Introduction to Singly Linked Lists

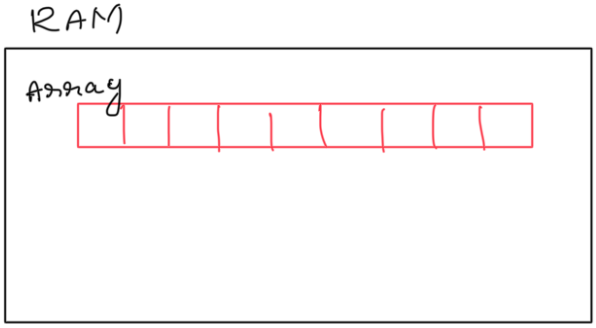
Day 35
14/08/2025

As we know the Arrays are fixed in size, if any new data to be added to the array then the New array with more size is created and Elements are copied to it.

To address this issue (using Memory inefficiently) The LinkedList is designed.

What are LinkedList ?

In case of Array Continous Memory Block in the Memory is allocated



To address limitations of array
1. fixed size (Not dynamic)
2. Wastage of Memory Space
3. avoid copying of the elements

Linked List

We allocate Memory for 1 Element at a time
In Case of Java & Python objects are created In Heap Memory

When we Create objects of NODE'S then the Memory get allocated In the Heap

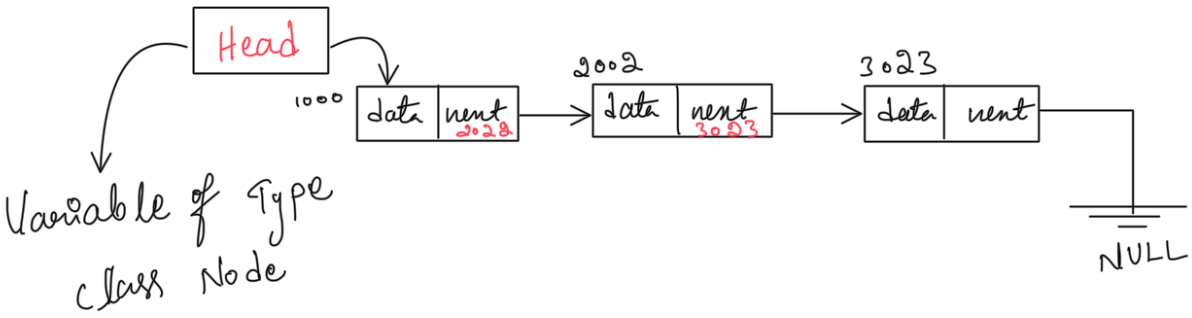
Node : 1 Element for which Memory allocated dynamically In the RAM → object

Node contains → Data → contains different types of data
→ Reference, Address of Next Node

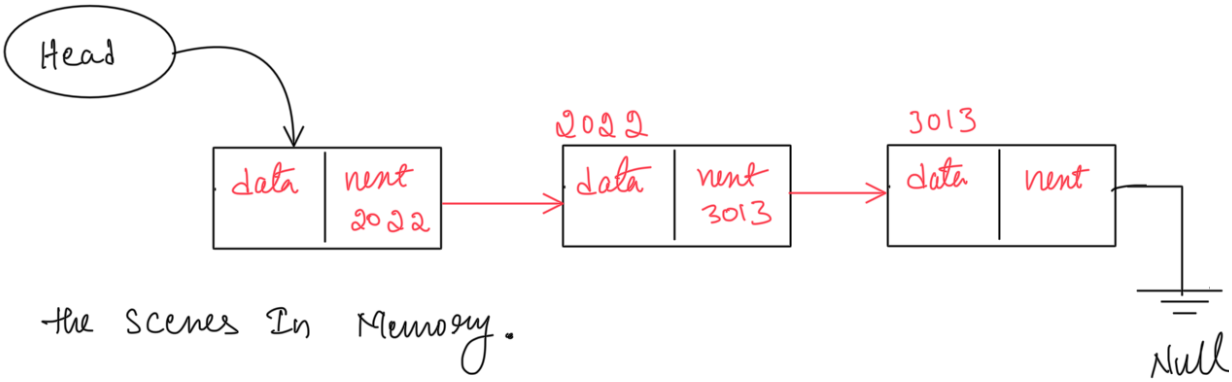
→ In case of Doubly linked list it's also have address of previous Node

NOTE:- Data Depends on Requirements.

