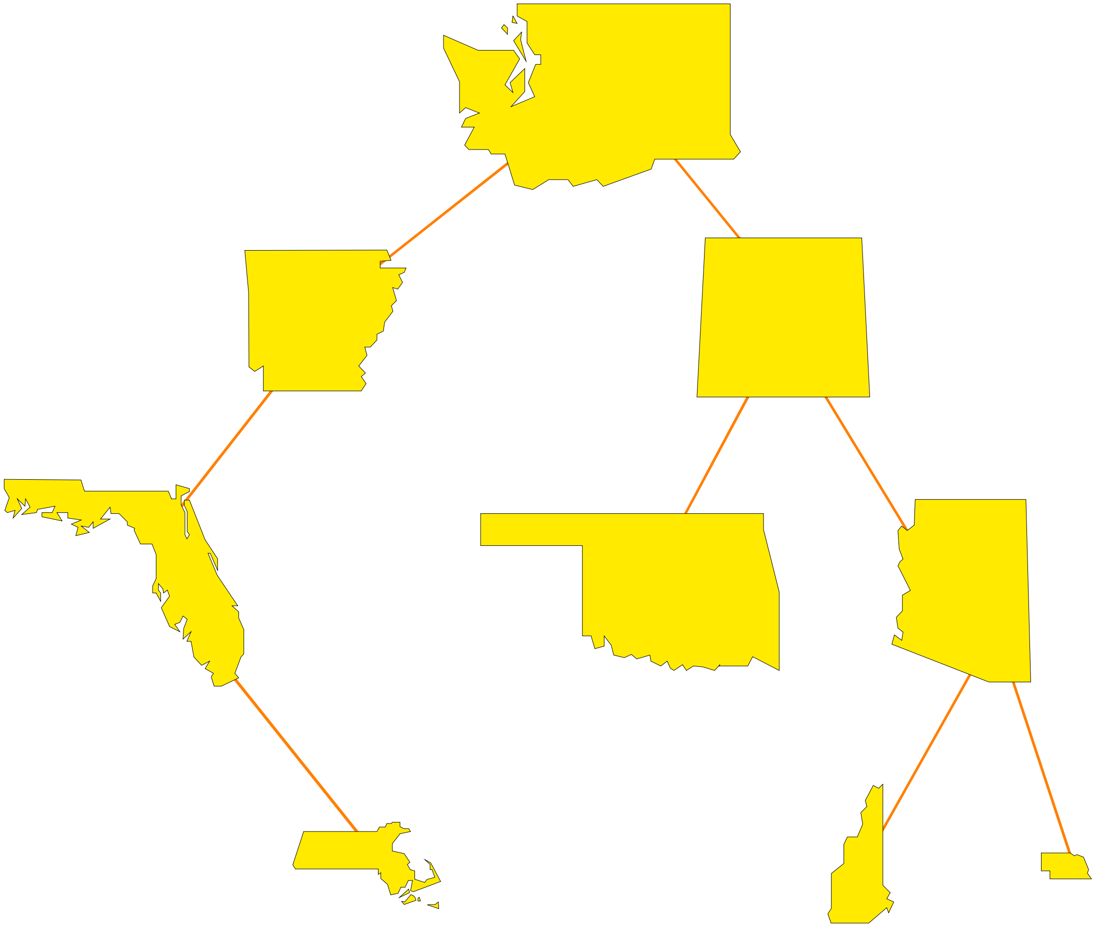
Trees

Binary Trees

- A binary tree has **nodes**, similar to nodes in a linked list structure.
- **Data** of one sort or another may be stored at each node.
- But it is the **connections** between the nodes which characterize a binary tree.

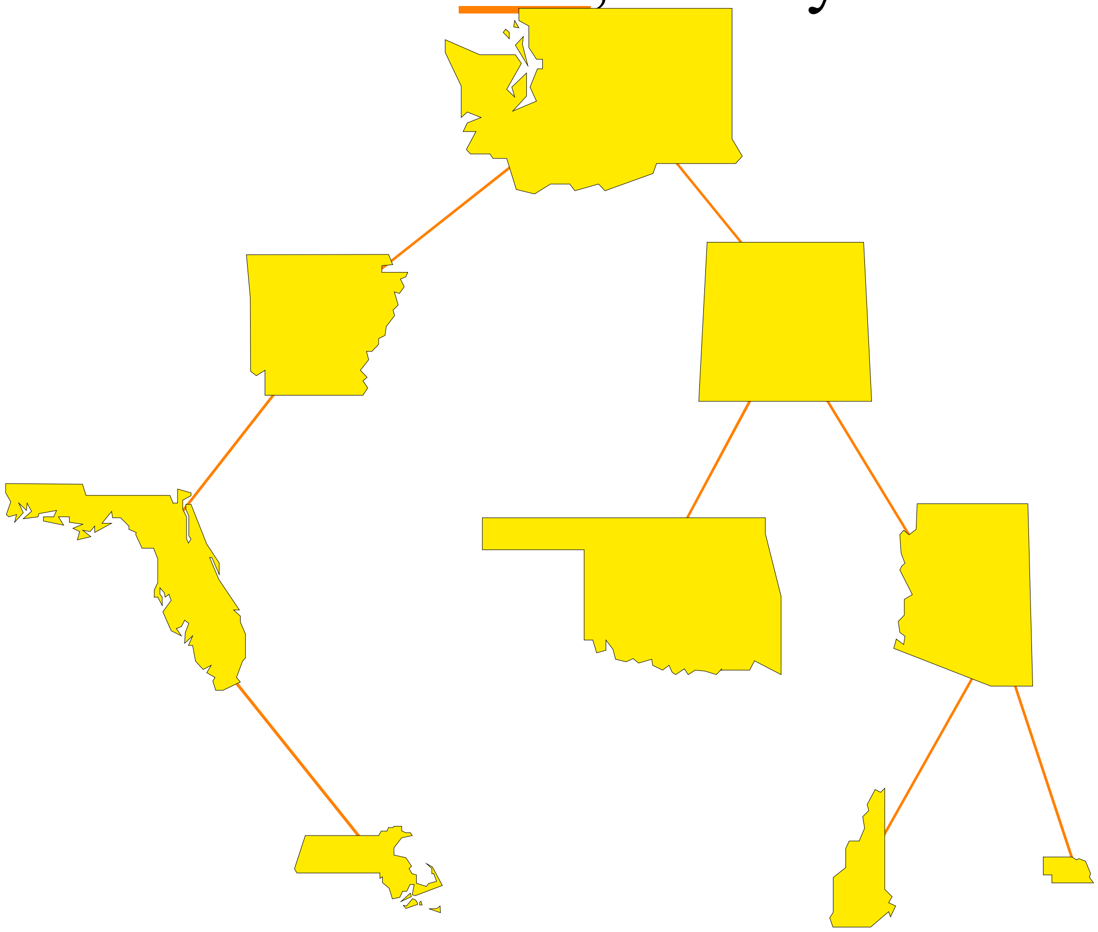
A Binary Tree of States

In this example, the data contained at each node is one of the 50 states.



A Binary Tree of States

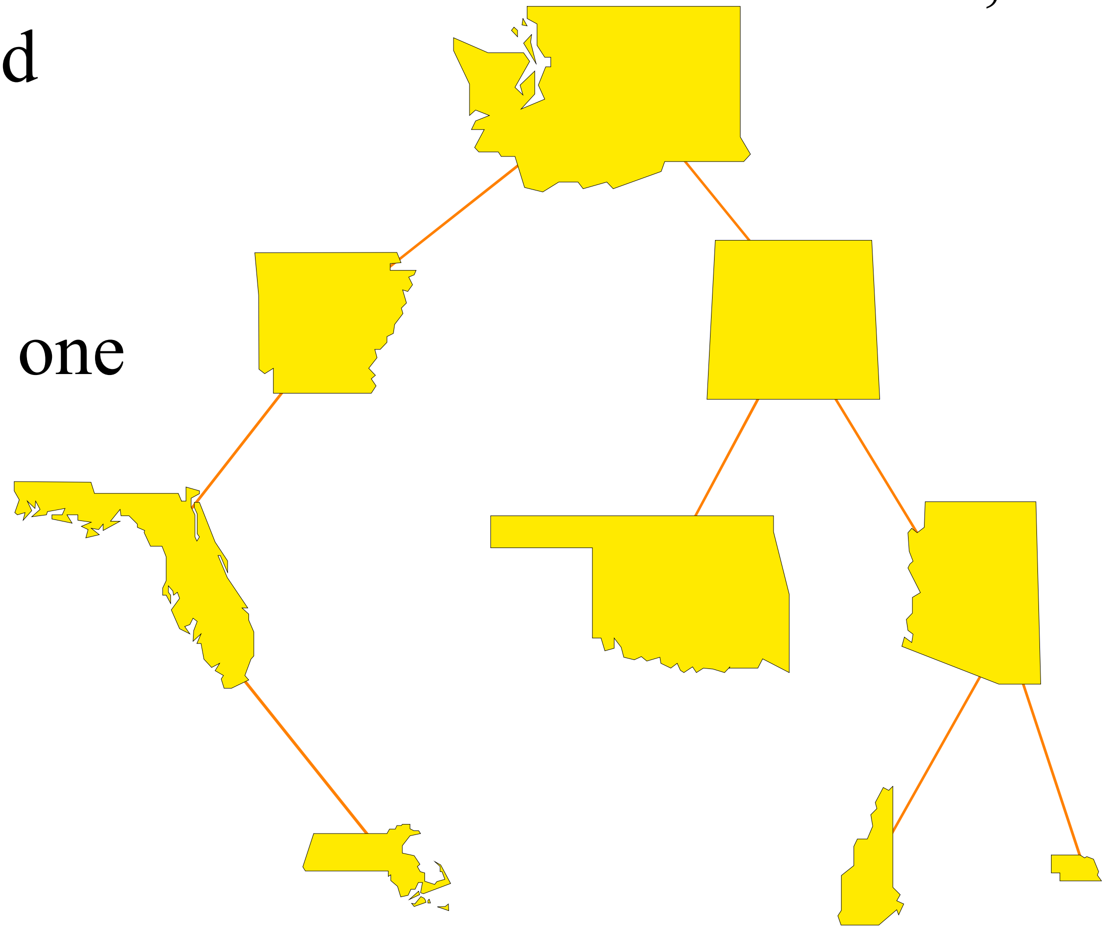
Each tree has a special node called its **root**, usually drawn at the top.



