

Binary Search Trees

Part One

Taking Stock: Where Are We?

- Stack
- Queue
- Vector
- string
- PriorityQueue
- Map
- Set
- Lexicon

- ✓ Stack
- Queue
- Vector
- string
- PriorityQueue
- Map
- Set
- Lexicon

Could do this with a dynamic array (in class) or a linked list (exercise to the reader).

- ✓ Stack
- ✓ Queue
- Vector
- string
- PriorityQueue
- Map
- Set
- Lexicon

Could do this with a linked list (in class) or as a dynamic array (exercise to the reader).

Almost always backed
by dynamic arrays.

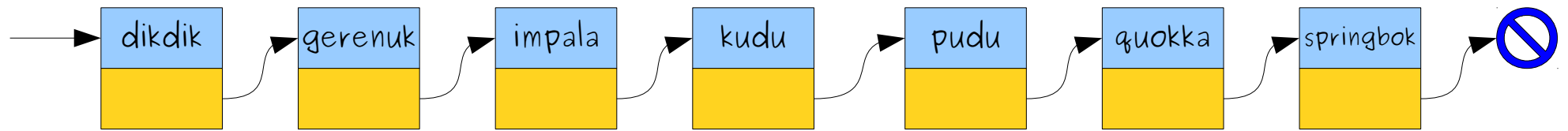
- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- PriorityQueue
- Map
- Set
- Lexicon

As a binary heap
on top of a
dynamic array
(Assignment 5!)

- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- ✓ PriorityQueue
- Map
- Set
- Lexicon

- ✓ Stack
- ✓ Queue
- ✓ Vector
- ✓ string
- ✓ PriorityQueue
- Map**
- Set**
- Lexicon**

Implementing Nonlinear Containers



What is the average cost of looking up an element in this list?

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

Intuition: Most elements are far from the front.

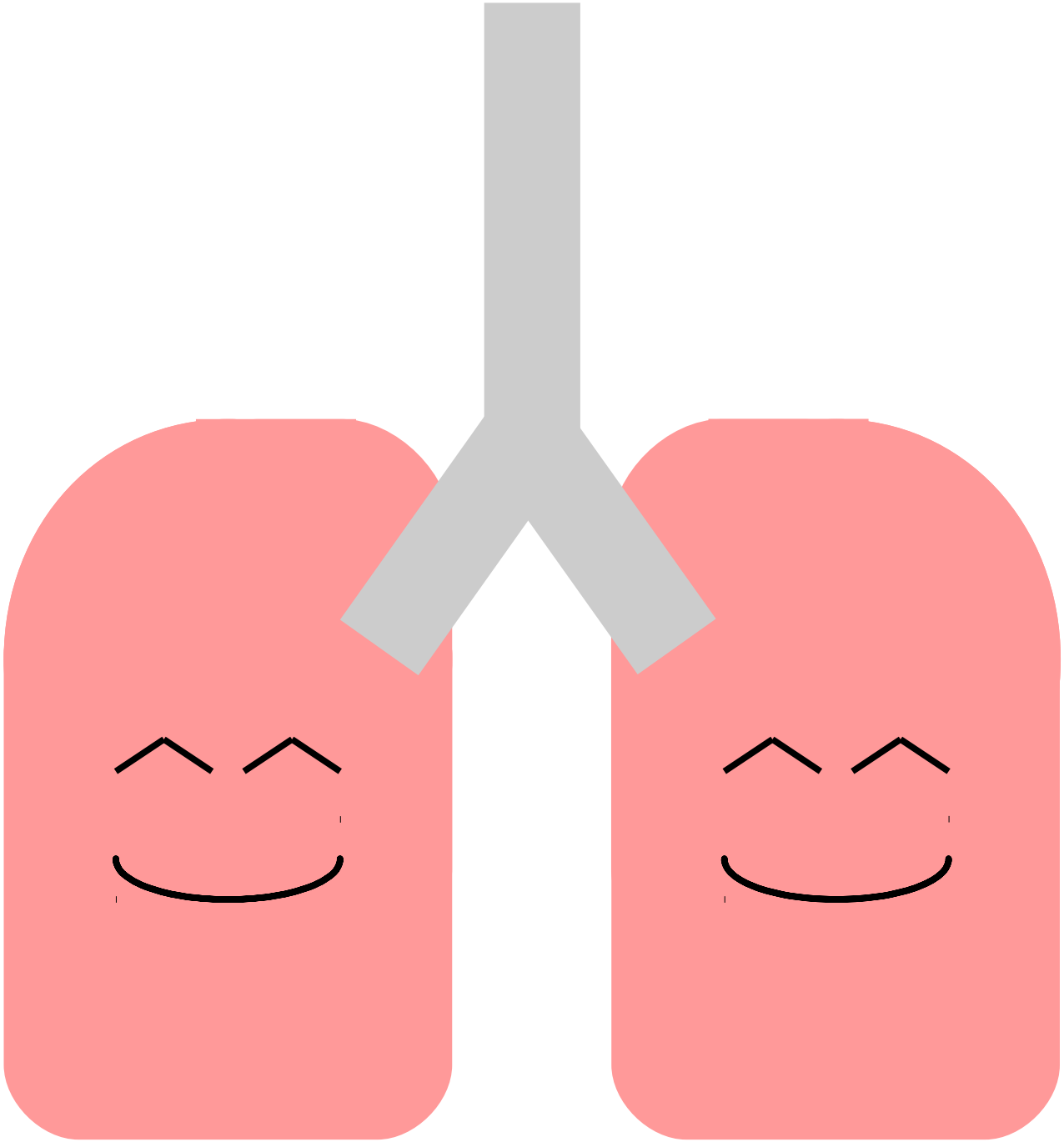
Can you chain a bunch of objects together
so that most of them are near the front?

An Interactive Analogy

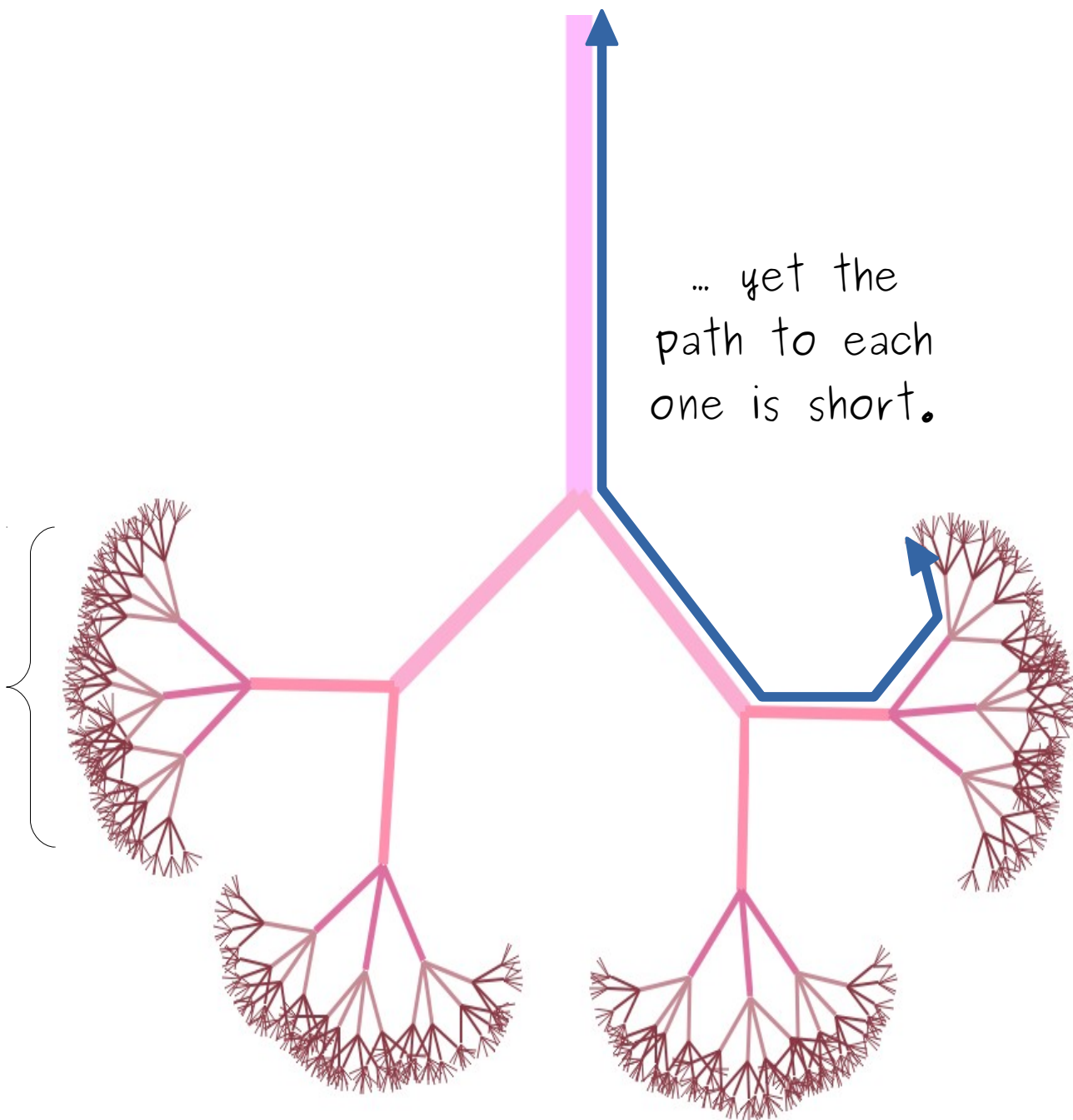
Take a deep breath.

And exhale.

Feel nicely oxygenated?

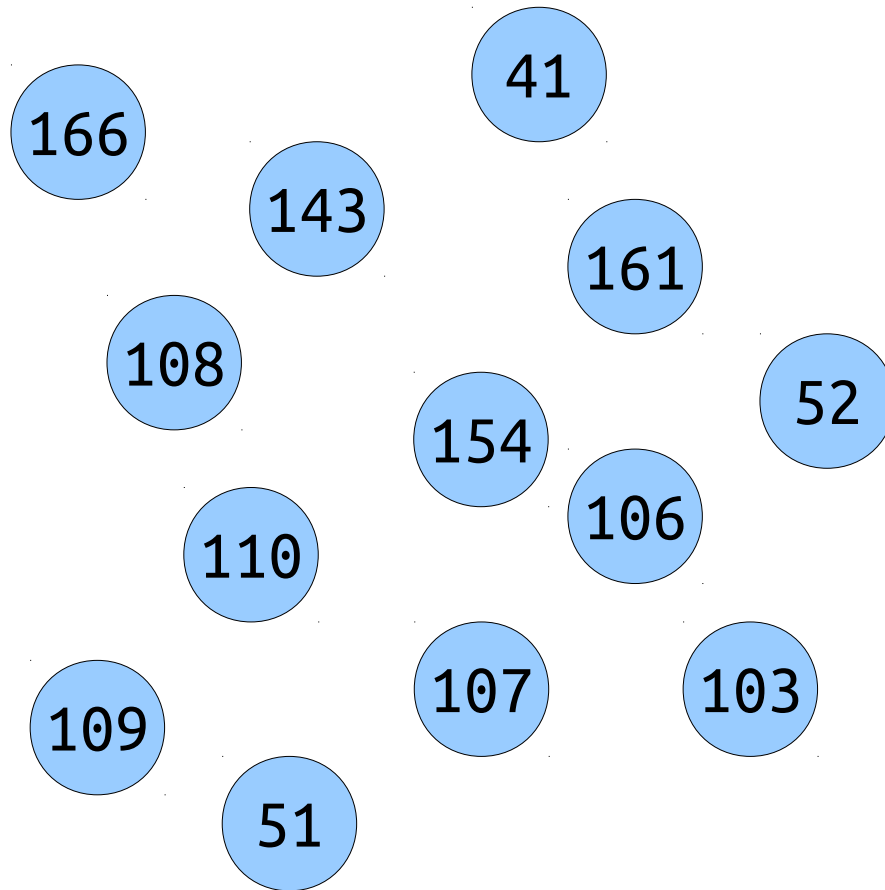


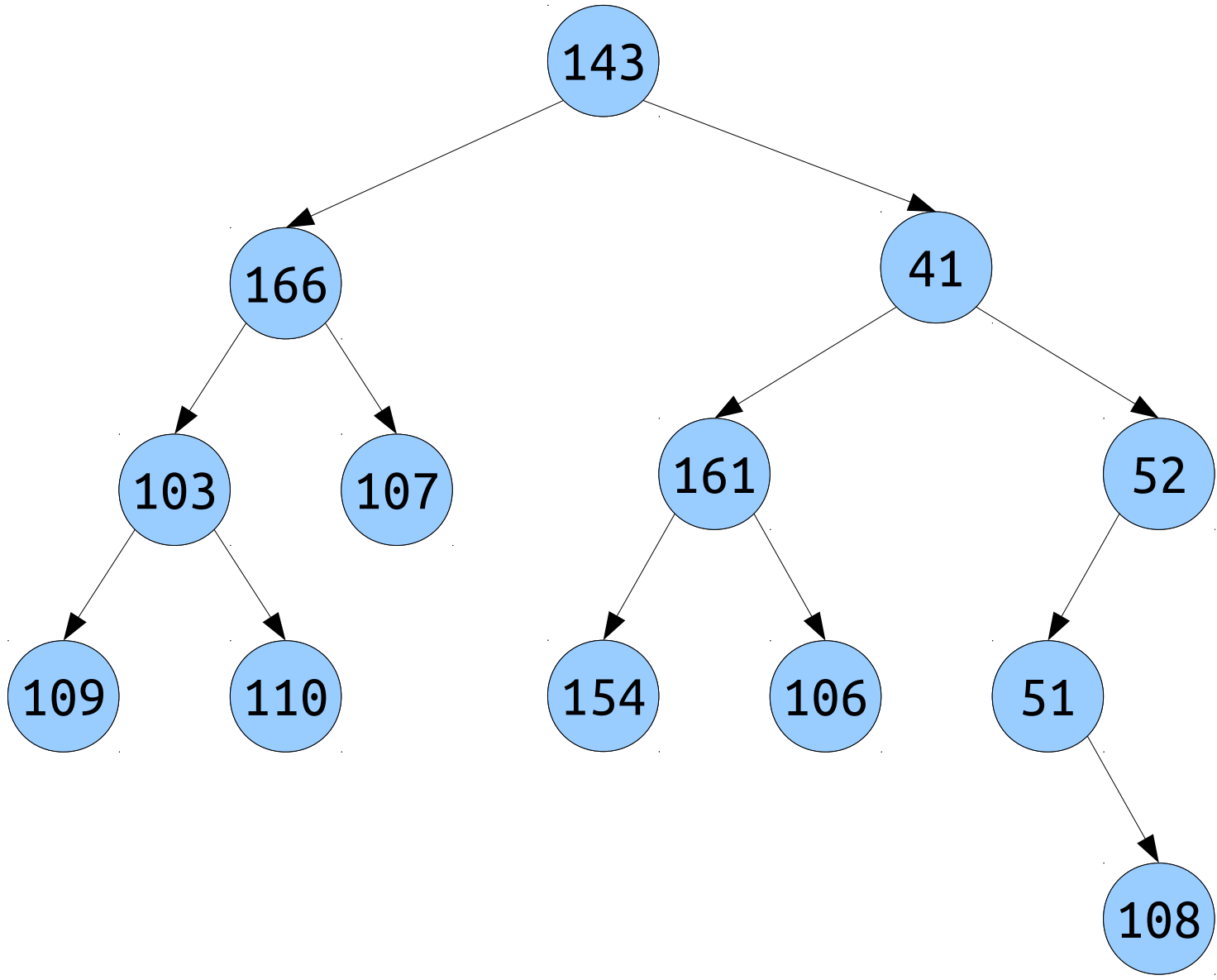
Your lungs
have about
500 million
alveoli...

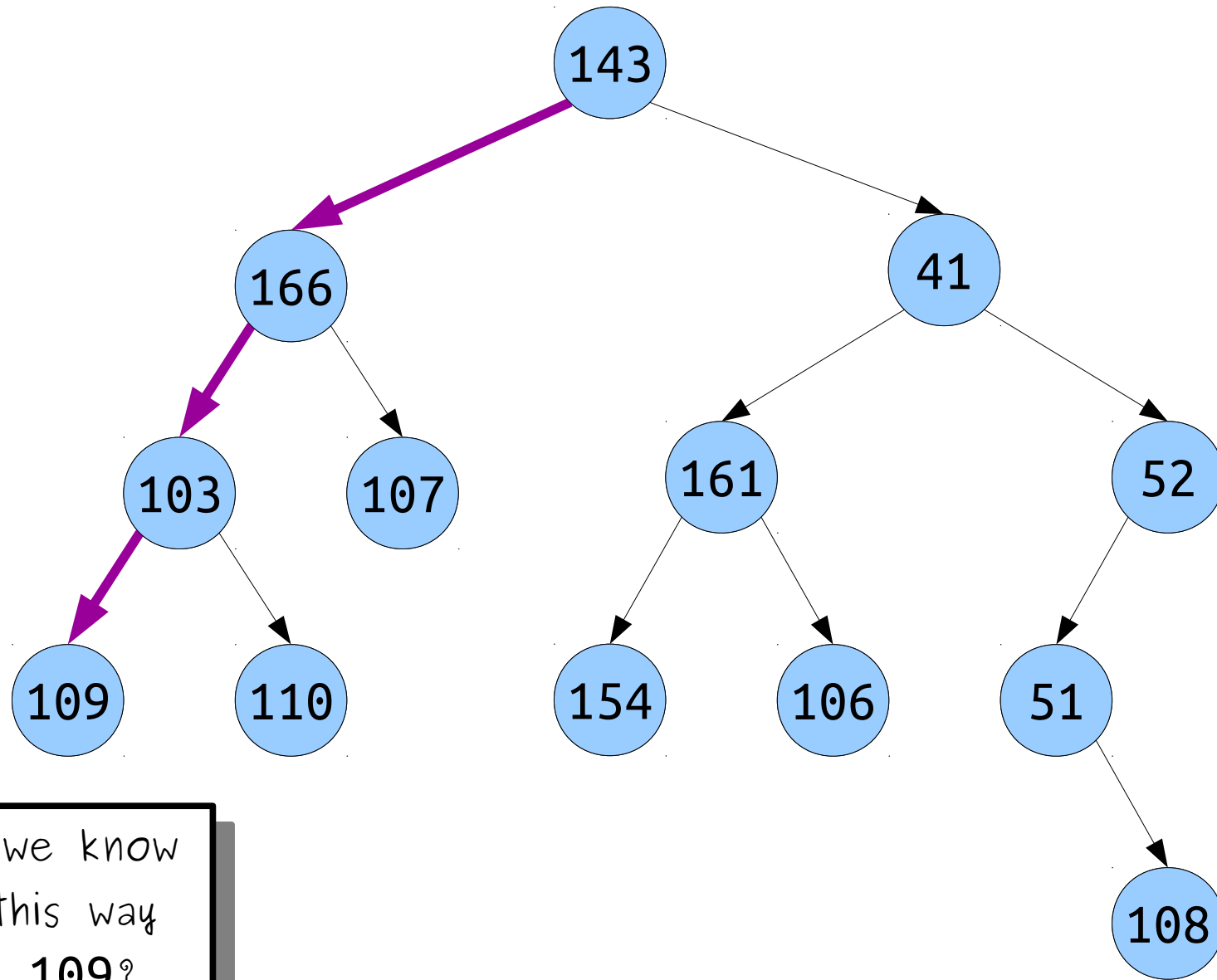


Key Idea: Most elements in a tree are near the ***root*** (top) of the tree.

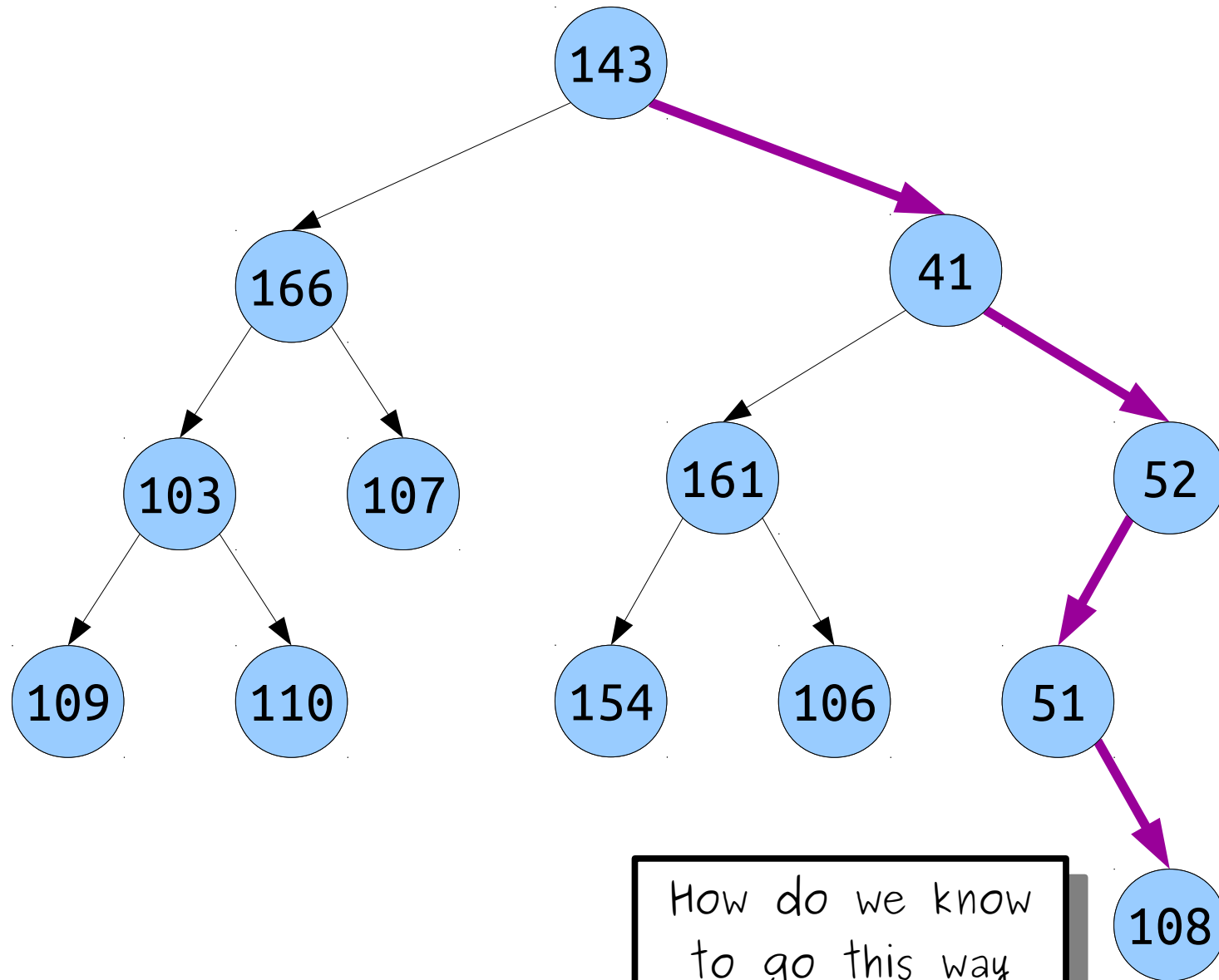
Harnessing this Insight





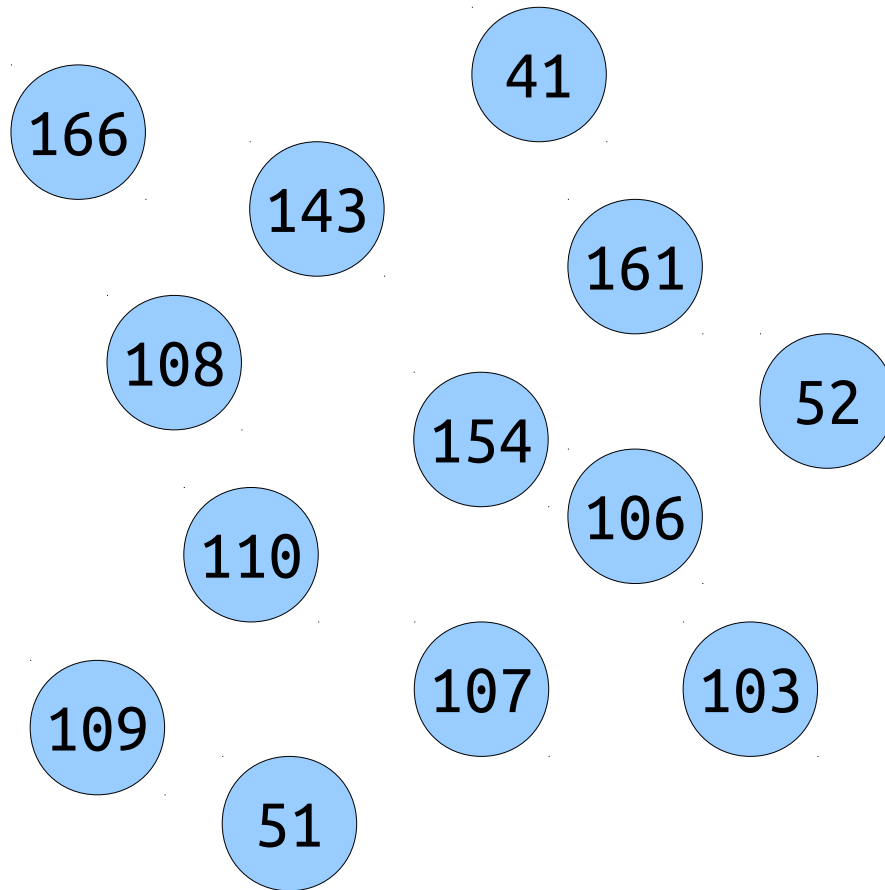


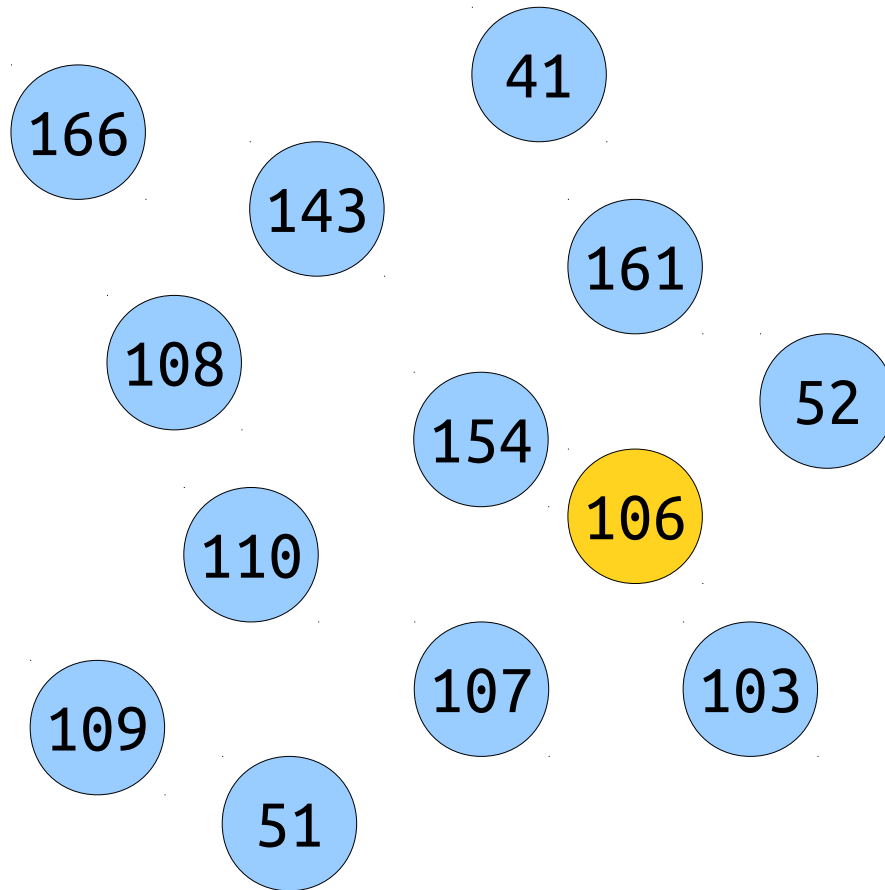
How do we know
to go this way
to get **109**?

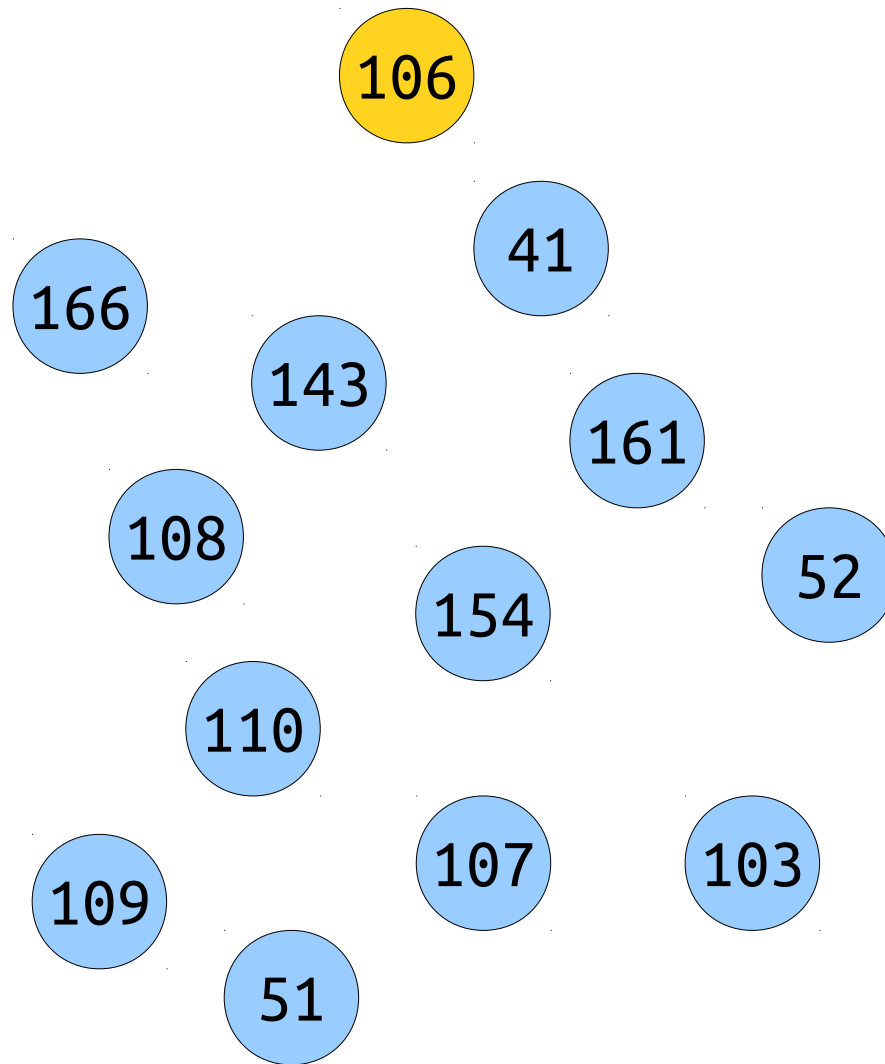


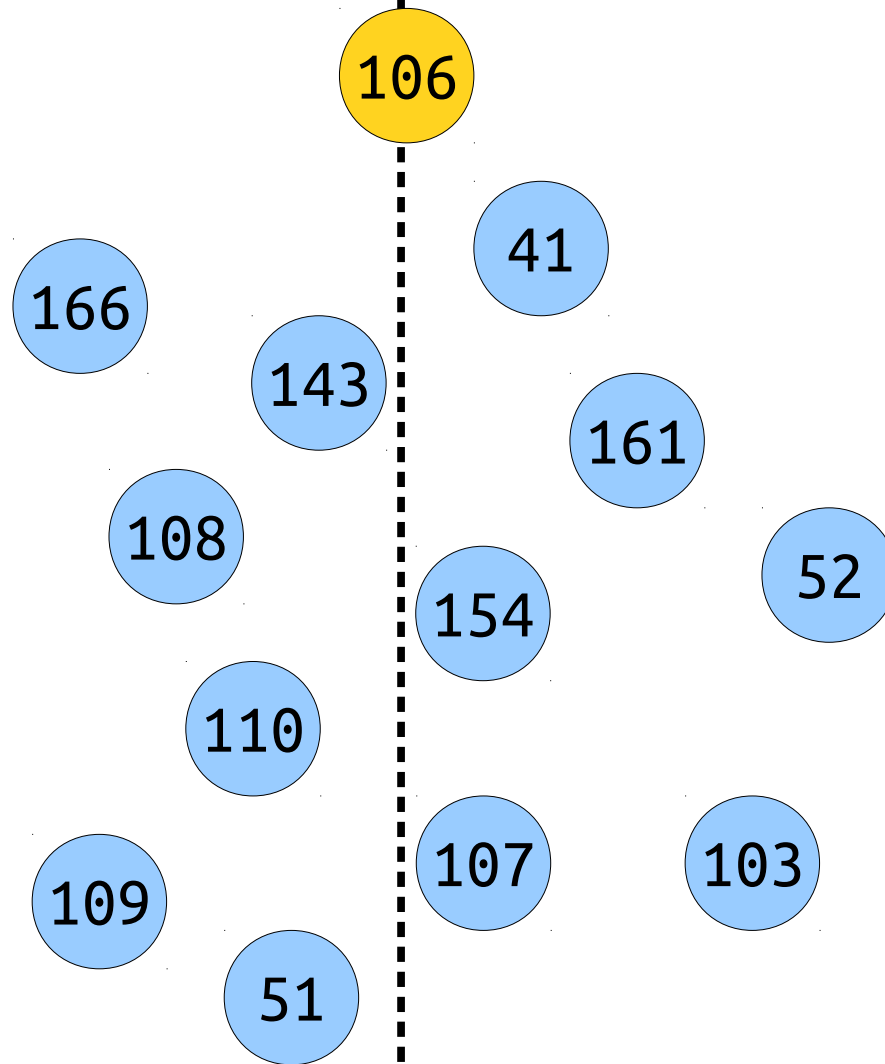
How do we know
to go this way
to get 108?

