Trees
Chapter 11

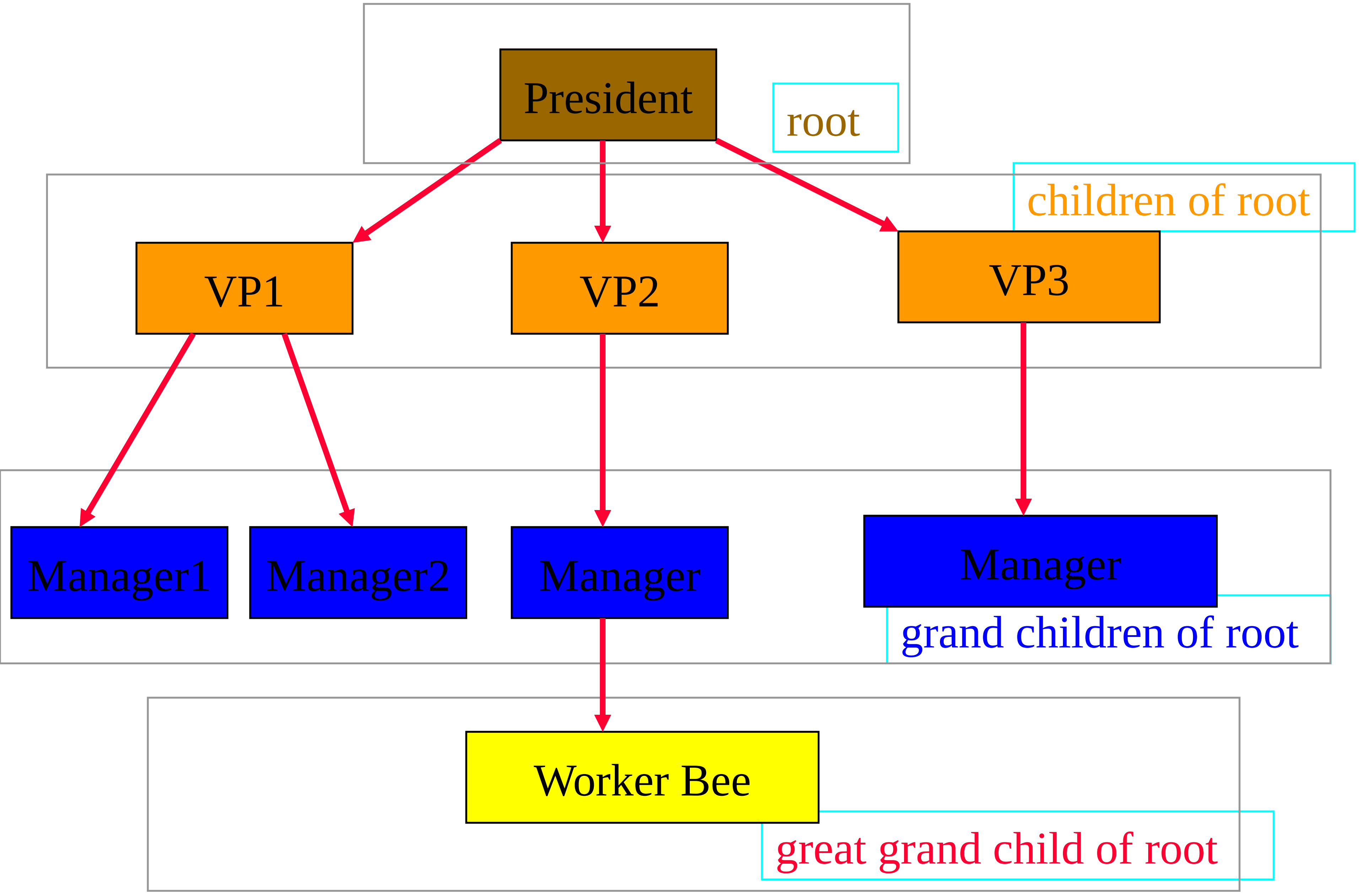
Linear Lists And Trees

- Linear lists are useful for serially ordered data.
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week.
  - Months in a year.
  - Students in this class.
- Trees are useful for hierarchically ordered data.
  - Employees of a corporation.
    - President, vice presidents, managers, and so on.

Hierarchical Data And Trees

- The element at the top of the hierarchy is the root.
- Elements next in the hierarchy are the children of the root.
- Elements next in the hierarchy are the grandchildren of the root, and so on.
- Elements that have no children are leaves.

Example Tree

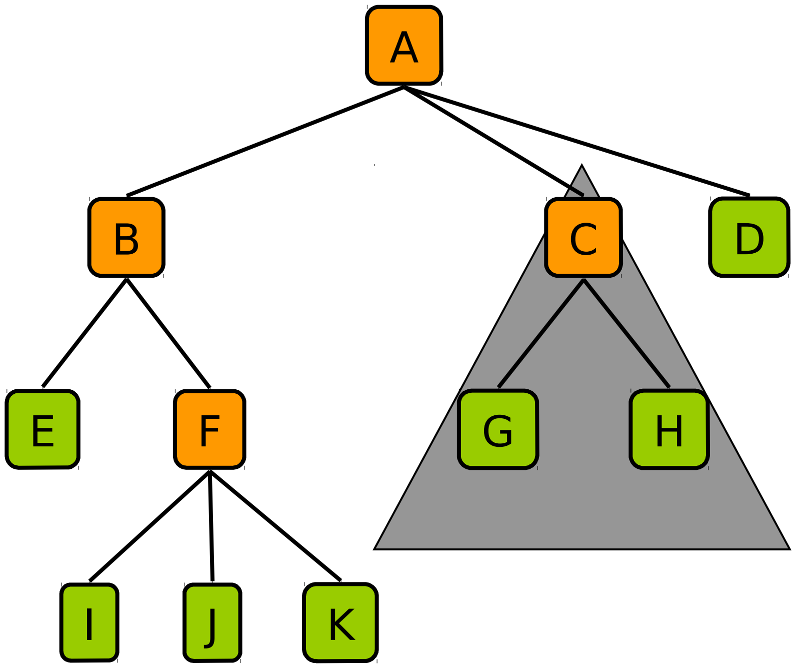


Definition

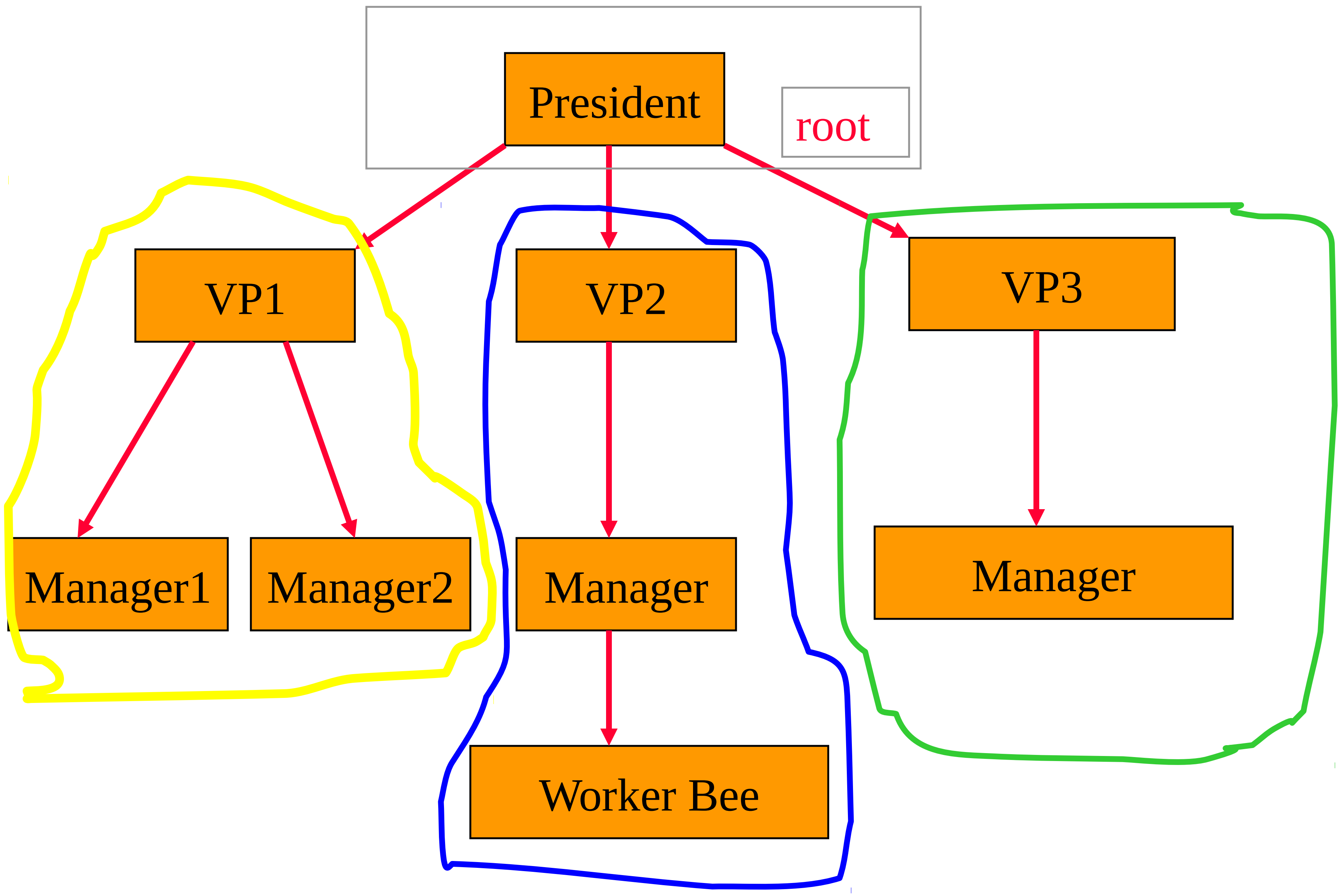
- A tree t is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of t.

Tree Terminology

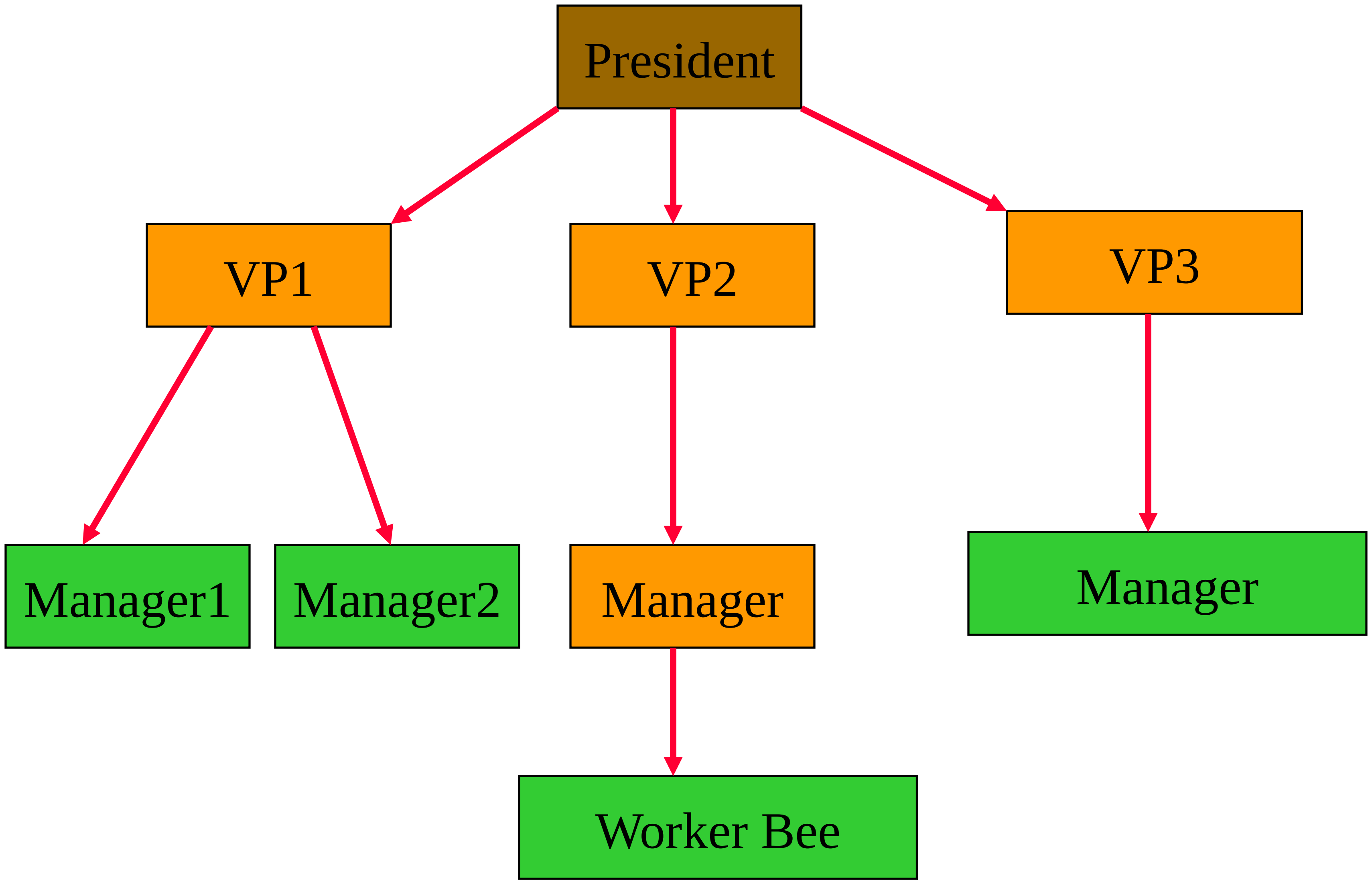
- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Degree** of a tree: the maximum number of its node.
- **Subtree**: tree consisting of a node and its descendants



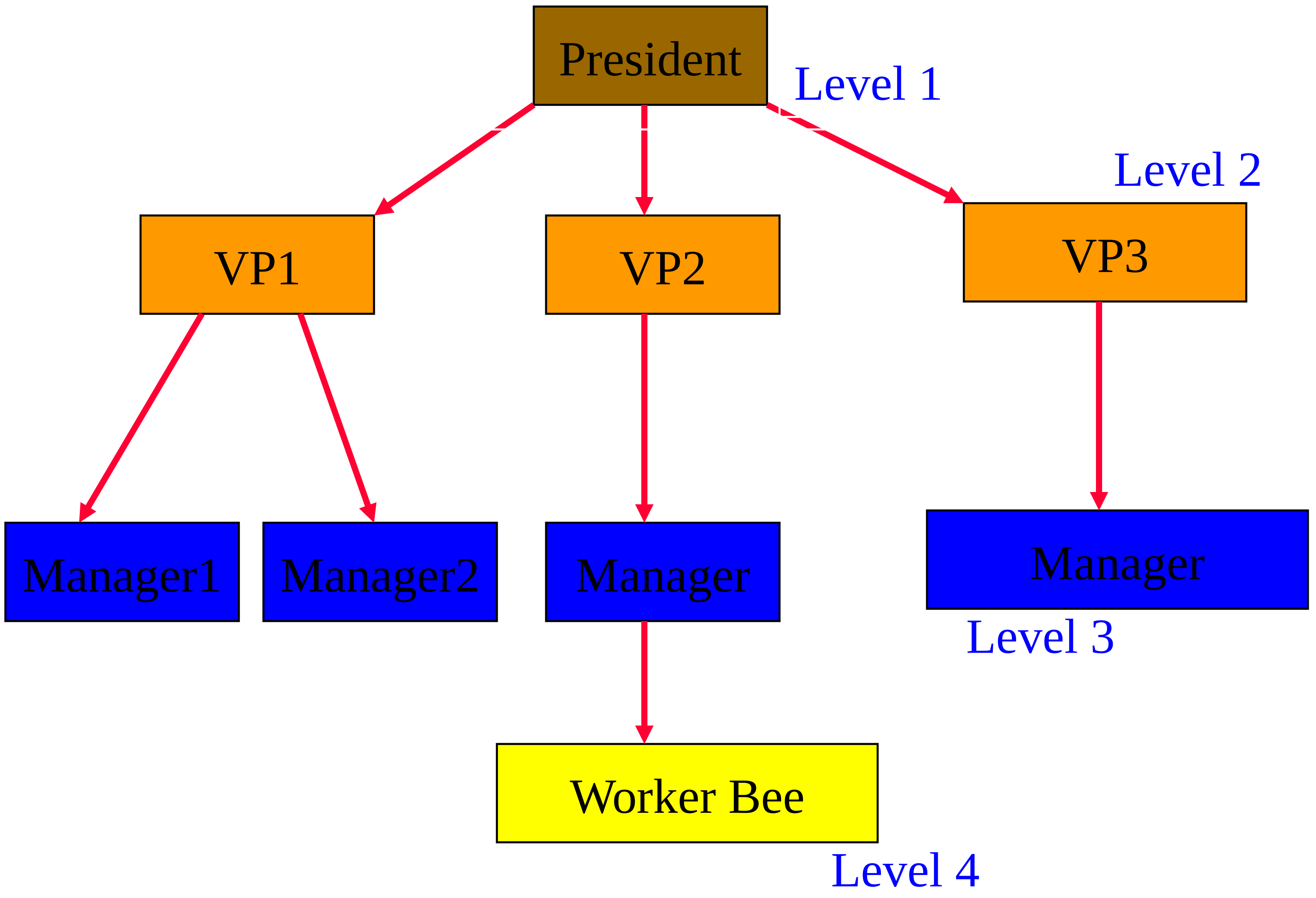
Subtrees


President
root
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

Leaves


President
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

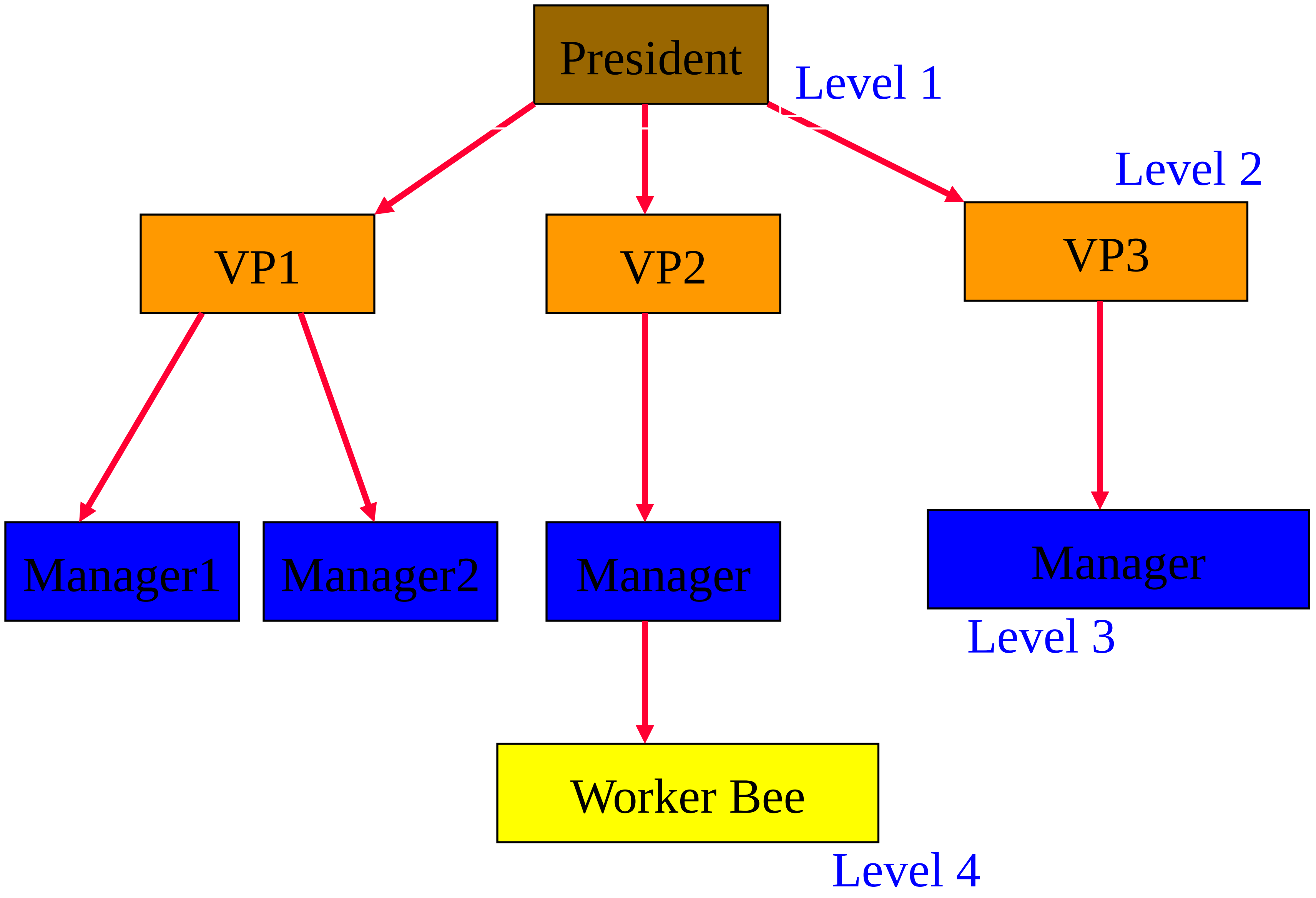
Levels


President
Level 1
VP1
VP2
VP3
Level 2
Manager1
Manager2
Manager
Manager
Level 3
Worker Bee
Level 4

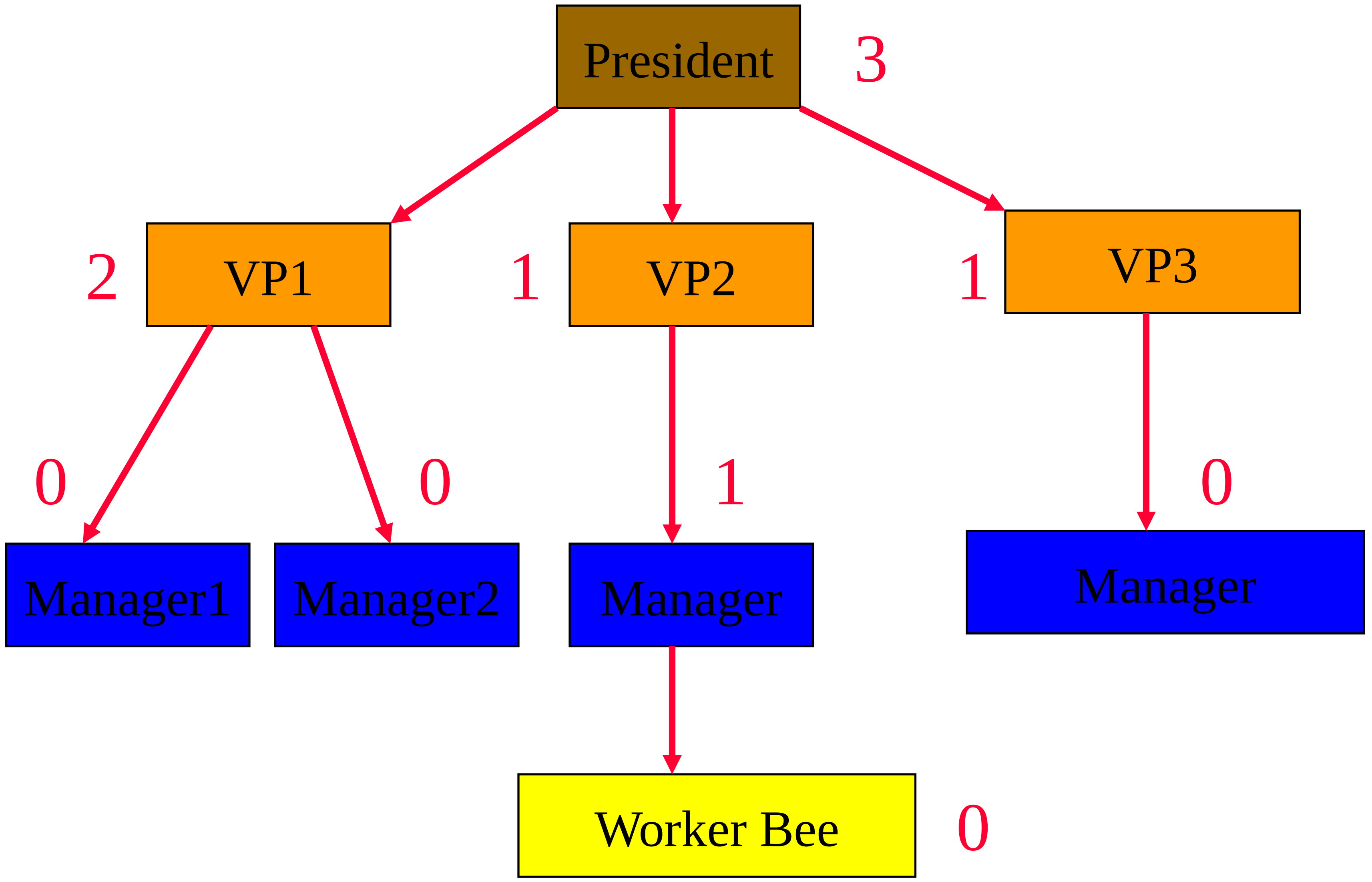
Caution

- Some texts start level numbers at 0 rather than at 1.
- Root is at level 0.
- Its children are at level 1.
- The grand children of the root are at level 2.
- And so on.
- We shall number levels with the root at level 1.

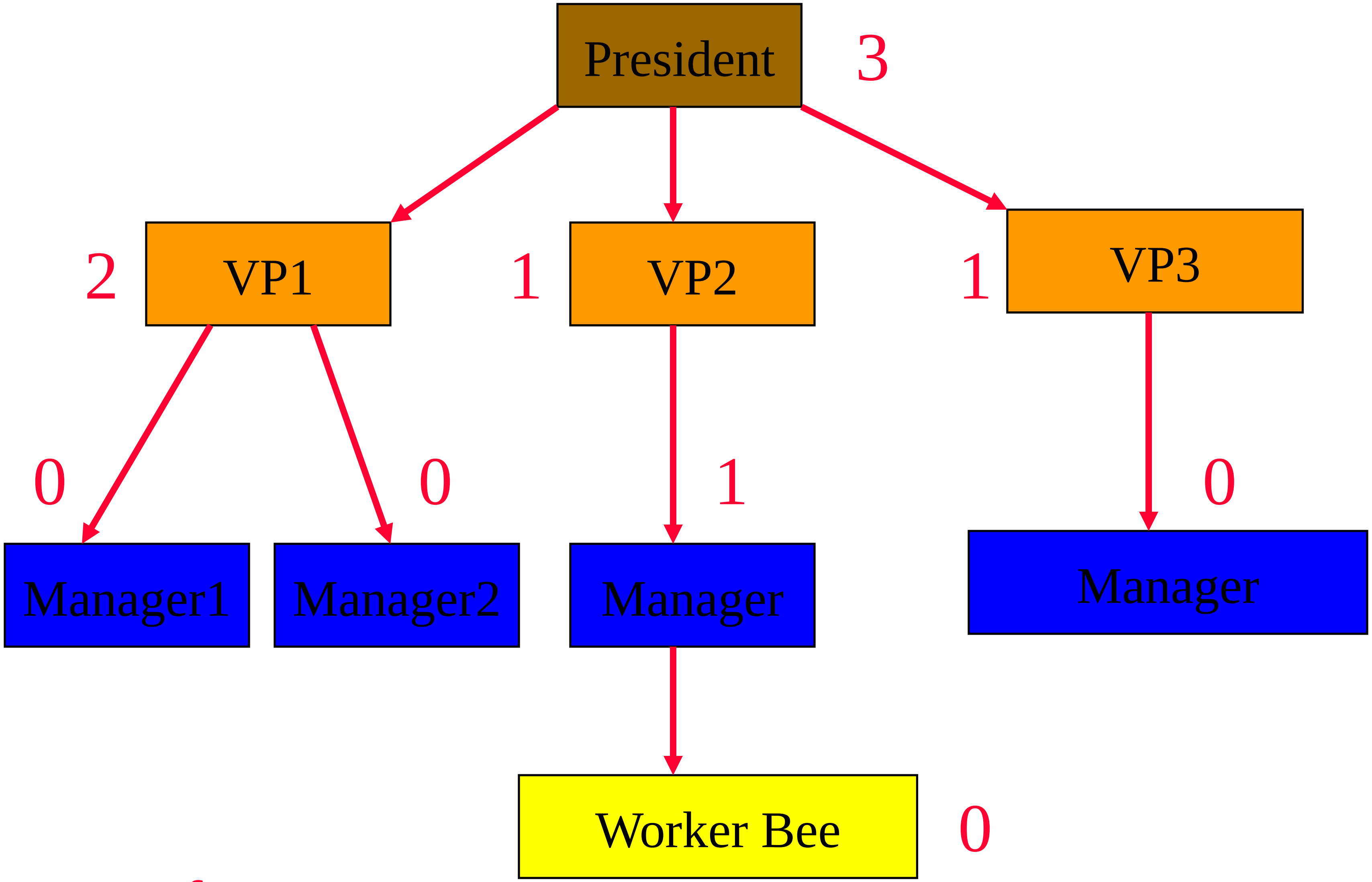
height = depth = number of levels



Node Degree = Number Of Children



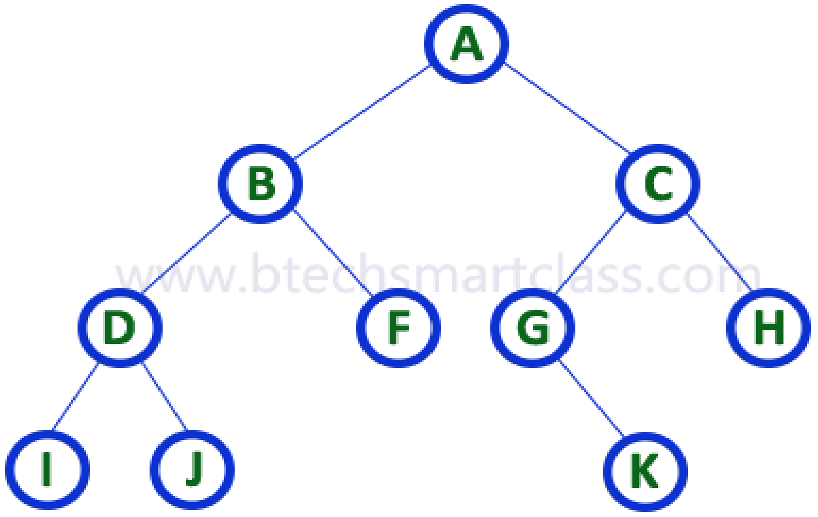
Tree Degree = Max Node Degree



Degree of tree = 3.

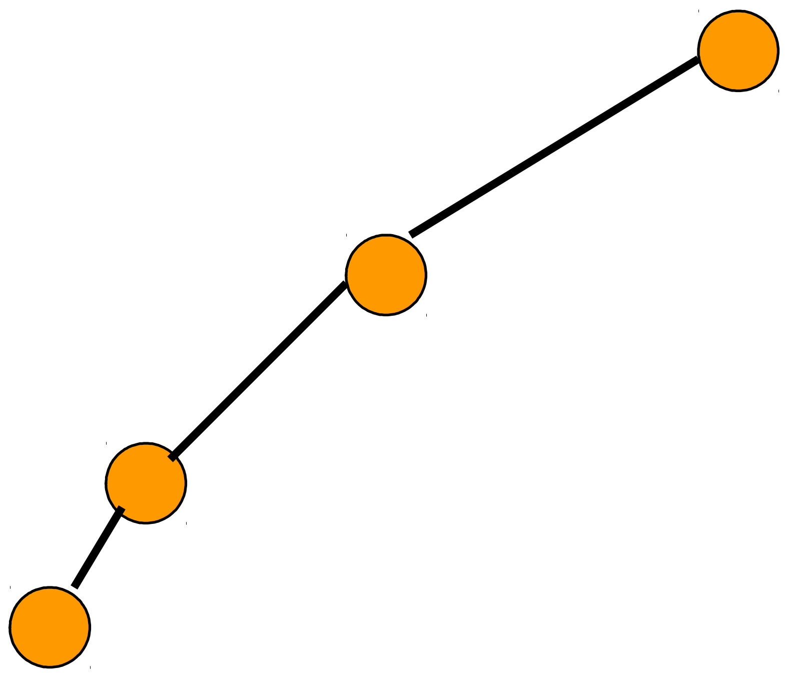
Binary Tree

- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.



