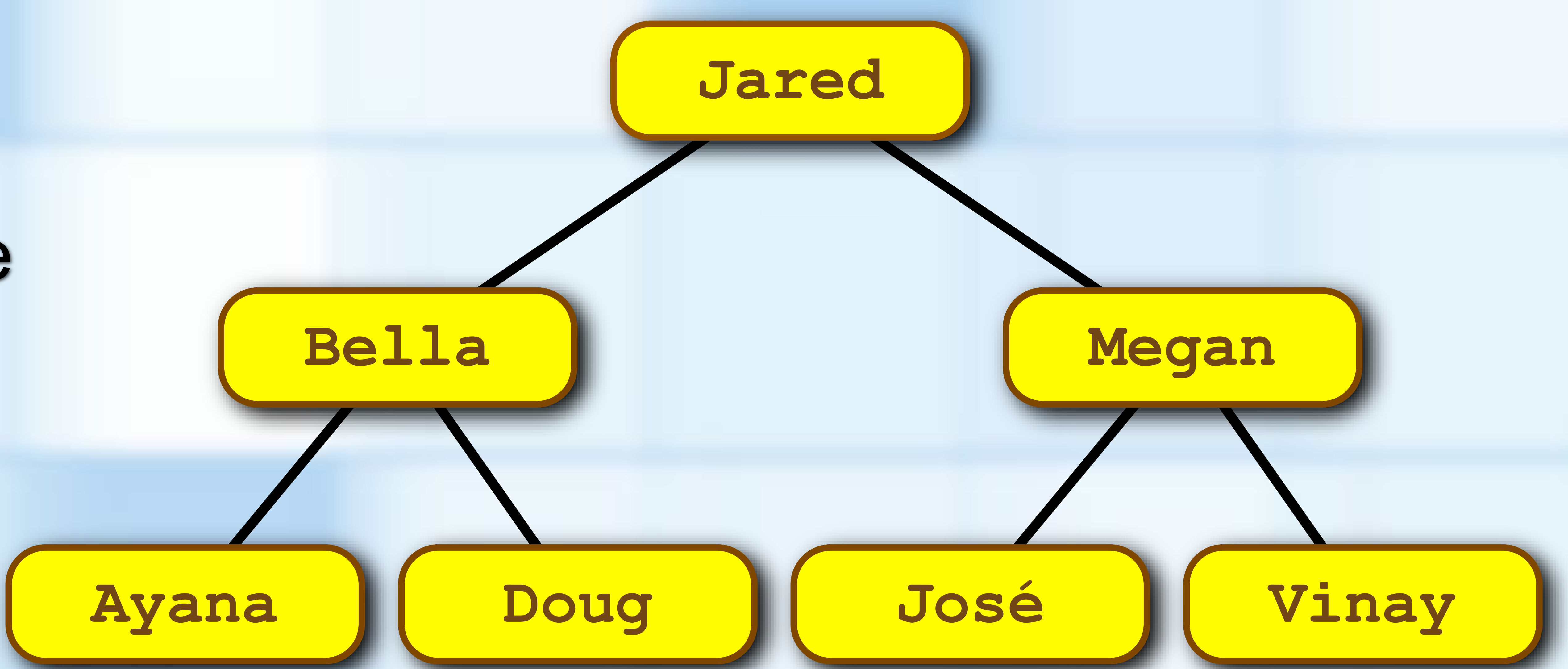
Overview of the Binary Search Tree

Binary Search Trees

- **Binary Search Tree**
  - Binary tree that has the following properties for each node **`n`**
    - **`n`**'s value is > all values in **`n`**'s left subtree T_L
    - **`n`**'s value is < all values in **`n`**'s right subtree T_R
    - Both T_L and T_R are binary search trees
- **A binary tree whose nodes contain objects and ...**
  - data in a node is greater than the data in the node's left child
  - data in a node is less than the data in the node's right child

