AVL Trees and AA Trees

More Balanced Binary Search Tree

SoftUni Team

Technical Trainers

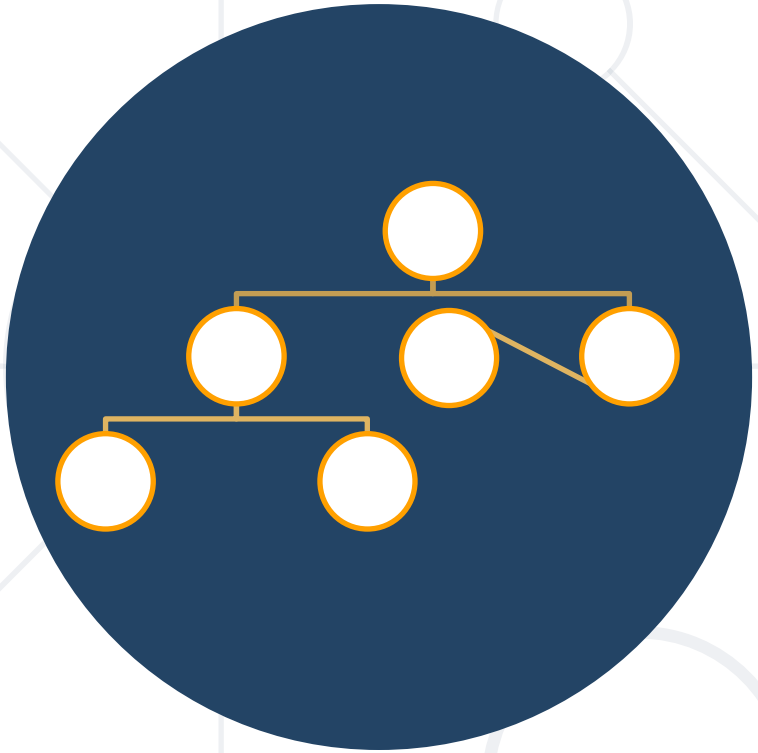
Software University

SoftUni

Software University

https://softuni.bg

Table of Contents



AA Tree

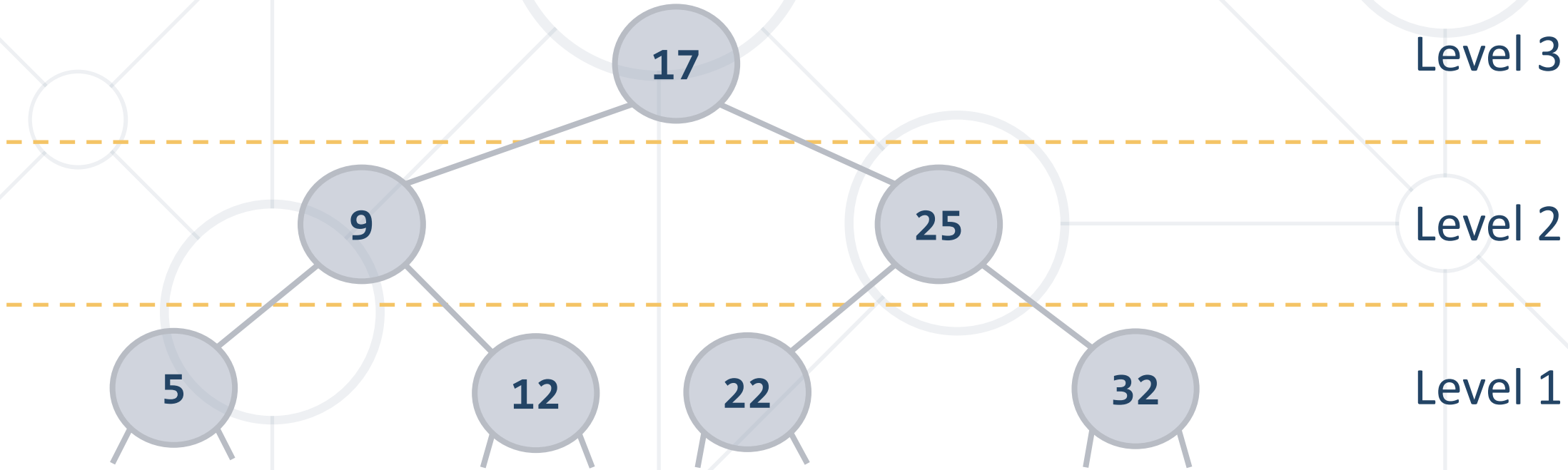
Definition

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

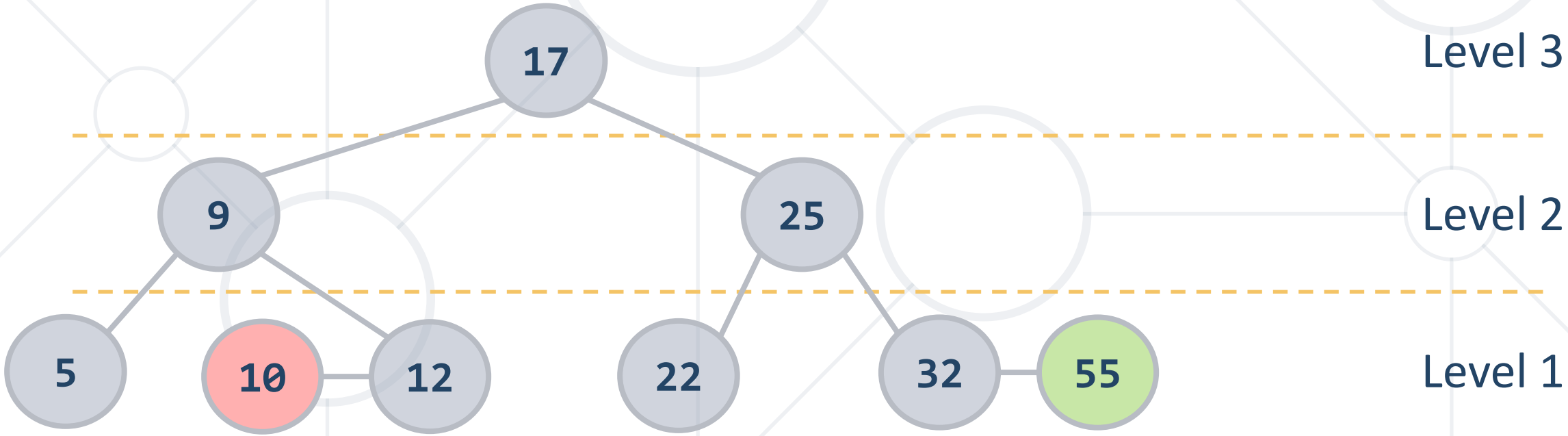


AA Tree

- AA tree invariants
  - The **level** of every **leaf node** is **1**
  - 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**

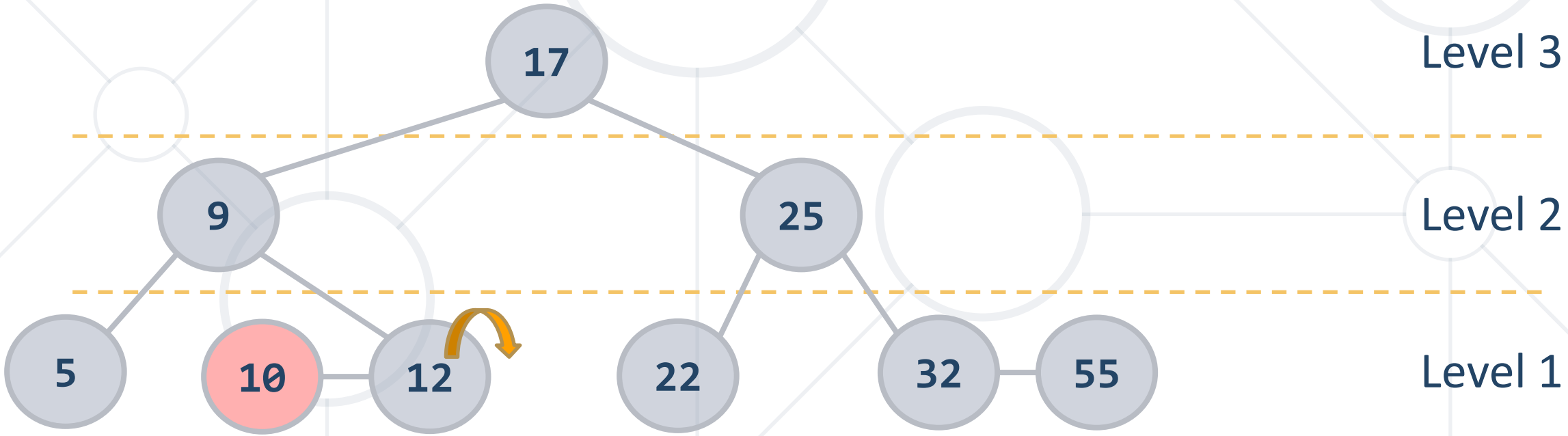
AA Tree

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



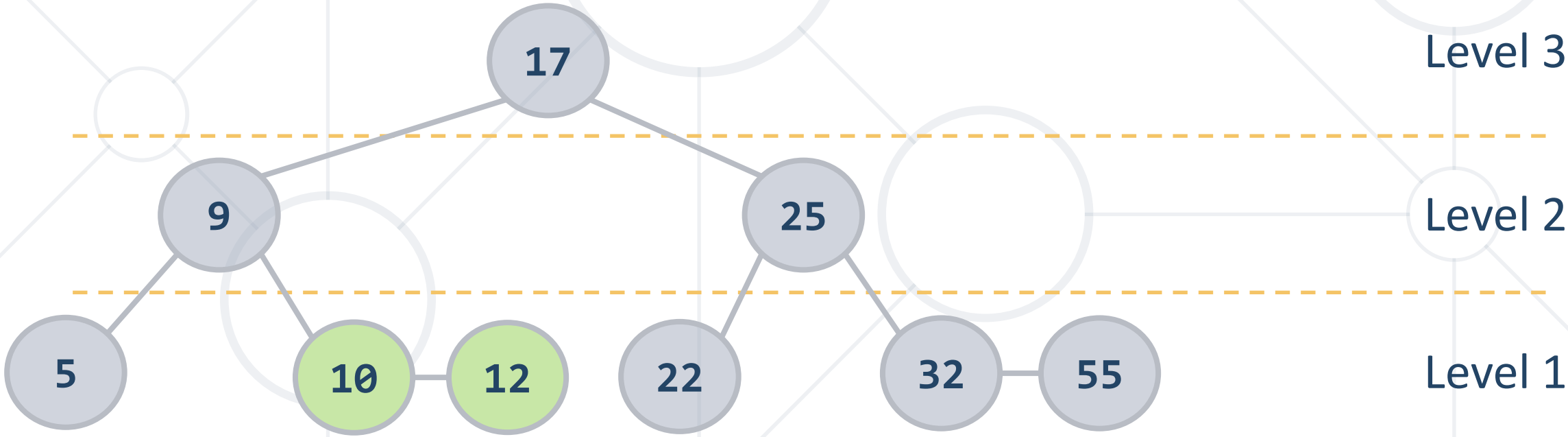
Skew

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



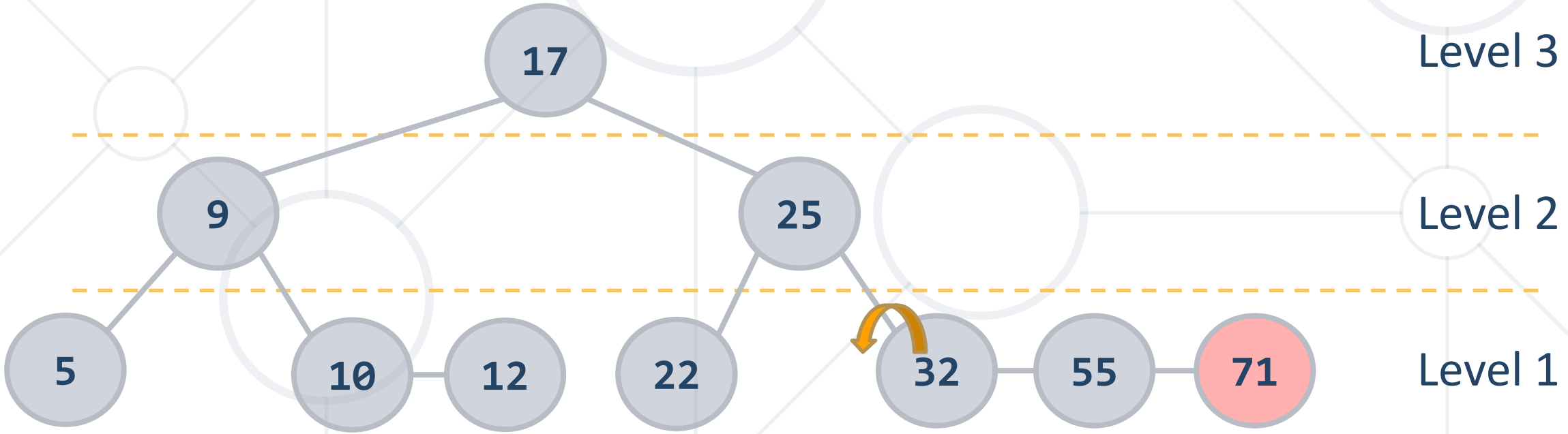
Skew (2)

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



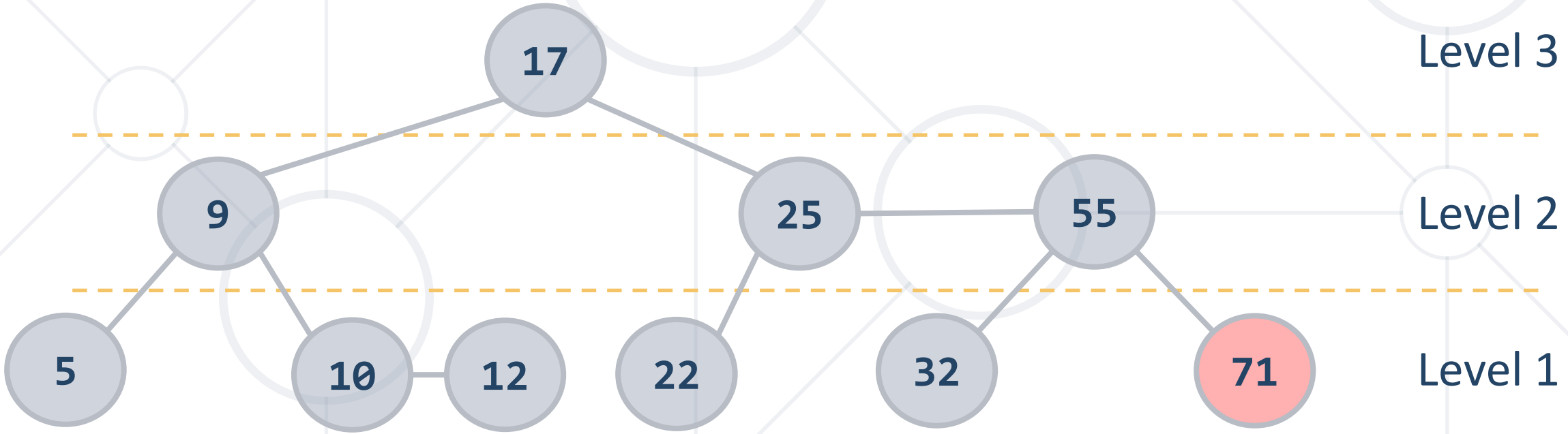
Split

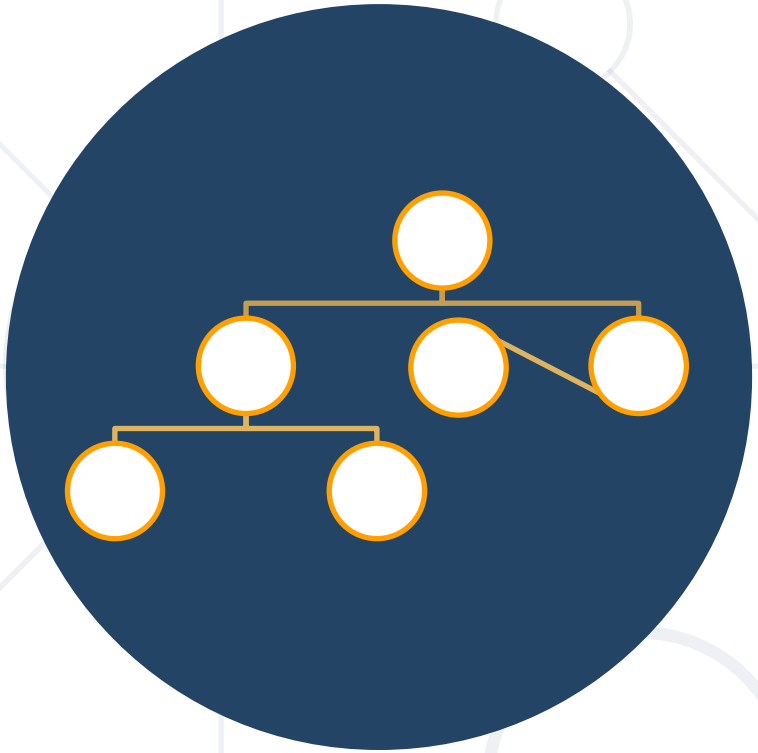
- Split operation is a **single left rotation**
- Split when an insertion or deletion **two consecutive right horizontal links**



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

AA Tree Insertion #1

- Insert: **2**

AA Tree Insertion #1

- Insert: **2**

AA Tree Insertion #1

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

AA Tree Insertion #1

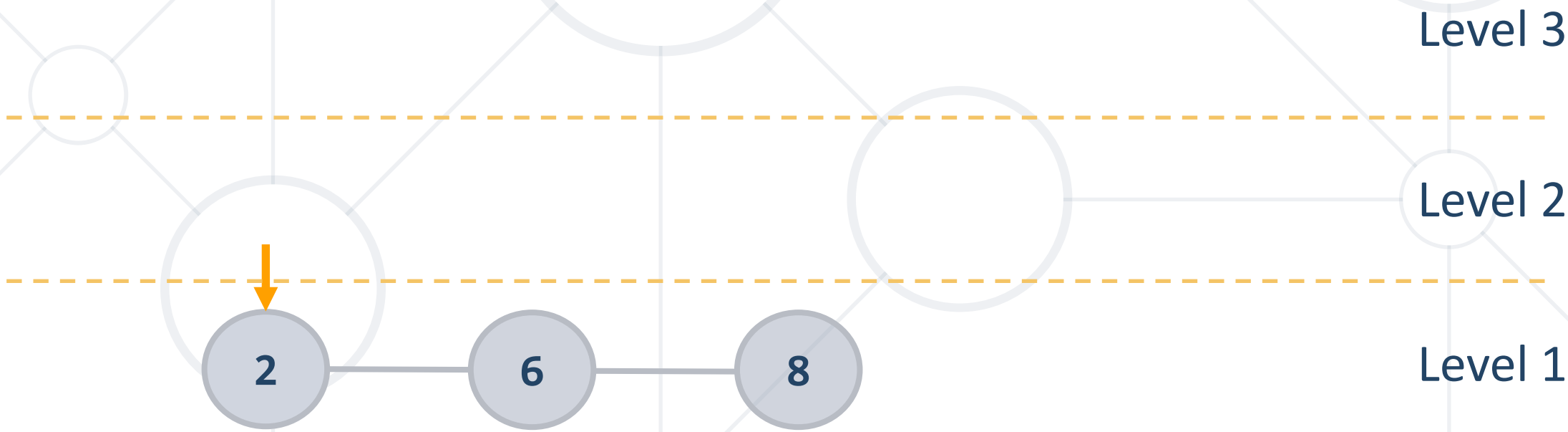
- **Left horizontal** link is **not allowed**
- Rotate 6 right (skew)

