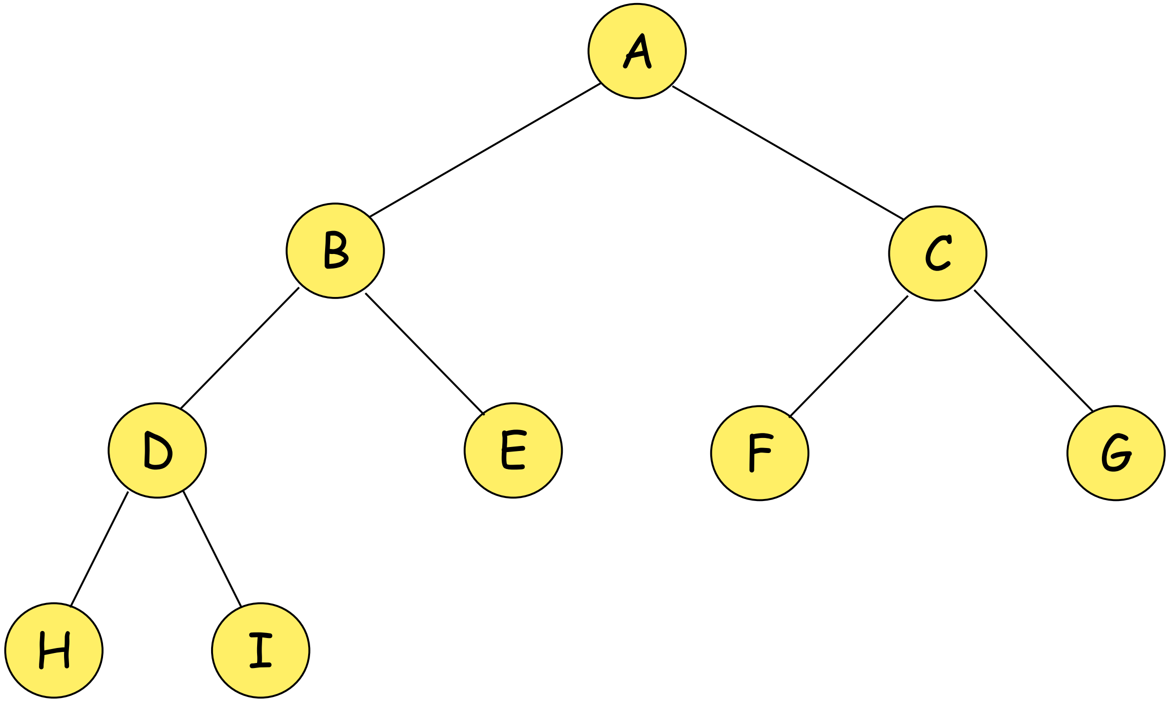
Threaded Binary Tree

- Threads
  - In a linked representation of a binary tree, there are more NULL links than actual pointers.
  - In a binary tree with n nodes containing 2n links, there are n+1 NULL links.
  - Perlis and Thornton devised a way to make use of NULL links.
  - Here the NULL links are replaced by pointers, called *threads*, to other nodes in the tree.

