CSE-281: Data Structures and Algorithms

Linked Lists
(Chapter-4)

Ref: Schaum's Outline Series, Theory and problems of Data Structures
By Seymour Lipschutz

And Online Resource

Eftekhar Hossain
Assistant Professor
Dept. of ETE, CUET

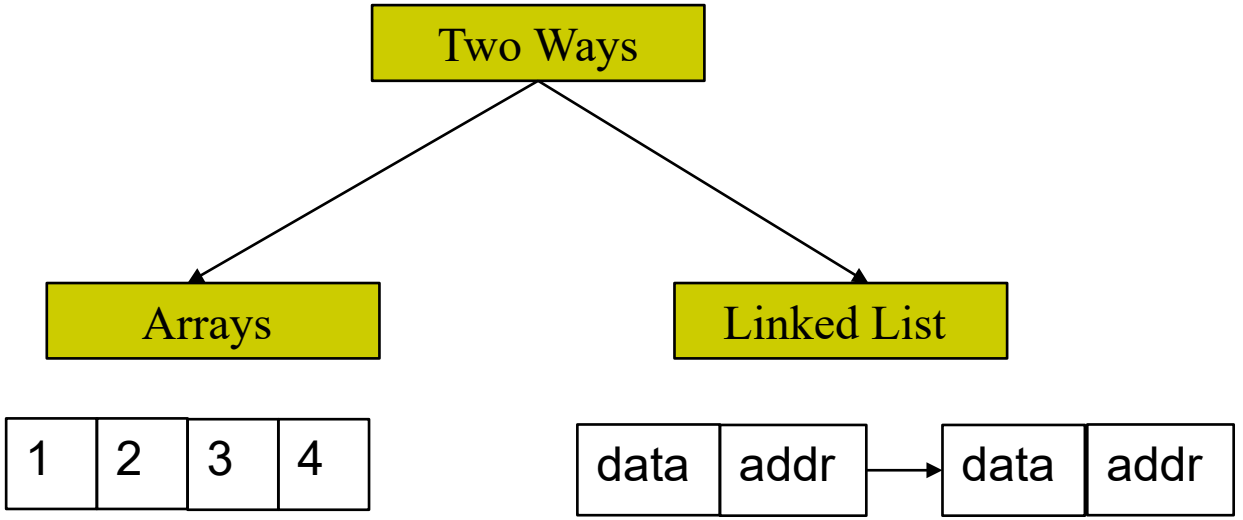
CUET

Topics to be Covered

- Difference between array and linked list
- Types of Linked List
- Singly Linked List
- Operations on Linked list

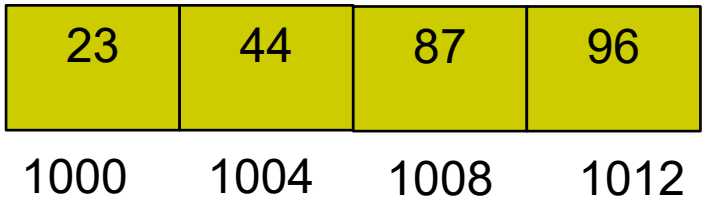
Introduction

- Suppose we want to arrange the name of the students according to the first letter of their names
- Mick, Jhon, Alpha, Mark, Rock , Tony
- We can do this in two ways



