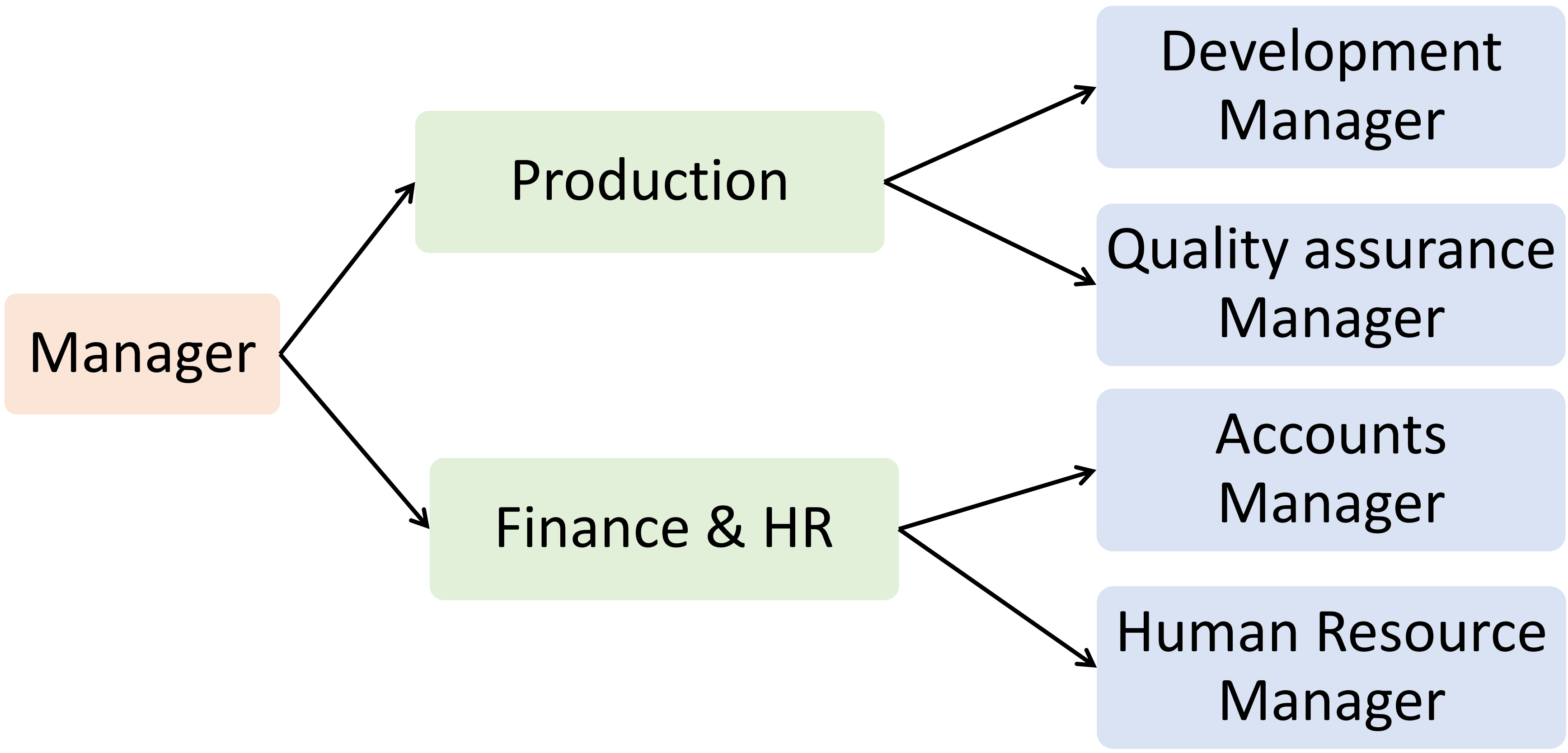
TREES

PROF. NAVRATI SAXENA

Non-Linear Data structures

- Each item can be connected to several other items
- Items have hierarchical relationship
- Each item is called node
- Provides efficient insertion and searching
- Very flexible data structures
- The different non-linear data structures are
  - Trees
  - Graphs

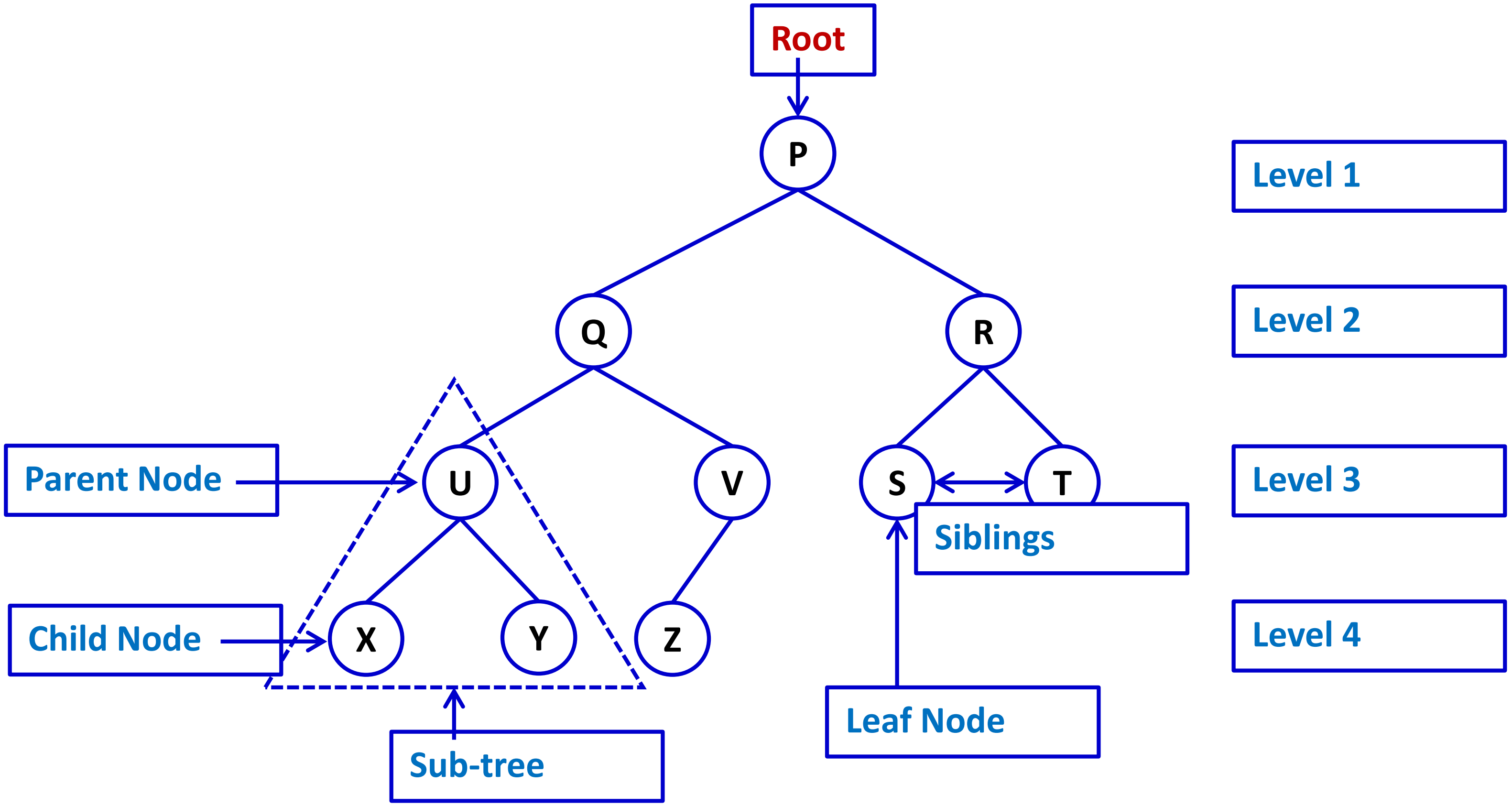
Trees



Manager
Production
Finance & HR
Development Manager
Quality assurance Manager
Accounts Manager
Human Resource Manager

