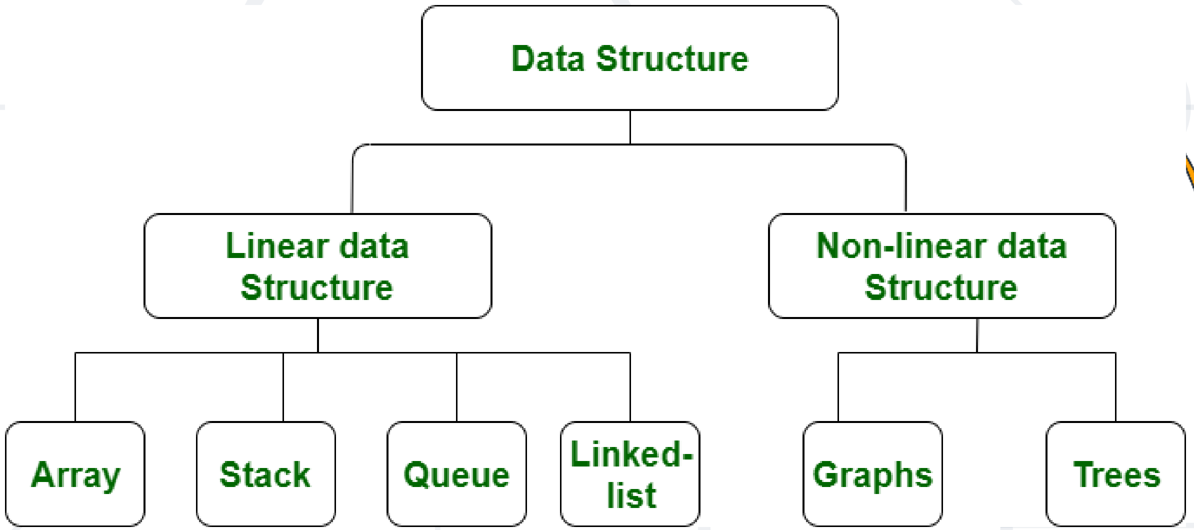
Intro to Data Structures

Data, Data Structures, Hash Tables



SoftUni Team

Technical Trainers

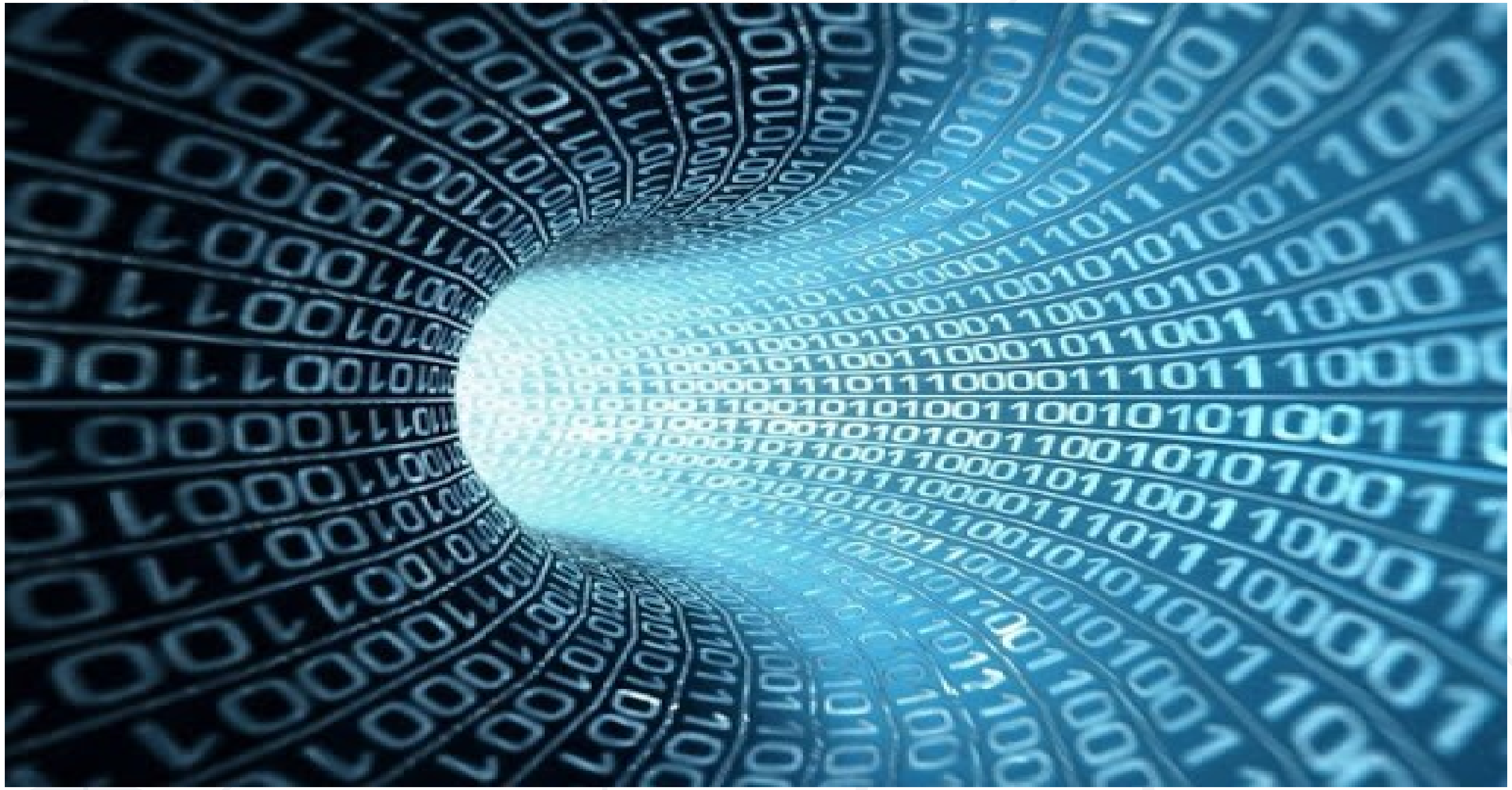
Software University

SoftUni

Software University

https://softuni.bg

Table of Contents



How is Data Stored in the Memory?

Data in Computing

Software University

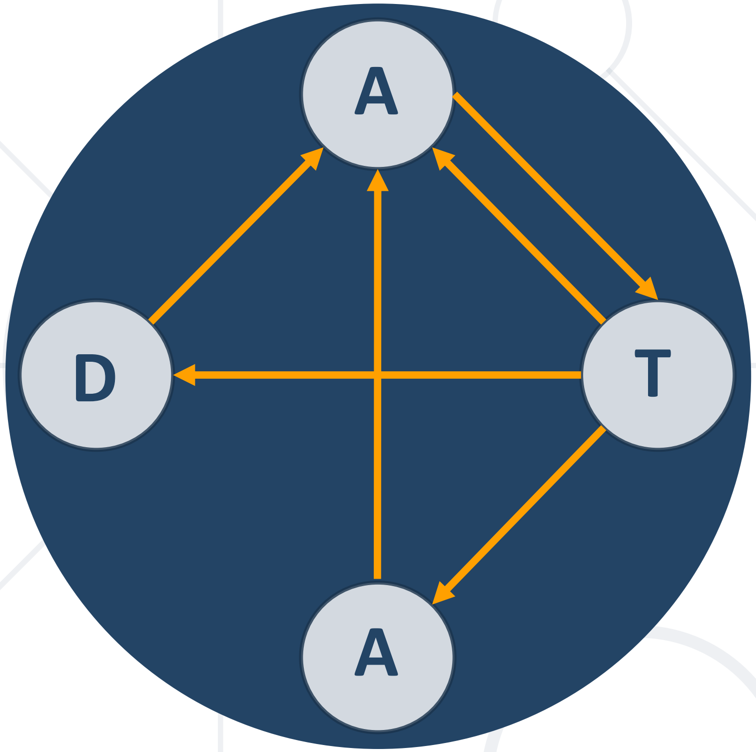
- Set of **symbols** gathered and translated for **some purpose**
- Simplified – bits of information stored in memory
  - If those bits remain **unused,** they don't do anything
- Example:

Binary Data	Translation
100 0001	65
100 0001	A

Data in Computing

- The way we **read** the data **retrieves the information** of the bits in different ways
  - However, bits have only **0** or **1** as values
- Example:

Type	Binary Data	Translation
Integer	0000 0100 0001	65
Character	0000 0100 0001	'A'
Double	0000 0100 0001	65.0
Instruction Code	0000 0100 0001	Store 65
Color	0000 0100 0001	



Overview

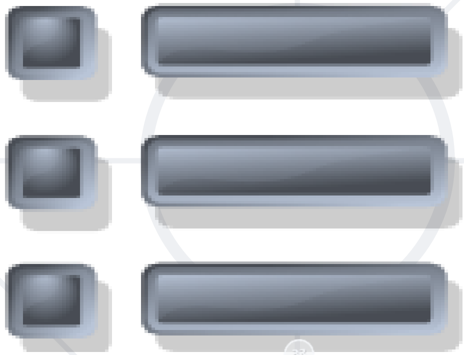
Data Structures

- **Data structure** – an **object** which takes responsibility for data **organization**, **storage**, **management** in **effective** manner
- Storing items **requires memory consumption**:

Data Structure	Size
int	= 4 bytes
float	= 4 bytes
long	= 8 bytes
int[]	≈ (Array length) * 4 bytes
List<double>	≈ (List size) * 8 bytes
Dictionary<int, int[]>	≈ (Dictionary size) * Entry bytes

Basic Data Structures

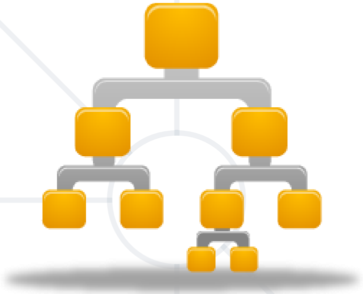
- Linear structures
  - Lists: fixed size and variable size sequences
  - Stacks: LIFO (**L**ast **I**n **F**irst **O**ut) structures
  - Queues: FIFO (**F**irst **I**n **F**irst **O**ut) structures

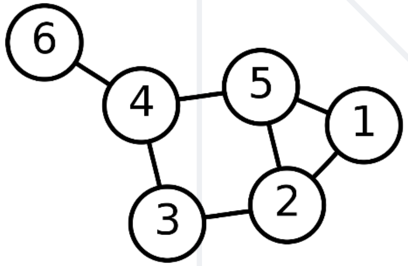


- Trees and tree-like structures
  - Binary, ordered search trees, balanced trees, etc.
- Dictionaries (maps, associative arrays)
  - Hold pairs (key → value)
  - Hash tables: use hash functions to search / insert

Basic Data Structures (2)