Example

- We want to store a list of numbers : 23, 44, 87, 96

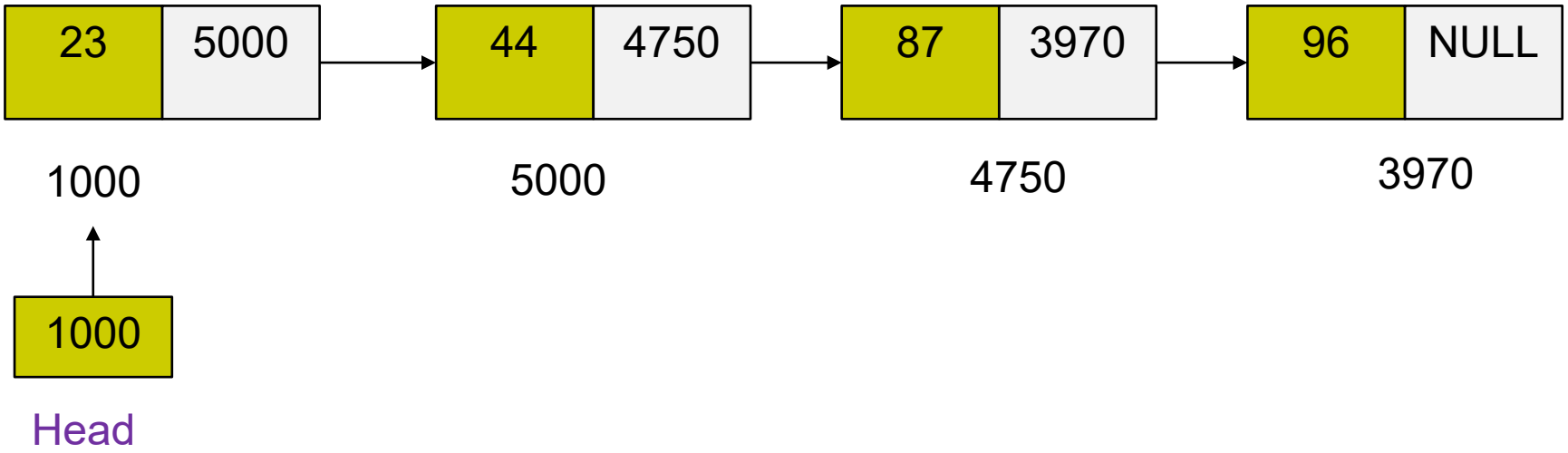


Assuming size of int is 4 Bytes

- In an array , elements are stored in consecutive memory locations
- Array is the sequential representation of the list.

Example

- We want to store a list of numbers : 23, 44, 87, 96



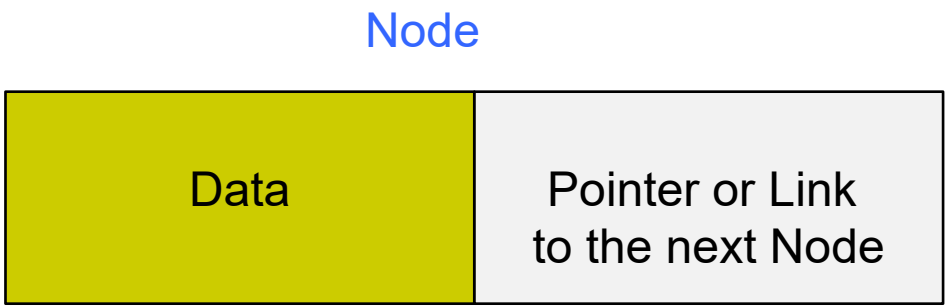
- Nodes are scattered here and there in memory but they are still connected with each other.
- Linked List is the link representation of List

Limitations of Arrays

- Arrays have a fixed dimension.
- Once the size of an array is decided it can not be increased or decreased during execution.
- Array elements are always stored in contiguous memory locations.
- Operations like insertion or deletion of the array are pretty tedious.
- To over come this limitations we use linked list.

Linked List

- Linked list is a linear data structure.
- It is a collection of elements called nodes.
- Each node contains two parts, i.e. DATA part and LINK part.
- The data contains elements and
- Link contains address of another node.



Array Vs Linked List

Array	Linked List
Array is a collection of elements of similar data type.	Linked List is an ordered collection of elements of same type, which are connected to each other using pointers.
Array supports **Random Access**, which means elements can be accessed directly using their index, like arr[0] for 1st element, arr[6] for 7th element etc. Hence, accessing elements in an array is **fast** with a constant time complexity of O(1).	Linked List supports **Sequential Access**, which means to access any element/node in a linked list, we have to sequentially traverse the complete linked list, upto that element. To access **nth** element of a linked list, time complexity is O(n).
In an array, elements are stored in **contiguous memory location** or consecutive manner in the memory.	In a linked list, new elements can be stored anywhere in the memory. Address of the memory location allocated to the new element is stored in the previous node of linked list, hence forming a link between the two nodes/elements.

