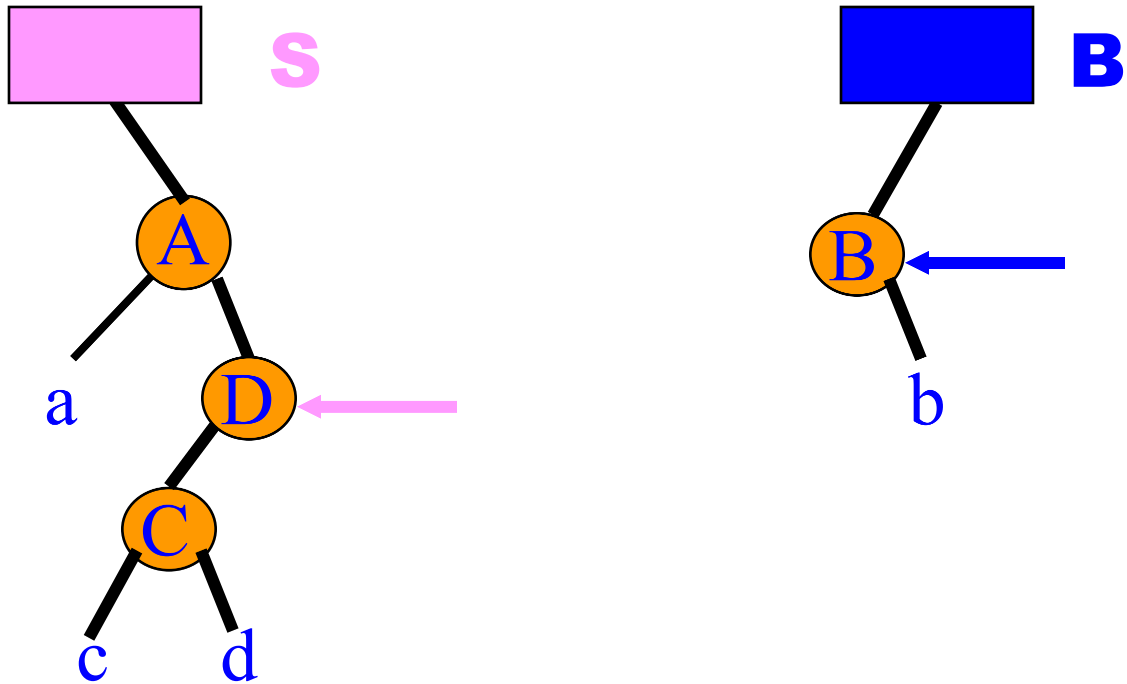


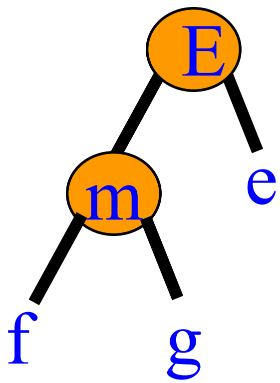
Same outcome as in split.

