Arrays and Linked Lists

Outline

- Types of Data Structure
- Abstract Data Type
- Array
- Linked Lists
  - Singly linked lists
  - Doubly linked lists
  - Circular linked lists

Types of Data Structure

- **Linear: In Linear data structure, values are arranged in linear fashion.**
  - **Array: Fixed-size**
  - **Linked-list: Variable-size**
  - **Stack: Add to top and remove from top - LIFO**
  - **Queue: Add to back and remove from front - FIFO**
  - **Priority queue: Add anywhere, remove the highest priority**

Types of Data Structure

- **Non-Linear: The data values in this structure are not arranged in order.**
  - **Hash tables: Unordered lists which use a ‘hash function’ to insert and search**
  - **Tree: Data is organized in branches.**
  - **Graph: A more general branching structure, with less strict connection conditions than for a tree**

Type of Data Structures

- **Homogeneous: In this type of data structures, values of the same types of data are stored.**
  - **Array**
- **Heterogeneous: In this type of data structures, data values of different types are grouped and stored.**
  - **Structures**
  - **Classes**

Abstract Data Type and Data Structure

- **Definition:-**
  - ***Abstract Data Types (ADTs)* stores data and allow various operations on the data to access and change it.**
  - **A mathematical model, together with various operations defined on the model**
  - **An ADT is a collection of data and associated operations for manipulating that data**
- **Data Structures**
  - **Physical implementation of an ADT**
  - **data structures used in implementations are provided in a language *(primitive* or *built-in)* or are built from the language constructs *(user-defined)***
  - **Each operation associated with the ADT is implemented by one or more subroutines in the implementation**

Abstract Data Type

- ADTs support *abstraction*, *encapsulation*, and *information hiding*.
- *Abstraction* is the structuring of a problem into well-defined entities by defining their data and operations.
- The principle of hiding the used data structure and to only provide a well-defined interface is known as *encapsulation*.

The Core Operations of ADT

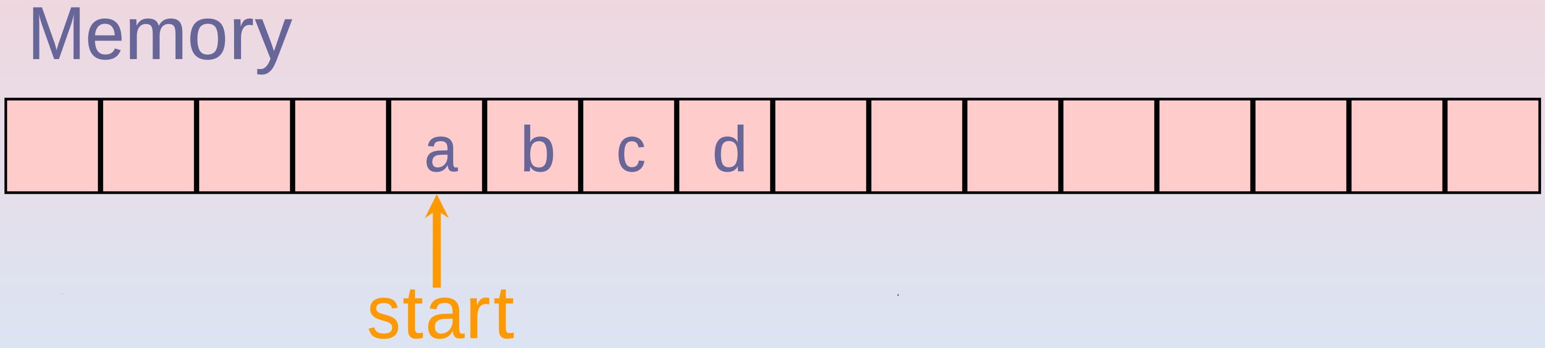
- **Every Collection ADT should provide a way to:**
  - ***add*** **an item**
  - ***remove*** **an item**
  - ***find*****,** ***retrieve*****, or** ***access*** **an item**

Is there a data structure that works well for all purpose?

- No single data structure works well for all purposes
- So it is important to know the strengths and limitations of several of them

Array

Values	2	5	1	9	6	7
Index	0	1	2	3	4	5



- Storing data in a sequential memory locations
- Access each element using integer index
- Very basic, popular, and simple
- int a[10]; int *a = new int(10);

Array

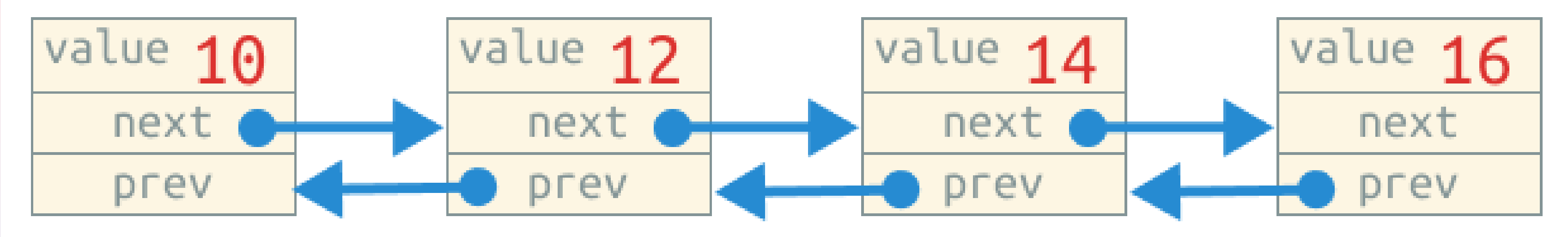
- **Array representation: You can access each value in constant time through its index.**
- **Arrays are a contiguous collection of elements that can be accessed randomly using an index.**
- **This access by index operation takes O(1) time.**
- **Insertions on an array have different times complexities.**
  - **O(1): constant time (on average) to append a value at the end of the array.**
  - **O(n): linear time to insert a value at the beginning or middle.**

