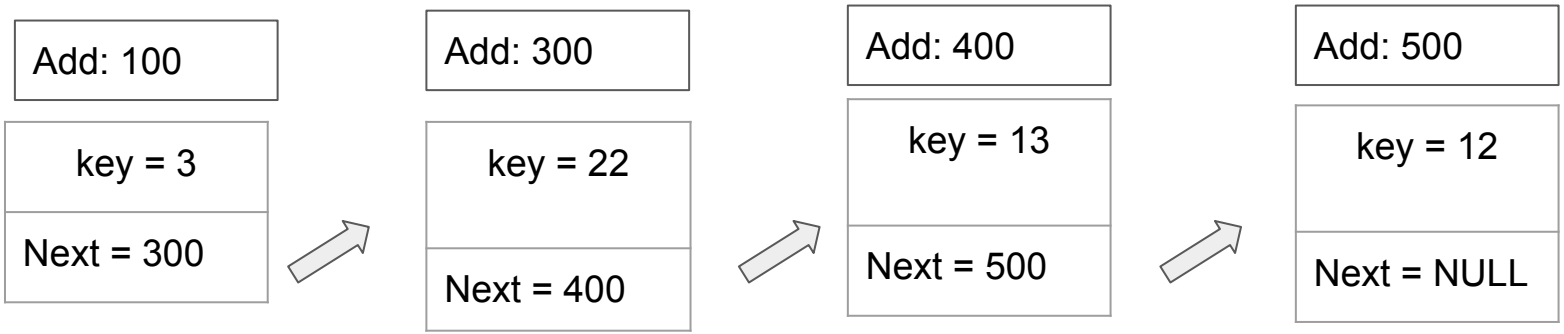
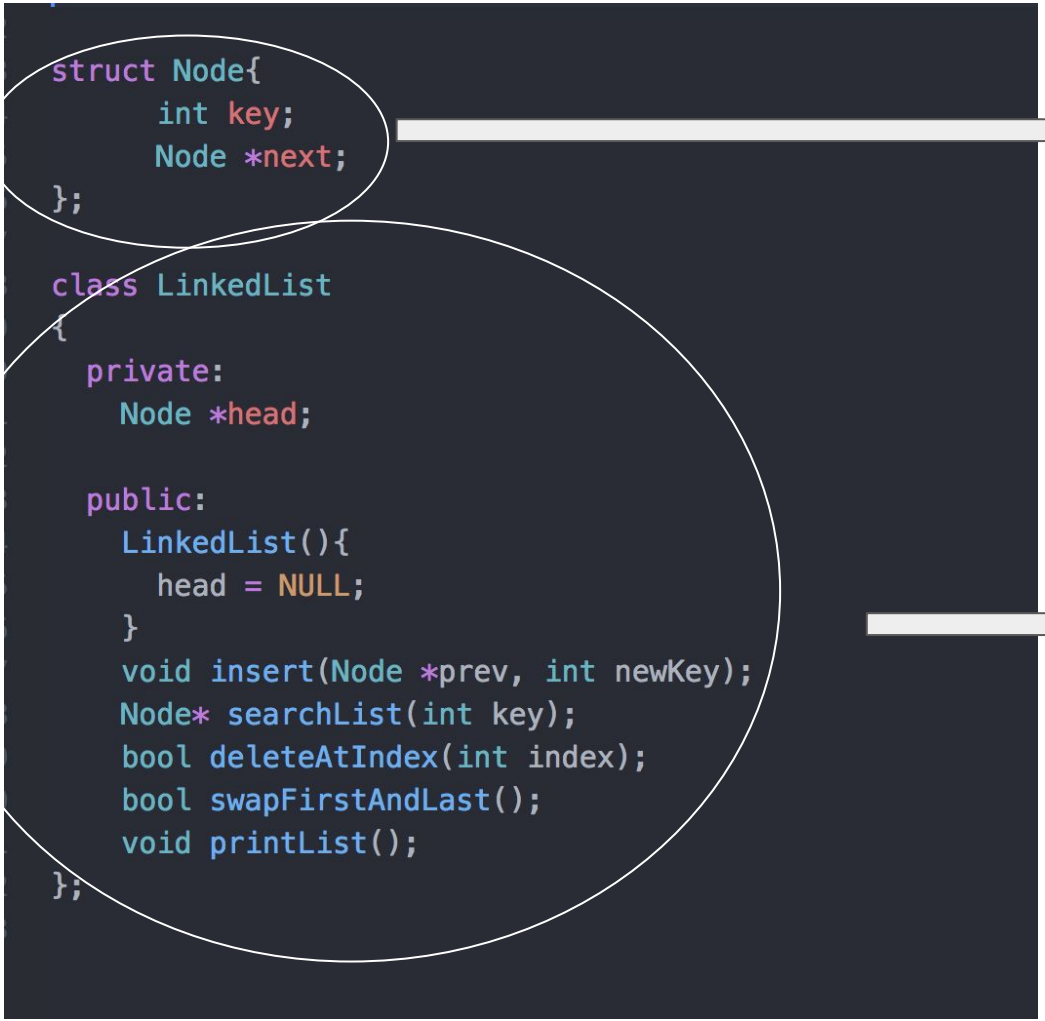
Node *head = 100



Arrays	Linked List
● Requires contiguous allocation of memory.	● Does not require contiguous memory allocation
● The allocated memory is equal to the upper limit irrespective of the usage.	● It is dynamic and flexible i.e. can expand and contract its size.
● Not useful if insertions and deletions are frequent.	● Useful if insertions and deletions are frequent.
● Allows random and sequential access.	● Allows sequential access.



