List

- List collection contains a group of elements of same type.
- Full Path: System.Collections.Generic.List
- The 'List' class is a generic class; so you need to specify data type of value while creating object.

List Collection

[0]	value0
[1]	value1
[2]	value2
[3]	value3
[4]	value4
[5]	value5
[6]	value6

'List' collection

```
List<type>  referenceVariable = new  List<type>( );
```

- It is dynamically sized. You can add, remove elements at any time.
- It allows duplicate values.
- It is index-based. You need to access elements by using zero-based index.
- It is not sorted by default. The elements are stored in the same order, how they are initialized.
- It uses arrays internally; that means, recreates array when the element is added / removed.
  - The 'Capacity' property holds the number of elements that can be stored in the internal array of the List. If you add more elements, the internal array will resized to the 'Count' of elements.

Properties

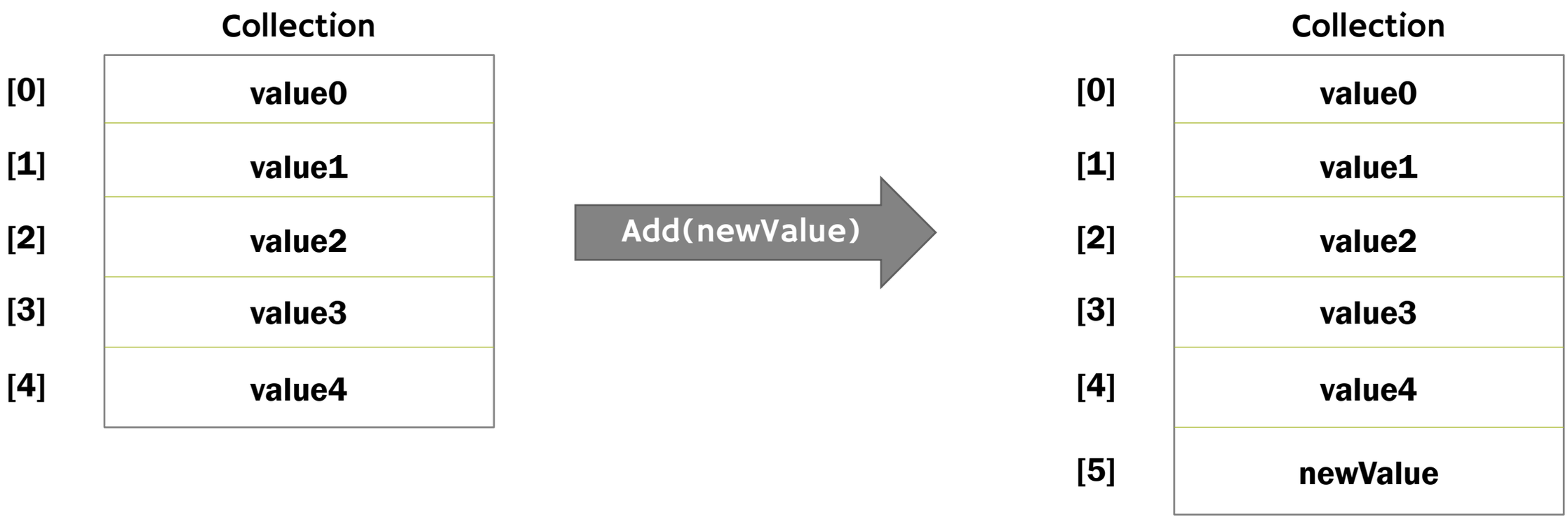
- Count
- Capacity

Methods

- Add(T)
- AddRange(IEnumerable<T>)
- Insert(int, T)
- InsertRange(int, IEnumerable<T>)
- Remove(T)
- RemoveAt(int)
- RemoveRange(int, int)
- RemoveAll(Predicate<T>)
- Clear()
- IndexOf(T)
- BinarySearch(T)
- Contains(T)
- Sort()
- Reverse()
- ToArray()
- ForEach(Action<T>)
- Exists(Predicate<T>)
- Find(Predicate<T>)
- FindIndex(Predicate<T>)
- FindLast(Predicate<T>)
- FindLastIndex(Predicate<T>)
- FindAll(Predicate<T>)
- ConvertAll(T)

Add()

Add() › This method adds a new element to the collection.



List - Add() method

```
void List.Add(T newValue)
```

AddRange()

AddRange() › This method adds a new set of elements to the collection.

