Hash Tables, Sets And Dictionaries

Hashing and Collisions

0	1	2	…	m-1
null	null	SoftUni	…	C#

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Key	Value
1	A
7	B

Hash Tables

Hashing and Collision Resolution

Hash Function

- Given a key of any type, convert it to an integer

Ivan — Hash Function — 398

Pesho — Hash Function — 511

```
class Person {
  string firstName;
  string lastName;
  int age;
}
```

Ivan
Petrov
25 — Hash Function — 25950

Hash Function (2)

```
class Person {
  string firstName;
  string lastName;
  int age;


  public override int GetHashCode() {

          Hash Function

  }
}
```

Hash Function (2)

```
class Person {
  string firstName;
  string lastName;
  int age;


  public override int GetHashCode() {
    int firstNameHash = firstName.GetHashCode() * age;
    int lastNameHash = lastName.GetHashCode() * age;

    return firstNameHash + lastNameHash;
  }
}
```

Hash Table

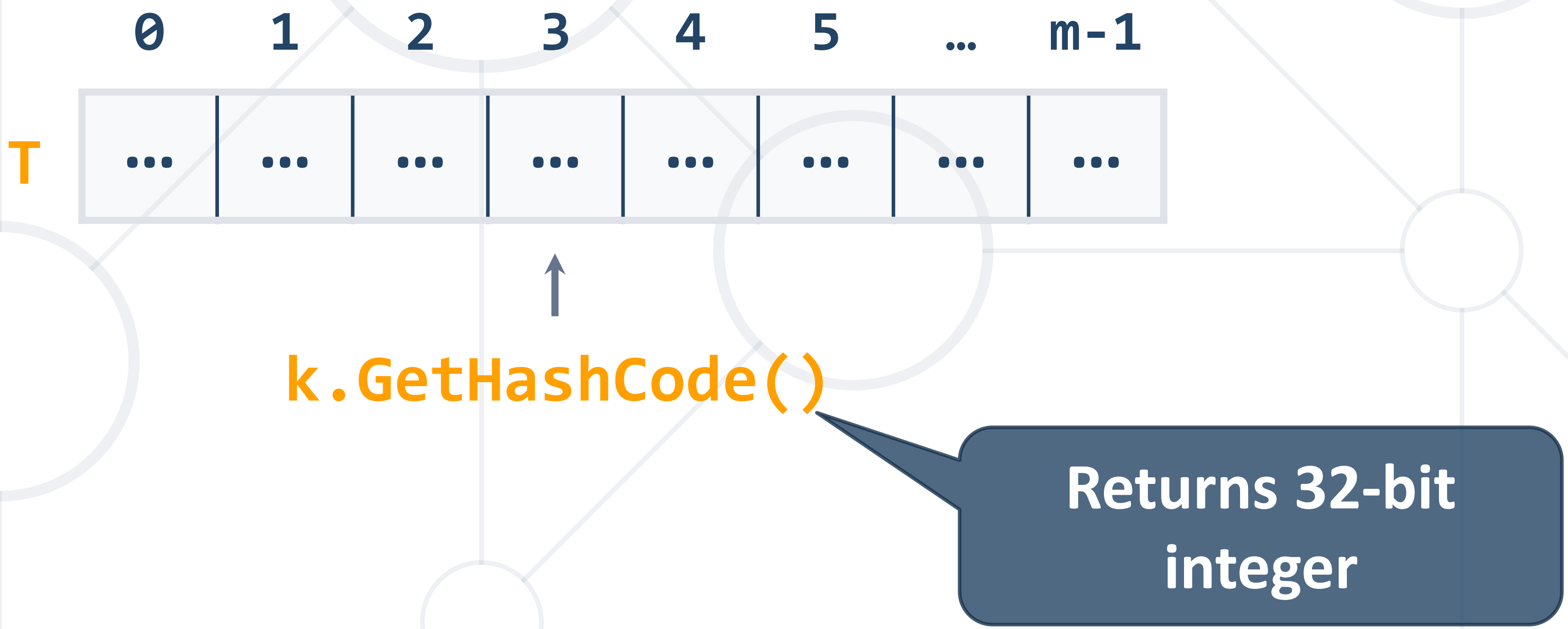
- A hash table is an array that holds a set of **{key, value} pairs**
- The process of mapping a key to a position in a table is called **hashing**

	0	1	2	3	4	5	…	m-1
T	…	…	…	…	…	…	…	…

Hash table of size m

Hash Functions and Hashing

- A hash table has **m** slots, indexed from **0** to **m-1**
- A hash function converts **keys** into array indices



Hashing Functions

- Perfect hashing function (PHF)
  - **h(k)**: one-to-one mapping of each key **k** to an integer in the range **[0, m-1]**
  - The PHF maps each key to a **distinct** integer within some manageable range
- Finding a perfect hashing function is impossible in most cases

Hashing Functions (2)

- Good hashing function
  - **Consistent** - equal keys must produce the same hash value
  - **Efficient** - efficient to compute the hash
  - **Uniform** - should uniformly distribute the keys

Hash Functions – Quiz

TIME'S

- Which of the following is **not** property of a **GetHashCode()** for strings
  - Can return a negative integer
  - Can take time proportional to the length of the string to compute
  - A string and its reverse will have the same hash code
  - Two strings with different hash code values are different strings

Hash Functions – Answer

- Which of the following is **not** property of a **GetHashCode()** for strings
  - Can return a negative integer
  - Can take time proportional to the length of the string to compute
  - A string and its reverse will have the same hash code
  - Two strings with different hash code values are different strings

"ab".GetHashCode() != "ba".GetHashCode()

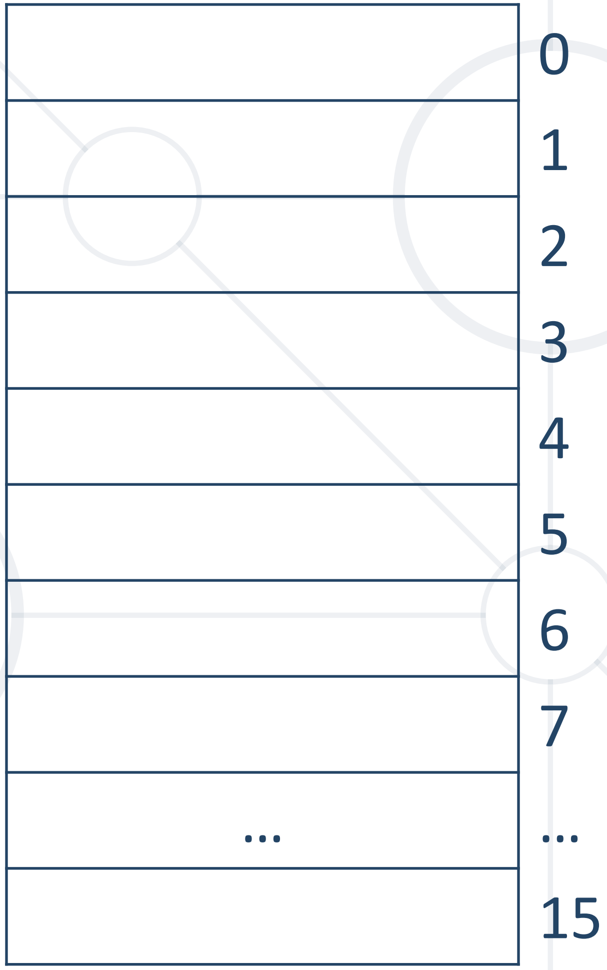
Modular Hashing

- Array with length 16
- Insert "Example"

511 is bigger than the table length

- Use the remainder of
`GetHashCode() / Array.Length`

511 % 16 = 15



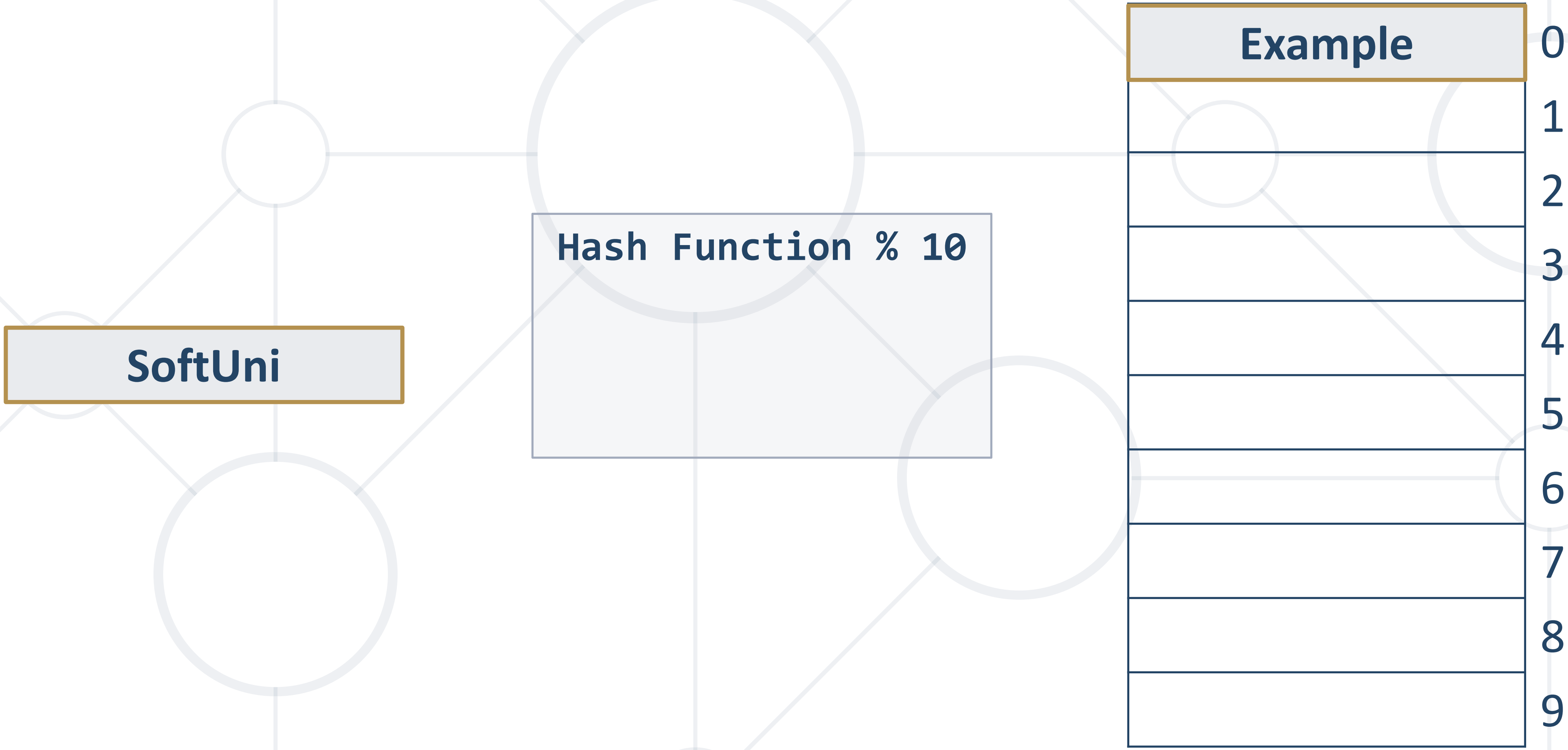
Adding to Hash Table

Example

Hash Function % 10

	0
	1
	2
	3
	4
	5
	6
	7
	8
	9

Adding to Hash Table (2)



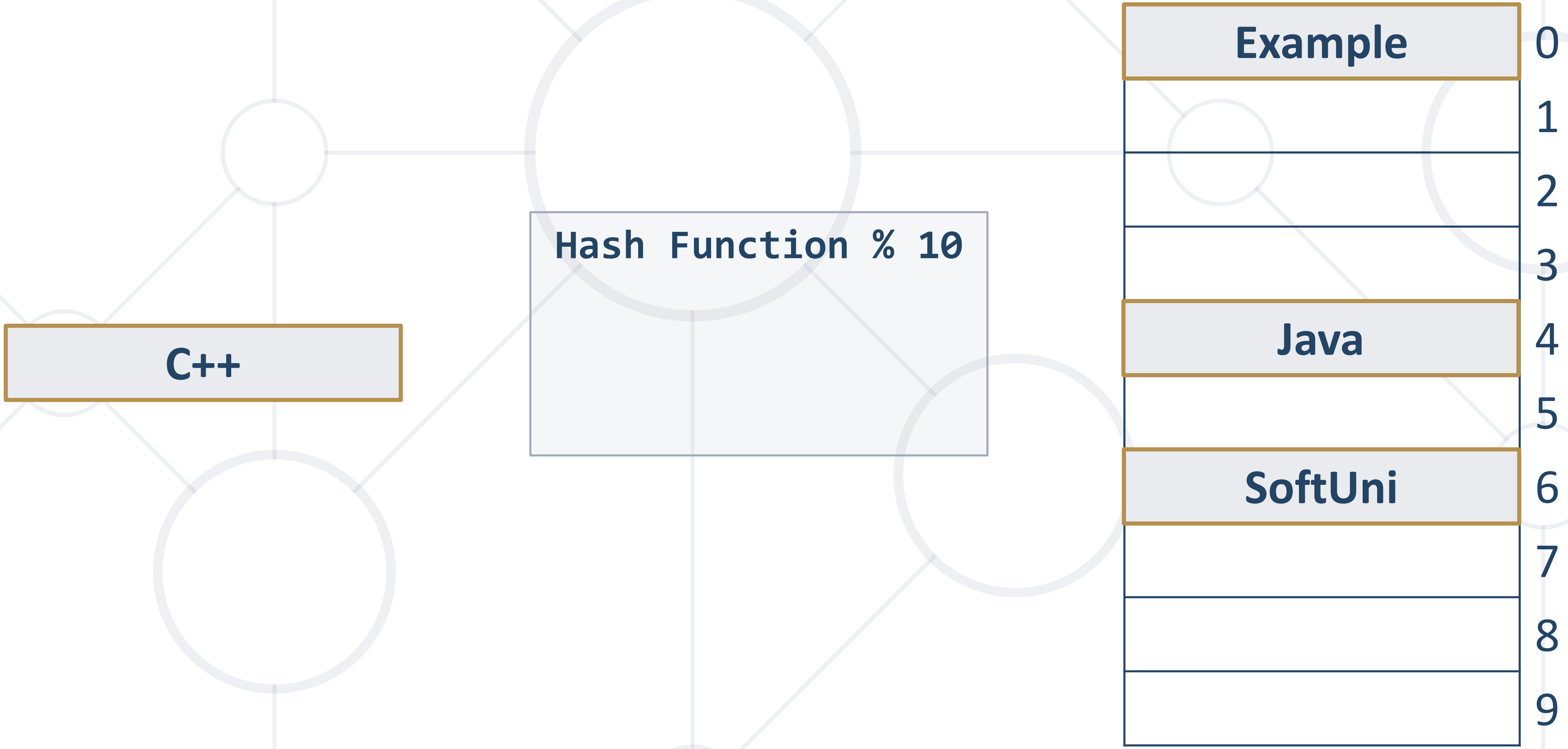
Adding to Hash Table (3)

