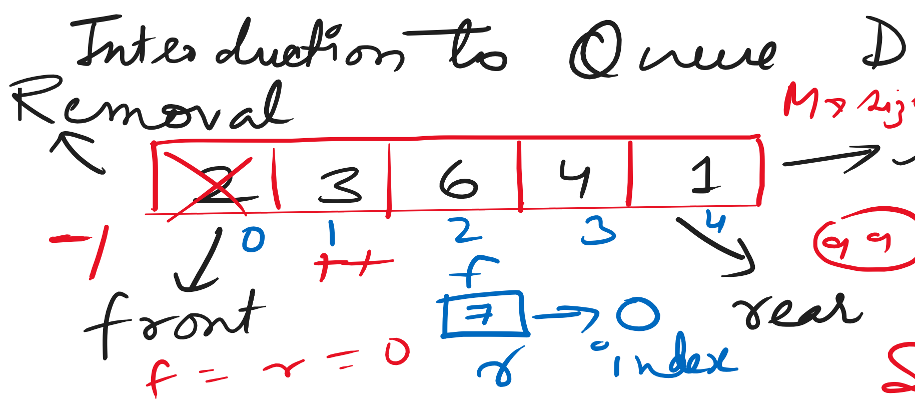
Introduction to Queue Data Structure: →



Empty Queue: front = rear = -1.
Single Element: f = r = 0.

First In First Out Data Structure
Removal of elements → 2, 3, 6, 4, 1
It can also be called Last In Last Out.

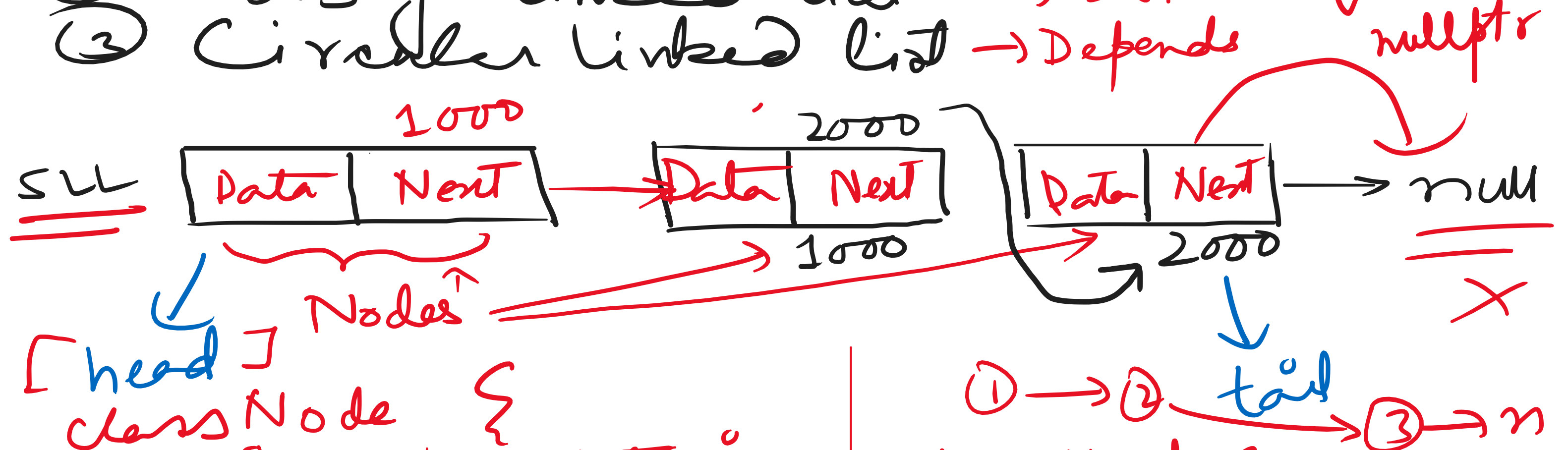
* Addition is always at the rear & removal/deletion is at the front.

* BFS Traversal Algorithms → Queue
Level Order Traversal

Introduction to Linked Lists:
These data structures have entity called nodes connected to each other via addresses/pointers/references. Each node has data associated with the node and the address of the (next/previous) nodes depending on the type of linked list.