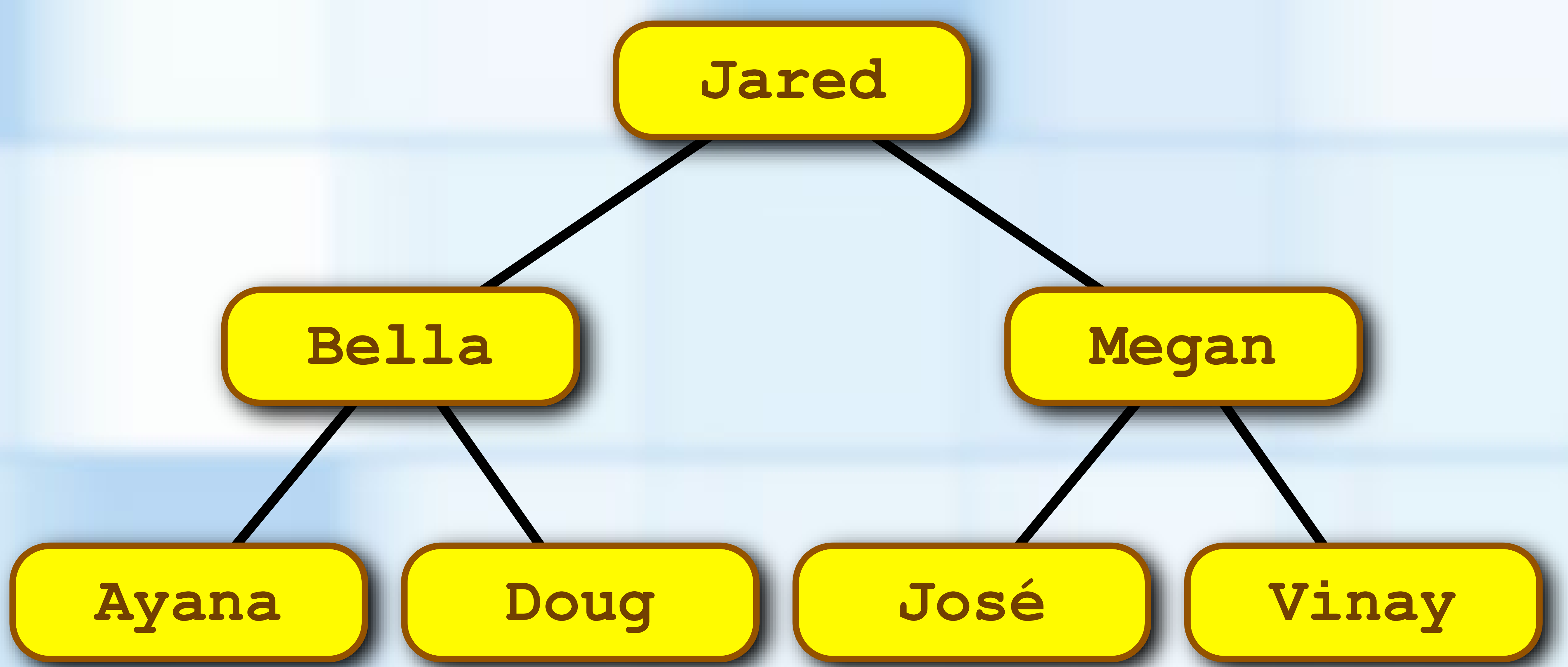


An In-Order Traversal is "In Order" for a Binary Search Tree

SEARCHING A BINARYSEARCHTREE

- **Searching for an entry**
  - Recursive implementation
  - Similar to Binary Search algorithm

