Data Structures and Algorithms (11)
CS F211

Searching - Re-visited

- **Binary tree** $O(\log n)$ ***if it stays balanced***
  - **Simple binary tree good for static collections**
  - **Low (preferably zero) frequency of insertions/deletions**

  ***but* my collection keeps changing!**
  - **It's dynamic**
  - **Need to keep the tree balanced**
- **First, examine some basic tree operations**
  - **Useful in several ways!**

Tree Traversals

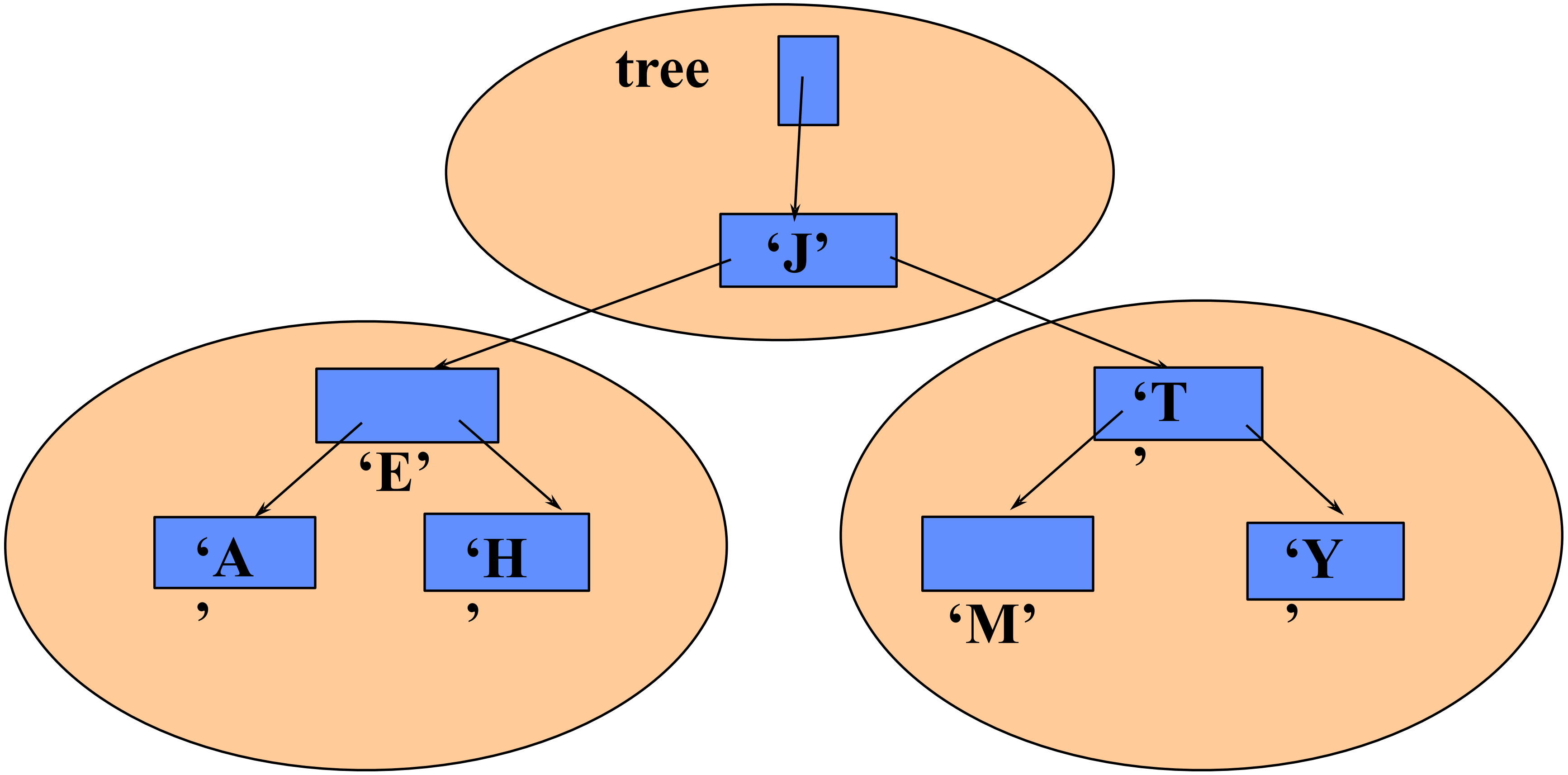
There are mainly three ways to traverse a tree:

Inorder Traversal

Postorder Traversal

Preorder Traversal

Inorder Traversal: A E H J M T Y



Visit left subtree first

Visit right subtree last

Inorder Traversal

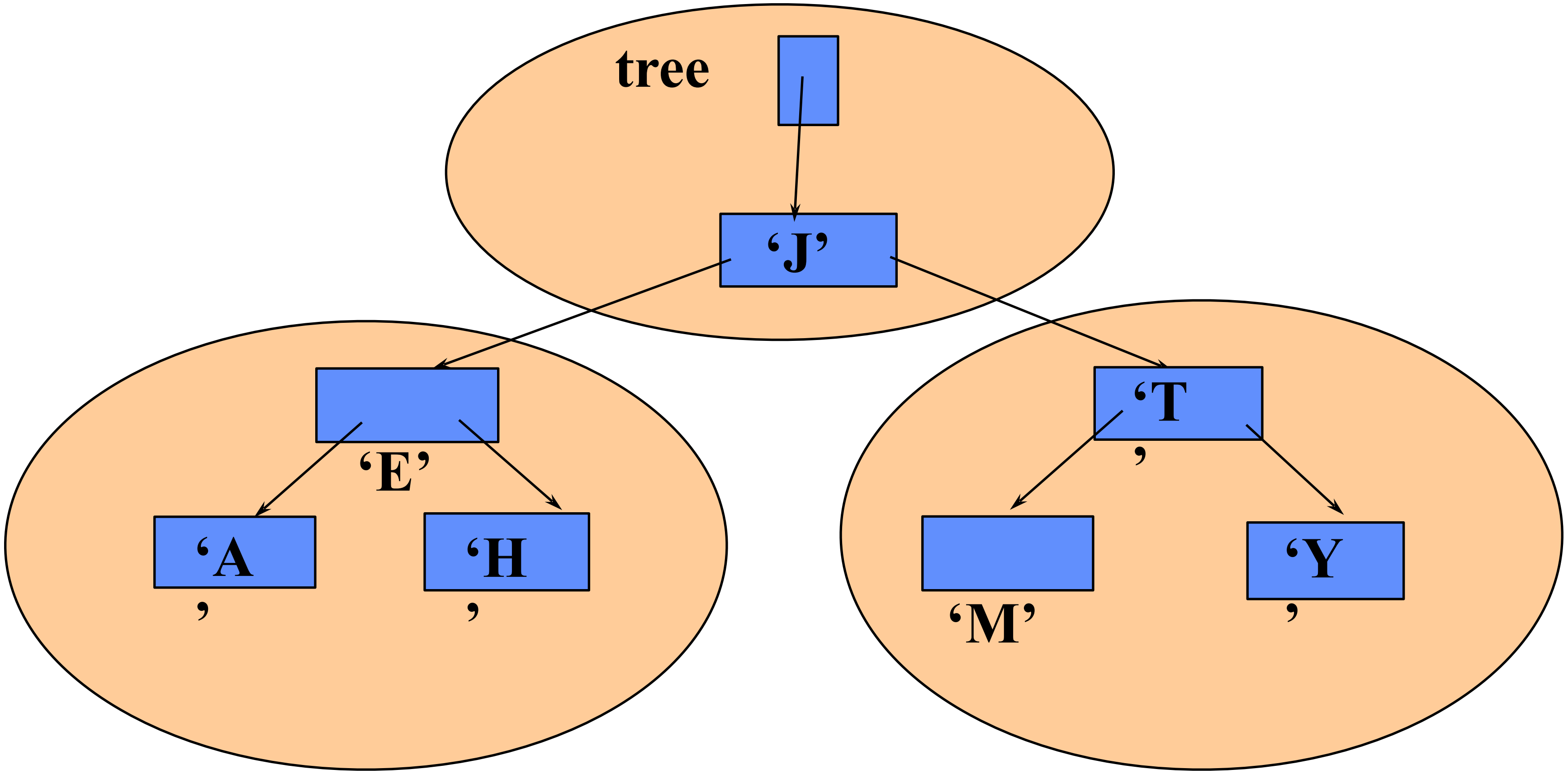
Visit the nodes in the left subtree, then visit the root of the tree, then visit the nodes in the right subtree

```
Inorder(tree)

If tree is not NULL
  Inorder(Left(tree))
  Visit Info(tree)
  Inorder(Right(tree))
```

(Warning: "visit" means that the algorithm does something with the values in the node, e.g., print the value)