Tree Terminology

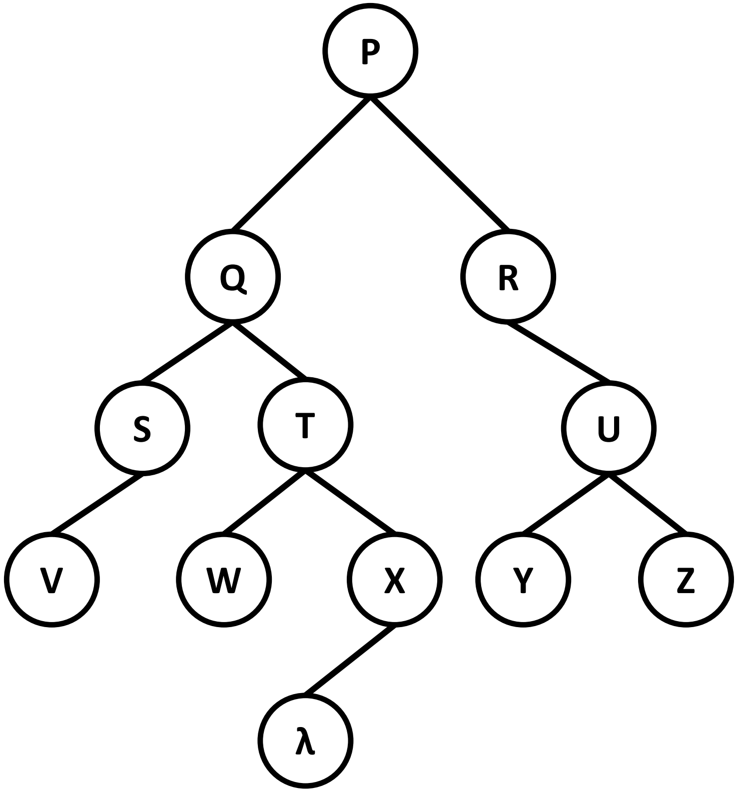
- Hierarchical organization
- Nodes connected by edges



Major Terminologies

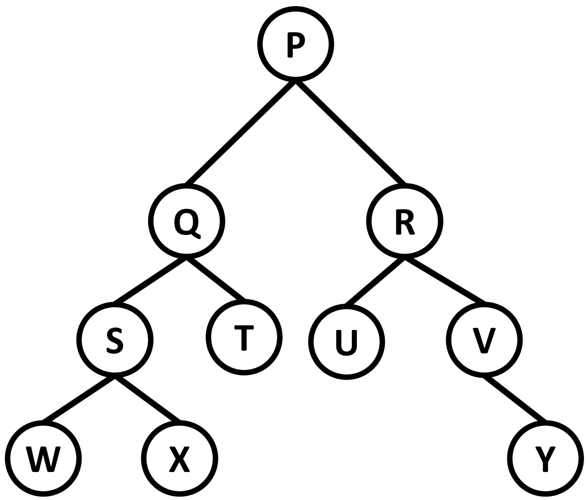
Root	Special node to refer the entire tree
Parent	Immediate predecessor of a node
Child	All immediate successors of a node
Siblings	All nodes having the same parent
Path	Successive edges from a source node to a destination node
Node Height	Maximum number of edges from a node to leaves
Tree Height	Height of the Root
Node Depth	Number of edges from Root to a node
Node Degree	Number of children of a node
Edge	Connection between any two nodes

Tree Terminology

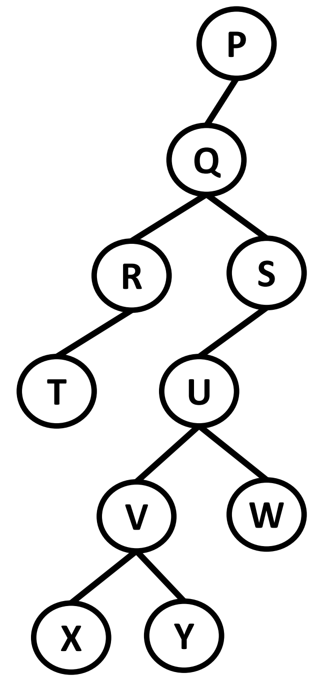


- **Size**: Number of nodes in the tree
  E.g. Tree in figure has **size = 12**
- **Depth of a node:** Distance (number of edges) from the root.
  - Depth of node P = 0
  - Depth of node S = 2
- **Height (*h*) of a tree:** Maximum depth of any node in the tree
- **Leaf Node**: Nodes of a tree at the maximum depth

Balanced Binary Trees



Balanced Binary Tree



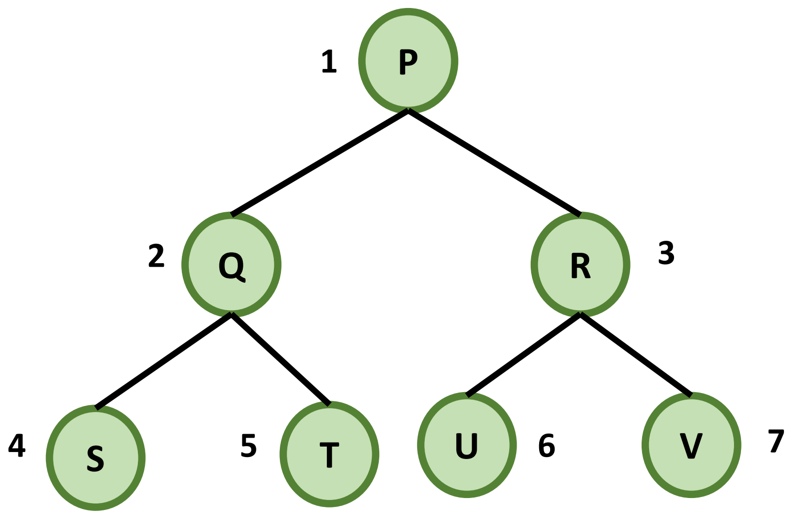
Unbalanced Binary Tree

Balanced Tree: Every level above the lowest is "full" (contains 2n nodes) *n = number of nodes in the previous level*

BINARY TREES

Binary Tree

- An ordered tree, each node having a maximum of 2 child nodes
- A binary tree consists of:
  - A node (called the root node)
  - Left and right sub trees
- Complete binary tree: binary tree with all the nodes having 2 subtrees

