COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 13

Time to delete from a linked list

LAST TIME...

- Multifile projects
- Linked Lists -
  - creating a list
  - inserting nodes at the head
  - traversing a list
  - inserting nodes at the tail
  - inserting a node at position

TODAY...

- Linked Lists -
  - deleting nodes in a list
    - at the head
    - at the tail
    - in the middle
    - with only one item in a list

“

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/25T1/CODE/WEEK_8

LINKED LISTS

DELETING

- Where can I delete in a linked list?
    - Nowhere (if it is an empty list - edge case!)
    - At the head (deleting the head of the list)
    - Between any two nodes that exist
    - At the tail (last node of the list)

LINKED LISTS

DELETING EMPTY LIST

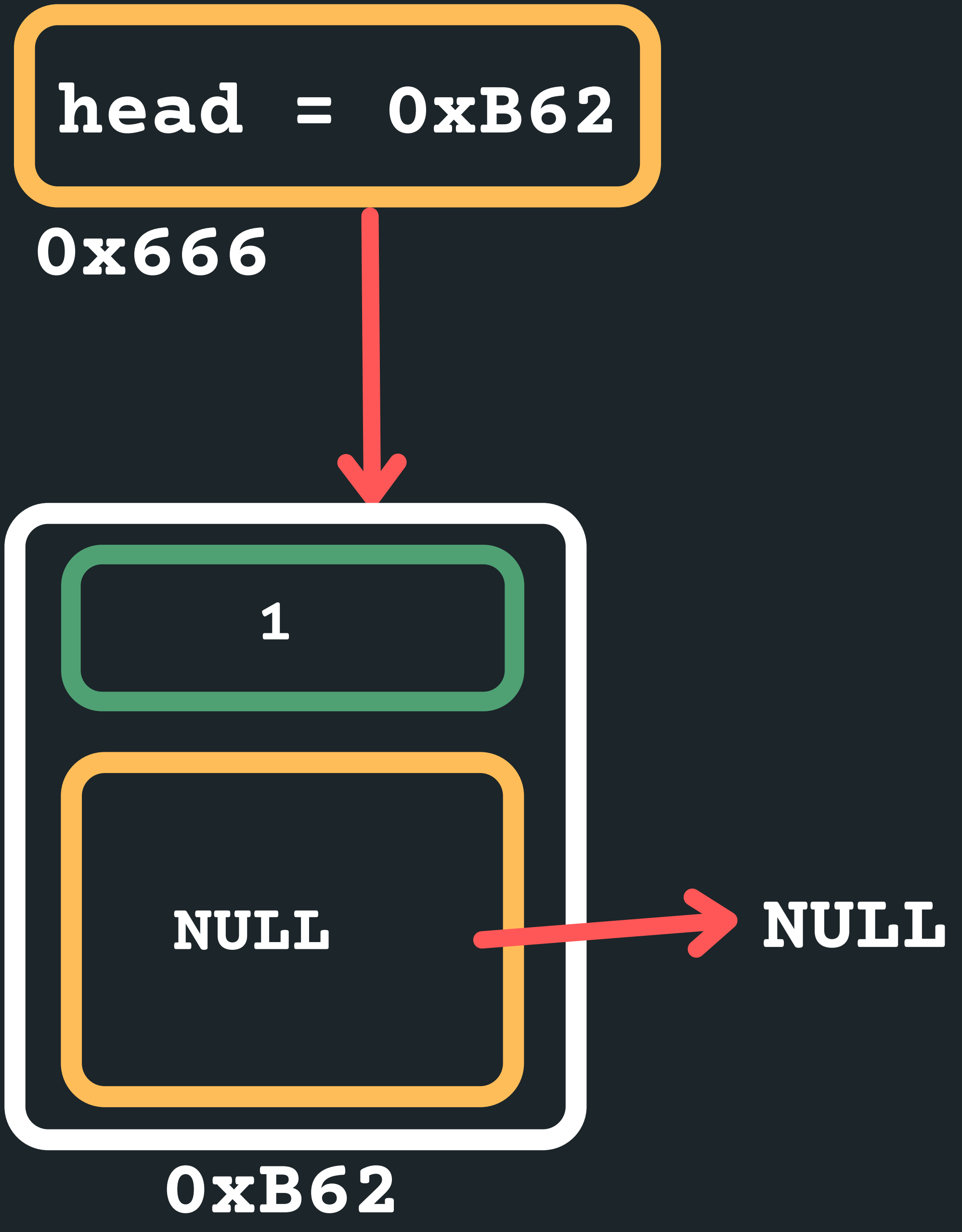
- Deleting when nowhere! (it is an empty list)
    - Check if list is empty
    - If it is - return NULL

```
struct node *current = head;
if (current == NULL){
    return NULL;
}
```

LINKED LISTS

DELETING ONE ITEM

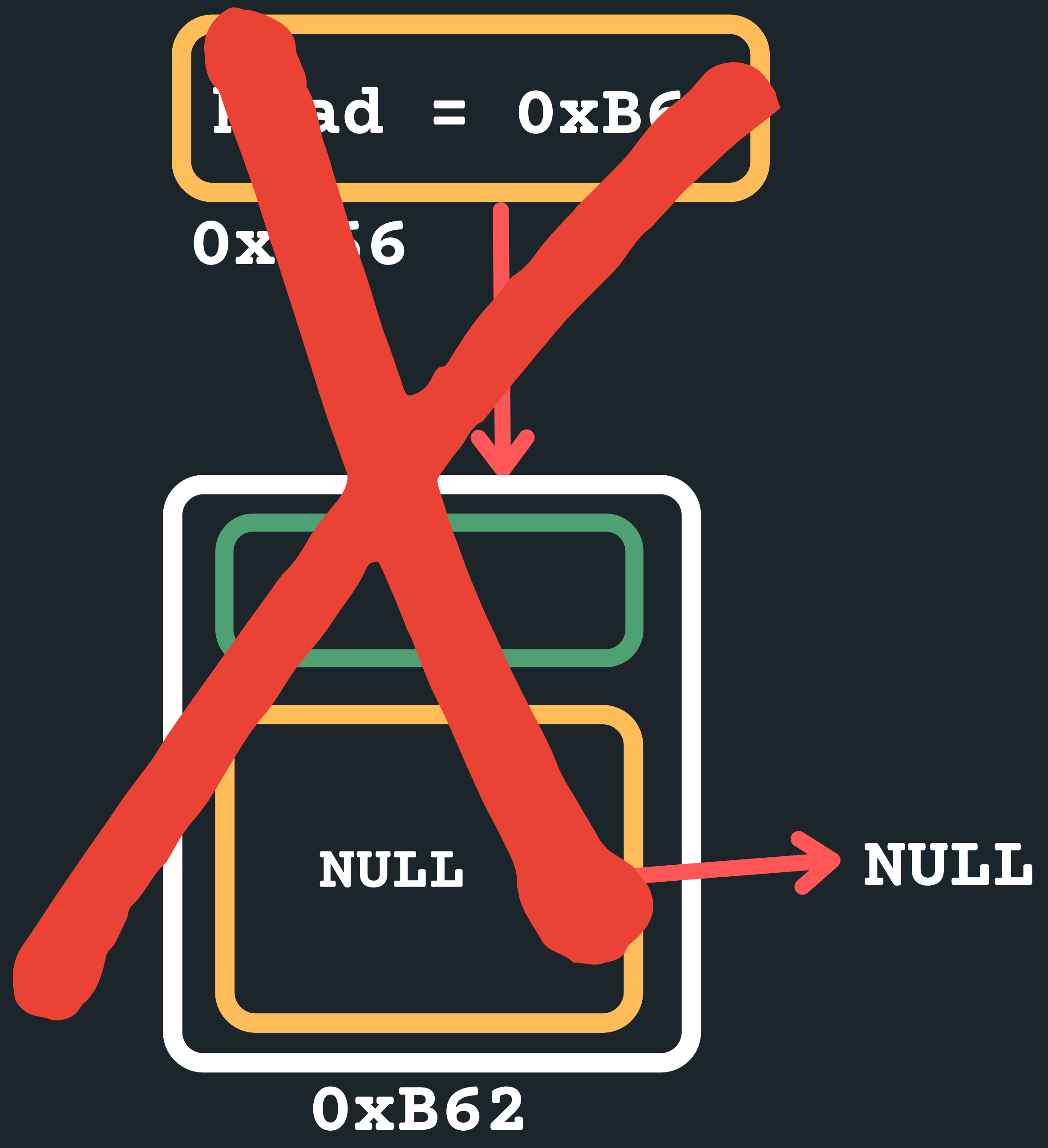
- Deleting when there is only one item in the list



LINKED LISTS

DELETING ONE ITEM

- Deleting when there is only one item in the list
    - free the head!

