Data Structures Augmentation

Choosing a Data Structure

SoftUni Team

Technical Trainers

Software University

SoftUni

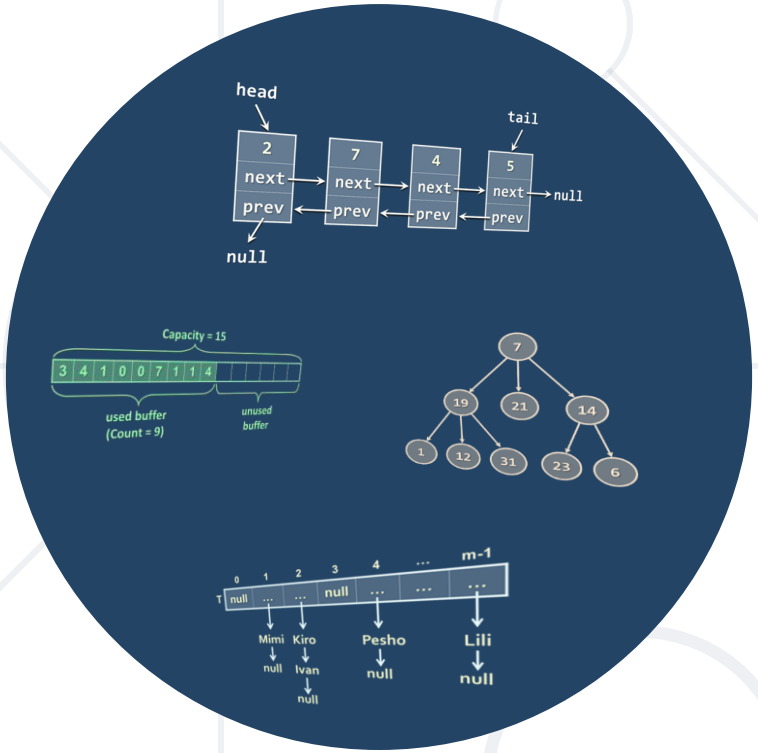
Software University

http://softuni.bg

Table of Contents

Software University

1. Classical Collection Data Structures - Summary
   - Linear Data Structures
   - Balanced Binary Search Trees
   - Hash Tables
2. Choosing a Collection Data Structure
3. Data Structures Augmentation



Choosing the Right DS

Lists vs. Hash Tables vs. Balanced Trees

Choosing a Collection - Array

- Array (T[])
  - Use when **fixed number of elements** need processing **by index**
    - No resize → for fixed number of elements only
    - **Add** / **delete** needs creating a new array + move **O(n)** elements
    - Compact and lightweight

Data Structure	Add	Find	Delete	Get-by-index
Static array: T[ ]	O(n)	O(n)	O(n)	O(1)

Choosing a Collection – Array Based List

- Resizable array-based list (**List<T>**)
  - Use when elements should be **added fast** and processed **by index**
  - Add (append to the end) has **O(1)** amortized complexity
  - The most-often used collection in programming

Data Structure	Add	Find	Delete	Get-by-index
Auto-resizable array-based list: **List<T>**	O(1)	O(n)	O(n)	O(1)

Choosing a Collection – Linked List

- Doubly-linked list (**LinkedList<T>**)
  - Use when elements should be **added at the both sides** of the list
  - Use when you need to **remove by a node reference**
  - Otherwise use resizable array-based list (**List<T>**)

Data Structure	Add	Find	Delete	Get-by-index
Double-linked list: `LinkedList<T>`	O(1)	O(n)	O(n)	O(n)

Choosing a Collection – Stack

- Stack (**Stack<T>**)
  - Use to implement **LIFO** (last-in-first-out) behavior
  - **List<T>** could also work well

Data Structure	Add	Find	Delete	Get-by-index
Stack: **Stack<T>**	O(1)	-	O(1)	-

Choosing a Collection – Queue

- Queue (**Queue<T>**)
  - Use to implement **FIFO** (first-in-first-out) behavior
  - **LinkedList<T>** could also work well

