CS11212 - **Spring** 2022

Data Structures & Introduction to **Algorithms**

Data Structures

Trees: Definitions and Properties

Ibrahim Albluwi

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

Candidate implementations.

	`insert(val)`	`remove(val)`	`contains(val)`
Unordered DLL Unordered SLL			

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

Candidate implementations.

both need searching in the list

	`insert(val)`	`remove(val)`	`contains(val)`
Unordered DLL	`O(n)`	`O(n)`	`O(n)`
Unordered SLL	`O(n)`	`O(n)`	`O(n)`

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

Candidate implementations.

both need searching in the list

	`insert(val)`	`remove(val)`	`contains(val)`
Unordered DLL	`O(n)`	`O(n)`	`O(n)`
Unordered SLL	`O(n)`	`O(n)`	`O(n)`

Bad!

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

Candidate implementations.

both need searching in the list

	`insert(val)`	`remove(val)`	`contains(val)`
Unordered DLL	`O(n)`	`O(n)`	`O(n)`
Unordered SLL	`O(n)`	`O(n)`	`O(n)`
Ordered DLL	`O(n)`	`O(n)`	`O(n)`
Ordered SLL	`O(n)`	`O(n)`	`O(n)`

Bad!

Maintaining order does not help!

A Set ADT

Problem. Design a data structure to support the following operations:

- `insert(val)   // add val to the set if it is not already in the set.`
- `remove(val)   // remove val from the set of items.`
- `contains(val) // check if val belongs to the set.`

Candidate implementations.

both need searching in the list

	`insert(val)`	`remove(val)`	`contains(val)`	
Unordered DLL	`O(n)`	`O(n)`	`O(n)`	Bad!
Unordered SLL	`O(n)`	`O(n)`	`O(n)`	
Ordered DLL	`O(n)`	`O(n)`	`O(n)`	Maintaining order does not help!
Ordered SLL	`O(n)`	`O(n)`	`O(n)`	
Unordered Array	`O(n)`	`O(n)`	`O(n)`	Bad!
Ordered Array	`O(n)`	`O(n)`	`O(log n)`	Great! insert/remove are bad!

A Set ADT

What is going on?

Great!

- Linked lists support efficient insertion and deletion.
  Provided that there is a pointer to the insertion or deletion position.
- Arrays support efficient search.
  Provided that the array is sorted.

Bad!

- Linked lists do not support efficient search.
  Sorted linked lists do not support efficient binary search because it requires direct access.
- Arrays do not support efficient insertion and removal at any position.
  Elements need to be shifted.

A Set ADT

What is going on?

Great!

- Linked lists support efficient insertion and deletion.
  Provided that there is a pointer to the insertion or deletion position.
- Arrays support efficient search.
  Provided that the array is sorted.

Bad!

- Linked lists do not support efficient search.
  Sorted linked lists do not support efficient binary search because it requires direct access.
- Arrays do not support efficient insertion and removal at any position.
  Elements need to be shifted.

Can we achieve the good of the two?

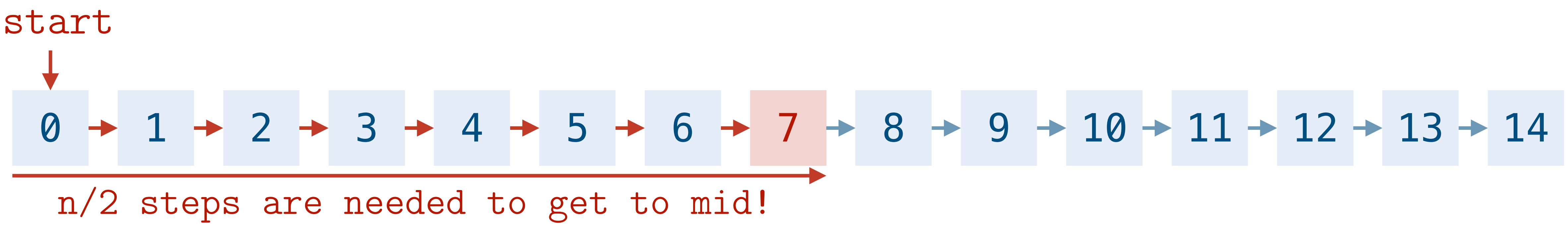
Efficient search (as in binary search)
Efficient insertion / removal (given a pointer to the insertion / removal position)

Binary Search on Linked Lists?

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → 14

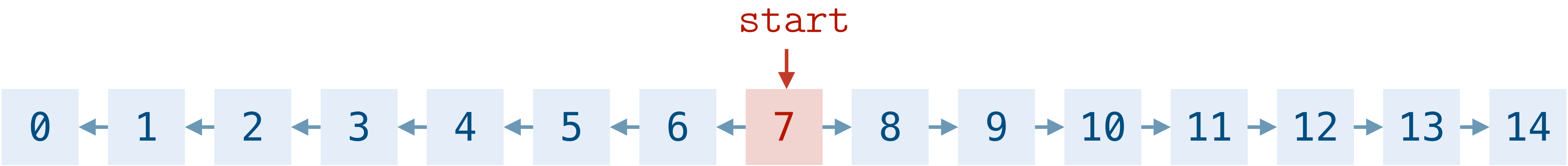
Can we perform *binary search* on a sorted linked list?

Binary Search on Linked Lists?



$O(n)$ operations to get to the middle element!

Binary Search on Linked Lists?



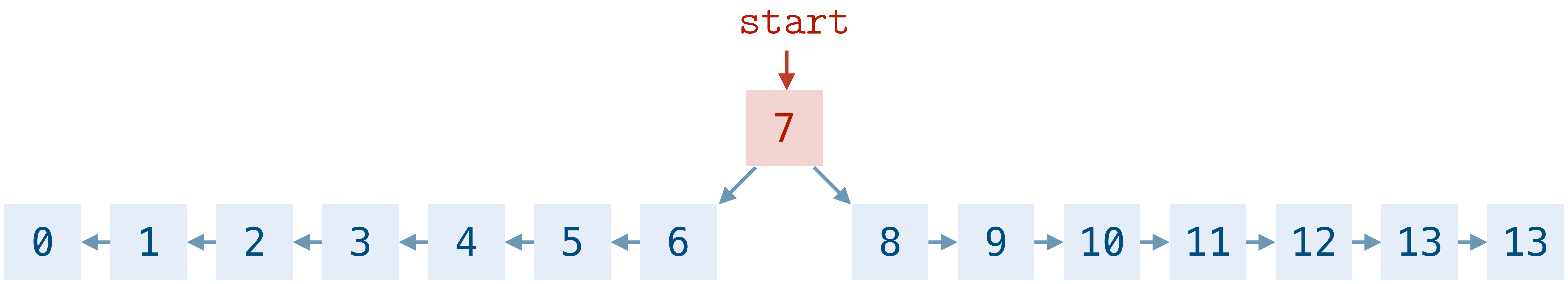
`start`

`0 ← 1 ← 2 ← 3 ← 4 ← 5 ← 6 ← 7 → 8 → 9 → 10 → 11 → 12 → 13 → 14`

Idea: Maintain a pointer to the middle element.

The middle element is now accessible in $O(1)$.

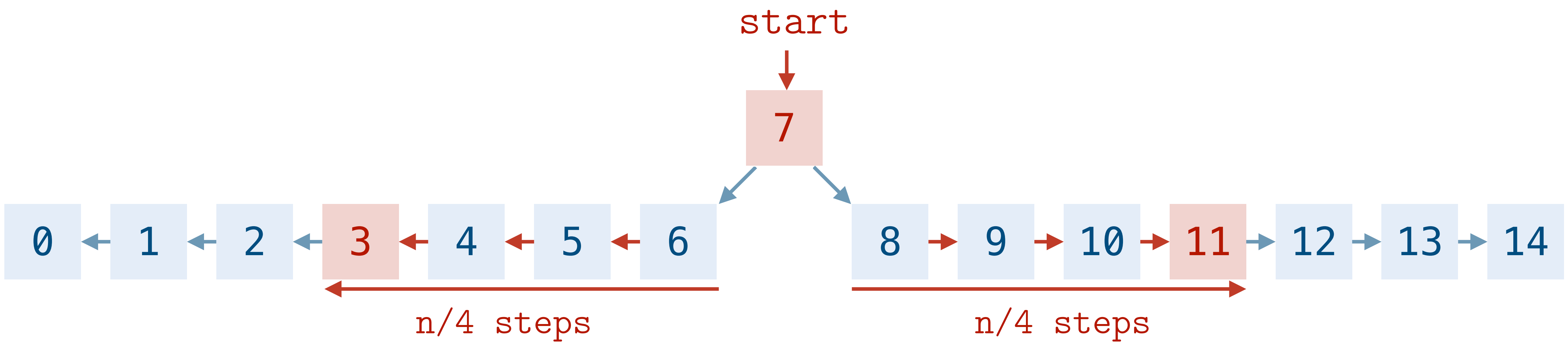
Binary Search on Linked Lists?



Idea: Maintain a pointer to the middle element.

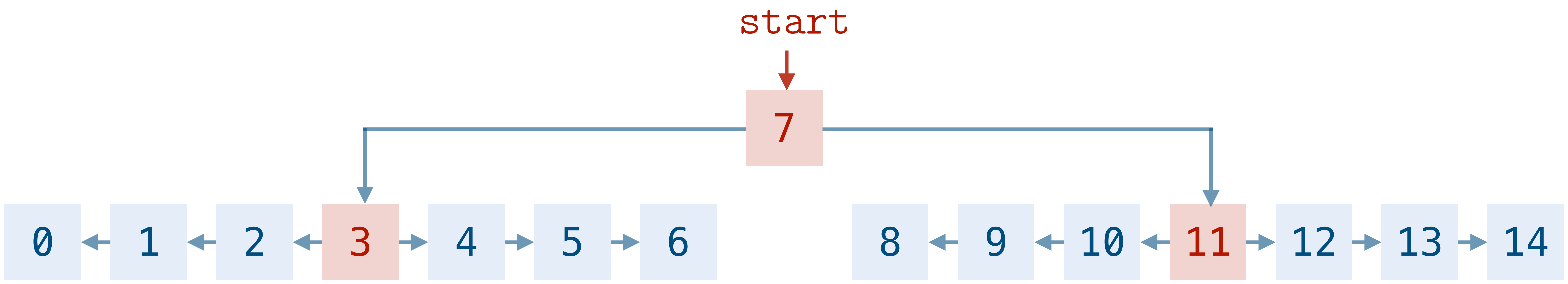
The middle element is now accessible in $O(1)$.

Binary Search on Linked Lists?



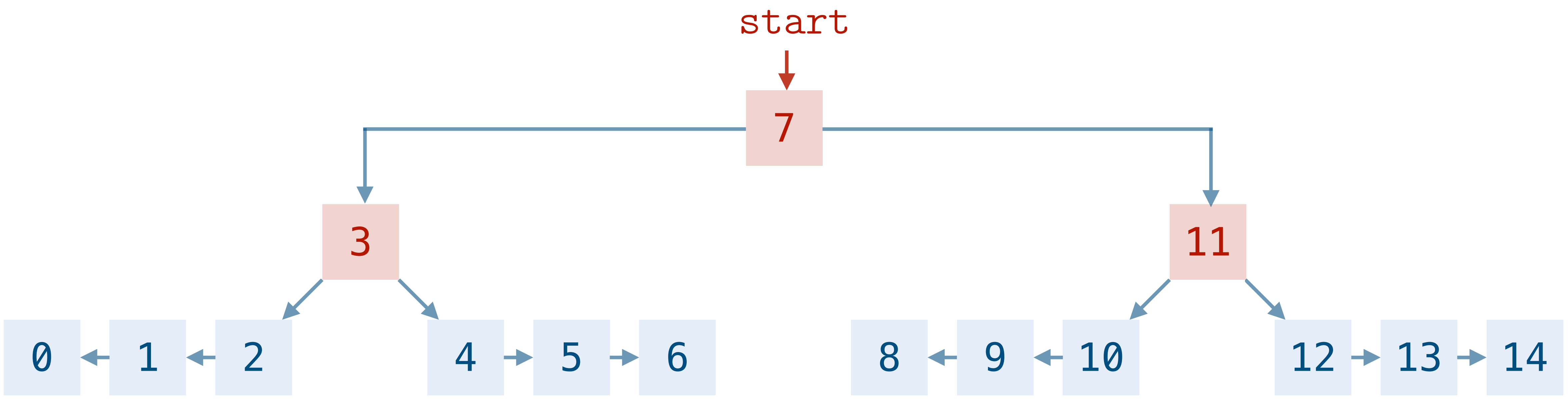
Still $O(n)$ to get to the middle of the left half or the middle of the right half!

Binary Search on Linked Lists?



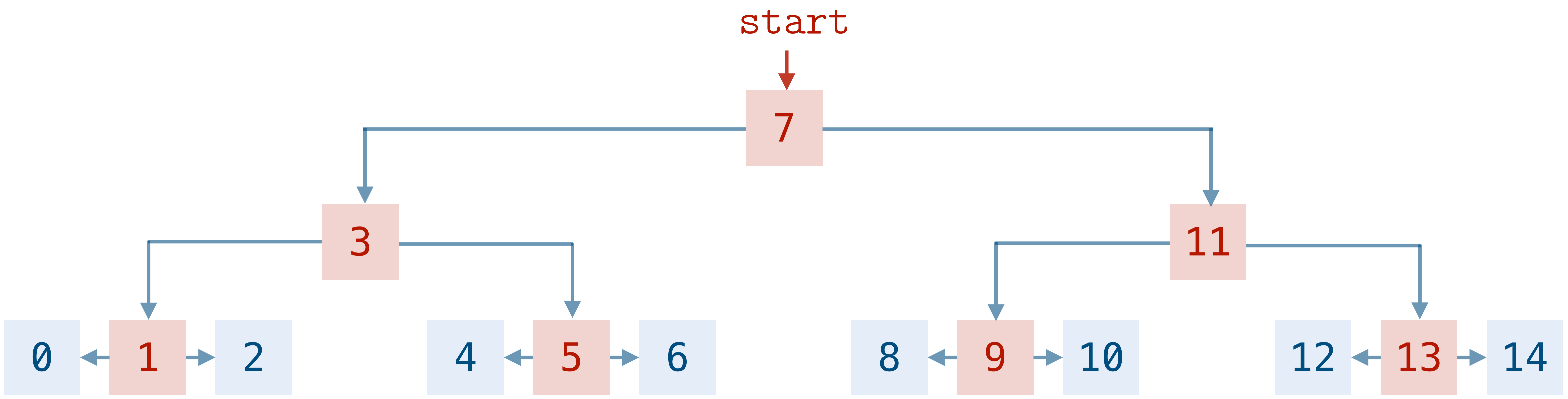
Idea (*again*): Maintain a pointer to the middle elements!

Binary Search on Linked Lists?



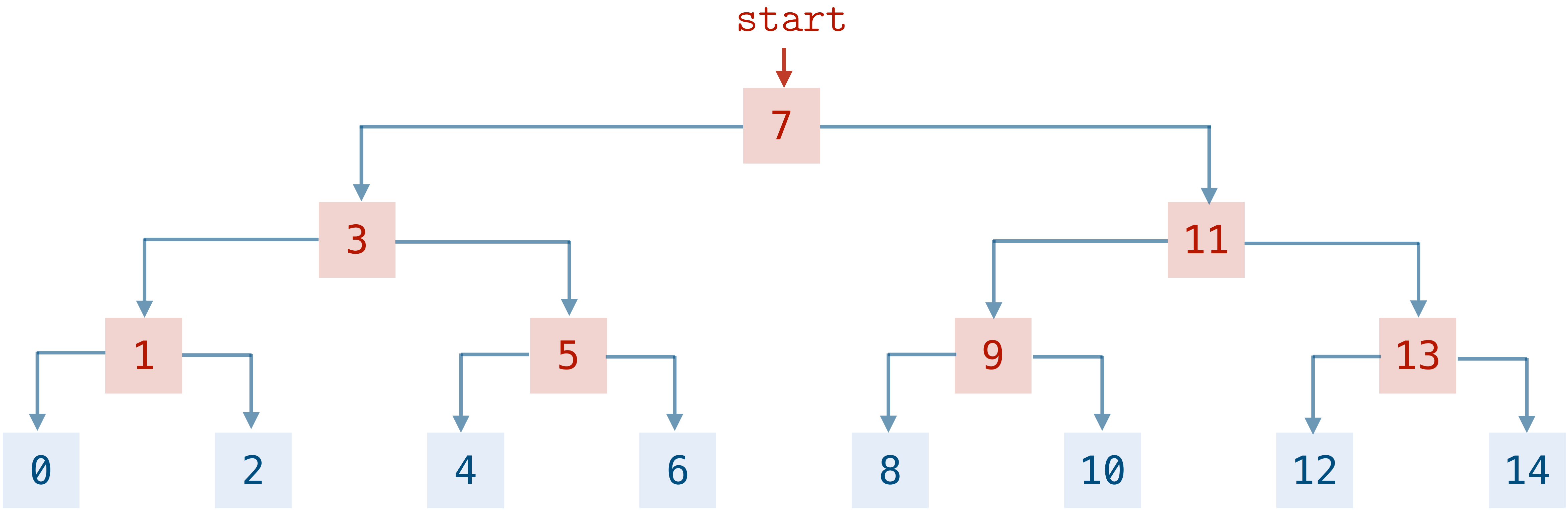
Idea (*again*): Maintain a pointer to the middle elements!

Binary Search on Linked Lists?



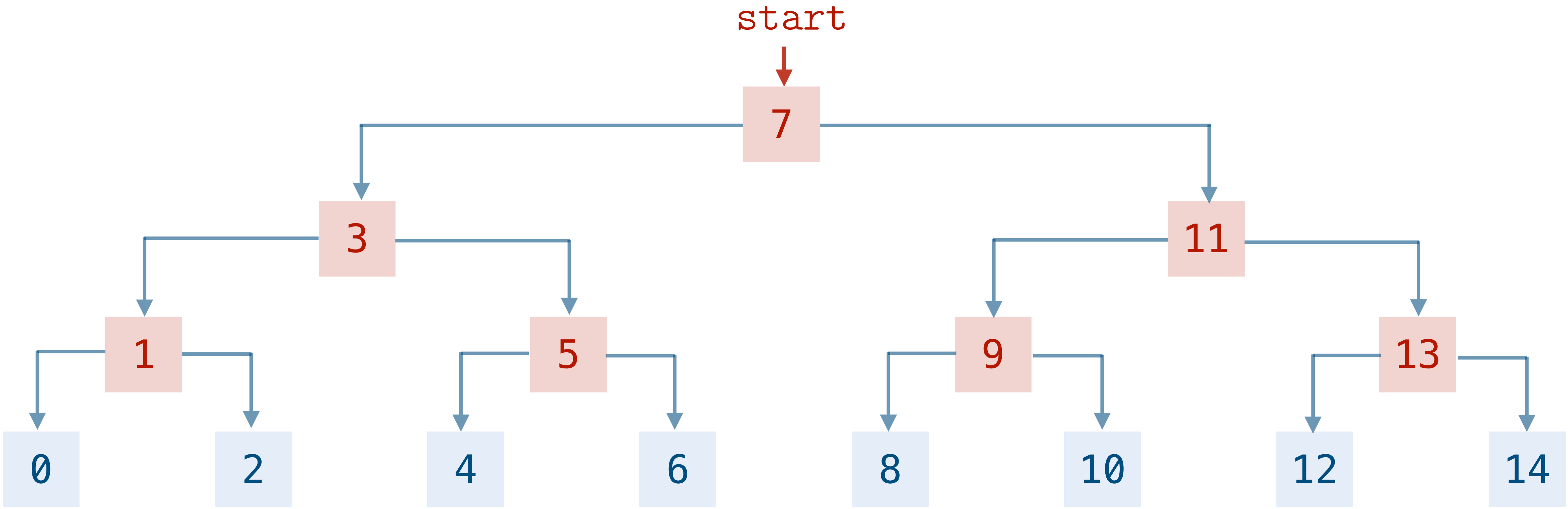
Idea (*again, again*): Maintain a pointer to the middle elements!

Binary Search on Linked Lists?



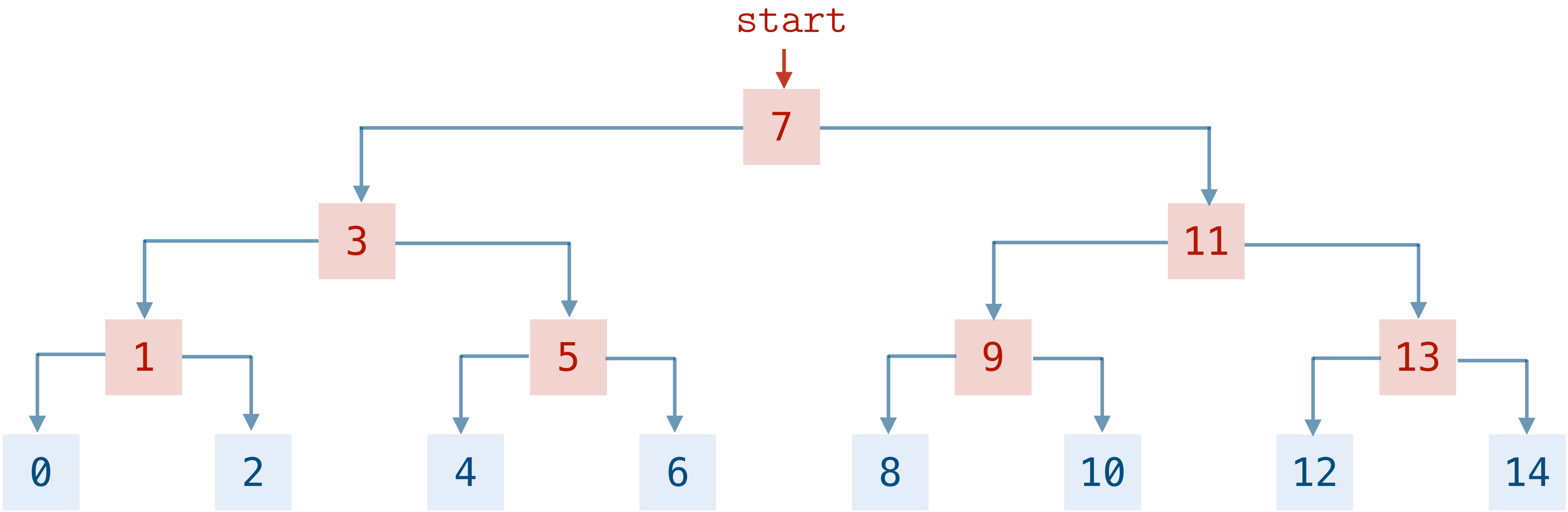
Idea (*again, again*): Maintain a pointer to the middle elements!

Binary Search on Linked Lists?



A Binary Search Tree!

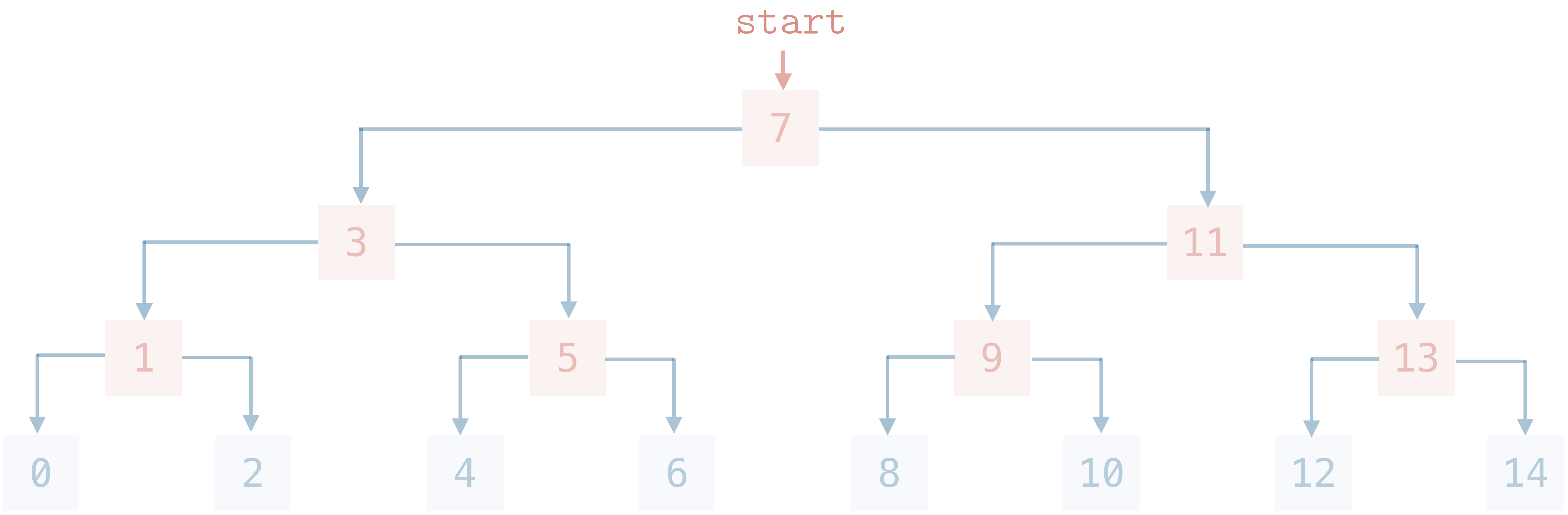
Proposed Implementation



A Binary Search Tree!

We need to build a linked *tree structure.*

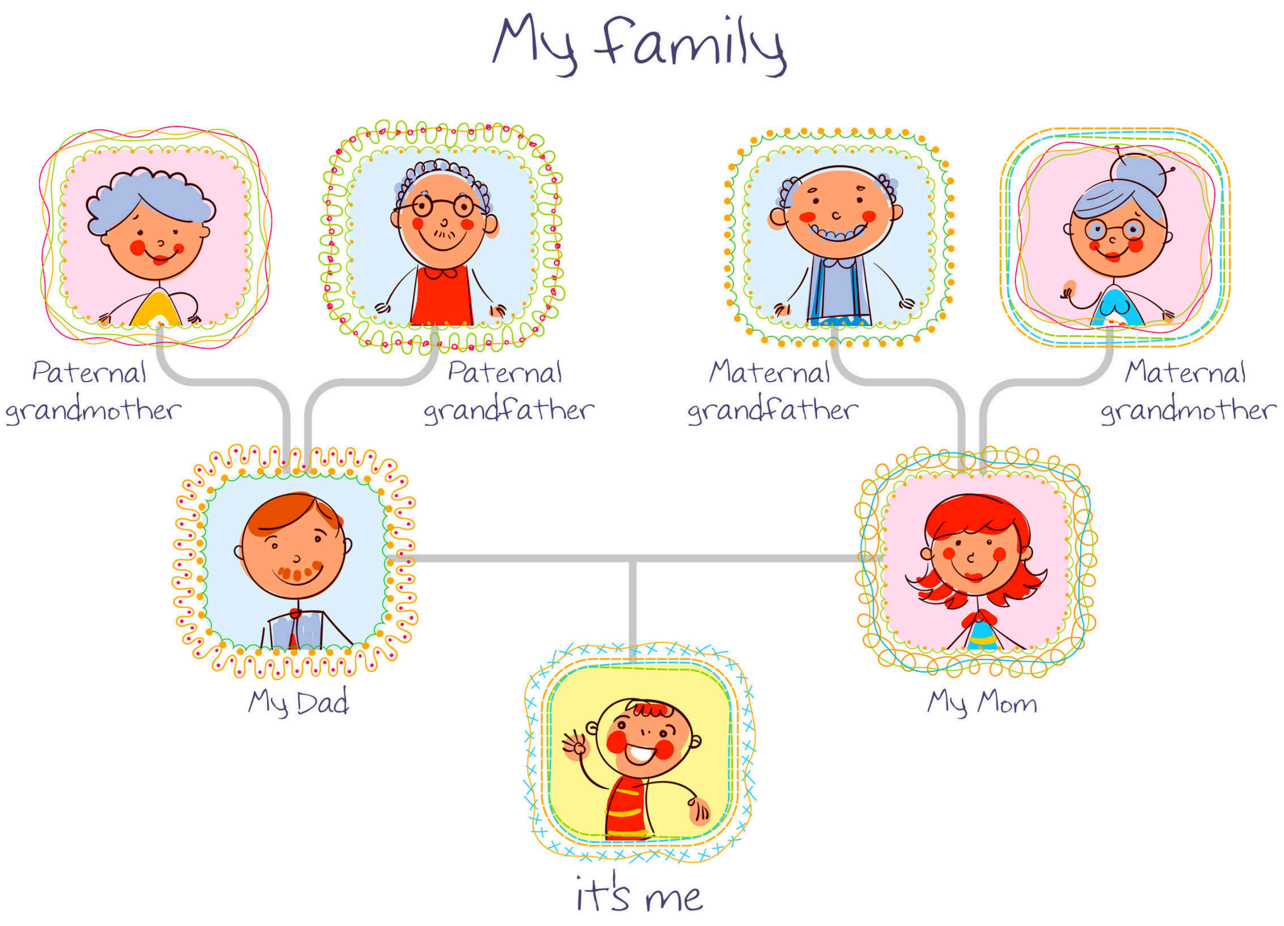
Proposed Implementation



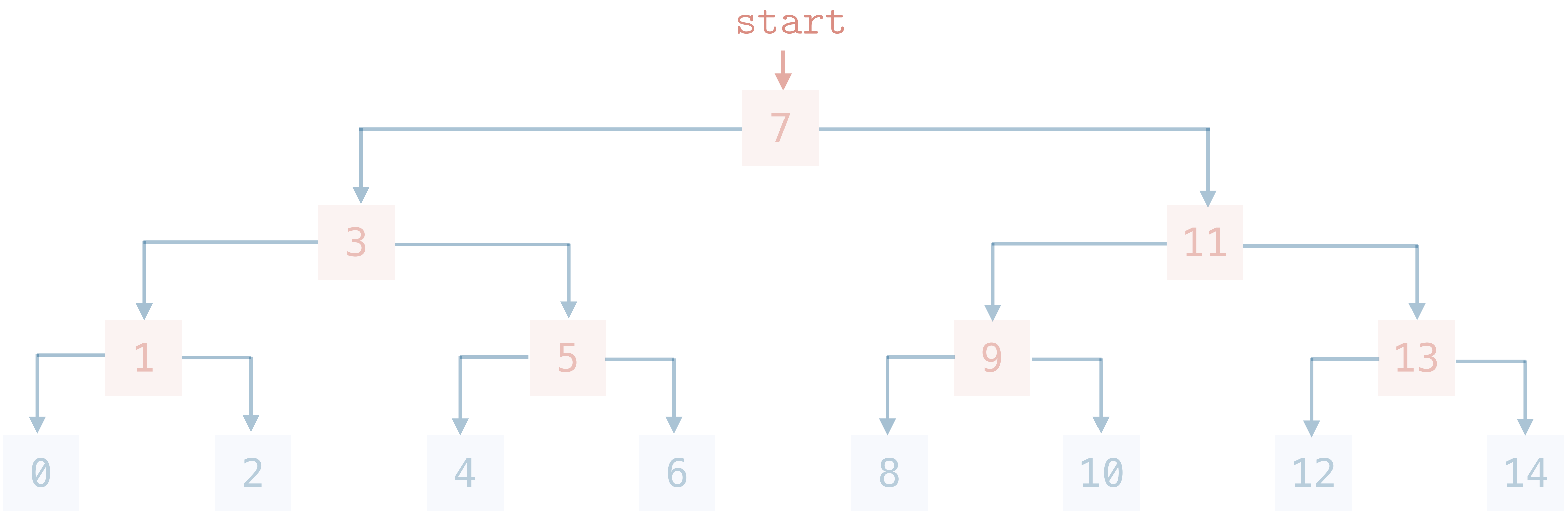
A Binary Search Tree!

We need to build a linked *tree structure.*

Trees are not useful only for efficient search. They are useful for representing hierarchies and relationships in general.



Proposed Implementation



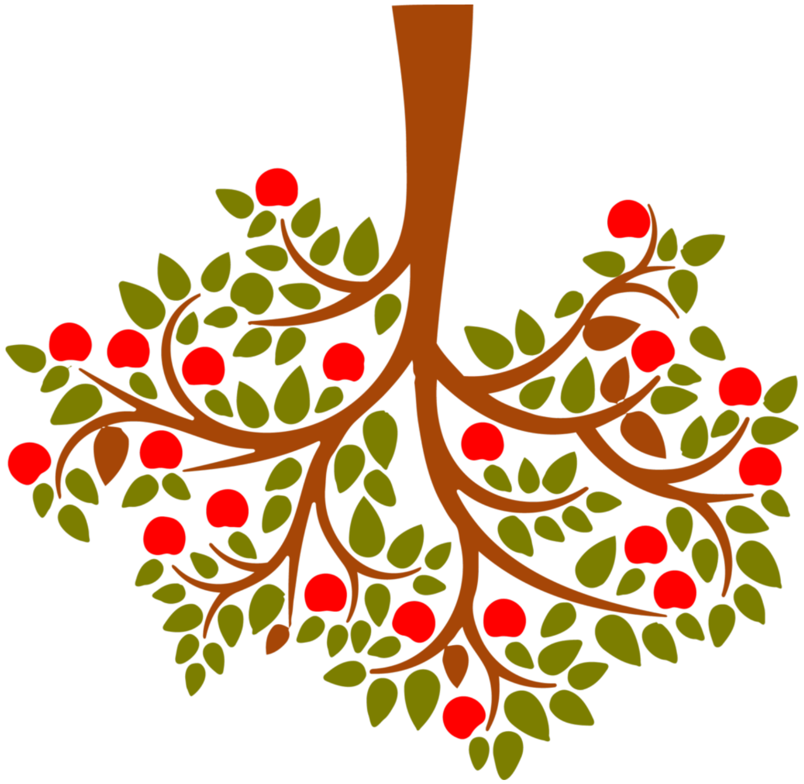
A Binary Search Tree!