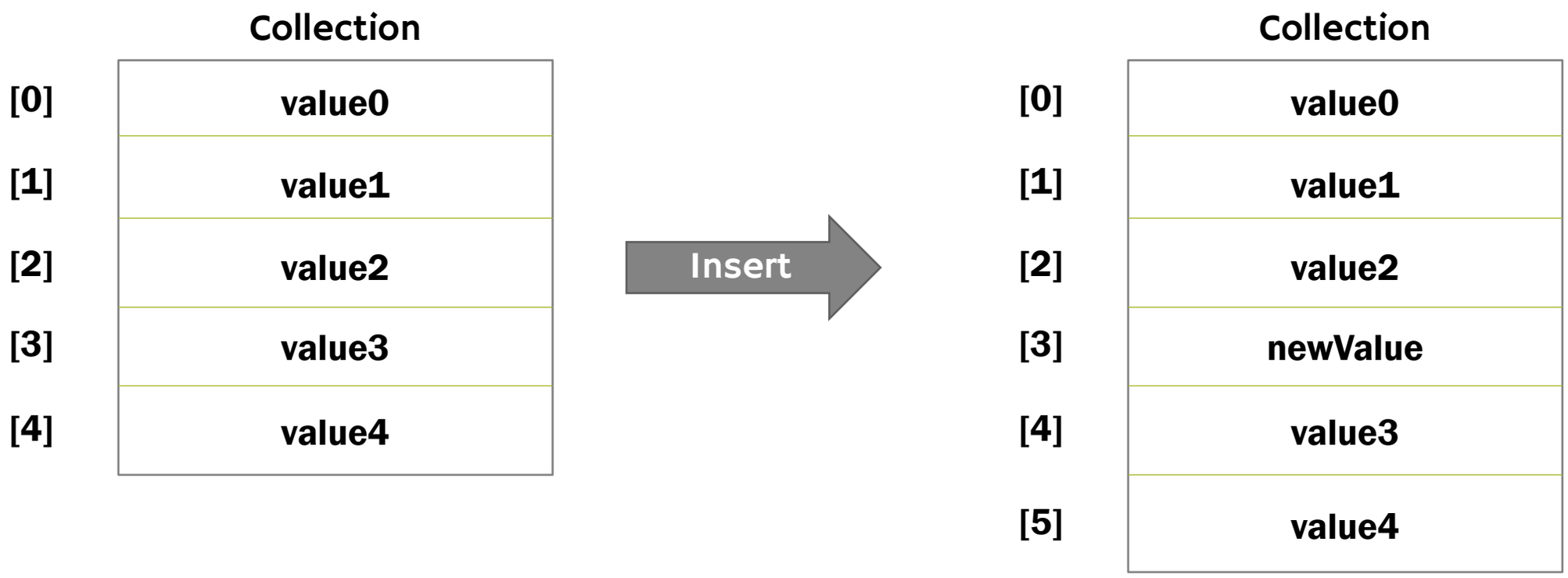
List - AddRange() method

```
void List.AddRange(IEnumerable<T> newValue)
```


List - AddRange() - Example

Insert()

- This method adds a new element to the collection at the specified index.



List - Insert() method
void List.Insert(int index, T newValue)

List - Insert() - Example
List.Insert(3, newValue)

InsertRange()

- This method adds a new set of elements to the collection at the specified index.

List - InsertRange() method

```
void List.InsertRange(int index, IEnumerable<T> newValue)
```

List - InsertRange() - Example

```
List.InsertRange(2, new List<int>() { newValue1, newValue2 } )
```

Remove()

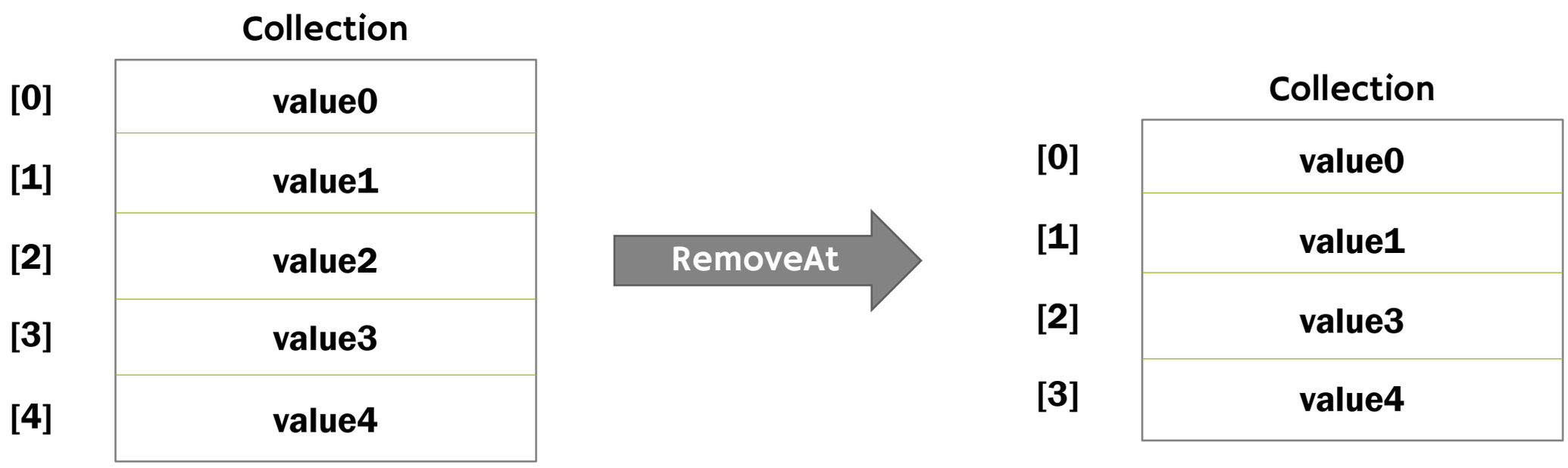
- This method removes the specified element from the collection.

List - Remove() method
void List.Remove(T newValue)

List - Remove() - Example
List.Remove(value2)

RemoveAt()

- This method removes an element from the collection at the specified index.