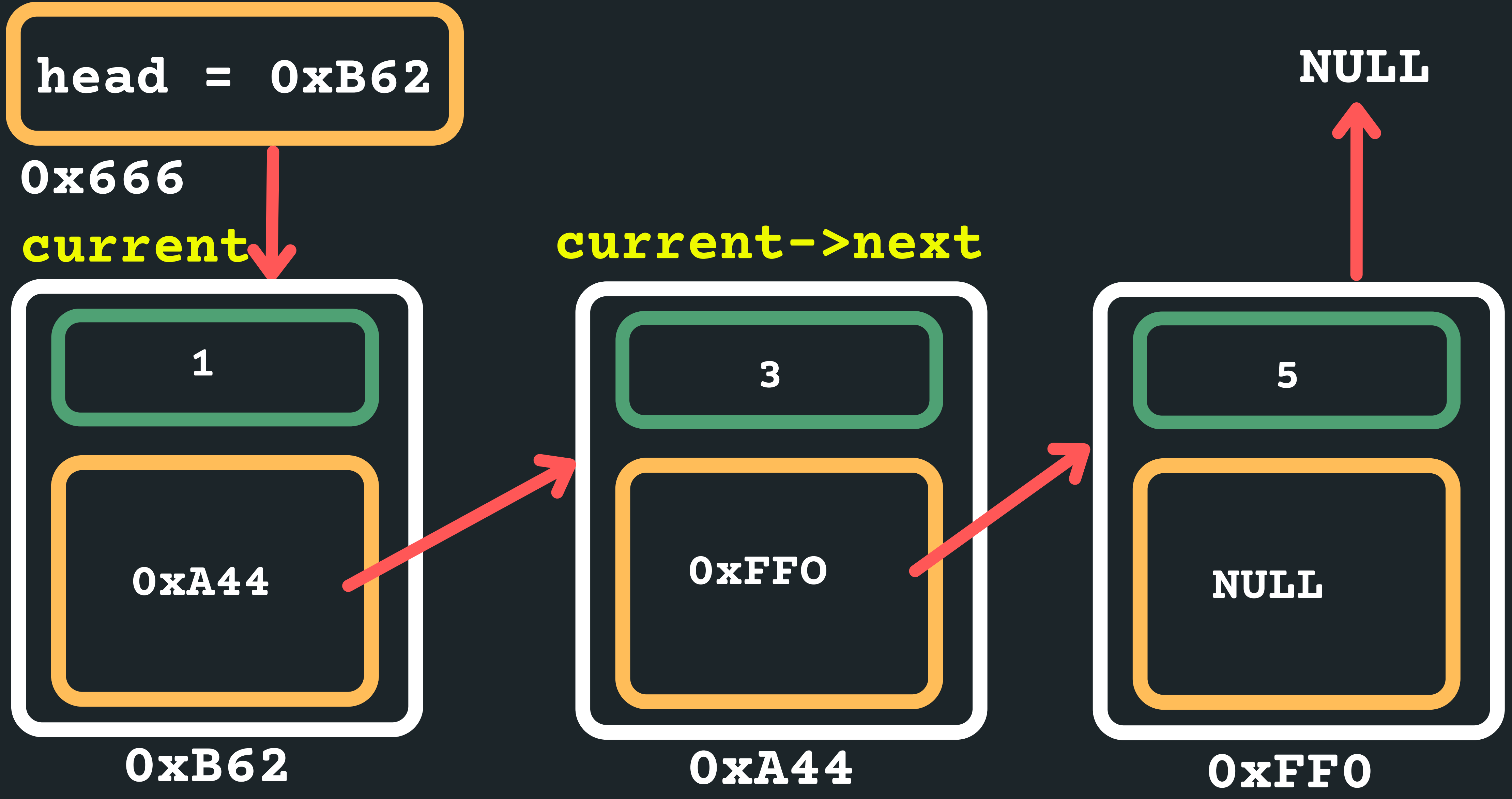


LINKED LISTS

DELETING THE HEAD WITH OTHER ITEMS

- Deleting when at the head of the list with other items in the list
    - Find the node that you want to delete (the head)

```
struct node *current = head
```