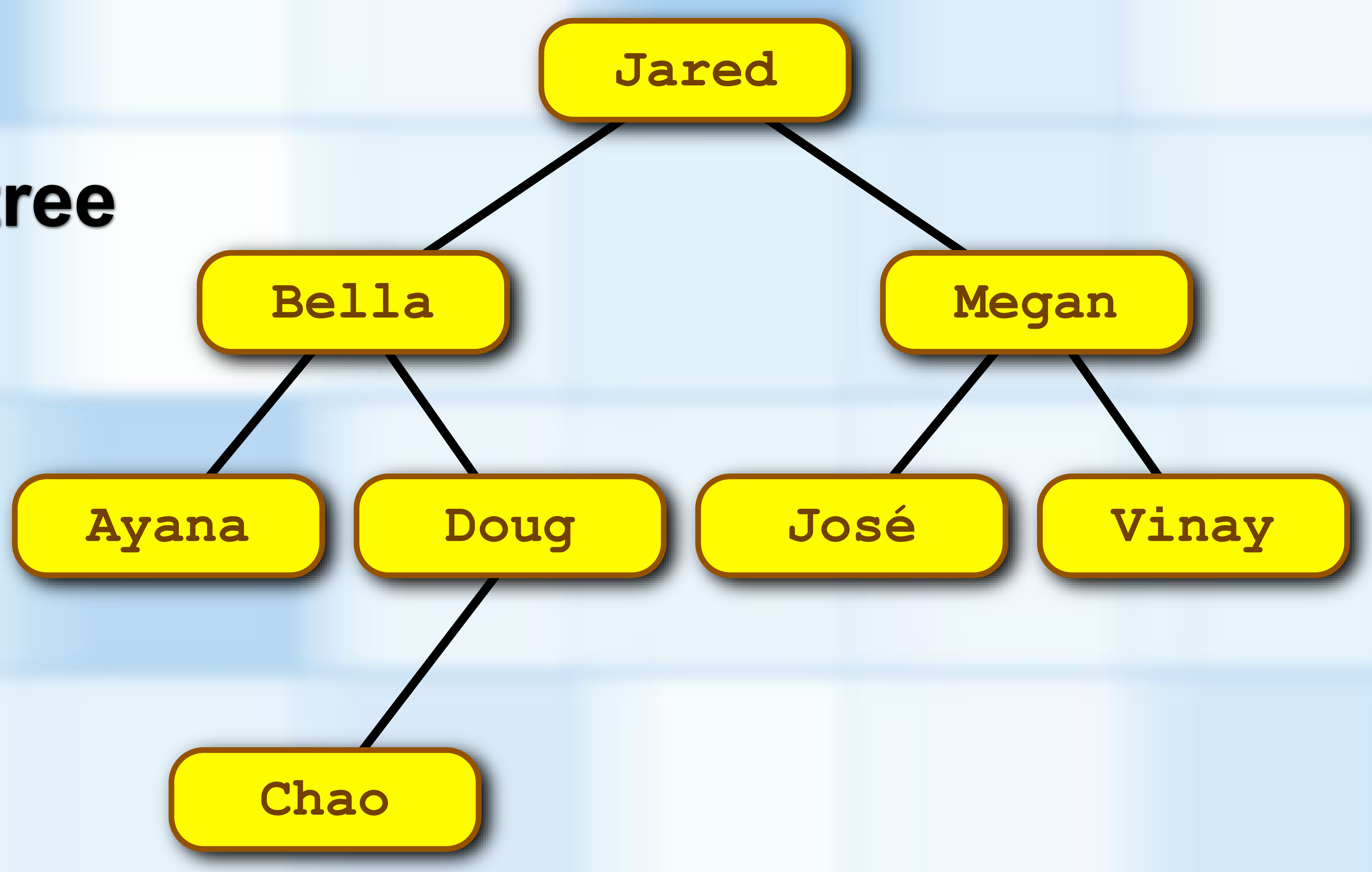
`findNode( Doug )`



ADDING TO A BINARY SEARCH TREE

- Must maintain binary search tree structure
- Every addition to a binary search tree adds a new leaf to the tree.

`add( Chao )`



BinarySearchTree Restrictions

```
#ifndef BINARY_TREE_INTERFACE_
#define BINARY_TREE_INTERFACE_

#include "NotFoundException.h"

template<class ItemType>
class BinaryTreeInterface
{
public:
  virtual bool isEmpty() const = 0;
  virtual int getHeight() const = 0;
  virtual int getNumberOfNodes() const = 0;
  virtual ItemType getRootData() const = 0;
  virtual void setRootData(const ItemType& someItem) = 0;
  virtual bool add(const ItemType& someItem) = 0;
  virtual bool remove(const ItemType& target) = 0;
  virtual void clear() = 0;
  virtual ItemType getEntry(const ItemType& target) const = 0;
  virtual bool contains(const ItemType& target) const = 0;
  virtual void preorderTraverse(
              std::function<void (ItemType&)> visit) const = 0;
  virtual void inorderTraverse(
              std::function<void (ItemType&)> visit) const = 0;
  virtual void postorderTraverse(
              std::function<void (ItemType&)> visit) const = 0;
  virtual ~BinaryTreeInterface() {  }
}; // end BinaryTreeInterface
#endif
```

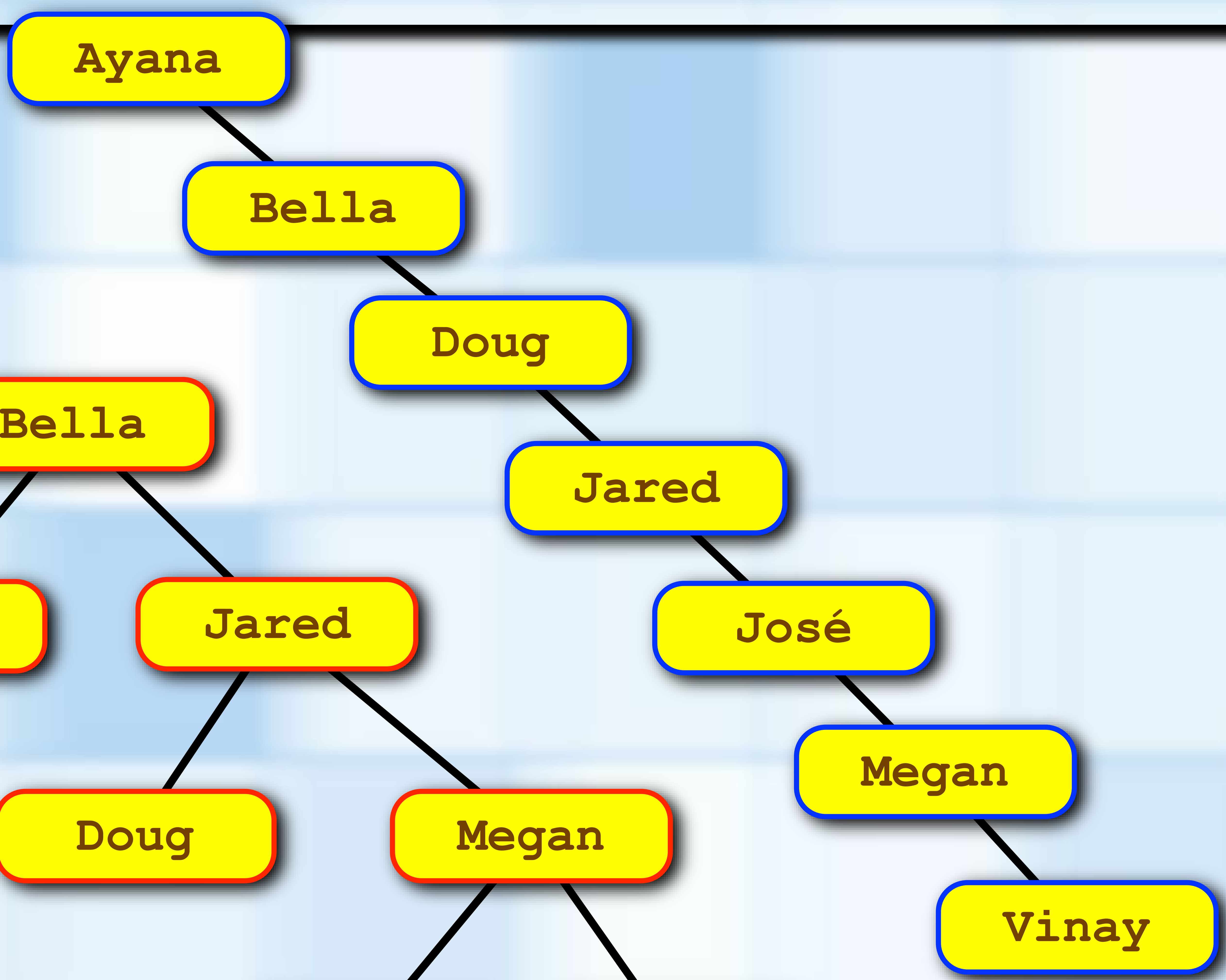
- "Disable" any methods that could change the value in a node