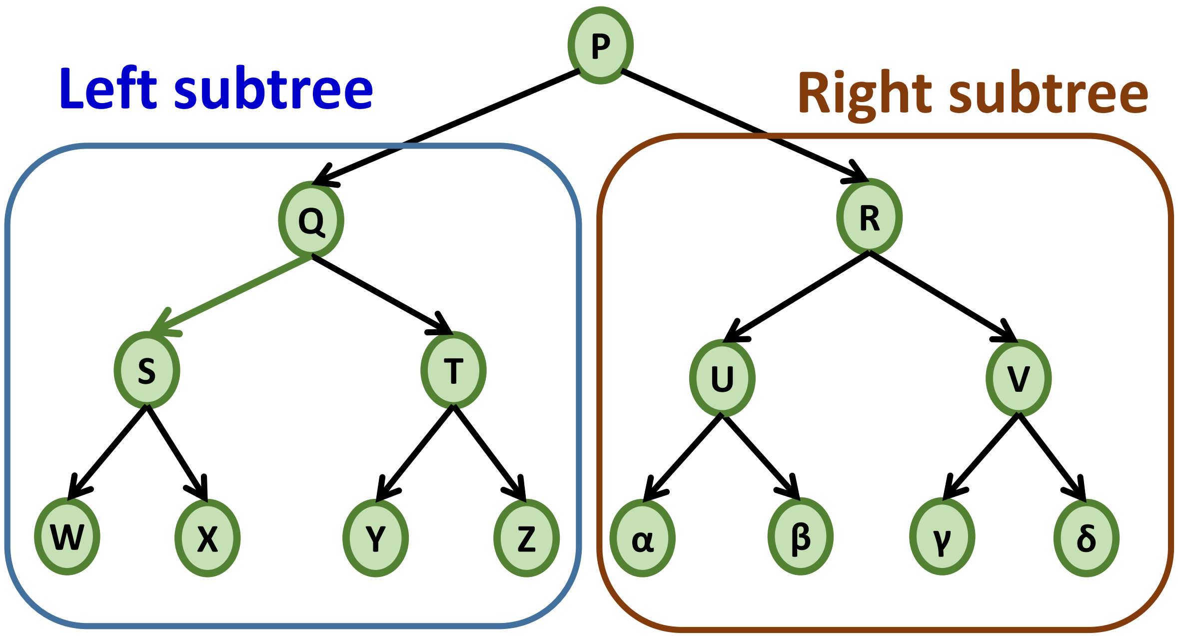


0	1	2	3	4	5	6	7
7	P	Q	R	S	T	U	V

COMPLETE BINARY TREE & ITS ARRAY REPRESENATTION

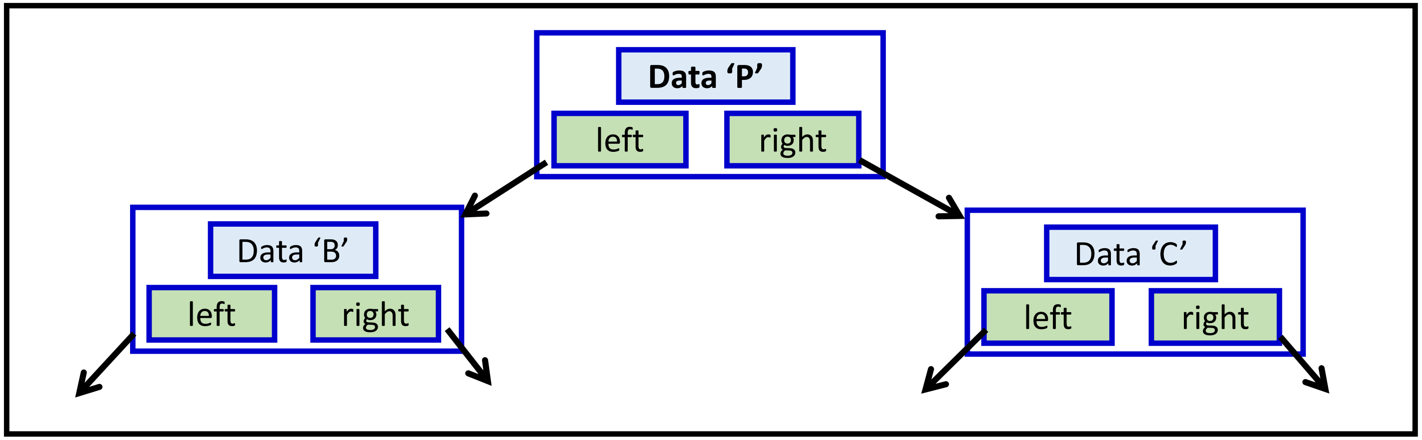
FORMAL DEFINITION

A binary tree is either empty or consists of a node called the root together with two binary trees called the left subtree and the right subtree.



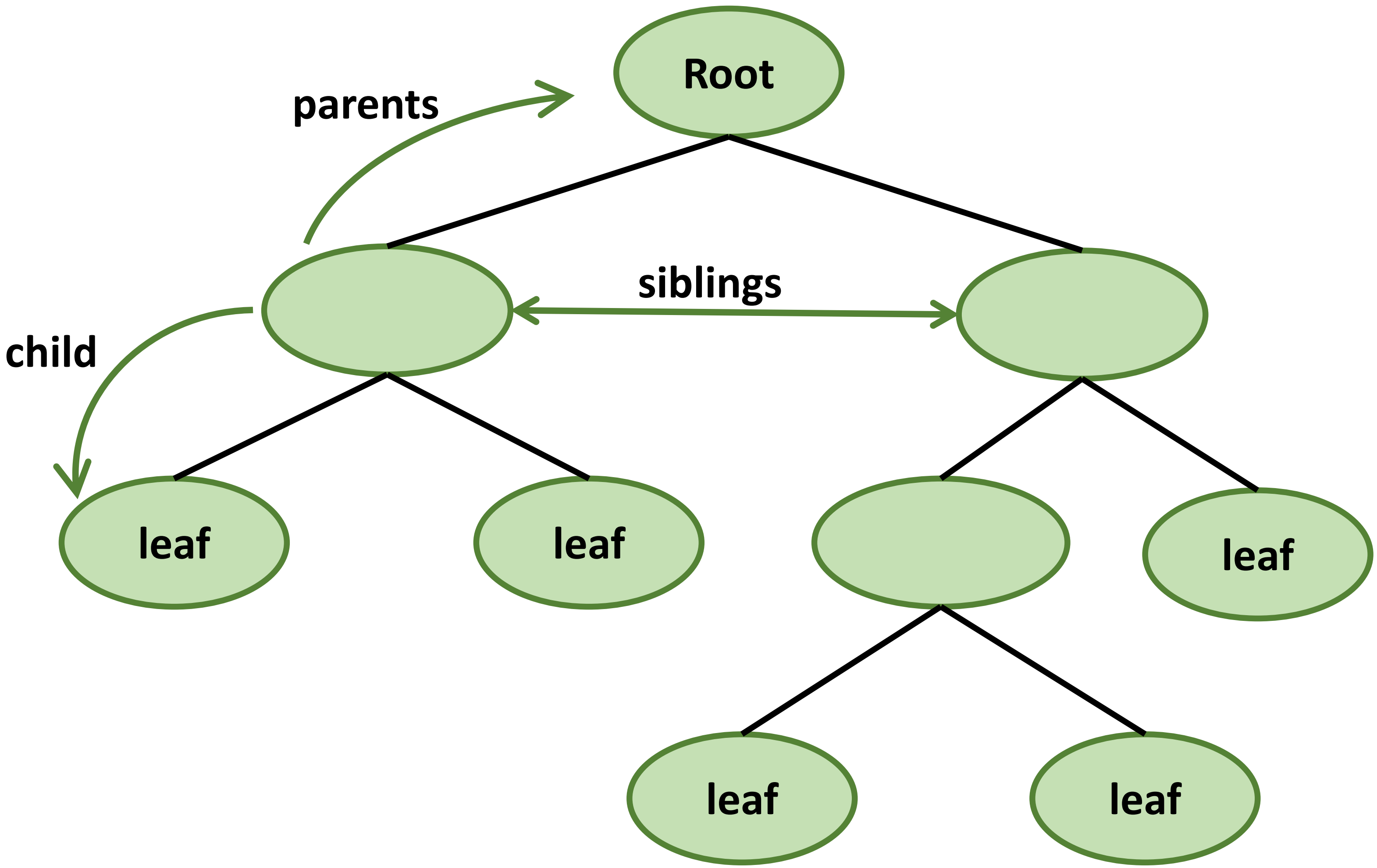
- **Number of Leaves (n) = 8**
- **Number of nodes (2n – 1) = 2 x 8 – 1 = 15**

Parts of a Binary Tree



- Maybe empty – contains no nodes
- If not empty, has a root node
- Root and every other node contains:
  - (1) A value (2) Pointer to a left child (may be null), and (3) Pointer to a right child (may be null)
- Each node is reachable from the root node by a *unique* path
- Leaf node: A node with neither a left child nor a right child

A Sample Binary Tree



Types of Binary Trees

- **Full binary tree/Proper binary tree/2-tree/strictly binary tree:**
  Every node other than the leaf node has two children