Java

Hash Function % 10

Example	0
	1
	2
	3
	4
	5
SoftUni	6
	7
	8
	9

Adding to Hash Table (4)



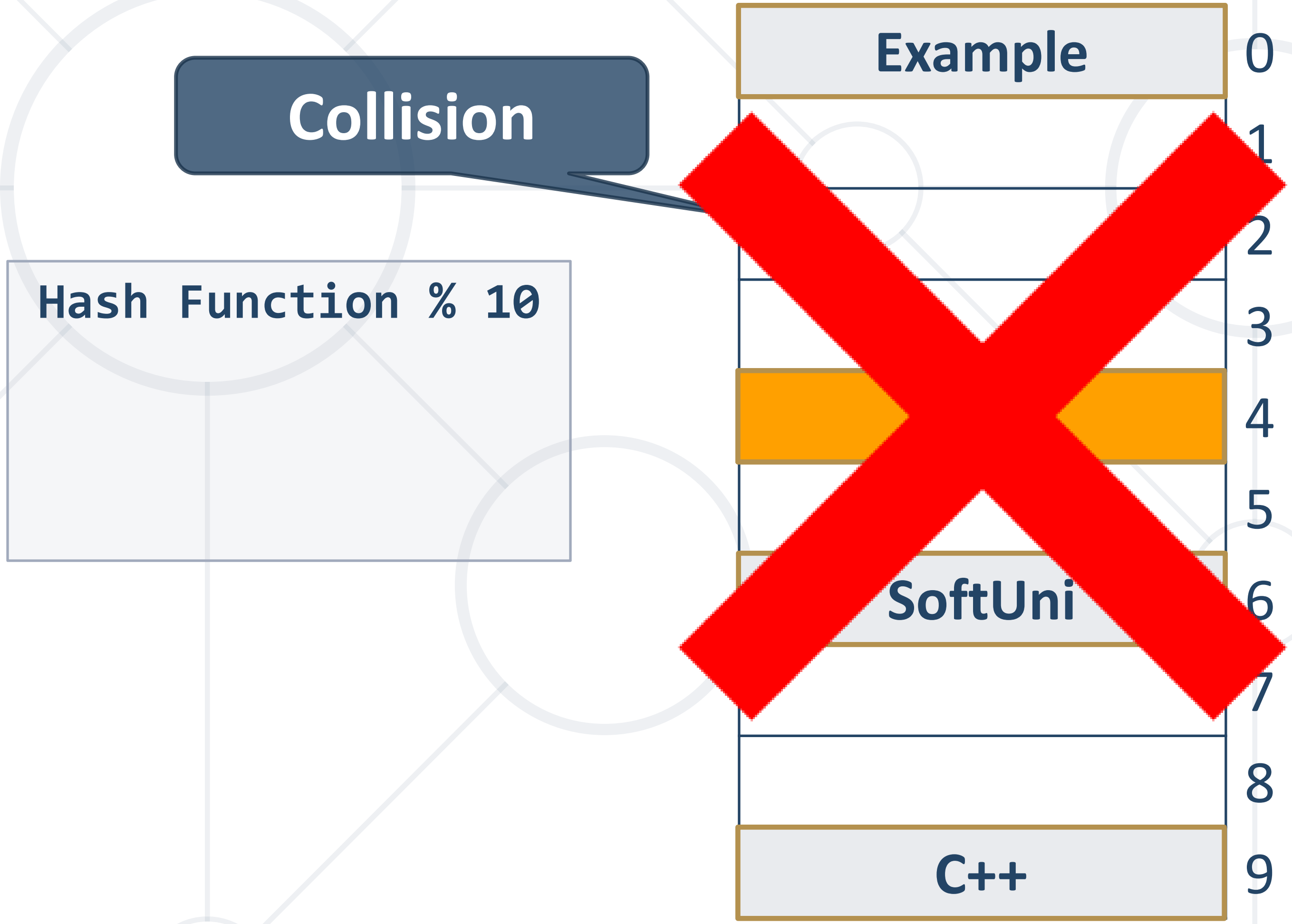
Adding to Hash Table (5)

C#

```
Hash Function % 10
```

Example	0
	1
	2
	3
Java	4
	5
SoftUni	6
	7
	8
C++	9

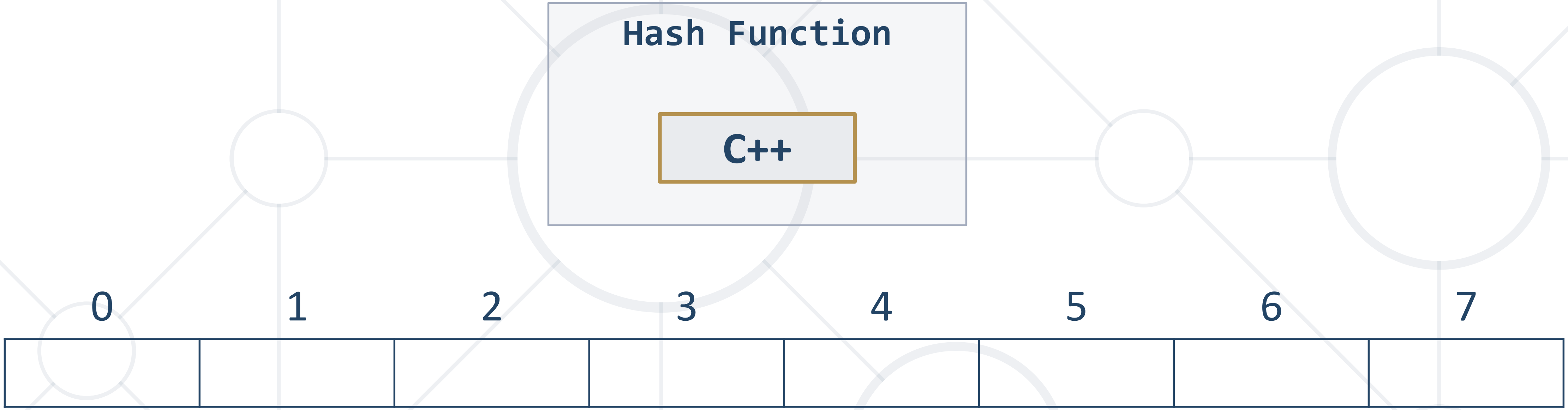
Adding to Hash Table (6)



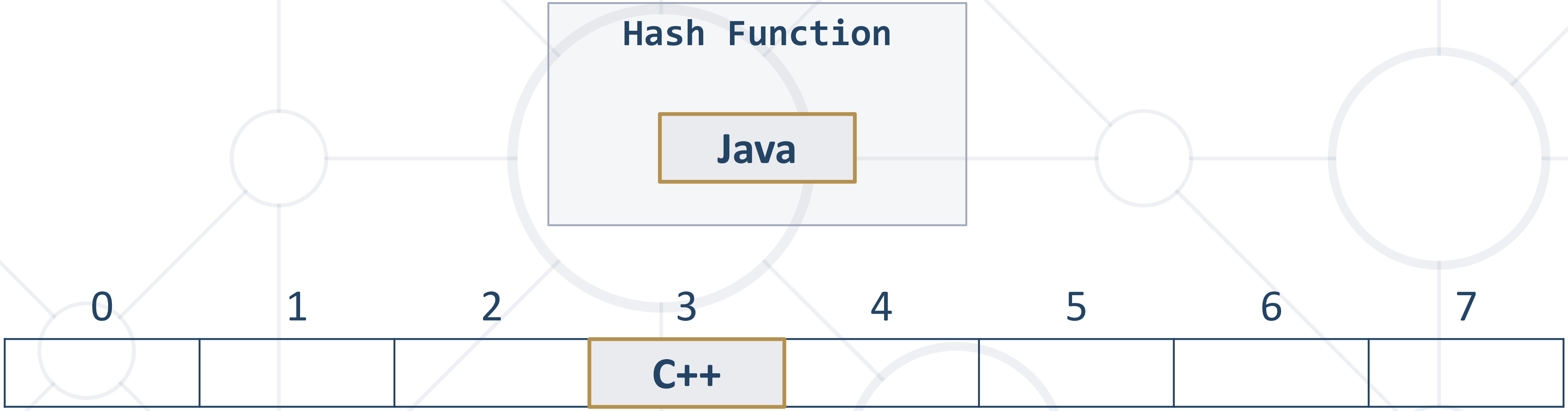
Collisions in a Hash Table

- A **collision** comes when **different keys** have the **same hash value**
  - $h(k_1) = h(k_2)$ for $k_1 \neq k_2$
- When the number of collisions is sufficiently small, the hash tables work quite well (fast)
- Several **collisions resolution strategies** exist
  - **Chaining** collided keys (+ values) in a list
  - Using **other slots** in the table (open addressing)
  - Cuckoo hashing
  - Many other