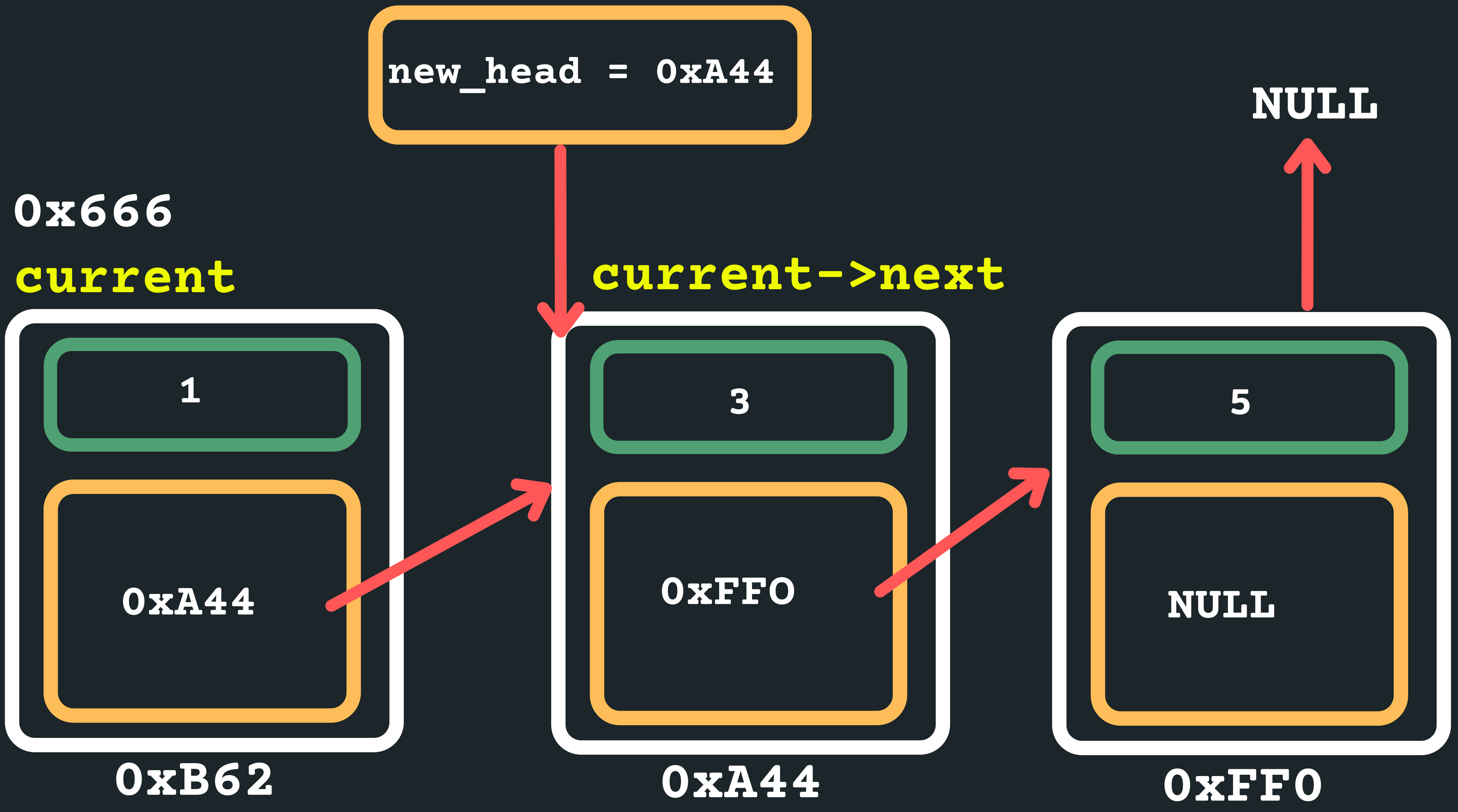


LINKED LISTS

DELETING THE HEAD WITH OTHER ITEMS

- Deleting when at the head of the list with other items in the list
    - Point the head to the next node

```
struct node *new_head = current->next;
```

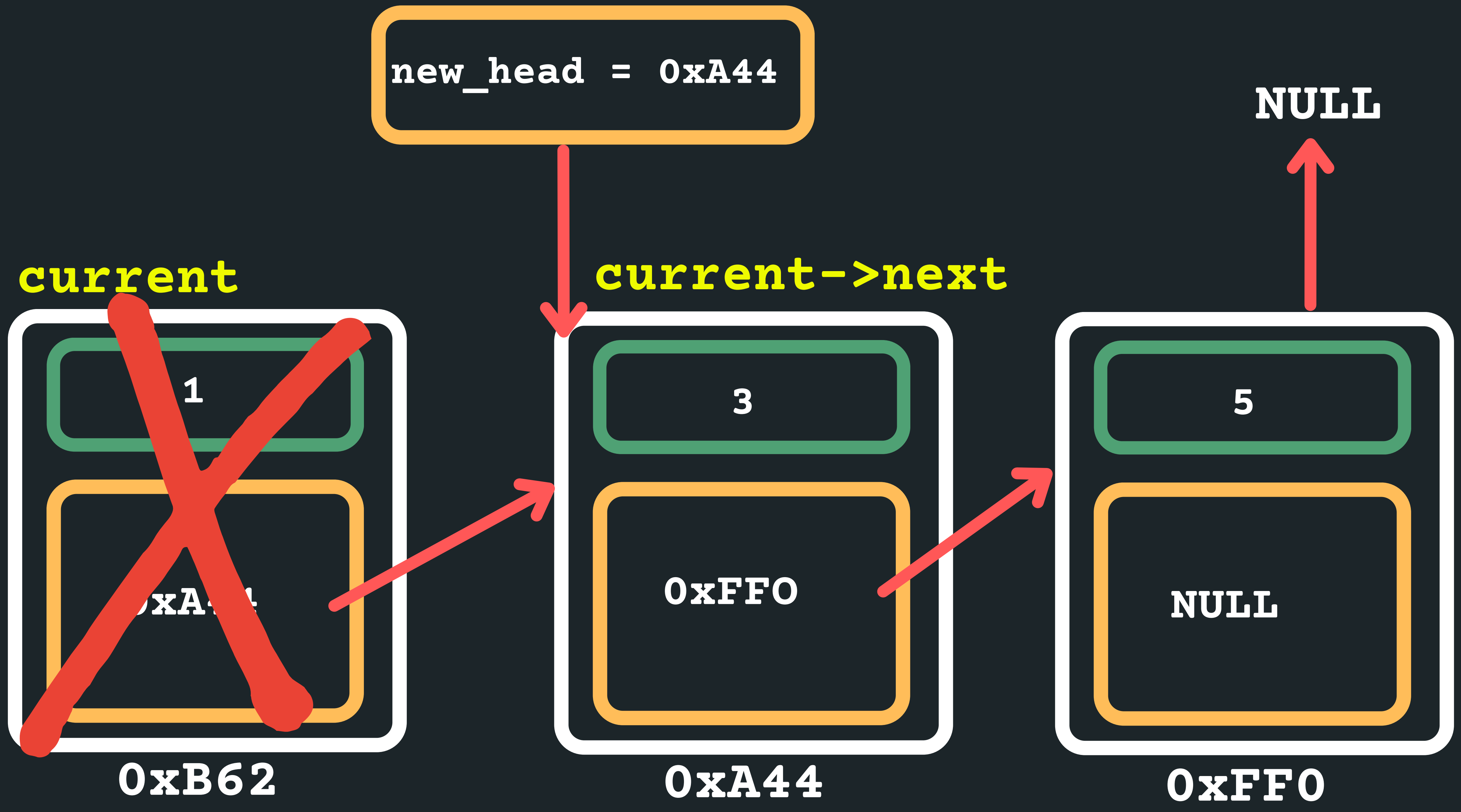


LINKED LISTS

DELETING THE HEAD WITH OTHER ITEMS

- Deleting when at the head of the list with other items in the list
    - Delete the current head

```
free(current);
```