A Binary Tree of States

Each node is permitted to have two links to other nodes, called the **left child** and the **right child**.

Some nodes have only one child.

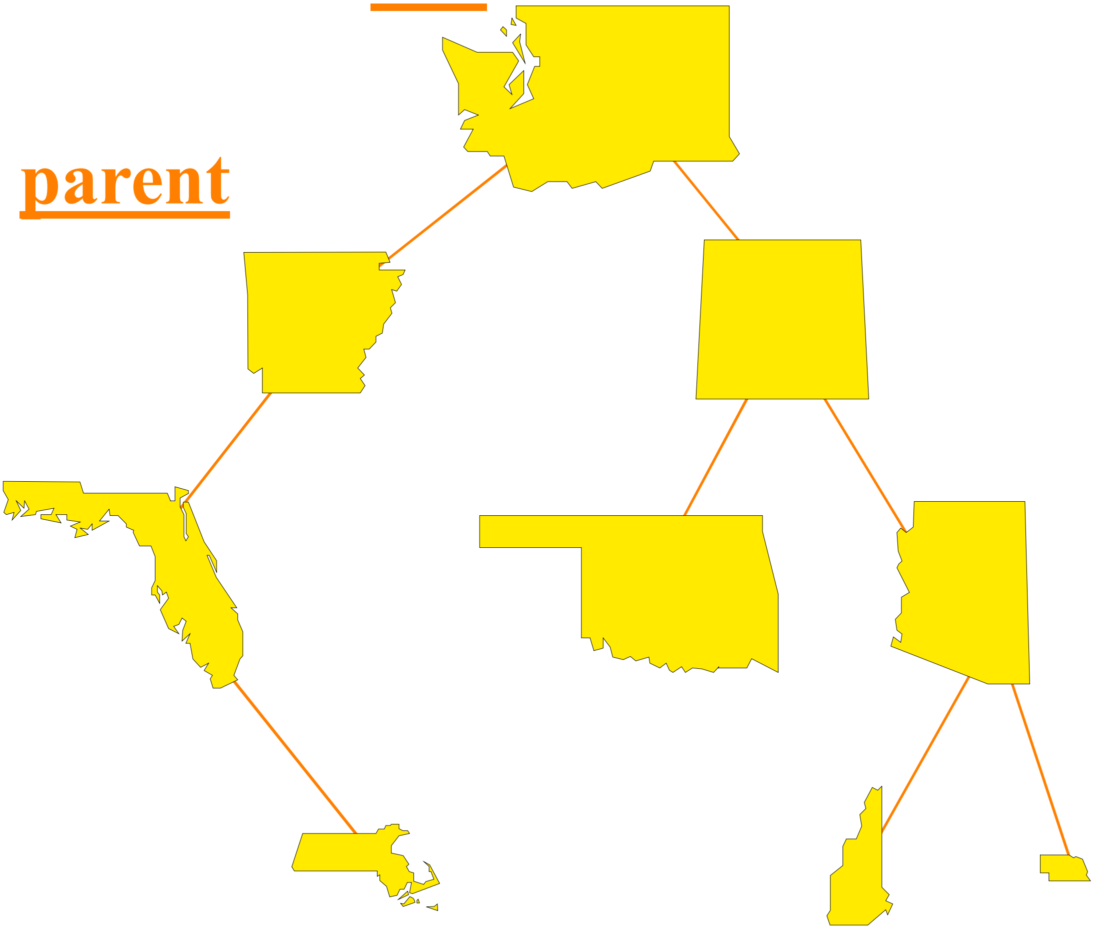


SANTA CLARA UNIVERSITY
THE JESUIT UNIVERSITY IN SILICON VALLEY

A Binary Tree of States

A node with no children is called a **leaf**.

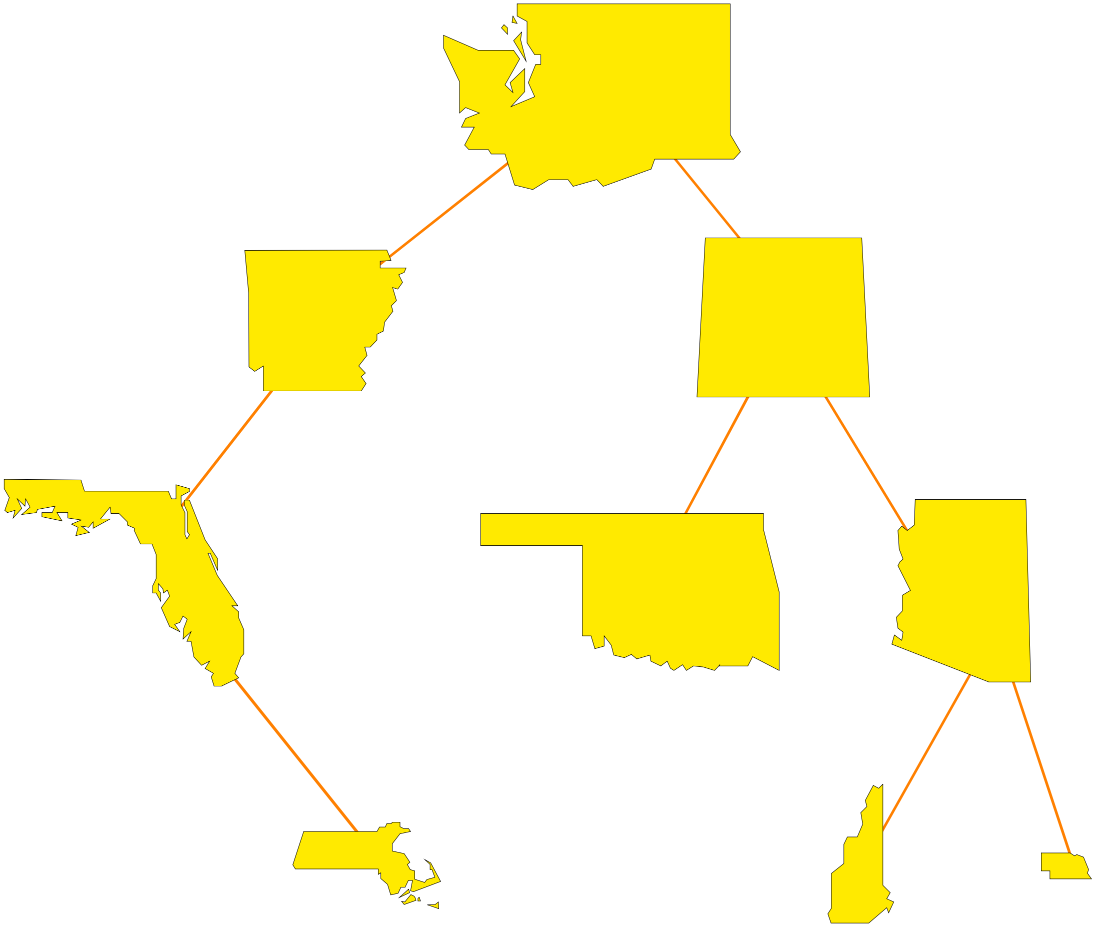
Each node is called the **parent** of its children.



A Binary Tree of States

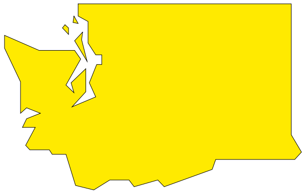
Two rules about parents:

- The root has no parent.
- Every other node has exactly one parent.



Complete Binary Trees

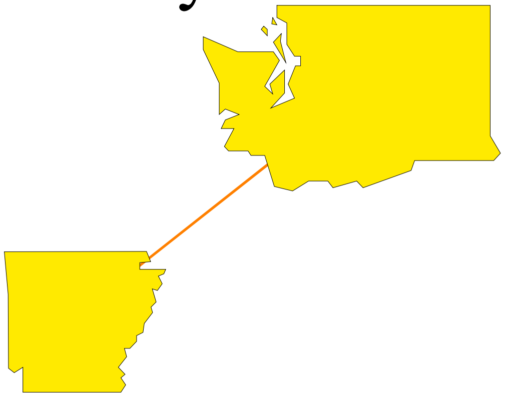
A complete binary tree is a special kind of binary tree which will be useful to us.



When a complete binary tree is built, its first node must be the root.

Complete Binary Trees

The second node of a complete binary tree is always the left child of the root...

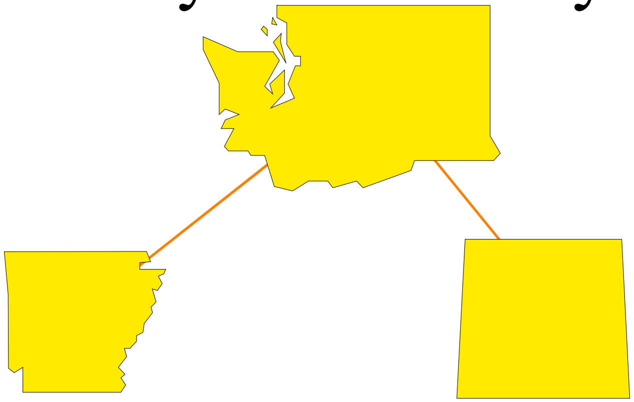


Complete Binary Trees

The second node of a complete binary tree is always the left child of the root...

