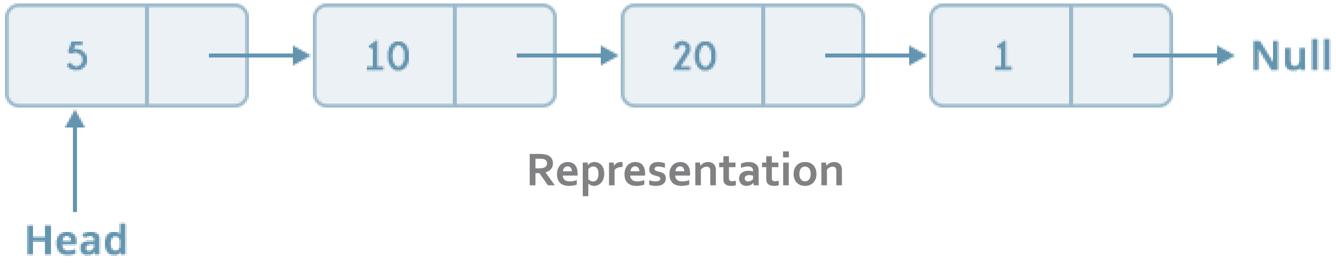
DATA STRUCTURES & ALGORITHMS

Linked Lists

Instructor: Engr. Laraib Siddiqui

Linked list

- It’s a linear data structure, consist of nodes.
- Nodes includes:
  - ✓ Data - elements
  - ✓ Address/Link - address of another node.



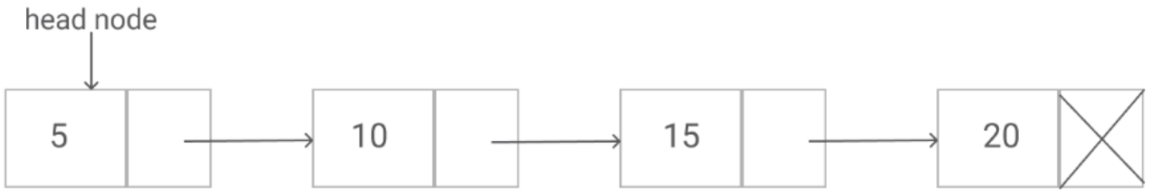
Representation

Linked list vs Arrays

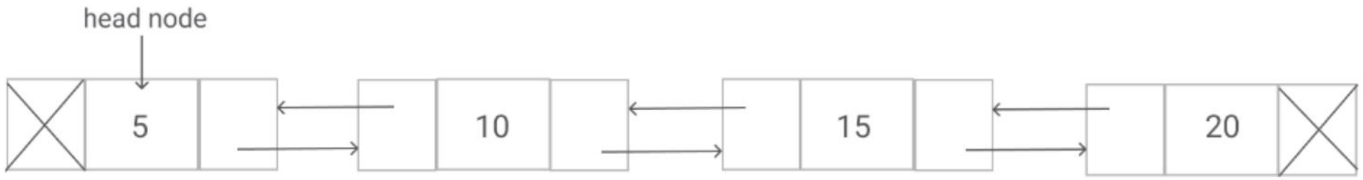
Array	Linked List
Size is fixed; resizing is expensive	Size is dynamic
Data can be access randomly	No random access
Insertion and Deletion operation takes more time.	Insertion and Deletion operations are fast in linked list.
Data is stored in consecutive location.	Data is randomly stored.

Types

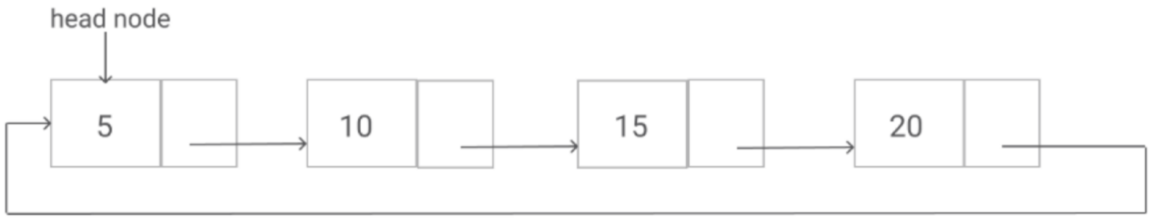
- Singly linked list – Item navigation is forward only.