AA Tree Insertion #1

- Insert: **8**

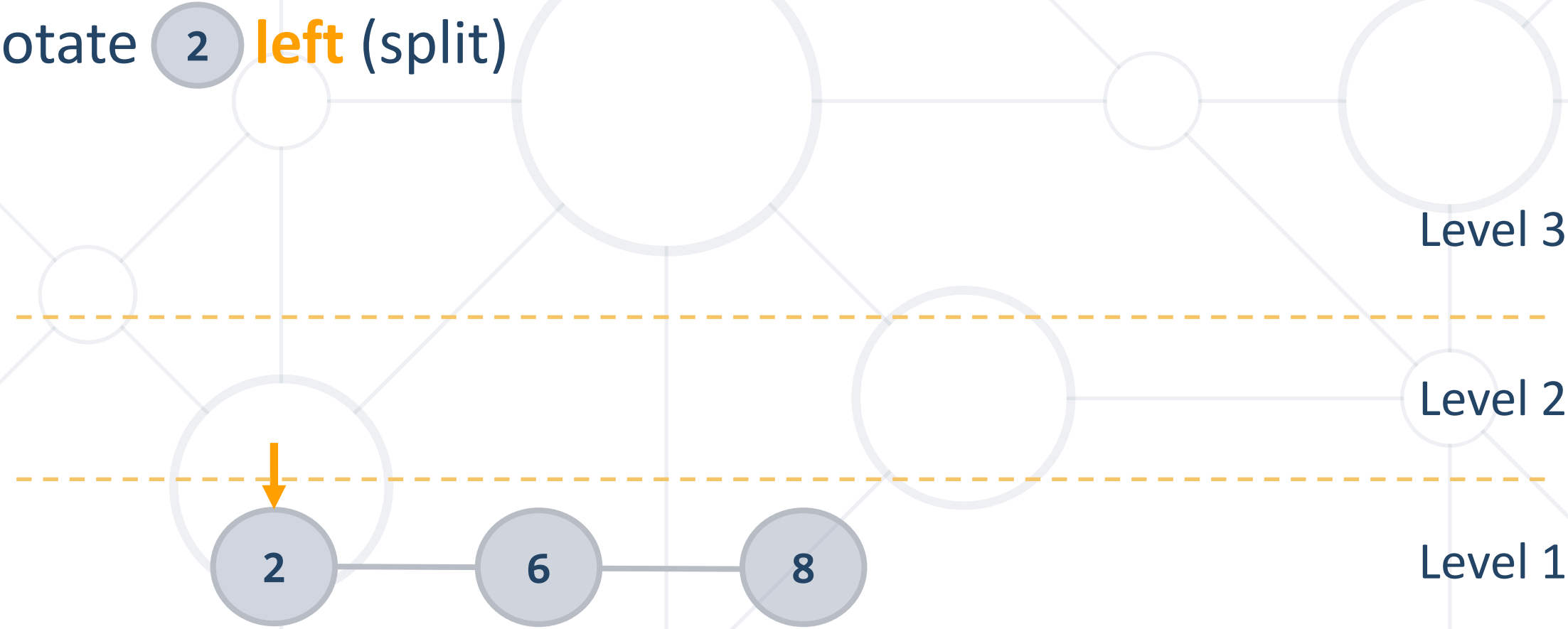
AA Tree Insertion #1

- Insert: **8**



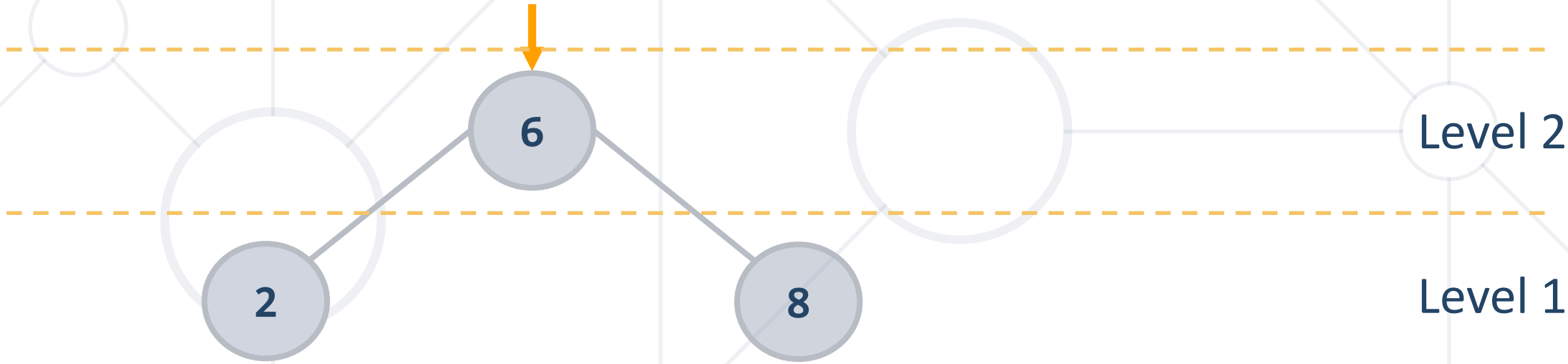
AA Tree Insertion #1

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



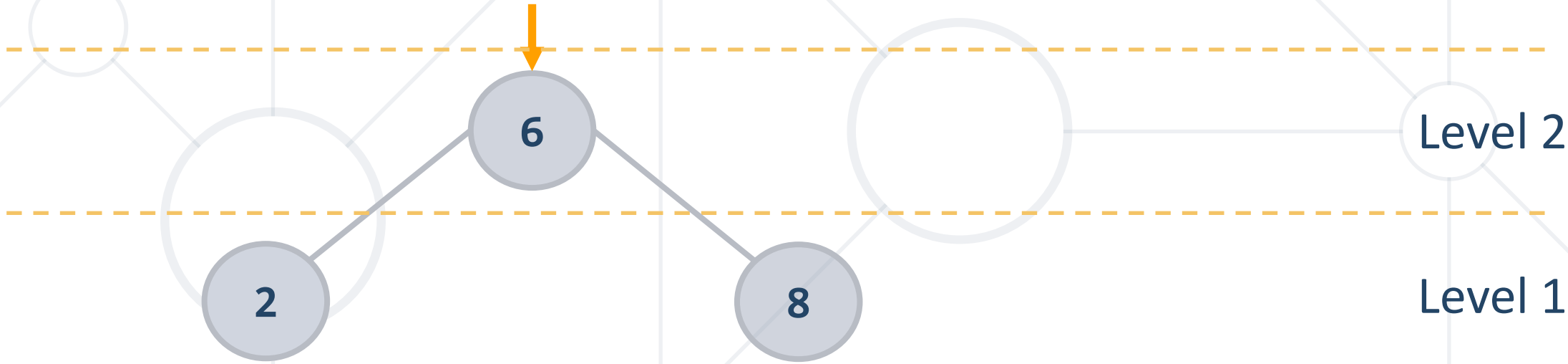
AA Tree Insertion #1

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



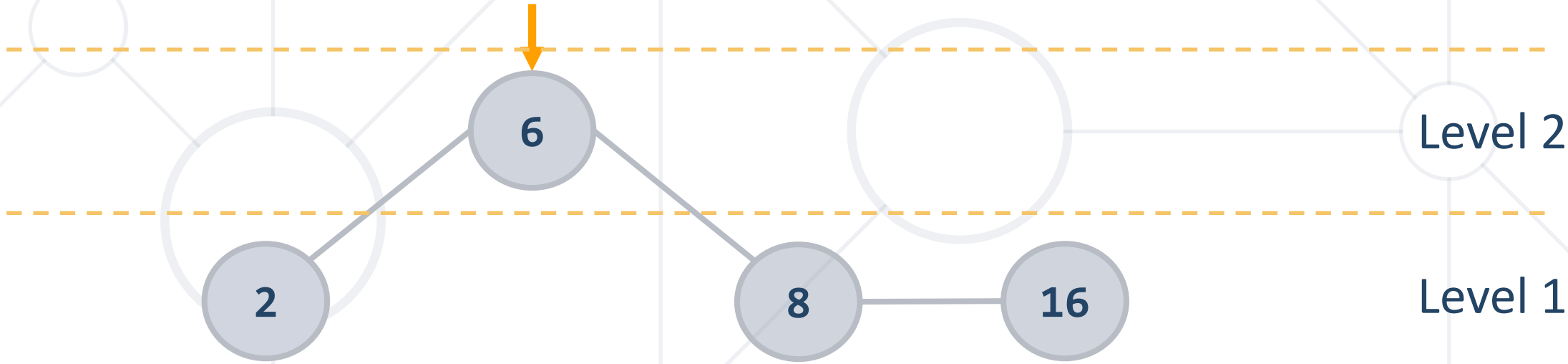
AA Tree Insertion #1

- Insert: **16**



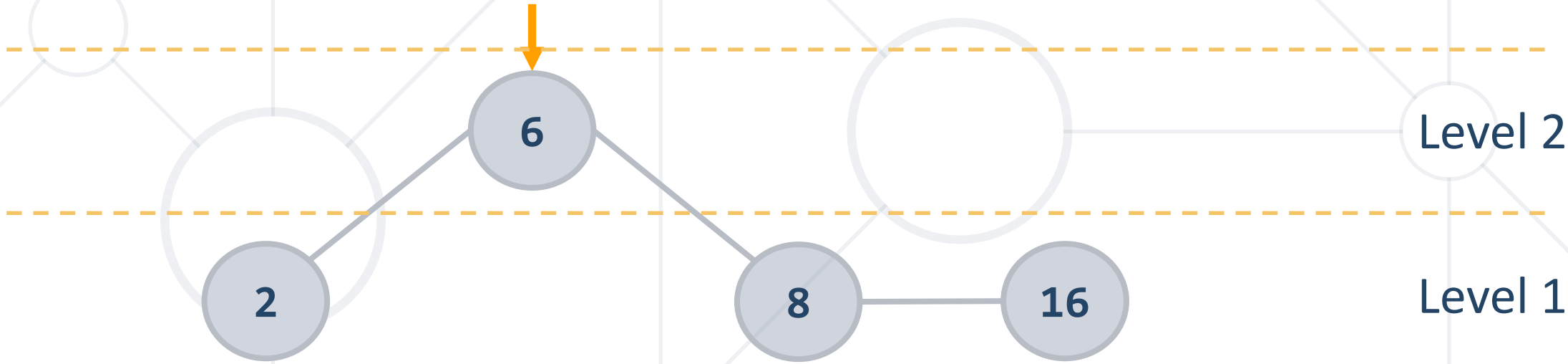
AA Tree Insertion #1

- Insert: **16**



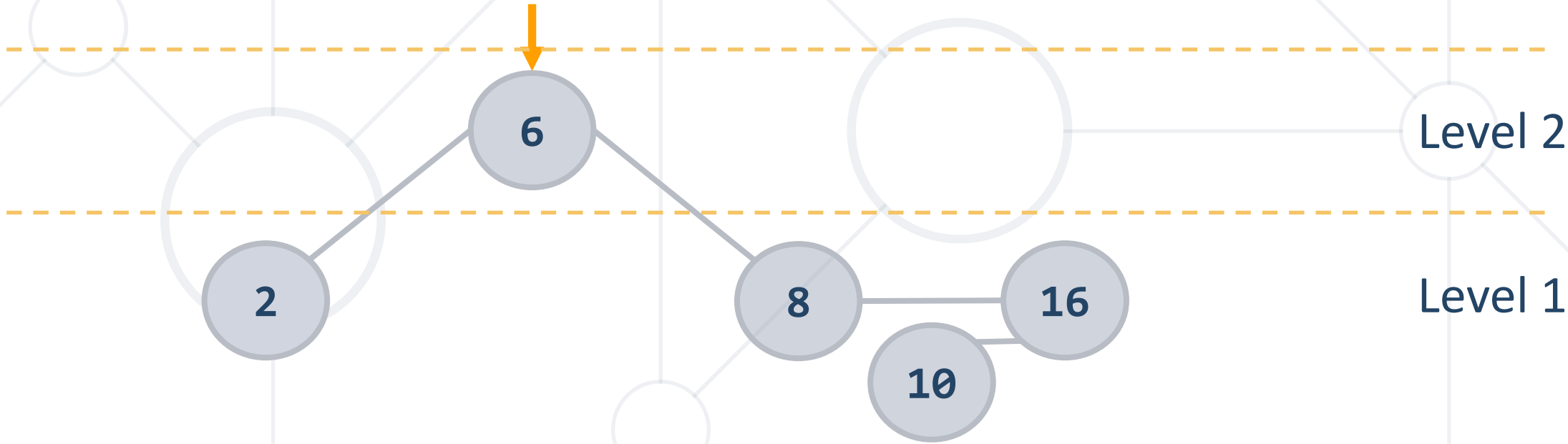
AA Tree Insertion #1

- Insert: **10**



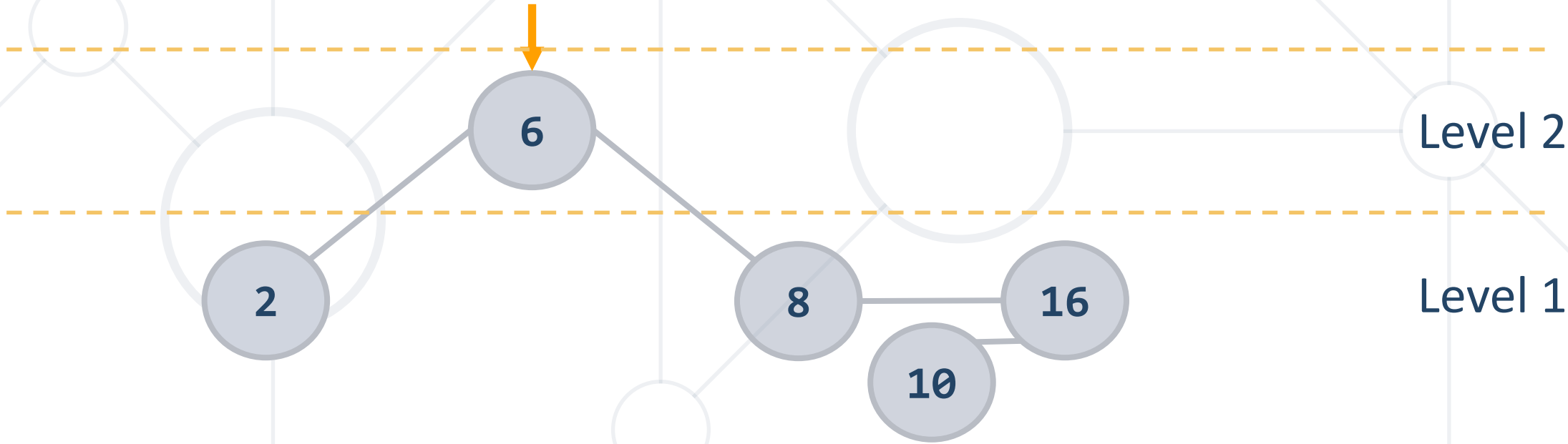
AA Tree Insertion #1

- Insert: **10**



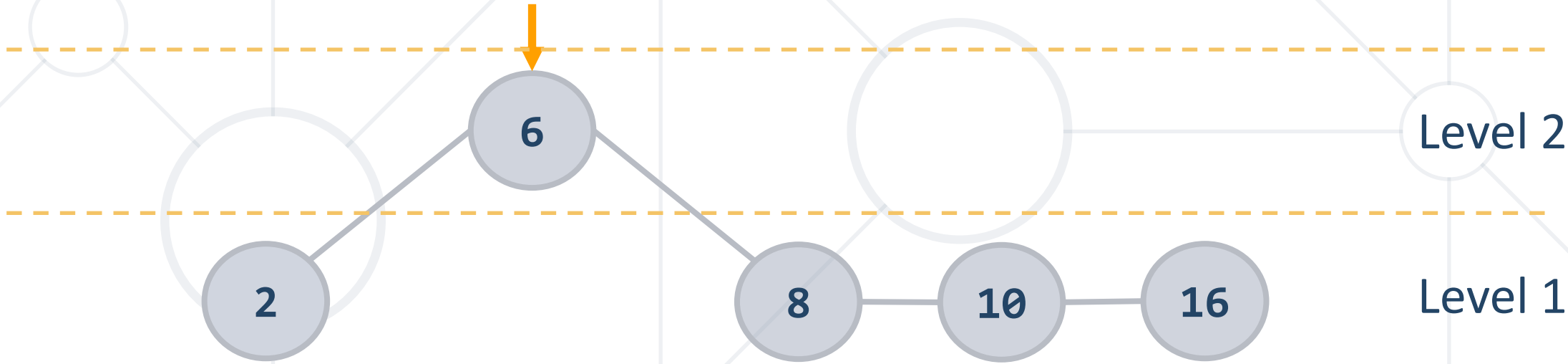
AA Tree Insertion #1

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



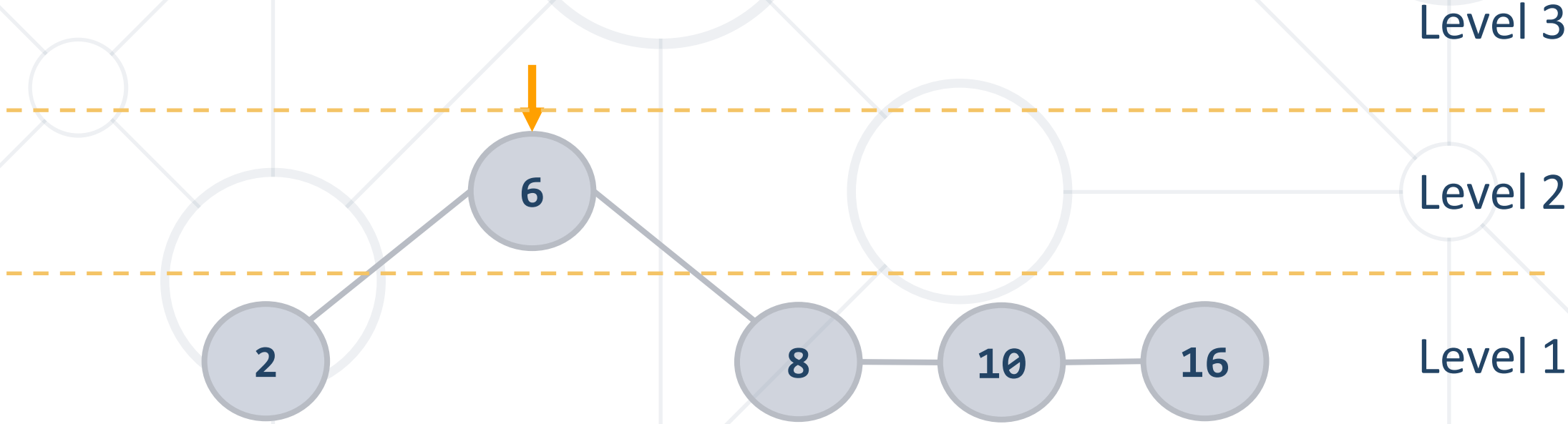
AA Tree Insertion #1

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



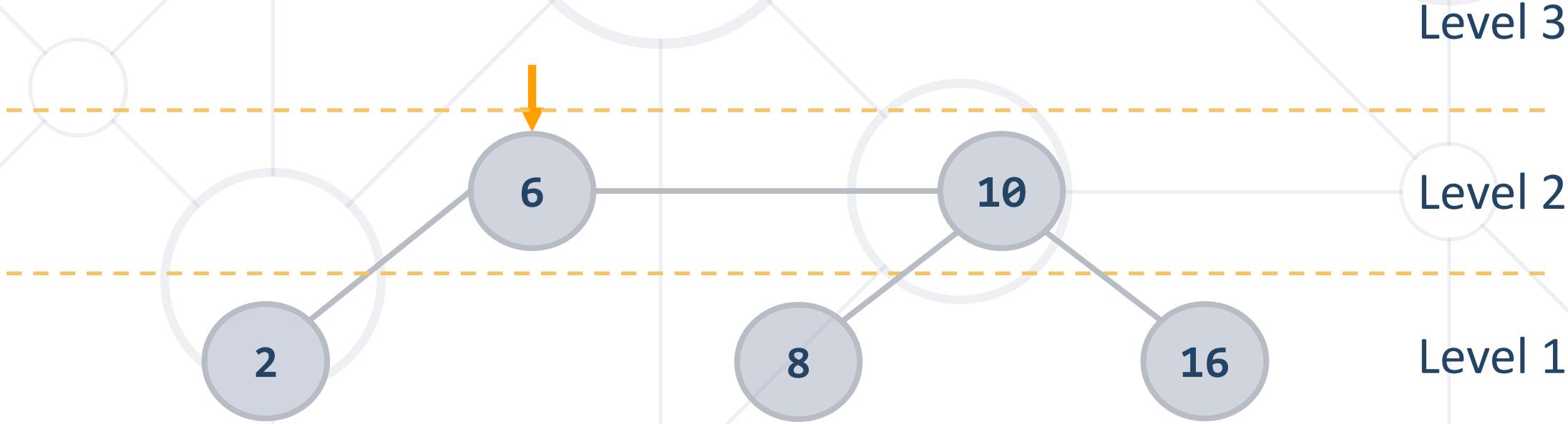
AA Tree Insertion #1

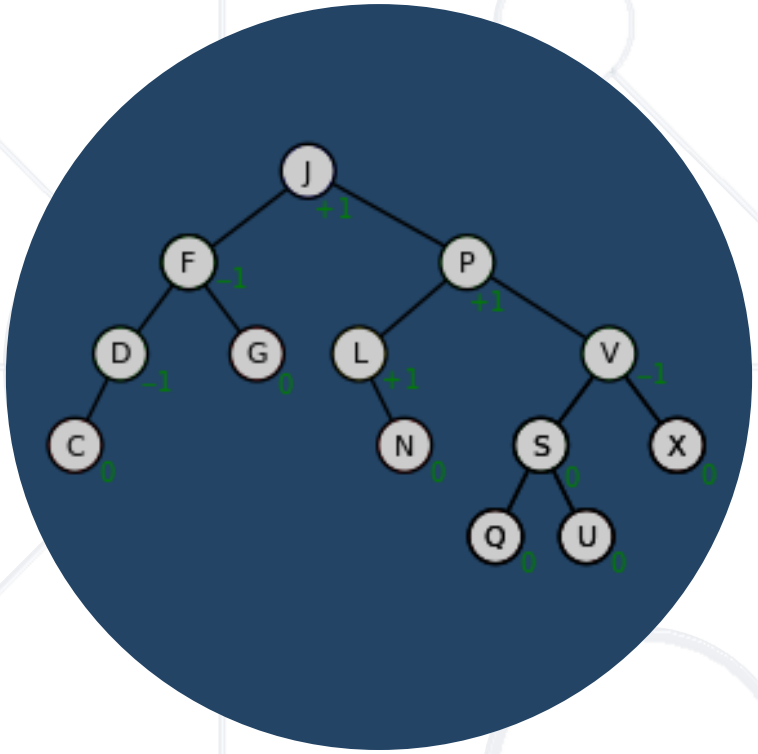
- Two consecutive right horizontal links
- Rotate 8 left (split)



