DESIGN OF ALGORITHM

TOPIC : BINARY TREE TRAVERSAL

Team members

G NITHISH	19BCS0012
K.V ADHEE VENAYAC	19BCS0046
A NAVEEN	19BCS0009

BINARY TREE TRAVERSAL

TREE:

- A tree is finite set of one or more nodes.

BINARY TREE:

- A binary tree is an important type of tree structure which occurs very often.
- A binary tree traversal is perform many operations that we often want to perform on trees.
- A full traversal produces a linear order for the information in a tree

BINARY TREE TRAVERSAL

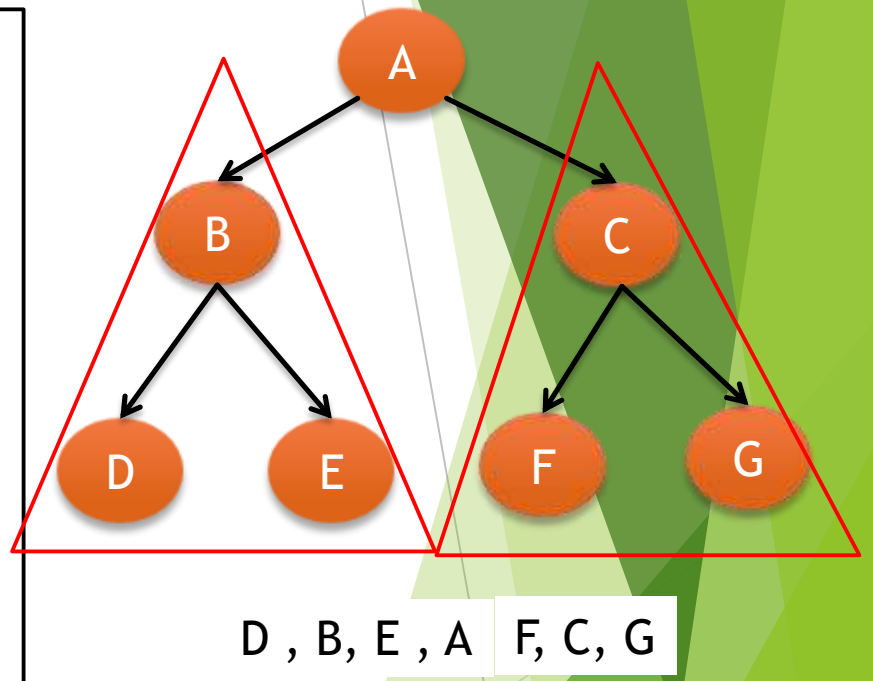
- This linear order may be familiar and useful.
- If we let L,D,R stand for moving left, printing the data, and moving right when at a node then there are six possible combinations of traversal
  - LDR
  - LRD
  - DLR
  - DRL
  - RDL
  - RLD

BINARY TREE TRAVERSAL

- If we adopt the convention that we traverse left before right then only three traversals remain:
  - LDR
  - LRD
  - DLR
- To these we assign the names as:
  - Inorder
  - Postorder
  - Preorder

Inorder Traversal(LDR)

- Informally this calls for moving down the tree towards the left until we can go to farther.
- Then you “visit” the node, move one node to the right and continue again.
- If you cannot move to the right, go back one more node.
- A precise way of describing this traversal is to write it as a recursive procedure



Algorithm Inorder Traversal:

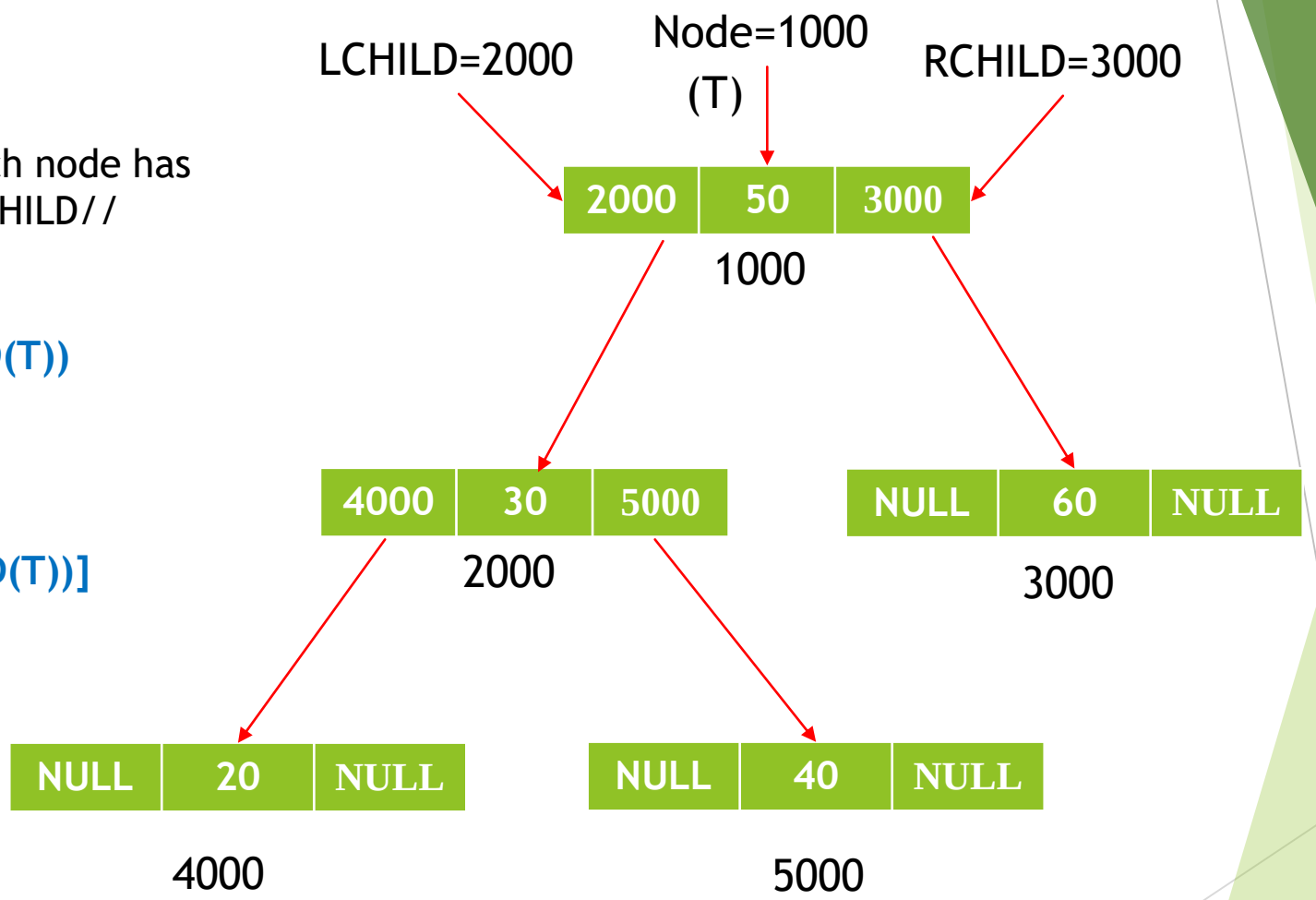
```
INORDER(T)

//T is a binary tree where each node has three fields L-
CHILD,DATA,R-CHILD//

        If (T-> L-CHILD ) then
              call INORDER(LCHILD(T))

        Print (DATA(T))

        If (T-> R-CHILD ) then
              call(INORDER(RCHILD(T))]

End INORDER
```

Application of In-order

- **In-order traversal** is very commonly used in binary search trees

Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

call INORDER(LCHILD(T))

Print (DATA(T))

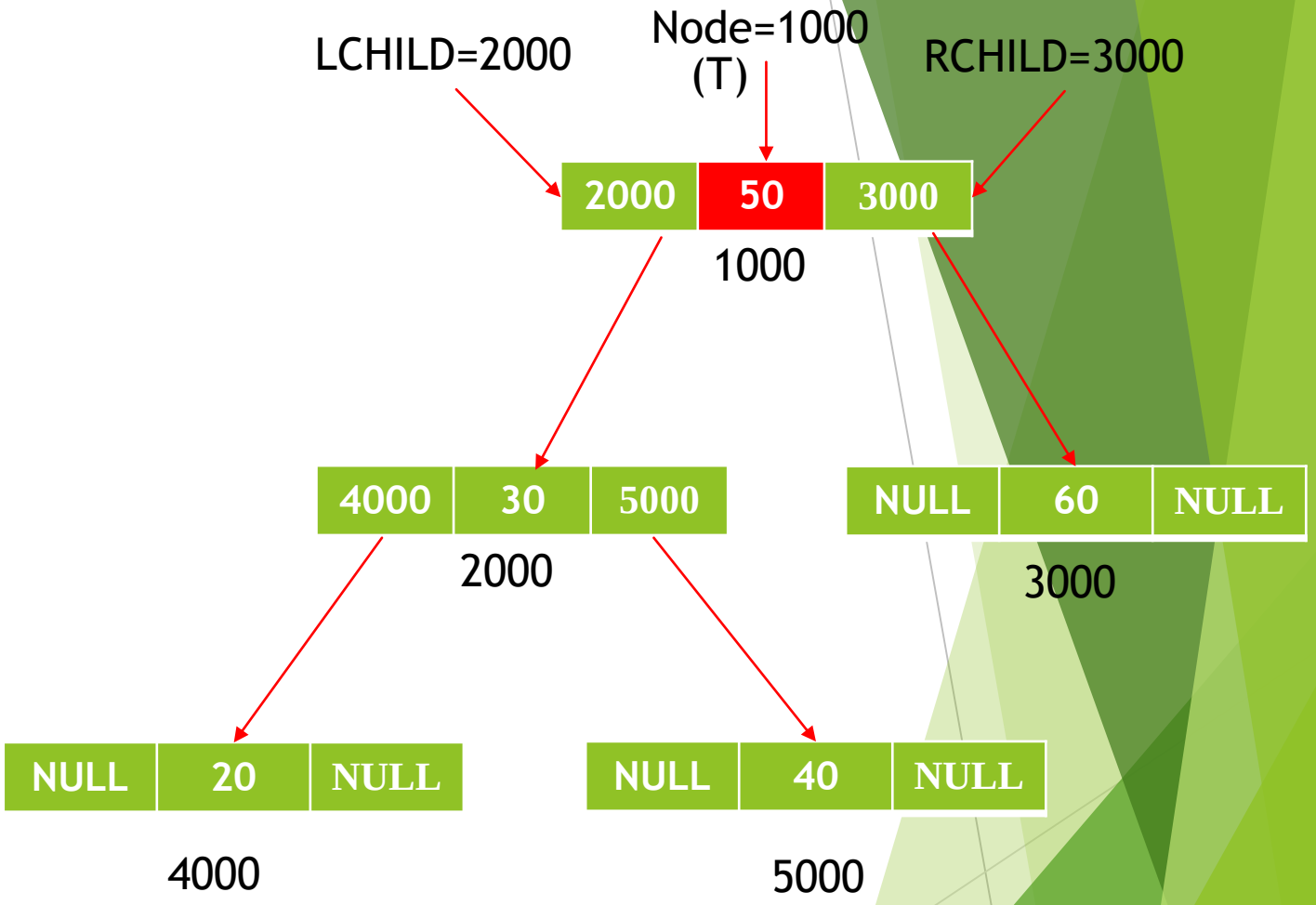
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER

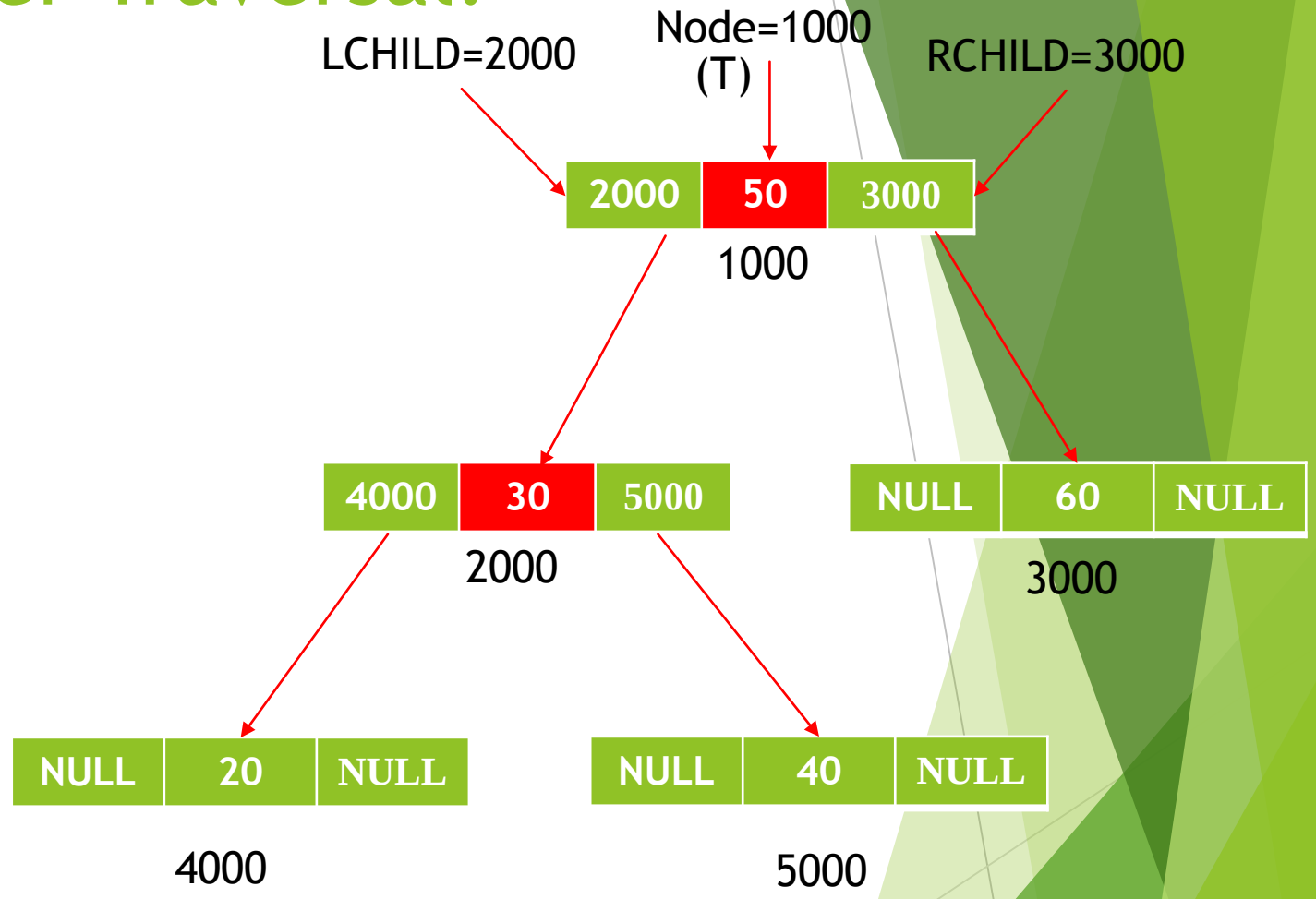
INORDER:

```
Inorder(t =1000)
{
        if(2000)
        inorder(2000)
```



Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



INORDER:

```
Inorder(t =1000)
{
        if(2000)
        inorder(2000)
```

```
Inorder(t =2000)
{
        if(4000)
        inorder(4000)
```

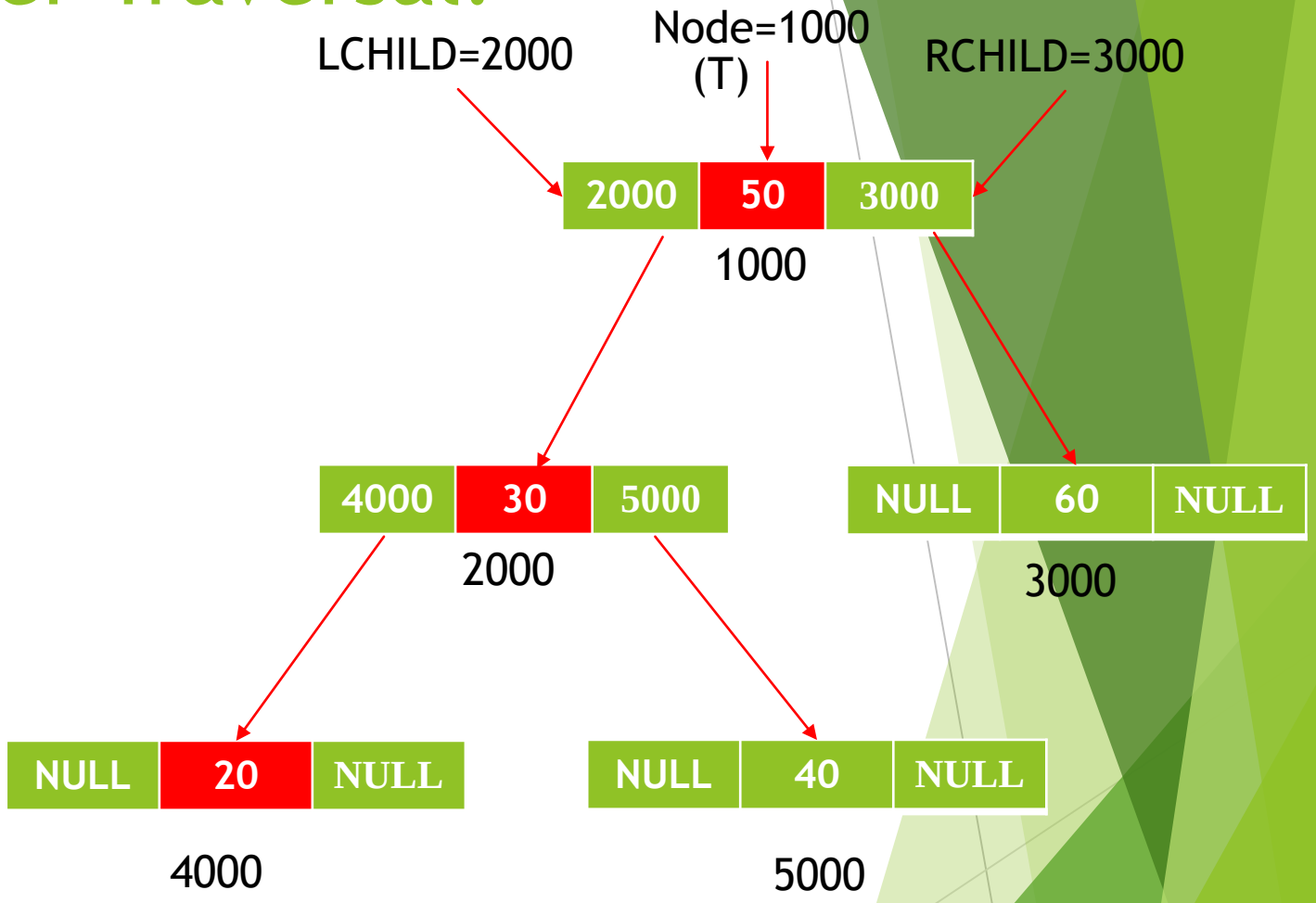
Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

```
If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
Print (DATA(T))
If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]
```

End INORDER



INORDER:

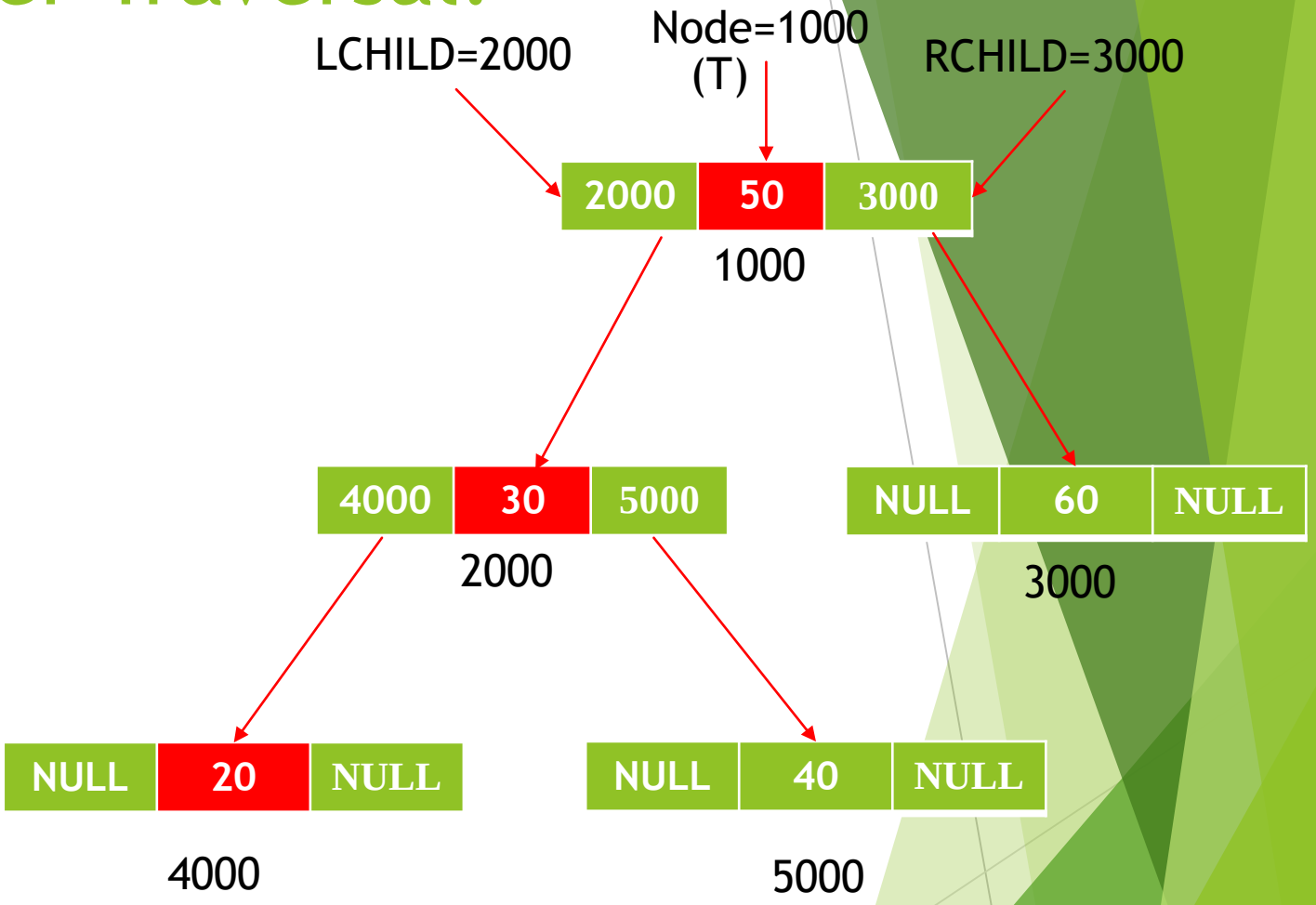
```
Inorder(t =1000)
{
        if(2000)
        inorder(2000)
```

```
Inorder(t =2000)
{
        if(4000)
        inorder(4000)
```

```
Inorder(t =4000)
{
        if(NULL)
```

Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



INORDER:

```
Inorder(t =1000)
{
        if(2000)
        inorder(2000)
```

```
Inorder(t =2000)
{
        if(4000)
        inorder(4000)
```

```
Inorder(t =4000)
{
        if(NULL)
        print(20)
```

Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

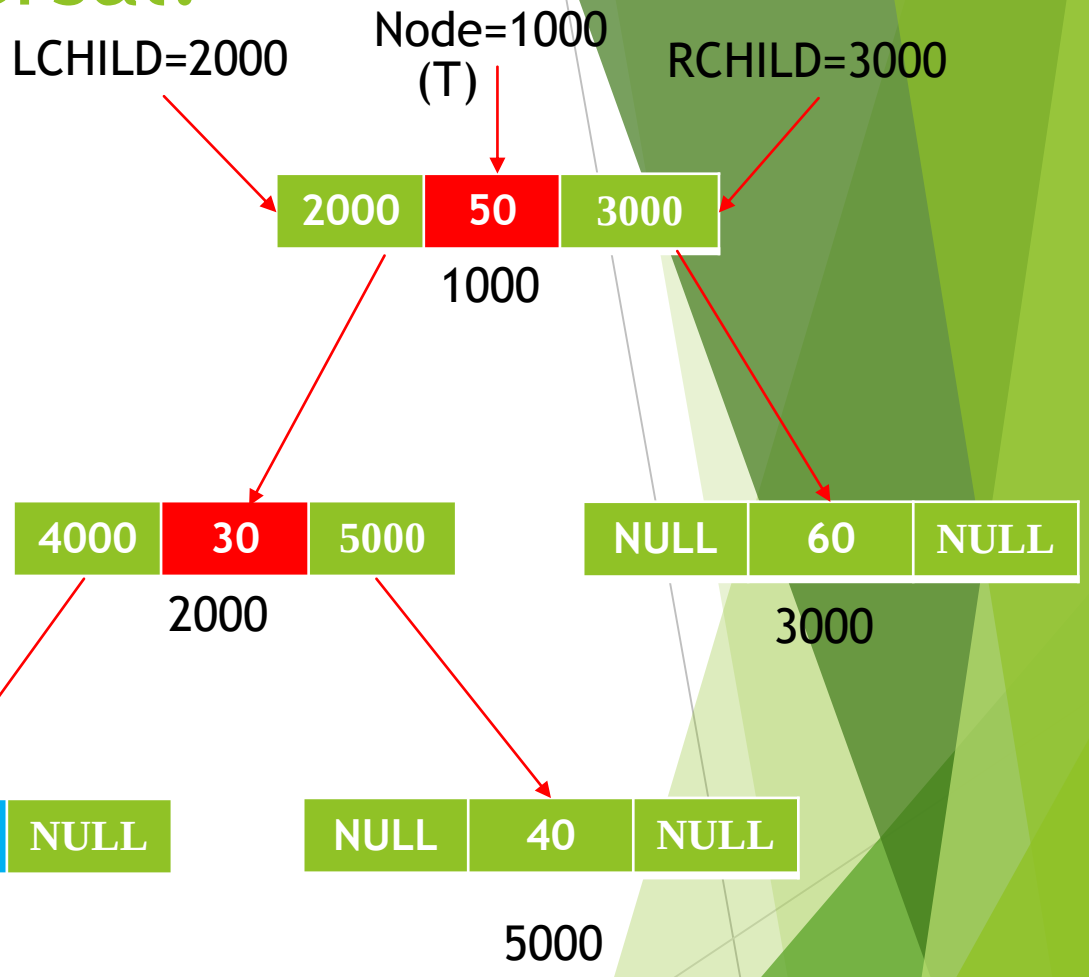
call INORDER(LCHILD(T))

Print (DATA(T))

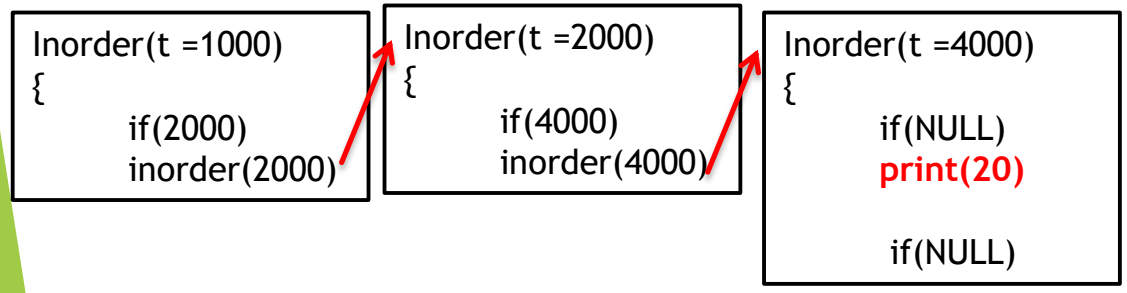
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER

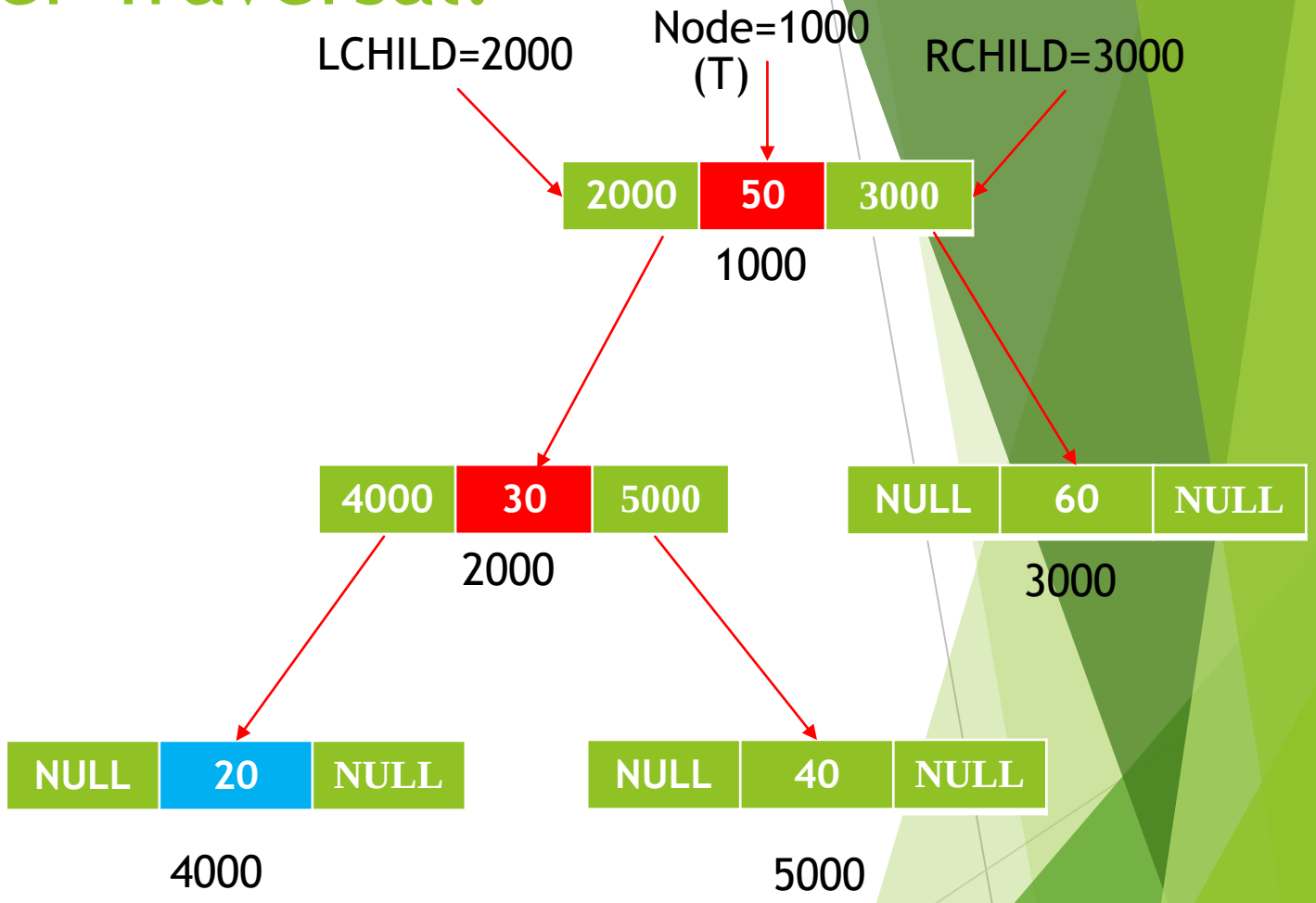


INORDER: 20

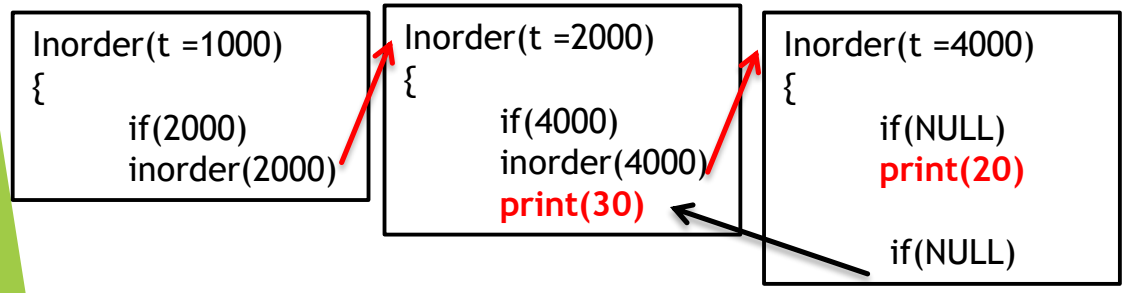


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```

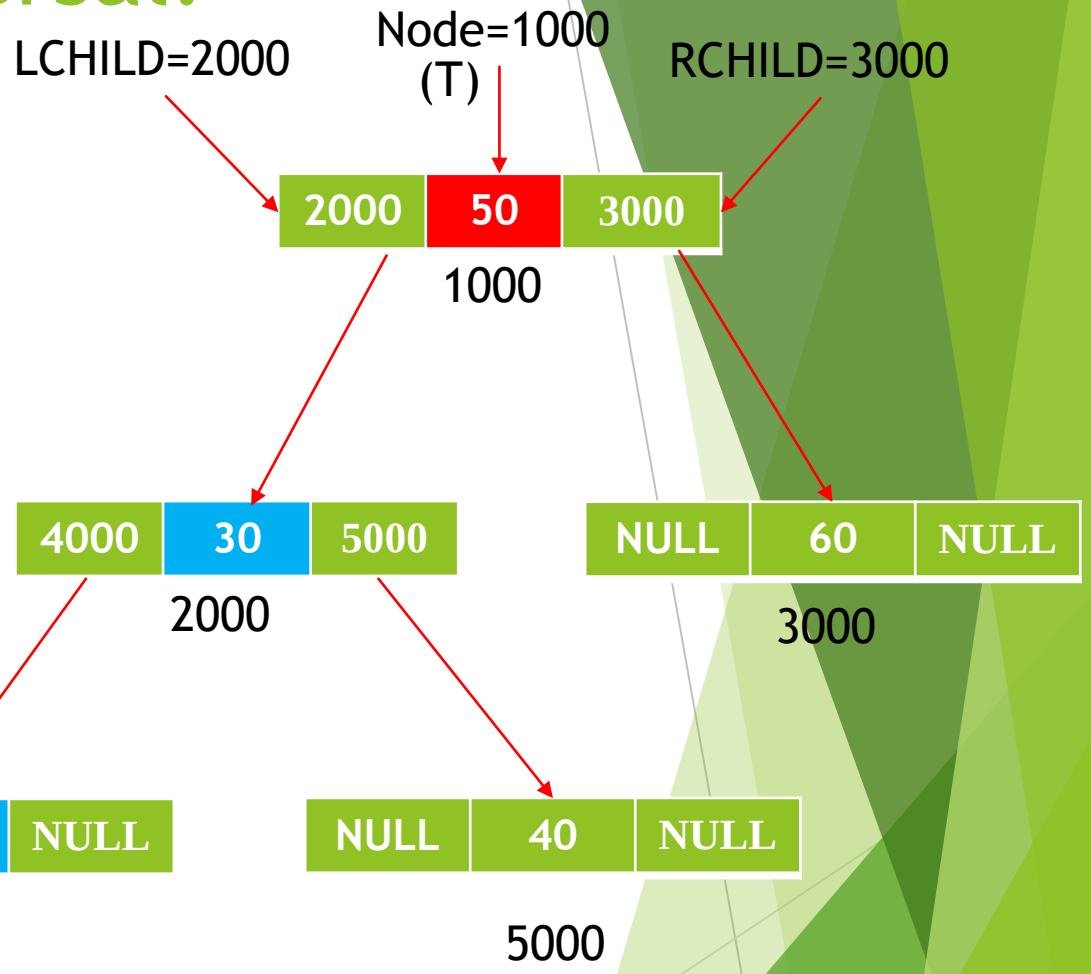


INORDER: 20

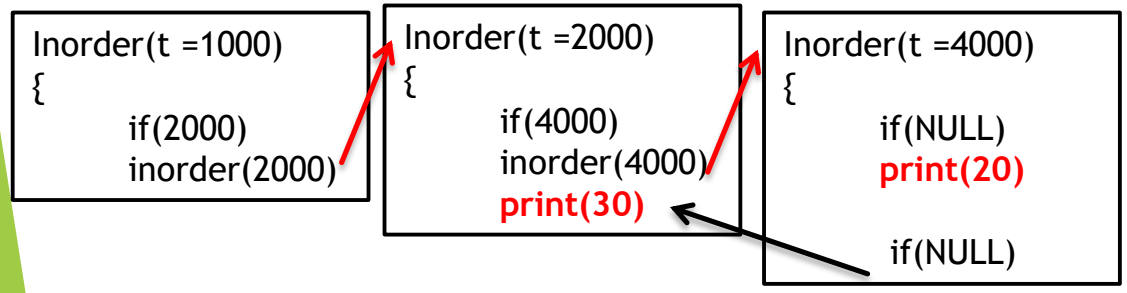


Algorithm Inorder Traversal:

```
INORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then

        call INORDER(LCHILD(T))

    Print (DATA(T))

    If (T-> R-CHILD ) then

        call(INORDER(RCHILD(T))]


End INORDER
```

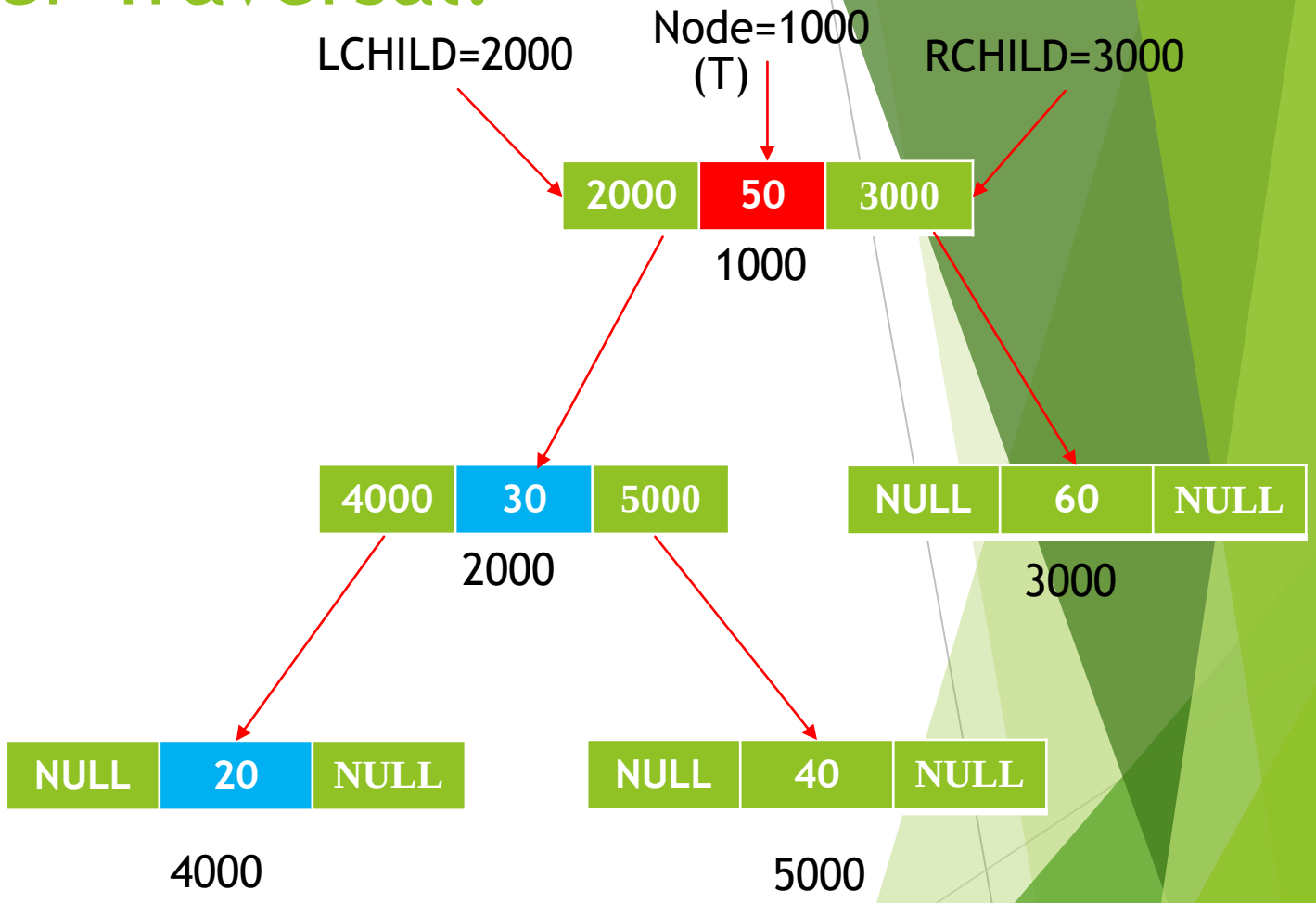


INORDER: 20, 30

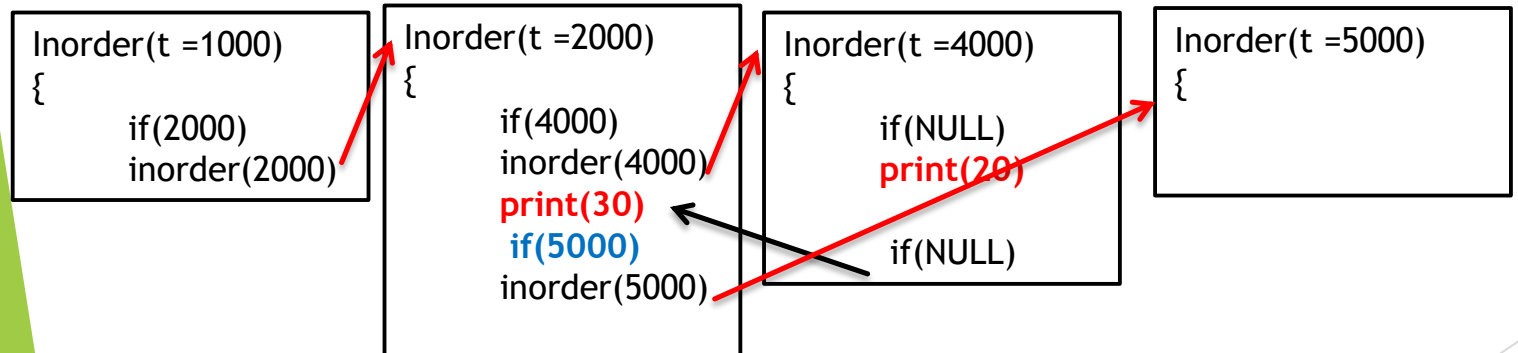


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```

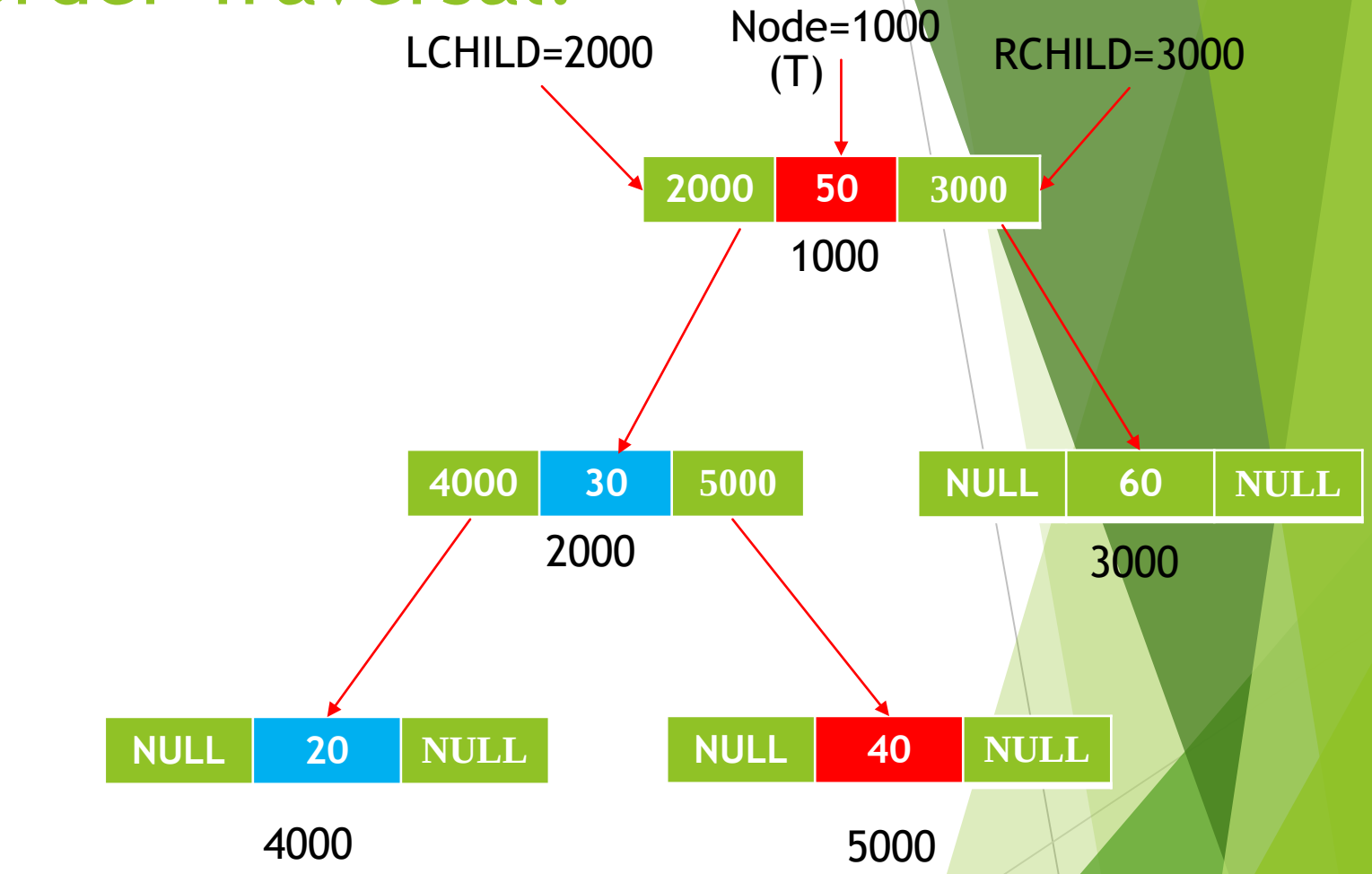


INORDER: 20, 30

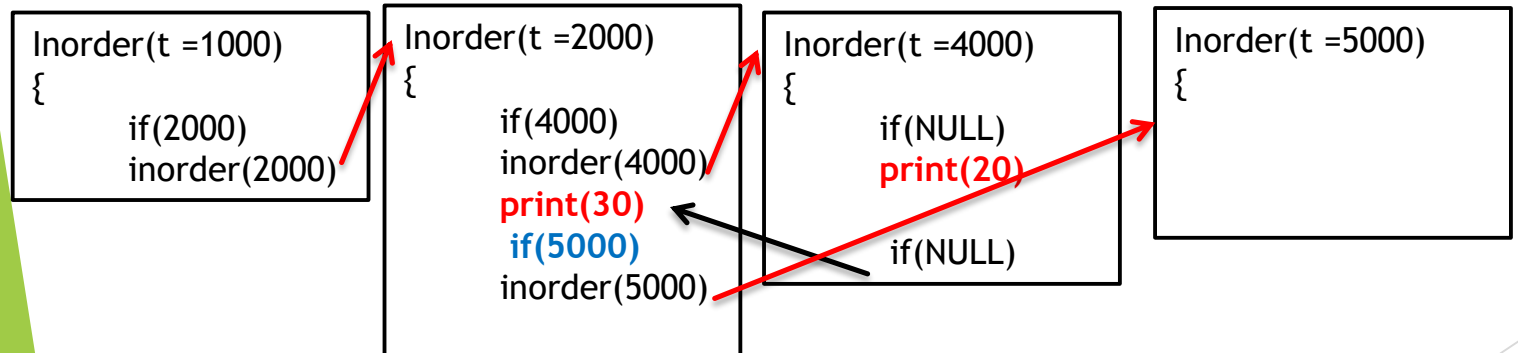


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



INORDER: 20, 30



Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

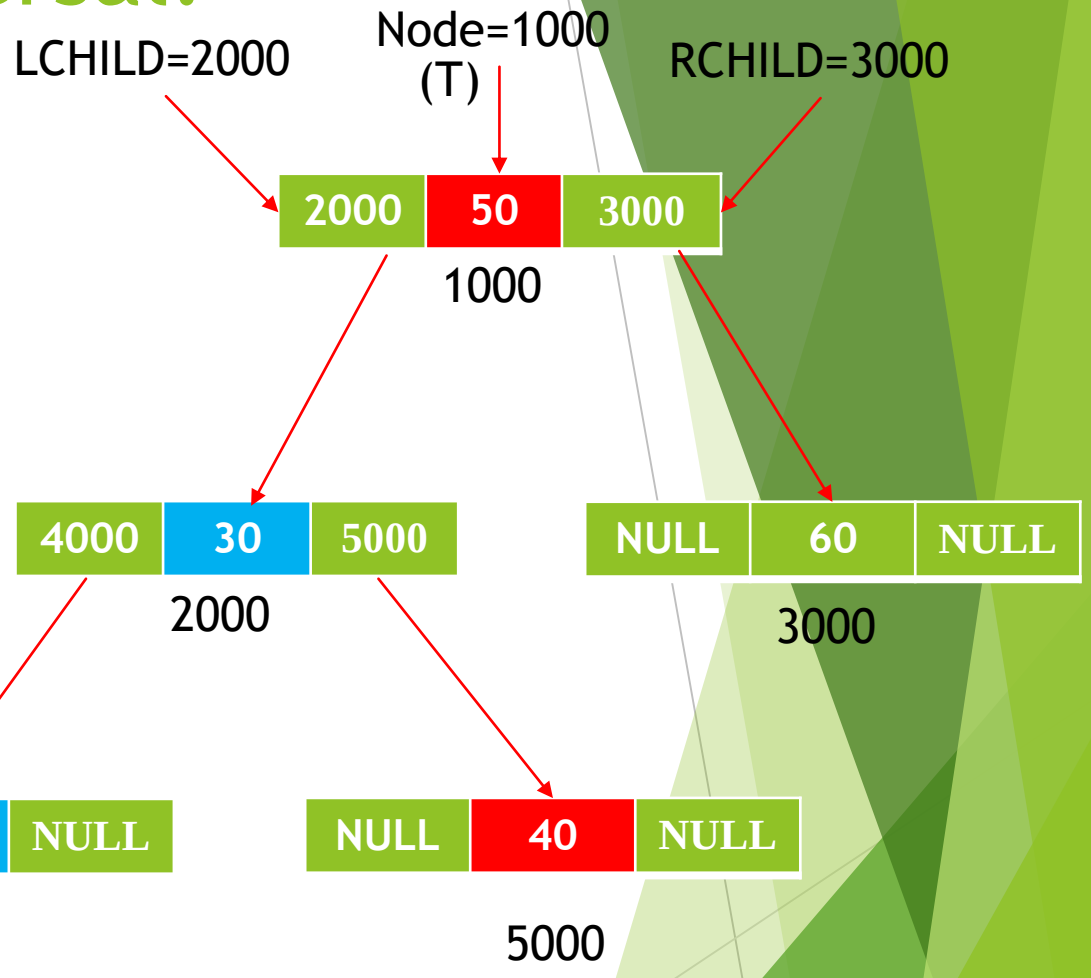
call INORDER(LCHILD(T))

Print (DATA(T))

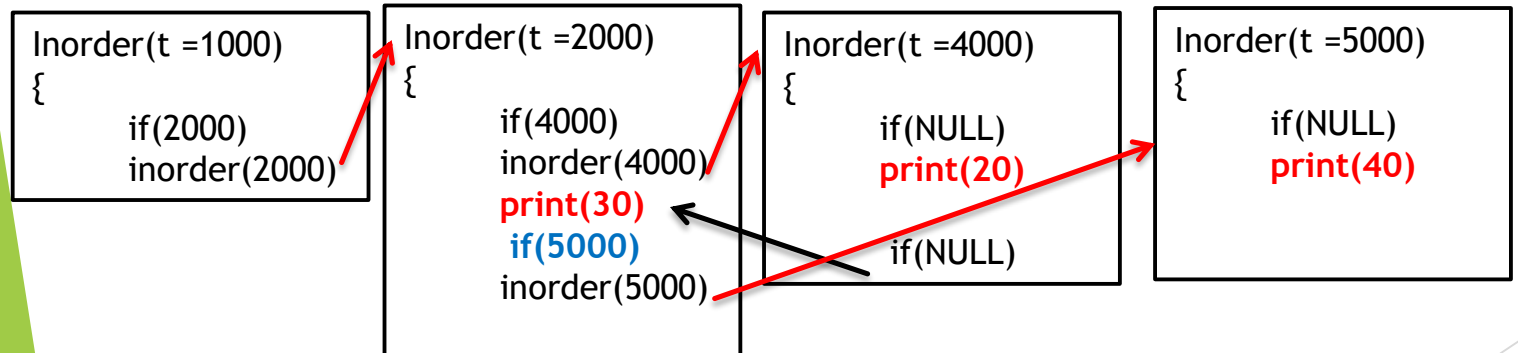
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER

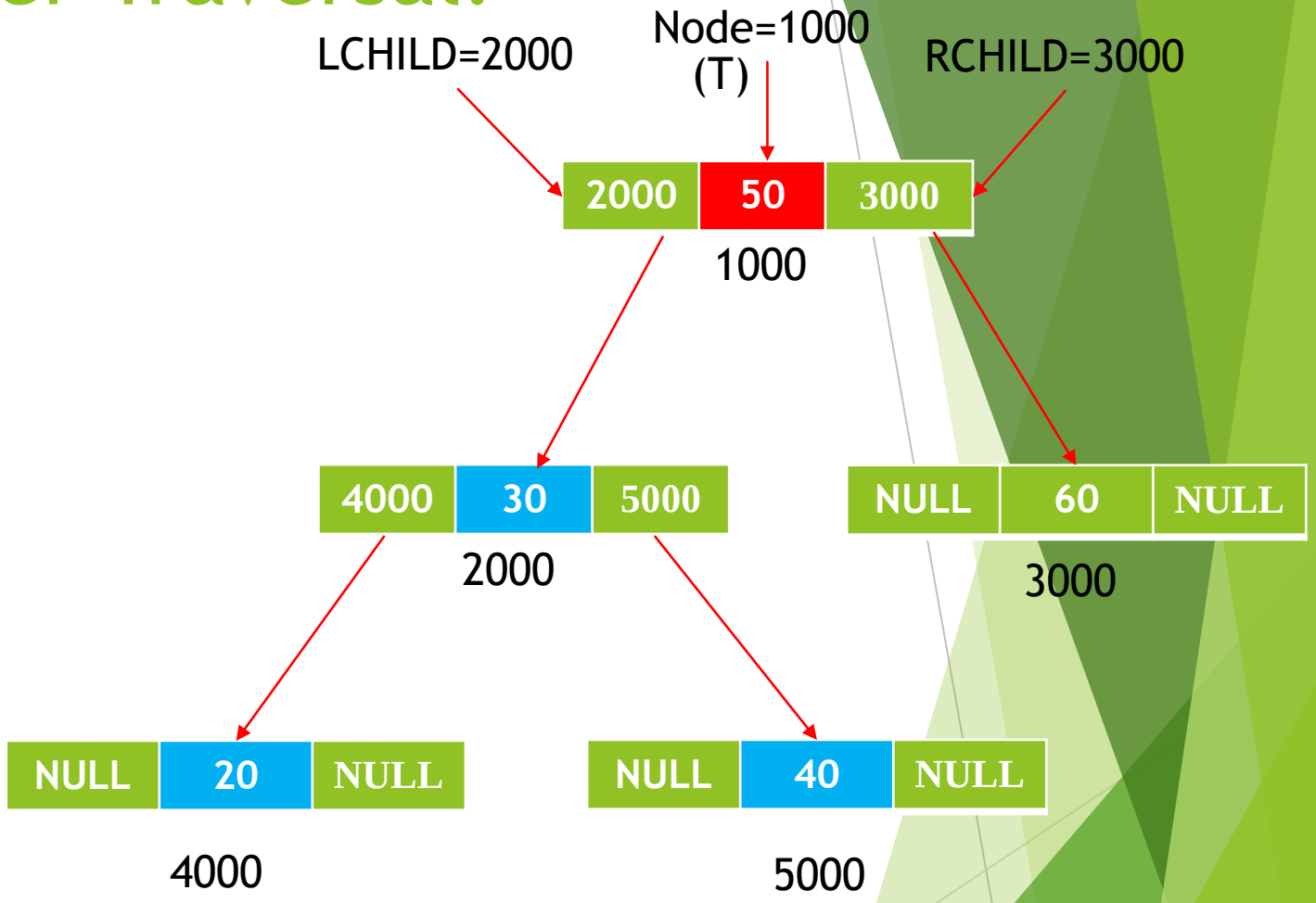


INORDER: 20, 30

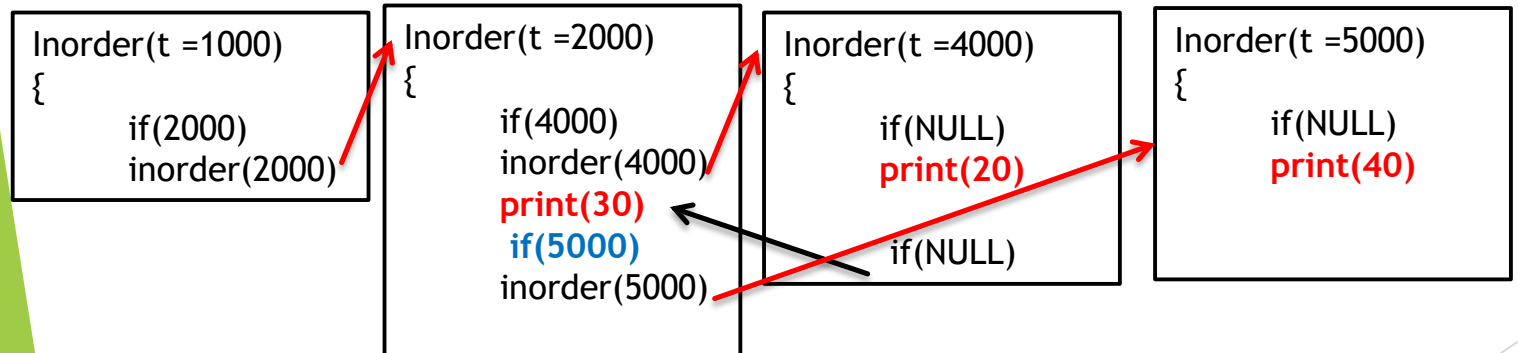


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



INORDER: 20, 30, 40



Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

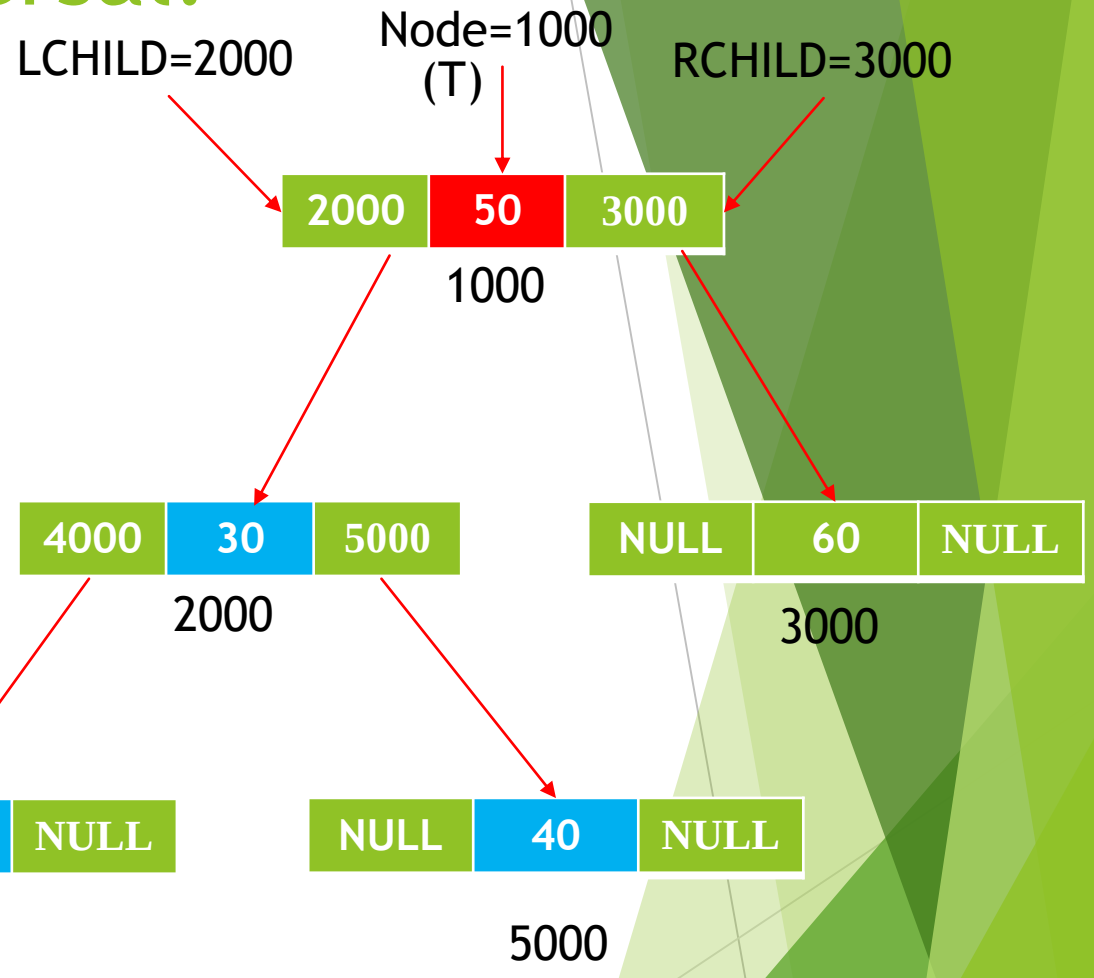
call INORDER(LCHILD(T))