- Doubly linked list – Items can be navigated forward and backward.



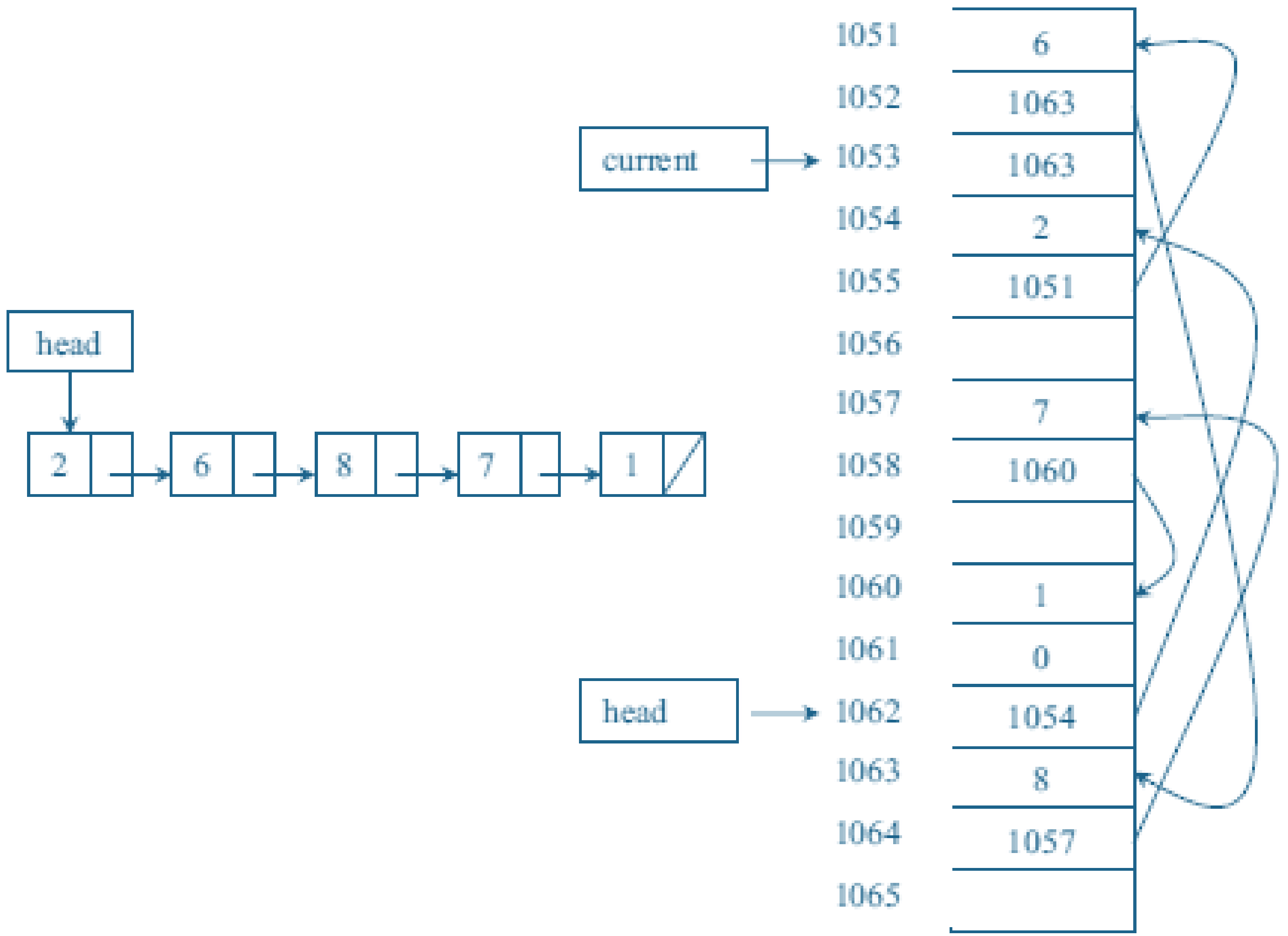
- Circular Linked List – Last item contains link of the first element as next and the first element has a link to the last element as previous.



