Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

# Linear Data Structures: Arrays, Lists, Queues, and Stacks

Data Structures

**Fernando Marson**

fernando.marson@universidadeeuropeia.pt

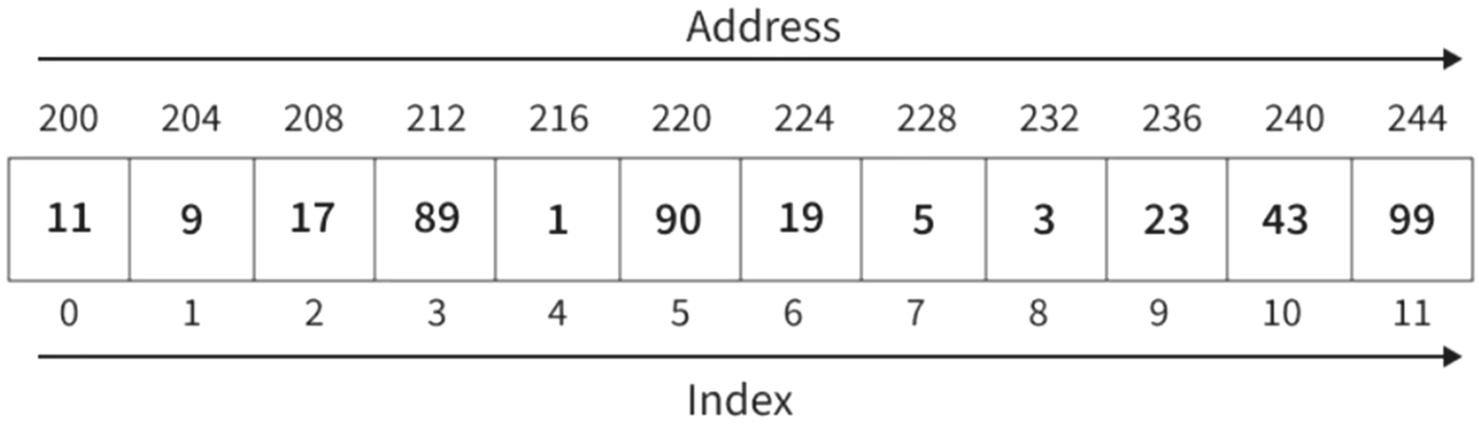
# Linear Data Structures

- Arrays
  - 1D (vectors)
  - 2D, 3D, n-dimensional (dense matrices)
  - Special Arrays (Jagged arrays, Sparse Matrices)
- Lists
  - Single Linked List
  - Double Linked List
- Queues
- Stacks

# Arrays

# 1D Array (Vector)

- Arrays are collections of elements with a **fixed size**, **indexed by integers**.
- They store **multiple values of the same data type** in a single variable, allowing efficient access to elements through their indices.

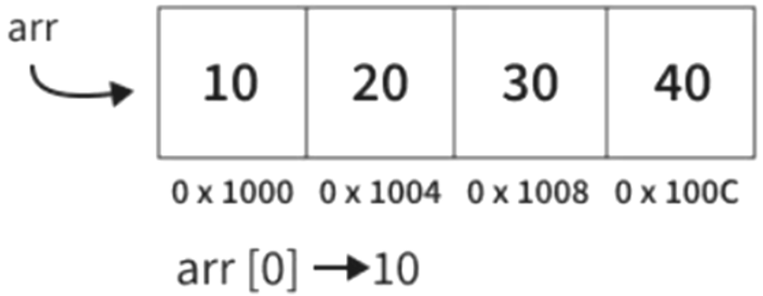



Source: https://www.geeksforgeeks.org

# 1D Array (Vector)

## Primitive Array

int []arr = {10, 20, 30, 40}




arr [0] → 10

Primitive arrays stores the values directly in the memory

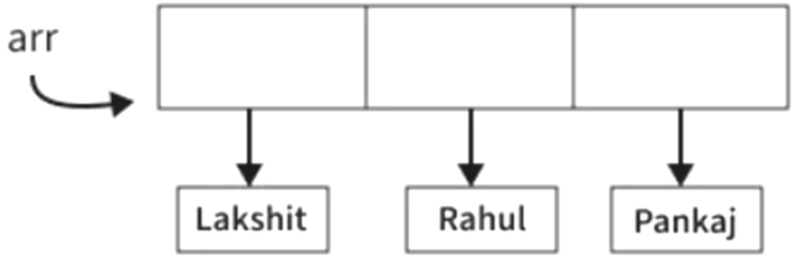
## Object Array

String []arr = new String [3]

arr [0] = “Lakshit”

arr [1] = “Rahul”

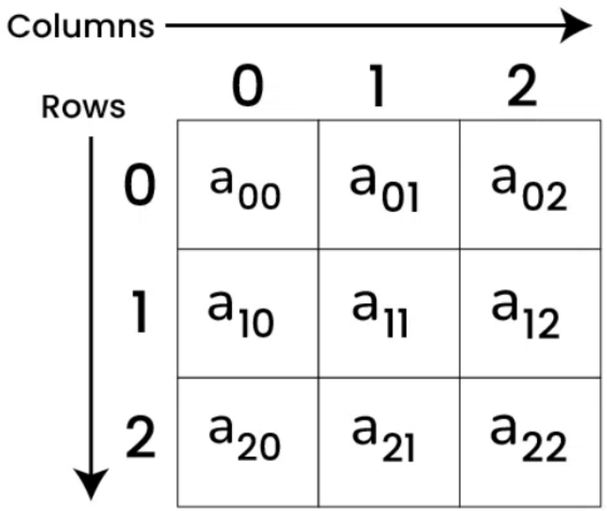
arr [2] = “Pankaj”



