Data Structures

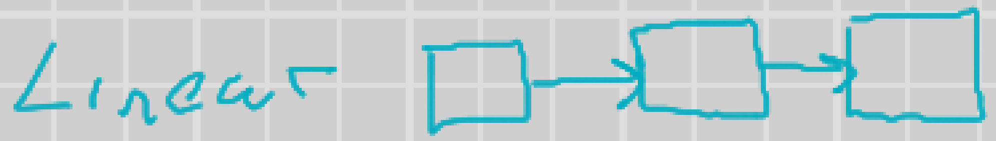
- Are containers where data is organized in a specific way and that have a set of specific operations to access and manipulate stored data.
- The simplest structures have a linear organization of data
  - lists, stacks, queues (1D arrays, hash tables)
  - Linear data structures are characterized by way they store data one element followed by another

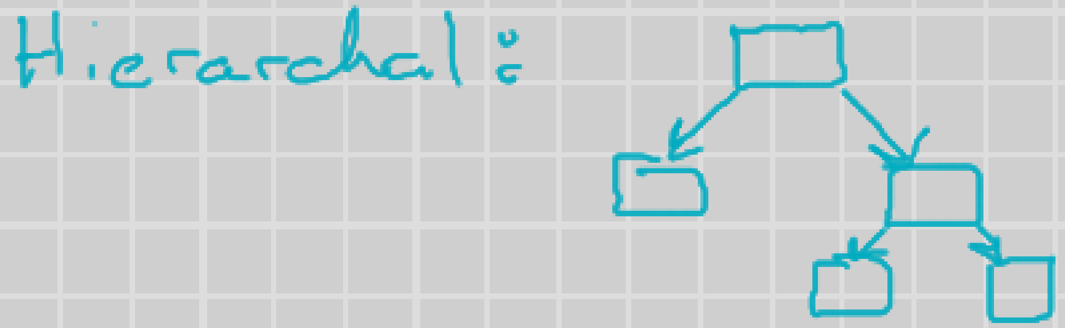
Another type of data structures have a hierarchal organization:

1) Trees
2) Heaps
3) Graphs

Data Structure:

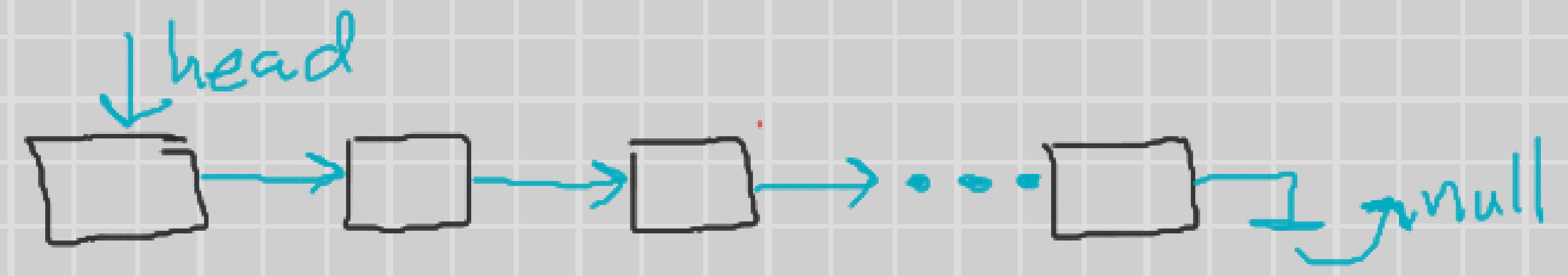
1) Is a CONTAINER of data.
2) Data is organized is a specific way. Lists are organized in a linear manner. Trees & Heaps are organized in a hierarchal fashion.
3) They have specific functions to manipulate data.

