CS24 Week 7 Lecture 2

Kyle Dewey

Overview

- Binary search
- Binary search trees

Binary Search

Motivation

- Say we have an array holding a million elements in arbitrary order
- How might we determine if a given element is contained within?

Linear Search

- Looking through all elements is often called a *linear search* or a *linear scan*
- What is the time complexity of this?

Linear Search

- Looking through all elements is often called a *linear search* or a *linear scan*
- What is the time complexity of this?
  - `O(N)`

Optimization

- What if we have the same array contents, but now they are in sorted order
- How might we take advantage of this?

Binary Search

- Start looking at the middlemost element
- If our element we are looking for is less than the middle element, then repeat this process on the lefthand side of the data
- If greater, repeat on the righthand side
- If equal, we found it
- If we have no data to look at, the element is not contained within

Example 1

Binary Search

Looking for: 3

0	3	4	7	10	12	15

Binary Search

Looking for: 3

3 < 7?

↓

0	3	4	7	10	12	15

Binary Search

Looking for: 3

3 < 7?   true; look left

0	3	4	7	10	12	15

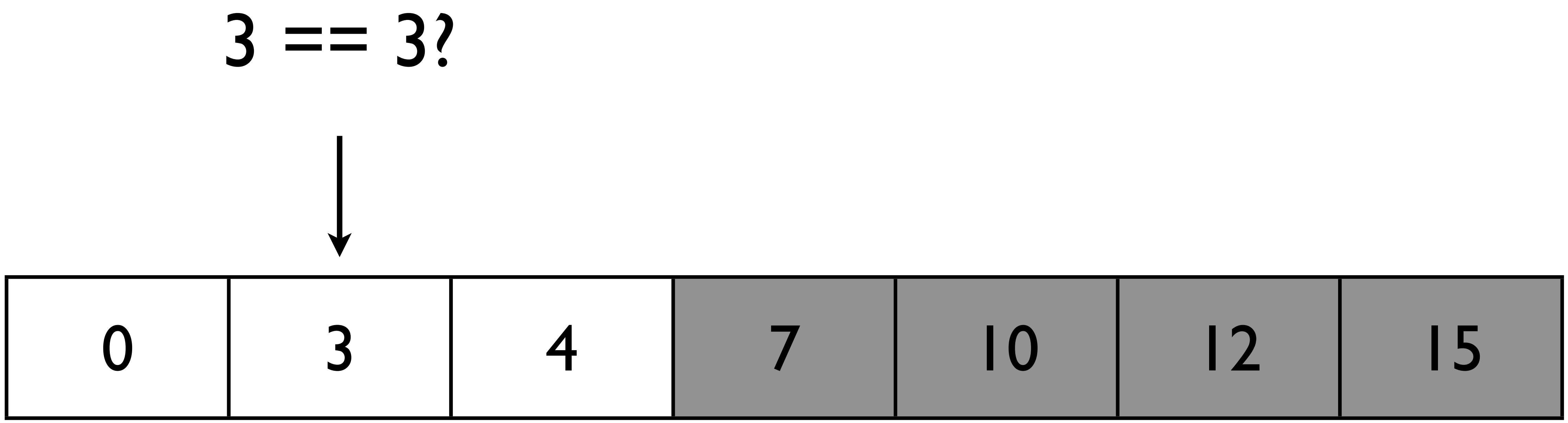
Binary Search

Looking for: 3

0	3	4	7	10	12	15

Binary Search

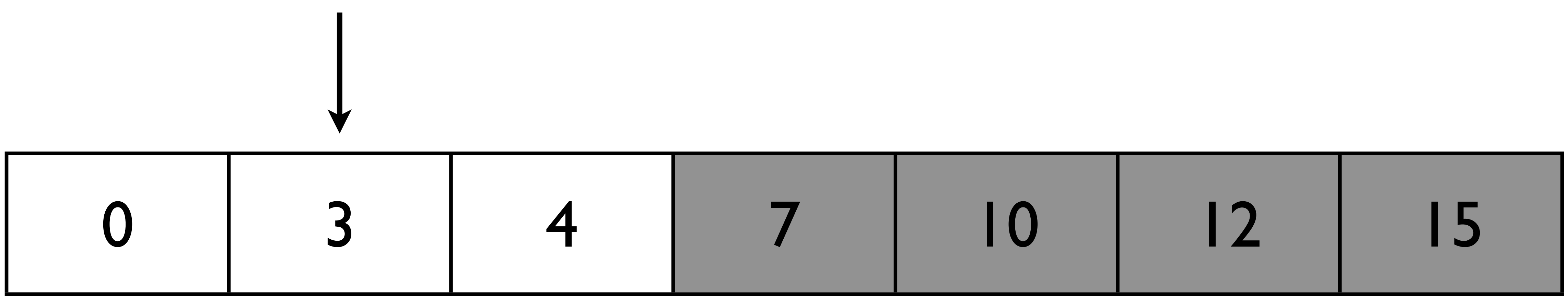
Looking for: 3



Binary Search

Looking for: 3

3 == 3? true; found it!



Example 2

Binary Search

Looking for: 10

0	3	4	7	10	12	15

Binary Search

Looking for: 10

10 < 7?

↓

0	3	4	7	10	12	15

Binary Search

Looking for: 10

10 < 7? false; look right

0	3	4	7	10	12	15

Binary Search

Looking for: 10

0	3	4	7	10	12	15

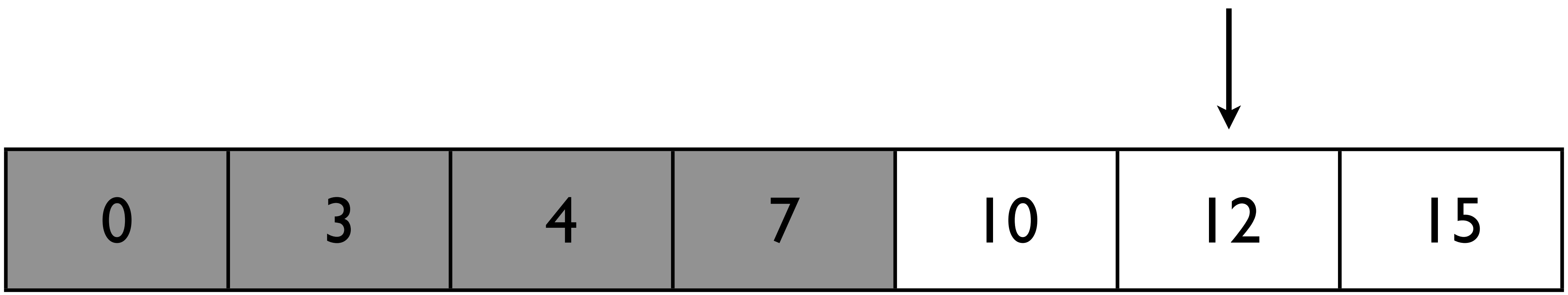
Binary Search

Looking for: 10

Binary Search

Looking for: 10

true; look left   10 < 12?



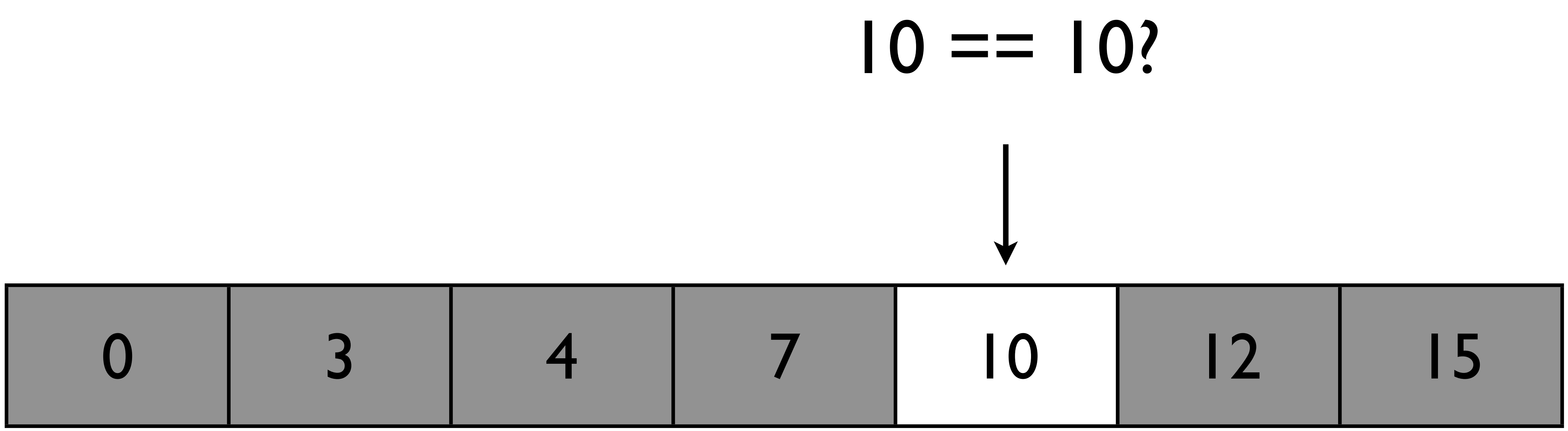
Binary Search

Looking for: 10

0	3	4	7	10	12	15

Binary Search

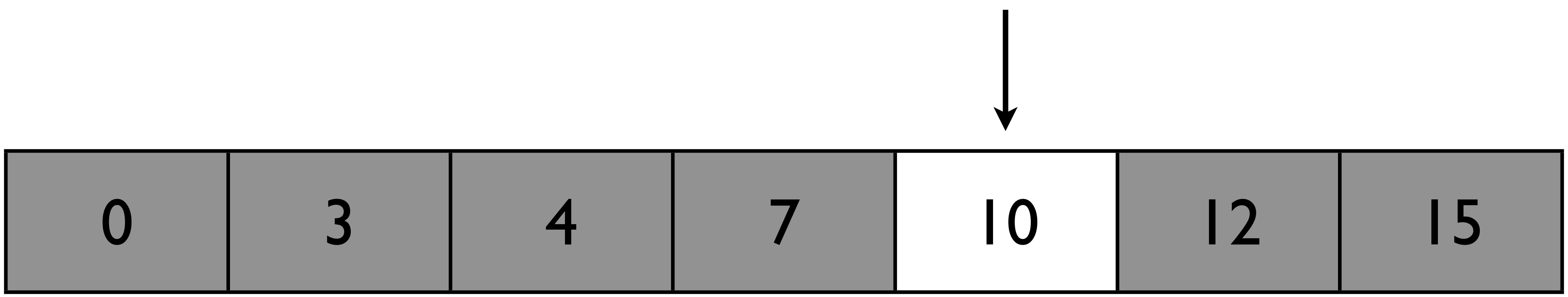
Looking for: 10



Binary Search

Looking for: 10

true; found it!   10 == 10?



Example 3

Binary Search

Looking for: 5

0	3	4	7	10	12	15

Binary Search

Looking for: 5

5 < 7?

↓

0	3	4	7	10	12	15

Binary Search

Looking for: 5

true; look left   5 < 7?