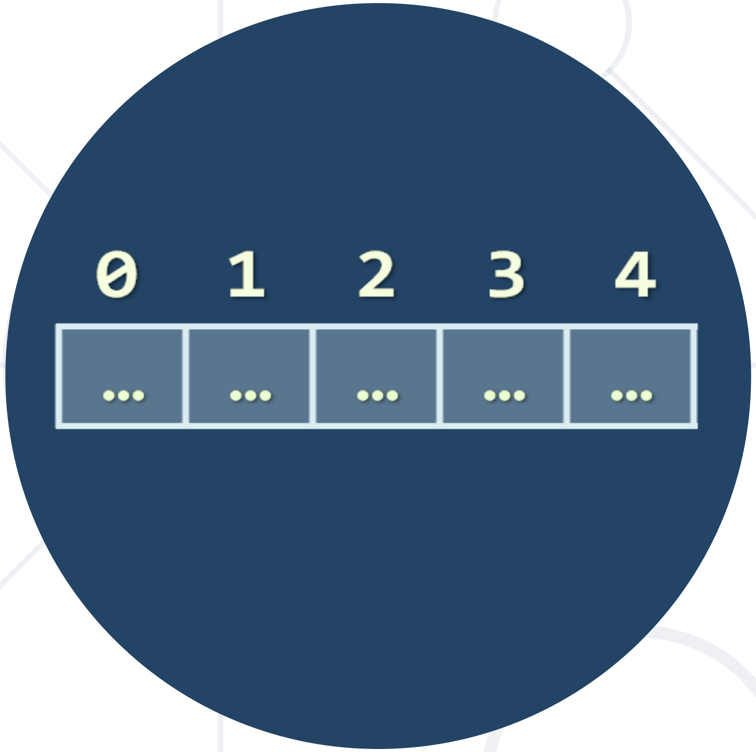
- **Sets**, **multi-sets** and **bags**
  - Set – collection of unique elements
  - Bag – collection of non-unique elements
- **Ordered sets** and **dictionaries**
- **Priority queues** / heaps
- **Special tree structures**
  - Suffix tree, interval tree, index tree, trie, rope, …
- **Graphs**
  - Directed / undirected, weighted / unweighted, connected / non-connected, cyclic / acyclic, …





Abstract Data Types (ADT)

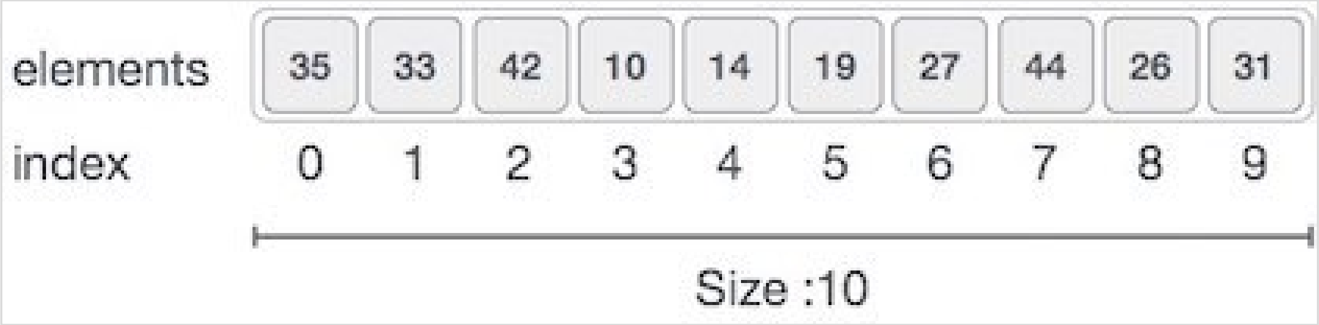
- **An Abstract Data Type** (ADT) is:
  - A set of **definitions of operations**
    - Defines what we can do with the structure
- ADT can have several different **implementations**
  - Different implementations can have different **efficiency**, **inner logic** and **resource needs**



Arrays and Lists

Array Data Structure

- **Arrays**
  - Very **lightweight**
  - Have a **fixed size**
  - Usually **built into the language**
- Many collections are implemented by using arrays
  - **List<T>** in C#
  - **Queue<T>** in C#
  - **Stack<T>** in C#



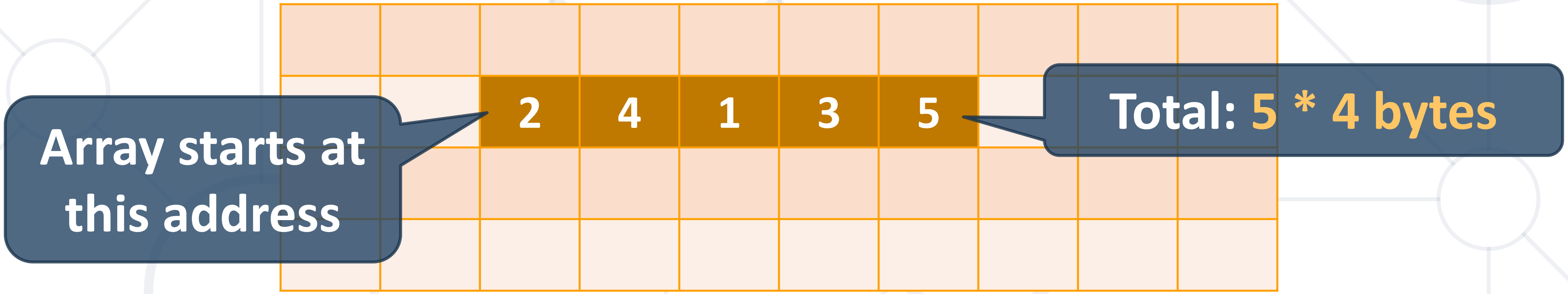
Why Arrays Are Fast?

- Arrays use a **single block of memory**

```
int[] array = { 2, 4, 1, 3, 5 };
```

int size is 4 bytes

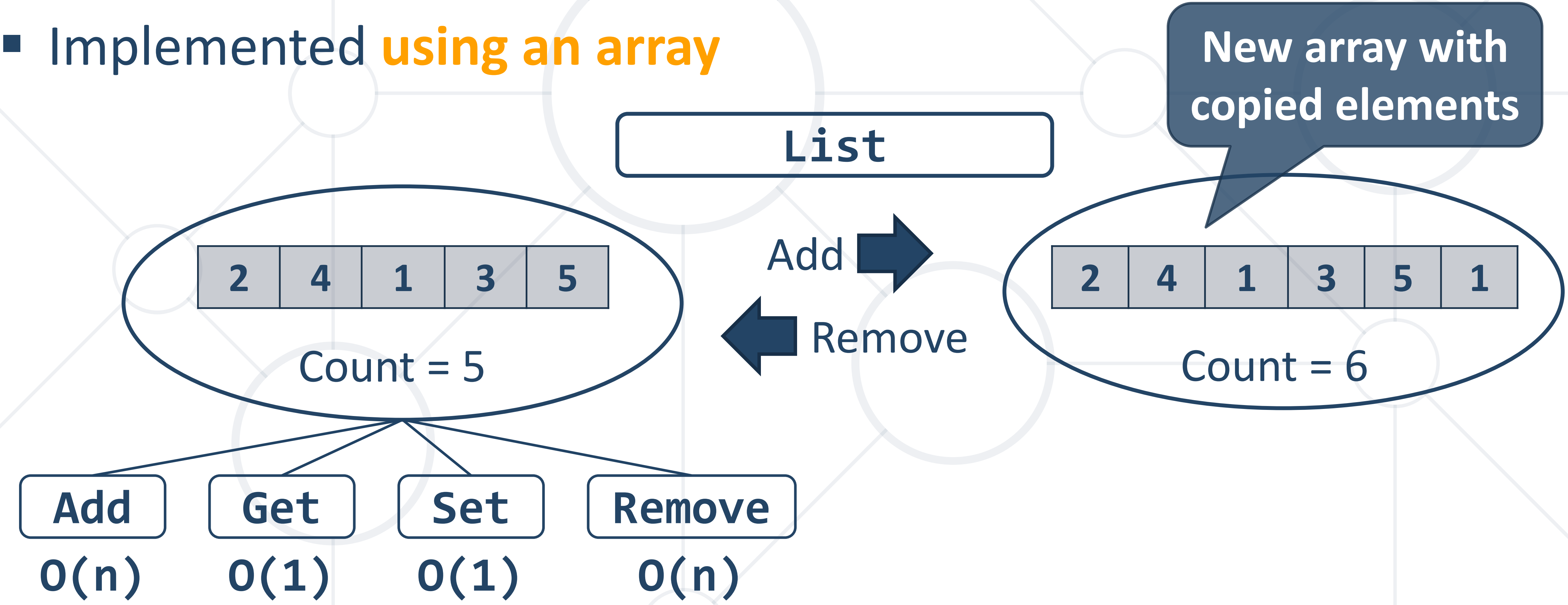
- Uses total of **array pointer + (N * element/pointer size)**



- **Array address** + **(element index** * **size)** = **element address**
- Arrays have a **fixed size** → to resize the array we **make a copy**

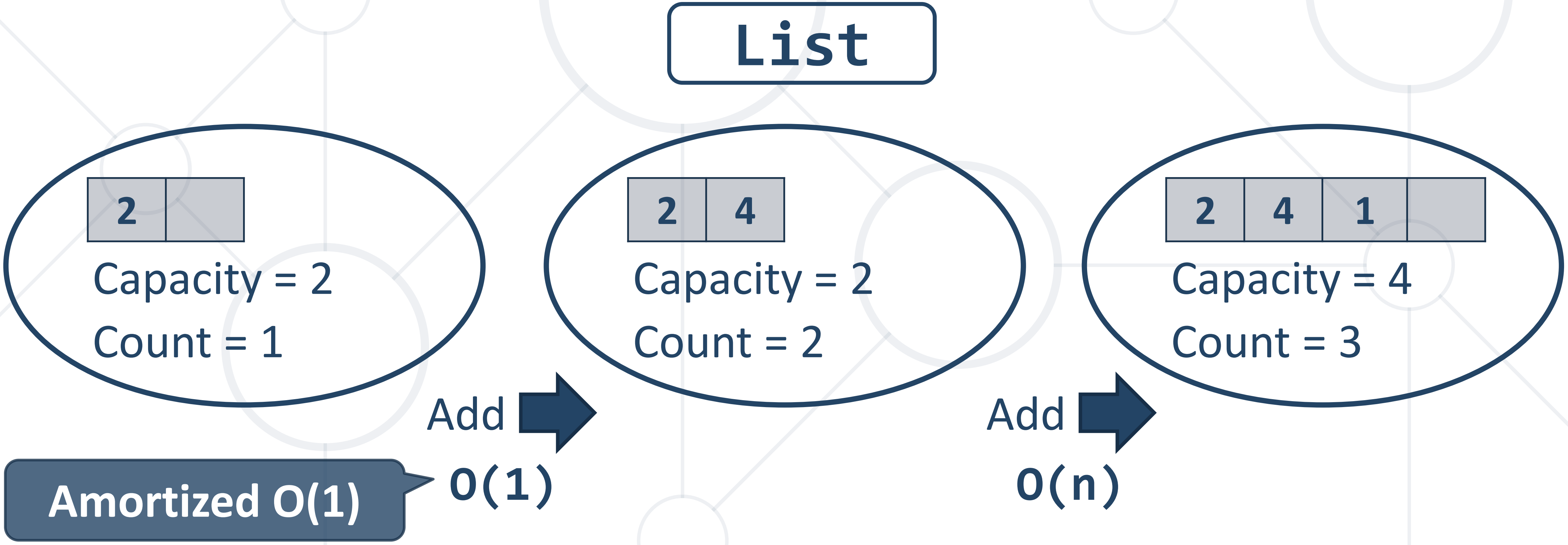
Dynamic Arrays (Lists): Resize +1

- **Dynamic (resizable) arrays** have a **variable size**
- Implemented **using an array**