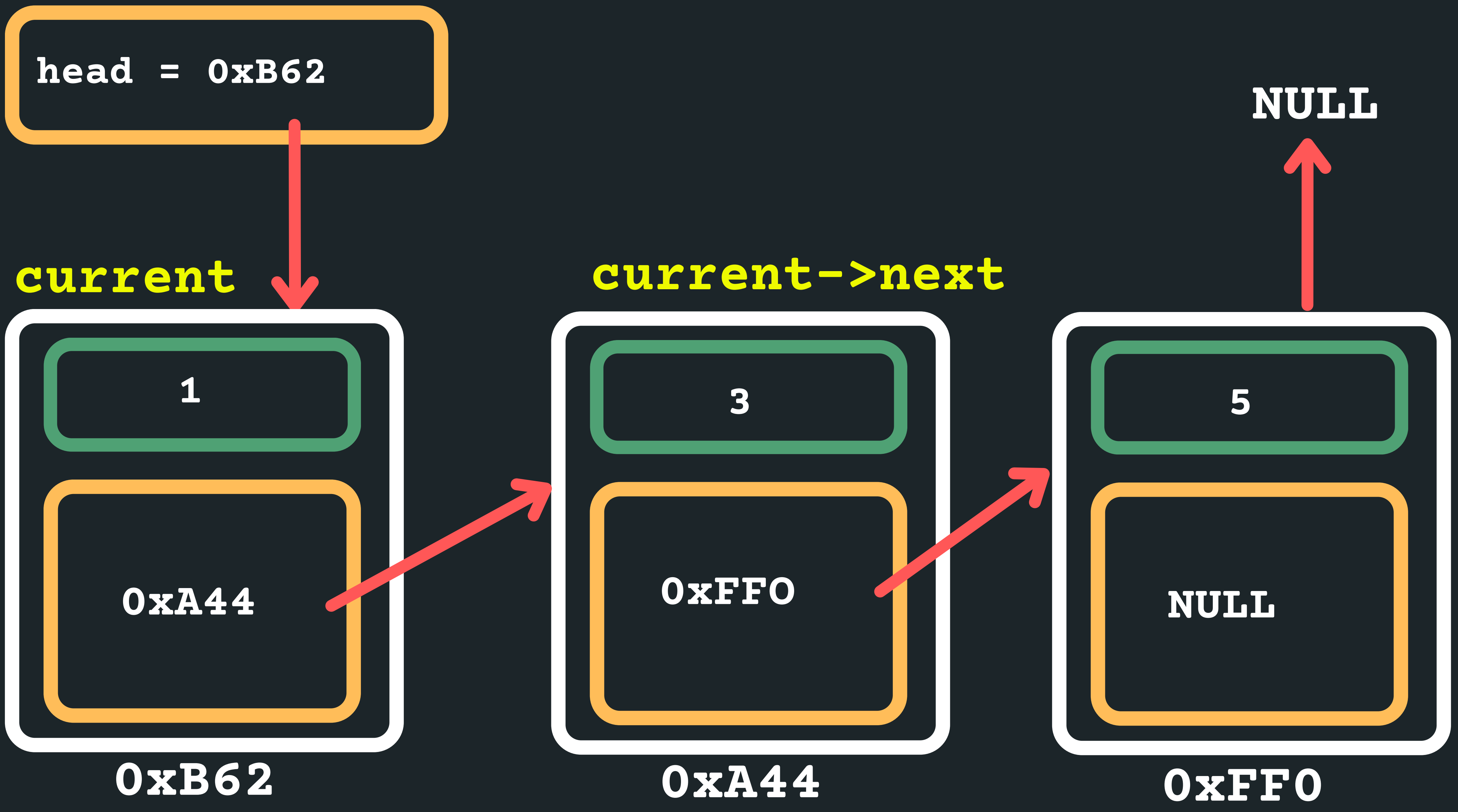


LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
    - Set the head to a variable current to keep track of the loop

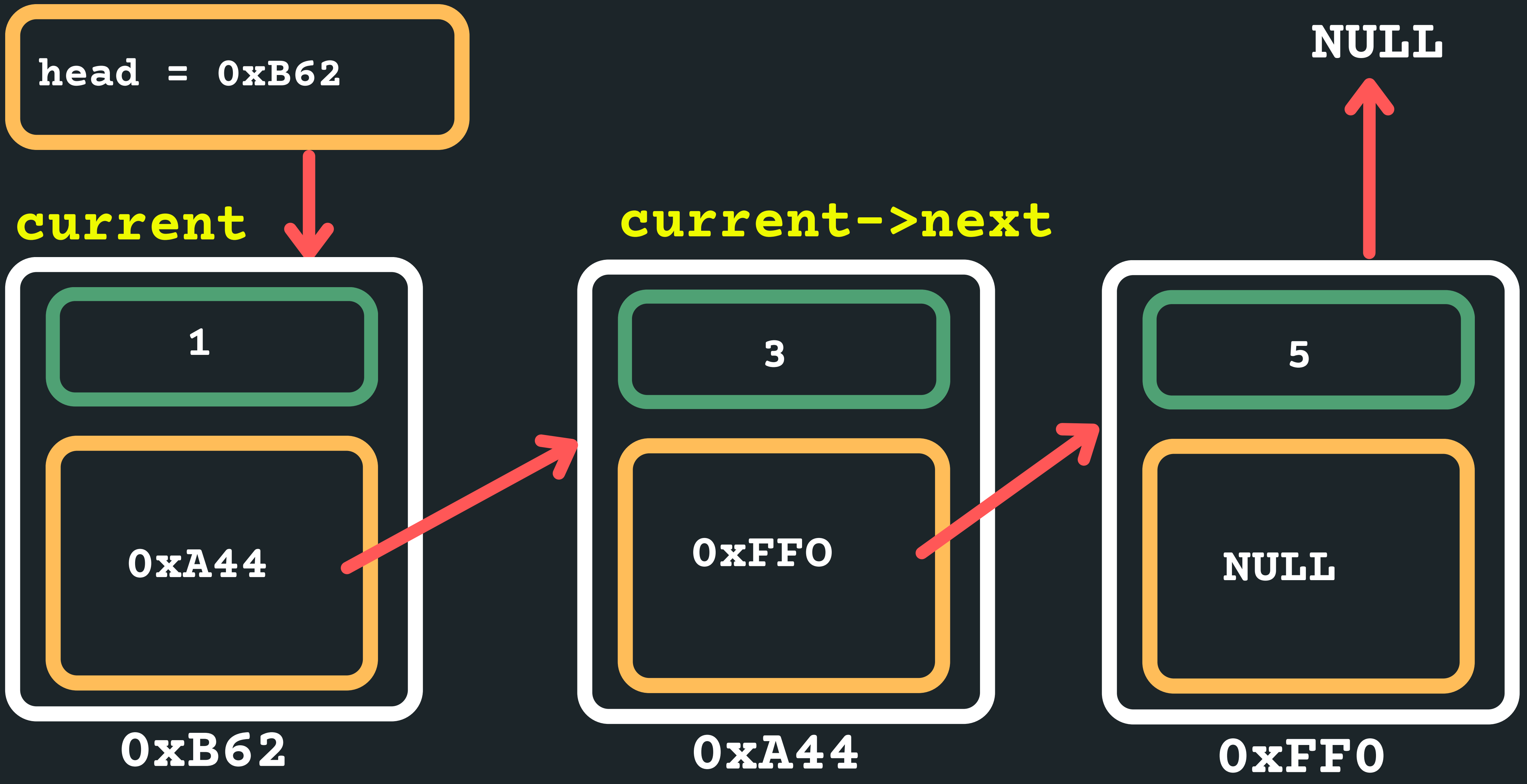
```
struct node *current = head
```



LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
  - Loop until you find the right node - what do we think loop until the node with 3 or the previous node? Remember that once you are on the node with 3, you have no idea what previous node was.