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









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

52

51

154

110

109

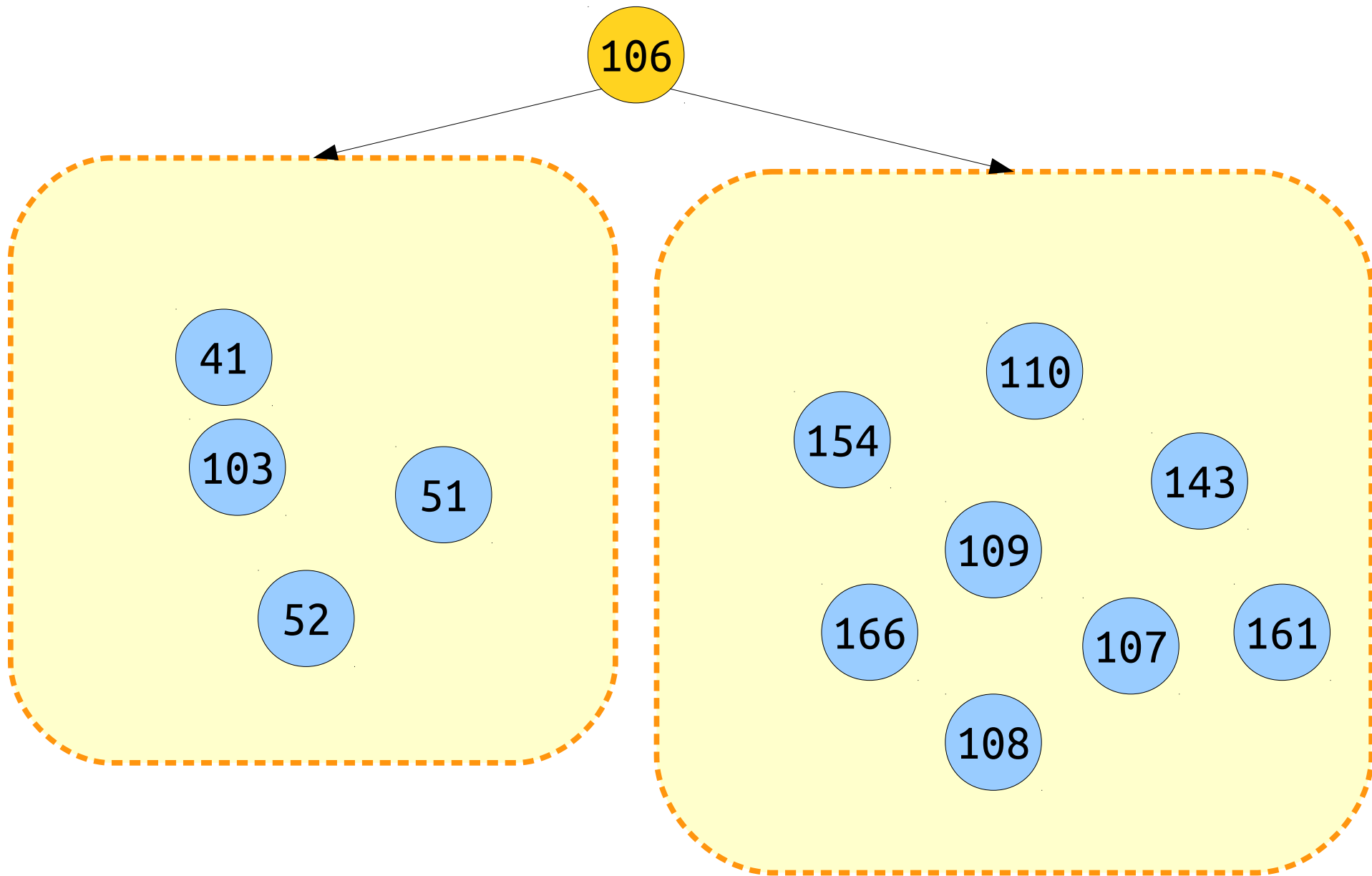
143

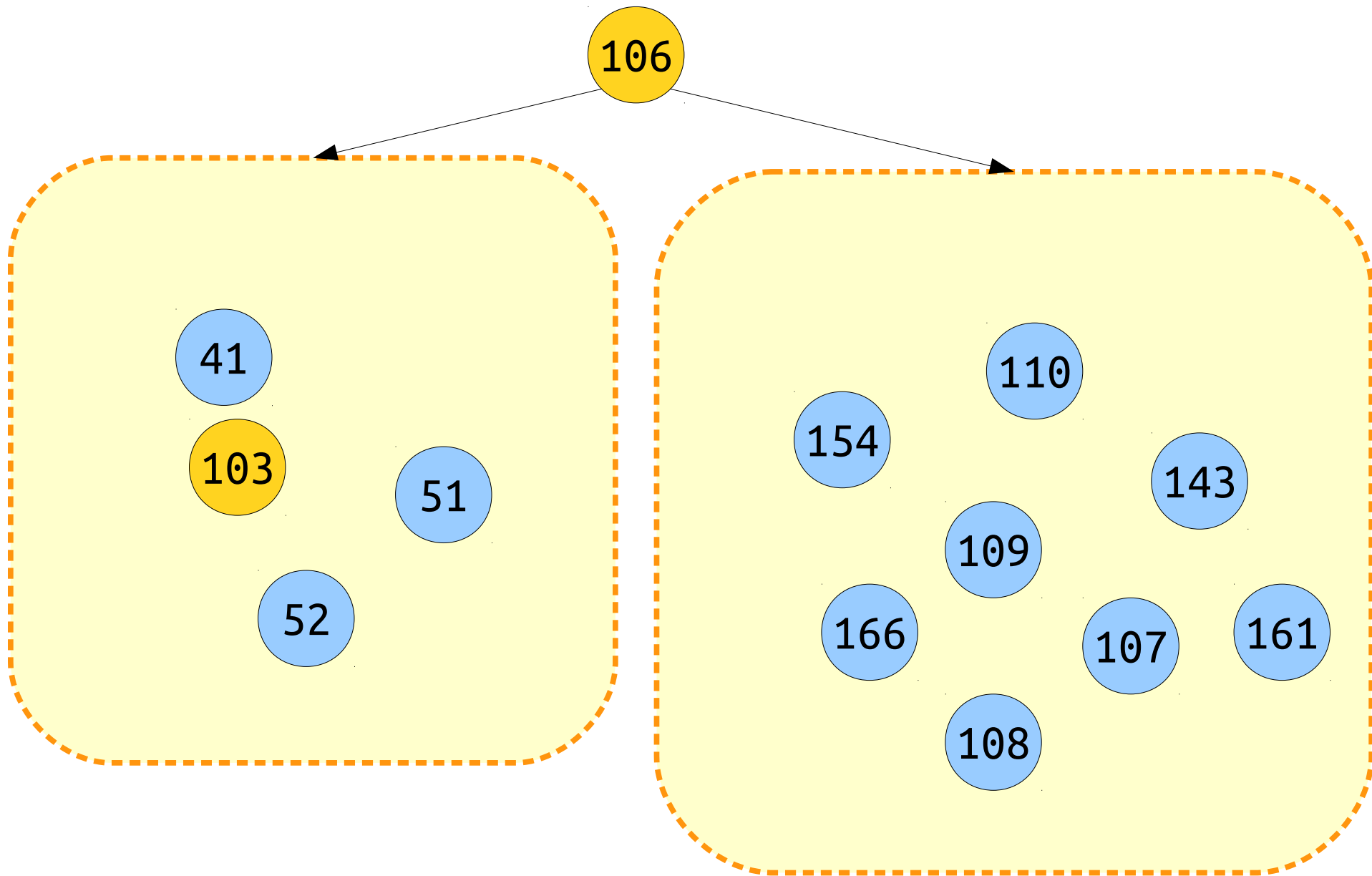
166

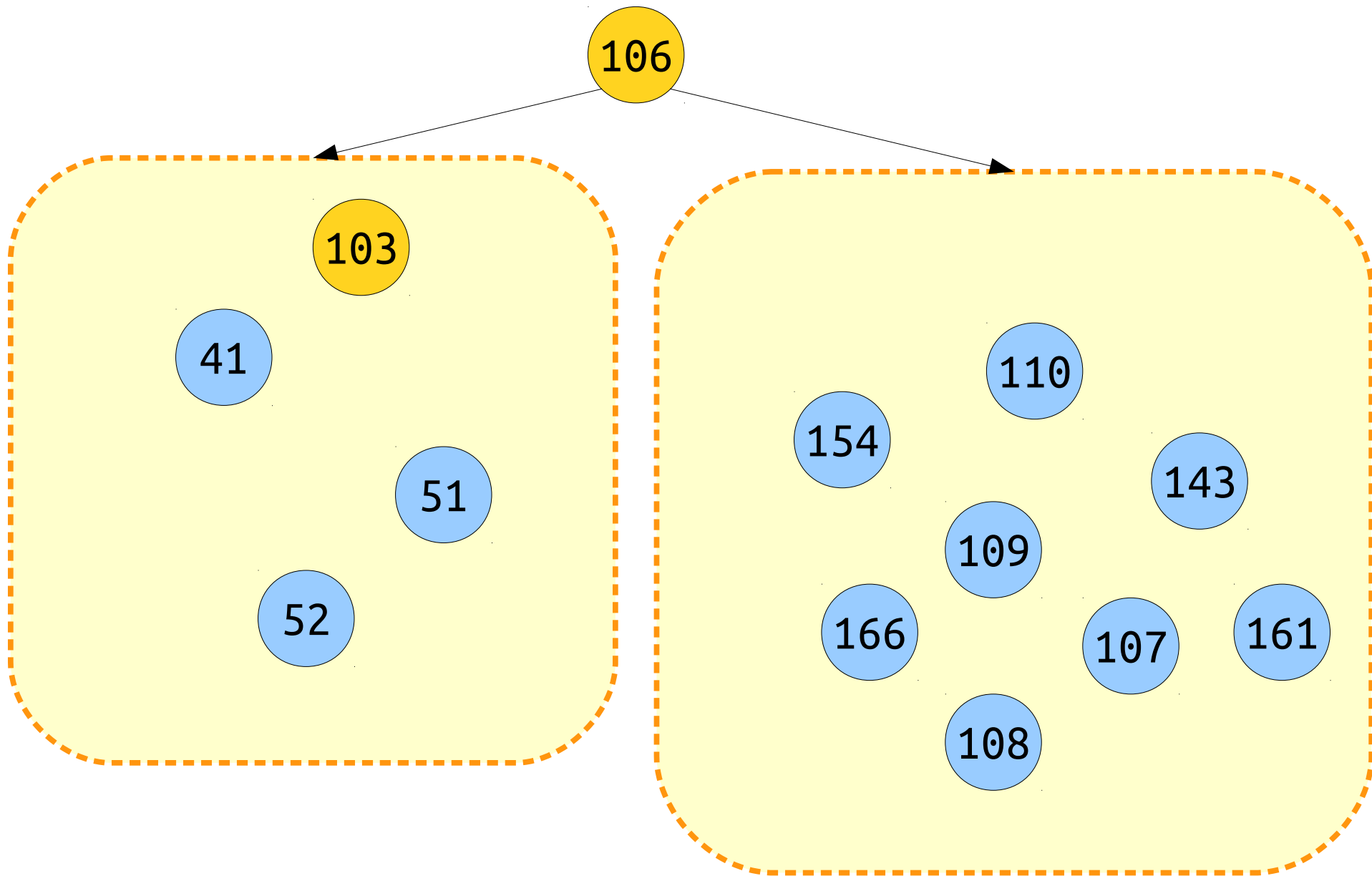
107

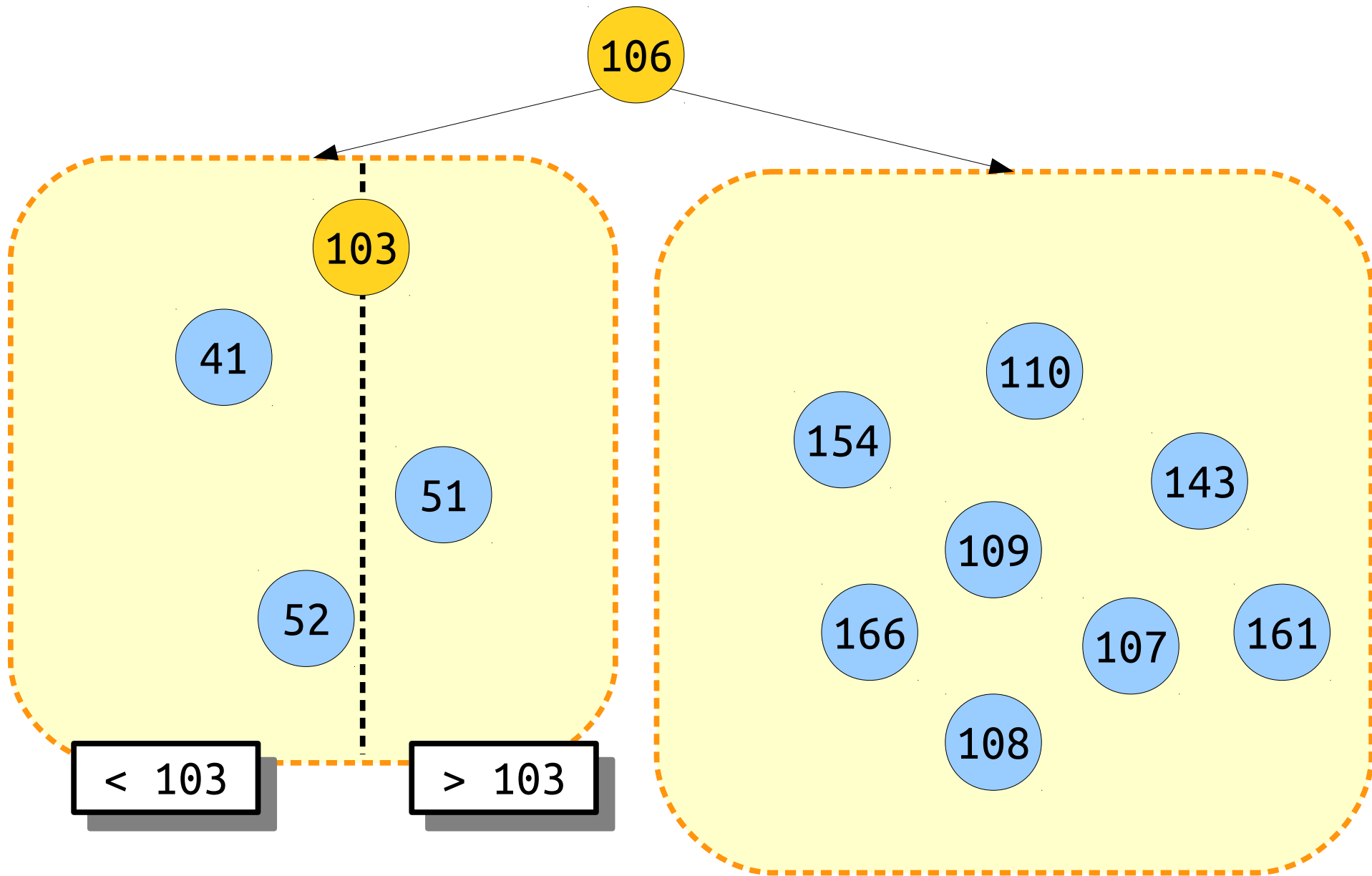
161

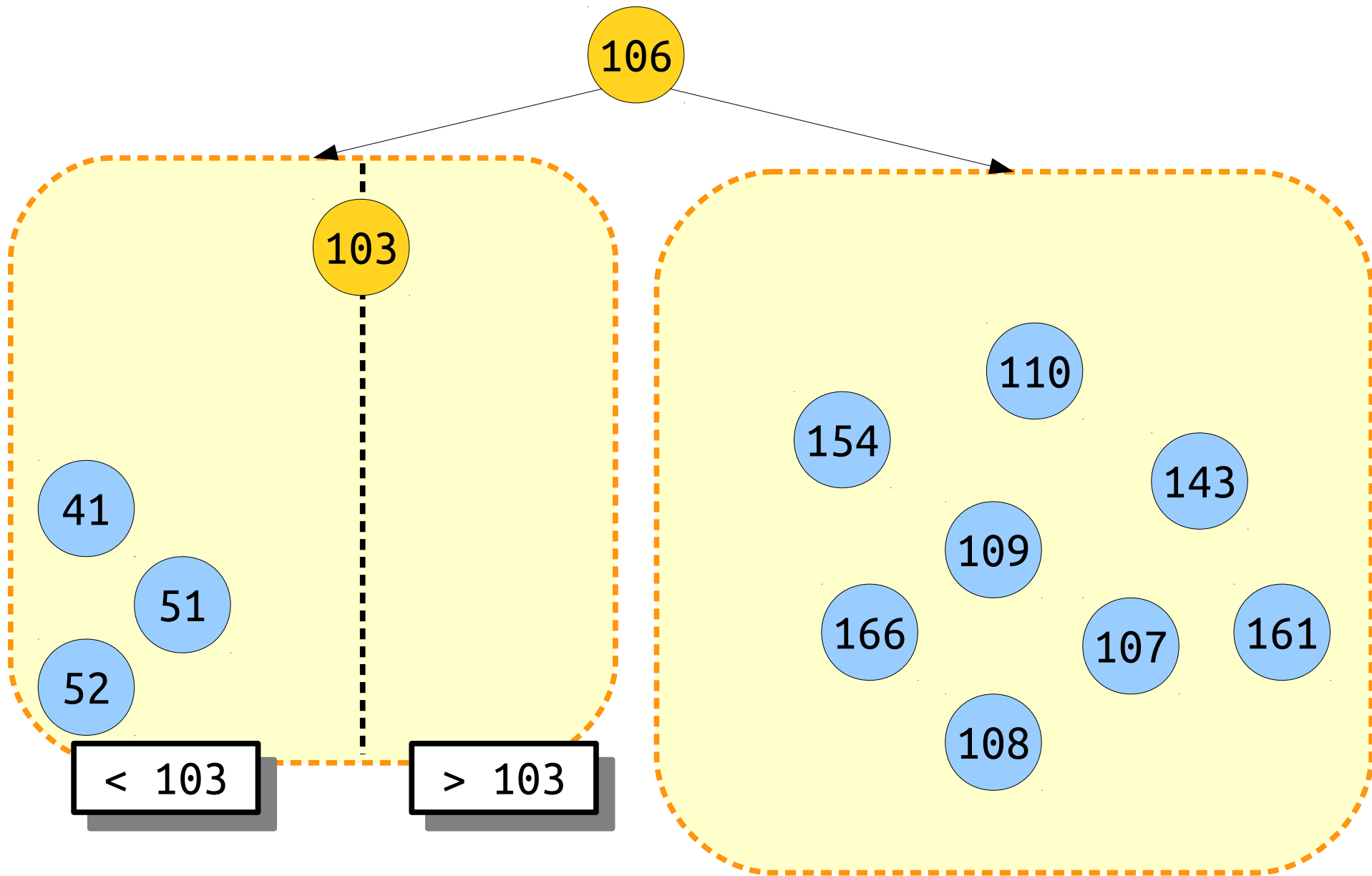
108

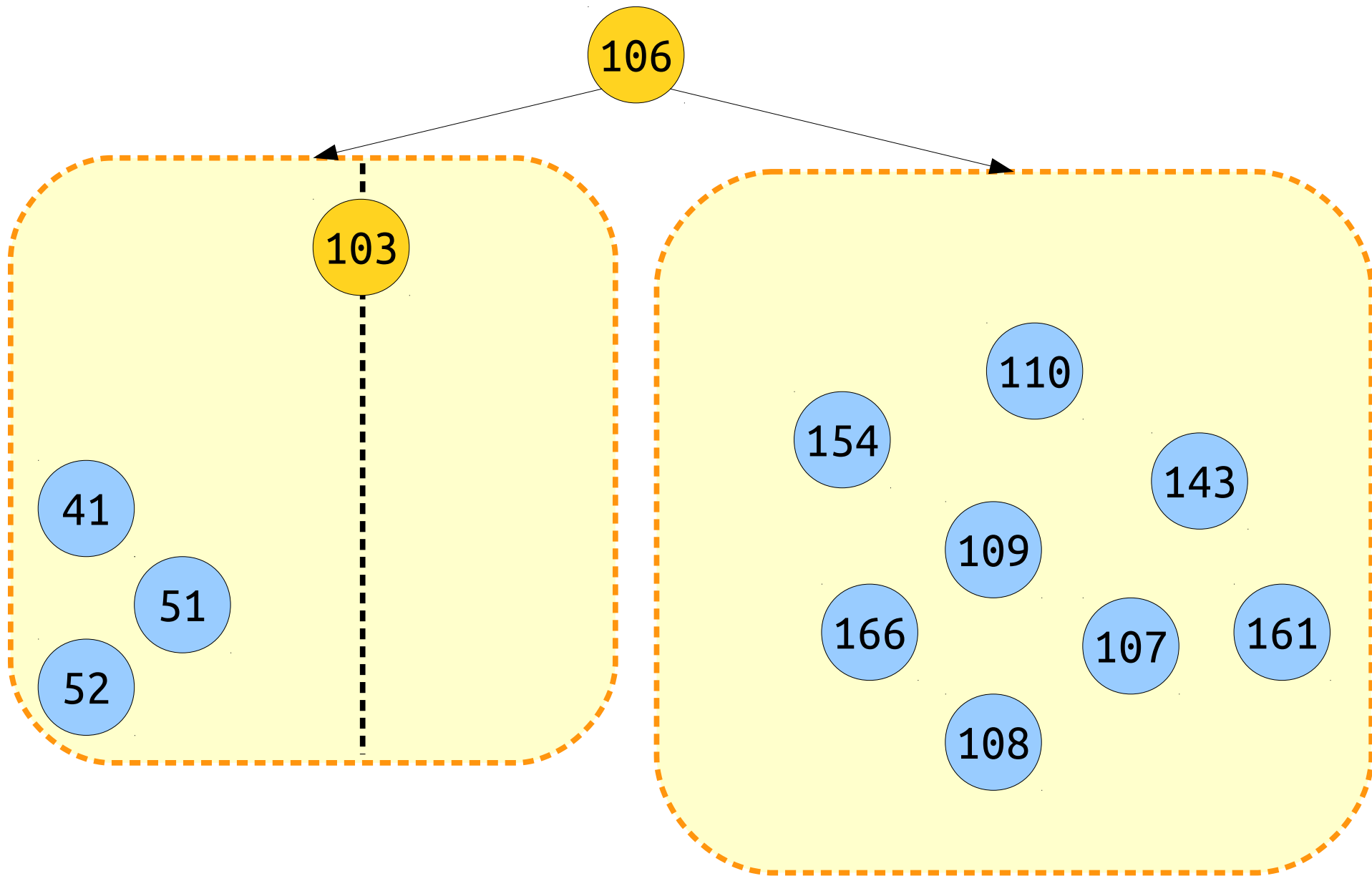


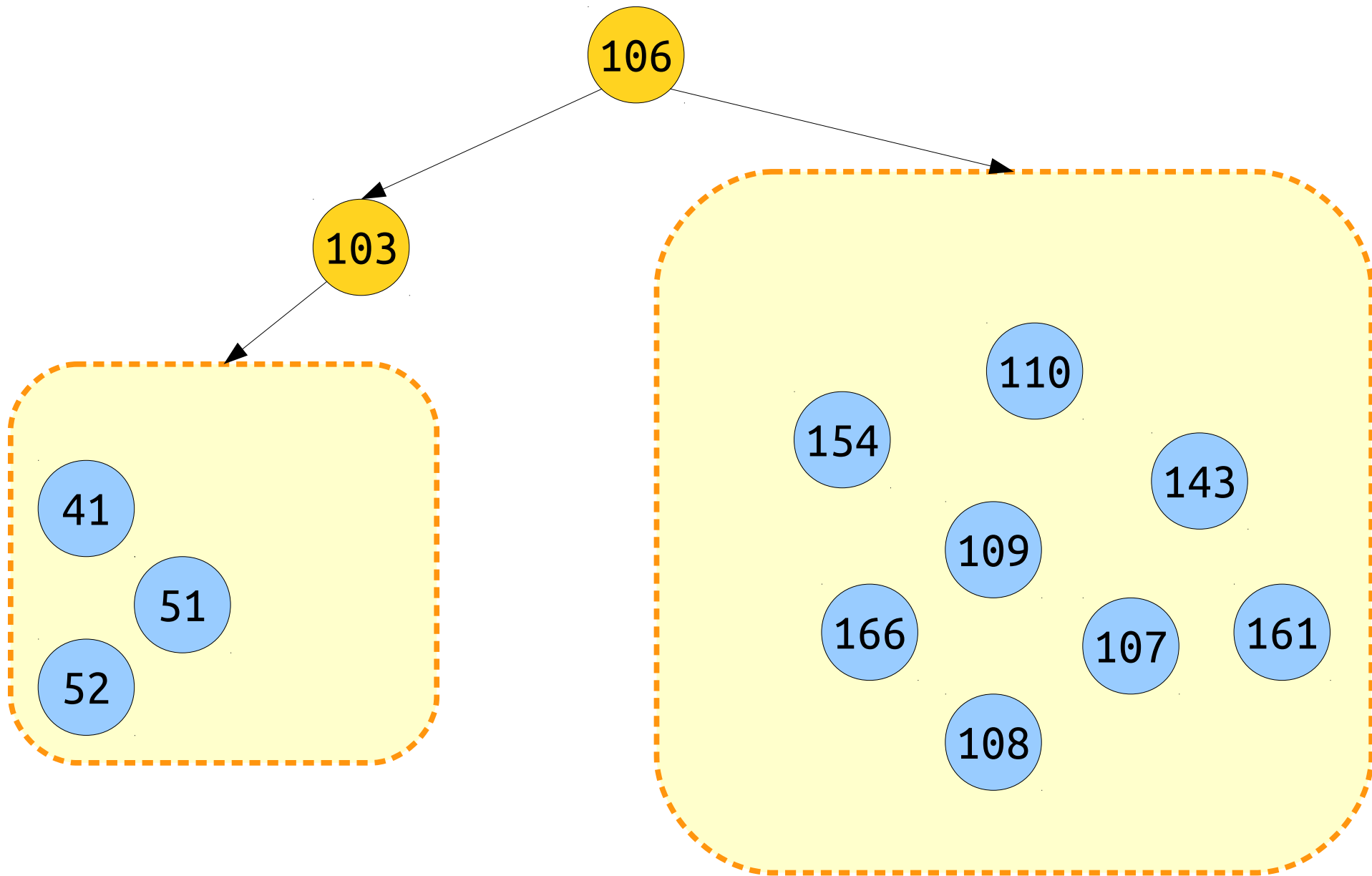


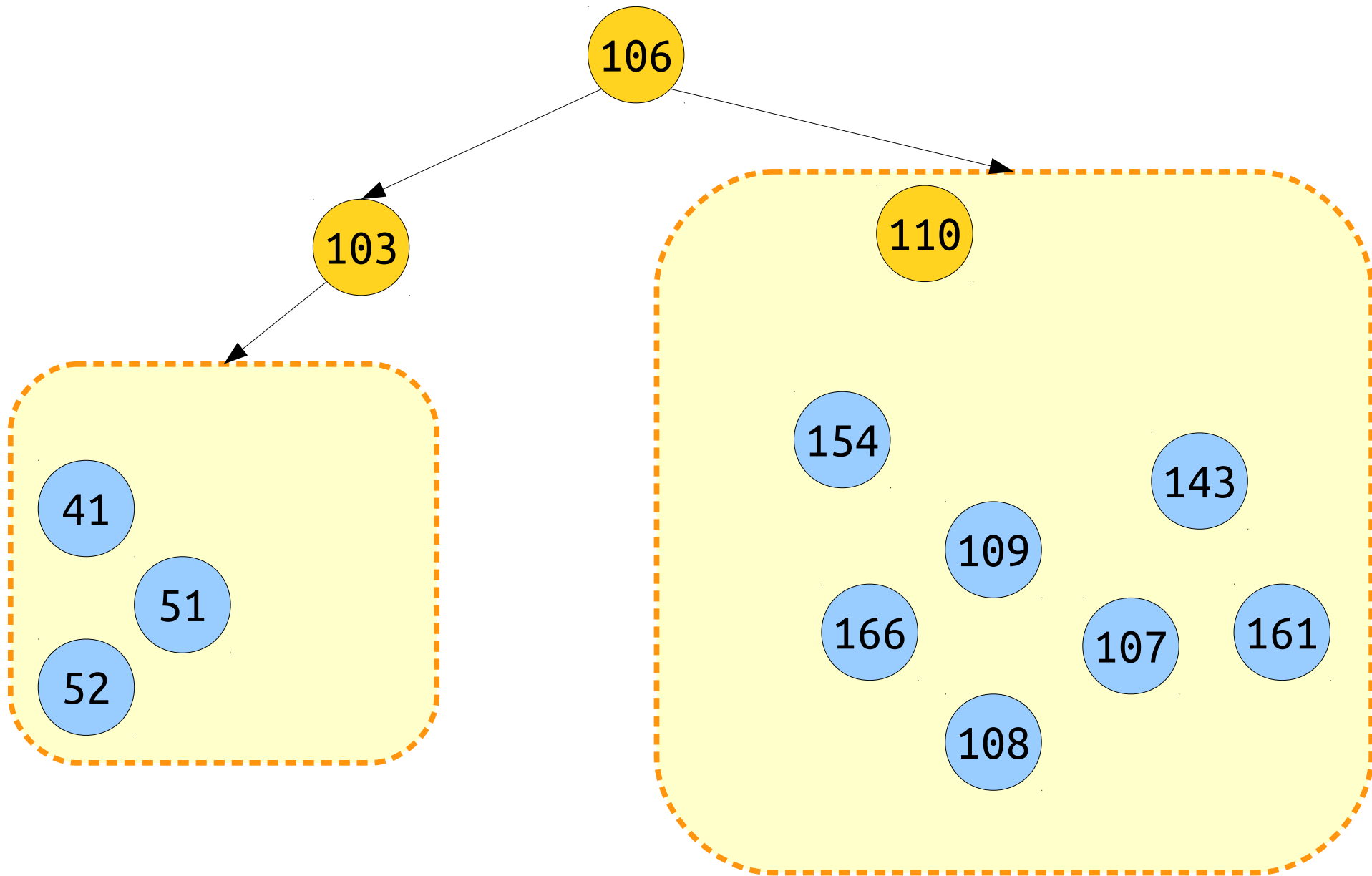