↓

0	3	4	7	10	12	15

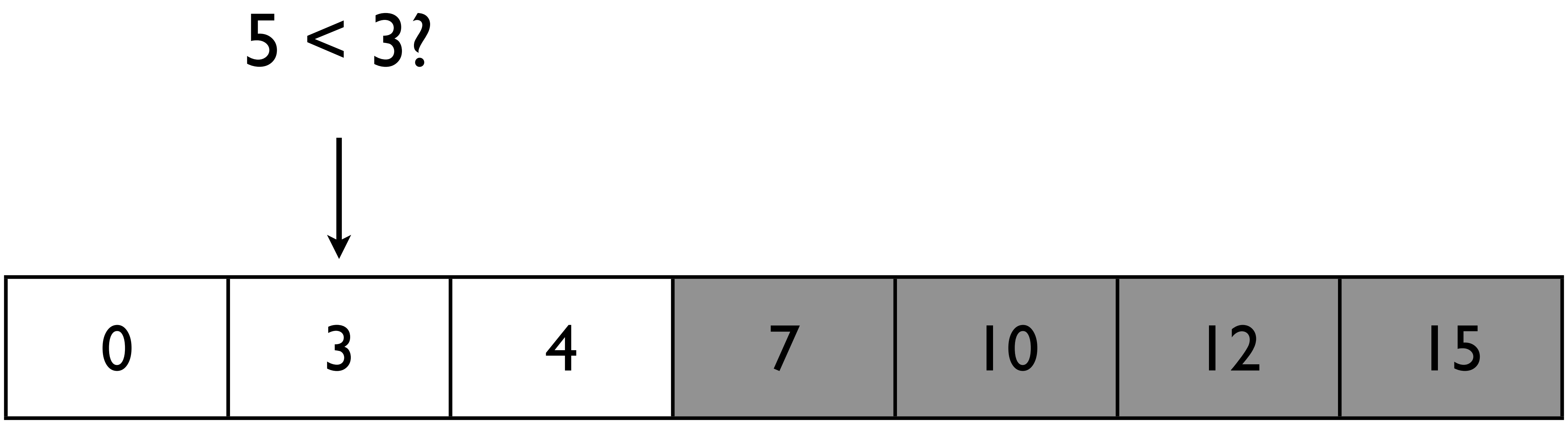
Binary Search

Looking for: 5

0	3	4	7	10	12	15

Binary Search

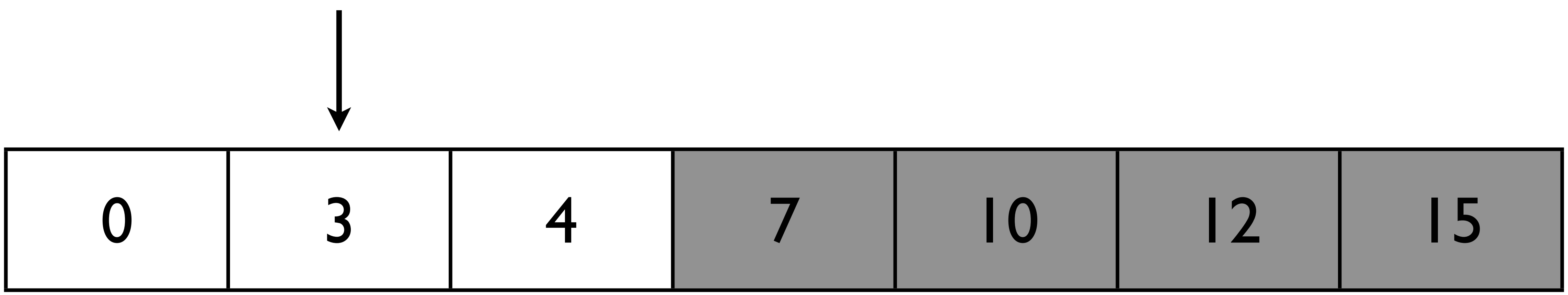
Looking for: 5



Binary Search

Looking for: 5

5 < 3? false; look right



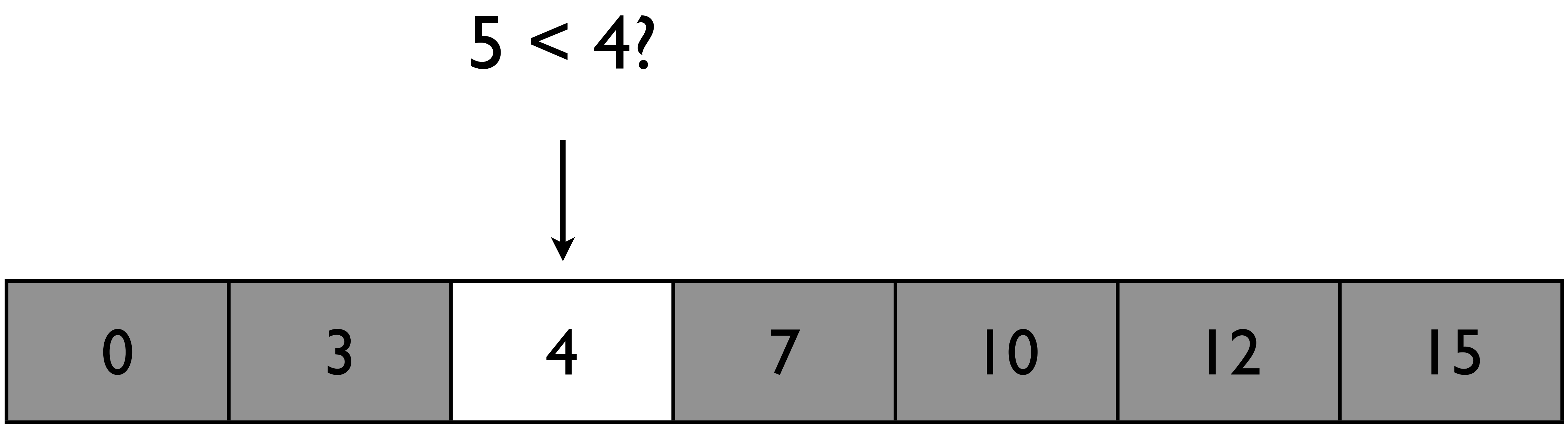
Binary Search

Looking for: 5

0	3	4	7	10	12	15

Binary Search

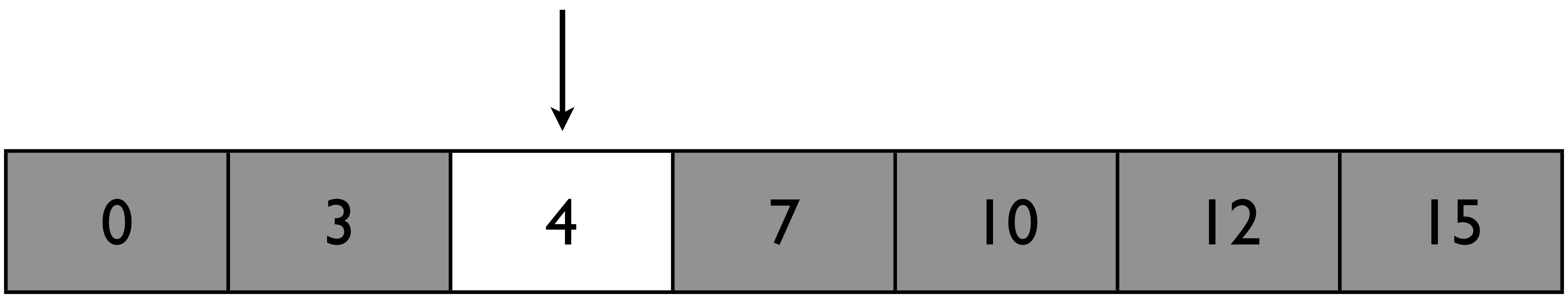
Looking for: 5



Binary Search

Looking for: 5

5 < 4?   false; look right



Binary Search

Looking for: 5

0	3	4	7	10	12	15

Binary Search

Looking for: 5

No possibilities remain - 5 is not within the array

0	3	4	7	10	12	15

Time Complexity

- Binary search has a special property: at each step, the total size of the input is cut in half
- Does this influence the time complexity?

Time Complexity

- Binary search has a special property: at each step, the total size of the input is cut in half
- Does this influence the time complexity?
  - Yes. An input size of `N` which is cut in half repeatedly shrinks rapidly

Time Complexity

- Repeatedly doubling something gets an exponential time complexity
- Here we do the opposite
- We end up with a logarithmic time complexity - `O(log(N))`

Arrays vs. Linked Lists

- We've been showing this for arrays, not for linked lists
- What sort of issues would a linked list representation have?

Arrays vs. Linked Lists

- We've been showing this for arrays, not for linked lists
- What sort of issues would a linked list representation have?
  - Cannot jump to a node in `O(1)`, instead is `O(N)`

Binary Search With Linked Lists

- Binary search is `O(log(N))` with arrays
- Accessing an arbitrary element of a linked list is `O(N)`
- What time complexity would binary search have on linked lists?

Binary Search With Linked Lists

- Binary search is `O(log(N))` with arrays
- Accessing an arbitrary element of a linked list is `O(N)`
- What time complexity would binary search have on linked lists?
    - `O(N * log(N))` - worse than linear search!

Binary Search Trees

Motivation

Problem Setup

- Consider Facebook, with ~1 billion users
  - Users added frequently
  - Users search for each other by name
- Addition and search should take milliseconds at most

Representation

- Addition and search should take milliseconds at most
  - What is wrong with an array?
  - What is wrong with a linked list?

Optimizing Addition

- Users should be able to be added within milliseconds
- How can we make this happen?

Optimizing Addition

- Users should be able to be added within milliseconds
- How can we make this happen?
  - Linked lists work well

Optimizing Search

- Users want to be able to search for other users by name within milliseconds
- How can we speed up search?

Optimizing Search

- Users want to be able to search for other users by name within milliseconds
- How can we speed up search?
  - Use binary search on an array

Conflicting Problems

- For rapid search, we want arrays
- For rapid addition, we want linked lists
- Need elements of both

Combining Both

- For rapid addition, linked data structures are best, like linked lists
- For rapid search, we need a way to split data in half efficiently, specifically in `O(1)`
- Let's revisit the binary search example and see what we can get out of it

0	3	4	7	10	12	15

Combining Both

- The lack of links prevents easy addition
    - We need links *somewhere*
- We need a way to quickly split data in half
- Any ideas?

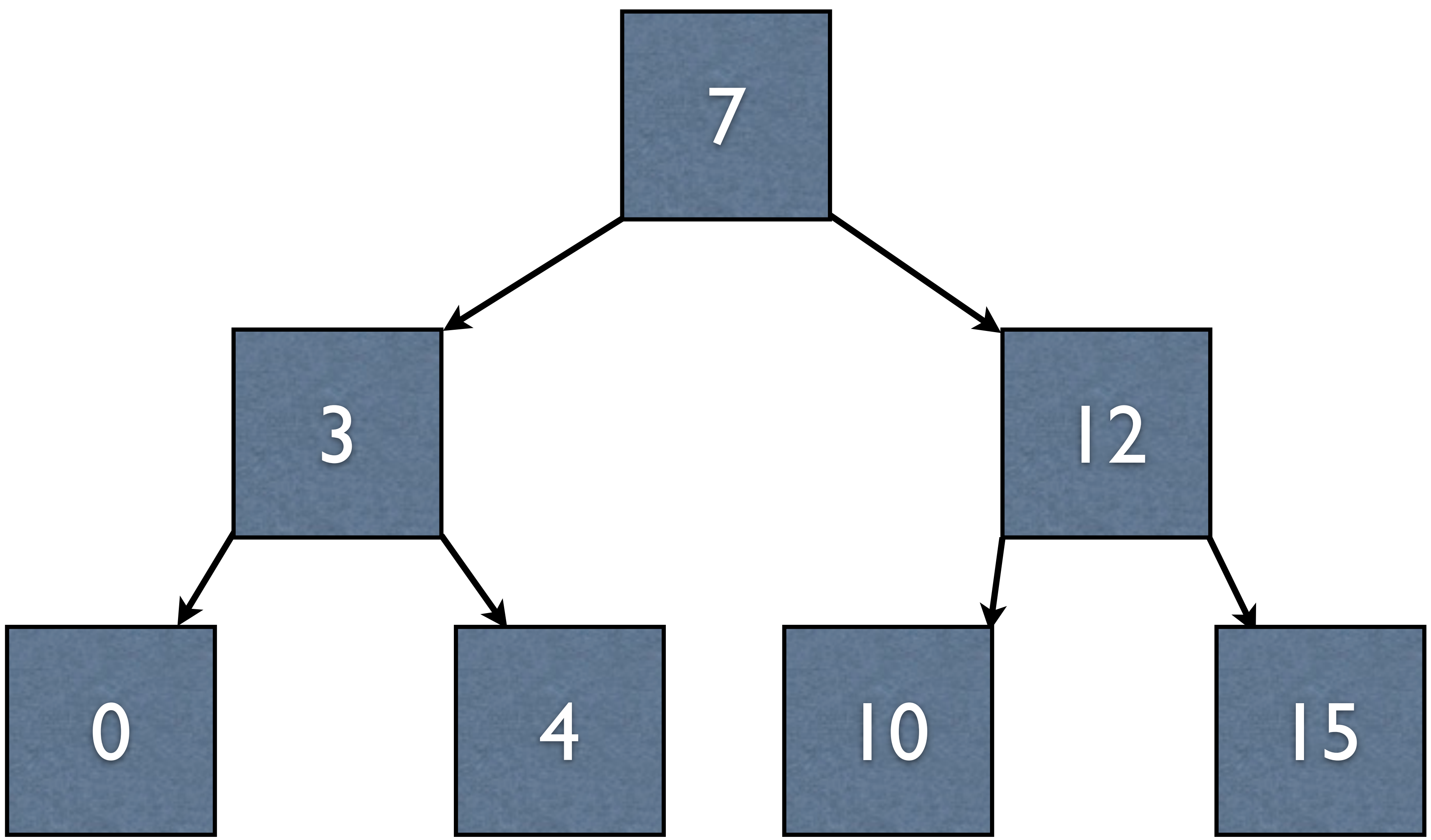
0	3	4	7	10	12	15

- Idea: add links at points which would split the data in half

0	3	4	7	10	12	15

- Idea: add links at points which would split the data in half
  - Needs two links per node

