Binary Search Trees

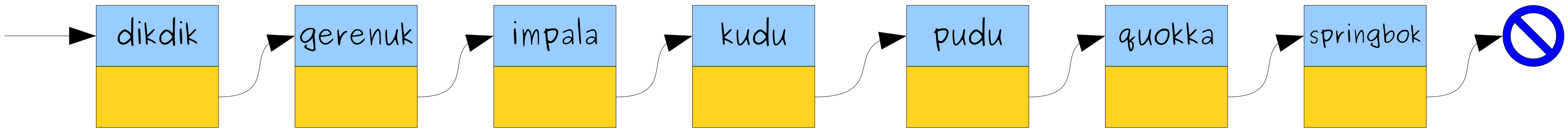
Part One

Way Back When...

Notice that everything is coming back in sorted order.

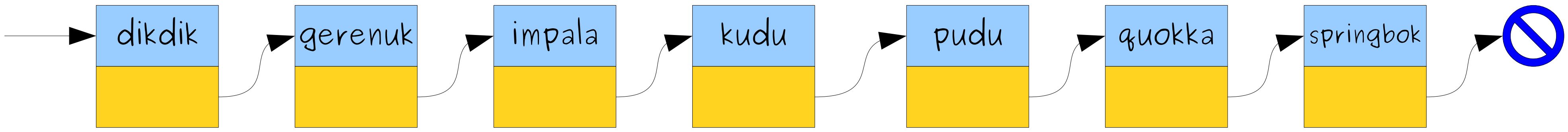
This wouldn’t be the case if we were using hash tables, since they don’t store elements that way.

What’s going on internally?



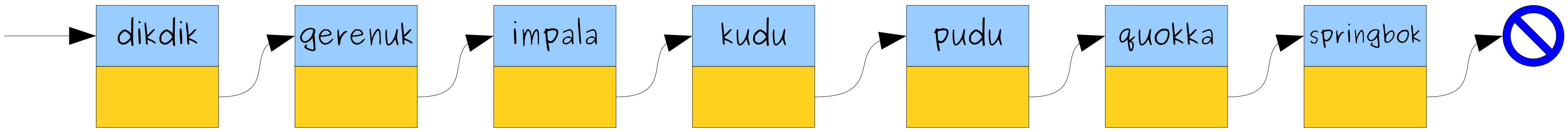
What is the average cost of searching for an element in an *n*-item linked list? Answer using big-O notation.

Formulate a hypothesis, but ***don't post anything in chat just yet***.



What is the average cost of searching for an element in an *n*-item linked list? Answer using big-O notation.

Now, ***post your best guess in chat***. Not sure? Answer with "??"



Answer: **O(*n*)**.

Intuition: Most elements are far from the front.

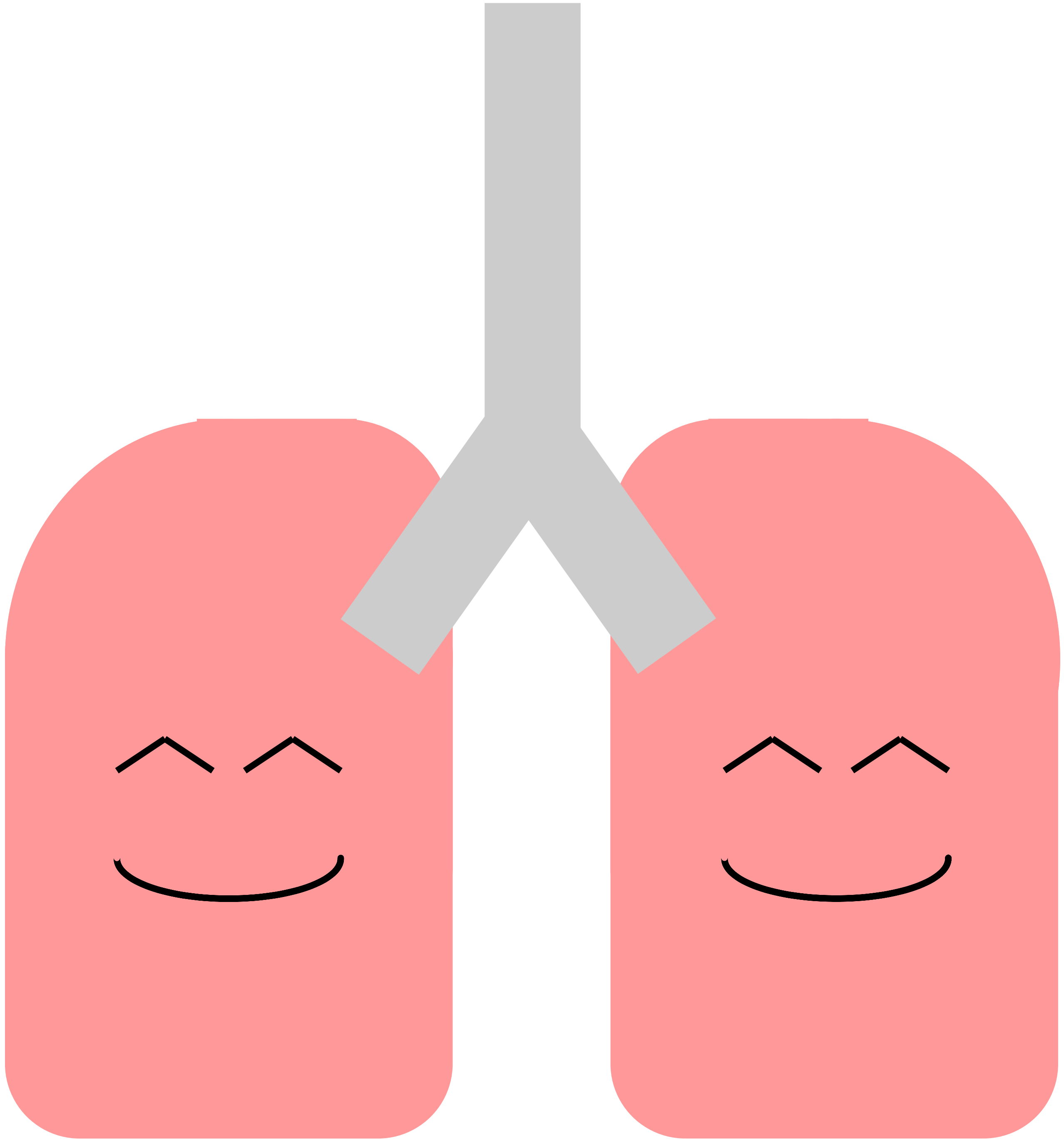
Can you chain a bunch of objects together so that most of them are near the front?

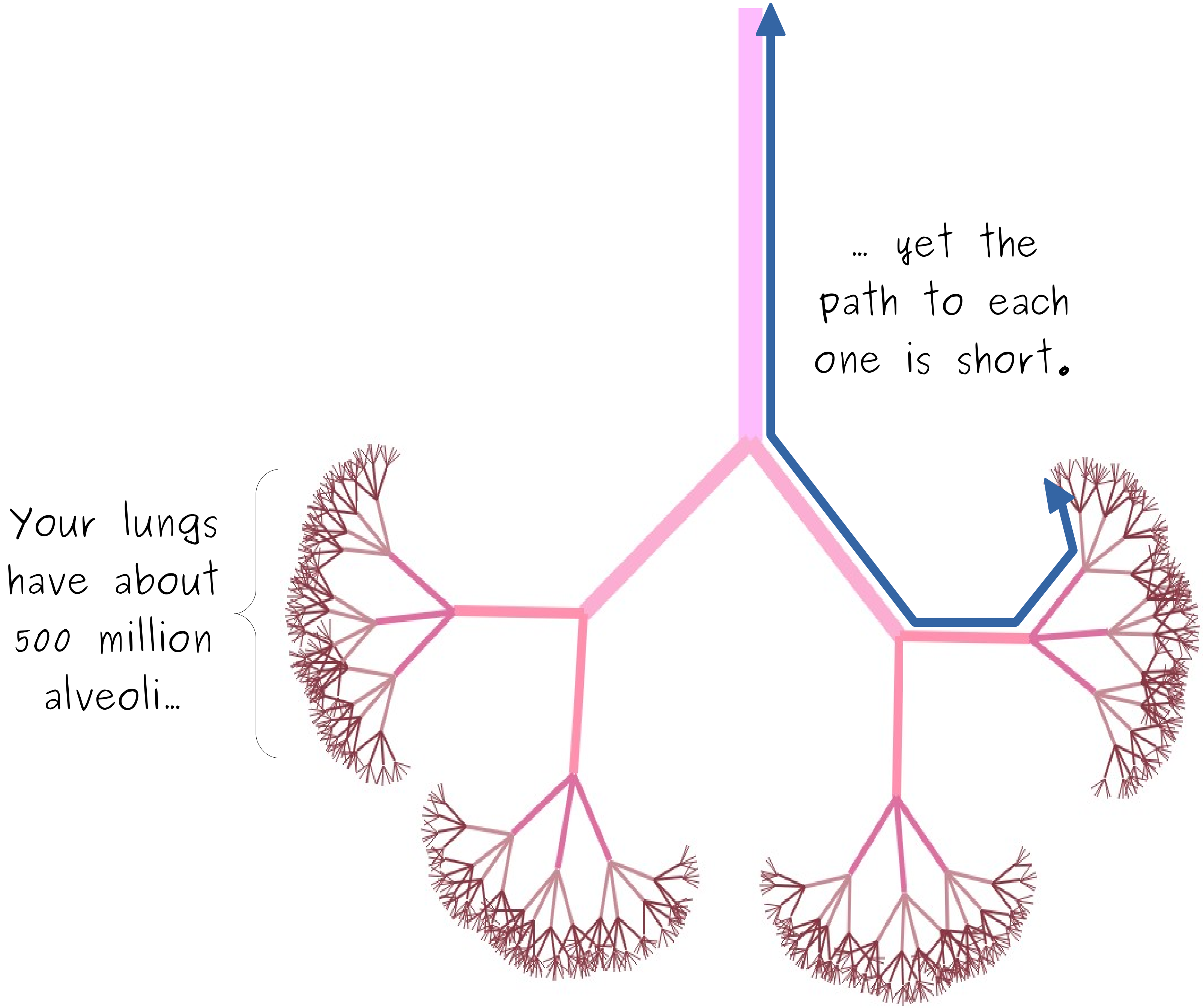
An Interactive Analogy

Take a deep breath.

And exhale.

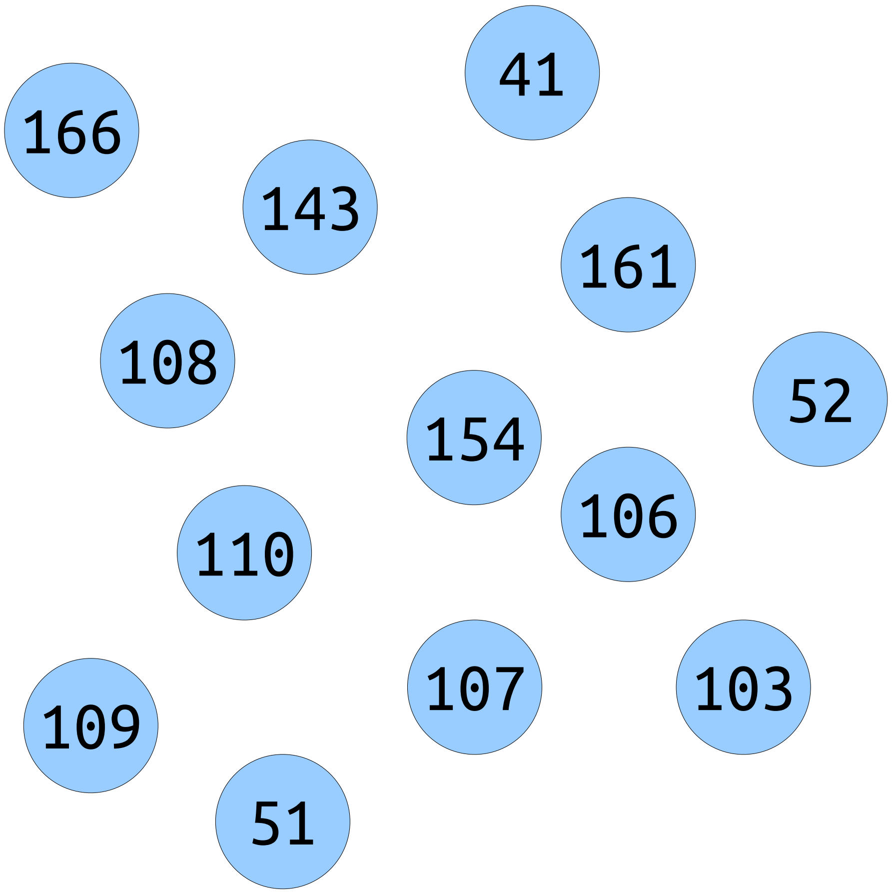
Feel nicely oxygenated?



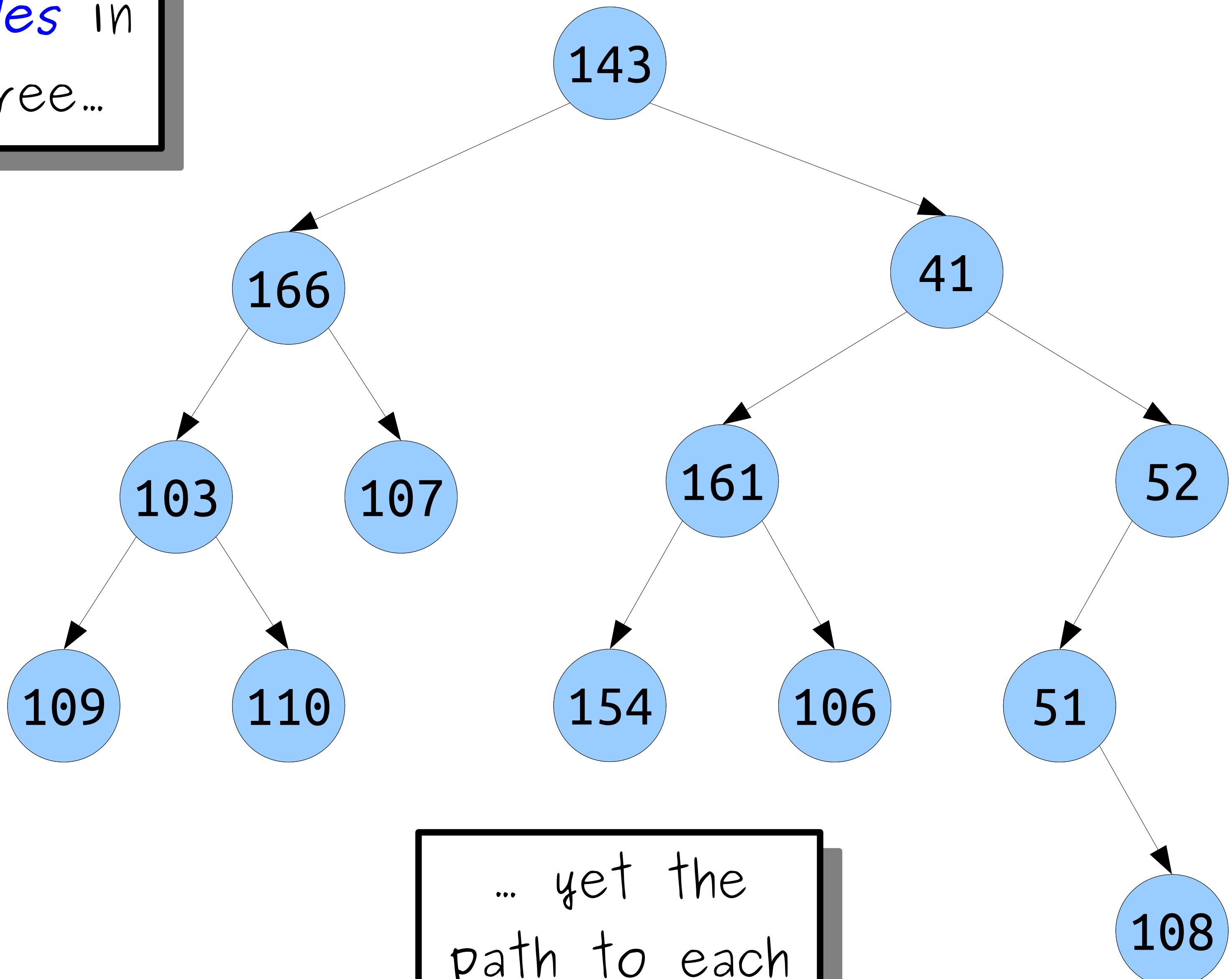

Your lungs have about 500 million alveoli…
… yet the path to each one is short.

Key Idea: The distance from the top of a tree to each node in the tree is small.

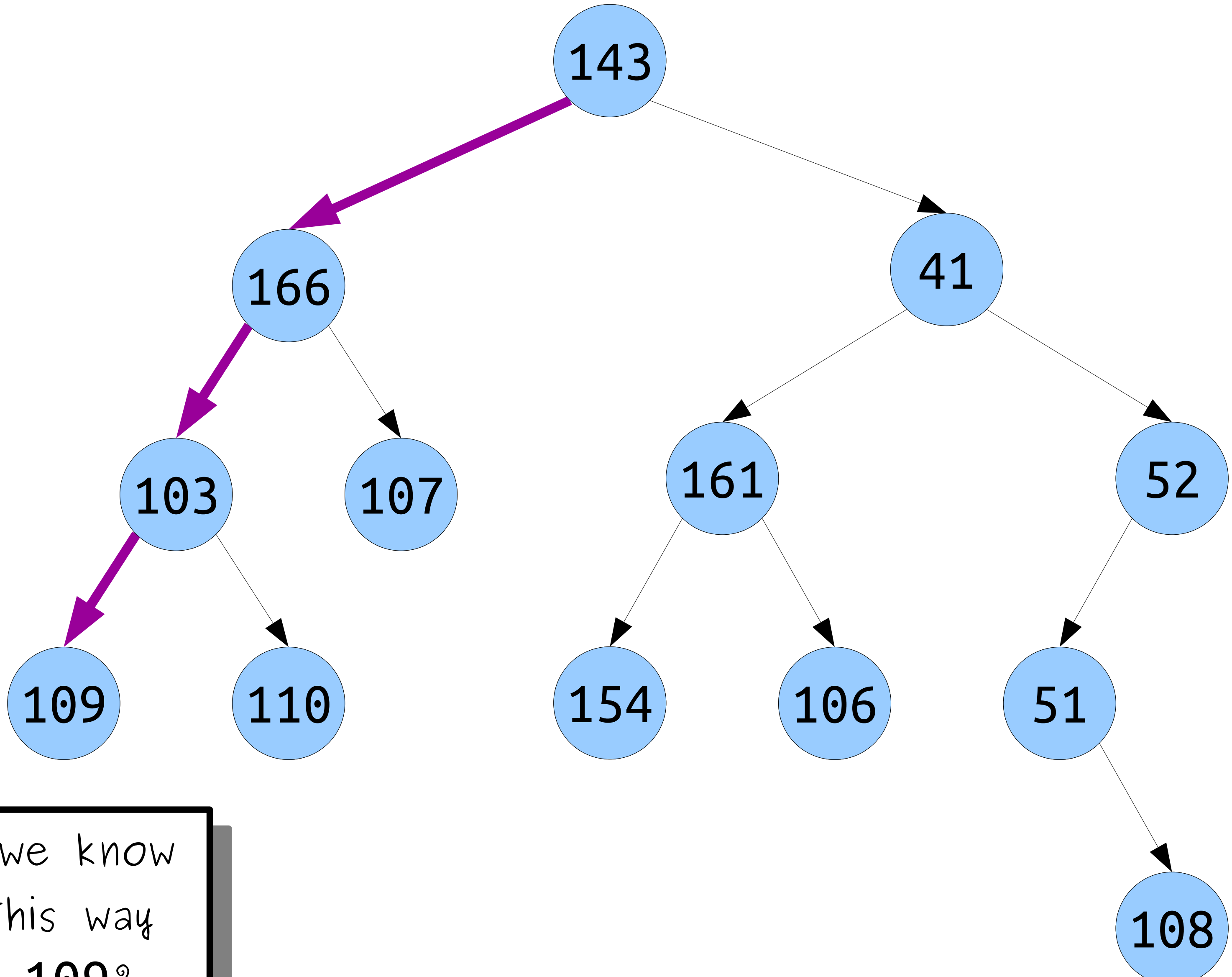
Harnessing this Insight


41
166
143
161
108
52
154
106
110
107
103
109
51

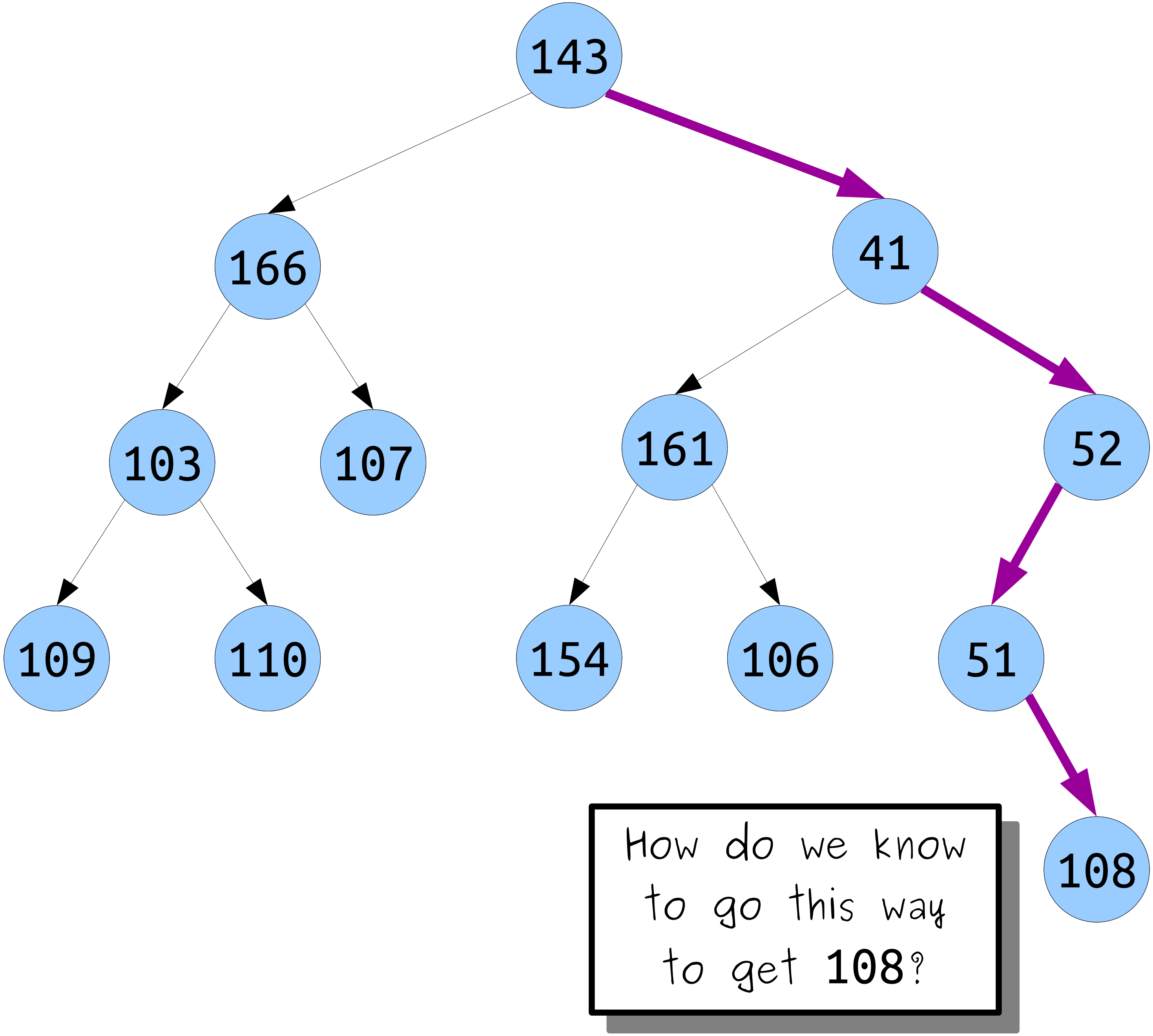
There are 13 *nodes* in this tree…



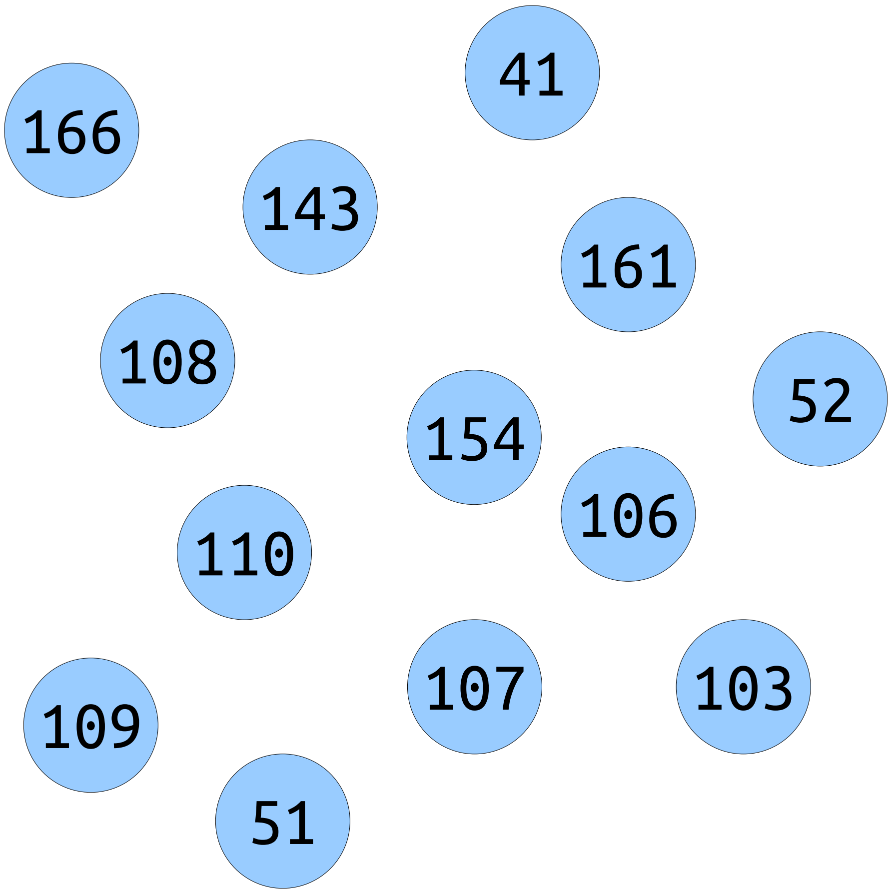
… yet the path to each one is short.

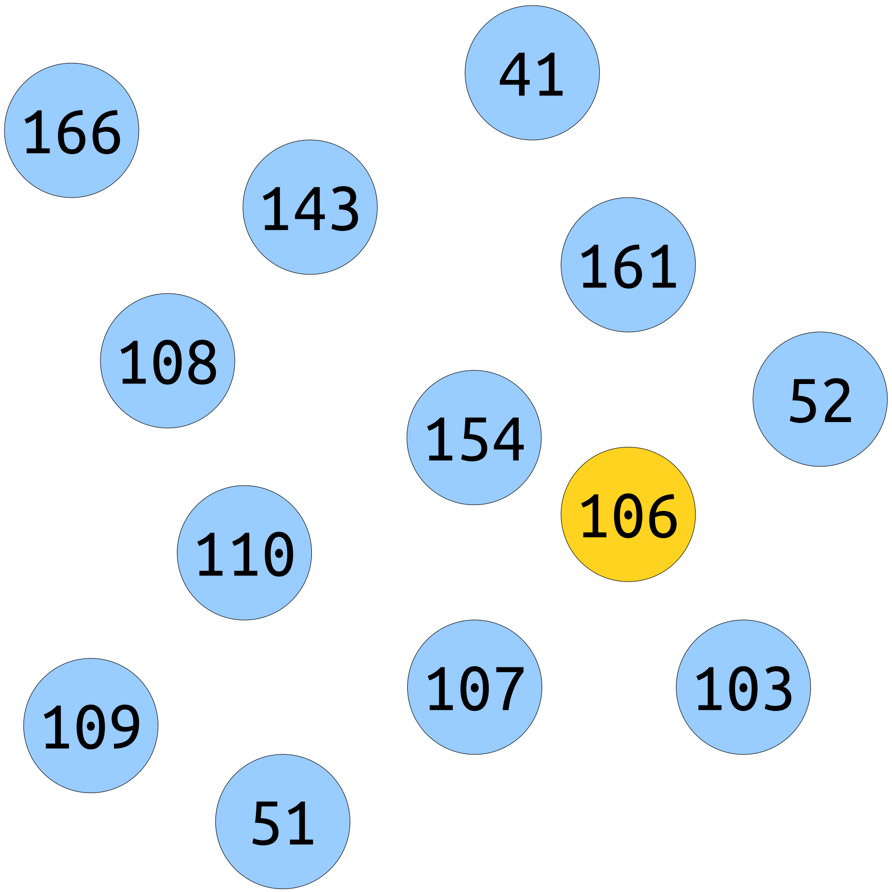


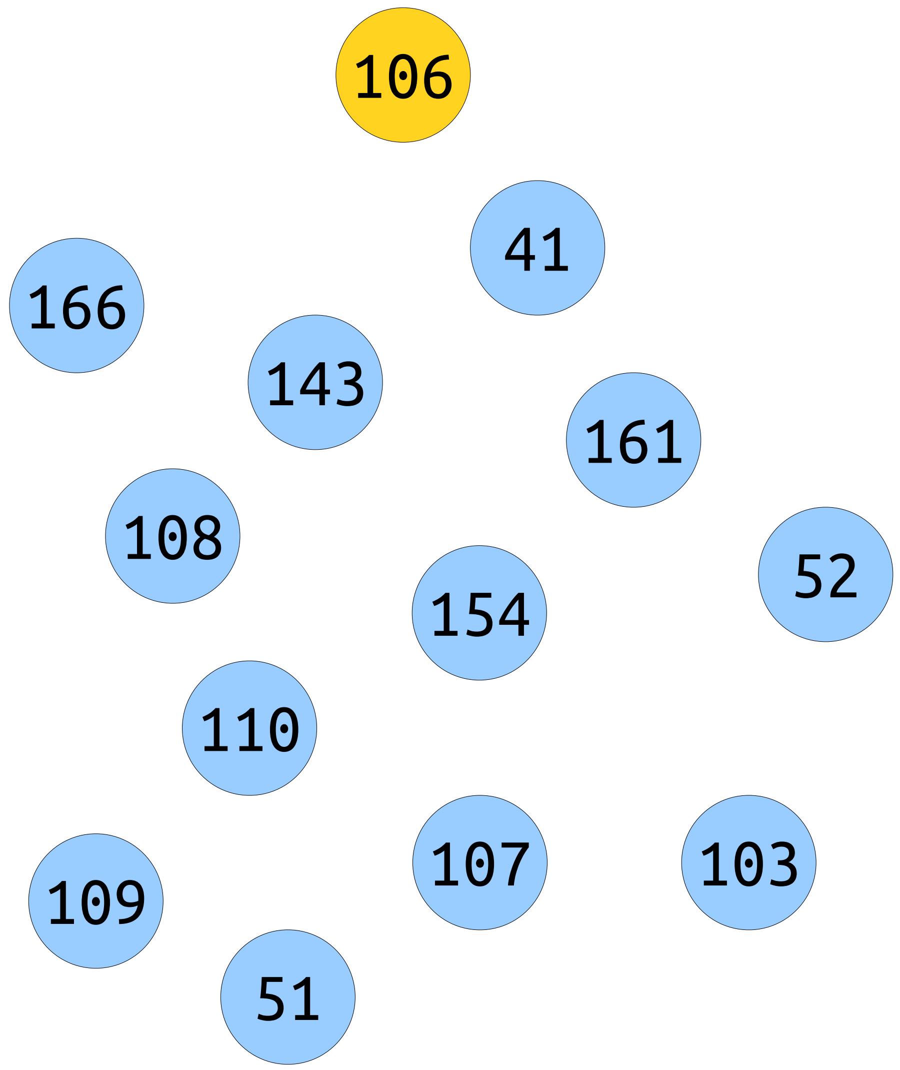
How do we know to go this way to get **109**?

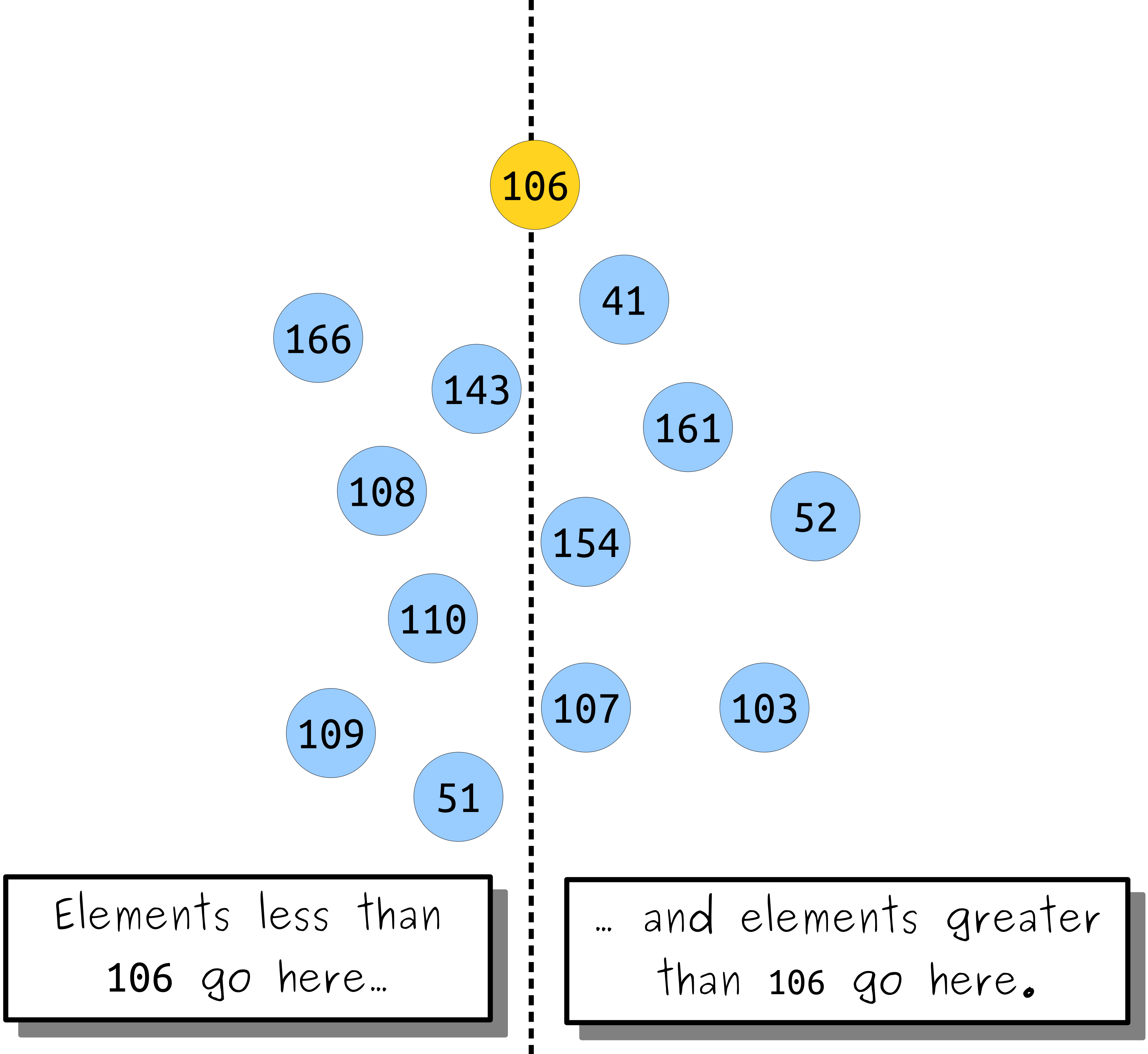

143
166
41
103
107
161
52
109
110
154
106
51
108
How do we know to go this way to get 108?