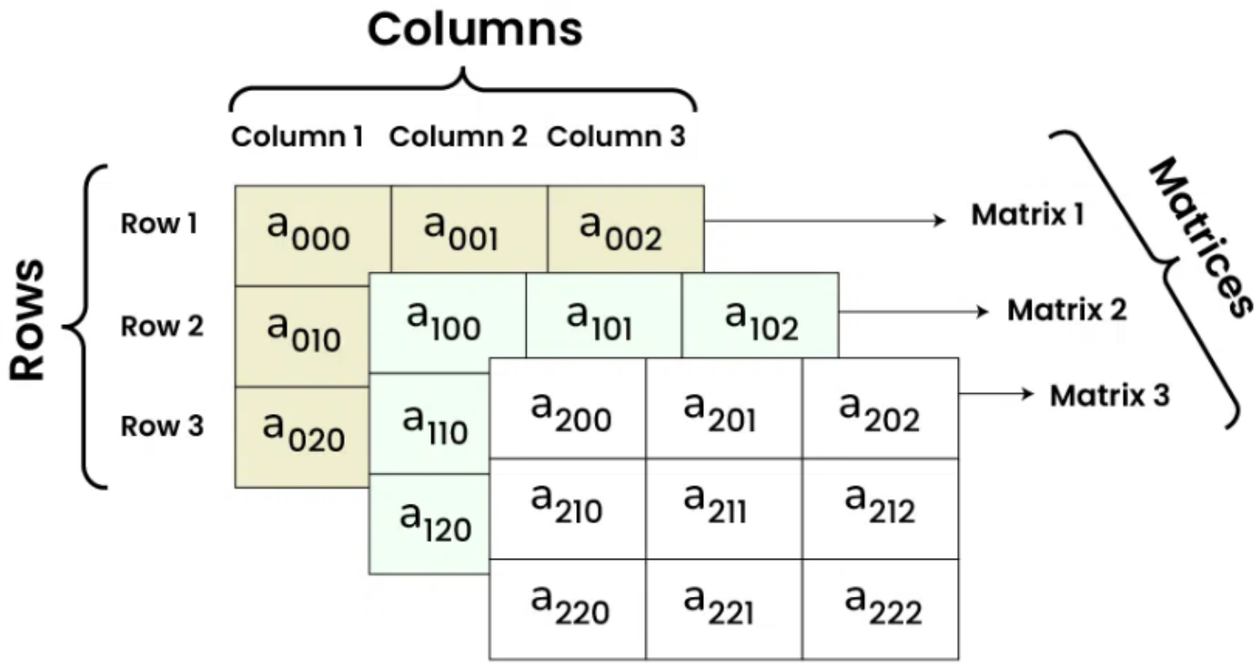

Each element of the object array stores a reference to separate string object

Source: https://www.geeksforgeeks.org

# N-Dimensional Arrays




2D

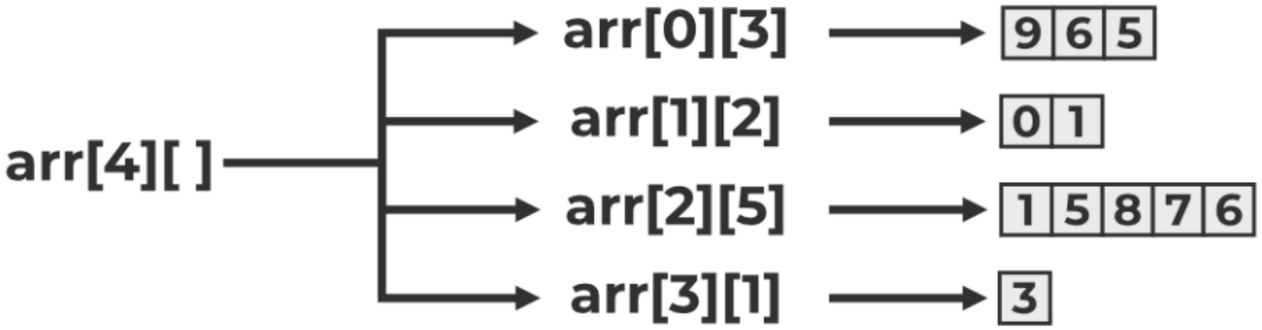



3D

Source: https://www.geeksforgeeks.org

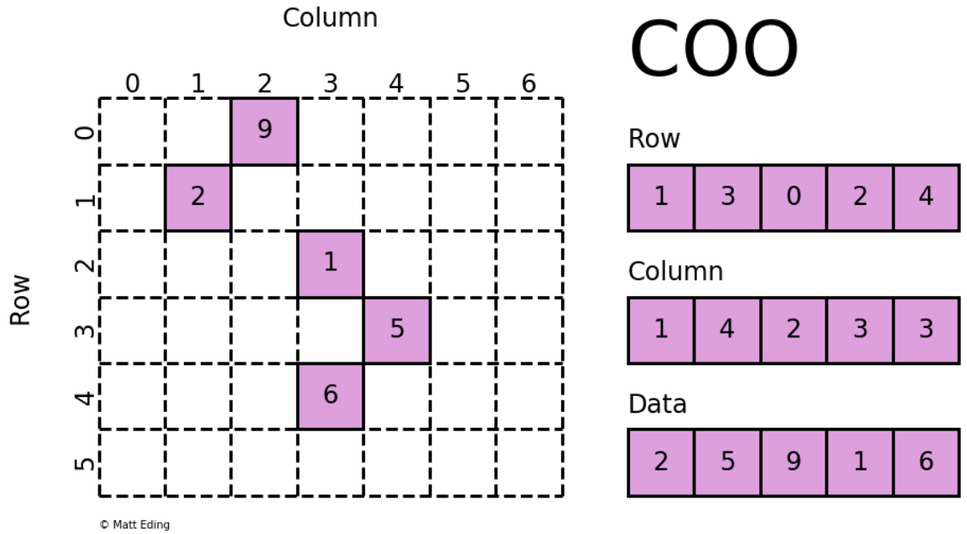
# Special Arrays

## Jagged/Irregular Arrays




Source: https://www.geeksforgeeks.org

## Sparse Matrices




Source: https://matteding.github.io

# Linked Lists

# Linked Lists

- Concept
  - Lists are **dynamic** collections.
  - Unlike arrays, lists **can grow and shrink** in size as needed.
  - They allow flexible manipulation of elements, with methods to add, remove, and insert items.