We need to build a linked *tree structure.*

Trees are not useful only for efficient search. They are useful for representing hierarchies and relationships in general.

Questions.

- How do we *insert* into the tree?
- How do we *remove* from the tree?
- How do we *search* the tree?
- How do we *traverse* the tree?
- What *other operations* can we perform?
- What *properties* do trees have?
- … ?



Tree Data Structures

- Definitions and properties

Operations on BSTs

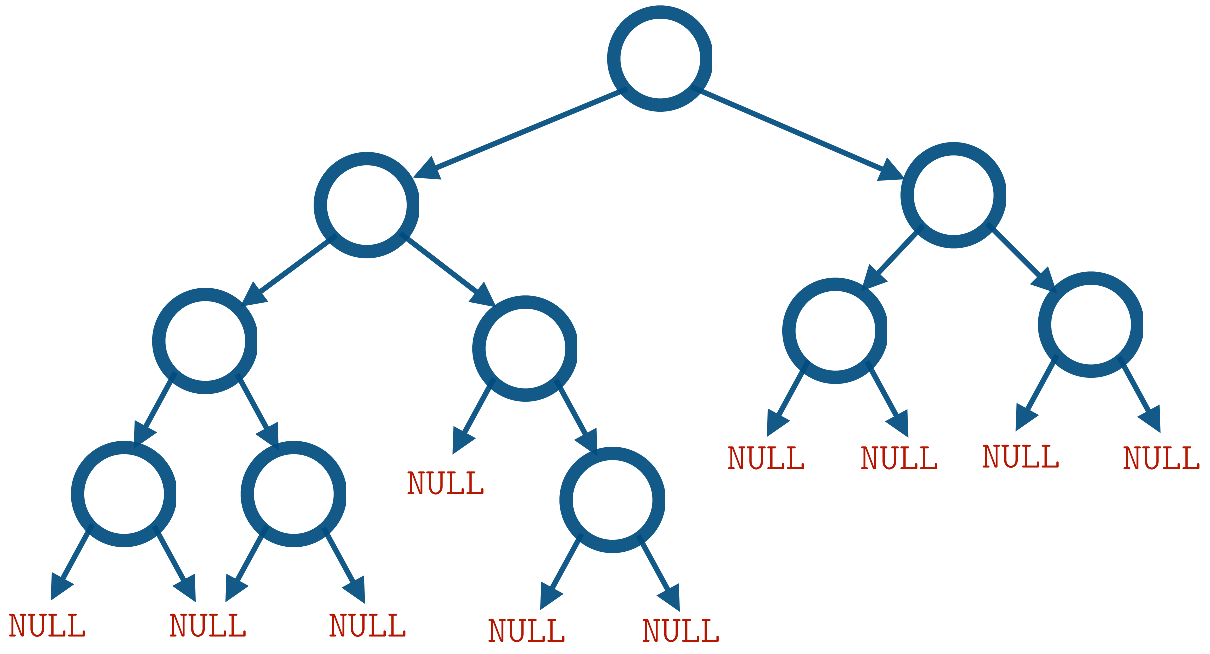
Balanced binary search trees

Tree traversals

Terminology

Definition. A *tree* is:

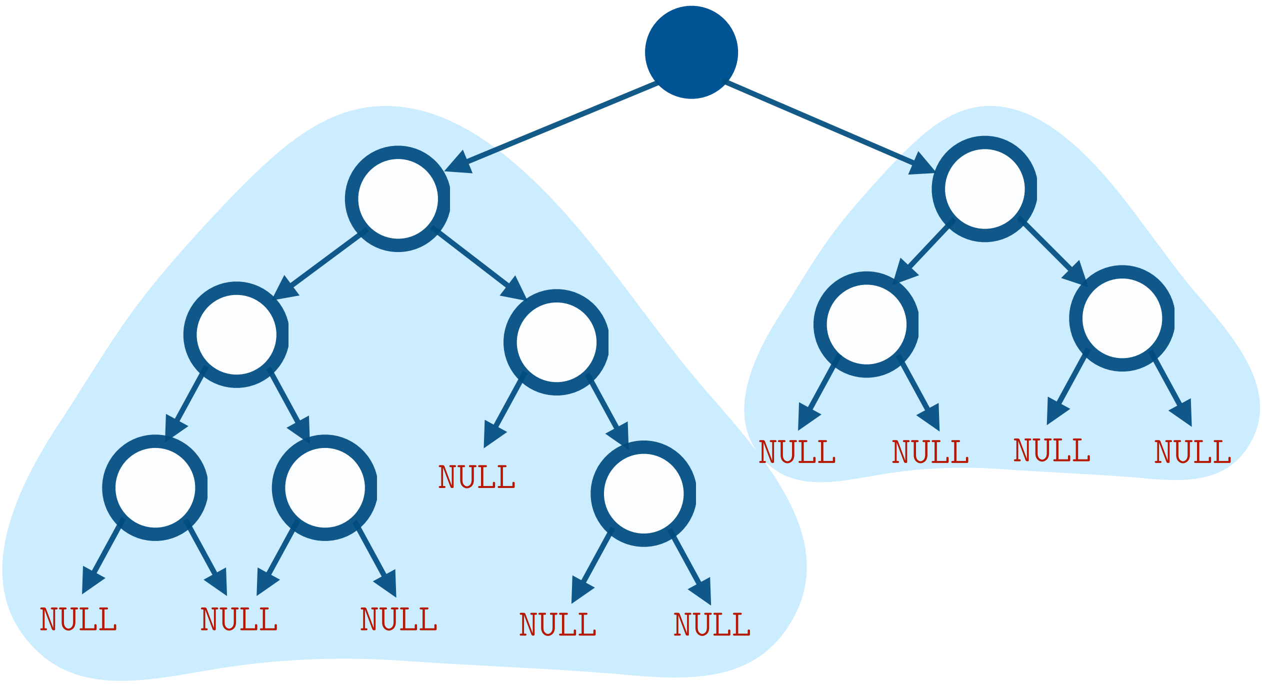
- `NULL` or
- a node connected to a set of *disjoint* trees.



Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

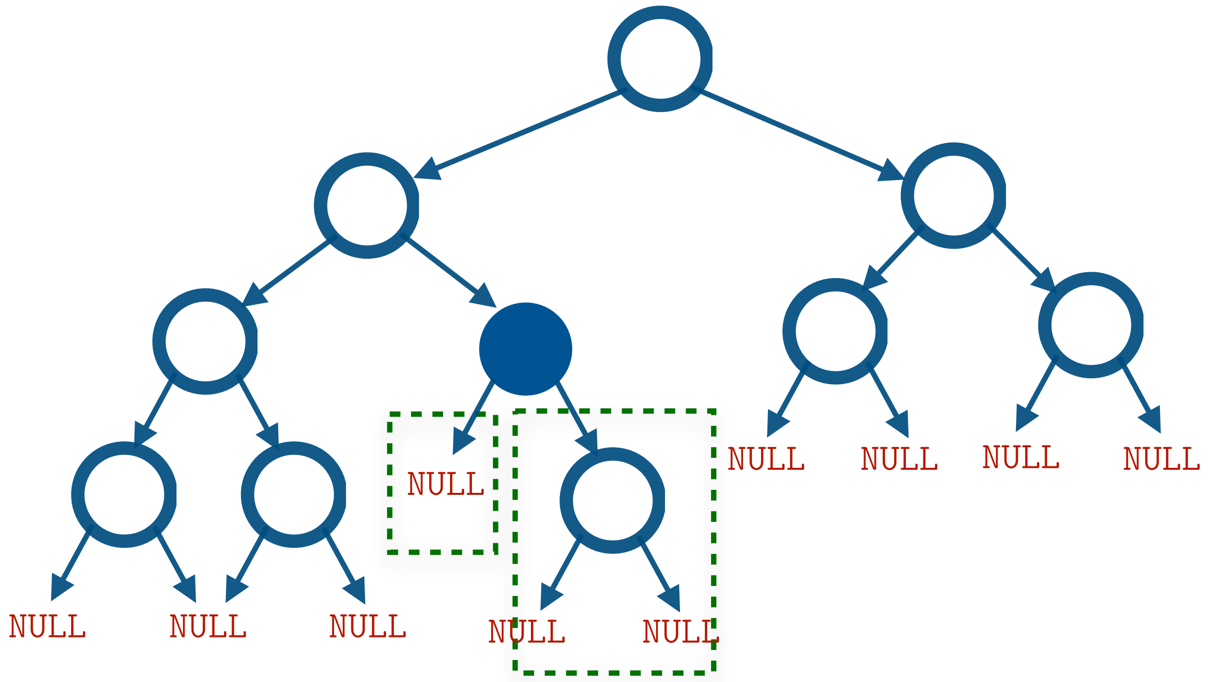


A node connected to two disjoint trees

Terminology

Definition. A *tree* is:

- NULL or
- a node connected to a set of *disjoint* trees.



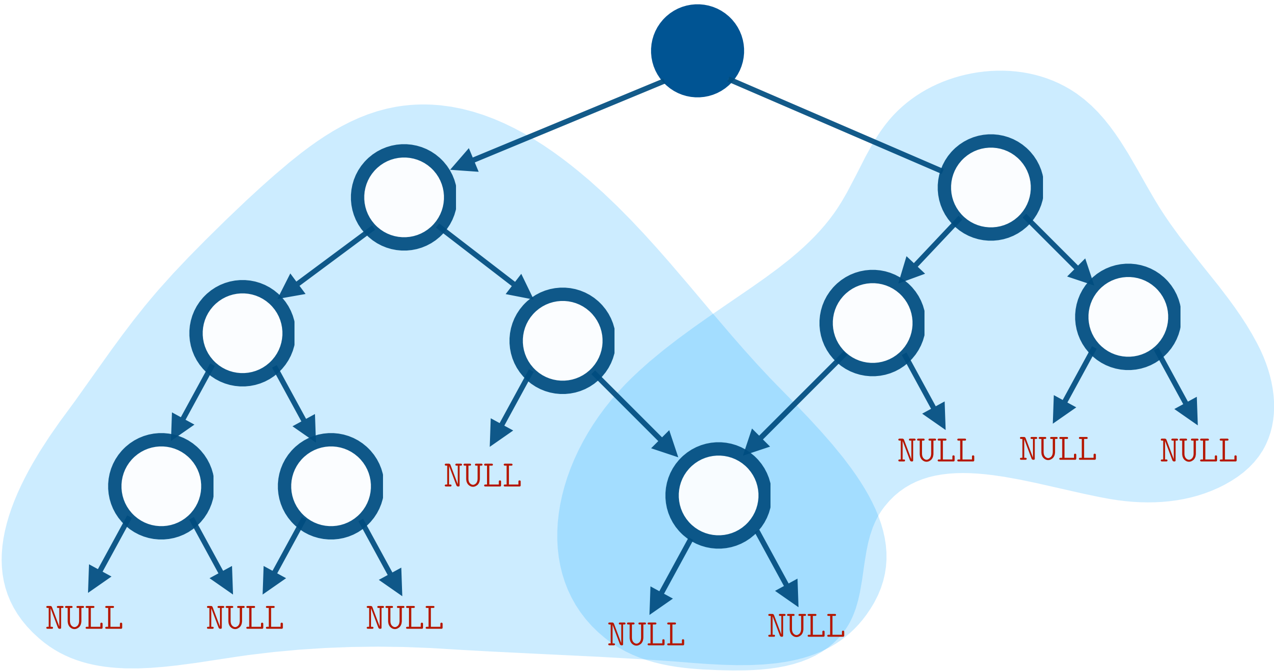
A node connected to two disjoint trees:

- A NULL tree and
- A node connected to two NULL trees

Terminology

Definition. A *tree* is:

- NULL or
- a node connected to a set of *disjoint* trees.

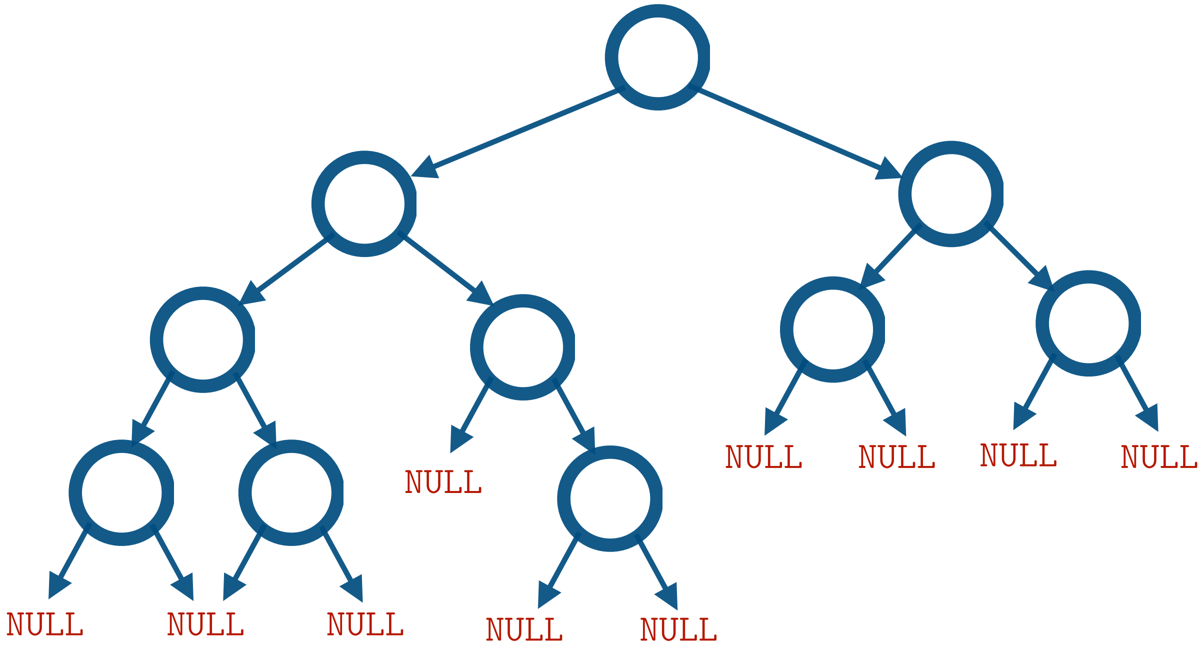


not a tree: A node connected to two overlapping trees

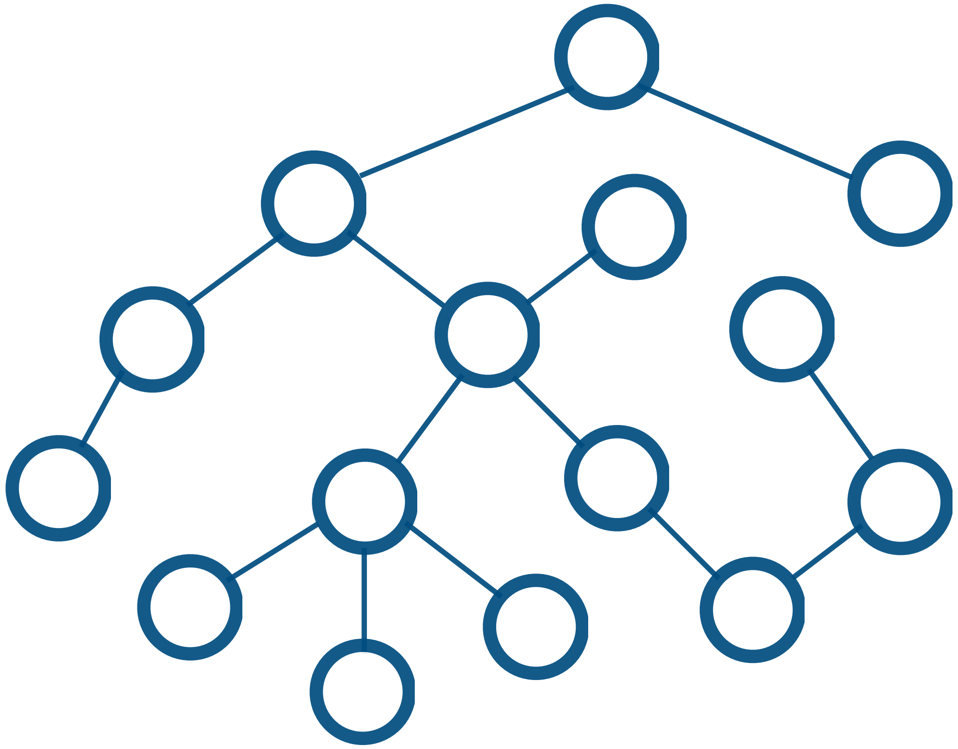
Terminology

Definition. A *tree* is:

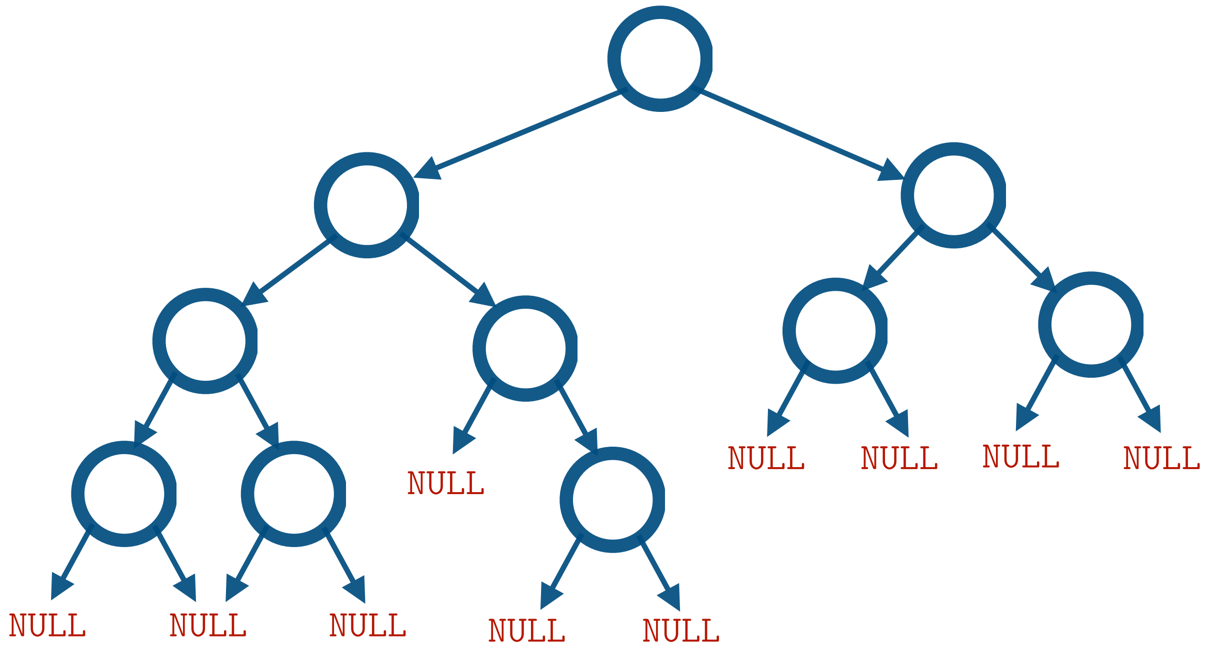
- `NULL` or
- a node connected to a set of *disjoint* trees.



Alternative definition (from graph theory).
A *tree* is a set of nodes in which any two nodes are connected by exactly one path



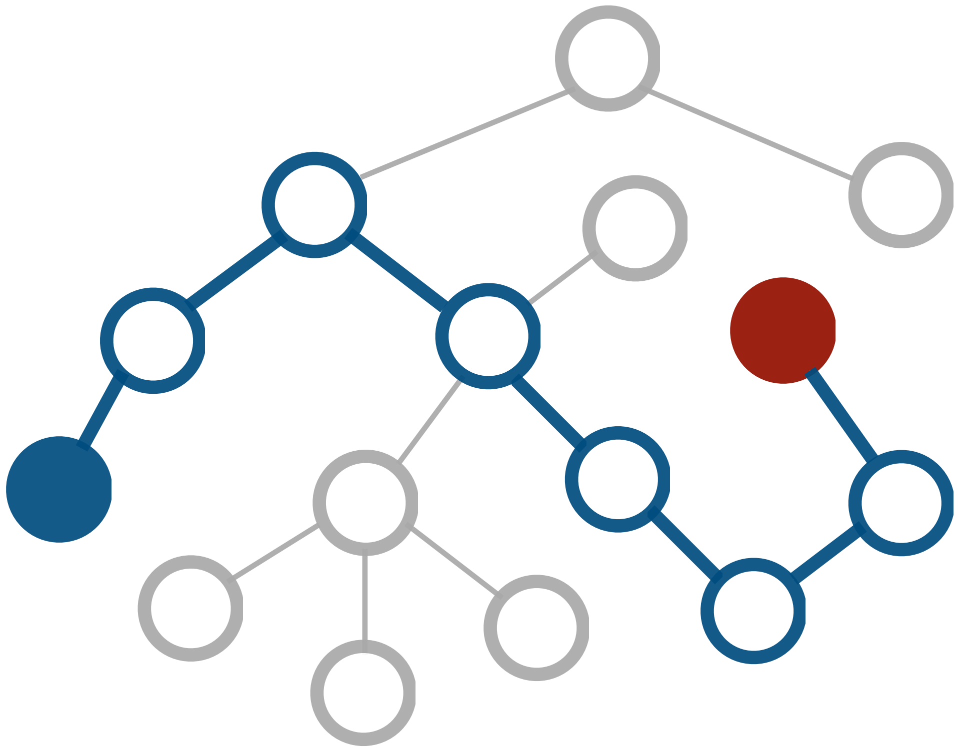
Terminology



Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

Alternative definition (from graph theory).
A *tree* is a set of nodes in which any two nodes are connected by exactly one path

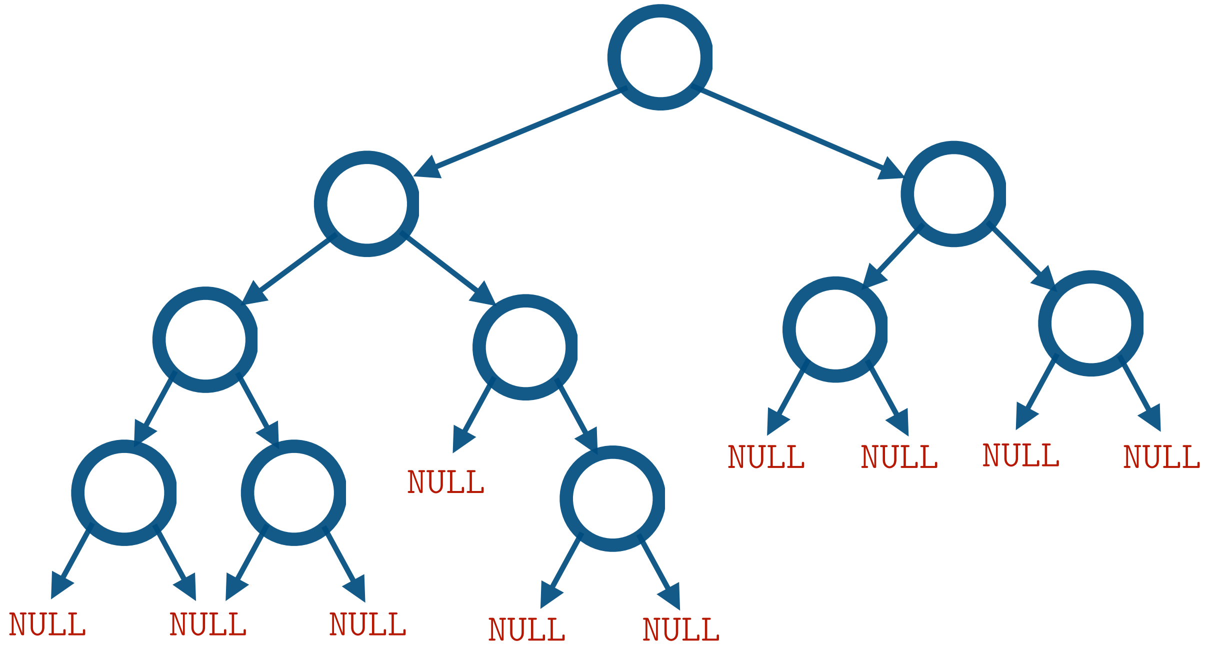


only one path from the red to the blue node

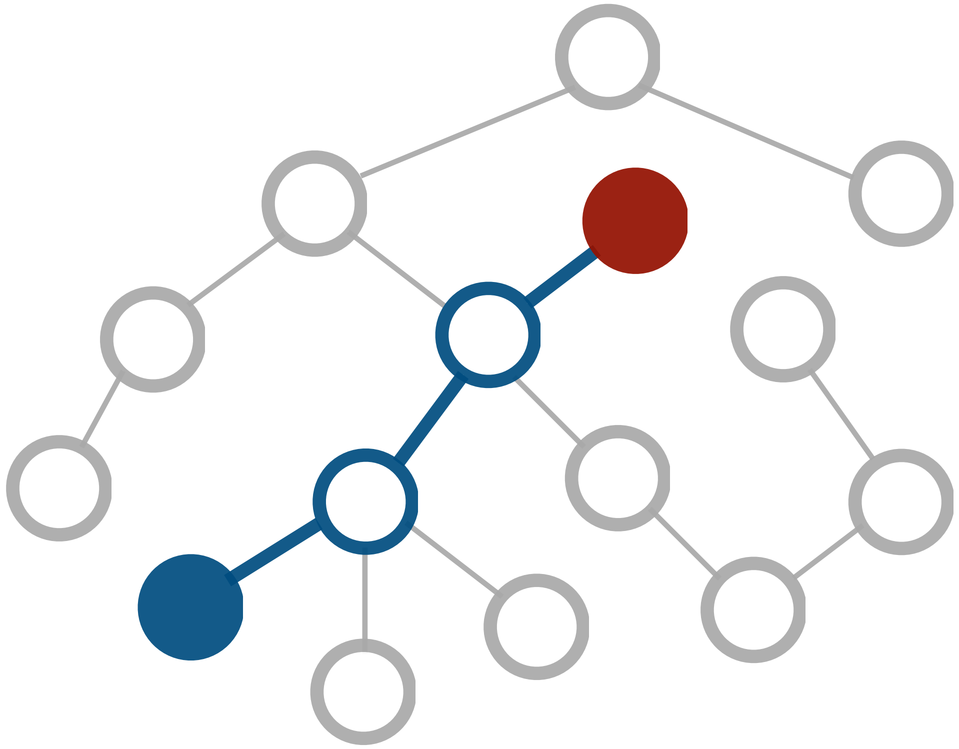
Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.



Alternative definition (from graph theory).
A *tree* is a set of nodes in which any two nodes are connected by exactly one path

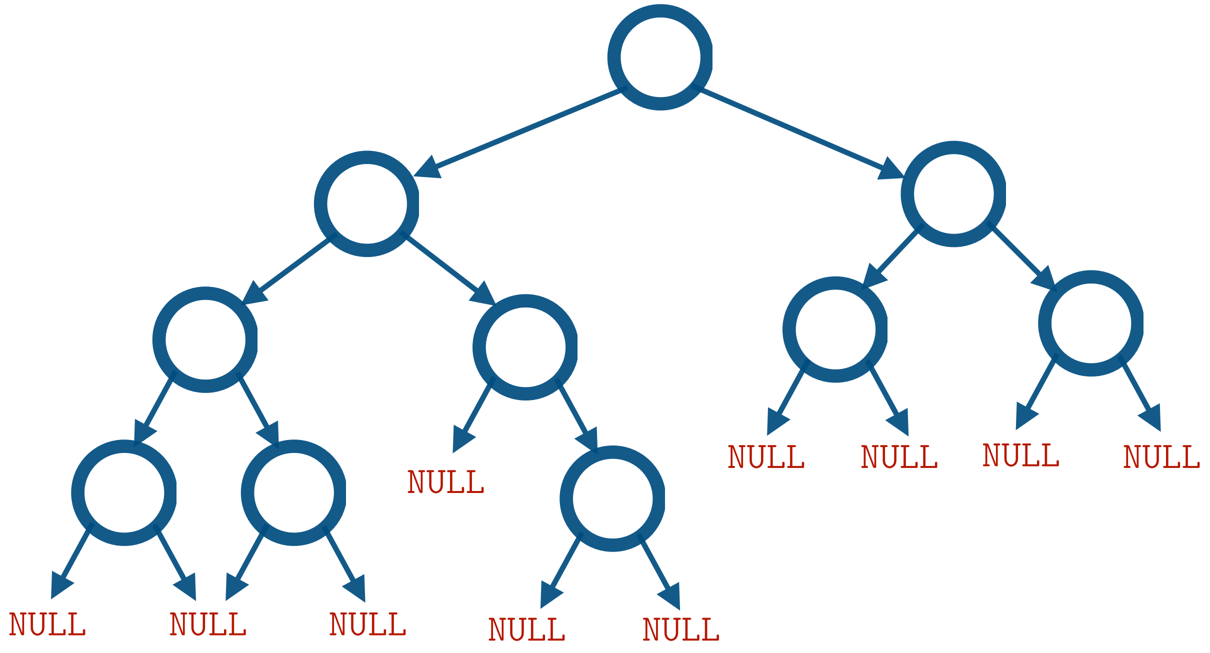


only one path from the red to the blue node

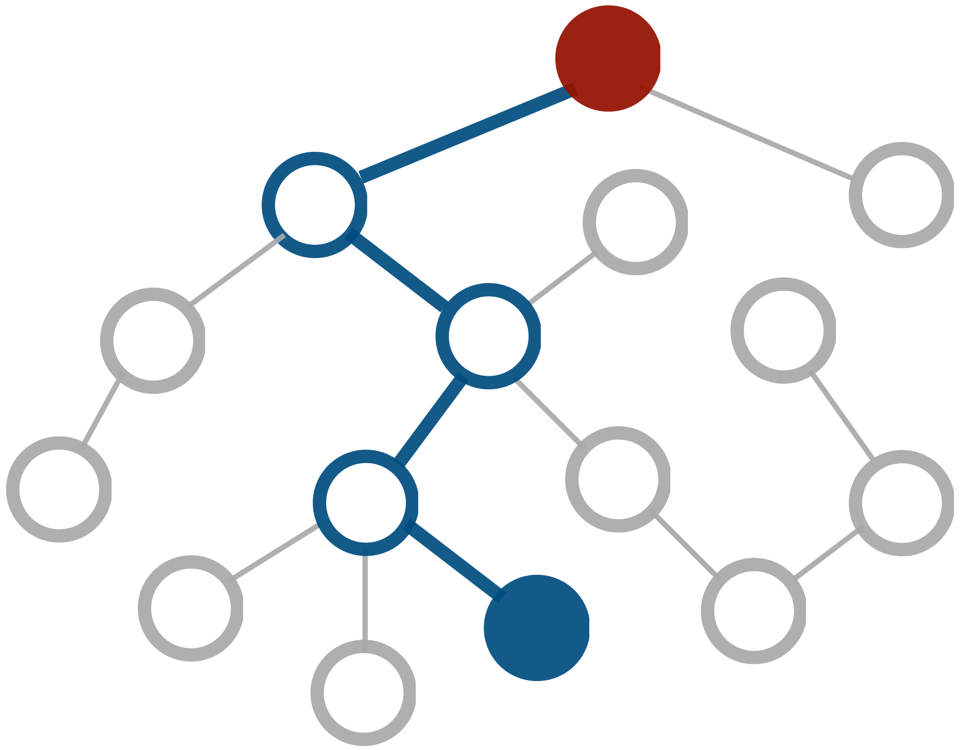
Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.



