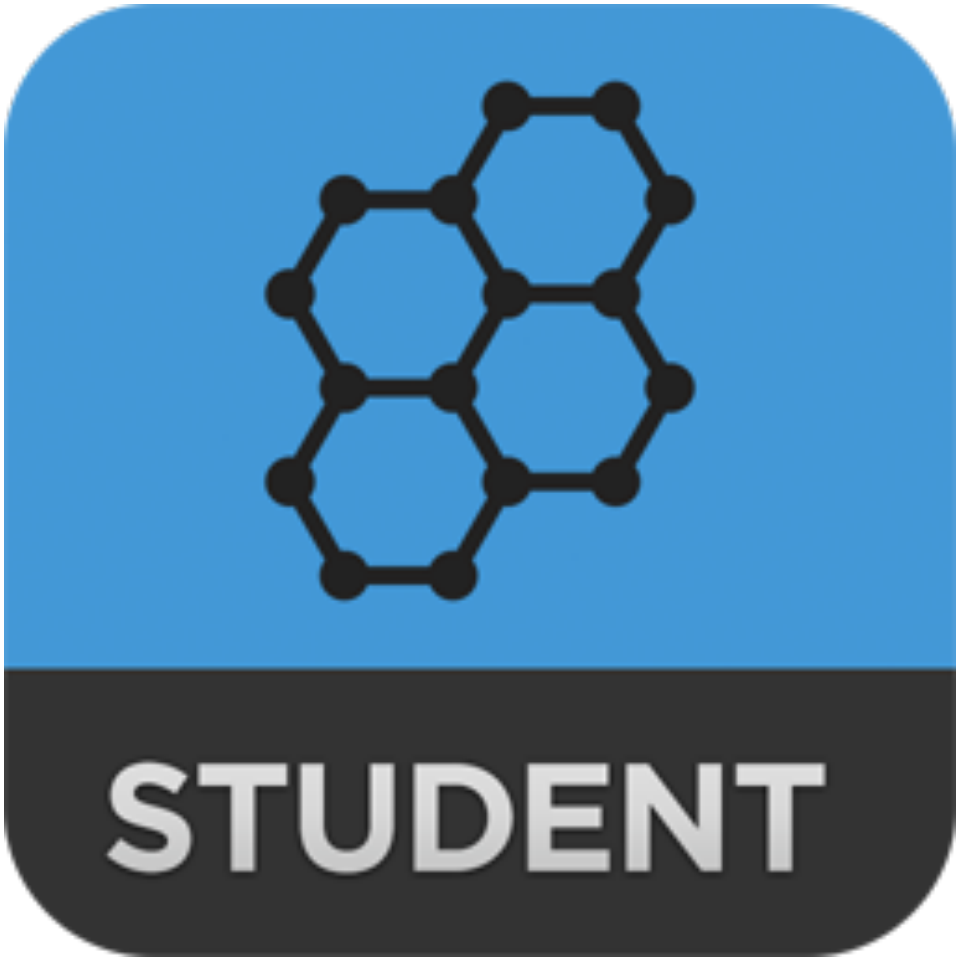

Trees 2
(Trees)
Chris Piech
CS 106B
Lecture 19
Feb 22, 2016

Socrative

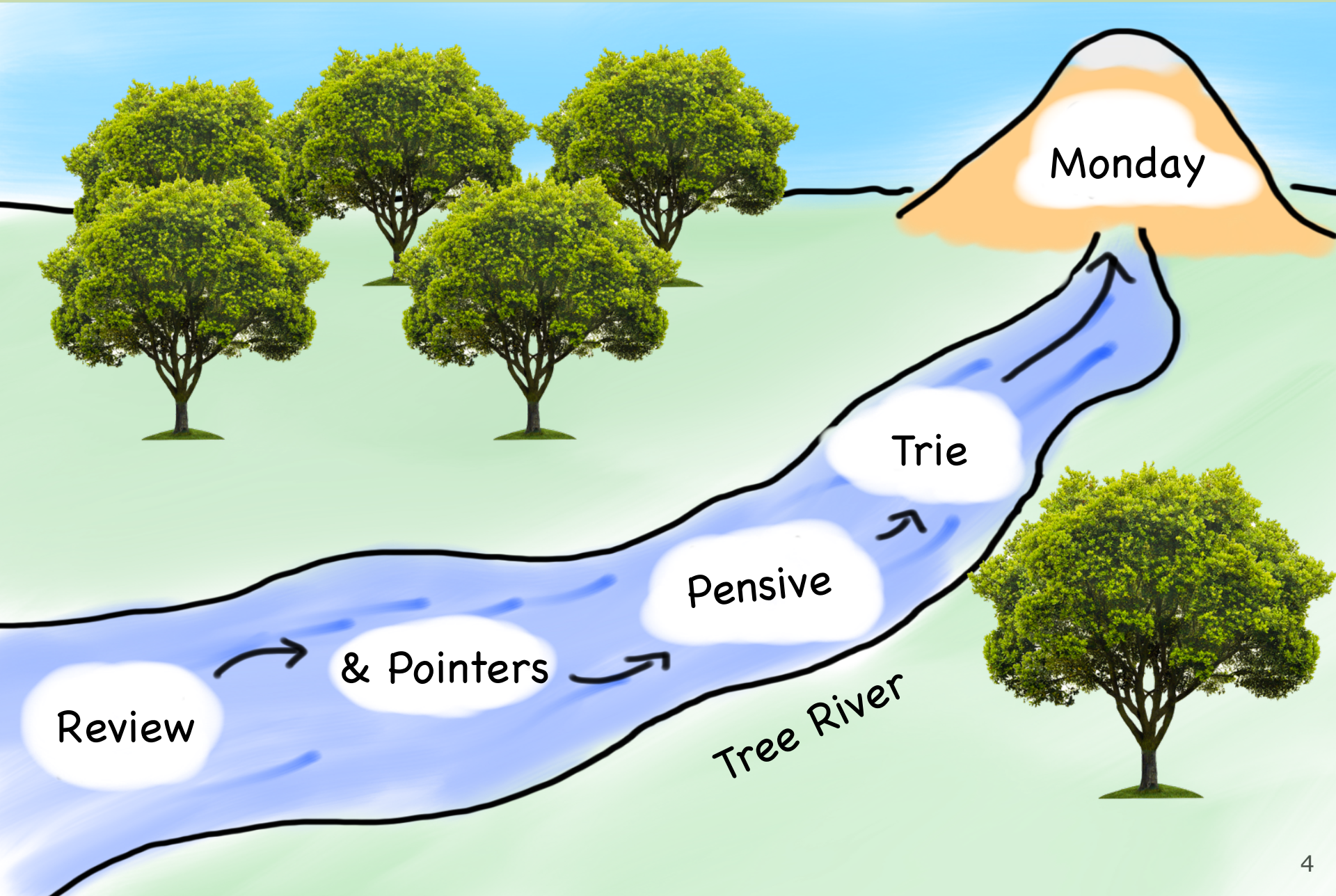


Room: **106BWIN16**

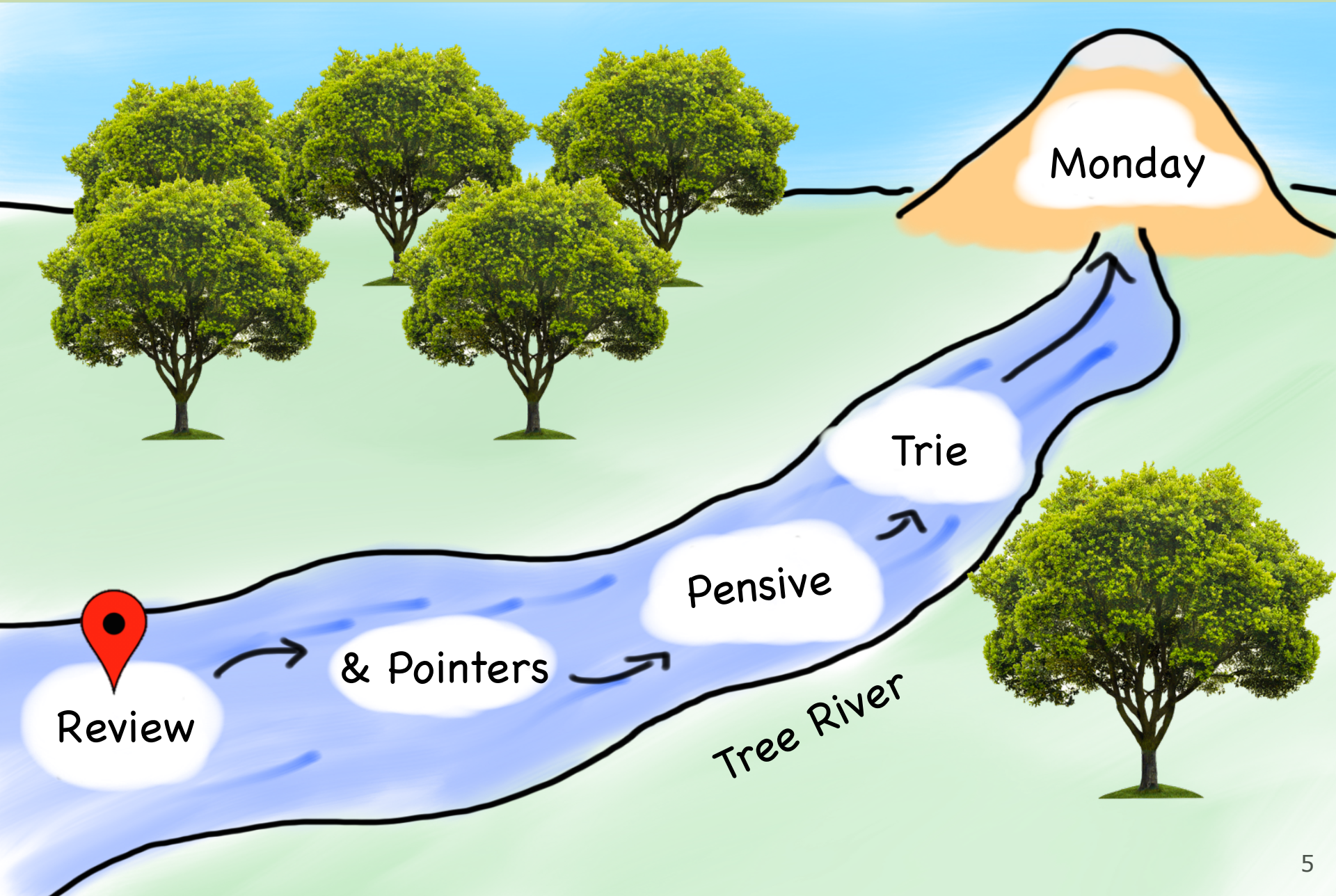
Today's Goal

1. Practice with trees
2. Pointers by reference
3. Be able to insert into a tree

Today's Route

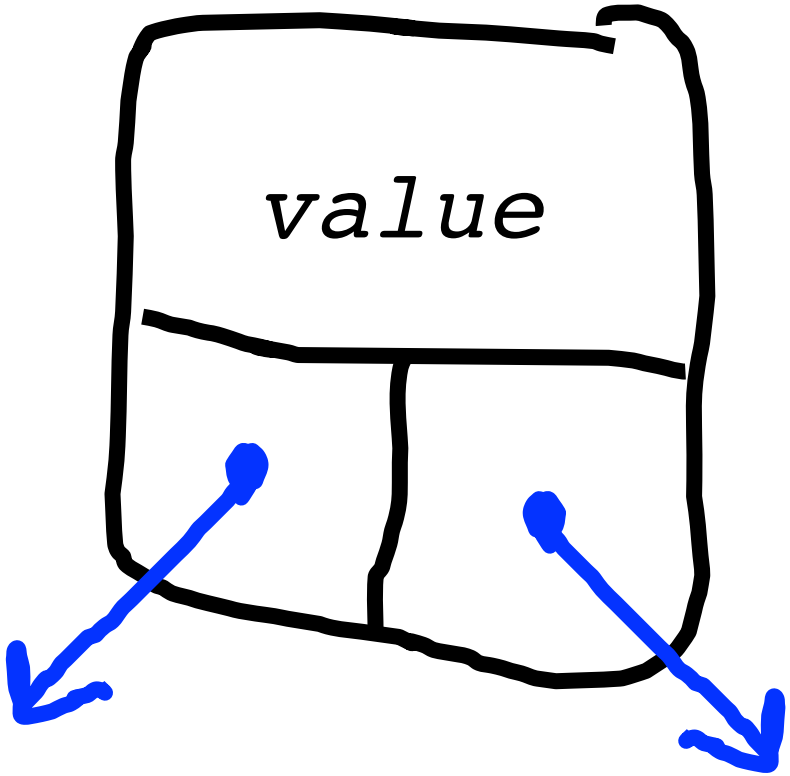


Today's Route



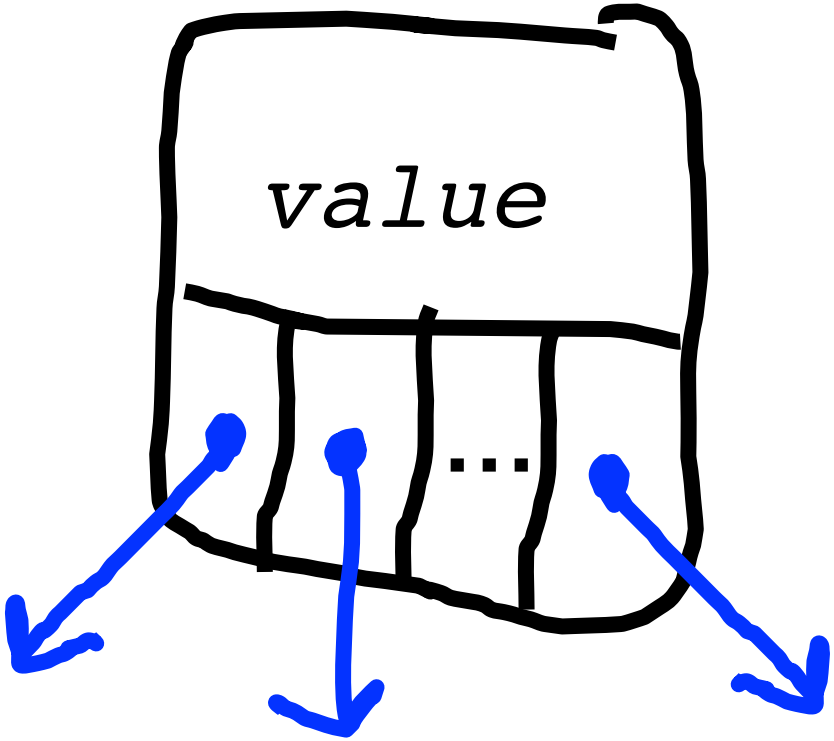
Binary Tree

```
struct Tree {
  string value;
  Tree * left;
  Tree * right;
};
```



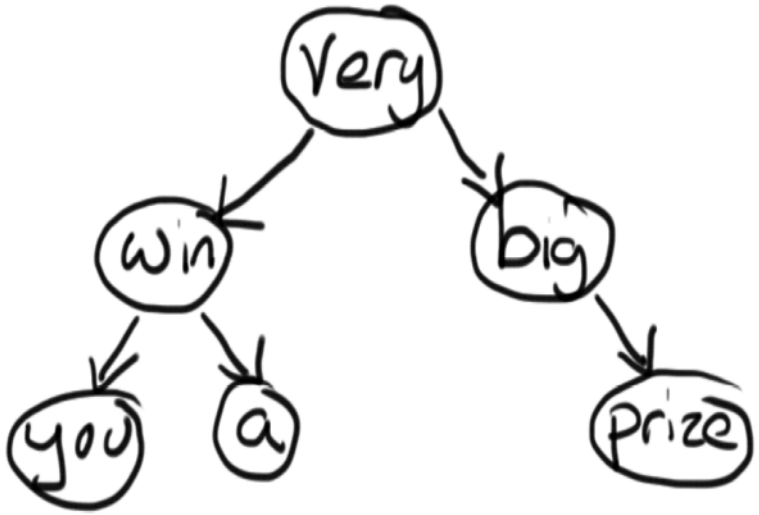
Tree

```
struct Tree {
   string value;
   Vector<Tree *> children;
};
```



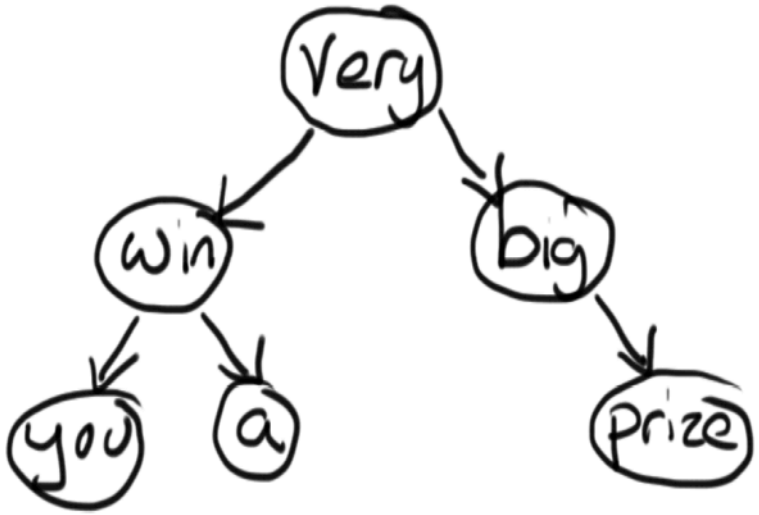
Game Show Tree

```
void doorOne(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    doorOne(tree->left);
    doorOne(tree->right);
}

void doorTwo(Tree * tree) {
    if(tree == NULL) return;
    doorTwo(tree->left);
    cout<<tree->value<<" ";
    doorTwo(tree->right);
}

Void doorThree(Tree * tree) {
    if(tree == NULL) return;
    doorThree(tree->left);
    doorThree(tree->right);
    cout<<tree->value<<" ";
}
```



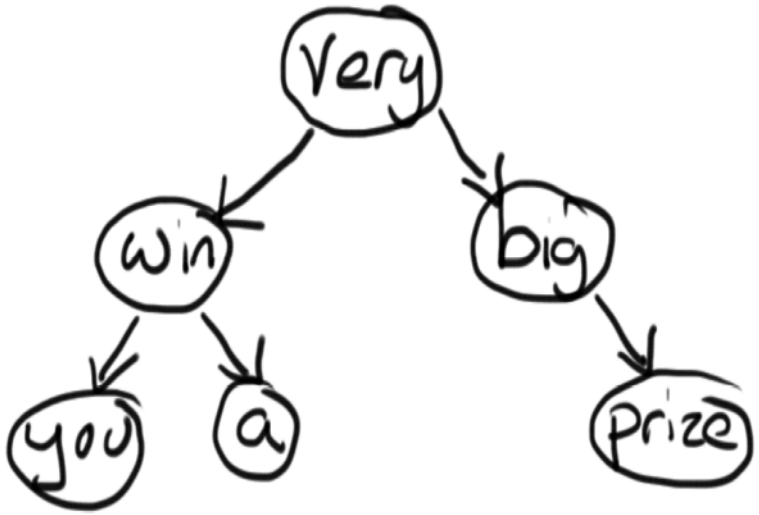
Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

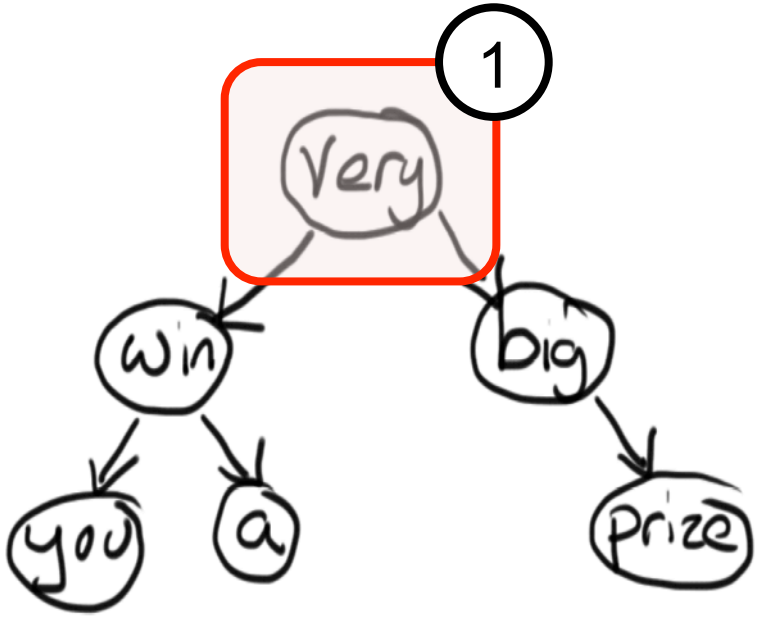
```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

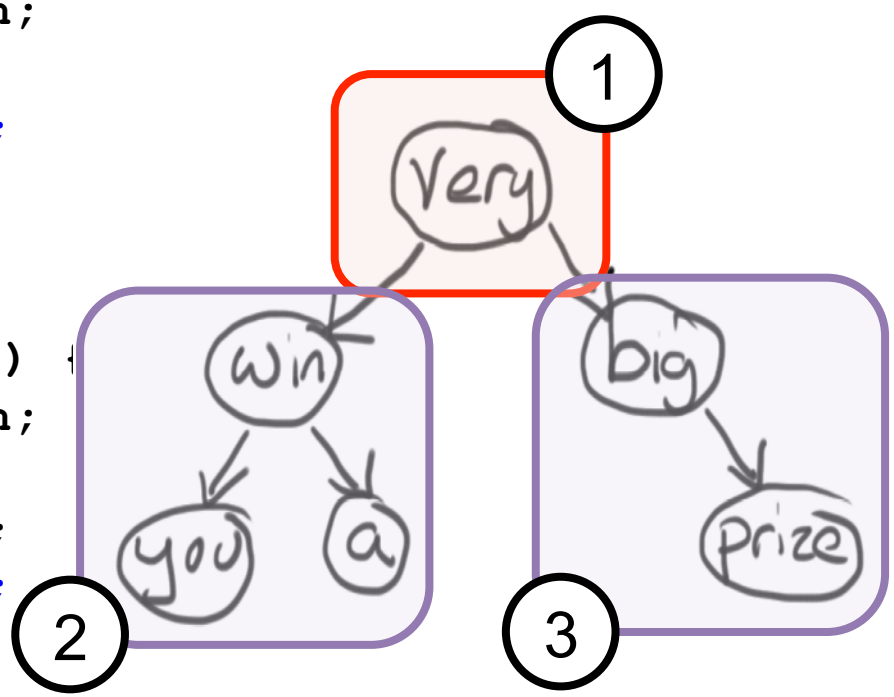
Root goes before children



Game Show Tree

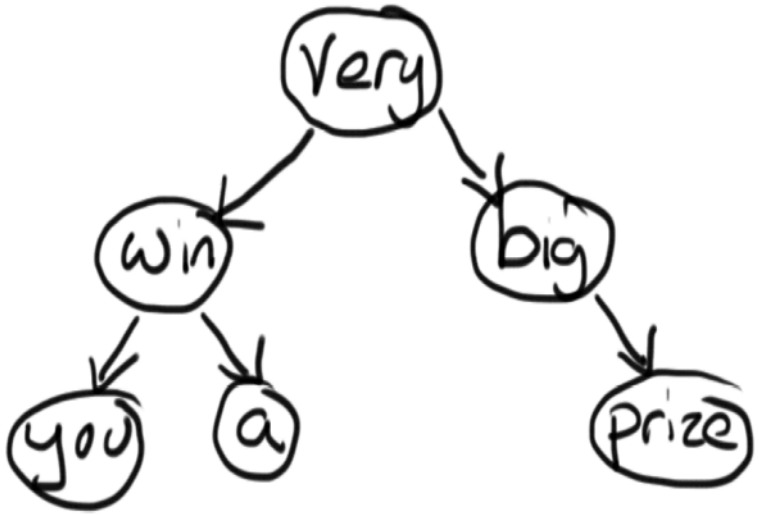
```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