Array: Problems

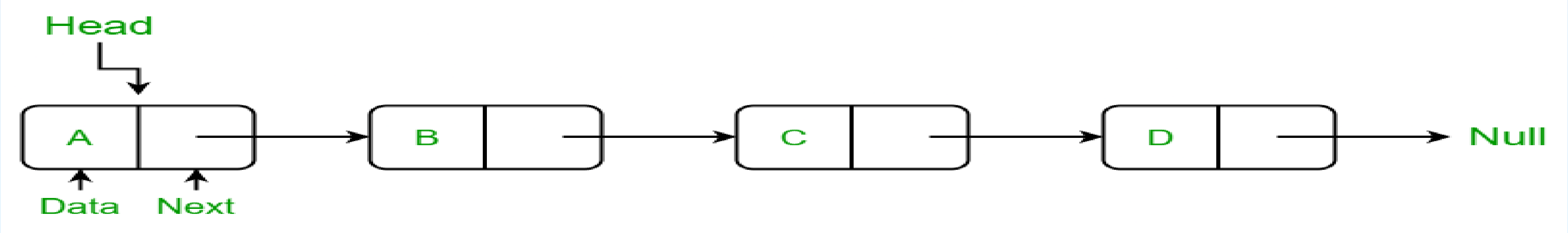
- New insertion and deletion: difficult
    - Need to shift to make space for insertion
    - Need to fill empty positions after deletion
- Why don’t we connect all elements just “logically” not “physically”?
    - Linked List

Linked List

- A List (or Linked List) is a **linear** data structure where each object has a **pointer** to the next one creating a **chain**. You can also have a back reference to the previous node.

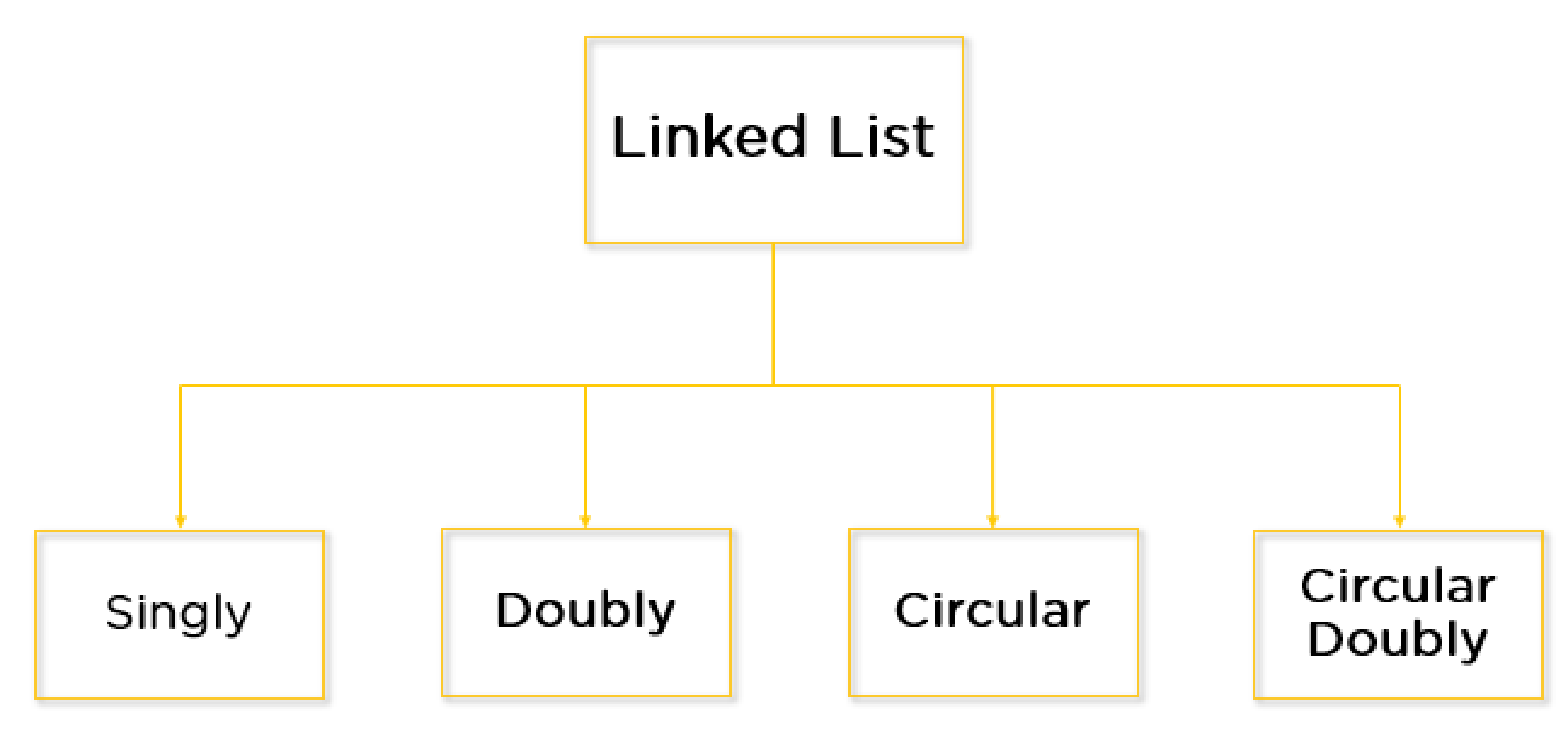


- The data doesn't have to be a number. It can be anything that you need (e.g., char, string, images, songs, menu items).