Comparison

	Singly	Doubly
Strength	Consume less memory Easy to implement	Traversing can be done in both directions
Weakness	Access previous elements not easy	Consume more memory

Memory Layout



Creation of a node

Using Structure

```
struct Node {
        int data;
        struct Node* next;
};
```

Using Class

```
class Node {
public:
  int data;
  Node* next;
};
```

Traversing

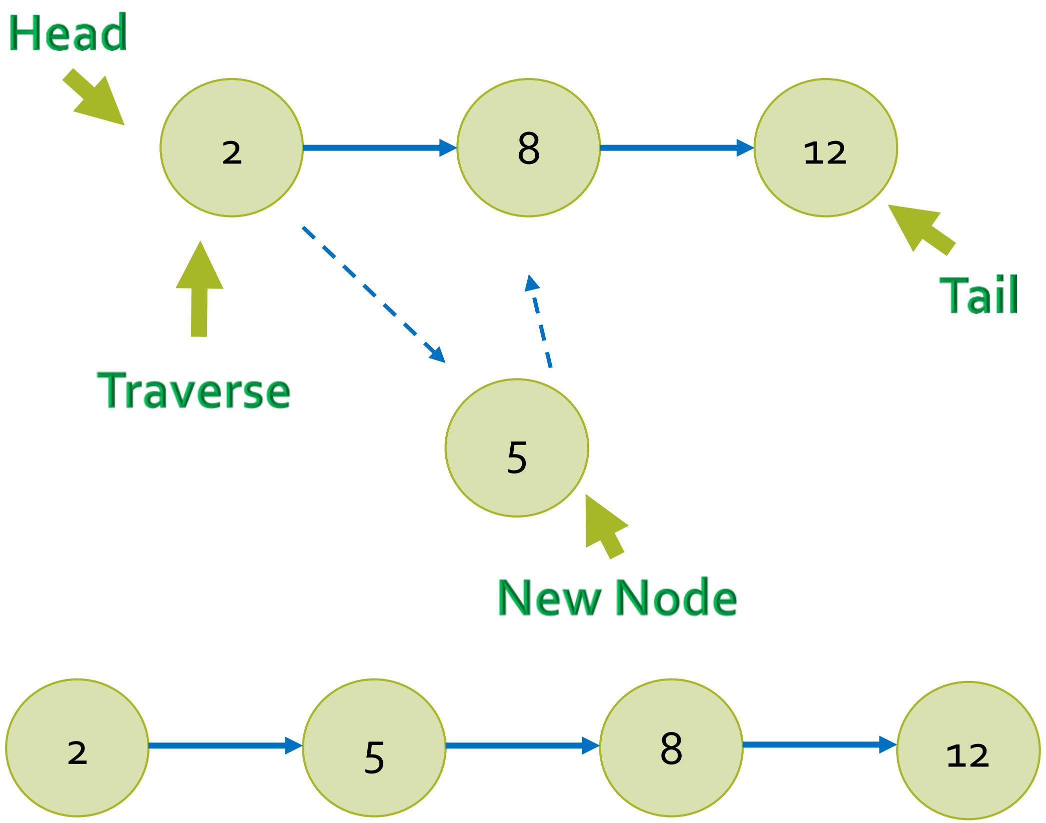
Let LIST be a linked list in memory. Following algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR: = START. [Initializes pointer PTR]
2. Repeat Steps 3 and 4 while PTR ≠ NULL.
3. Apply PROCESS to INFO [PTR]
4. Set PTR:= LINK[PTR]. [PTR now points to the next node.]

   [End of Step 2 loop.]