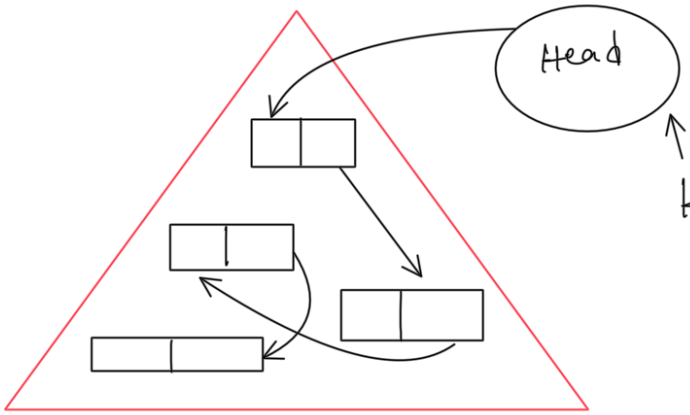
How Linked List Looks Like Visually.



Variable of Type class Node



Behind the Scenes In Memory.



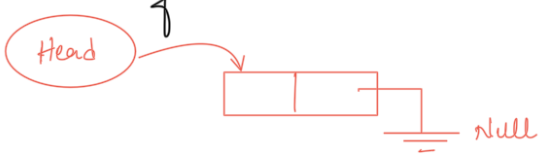
Variable of type "Node" class

Head pointes

Initially Head is pointing to Null

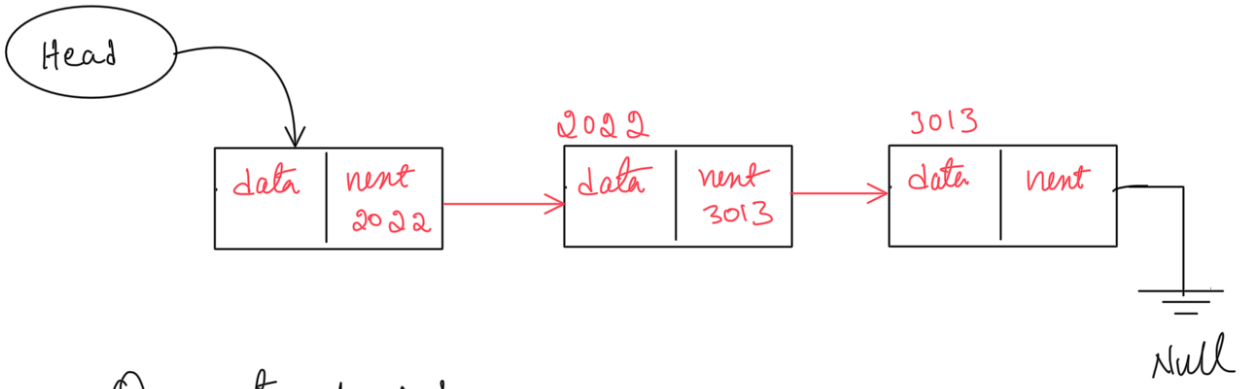
After creating a Node the the Head will point to the first Node which is Created.

