COMP1511 PROGRAMMING FUNDAMENTALS

# LECTURE 12

Linked Lists - What is happening?

What is it? Inserting at the head, traversing it, inserting at the tail

# LAST TIME...

- Technical disaster galore
- Green screens of death
- Malloc and free

# TODAY...

- Linked Lists - what is it?
- Linked list - insert at the head
- Linked list - traversal
- Linked list - insert at the tail

## WHERE IS THE CODE?

# Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T3/LIVE/WEEK07/

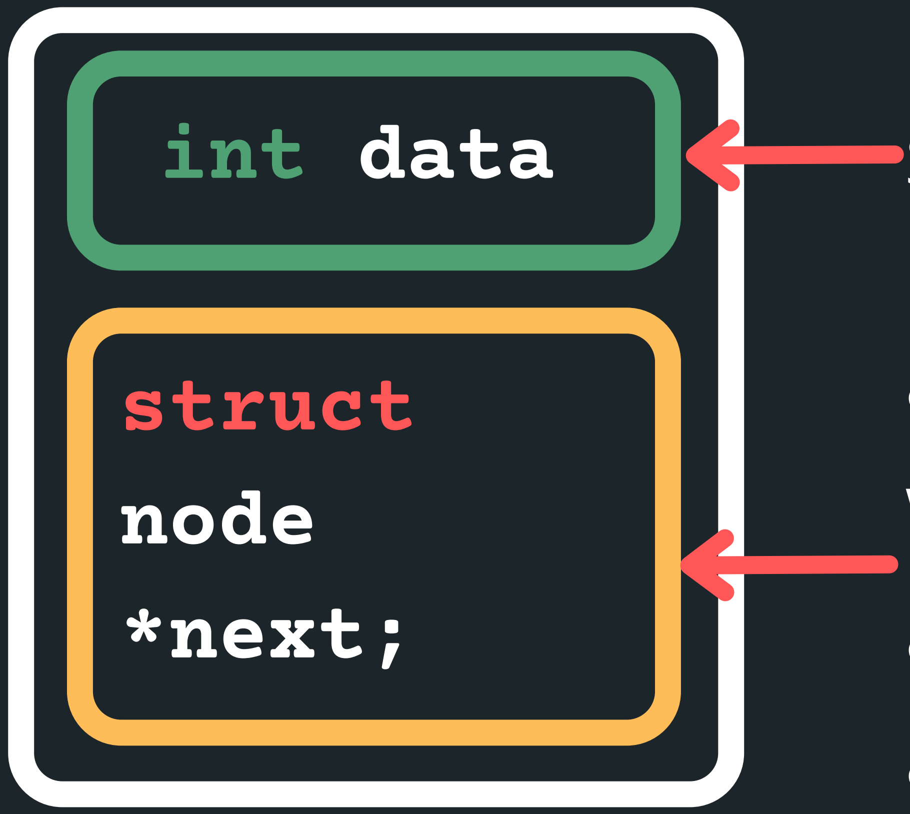
# A LINKED LIST IS MADE UP OF NODES

## WHAT IS A NODE?

- Each node has some data and a pointer to the next node (of the same data type), creating a linked structure that forms the list
- Let me propose a node structure like this:

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

node



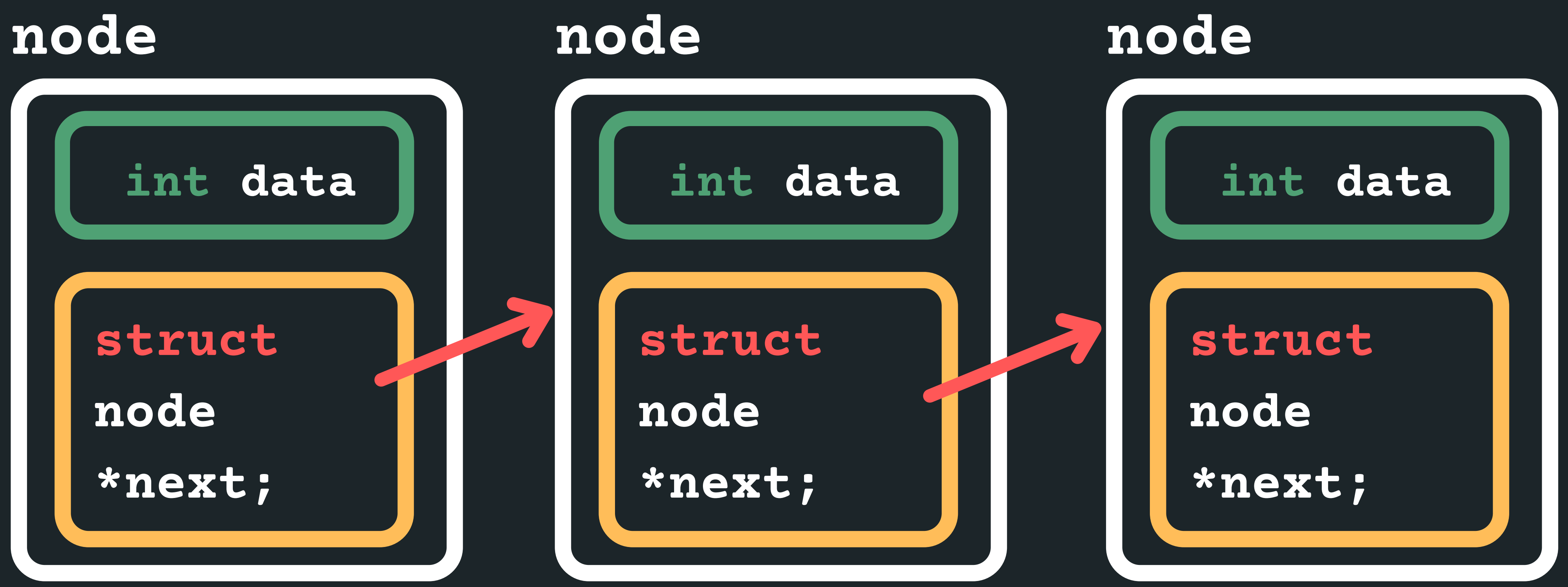

some data of type int

a pointer to the next node, which also has some data and a pointer to the node after that... etc

# A LINKED LIST IS MADE UP OF MANY NODES

## THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- We can create a linked list, by having many nodes together, with each struct node next pointer giving us the address of the node that follows it

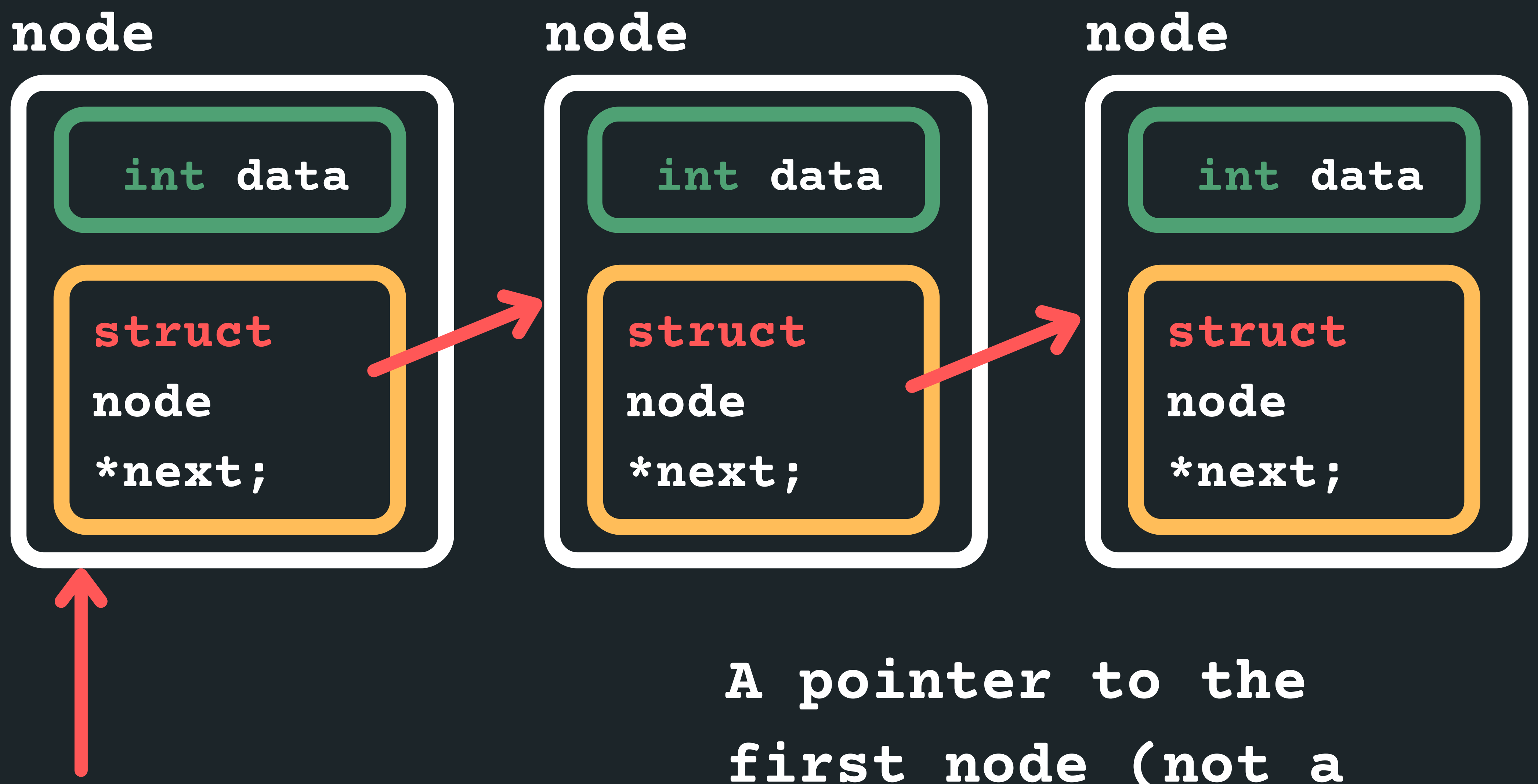



- But how do I know where the linked list starts?

