Red-Black Trees and AA Trees

Node Color, Insertions and Rotations

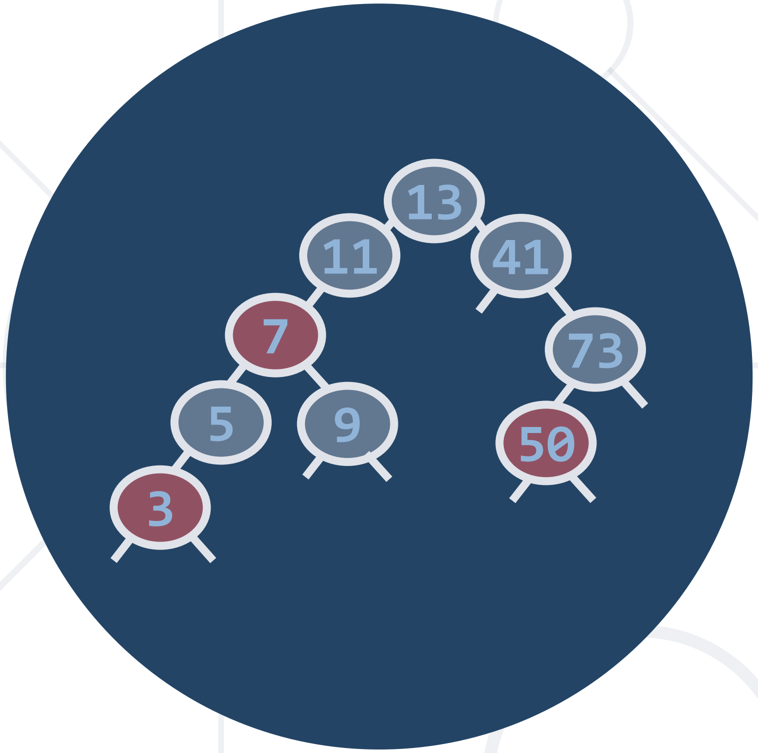
SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents



Red-Black Tree

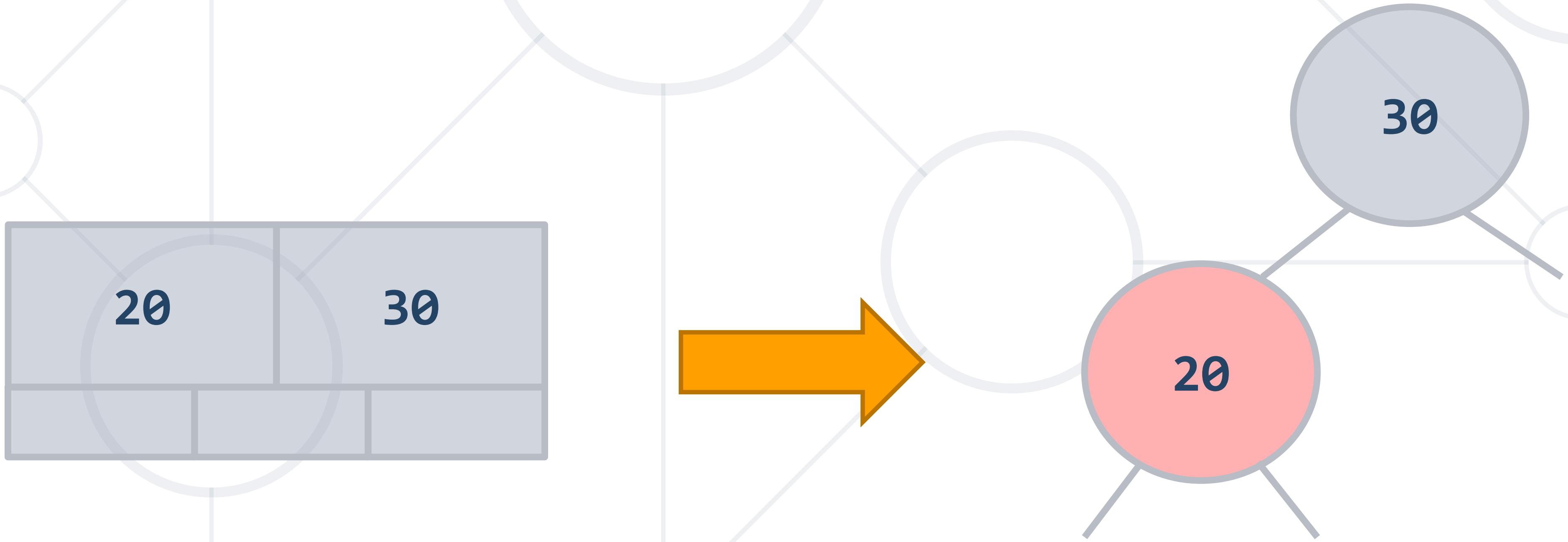
Simple Representation of a 2-3 Tree

Why Yet Another Balanced BST?

- We want operations to happen at:
  - O(log(n)) **not** O(h) where **h** in worst case is **n**
- **AVL** vs **Red-Black** trees:
  - The AVL trees are more balanced that causes more rotations during **insertion** and **deletion**
  - if your application involves many frequent insertions and deletions, then **Red Black** trees should be preferred

Representing 3-Nodes from 2-3 Tree

- We will represent 3-nodes with a **left-leaning** red nodes
- Nodes with values between the 2 nodes will be to the **right** of the **red** node



Red-Black Tree Properties

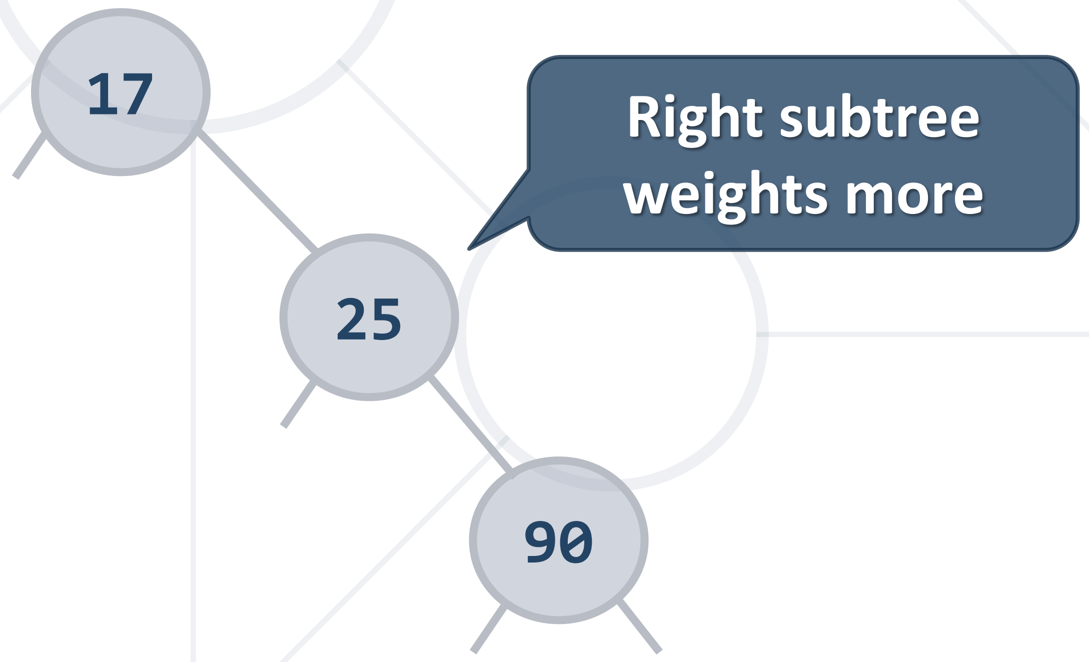
- All **leaves** are black
- The **root** is black
- No node has **two red links** connected to it
- Every path from a given **node** to its **descendant leaf** nodes contains the **same** number of **black** nodes
- Red links **lean** left

Rebalancing Trees

Rotations

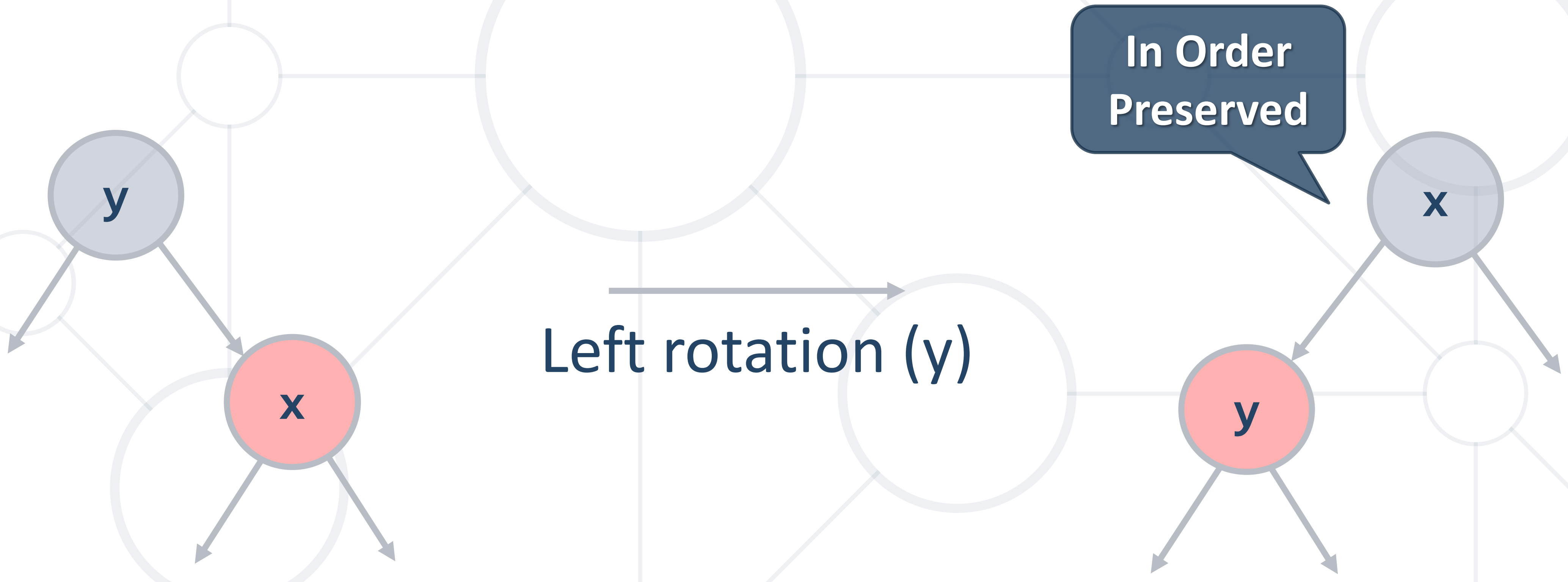
Rotations

- Rotations are used to correct the balance of a tree
- Balance can be measured in height, depth, size etc. of subtrees



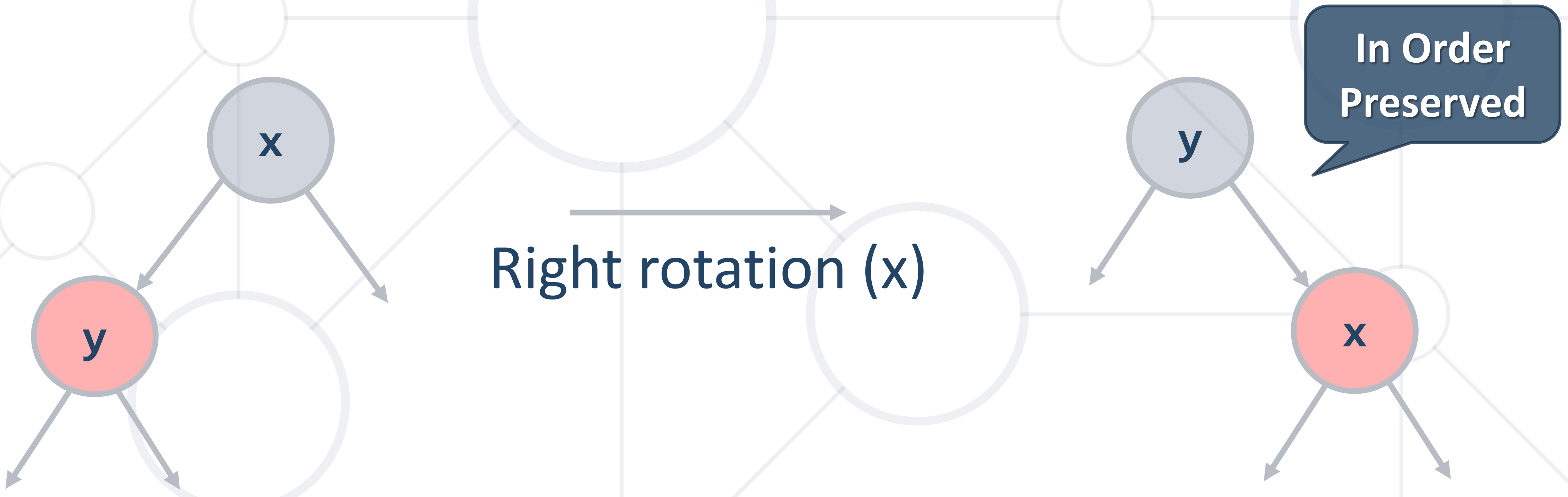
Left Rotation

- Orient a right-leaning red link to lean left



Right Rotation

- Orient a left-leaning red link to lean right (temporarily)