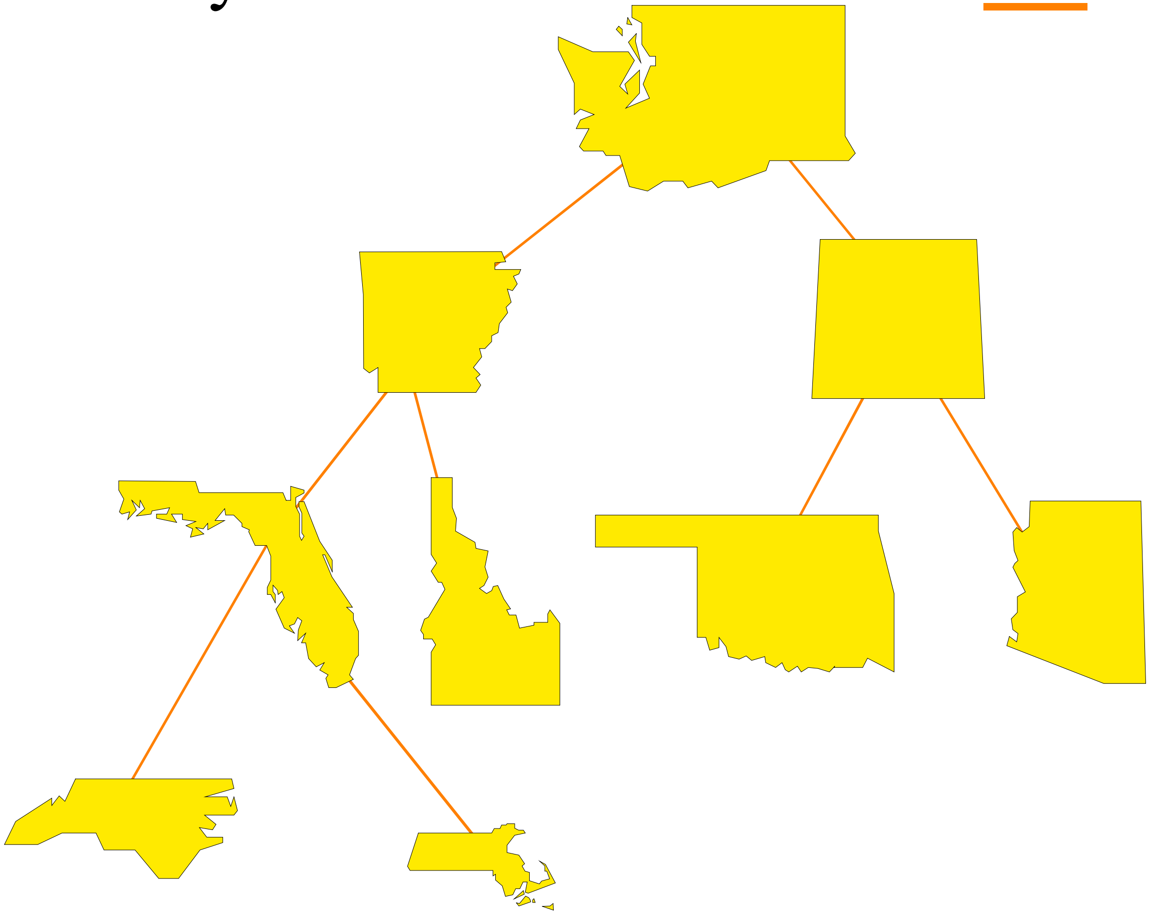
... and the third node is always

the right child of the root.

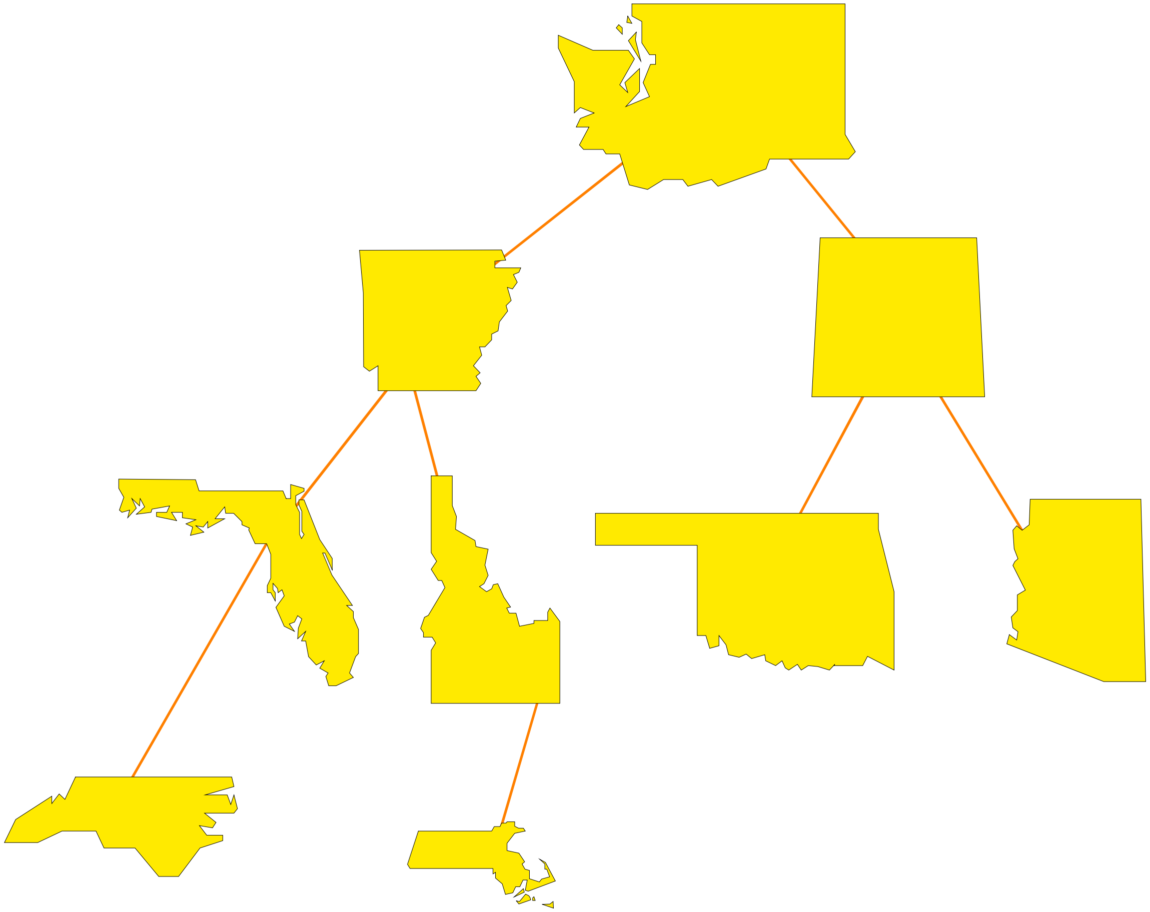


Complete Binary Trees

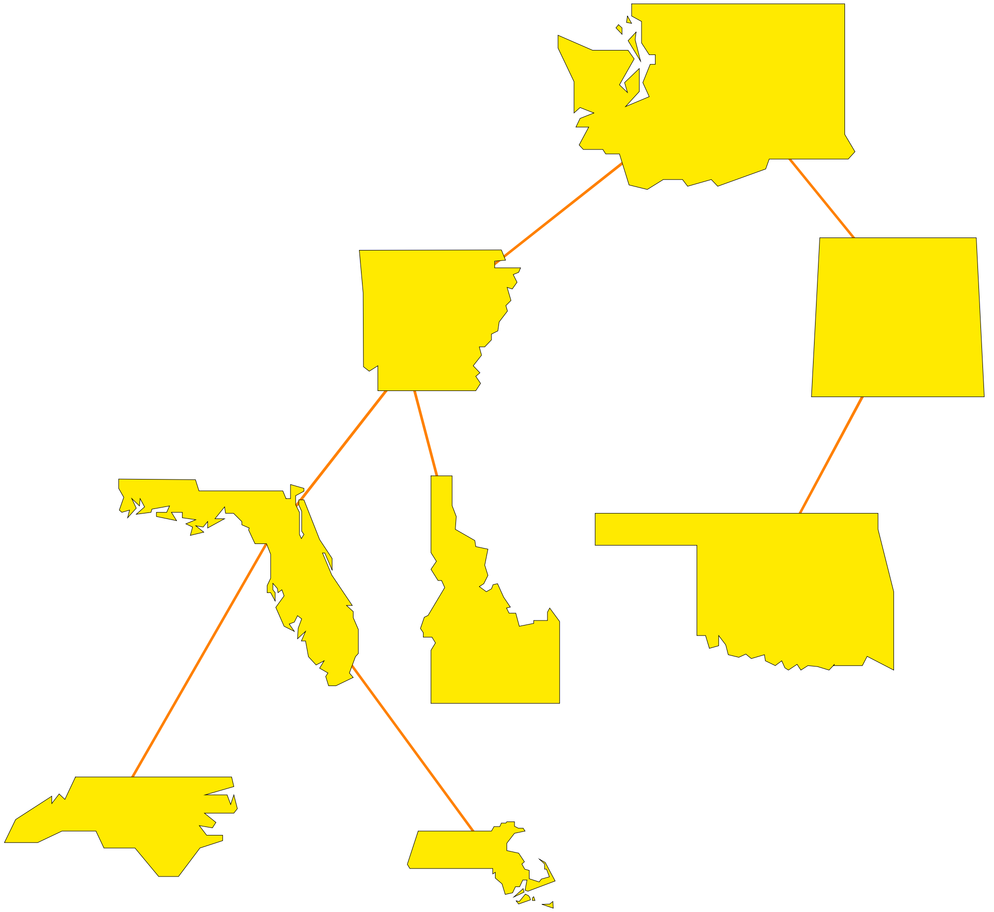
The next nodes must always fill the next level from **left to right**.



