Linked Lists

List Abstract Data Type (ADT)

- It contains same type of elements (List of names, List of numbers, …)
- First define the data and operations
- No implementation details

List ADT

- It contains same type of elements (List of names, List of numbers, …)
- First define the data and operations
- No implementation details

What are List operations?

List ADT

- It contains same type of elements (List of names, List of numbers, …)
- First define the data and operations
- No implementation details

What are List operations?

- get() : Return an element at a given position
- insertLast() : Insert an element at the end of the List
- insertAt() : Insert an element at a given position
- insertBeg() : Insert an element at the beginning of the list.
- remove() : Remove an element at a given position
- replace() : Replace an element at a position by another element
- size() : Return the size of List (number of elements)

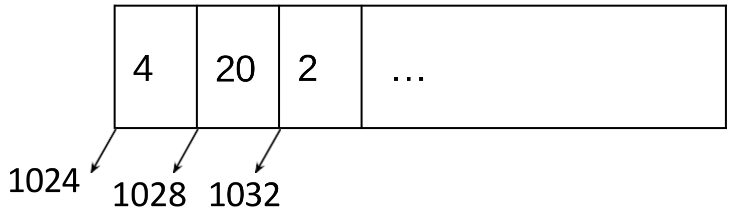
.

.

.

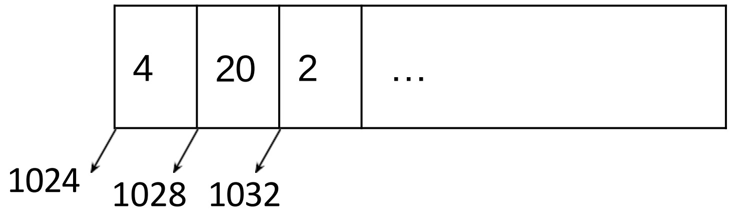
List Array Implementation

int arr[30];



List Array Implementation

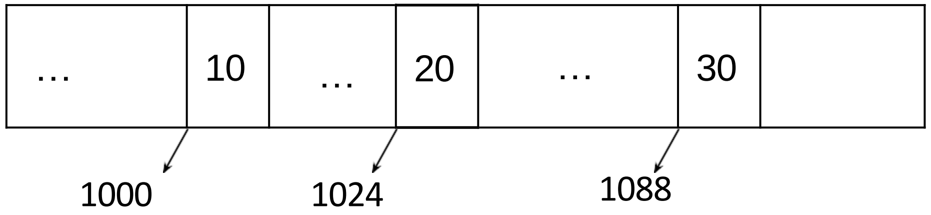
int arr[30];



Fixed size! What if we want to add more elements?

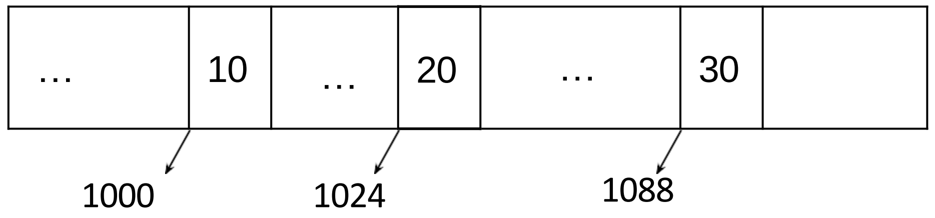
Linked List Implementation

- insert(10)
- insert(20)
- insert(30)



Linked List Implementation

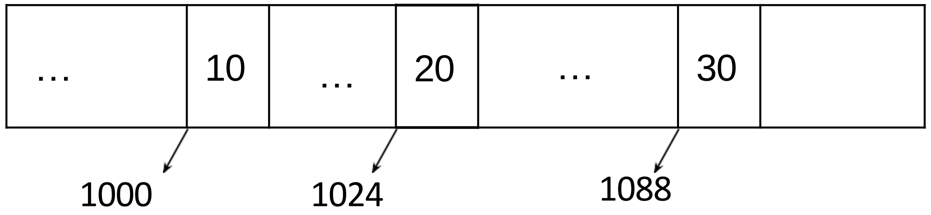
- insert(10)
- insert(20)
- insert(30)



Add an element at a time!
It can be added to any cell of memory!

Linked List Implementation

- insert(10)
- insert(20)
- insert(30)

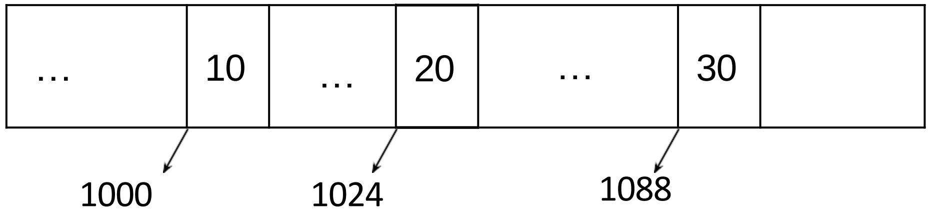


Add an element at a time!
It can be added to any cell of memory!

How to link them?

Linked List Implementation

- insert(10)
- insert(20)
- insert(30)

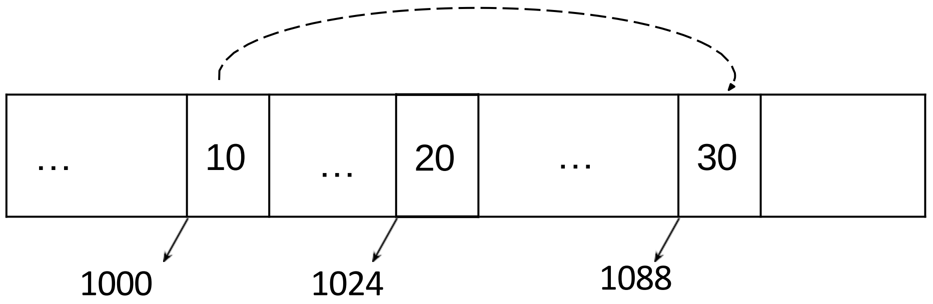


Add an element at a time!
It can be added to any cell of memory!

How to link them? Keep the address of next element!

Linked List Implementation

- insert(10)
- insert(20)
- insert(30)

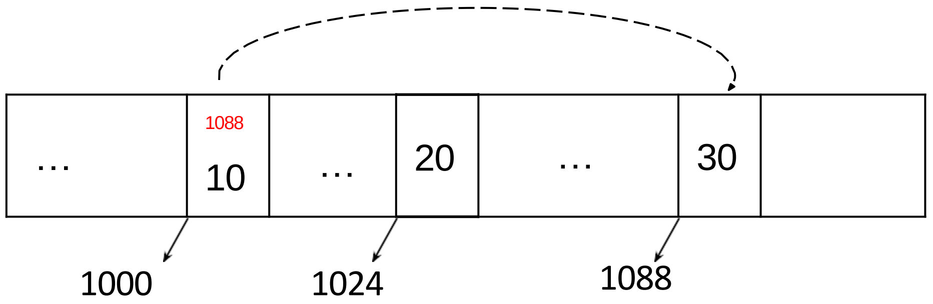


Add an element at a time!
It can be added to any cell of memory!

How to link them? Keep the address of next element!

Linked List Implementation

- insert(10)
- insert(20)
- insert(30)

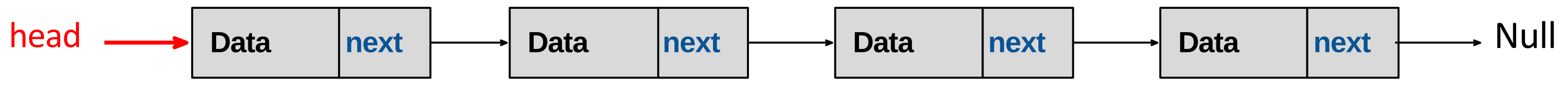


Add an element at a time!
It can be added to any cell of memory!

How to link them? Keep the address of next element!

Linked List Definition

- A data structure to represent a sequence of items with the same type
- Each item in the list points at the item immediately after it
- Usually, we keep an extra pointer, called head, to point at the first item
- Usually, an item in the list is called a node
- Each node includes a data and a next pointer



Linked List vs Arrays

Arrays:

- They are fixed in size.
- They are contiguous in memory.

Pros:

- Arrays give us O(1) access to the nth element.

Cons:

- Insertion and deletion from an arbitrary index in an array (assuming we want to shift everything over to fill in the gap or make room for the new element) are potentially expensive: O(n)
- Arrays don't grow to accommodate extra elements.
- Arrays might be larger than strictly necessary.

Linked List vs Arrays

Linked Lists:

- A data structure consisting of nodes that are linked together with pointers
- The length of the list is dynamic, and nodes can be located all over the place in memory. (They don't have to be contiguous.)

Pros:

- Insertion and deletion at arbitrary positions in the list doesn't require us to shift elements all over the place. Thus, the operations are pretty cheap (that is, after we find the element we're looking for!).
- Length the of the list can change dynamically.
- We don't have to find a contiguous block of memory for the list. Nodes can be
- scattered throughout memory.

Cons:

- We no longer have direct access to each element.
- Takes up more memory than an array; we have to store lots of addresses!

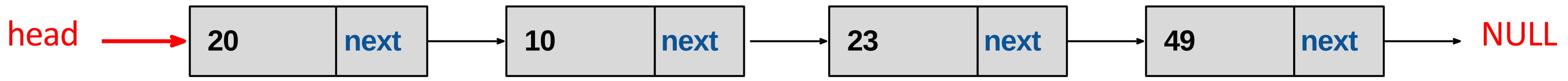
Linked List implementation

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

- A linked list including the above node structure is called **singly** linked list

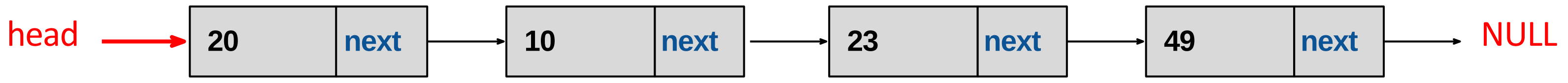
Singly Linked List Implementation

- `get(1);`



Singly Linked List Implementation

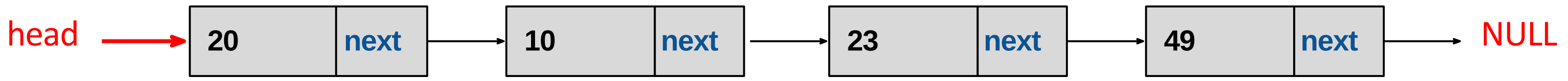
- `get(1);`



Time complexity?

Singly Linked List Implementation

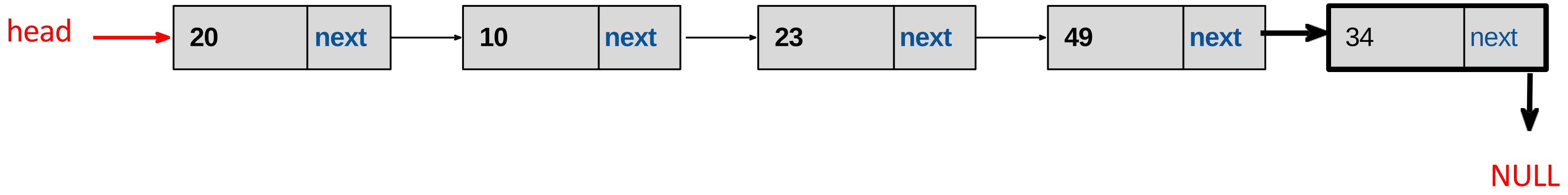
- `get(1);`



Time complexity? O(n) where n is the size of the linked list

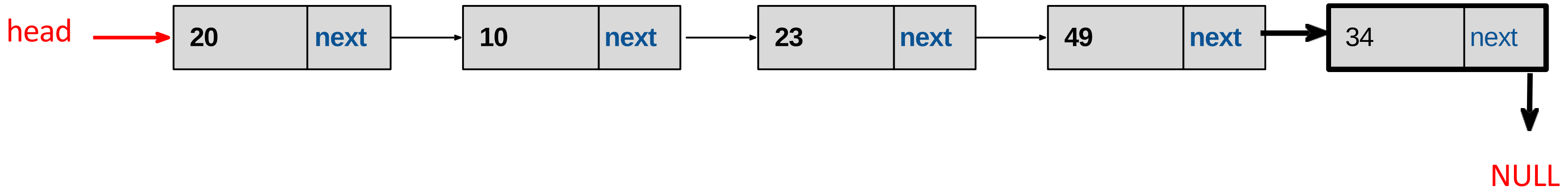
Singly Linked List Implementation

- `insertLast(34);`



Singly Linked List Implementation

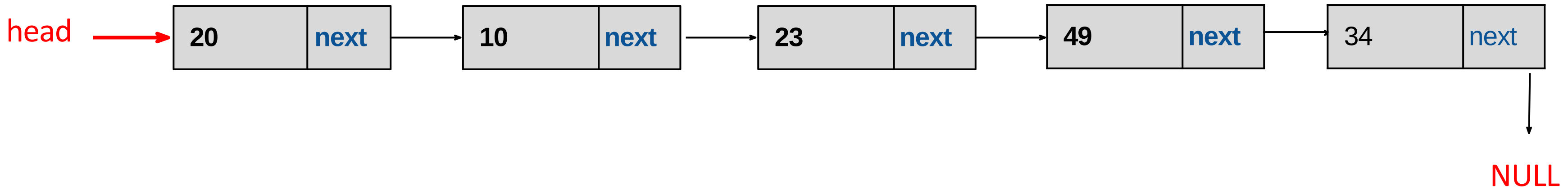
- `insertLast(34);`



Time complexity? O(n)

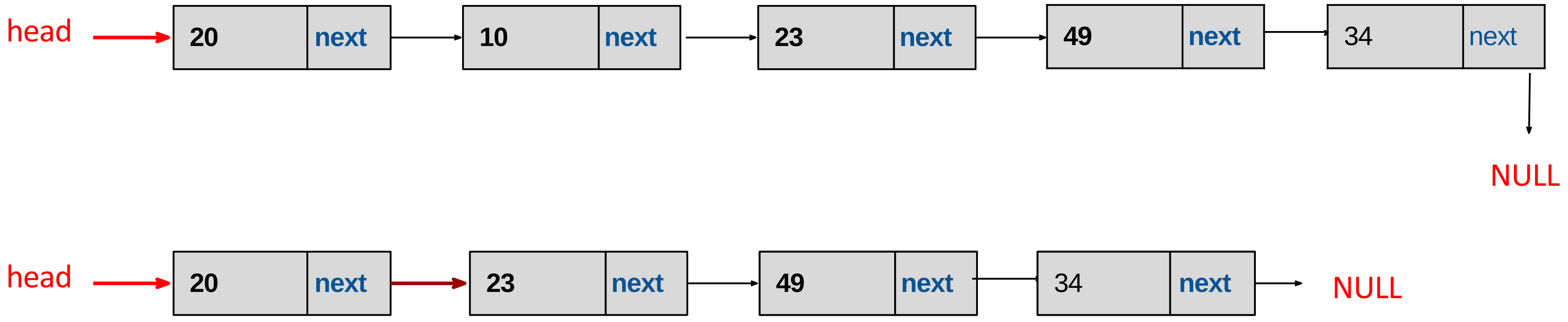
Singly Linked List Implementation

- `remove(1);`



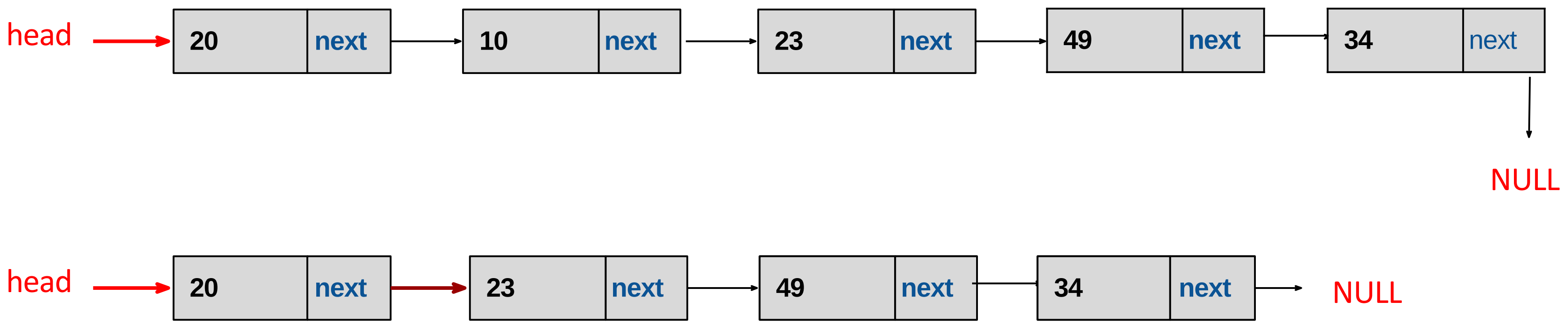
Singly Linked List Implementation

- `remove(1);`



Singly Linked List Implementation

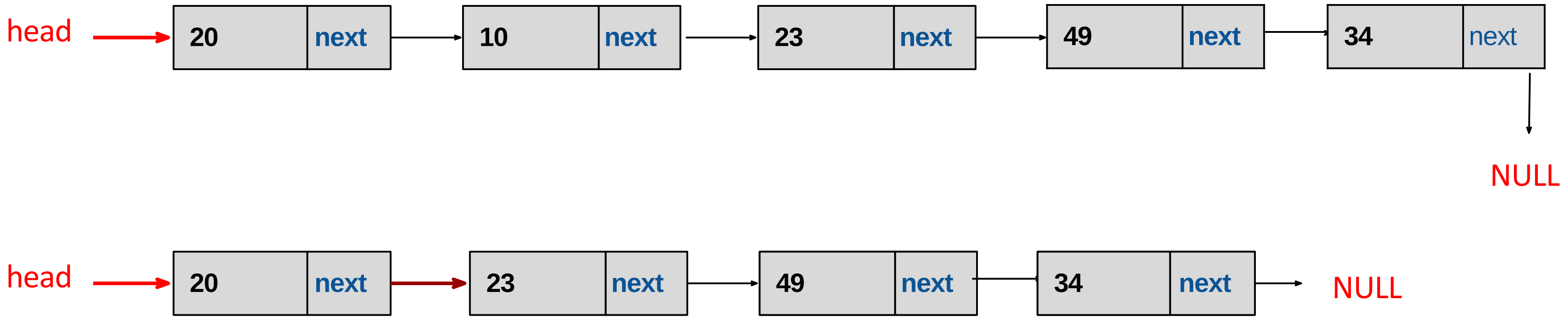
- `remove(1);`



Time Complexity?

Singly Linked List Implementation

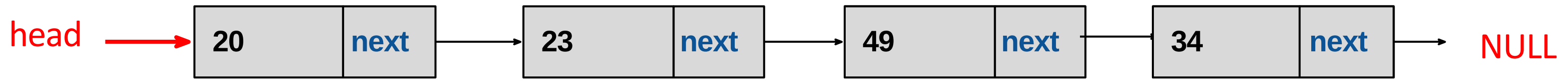
- `remove(1);`



Time Complexity? O(n)

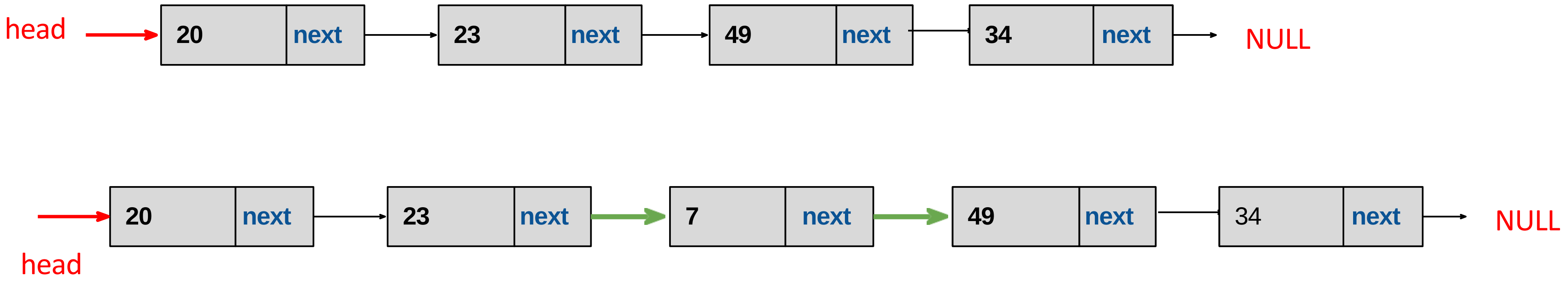
Singly Linked List Implementation

- `insertAt(7,2);`



Singly Linked List Implementation

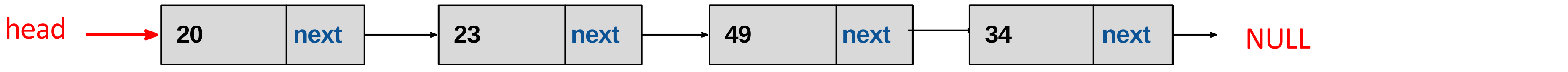
- `insertAt(7,2);`

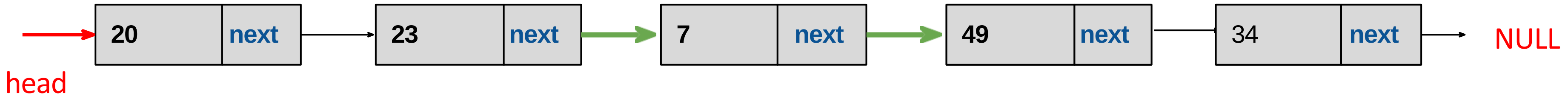


Time Complexity?

Singly Linked List Implementation

- `insertAt(7,2);`

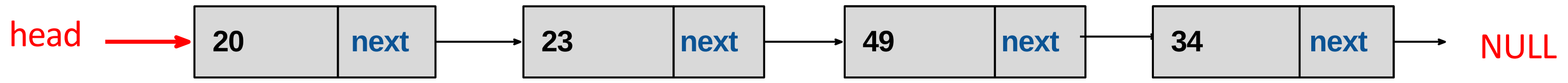




Time Complexity? O(n)

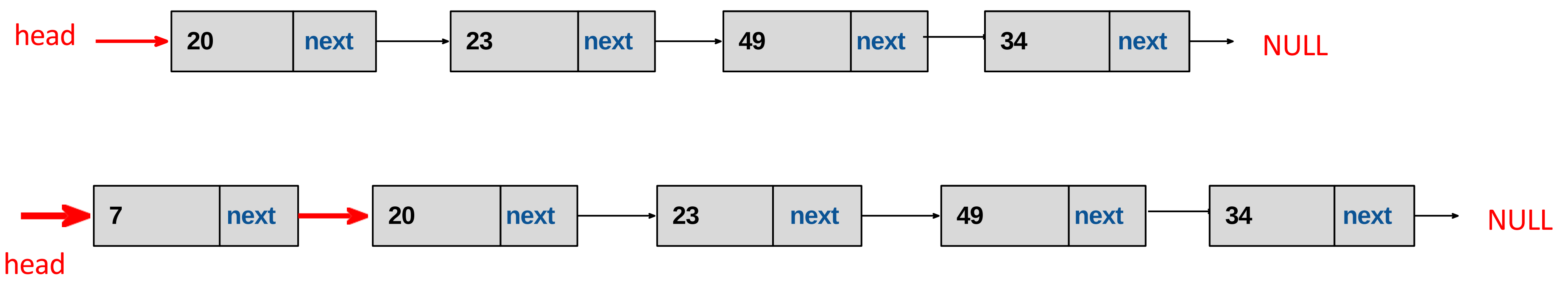
Singly Linked List Implementation

- `insertBeg(7);`



Singly Linked List Implementation

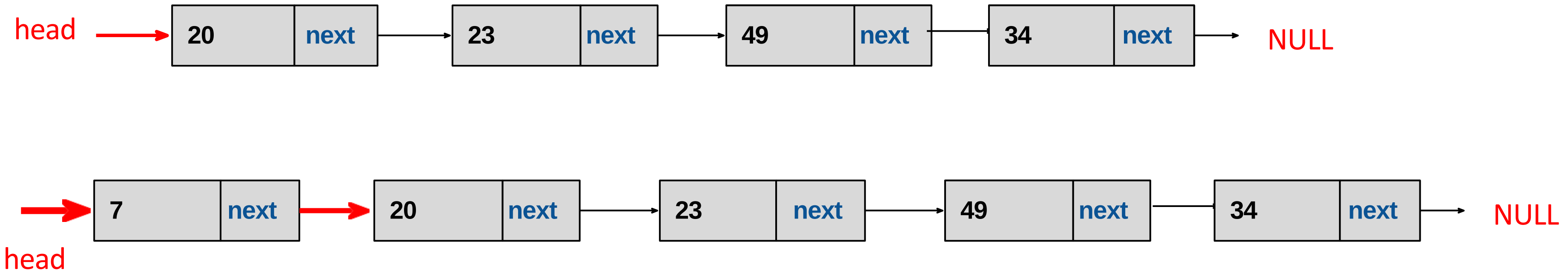
- `insertBeg(7);`



Time complexity?

Singly Linked List Implementation

- `insertBeg(7);`

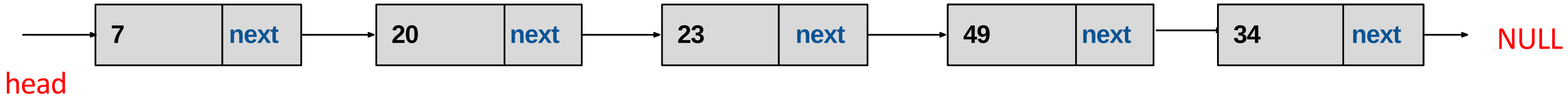


Time complexity? O(1)

For the previous functions, in the worst case scenario, we need to traverse the whole linked list. However, for insertBeg(int i) function, we don't need to traverse the linked list as we are adding a new node with data "i" in the front of the list (we only need to update a couple of pointers)

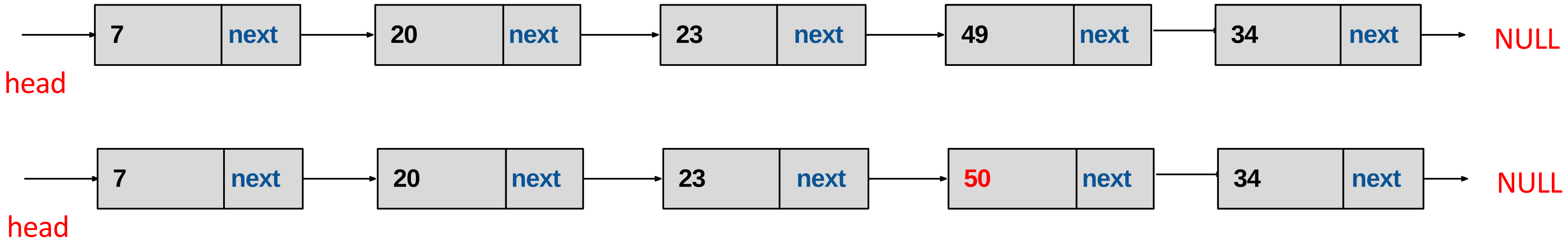
Singly Linked List Implementation

- `replace(3,50);`



Singly Linked List Implementation

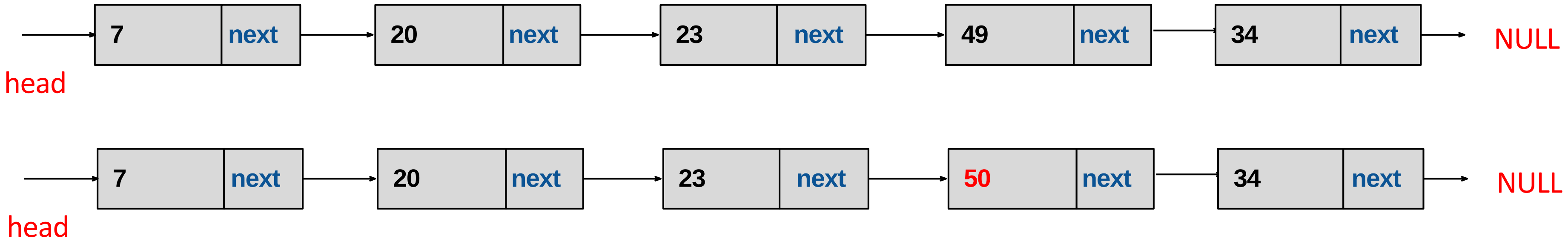
- `replace(3,50);`



Time complexity?

Singly Linked List Implementation

- `replace(3,50);`



Time complexity? O(n)

In the worse case we are going to replace the last element of the list. So, we need to traverse the whole list

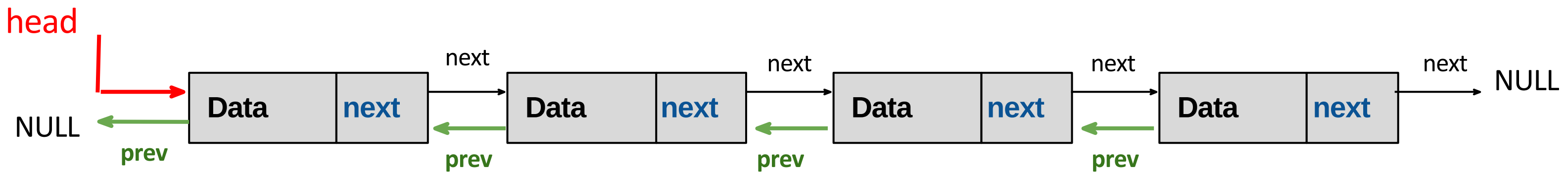
Singly Linked List Implementation

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

- Let’s implement some of singly linked list functions!

Doubly Linked List (DLL) implementation

A DLL can be traversed in both forward and backward direction



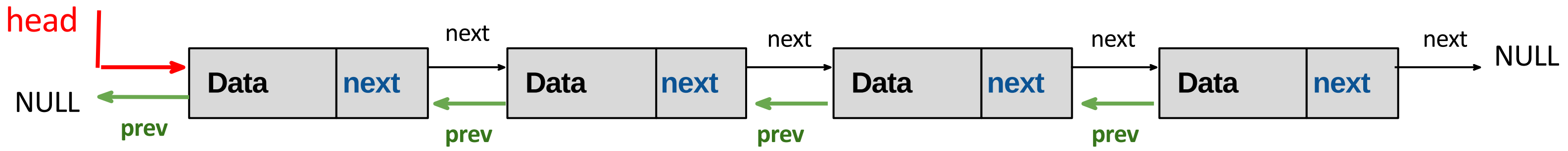
Doubly Linked List (DLL) implementation

Advantages over Singly linked list

- A DLL can be traversed in both forward and backward direction
- The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
- We can quickly insert a new node before a given node

Disadvantage

- All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers



Doubly Linked List implementation

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

Exercise :

- Given a node as prev_node, insert a new node after the given node:

```
        void insertAfter(Node* prevNode, int newData)
```

Other resources

A Comprehensive Guide To Singly Linked List Using C++
(No best practices, but easy to understand)
https://www.codementor.io/@codementorteam/a-comprehensive-guide-to-implementation-of-singly-linked-list-using-c_plus_plus-ondlm5azr

(Better practices)

https://github.com/kamal-choudhary/singly-linked-list/blob/master/Linked%20List.cpp

https://gist.github.com/csaybar/d0451939e79c16456095d6ed59bd718f

Another example

https://gist.github.com/charlierm/5691020