Print (DATA(T))

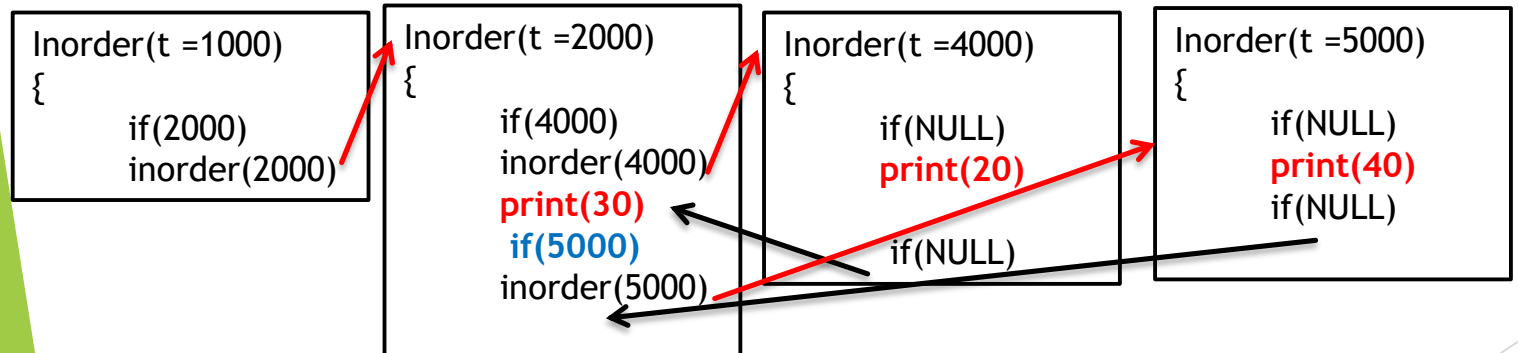
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER

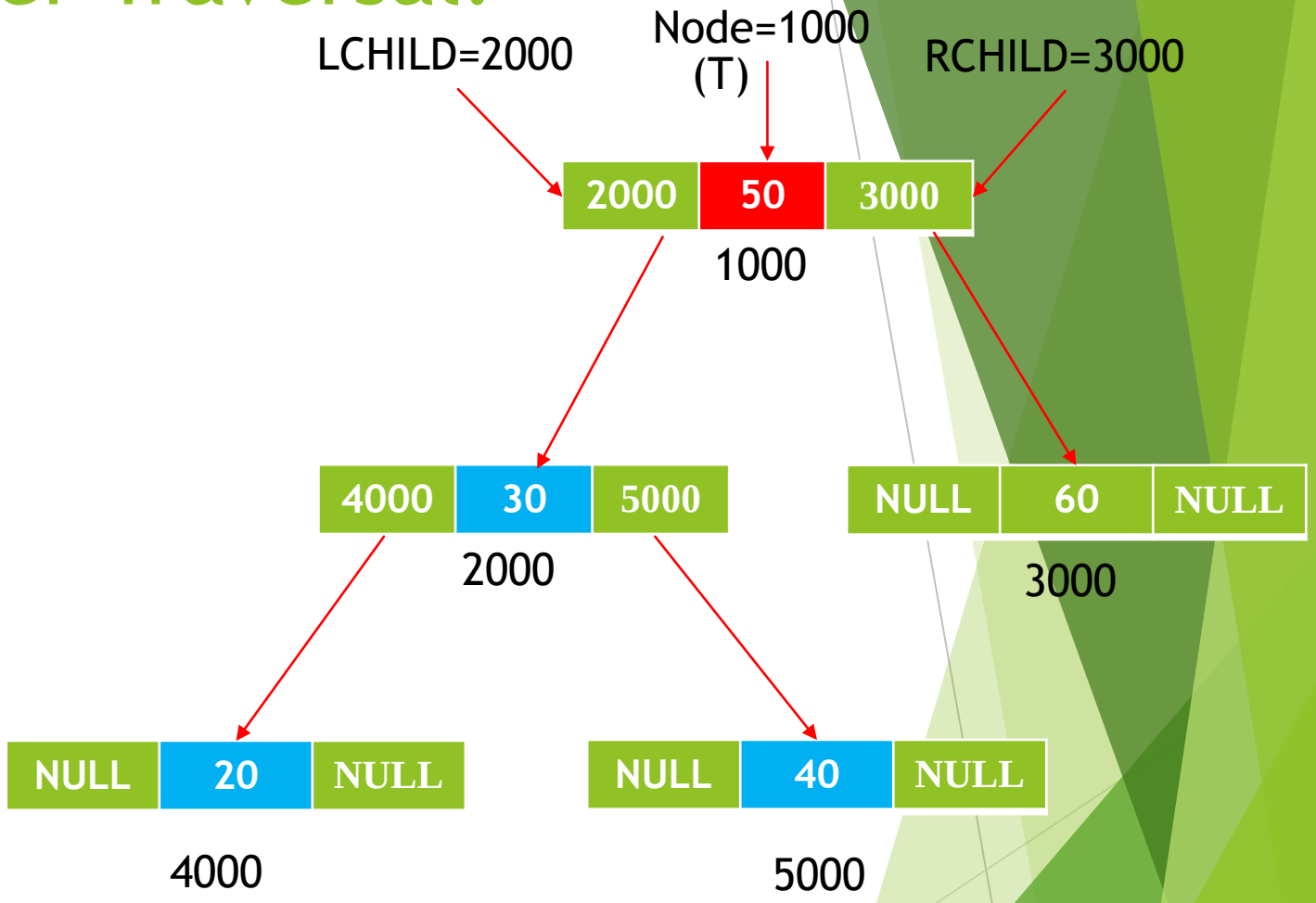


INORDER: 20, 30, 40

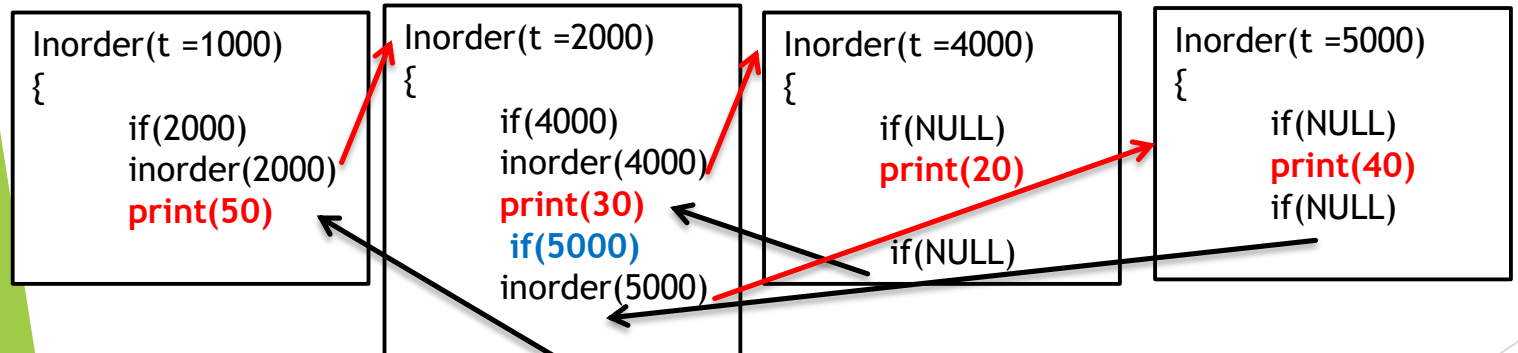


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```

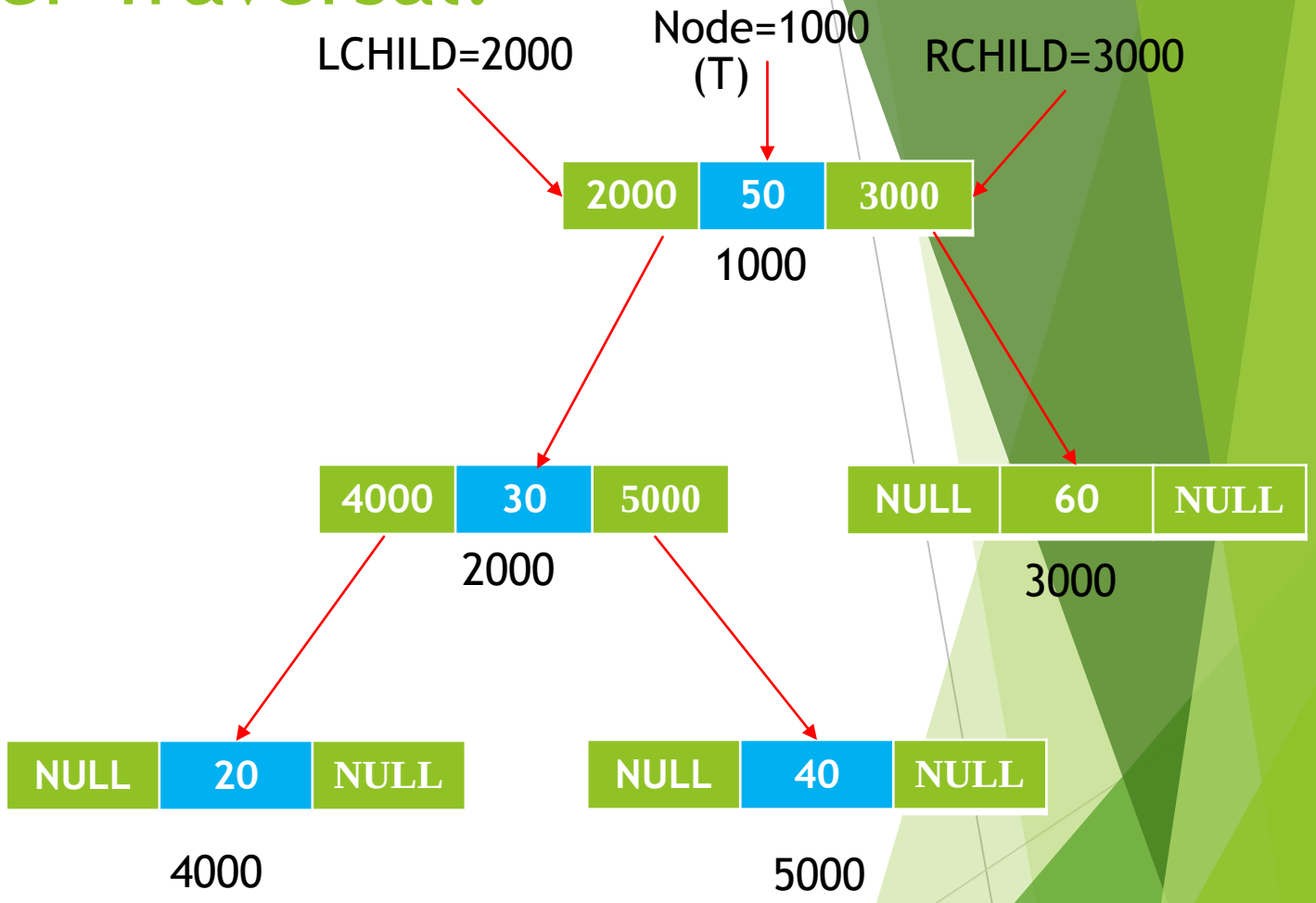


INORDER: 20, 30, 40

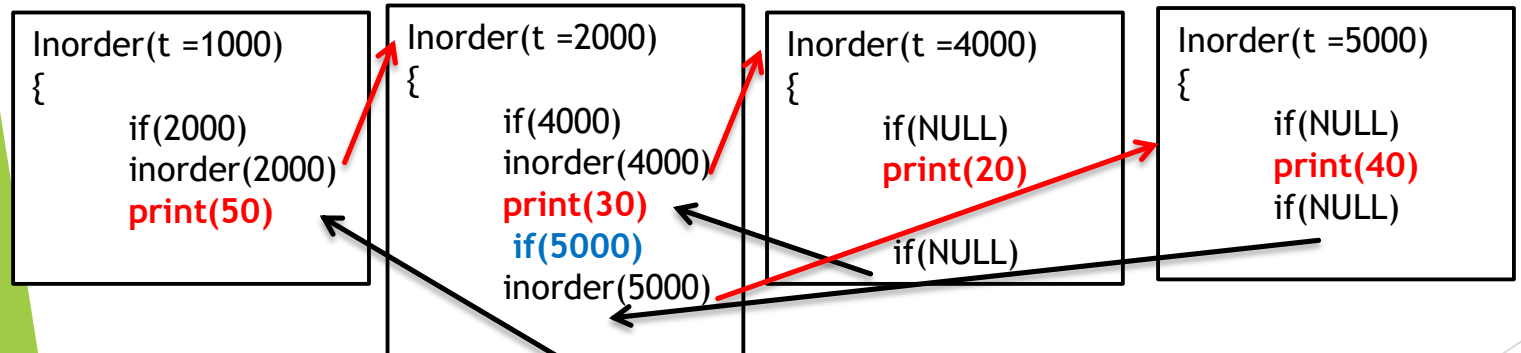


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]

End INORDER
```



INORDER: 20, 30, 40, 50



Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

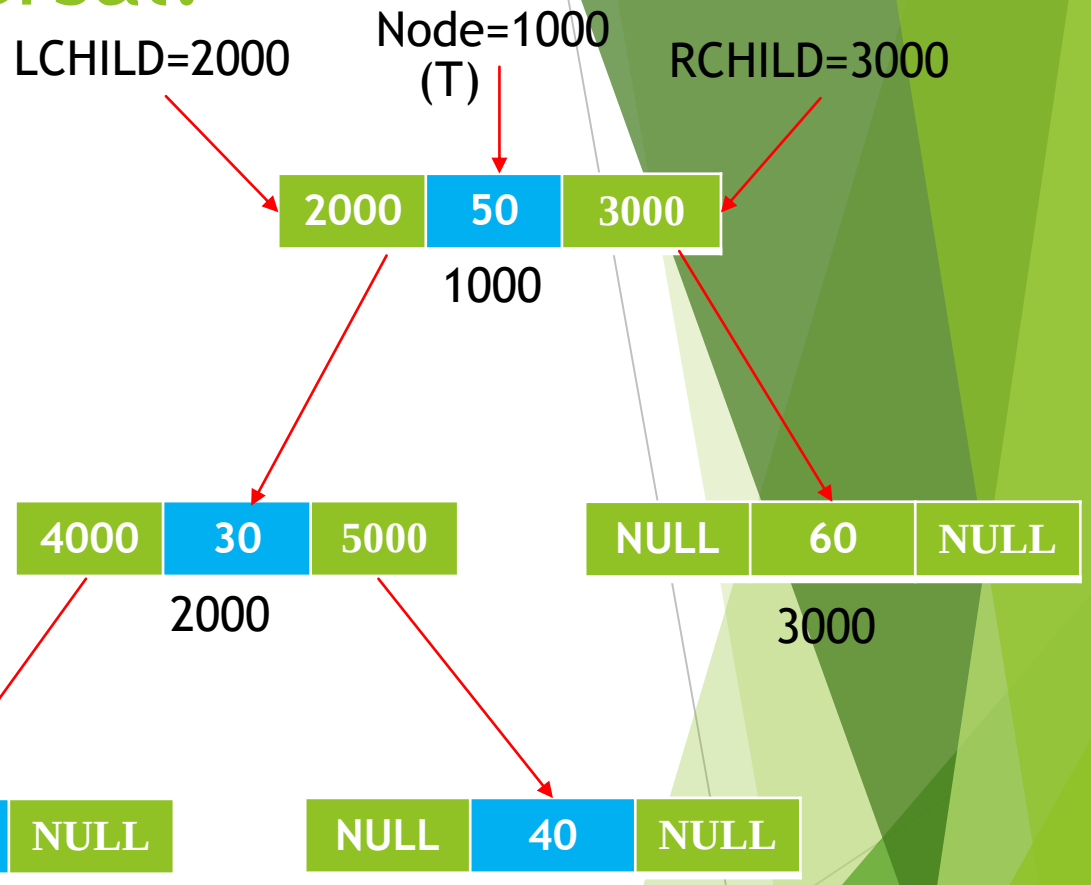
call INORDER(LCHILD(T))

Print (DATA(T))

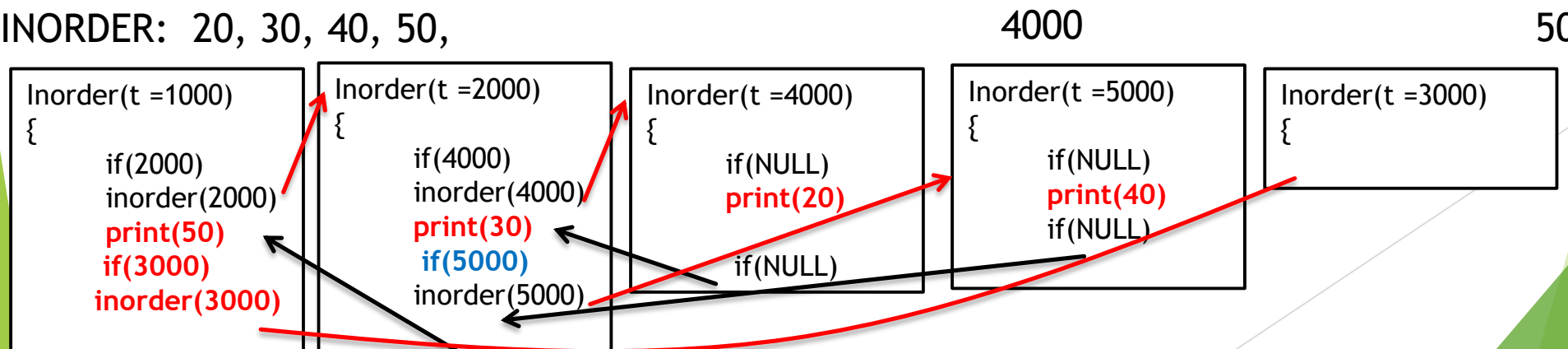
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER



INORDER: 20, 30, 40, 50,



Algorithm Inorder Traversal:

INORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

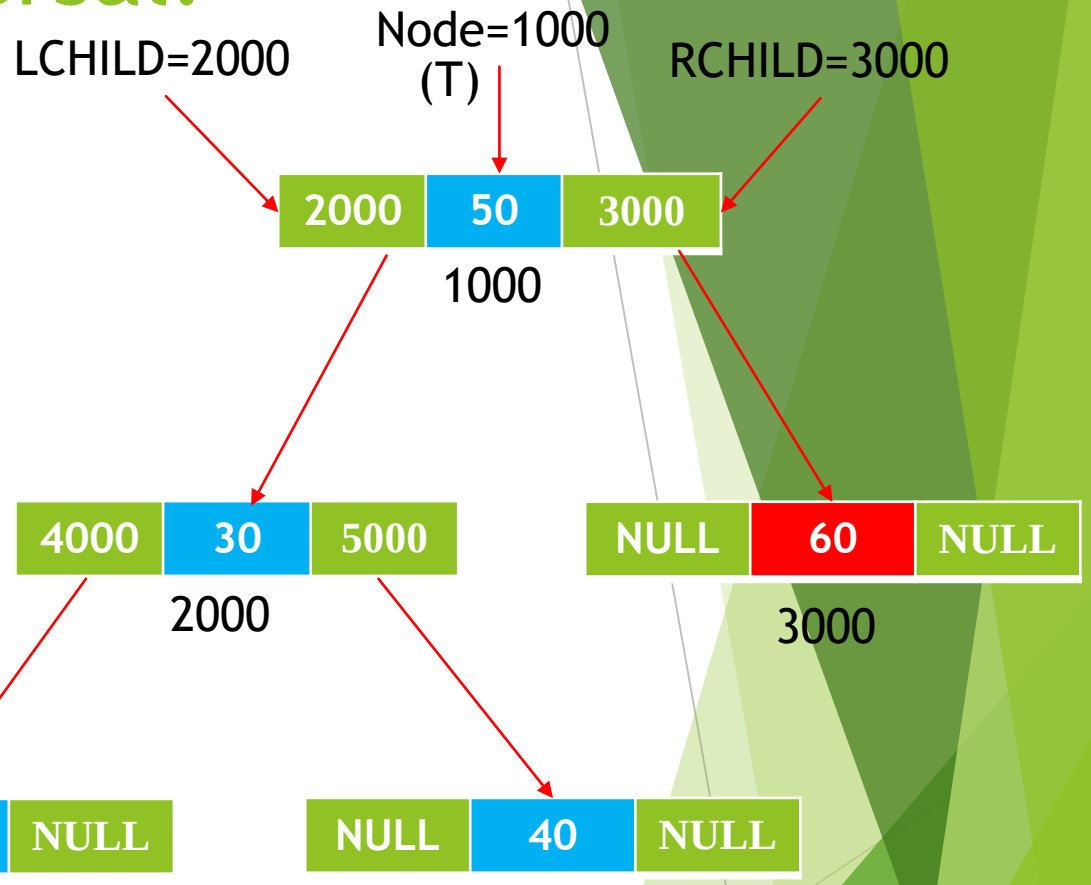
call INORDER(LCHILD(T))

Print (DATA(T))

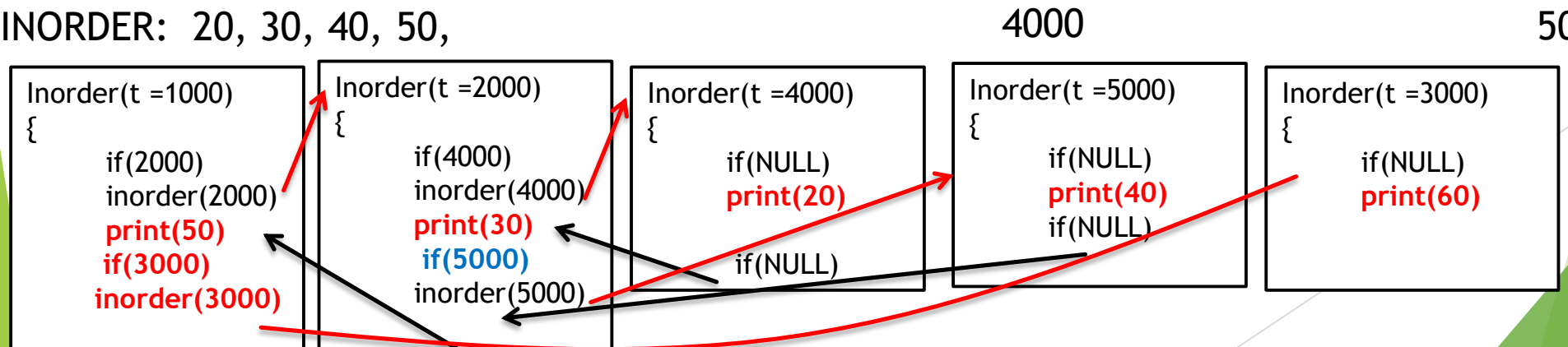
If (T-> R-CHILD) then

call(INORDER(RCHILD(T))]

End INORDER

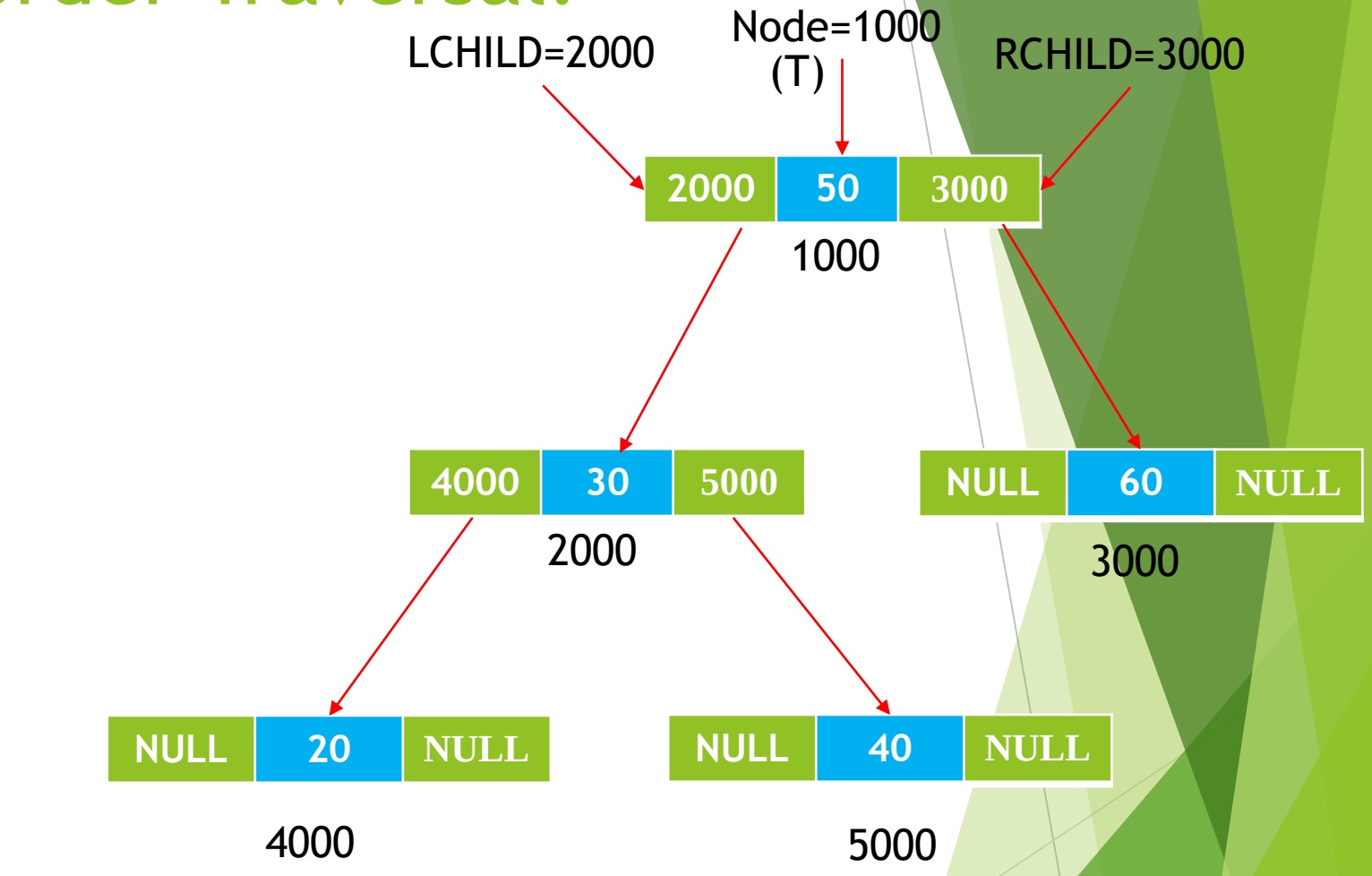


INORDER: 20, 30, 40, 50,

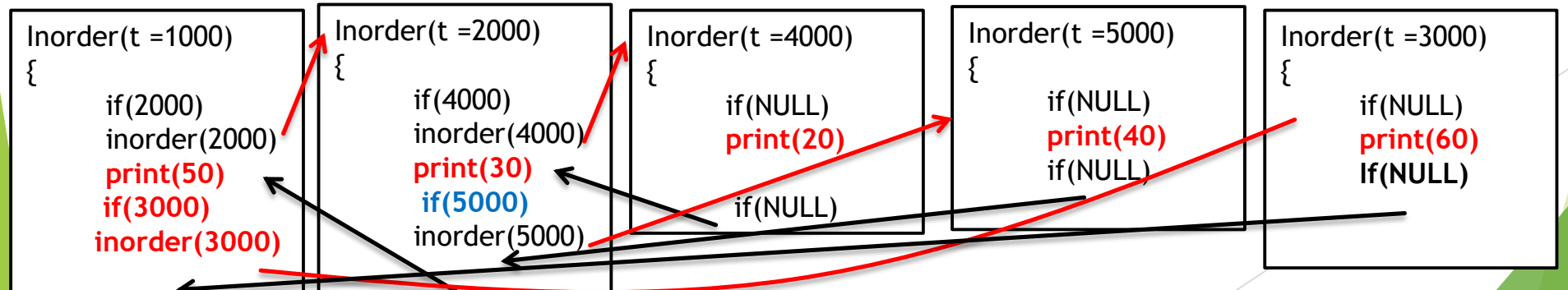


Algorithm Inorder Traversal:

```
INORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//
    If (T-> L-CHILD ) then
        call INORDER(LCHILD(T))
    Print (DATA(T))
    If (T-> R-CHILD ) then
        call(INORDER(RCHILD(T))]
End INORDER
```



INORDER: 20, 30, 40, 50, 60

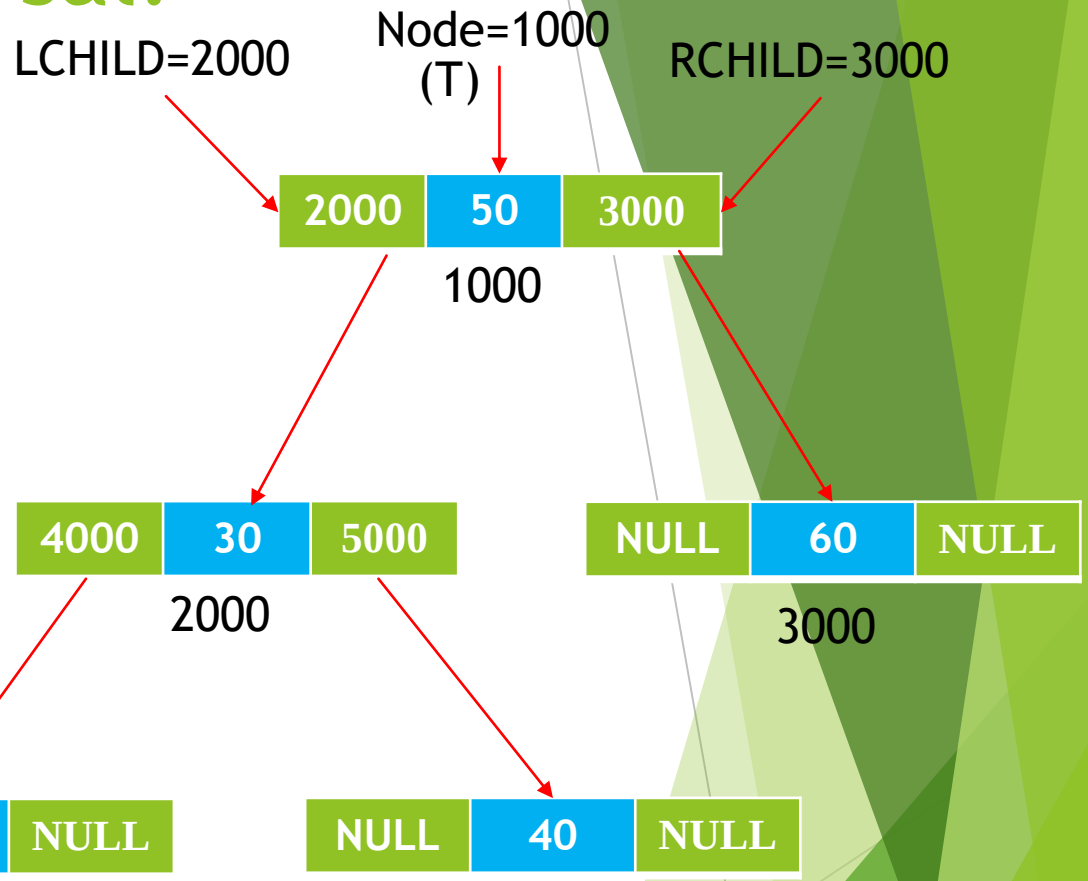


Program Inorder Traversal:

```
struct Node {
    int data;
    struct Node *left, *right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
```

```
int main()
{
    struct Node* root = new Node(50);
    root->left = new Node(30);
    root->right = new Node(60);
    root->left->left = new Node(20);
    root->left->right = new Node(40);
cout << "\nInorder traversal of binary tree is \n";
 printInorder(root);
```

```
void printInorder(struct Node*
node)
{
  if ( node ->left)
     printInorder( node->left);
   cout<< node->data << " ";
   if (node ->right)
     printInorder( node->right);
}
```



INORDER: 20, 30, 40, 50, 60

```
Inorder(t =1000)
{
    if(2000)
    inorder(2000)
    print(50)
    if(3000)
    inorder(3000)
```

```
Inorder(t =2000)
{
    if(4000)
    inorder(4000)
    print(30)
     if(5000)
    inorder(5000)
```

```
Inorder(t =4000)
{
    if(NULL)
    print(20)
    if(NULL)
```

```
Inorder(t =5000)
{
    if(NULL)
    print(40)
    if(NULL)
```

```
Inorder(t =3000)
{
    if(NULL)
    print(60)
    If(NULL)
```

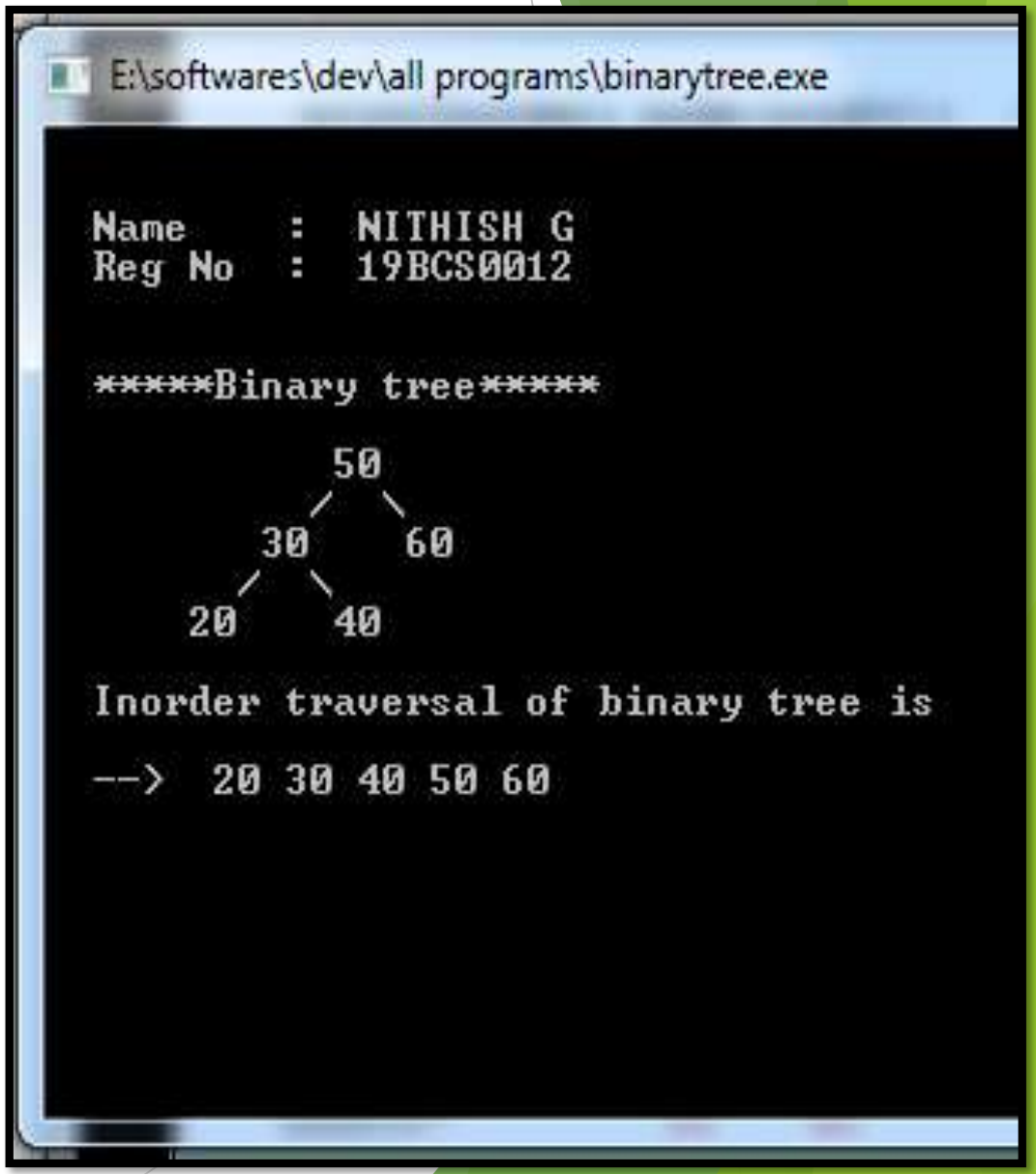
Program Inorder Traversal:

```
struct Node {
    int data;
    struct Node *left,
*right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
```

```
void printInorder(struct Node*
node)
{

 if ( node ->left)

    printInorder( node->left);

   cout<< node->data << " ";

   if (node ->right)

    printInorder( node->right);
}
```

```
int main()

{

   struct Node* root = new Node(50);

   root->left = new Node(30);

   root->right = new Node(60);

   root->left->left = new Node(20);

   root->left->right = new Node(40);

   cout << "\n\n  Name    :  NITHISH G ";

   cout << "\n  Reg No  :  19BCS0012 \n";

   cout << "\n\n  *****Binary tree*****\n\n";

    cout<<"          50            \n";

   cout<<"         / \\            \n";

   cout<<"        30   60          \n";

   cout<<"       / \\             \n";

   cout<<"      20   40          \n ";

cout << "\nInorder traversal of binary tree is \n“;

printInorder(root);
```

Output



Preoder Traversal:

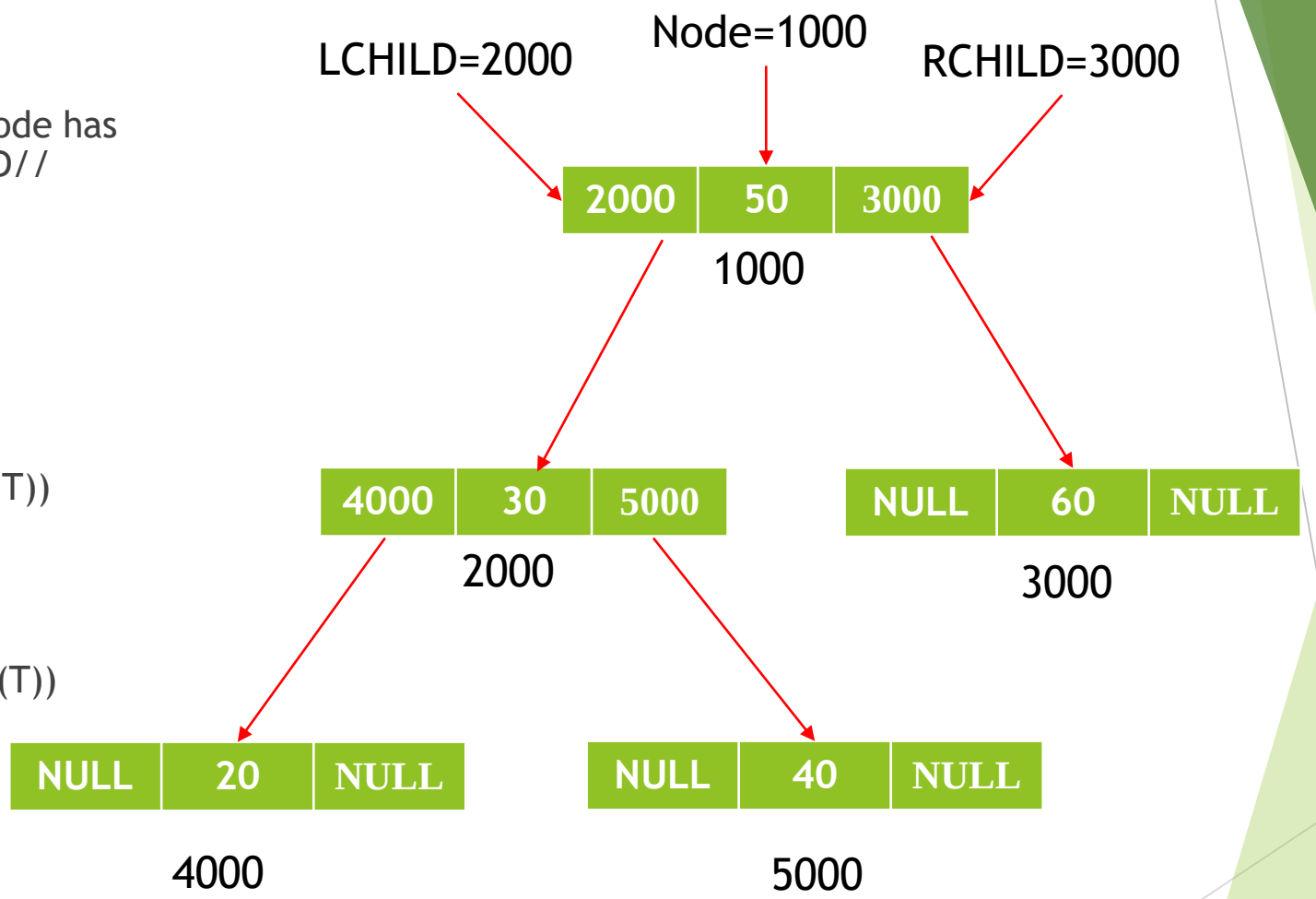
- In this traversal method, the root node is visited first,
- Then the left subtree and
- finally the right subtree.
- Until all nodes are traversed
  - Step 1 – Visit root node.
  - Step 2 – Recursively traverse left subtree.
  - Step 3 – Recursively traverse right subtree.

Algorithm Preorder Traversal:

```
PREORDER(T)
//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

        Print (DATA(T))

        If (T-> L-CHILD ) then
                call PREORDER(LCHILD(T))

        If (T-> R-CHILD ) then
                Call PREORDER(RCHILD(T))

End PREORDER
```

Algorithm Preorder Traversal:

```
PREORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    Print (DATA(T))

    If (T-> L-CHILD ) then
        call PREORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call PREORDER(RCHILD(T))

End PREORDER
```



Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

Print (DATA(T))

If (T-> L-CHILD) then

call PREORDER(LCHILD(T))

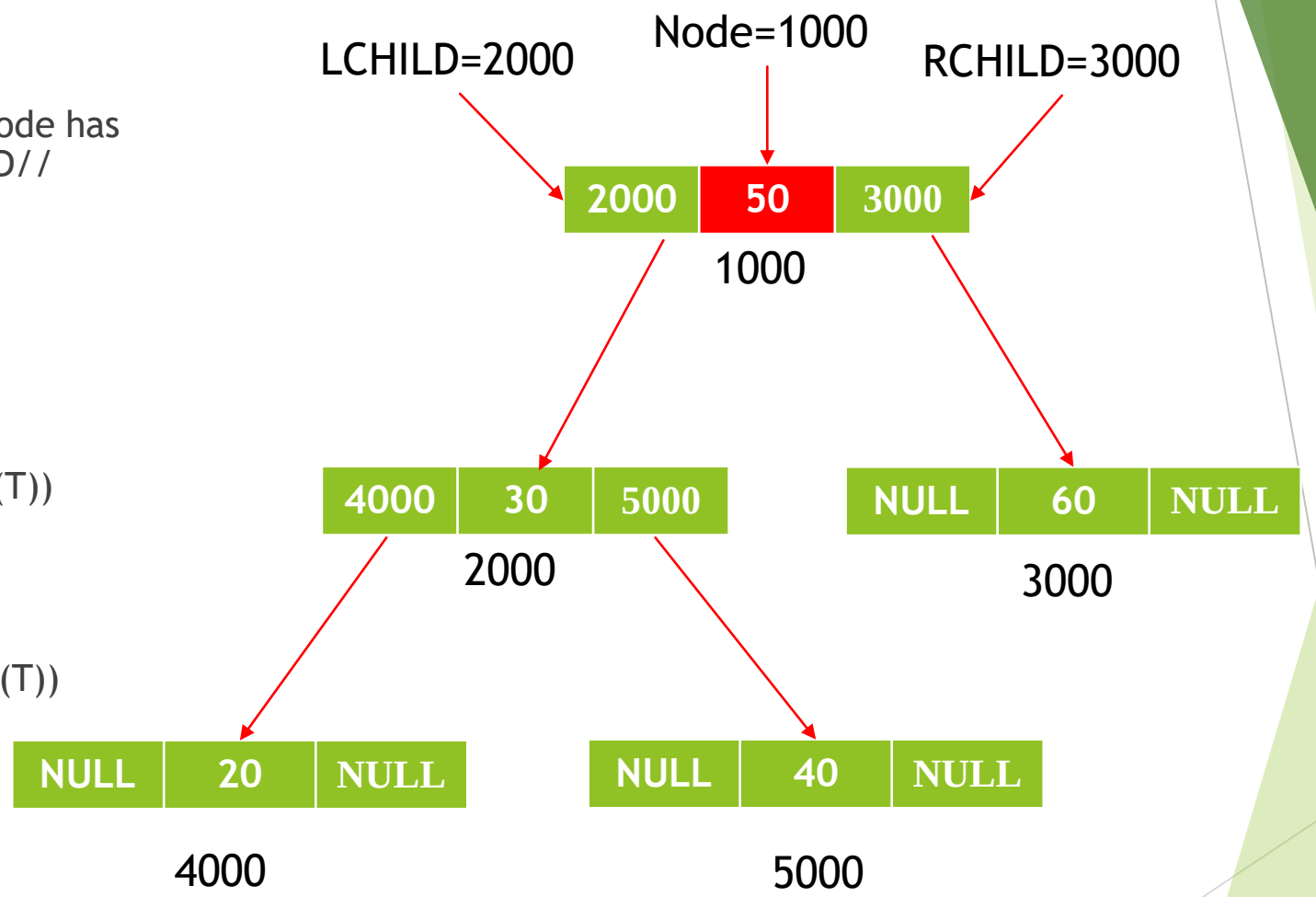
If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER

PREORDER:

```
PREORDE(t =1000)
{

}
```



Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

// Here we will print the root node 1st

Print (DATA(T))

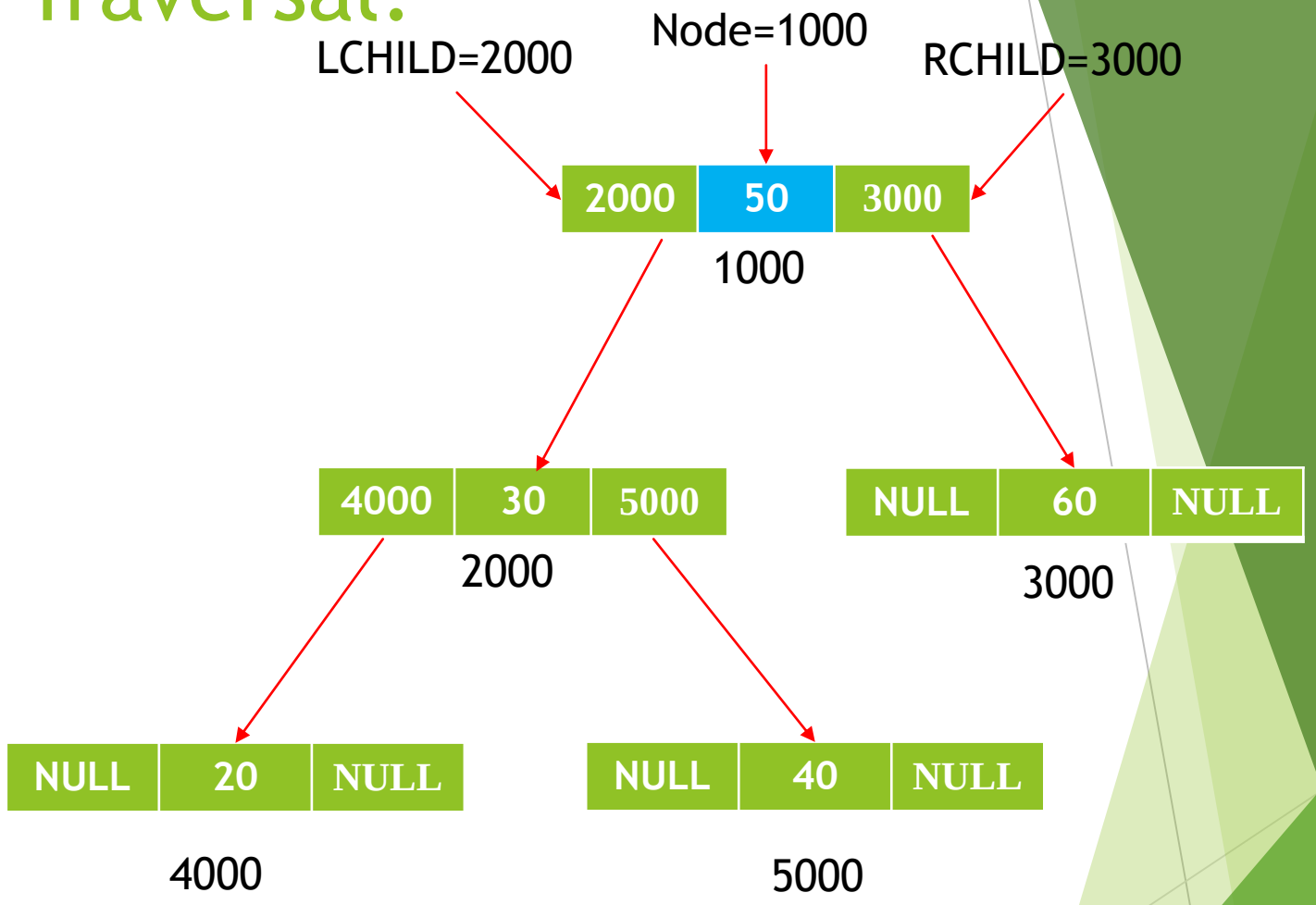
If (T-> L-CHILD) then

call PREORDER(LCHILD(T))

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER **PREORDER: 50**




```
PREORDE(t =1000)
{
        PRINT (50)
        ROOT NODE
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

Print (DATA(T))

//Then it will call the left child 2000(30)

If (T-> L-CHILD) then

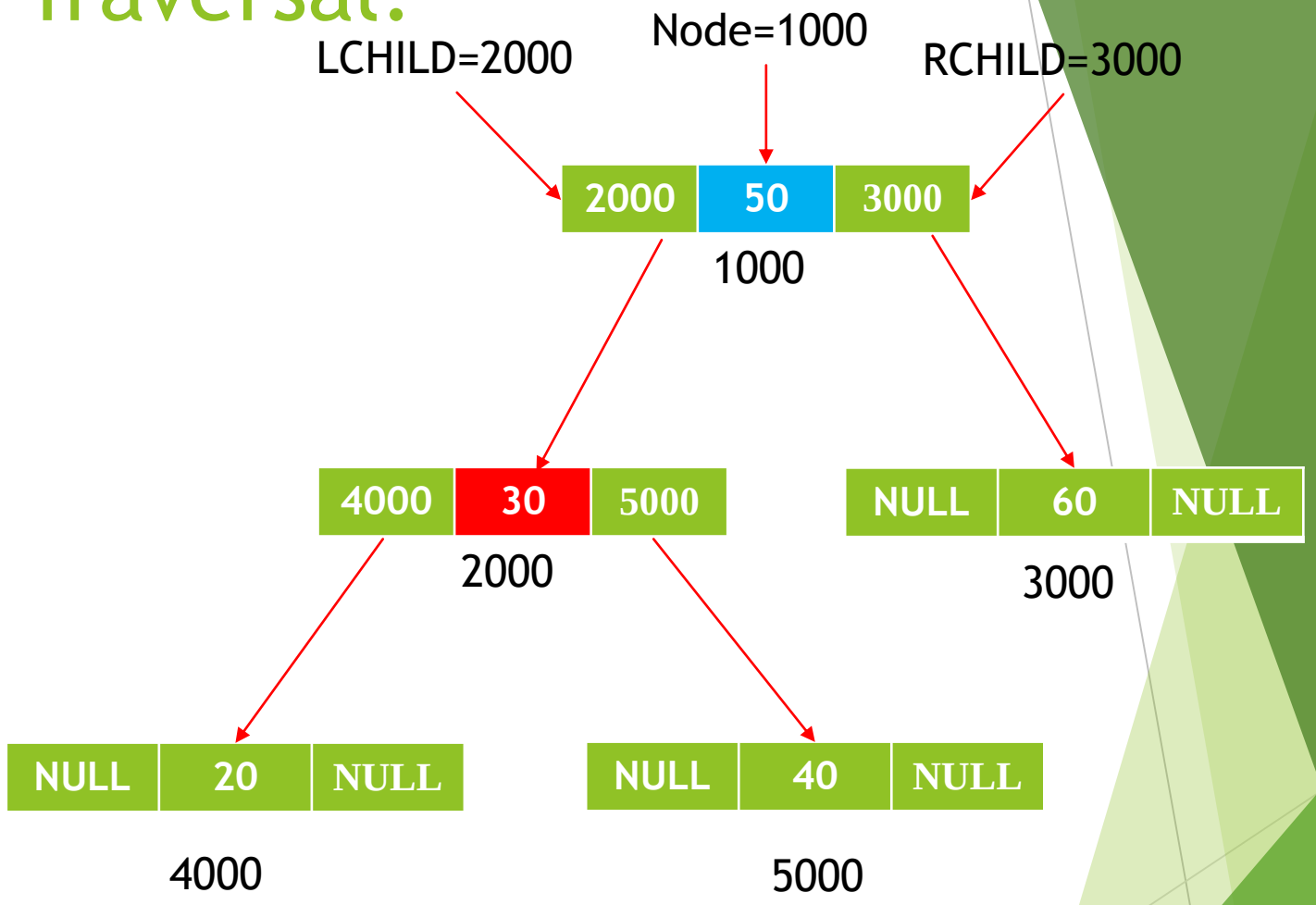
call PREORDER(LCHILD(T))