0	3	4	7	10	12	15



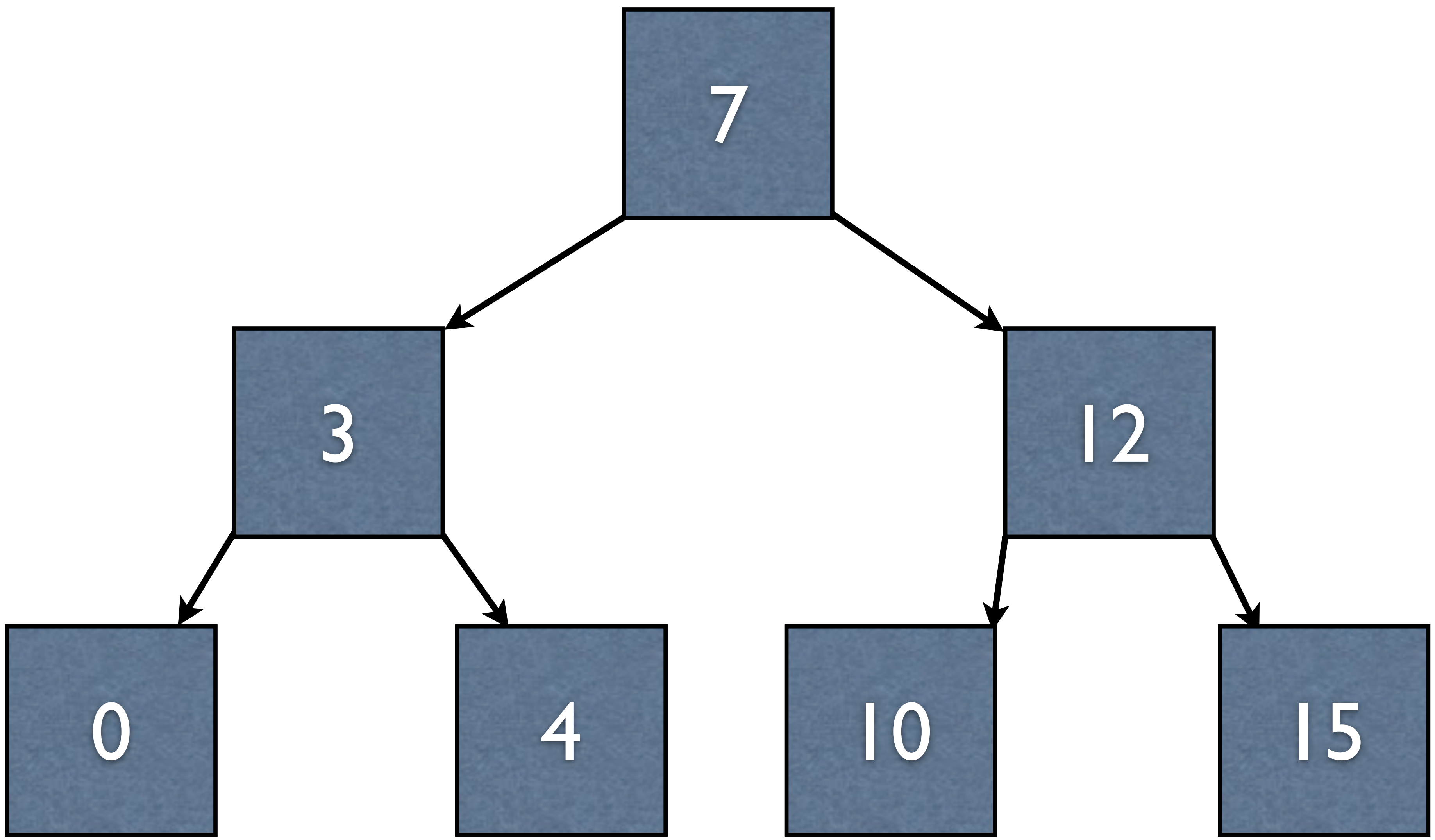
Binary Search Tree

- This representation is known as a *binary search tree*
  - Binary: each node has two child nodes
  - Search: search is efficient
  - Tree: forms a tree (each node has at most one parent)