List - RemoveAt() method
`void List.RemoveAt(int index)`

List - RemoveAt() - Example
`List.RemoveAt(2)`

RemoveRange()

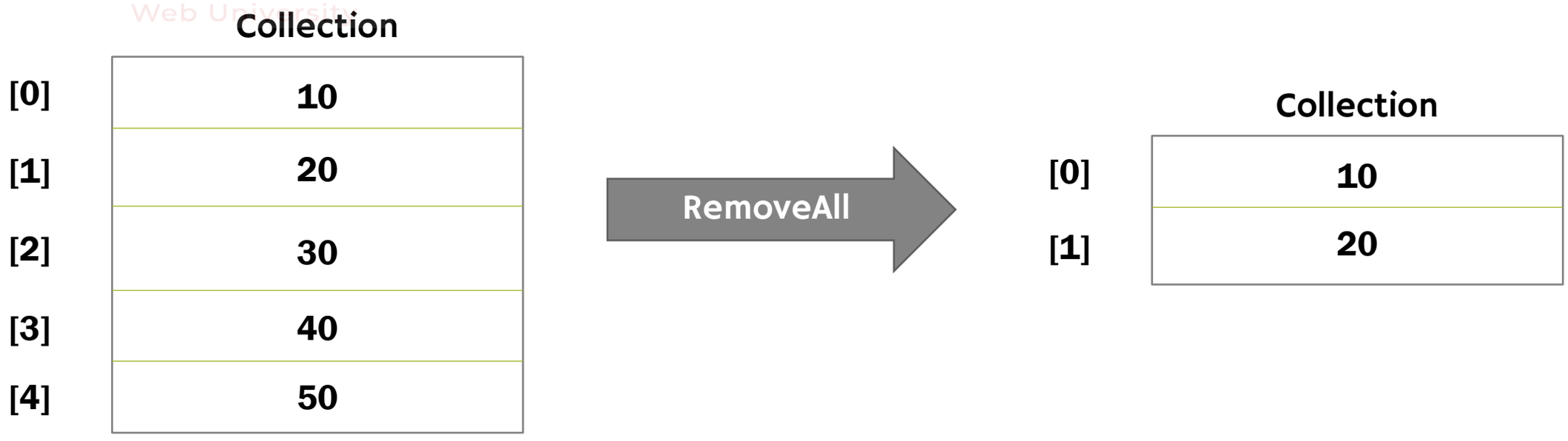
- This method removes specified count of elements starting from the specified startIndex.

List - RemoveRange() method
void List.RemoveRange(int index, int count)

List - RemoveRange() - Example
List.RemoveRange(1, 2)

RemoveAll()

- This method removes all the elements that are matching with the given condition.
- You can write your condition in the lambda expression of Predicate type.

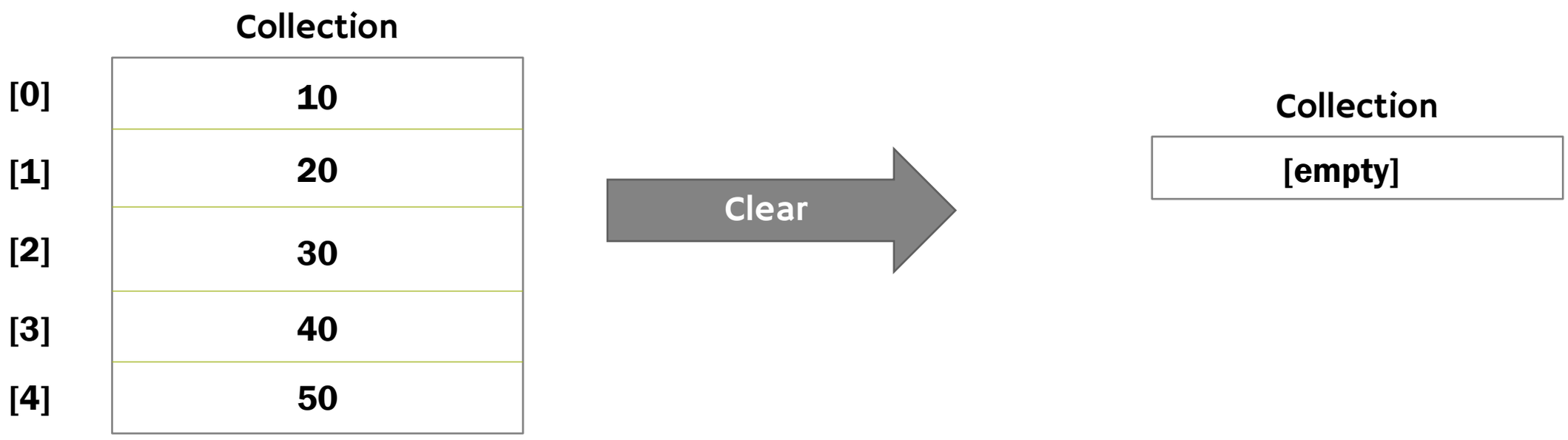


List - RemoveAll() method
void List.RemoveAll(value => condition)

List - RemoveAll() - Example
List.RemoveAll(n => n >= 30)

Clear()

- This methods removes all elements in the collection.