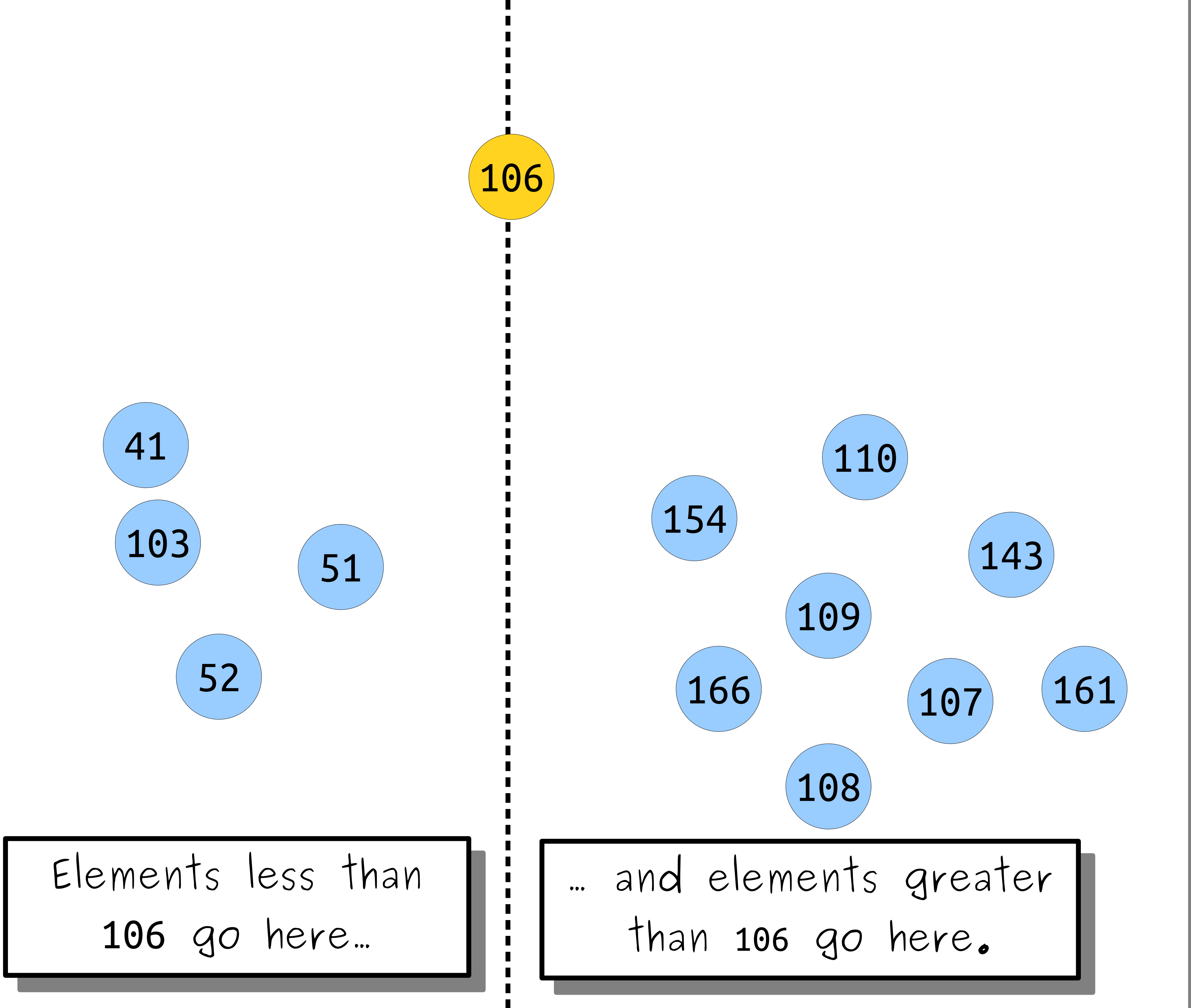
Goal: Store elements in a tree structure where there's an easy way to find them.

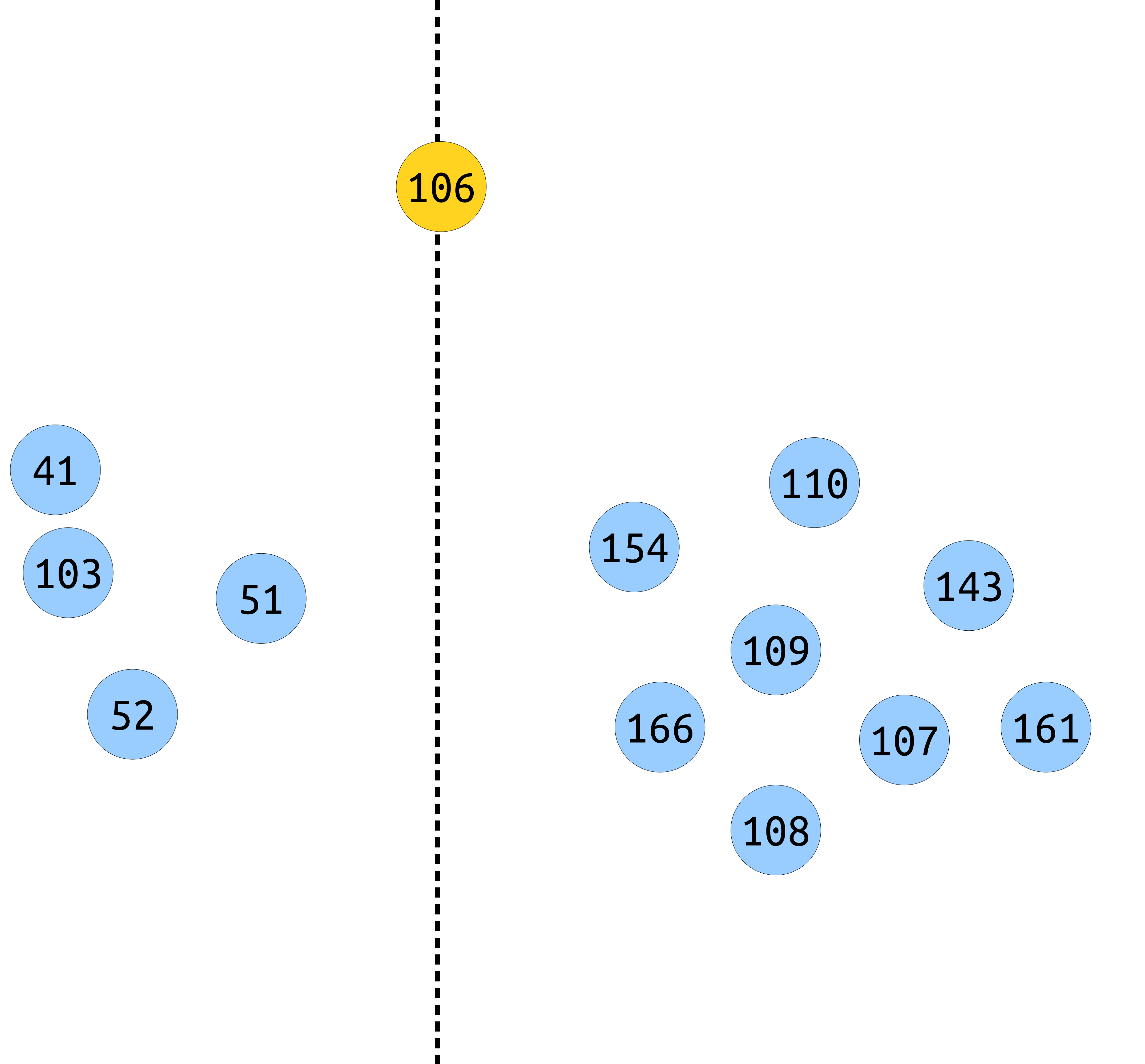

41
166
143
161
108
52
154
106
110
107
103
109
51

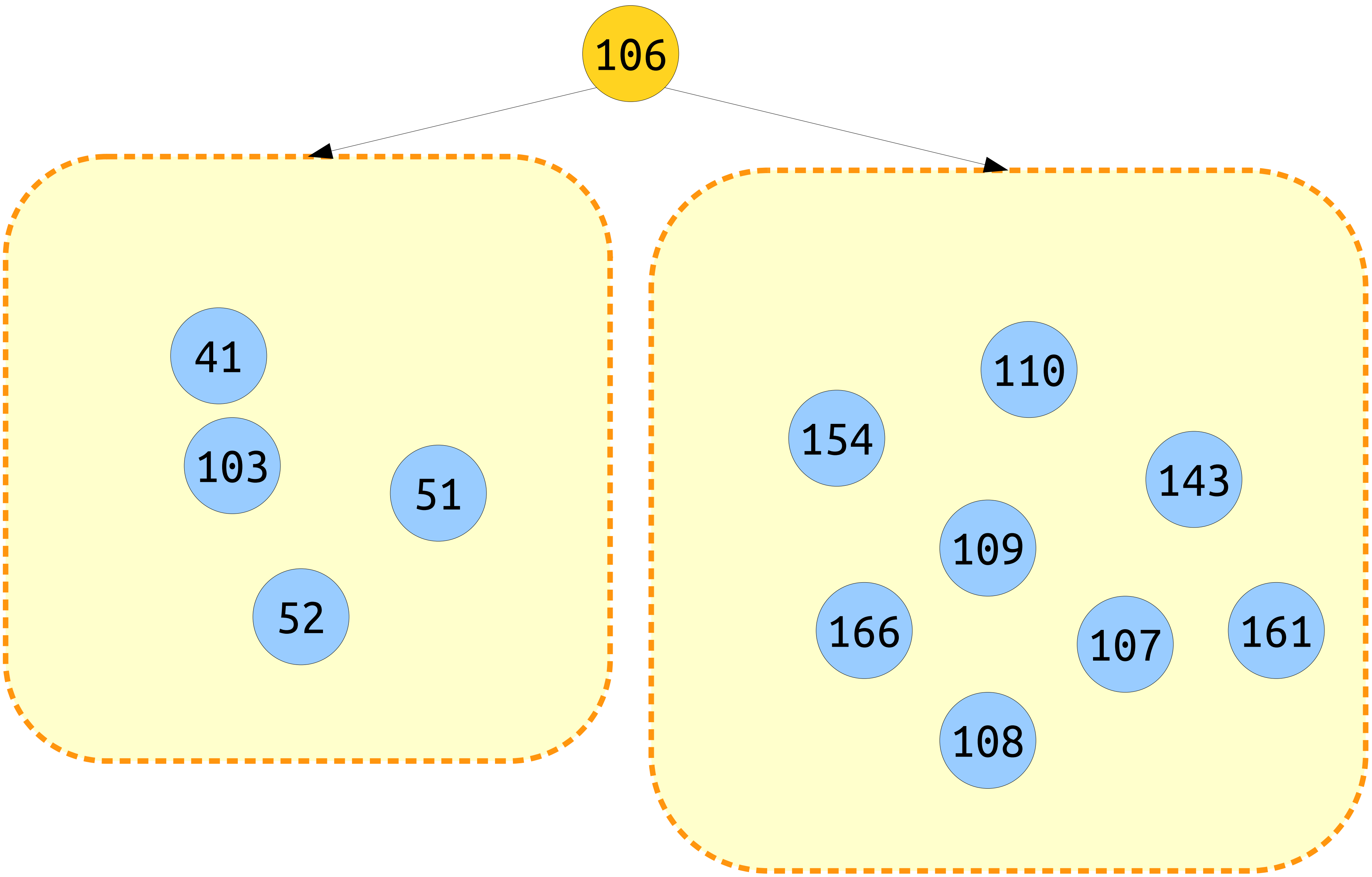

41
166
143
161
108
52
154
106
110
107
103
109
51

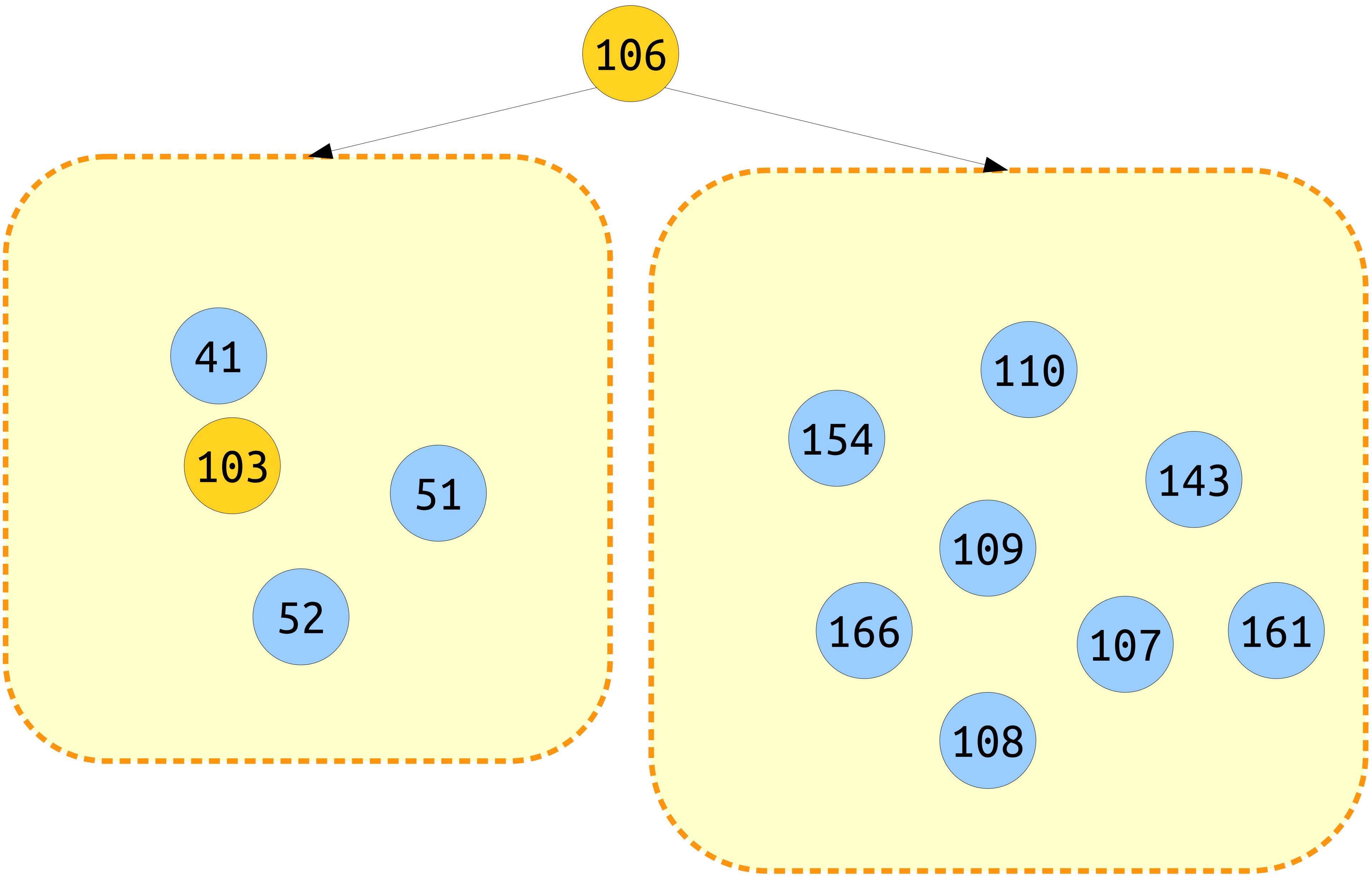

106
41
166
143
161
108
154
52
110
107
103
109
51

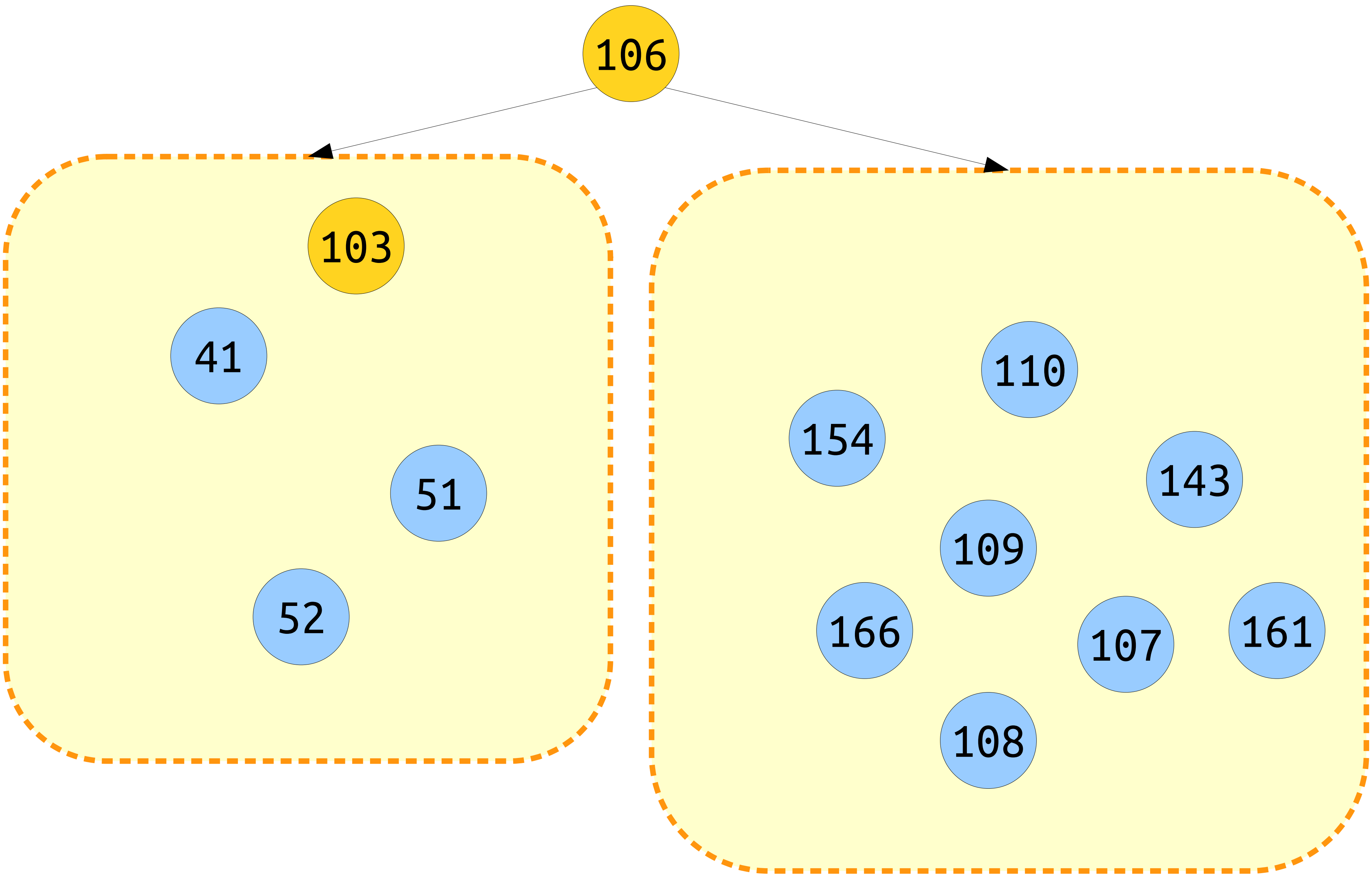

106
166
41
143
161
108
52
154
110
107
103
109
51
Elements less than 106 go here...
... and elements greater than 106 go here.

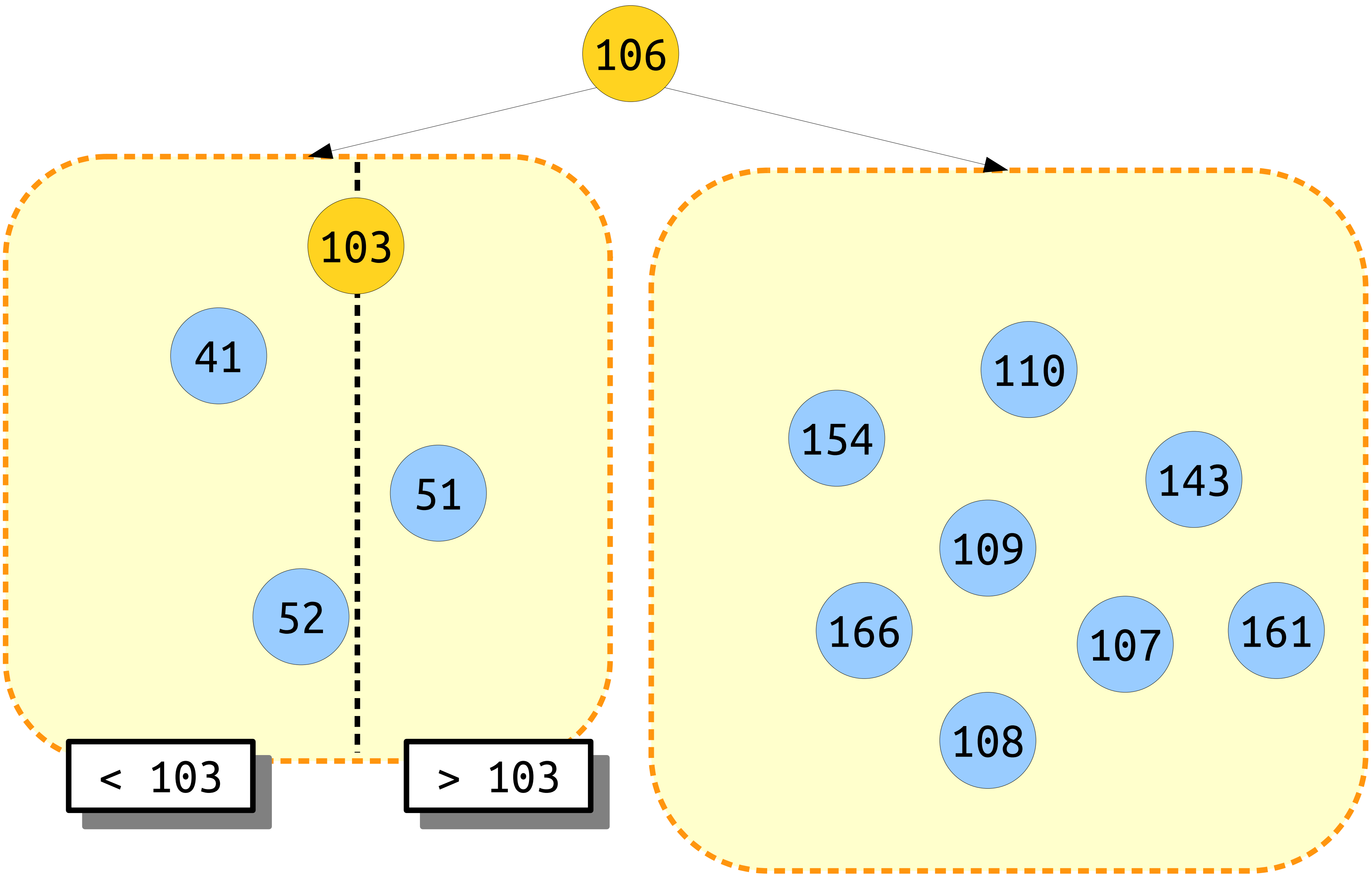

106
41
103
51
52
110
154
143
109
166
107
161
108
Elements less than 106 go here…
… and elements greater than 106 go here.