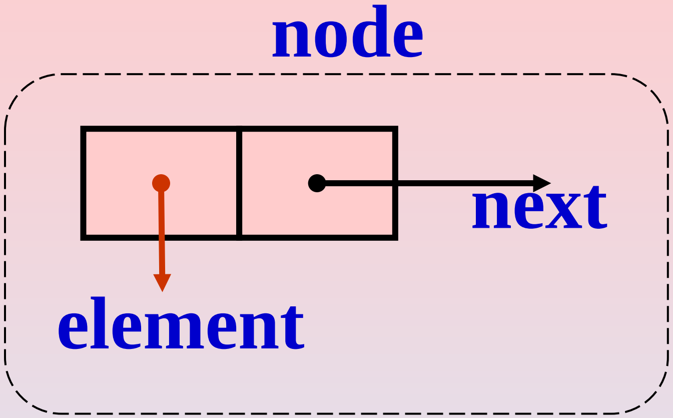
Types of Linked List

1. Singly linked lists.
2. Doubly linked lists.
3. Circular linked lists.
4. Circular doubly linked lists.

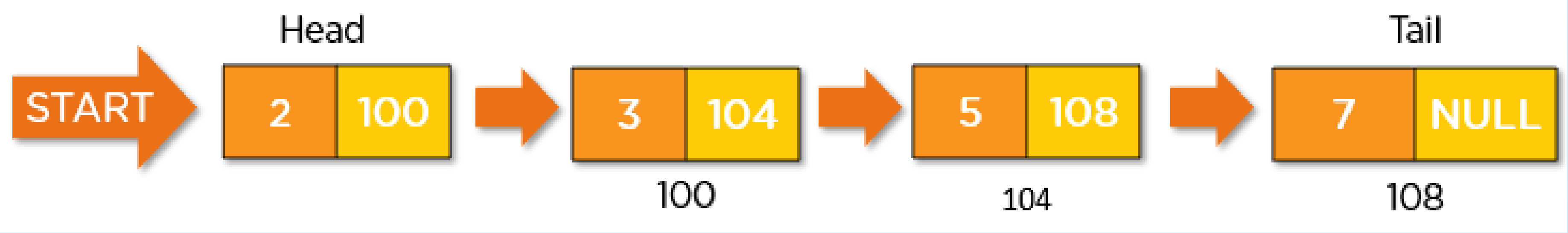


Singly Linked List

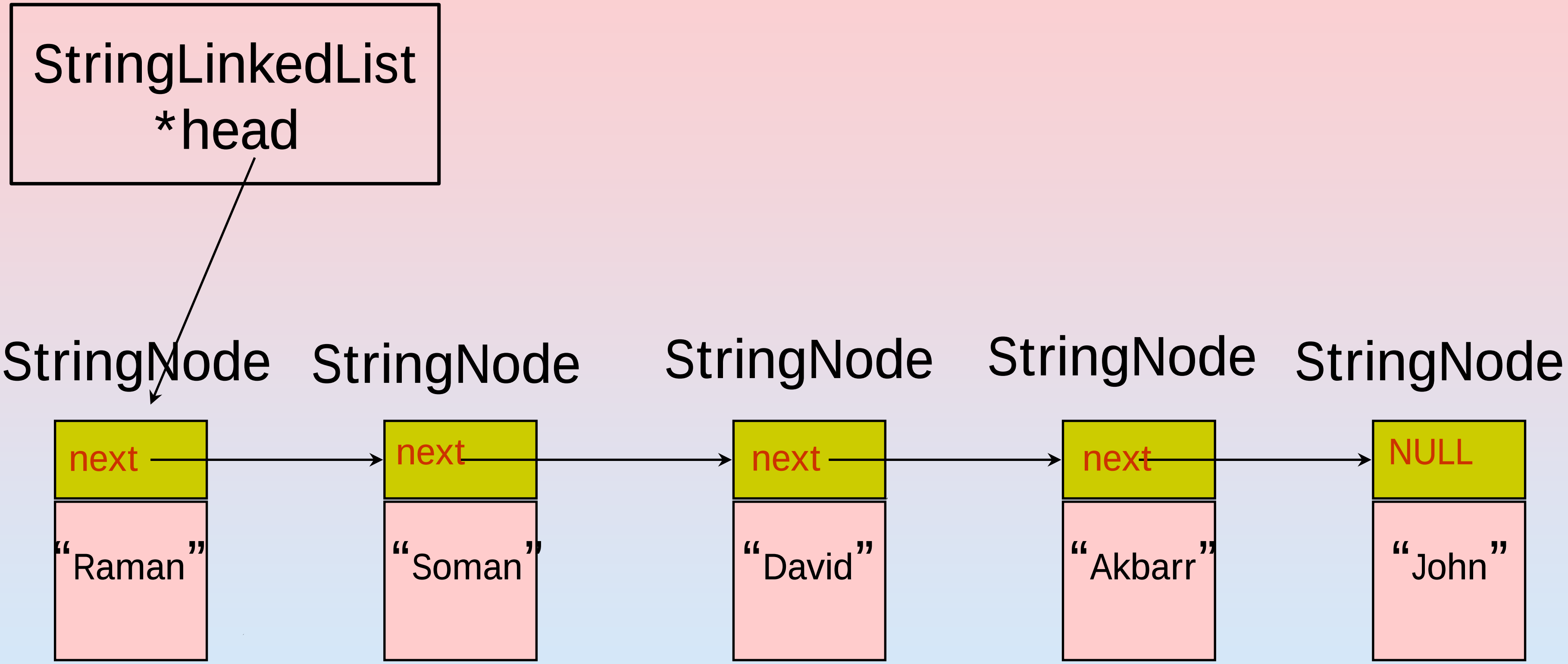
- A singly linked list is a concrete data structure consisting of a sequence of **nodes**
- Each node stores
  - element
  - link to the next node



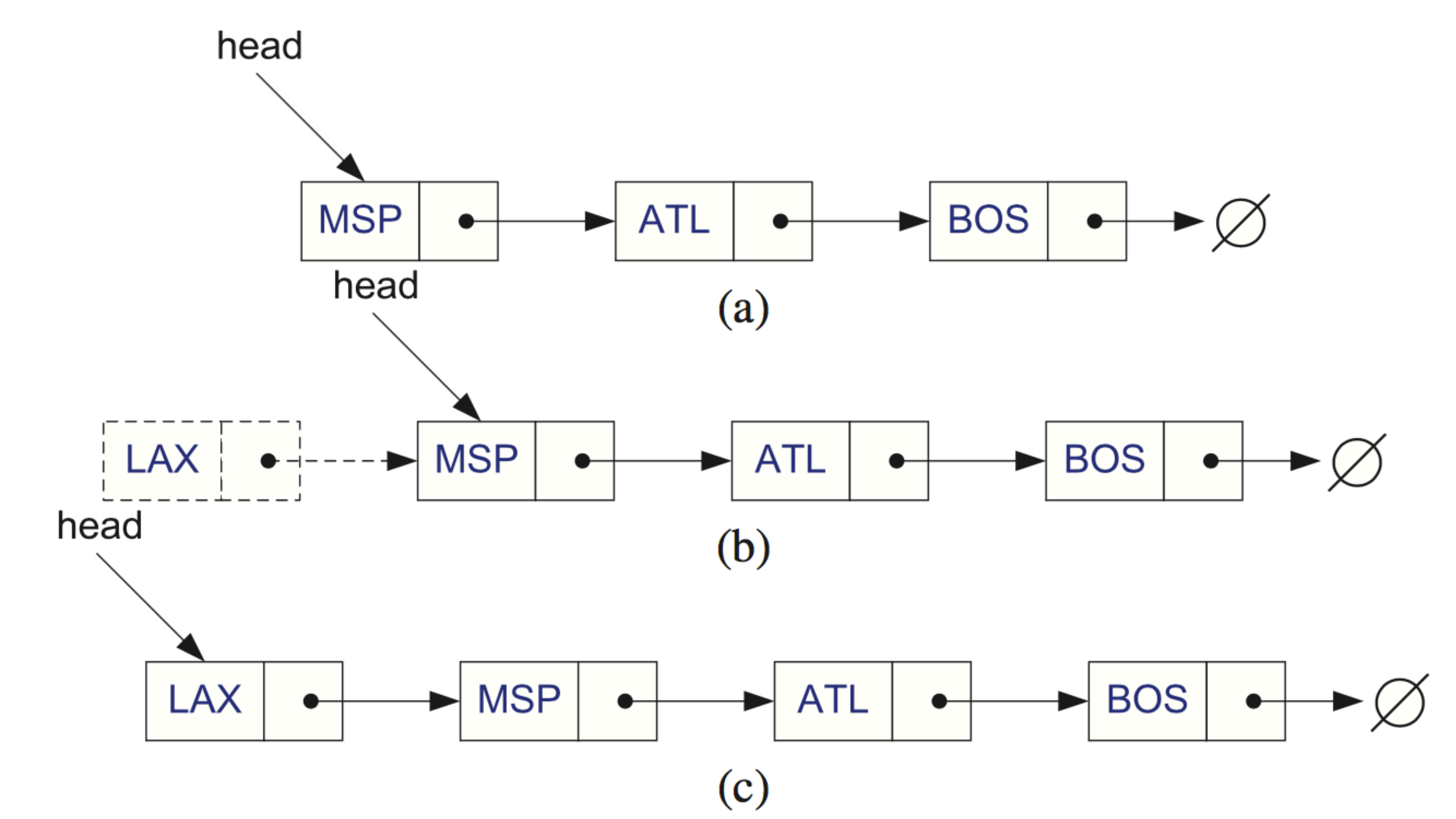
- A singly linked list is a unidirectional linked list.
- Traversal is possible only in one direction, i.e., from head node to tail node.