# A LINKED LIST IS MADE UP OF MANY NODES

## THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- What about a pointer to the first node?




A pointer to the first node

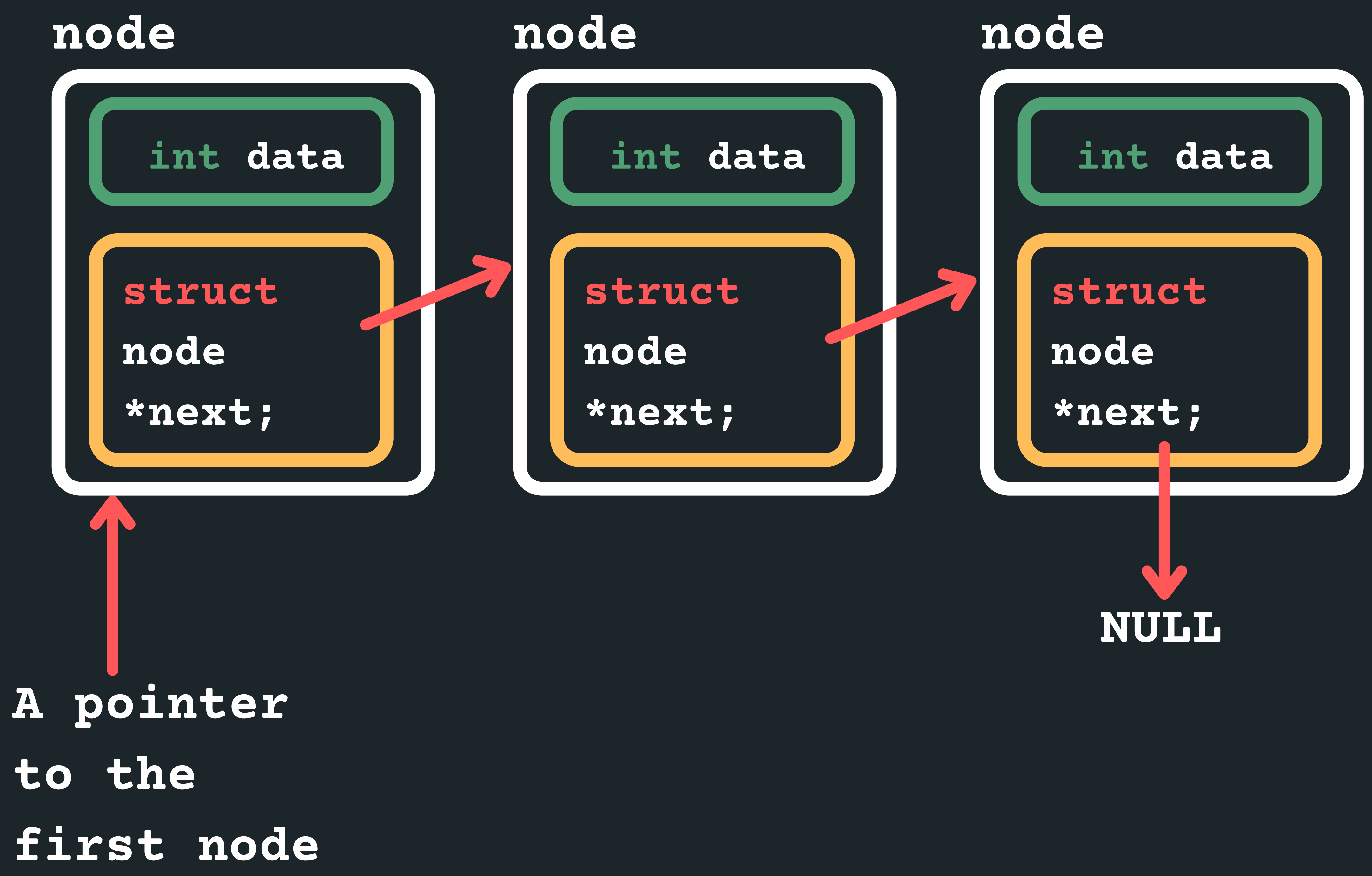
A pointer to the first node (not a node itself, but has the memory address of where the first node is!

- How do I know when my list is finished?

# A LINKED LIST IS MADE UP OF MANY NODES

## THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

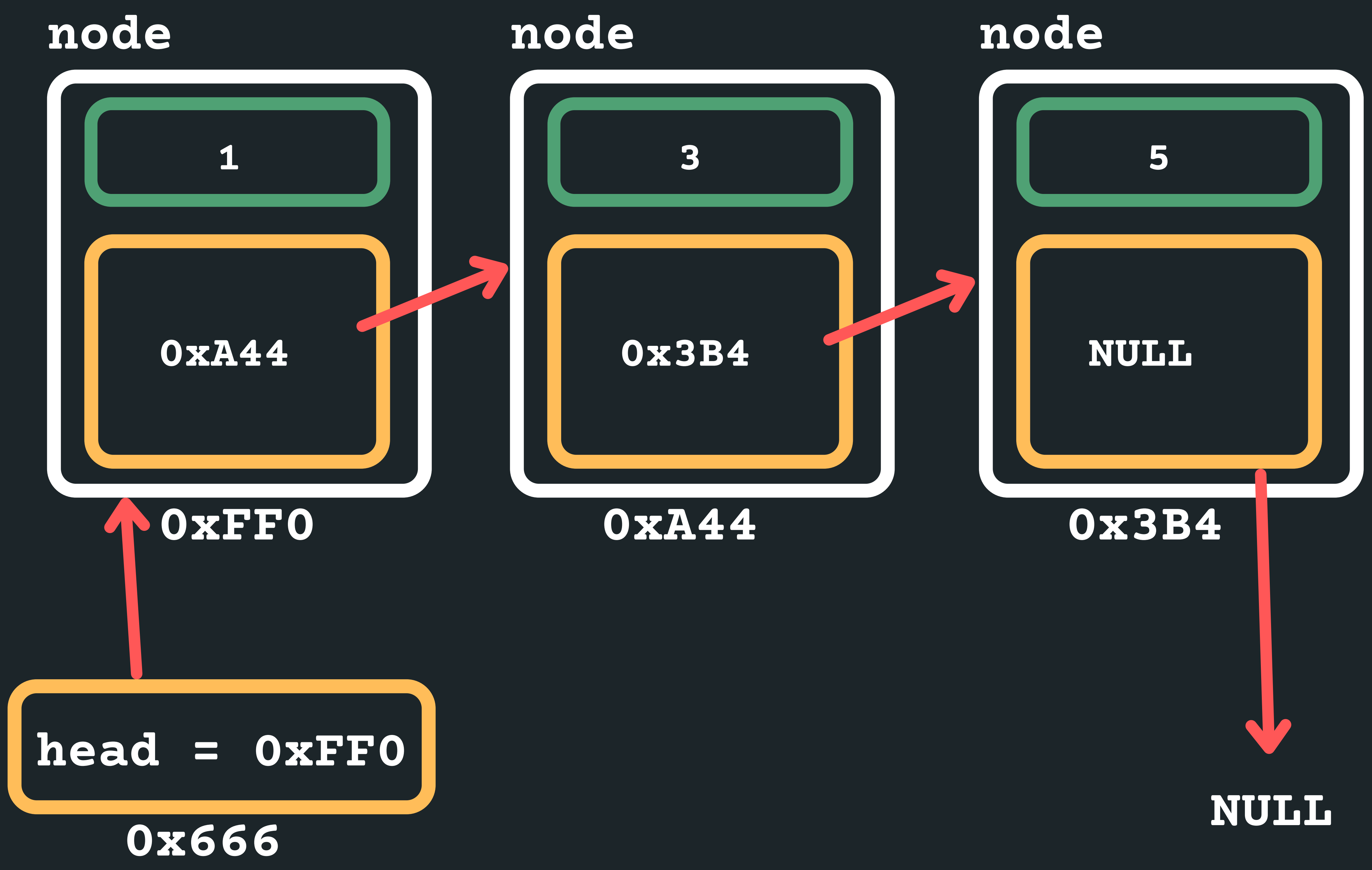
- Pointing to a NULL at the end!



# A LINKED LIST IS MADE UP OF MANY NODES

THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- For example, a list with: 1, 3, 5



# A LINKED LIST

## WHY?

- Linked lists are dynamically sized, that means we can grow and shrink them as needed - efficient for memory!
- Elements of a linked list (called nodes) do NOT need to be stored contiguously in memory, like an array.
- We can add or remove nodes as needed anywhere in the list, without worrying about size (unless we run out of memory of course!)
- We can change the order in a linked list, by just changing where the next pointer is pointing to!
- Unlike arrays, linked lists are not random access data structures! You can only access items sequentially, starting from the beginning of the list.

# A LINKED LIST

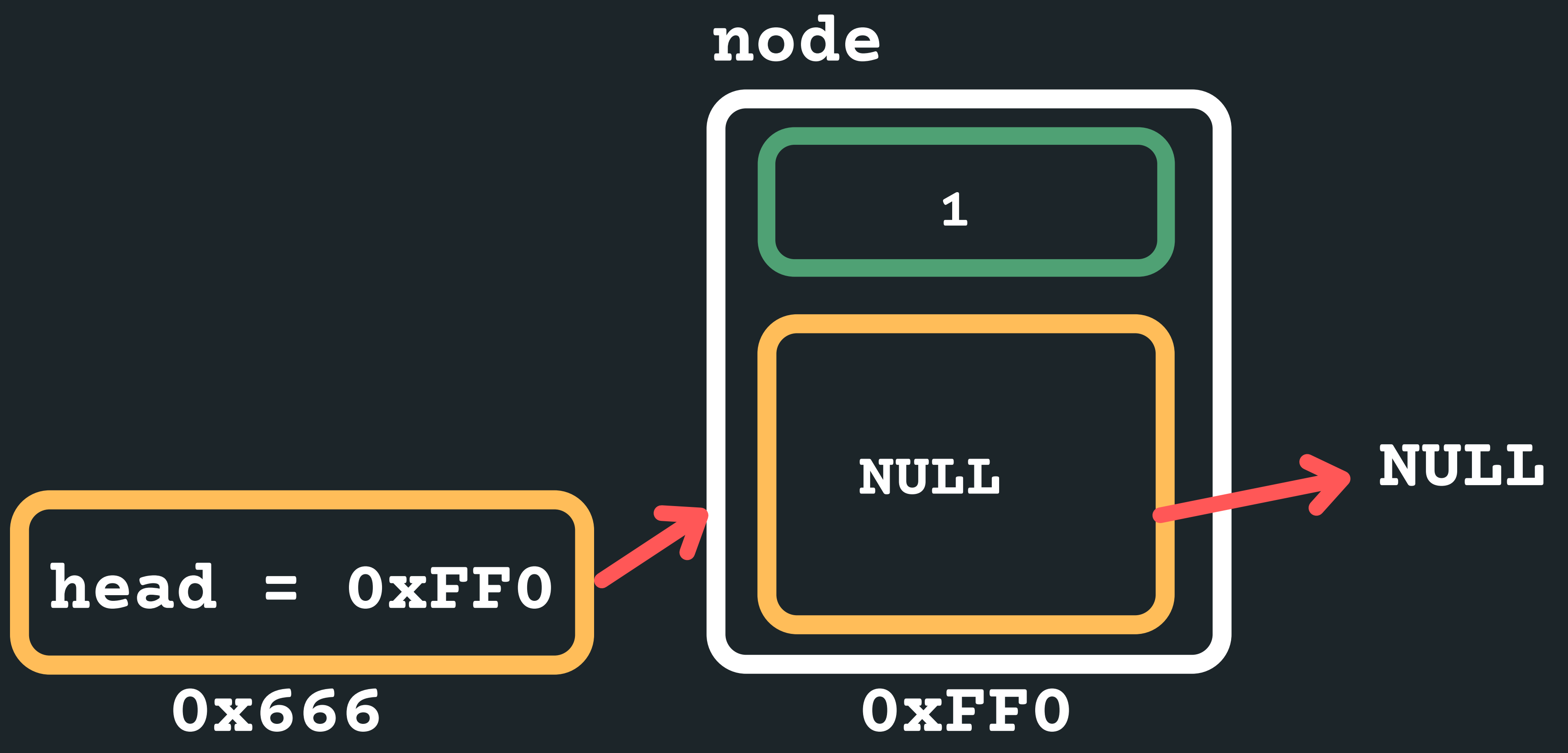
## HOW DO WE CREATE ONE AND INSERT INTO IT?

- In order to create a linked list, we would need to
  - Define  struct for a node,
  - A pointer to keep track of where the start of the list is and
  - A way to create a node and then connect it into our list...