Root goes before children



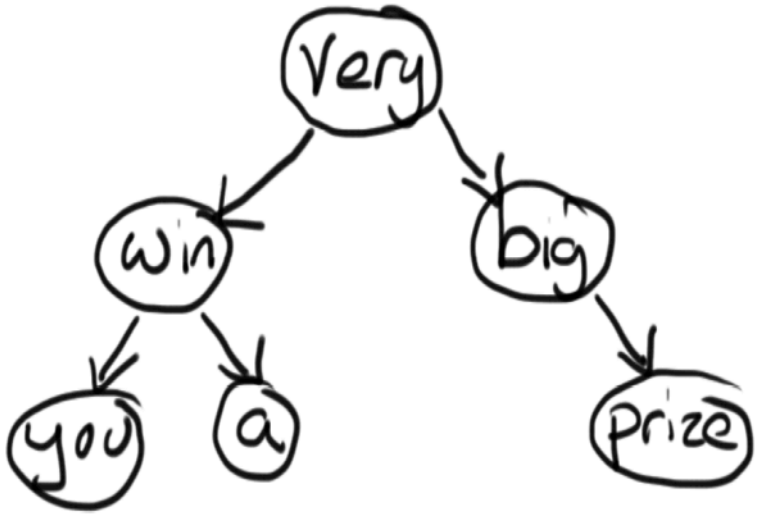
Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

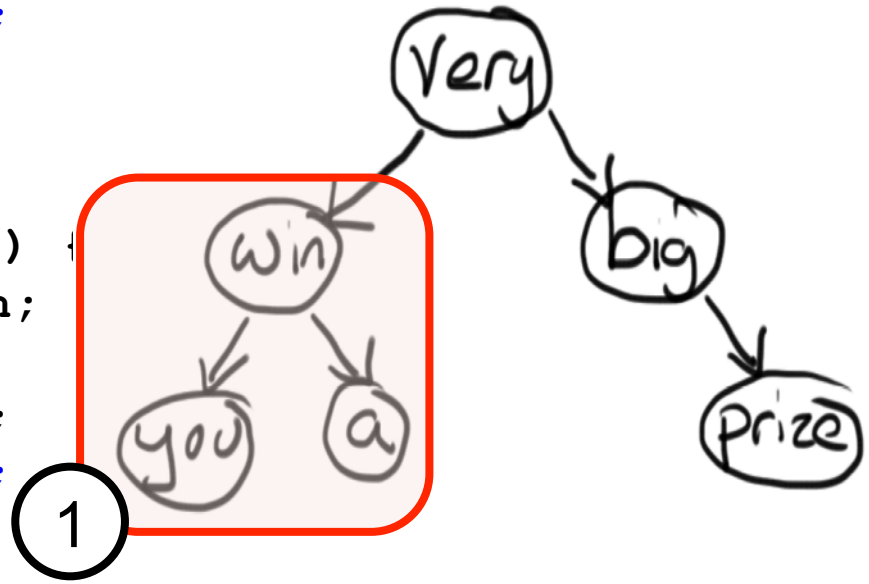
```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

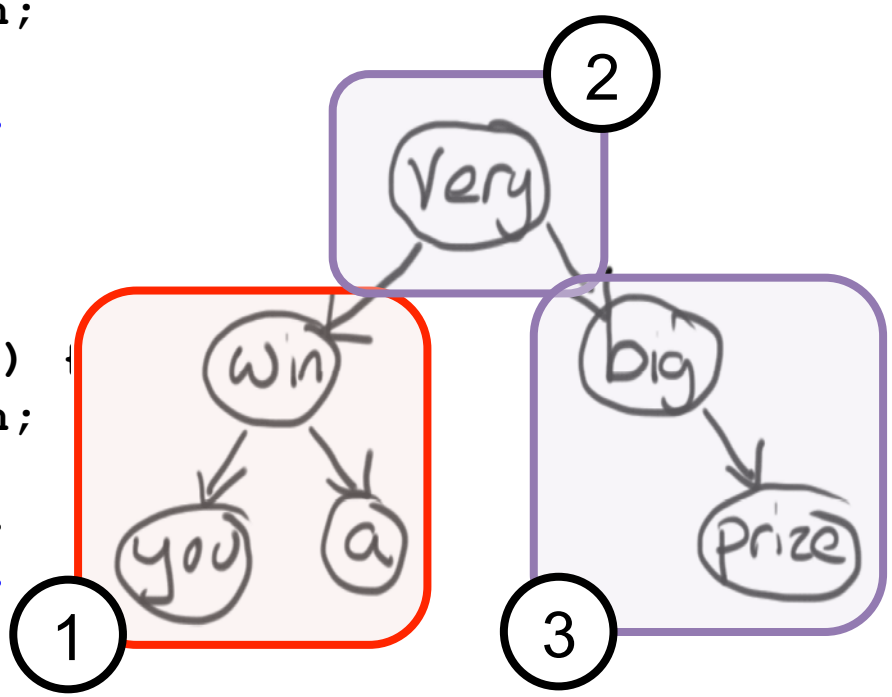
Left, Root, Right



Game Show Tree

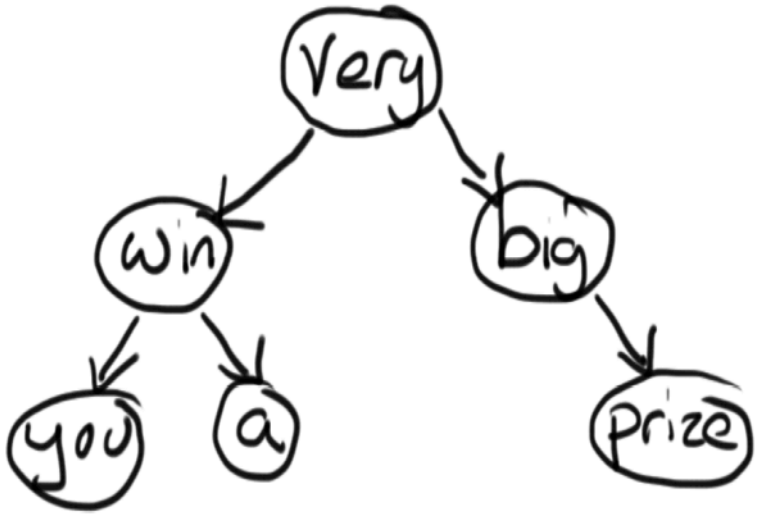
```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

Void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

Left, Root, Right



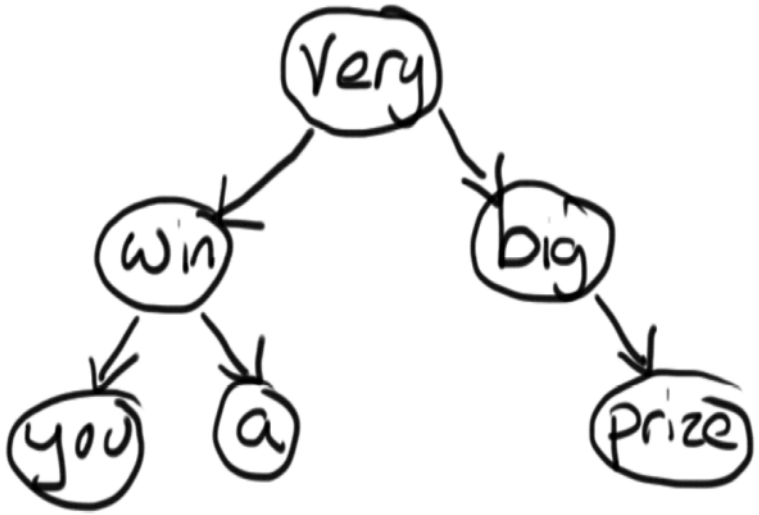
Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

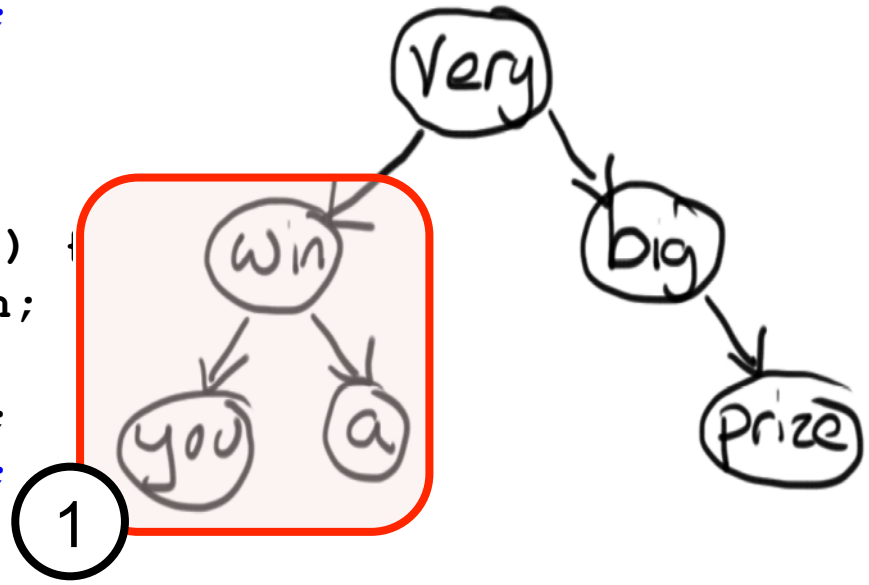
```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```



Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

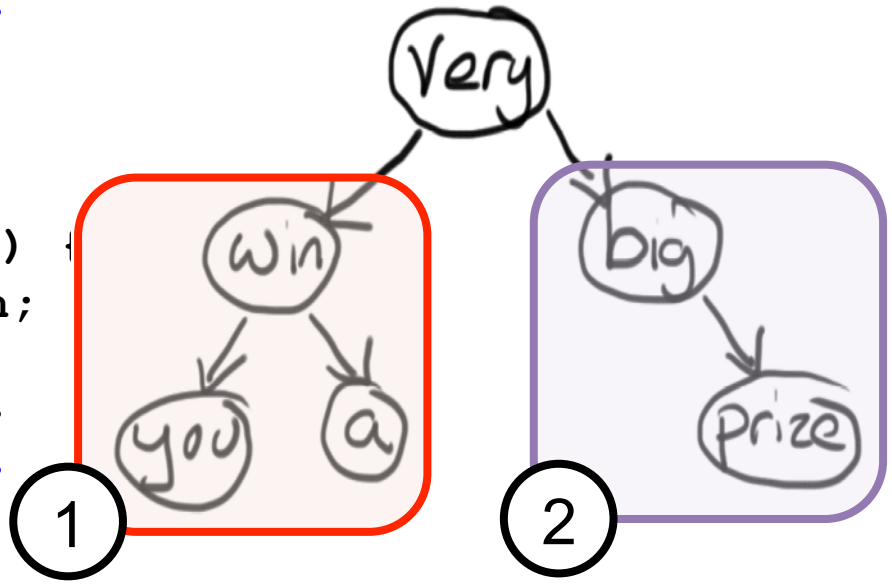
Children go before root



Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

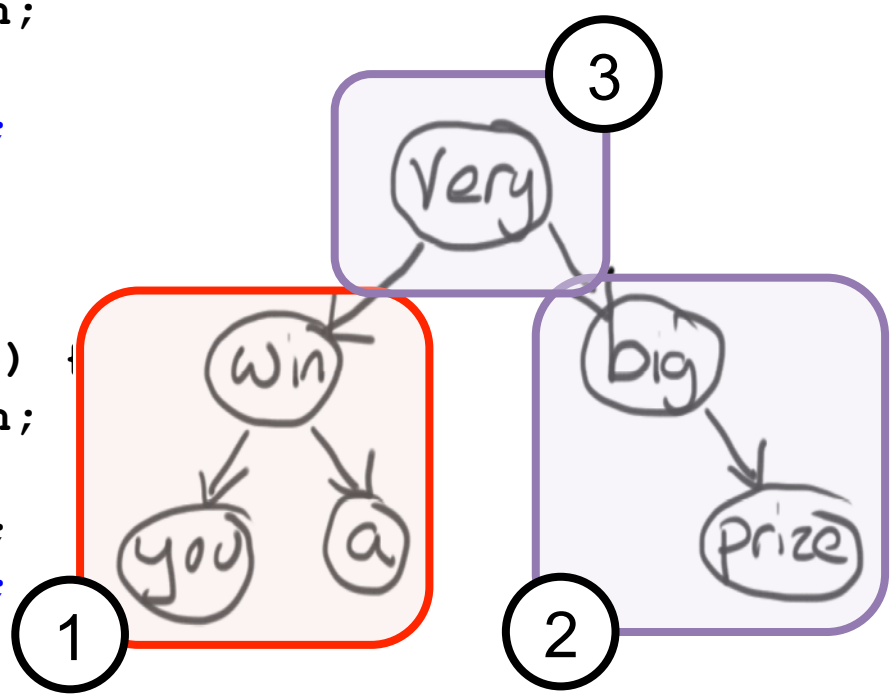
Children go before root



Game Show Tree

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    cout<<tree->value<<" ";
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    cout<<tree->value<<" ";
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    cout<<tree->value<<" ";
}
```

Children go before root



Free Tree

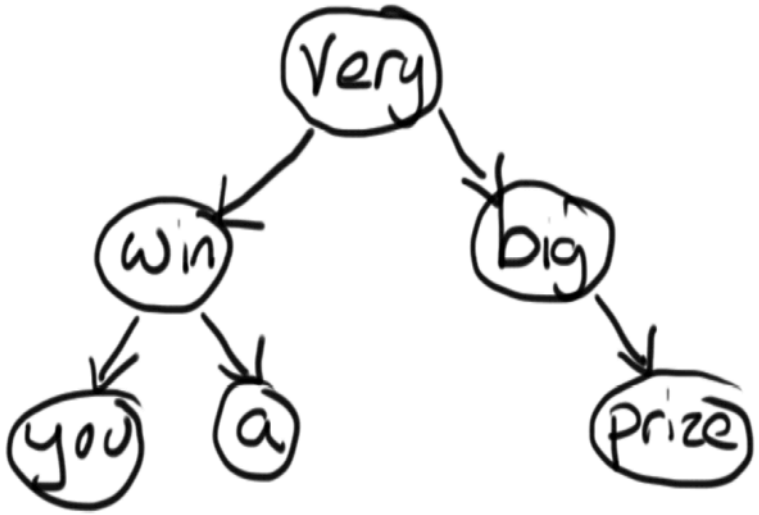
STUDENT

```
void preOrder(Tree * tree) {
    if(tree == NULL) return;
    delete tree;
    preOrder(tree->left);
    preOrder(tree->right);
}

void inOrder(Tree * tree) {
    if(tree == NULL) return;
    inOrder(tree->left);
    delete tree;
    inOrder(tree->right);
}

void postOrder(Tree * tree) {
    if(tree == NULL) return;
    postOrder(tree->left);
    postOrder(tree->right);
    delete tree
}
```

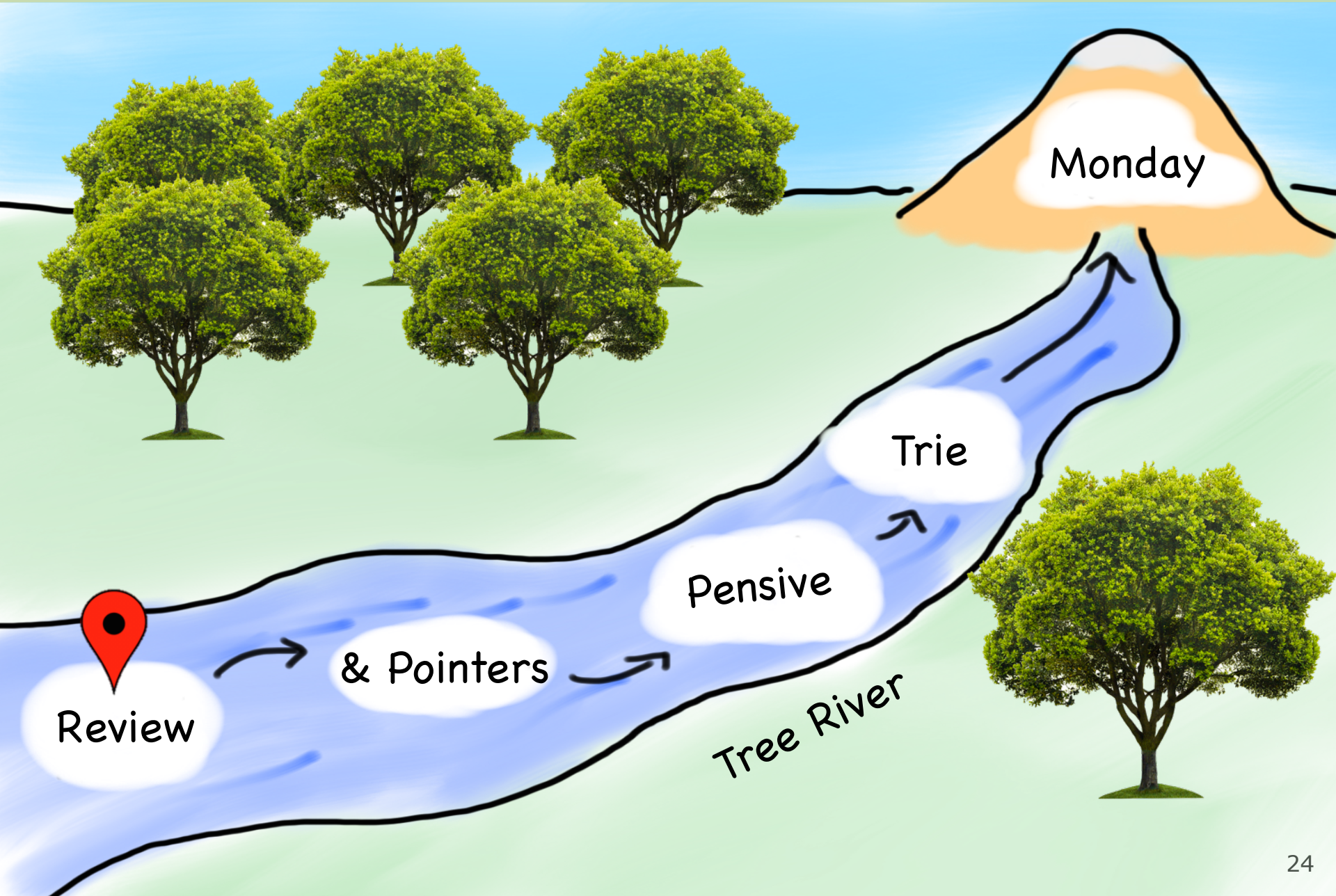
a) preOrder

b) inOrder

c) postOrder



Today's Route



You can achieve pass by reference using pointers

Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

Pass by Pointer


```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

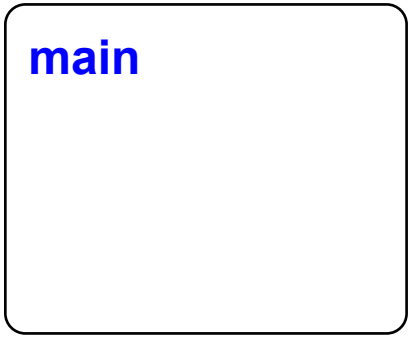
a) 5 b) 10 c) random d) crash

Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

Heap



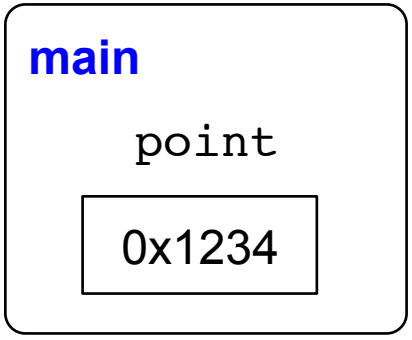
Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

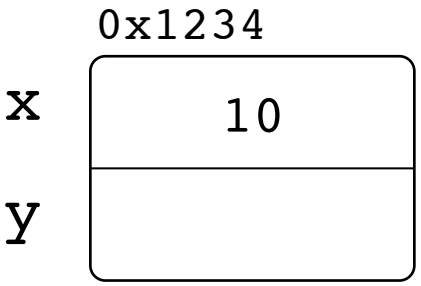
Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

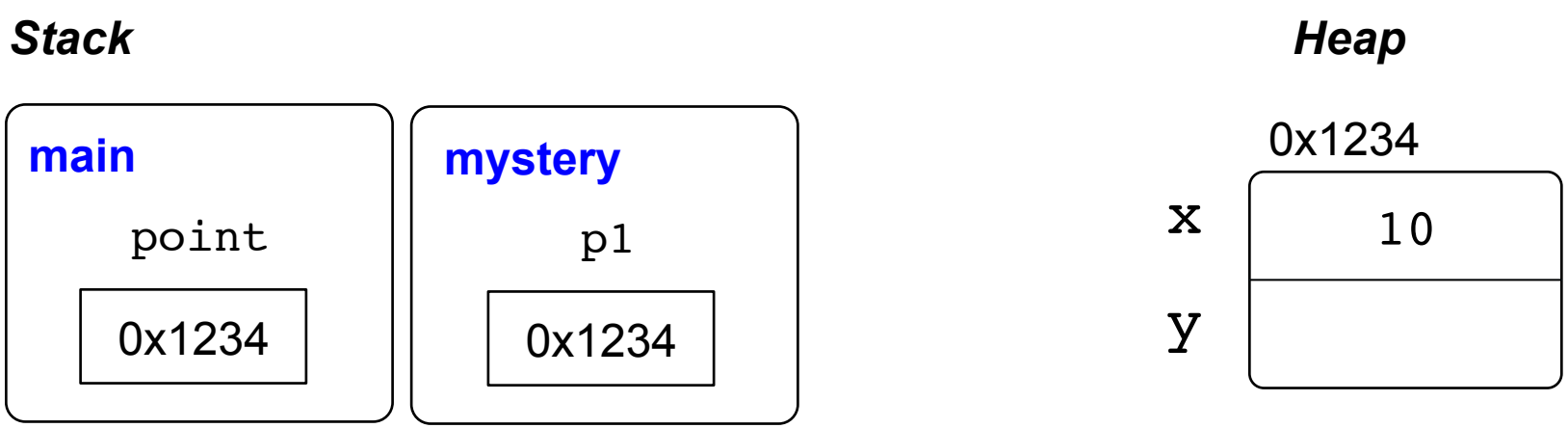


Heap



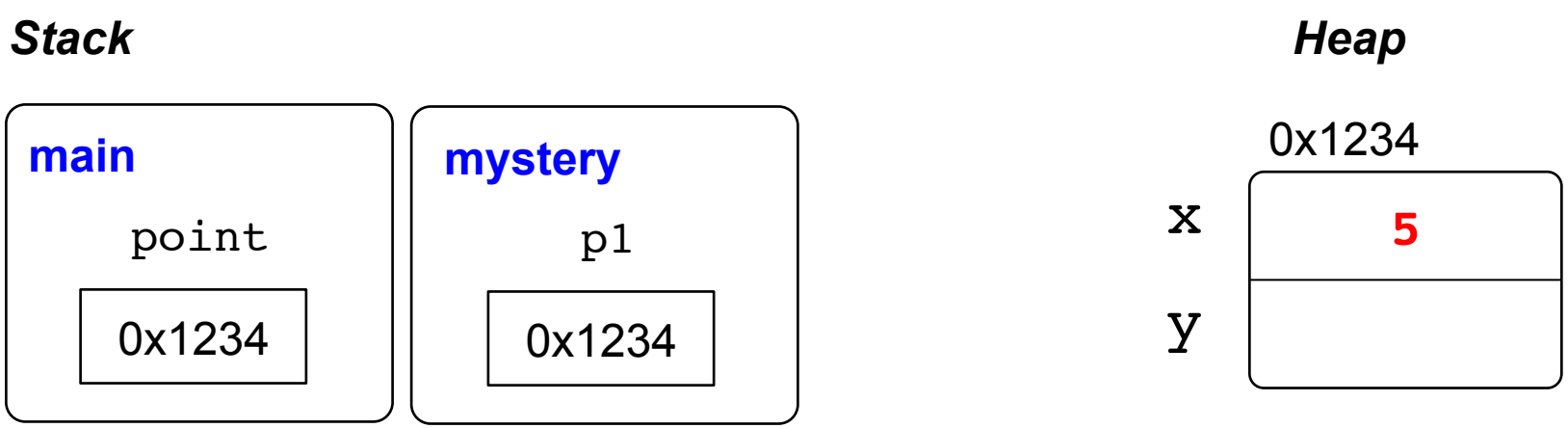
Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```



Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```



Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

Pass by Pointer