Singly Linked List of Strings: Picture

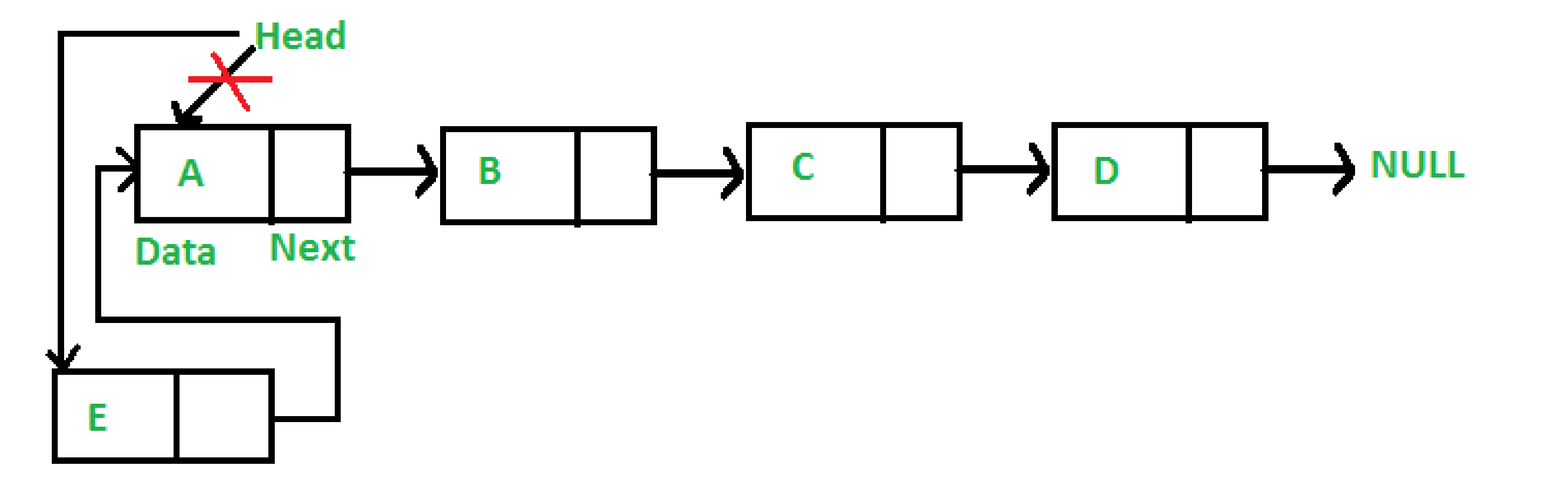


Inserting at the Head node

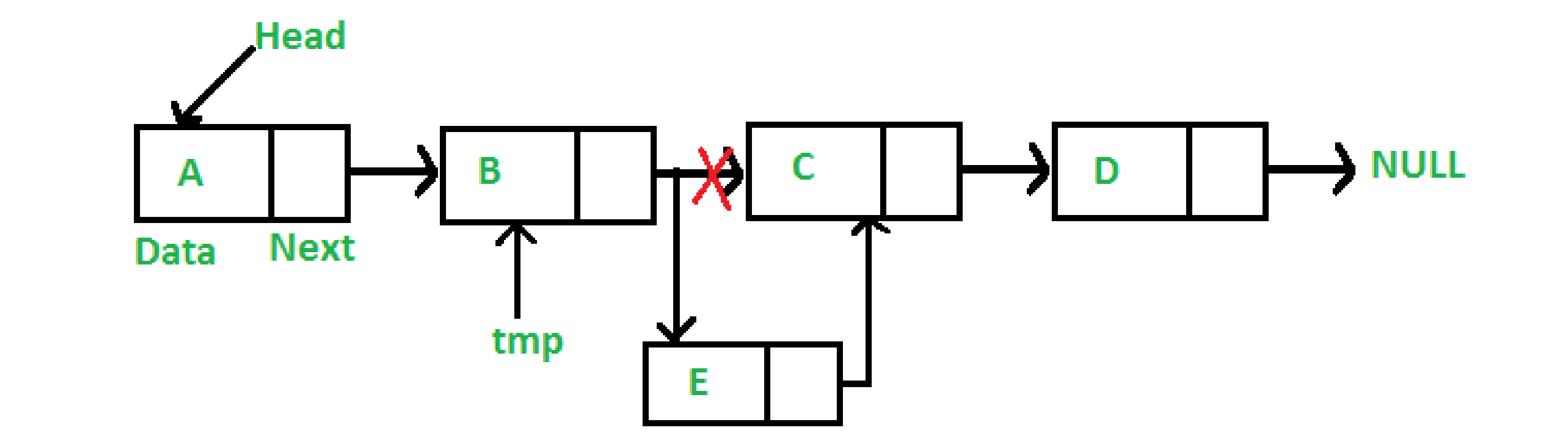


1. Allocate a new node
2. Insert a new element
3. Have the new node point to the old head
4. Update head to point to new node