5. Exit

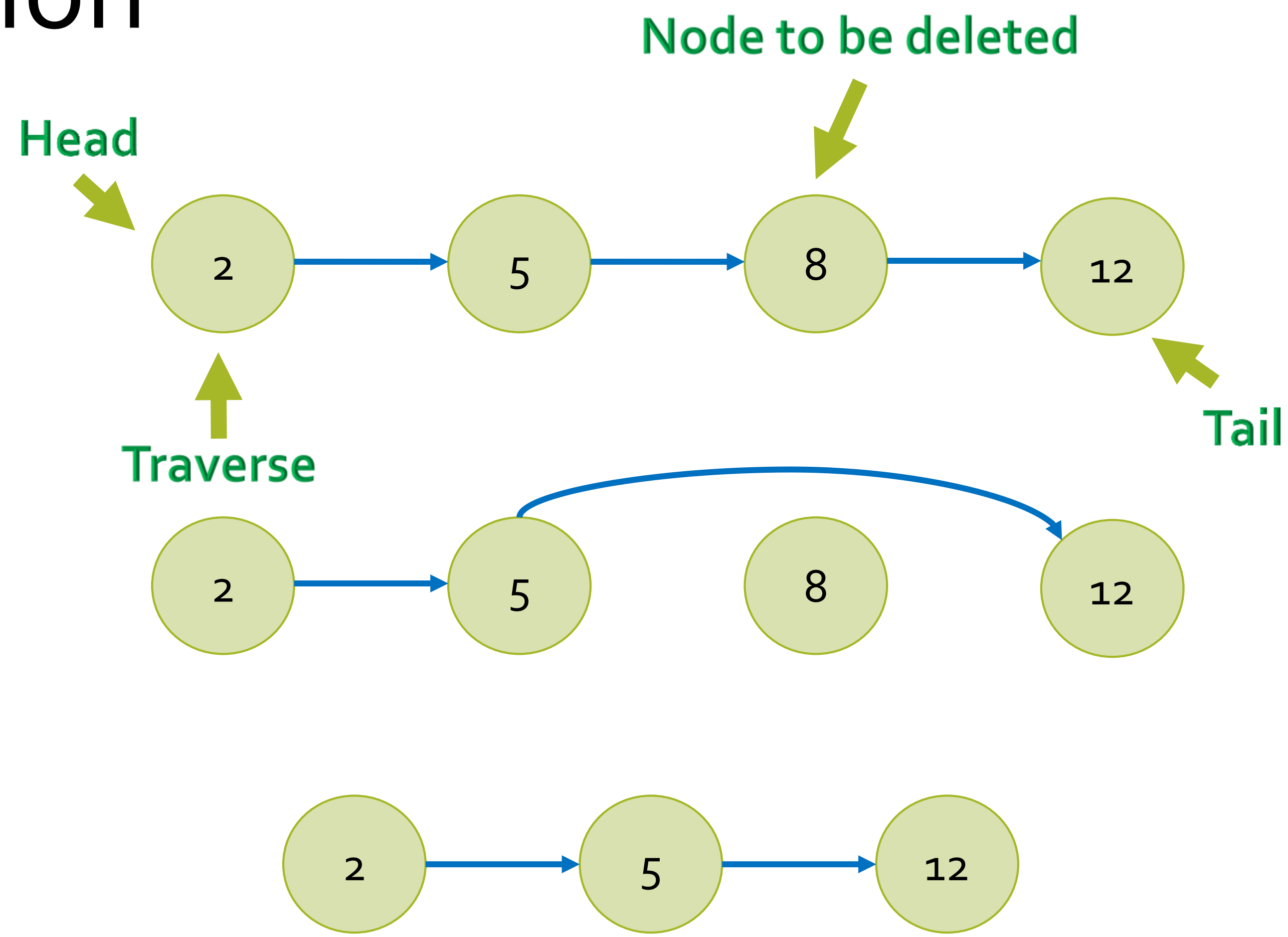
Insertion



Insertion at the beginning

1. Create a new node and assign the address to any node say ptr.
2. OVERFLOW,IF(PTR = NULL)
   write : OVERFLOW and EXIT.
3. ASSIGN INFO[PTR] = ITEM
4. IF(START = NULL)
   ASSIGN NEXT[PTR] = NULL
   ELSE
   ASSIGN NEXT[PTR] = START
5. ASSIGN START = PTR
6. EXIT

Deletion



Deletion

1. Check whether list is Empty (head == NULL)
2. If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
3. If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
4. Check whether list is having only one node (temp → next == NULL)
5. If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
6. If it is FALSE then set head = temp → next, and delete temp.

Complexity

	Singly	Doubly
Access	O(n)	O(n)
Search	O(n)	O(n)
Delete	O(1)	O(1)
Insert	O(1)	O(1)

Applications

- Queue & stack implementation.
- Model real world objects.
- Model repeating event cycles.
- Separate chaining.
- Adjacency list for graphs.