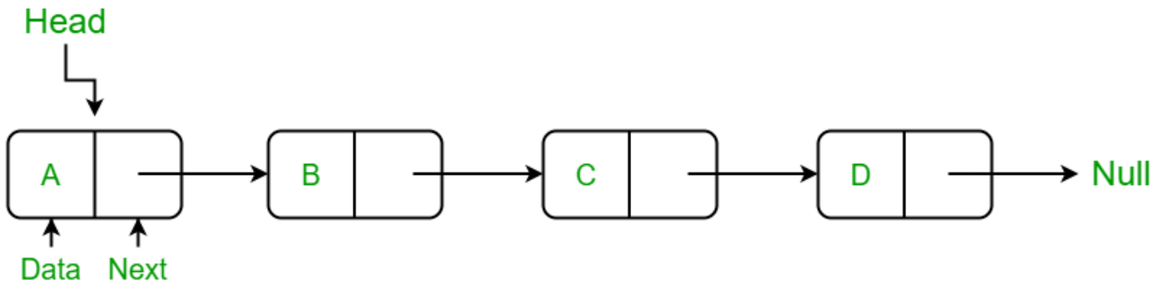
# Types of Lists

- Simple linked lists
- Circular linked lists
- Double linked lists
- Circular double linked lists

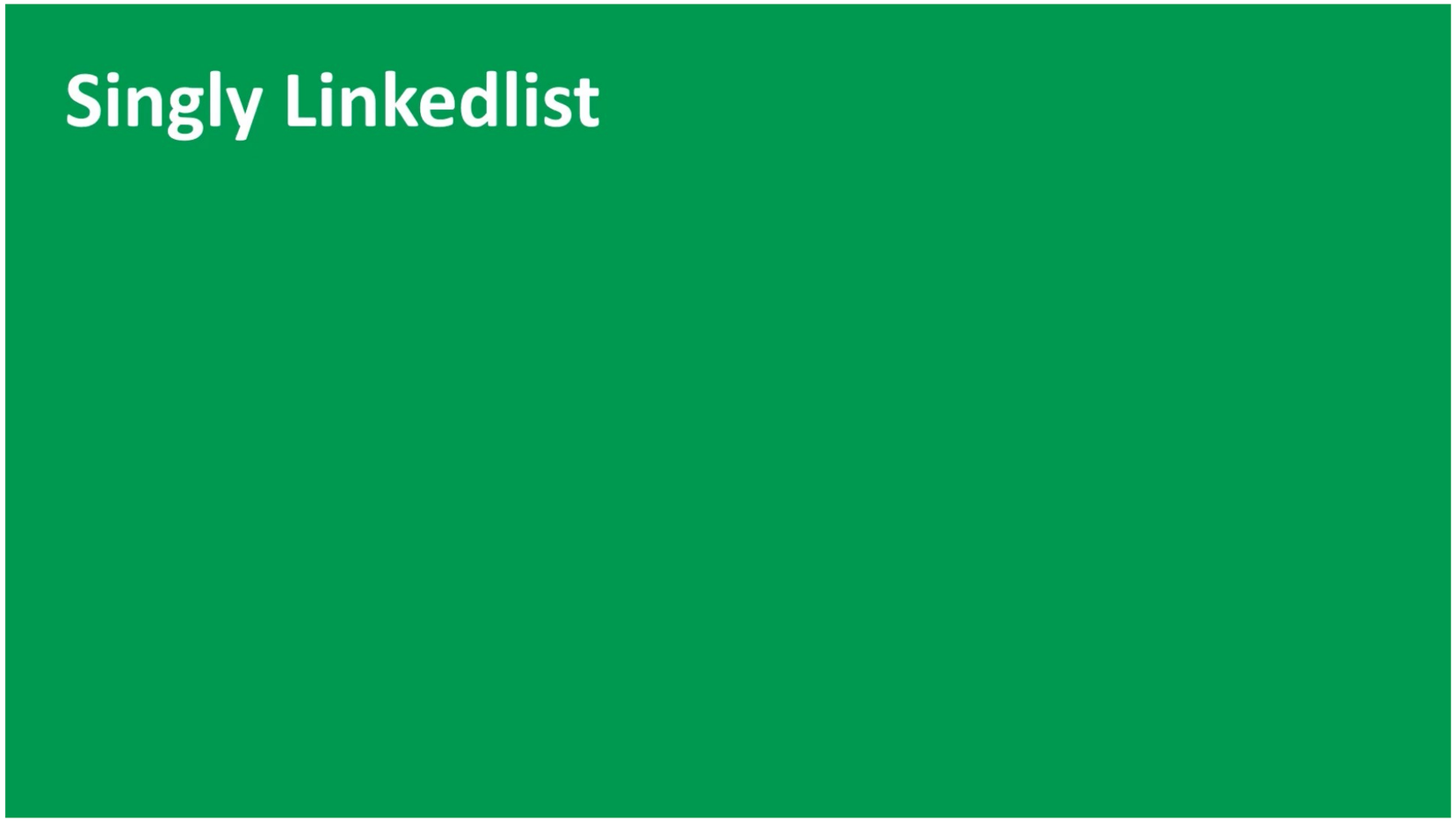
# Single Linked Lists

- Concept
  - A singly linked list is a linear data structure consisting of nodes. Each node contains **two parts**: the **data** and a **reference** (or link) **to the next node** in the sequence.