Array Vs Linked List

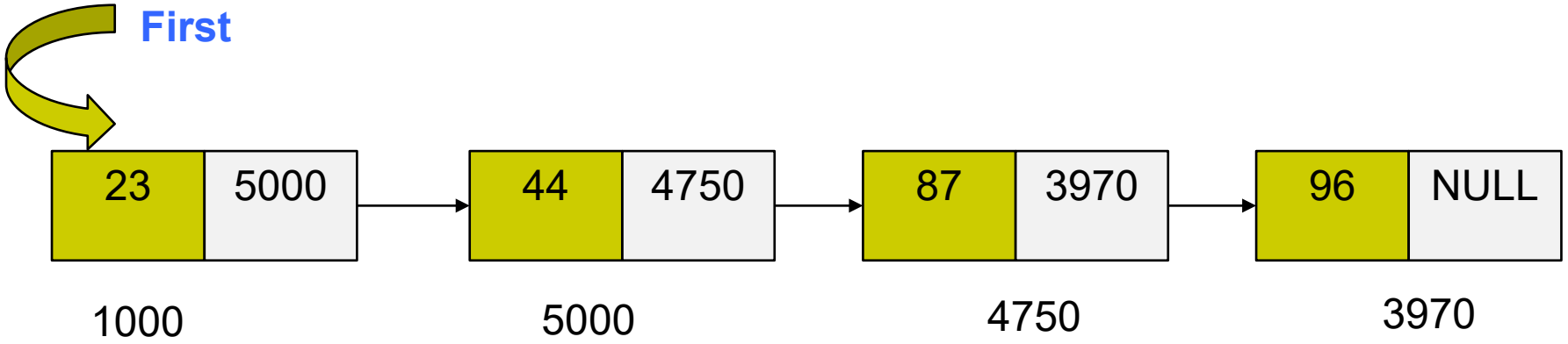
Array	Linked List
In array, **Insertion and Deletion** operation takes more time, as the memory locations are consecutive and fixed.	Insertion and Deletion operations are **fast** in linked list.
Memory is allocated as soon as the array is declared, at **compile time**. It's also known as **Static Memory Allocation**.	Memory is allocated at **runtime**, as and when a new node is added. It's also known as **Dynamic Memory Allocation**.
In array, each element is independent and can be accessed using it's index value.	In case of a linked list, each node/element points to the next, previous, or maybe both nodes.
Size of the array must be specified at time of array declaration.	Size of a Linked list is variable. It grows at runtime, as more nodes are added to it.
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically there is no waste of memory.

Types of Linked List

- Single Linked List
- Double Linked List
- Circular Linked List
- Circular Double Linked List

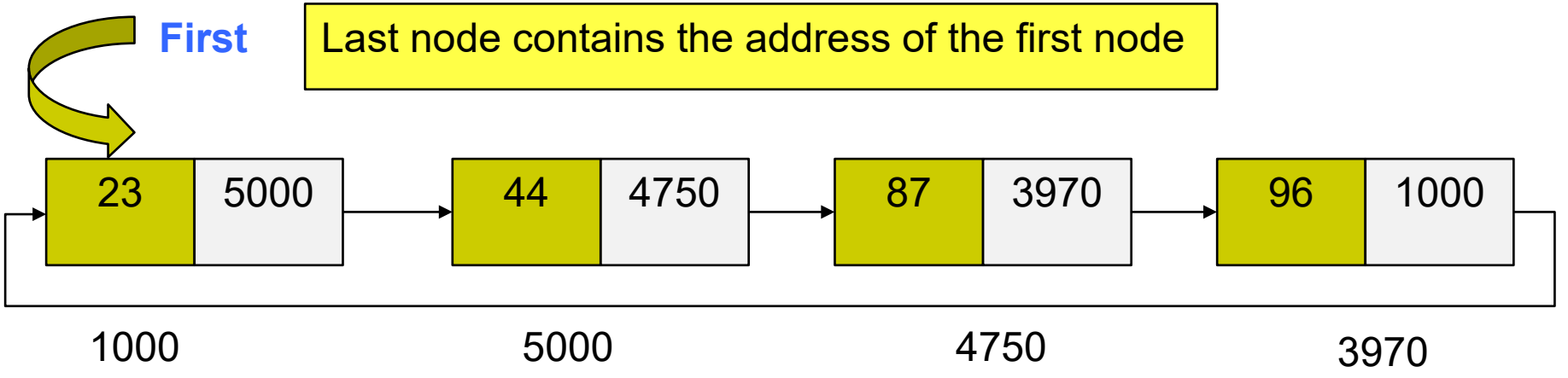
Single Linked List (SLL)

- A single linked list is one in which all the nodes are linked together in some sequential manner .