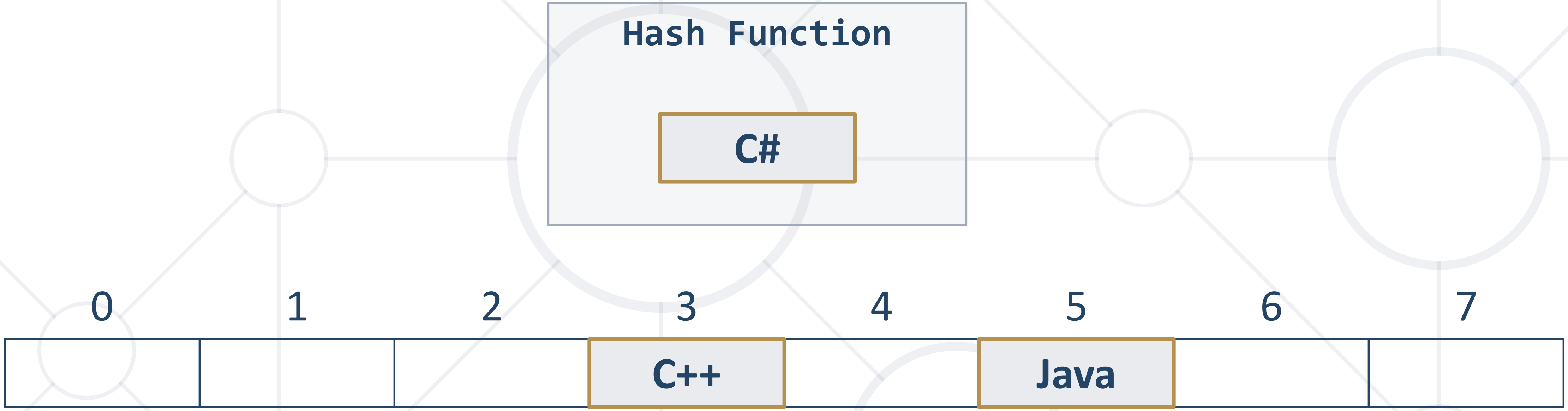
Collision Resolution: Chaining



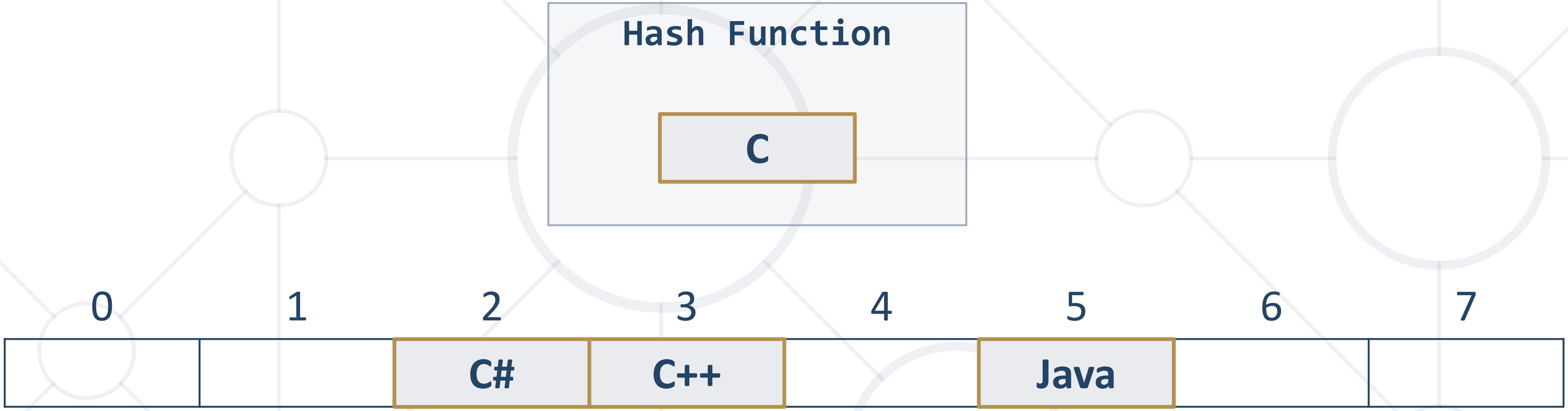
Collision Resolution: Chaining



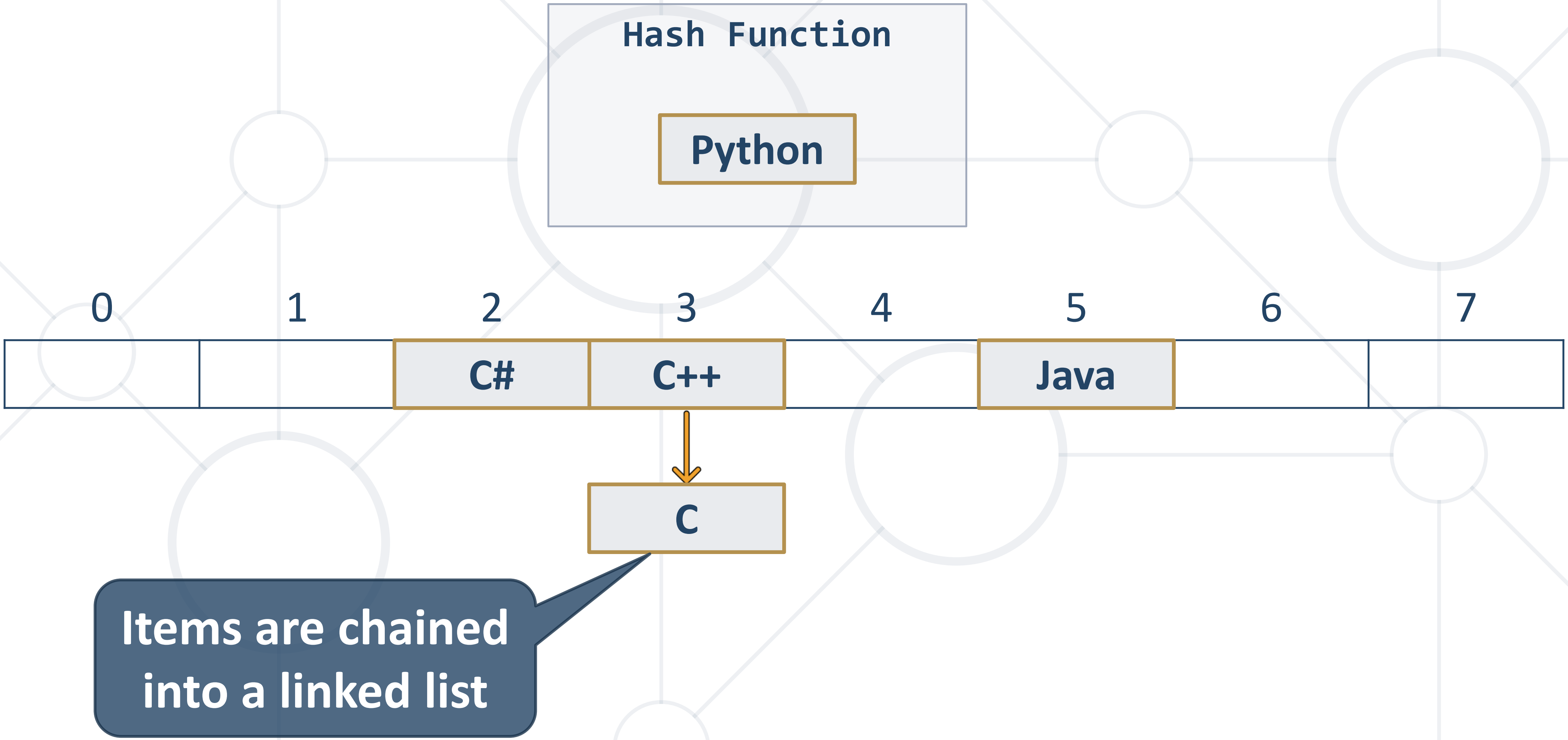
Collision Resolution: Chaining



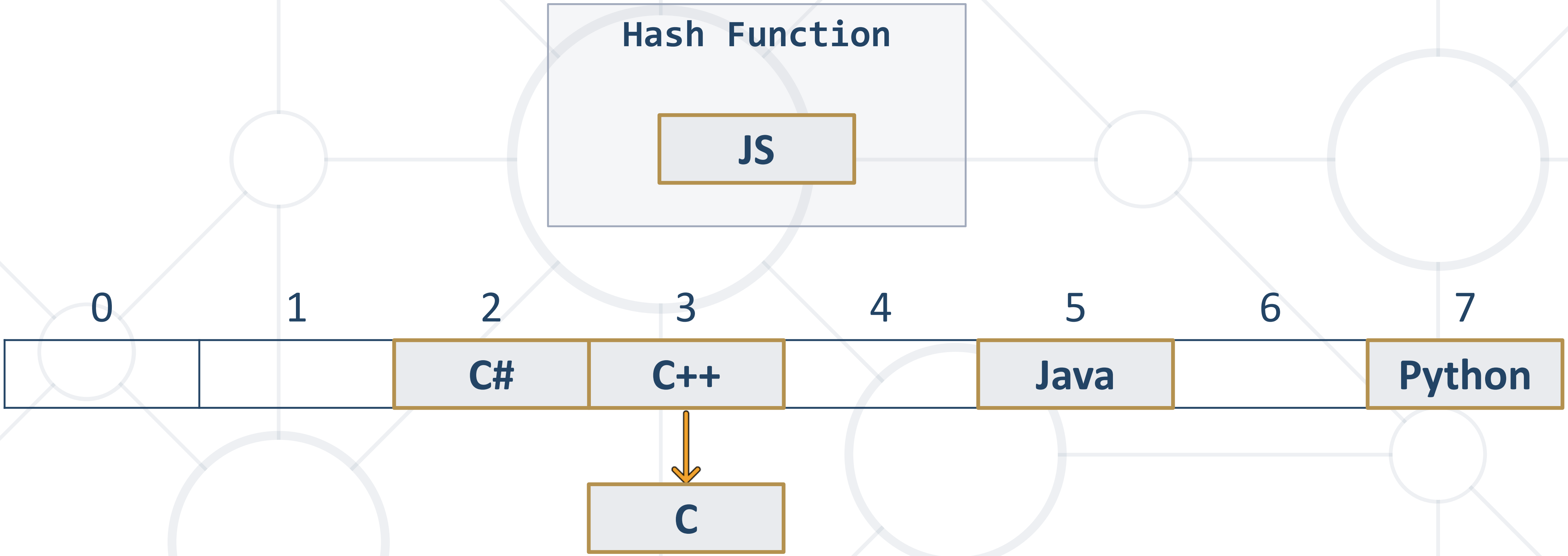
Collision Resolution: Chaining



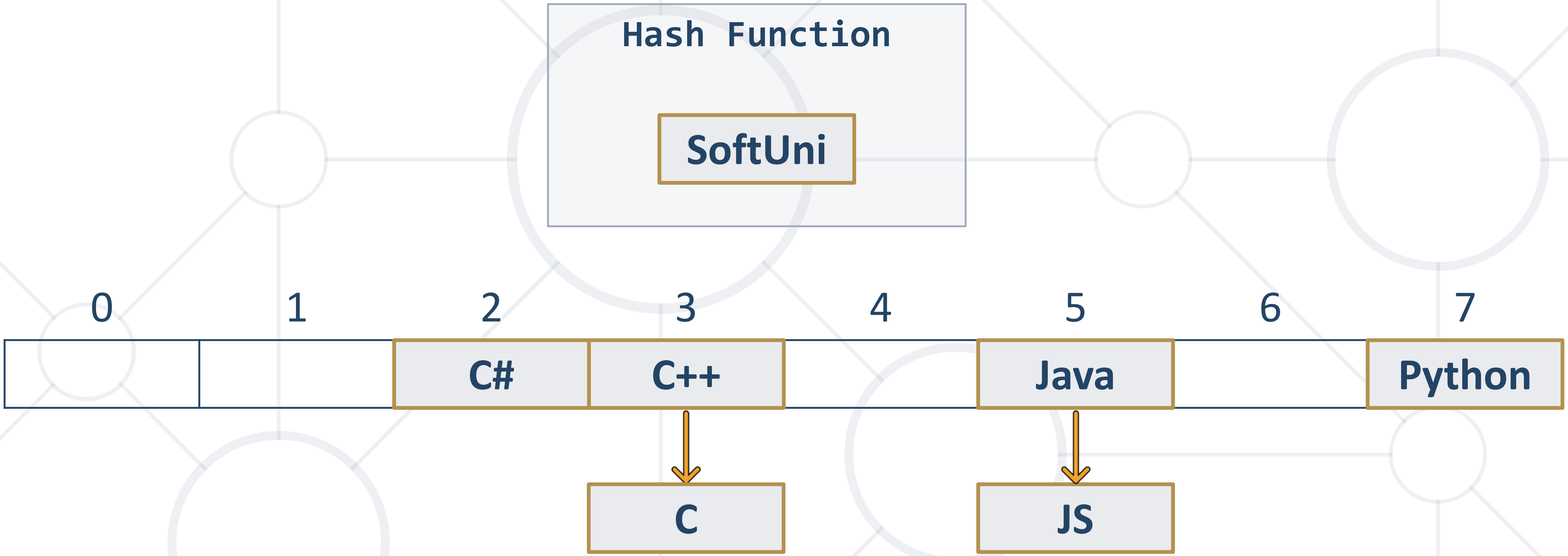
Collision Resolution: Chaining



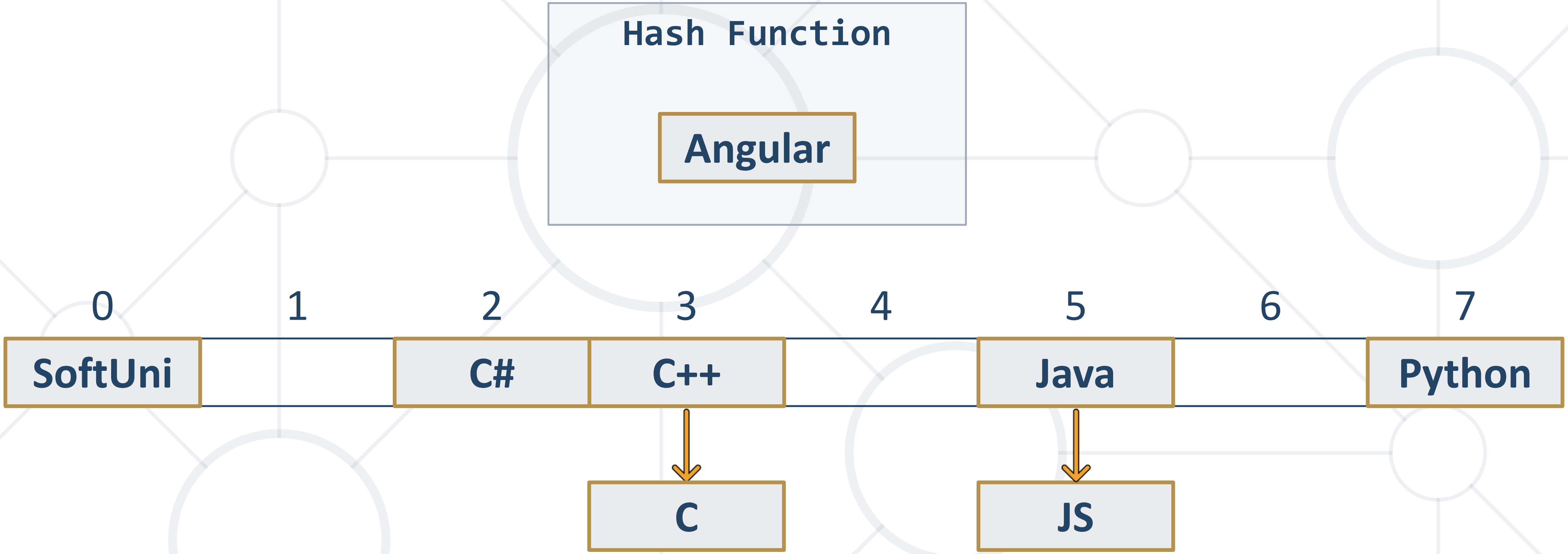
Collision Resolution: Chaining



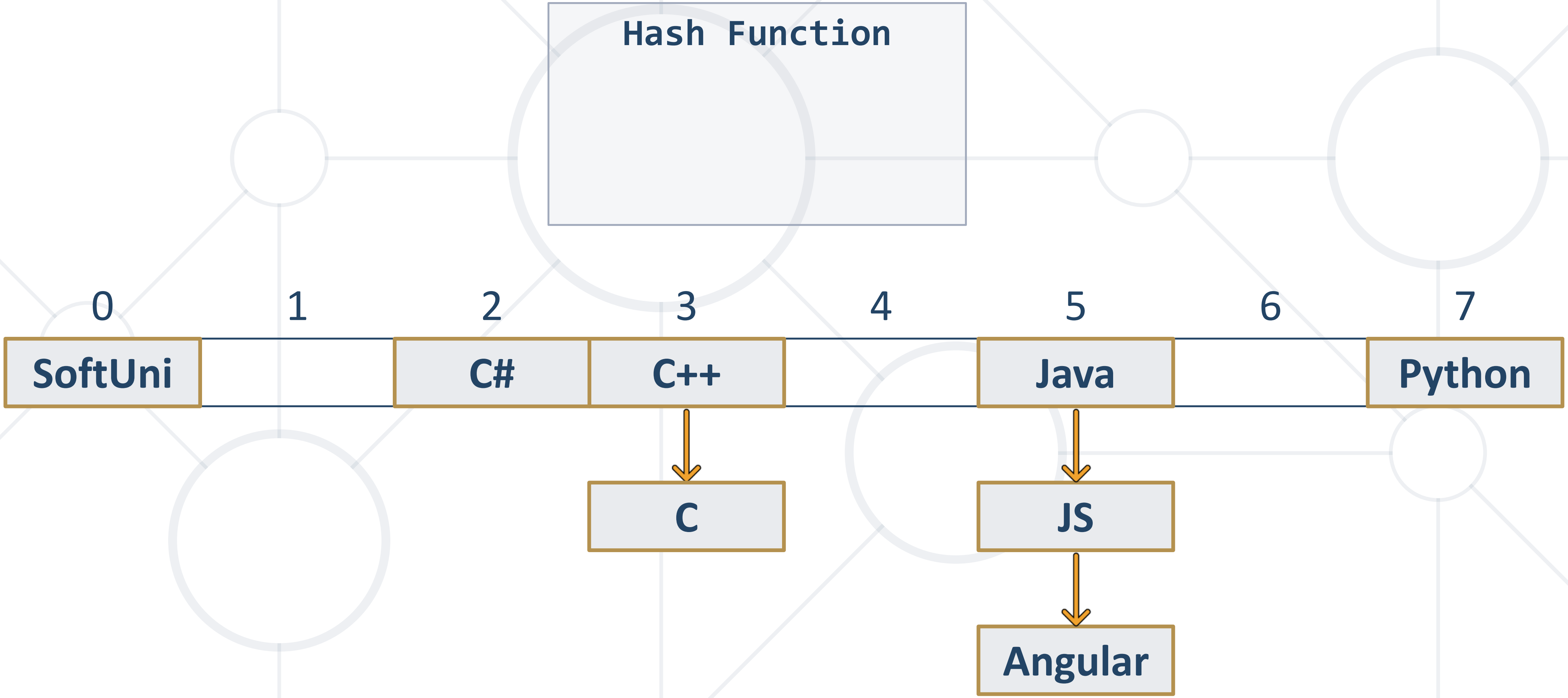
Collision Resolution: Chaining



Collision Resolution: Chaining



Collision Resolution: Chaining



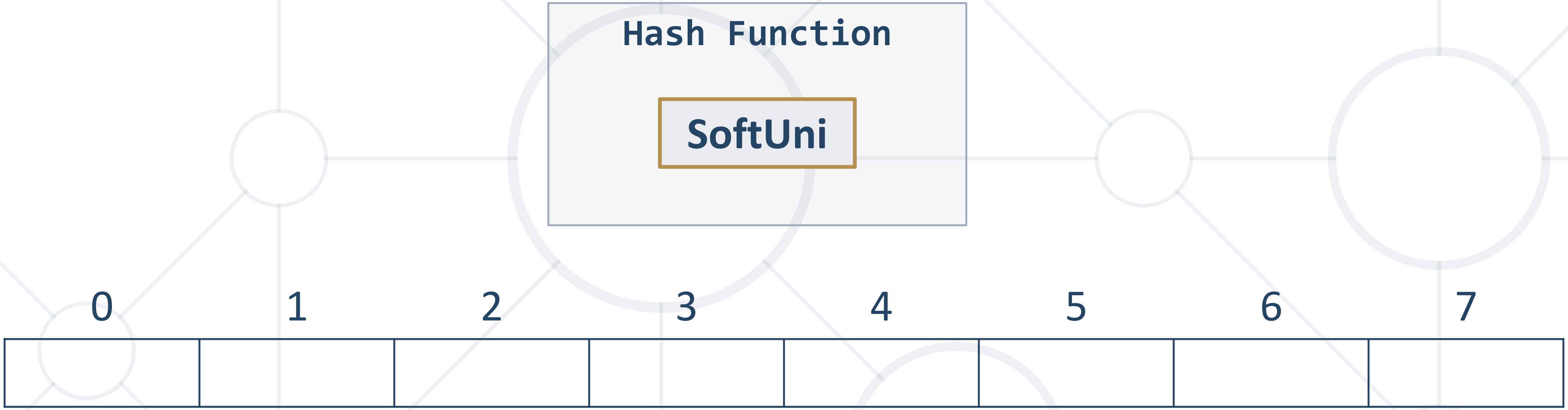
Collision Resolution: Open Addressing

- **Open addressing** as collision resolution strategy means to take another slot in the hash-table in case of collision, e.g.
  - **Linear probing**: take the next empty slot just after the collision