# A LINKED LIST

## HOW DO WE CREATE ONE AND INSERT INTO IT?

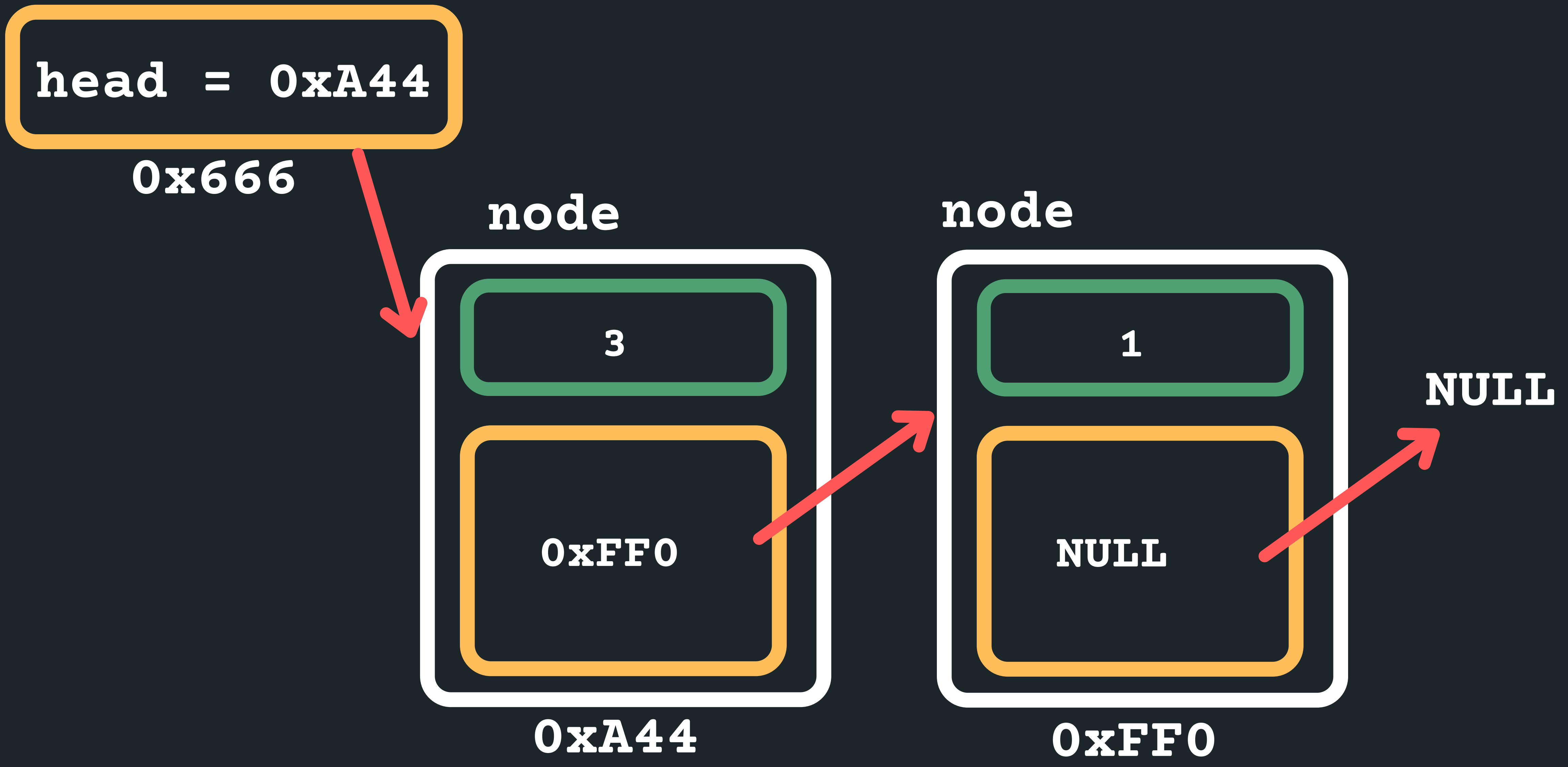
- Let's say we wanted to create a linked list with 5, 3, 1
  - Let's create the first node to start the list!
  - A pointer to keep track of where the start of the list is and by default the first node of the list
  - It will point to NULL as there are no other nodes in this list.



# A LINKED LIST

## HOW DO WE CREATE ONE AND INSERT INTO IT?

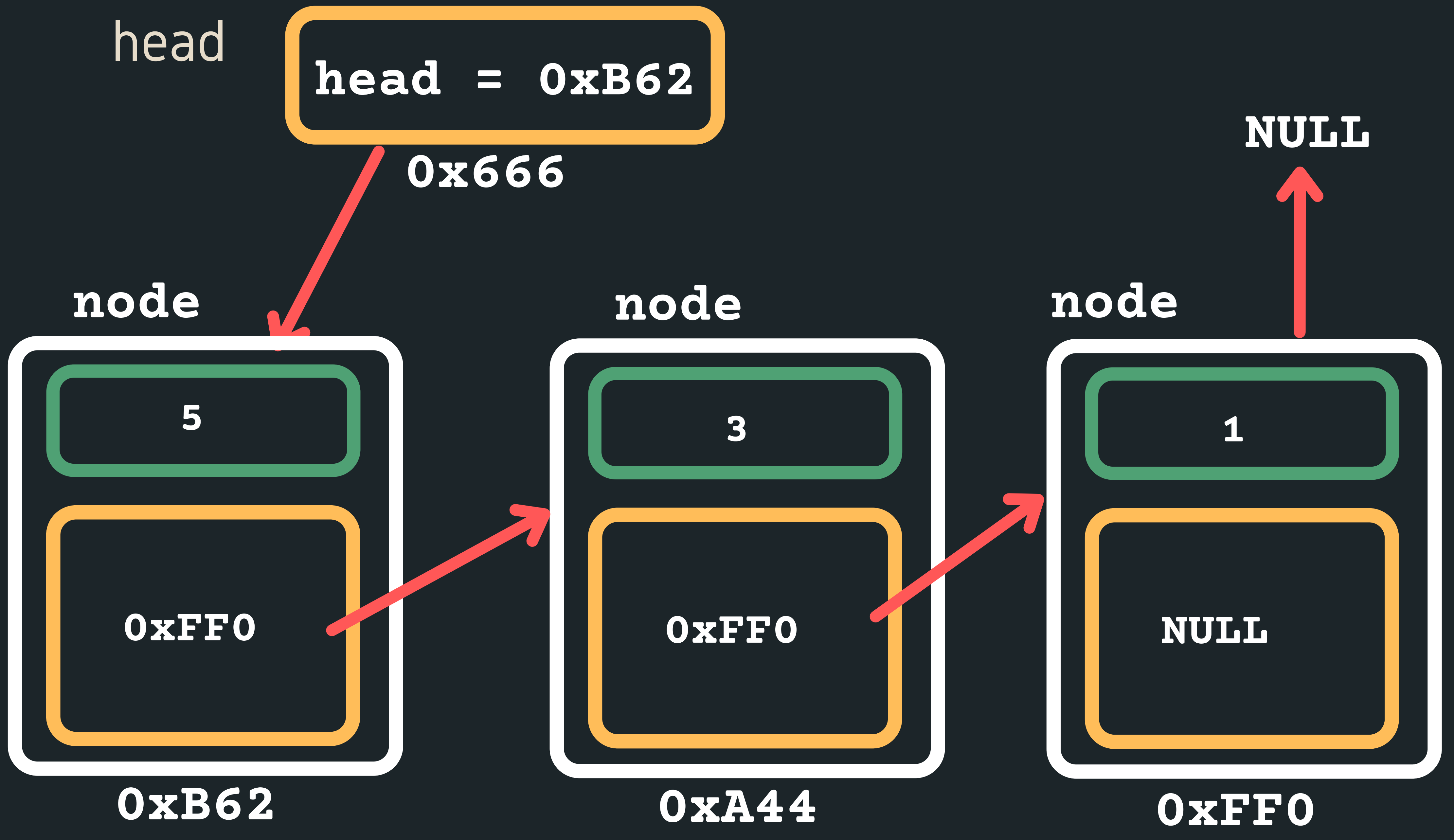
- Create the next node to store 3 into (you need memory)
- Assign 3 to data
- and insert it at the beginning so the head would now point to it and the new node would point to the old head



# A LINKED LIST

## HOW DO WE CREATE ONE AND INSERT INTO IT?

- Create the next node to store 5 into (you need memory)
- Assign 5 to data
- and insert it at the beginning so the head would now point to it and the new node would point to the old head



# BREAK TIME...

You have five boxes in a row numbered 1 to 5, in one of which, a cat is hiding. Every night he jumps to an adjacent box, and every morning you have one chance to open a box to find him. How do you win this game of hide and seek - what is your strategy? What if there are n boxes?

# A LINKED LIST

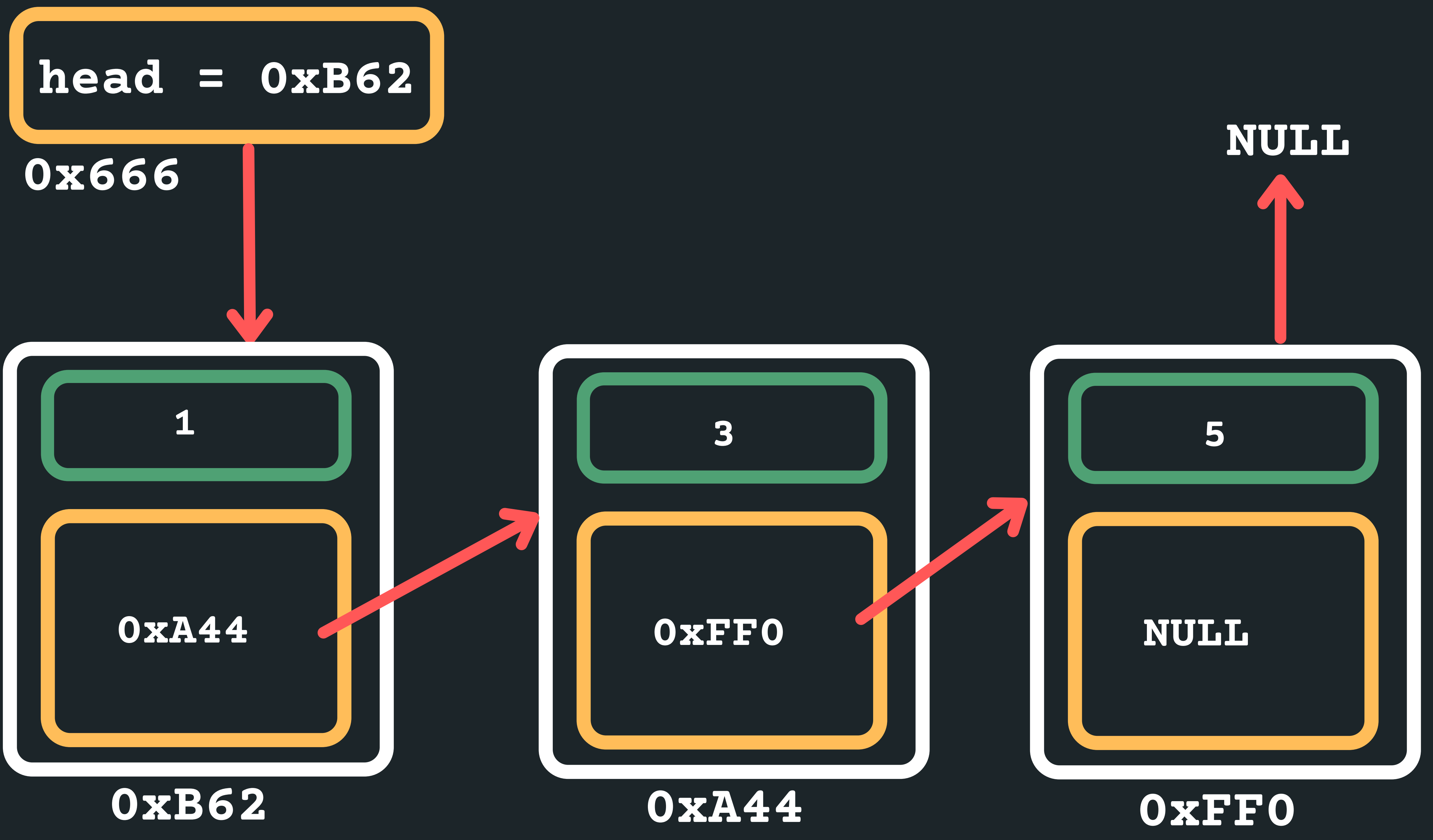
## PUTTING IT ALL TOGETHER IN CODE

1. Define our struct for a node
2. A pointer to keep track of where the start of the list is:
   - The pointer would be of type struct node, because it is pointing to the first node
   - The first node of the list is often called the 'head' of the list (last element is often called the 'tail')
3. A way to create a node and then connect it into our list...
   - Create a node by first creating some space for that node (malloc)
   - Initialise the data component on the node
   - Initialise where the node is pointing to
4. Make sure last node is pointing to NULL