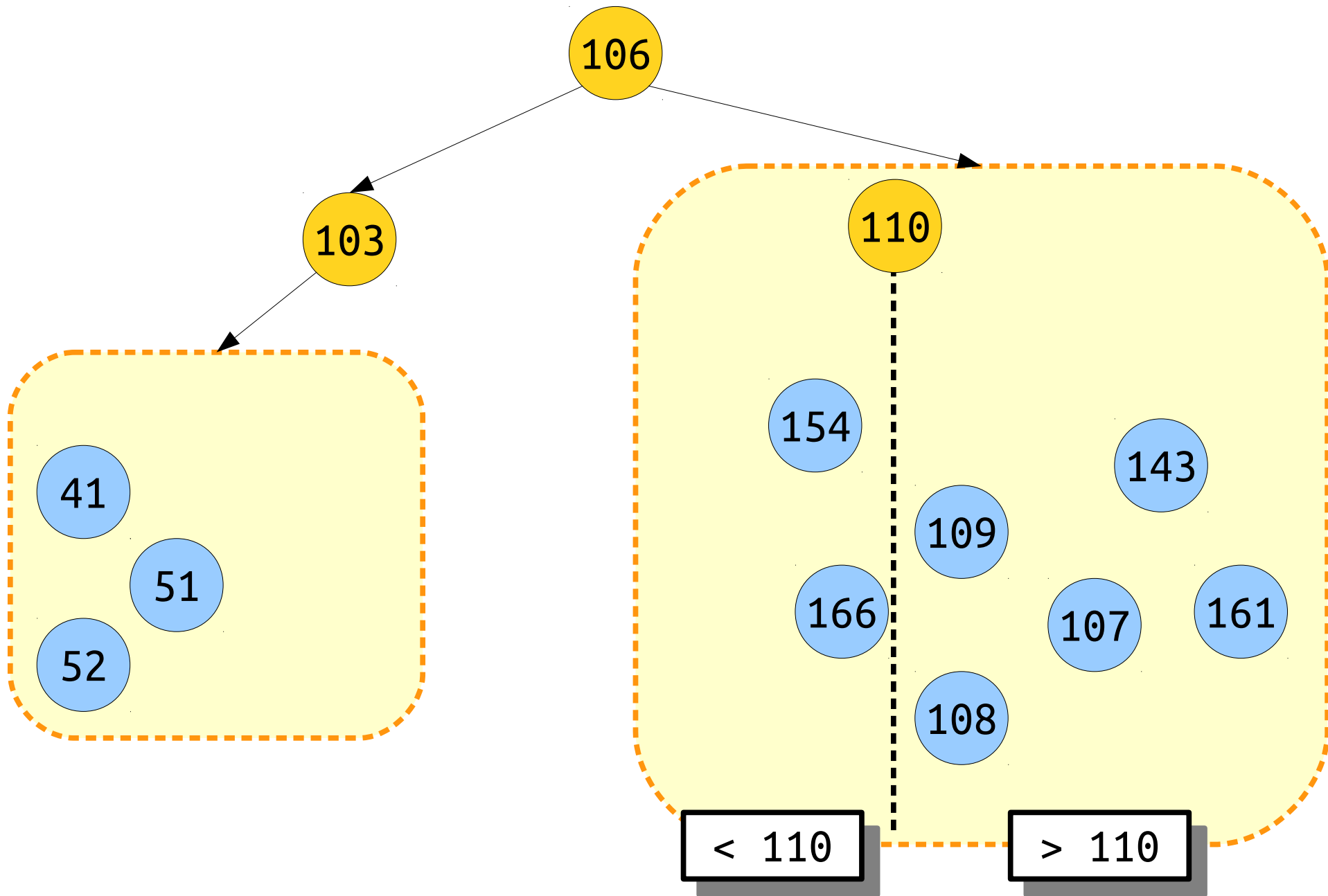


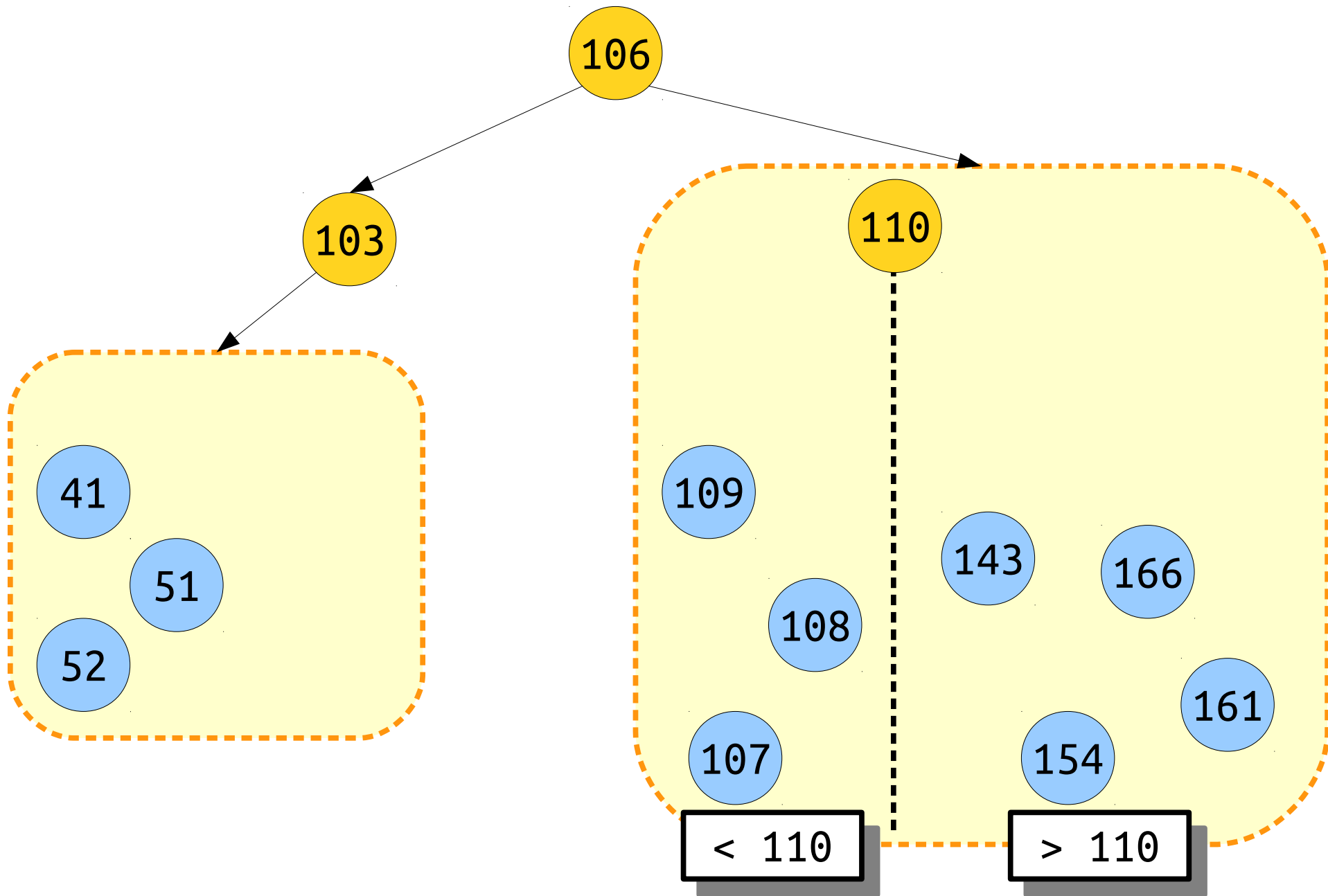


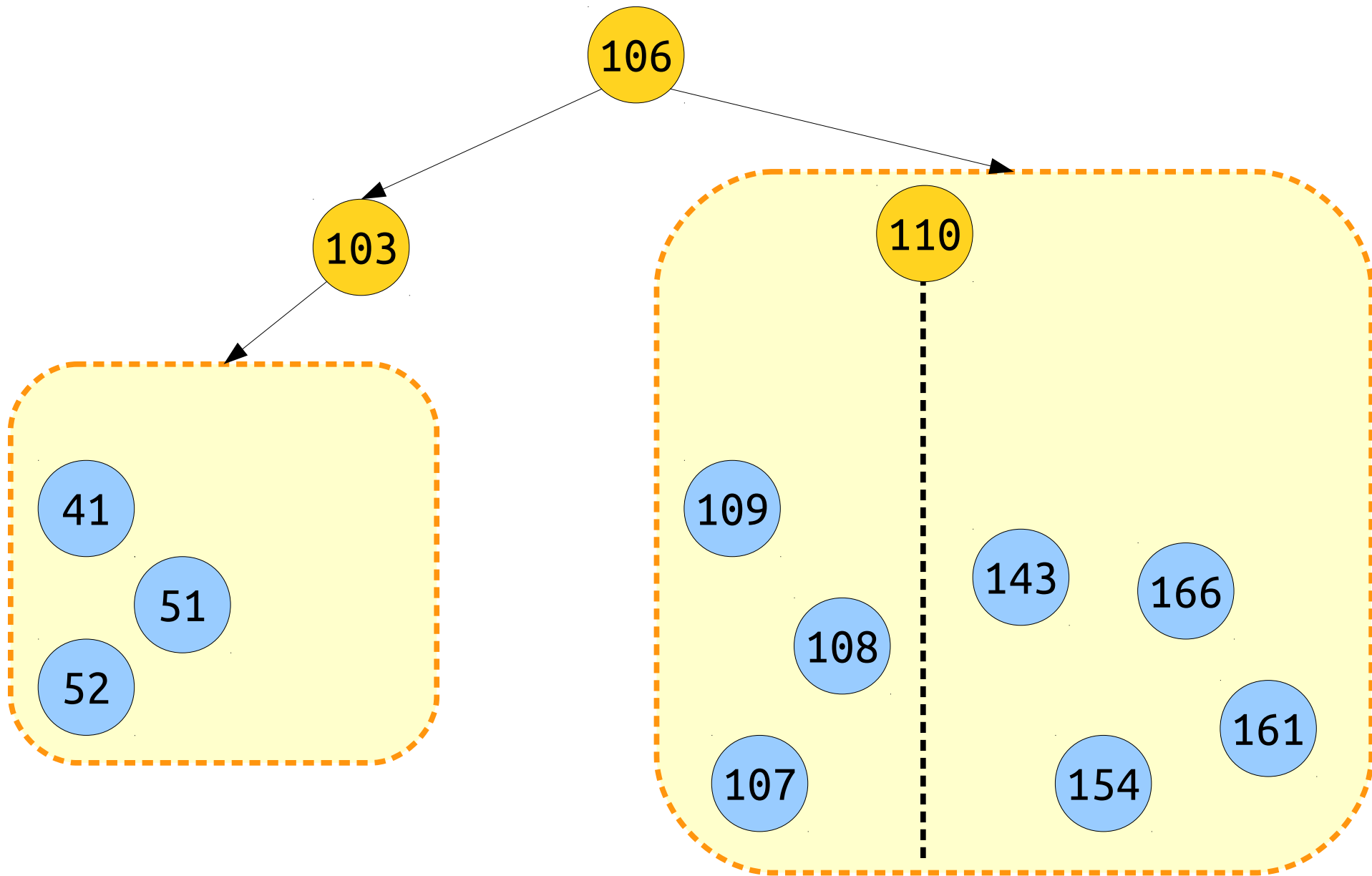


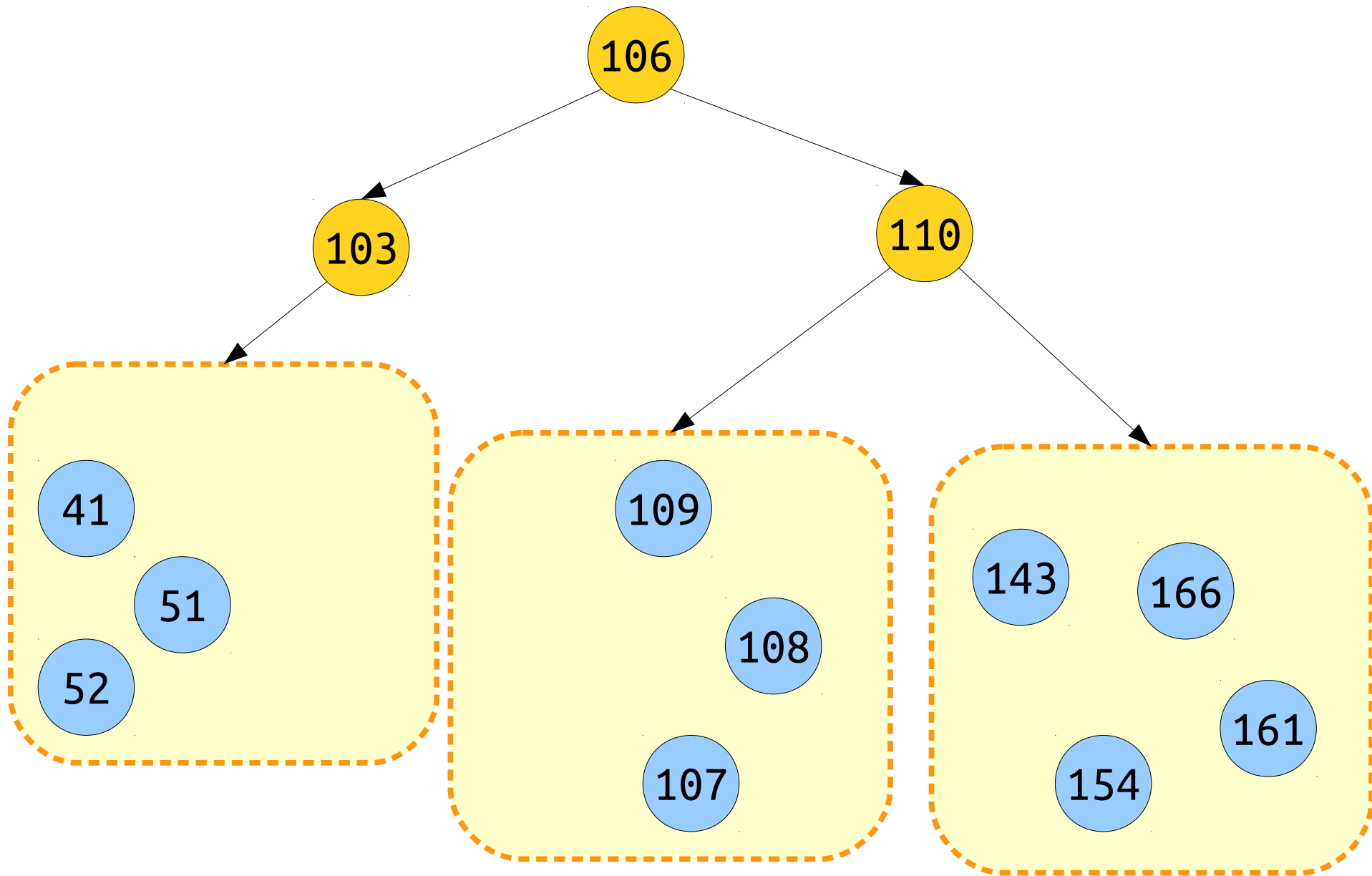


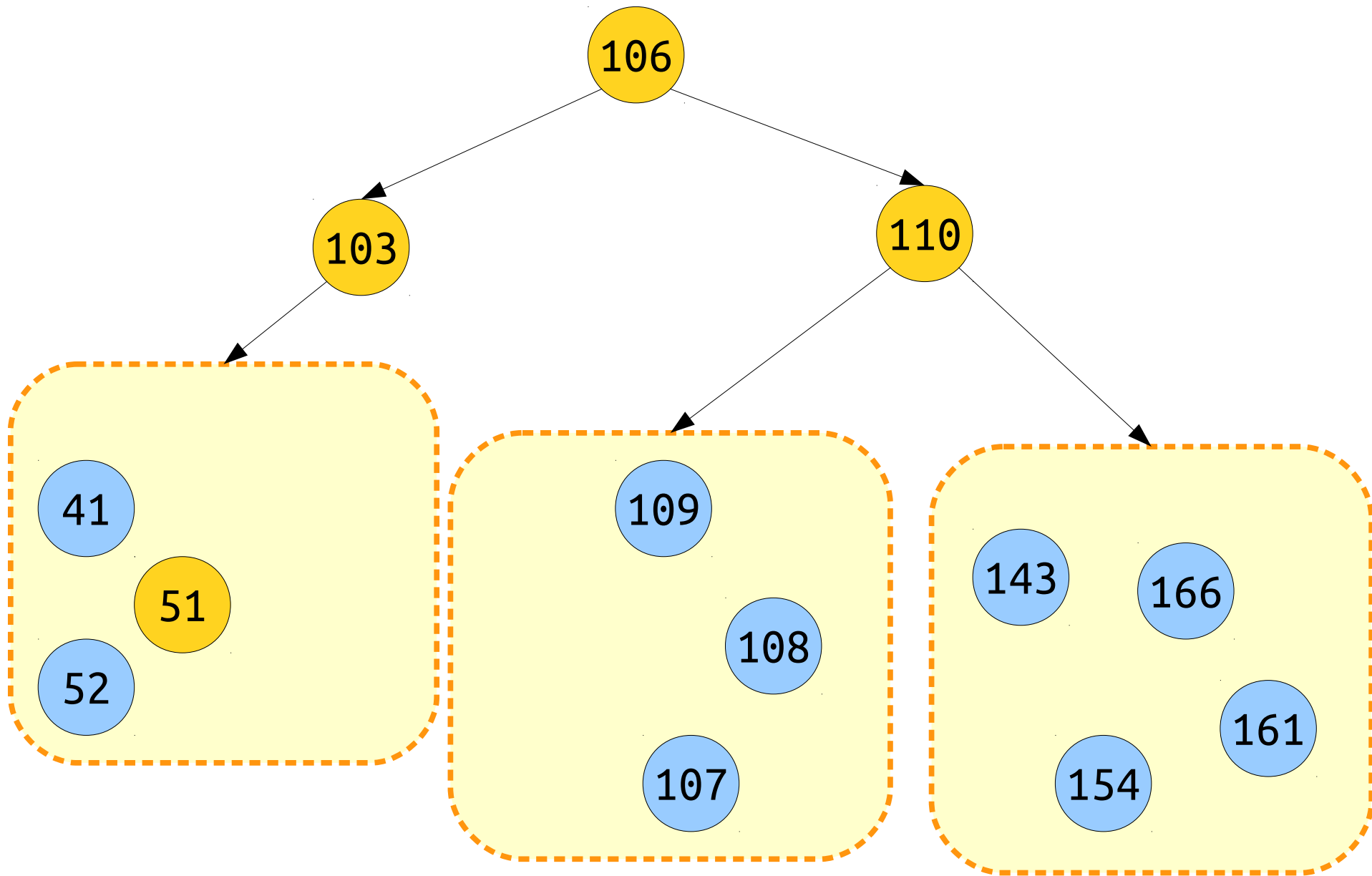


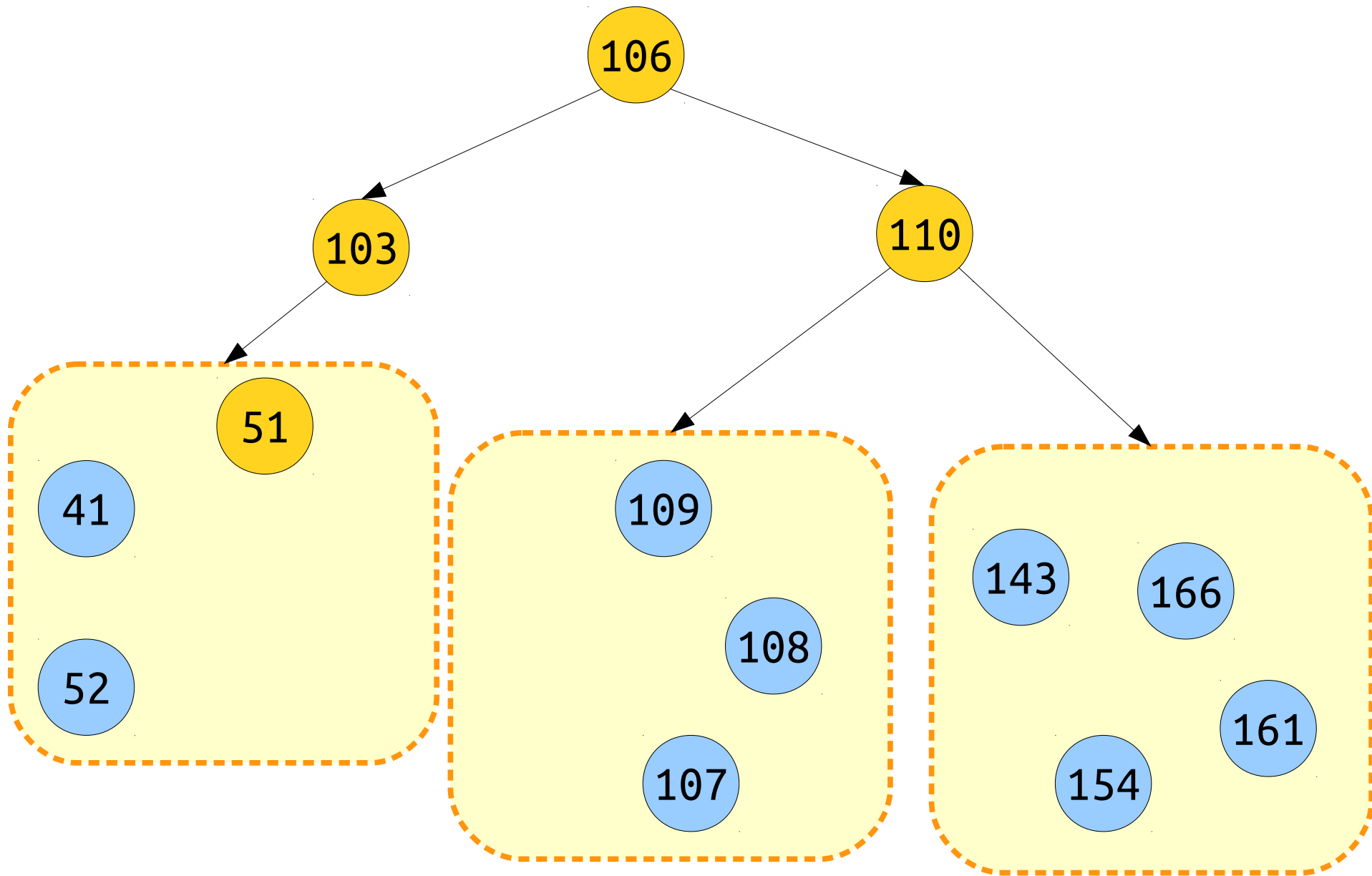


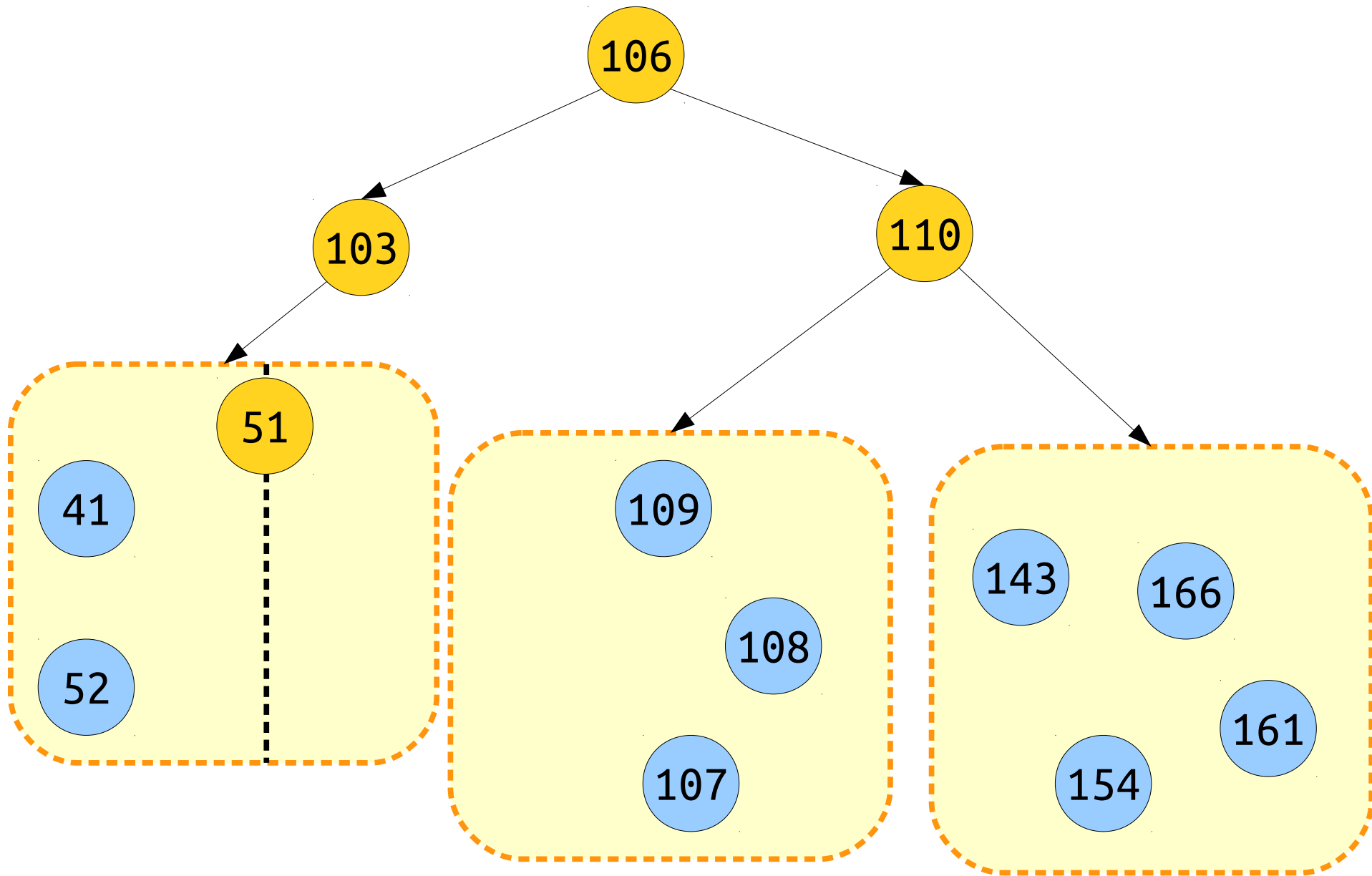


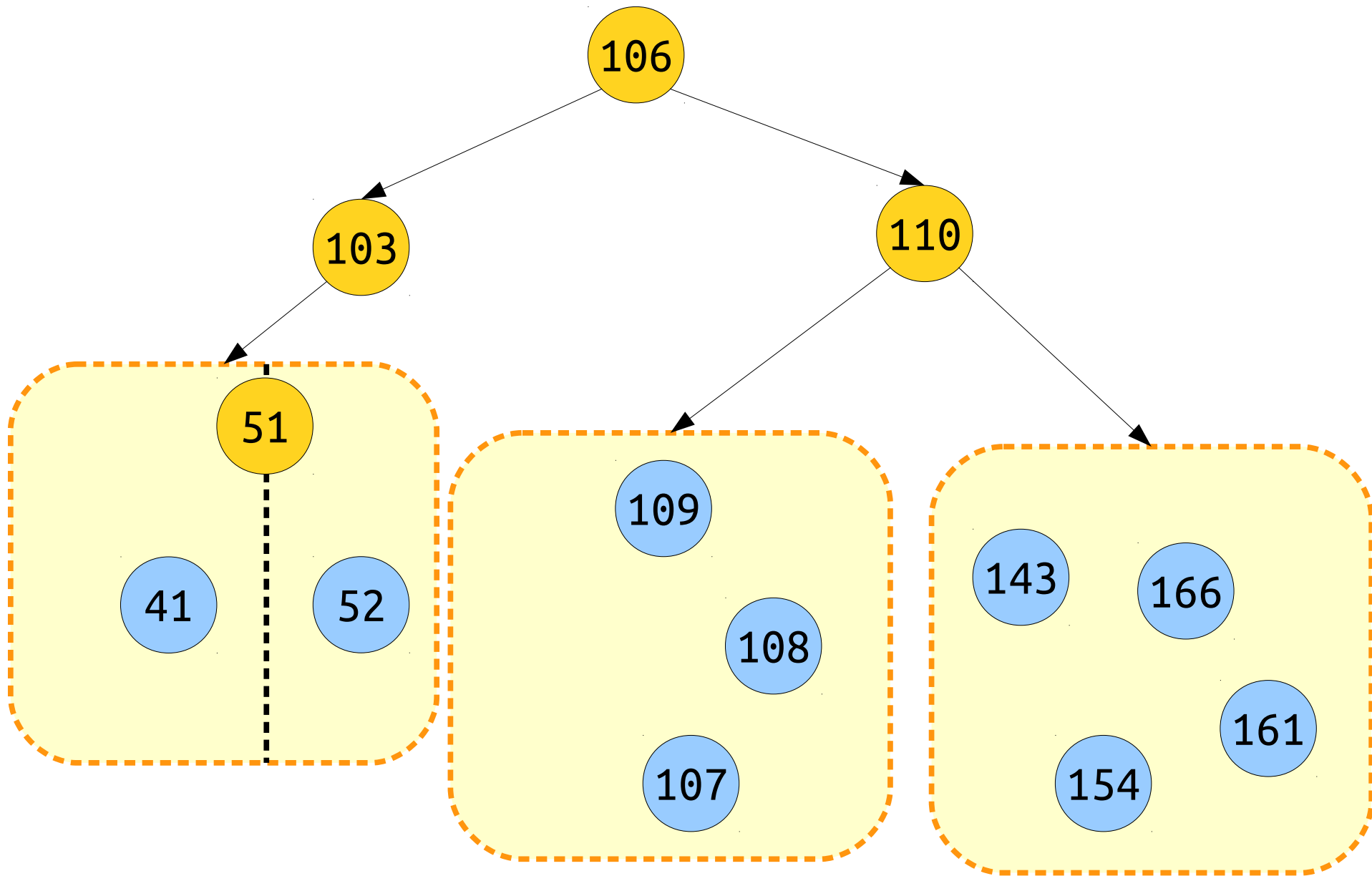


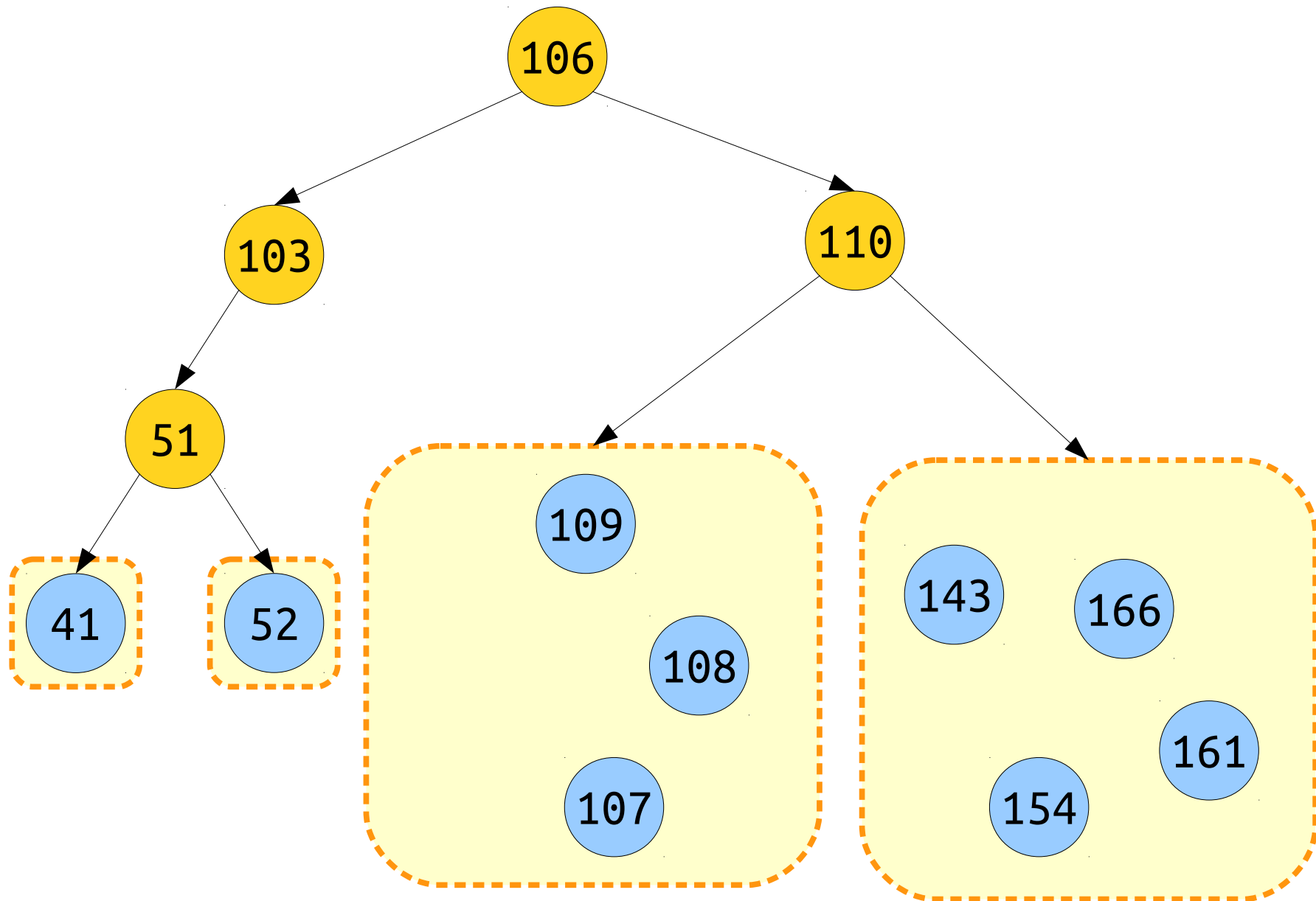


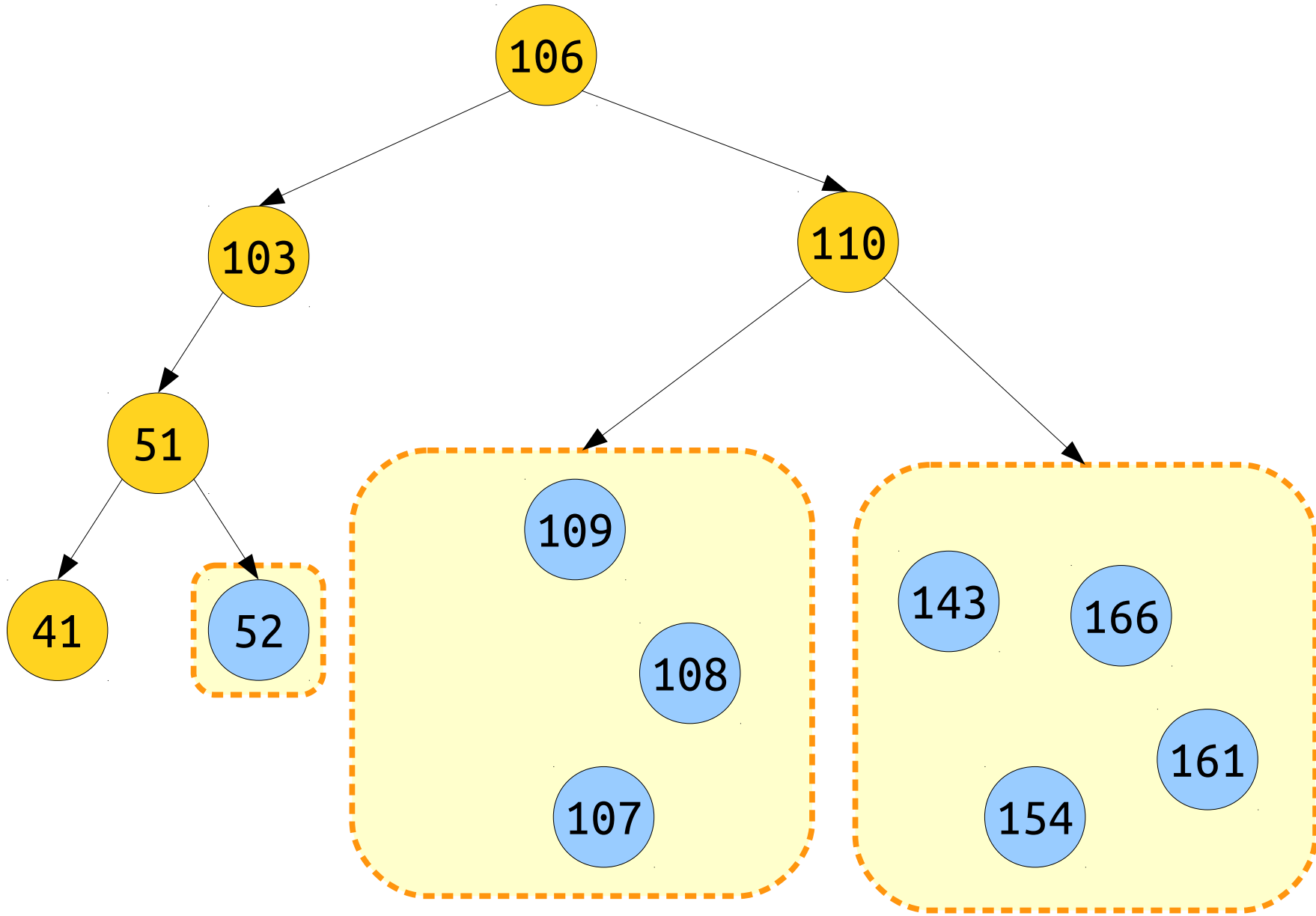


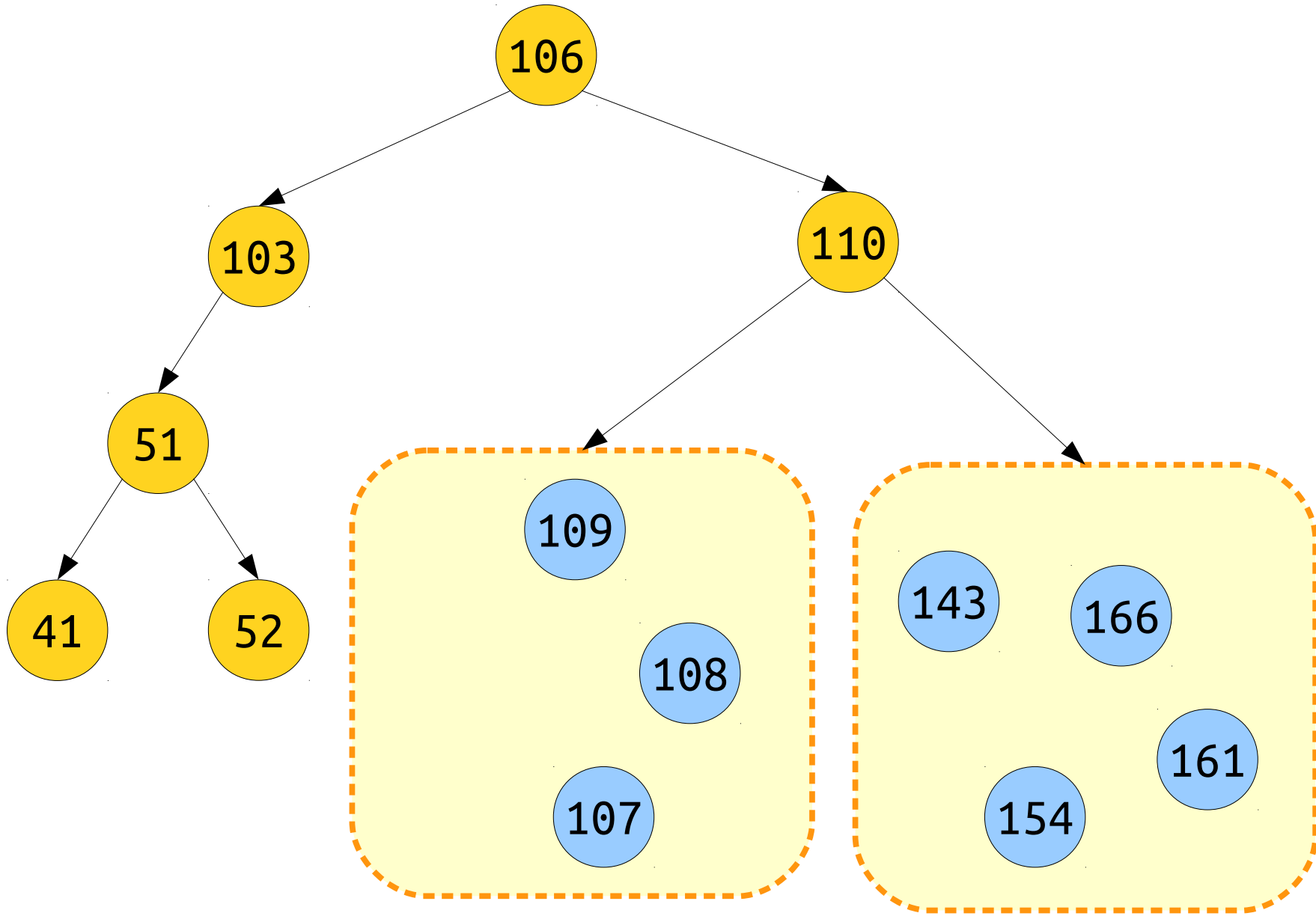




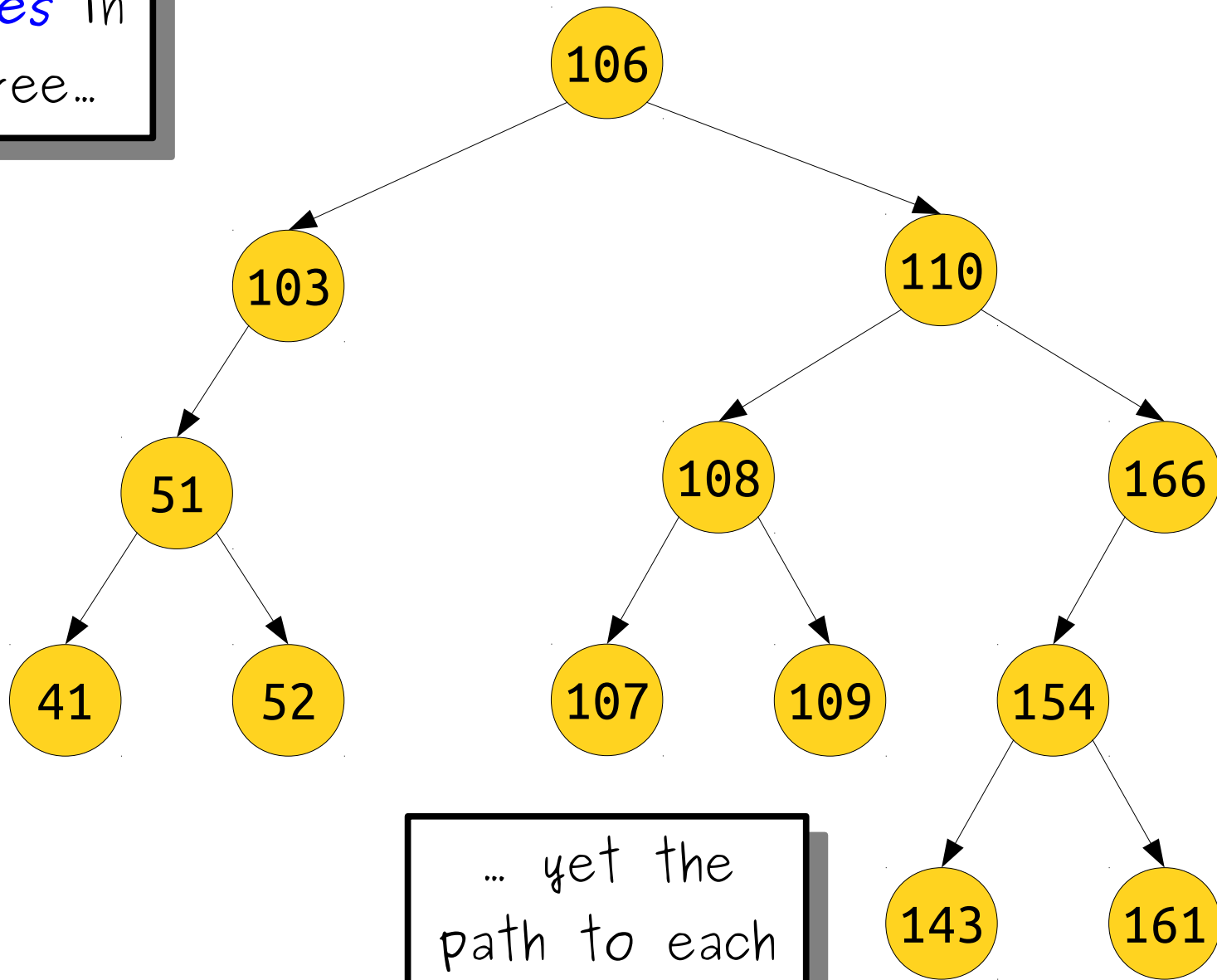