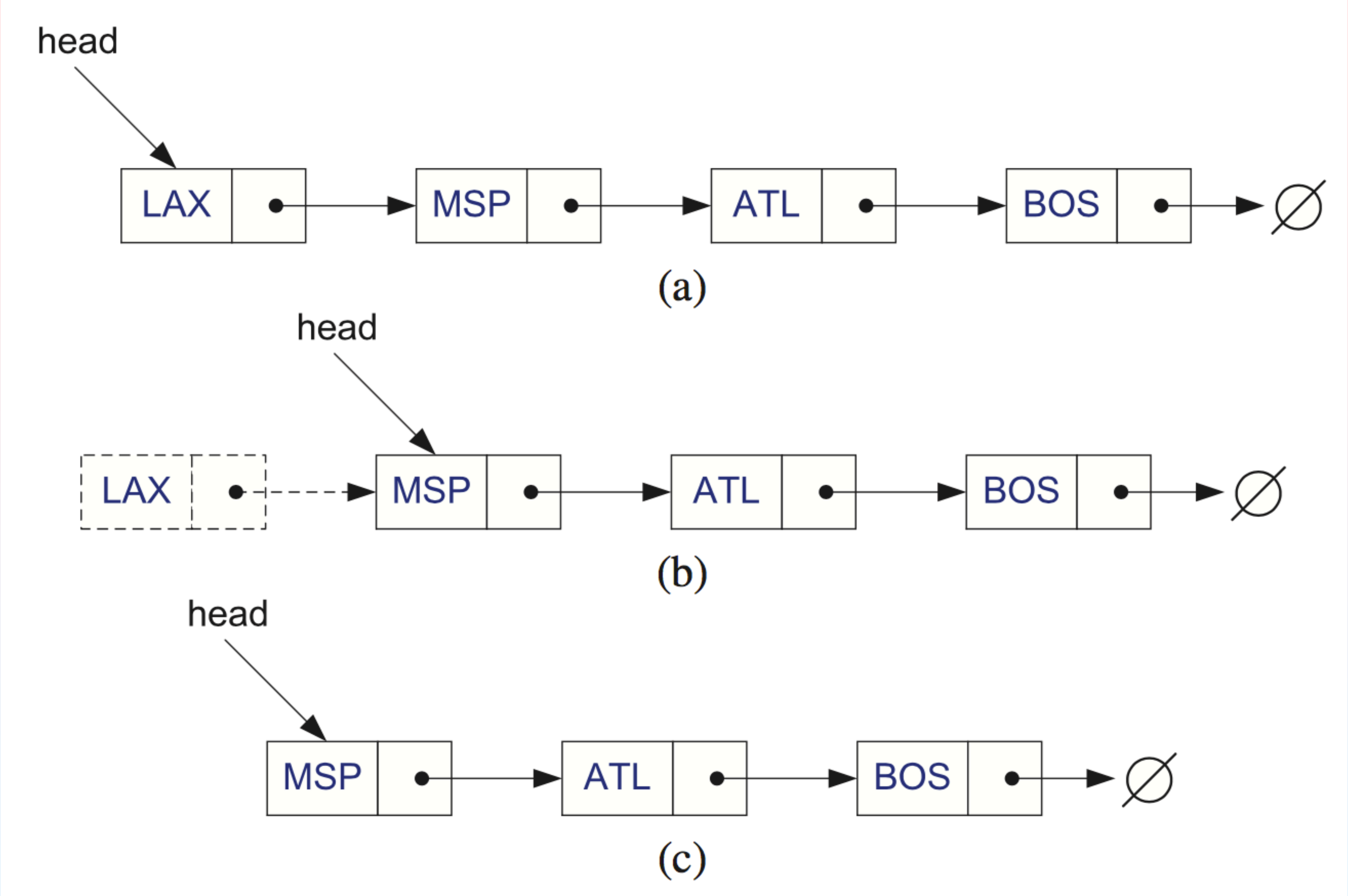
Inserting at the Head node



Inserting node after a given node



Removing at the Head



1. **Update head to point to next node in the list**
2. **Allow garbage collector to reclaim the former first node**

Inserting at the Tail and Removing at the Tail

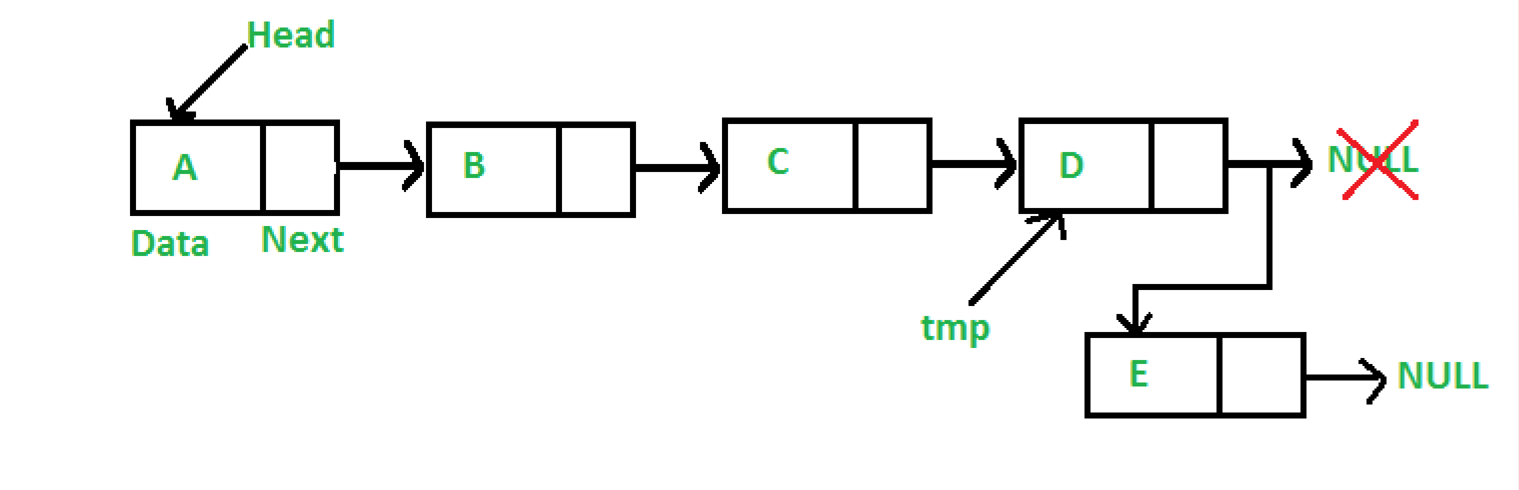
1. **Allocate a new node**
2. **Insert new element**
3. **Have new node point to null**
4. **Have old last node point to new node**
5. **Update tail to point to new node**