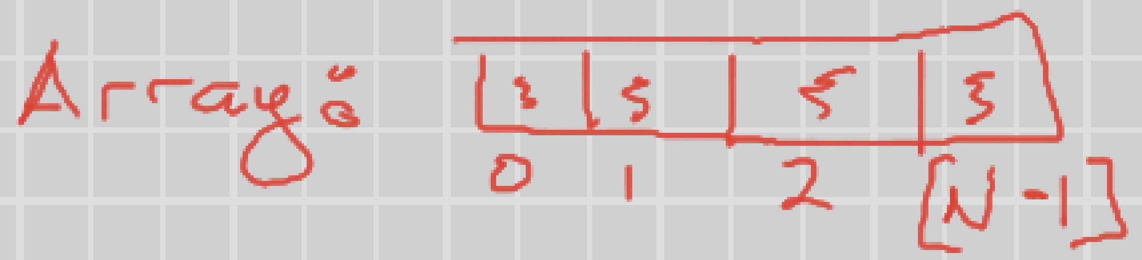




Linked Lists:

- Is a linear data structure, where data can be stored and visualised with one element immediately follwoing another.

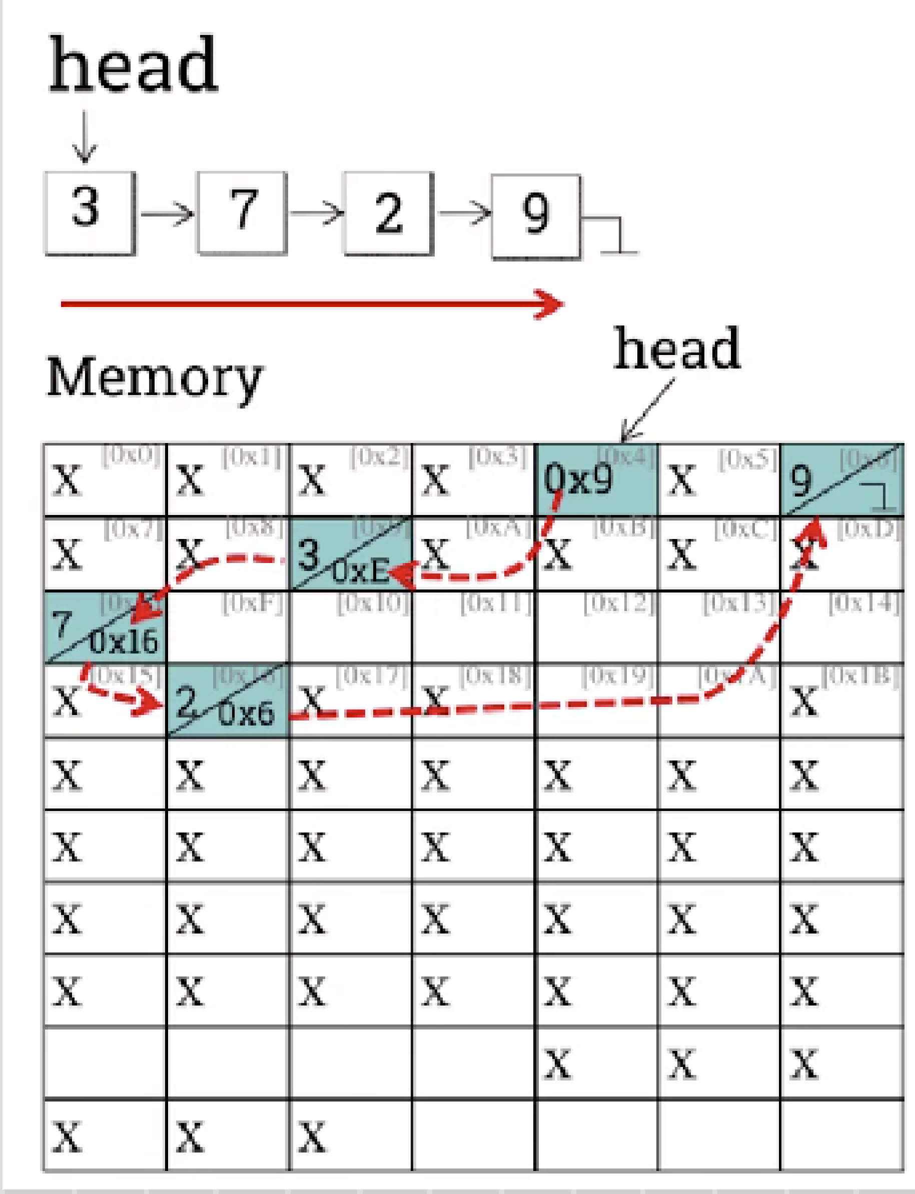




- The difference between he array and linked list is the way they are stored in memory.
- The way we access data from the structures.