Postorder Traversal: A H E M Y T J



Visit left subtree first

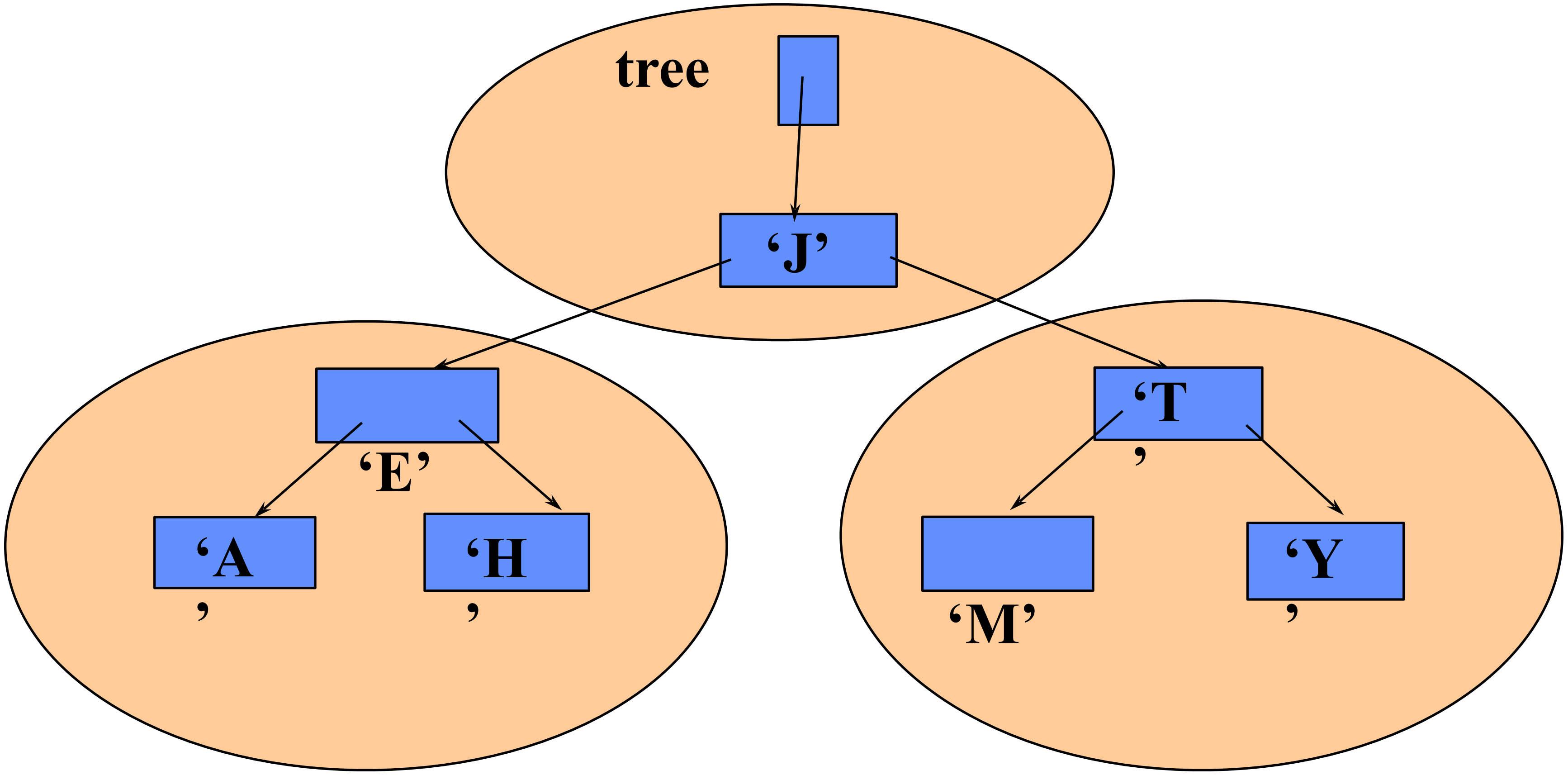
Visit right subtree second

Postorder Traversal

Visit the nodes in the left subtree first, then visit the nodes in the right subtree, then visit the root of the tree

```
Postorder(tree)
If tree is not NULL
  Postorder(Left(tree))
  Postorder(Right(tree))
  Visit Info(tree)
```

Preorder Traversal: J E A H T M



Visit left subtree second

Visit right subtree last

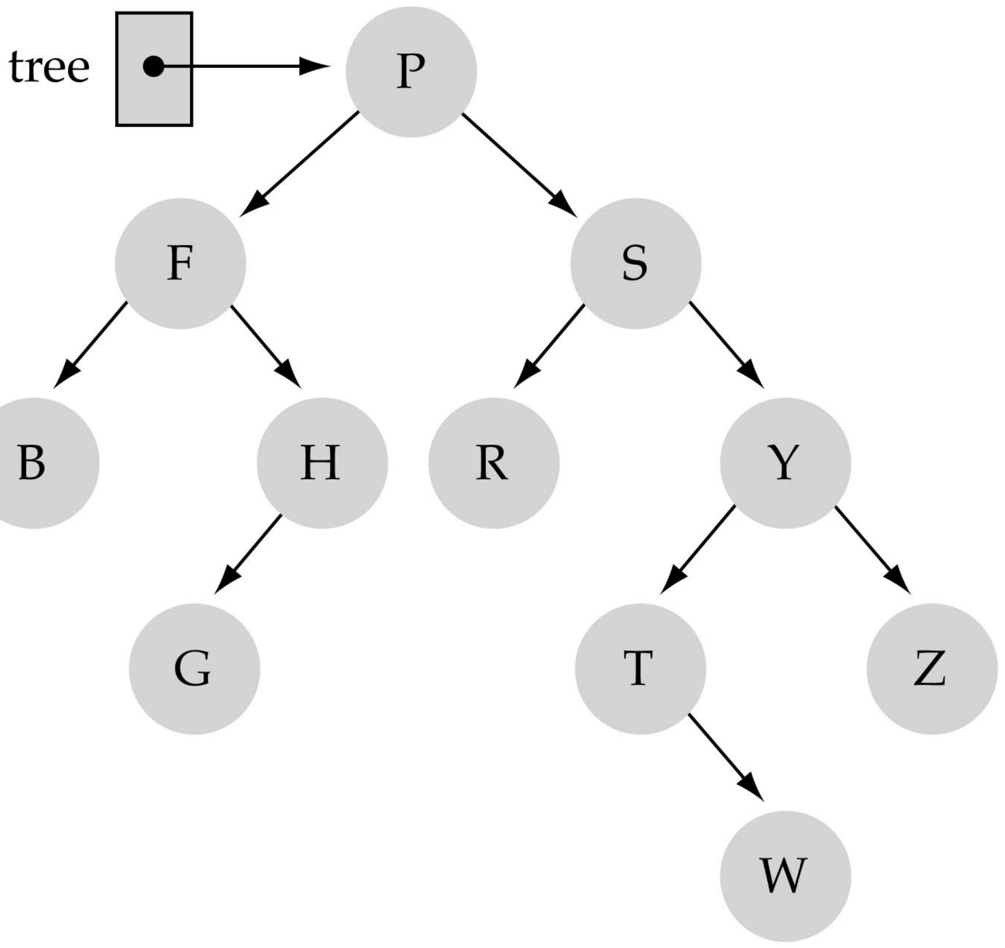
Preorder Traversal

- **Visit the root of the tree first, then visit the nodes in the left subtree, then visit the nodes in the right subtree**

```
Preorder(tree)
If tree is not NULL
  Visit Info(tree)
  Preorder(Left(tree))

  Preorder(Right(tree))
```