Insertion at the tail

1. …
2. ...
3. …
4. …

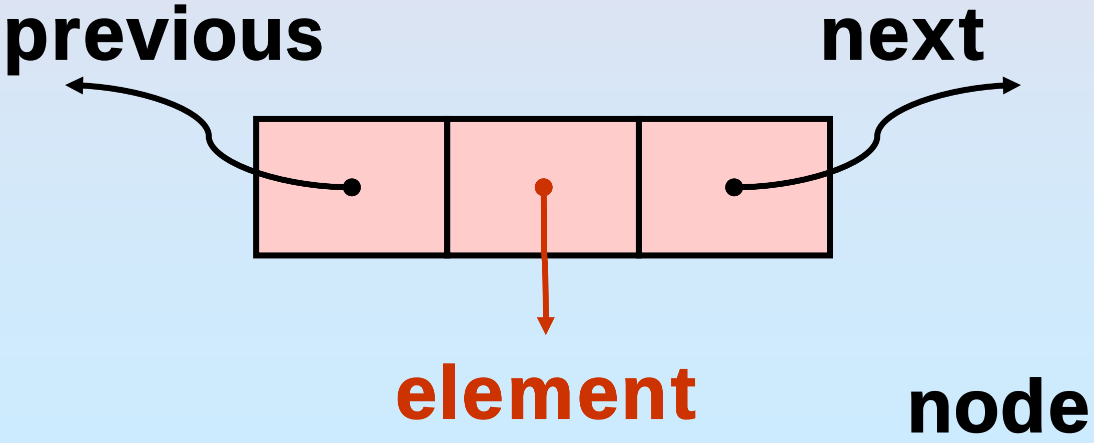
Removal at the tail

Inserting node at the Tail node

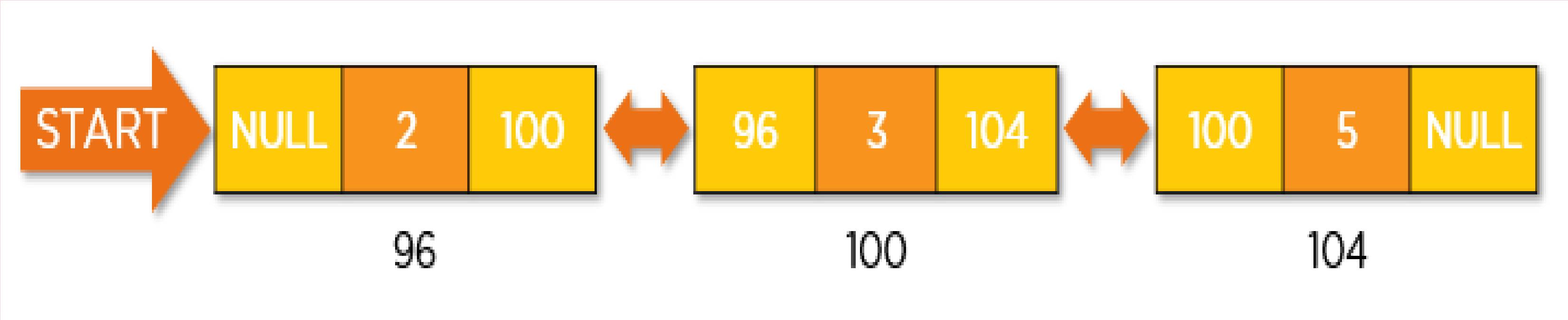


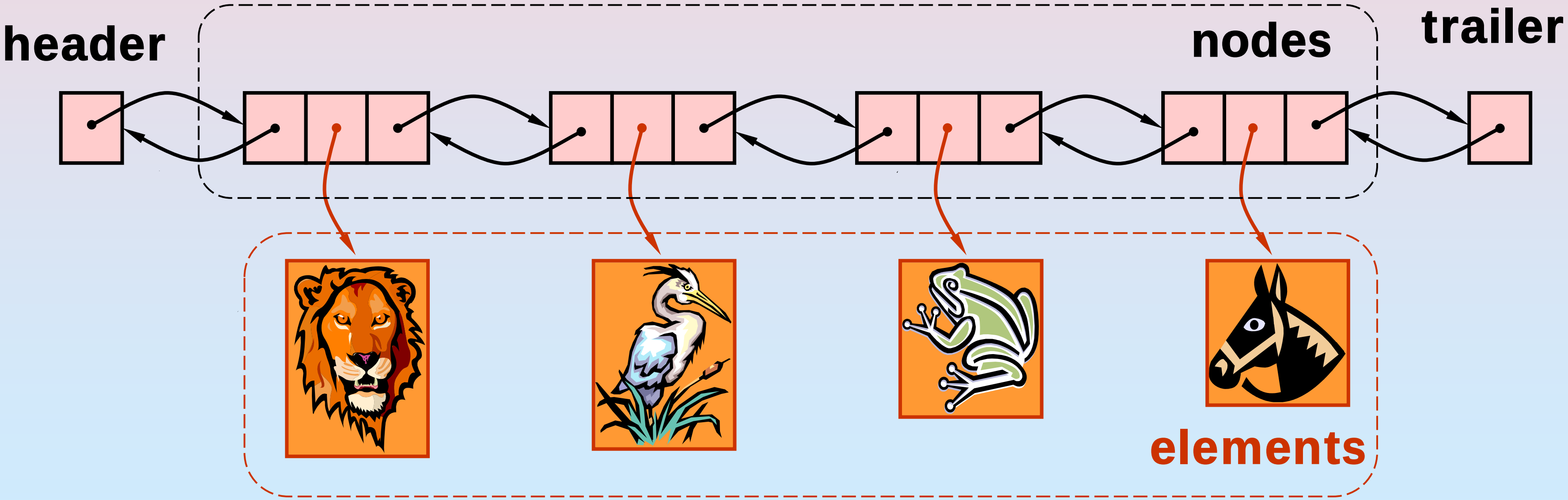
Doubly Linked List

- A doubly linked list is a bi-directional linked list.
- Traversal is possible in both the directions.
- Has two pointers, next and previous.
- next pointer points to successor node, previous pointer points to the predecessor node.