```
void mystery(Point * p1) {
    p1->x = 5;
}

int main() {
    Point * point = new Point;
    point->x = 10;
    mystery(point);
    cout << point->x << endl;
}
```

What does this do?

Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

Heap

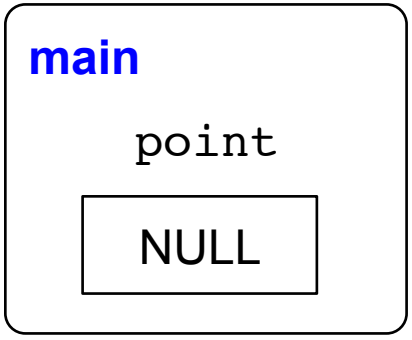
main

Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

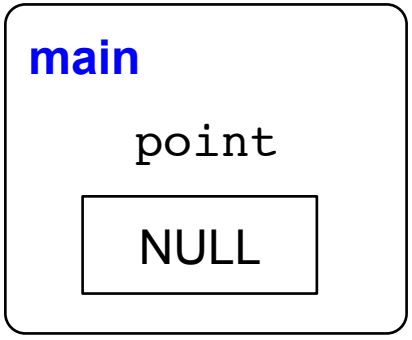
Heap



Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

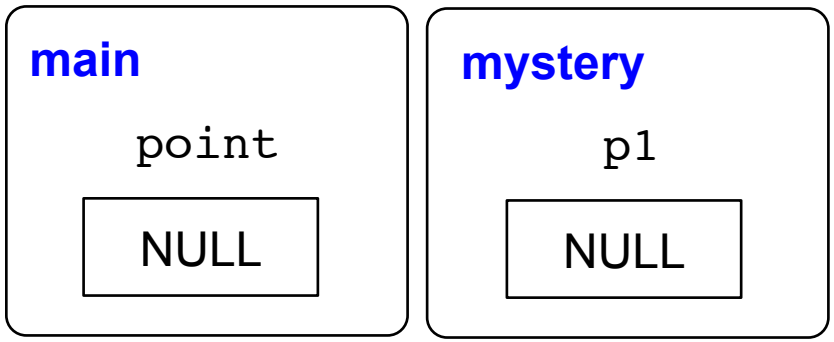
Stack ***Heap***



Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

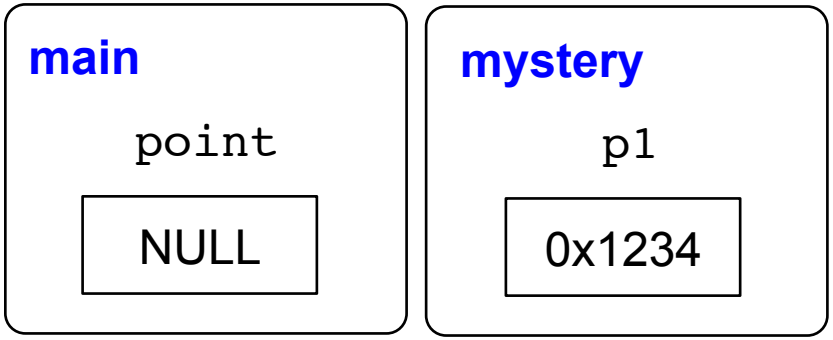
Stack ***Heap***



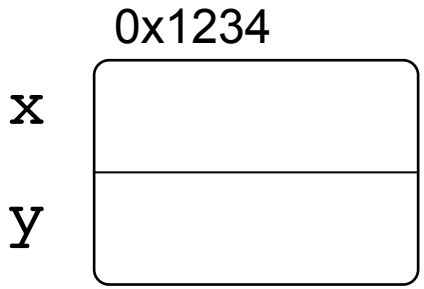
Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack



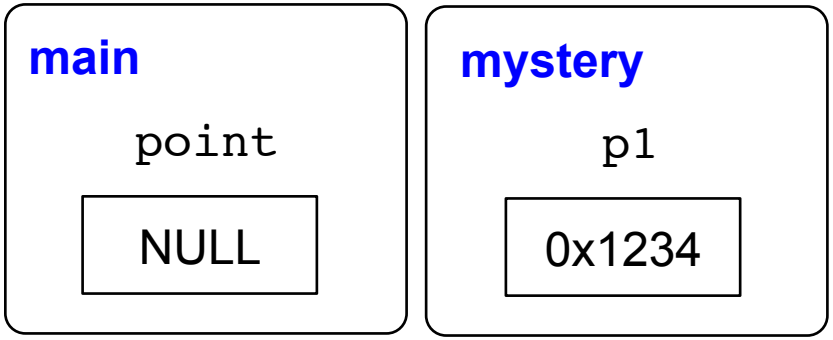
Heap



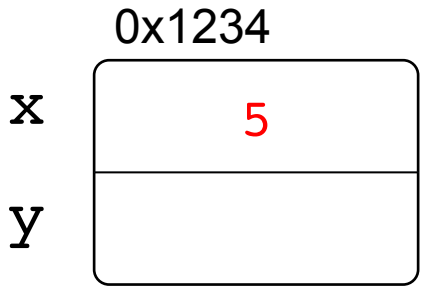
Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack



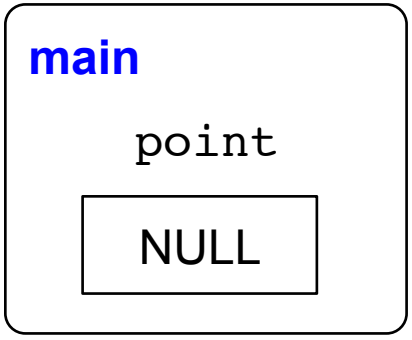
Heap



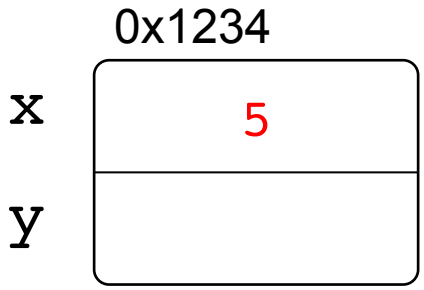
Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

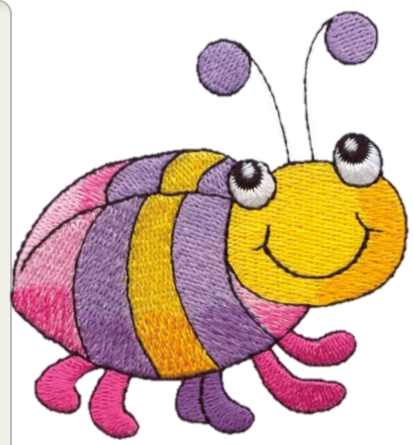


Heap



Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```



Stack

main

point

NULL

Heap

0x1234

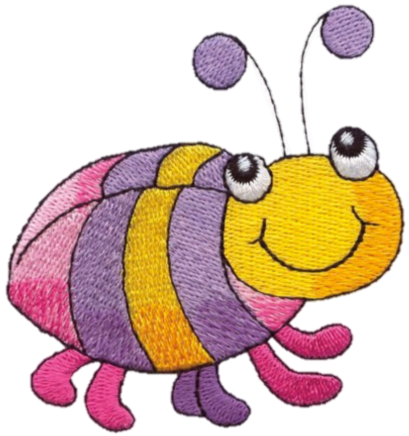
x | 5

y |

What went wrong?

Pass by Pointer

```
void mystery(Point * p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```



How about now?

Pointer by Reference

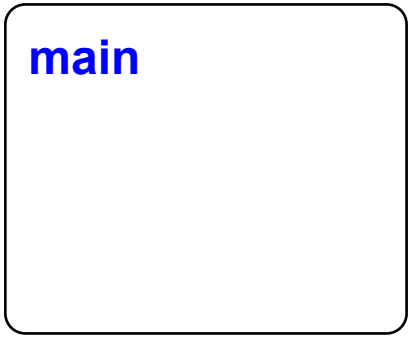
```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

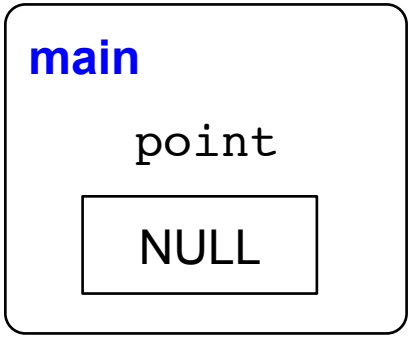
Heap



Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack ***Heap***

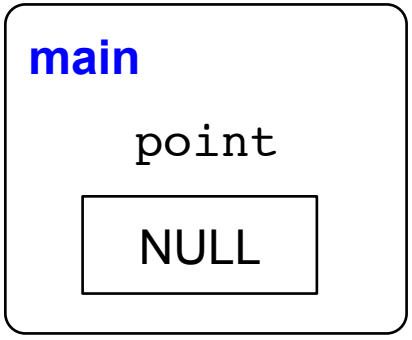


Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

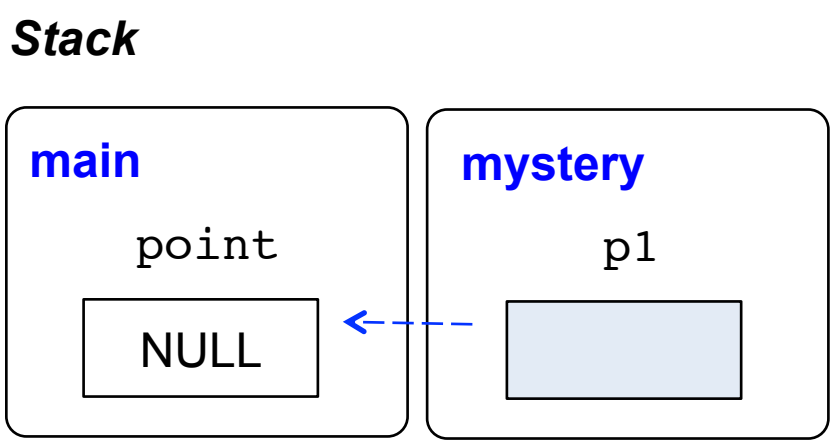
Stack

Heap



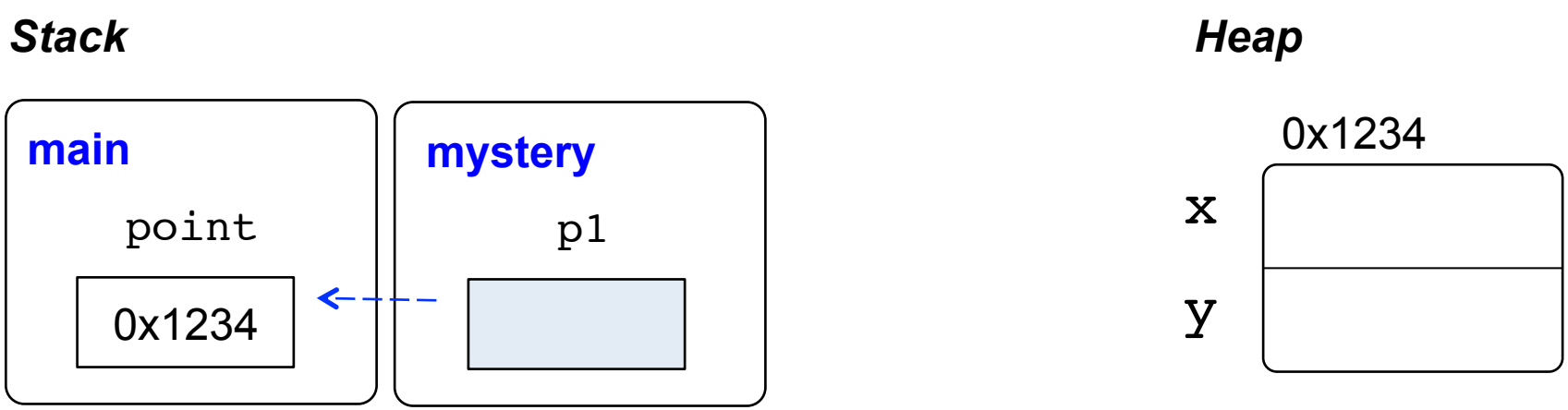
Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```



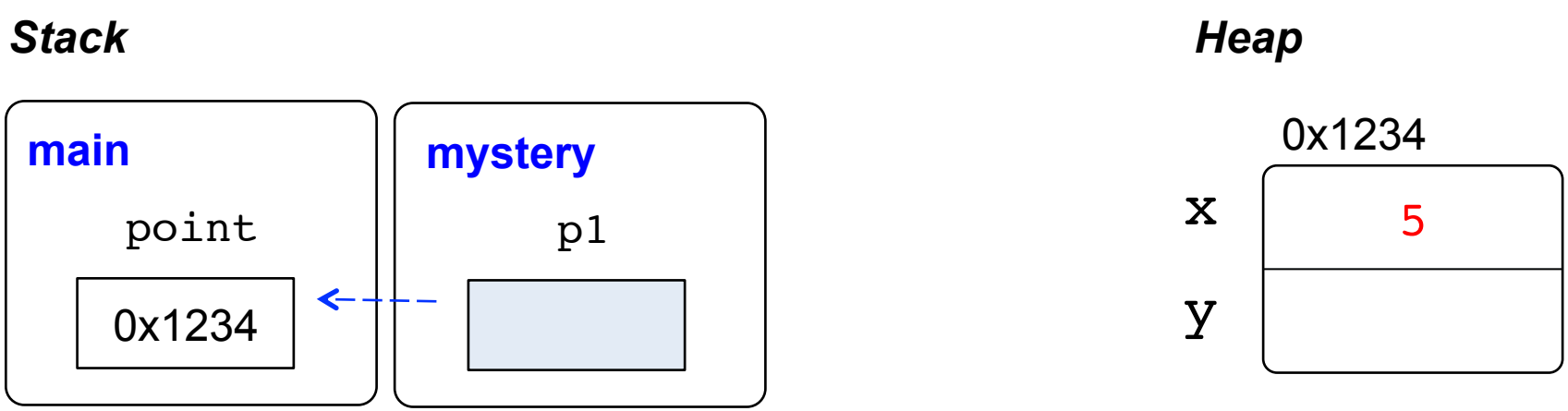
Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```



Pointer by Reference

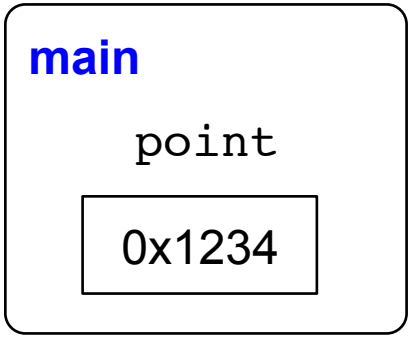
```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```



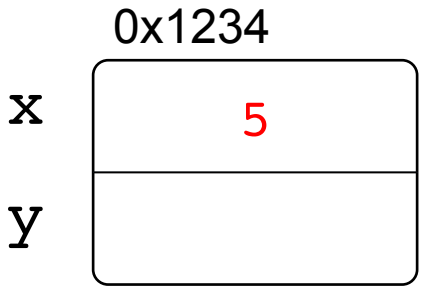
Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack



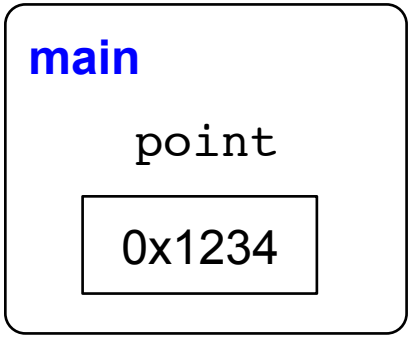
Heap



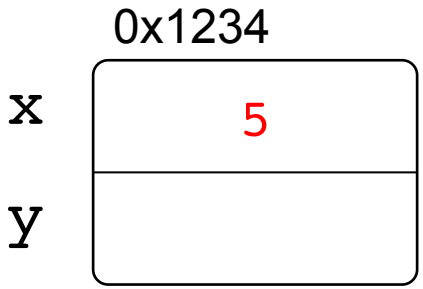
Pointer by Reference

```
void mystery(Point * & p1) {
    p1 = new Point;
    p1->x = 5;
}

int main() {
    Point * point = NULL;
    mystery(point);
    cout << point->x << endl;
}
```

Stack

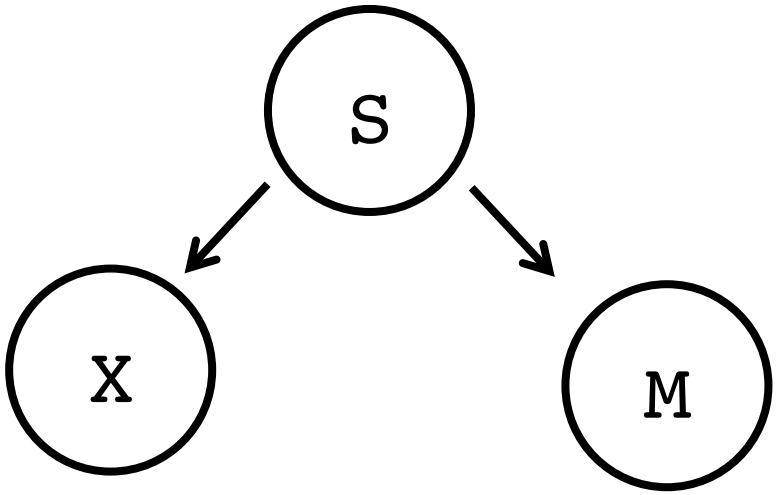


Heap

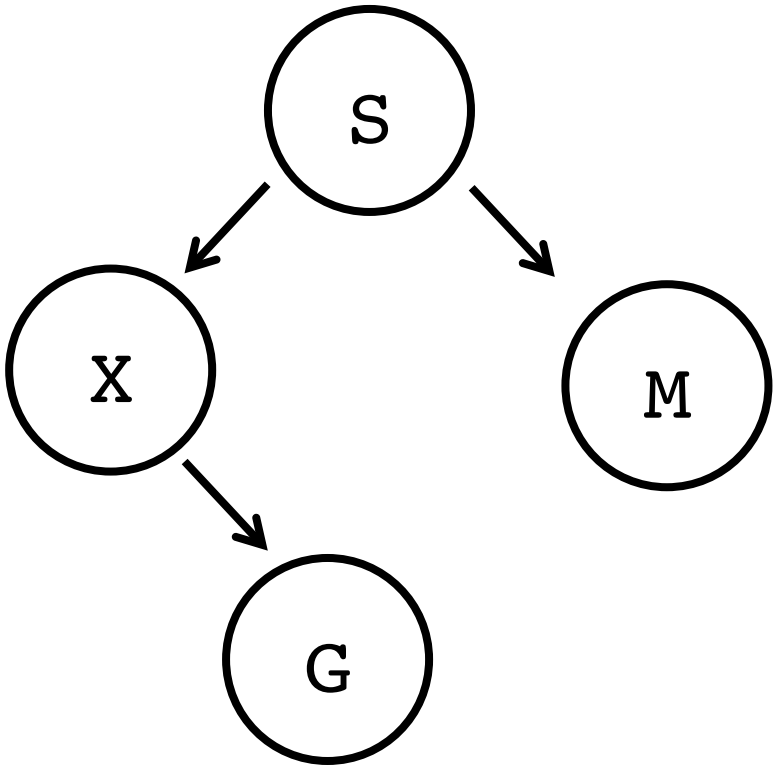


Add Random Leaf

Add Random Leaf

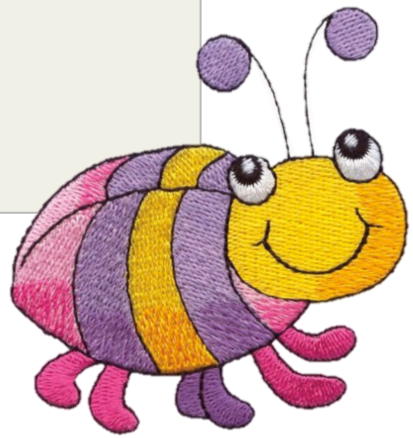


Add Random Leaf



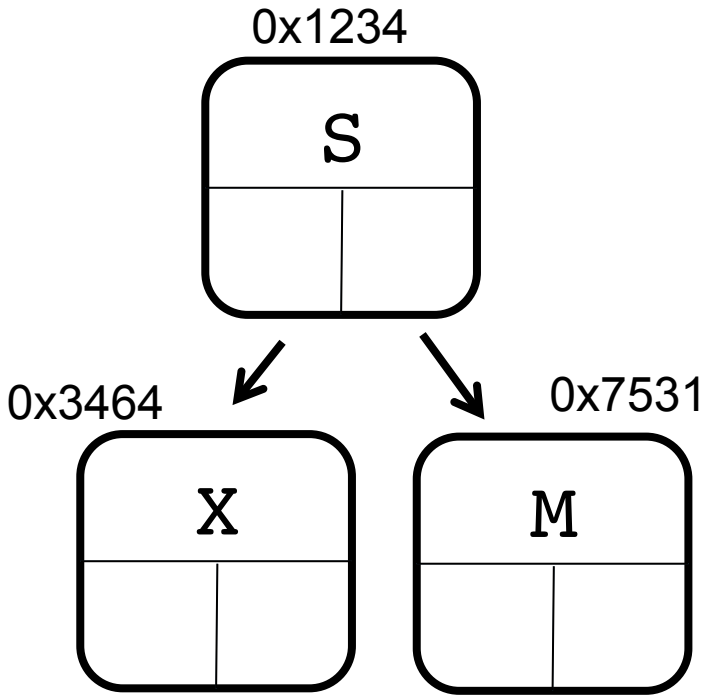
Add Random Leaf

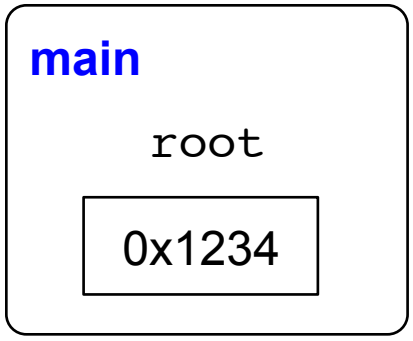
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```



Add Random Leaf

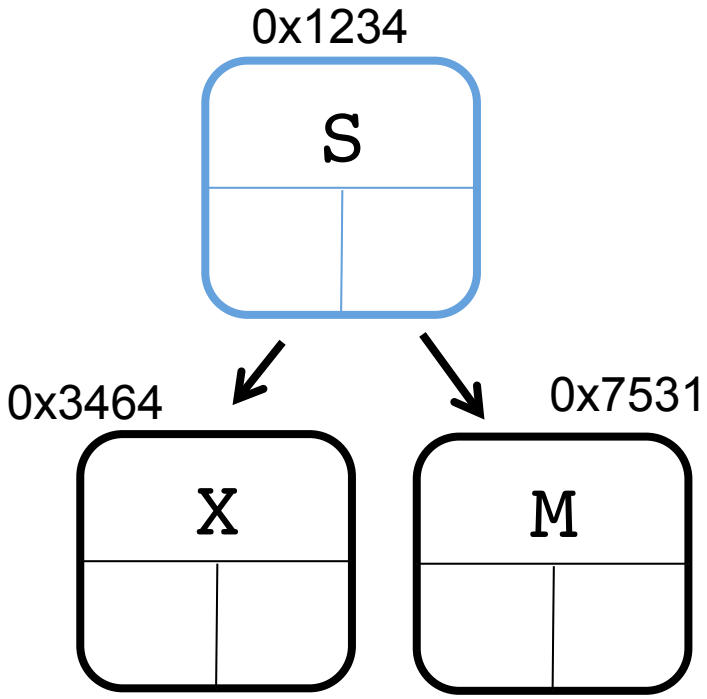
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

