Before we move on from linked lists - the doubly linked list

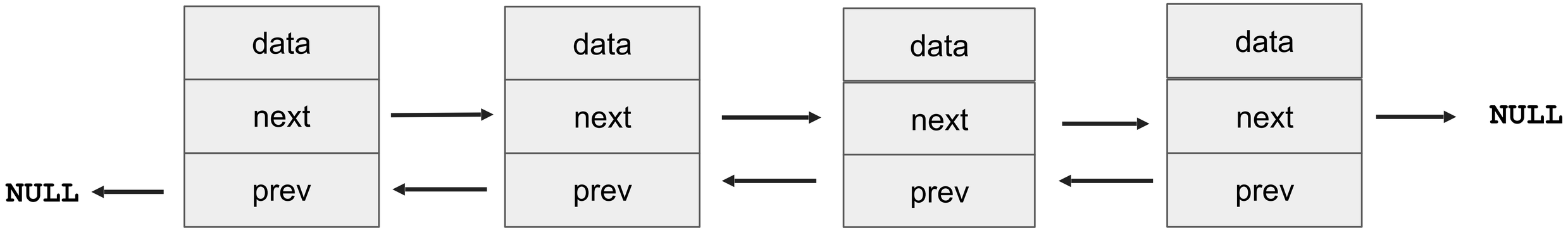
```
struct node {
  int data;
  struct node* next;
  struct node* prev
}
```

A doubly linked list has pointers to the next element as well as pointers to the previous element.

This makes going back and forth in the list far easier, a lot of the tricky stuff involved in singly linked lists is eliminated

This requires additional memory in every node to store the previous pointer

HOW DO DOUBLY LINKED LISTS LOOK IN MEMORY?



NOTE THAT THE PREV POINTER OF THE FIRST ELEMENT AND THE NEXT POINTER OF THE LAST ELEMENT POINTS TO NULL

DELETE NODE IN DOUBLY LINKED LIST

```
void delete_node(struct node** headRef, int data) {
  assert(headRef != NULL);

  struct node* head = *headRef;
  while (head != NULL && head->data != data) {
    head = head->next;
  }

  if (head == NULL) {
    return;
  }

  // head will point to the node which should be deleted.
  if (head->prev != NULL) {
    head->prev->next = head->next;
  } else {
    *headRef = head->next;
  }

  if (head->next != NULL) {
    head->next->prev = head->prev;
  }

  head->prev = NULL;
  head->next = NULL;
  free(head);
}
```

WALK THE LIST TO FIND A NODE WITH DATA WHICH MATCHES THE INPUT DATA

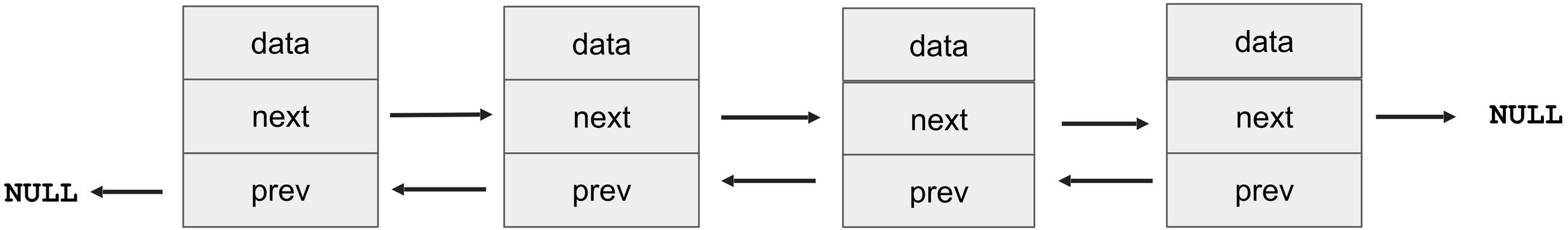
IF NO NODE IS FOUND THEN WE JUST RETURN, THERE IS NO NODE TO DELETE

HEAD CURRENTLY POINTS TO THE NODE WHICH HAS TO BE DELETED, IF THERE IS A PREVIOUS ELEMENT, IT NEEDS TO BYPASS THE CURRENT, ADJUST THE POINTERS ACCORDINGLY

IF THE PREV POINTER IS NULL THEN THIS IS THE FIRST ELEMENT OF THE LIST, ADJUST THE HEAD ACCORDINGLY

IF THERE IS A NEXT ELEMENT ADJUST ITS PREVIOUS POINTER

DELETING A NODE IN A DOUBLY LINKED LIST



DELETING A NODE IN A DOUBLY LINKED LIST