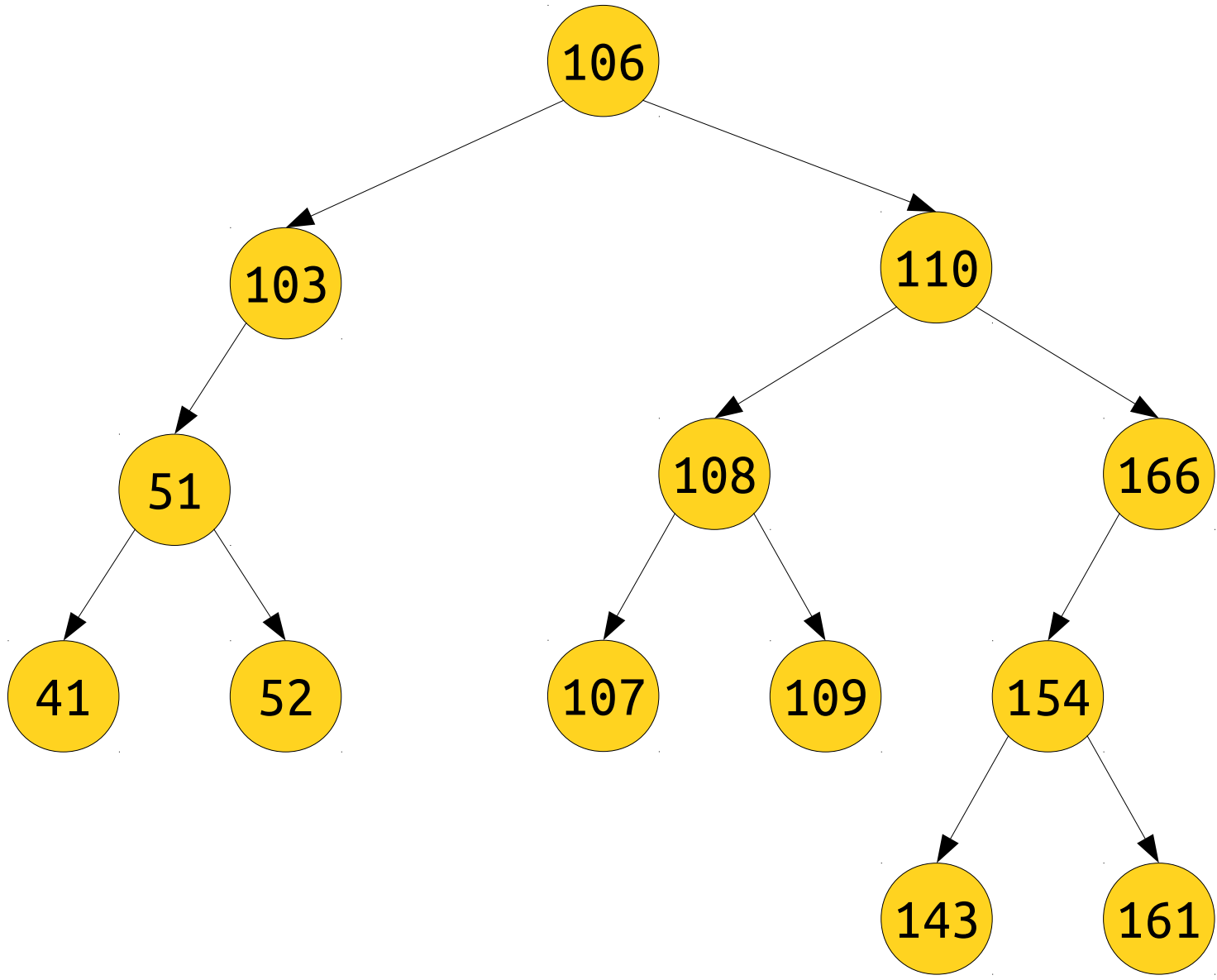


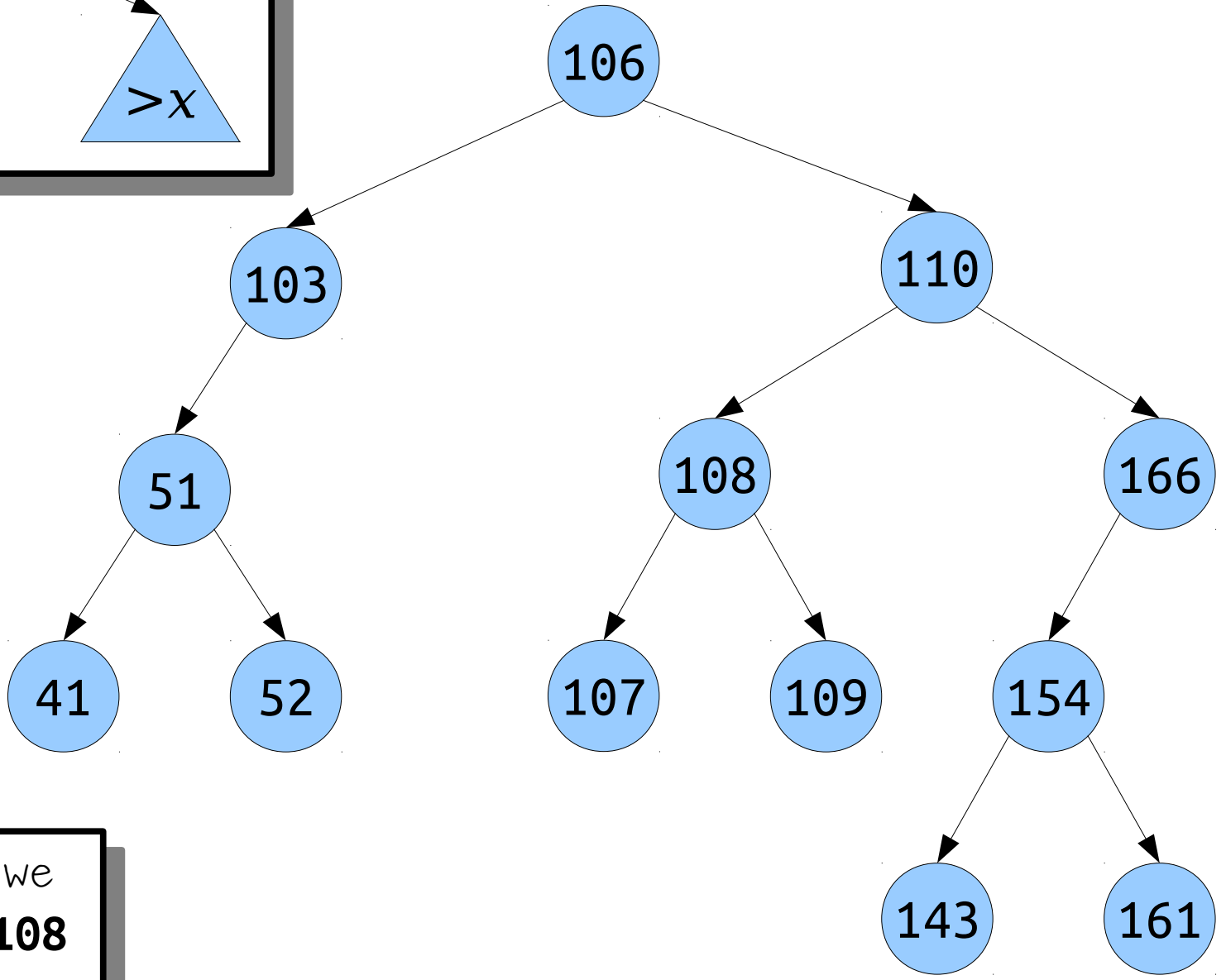
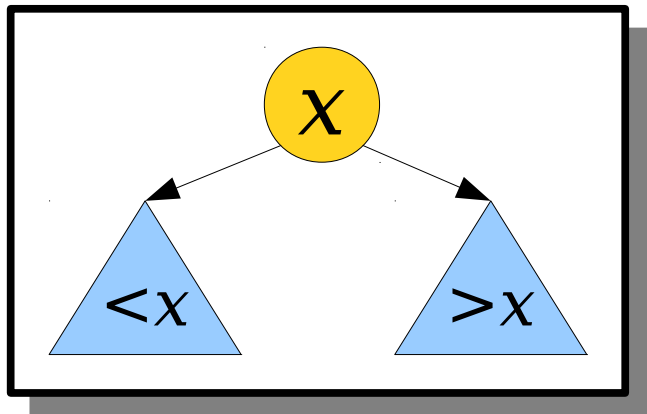


There are
13 *nodes* in
this tree...

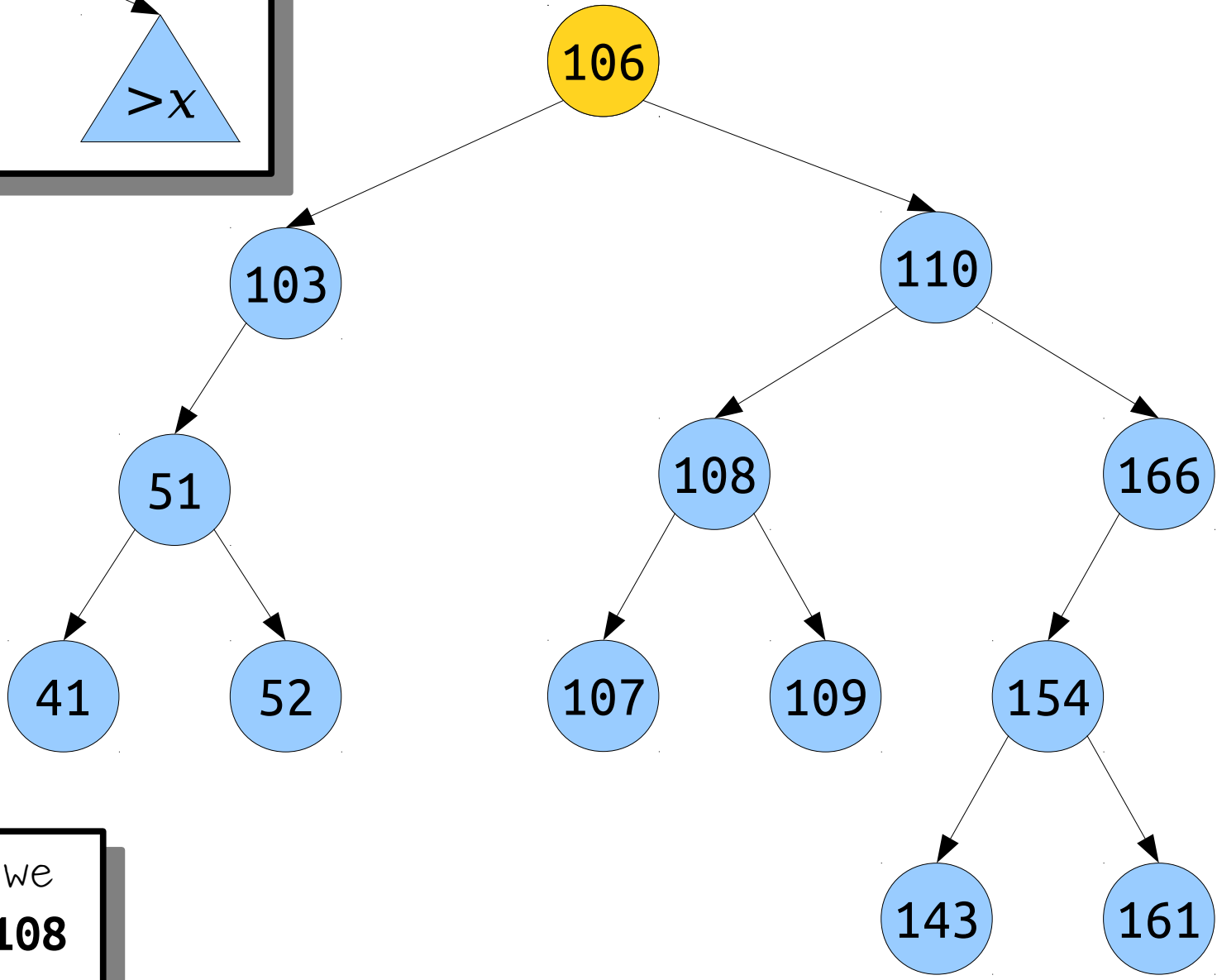
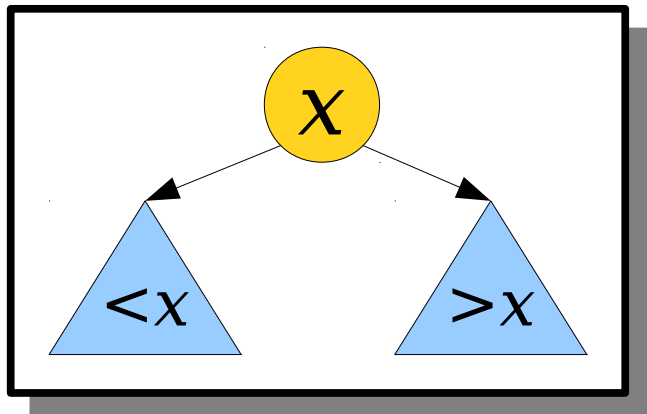


... yet the
path to each
one is short.

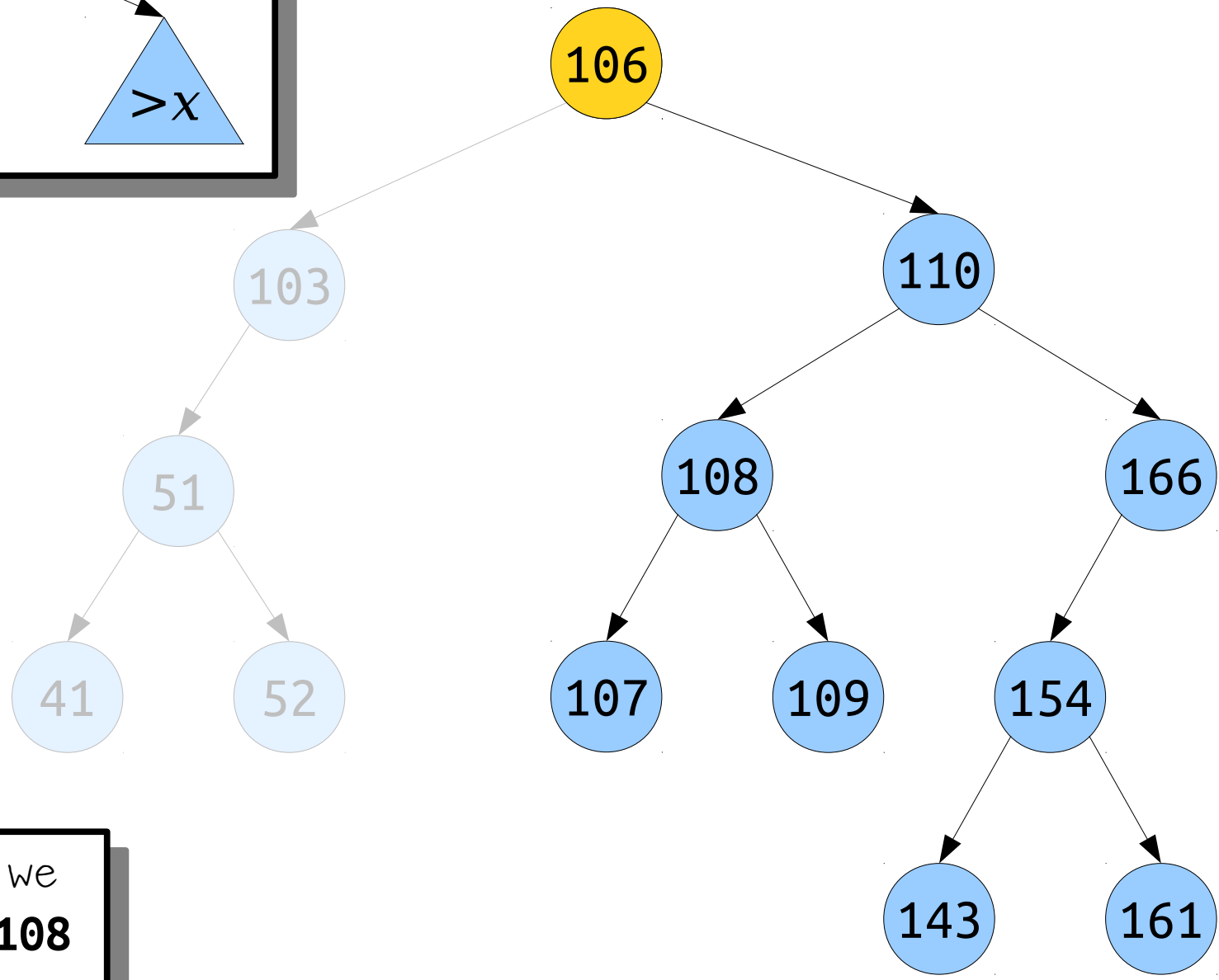
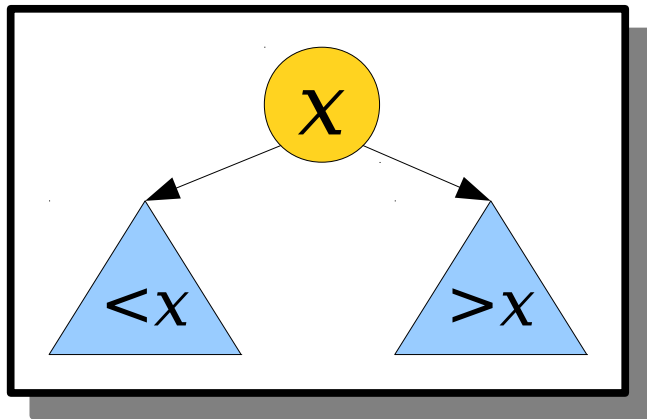




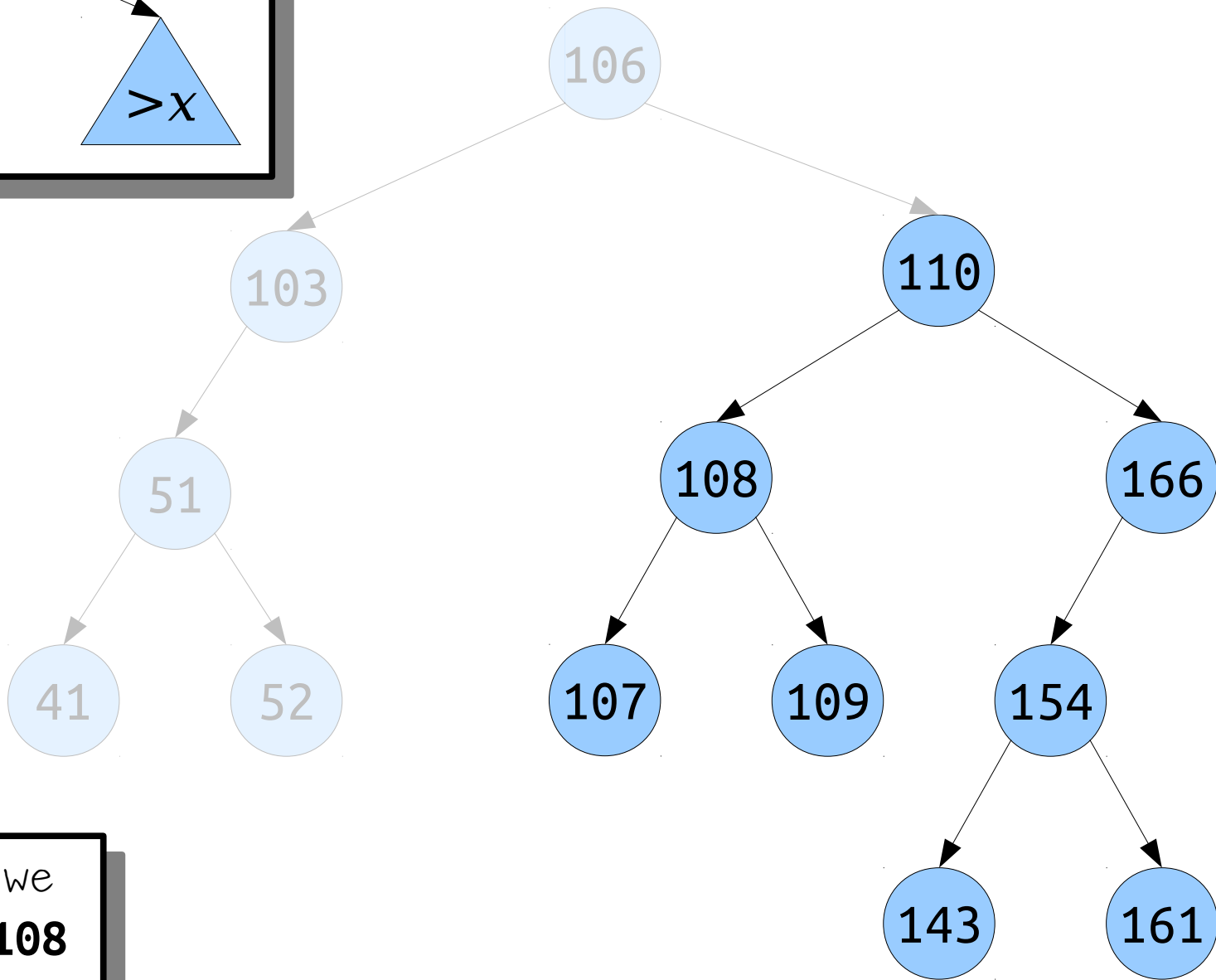
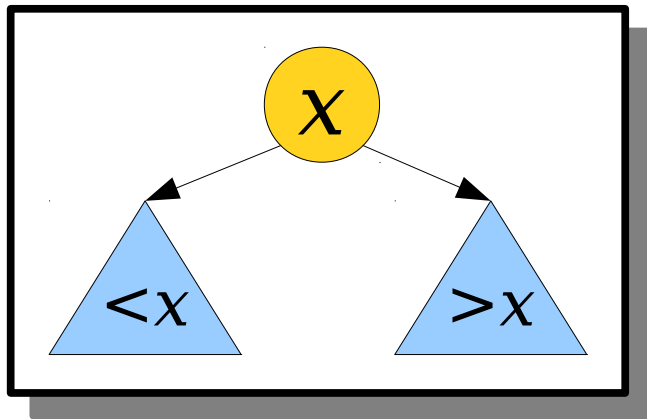
How can we check if **108** is in this tree?