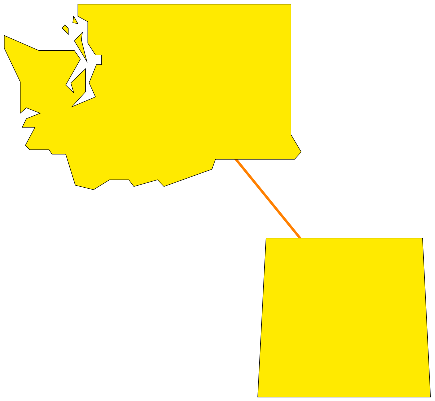
Is This Complete?



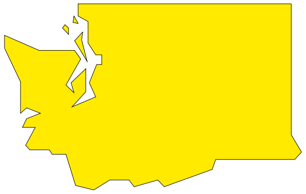
Is This Complete?



Is This Complete?

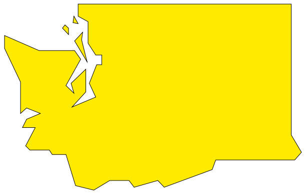


Is This Complete?



Full Binary Trees

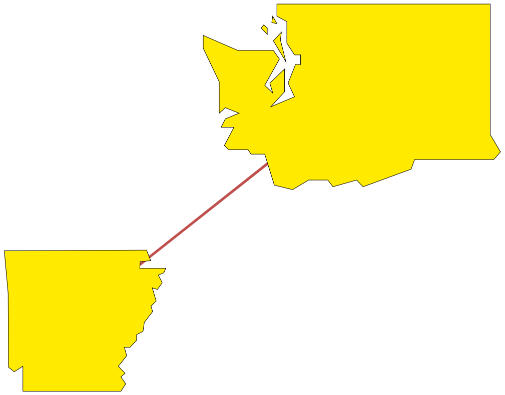
A full binary tree is a special kind of complete binary tree



When a full binary tree is built, its first node must be the root.

Full Binary Trees

The second node of a full binary tree is always the left child of the root...

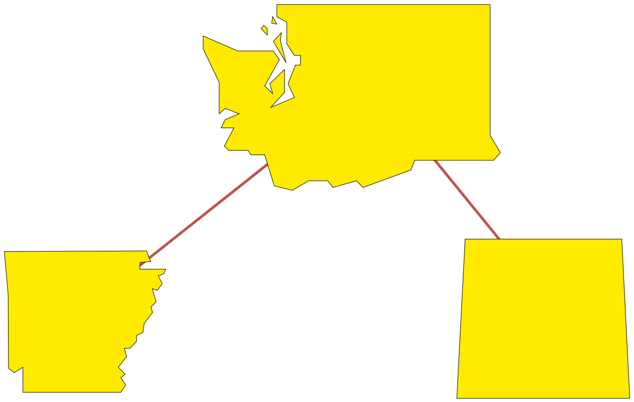


not FULL yet

Full Binary Trees

The second node of a full binary tree is always the left child of the root...

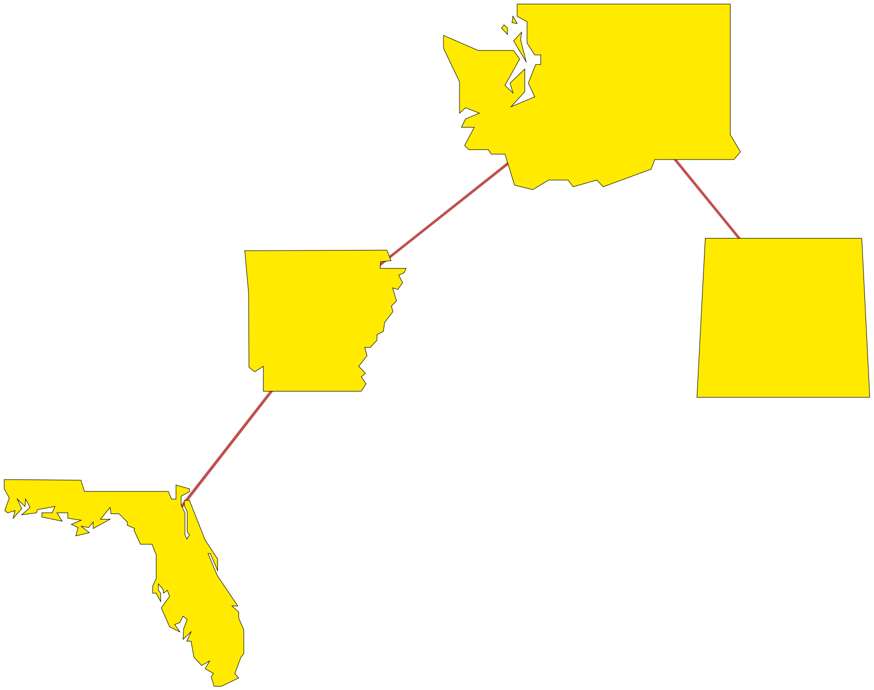
... and you MUST have the third node which always the right child of the root.



FULL

Full Binary Trees

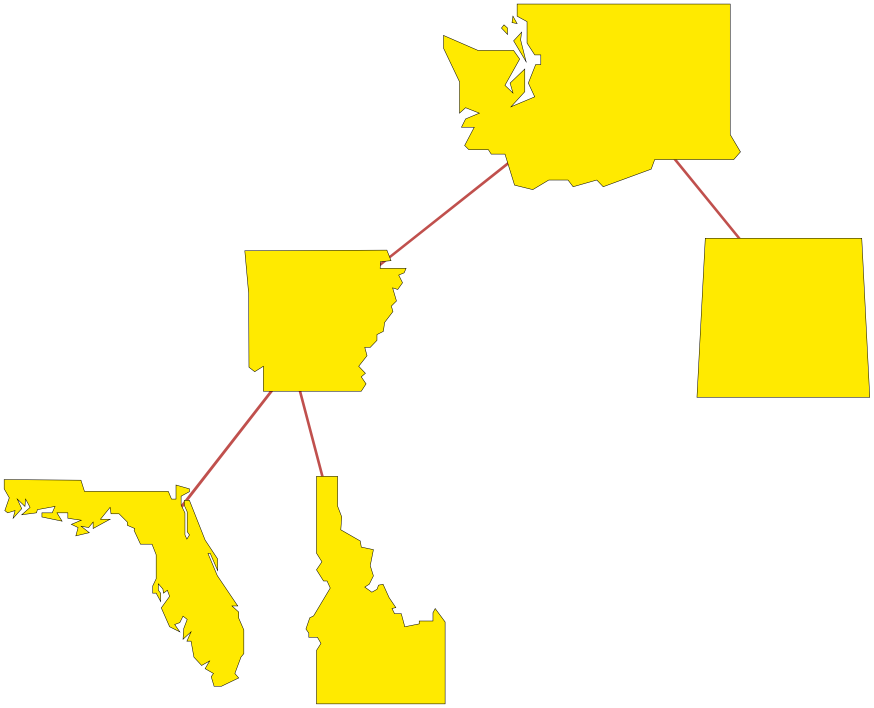
The next nodes must always fill the next level from **left to right**.



not FULL yet

Full Binary Trees

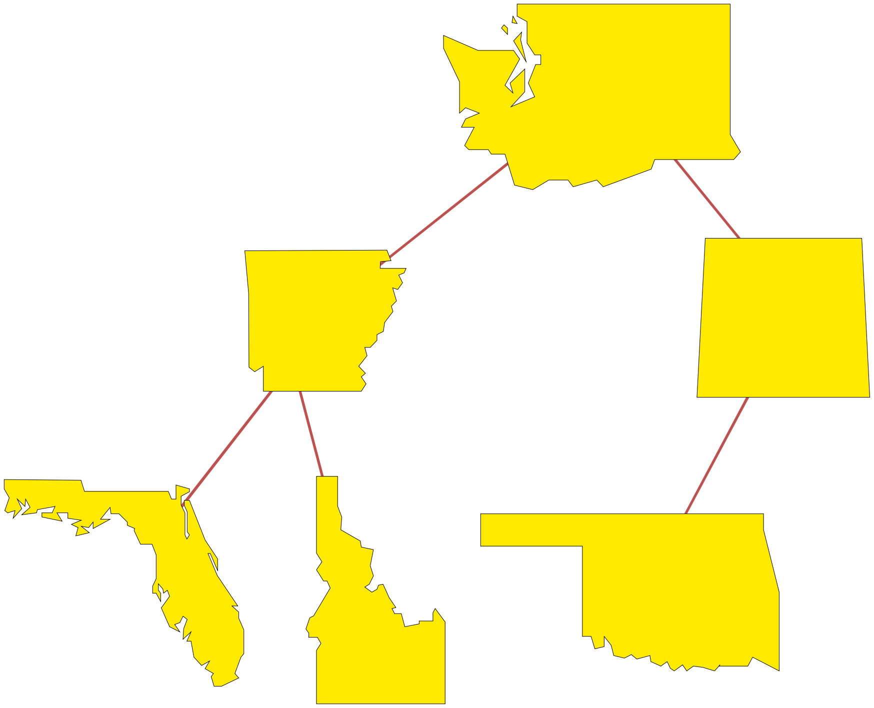
The next nodes must always fill the next level from **left to right**.



