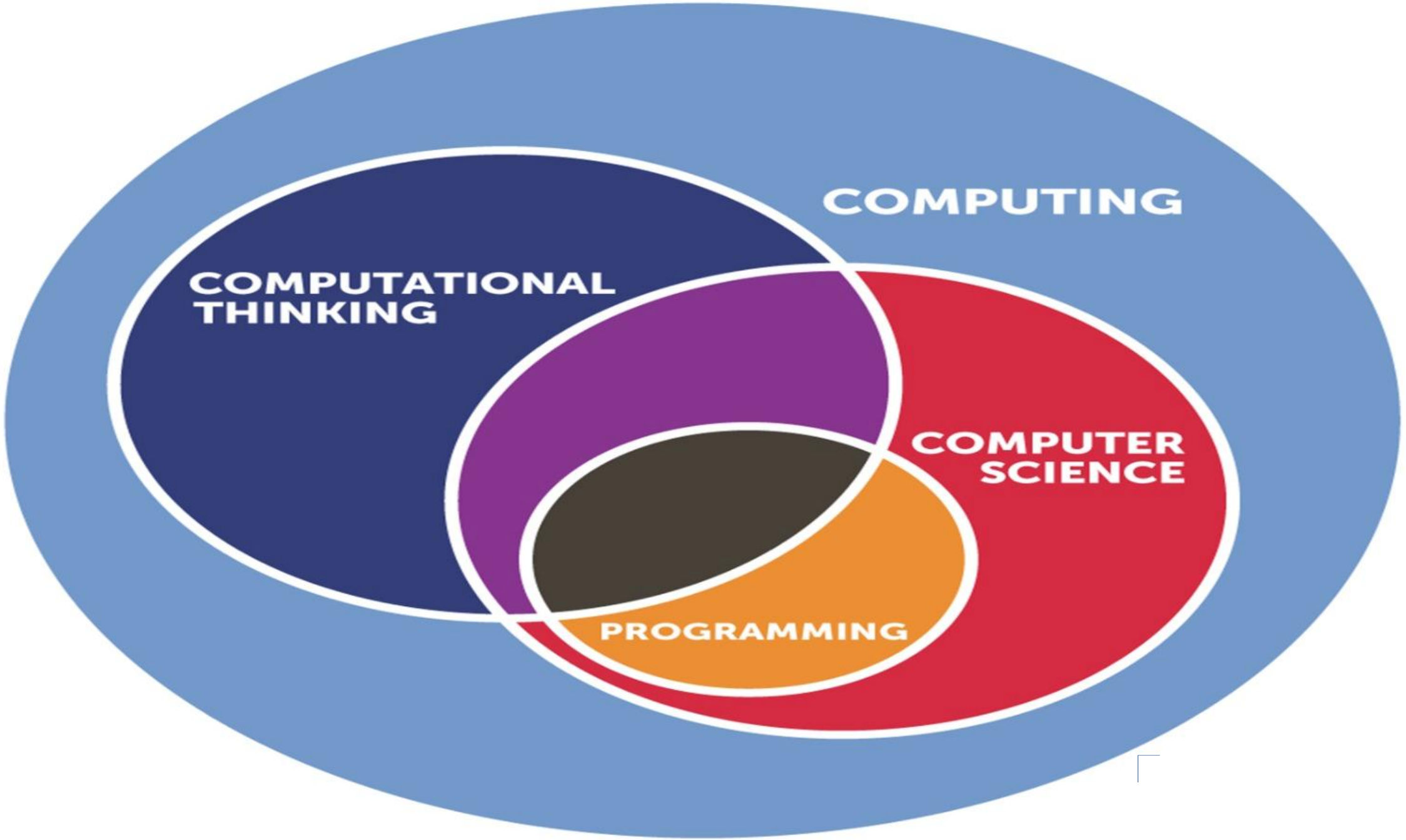
Kiran Waghmare

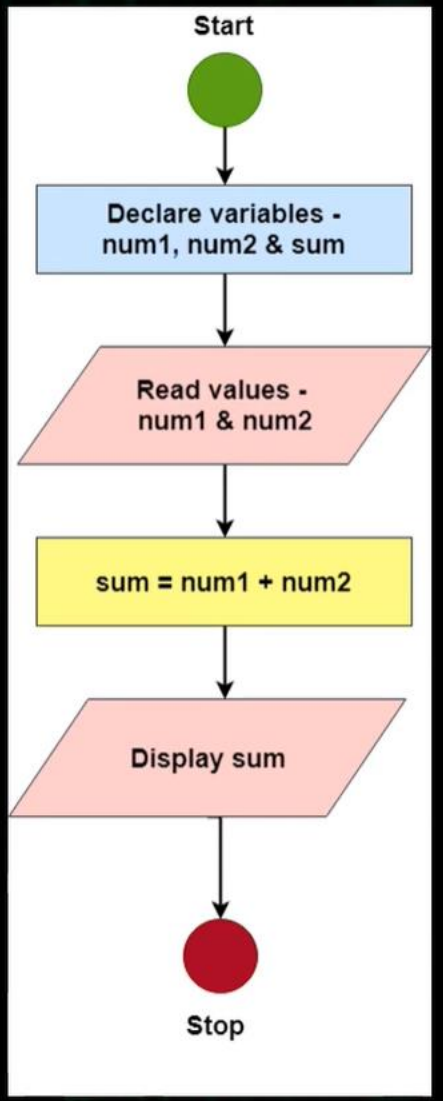
CDAC Mumbai


COMPUTING
COMPUTATIONAL THINKING
COMPUTER SCIENCE
PROGRAMMING

Example of an Algorithm in Programming –

Write an algorithm to add two numbers entered by user. –

1. Step 1: Start
2. Step 2: Declare variables num1, num2 and sum.
3. Step 3: Read values num1 and num2.
4. Step 4: Add num1 and num2 and assign the result to sum.(sum←num1+num2)
5. Step 5: Display sum
6. Step 6: Stop



Analysis of Algorithms

- **How good is the algorithm?**
  - Correctness
  - Time efficiency
  - Space efficiency