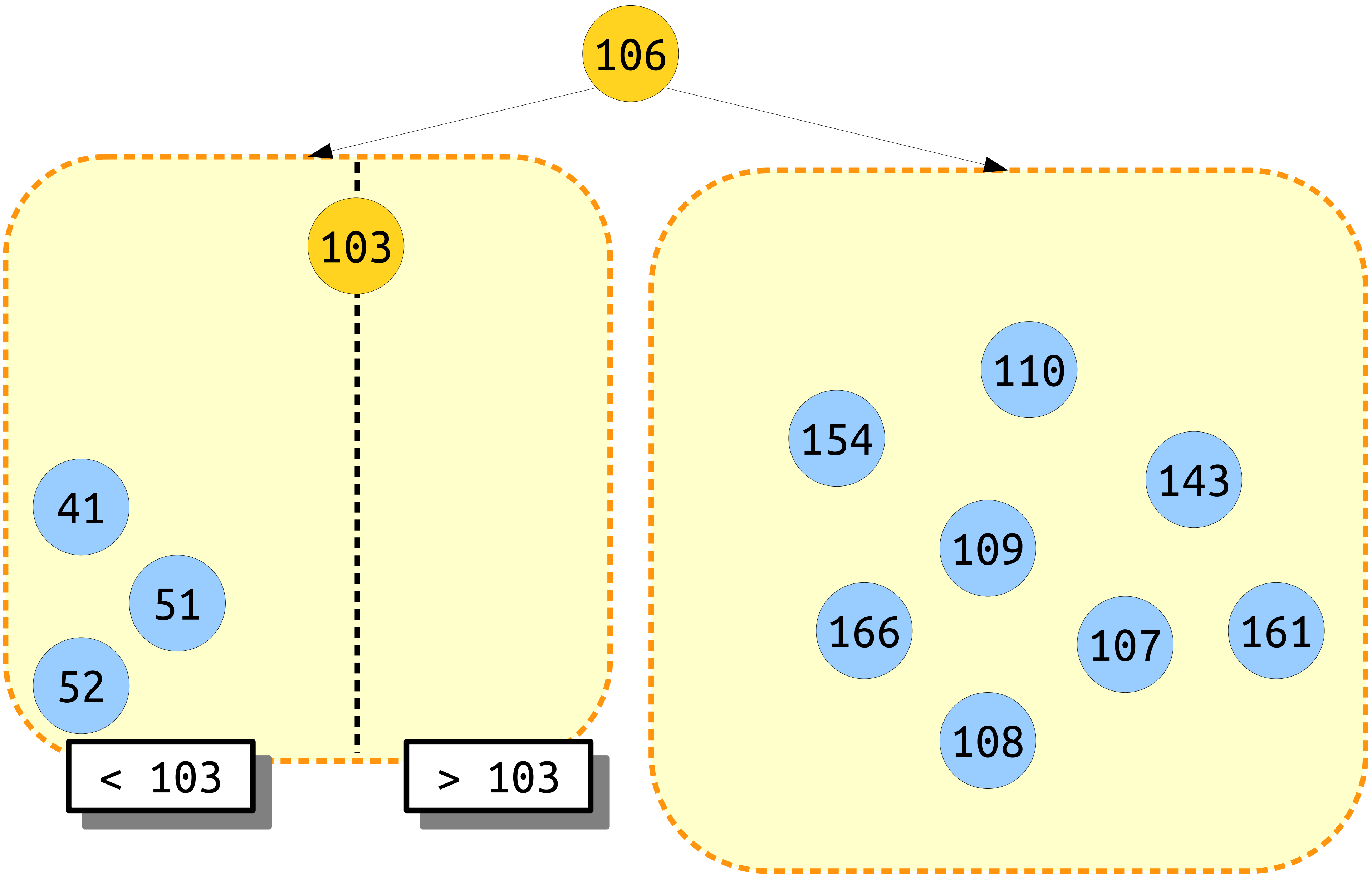

106
41
103
51
52
110
154
143
109
166
107
161
108

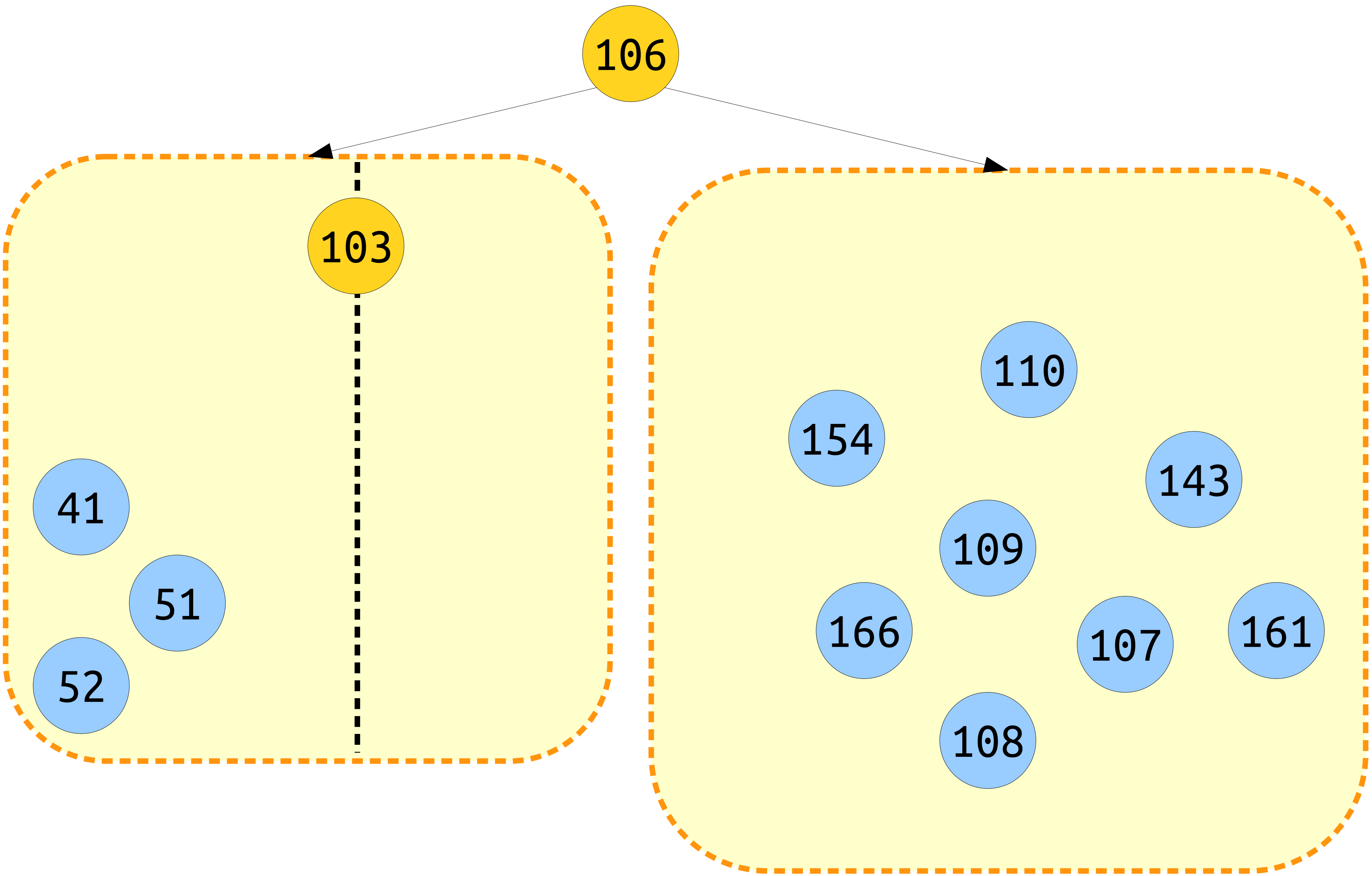

106
41
103
51
52
110
154
143
109
166
107
161
108

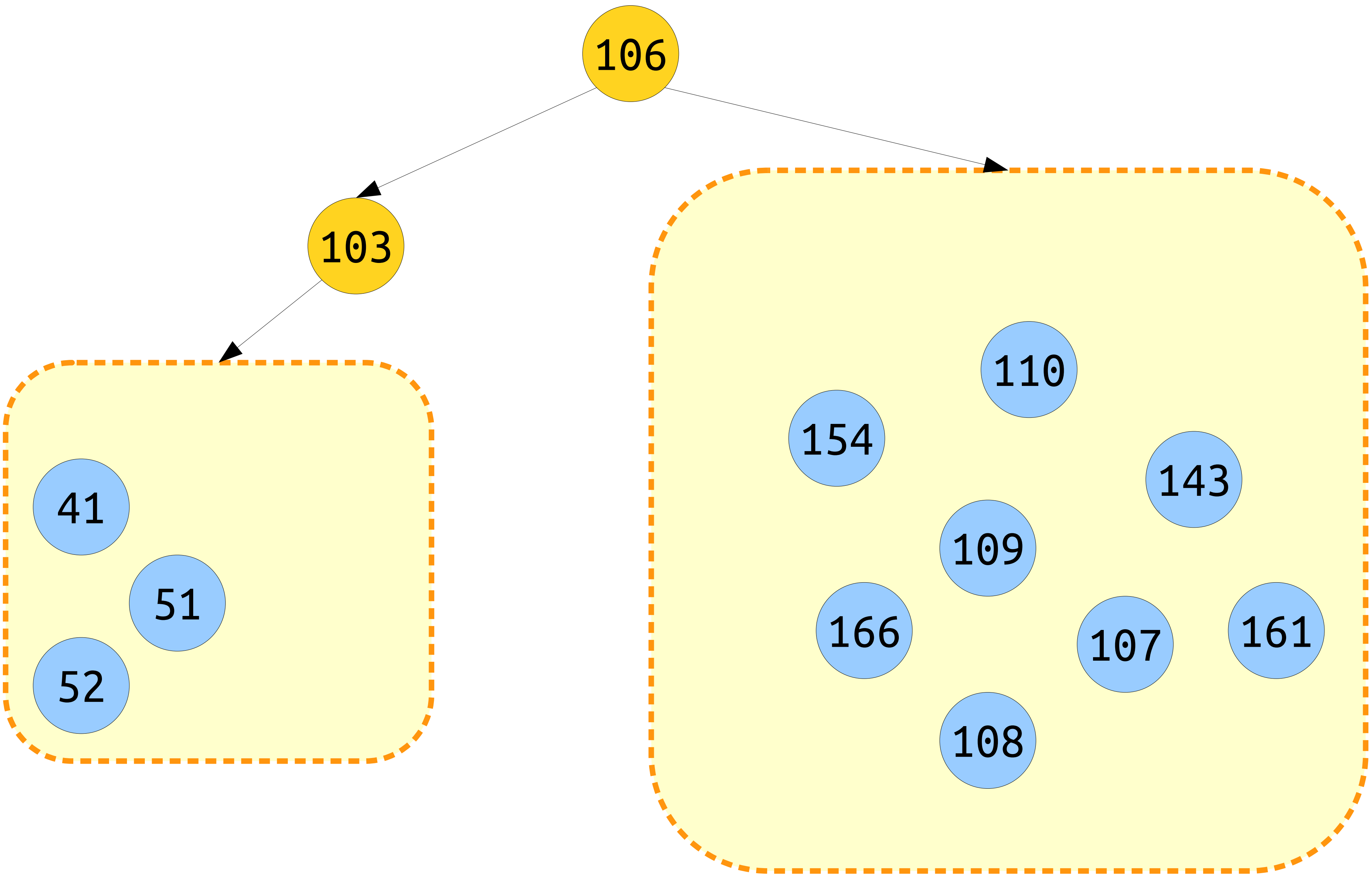

106
41
103
51
52
110
154
143
109
166
107
161
108

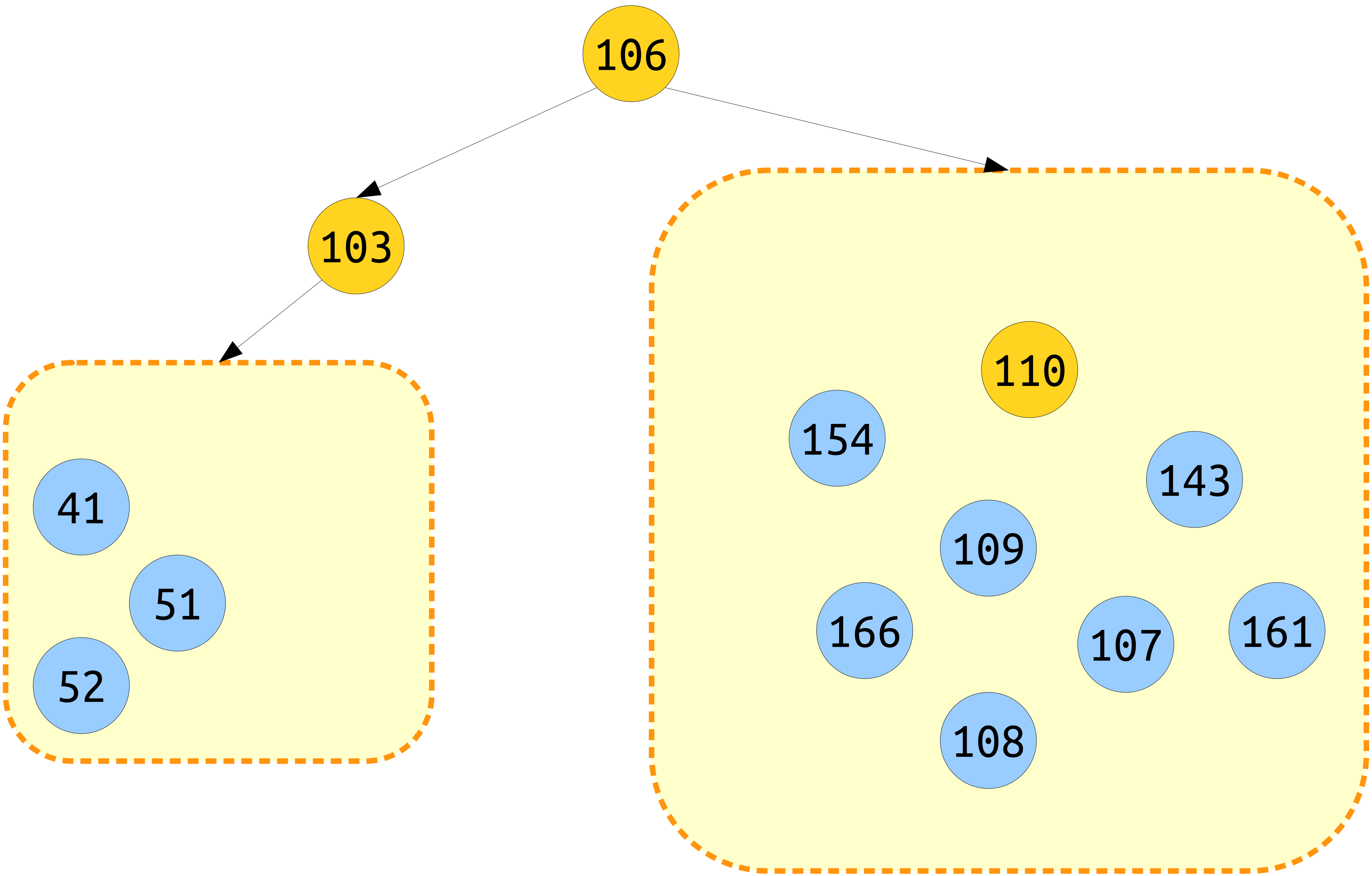

106
103
41
51
52
110
154
143
109
166
107
161
108


106
103
41
51
52
< 103
> 103
110
154
143
109
166
107
161
108

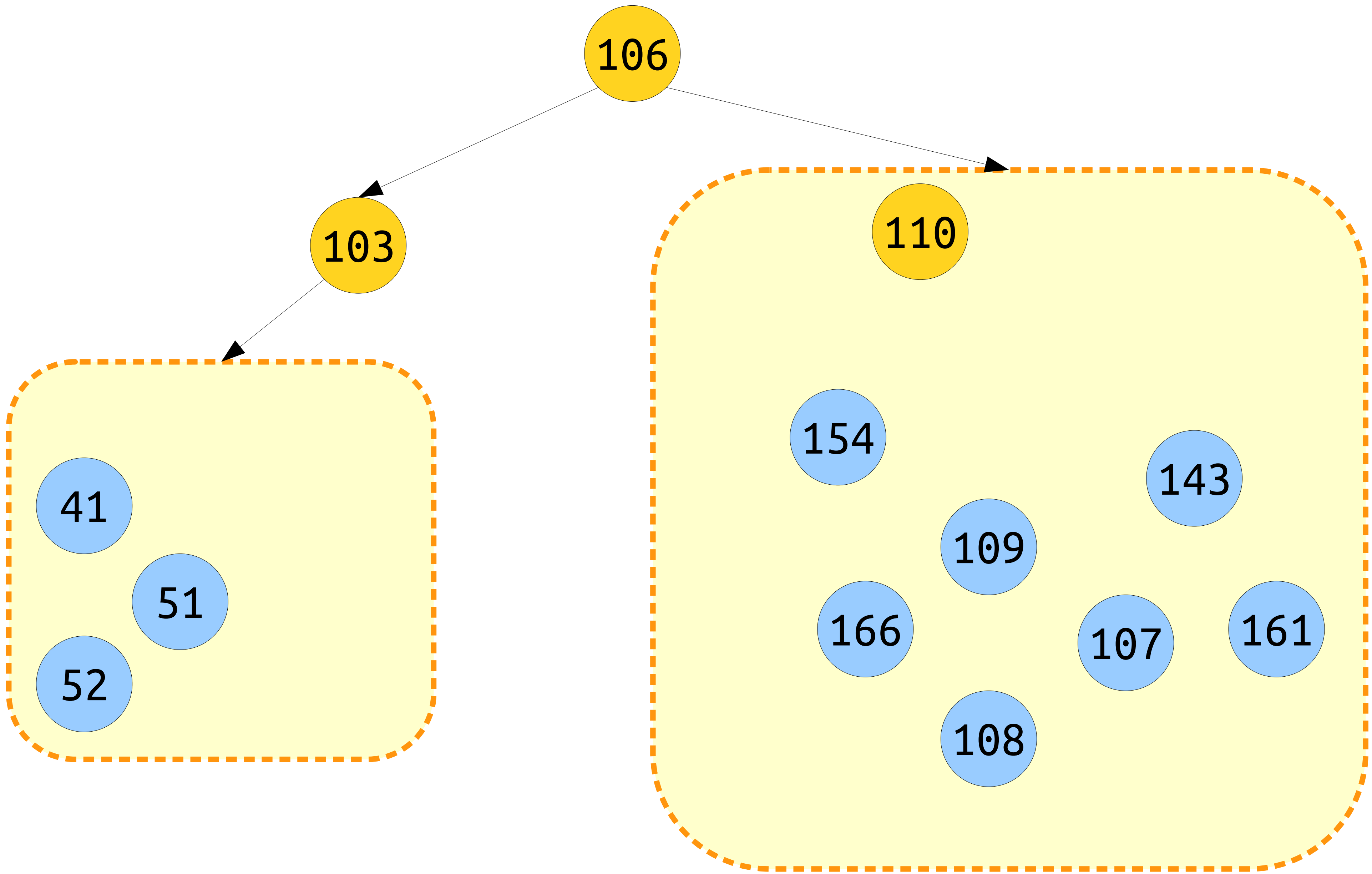

106
103
41
51
52
< 103
> 103
110
154
143
109
166
107
161
108

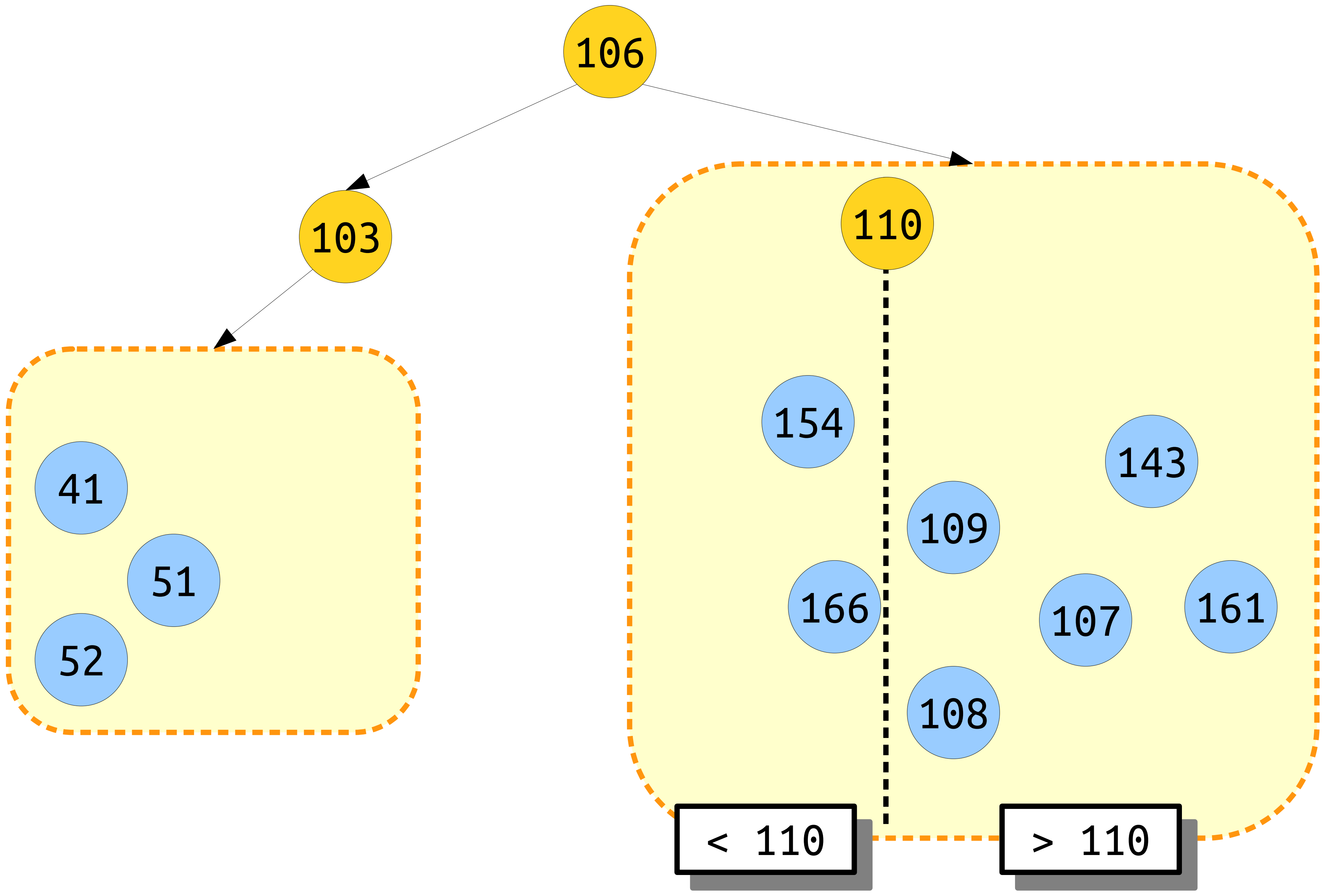

106
103
41
51
52
110
154
143
109
166
107
161
108

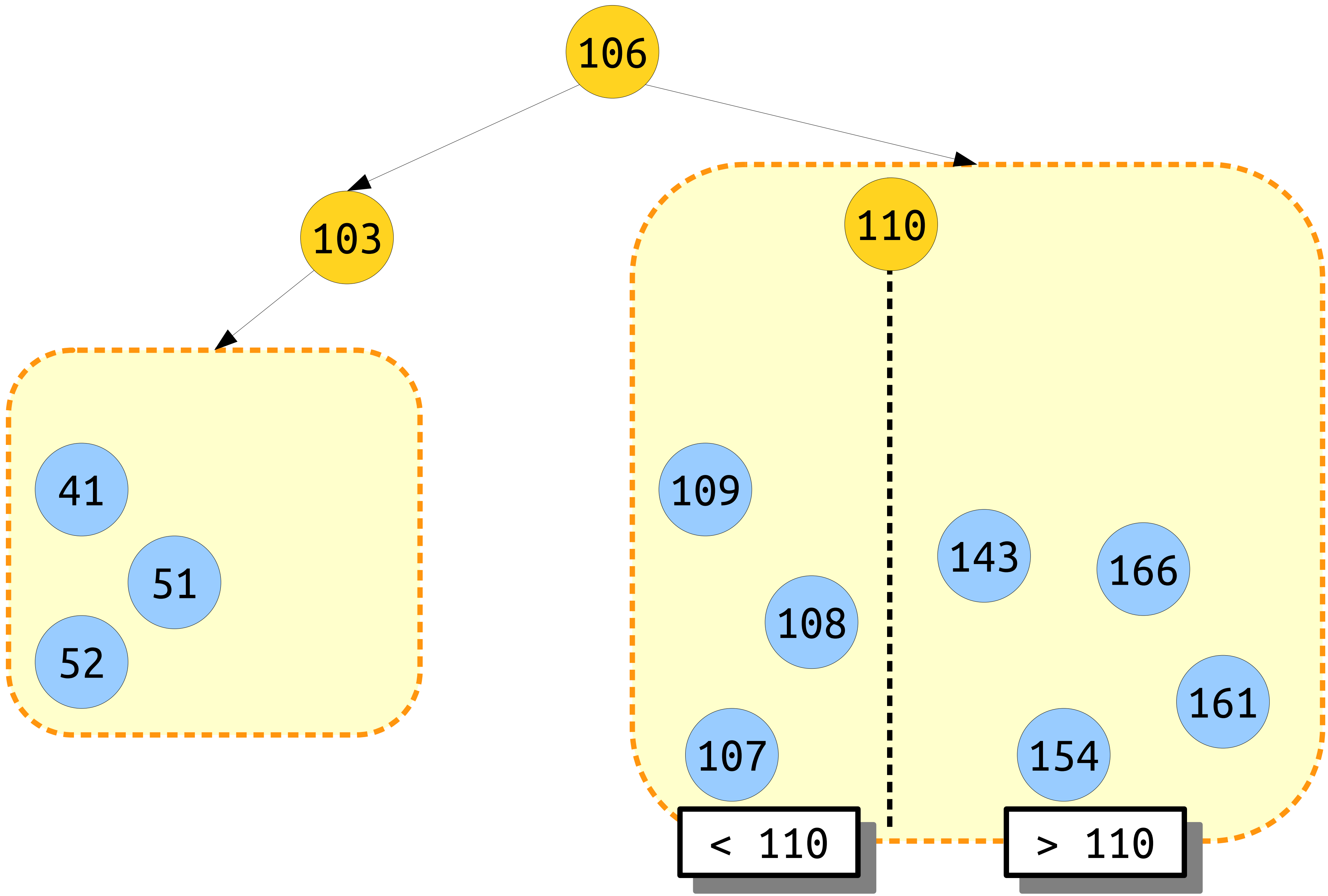

106
103
41
51
52
110
154
143
109
166
107
161
108

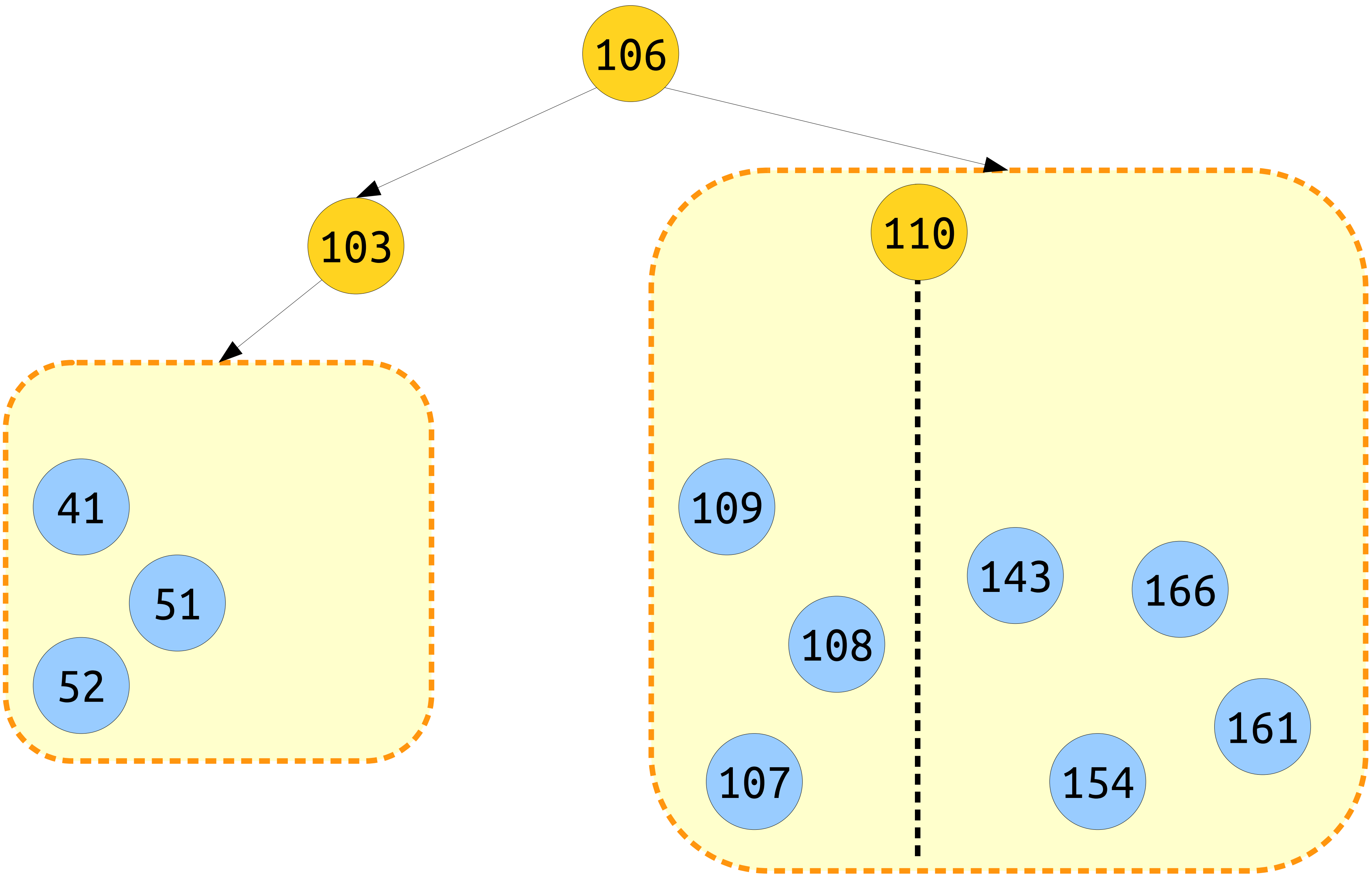


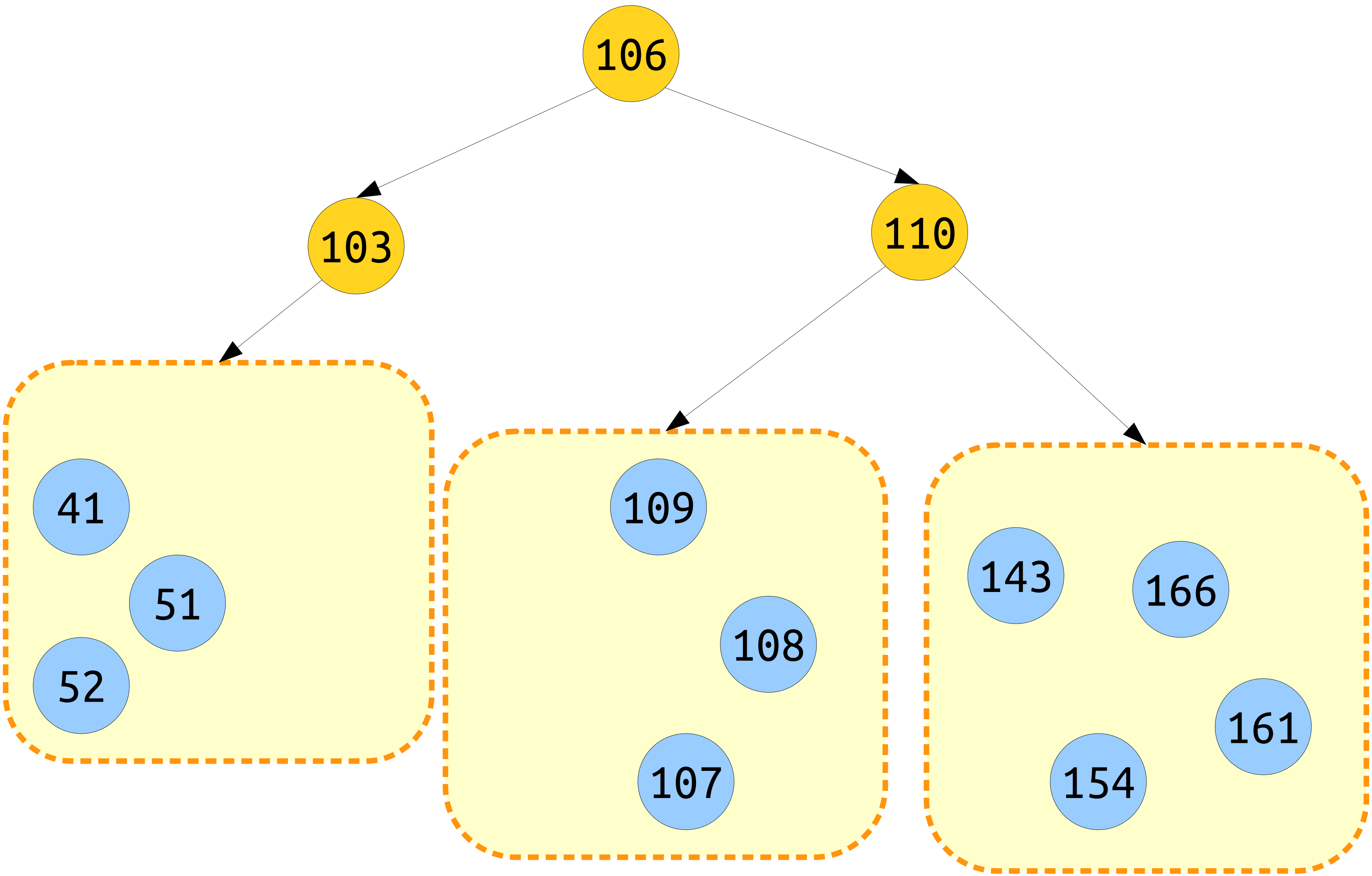
106
103
110
154
143
41
109
51
166
107
161
52
108