- **Complete binary tree/Perfect binary tree/ full binary tree**: All *leaves* are at the same *depth* or same *level*, and in which every parent has two children

- **Balanced binary tree**: The *depth* of the two subtrees of every node never differ by more than 1

Binary Tree Traversals

- **Recursive Definition**: Consists of root, a left subtree, and a right subtree
- Traverse/walk: visit each node in the binary tree exactly once
- Tree traversals are naturally **recursive**
- Since a binary tree has **three "parts**," there are **six** possible ways to traverse the binary tree:

1. **root, left, right**
2. **left, root, right**
3. **left, right, root**

4. **root, right, left**
5. **right, root, left**
6. **right, left, root**

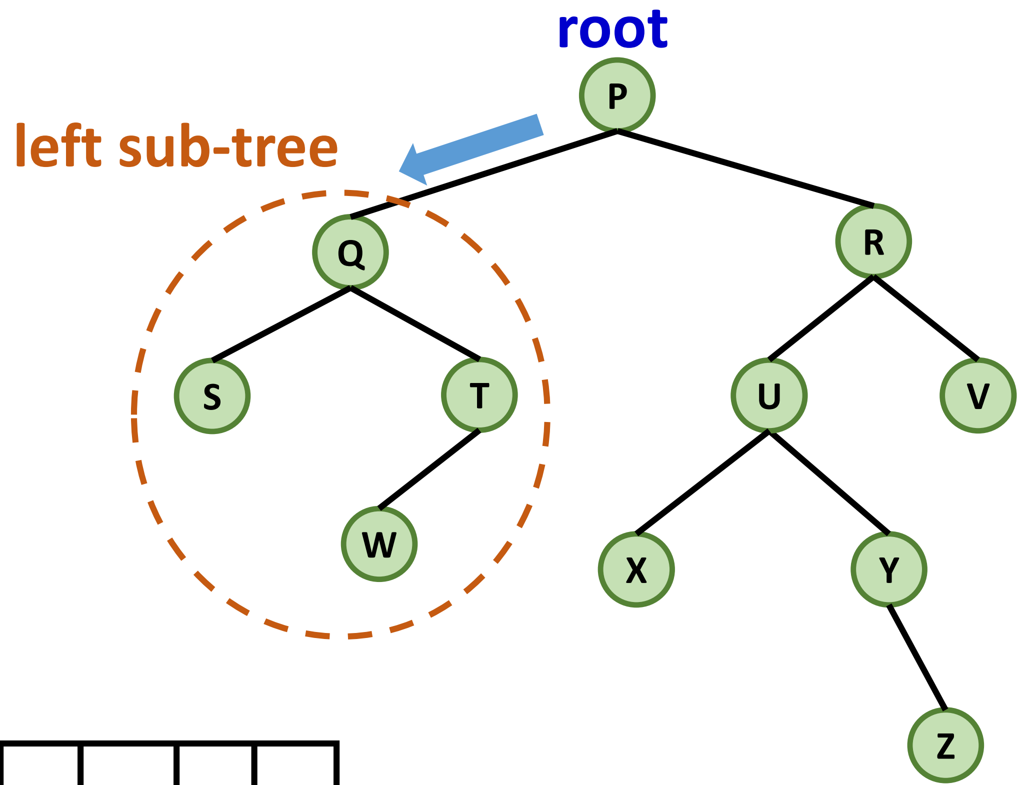
- **left** and **right** are **recursive calls** to **left** and **right sub-trees** respectively

DFS (Depth First Search) – Types

1. root, left, right	→	Pre-order Traversal
2. left, root, right	→	In-order Traversal
3. left, right, root	→	Post-order Traversal

DFS: Pre-order Traversal (1/14)

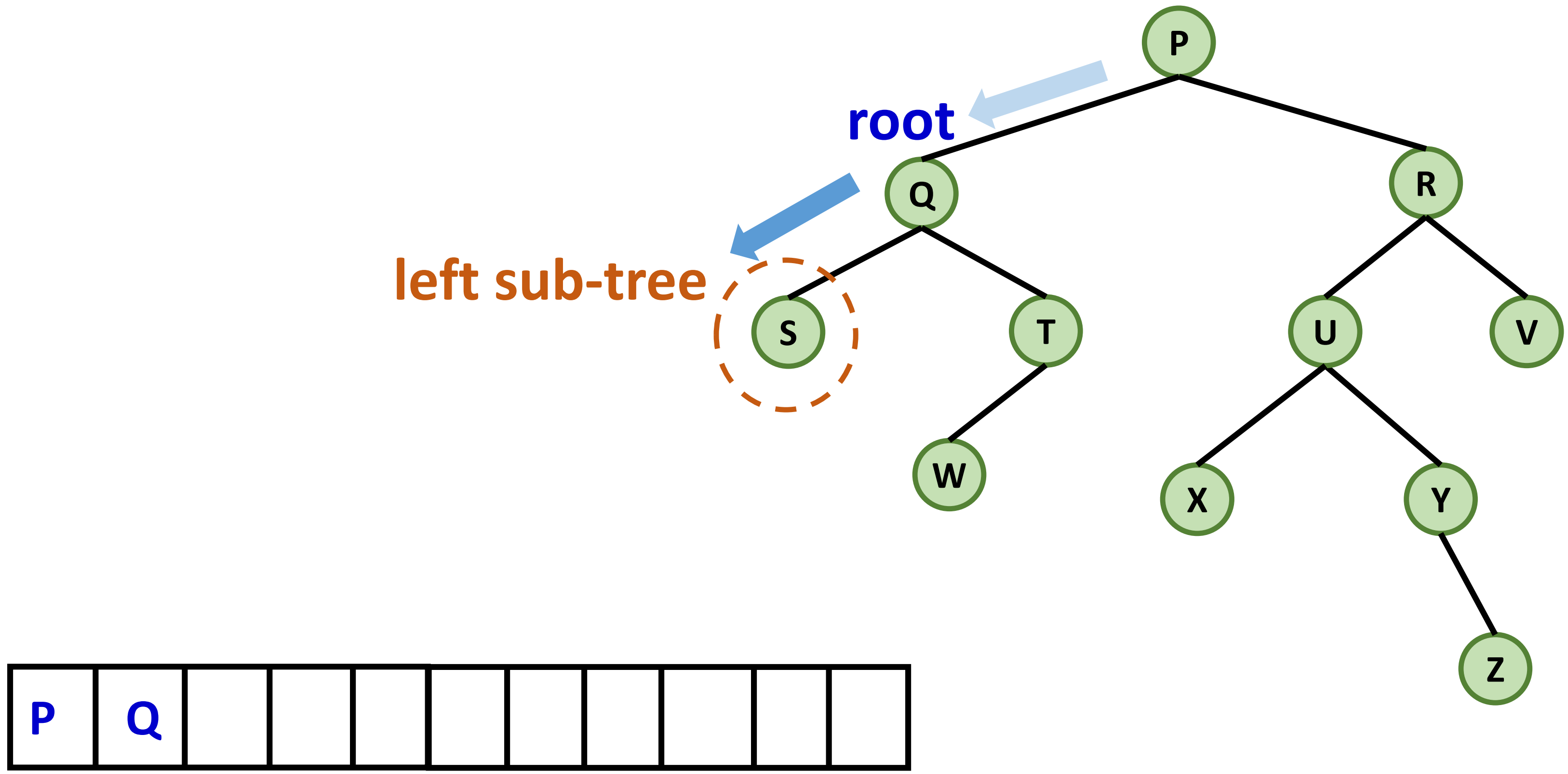
- **root -> left -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