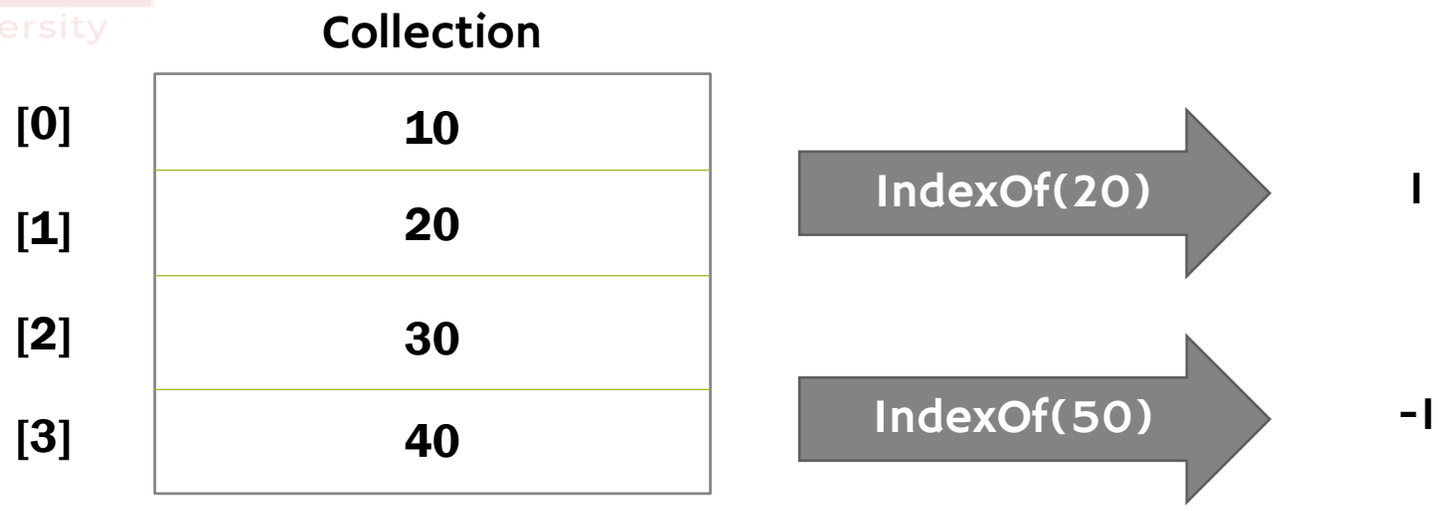


List - Clear() method
void List.Clear( )

List - Clear() - Example
List.Clear( )

IndexOf()

- This method searches the collection for the given value.
  - If the value is found, it returns its index.
  - If the value is not found, it returns -1.



List - IndexOf() method
int List.IndexOf(T value, int startIndex)

List - IndexOf() - Example
List.IndexOf(20)

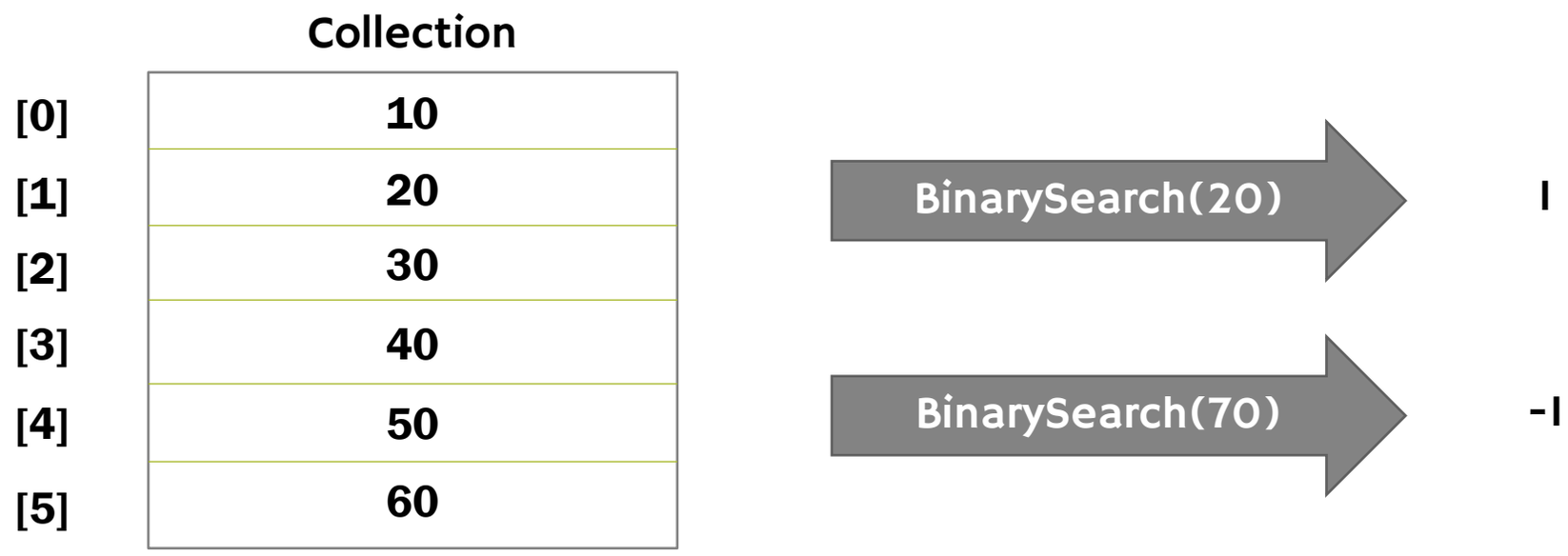
- The "IndexOf" method performs linear search. That means it searches all the elements of the collection, until the search value is found. When the search value is found in the collection, it stops searching and returns its index.
- The linear search has good performance, if the collection is small. But if the collection is larger, Binary search is recommended to improve the performance.

Parameters

- **value:** This parameter represents the actual value that is to be searched.
- **startIndex:** This parameter represents the start index, from where the search should be started.

BinarySearch()

- This method searches the array for the given value.
  - If the value is found, it returns its index.
  - If the value is not found, it returns -1.



List - BinarySearch() method
int List.BinarySearch(T value)

List - BinarySearch() - Example
List.BinarySearch(20)

- The "Binary Search" requires a collection, which is already sorted.
  - On unsorted collections, binary search is not possible.
- It directly goes to the middle of the collection (collection size / 2), and checks that item is less than / greater than the search value.
- If that item is greater than the search value, it searches only in the first half of the collection.
- If that item is less than the search value, it searches only in the second half of the array.
- Thus it searches only half of the array. So in this way, it improves performance

Parameters

- value: This parameter represents the actual value that is to be searched.