Circular Singly Linked List

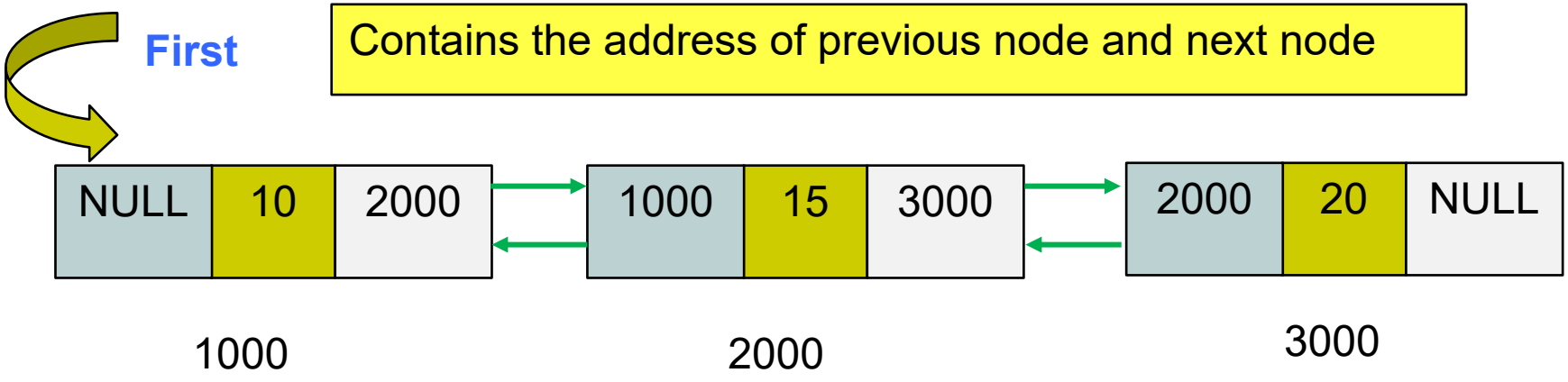
- A circular linked list is one which has no beginning and no ending.
- The null pointer in the last node of a linked list is replaced with the address of its first node such a list is called circular linked list.



Double Linked List

- A single linked list has some disadvantages
  - That it can traverse it in one direction.
  - Many applications require searching
  - backward and forward travelling

- A doubly linked list is divided into three parts When each node is divided into three parts.
- They are two link parts and one data part.



Operations on Linked List

- The basic operations on Linked lists are
  - Creation
  - Insertion
  - Deletion
  - Traversing
- The creation operation is used to create a linked list.
- Insertion operation is used to insert a new node in the linked list at the specified position.
- A new node may be inserted at the beginning of a linked list, at the end of the linked list , at the specified position in a linked list.
- If the list itself is empty , then the new node is inserted as a first node.

