Data Structure

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

SLLL Time Complexity :

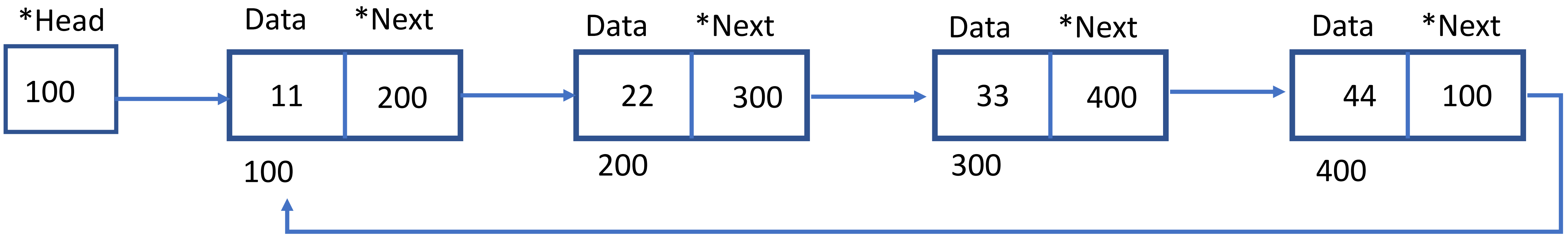
- add node into the linked list at last position (slll): -
  - we can add as many as we want number of nodes into slll in O(n)
  - Best Case : Ω(1)
  - Worst Case : O(n)
  - Average Case : θ(n)
- add node into the linked list at first position (slll): -
  - we can add as many as we want number of nodes into slll in O(1)
  - Best Case : Ω(1)
  - Worst Case : O(1)
  - Average Case : θ(1)
- add node into the linked list at specific position (in between pos) (slll): -
  - we can add as many as we want number of nodes into slll in O(n)
  - Best Case : Ω(1) => if pos == 1
  - Worst Case : O(n)
  - Average Case : θ(n)

SLLL Time Complexity :

- delete node from the linked list at first position –
  - we can delete node which is first pos from slll in O(1) time.
  - Best Case : Ω(1)
  - Worst Case : O(1)
  - Average Case : θ(1)
- delete node from the linked list at last position: -
  - we can delete node which is last pos from slll in O(n) time.
  - Best Case : Ω(1) => if list contains only one node
  - Worst Case : O(n)
  - Average Case : θ(n)
- delete node from the linked list at specific position (in between position) –
  - we can delete node which is first pos from slll in O(n) time.
  - Best Case : Ω(1) => if pos == 1
  - Worst Case : O(n) => if pos == max+1
  - Average Case : θ(n)

Singly Circular Linked List :

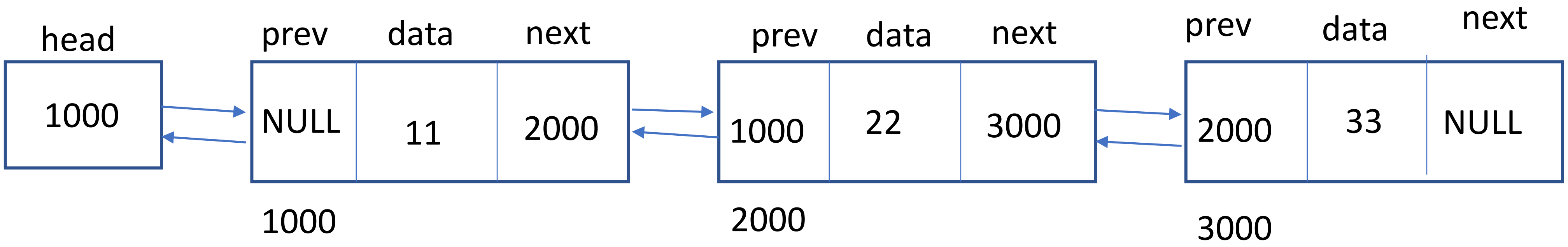
- In this list the last node is linked to the first node. The address of first node is stored in the pointer of the last node.



- Limitations :
  - It is considered as the most inefficient linked list as the add and delete operations on first position also require the traversal till last node to update the pointer.
  - We can traverse only in forward direction.
  - All the operations on this list require O(n) time.

Doubly Linear Linked List :

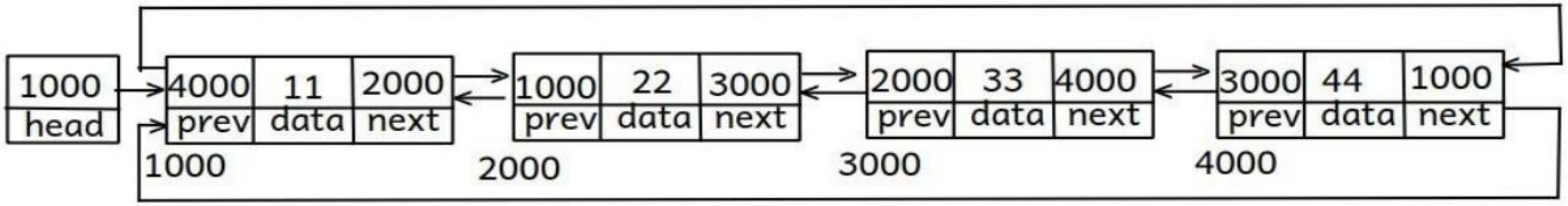
- It is a linked list in which head always contains an address of first element, if list is not empty.
- Each node has three parts:
  - data part: contains data of any primitive/non-primitive type.
  - pointer part(next): contains an address of its next element/node.
  - pointer part(prev): contains an address of its previous element/node.
- next part of last node & prev part of first node point to NULL.



- Limitations :
  - Add last and delete last operations are not efficient as it takes O(n) time.
  - We can starts traversal only from first node, and hence to overcome these limitations Doubly Circular Linked List has been designed.

Doubly Circular Linked List :

- It is a linked list in which head always contains an address of first node, if list is not empty.
- each node has three parts:
  - data part: contains data of any primitive/non-primitive type.
  - pointer part(next): contains an address of its next element/node.
  - pointer part(prev): contains an address of its previous element/node.
- next part of last node contains an address of first node & prev part of first node contains an address of last node.



Linked List :

Advantages of Doubly Circular Linked List:

- DCLL can be traverse in forward as well as in a backward direction.

-Add last, add first, delete last & delete first operations are efficient as it takes O(1) time and are convenient as well.

- Traversal can be start either from first node or from last node.

- Any node can be revisited.

- Previous node of any node can be accessed from it

Array v/s Linked List:

- Array is **static** data structure whereas linked list is dynamic data structure.

-Array elements can be accessed by using **random access** method which is efficient than linked list elements which can be accessed by **sequential access** method.

- Addition & Deletion operations are efficient on linked list than on an array.

- Array elements gets stored into the **stack section**, whereas linked list elements gets stored into **heap section.**

-In a linked list extra space is required to maintain link between elements, whereas in an array to maintain link between elements is the job of **compiler**.

Thank You!