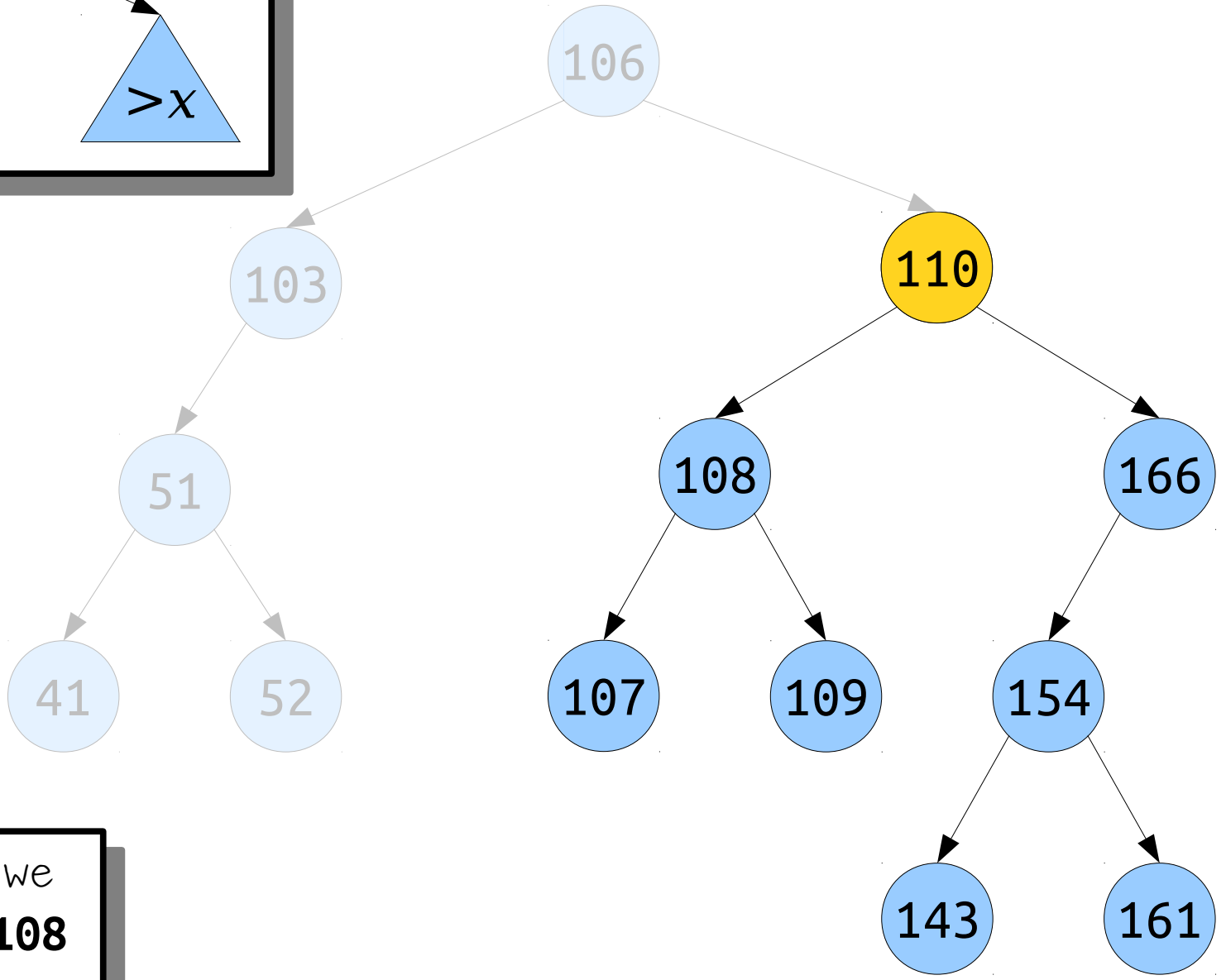
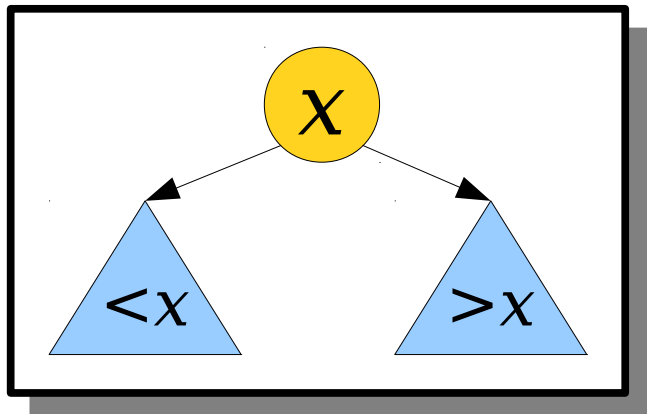
How can we check if **108** is in this tree?



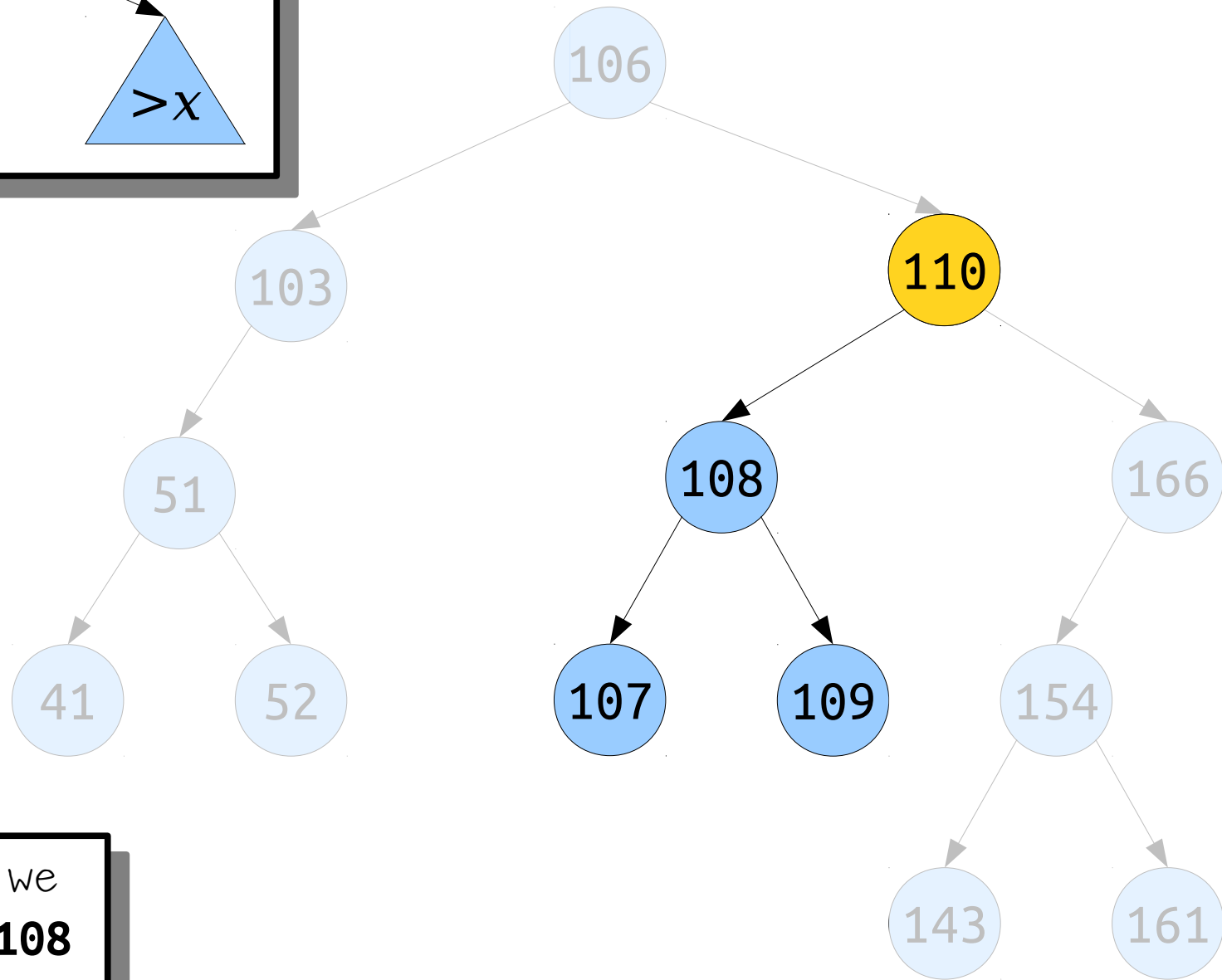
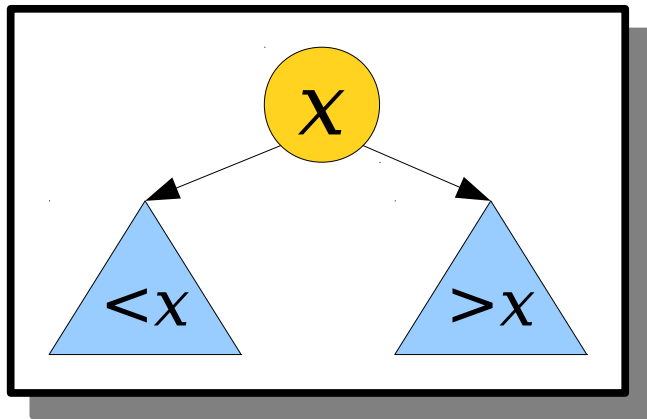
How can we check if **108** is in this tree?



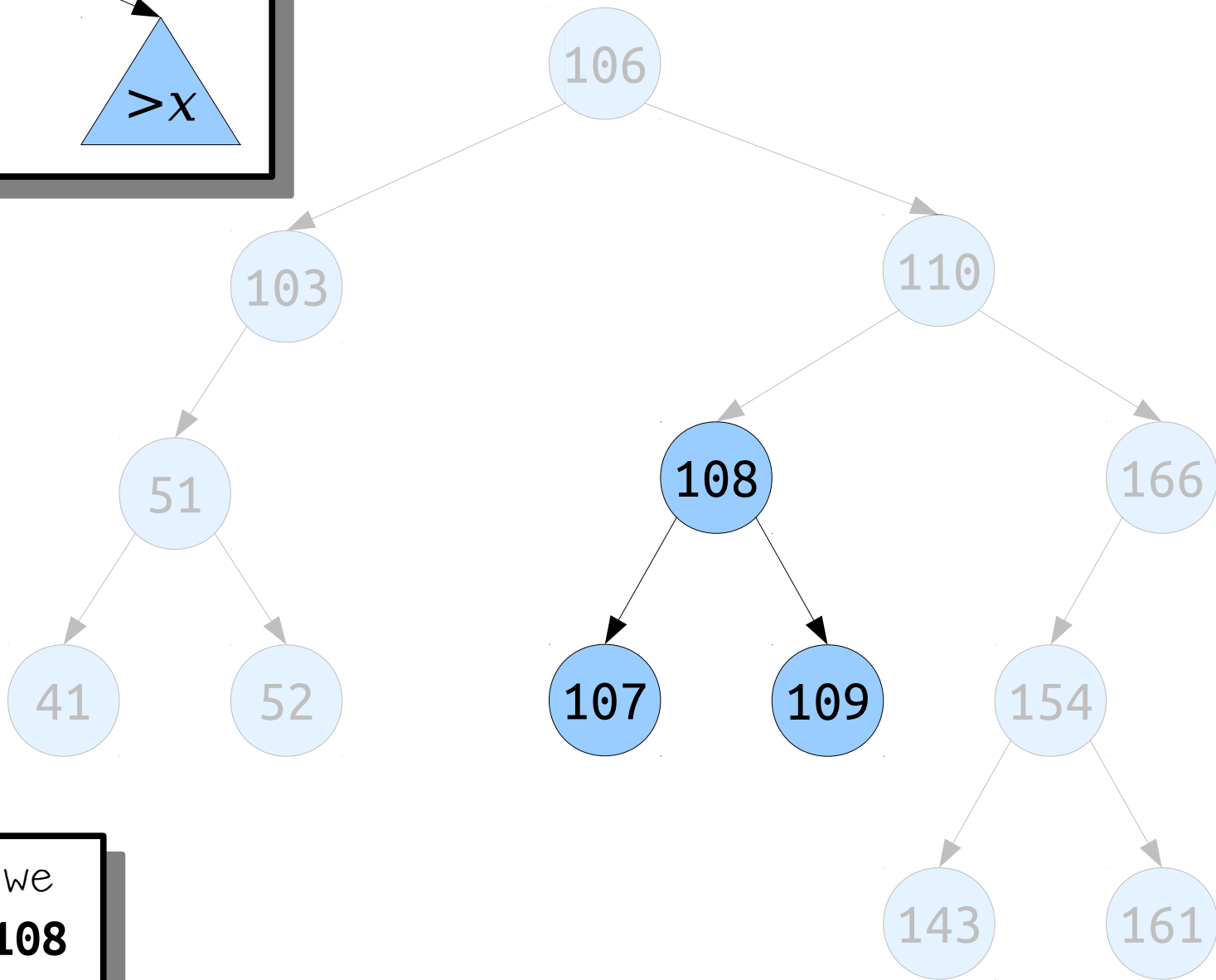
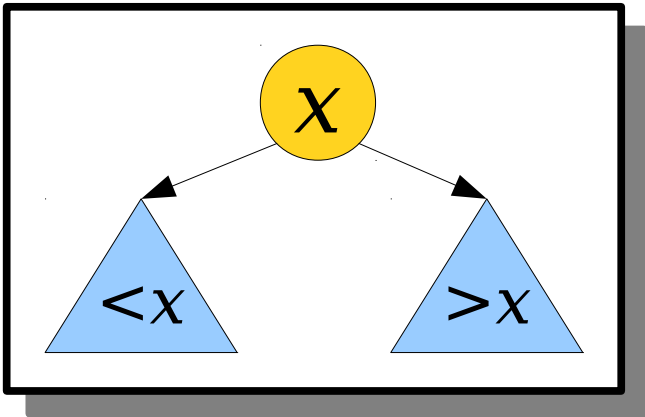
How can we check if **108** is in this tree?



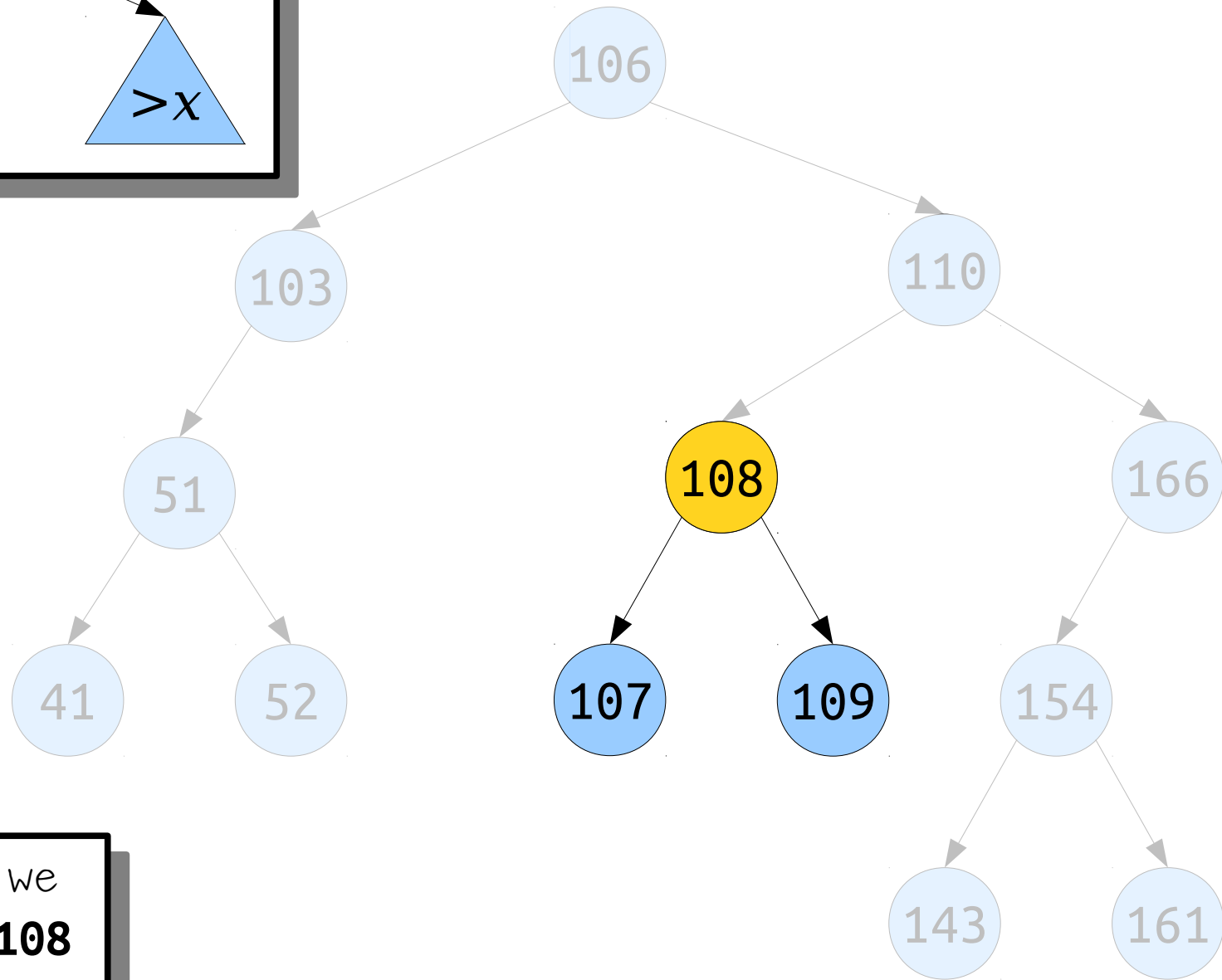
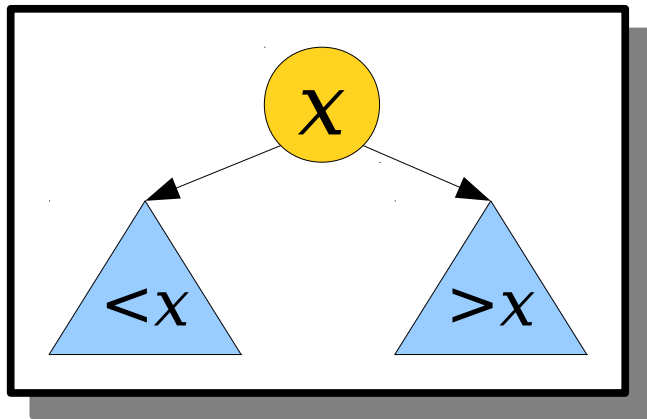
How can we check if **108** is in this tree?



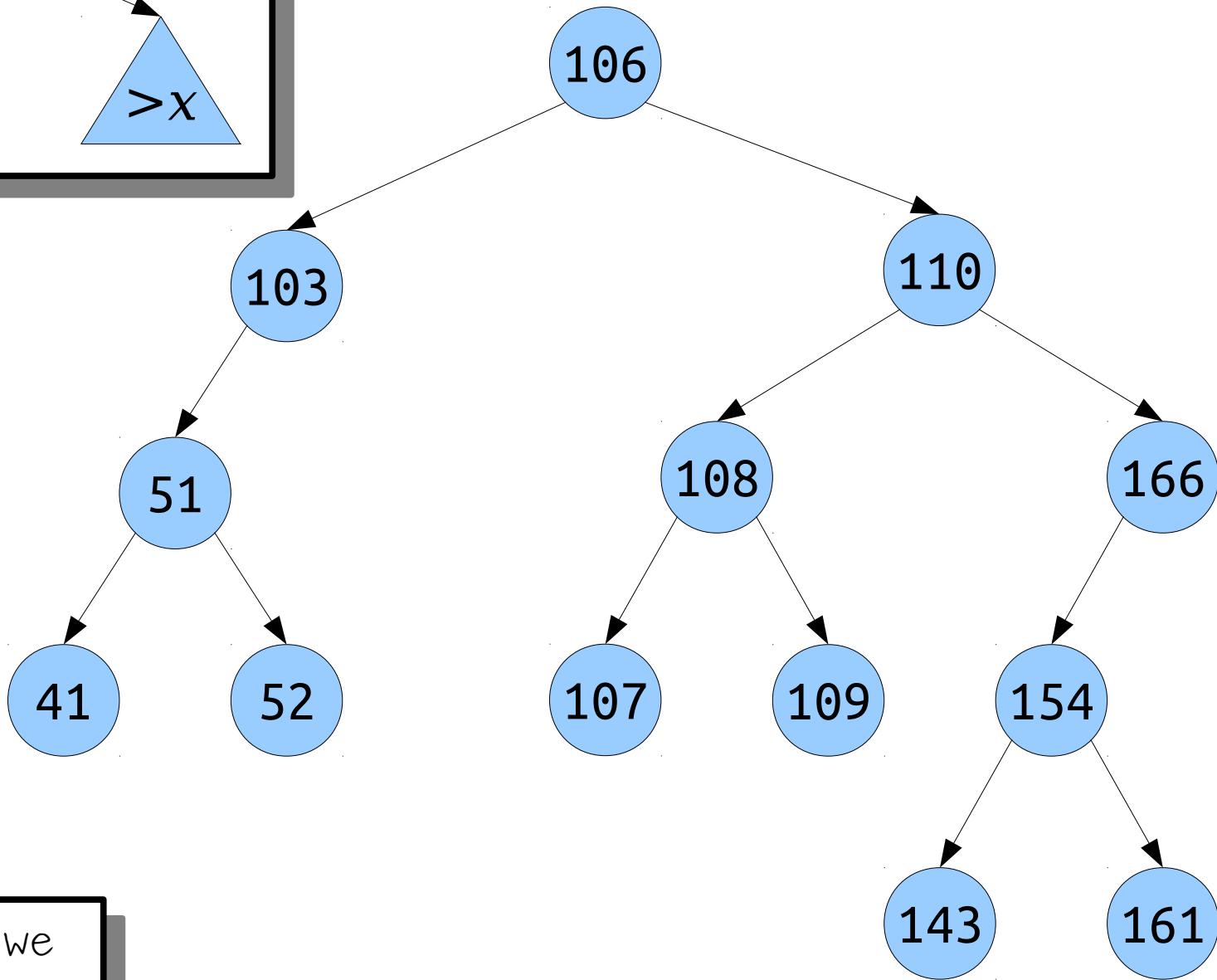
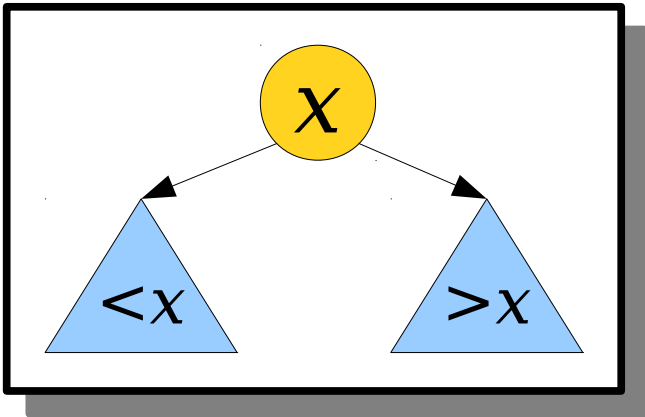
How can we check if **108** is in this tree?



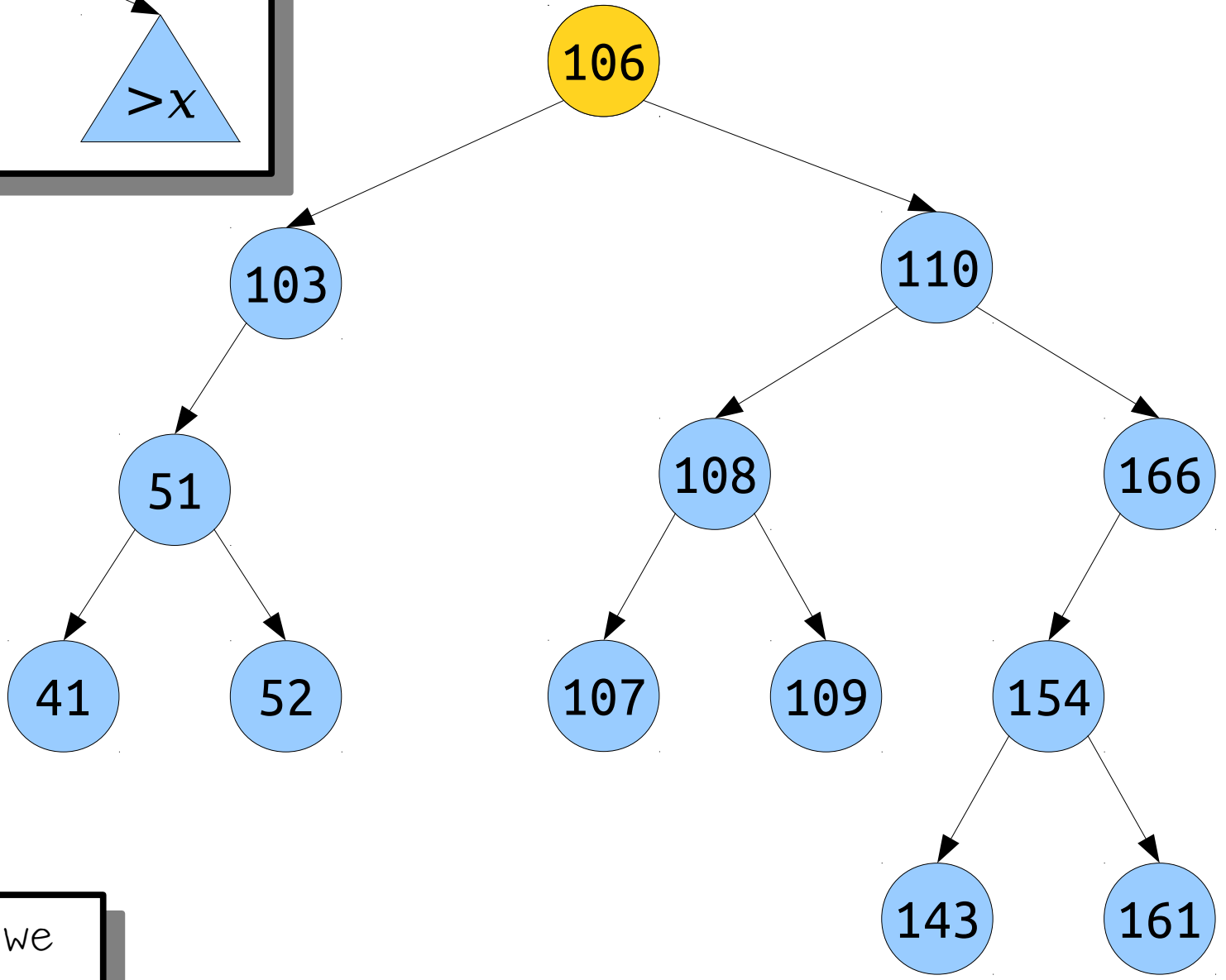
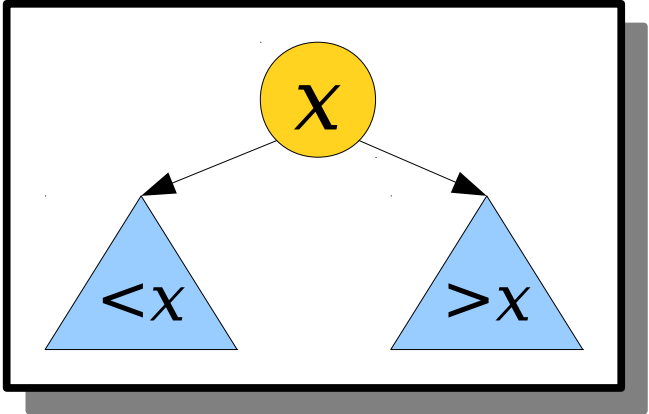
How can we check if **108** is in this tree?



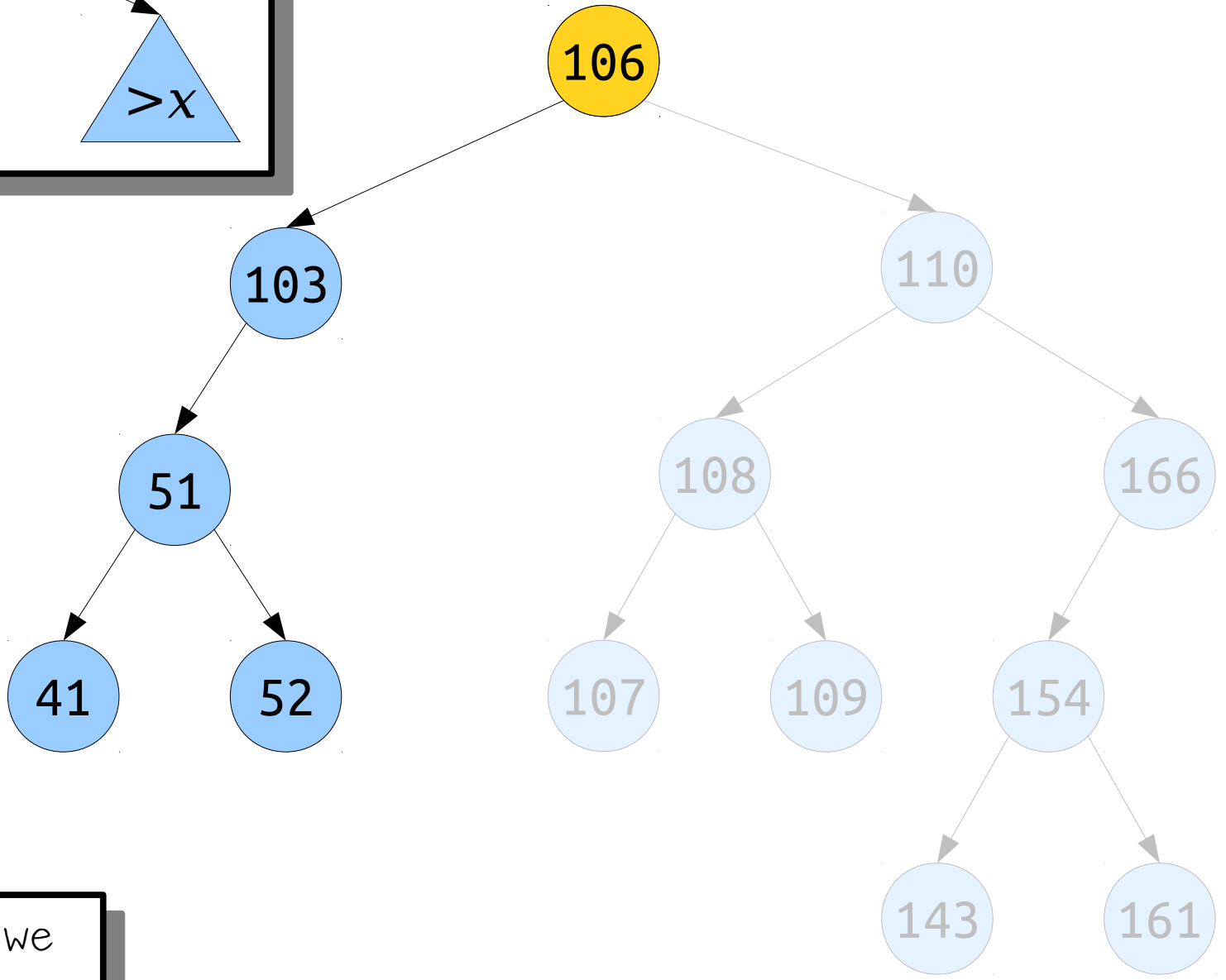
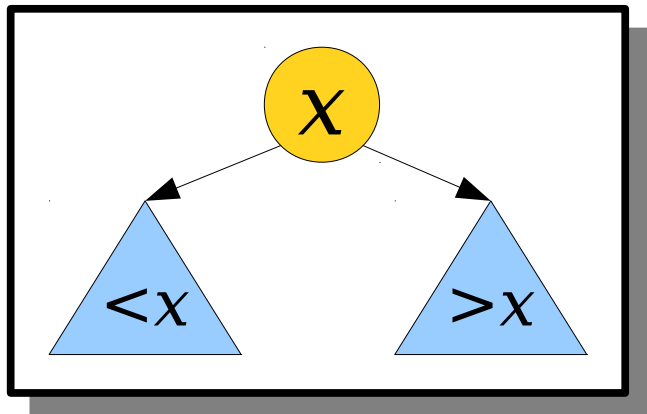
How can we check if **108** is in this tree?