# A LINKED LIST IS MADE UP OF MANY NODES

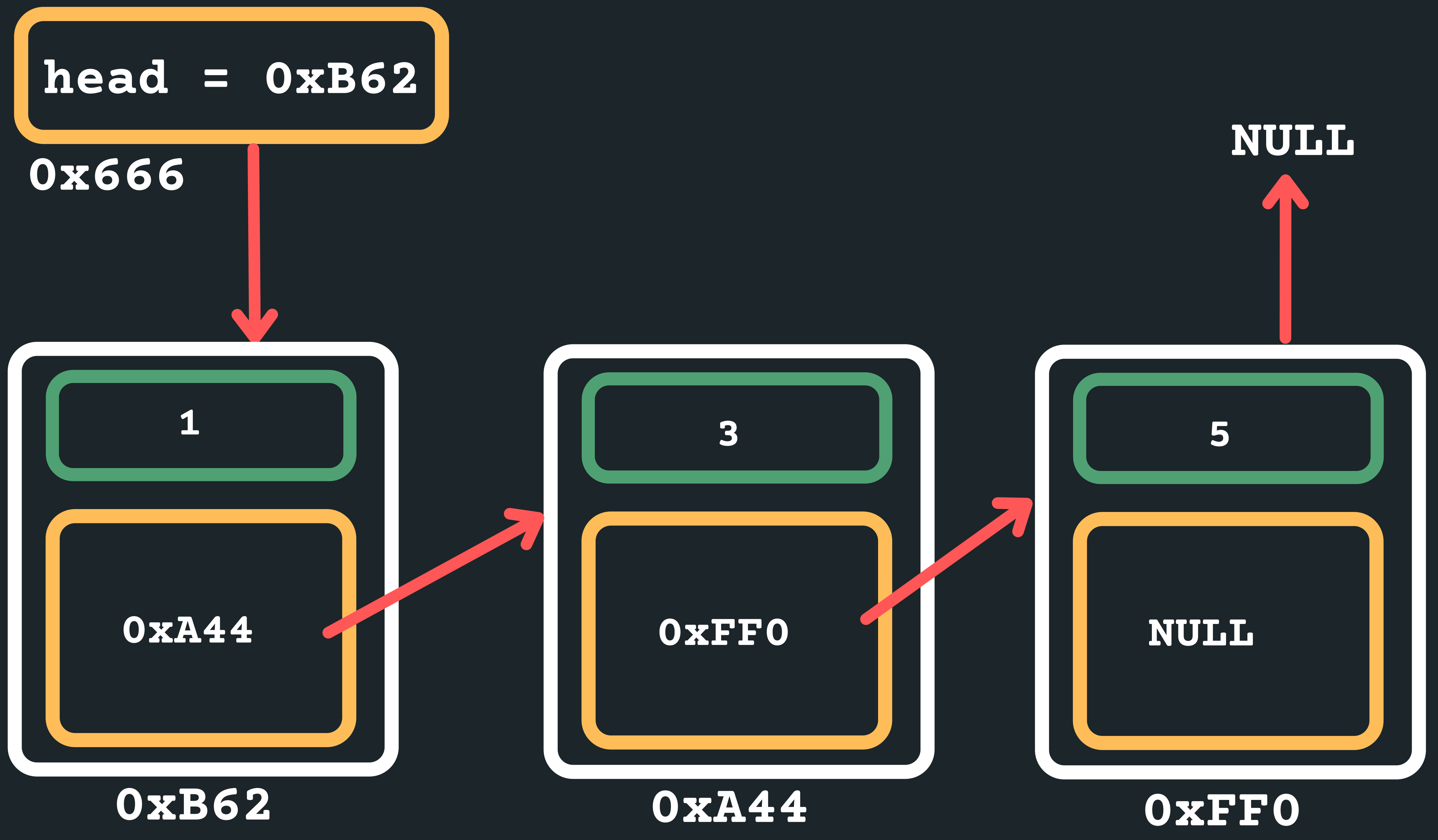
## THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- For example a list with 1, 3, 5



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

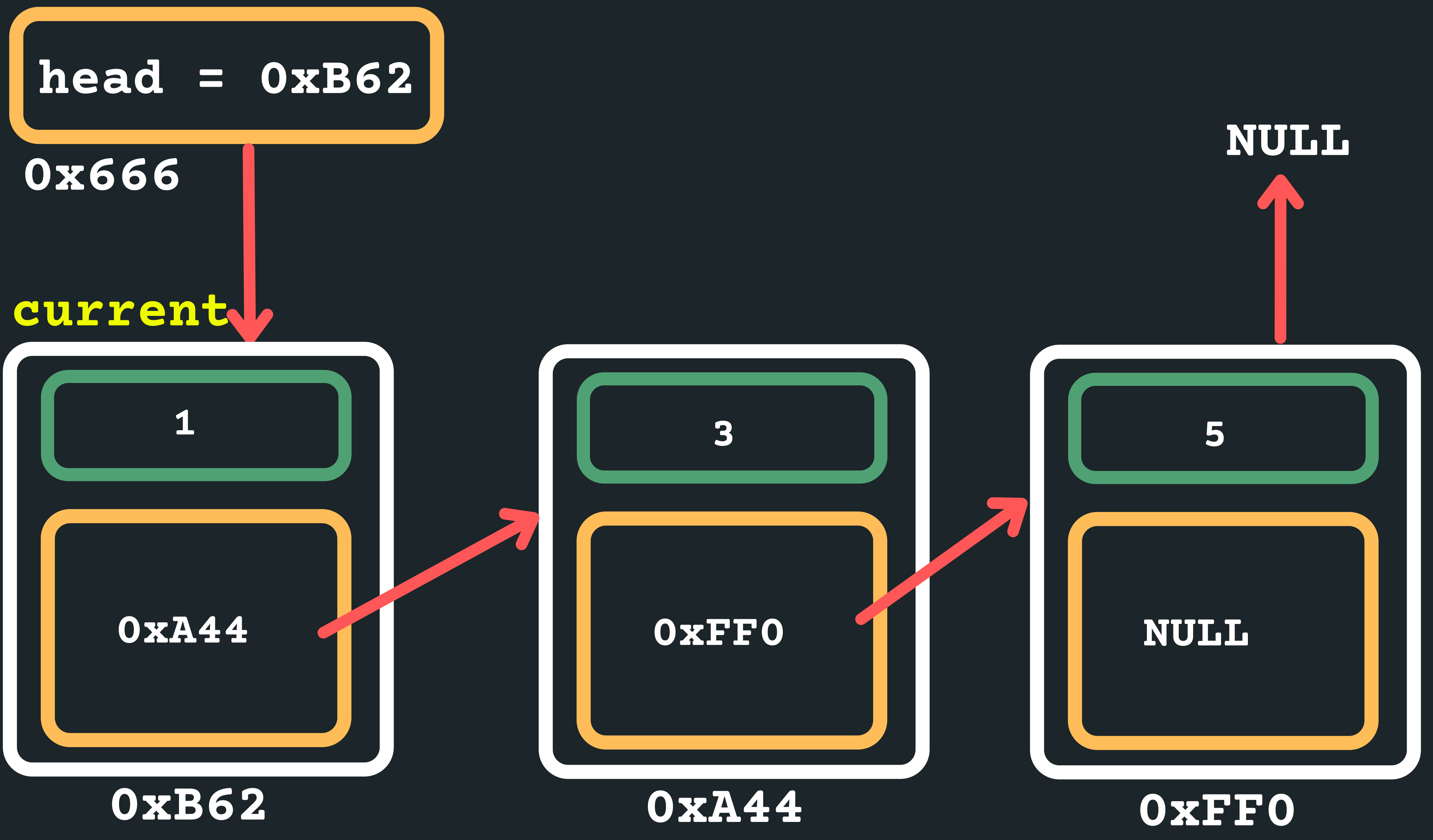
- How do you think we can move through the list to start a the head and then move to each subsequent node until we get to the end of the list...



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Set your head pointer to the current pointer to keep track of where you are currently located....