AA Tree Insertion #1

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





AVL Trees

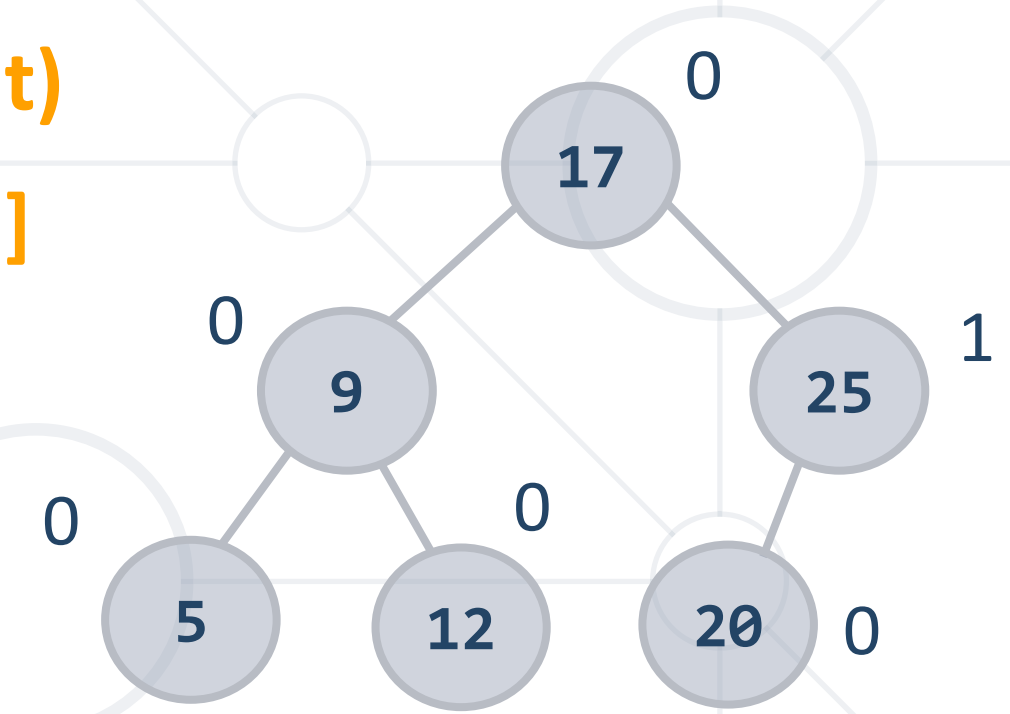
Properties and Rotations

AVL Tree

- AVL tree is a self-balancing binary-search tree (visualization)
  - Height of two subtrees can **differ by at most 1**
- **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

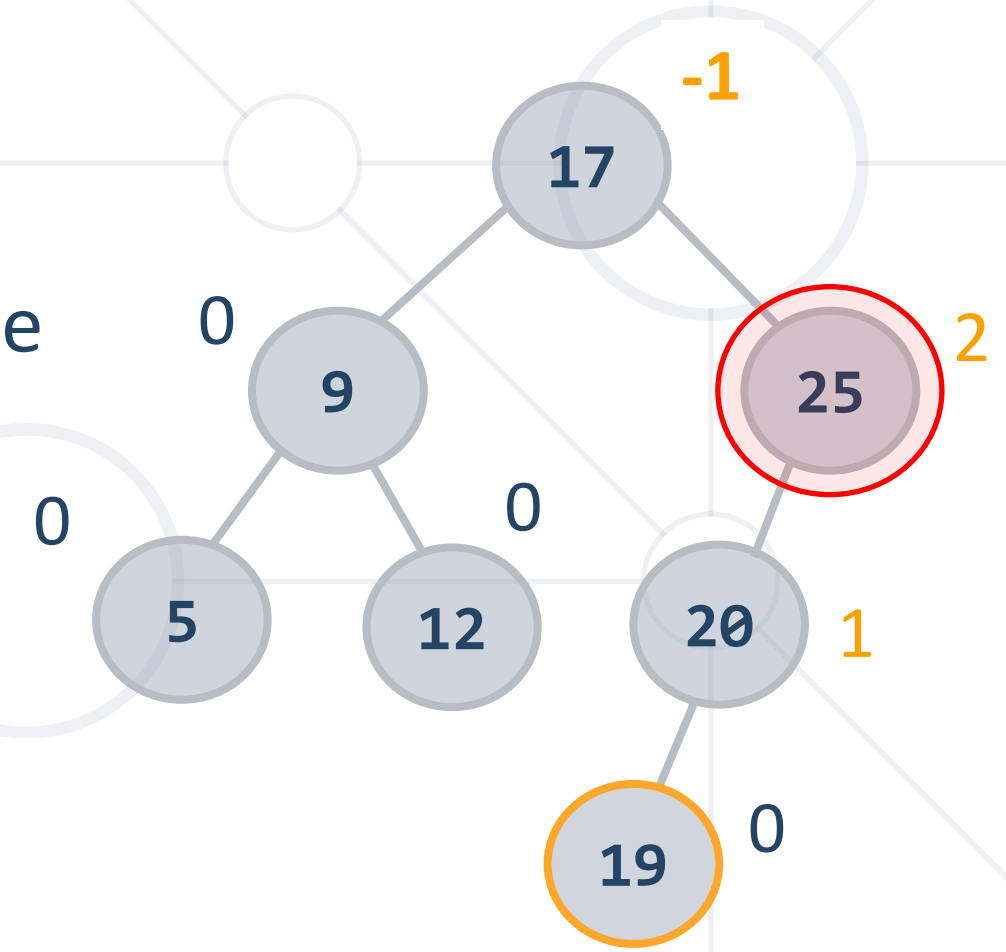
AVL Tree Rebalancing

- Height difference is measured by a balance factor (BF)
- **BF(Tree) = Height(Left) – Height(Right)**
  - BF of any node is in the range **[-1, 1]**
  - If BF becomes **-2** or **2** rebalance



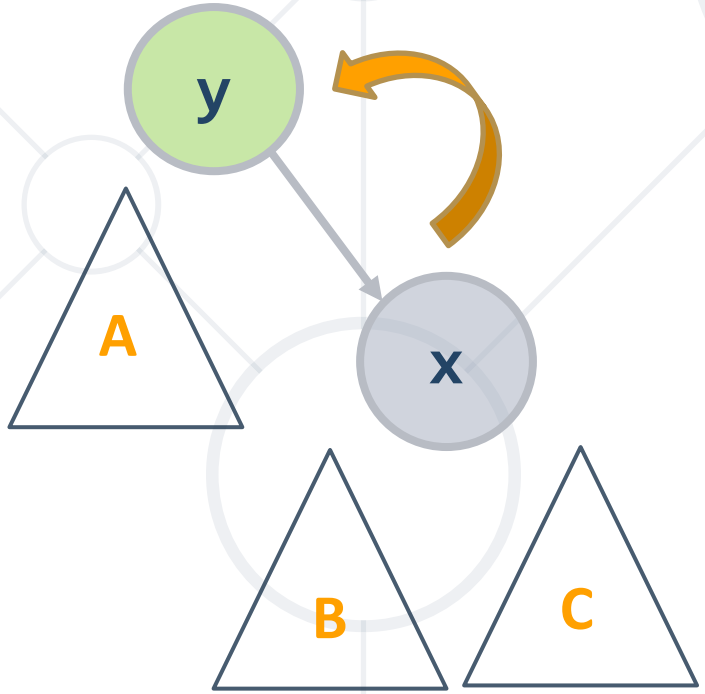
AVL Tree Rebalancing

- Rebalancing is done by **retracing**
  - Start from inserted node's parent and go up to root
  - Perform **rotations** to restore balance

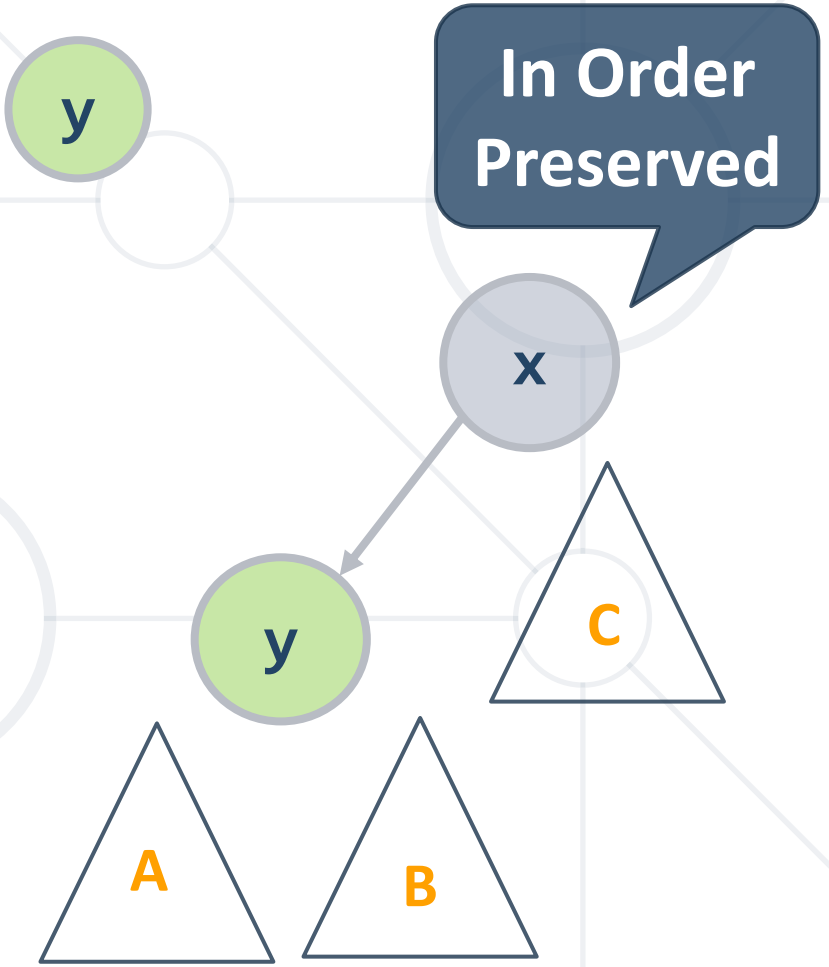


Left Rotation

- Set y to be child of x
- Set Left Child of x to be Right Child of y

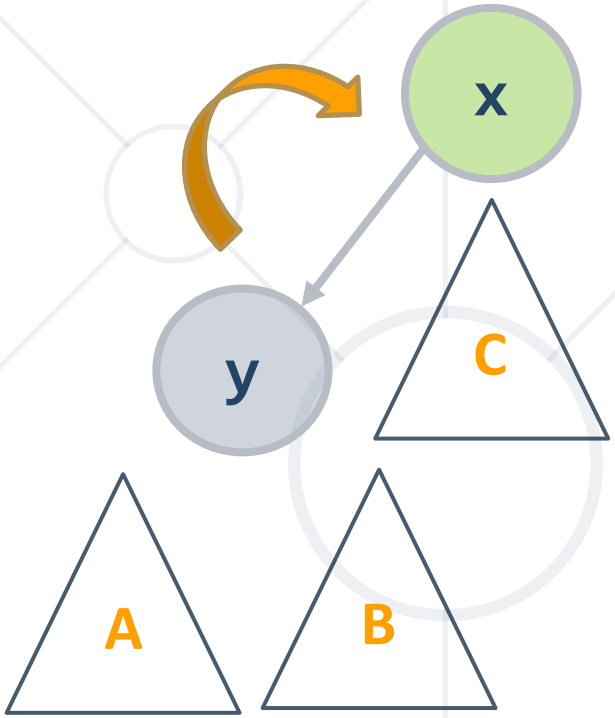


Left rotation (y)

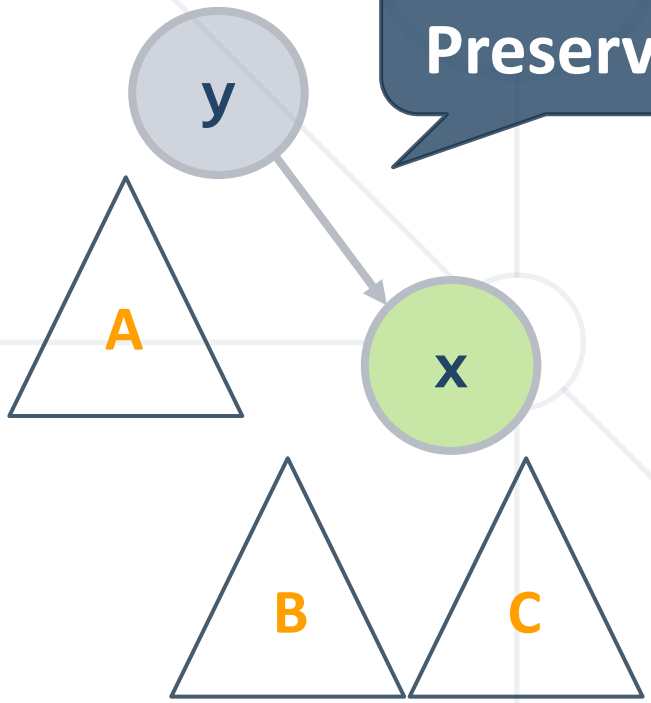


Right Rotation

- Set x to be child of y
- Set Right Child of y to be Left Child of x



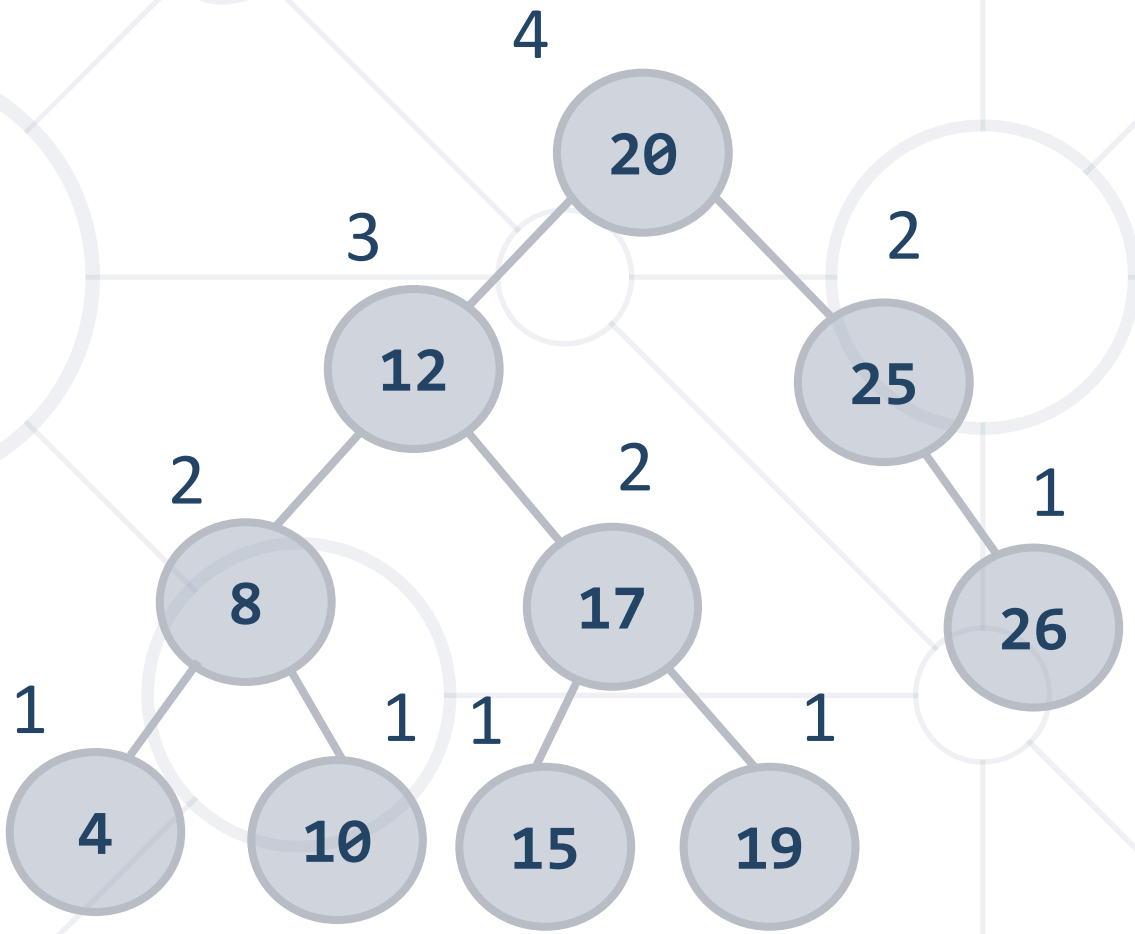
Right rotation (x)



In Order Preserved

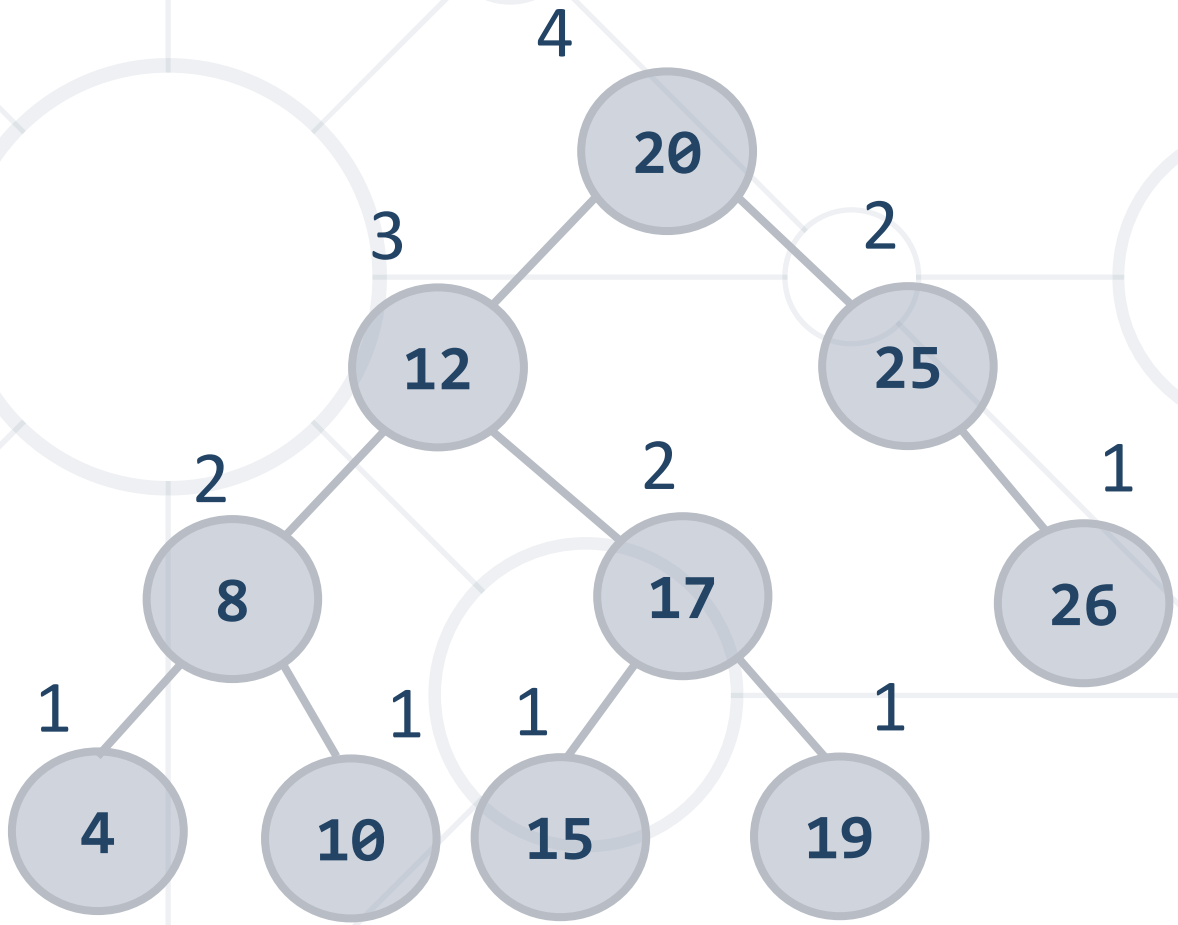
AVL Tree Insertion Algorithm

- Insert like in **ordinary BST**
- Retrace **up** to root
  - Modify balance / height
  - If balance factor **∉ [-1,1]** rebalance