    - **h(key, i) = h(key) + i**
    - where **i** is the attempt number: 0, 1, 2, …
    - **h(key) + 1, h(key) + 2, h(key) + 3**, etc.

Collision Resolution: Open Addressing (2)

- **Quadratic probing**: the i^{th} next slot is calculated by a quadratic polynomial (c_1 and c_2 are some constants)
  - $h(key, i) = h(key) + c_1 * i + c_2 * i^2$
  - $h(key) + 1^2$, $h(key) + 2^2$, $h(key) + 3^2$, etc.
- **Re-hashing**: use separate (second) hash-function for collisions
  - $h(key, i) = h_1(key) + i * h_2(key)$

Collision Resolution: Linear Probing



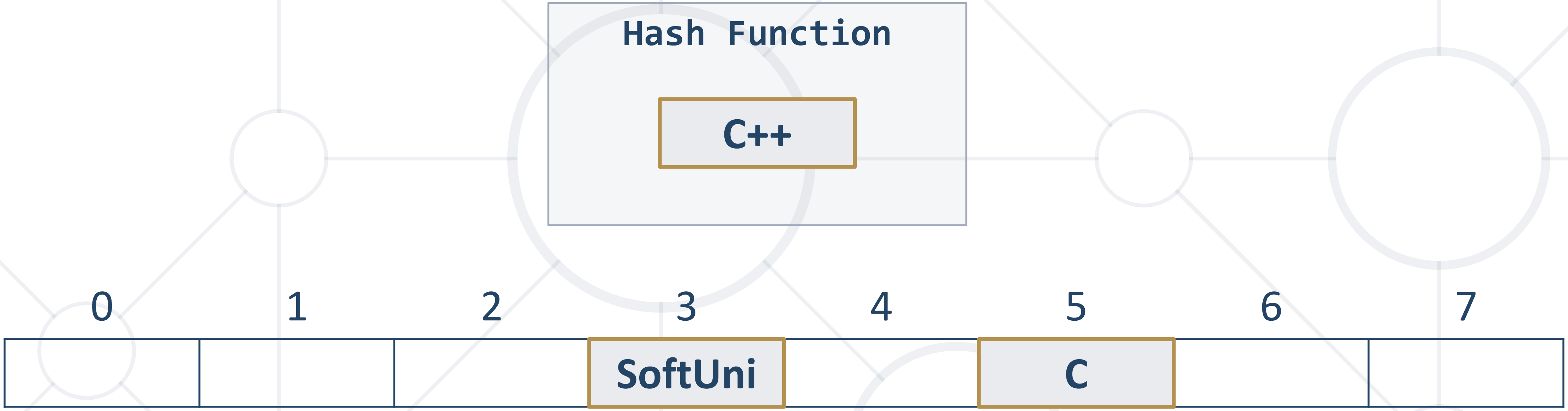
Collision Resolution: Linear Probing

Hash Function

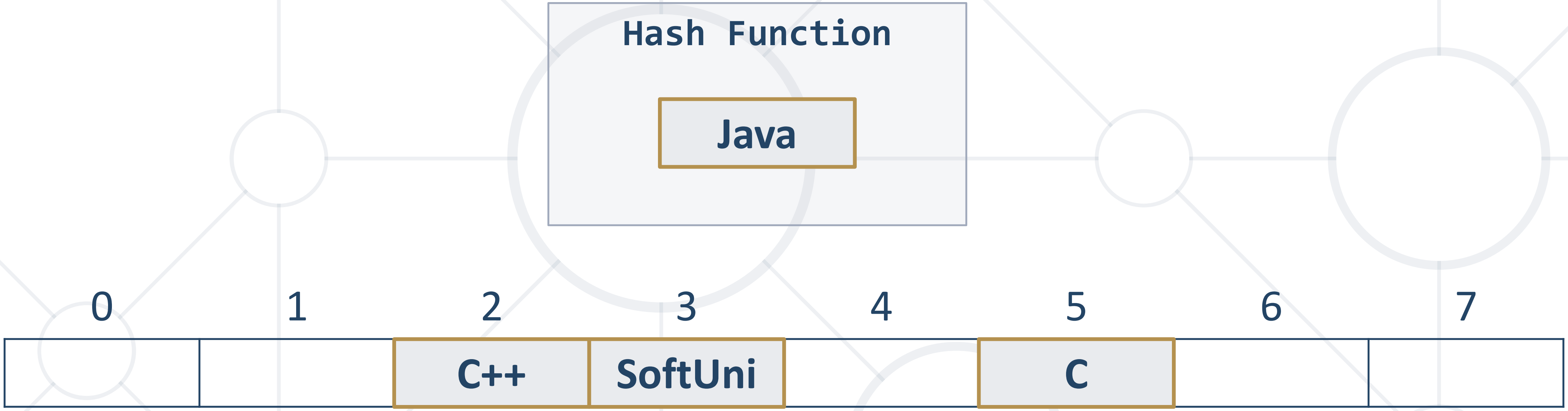
C

0	1	2	3	4	5	6	7
			SoftUni				

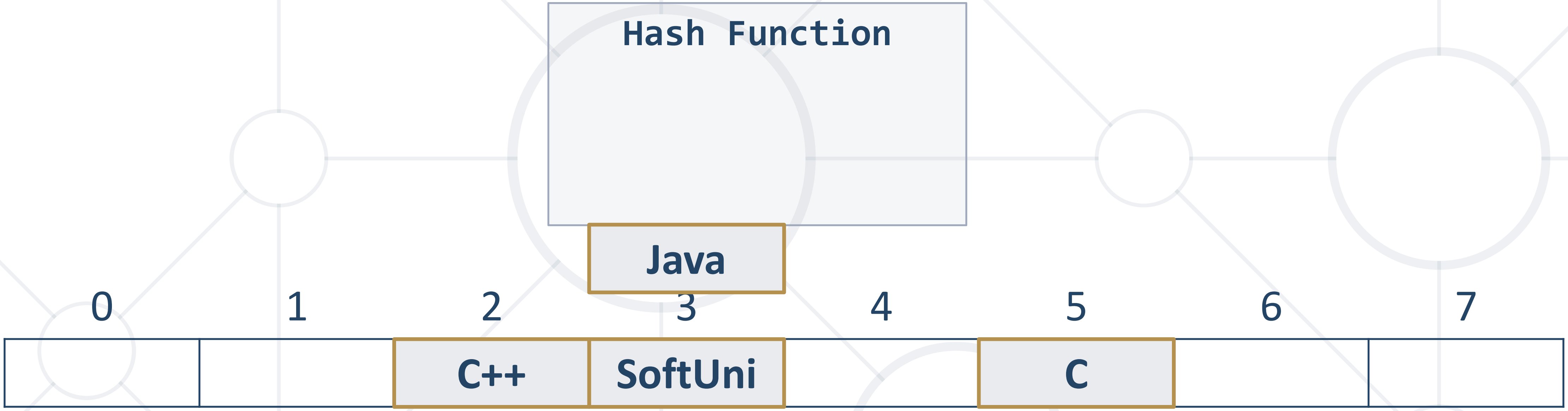
Collision Resolution: Linear Probing



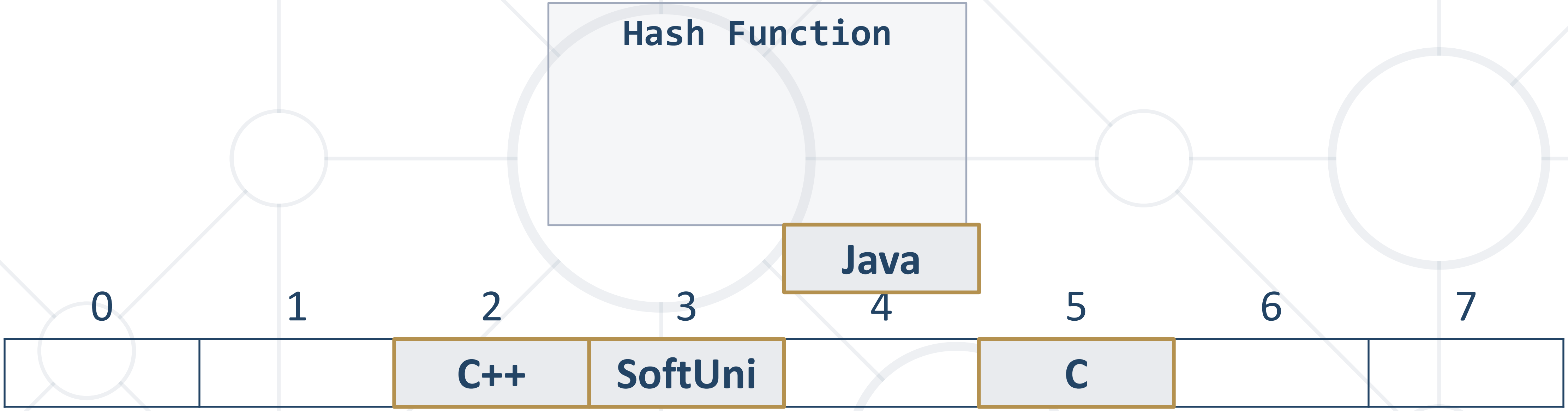
Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