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

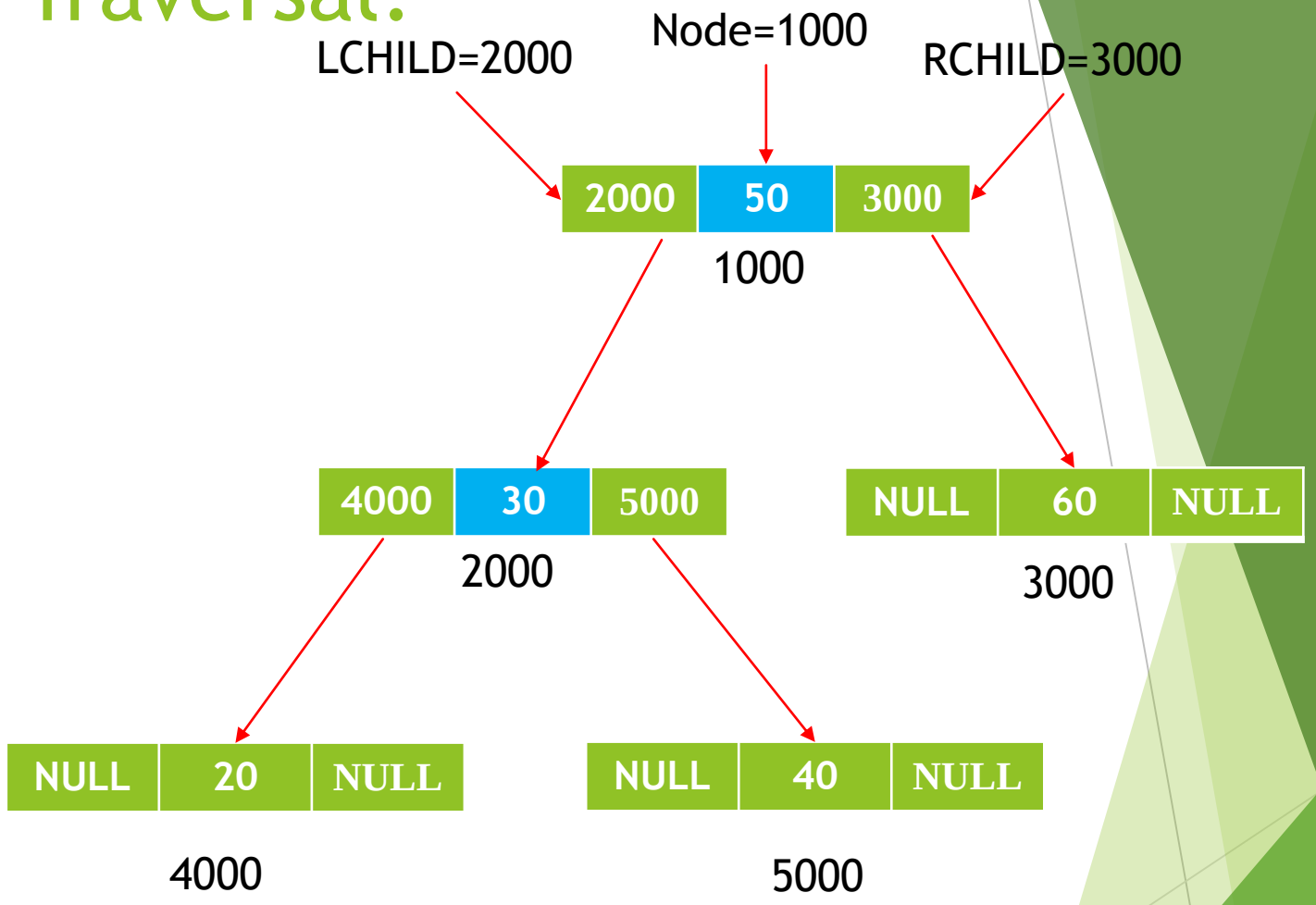
End PREORDER **PREORDER: 50**

```
PREORDE(t =1000)
{
        PRINT (50) ROOT NODE
        IF(2000) L-CHILD
        PREORDER(2000)
}
```



Algorithm Preorder Traversal:

```
PREORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    //print the left child 2000(30)
    Print (DATA(T))
    If (T-> L-CHILD ) then
        call PREORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call PREORDER(RCHILD(T))

End PREORDER
```



PREORDER: 50

```
PREORDE(t =1000)
{
        PRINT (50)  ROOT NODE
        IF(2000) L-CHILD
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)

}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

Print (DATA(T))

//Then it will call the left child 4000(20)

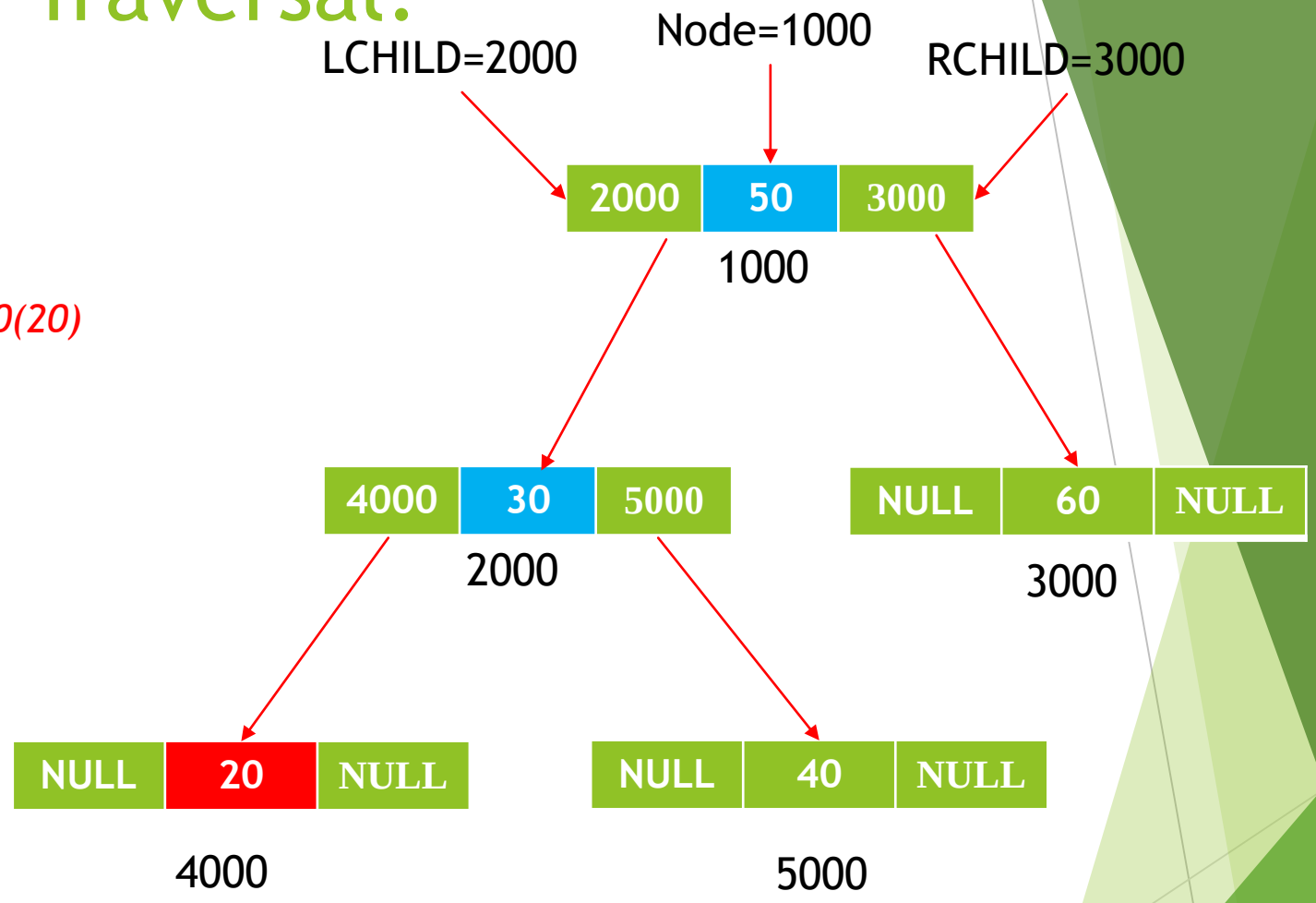
If (T-> L-CHILD) then

call PREORDER(LCHILD(T))

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER



PREORDER: 50,30

```
PREORDE(t =1000)
{
        PRINT (50) ROOT NODE
        IF(2000) L-CHILD
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(4000) L-CHILD
        PREORDER(4000)
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

//print the *left child 4000(20)*

Print (DATA(T))

If (T-> L-CHILD) then

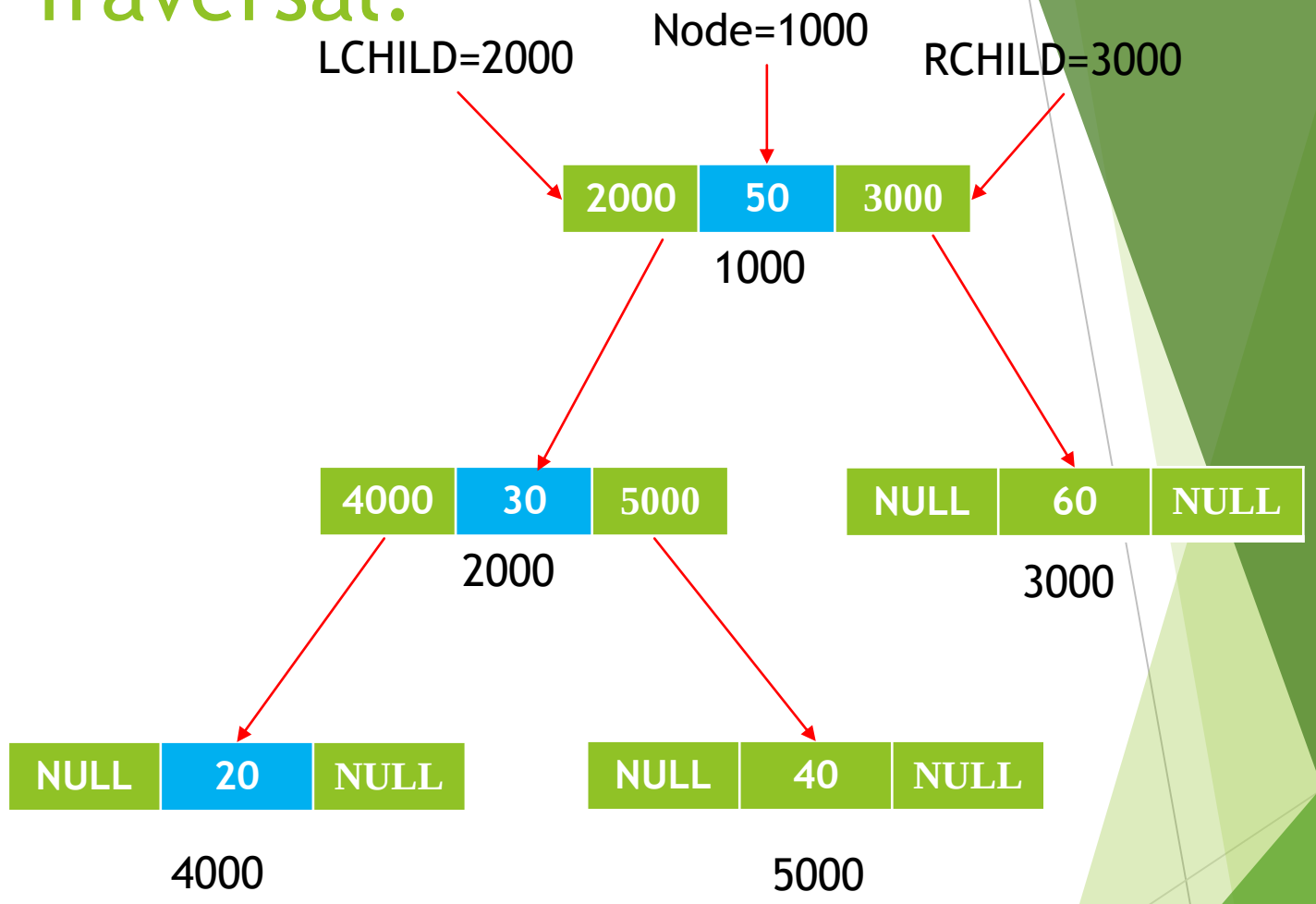
call PREORDER(LCHILD(T))

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER

PREORDER: 50,30




```
PREORDE(t =1000)
{
        PRINT (50) ROOT NODE
        IF(2000) L-CHILD
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(4000) L-CHILD
        PREORDER(4000)
}
```

```
PREORDE(t =4000)
{
        PRINT (20)
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

//it will pass the null

Print (DATA(T))

If (T-> L-CHILD) then//condition fails

call PREORDER(LCHILD(T))

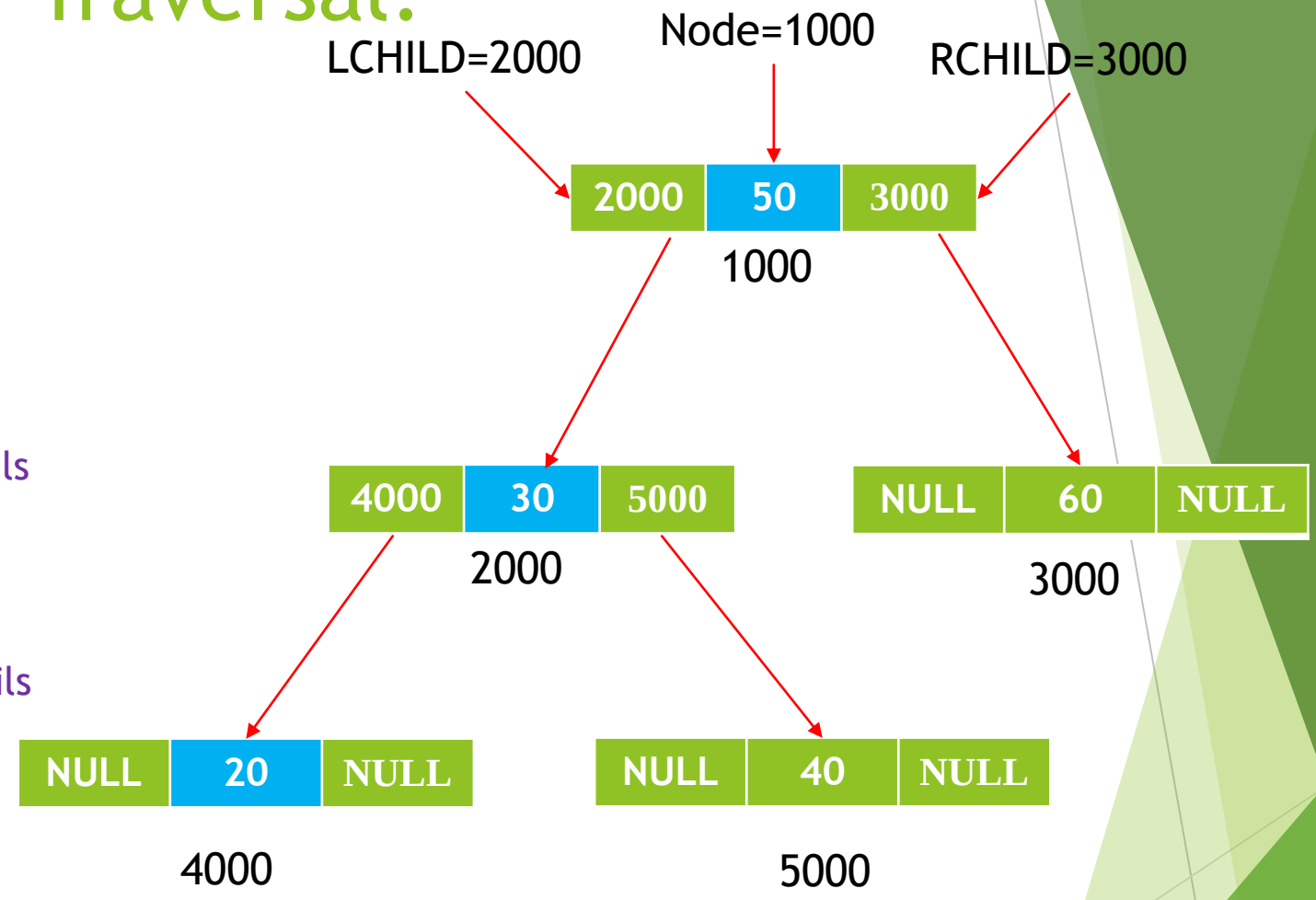
If (T-> R-CHILD) then//condition fails

Call PREORDER(RCHILD(T))

//back to parent node

End PREORDER

PREORDER: 50,30,20




```
PREORDE(t =1000)
{
        PRINT (50) ROOT NODE
        IF(2000) L-CHILD
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(4000)
        PREORDER(4000)
        L-CHILD
}
```

```
PREORDE(t =4000)
{
        PRINT (20)
         IF(NULL)
        PREORDER(NULL)
        BACK TO PARENT NODE
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

Print (DATA(T))

If (T-> L-CHILD) then

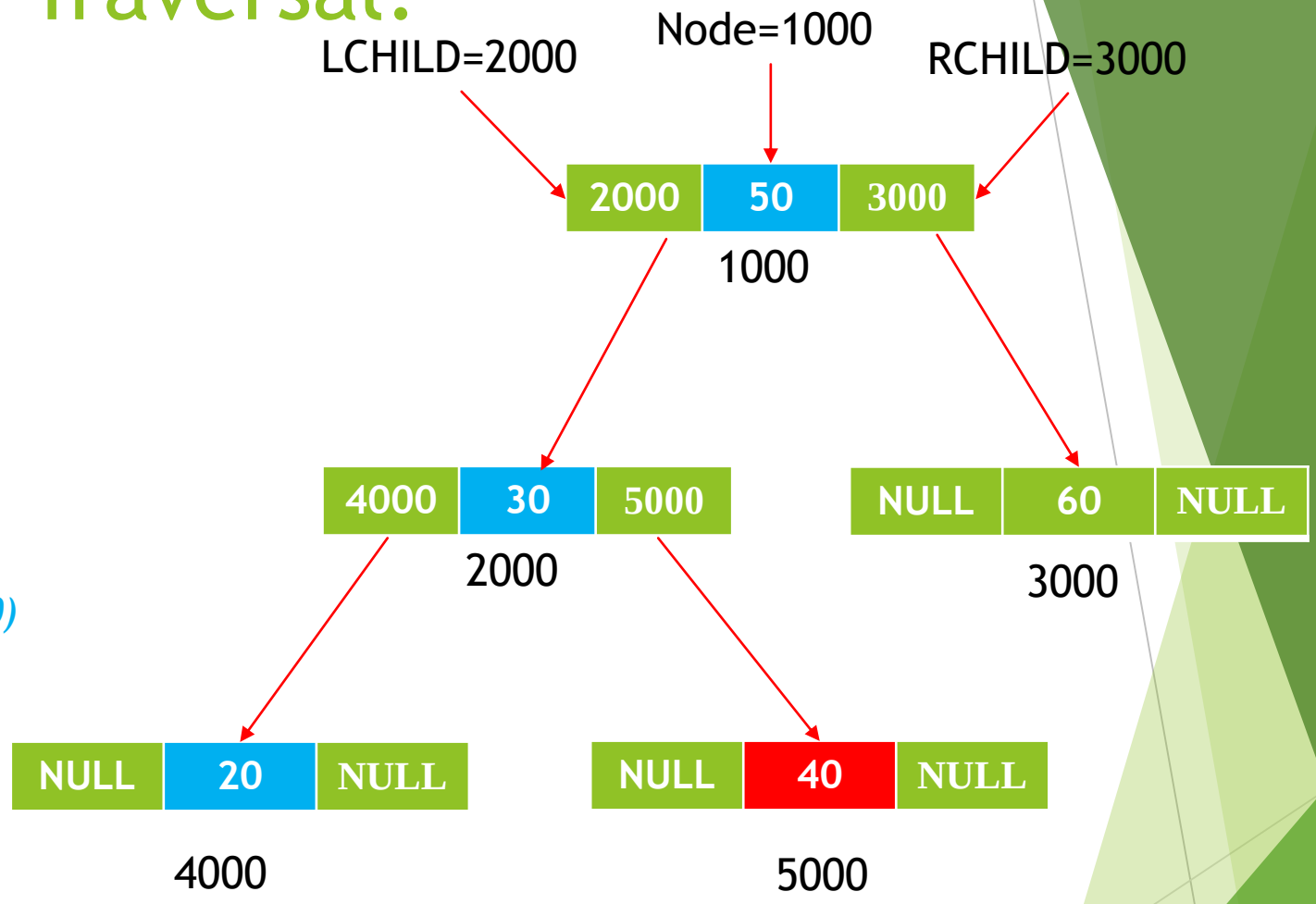
call PREORDER(LCHILD(T))

//Then it will call the right child 5000(40)

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER



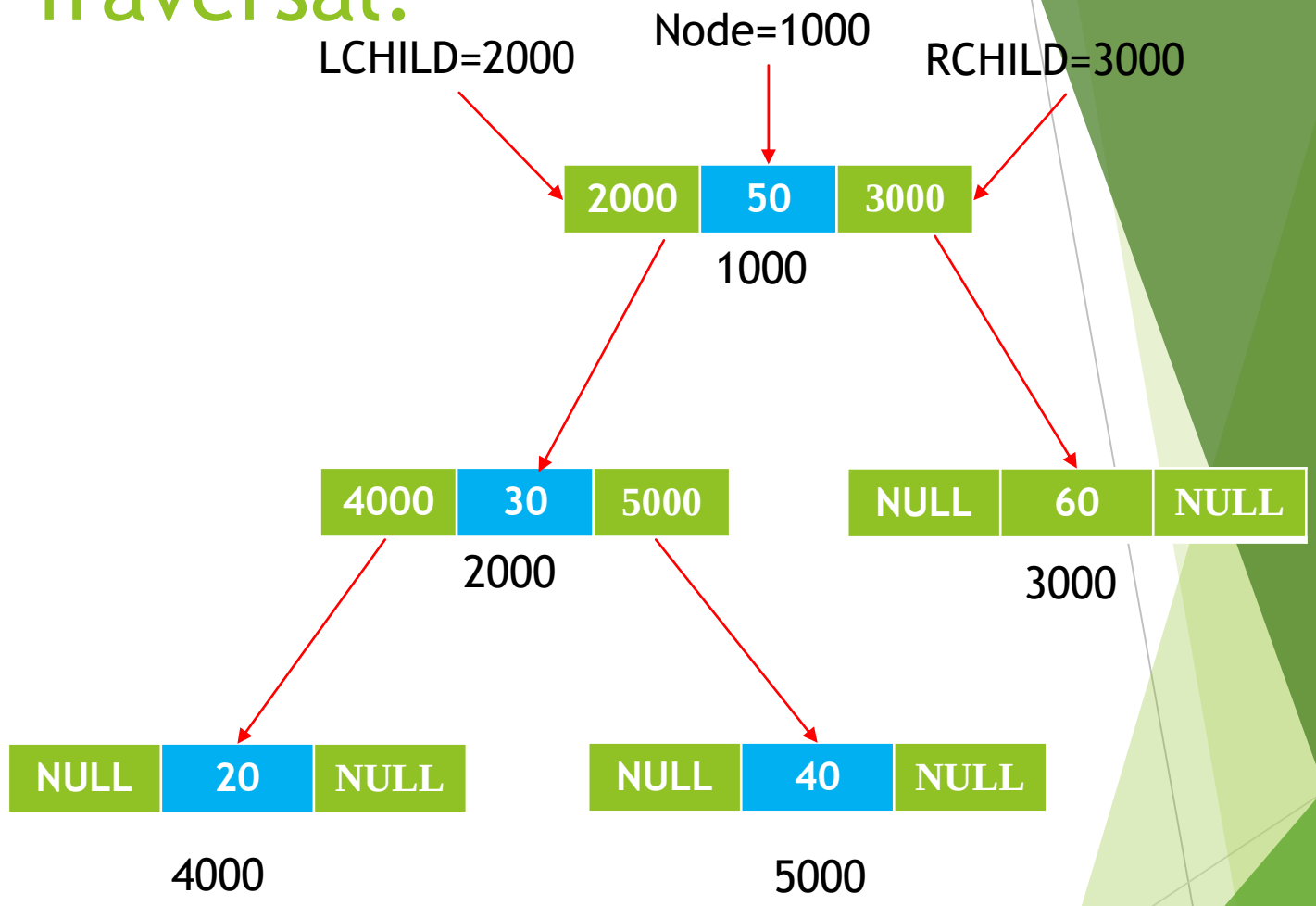
PREORDER: 50,30,20

```
PREORDE(t =1000)
{
        PRINT (50)
        IF(2000)
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(5000)
        PREORDER(5000)
         R-CHILD
}
```

Algorithm Preorder Traversal:

```
PREORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

//print the left child 4000(20)

        Print (DATA(T))

        If (T-> L-CHILD ) then

                call PREORDER(LCHILD(T))


        If (T-> R-CHILD ) then

                Call PREORDER(RCHILD(T))


End PREORDER
```



PREORDER: 50,30,20

```
PREORDE(t =1000)
{
        PRINT (50)
        IF(2000)
        PREORDER(2000)
}
```

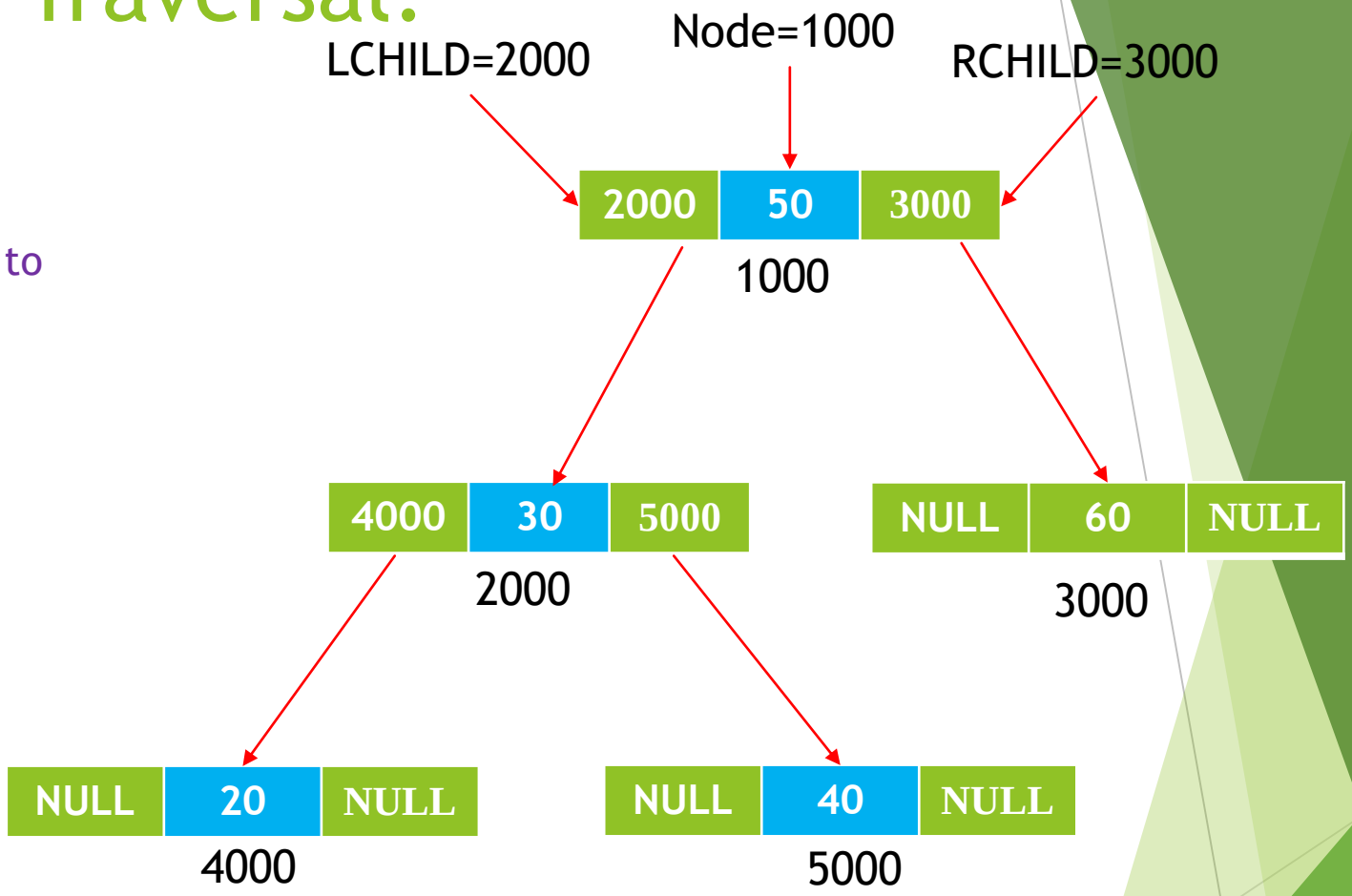
```
PREORDE(t =2000)
{
        PRINT (30)
        IF(5000)
        PREORDER(5000)
         R-CHILD
}
```

```
PREORDE(t =2000)
{
        PRINT (40)
}
```

Algorithm Preorder Traversal:

```
PREORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

//it will pass the null condition fails back to
parent node 2000(30)

        Print (DATA(T))

        If (T-> L-CHILD ) then

                call PREORDER(LCHILD(T))

        If (T-> R-CHILD ) then

                Call PREORDER(RCHILD(T))

End PREORDER
```