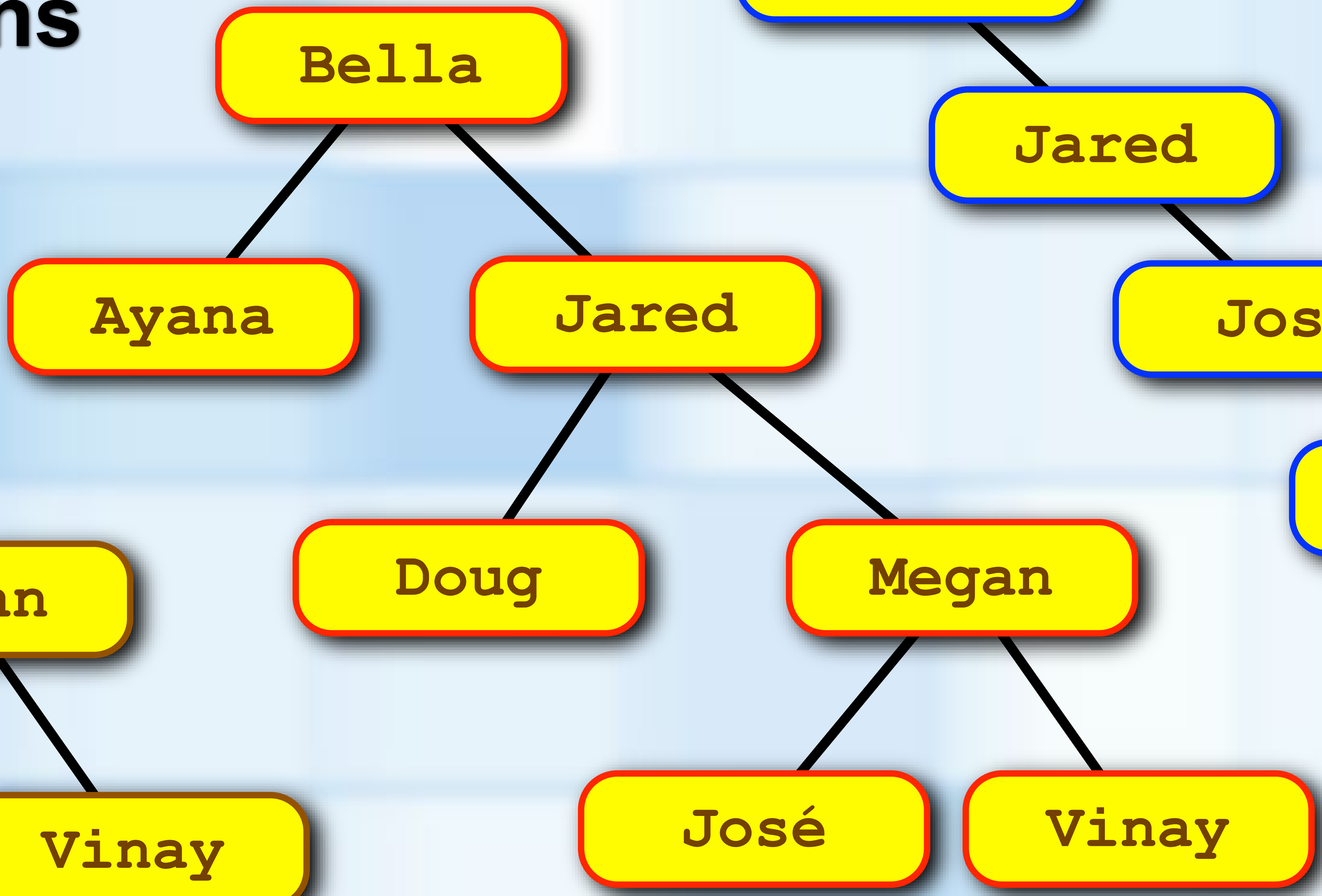
```
template<class ItemType>
void BinarySearchTree<ItemType>::
            setRootData(const ItemType& newItem)
{
  throw PrecondViolatedExcept("Cannot change root in a BST!");
}  // end setRootData
```