if Head is Null

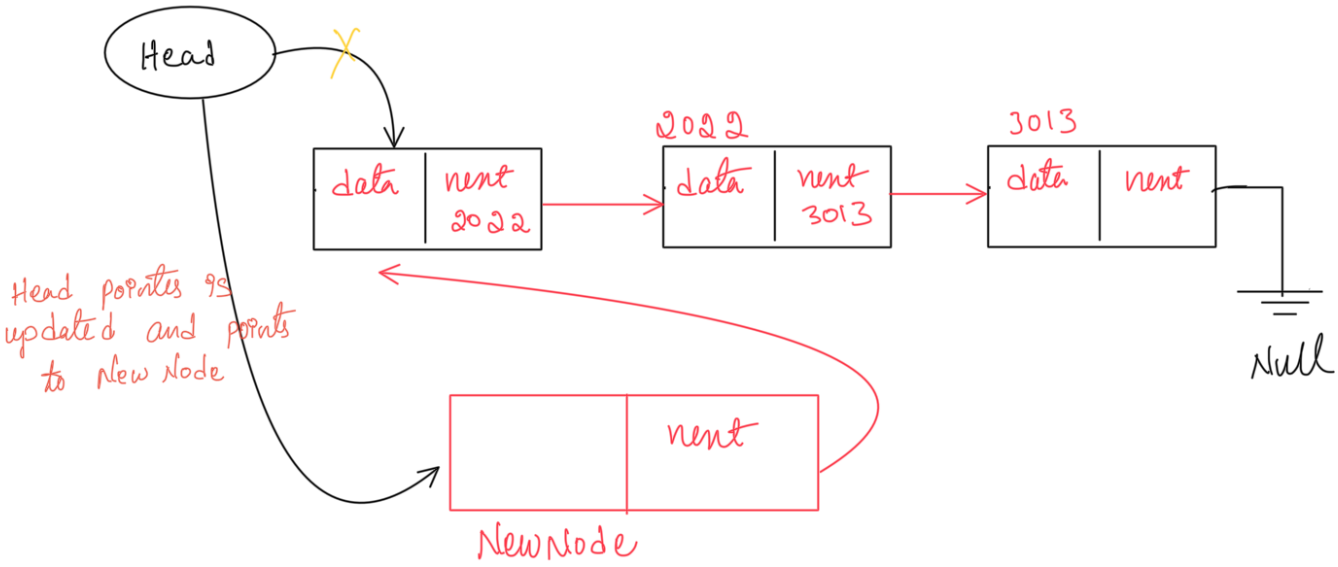


create a New Node and point the Head to New Node

Inserting At Front

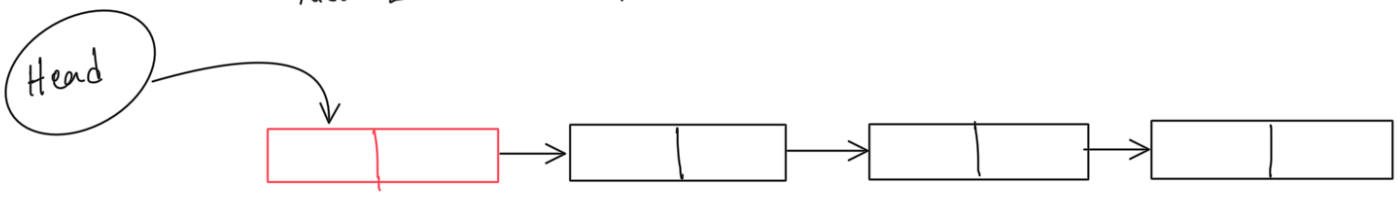


1. Create New Node
2. Point New Node.next to Head
3. Update Head to New Node



The Node is successfully added to the linked list

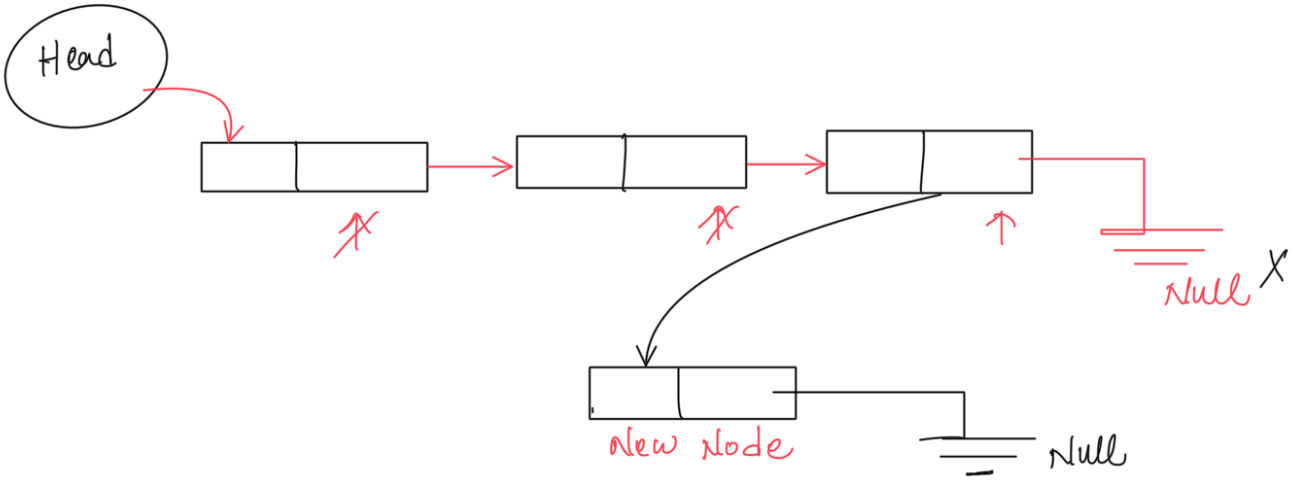
New Linked List Looks Like

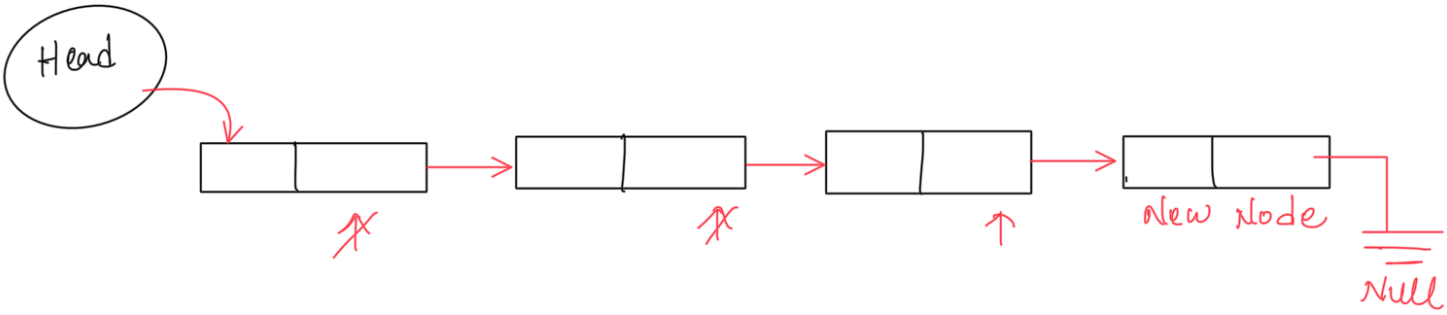


Inserting At End