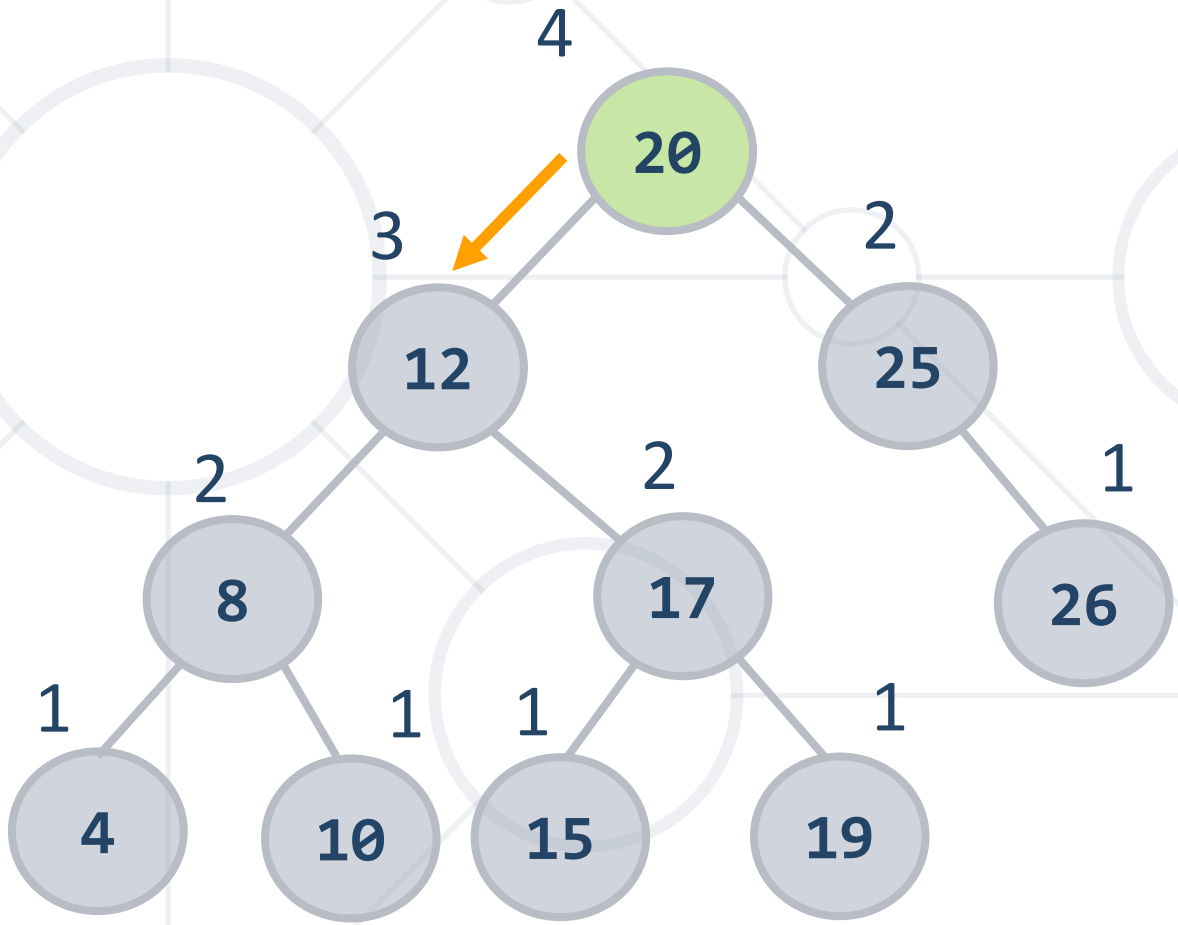
AVL Insertion

- Insert **11**



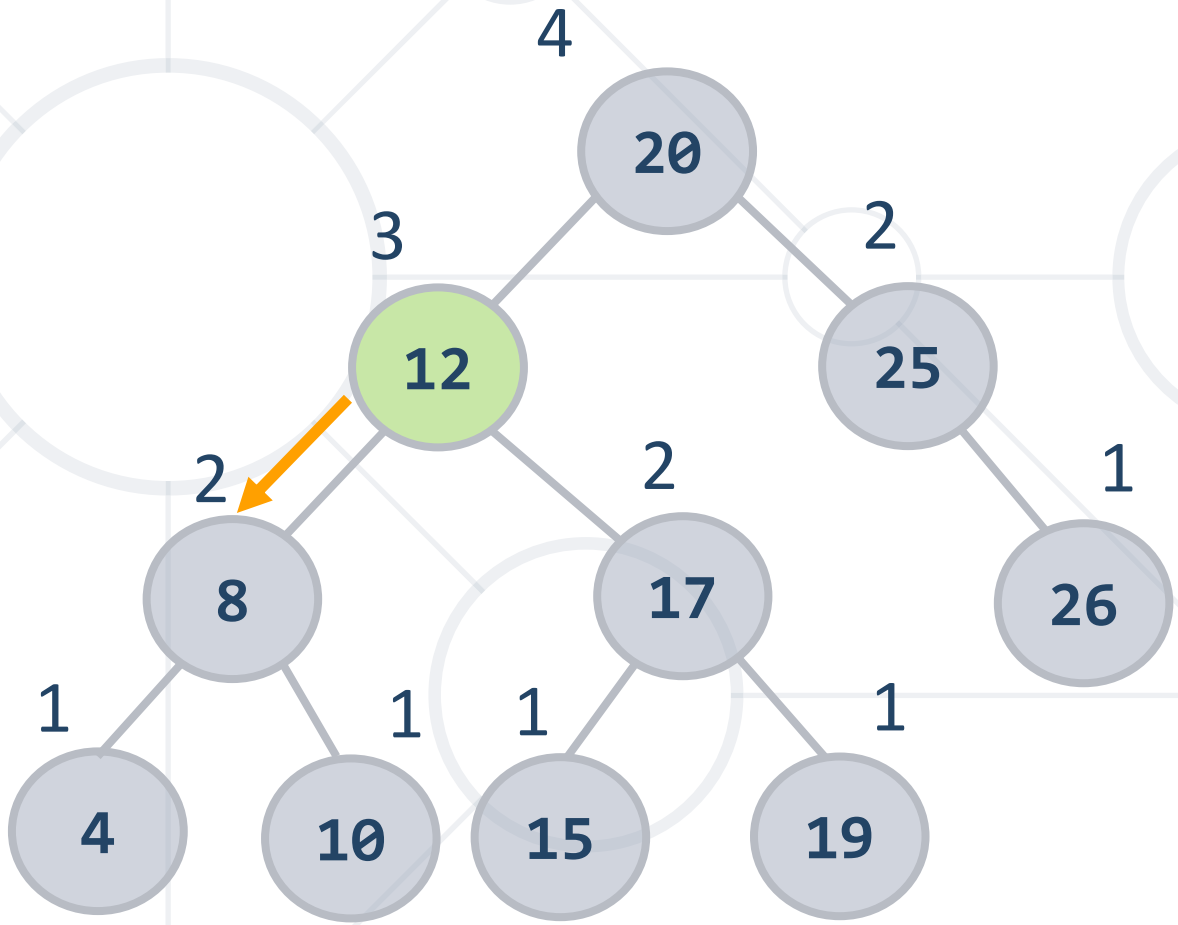
AVL Insertion

- 11 < 20 go left



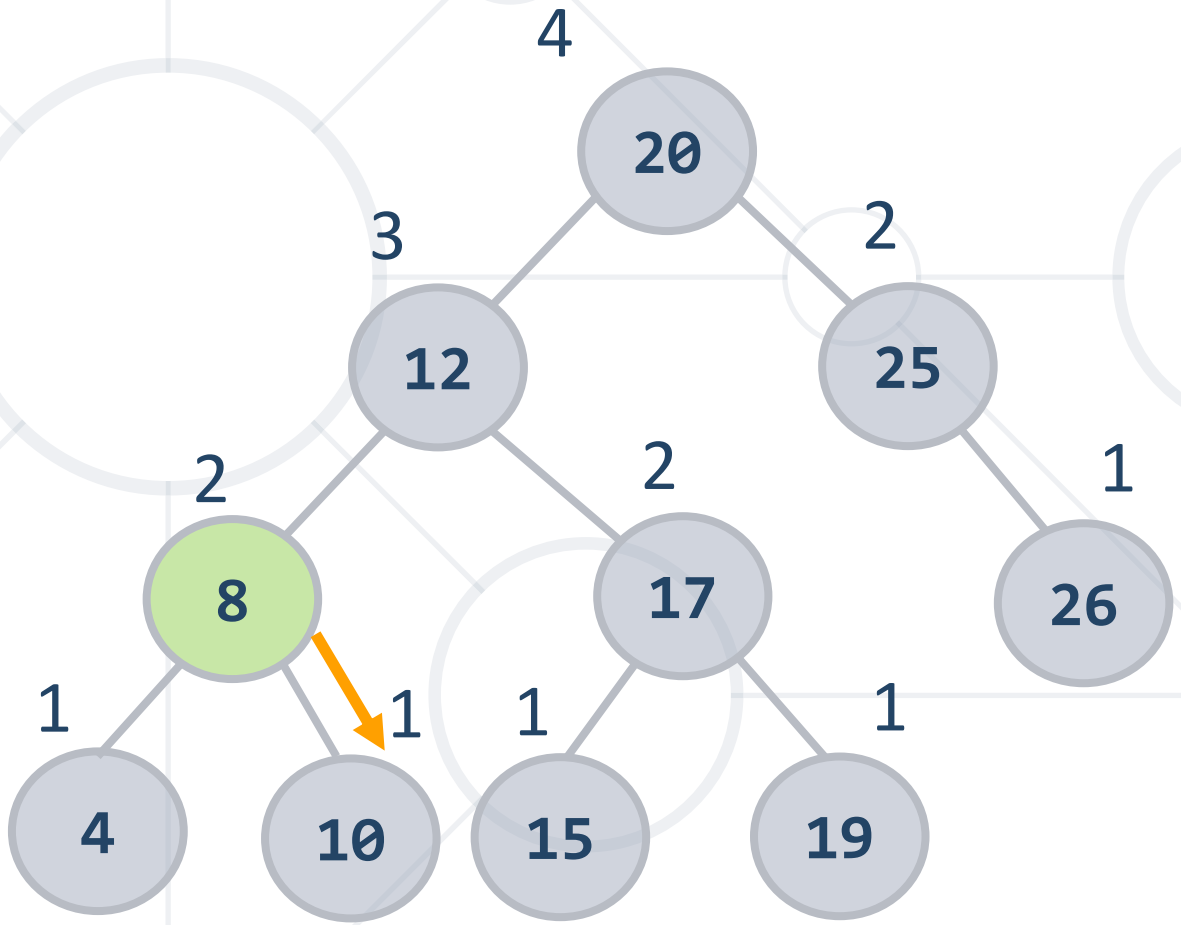
AVL Insertion

- 11 < 12 go left



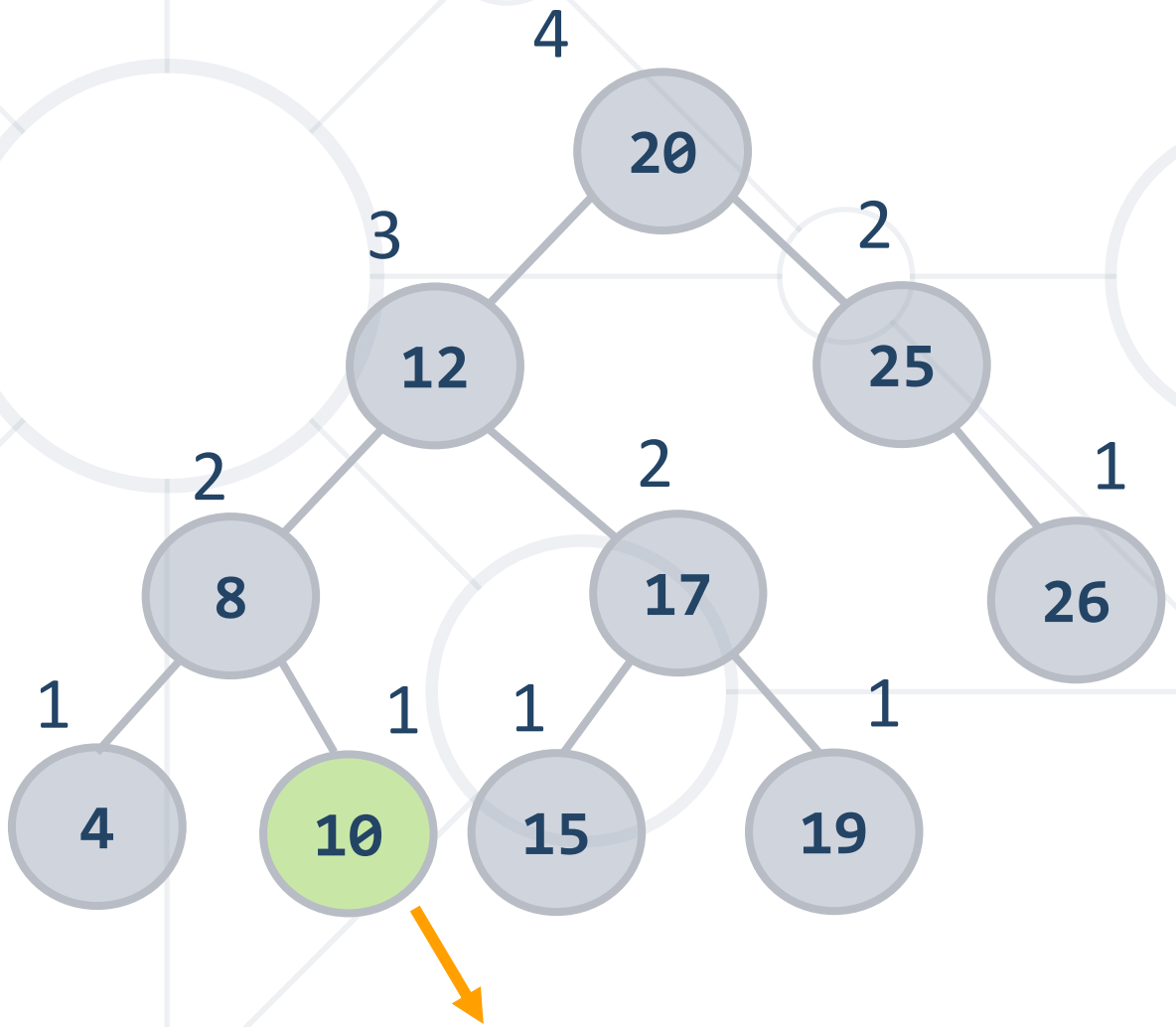
AVL Insertion

- 11 > 8 go right



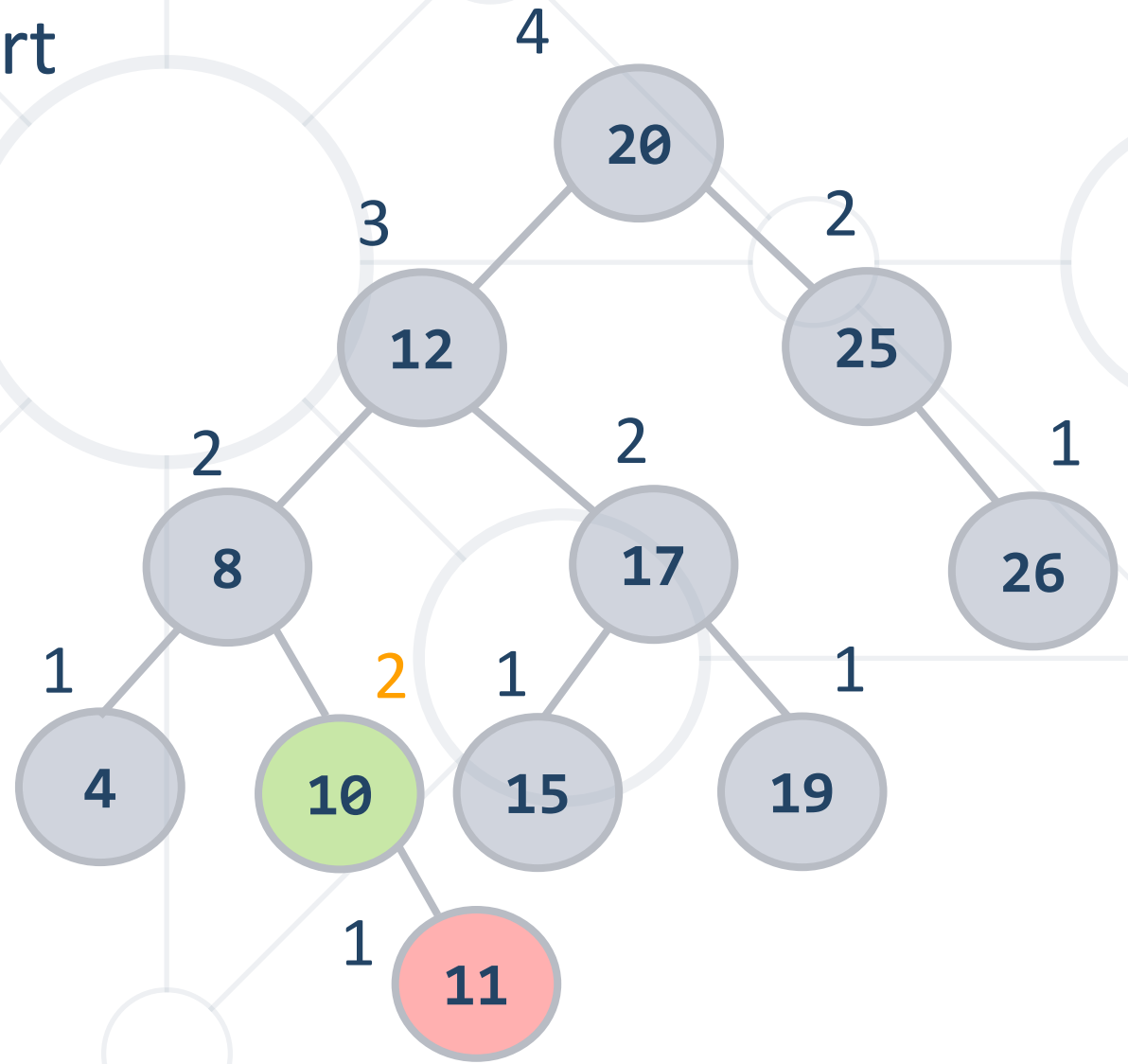
AVL Insertion

- 11 > 10 go right



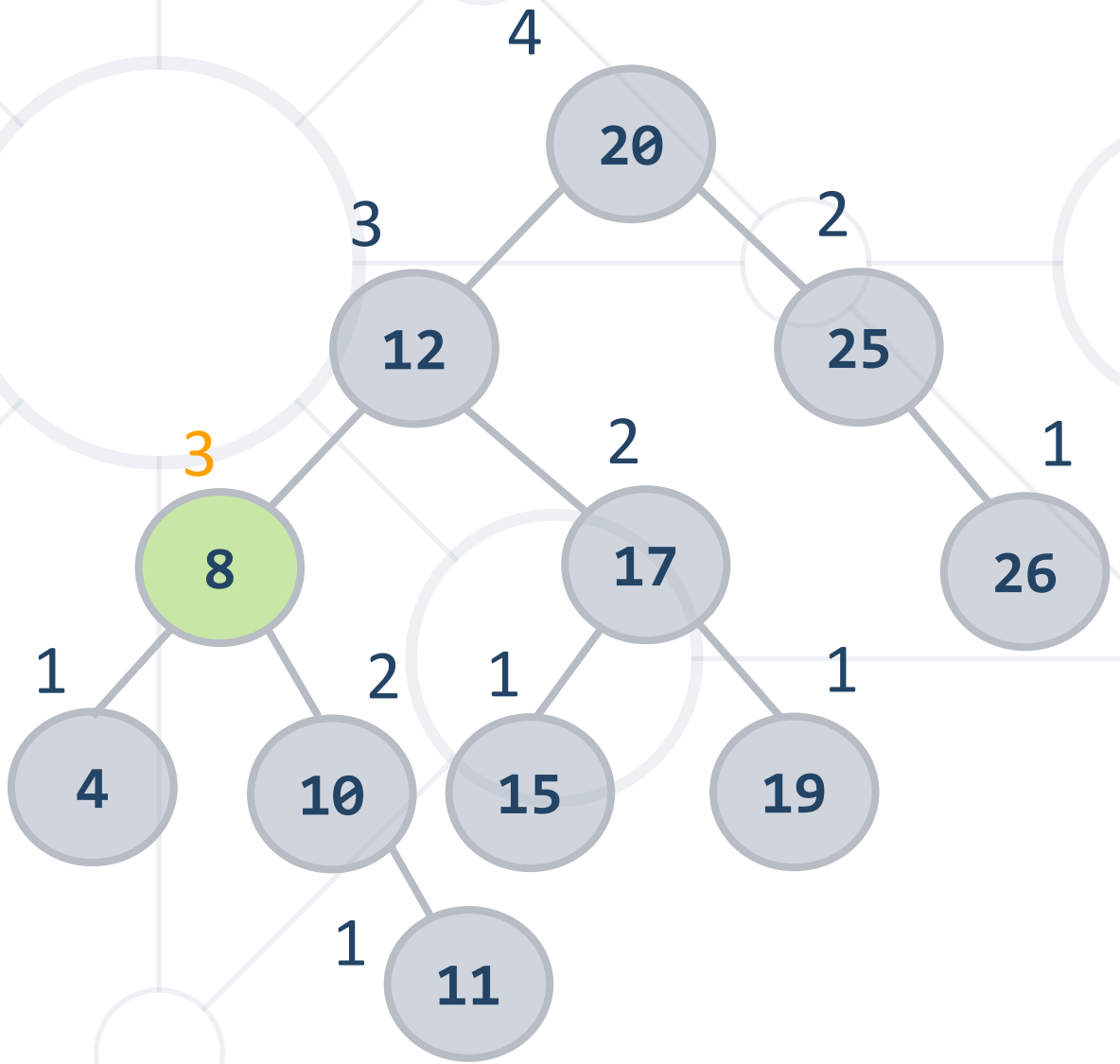
AVL Insertion

- Right node is **null** insert
- Update 10 height
- 10 balance is **-1**



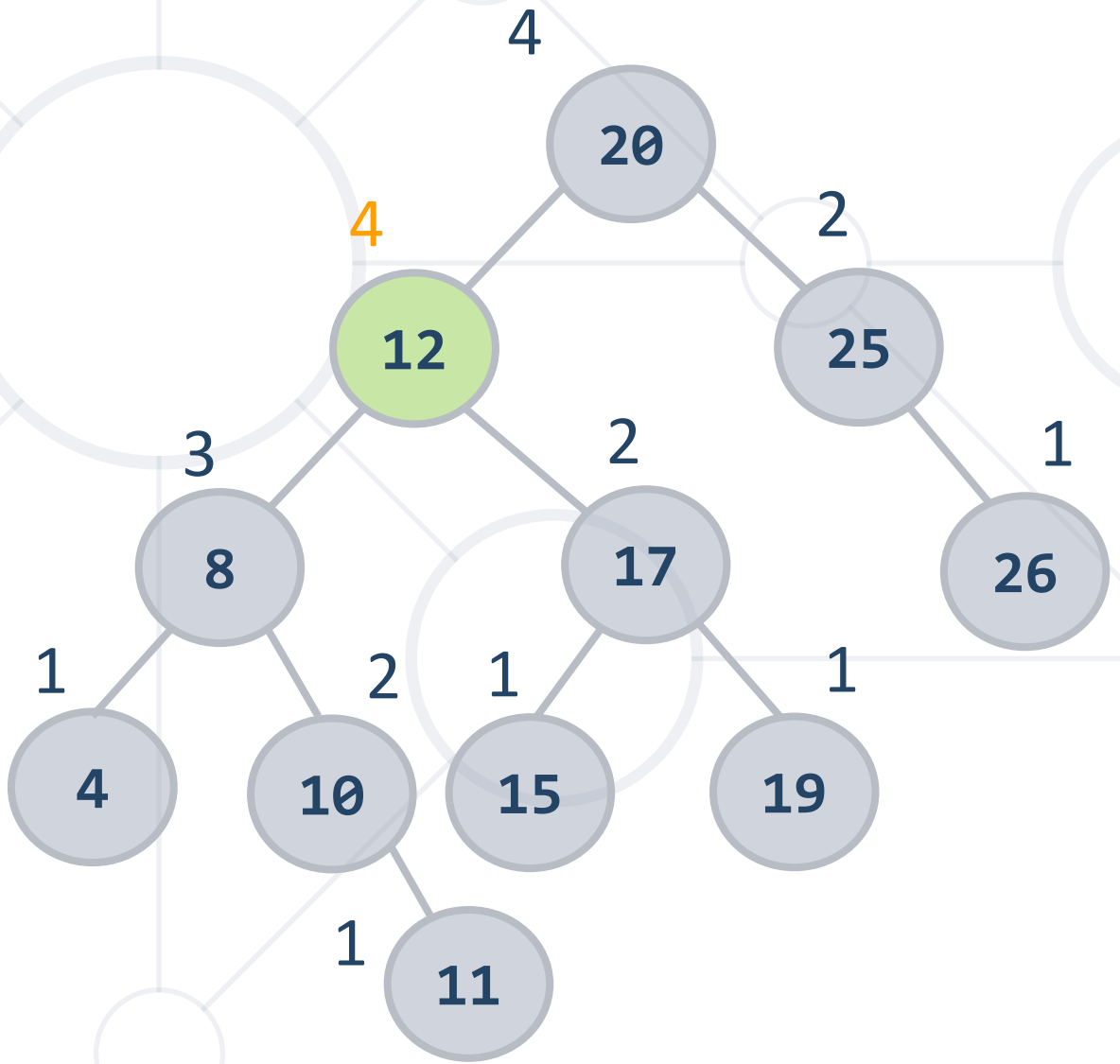