An array requires a contiguous chunk of memeory, with (N) memory positions. Your computer needs to search for N contiguous memory positions. Acces is easier with ann array, looping involves less complex operations.

- Elements can be found simply by incrementing the current position by 1.

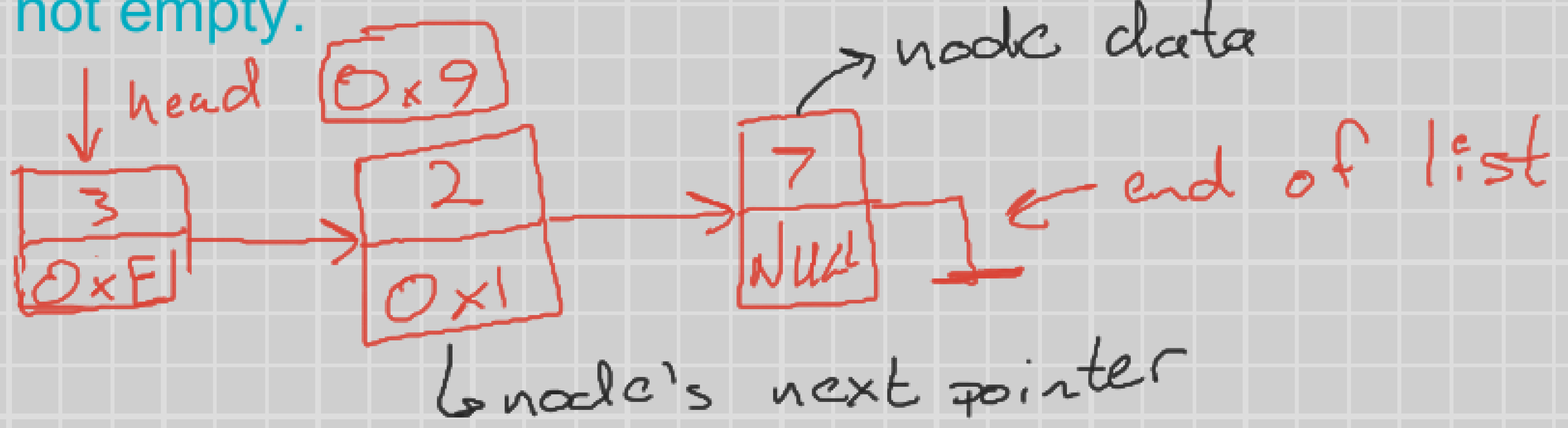


A linked list doesn't require continguous memory. The computer will create a list one element at a time, every time an element is created a new node is allocated to any available memory postion. Memory is allocated dynamically, can grow or shrink as needed.