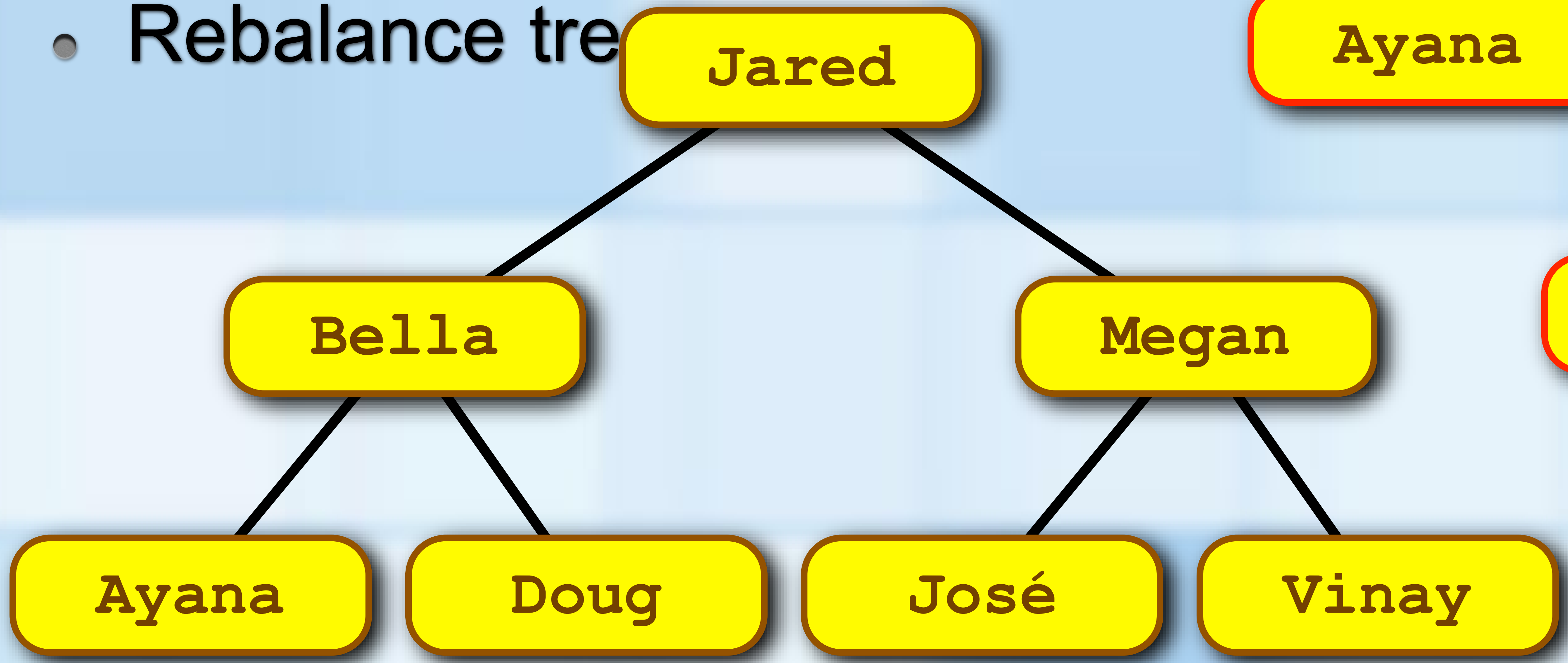
AVL Trees

AVL Trees

- **Adel’sen-Vel’ski and Landis Trees**
- **Maintains a balanced binary search tree through rotations**
- **After each add**
  - Rebalance tre