AVL Insertion

- Update 8 height
- 8 balance is -1



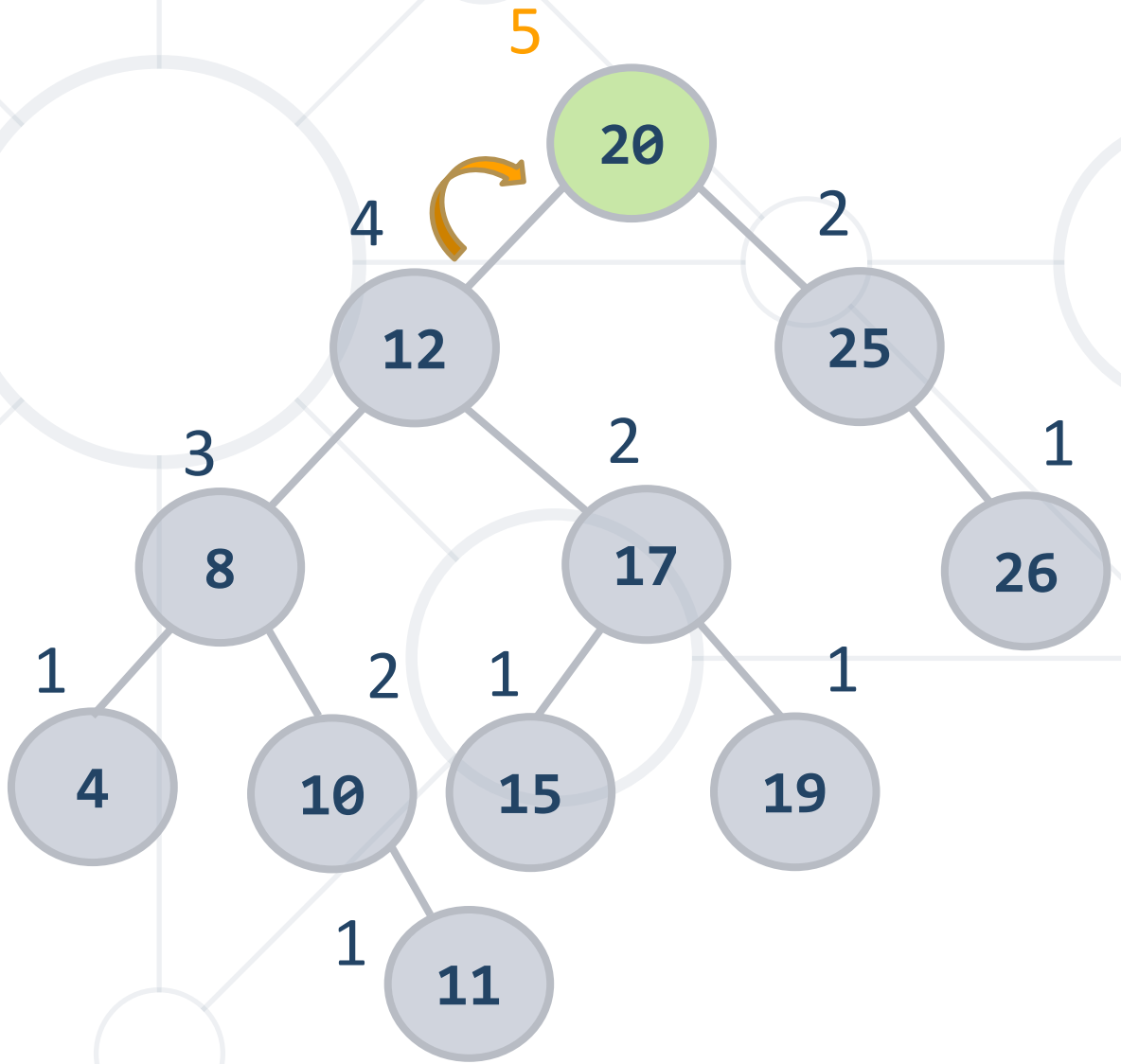
AVL Insertion

- Update 12 height
- 12 balance is 1



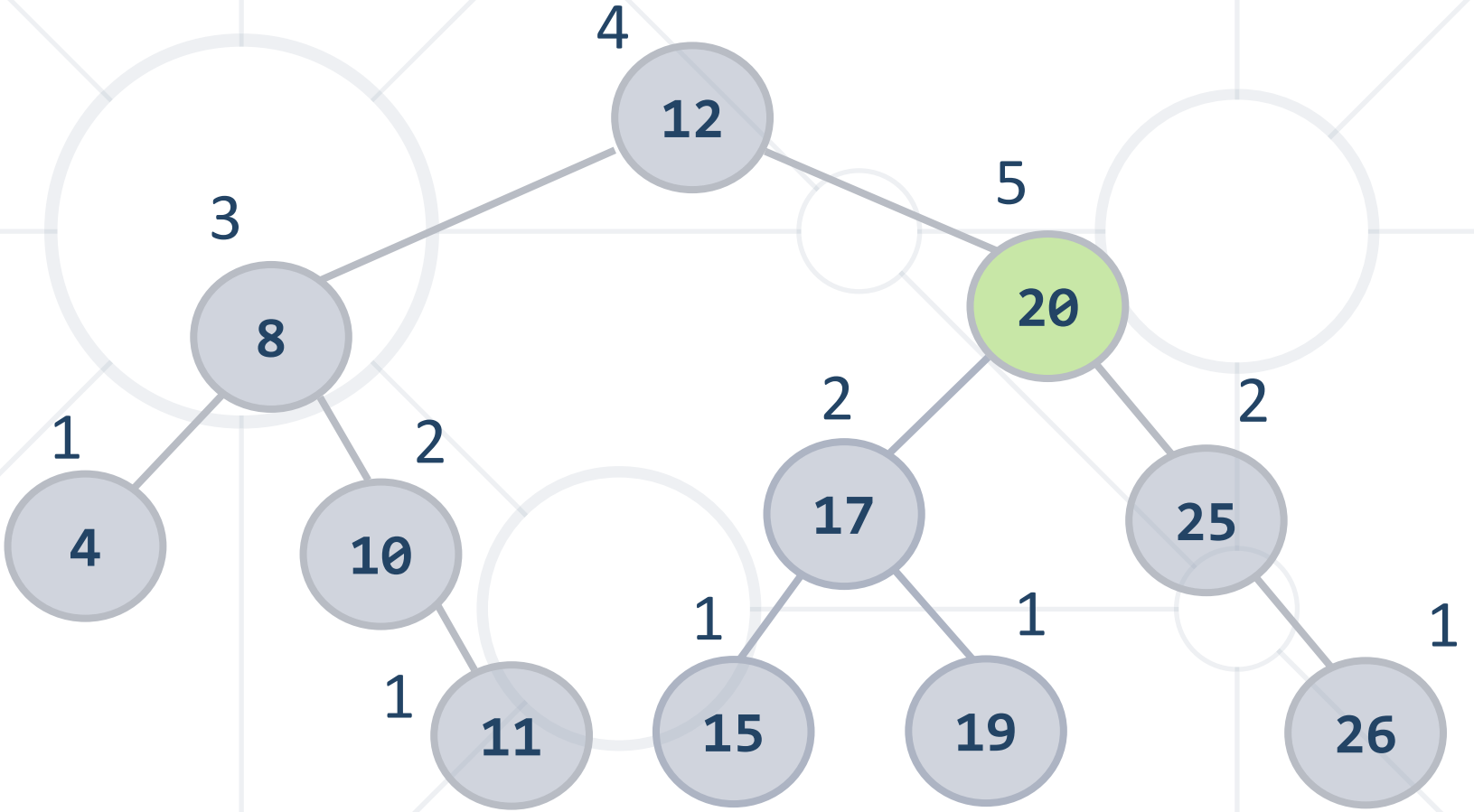
AVL Insertion

- Update 20 height
- 20 balance is 2
- 20 is left heavy
- Rotate 20 right



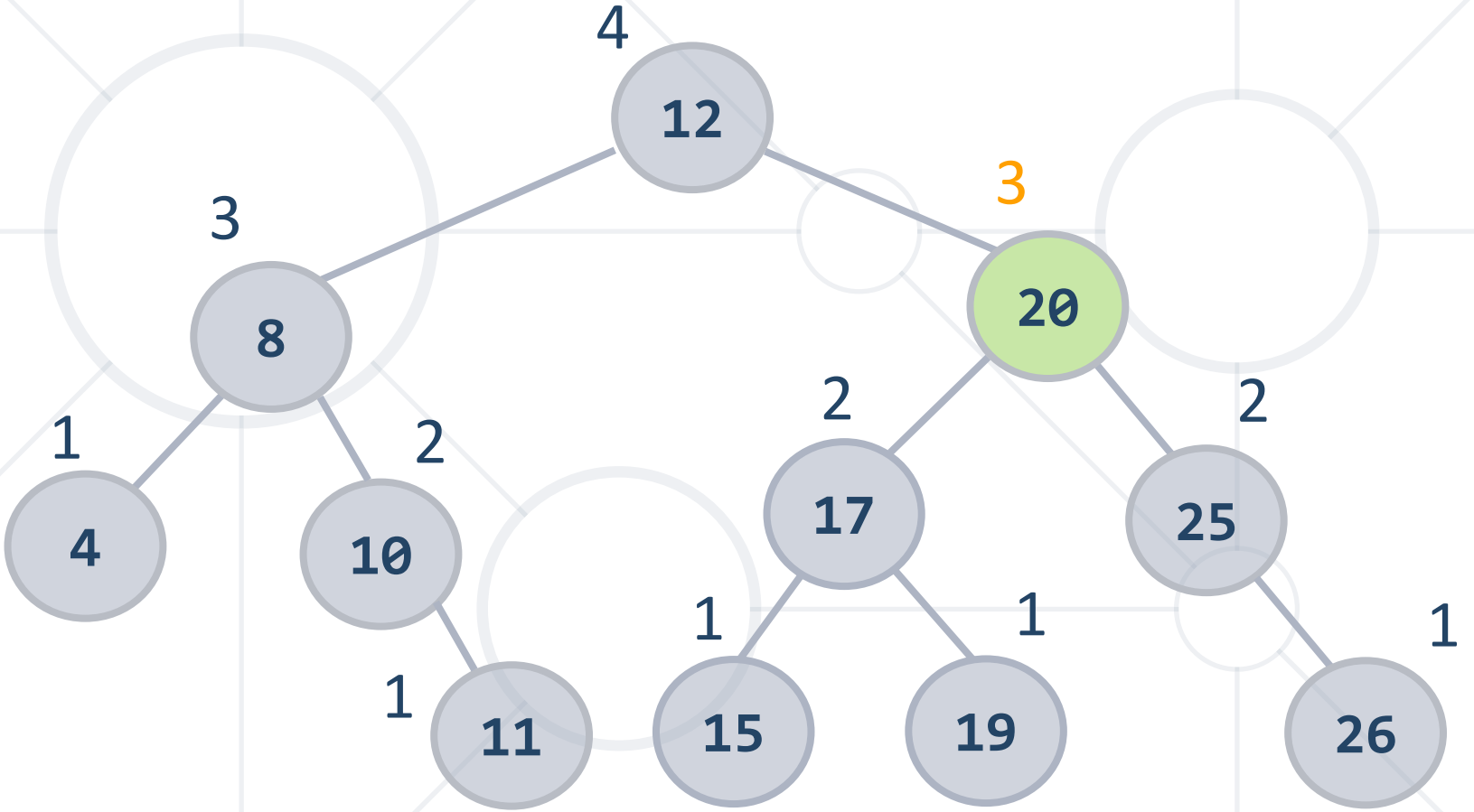
AVL Insertion

- 17 Switch parent



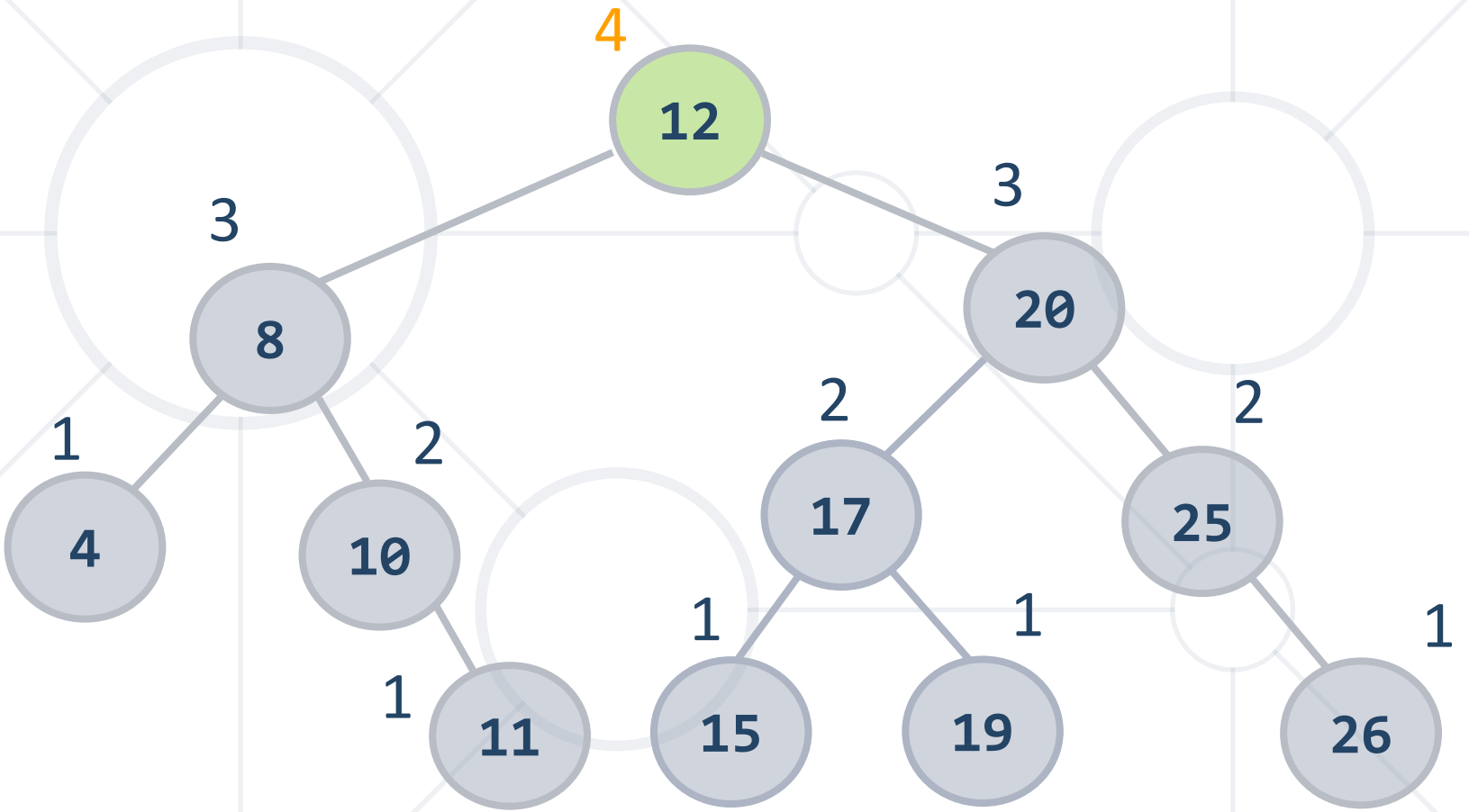
AVL Insertion

- Update 20 height

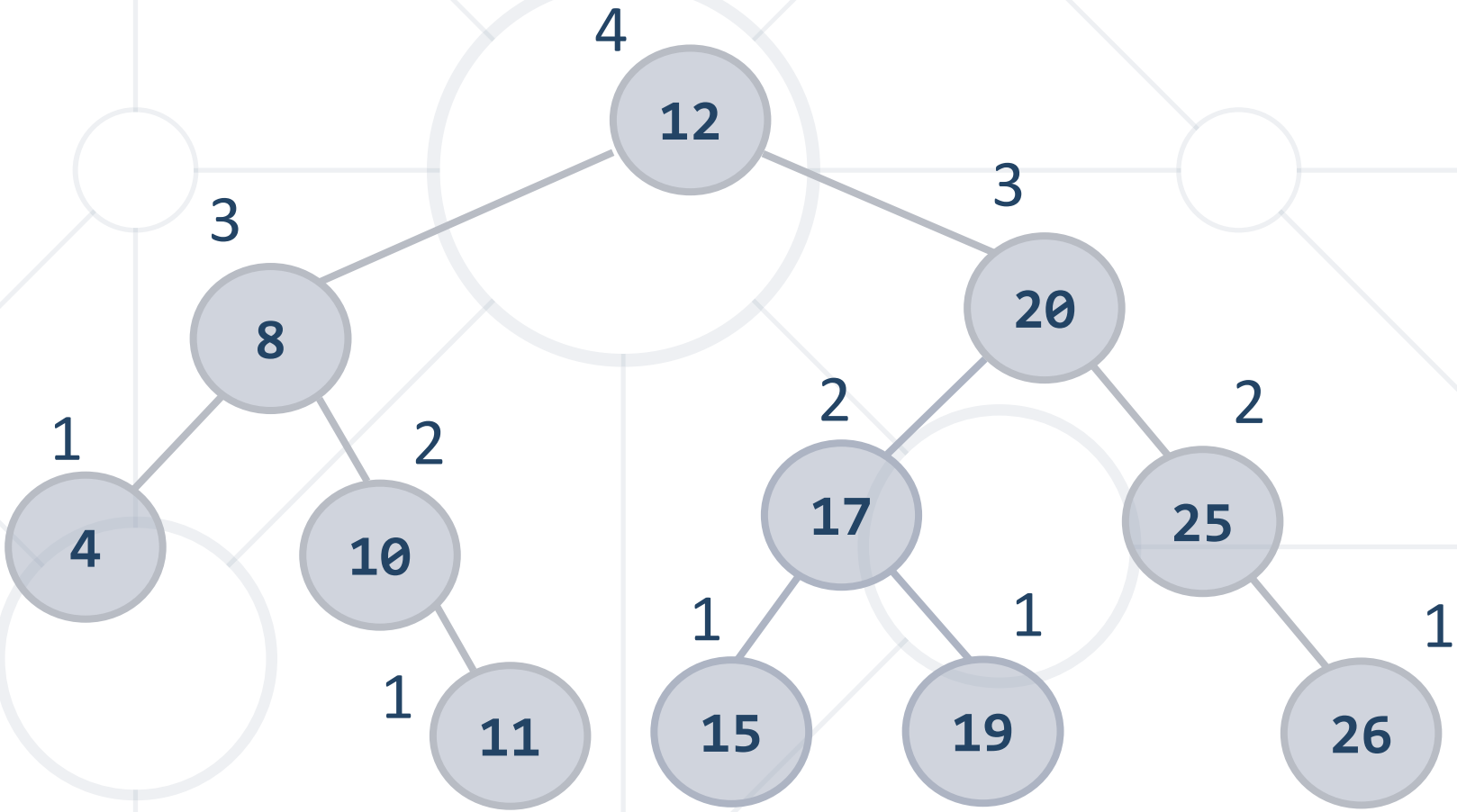


AVL Insertion

- 12 Update height
- 12 Balance is **0**



