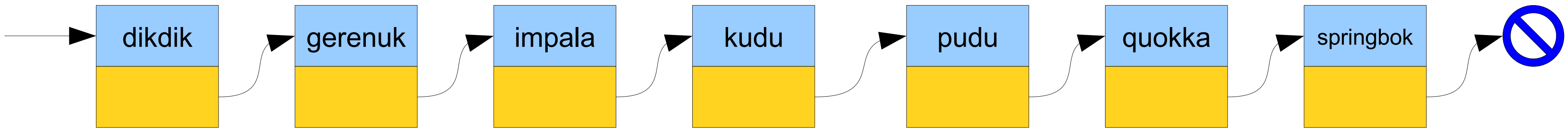
Binary Search Trees

Part One



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

Answer online at
https://cs106b.stanford.edu/pollev

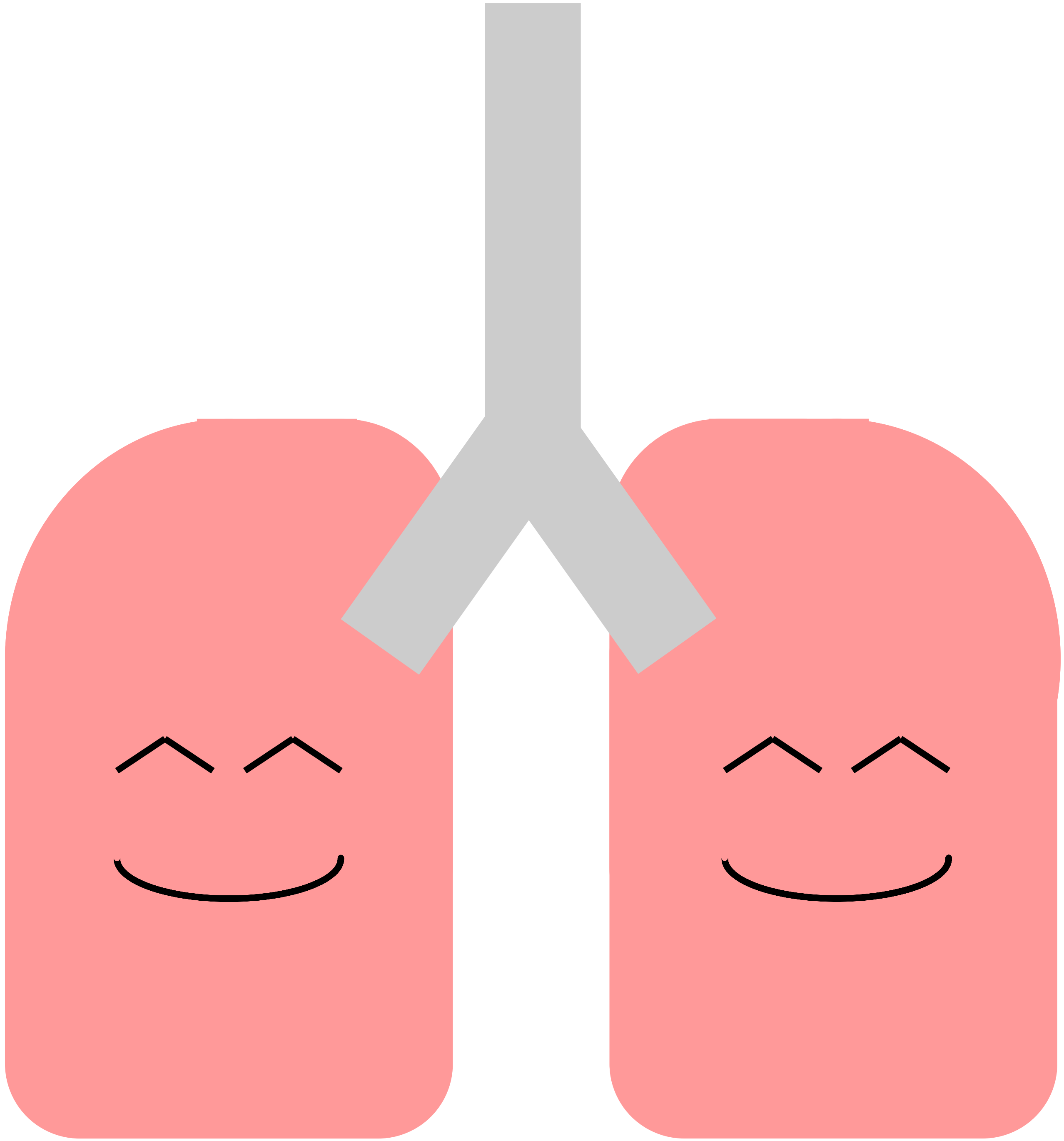
Can you chain a bunch of objects together so that most of them are near the front?

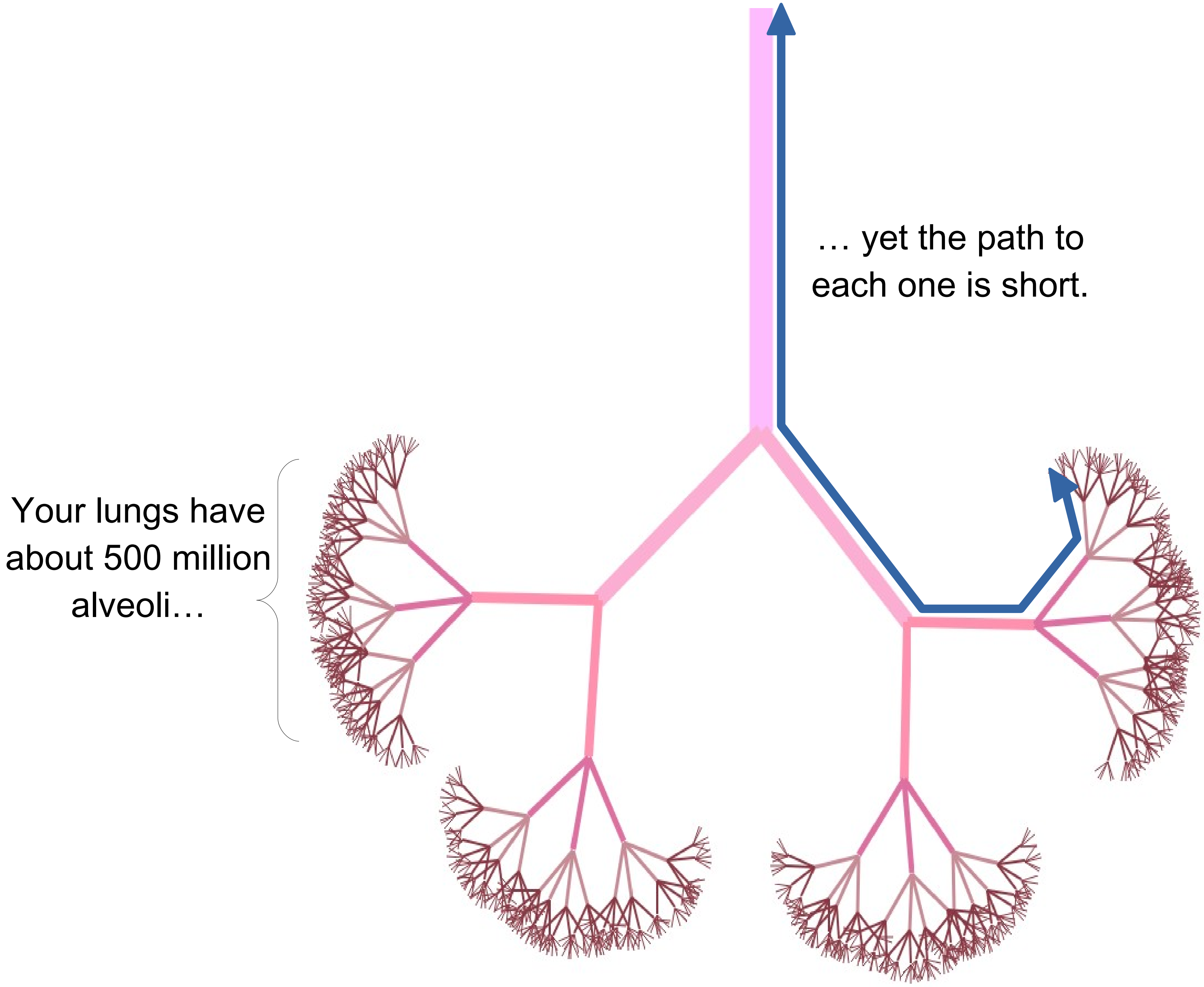
An Interactive Analogy

Take a deep breath.

And exhale.

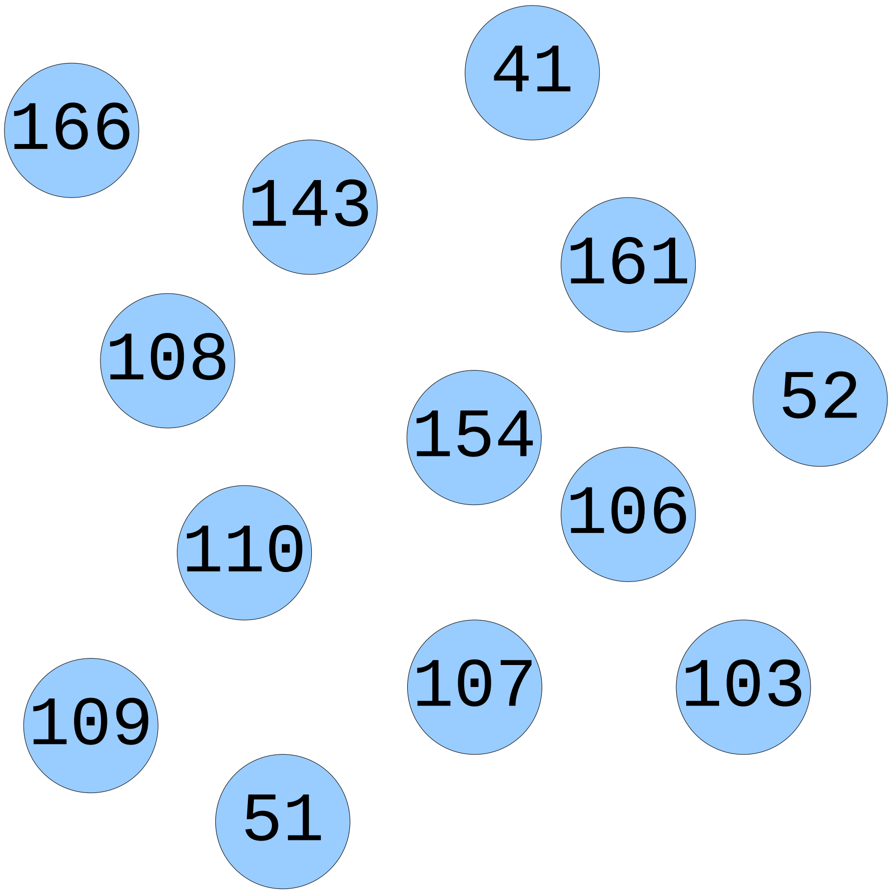
Feel nicely oxygenated?



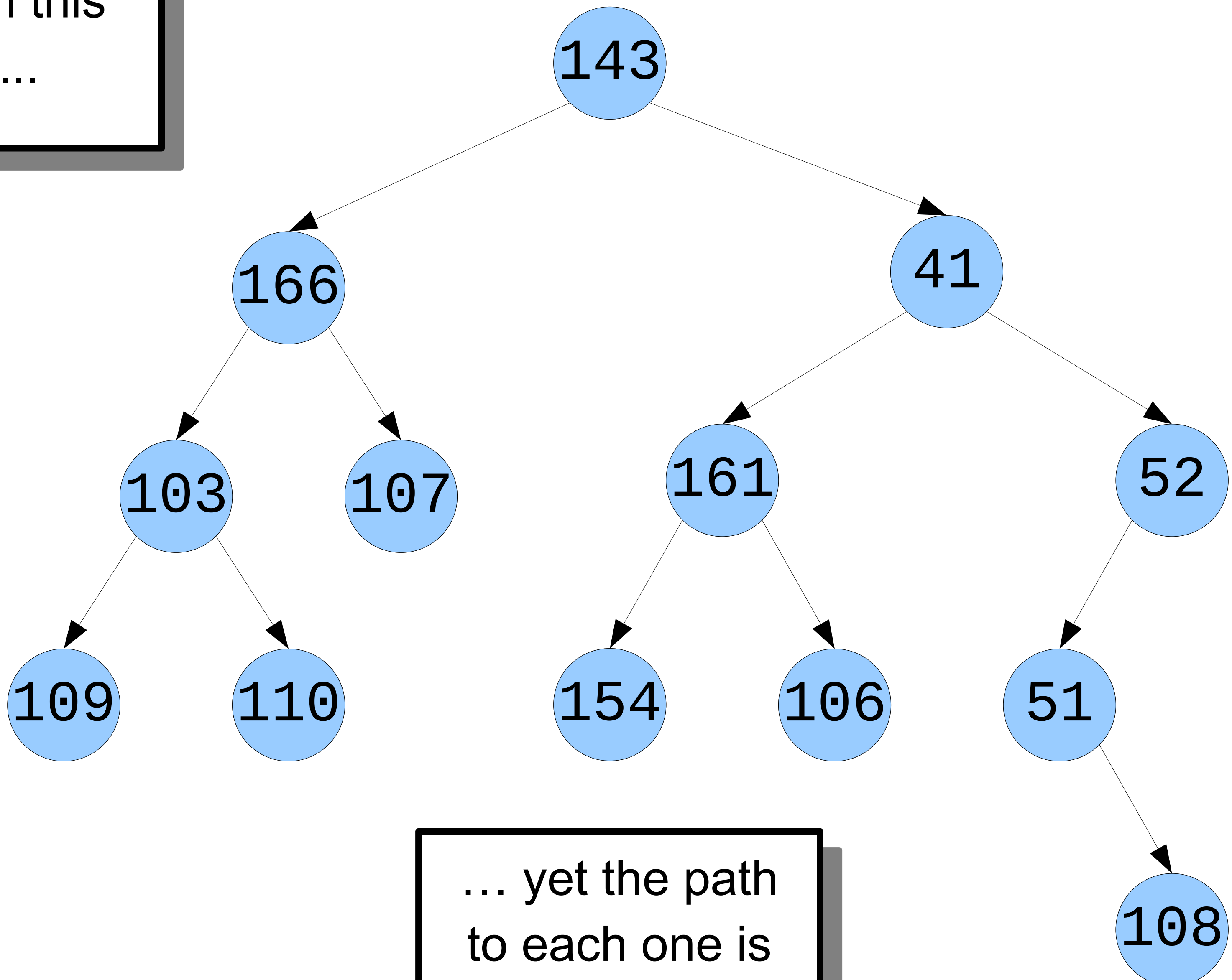

Your lungs have about 500 million alveoli…
… yet the path to each one is short.

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

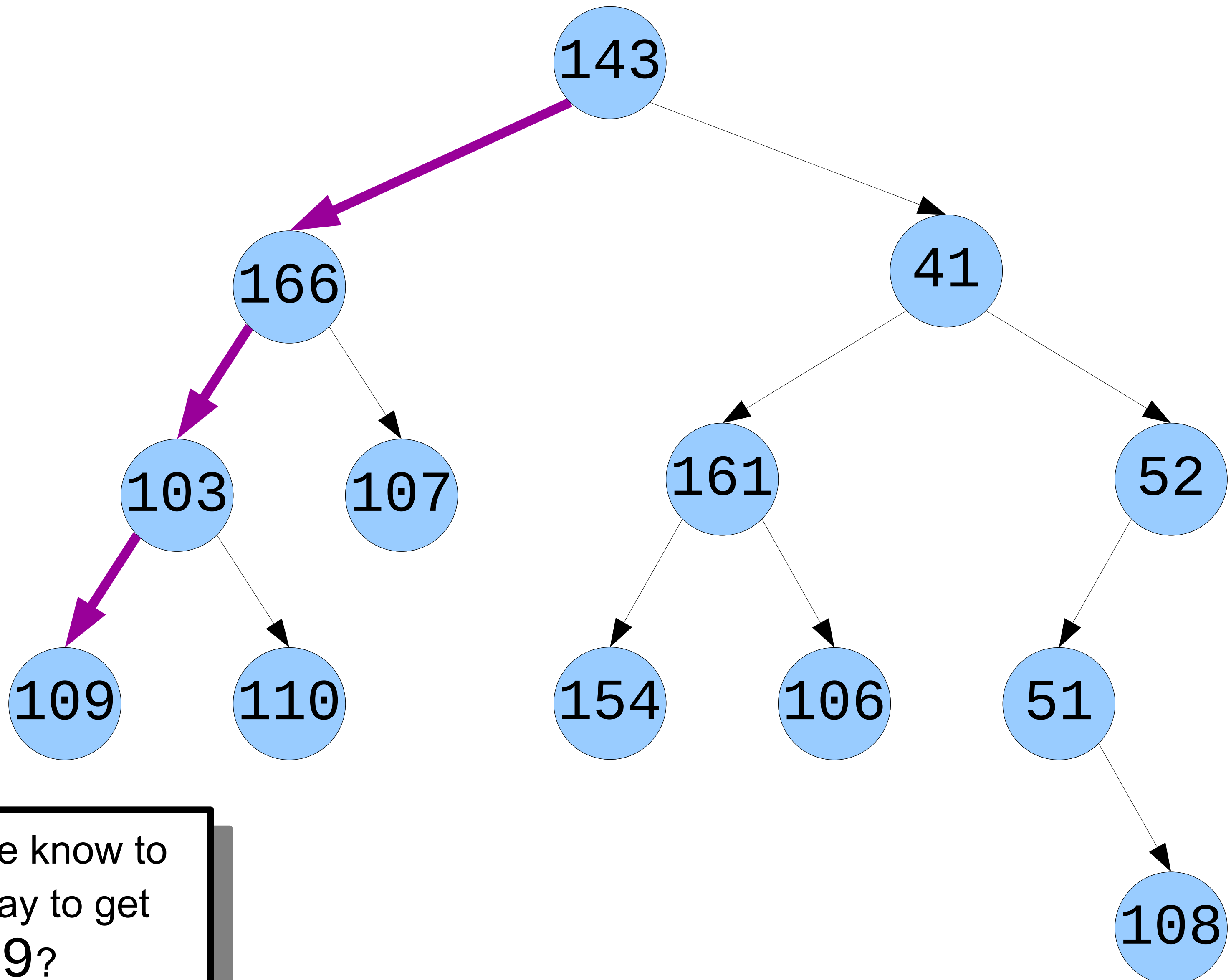
Harnessing this Insight


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

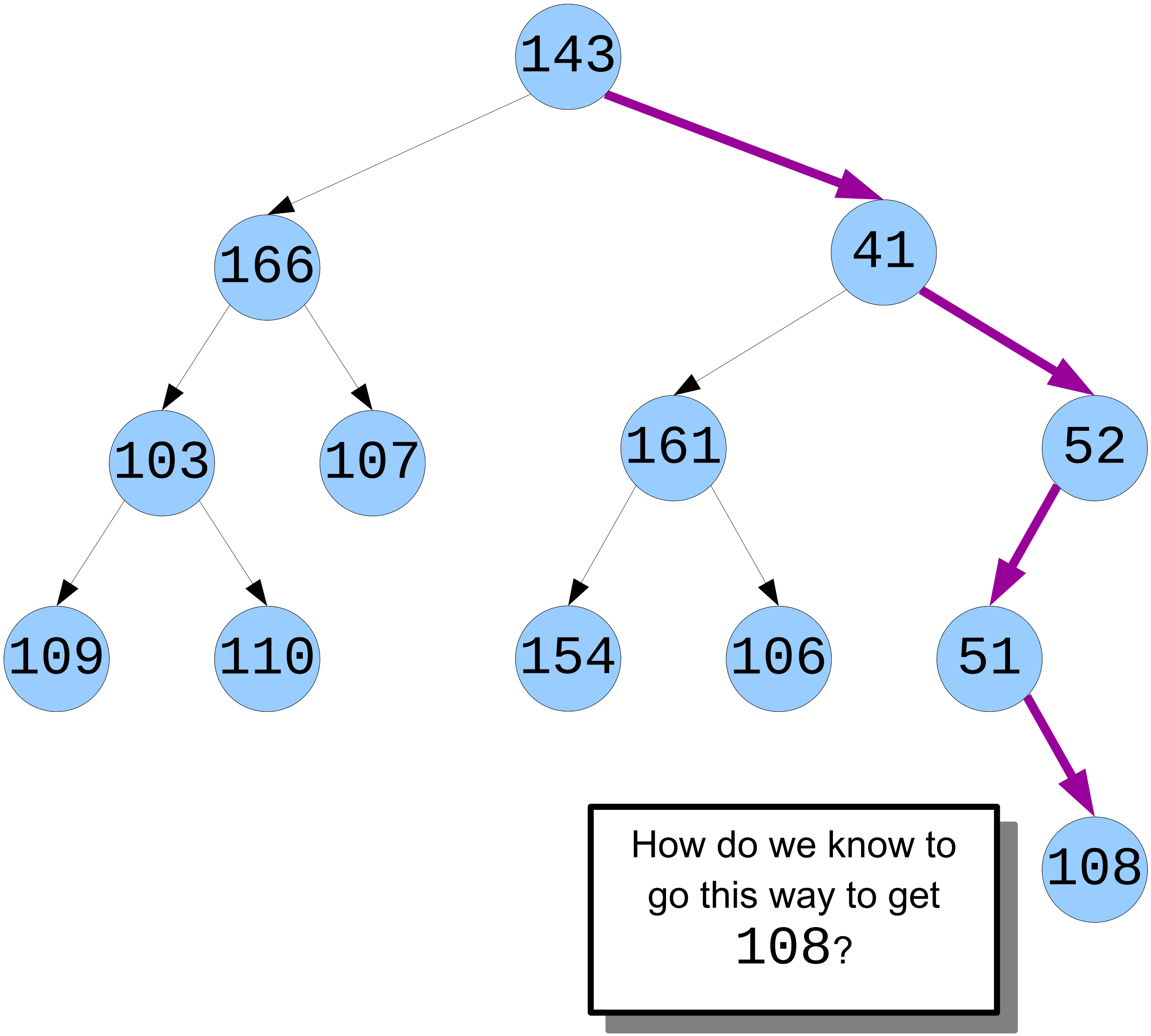
There are 13 ***nodes*** in this tree...



… yet the path to each one is short.

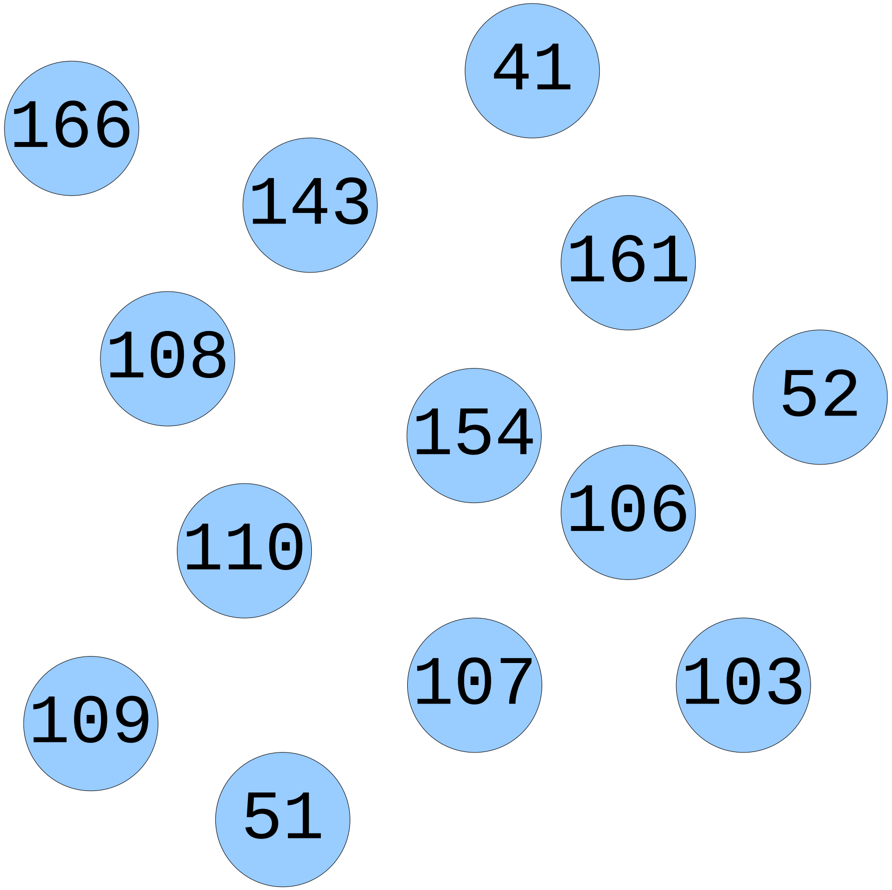


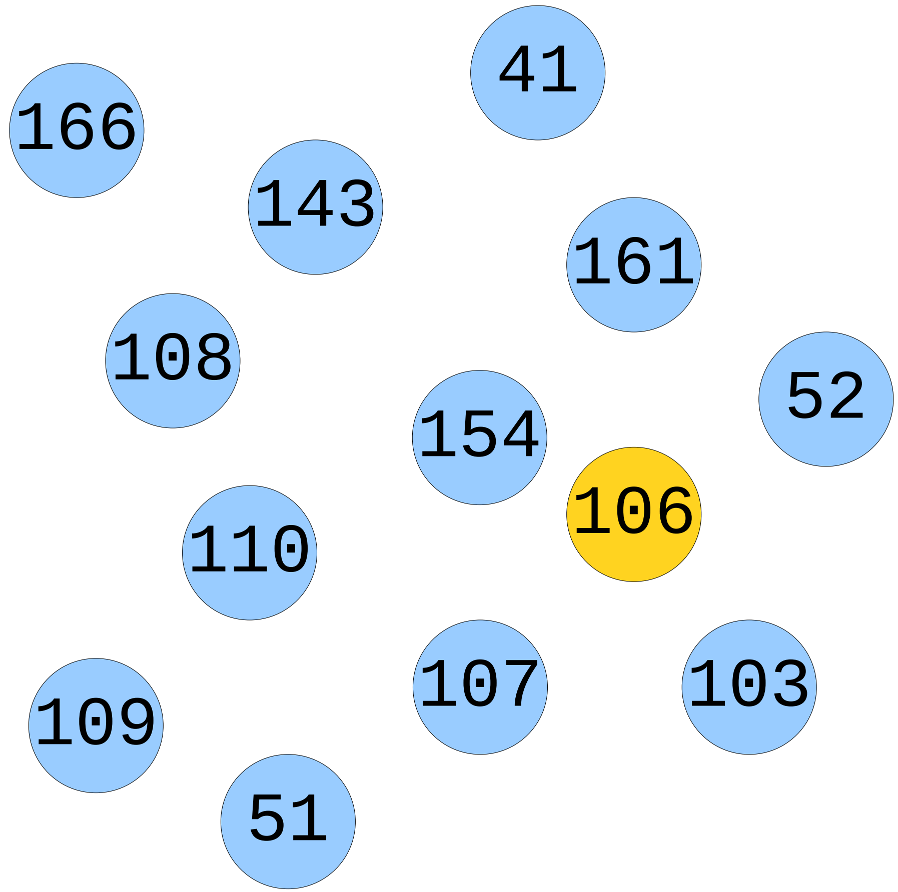
How do we know to go this way to get 109?

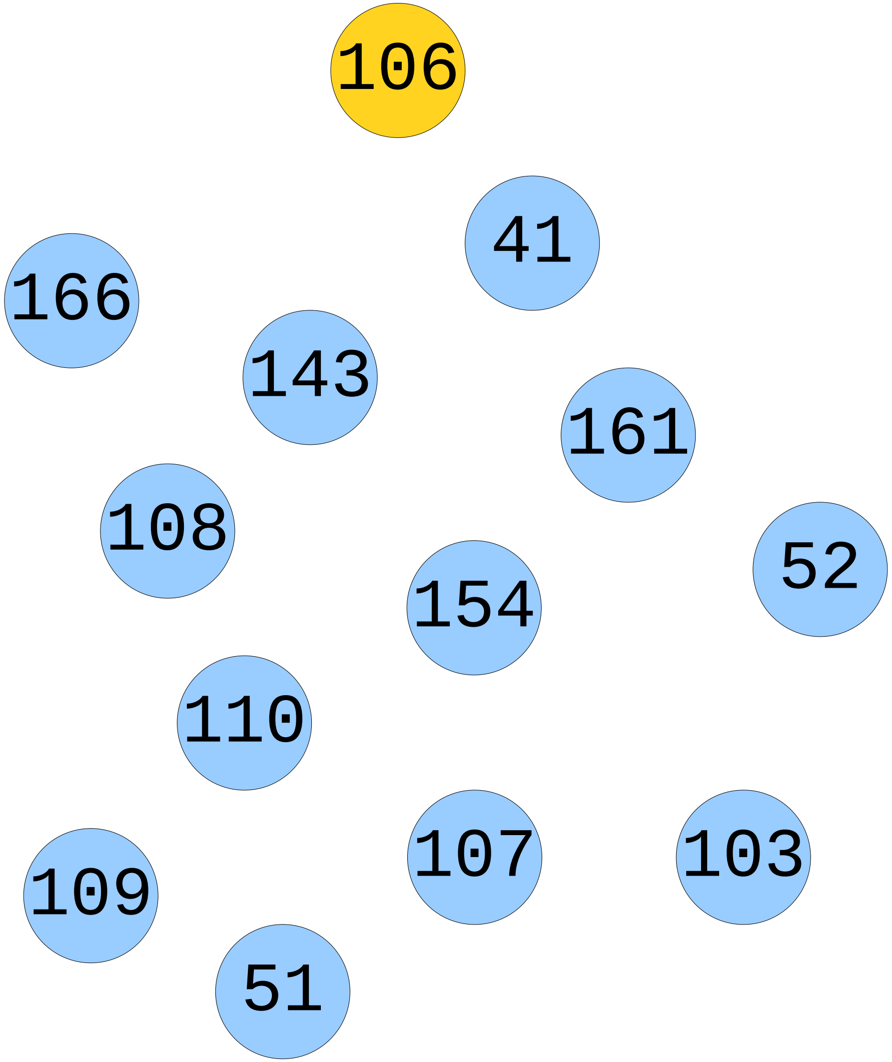


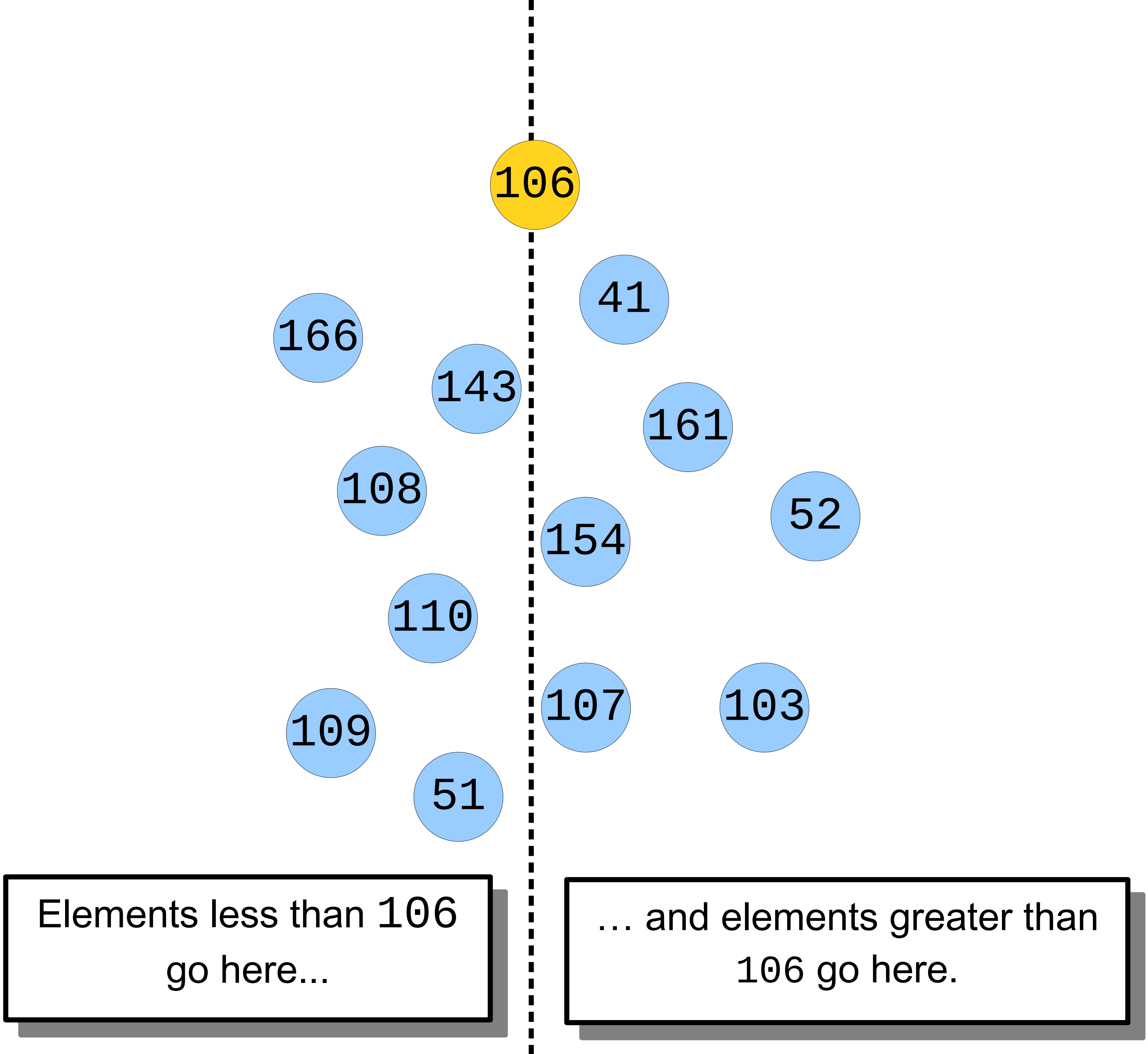
143
166
41
103
107
161
52
109
110
154
106
51
108
How do we know to go this way to get 108?