Traverse from Head Node till we get a last Node.
Last Node → Have next address Null, Last Node doesn't Have Any Neighbour.

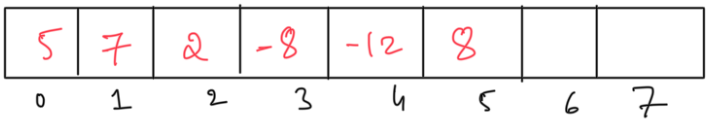
Assign New Node Address to the Last Node's Next.





Inserting In the Middle

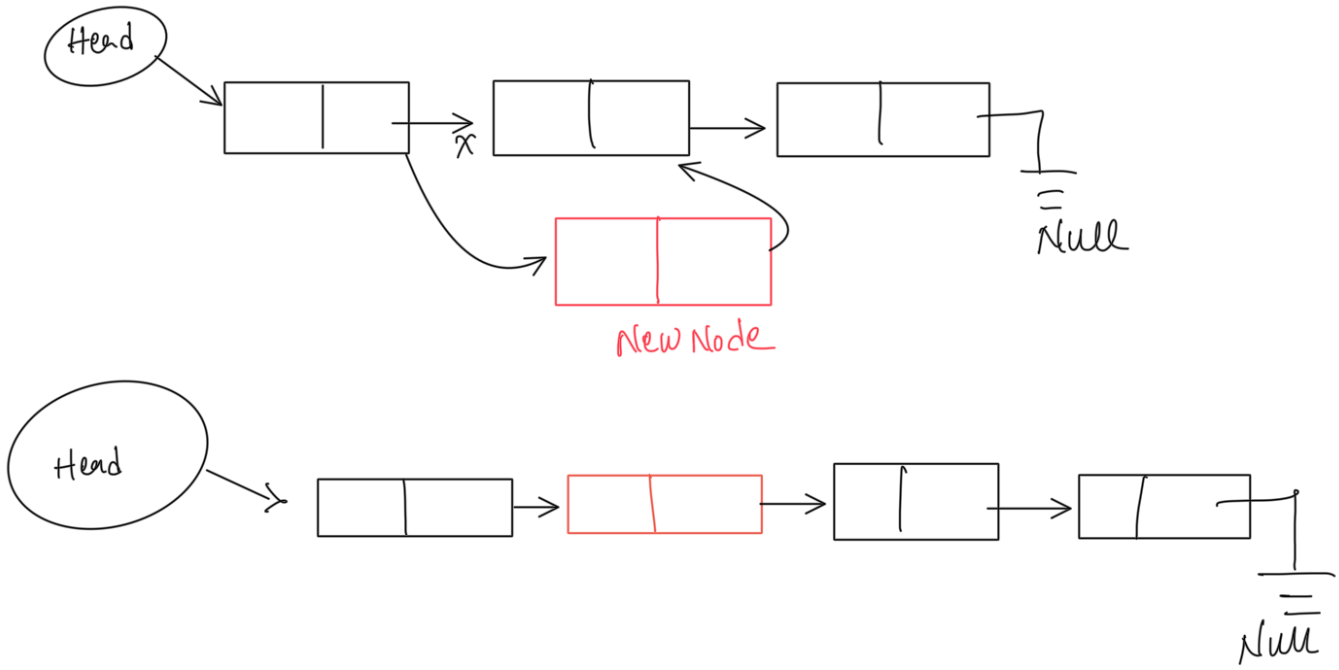
In Case of Arrays



If we want to insert element at index 3
We Have to shift the Index

Space is Created and Shifted the others Elements to insert at position.