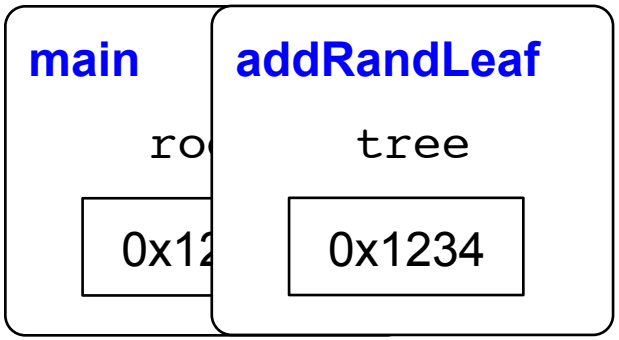




Add Random Leaf

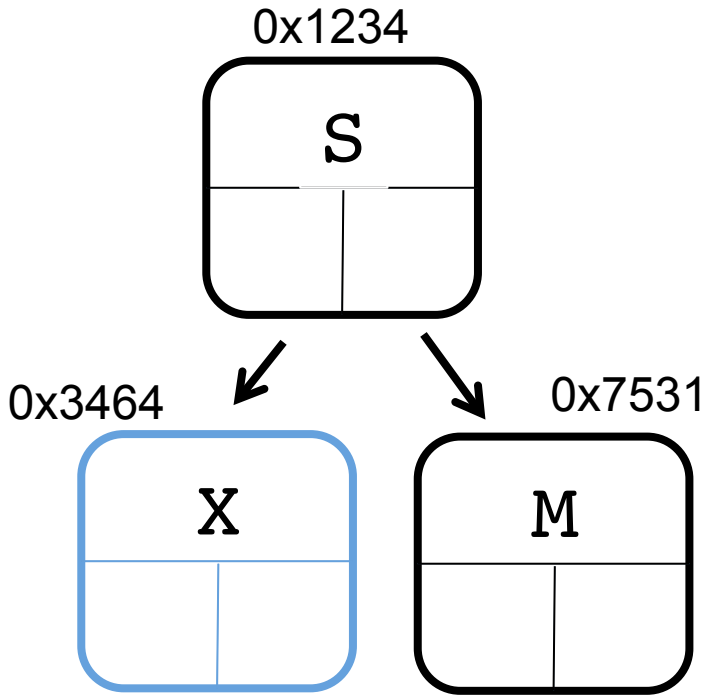
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

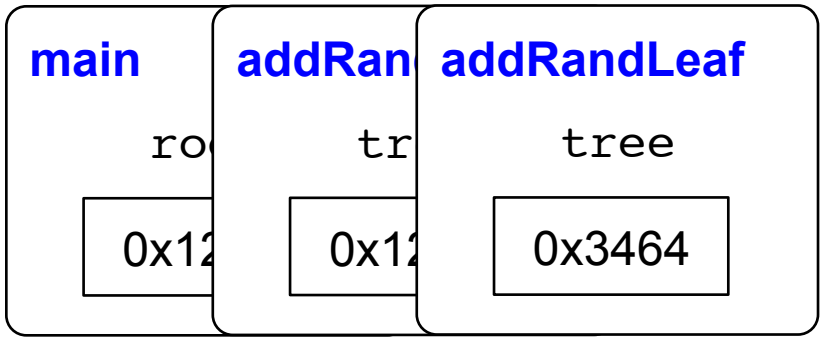




Add Random Leaf

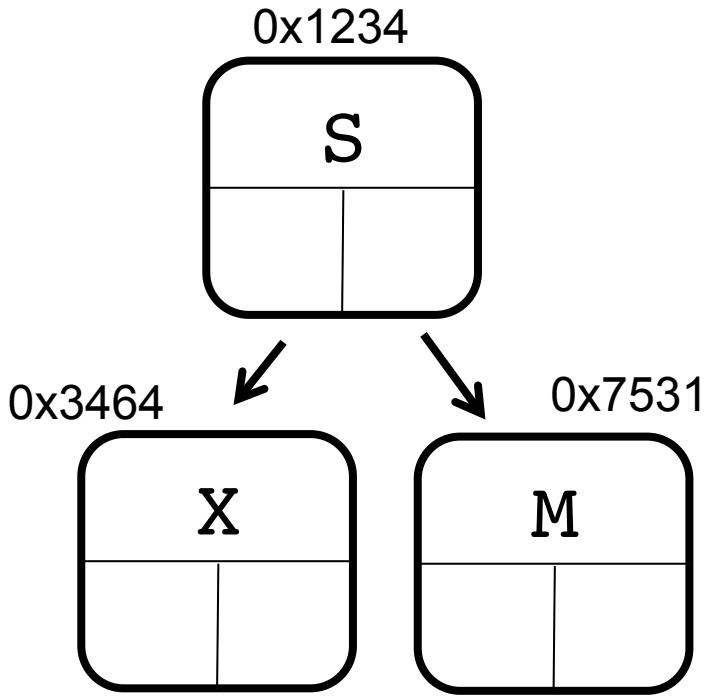
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

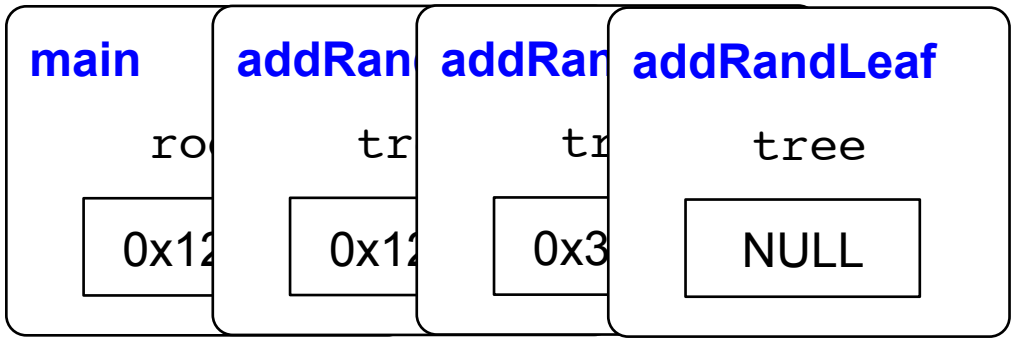




Add Random Leaf

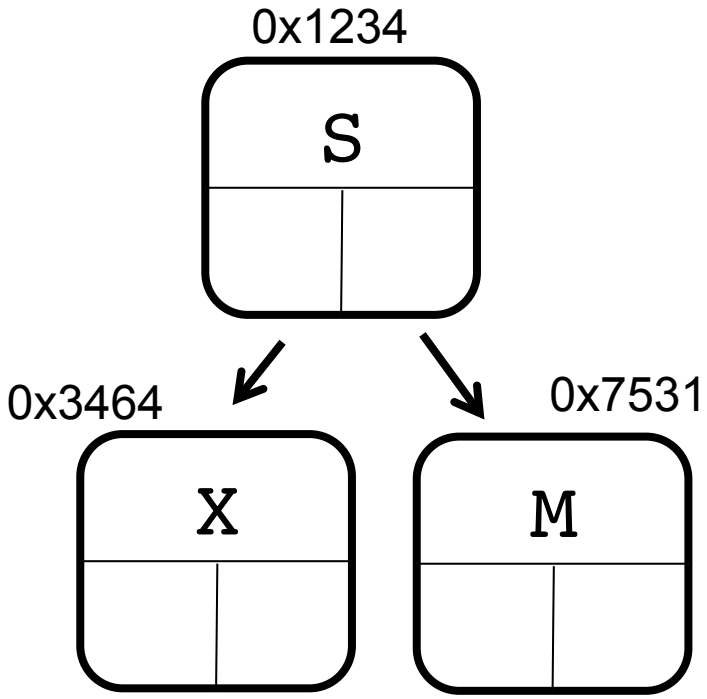
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

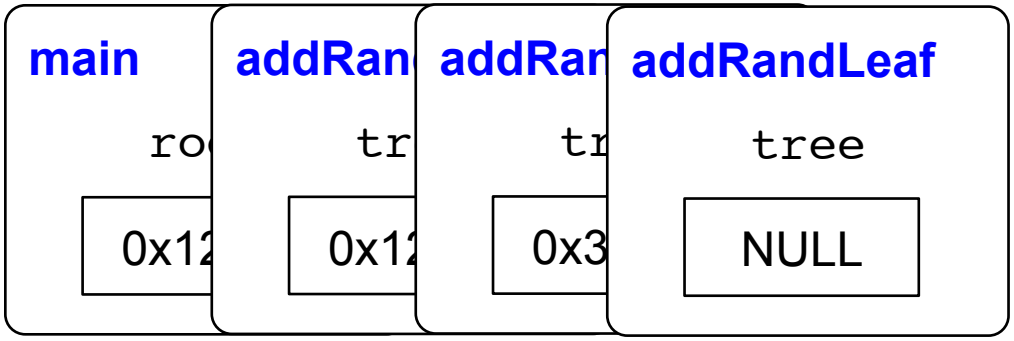




Add Random Leaf

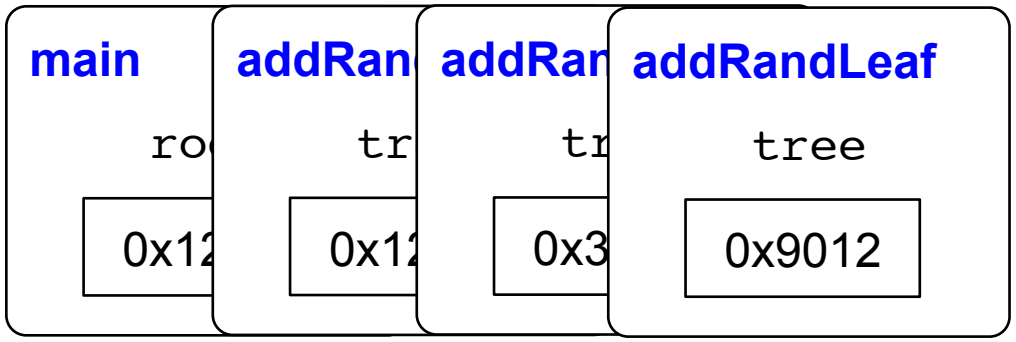
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

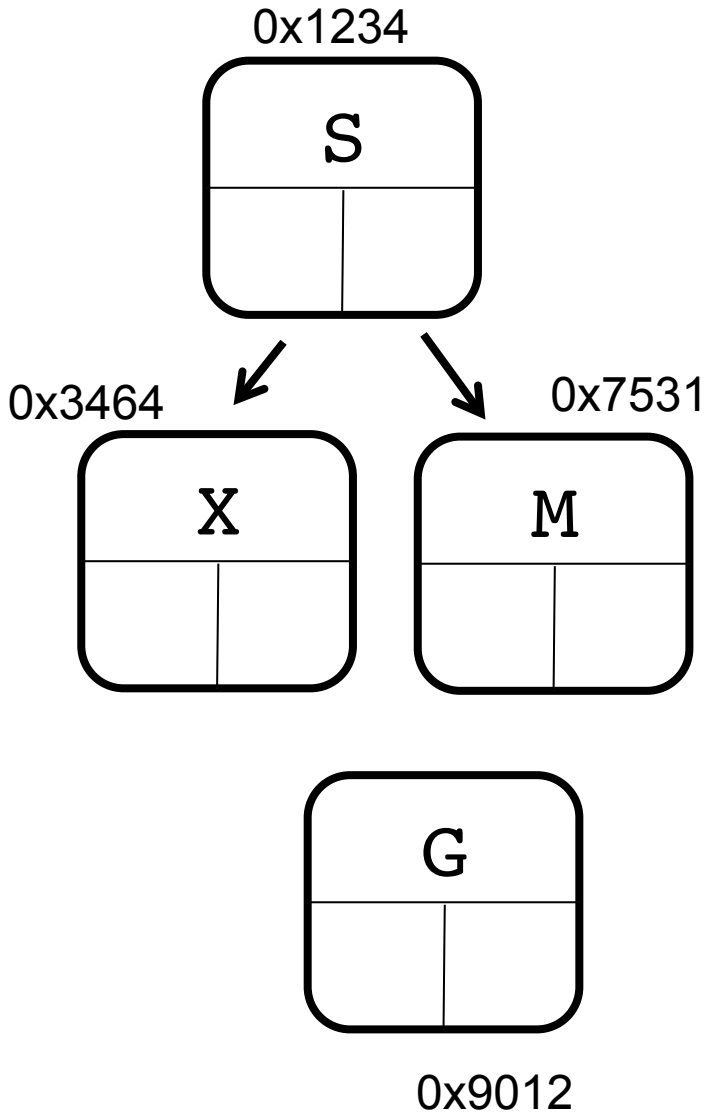




Add Random Leaf

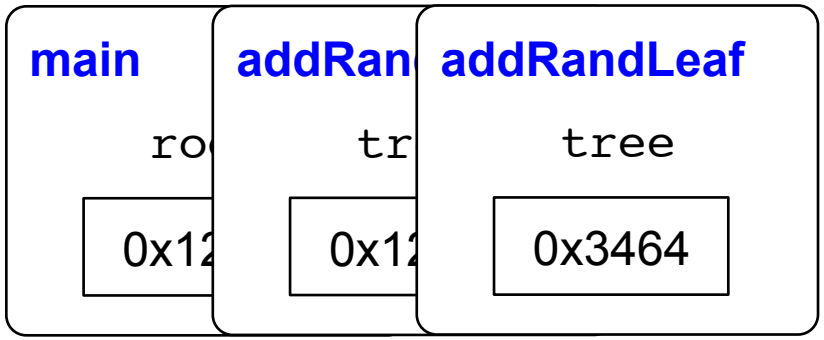
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

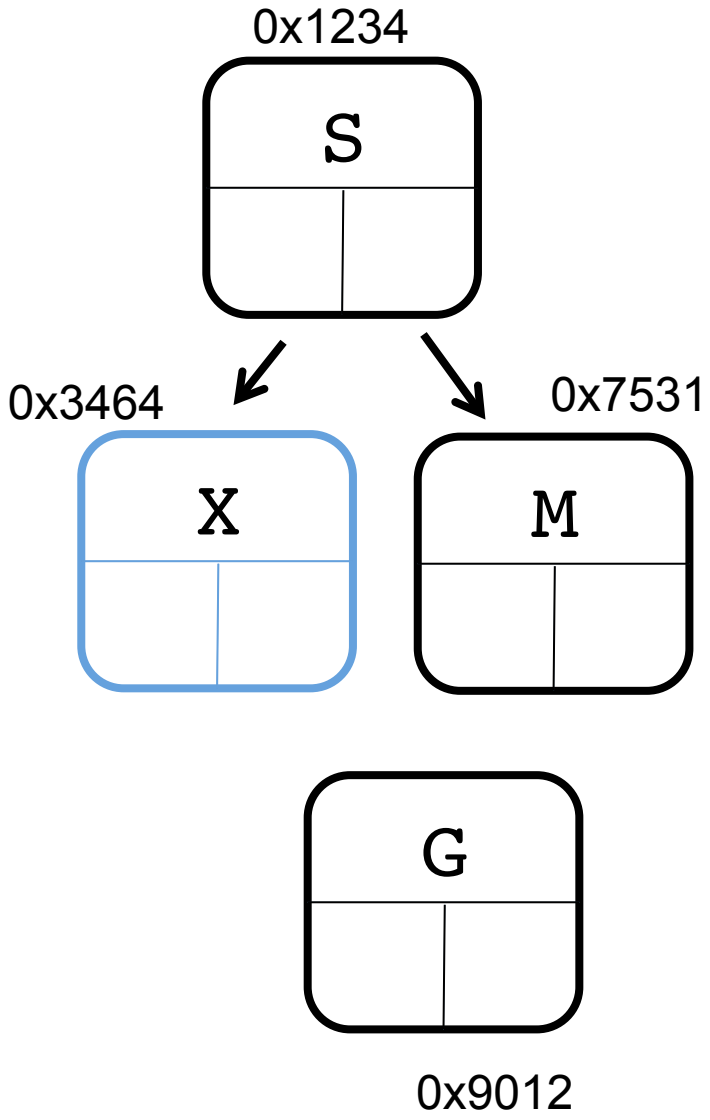




Add Random Leaf

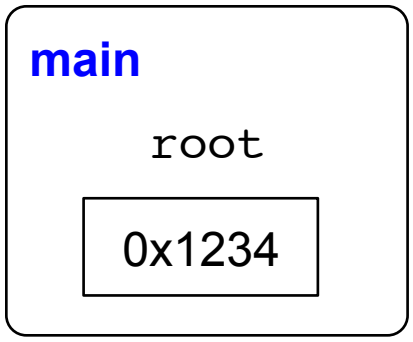
```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

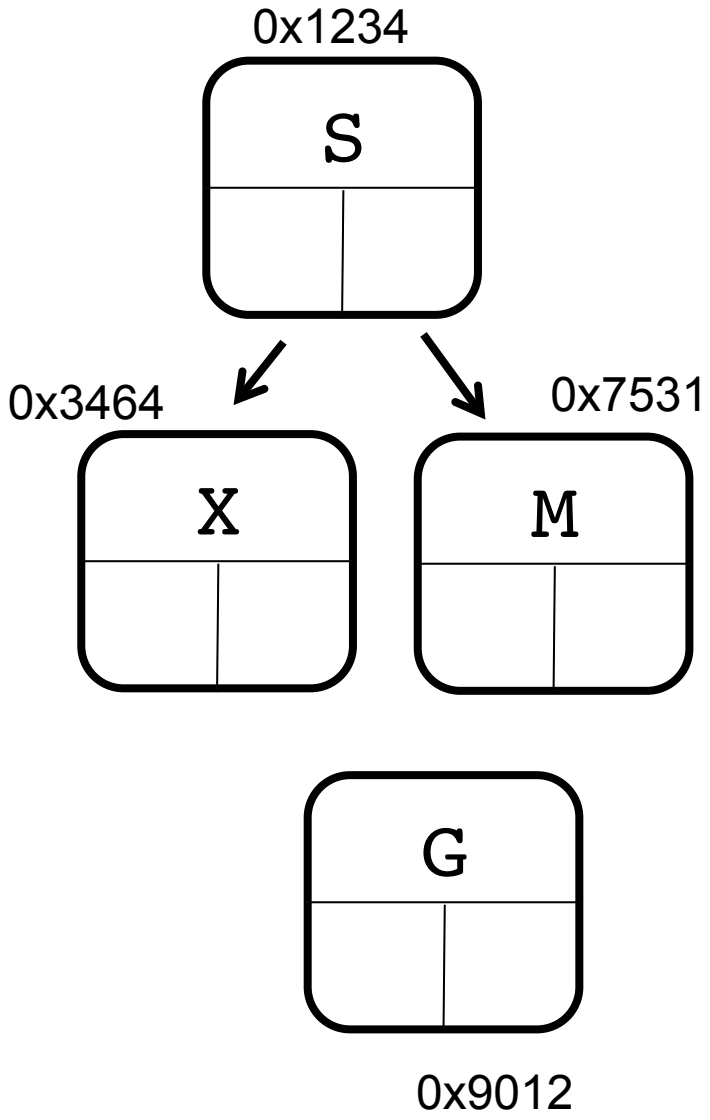




Add Random Leaf

```
void addRandomLeaf(Tree * tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```