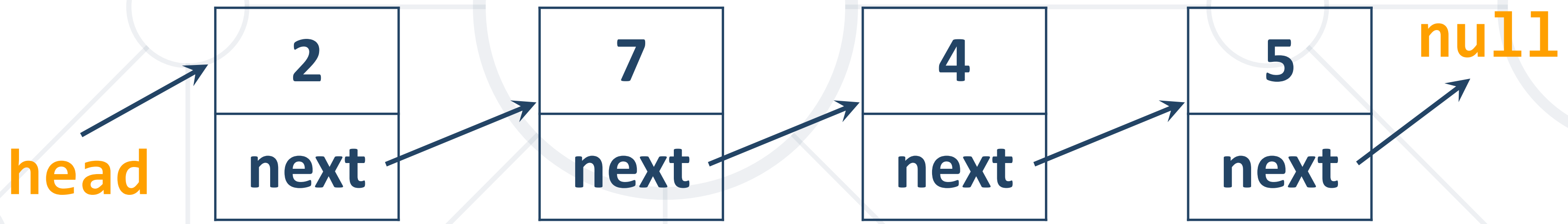
Dynamic Arrays (Lists): Resize *2 – Add O(1)

- Resizable arrays: **double** their **capacity** when needed
- Copying occurs **log(n)** times → n = 10^9, only ~30 copies

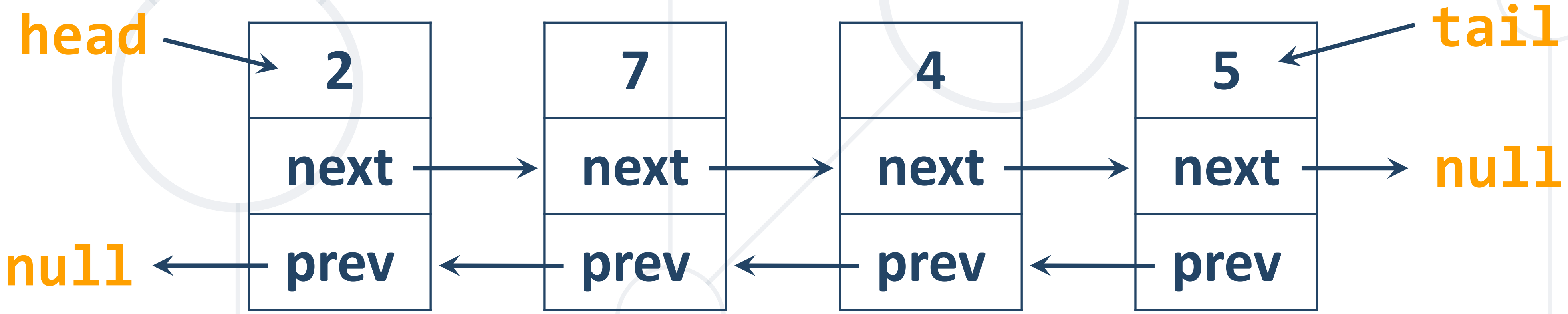


Linked List

- **Linked list** == dynamic (pointer-based) list implementation
- **Singly-linked list**: each item has **`value`** and **`next`**



- **Doubly-linked list**: each item has **`value`**, **`next`** and **`prev`**



Example: LinkedList<T>

```
static void Main()
{
    var list = new LinkedList<string>();
    list.AddFirst("First");
    list.AddLast("Last");
    list.AddAfter(list.First, "After First");
    list.AddBefore(list.Last, "Before Last");

    Console.WriteLine(String.Join(", ", list));

    // Result: First, After First, Before Last, Last
}
```

key	value
John Smith	+1-555-8976
Sam Doe	+1-555-5030
Sam Smith	+1-555-4542
John Doe	+1-555-3527

Dictionary<K, V>

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**

Example: Dictionary<K, V>

```
var studentGrades = new Dictionary<string, int>();
studentGrades.Add("Ivan", 4);
studentGrades.Add("Peter", 6);
studentGrades.Add("Maria", 6);
studentGrades.Add("George", 5);
int peterGrade = studentGrades["Peter"];
Console.WriteLine("Peter's grade: {0}", peterGrade);
  Console.WriteLine("Students and their grades:");
foreach (var pair in studentGrades)
  Console.WriteLine("{0} --> {1}", pair.Key, pair.Value);
```

- Source code of **Dictionary<TKey, TValue>**: https://github.com/microsoft/referencesource

SortedDictionary<TKey,TValue>

- **SortedDictionary<TKey,TValue>** 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<TKey,TValue>** when you need the elements **sorted by key** – based on **balanced search tree**
  - Otherwise use **Dictionary<TKey,TValue>** – it has better performance – based on **hash table**

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**

	`0`	`1`	`2`	`3`	`4`	`5`	`…`	`m-1`
T	…	…	…	…	…	…	…	…

`k.GetHashCode()`

Returns 32-bit integer

Adding to Hash Table

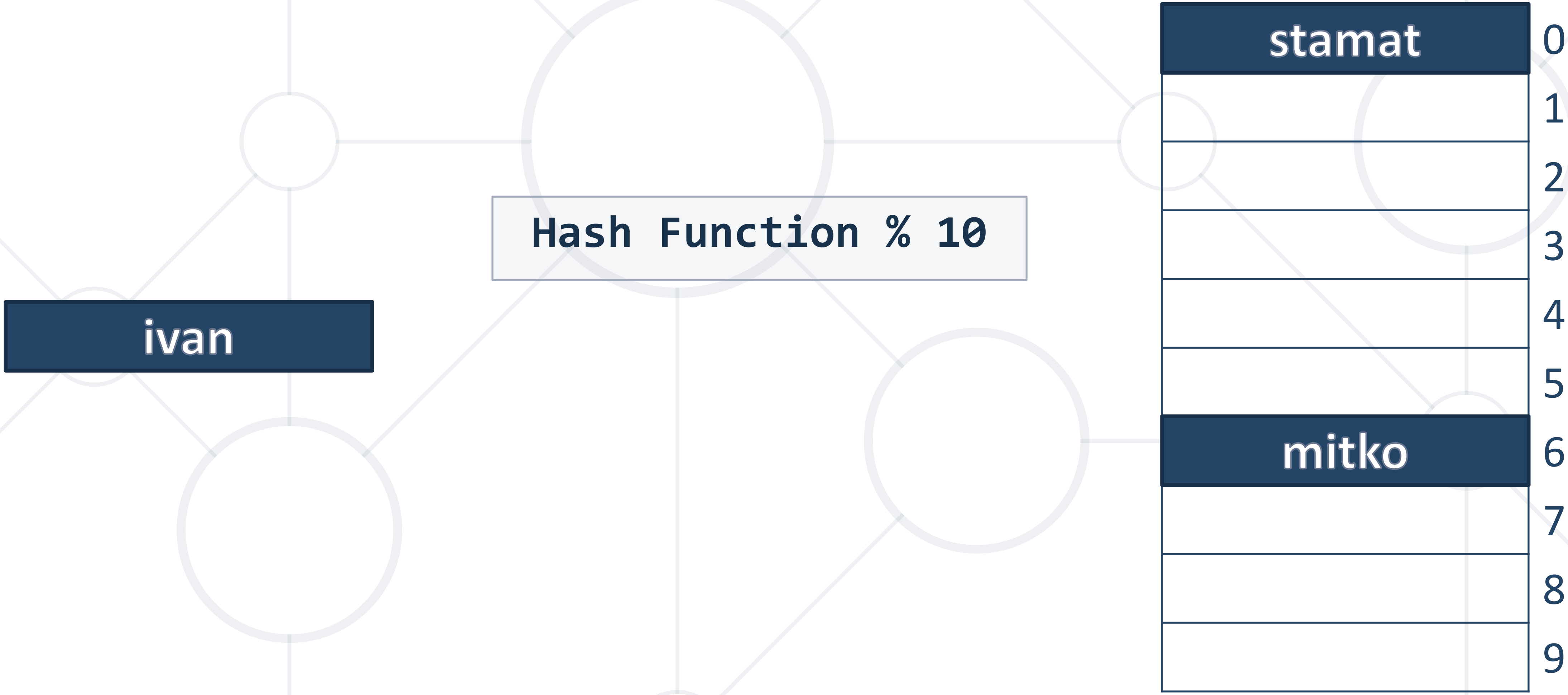
stamat

`Hash Function % 10`

	0
	1
	2
	3
	4
	5
	6
	7
	8
	9

Adding to Hash Table (2)

Adding to Hash Table (3)



Adding to Hash Table (4)

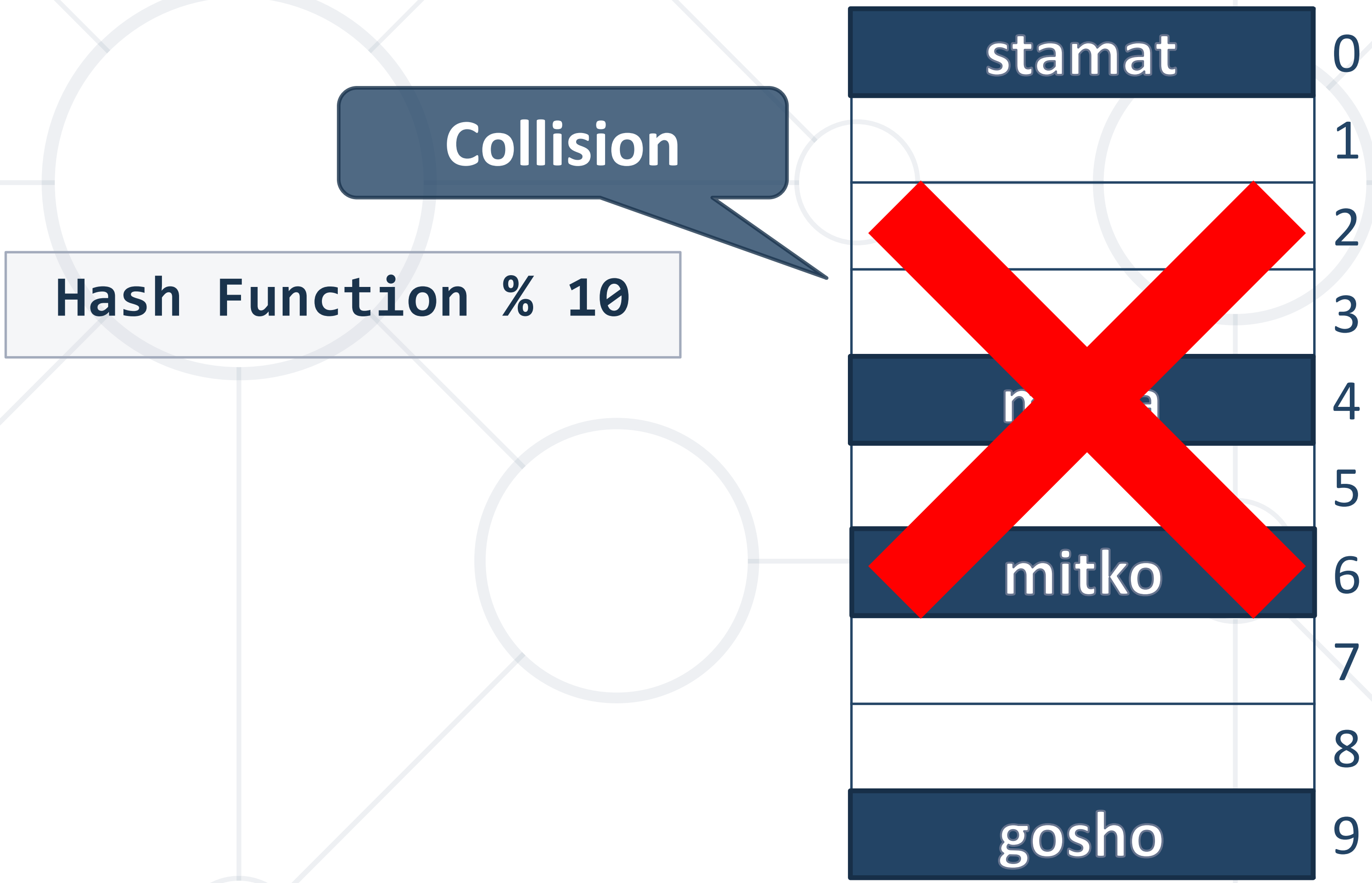
Adding to Hash Table (5)

maria

`Hash Function % 10`

stamat	0
	1
	2
	3
ivan	4
	5
mitko	6
	7
	8
gosho	9

Adding to Hash Table (6)