Search Example

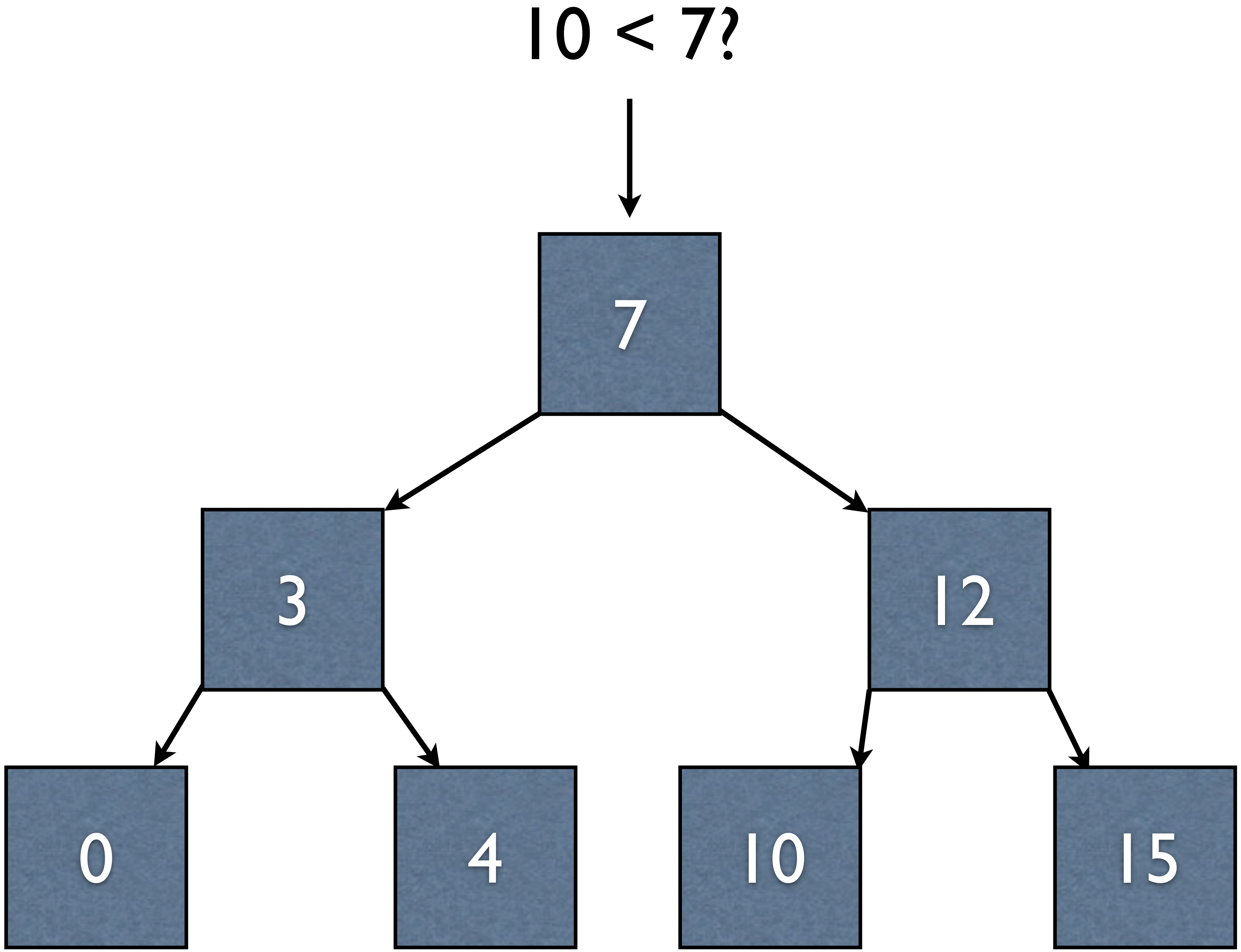
Search

Looking for: 10



Search

Looking for: 10

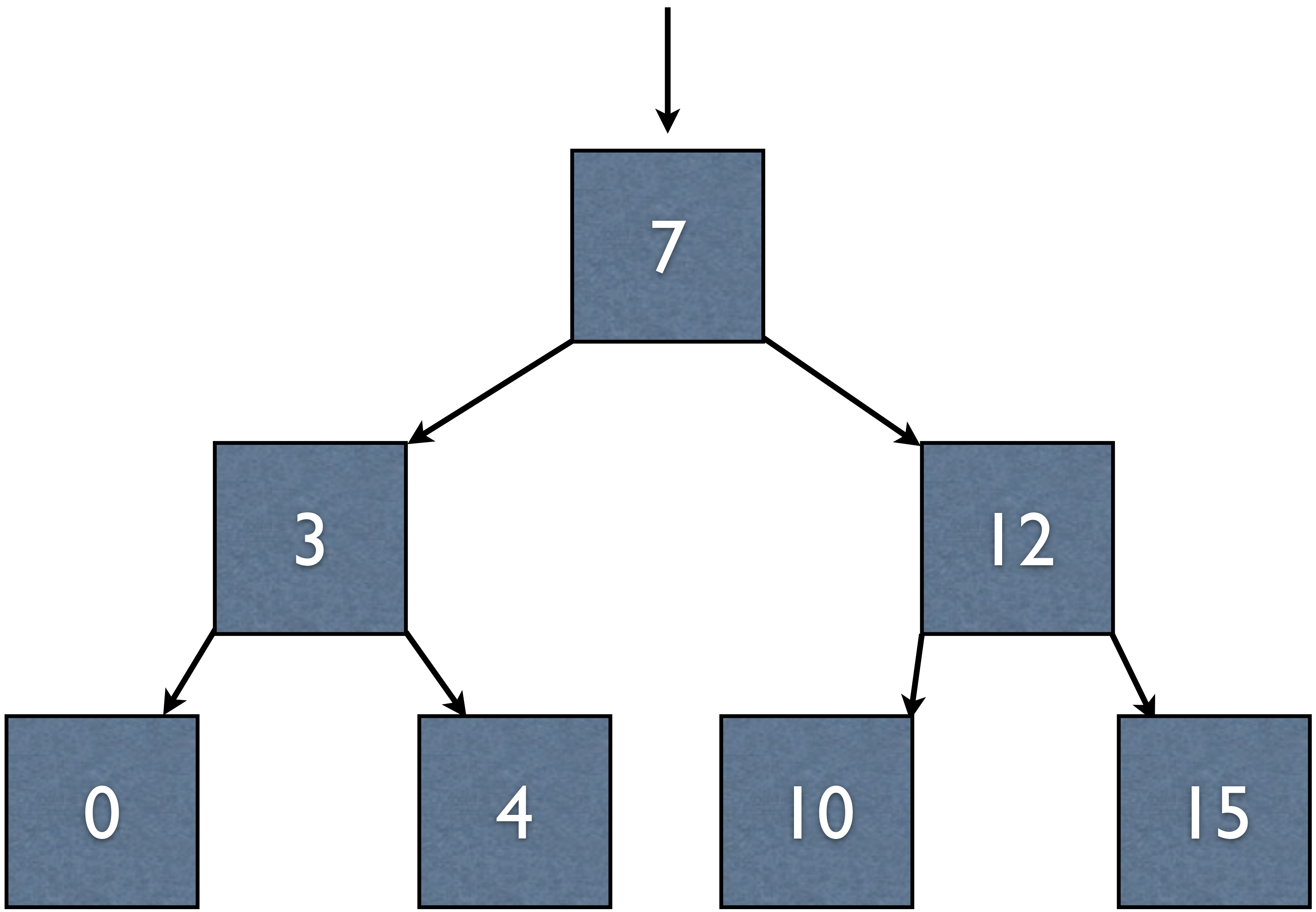


Search

Looking for: 10

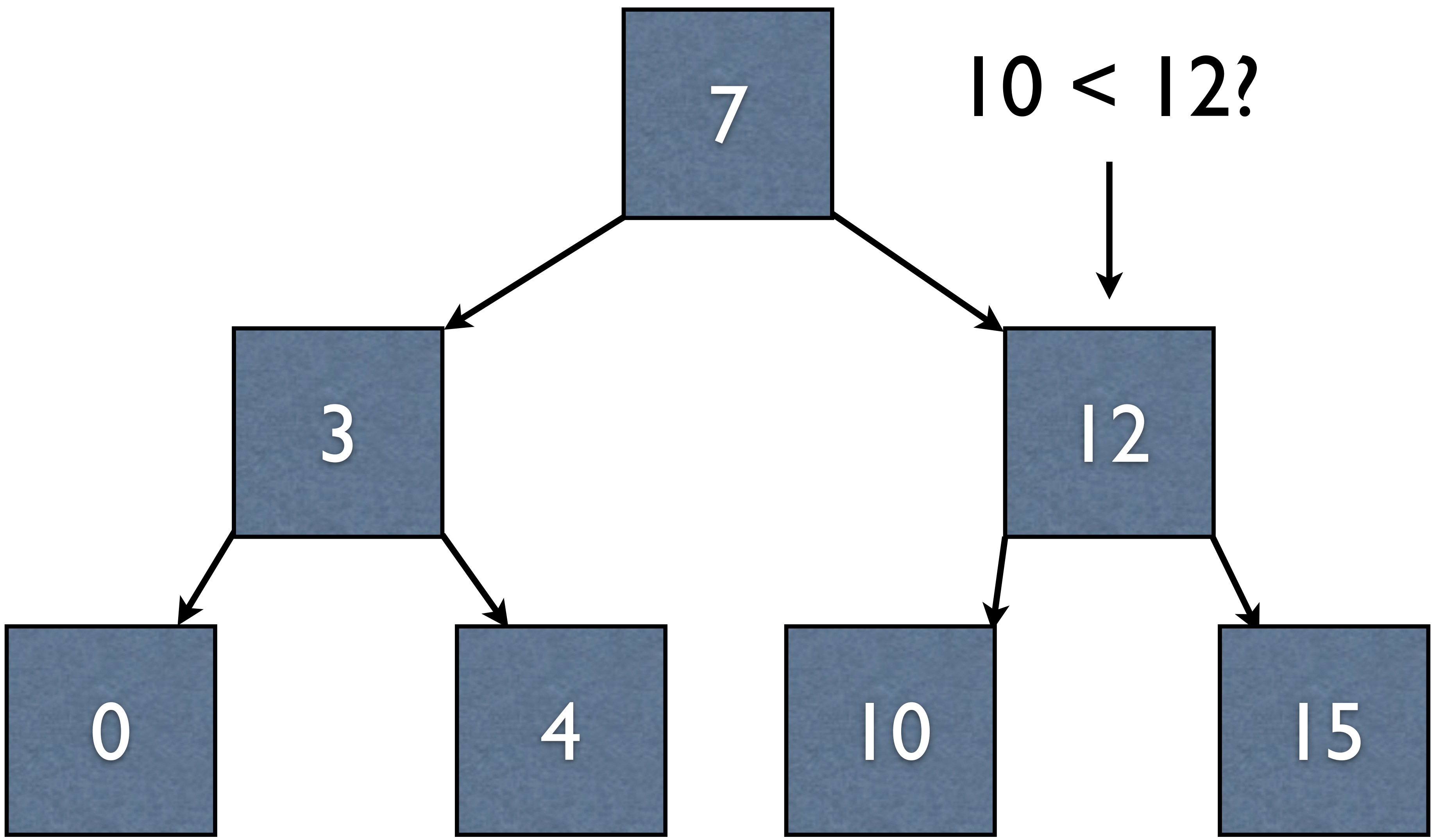
10 < 7?