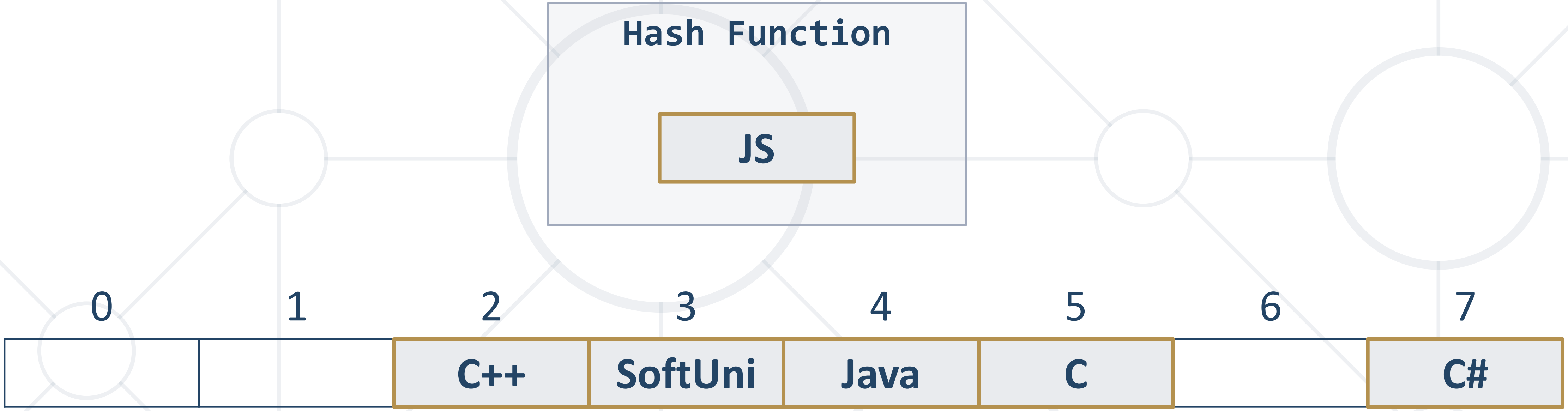
Collision Resolution: Linear Probing

Hash Function

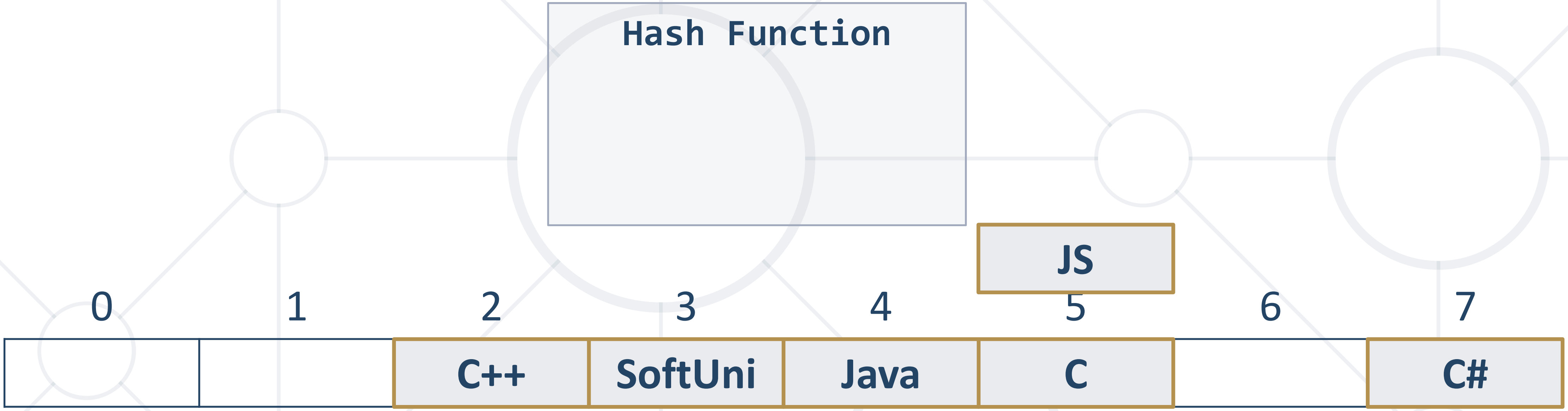
C#

0	1	2	3	4	5	6	7
		C++	SoftUni	Java	C		

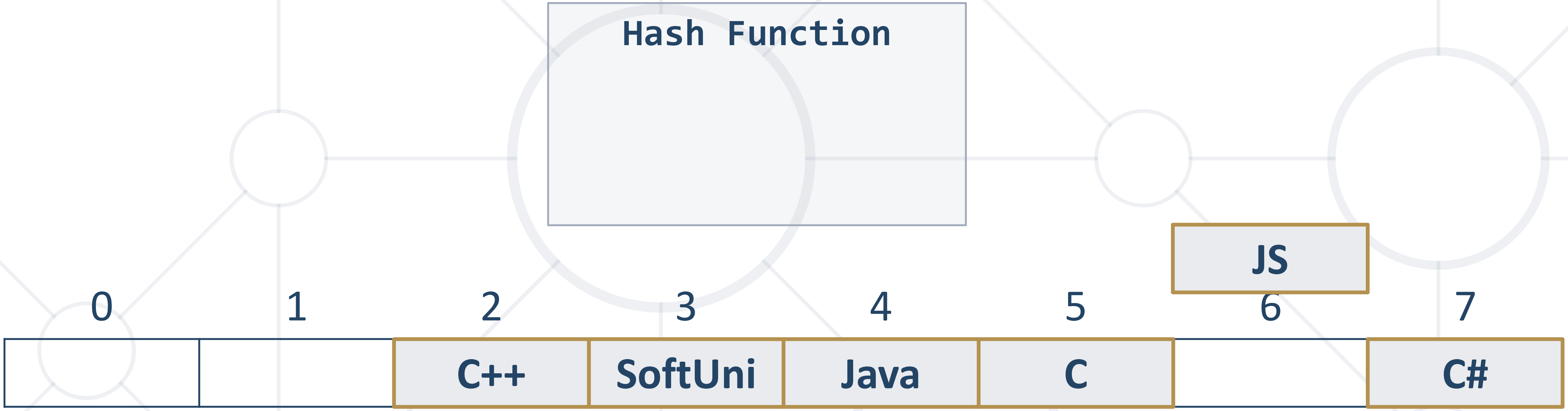
Collision Resolution: Linear Probing



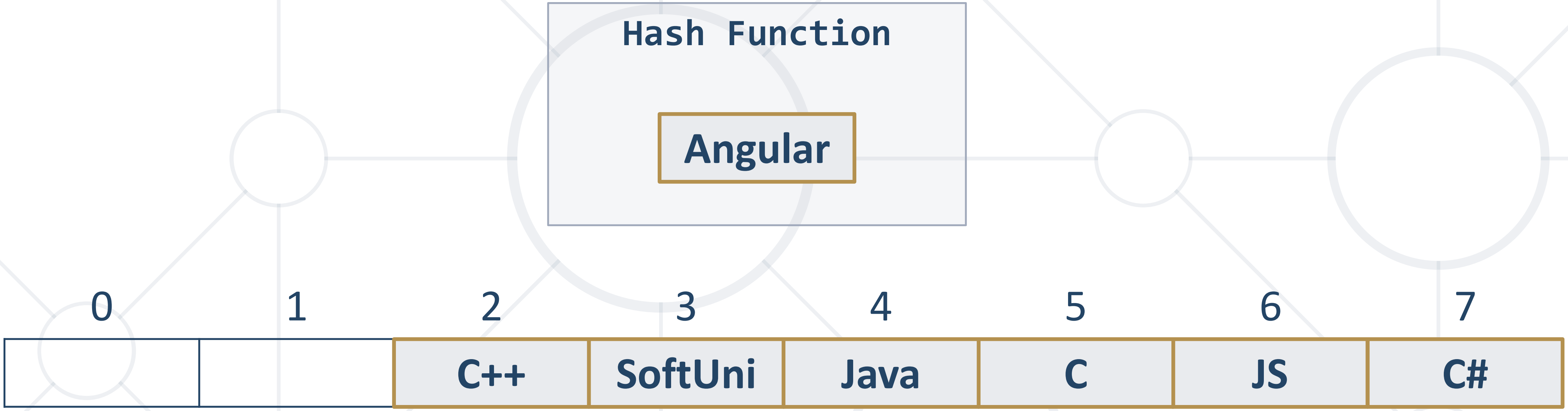
Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