Contains()

- This method searches the specified element and returns 'true', if it is found; but returns 'false', if it is not found.

List - Contains() method
bool List.Contains(T value)

Contains() - Example
List.Contains(30)

Sort()

- This method sorts the collection in ascending order.

Collection

[0]	100
[1]	950
[2]	345
[3]	778
[4]	20

Sort →

Collection

[0]	20
[1]	100
[2]	345
[3]	778
[4]	950

List - Sort() method

```
void List.Sort( )
```

List - Sort() - Example

```
List.Sort( )
```

Reverse()

- This method reverses the collection.

Collection

[0]	100
[1]	950
[2]	345
[3]	778
[4]	20

Collection

[0]	20
[1]	778
[2]	345
[3]	950
[4]	100

List - Reverse() method

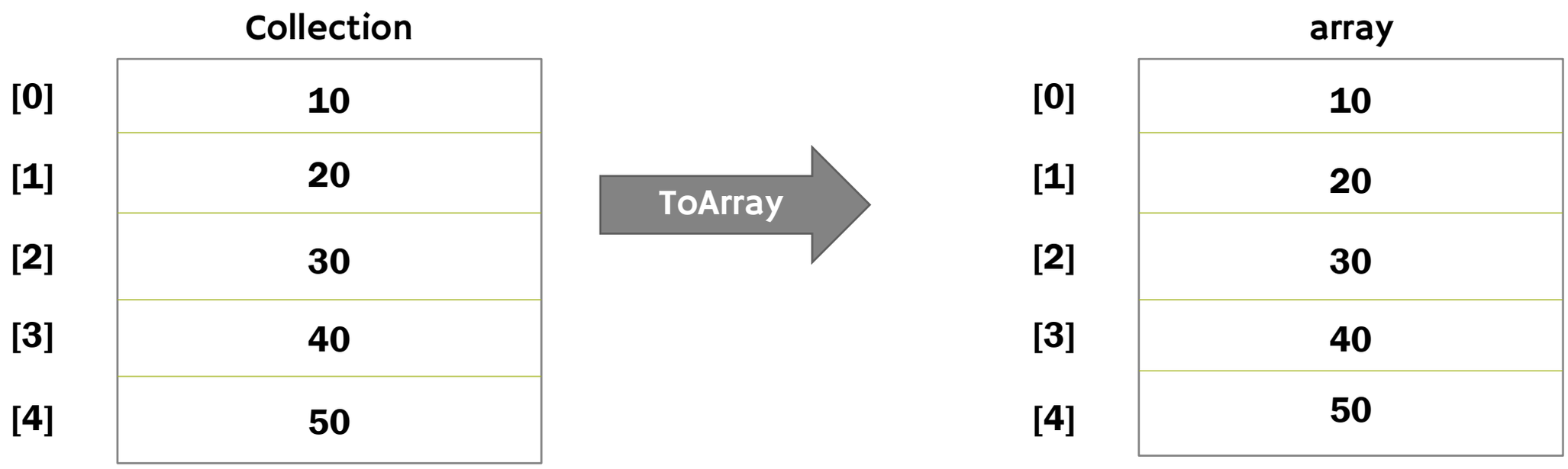
```
void List.Reverse( )
```

List - Reverse() - Example

```
List.Reverse( )
```

ToArray()

- This method converts the collection into an array with same elements.

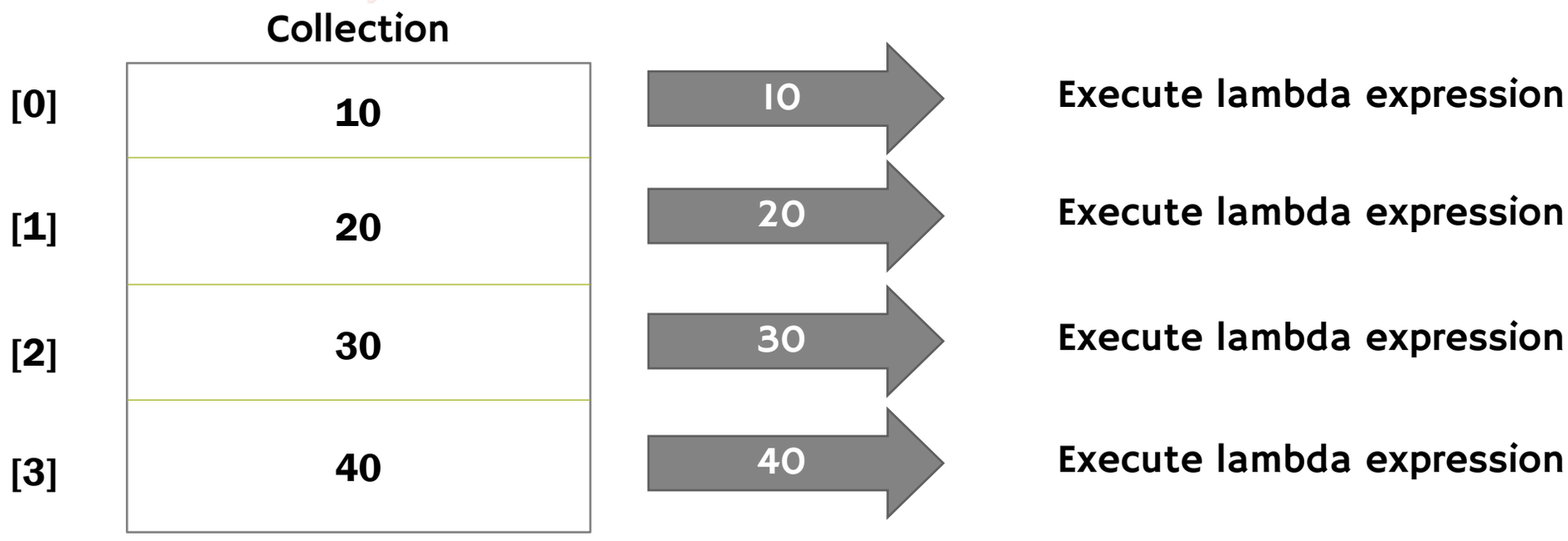


List - ToArray() method
T[ ] List.ToArray( )

List - ToArray() - Example
List.ToArray( )

ForEach()

- This method executes the lambda expression once per each element.