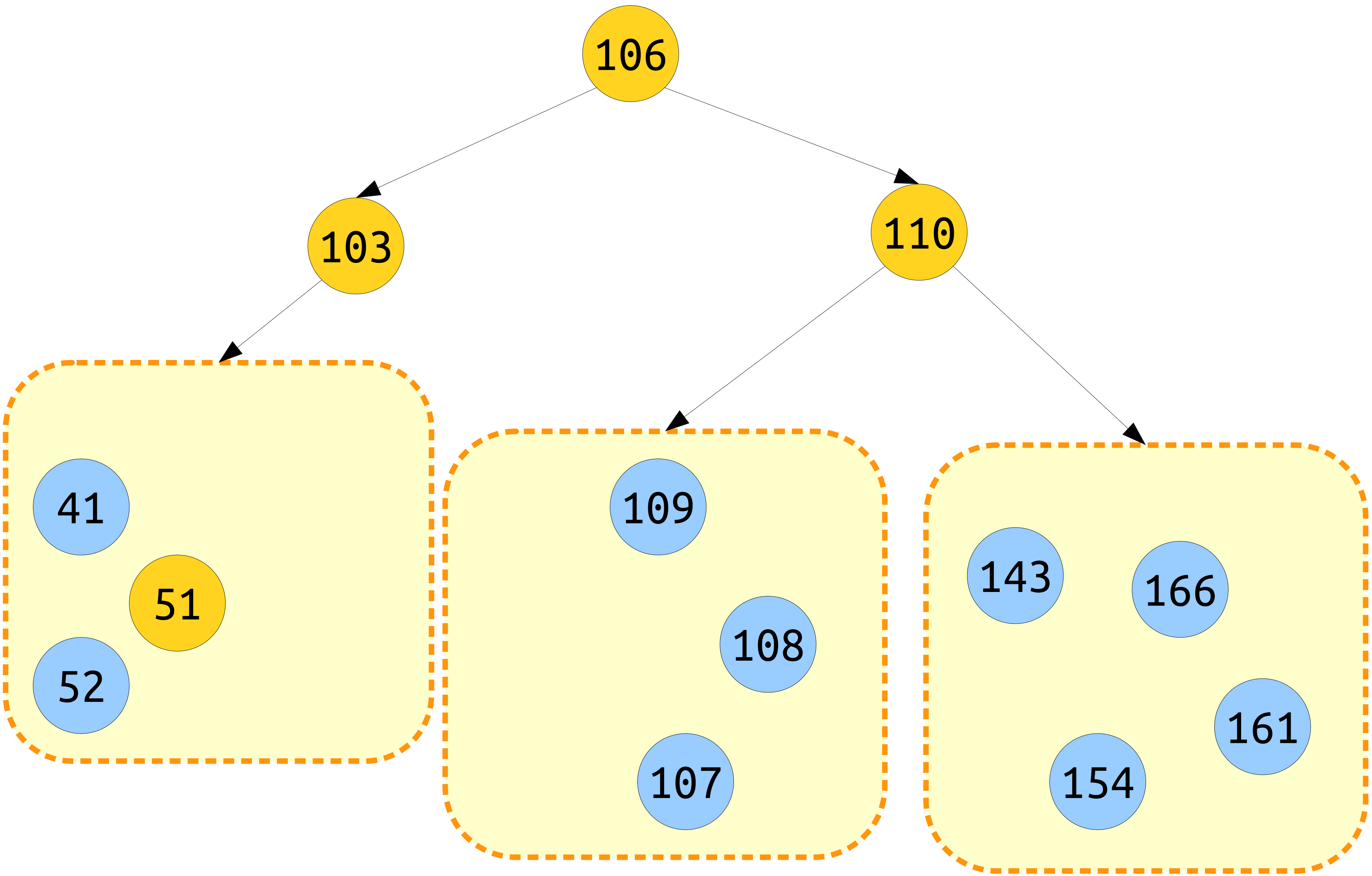

106
103
110
41
51
52
154
143
109
166
107
161
108

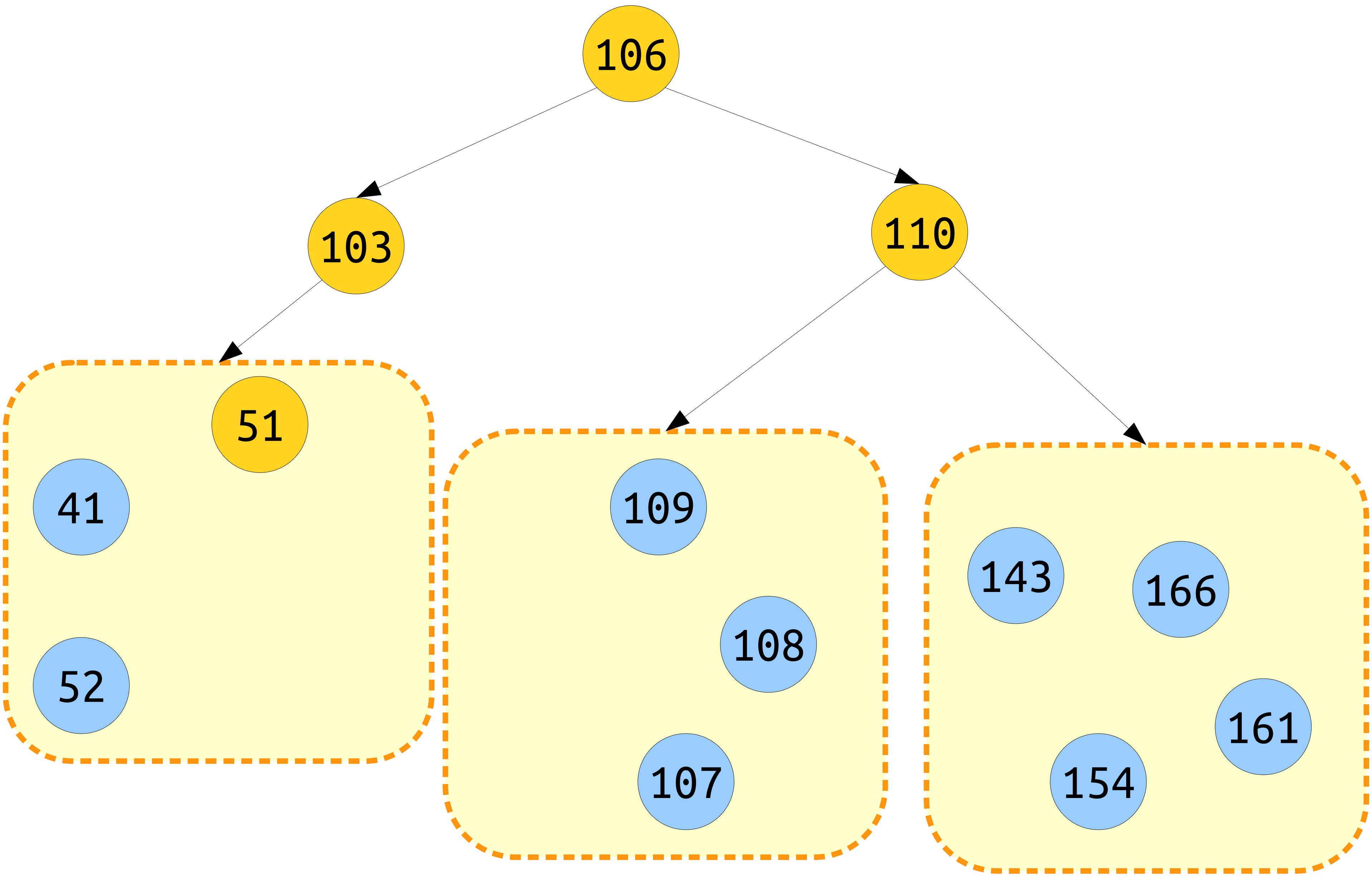

106
103
110
41
51
52
154
143
109
166
107
161
108
< 110
> 110


106
103
110
41
51
52
109
108
107
143
166
161
154
< 110
> 110

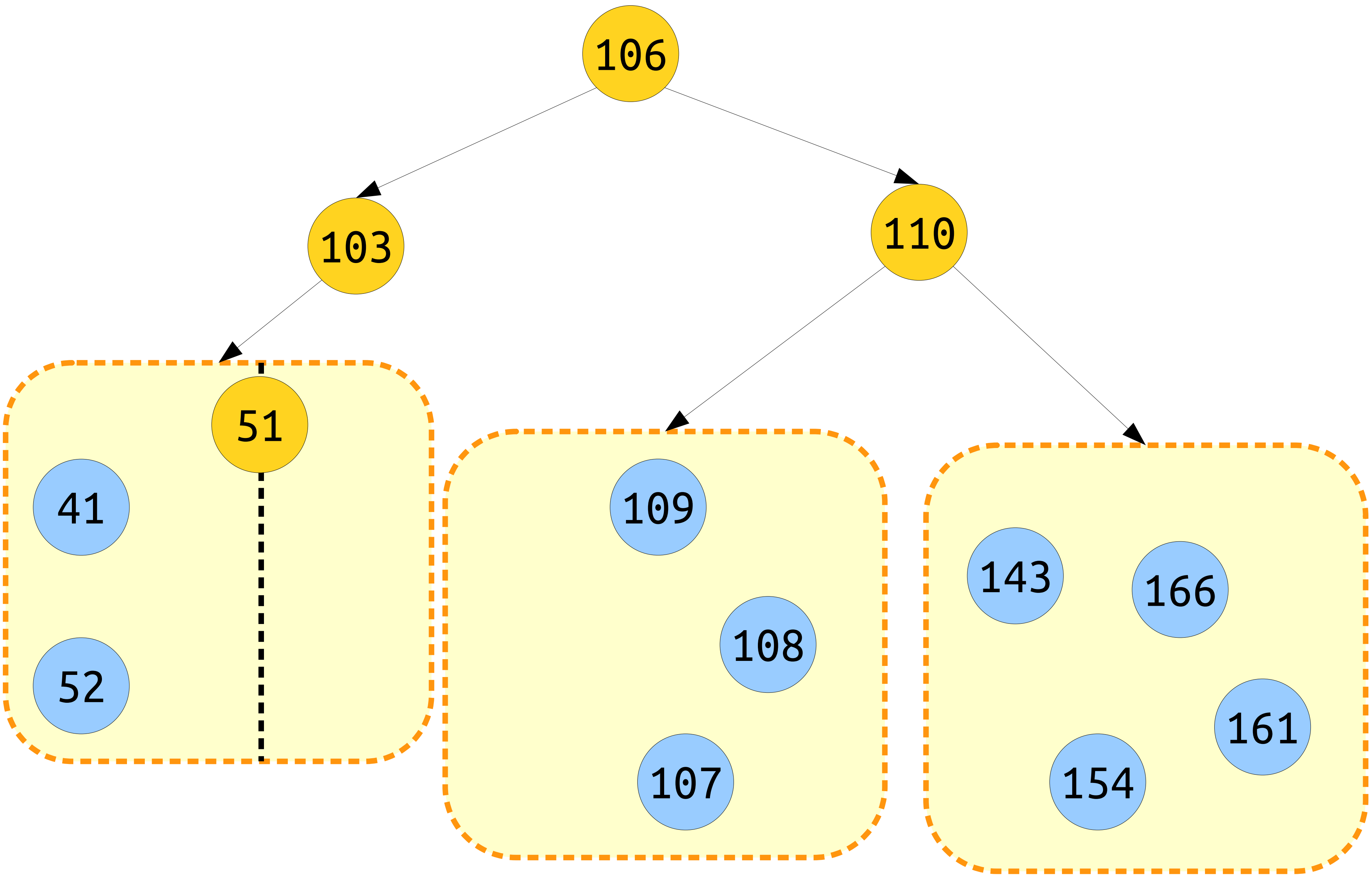

106
103
110
41
51
52
109
108
107
143
166
161
154

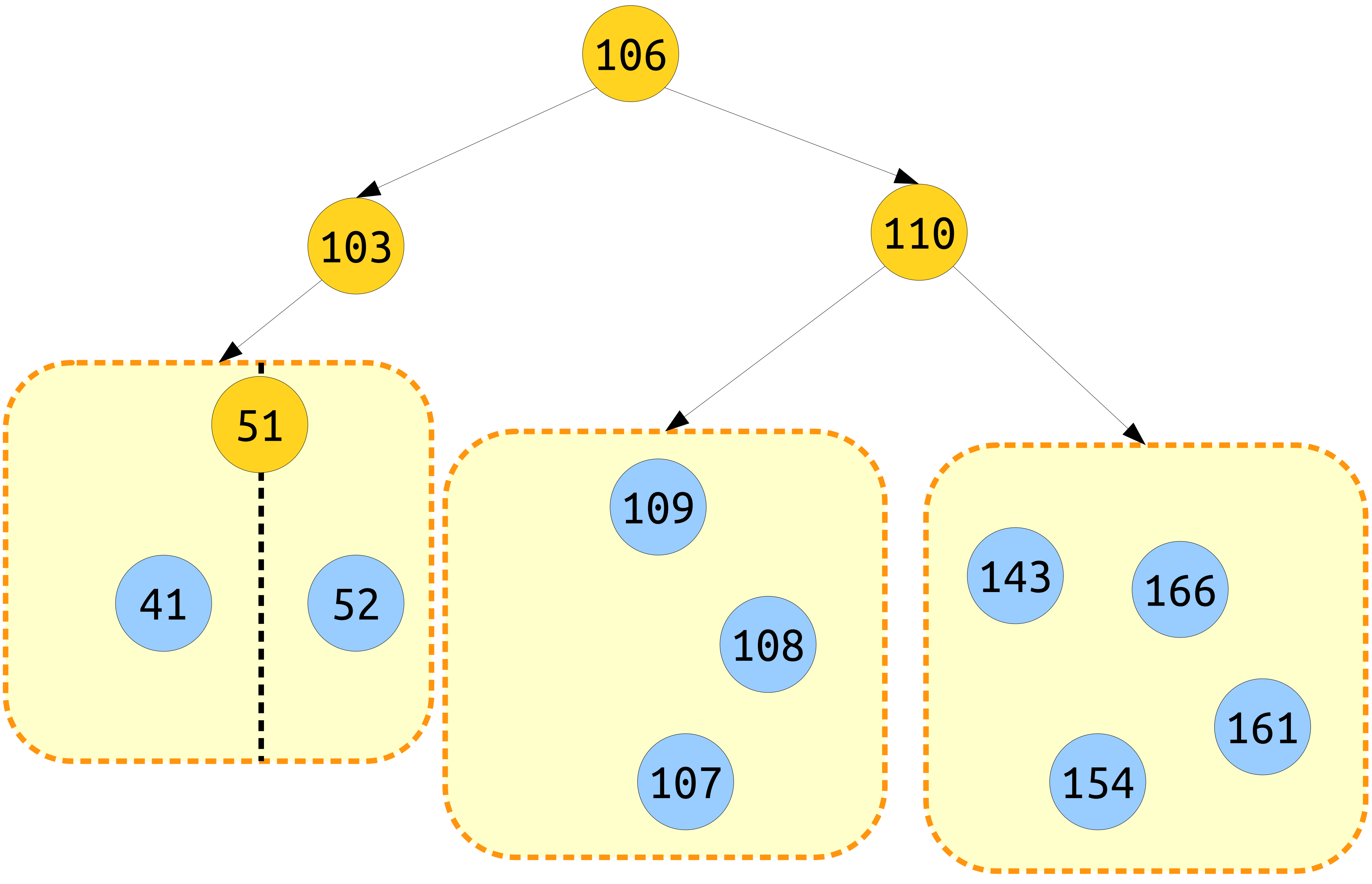

106
103
110
41
51
52
109
108
107
143
166
161
154

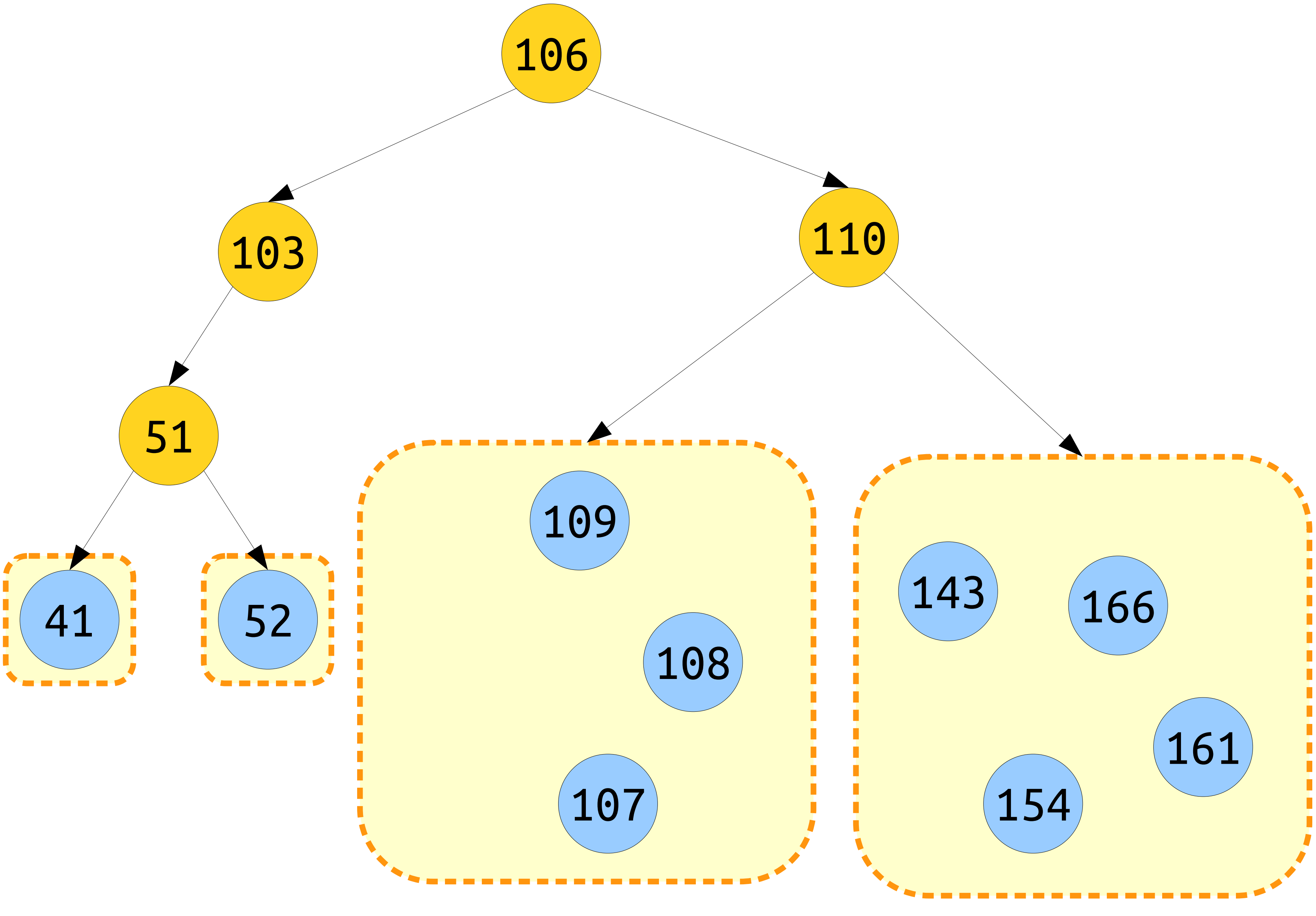

106
103
110
41
51
52
109
108
107
143
166
161
154

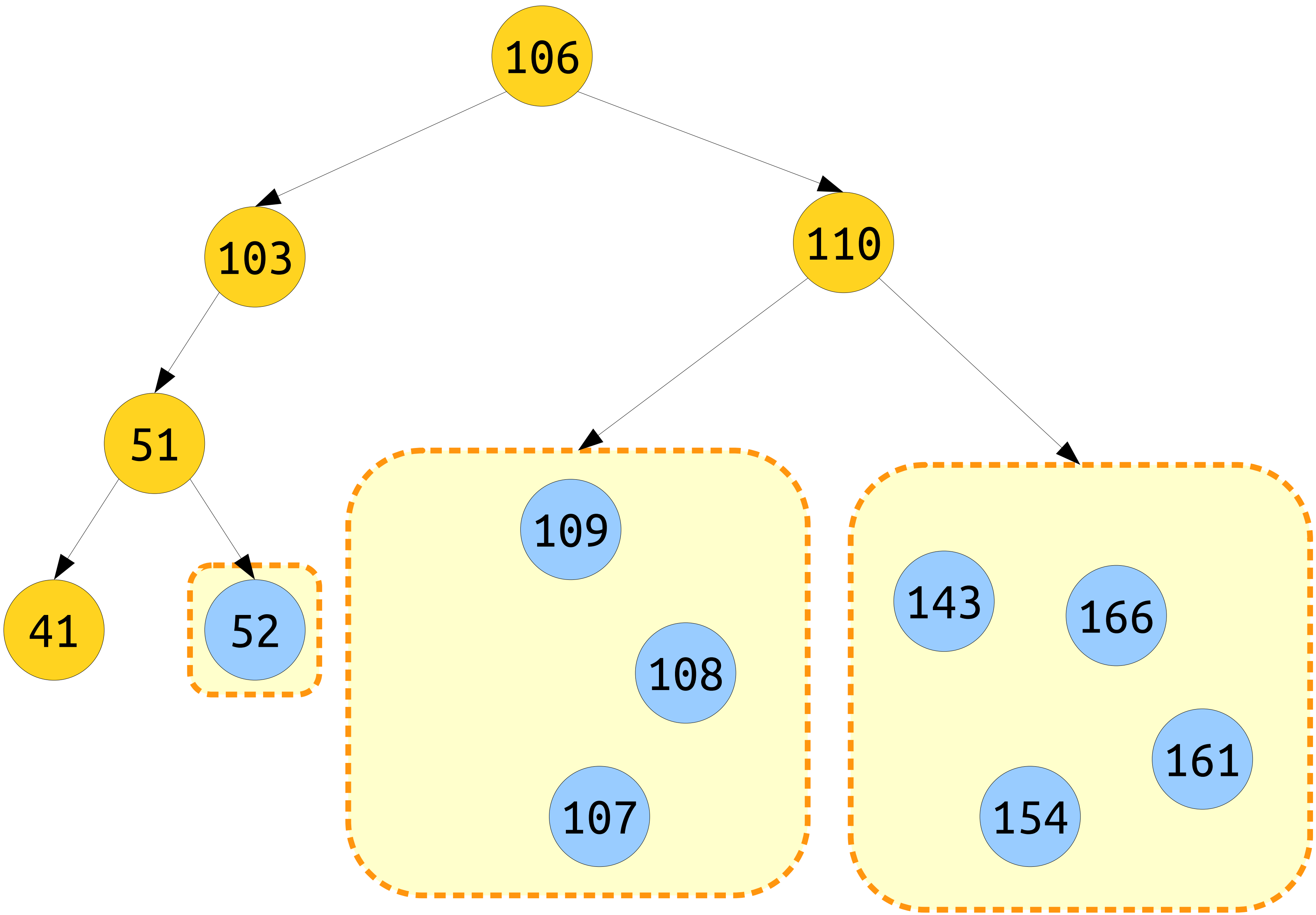


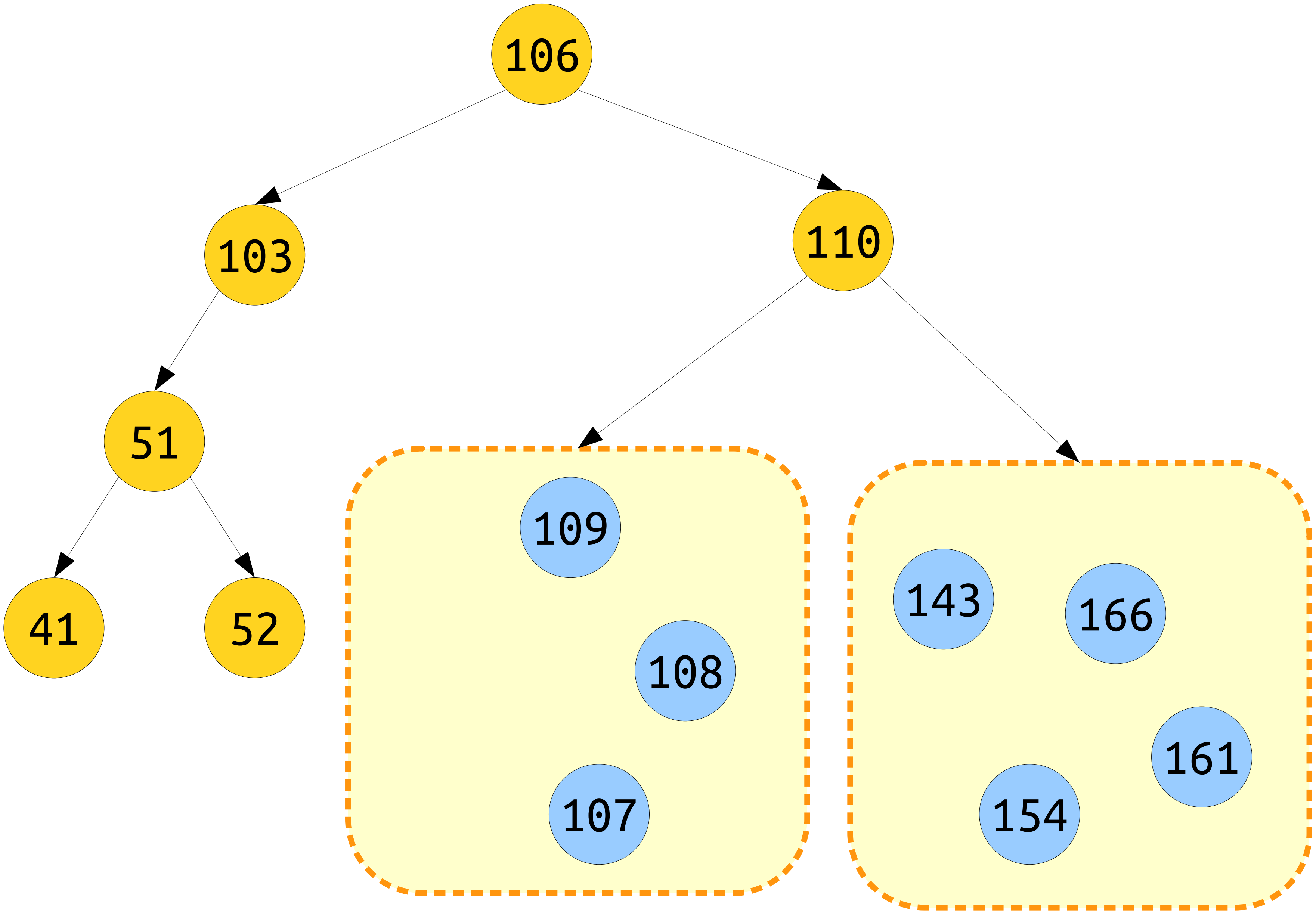
106
103
110
51
41
52
109
108
107
143
166
161
154


106
103
110
51
41
52
109
108
107
143
166
161
154


106
103
110
51
41
52
109
108
107
143
166
161
154

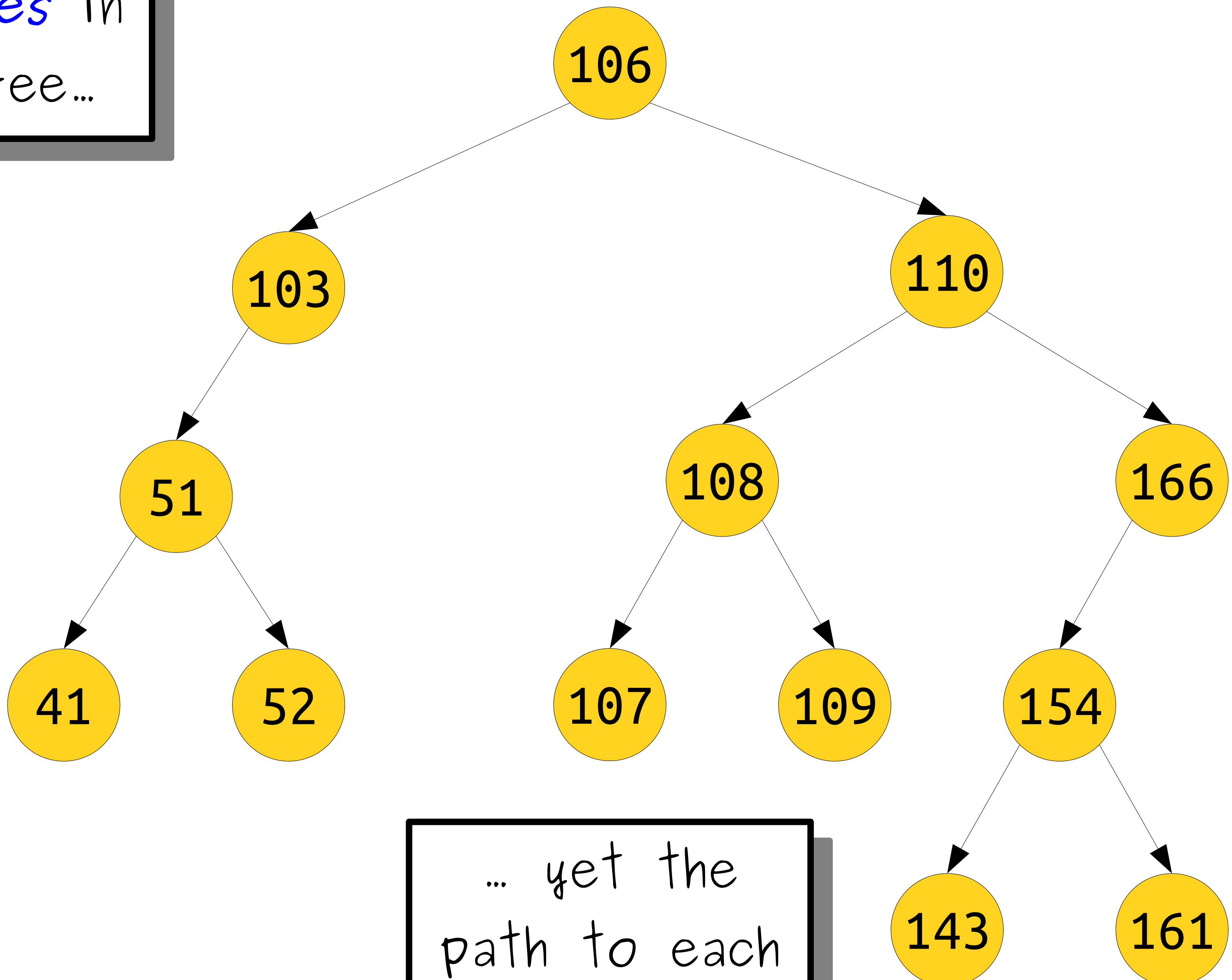

106
103
110
51
109
143
166
41
52
108
161
107
154


106
103
110
51
41
52
109
108
107
143
166
161
154

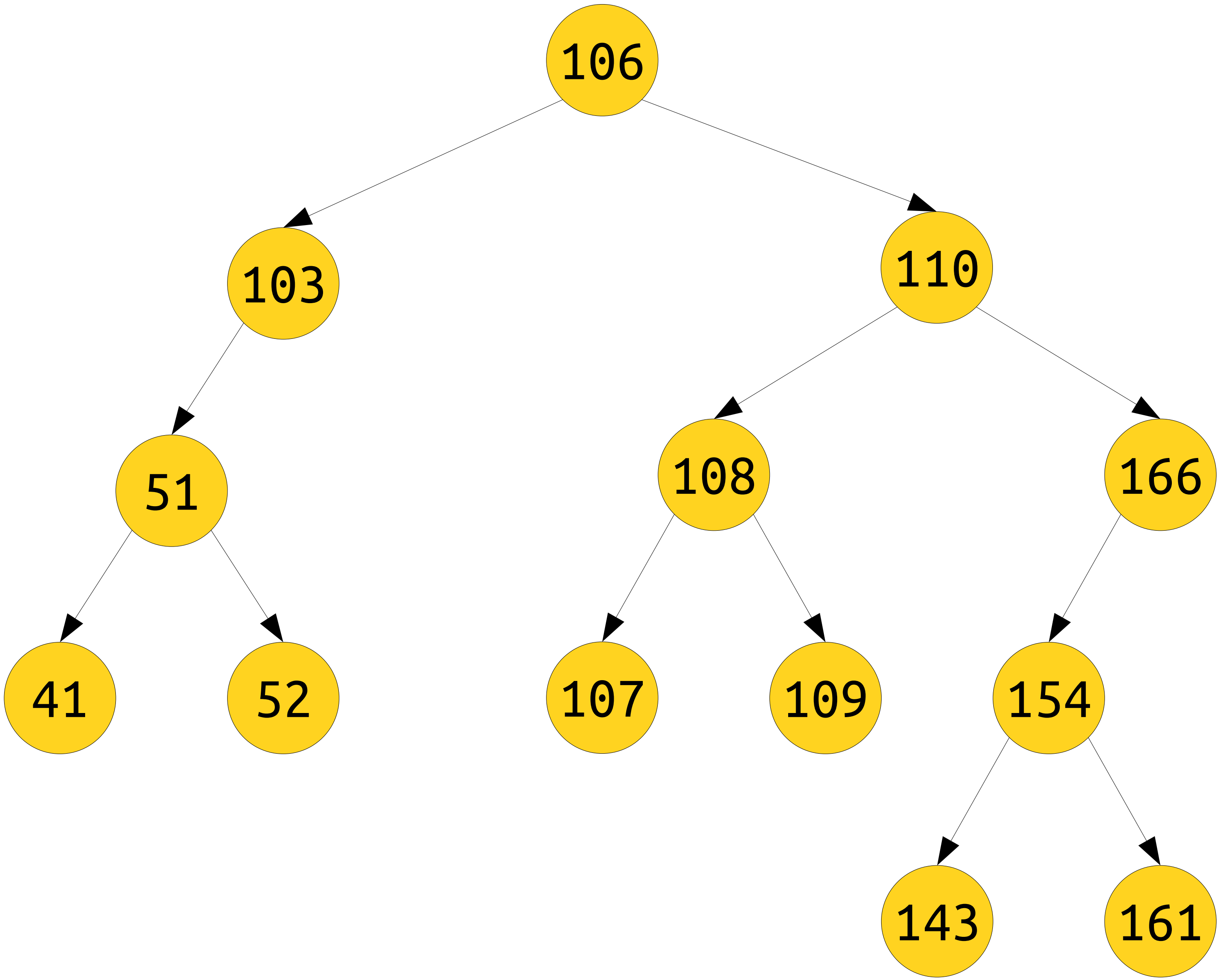


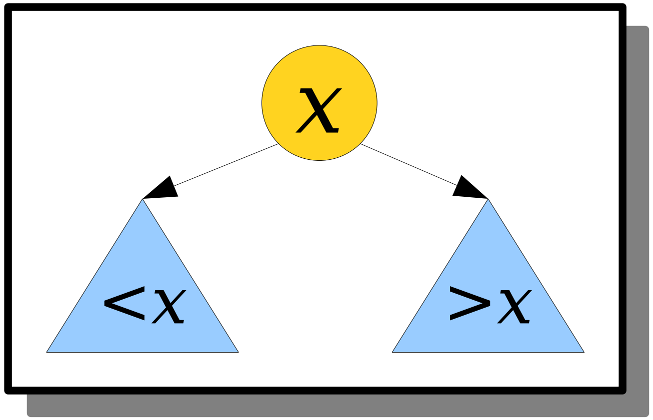
106
103
110
51
41
52
109
108
107
143
166
161
154

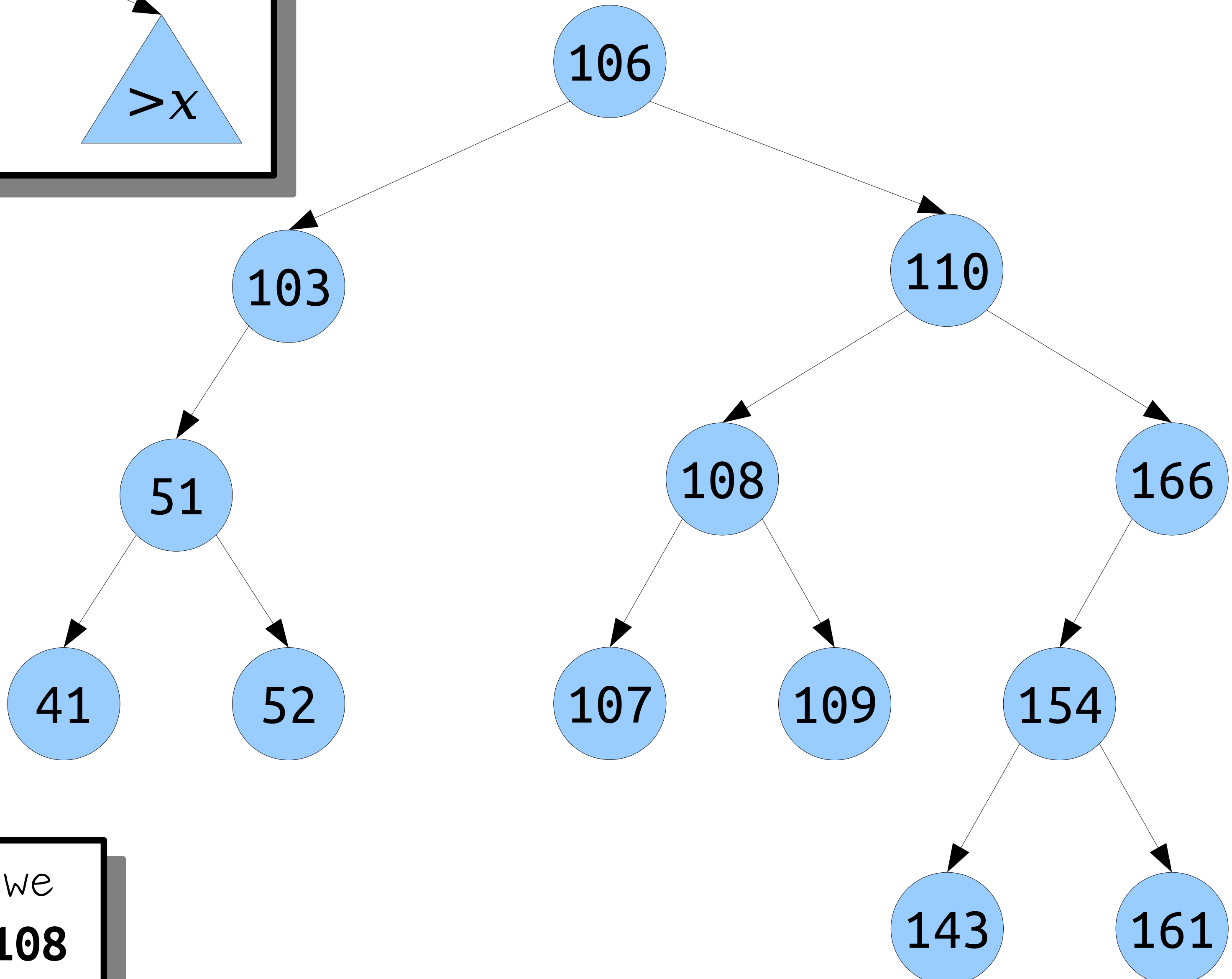
There are 13 *nodes* in this tree...



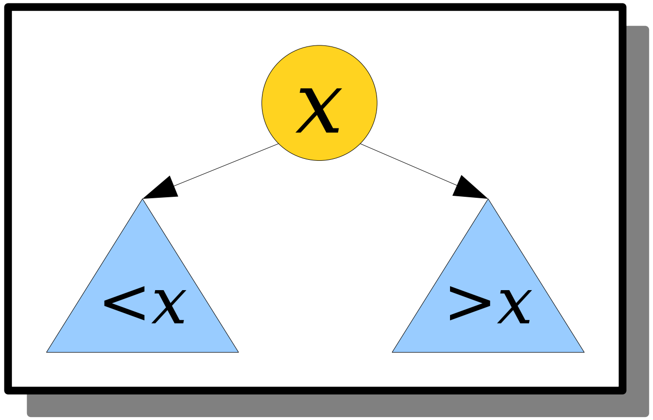
... yet the path to each one is short.

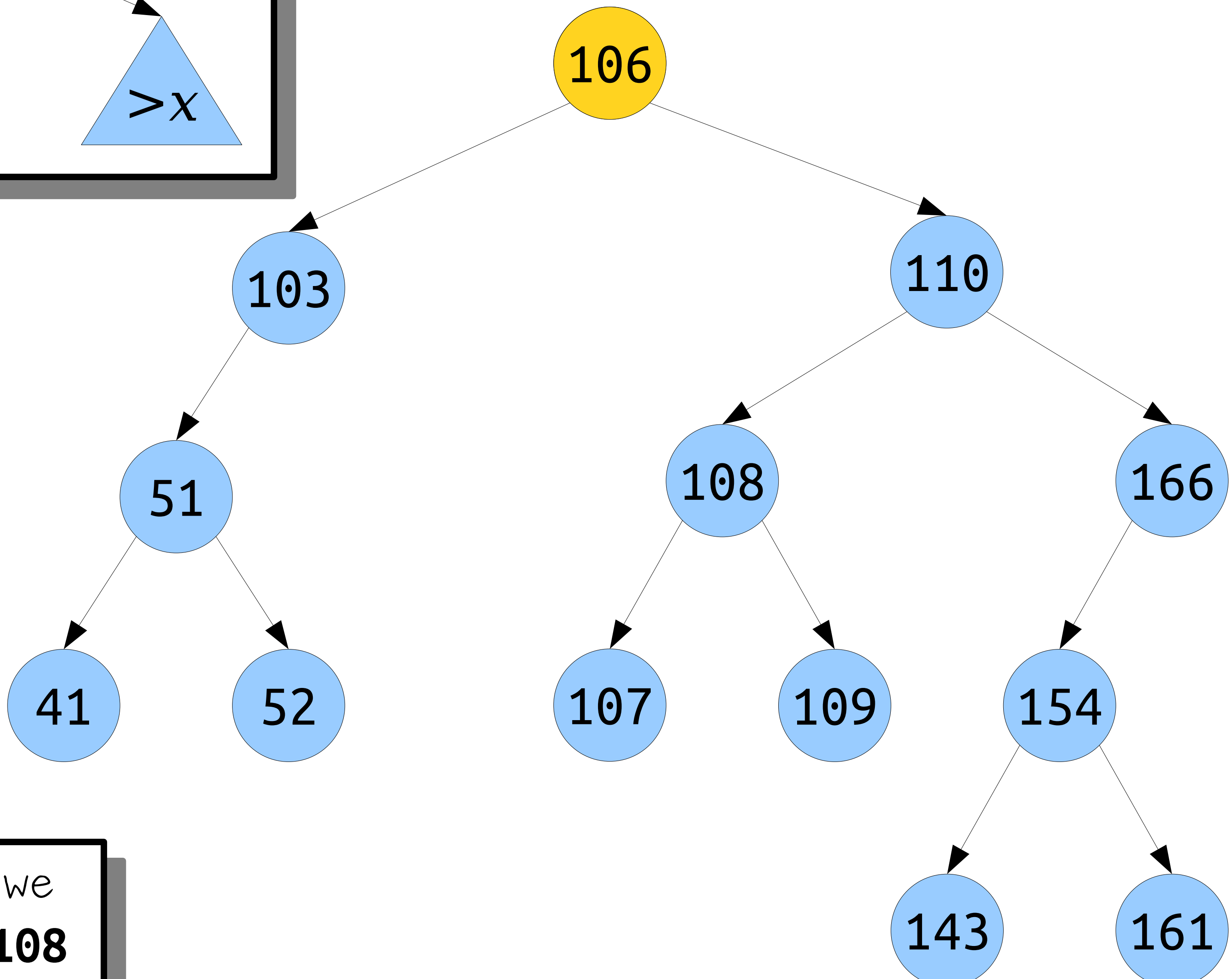

106
103
110
51
108
166
41
52
107
109
154
143
161



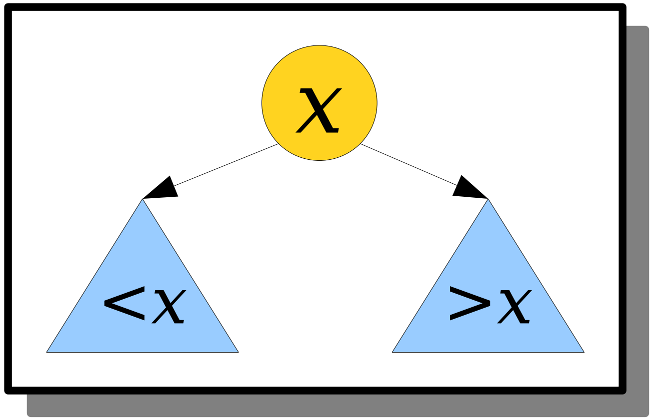


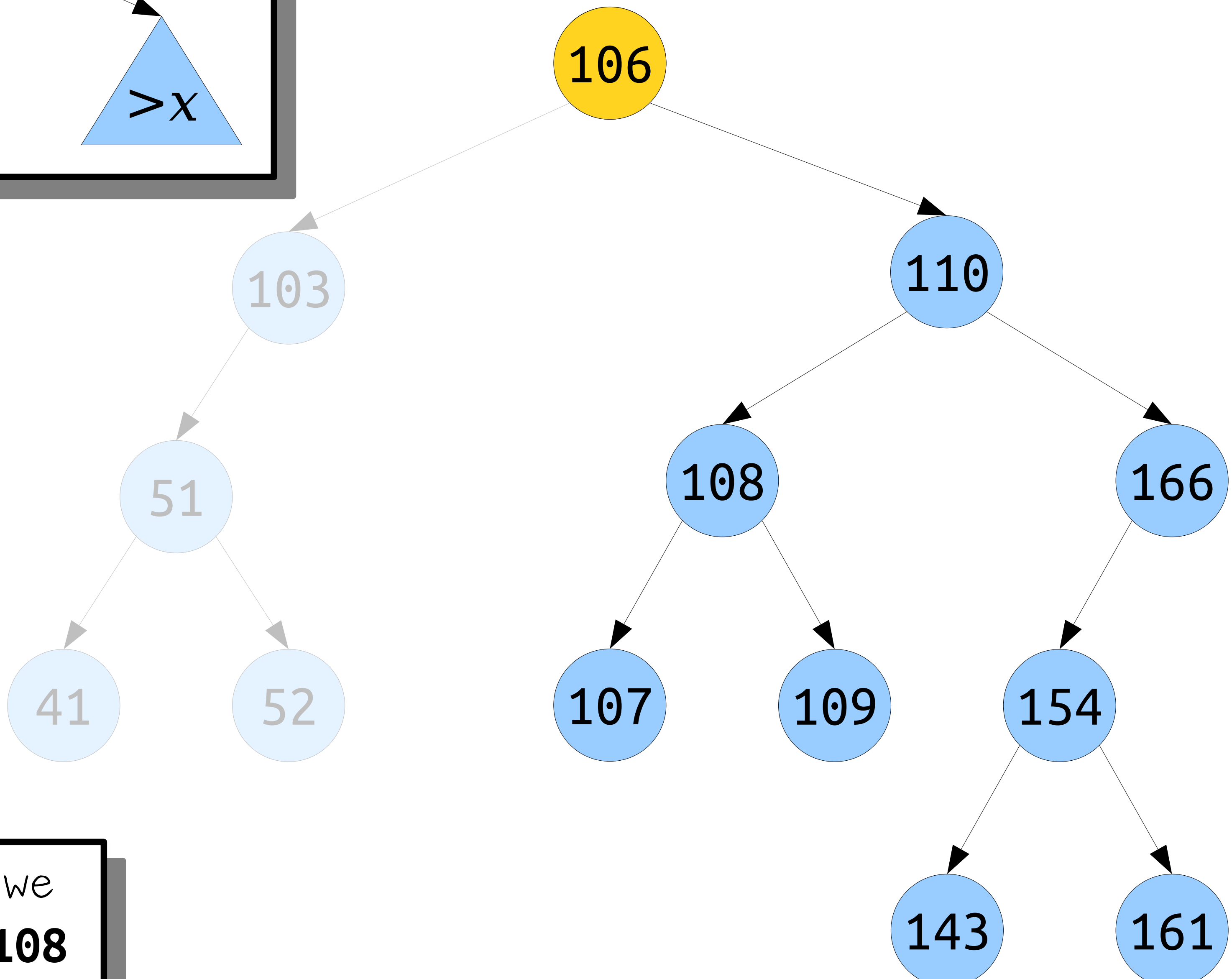
How can we check if **108** is in this tree?



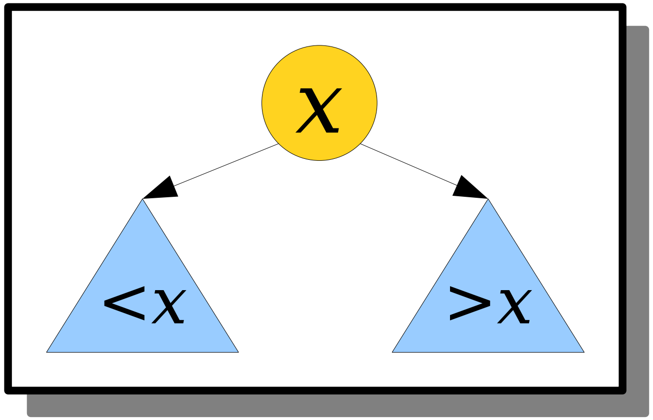


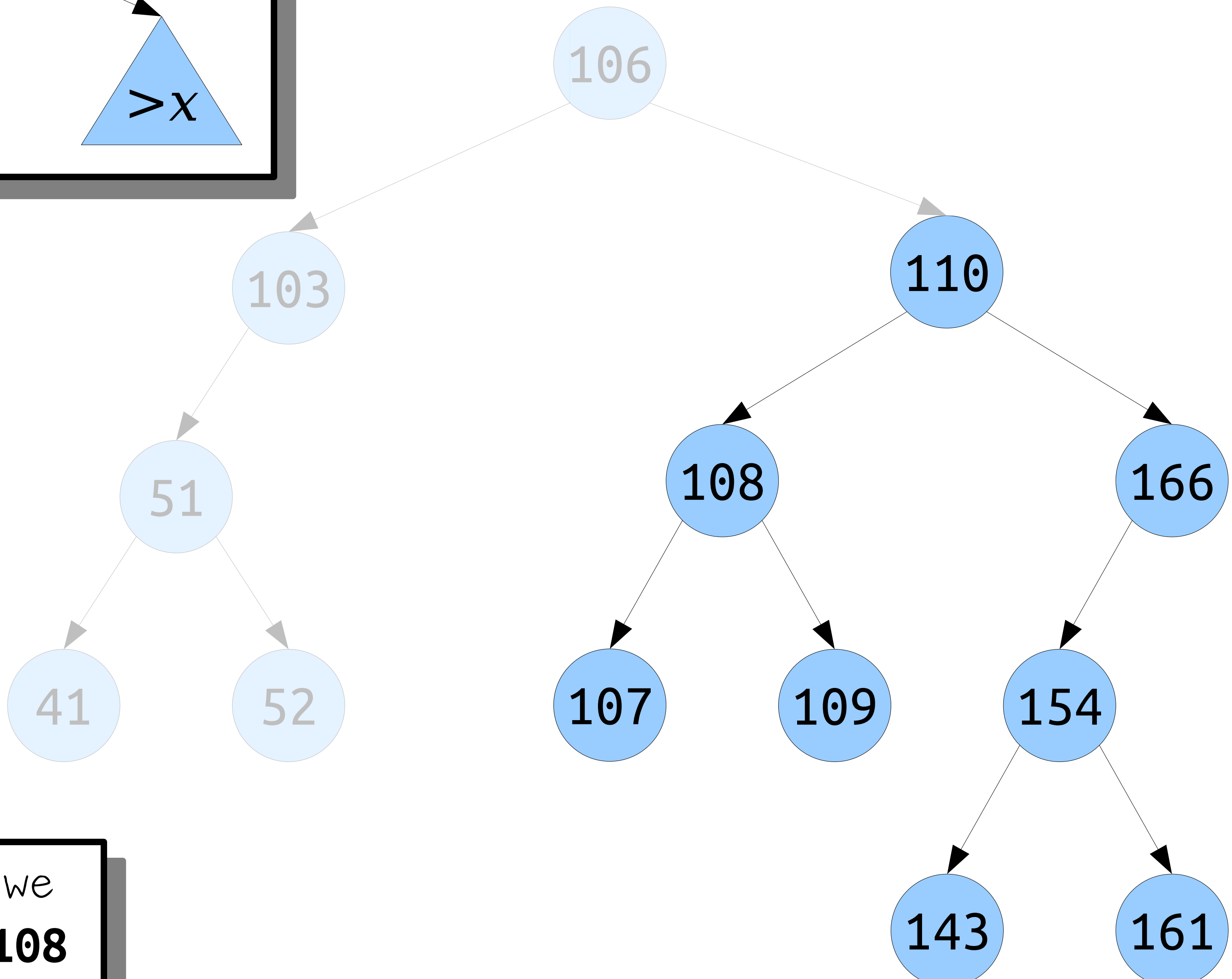
How can we check if **108** is in this tree?



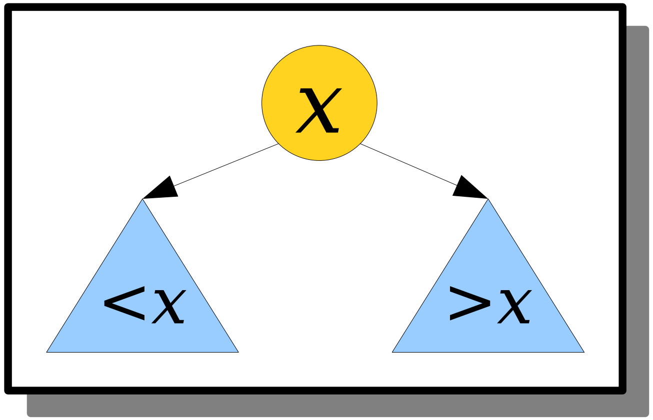


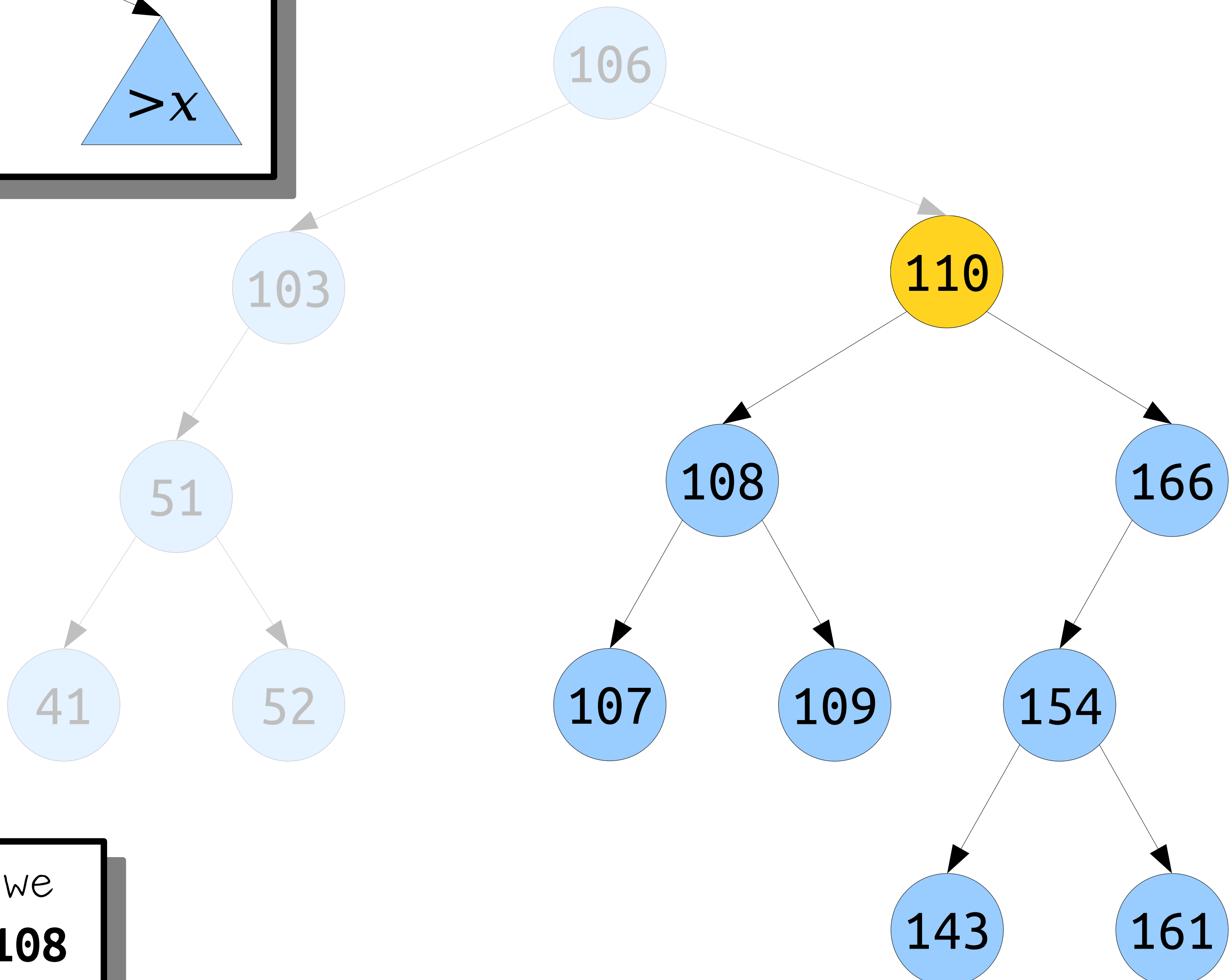
How can we check if **108** is in this tree?