- **Does there exist a better algorithm?**
  - Lower bounds
  - Optimality

Analysis of Algorithms

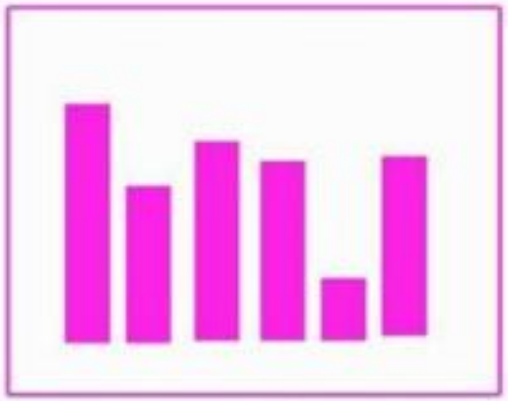
- An algorithm is said to be efficient and fast, if **it takes less time to execute and consumes less memory space**.
- The performance of an algorithm is measured on the basis of following properties :

1.Time Complexity

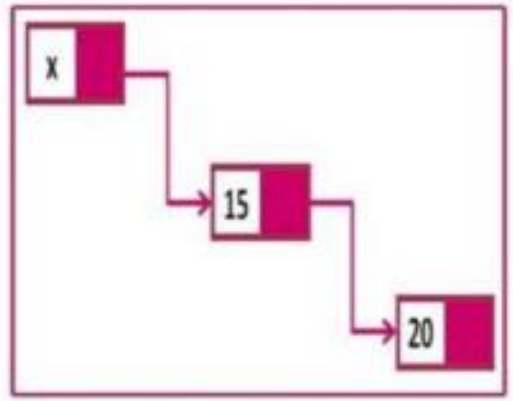
2.Space Complexity

Some Well-known Computational Problems

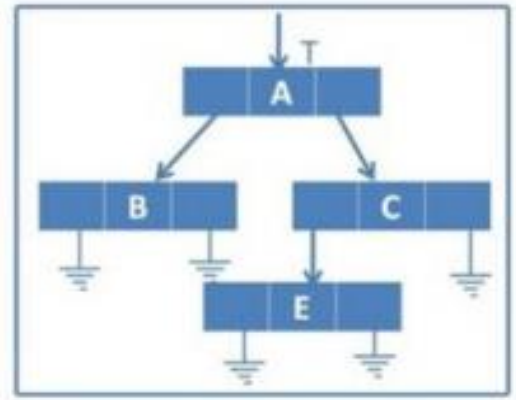
- **Sorting**
  - e.g., school days…height wise, now rotation wise
- **Searching**
  - E.g. read books. Alexa, google
- **Shortest paths in a graph**
- **Minimum spanning tree**
- **Primality testing**
- **Traveling salesman problem**
- **Knapsack problem**
- **Chess**
- **Towers of Hanoi**



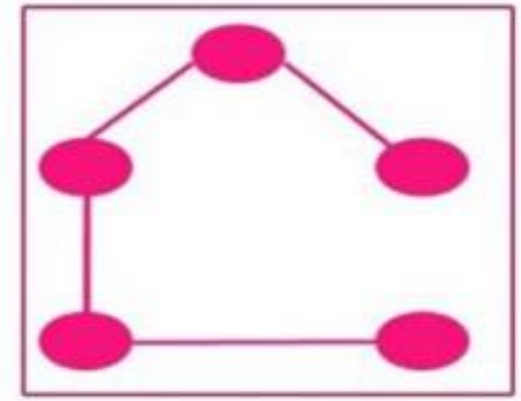
Sorting



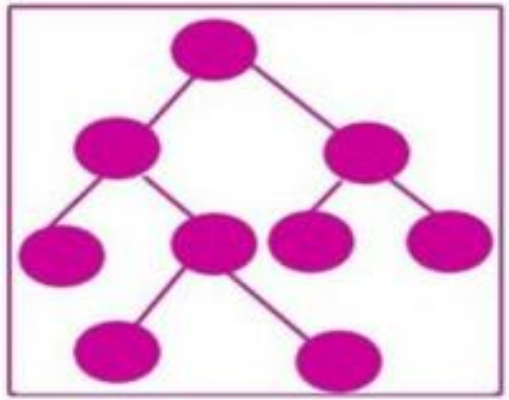
Link list



list

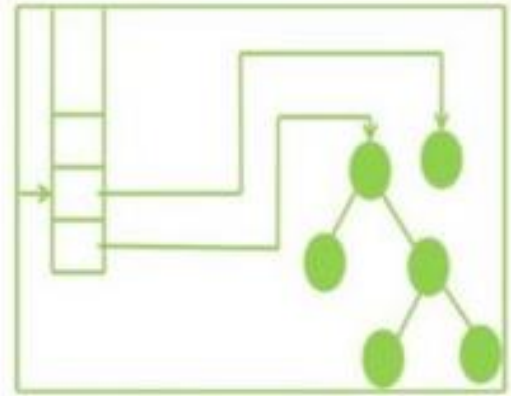


spanning tree

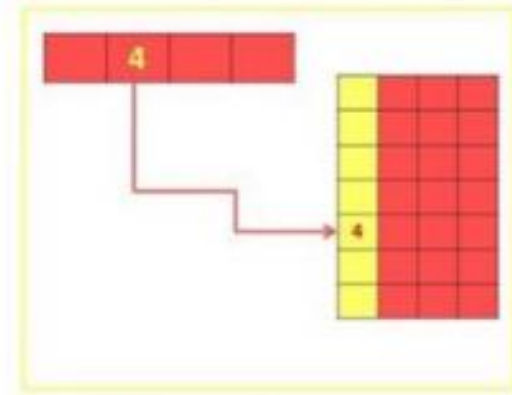


Tree

Graph



Stack



Hashing

By...navinkumardhoprephotography.com

Types of Data Structures

- Data structures can broadly be categorized into two types:
  - Primitive Data Structures:
    - These are the basic data types that are directly operated upon by the machine instructions. Examples include integers, floats, characters, etc.
  - Abstract Data Structures (ADS):
    - These are complex data structures that are built using primitive data types. They provide more flexibility and are implemented using programming languages. Examples include arrays, linked lists, stacks, queues, trees, graphs, hash tables, etc.

Common Data Structures and Their Applications

Let's explore some common abstract data structures and their practical applications:

- **1. Arrays**
- **Description**: Arrays are a collection of elements of the same data type stored in contiguous memory locations.
- **Applications**:
  - Storing and accessing sequential data (e.g., list of student grades).
  - Implementing matrices for mathematical operations.
  - Sorting algorithms like quicksort and mergesort.
- **2. Linked Lists**
- **Description**: Linked lists are a linear data structure where each element is a separate object (node) containing data and a reference (link) to the next node.
- **Applications**:
  - Implementing dynamic memory allocation.
  - Implementing stacks and queues.
  - Undo functionality in text editors.