Alternative definition (from graph theory).
A *tree* is a set of nodes in which any two nodes are connected by exactly one path

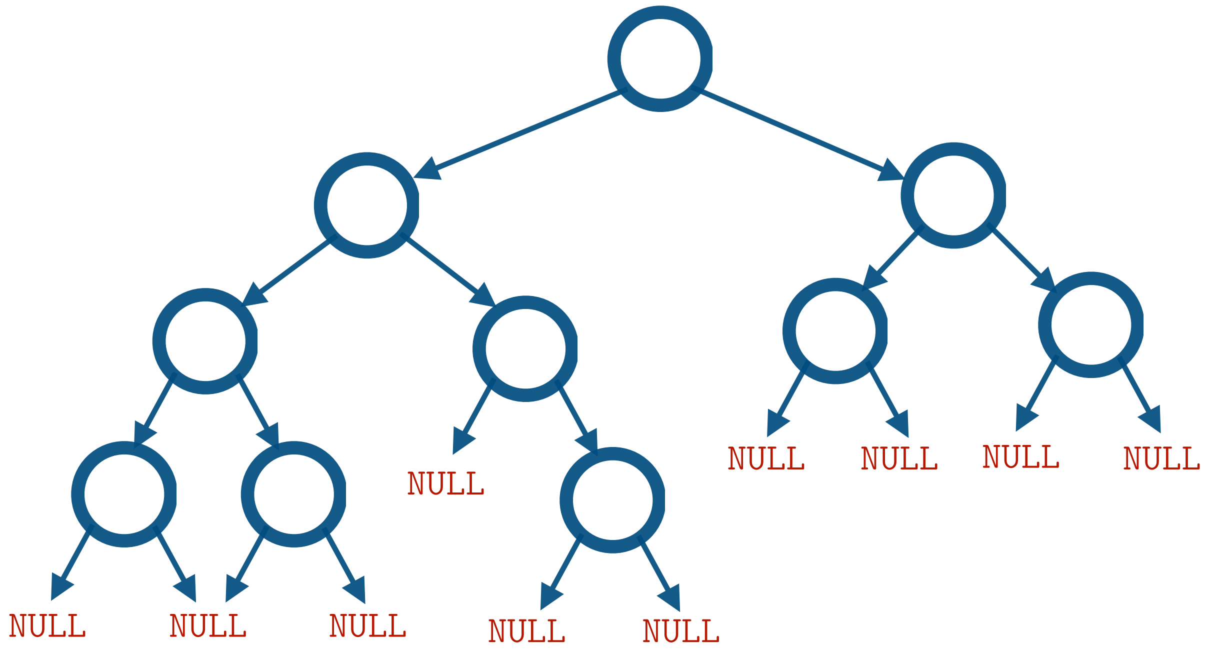


only one path from the red to the blue node

Terminology

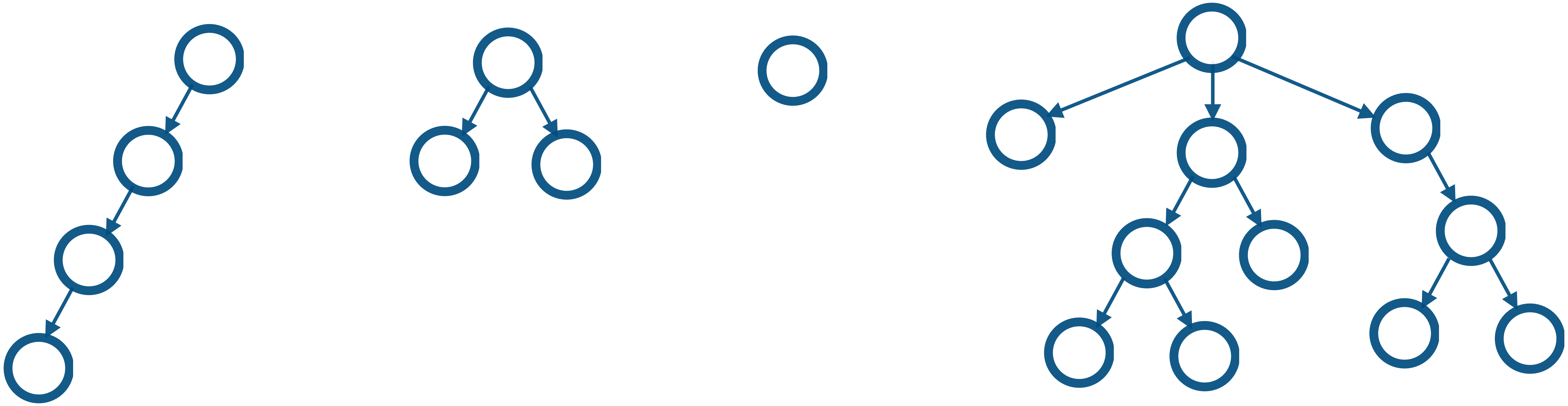
Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.



A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.



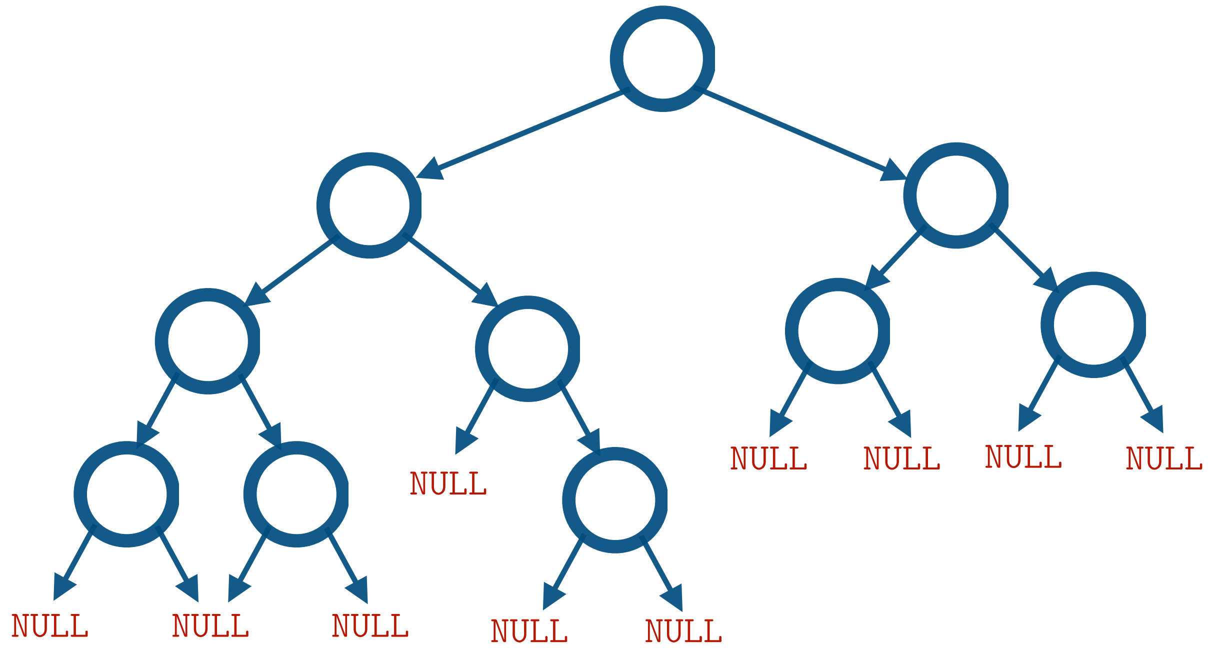
Terminology

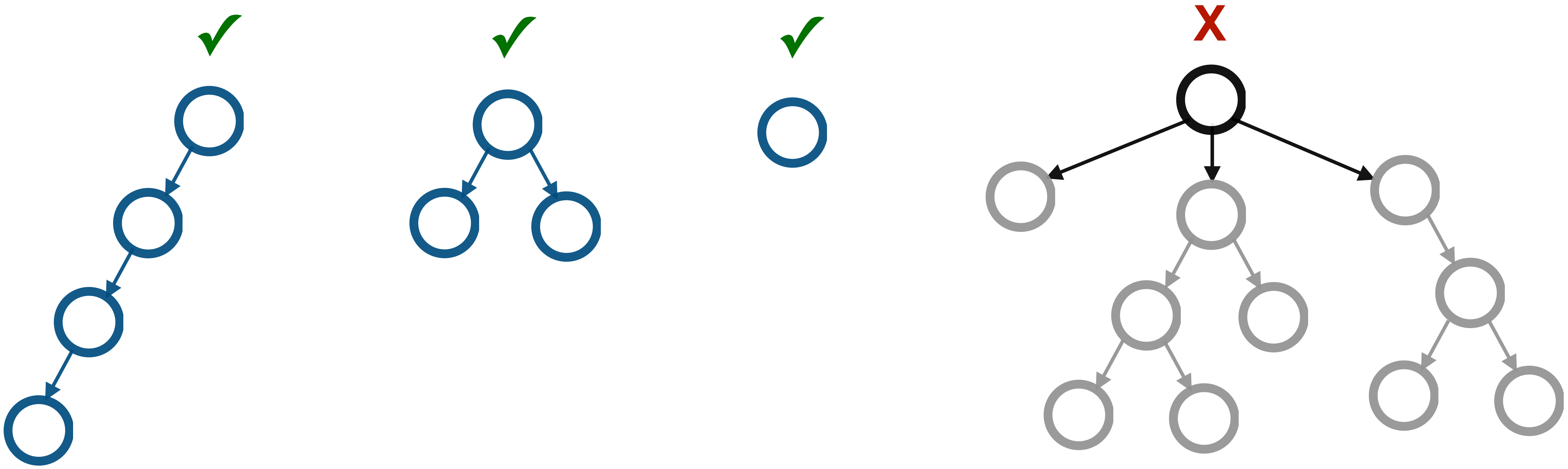
Definition. A *tree* is:

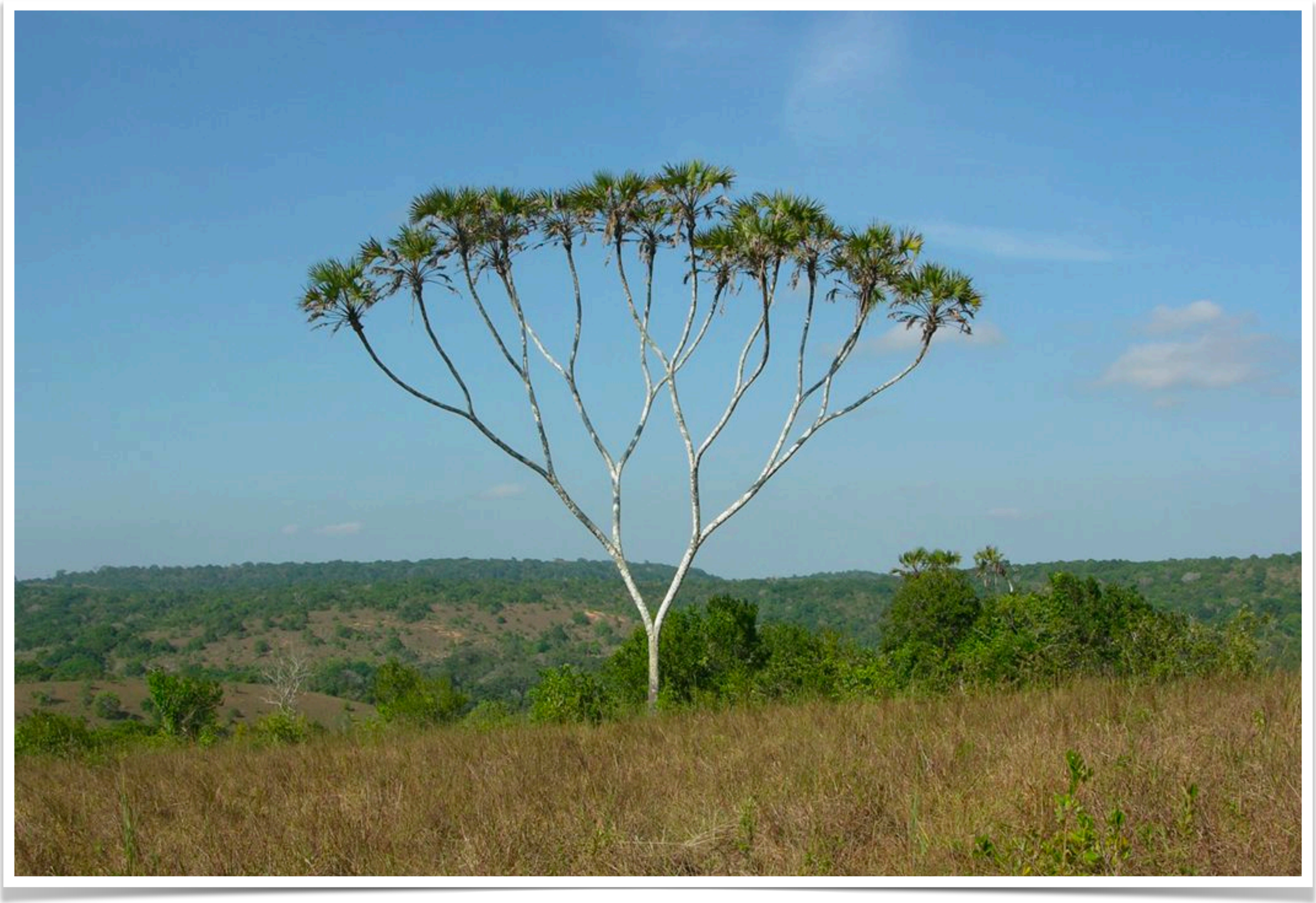
- `NULL` or
- a node connected to a set of *disjoint* trees.

A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.







A binary tree in nature!

Binary trees?

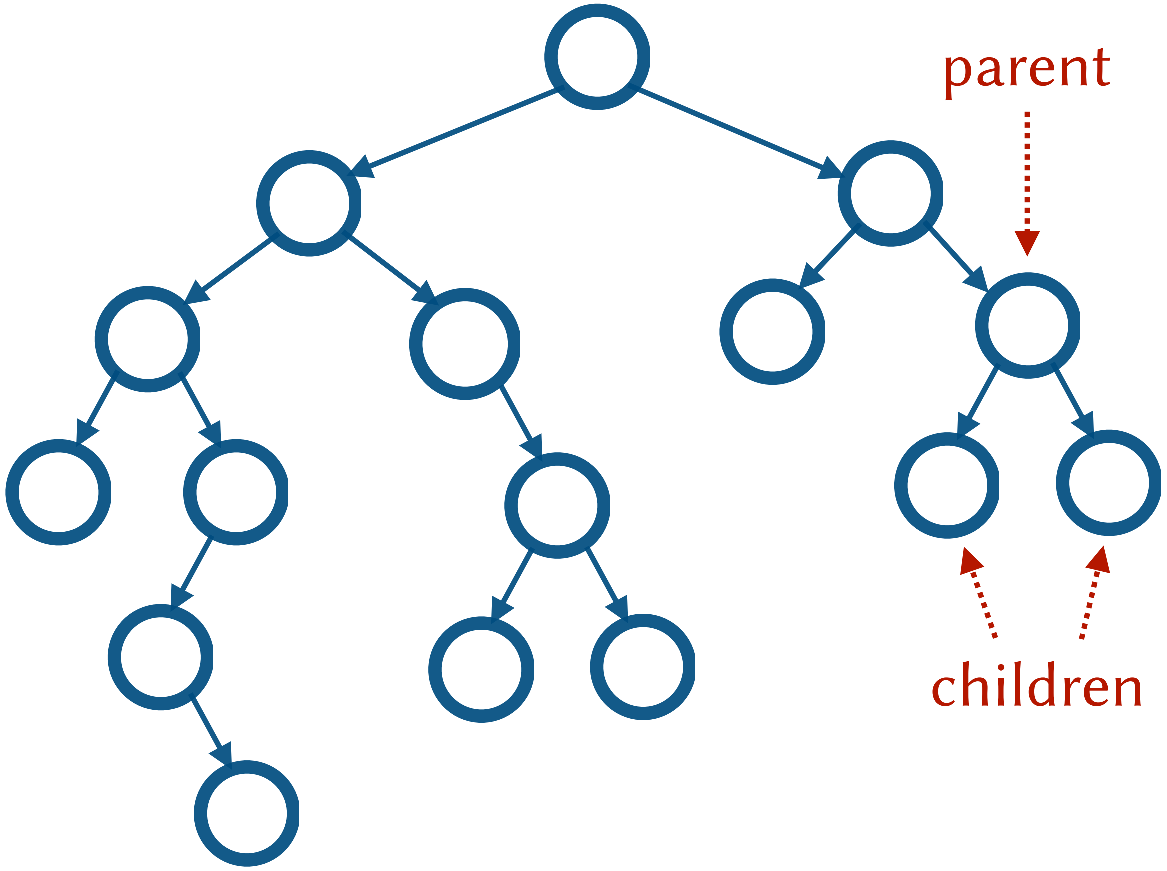
Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.



nodes that share the same parent are siblings

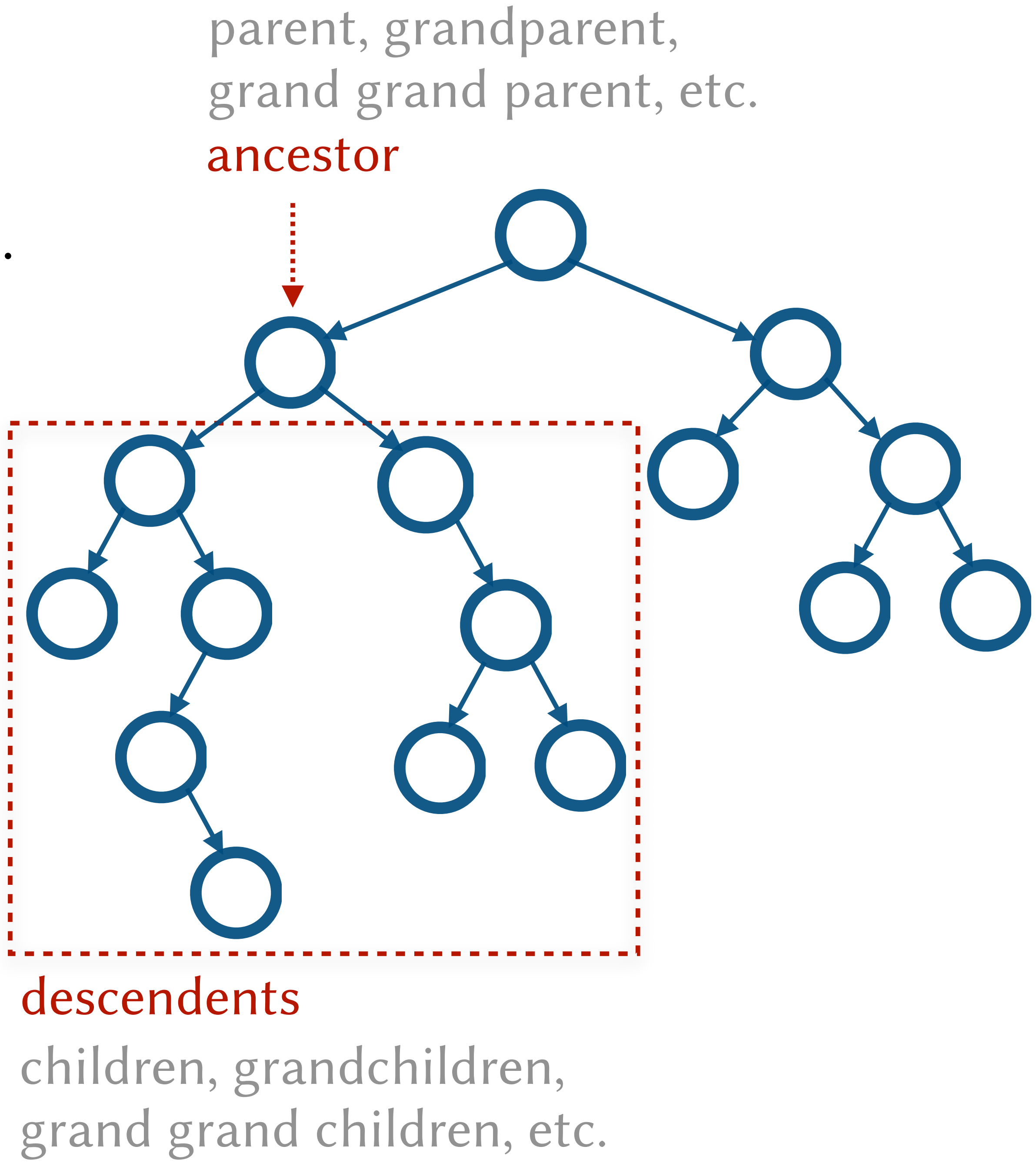
Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.



Terminology

Definition. A *tree* is:

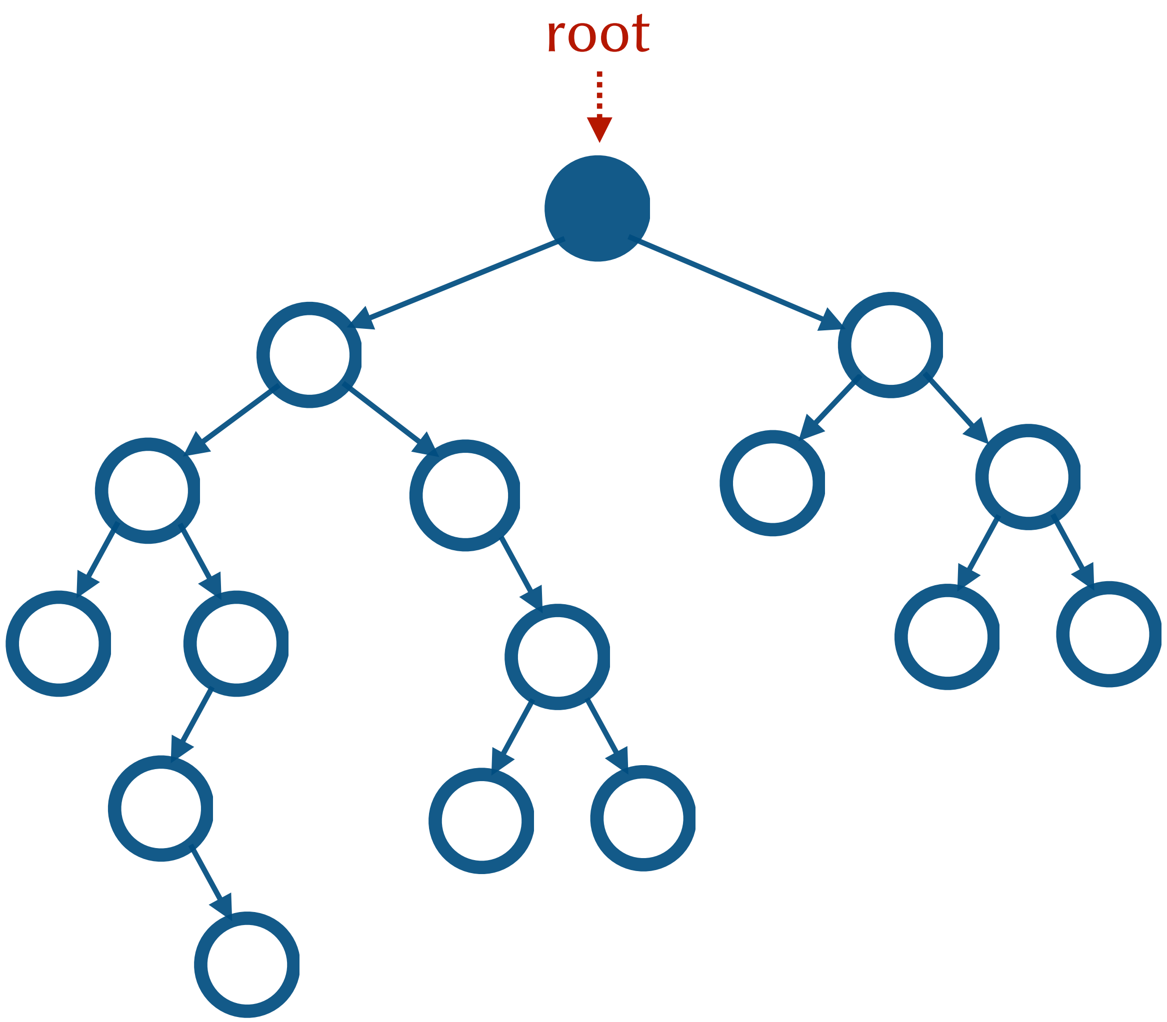
- `NULL` or
- a node connected to a set of *disjoint* trees.

A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.

Every tree has one root node.

The root node has no parent.
The root is an ancestor for all nodes.
All nodes are descendants of the root.



Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

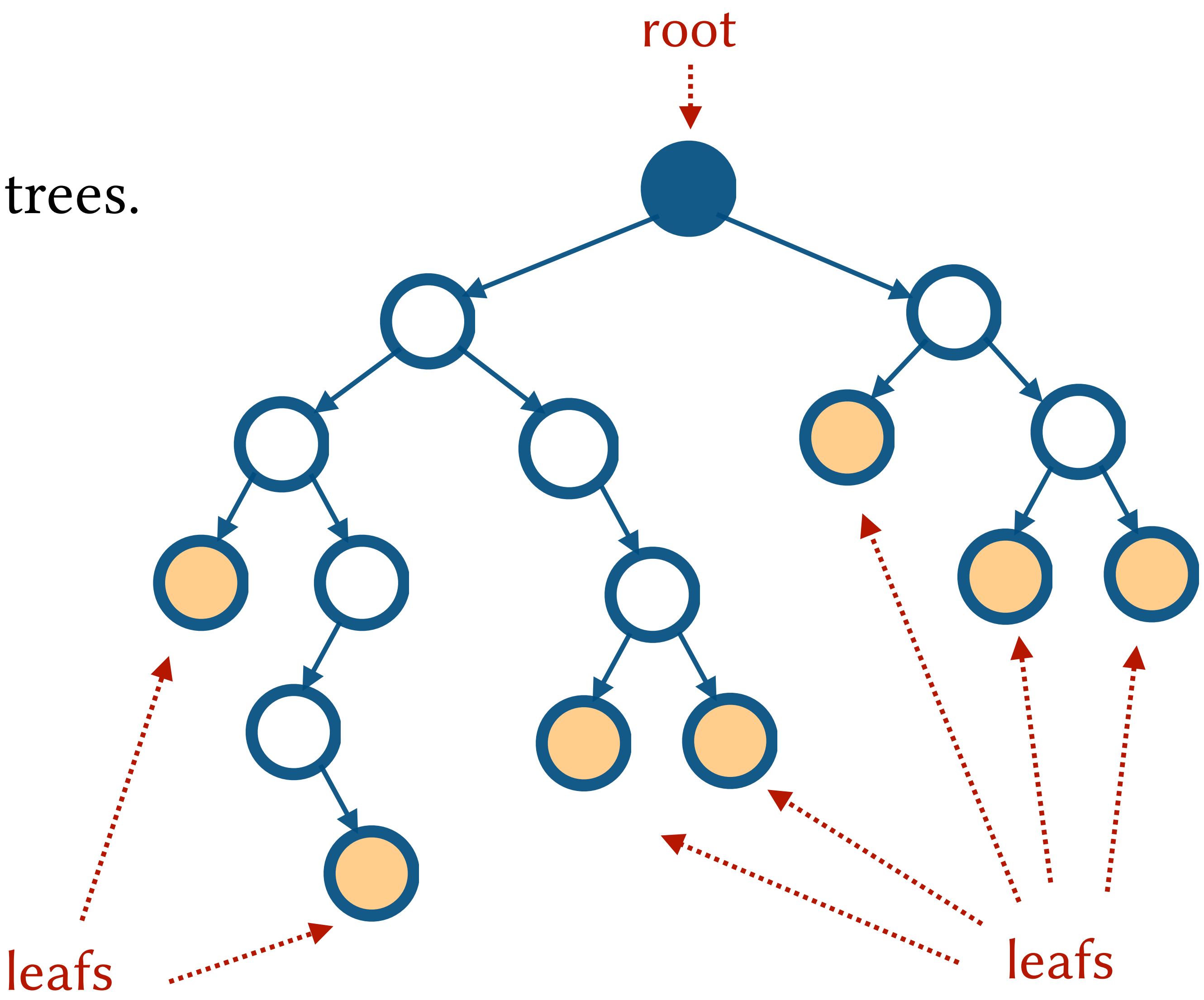
A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.

Every tree has one root node.

The root node has no parent.
The root is an ancestor for all nodes.
All nodes are descendants of the root.

A leaf node has no children.



Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

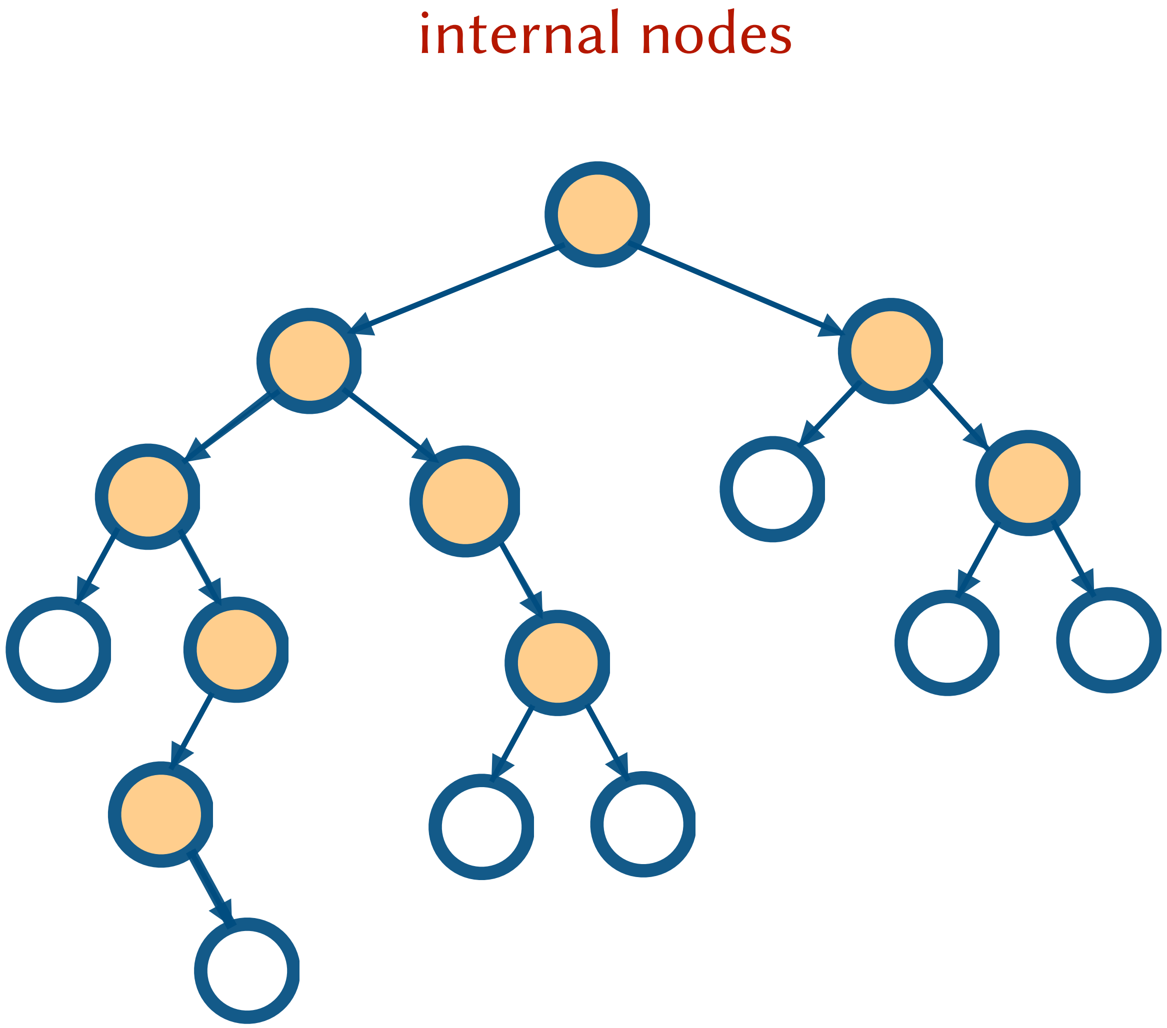
A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.

Every tree has one root node.
The root node has no parent.
The root is an ancestor for all nodes.
All nodes are descendants of the root.

A leaf node has no children.

Nodes that are not leafs are called internal nodes.
Leafs are also called "external nodes".



Terminology

Definition. A *tree* is:

- `NULL` or
- a node connected to a set of *disjoint* trees.

A binary tree is:

- `NULL` or
- a node connected to at most two disjoint binary trees.

Every tree has one root node.

The root node has no parent.
The root is an ancestor for all nodes.
All nodes are descendants of the root.

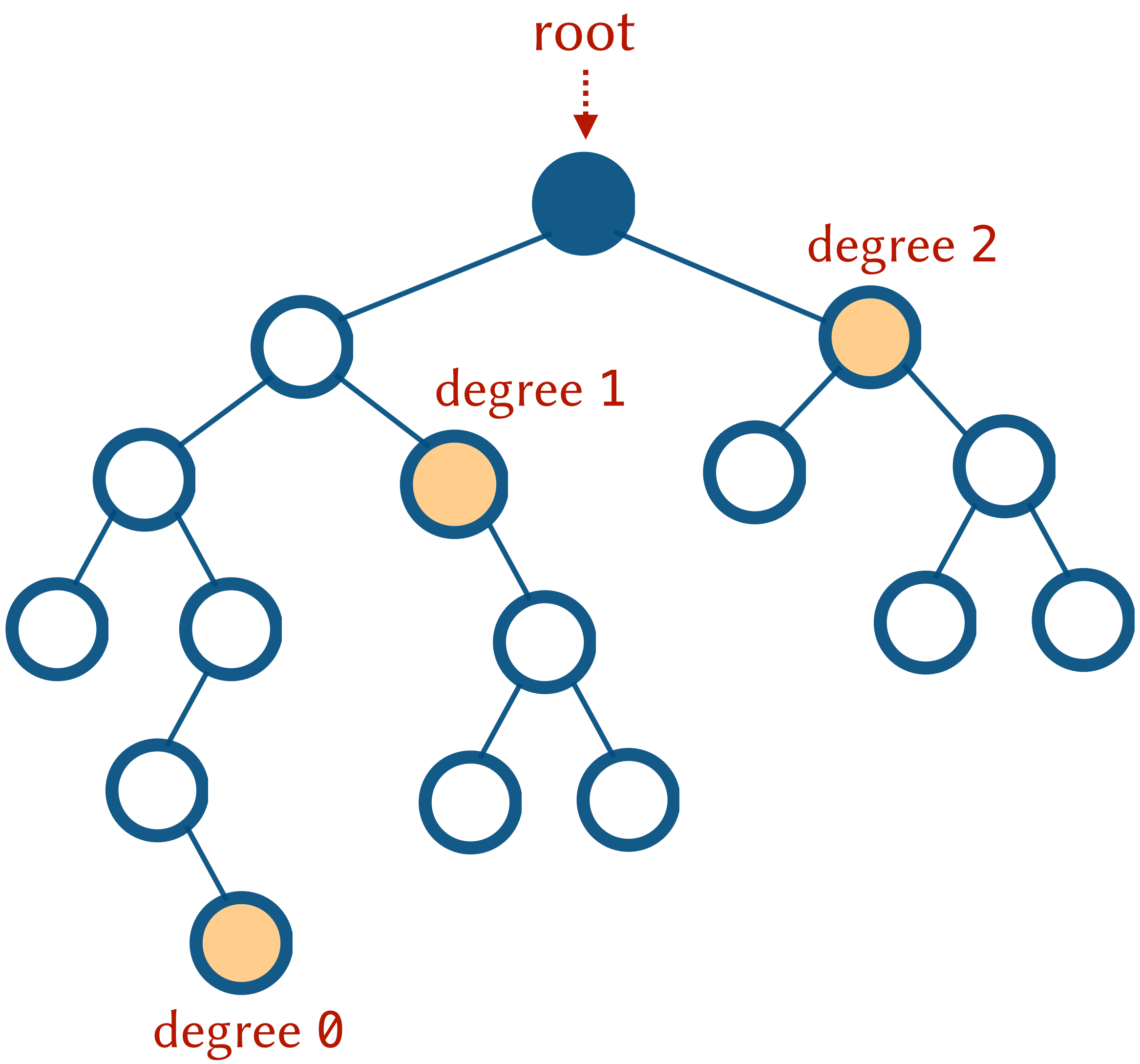
A leaf node has no children.