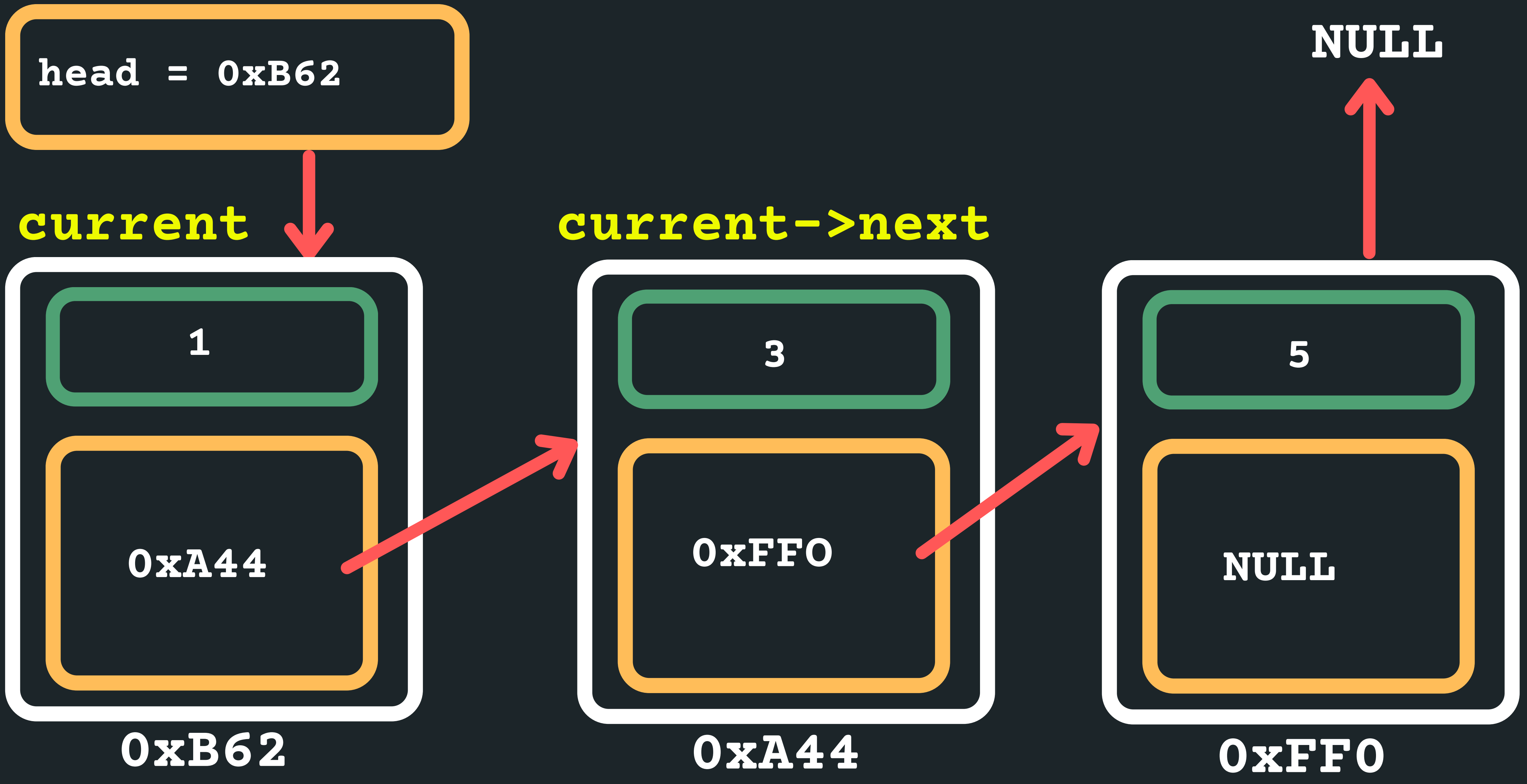


LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
    - So stop at a previous node (when the next is = 3)

```
while (current->next->data != 3){
    current = current->next;
}
```

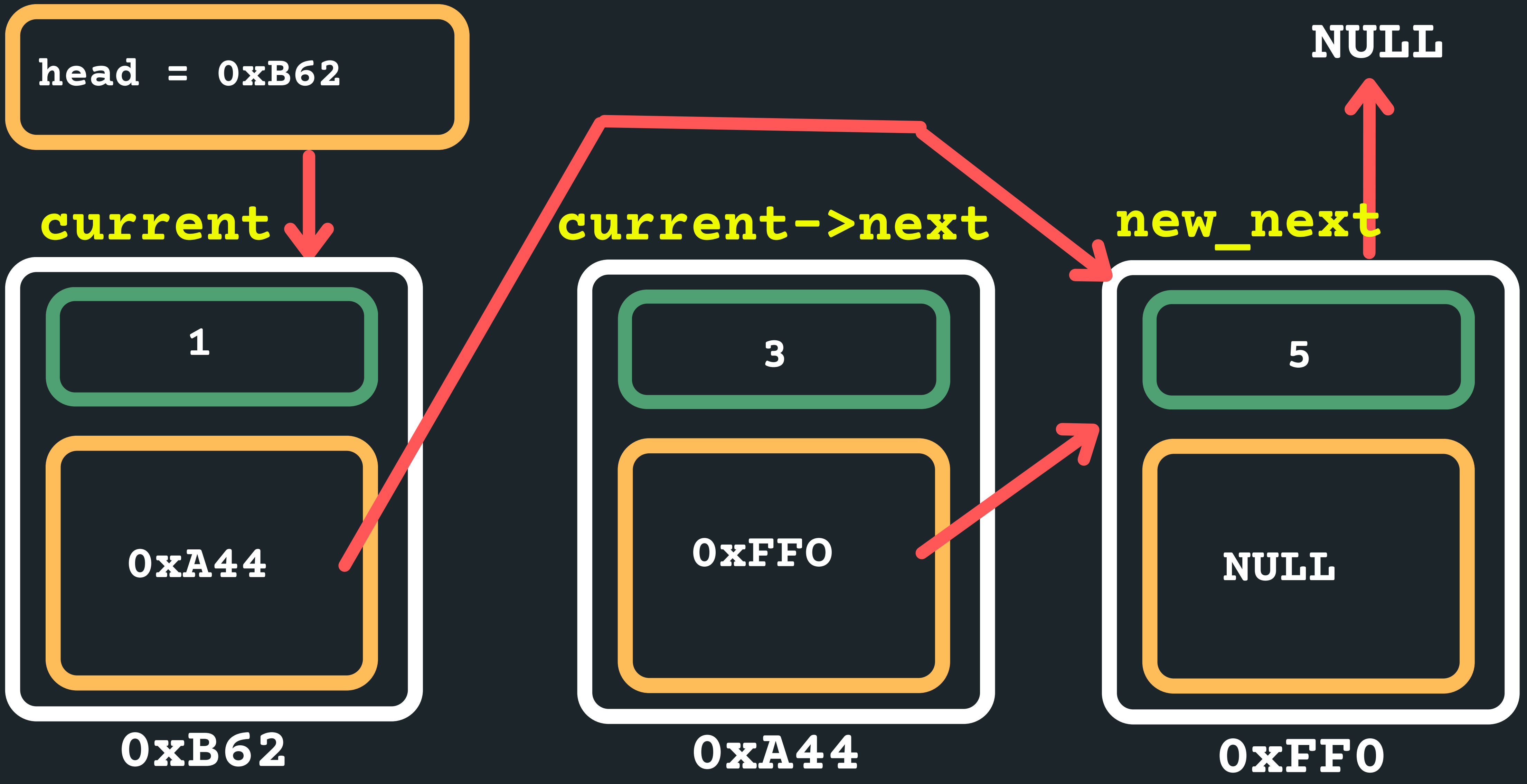


LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
    - Create new next node to store address

```
struct node *new_next = current->next->next;
```