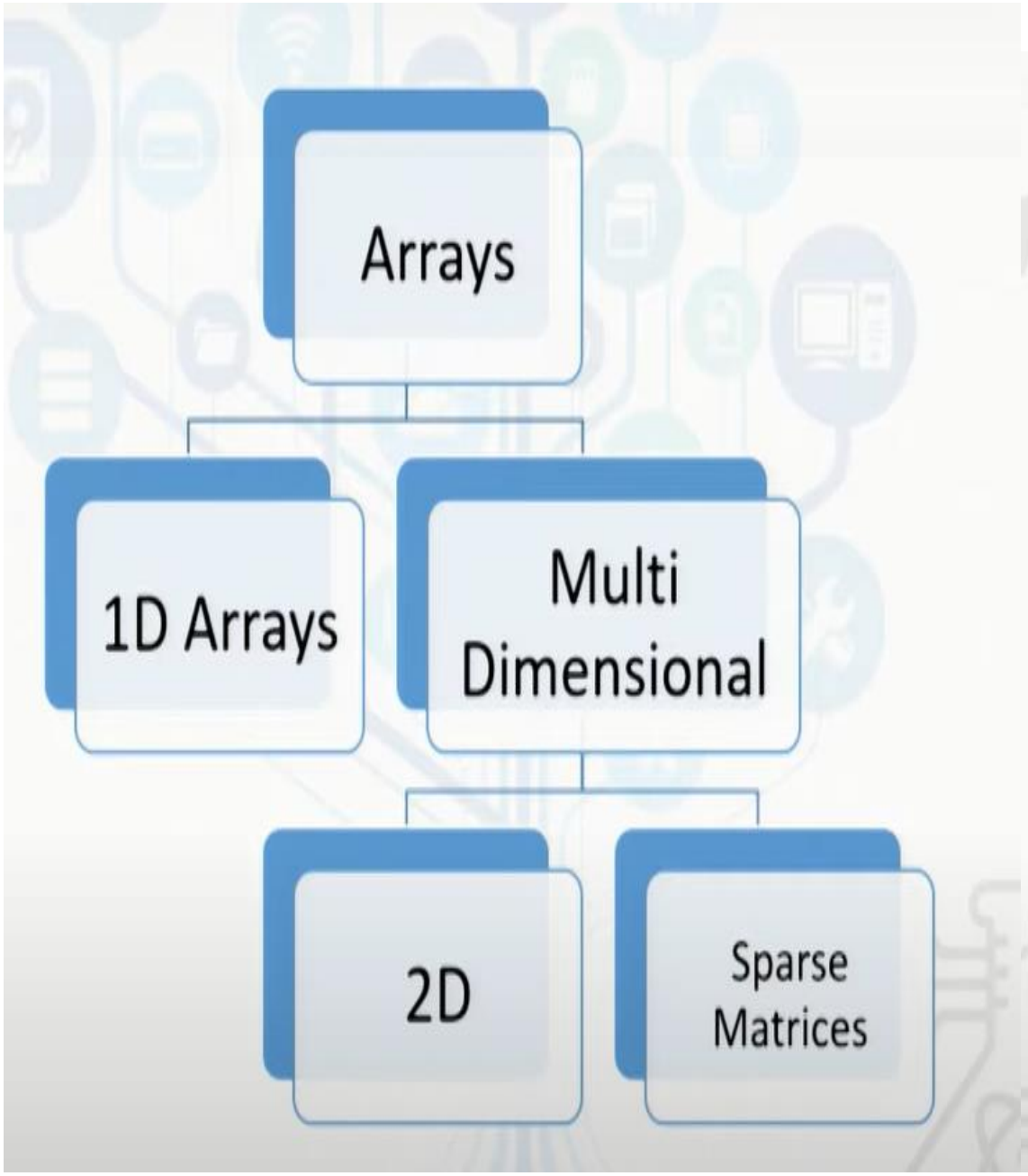
- **3. Stacks**
- **Description**: Stacks are a collection of elements with Last In First Out (LIFO) order.
- **Applications**:
  - Implementing function calls (call stack).
  - Expression evaluation and syntax parsing.
  - Backtracking algorithms.
- **4. Queues**
- **Description**: Queues are a collection of elements with First In First Out (FIFO) order.
- **Applications**:
  - Job scheduling in operating systems.
  - Breadth-first search (BFS) in graphs.
  - Print spooling in printers.