Data Structure	Add	Find	Delete	Get-by-index
Queue: **Queue<T>**	O(1)	-	O(1)	-

Choosing a Collection – Map

Software University

- Hash-table-based map (**Map<K, V>**)
  - Fast **add key-value pairs** + fast **search by key** – O(1)
  - Keys have **no particular order**
  - Keys should implement **hashCode(…)** and **equals(…)**

Data Structure	Add	Find	Delete	Get-by-index
Hash-table: `Map<K, V>`	O(1)	O(1)	O(1)	-

Choosing a Collection – Tree Map

- Balanced tree-based map (**TreeMap<K, V>**)
  - Elements are **ordered** by key
  - Fast **add key-value pairs** + fast **search by key** + fast **sub-range**
  - Keys should be **Comparable<K>**
  - Balanced trees → slower than hash-tables: **O(log n)** vs. **O(1)**

Data Structure	Add	Find	Delete	Get-by-index
Balanced tree-based dictionary: `TreeMap<K, V>`	`O(log n)`	`O(log n)`	`O(log n)`	-

Choosing a Collection – Multi Map

- Hash-table-based multi-dictionary (**MultiMap<K, V>**)
  - Fast **add key-value** + fast **search by key** + **multiple values** by key
  - Add by existing key **appends a new value** for the same key
  - Keys have **no particular order**

Data Structure	Add	Find	Delete	Get-by-index
Hash-table-based multi-dictionary: **MultiMap<K, V>**	O(1)	O(1)	O(1)	-

Choosing a Collection – Tree Multi Map

- Tree-based multi-dictionary (**TreeMultiMap<K, V>**)
  - Keys are **ordered** by key
  - Fast **add key-value** + fast **search by key** + fast **sub-range**
  - Add by existing key appends a new value for the same key

Data Structure	Add	Find	Delete	Get-by-index
Tree-based multi-dictionary: **TreeMultiMap<K, V>**	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Hash Set

- Hash-table-based set (**HashSet<T>**)
  - **Unique** values + fast **add** + fast **contains**
  - Elements have **no particular order**
  - Elements should implement **hashCode(…)** and **equals(…)**

Data Structure	Add	Find	Delete	Get-by-index
Hash-table-based set: **HashSet<T>**	O(1)	O(1)	O(1)	-

Choosing a Collection – Tree Set

- Balanced tree-based set (**TreeSet<T>**)
  - **Unique** values + **sorted order**
  - Fast **add** + fast **contains** + fast **sub-range**
  - Elements should be **Comparable<T>**

Data Structure	Add	Find	Delete	Get-by-index
Balanced tree-based set: **TreeSet<T>**	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Hash Bag

- Hash-table-based bag (**Bag<T>**)
  - Bags allow **duplicates**
  - Fast **add** + fast **find** + fast **contains**
  - Elements have **no particular order**

Data Structure	Add	Find	Delete	Get-by-index
Hash-table-based bag: **Bag<T>**	O(1)	O(1)	O(1)	-

Choosing a Collection – Tree Bag

- Balanced tree-based bag (**TreeBag<T>**)
  - Allow **duplicates**, **sorted order**
  - Fast **add** + fast **find** + fast **contains**
  - Access by **sorted index** + extract **sub-range**

Data Structure	Add	Find	Delete	Get-by-index
Balanced tree-based bag: **TreeBag<T>**	O(log n)	O(log n)	O(log n)	-

Choosing a Collection – Special DS

- Priority Queue (**Heap**) – **fast max**/**min** element
- **Rope** – fast **add**/**remove** by index
- **Prefix** tree (Trie) – fast **prefix search**
- **Suffix** tree – fast **suffix search**
- **Interval** tree – fast **interval search**
- **K-d** trees, **Quad** trees – fast **geometric distance search**

Data Structure Efficiency – Comparison

Software University

Data Structure	Add	Find	Delete	Get-by-index
Static array: `T[]`	O(n)	O(n)	O(n)	O(1)
Double-linked list: `LinkedList<T>`	O(1)	O(n)	O(n)	O(n)
Auto-resizable array-based list: `List<T>`	O(1)	O(n)	O(n)	O(1)
Stack: `Stack<T>`	O(1)	-	O(1)	-
Queue: `Queue<T>`	O(1)	-	O(1)	-