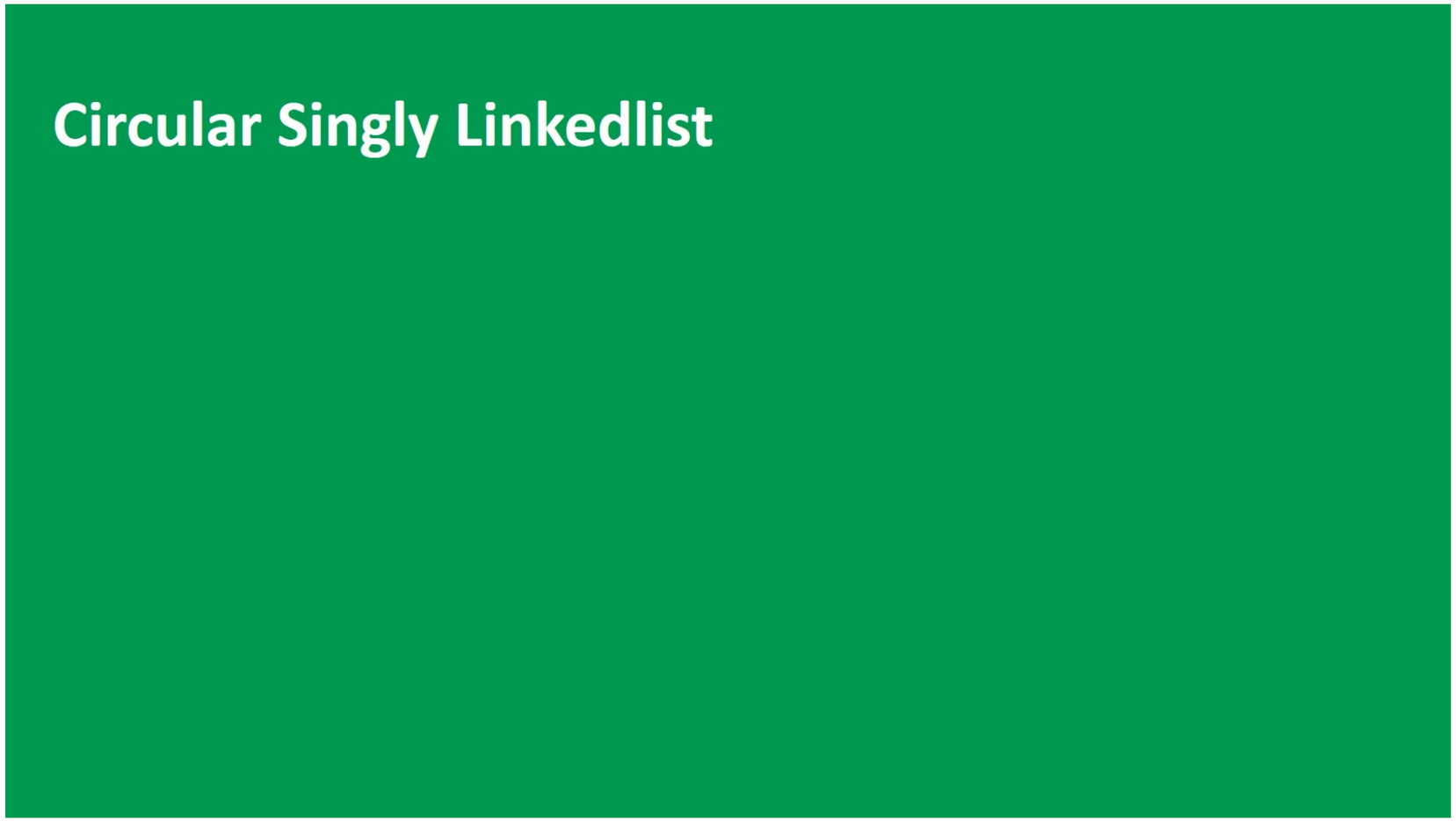
  - The list **starts** with a **head node**, and it **ends** when a node's **next reference points to null**.