Threaded Binary Tree

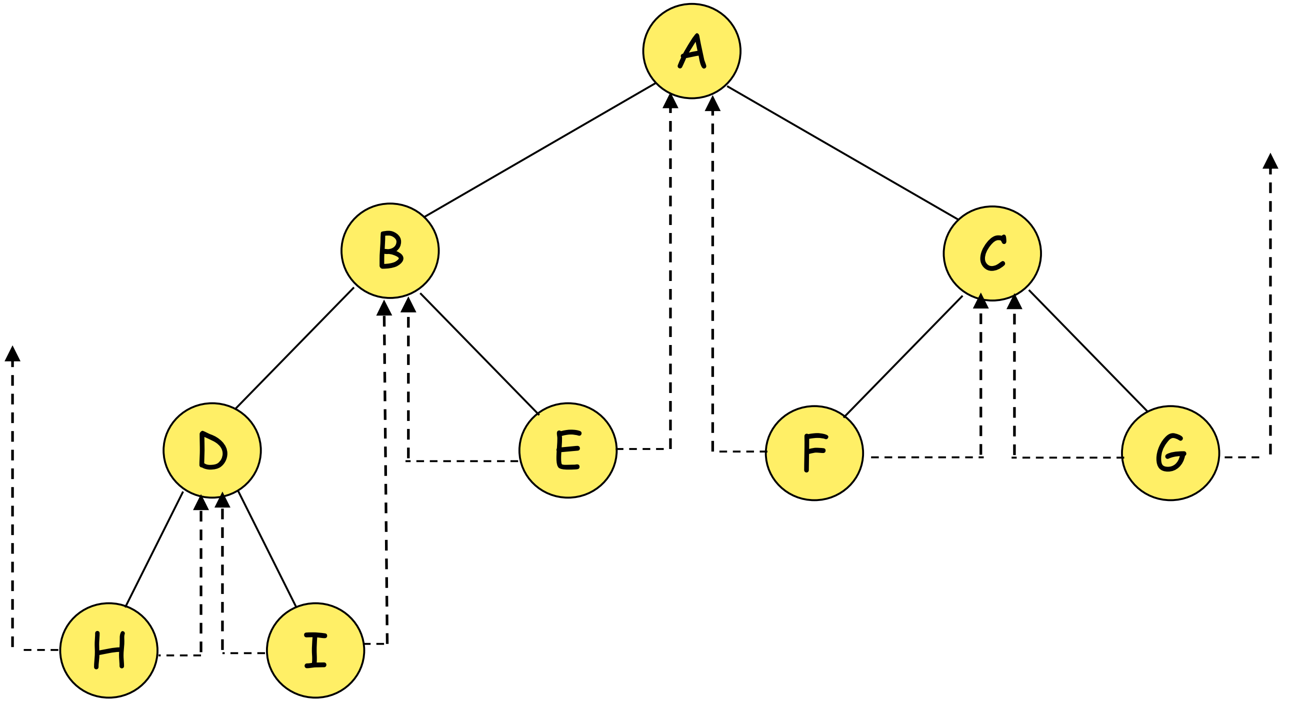
- Threading Rules
  - A NULL RightChild field at node p is replaced by a pointer to the node that would be visited after p when traversing the tree in inorder. That is, it is replaced by the inorder successor of p.
  - A NULL LeftChild link at node p is replaced by a pointer to the node that immediately precedes node p in inorder (i.e., it is replaced by the inorder predecessor of p).

A Binary Tree



Inorder sequence: H, D, I, B, E, A, F, C, G

Threaded Tree Corresponding to Given Binary Tree



Inorder sequence: H, D, I, B, E, A, F, C, G

Threads

- To distinguish between normal pointers and threads, two boolean fields, LeftThread and RightThread, are added to the record in memory representation.
  - t->leftThread= TRUE

    => t->lchild is a **thread**
  - t->leftThread= FALSE

    ⇒t->lchild is a **pointer** to the left child.
  - t->rightThread= TRUE

    => t->rchild is a **thread**
  - t->rightThread= FALSE

    => t->rchild is a **pointer** to the right child.

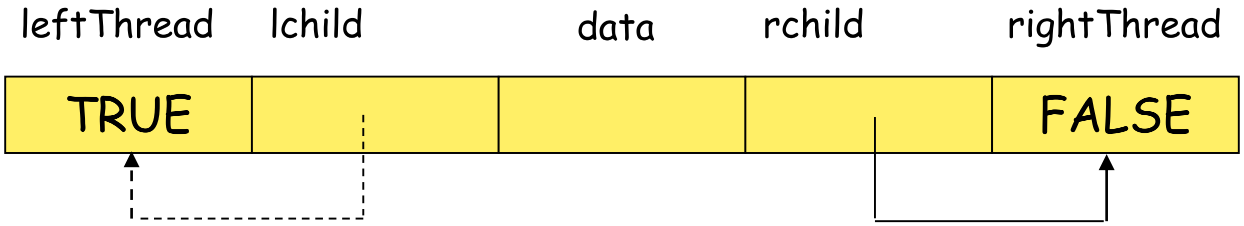
Threaded Binary Tree Node Structure Declaration

```
typedef struct threadedTree *threadedPointer;

struct threadedTree{
        short int leftThread;
        threadedPointer lchild;
        char data;
        threadedPointer rchild;
        short int rightThread;
};
```