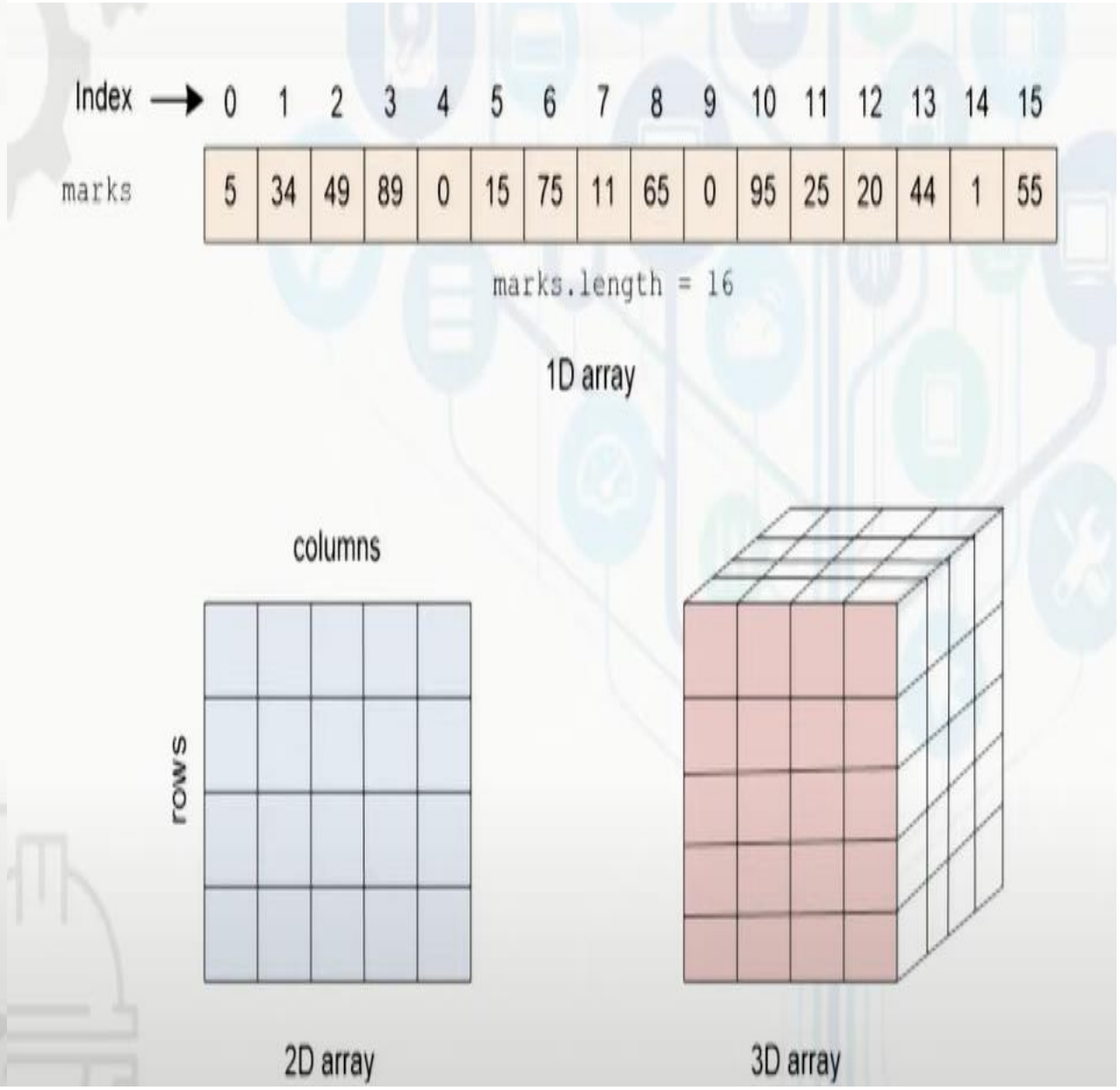
- **5. Trees**
- **Description**: Trees are hierarchical data structures consisting of nodes connected by edges.
- **Applications**:
  - Representing hierarchical data (e.g., file systems).
  - Binary Search Tree (BST) for efficient searching, insertion, and deletion.
  - Decision-making algorithms (e.g., minimax algorithm in game theory).
- **6. Graphs**
- **Description**: Graphs are a collection of nodes (vertices) and edges connecting them.
- **Applications**:
  - Social networks (friendship connections).
  - Shortest path algorithms (Dijkstra's algorithm).
  - Network flow algorithms.

- **7. Hash Tables**
- **Description**: Hash tables (or hash maps) store data in key-value pairs and use hash functions to compute an index where an element is stored or retrieved.
- **Applications**:
  - Implementing associative arrays (e.g., dictionary).
  - Database indexing.
  - Caching and memoization techniques.

Real-Life Applications of Array:

- An array is frequently used to store data for mathematical computations.
- It is used in image processing.
- It is also used in record management.
- Book pages are also real-life examples of an array.
- It is used in ordering boxes as well.


Arrays
1D Arrays
Multi Dimensional
2D
Sparse Matrices

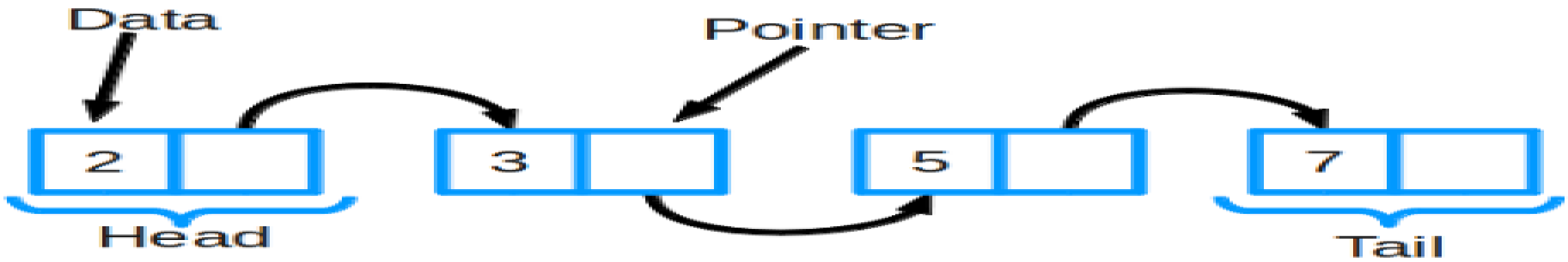

Index → 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
marks 5 34 49 89 0 15 75 11 65 0 95 25 20 44 1 55
marks.length = 16
1D array
columns
rows
2D array
3D array

Types of Array

- **One-dimensional arrays:**
  - Store a single row of elements.
- **Multidimensional arrays:**
  - Store multiple rows of elements.
- **Array Operations**
  - Traversal:
    - Visiting each element in a specific order (e.g., sequential, reverse).
  - Insertion:
    - Adding a new element at a specific index.
  - Deletion:
    - Removing an element from a specific index.
  - Searching:
    - Finding the index of an element.

- **Applications of Array**
- **Data storage:**
  - For processing purposes.
- **Implementing data structures:**
  - Stacks, queues, etc.
- **Data representation:**
  - Tables, matrices.
- **Creating dynamic data structures:**
  - Linked lists, trees.

Linked list



Linked List

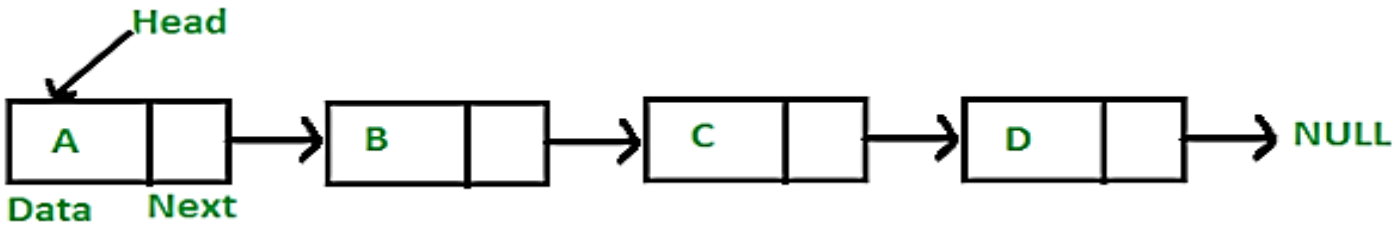
- **A linked list is a sequence of data structures, which are connected together via links.**

- **Linked List is a sequence of links which contains items.**
- **Each link contains a connection to another link.**
- **Linked list is the second most-used data structure after array.**
- **Following are the important terms to understand the concept of Linked List.**

1. **Link** – Each link of a linked list can store a data called an **element**.
2. **Next** – Each link of a linked list contains a link to the next link called **Next.**
3. **LinkedLis**t – A Linked List contains the **connection link** to the first link called **First**.