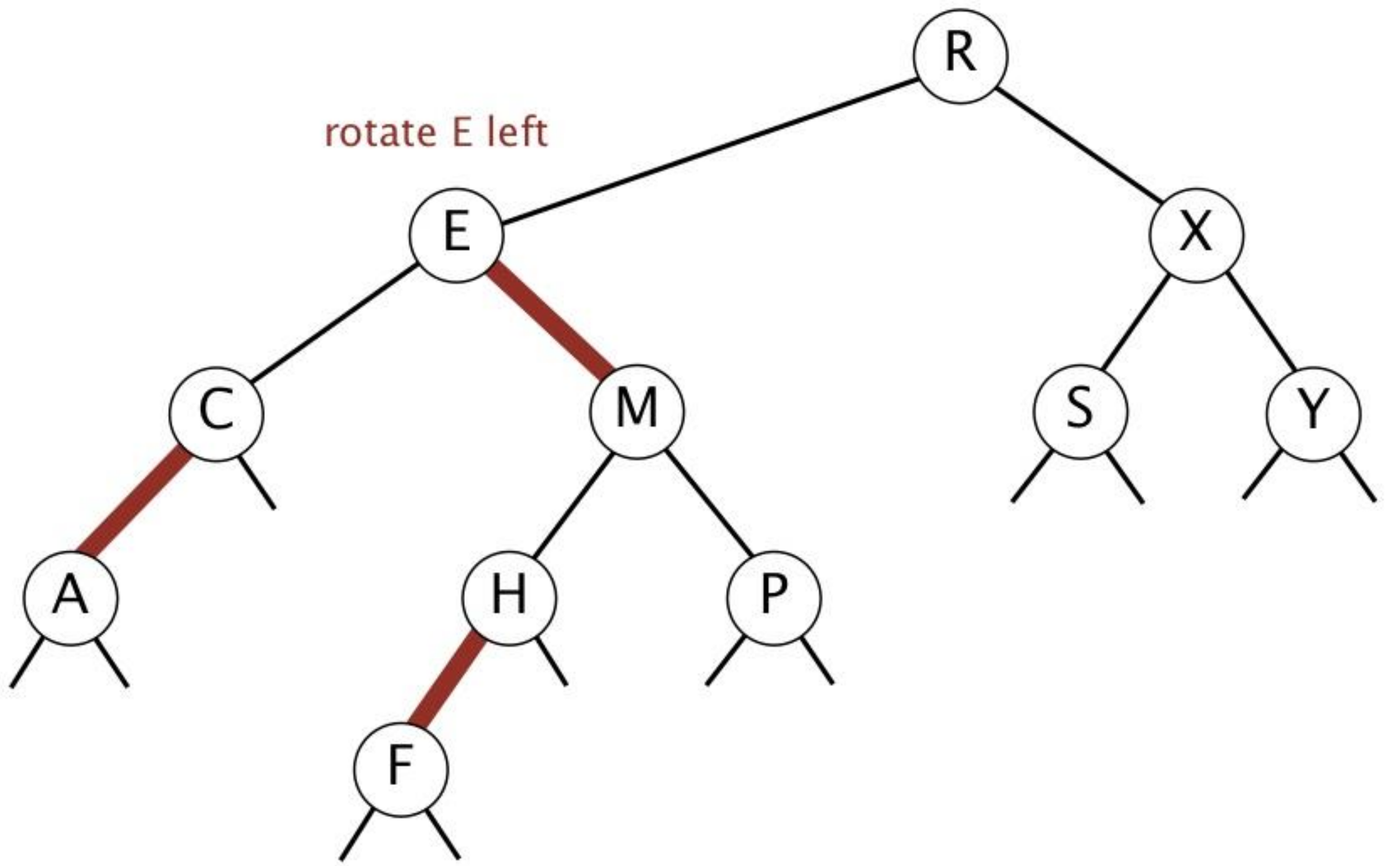
Rotations - Quiz

A. R E X C M S Y A H P F

B. R M X E H S Y C F P A

C. R M X E P S Y C H A F

D. R C X A E S Y M H P F



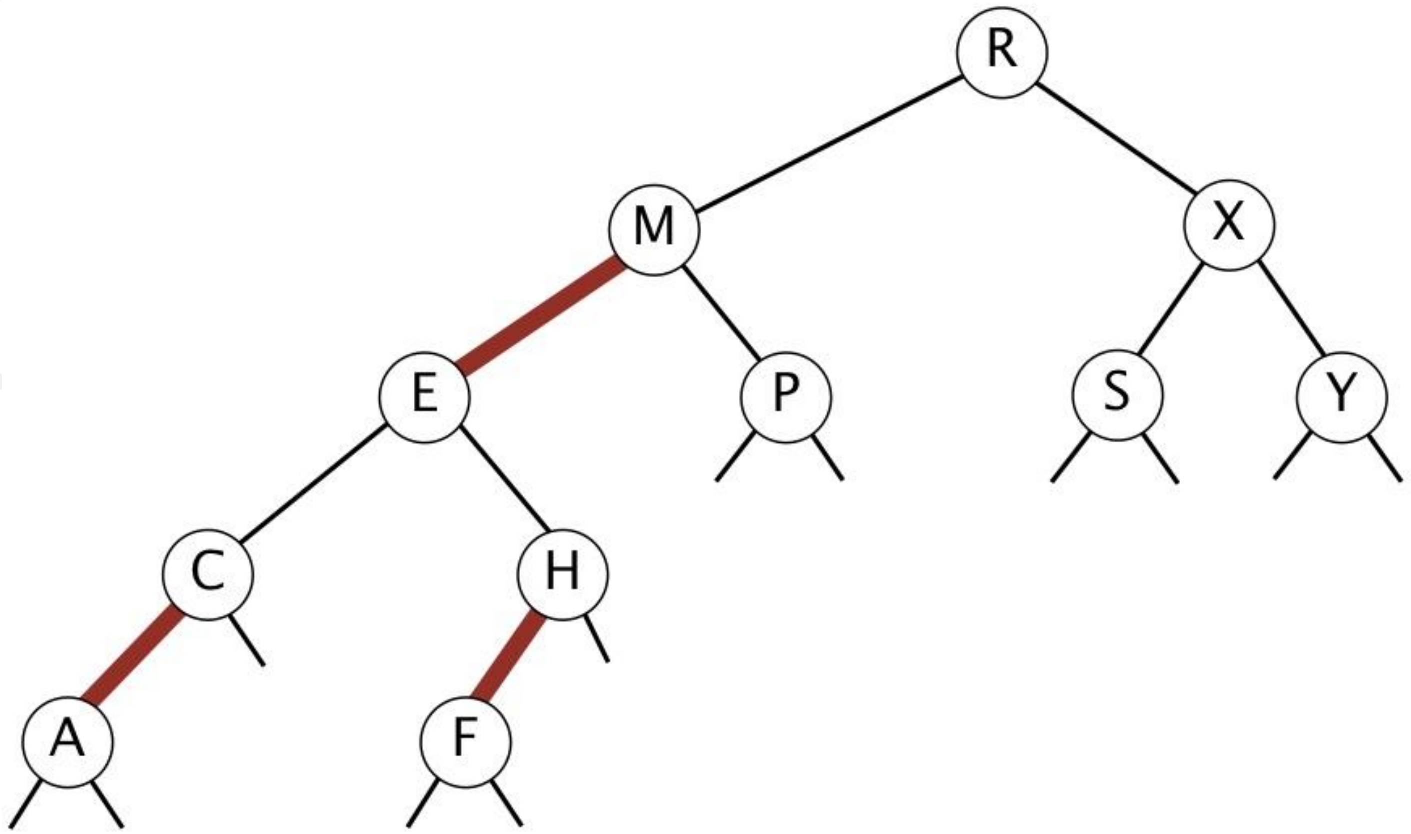
Rotations - Answer

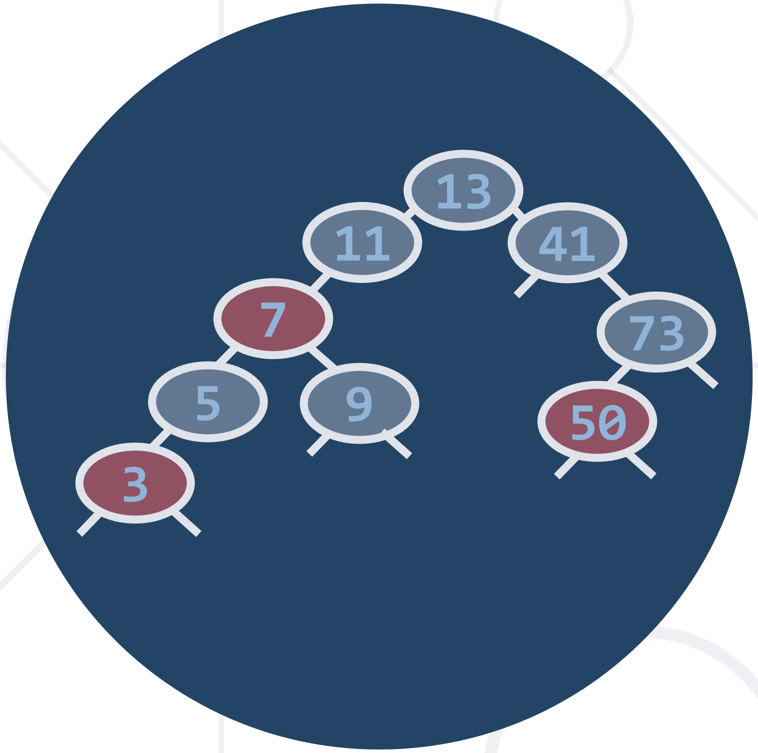
A. R E X C M S Y A H P F

B. R M X E H S Y C F P A

C. R M X E P S Y C H A F

D. R C X A E S Y M H P F





Red-Black Tree

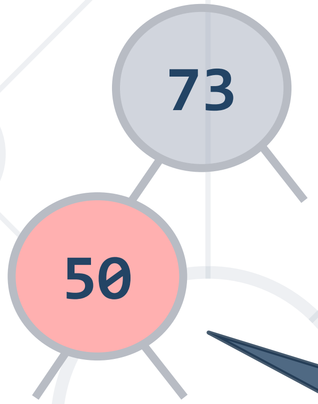
Insertion Algorithm

Insertion Algorithm

- **Locate** the node position
- Create new **red** node
- **Add** the new node to the tree
- **Balance** the tree if needed

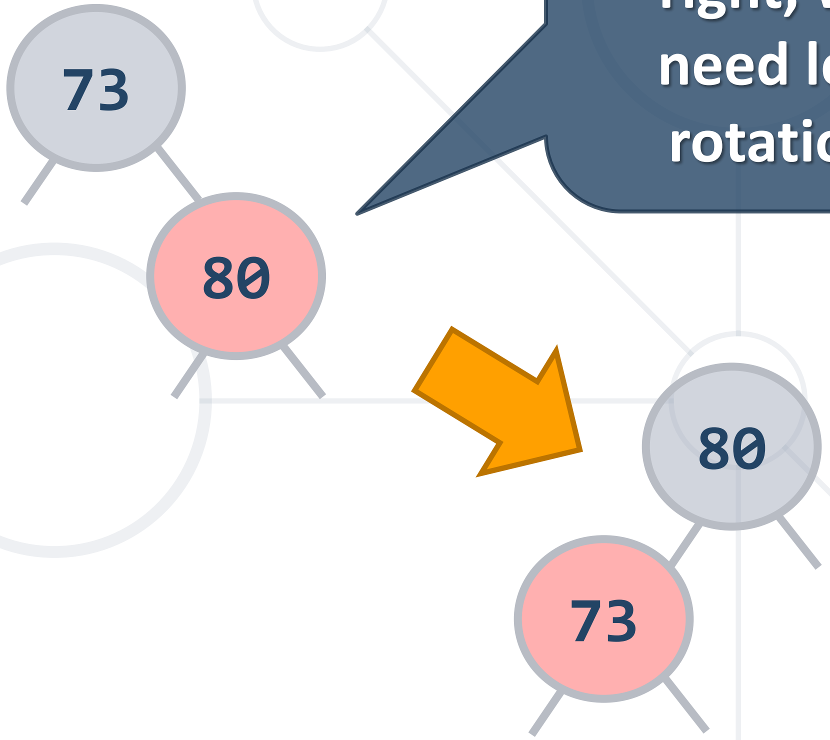
Insertion

- Insert into a single 2-node:
- Smaller element



The red node is leaning left

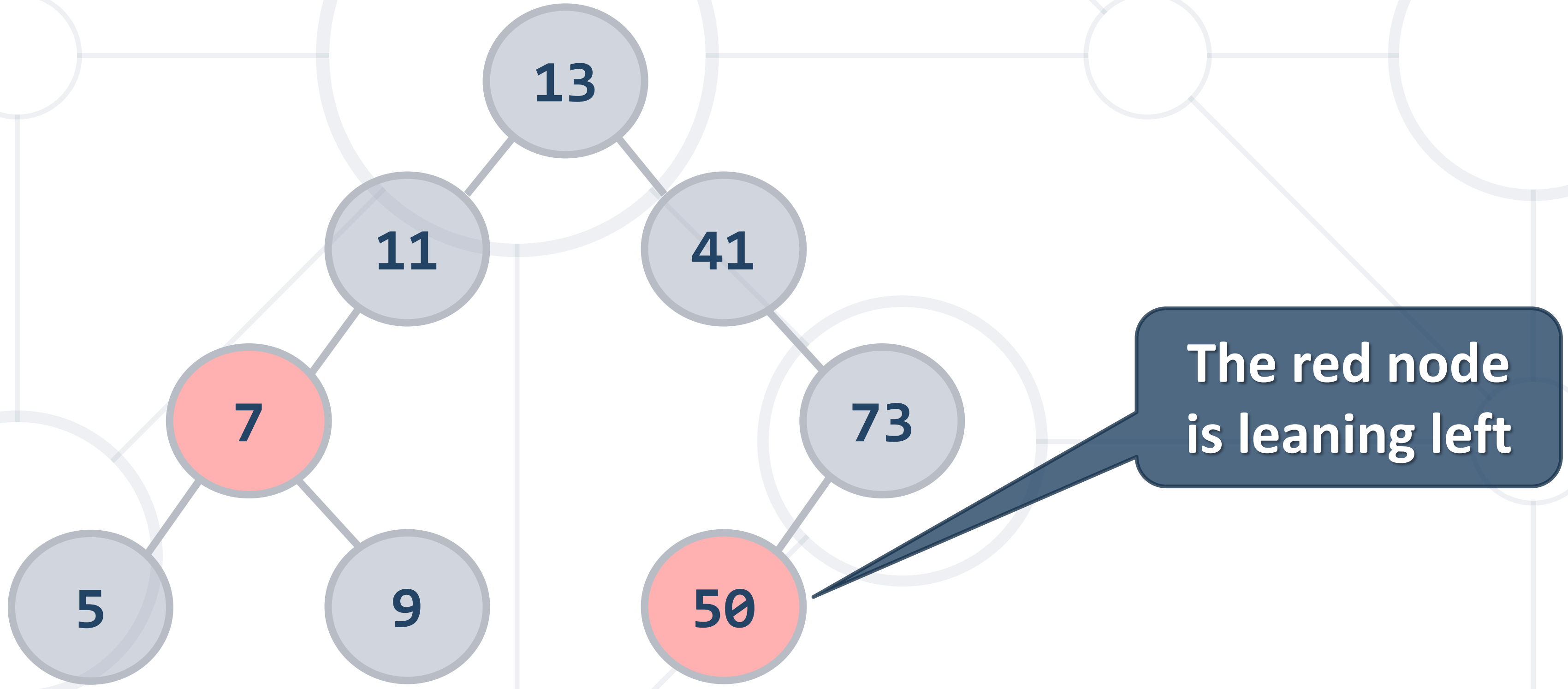
- Larger element



The red node is leaning right, we need left rotation

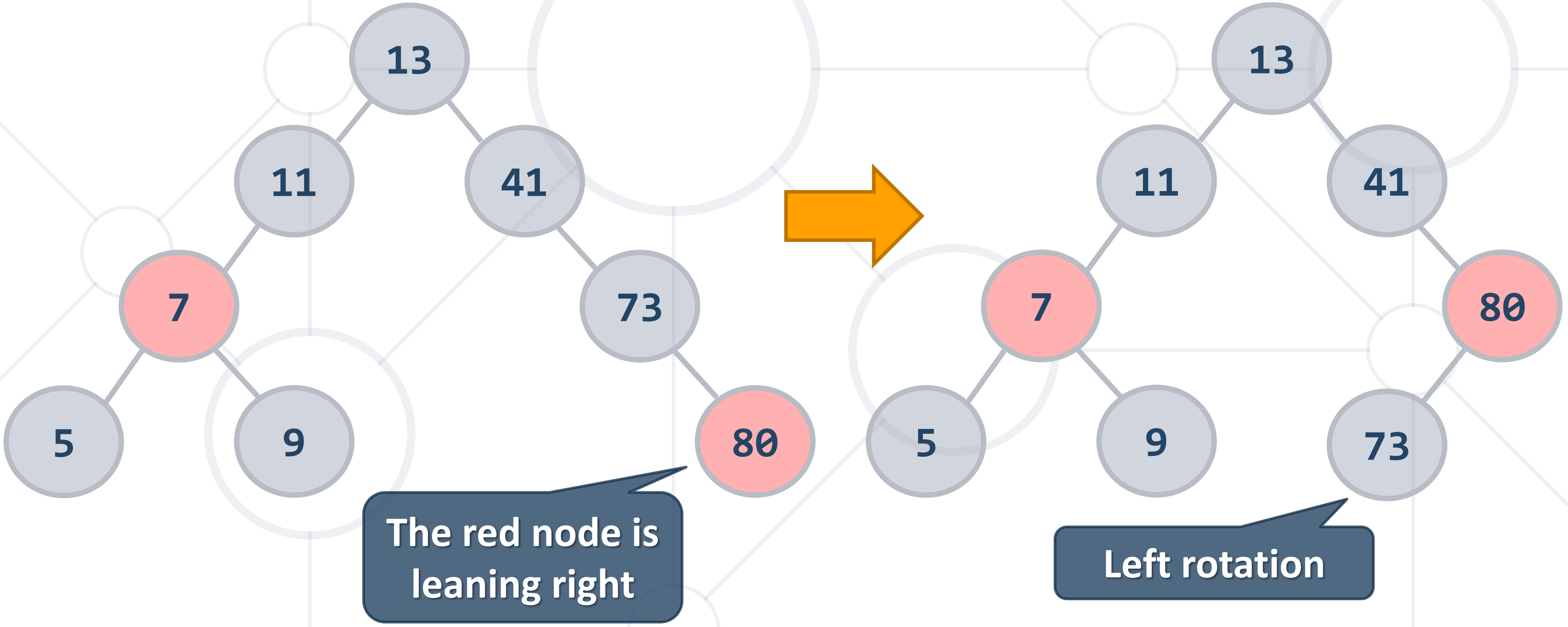
Insertion (2)

- Insert **smaller** item into a 2-node at the bottom:



Insertion (3)

- Insert **larger** item into a 2-node at the bottom:

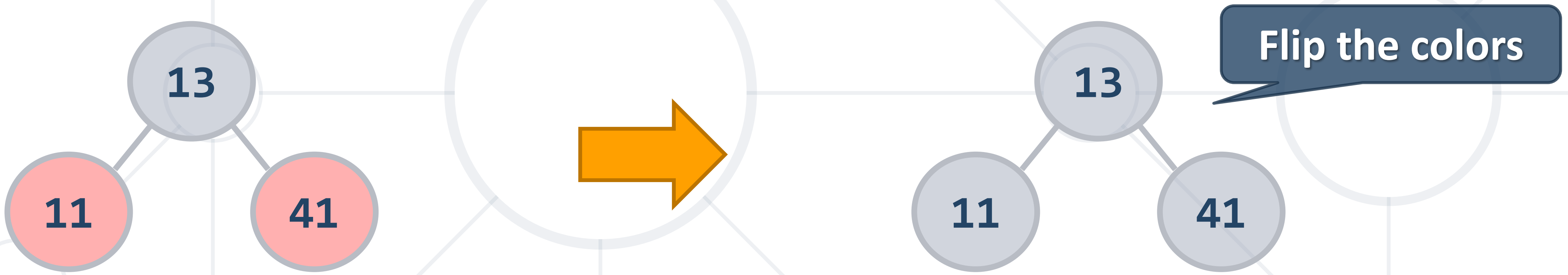


Insertion Into 3-Node

- 3 cases:
  - The element is **smaller** than both keys
  - The element is **larger** than both keys
  - The element is **between** the 2 keys

Insertion Into 3-Node (2)

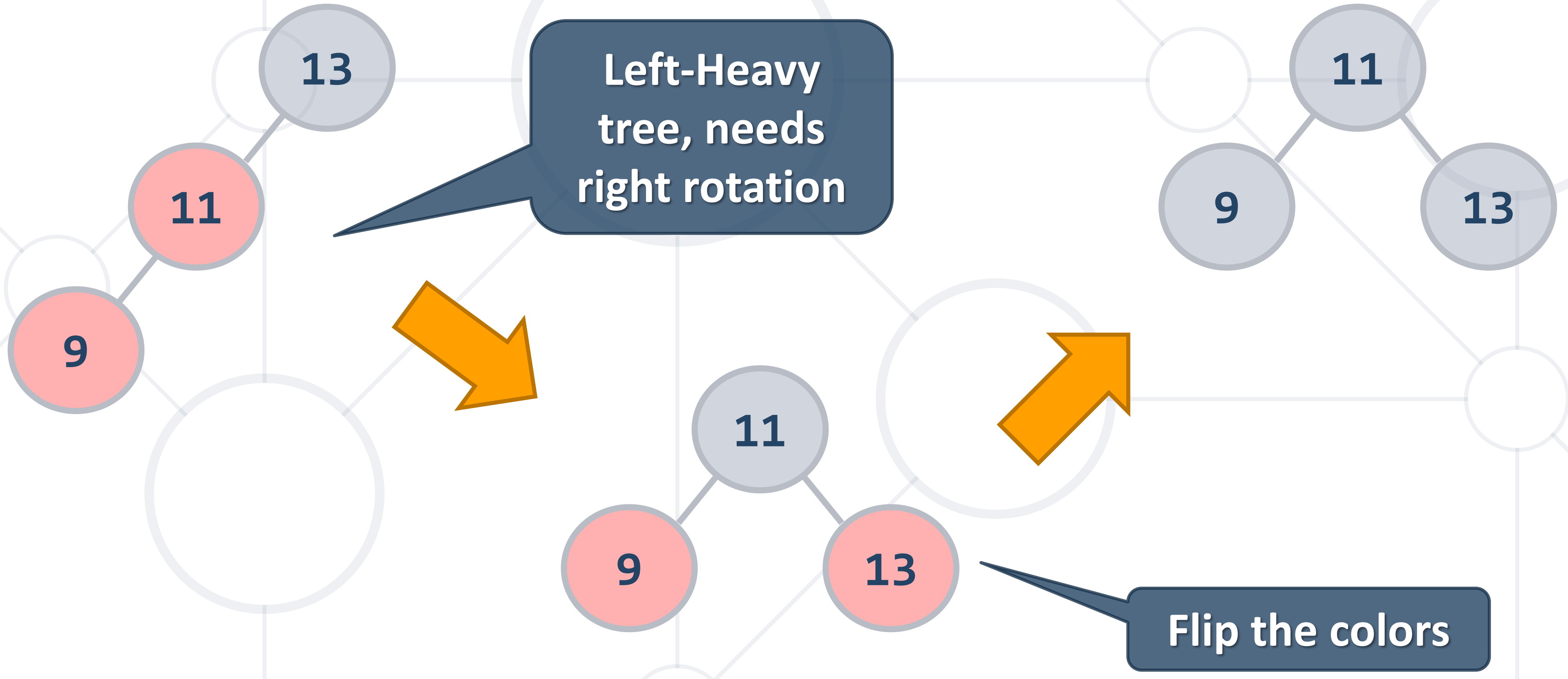
- Larger than both keys:



- Flipping the colors increases the tree height, which maintains the 1-1 correspondence to 2-3 trees

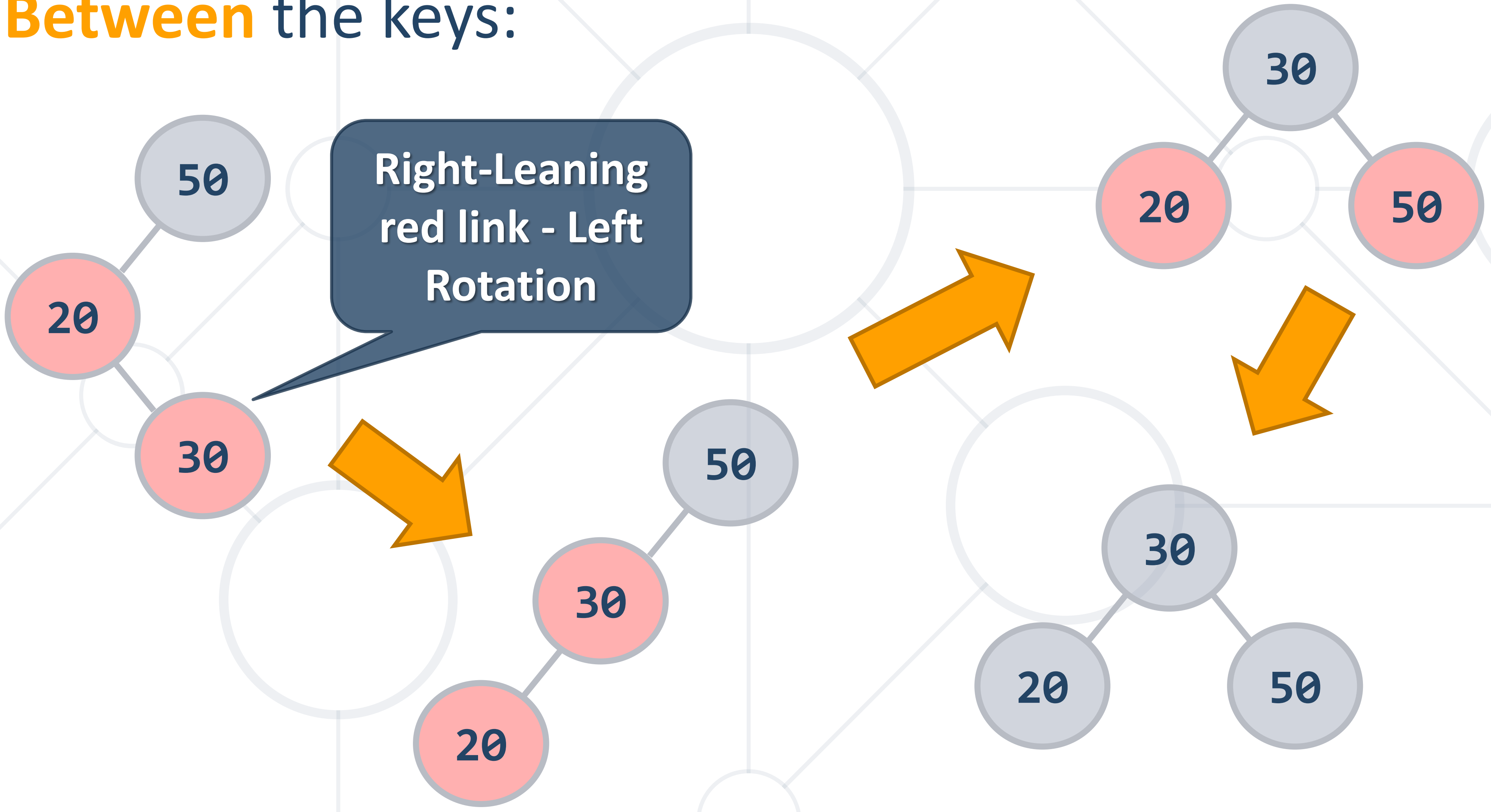
Insertion Into 3-Node (3)

- Smaller than both keys:



Insertion Into 3-Node (4)

- **Between** the keys:



Flipping Colors

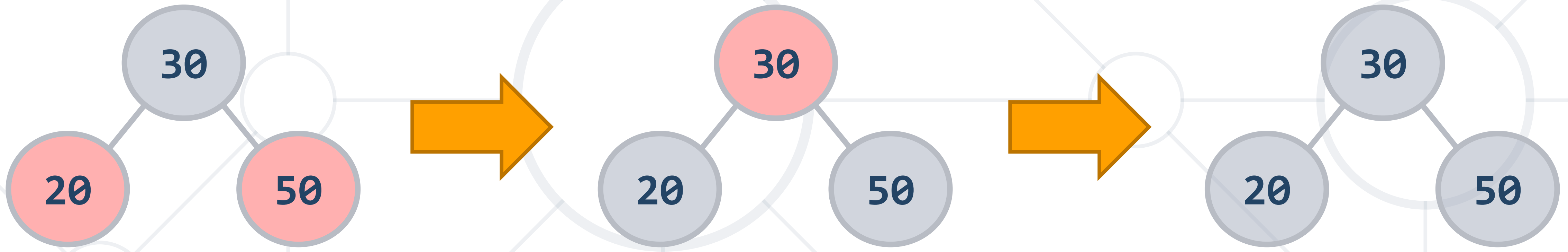
- Flipping the colors should also change the **parent color** to **red**

```
void flipColors(Node<T> node) {
  node.color = RED;
  node.left.color = BLACK;
  node.right.color = BLACK;
}
```

- Preserves perfect **black balance** in the tree!

Keeping Black Root

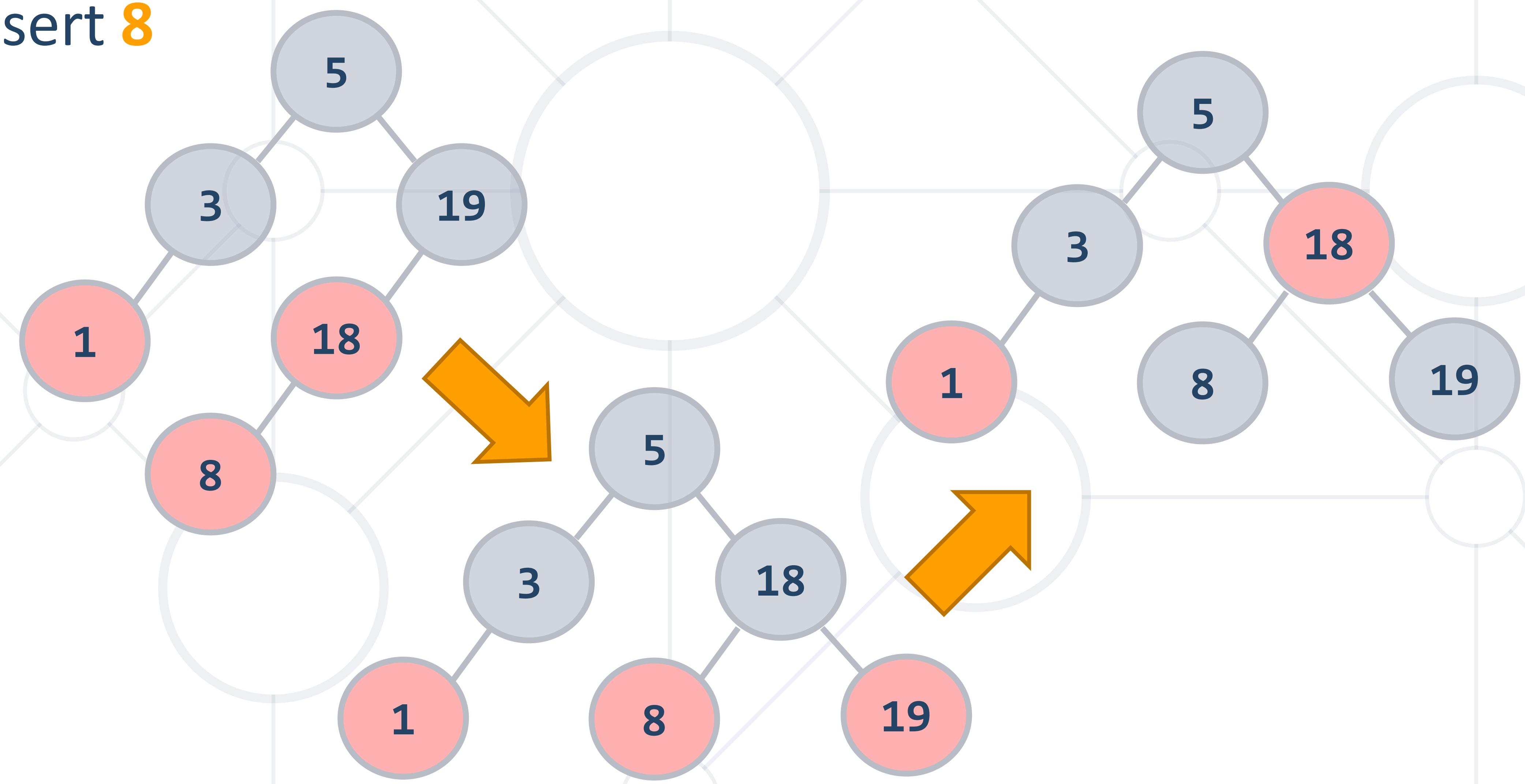
- Insert on a single node (root):



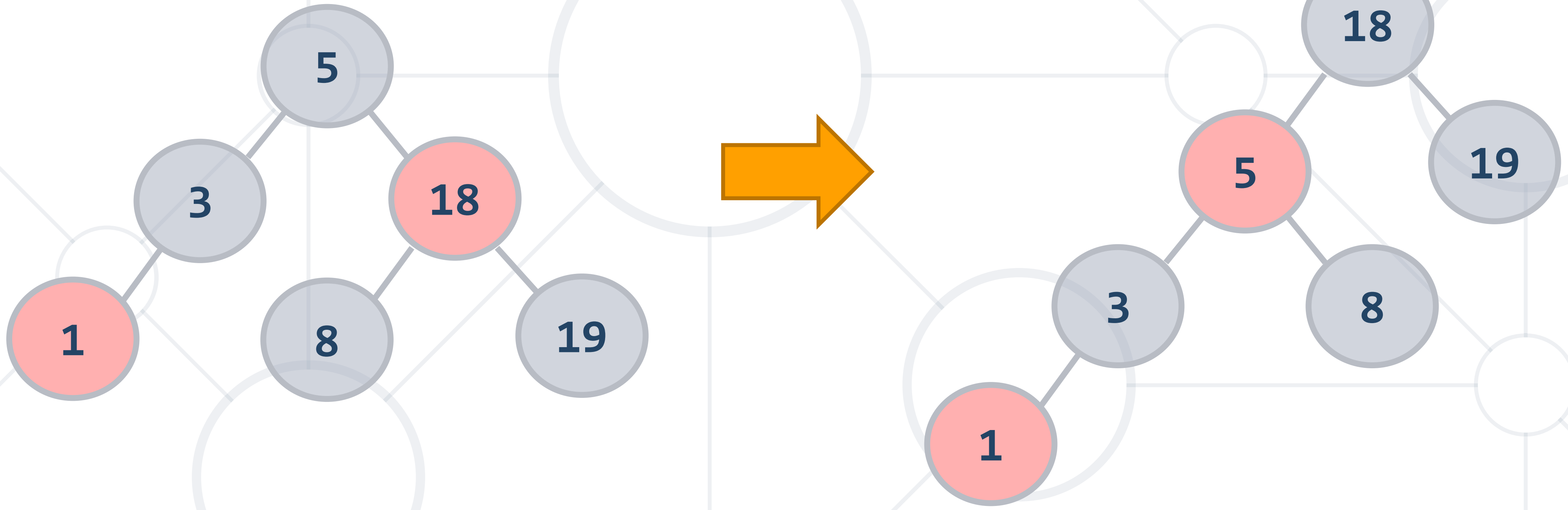
- Each time the root switches colors, the height of the tree is increased

Insert Into 3-Node at the Bottom

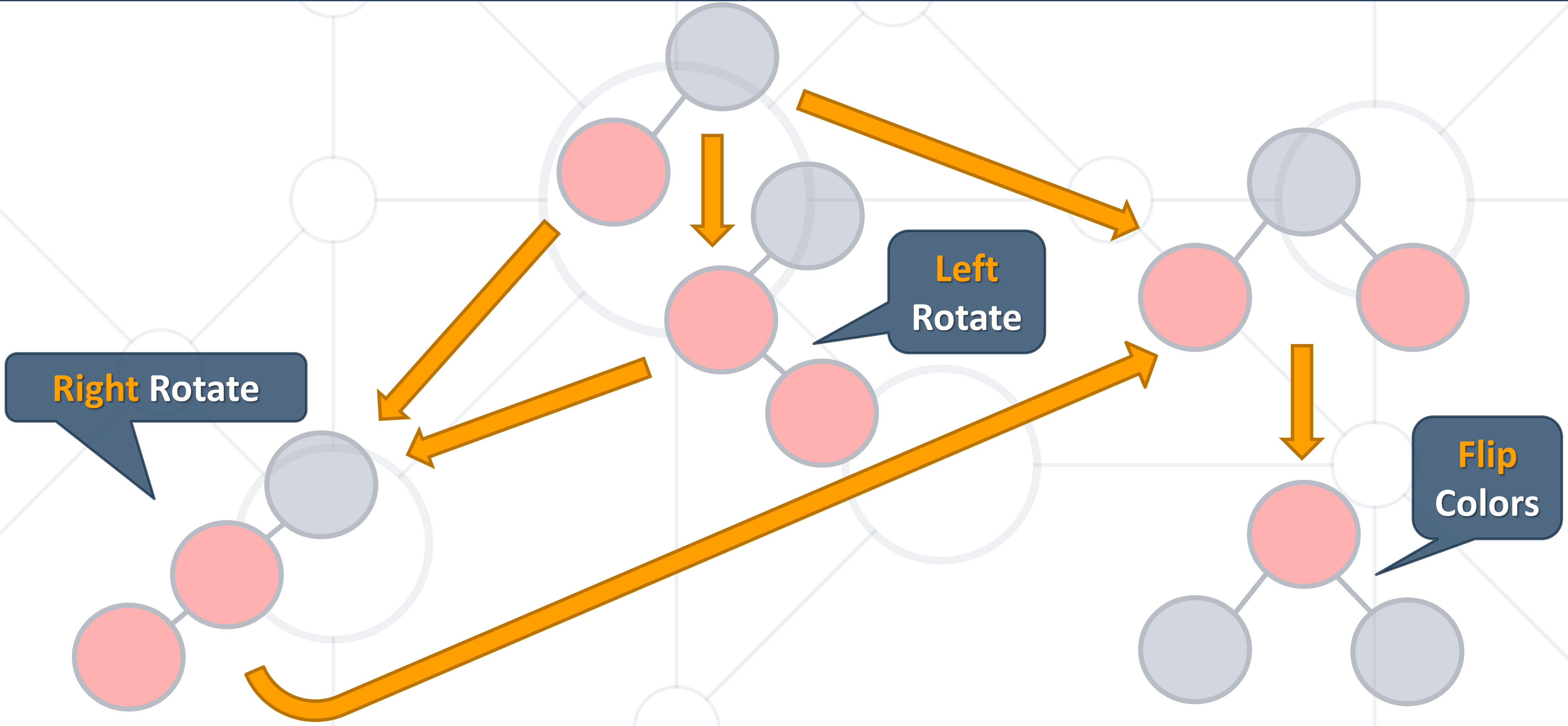
- Insert 8

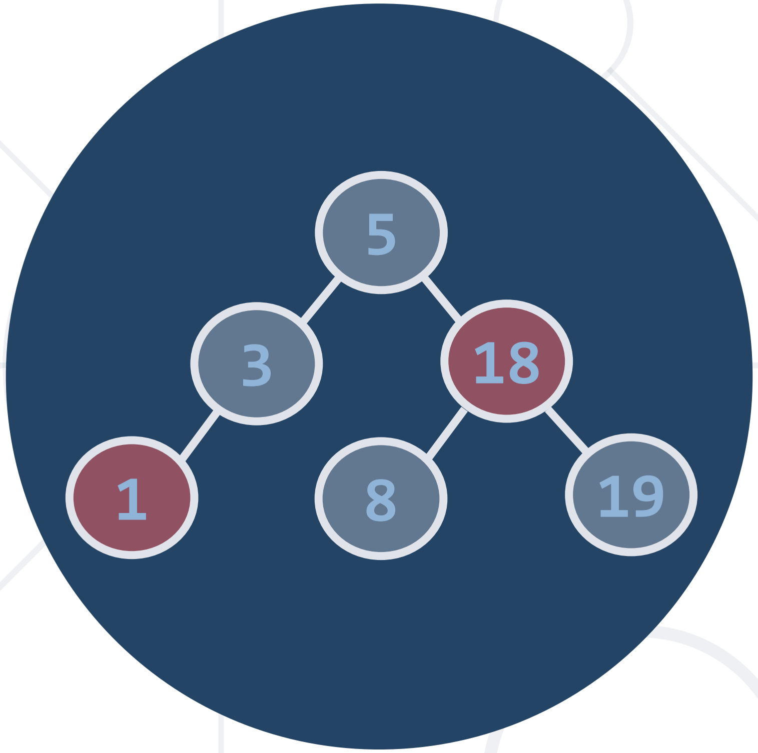


Insert Into 3-Node at the Bottom (2)



Overall Insertion Process





Red-Black Tree

Insertion Implementation

Changes to the BST Class

```
class RedBlackTree<T> {
  private static final boolean RED = true;
  private static final boolean BLACK = false;

  private static class Node<T> {
    public boolean color;
    public Node(T value, bool color) {
      // TODO: Add setup logic here
    }
}
```

Changes to the BST Class (2)

```
class RedBlackTree<T> {

  private boolean isRed(Node<T> node)

  private Node<T> rotateLeft(Node<T> node)

  private Node<T> rotateRight(Node<T> node)

  private void flipColors(Node<T> node)
}
```

Rotate Right

```java
private Node<T> rotateRight(Node<T> node) {
  Node<T> temp = node.left;
  node.left = temp.right;
  temp.right = node;
  temp.color = node.color;
  node.color = RED;
  node.count = 1 + count(node.left) + count(node.right);

  return temp;
}
```

Rotate Left

```
private Node<T> rotateLeft(Node<T> node) {
  Node<T> temp = node.right;
  node.right = temp.left;
  temp.left = node;

  // Same operations as rotateRight()

  return temp;
}
```

Insert

```
private boolean isRed(Node<T> node) {
  if (node == null) return false;
  return node.color;
}
```

```
public void insert(T element) {
  this.root = this.insert(element, this.root);
  this.root.color = BLACK;
}
```

Insert(2)

```
private Node<T> insert(T element, Node<T> node) {
  if (node == null) node = new Node<>(element, RED);

  // Recursive calls to go left or right

  if (this.isRed(node.right) && !this.isRed(node.left))
    node = this.rotateLeft(node);
  if (this.isRed(node.left) && this.isRed(node.left.left))
    node = this.rotateRight(node);
  if (this.isRed(node.left) && this.isRed(node.right))
    this.flipColors(node);

  // Increase count
}
```

Red-Black Tree - Quiz

TIME'S

- Suppose that you insert **n** keys in ascending order into a red-black BST. What is the height of the resulting tree?
  - Constant
  - Logarithmic
  - Linear
  - Linearithmic

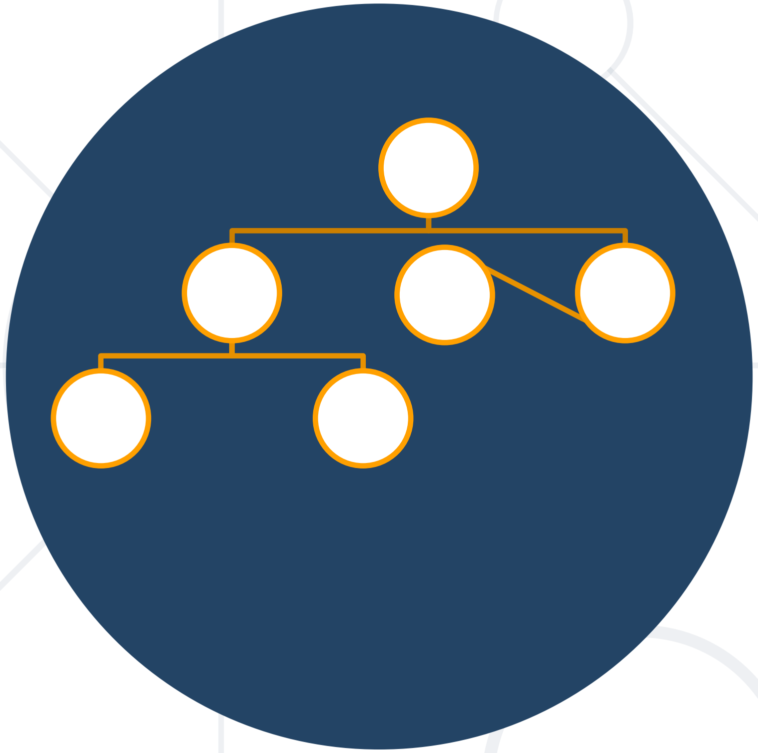
Red-Black Tree - Answer

- Suppose that you insert **n** keys in ascending order into a red-black BST. What is the height of the resulting tree?
  - Constant
  - Logarithmic ✓
  - Linear
  - Linearithmic

The height of any red–black BST on n keys (regardless of the order of insertion) is guaranteed to be between $\log_2 n$ and $2\log_2 n$

Red-Black Tree - Summary

Structure	Worst case			Average case	
	Search	Insert	Delete	Search Hit	Insert
BST	N	N	N	1.39 lg N	1.39 lg N
2-3 Tree	c lg N	c lg N	c lg N	c lg N	c lg N
Red-Black	2 lg N	2 lg N	2 lg N	lg N	lg N



AA Tree

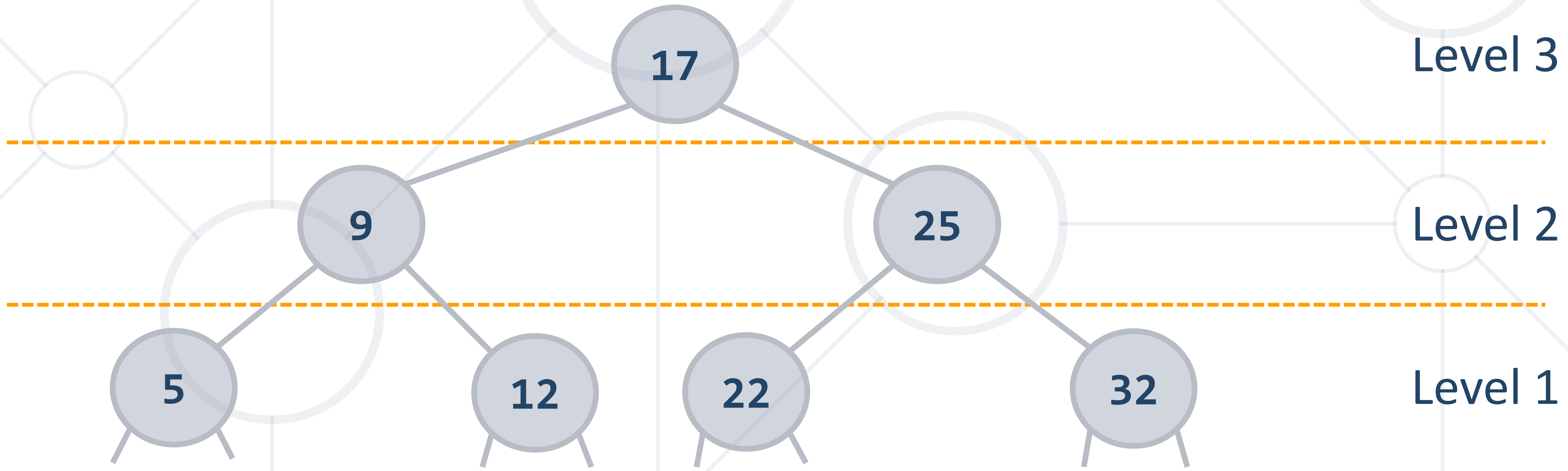
Definition

Why AA Trees

- **Red-Black** vs **AA** trees:
  - The implementation and number of rotation cases in Red-Black Trees is **complex**. AA trees **simplifies** the algorithm
  - It eliminates **half** of the restructuring process by eliminating half of the **rotation** cases, which is easier to code
  - It **simplifies** the deletion process by removing multiple cases

AA Tree

- Utilizes the concept of **levels**
- **Level** - the **number of left links** on the path to a **null** node

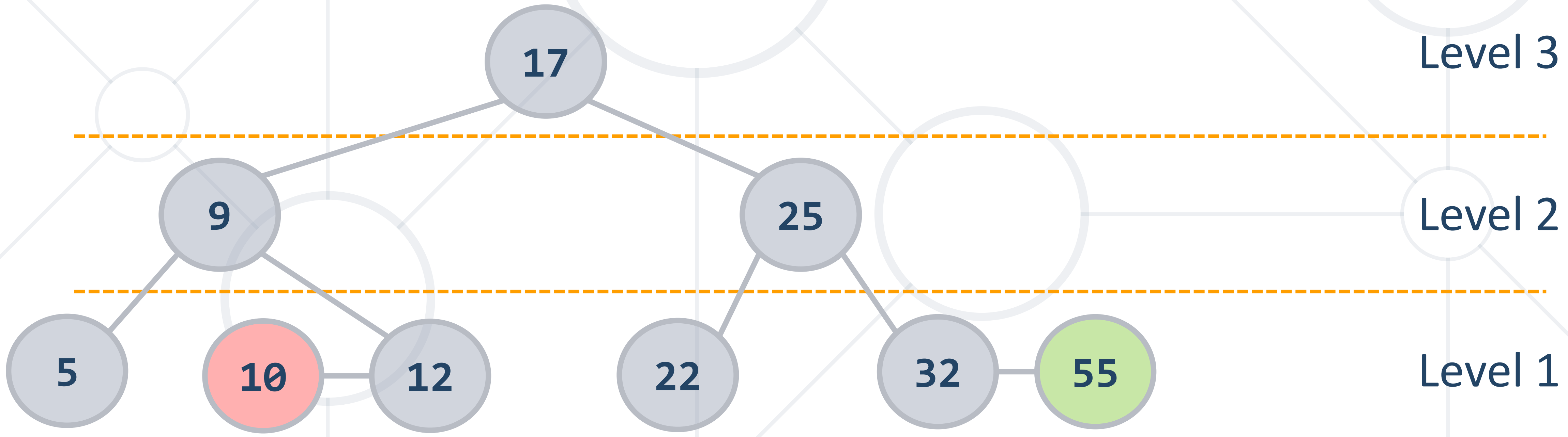


AA Tree

- AA tree invariants
  - The **level** of every **leaf node** is **one**
  - Every **left child** has **level one less** than its **parent**
  - Every **right child** has **level equal** to or **one less** than its **parent**
  - **Right grandchildren** have **levels less** than their **grandparents**
  - Every node of level greater than one **has two children**

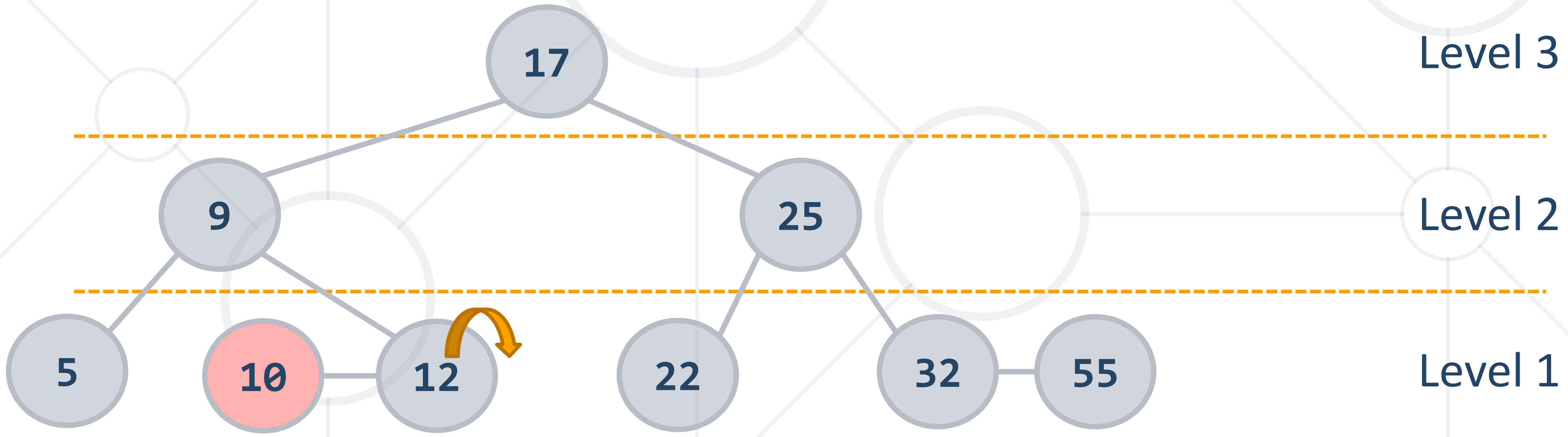
AA Tree

- Right horizontal links are possible
- Left horizontal links are not allowed



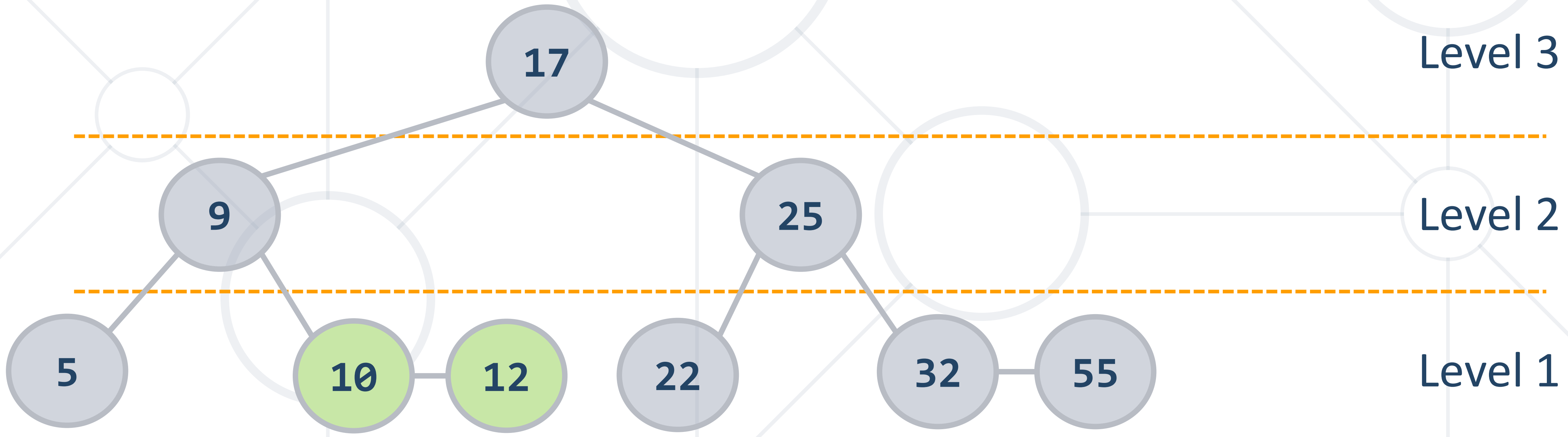
Skew

- Skew operation is a single **right** rotation
- Skew when an **insertion** or **deletion** creates a horizontal **left link**



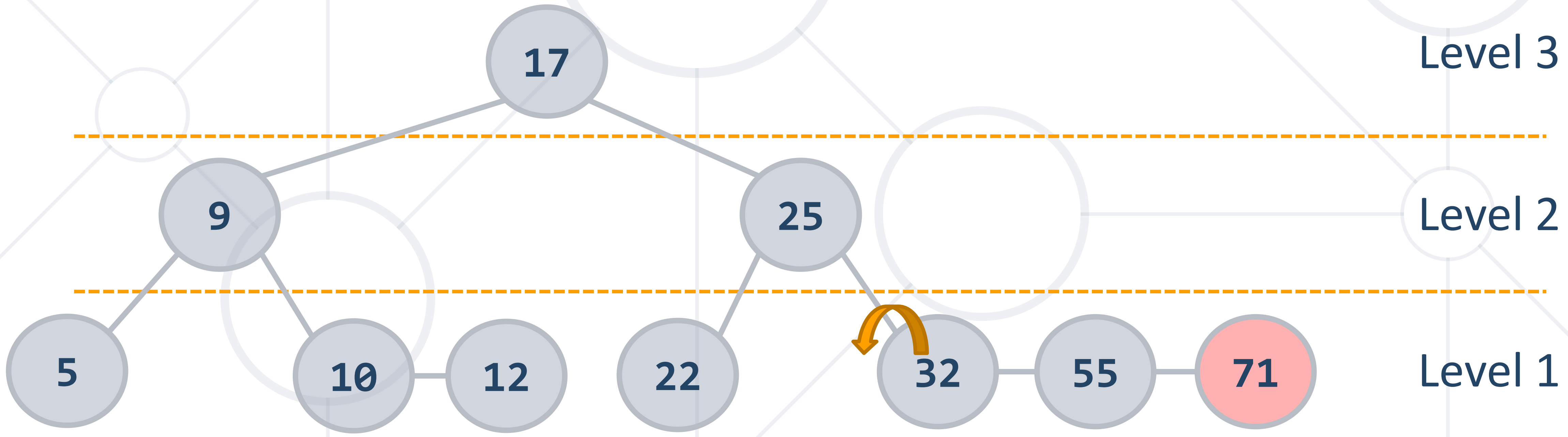
Skew (2)

- Skew operation is a **single right rotation**
- Skew when an insertion or deletion **creates a horizontal left link**



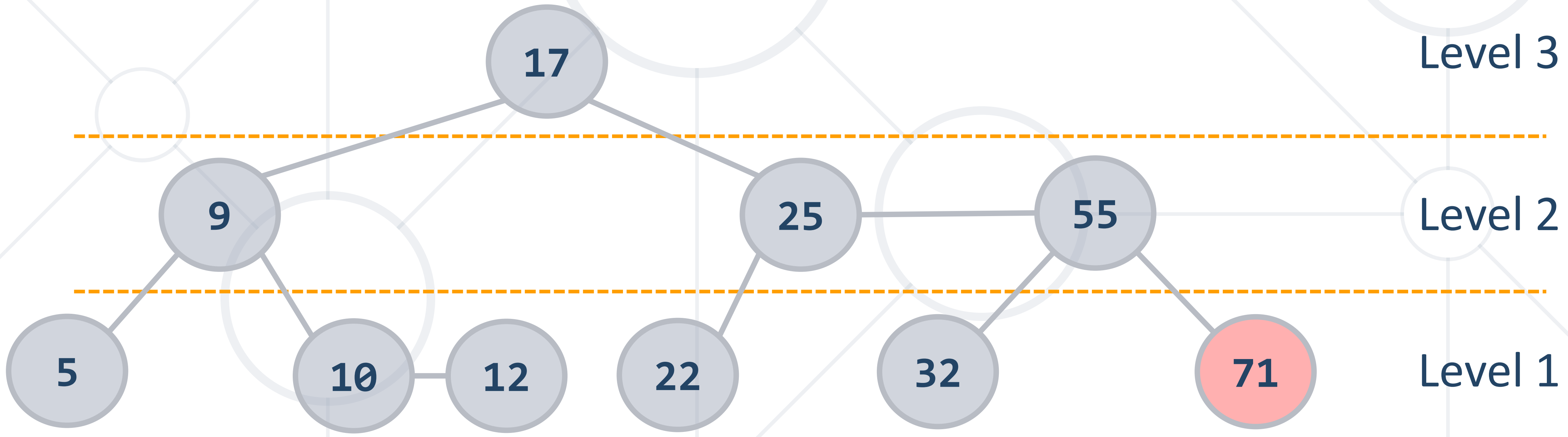
Split

- Split operation is a **single left rotation**
- Split when an insertion or deletion **two consecutive right horizontal links**



Split

- Split operation is a **single left rotation**
- Split when an insertion or deletion **two consecutive right horizontal links**

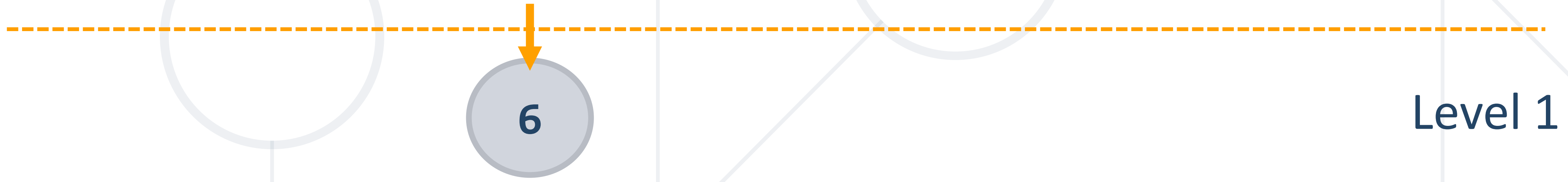


AA Tree

Insertion Algorithm

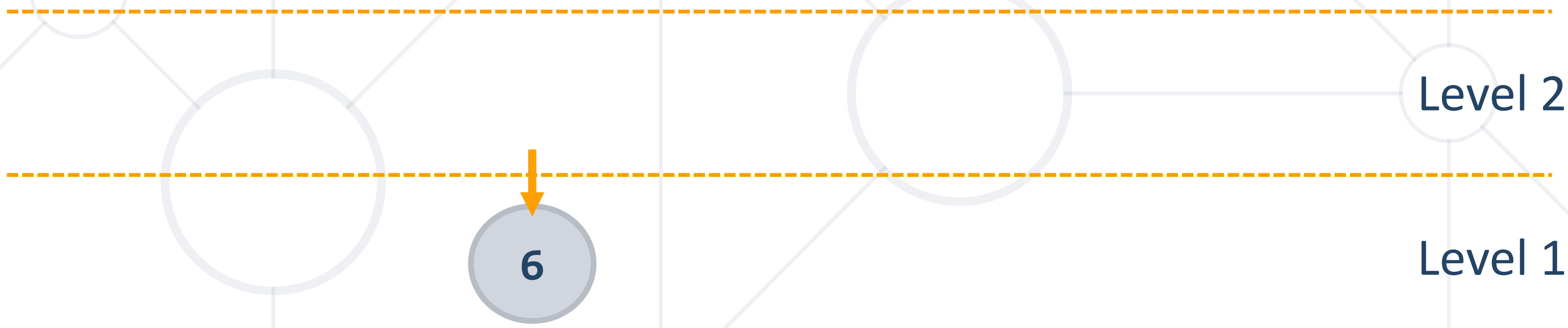
AA Tree Insertion #1

- Insert: **6**
- **New nodes** are always inserted at **Level 1**



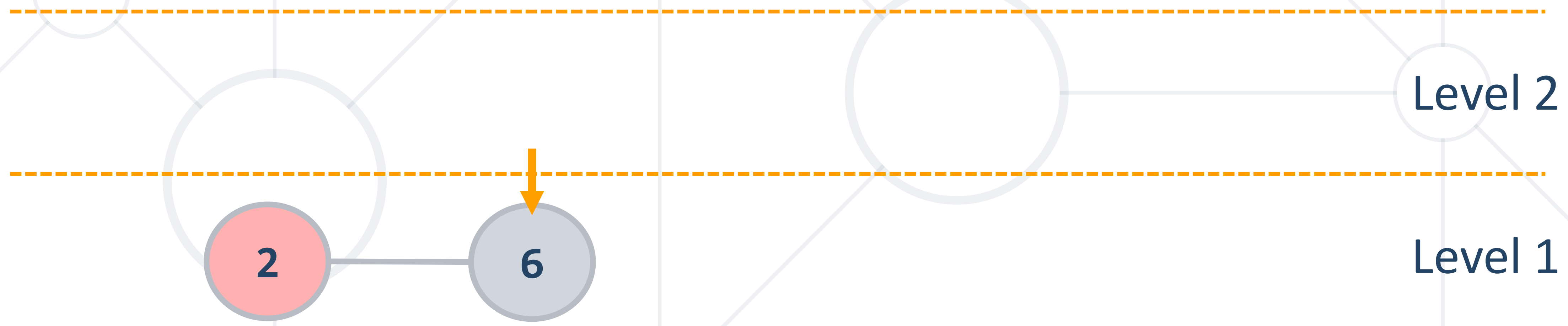
AA Tree Insertion #1

- Insert: **2**



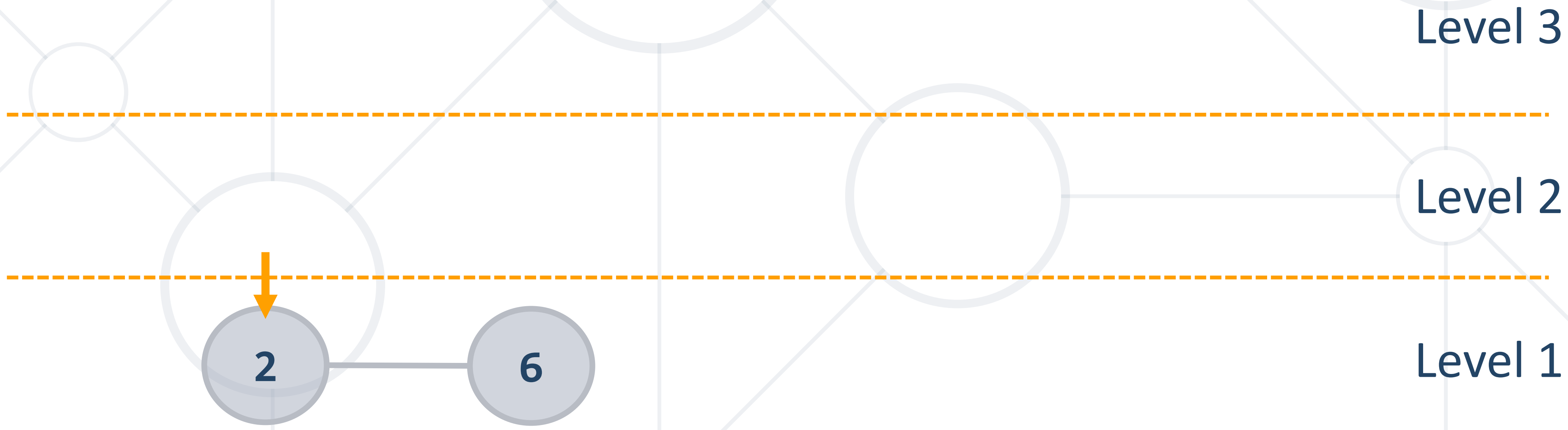
AA Tree Insertion #1

- **Left horizontal** link is **not allowed**
- Rotate 6 right (skew)



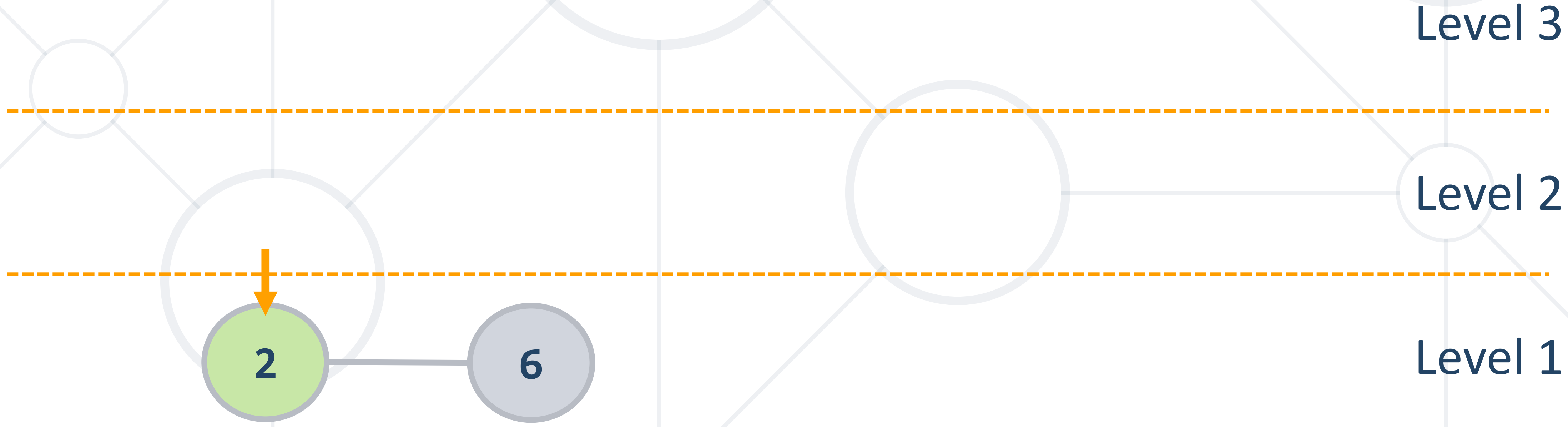
AA Tree Insertion #1

- Insert: **8**



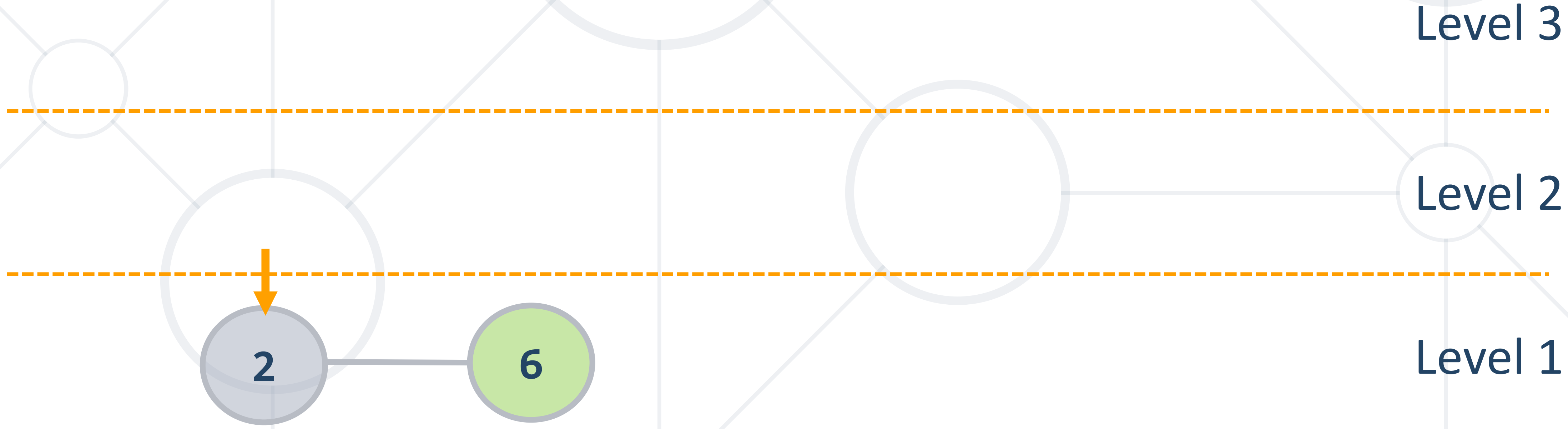
AA Tree Insertion #1

- Insert: **8**



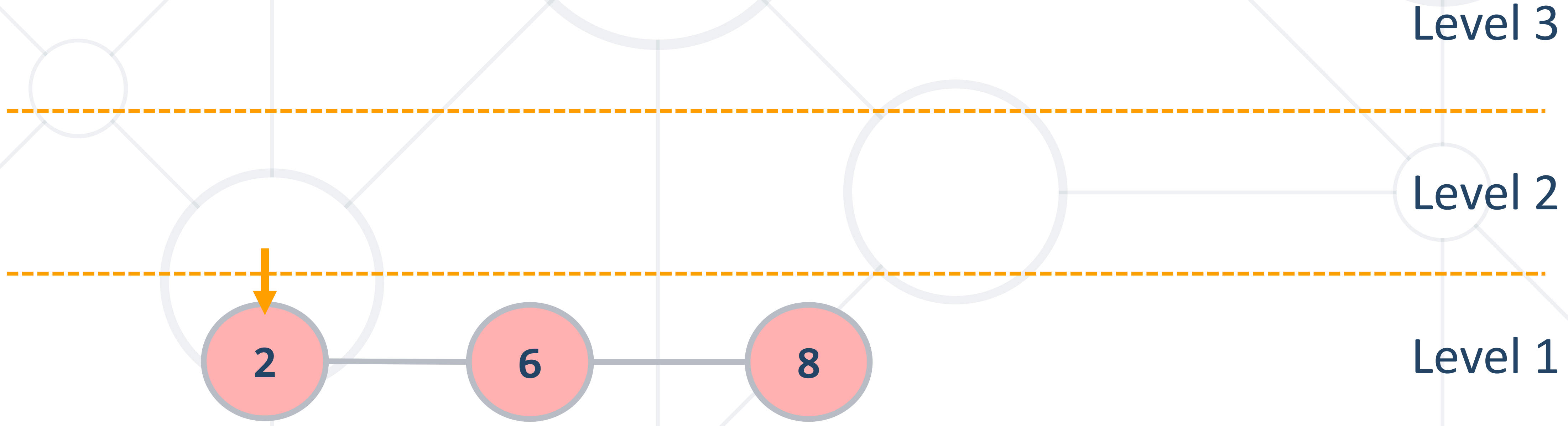
AA Tree Insertion #1

- Insert: 8



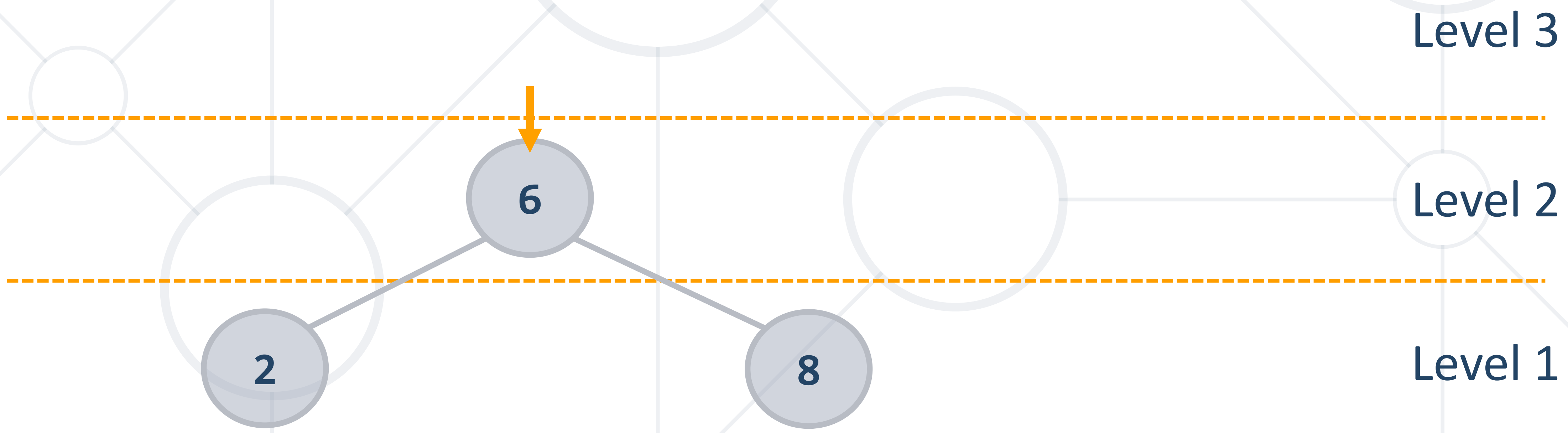
AA Tree Insertion #1

- Two consecutive right horizontal links
- Rotate 2 left (split)