Tree Traversals



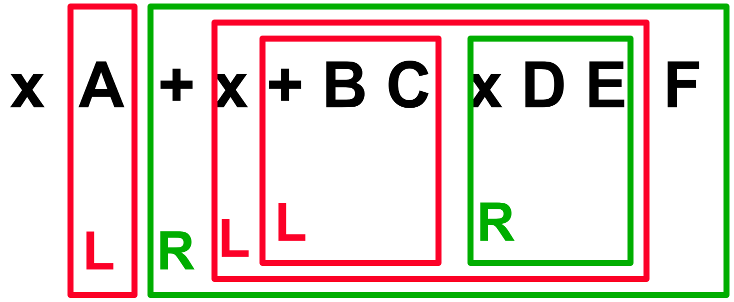
Inorder:	B	F	G	H	P	R	S	T	W	Y	Z
Preorder:	P	F	B	H	G	S	R	Y	T	W	Z
Postorder:	B	G	H	F	R	W	T	Z	Y	S	P

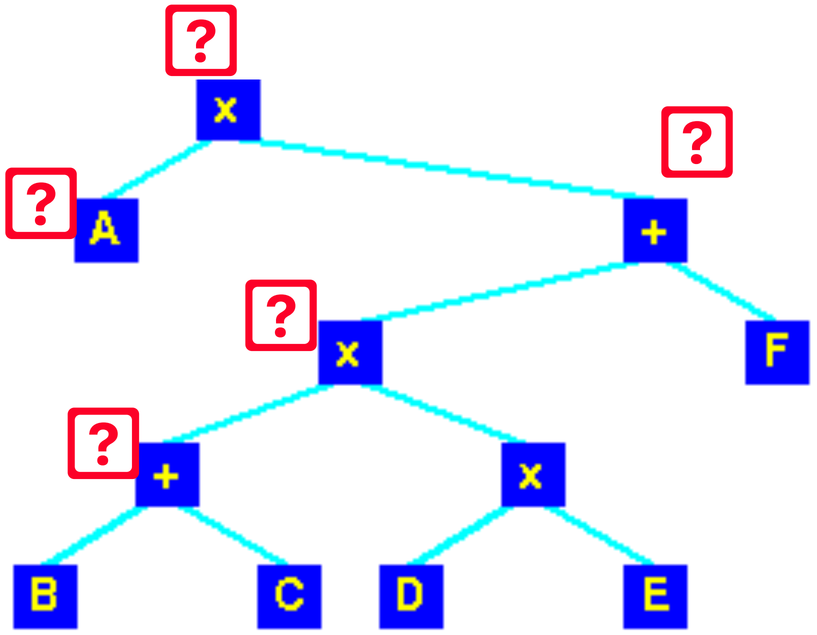
Tree Traversal

- Traversal = visiting every node of a tree
- Three basic alternatives

? Pre-order

- Root
- Left sub-tree
- Right sub-tree



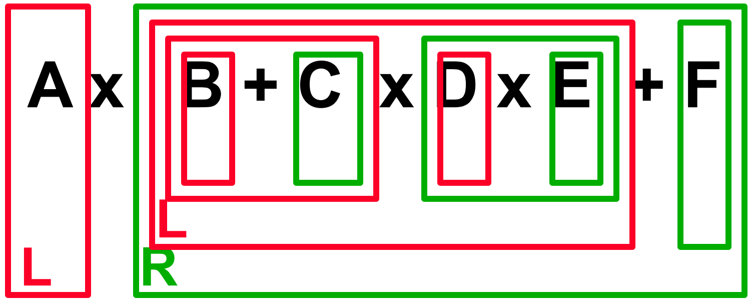


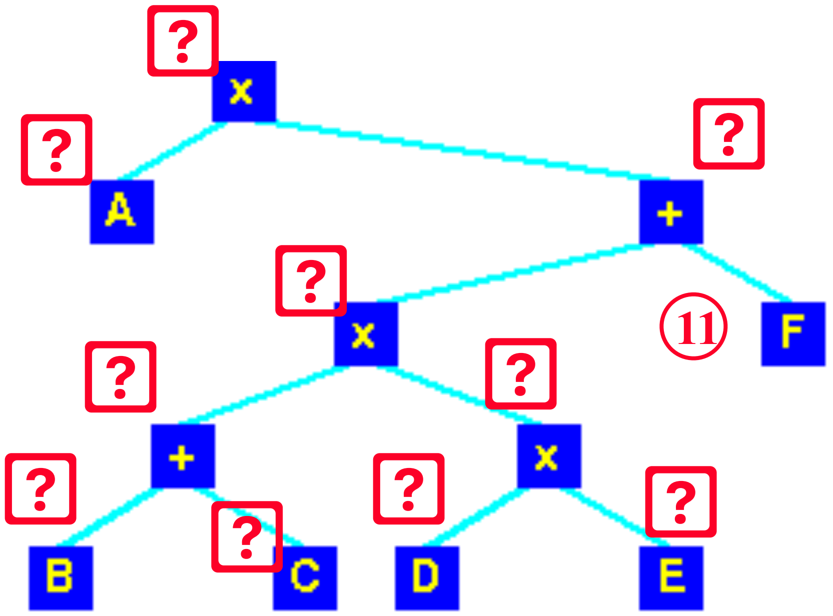
Tree Traversal

- Traversal = visiting every node of a tree
- Three basic alternatives

? In-order

- Left sub-tree
- Root
- Right sub-tree



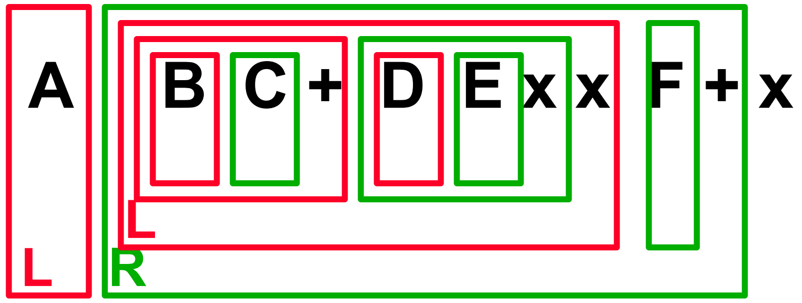


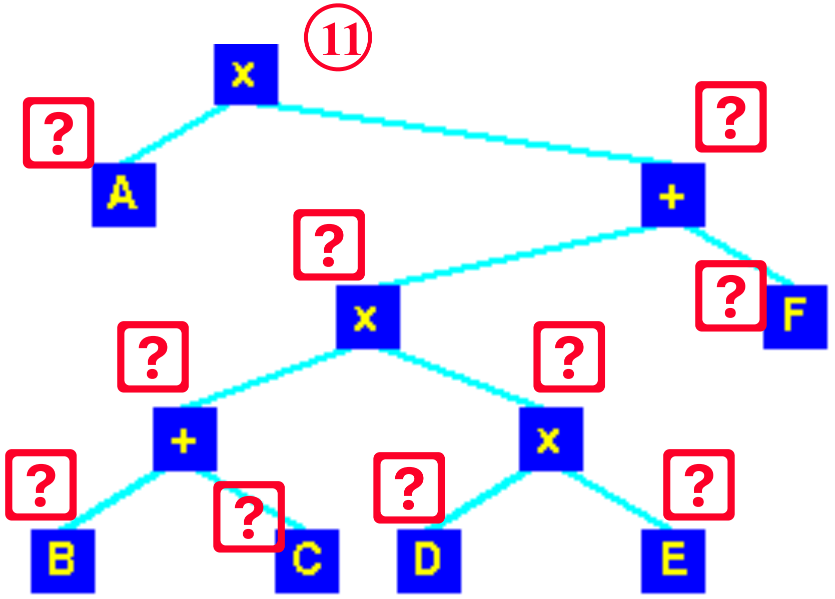
Tree Traversal

- Traversal = visiting every node of a tree
- Three basic alternatives

? Post-order

- Left sub-tree
- Right sub-tree
- Root





Tree Traversal

? **Post-order**

- Left sub-tree
- Right sub-tree
- Root

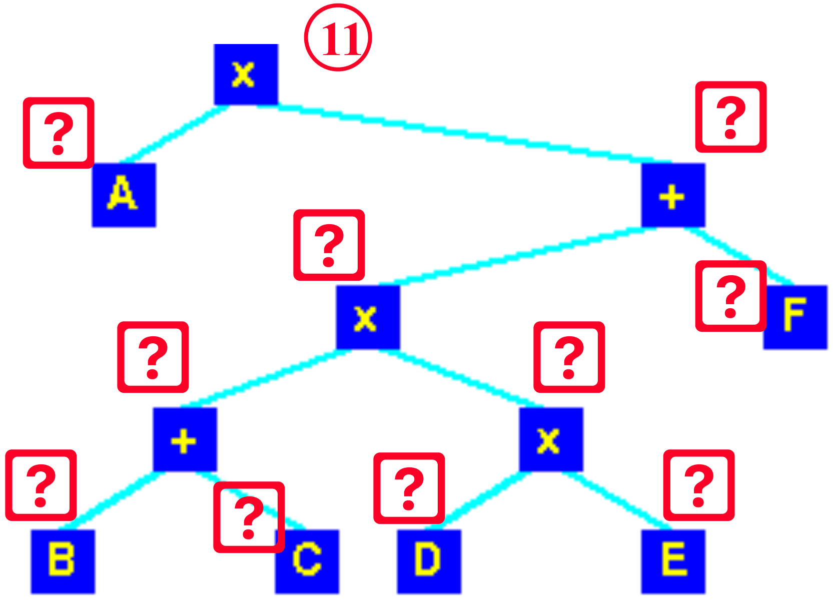
? ***Reverse-Polish***

(A (((BC+)(DEx) x) F +)x)

- **Normal algebraic form**

(A x(((B+C)(DxE))+F))

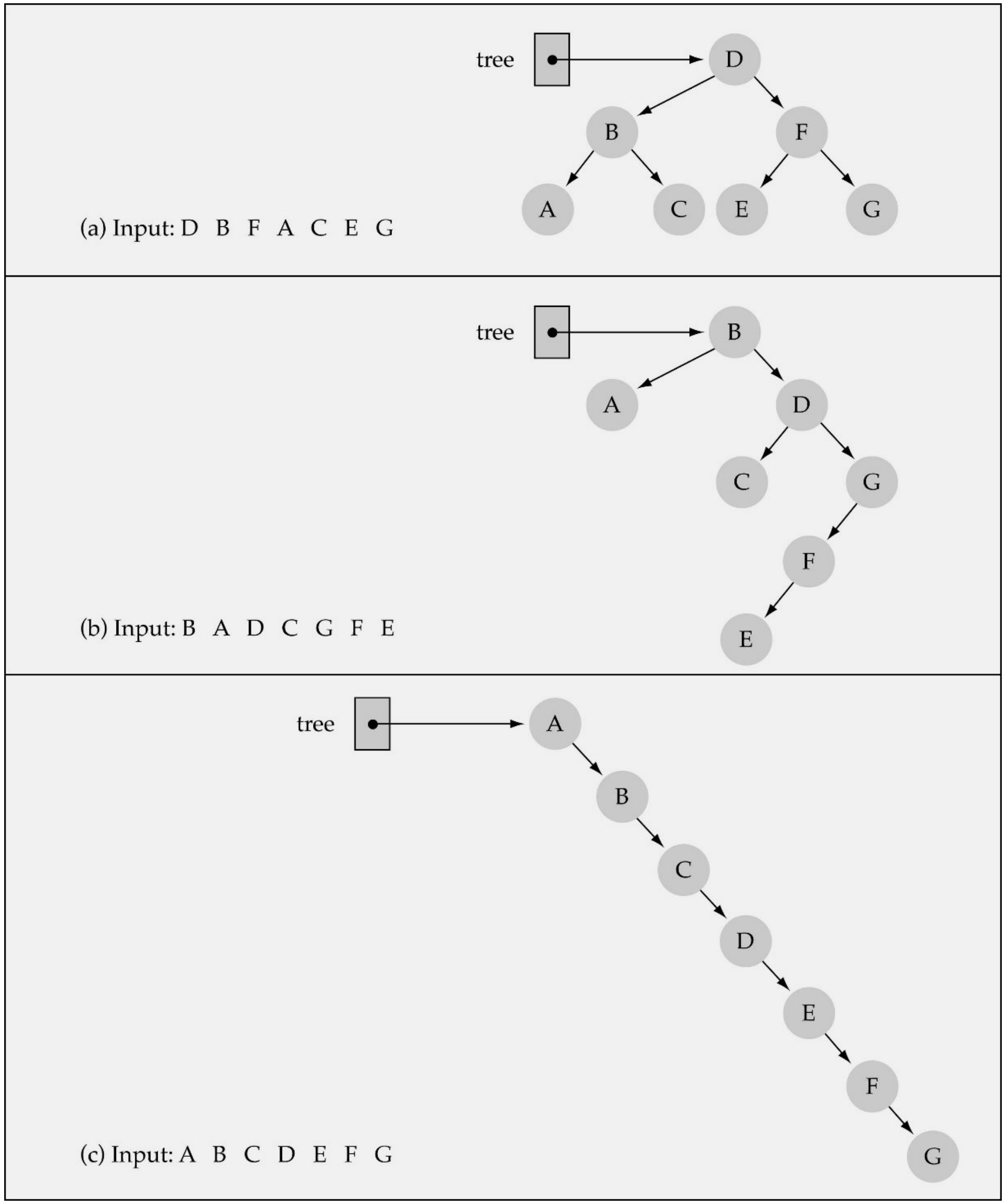
= which traversal?



Does the order of inserting elements into a binary search tree matter?

- ✓ Yes, certain orders produce very unbalanced trees!!
- ✓ Unbalanced trees are not desirable because search time increases!!
- ✓ There are advanced tree structures (e.g.,"red-black trees") which guarantee balanced trees

Does the order of inserting elements into a tree matter? (cont.)



Function DeleteItem

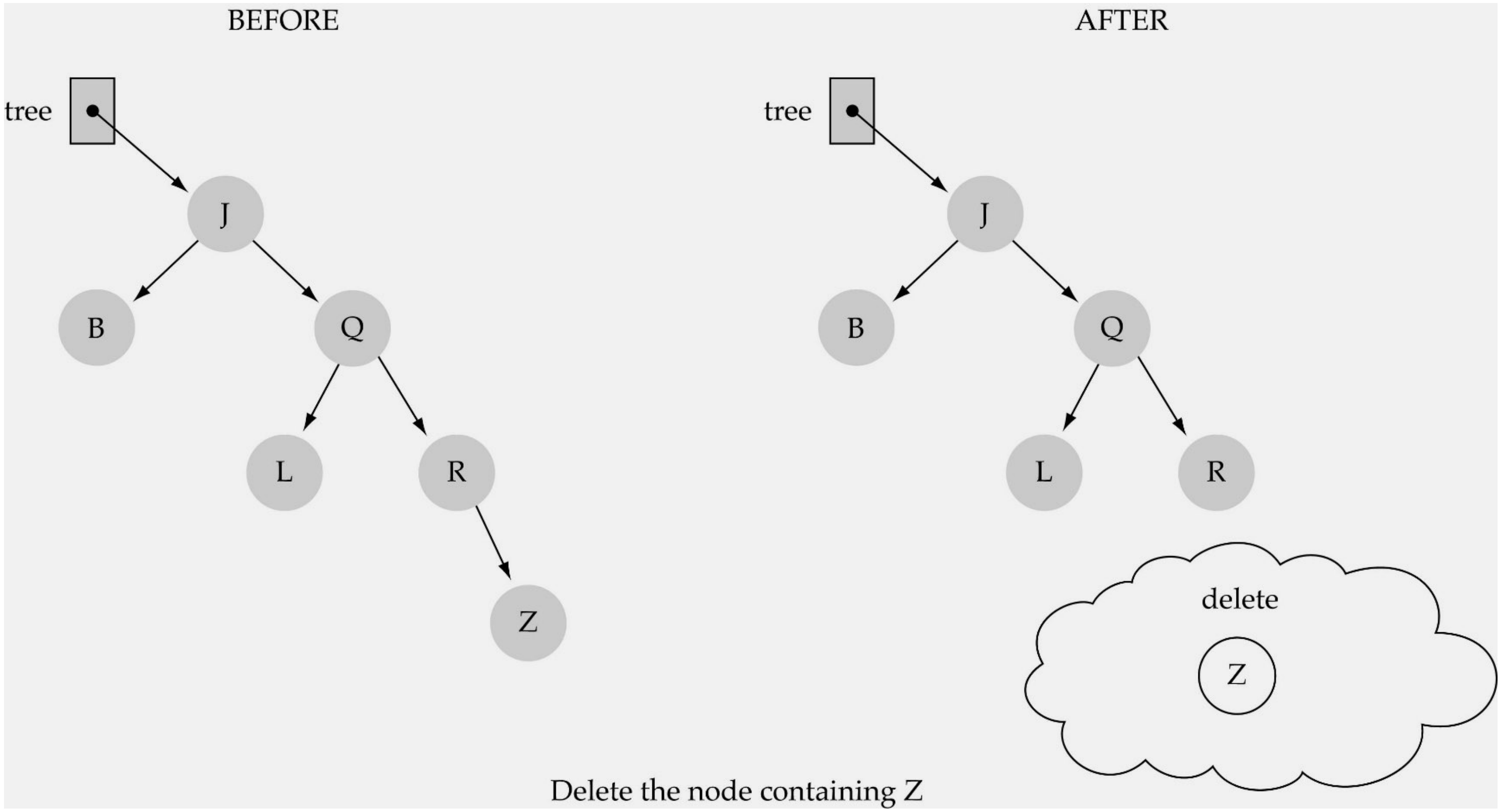
- ✓ First, find the item; then, delete it
- ✓ Important: binary search tree property must be preserved!!
- ✓ We need to consider three different cases:

  (1) Deleting a leaf

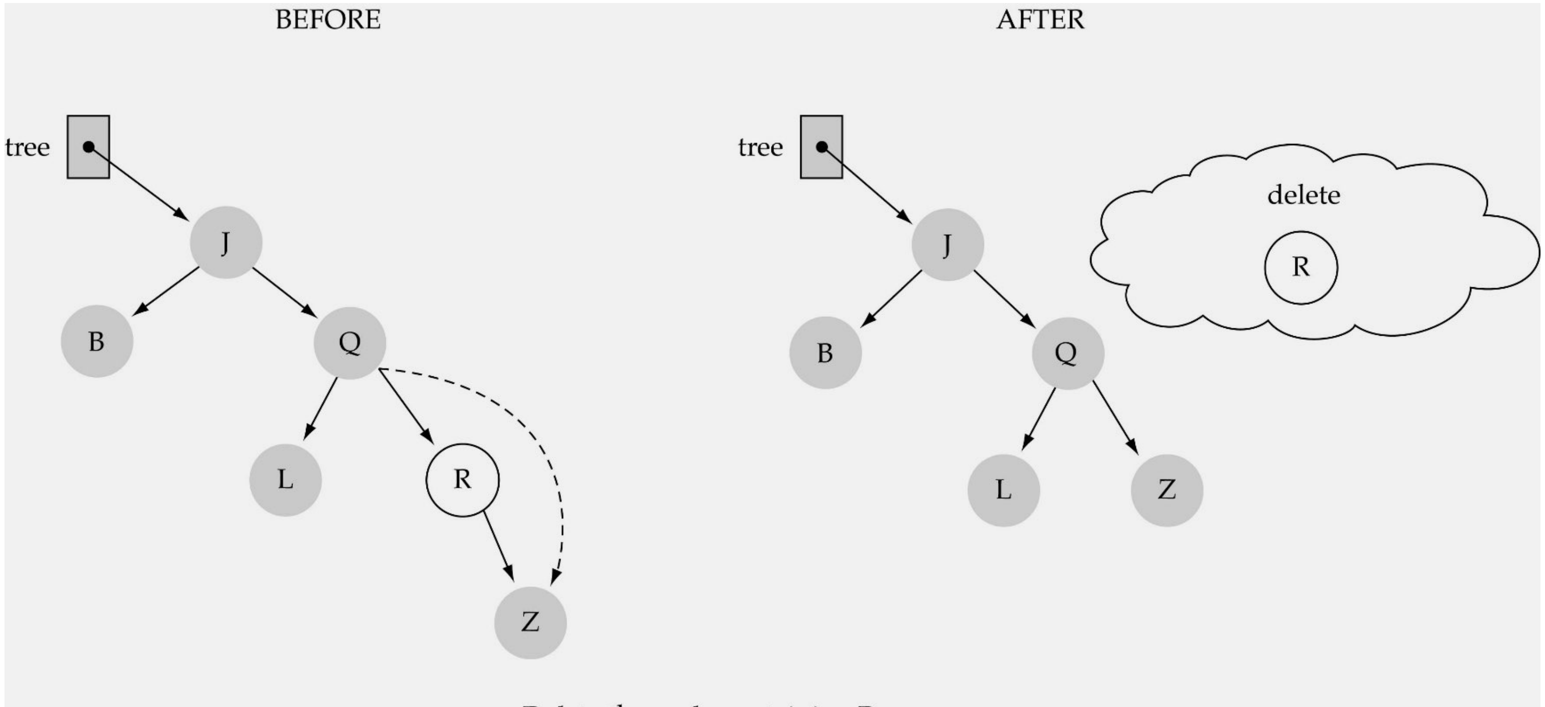
  (2) Deleting a node with only one child

  (3) Deleting a node with two children

(1) Deleting a leaf

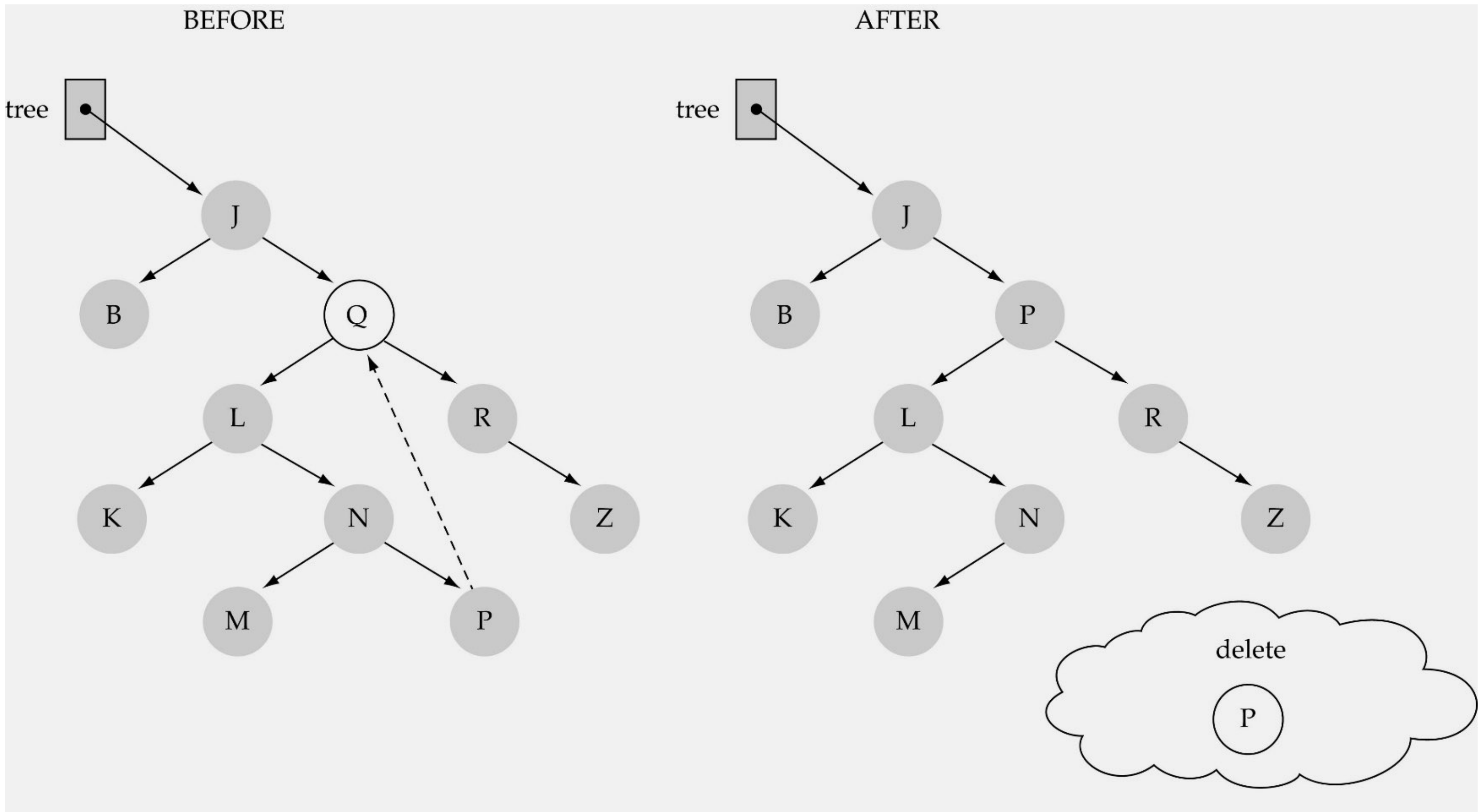


(2) Deleting a node with only one child



Delete the node containing R

(3) Deleting a node with two children



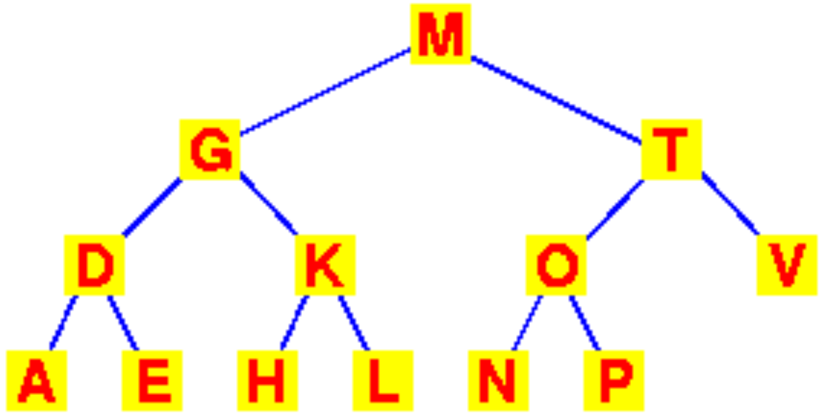
Delete the node containing Q

(3) Deleting a node with two children (cont.)

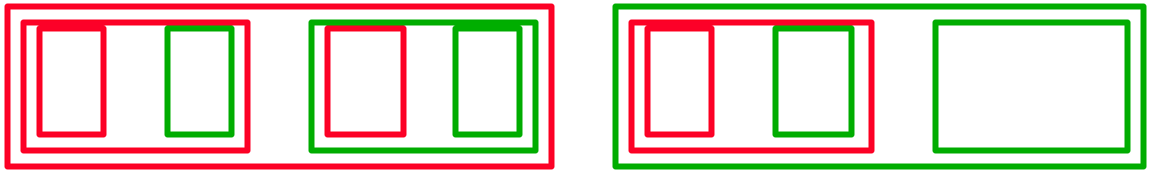
- ✓ Find predecessor (it is the rightmost node in the left subtree)
- ✓ Replace the data of the node to be deleted with predecessor's data
- ✓ Delete predecessor node

Trees - Searching

- **Binary search tree**
  - **Produces a sorted list by in-order traversal**



- **In order: A D E G H K L M N O P T V**



Thank You!!