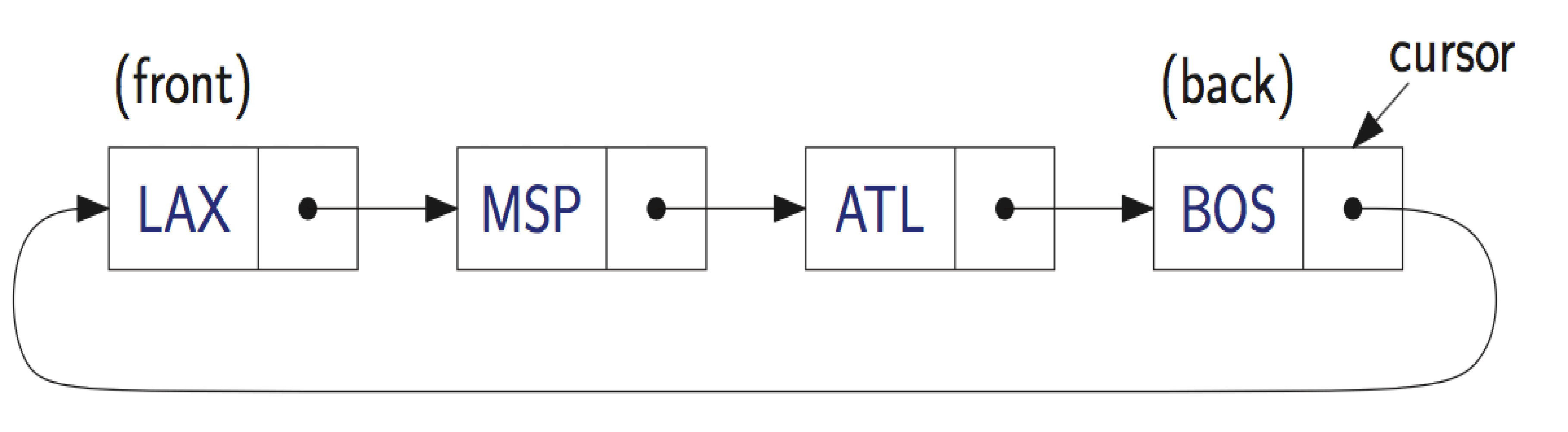
Doubly Linked List





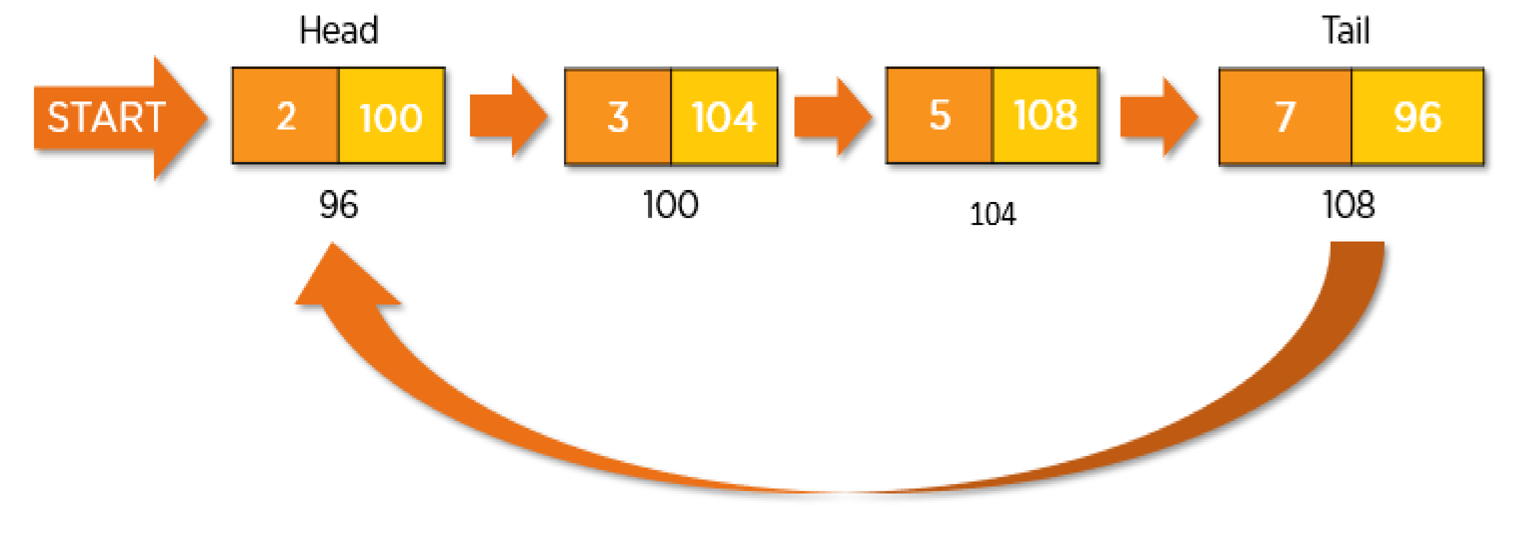
Circular Linked List

- In Circular, singly linked list, Pointer in the last node points back to the first node
- Rather than having a head or a tail, it forms a cycle
- Cursor
  - A virtual starting node
  - This can be varying as we perform operations