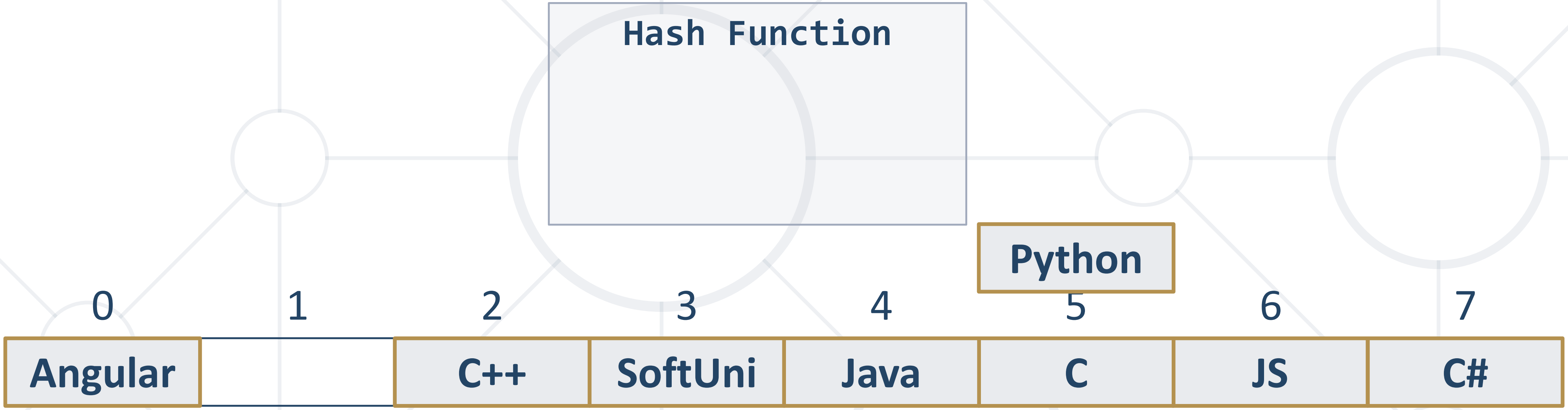
Collision Resolution: Linear Probing

Hash Function

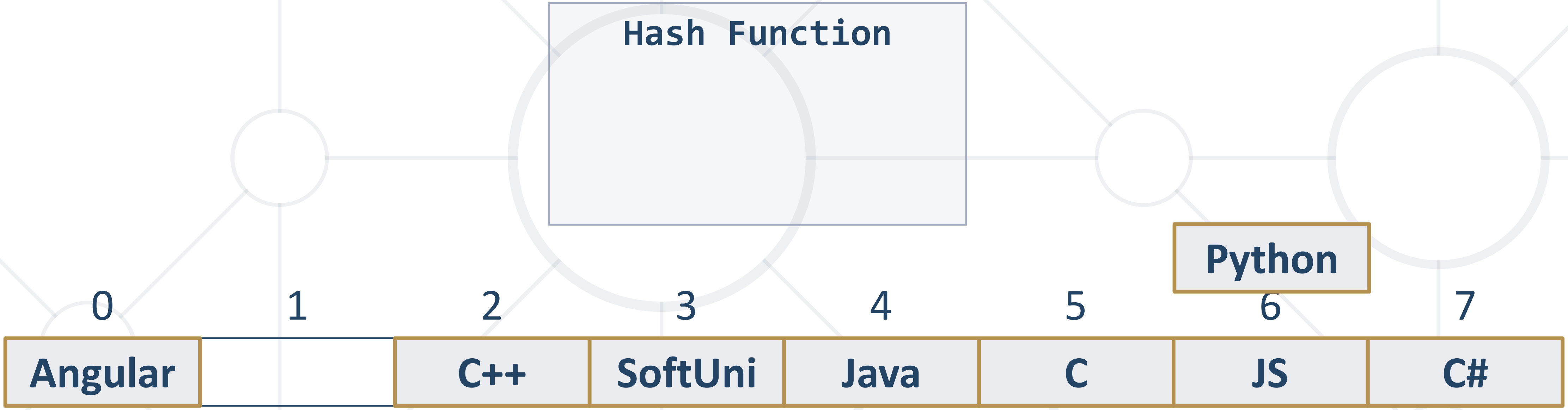
Python

0	1	2	3	4	5	6	7
Angular		C++	SoftUni	Java	C	JS	C#

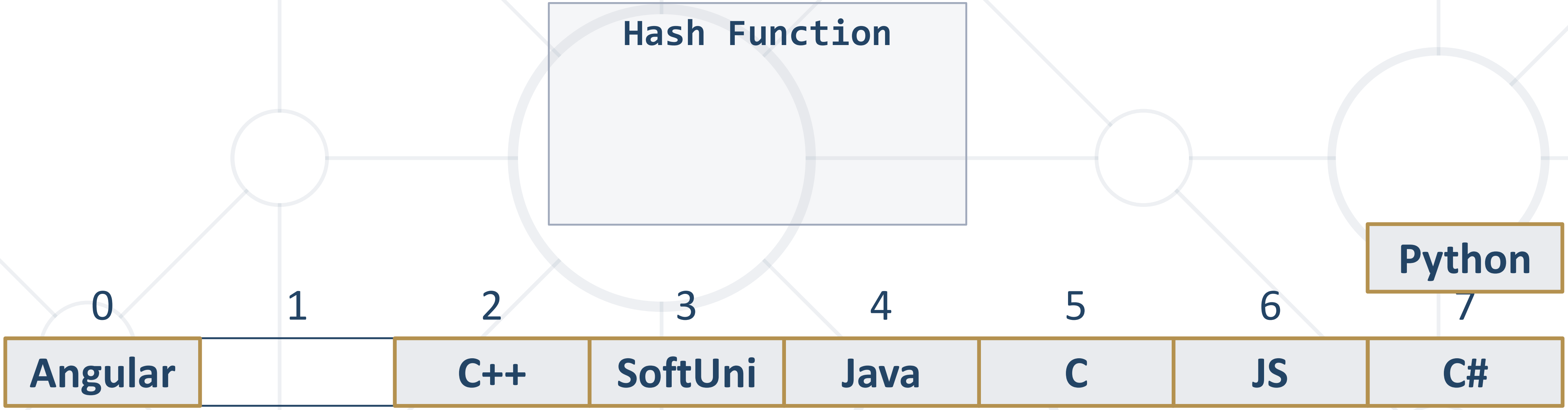
Collision Resolution: Linear Probing



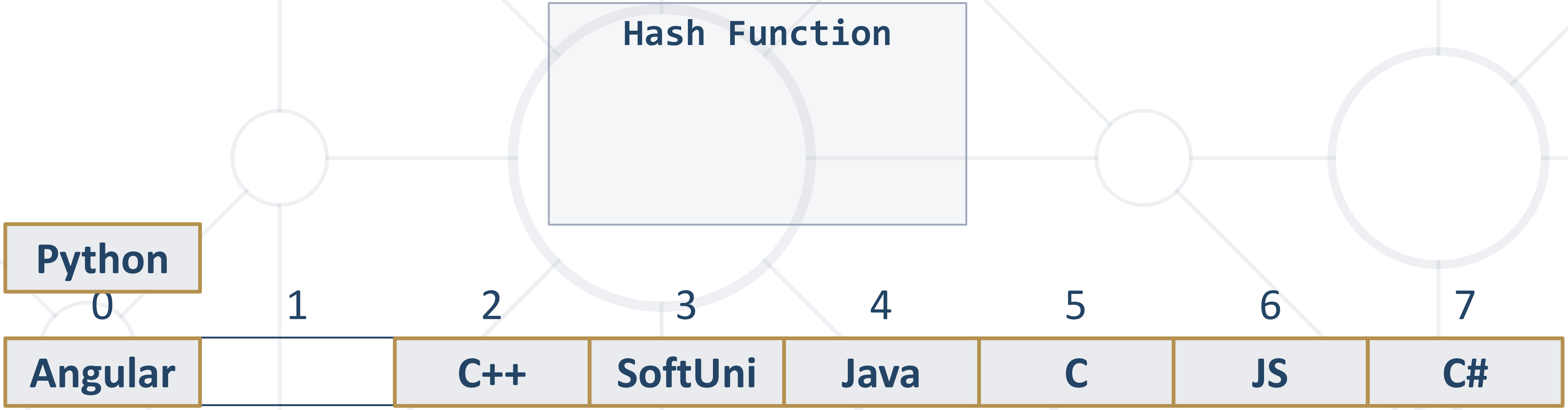
Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing

Hash Function

Python

0	1	2	3	4	5	6	7
Angular		C++	SoftUni	Java	C	JS	C#

Collision Resolution: Linear Probing

Hash Function

0	1	2	3	4	5	6	7
Angular	Python	C++	SoftUni	Java	C	JS	C#

Linear Probing – Quiz

TIME'S

- What is the average running time of delete in linear-probing hash table? Your hash function satisfies the uniform hashing assumption and that the hash table is at most 50% full.
  - O(1)
  - O(log N)
  - O(N)
  - O(N log N)

Linear Probing – Answer

- What is the average running time of delete in linear-probing hash table? Your hash function satisfies the uniform hashing assumption and that the hash table is at most 50% full.
  - O(1) ✔
  - O(log N)
  - O(N)
  - O(N log N)

Hash Table Performance

- The hash-table performance depends on the probability of collisions - Less collisions = faster add / find / delete operations
  - **Collisions resolution** algorithm
  - **Fill factor** (used buckets / all buckets)

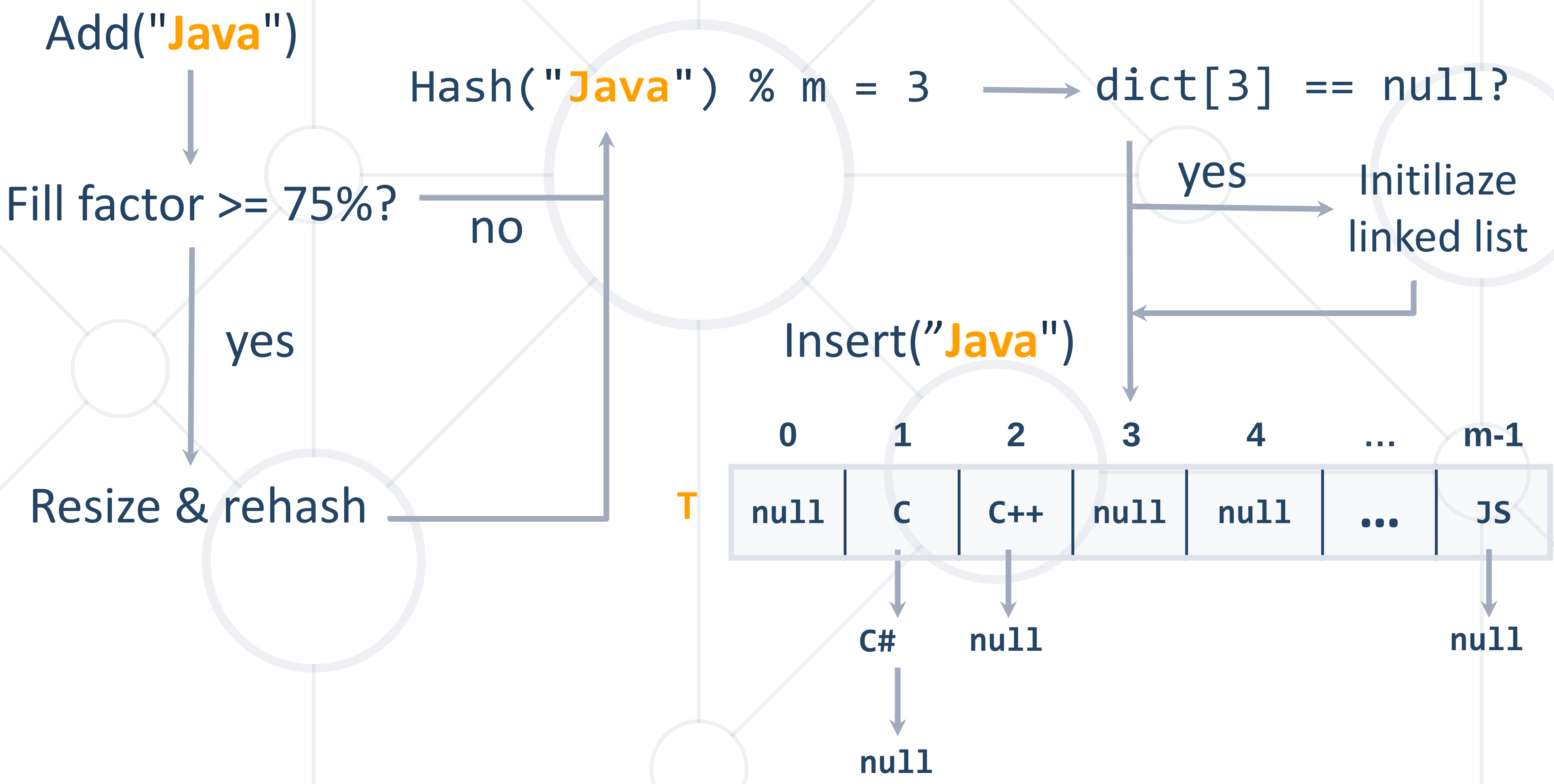
Hash Tables Efficiency

- **Add** / **Find** / **Delete** take just few primitive operations
  - Speed does not depend on the size of the hash-table
  - Amortized complexity **O(1)** – constant time
- Example:
  - Finding an element in a **hash-table** holding **1 000 000 elements** takes average just **1-2 steps**
  - Finding an element in an **array** holding **1 000 000 elements** takes average **500 000 steps**

How Big the Hash-Table Should Be?

- The **load factor** (fill factor) **= used cells / all cells**
  - How much the hash table is filled, e.g. 65%
- Smaller fill factor leads to less collisions (faster average seek time)
- Recommended fill factors:
  - When **chaining** is used as collision resolution less than **75%**
  - When **open addressing** is used less than **50%**

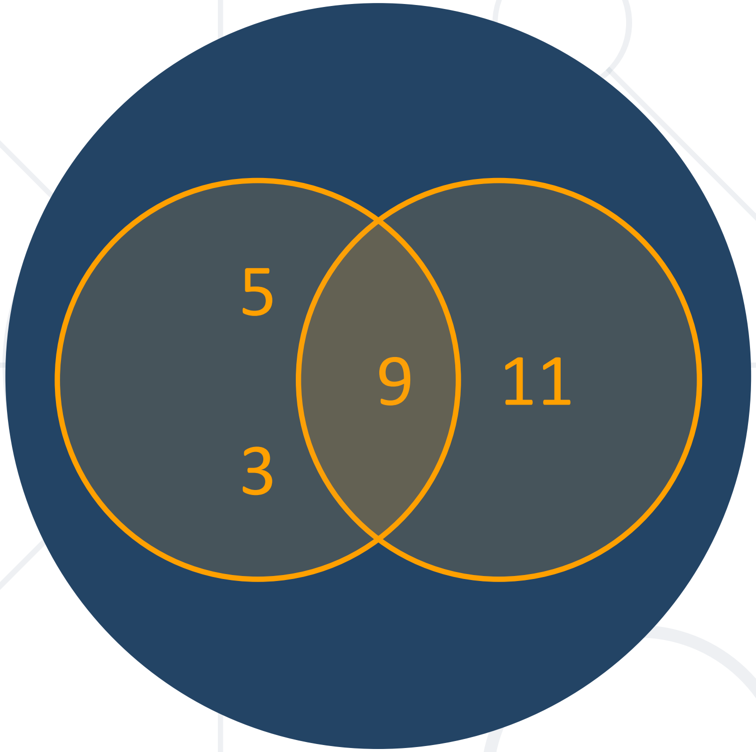
Adding Item to Hash Table with Chaining



Key	Value
1	A
7	B

Lab Exercise

Implement a Hash-Table with Chaining



Sets and Bags

Set Operations

Set and Bag ADTs

- The abstract data type (ADT) "**Set**" keeps a set of elements with no duplicates
- Sets with duplicates are also known as ADT "**Bag**"
- Set specific operations:
  - **UnionWith(set)**
  - **IntersectWith(set)**
  - **ExceptWith(set)**
  - **SymmetricExceptWith(set)**