Operations on Linked List

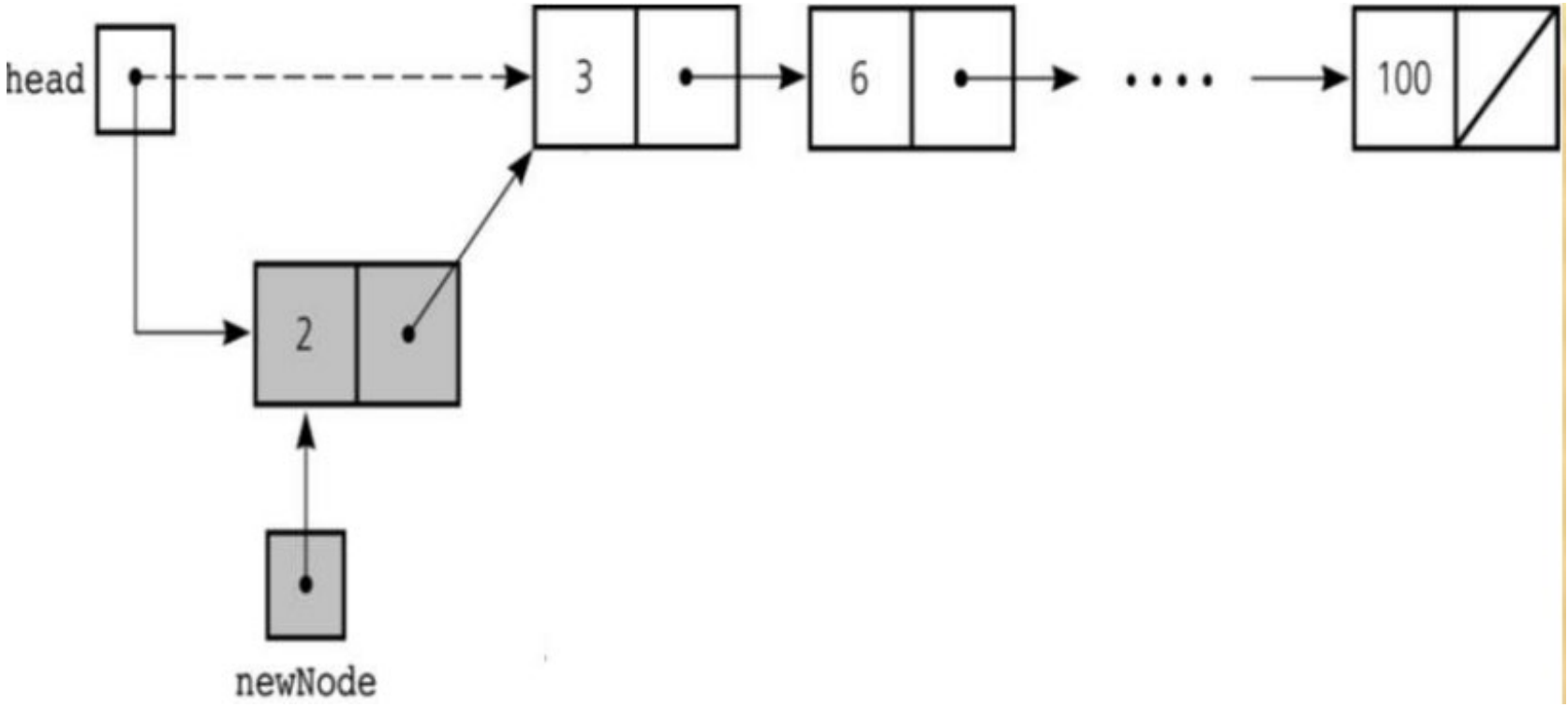
- Deletion operation is used to delete on item from the linked list. It may be deleted from the beginning of a linked list, specified position in the list.
- Traversing operation is a process of going through all the nodes of a linked list from one end to the another end.
- If we start traversing from the very first node towards the last node , It is called forward traversing.
- If the traversal start from the last node towards the first node , it is called backward traversing.

Inserting a node in SLL

- There are 3 cases here:-
  - Insertion at the beginning
  - Insertion at the end
  - Insertion after a particular node

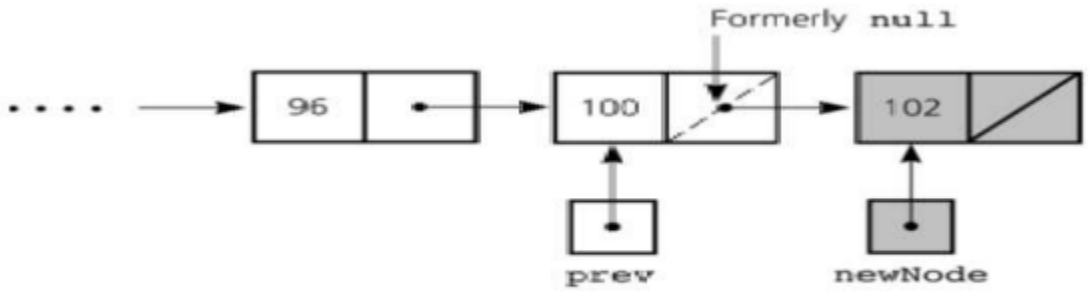
- There are two steps to be followed for inserting at the beginning:-
  - Make the next pointer of the node point towards the first node of the list
  - Make the start pointer point towards this new node
  - If the list is empty simply make the start pointer point towards the new node;