PREORDER: 50,30,20,40




```
PREORDE(t =1000)
{
        PRINT (50)
        IF(2000)
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(5000)
        PREORDER(5000)
        R-CHILD

}
```

```
PREORDE(t =2000)
{
        PRINT (40)
        IF(null)
        PREORDER(null)
        BACK TO PARENT NODE
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

//it will pass the null condition fails back to parent node and root node 1000(50)

Print (DATA(T))

If (T-> L-CHILD) then

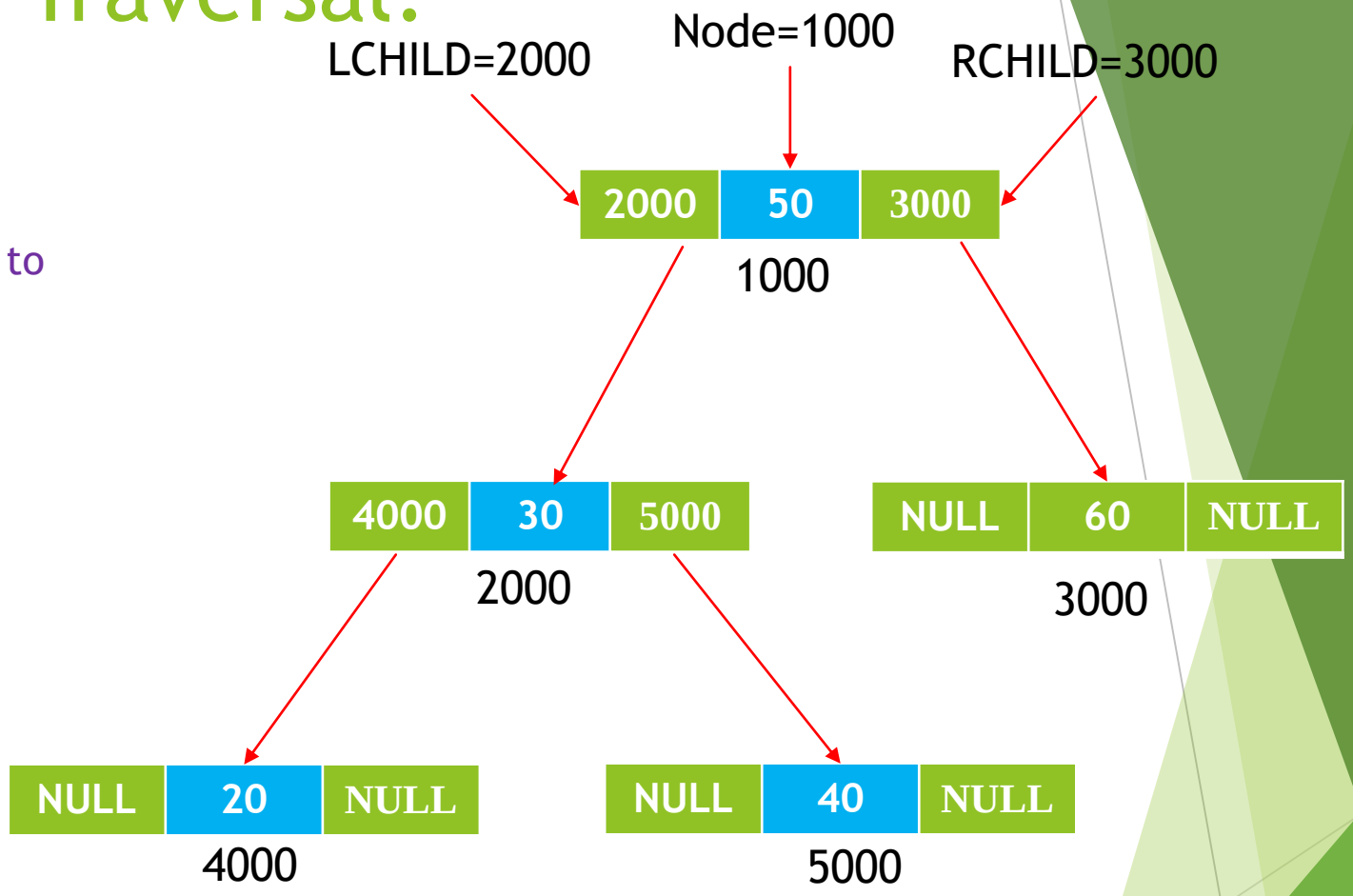
call PREORDER(LCHILD(T))

If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER

PREORDER: 50,30,20,40




```
PREORDE(t =1000)
{
        PRINT (50)
        IF(2000)
        PREORDER(2000)
}
```

```
PREORDE(t =2000)
{
        PRINT (30)
        IF(5000)
        PREORDER(5000)
        R-CHILD
BACK TO PARENT NODE
}
```

```
PREORDE(t =2000)
{
        PRINT (40)
        IF(null)
        PREORDER(null)
        BACK TO PARENT NODE
}
```

Algorithm Preorder Traversal:

PREORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

Print (DATA(T))

If (T-> L-CHILD) then

call PREORDER(LCHILD(T))

//Then it will call the right child 5000(40)

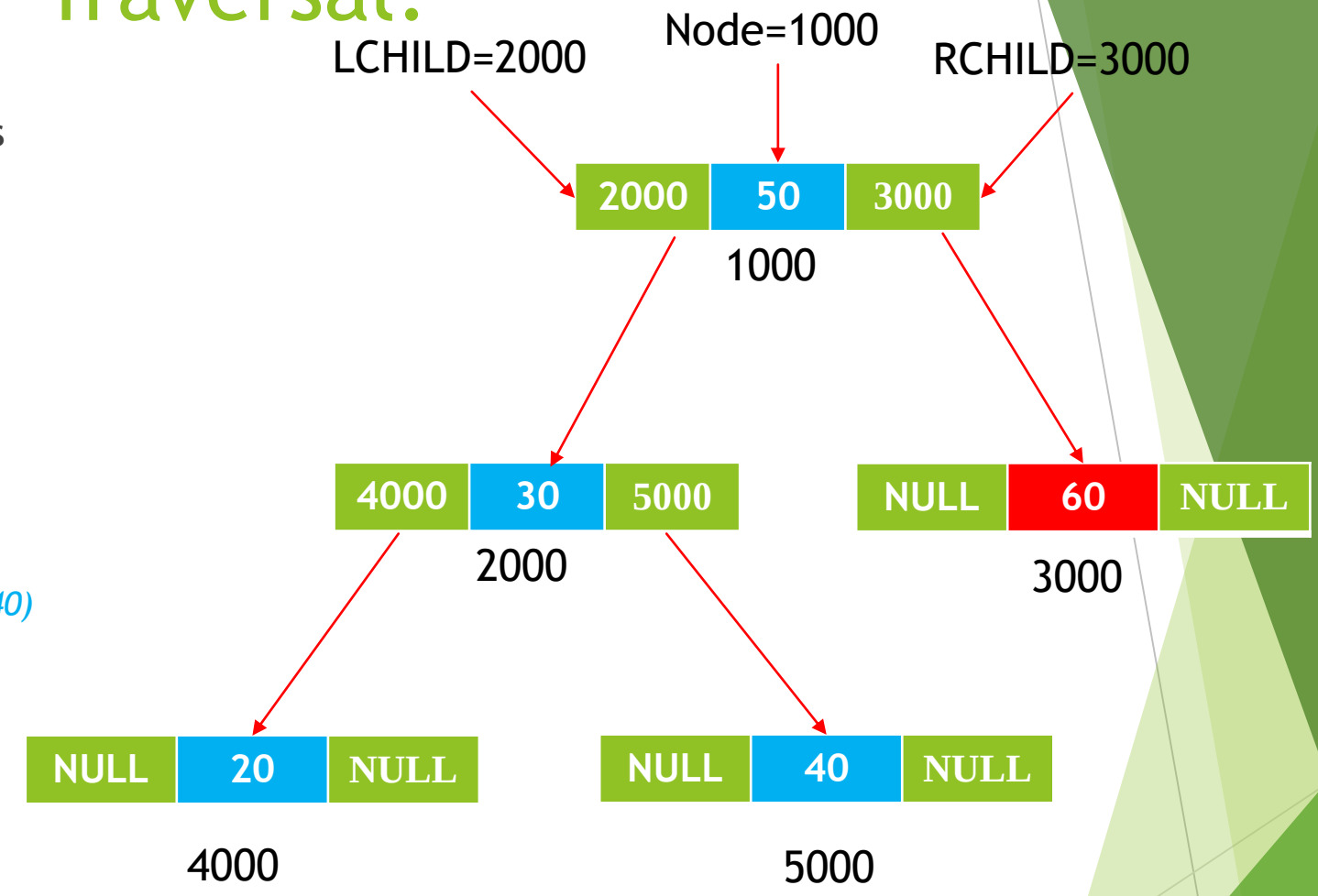
If (T-> R-CHILD) then

Call PREORDER(RCHILD(T))

End PREORDER

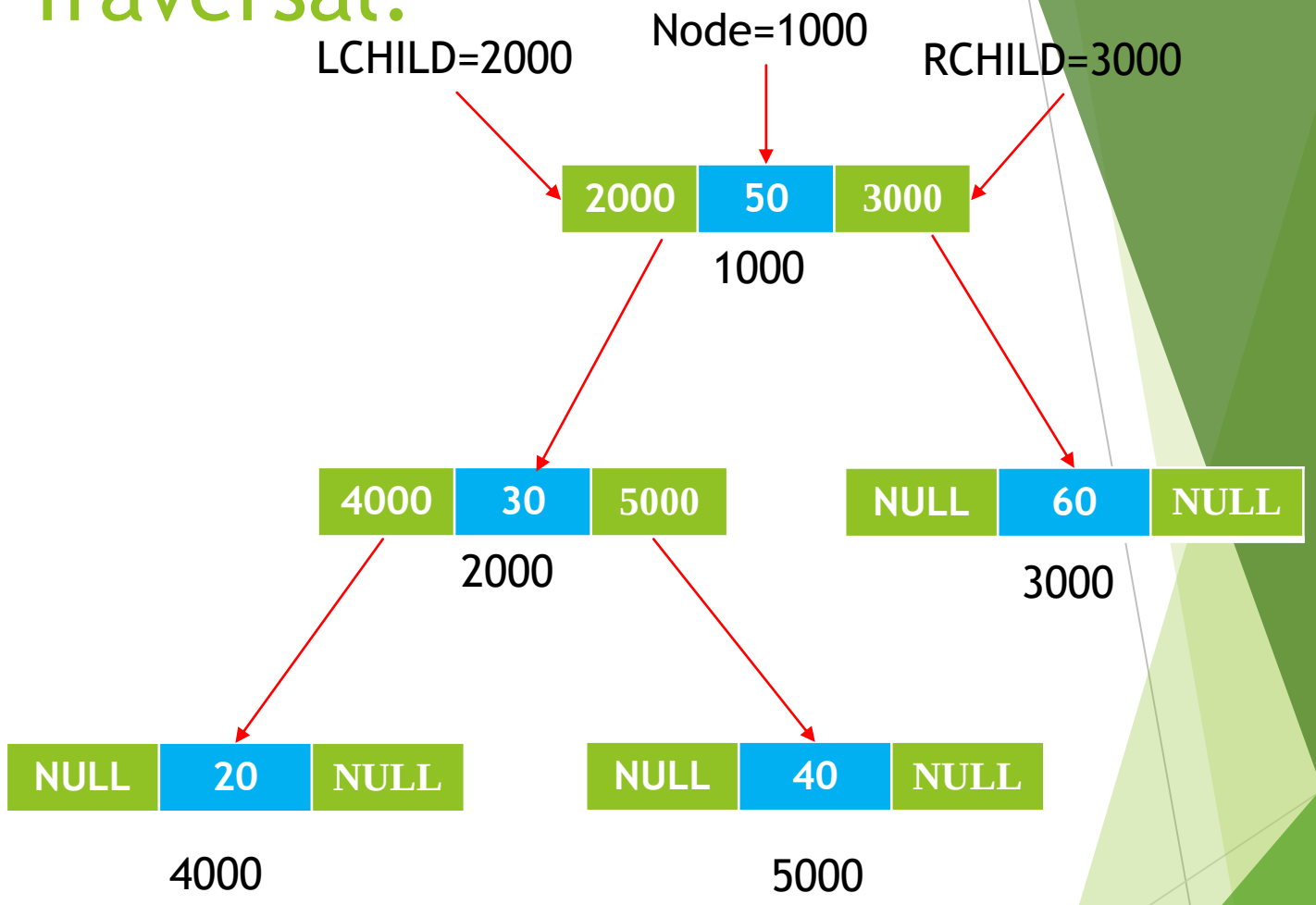
PREORDER: 50,30,20,40

```
PREORDE(t =1000)
{
        PRINT (50)
        IF(3000)
        PREORDER(3000)
        R-CHILD
}
```



Algorithm Preorder Traversal:

```
PREORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

//print the left child 3000(60)

    Print (DATA(T))


    If (T-> L-CHILD ) then

        call PREORDER(LCHILD(T))

    If (T-> R-CHILD ) then

        Call PREORDER(RCHILD(T))

End PREORDER
```




```
PREORDE(t =1000)
{
        IF(3000)
        PREORDER(3000)
        R-CHILD
}
```

```
PREORDE(t =3000)
{
        PRINT (60)
        IF(NULL)
        PREORDER(NULL)

        }
```

PREORDER: 50,30,20,40,60

Algorithm Preorder Traversal:

```
struct Node {

    int data;

    struct Node *left, *right;

    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
```

```
int main()
{
    struct Node* root = new Node(50);
    root->left = new Node(30);
    root->right = new Node(60);
    root->left->left = new Node(20);
    root->left->right = new Node(40);
cout << "\n preorder traversal of binary tree is \n";
 printpreorder(root);
```

```
void printInorder(struct Node*
node)
{
 cout<< node->data << " ";
 if ( node ->left)
    printpreorder( node->left);
   if (node ->right)
    printpreorder( node->right);
```

OUTPUT :

```
E:\winter sem 2020\c++ doa\preorder.exe

preorder traversal of binary tree is
50 30 20 40 60
--------------------------------
Process exited after 0.3017 seconds with return value 0
Press any key to continue . . .
```

Post-order Traversal:

- In this traversal method, The root node is visited last, hence the name.
- First we traverse the left subtree,
- then the right subtree and
- finally the root node.
- Until all nodes are traversed :
  - Step 1 – Recursively traverse left subtree.
  - Step 2 – Recursively traverse right subtree.
  - Step 3 – Visit root node.

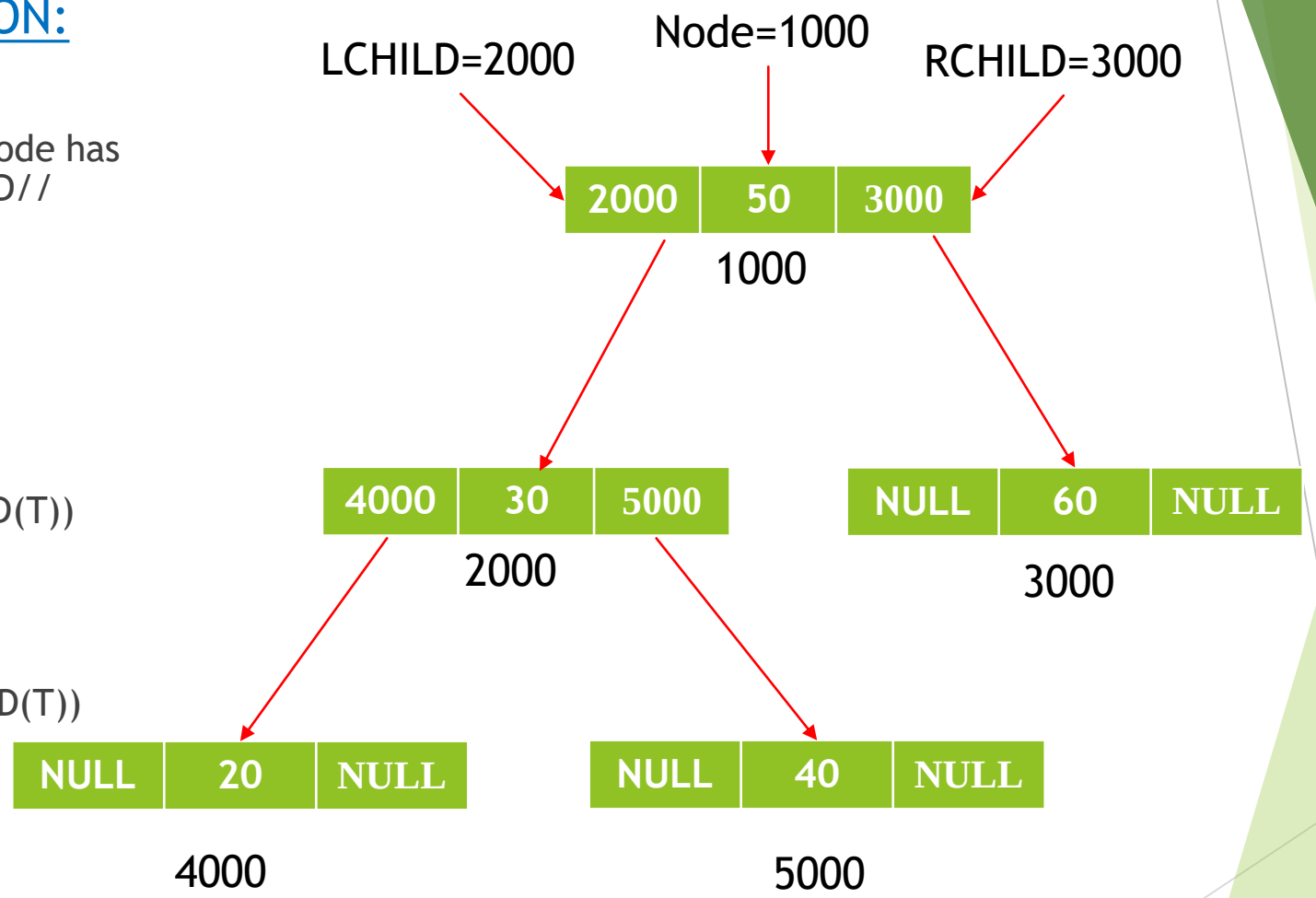
Algorithm Post-order Traversal:

```
POSTORDER(T)
//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call POSTORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call POSTORDER(RCHILD(T))

    Print (DATA(T))

End POSTORDER
```

Algorithm Post-order Traversal:

EXAMPLE & EXPLANATION:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

        Print (DATA(T))

        If (T-> L-CHILD ) then
                call POSTORDER(LCHILD(T))

        If (T-> R-CHILD ) then
                Call POSTORDER(RCHILD(T))

End POSTORDER
```



Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

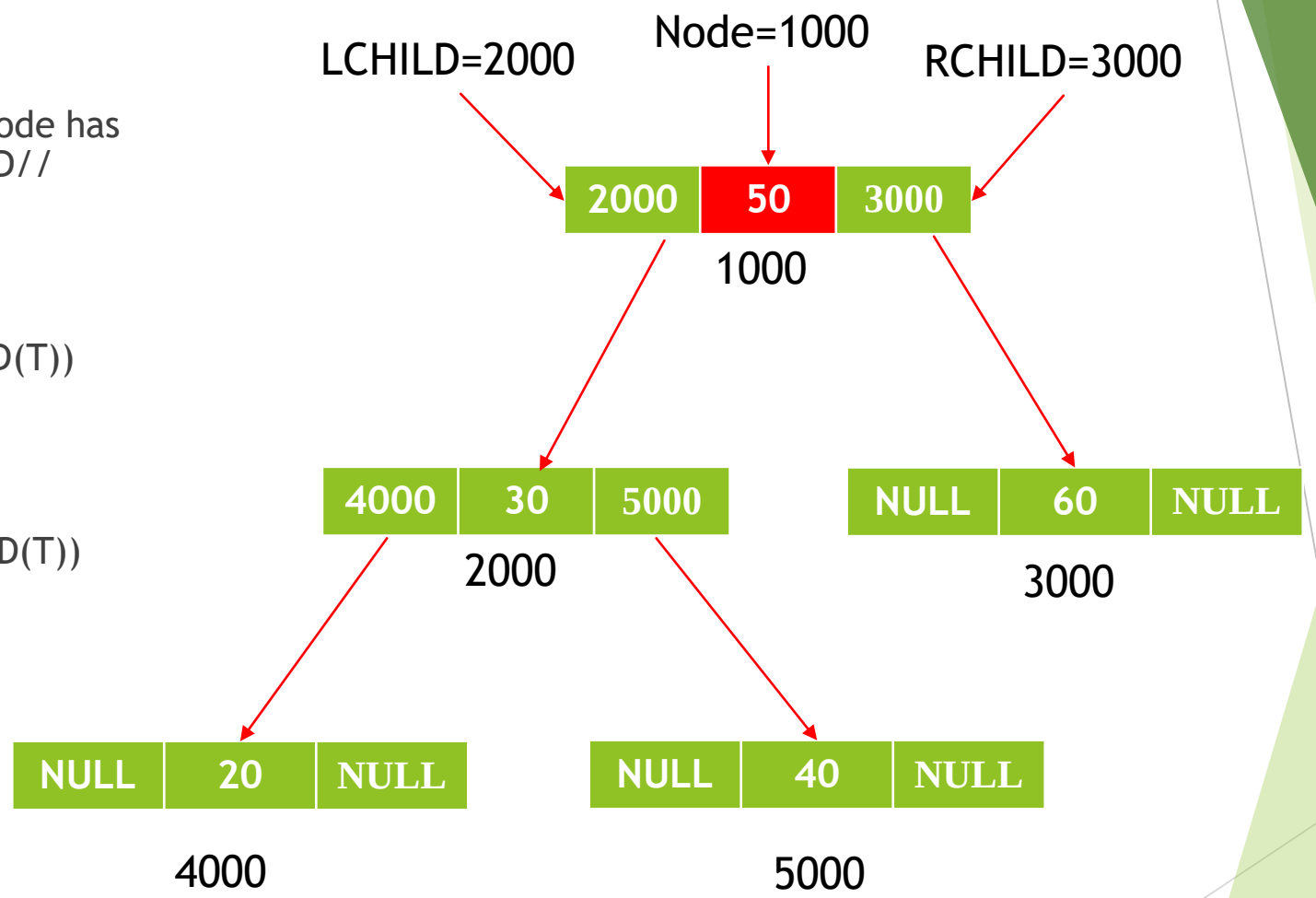
call POSTORDER(LCHILD(T))

If (T-> R-CHILD) then

Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER

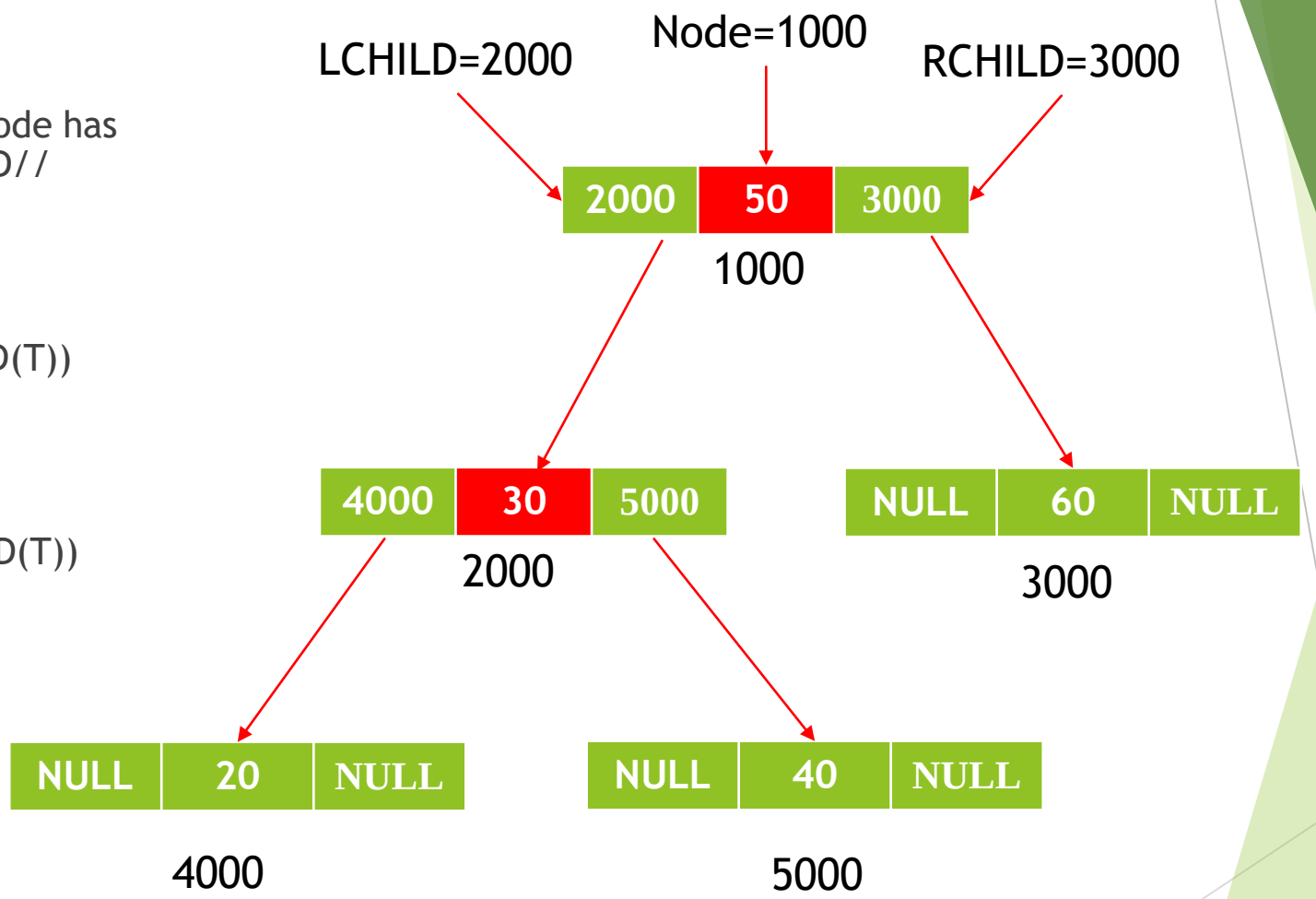


POSTORDER:

```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

Algorithm Post-order Traversal:

```
POSTORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call POSTORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call POSTORDER(RCHILD(T))

    Print (DATA(T))

End POSTORDER
```



POSTORDER:

```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)
```

Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

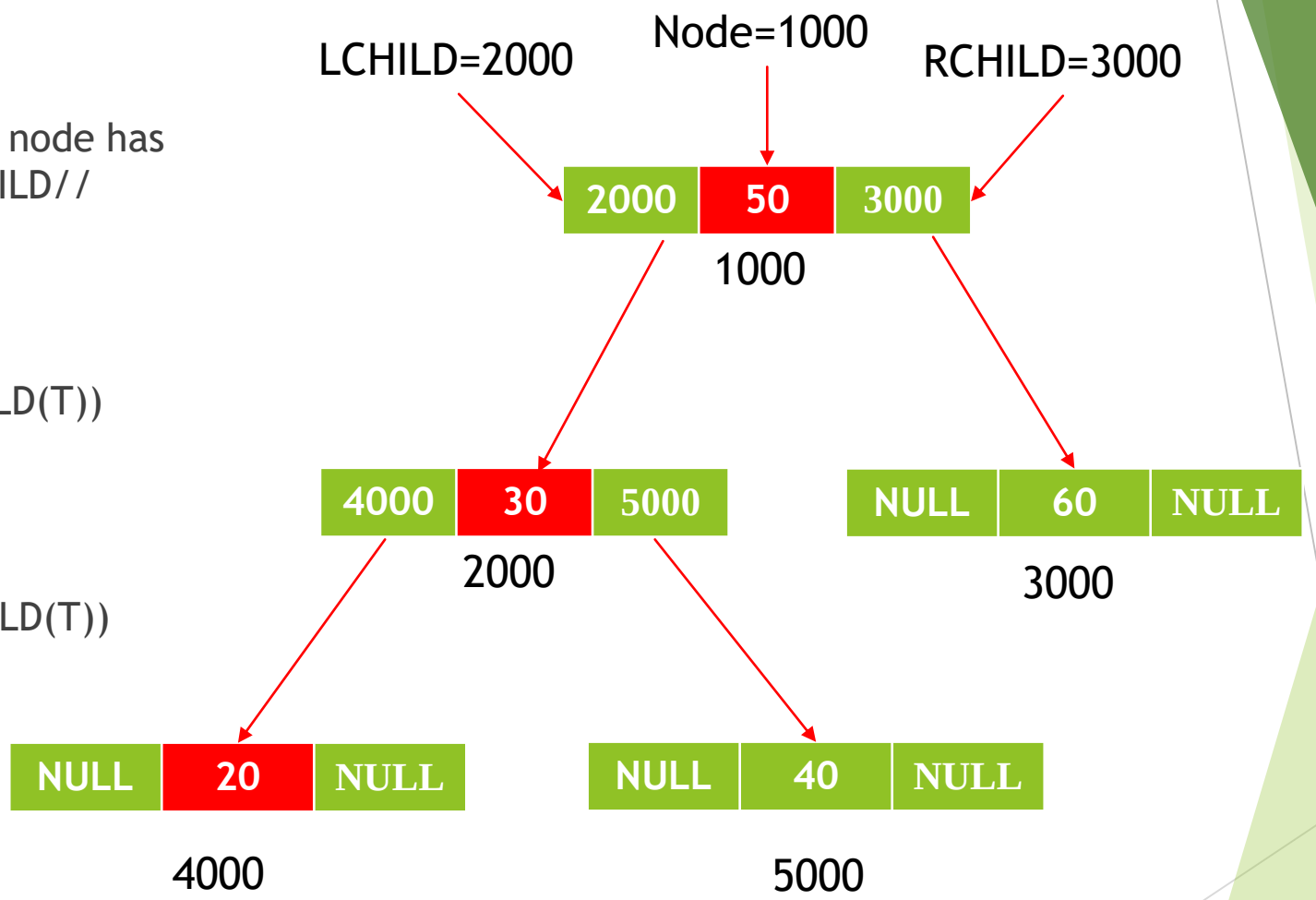
call POSTORDER(LCHILD(T))

If (T-> R-CHILD) then

Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER



POSTORDER:

```
Postorder(t =1000)
{
        if(2000)
        postorder(2000)
```

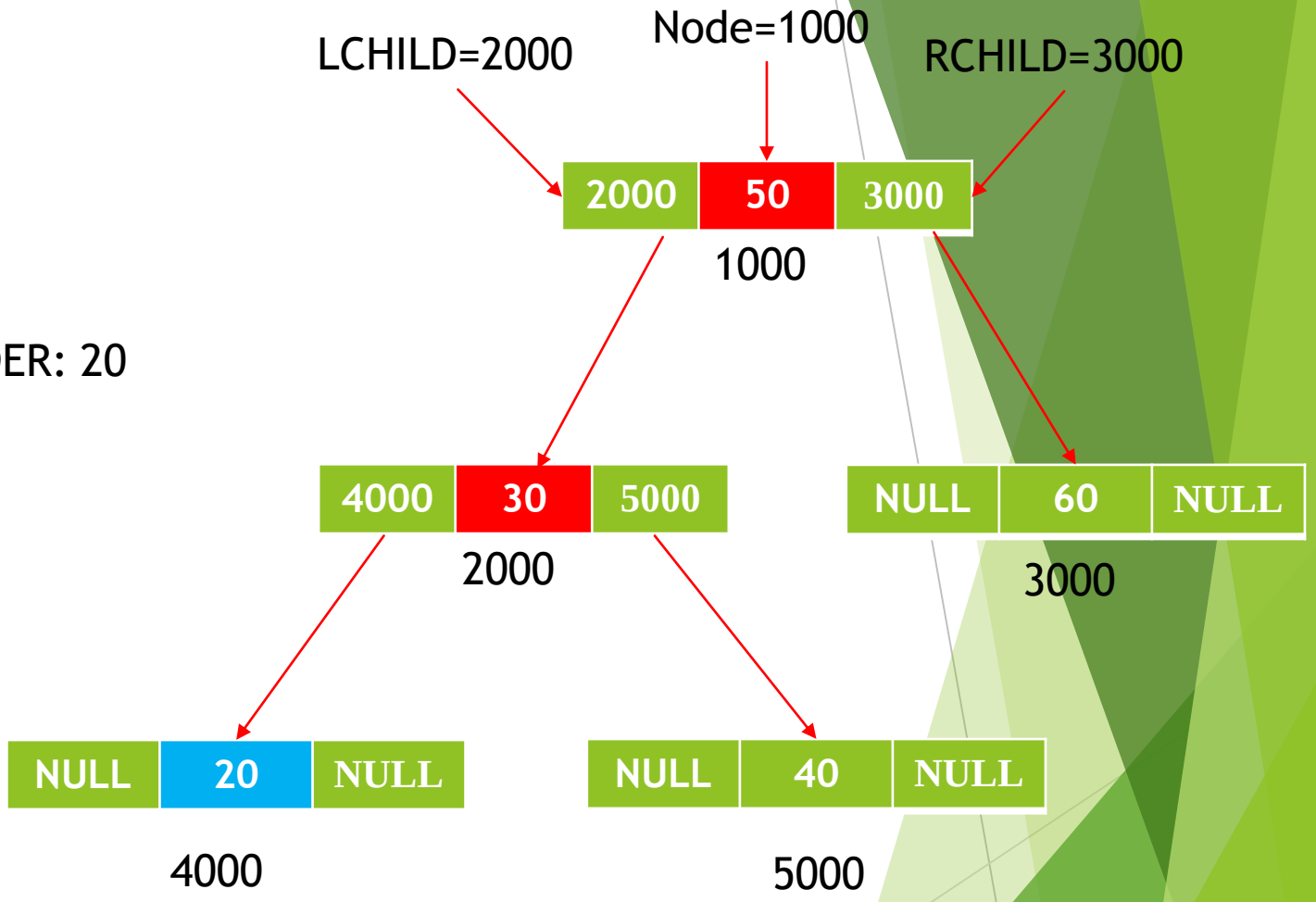
```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)
```

```
Postorder(t =4000)
{
        if(NULL)
```

Algorithm Post-order Traversal:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call POSTORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call POSTORDER(RCHILD(T))

    Print (DATA(T))

End POSTORDER
```

POSTORDER: 20




```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

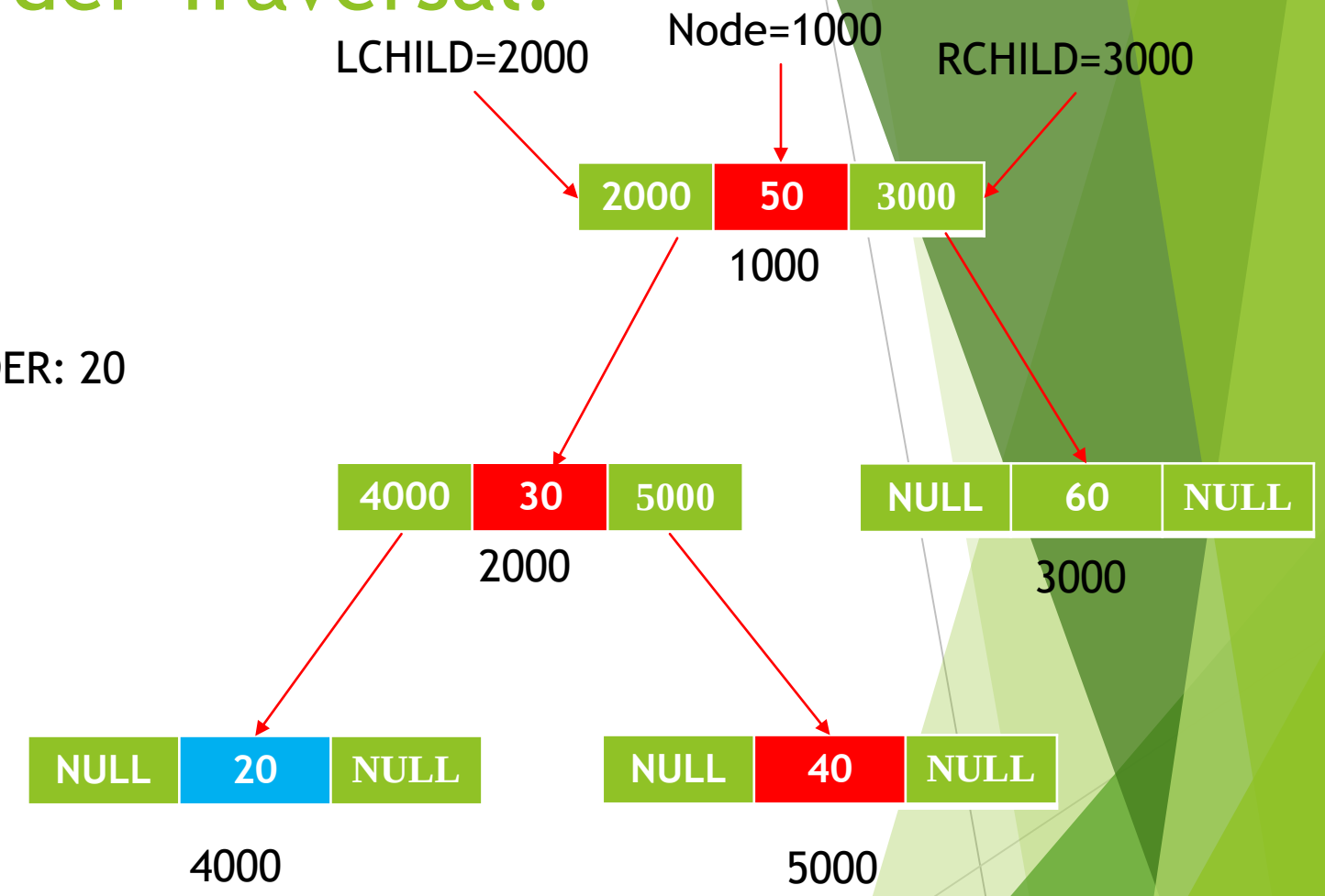
```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)
```

```
Postorder(t =4000)
{
        if(NULL)
        print(20)
```

Algorithm Post-order Traversal:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call POSTORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call POSTORDER(RCHILD(T))

    Print (DATA(T))

End POSTORDER
```

POSTORDER: 20




```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

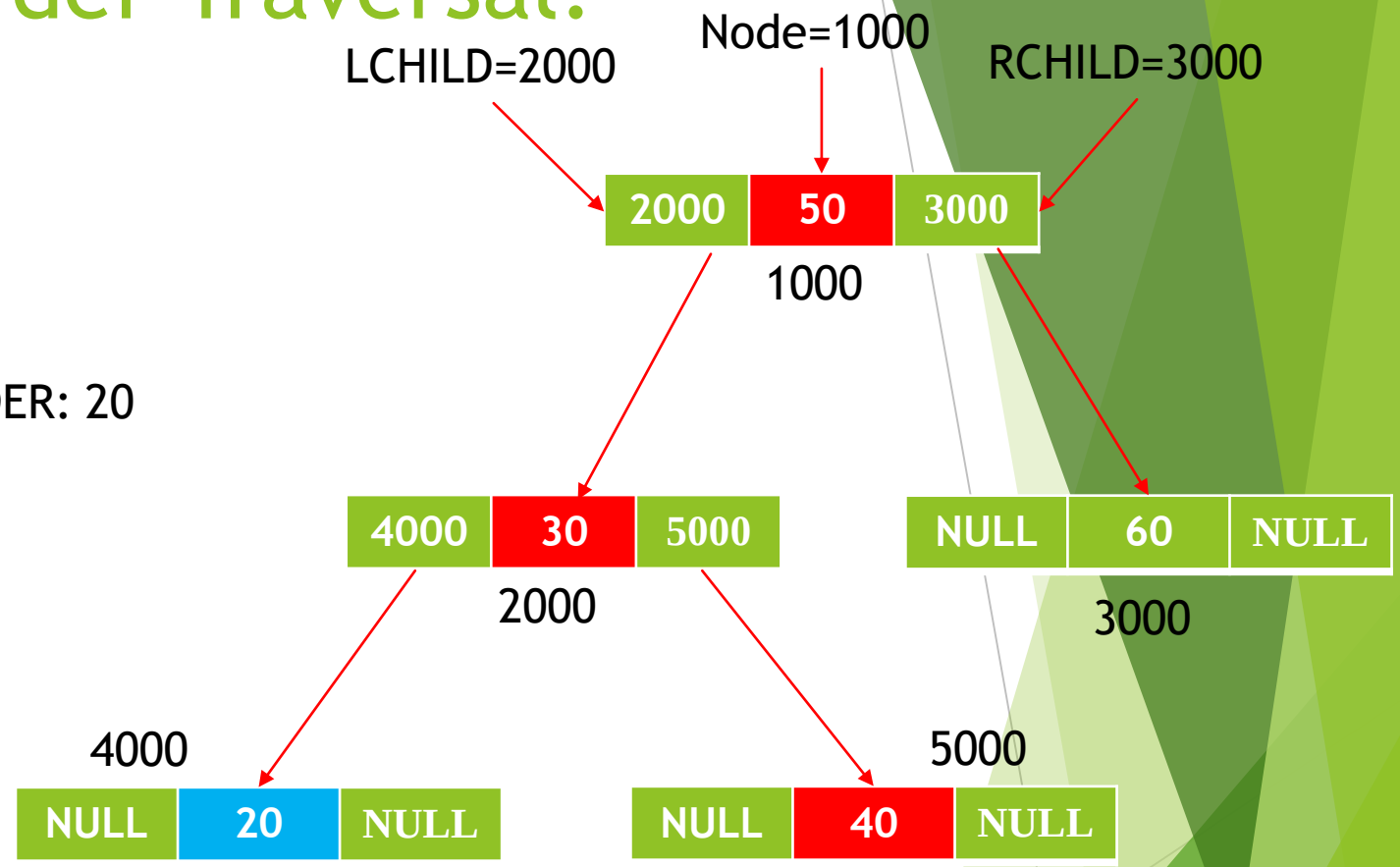
```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)
```

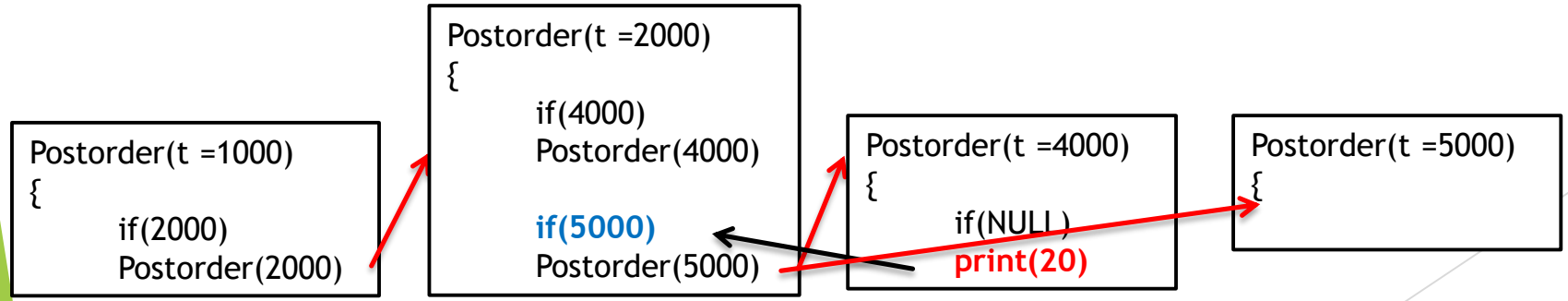
```
Postorder(t =4000)
{
        if(NULL)
        print(20)
```

Algorithm Post-order Traversal:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call POSTORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call POSTORDER(RCHILD(T))

    Print (DATA(T))

End POSTORDER
```

POSTORDER: 20

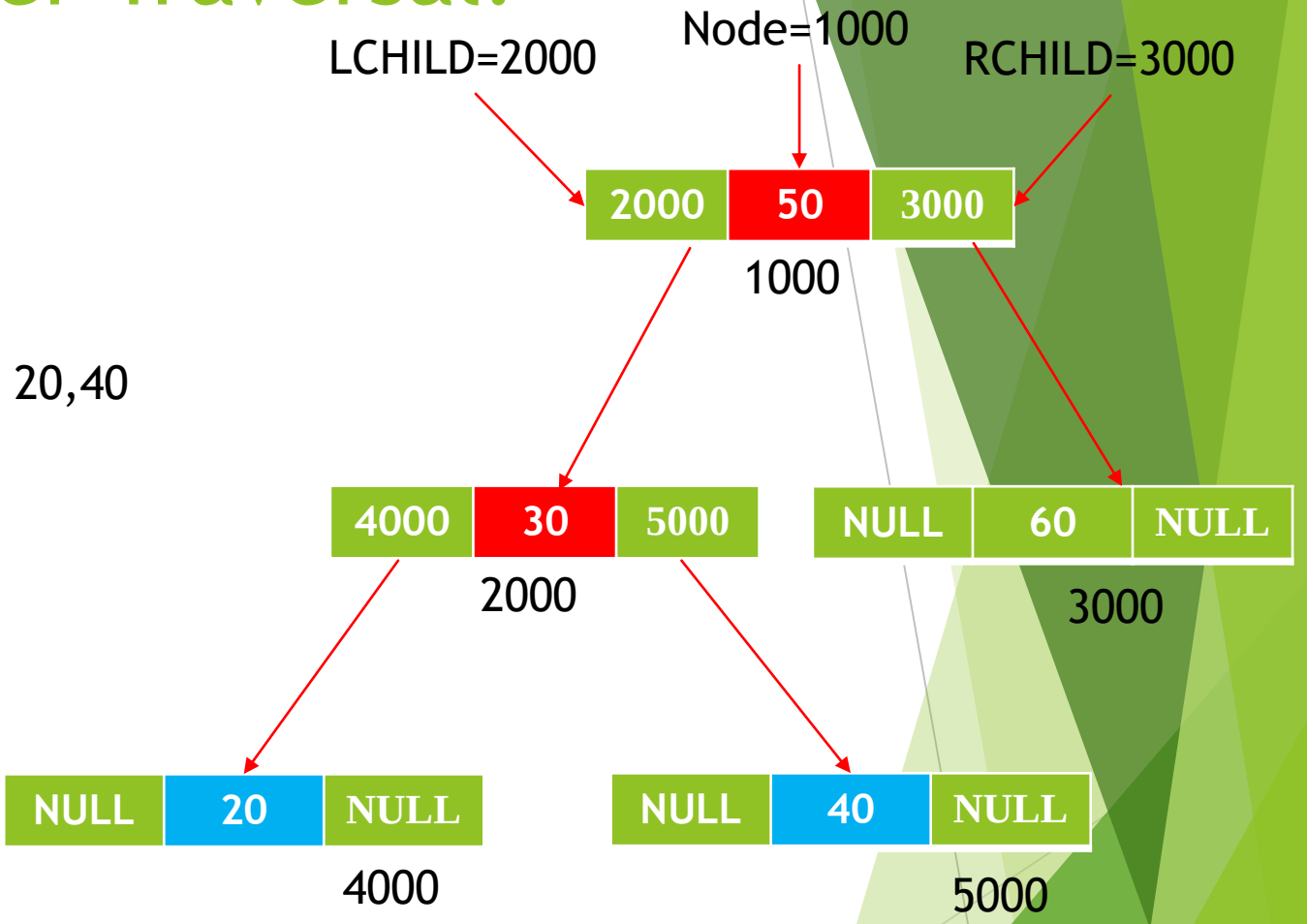




Algorithm Post-order Traversal:

```
POSTORDER(T)
//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call PREORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call PREORDER(RCHILD(T))

    Print (DATA(T))

End PREORDER
```

POSTORDER: 20,40




```
Postorder(t =1000)
{
    if(2000)
    Postorder(2000)
```

```
Postorder(t =2000)
{
    if(4000)
    Postorder(4000)

    if(5000)
    Postorder(5000)
```

```
Postorder(t =4000)
{
    if(NULL)
    print(20)
```

```
Postorder(t =5000)
{
    if(NULL)
    print(40)
```

Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

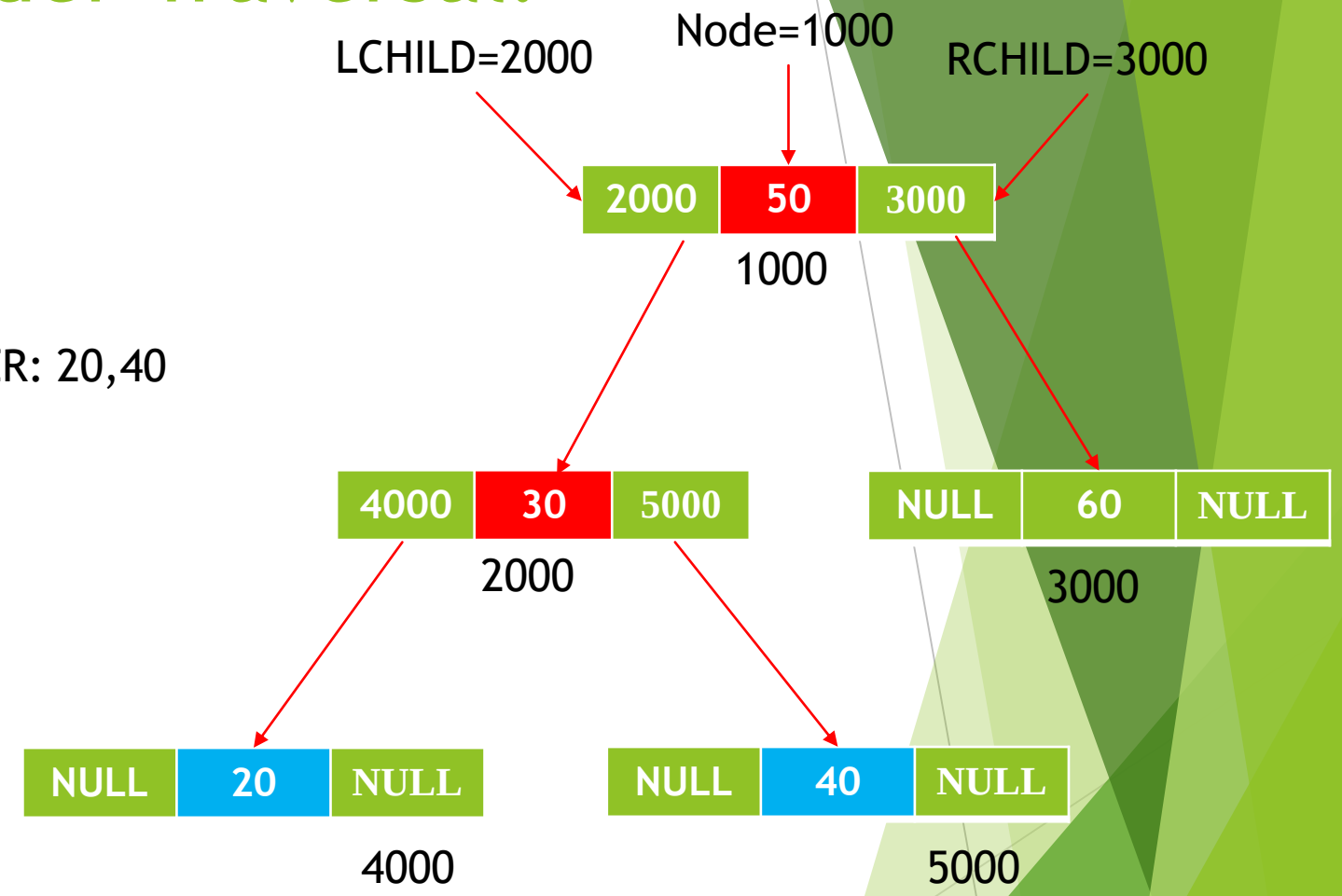
If (T-> L-CHILD) then
call PREORDER(LCHILD(T))

If (T-> R-CHILD) then
Call PREORDER(RCHILD(T))

Print (DATA(T))

End PREORDER

POSTORDER: 20,40




```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)

        if(5000)
        Postorder(5000)
```

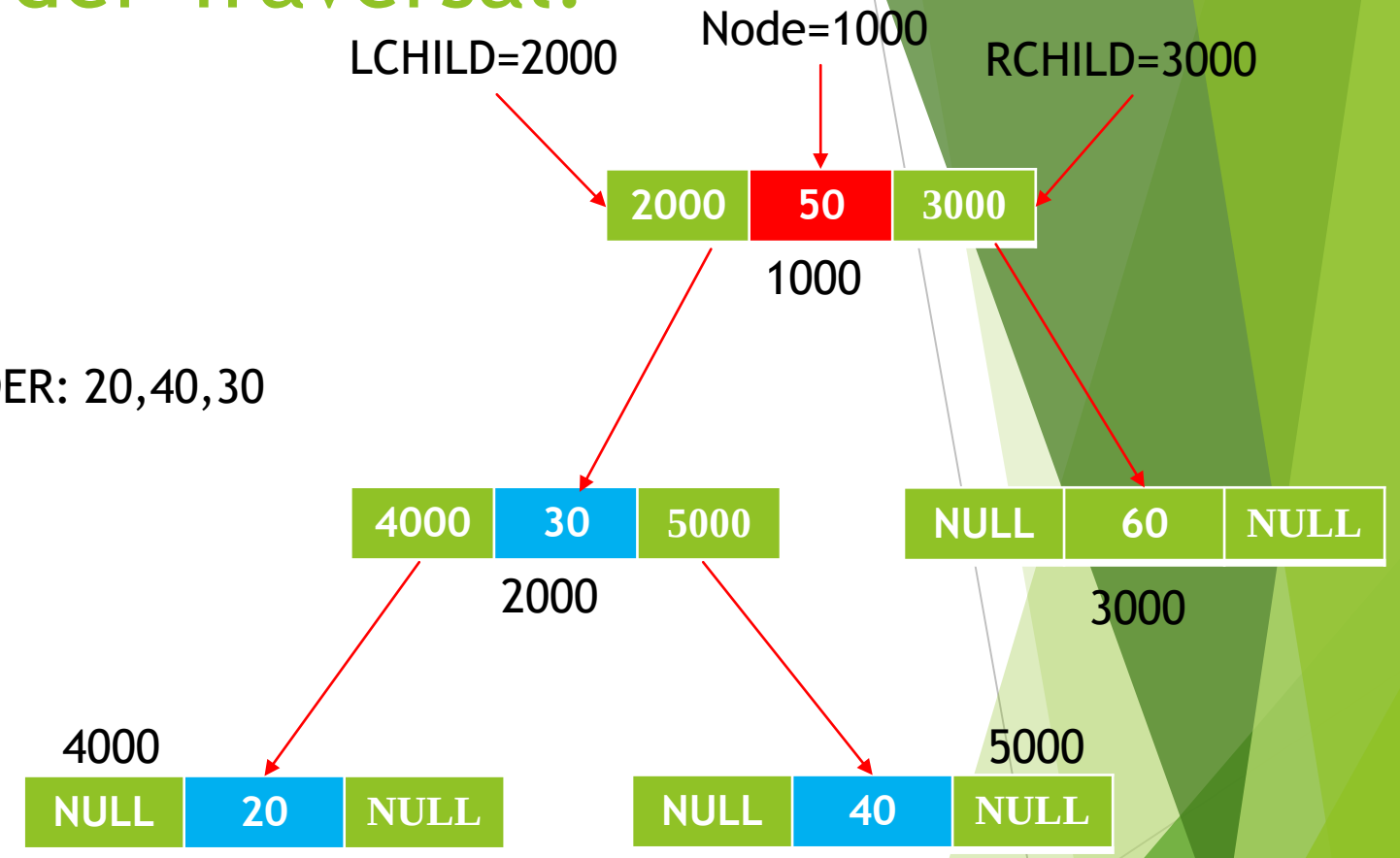
```
Postorder(t =4000)
{
        if(NULL)
        print(20)
```

```
Postorder(t =5000)
{
        if(NULL)
        print(40)

        if(NULL)
```

Algorithm Post-order Traversal:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then
        call PREORDER(LCHILD(T))

    If (T-> R-CHILD ) then
        Call PREORDER(RCHILD(T))

    Print (DATA(T))

End PREORDER
```

POSTORDER: 20,40,30




```
Postorder(t =1000)
{
        if(2000)
        Postorder(2000)
```

```
Postorder(t =2000)
{
        if(4000)
        Postorder(4000)

        if(5000)
        Postorder(5000)

        print(30)
```

```
Postorder(t =4000)
{
        if(NULL)
        print(20)
```

```
Postorder(t =5000)
{
        if(NULL)
        print(40)

        if(NULL)
```

Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

call POSTORDER(LCHILD(T))

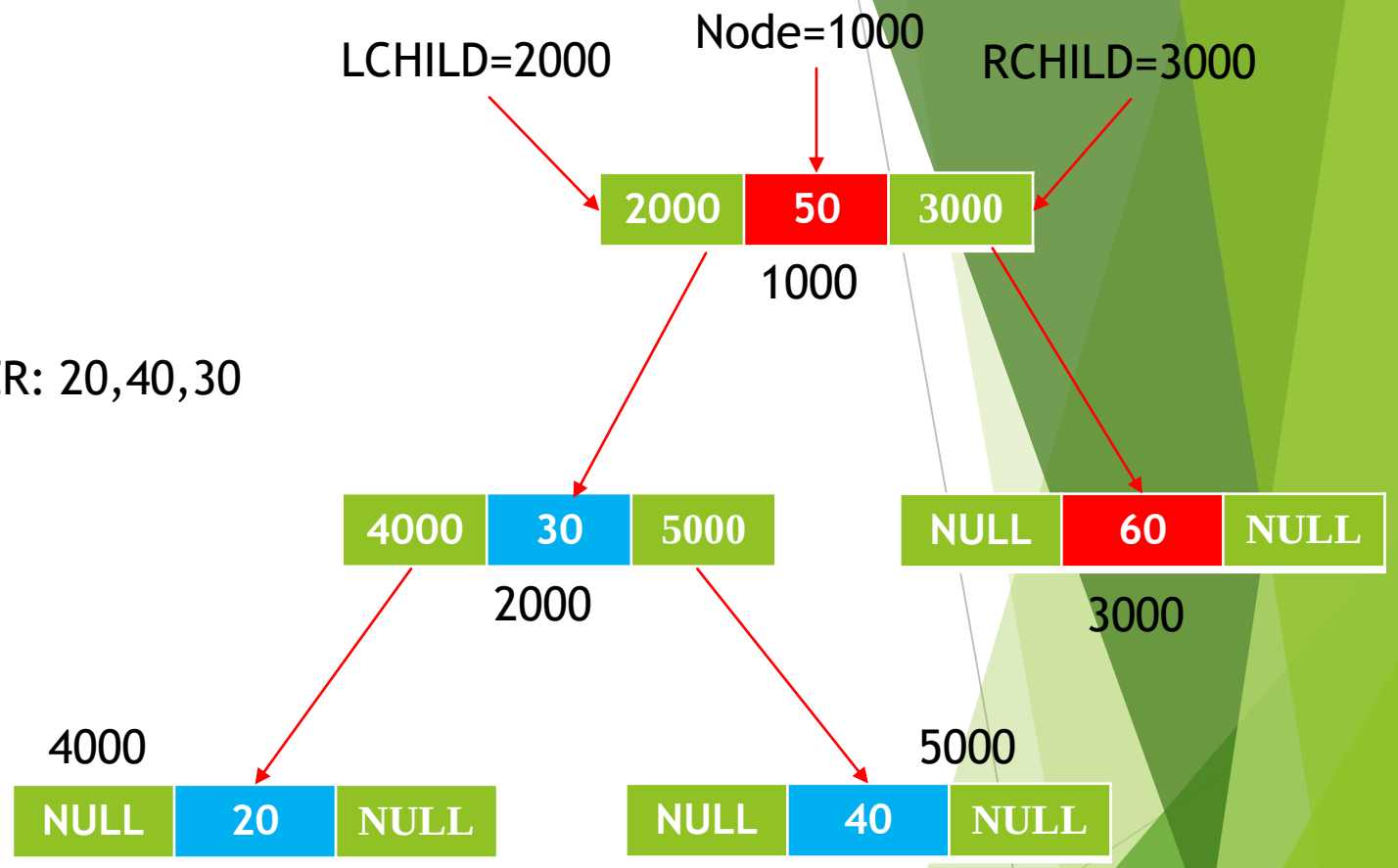
If (T-> R-CHILD) then

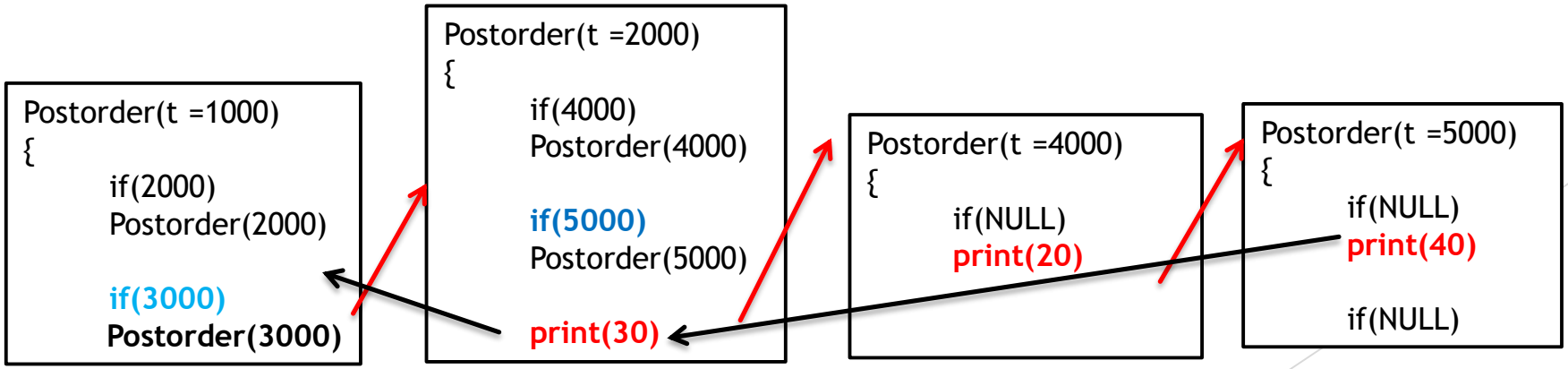
Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER

POSTORDER: 20,40,30





Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

call POSTORDER(LCHILD(T))

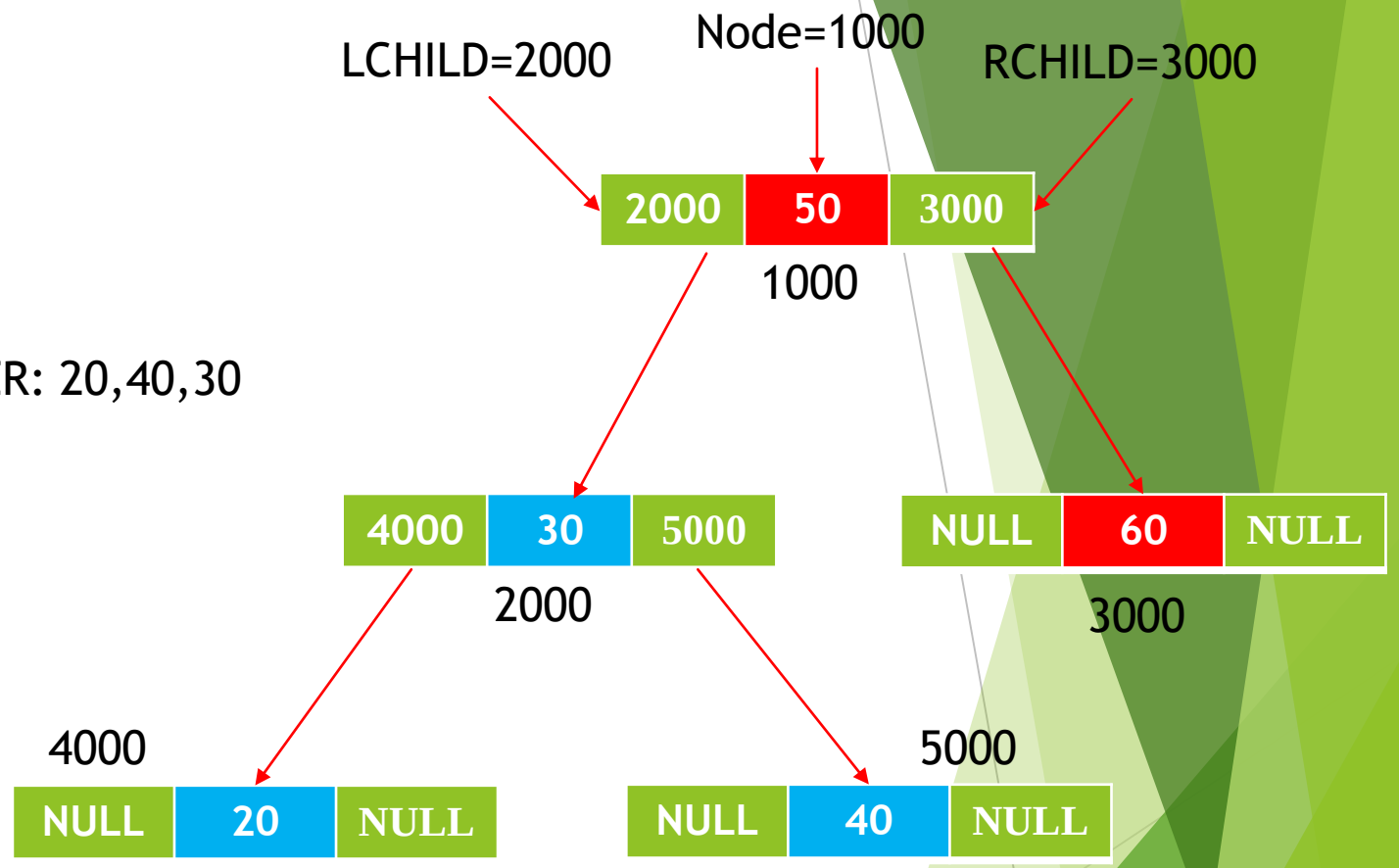
If (T-> R-CHILD) then

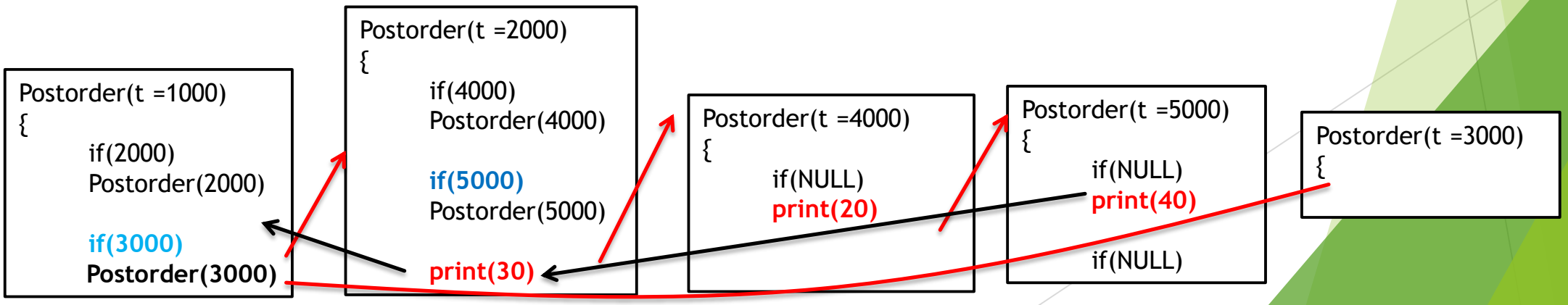
Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER

POSTORDER: 20,40,30





Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

call POSTORDER(LCHILD(T))

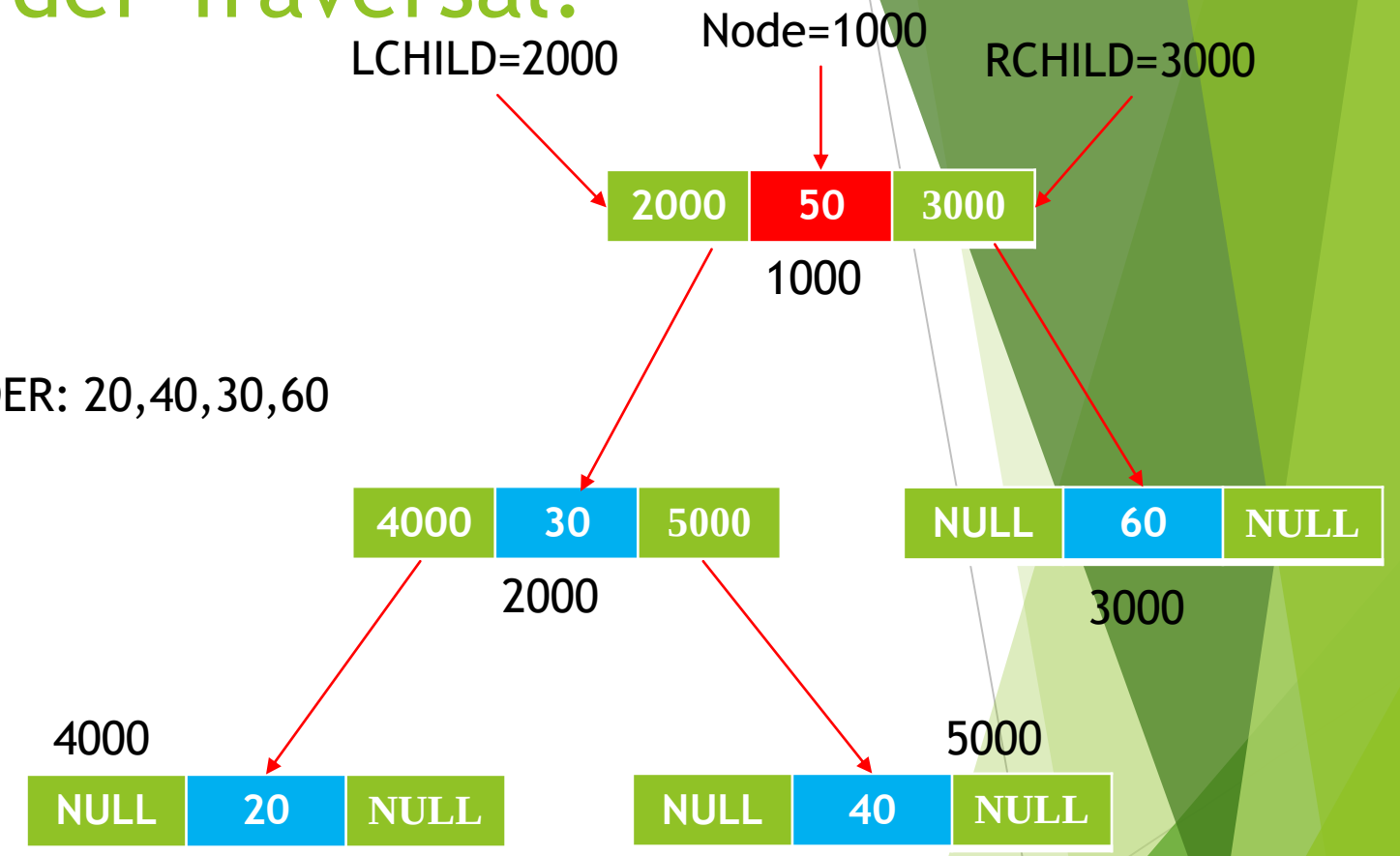
If (T-> R-CHILD) then

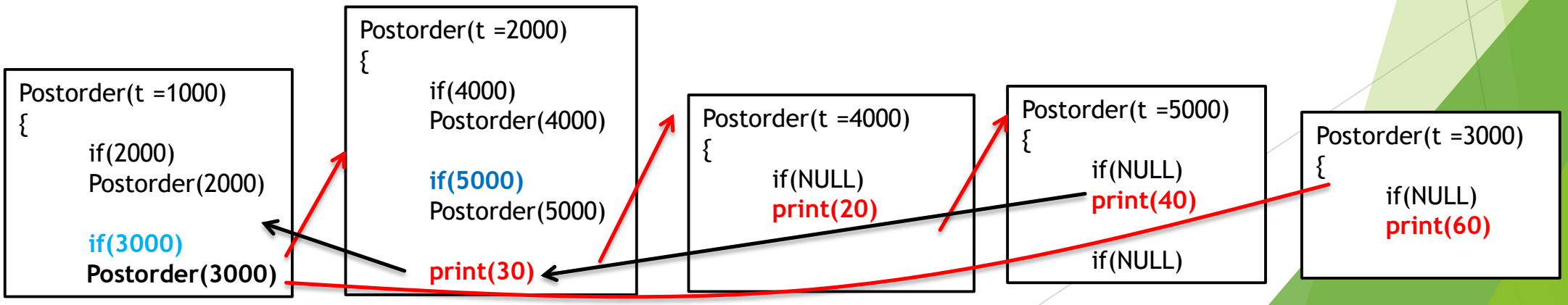
Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER

POSTORDER: 20,40,30,60





Algorithm Post-order Traversal:

POSTORDER(T)

//T is a binary tree where each node has three fields L-CHILD,DATA,R-CHILD//

If (T-> L-CHILD) then

call POSTORDER(LCHILD(T))

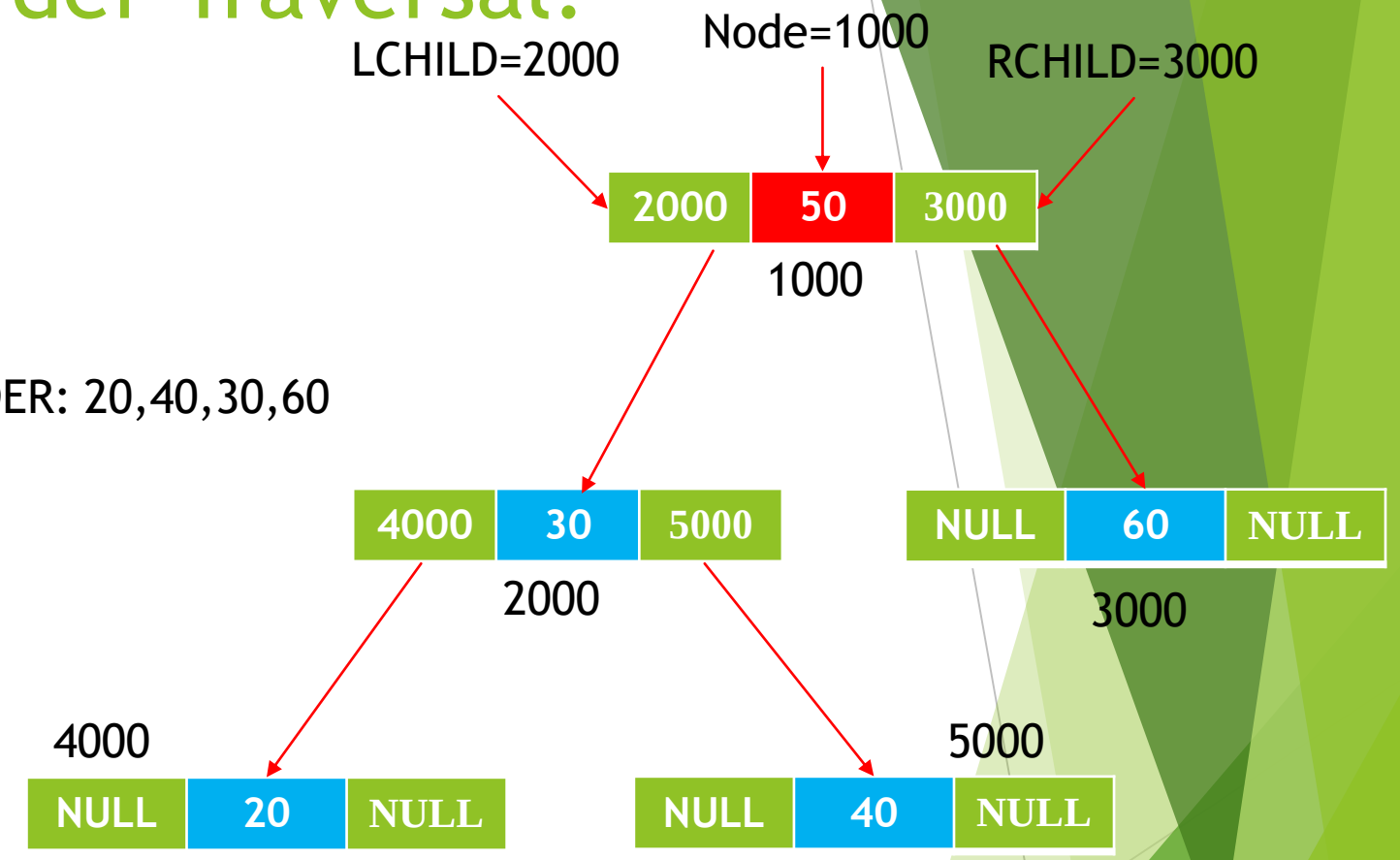
If (T-> R-CHILD) then

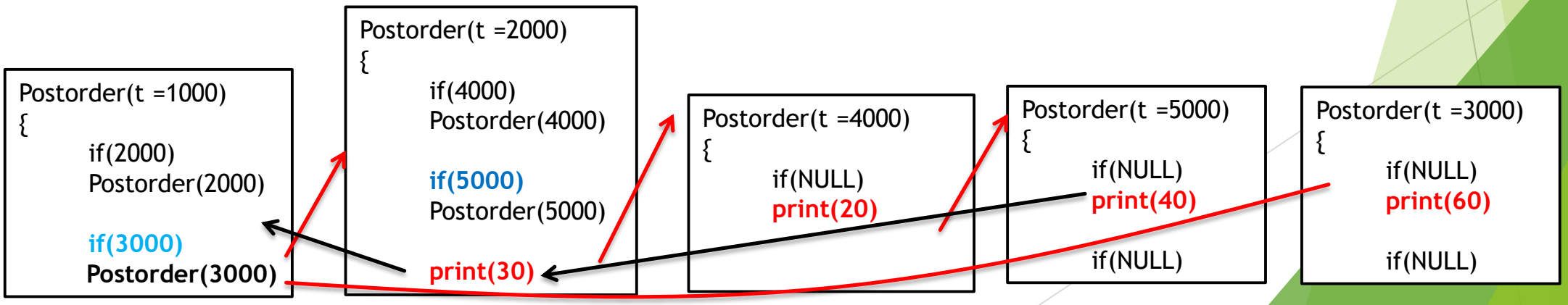
Call POSTORDER(RCHILD(T))

Print (DATA(T))

End POSTORDER

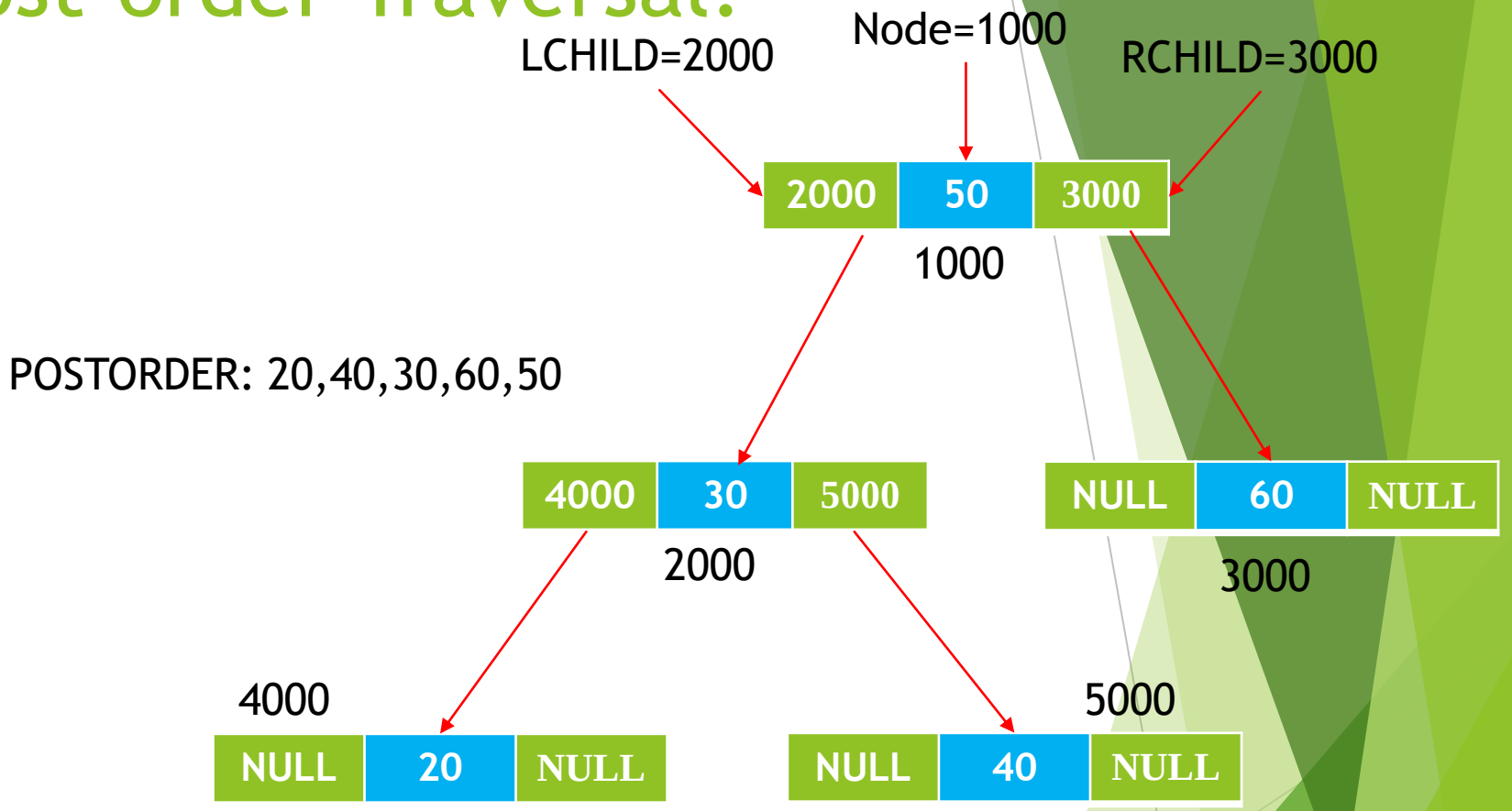
POSTORDER: 20,40,30,60

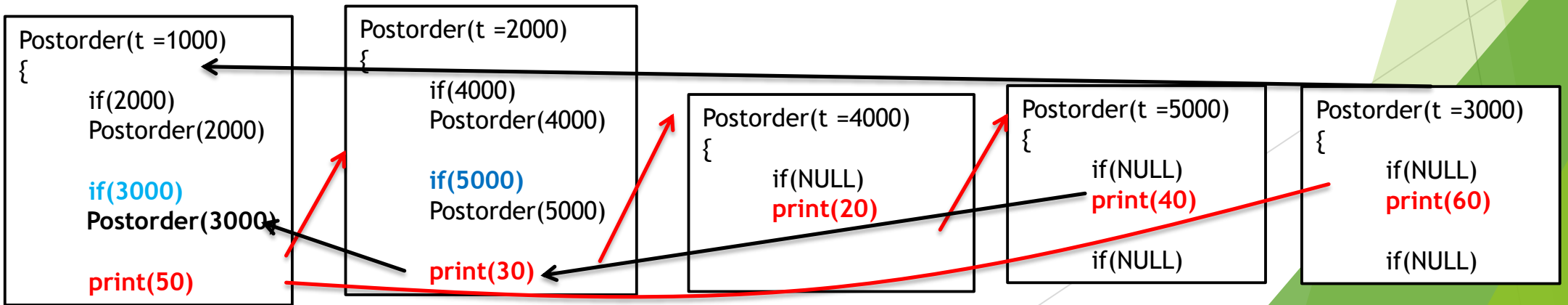




Algorithm Post-order Traversal:

```
POSTORDER(T)

//T is a binary tree where each node has
three fields L-CHILD,DATA,R-CHILD//

    If (T-> L-CHILD ) then

        call POSTORDER(LCHILD(T))


    If (T-> R-CHILD ) then

        Call POSTORDER(RCHILD(T))

    Print (DATA(T))


End POSTORDER
```



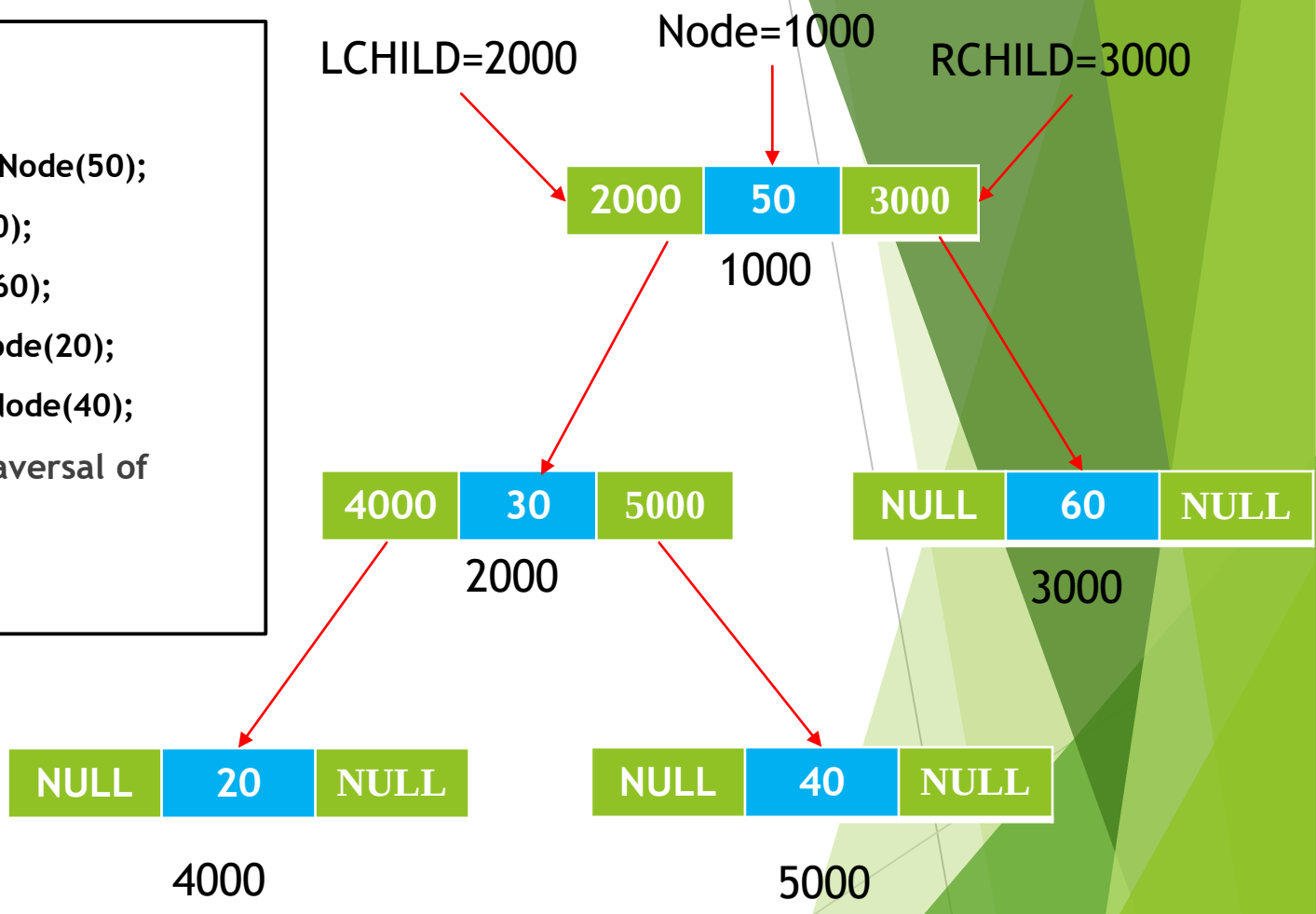


Program For Post-order Traversal:

```
struct Node {

    int data;

    struct Node *left, *right;

    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
```

```
int main()
{
    struct Node* root = new Node(50);
    root->left = new Node(30);
    root->right = new Node(60);
    root->left->left = new Node(20);
    root->left->right = new Node(40);
    cout << "\nPostorder traversal of
binary   tree is \n";
    printPostorder(root);
```

```
void printPostorder(struct Node*
node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}
```



POSTORDER: 20,40,30,60,50

OUTPUT:

```
Preorder traversal of binary tree is
50 30 20 40 60
Inorder traversal of binary tree is
20 30 40 50 60
Postorder traversal of binary tree is
20 40 30 60 50

...Program finished with exit code 0
Press ENTER to exit console.
```

THANK YOU!