Source: https://www.geeksforgeeks.org



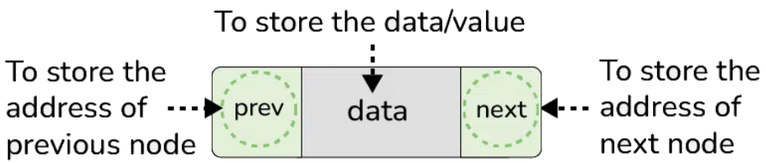

**Source:** https://www.youtube.com/@visualhow

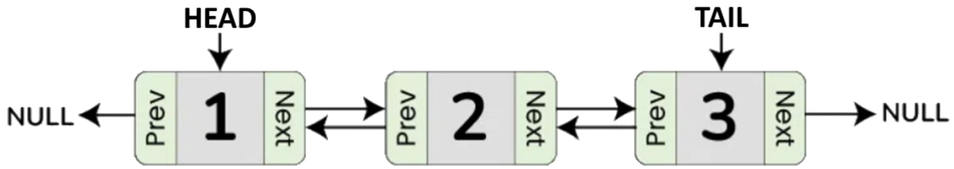



**Source:** https://www.youtube.com/@visualhow

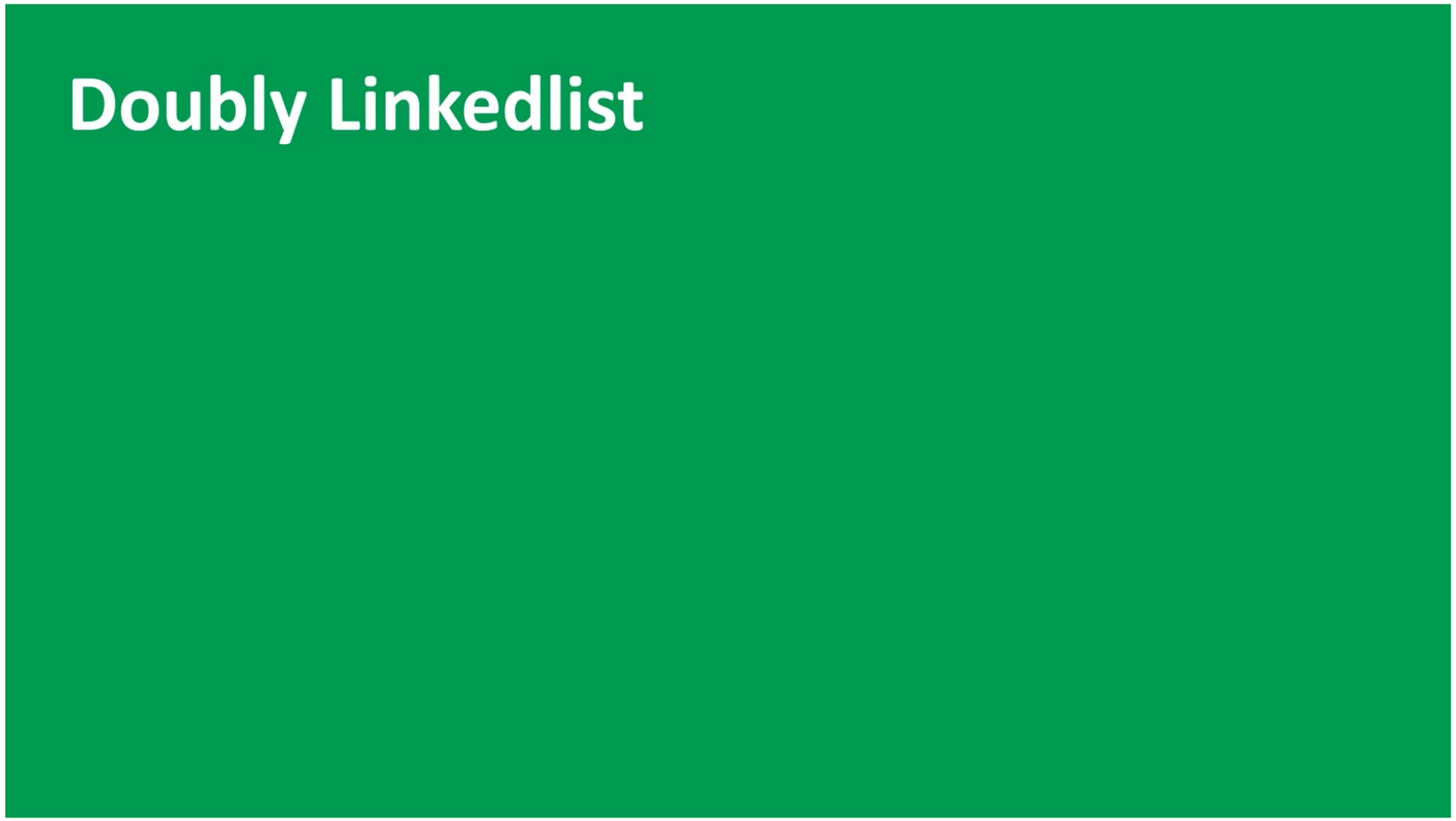
# Double Linked Lists

- Concept
  - A doubly linked list is like a singly linked list but **with an additional reference in each node**, allowing traversal in both forward and backward directions.
  - Each node contains **three parts**: the **data**, a reference to the **next node**, and a reference to the **previous node**.
  - There are **two special nodes**: **head** and **tail**.