Linked List Representation

- **Linked list can be visualized as a chain of nodes, where every node points to the next node.**



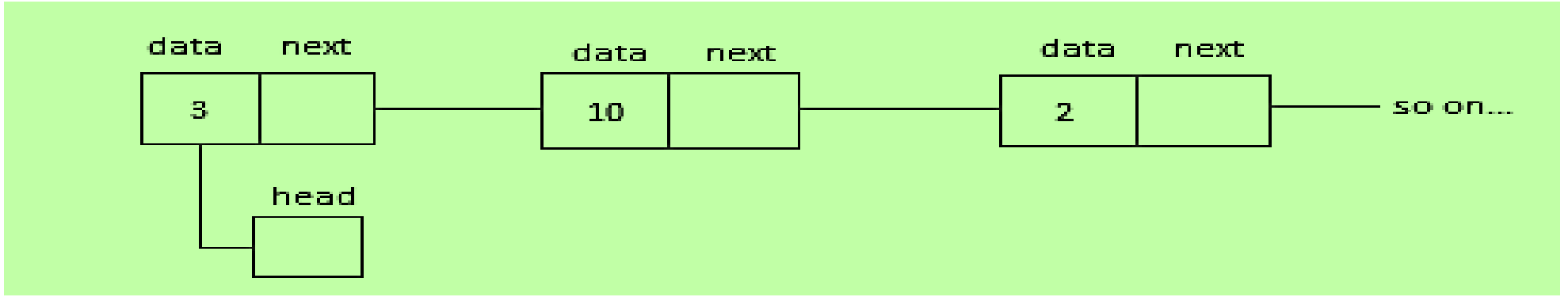
- **As per the above illustration, following are the important points to be considered.**

1. Linked List contains a **link element** called **first.**
2. Each link carries a **data field(s)** and a **link field** called **next.**
3. Each link is **linked with its next link** using its **next link.**
4. **Last link carries a link as null** to mark the end of the list.

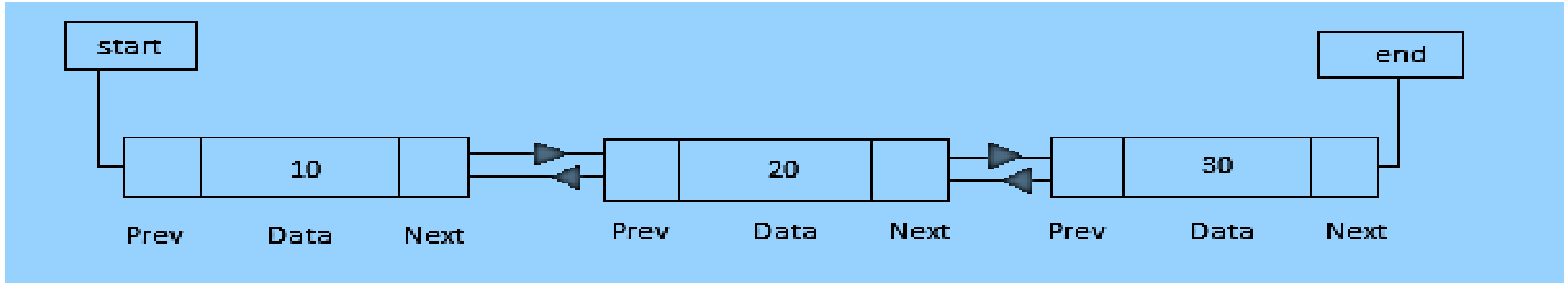
Types of Linked List

- **Following are the various types of linked list.**
  1. **Simple Linked List** – Item navigation is forward only.
  2. **Doubly Linked List** – Items can be navigated forward and backward.
  3. **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