not FULL yet

Full Binary Trees

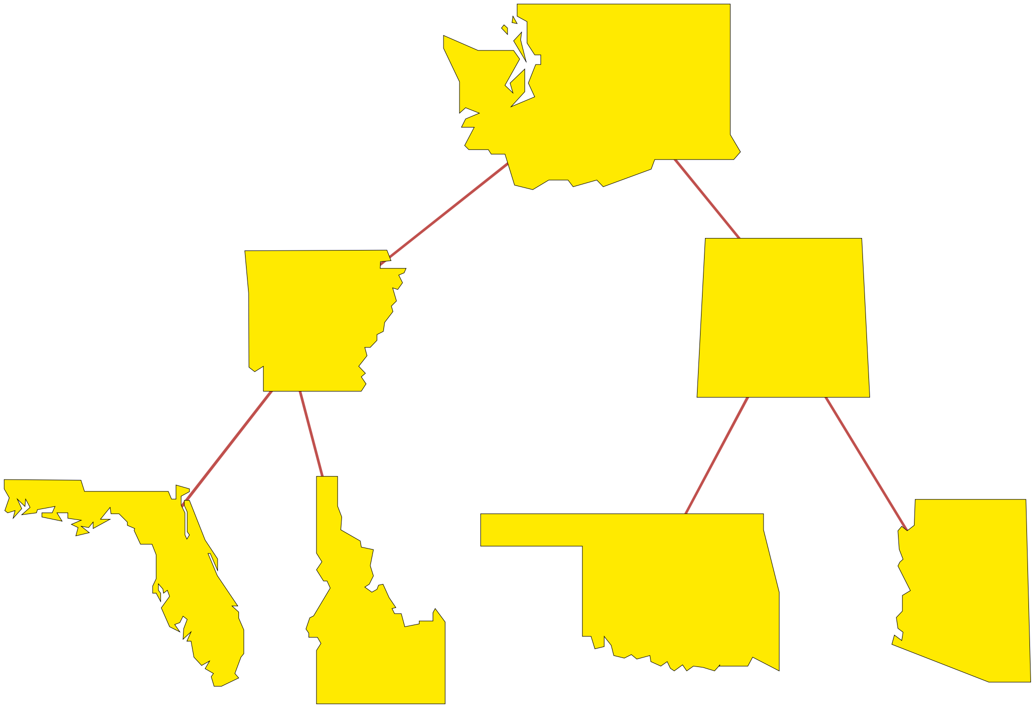
The next nodes must always fill the next level from **left to right**.



not FULL yet

Full Binary Trees

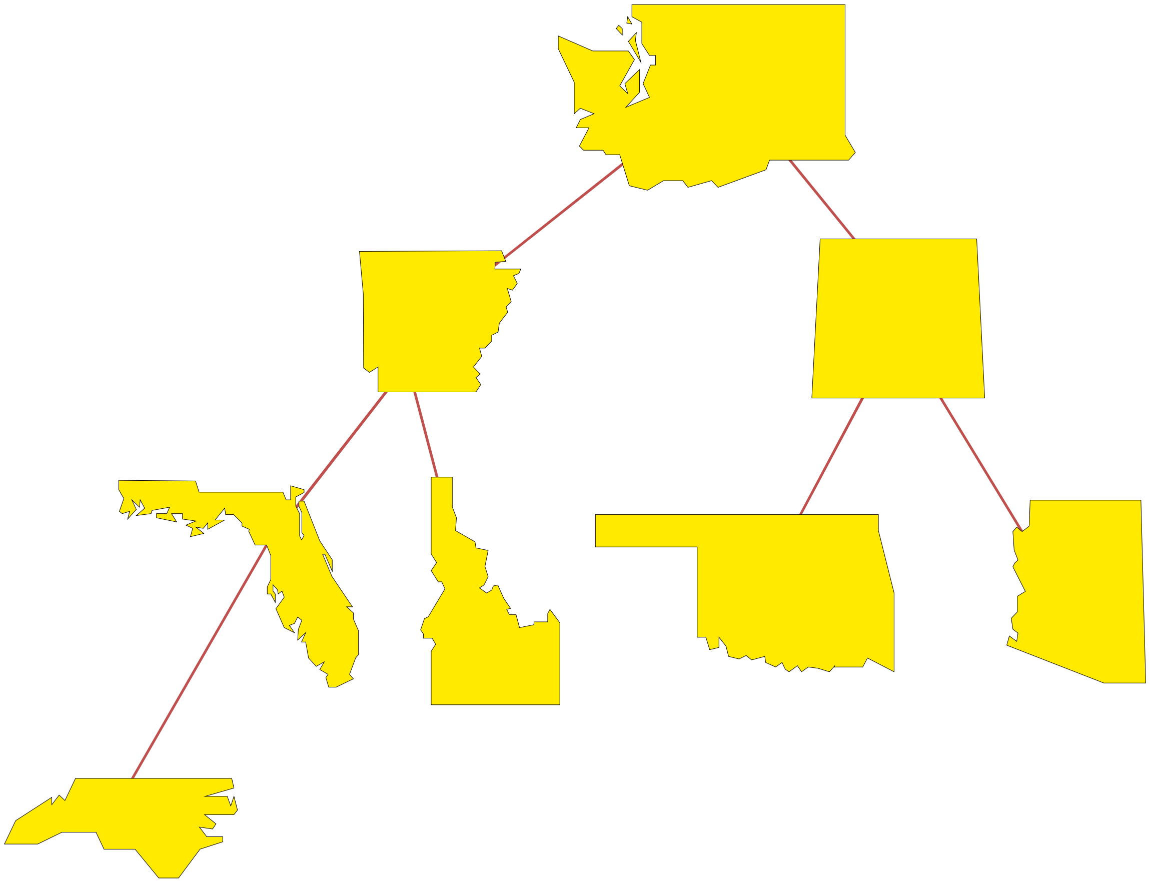
The next nodes must always fill the next level from **left to right**...until every leaf has the same depth (2)



FULL!

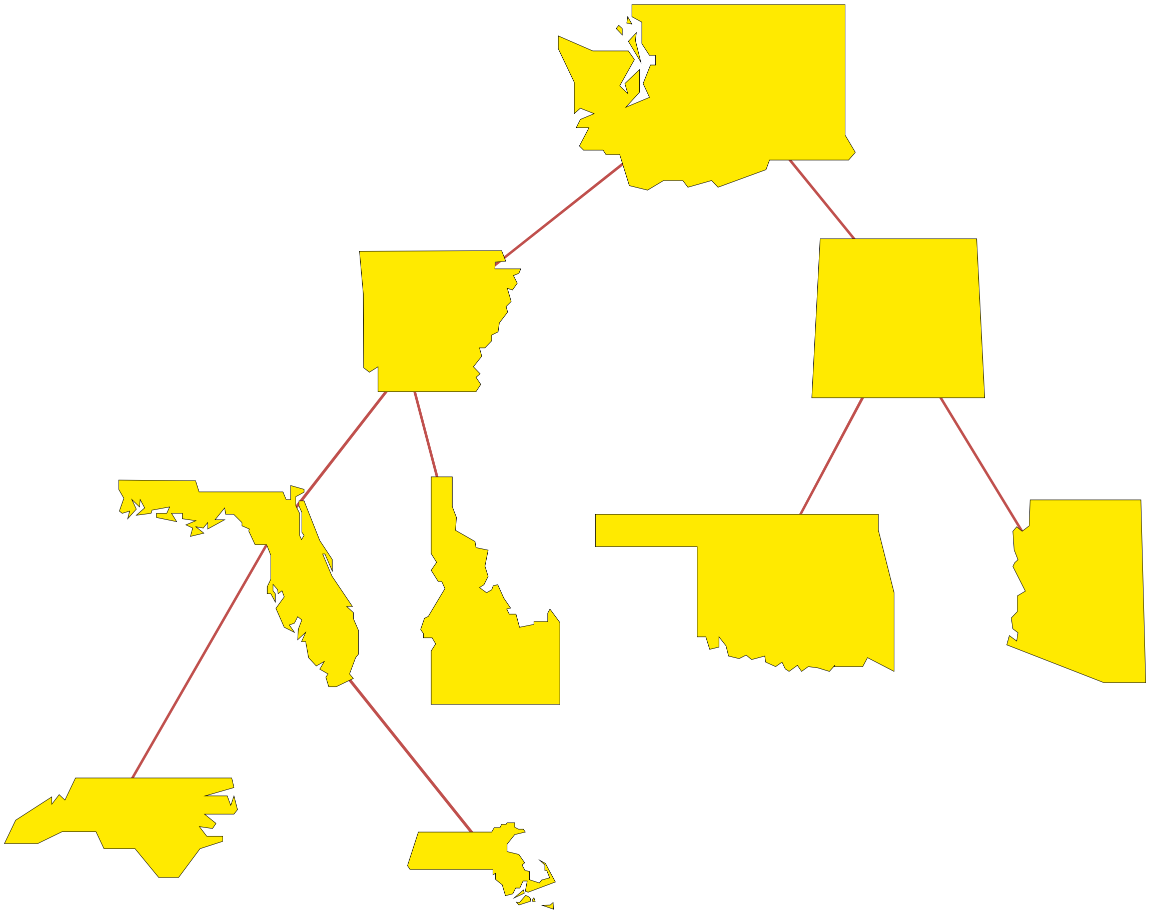
Full Binary Trees

The next nodes must always fill the next level from **left to right**.