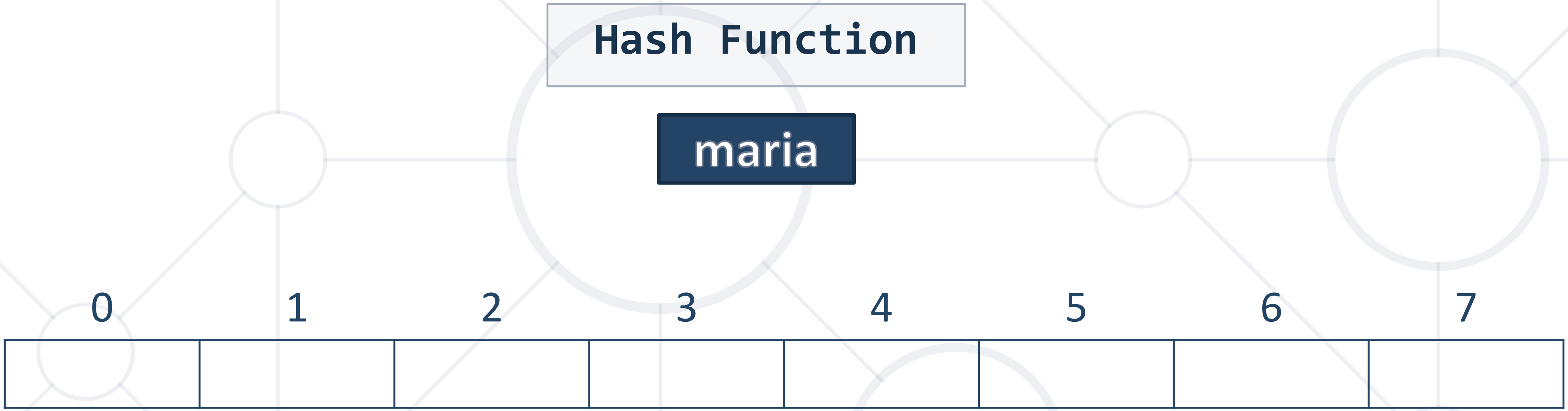
Collisions in a Hash Table

- **Collision** == **different keys** have the **same hash value**

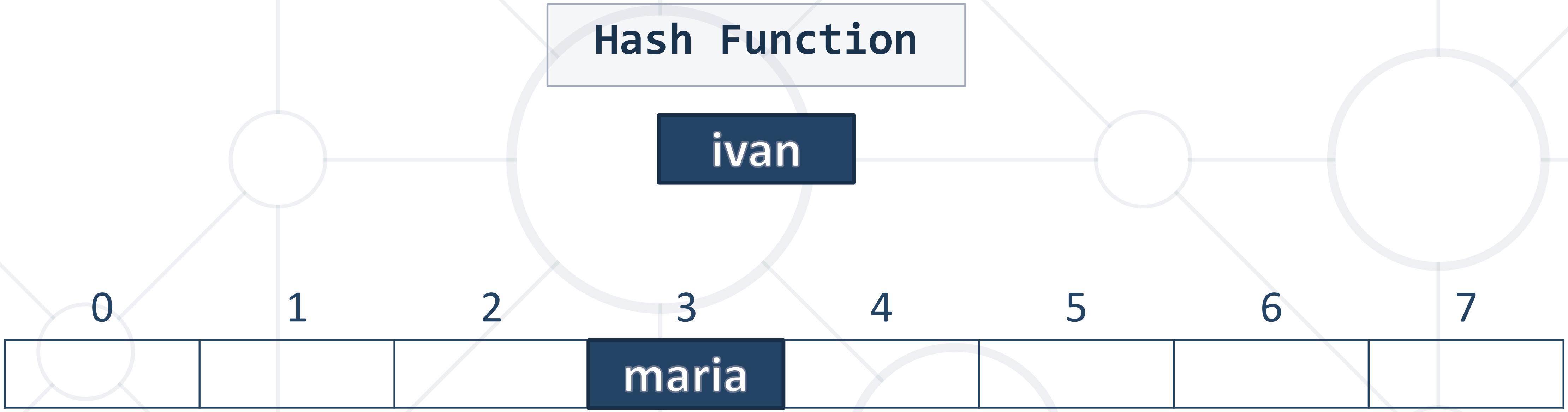
  $h(k_1) = h(k_2)$ for $k_1 \neq k_2$

- 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)
  - Many others

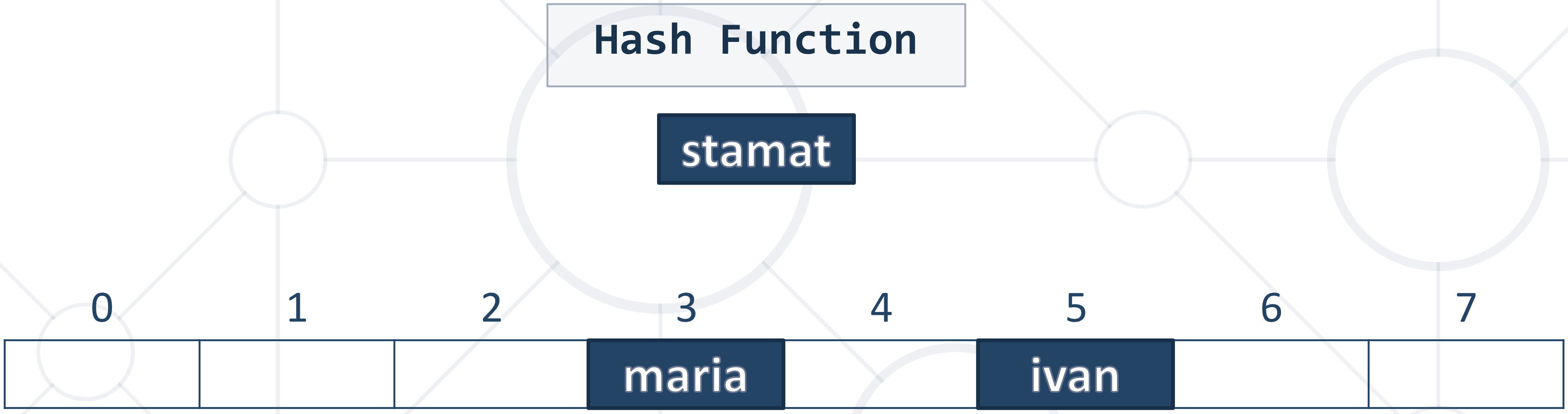
Collision Resolution: Chaining



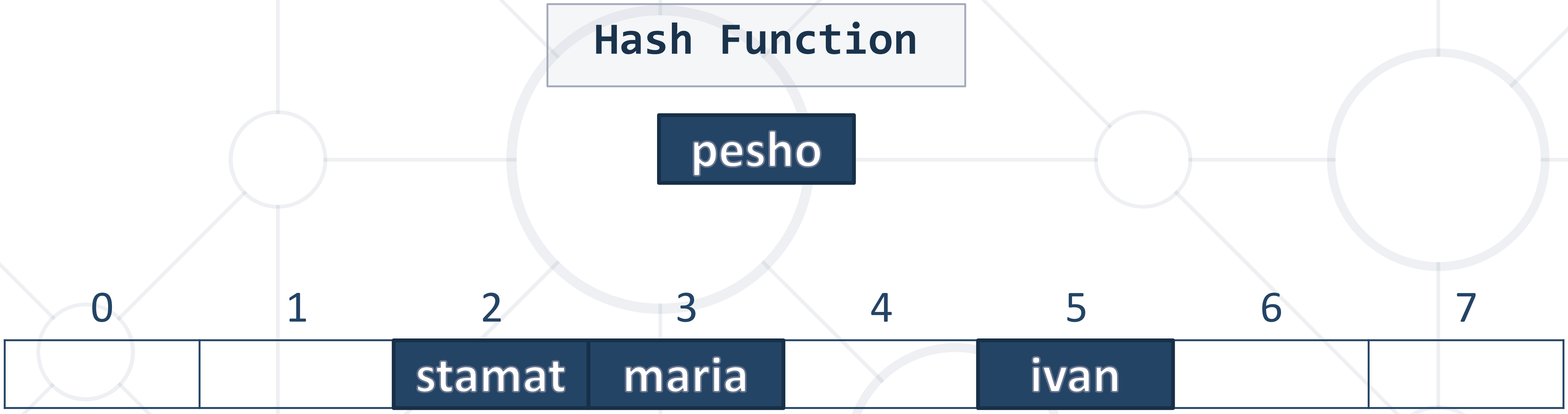
Collision Resolution: Chaining



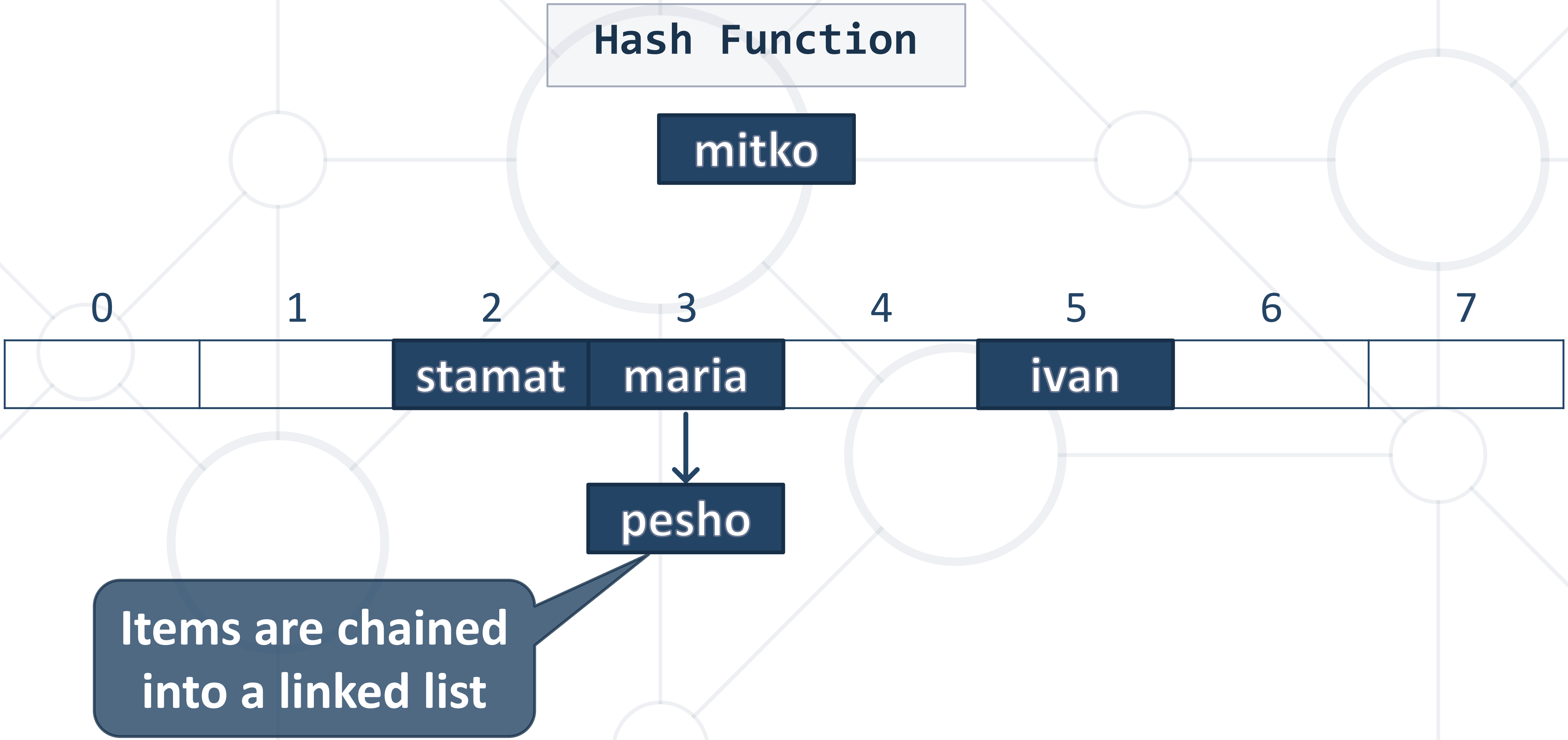