Data Structure Efficiency – Comparison (2)

Software University

Data Structure	Add	Find	Delete	Get-by-index
Hash-table: `Map<K, V>`	O(1)	O(1)	O(1)	-
Balanced tree-based dictionary: `TreeMap<K, V>`	O(log n)	O(log n)	O(log n)	-
Hash-table-based set: `HashSet<T>`	O(1)	O(1)	O(1)	-
Balanced tree-based set: `TreeSet<T>`	O(log n)	O(log n)	O(log n)	-

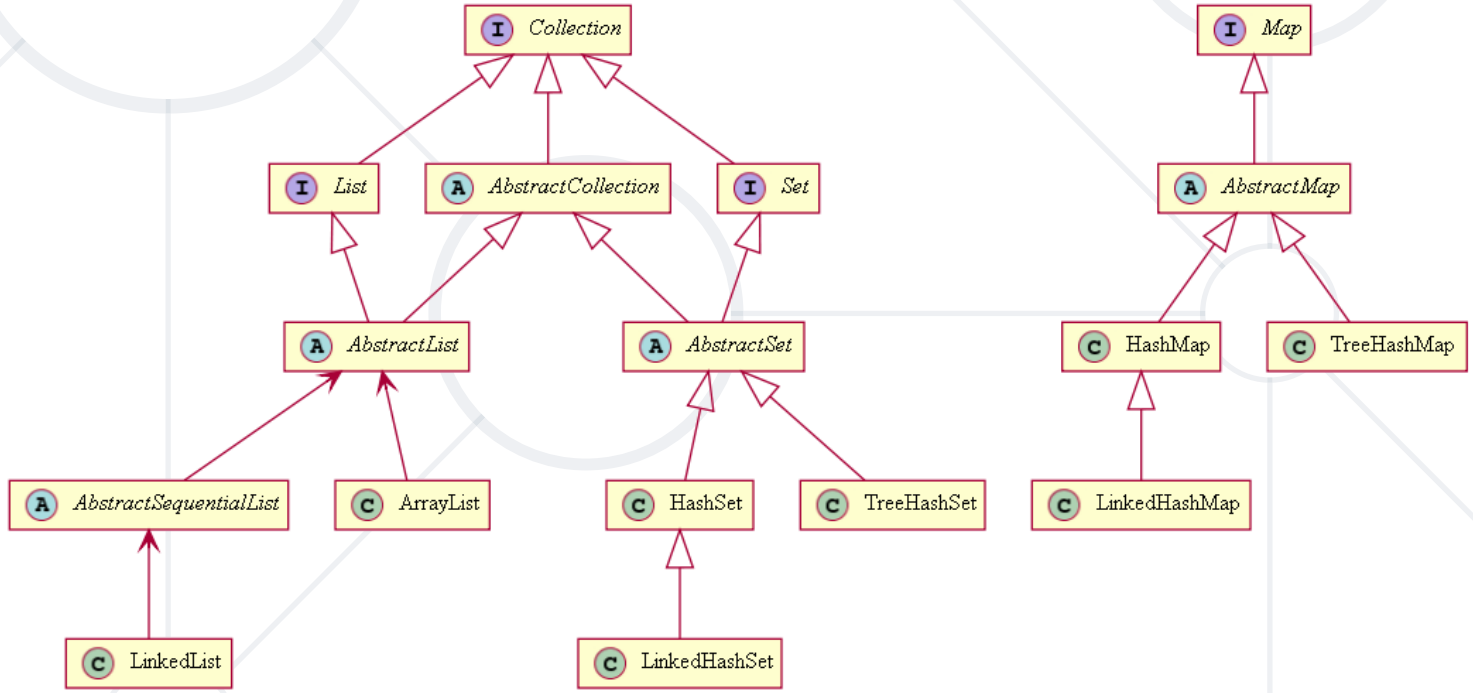
Data Structure Efficiency – Comparison (3)

Software University

Data Structure	Add	Find	Delete	Get-by-index
Hash-table-based multi-dictionary: `MultiMap<K, V>`	O(1)	O(1)	O(1)	-
Tree-based multi-dictionary: `TreeMap<K, V>`	O(log n)	O(log n)	O(log n)	-
Hash-table-based bag: `Bag<T>`	O(1)	O(1)	O(1)	-
Balanced tree-based bag: `TreeBag<T>`	O(log n)	O(log n)	O(log n)	-