Minimum Number Of Nodes

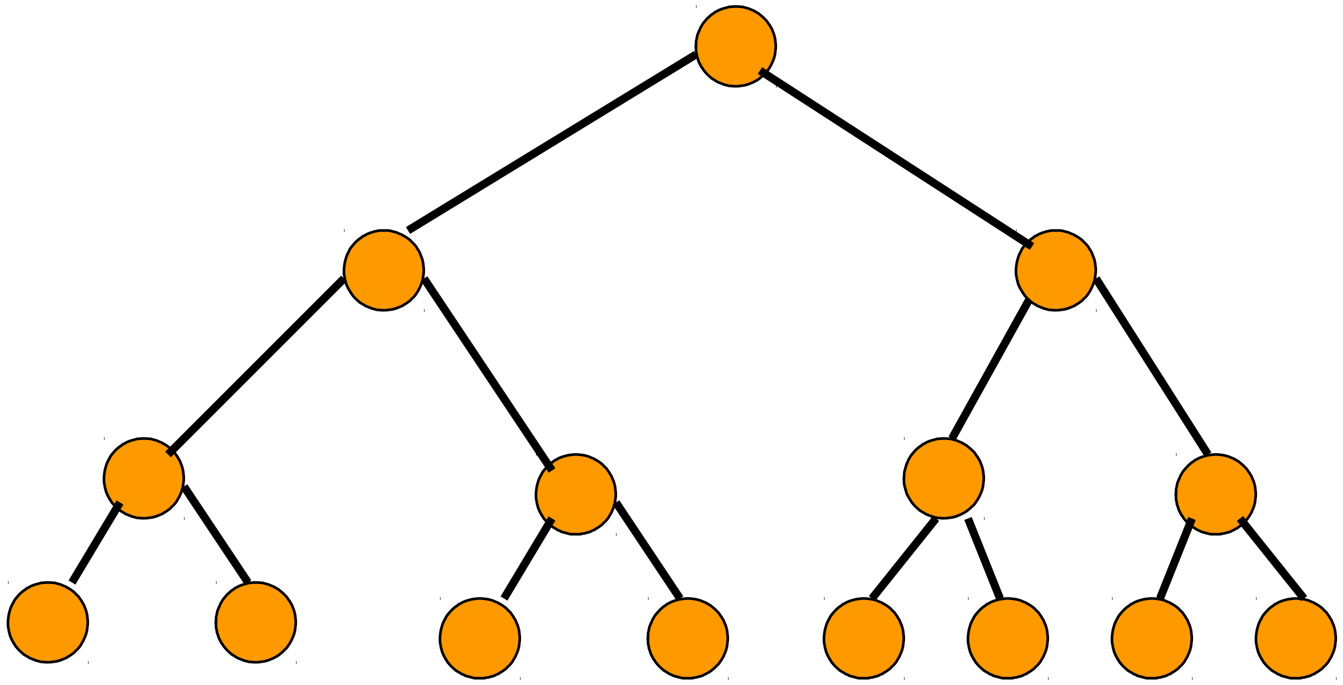
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

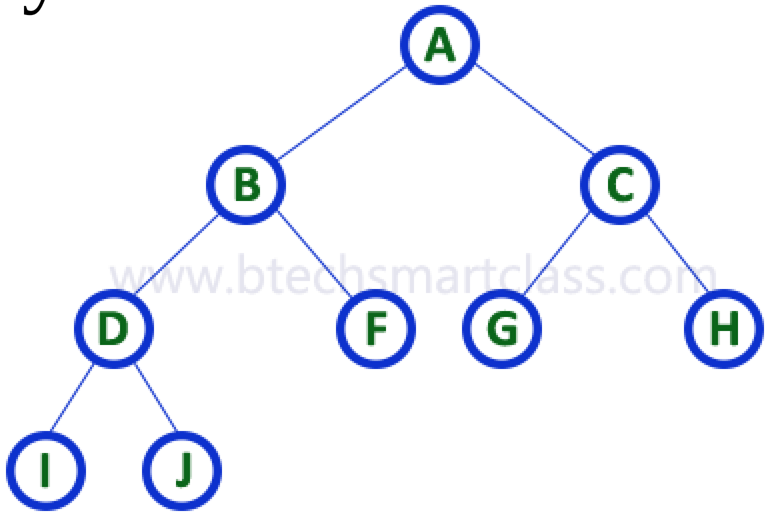
$= 2^h - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h.
- $h <= n <= 2^h - 1$
- $\log_2(n+1) <= h <= n$

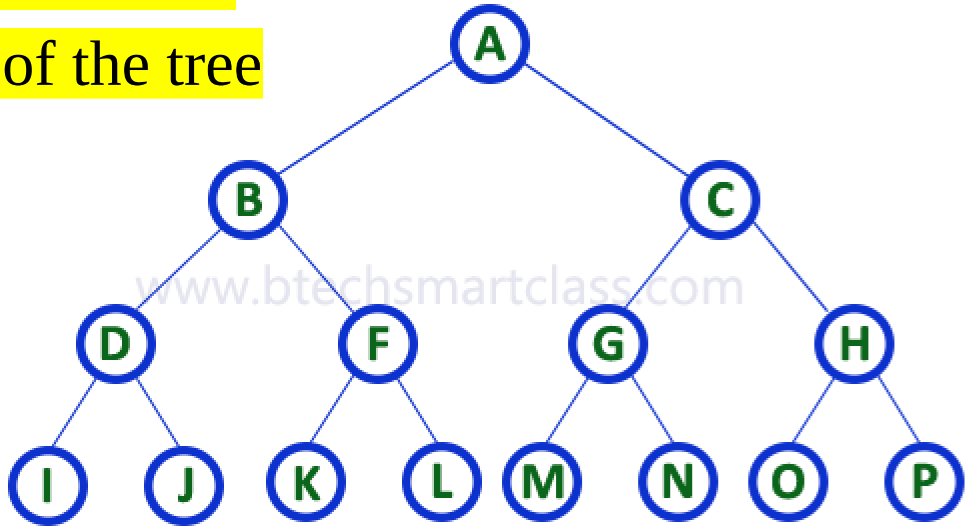
Full Binary Tree

- A binary tree in which every node has either two or zero number of children is called Full Binary Tree



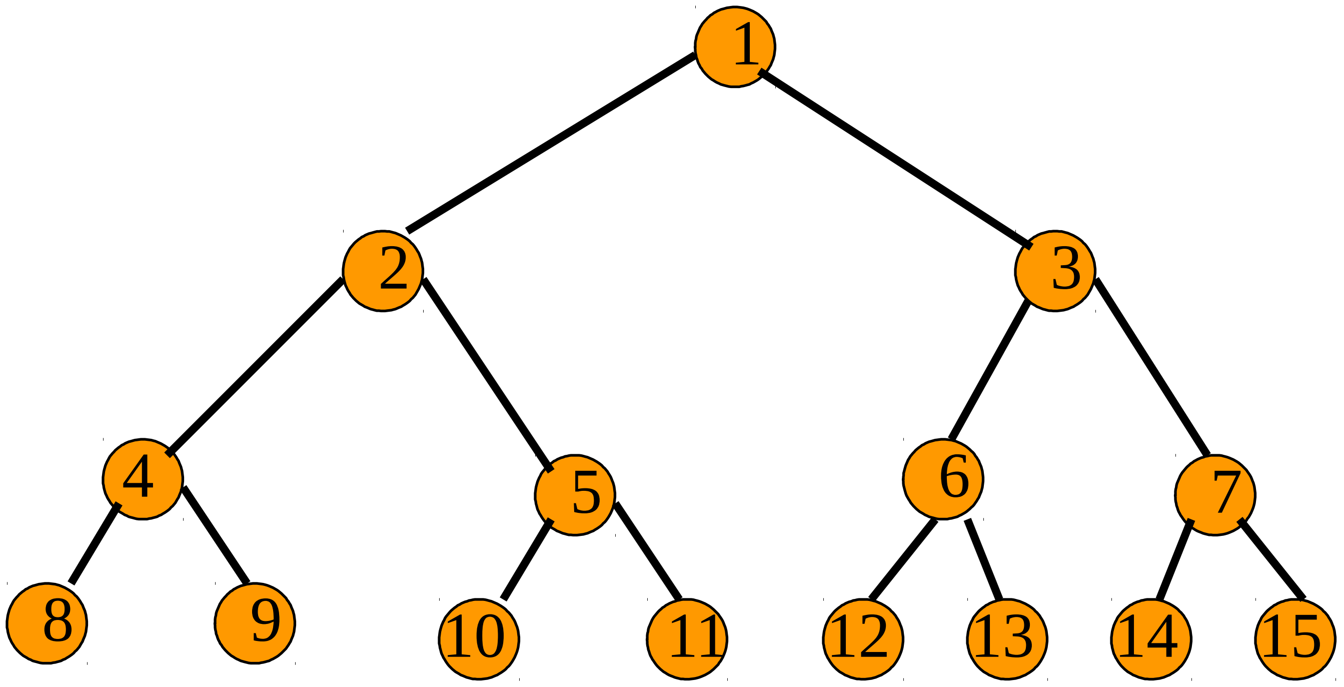
Complete Binary Tree

- A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree. Complete binary tree is also called as **Perfect Binary Tree**
- **Number of nodes = $2^{d+1} - 1$**
- **Number of leaf nodes = 2^{d}**
- Where, d – Depth of the tree

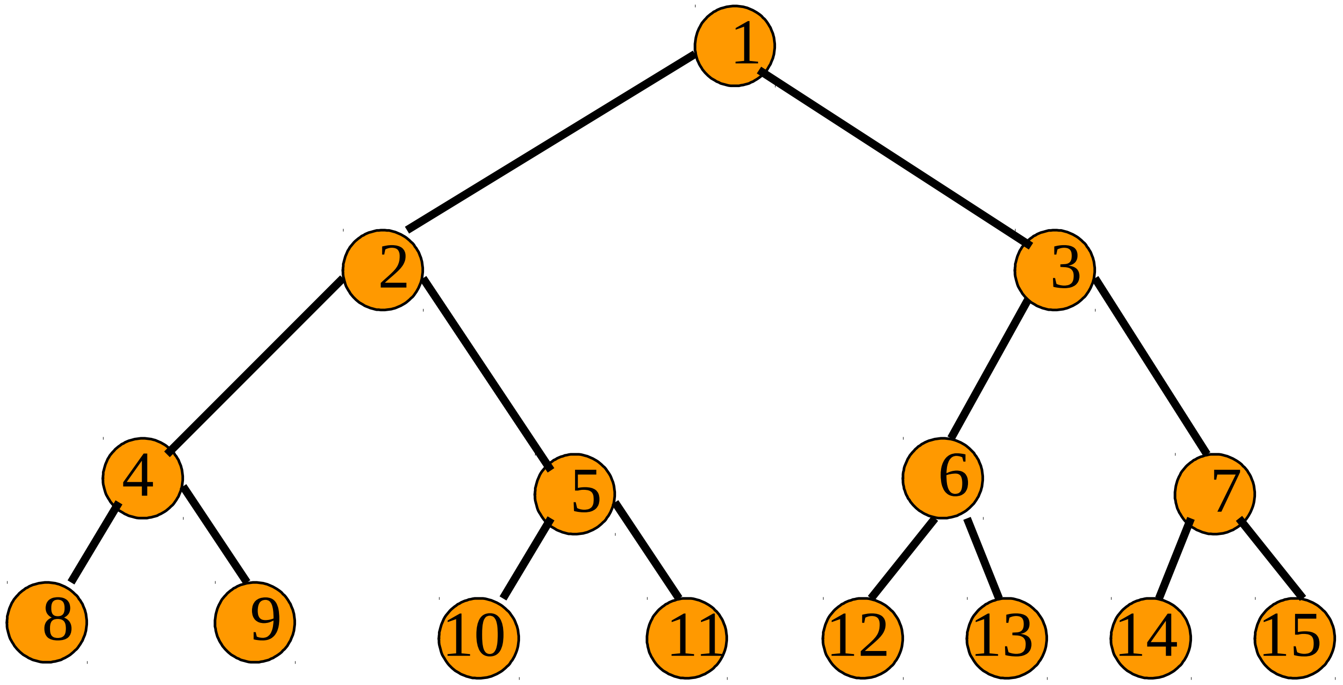


Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.

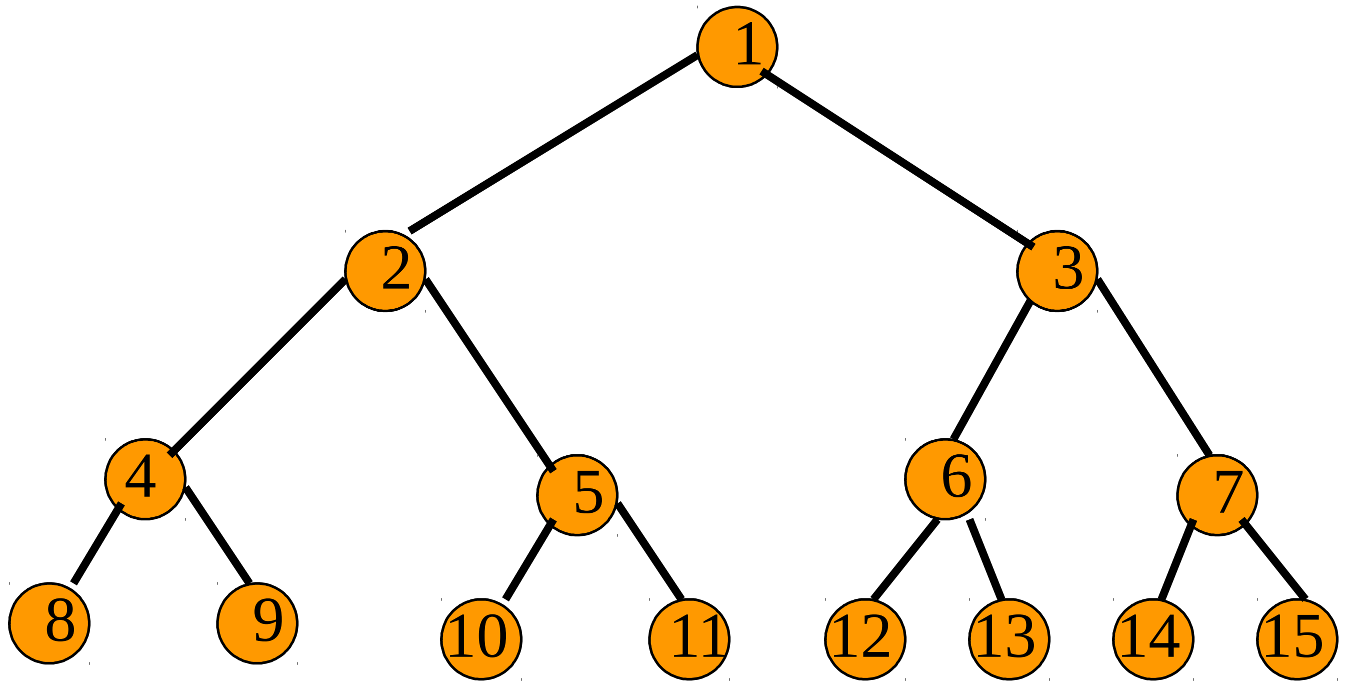


Node Number Properties



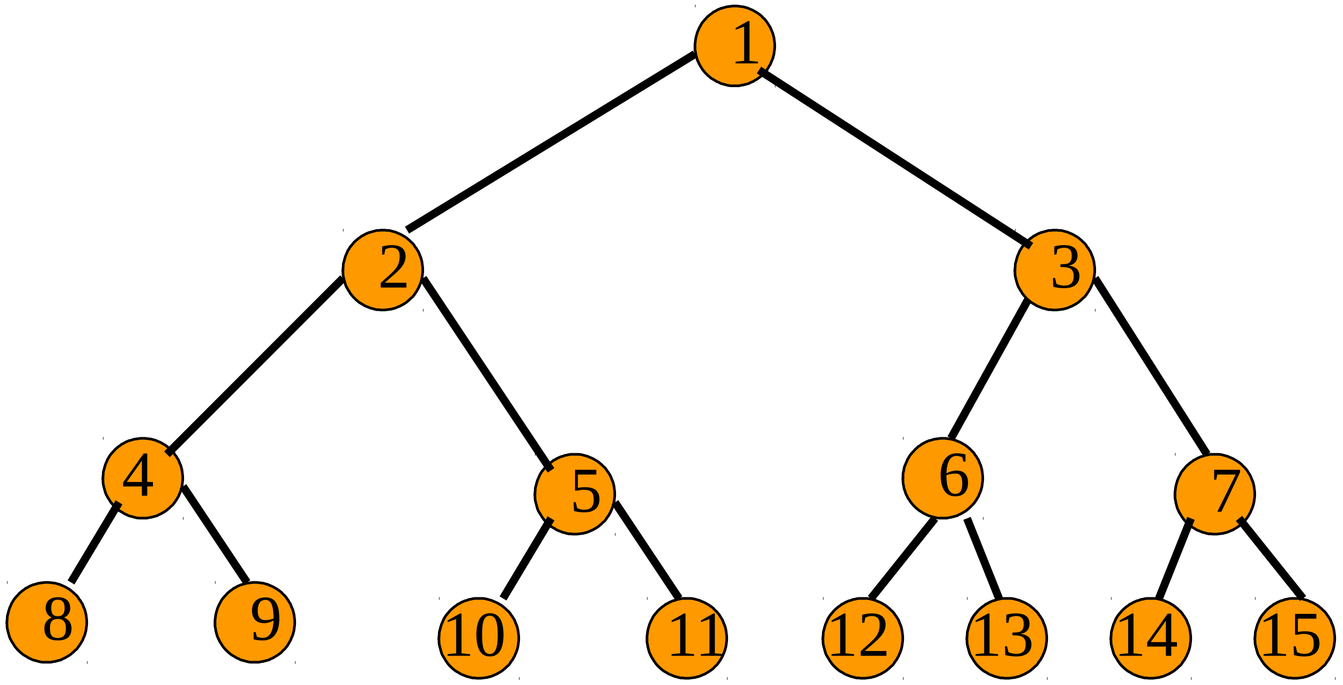
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

Node Number Properties



- Left child of node i is node 2i, unless 2i > n, where n is the number of nodes.
- If 2i > n, node i has no left child.

Node Number Properties

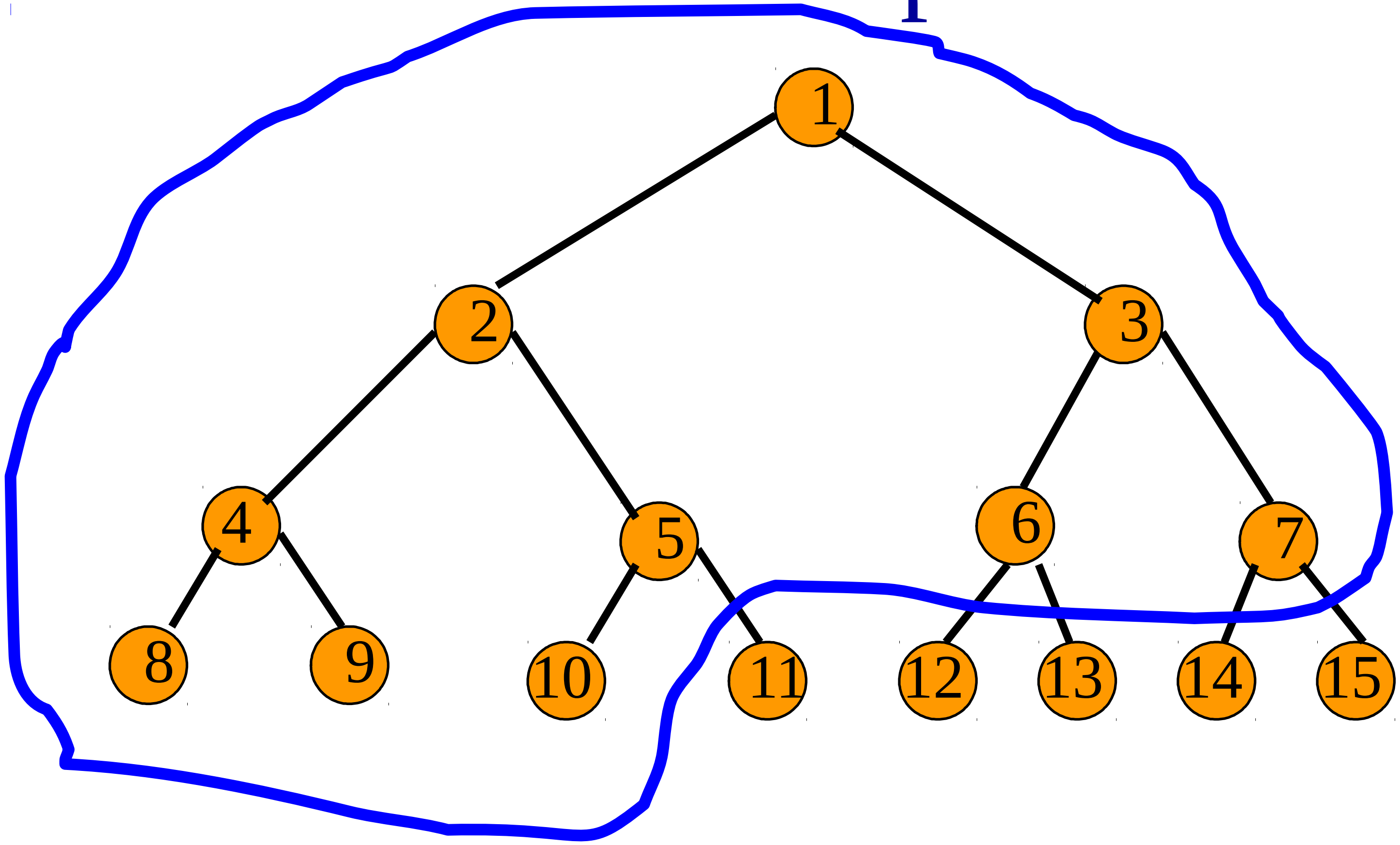


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i+1 > n, node i has no right child.

Complete Binary Tree With n Nodes

- Start with a full binary tree that has at least n nodes.
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree.

Example



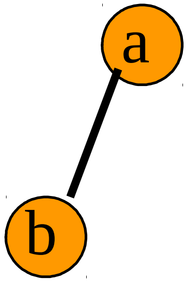
- Complete binary tree with 10 nodes.

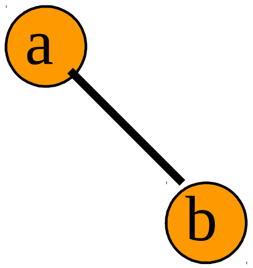
Differences Between A Tree & A Binary Tree

- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- A binary tree may be empty; a tree cannot be empty.

Differences Between A Tree & A Binary Tree

- The subtrees of a binary tree are ordered; those of a tree are not ordered.





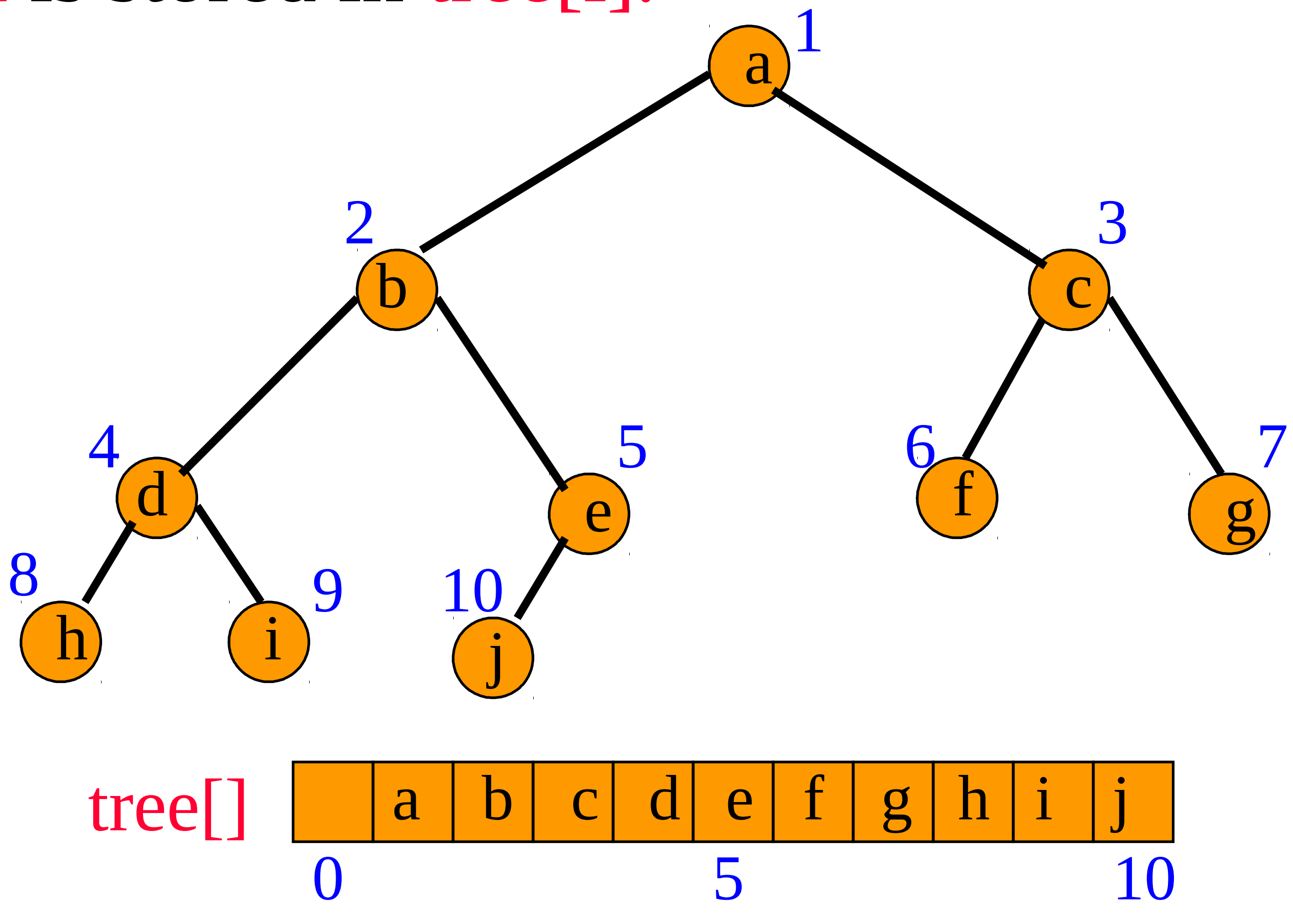
- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Binary Tree Representation

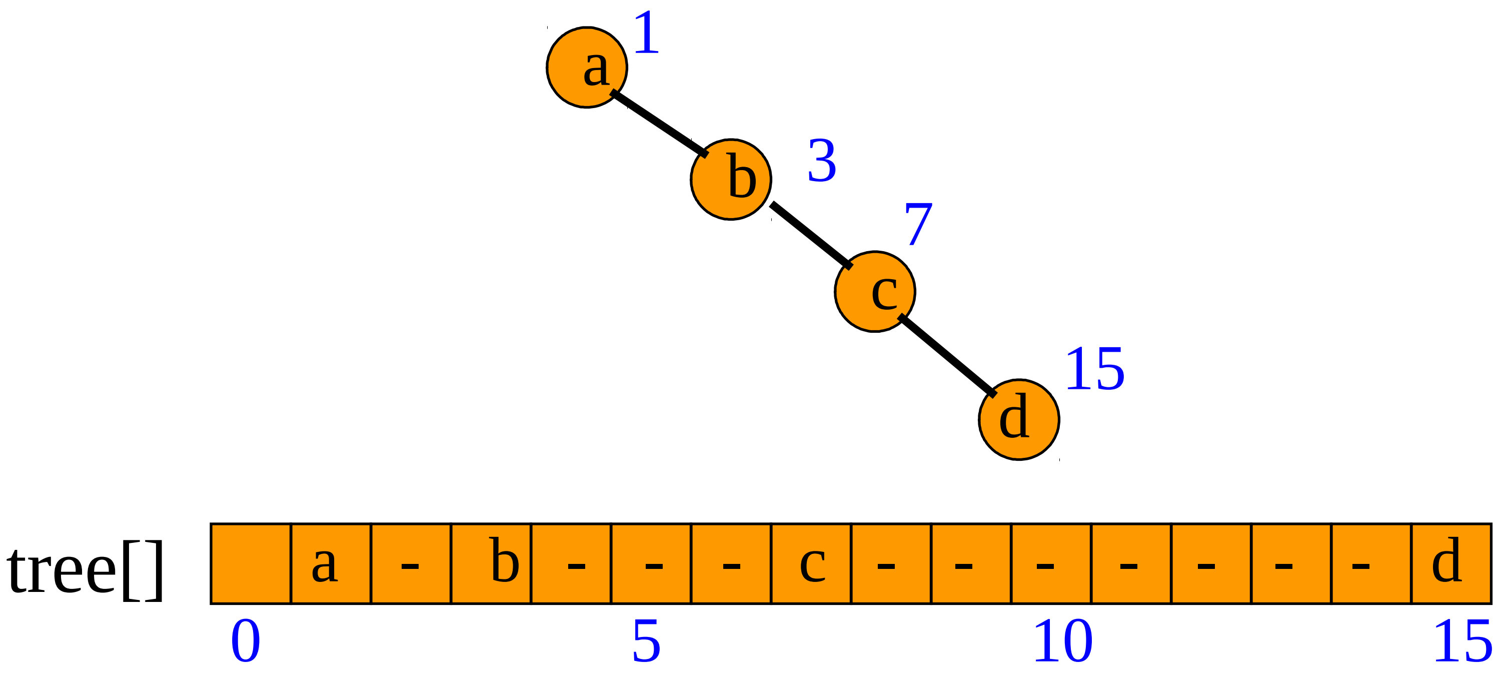
1. Sequential representation using arrays
2. List representation using Linked list

Array Representation

- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].

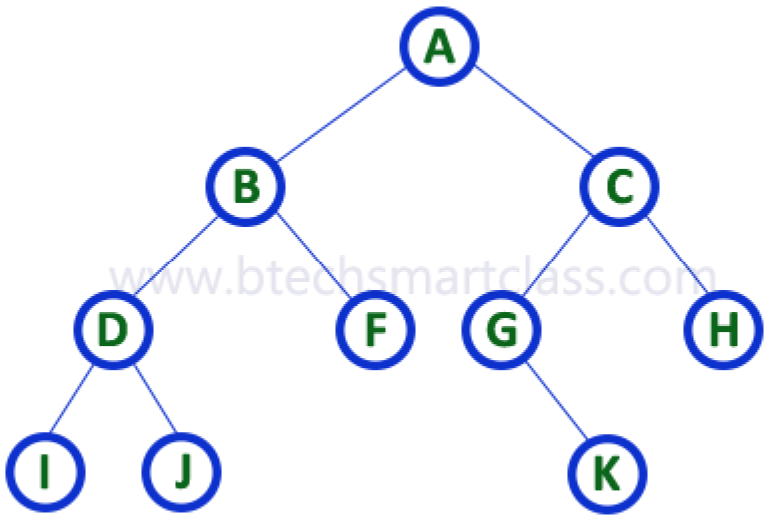


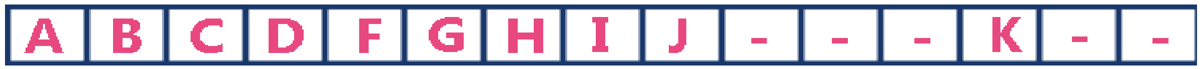
Right-Skewed Binary Tree



- An n node binary tree needs an array whose length is between n+1 and 2^n.

Sequential representation





A	B	C	D	F	G	H	I	J	-	-	-	K	-	-

- To represent a binary tree of depth **‘d'** using array representation, we need one dimensional array with a maximum size of $\mathbf{2^{d+1} - 1}$.

Sequential representation

- Advantages:
  - Direct access to all nodes (Random access)
- Disadvantages:
  - Height of tree should be known
  - Memory may be wasted
  - Insertion and deletion of a node is difficult

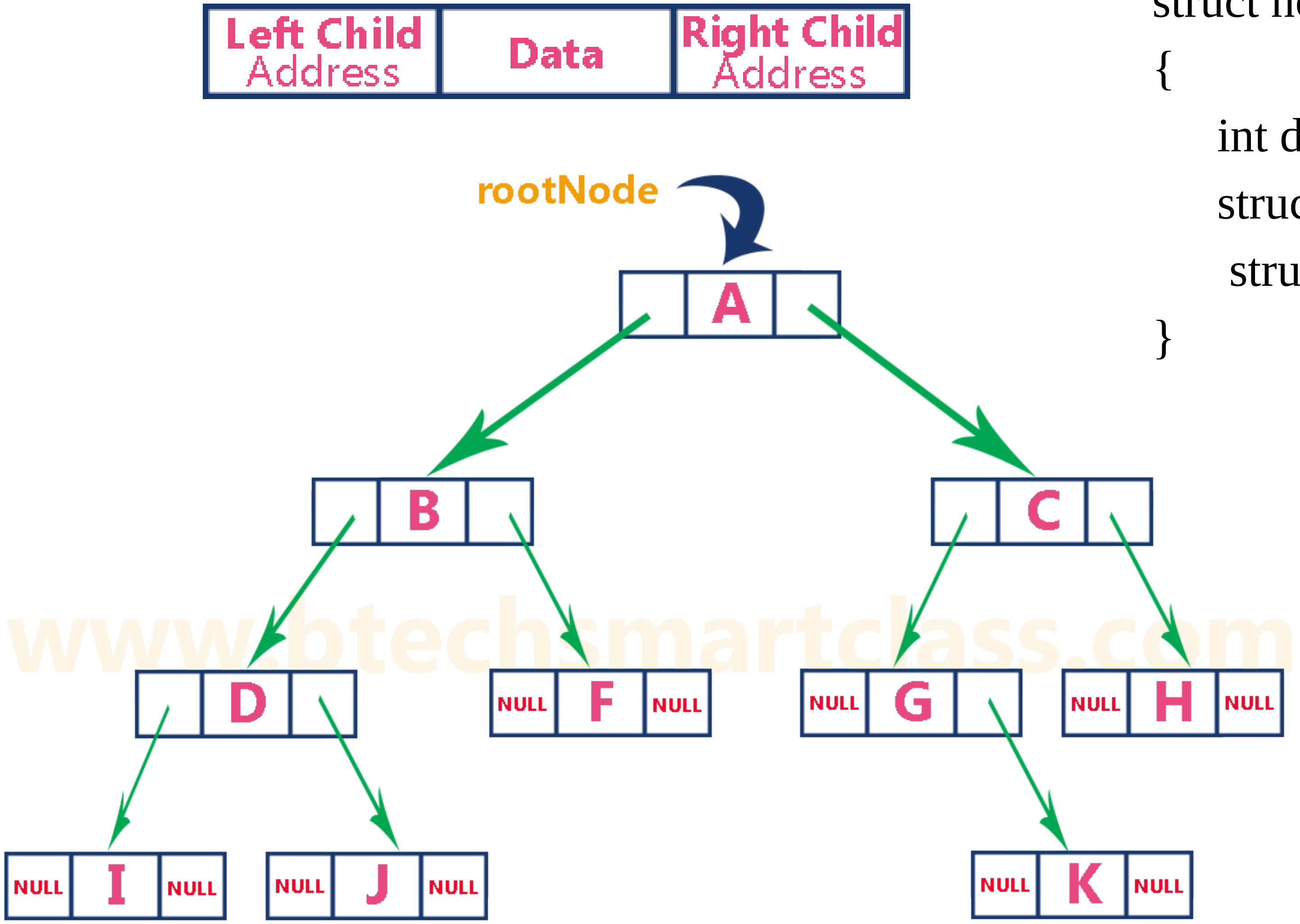
Linked Representation

- Each binary tree node is represented as an object whose data type is TreeNode.
- The space required by an n node binary tree is n * (space required by one node).

The Struct binaryTreeNode

```
template <class T>
class TreeNode
{
   T data;
   TreeNode<T> *leftChild,
               *rightChild;
   TreeNode()
      {leftChild = rightChild = NULL;}
   // other constructors come here
};
```

List representation



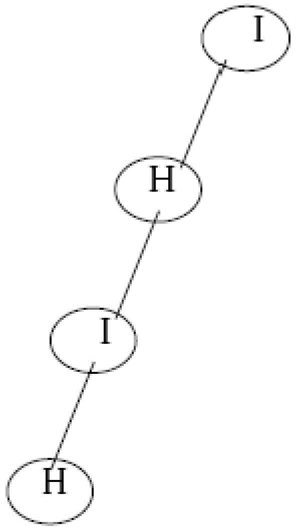
```
struct node
{
      int data;
      struct node *left;
       struct node *rifgt;
}
```

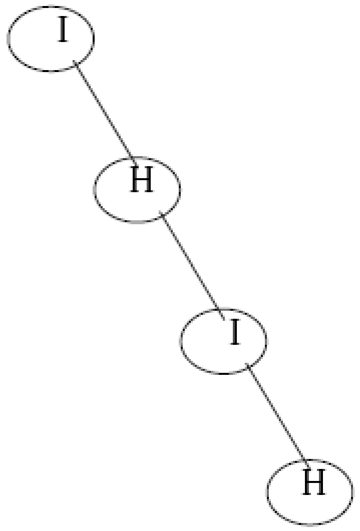
List representation

- Advantages:
  - Height of tree need not be known
  - No memory wastage
  - Insertion and deletion of a node is done without affecting other nodes
- Disadvantages:
  - Direct access to node is difficult
  - Additional memory required for storing address of left and right node

Left Skewed and Right Skewed Trees

- Binary tree has only left sub trees - Left Skewed Trees
- Binary tree has only right sub trees - Right Skewed Trees





Arithmetic Expressions

- (a + b) * (c + d) + e – f/g*h + 3.25
- Expressions comprise three kinds of entities.
  - Operators (+, -, /, *).
  - Operands (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.).
  - Delimiters ((,)).