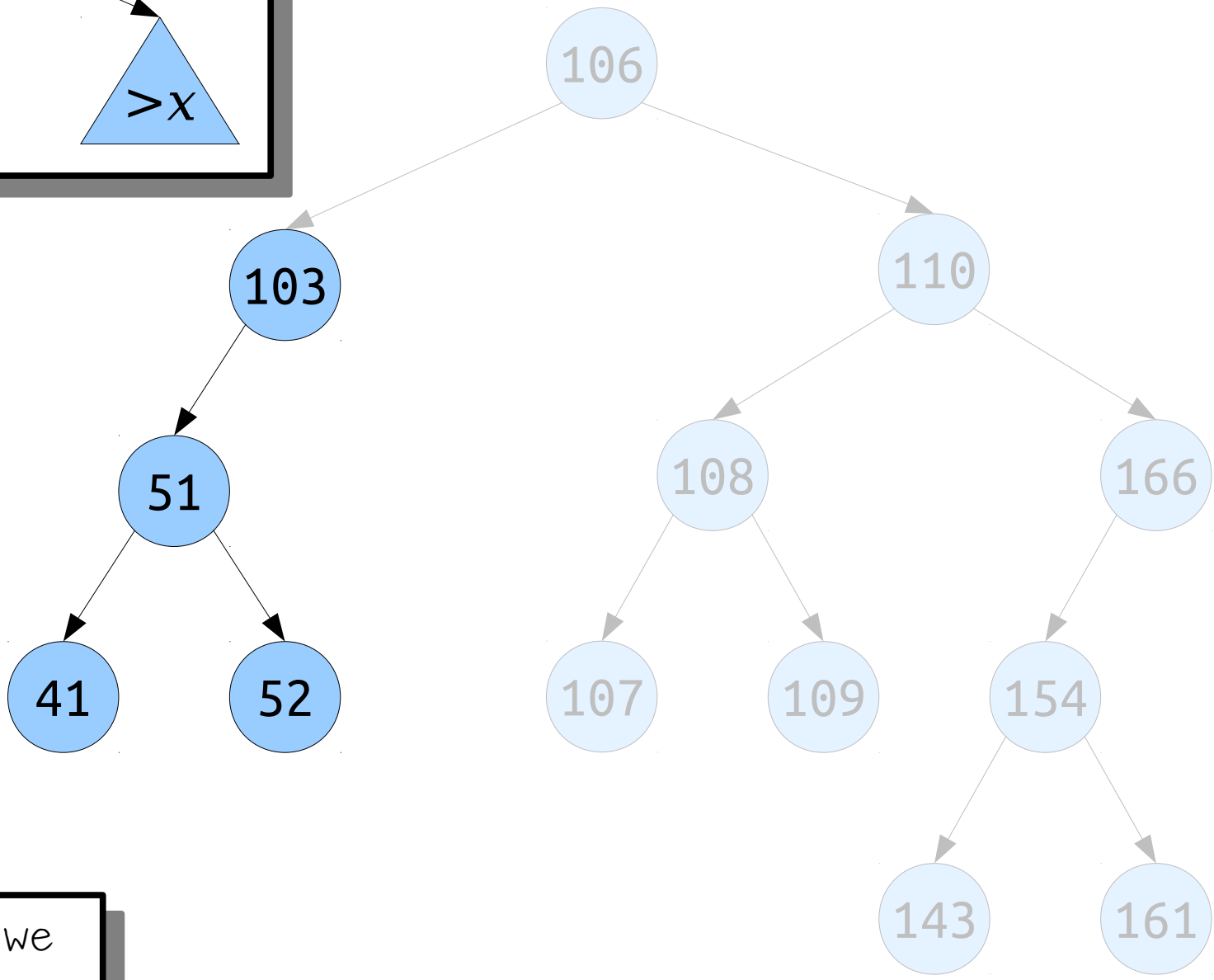
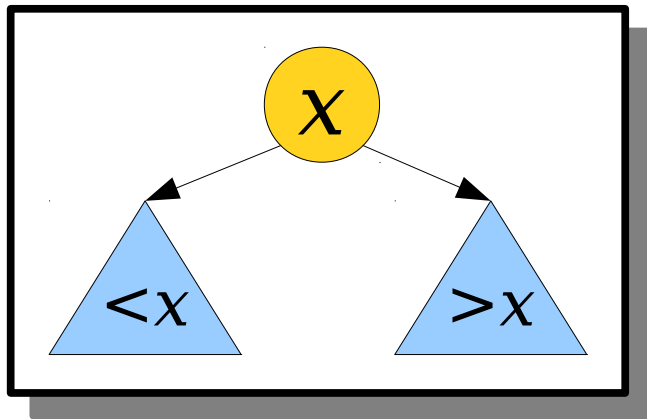
How can we check if **83** is in this tree?



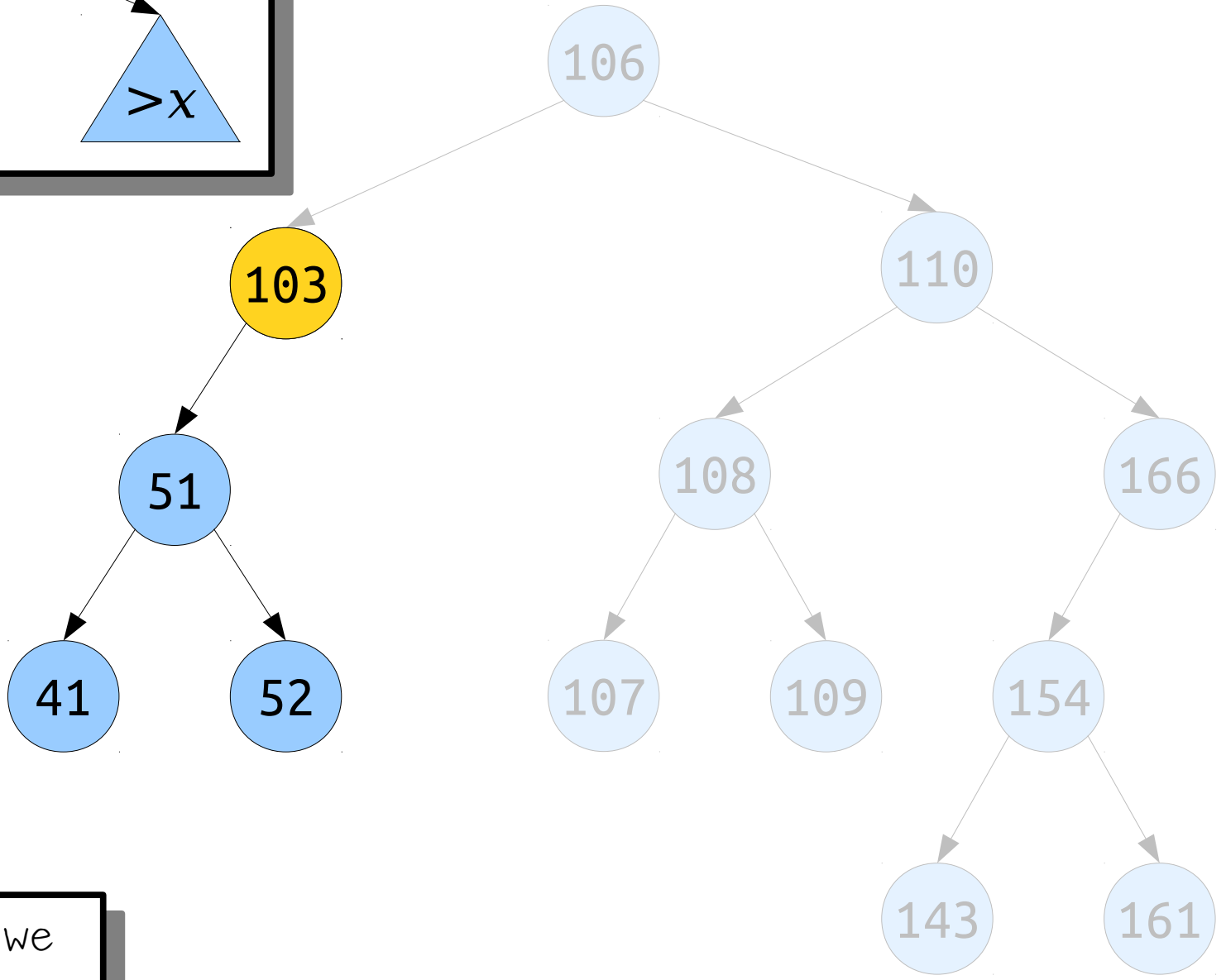
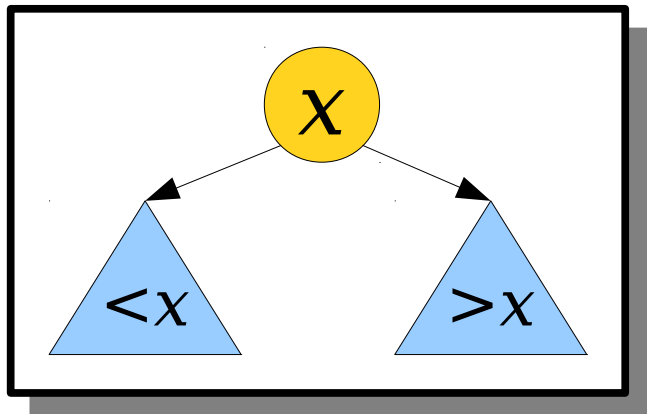
How can we check if **83** is in this tree?



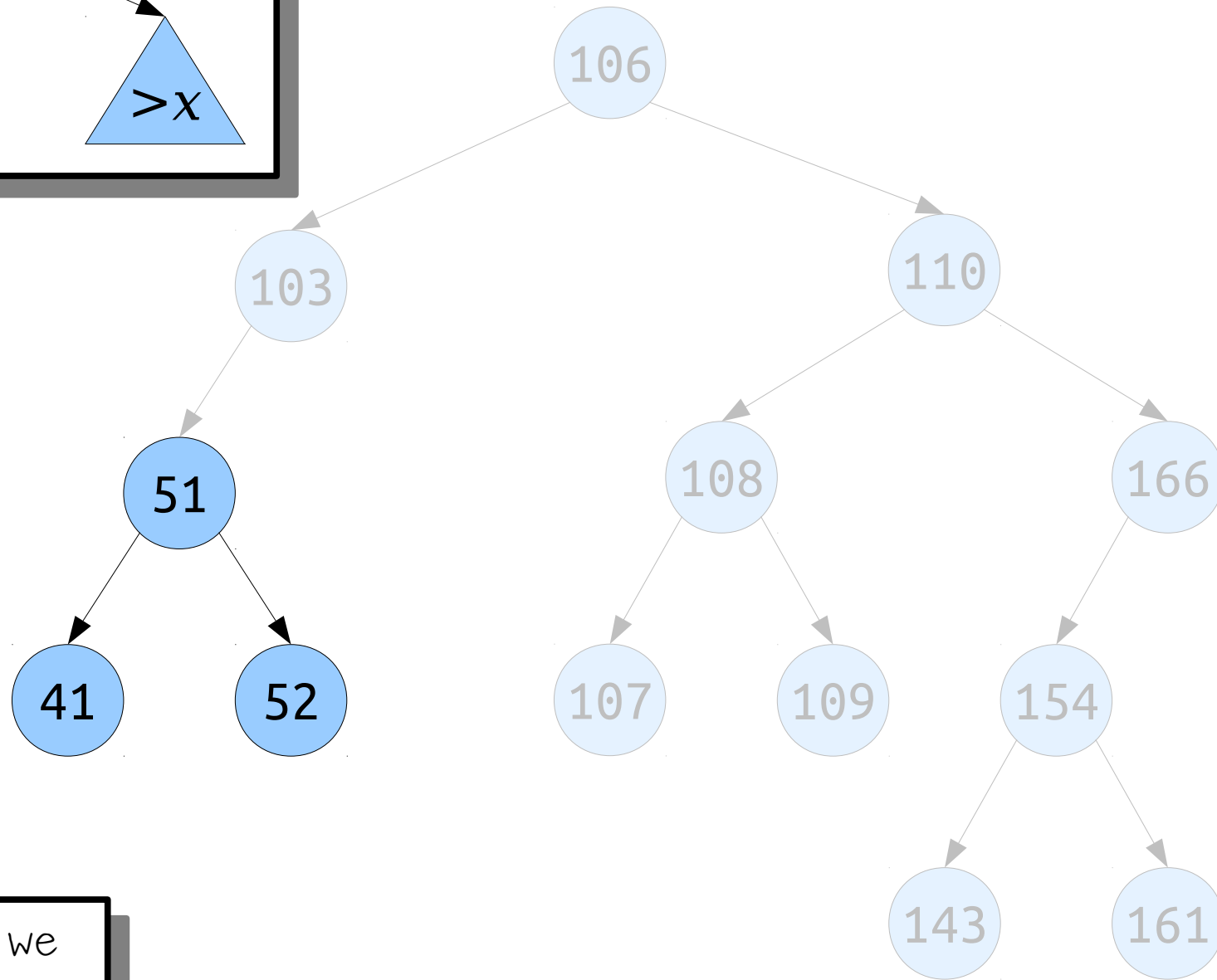
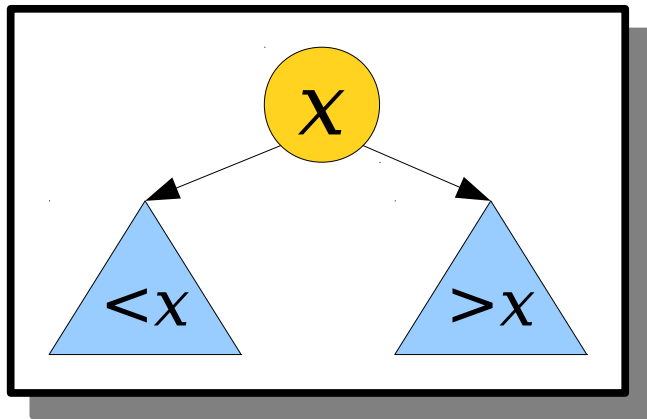
How can we check if **83** is in this tree?



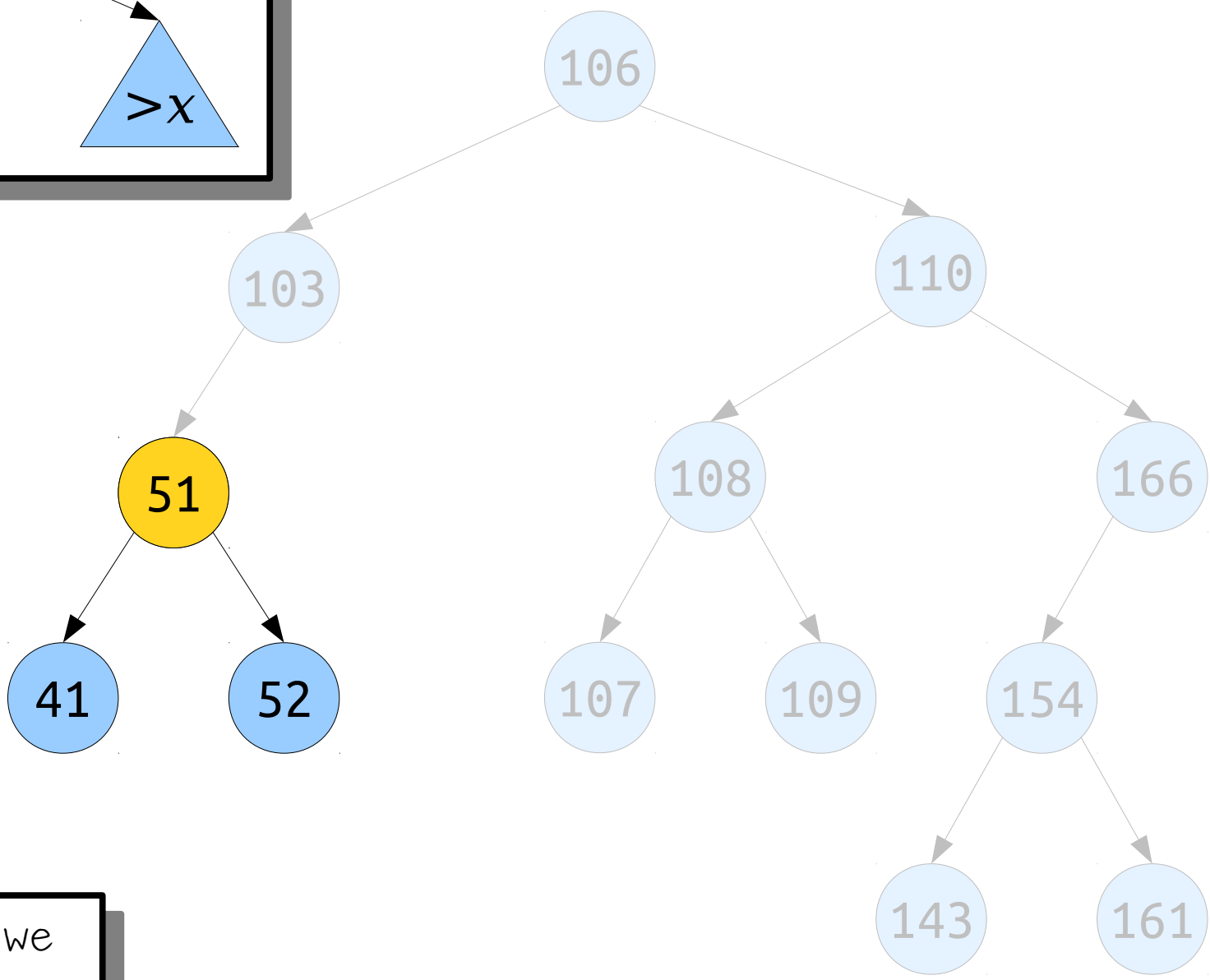
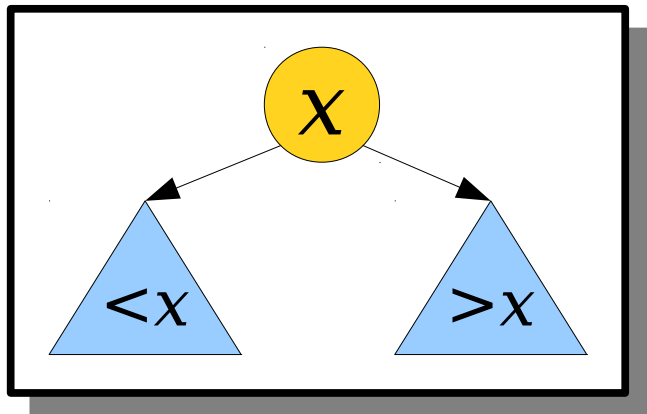
How can we check if **83** is in this tree?



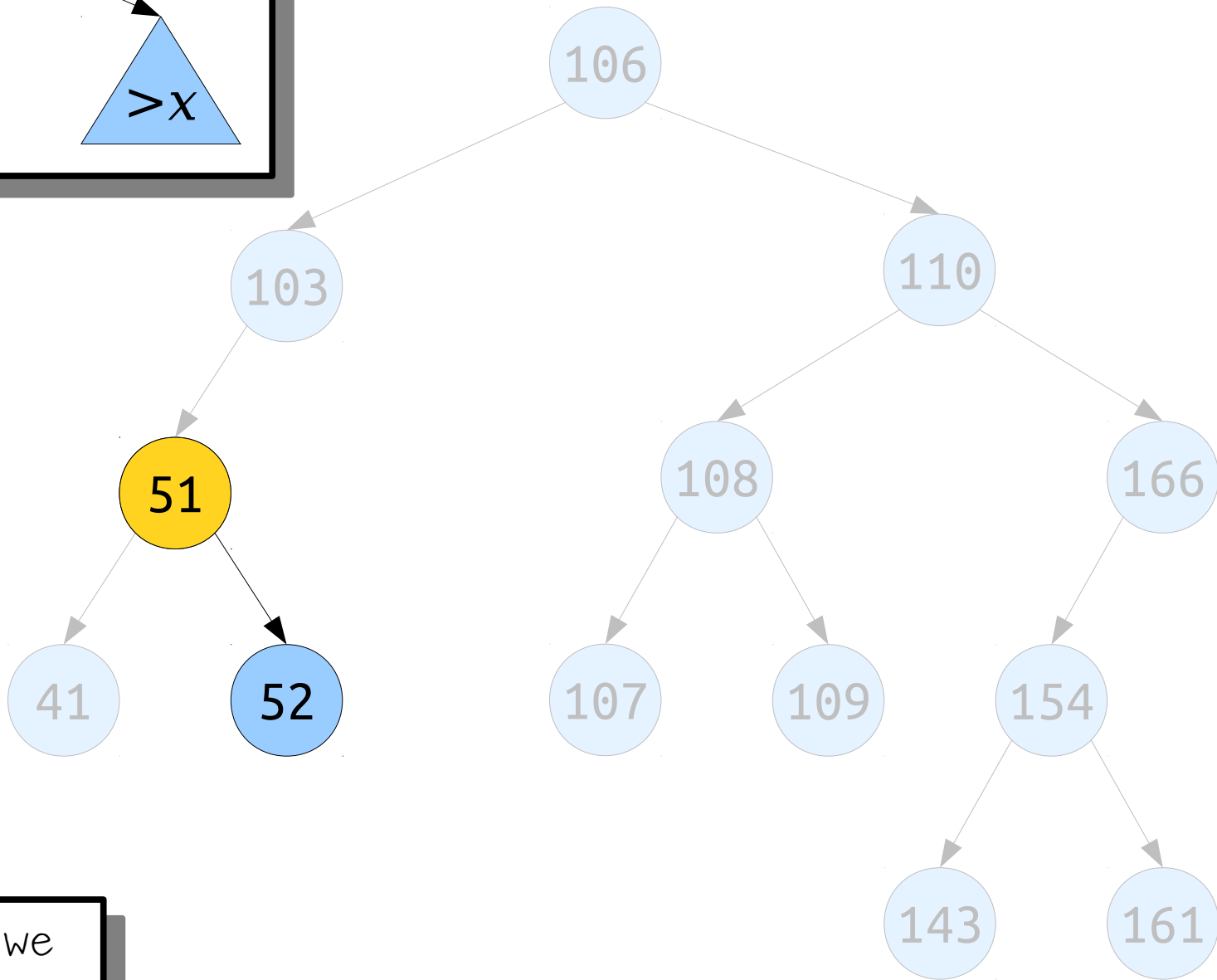
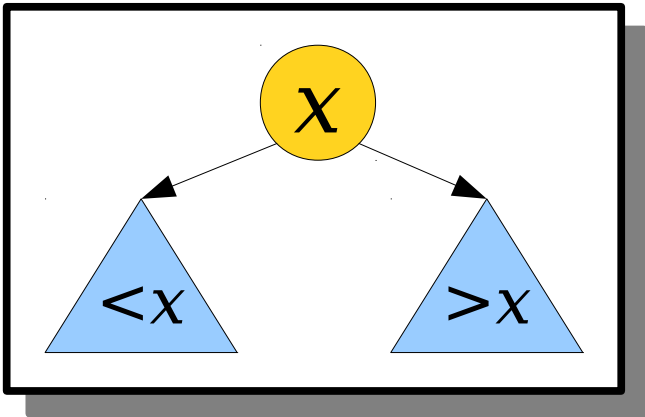
How can we check if **83** is in this tree?



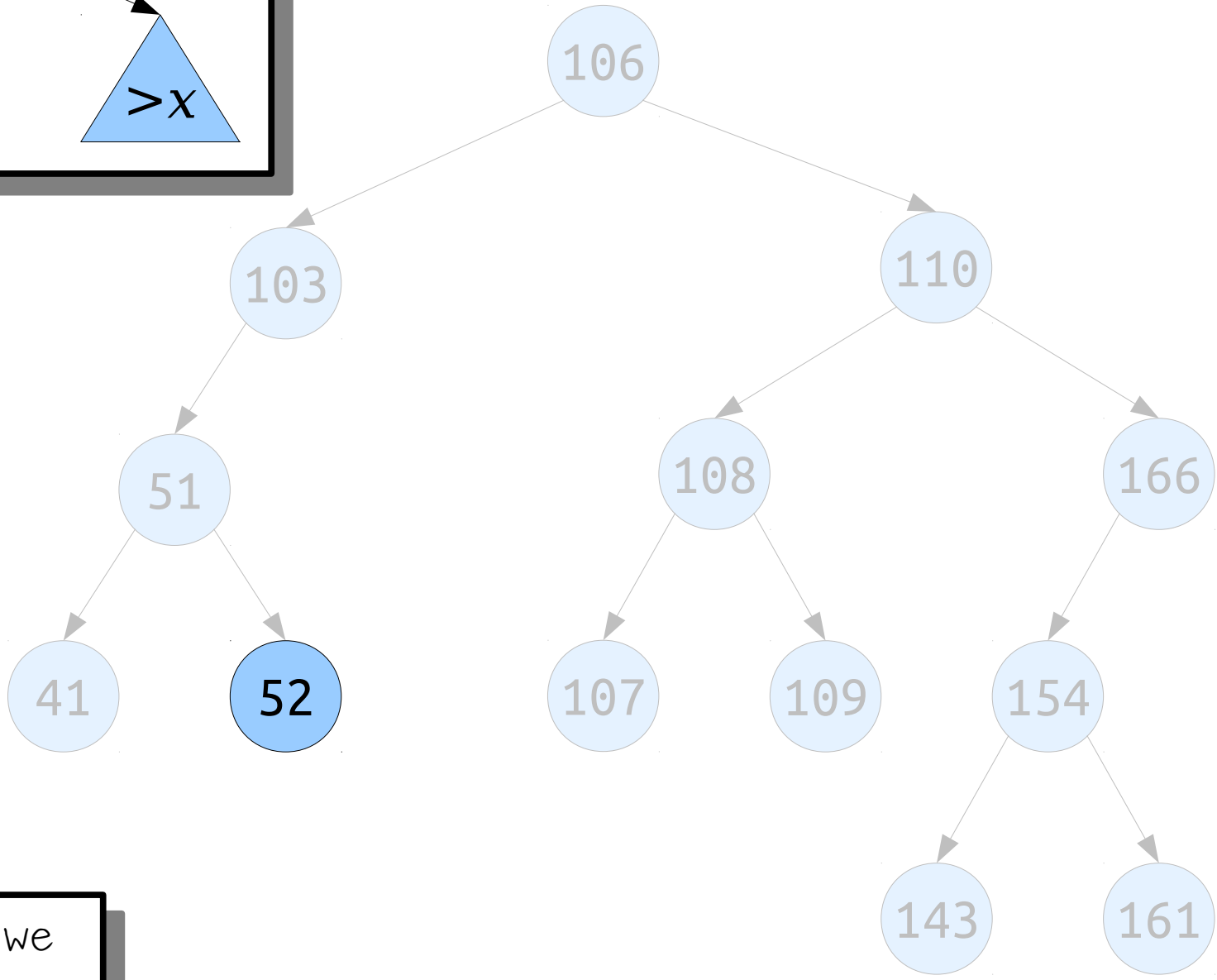
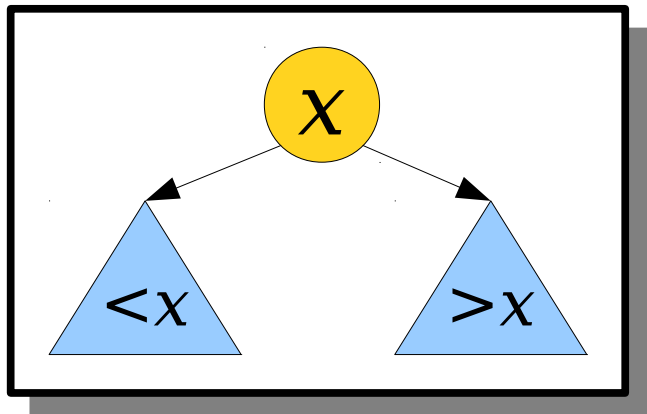
How can we check if **83** is in this tree?



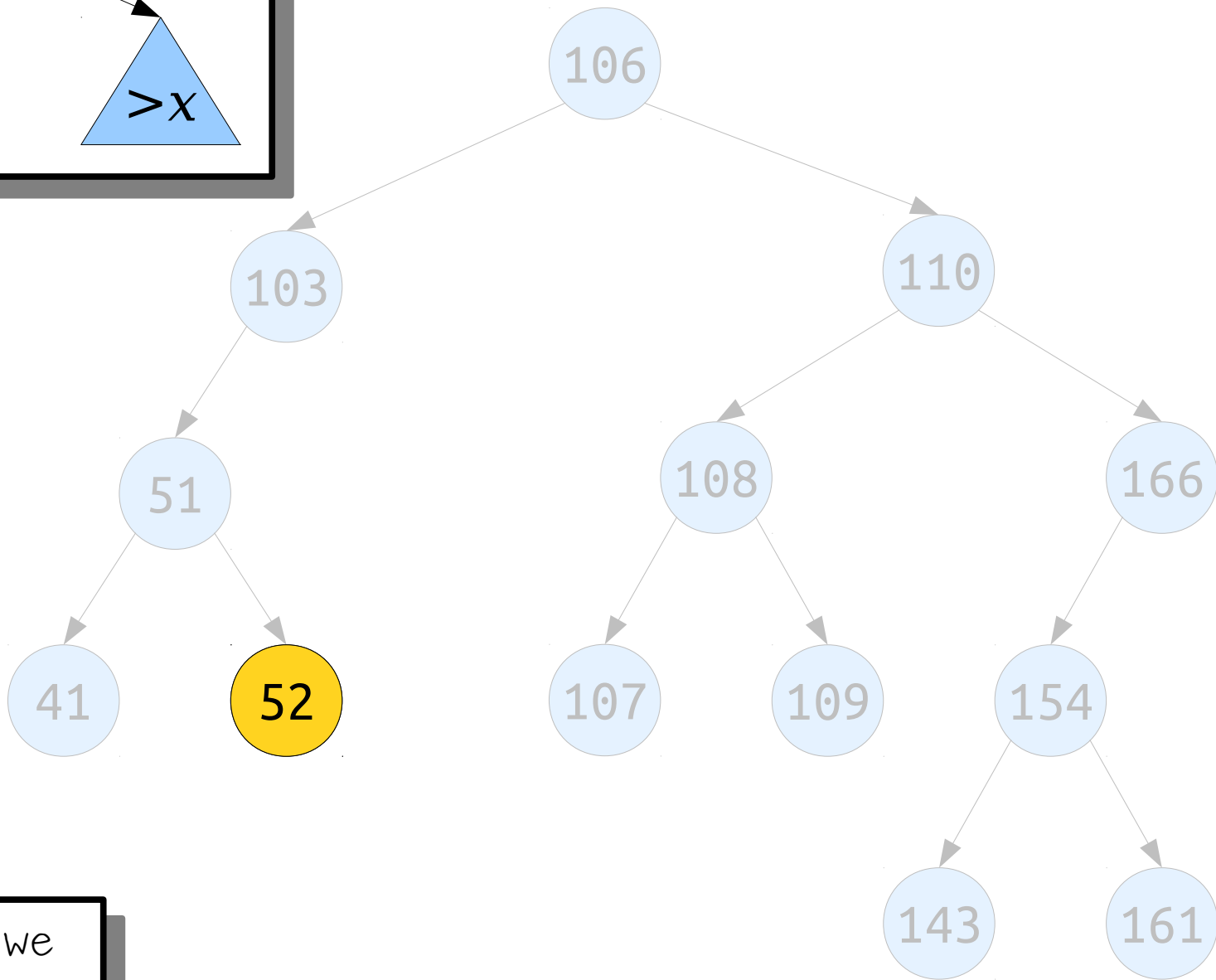
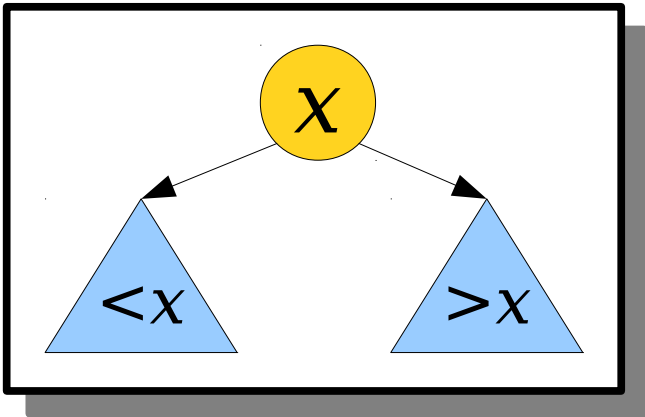
How can we check if **83** is in this tree?