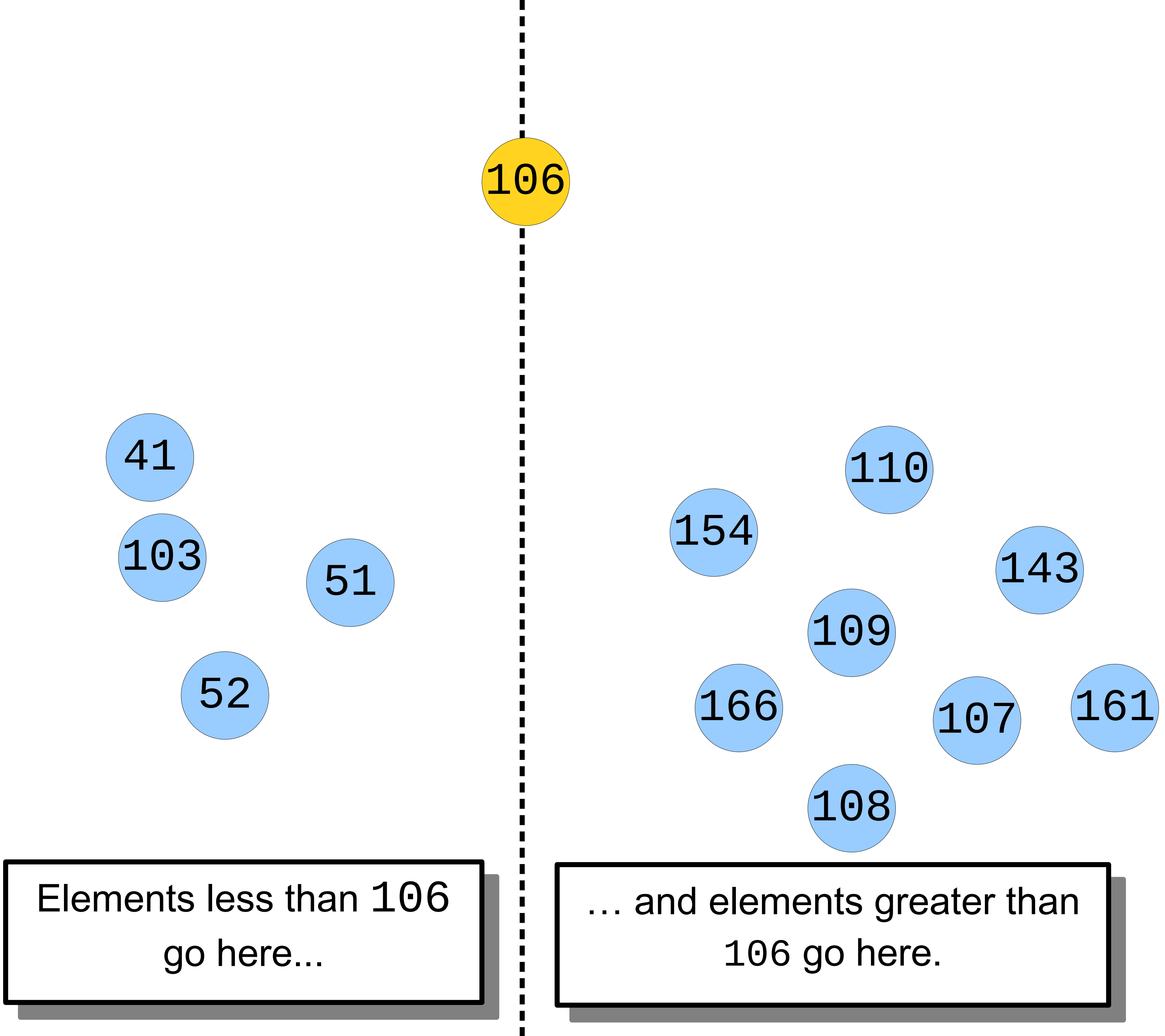
Goal: Store elements in a tree structure where there's an easy way to find them.

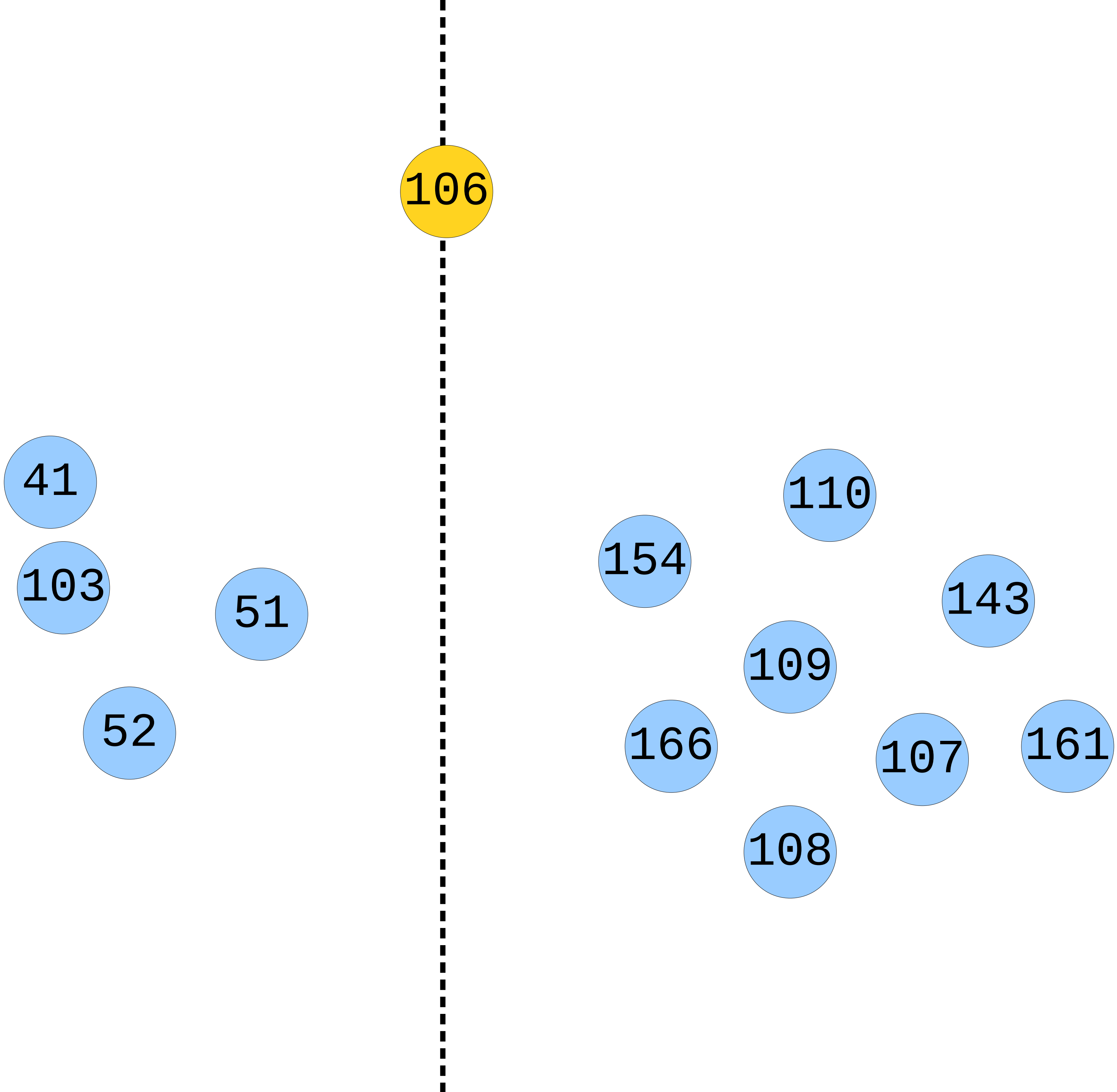

41
166
143
161
108
52
154
106
110
107
103
109
51

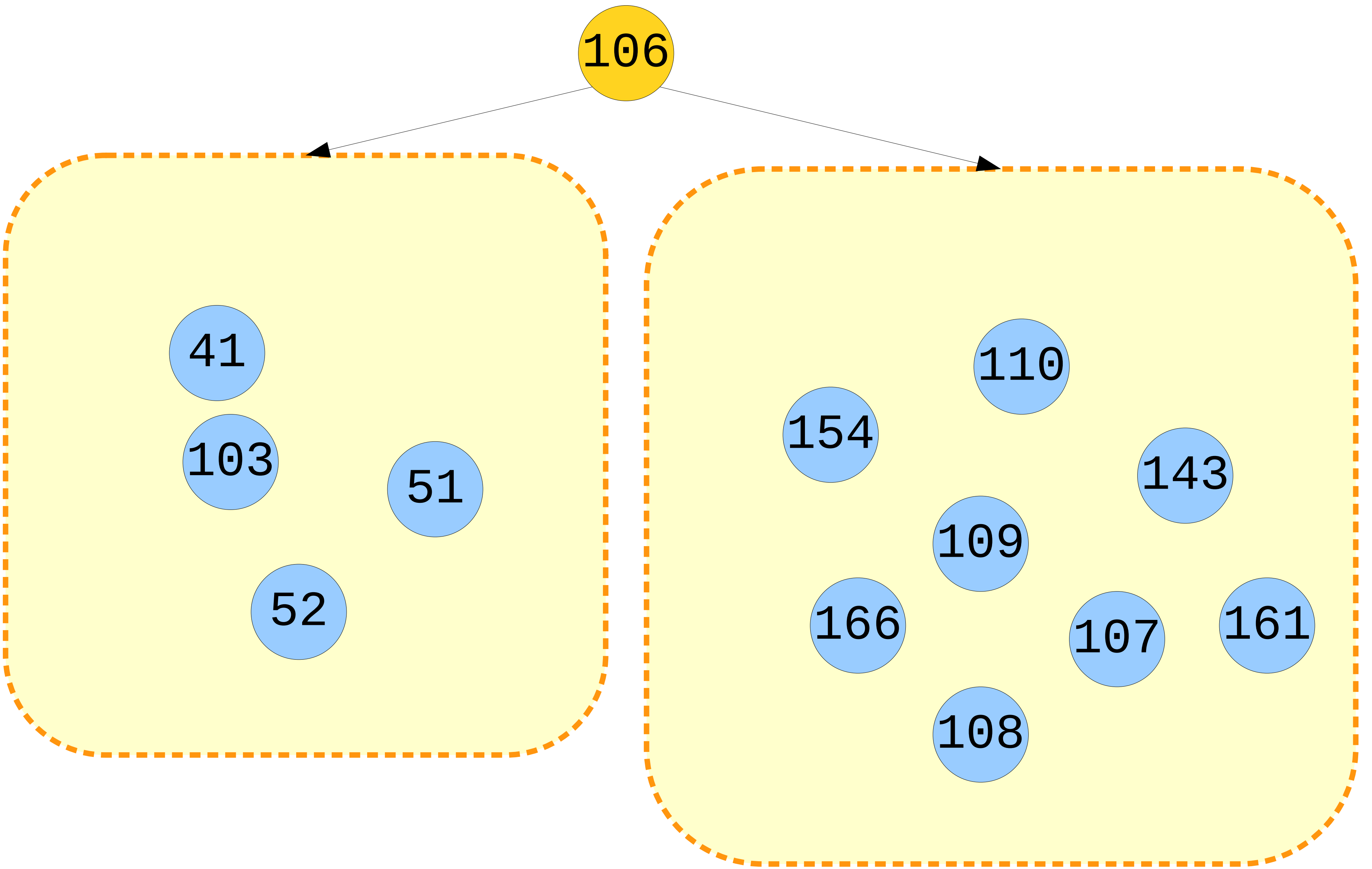

41
166
143
161
108
52
154
106
110
107
103
109
51

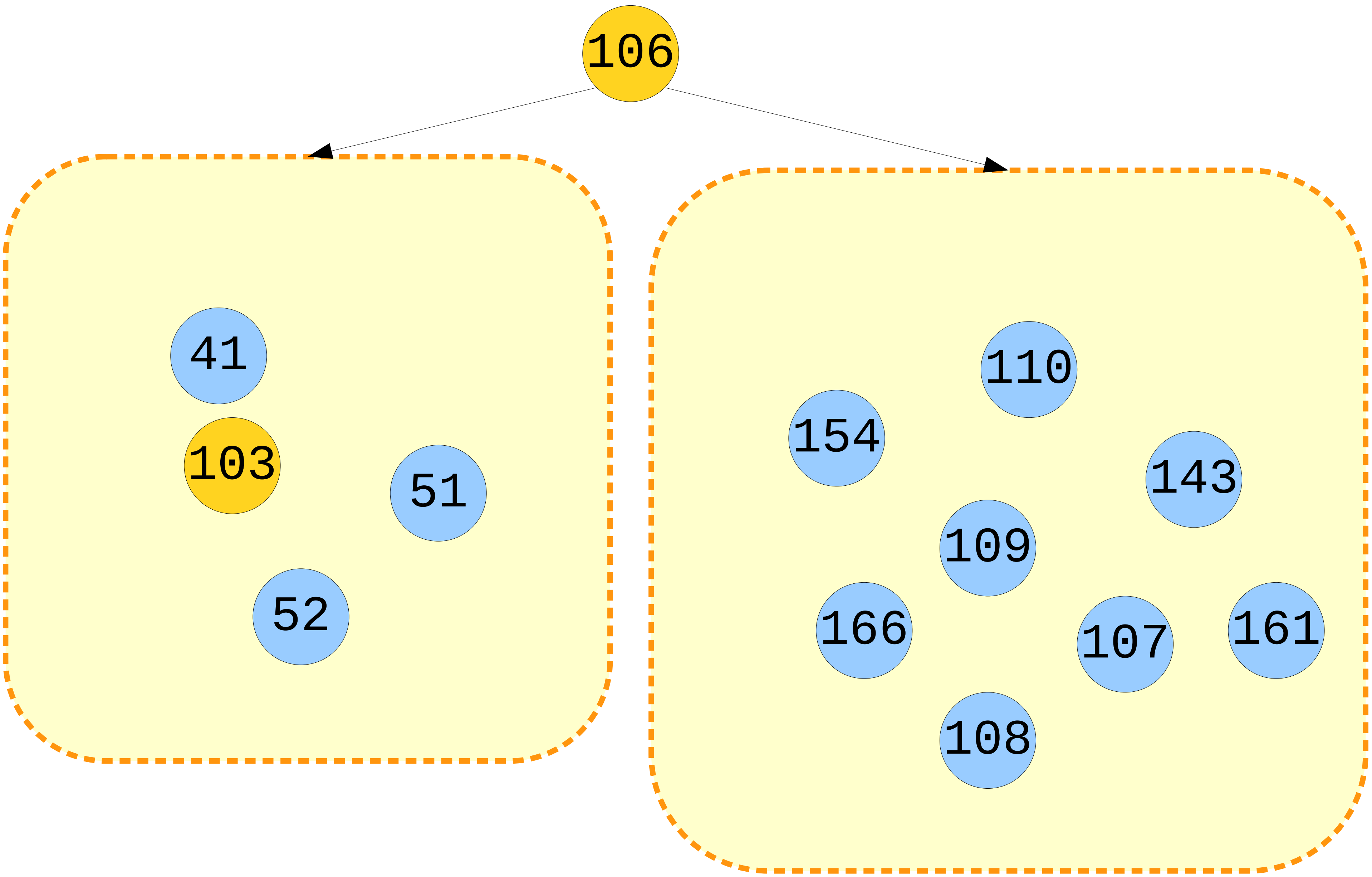

106
41
166
143
161
108
52
154
110
107
103
109
51

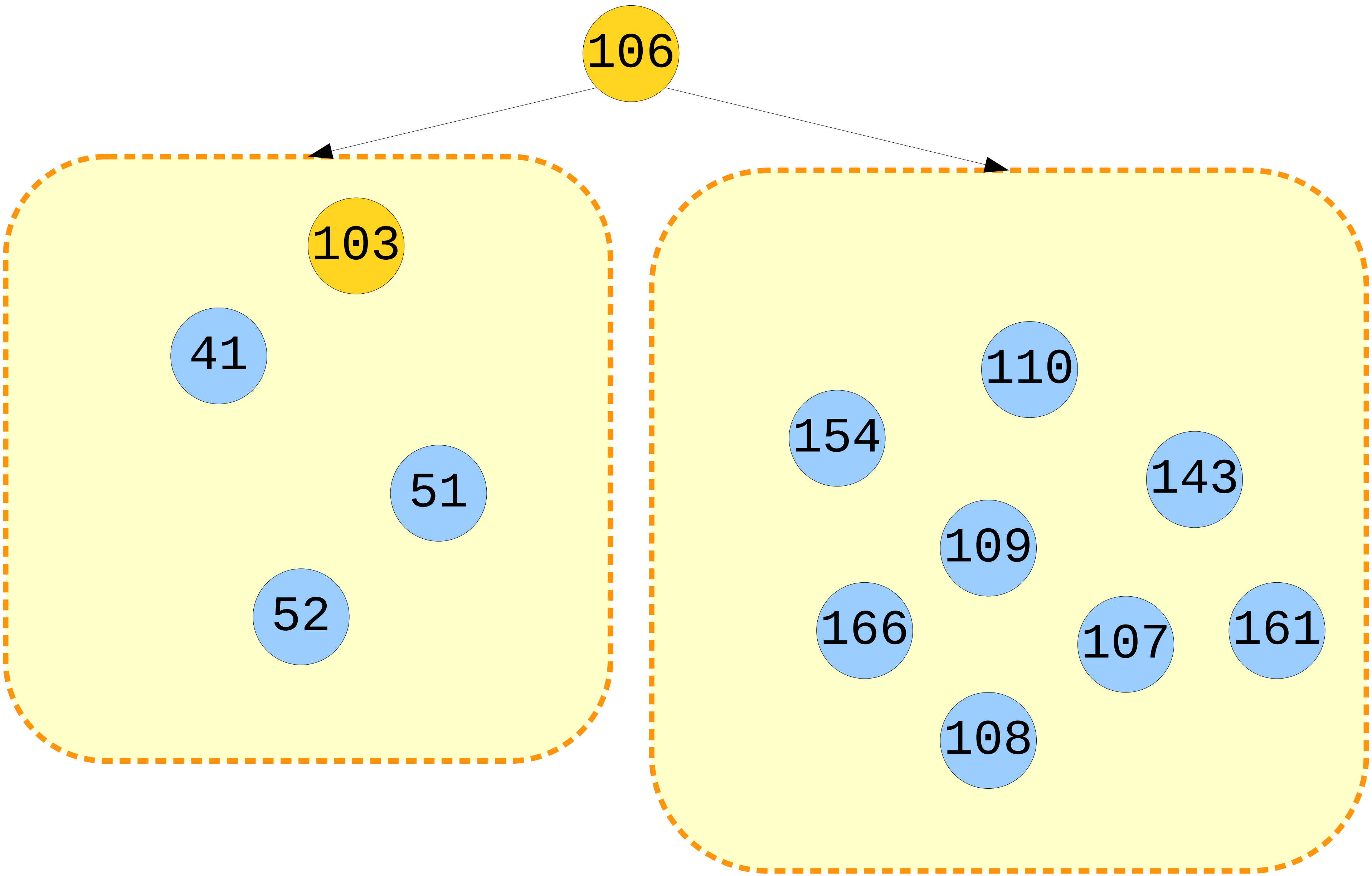

106
166
143
108
110
109
51
41
161
52
154
107
103
Elements less than 106 go here...
… and elements greater than 106 go here.


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.

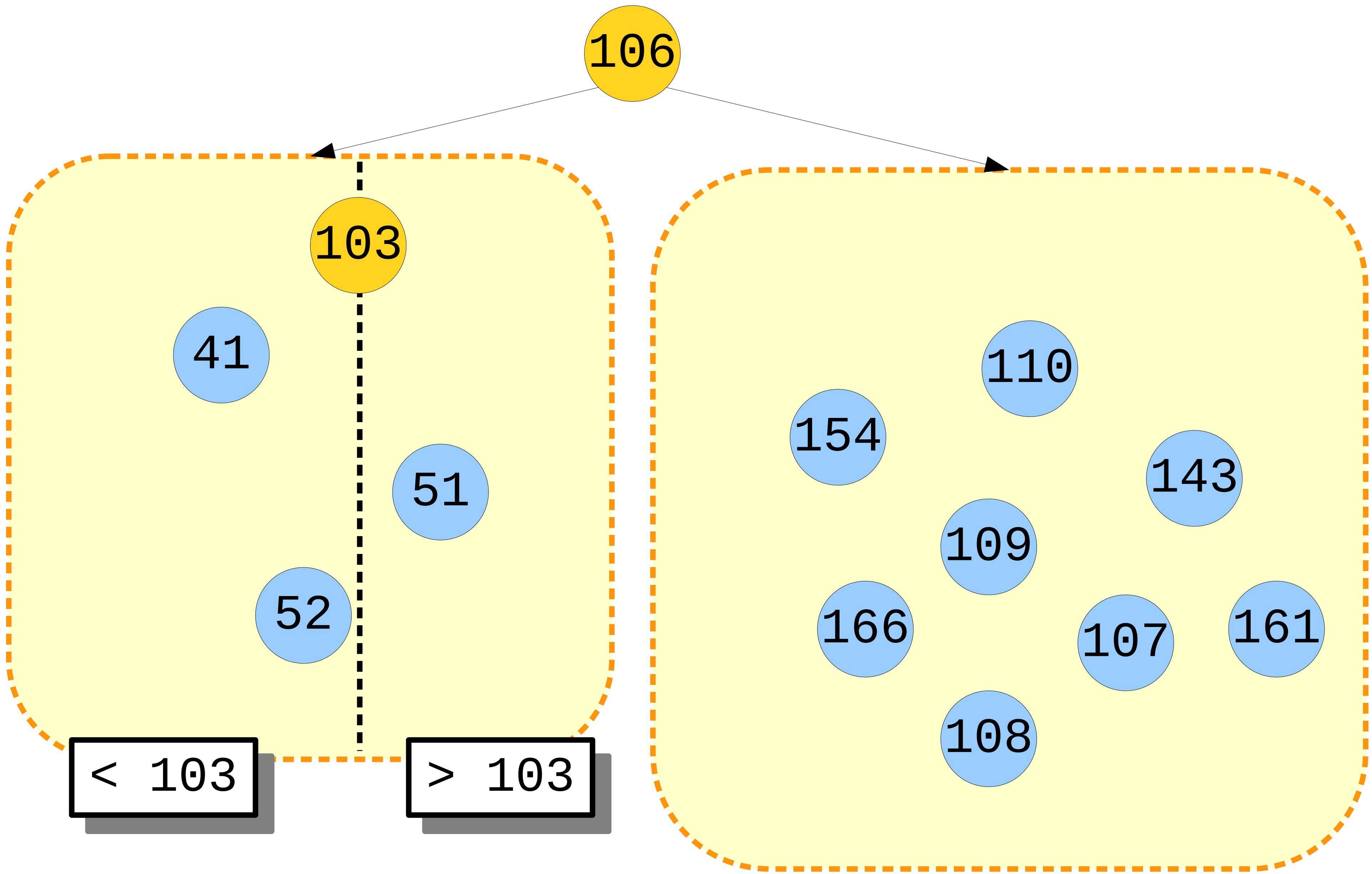

106
41
110
154
103
51
143
109
52
166
107
161
108

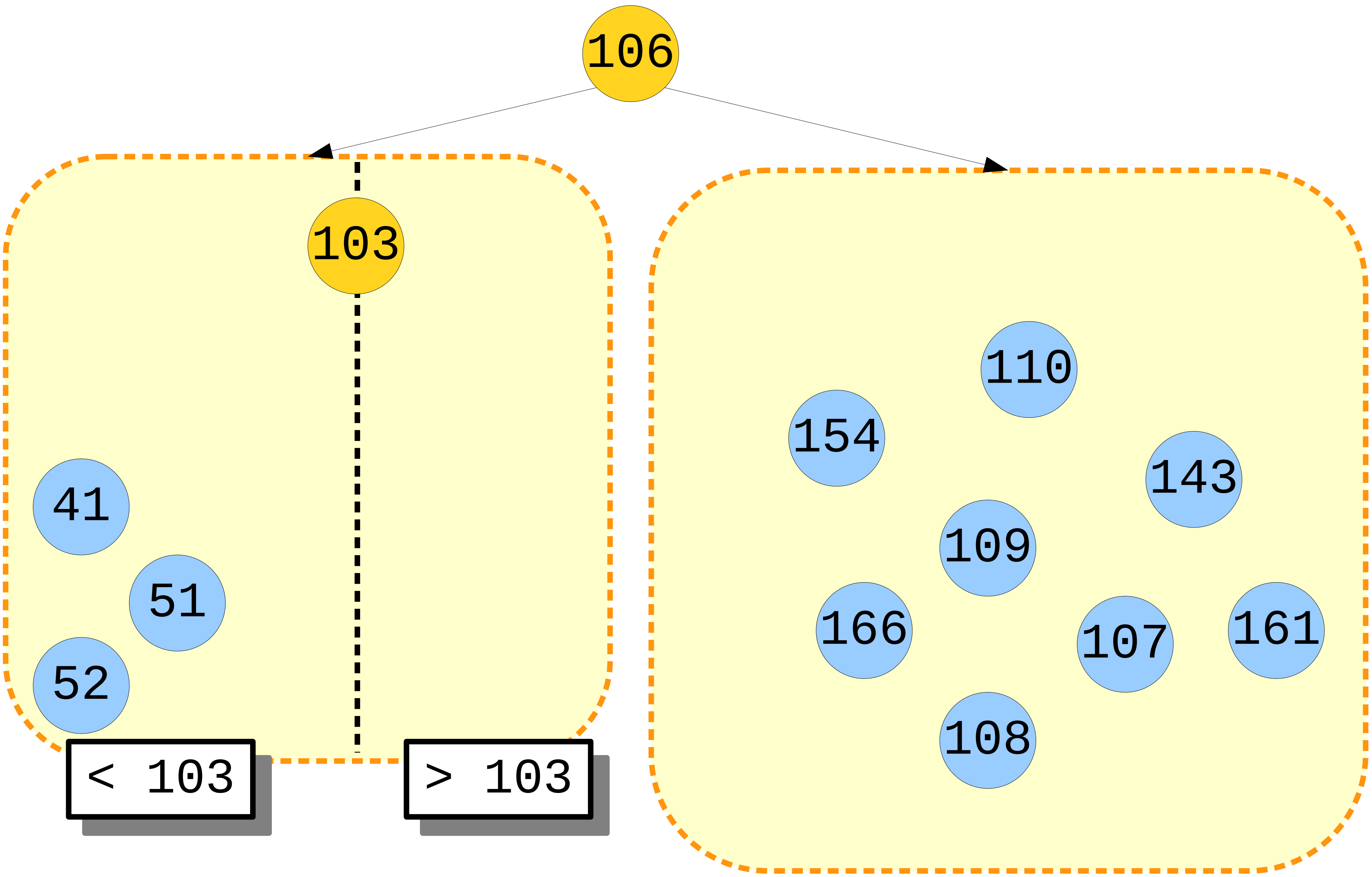

106
41
103
51
52
110
154
143
109
166
107
161
108

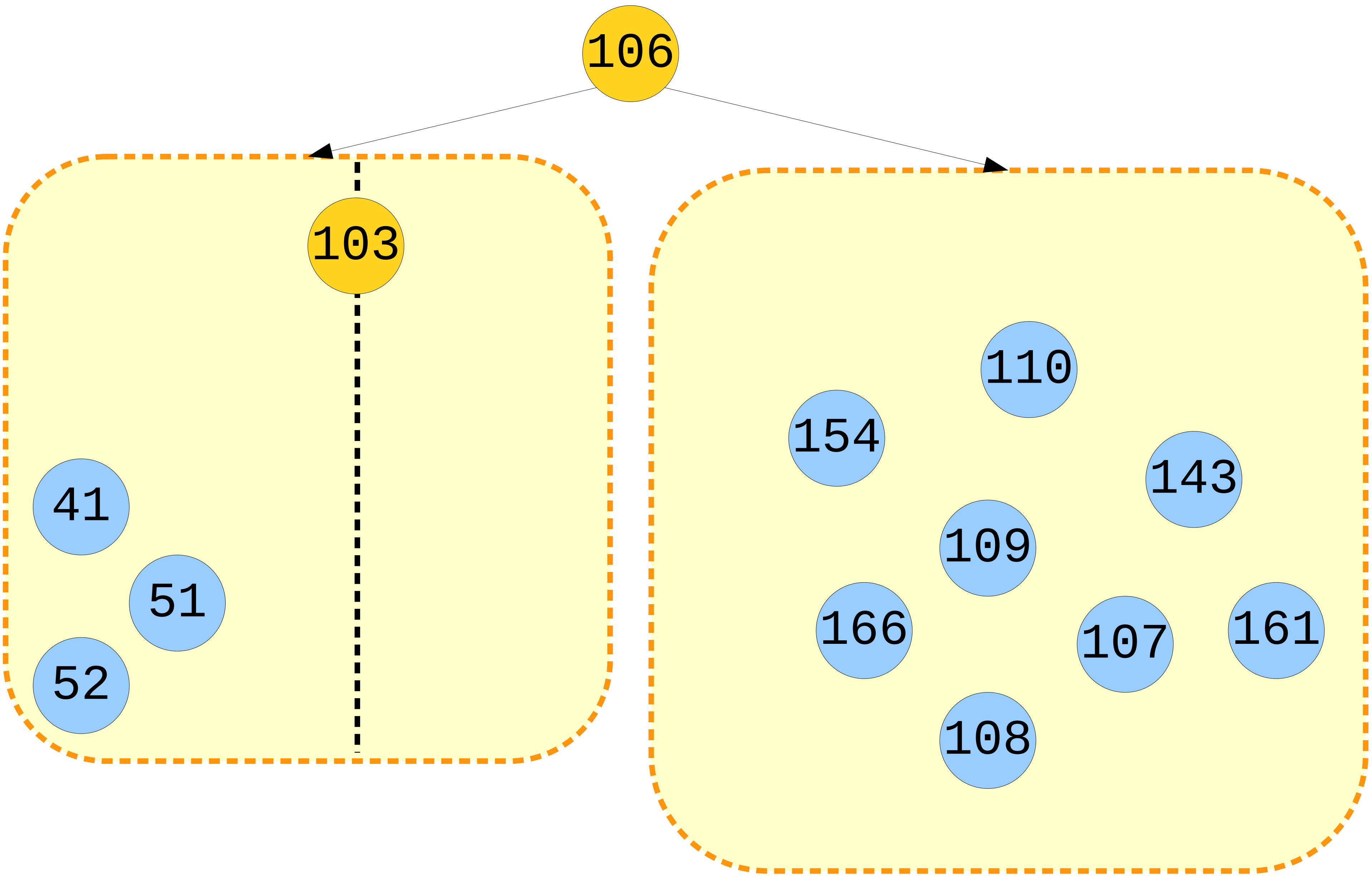

106
41
103
51
52
110
154
143
109
166
107
161
108

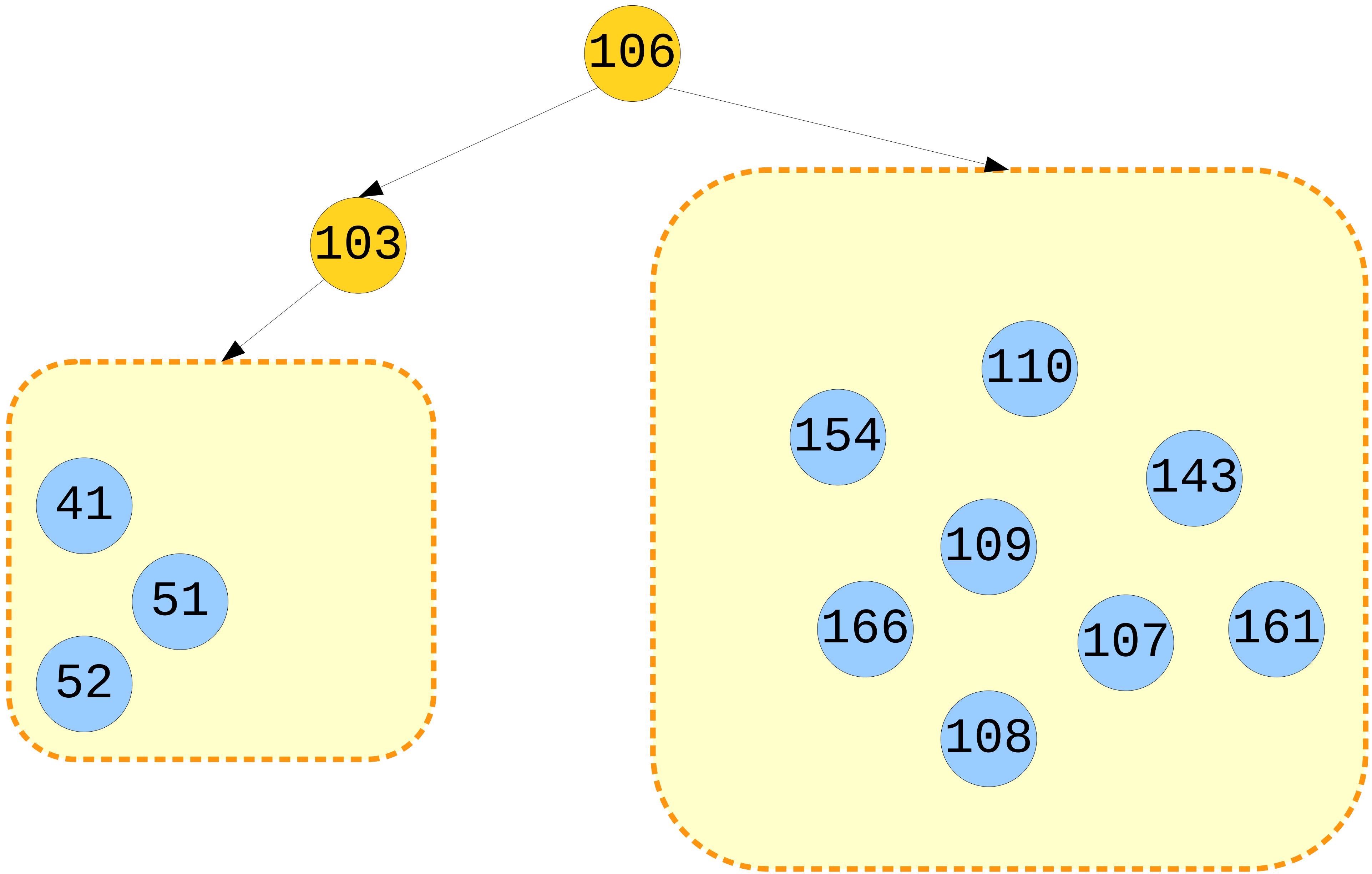


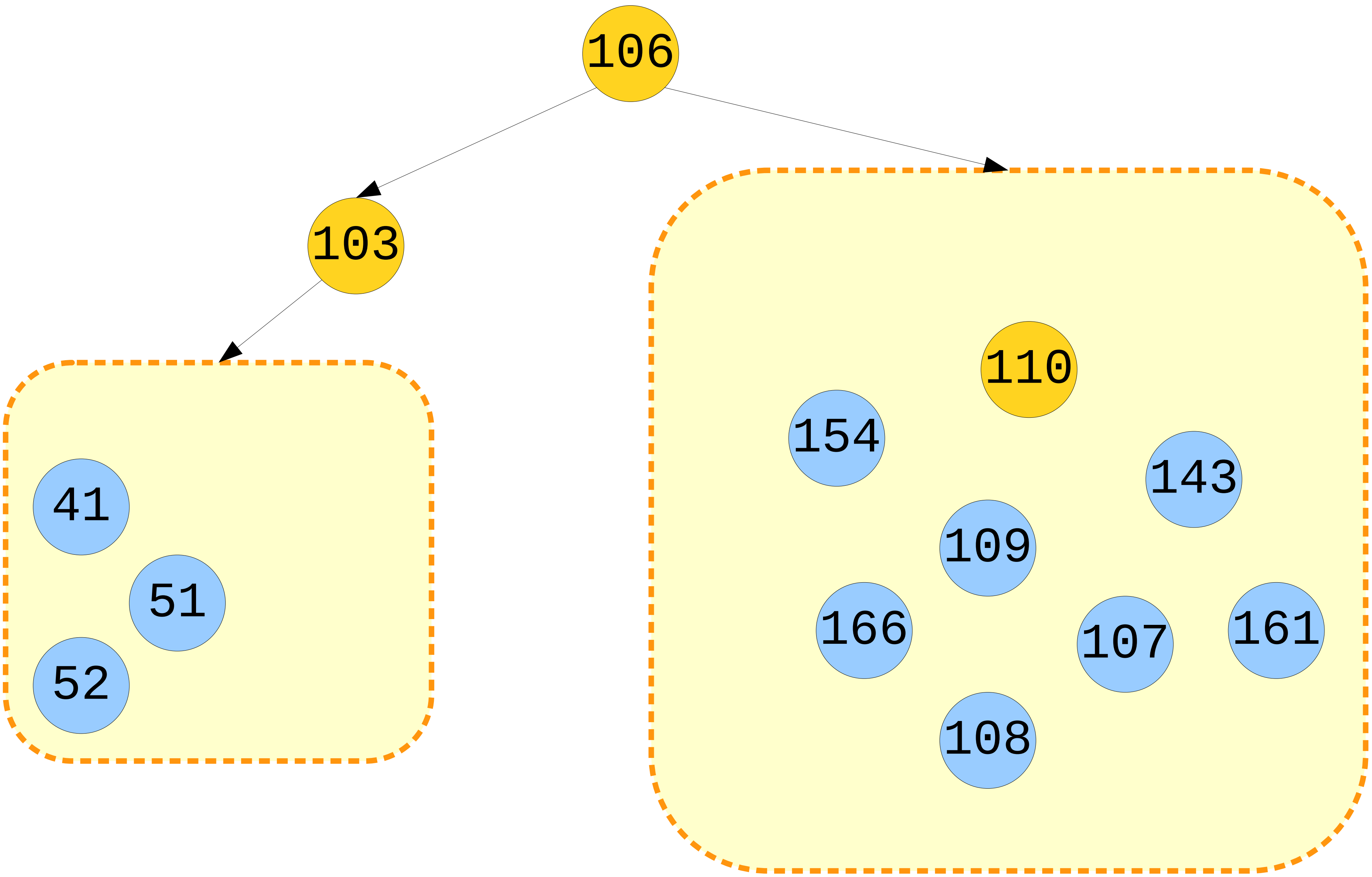
106
103
41
51
52
110
154
143
109
166
107
161
108