Java – Collections/Guava APIs

- All commonly used collections
  - java.util.Collections
  - com.google.common.collect



Data Structures Augmentation

Combining Data Structures

Software University

- Many scenarios → combine several DS
  - No ideal DS → choose between space and time
- For example, we can combine:
  - A **hash-table** for fast **search by key_1** (e.g. name)
  - A **hash-table** for fast **search by {key_2 + key_3}** (e.g. name + town)
  - A **balanced search tree** for fast **extract-range(start_key … end_key)**
  - A **rope** for fast **access-by-index**
  - A **balanced search tree** for fast **access-by-sorted-index**

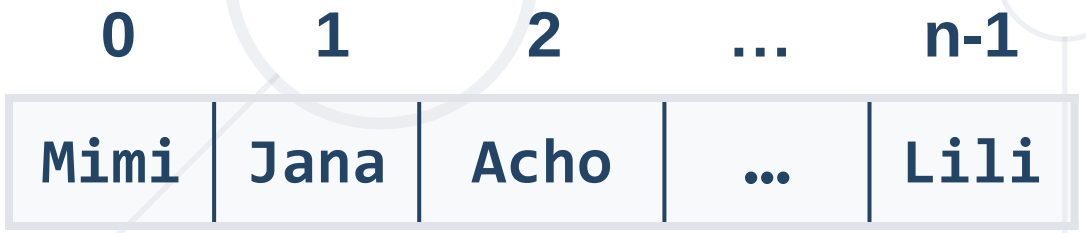
Problem: Collection of People

- Design a data structure that efficiently implements:

Operation	Return Type
add(email, name, age, town)	**boolean** – unique email
find(email)	**Person** or **null**
delete(email)	**boolean**
findAll(email_domain)	**Iterable<P>** – sorted by email
findAll(name, town)	**Iterable<P>** – sorted by email
findAll(start_age, end_age)	**Iterable<P>** – sorted by age, email
findAll(start_age, end_age, town)	**Iterable<P>** – sorted by age, email