Nodes that are not leafs are called internal nodes.

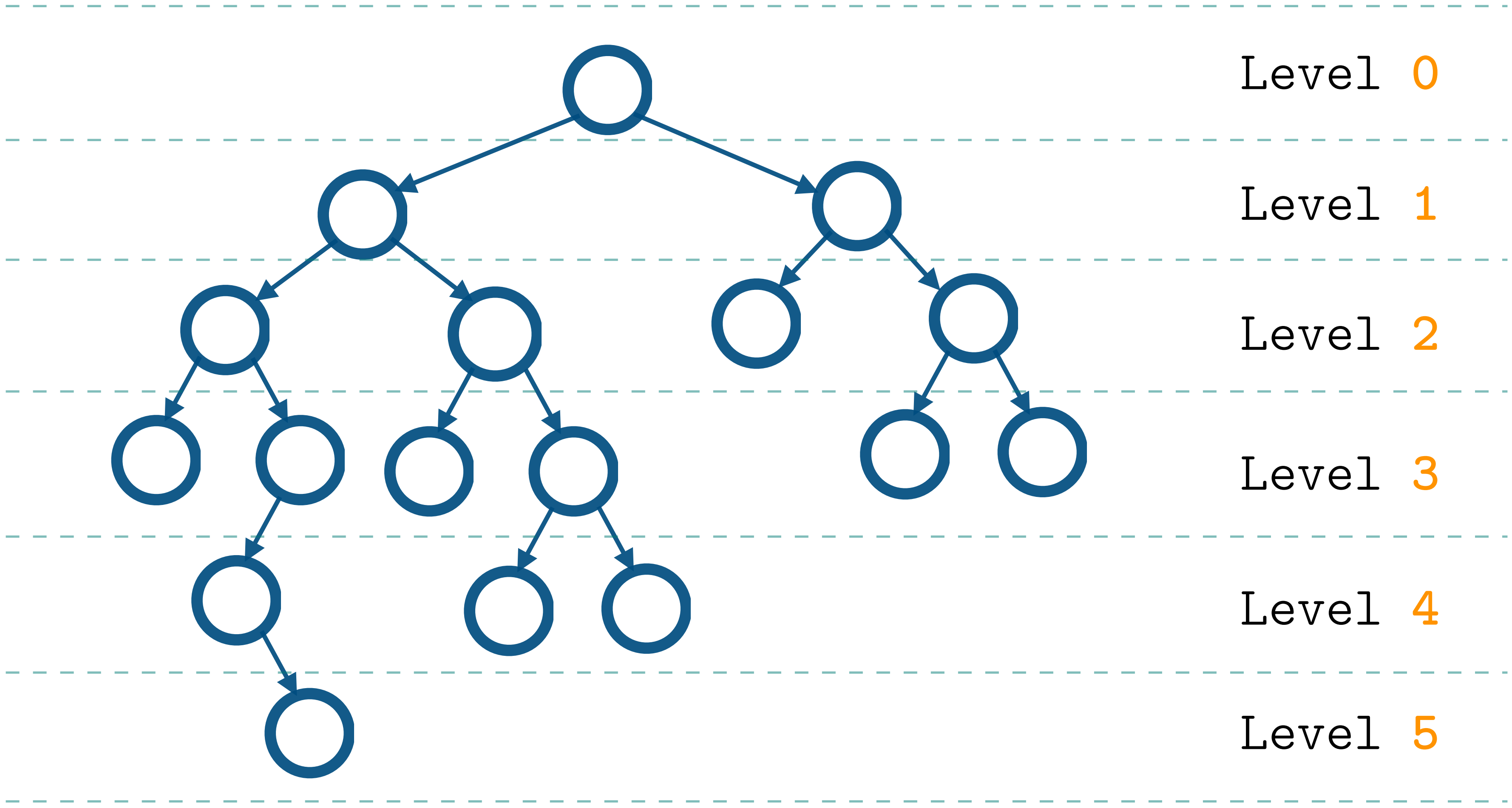
Leafs are also called "external nodes".

The degree of a node is the number of its children.
The degree of a tree is the maximum degree of a node in the tree.

A leaf has degree 0. The illustrated tree has degree 2.



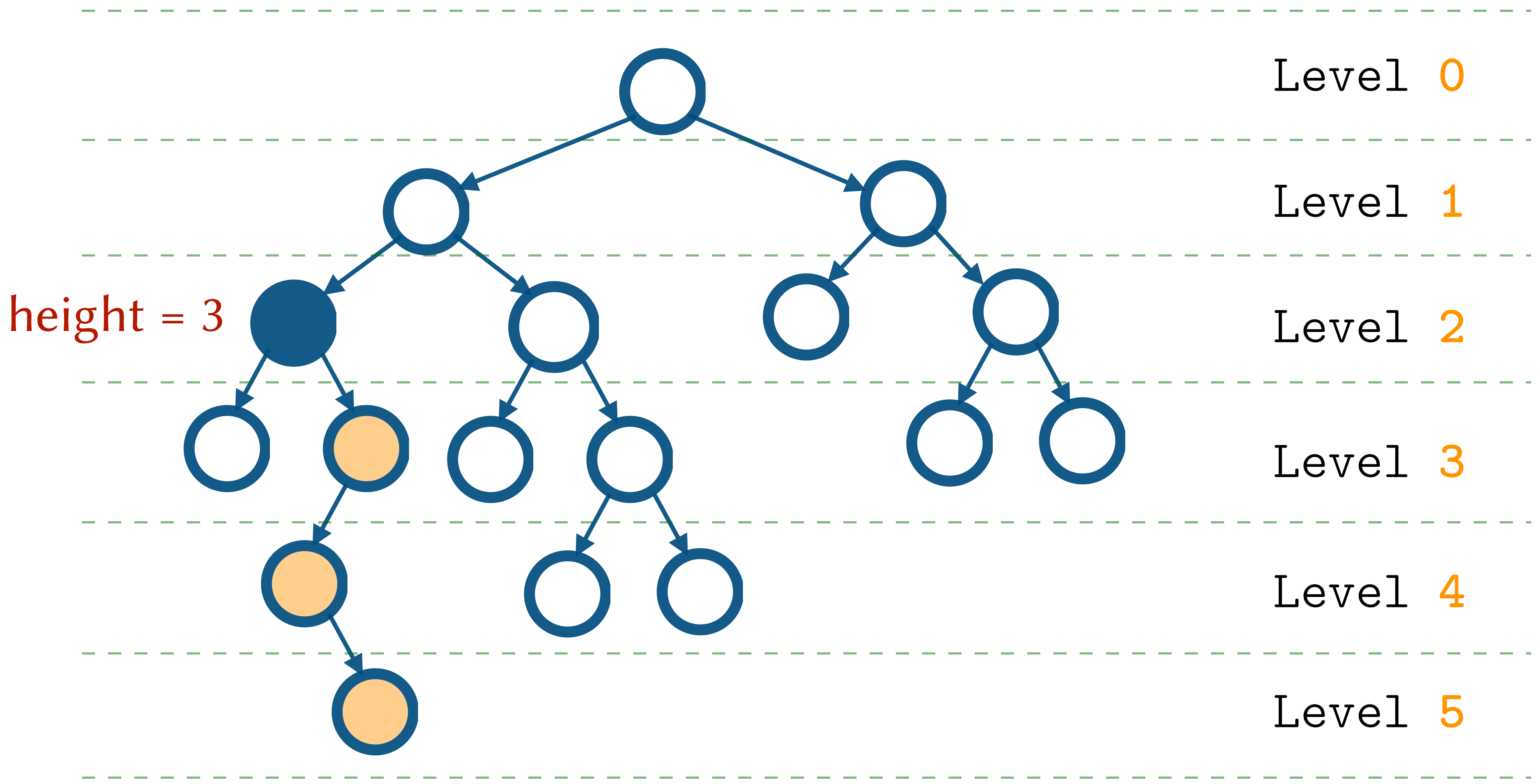
Terminology



The depth of a node is its level in the tree.

The root has depth 0.

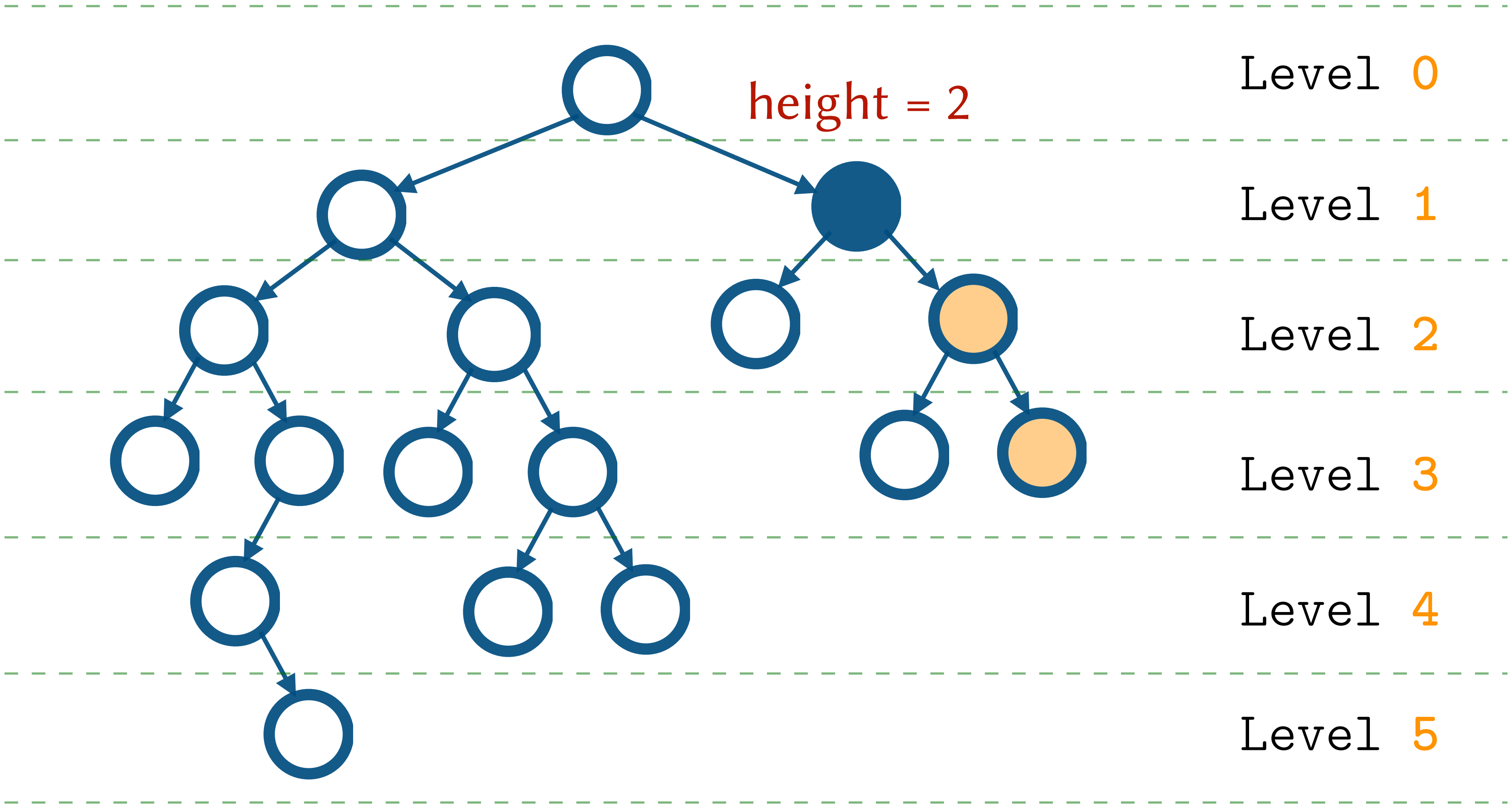
Terminology



The depth of a node is its level in the tree.
The root has depth 0.

The height of a node is the maximum number of levels below it.
All leafs have height 0.

Terminology



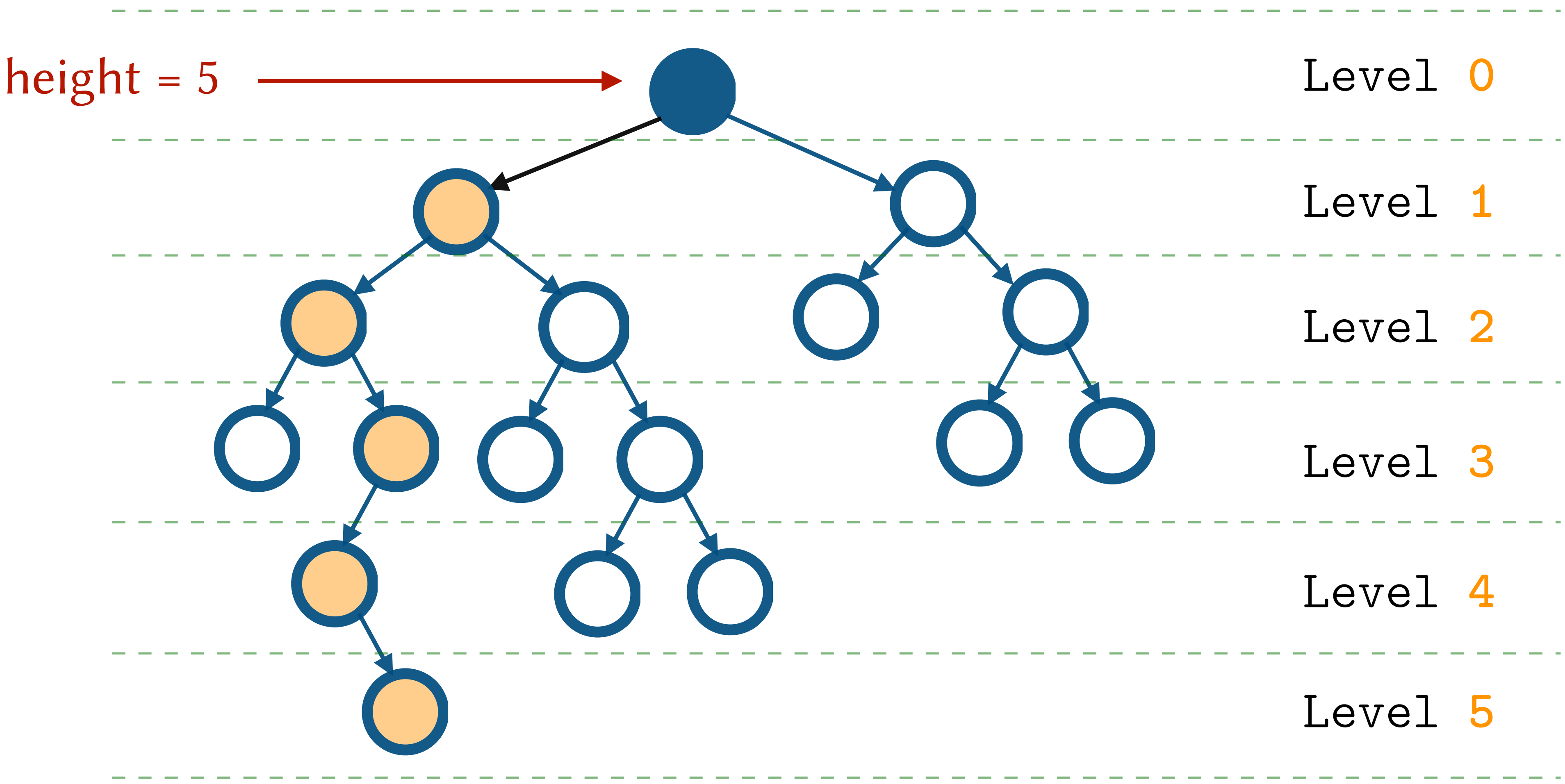
The depth of a node is its level in the tree.

The root has depth 0.

The height of a node is the maximum number of levels below it.

All leafs have height 0.

Terminology



The depth of a node is its level in the tree.

The root has depth 0.

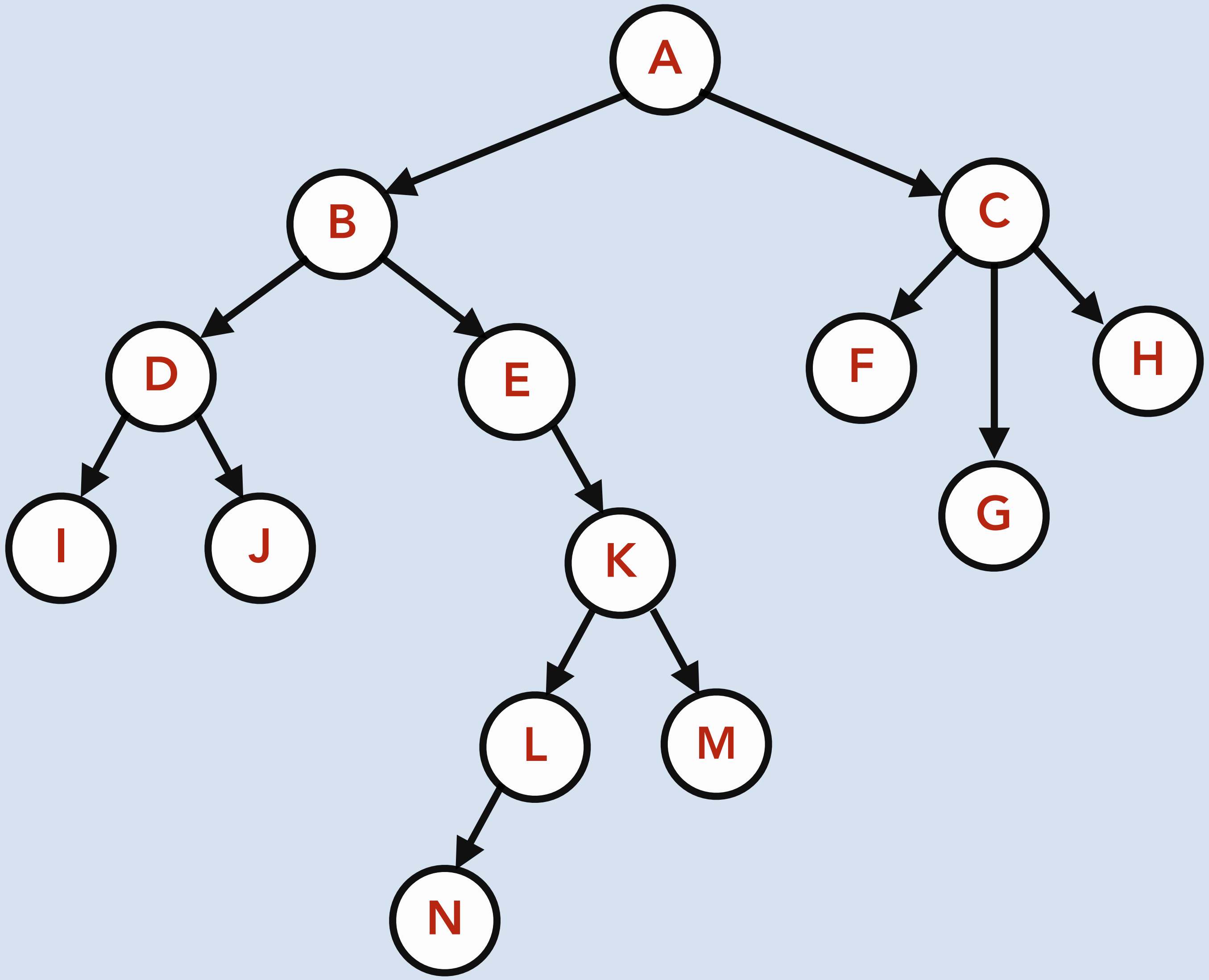
The height of a node is the maximum number of levels below it.

All leafs have height 0.

The height of a tree is the height of the root.

Also called "tree depth".

Exercise



Root =

Leafs =

Tree height =

Tree Degree =

Ancestors of **L** =

Height of **E** =

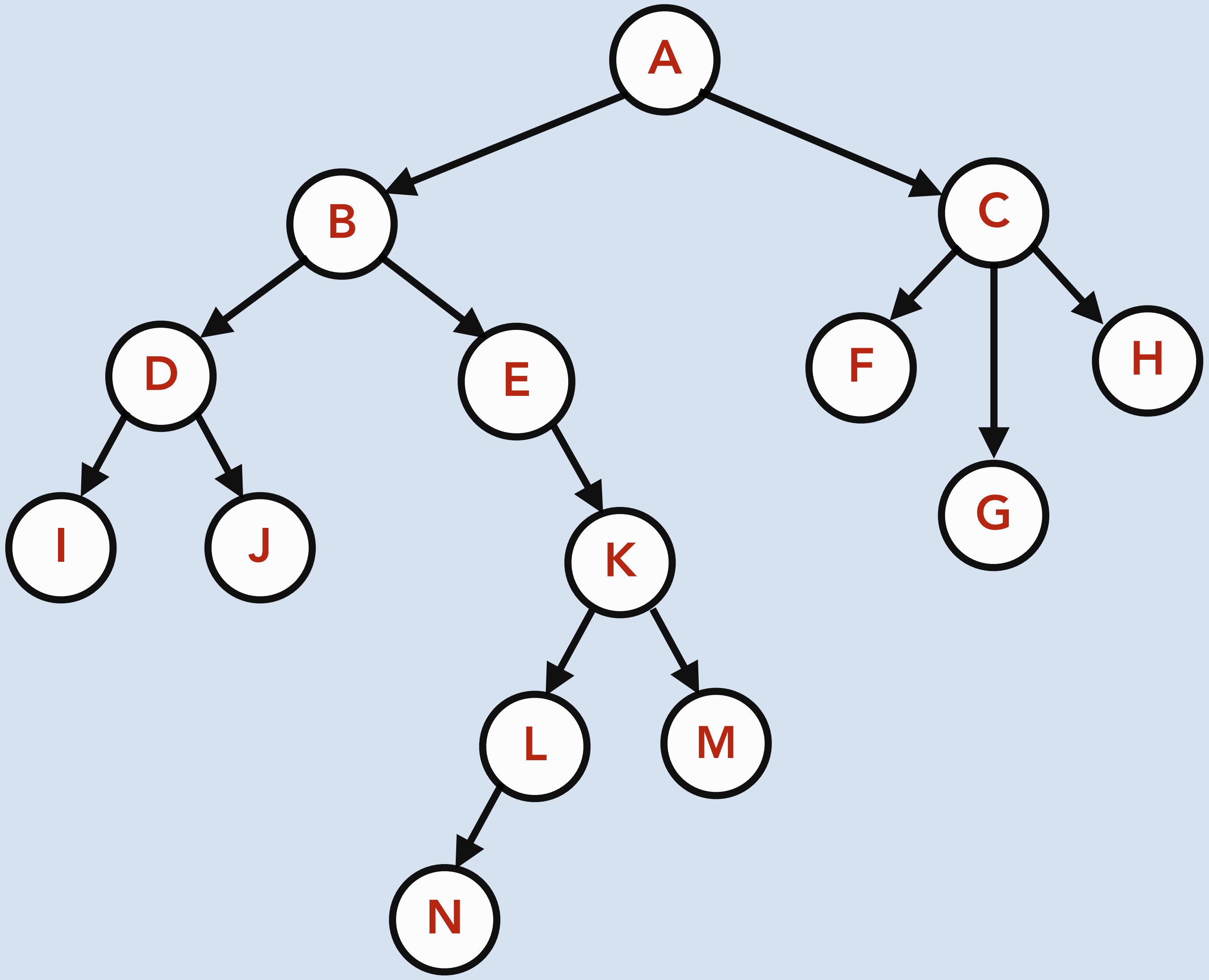
Depth of **E** =

Degree of **E** =

Descendants of **E** =

Siblings of **G** =

Exercise



Root = **A**

Leafs = **I J N M F G H**

Tree height = **5**

Tree Degree = **3**

Ancestors of **L** = **K E B A**

Height of **E** = **3**

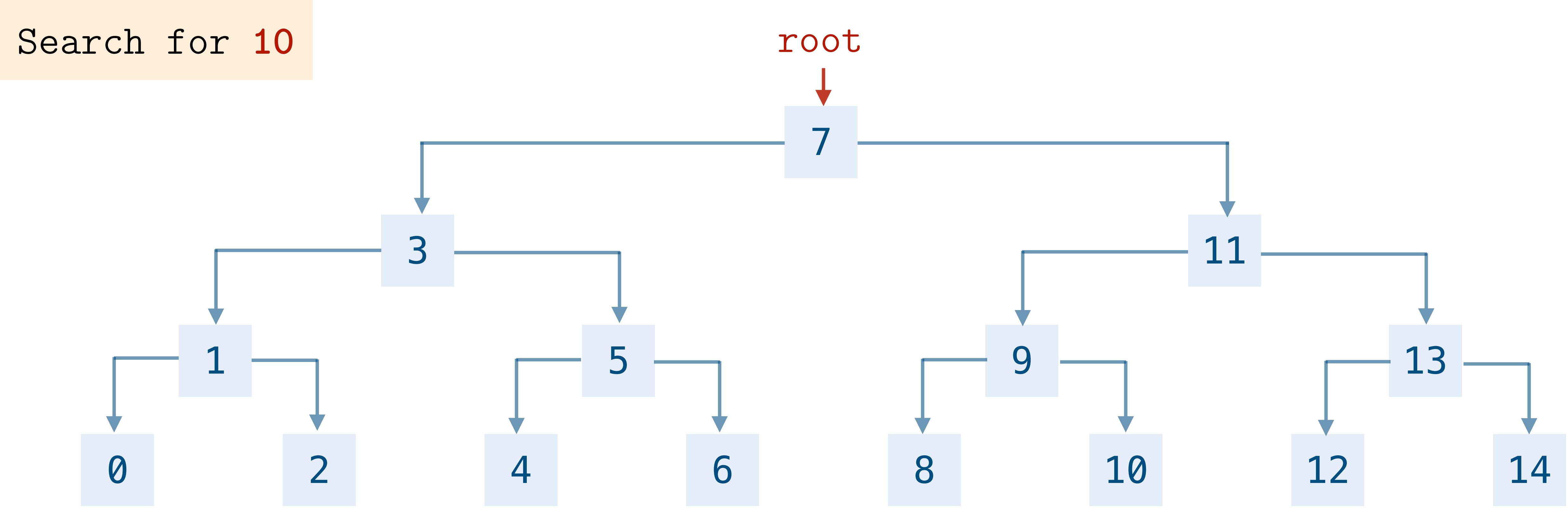
Depth of **E** = **2**

Degree of **E** = **1**

Descendants of **E** = **K L M N**

Siblings of **G** = **F H**

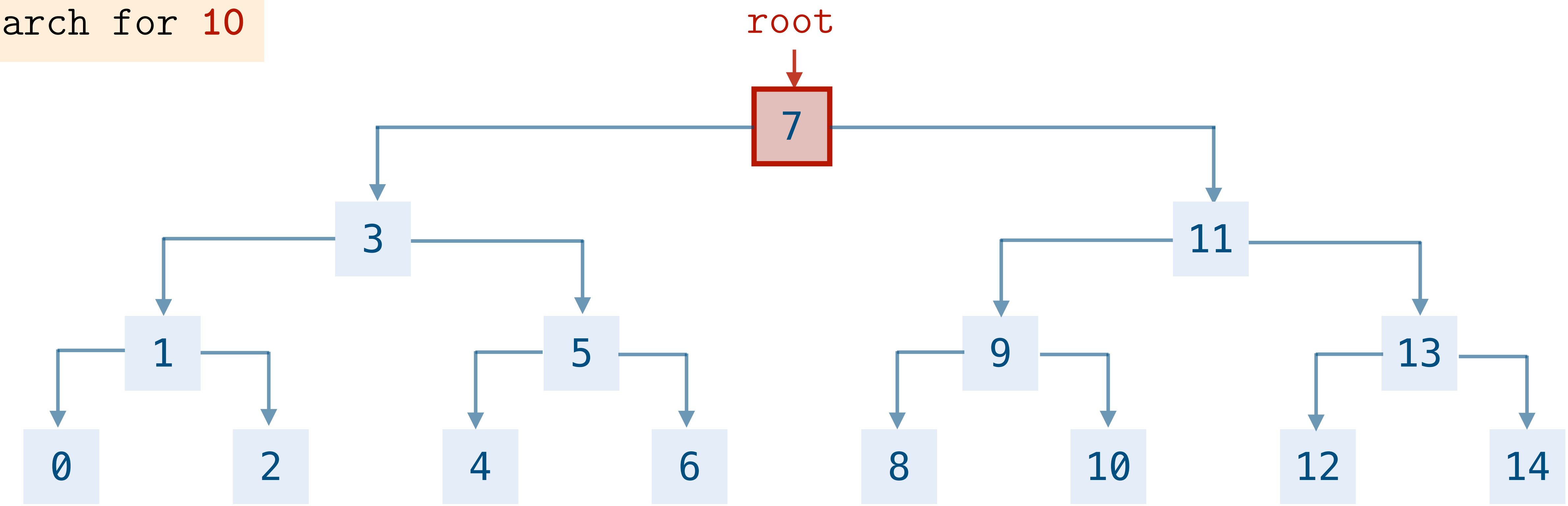
Binary Search Trees



Informally. A binary search tree (BST) is a binary tree that allows performing binary search!

Binary Search Trees

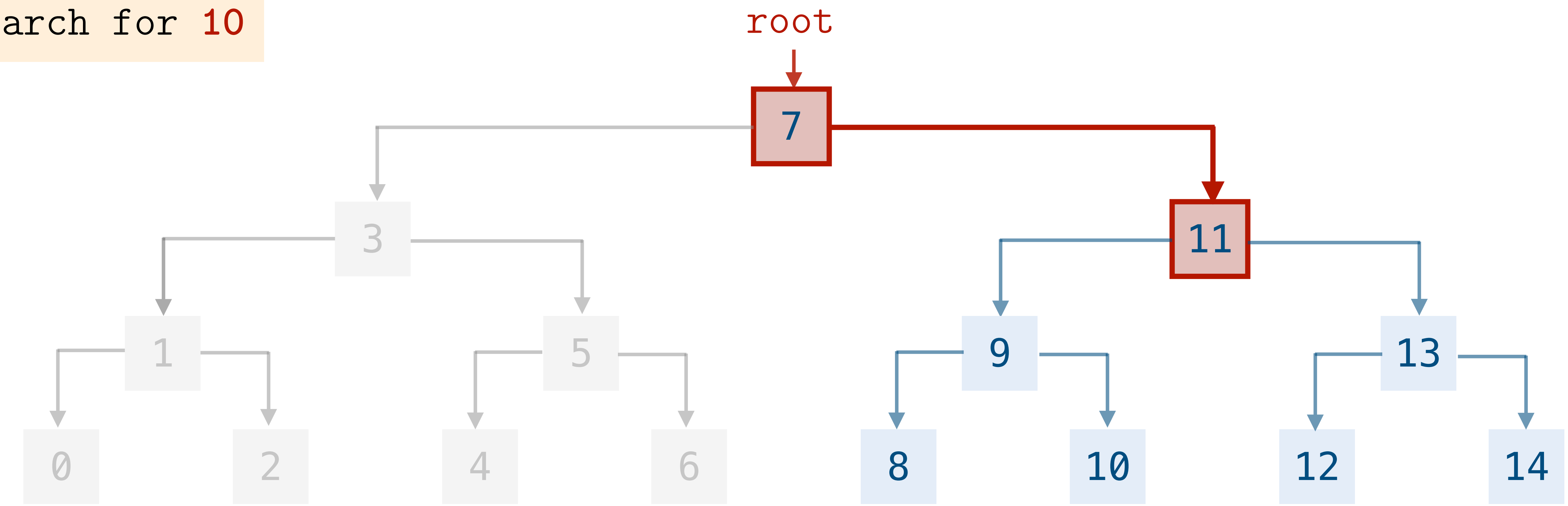
`Search for 10`



Informally. A binary search tree (BST) is a binary tree that allows performing binary search!