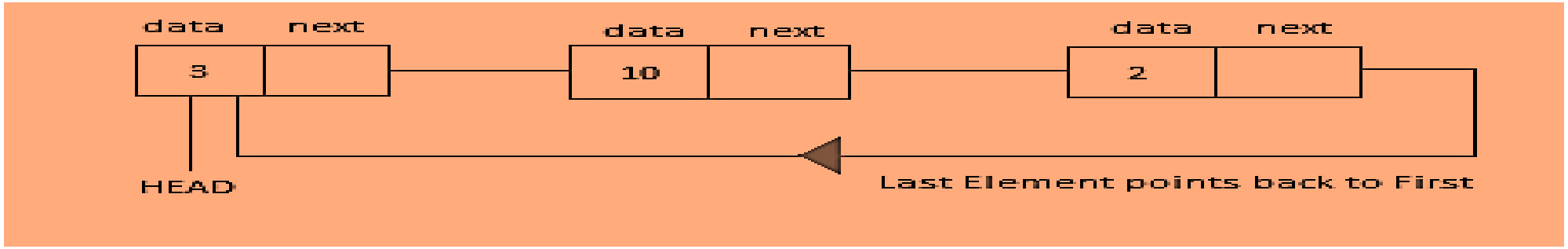
- **Simple Linked List**



- **Doubly Linked List**

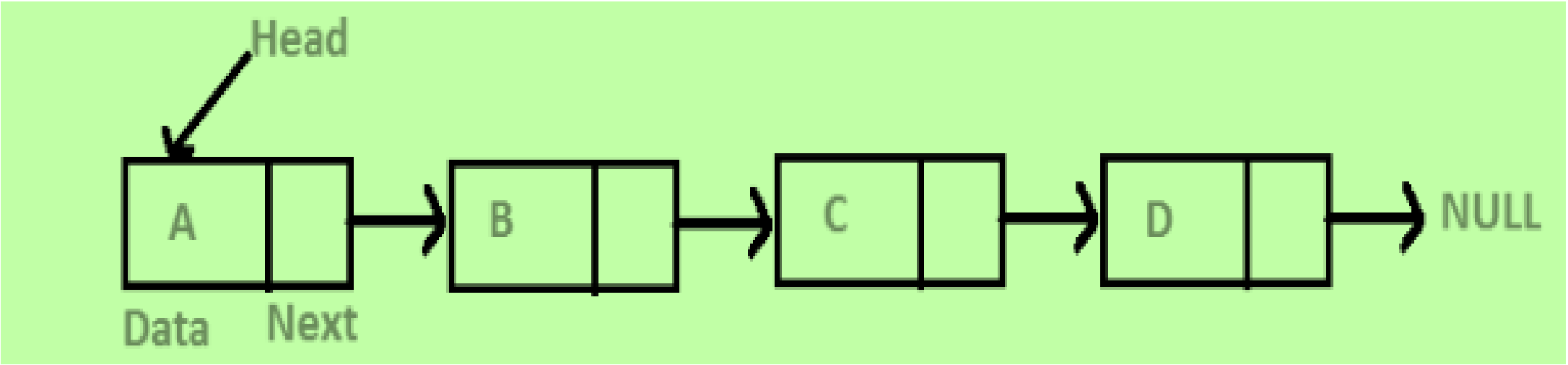


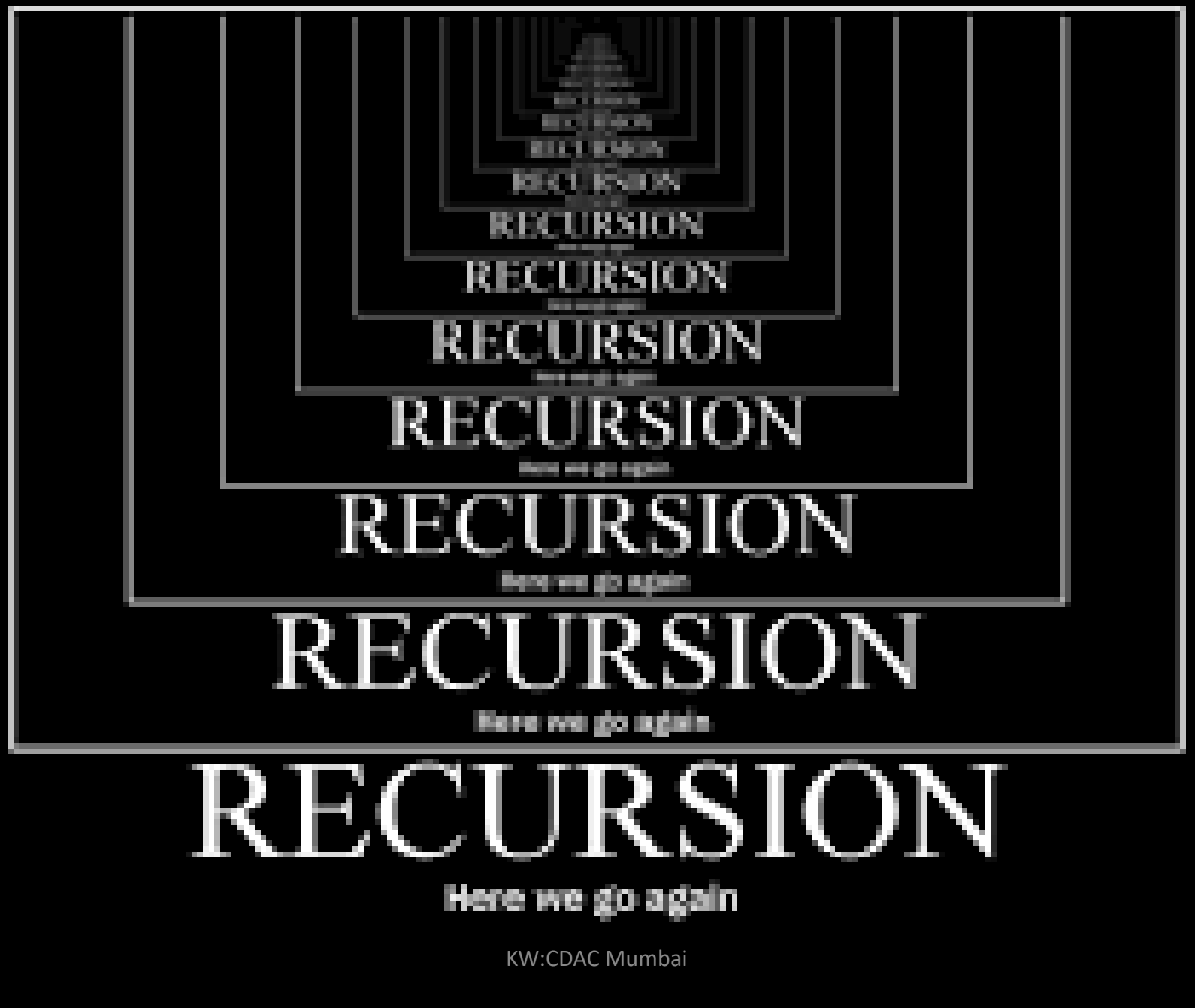
- **Circular Linked List**

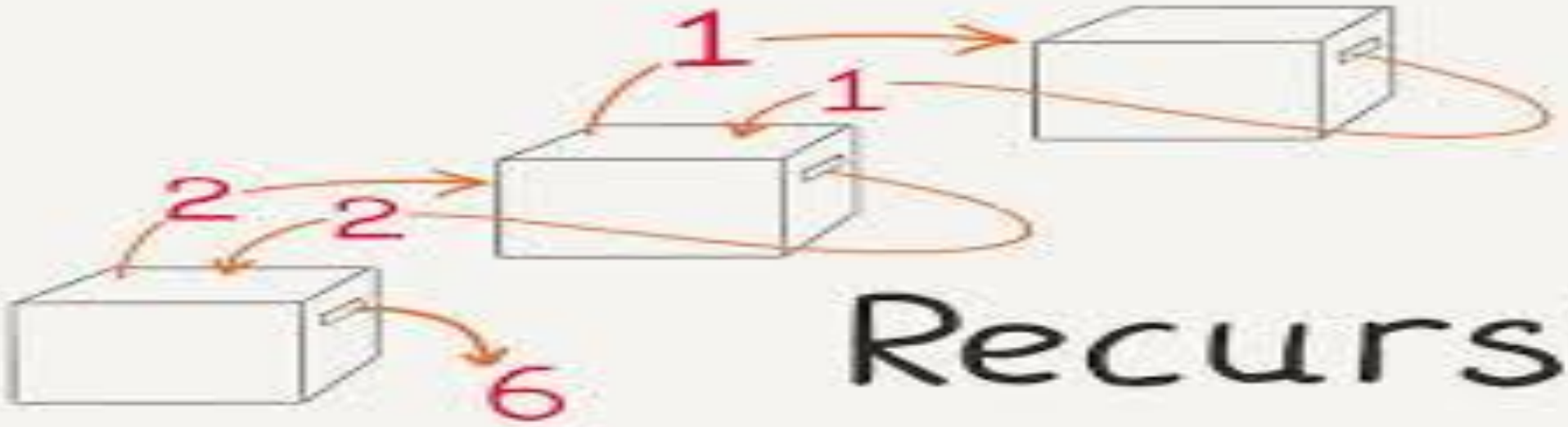


Singly Linked List

- Singly Linked Operations: Insert, Delete, Traverse, search, Sort, Merge



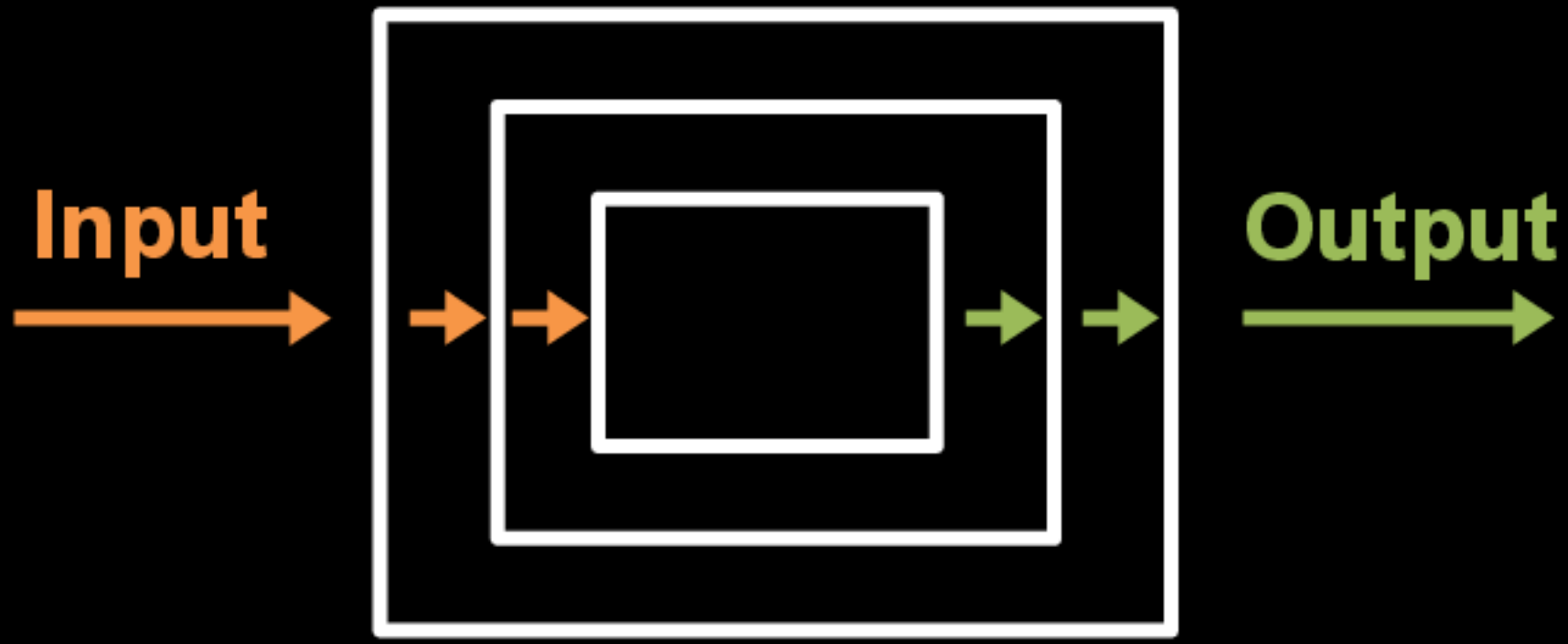

RECURSION
Here we go again



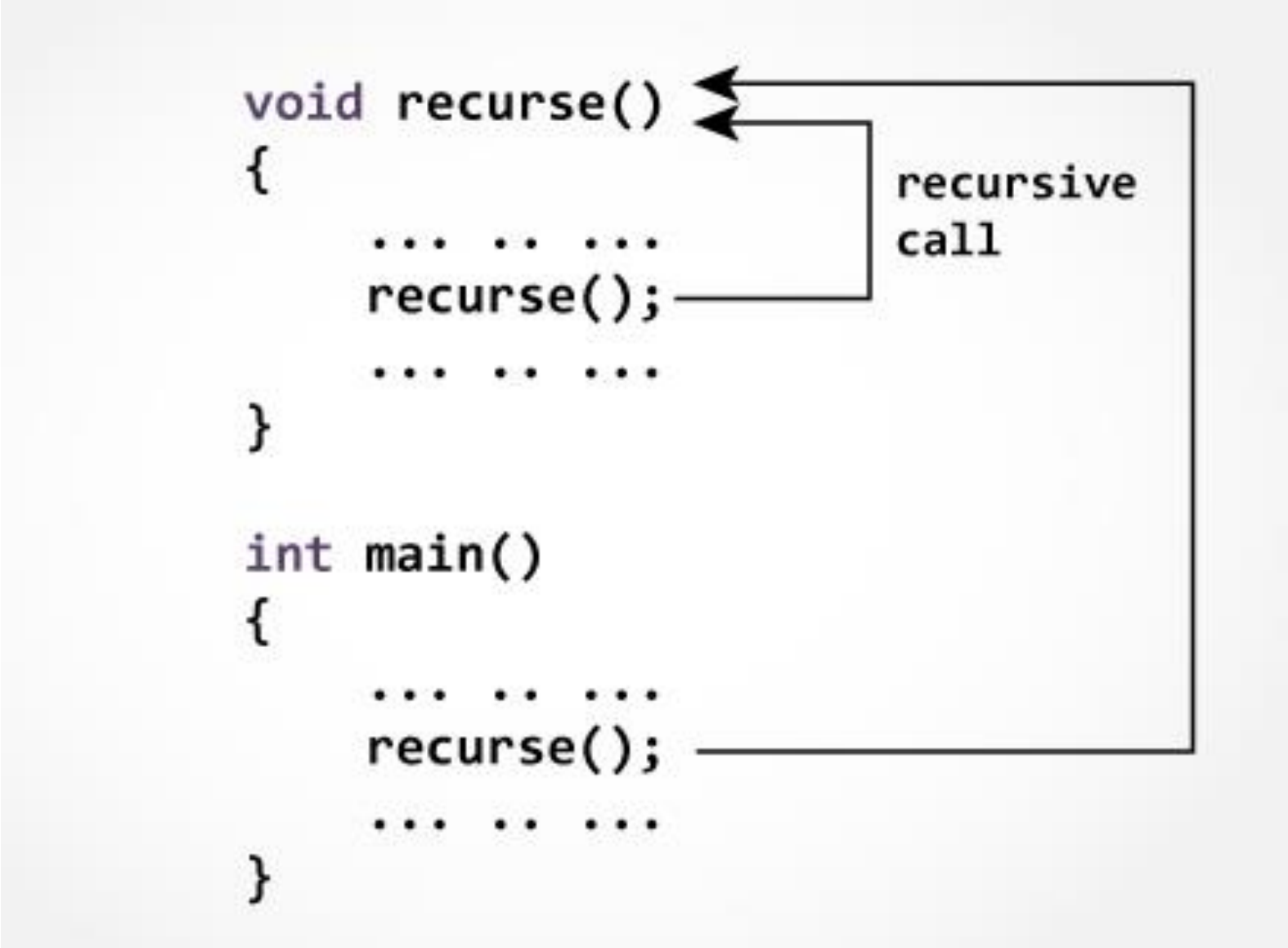
Topics

1. Recursive definitions and Processes
2. Writing Recursive Programs