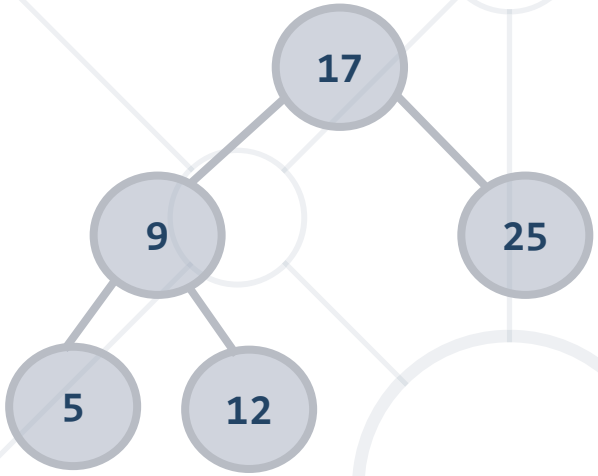
AVL Insertion



AVL Tree - Quiz

TIME'S

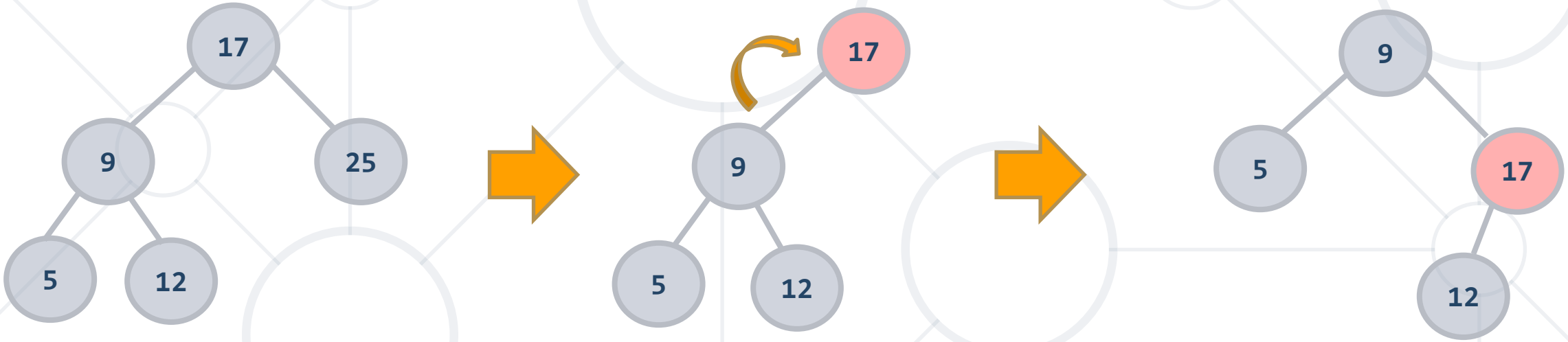
- Delete **25**. What will be the **resulting tree**?

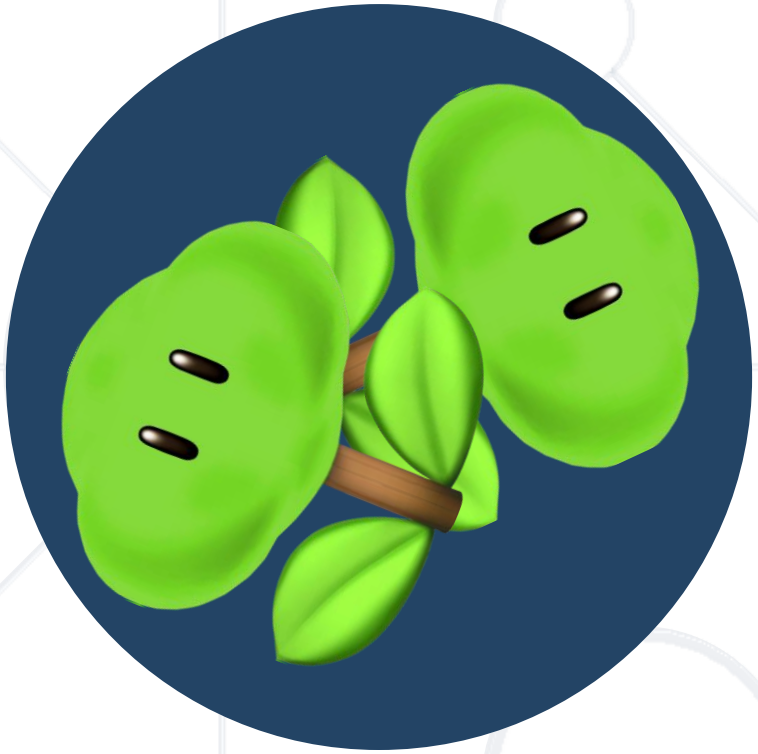


AVL Tree - Quiz

TIME'S UP!

- Delete **25**. What will be the **resulting tree**?



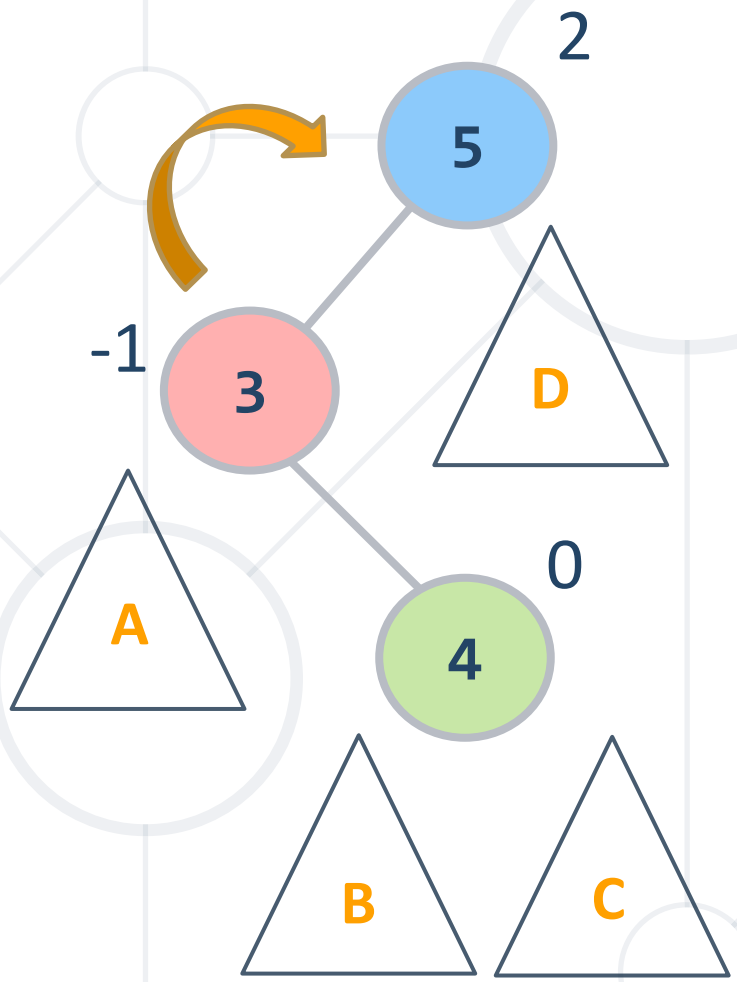


Double Rotations

Double Left, Double Right Rotation

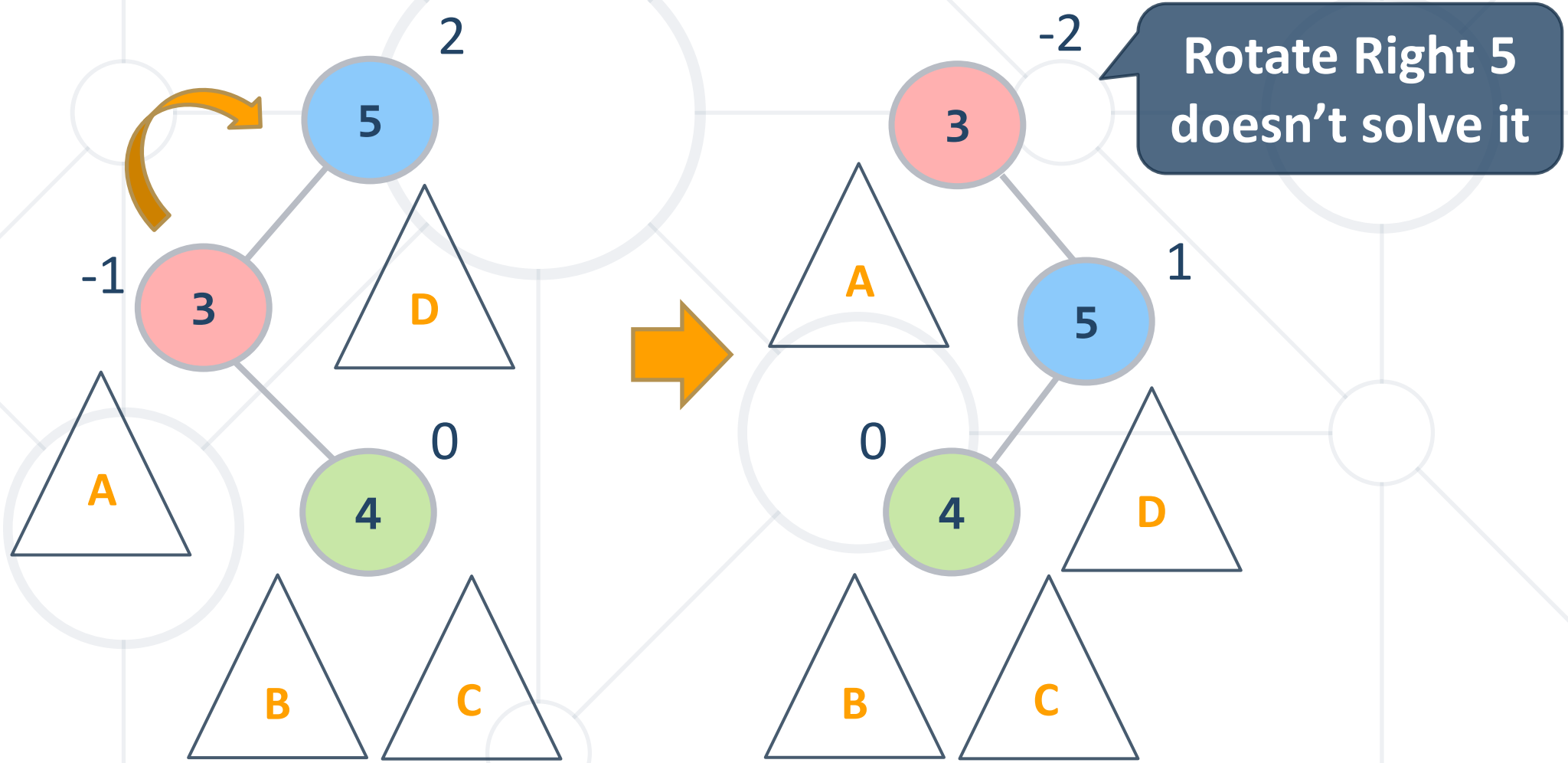
Single Rotation Problem

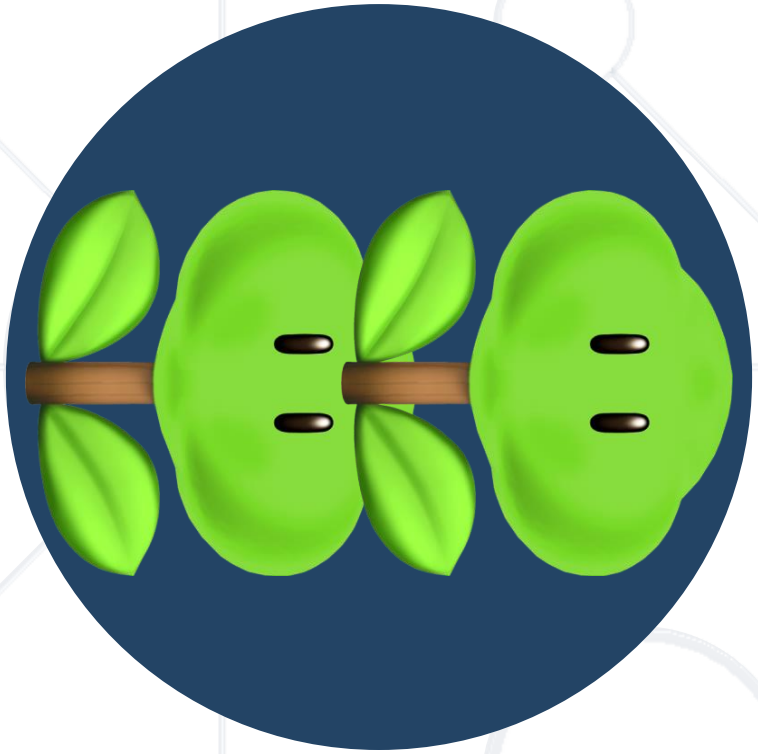
- Insert **4**



Single Rotation Problem (2)