P										

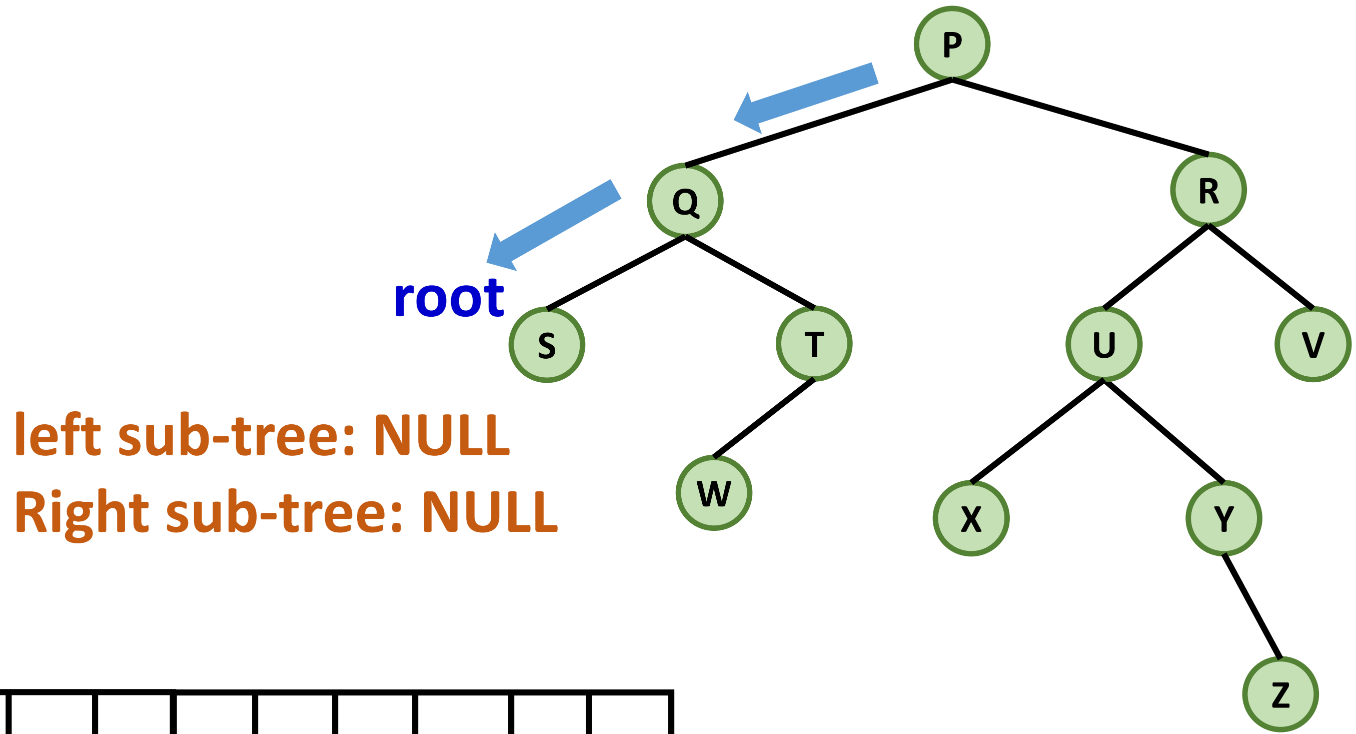
DFS: Pre-order Traversal (2/14)

- **root -> left -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



DFS: Pre-order Traversal (3/14)

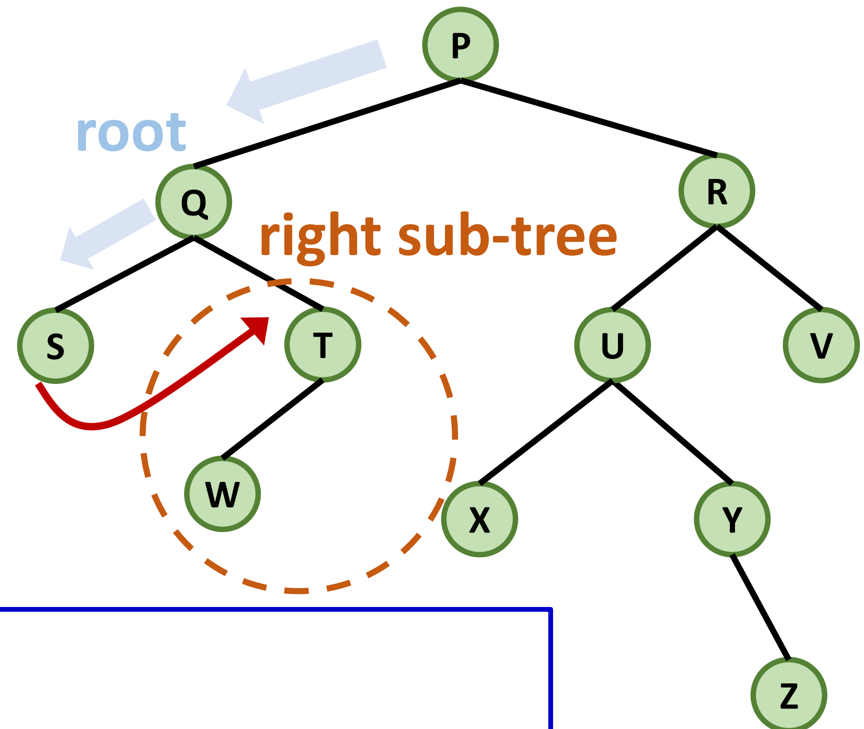
- **root -> left -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



P	Q	S								

DFS: Pre-order Traversal (4/14)

root -> left -> right



Root S has no left or right sub-tree

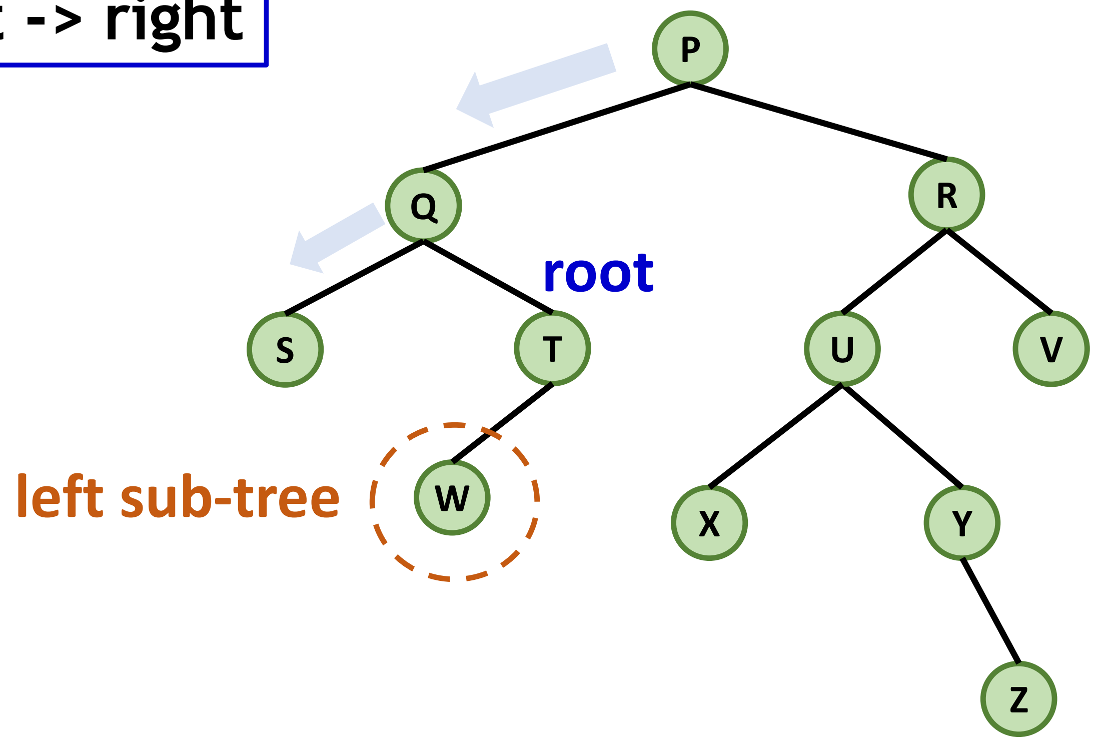
⇒ Completed all left sub-trees originated at root P, Q, S