Collision Resolution: Chaining



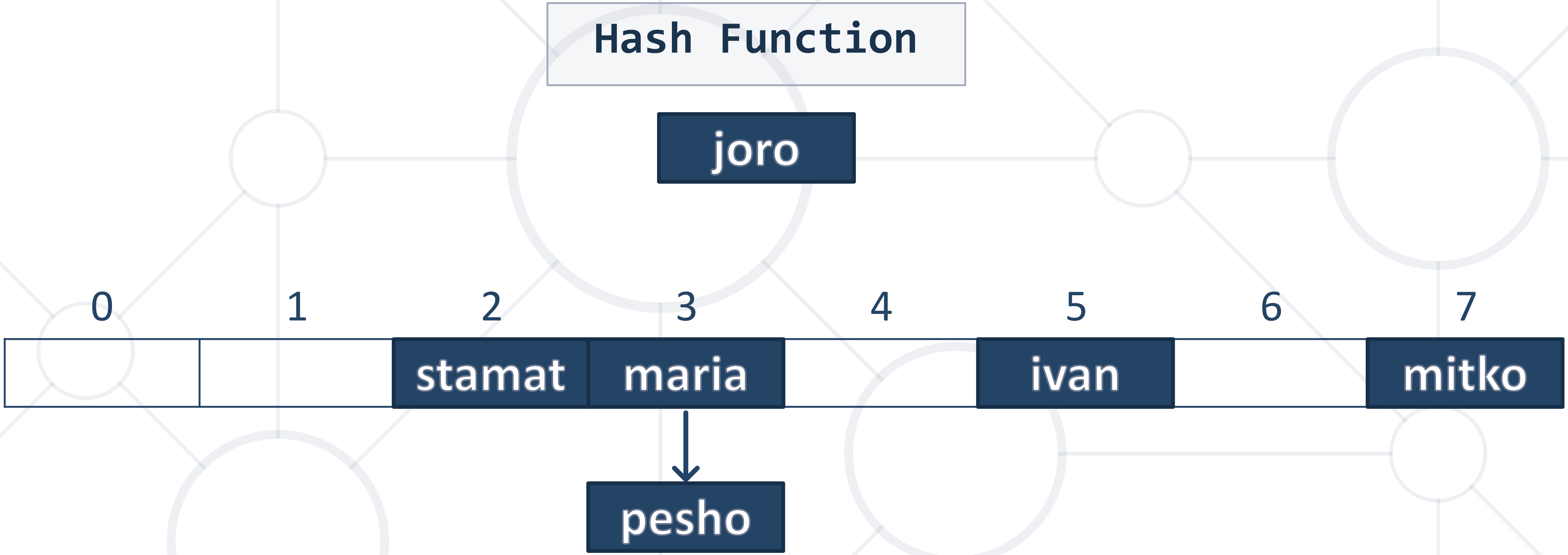
Collision Resolution: Chaining



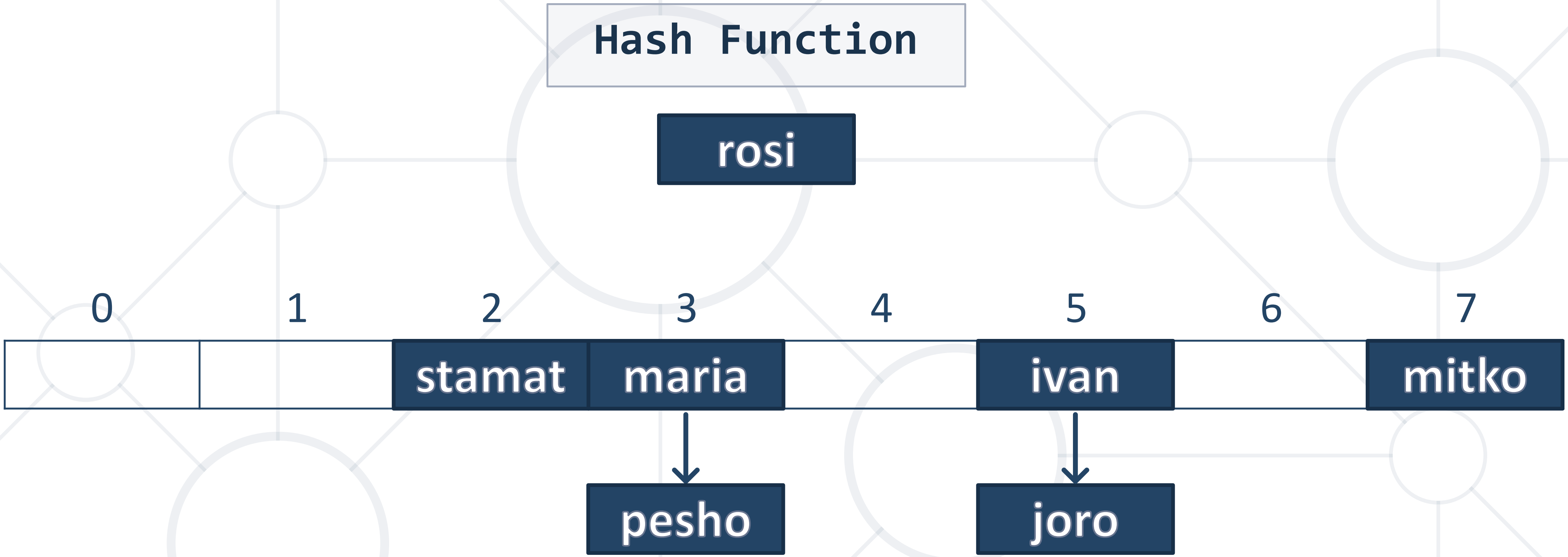
Collision Resolution: Chaining



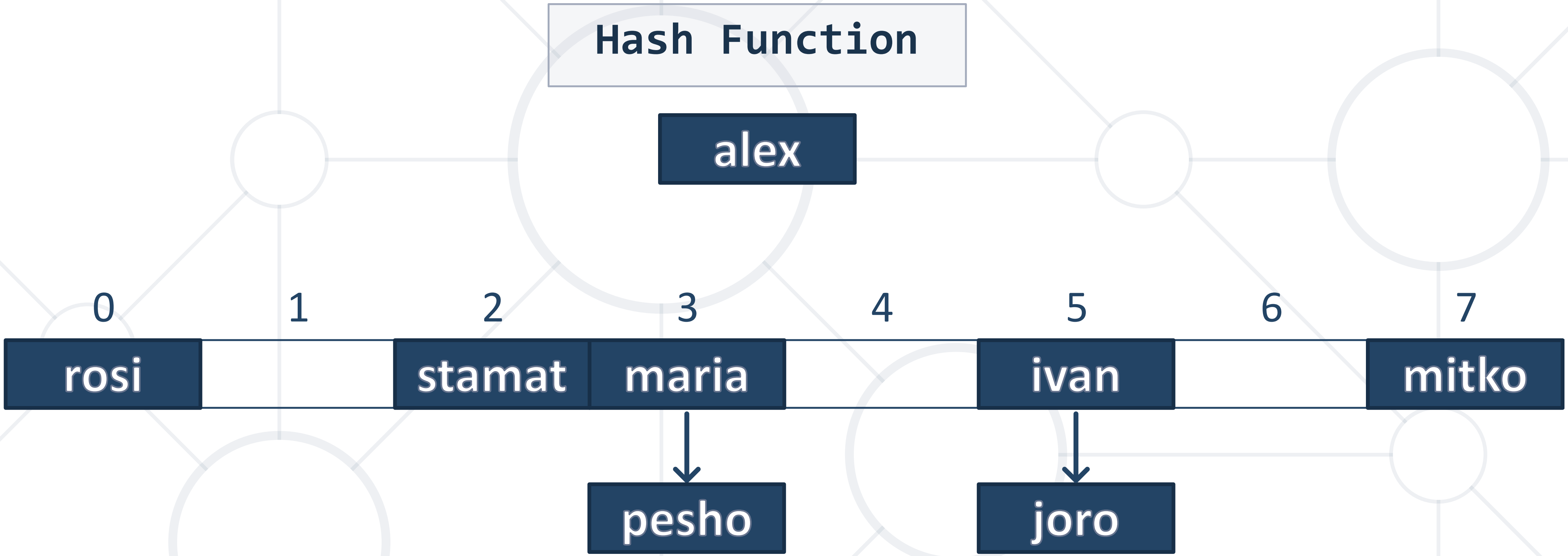
Collision Resolution: Chaining



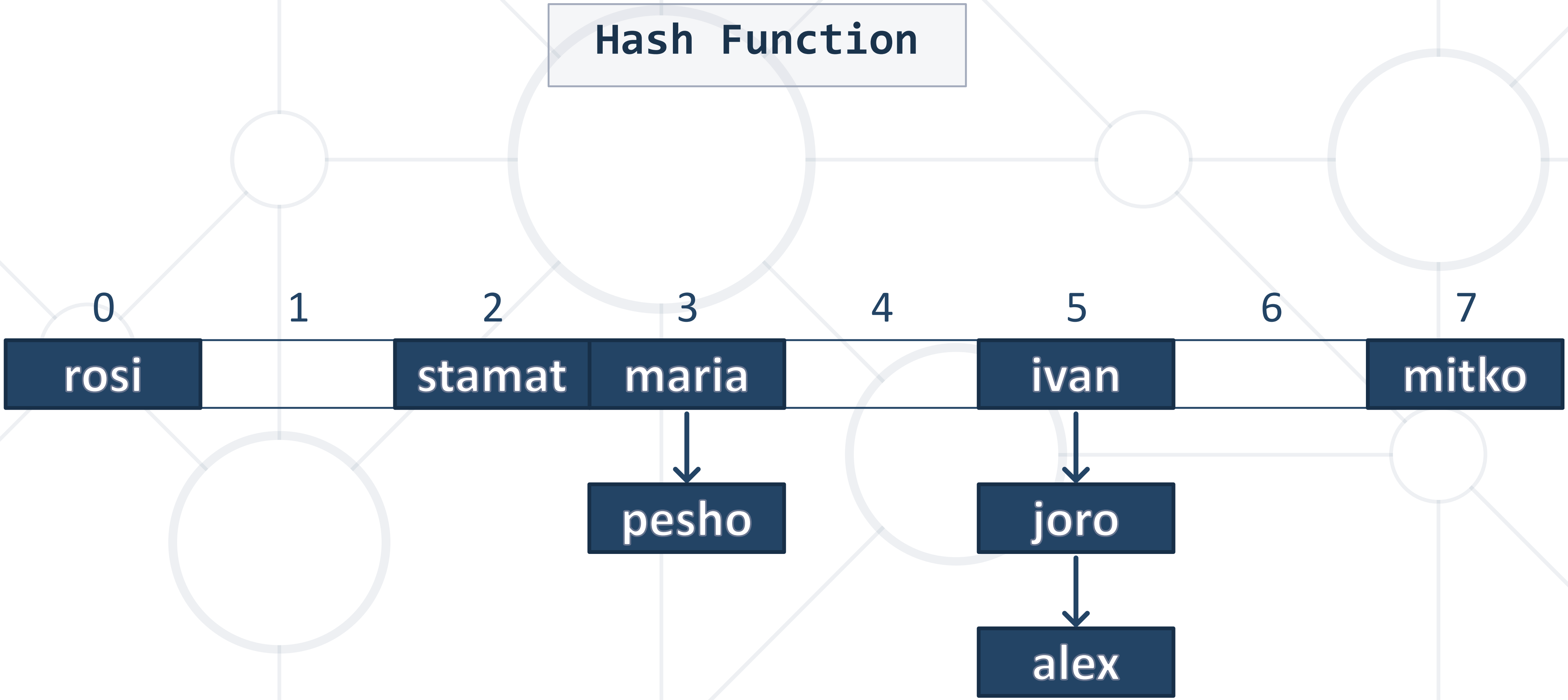
Collision Resolution: Chaining



Collision Resolution: Chaining



Collision Resolution: Chaining



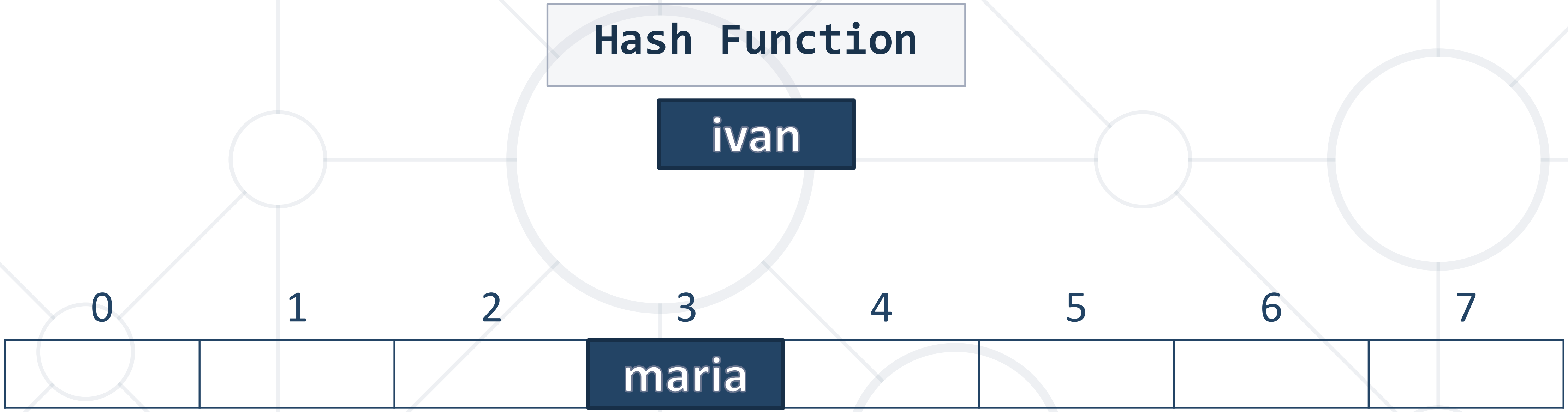
Collision Resolution: Linear Probing

Hash Function