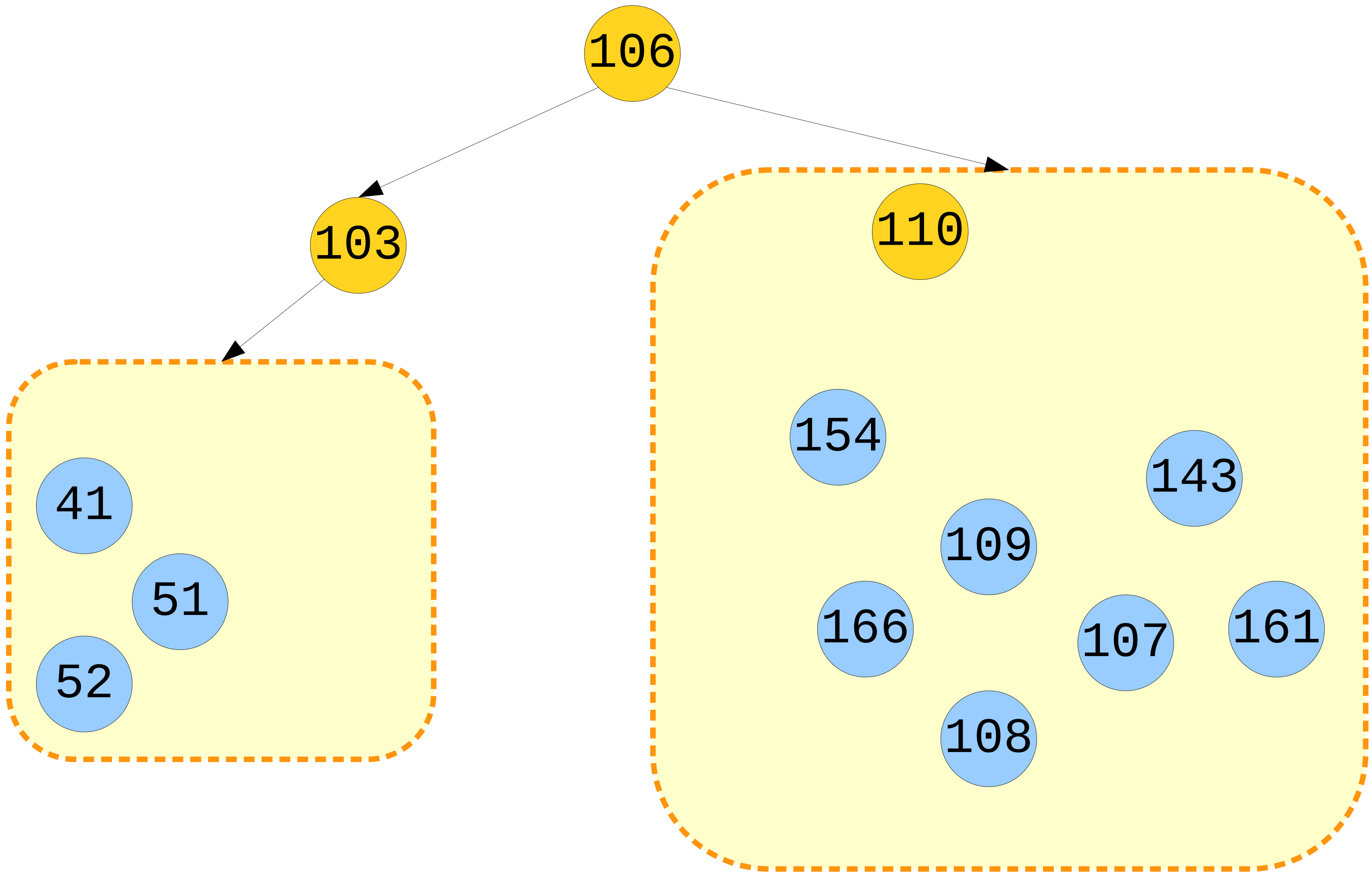

106
103
41
51
52
< 103
> 103
110
154
143
109
166
107
161
108

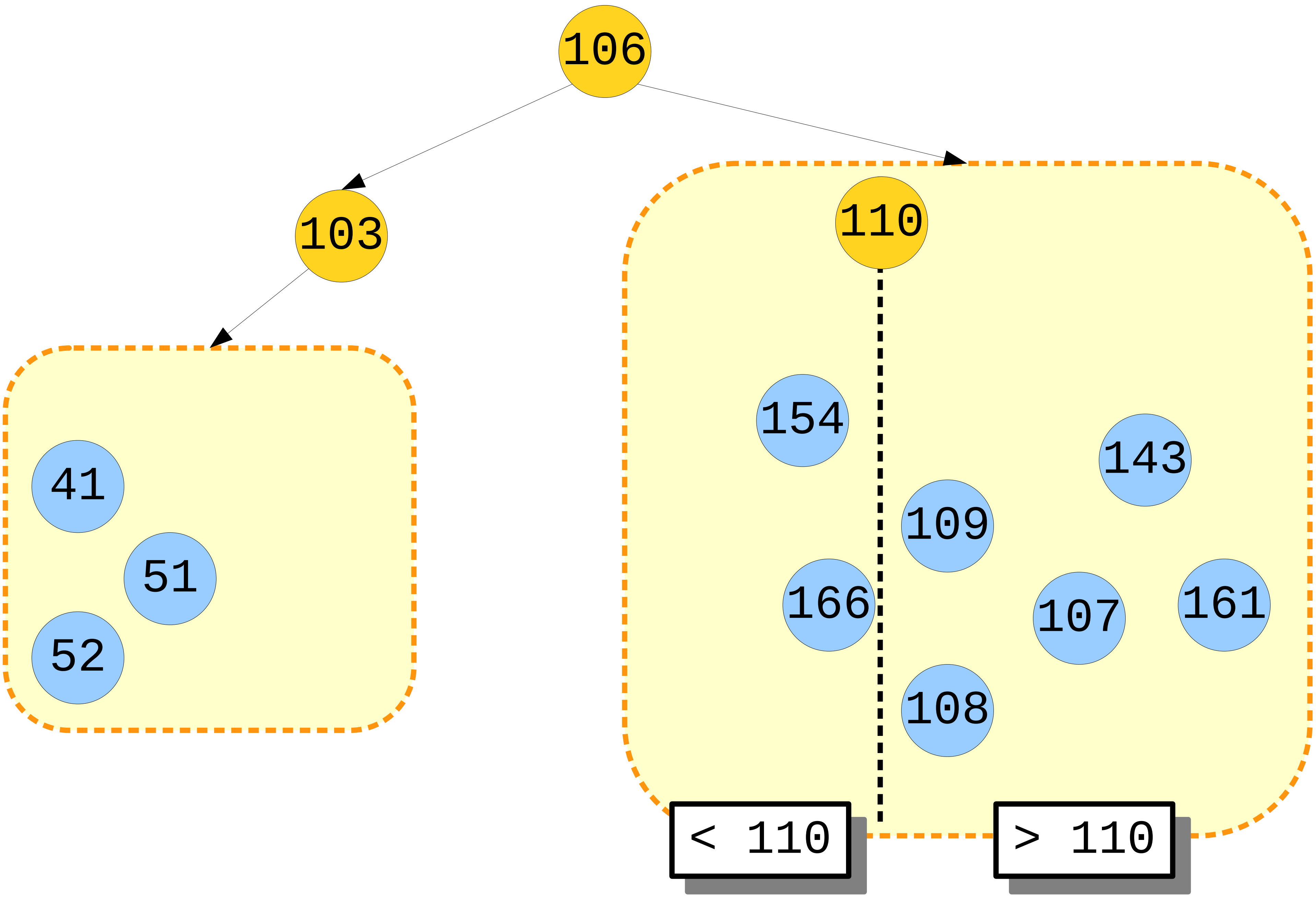

106
103
41
51
52
< 103
> 103
110
154
143
109
166
107
161
108

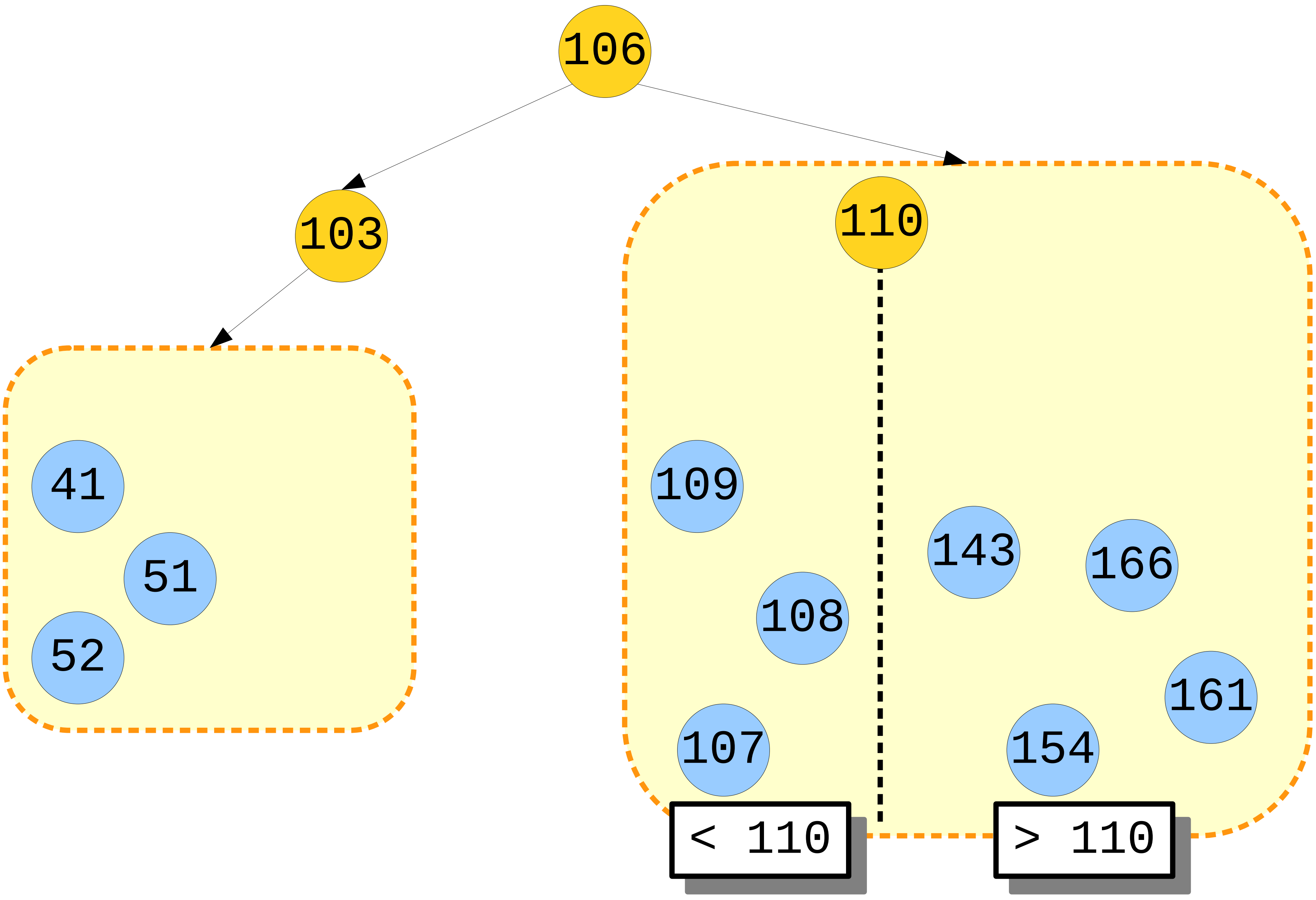

106
103
41
51
52
110
154
143
109
166
107
161
108

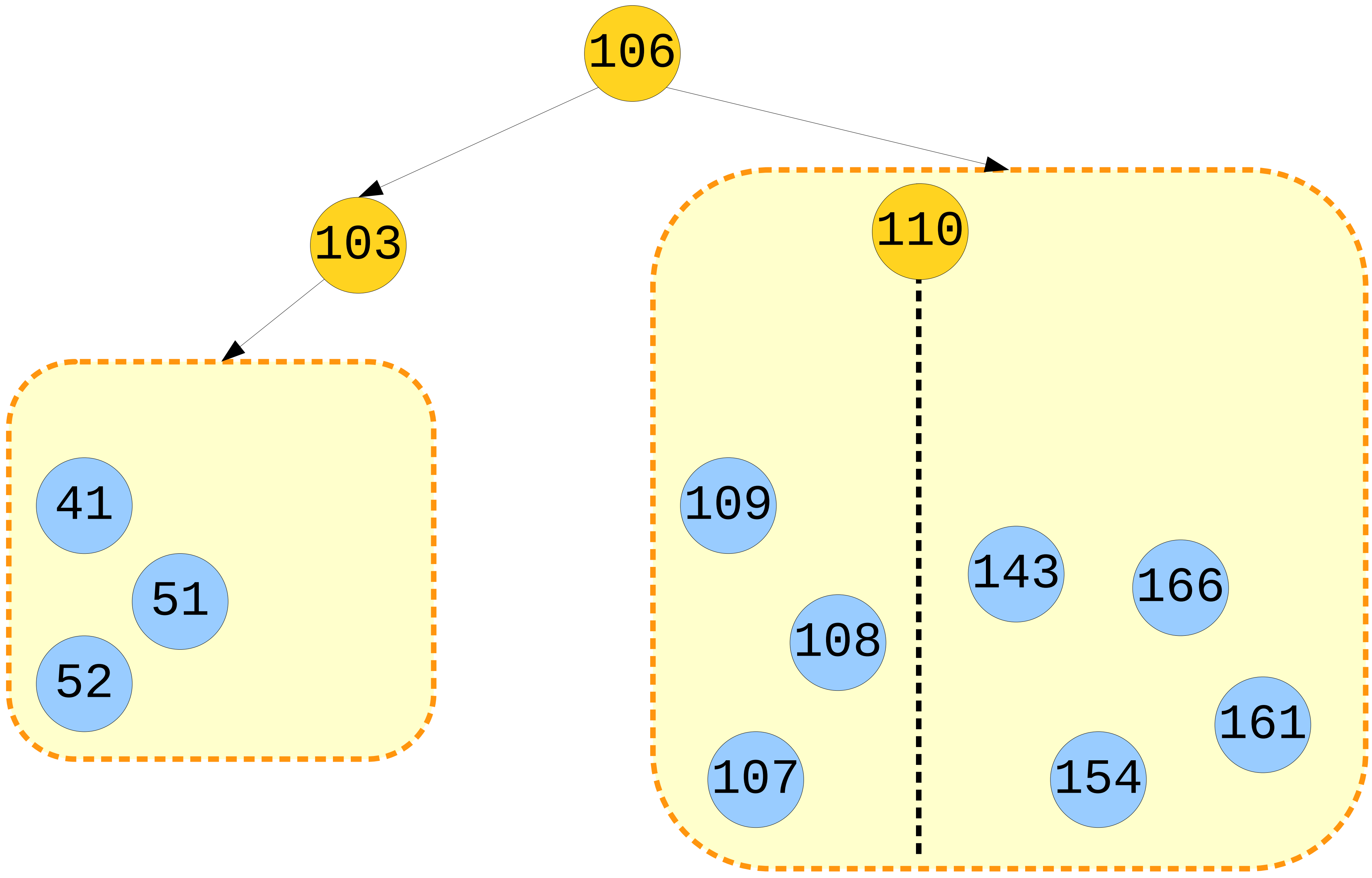

106
103
41
51
52
110
154
143
109
166
107
161
108

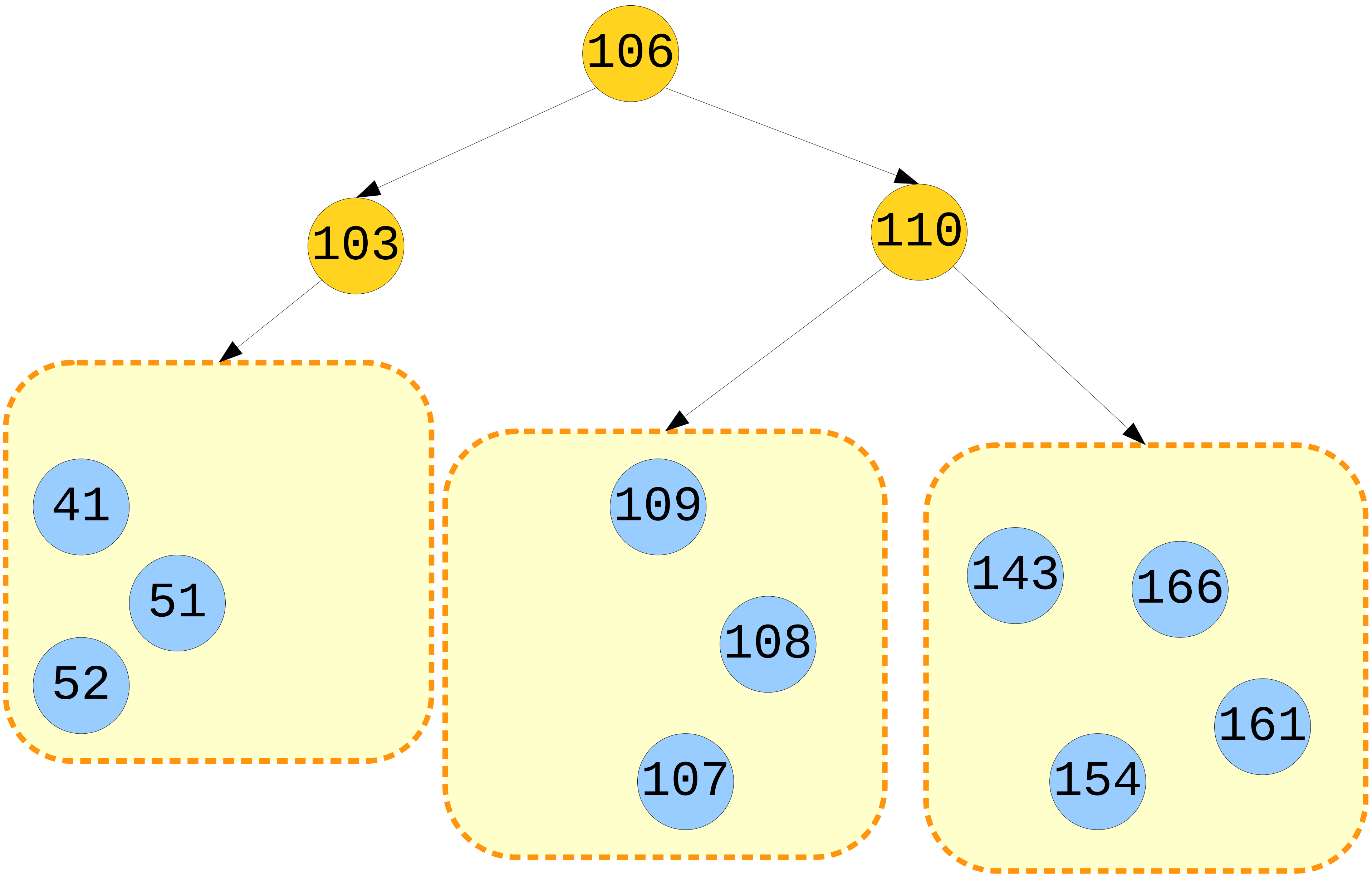

106
103
41
51
52
110
154
143
109
166
107
161
108

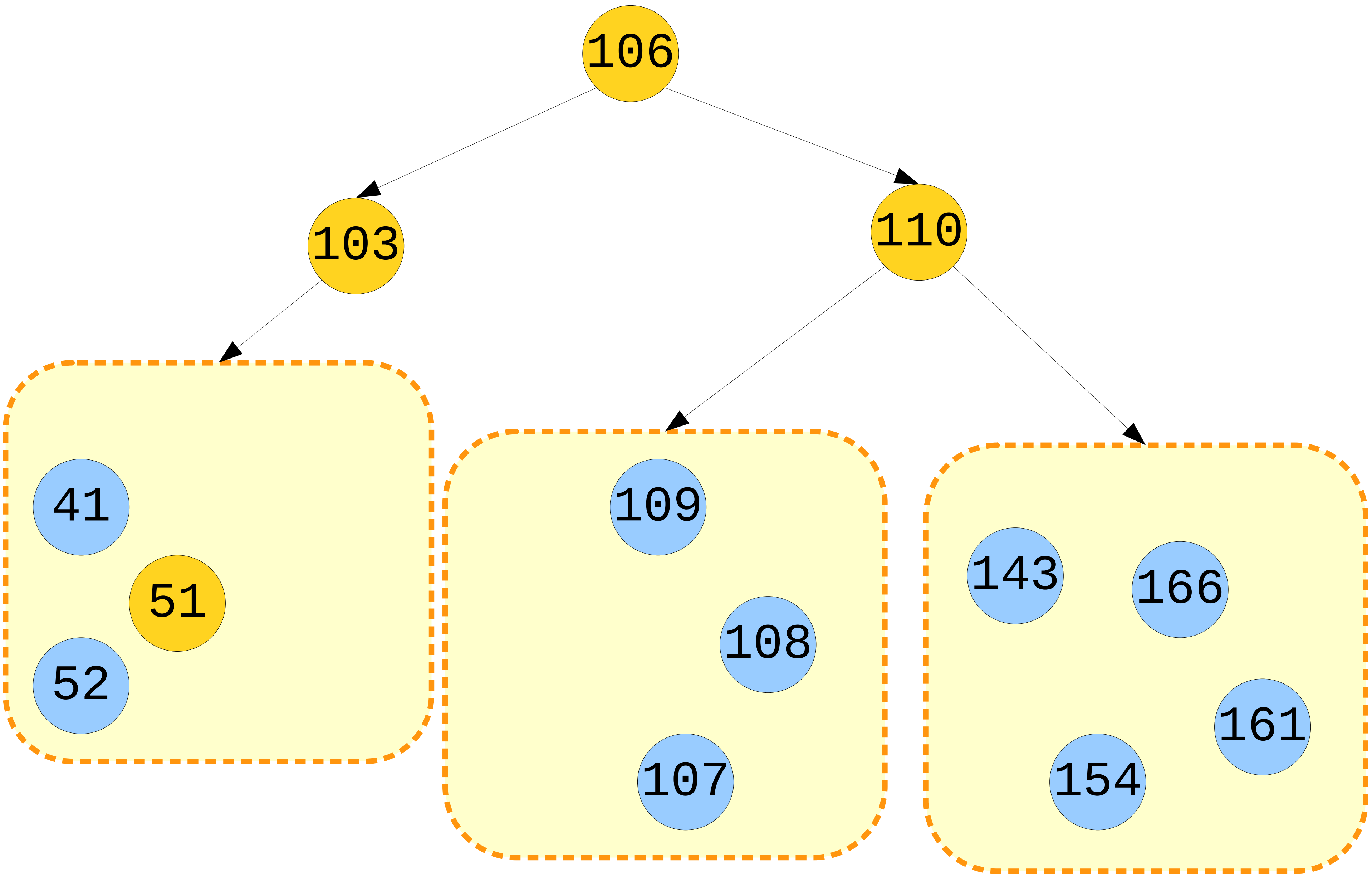

106
103
110
41
51
52
154
143
109
166
107
161
108

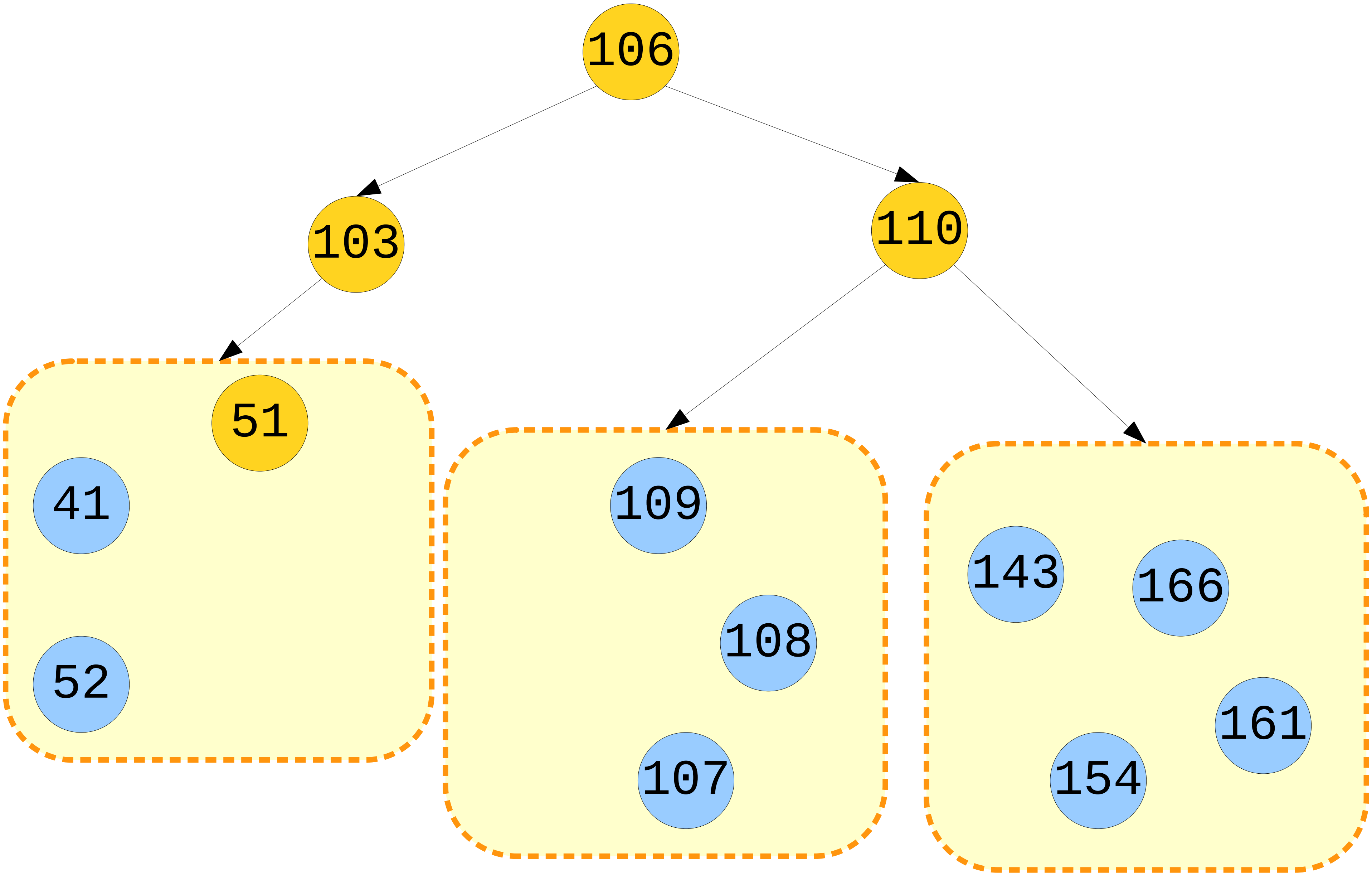

106
103
110
41
51
52
154
166
109
108
143
107
161
< 110
> 110

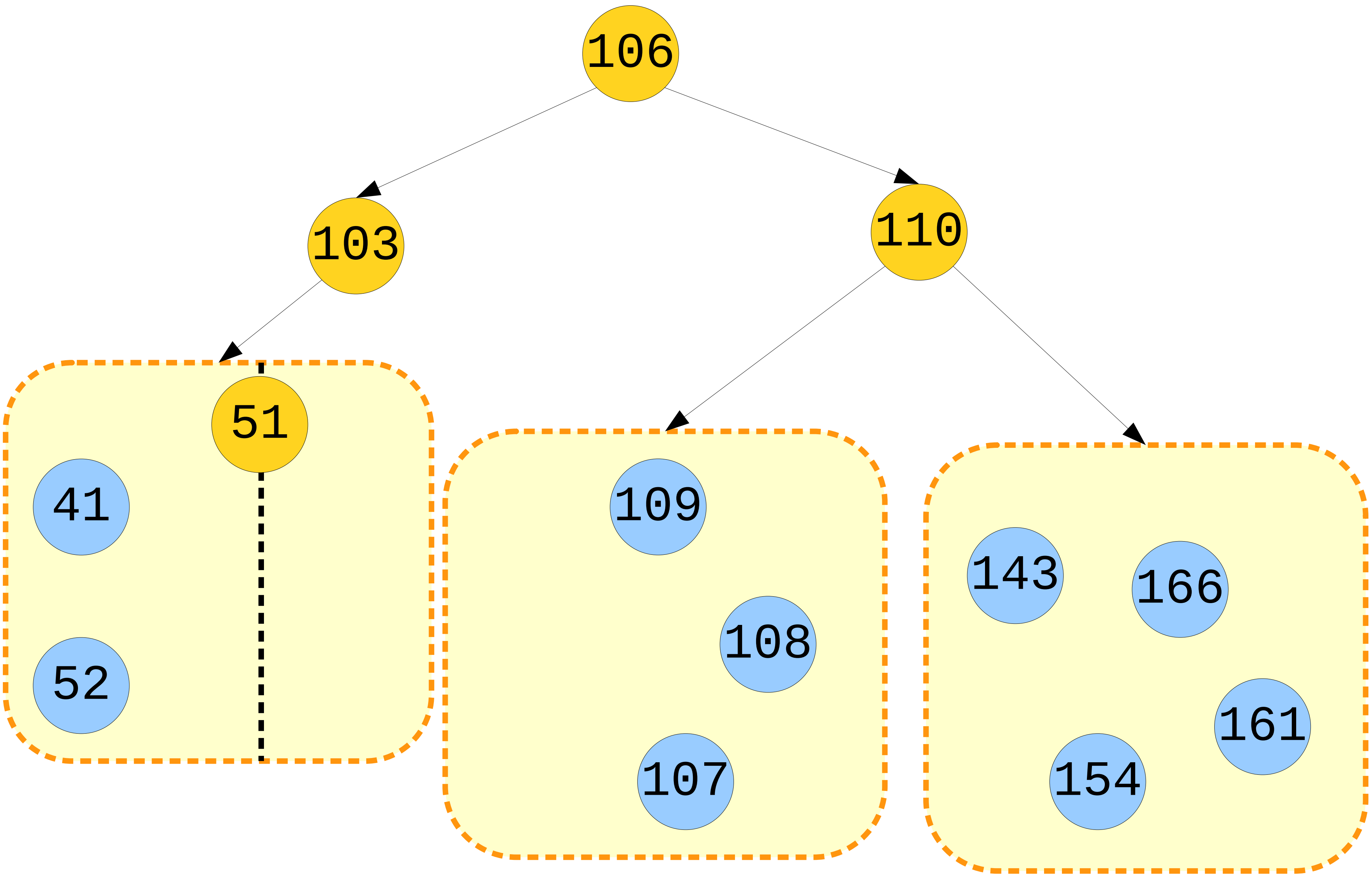

106
103
110
41
51
52
109
108
107
143
166
161
154
< 110
> 110

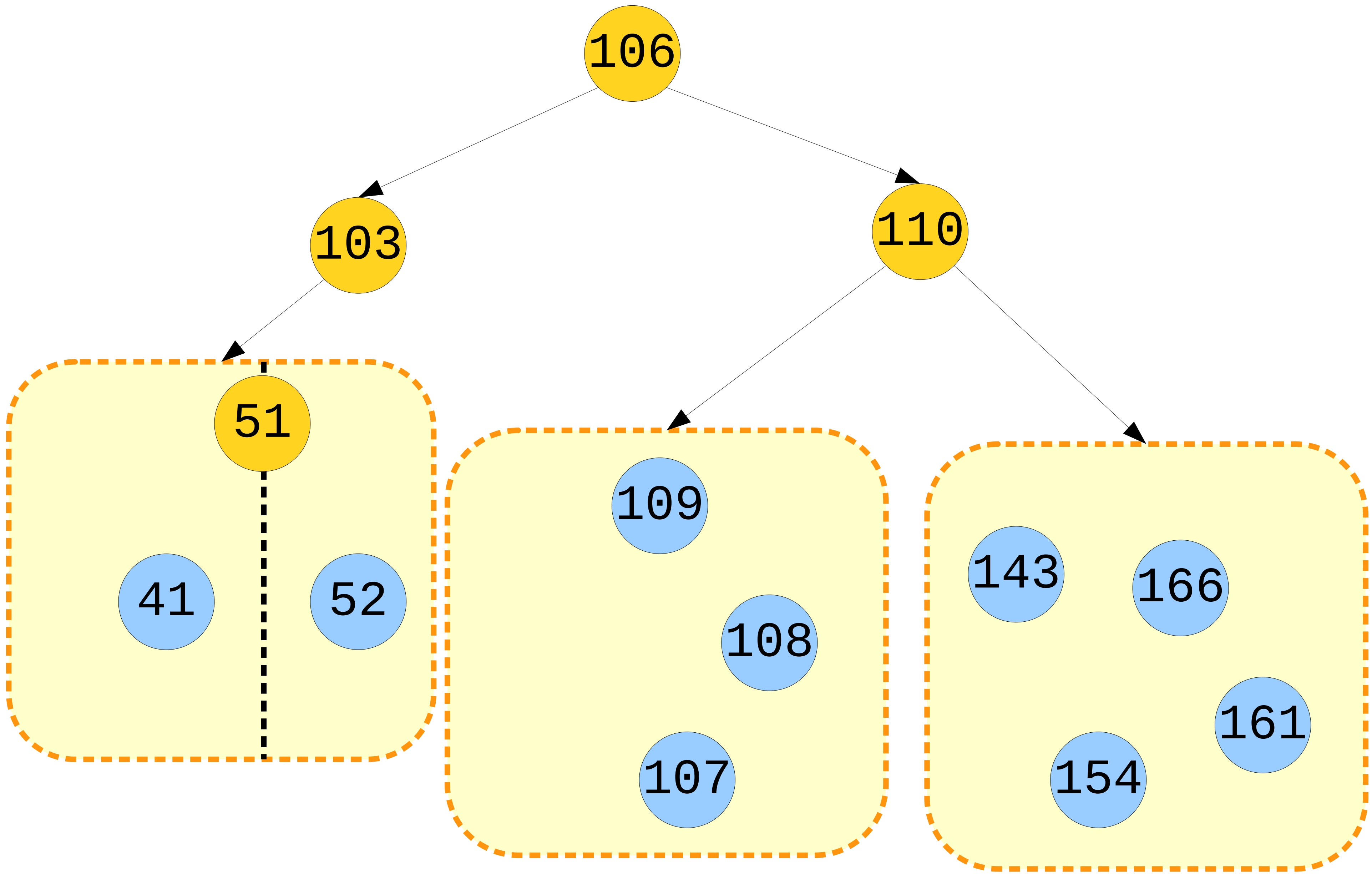

106
103
110
41
51
52
109
108
107
143
166
161
154

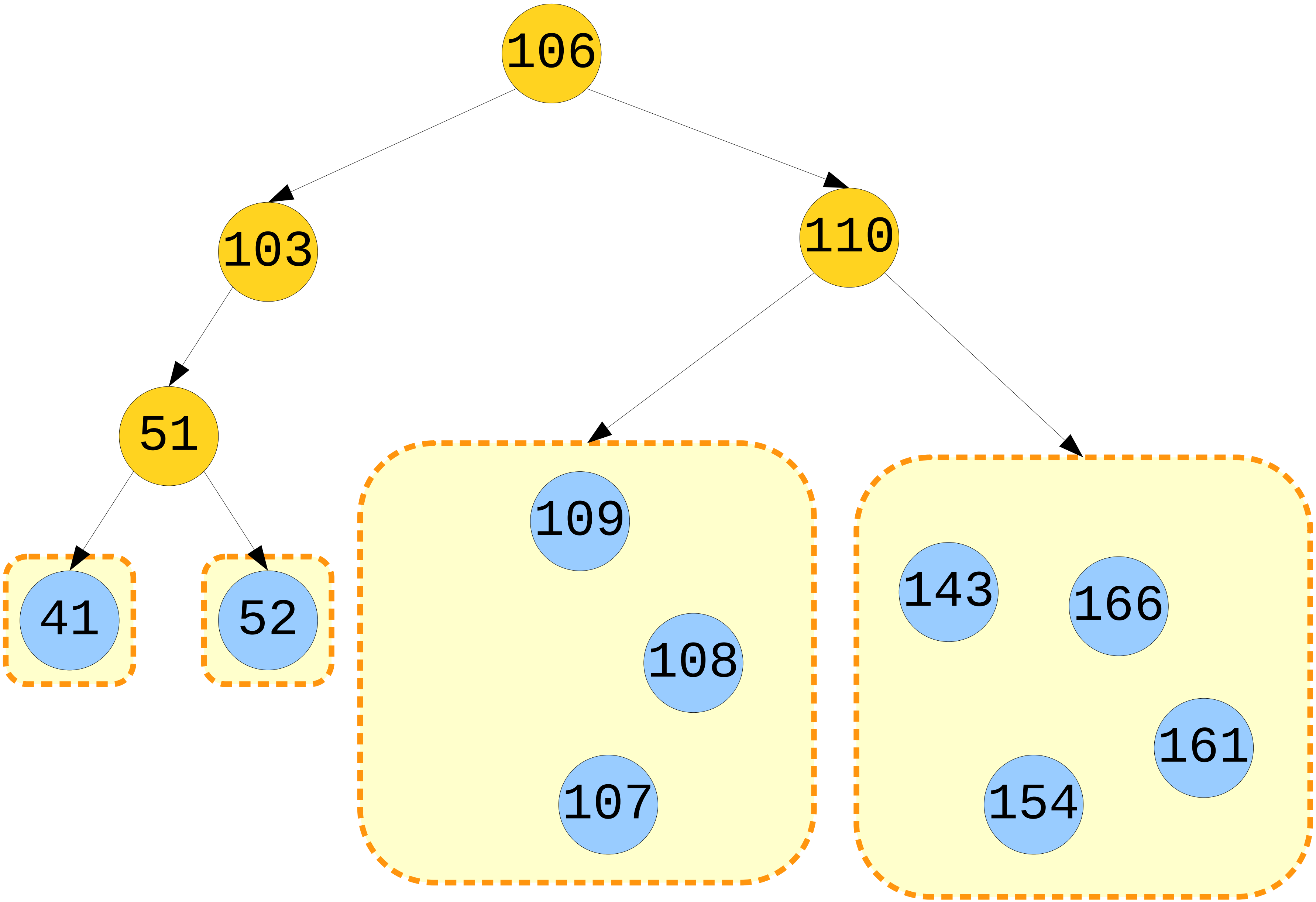

106
103
110
41
51
52
109
108
107
143
166
161