Inserting a node at beginning



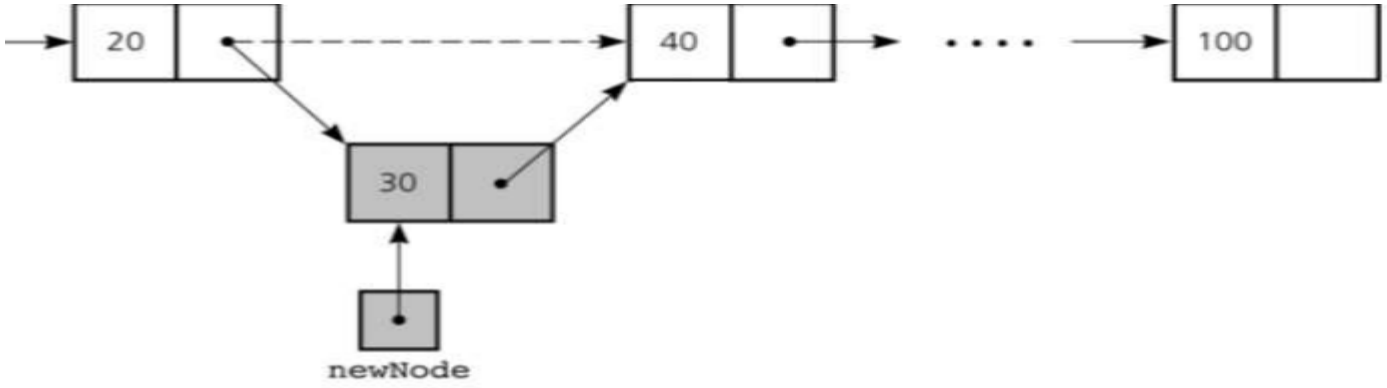
Inserting a node at the end

- Here we simply need to make the next pointer of the last node point to the new node



Inserting after an element

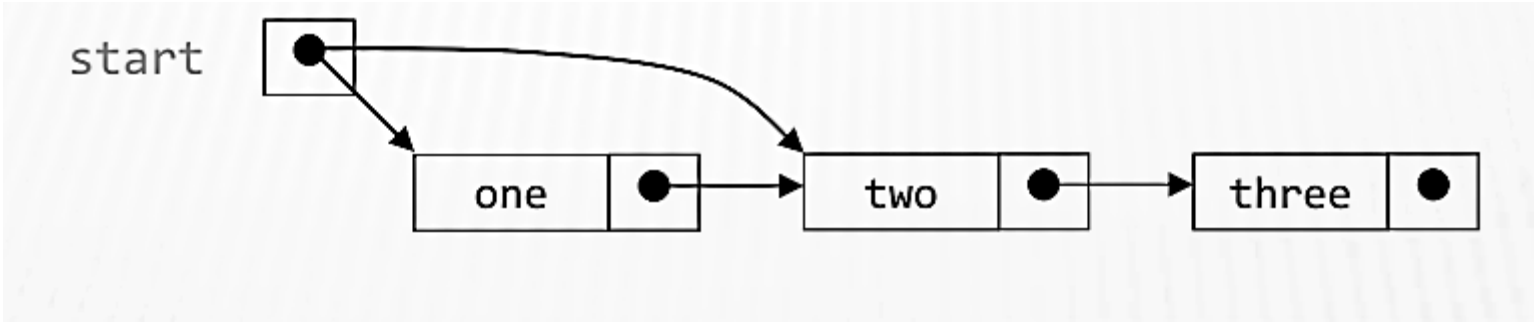
- Here we again need to do 2 steps :-
  - Make the next pointer of the node to be inserted point to the next node of the node after which you want to insert the node
  - Make the next pointer of the node after which the node is to be inserted, point to the node to be inserted