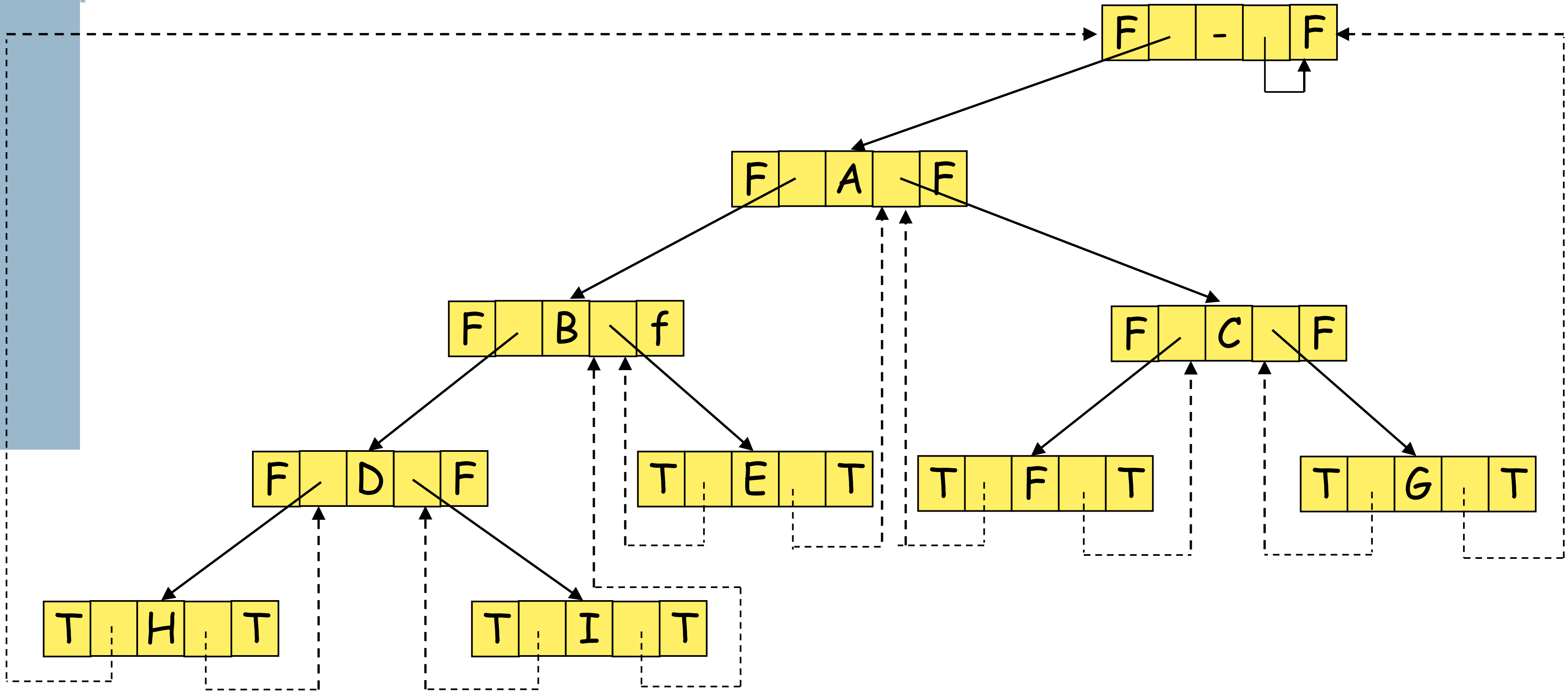
Threads (Cont.)

- To avoid dangling threads, a head node is used in representing a binary tree.
- The original tree becomes the left subtree of the head node.
- Empty Binary Tree



leftThread	lchild	data	rchild	rightThread
TRUE				FALSE

Memory Representation of Threaded Tree



Finding the inorder successor without stack

- By using the threads, we can perform an inorder traversal without making use of a stack.

```
threadedPointer insucc(threadedPointer node)
{//Return the inorder successor of node
     threadedPointer temp = node-> rchild;
    if (node->rightThread==FALSE)
          while (temp->leftThread==FALSE)
                      temp = temp -> lchild;
    return temp;
    }
```