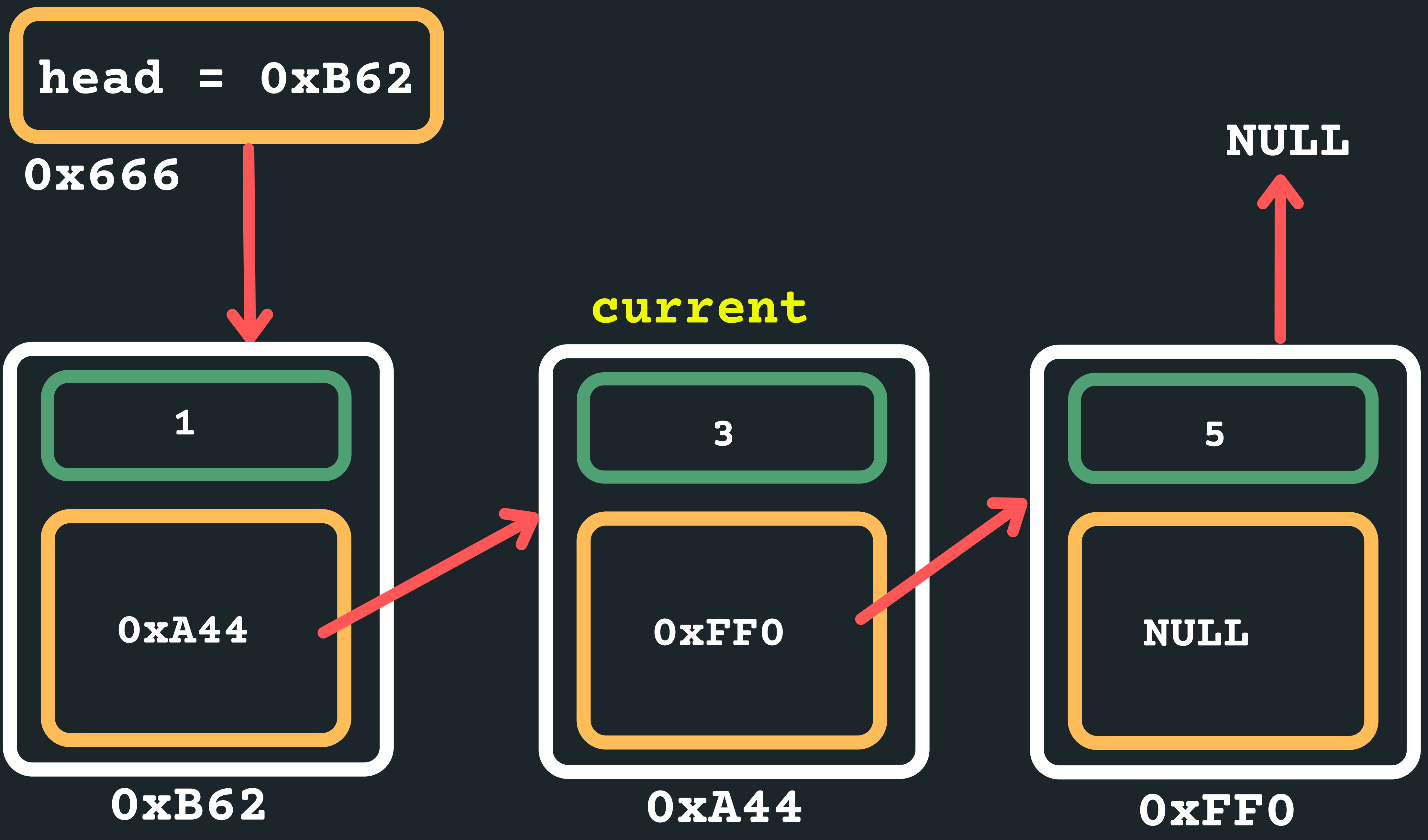
```
struct node *current = head
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

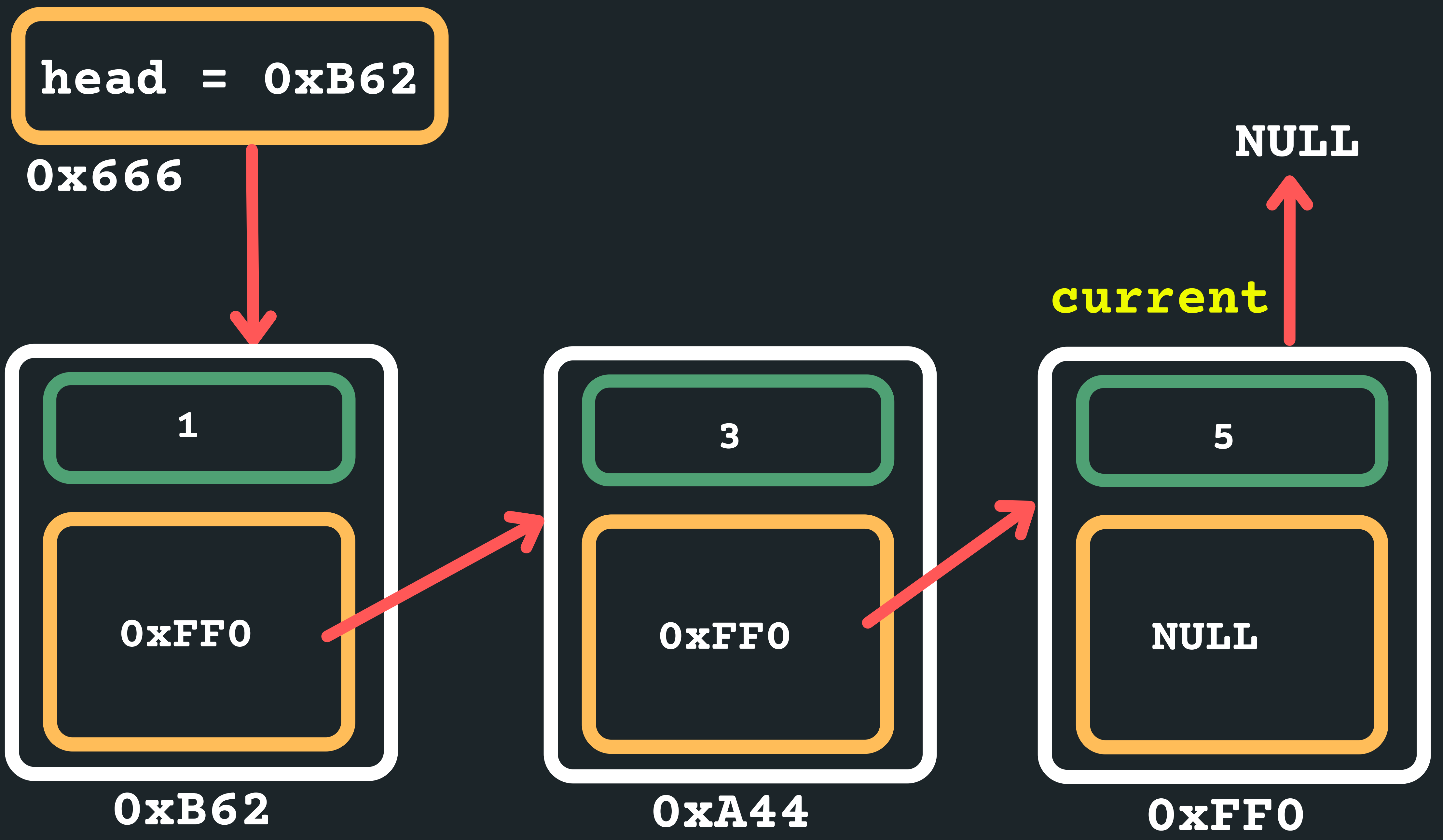
```
current = current->next
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```



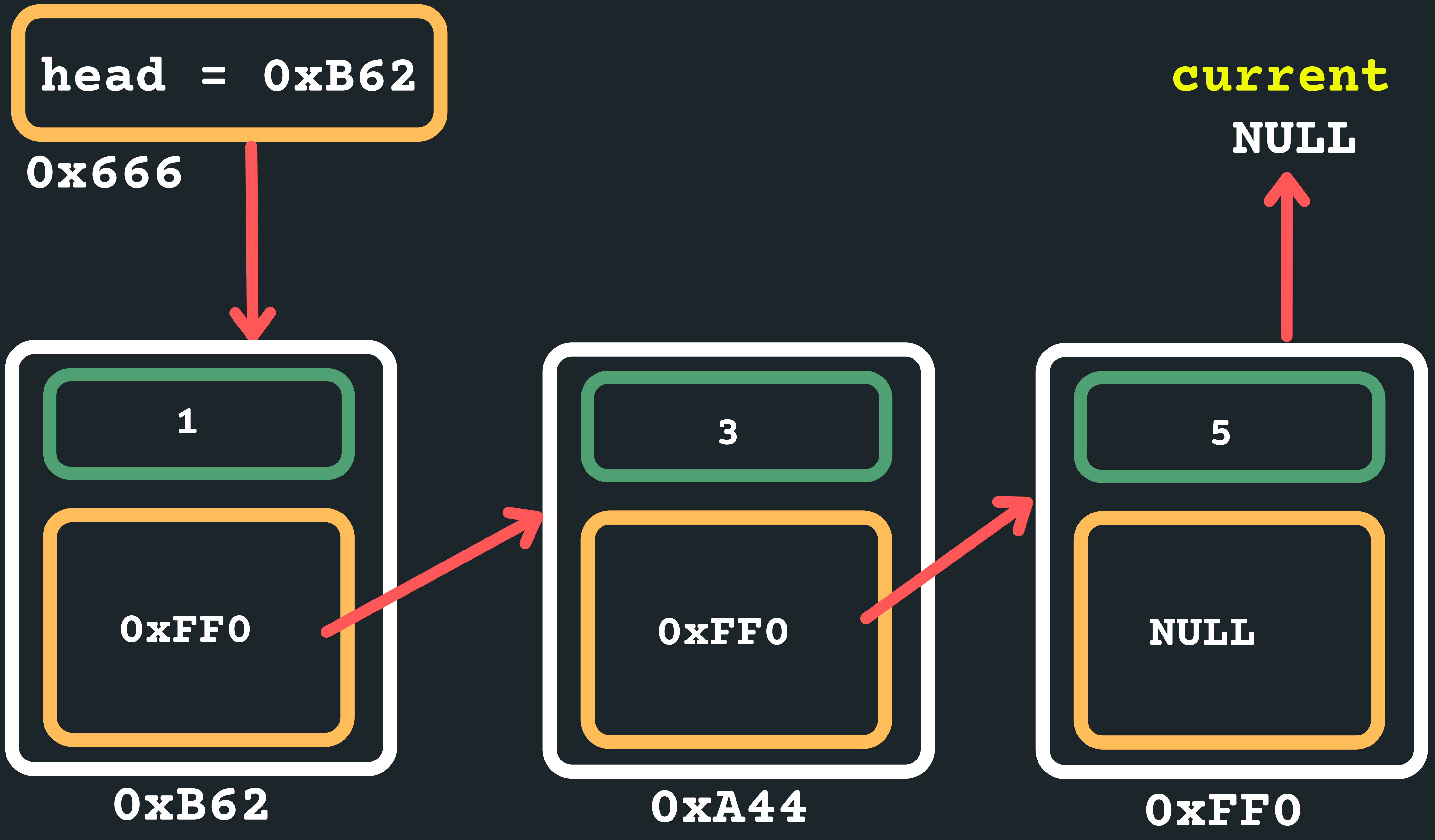
# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```