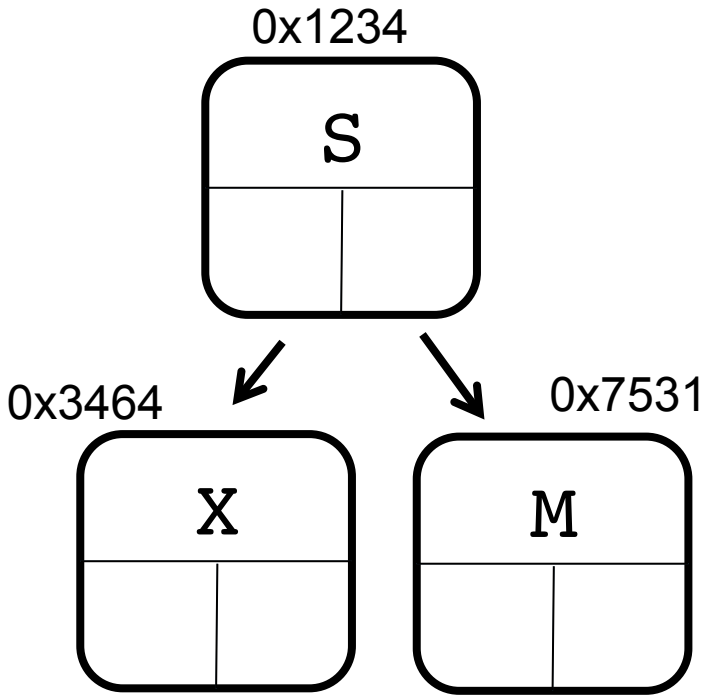


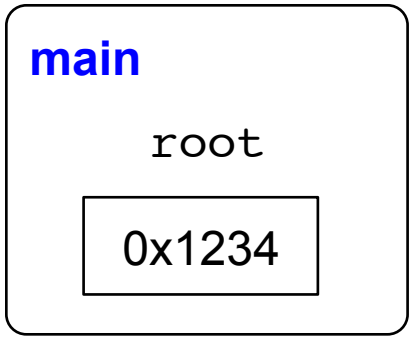
Add Random Leaf

```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

Add Random Leaf

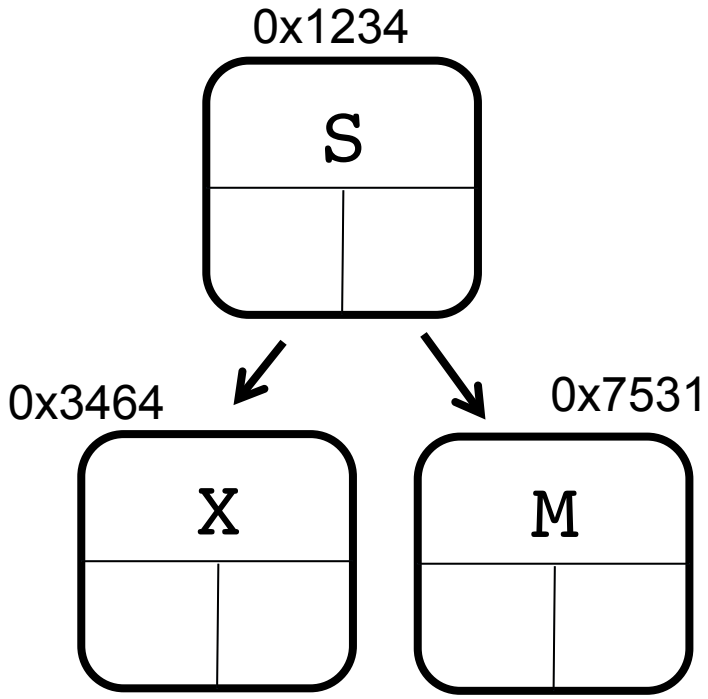
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

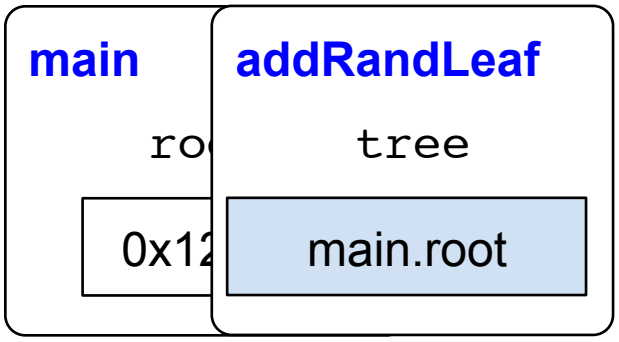




Add Random Leaf

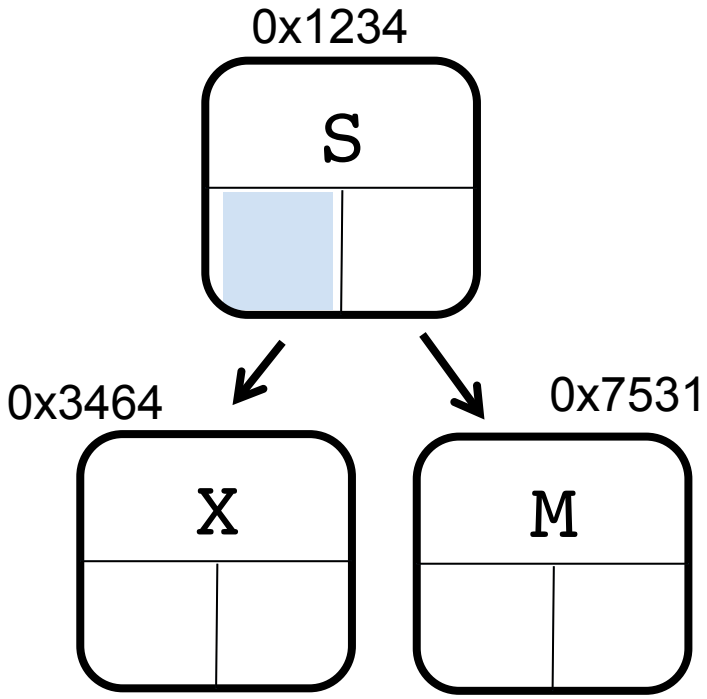
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

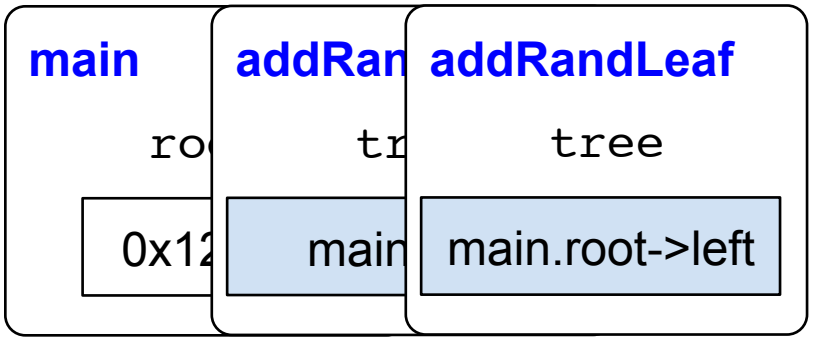




Add Random Leaf

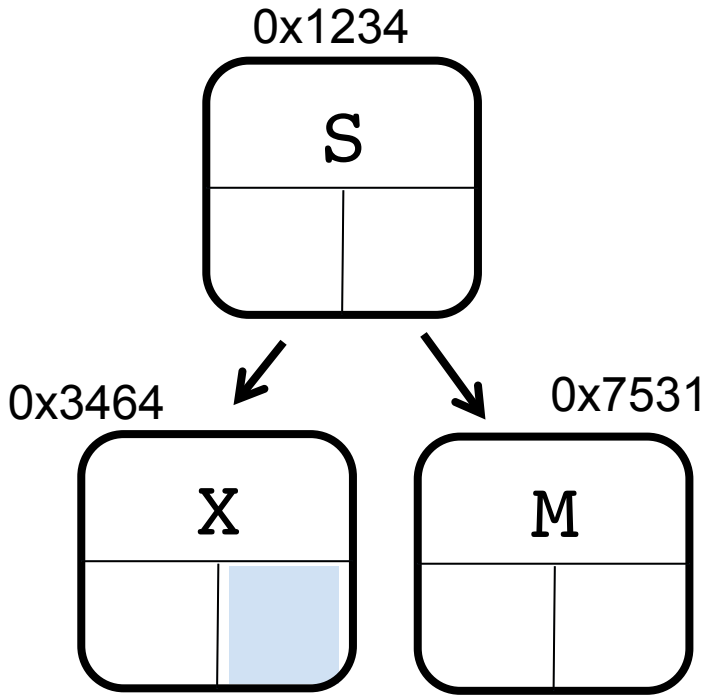
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

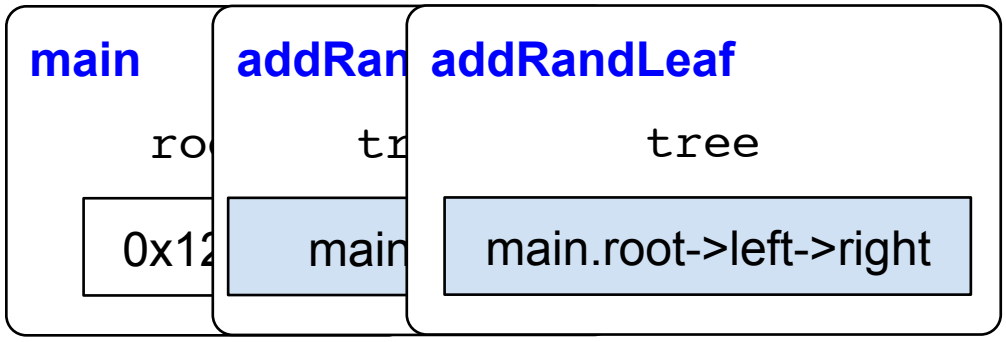




Add Random Leaf

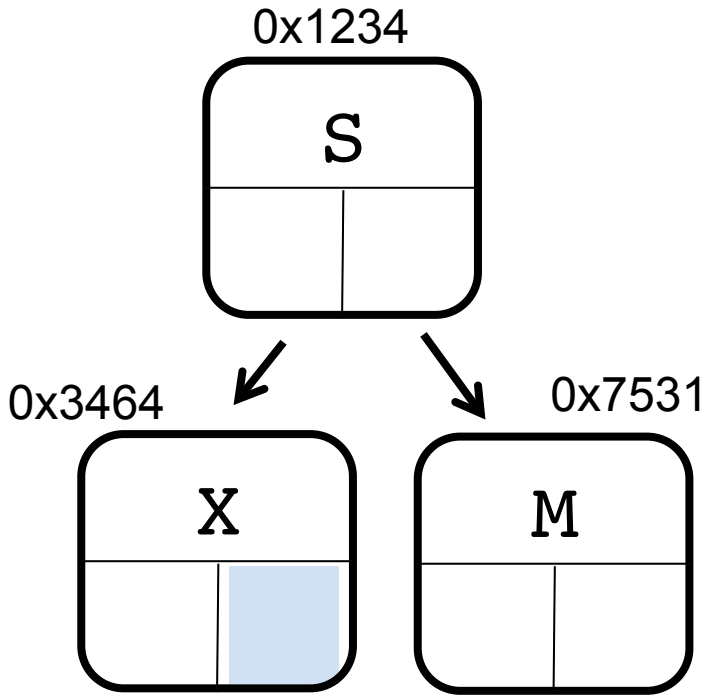
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

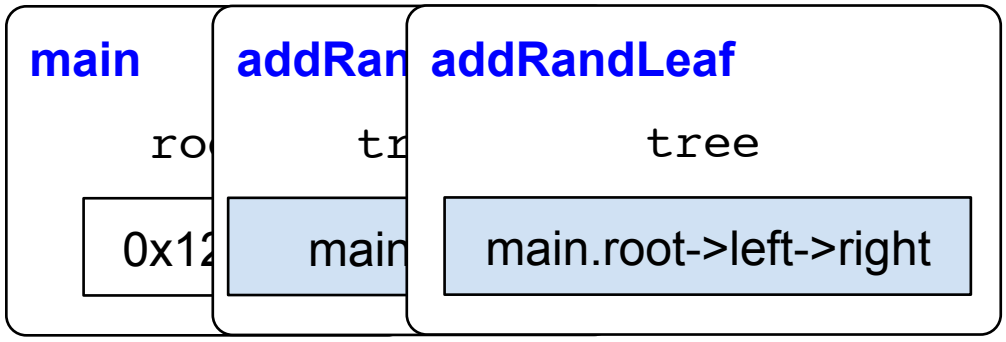




Add Random Leaf

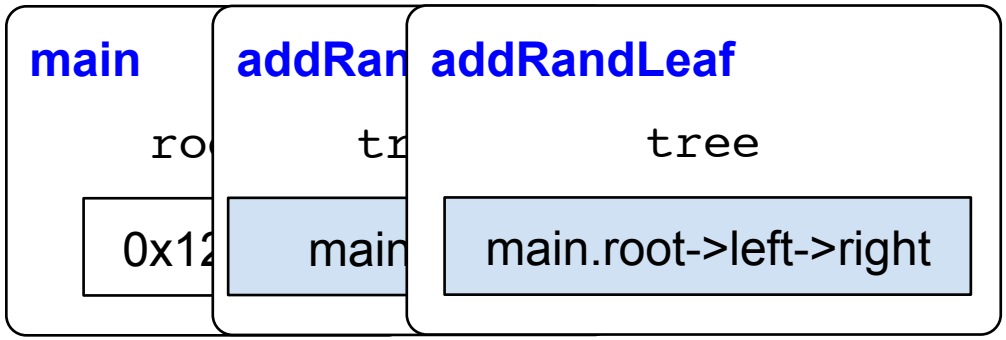
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

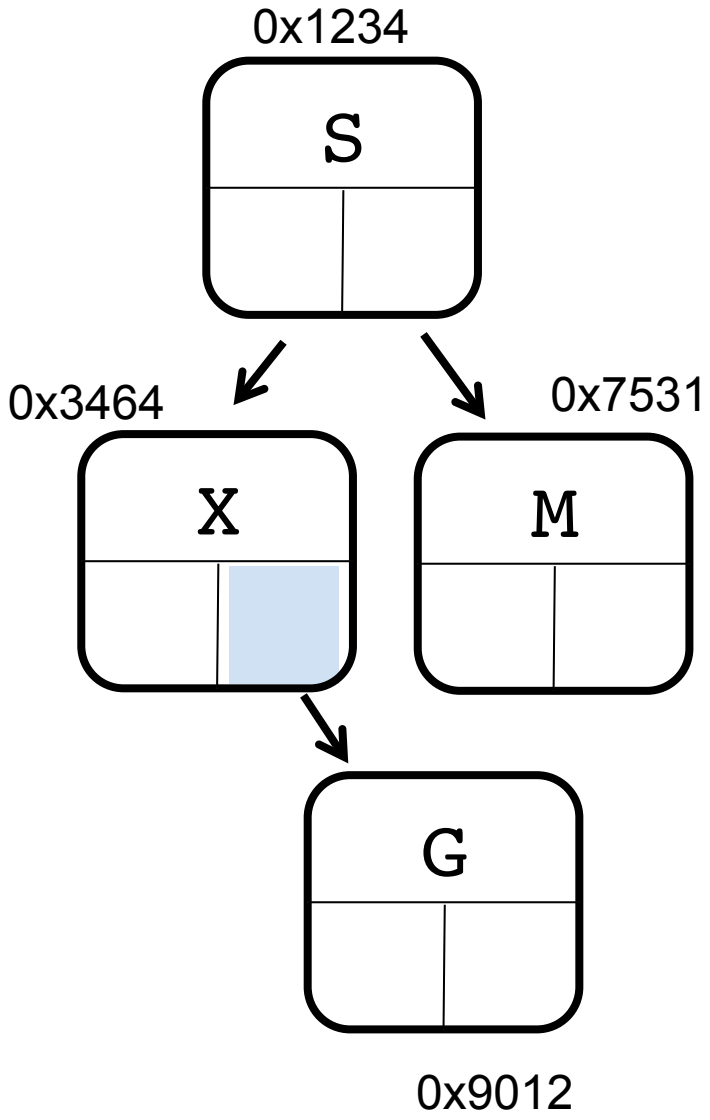




Add Random Leaf

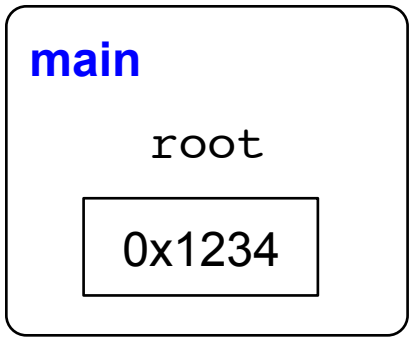
```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

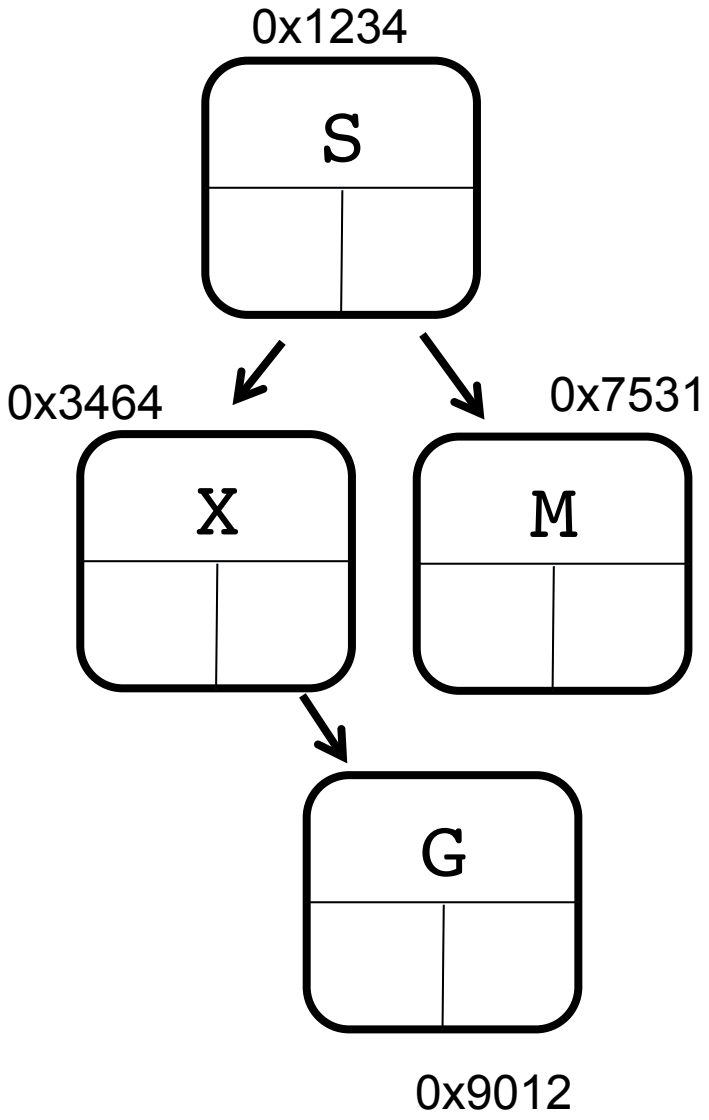




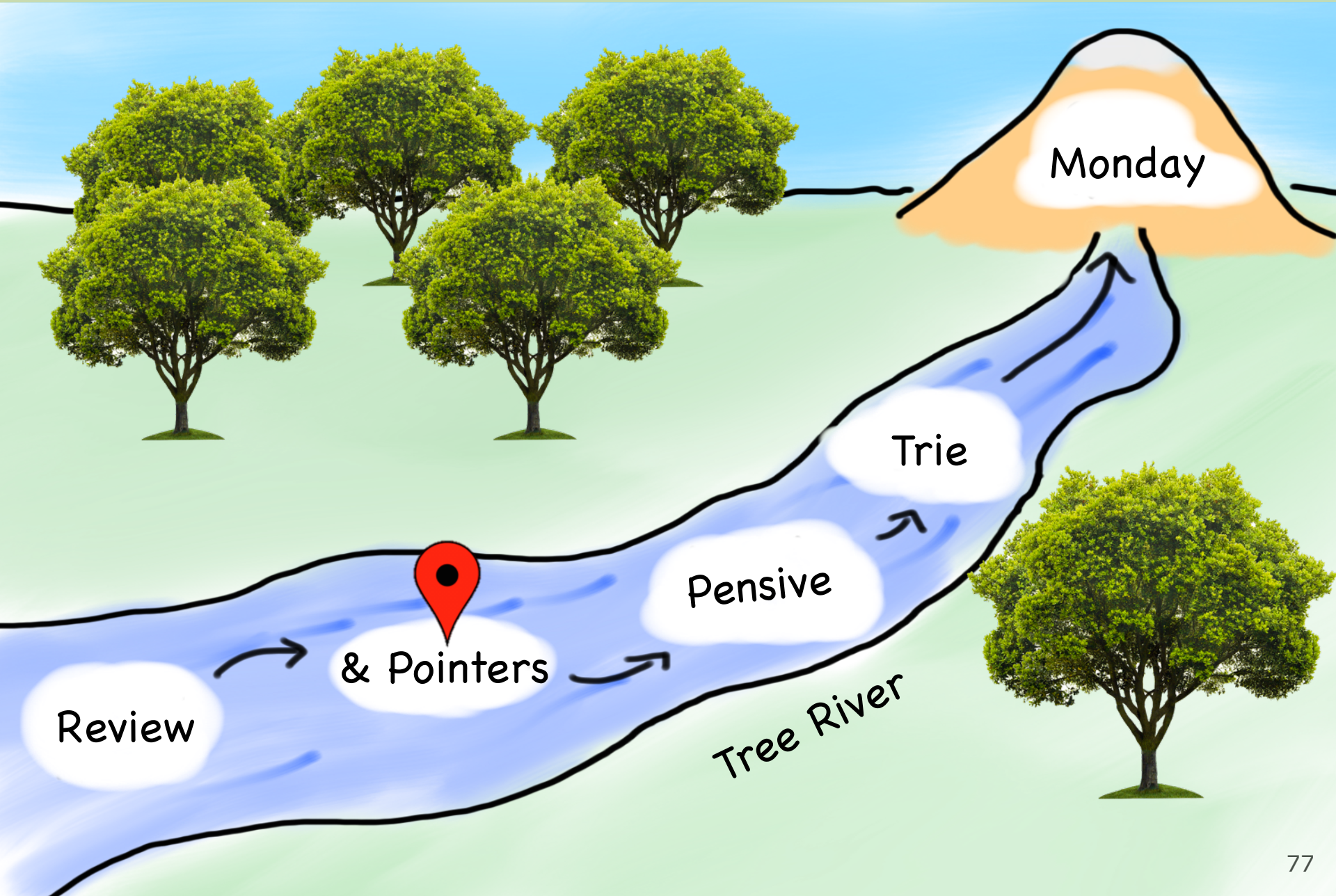
Add Random Leaf

```
void addRandomLeaf(Tree * & tree) {
  if(tree == NULL) {
    tree = new Tree;
    tree->value = randomChar();
    return;
  }
  if(randomBool()) {
    addRandomLeaf(tree->left);
  } else {
    addRandomLeaf(tree->right);
  }
}
```

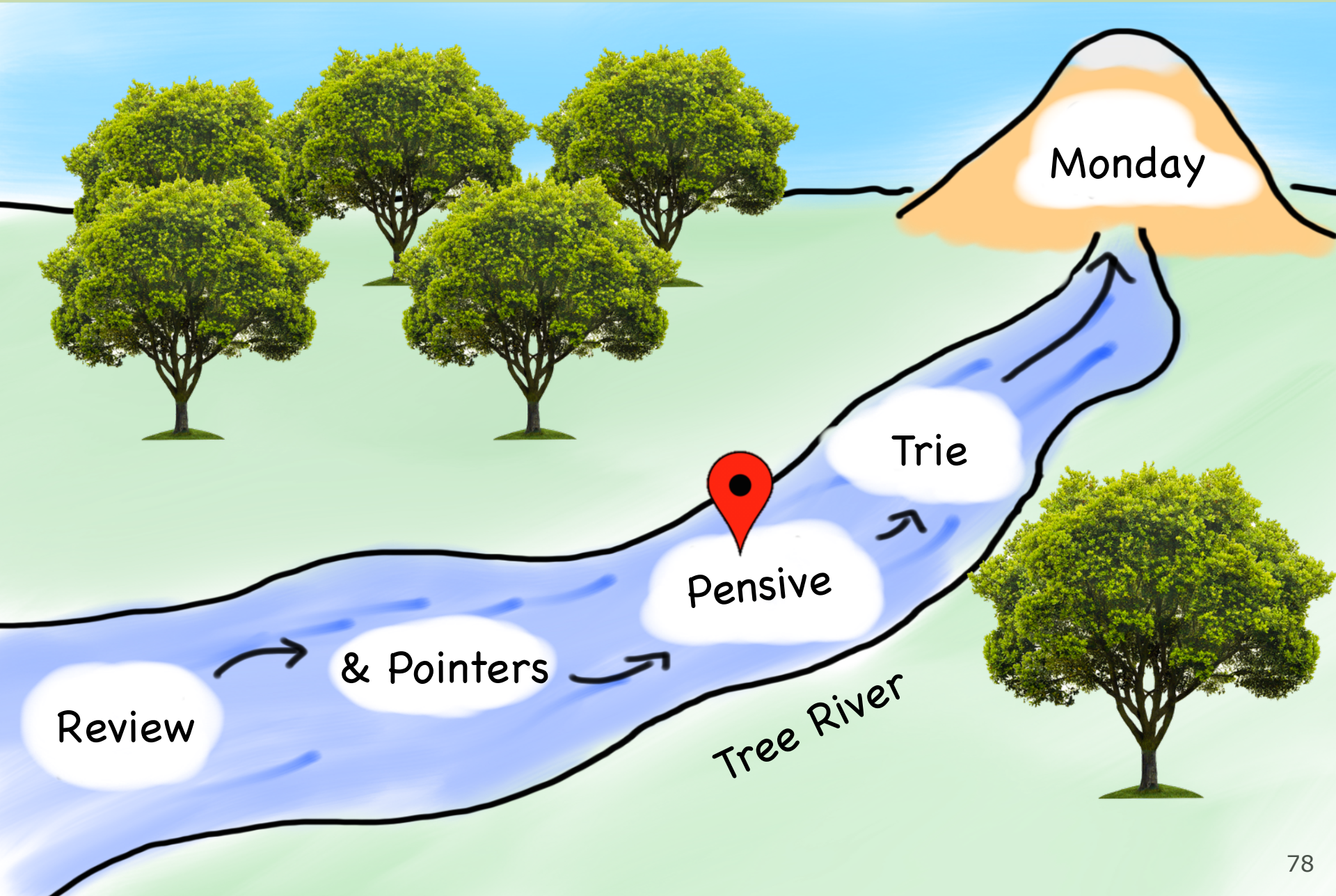


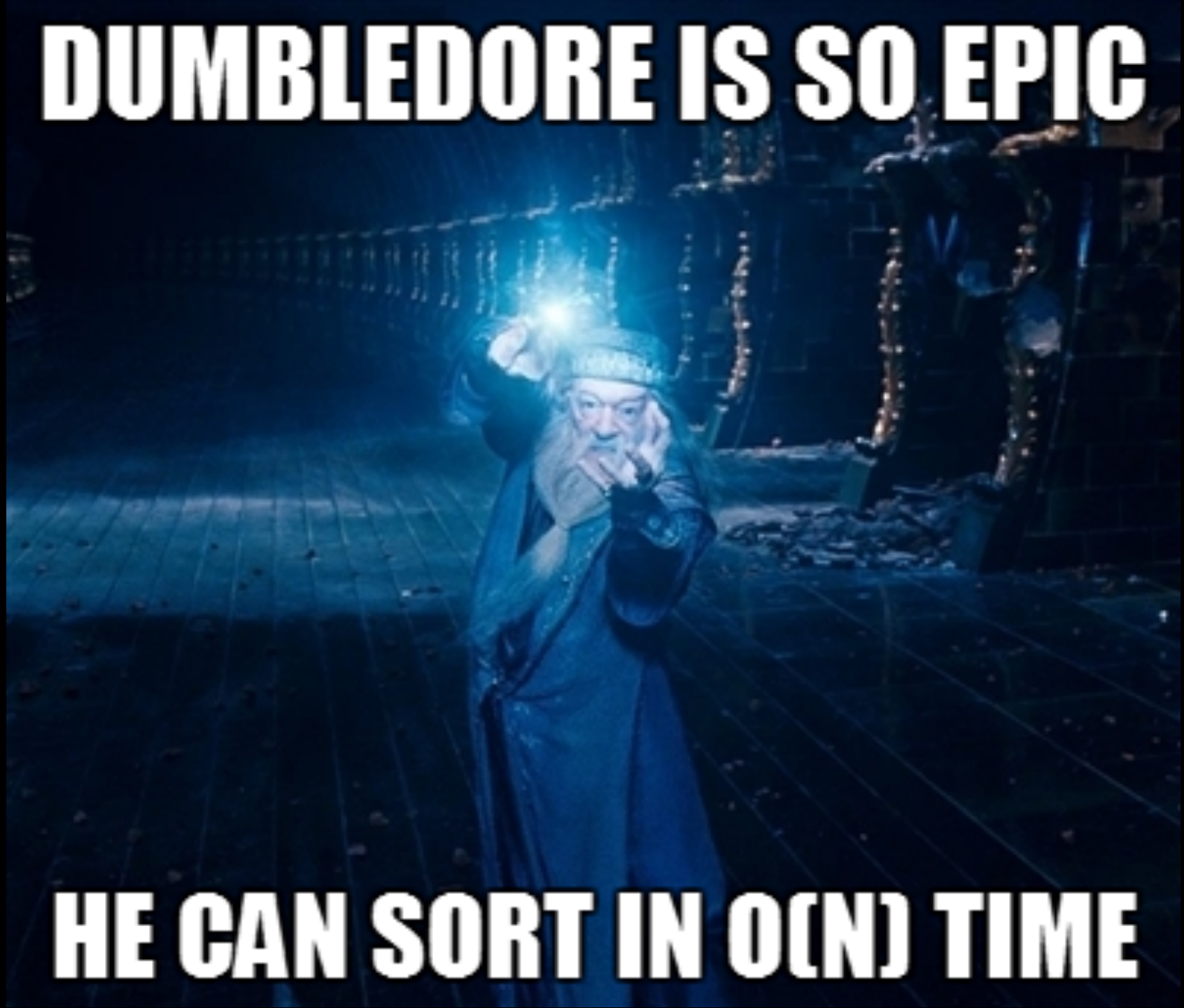


Today's Route

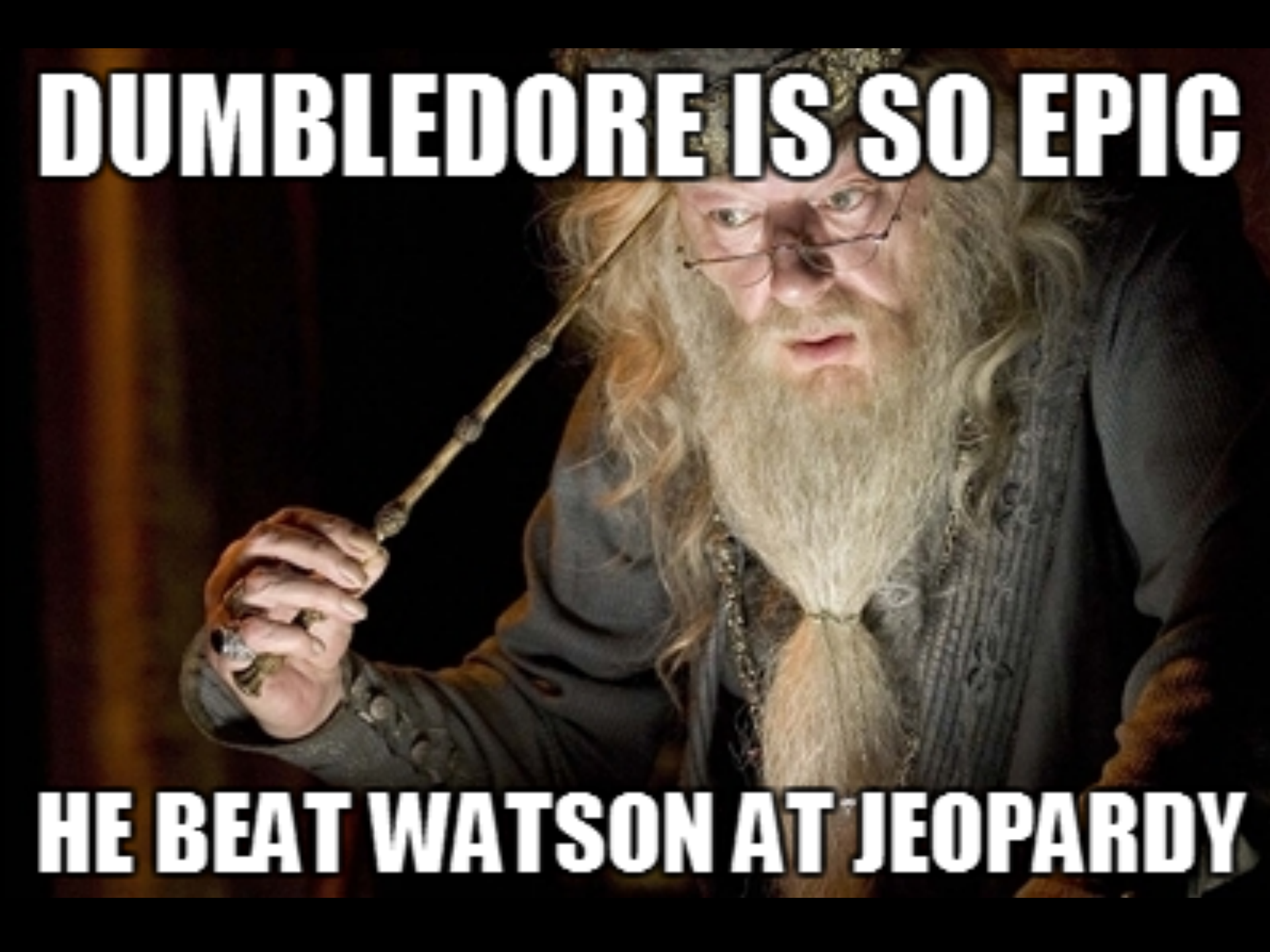


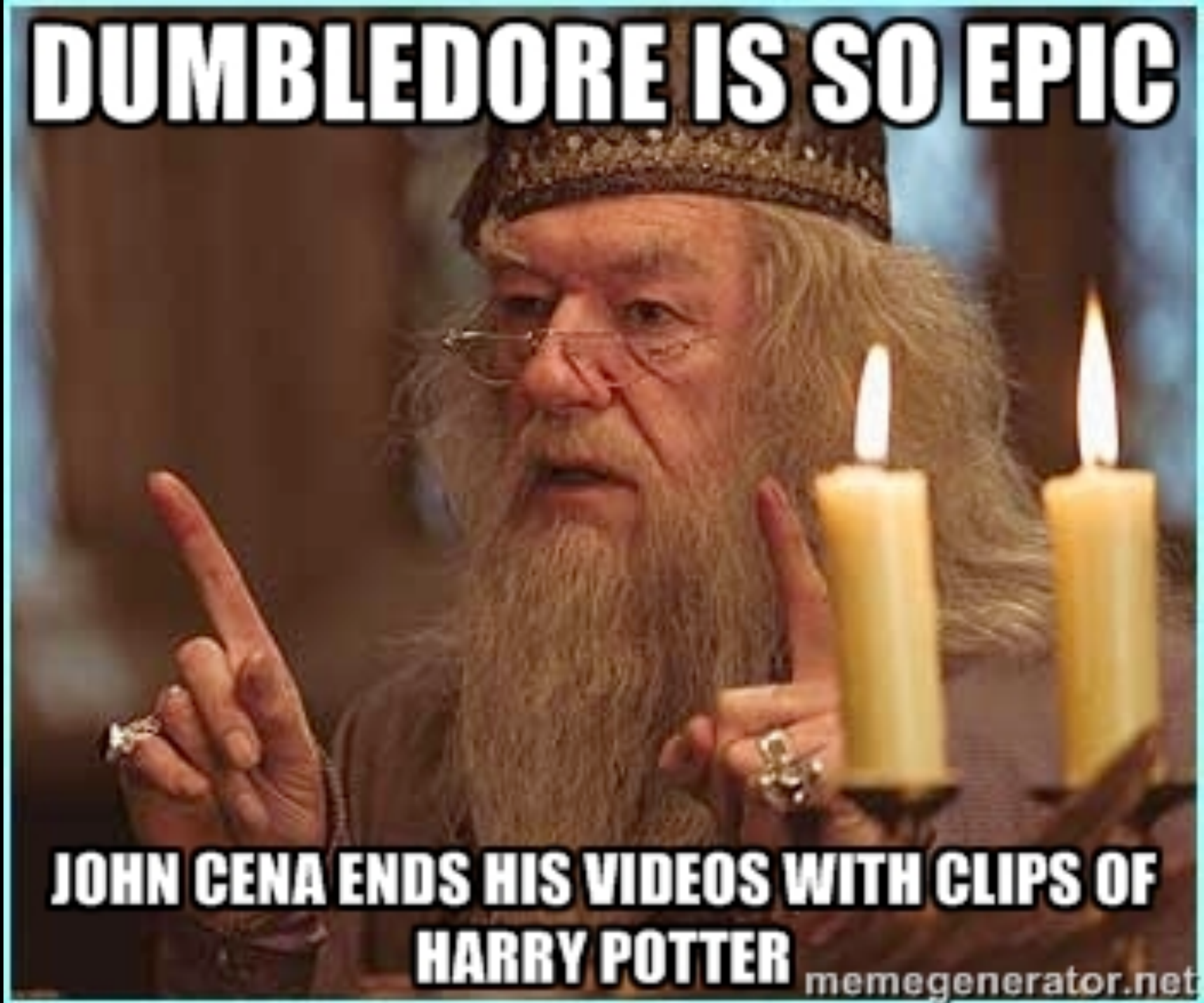
Today's Route

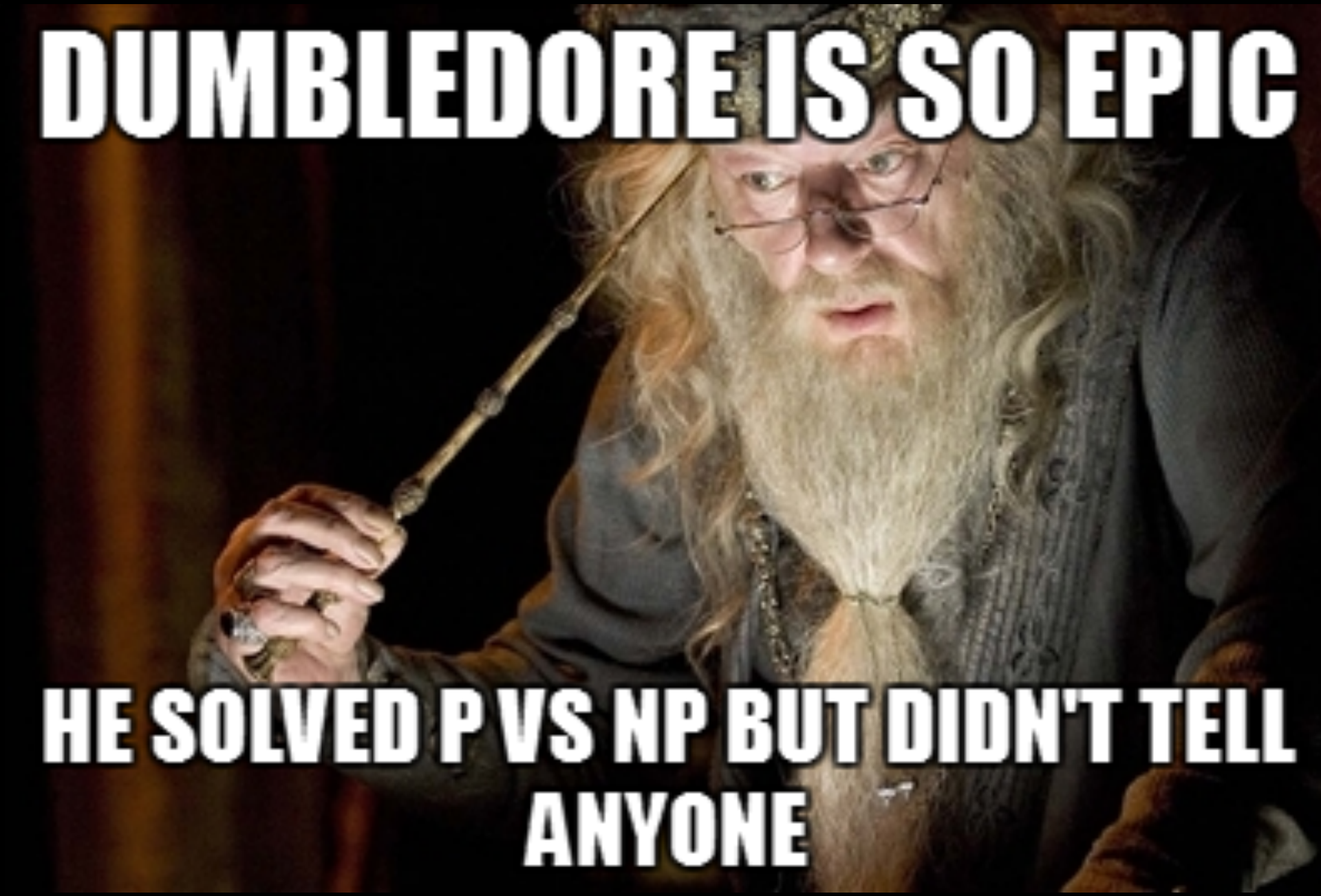



DUMBLEDORE IS SO EPIC
HE CAN SORT IN O(N) TIME


DUBLEDORE IS SO EPIC
HE PROGRAMS IN (C++)++


DUMBLEDORE IS SO EPIC
HE BEAT WATSON AT JEOPARDY


DUMBLEDORE IS SO EPIC
JOHN CENA ENDS HIS VIDEOS WITH CLIPS OF HARRY POTTER
memegenerator.net

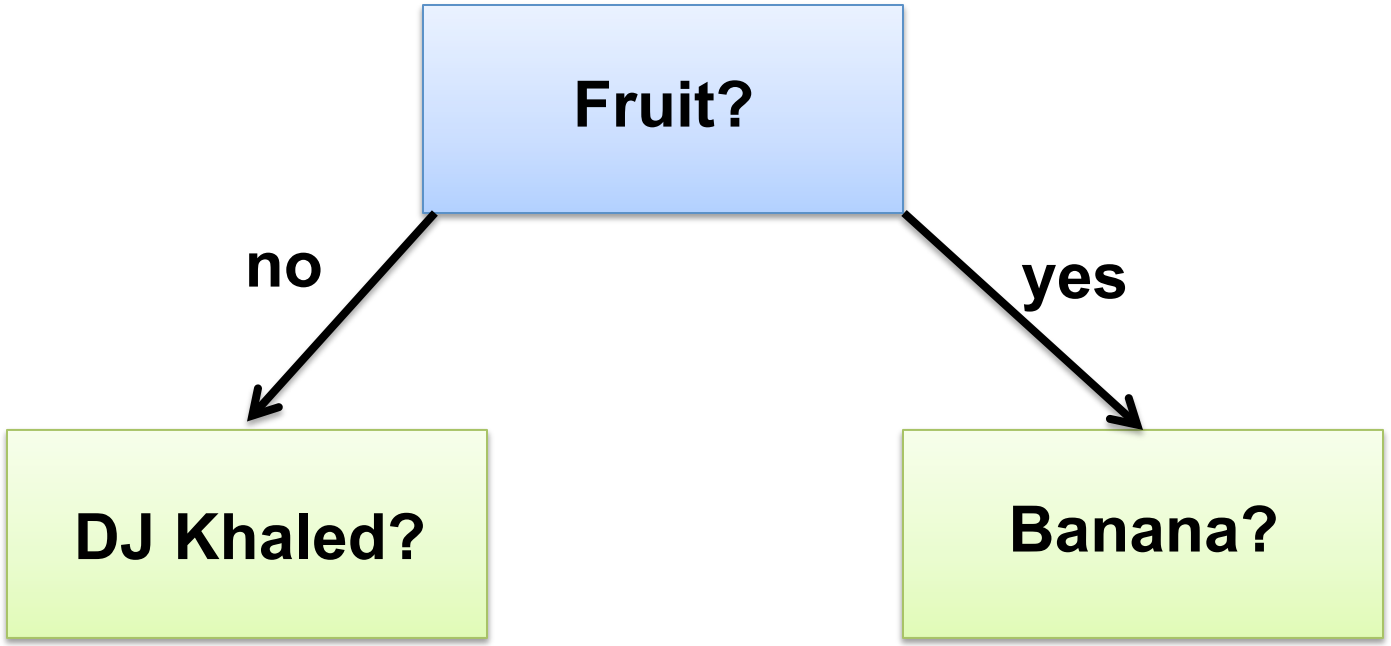

DUMBLEDORE IS SO EPIC
HE SOLVED P VS NP BUT DIDN'T TELL ANYONE

How come Dumbledore knows everything?

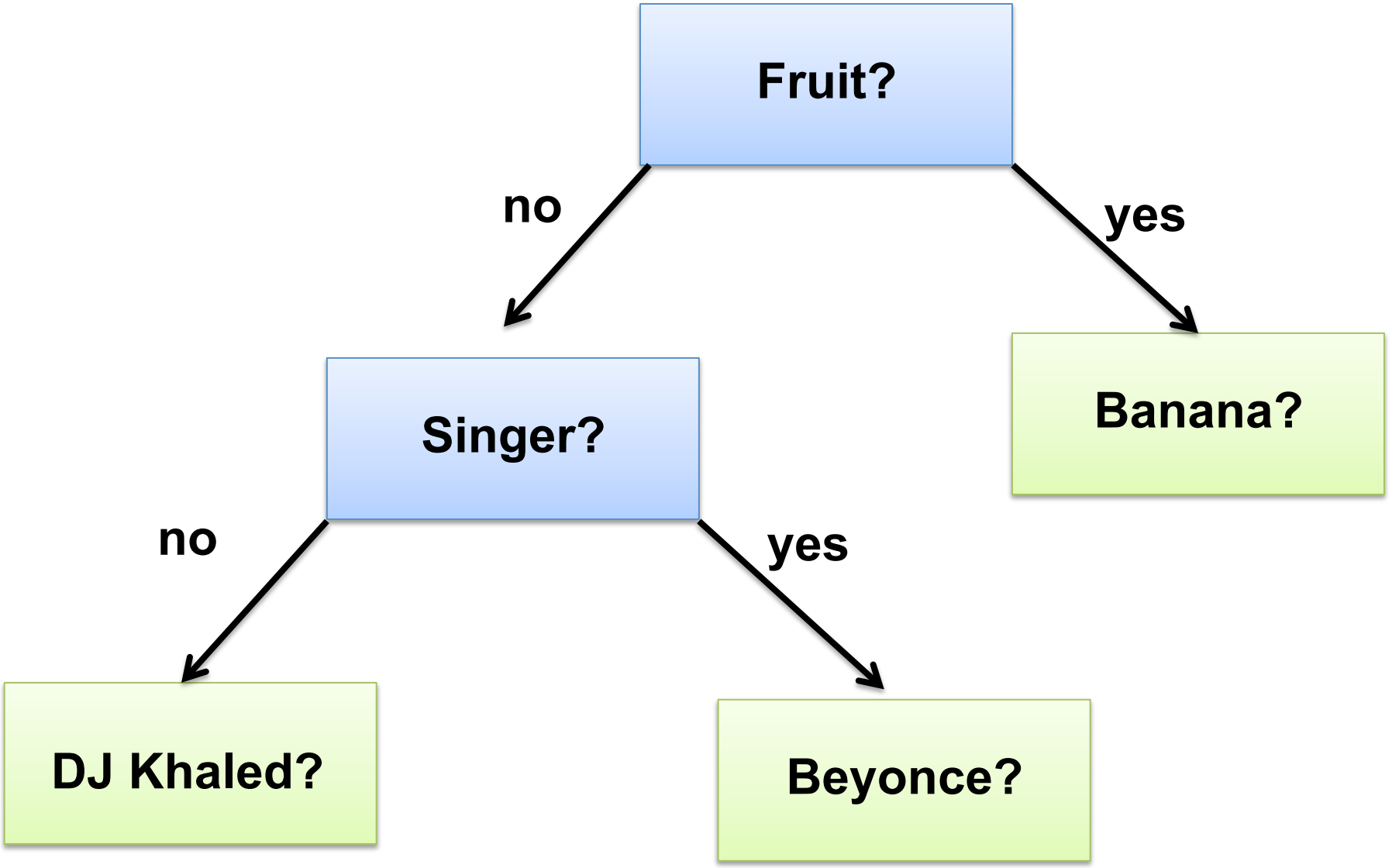
Pensive

Pensive Demo

Pensive

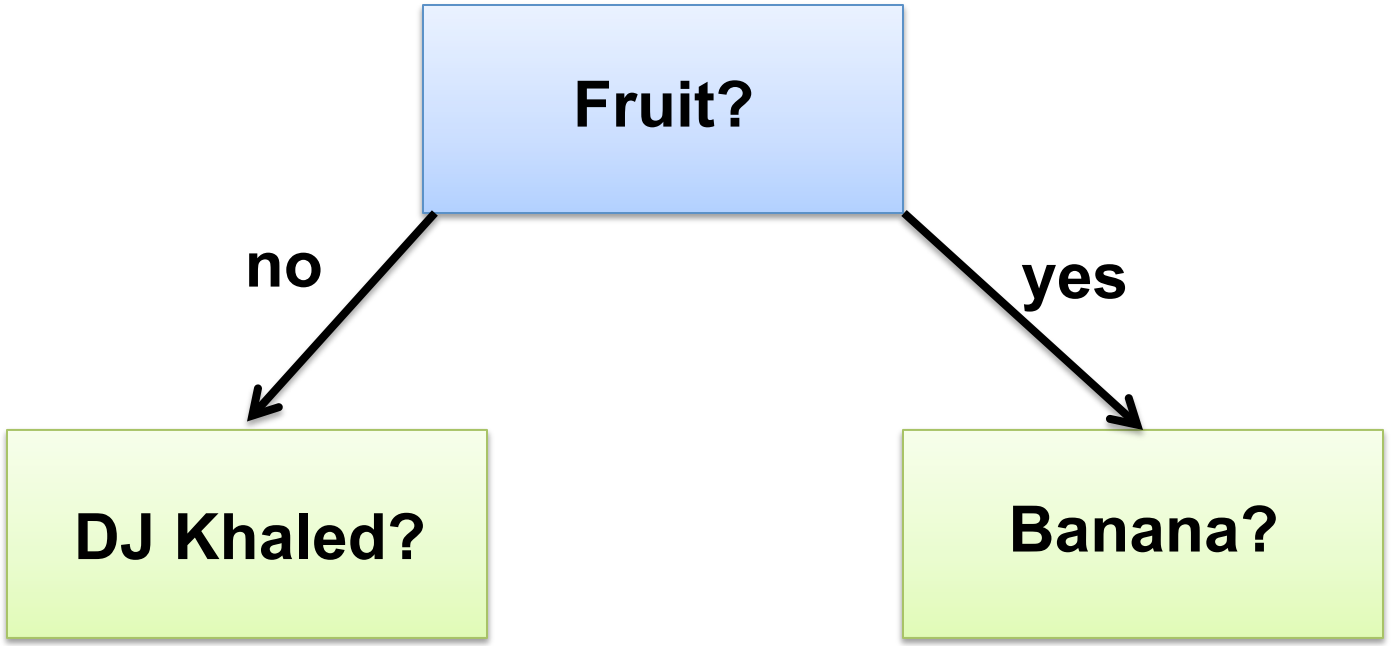


Pensive



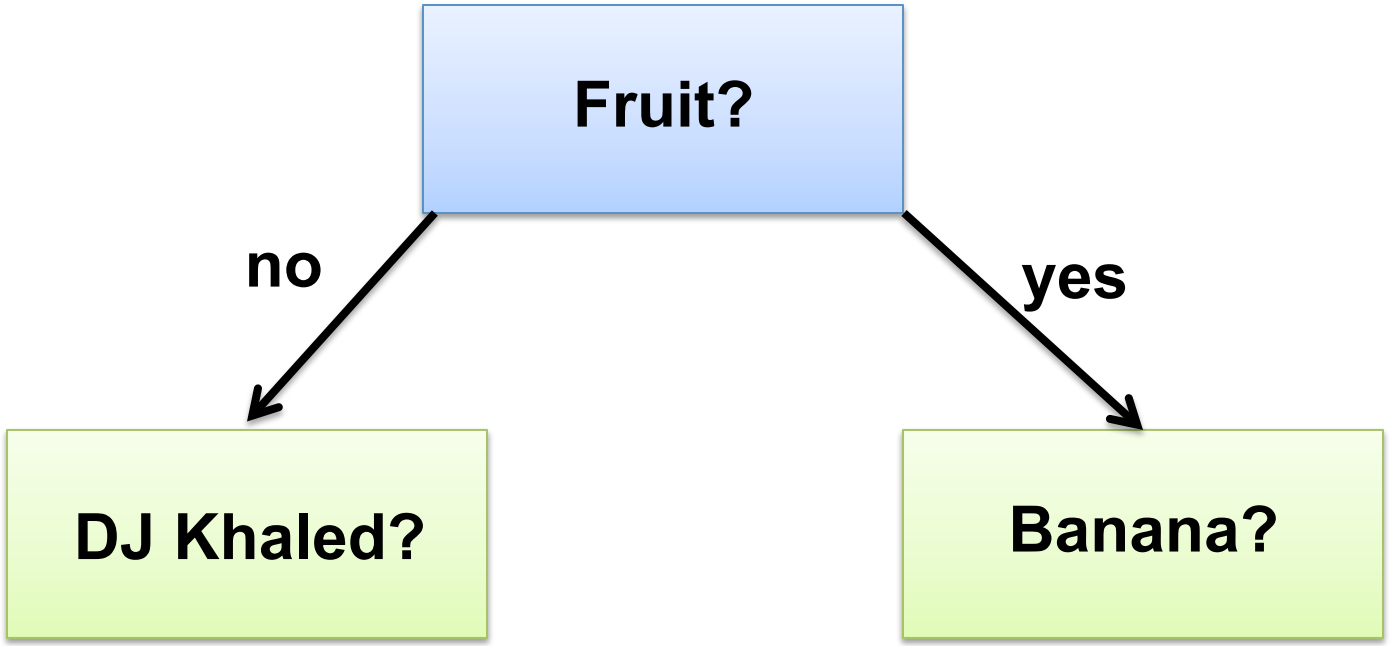
Slowly!

Pensive

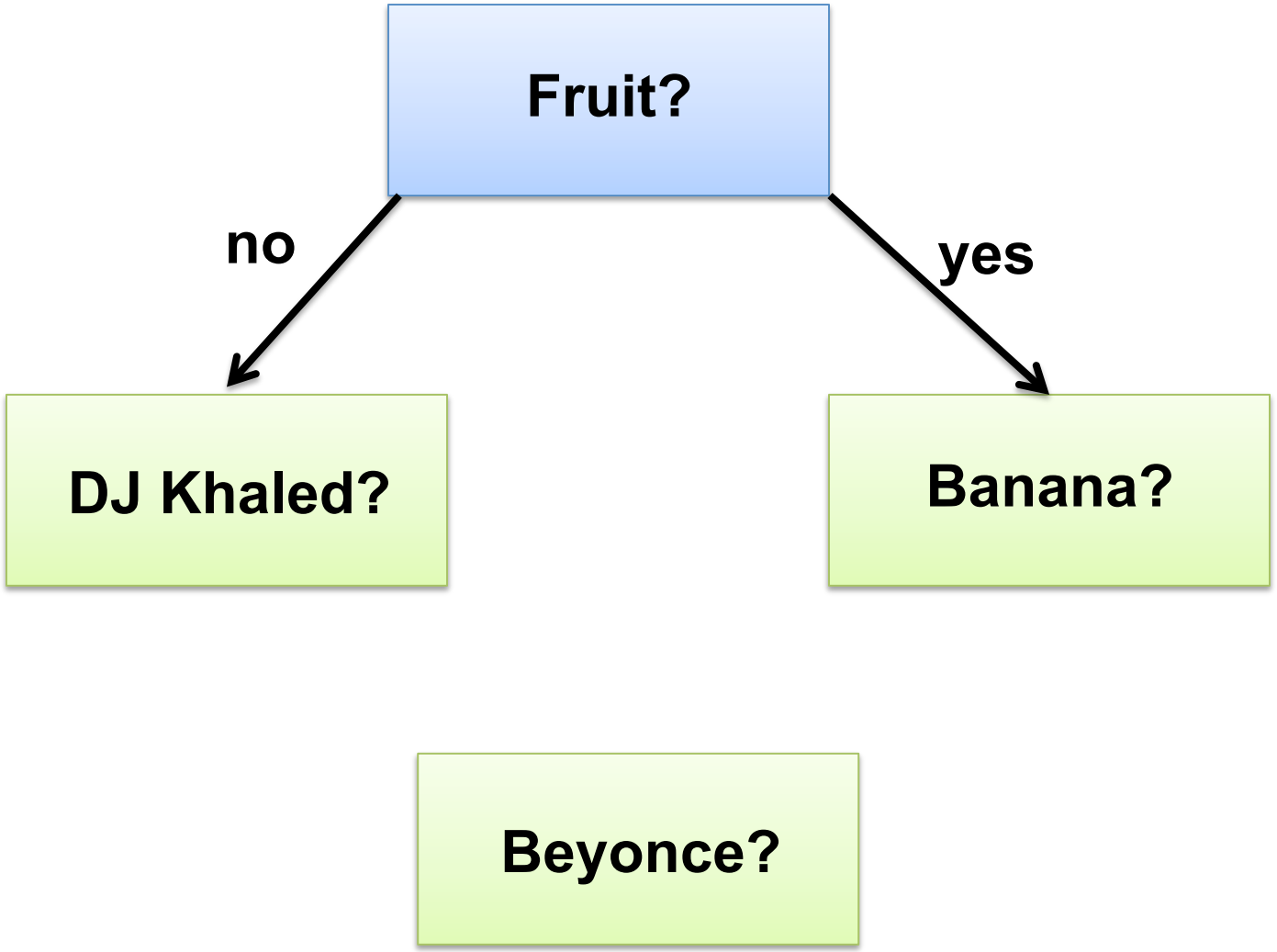



Chiquita
Chiquita
Chiquita
Chiquita
Chiquita

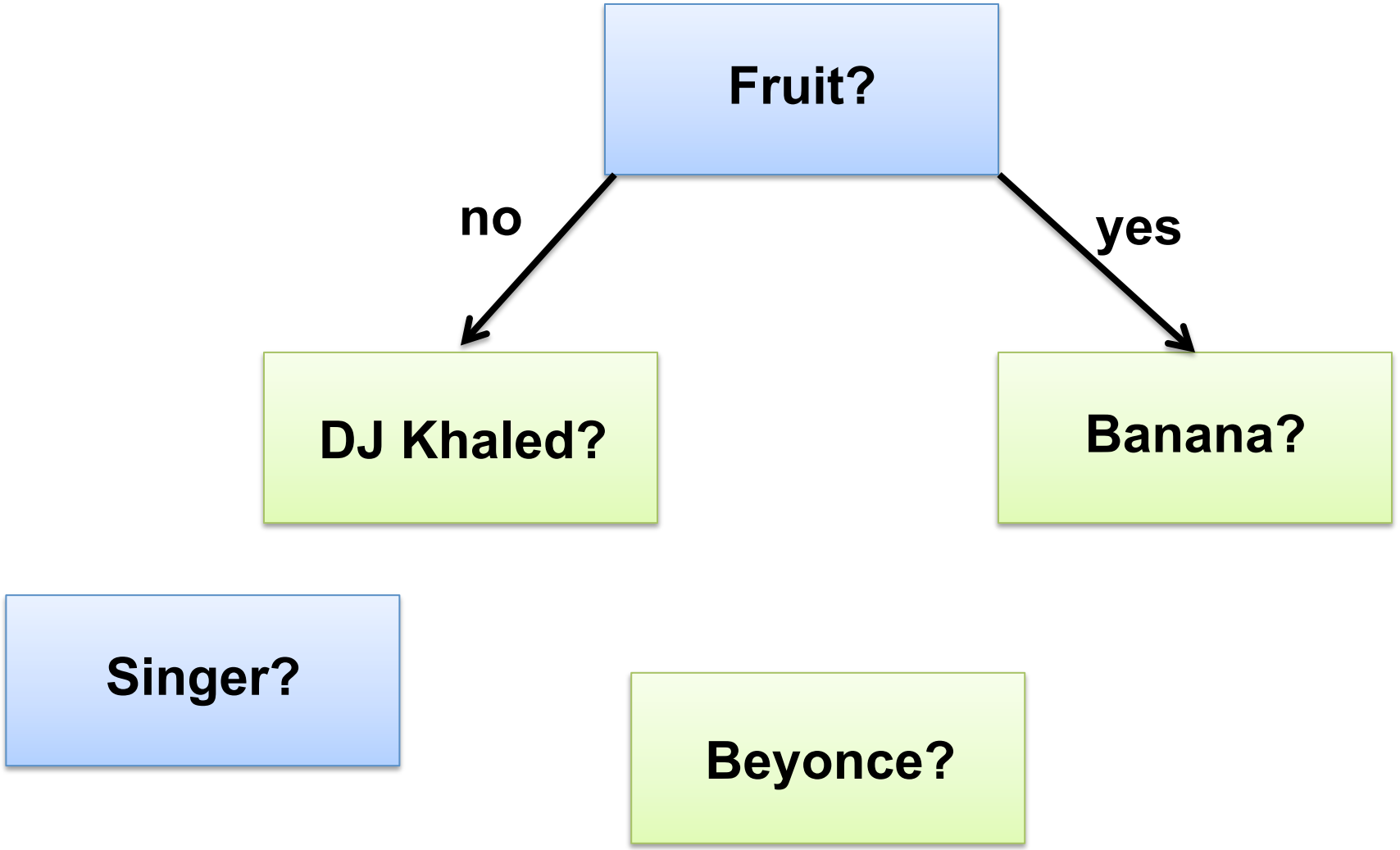
Pensive



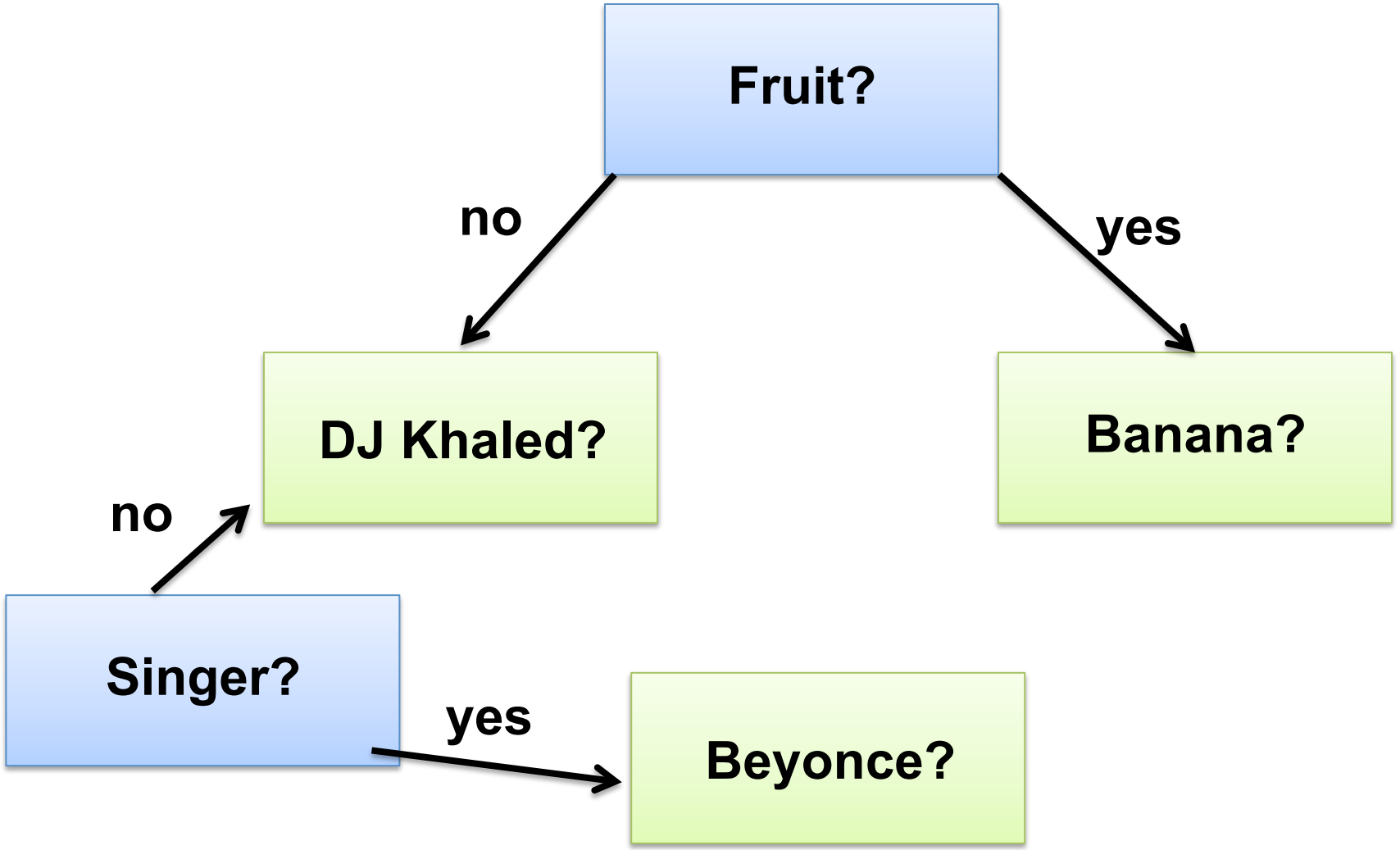
Pensive



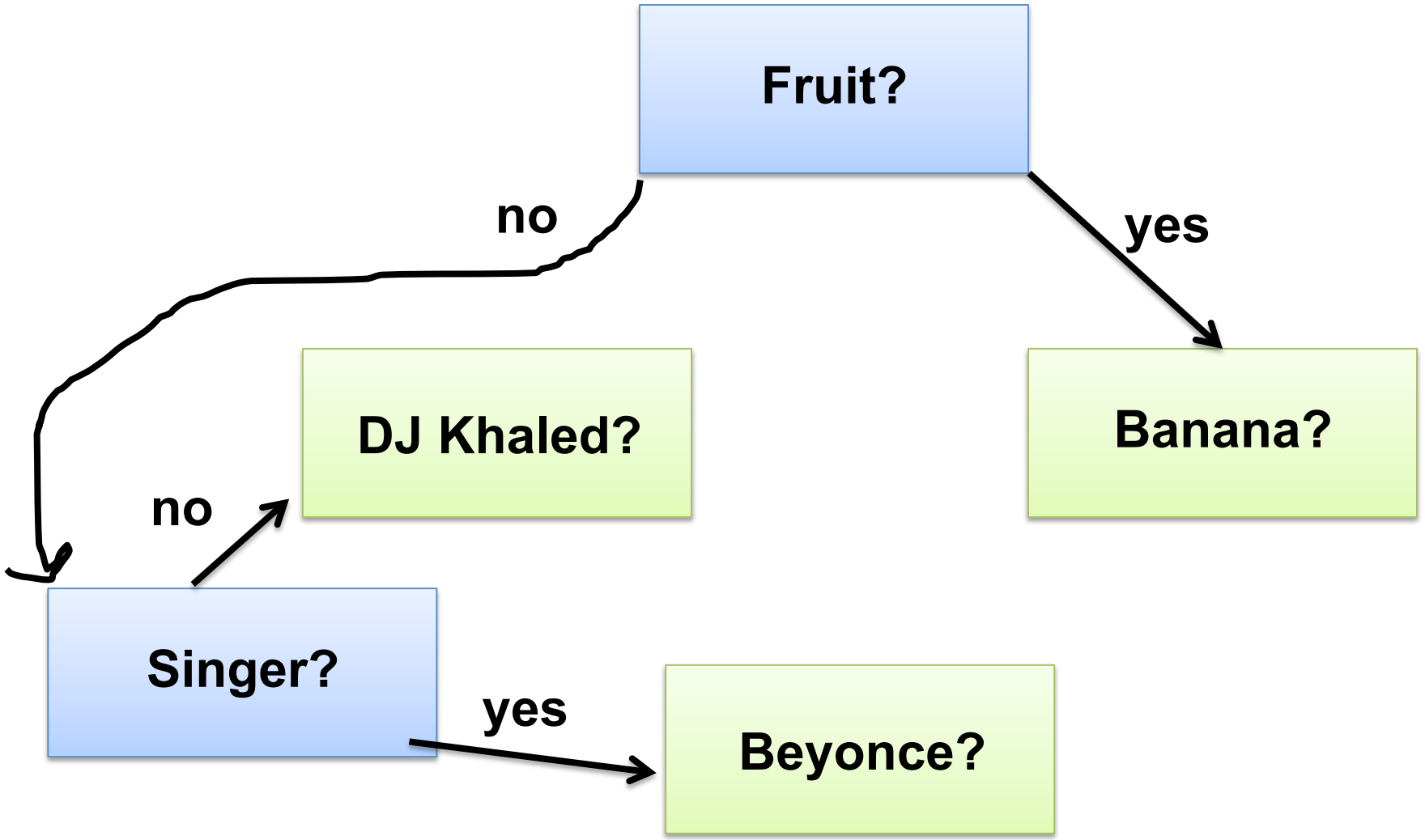
Pensive



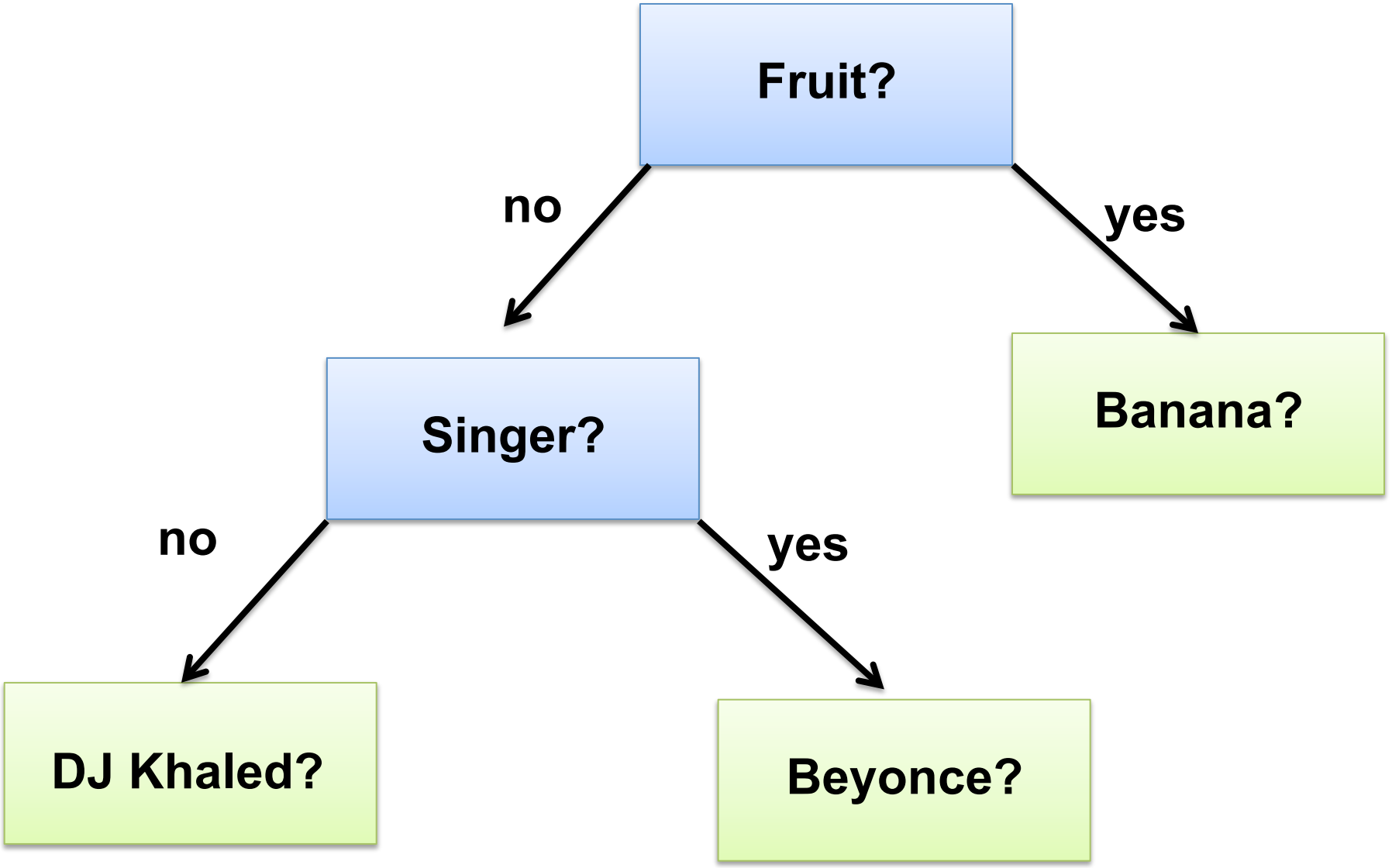
Pensive

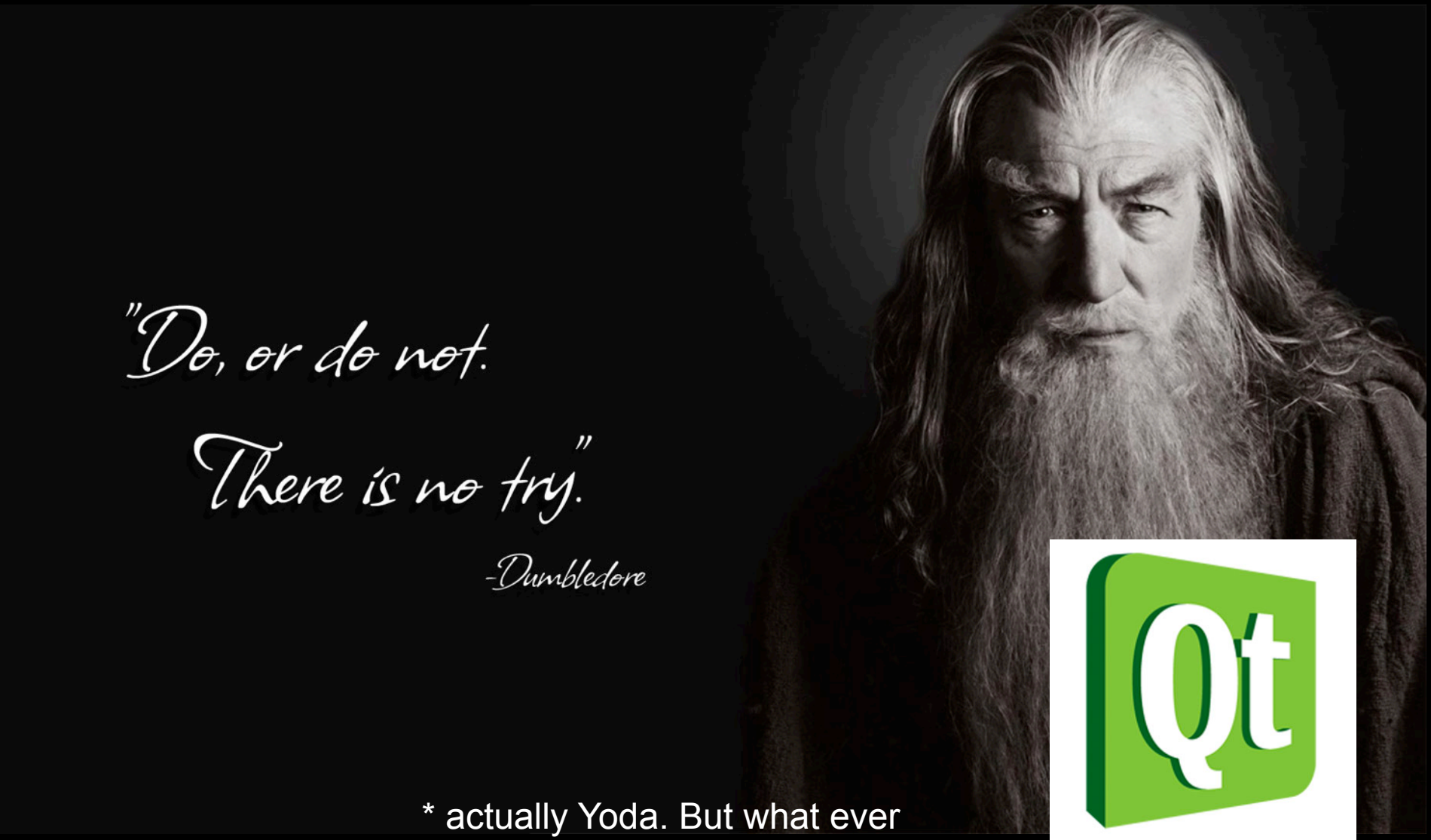


Pensive



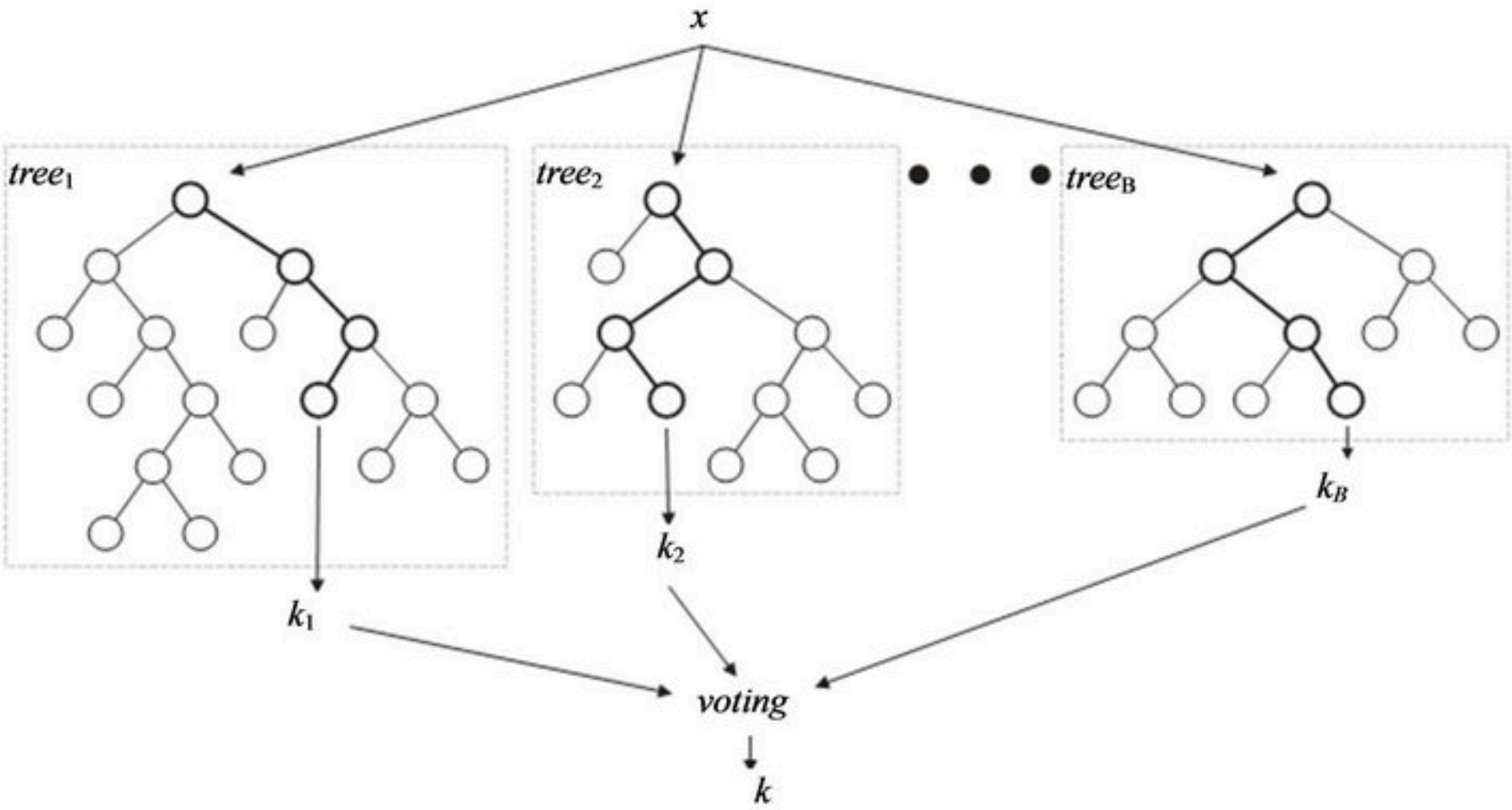
Pensive



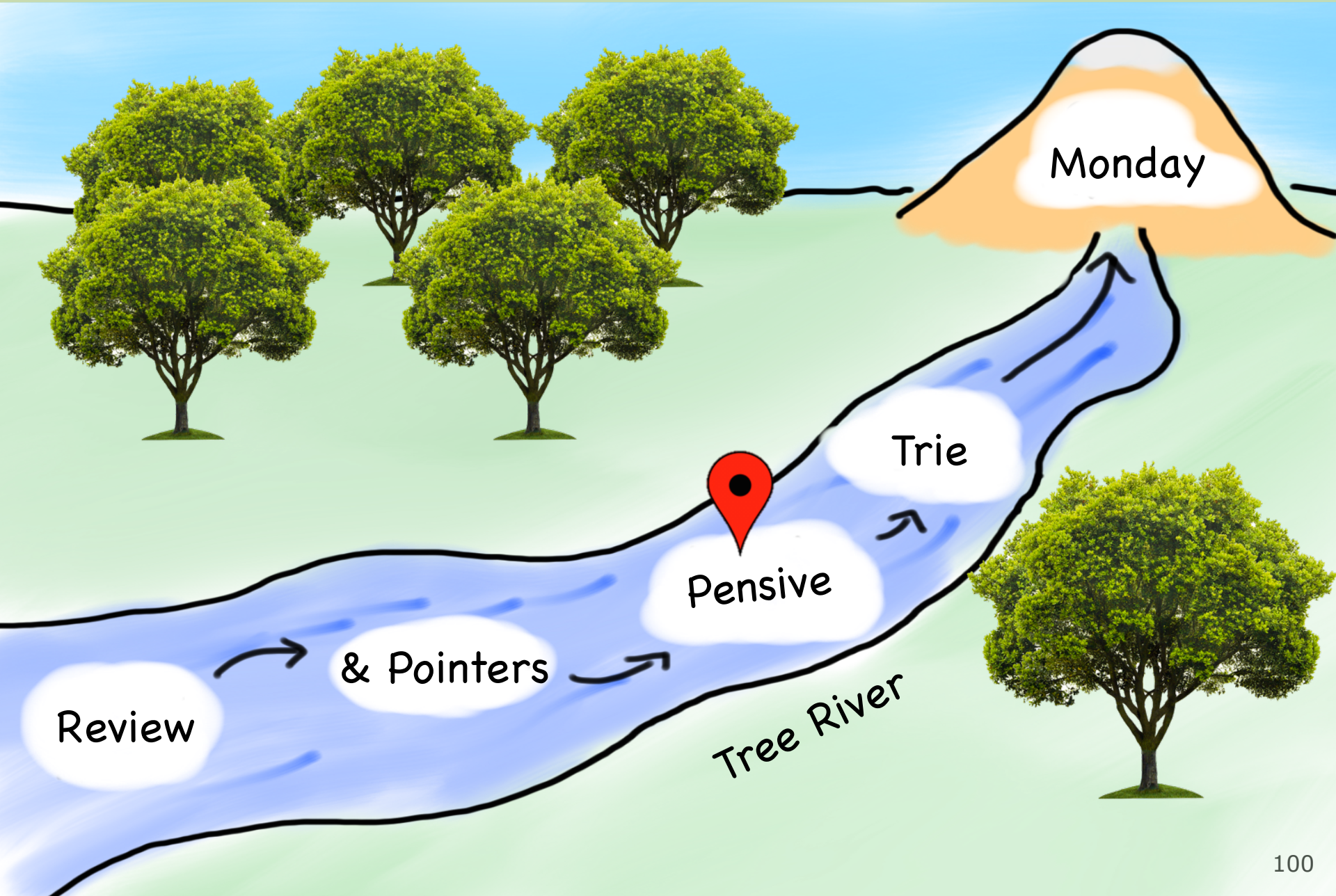


* actually Yoda. But what ever

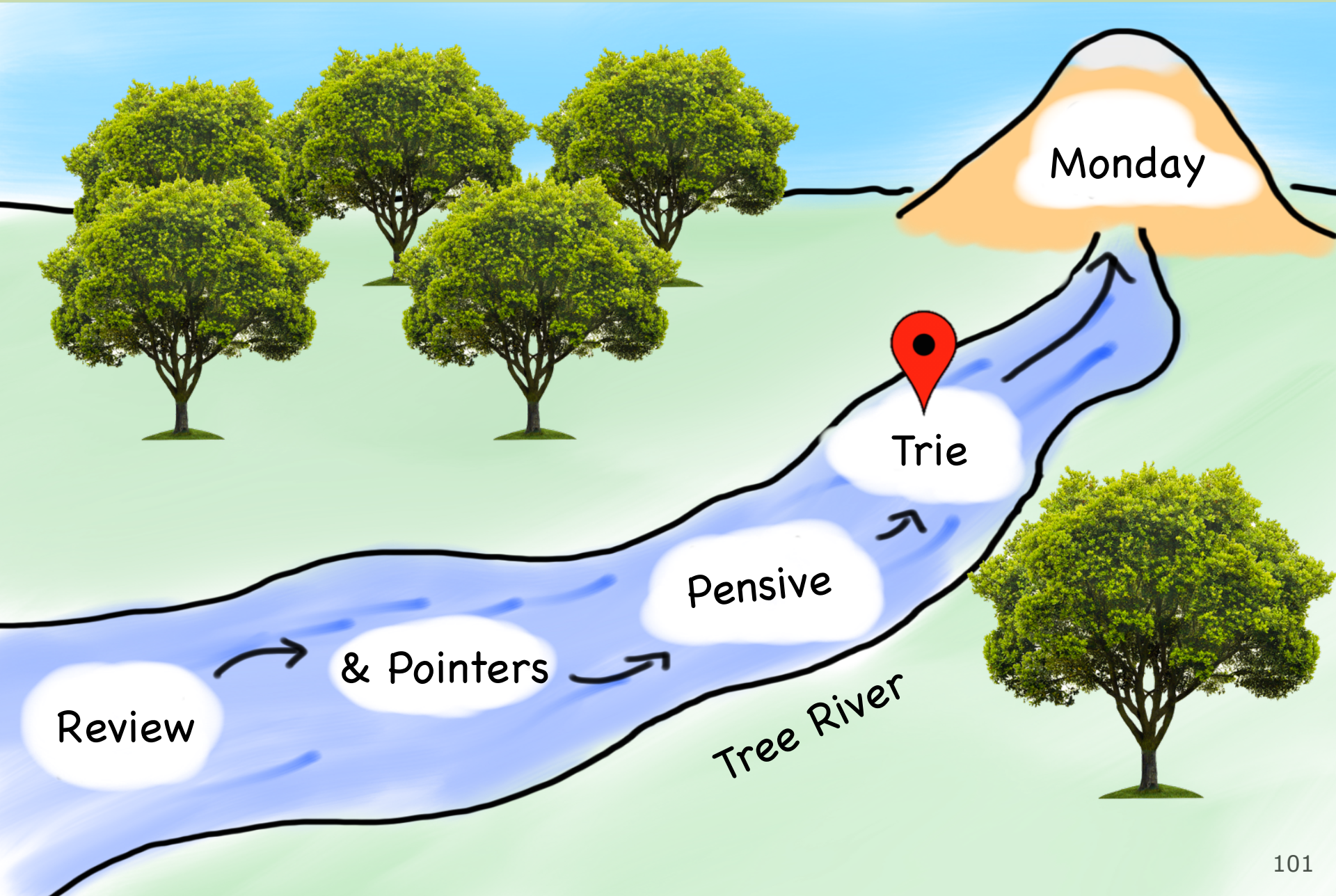
Random Forest



Today's Route



Today's Route

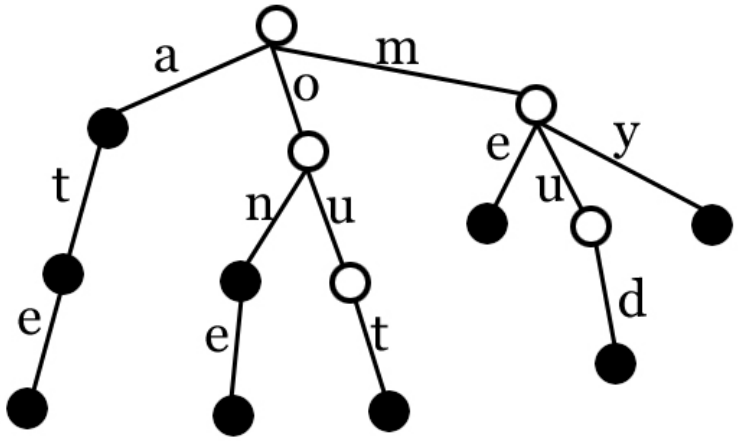


Trie (prefix tree)

trie ("try"): A tree structure optimized for "prefix" searches