List - ForEach() method

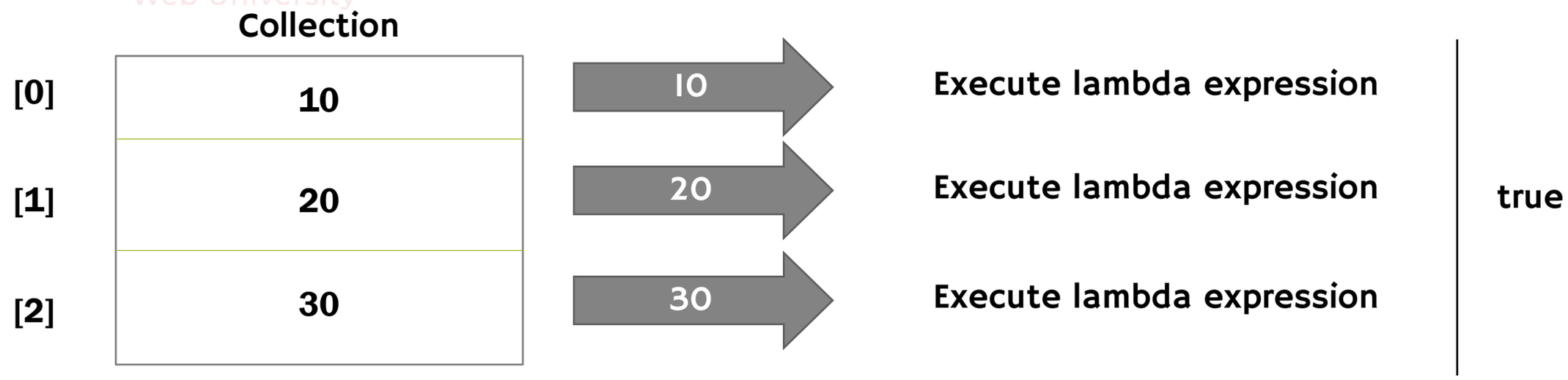
```
void List.ForEach( Action<T> )
```

List - ForEach() - Example

```
List.ForEach( n => { Console.WriteLine(n); } )
```

Exists()

- This method executes the lambda expression once per each element.
- It returns true, if at least one element matches with the given condition; but returns false, if no element matches with the given condition.

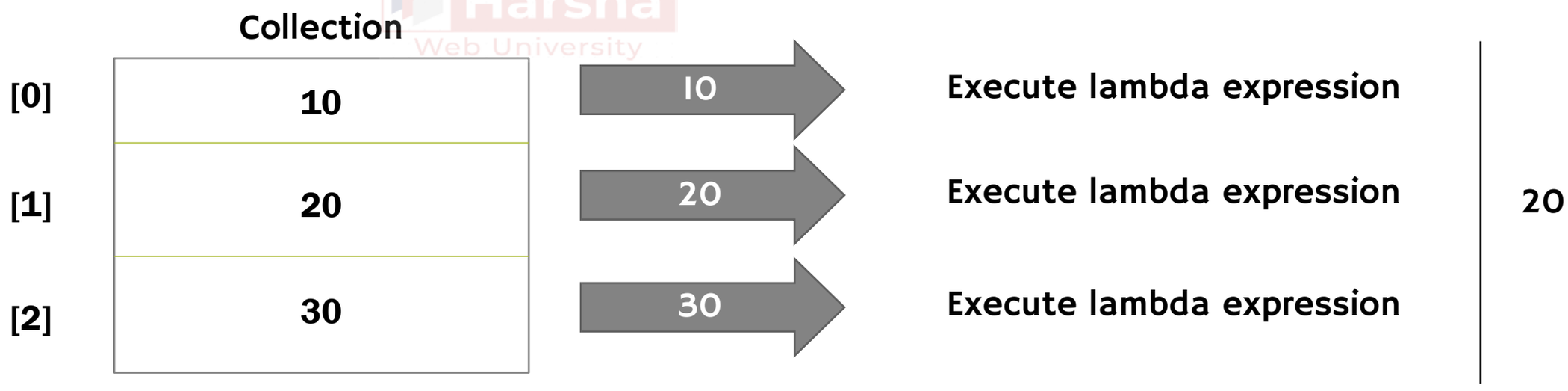


List - Exists() method
bool List.Exists(Predicate<T>)

List - Exists() - Example
List.Exists(n => n > 15)

Find()

- This method executes the lambda expression once per each element.
- It returns the first matching element, if at least one element matches with the given condition; but returns the default value, if no element matches with the given condition.