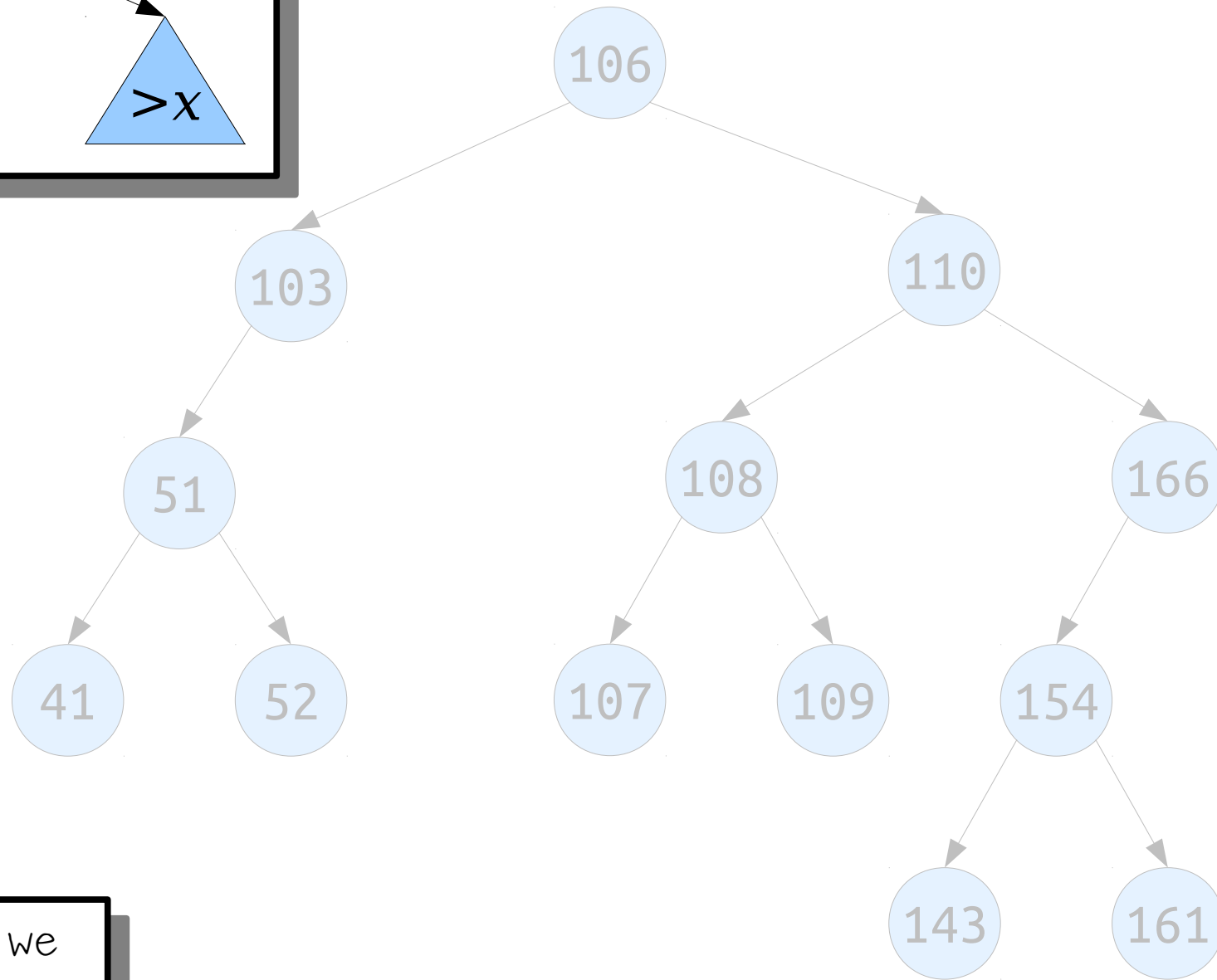
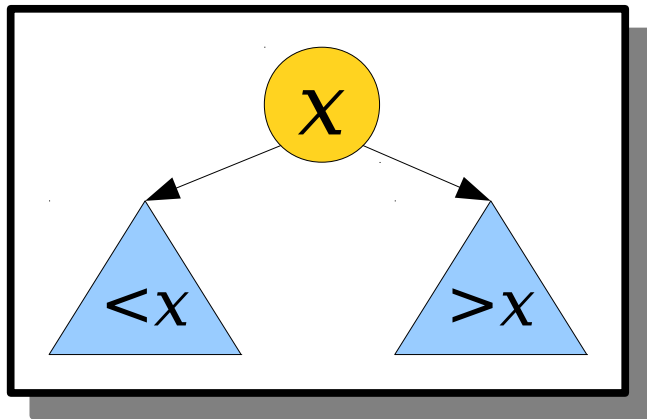
How can we check if **83** is in this tree?



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



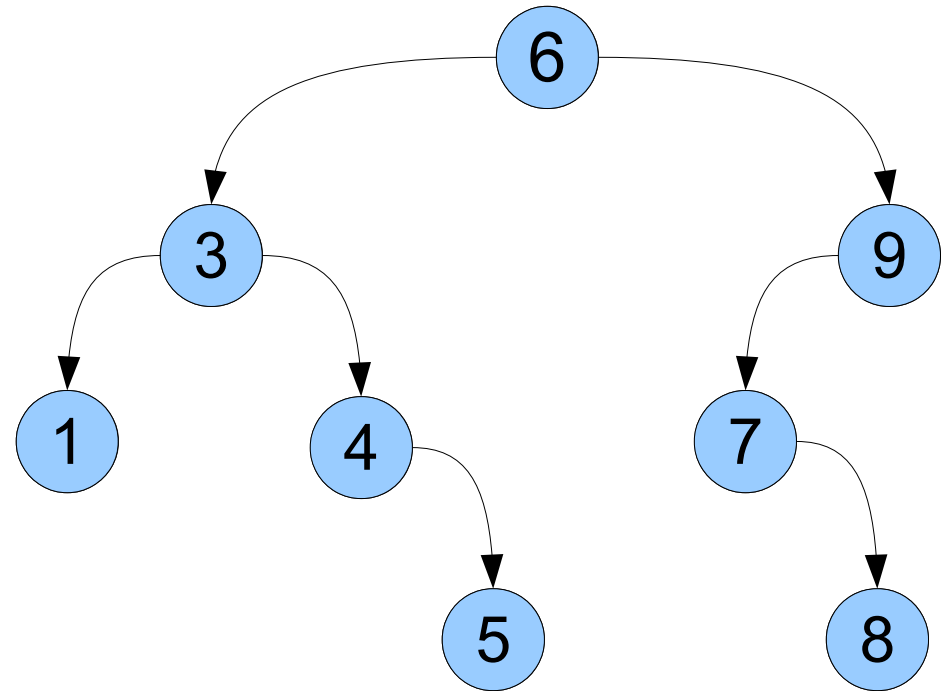
How can we check if **83** is in this tree?



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

Binary Search Trees

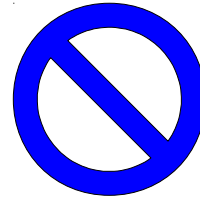
- The data structure we have just seen is called a **binary search tree** (or **BST**).
- The tree consists of a number of **nodes**, each of which stores a value and has zero, one, or two **children**.
- All values in a node's left subtree are **smaller** than the node's value, and all values in a node's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



A Binary Search Tree Is Either...

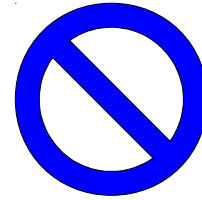
an empty tree,
represented by

`nullptr`

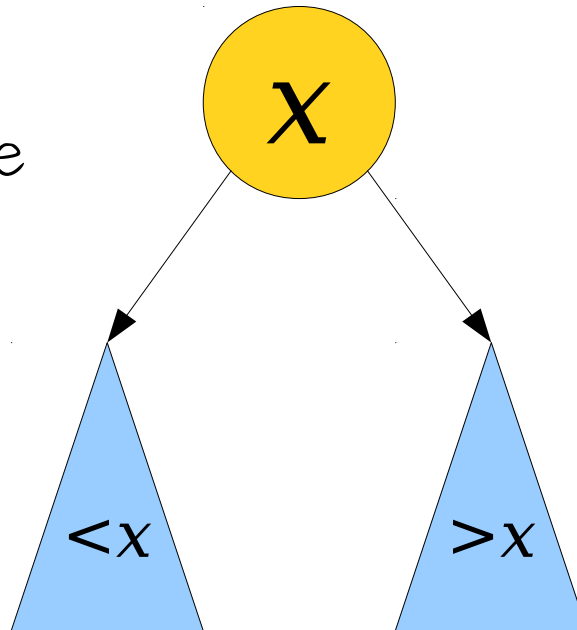


A Binary Search Tree Is Either...

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!

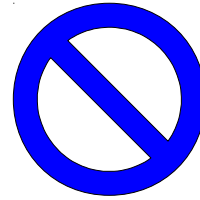
Searching Trees



A Binary Search Tree Is Either...

an empty tree,
represented by

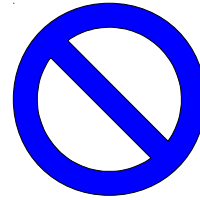
`nullptr`



A Binary Search Tree Is Either...

an empty tree,
represented by

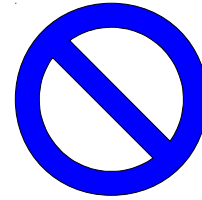
`nullptr`



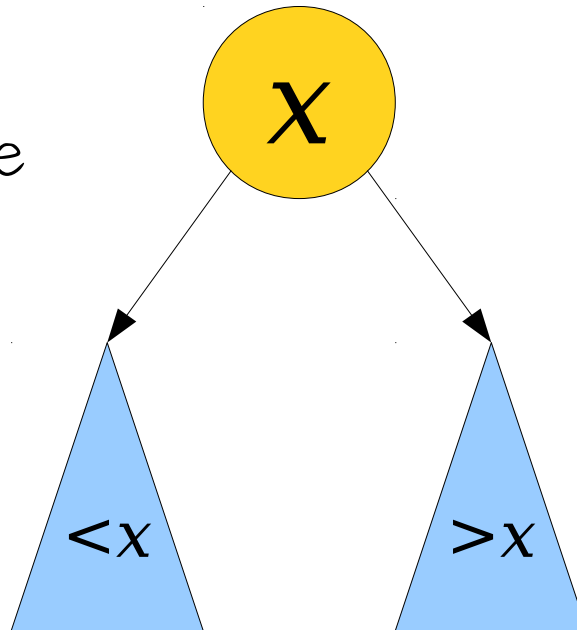
If you're looking for something in an empty BST, it's not there! Sorry.

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.

Good exercise:

Rewrite this function iteratively!

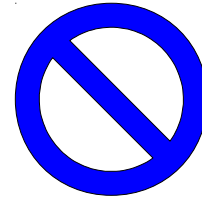
Walking Trees



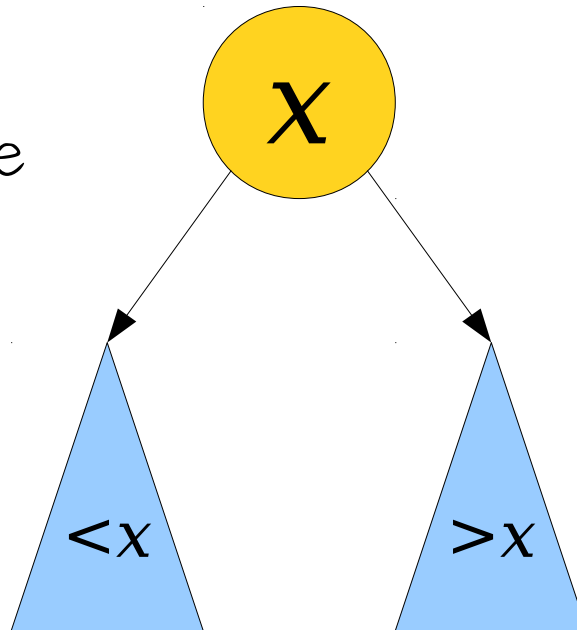
Print all the values in a BST,
in sorted order.

A Binary Search Tree Is Either...

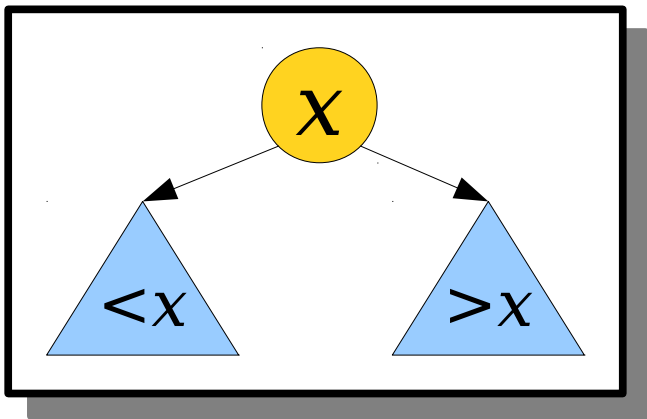
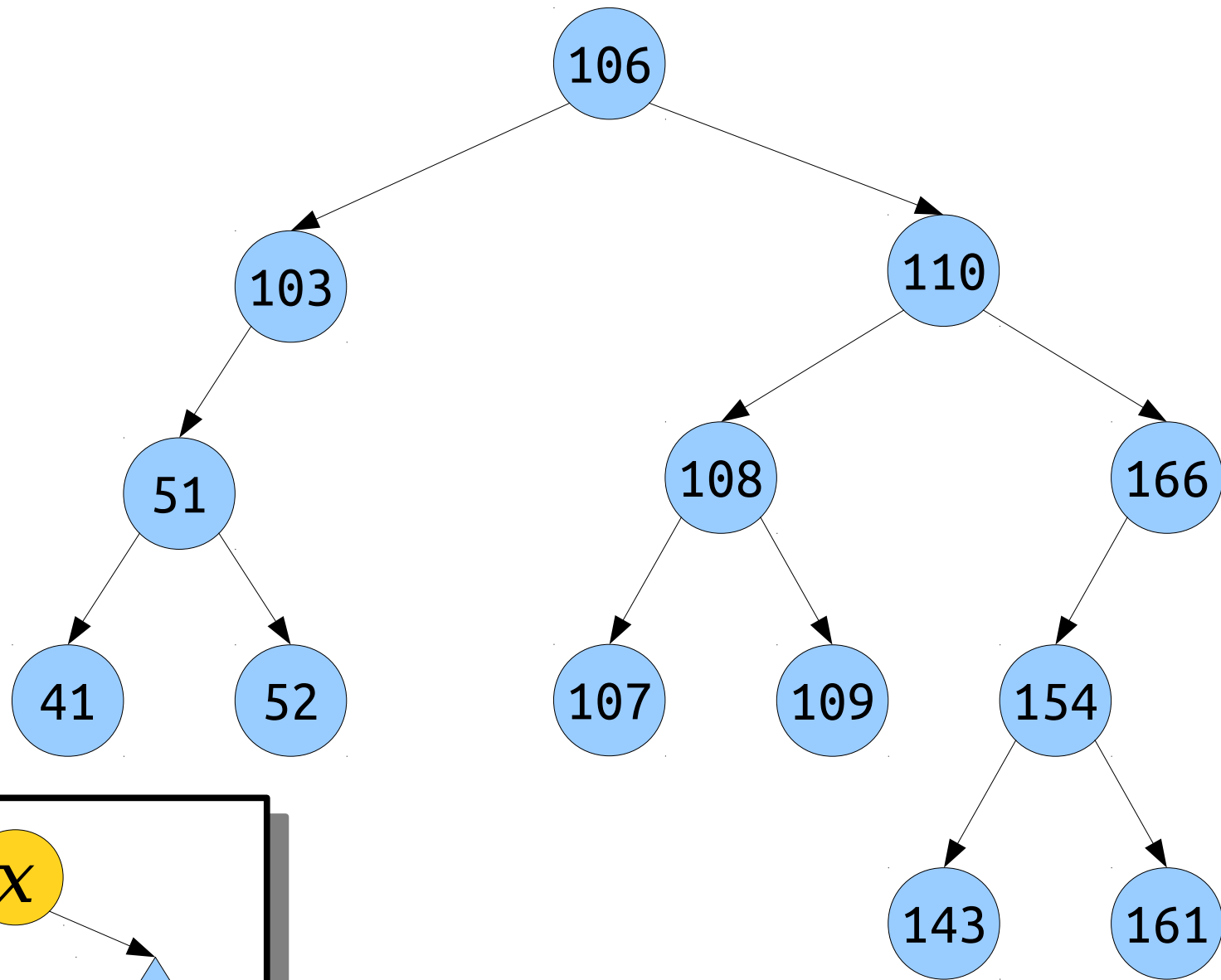
an empty tree,
represented by
nullptr, or...



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



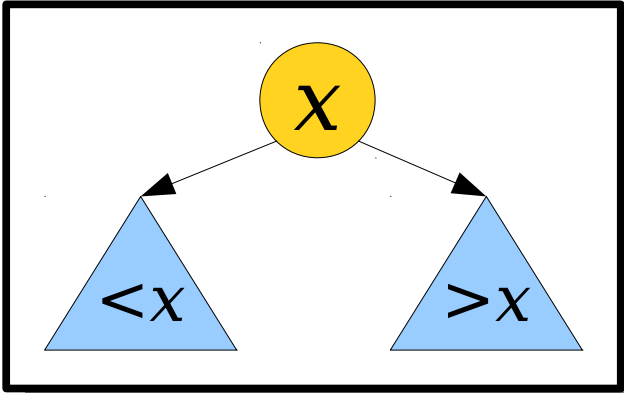
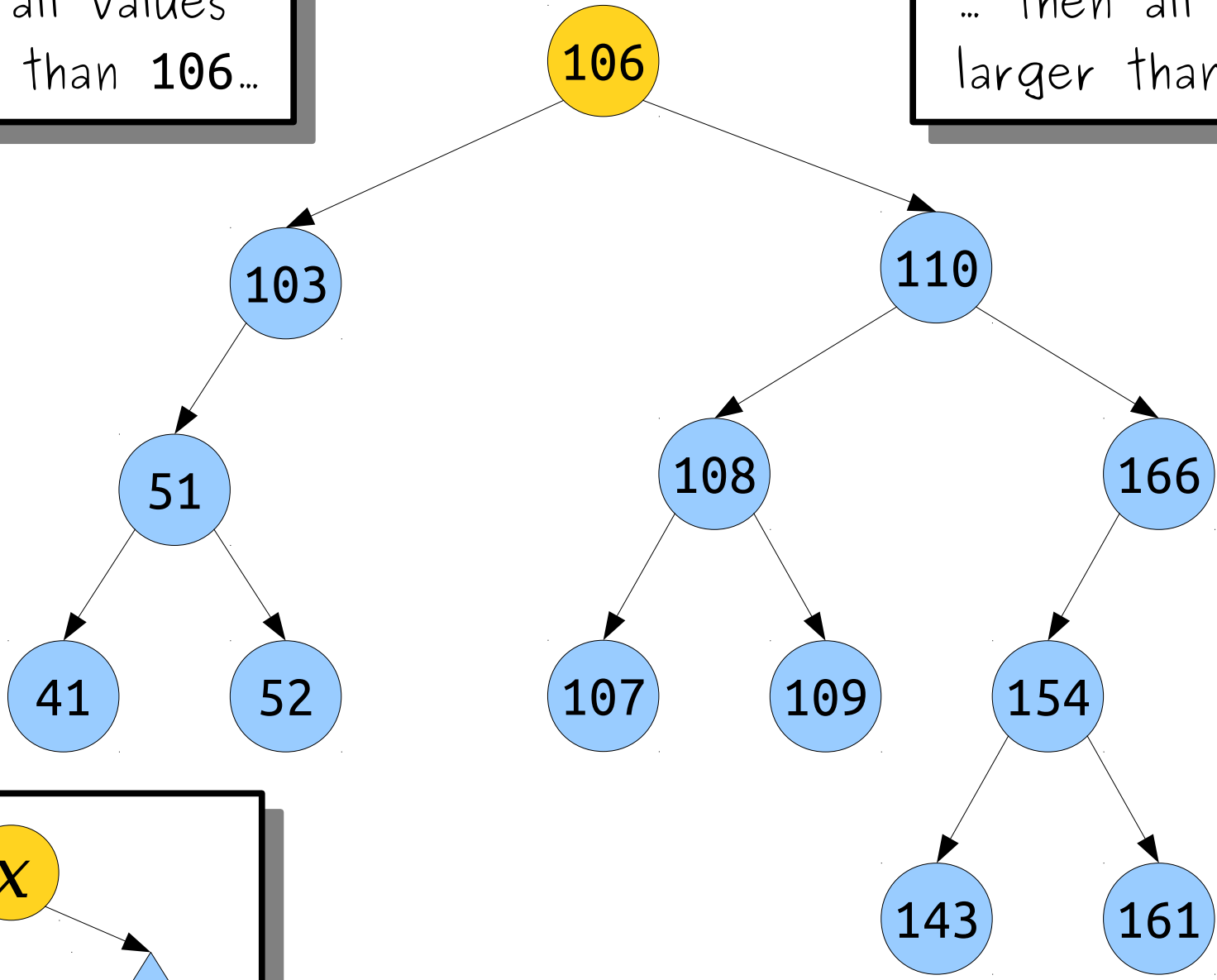
... and whose right
subtree is a BST
of larger values.



...then 106...

Print all values smaller than 106...

... then all values larger than 106.



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!

Time-Out for Announcements!



WiCS Speed Mentoring Night

network with grad students + learn about their research!

who Grad students sharing their research!
date + time FEB 26 @ 6-7 PM
location Gates 219



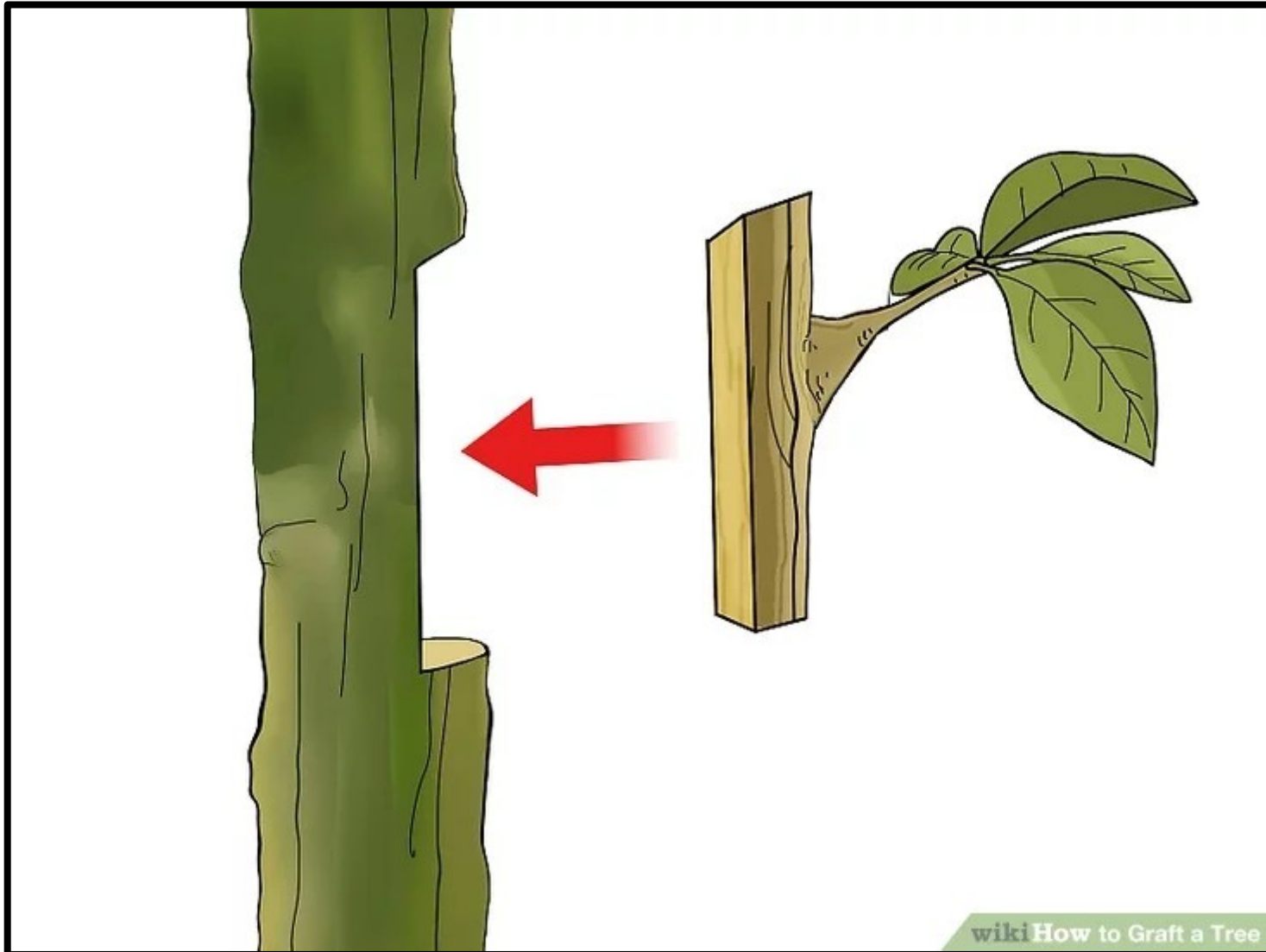
For more details, check out the [*Facebook event*](#).

Assignment 5

- Assignment 5 (***Data Sagas***) is due on Wednesday at the start of class.
 - Have questions? Stop by the LaIR (for coding questions) or CLaIR (for conceptual questions!)
 - Recommendation: aim to complete the first three parts of this assignment by Monday.

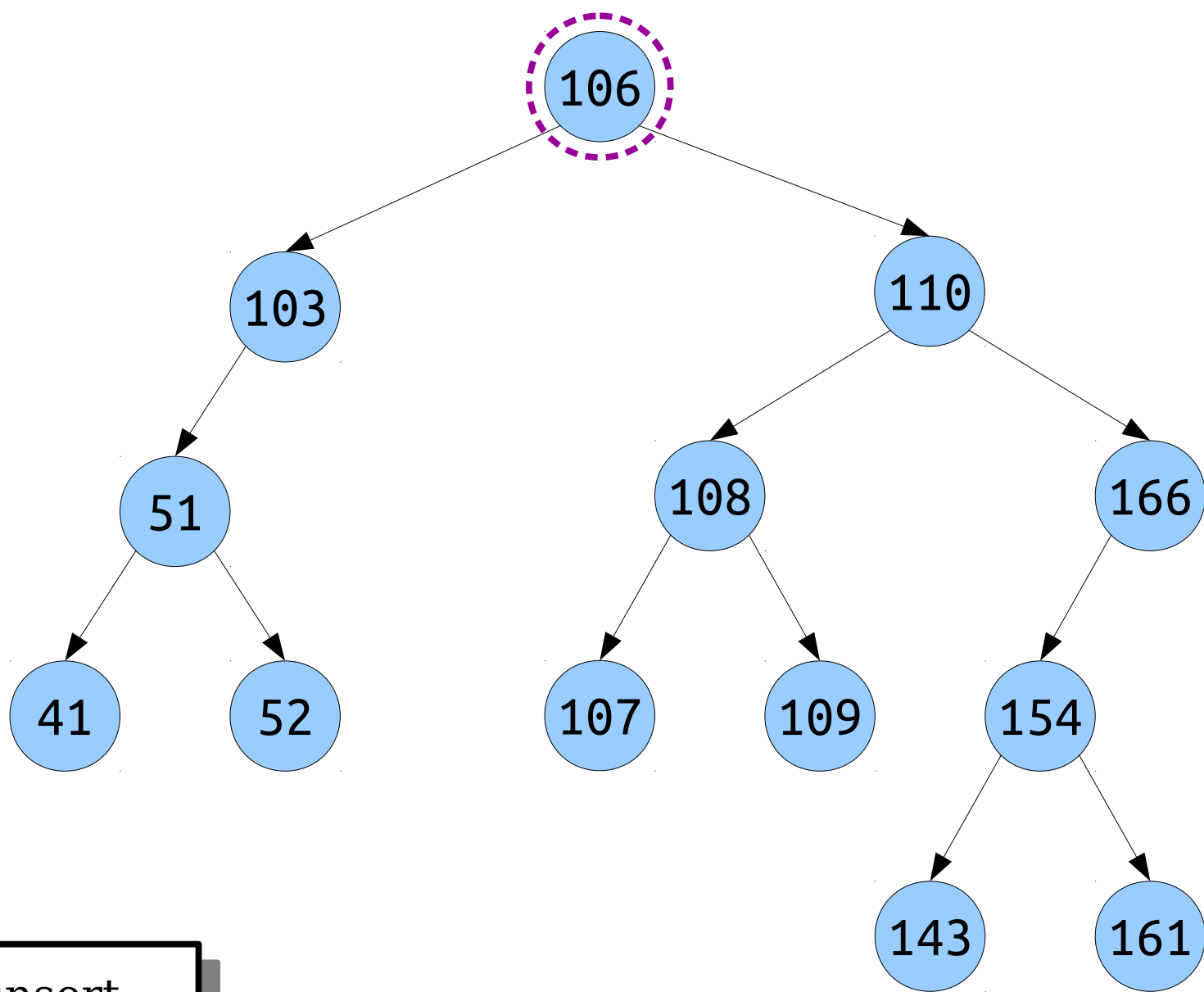
Back to our regularly
scheduled programming...

Adding to Trees



wikiHow to Graft a Tree

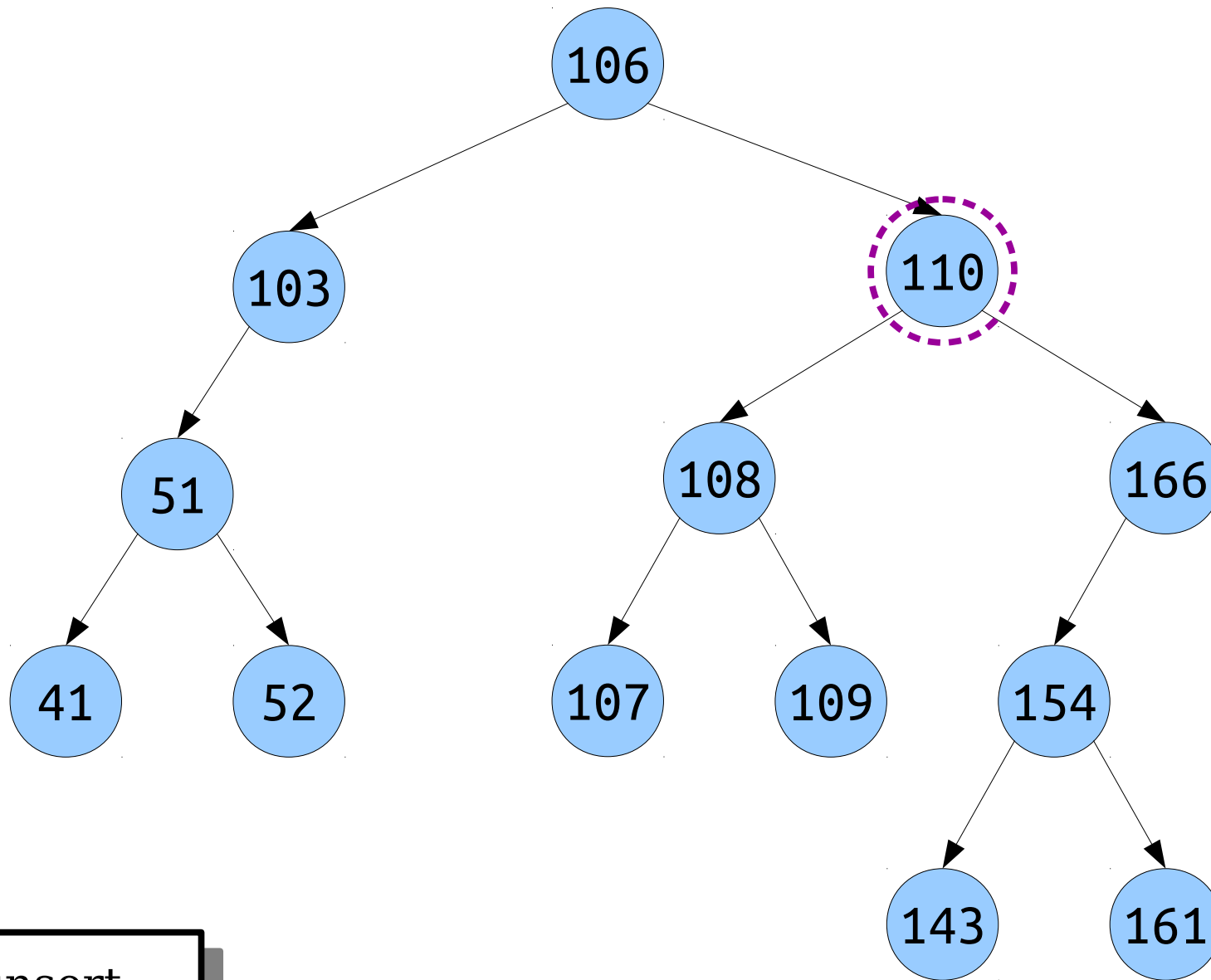
Thanks,
WikiHow!