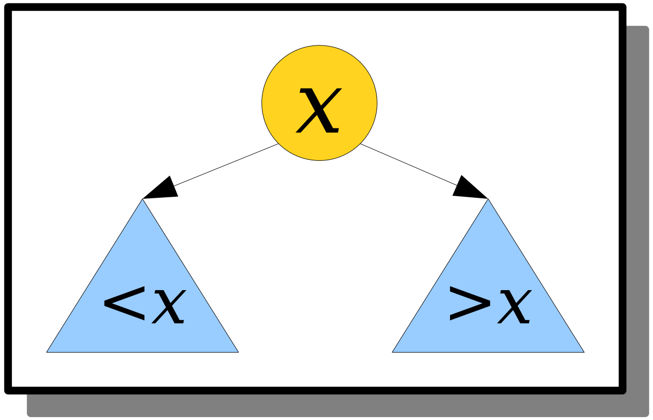


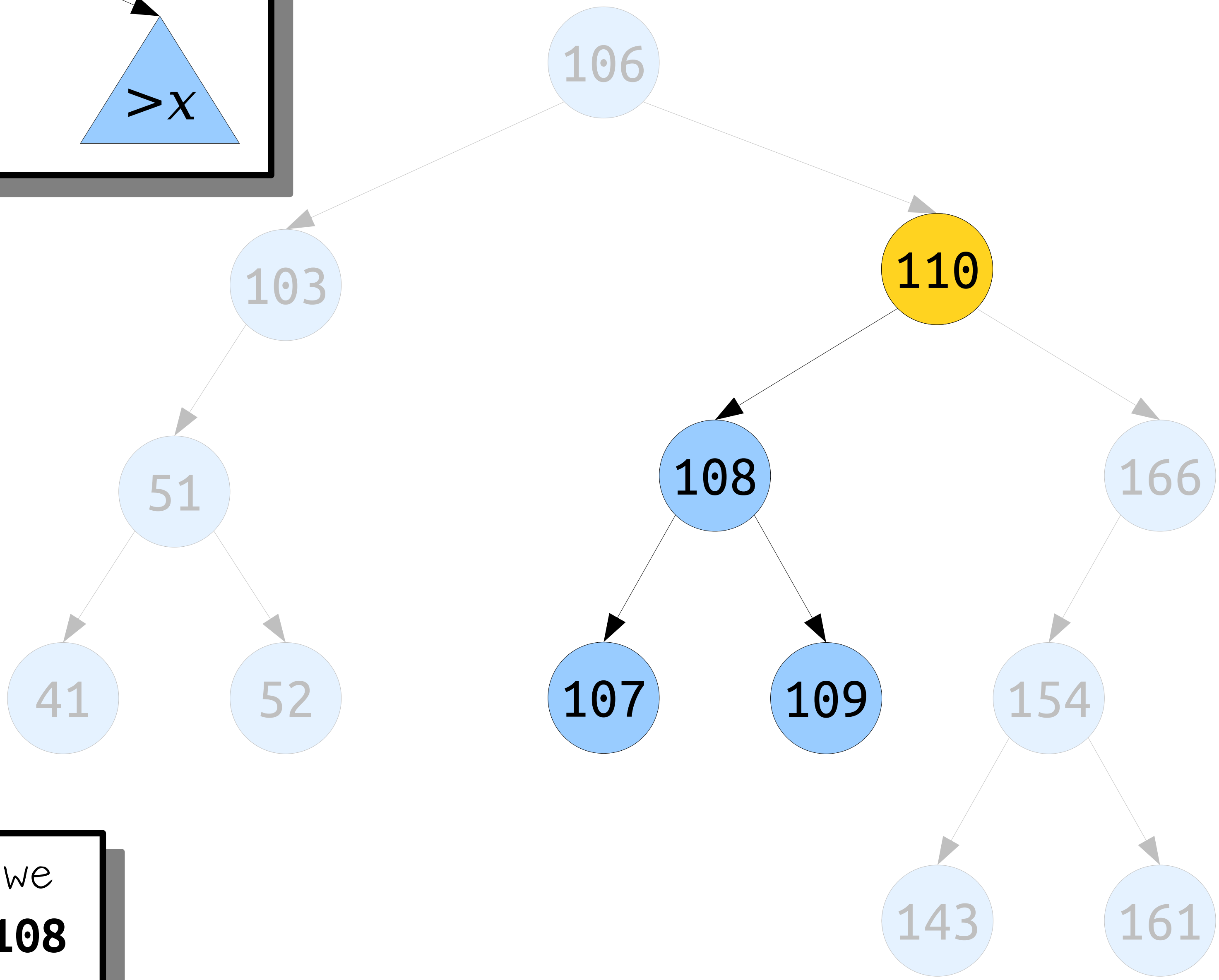
How can we check if **108** is in this tree?



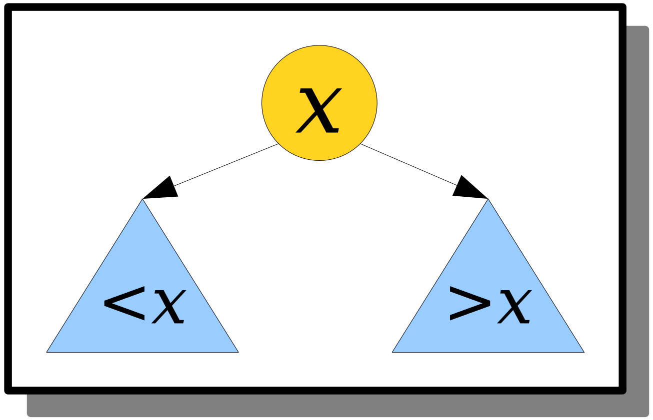


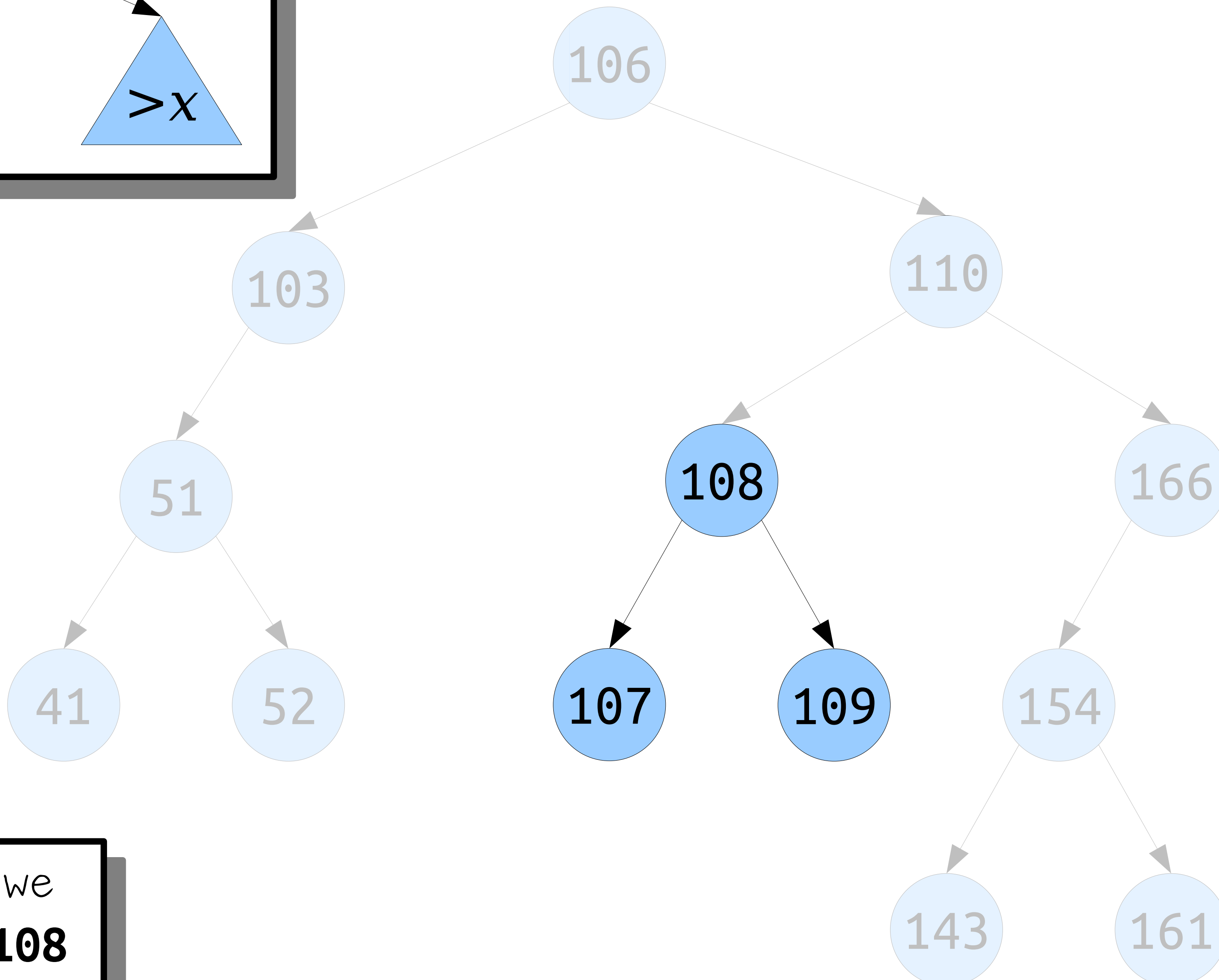
How can we check if **108** is in this tree?



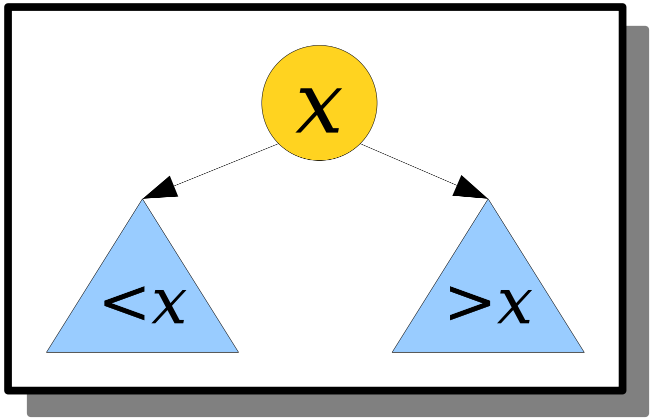


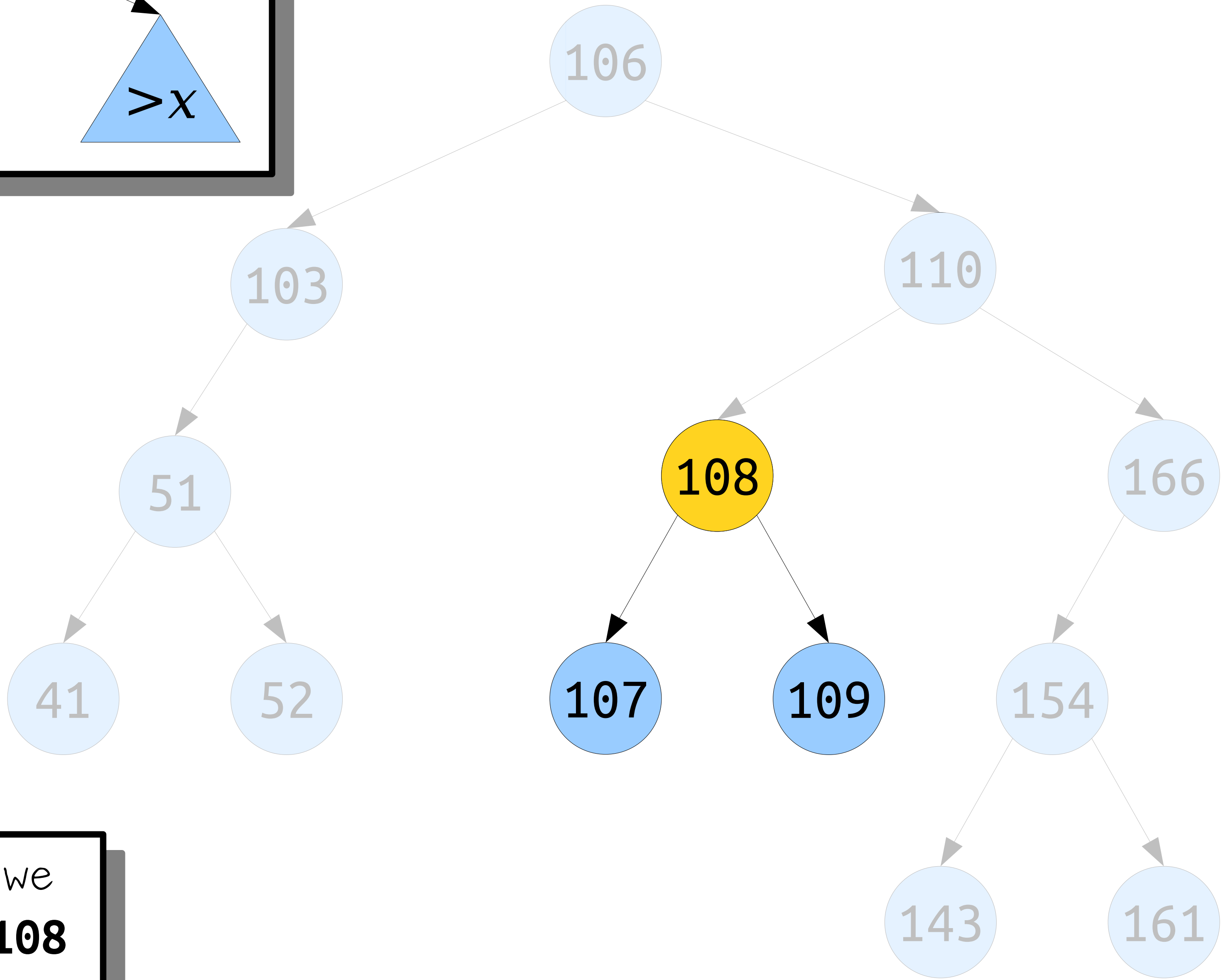
How can we check if **108** is in this tree?



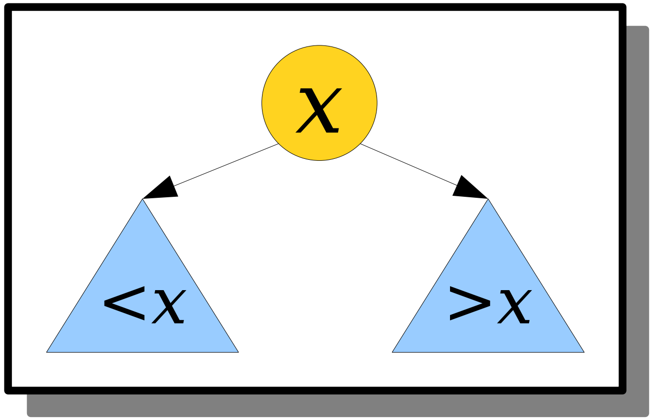


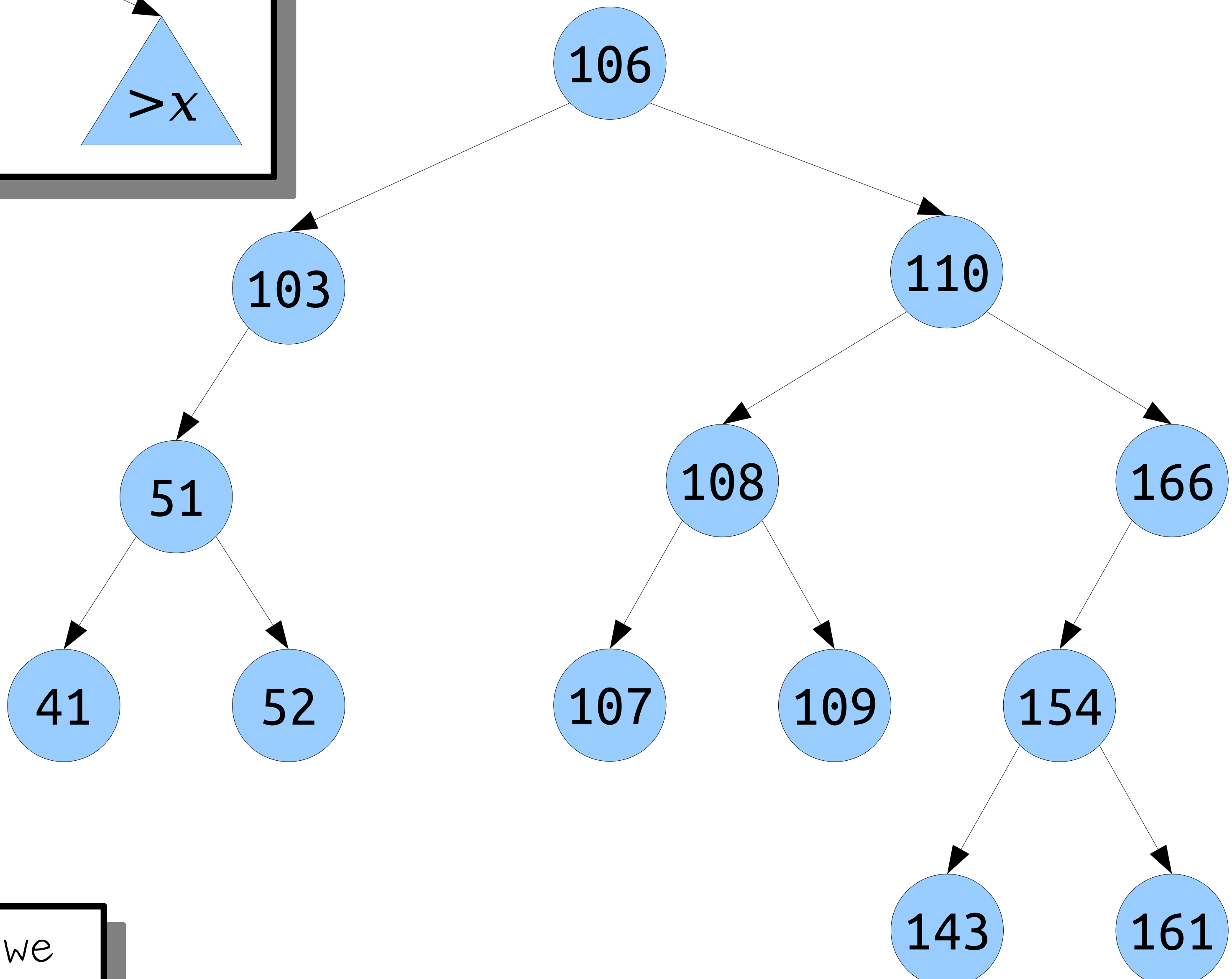
How can we check if **108** is in this tree?



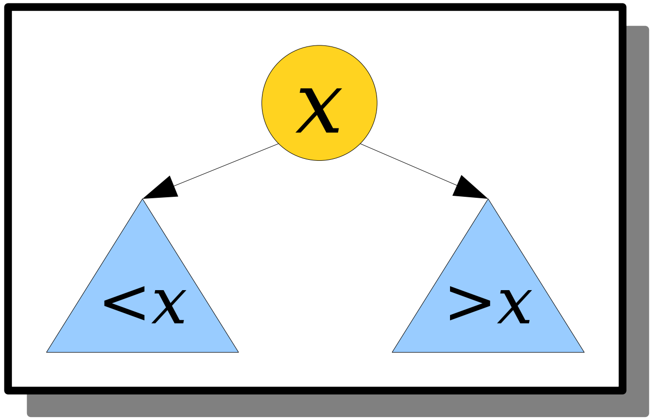


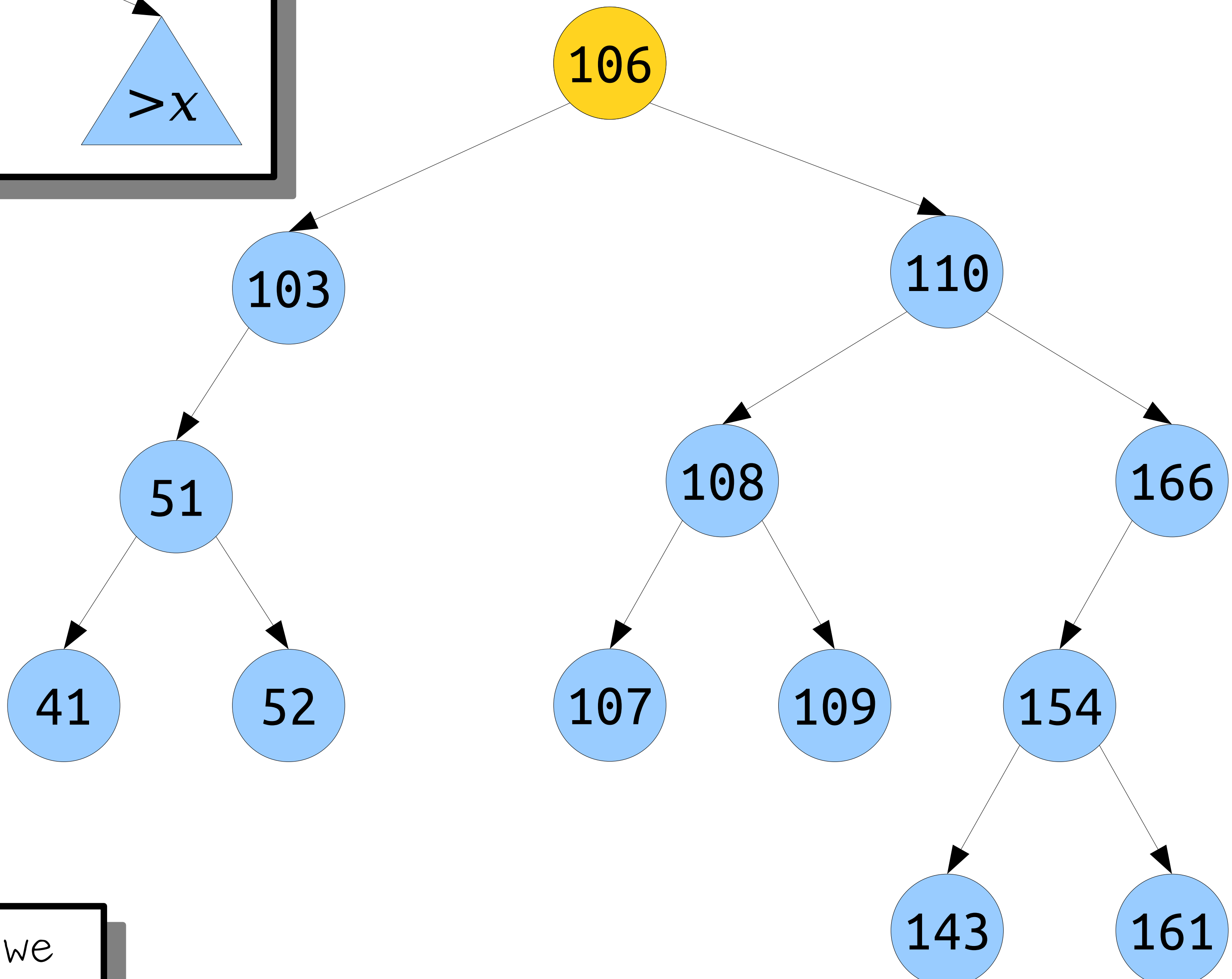
How can we check if **108** is in this tree?



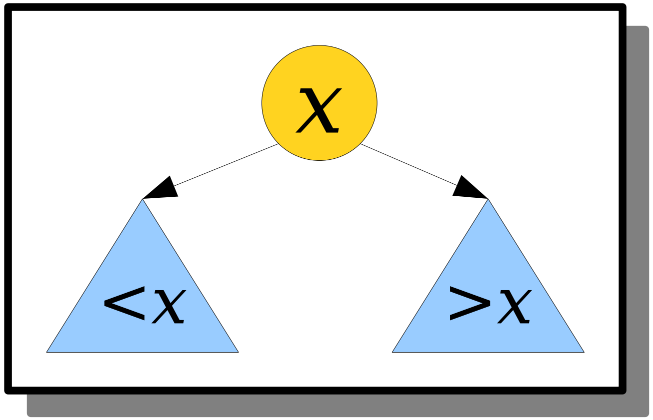


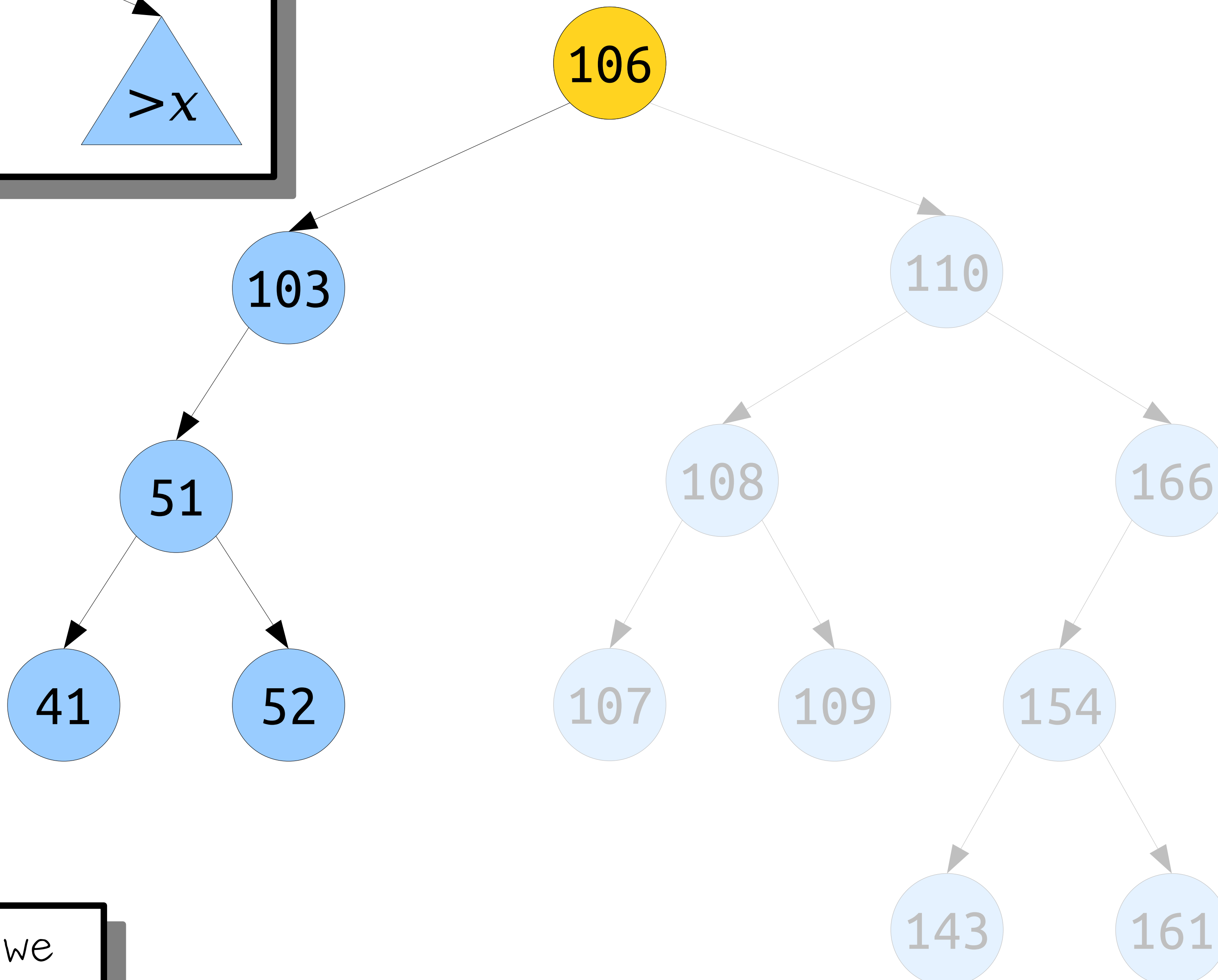
How can we check if **83** is in this tree?



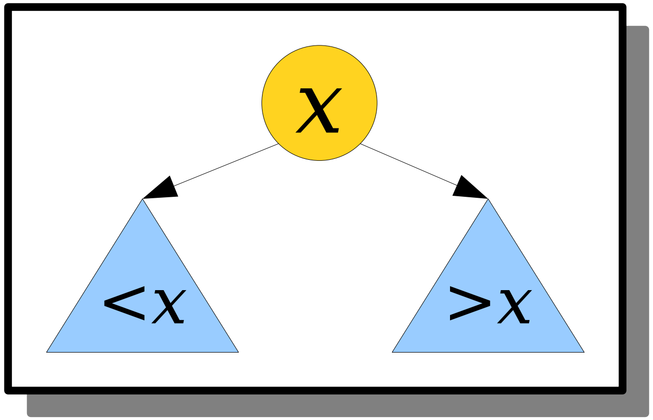


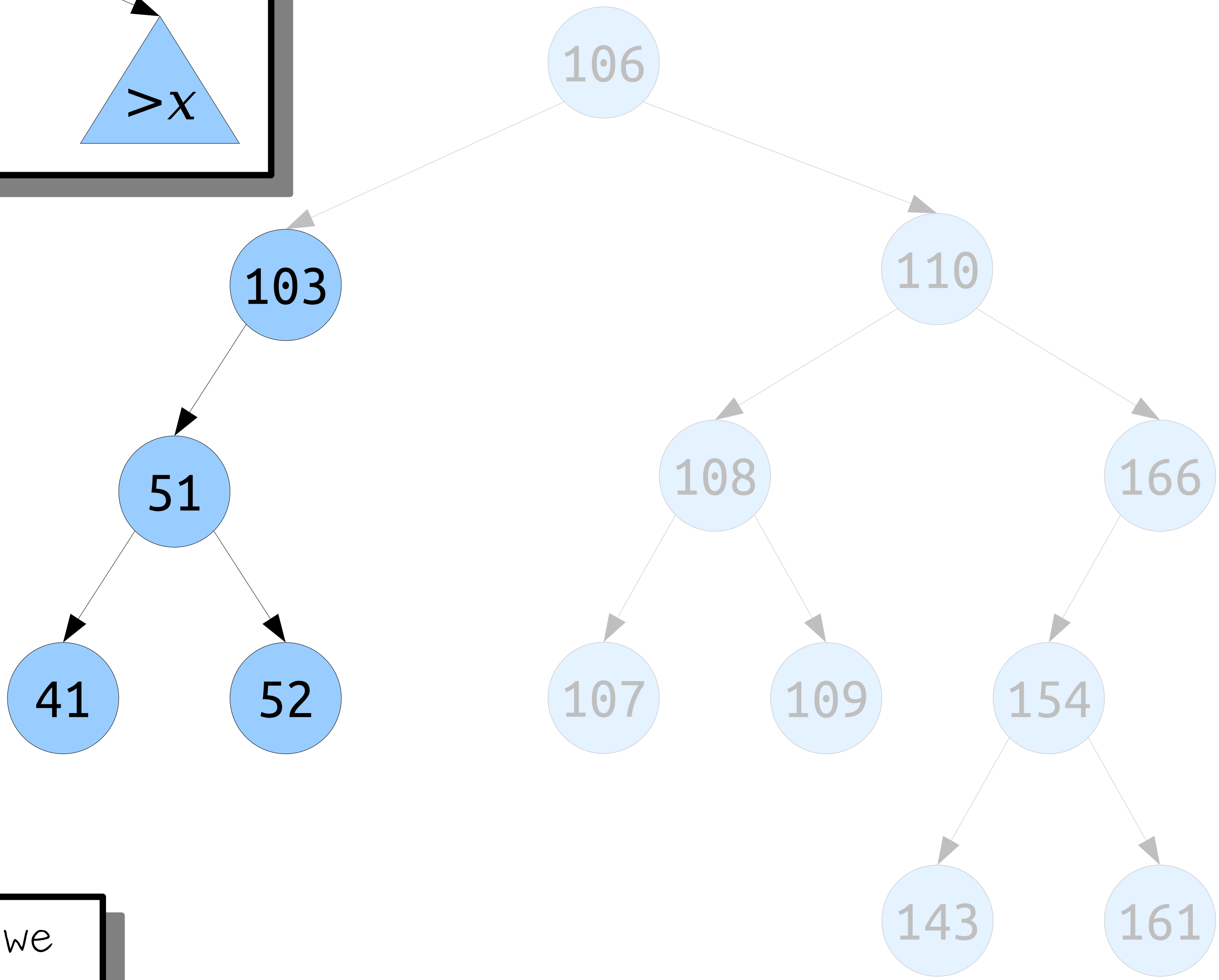
How can we check if **83** is in this tree?



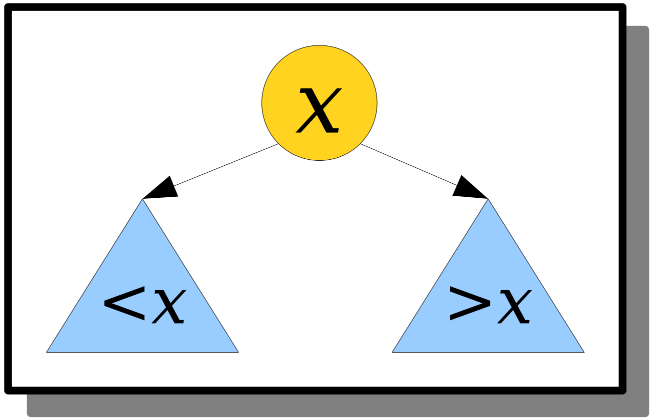


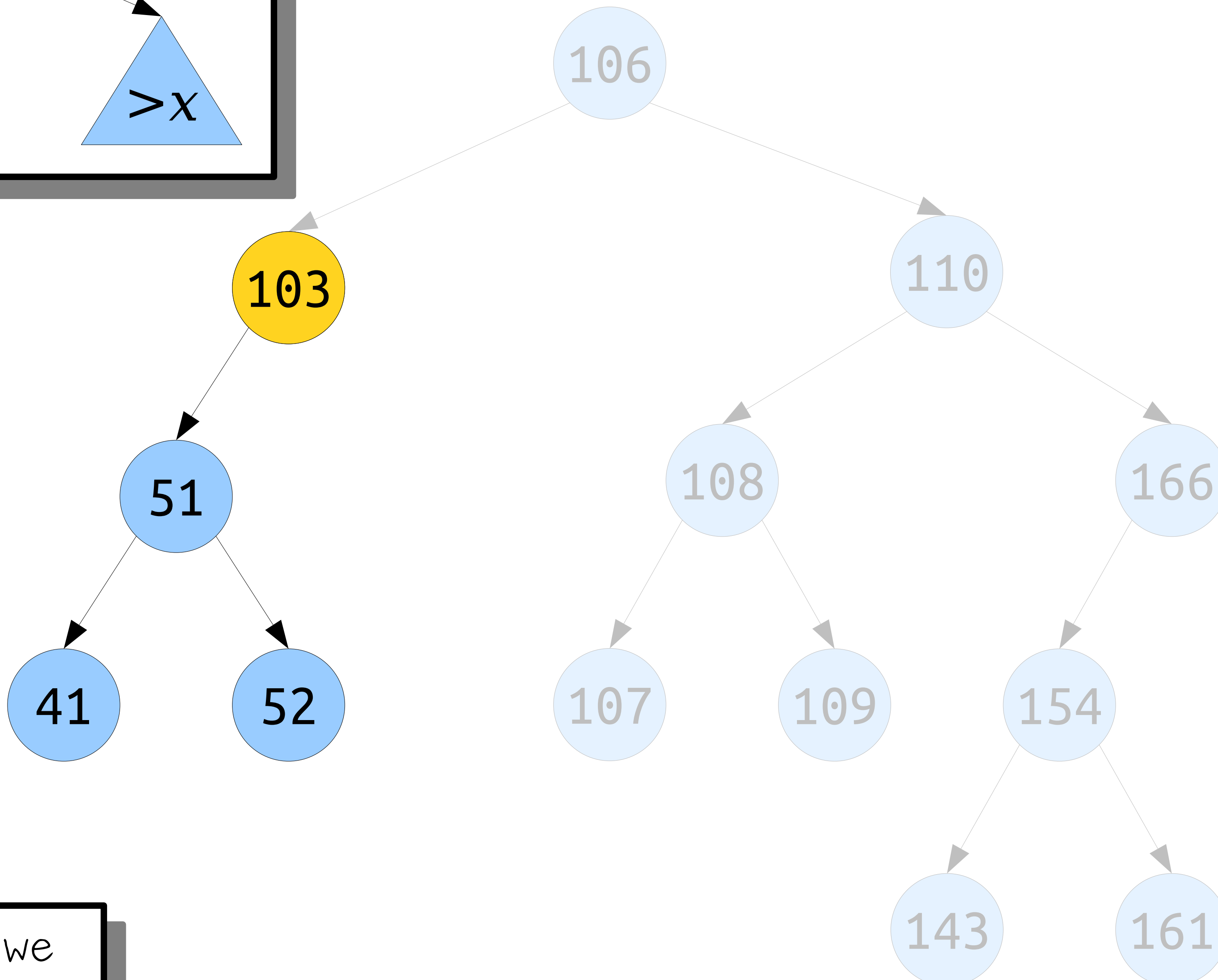
How can we check if **83** is in this tree?