RR Move



RR Move

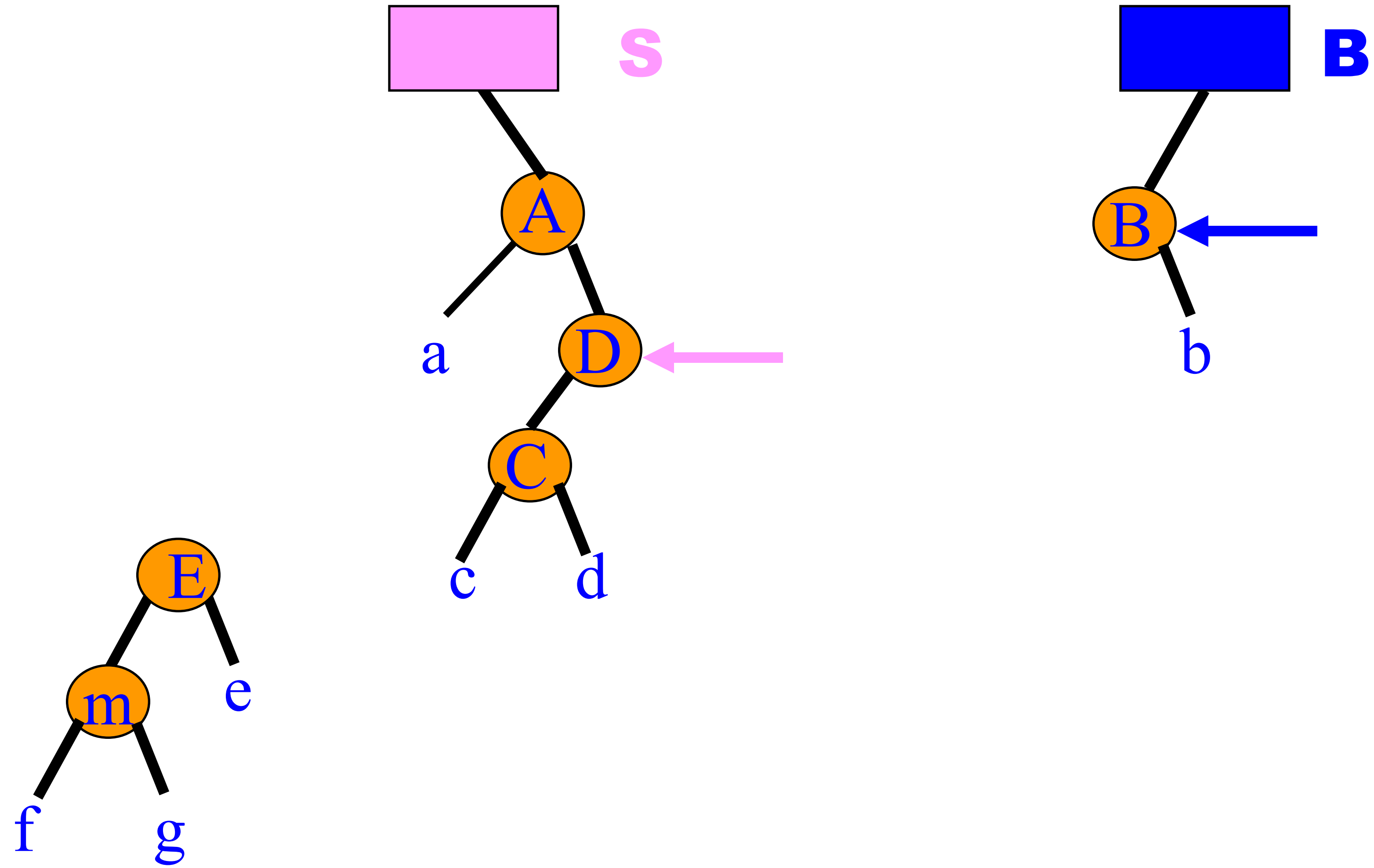