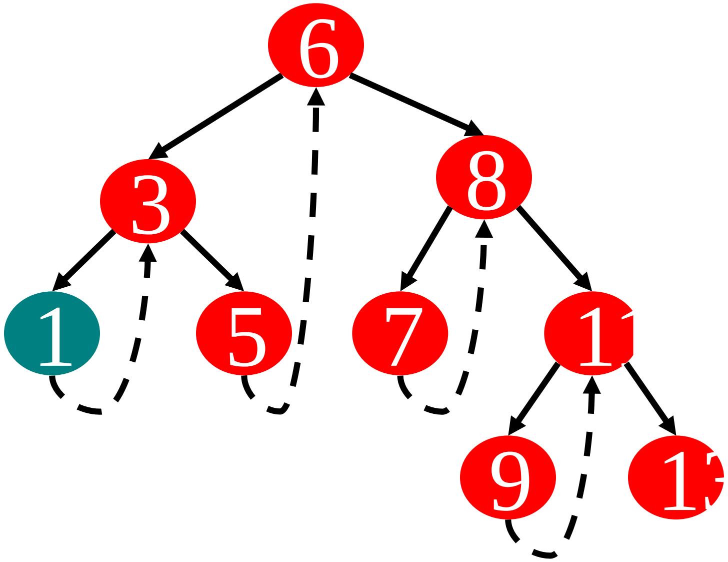
Inorder Traversal of a threaded Binary Tree

```
void tinorder(threadedPointer treehead)
{
   threadedPointer temp = treehead;
   while(1){
      temp = insucc(temp);
      if (temp == treehead) break;
      printf(“%c”, temp->data);
   }
}
```

Threaded Tree Traversal

- We start at the leftmost node in the tree, print it, and follow its right thread
- If we follow a thread to the right, we output the node and continue to its right
- If we follow a link to the right, we go to the leftmost node, print it, and continue