154

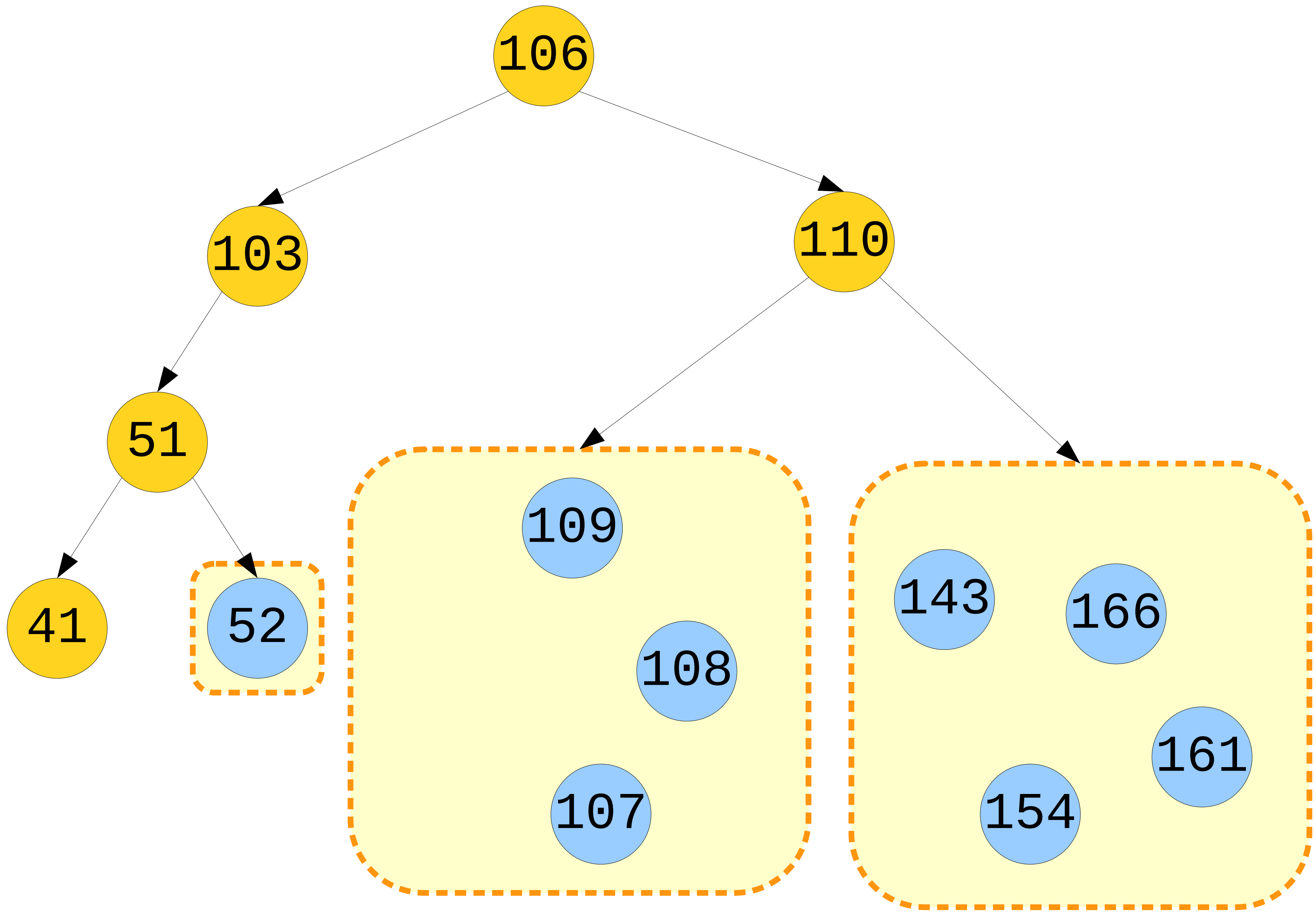

106
103
110
41
51
52
109
108
107
143
166
161
154

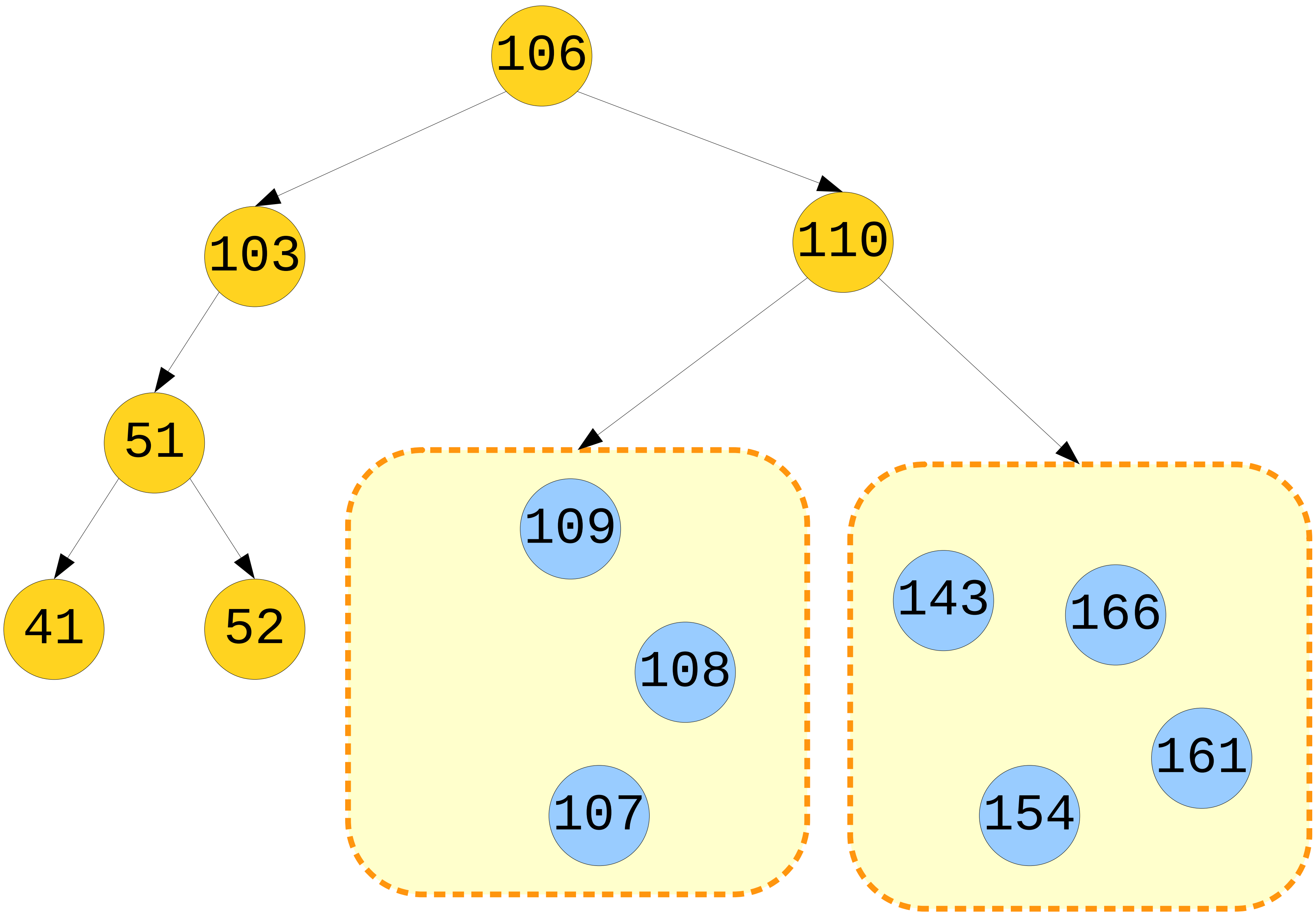

106
103
110
51
41
52
109
108
107
143
166
161
154


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

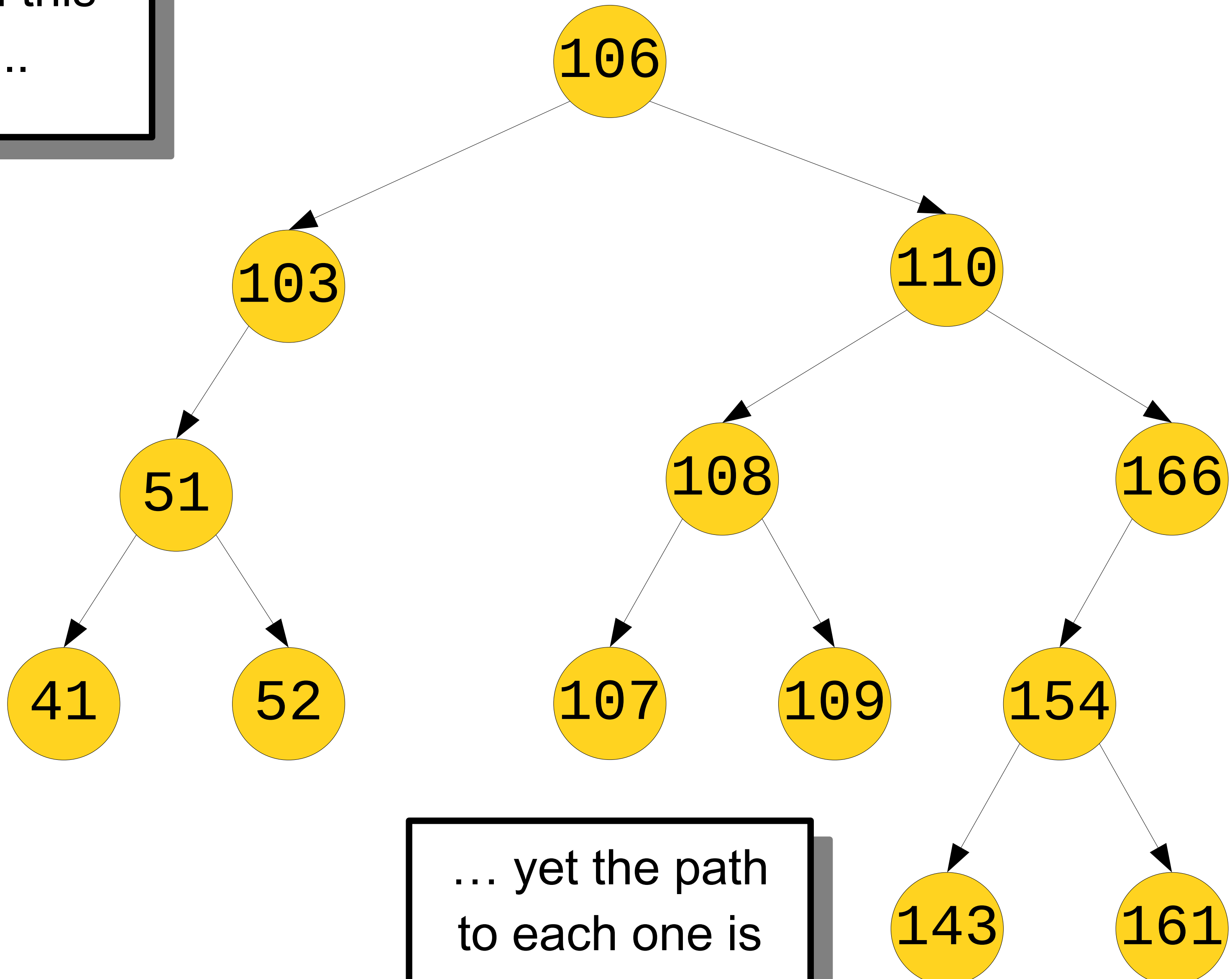

106
103
110
51
41
52
109
108
107
143
166
161
154


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

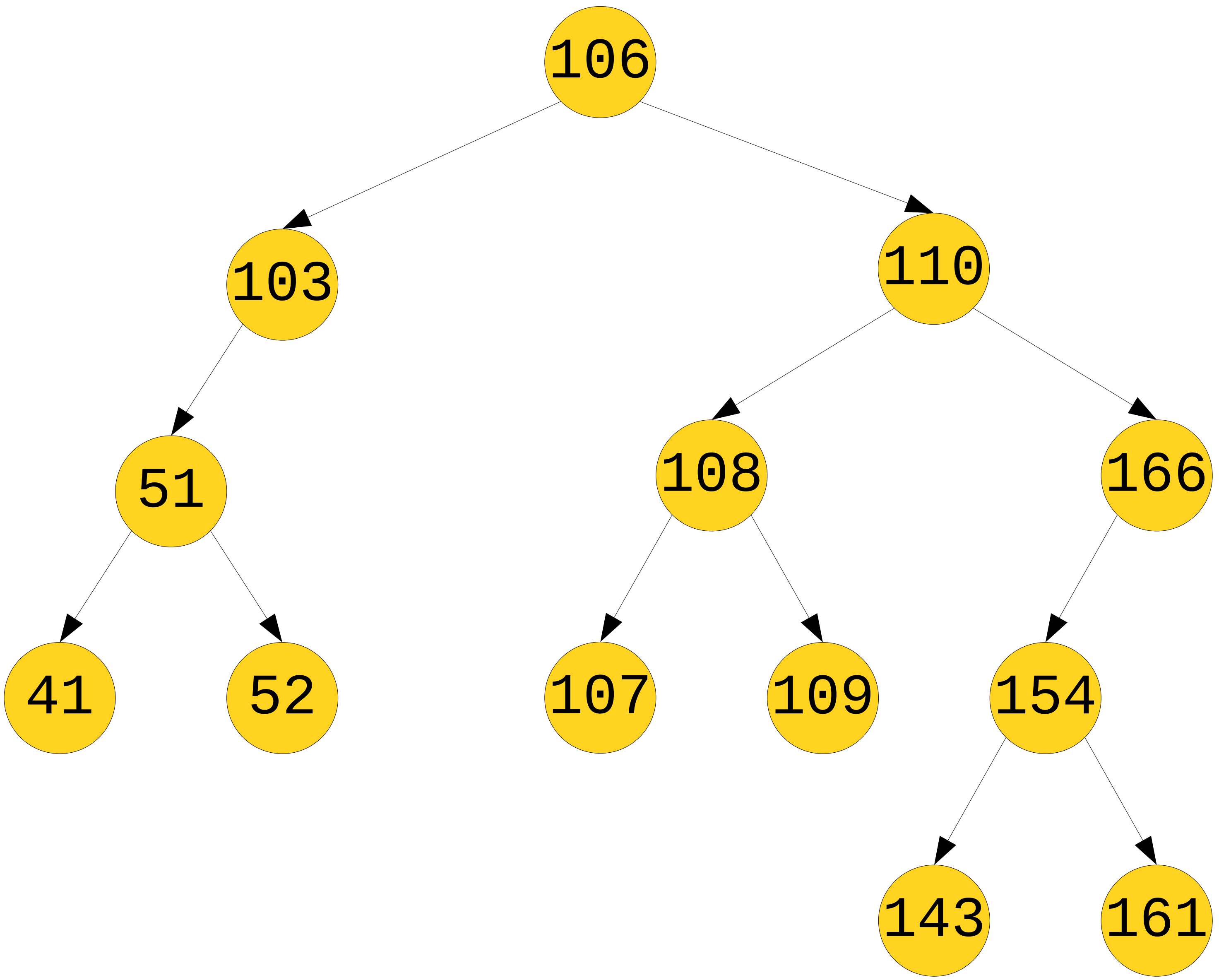

106
103
110
51
109
41
52
143
166
108
161
107
154

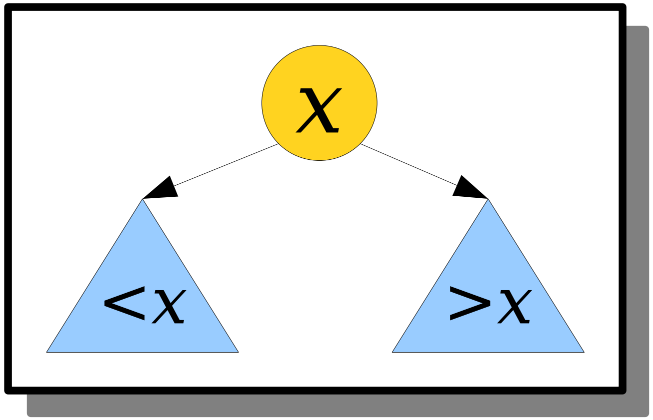

106
103
110
51
41
52
109
108
107
143
166
161
154

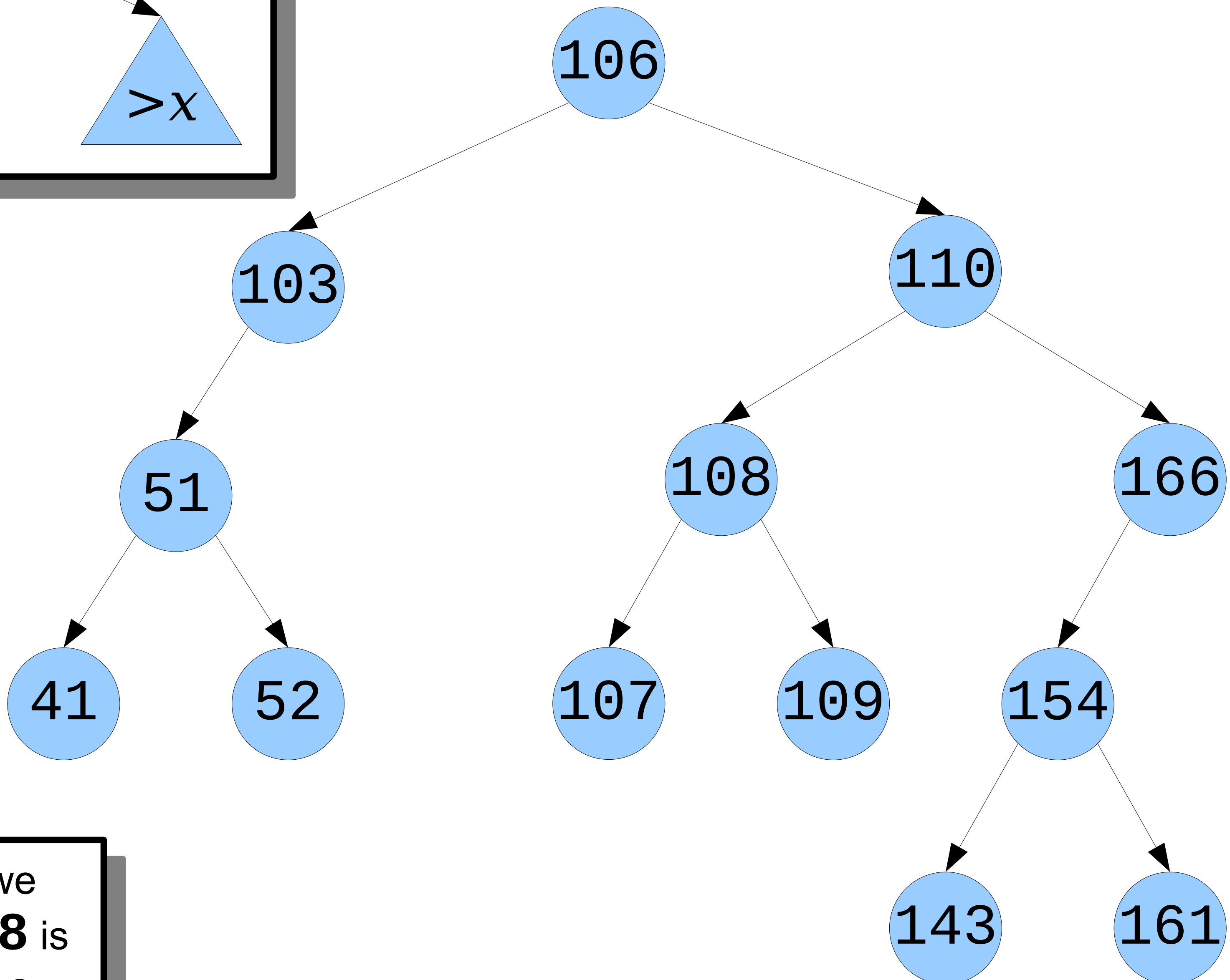
There are 13 ***nodes*** in this tree...



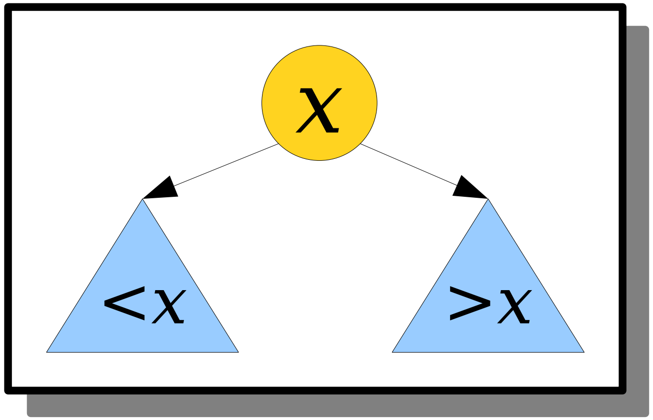
… yet the path to each one is short.

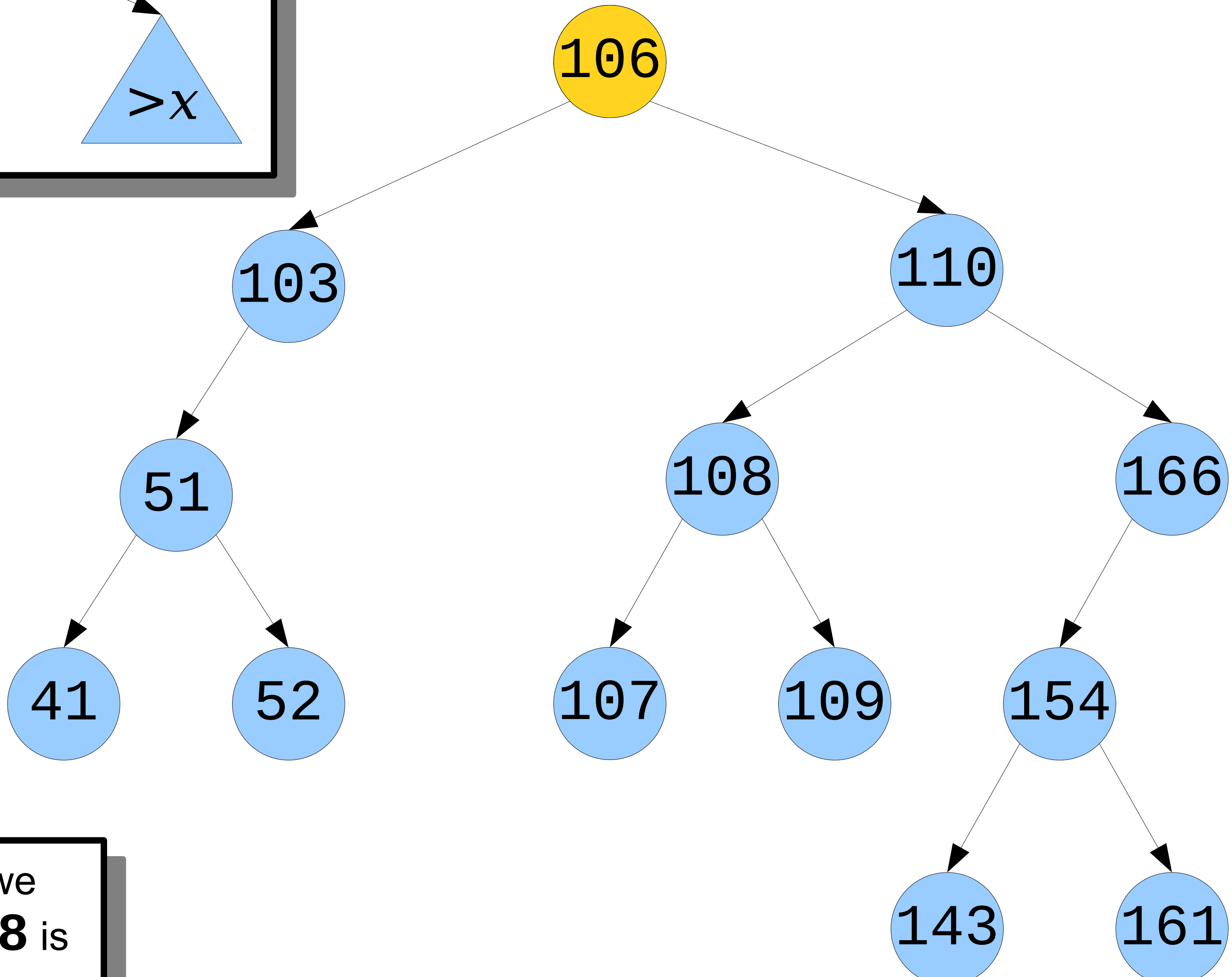

106
103
110
51
108
166
41
52
107
109
154
143
161



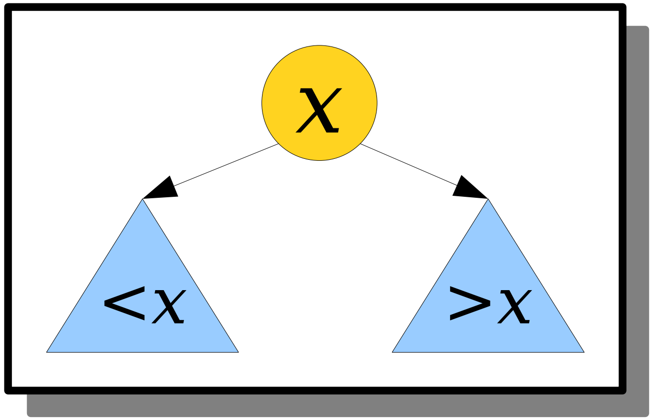


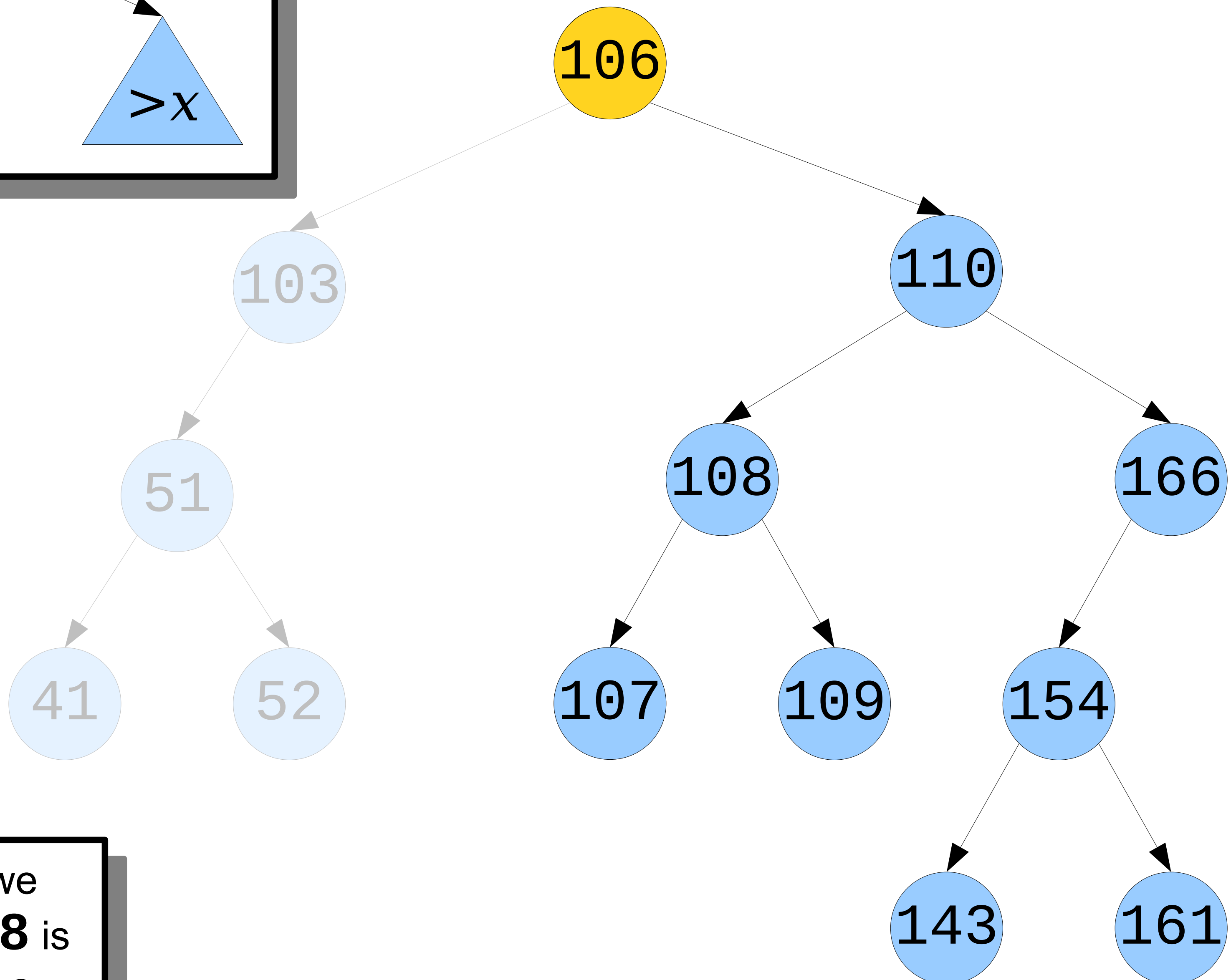
How can we check if **108** is in this tree?