Binary Search Trees

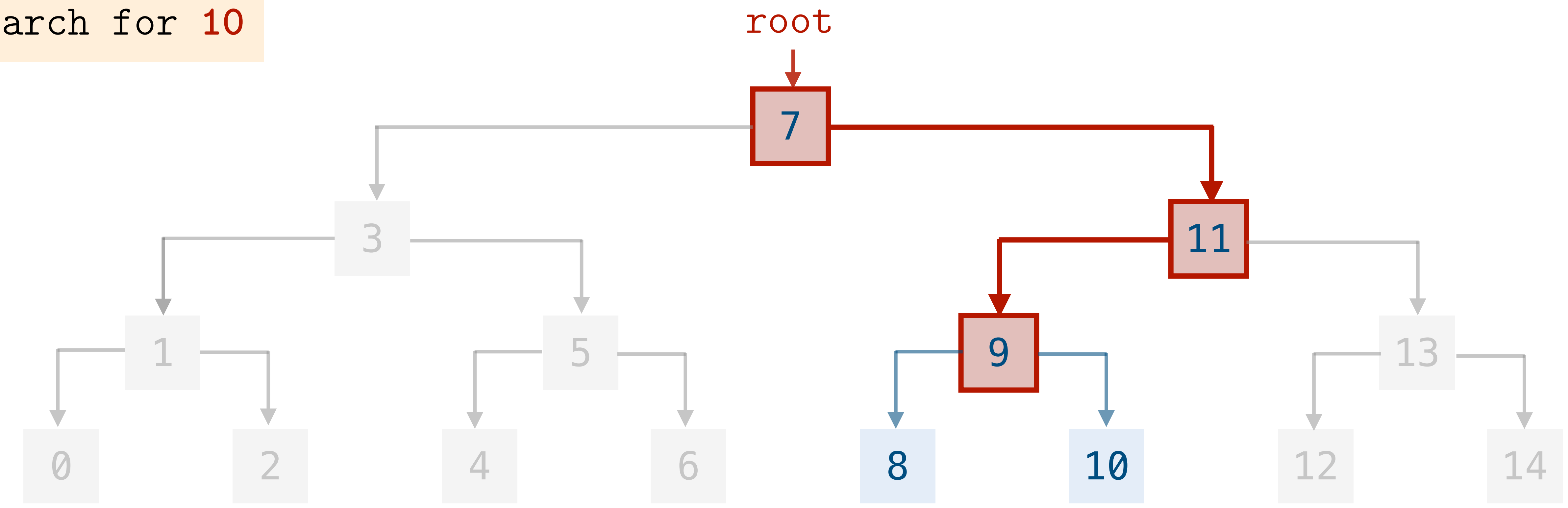
`Search for 10`



Informally. A binary search tree (BST) is a binary tree that allows performing binary search!

Binary Search Trees

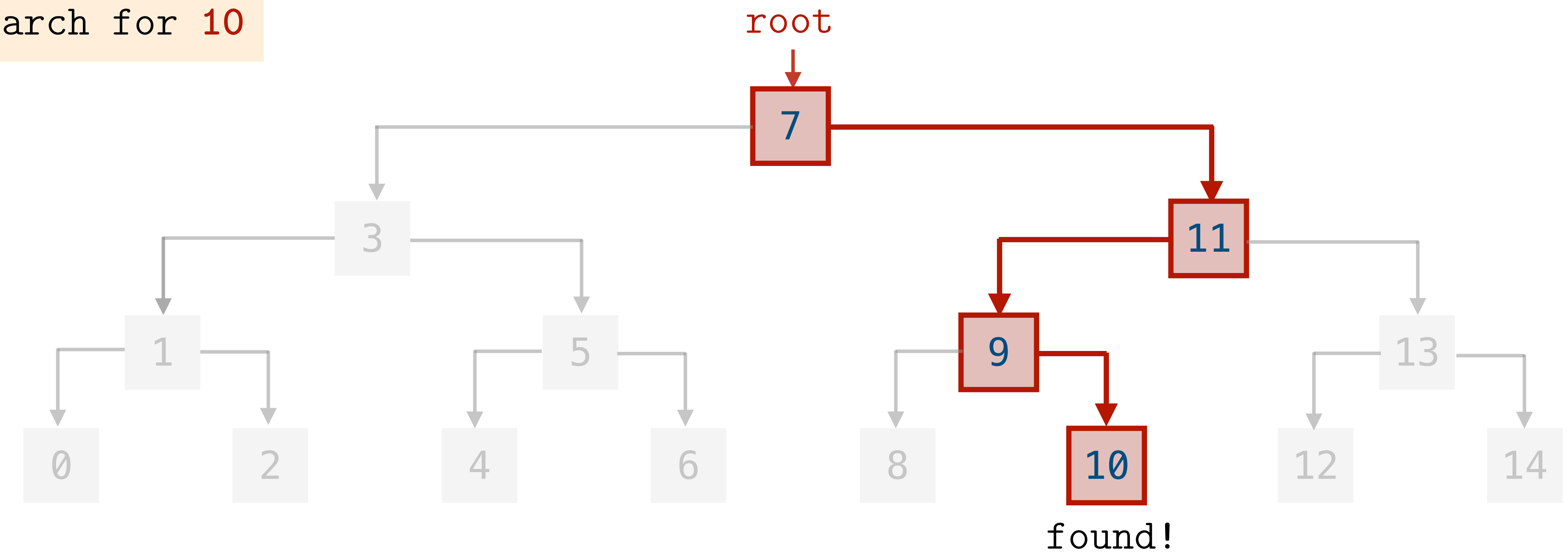
`Search for 10`



Informally. A binary search tree (BST) is a binary tree that allows performing binary search!

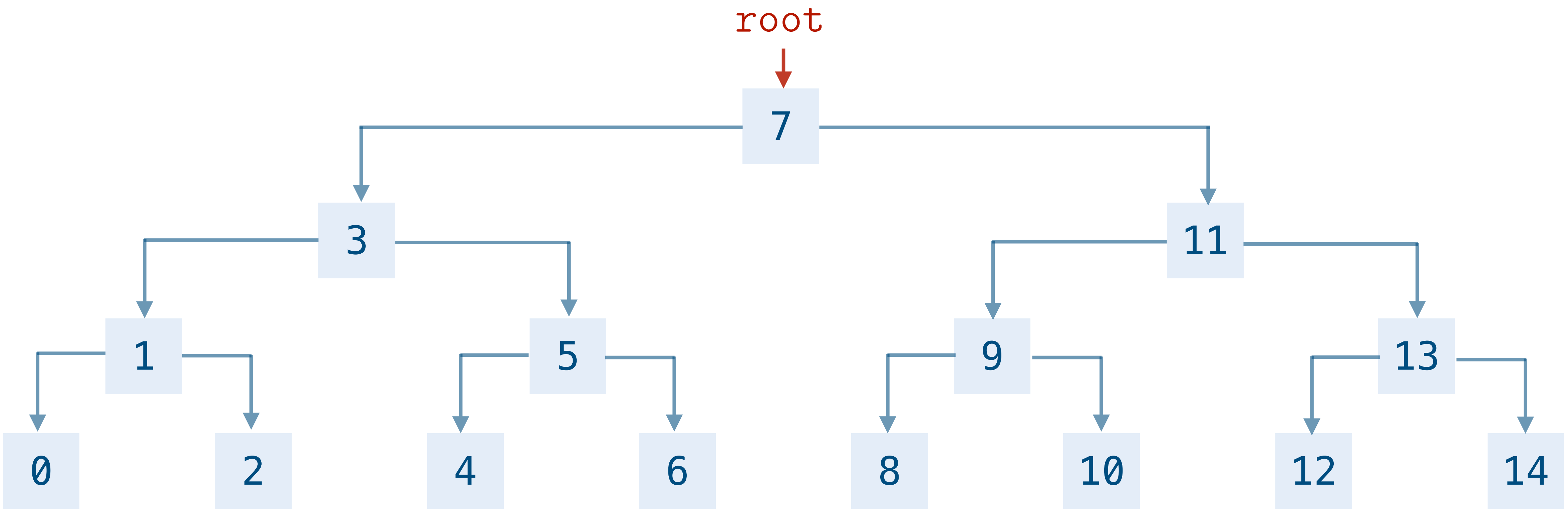
Binary Search Trees

`Search for 10`



Why is binary search possible on such a tree?

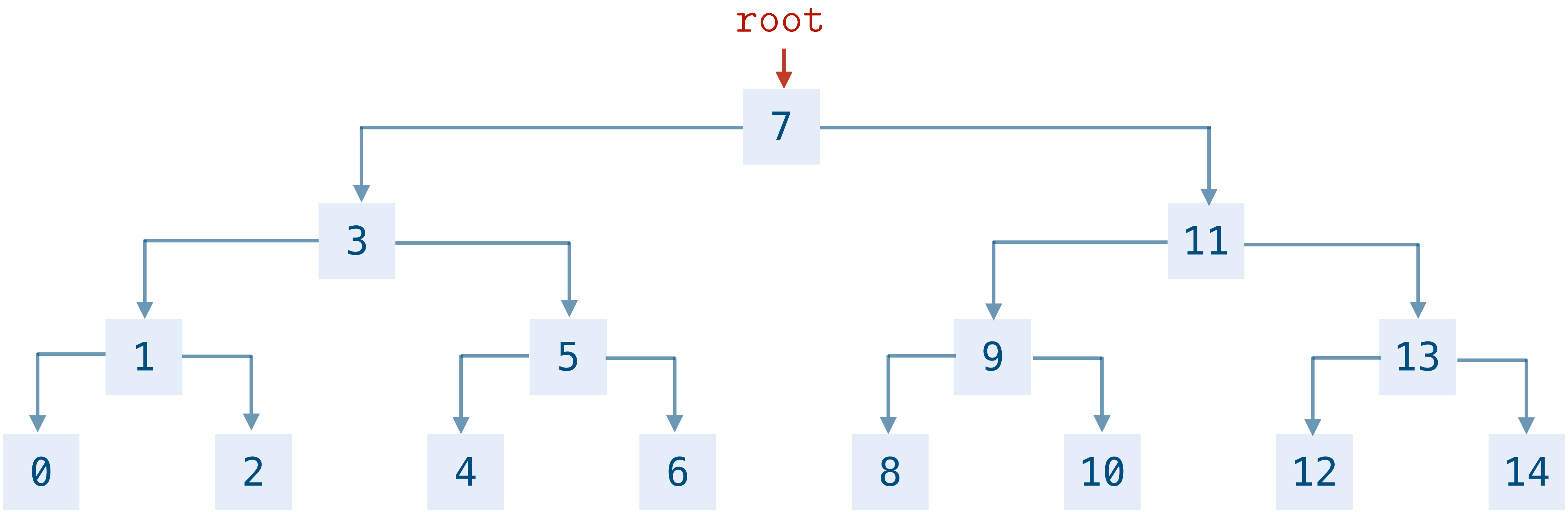
Binary Search Trees



Definition. A binary search tree (`BST`) is a binary tree where each node is:

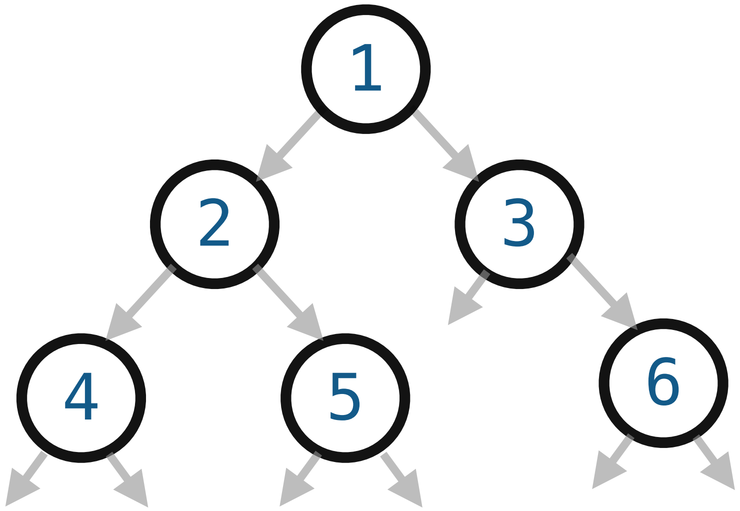
- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

Binary Search Trees

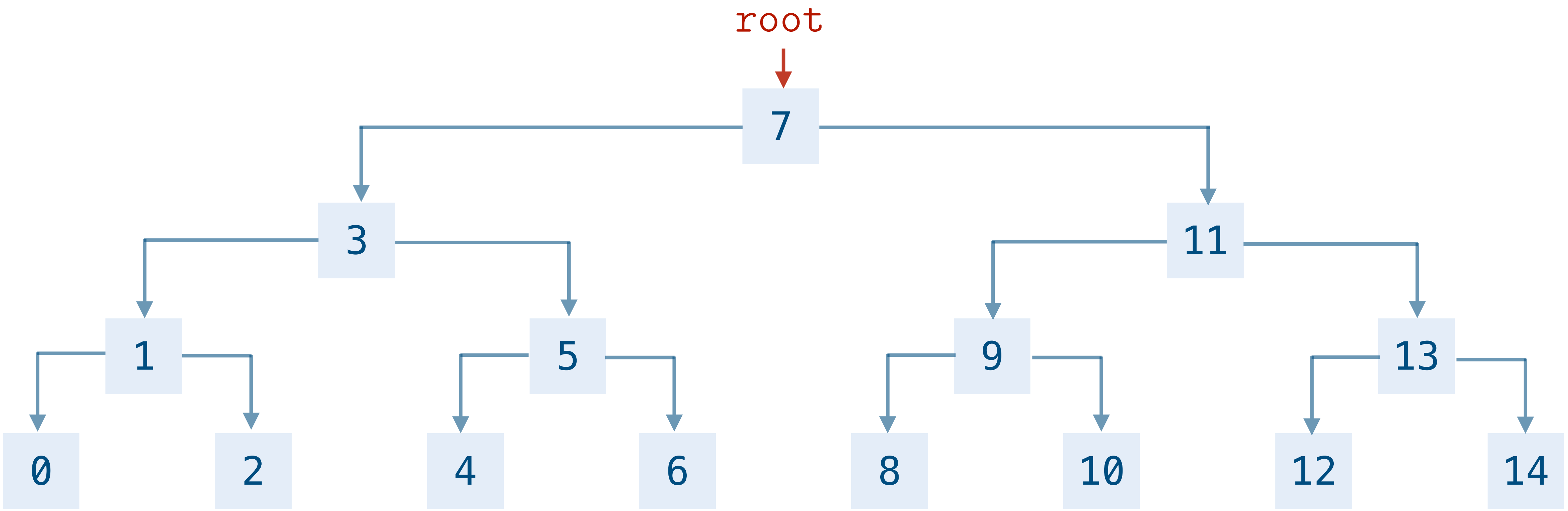


Definition. A binary search tree (`BST`) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

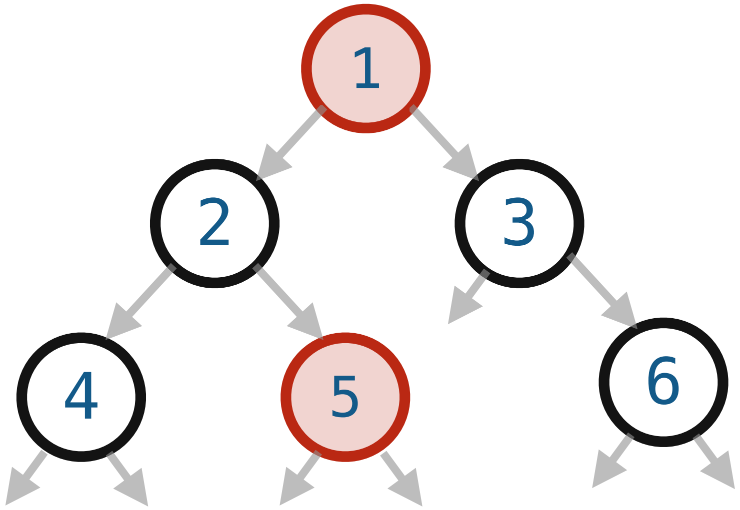


Binary Search Trees



Definition. A binary search tree (BST) is a binary tree where each node is:

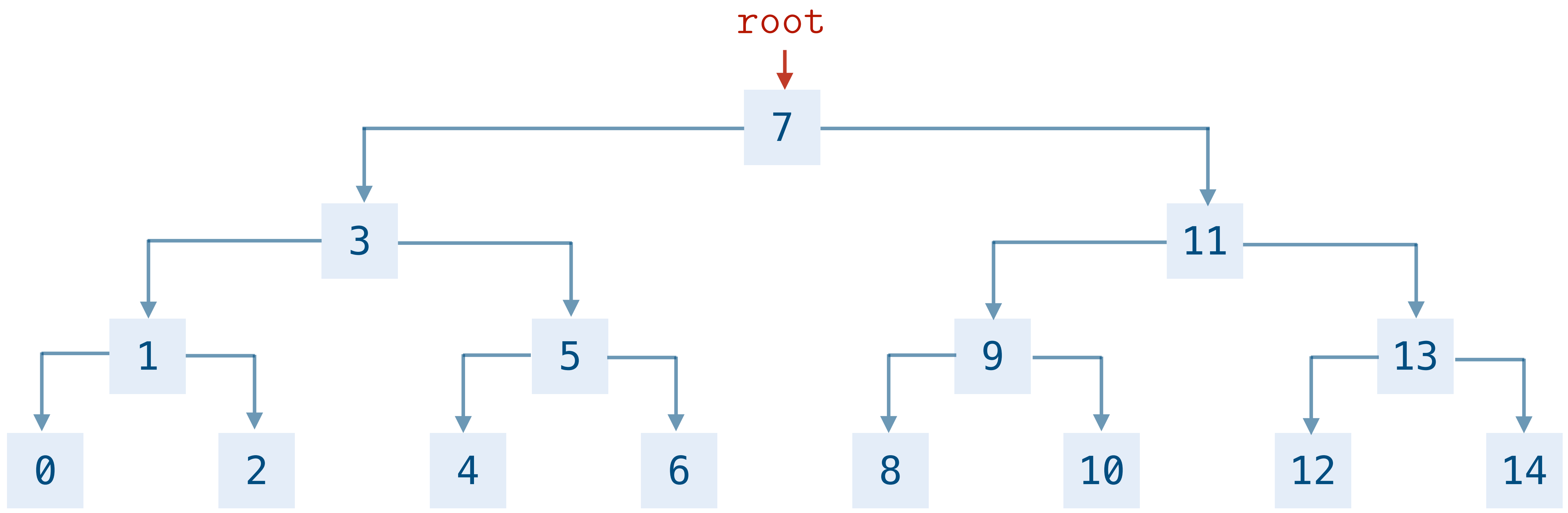
- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

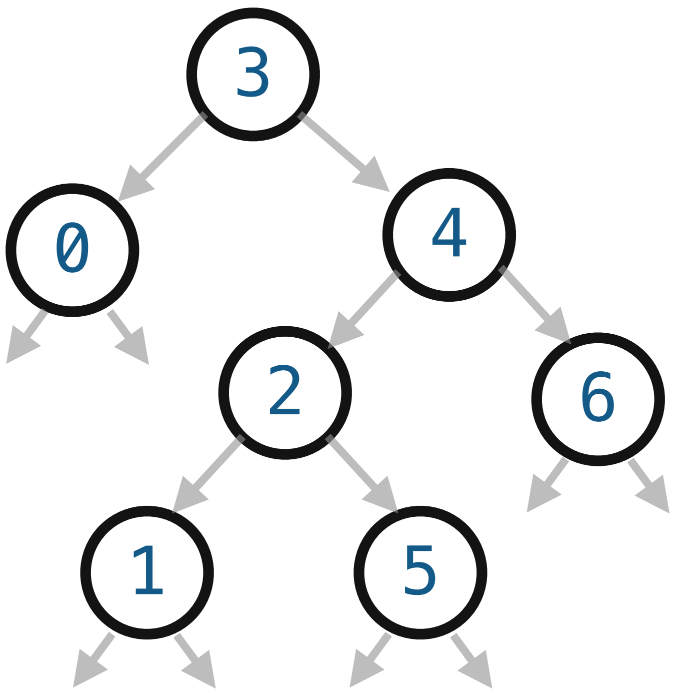


NOT a BST

5 is to the left of 1

Binary Search Trees

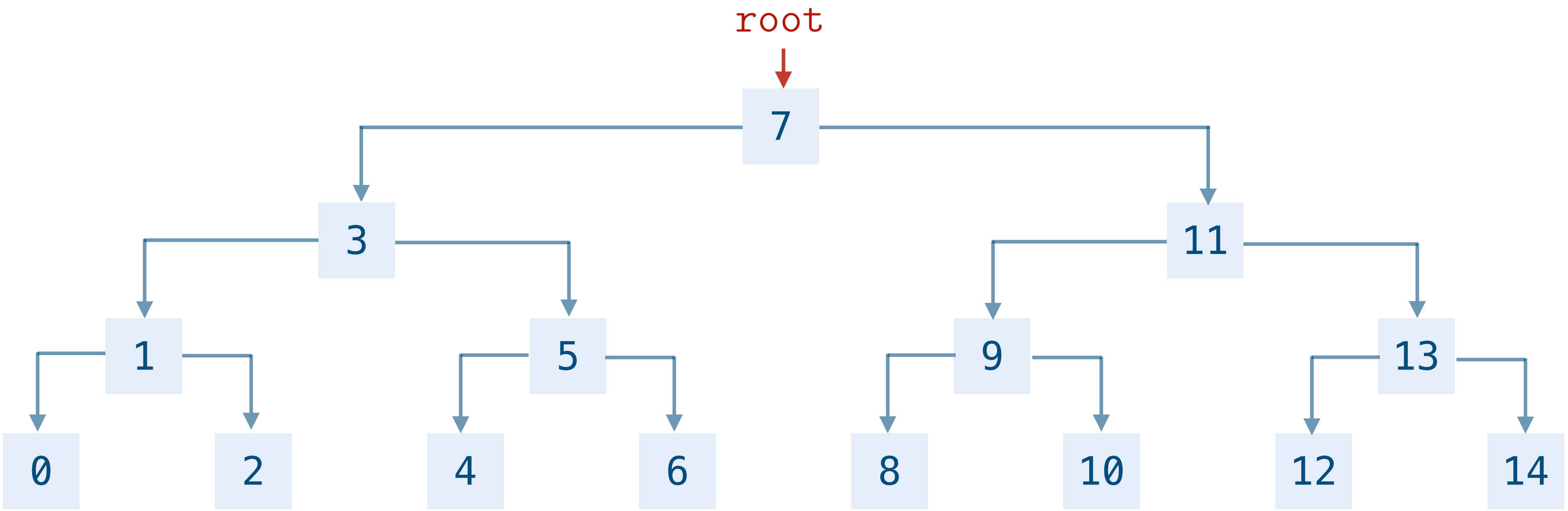




Definition. A binary search tree (BST) is a binary tree where each node is:

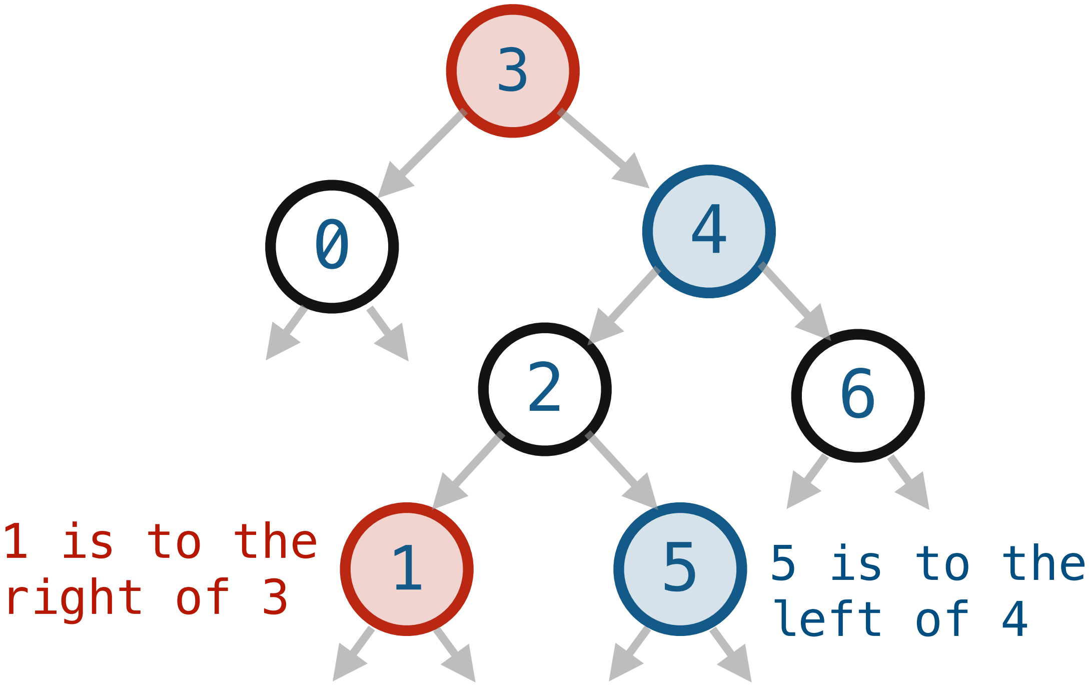
- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

Binary Search Trees



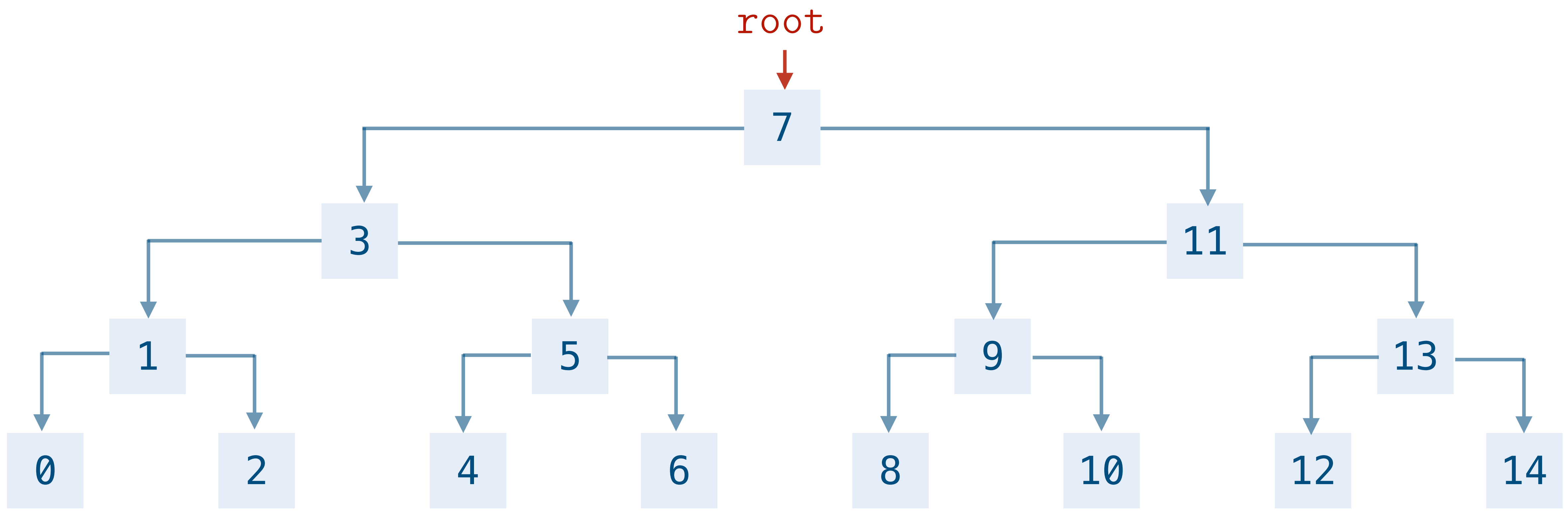
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



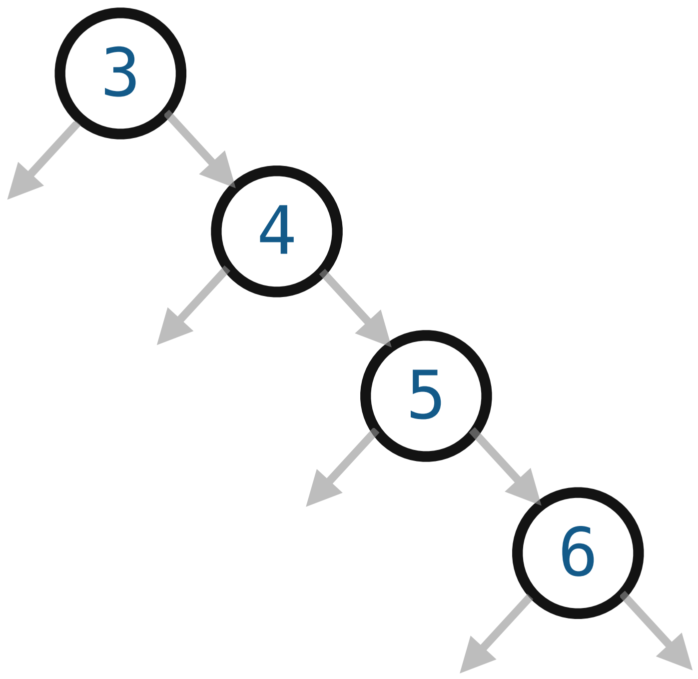
NOT a BST

Binary Search Trees

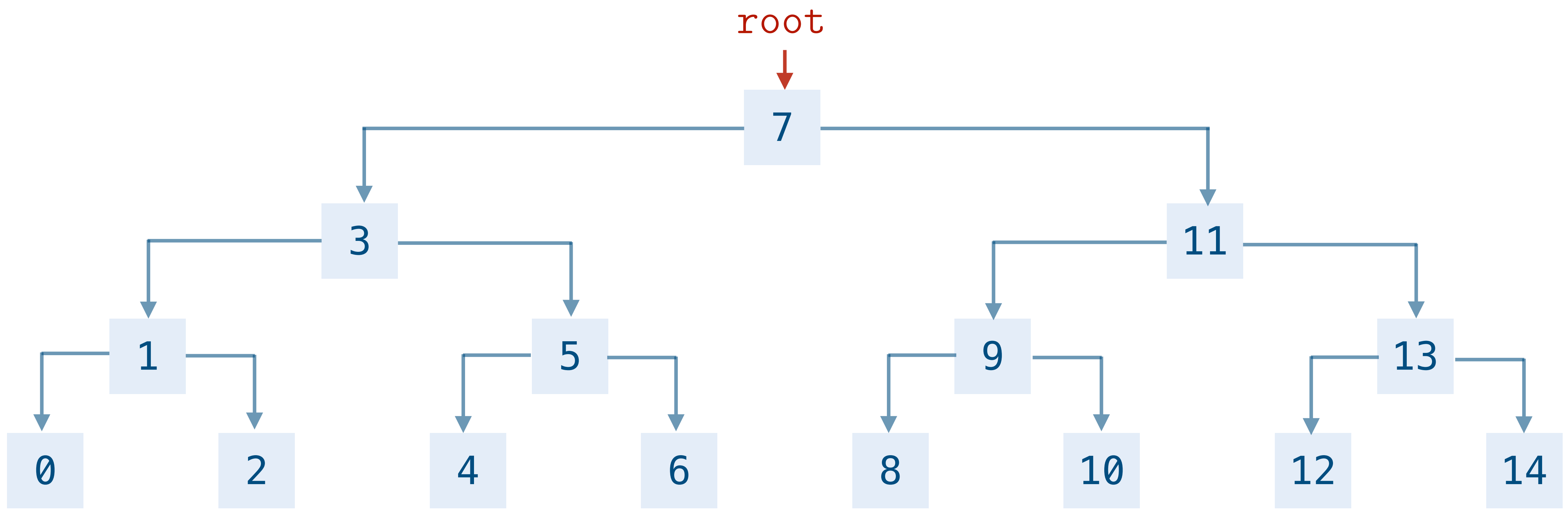


Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

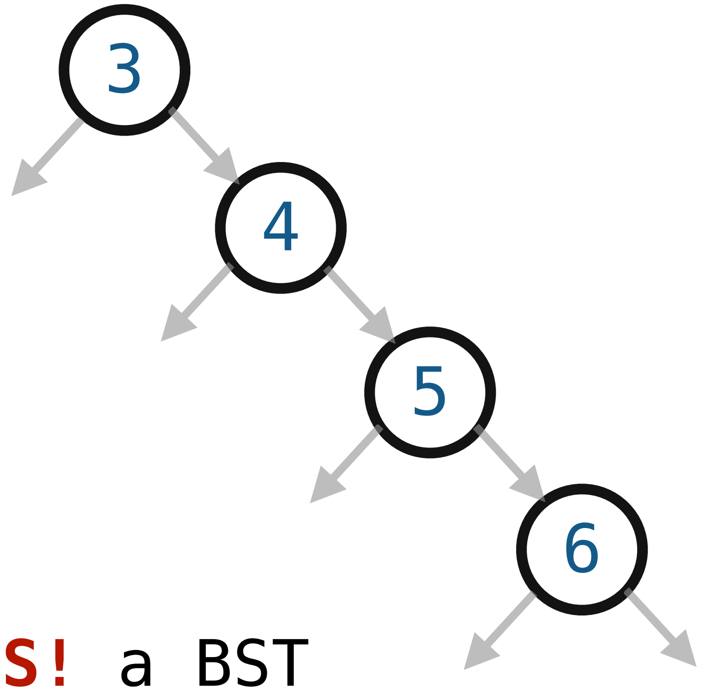


Binary Search Trees



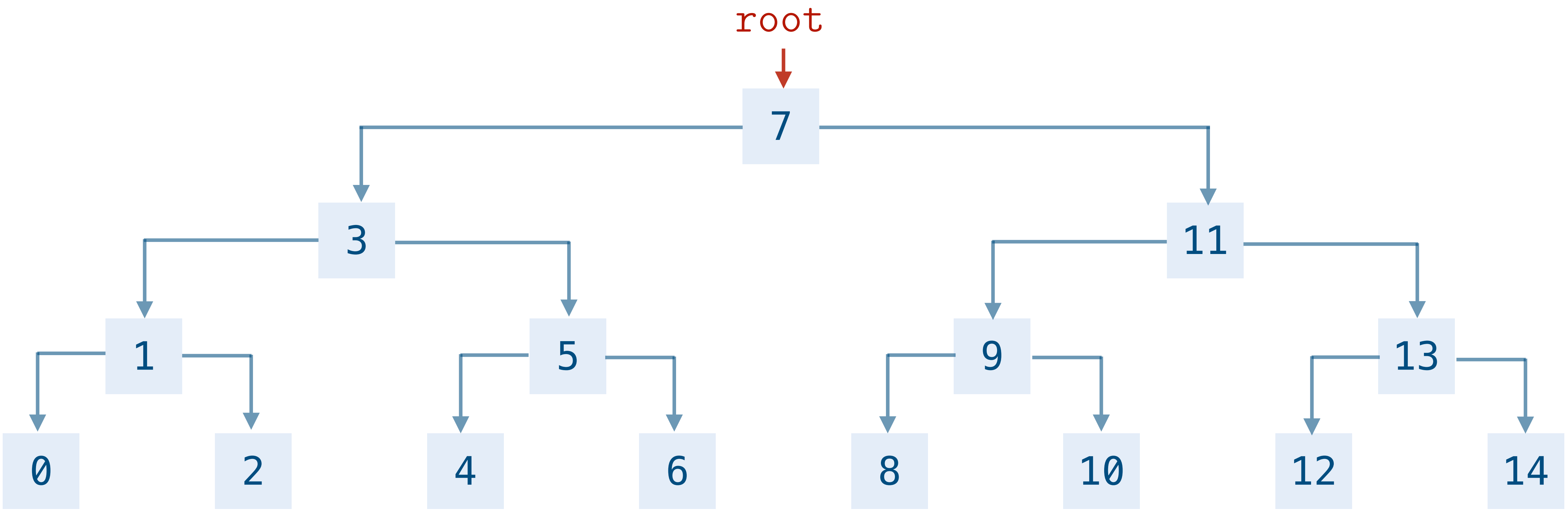
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