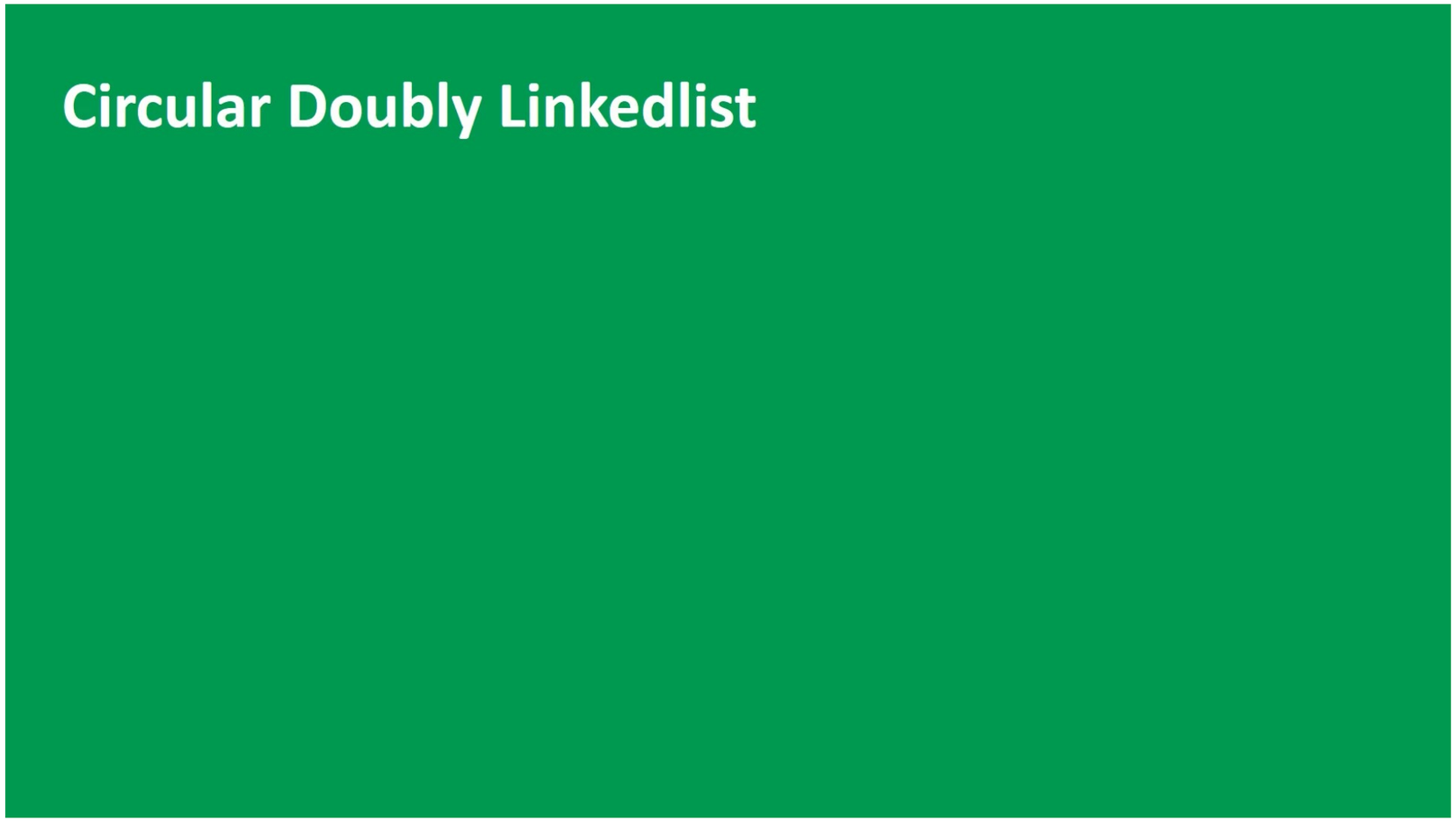



**Source**: https://www.geeksforgeeks.org




**Source:** https://www.youtube.com/@visualhow




**Source:** https://www.youtube.com/@visualhow

# Linked Lists - Operations

- Operations
  - Insertion
    - At the beginning
    - At the end
    - At a given position
  - Deletion
    - From the beginning
    - From the end
    - From a given position
  - Traversal (access each position, one by one)
  - Searching
  - Updating (modifying the value of a node)
  - Checking for list emptiness
  - Counting the number of nodes
  - Reversing the list

# Queues

# Queues

- A **queue** is a data structure that follows the **First In**, **First Out** (**FIFO**) principle.
- This means that **the first element added** to the queue will be **the first one to be removed**.


- **Queues** are commonly used in various applications, such as **managing tasks in a printer queue, handling requests in web servers**, or **any scenario where order matters**.

# Queues

- Main terms/operations
  - **Enqueue**: The operation of **adding an element to the back** of the queue.
  - **Dequeue**: The operation of **removing the front element** from the queue.
  - **Front**: The **element at the front** of the queue, which will be removed next.
  - **Back/Rear/Tail**: The **element at the back** of the queue, which was added last.
  - **Empty**: A check to determine if the queue has no elements.
  - **Size**: Returns the number of elements currently in the queue.

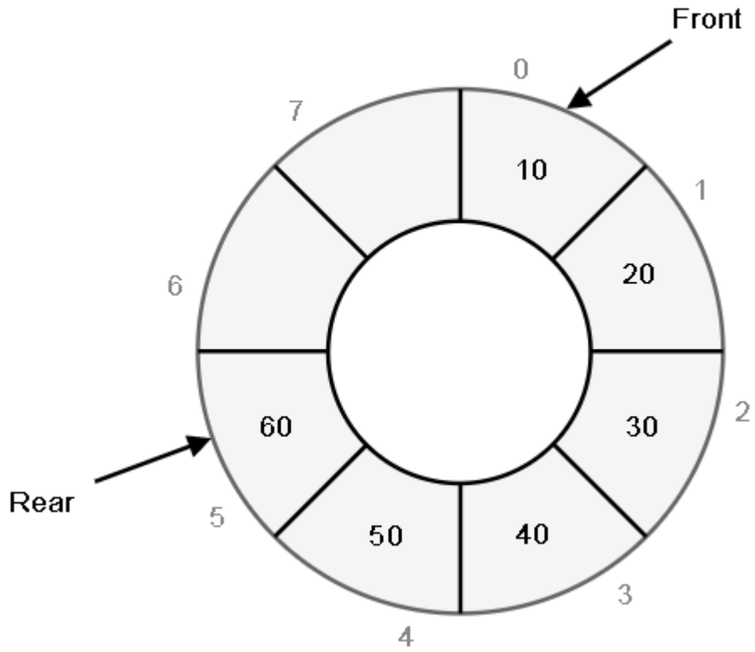
# Types of Queues

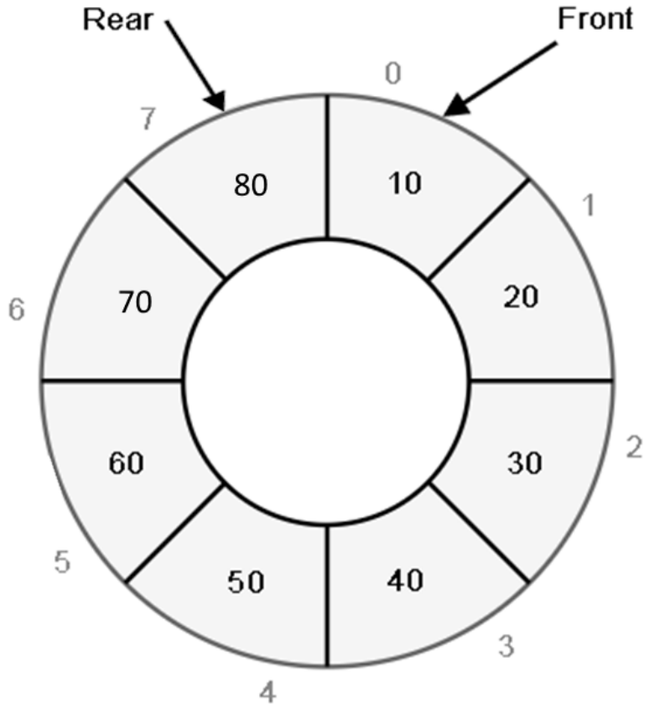
- Simple queues
- Circular queues
- Double-ended queues (aka Deque)
- Priority queues