Each node of the linked list is represented by this struct **‘Node’**

The class represents a **blueprint**.

It has functions and variables which can be considered as properties of it.

Since it is a blueprint, we can create ‘instances’ of it.

```cpp
#include <iostream>
#include "LinkedList.h"

using namespace std;

int main()
{
  LinkedList li;
  cout<<"Adding nodes to List:"<<endl;
  // 2
  li.insert(NULL,2);
  li.printList();
  // -1->2
  li.insert(NULL, -1);
  li.printList();
  // -1->2->-7
  li.insert(li.searchList(2),-7);
  li.printList();
  // -1->2->-7->10
```

An instance of LinkedList, **li.**

We are calling the functions **insert()** and **printList()** on this instance.

insert(NULL,2)- represents that the previous pointer is NULL and we are inserting a 2.

```
// Add a new node to the list
void LinkedList::insert(Node* prev, int newKey){
```

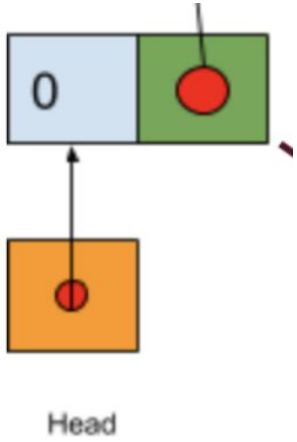
Arguments:

- A pointer to the node after which we insert the **newNode**.
- The key inside the **newNode**

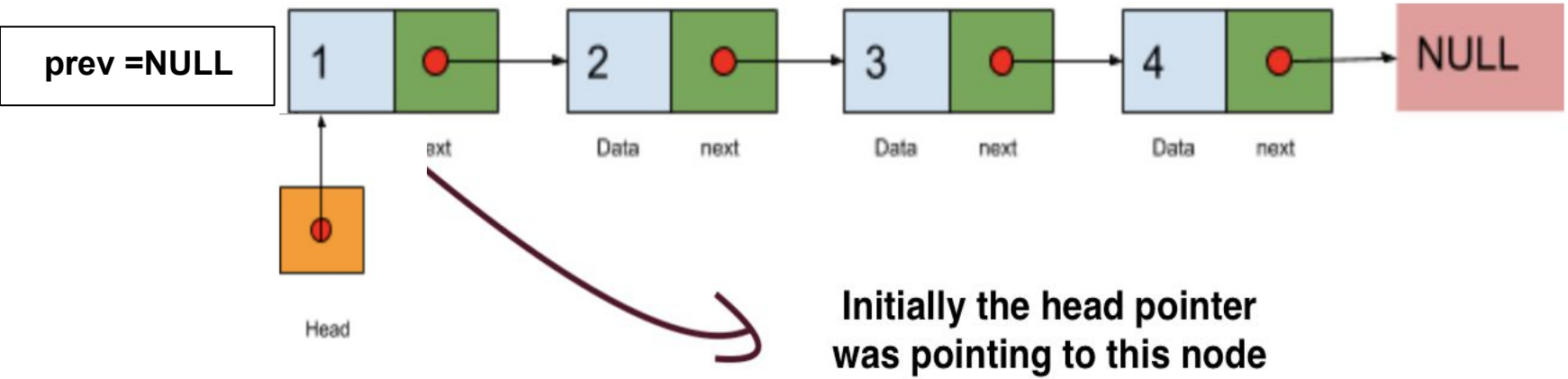
```
// Add a new node to the list
void LinkedList::insert(Node* prev, int newKey){

  //Check if head is Null i.e list is empty
  if(head == NULL){
    head = new Node;
    head->key = newKey;
    head->next = NULL;
  }

  // if list is not empty, look for prev and append our node there
  else if(prev == NULL)
  {
      Node* newNode = new Node;
      newNode->key = newKey;
      newNode->next = head;
      head = newNode;
  }

  else{

      //TODO

    }
  }
```

If head is NULL:

- It is an empty linked list.
- Create a new Node and make it the **head.**

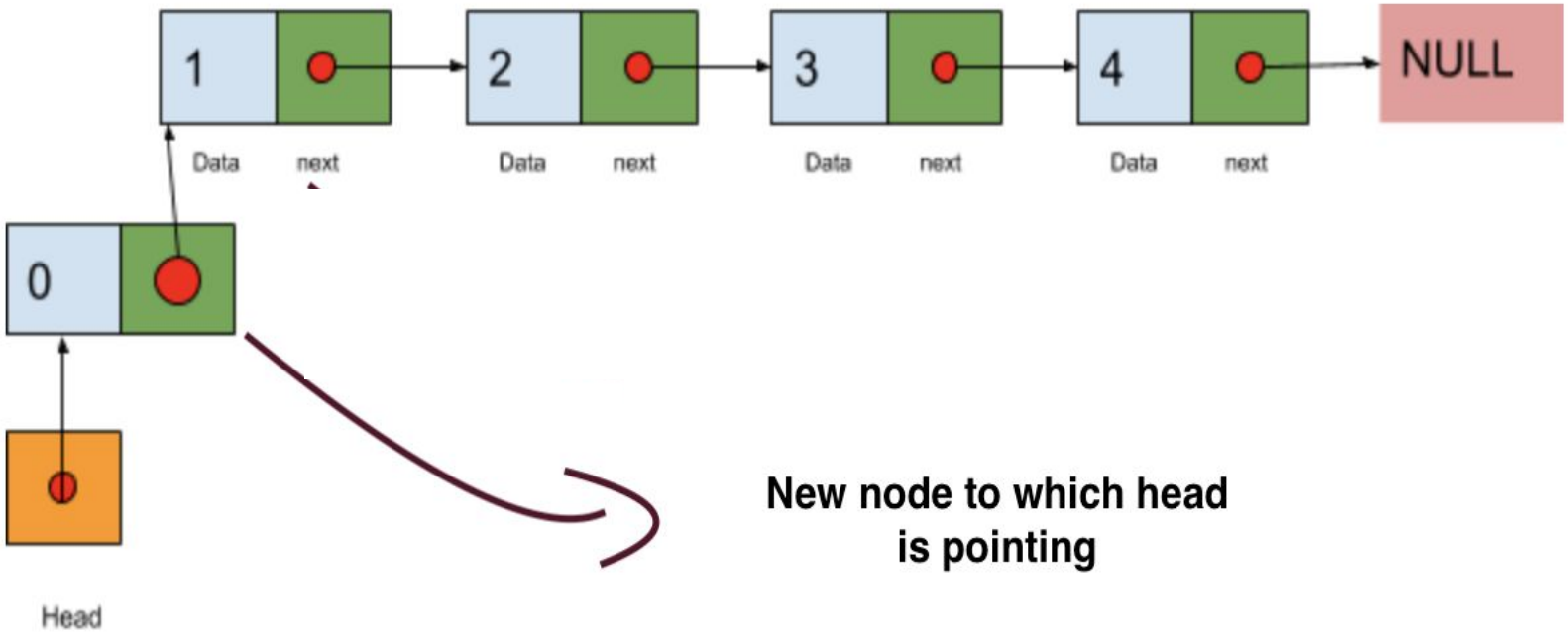


Insert node at the beginning(**prev** is NULL)



prev is **NULL** because there is no Node before the **head** pointer.

Insert node at the beginning(**prev** is NULL)



```
// Add a new node to the list
void LinkedList::insert(Node* prev, int newKey){

  //Check if head is Null i.e list is empty
  if(head == NULL){
    head = new Node;
    head->key = newKey;
    head->next = NULL;
  }

  // if list is not empty, look for prev and append our node there
  else if(prev == NULL)
  {
      Node* newNode = new Node;
      newNode->key = newKey;
      newNode->next = head;
      head = newNode;
  }

  else{

      //TODO

    }
  }
```

If prev is NULL and head is not NULL:

- Inserting before the current head.
- Create a new Node.
- Update its next to head.
- Make the new node as **head.**

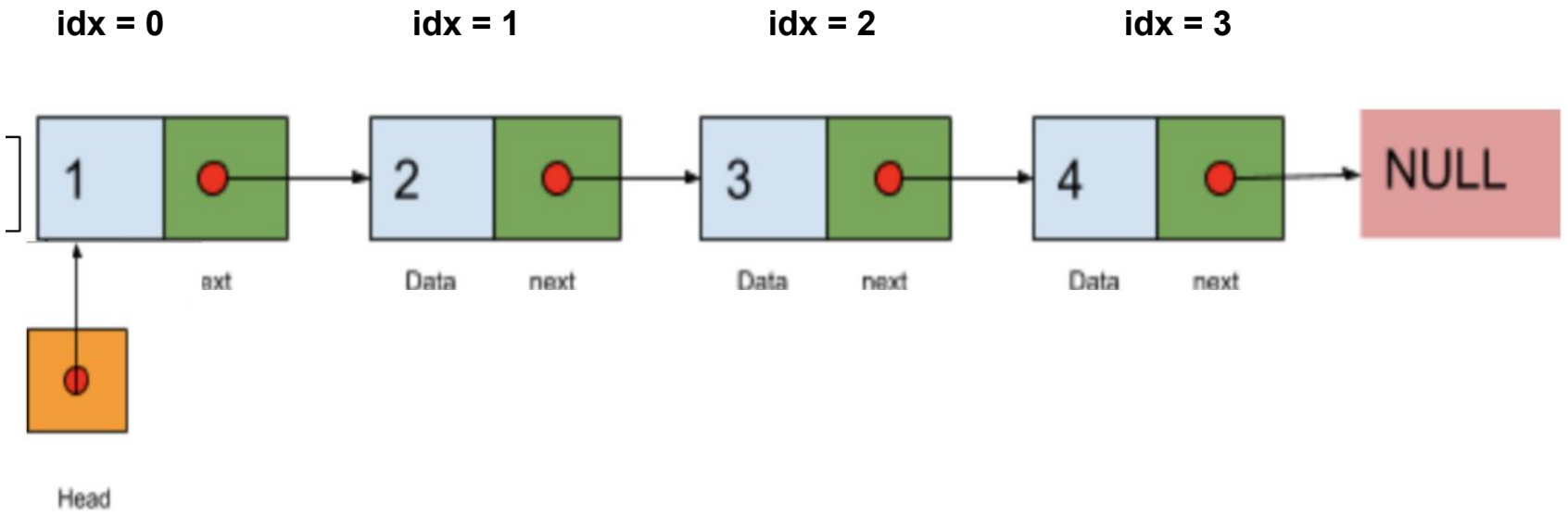
```
// Add a new node to the list
void LinkedList::insert(Node* prev, int newKey){

  //Check if head is Null i.e list is empty
  if(head == NULL){
    head = new Node;
    head->key = newKey;
    head->next = NULL;
  }

  // if list is not empty, look for prev and append our node there
  else if(prev == NULL)
  {
      Node* newNode = new Node;
      newNode->key = newKey;
      newNode->next = head;
      head = newNode;
  }

  else{

      //TODO

    }
  }
```

When prev is not NULL.

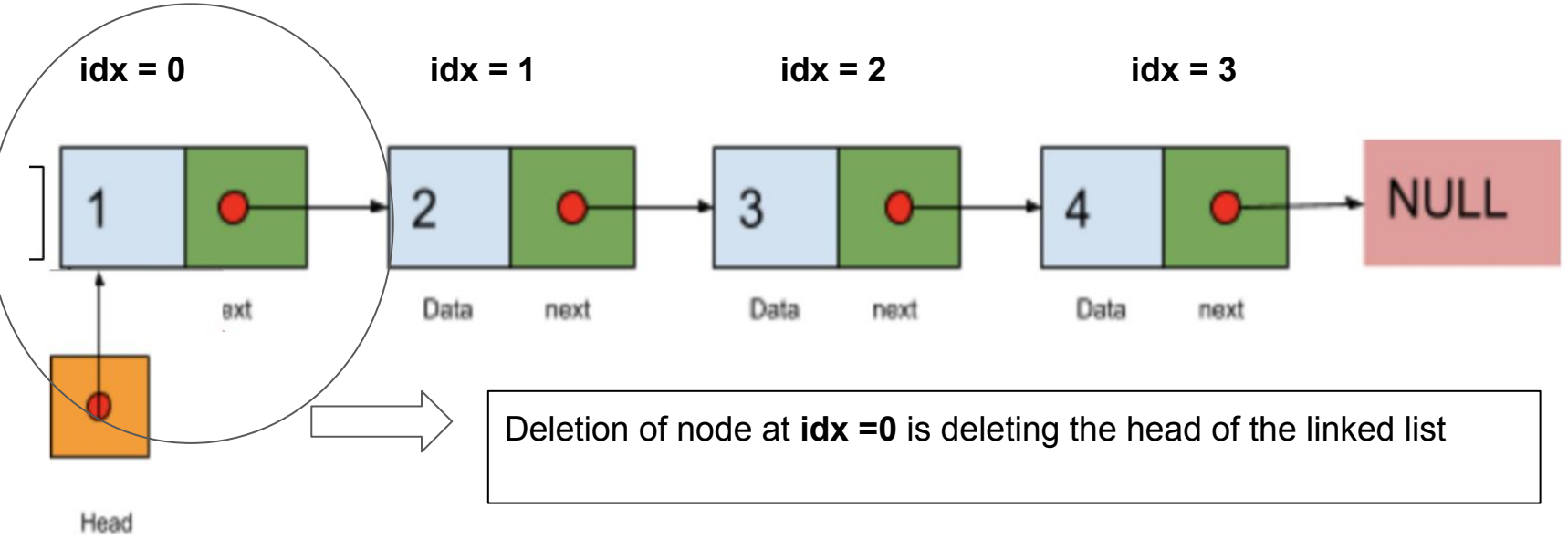
```
else{

    Node* newNode = new Node;
    newNode->key = newKey;
    newNode->next = prev->next;
    prev->next = newNode;

  }
}
```

- Allocate memory for the **newNode**
- Update the **key** in it
- Update the **next** pointers for **newNode** and **prev**.

Deletion of a node at an index in a linked list

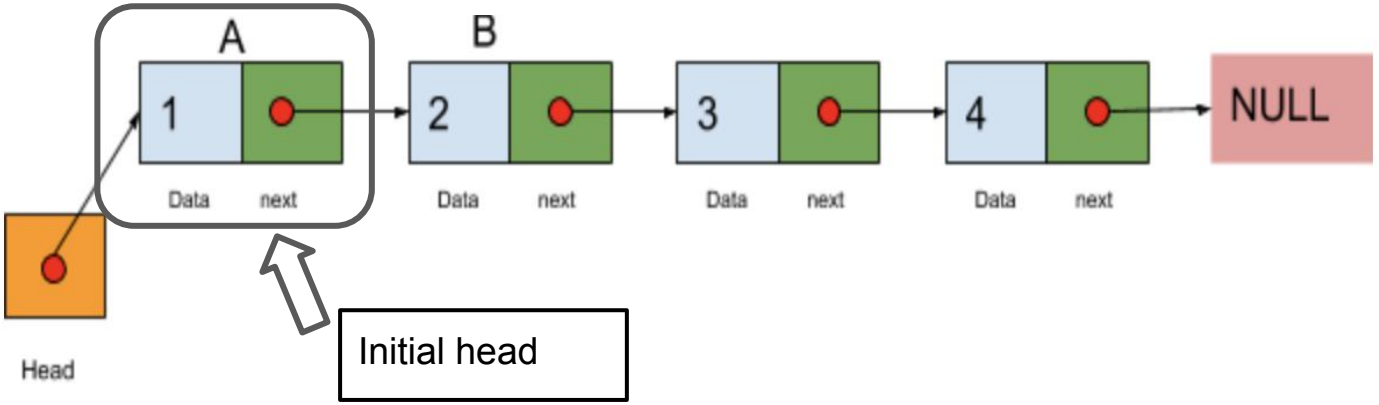


Deletion of a node at an index in a linked list

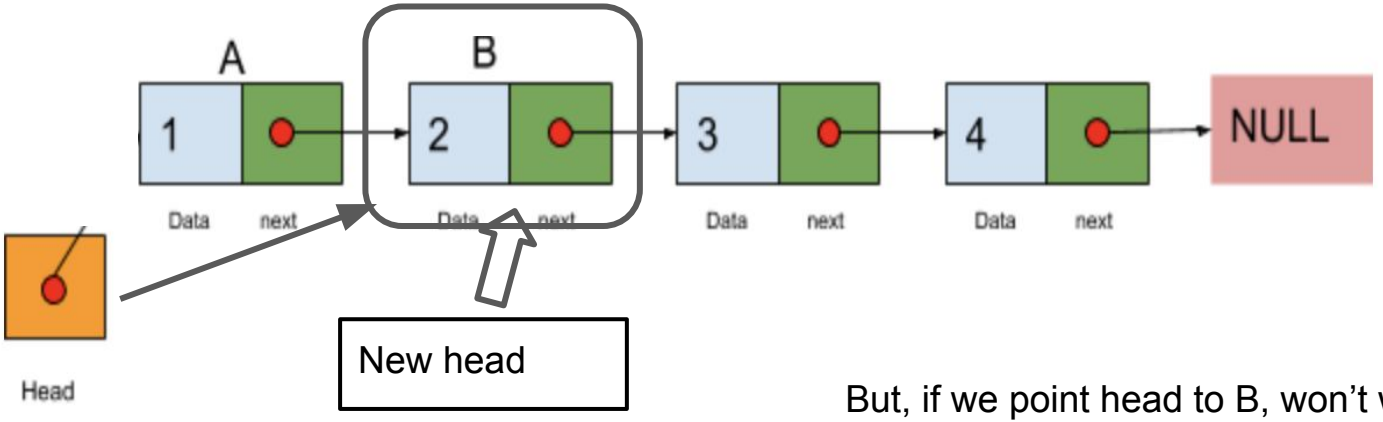


Deletion of node at **idx =0** is deleting the head of the linked list

Delete the head of a linked list



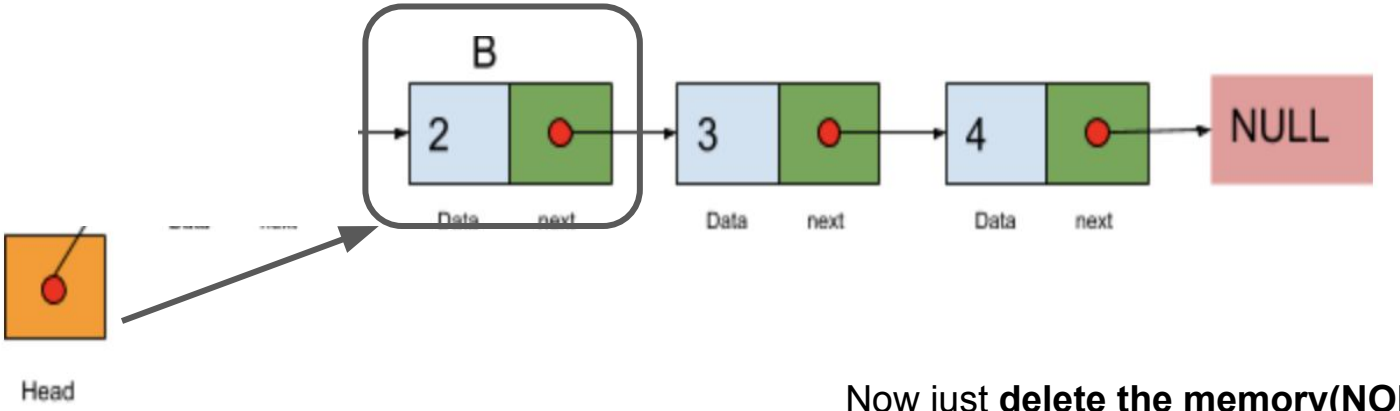
Delete the head of a linked list



But, if we point head to B, won't we lose A's address?

So we store it in a **temporary pointer variable**

Delete the head of a linked list



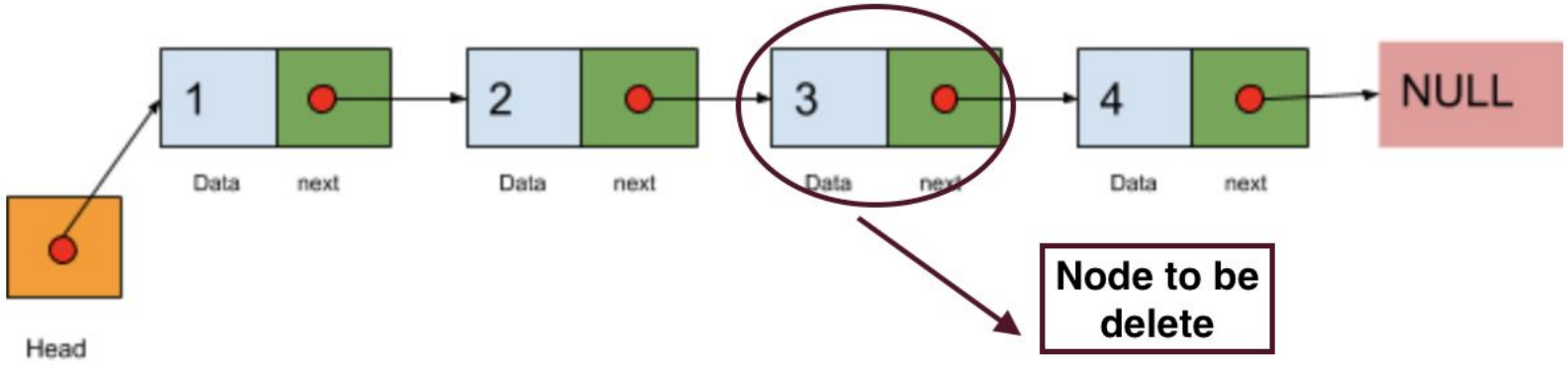
Now just **delete the memory(NODE A)** pointed by the **temp** Variable

Delete the head of a linked list

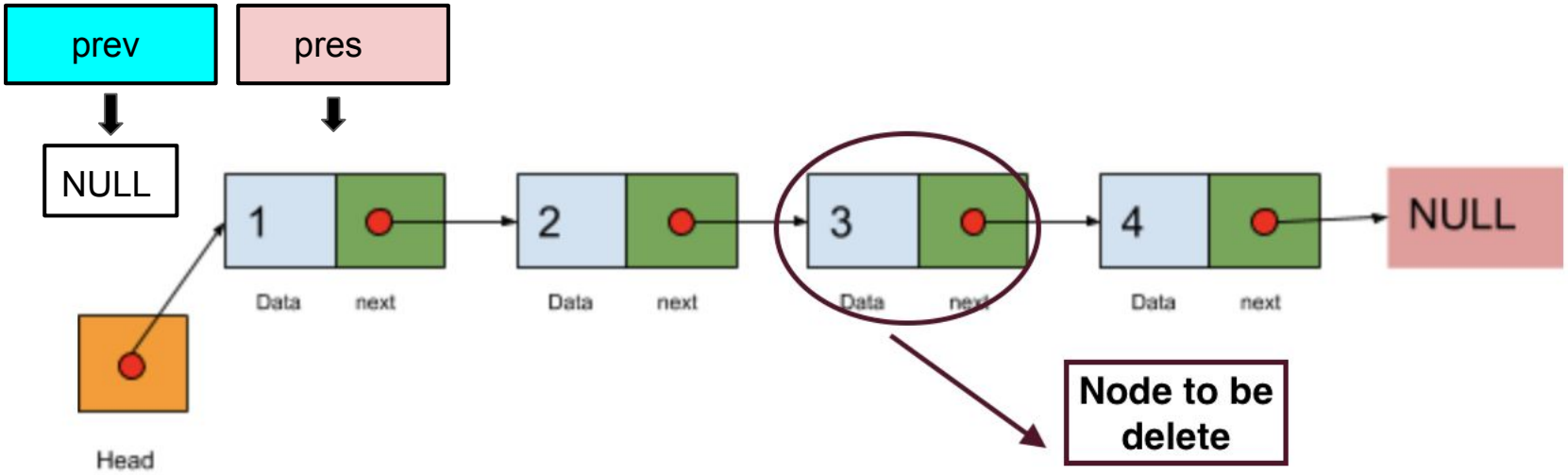
Steps:

- Store the **head pointer** in a **temp** variable -- **(Why?)**
- Update the **head pointer** to the second node.
- **Deallocate** memory for the **temp** variable

Delete node at any other index of the list

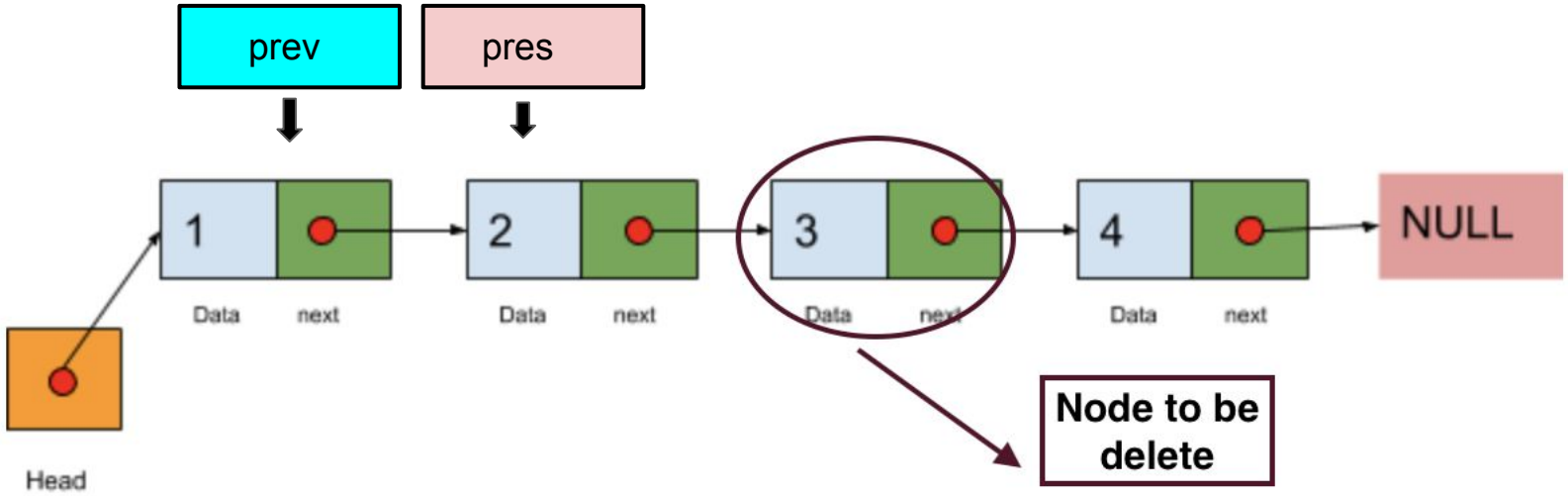


Delete node at any other index of the list (**idx** =2)

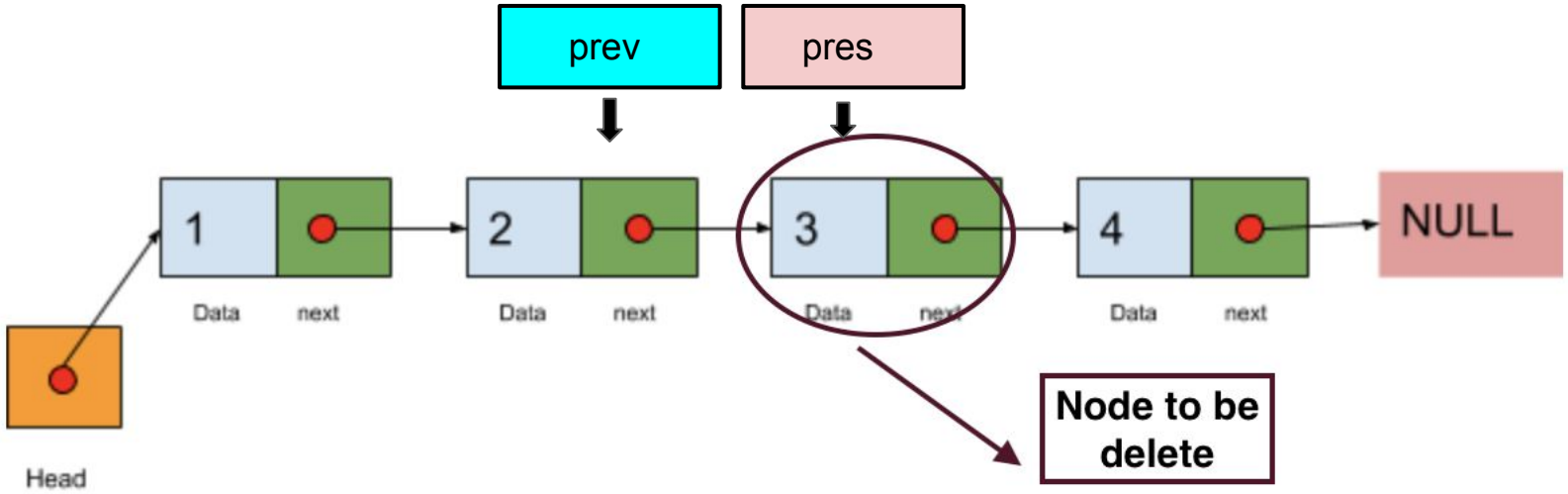


- Create two pointers, **prev** and **pres**.
- **pres** initialized to **head** and **prev** to NULL

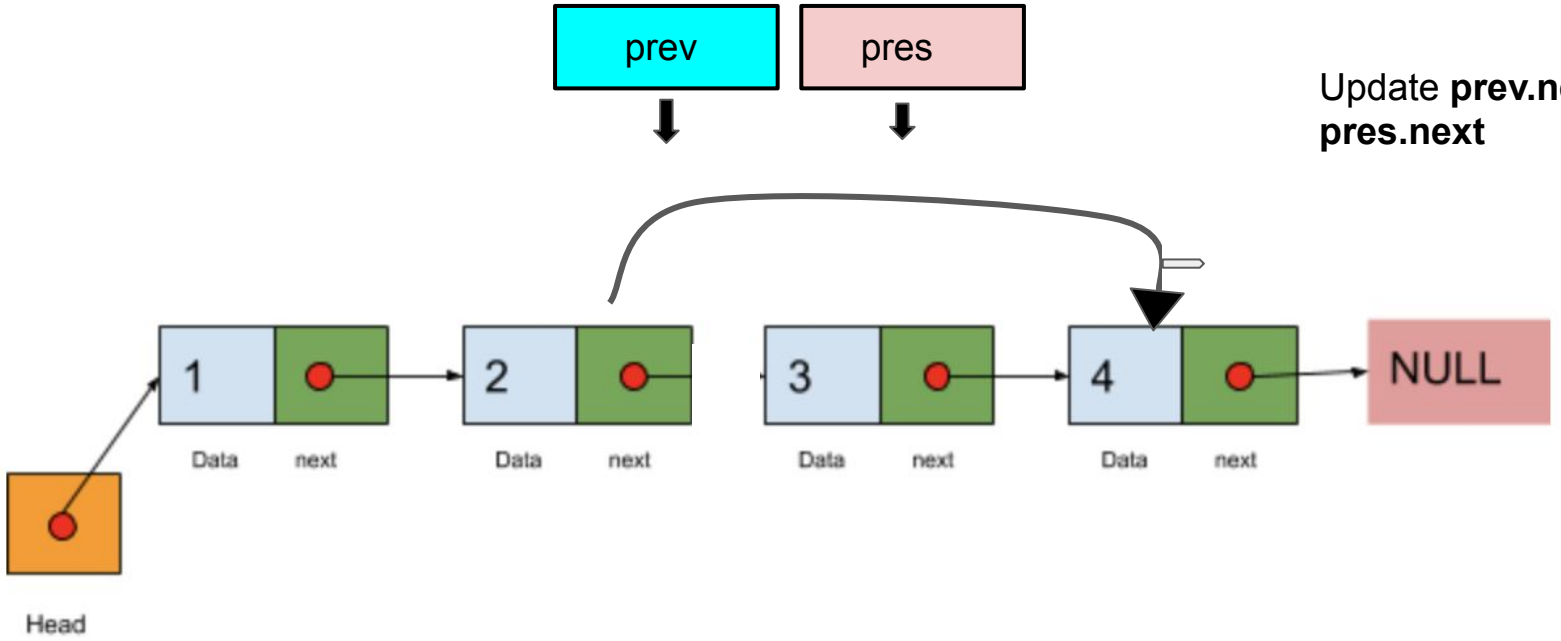
Delete node at any other index of the list (**idx** =2)



Delete node at any other index of the list (**idx** =2)

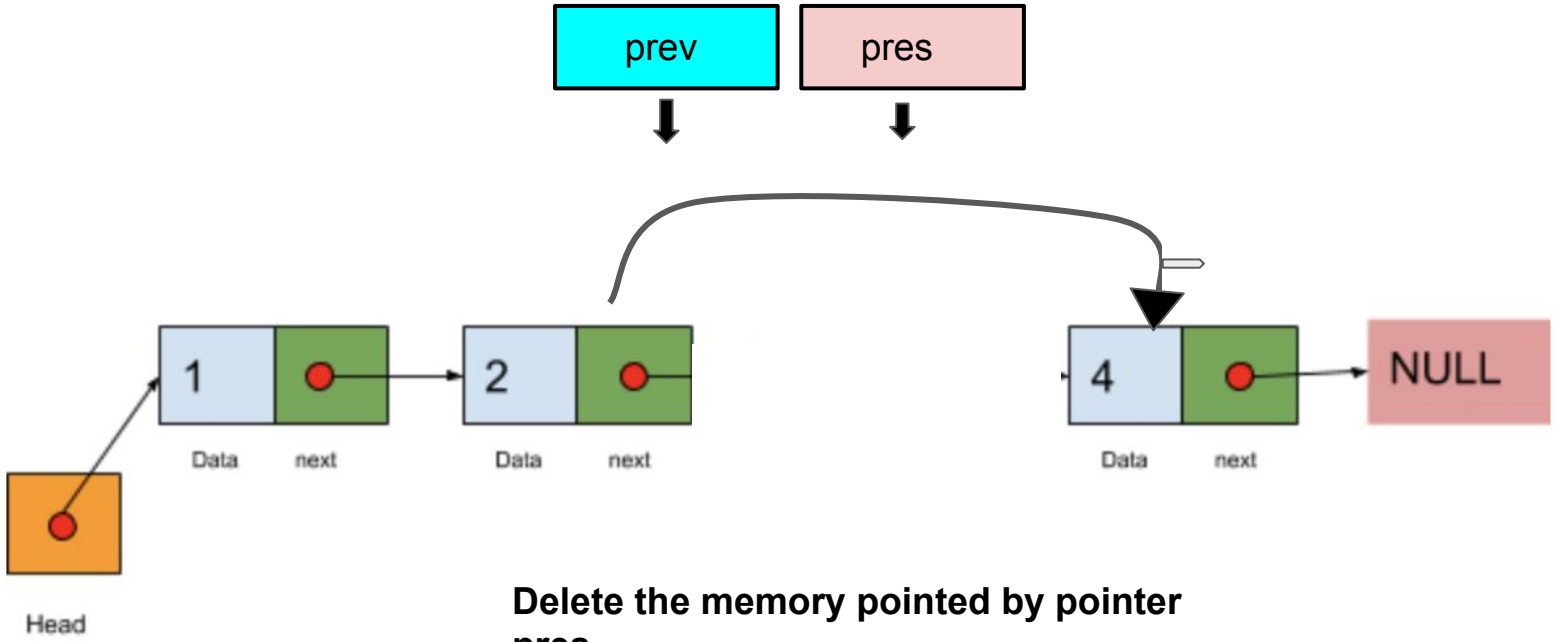


Delete node at an index of the list



Update **prev.next** to **pres.next**

Delete node at any other index of the list (**idx** =2)



Delete the memory pointed by pointer pres

Delete the node at any index,idx= n

Steps:

if n==0: **Deletion of head:**

- Handle deletion of the head of the Linked List.

if n != 0:

- Using two pointers **pres** and **prev**, reach the node at index n.
- Update the **prev.next** pointer to **pres.next**
- Deallocate memory for pres
- Update the **isDeleted** flag

```
void LinkedList::insert(Node* prev, int newKey){

        //Check if head is Null i.e list is empty
        if(head == NULL){
                head = new Node;
                head->key = newKey;
                head->next = NULL;
        }

        else if(prev == NULL)
        {
                Node* newNode = new Node;
                newNode->key = newKey;
                newNode->next = head;
                head = newNode;
        }

        else{
        //TO-DO



          }
}
```

If head is NULL:

- It is an empty linked list.
- Create a new Node and make it the **head.**

If prev is NULL and head is not NULL:

- Inserting before the current head.
- Create a new Node.
- Update its next to head.
- Make the new node as **head.**

When prev is not NULL.

- Allocate memory for the **newNode**
- Update the **key** in it
- Update the **next** pointers for **newNode** and **prev**.