Deleting a node in SLL

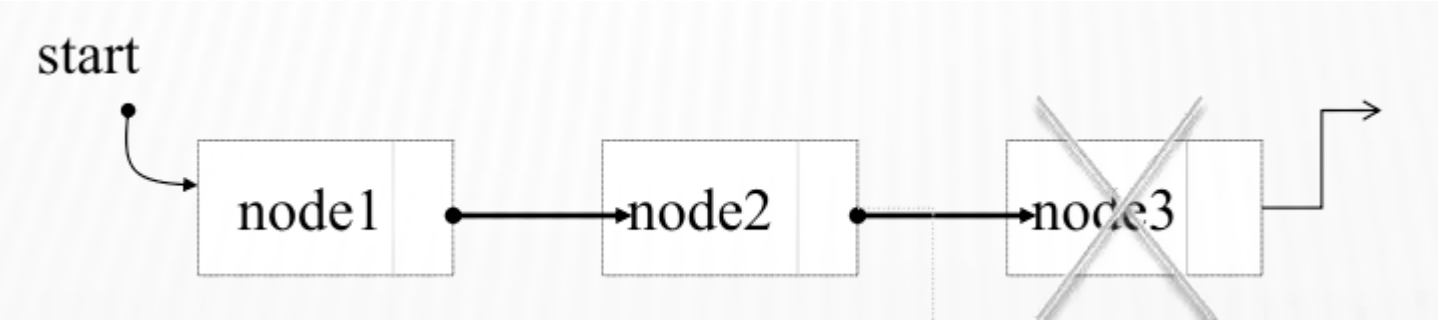
- Here also we have three cases:-
  - Deleting the first node
  - Deleting the last node
  - Deleting the intermediate node

- DELETING THE FIRST NODE
- Here we apply 2 steps:-
- Making the start pointer point towards the 2nd node
- Deleting the first node using delete keyword



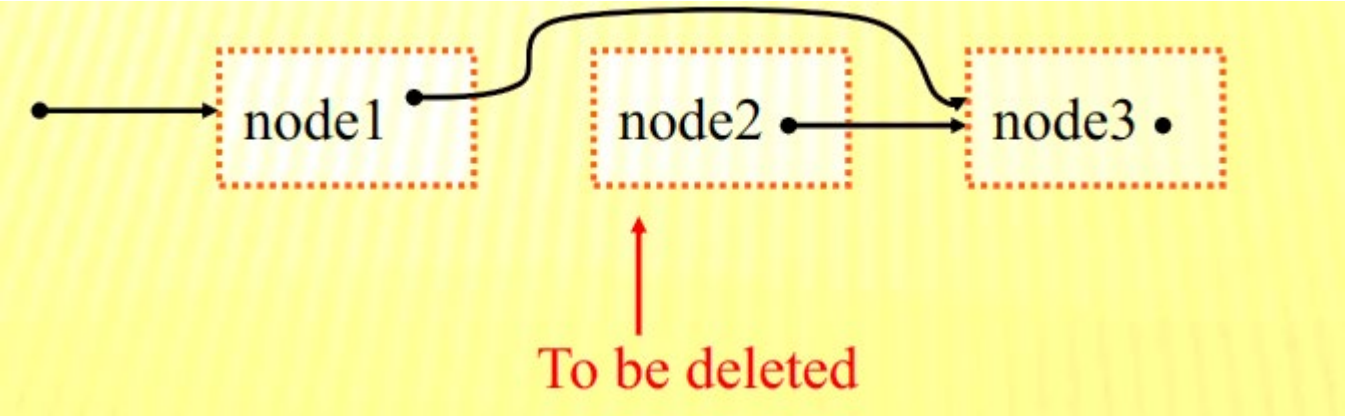
Deleting the Last node

- Here we apply 2 steps:-
  - Making the second last node's next pointer point to NULL
  - Deleting the last node via delete keyword



Deleting a particular node

- Here we make the next pointer of the node previous to the node being deleted ,point to the successor node of the node to be deleted and then delete the node using delete keyword



Advantages of Linked lists

- We can dynamically allocate memory space as needed.
- We can release the unused space in the situation where the allocated space seems to be more.
- Operation related to data elements like insertions or deletion are more simplified.
- Operation like insertion or deletion are less time consuming.
- Linked lists provide flexibility in allowing the items to be arranged efficiently.

Thank You