When should I be stopping?

```
while (current != NULL)
```



# SO TRAVERSING A LINKED LIST...

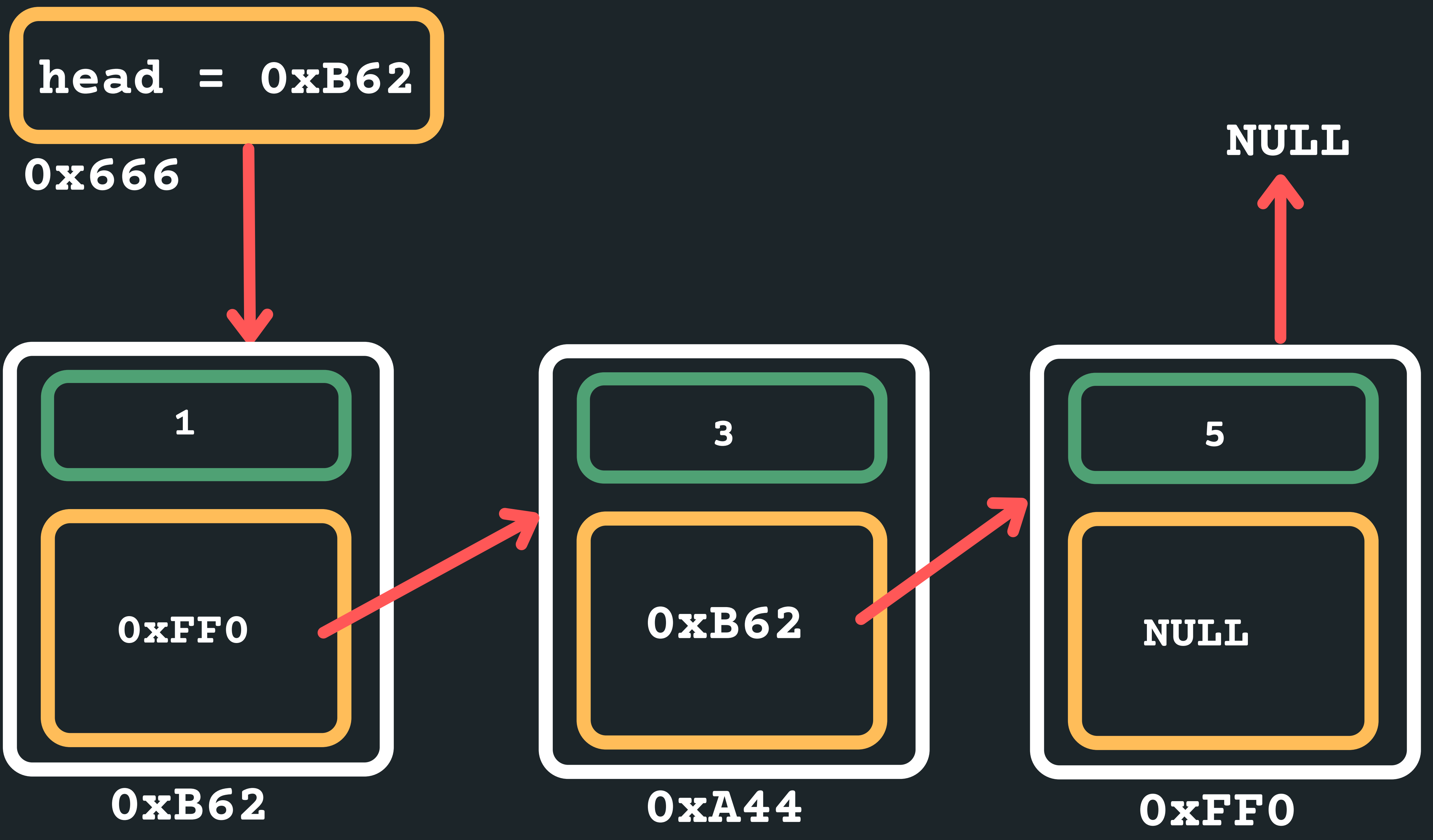
- The only way we can make our way through the linked list is like a scavenger hunt, we have to follow the links from node to node (sequentially! we can’t skip nodes)
- We have to know where to start, so we need to know the head of the list
- When we reach the NULL pointer, it means we have come to the end of the list.

# SO NOW, LET'S PRINT EACH NODE OUT…

```
void print_list(struct node *head){
    struct node *current = head;
    while (current != NULL){
        printf("%d\n", current->data);
        current = current->next;
    }
}
```

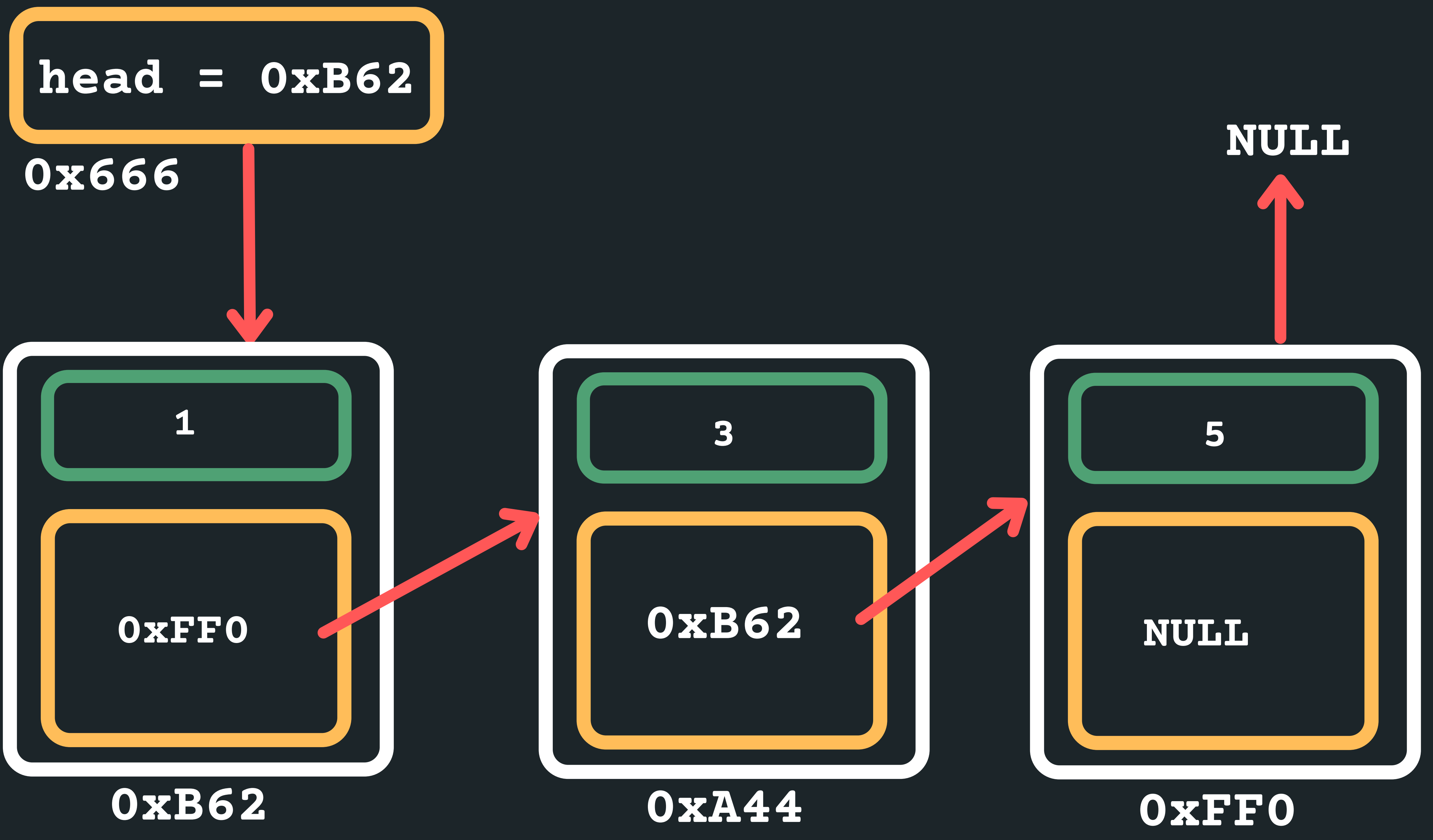
# INSERTING ANYWHERE IN A LINKED LIST...

- Where can I insert in a linked list?
  - At the head (what we just did!)
  - Between any two nodes that exist (next lecture!)
  - After the tail as the last node (now!)



# INSERTING ANYWHERE IN A LINKED LIST...

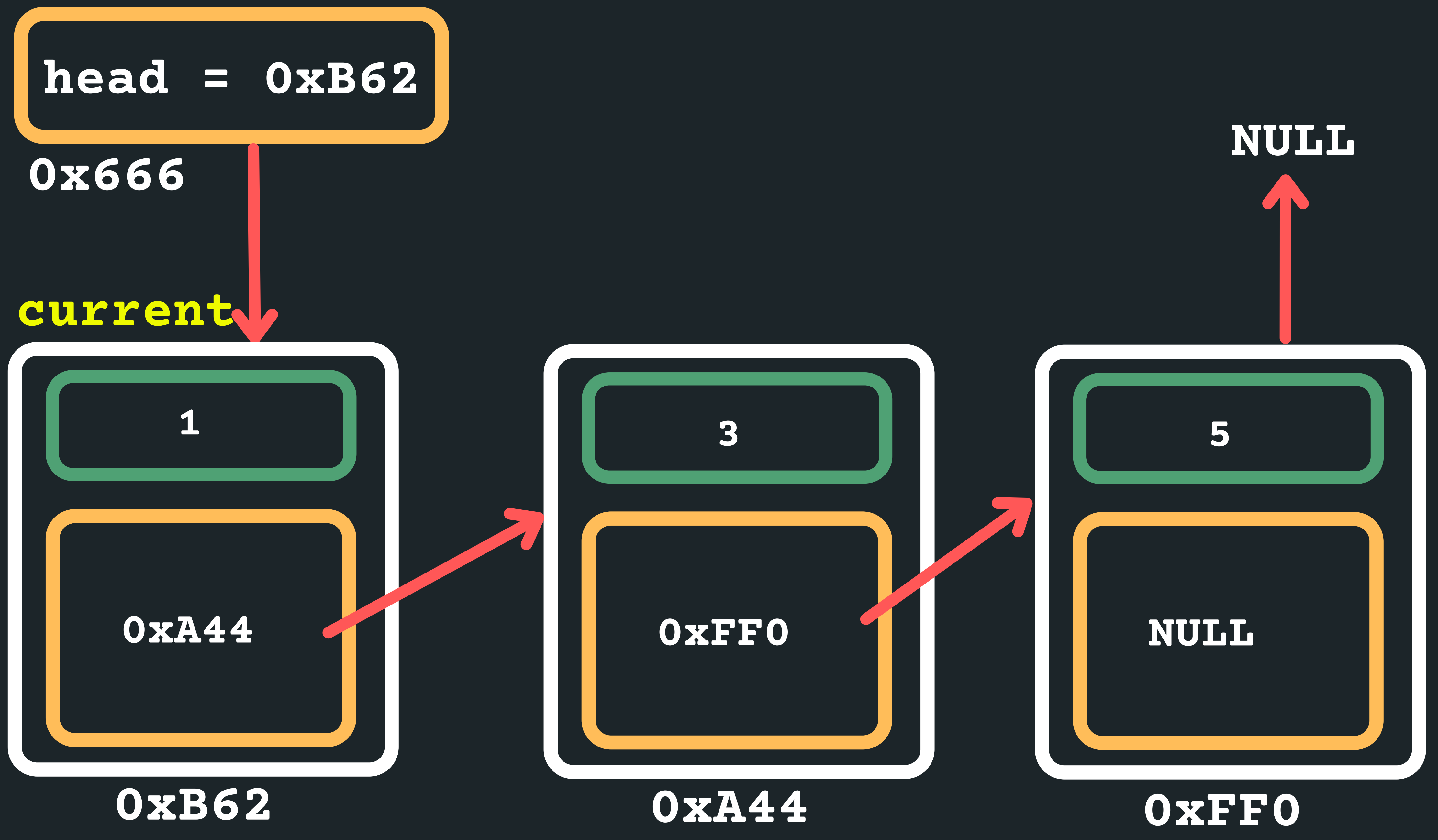
- Where can I insert in a linked list?
  - At the head (what we just did!)
  - Between any two nodes that exist (next lecture!)
  - After the tail as the last node (now!)



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Set your head pointer to the current pointer to keep track of where you are currently located....