e.g. Do any words in the set begin with the prefix `"chr"`?

This is how the Stanford `Lexicon` class is implemented



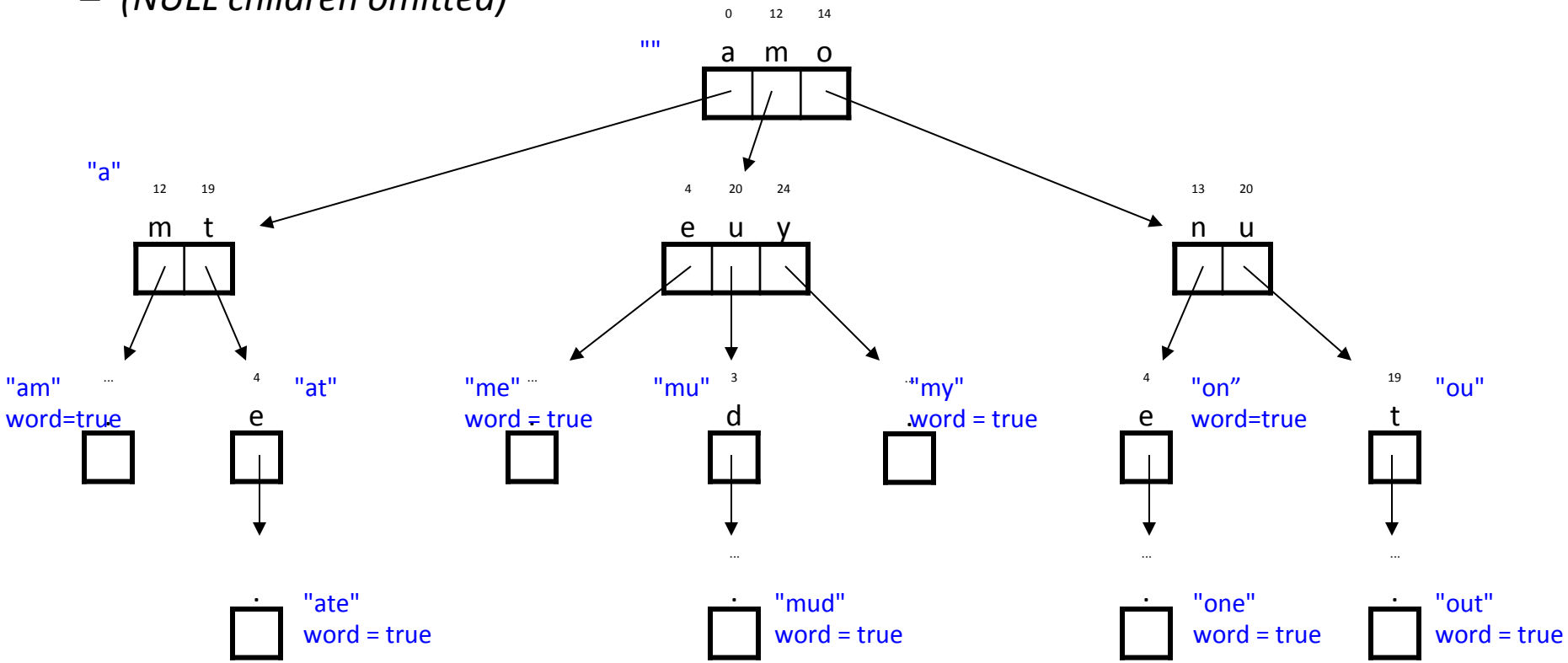
Trie

The idea: instead of a binary tree, use a "26-ary" tree
each node has 26 children for A-Z
add words to the trie by walking
down the appropriate child pointer
(e.g. "ATE" → A, T, E)

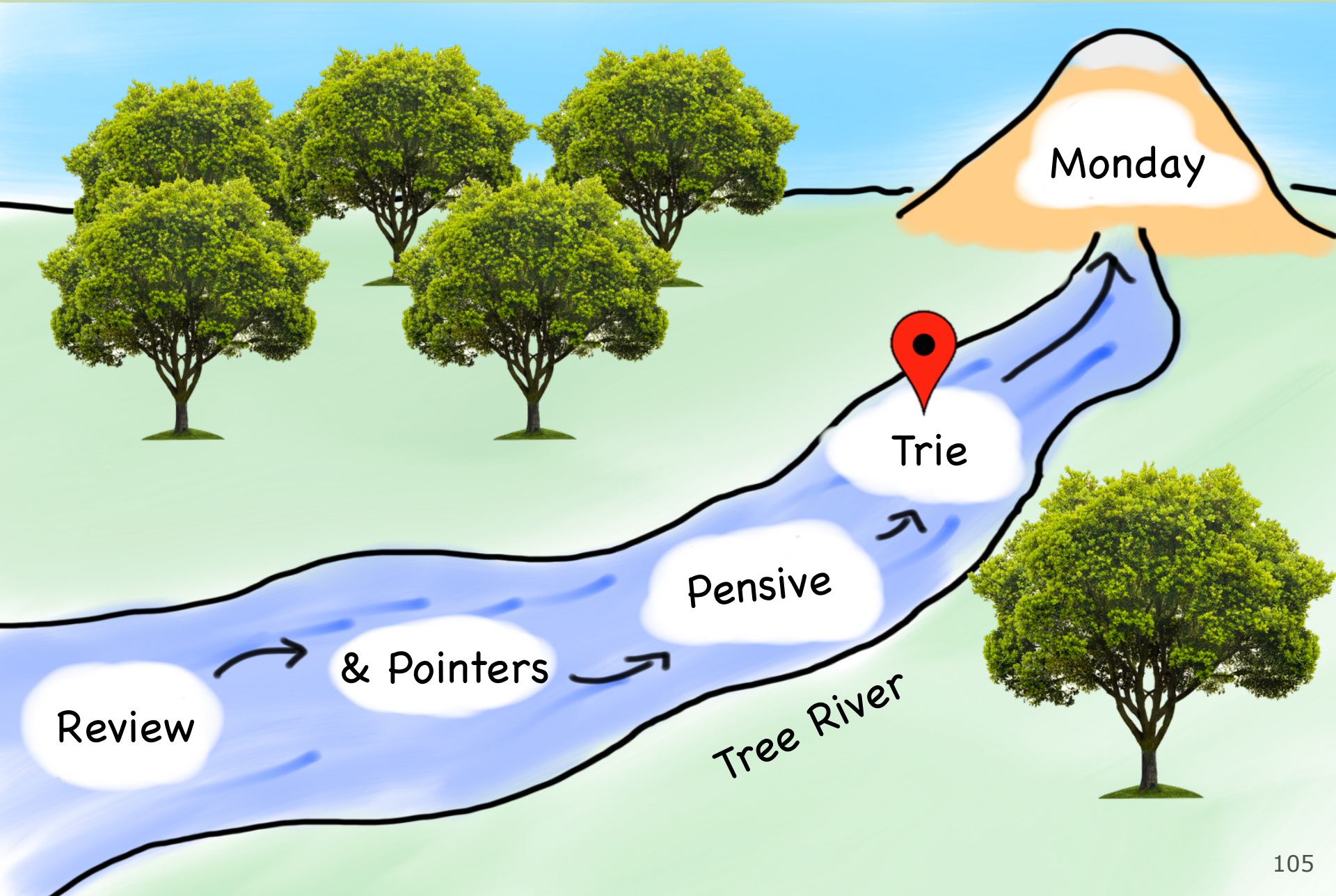
```
struct TrieNode {
    bool word;
    TrieNode* children[26];
}
```

Trie with data

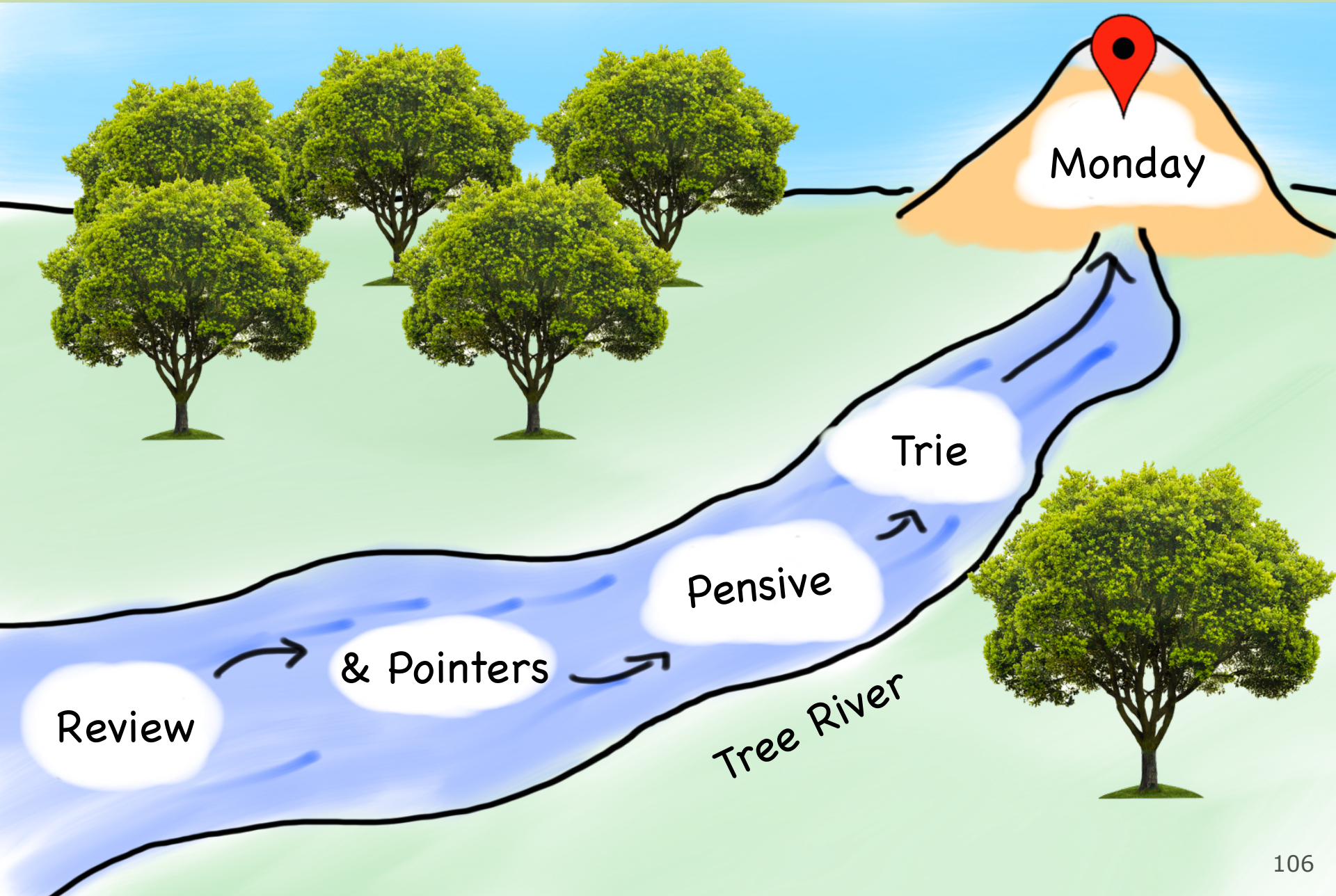
- After adding "am", "ate", "me", "mud", "my", "one", "out":
  - *(NULL children omitted)*



Today's Route



Today's Route



Today's Goal

1. Practice with trees
2. Pointers by reference
3. Be able to insert into a tree