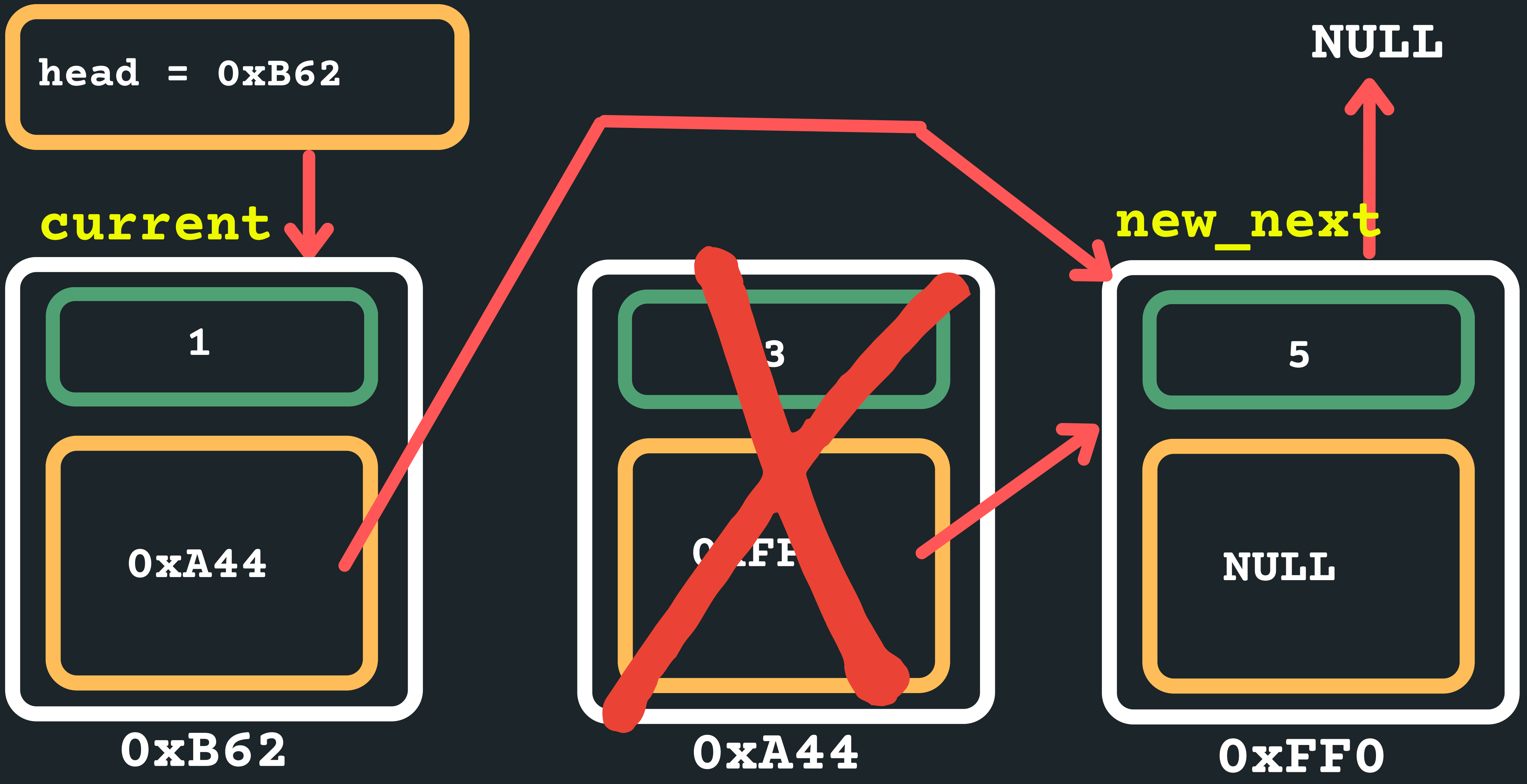


LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
    - Delete current->next

```
free(current->next);
```

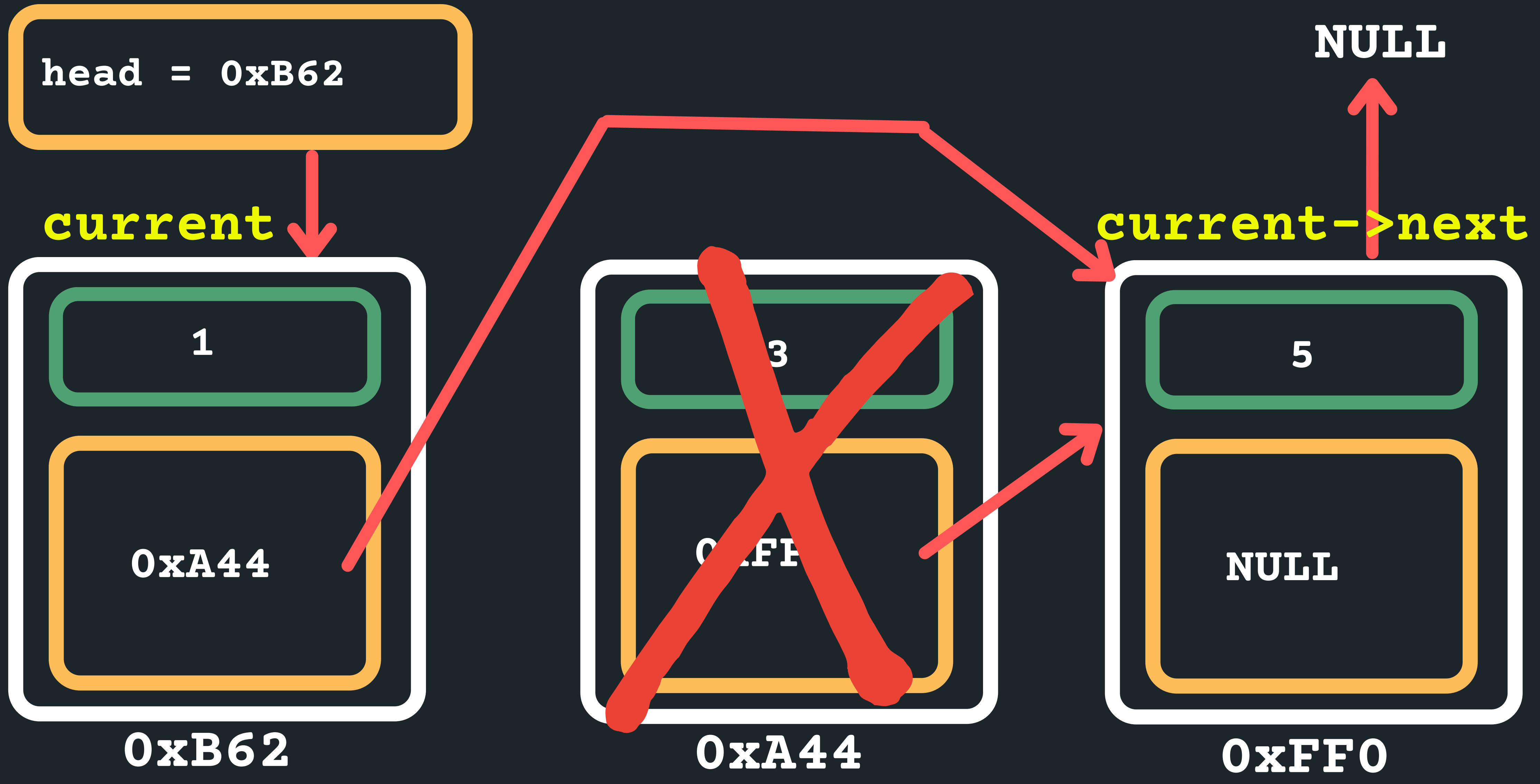


LINKED LISTS

DELETING IN MIDDLE OF TWO NODES

- Deleting when in the middle of two nodes (for example, node with 3)
  - Set the new current->next to the new_next node

```
current->next = new_next;
```