**Source:** https://www.youtube.com/@visualhow

# Circular Queue

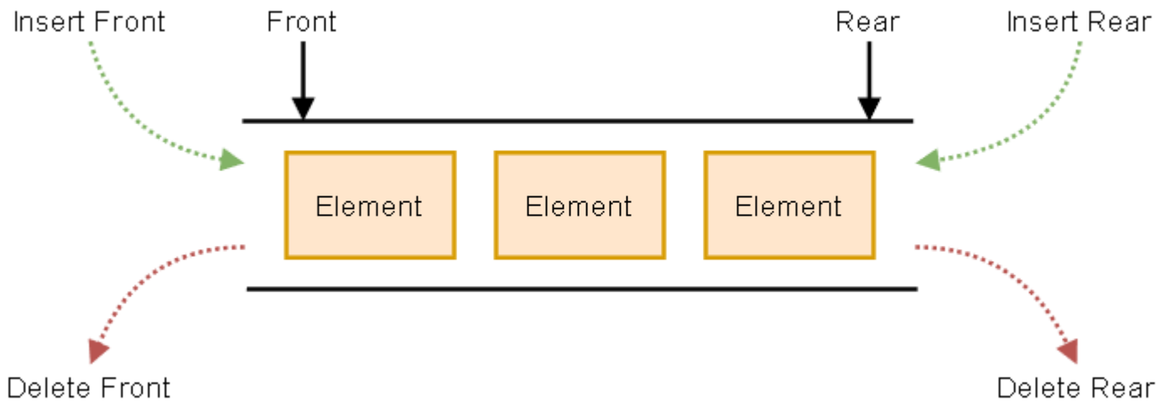
- The **last position is connected back to the first position**, making it circular.



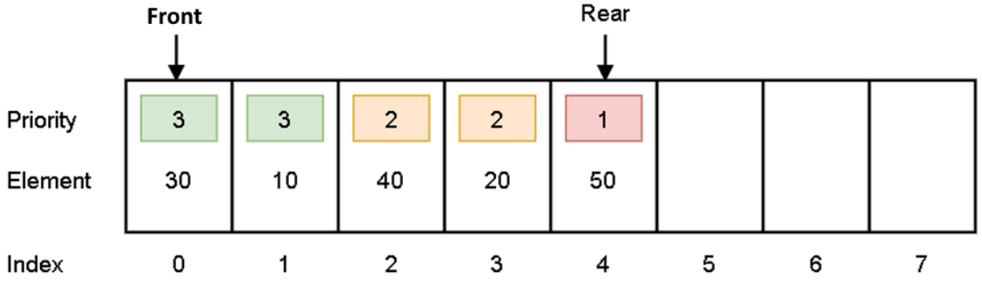
# Double-Ended Queue (Deque)

- Allows **insertion and removal from both ends** of the queue.

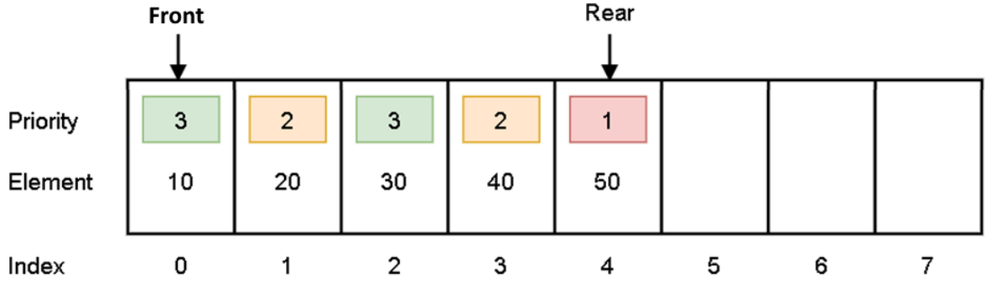


# Priority Queue

- **Each element has a priority**; elements with higher priority are **dequeued** before lower-priority elements.




**Ordered**

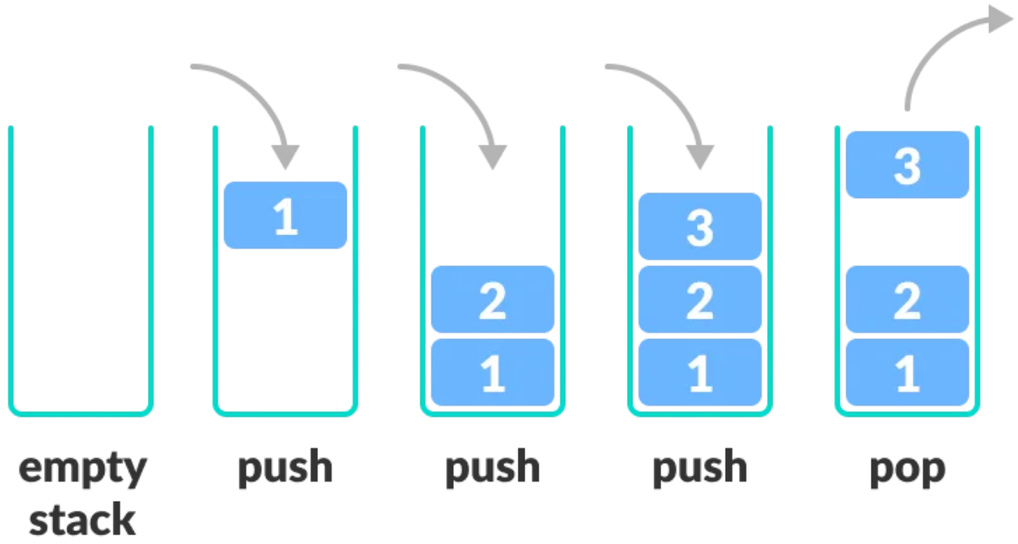



**Unordered**

# Stacks

# Stack

- A **stack** is a data structure that follows the **Last In**, **First Out** (**LIFO**) principle.
- This means that the **last element added** to the stack will be the **first one to be removed**.