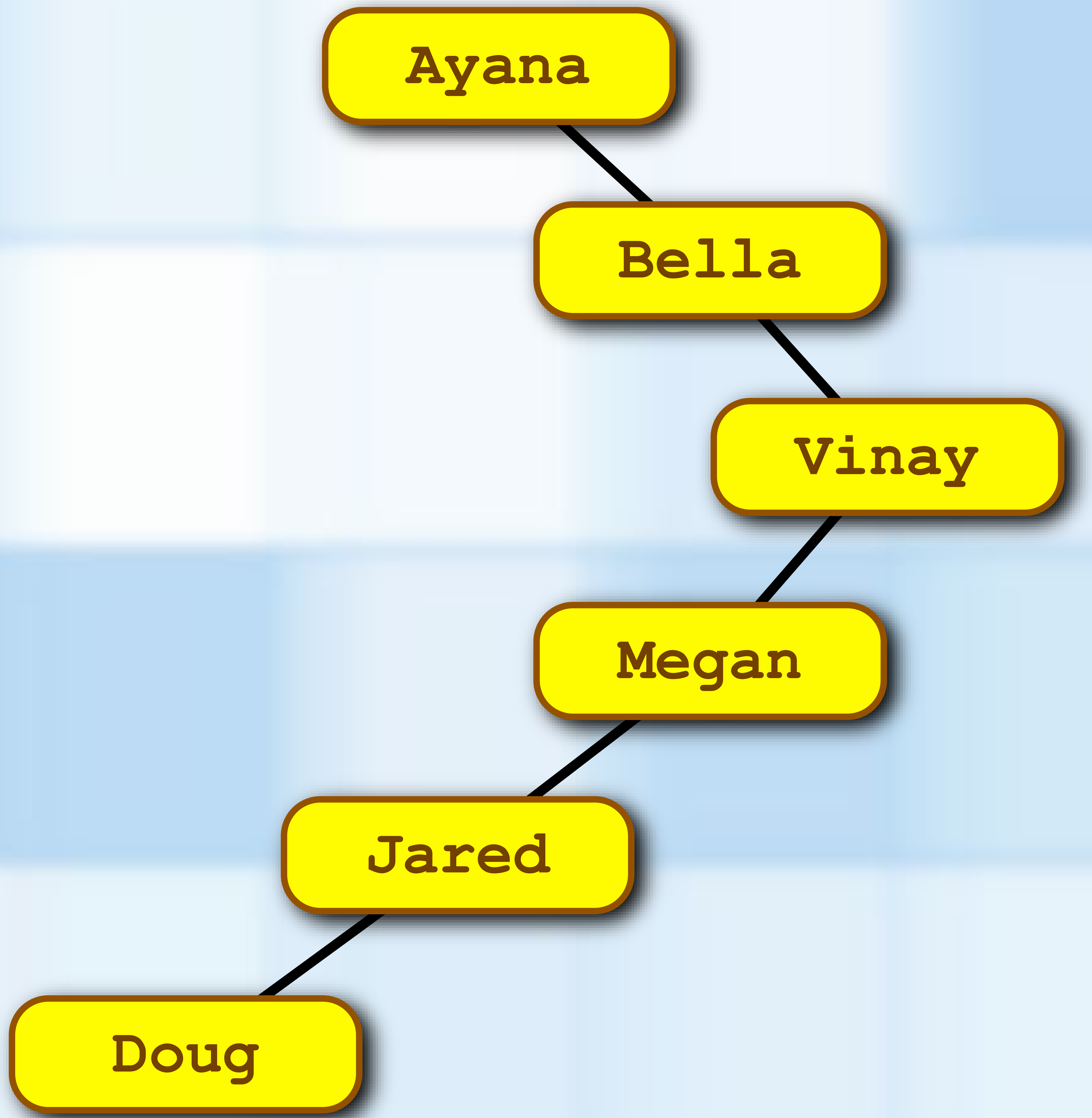


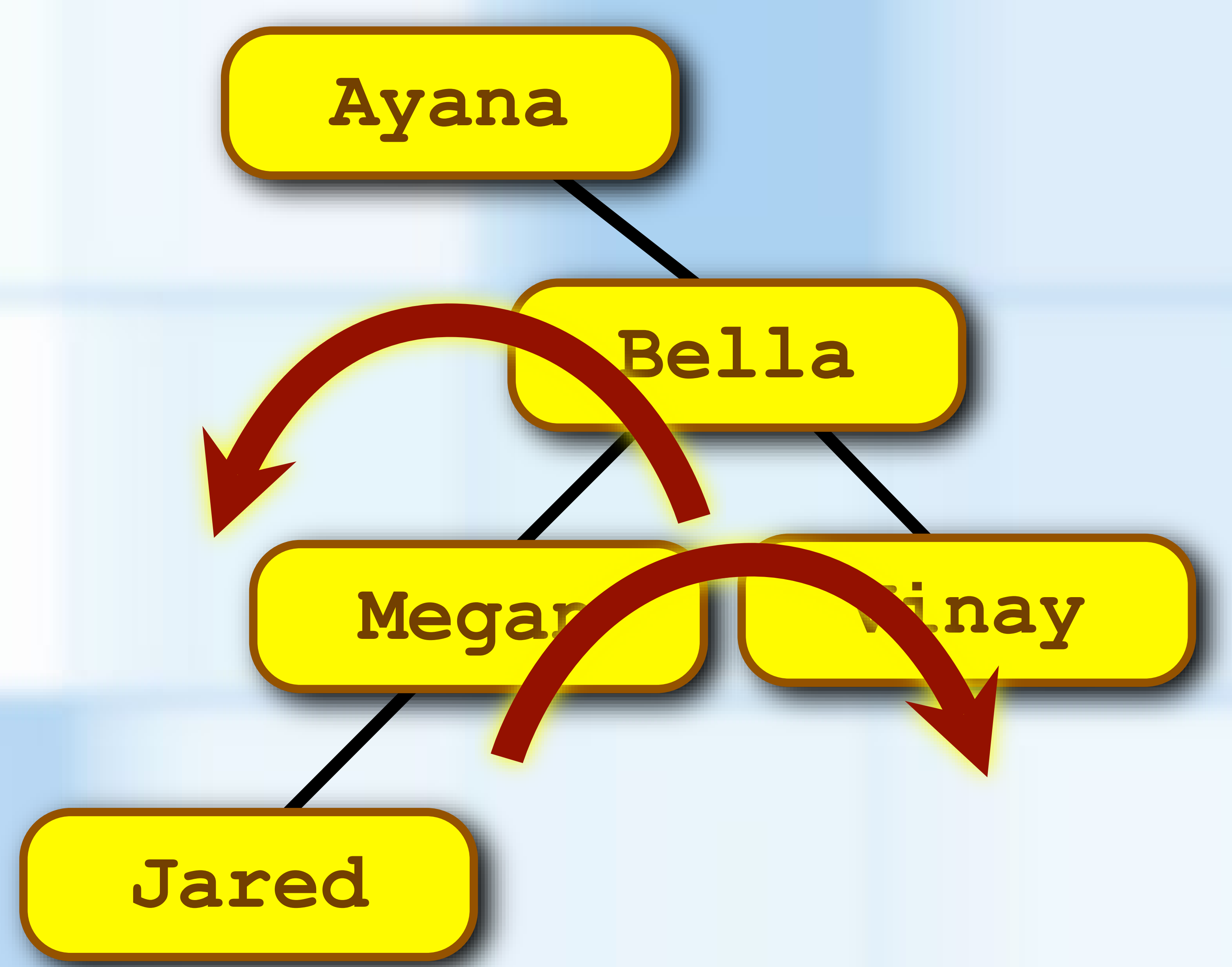




AVL Trees

Ayana