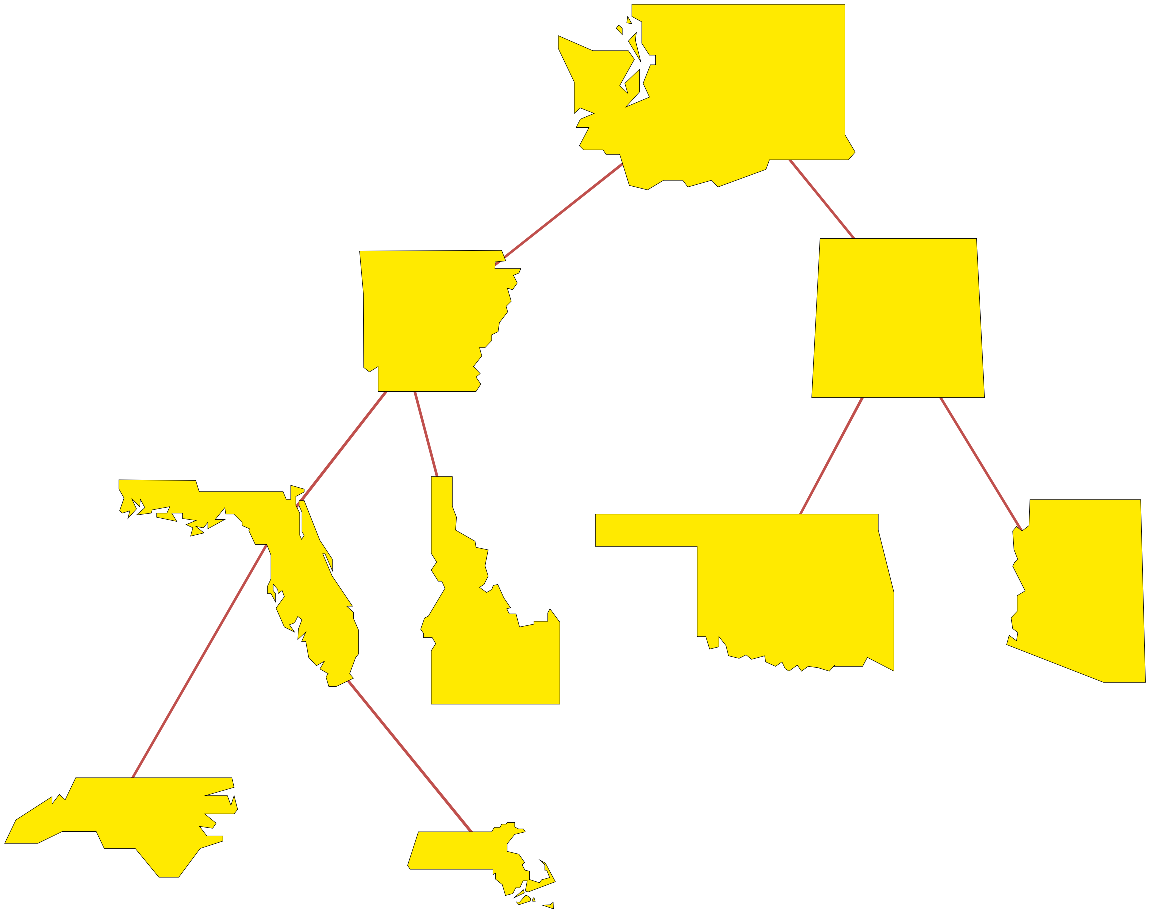


Full Binary Trees

The next nodes must always fill the next level from **left to right**.

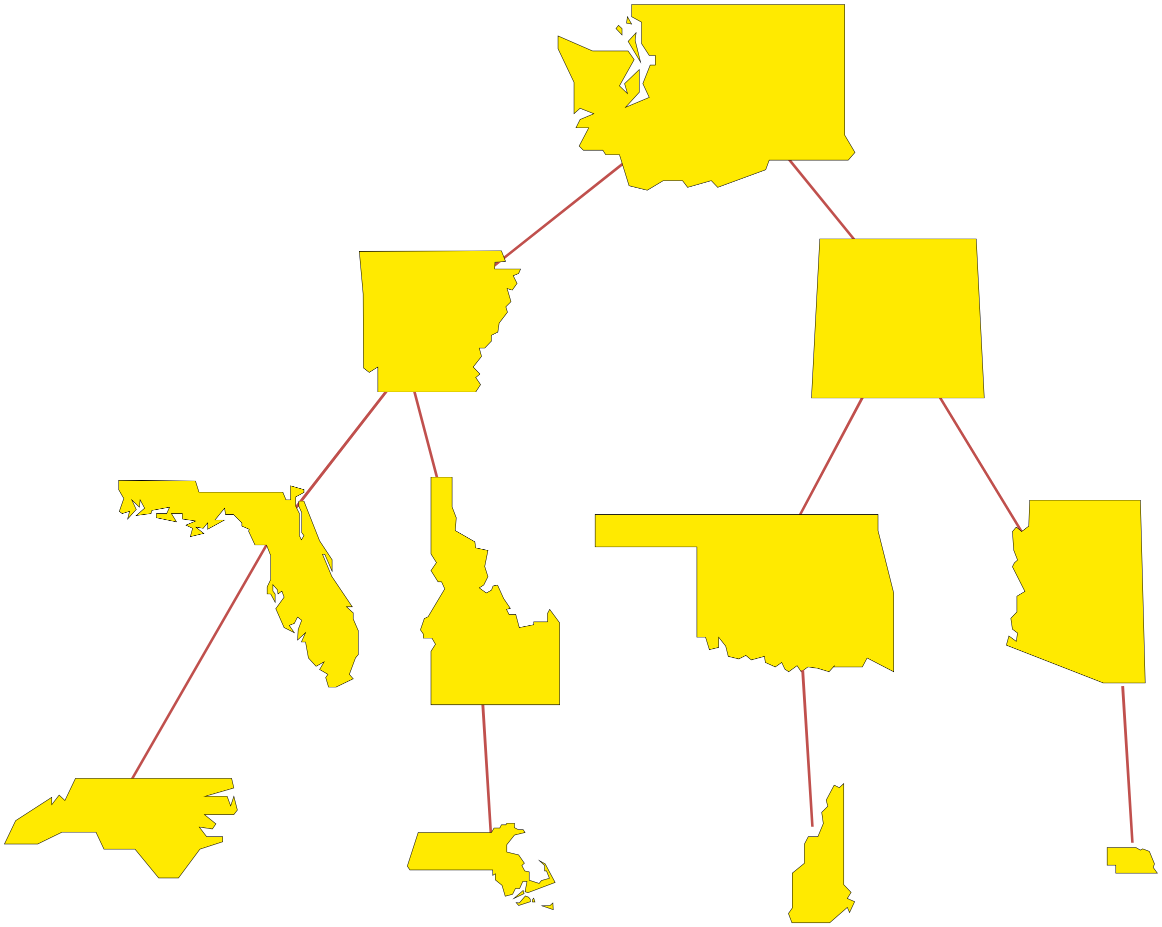


Is This Full?

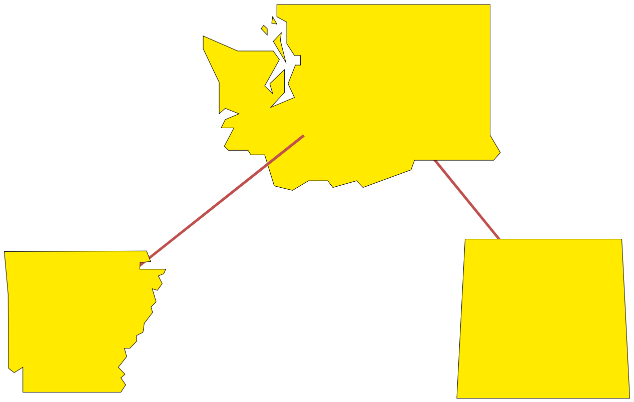


SANTA CLARA UNIVERSITY
THE JESUIT UNIVERSITY IN SILICON VALLEY

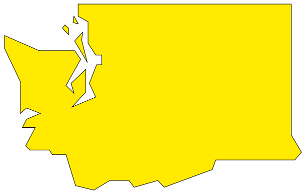
Is This Full?



Is This Full?

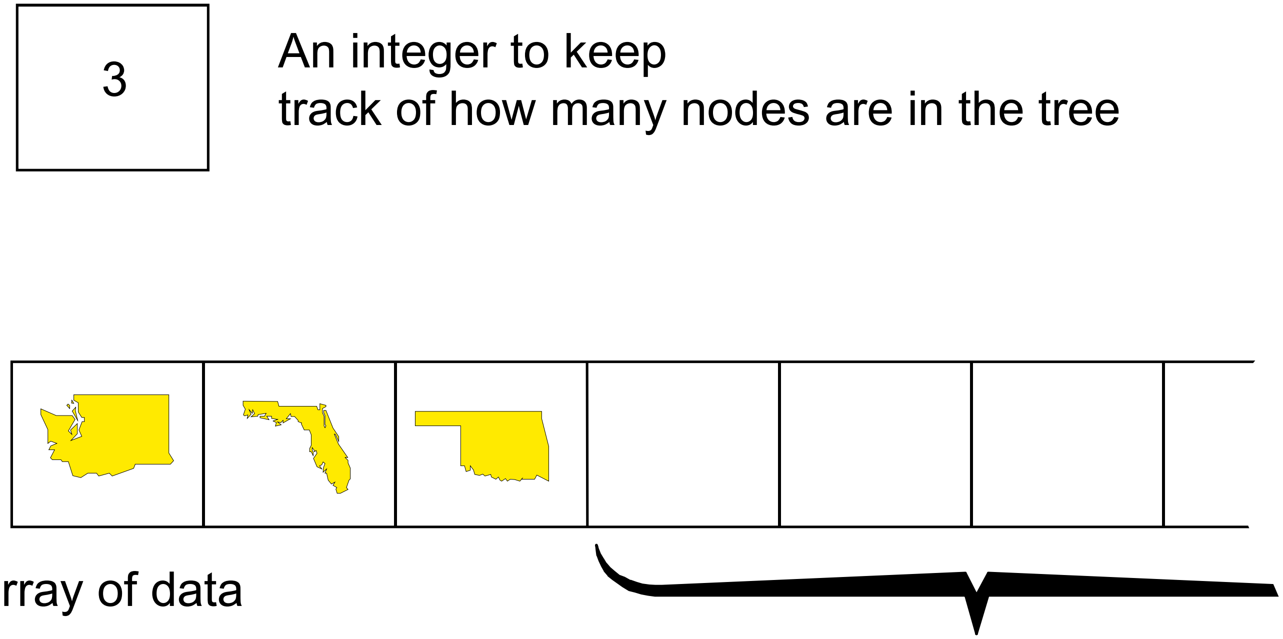


Is This Full?



Array implementation of a Complete Binary Tree

- We will store the date from the nodes in a partially-filled array.