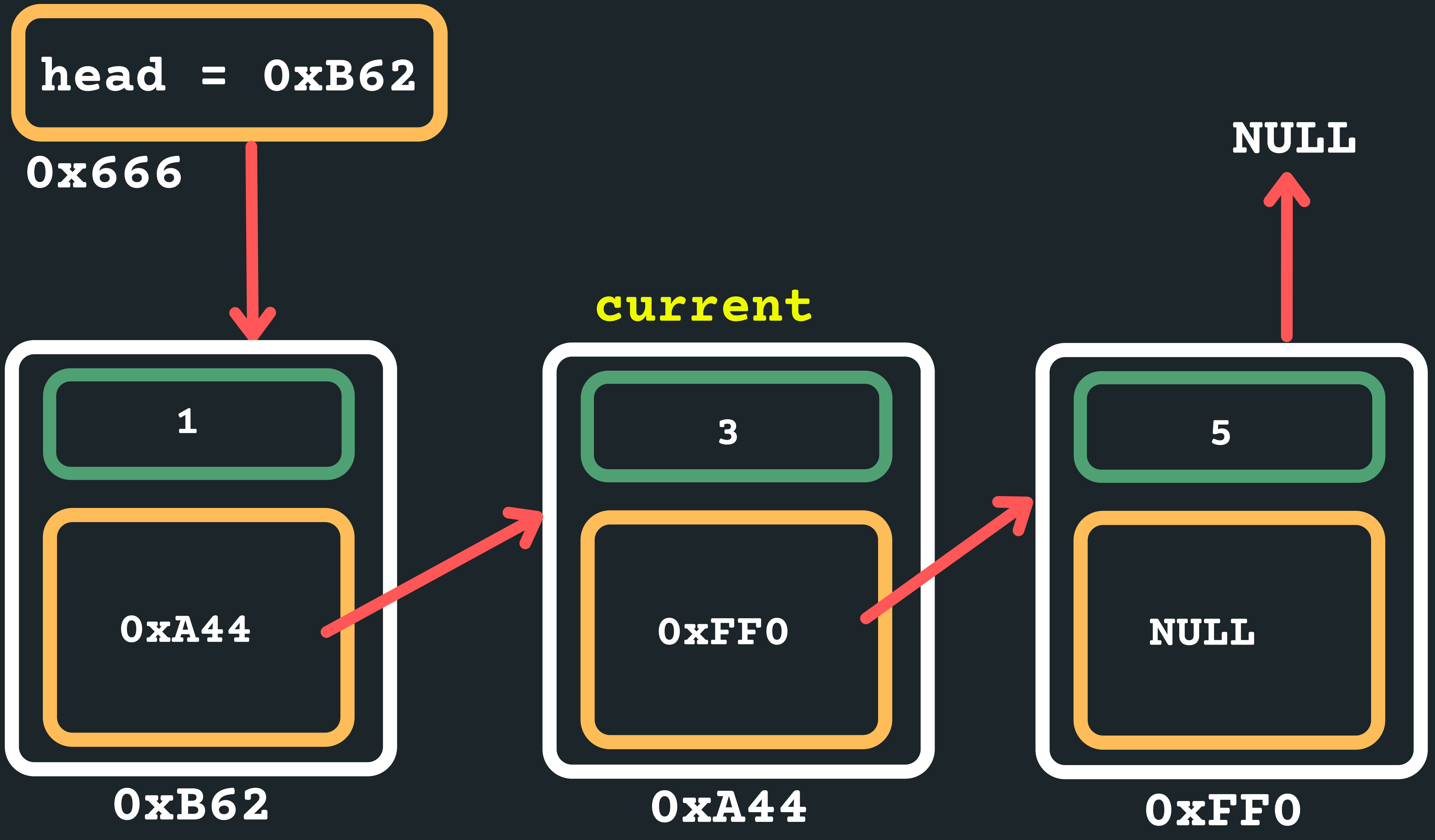
```
struct node *current = head
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

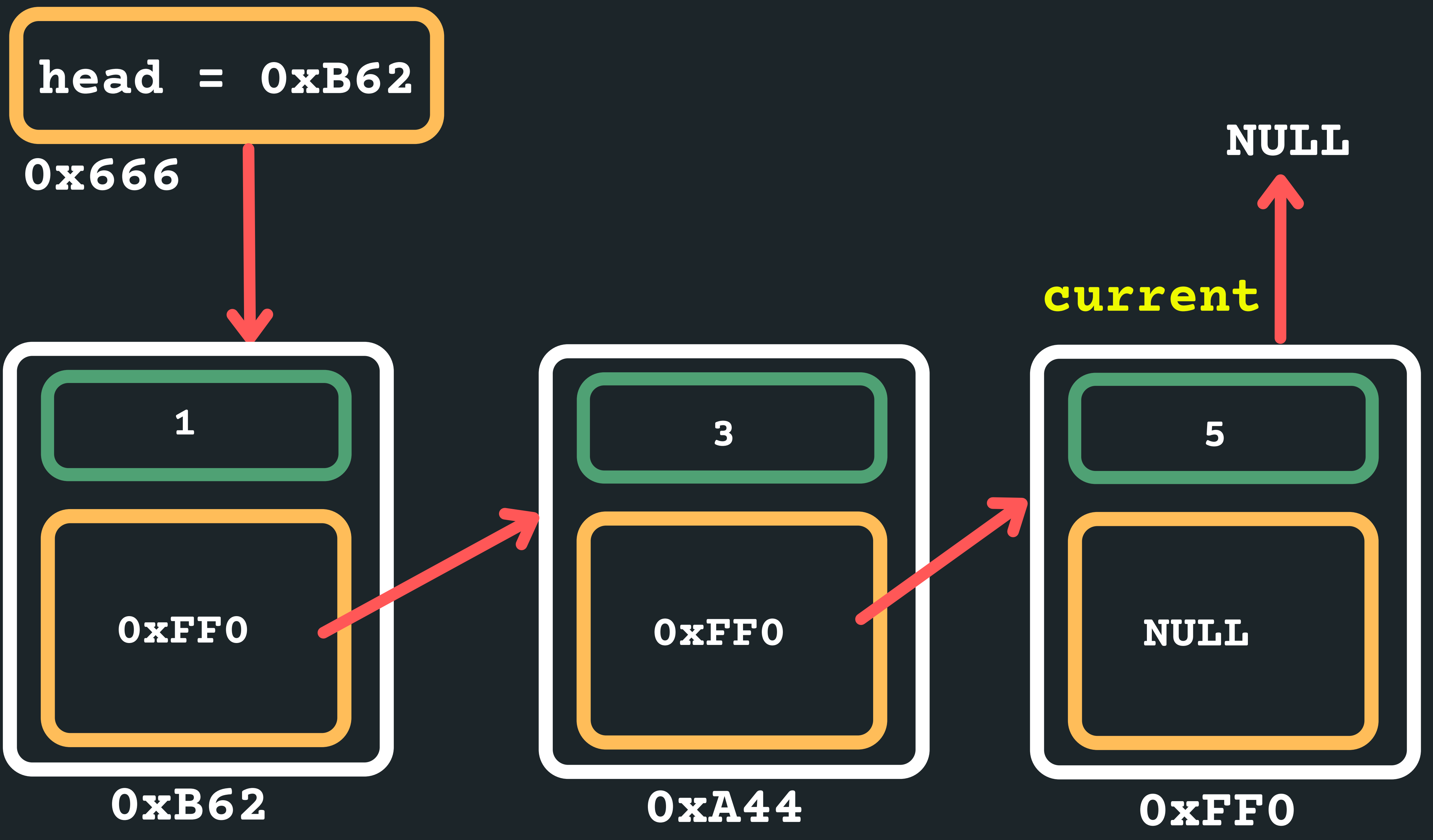
```
current = current->next
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```



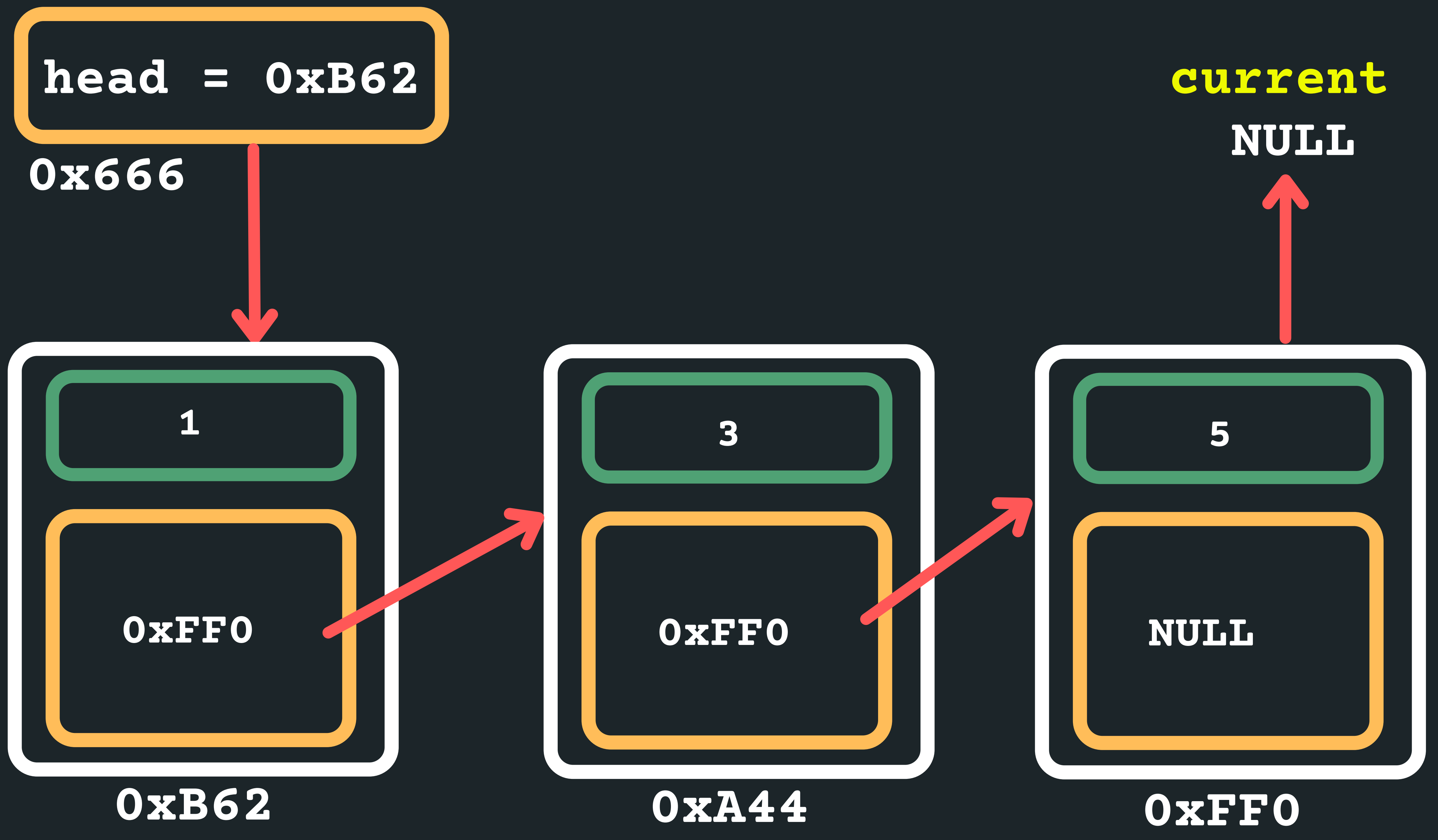
# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```