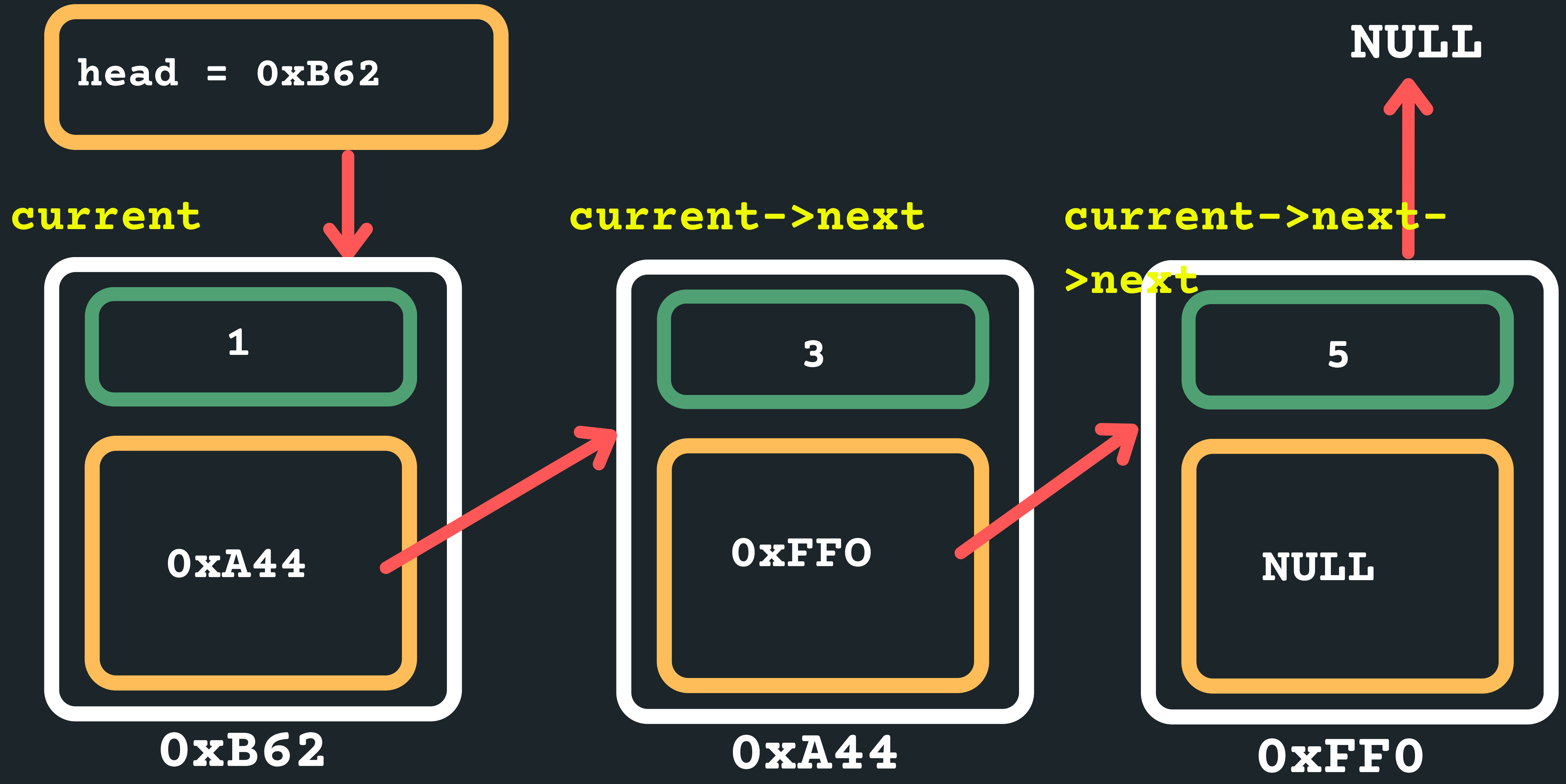


LINKED LISTS

DELETING THE TAIL

- Deleting when in the tail
    - Set the current pointer to the head of the list

```
struct node *current = head
```

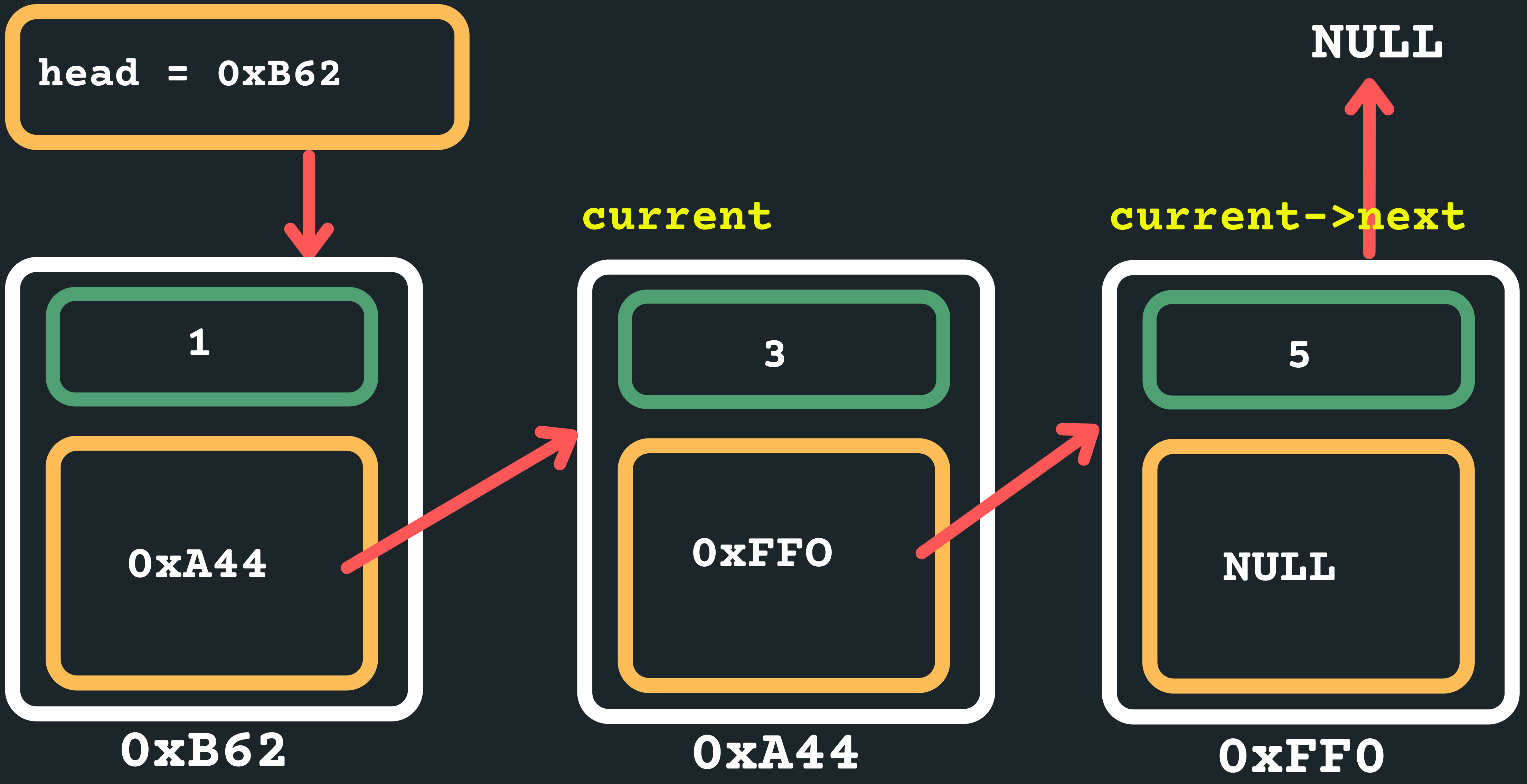


LINKED LISTS

DELETING THE TAIL

- Deleting when in the tail
    - Find the tail of the list (should I stop on the tail or before the tail?)
    - If the next is NULL than I am at the tail...

```
while (current->next->next != NULL){
    current = current->next;
}
```

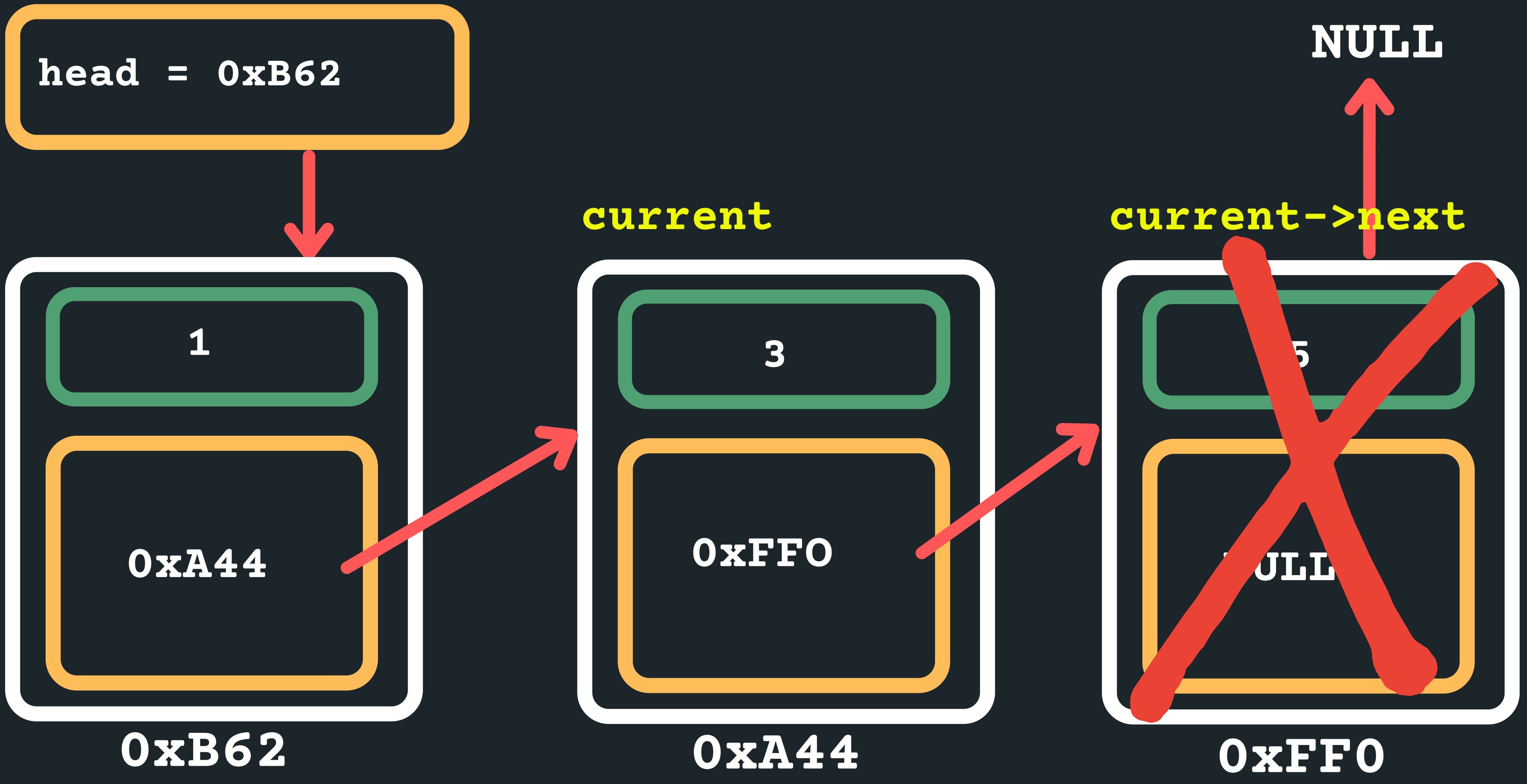


LINKED LISTS

DELETING THE TAIL

- Deleting when in the tail
  - Delete the current->next node

```
free(current->next);
```

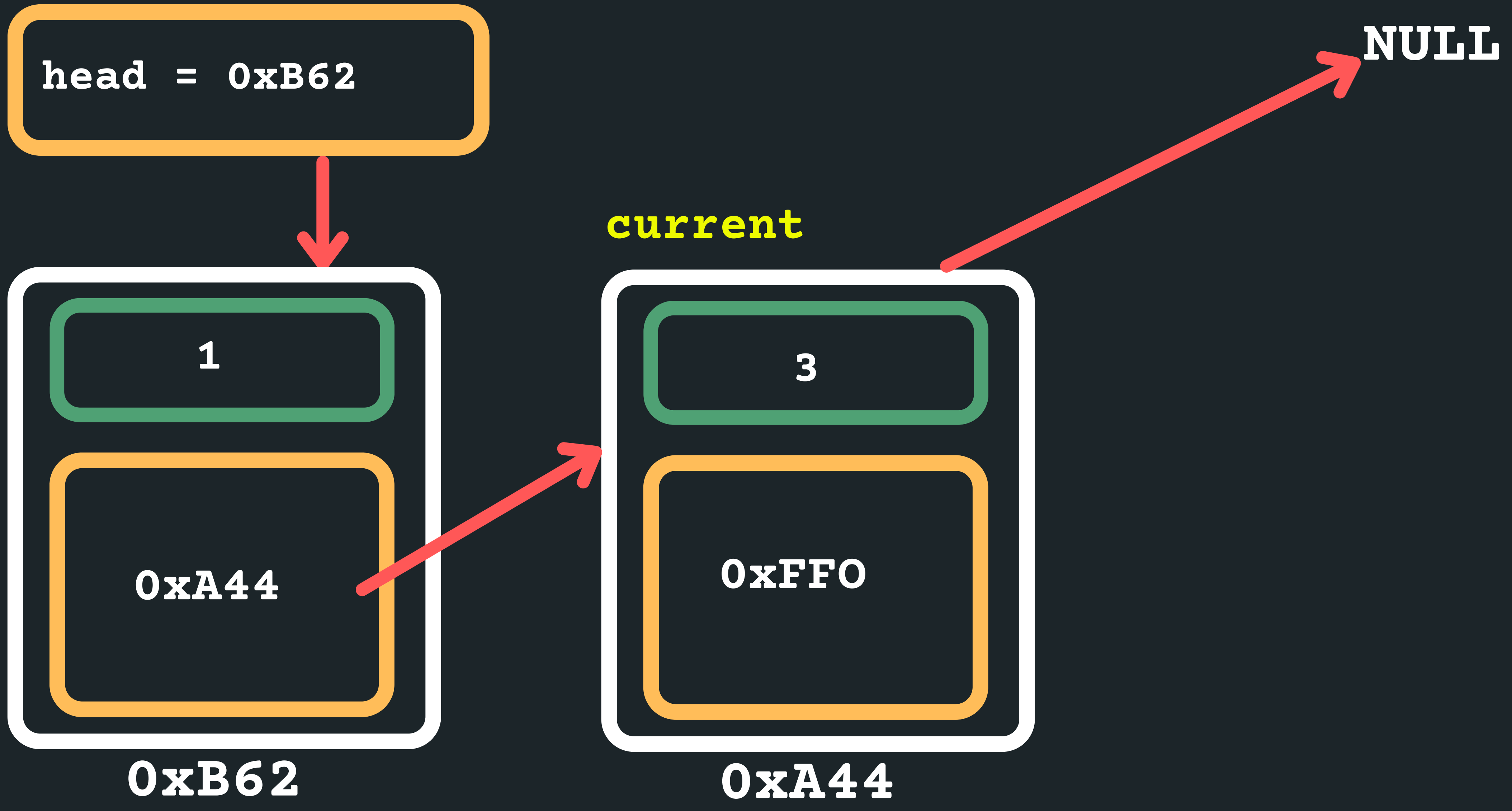


LINKED LISTS

DELETING THE TAIL

- Deleting when in the tail
  - Point my current->next node to a NULL

```
current->next = NULL;
```



LINKED LISTS

DELETING A NODE

- In all instances, we follow a similar structure of what to do when deleting a node. Please draw a diagram for yourself to really understand what you are deleting and the logic of deleting in a particular way.
- To delete a node in a linked list:
  - Find the previous node to the one that is being deleted
  - Change the next of the previous node
  - Free the node that is to be deleted
  - Consider possible edge cases, deleting if there is nothing in the list, deleting when there is only one item in the list, deleting the head of the list, deleting the tail of the list, etc.

LINKED LISTS

DELETING A NODE

```c
struct node *delete_node (struct node *head, int data) {
    // Create a current pointer set to the head of the list
    struct node *current = head;
    // Sometimes it is helpful to keep track of a previous node
    // to the current as that means you won't lose it....
    struct node *previous = NULL; // If the current node is at head, that
                                  // means the previous node is at NULL

    // What happens if we have an empty list?
    if (current == NULL) {
        return NULL;
    } else if (current->data == data) {
    // What happens if we need to delete the item that is
    // the head of the list?
        struct node *new_head = current->next;
        free(current);
        return new_head;
        // This will return whatever was after current as the
        // new head. If there is only one node in the list and
        // it is the one to be deleted, it will capture this (NULL)
    }

    // Otherwise start looping through the list to find the data
    // 1. Find the previous node to the one you want to delete
    while (previous->next->data != data && current->next != NULL) {
        previous = current;
        current = current->next;
    }

    // 2. If the current node is the one to be deleted
    if (previous->next->data == data) {
        //point the next node to the new pointer
        previous->next = current_next;
        // 3. free the node to be deleted
        free(current);

    }
    return head;
}
```

Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://forms.office.com/r/vJp6AR1R42

WHAT DID WE LEARN TODAY?

LINKED LISTS - DELETING

linked_list.c

LINKED LISTS - MULTI-FILE

linked_list.c

linked_list.h

main.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au