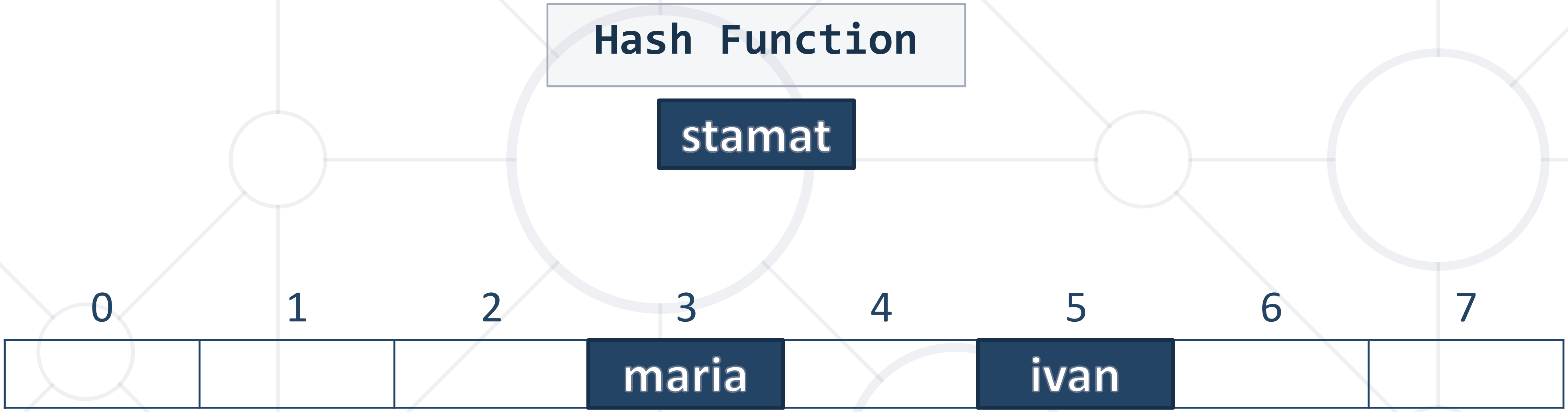
maria

0	1	2	3	4	5	6	7

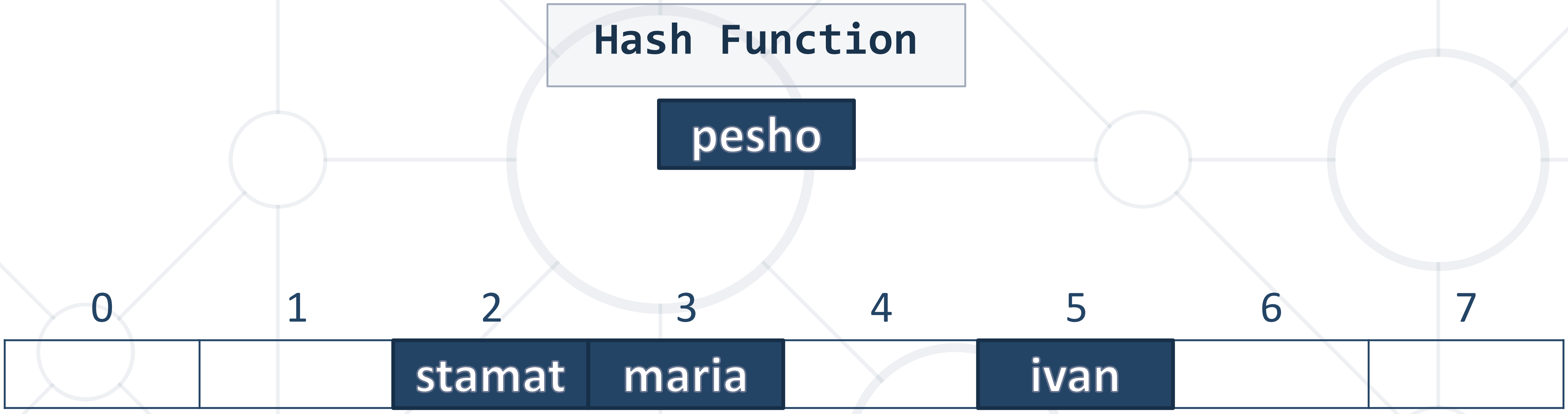
Collision Resolution: Linear Probing



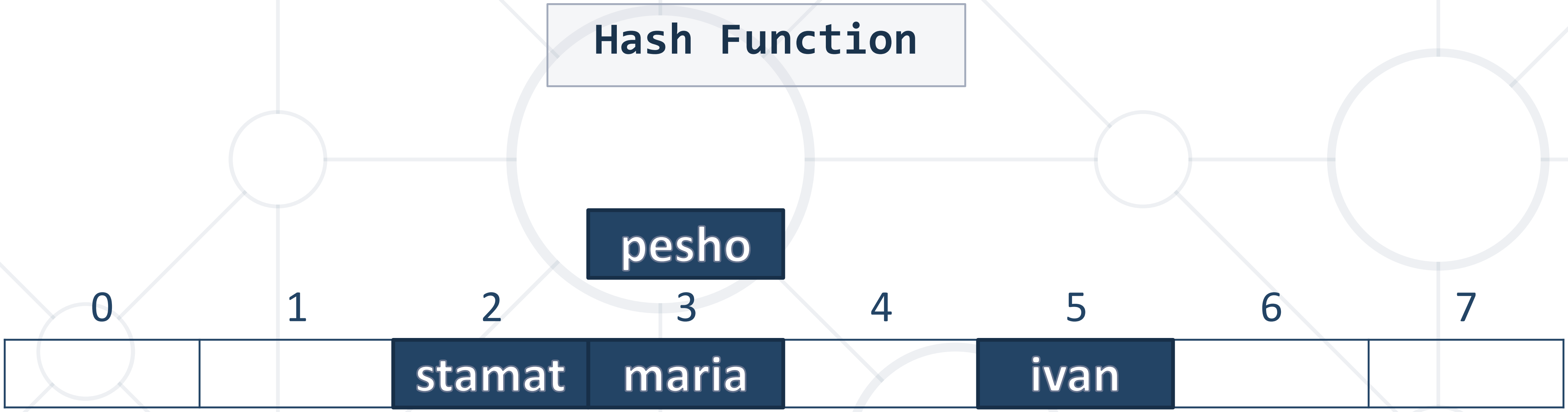
Collision Resolution: Linear Probing



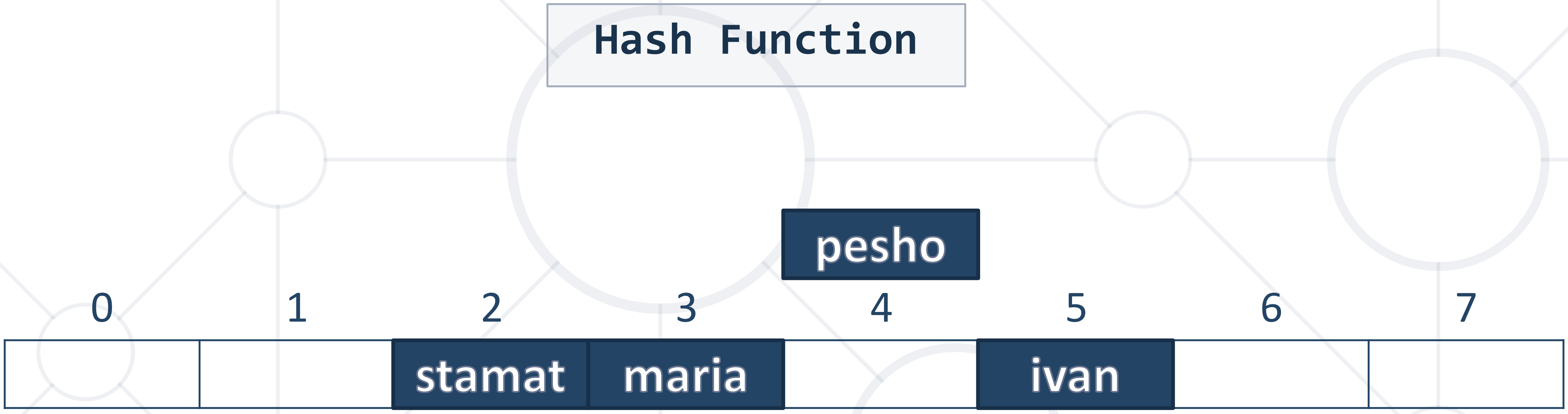
Collision Resolution: Linear Probing



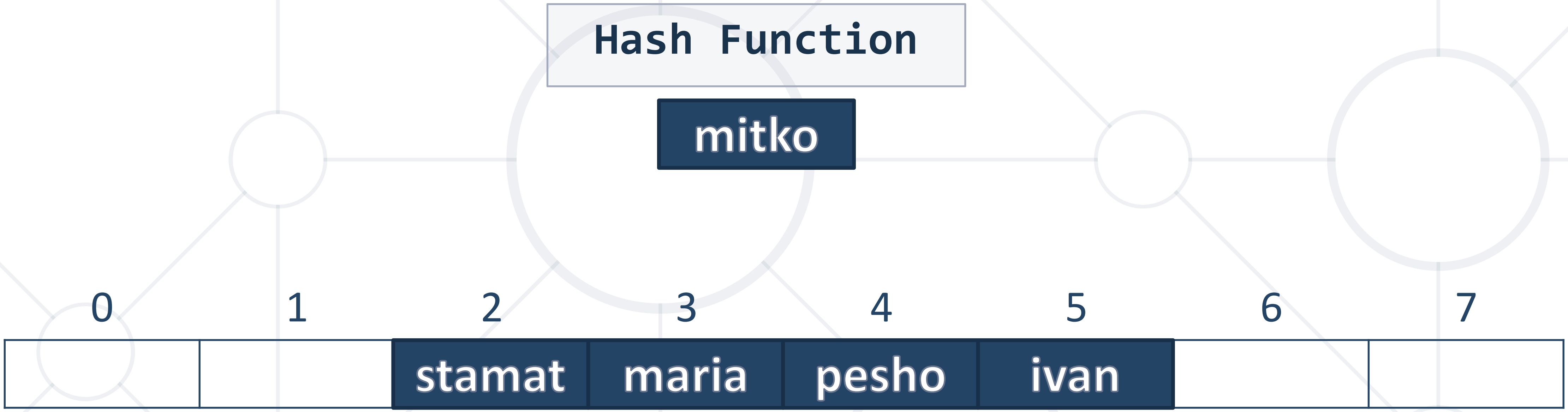
Collision Resolution: Linear Probing



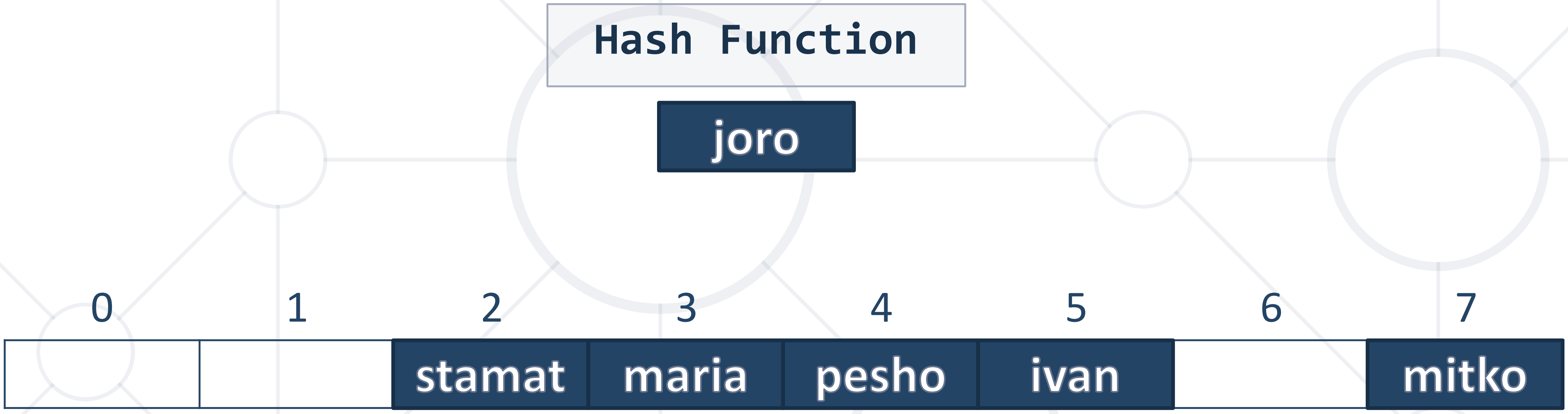
Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing



Collision Resolution: Linear Probing

Collision Resolution: Linear Probing

Hash Function

joro

0	1	2	3	4	5	6	7
		stamat	maria	pesho	ivan		mitko

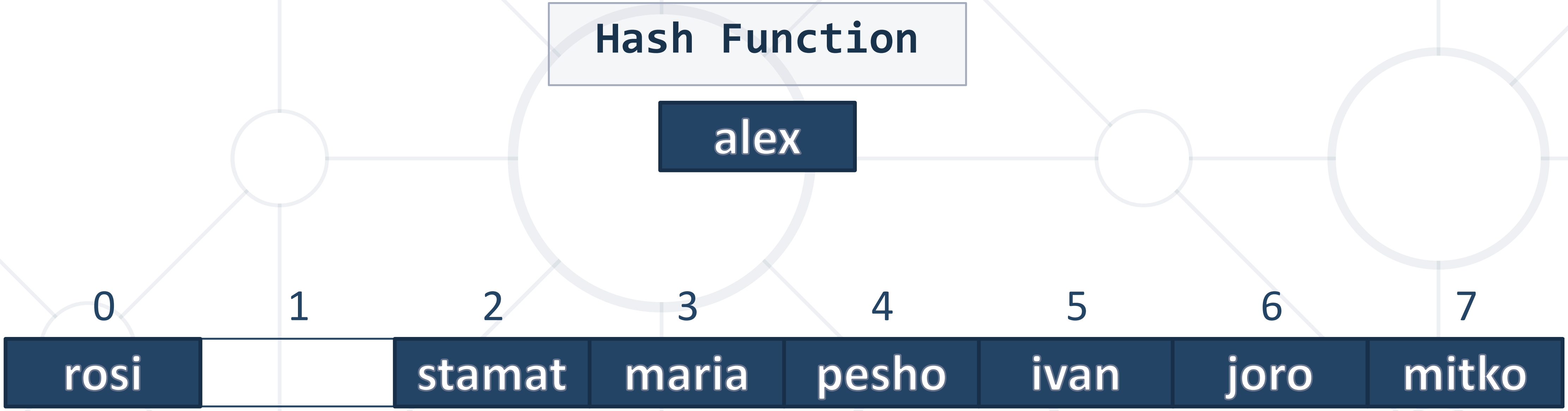
Collision Resolution: Linear Probing

Hash Function

rosi

0	1	2	3	4	5	6	7
		stamat	maria	pesho	ivan	joro	mitko

Collision Resolution: Linear Probing



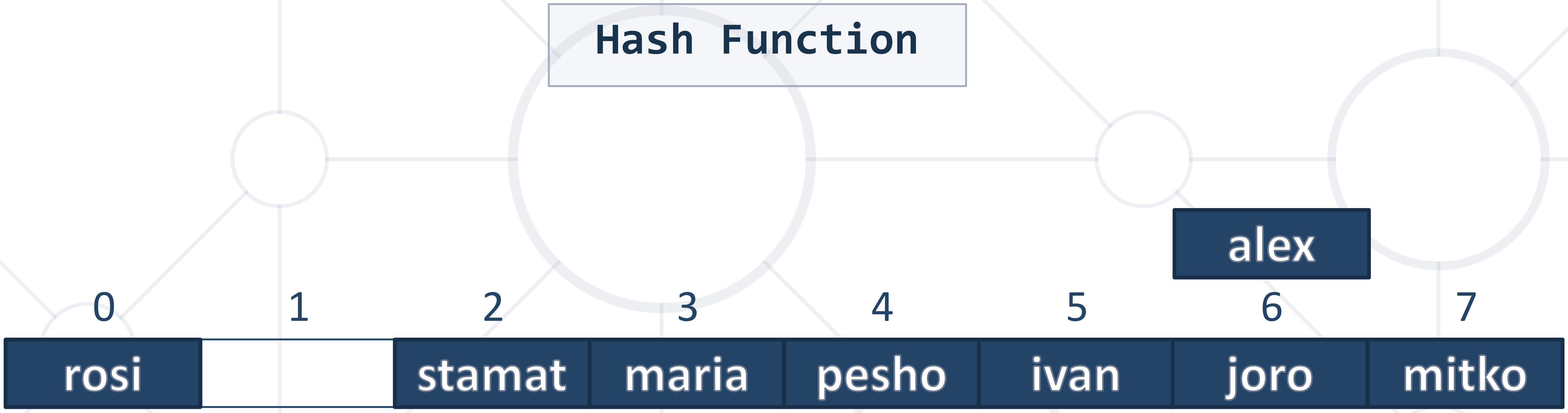
Collision Resolution: Linear Probing

Hash Function

alex

0	1	2	3	4	5	6	7
rosi		stamat	maria	pesho	ivan	joro	mitko

Collision Resolution: Linear Probing



Collision Resolution: Linear Probing

Hash Function

alex

0	1	2	3	4	5	6	7
rosi		stamat	maria	pesho	ivan	joro	mitko

Collision Resolution: Linear Probing

Hash Function

alex

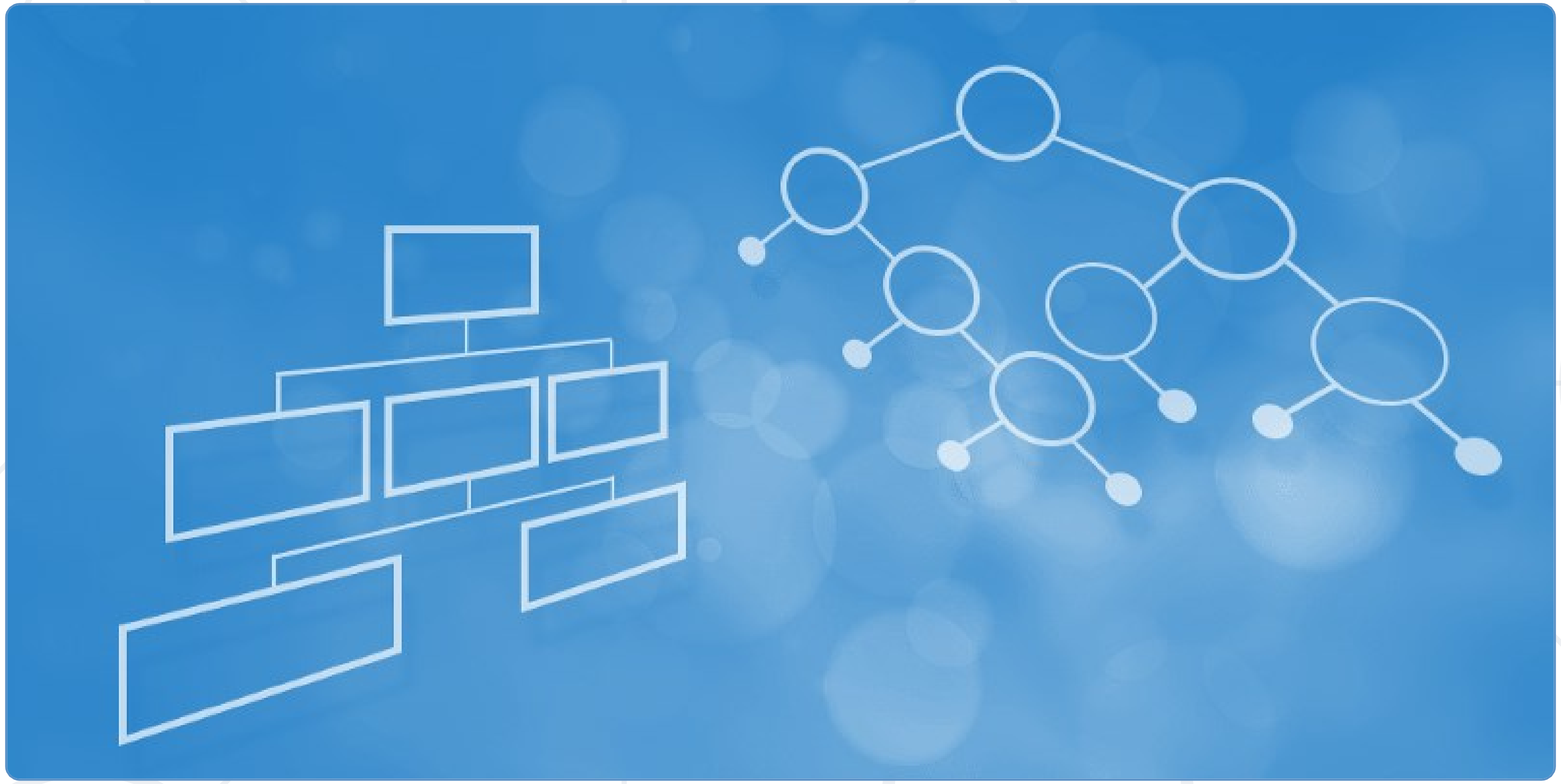
0	1	2	3	4	5	6	7
rosi		stamat	maria	pesho	ivan	joro	mitko

Collision Resolution: Linear Probing

Collision Resolution: Linear Probing

`Hash Function`

0	1	2	3	4	5	6	7
rosi	alex	stamat	maria	pesho	ivan	joro	mitko

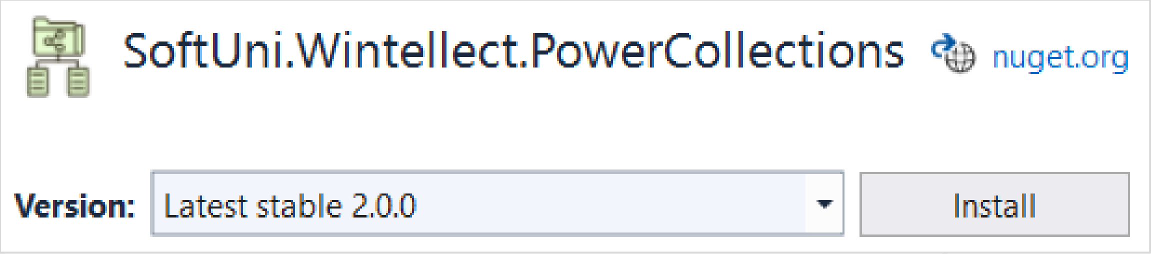


Examples

OrderedBag<T>

- **OrderedBag<T>**
  - A **bag** (multi-set) based on balanced search tree
  - Contains **<Key, Value> pairs**
  - Any number of elements may have **the same key**
  - Add / Find / Remove work in time O(log(N))
  - **`T`** should implement **`IComparable<T>`**

- To use **`OrderedBag<T>`**, install **Softuni.Wintellect.PowerCollections** from NuGet Packages



Example: OrderedBag<T>

- Use the class **OrderedBag<T>** to read a list of words and print them in a **sorted order**

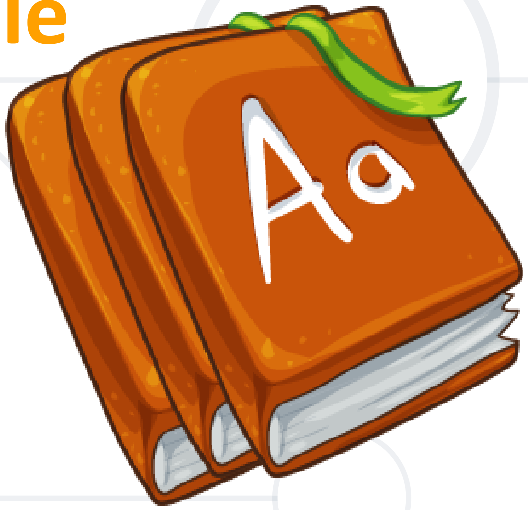
```
OrderedBag<string> bag = new OrderedBag<string>();
bag.Add("Peter");
bag.Add("Maria");
bag.Add("Ana");
bag.Add("Nina");
bag.Add("Mitko");

foreach (var element in bag)
{
   Console.WriteLine(element);
}
```

Microsoft Visual Studio

```
Ana
Maria
Mitko
Nina
Peter
```

MultiDictionary<TKey, TValue>

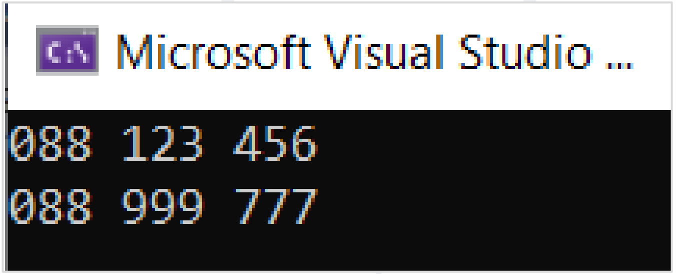
- **MultiDictionary<TKey, TValue>**
  - A dictionary (map) implemented by **hash-table**
  - **Allows duplicates** (configurable)
  - Add / Find / Remove work in time O(1)
  - Like `Dictionary<TKey, List<TValue>>`