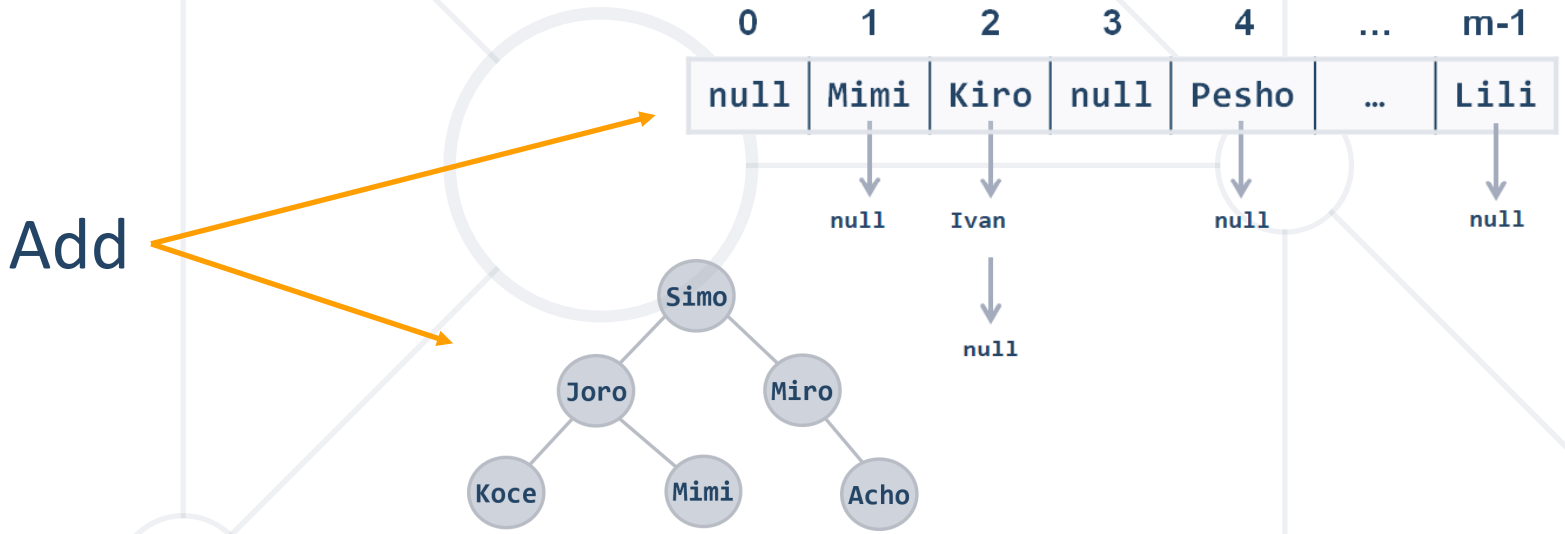
List Based Solution

- List based solution – **single list** for all operations
  - **Easy** to **implement**
  - Easy to achieve **correct behavior**
  - Useful for **creating unit tests**



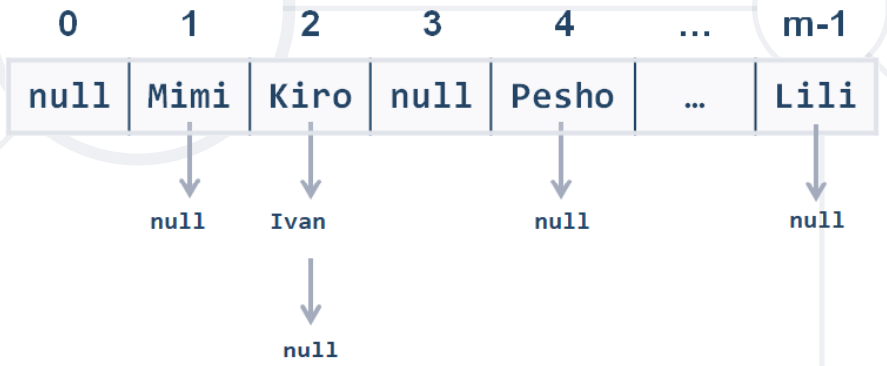
Solution: Add Person

- boolean **add**(email, name, age, town)
  - Create a **Person** object to hold { ***email + name + age + town*** }
  - Add the new person to **all** underlying data structures