Inserting Elements at the Middle of Linked List is more time efficient Its just changing references accordingly.



Feature	Array	Linked List
Access time	**O(1)**	**O(n)**
Insertion/Deletion	**O(n)**	**O(1)** (if pointer known)
Memory usage	Low	Higher (extra pointers)
Size flexibility	Fixed	Dynamic
Cache friendliness	High	Low

Array

✅ **Advantages**

1. **Direct Access (Random Access)**
   - Access any element in **O(1)** time using index.
   - Example: `arr[5]` is instant.
2. **Memory Locality**
   - Elements are stored **contiguously**, improving CPU cache performance.
3. **Ease of Use**
   - Simple to implement and use for fixed-size collections.
4. **Low Memory Overhead**
   - Only stores the actual data (no extra pointers like in linked lists).

Linked list

✅ **Advantages**

1. **Dynamic Size**
   - Can grow/shrink at runtime without resizing.
2. **Efficient Insertion/Deletion**
   - Inserting/removing at the **beginning** or **middle** is **O(1)** (if you already have the pointer).
3. **No Wasted Space for Fixed Size**
   - Memory is allocated as needed.

❌ **Disadvantages**

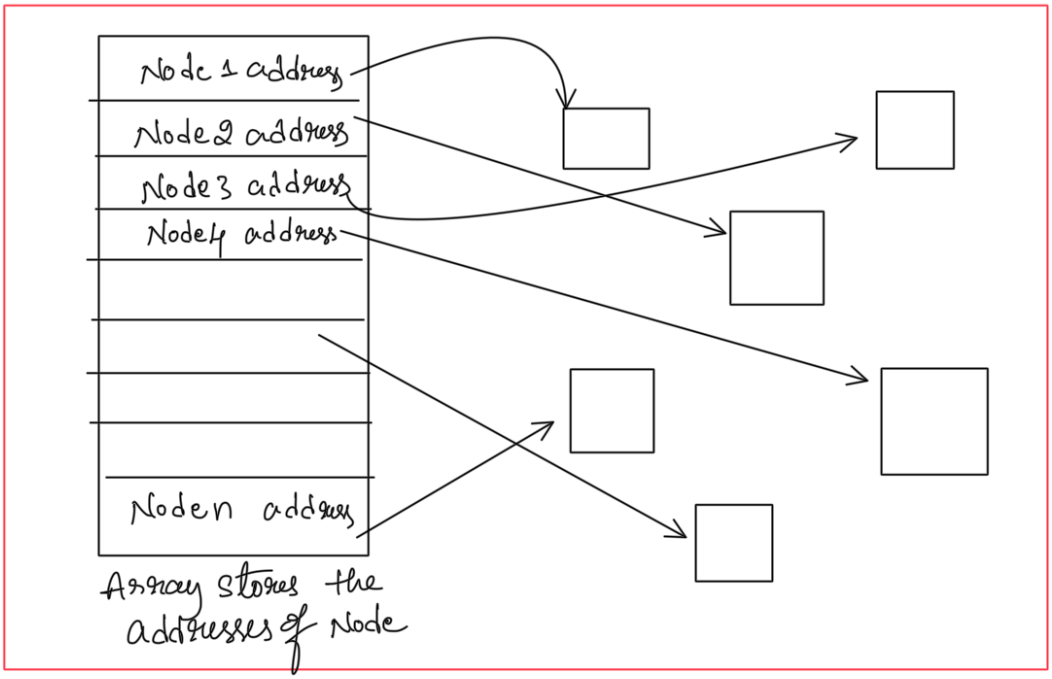
1. **No Random Access**
   - Must traverse nodes one by one (**O(n)**) to access element.

❌ **Disadvantages**

1. **Fixed Size**
   - Size must be known in advance; resizing is costly (copying elements).
2. **Insertion/Deletion Costly**
   - Adding/removing in the middle requires shifting elements (**O(n)** time).
3. **Wasted Space**
   - If array is oversized, unused slots waste memory.

2. **Extra Memory Overhead**
   - Each node stores extra pointer(s) (next, sometimes prev).
3. **Poor Cache Performance**
   - Elements may be scattered in memory, reducing CPU cache efficiency.
4. **More Complex Implementation**
   - Harder to manage (need careful handling of pointers to avoid memory leaks).

ArrayList



To overcome the disadvantages of LinkedList ArrayList Come into picture.

ArrayList ⟶ Combination of Array and Linked List