- To use `MiltiDictionary<TKey, TValue>`, install **SoftUni.Wintellect.PowerCollections** from NuGet Packages

Example: MultiDictionary<K, V>

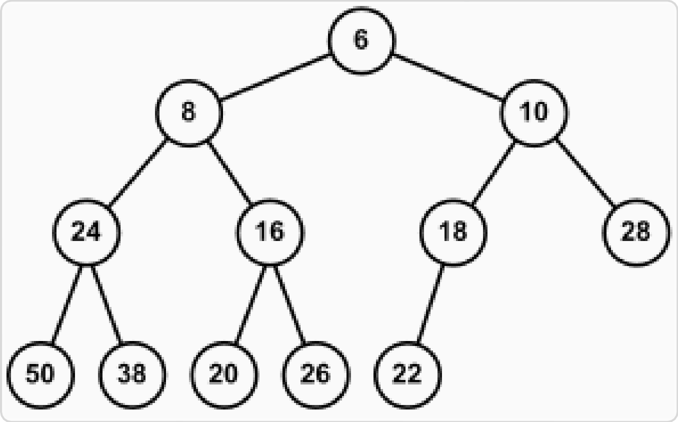
- Use the **MultiDictionary<K, V>** class to read a **phone book**, where each person can have **multiple phone numbers**:
  - Peter → 088 123 456
  - Maria → 089 999 888
  - Peter → 088 999 777
- Find the phone numbers for "Peter"

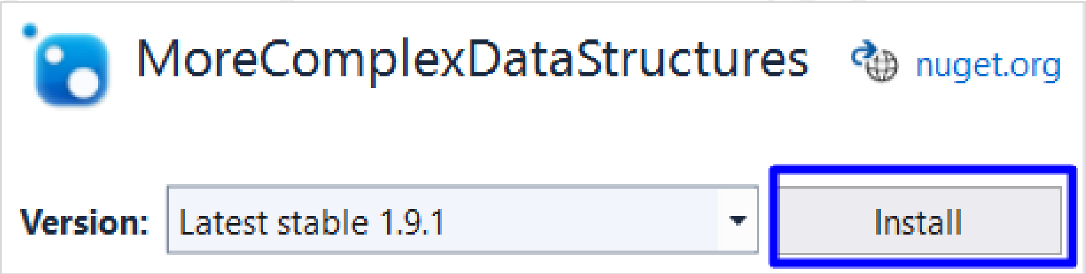



```
MultiDictionary<string, string> phoneBook =
  new MultiDictionary<string, string>(true);
phoneBook.Add("Peter", "088 123 456");
phoneBook.Add("Maria", "089 999 888");
phoneBook.Add("Peter", "088 999 777");

foreach (var phoneNum in phoneBook["Peter"])
{
   Console.WriteLine(phoneNum);
}
```

MaxHeap<T> (Binary Pyramid)

- **Heap<T>**
  - Tree-based data structure, stored in array
  - Fast retrieve of **min** and **max** element
- Heaps hold the **heap property** for each node:
  - **Min heap**: parent ≤ children
  - **Max heap**: parent ≥ children
- To use **MaxHeap<T>**, install NuGet package **MoreComplexDataStructures**

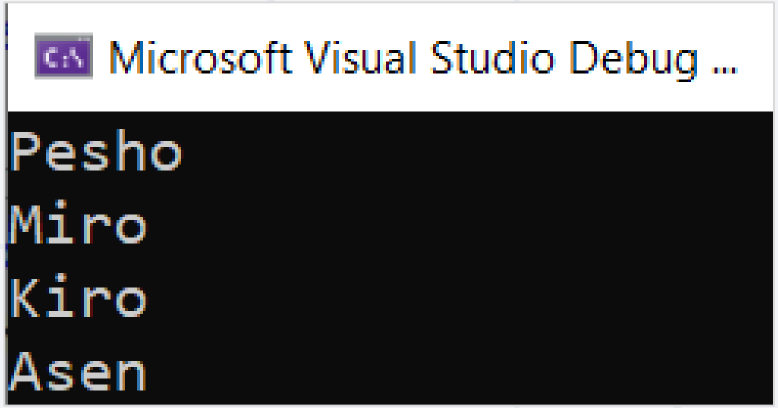




Example: MaxHeap<T>

- Use the **MaxHeap<T>** class to **sort** names in **descending order**
  - Print each name, using the **ExtractMax()** method

```
MaxHeap<string> heap = new MaxHeap<string>();
heap.Insert("Pesho");
heap.Insert("Kiro");
heap.Insert("Asen");
heap.Insert("Miro");

while (heap.Count > 0)
{
    Console.WriteLine(heap.ExtractMax());
}
```



Summary

- **Data structures** organize data in computer systems for efficient use
  - Abstract data types (**ADT**) describe a set of operations
- **Linear data structures: arrays, lists, stack, queue, linked list**
- **Dictionaries and hash tables**
- **Complex data structures: Bag, Heap, …**

Software University

Questions?

SoftUni

- Software University
- SoftUni Svetlina
- SoftUni Creative
- SoftUni Digital
- SoftUni Foundation
- SoftUni Kids

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://softuni.org
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

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

Software University

FORUM