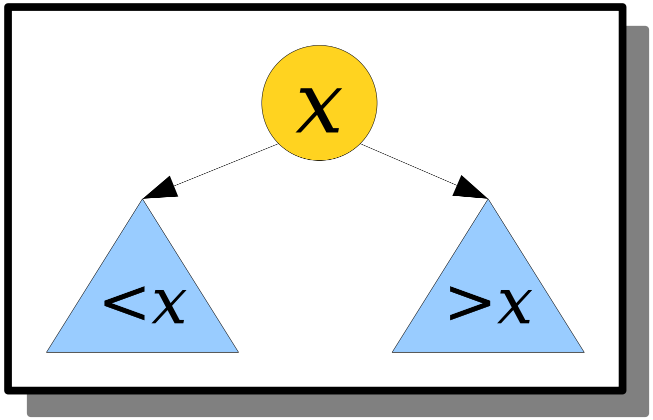


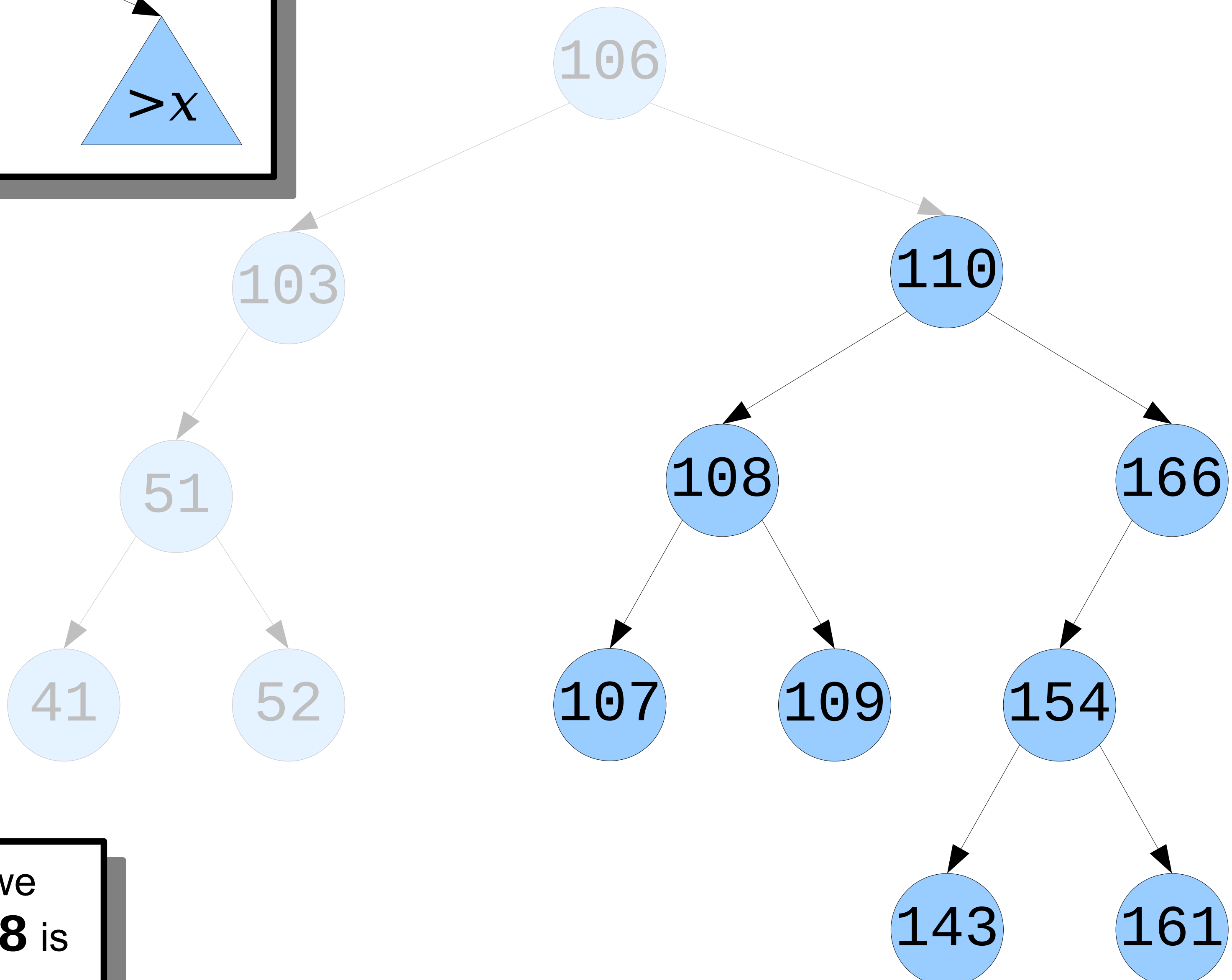
How can we check if **108** is in this tree?



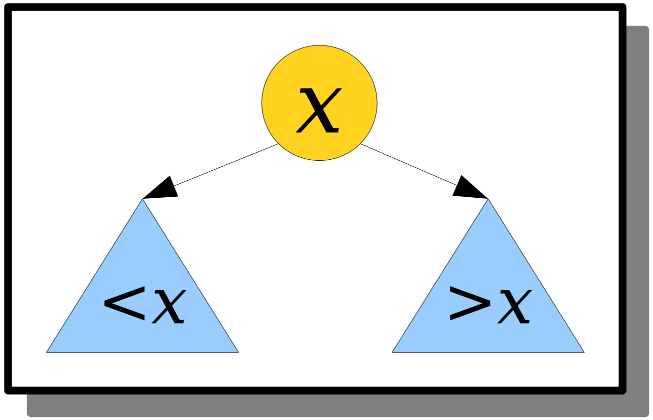


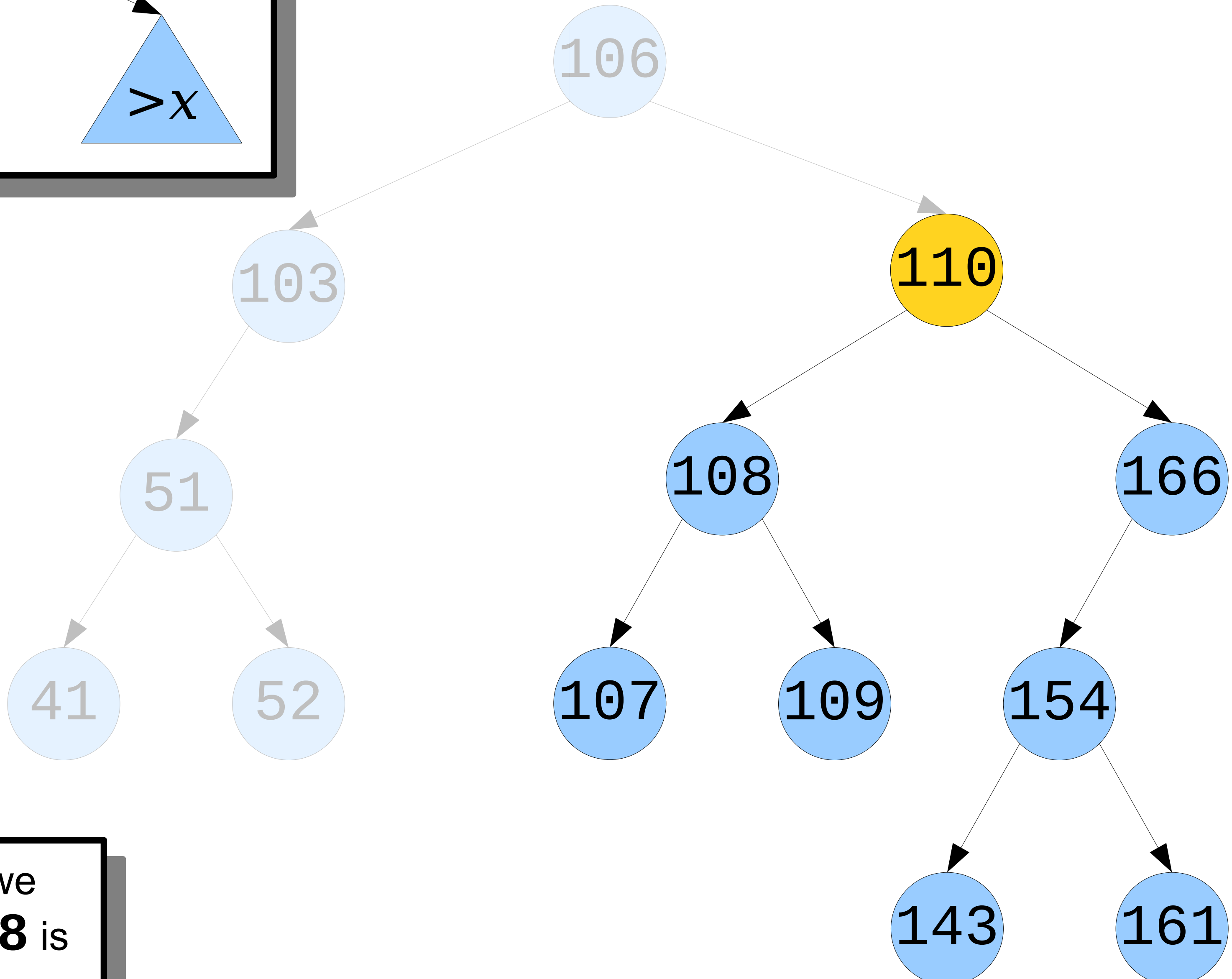
How can we check if **108** is in this tree?



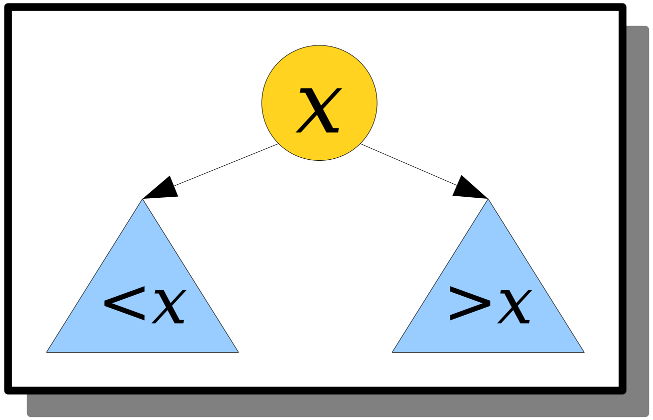


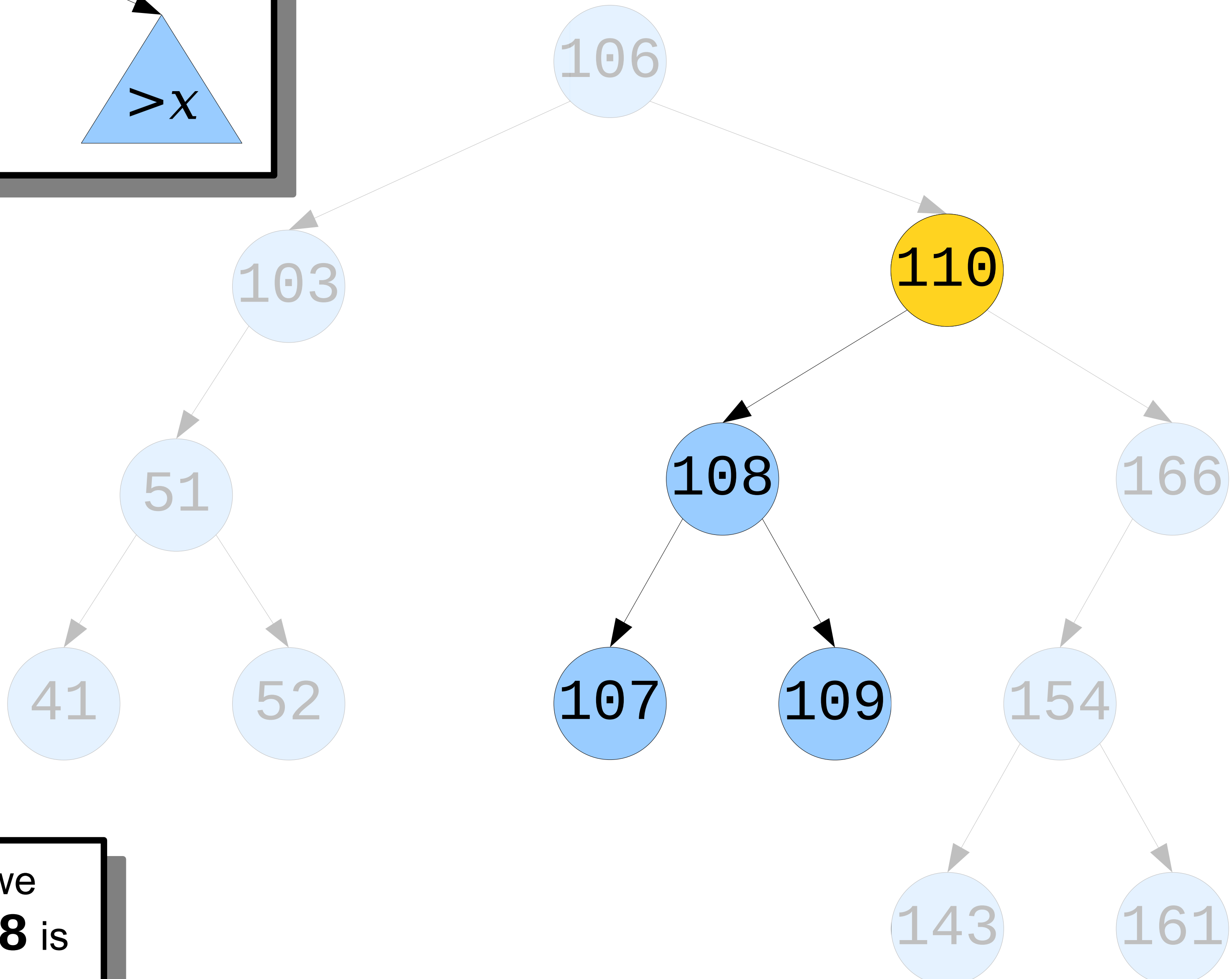
How can we check if **108** is in this tree?



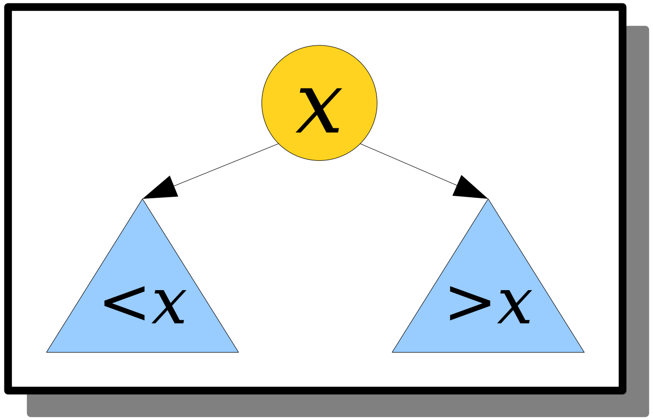


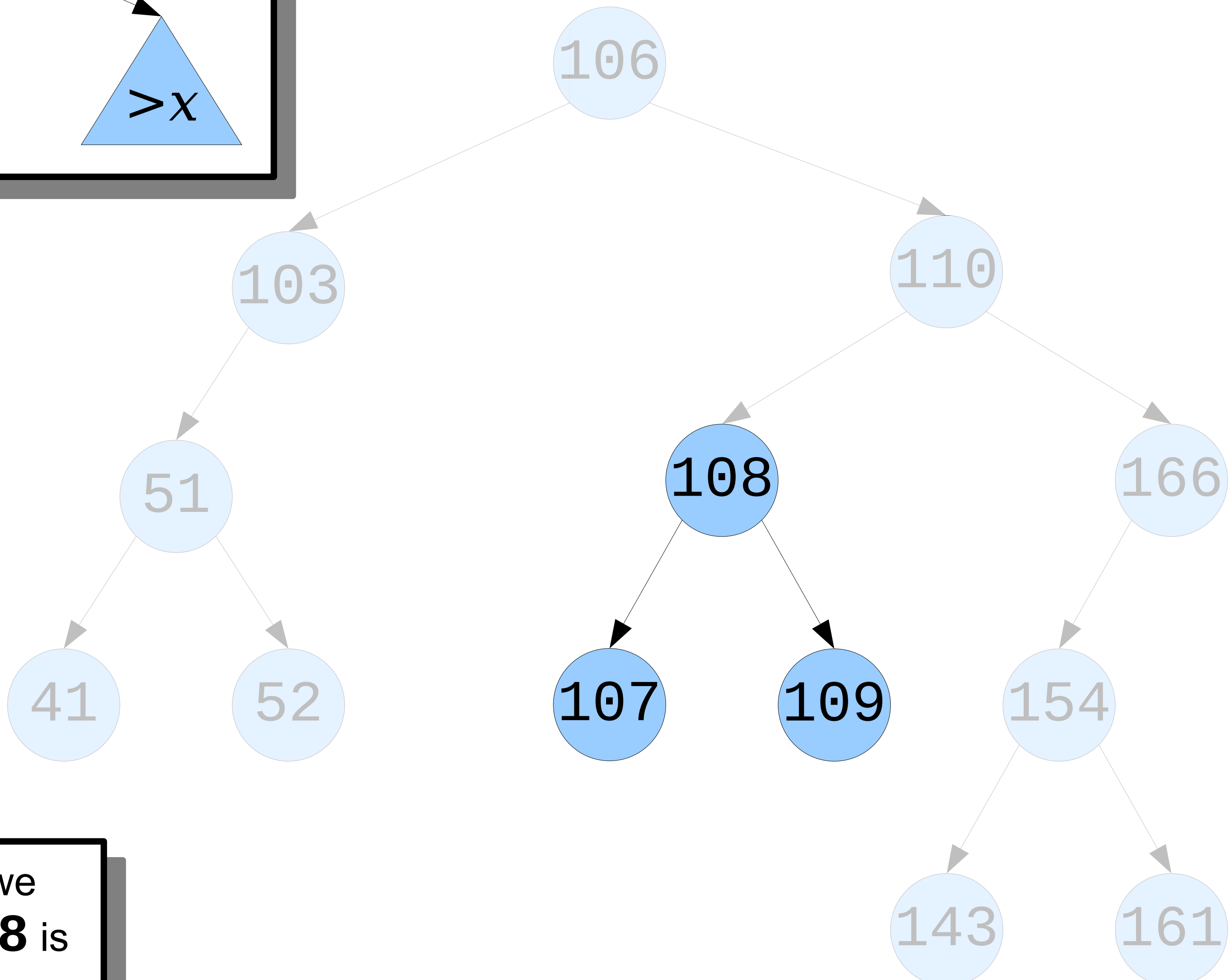
How can we check if **108** is in this tree?



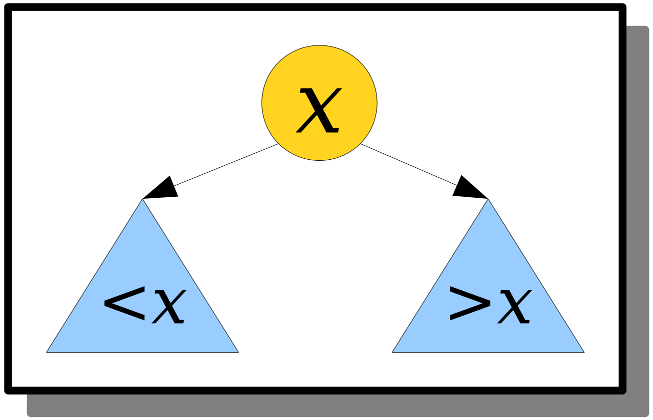


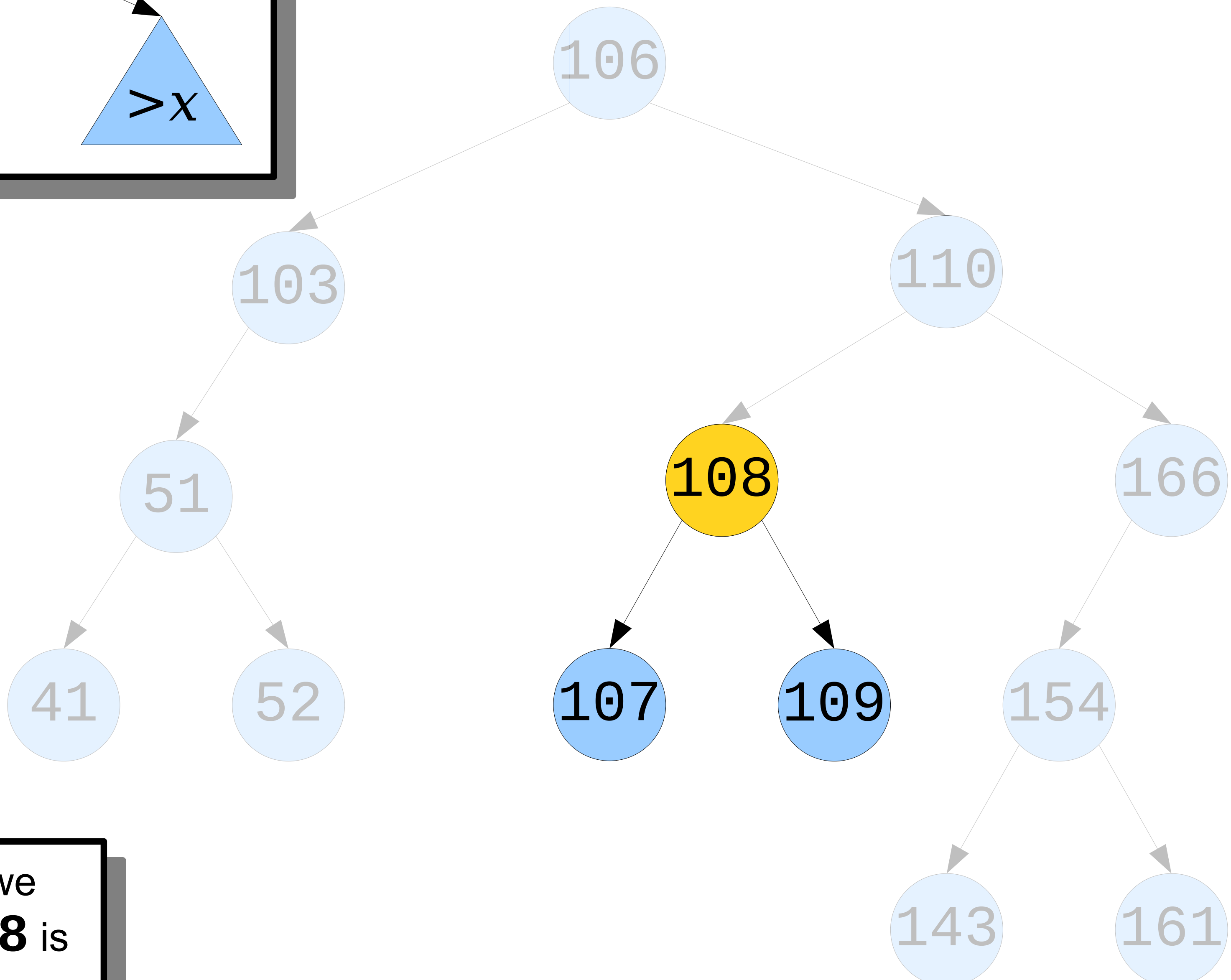
How can we check if **108** is in this tree?



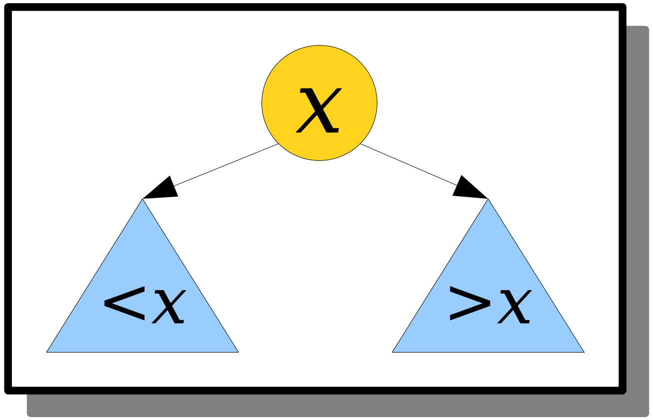


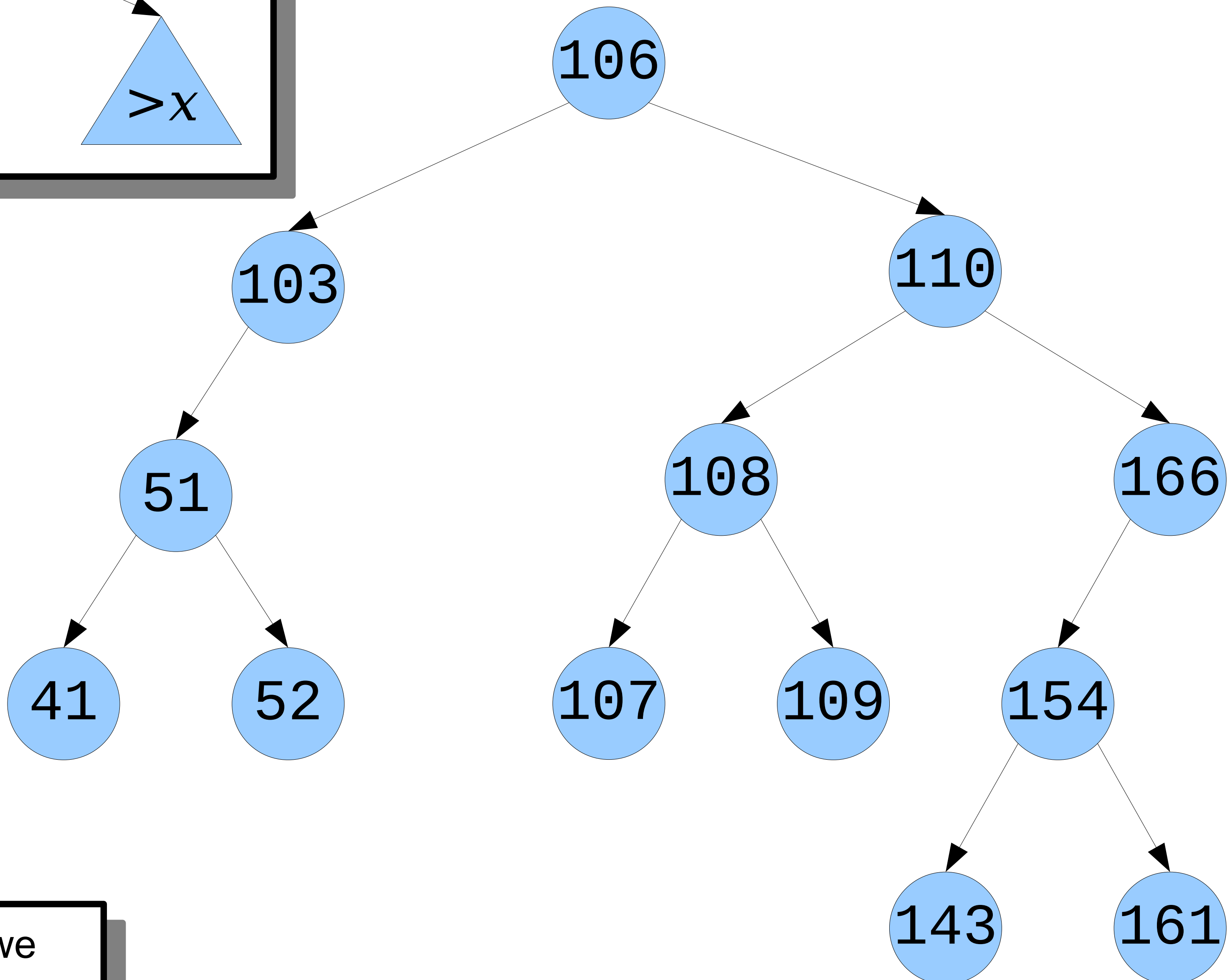
How can we check if **108** is in this tree?



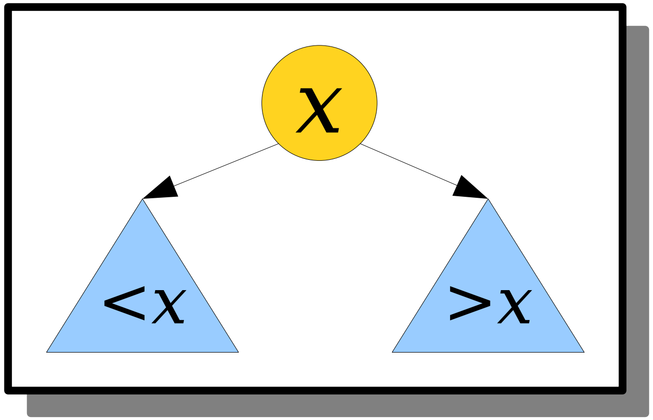


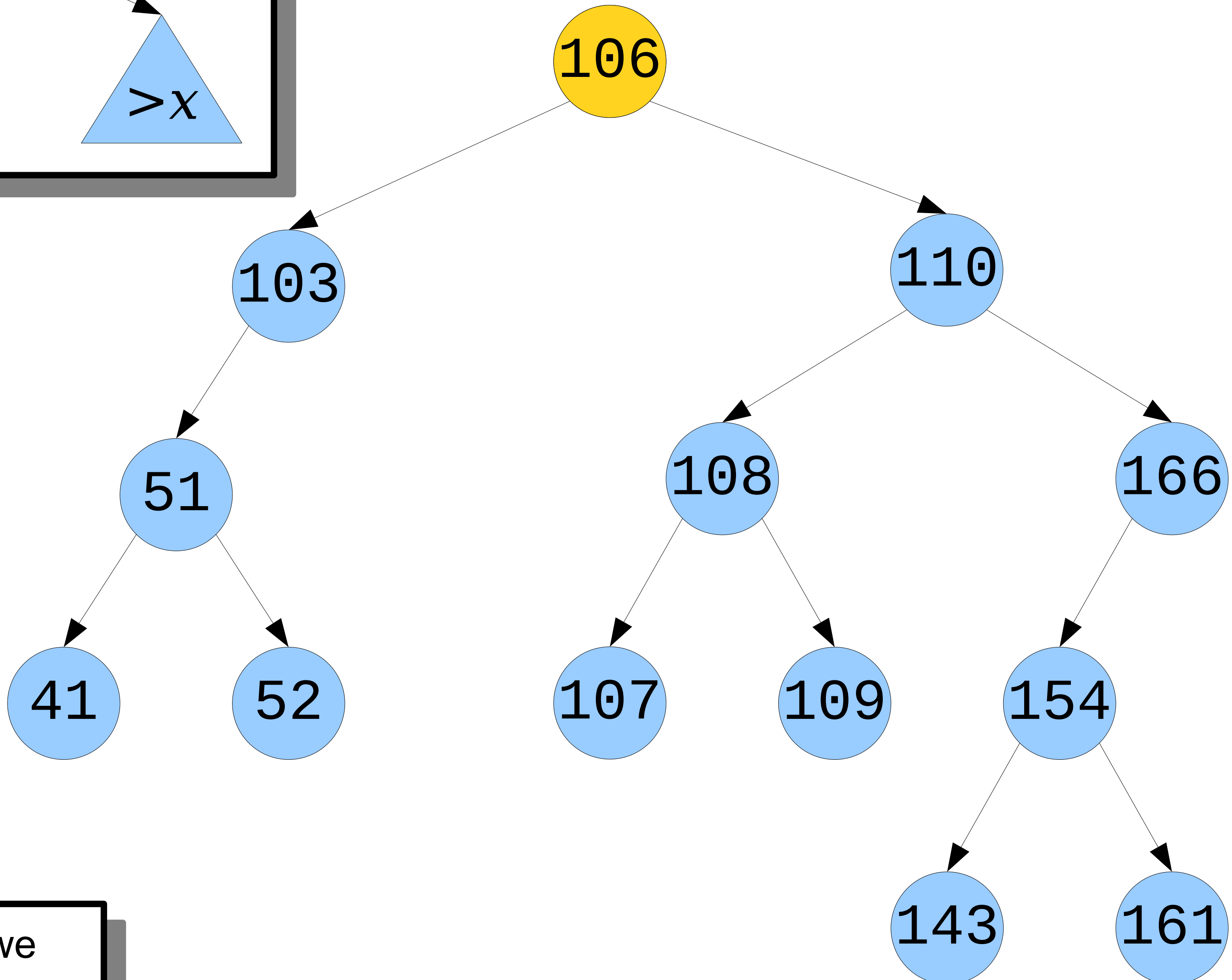
How can we check if **108** is in this tree?



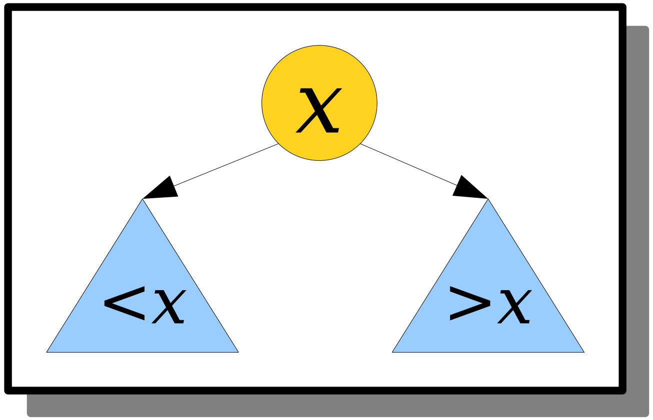


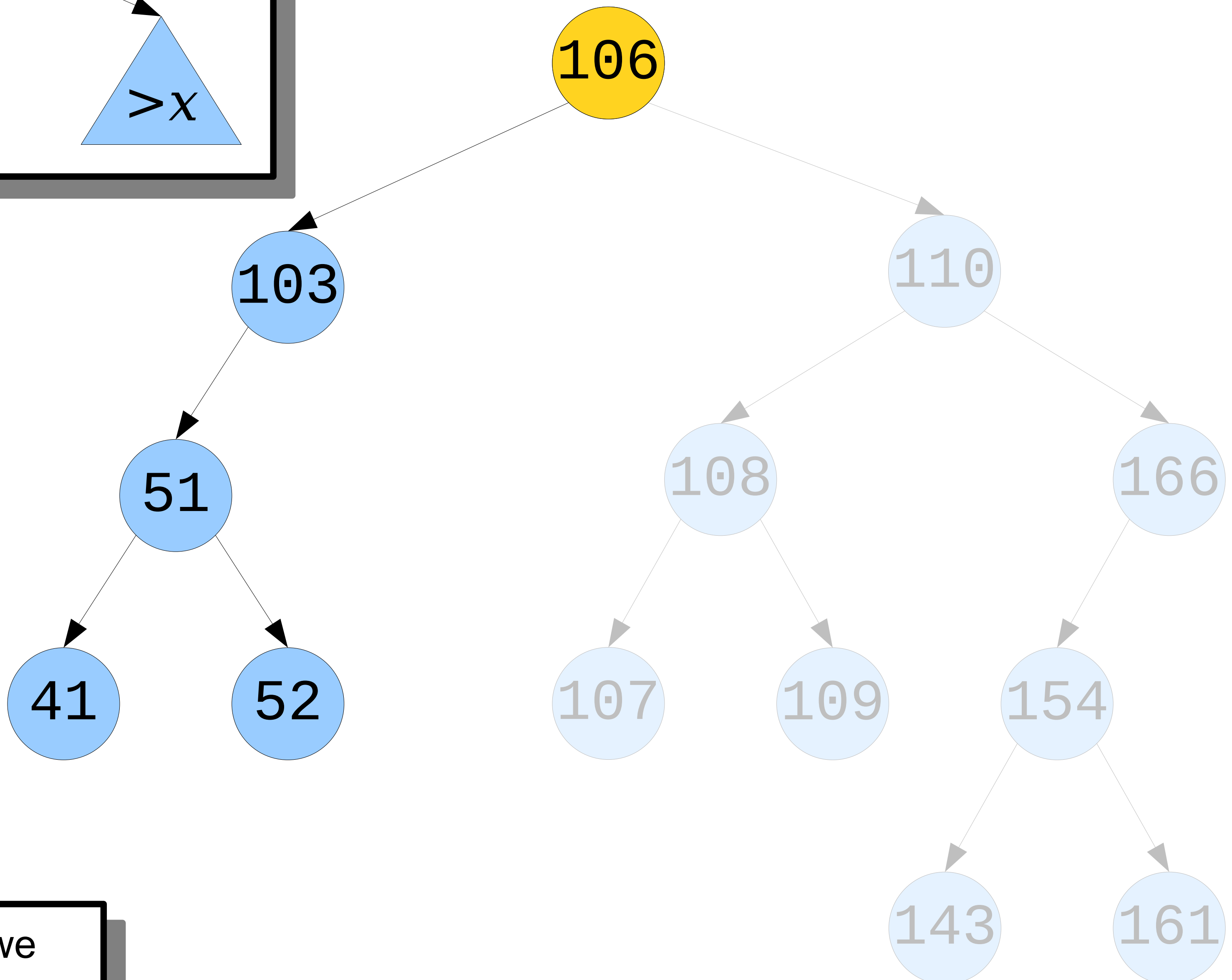
How can we check if **83** is in this tree?