# Stack

- Main operations:
  - **push**: The operation of **adding an element into** the stack.
  - **pop**: The operation of **removing an element from** the stack.
  - **peek**: Return the **element at the top** of the queue, without remove it.

  - **isEmpty**: A check to determine if the stack has no elements.
  - **size**: Returns the number of elements currently in the stack.

# Stack

**Source:** https://www.youtube.com/@visualhow

# Collections in C#

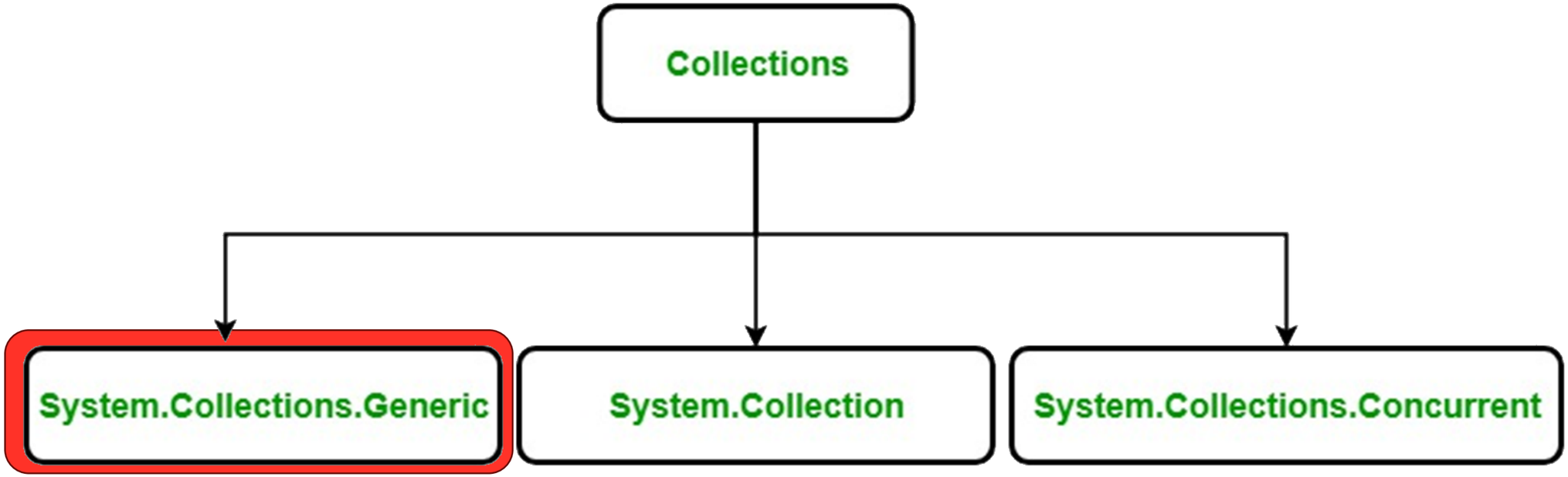
# Collections C#

- Collections provide **standardized ways** for a program **to manage groups of objects**.

- They are a **set of classes** designed to hold elements in a general, reusable way.

- Using **collections** you can **store**, **update**, **delete**, **retrieve**, **search**, **sort**, and perform **other common operations** on those objects.

# Collections C#




**Source**: https://www.geeksforgeeks.org

# System.Collections.Generic

- **System.Collections.Generic** is a .NET namespace with type-safe, generic collection classes and interfaces.
- It avoids boxing (convertion) for value types and gives compile-time type safety.
- Use it for most collections in modern C# code.

# System.Collections.Generic

- List<T>
- LinkedList<T>
- SortedList<T>

- Queue<T>
- Stack<T>

- ~~ArrayList~~<T>: It's an older collection, used only with legacy applications.

# List

- A dynamic array (backed by a contiguous T[]).
- Fast indexed access and iteration.
- Use when you need random access or most operations are adding/iterating.

# List

- A dynamic array (backed by a contiguous T[]).
- Fast indexed access and iteration.
- Use when you need random access or most operations are adding/iterating.

# LinkedList

- A doubly linked list (nodes with prev/next).
- Fast insert/remove when you have a node reference, but slow indexed access.
- Use when you need cheap inserts/removes in the middle and stable node identity.

# SortedList

- SortedList<TKey,TValue>: A collection that stores key/value pairs sorted by key.
- Internally it uses **two arrays** (one for keys, one for values) kept in sorted order.
- Use when you need fast indexed access by position (it exposes Keys and Values collections with indexers) and when the data is relatively static or small, so the cost of shifting is acceptable.

# Queue

- A FIFO (first-in, first-out) collection.
  - Enqueue adds to the tail;
  - Dequeue removes from the head.
- Good for processing items in arrival order (tasks, message buffers, …).
- Use when order matters and you always remove the oldest item.

# Stack

- A LIFO (last-in, first-out) collection.
  - Push adds to the top;
  - Pop removes the most recently added item.
- Good for reversing order, backtracking algorithms, or managing nested/recursive state (function call simulation, undo stacks)
- Use when you always need the most recent item first.

# Links

- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/collections
- https://www.geeksforgeeks.org/c-sharp/collections-in-c-sharp/
- https://www.codecademy.com/learn/learn-intermediate-c-sharp/modules/c-sharp-collections/cheatsheet