Let's insert

147

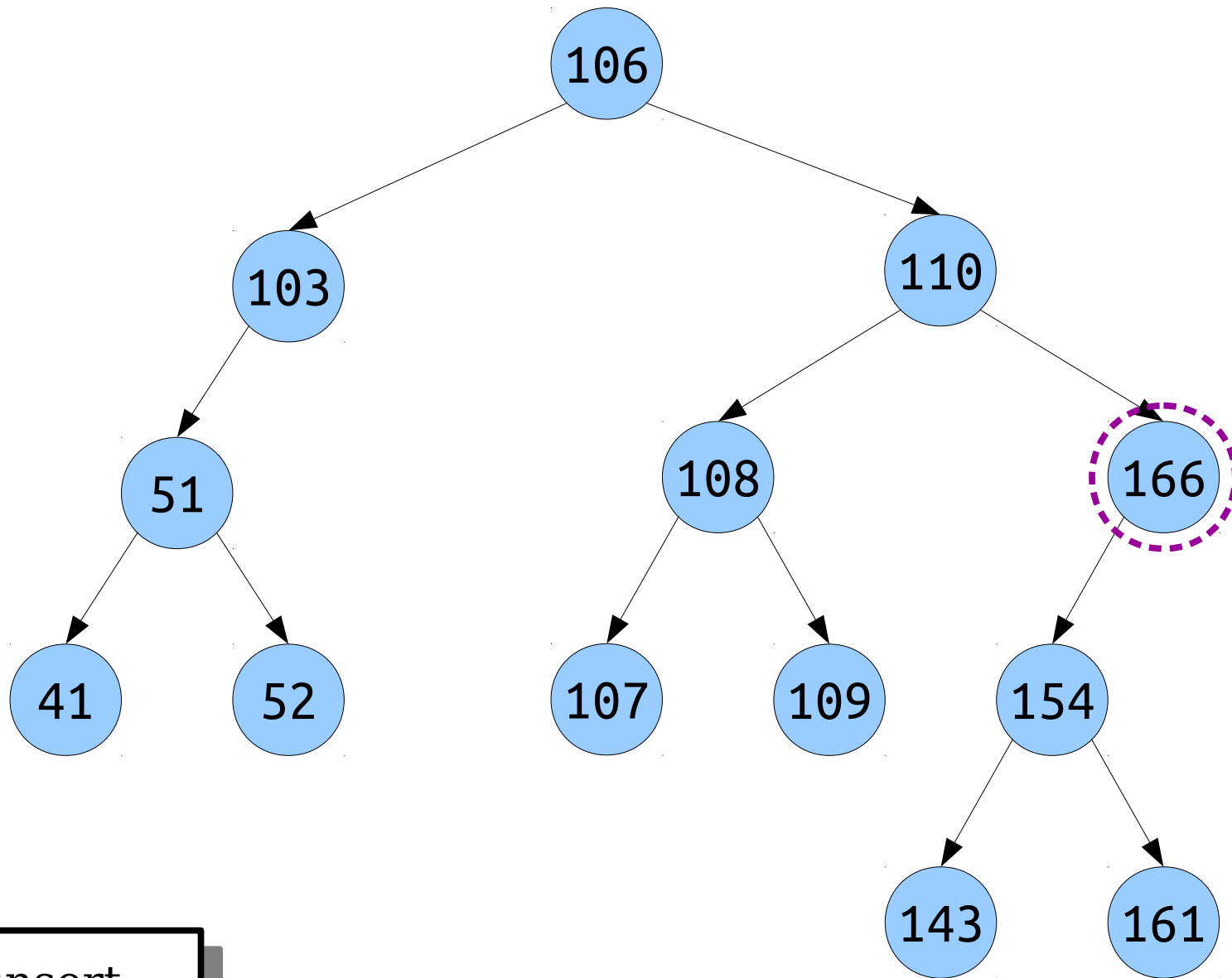
into this tree.



Let's insert

147

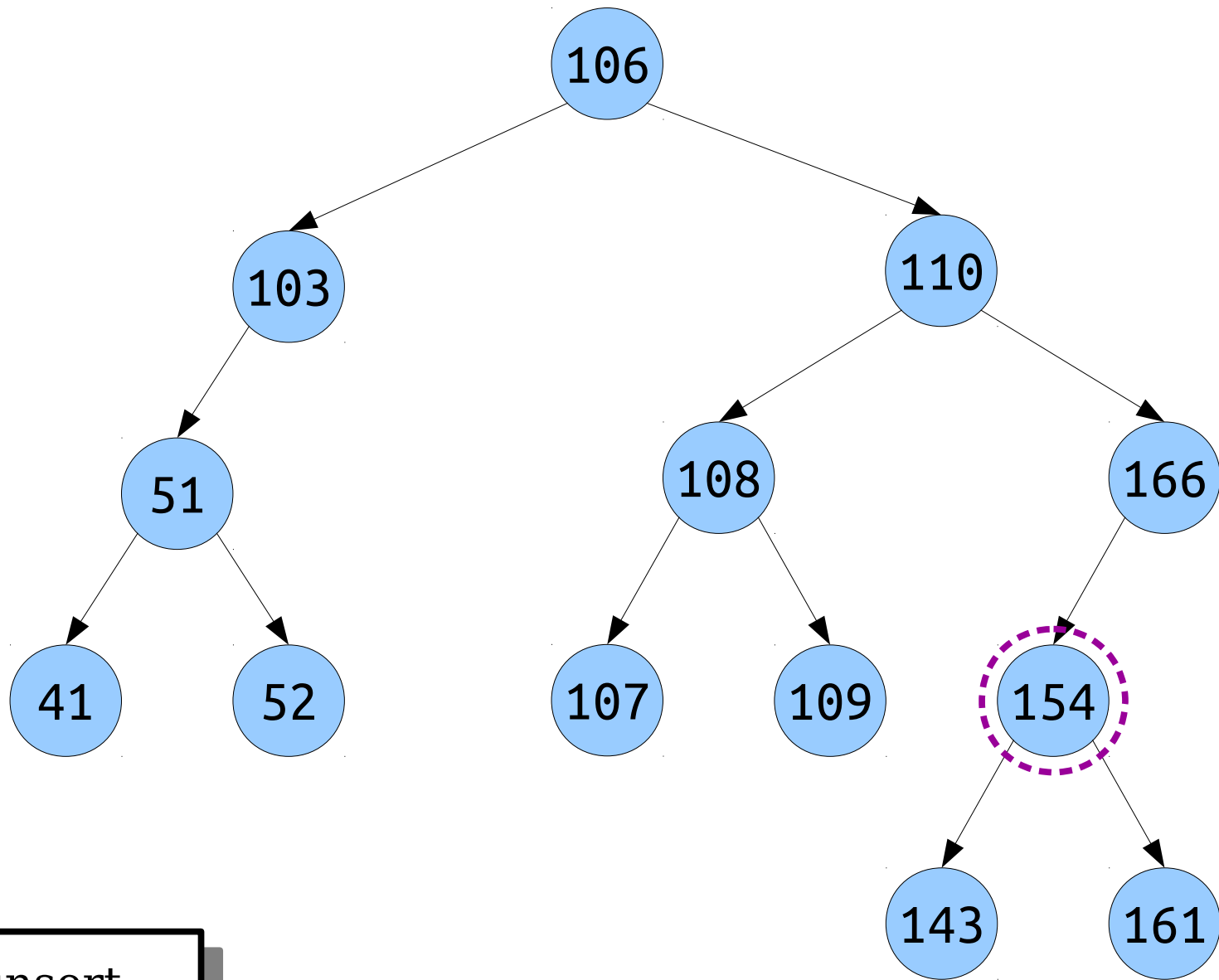
into this tree.



Let's insert

147

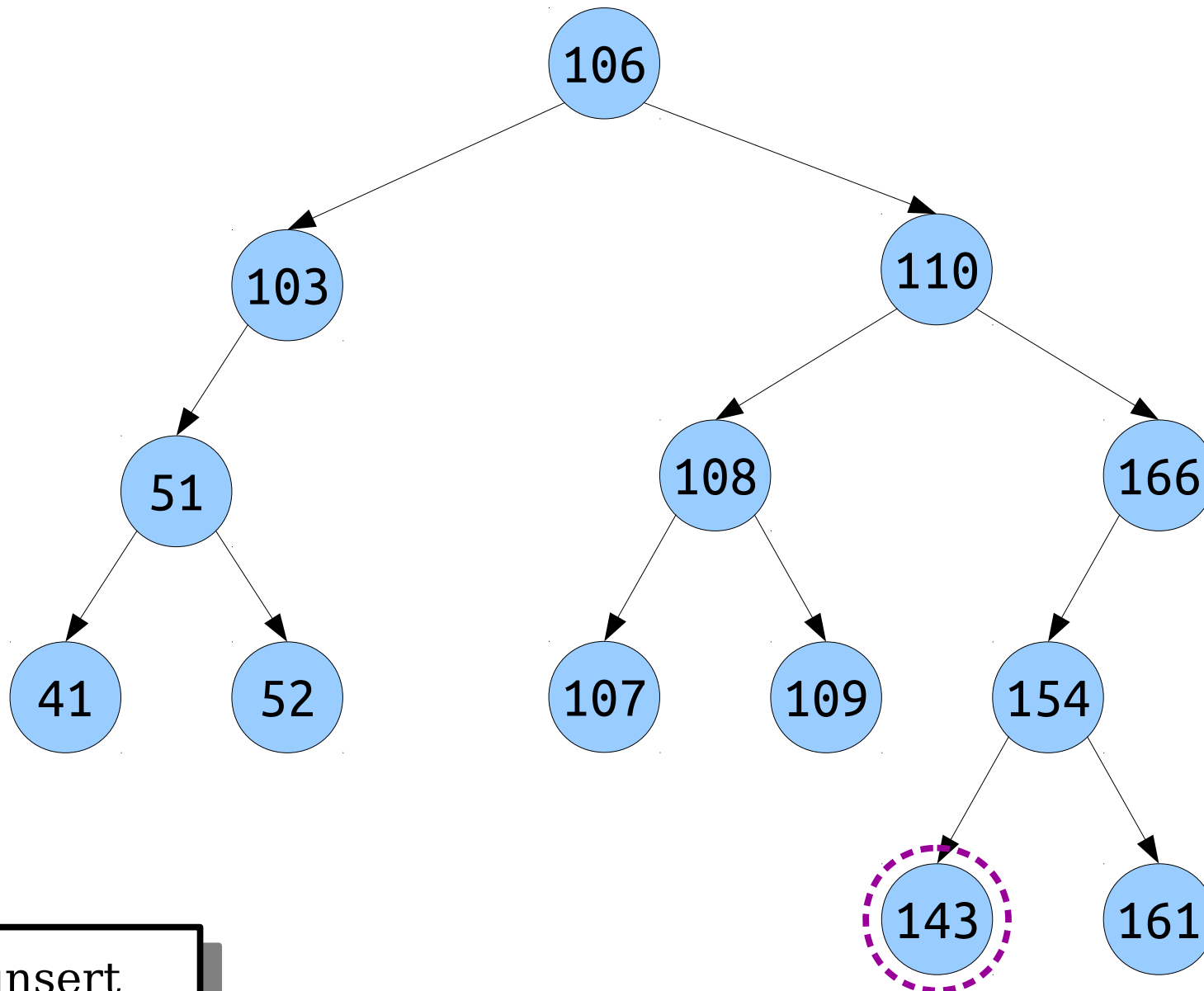
into this tree.



Let's insert

147

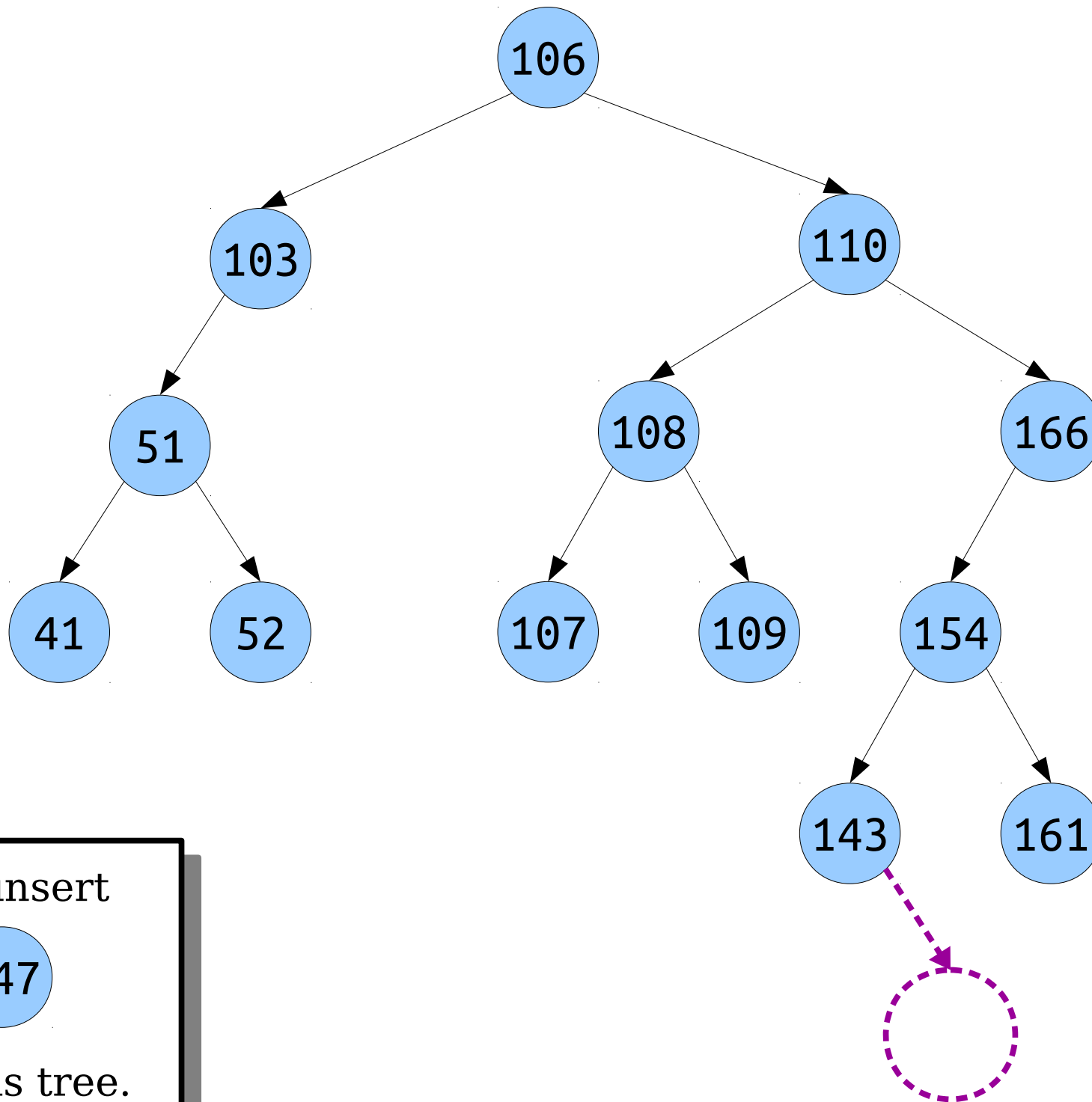
into this tree.



Let's insert

147

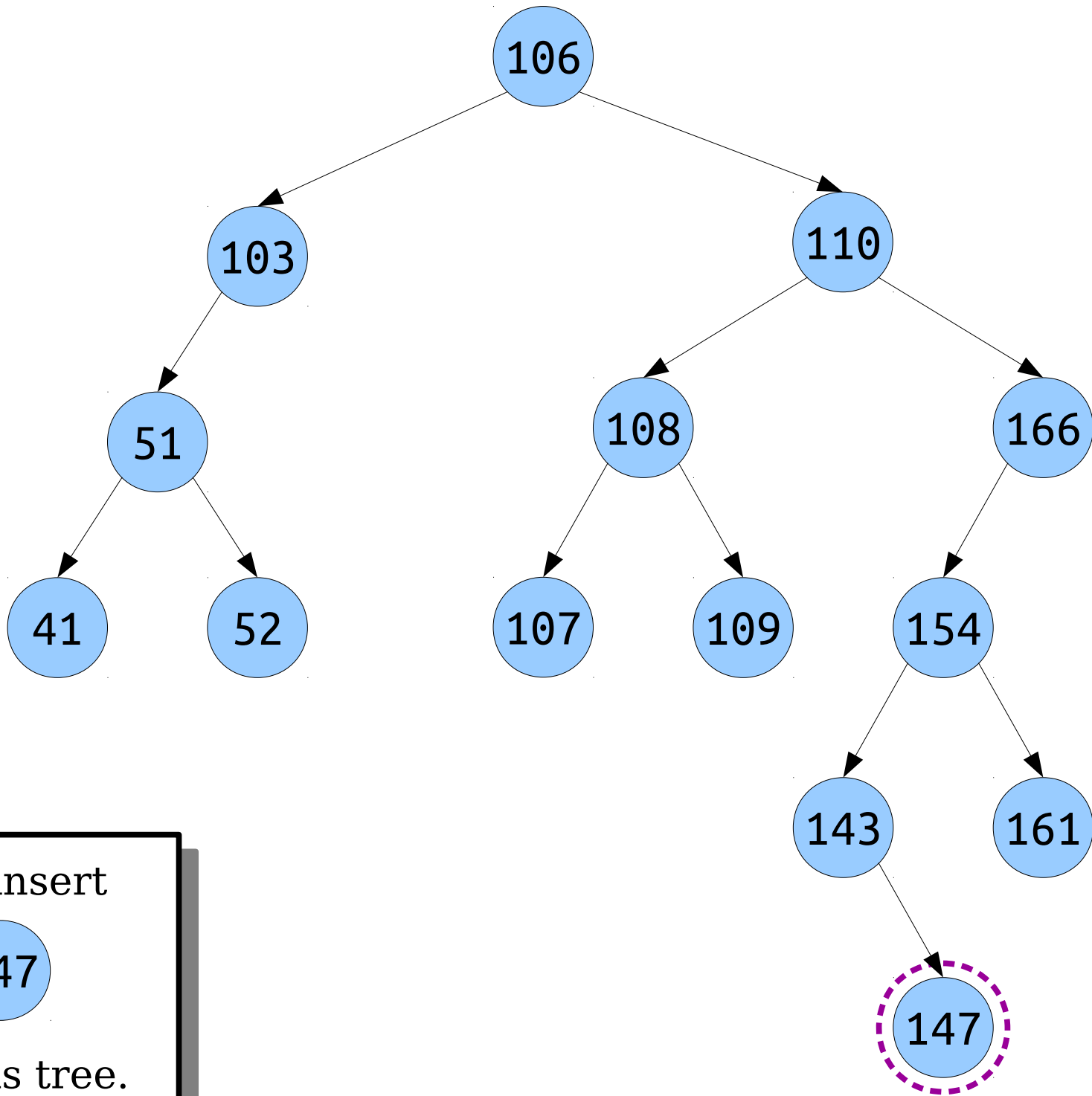
into this tree.



Let's insert

147

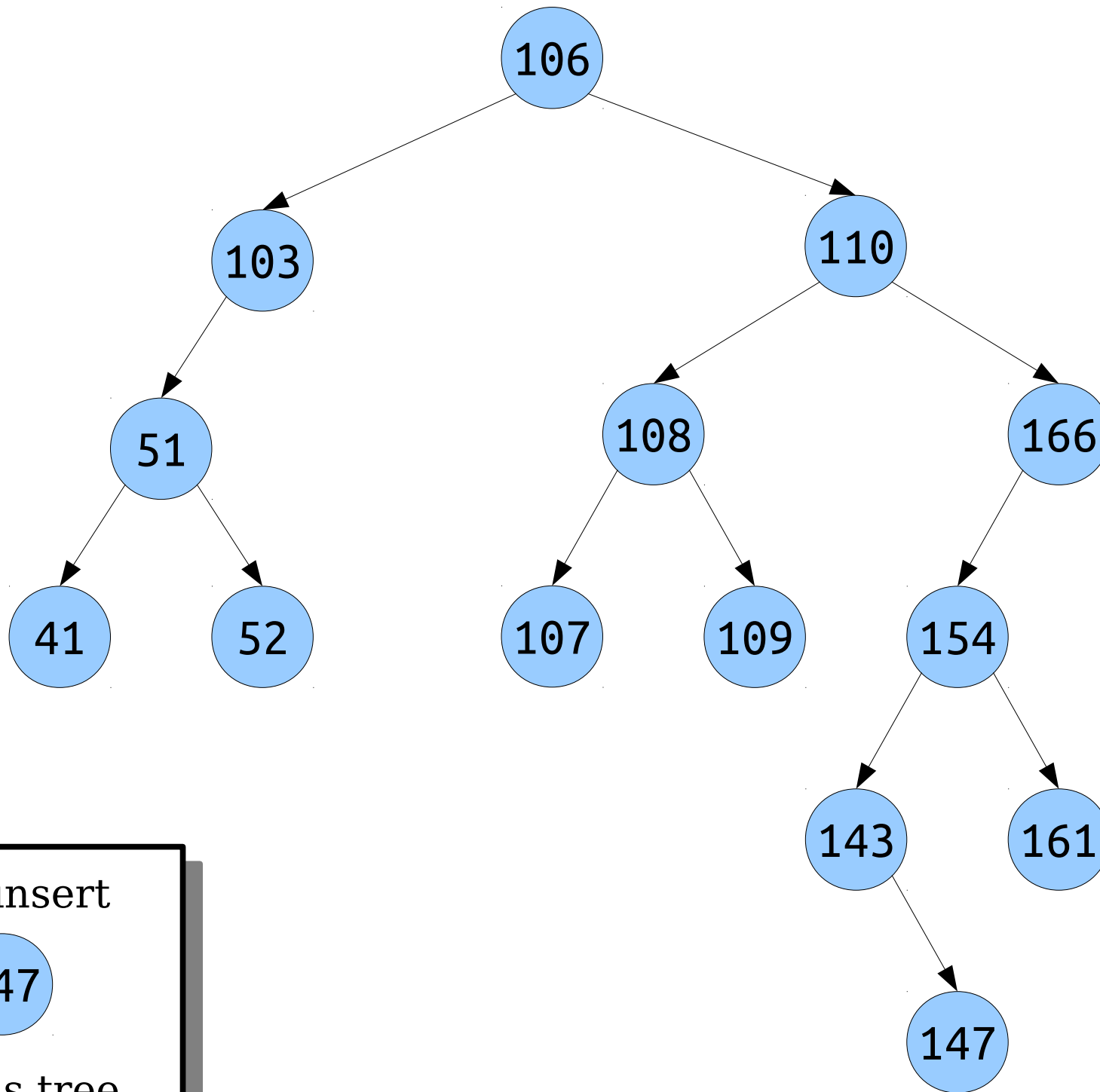
into this tree.



Let's insert

147

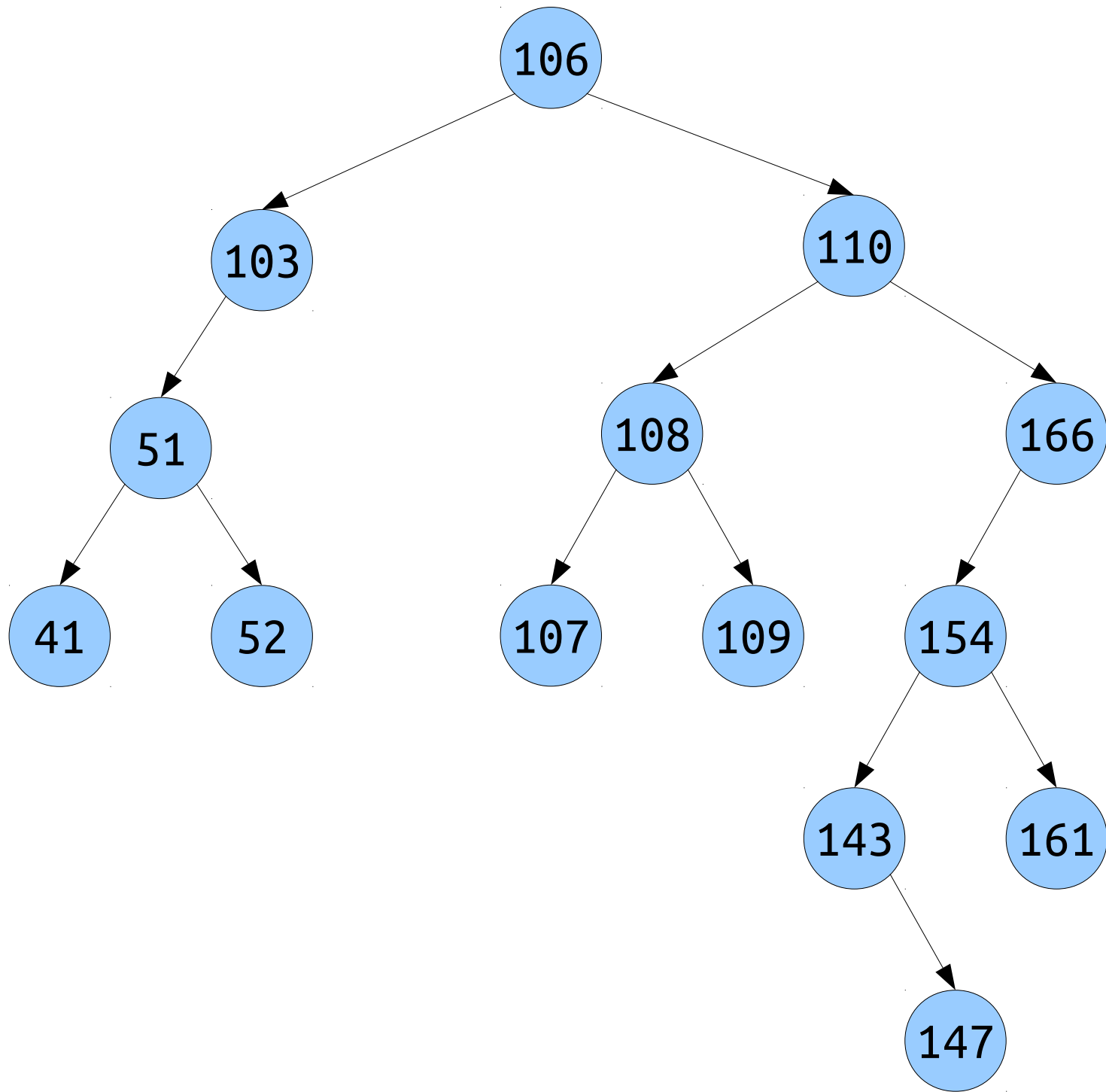
into this tree.

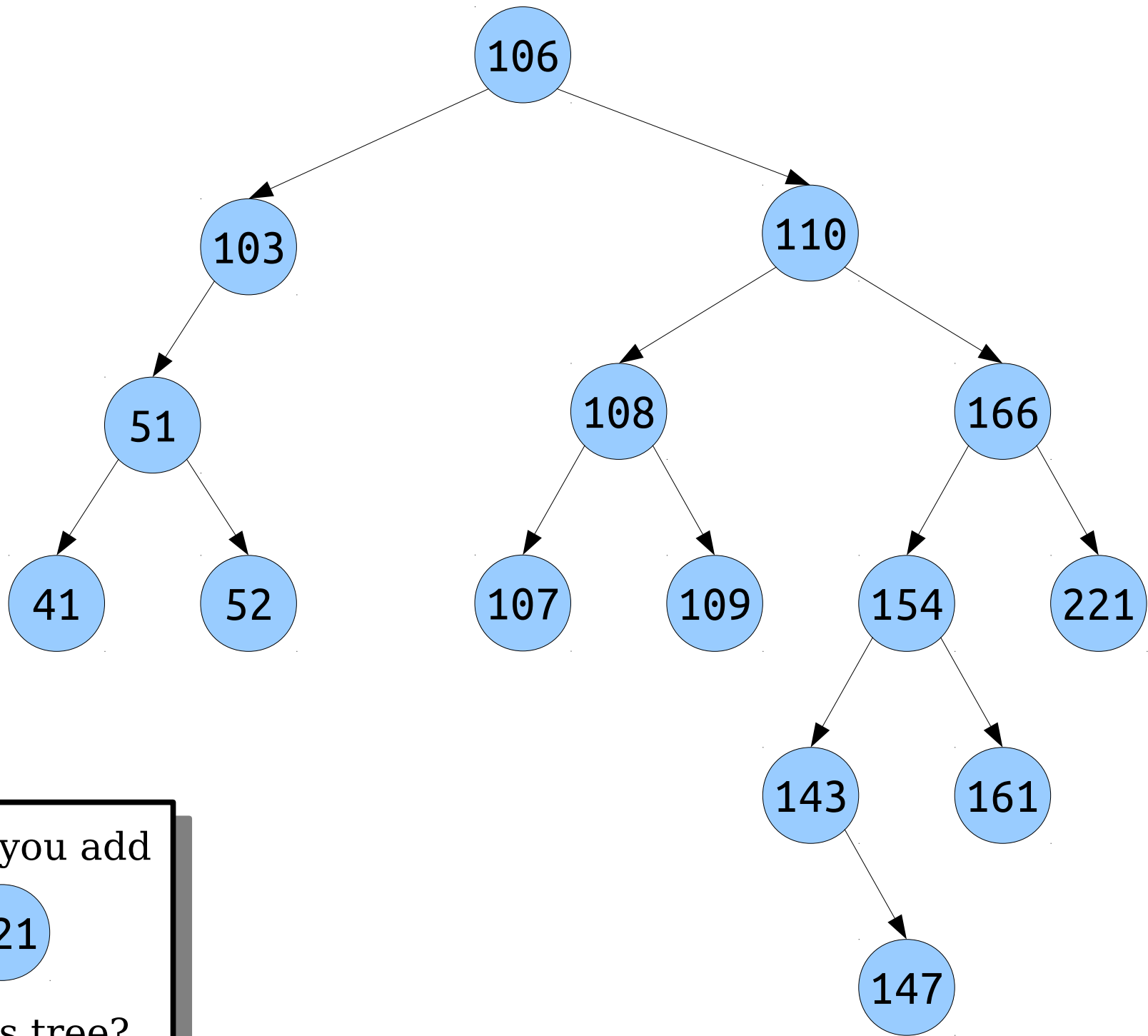


Let's insert

147

into this tree.





How do you add

221

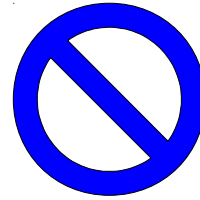
into this tree?

Let's Code it Up!

A Binary Search Tree Is Either...

an empty tree,
represented by

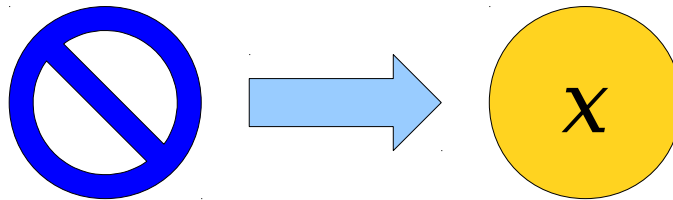
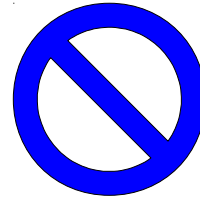
`nullptr`



A Binary Search Tree Is Either...

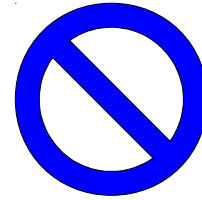
an empty tree,
represented by

`nullptr`

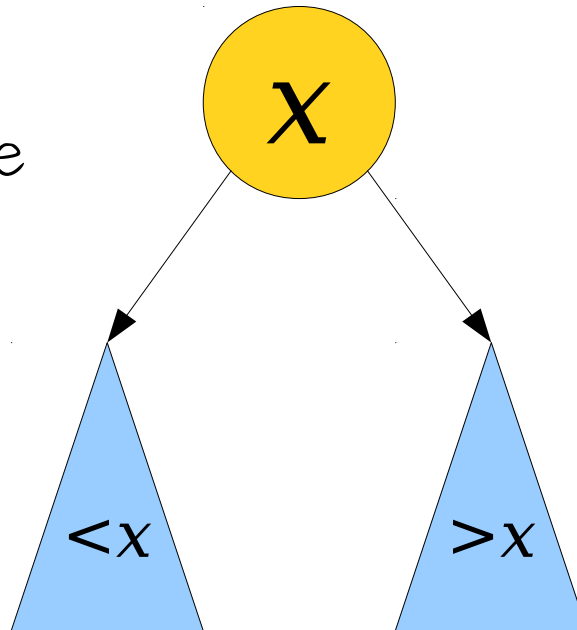


A Binary Search Tree Is Either...

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.

Your Action Items

- ***Keep working on Assignment 5.***
 - Aim to complete multiway merge, lower bound search, and heap priority queue by Monday.
- ***Read Chapter 16 of the textbook.***
 - There's a bunch of BST topics in there, along with a different intuition for how they work.

Next Time

- ***More BST Fun***
 - Some other cool tricks and techniques!
- ***Custom Types in Sets***
 - Resolving a longstanding issue.