Operator Degree

- Number of operands that the operator requires.
- Binary operator requires two operands.
  - a + b
  - c / d
  - e - f
- Unary operator requires one operand.
  - + g
  - - h

Infix Form

- Normal way to write an expression.
- Binary operators come in between their left and right operands.
  - a * b
  - a + b * c
  - a * b / c
  - (a + b) * (c + d) + e – f/g*h + 3.25

Operator Priorities

- How do you figure out the operands of an operator?
  - a + b * c
  - a * b + c / d
- This is done by assigning operator priorities.
  - priority(*) = priority(/) > priority(+) = priority(-)
- When an operand lies between two operators, the operand associates with the operator that has higher priority.

Tie Breaker

- When an operand lies between two operators that have the same priority, the operand associates with the operator on the left.
  - a + b - c
  - a * b / c / d

Delimiters

- Subexpression within delimiters is treated as a single operand, independent from the remainder of the expression.
  - (a + b) * (c – d) / (e – f)

Infix Expression Is Hard To Parse

- Need operator priorities, tie breaker, and delimiters.
- This makes computer evaluation more difficult than is necessary.
- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters.
- So it is easier for a computer to evaluate expressions that are in these forms.

Postfix Form

- The postfix form of a variable or constant is the same as its infix form.
  - a, b, 3.25
- The relative order of operands is the same in infix and postfix forms.
- Operators come immediately after the postfix form of their operands.
  - Infix = a + b
  - Postfix = ab+

Postfix Examples

- Infix = a + b * c
  - Postfix = a b c * +

- Infix = a * b + c
  - Postfix = a b * c +

- Infix = (a + b) * (c – d) / (e + f)
  - Postfix = a b + c d - * e f + /

Unary Operators

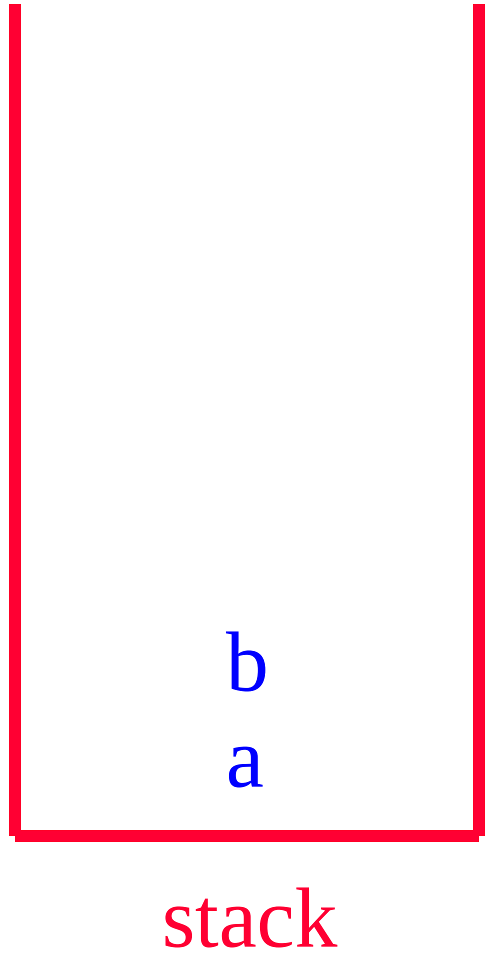
- Replace with new symbols.
  - + a => a @
  - + a + b => a @ b +
  - - a => a ?
  - - a-b => a ? b -

Postfix Evaluation

- Scan postfix expression from left to right pushing operands on to a stack.
- When an operator is encountered, pop as many operands as this operator needs; evaluate the operator; push the result on to the stack.
- This works because, in postfix, operators come immediately after their operands.

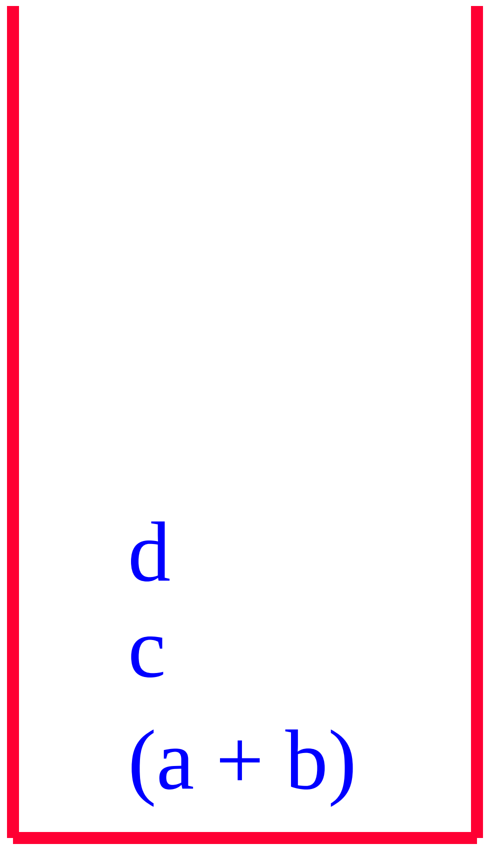
Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



Postfix Evaluation

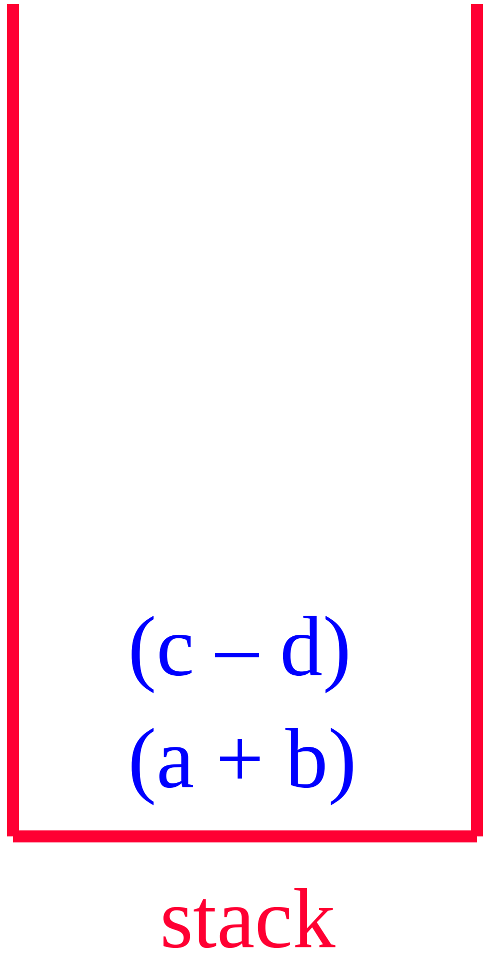
- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



stack

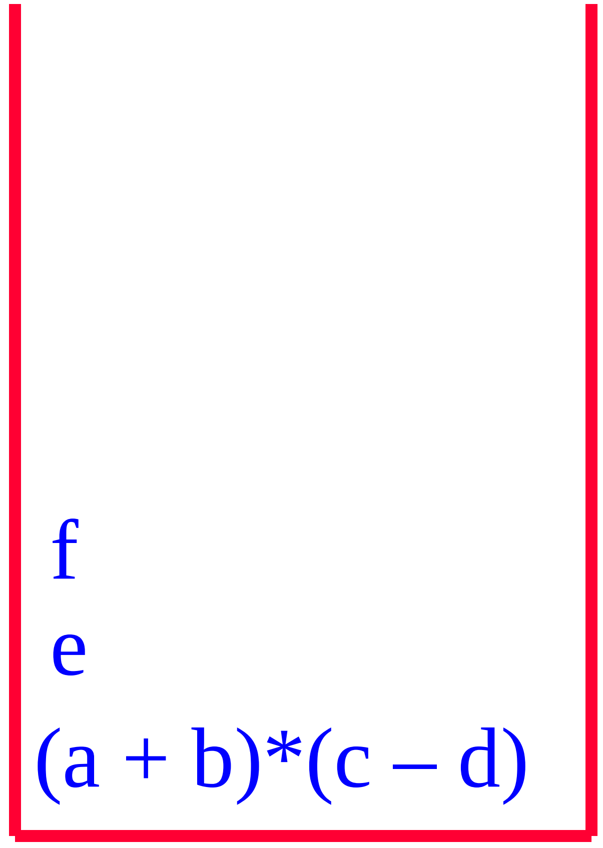
Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /



Postfix Evaluation

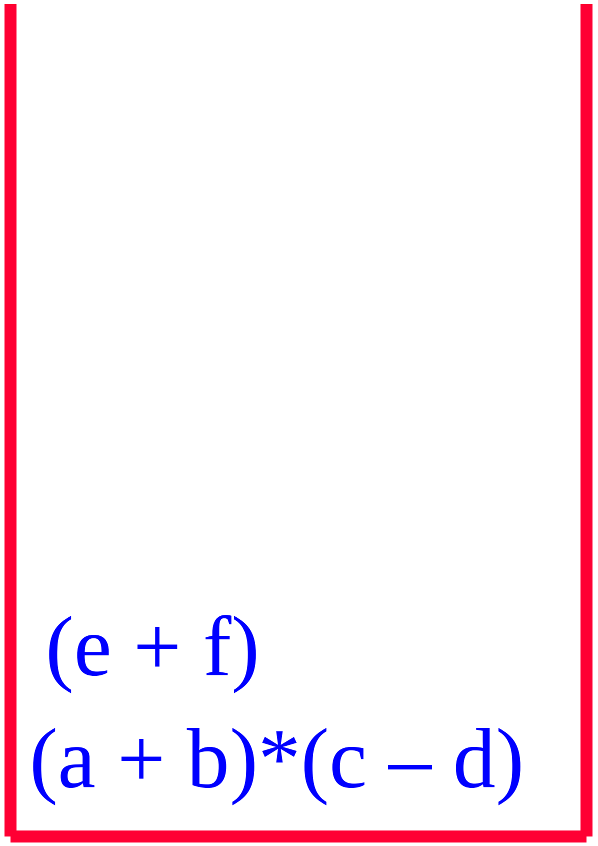
- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



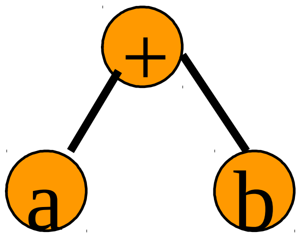
stack

Prefix Form

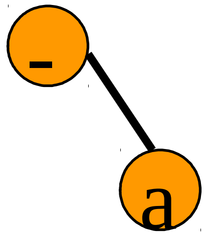
- The prefix form of a variable or constant is the same as its infix form.
  - a, b, 3.25
- The relative order of operands is the same in infix and prefix forms.
- Operators come immediately before the prefix form of their operands.
  - Infix = a + b
  - Postfix = ab+
  - Prefix = +ab

Binary Tree Form

- a + b

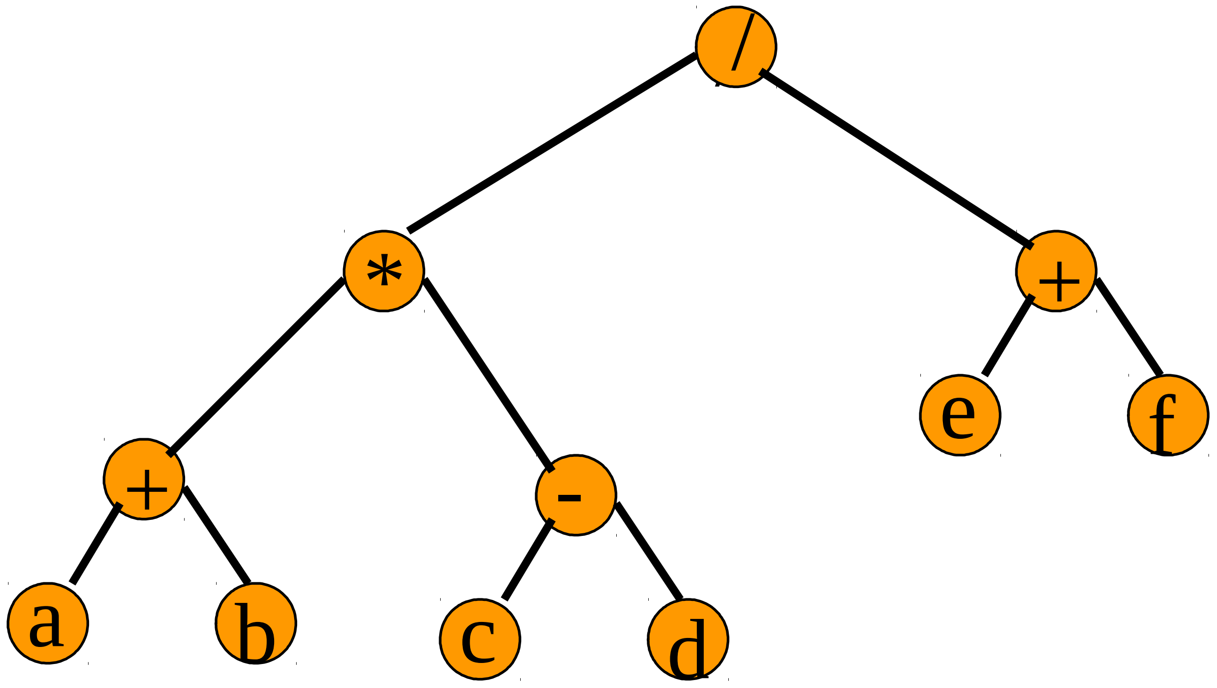


- - a



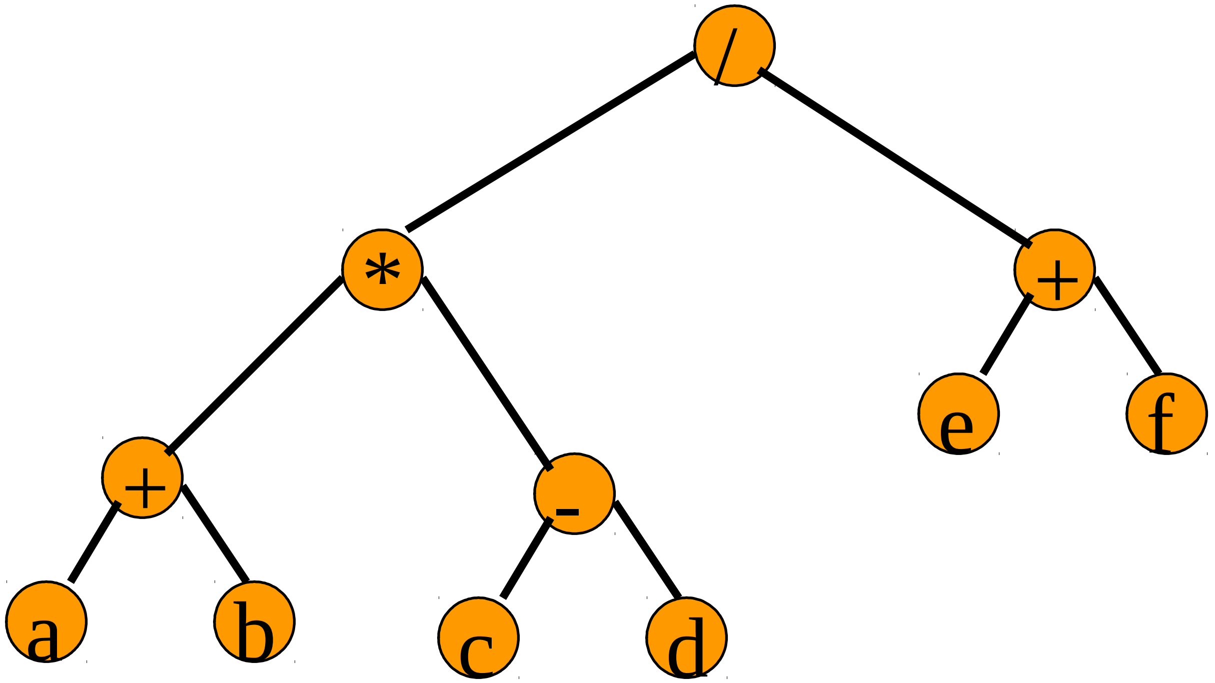
Binary Tree Form

- (a + b) * (c – d) / (e + f)



Merits Of Binary Tree Form

- Left and right operands are easy to visualize.
- Code optimization algorithms work with the binary tree form of an expression.
- Simple recursive evaluation of expression.



Properties of Binary Trees

Property 11.1 *The drawing of every binary tree with n elements, $n > 0$, has exactly $n-1$ edges.*

Proof Every element in a binary tree (except the root) has exactly one parent. There is exactly one edge between each child and its parent. So the number of edges is $n-1$. ■

Property 11.2 *A binary tree of height h, $h \geq 0$, has at least h and at most $2^h - 1$ elements in it.*

Proof Since each level has at least one element, the number of elements is at least h. As each element can have at most two children, the number of elements at level i is at most 2^{i-1}, $i > 0$. For $h = 0$, the total number of elements is 0, which equals $2^0 - 1$. For $h > 0$, the number of elements cannot exceed $\sum_{i=1}^{h} 2^{i-1} = 2^h - 1$. ■

Property 11.3 *The height of a binary tree that contains n, $n \geq 0$, elements is at most n and at least $\lceil \log_2(n+1) \rceil$.*

Proof Since there must be at least one element at each level, the height cannot exceed n. From Property 11.2 we know that a binary tree of height h can have no more than $2^h - 1$ elements. So $n \leq 2^h - 1$. Hence $h \geq \log_2(n+1)$. Since h is an integer, we get $h \geq \lceil \log_2(n+1) \rceil$. ■

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree.
- In a traversal, each element of the binary tree is visited exactly once.
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Traversing the Tree

- Visiting each node in a specified order.
- Three simple ways to traverse a tree:
  - Inorder
  - Preorder
  - Postorder

Inorder Traversing

Inorder traversal will cause all the nodes to be visited in ascending order.

- Steps involved in Inorder traversal (recursion) are:

1. -- Call itself to traverse the node's left subtree
2. -- Visit the node (e.g. display a key)
3. -- Call itself to traverse the node's right subtree.

```
inOrder( node lroot)
{
    If (lroot != null) {
        inOrder(lroot.leftChild());
        System.out.print(lroot.iData + “  “);
         inOrder(lroot.rightChild());
}
```

Tree Traversal (continued)

- Sequence of preorder traversal: e.g. use for infix parse tree to generate prefix

  -- Visit the node

  -- Call itself to traverse the node’s left subtree

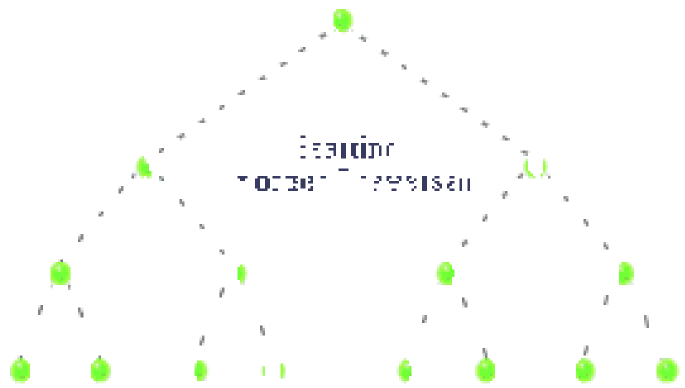
  -- Call itself to traverse the node’s right subtree
- Sequence of postorder traversal: e.g. use for infix parse tree to generate postfix

  -- Call itself to traverse the node’s left subtree

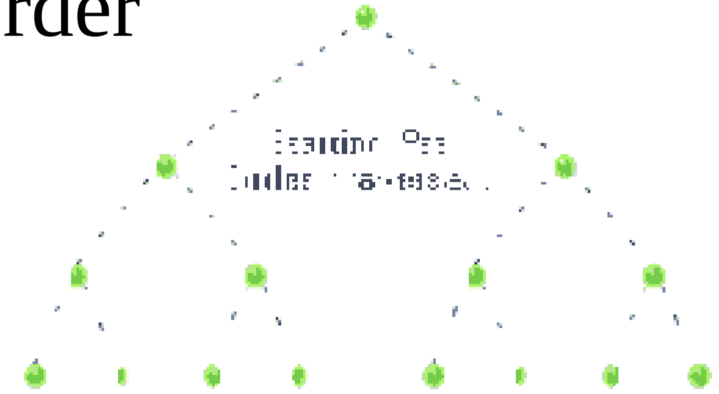
  -- Call itself to traverse the node’s right subtree

  -- Visit the node.

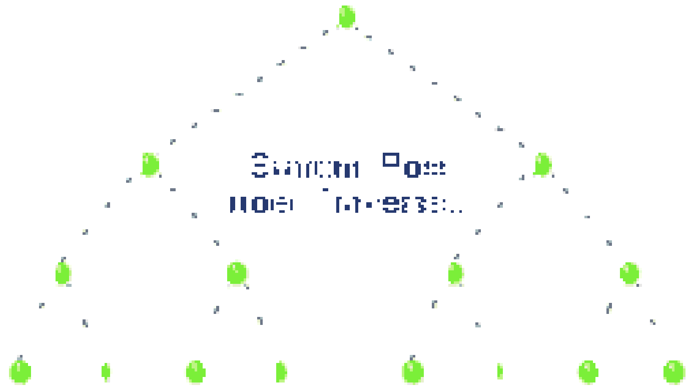
- Inorder



- Preorder



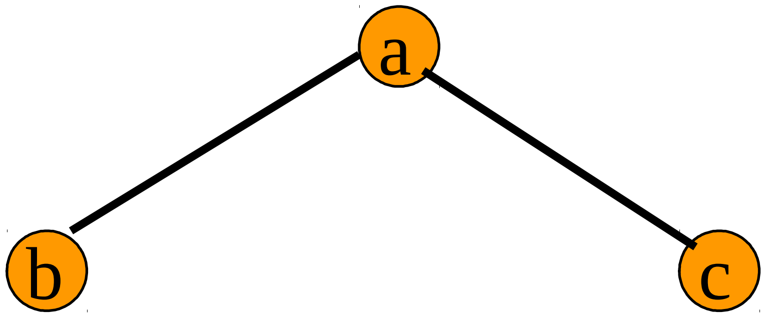
- Postorder



Preorder Traversal

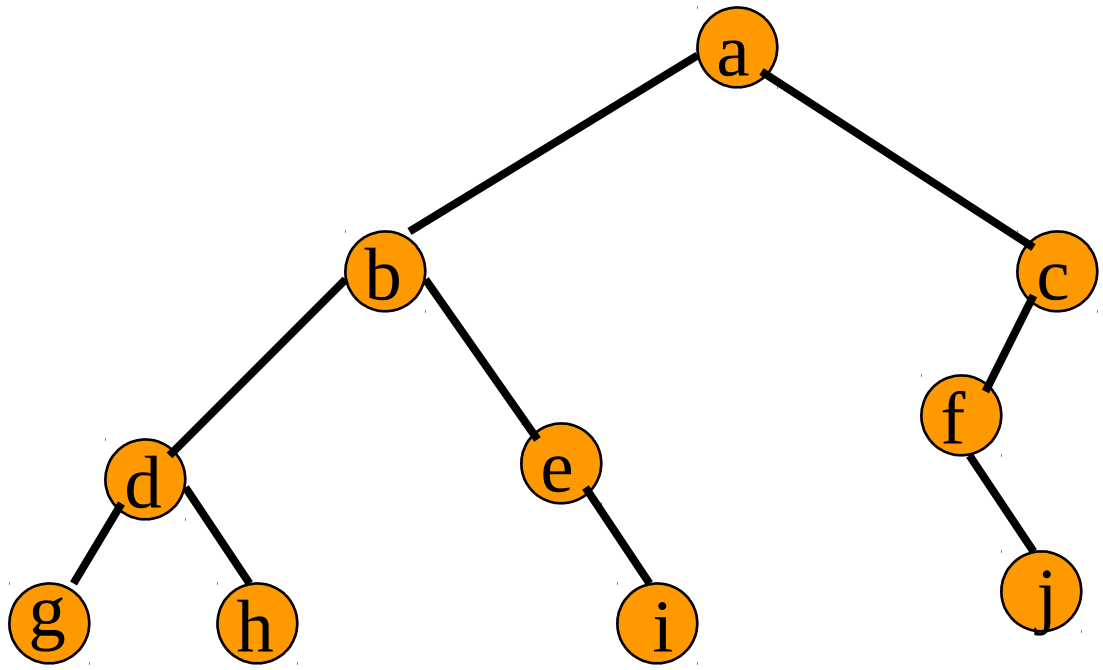
```
template <class T>
void PreOrder(TreeNode<T> *t)
{
   if (t != NULL)
   {
      Visit(t);
      PreOrder(t->leftChild);
      PreOrder(t->rightChild);
   }
}
```

Preorder Example (Visit = print)



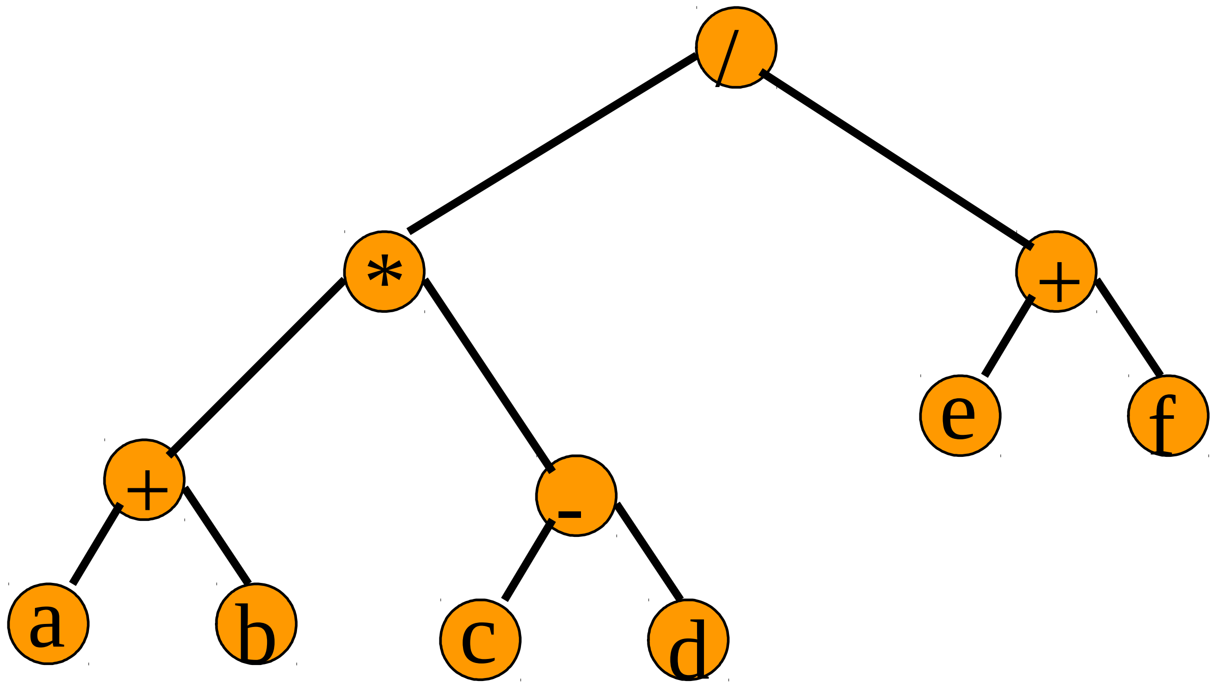
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



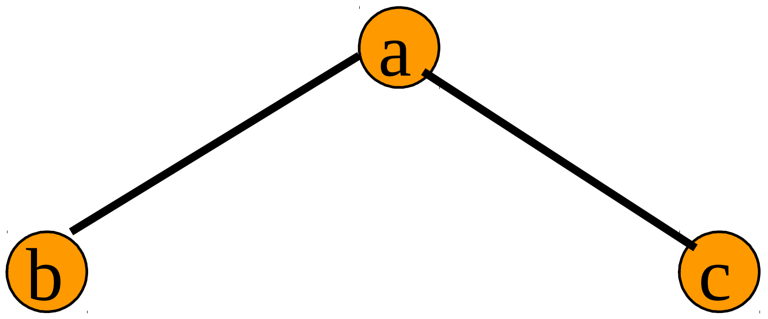
/ * + a b - c d + e f

Gives prefix form of expression!

Inorder Traversal

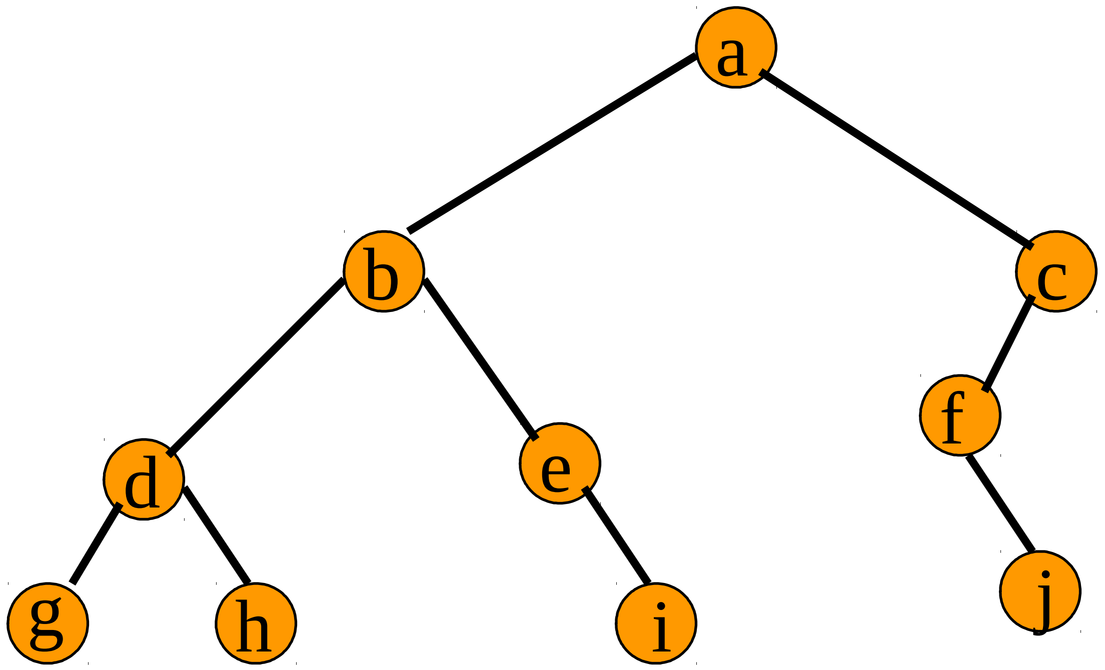
```
template <class T>
void InOrder(TreeNode<T> *t)
{
   if (t != NULL)
   {
      InOrder(t->leftChild);
      Visit(t);
      InOrder(t->rightChild);
   }
}
```

Inorder Example (Visit = print)



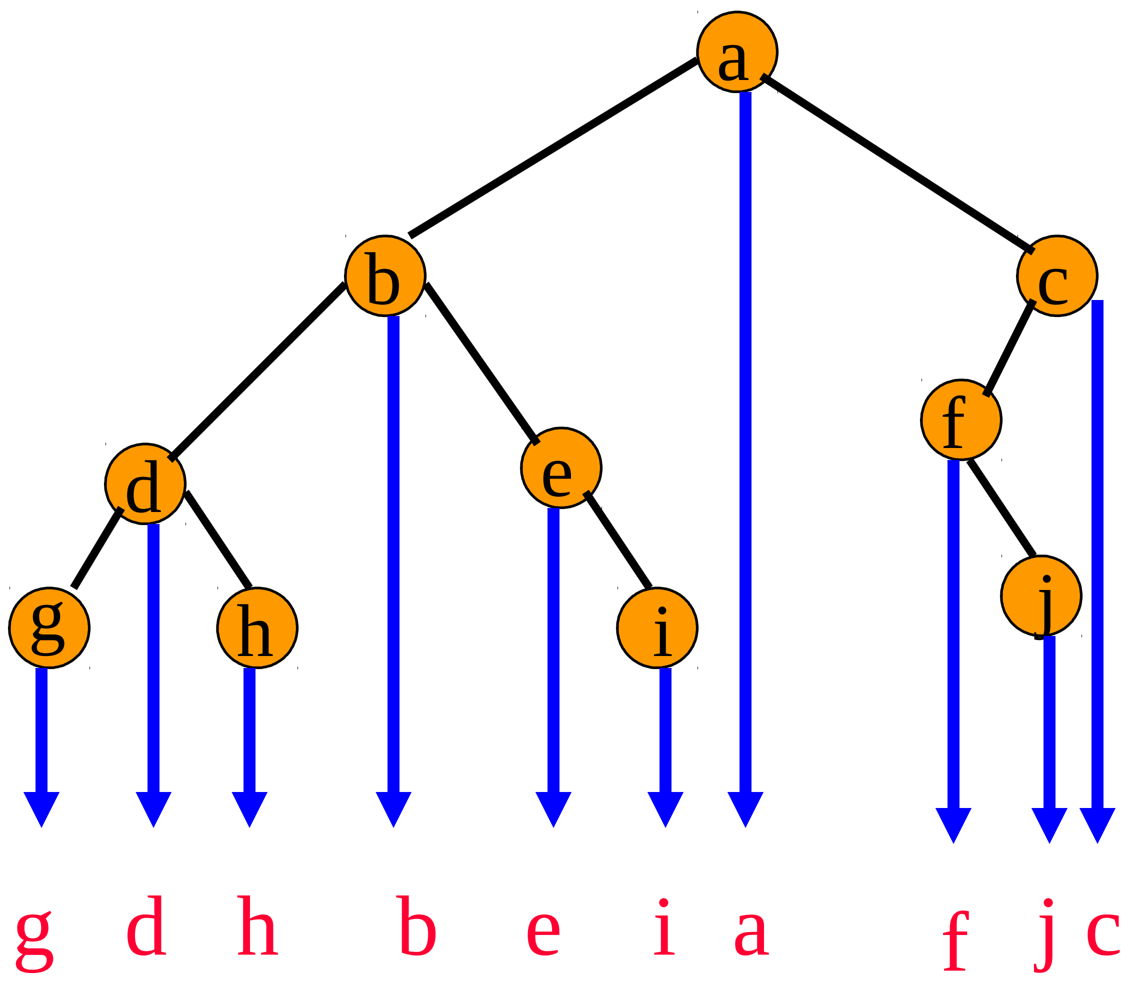
b a c

Inorder Example (Visit = print)

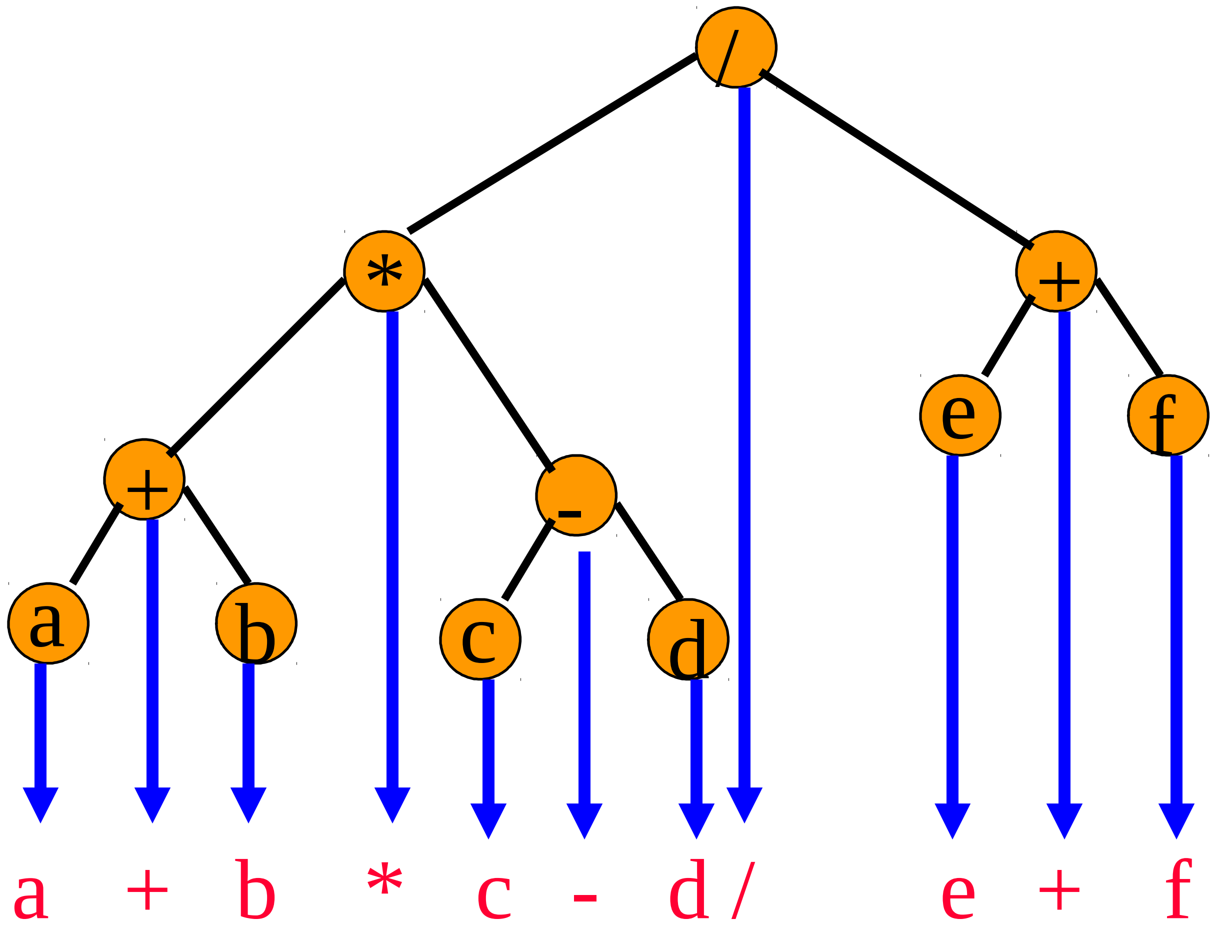


g d h b e i a f j c

Inorder By Projection (Squishing)



Inorder Of Expression Tree

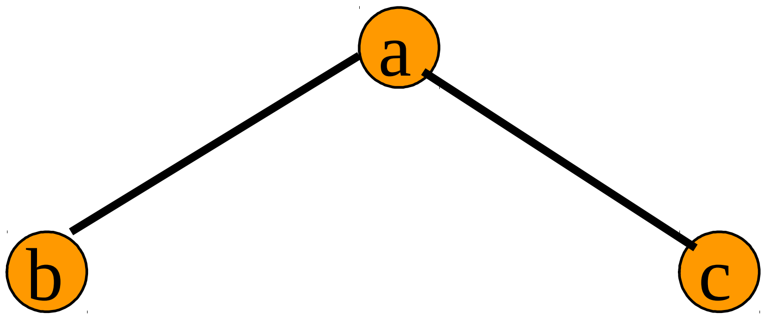


Gives infix form of expression

Postorder Traversal

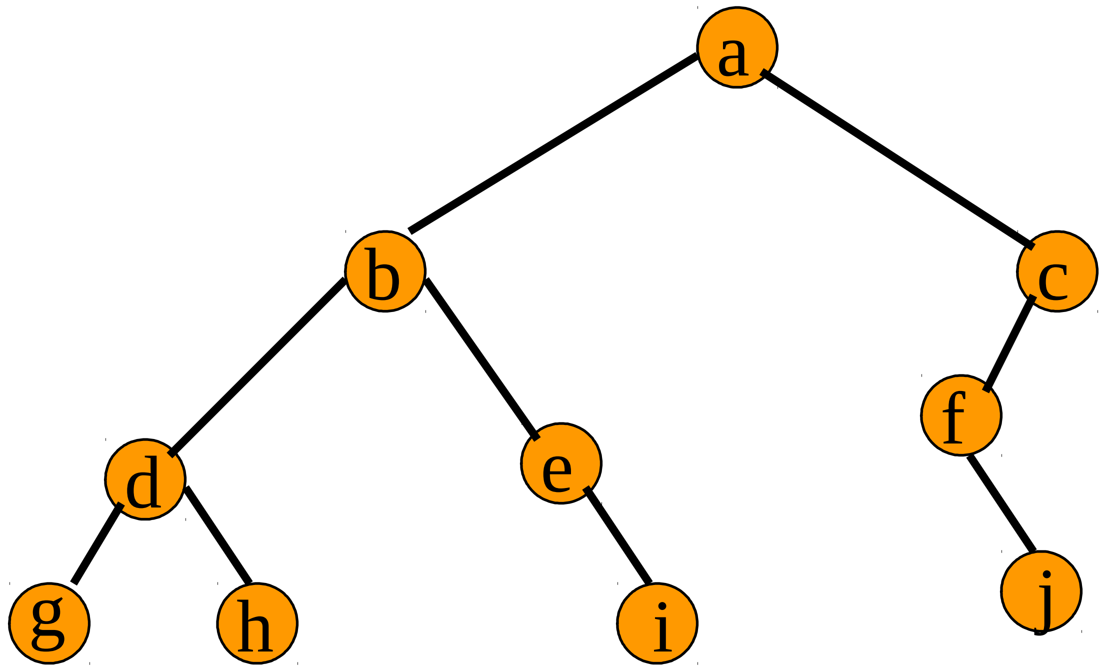
```
template <class T>
void PostOrder(TreeNode<T> *t)
{
   if (t != NULL)
   {
      PostOrder(t->leftChild);
      PostOrder(t->rightChild);
      Visit(t);
   }
}
```

Postorder Example (Visit = print)



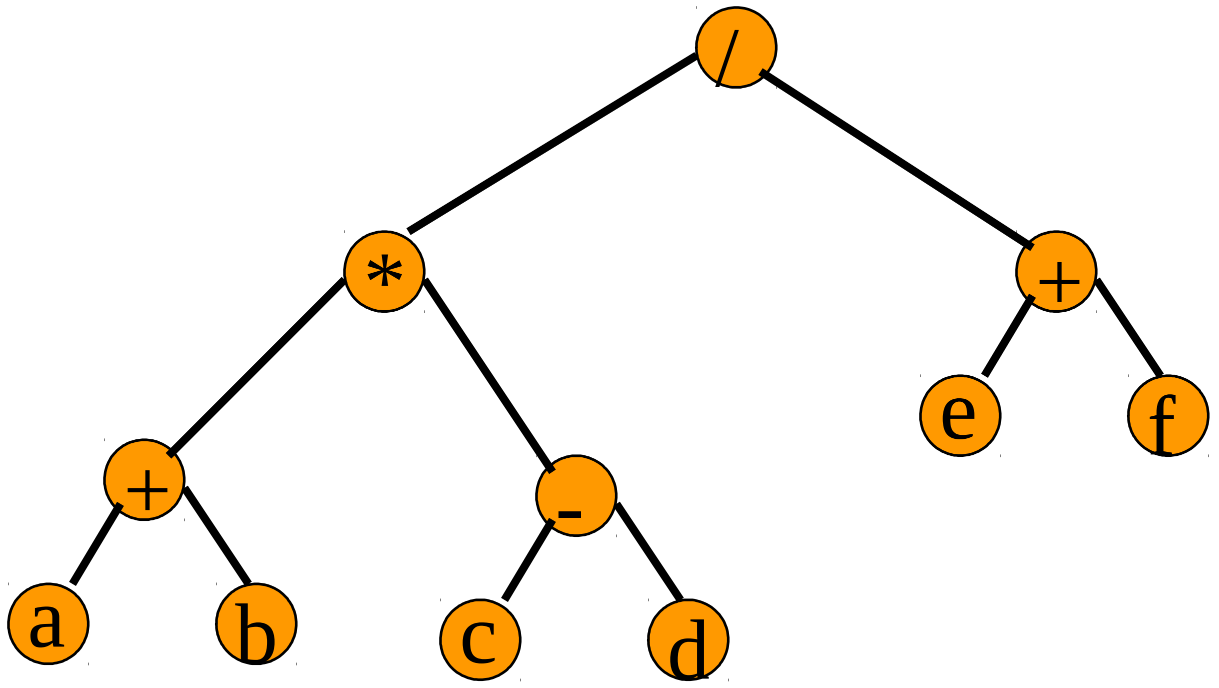
b c a

Postorder Example (Visit = print)



g h d i e b j f c a

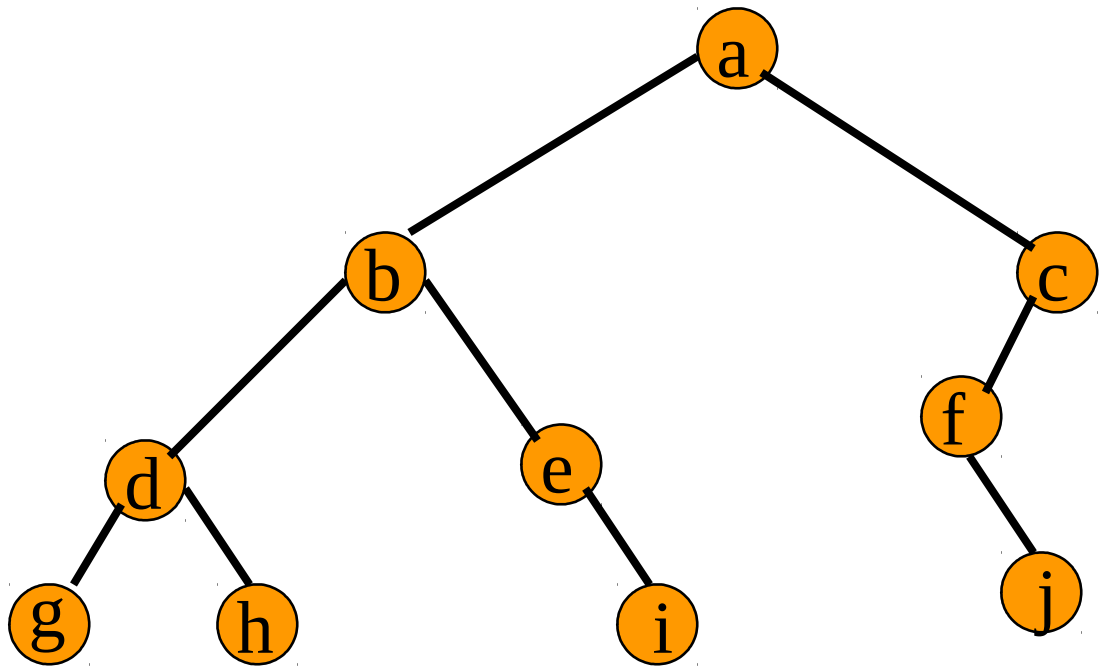
Postorder Of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression!

Traversal Applications

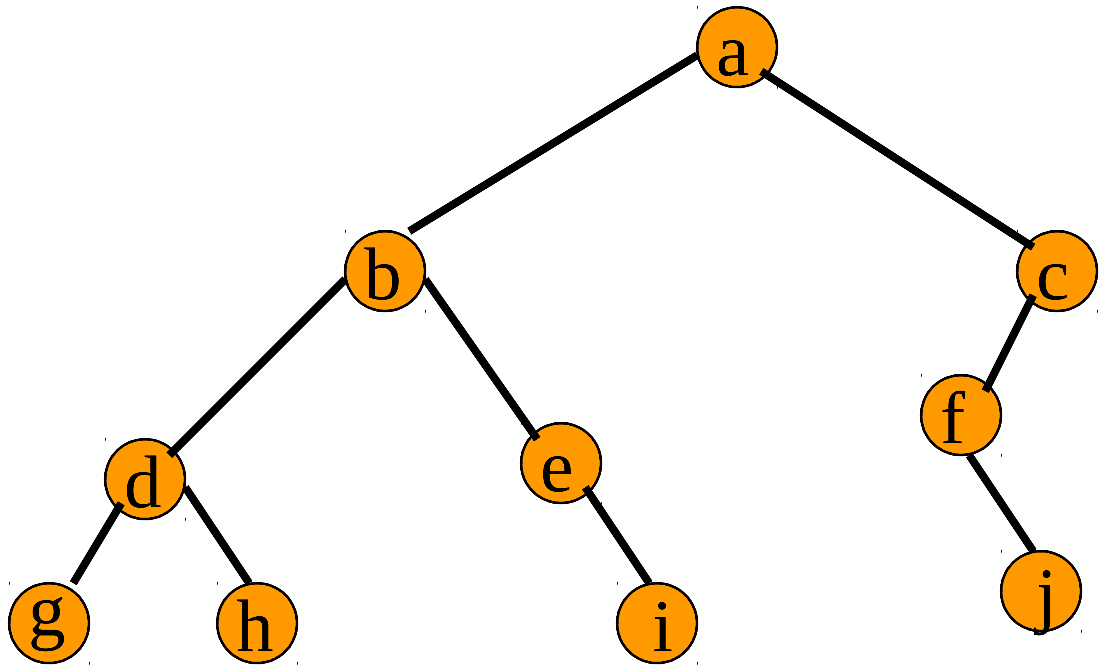


- Make a clone.
- Determine height.
- Determine number of nodes.

Level Order

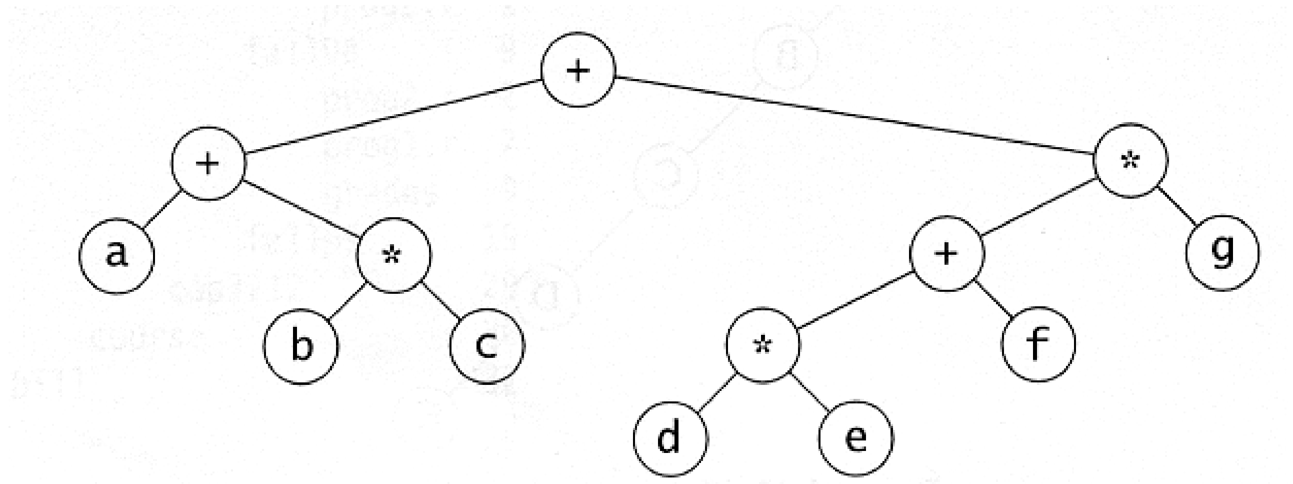
```
Let t be the tree root.
while (t != NULL)
{
    visit t and put its children on a FIFO queue;
    if FIFO queue is empty, set t = NULL;
    otherwise, pop a node from the FIFO queue
   and call it t;
}
```

Level-Order Example (Visit = print)



a b c d e f g h i j

prefix expression ++a*bc*+*defg
postfix expression abc*+de*f+g*+
infix expression a+b*c+d*e+f*g

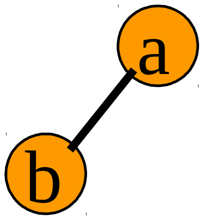


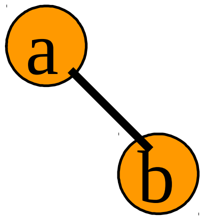
Expression tree for (a + b * c) + ((d * e + f) * g)

Some Examples

preorder

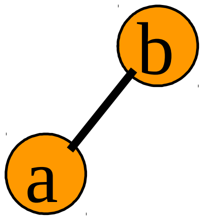
= ab

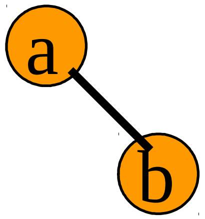




inorder

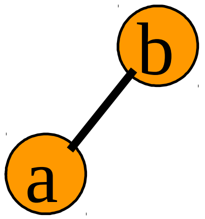
= ab

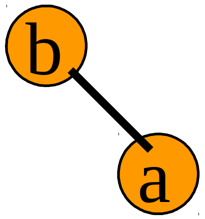




postorder

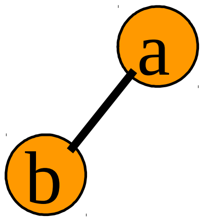
= ab

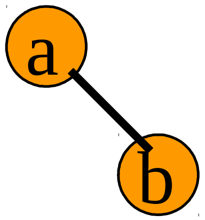




level order

= ab

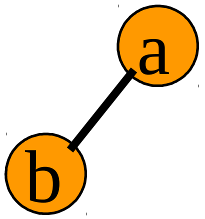


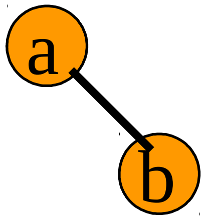


Preorder And Postorder

preorder = ab

postorder = ba

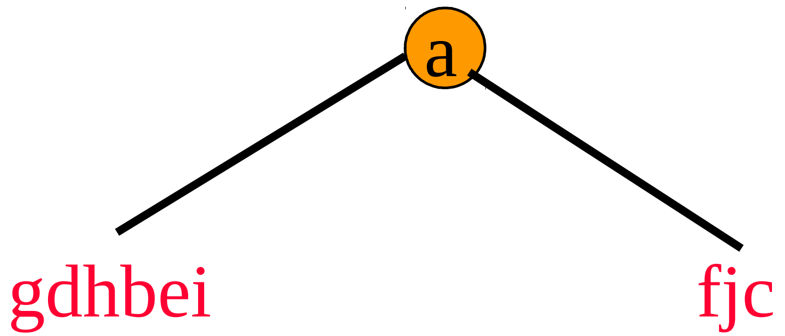




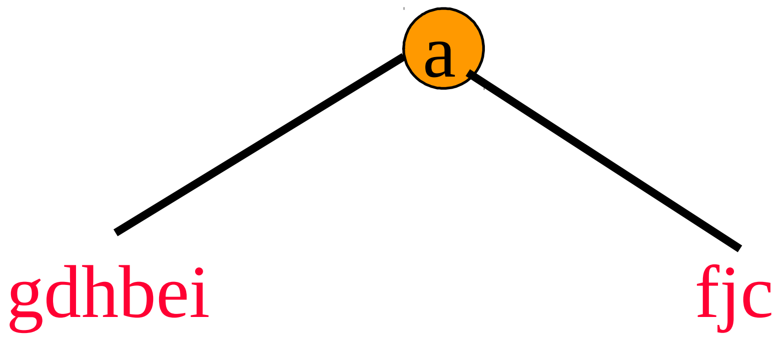
- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order.
- Nor do postorder and level order.

Inorder And Preorder

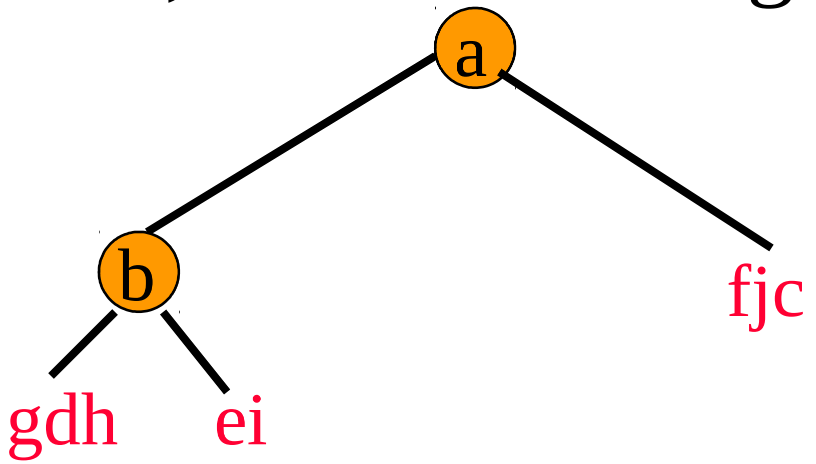
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



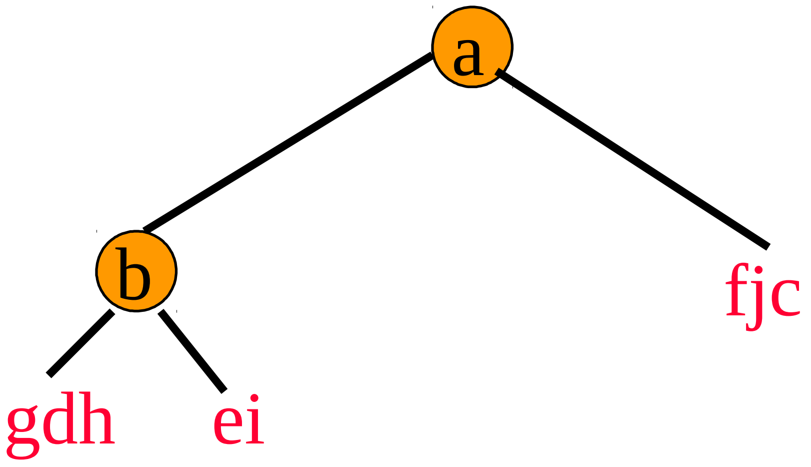
Inorder And Preorder



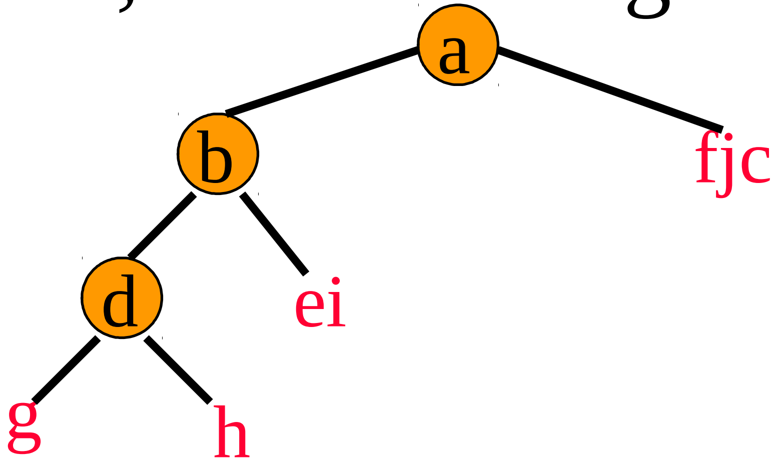
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder



- preorder = a b d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.



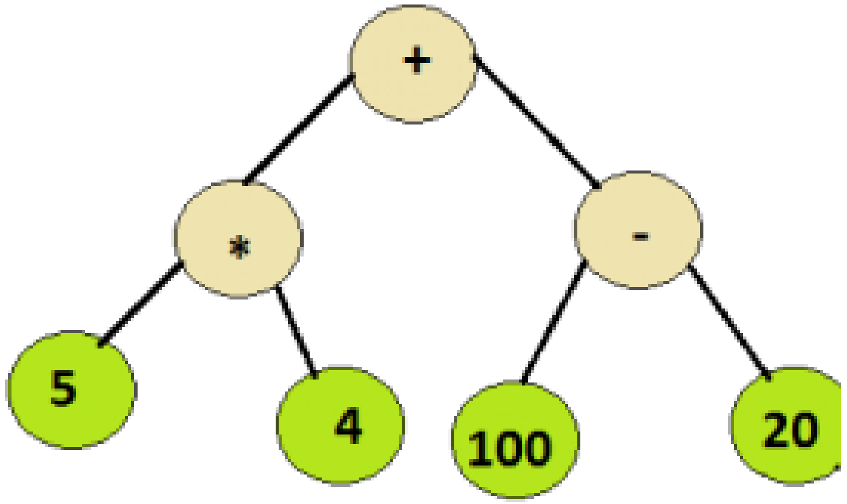
Inorder And Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

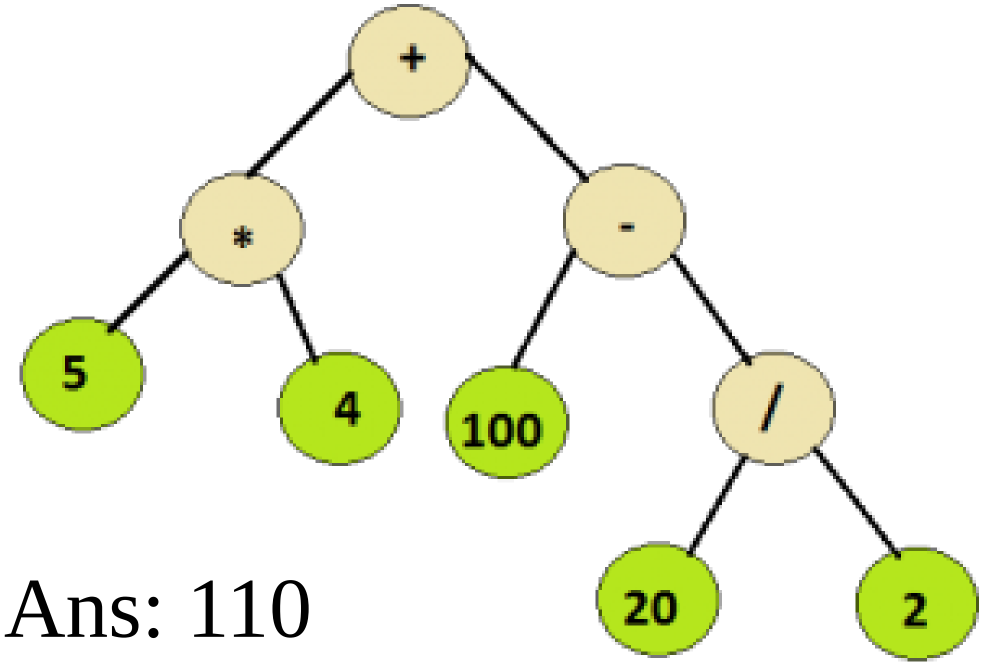
Inorder And Level Order

- Scan level order from left to right using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

xpression



Ans: 100



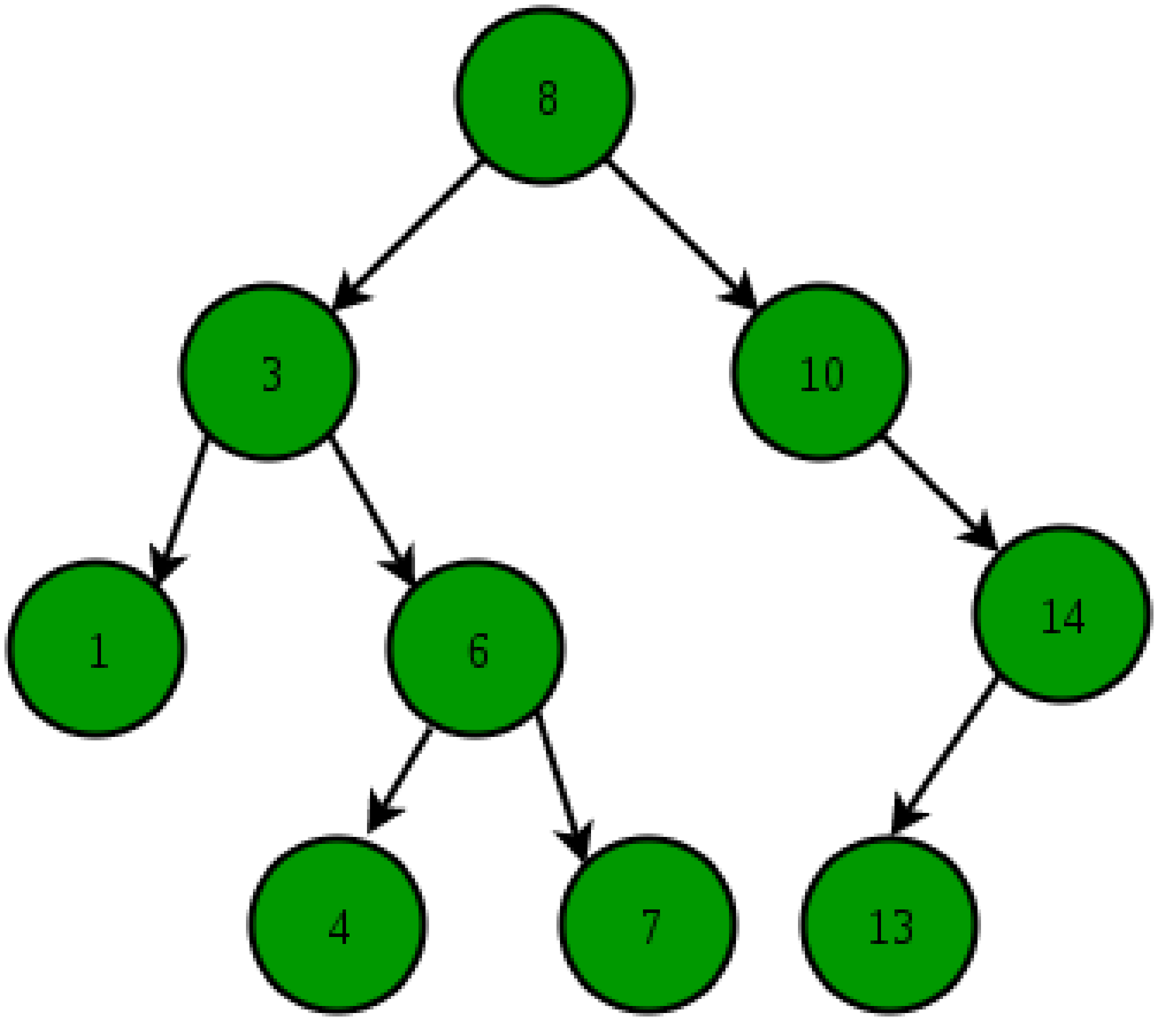
Ans: 110

Binary Search Tree

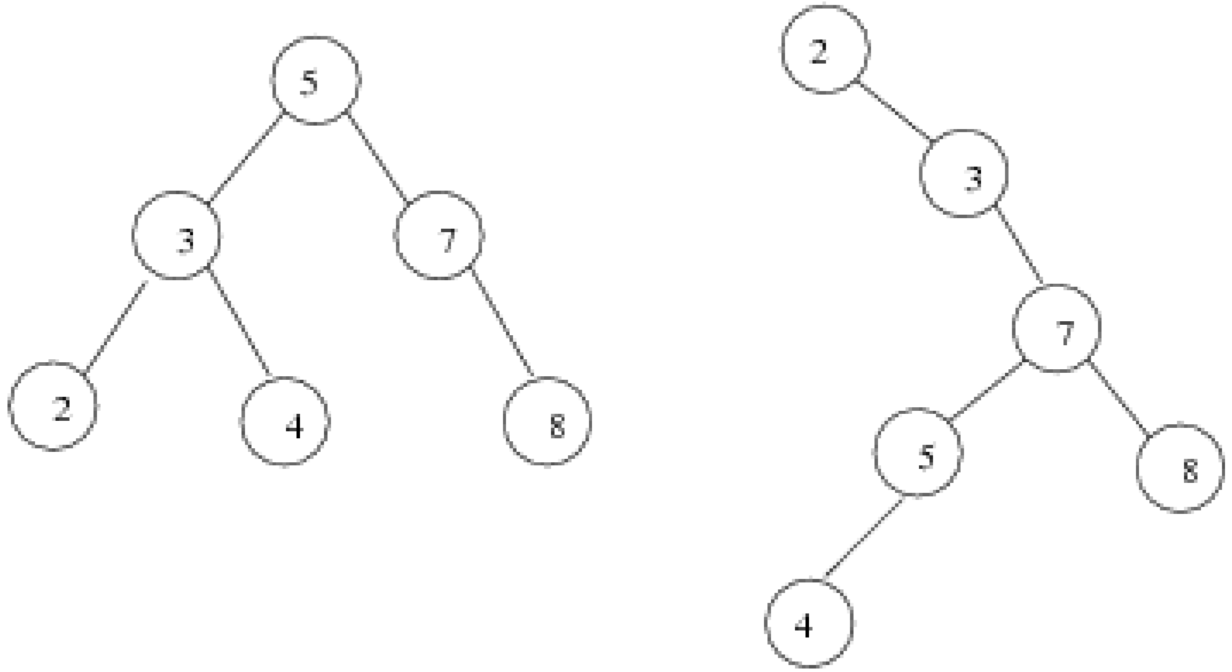
Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- The left and right subtree each must also be a binary search tree.

Binary Search Tree



Two binary search trees representing the s



- Average depth of a node is O(log N); maximum depth of a node is O(N)

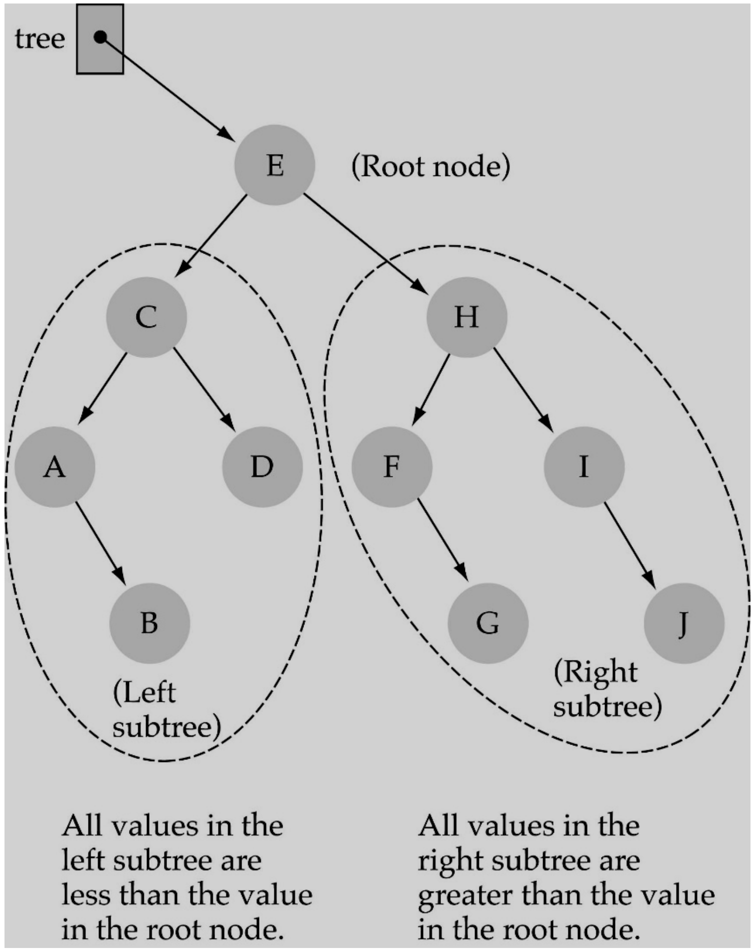
Some Binary Tree Operations

- Determine the height.
- Determine the number of nodes.
- Make a clone.
- Determine if two binary trees are clones.
- Display the binary tree.
- Evaluate the arithmetic expression represented by a binary tree.
- Obtain the infix form of an expression.
- Obtain the prefix form of an expression.
- Obtain the postfix form of an expression.

Finding a Node

- To find a node given its key value, start from the root.
- If the key value is same as the node, then node is found.
- If key is greater than node, search the right subtree, else search the left subtree.
- Continue till the node is found or the entire tree is traversed.
- Time required to find a node depends on how many levels down it is situated, i.e. O(log N).

How to search a binary search tree?

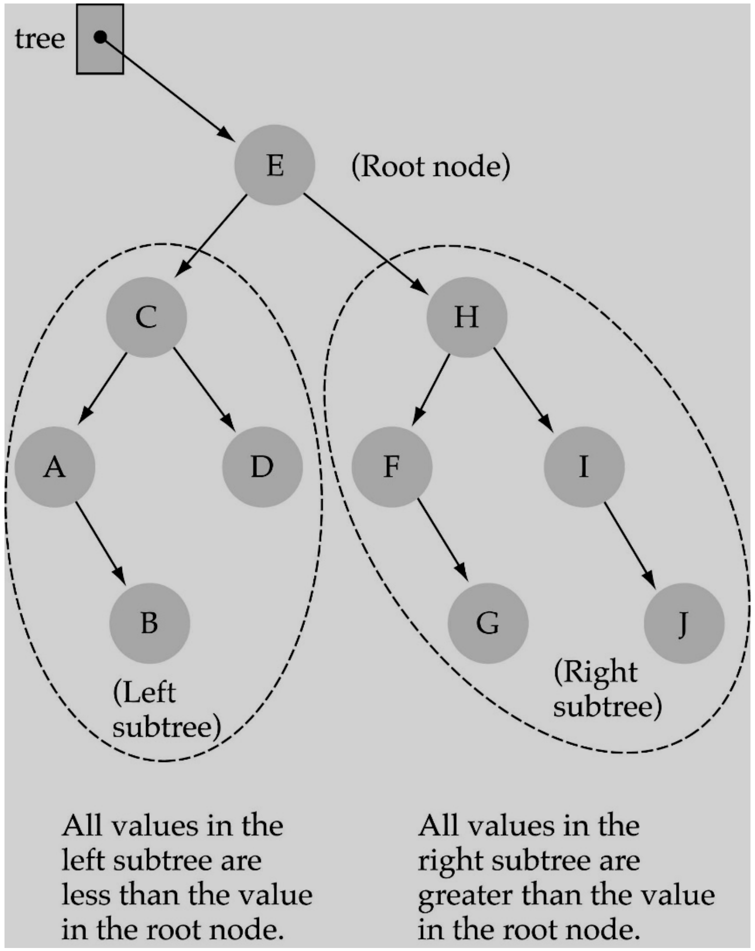

tree
E
(Root node)
C
H
A
D
F
I
B
G
J
(Left subtree)
(Right subtree)
All values in the left subtree are less than the value in the root node.
All values in the right subtree are greater than the value in the root node.

(1) Start at the root

(2) Compare the value of the item you are searching for with the value stored at the root

(3) If the values are equal, then *item found*; otherwise, if it is a leaf node, then *not found*

How to search a binary search tree?


tree
E
(Root node)
C
H
A
D
F
I
B
G
J
(Left subtree)
(Right subtree)
All values in the left subtree are less than the value in the root node.
All values in the right subtree are greater than the value in the root node.

(4) If it is less than the value stored at the root, then search the left subtree

(5) If it is greater than the value stored at the root, then search the right subtree

(6) Repeat steps 2-6 for the root of the subtree chosen in the previous step 4 or 5

Inserting a Node

- To insert a node we must first find the place to insert it.
- Follow the path from the root to the appropriate node, which will be the parent of the new node.
- When this parent is found, the new node is connected as its left or right child, depending on whether the new node’s key is less or greater than that of the parent.

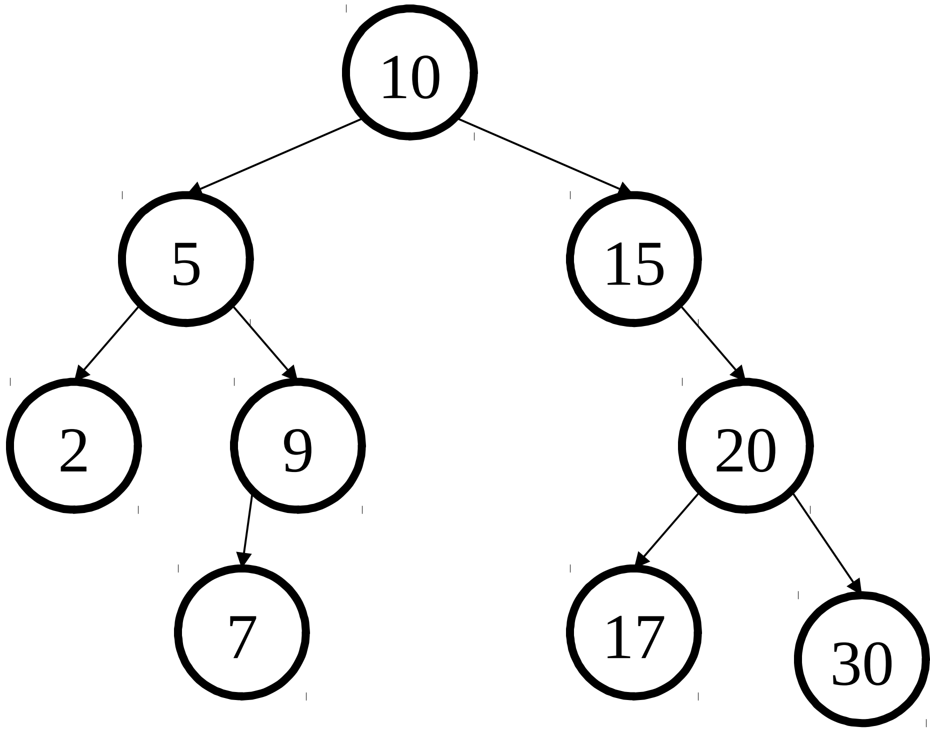
Finding Maximum and Minimum Values

- For the minimum,
  - go to the left child of the root and keep going to the left child until you come to a leaf node. This node is the minimum.
- For the maximum,
  - go to the right child of the root and keep going to the right child until you come to a leaf node. This node is the maximum.

FindMin, FindMax

- Find minimum

- Find maximum



Deleting a Node

- Start by finding the node you want to delete.
- Then there are three cases to consider:
  1. The node to be deleted is a leaf
  2. The node to be deleted has one child
  3. The node to be deleted has two children

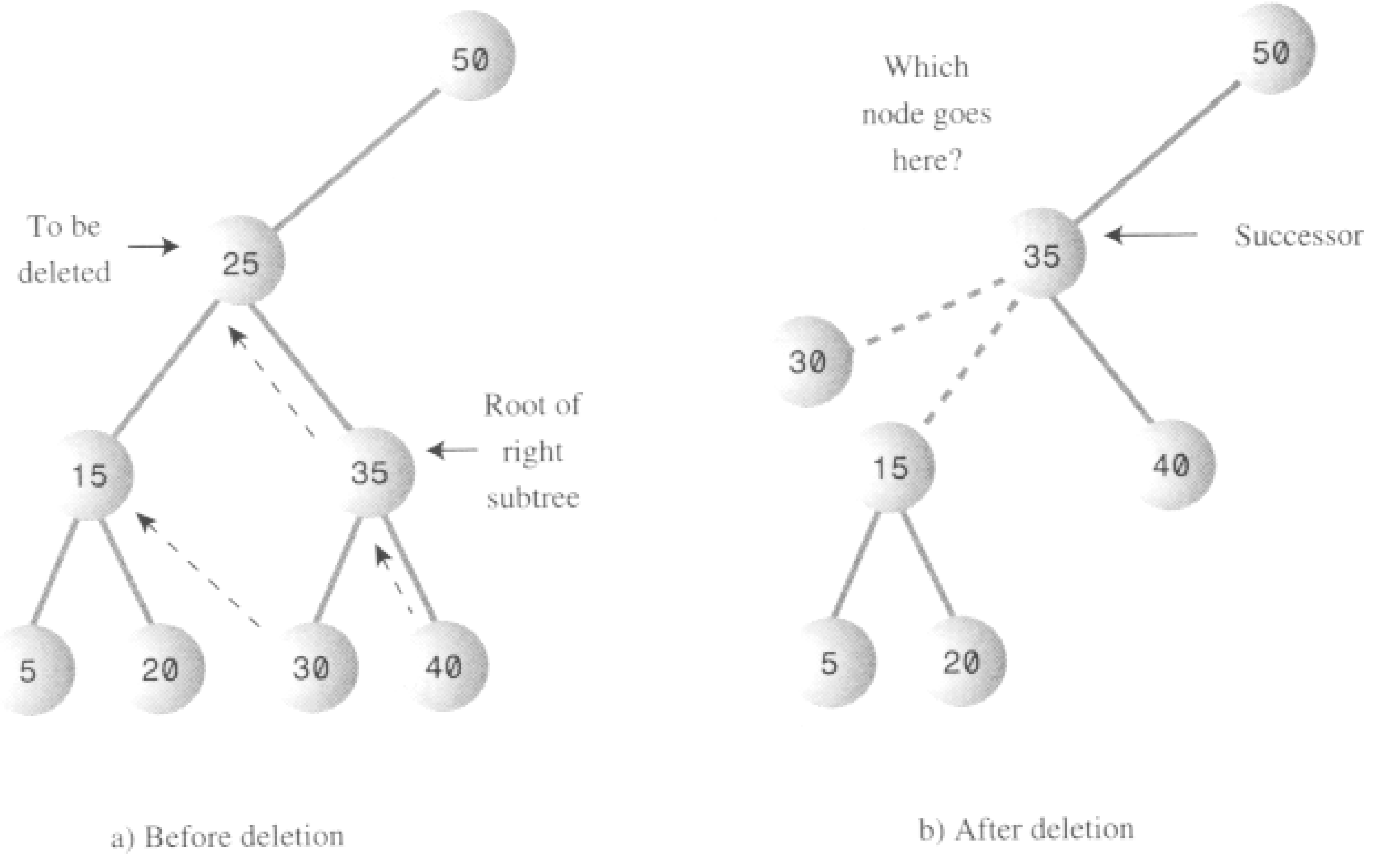
Deletion cases: Leaf Node

- To delete a leaf node, simply change the appropriate child field in the node’s parent to point to *null*, instead of to the node.
- The node still exists, but is no longer a part of the tree.
- Because of Java’s garbage collection feature, the node need not be deleted explicitly.

Deletion: One Child

- The node to be deleted in this case has only two connections: to its parent and to its only child.
- Connect the child of the node to the node’s parent, thus cutting off the connection between the node and its child, and between the node and its parent.

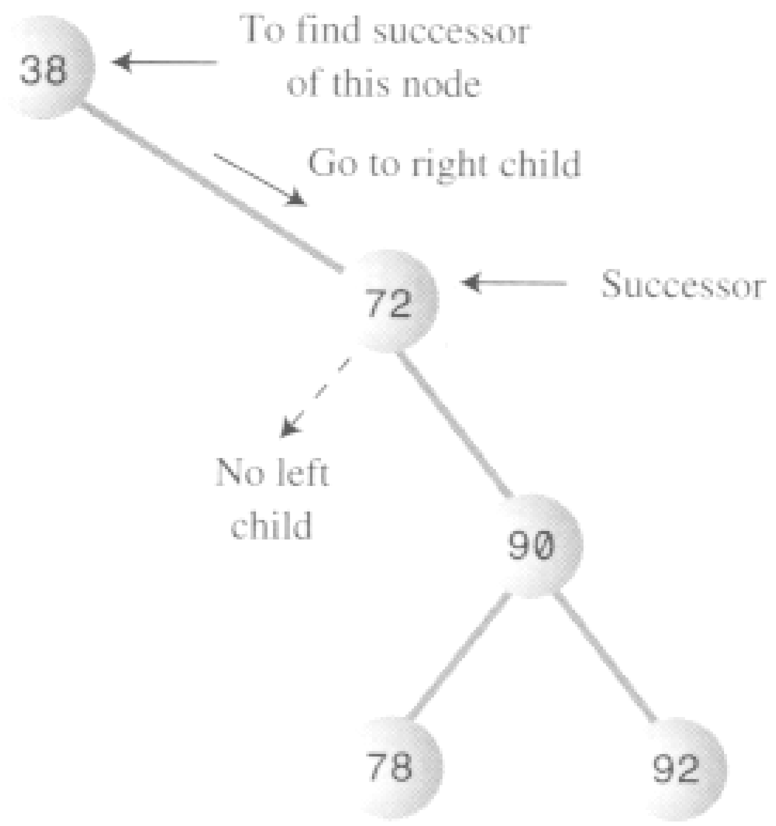
Deletion: Two Children



Deletion: Two Children

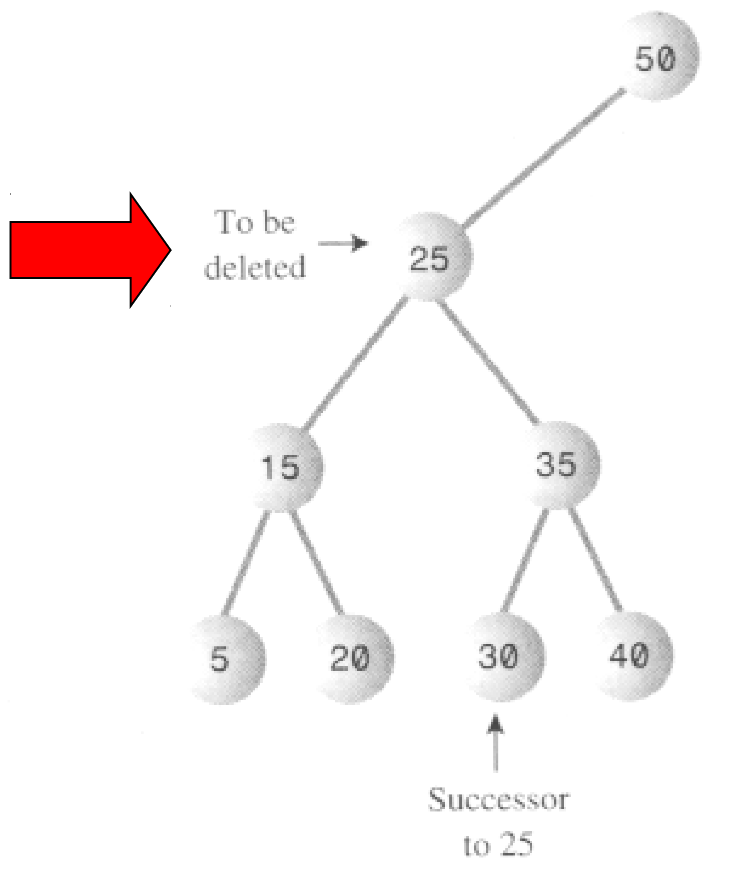
- To delete a node with two children, replace the node with its inorder successor.
- For each node, the node with the next-highest key (to the deleted node) in the subtree is called its inorder successor.
- To find the successor,
  - start with the original (deleted) node’s right child.
  - Then go to this node’s left child and then to its left child and so on, following down the path of left children.
  - The last left child in this path is the successor of the original node.

Find successor

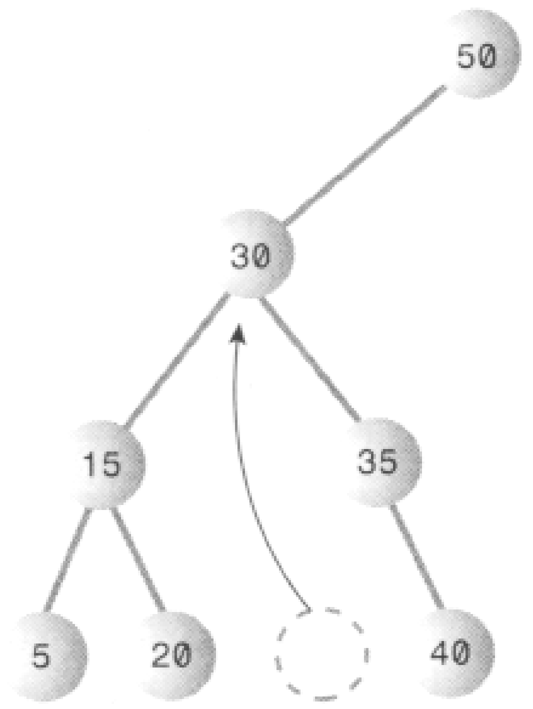


The right child is the successor.

Delete a node with subtree (case 1)

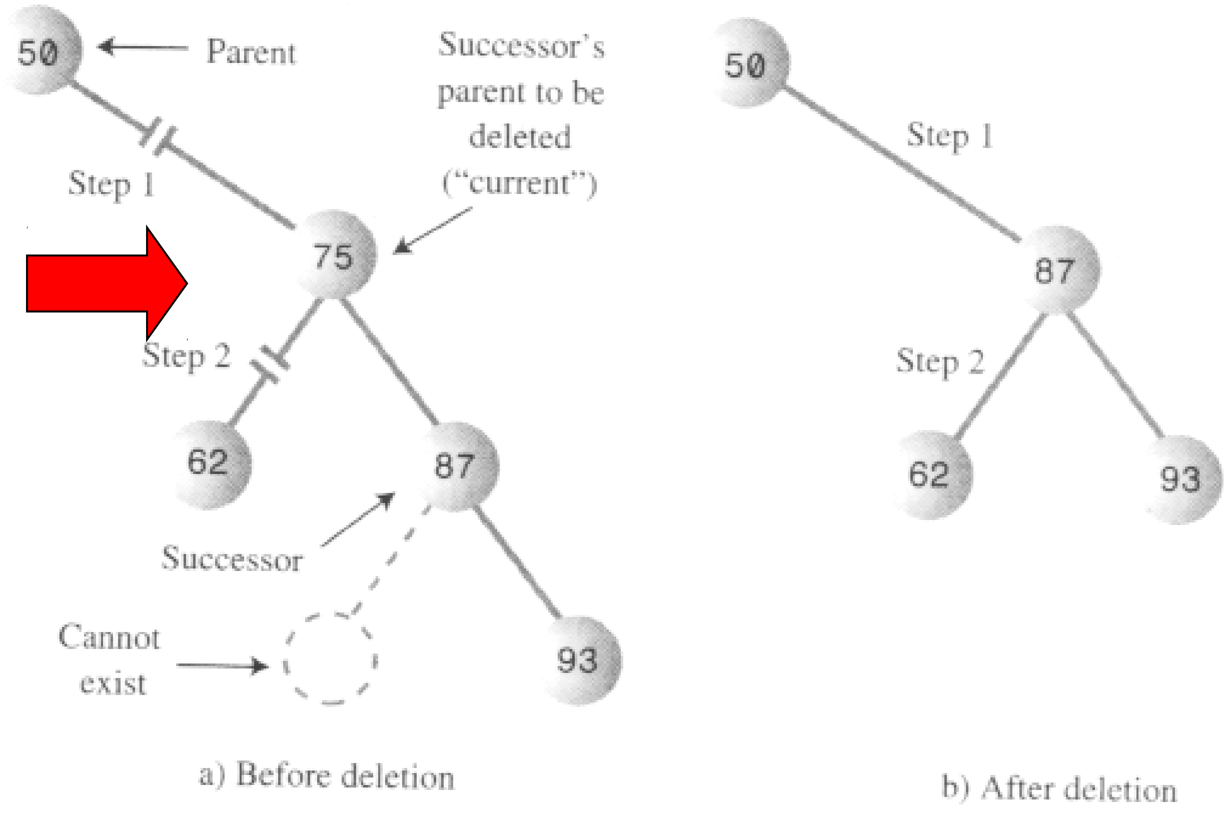


a) Before deletion

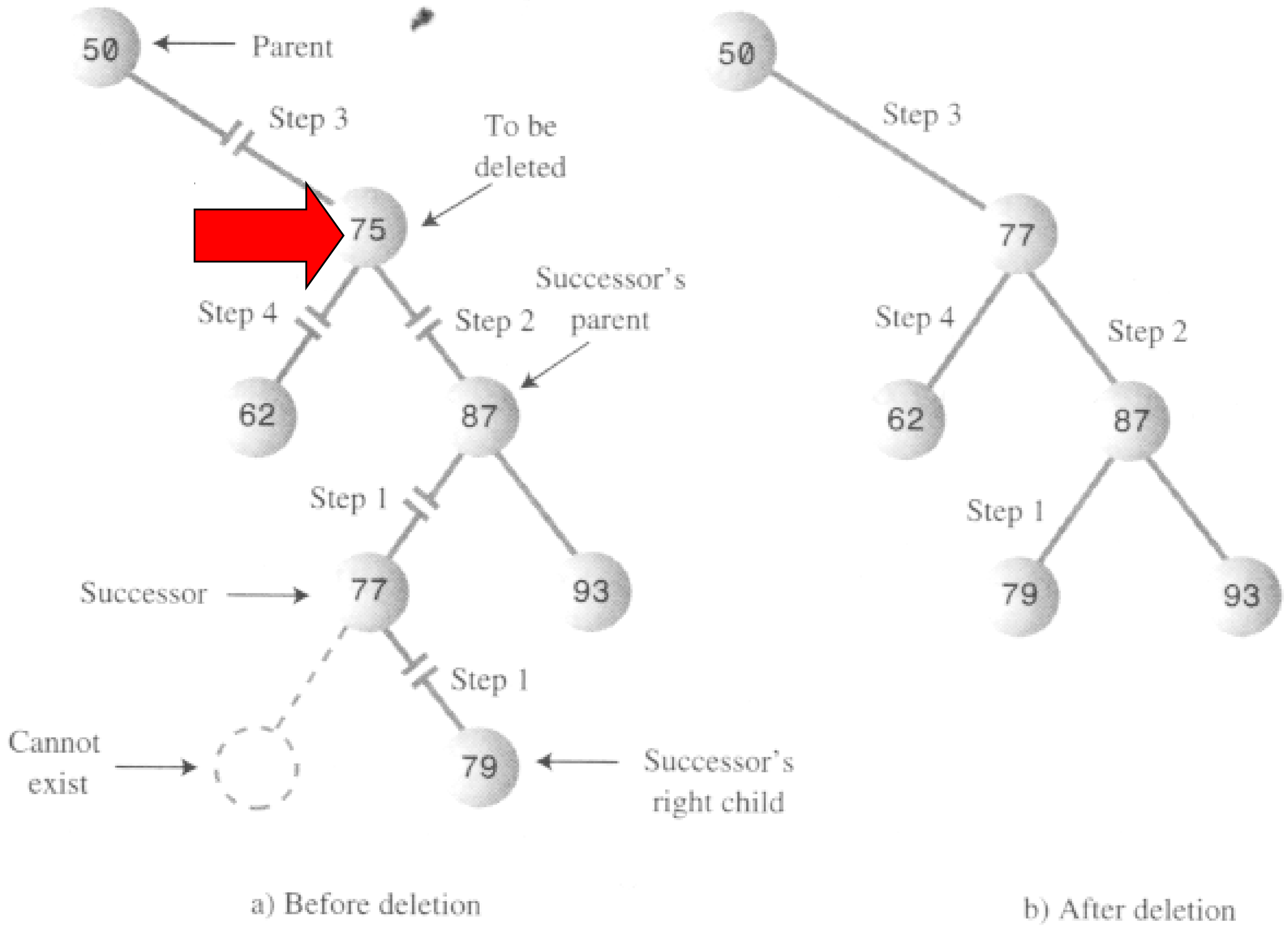


b) After deletion

Delete a node with subtree (case 2)



Delete a node with subtree (case 3)



Exercise

Draw two binary trees whose preorder listing is ‘abcdefgh’ and whose postorder listing is ‘dcbgfhea’. List the nodes of binary trees in inorder and levelorder.

Solution:

Here is one tree:

```
         a
       /   \
     b       e
      \     / \
       c   f   h
        \   \
         d   g
```

The inorder listing is `bcdafgeh`

The level order listing is `abecfhdg`

Here is another tree:

```
         a
       /   \
     b       e
      \     / \
       c   f   h
      /   /
     d   g
```

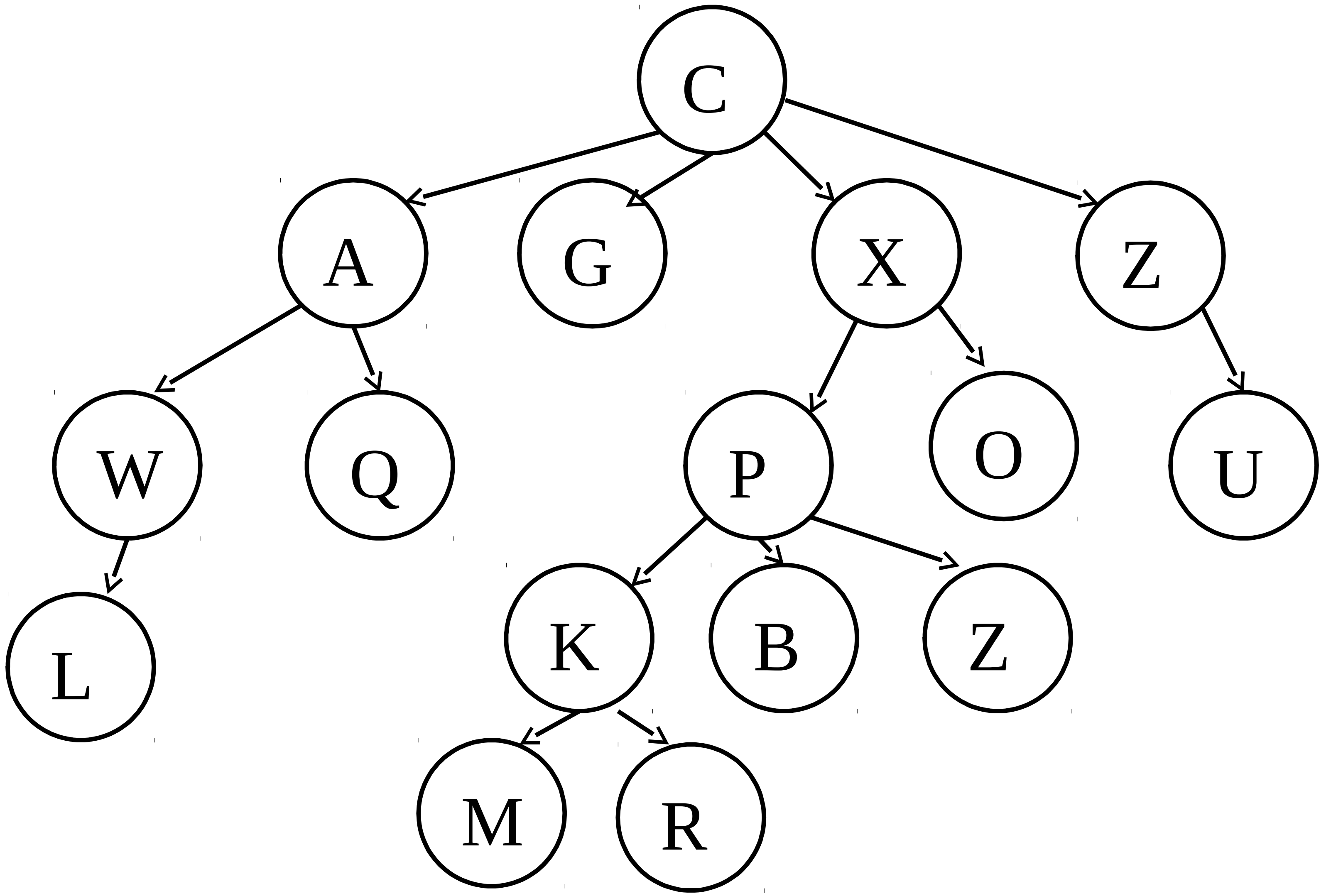
The inorder listing is `bdcagfeh`

The level order listing is `abecfhdg`

Breadth First

- A level order traversal of a tree could be used as a breadth first search
- Search all nodes in a level before going down to the next level

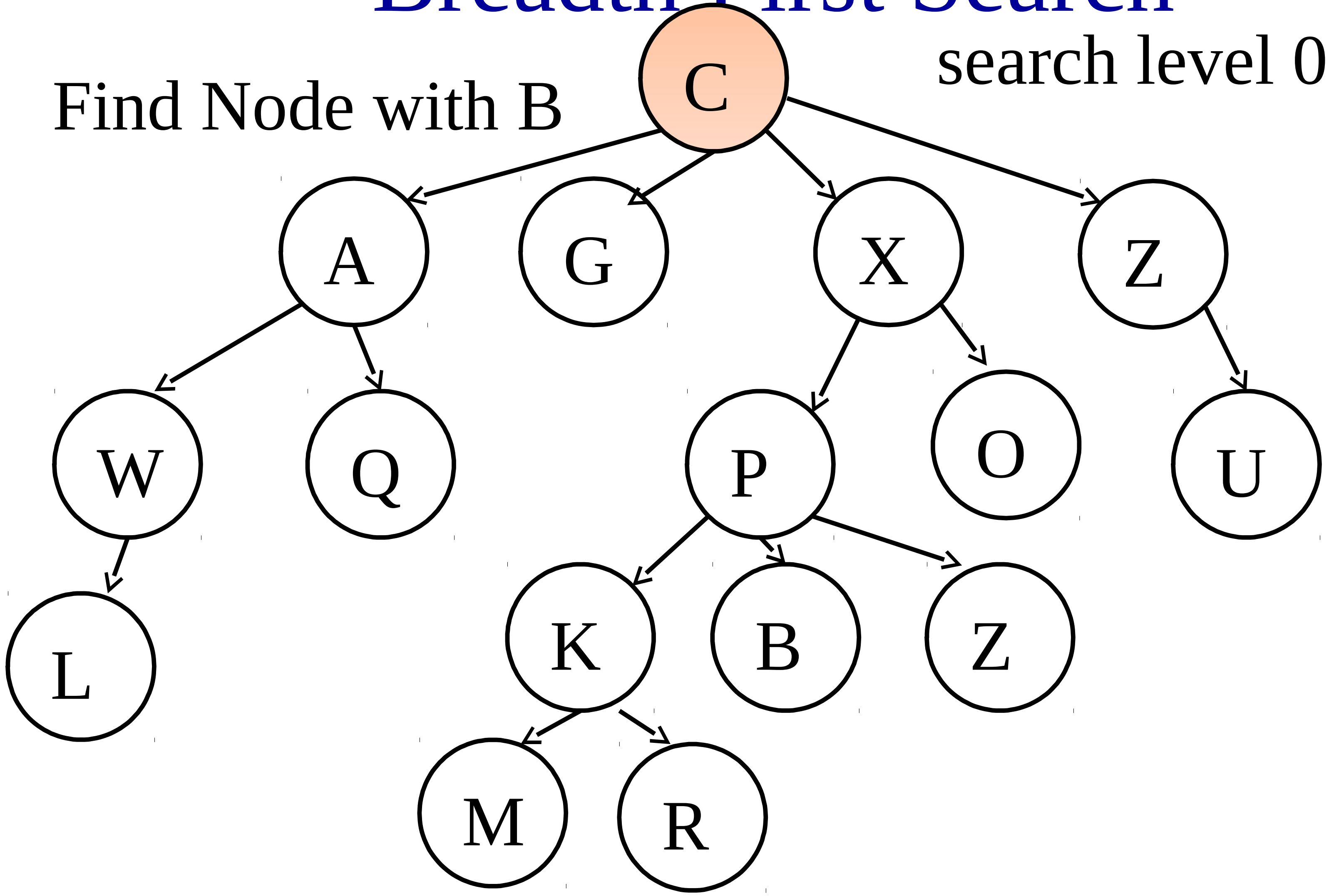
Breadth First Search of Tree



Breadth First Search

Find Node with B

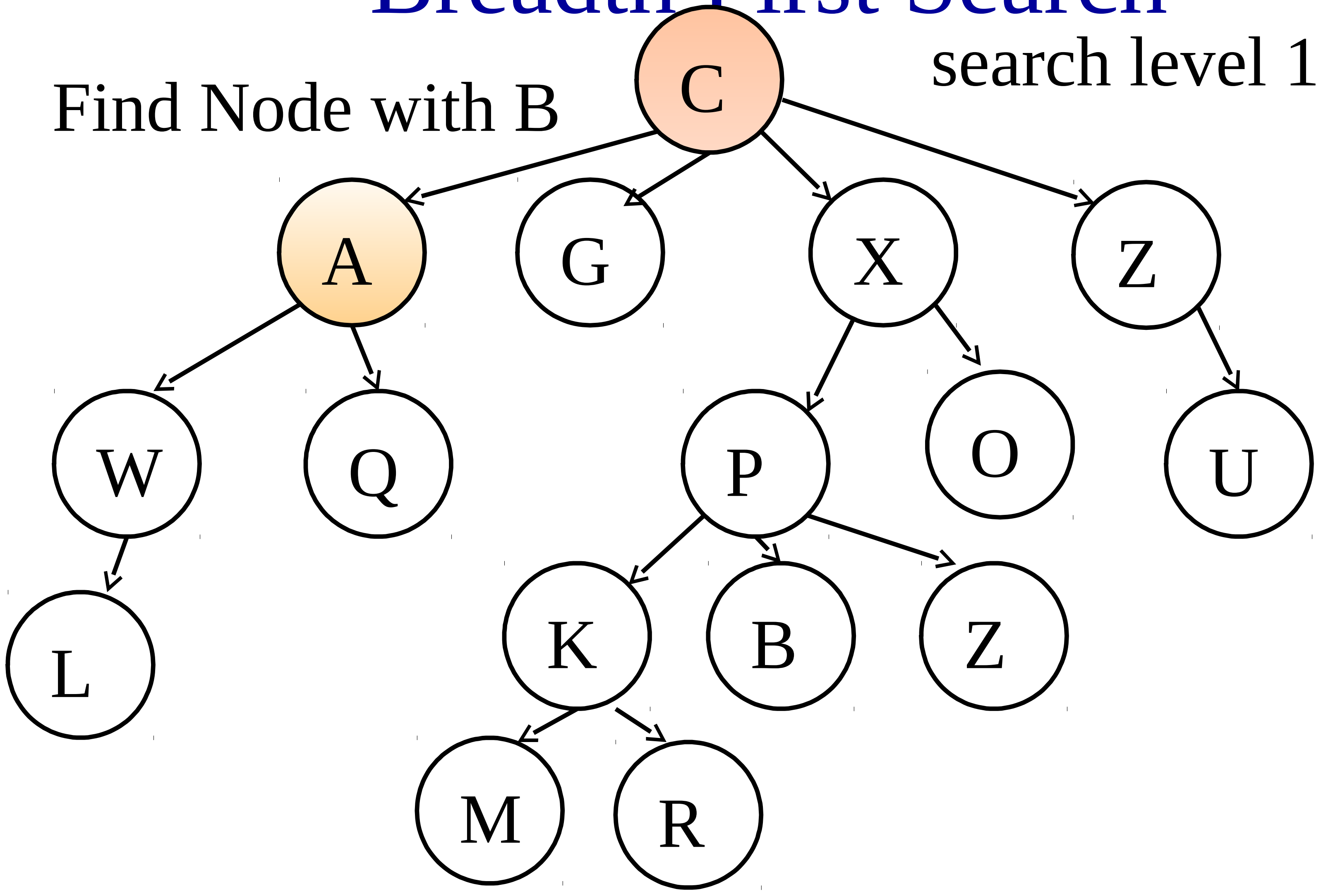
search level 0 first



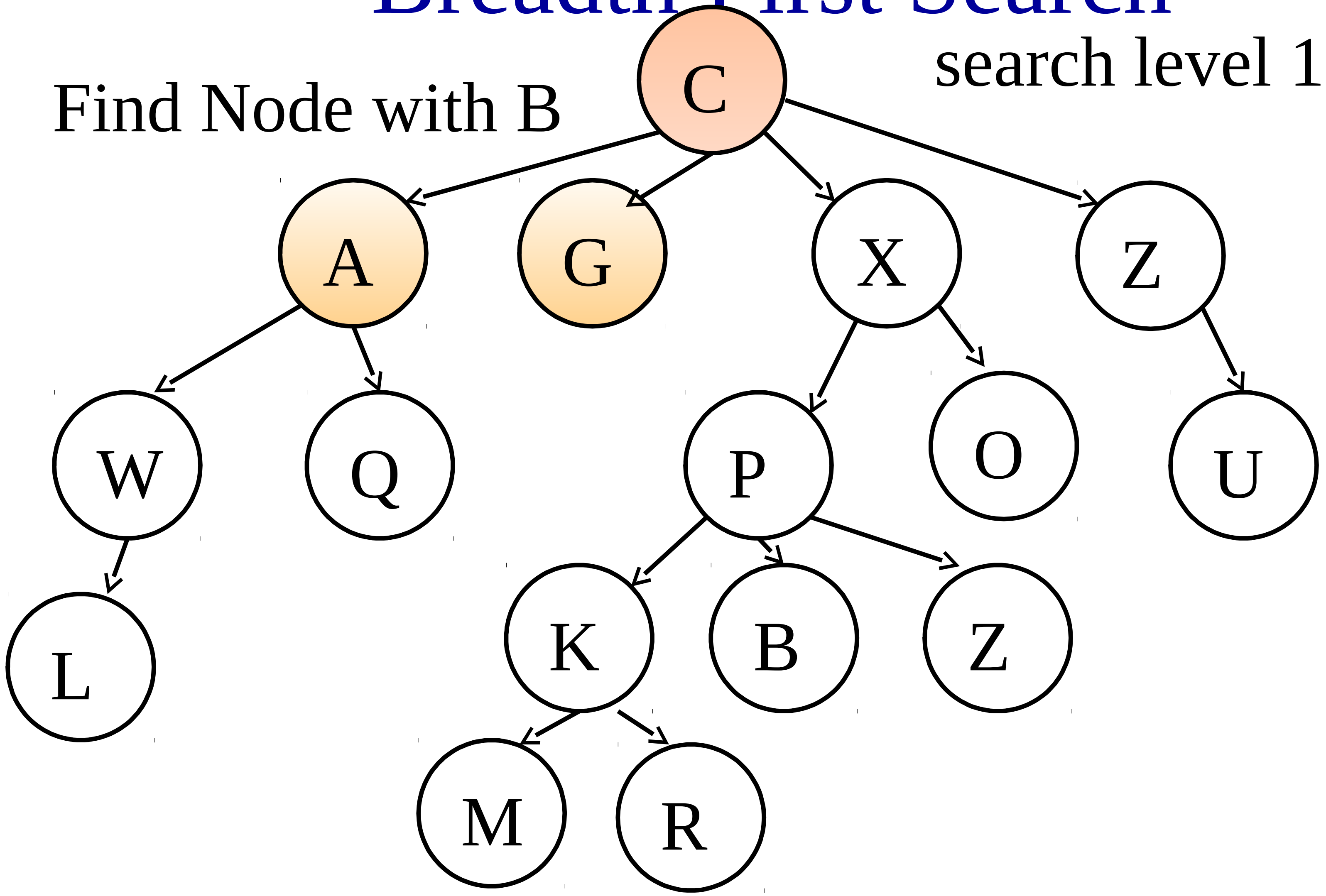
Breadth First Search

search level 1

Find Node with B



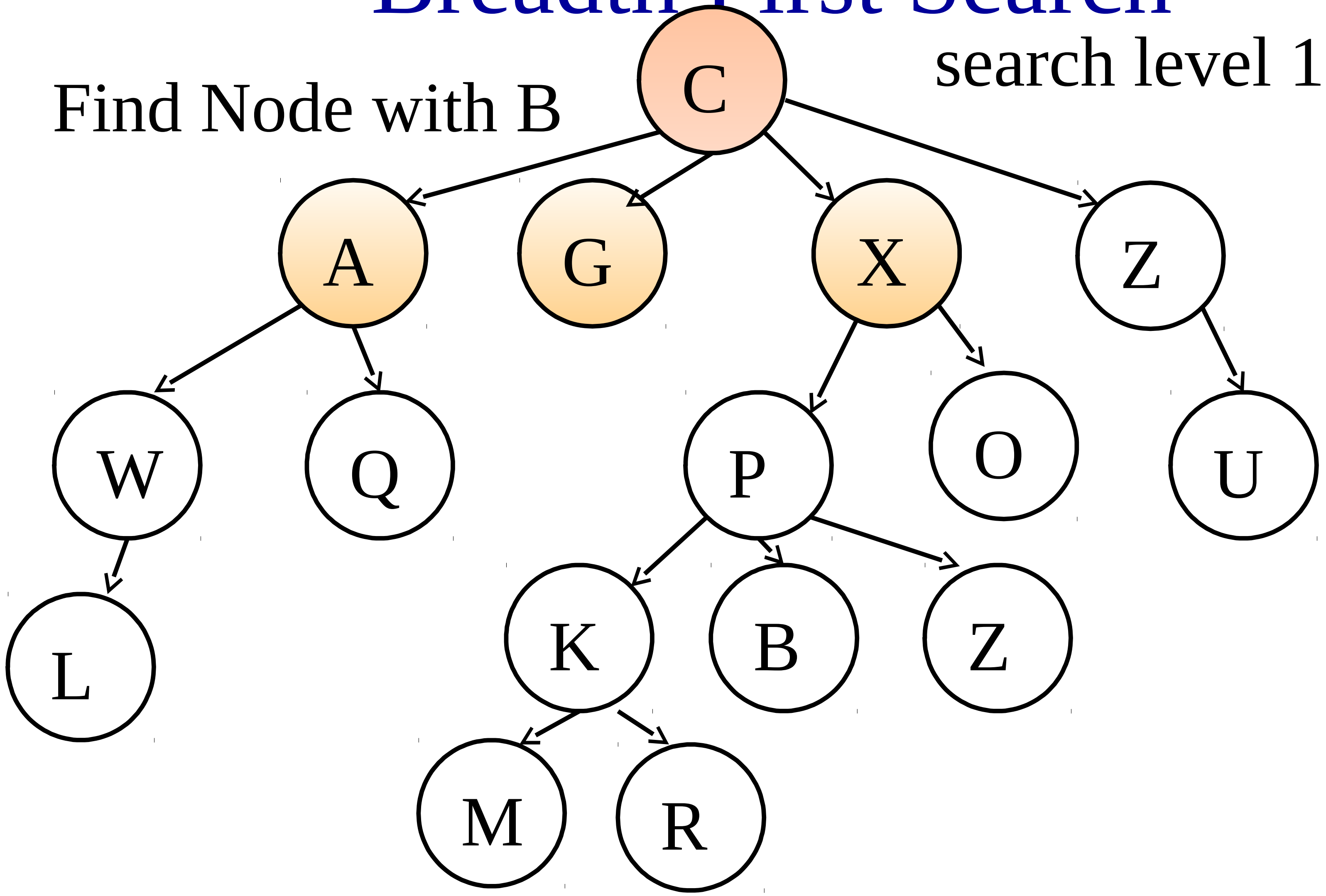
Breadth First Search



Breadth First Search

Find Node with B

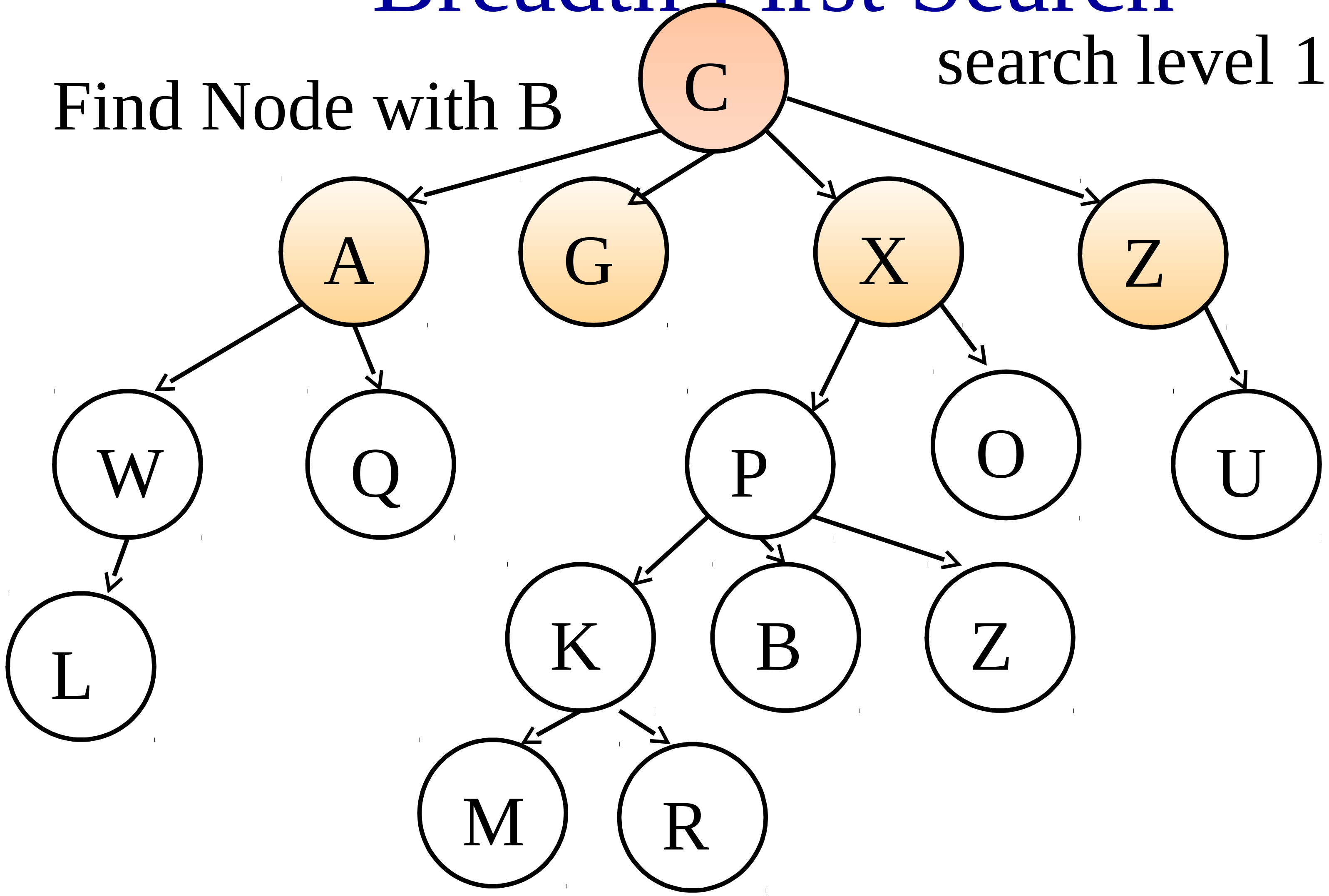
search level 1



Breadth First Search

Find Node with B

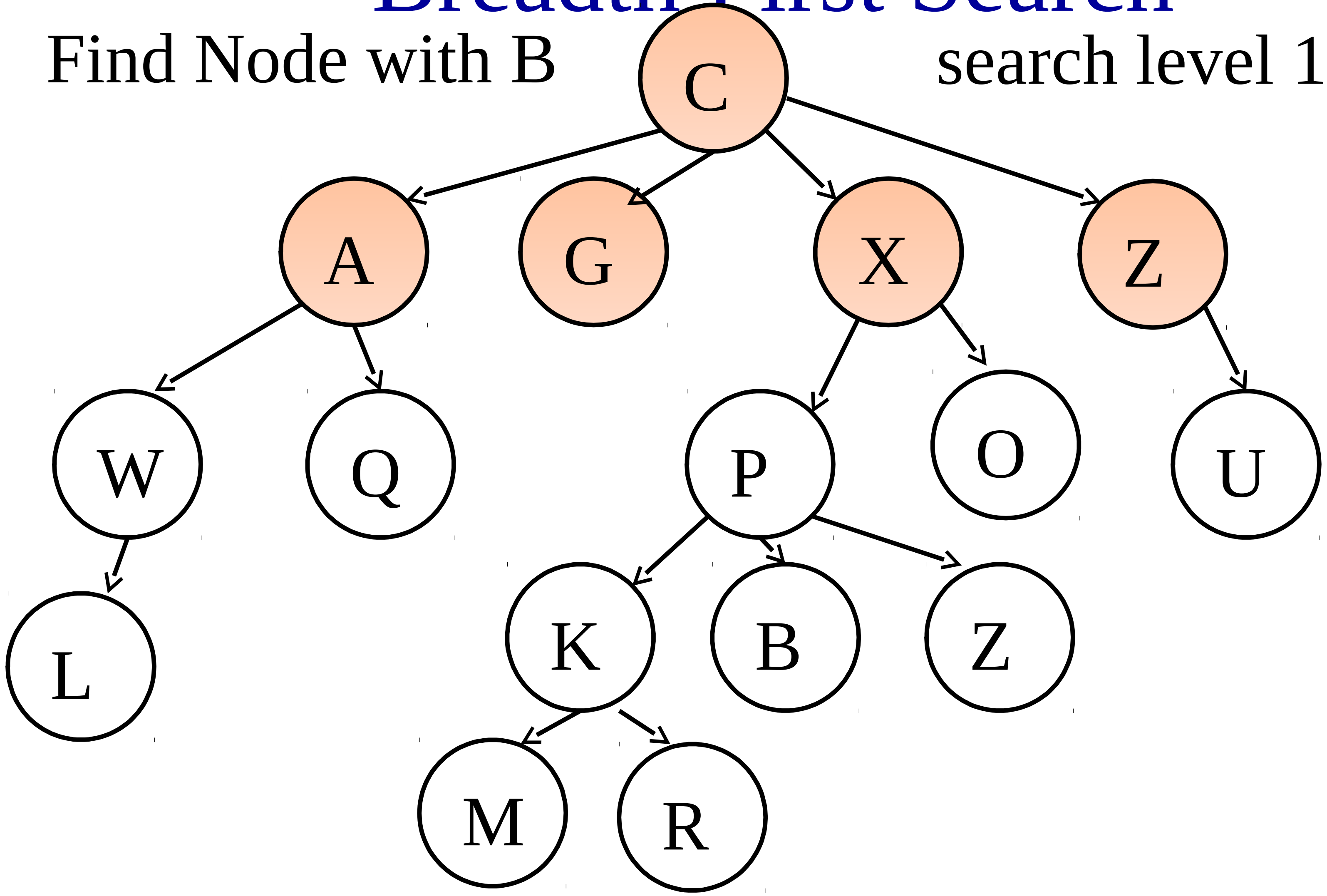
search level 1



Breadth First Search

Find Node with B

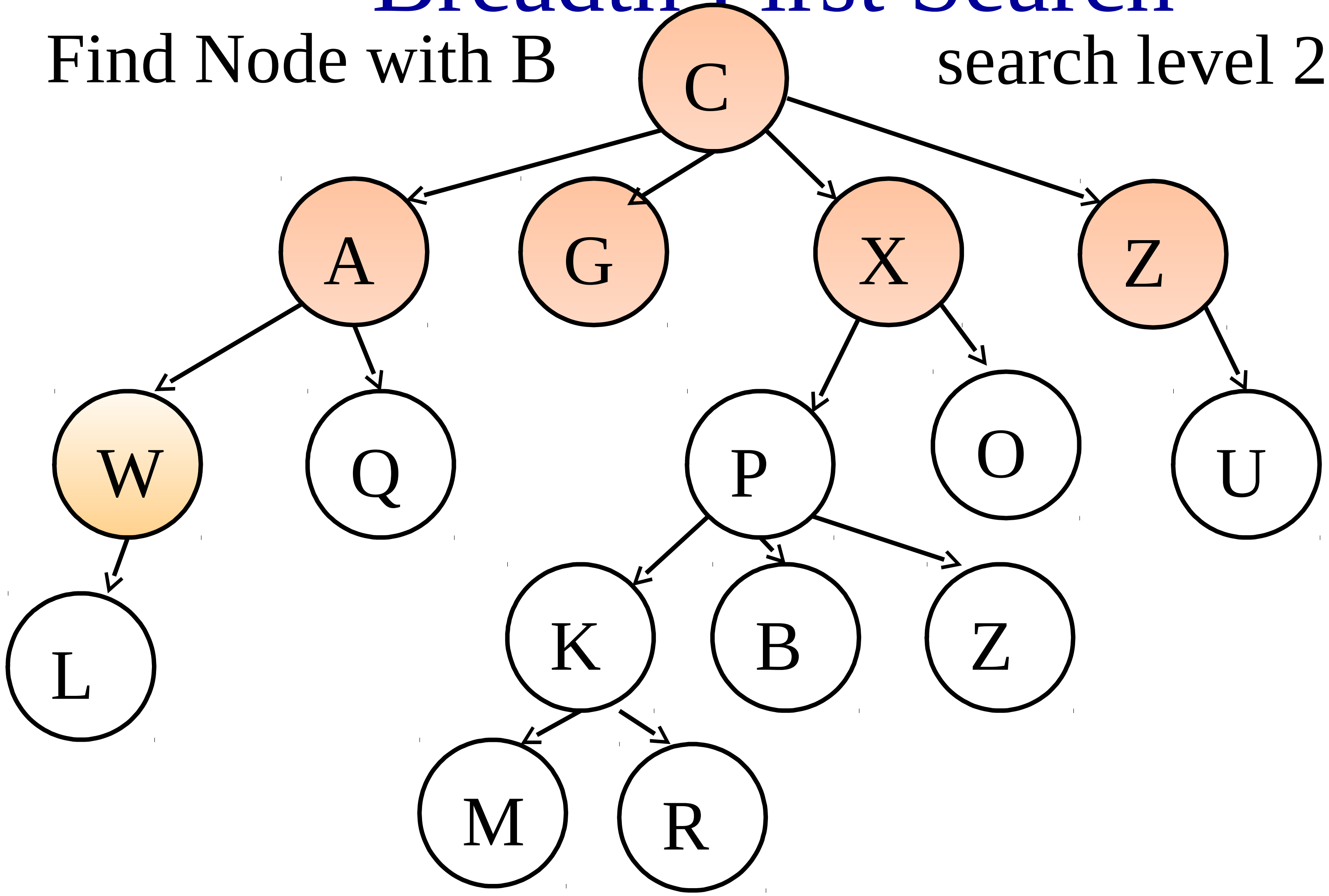
search level 1 next



Breadth First Search

Find Node with B

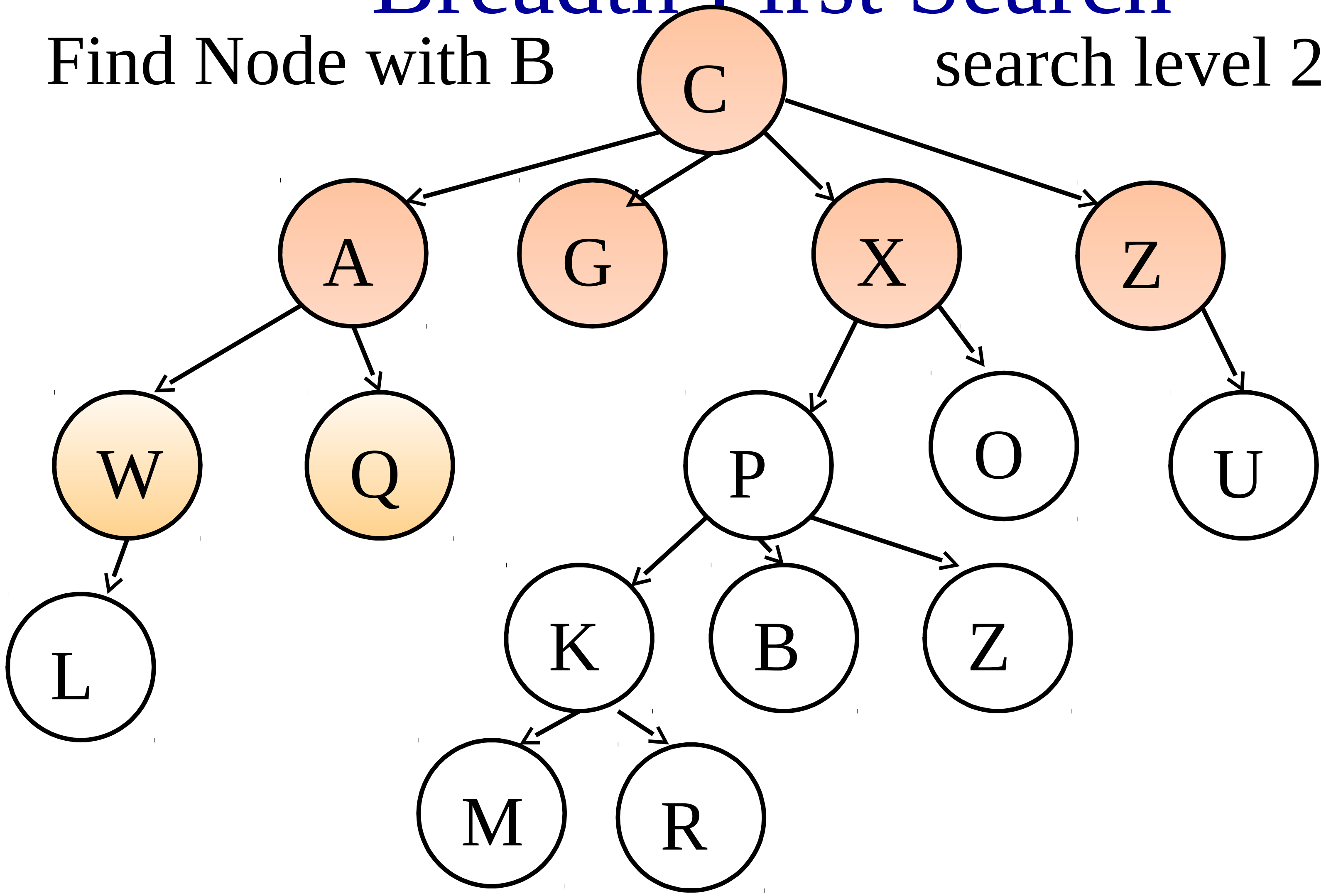
search level 2 next



Breadth First Search

Find Node with B

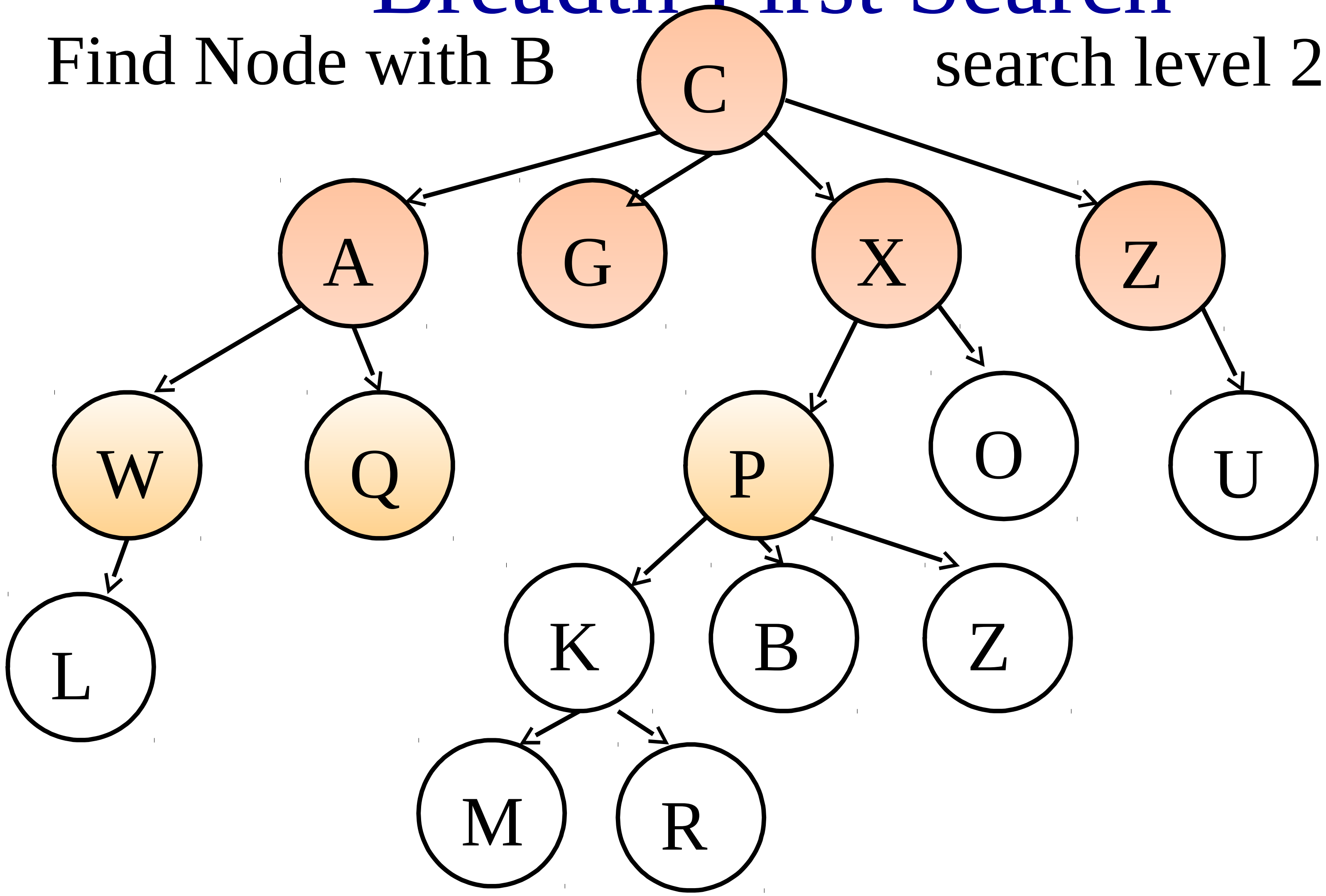
search level 2 next



Breadth First Search

Find Node with B

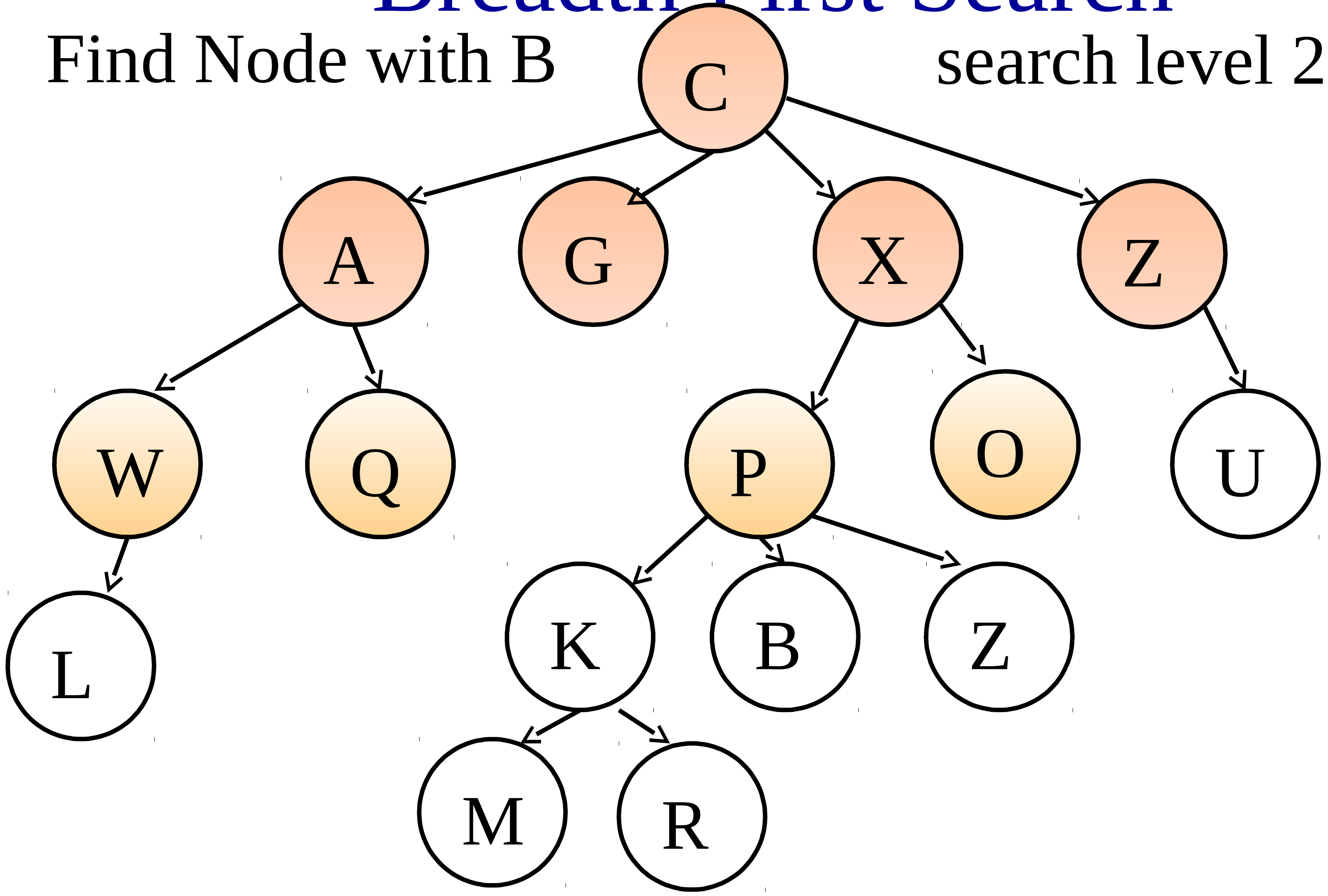
search level 2 next



Breadth First Search

Find Node with B

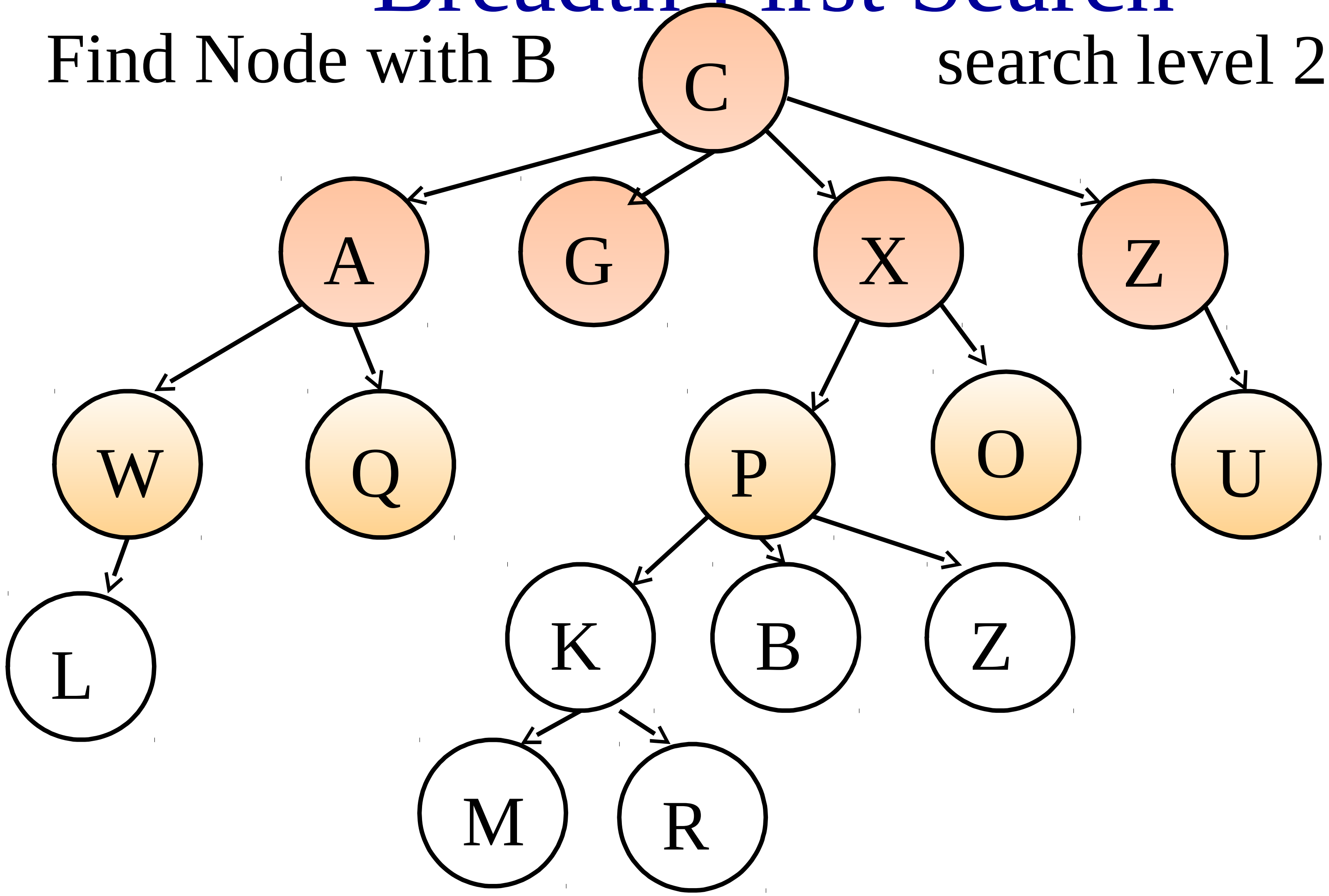
search level 2 next



Breadth First Search

Find Node with B

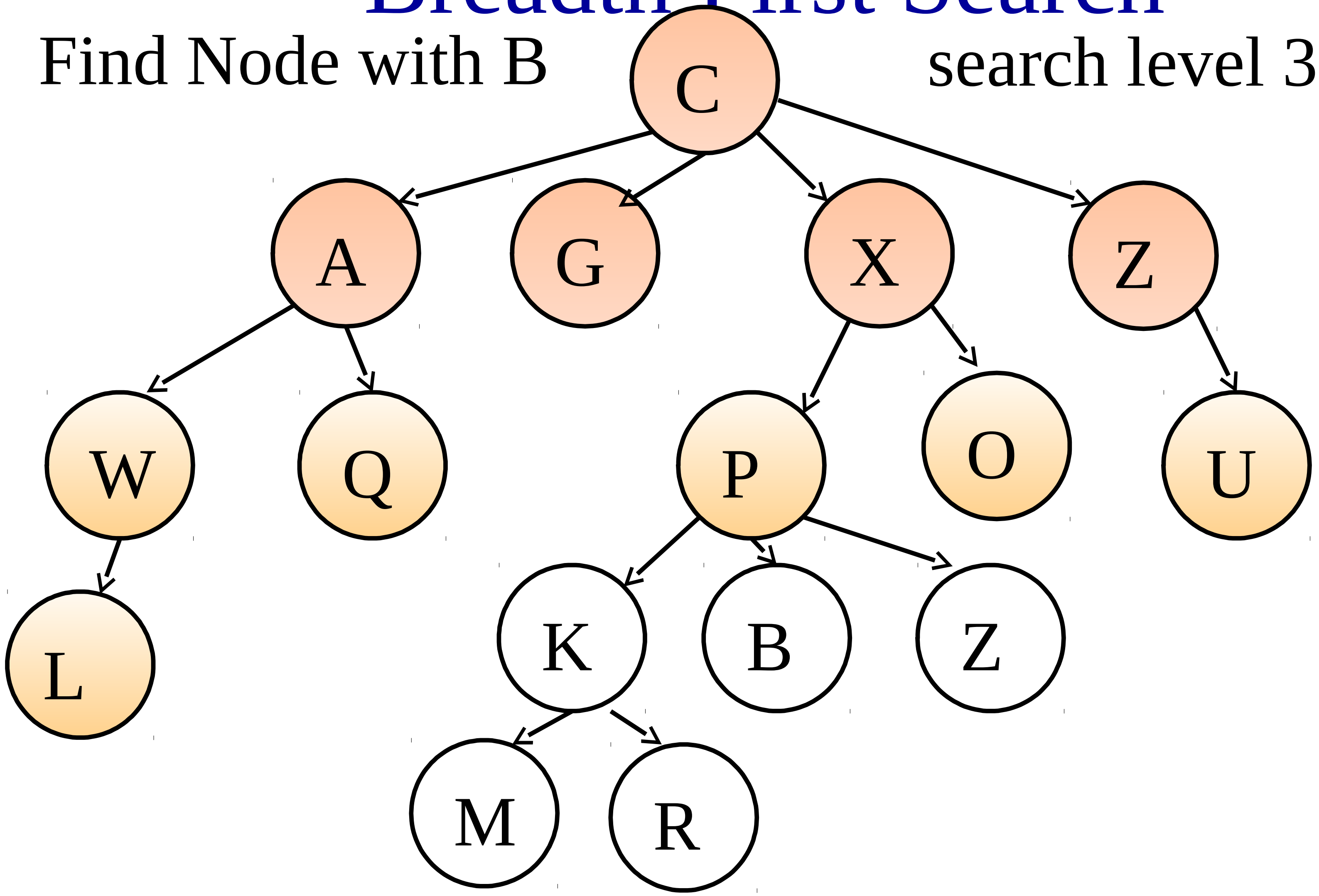
search level 2 next



Breadth First Search

Find Node with B

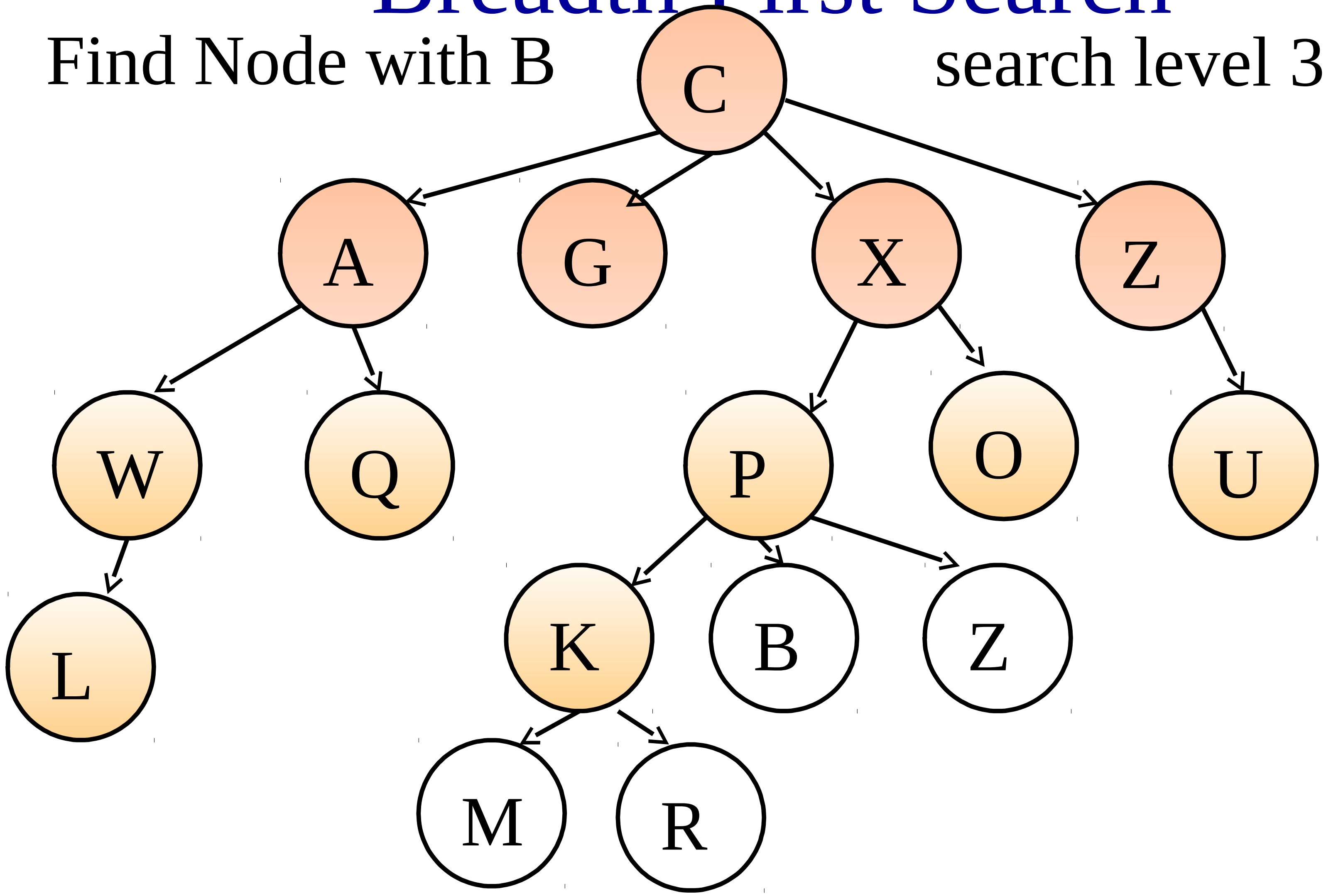
search level 3 next



Breadth First Search

Find Node with B

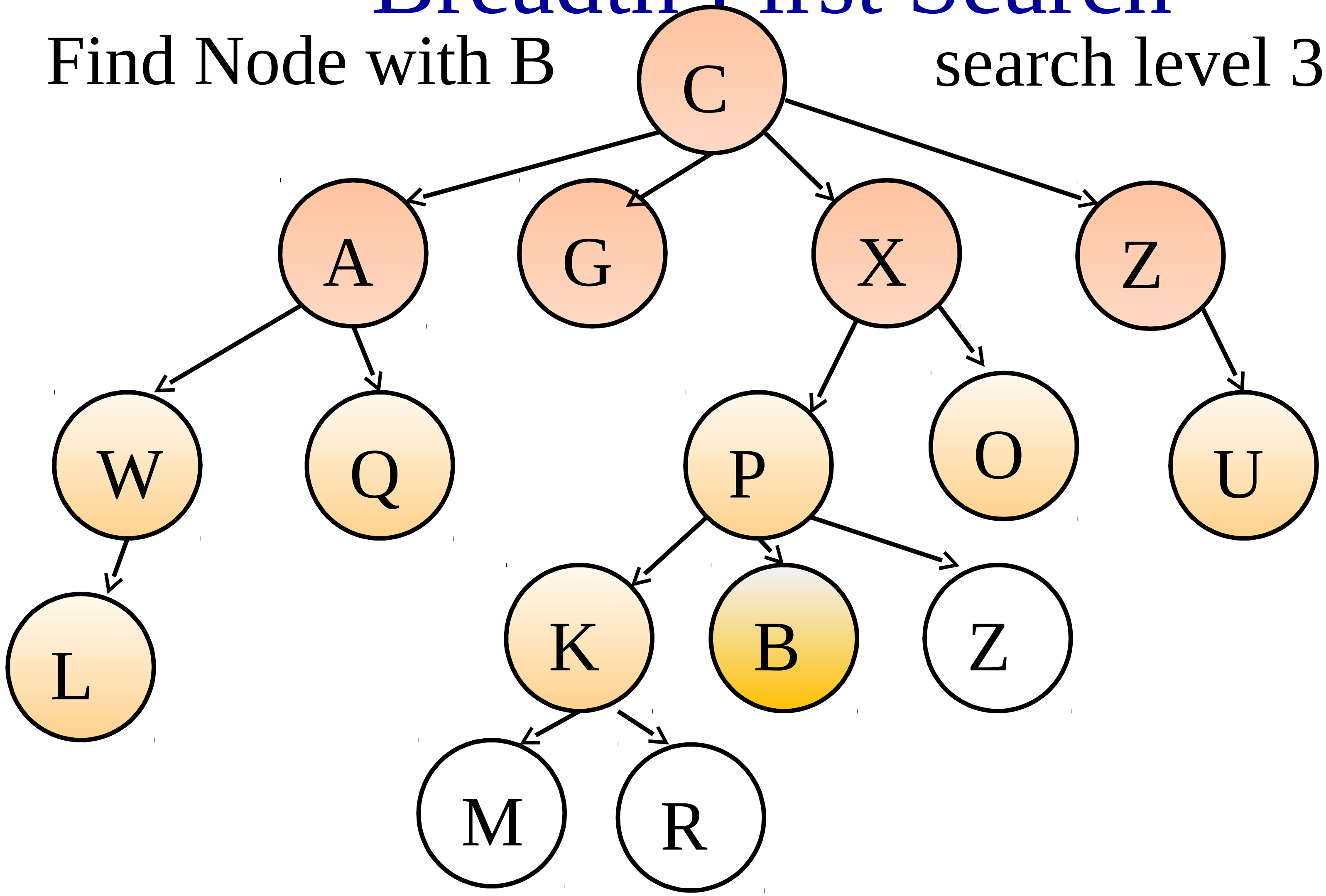
search level 3 next



Breadth First Search

Find Node with B

search level 3 next

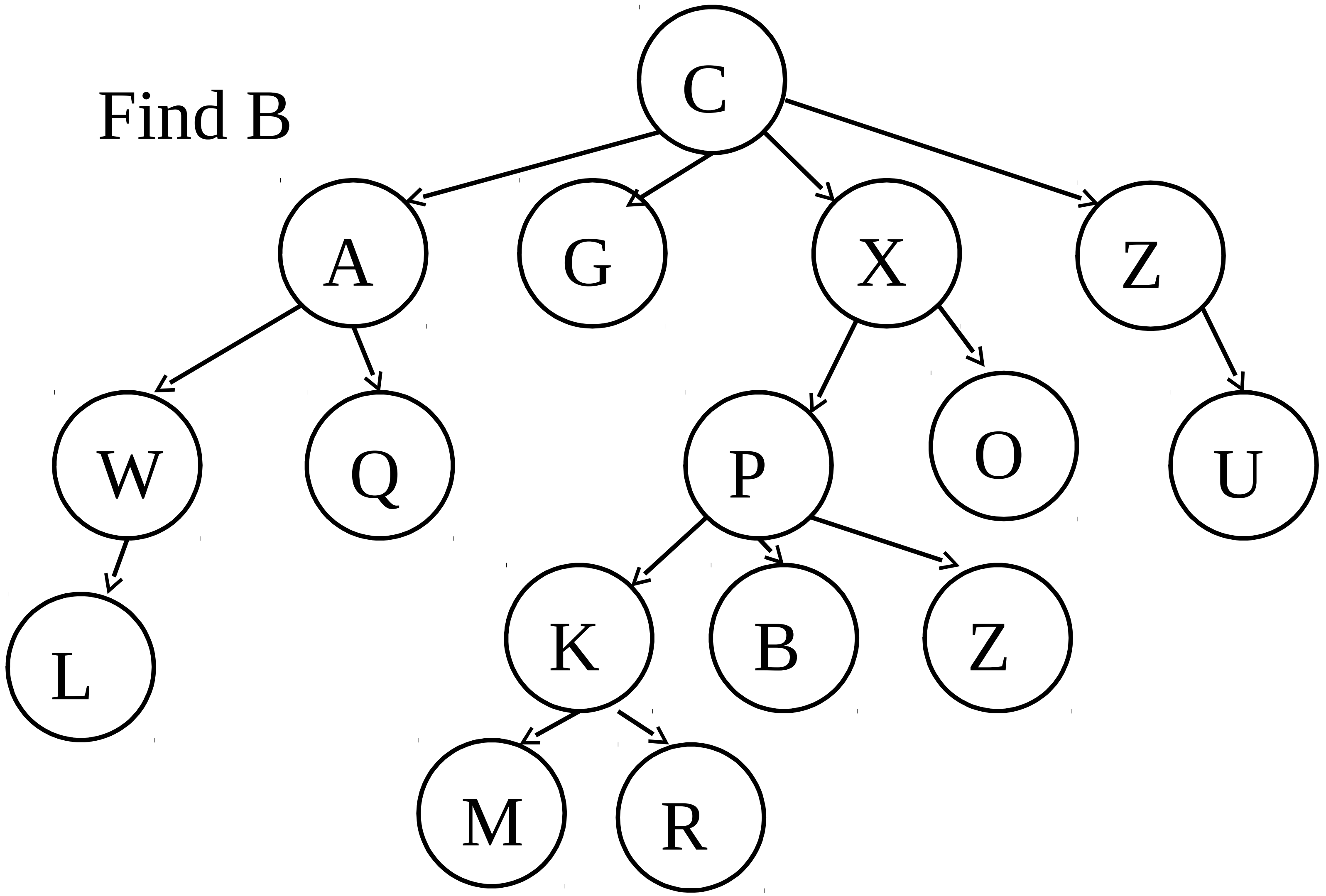


BFS - DFS

- Breadth first search typically implemented with a Queue
- Depth first search typically implemented with a stack, implicit with recursion or iteratively with an explicit stack

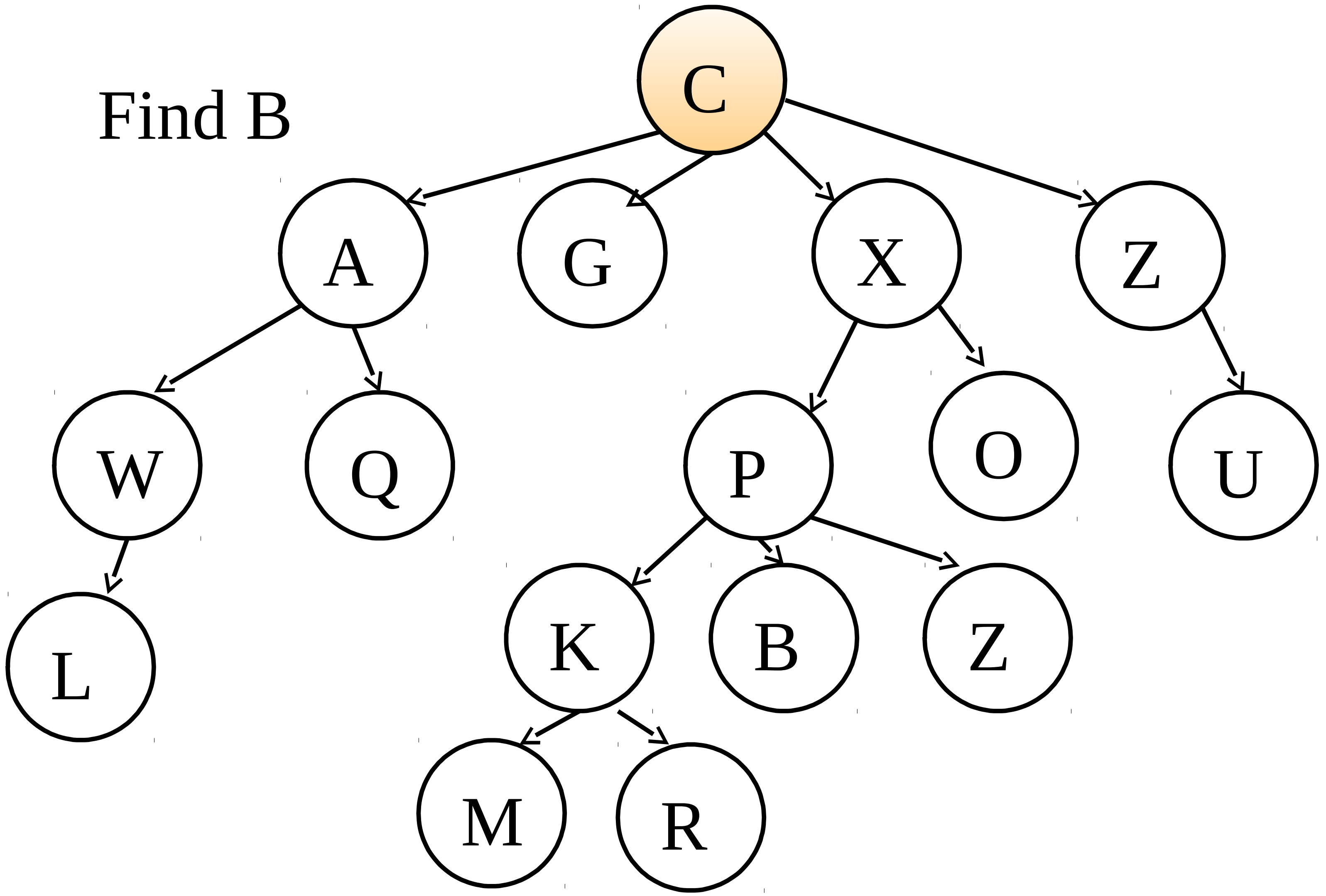
Depth First Search of Tree

Find B



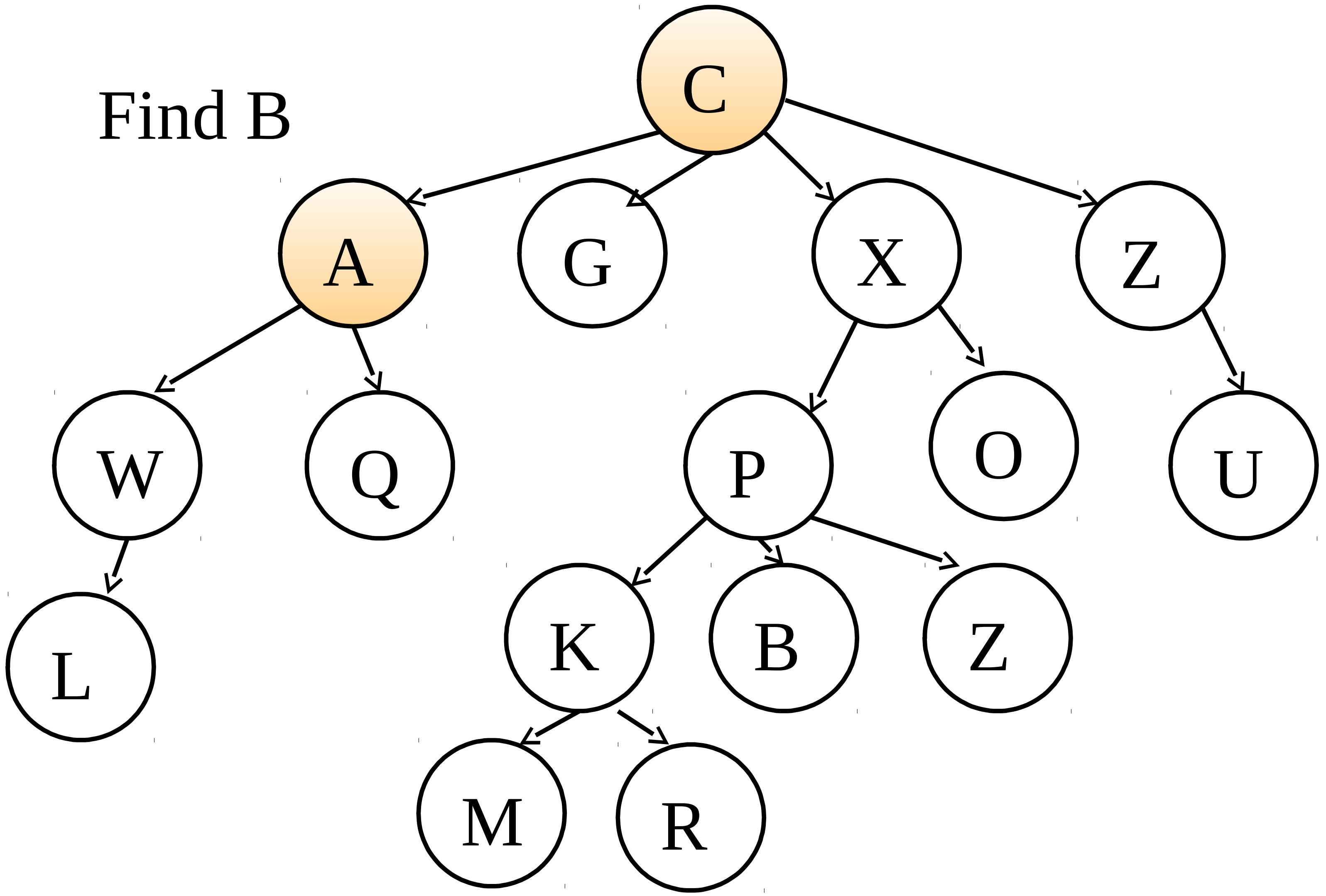
Depth First Search of Tree

Find B



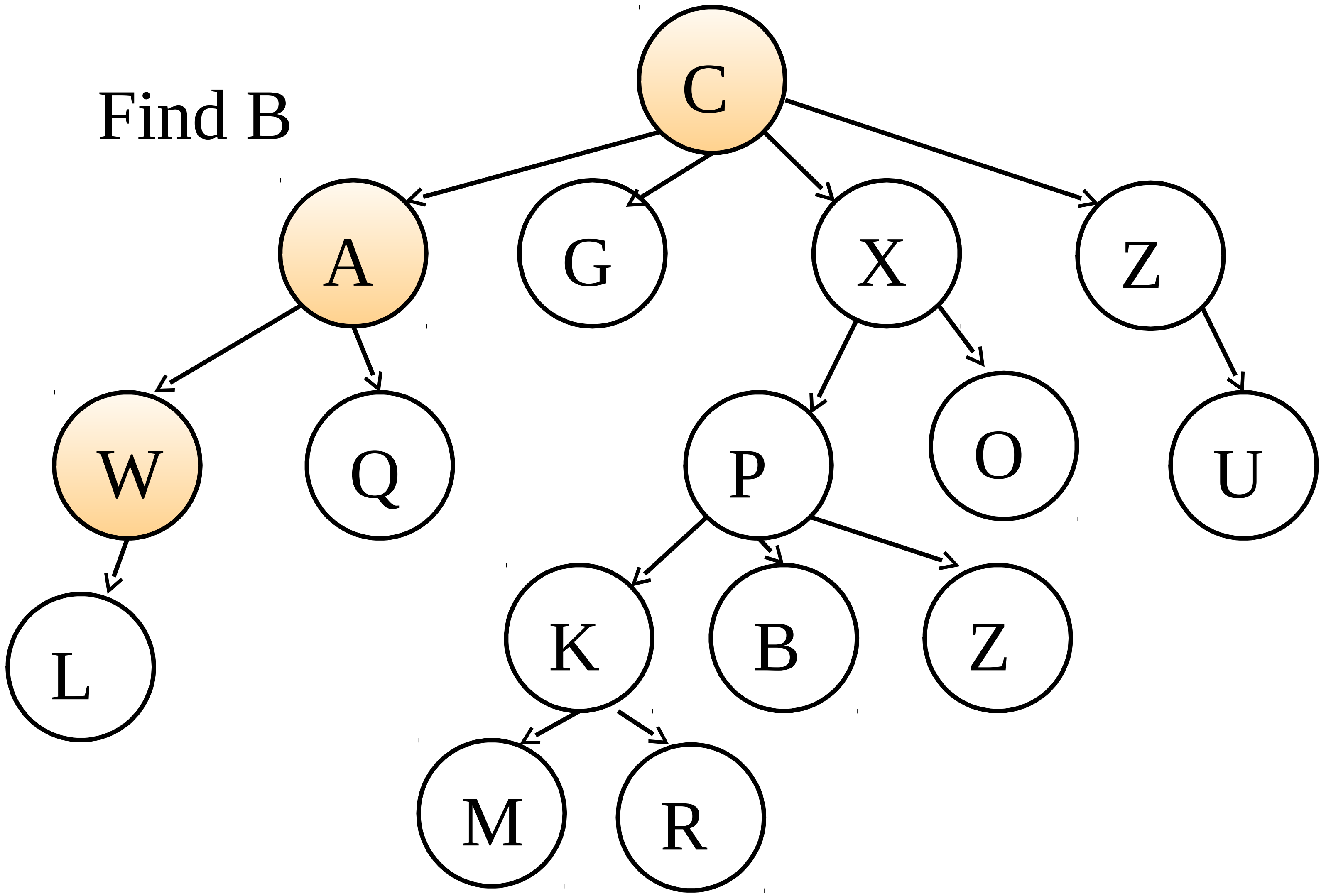
Depth First Search of Tree

Find B



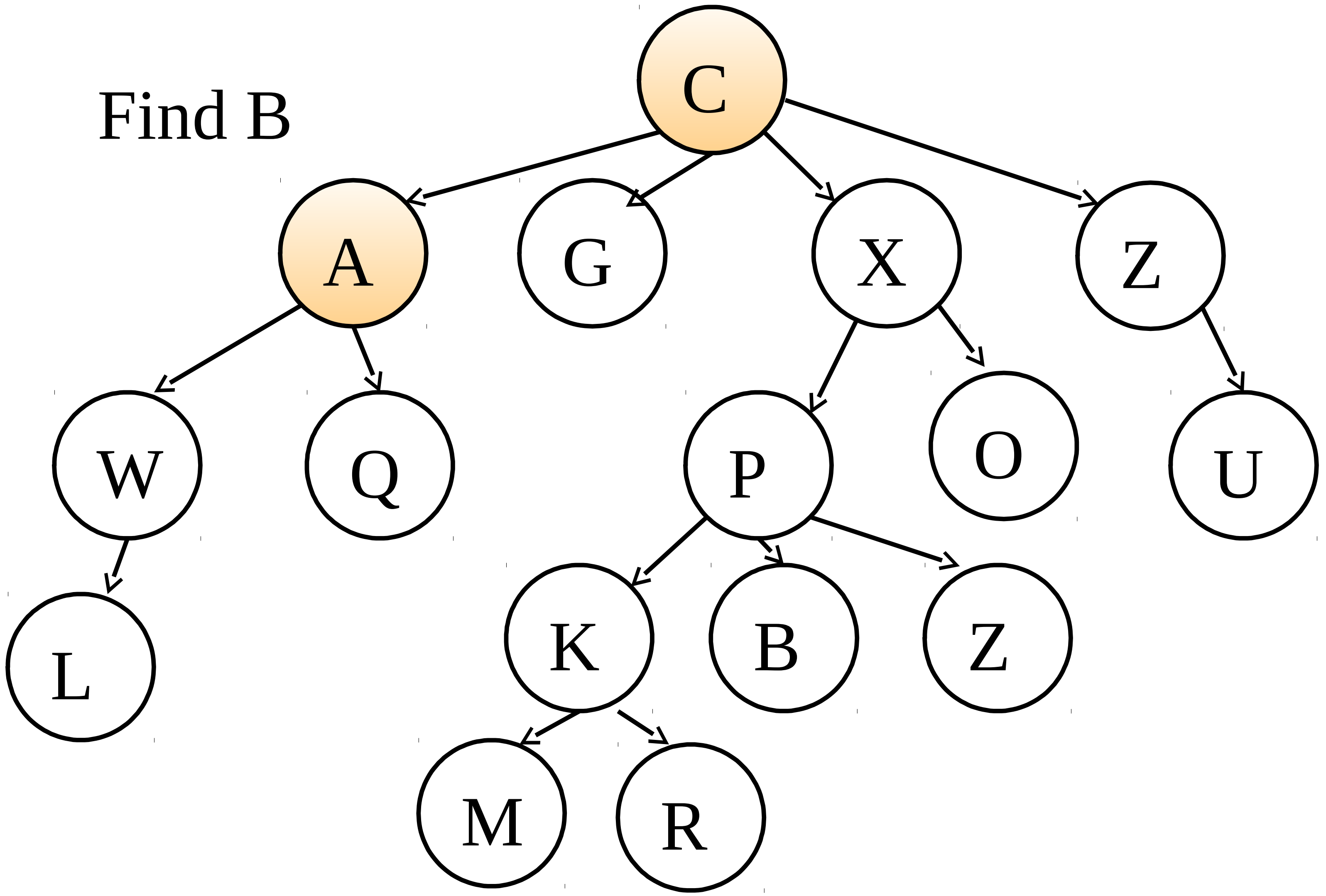
Depth First Search of Tree

Find B



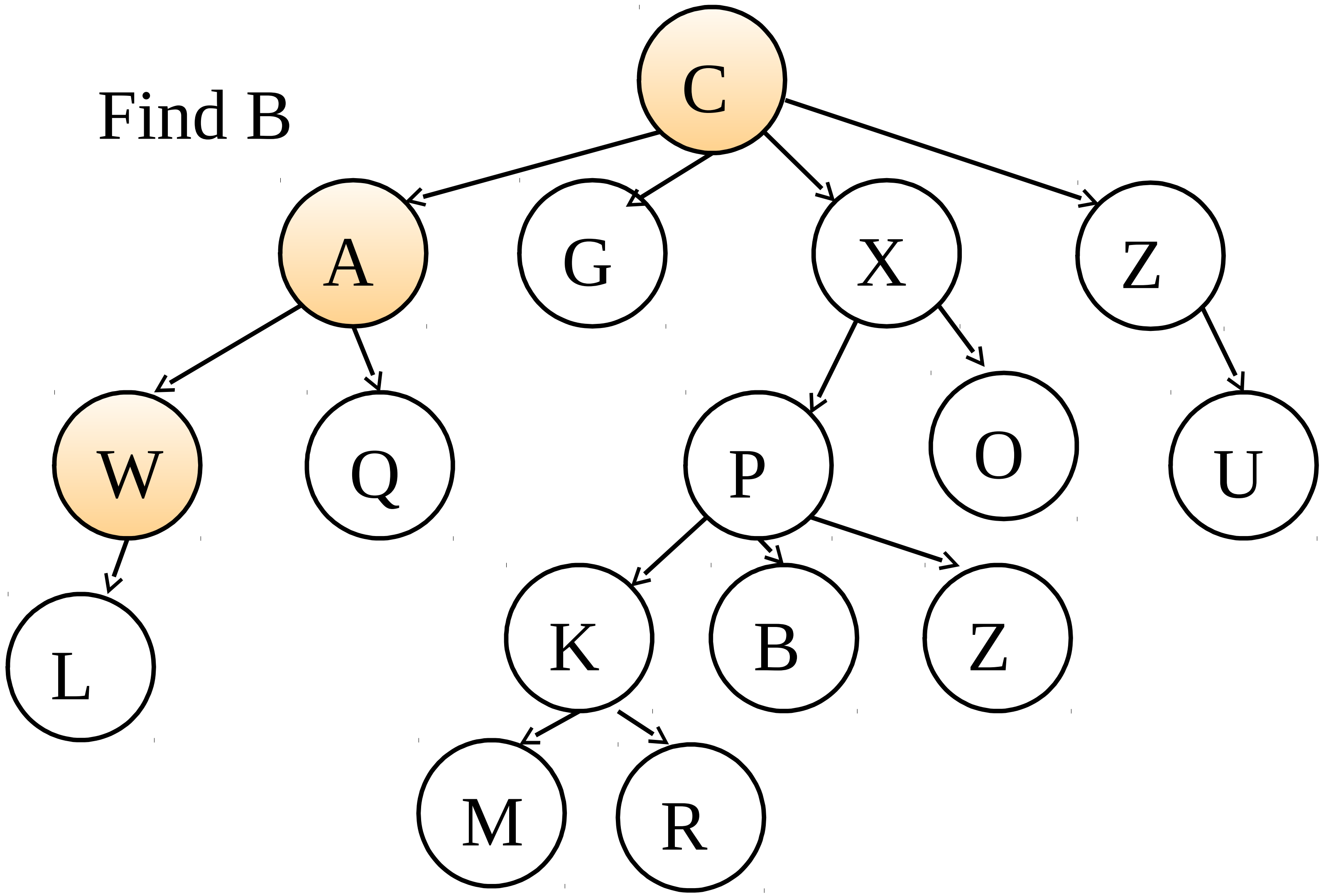
Depth First Search of Tree

Find B



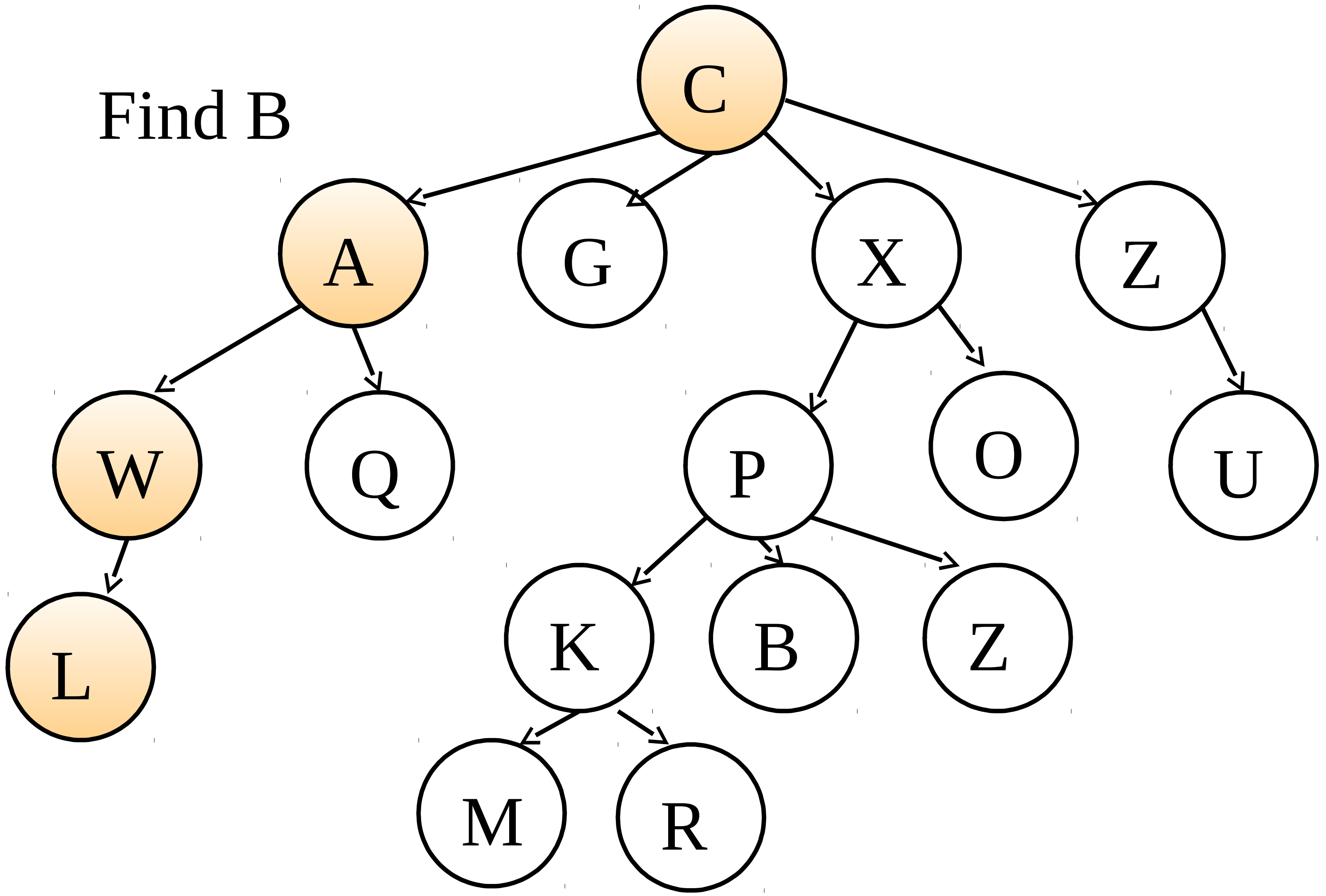
Depth First Search of Tree

Find B



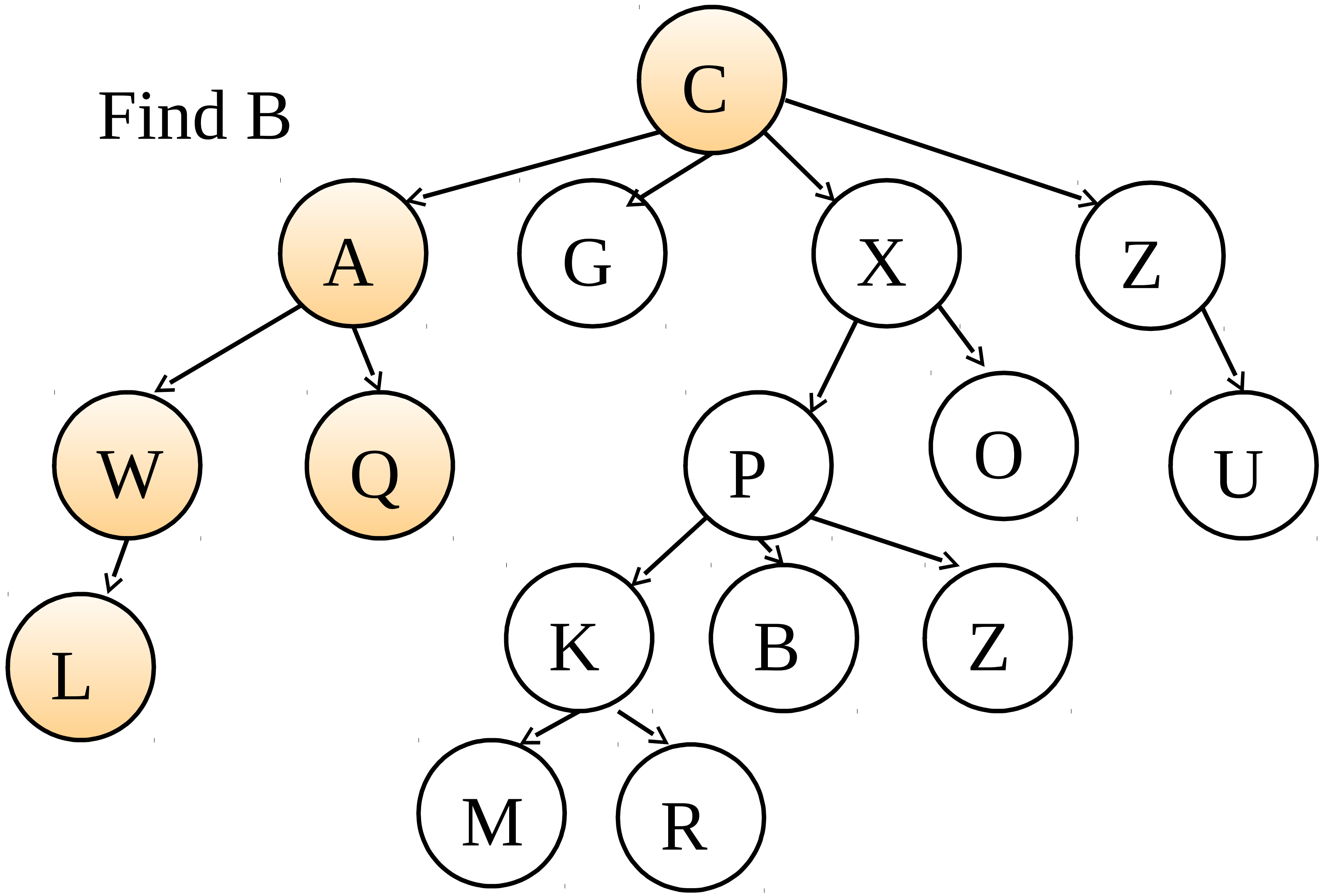
Depth First Search of Tree

Find B



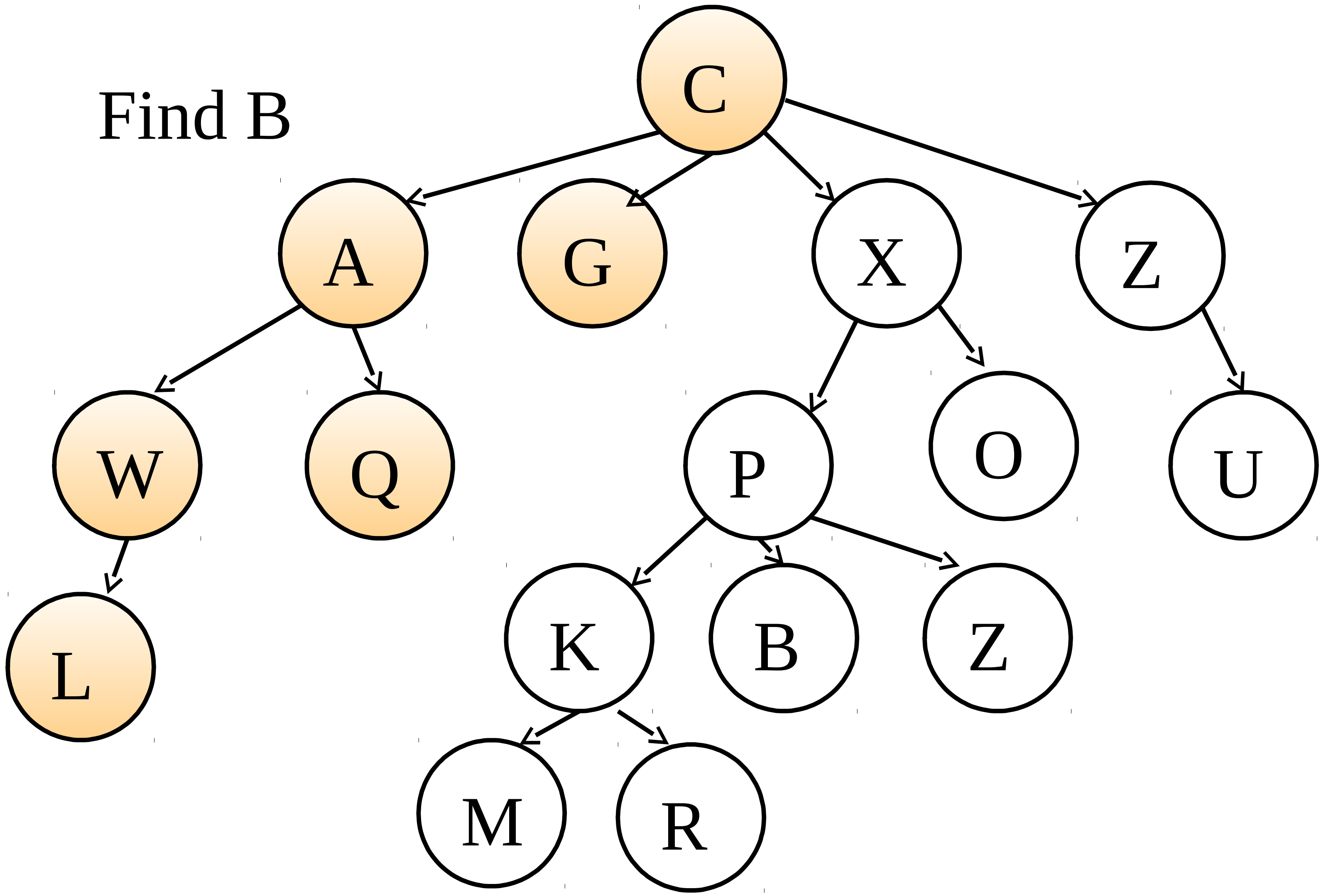
Depth First Search of Tree

Find B



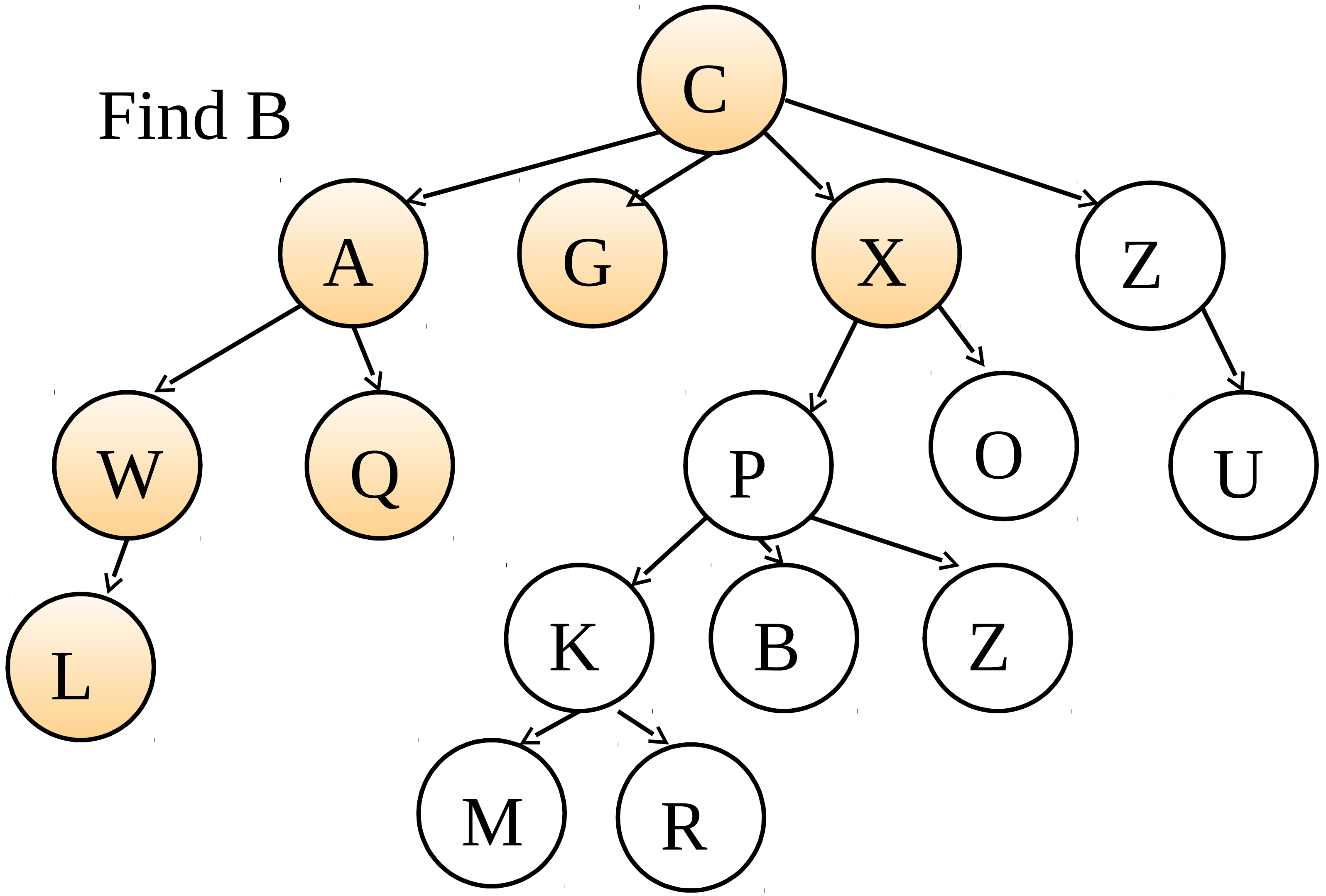
Depth First Search of Tree

Find B



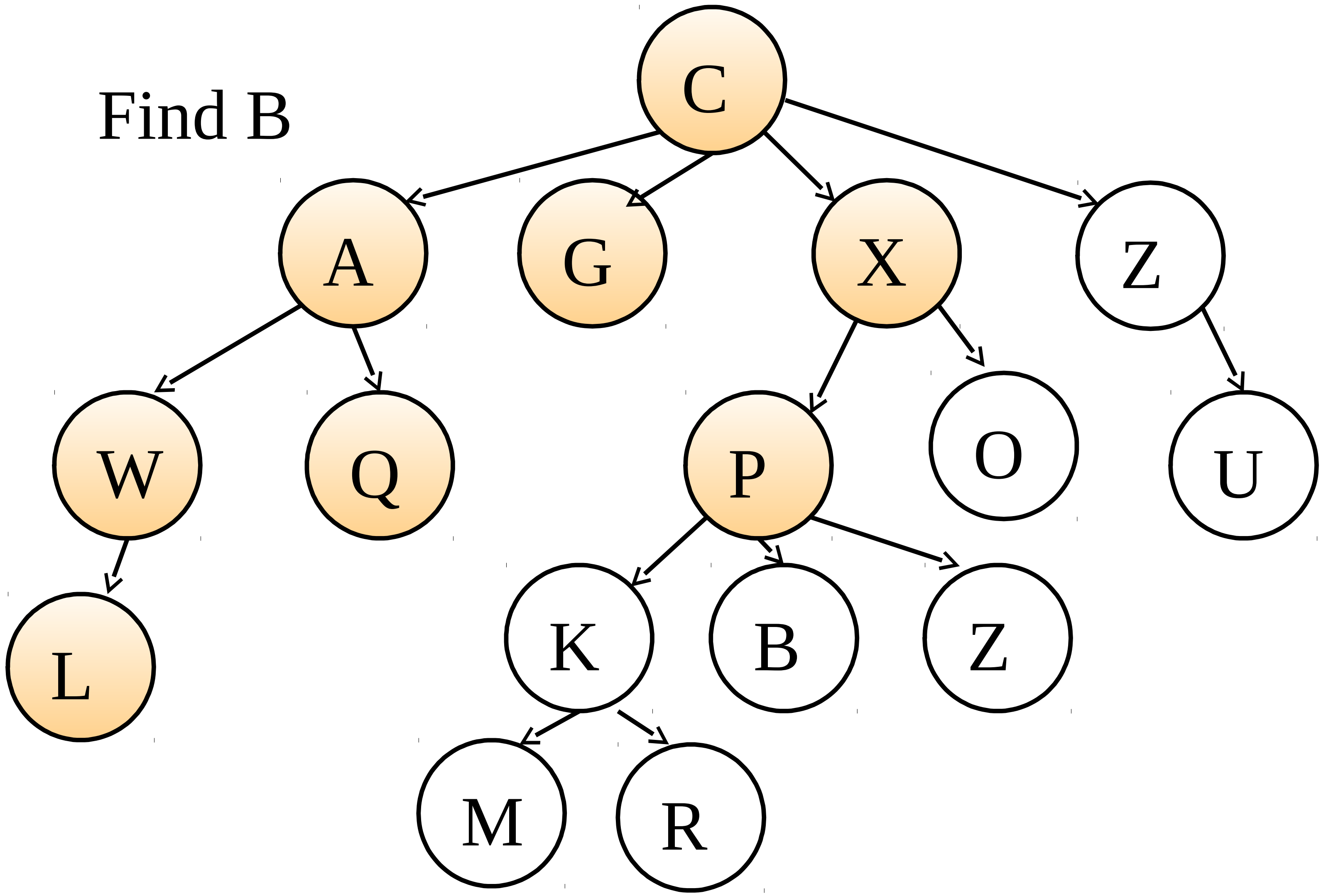
Depth First Search of Tree

Find B



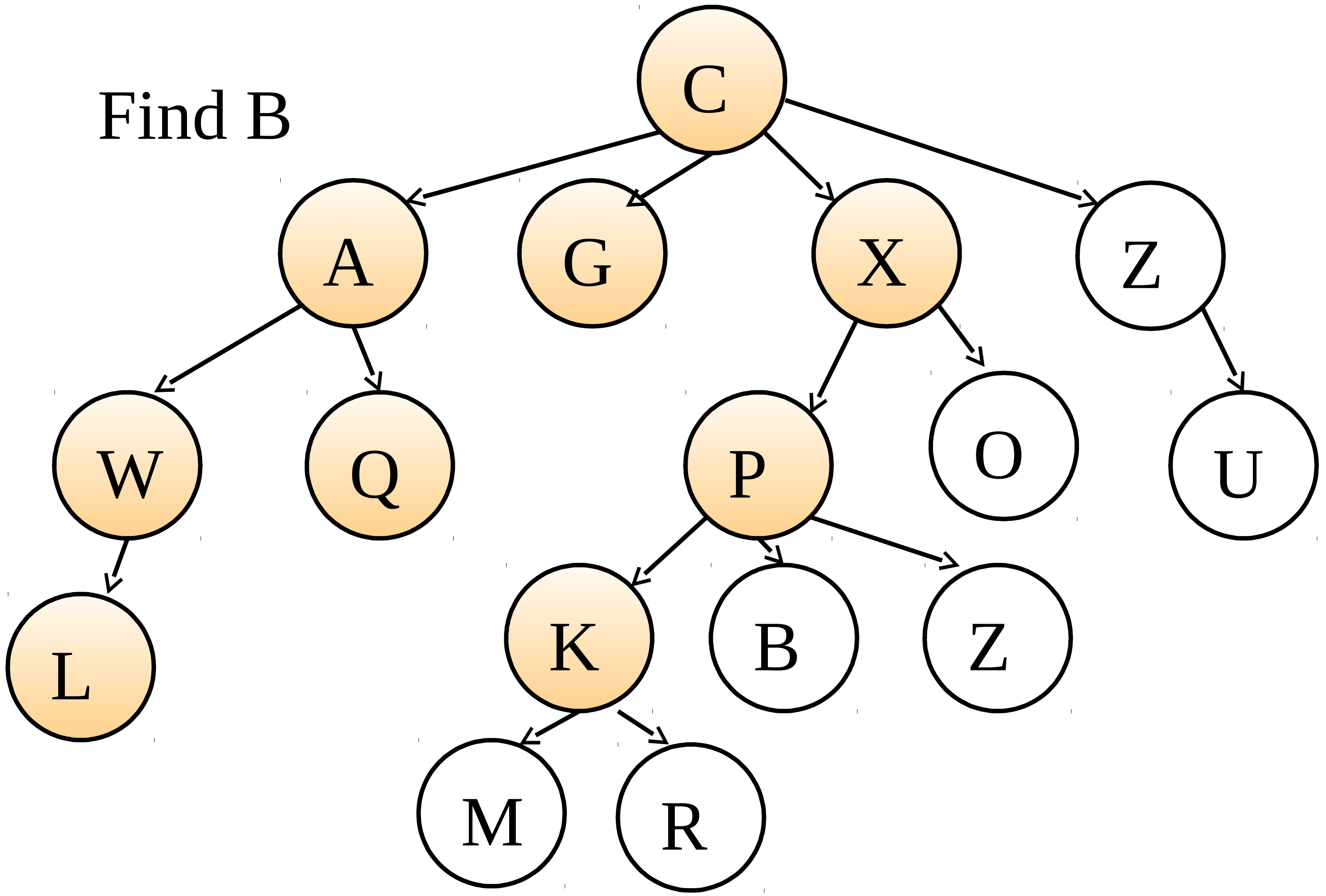
Depth First Search of Tree

Find B



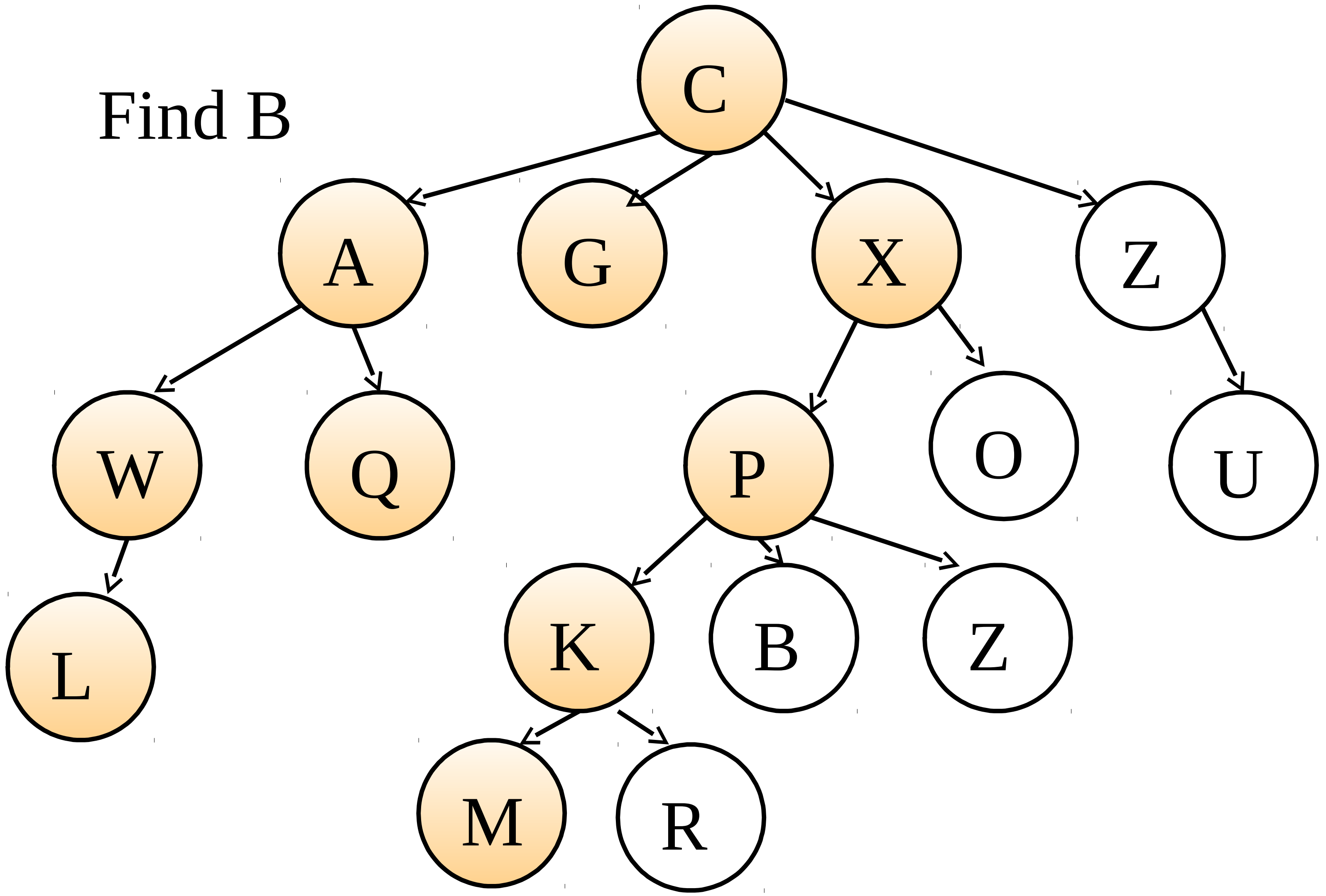
Depth First Search of Tree

Find B



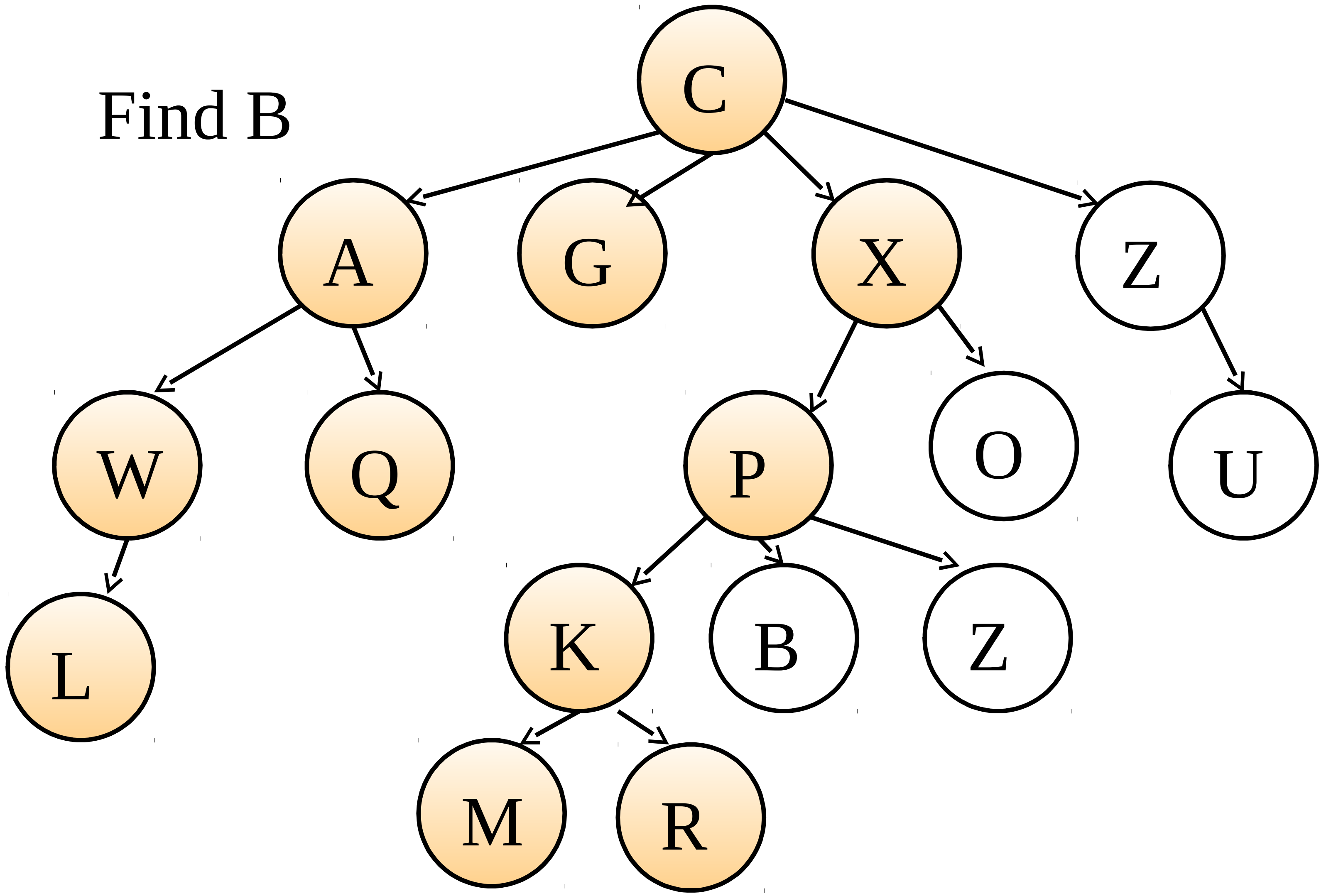
Depth First Search of Tree

Find B



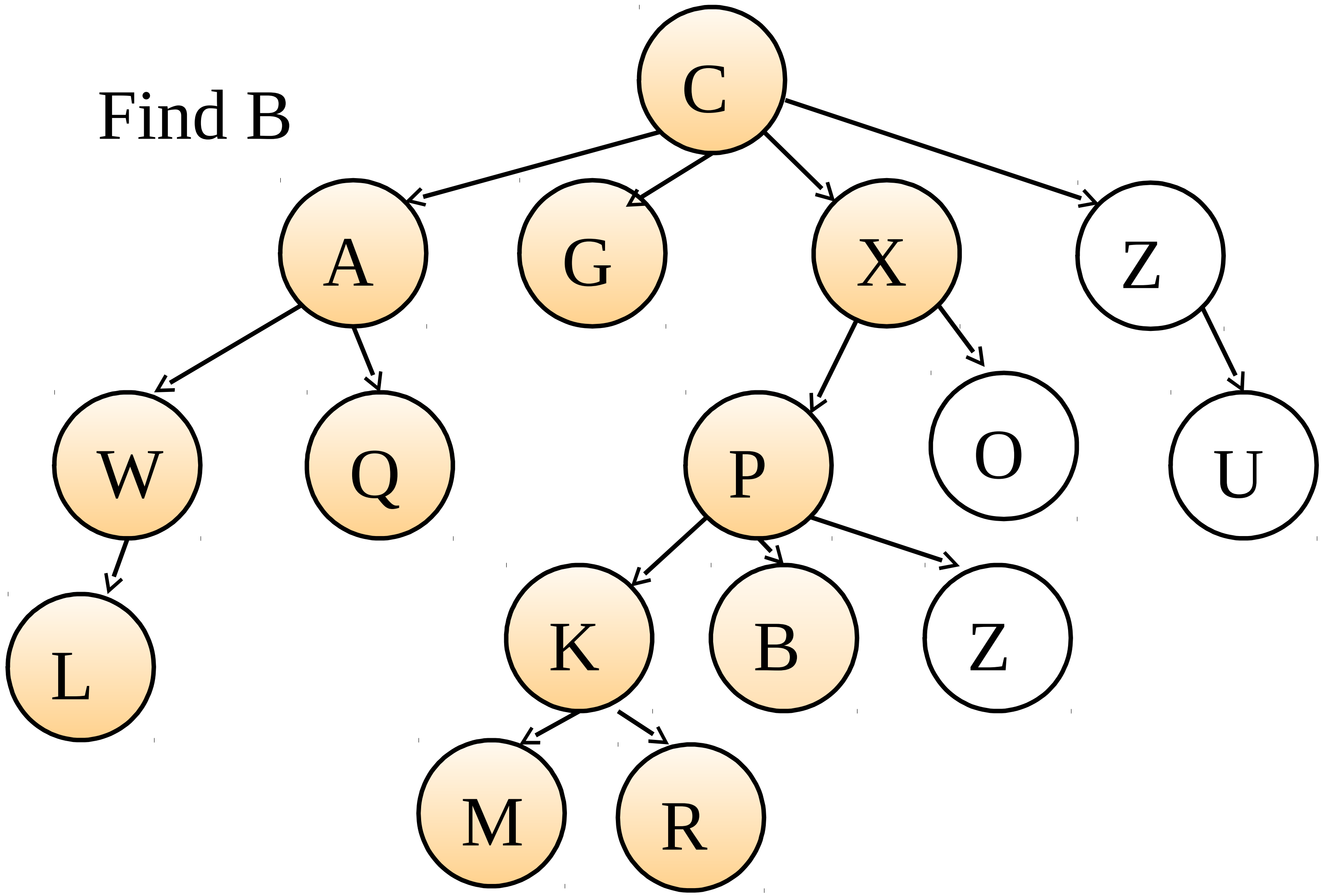
Depth First Search of Tree

Find B



Depth First Search of Tree

Find B


C
A
G
X
Z
W
Q
P
O
U
L
K
B
Z
M
R