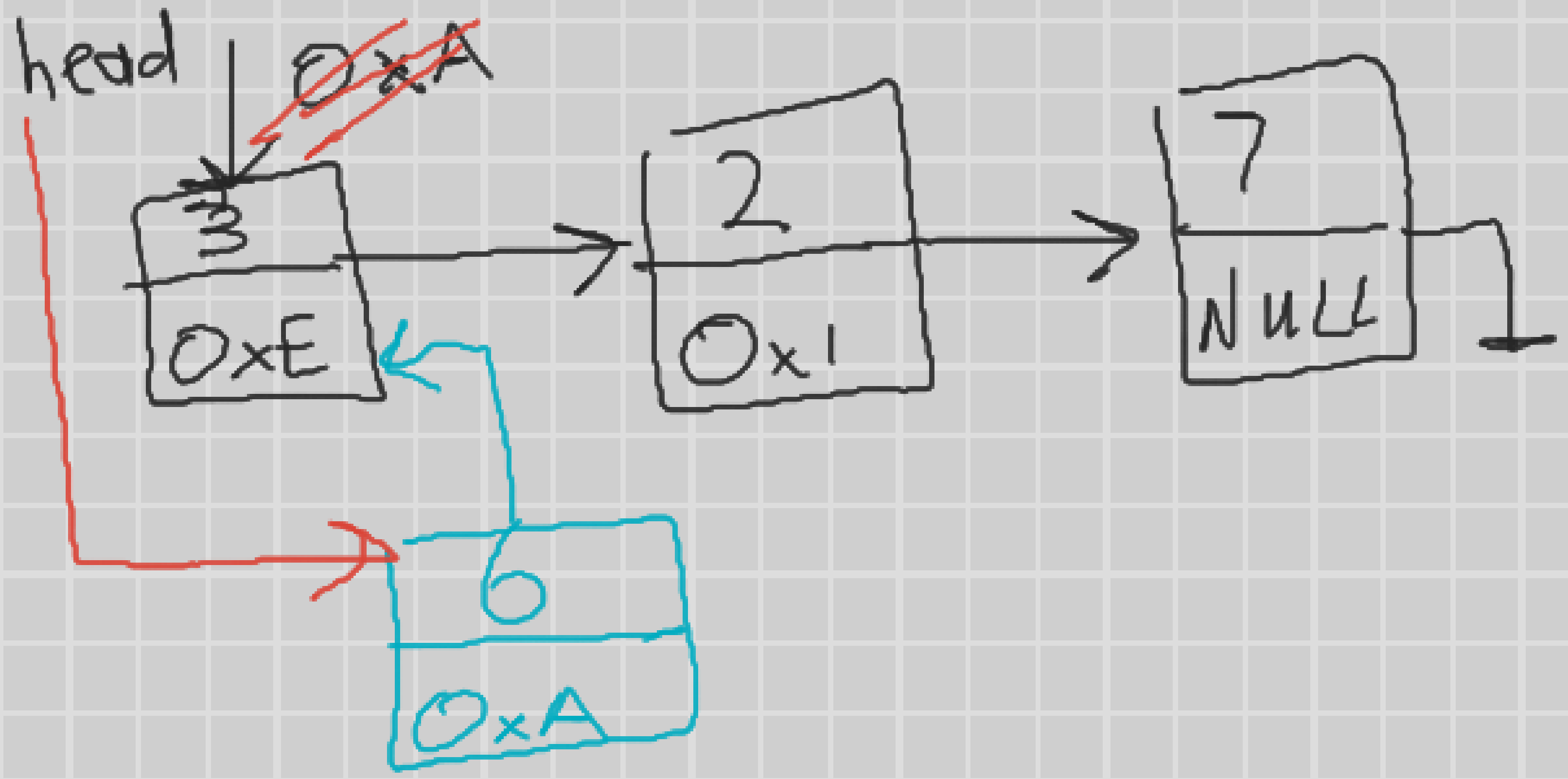
- Each element in a linked list points to the next one.
- Each element (we'll call it a node) stores two things (in a singly linked list):
  1) data
  2) a pointer to the next node (stores memory address)