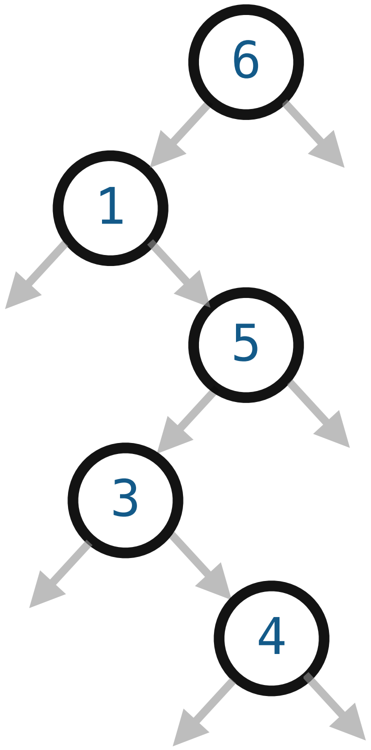
YES! a BST

Binary Search Trees

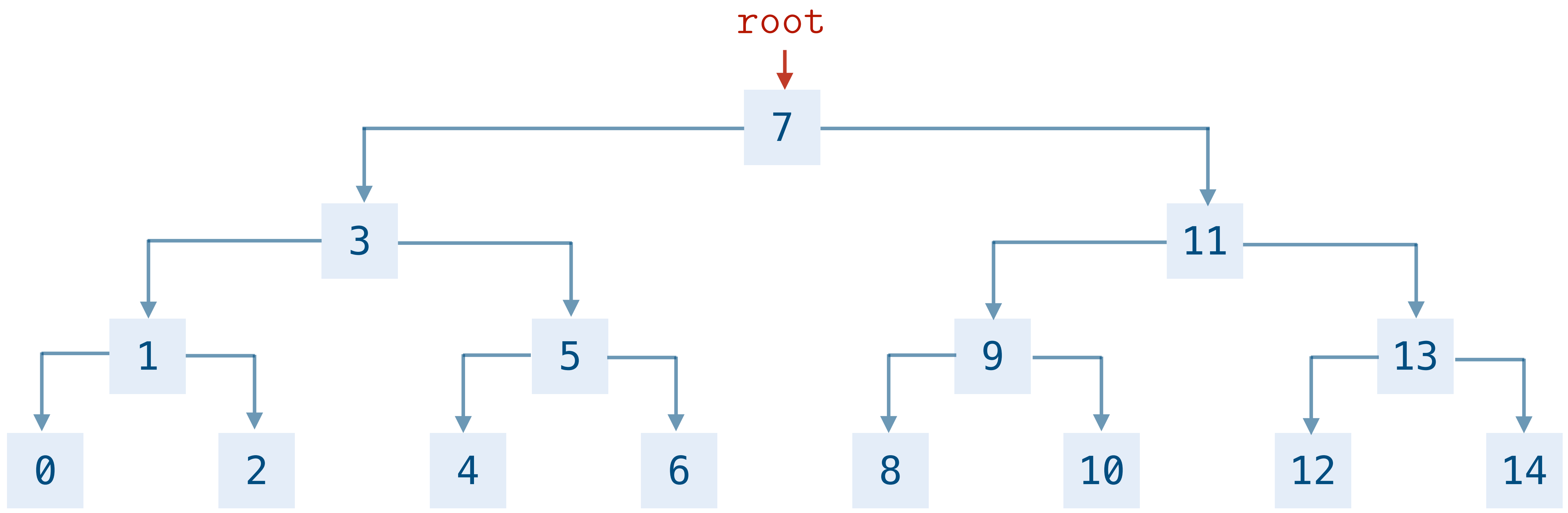


Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

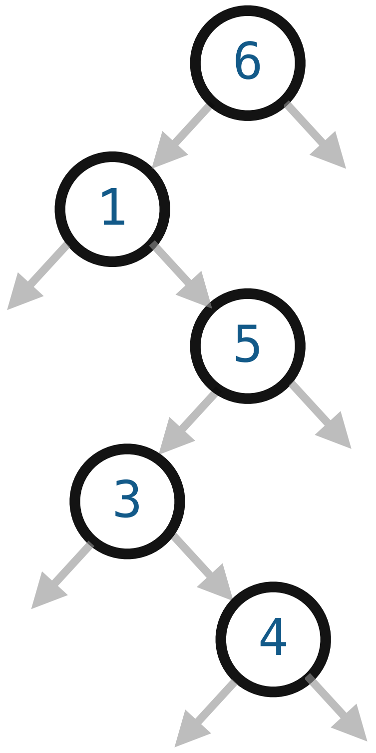


Binary Search Trees



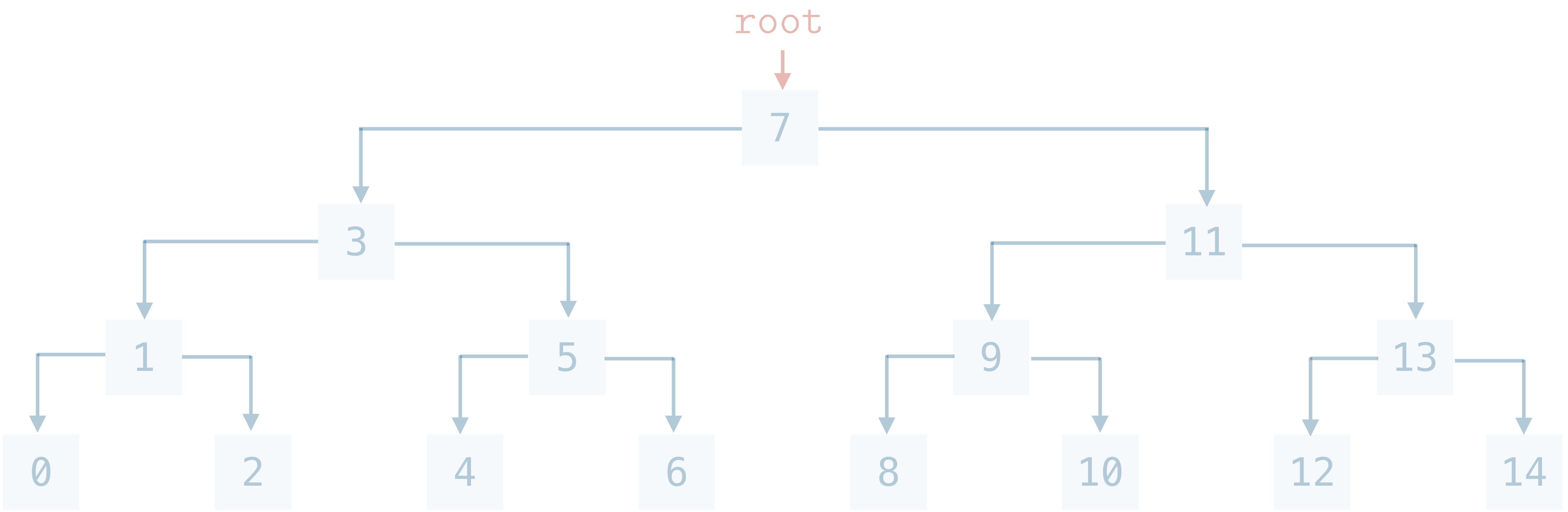
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



YES! a BST

Binary Search Trees



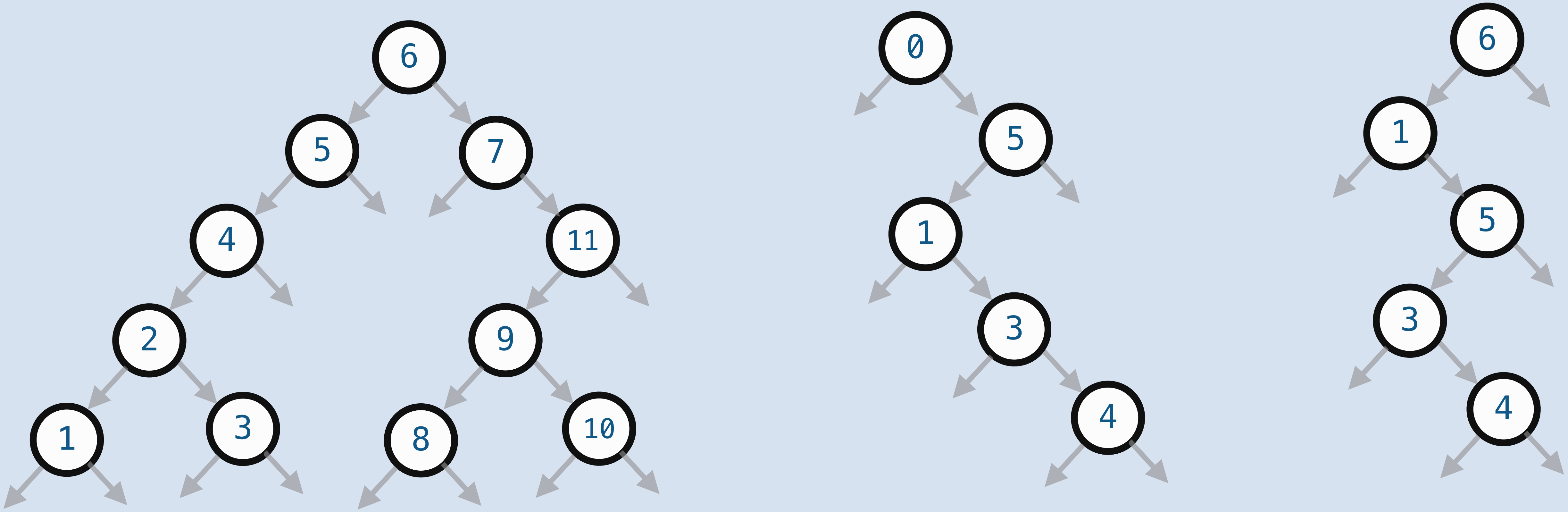
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

This applies also to every element in any sorted array!

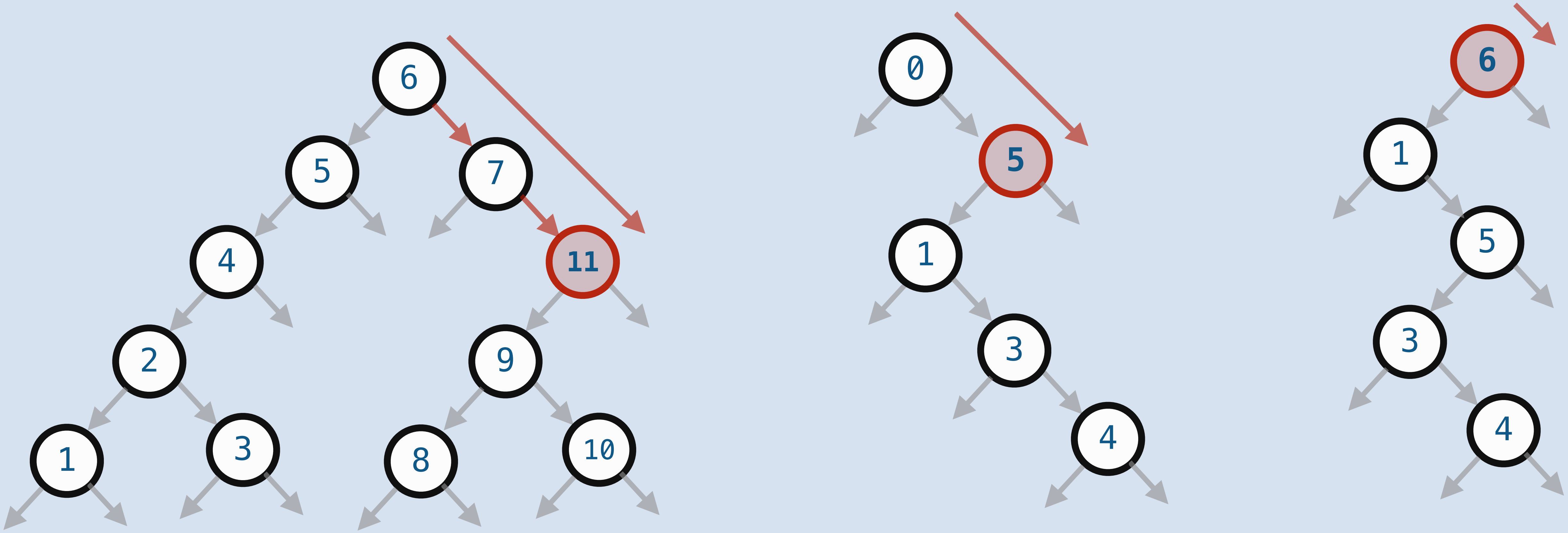
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise



Question. Which node is always the *maximum* node in a BST?

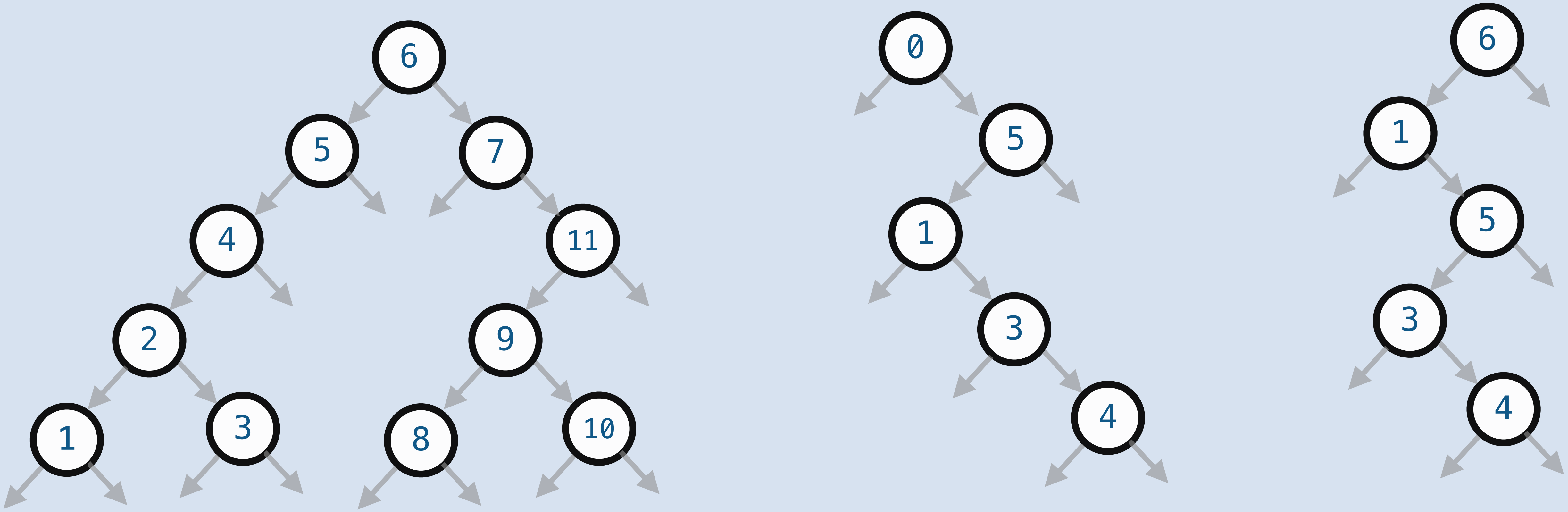
Exercise



Question. Which node is always the *maximum* node in a BST?

Answer. The right-most node.

Exercise

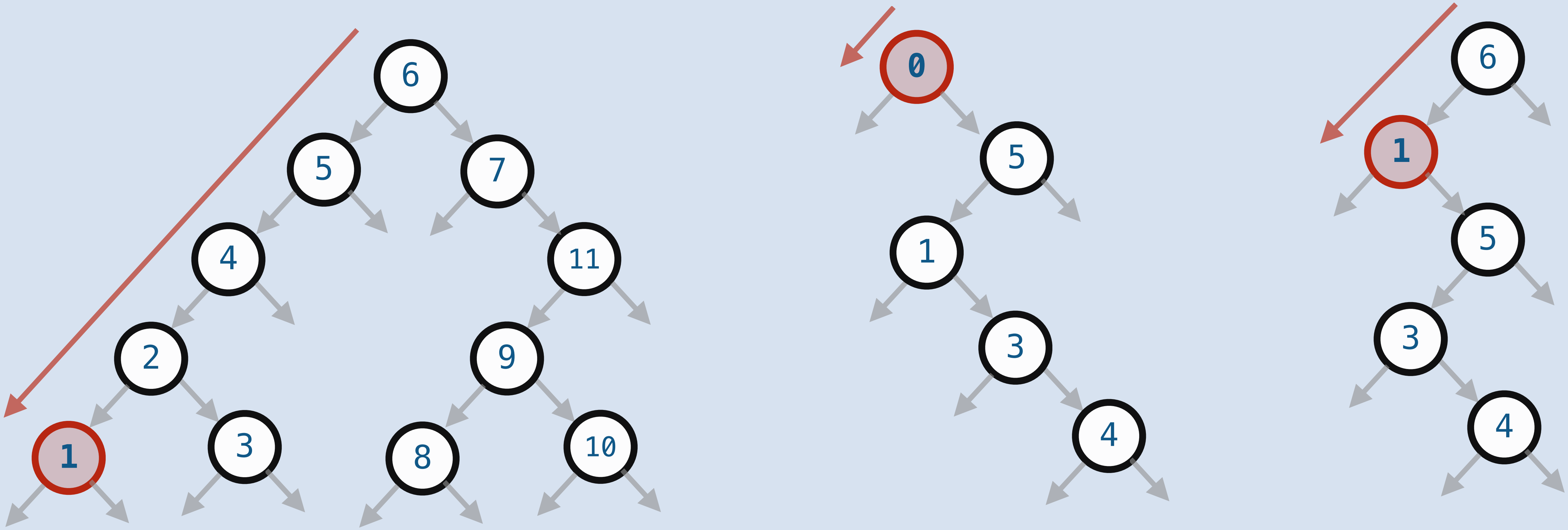


Question. Which node is always the *maximum* node in a BST?

Answer. The right-most node.

Question. Which node is always the *minimum* node in a BST?

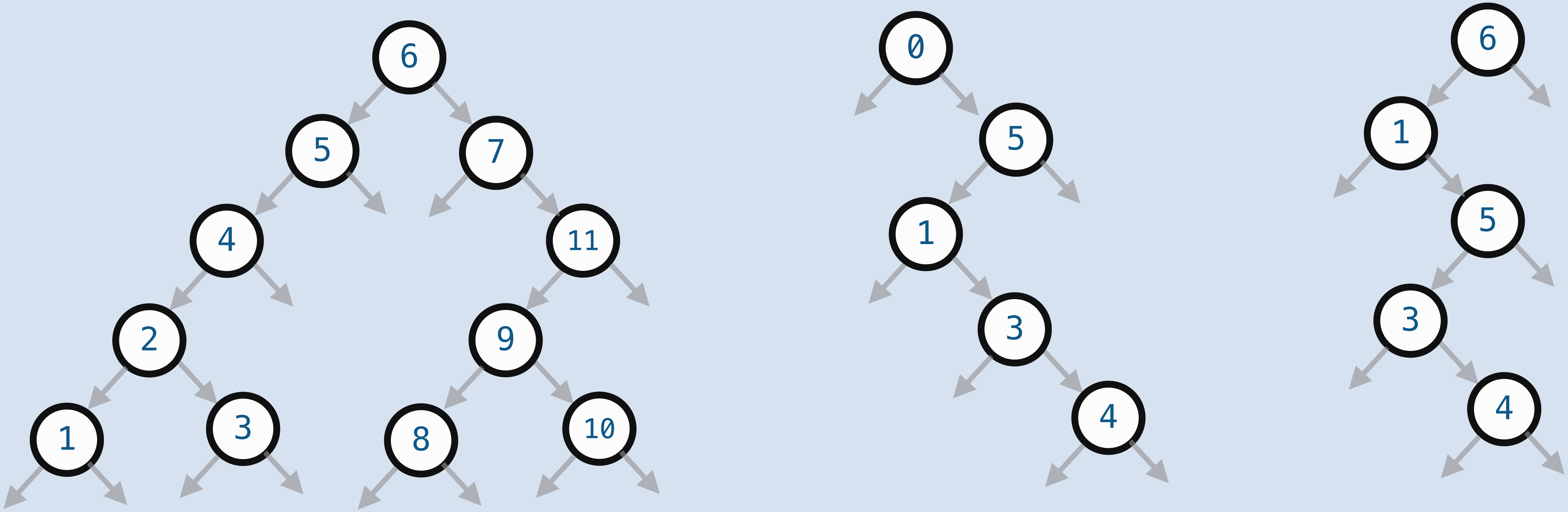
Exercise



Question. Which node is always the *maximum* node in a BST?
Answer. The right-most node.

Question. Which node is always the *minimum* node in a BST?
Answer. The left-most node.

Exercise



Question. Which node is always the *maximum* node in a BST?

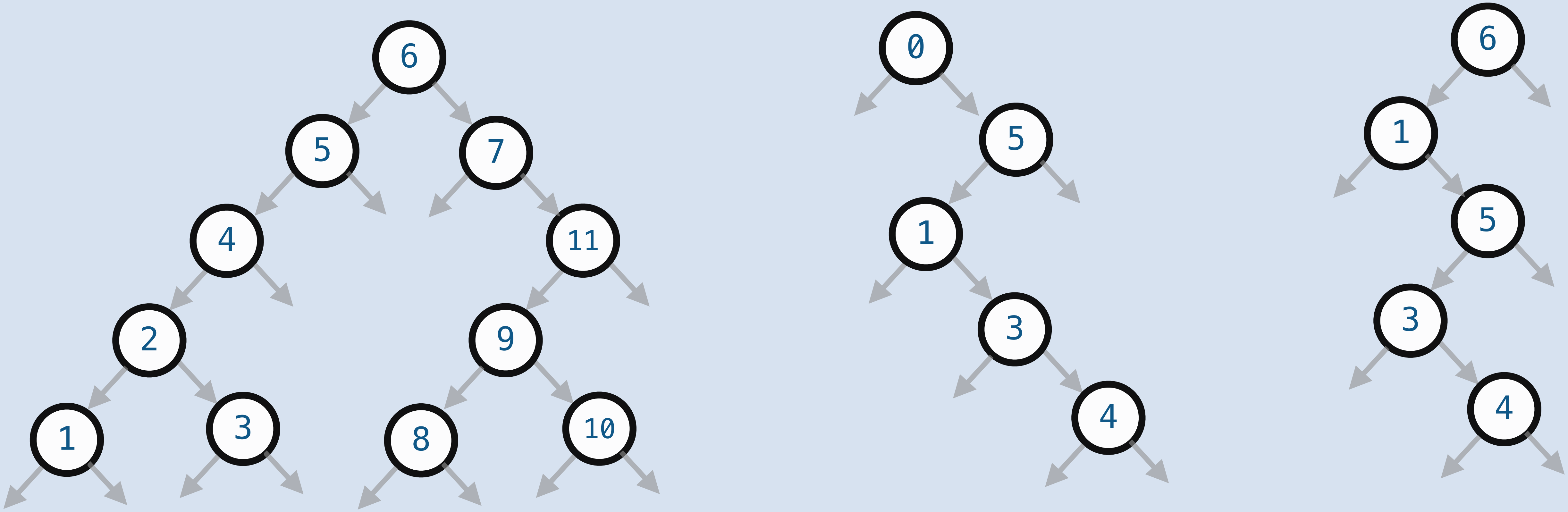
Answer. The right-most node.

Question. Which node is always the *minimum* node in a BST?

Answer. The left-most node.

Question. Which node is always the *median* node in a BST?

Exercise

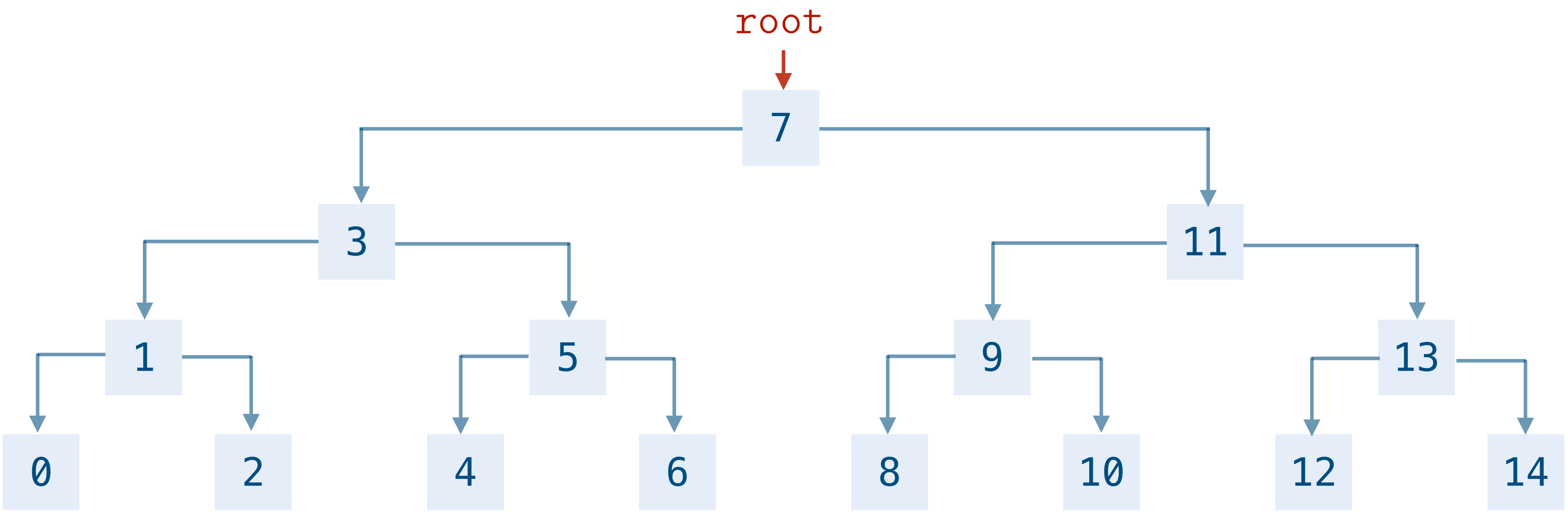


Question. Which node is always the *maximum* node in a BST?
Answer. The right-most node.

Question. Which node is always the *minimum* node in a BST?
Answer. The left-most node.

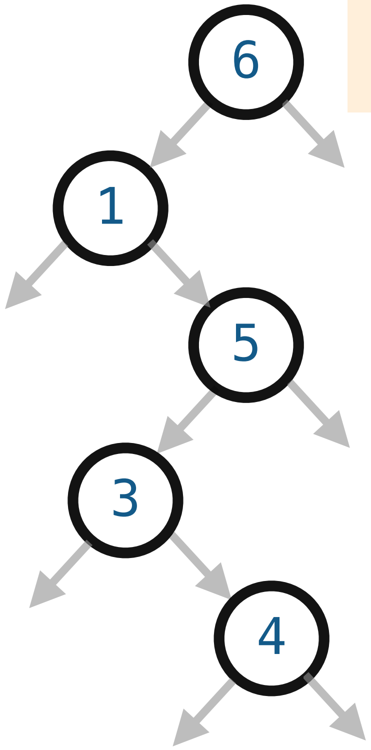
Question. Which node is always the *median* node in a BST?
Answer. Impossible to tell without performing a search for the median. *stay tuned!*

Binary Search Trees



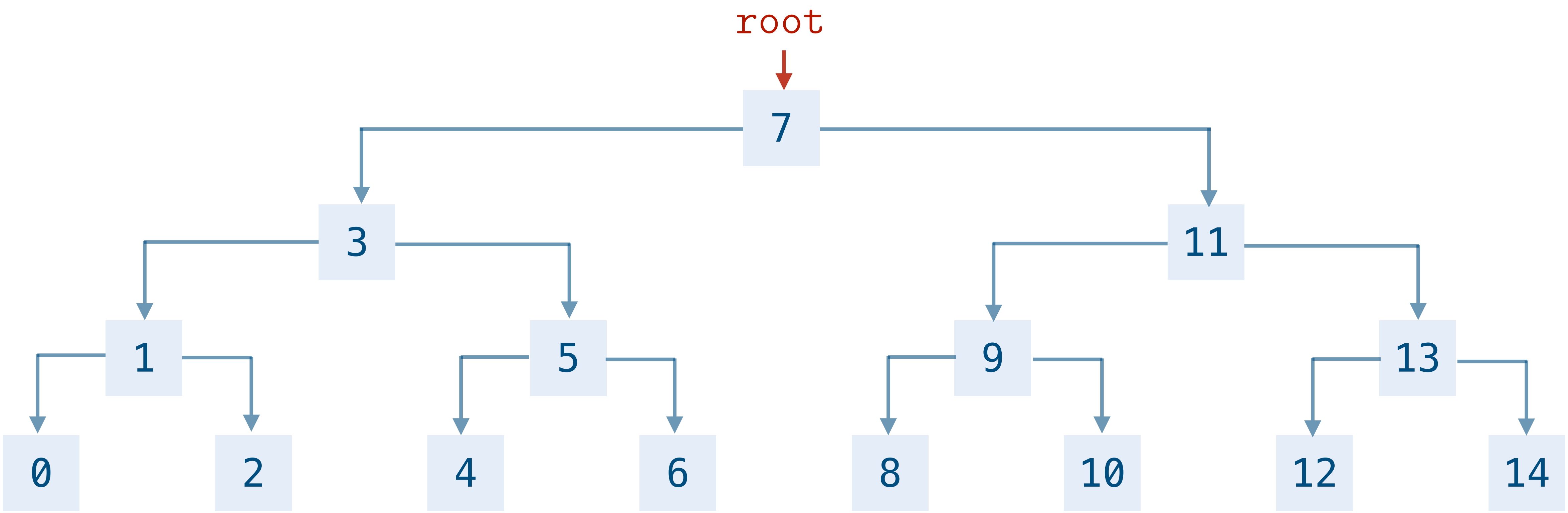
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



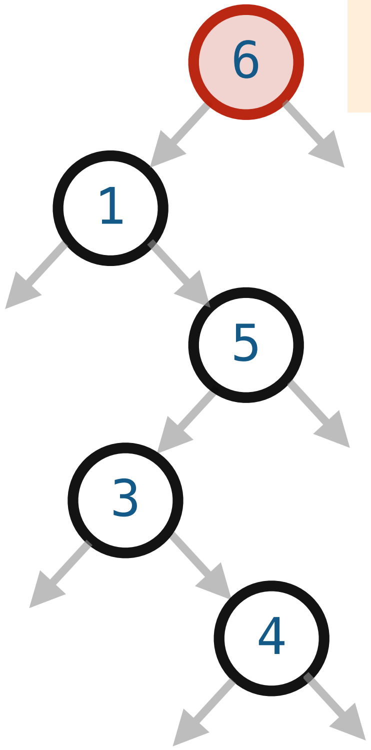
Search for 2

Binary Search Trees



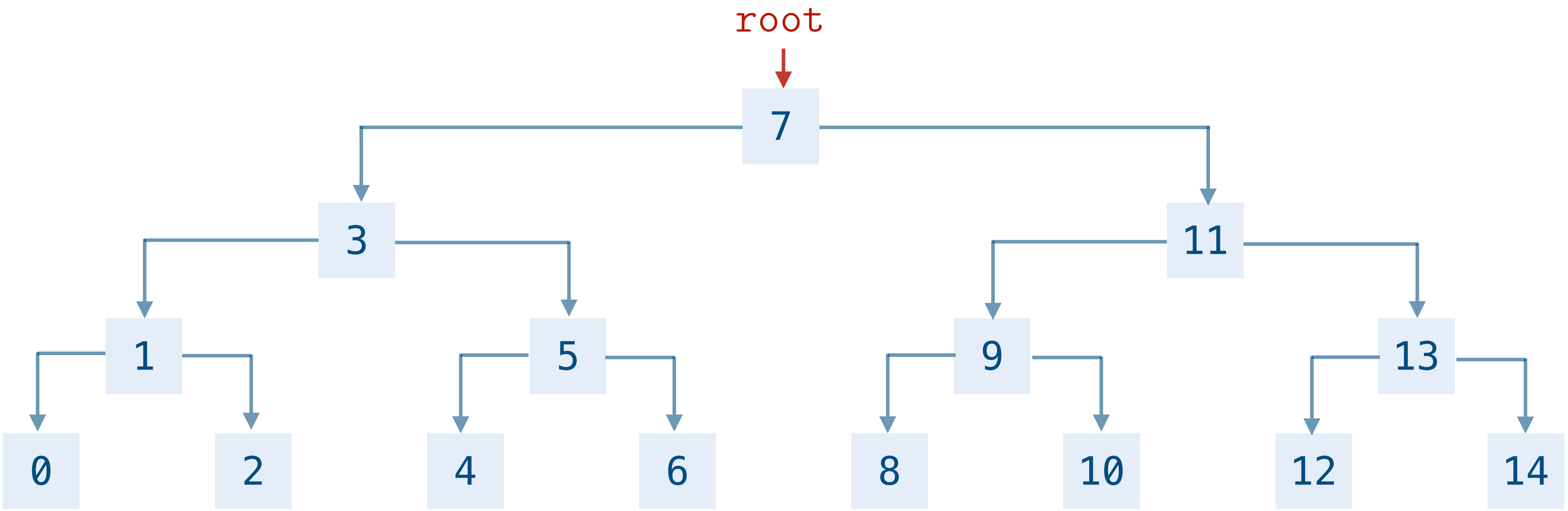
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