false; look right



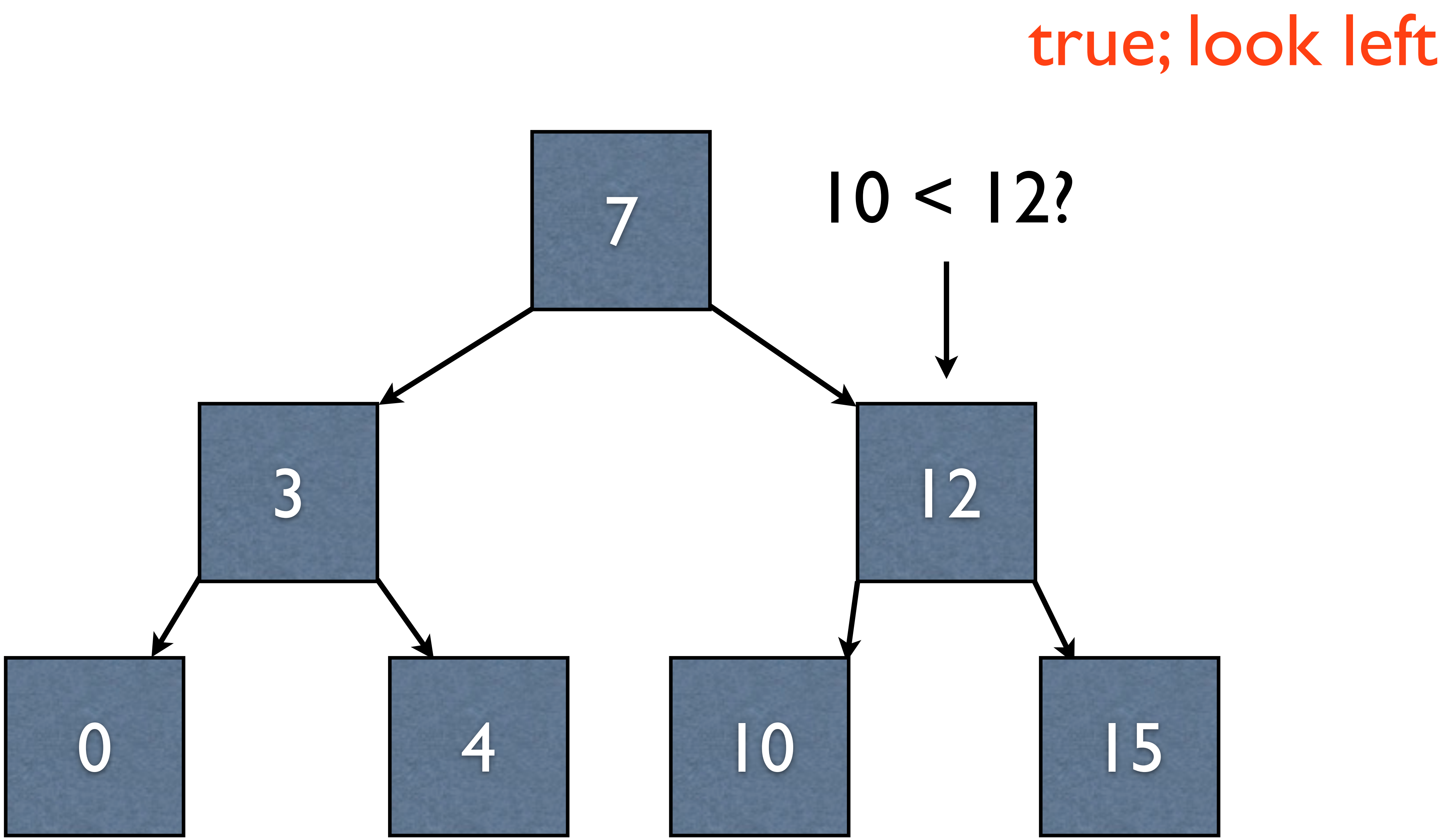
Search

Looking for: 10



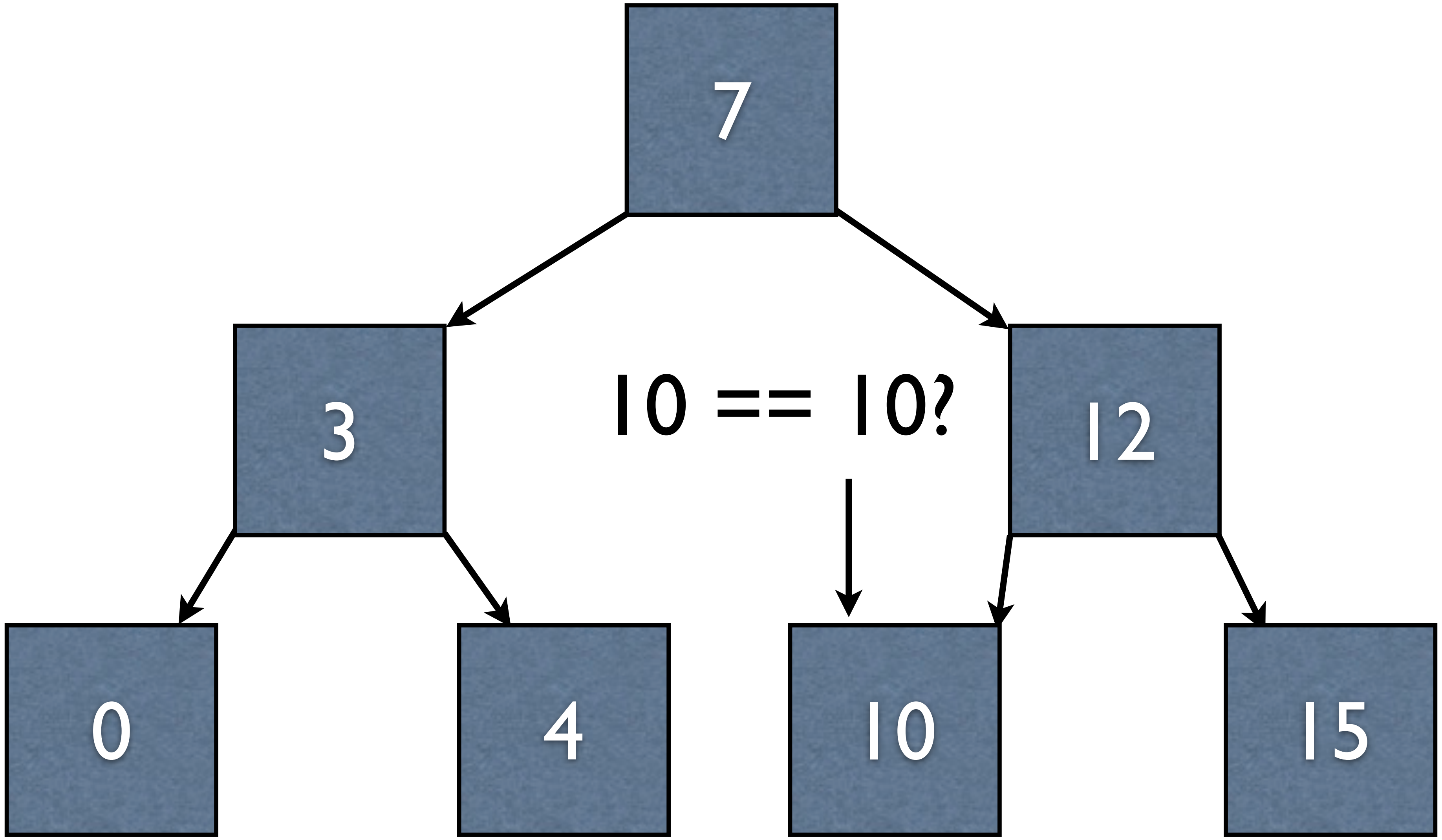
Search

Looking for: 10



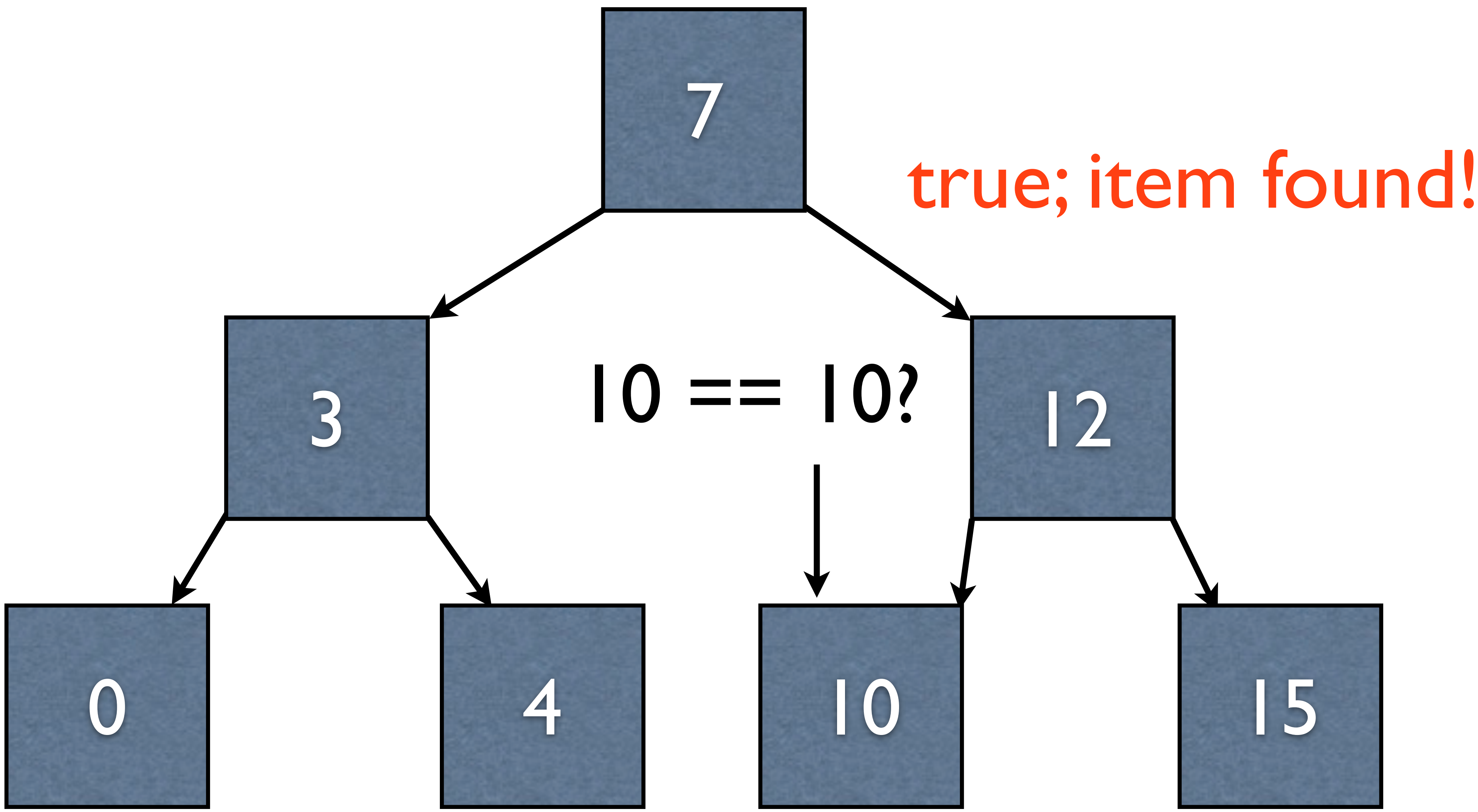
Search

Looking for: 10



Search

Looking for: 10



On Search

- At each point, we still cut the input in half
- Now, in order to get to the next half, we simply traverse a link - `O(1)`
- Search is overall `O(log(N))` as shown

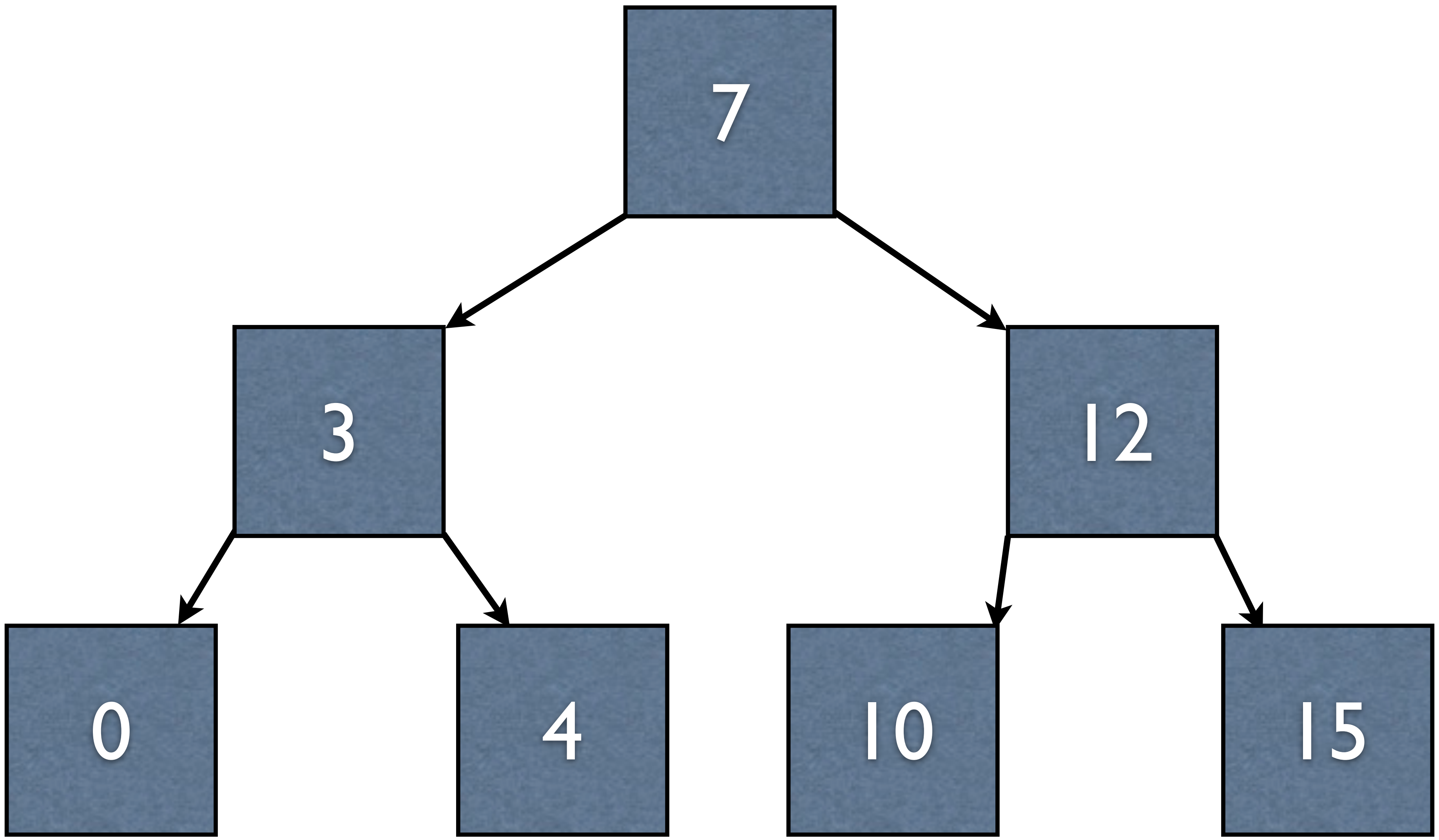
Insertion

- Nodes need to be inserted in sorted order
- While duplicates are possible with some forms of trees, we consider a tree where duplicates are impossible
  - Trying to insert a duplicate changes nothing in the tree

Insertion Example

Insertion

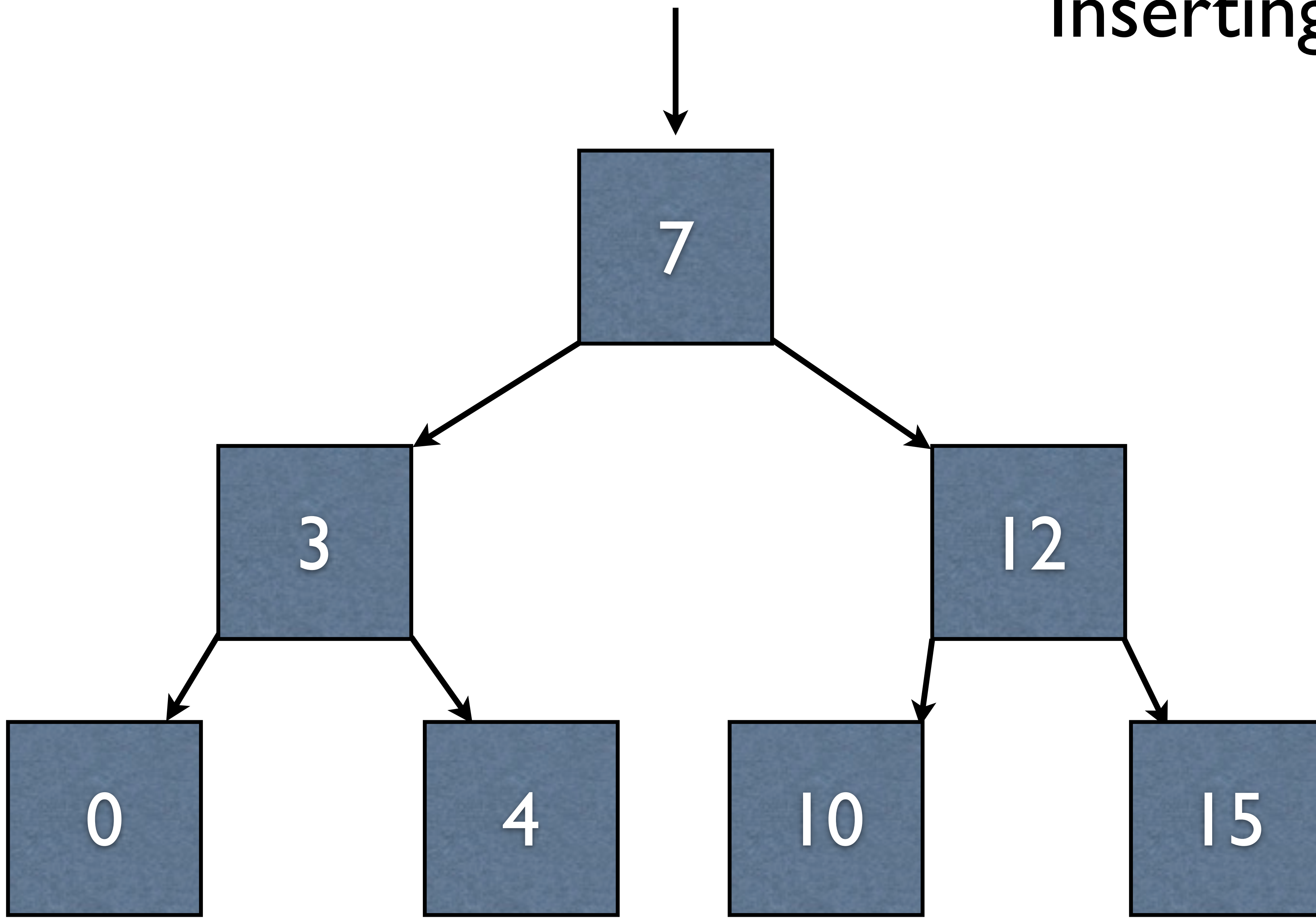
Inserting: 5



Insertion

5 < 7?

Inserting: 5

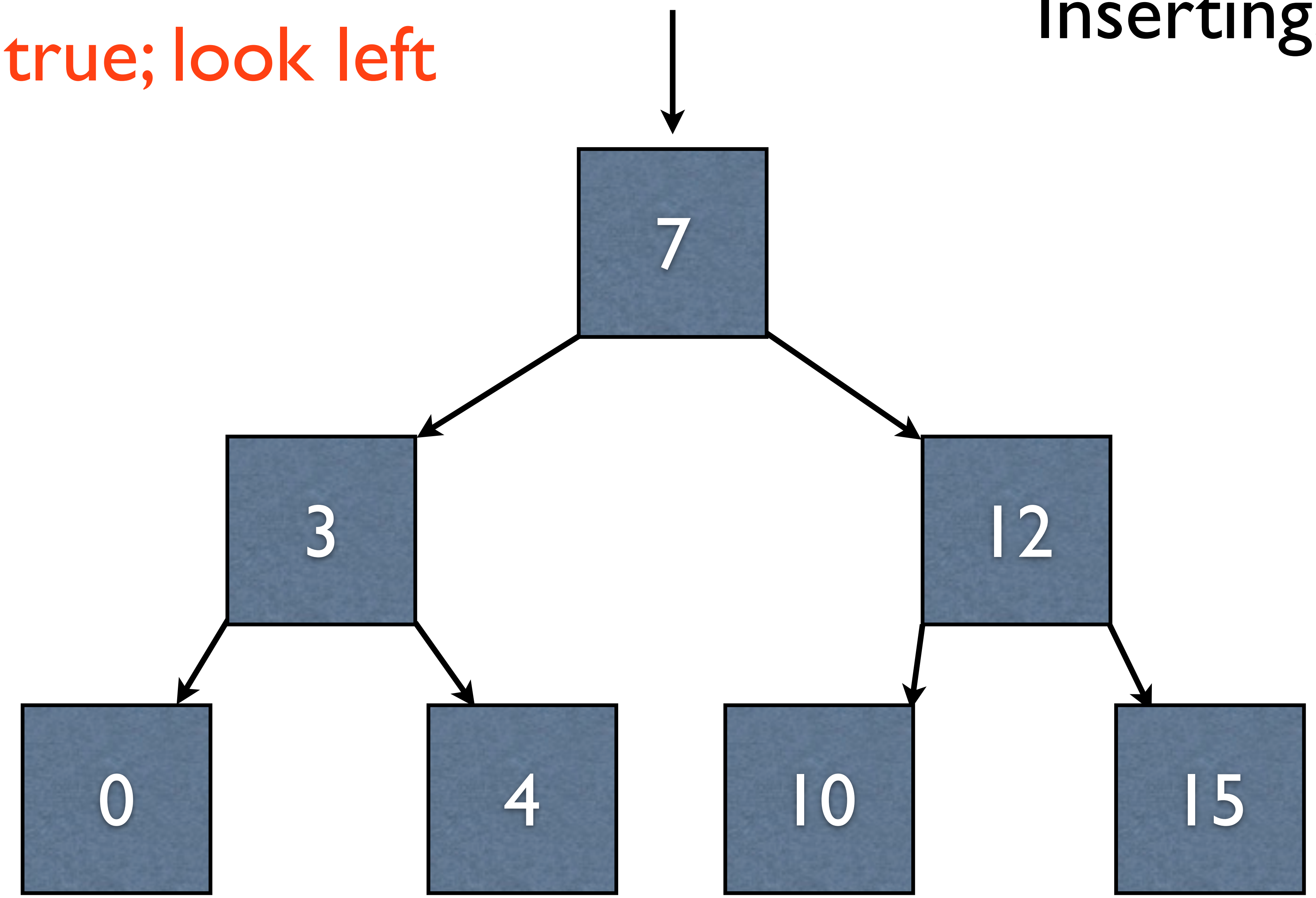


Insertion

5 < 7?