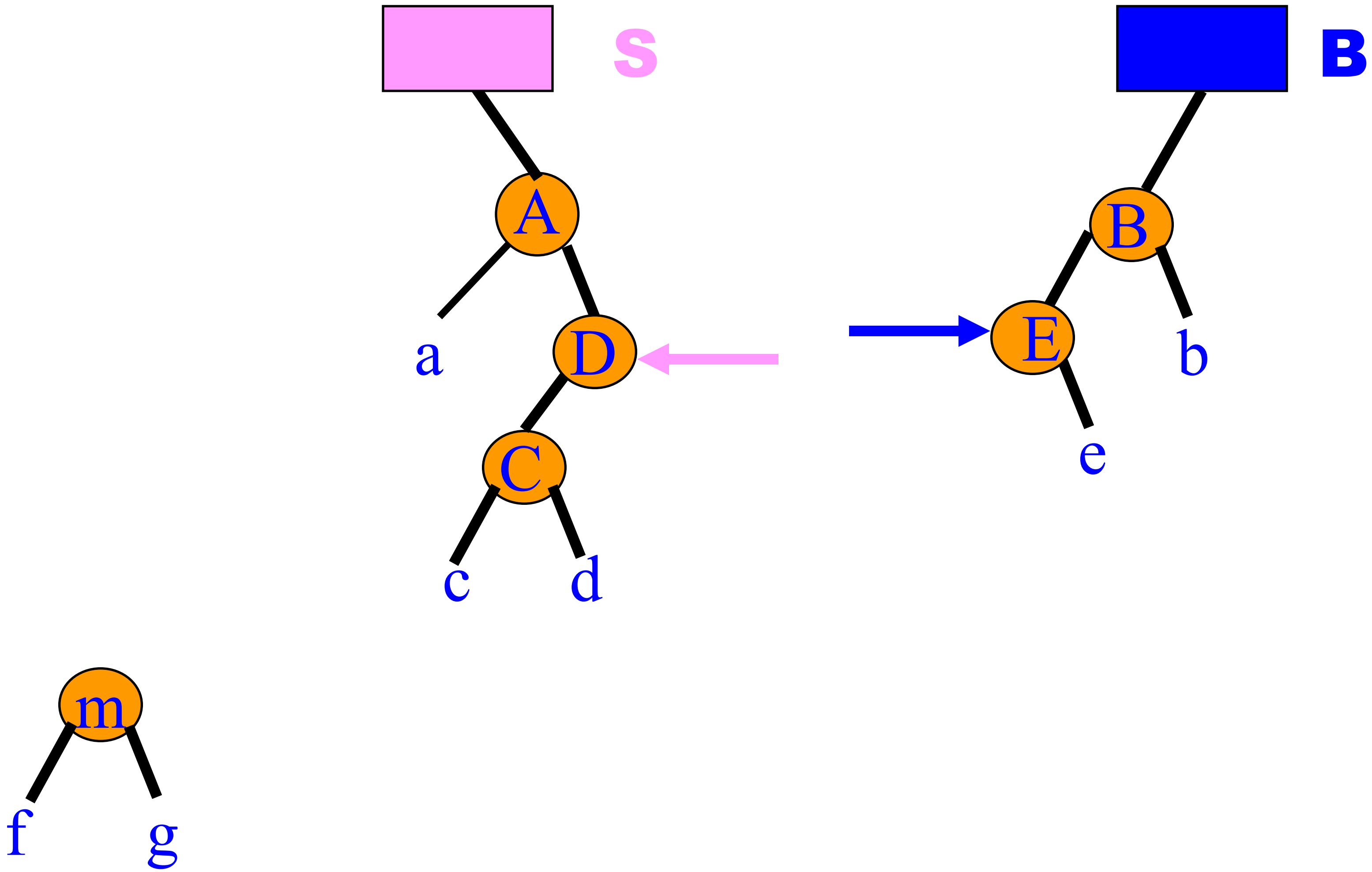
Rotation performed.

Outcome is different from split.