Known as relative complement in math

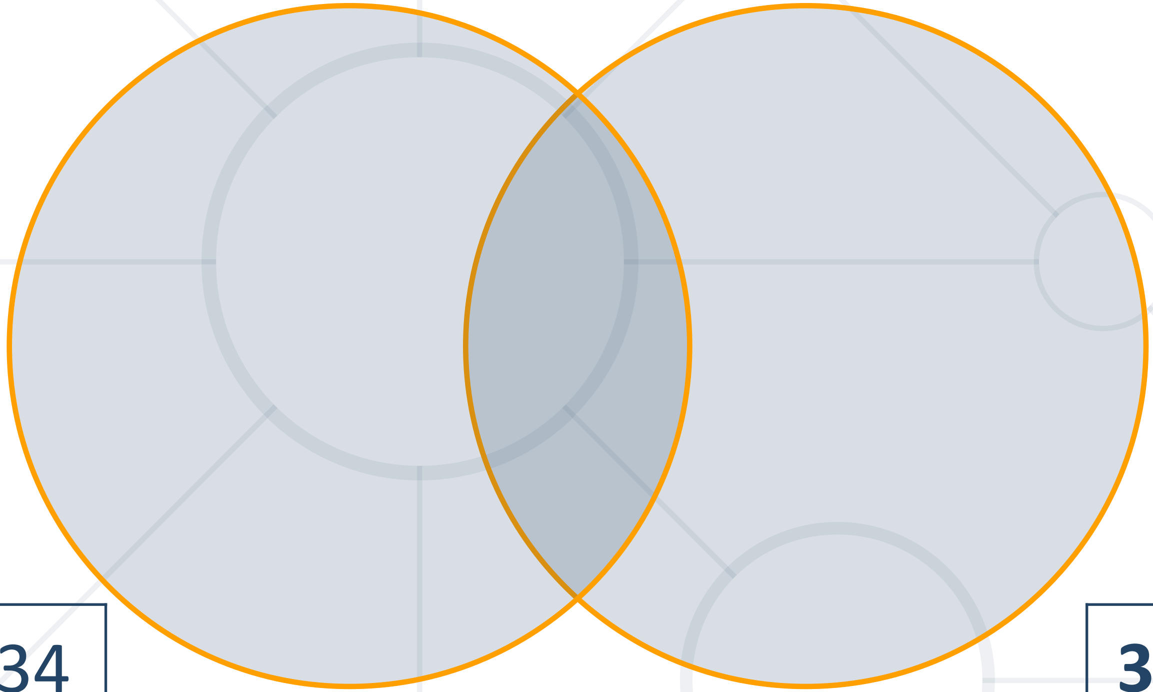
Known as symmetric difference

Union

1	5	2	11	8	34

3	2	9	34	8	0

Union

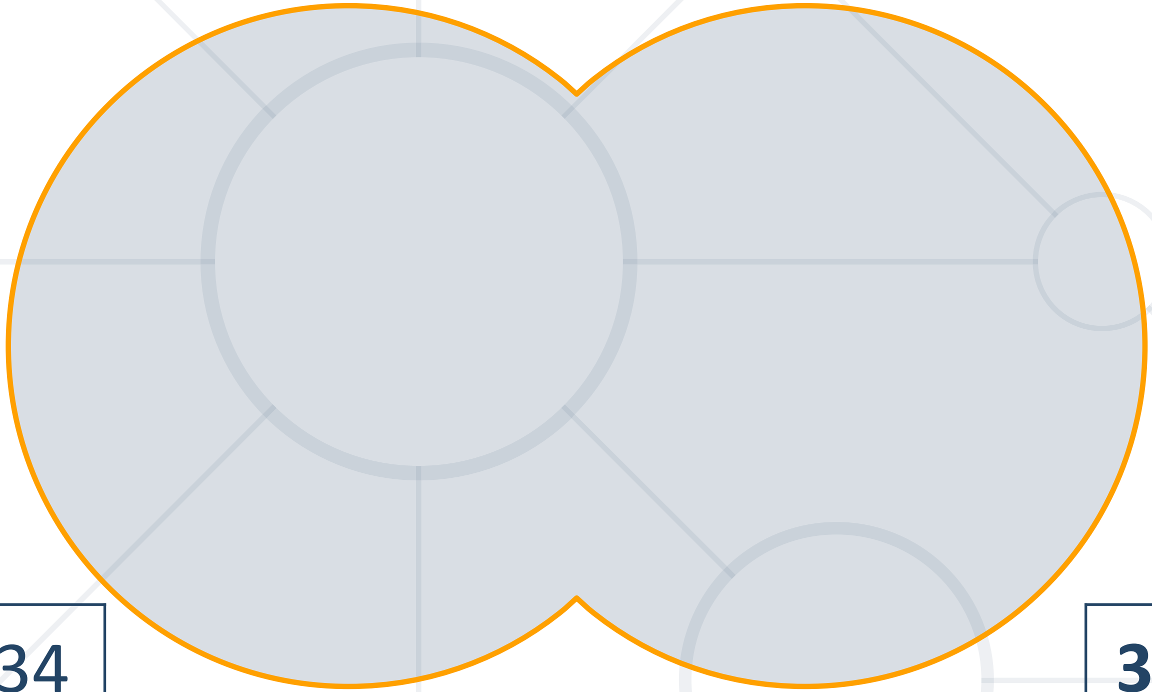


1	5	2	11	8	34

3	2	9	34	8	0

1	5	2	11	8	34	3	9	0

Union



1	5	2	11	8	34

3	2	9	34	8	0

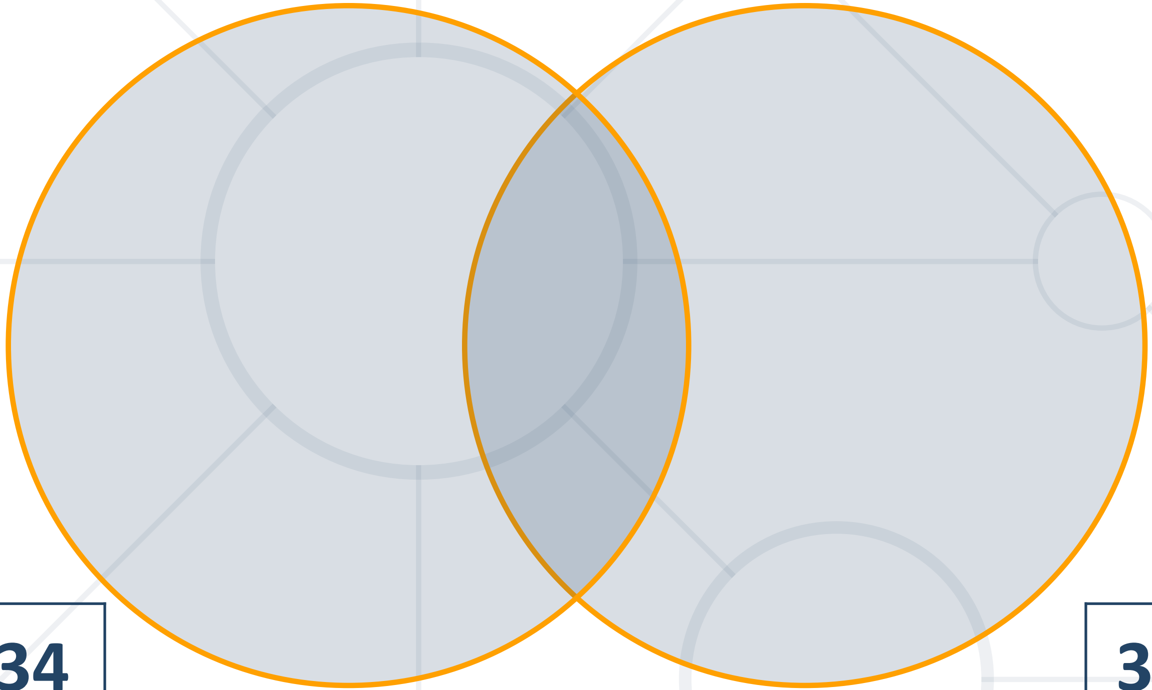
1	5	2	11	8	34	3	9	0

Intersects

1	5	2	11	8	34

3	2	9	34	8	0

Intersects

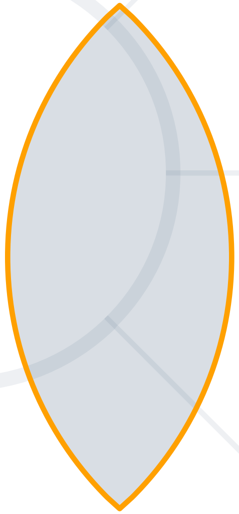


1	5	2	11	8	34

3	2	9	34	8	0

2	8	34

Intersects



1	5	2	11	8	34

3	2	9	34	8	0

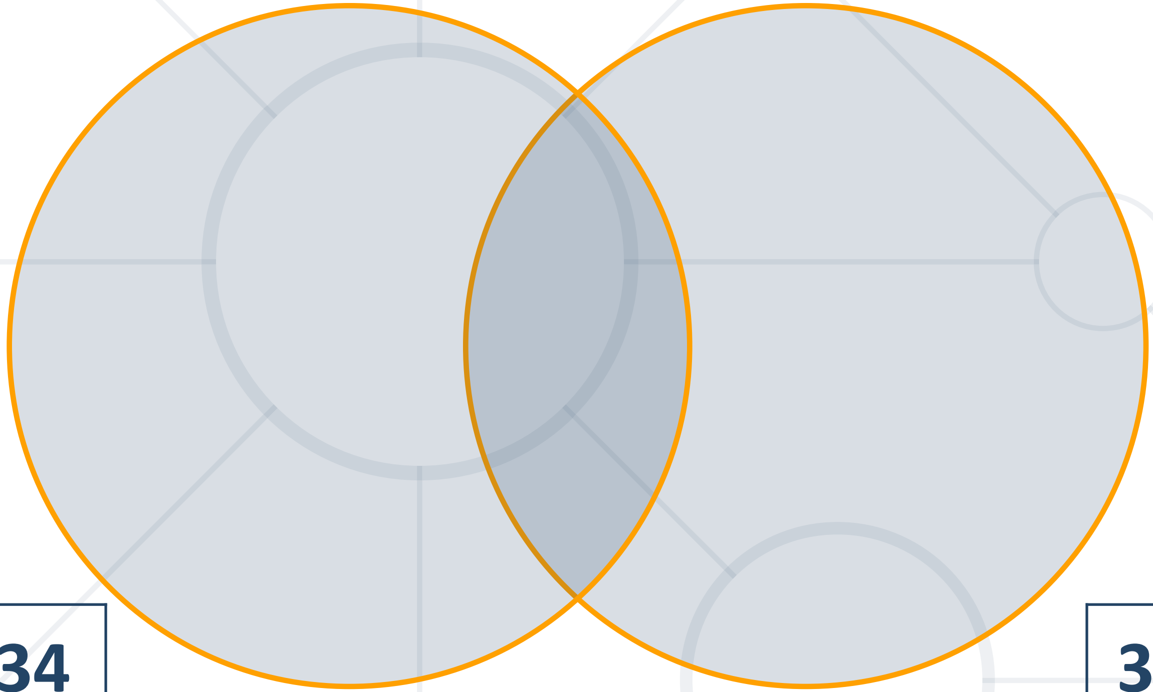
2	8	34

Except

1	5	2	11	8	34

3	2	9	34	8	0

Except

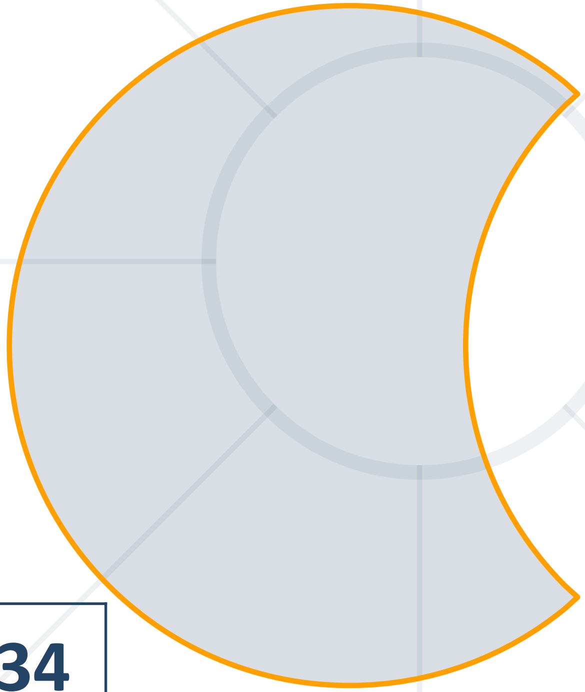


1	5	2	11	8	34

3	2	9	34	8	0

1	5	11

Except

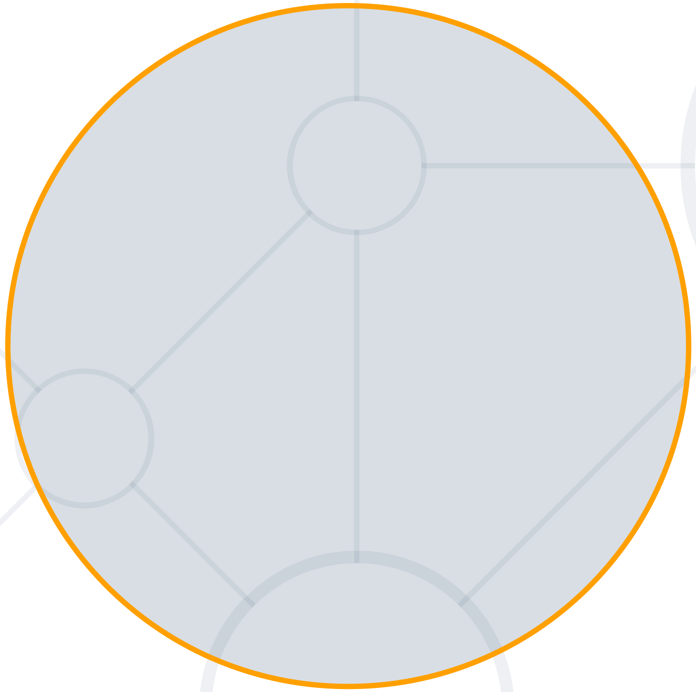


1	5	2	11	8	34

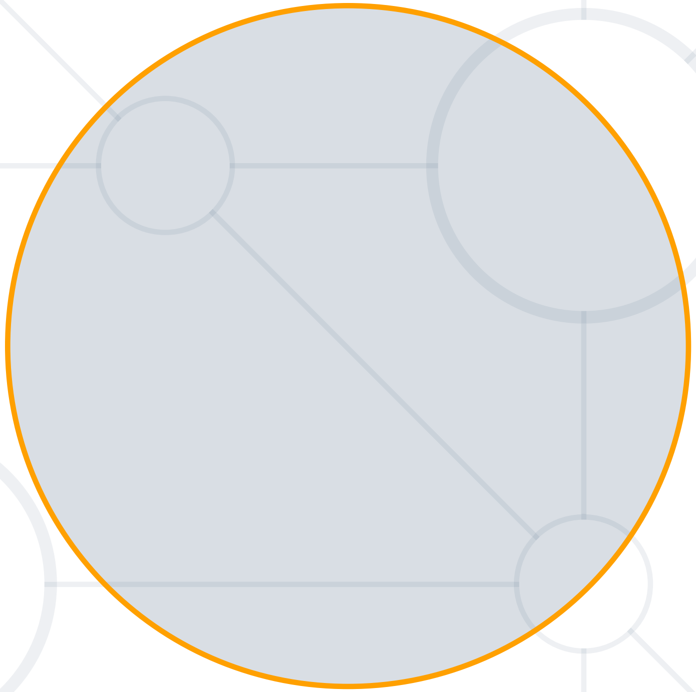
3	2	9	34	8	0

1	5	11

Symmetric Except

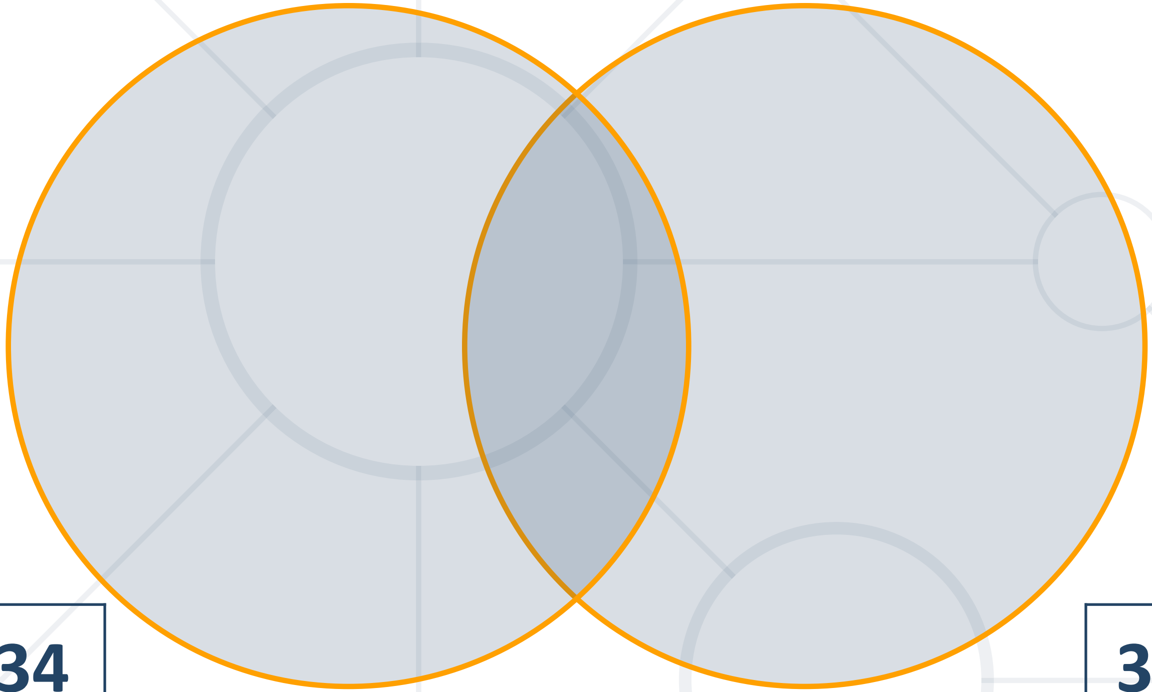


1	5	2	11	8	34



3	2	9	34	8	0

Symmetric Except

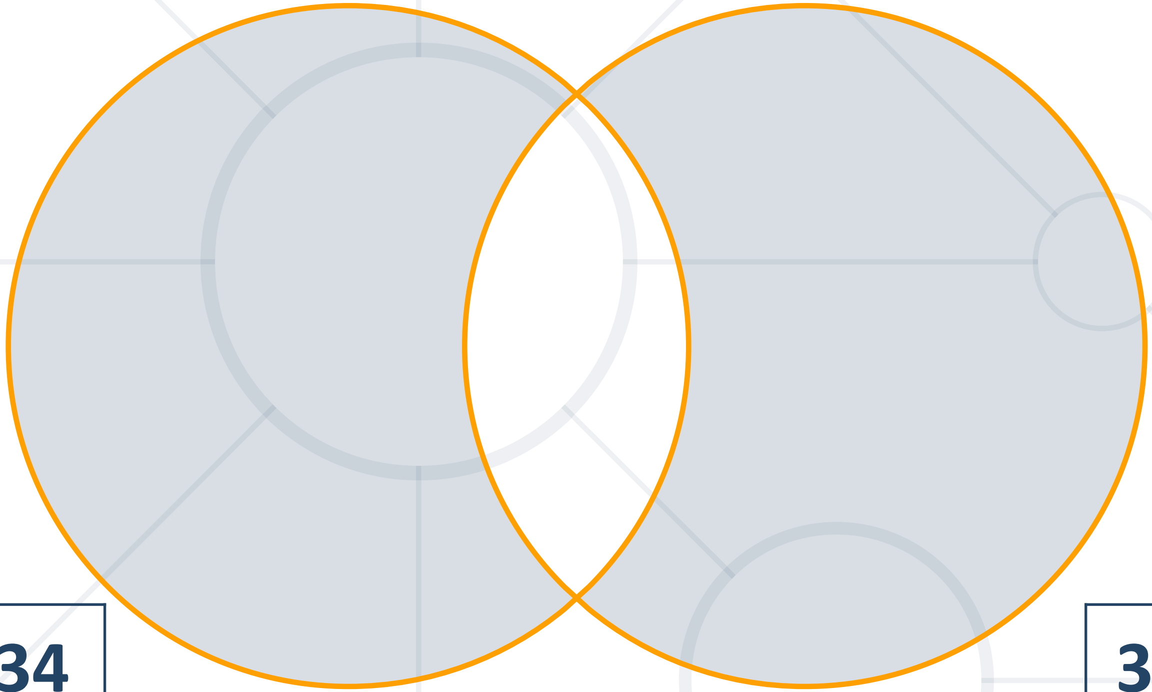


1	5	2	11	8	34

3	2	9	34	8	0

1	5	11	3	9	0

Symmetric Except



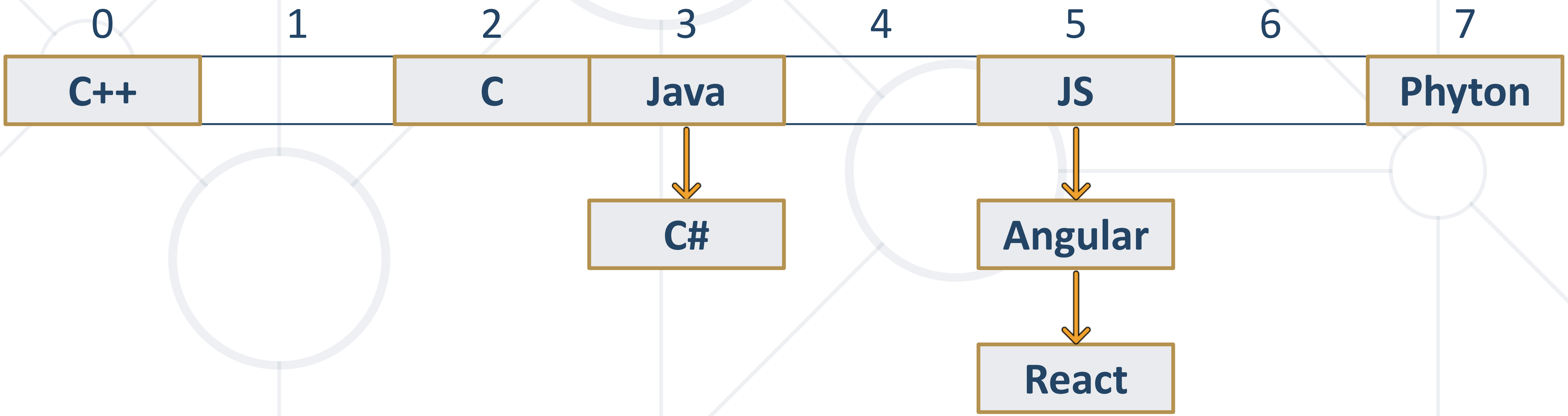
1	5	2	11	8	34

3	2	9	34	8	0

1	5	11	3	9	0

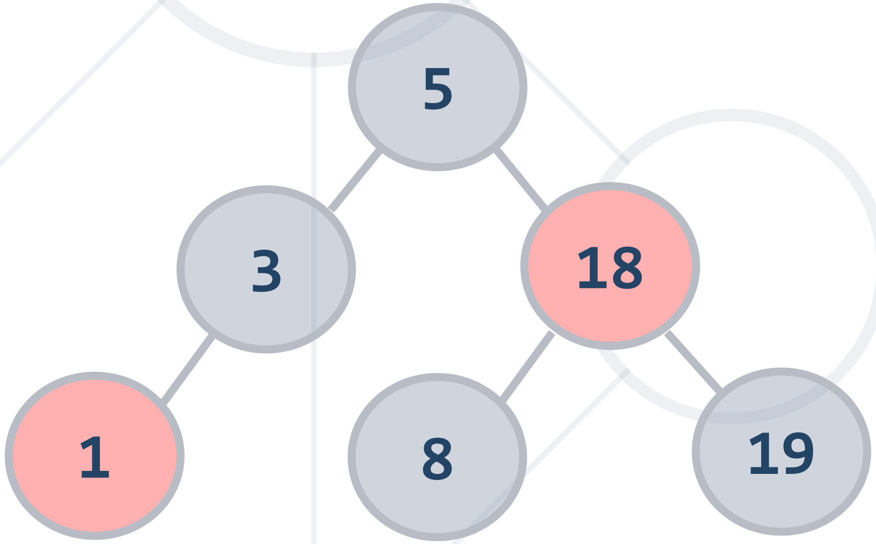
HashSet<T>

- **HashSet<T>** implements ADT **Set** by hash table
  - Elements are in no particular order
- All major operations are fast: **Add** / **Delete** / **Contains**



SortedSet<T>

- **SortedSet<T>** implements ADT **Set** by balanced search tree (red-black tree)
  - Elements are sorted in increasing order



Sets – Quiz

TIME'S

- For given sets - {1, 2, 3, 4, 5} and {3, 4, 5, 6, 7}, what is the operation that will give us the following result: {1, 2, 6, 7}
  - Union
  - Intersects
  - Except
  - SymmetricExcept

Sets – Answer

- For given sets - {1, 2, 3, 4, 5} and {3, 4, 5, 6, 7}, what is the operation that will give us the following result: {1, 2, 6, 7}
  - Union
  - Intersects
  - Except
  - SymmetricExcept ✔

Dictionaries

Definition and Operations

The Dictionary (Map) ADT

- The abstract data type (ADT) "**dictionary**" maps key to values
  - Also known as "**map**" or "**associative array**"
  - Holds a set of **{key, value} pairs**
- Many implementations
  - Hash table, balanced tree, list, array, ...

key	value
`John Smith`	`+1-555-8976`
`Sam Doe`	`+1-555-5030`

ADT Map – Example

- Sample dictionary:

Key	Value
C#	Modern general-purpose object-oriented programming language
PHP	Popular server-side scripting language for Web development
compiler	Software that transforms a computer program to executable machine code
…	…

Dictionary <Key, Value>

- Major operations:
  - **Add(key, value)** – adds an element by key + value
  - **Remove(key)** – removes a value by key
  - **this[key] = value** – add / replace element by key
  - **this[key]** – returns the value by key
  - **Keys** – returns a collection of all keys (in order of entry)
  - **Values** – returns a collection of all values (in order of entry)