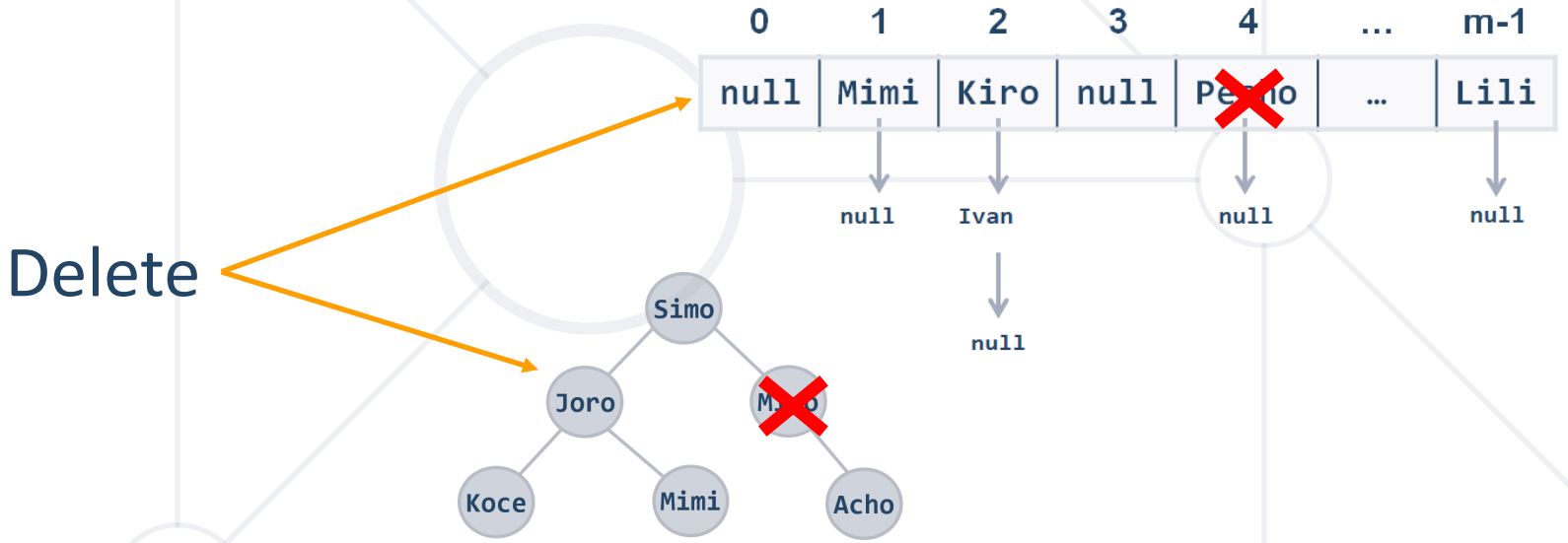
Solution: Find by Email

- Person **find(email)**
  - Use a **hash-table** to map { *email → person* }
  - Complexity – O(1)



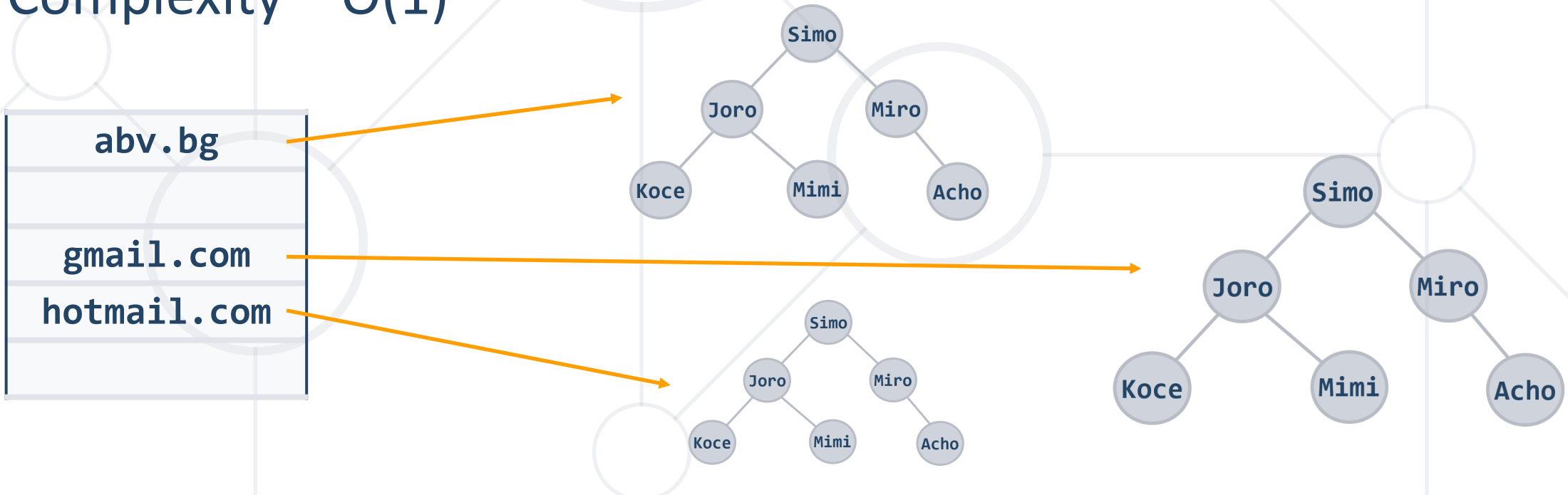
Solution: Delete

- bool **delete**(**email**)
  - Find the person by email in the underlying **hash-table**
  - Delete the person from **all** underlying data structures
  - Complexity – O(log n)