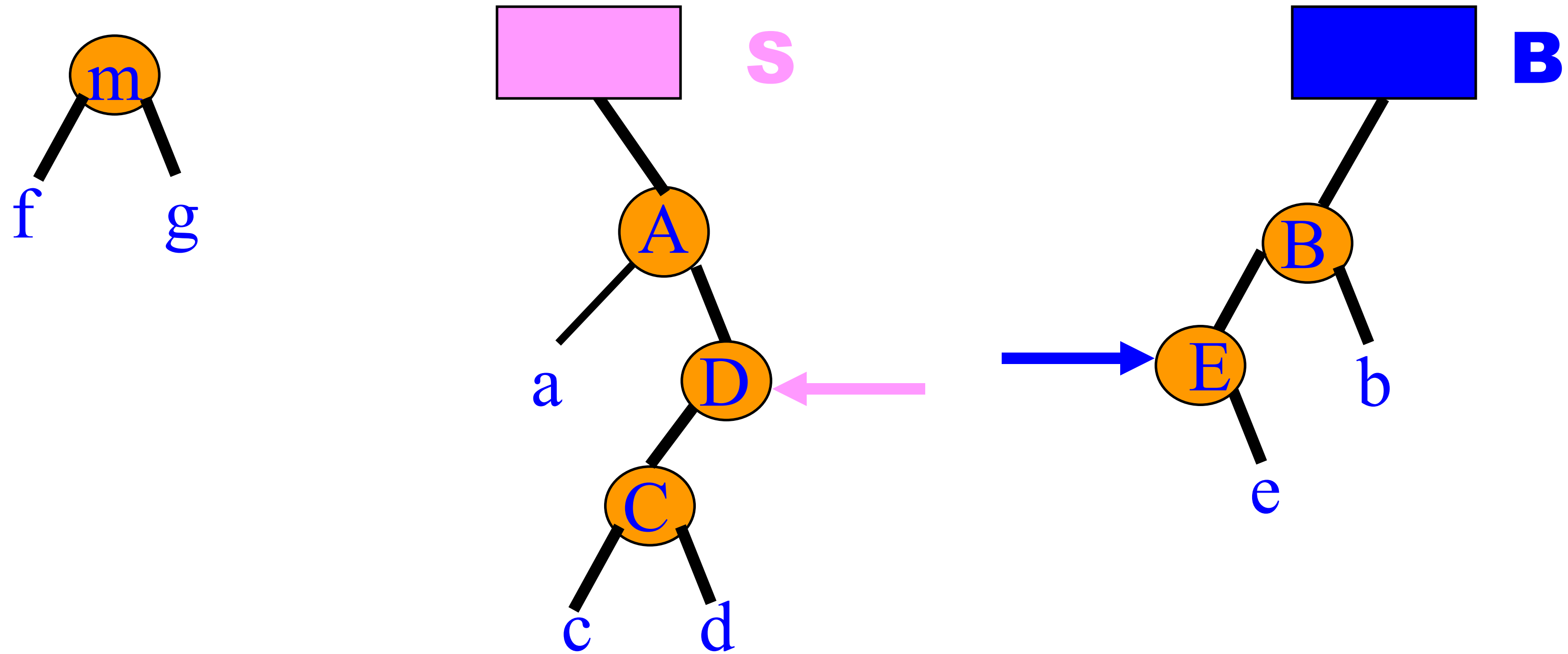
L Move



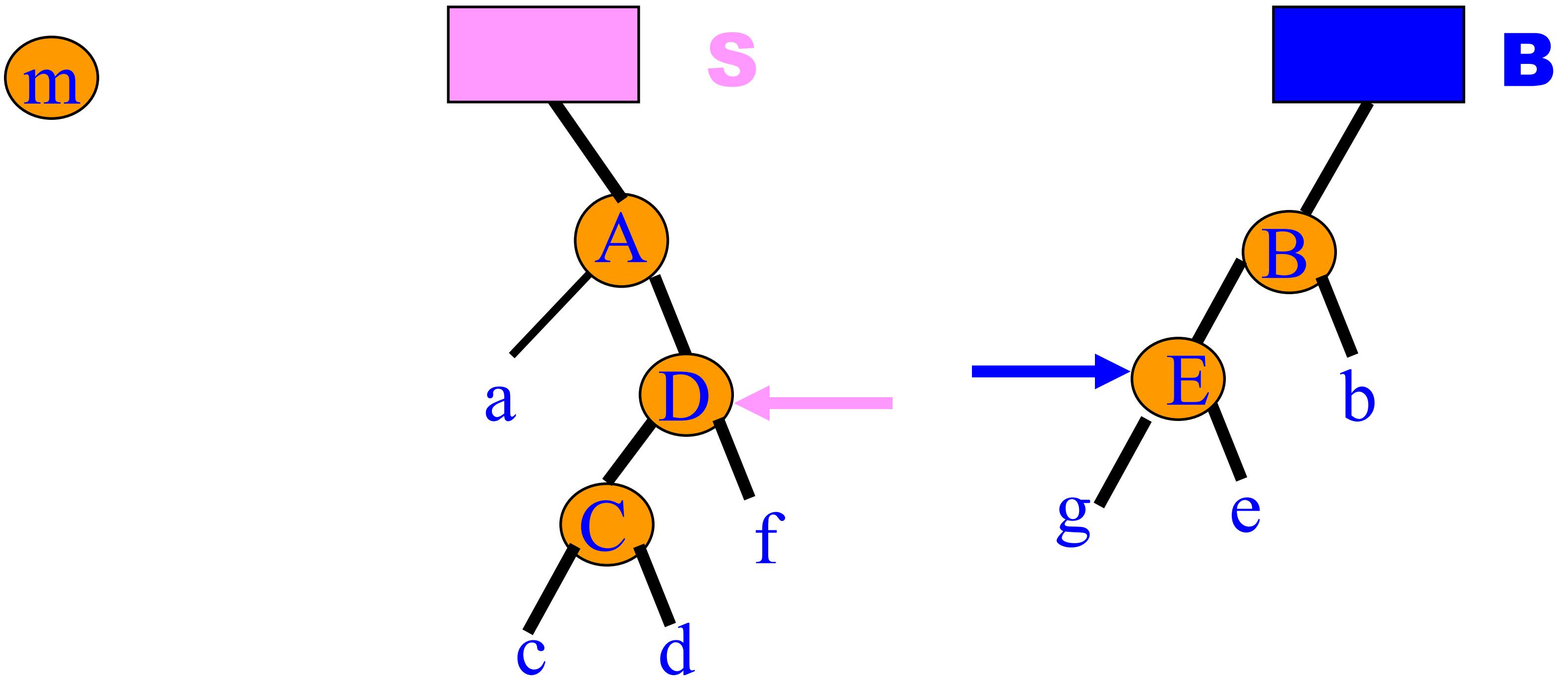
L Move



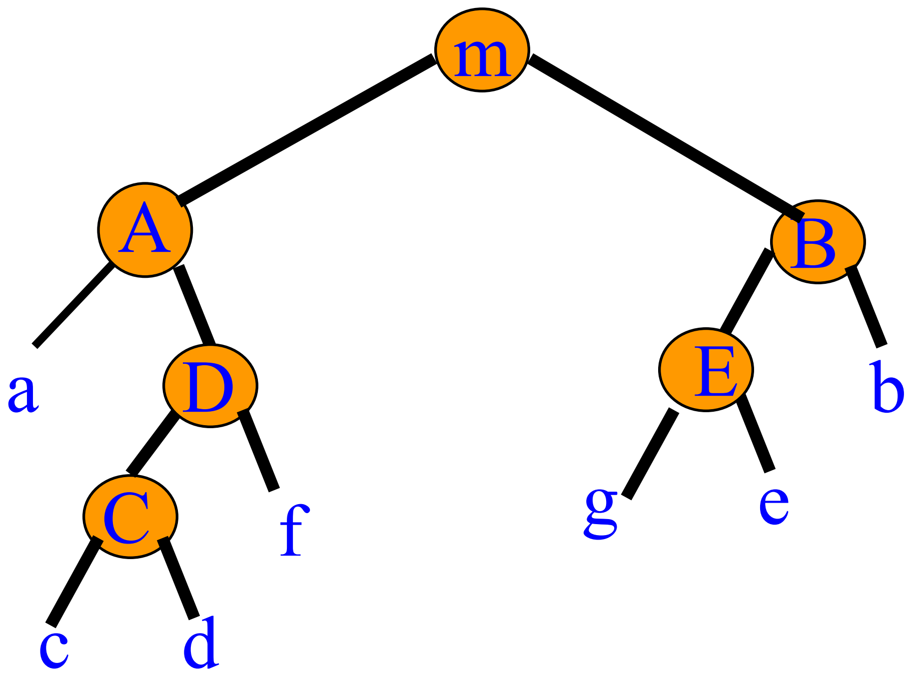
Wrap Up



Wrap Up



Wrap Up



Bottom Up vs Top Down

- Top down splay trees are faster than bottom up splay trees.