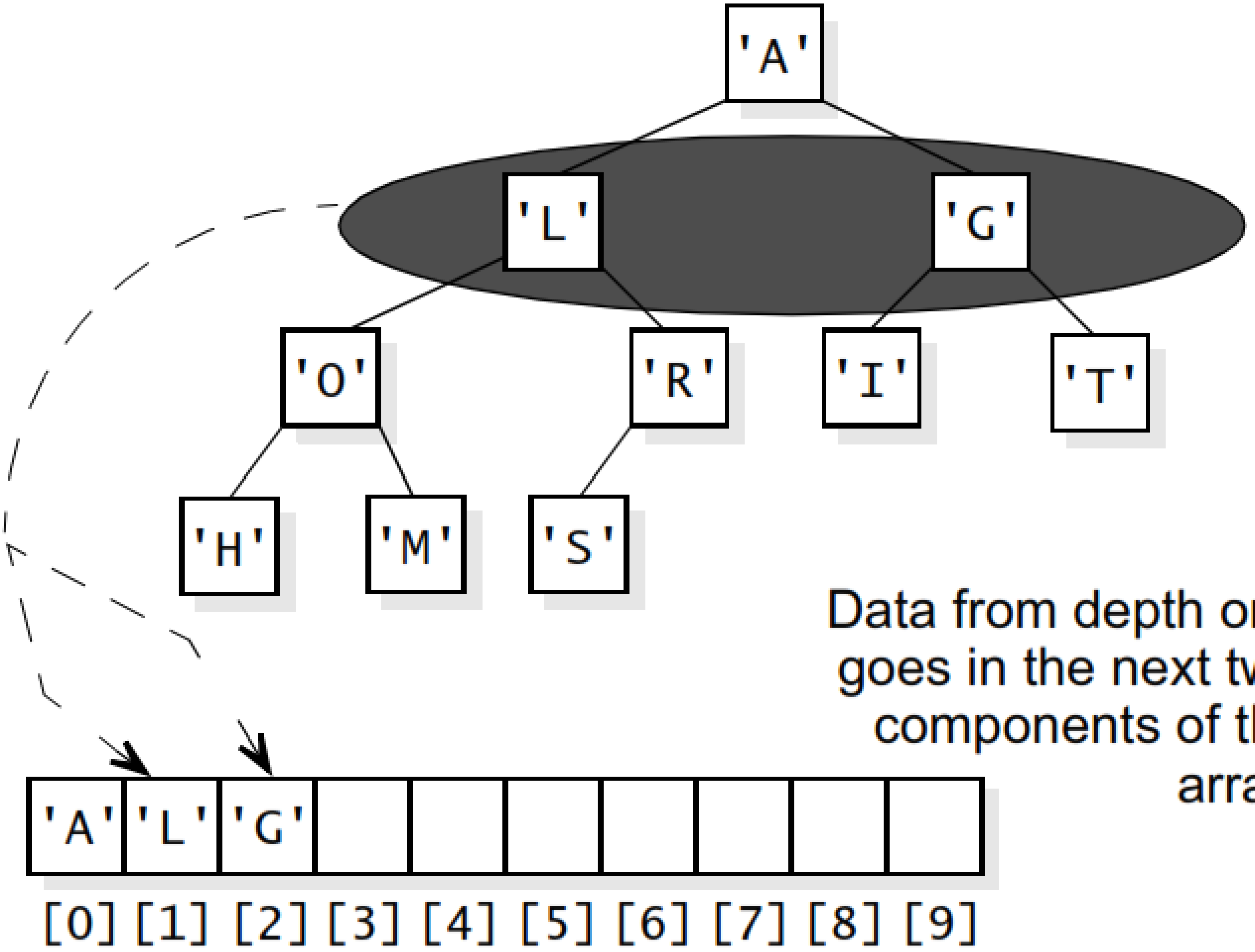


An integer to keep track of how many nodes are in the tree

An array of data

We don't care what's in this part of the array.

Array implementation of a Complete Binary Tree



Data from depth one goes in the next two components of the array.

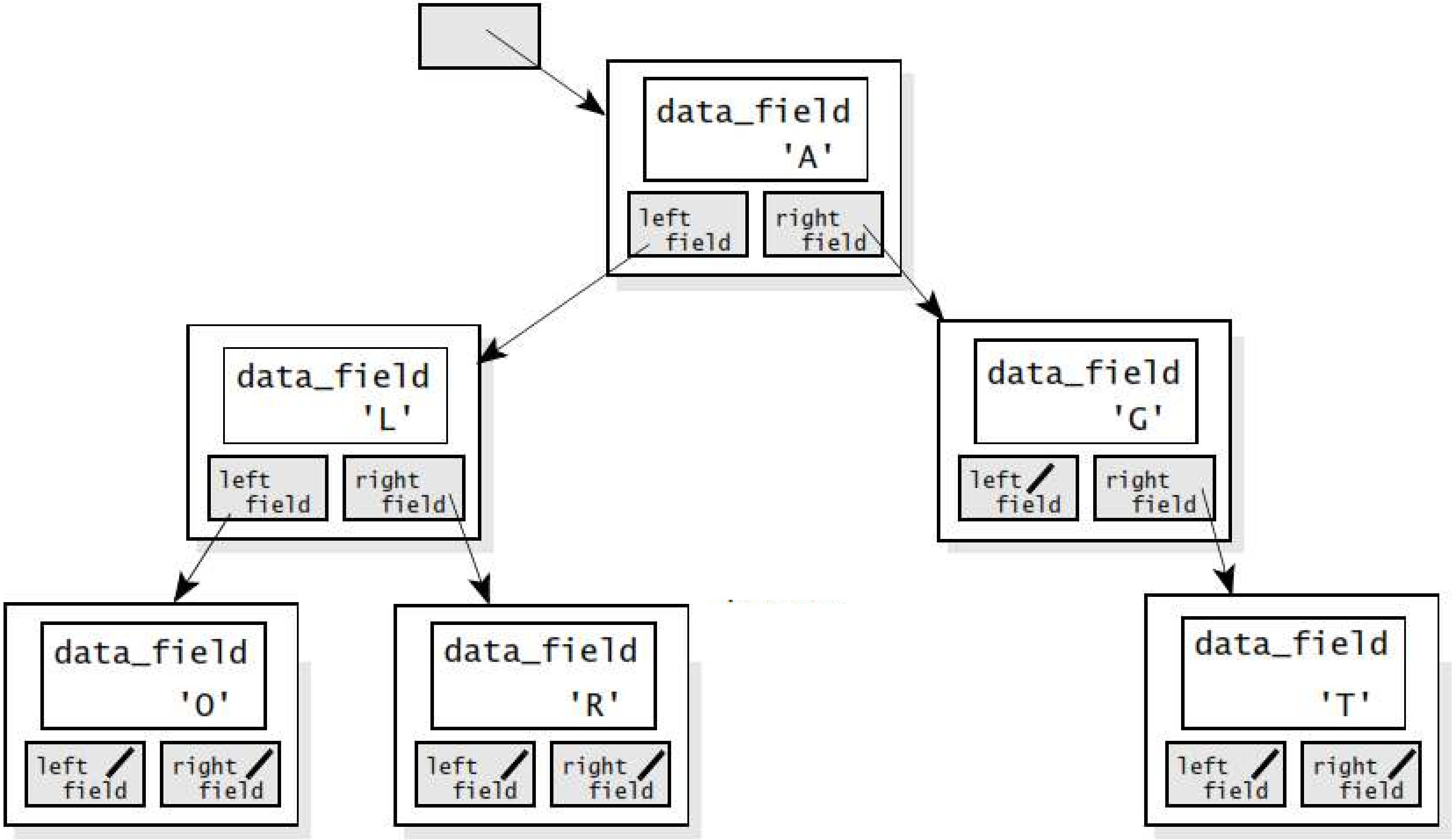
Array implementation of a Complete Binary Tree

- Root is at [0]
- Parent of node in [i] is at [(i-1)/2]
- Children (if exist) of node [i] is at [2i+1] and [2i+2]
- Total node number
    - $2^0+2^1+2^2+\ldots+2^{d-1}+r$, $r <= 2^d$, d is the depth

In a complete binary tree with 10,000 nodes,
suppose that a node has its value stored in location 4999.

Where is the value stored for this node's parent?
Where are the values stored for its left child and right child?

Binary Tree implementation with a Class for Nodes



Binary Tree Nodes

- Each node of a binary tree is stored in an object of a new *binary_tree_node* class
- Each node contains data as well as pointers to its children
- An entire tree is represented as a pointer to the root node

binary_tree_node Class

```
template <class Item>
   class binary_tree_node
   {
   public:
      …

   private:
      Item data_field;
      binary_tree_node *left_field;
      binary_tree_node *right_field;
   };
```

```
// TYPEDEF
typedef Item value_type;
// CONSTRUCTOR
binary_tree_node(
    const Item& init_data = Item( ),
    binary_tree_node* init_left = NULL,
    binary_tree_node* init_right = NULL
)
{
    data_field = init_data;
    left_field = init_left;
    right_field = init_right;
}
```