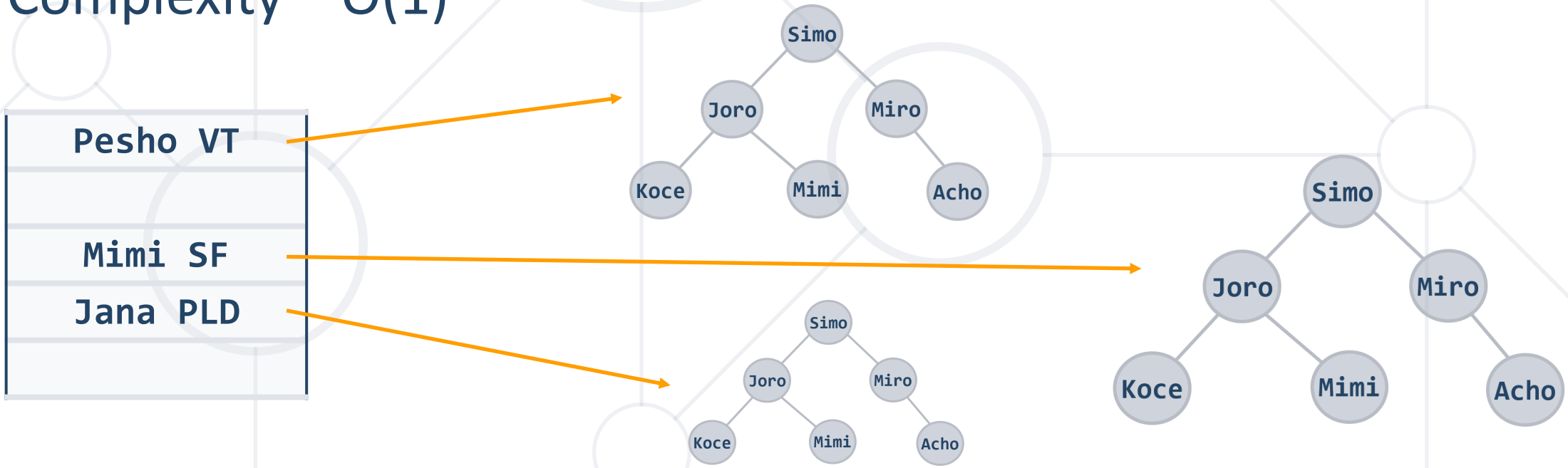
Solution: Find by Domain

- Iterator<Person> **findAll**(email_domain)
  - Use a **hash-table** to map {***email_domain → SortedSet<Person>***}
  - Get email_domain by the email when adding persons
  - Complexity – O(1)