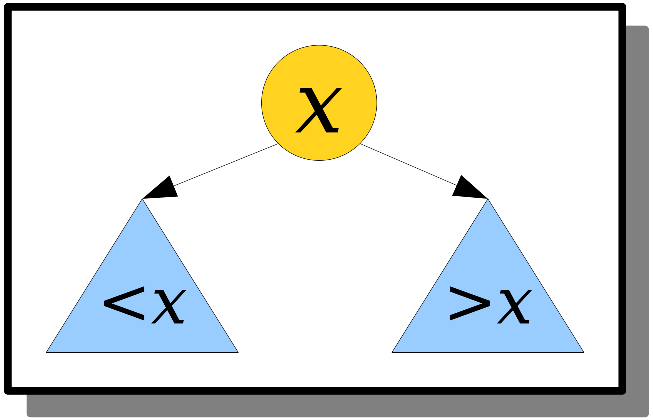


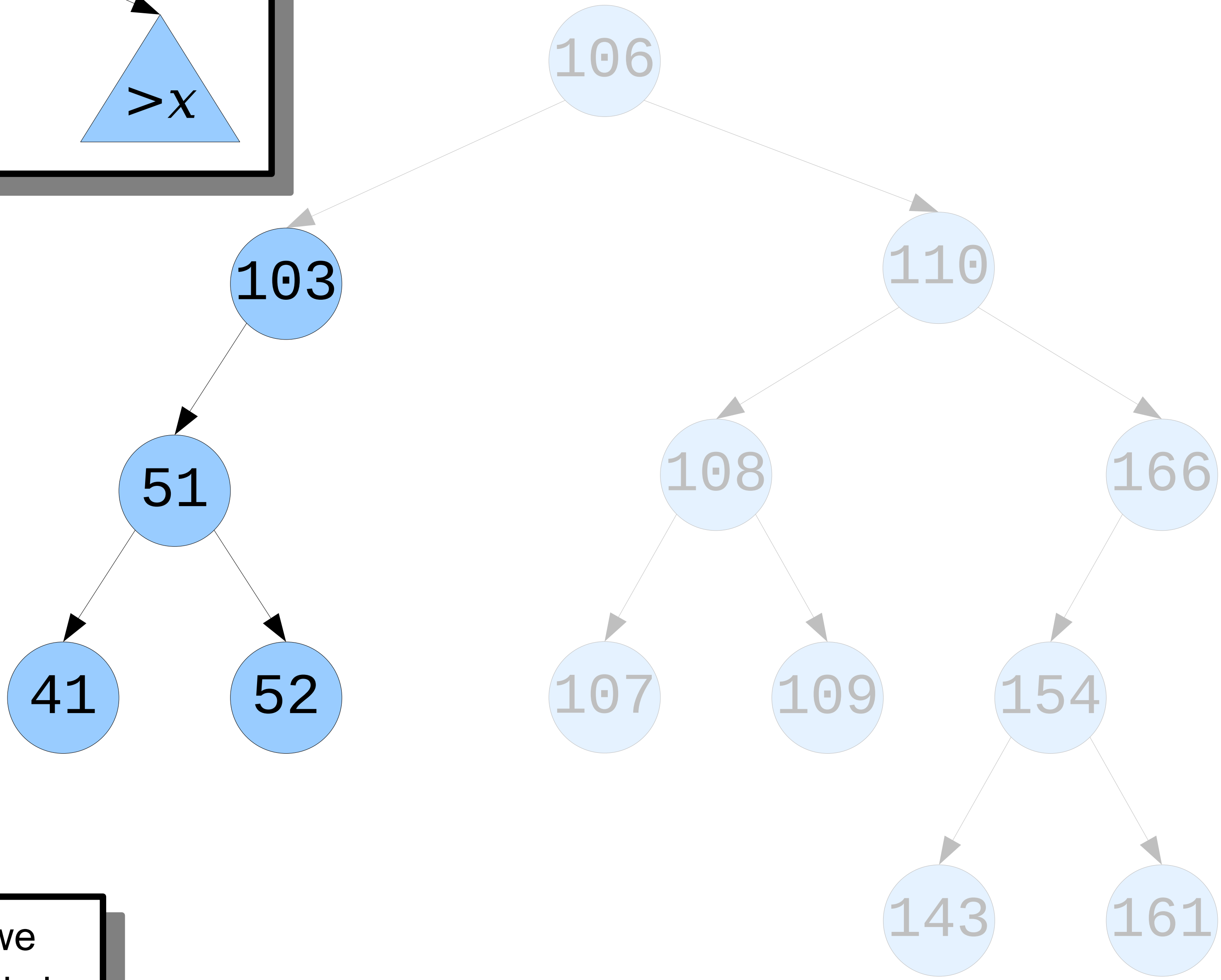
How can we check if **83** is in this tree?



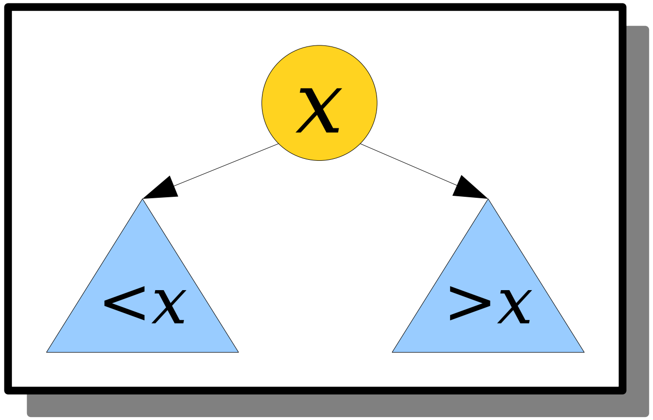


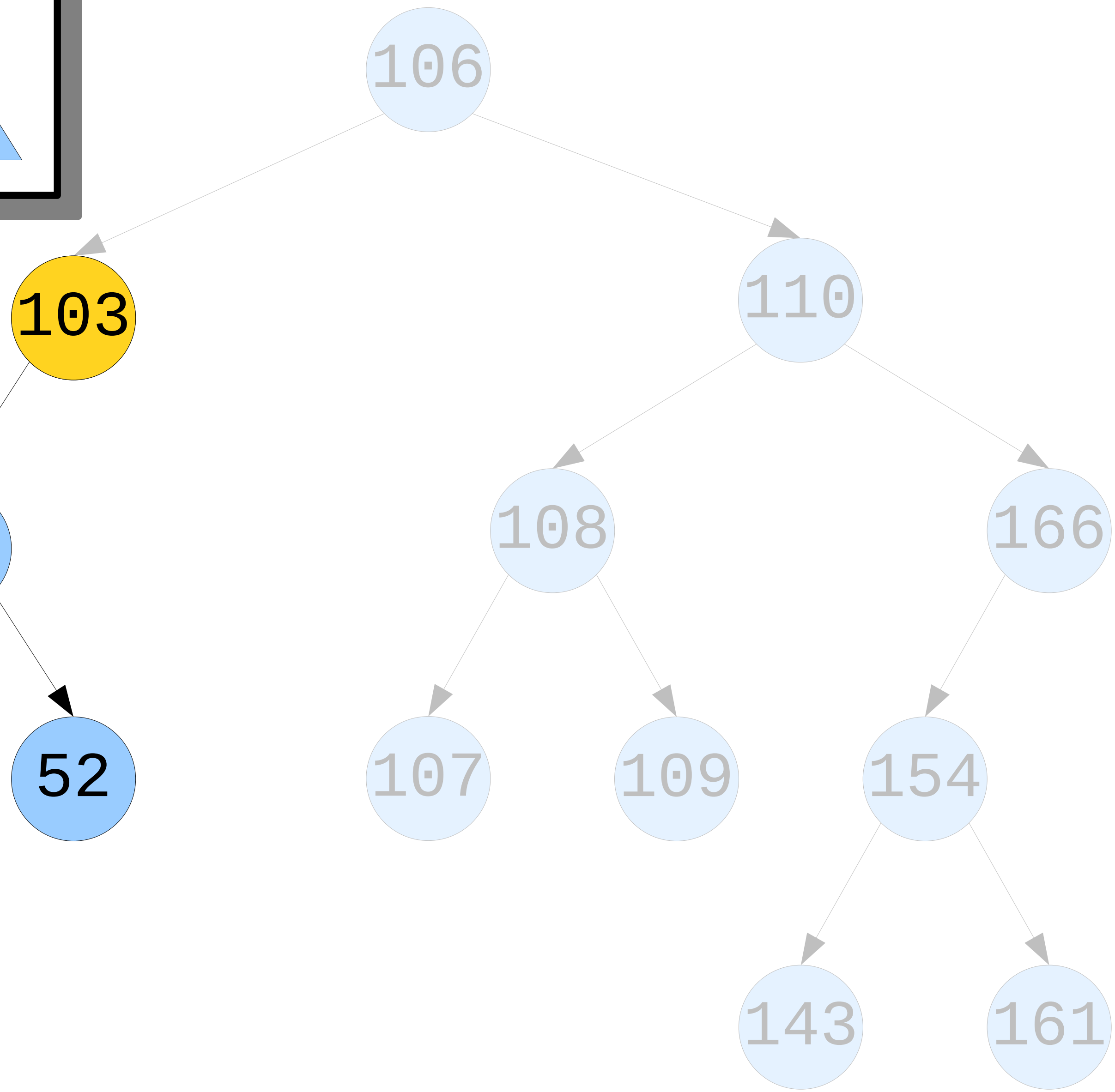
How can we check if **83** is in this tree?



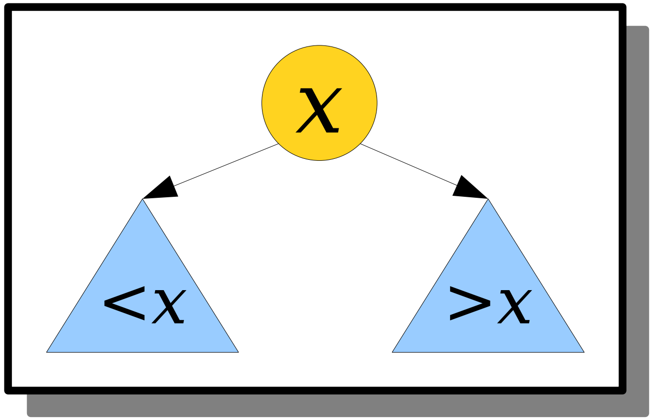


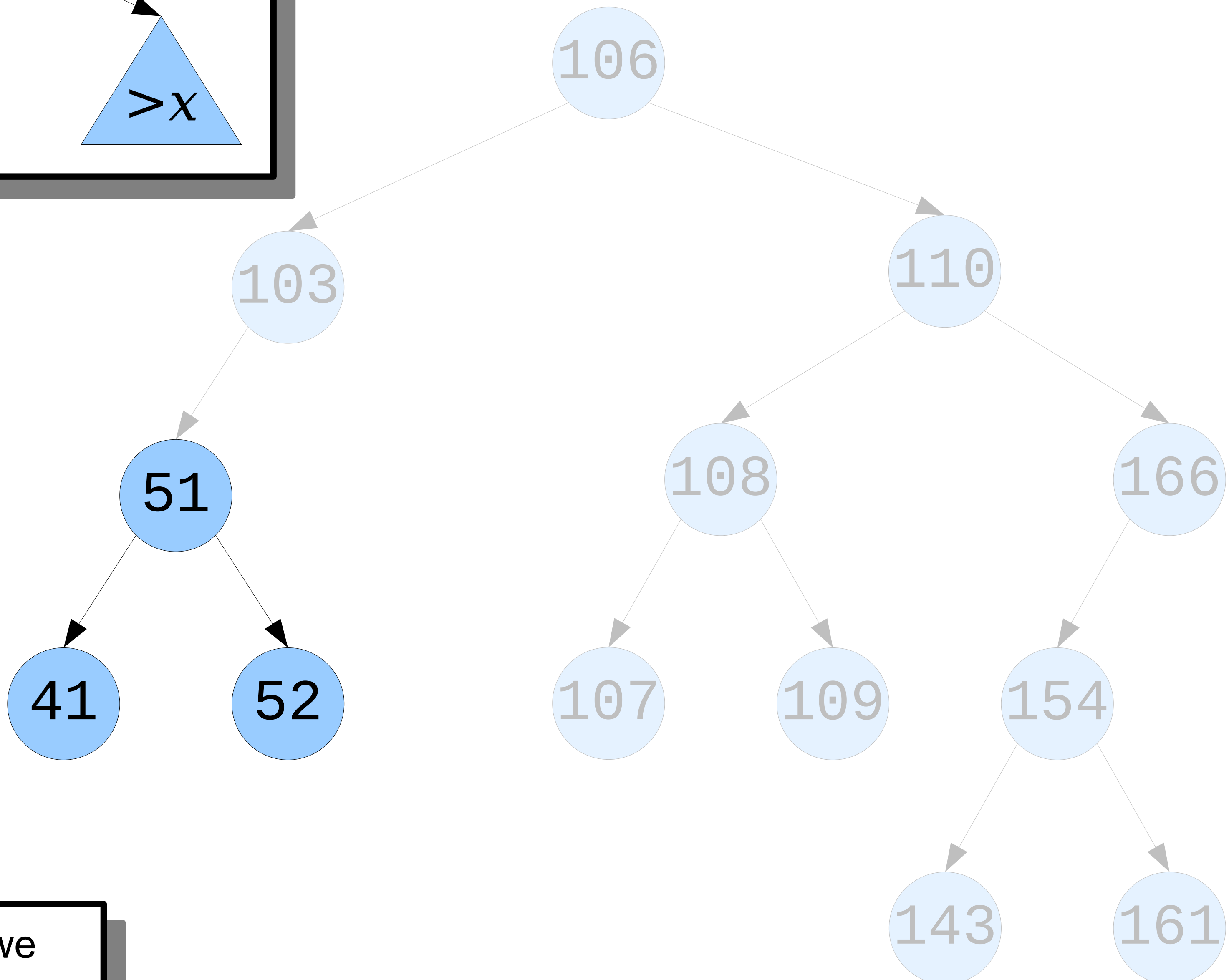
How can we check if **83** is in this tree?



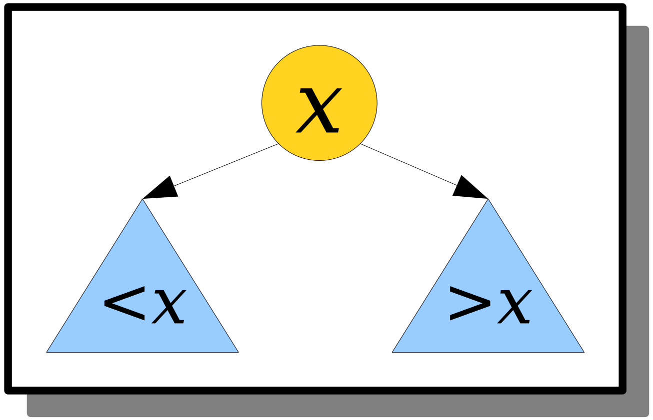


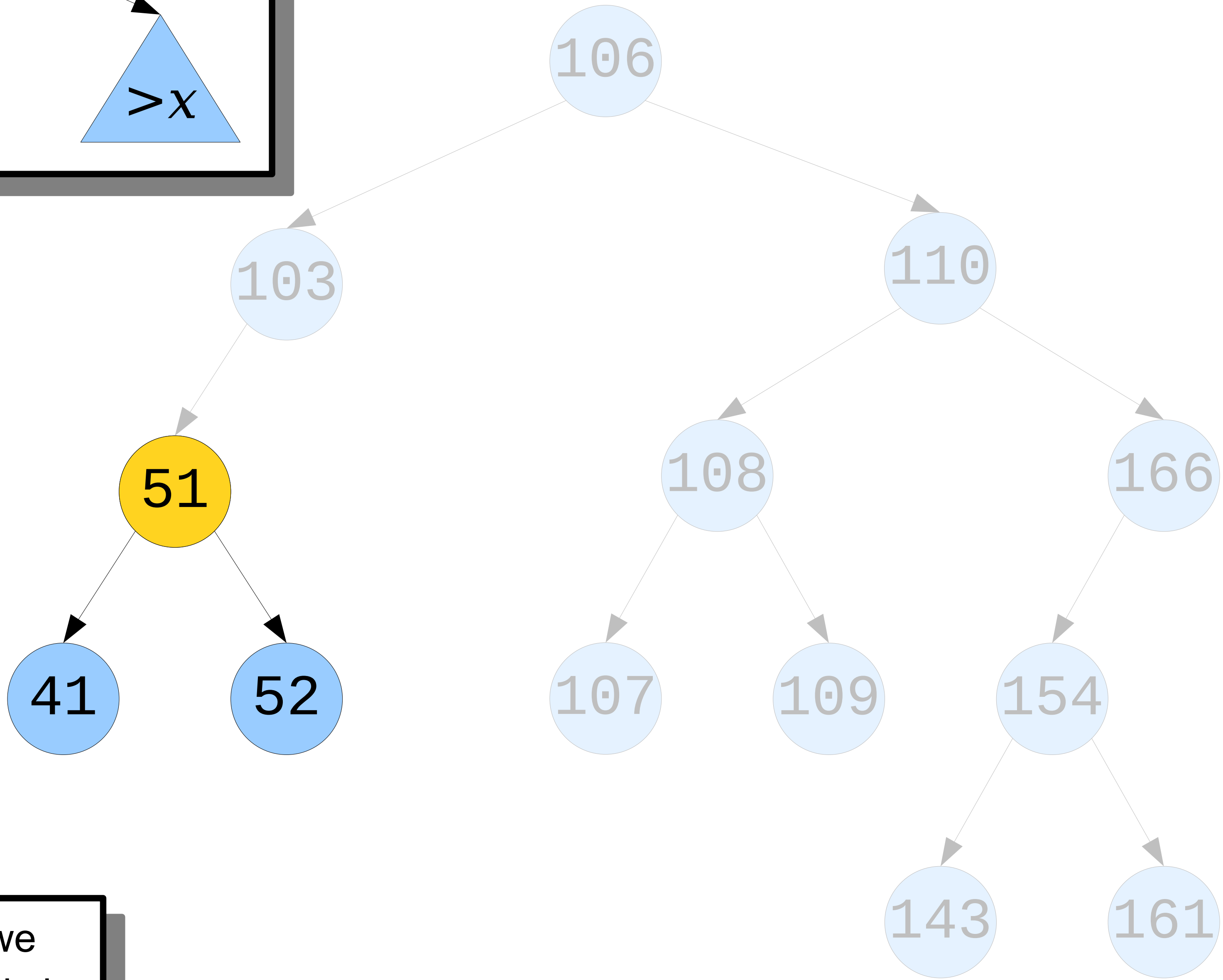
How can we check if **83** is in this tree?



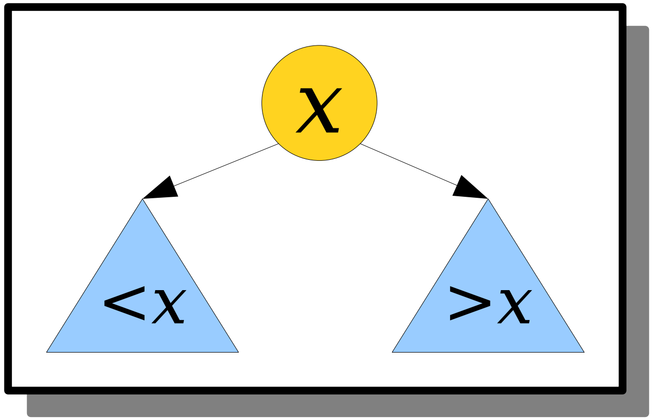


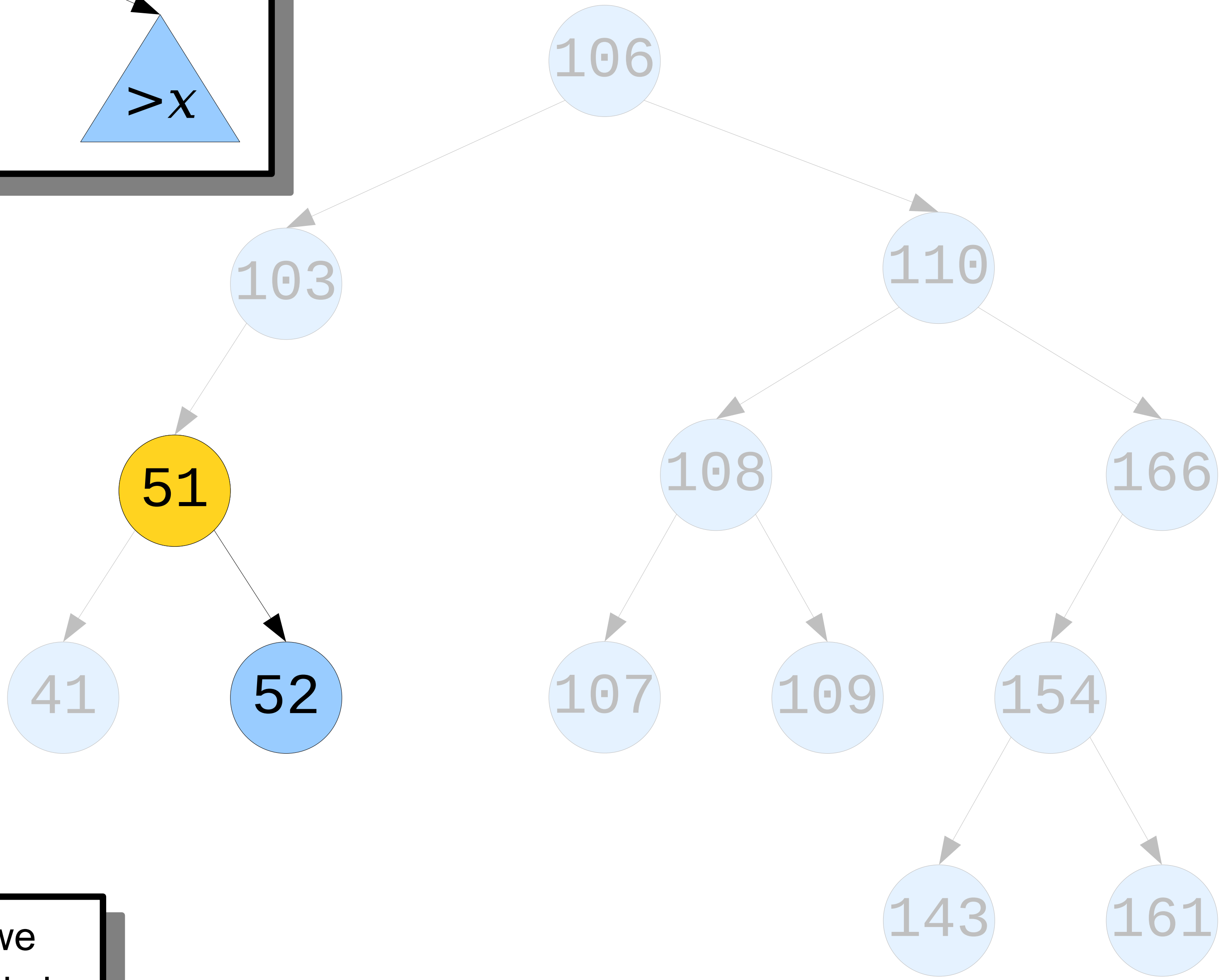
How can we check if **83** is in this tree?



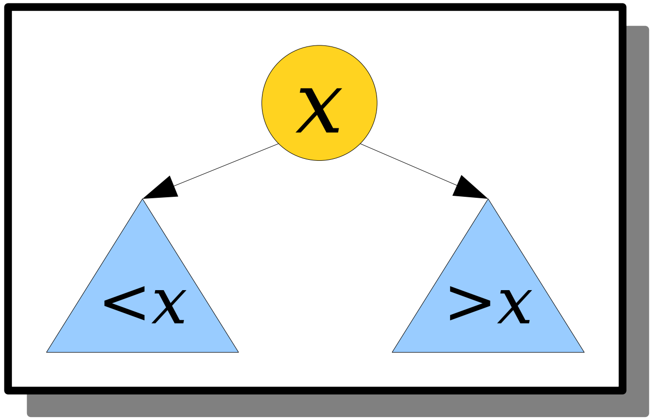


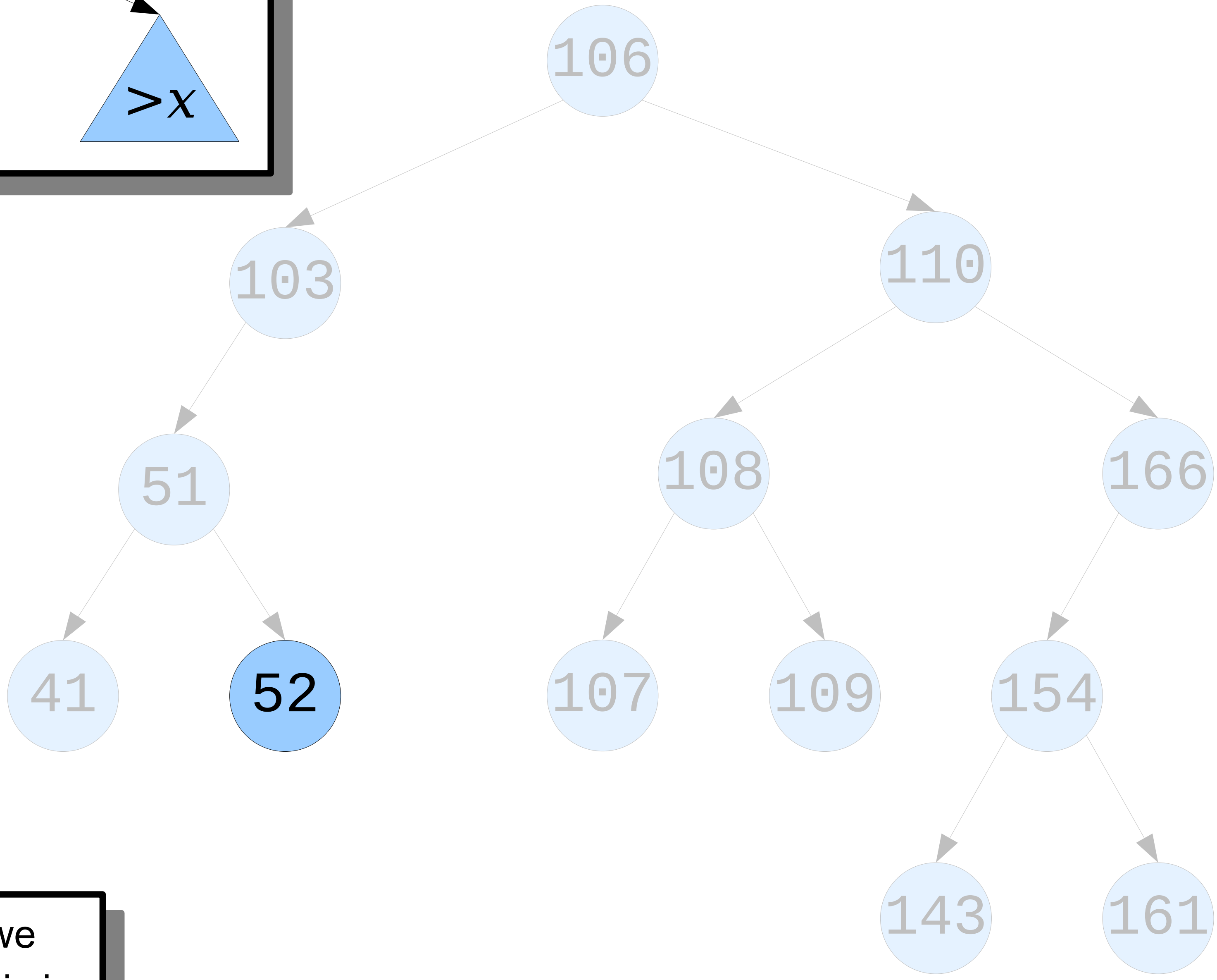
How can we check if **83** is in this tree?



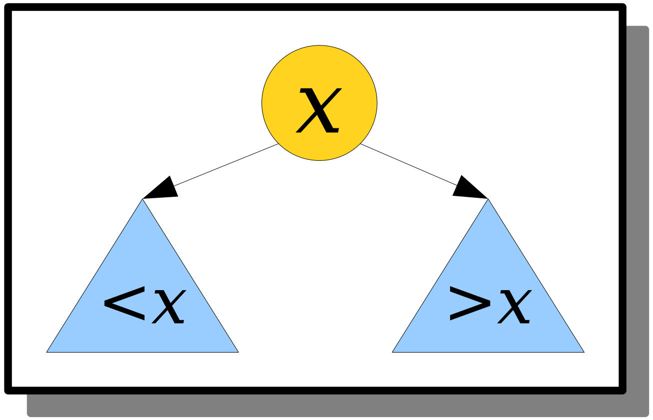


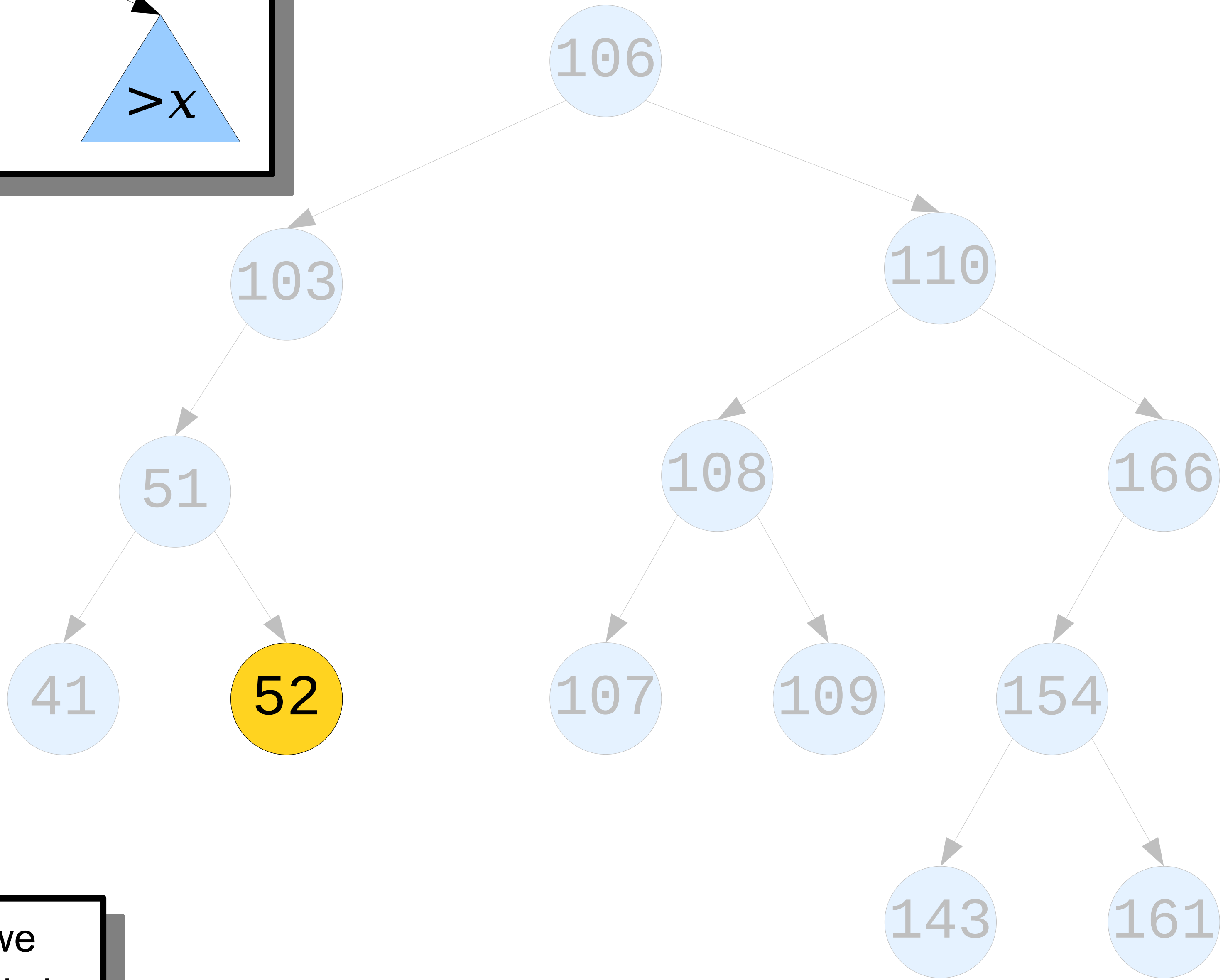
How can we check if **83** is in this tree?



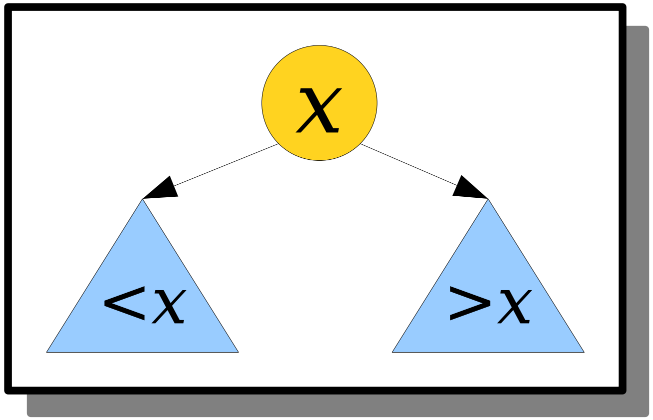


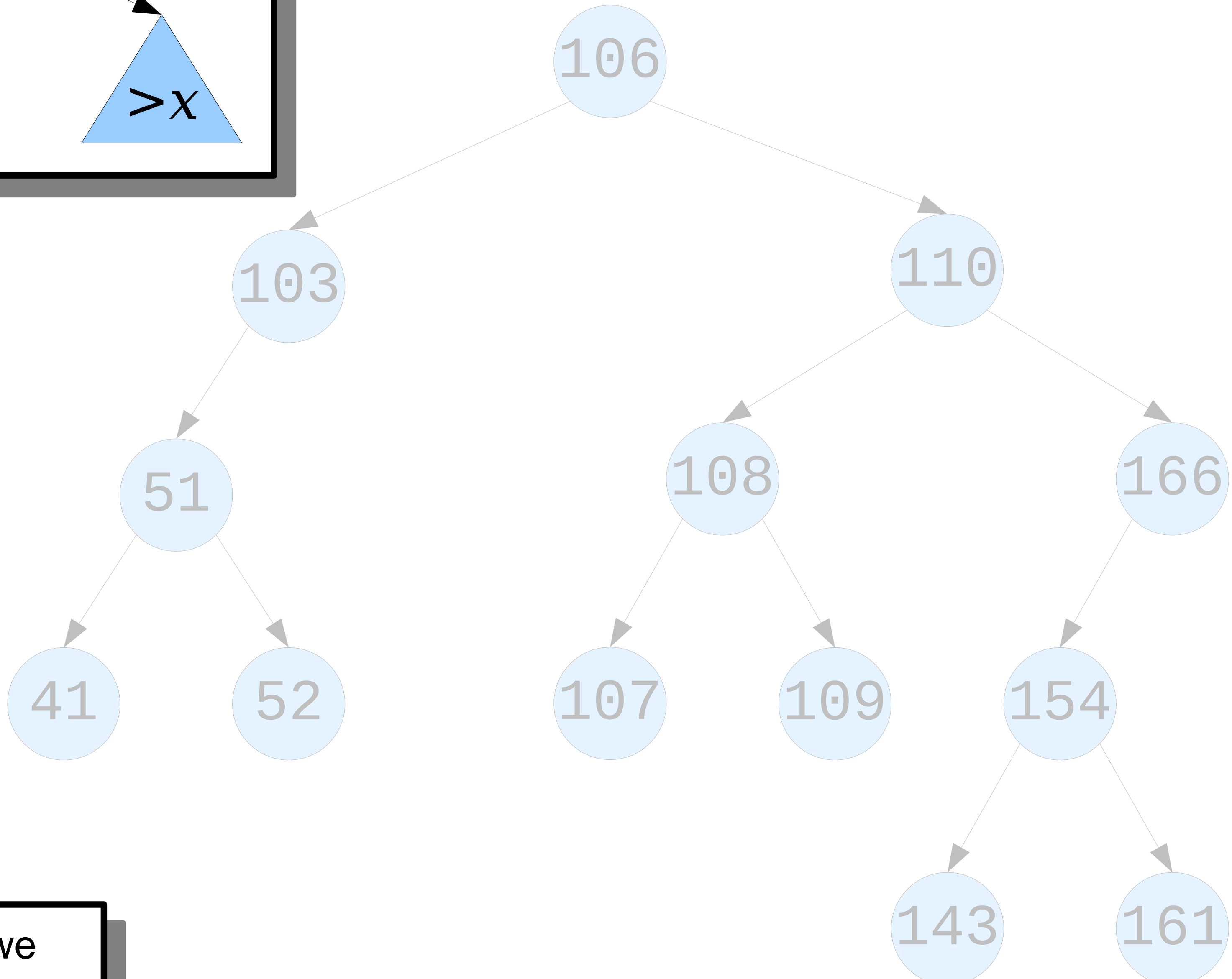
How can we check if **83** is in this tree?





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

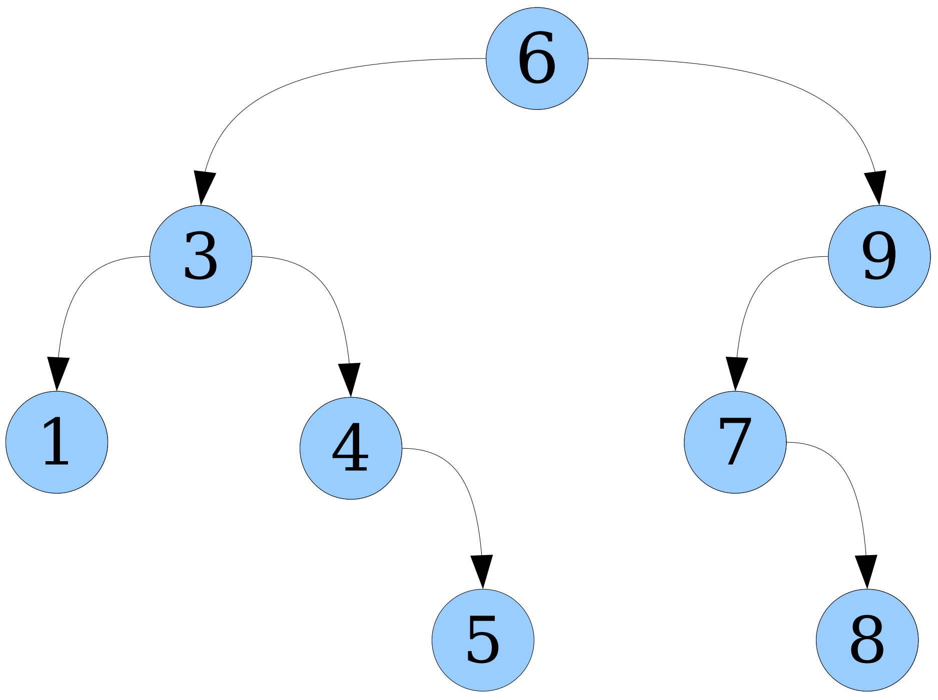


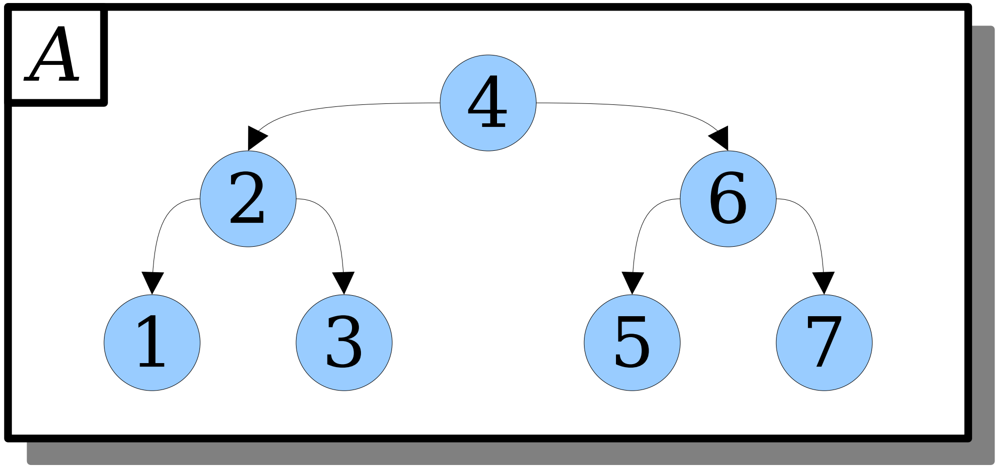


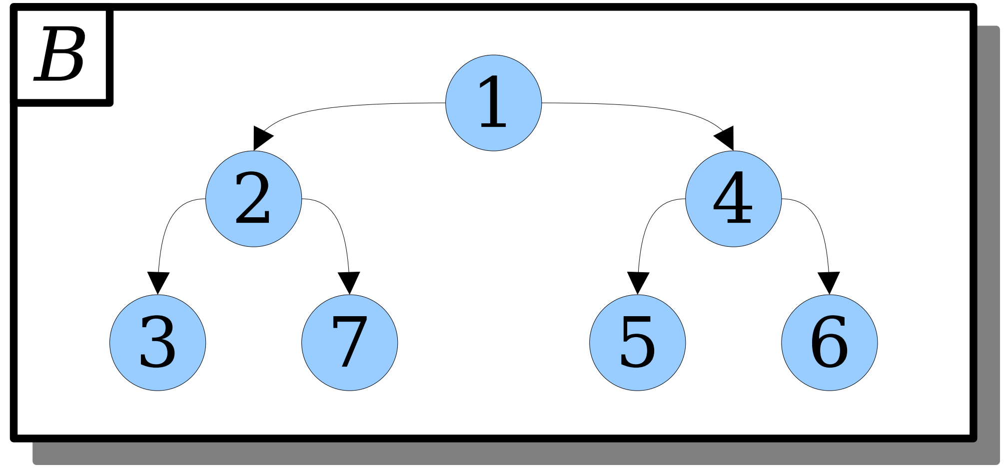
How can we check if **83** is in this tree?

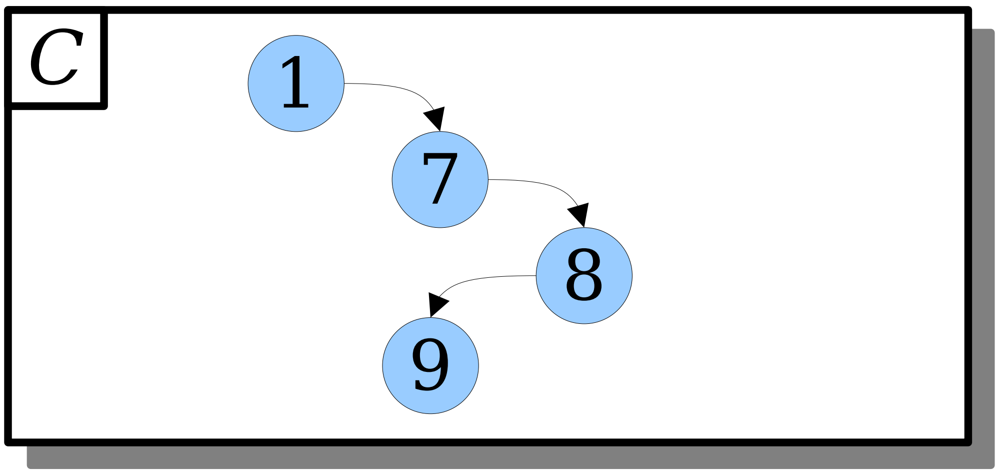
Binary Search Trees

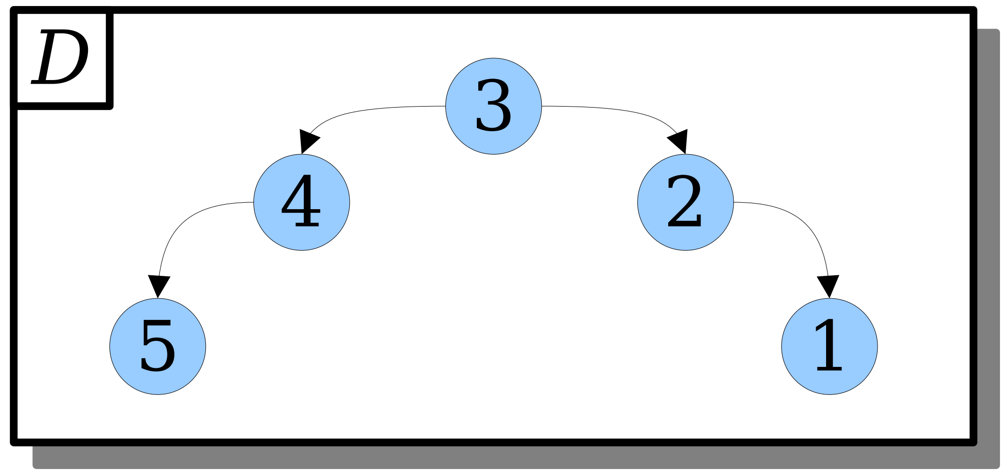
- The data structure we have just seen is called a ***binary search tree*** (or ***BST***).
- The tree consists of a number of ***nodes***, each of which stores a value and has zero, one, or two ***children***.
- All values in a node's left subtree are ***smaller*** than the node's value, and all values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

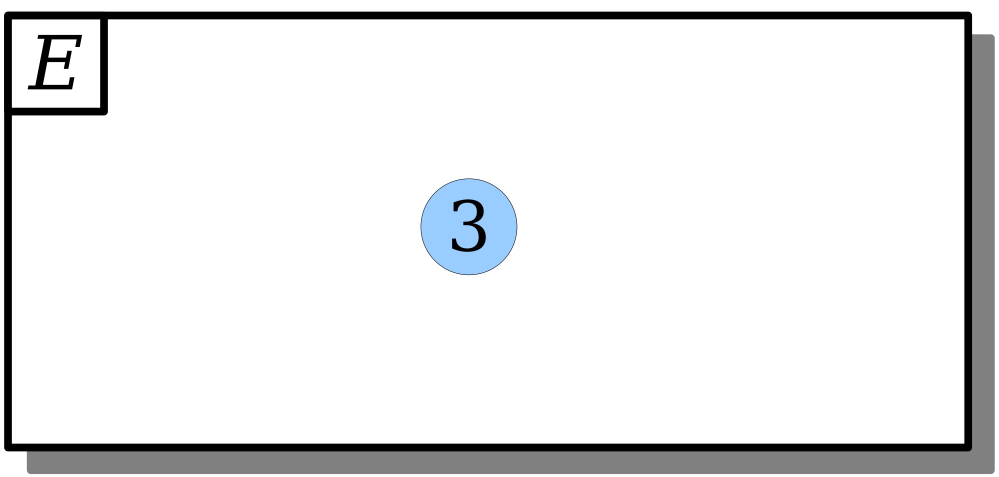










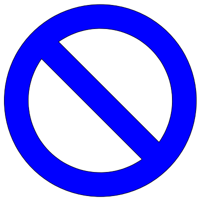


Which of these are BSTs?

Answer online at
https://cs106b.stanford.edu/pollev

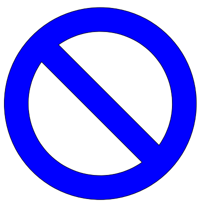
A Binary Search Tree Is Either…

an empty tree,
represented by
`nullptr`

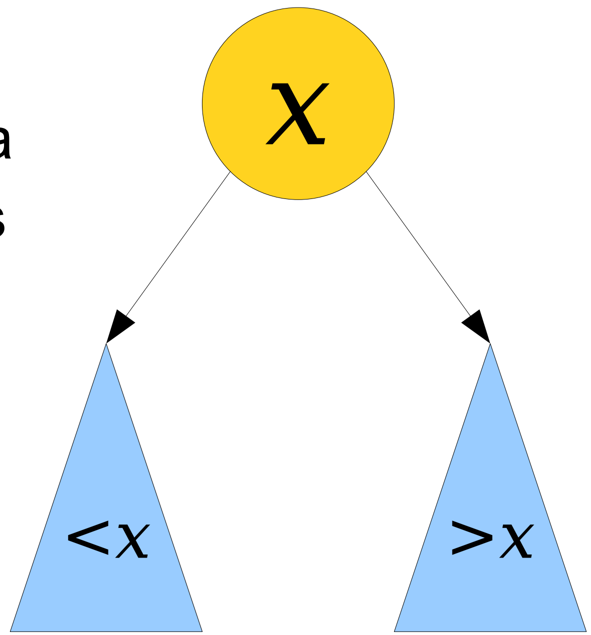


A Binary Search Tree Is Either…

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

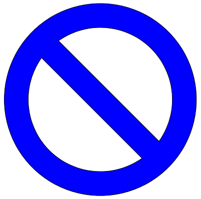
```
struct Node {
    Type value;
    Node* left;  // Smaller values
    Node* right; // Bigger values
};
```

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

Searching Trees

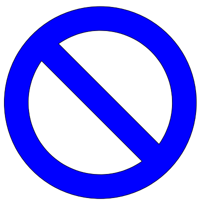
A Binary Search Tree Is Either…

an empty tree,
represented by
`nullptr`



A Binary Search Tree Is Either…

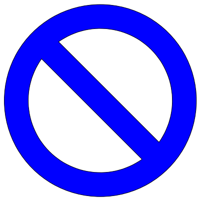
an empty tree, represented by **`nullptr`**



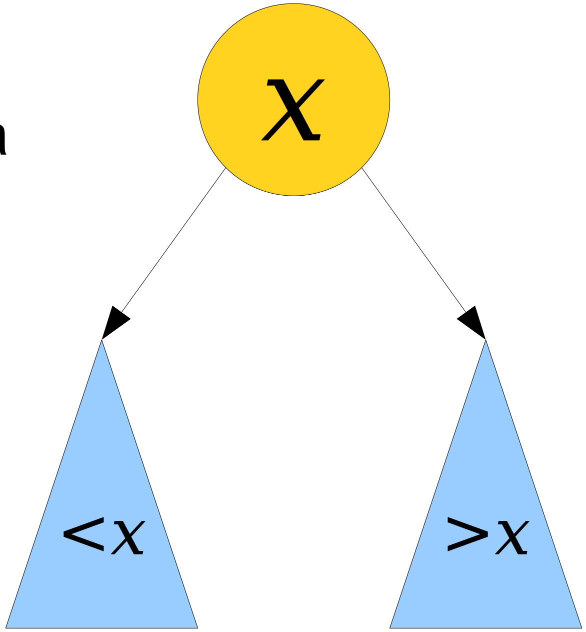
If you're looking for something in an empty BST, it's not there! Sorry.

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
…

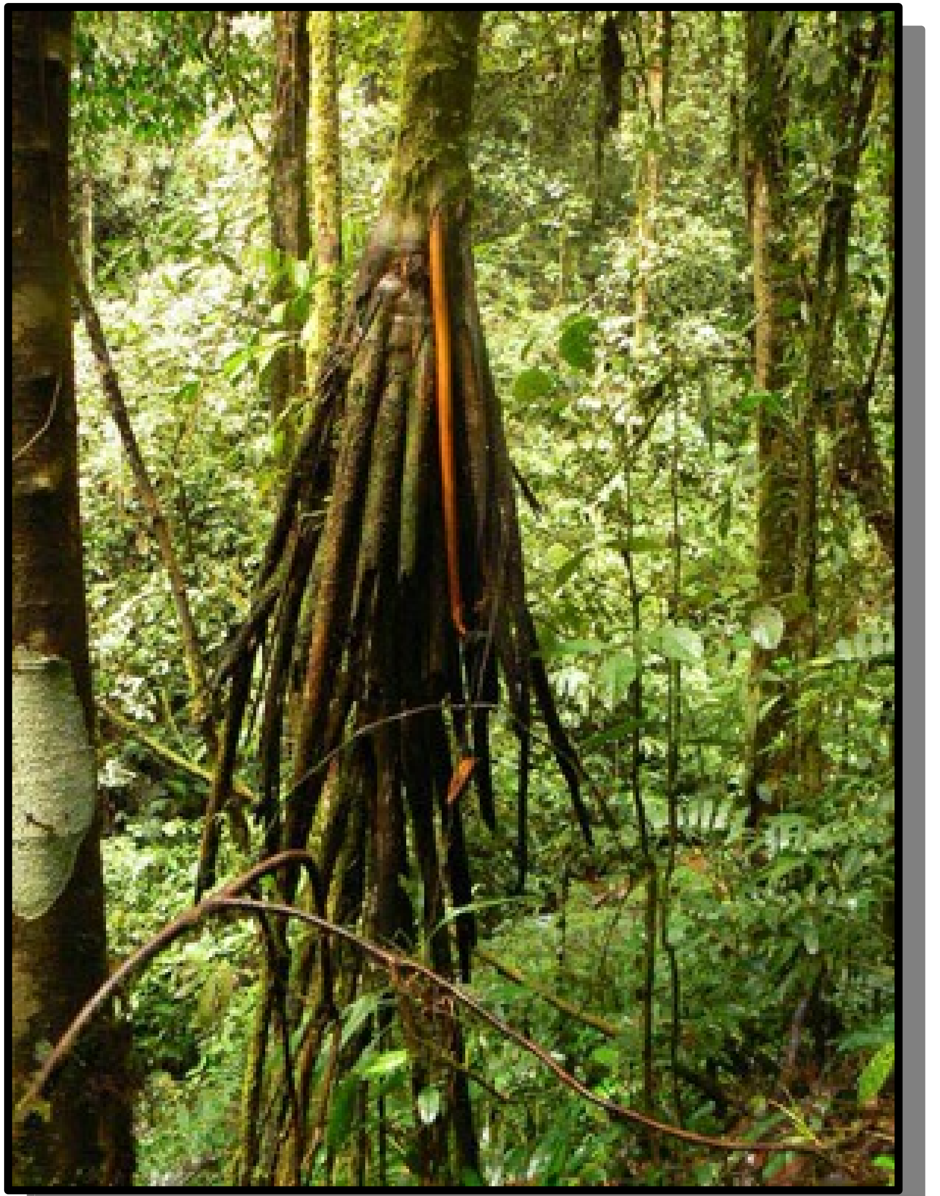


… and whose right
subtree is a BST of
larger values.

Good exercise:

Rewrite this function iteratively!

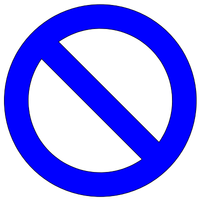
Walking Trees



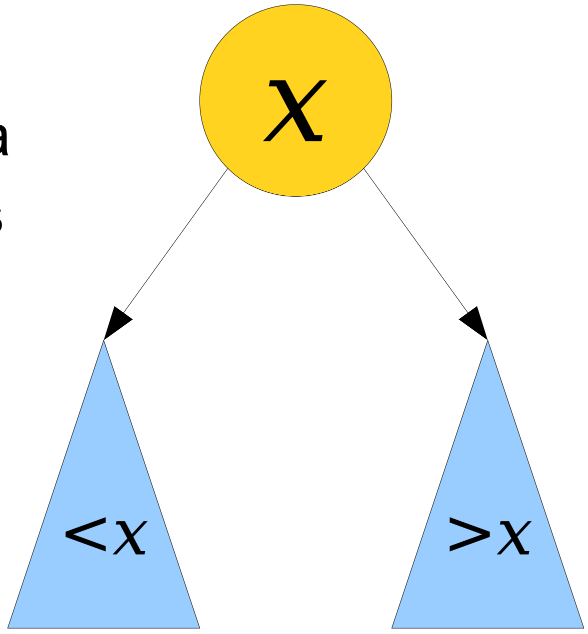
Print all the values in a BST,
in sorted order.

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
…

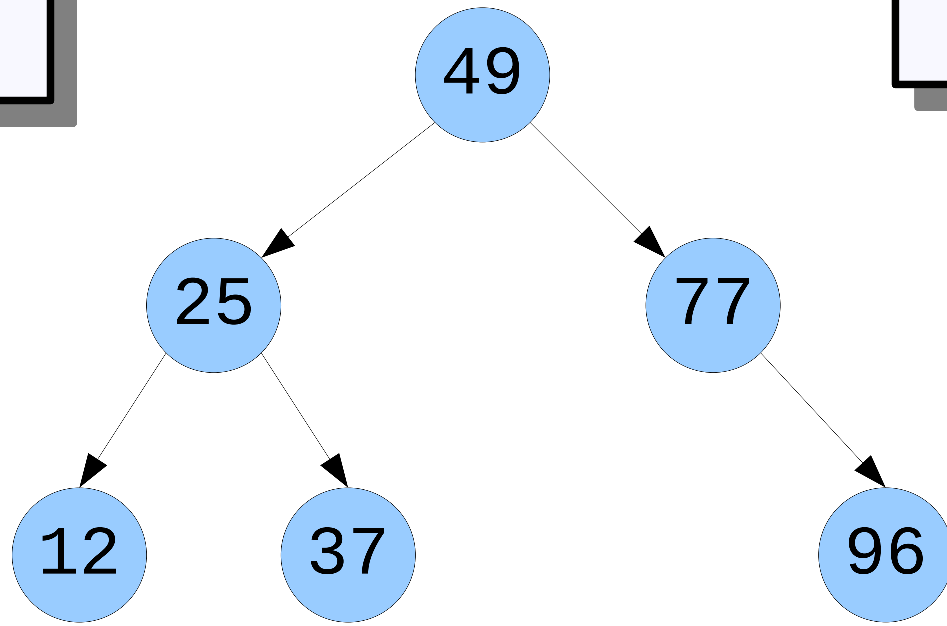


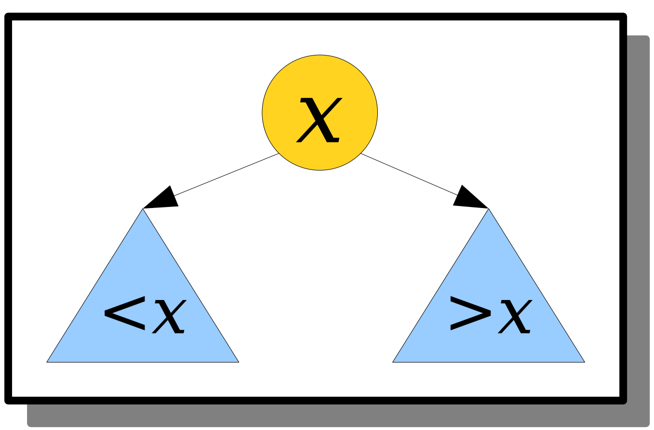
… and whose right
subtree is a BST of
larger values.

Print all values that come before 49...

…then 49…

… then all values that come after 49.



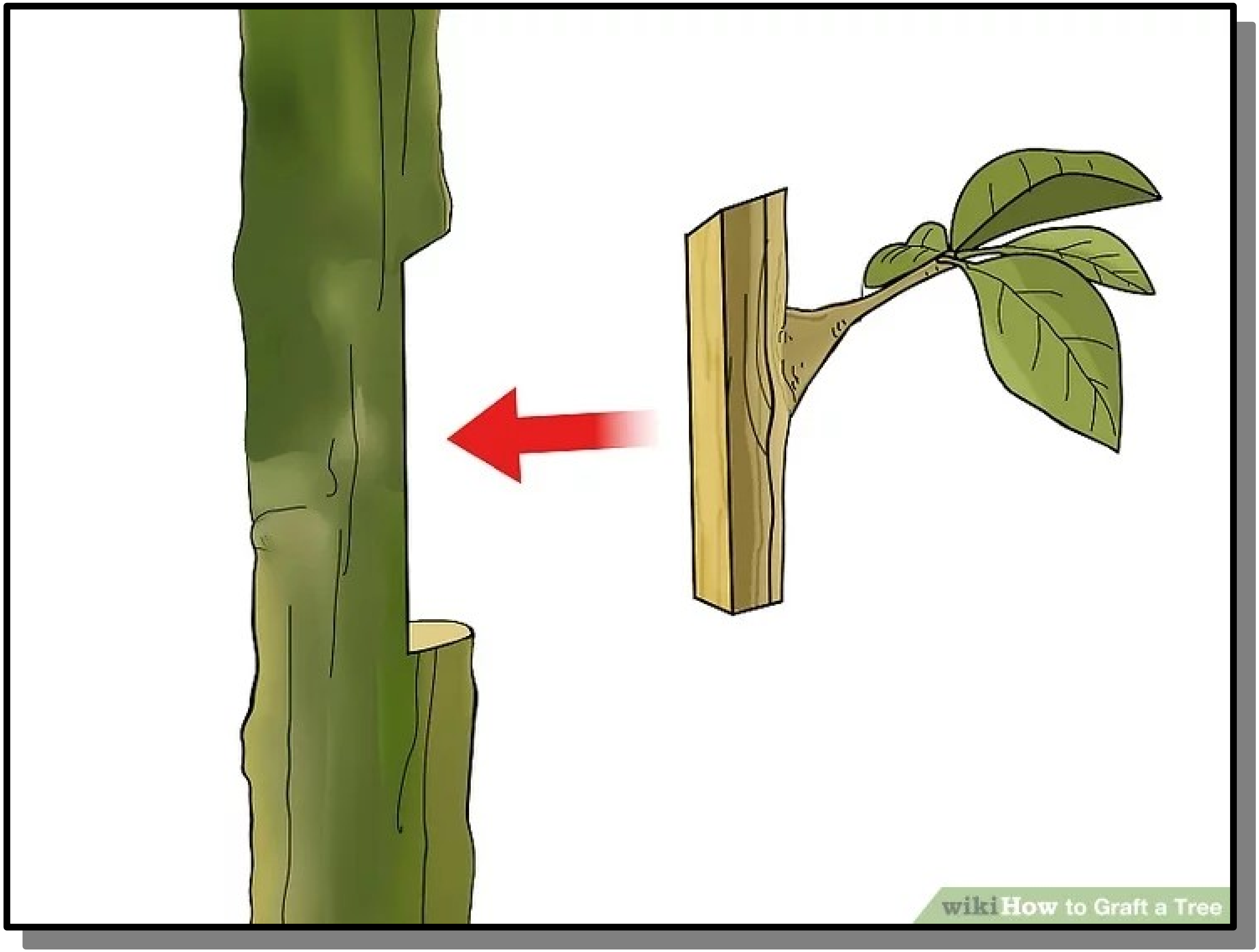


Inorder Traversals

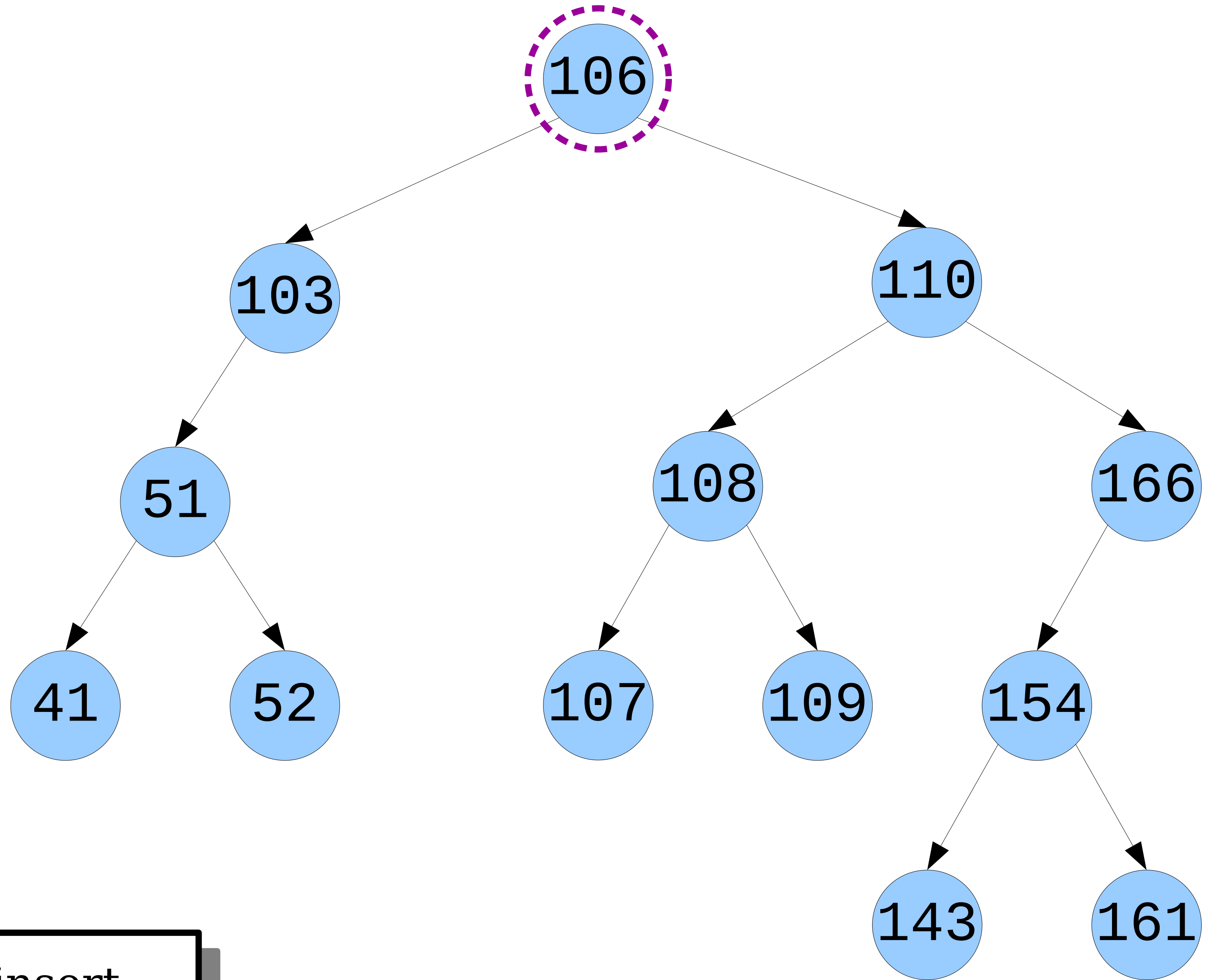
- The particular recursive pattern we just saw is called an ***inorder traversal*** of a binary tree.
- Specifically:
  - Recursively visit all the nodes in the left subtree.
  - Visit the node itself.
  - Recursively visit all the nodes in the right subtree.

Challenge problem:
Rewrite this function iteratively!

Adding to Trees



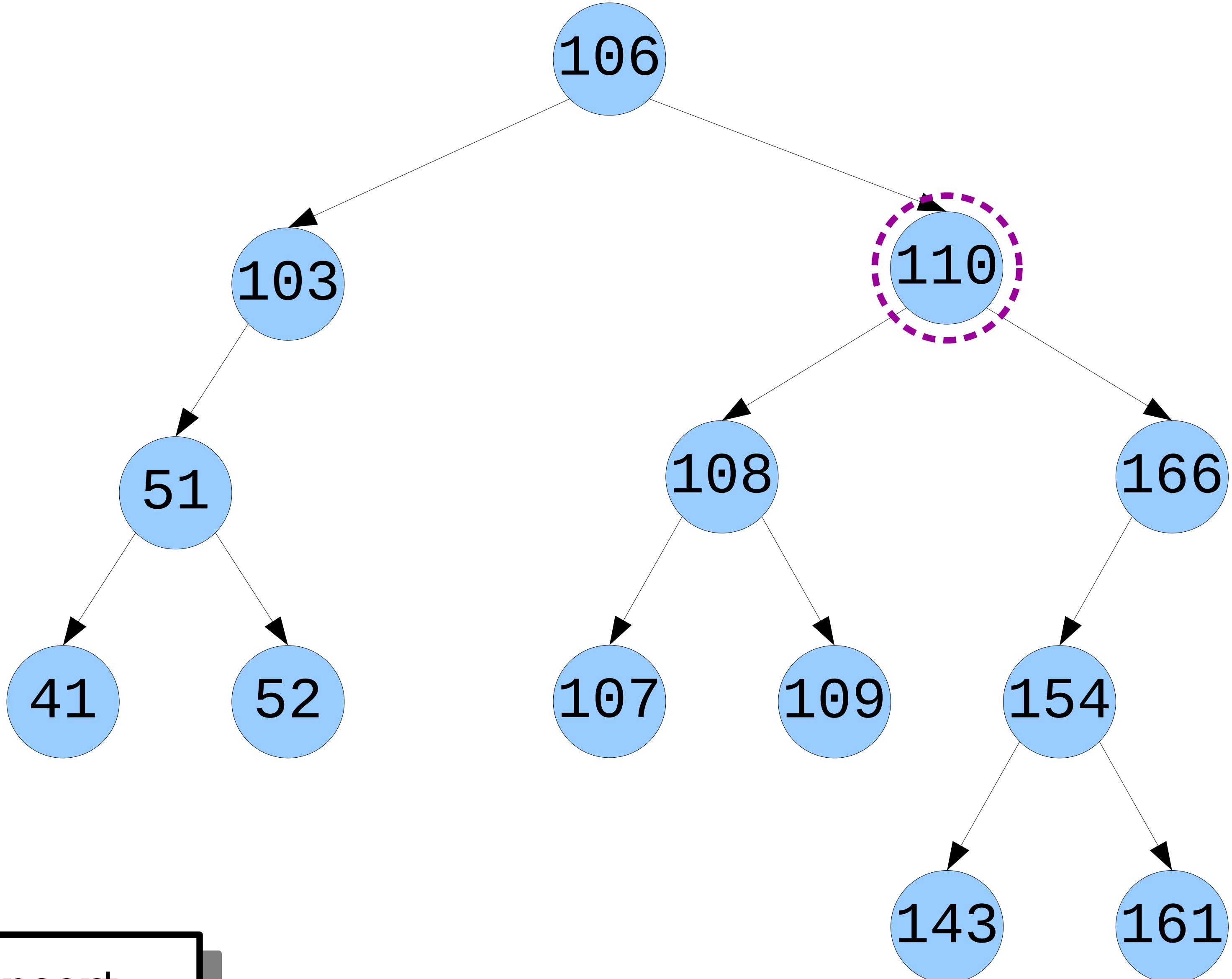
Thanks, WikiHow!



Let’s insert

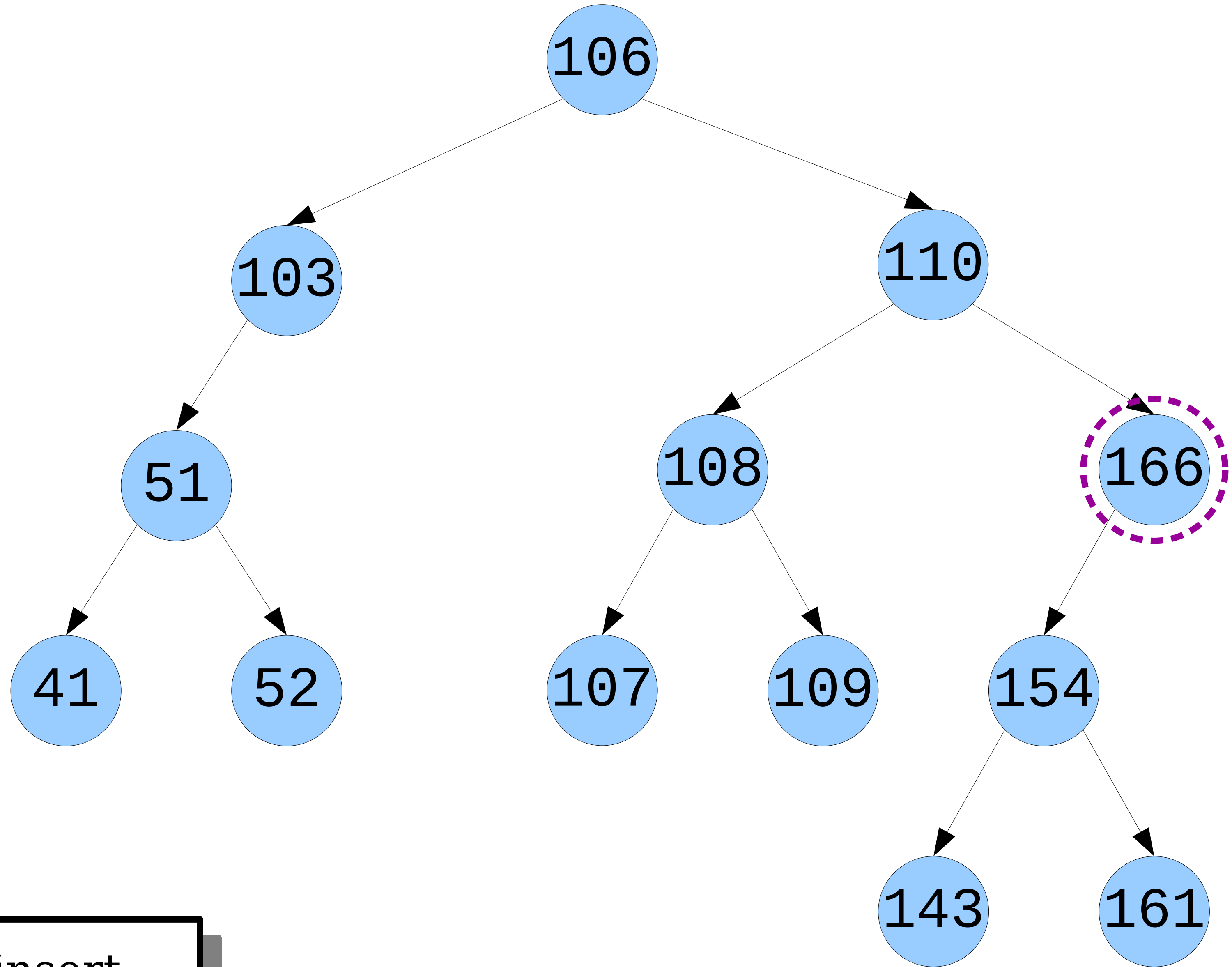
147

into this tree.



Let’s insert

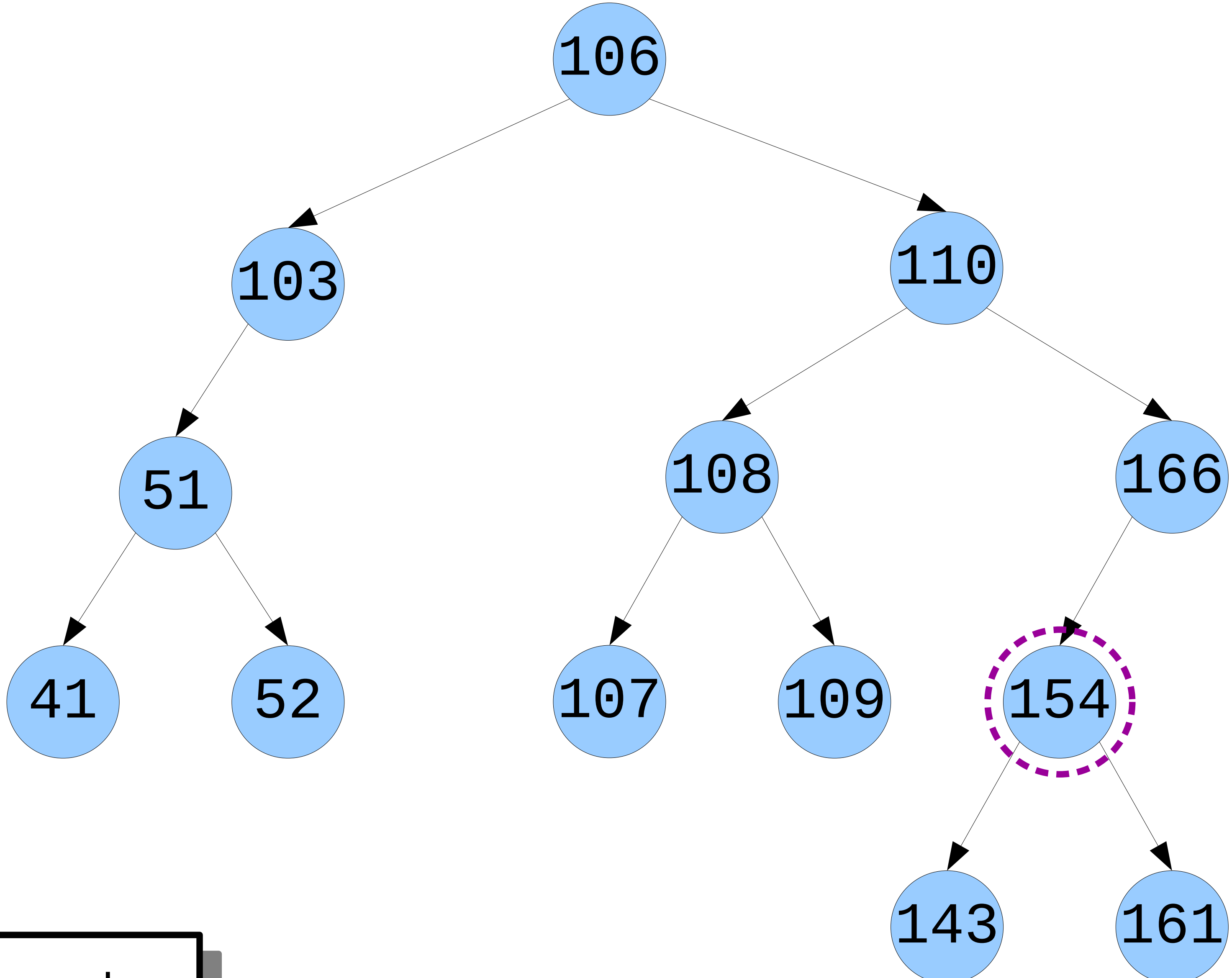
into this tree.



Let’s insert

147

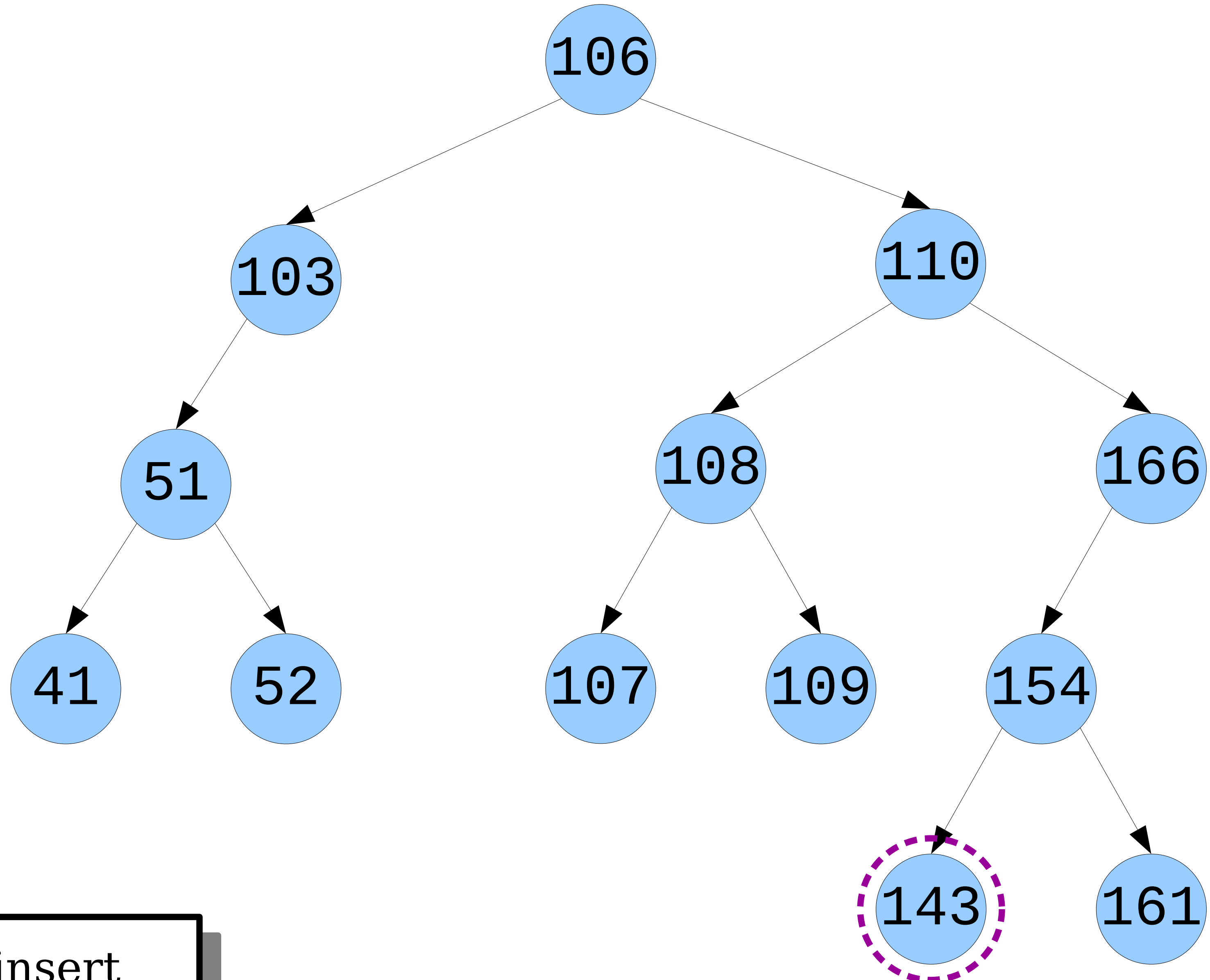
into this tree.



Let’s insert

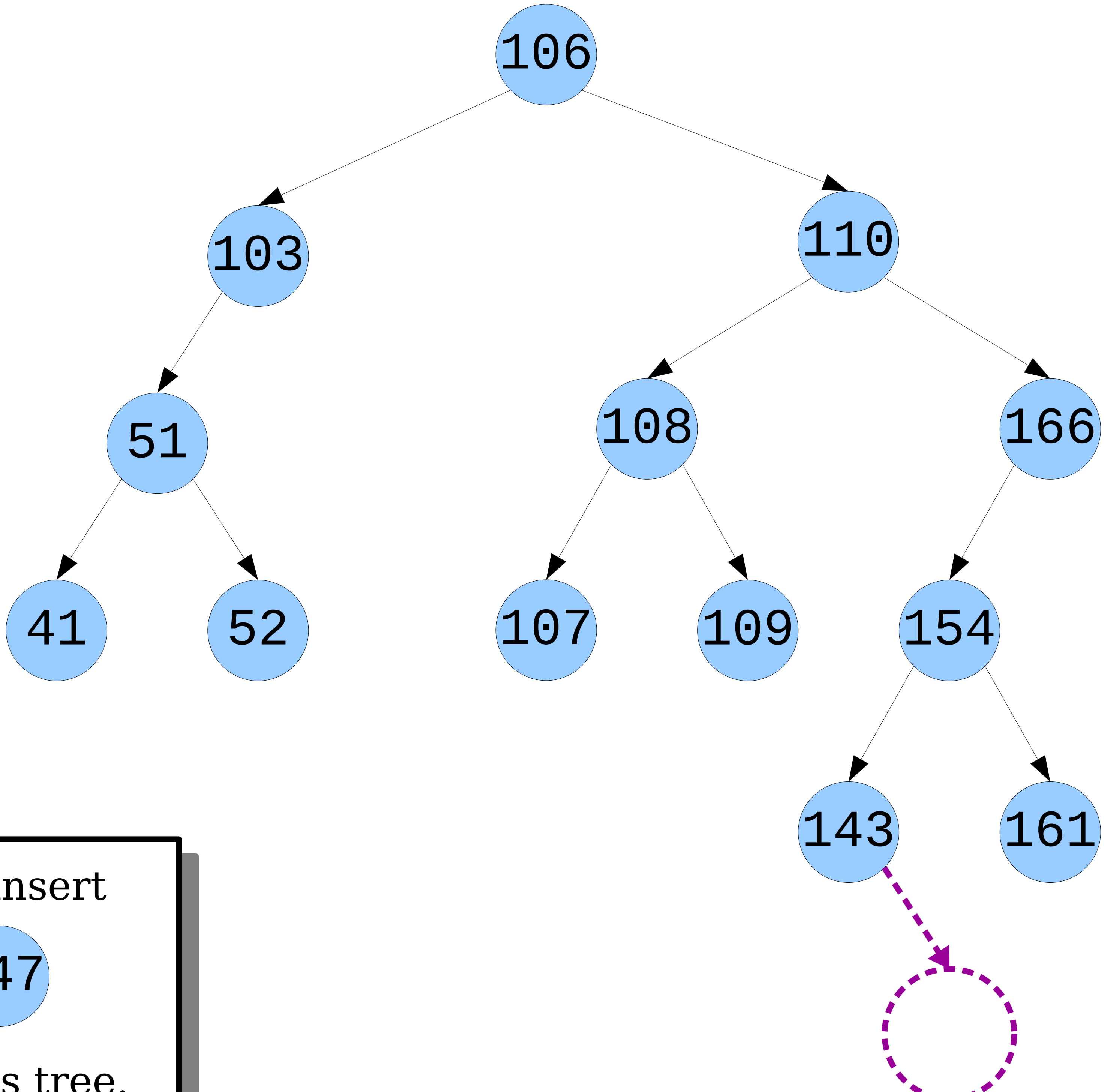
147

into this tree.



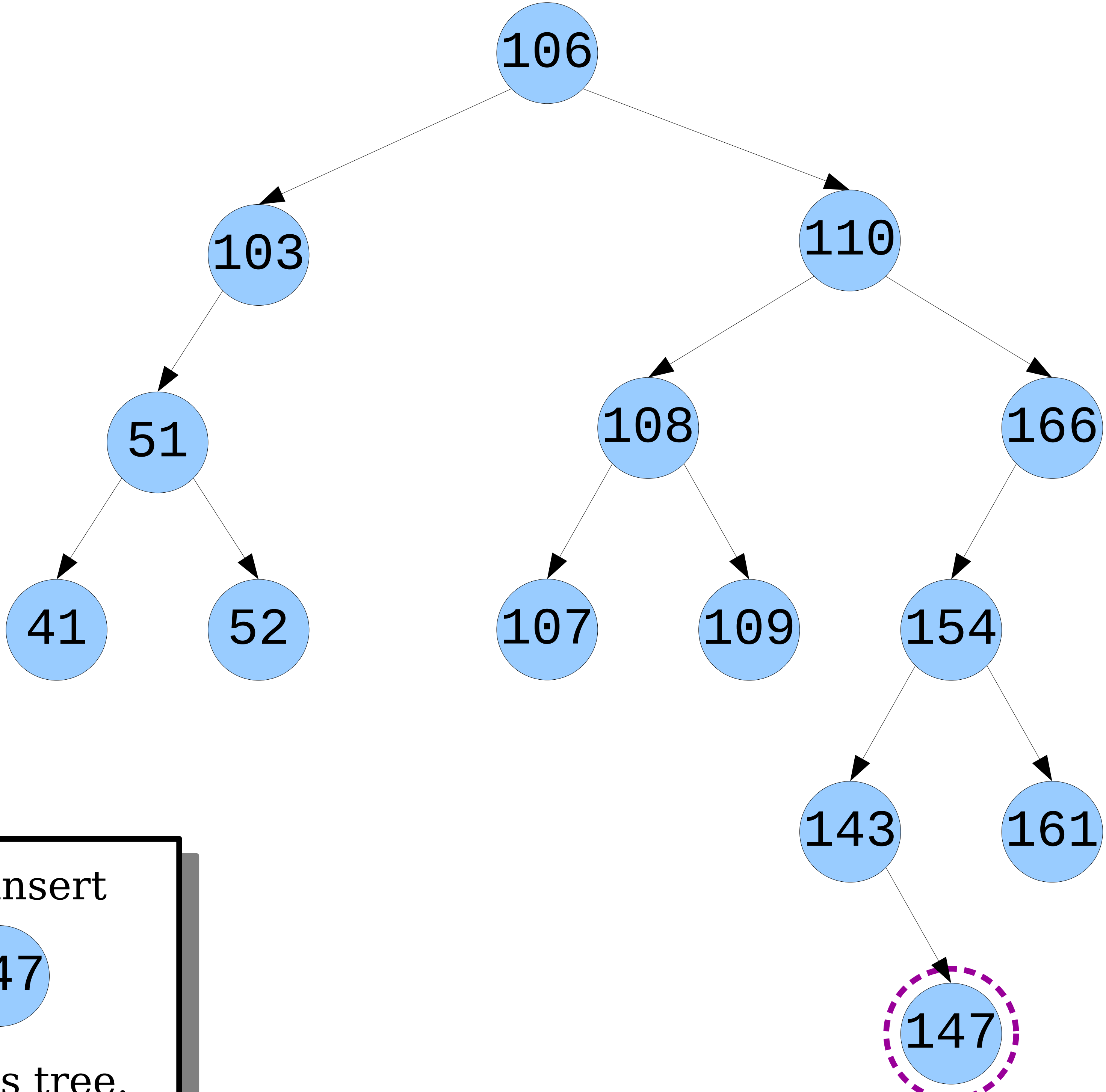
Let’s insert

into this tree.



Let’s insert

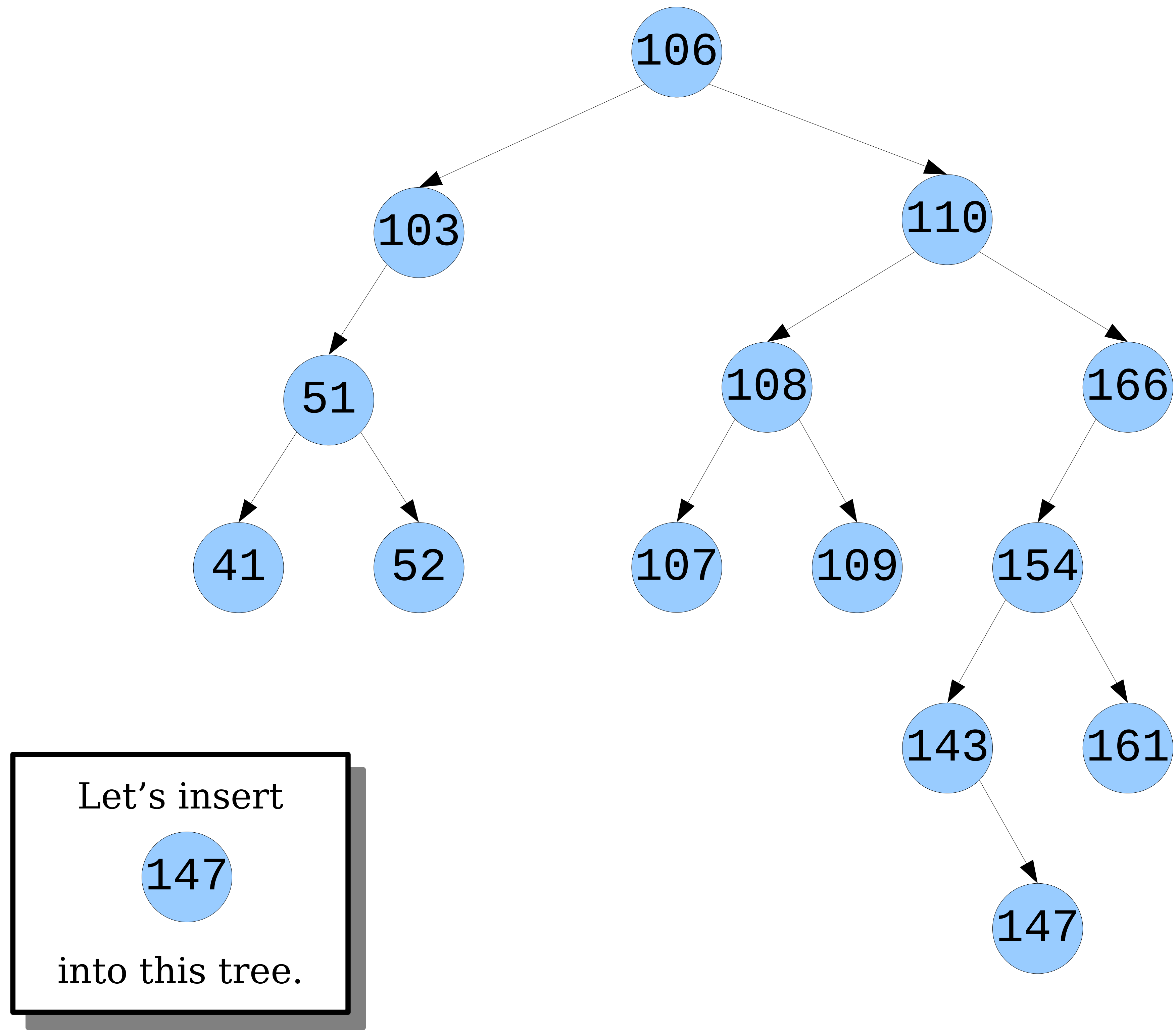
into this tree.

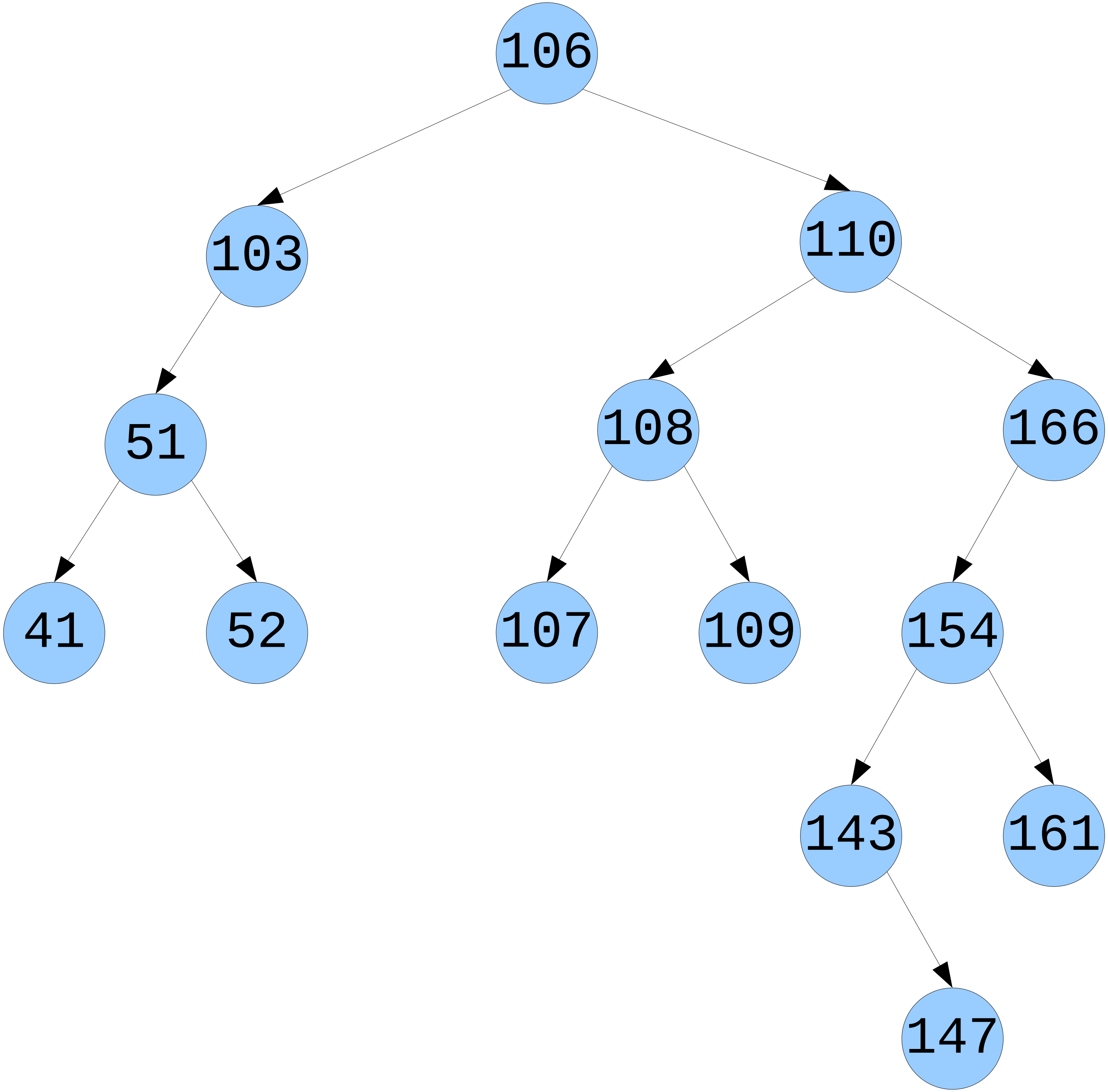


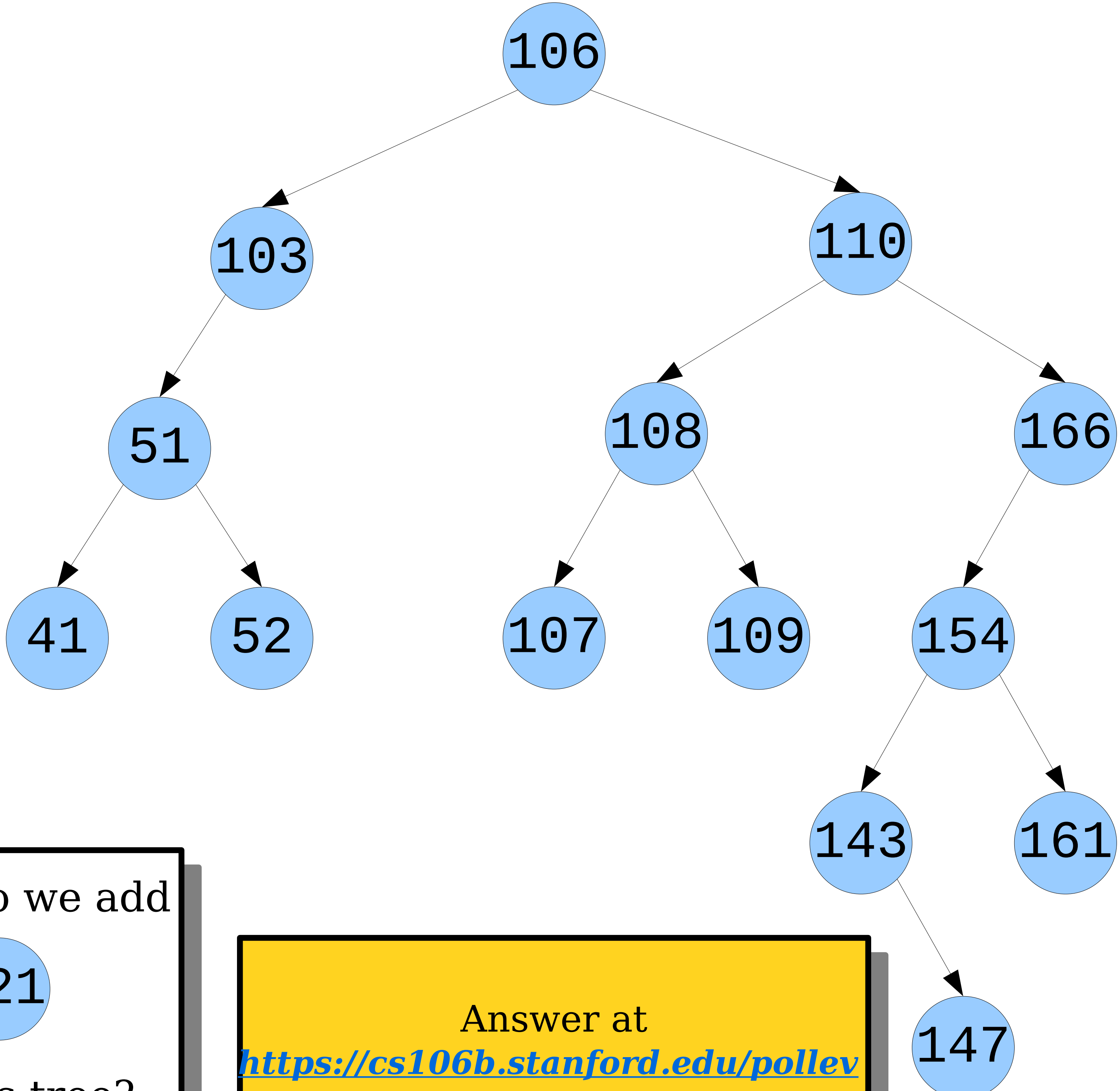
Let's insert

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.


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



Where do we add

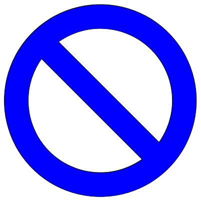
into this tree?

Answer at
https://cs106b.stanford.edu/pollev

Let's Code it Up!

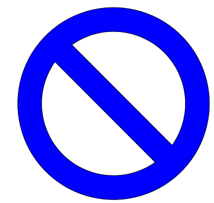
A Binary Search Tree Is Either…

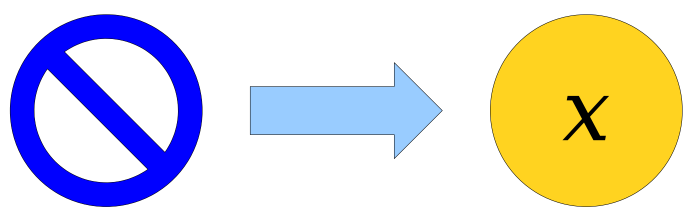
an empty tree,
represented by
`nullptr`



A Binary Search Tree Is Either…

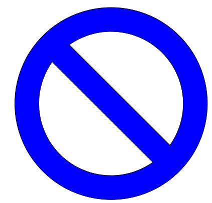
an empty tree,
represented by
`nullptr`



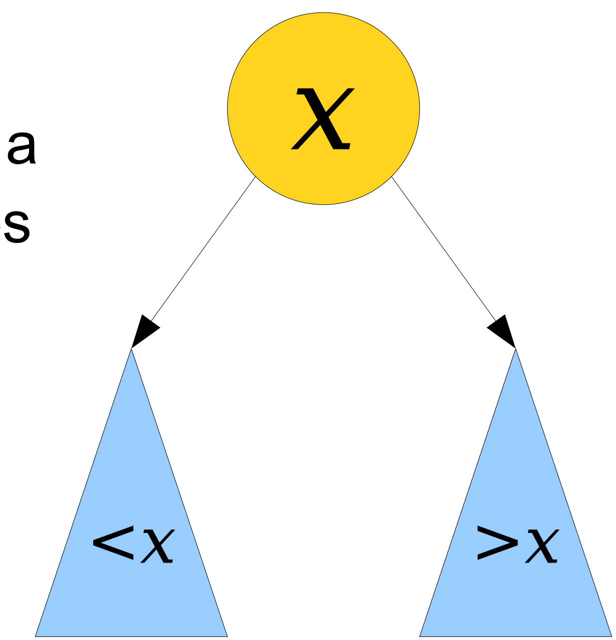


A Binary Search Tree Is Either…

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.

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 7.***
  - To follow our recommended timetable, aim to complete the labyrinth exercise and the doubly-linked list warmup by Sunday, and start working on Particle Systems by Monday.

Next Time

- ***Applied Ethics***
  - What happens when you deploy priority queues in the real world?
  - What benefits and harms can result?
  - Learn from our resident ethicist Diana Acosta Navas!