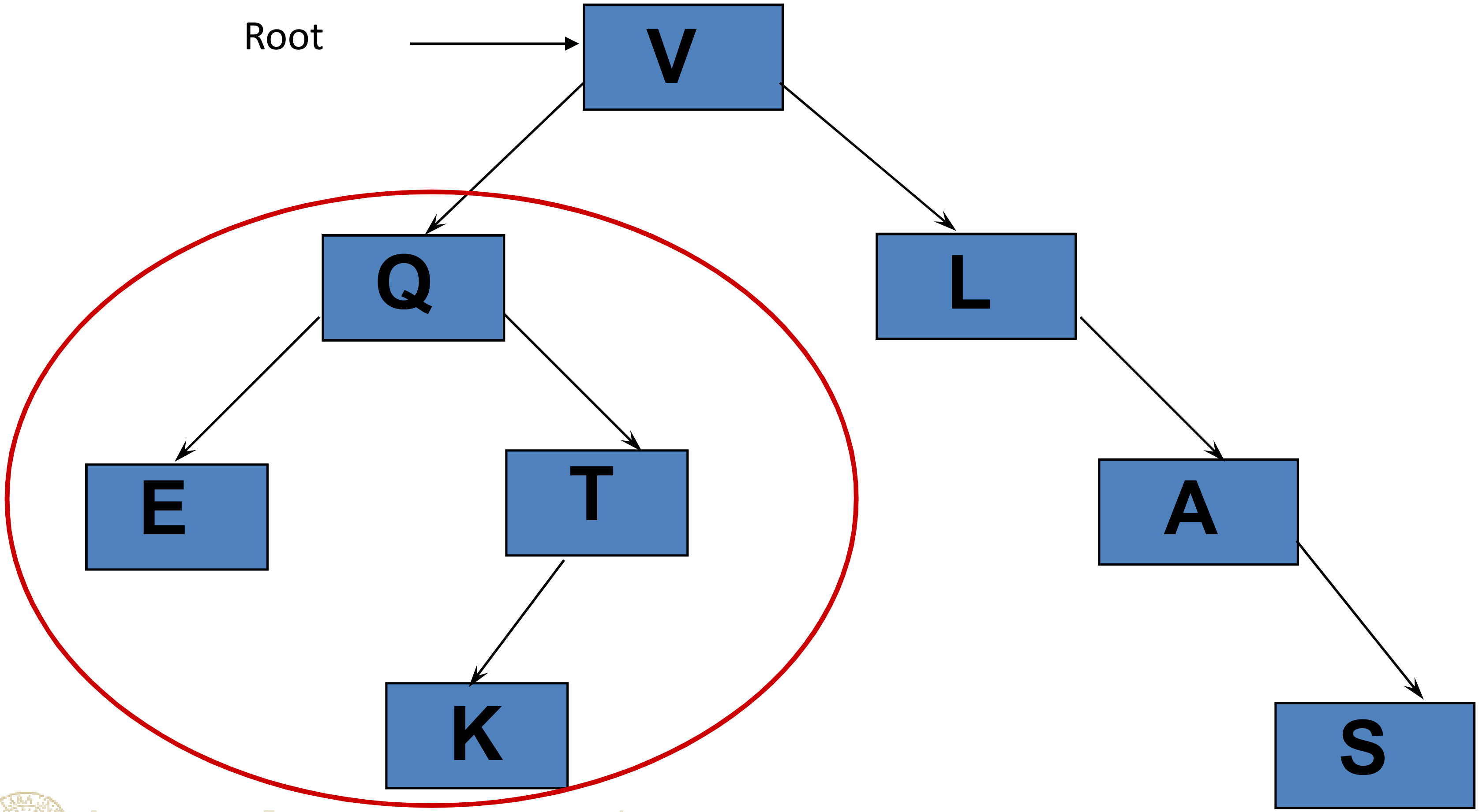
```
// MODIFICATION MEMBER FUNCTIONS
Item& data( ) { return data_field; }
binary_tree_node* left( ) { return left_field; }
binary_tree_node* right( ) { return right_field; }
void set_data(const Item& new_data) { data_field = new_data; }
void set_left(binary_tree_node* new_left) { left_field = new_left; }
void set_right(binary_tree_node* new_right)
                                      { right_field = new_right; }

// CONST MEMBER FUNCTIONS
const Item& data( ) const { return data_field; }
const binary_tree_node* left( ) const { return left_field; }
const binary_tree_node* right( ) const { return right_field; }
bool is_leaf( ) const
    { return (left_field == NULL) && (right_field == NULL); }
```

Creating and Manipulating Trees

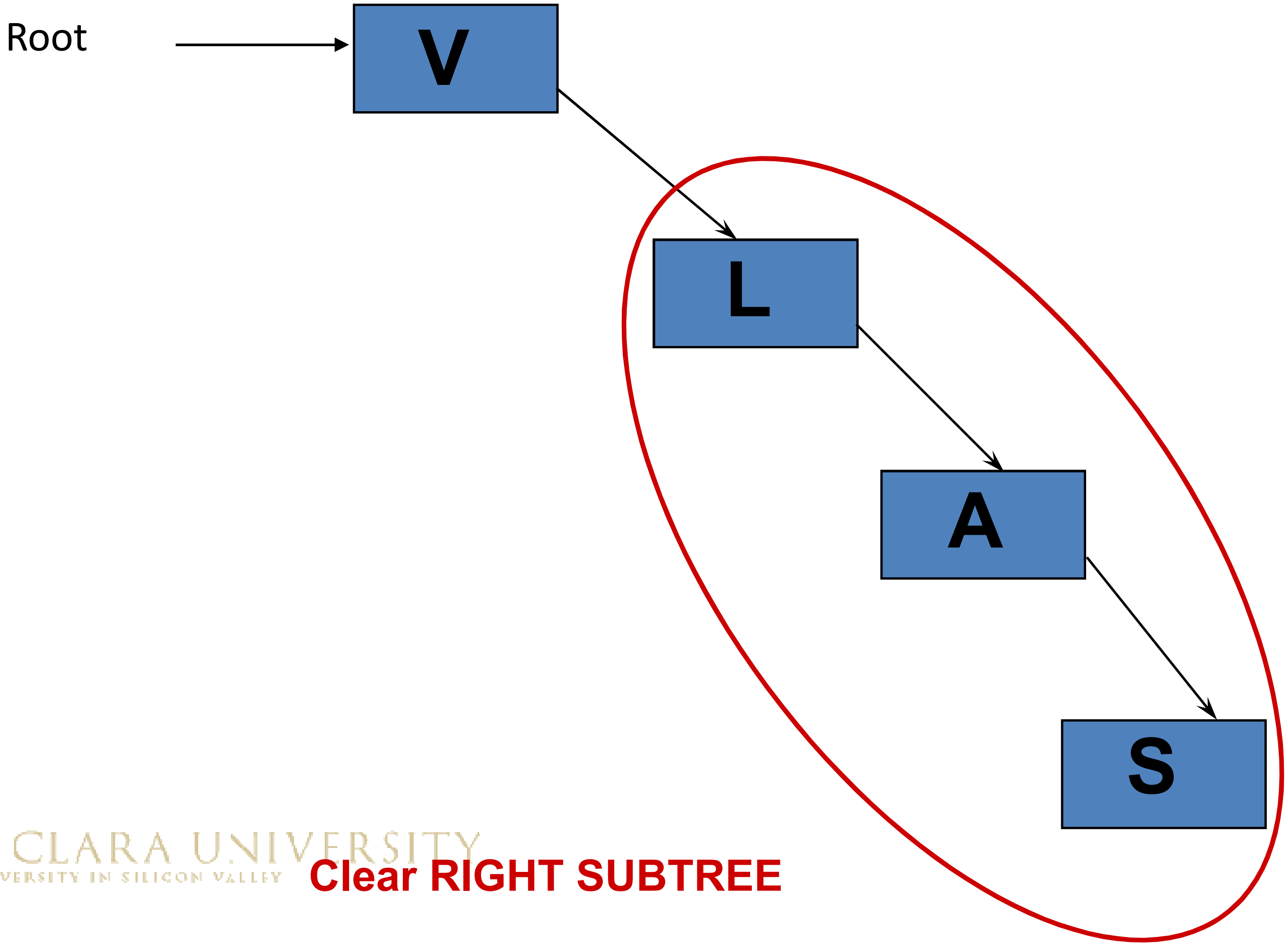
- Consider only two functions
  - Clearing a tree
    - ✓Return nodes of a tree to the heap
  - Copying a tree
- The Implementation is easier than it seems
  - if we use recursive thinking

Clearing a Tree



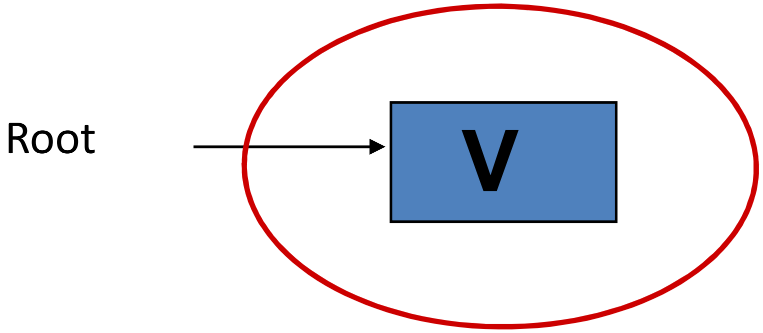
Clear LEFT SUBTREE

Clearing a Tree



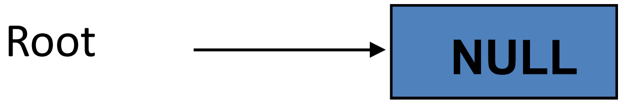
Clear RIGHT SUBTREE

Clearing a Tree



Return root node to the heap

Clearing a Tree


Root

NULL

Set the root pointer to NULL

Clear a Tree

- key: recursive thinking

```
template <class Item>
void tree_clear(binary_tree_node<Item>*& root_ptr)
// Library facilities used: cstdlib
{
    if (root_ptr != NULL)
    {
        tree_clear( root_ptr->left( ) ); // clear left subtree
        tree_clear( root_ptr->right( ) ); // clear right subtree
        delete root_ptr;   // return root node to the heap
        root_ptr = NULL; // set root pointer to the null
    }
}
```

Copy a Tree

```
template <class Item>
binary_tree_node<Item>* tree_copy
                   (const binary_tree_node<Item>* root_ptr)
// Library facilities used: cstdlib
{
    binary_tree_node<Item> *l_ptr;
    binary_tree_node<Item> *r_ptr;

    if (root_ptr == NULL)
        return NULL;
    else
    {
        // copy the left sub_tree
        l_ptr = tree_copy( root_ptr->left( ) );
        // copy the right sub_tree
        r_ptr = tree_copy( root_ptr->right( ) );
        return new binary_tree_node<Item>
                          (root_ptr->data( ), l_ptr, r_ptr);
    }  // copy the root node and set the root pointer
}
```

Binary Tree Traversals

- pre-order traversal
  - root (left sub_tree) (right sub_tree)
- in-order traversal
  - (left sub_tree) root (right sub_tree)
- post-order traversal
  - (left sub_tree) (right sub_tree) root
- backward in-order traversal
  - (right sub_tree) root (left sub_tree)