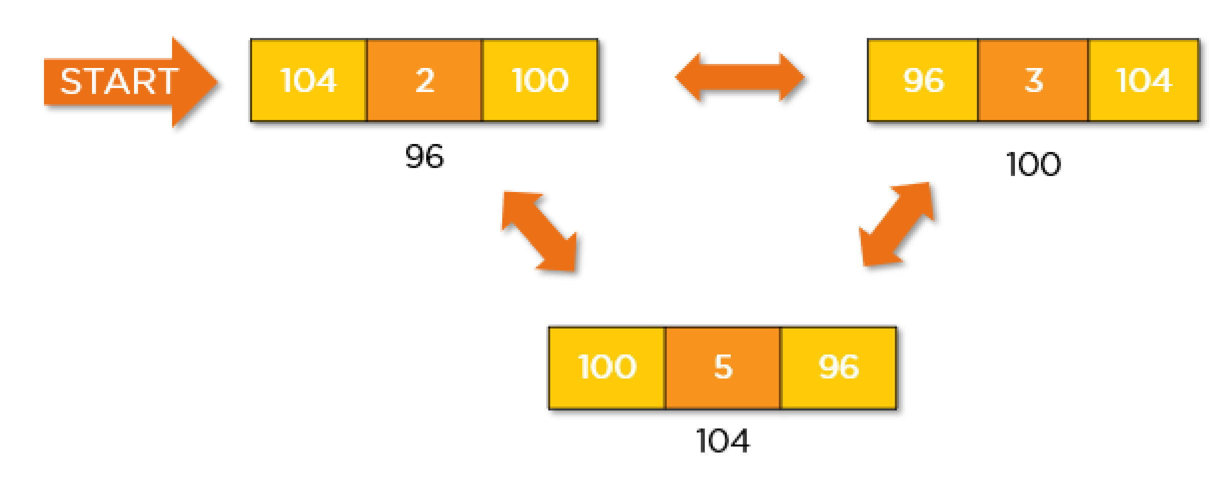
Circular Linked List

- A circular Linked list is a unidirectional linked list.
- Traversal possible only in one direction.
- While traversing, you need to be careful and stop traversing when you revisit the head node.

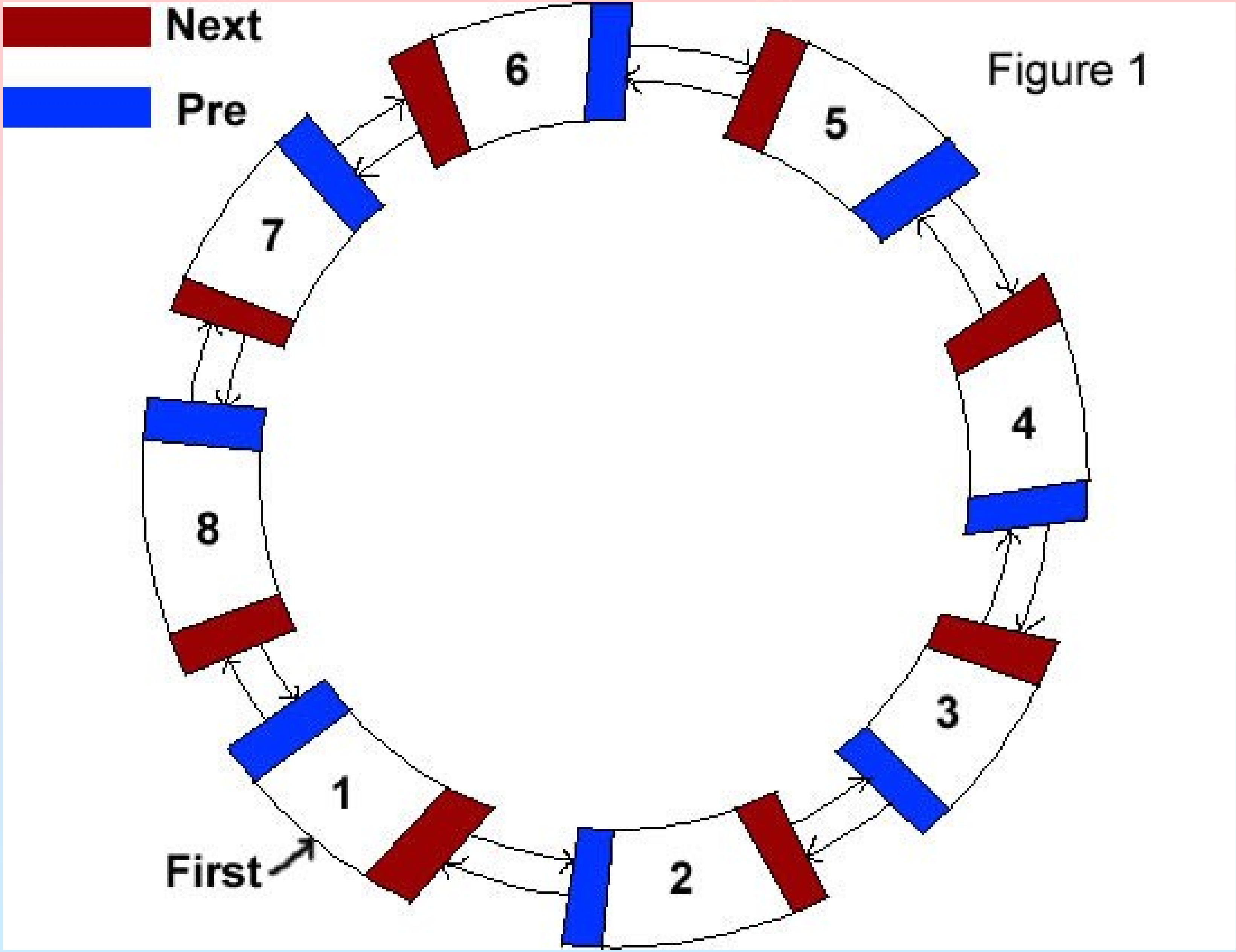


Circular Doubly Linked List

- **It is a mixture of a doubly linked list and a circular linked list**
- **Like the doubly linked list, it has previous pointer**
- **Like the circular linked list, its last node points at the head node.**
- **It is a bi-directional list. So, traversal is possible in both directions.**



Circular Doubly Linked List



Applications of linked list

Applications of linked list in computer science:

1. **Implementation of stacks and queues**
2. **Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.**
3. **Dynamic memory allocation: Uses a linked list of free blocks.**
4. **Maintaining a directory of names.**
5. **Performing arithmetic operations on long integers.**
6. **Manipulation of polynomials by storing constants in the node of the linked list.**
7. **Representing sparse matrices.**

Applications of linked list

Applications of linked list in the real world:

1. **Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.**

2. **Previous and next page in a web browser – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.**

3. **Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.**

Applications of linked list

Applications of Circular Linked Lists:

- **Useful for implementation of a queue. There is no need to maintain two-pointers for the front and rear if we use a circular linked list. Maintain a pointer to the last inserted node and the front can always be obtained as next of last.**
- **Circular Doubly Linked Lists are used for the implementation of advanced data structures like the Fibonacci Heap.**

Applications of linked list

Applications of Circular Linked Lists :

- Circular lists are useful in applications to go around the list repeatedly. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

Applications of linked list

Application of Doubly Linked Lists:

- Redo and Undo functionality.
- Use of the Back and forward button in a browser.
- The most recently used section is represented by the Doubly Linked list.
- Other Data structures like Stack, HashTable, and BinaryTree can also be applied by Doubly Linked List.