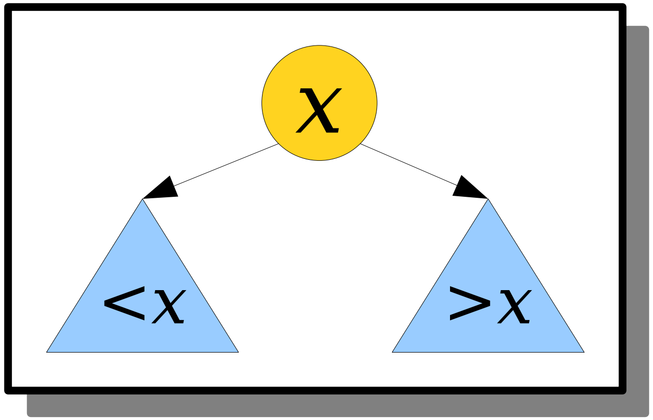


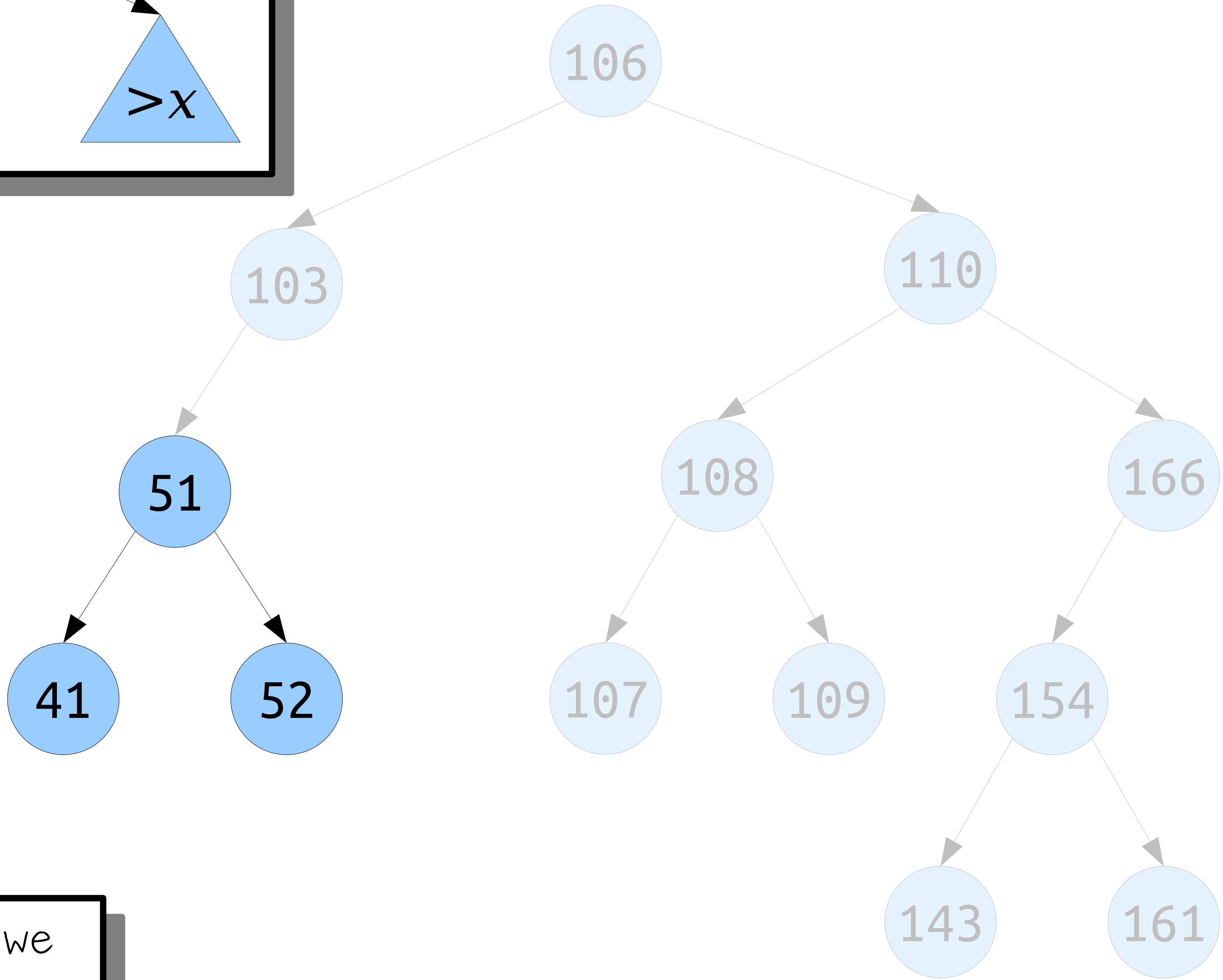
How can we check if **83** is in this tree?



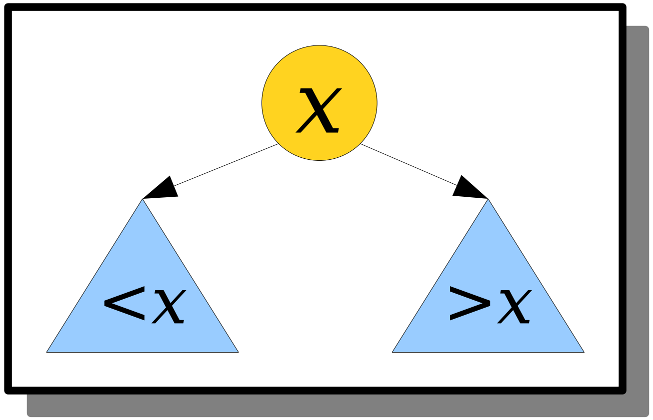


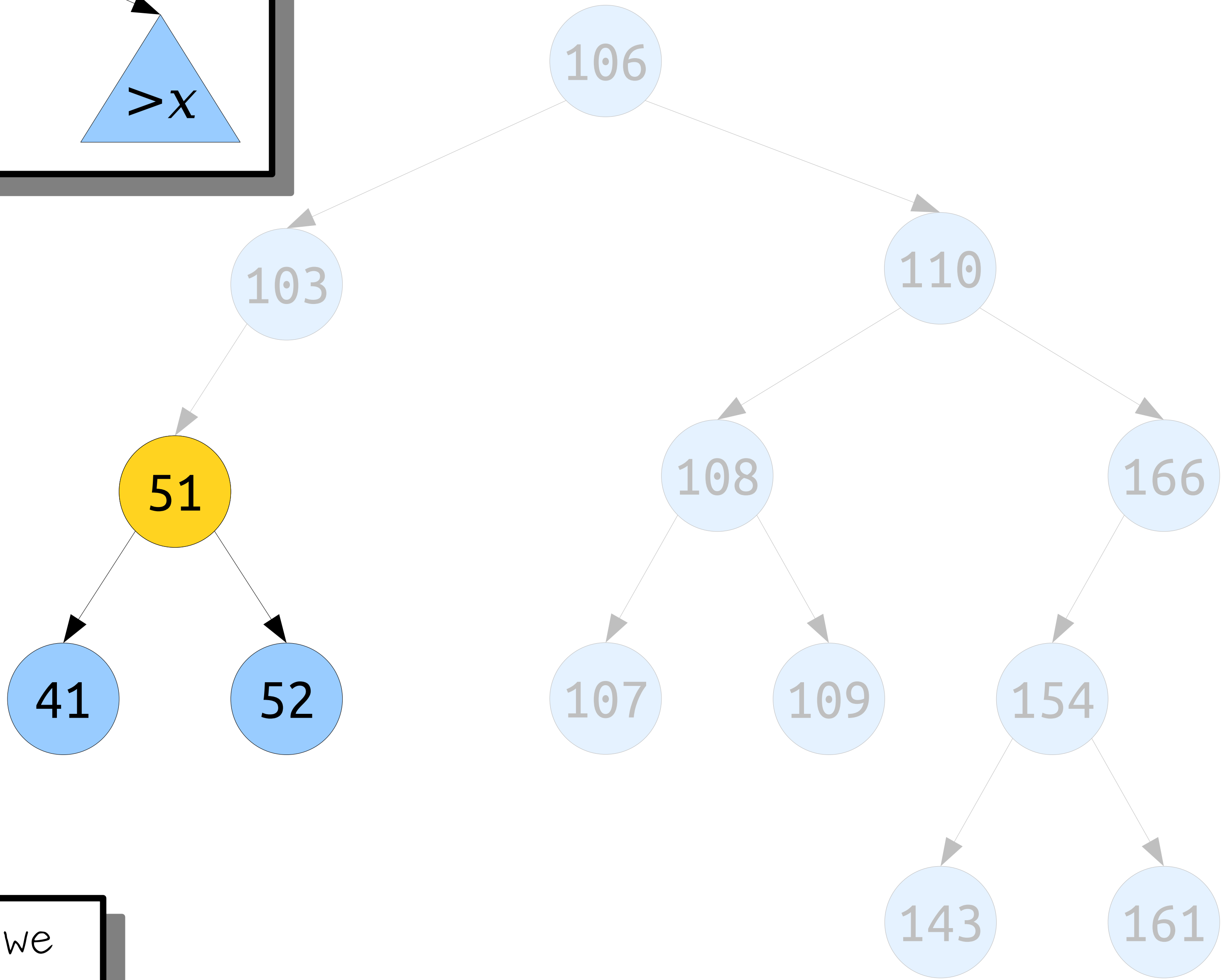
How can we check if **83** is in this tree?



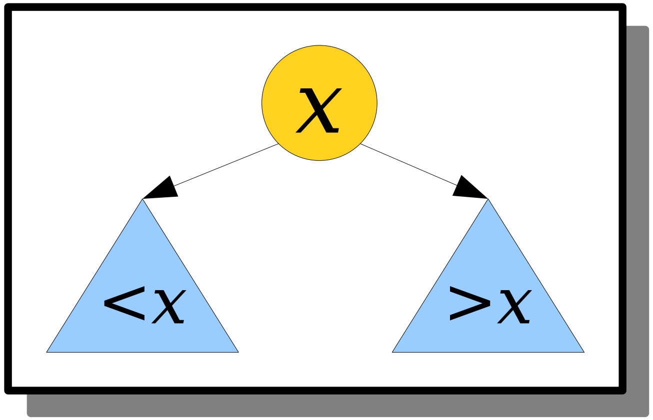


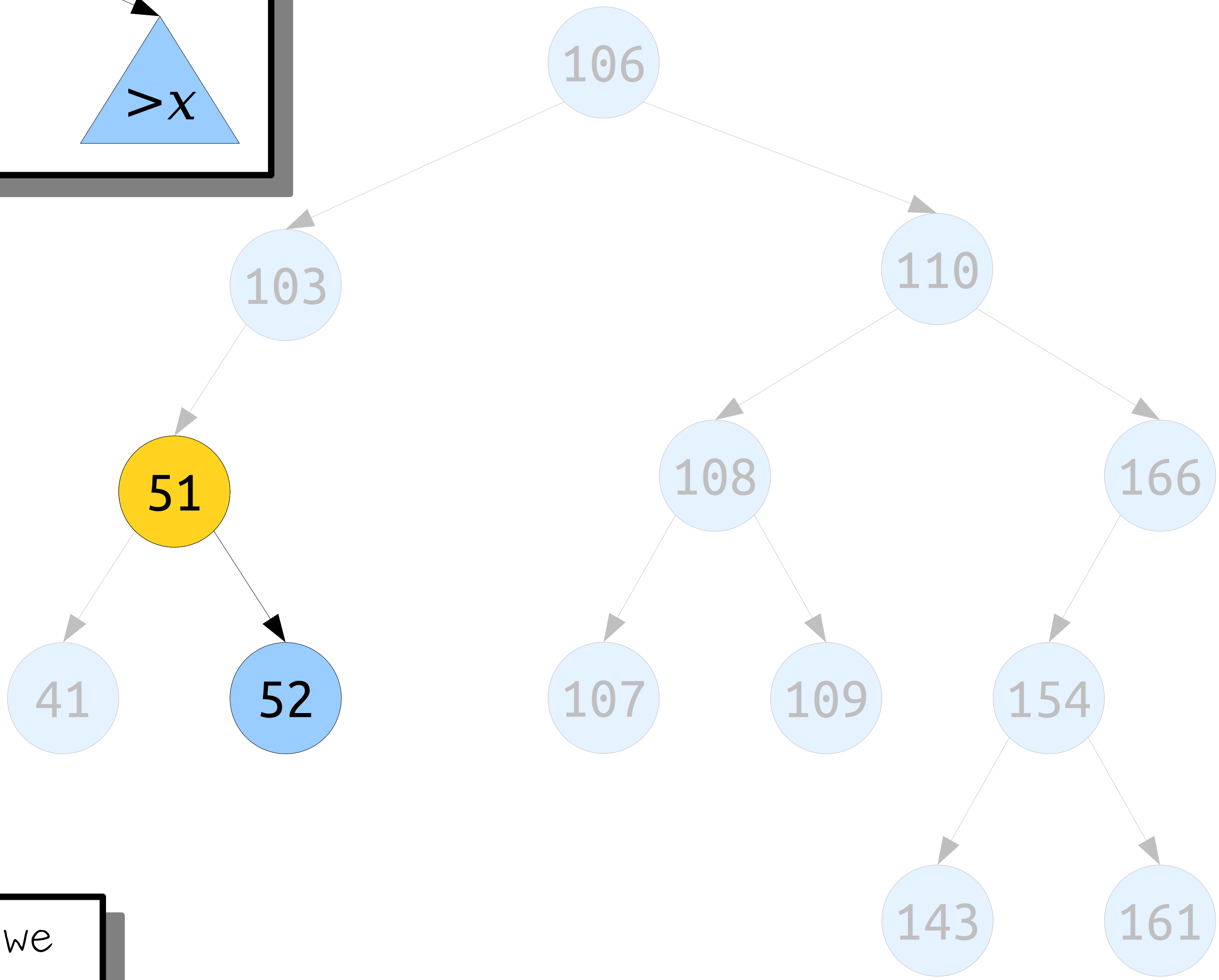
How can we check if **83** is in this tree?



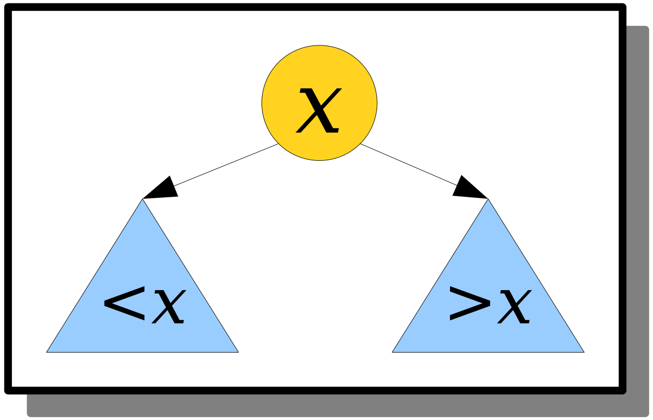


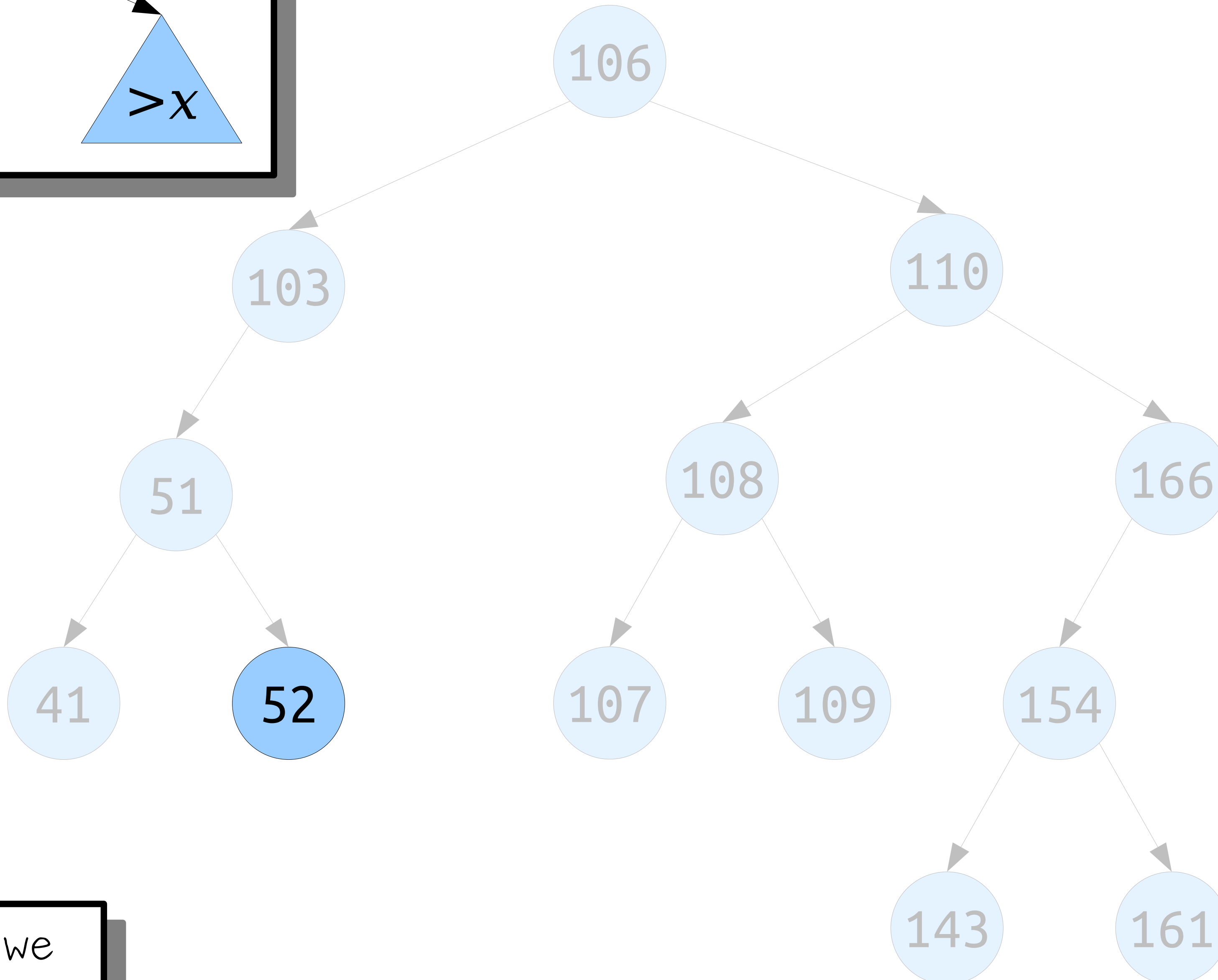
How can we check if **83** is in this tree?



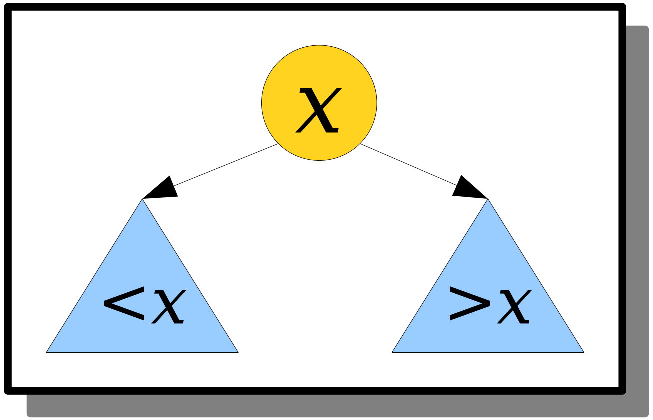


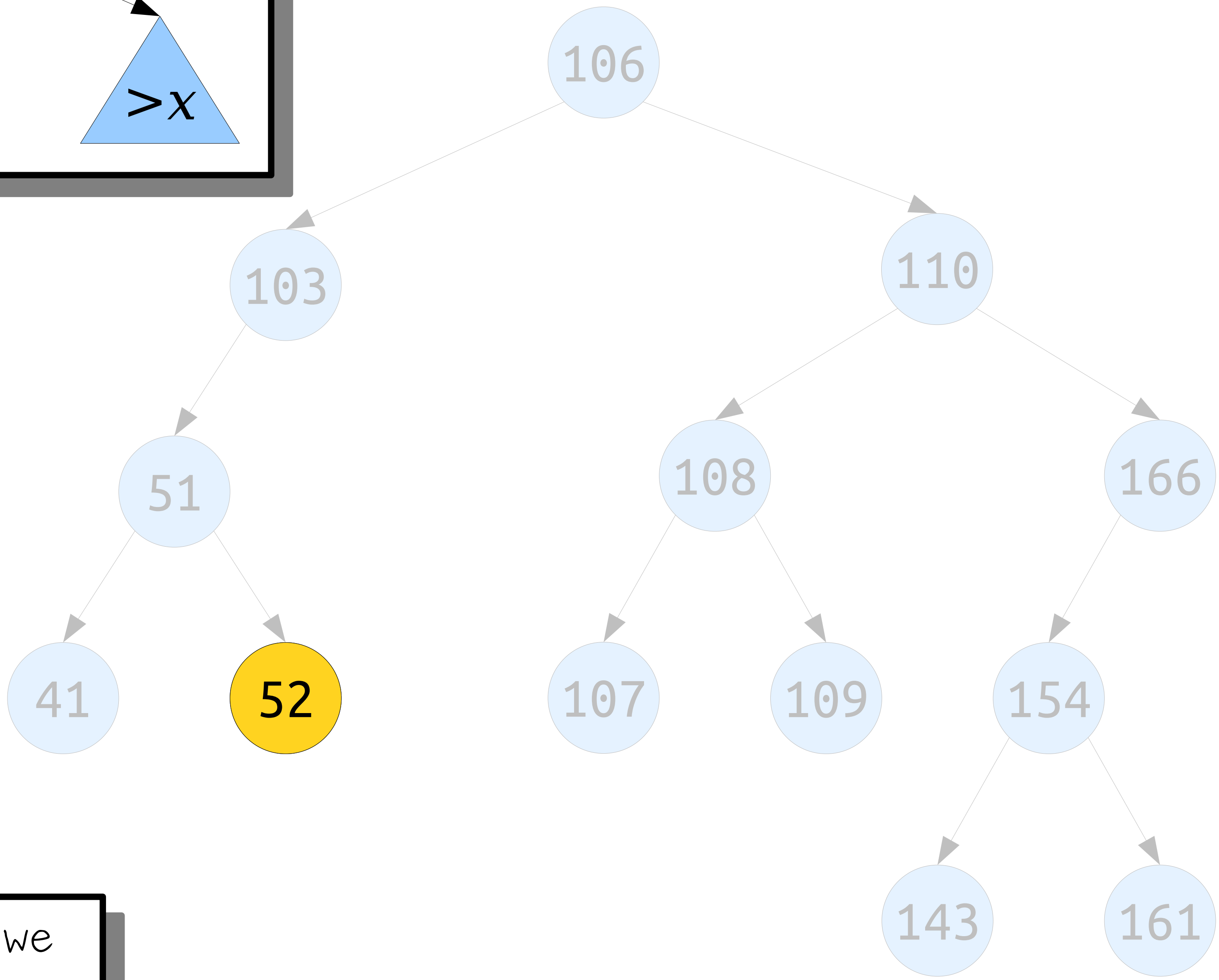
How can we check if **83** is in this tree?



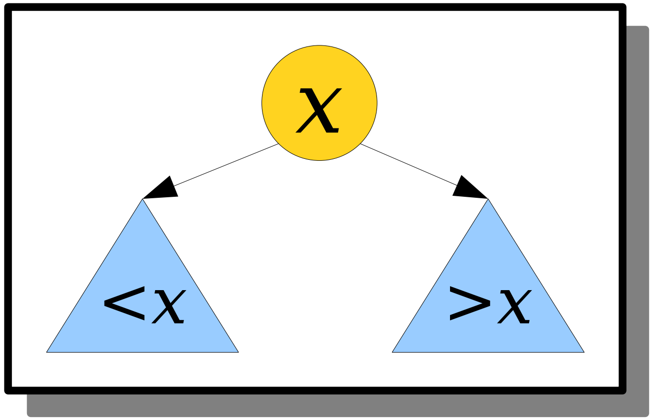


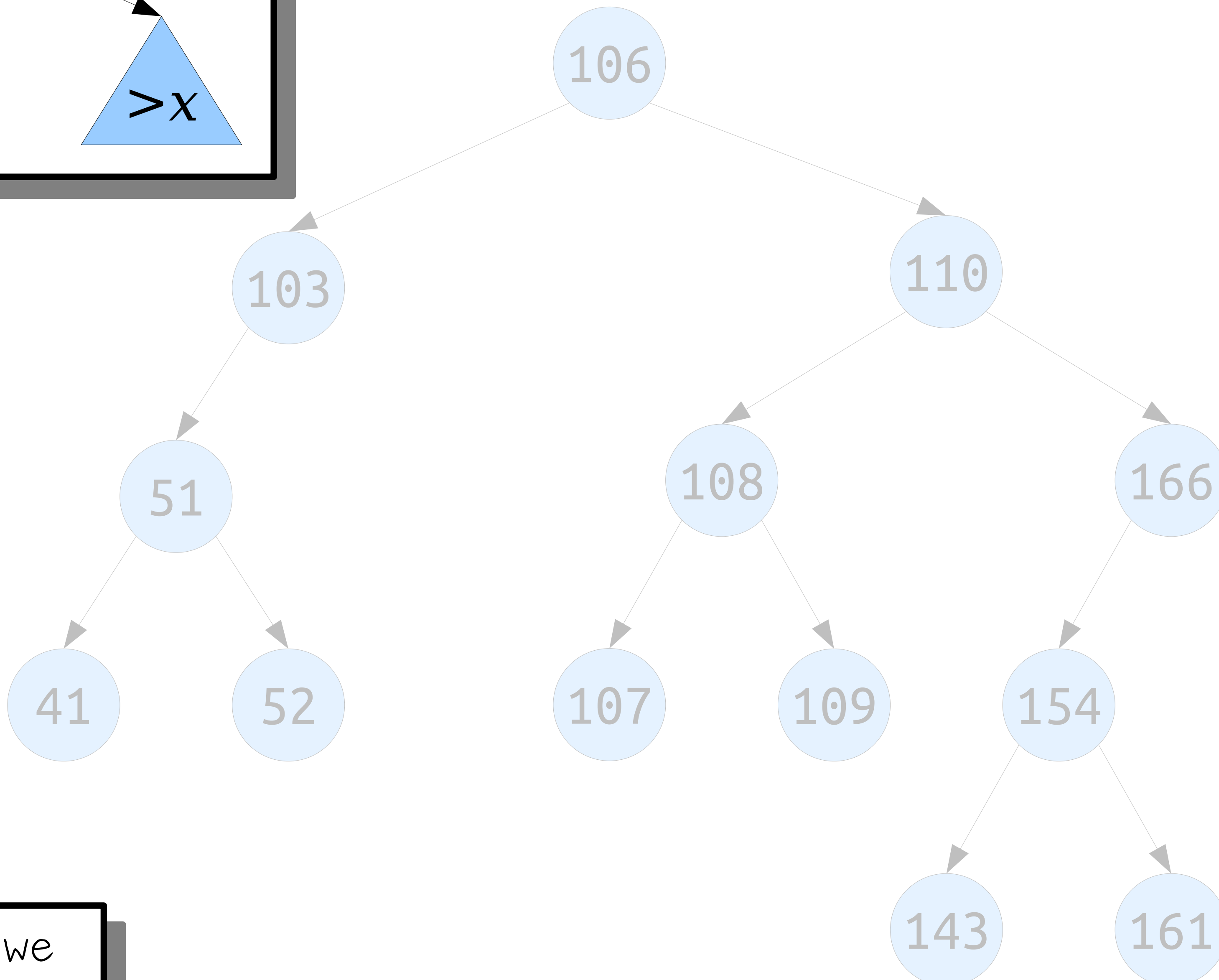
How can we check if **83** is in this tree?





How can we check if **83** is in this tree?

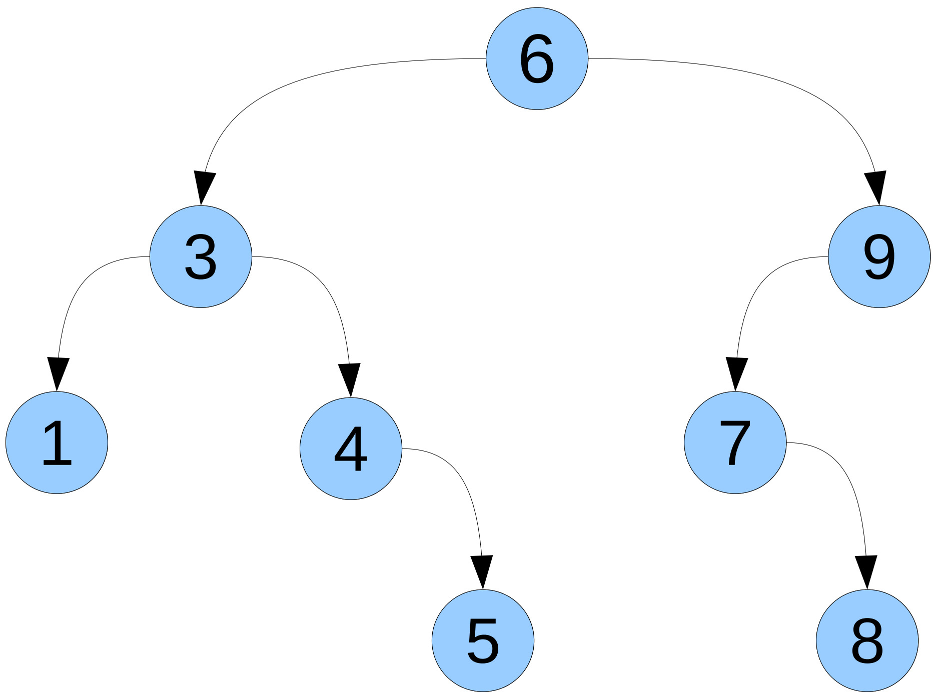




How can we check if **83** is in this tree?

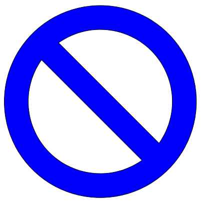
Binary Search Trees

- The data structure we have just seen is called a ***binary search tree*** (or ***BST***).
- The tree consists of a number of ***nodes***, each of which stores a value and has zero, one, or two ***children***.
- All values in a node's left subtree are ***smaller*** than the node's value, and all values in a node's right subtree are ***greater*** than the node's value.



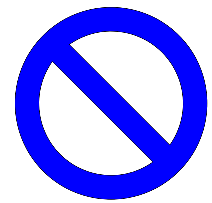
A Binary Search Tree Is Either…

an empty tree,
represented by
`nullptr`

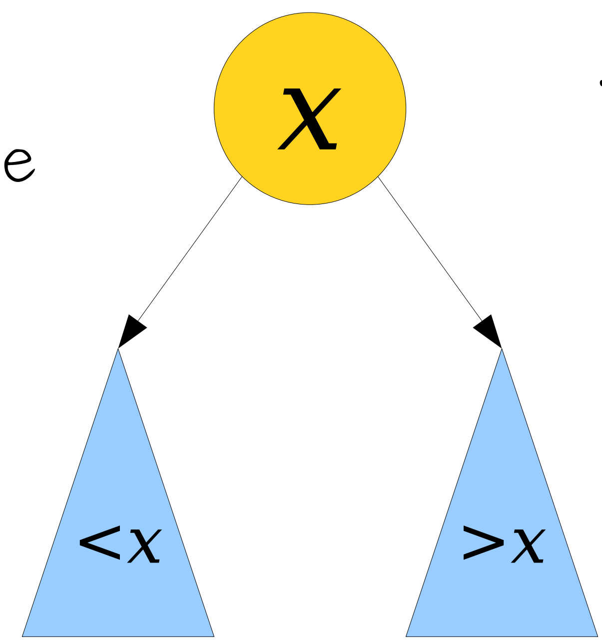


A Binary Search Tree Is Either…

an empty tree, represented by **nullptr**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.

Binary Search Tree Nodes

```
struct Node {
    Type value;
    Node* left;  // Smaller values
    Node* right; // Bigger values
};
```

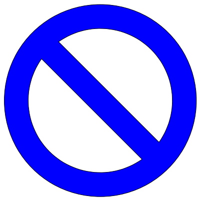
Kinda like a linked list, but with two pointers instead of just one!

Searching Trees

A Binary Search Tree Is Either…

an empty tree,
represented by

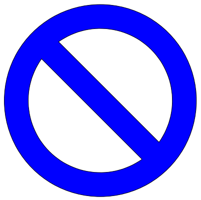
`nullptr`



A Binary Search Tree Is Either…

an empty tree,
represented by

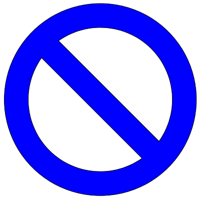
`nullptr`



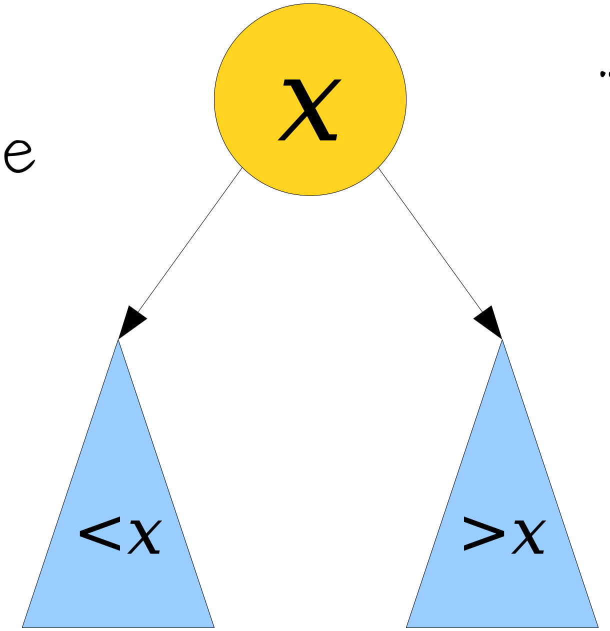
If you're looking for something in an empty BST, it's not there! Sorry.

A Binary Search Tree Is Either…

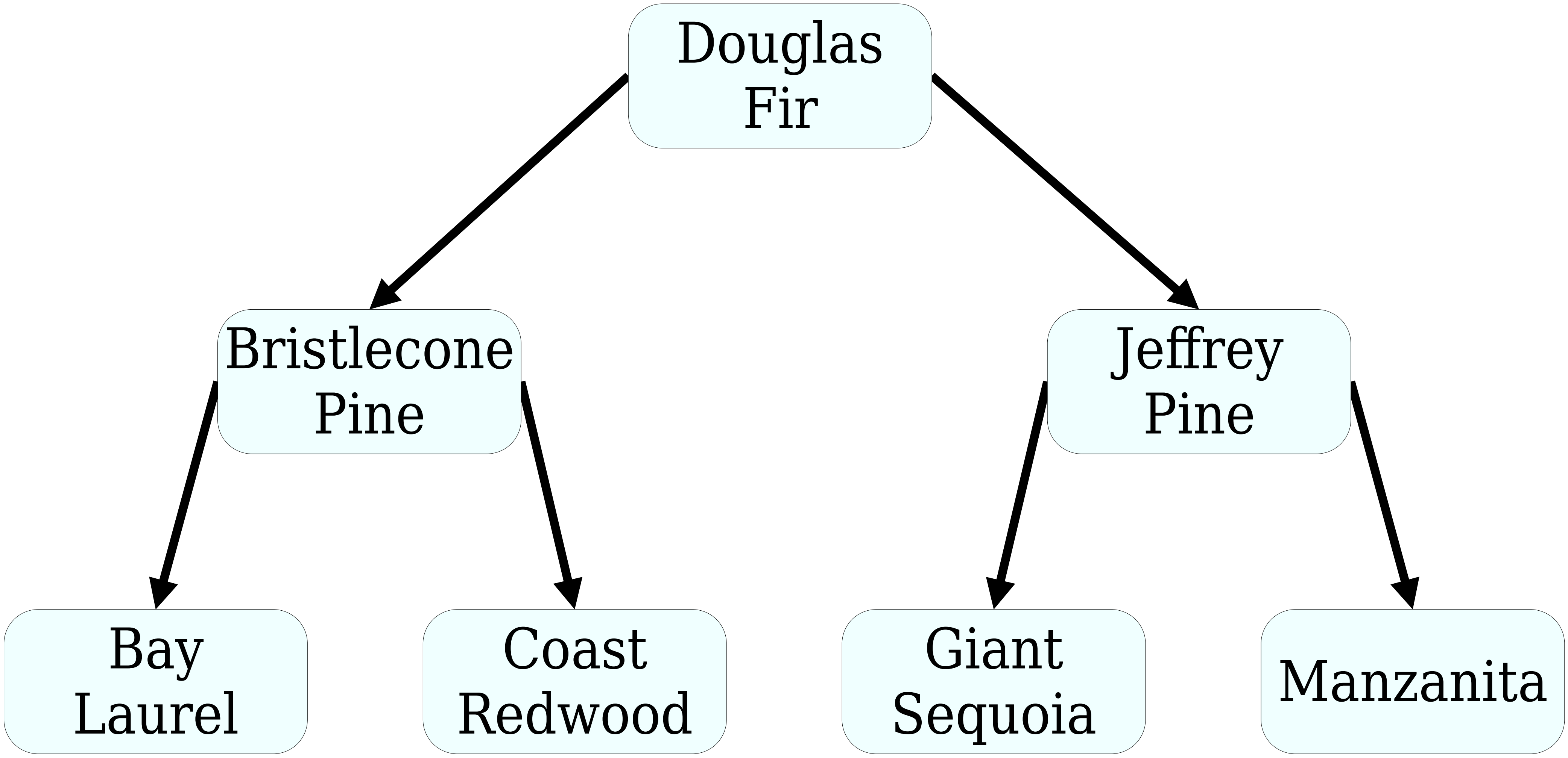
an empty tree, represented by **nullptr**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.

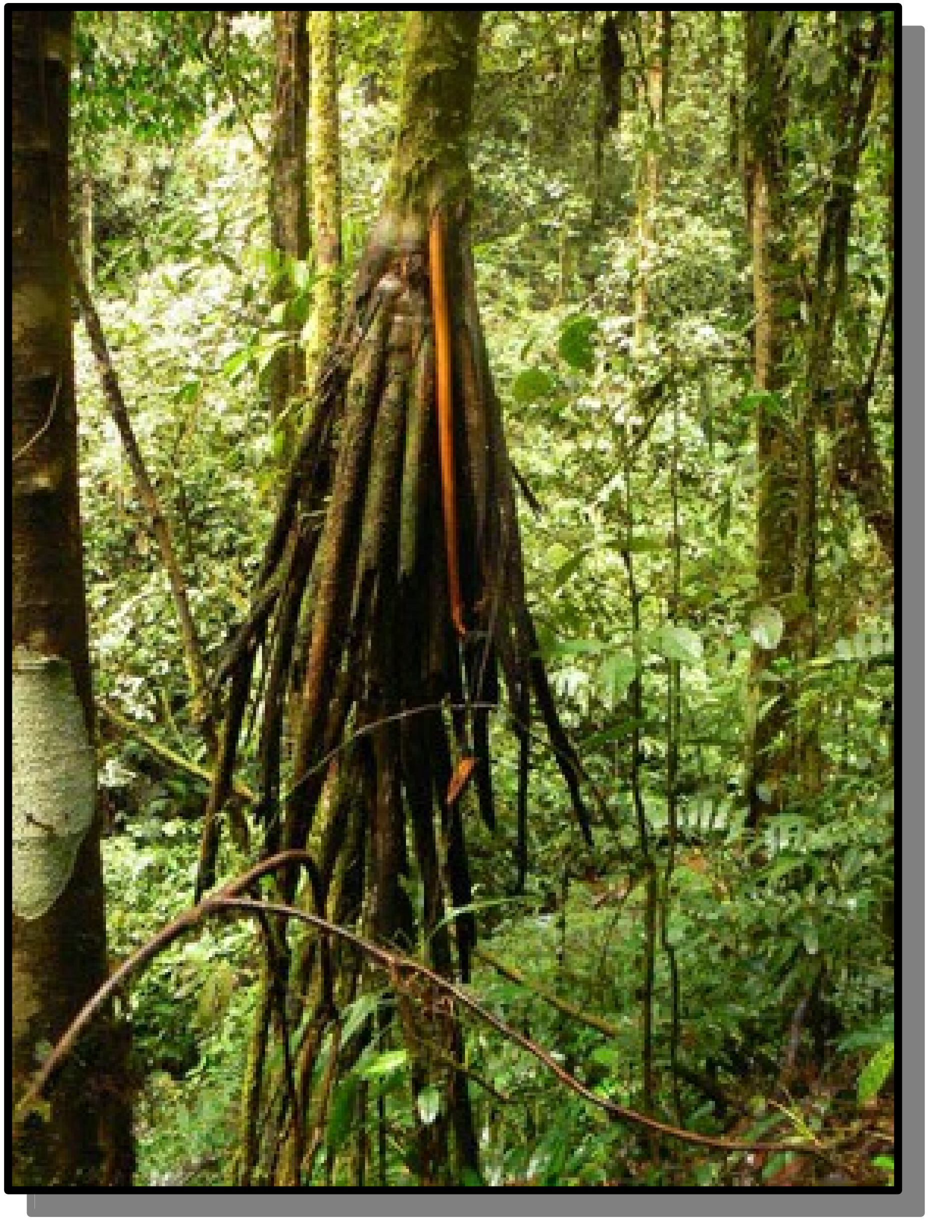


Douglas Fir
Bristlecone Pine
Jeffrey Pine
Bay Laurel
Coast Redwood
Giant Sequoia
Manzanita

Good exercise:

Rewrite this function iteratively!

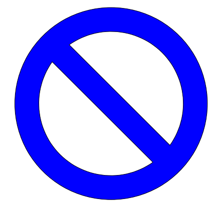
Walking Trees



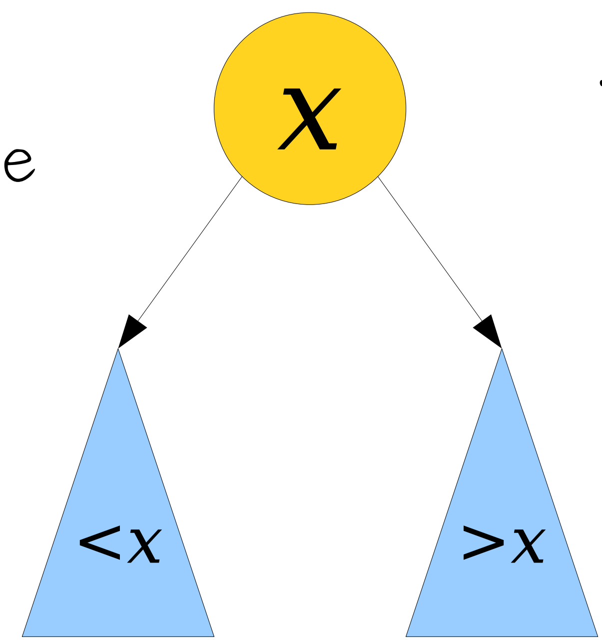
Print all the values in a BST,
in sorted order.

A Binary Search Tree Is Either…

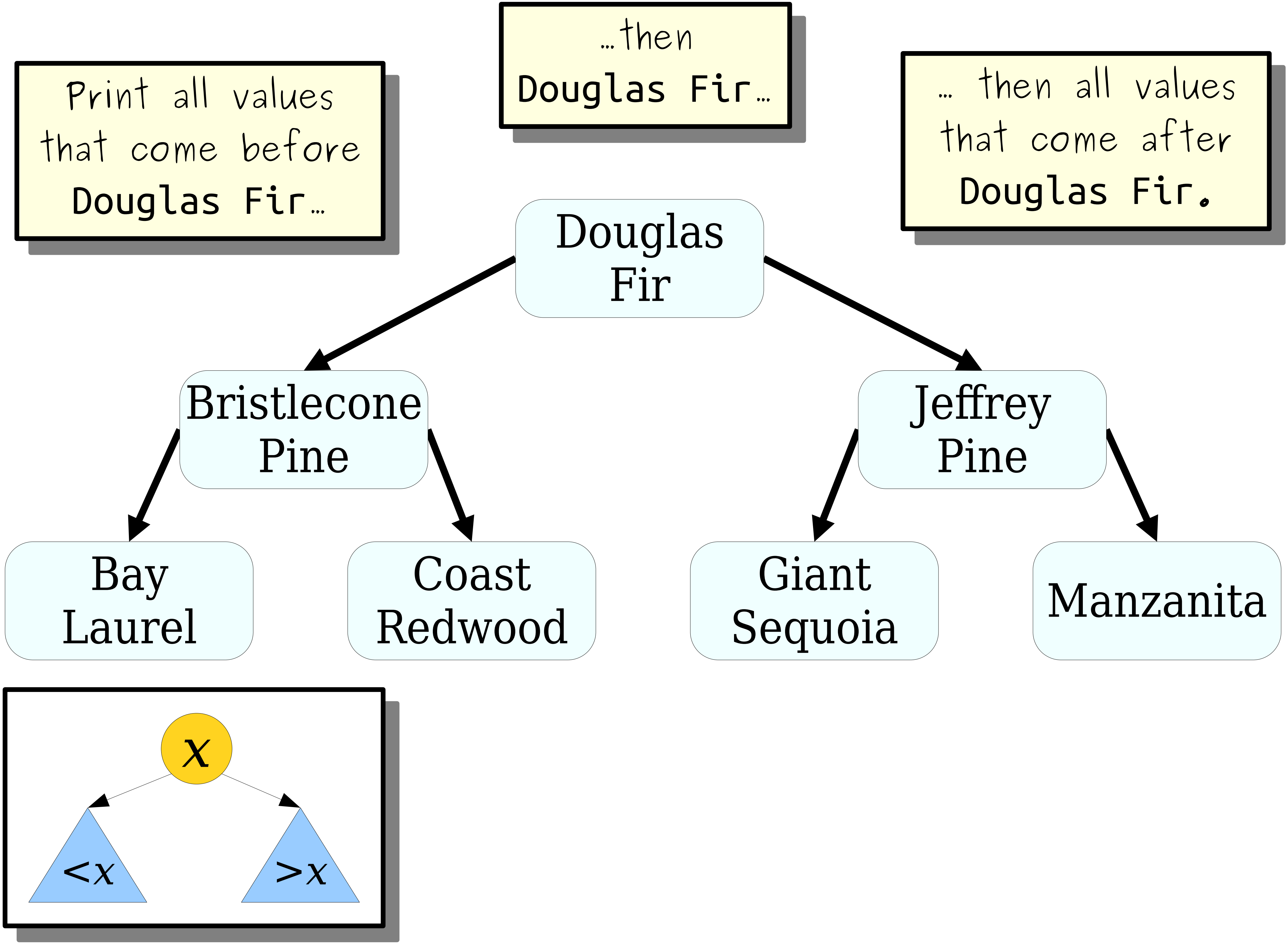
an empty tree, represented by **nullptr**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.



Print all values that come before Douglas Fir…
…then Douglas Fir…
… then all values that come after Douglas Fir.
Douglas Fir
Bristlecone Pine
Jeffrey Pine
Bay Laurel
Coast Redwood
Giant Sequoia
Manzanita
x
<x
>x

Inorder Traversals

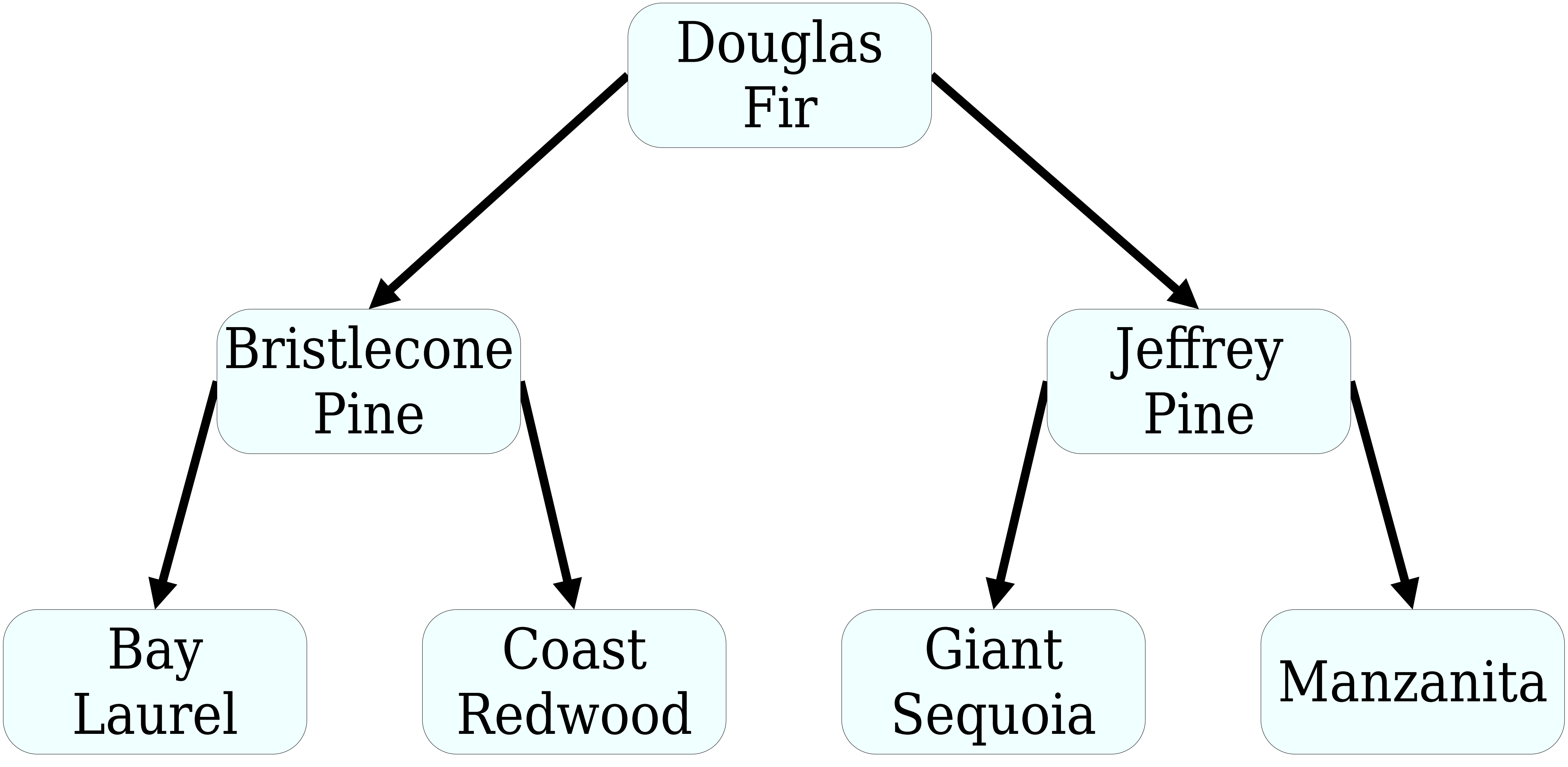
- The particular recursive pattern we just saw is called an ***inorder traversal*** of a binary tree.
- Specifically:
  - Recursively visit all the nodes in the left subtree.
  - Visit the node itself.
  - Recursively visit all the nodes in the right subtree.

What will happen if we swap these two lines?

Formulate a hypothesis, but ***don’t post anything in chat just yet***.

What will happen if we swap these two lines?

Now, ***post your best guess in chat***. Not sure? Just answer with “??.”


Douglas Fir
Bristlecone Pine
Jeffrey Pine
Bay Laurel
Coast Redwood
Giant Sequoia
Manzanita

Challenge problem:
Rewrite this function iteratively!

Time-Out for Announcements!

Assignment 8

- Assignment 7 was due today at the start of class.
  - Grace period ends this Sunday at 11:30AM Pacific time.
- Assignment 8 (***The Adventures of Links***) goes out today. It's due next Friday at the normal time.
  - Use the debugger to explore memory and escape from a maze!
  - Use linked lists to manipulate DNA sequences!

Second Midterm Logistics

- Our second midterm exam is next week.
- It’ll be a 48-hour take home exam that goes out Friday, March 12th at 12:30PM and comes due Sunday, March 14th at 12:30PM.
- Topic coverage is as follows:
  - The main focus will be Assignment 4 – 7 and Lectures 10 – 18 (backtracking through hashing).
  - Content from Assignment 8 and Lectures 19 – 25 are also fair game, but will not be emphasized as much.

Second Midterm Logistics

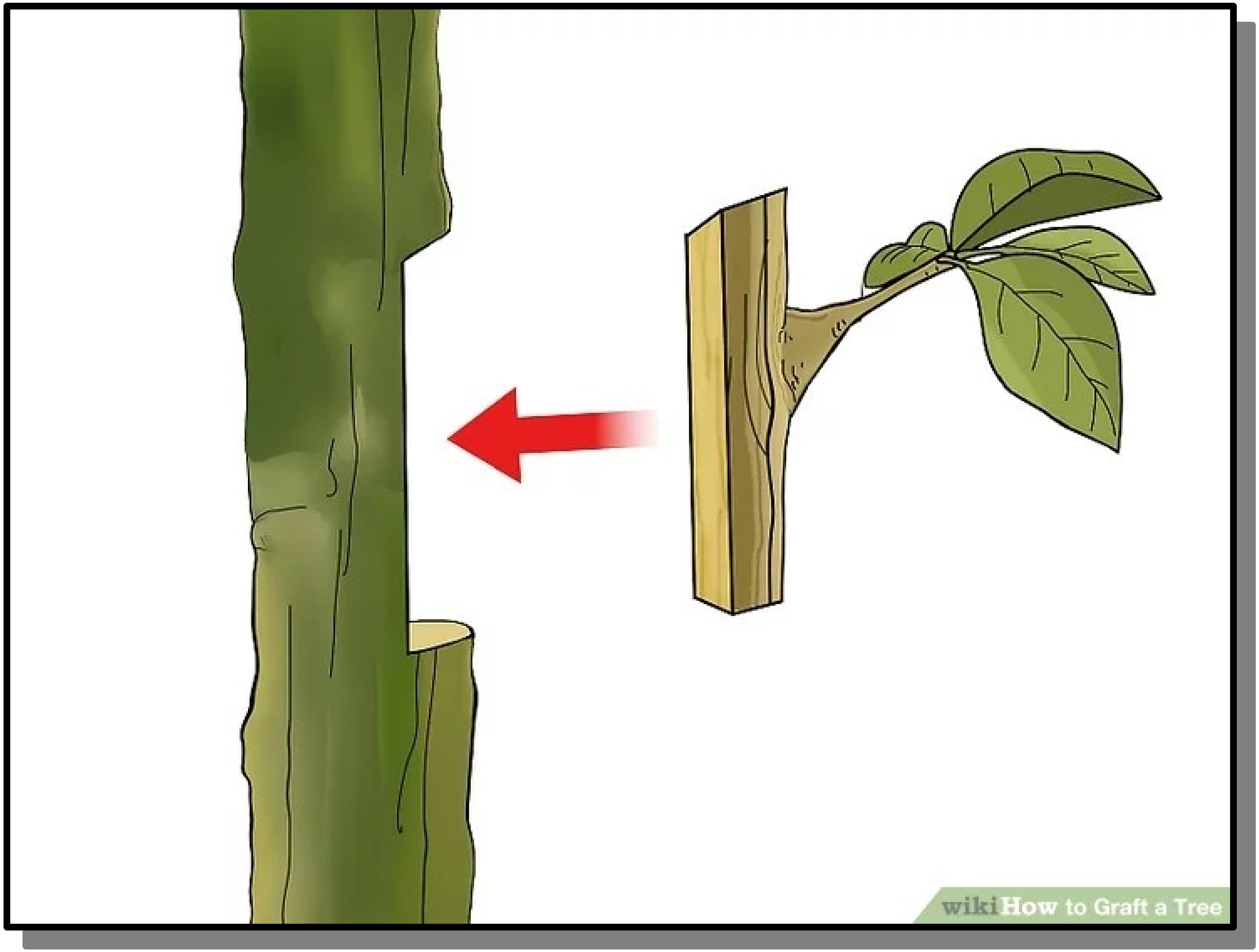
- The exam format is the same as last time, with the following changes: ***our style expectations on the exam are the same for the assignments***.
- For example, you should comment your code as you do in the assignments and should follow standard coding conventions.

Practice Problems

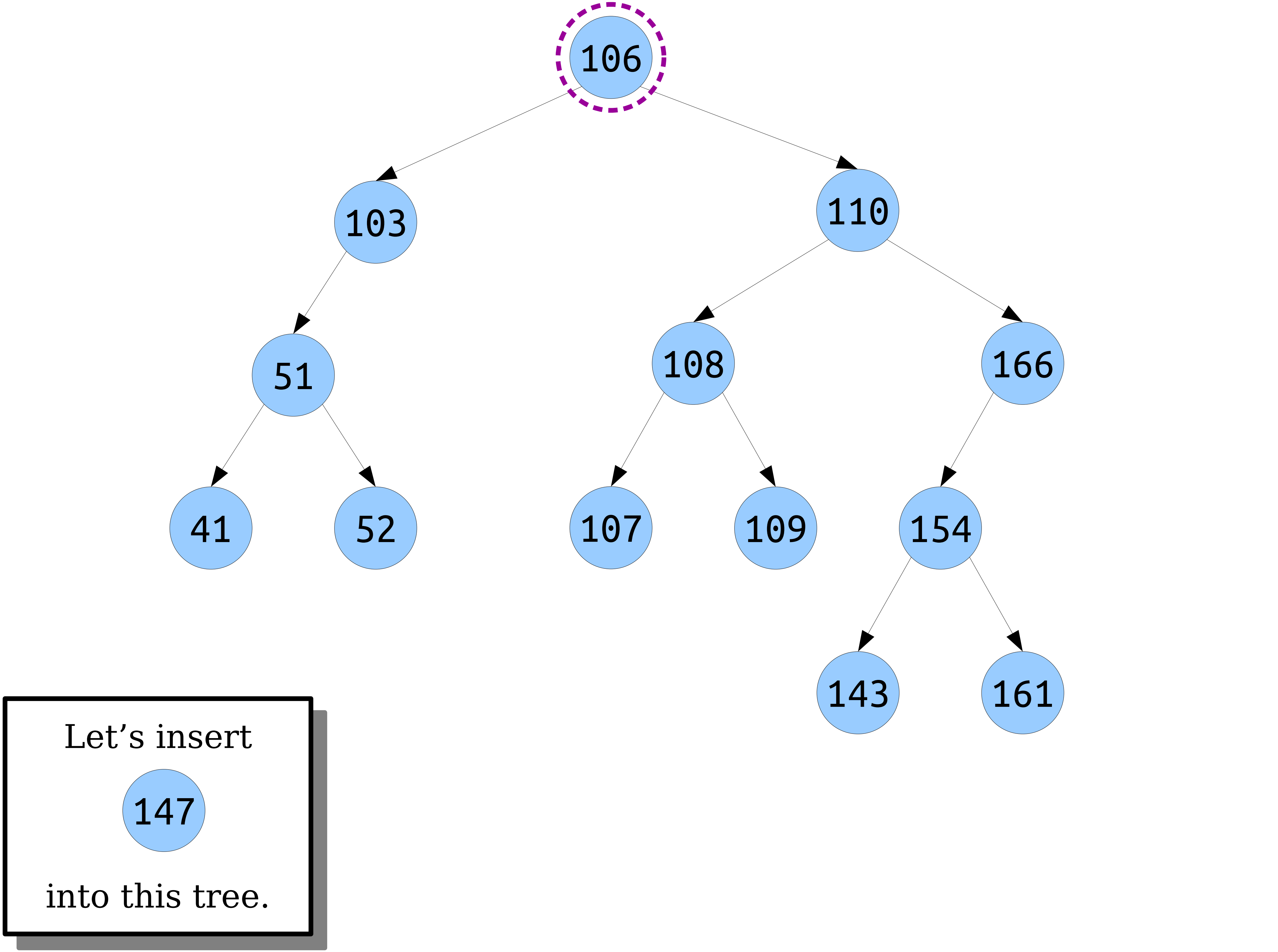
- We’ve posted a set of practice problems for the midterm to the course website, along with solutions.
- These practice problems are compiled from several previous exams. The style and form of the questions are similar to what we might ask, but the number of questions and total length is not representative.

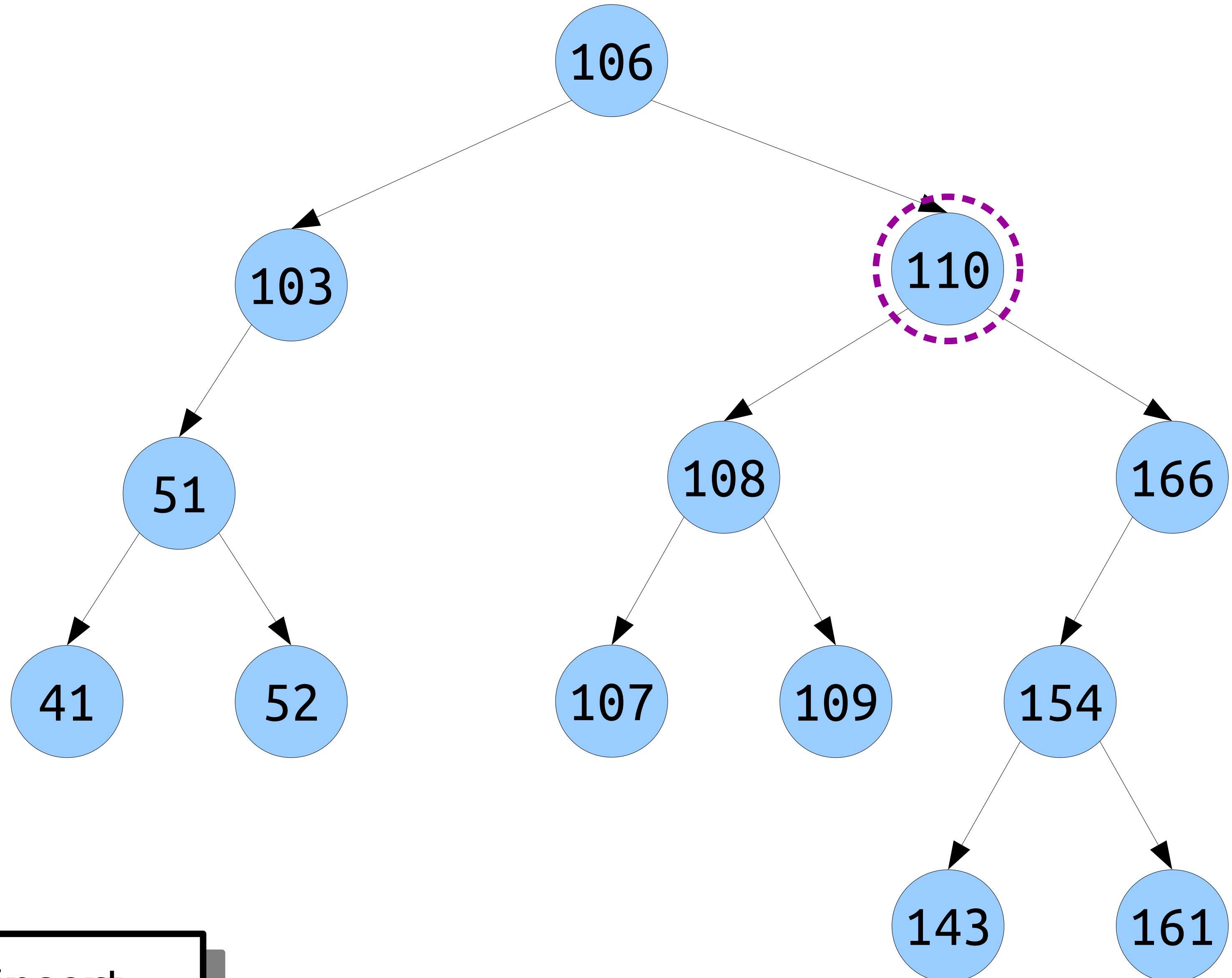
Re-tree-t into the forest…

Adding to Trees



Thanks, WikiHow!

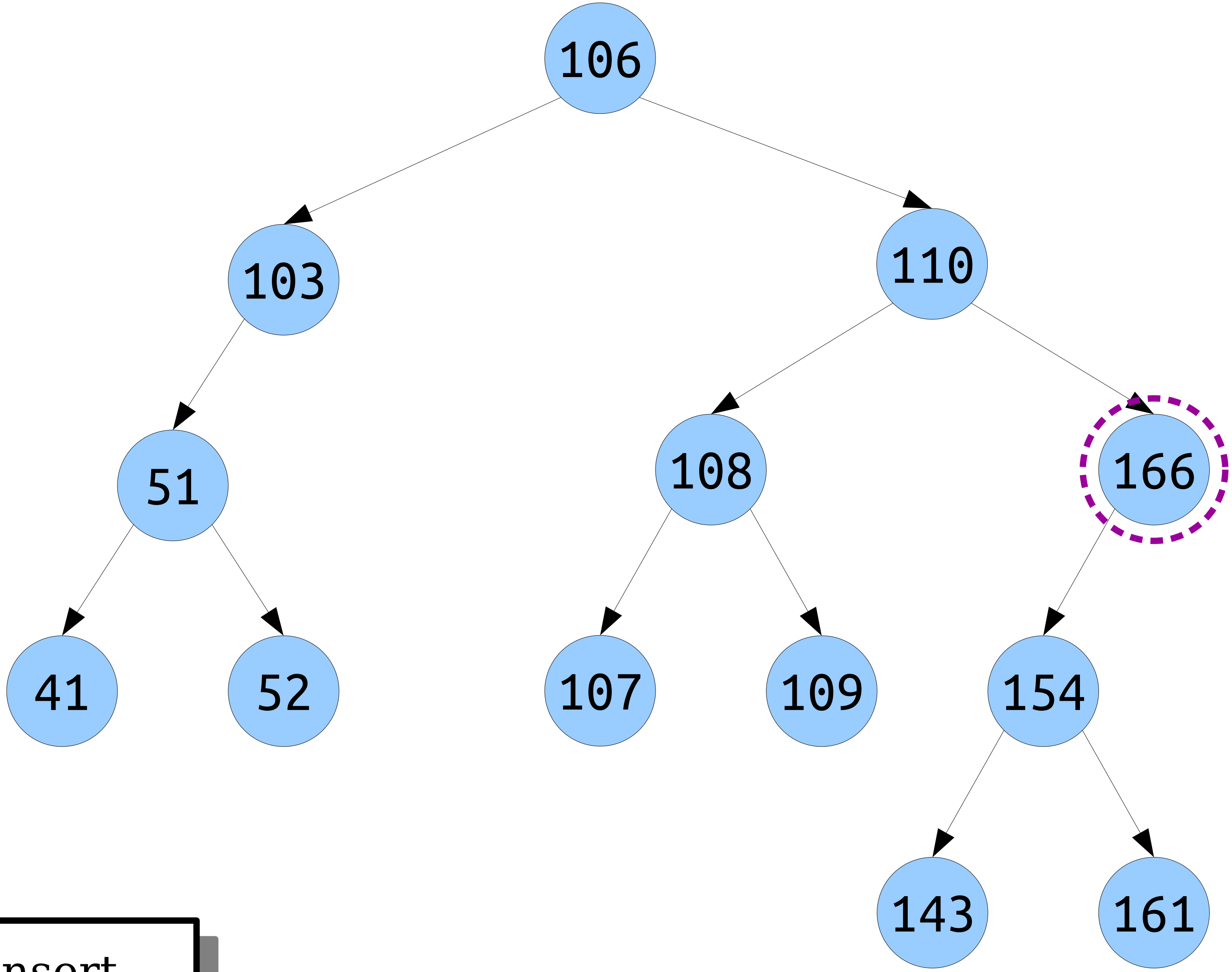

106
103
110
51
108
166
41
52
107
109
154
143
161
Let's insert
147
into this tree.



Let’s insert

147

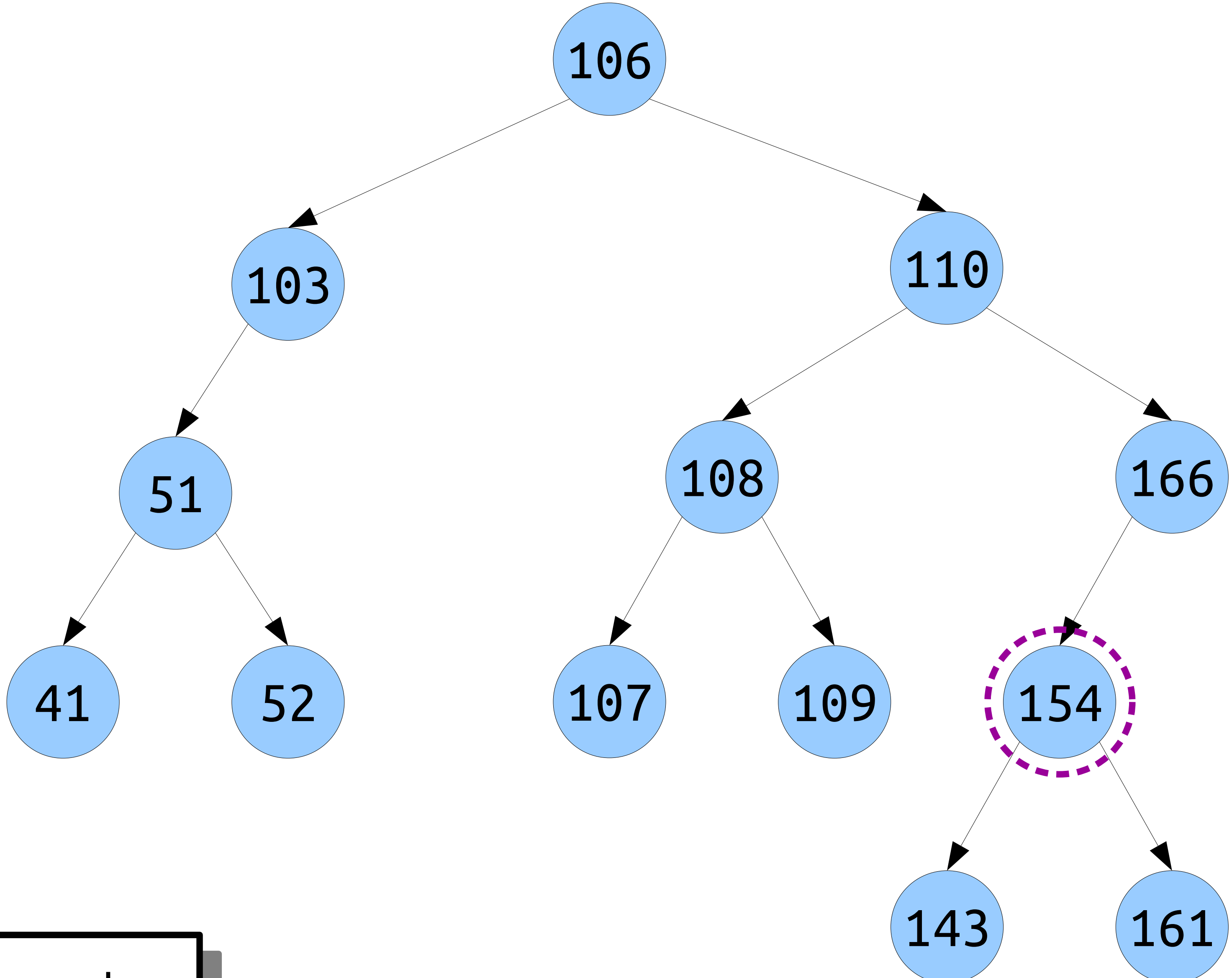
into this tree.