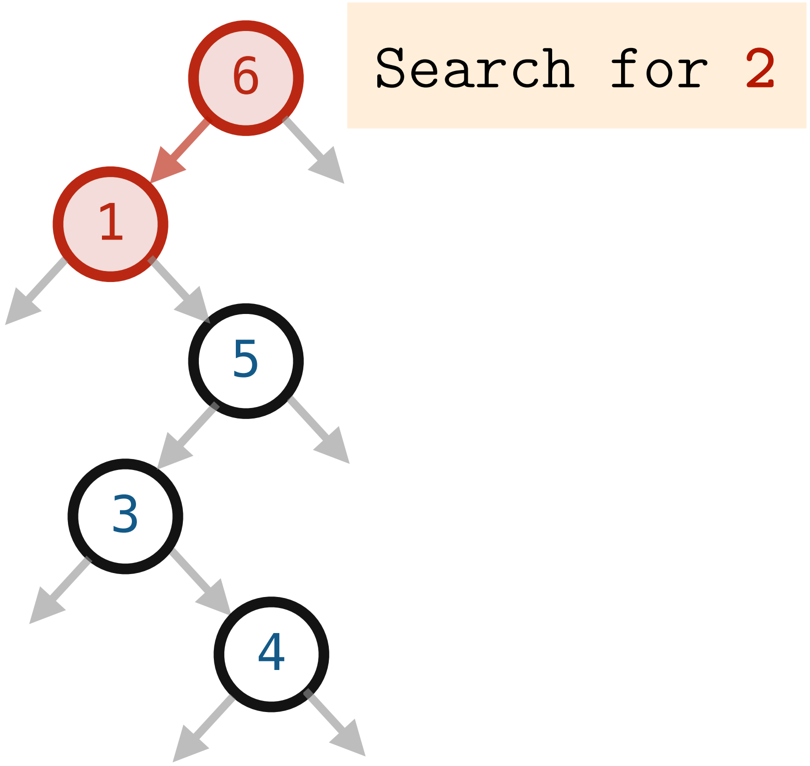
`Search for 2`

Binary Search Trees

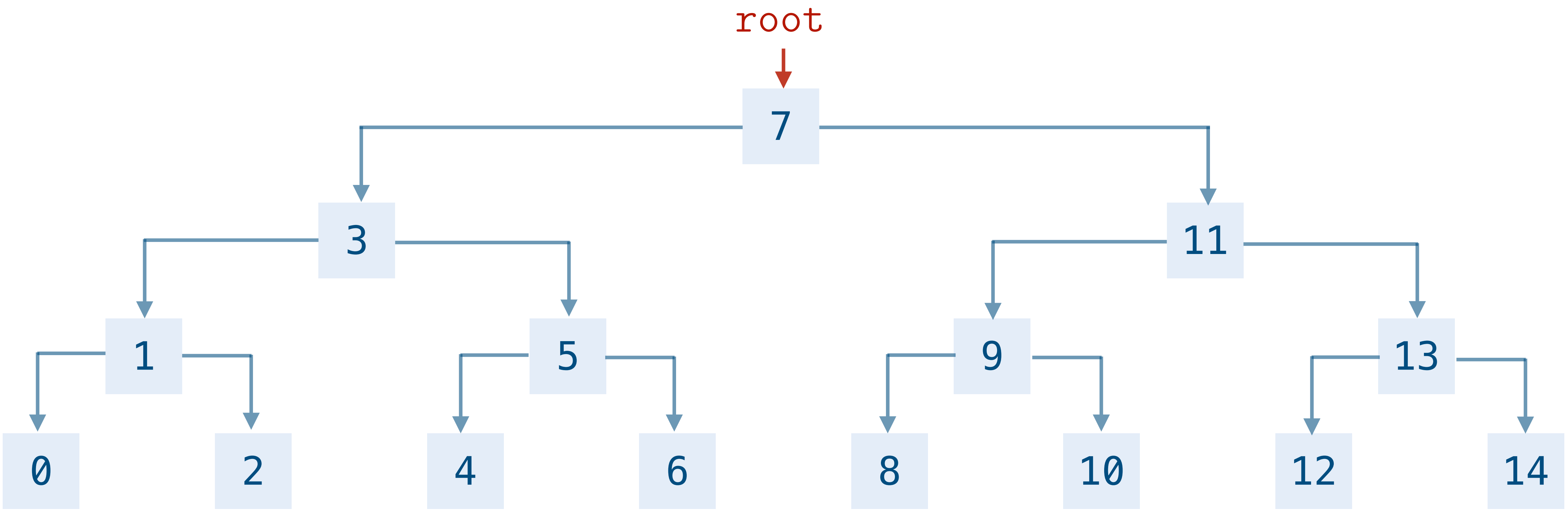


Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

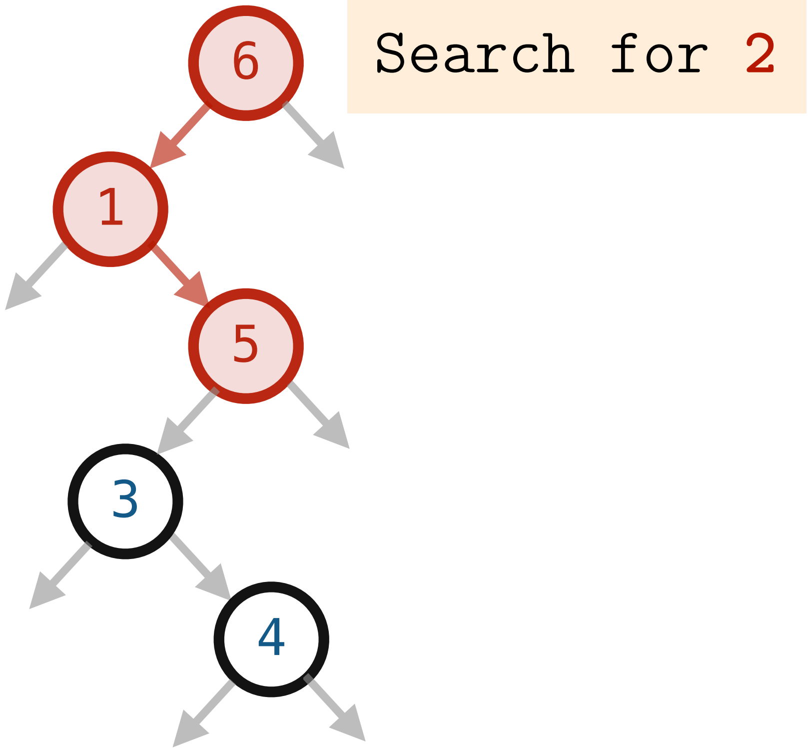


Binary Search Trees



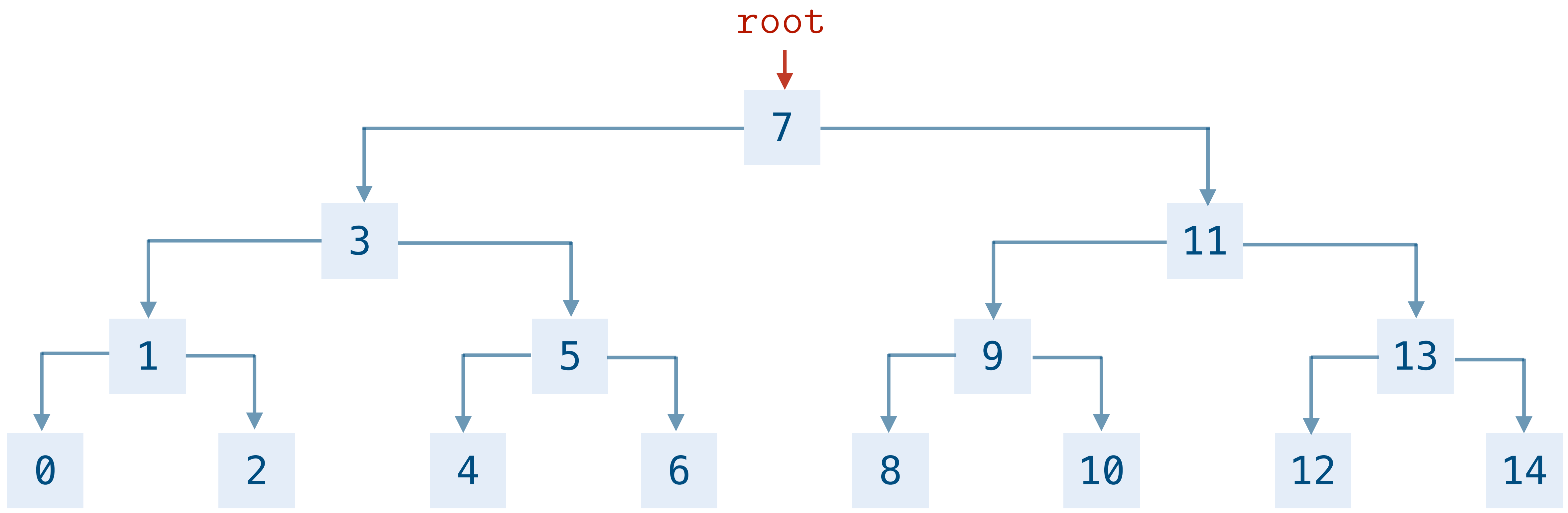
Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



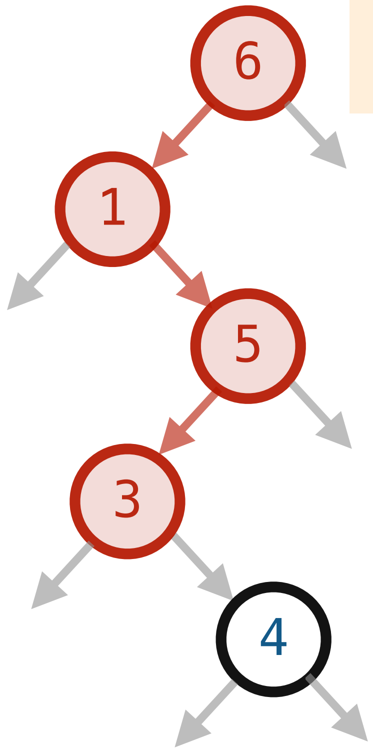
Search for 2

Binary Search Trees



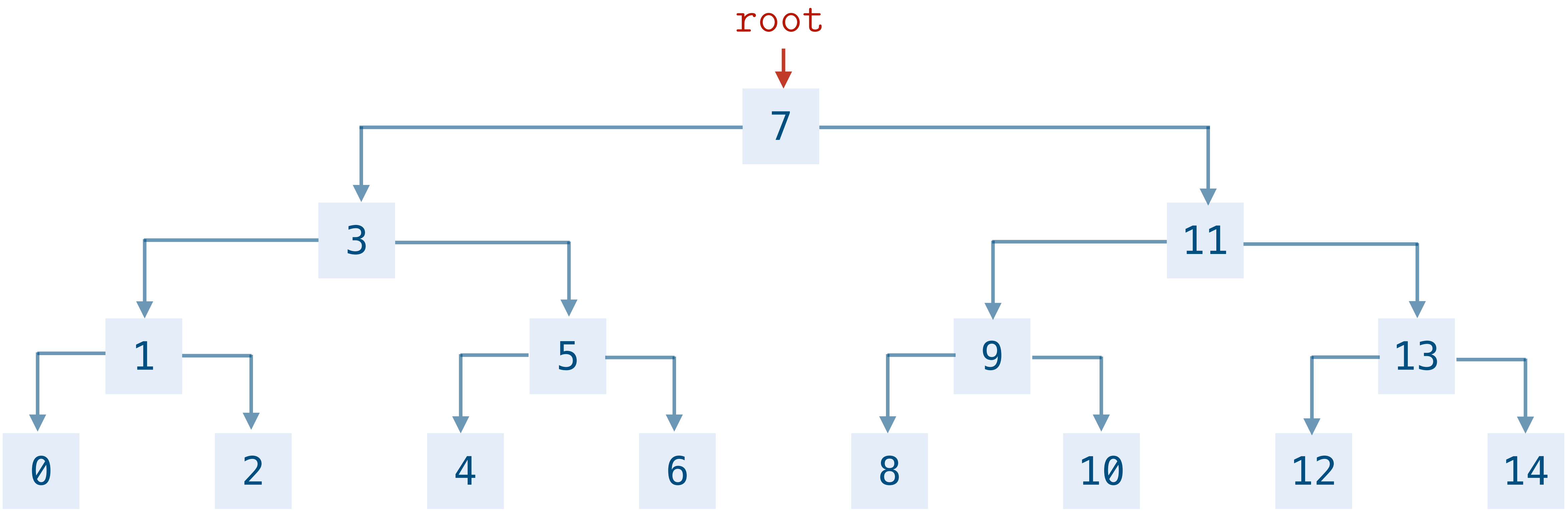
Definition. A binary search tree (`BST`) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



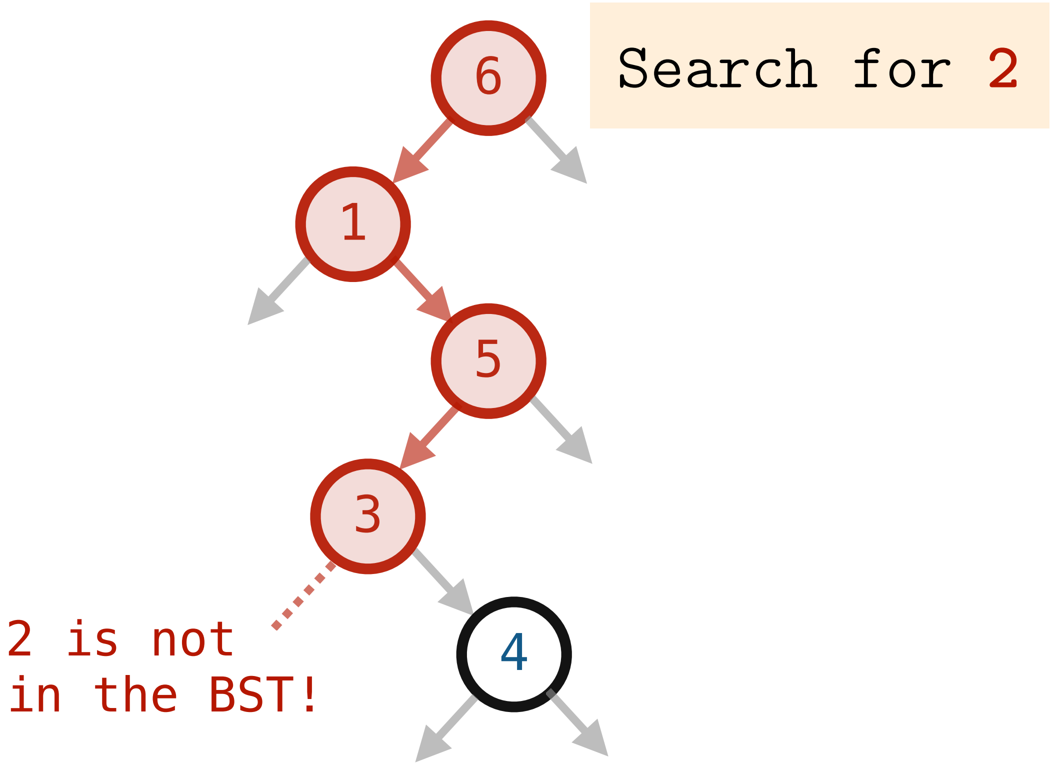
`Search for 2`

Binary Search Trees

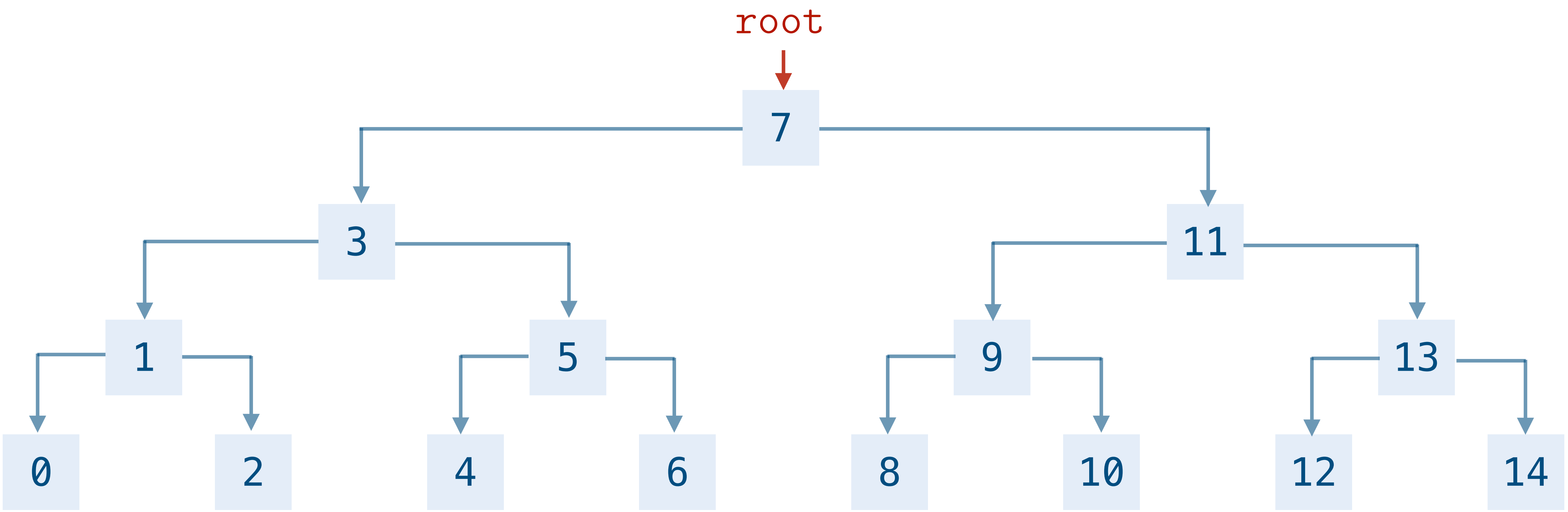


Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.



Binary Search Trees

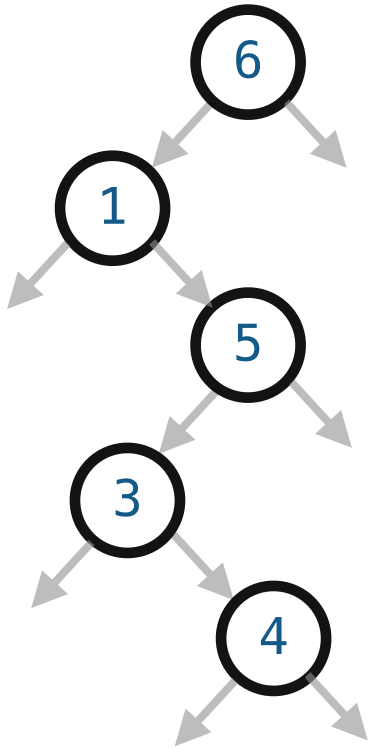


Definition. A binary search tree (BST) is a binary tree where each node is:

- larger than all the nodes to its left.
- smaller than all the nodes to its right.
- a binary search tree.

! Searching a BST requires *O*(*height*) compares.

what is the height of a BST?



YES! a BST

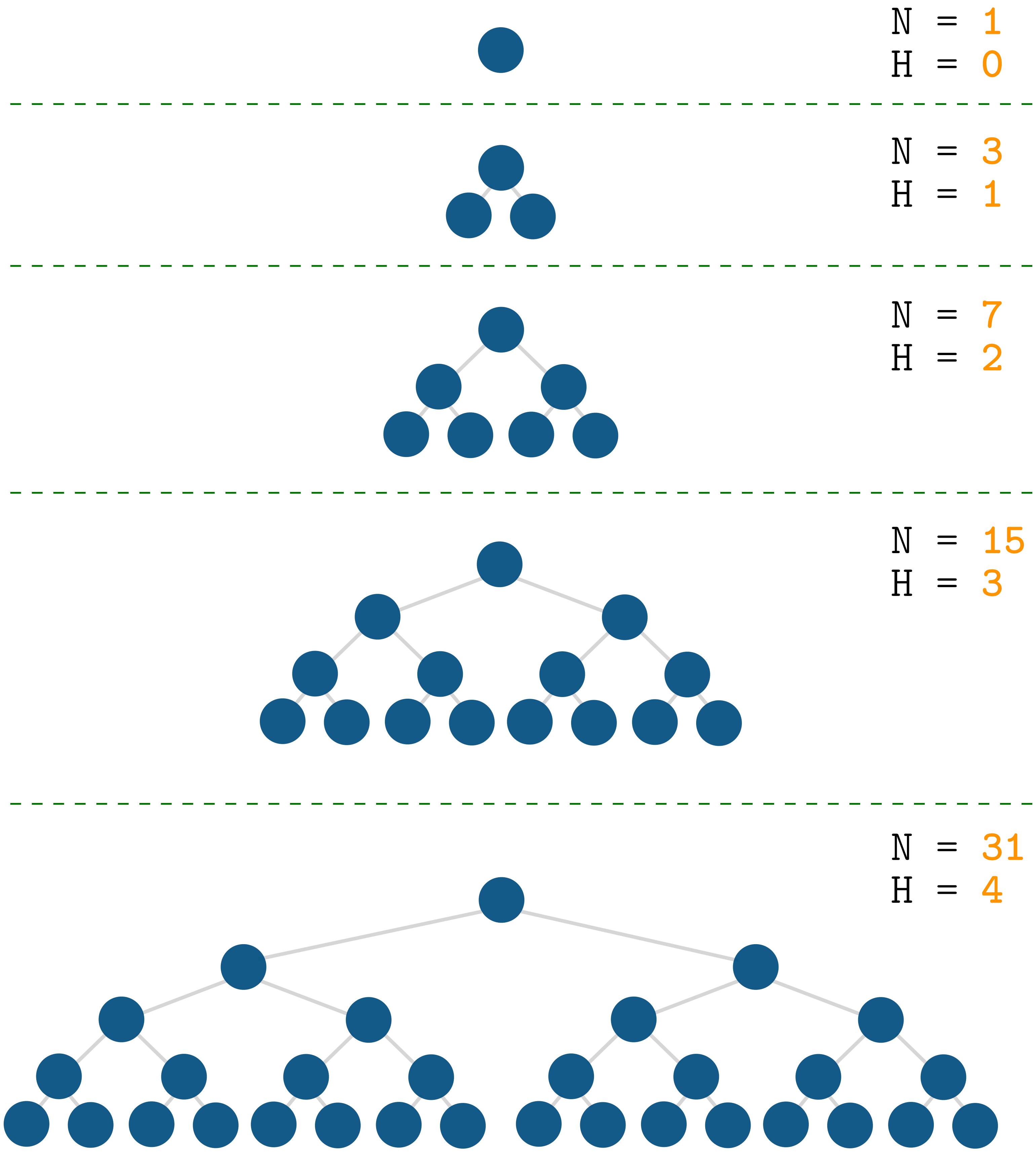
Perfect Binary Trees

Definition.

- All levels are full.
- All leafs are in the last level and all internal nodes have exactly two children.

CAUTION

You might find different definitions online. Some texts call this tree *complete*



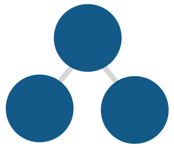
Perfect Binary Trees

Definition.

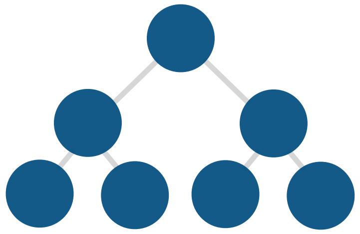
- All levels are full.
- All leafs are in the last level and all internal nodes have exactly two children.

$$\text{\# of nodes } (N) = 2^{\text{height} + 1} - 1$$

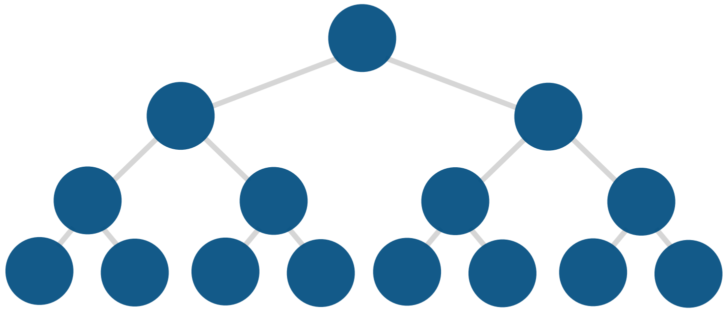
N = 1
H = 0



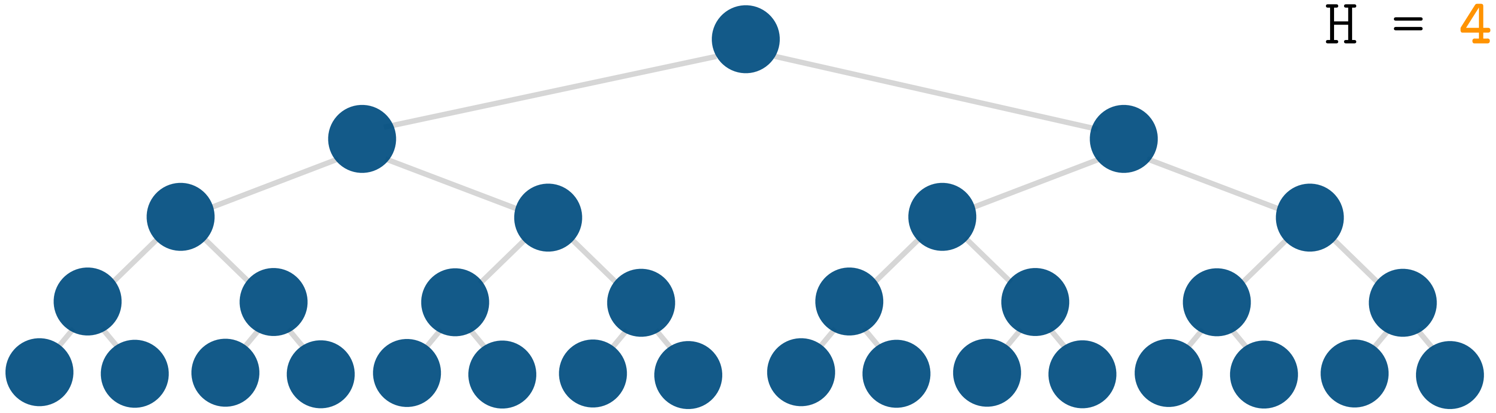
N = 3
H = 1



N = 7
H = 2



N = 15
H = 3



N = 31
H = 4

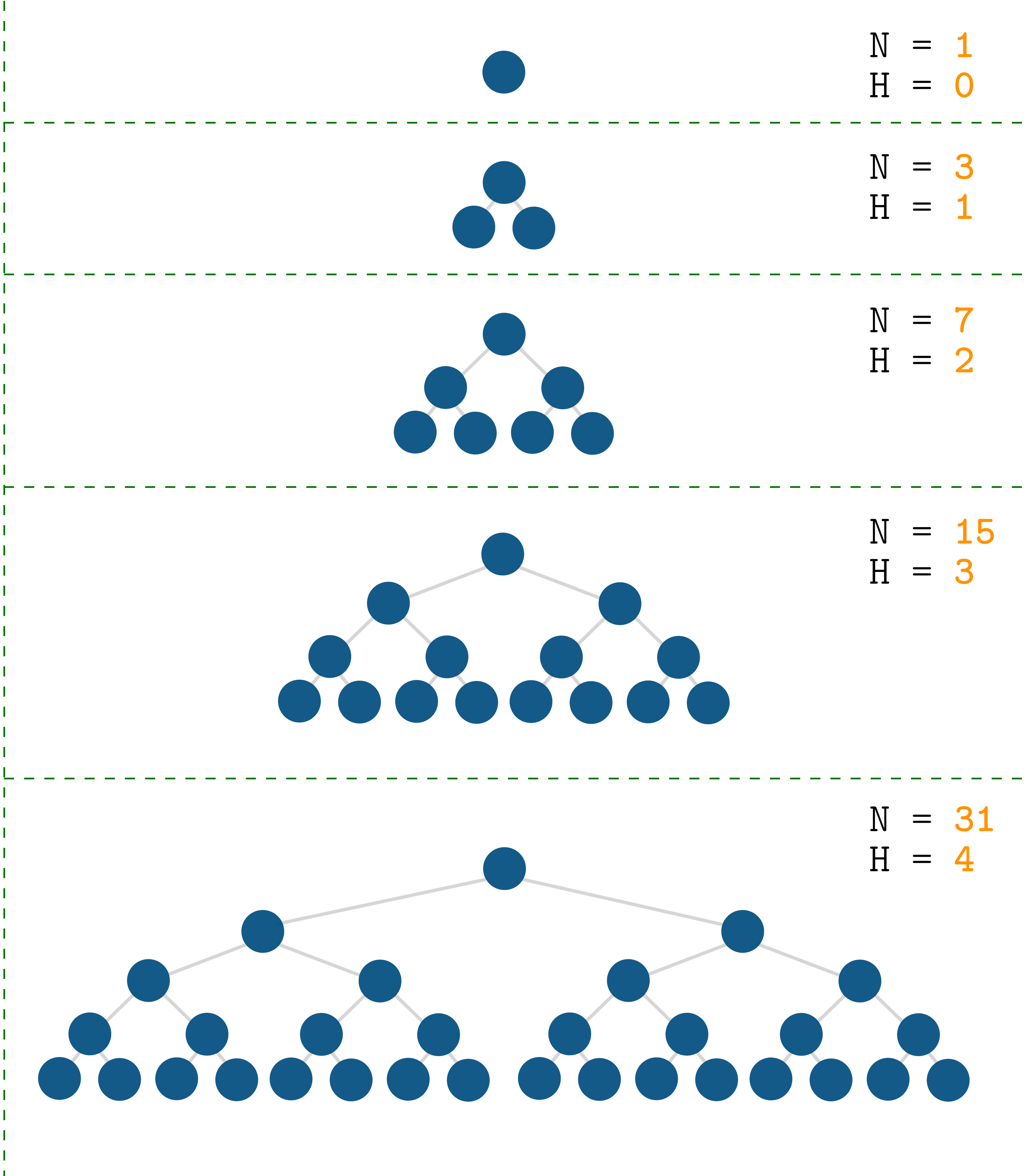
Perfect Binary Trees

Definition.

- All levels are full.
- All leafs are in the last level and all internal nodes have exactly two children.

$$\# \text{ of nodes } (N) = 2^{\text{height } + 1} - 1$$

$$\text{Height} = \log_2(N + 1) - 1$$



Perfect Binary Trees

Definition.

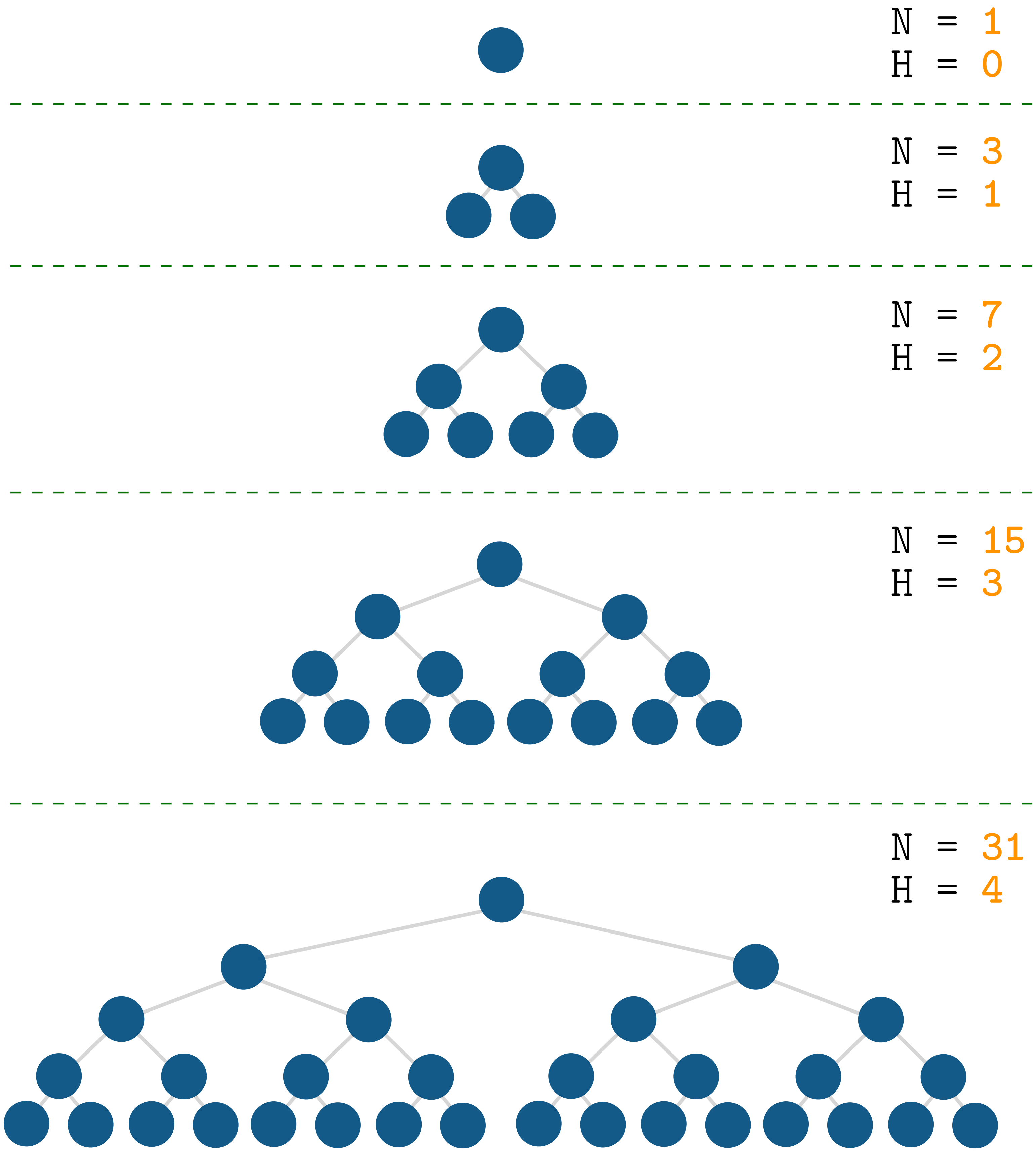
- All levels are full.
- All leafs are in the last level and all internal nodes have exactly two children.

$$\# \text{ of nodes } (N) = 2^{\text{height} + 1} - 1$$

$$\text{Height} = \log_2(N + 1) - 1$$

IMPORTANT!