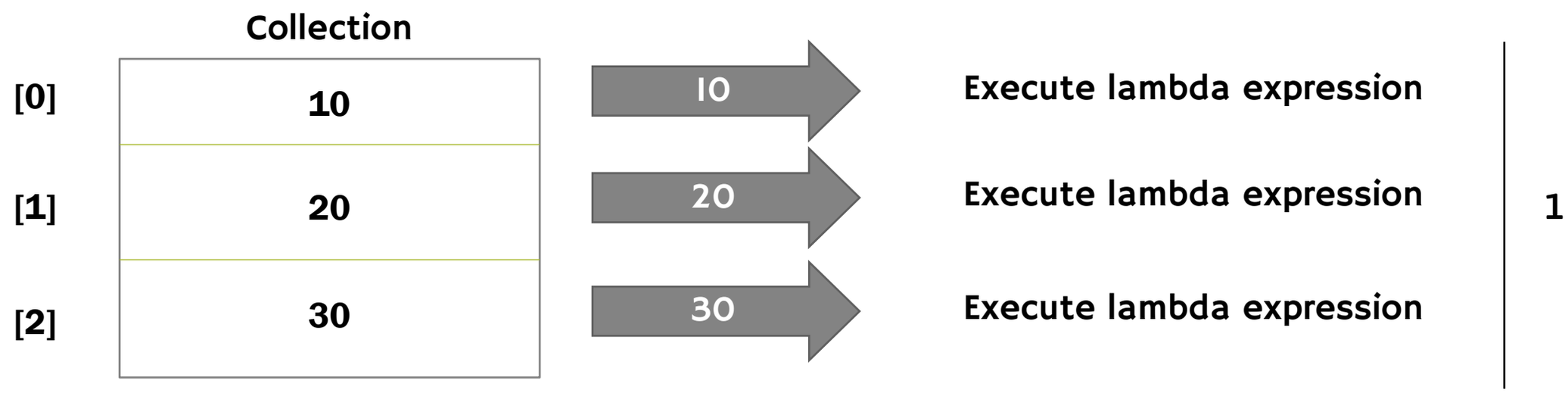


List - Find() method
T List.Find(Predicate<T>)

List - Find() - Example
List.Find(n => n > 15)

FindIndex()

- This method executes the lambda expression once per each element.
- It returns index of the first matching element, if at least one element matches with the given condition; but returns -1, if no element matches with the given condition.

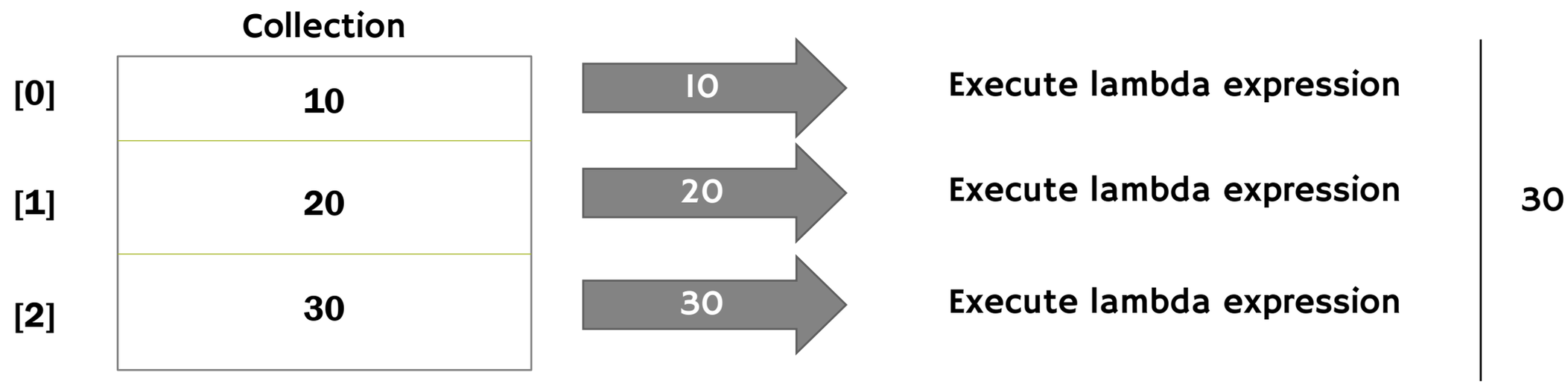


List - FindIndex() method
`int List.FindIndex( Predicate<T> )`

List - FindIndex() - Example
`List.FindIndex( n => n > 15 )`

FindLast()

- This method executes the lambda expression once per each element.
- It returns the last matching element, if at least one element matches with the given condition; but returns the default value, if no element matches with the given condition.