⇒ Visit the right sub-tree of “root (S) = Q”

P	Q	S								

DFS: Pre-order Traversal (5/14)

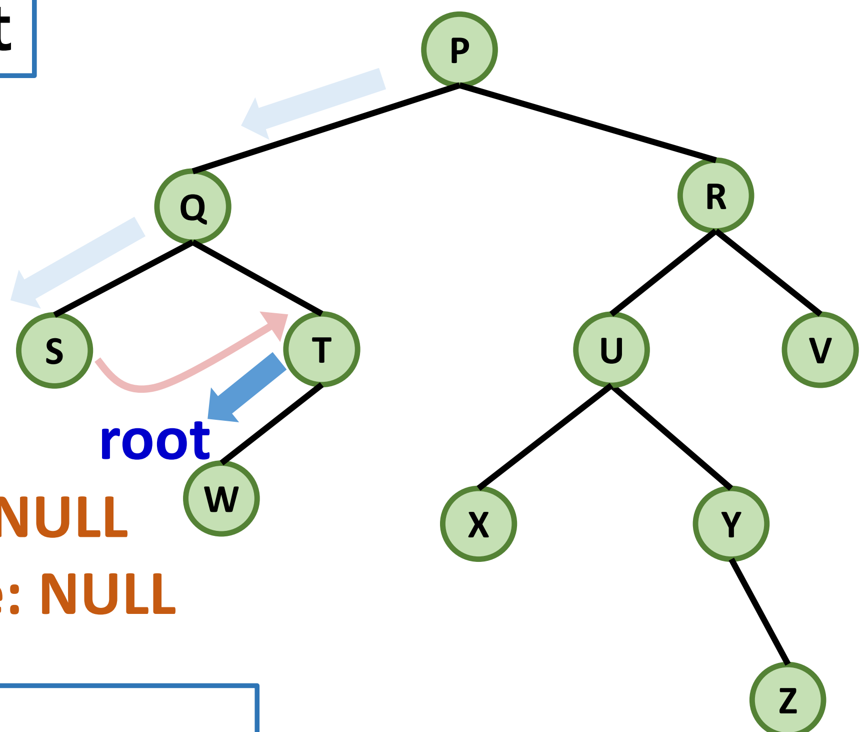
root -> left -> right



P	Q	S	T							

DFS: Pre-order Traversal (6/14)

root, left, right



left sub-tree: NULL

Right sub-tree: NULL

- Following same as before

P	Q	S	T	W						

DFS: Pre-order Traversal (7/14)

root -> left -> right

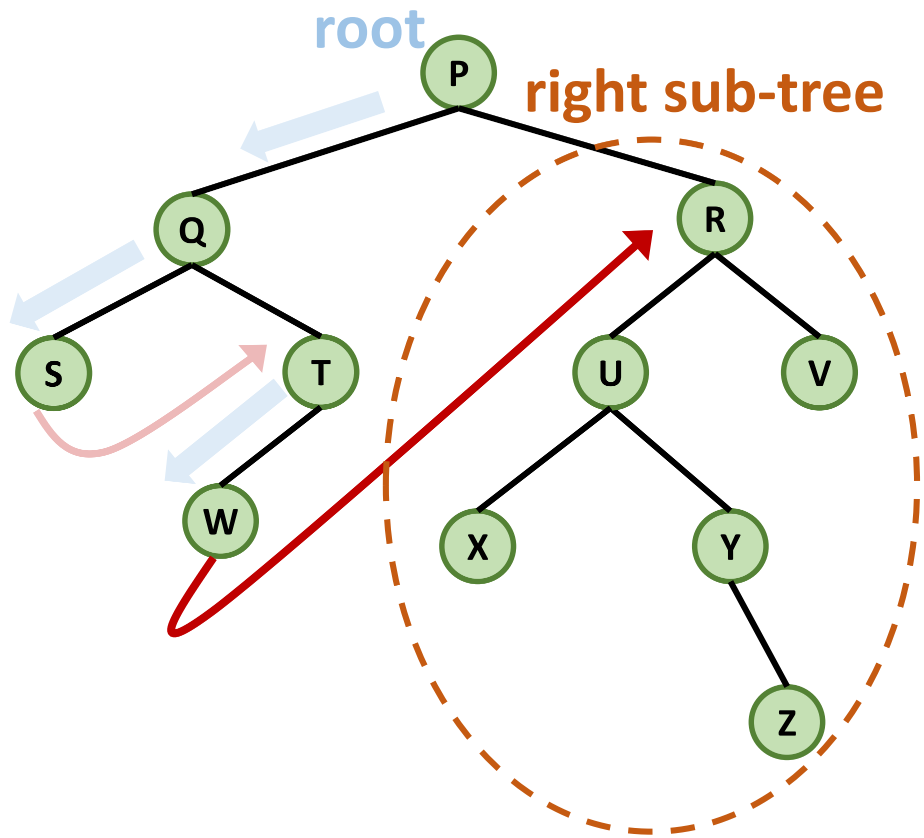
W: No left or right sub-tree

Root(W) = T: No right sub-tree

Root(T) = Q: Right sub-tree visited

Root(Q) = P: Left sub-tree visited

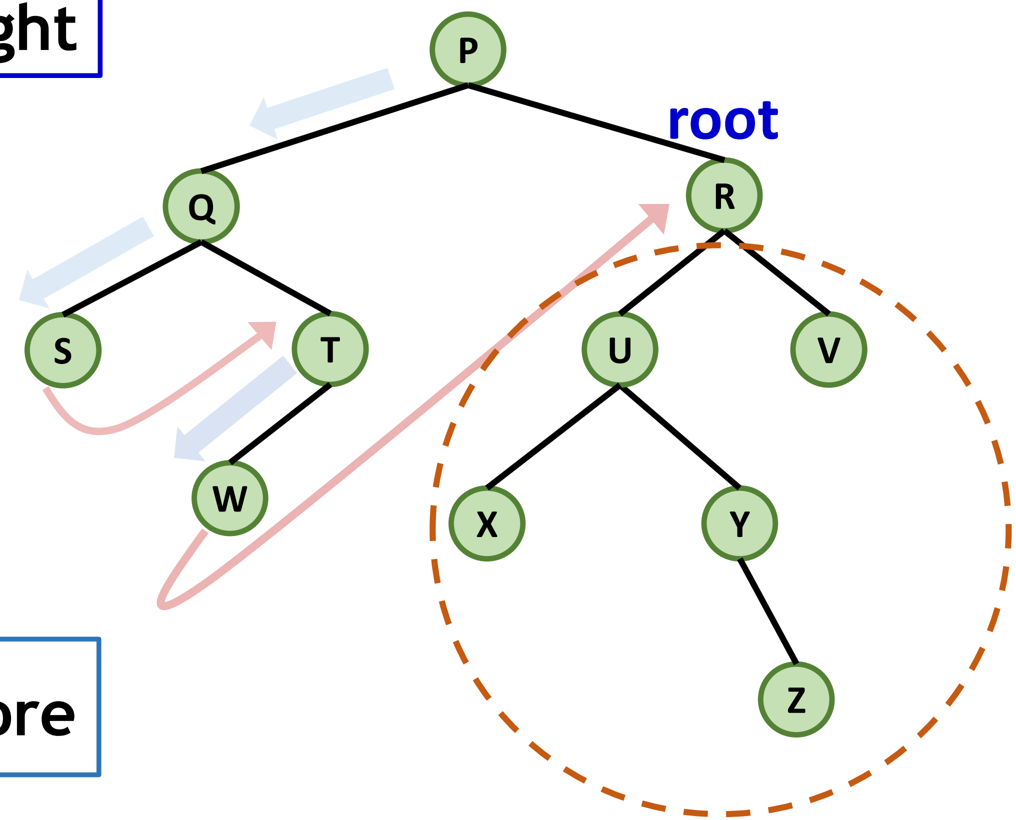
Visit right sub-tree



P	Q	S	T	W						

DFS: Pre-order Traversal (8/14)