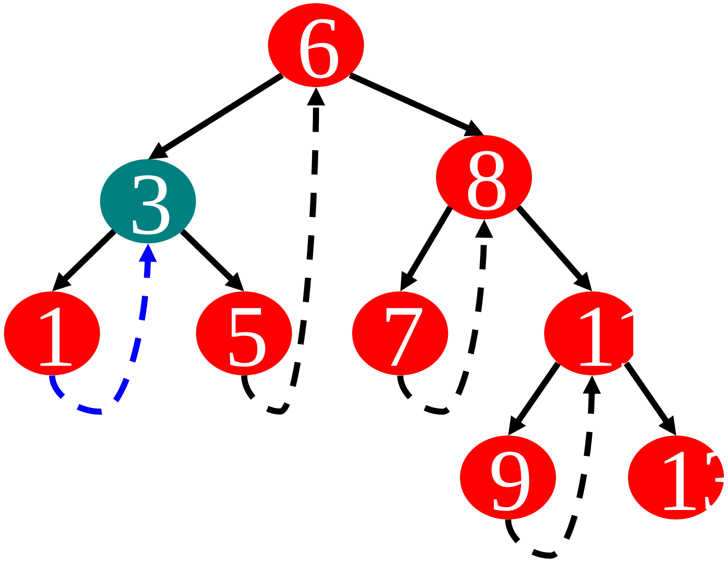
Threaded Tree Traversal



Output
1

Start at leftmost node, print it

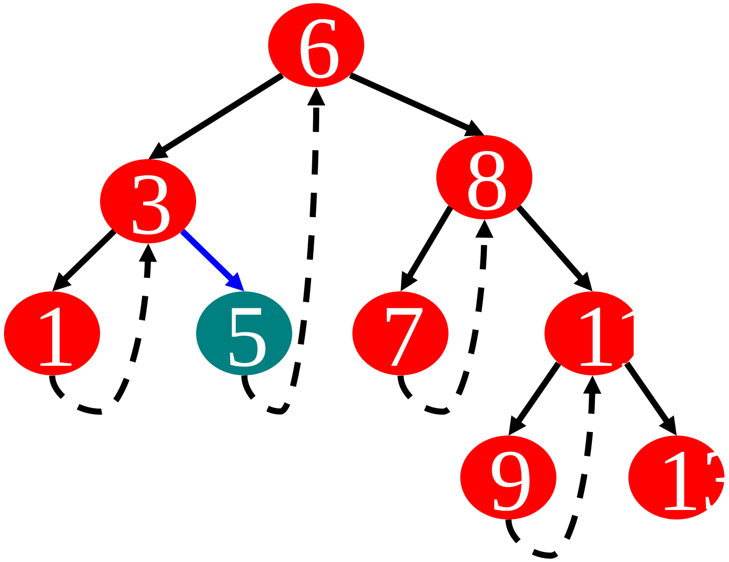
Threaded Tree Traversal



Output
1
3

Follow thread to right, print node

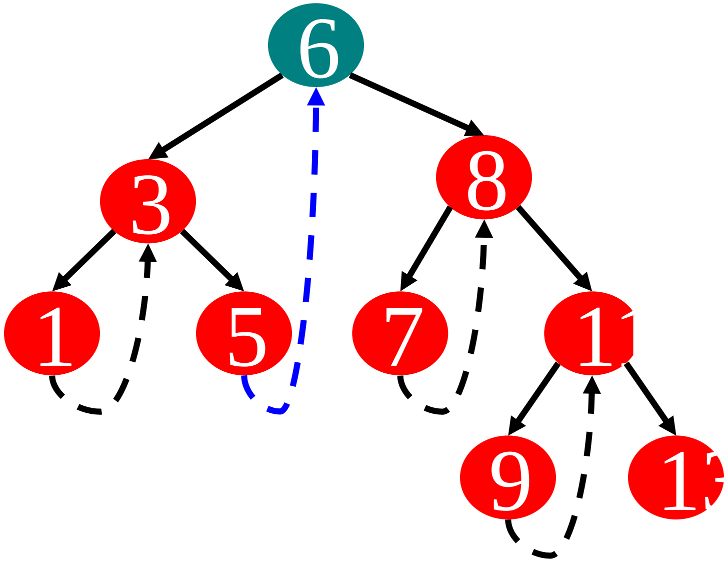
Threaded Tree Traversal



Output
1
3
5

Follow link to right, go to leftmost node and print

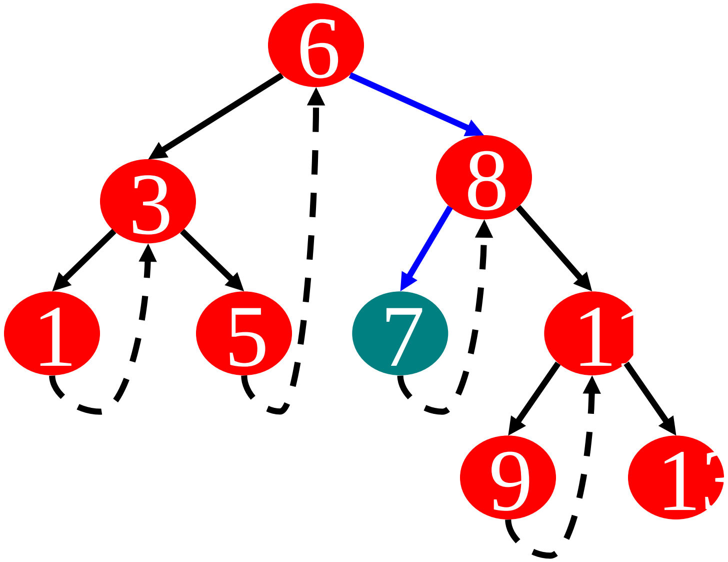
Threaded Tree Traversal



Output
1
3
5
6

Follow thread to right, print node

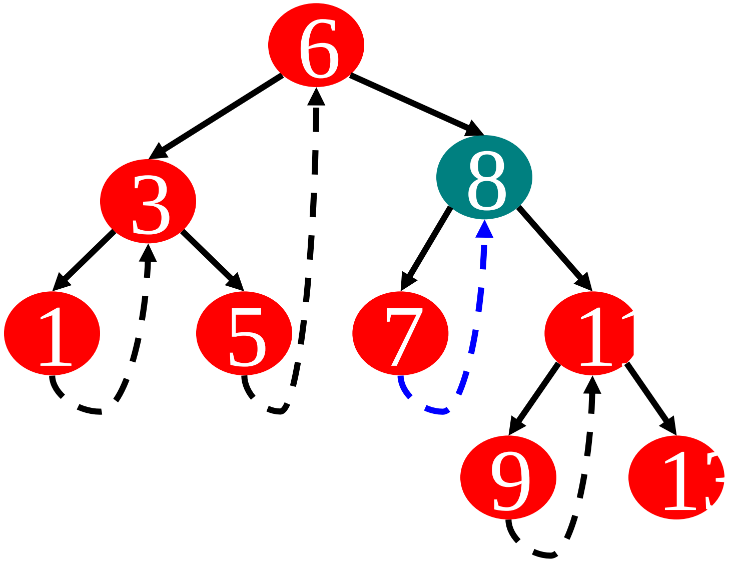
Threaded Tree Traversal



Output
1
3
5
6
7

Follow link to right, go to