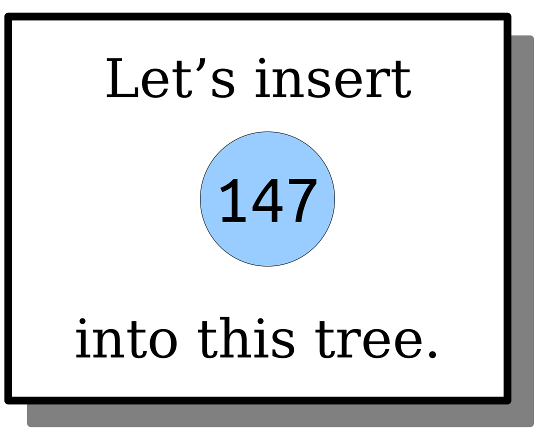


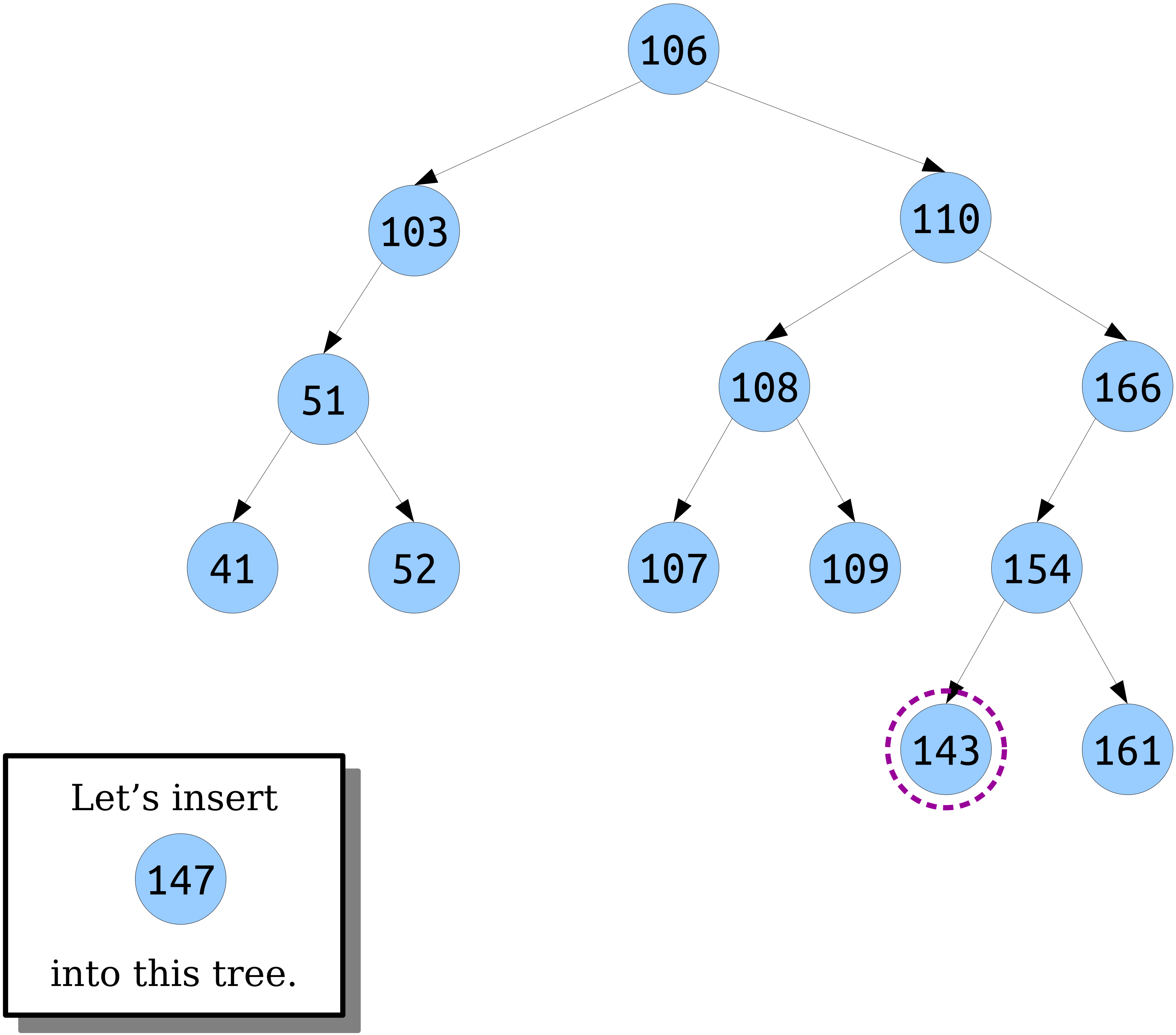
Let’s insert

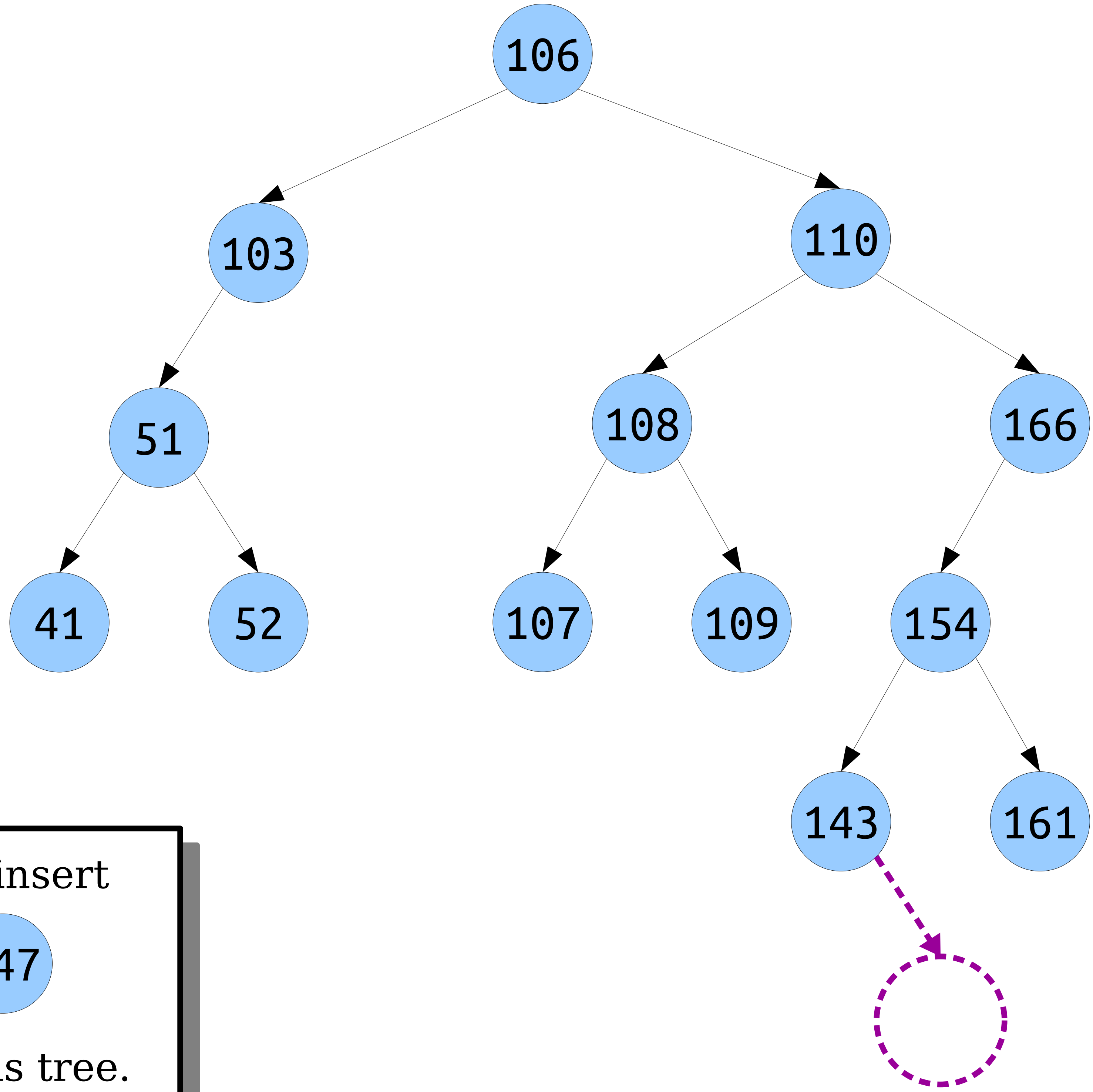
147

into this tree.


106
103
110
51
108
166
41
52
107
109
154
143
161


Let's insert
147
into this tree.

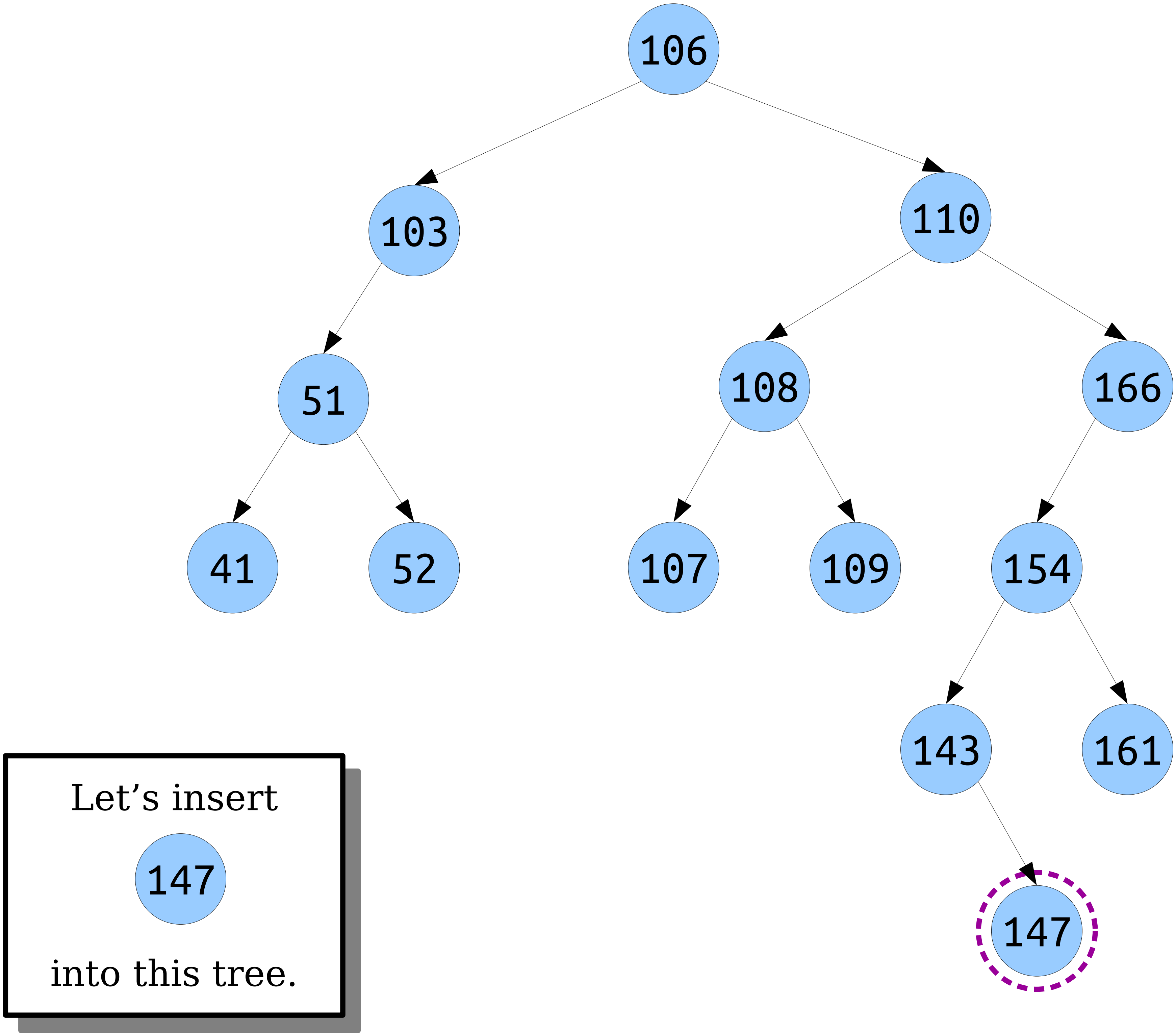

106
103
110
51
108
166
41
52
107
109
154
143
161
Let’s insert
147
into this tree.

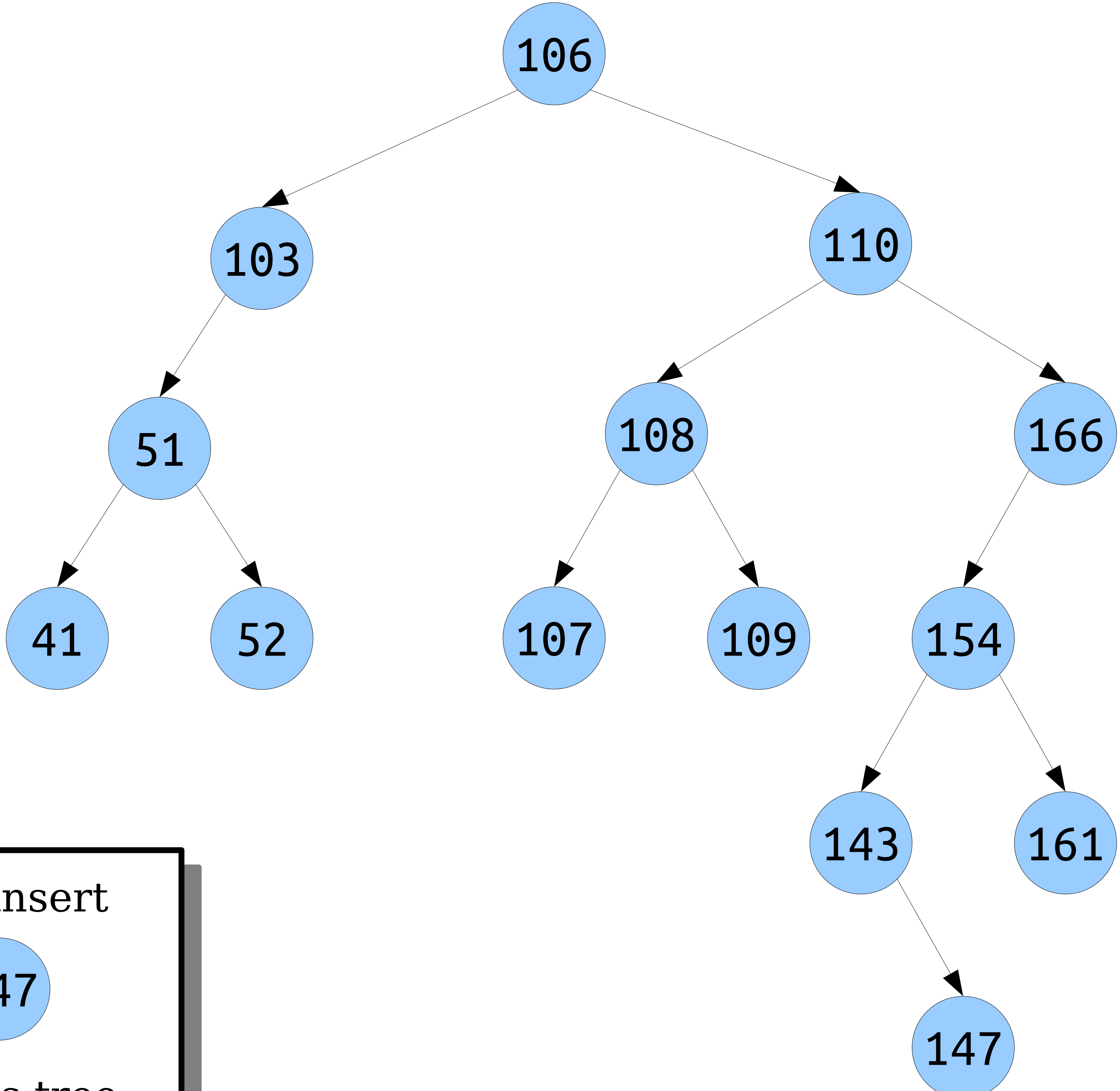


Let’s insert

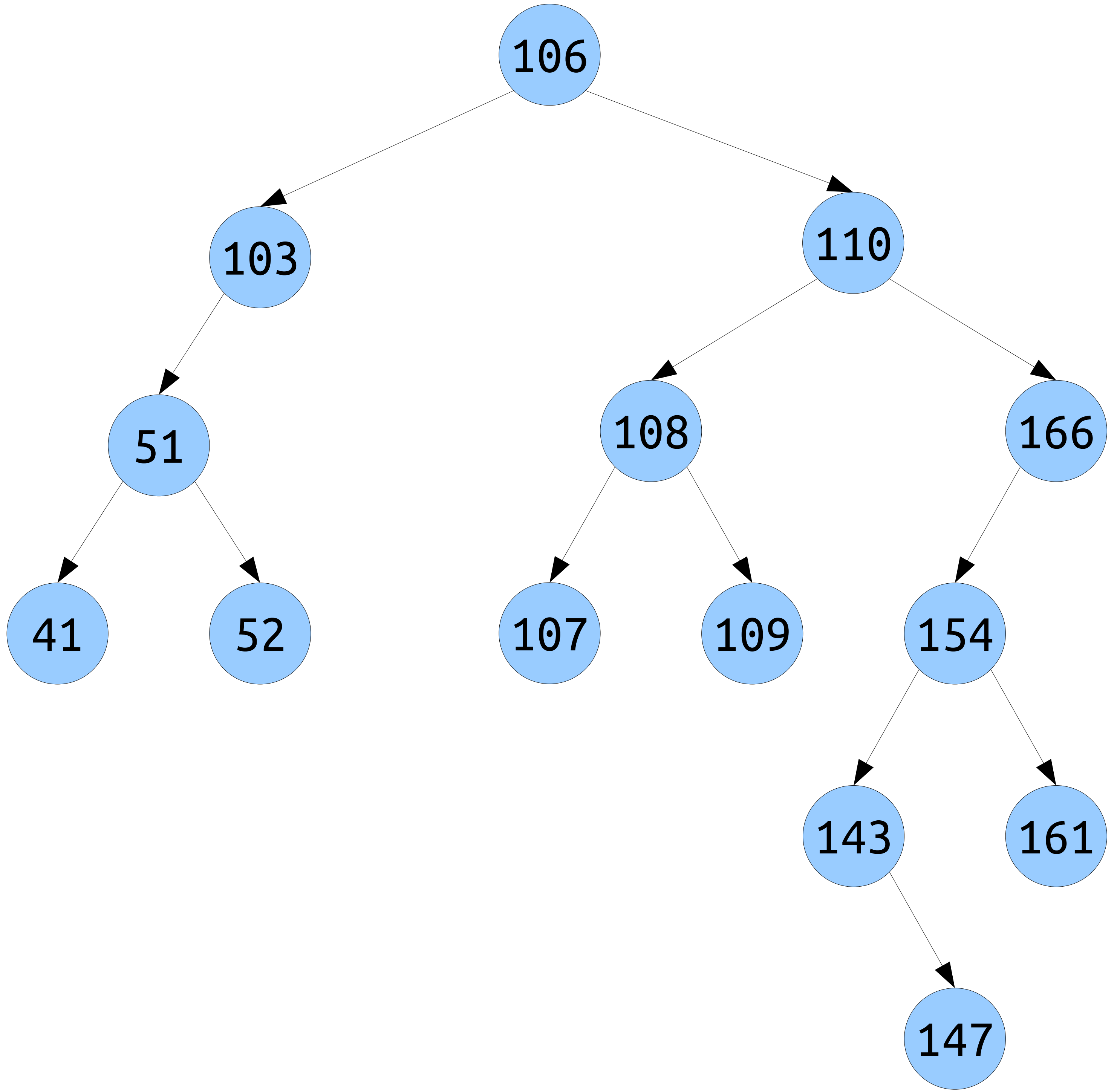
147

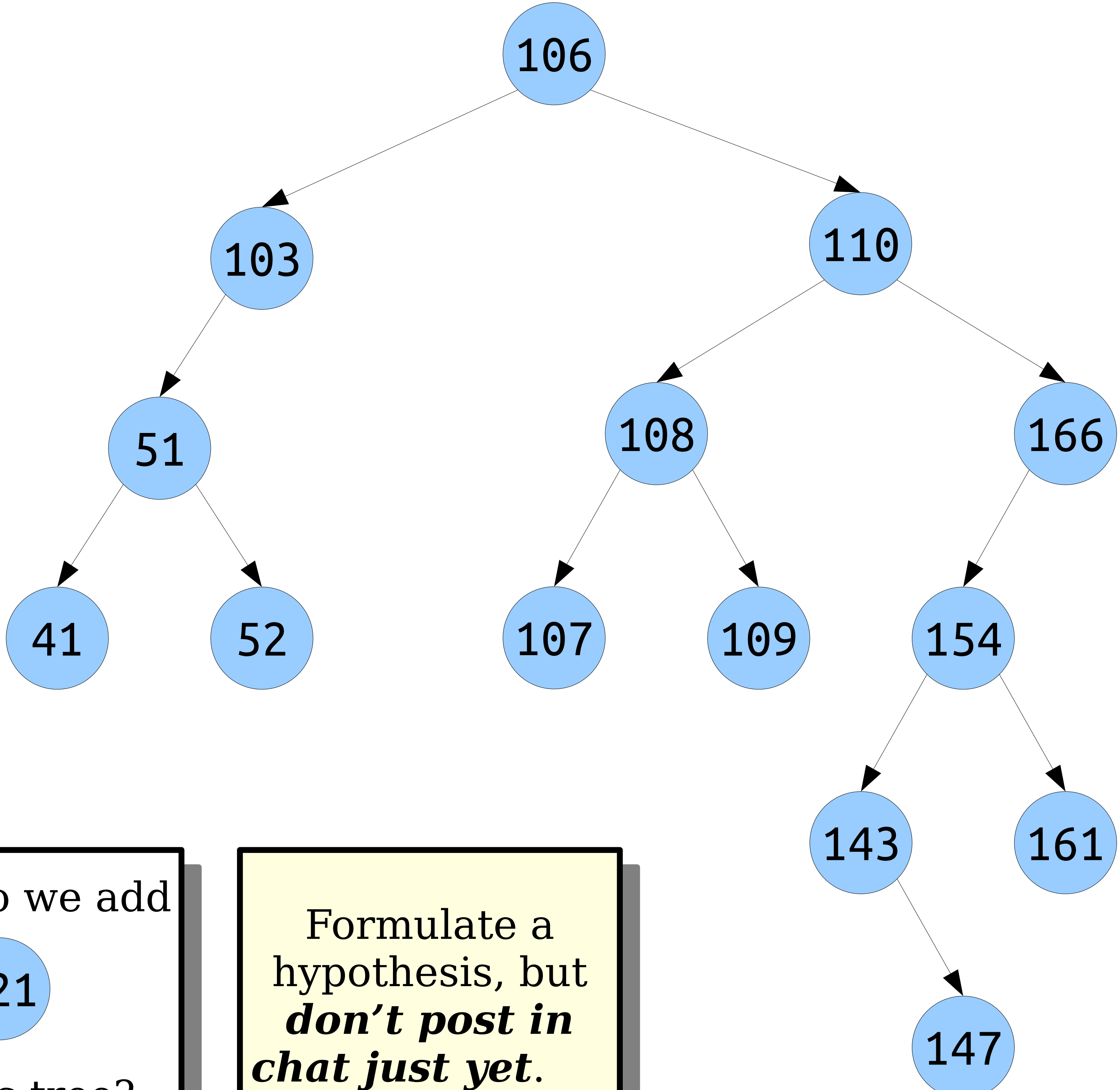
into this tree.


106
103
110
51
108
166
41
52
107
109
154
143
161
147
Let’s insert
147
into this tree.



Let’s insert 147 into this tree.


106
103
110
51
108
166
41
52
107
109
154
143
161
147

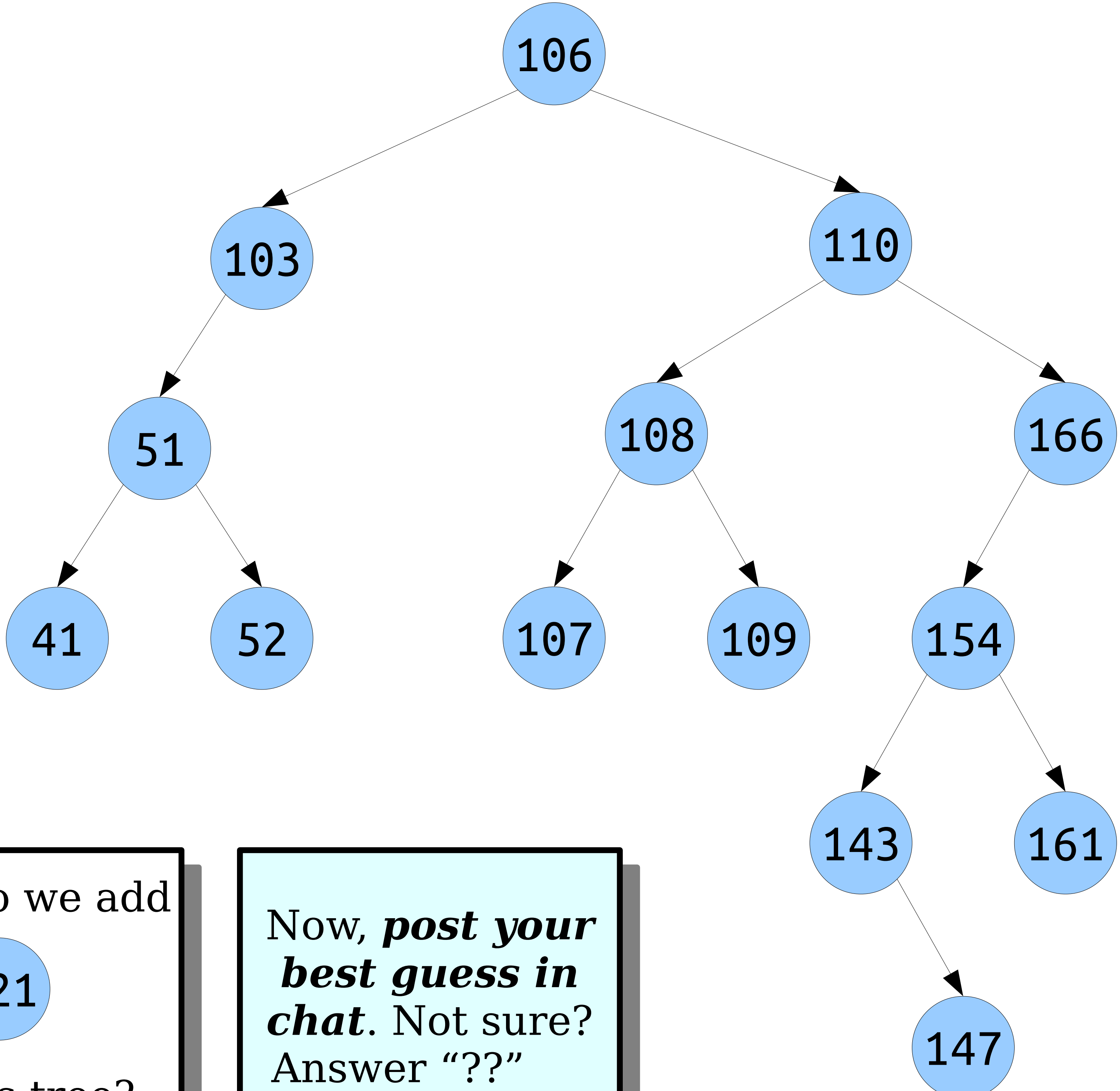


Where do we add

221

into this tree?

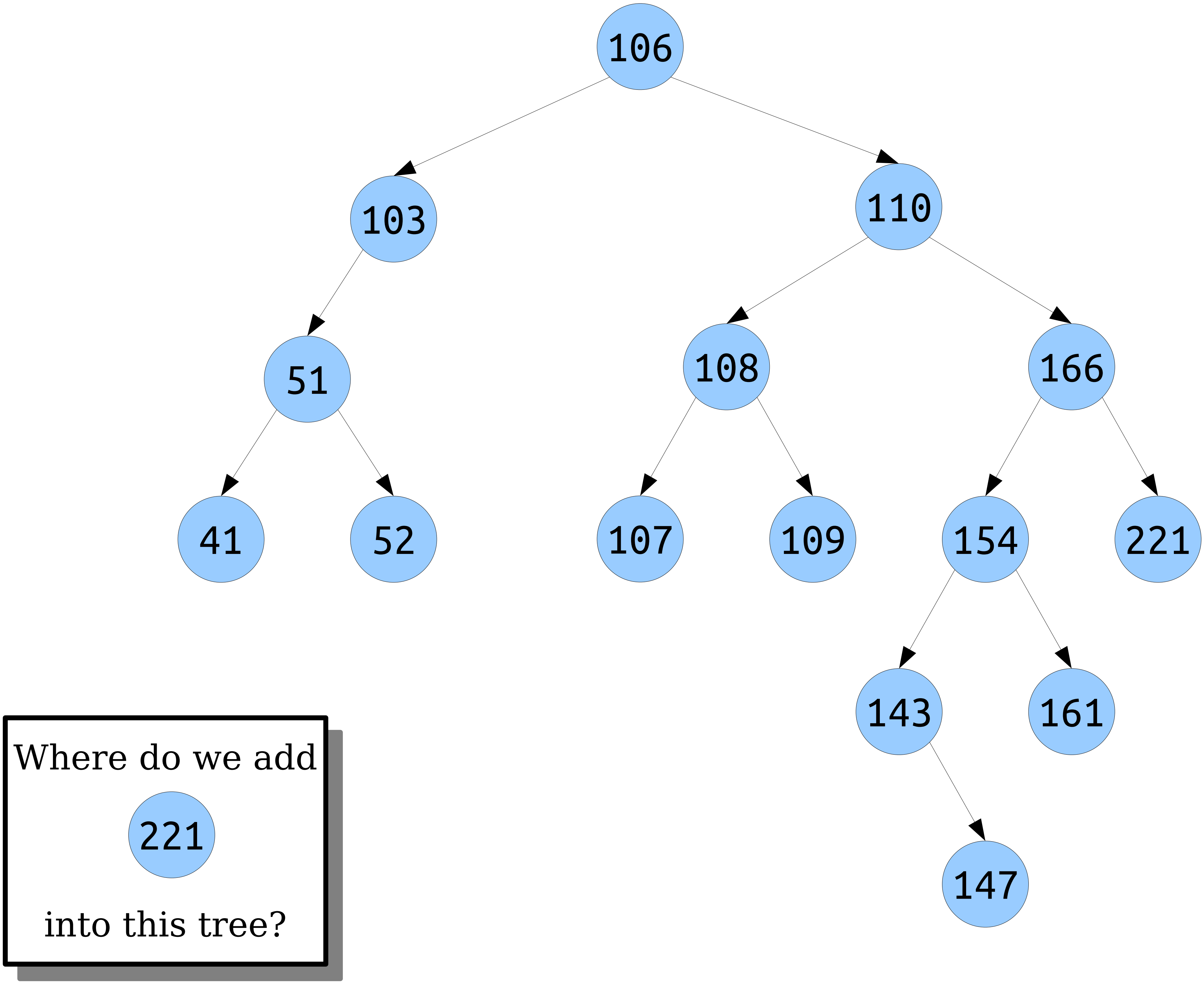
Formulate a hypothesis, but ***don’t post in chat just yet***.



Where do we add

into this tree?

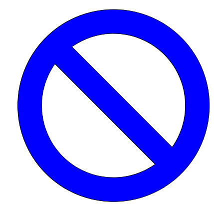
Now, ***post your best guess in chat***. Not sure? Answer "??"


106
103
110
51
108
166
41
52
107
109
154
221
143
161
147
Where do we add
221
into this tree?

Let's Code it Up!

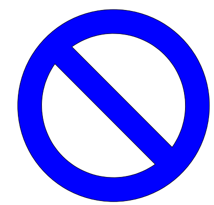
A Binary Search Tree Is Either…

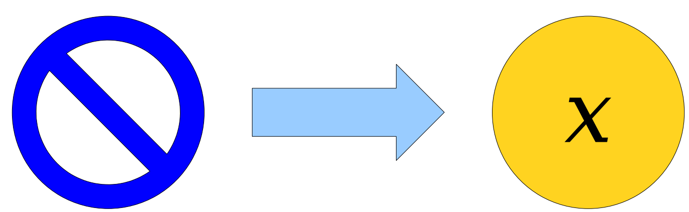
an empty tree,
represented by
`nullptr`



A Binary Search Tree Is Either…

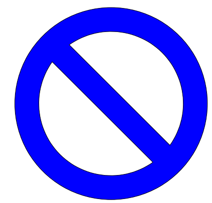
an empty tree, represented by `nullptr`



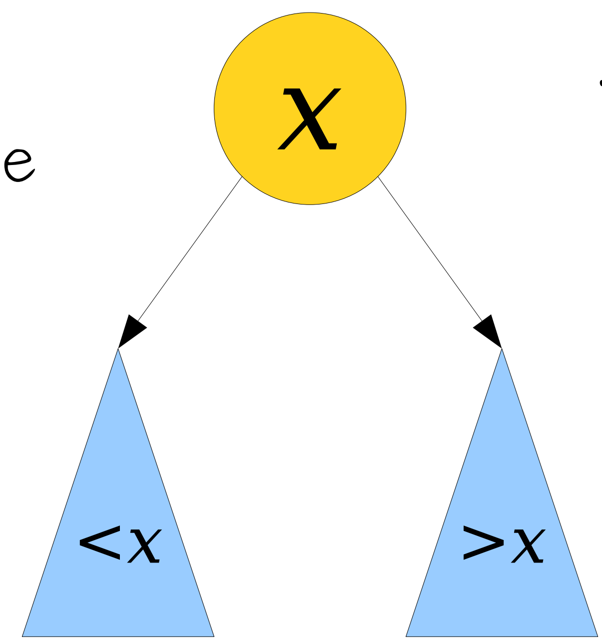


A Binary Search Tree Is Either…

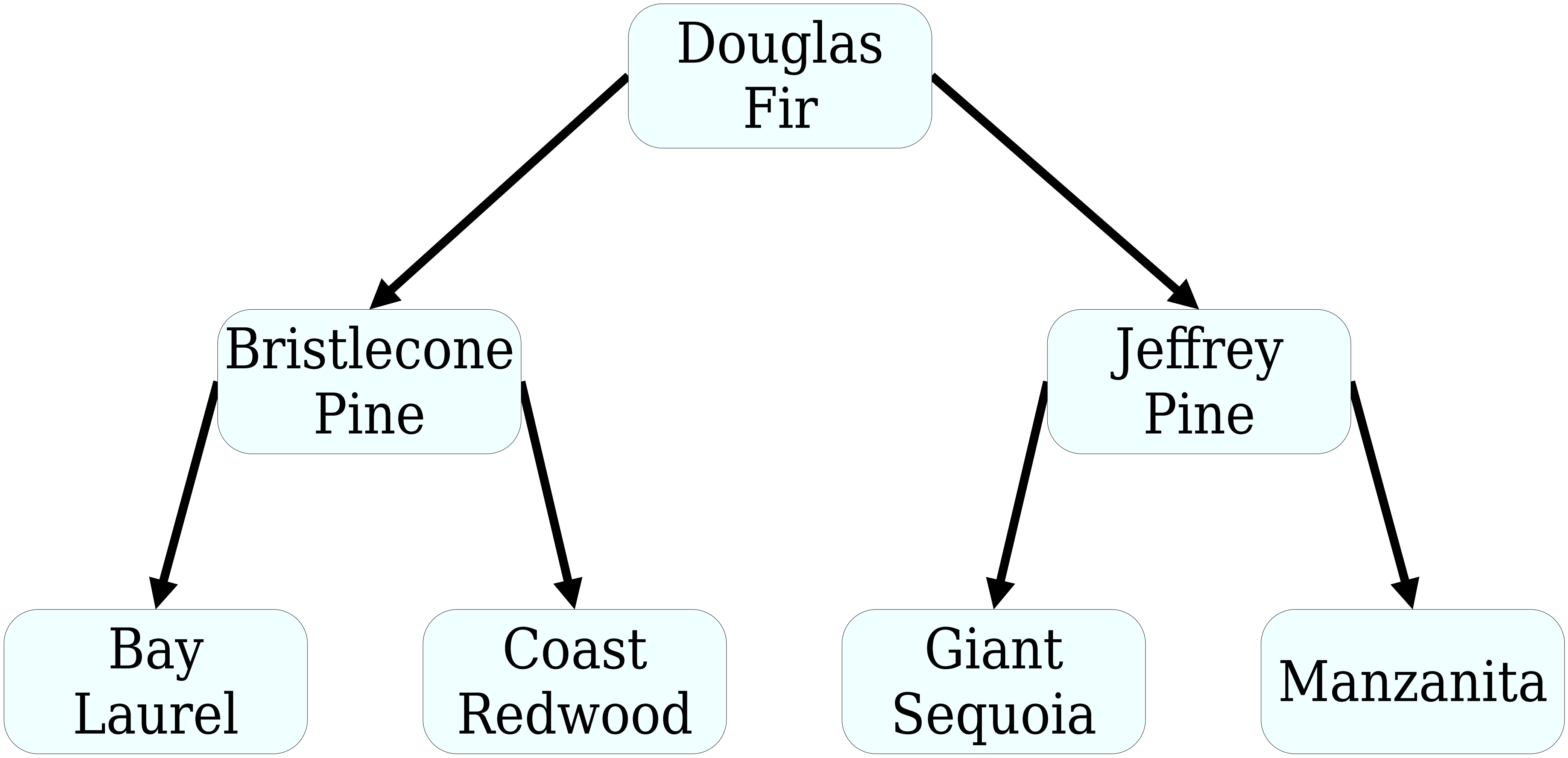
an empty tree, represented by **`nullptr`**, or…



… a single node, whose left subtree is a BST of smaller values …



… and whose right subtree is a BST of larger values.


Douglas Fir
Bristlecone Pine
Jeffrey Pine
Bay Laurel
Coast Redwood
Giant Sequoia
Manzanita

Your Action Items

- ***Read Chapter 16.1 – 16.2.***
  - There's a bunch of BST topics in there, along with a different intuition for how they work.
- ***Start Assignment 8.***
  - See if you can escape from your labyrinths by Monday!

Next Time

- ***Tree Heights***
  - Many trees can hold the same keys. How do we compare them?
- ***Freeing Trees***
  - Reclaiming memory in a tree.
- ***Range Searches***
  - Quickly finding all values in a range.