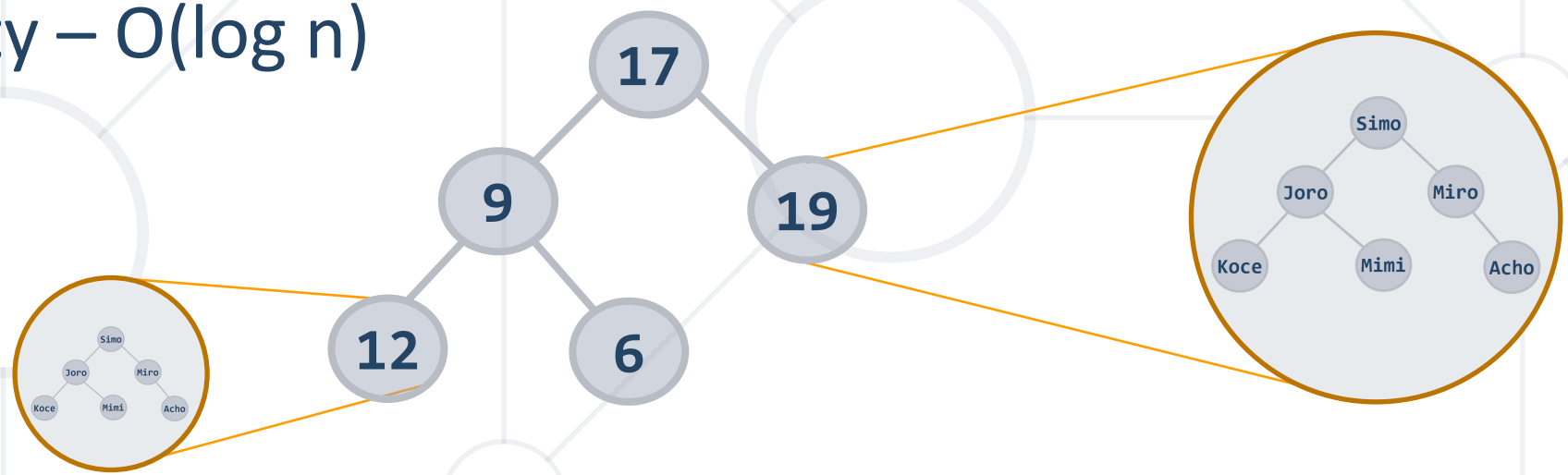
Solution: Find by Name + Town

- Iterator<Person> **findAll**(name, town)
  - **Combine the keys** {*name + town*} into a single string name_town
  - Use a **hash-table** to map {***name_town → SortedSet<Person>***}
  - Complexity – O(1)



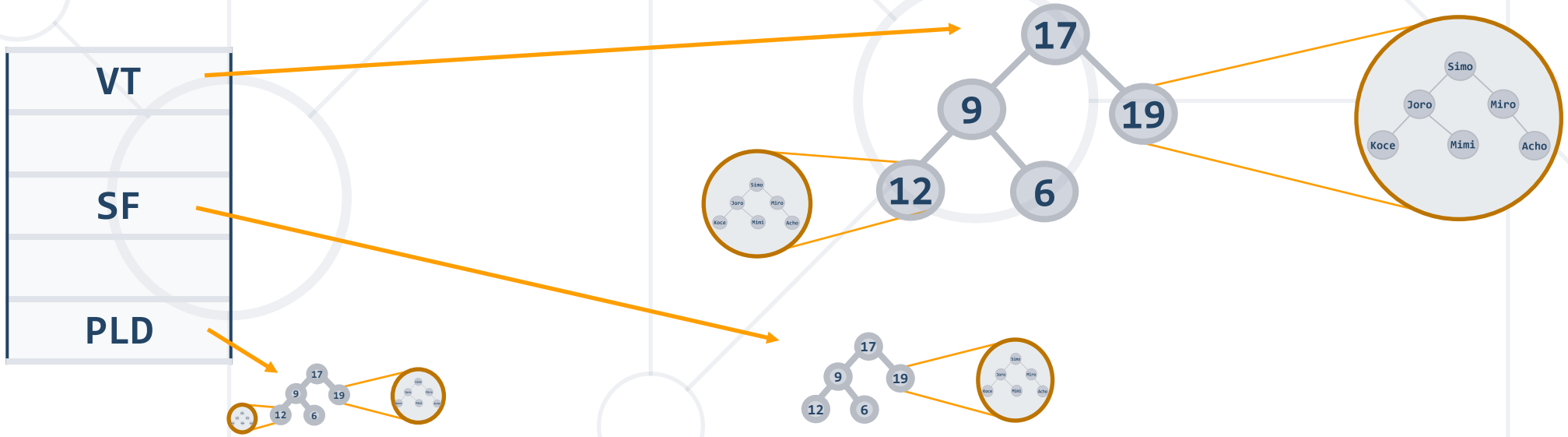
Problem: Collection of People

- Iterator<Person> **findAll**(*start_age, end_age*)
  - Use a **balanced search tree** to keep all persons ordered by age:
    - **TreeMap<Integer, TreeSet<Person>>**
  - Use the **range**(*start_age, end_age*) operation in the tree
  - Complexity – O(log n)



Problem: Collection of People

- Iterator<Person> **findAll**(start_age, end_age, town)
- Use a **hash-table** to map {***town → people by ages***}
  - People by ages can be stored as **balanced search tree**:
    - **TreeMap<Integer, SortedSet<Person>>**



Summary

Software University

- Different data structures have different efficiency for their operations
  - **List-based collections** provide **fast append** and access-by-index, but **slow find** and delete
  - The **fastest** add / find / delete structure is the **hash table** – O(1) for all operations
  - **Balanced trees** are **ordered** – O(log n) for add / find / delete + range(start, end)
- **Data structures Augmentation** is often essential
  - E.g. combine multiple hash-tables to find by different keys

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