leftmost node and print

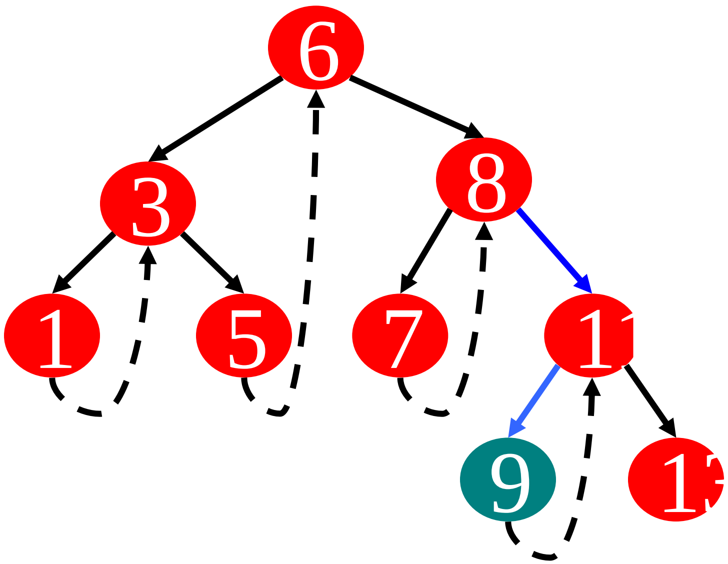
Threaded Tree Traversal



Output
1
3
5
6
7
8

Follow thread to right, print node

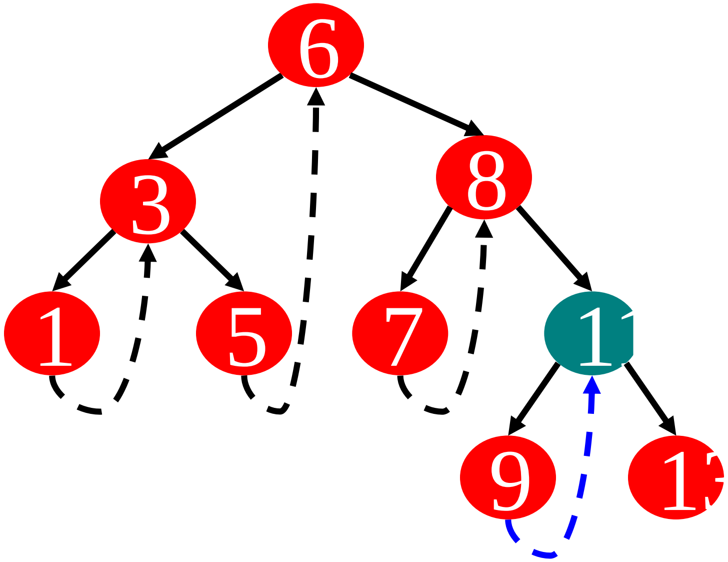
Threaded Tree Traversal



Output
1
3
5
6
7
8
9

Follow link to right, go to leftmost node and print

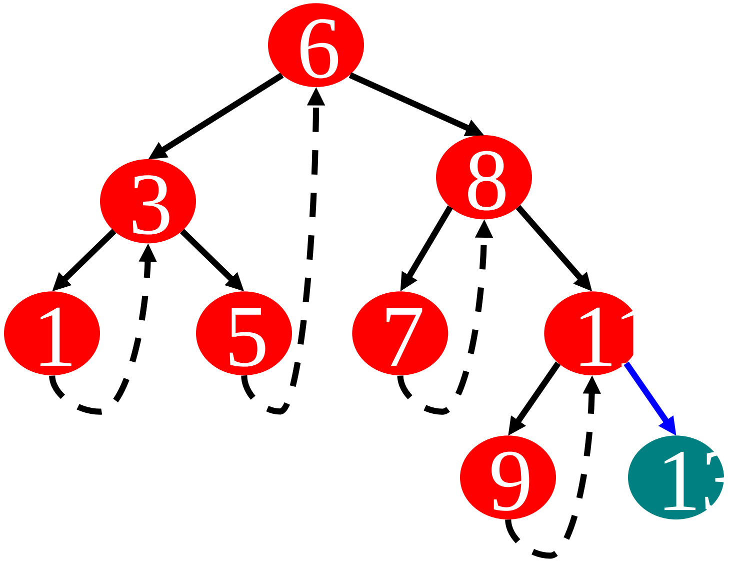
Threaded Tree Traversal



Output
1
3
5
6
7
8
9
11

Follow thread to right, print node

Threaded Tree Traversal



Output
1
3
5
6
7
8
9
11
13

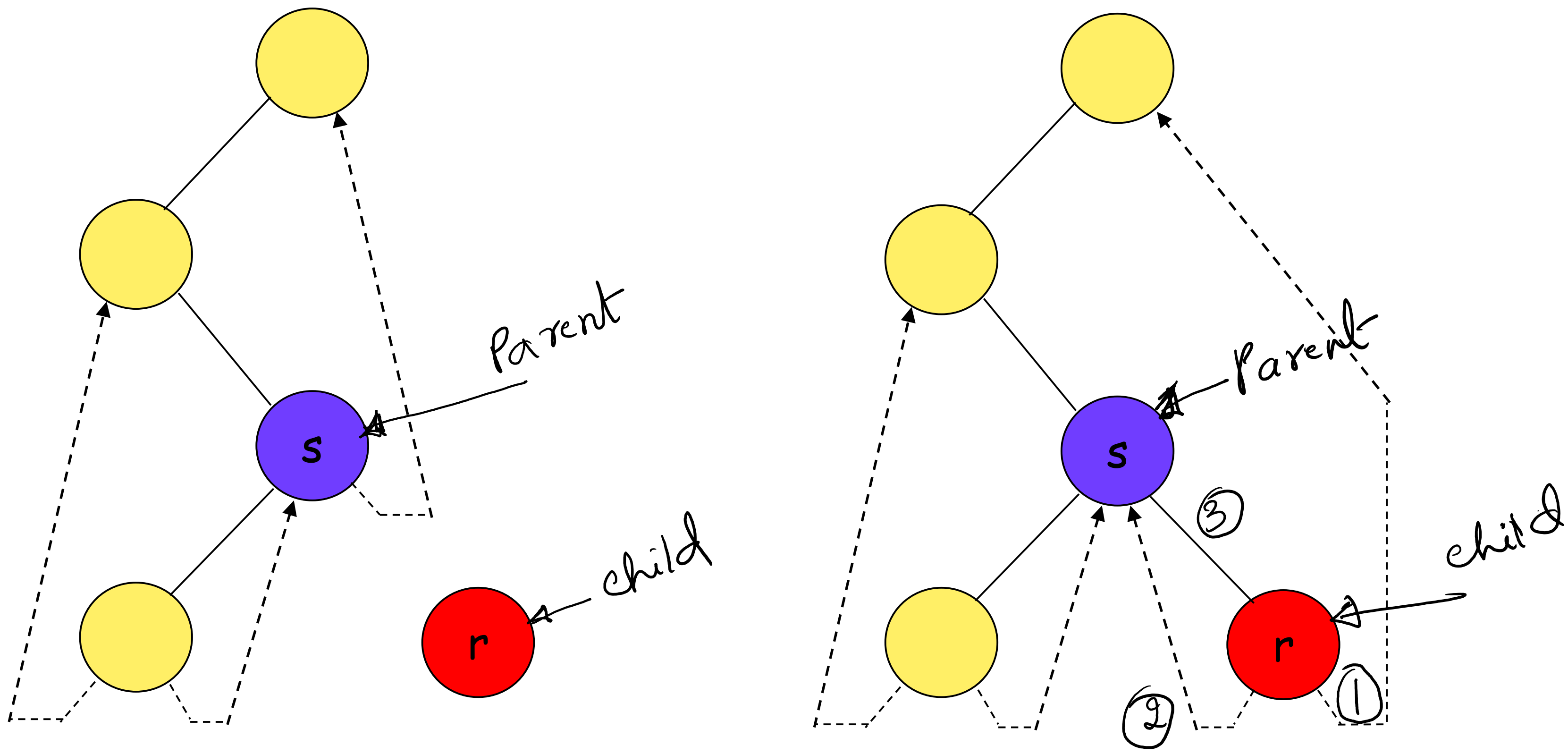
Follow link to right, go to leftmost node and print

Inserting A Node to AThreaded Binary Tree

- Inserting a node r as the right child of a node s.
  - If s has an empty right subtree, then the insertion is simple (as shown in diagram next slide)
  - If the right subtree of s is not empty, then, this right subtree is made the right subtree of r after insertion. When this is done, r becomes the inorder predecessor of a node that has a leftThread==TRUE field, and consequently there is an thread which has to be updated to point to r. The node containing this thread was previously the inorder successor of s. Figure illustrates the insertion for this case.