- Rotate a node with opposite balanced child



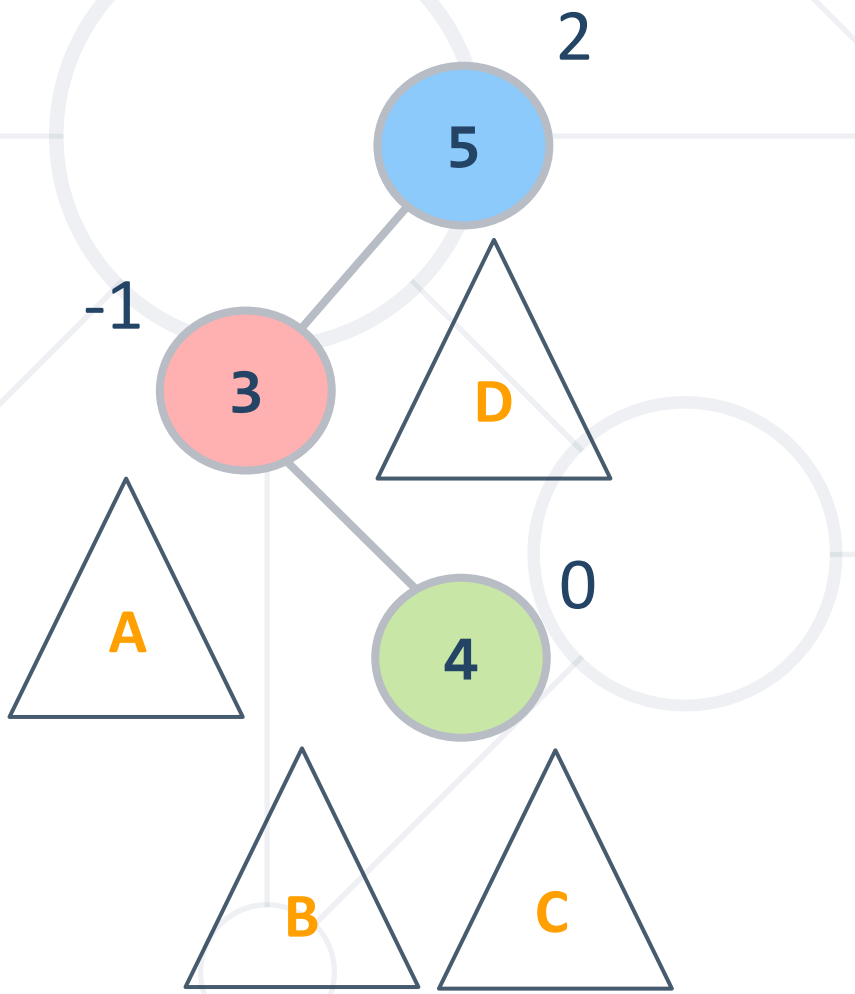


Double Right Rotation

Right-Left

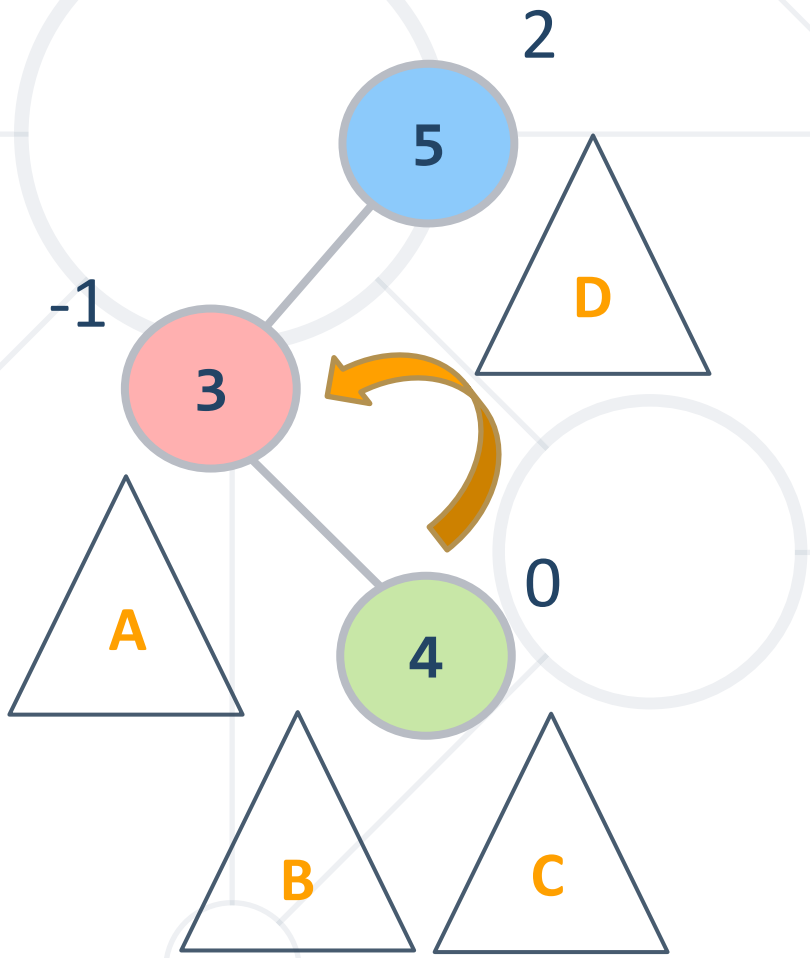
Double Right Rotation

- Rotate Right (node) with negatively balanced Left Child



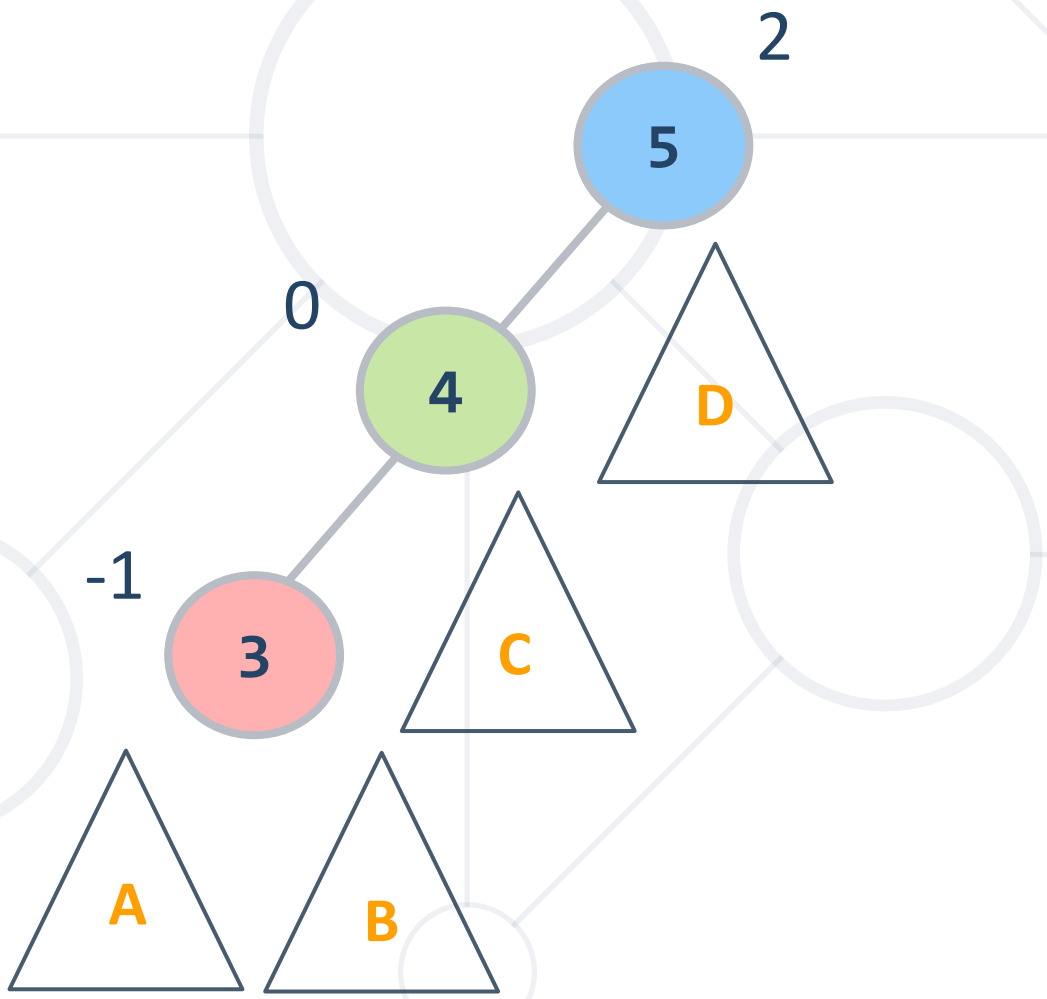
Double Right Rotation

- Left Rotate 3



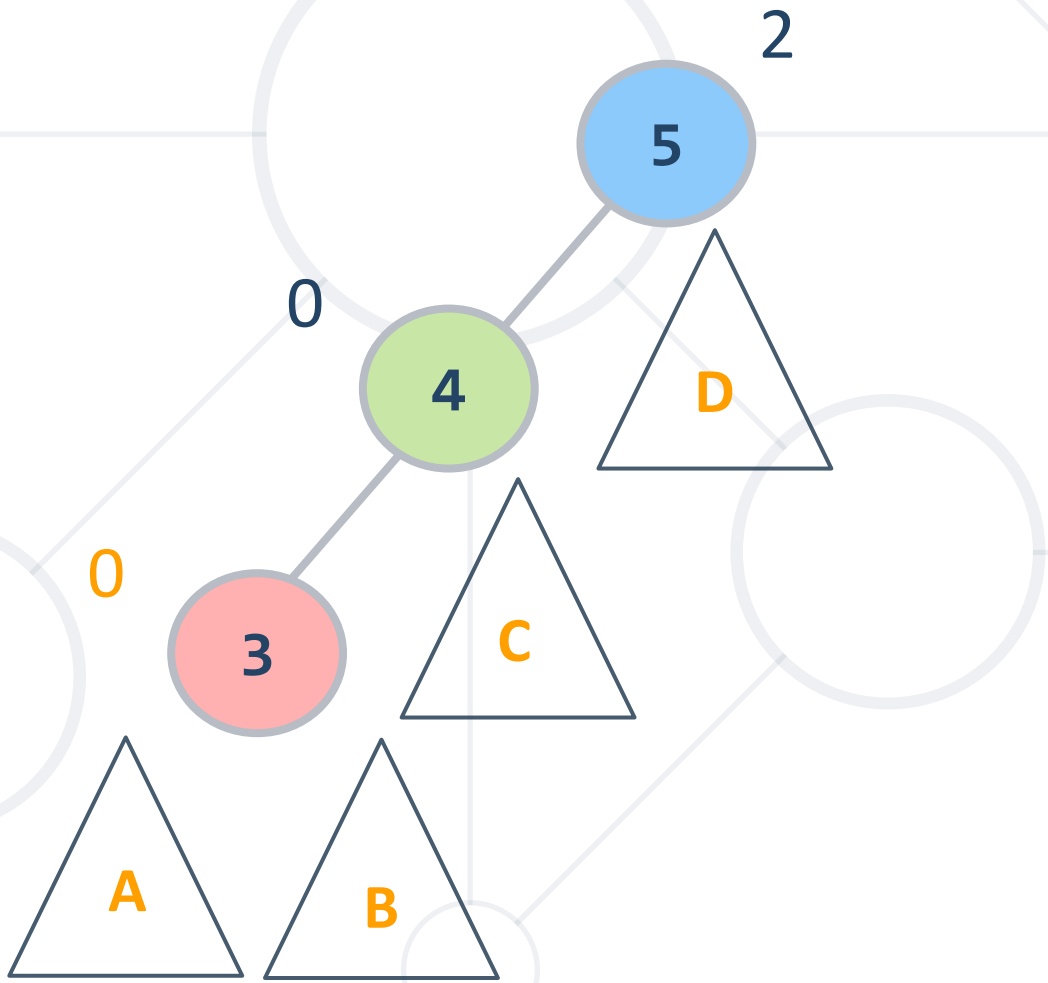
AVL Tree - Double Rotations

- Update Balance **3**



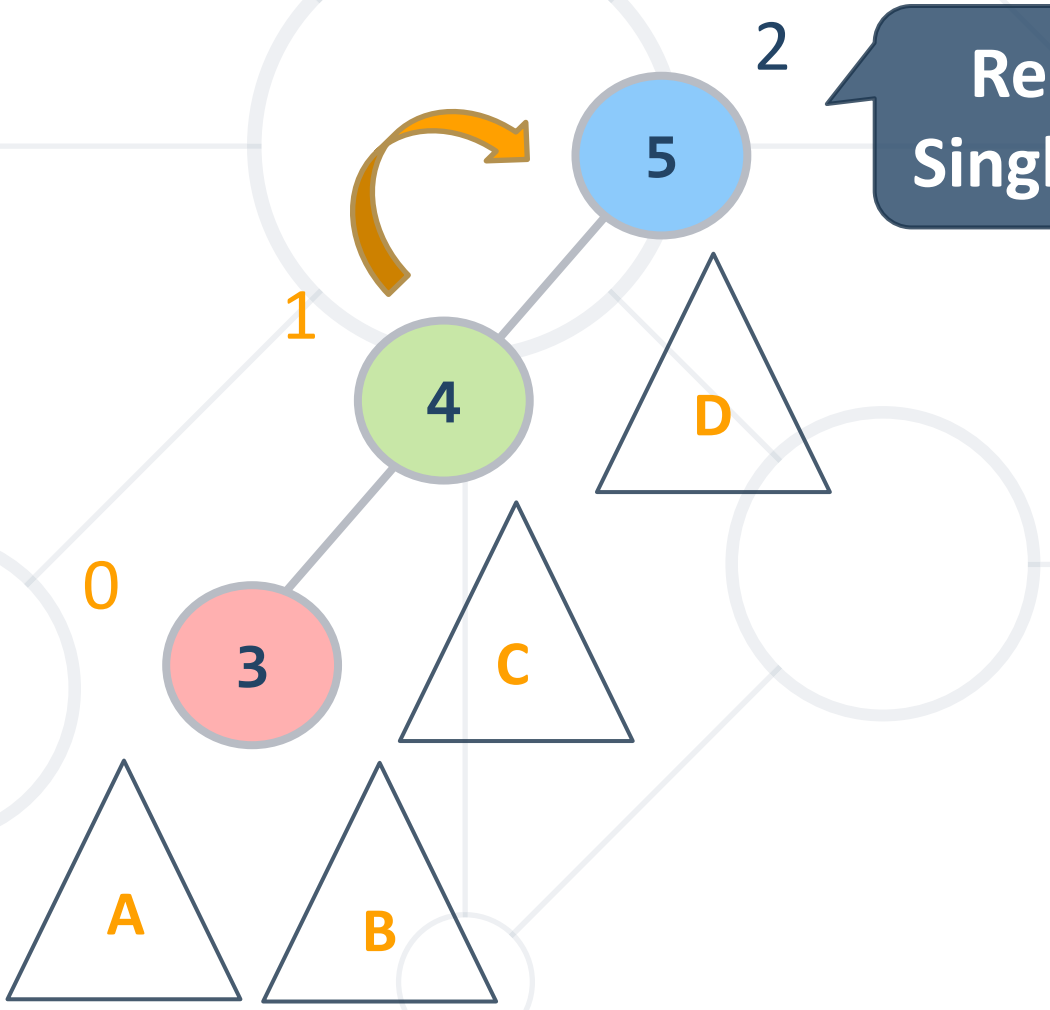
AVL Tree - Double Rotations

- Update Balance **3**



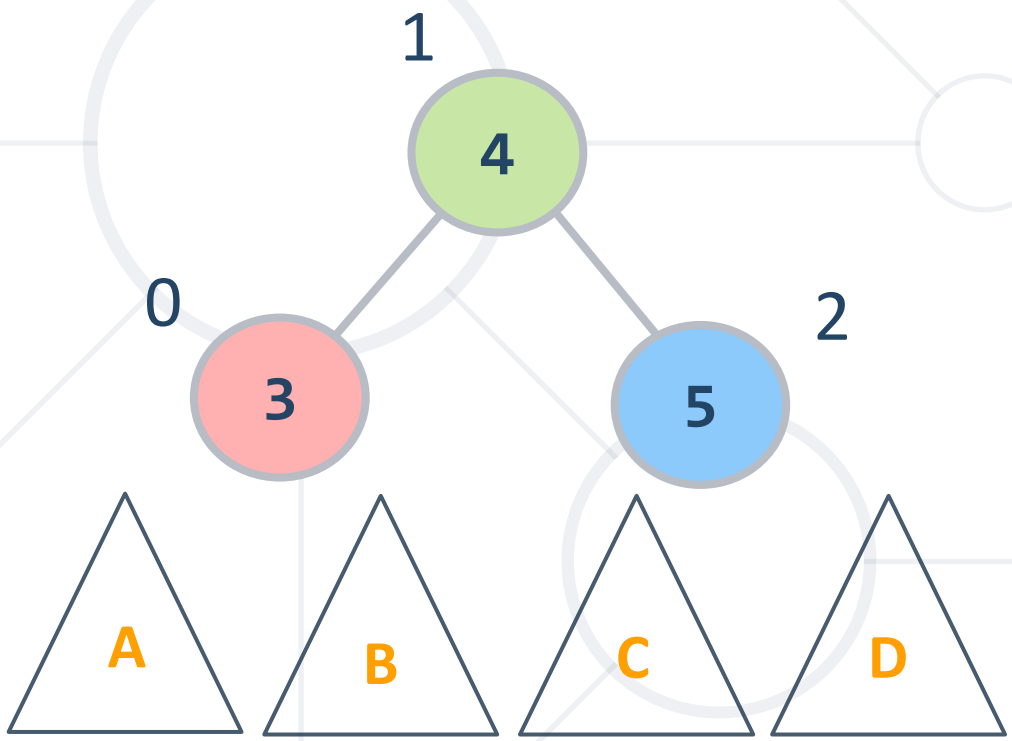
AVL Tree - Double Rotations

- Right Rotate **(5)**



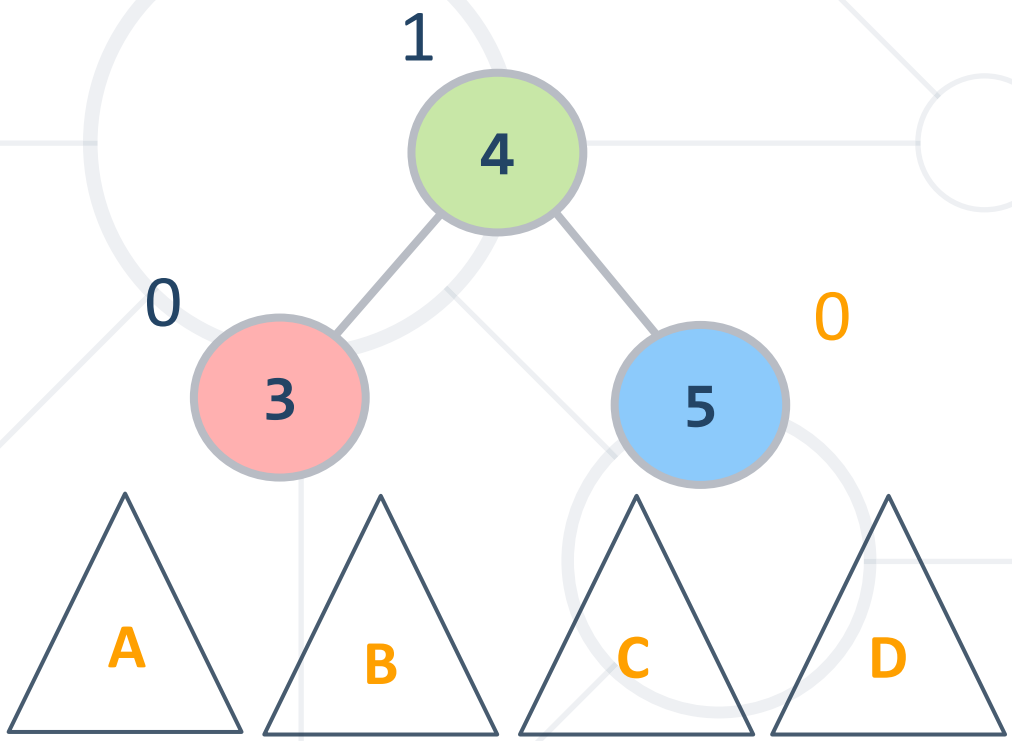
AVL Tree - Double Rotations

- Update Balance 5



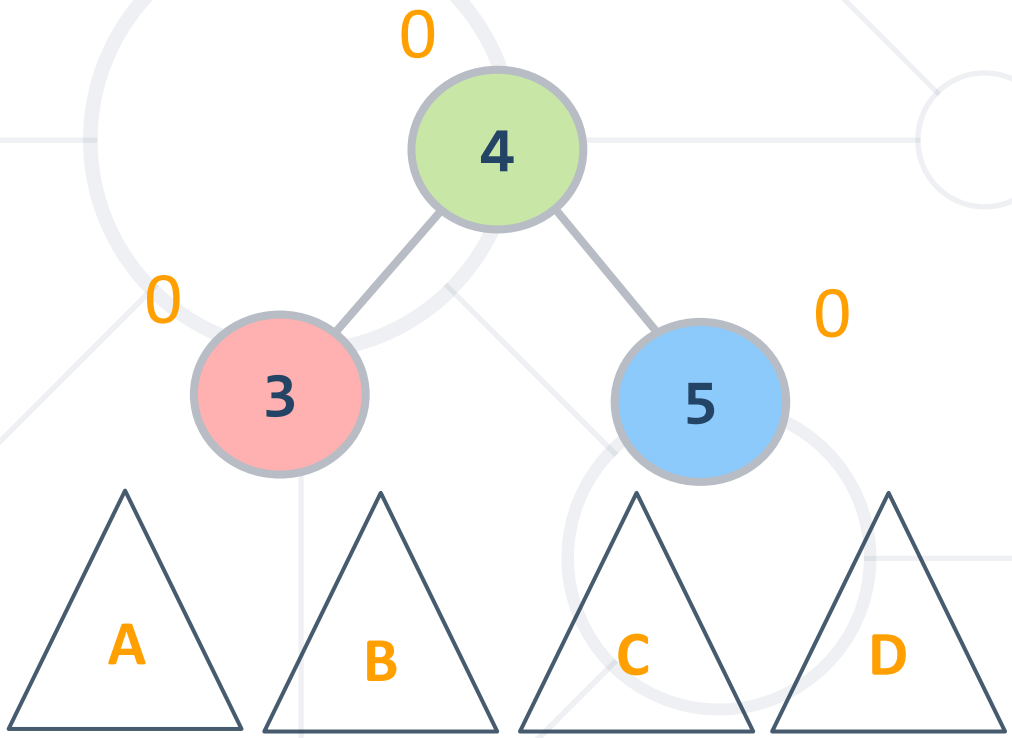
AVL Tree - Double Rotations

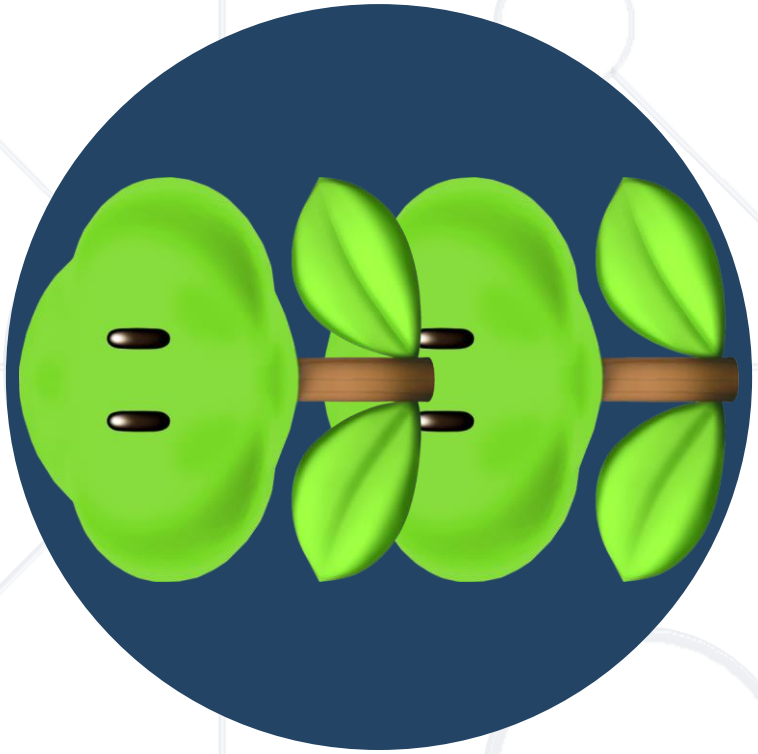
- Update Balance **4**



AVL Tree - Double Rotations

- Balance Restored!