root -> left -> right

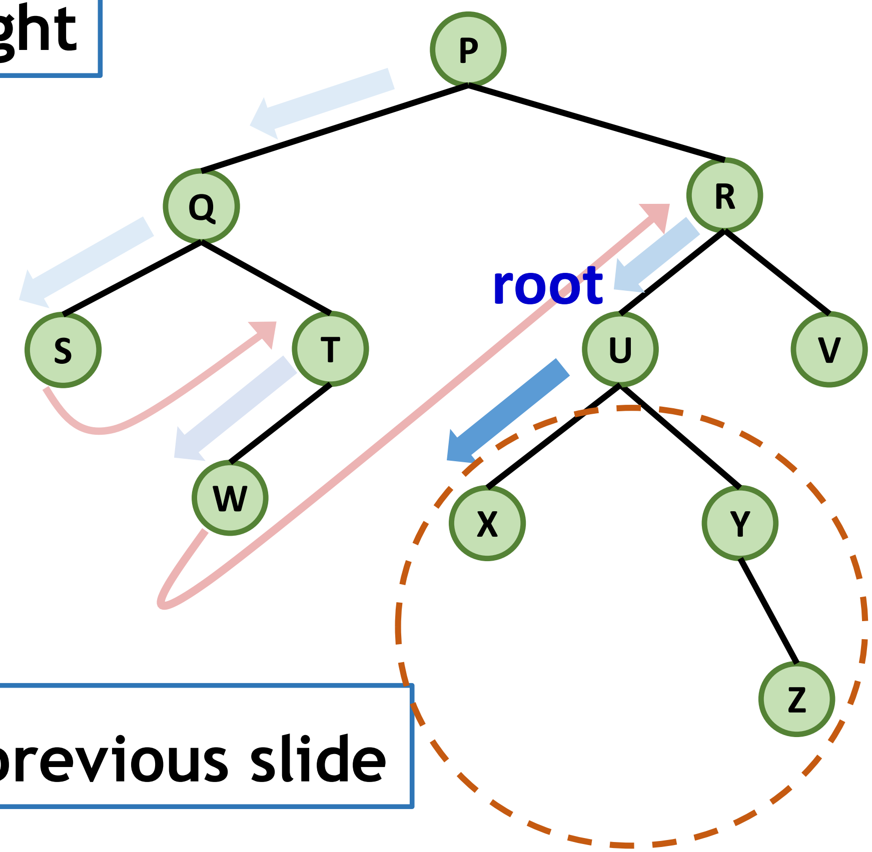


Do same as before

P	Q	S	T	W	R					

DFS: Pre-order Traversal (9/14)

root -> left -> right

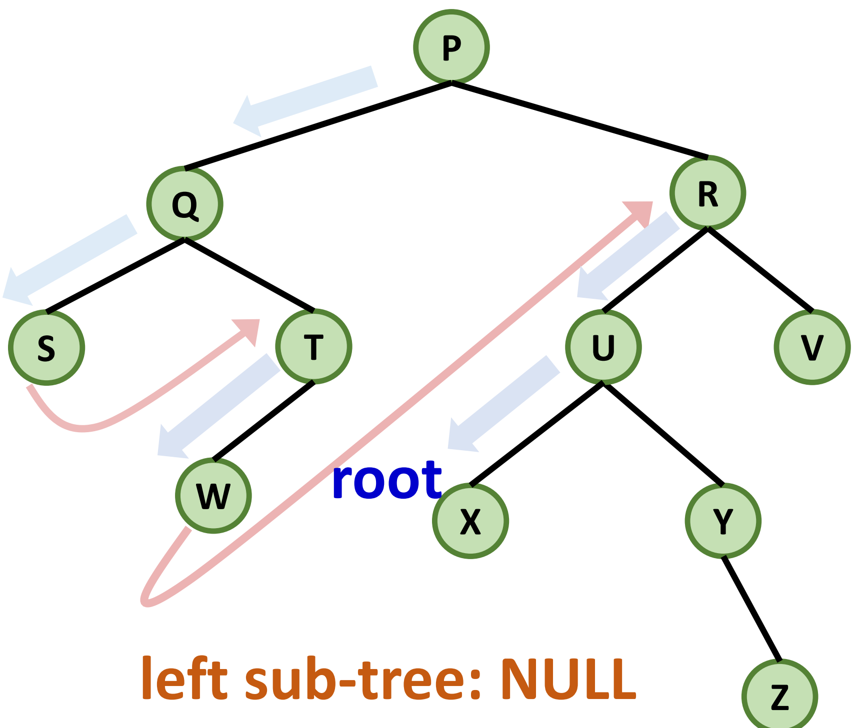


Following same as previous slide

P	Q	S	T	W	R	U				

DFS: Pre-order Traversal (10/14)

root -> left -> right



left sub-tree: NULL
Right sub-tree: NULL

Root X has no left or right sub-tree

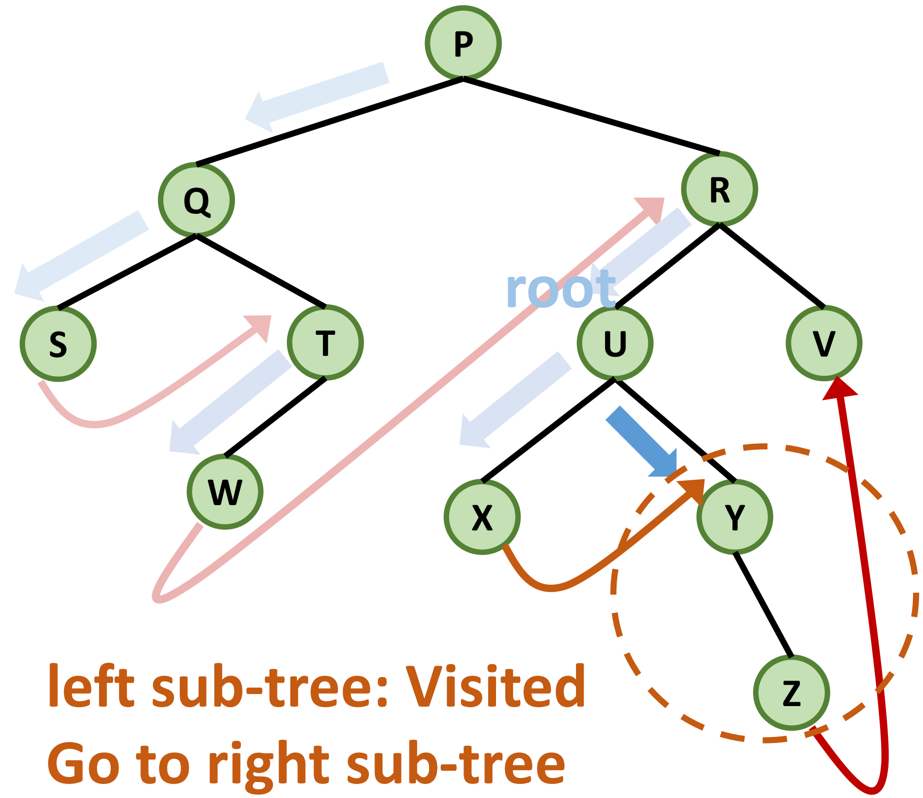
$\Rightarrow$ **Completed all left sub-trees originated at root R, U, X**

$\Rightarrow$ **Visit the right sub-tree of "root (X) = U"**

P	Q	S	T	W	R	U	X			

DFS: Pre-order Traversal (11/14)