3. Efficiency in Recursion
4. Towers of Hanoi problem.

Recursion



How does Recursion works?



Recursion

- Any function which calls itself directly or indirectly is called Recursion and the corresponding function is called as recursive function.
- A recursive method solves a problem by calling a copy of itself to work on a smaller problem.
- It is important to ensure that the recursion terminates.
- Each time the function call itself with a slightly simple version of the original problem.
- Using recursion, certain problems can be solved quite easily.
- E.g: Tower of Hanoi (TOH), Tree traversals, DFS of Graph etc.,

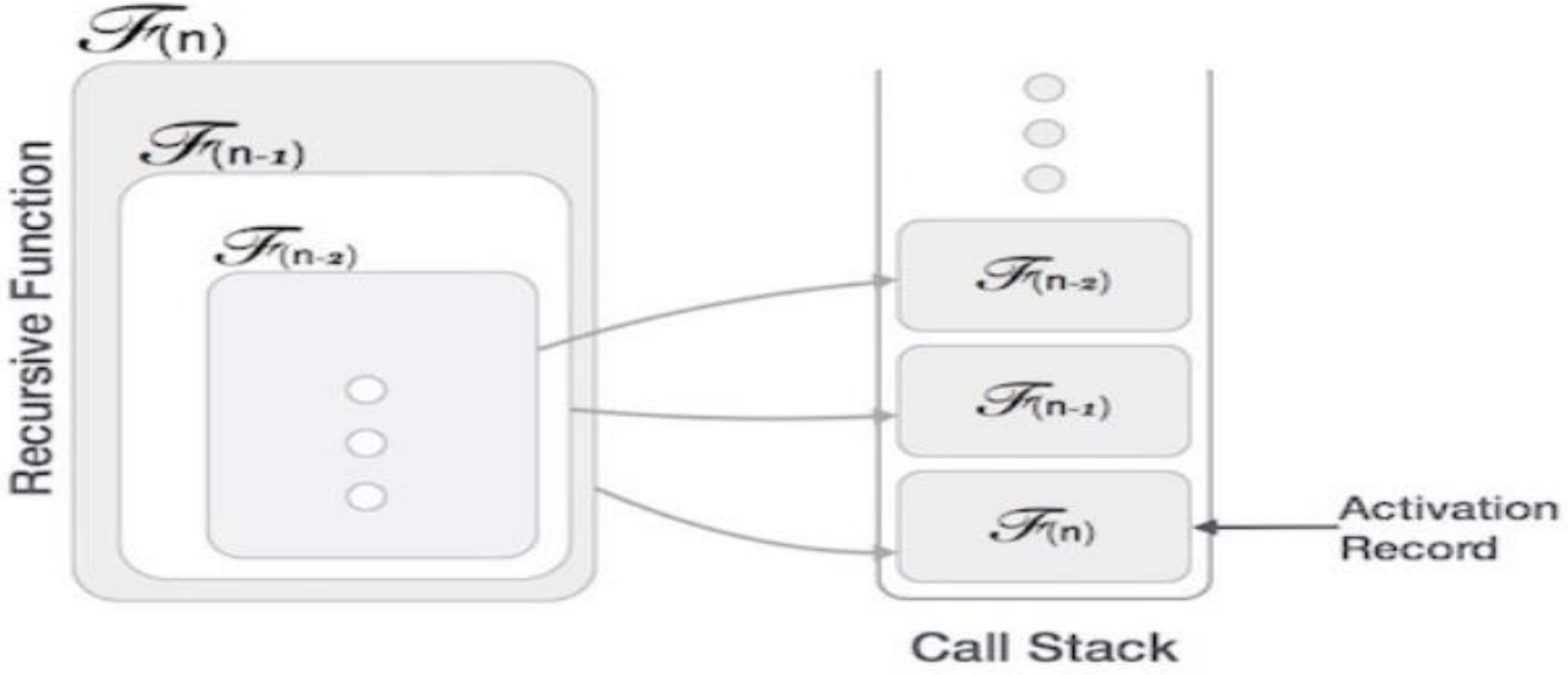
What is base condition in recursion?

- In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

- In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How Data Structure Recursive function is implemented?



Why Algorithms?

- Fibonacci numbers
  - Compute first N Fibonacci numbers using iteration.
  - … using recursion.
- Write the code.
- Try for N=5, 10, 20, 50, 100
- What do you see? Why does this happen?