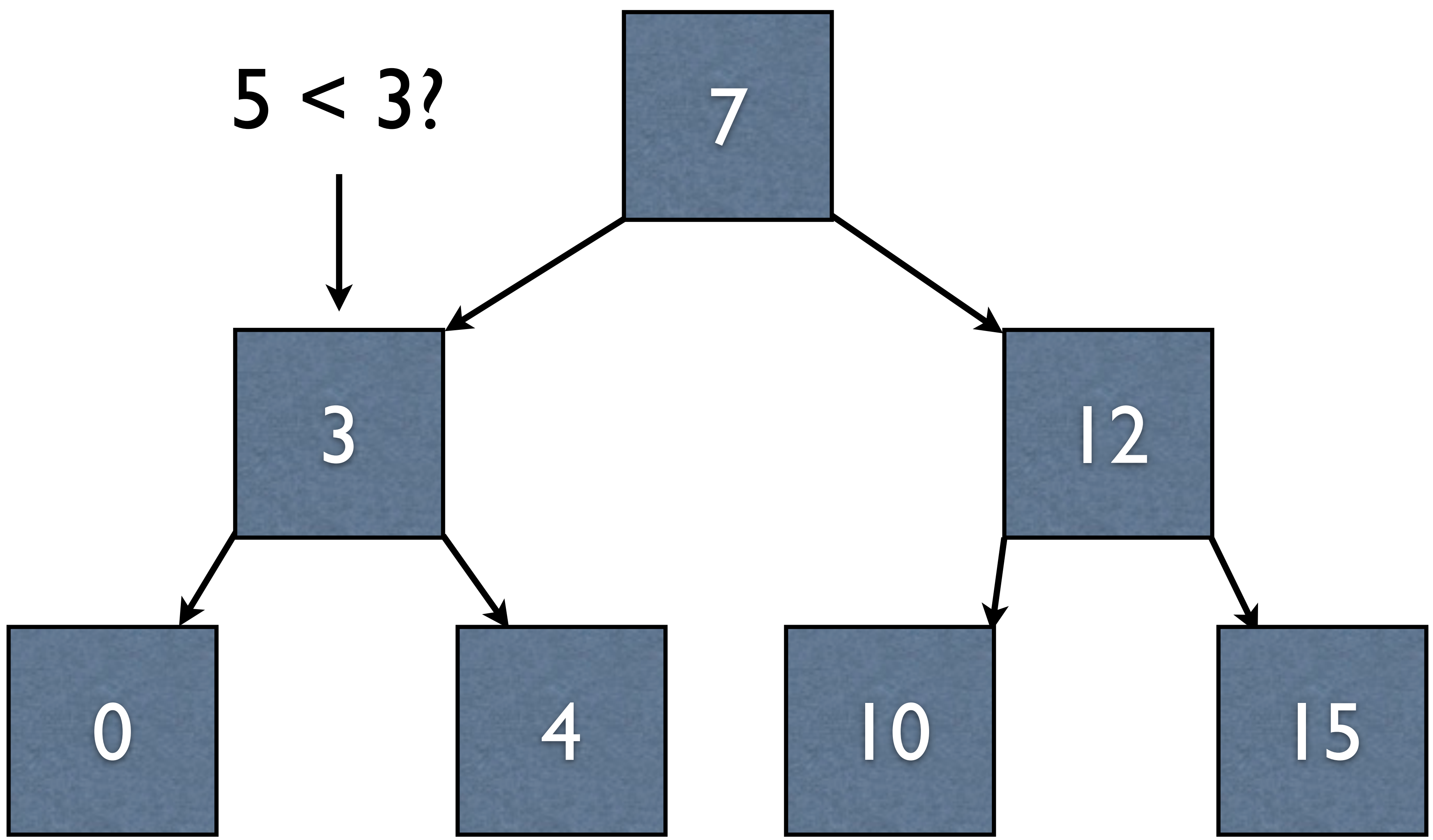
Inserting: 5

true; look left



Insertion

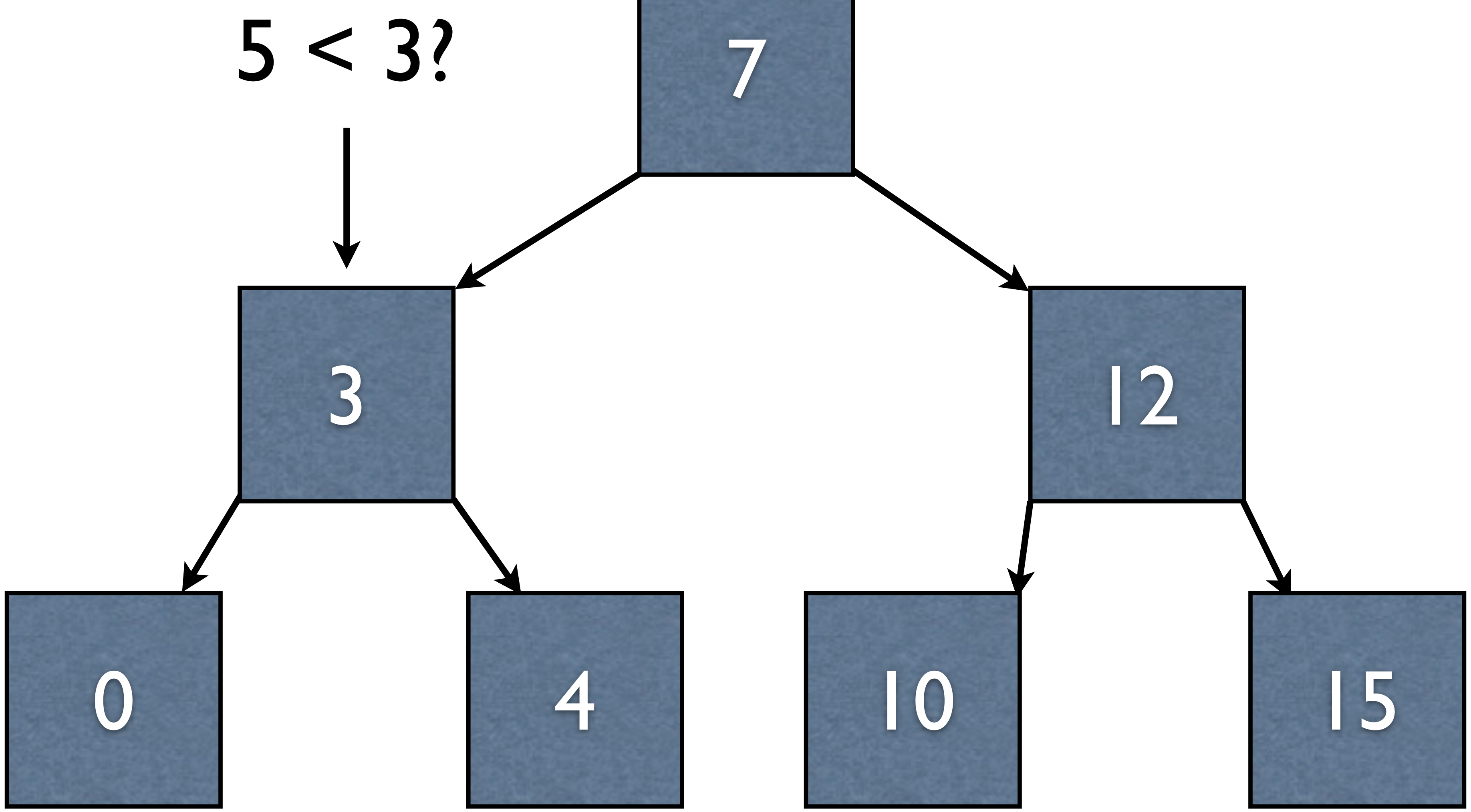
Inserting: 5



Insertion

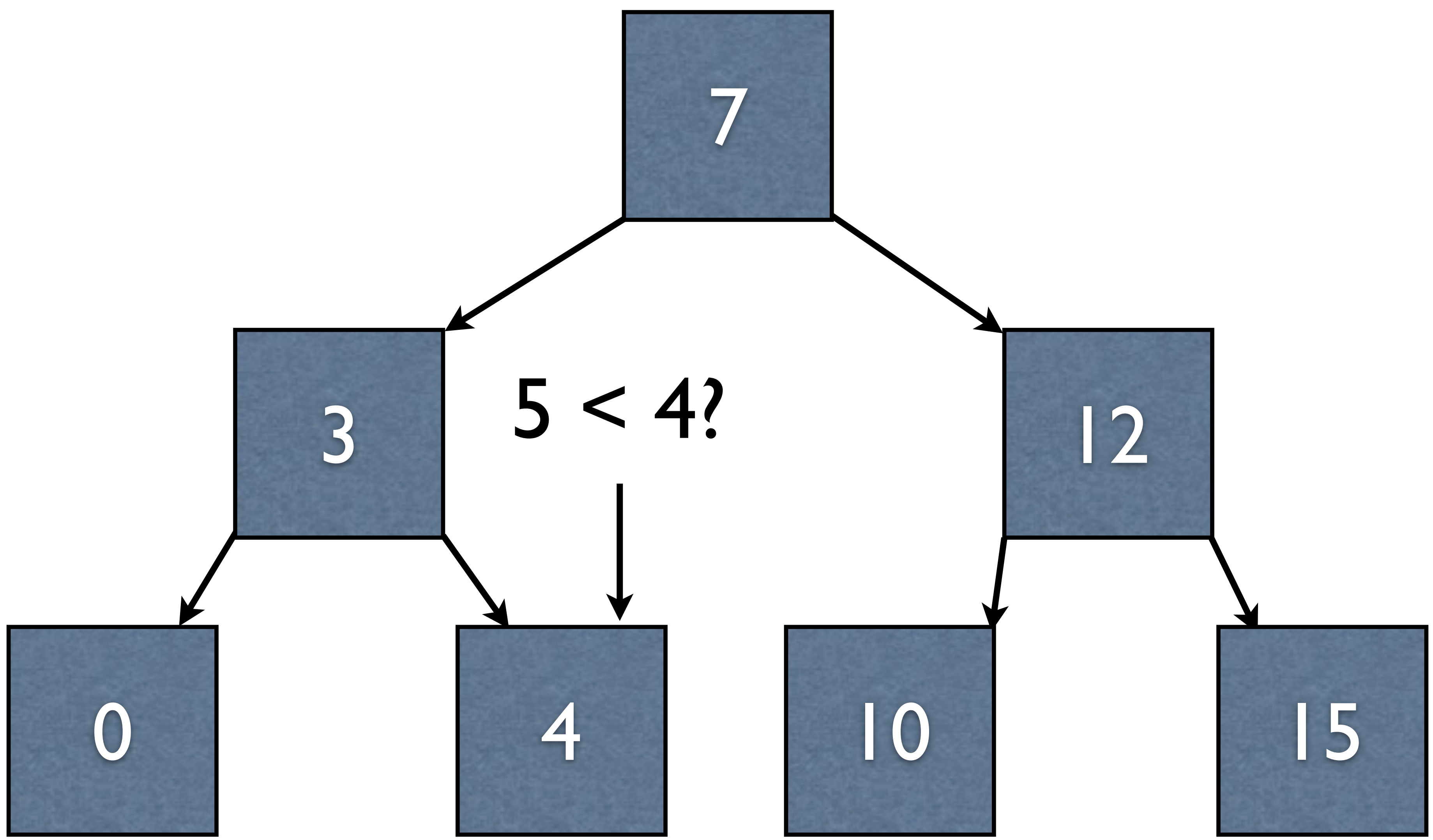
Inserting: 5

false; look right



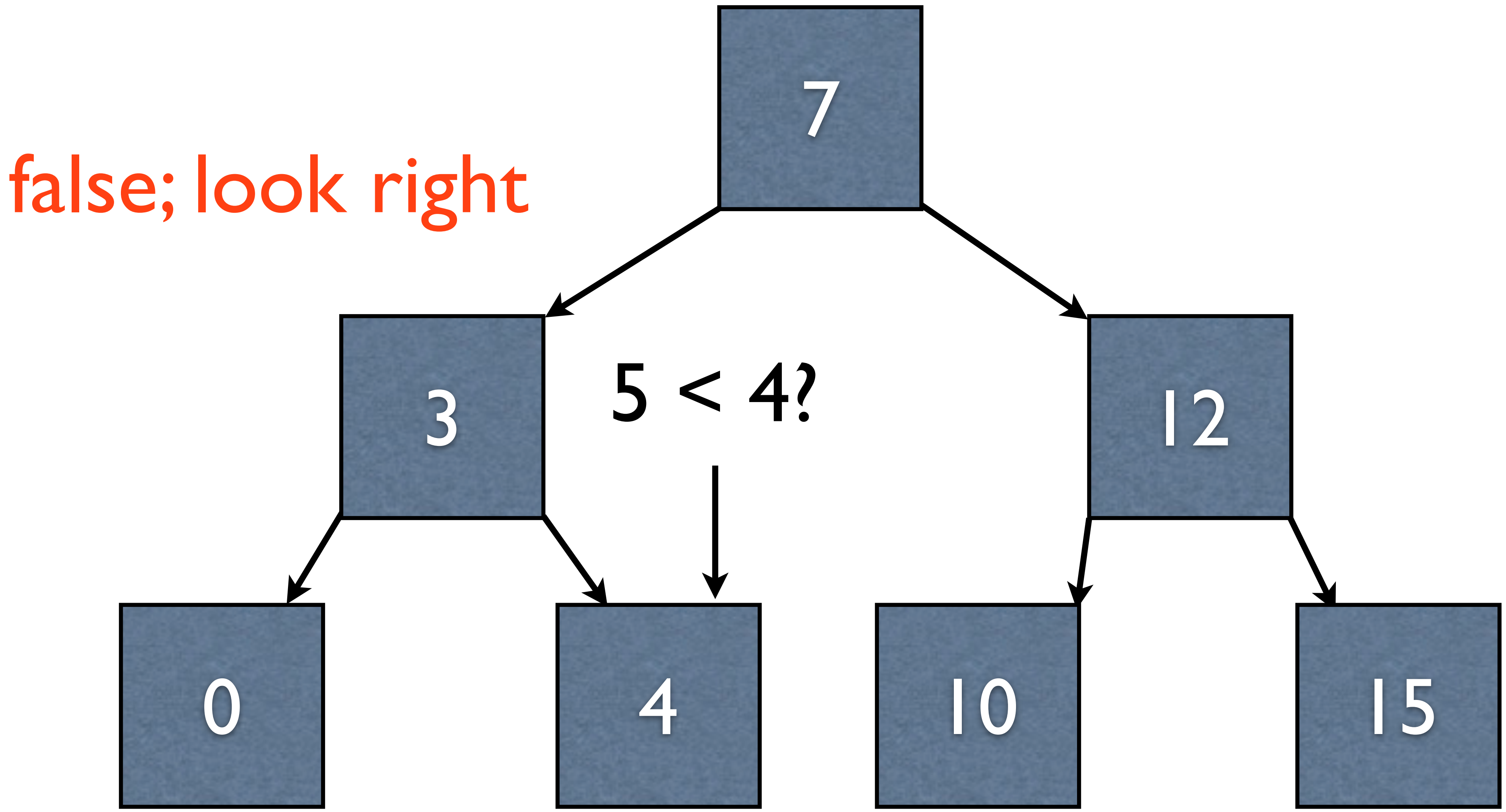
Insertion

Inserting: 5



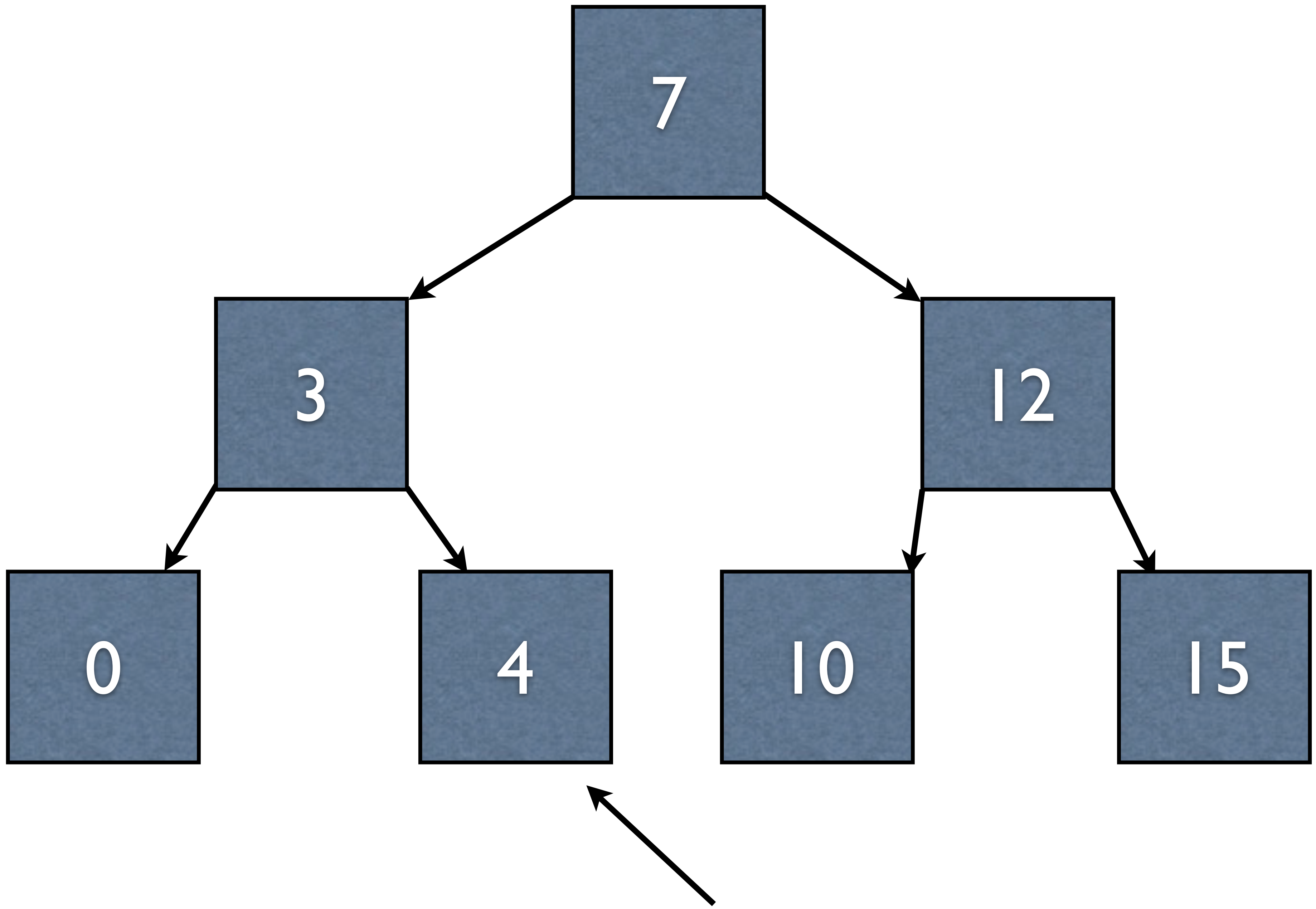
Insertion

Inserting: 5



Insertion

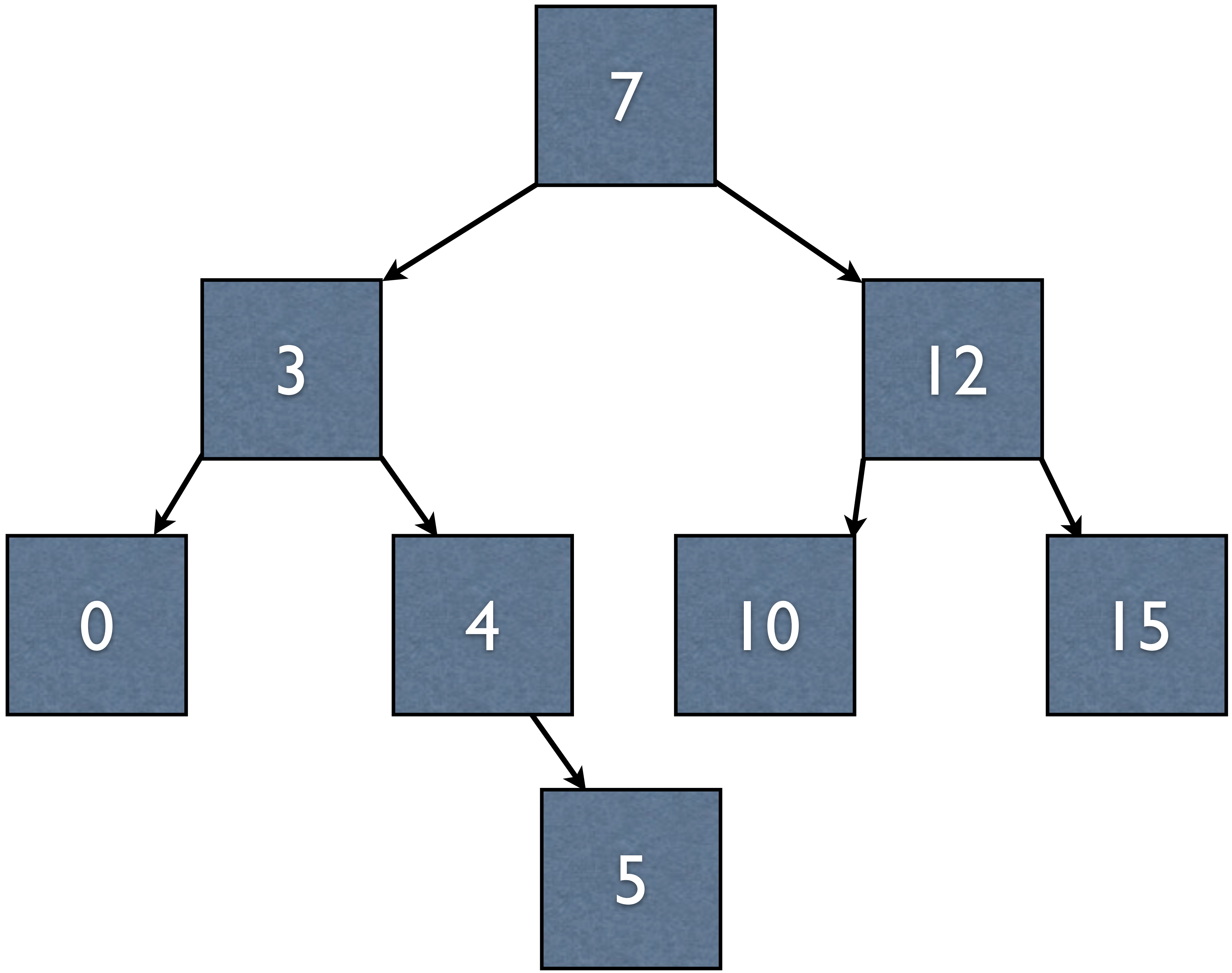
Inserting: 5



No node on right - insert here

Insertion

Inserting: 5



Remaining Issues

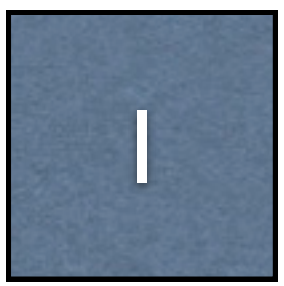
- It turns out that we may not always split data in half with this
- After a long chain of insertions, the tree may become *unbalanced*, meaning we rarely split in half
- Inserting data that's already sorted into an empty tree sees this problem