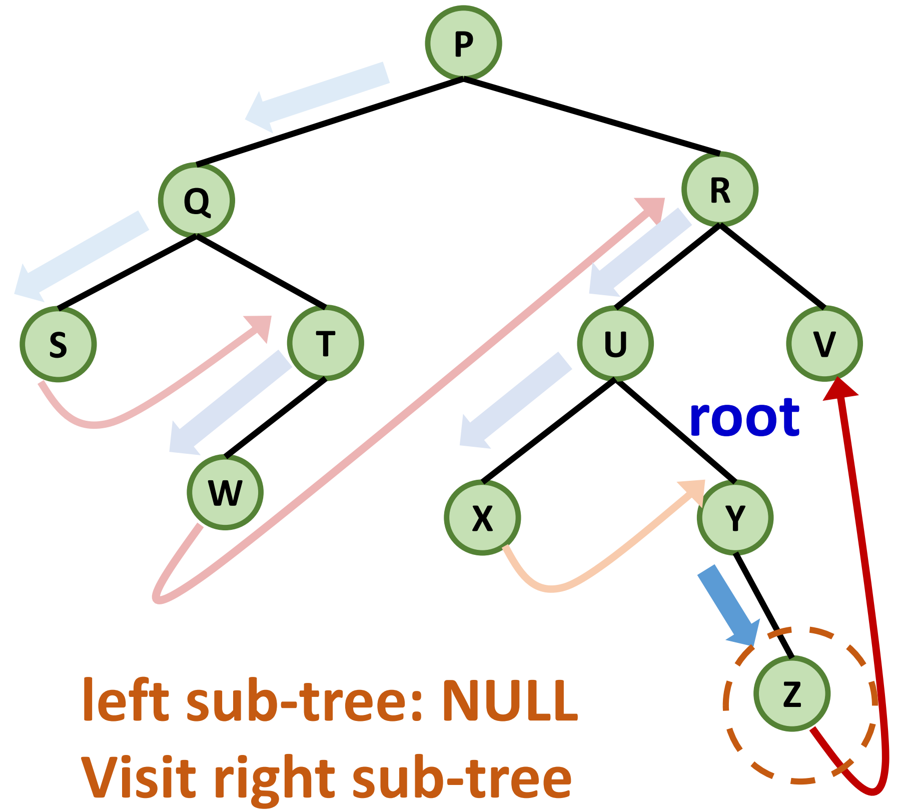
root -> left -> right



P	Q	S	T	W	R	U	X			

DFS: Pre-order Traversal (12/14)

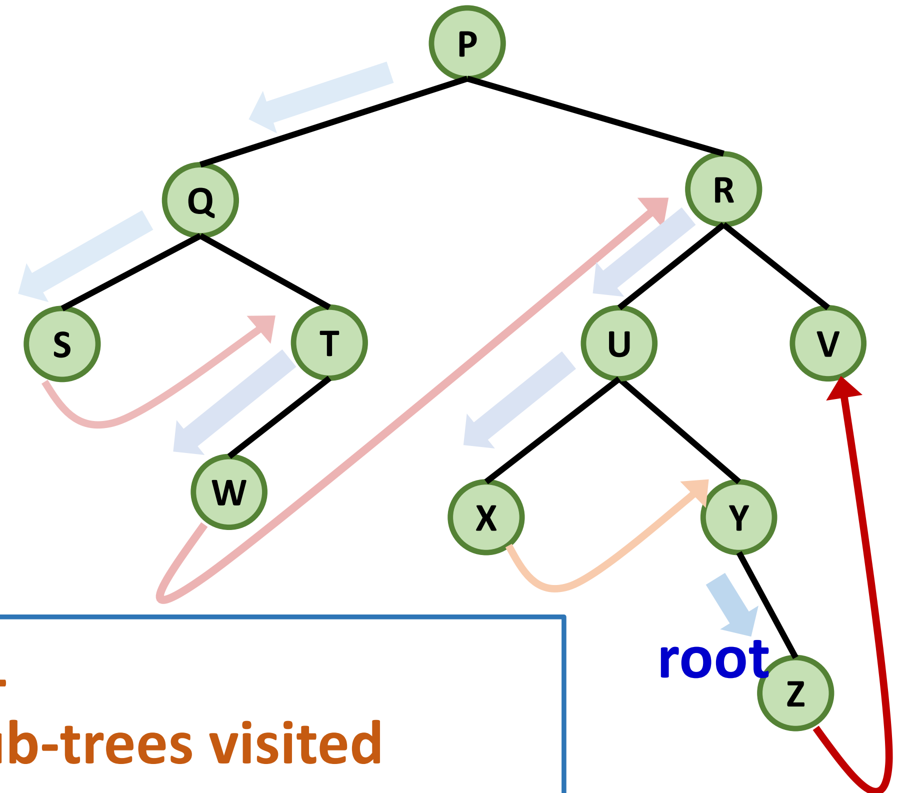
root -> left -> right



P	Q	S	T	W	R	U	X	Y		

DFS: Pre-order Traversal (13/14)

root -> left -> right

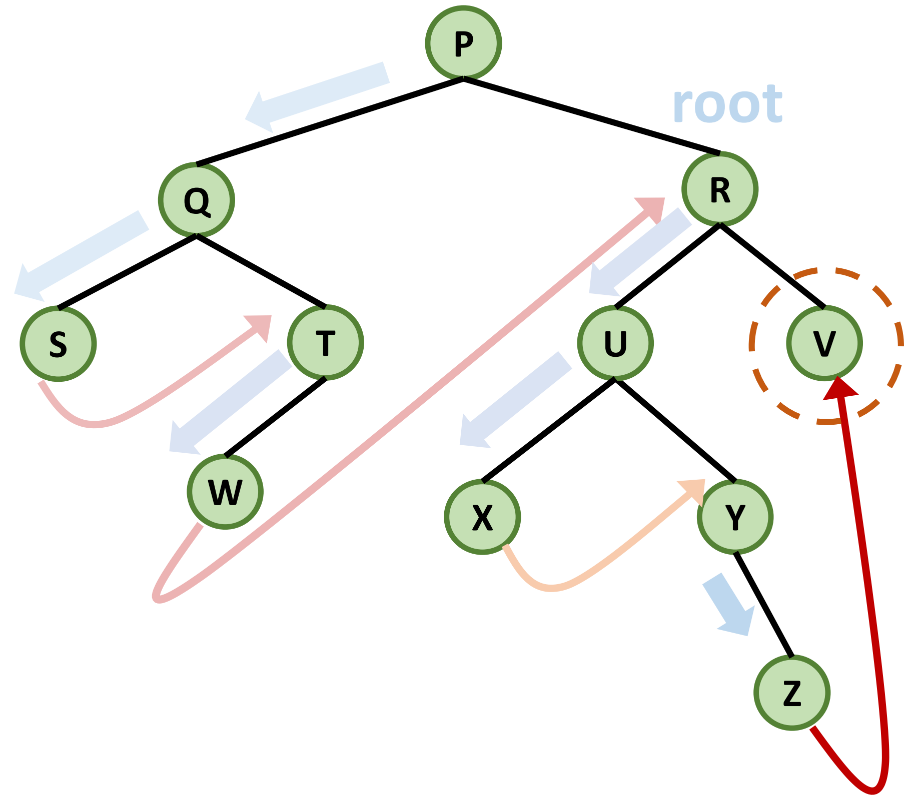


- **left and right sub-tree: NULL**
- **Root (Z) = Y: left and right sub-trees visited**
- **Root (Y) = U: left and right sub-trees visited**
- **Root (U)=R: left sub-tree visited => visit right**

P	Q	S	T	W	R	U	X	Y	Z	

DFS: Pre-order Traversal (14/14)

root -> left -> right



P	Q	S	T	W	R	U	X	Y	Z	V

Thank you!