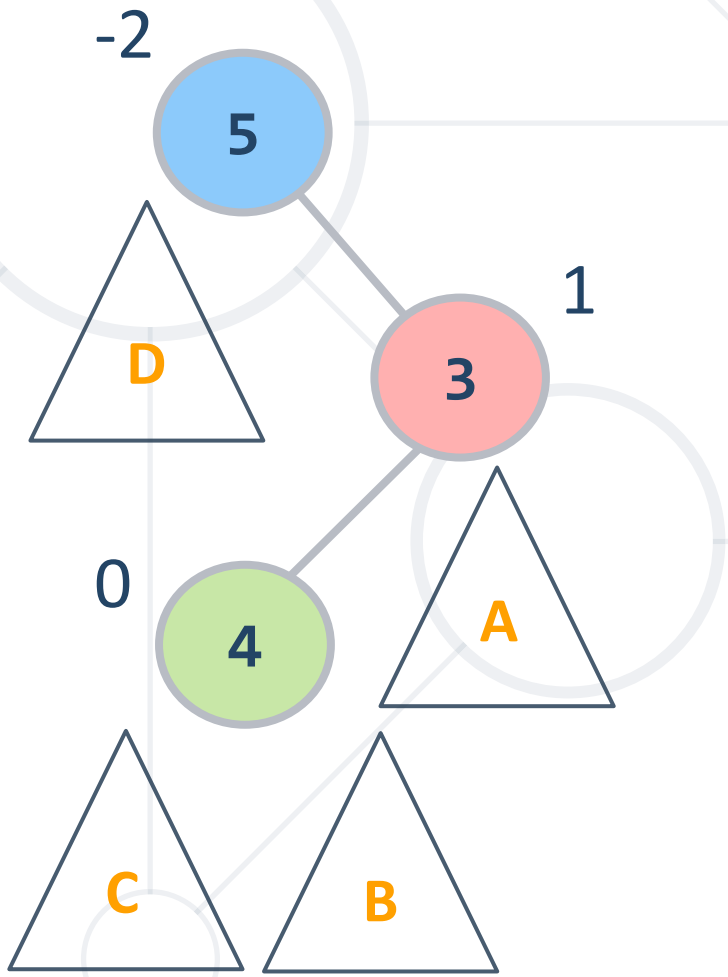


Double Left Rotation

Left-Right

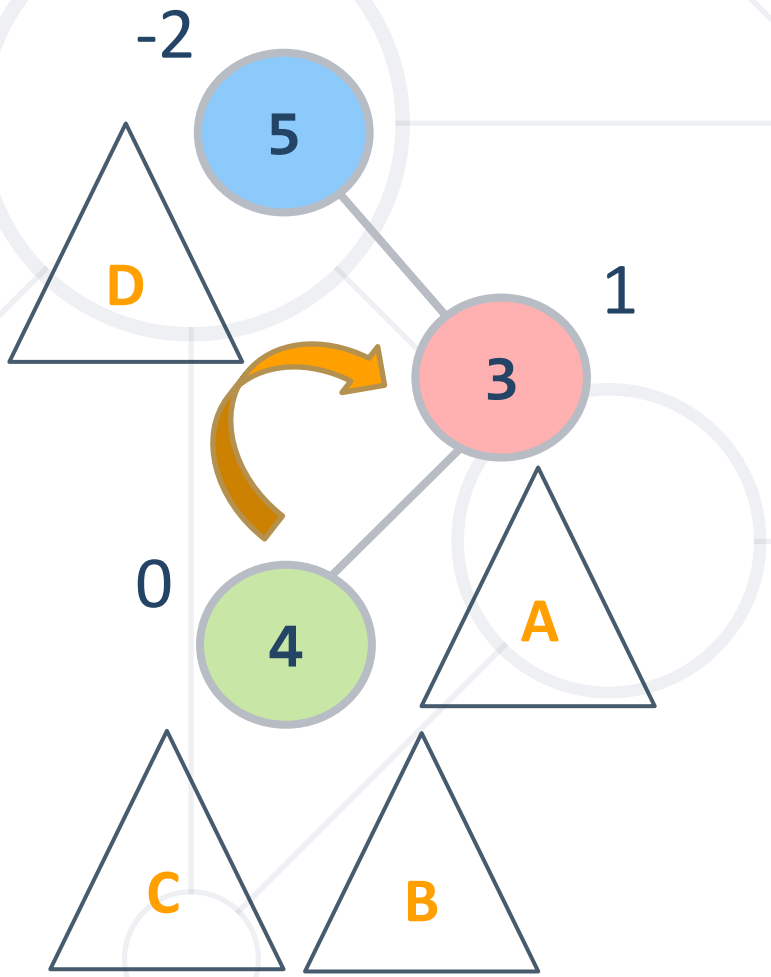
AVL Tree - Double Rotations

- Rotate **Left (node)** with positively balanced **Right Child**



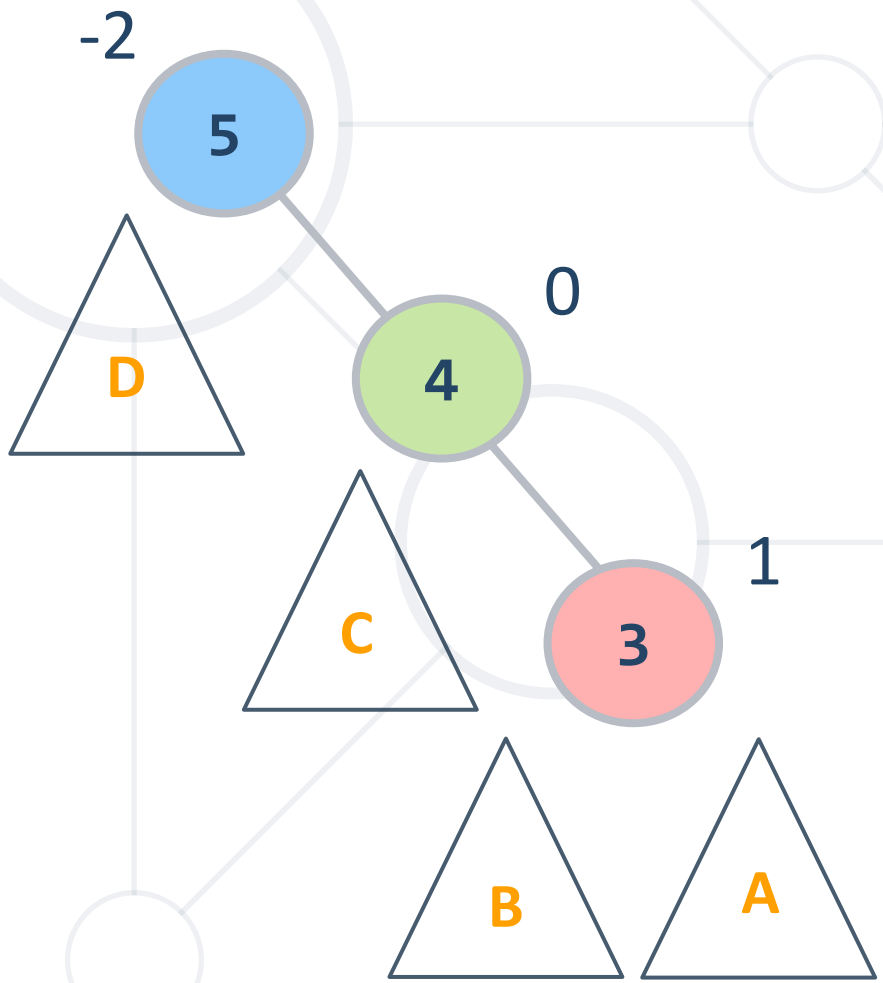
AVL Tree - Double Rotations

- Rotate Right **(3)**



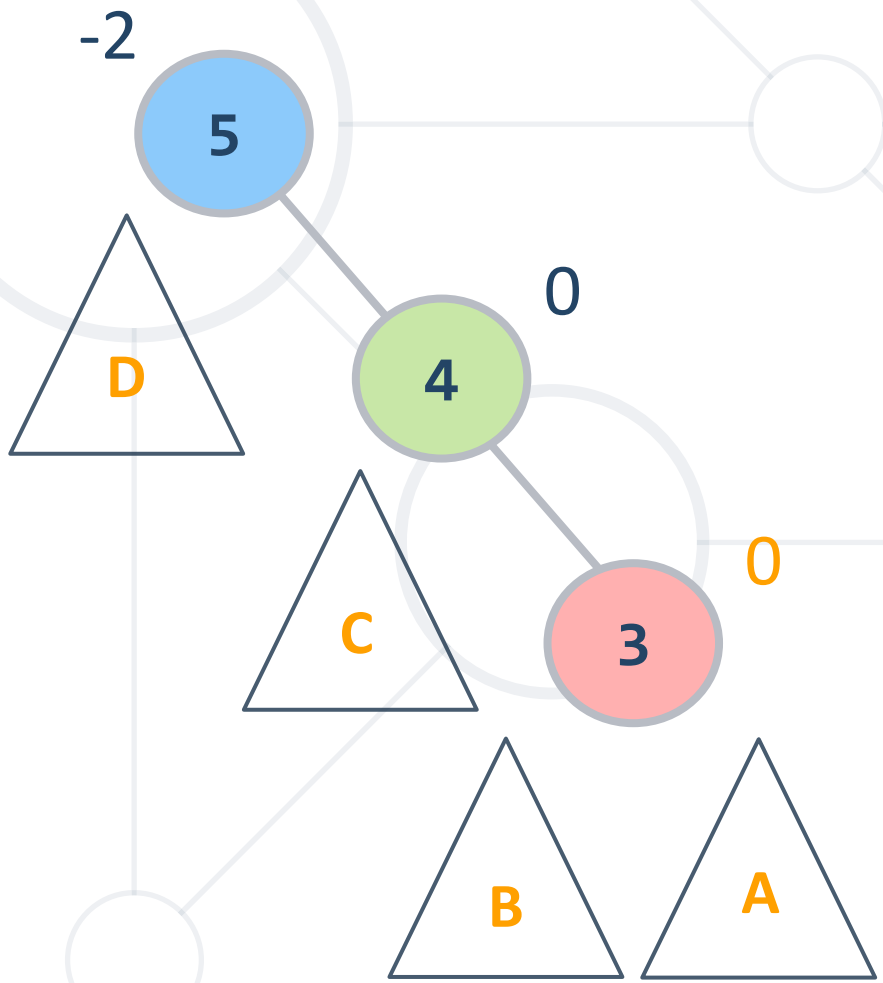
AVL Tree - Double Rotations

- Update Balance **(3)**



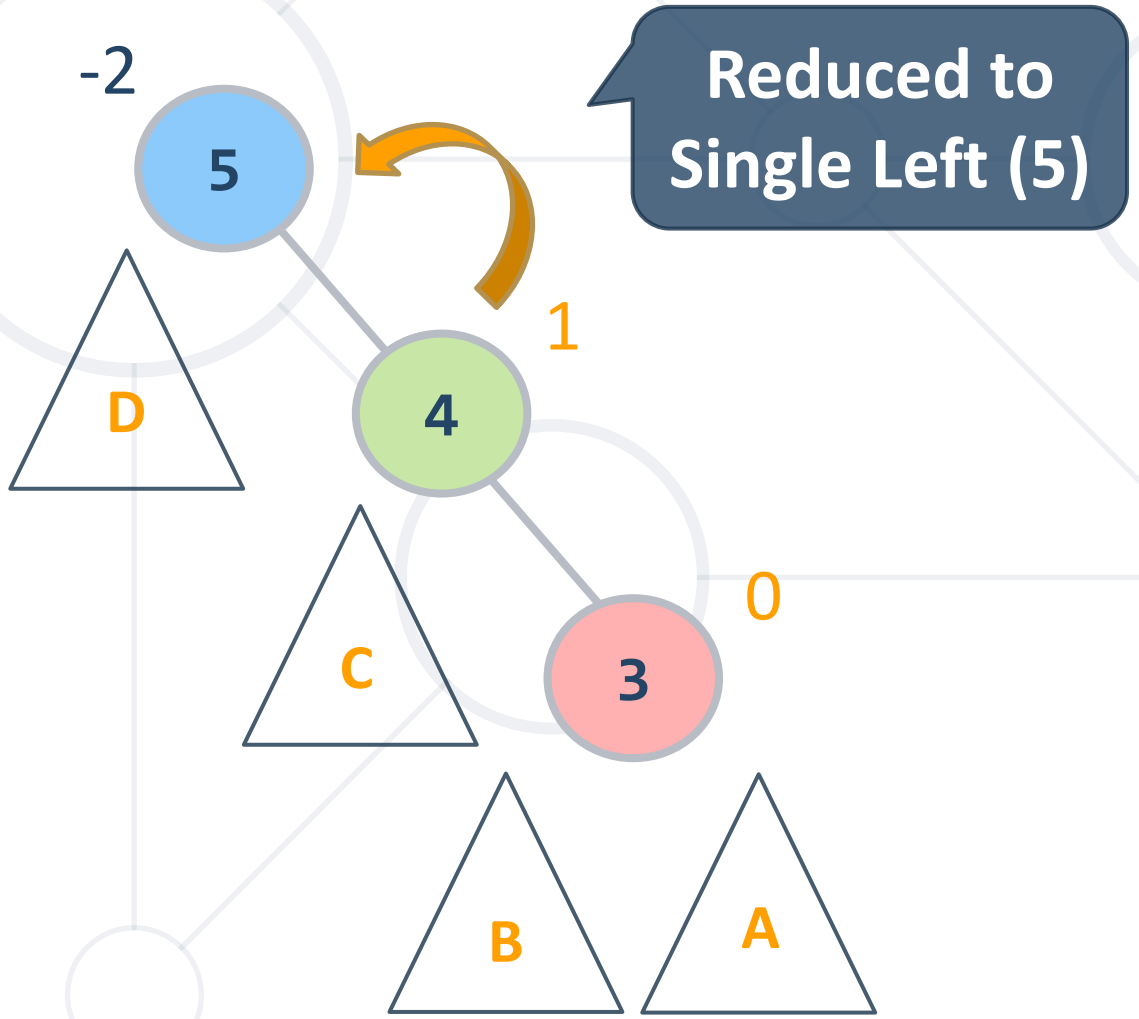
AVL Tree - Double Rotations

- Update Balance **(3)**



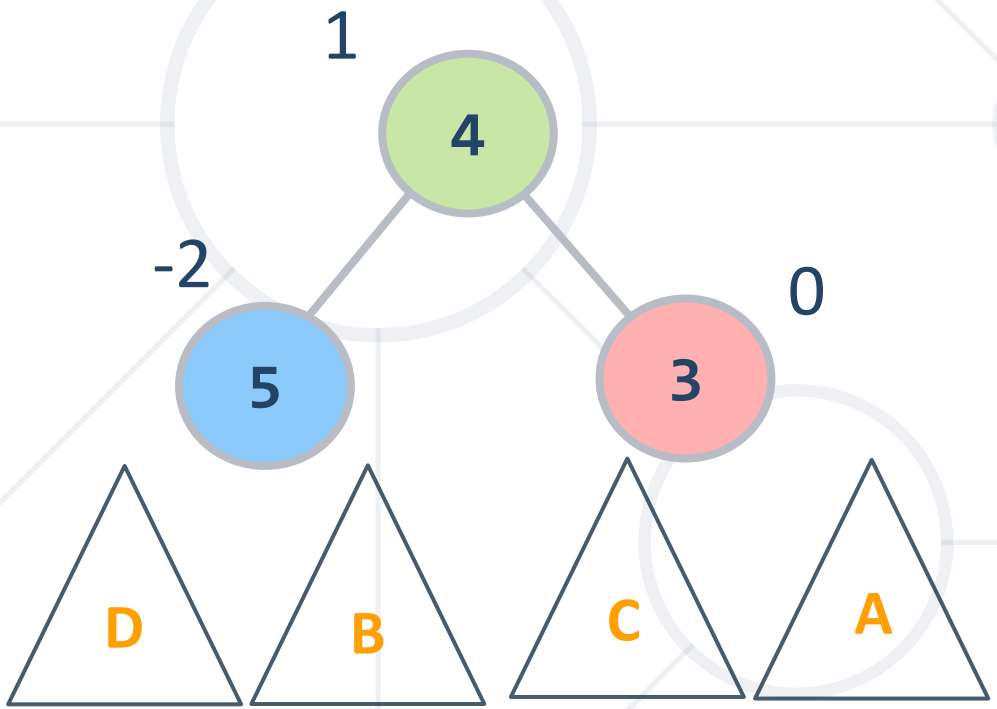
AVL Tree - Double Rotations

- Rotate Left **(5)**



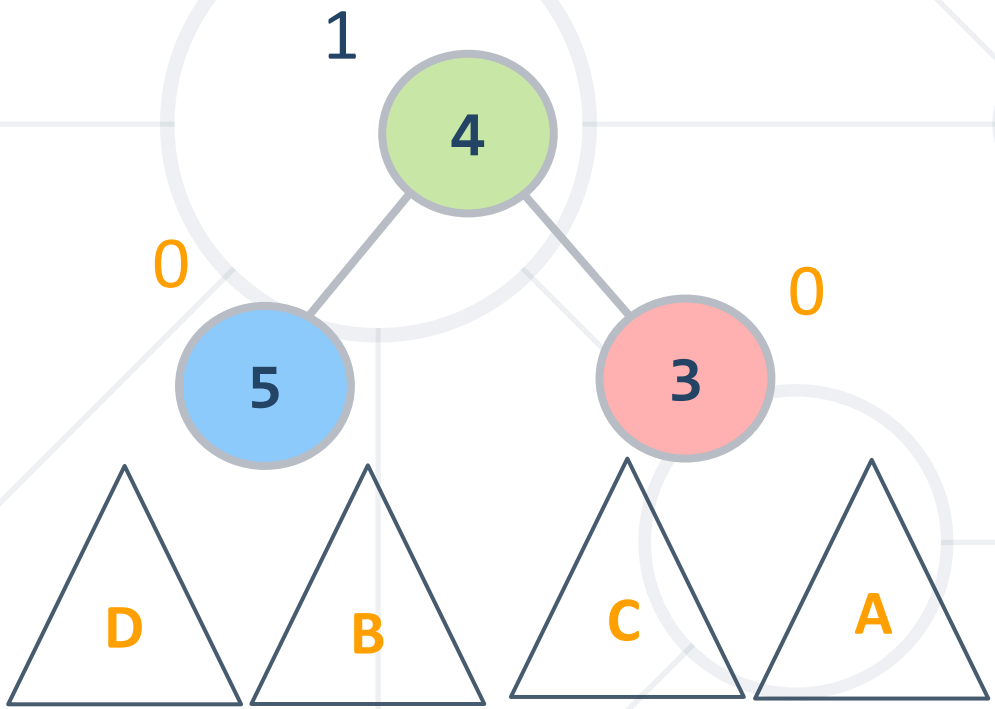
AVL Tree - Double Rotations

- Update Balance **(5)**



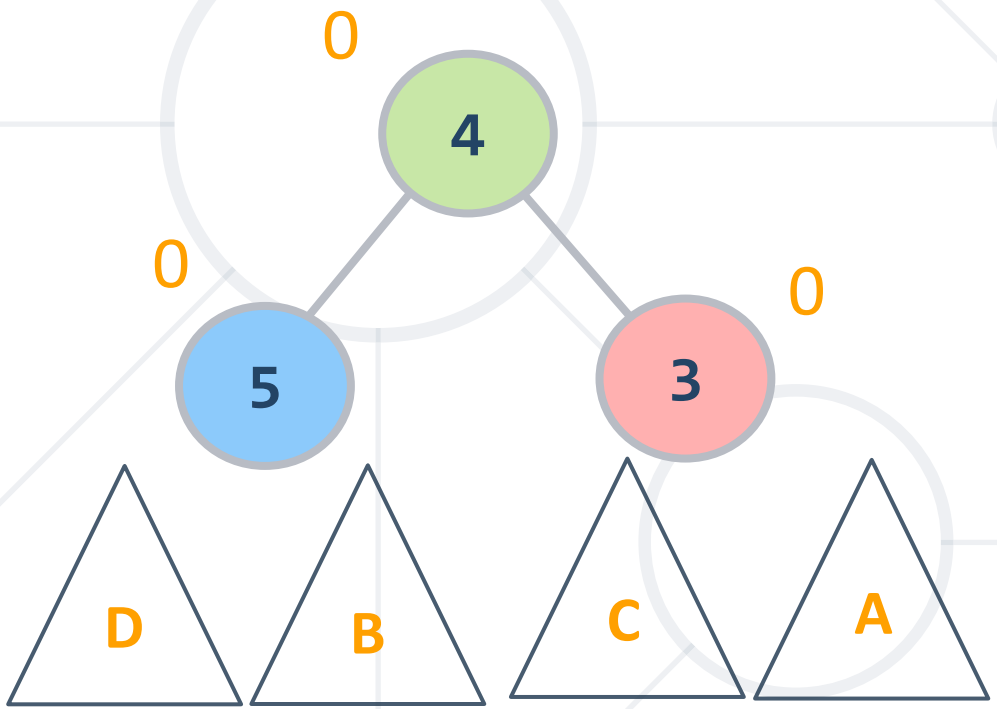
AVL Tree - Double Rotations

- Update Balance **(5)**



AVL Tree - Double Rotations

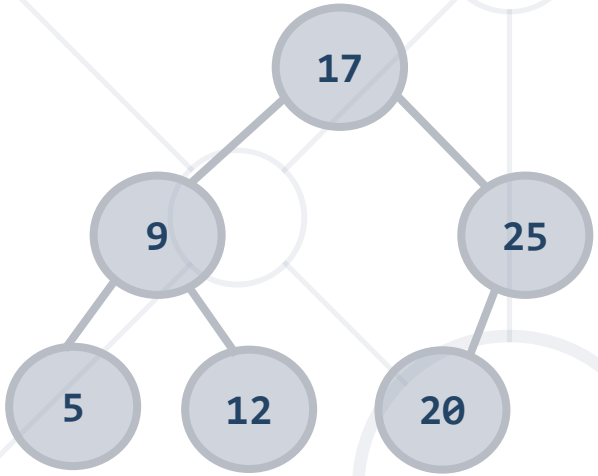
- Update Balance **(4)**



AVL Tree - Quiz

TIME'S

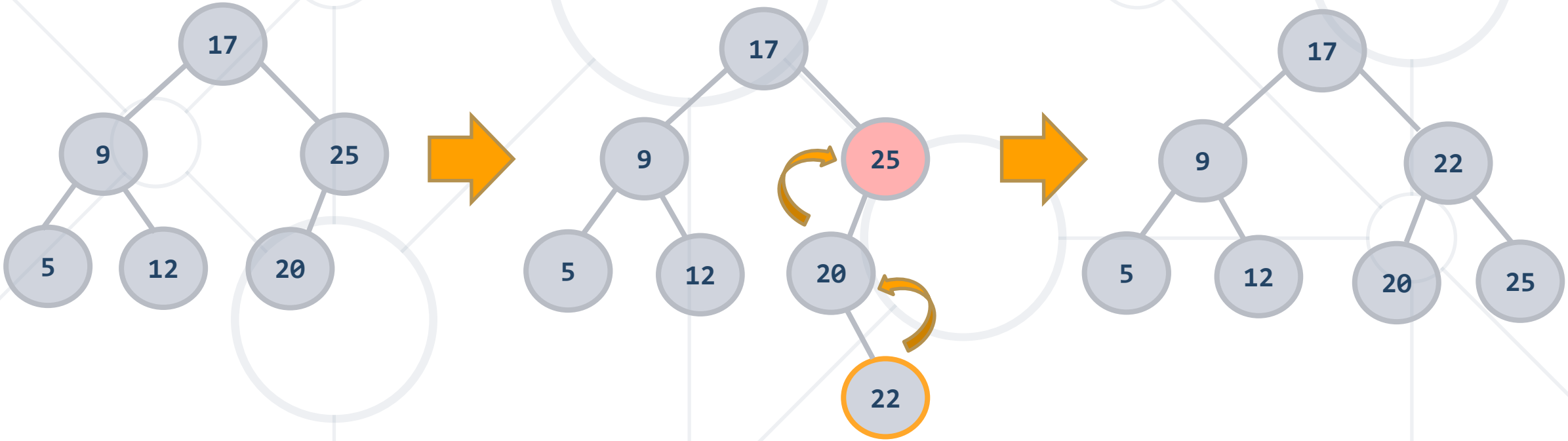
- Insert **22**. What will be the **resulting tree**?



AVL Tree - Quiz

TIME'S UP!

- Insert **22**. What will be the **resulting tree**?



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

AVL 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
AVL Tree	1.44 lg N	1.44 lg N	1.44 lg N	lg N	lg N

Summary

- AA Trees simplify are really like Red-Black trees, but have simplified Insert operation, without any major impact to performance
- AVL Trees are almost perfectly balanced
  - Good when searches are more frequent than insertions/deletions

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

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg