Al Hussein
Technical University
جامعة الحسين التقنية

Data Structures and Algorithms

Linked List

Prepared by:

Eng. Malek Al-Louzi

School of Computing and Informatics – Al Hussein Technical University

Spring 2021/2022

Outlines

- Introduction to Linked List
- Array Limitations
- Why we need Linked List
- Linked List – Insertion
- Array vs Linked List
- Doubly Linked List

Introduction to Linked List

- We implemented a dynamic List using Arrays.
  - It is inefficient in terms of memory usage.
- When we use arrays, we have some limitations.

To understand Linked List,

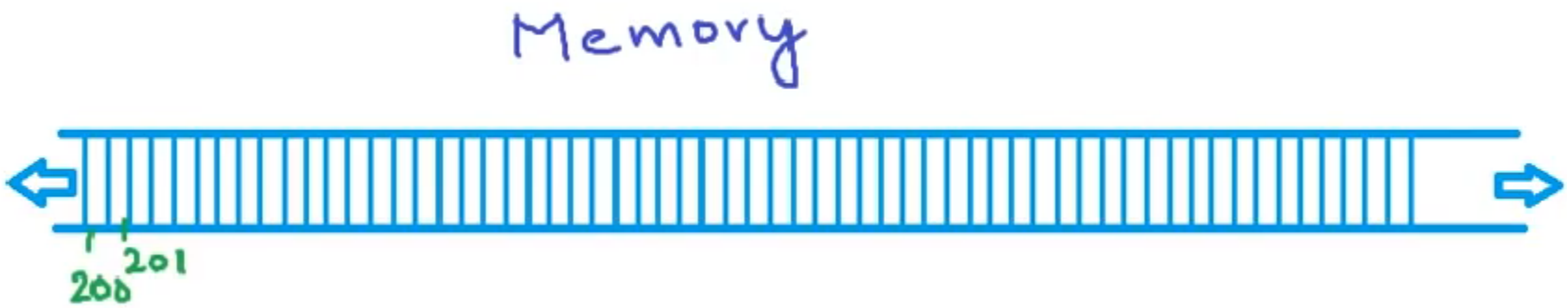
we need to understand

the

of Arrays !

Array Limitations (1/7)

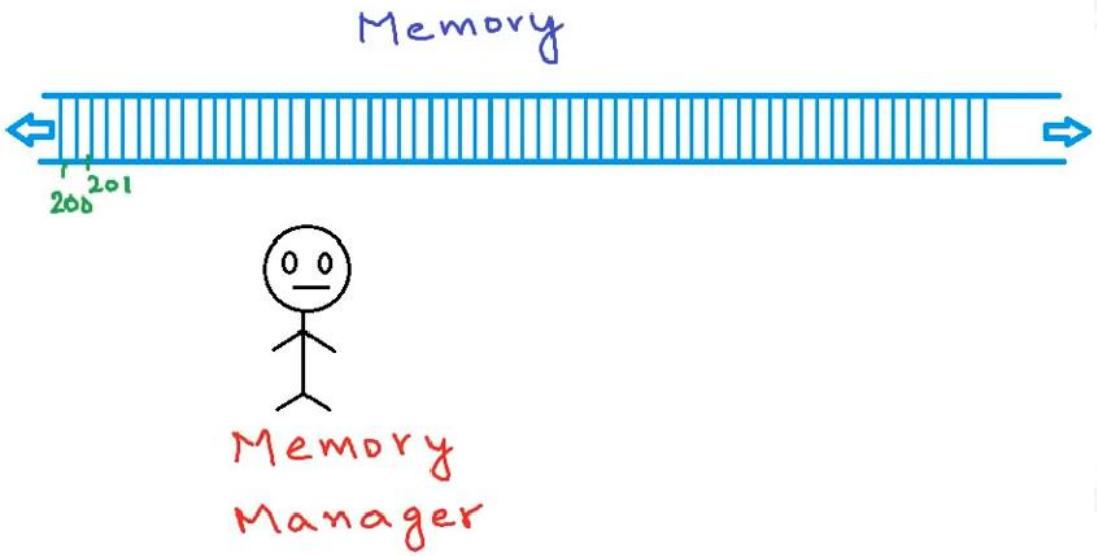
- The following is a section of computer memory:



- Each partition here is One Byte of memory.
- Each Byte has a unique address.

Array Limitations (2/7)

- Memory is a crucial resource; all the applications are using it.
- Let us suppose the Computer gives the role of managing memory to a memory manager.
  - Keeps track of what part of memory is free and what part is allocated.
  - Any application needs a memory to store data.

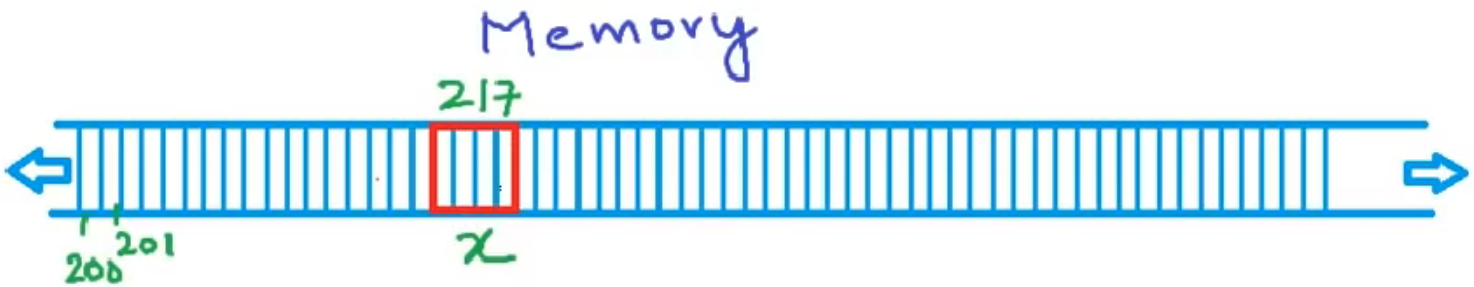


Array Limitations (3/7)

- The programmer can communicate with the memory manager through High Level languages.

Programmer

int x;



Memory Manager

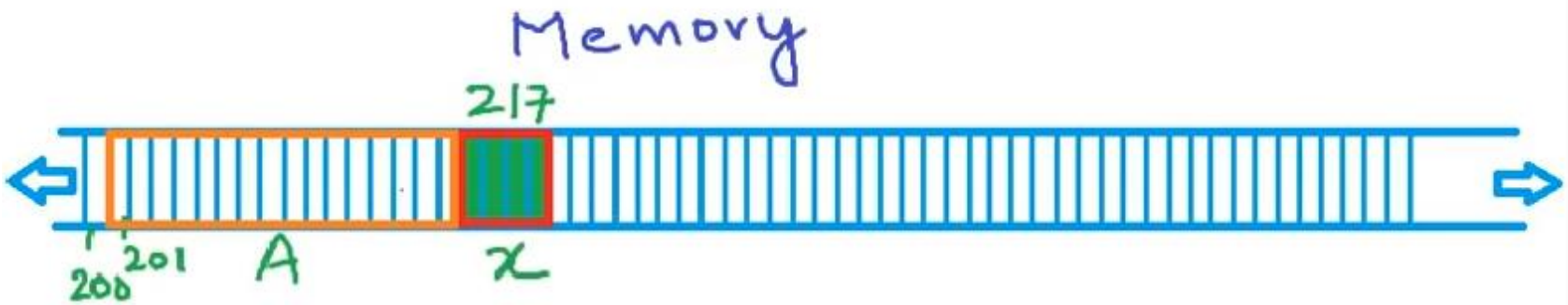
Array Limitations (4/7)

- The memory manager will do:
  - Searching for a free space to store an integer (Four Bytes).
  - The address of **x** is the address of the first Byte of the Four bytes.
- The programmer now wants to store a List of Integers, he supposed that the maximum number of this List will be Four.
  - Elements of Array are always contiguous in the Memory.

Array Limitations (5/7)

Programmer

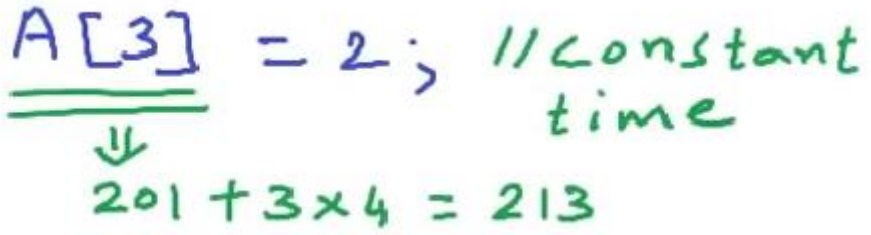
```
int x;
x = 8;
int A[4];
```



- The memory manger will search for 16 contiguous free bytes for the array.

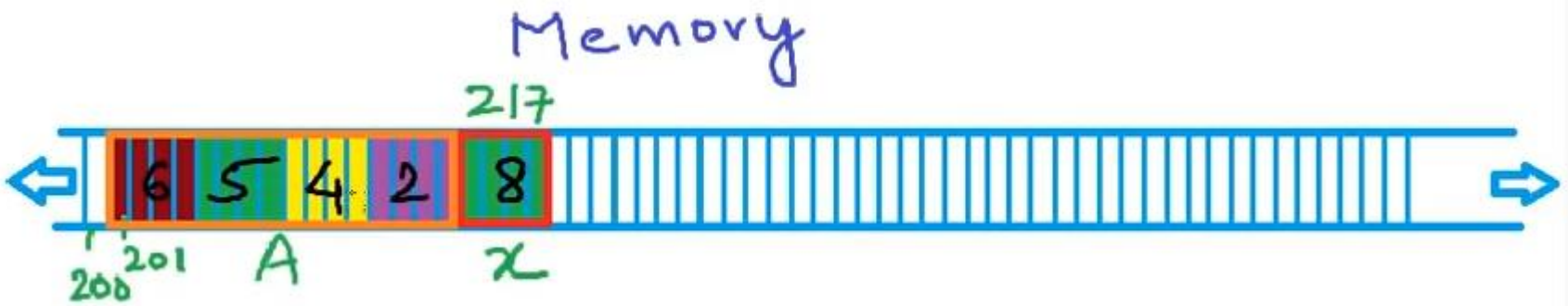
Array Limitations (6/7)

- The programmer can access any element in the array.
  - The programmer's application knows exactly where is that element.
  - This will take a constant time.
  - This is the good thing about arrays.
  - So irrespective of the size of array, we can access any element in the array in **Constant Time (O(1))**.



Array Limitations (7/7)

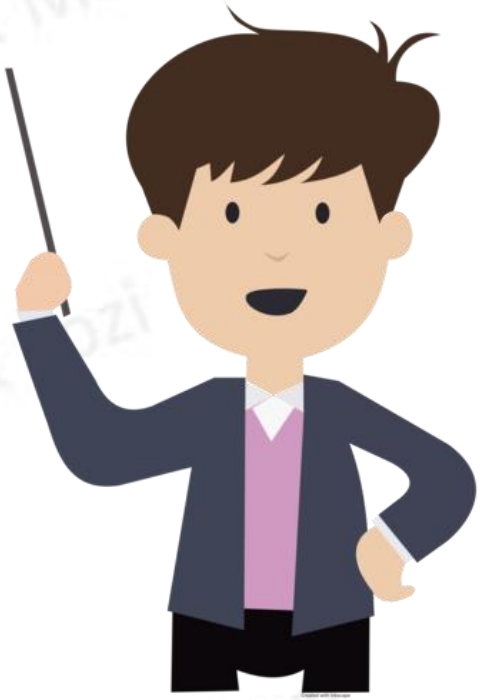
- If the programmer filled his List using the array:



- If the programmer needs to insert one more element to the list!
- It is clear now why we cannot increase the size of the same array.
  - The variable **x** is next to the array block.
  - It might be any other variable that declared in the program.

The solution is to use a data structure named

“Linked List”

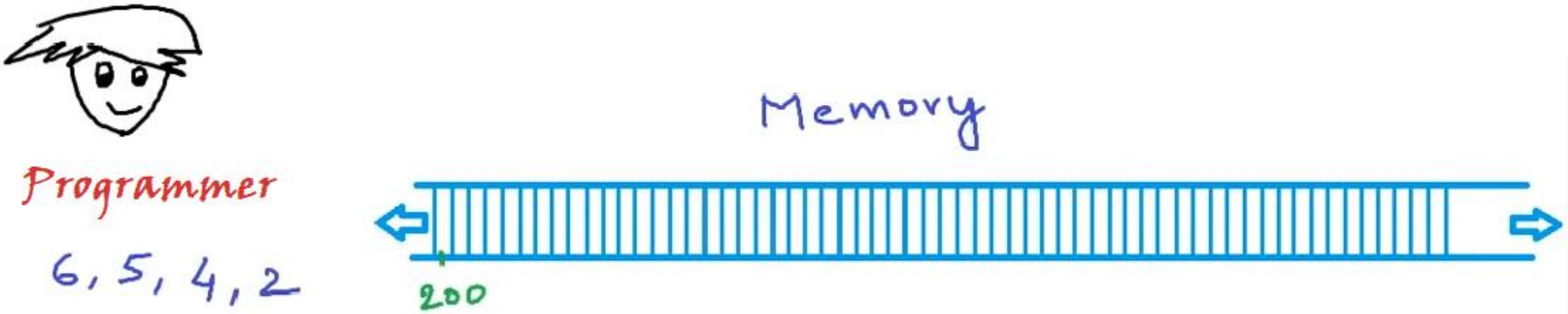


Linked List (1/12)

- Instead of declaring a one contiguous block of memory for all elements, we can declare one element at a time in a separate request.

Linked List (2/12)

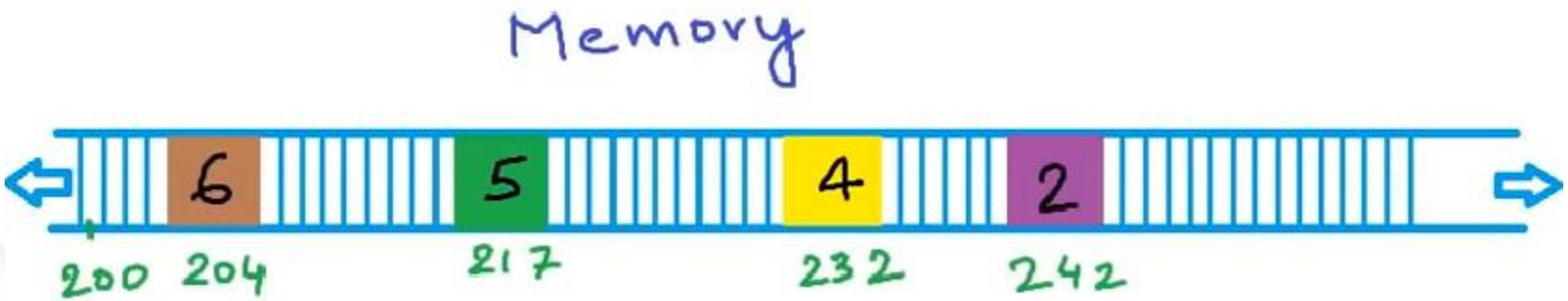
- The programmer wants to store a List of Four integers.



- He will declare one element at a time.

Linked List (3/12)

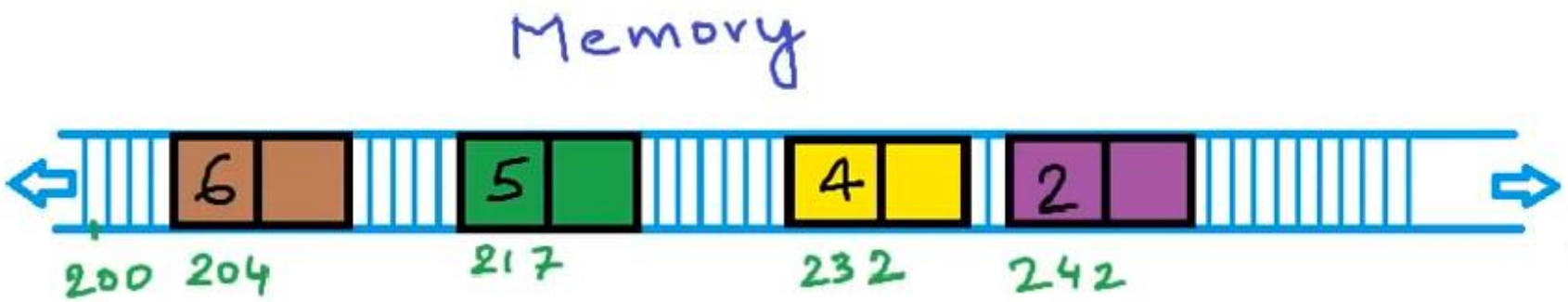
- The locations of the elements may not be adjacent, because of the separate declarations.



- We need to store additional information, for the order of the elements in the List (First element, second element, and so on).

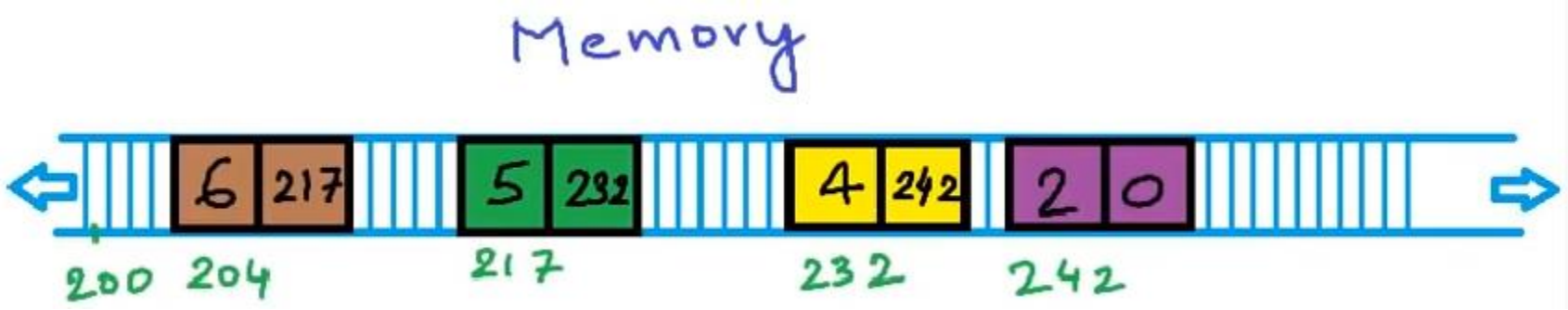
Linked List (4/12)

- So, we need to link these separate elements together some how.
- In Array it was simple because we have a one contiguous block of memory.
- To link the elements together, we can store extra information with each element, we will call it a block now.



Linked List (5/12)

- We store two parts in each block.
  1. One part for the data.
  2. The other part is a reference for the next block.



- The reference in the last block is 0 (NULL).
  - Means there is no next block after this block.

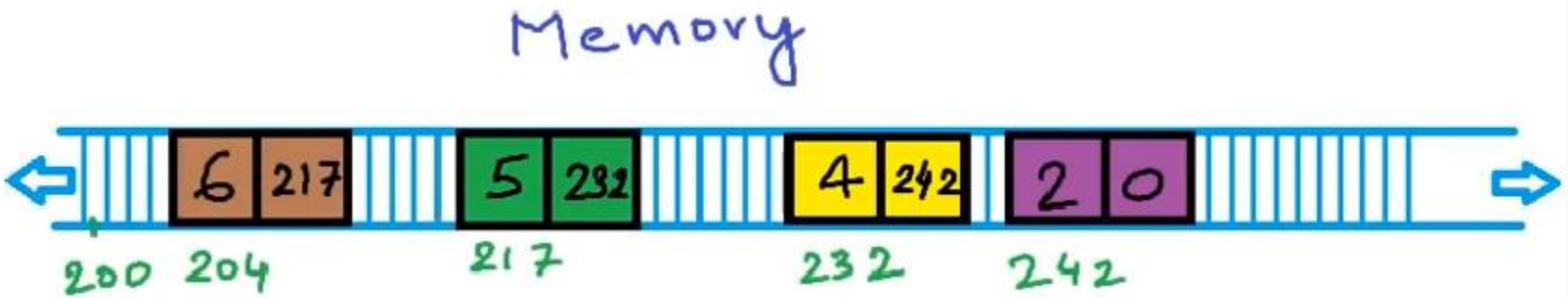
Linked List (6/12)

- We will call this block (data and a reference to next block) a **Node.**
- To declare this Node, we can use a class called Node with two fields.
  - Field for Data and another field for the next Node.
  - The Data could be any thing we want (String, char, …etc).

```
class Node{

    int data;
    Node next;
}
```

Linked List (7/12)

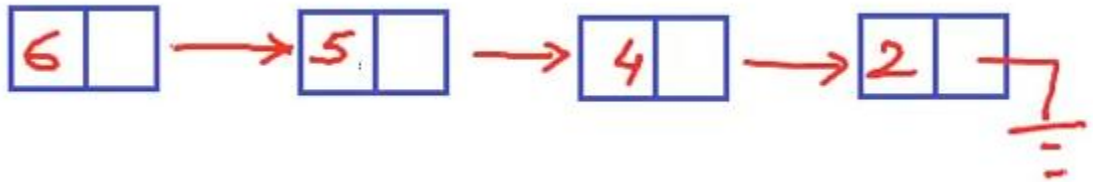
- The programmer now can declare an object for each element in the List from the Node class.
- So, if we store the List like this in the memory:
  - **Noncontiguous Nodes connected to each other.**



- This is a **Linked List** data structure.

Linked List (8/12)

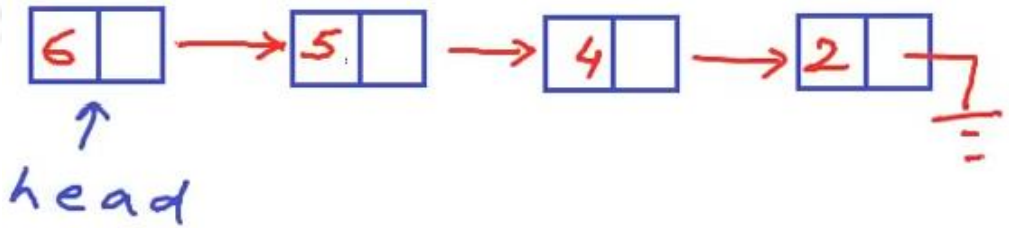
- Logical view of Linked List data structure:



- Data is stored in each Node.
- Each Node stores Data and a reference to the next Node.
- Each Node will point to the next Node.

Linked List (9/12)

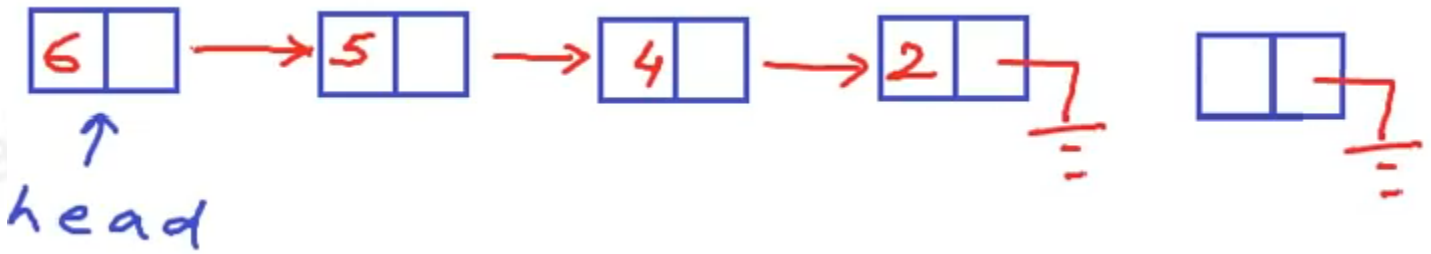
- The first Node called the head Node.



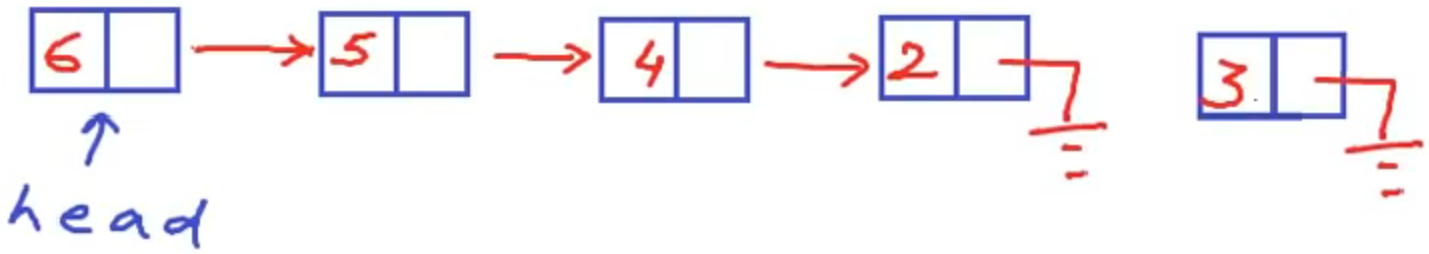
- The main information about the List that we keep all the time is a reference to the head Node.
- To traverse the Linked List, we always start at the head Node.
  - From the head Node we can know where is the next Node.
  - From the next Node we can know where is the next Node, and so on.
  - This is the only way to access the elements in the linked list.

Linked List (10/12)

- If we want to insert a Node in the linked list, for example we want to add number 3 to the List:
  - First, we create a new Node.

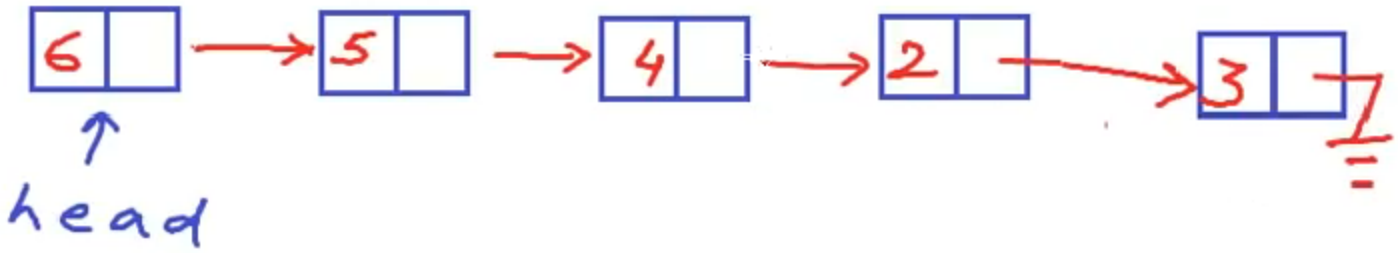


  - Fill the data in this new Node.

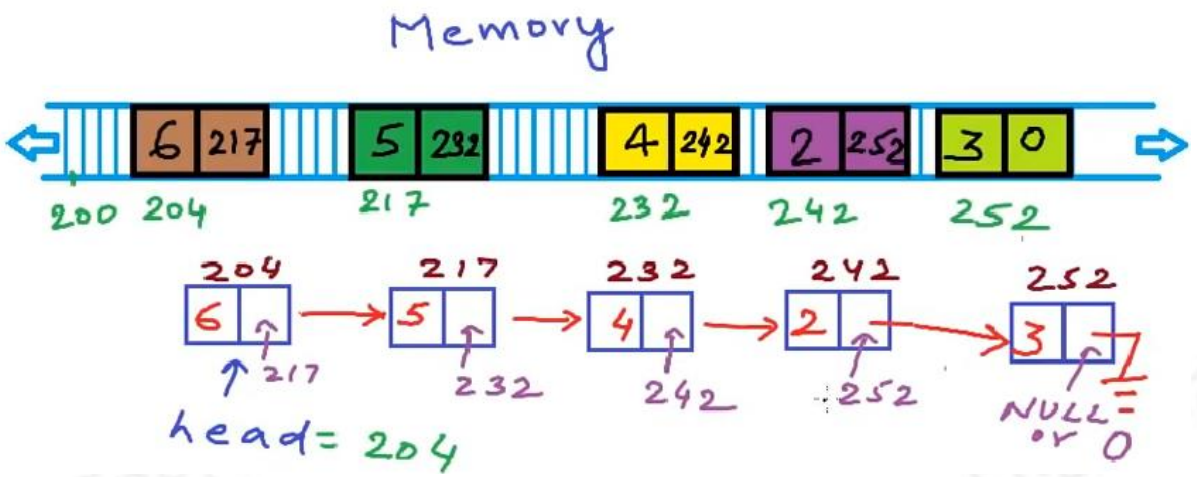


Linked List (11/12)

- Update the reference part in the last Node.



- In the memory, this List will be like this:

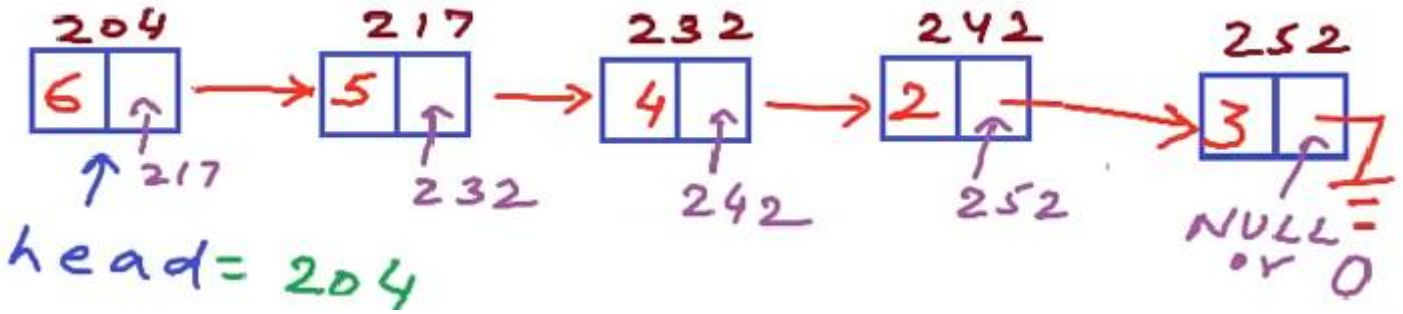


Linked List (12/12)

- The main information about the List that we keep all the time is a reference to the head Node.
- Unlike arrays, we cannot access elements in a constant time.
- Access to elements in Linked List is O(n) in the worst case.
  - Worst case means that the element that we want to access is at the end of the linked list, so we need to traverse all the elements.

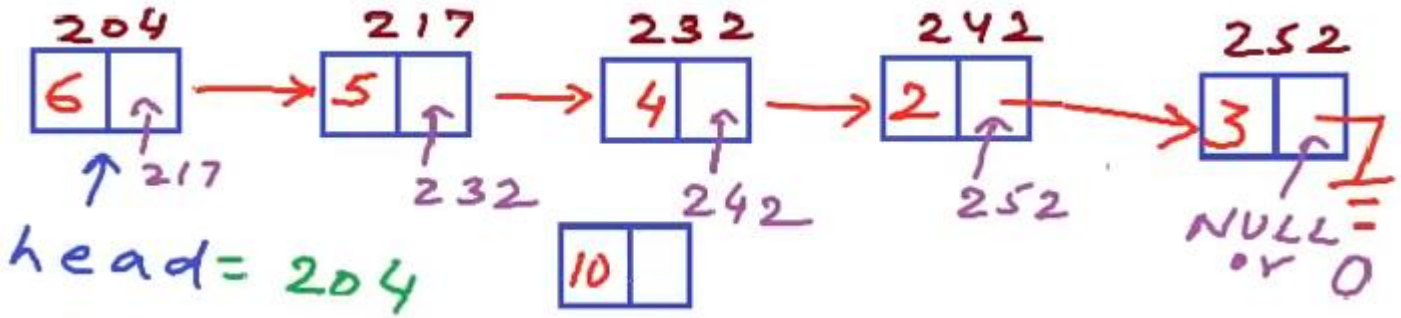
Linked List – Insertion (1/3)

- Insertion in the linked list, we can insert anywhere in the linked list.
- We just need to declare a new Node and update some links.
- For example, we want to insert number 10 in the third position in the list.

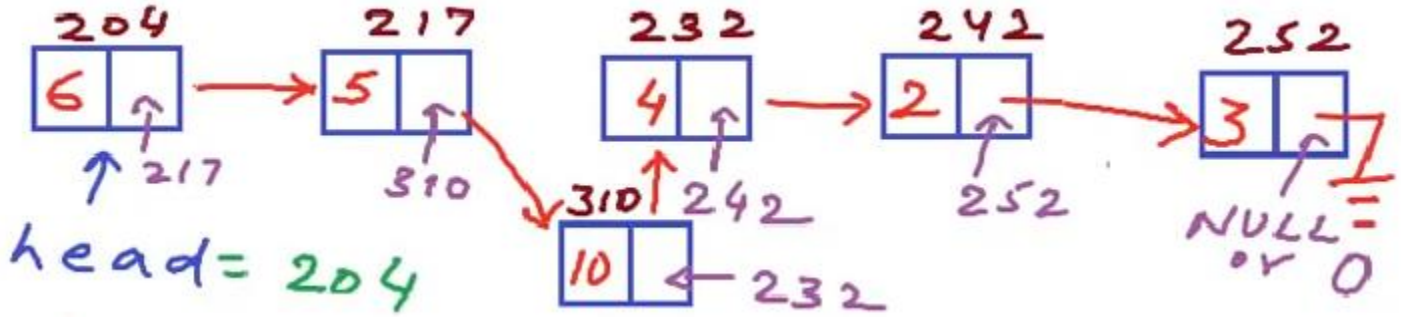


Linked List – Insertion (2/3)

- First, we create a new Node, and store the value 10 in the data part.



- Adjust the reference part in the second node and in the new node.



Linked List – Insertion (3/3)

- To insert, we should traverse this list to the particular position.
  - Time complexity will be O(n).
  - The insertion is simple, without shifting elements like array.
- To delete an element also will take O(n).
- We can see there is no extra use of memory (only for references).
  - We can add a node when want and delete a node when we want.
  - We don’t have to guess the size of the List before creating the Linked List.

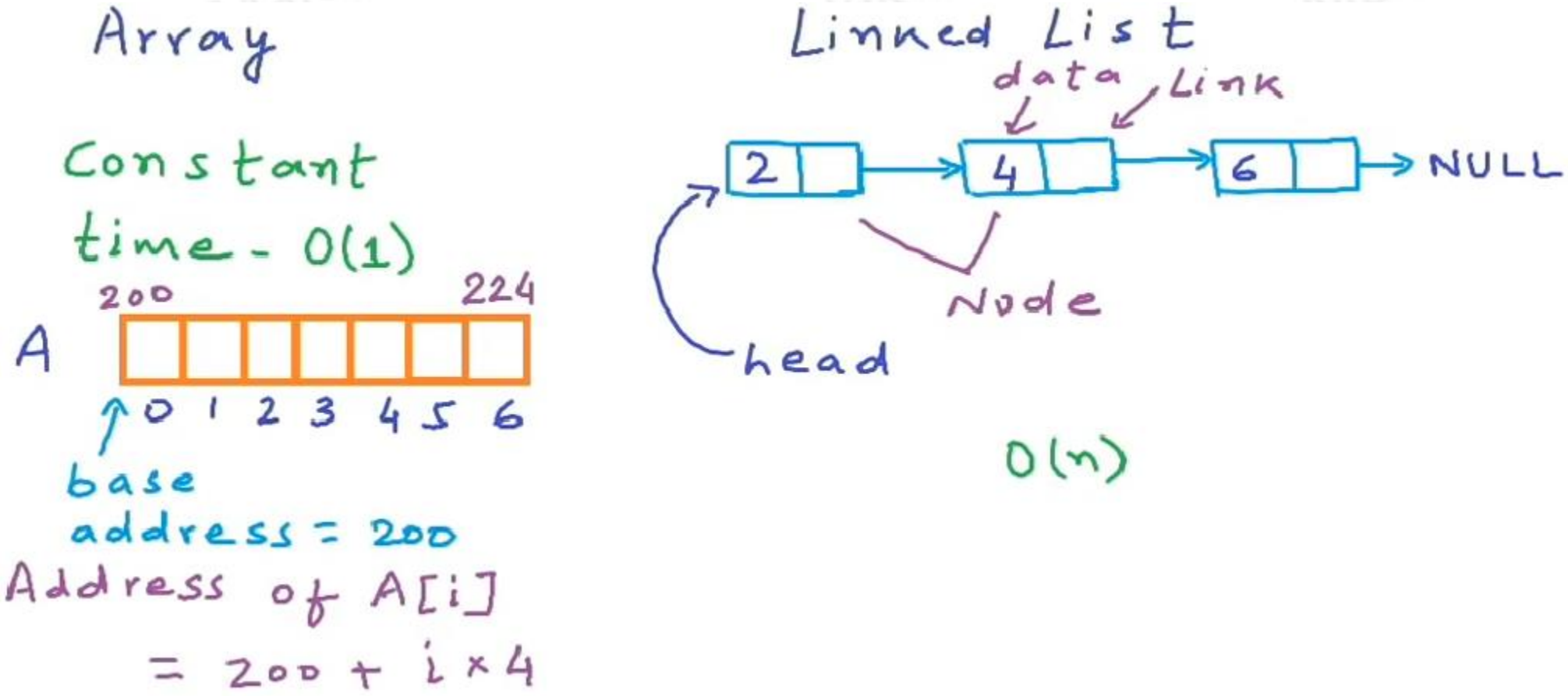
Which one is Better?

Array vs Linked List (1/5)

- There is no such a question in data structures.
  - One data structure is better than another data structure!
- One data structure maybe good for one kind of requirement while another data structure can be good for another kind of requirement.
- It depends on different factors:
  1. What is the most frequent operation that you want to perform with the data structure.
  2. The size of the data.

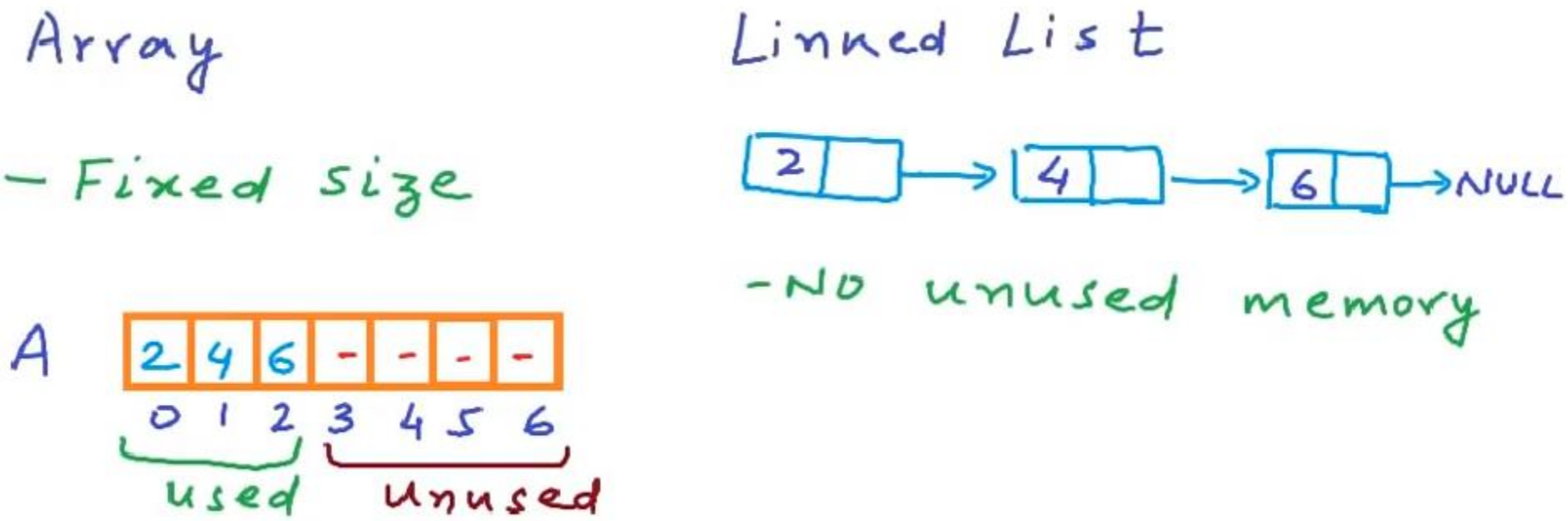
Array vs Linked List (2/5)

- Cost of accessing an element:



Array vs Linked List (3/5)

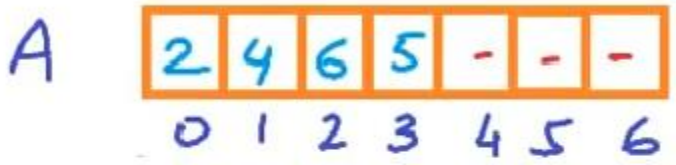
- Memory requirements:



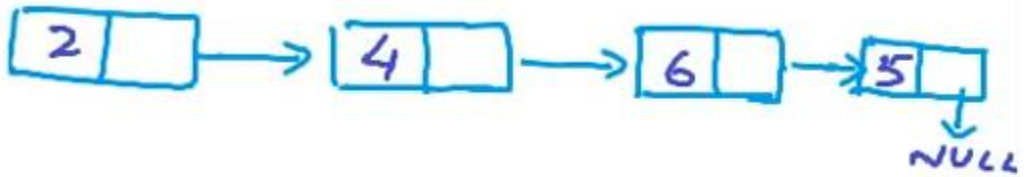
Array vs Linked List (4/5)

- Cost of inserting an element:

Array



Linked List



a) at beginning - O(n) O(1)

b) at end - O(1) O(n)

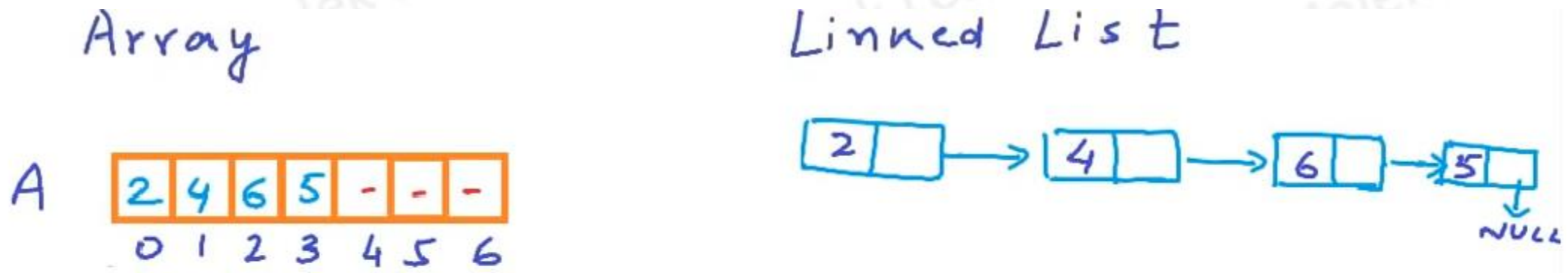
if array is not full

O(n) - array full

c) at i^{th} position - O(n) O(n)

Array vs Linked List (5/5)

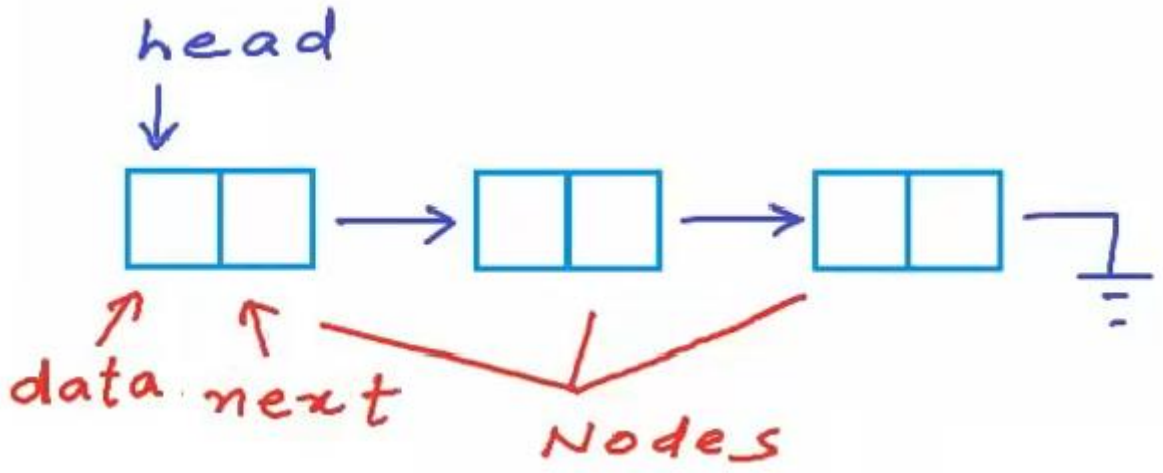
- Cost of deleting an element:



	Array	Linked List
a) at beginning -	O(n)	O(1)
b) at end -	O(1)	O(n)
c) at i^{th} position -	O(n)	O(n)

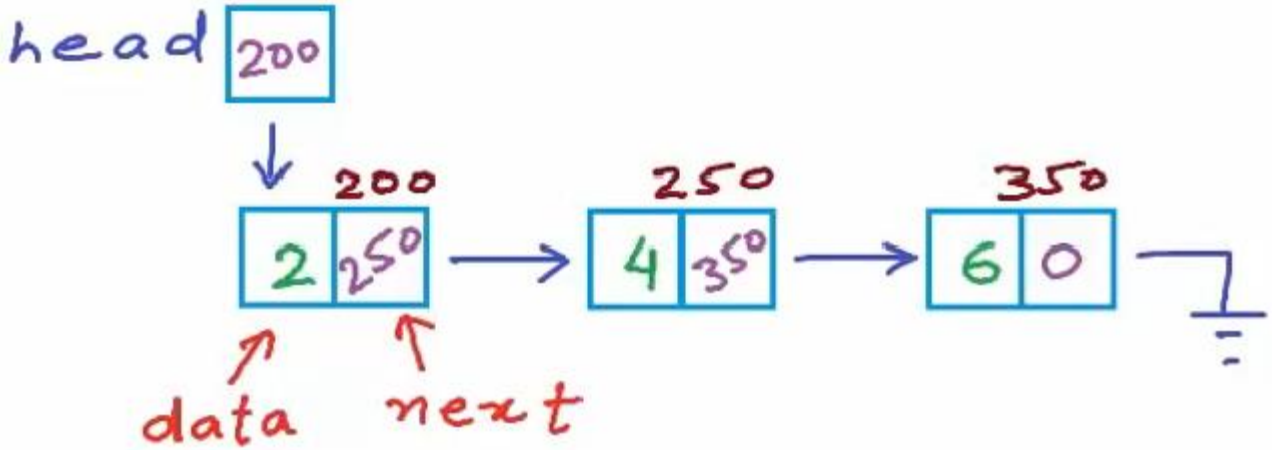
Doubly Linked List (1/6)

- Linked List is a collection of entities called Nodes.
- Each Node contains two fields: data and a reference to the next Node.
- The identity of the linked list is a reference to the head Node.



Doubly Linked List (2/6)

- We have the following linked list:



- By default, such a List is called **Singly** Linked List.

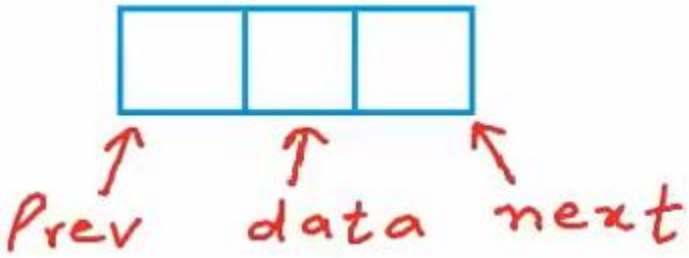
Doubly Linked List (3/6)

- There is another form of the Linked List called Doubly Linked List.
- In Double Linked List, each Node would have two links.
  - One to the next Node.
  - Another to the previous Node.

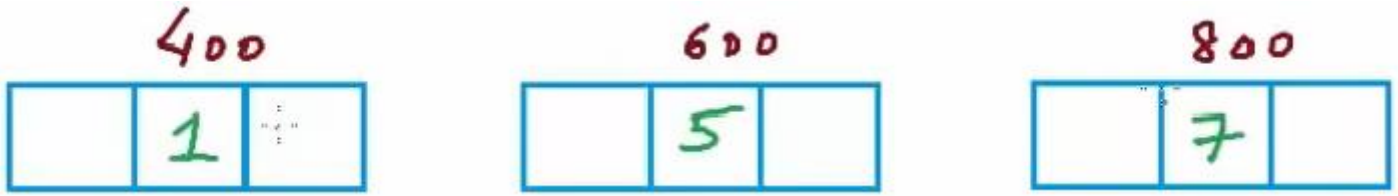
```
class Node{

    int data;
    Node next;
    Node previous;

}
```

Doubly Linked List (4/6)

- The Node will be like this:

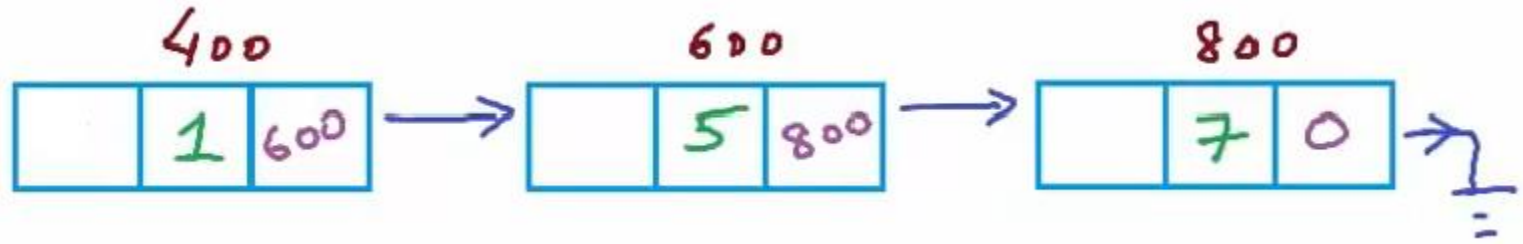


- Let us create a Doubly linked list of integers:

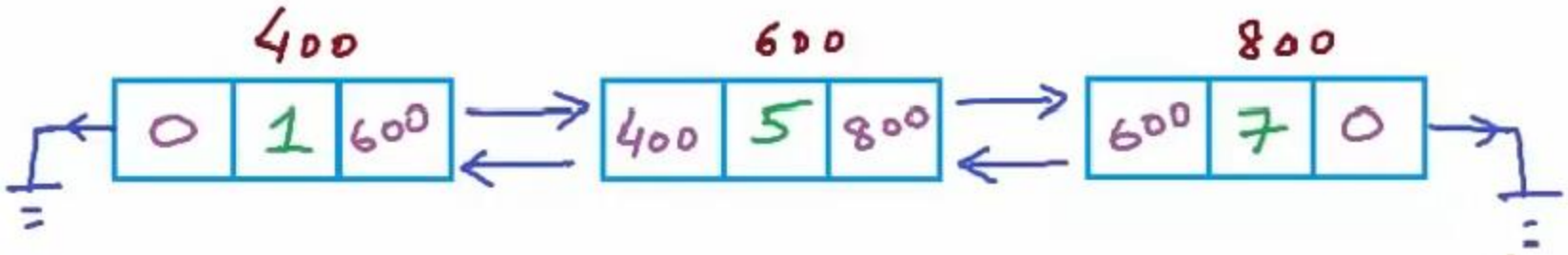


Doubly Linked List (5/6)

- Three Nodes, we will fill the reference to the next node:

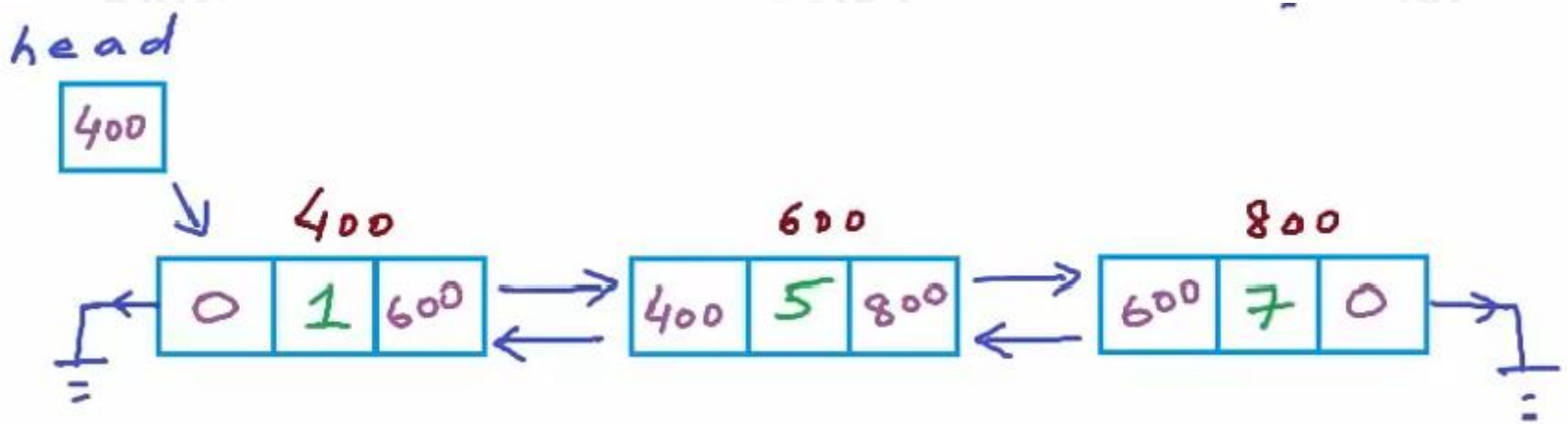


- Then the reference to the previous node:



Doubly Linked List (6/6)

- The last thing a reference to the head Node.



Any Questions???…