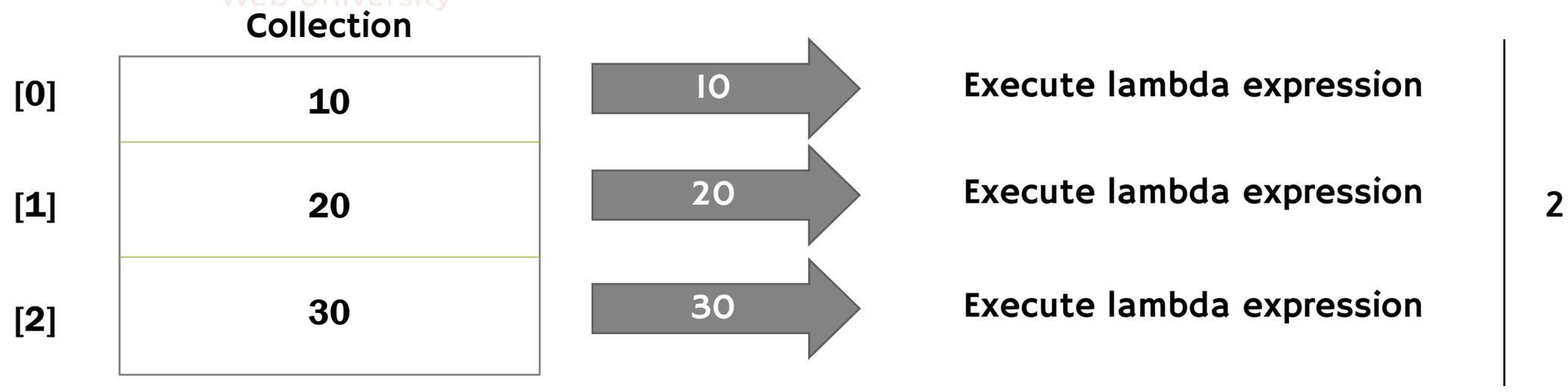


List - FindLast() method
T List.FindLast(Predicate<T>)

List - FindLast() - Example
List.FindLast(n => n > 15)

FindLastIndex()

- This method executes the lambda expression once per each element.
- It returns index of the last matching element, if at least one element matches with the given condition; but returns -1, if no element matches with the given condition.

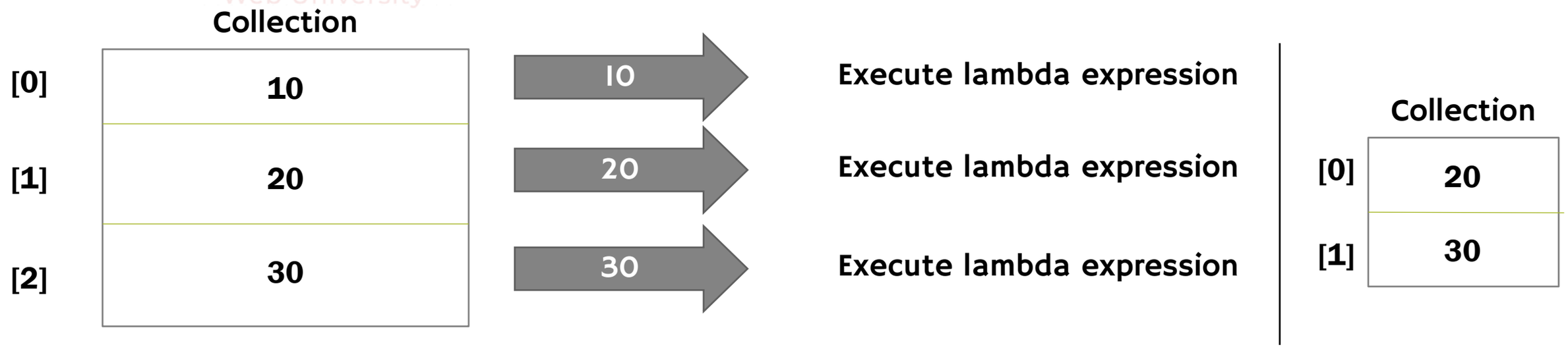


List - FindLastIndex() method
`int List.FindLastIndex( Predicate<T> )`

List - FindLastIndex() - Example
`List.FindLastIndex( n => n > 15 )`

FindAll()

- This method executes the lambda expression once per each element.
- It returns all matching elements as a collection, if there are one or more matching elements; but returns empty collection if no matching elements.

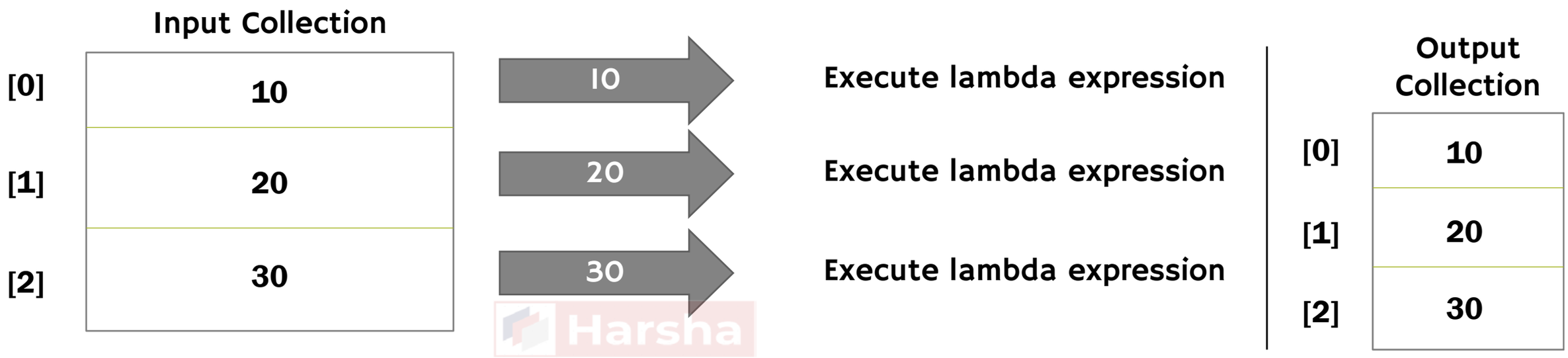


List - FindAll() method
List<T> List.FindAll(Predicate<T>)

List - FindAll() - Example
List.FindAll(n => n > 15)

ConvertAll()

- This method executes the lambda expression once per each element.
- It adds each returned element into a new collection and returns the same at last; thus it converts all elements from the input collection as output collection.



List - ConvertAll() method
List<TOutput> List.ConvertAll(Converter<TInput, TOutput>)

List - ConvertAll() - Example
List.ConvertAll(n => Convert.ToDouble(n))