The height of a *perfect* binary tree is *logarithmic* in the number of nodes in the tree!



Perfect Binary Trees

Definition.

- All levels are full.
- All leafs are in the last level and all internal nodes have exactly two children.

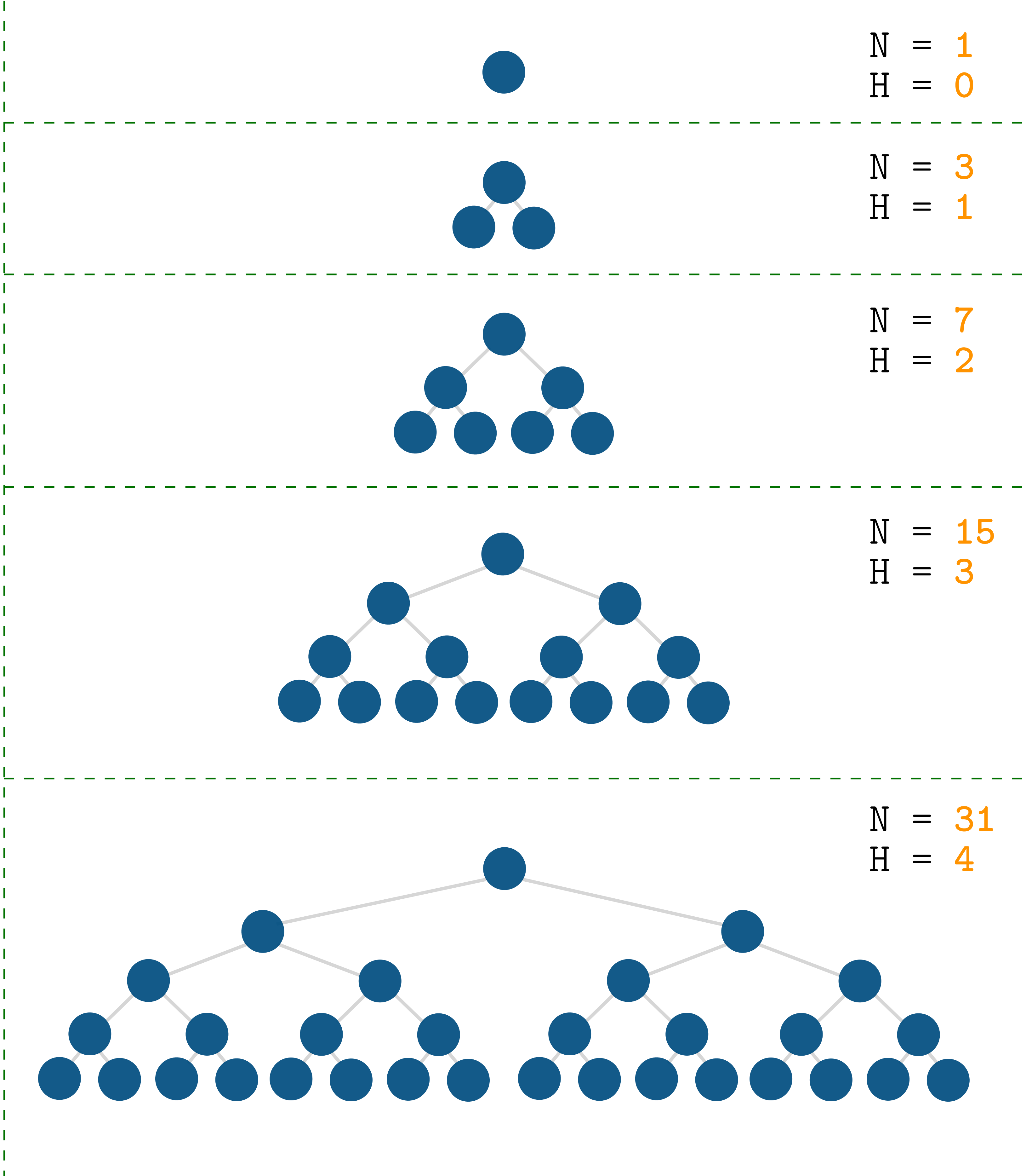
$$\text{\# of nodes } (N) = 2^{\text{height} + 1} - 1$$

$$\text{Height} = \log_2(N + 1) - 1$$

$$\text{\# of leafs} = 2^{\text{height}}$$

$$\text{\# of leafs} = \frac{N + 1}{2}$$

$$\text{\# of internal nodes} = \frac{N - 1}{2}$$



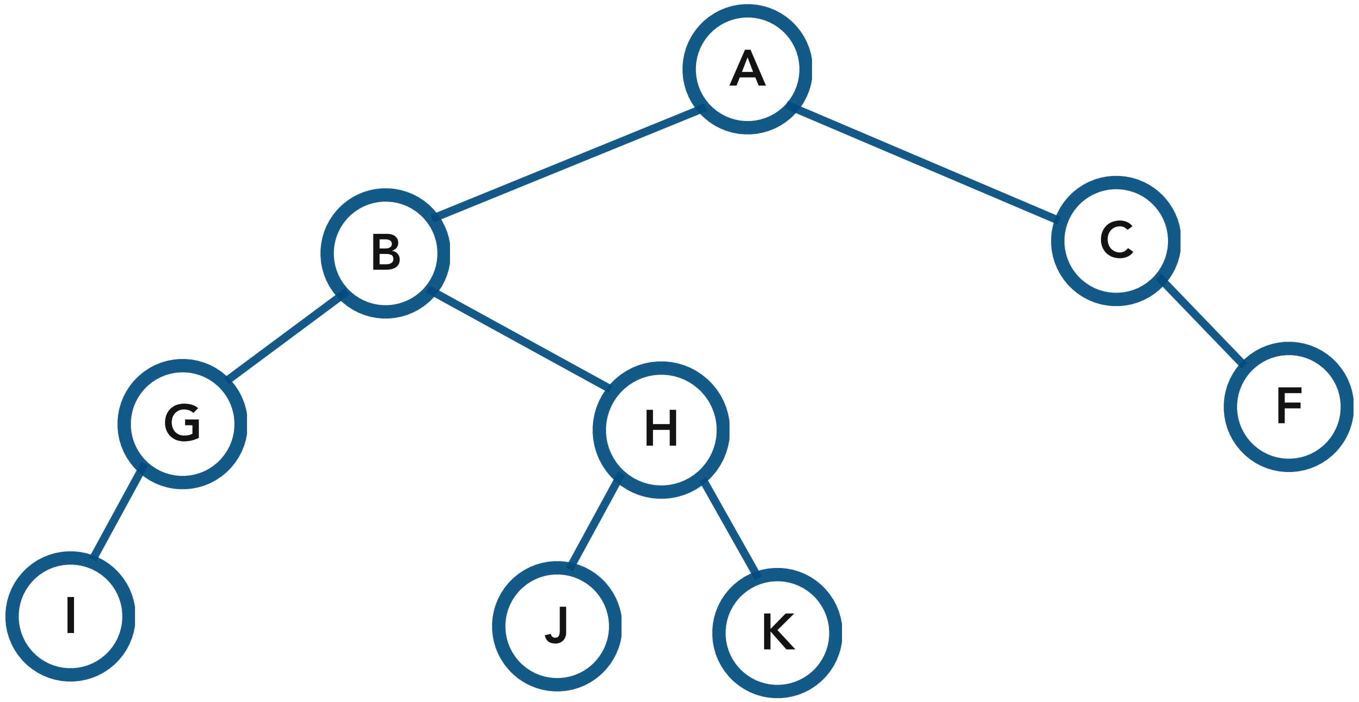
Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one.

Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

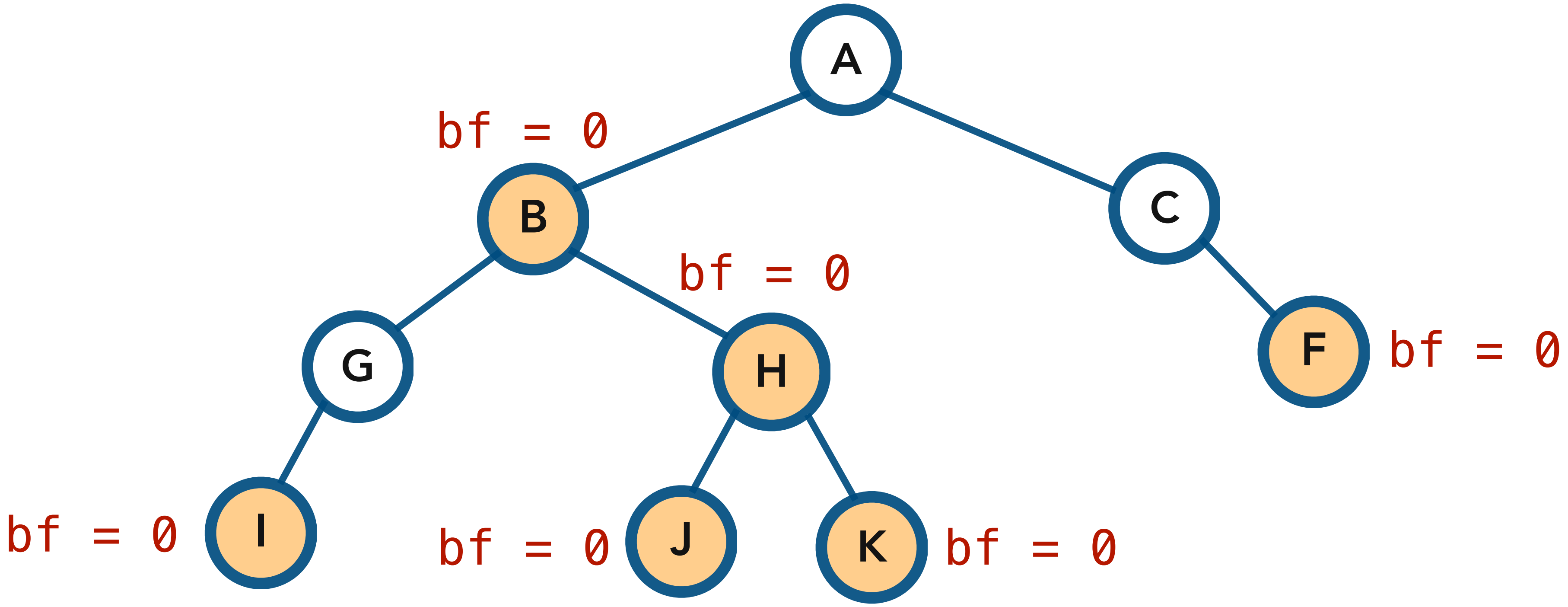
Balance Factor of a node (bf) = height of left child – height of right child



Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

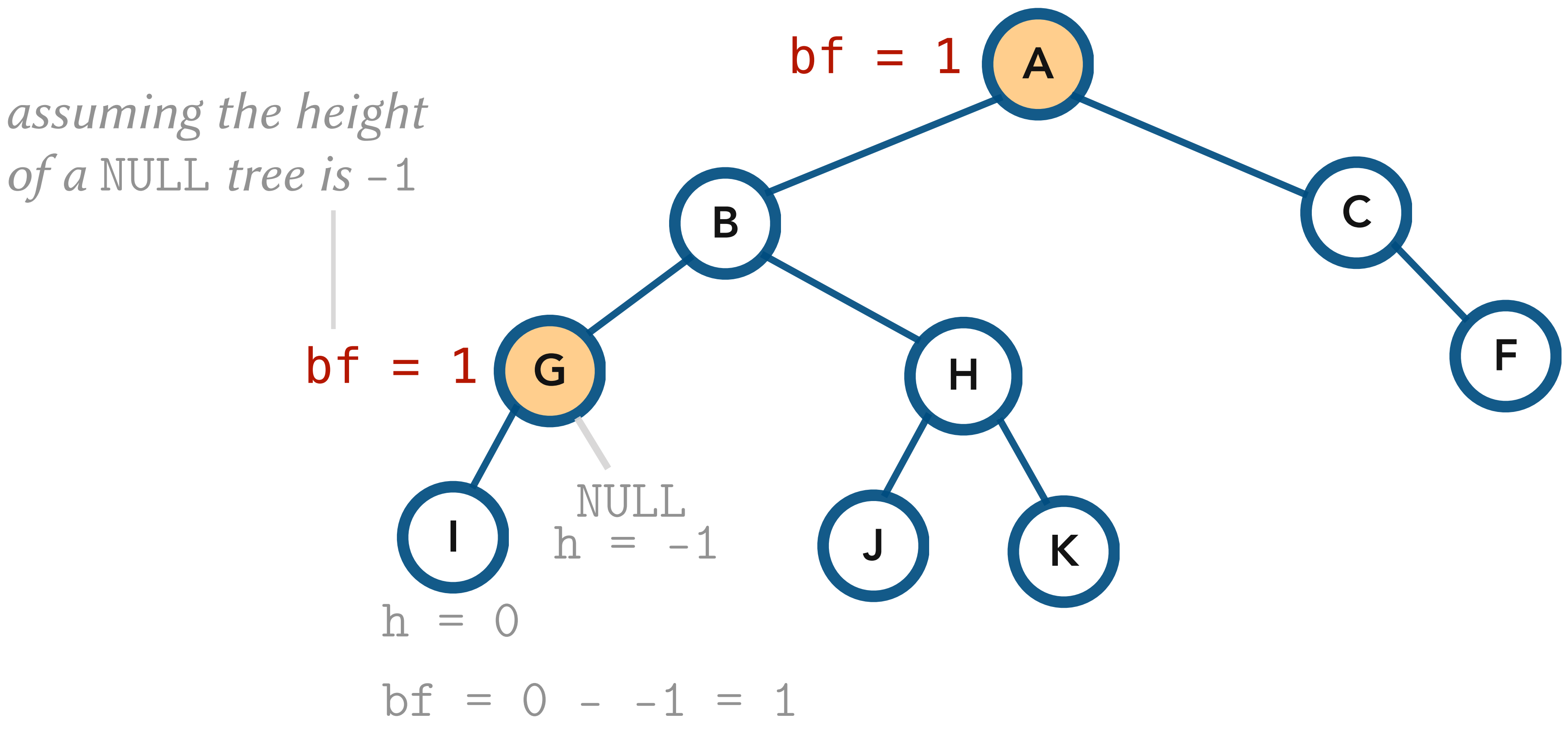
Balance Factor of a node (bf) = height of left child – height of right child



Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

Balance Factor of a node (bf) = height of left child – height of right child

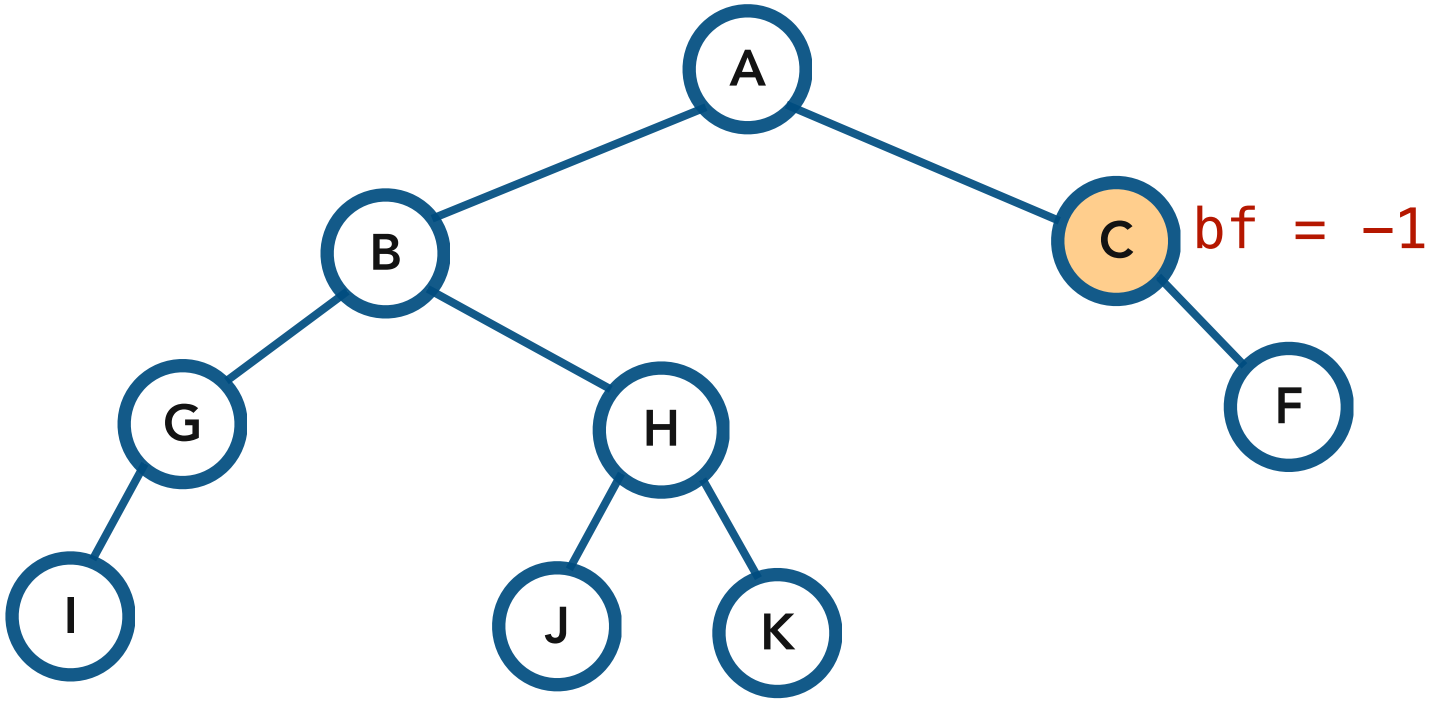


assuming the height of a NULL *tree is* -1

Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

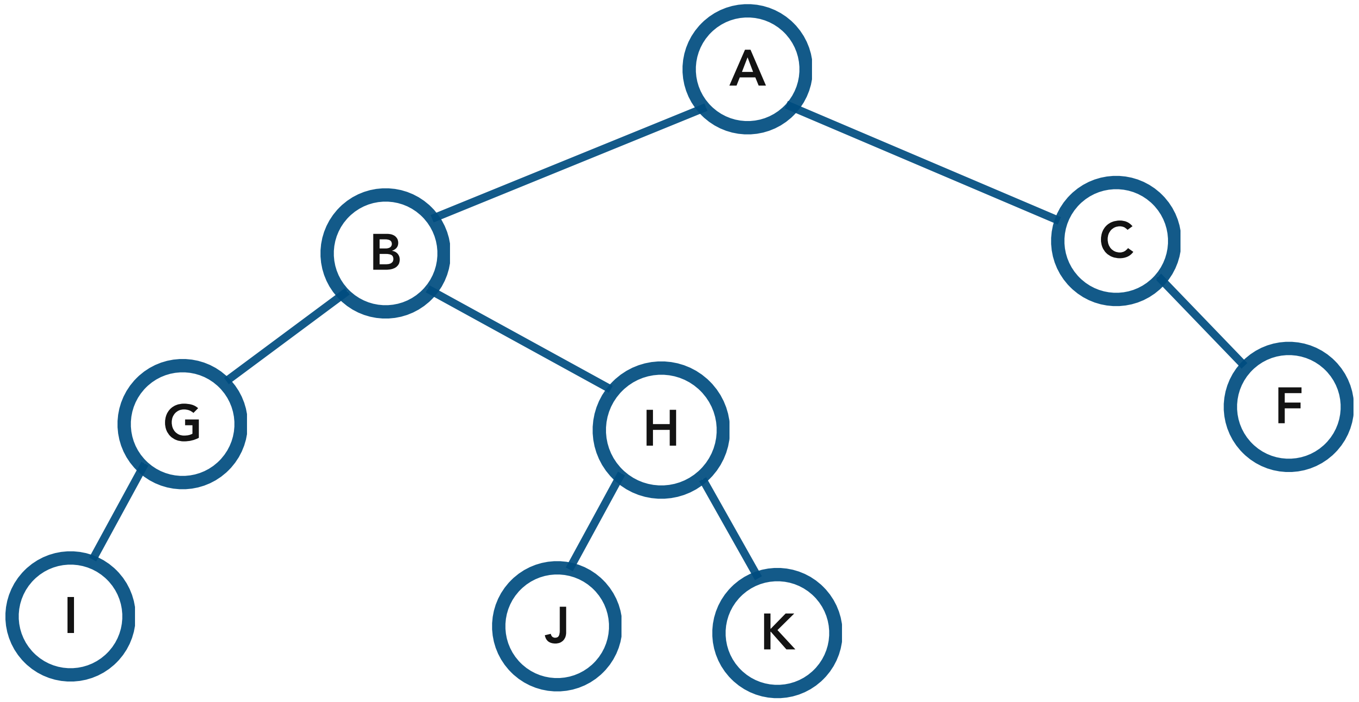
Balance Factor of a node (bf) = height of left child – height of right child



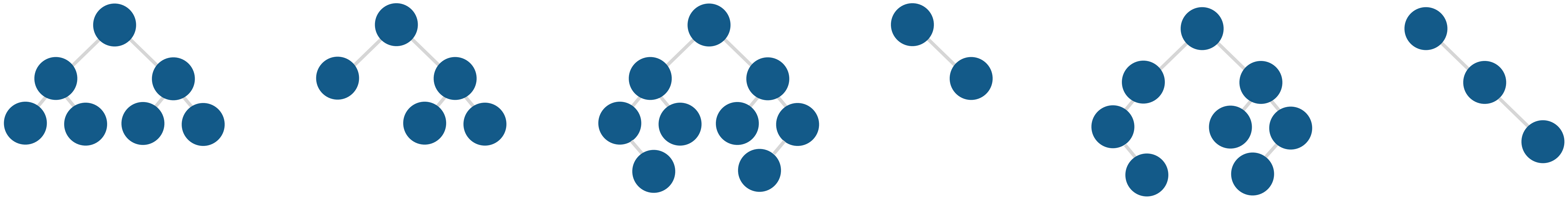
Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

Balance Factor of a node (bf) = height of left child – height of right child



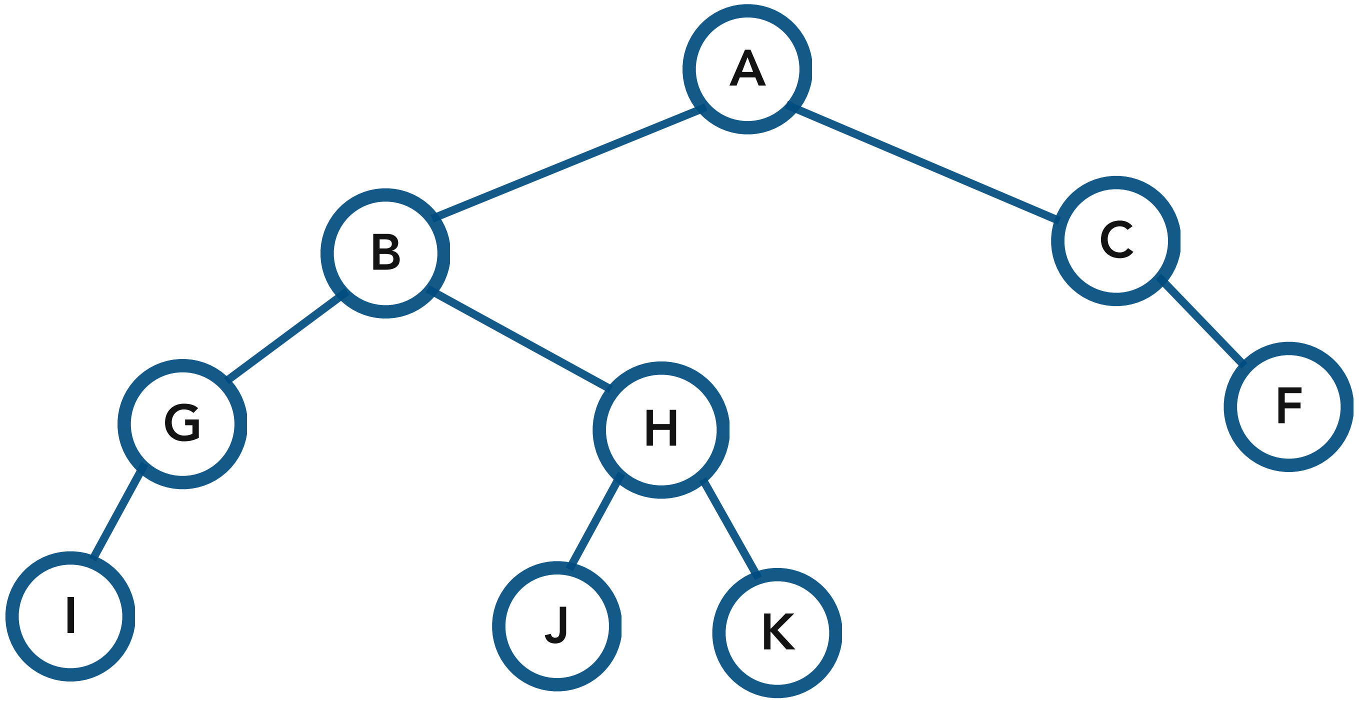
Which trees are balanced?



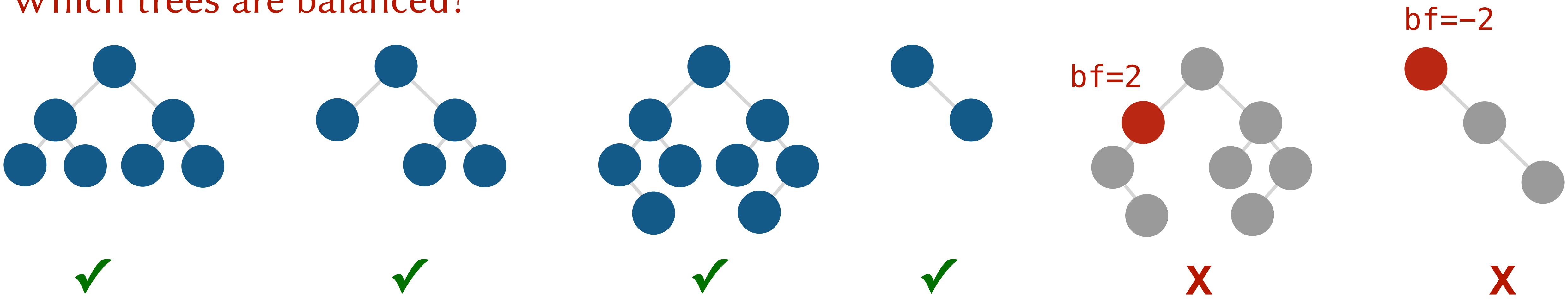
Balanced Binary Trees

Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

Balance Factor of a node (bf) = height of left child – height of right child



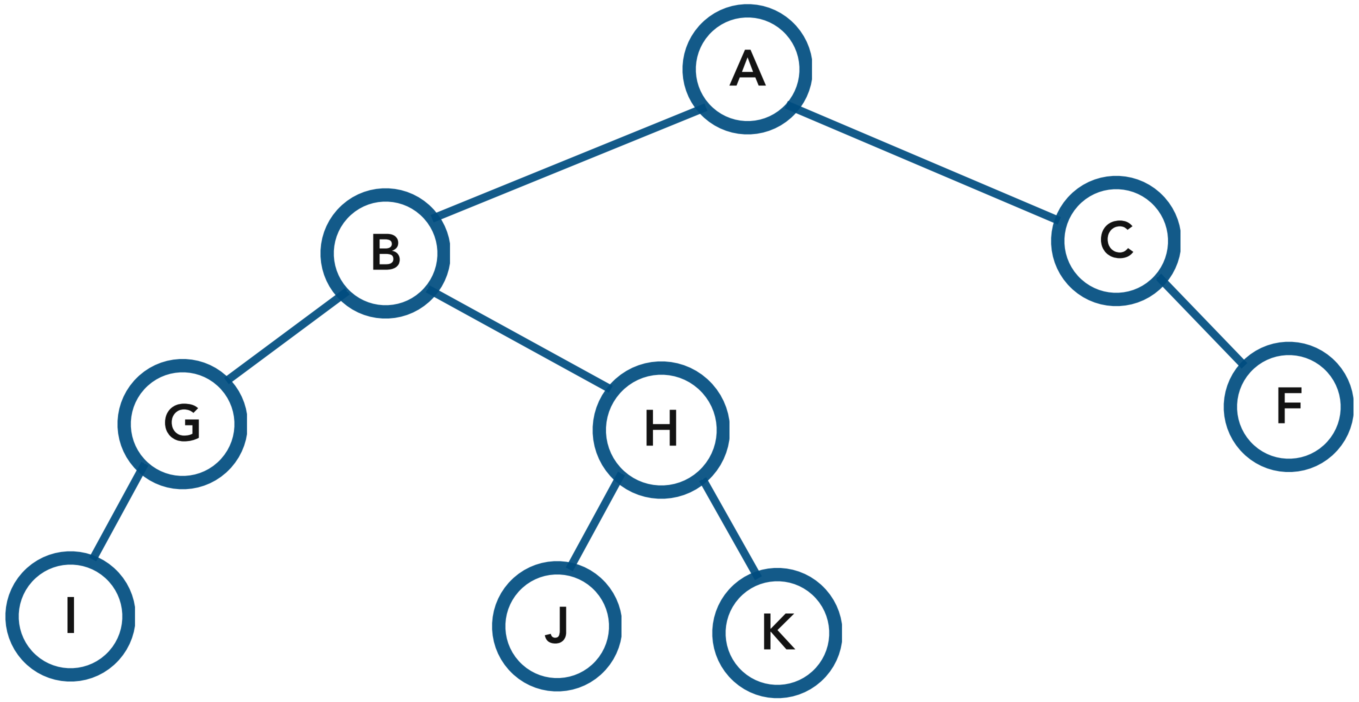
Which trees are balanced?



Balanced Binary Trees

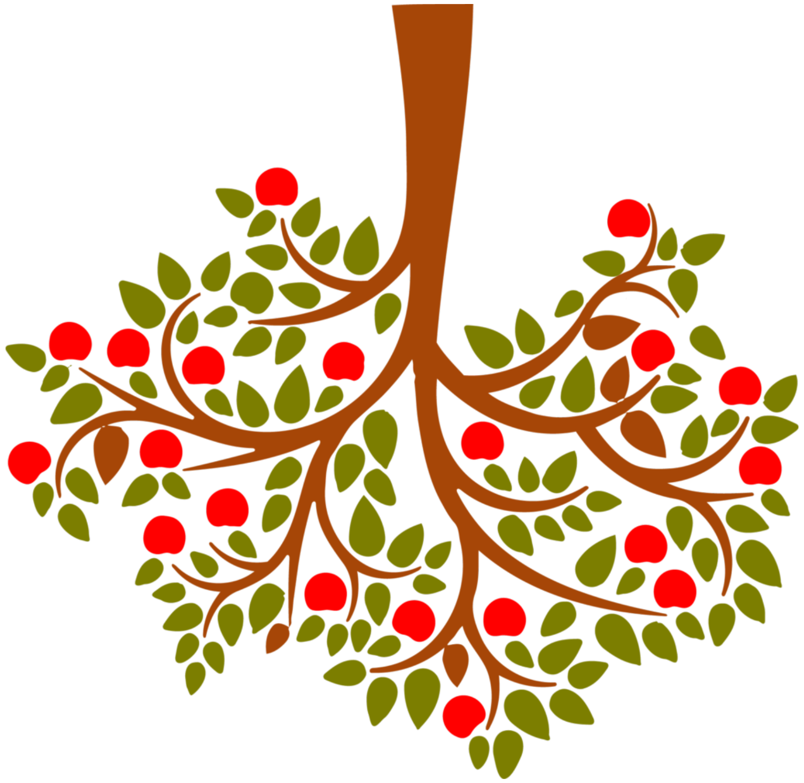
Balanced Binary Tree. the heights of the two child subtrees of any node differ by at most one. I.e. the balance factor for every node = 0, 1 or -1.

Balance Factor of a node (bf) = height of left child – height of right child



IMPORTANT!

The height of a *balanced* binary tree is *logarithmic* in the number of nodes in the tree.



Tree Data Structures

- Definitions and properties
- ● Basic Operations
- Balanced binary search trees
- Tree traversals