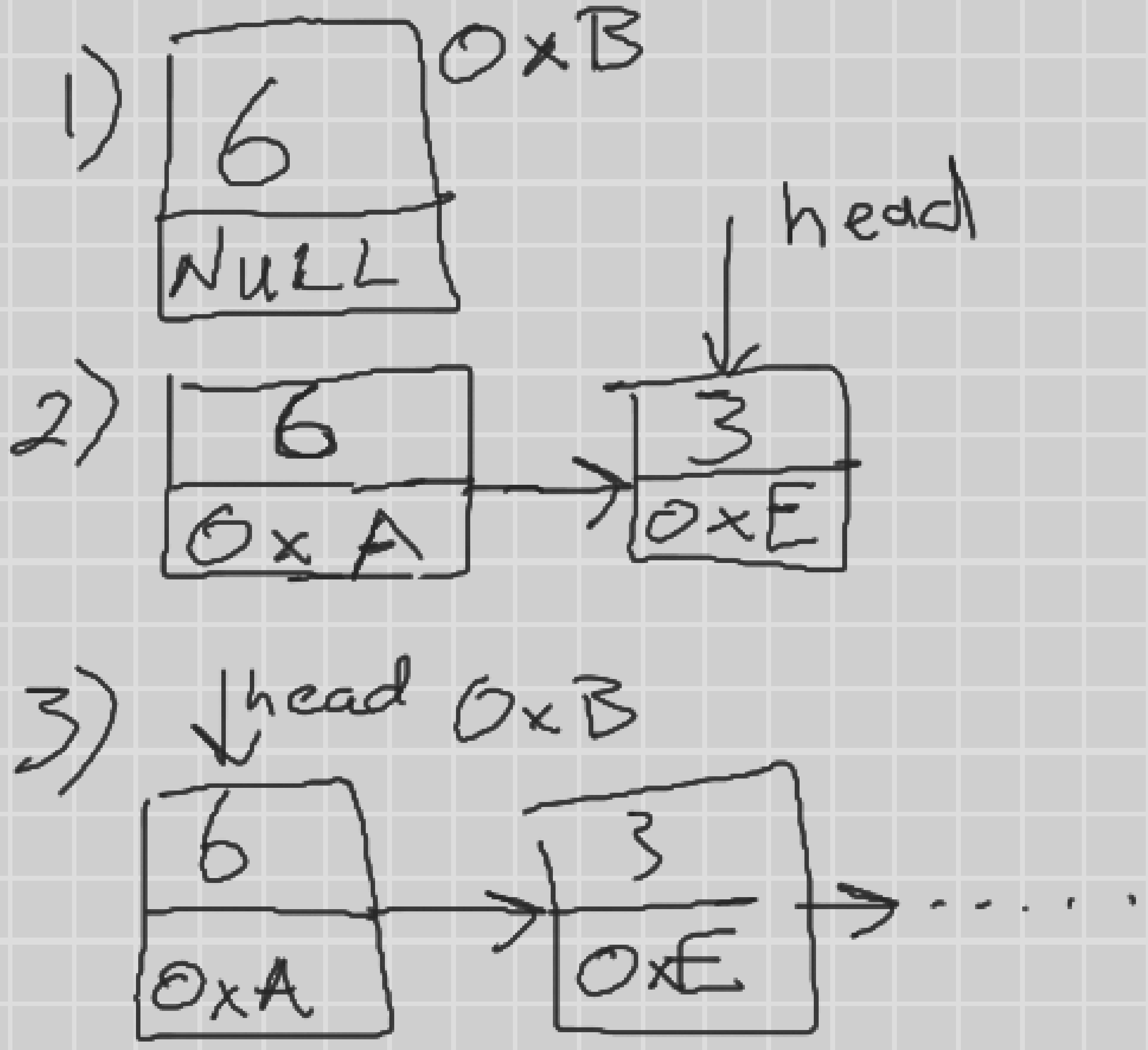
(A pointer is a memory address)

- The end of a linked list points to a special address know as NULL.
- The first time a list is created the HEAD points to NULL, if the list is empty.
- The HEAD of a linked list only stores a memory address to the first node, if the linked list is not empty.



Linked List Operation: INSERT





To insert an new node, we must take the following steps:
1) Create a new node, have ~~it pointing~~ to NULL initially.

Let's insert the node at the beginning of the list.

2) Point the next pointer to the address of the previous node that was at the beginning.

3) Point the head to the address of the new node.

```
function Insert(data, head)
    newNode = NEW NODE(x)
    newNode.next = head
    head = newNode
end function
```