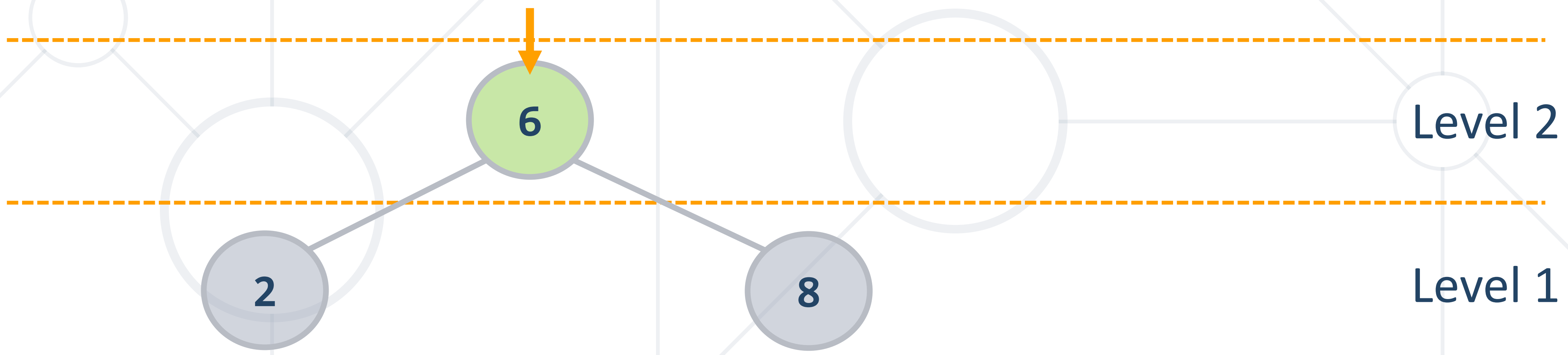
AA Tree Insertion #1

- Insert: **16**



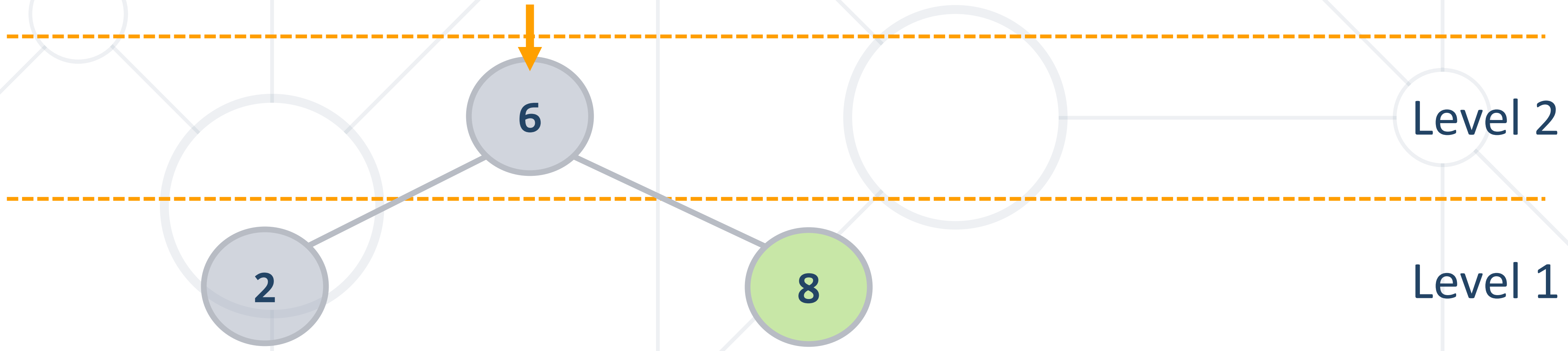
AA Tree Insertion #1

- Insert: **16**



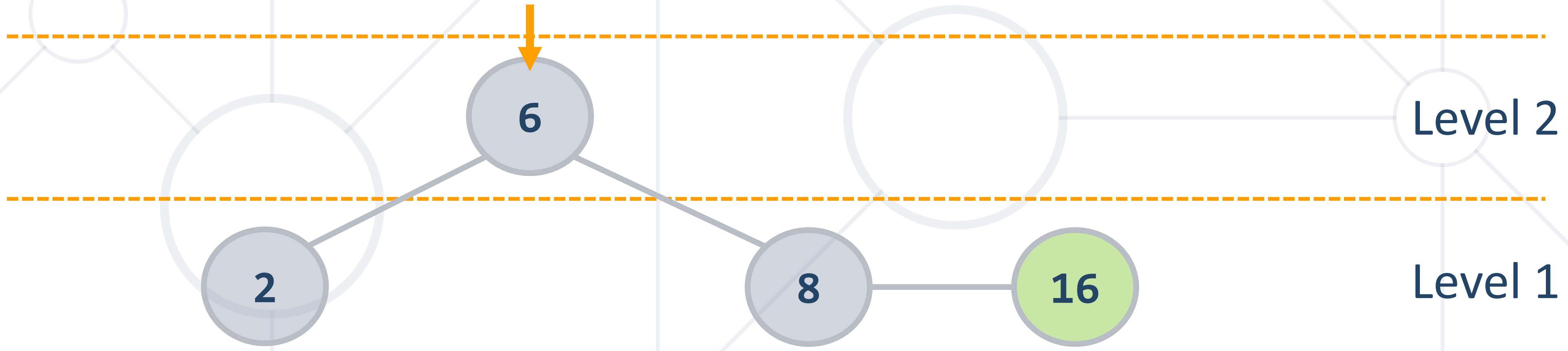
AA Tree Insertion #1

- Insert: **16**



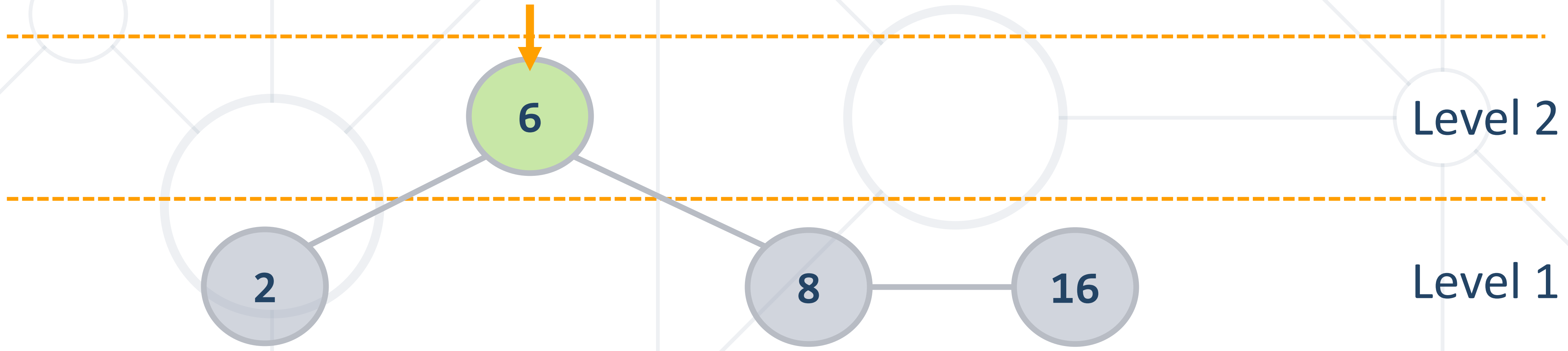
AA Tree Insertion #1

- Insert: **16**



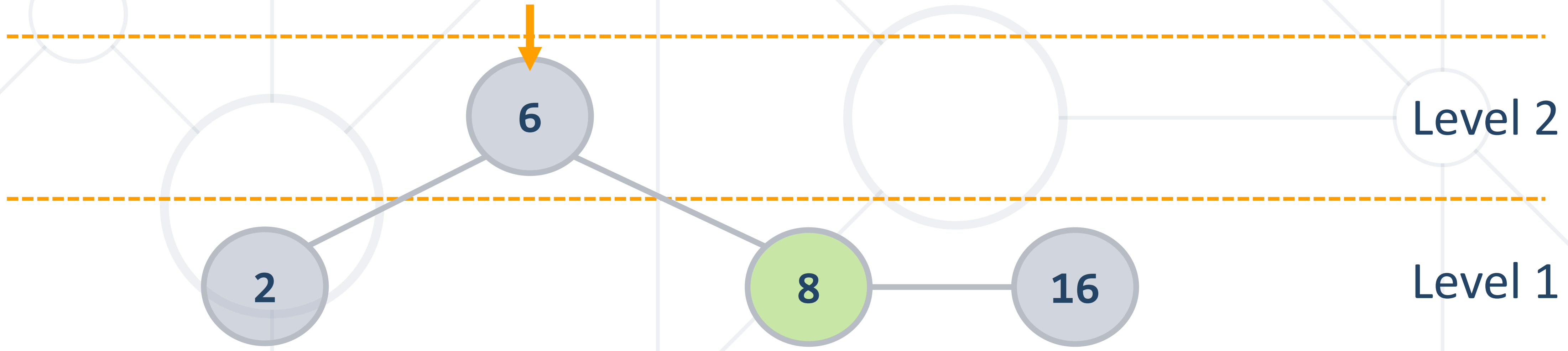
AA Tree Insertion #1

- Insert: **10**



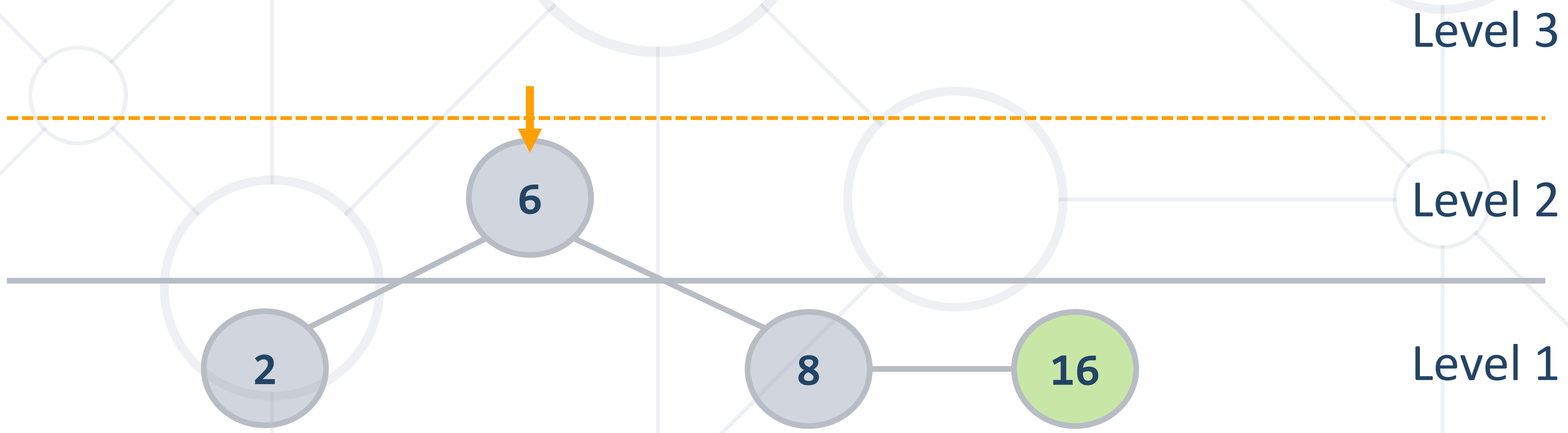
AA Tree Insertion #1

- Insert: 10



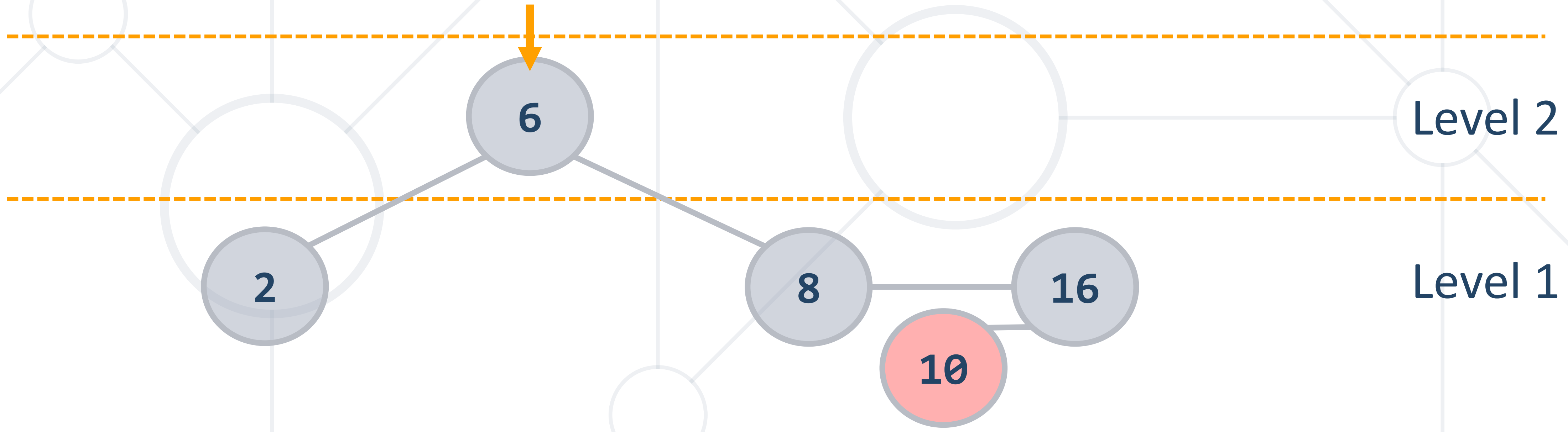
AA Tree Insertion #1

- Insert: 10



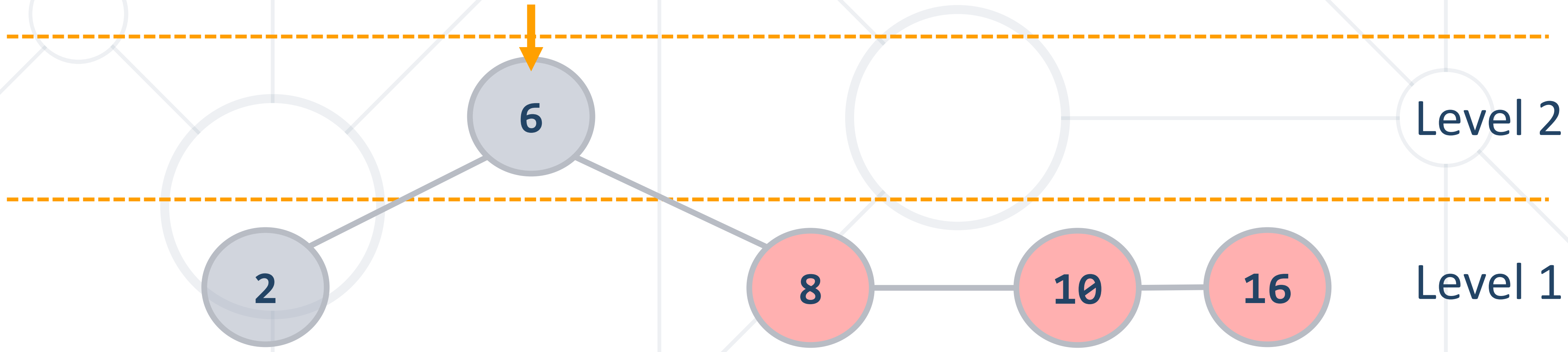
AA Tree Insertion #1

- Horizontal left link not allowed
- Rotate 16 right (skew)



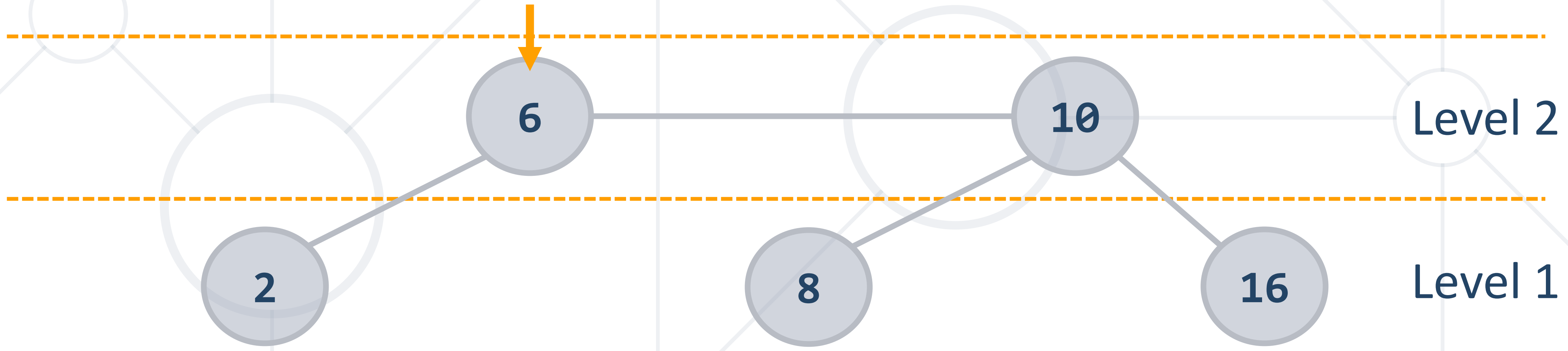
AA Tree Insertion #1

- Two consecutive right horizontal links
- Rotate 8 left (split)



AA Tree Insertion #1

- Two consecutive right horizontal links
- Rotate 8 left (split)



Trees - Quiz

TIME'S

- Consider **web application** in which
  - **searches** are **far more frequent** than **insertions/deletions**
- Which of the following do you prefer:
  - AVL
  - Linked List
  - Red-Black
  - B+

Trees - Quiz

TIME'S UP!

- Consider web application in which
  - searches are far more frequent than insertions/deletions
- Which of the following do you prefer:
  - AVL ✓
  - Linked List
  - Red-Black
  - B+

AVL trees are more rigidly balanced, so they have faster search

Summary

- Red-Black Trees are widely used
  - Insertion is easy
  - Balance by color
  - Color is a single byte
- AA Trees
  - Insertion algorithm

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License