Dictionary<Key, Value> (2)

- Major operations:
  - **ContainsKey(key)** – checks if given key exists in the dictionary
  - **ContainsValue(value)** – checks whether the dictionary contains given value
    - Warning: slow operation – **O(n)**
  - **TryGetValue(key, outvalue)**
    - if the key is found, returns it in the value
    - otherwise returns false

SortedDictionary<Key, Value>

- **SortedDictionary<Key, Value>** implements the ADT "dictionary" as self-balancing search tree
  - Elements are arranged in the tree ordered by key
  - Traversing the tree returns the elements in increasing order
  - **Add** / **Find** / **Delete** perform **log N** operations
- Use **SortedDictionary<Key, Value>** when you need the elements sorted by key
  - Otherwise use **Dictionary<Key, Value>** – it has better performance

Dictionaries – Quiz

TIME'S

- Which built-in implementation of **IDictionary<Key, Value>** sorts the items by value?
  - Dictionary<Key, Value>
  - SortedDictionary<Key, Value>
  - None

Dictionaries – Answer

- Which built-in implementation of **IDictionary<Key, Value>** sorts the items by value?
  - Dictionary<Key, Value>
  - SortedDictionary<Key, Value>
  - None

Hash Tables – Quiz

TIME'S

- Which is the main reason to use a hash table instead of a red-black BST?
  - Supports more operations efficiently
  - Better worst-case performance guarantee
  - Better performance in practice on typical inputs

Hash Tables – Answer

- Which is the main reason to use a hash table instead of a red-black BST?
  - Supports more operations efficiently
  - Better worst-case performance guarantee
  - Better performance in practice on typical inputs ✔

Comparing Keys

Using Custom Key Classes

Comparasion Methods

- Dictionary<Key, Value> relies on
  - Object.Equals() – for comparing the keys
  - Object.GetHashCode() – for calculating the hash codes of the keys
- SortedDictionary<Key, Value> relies on IComparable<Key> for ordering the keys

Implementing Equals() and HashCode()

```
public class Point {
  public int x, y;
  public override bool Equals(Object obj) {
     if (!obj is Point) || (obj == null) return false;
     Point p = (Point) obj;
     return (x == p.x) && (y == p.y);
  }

  public int GetHashCode() {
      return (x << 16 | y >> 16) ^ y;
  }
}
```

Implementing IComparable<T>

```
public class Point : IComparable<Point> {
  public int x, y;

  public int CompareTo(Point other) {
    if (x != other.x) {
      return this.X.CompareTo(other.x);
    }
    else {
      return this.y.CompareTo(other.y);
    }
  }
}
```

Summary

- **Hash-tables** map keys to values
  - Rely on hash-functions to distribute the keys in the table
  - Collisions needs resolution algorithm (e.g., chaining)
  - Very fast add / find / delete – **O(1)**
- **Sets** hold a group of elements
- **Dictionaries** map key to value

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