Insertion of r As A Right Child of s in A Threaded Binary Tree

case (a) (Empty rt subtree) for s

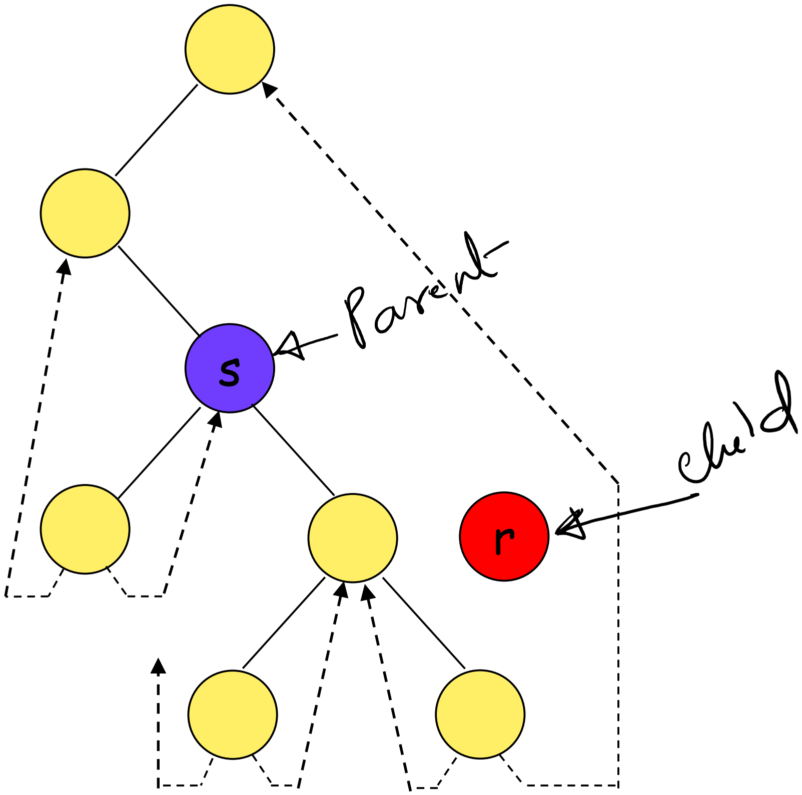


before

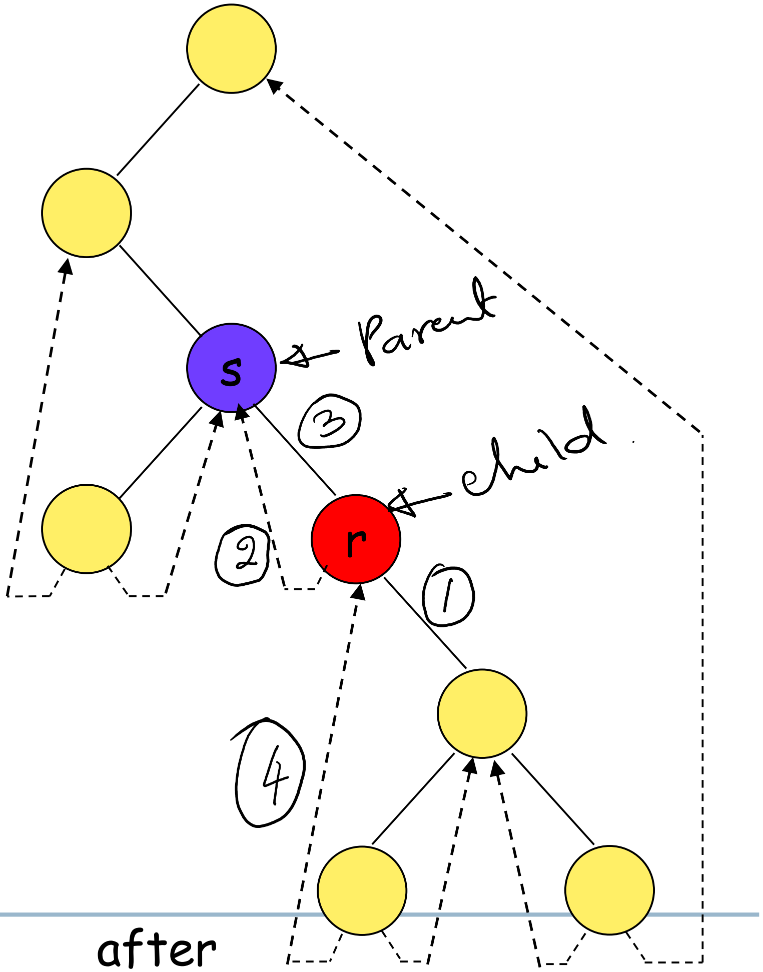
after

Insertion of r As A Right Child of s in A Threaded Binary Tree (Cont.)

case (b) (nonempty right subtree for s)



before



after