Bella
Vinay
Megan
Jared
Doug

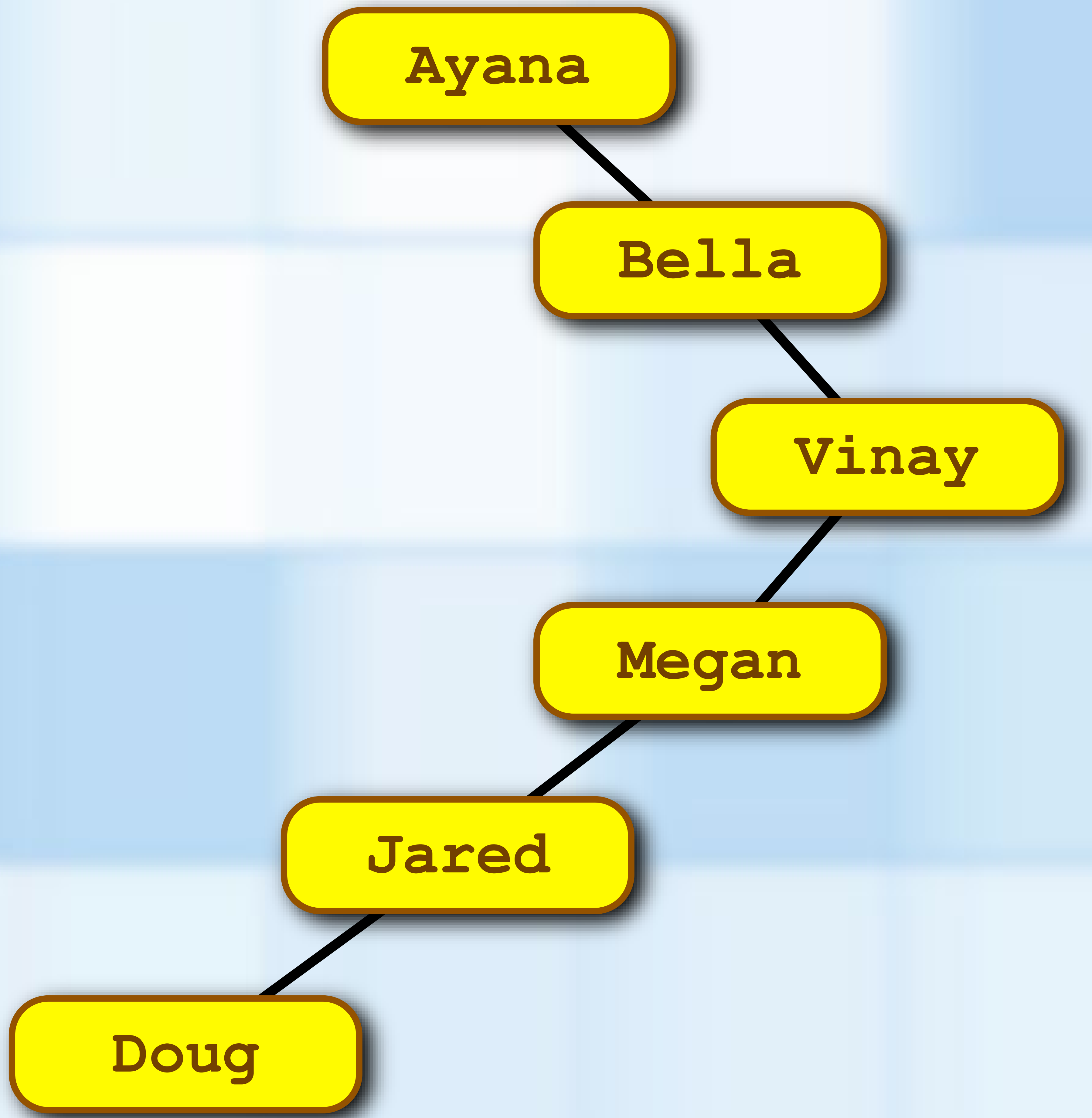


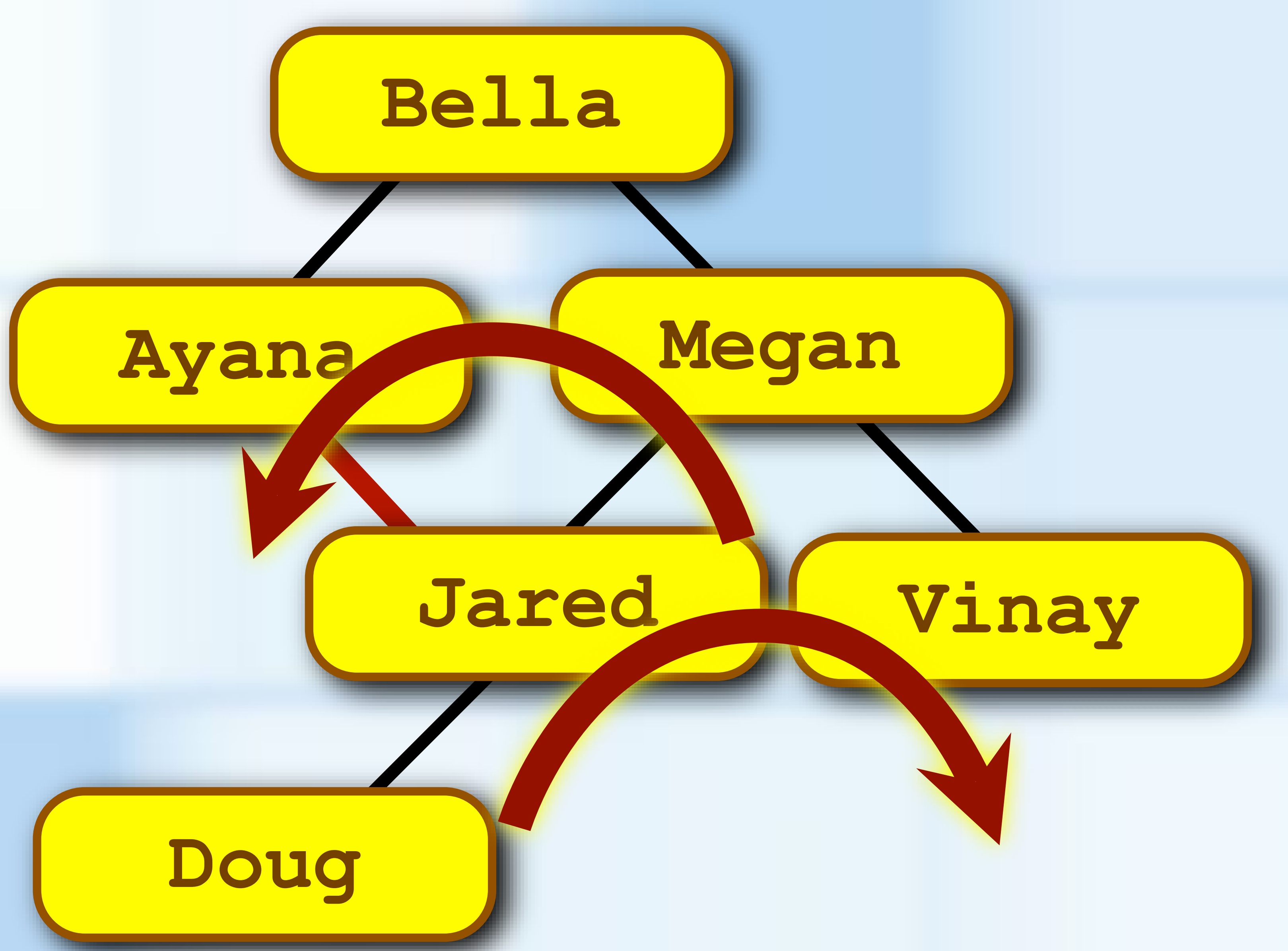


Right Left Single Rotation

AVL Trees

Ayana
Bella
Vinay
Megan
Jared
Doug





Double Rotation