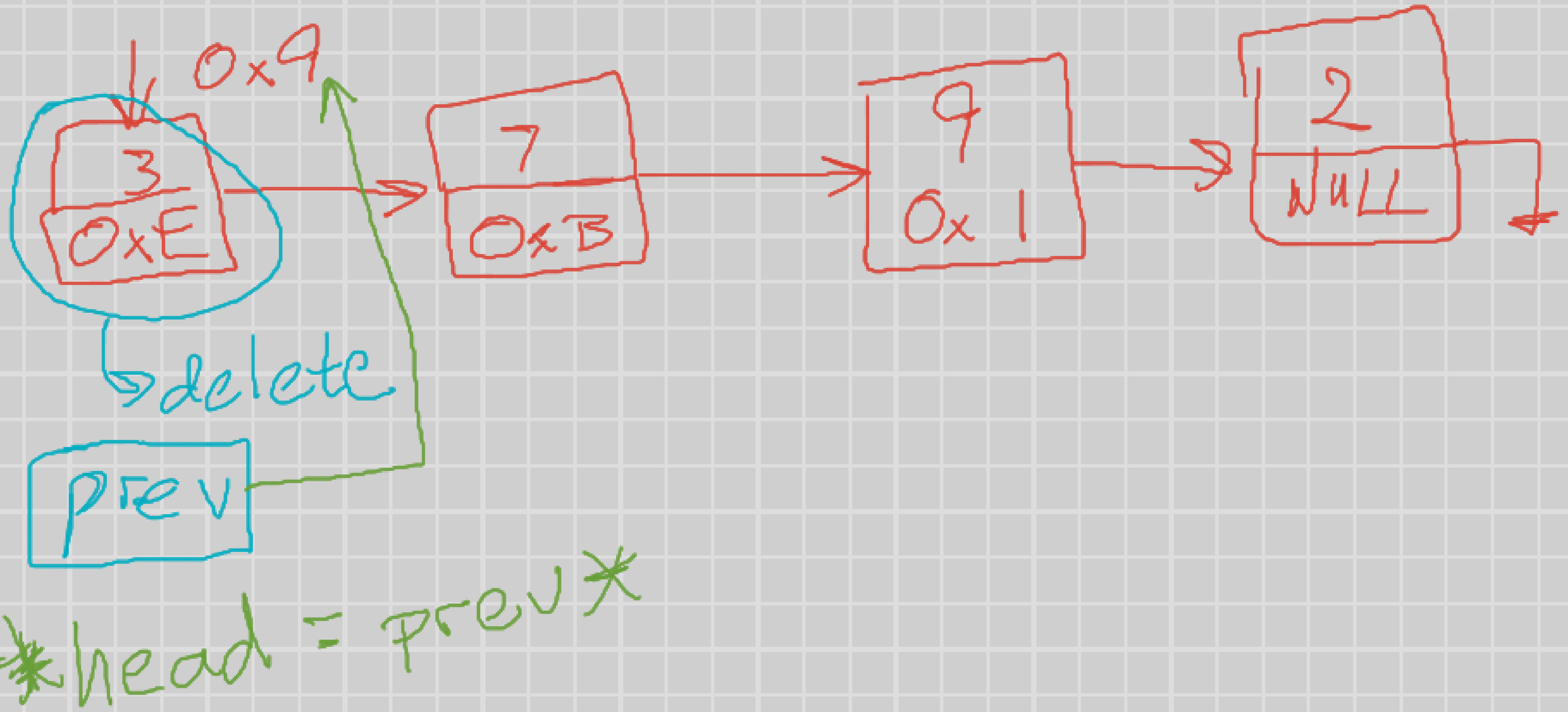
*Note that every vairable used is a pointer

If we are inserting a new node at the beginning of the list, we will carry out a constant number of operations. This will have a Time Complexity $T(N) = \theta(1)$.

If we insert the new node at the end of the list we always have to traverse the entire list until we get to the node that has its next pointer pointing to NULL. Then we must make that node point to the new one, and the new one must point to NULL.

If we insert the new node at an arbitrary position then we must traverse the list until we find the position we want to place the node and modify its next pointer to point to the address of the next pointer of the node that comes before it, and modify the next pointer of the node that comes before to point to the new node.

Linked List Operation: DELETE



prev.next = prev.next.next

```
function DELETE($list, x$)
    $tmp \leftarrow list.head$
    $prev \leftarrow NULL$
    if $tmp = NULL$ then
        print("There is nothing to delete")
        return $list$
    end if
    if $tmp.key = x$ then
        $list.head \leftarrow tmp.next$
        return $list$
    end if
    $prev \leftarrow tmp$
    $tmp \leftarrow tmp.next$
    while $tmp \neq NULL$ do
        if $tmp.key = x$ then
            $prev.next \leftarrow tmp.next$
            return $list$
        end if
        $prev \leftarrow tmp$
        $tmp \leftarrow tmp.next$
    end while
    print("Element not found")
end function
```

Empty list

Delete head

Searching for the element