Already Sorted Data

Data Remaining: `1, 2, 3, 4, 5`

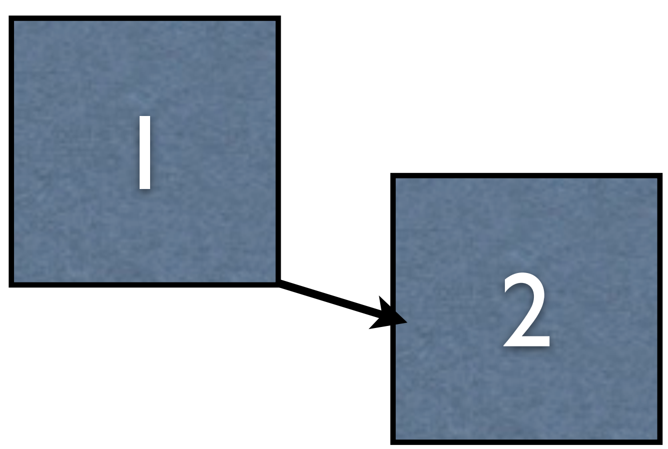
Already Sorted Data

Data Remaining: `2, 3, 4, 5`



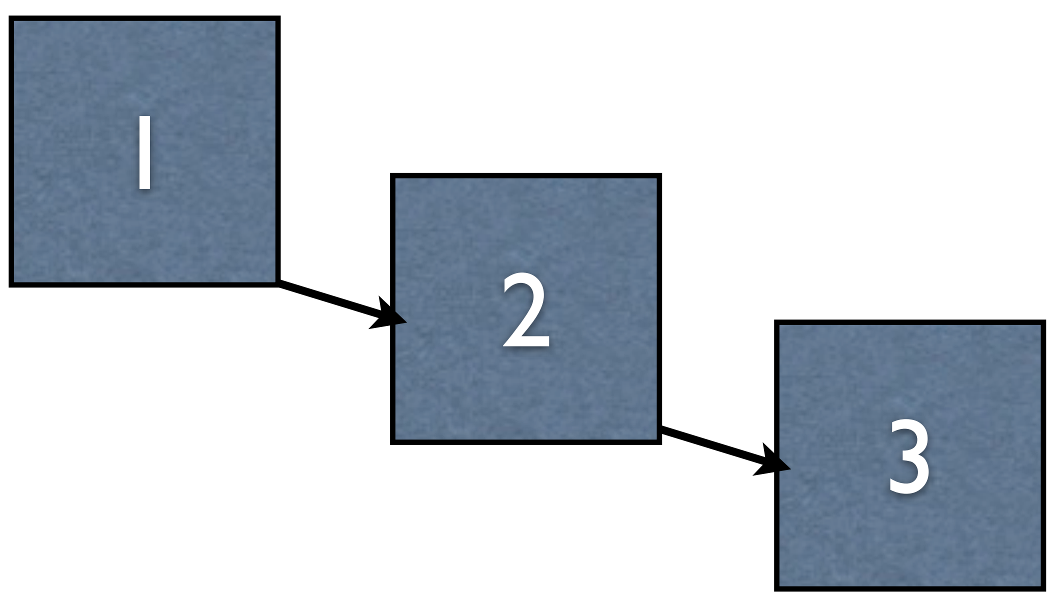
Already Sorted Data

Data Remaining: `3, 4, 5`



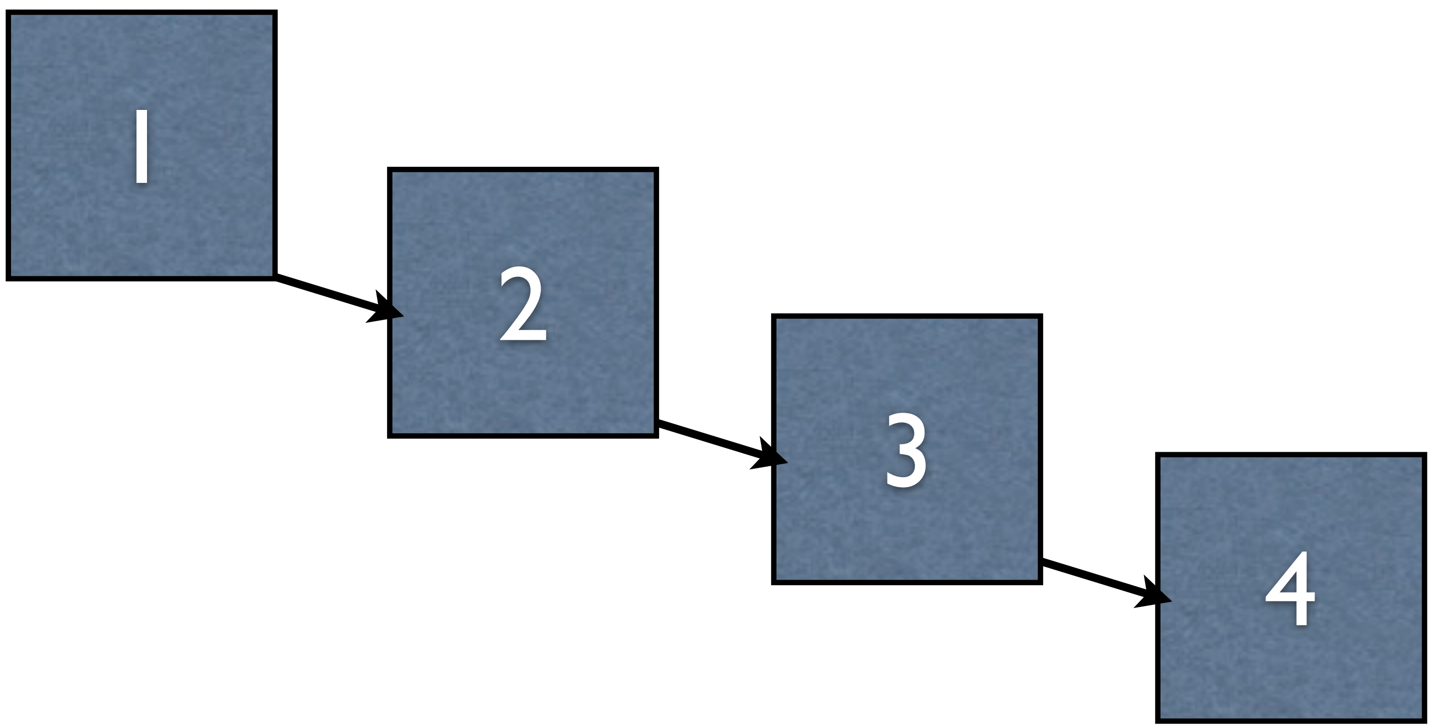
Already Sorted Data

Data Remaining: 4, 5



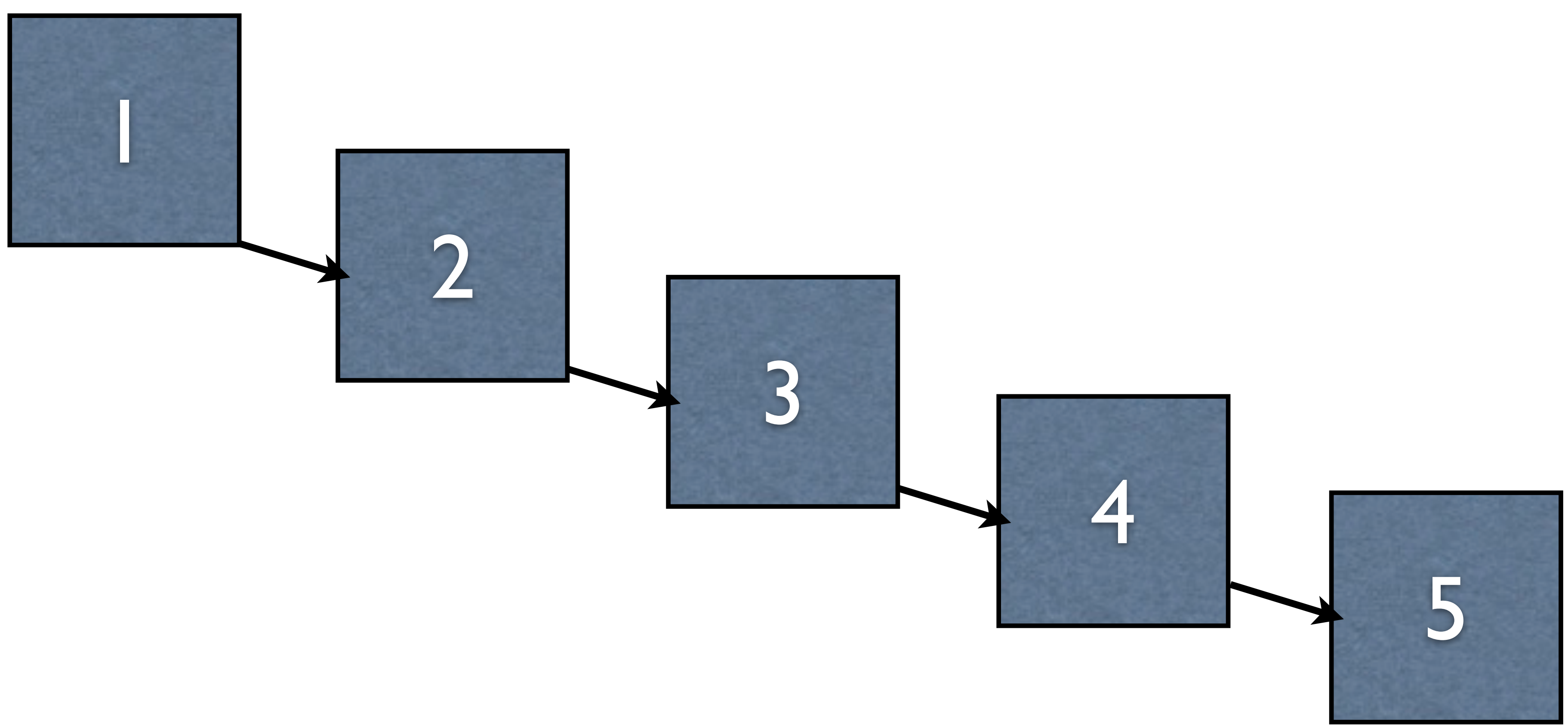
Already Sorted Data

Data Remaining: 5



Already Sorted Data

Data Remaining: None



Big Problem

- Worst case, search and insertion are still `O(N)`, because we do not guarantee the tree will split things up evenly
- There are ways to fix this to guarantee `O(log(N))` time complexity, but they are beyond this class