Real life examples: ① Web Browsers → Google Chrome
Back | Current | Forward
② Music Playlist ← spotify
<< < (▷) > >>

Based on Traversal:
① Singly Linked List → Forward
② Doubly Linked List → Both ways
③ Circular Linked List → Depends



```
class Node {
  i, c, s, b   data;
  Node* next;
}
```

① → ② → ③ → n

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

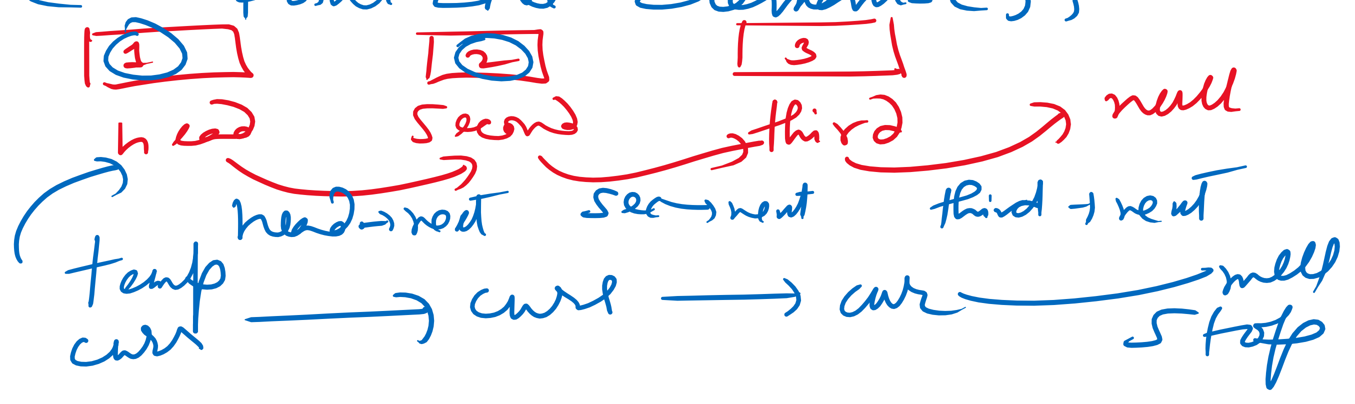
Difference b/w Array & Linked Lists

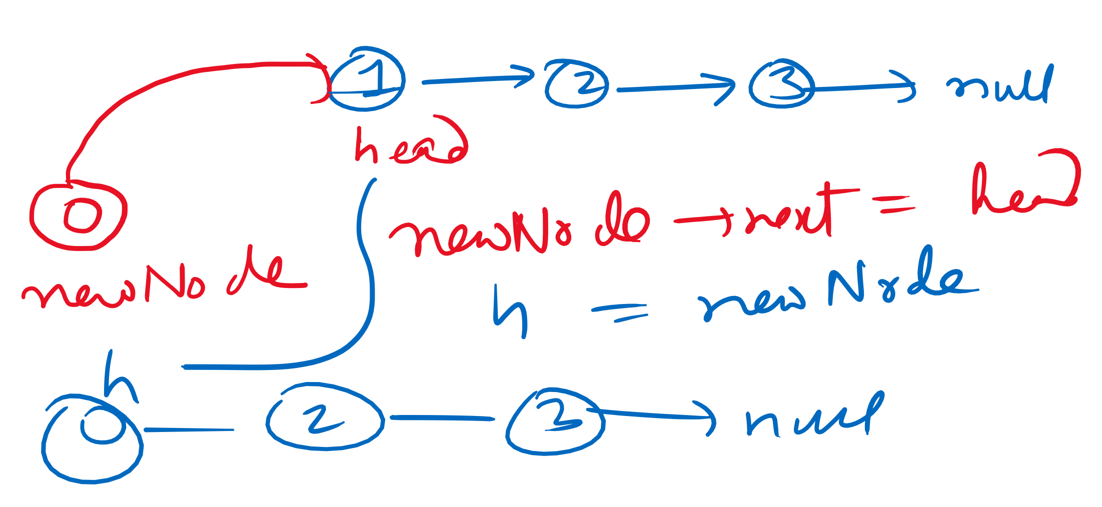
Array (Access)	Linked List
1 2 4 5 6 7 (index 0 1 2 3 4 5)	① → ② → ③ → null (h, t)
① Search Operation: 5th Element: arr[4] → O(1) index based search	① Search Operation: 3rd node: O(n) → we traverse all the elements
⑪ Insertion: insert 3 at index 2; 1 2 3 (0 1 2) Shift O(n)	⑪ Insertion → 3 steps; ① ~~→~~ ③ → null, ① → ② → ③, O(1)

* Important Questions & functions:

① Insert
↳ Insert at head
↳ Insert at tail
↳ Insert after specific
(Create a new Node)
↔ print List Elements();

② Delete
↳ Delete head
↳ Delete tail
↳ Delete target





newNode → next = head
h = newNode