When should I be stopping?

```
while (current != NULL)
```



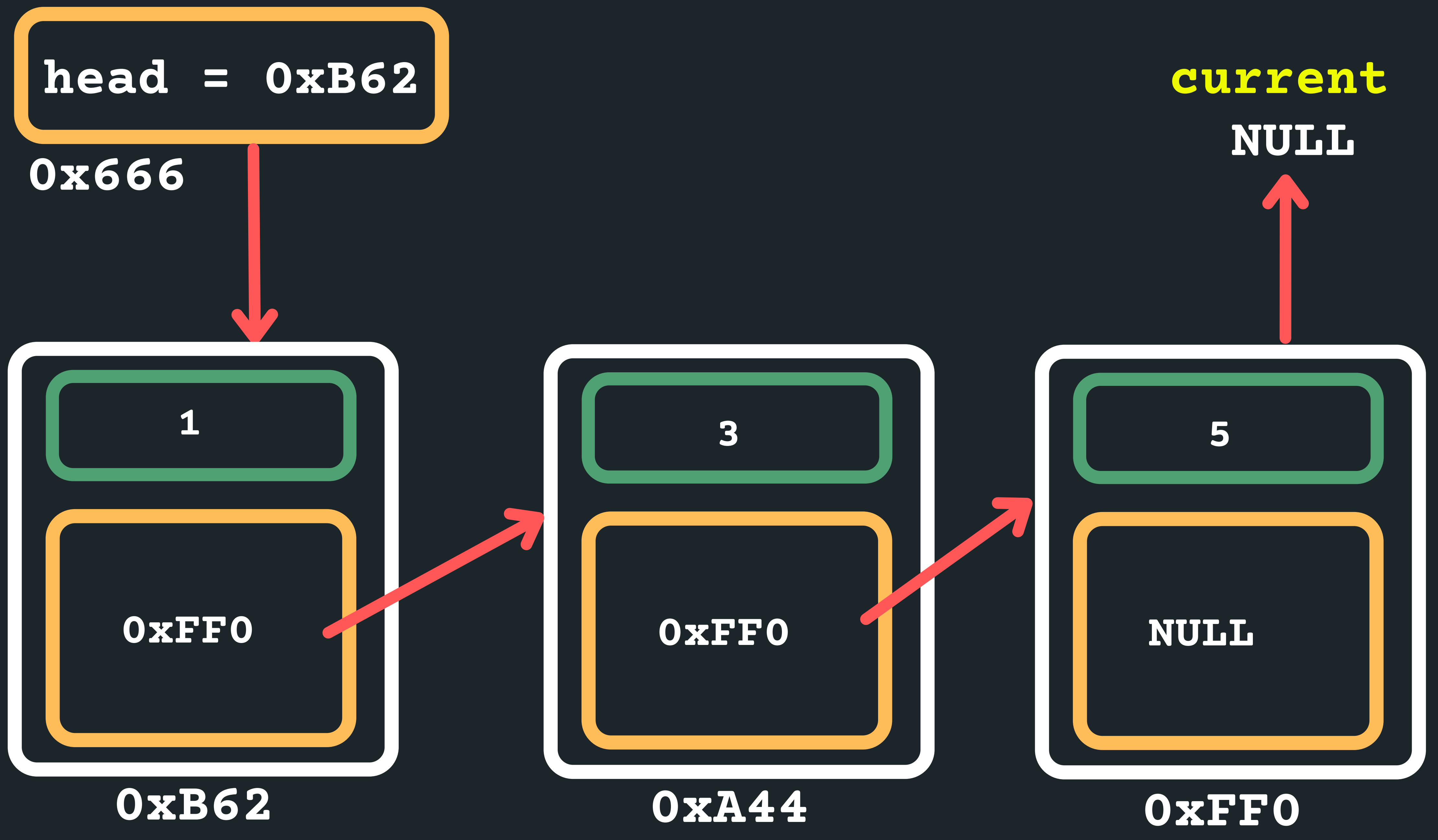
# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```

When should I be stopping? If you stop at current = NULL that means you won't know what the address of the previous node is!

```
while (current != NULL)
```



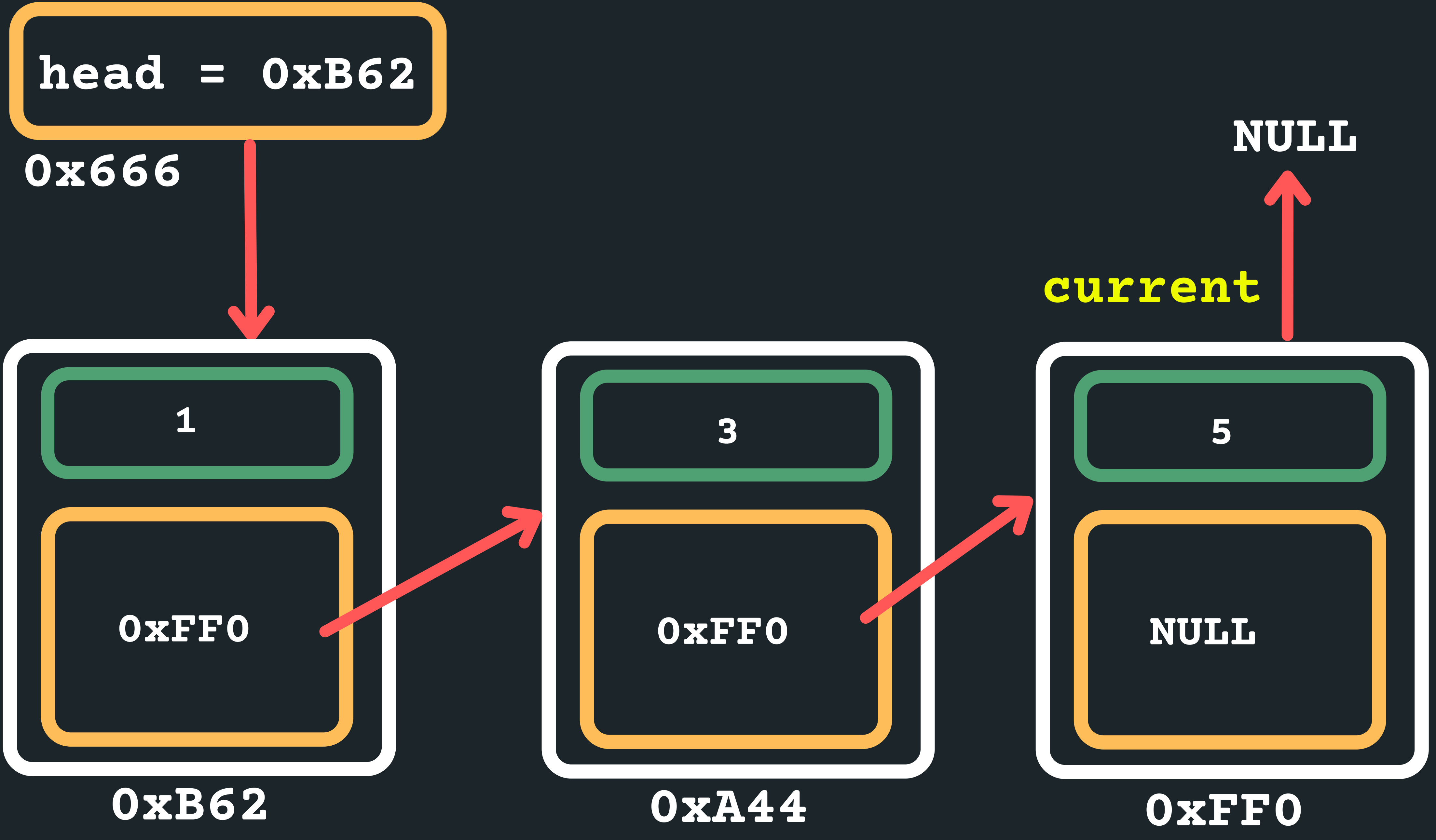
# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now how would we move the current along?

```
current = current->next
```

So let's stop at the last node...

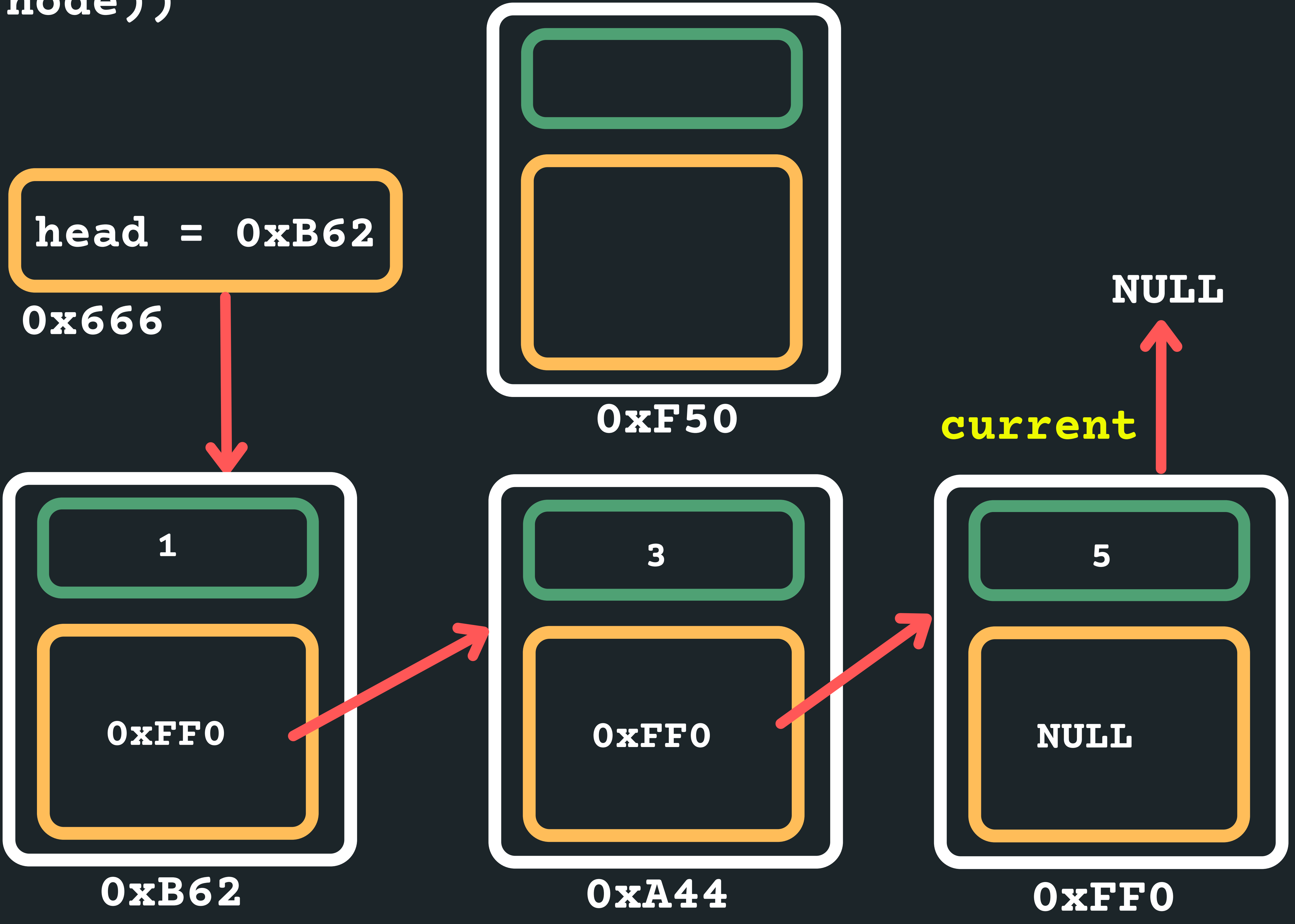
```
while (current->next != NULL)
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now we want to create a new node to insert:

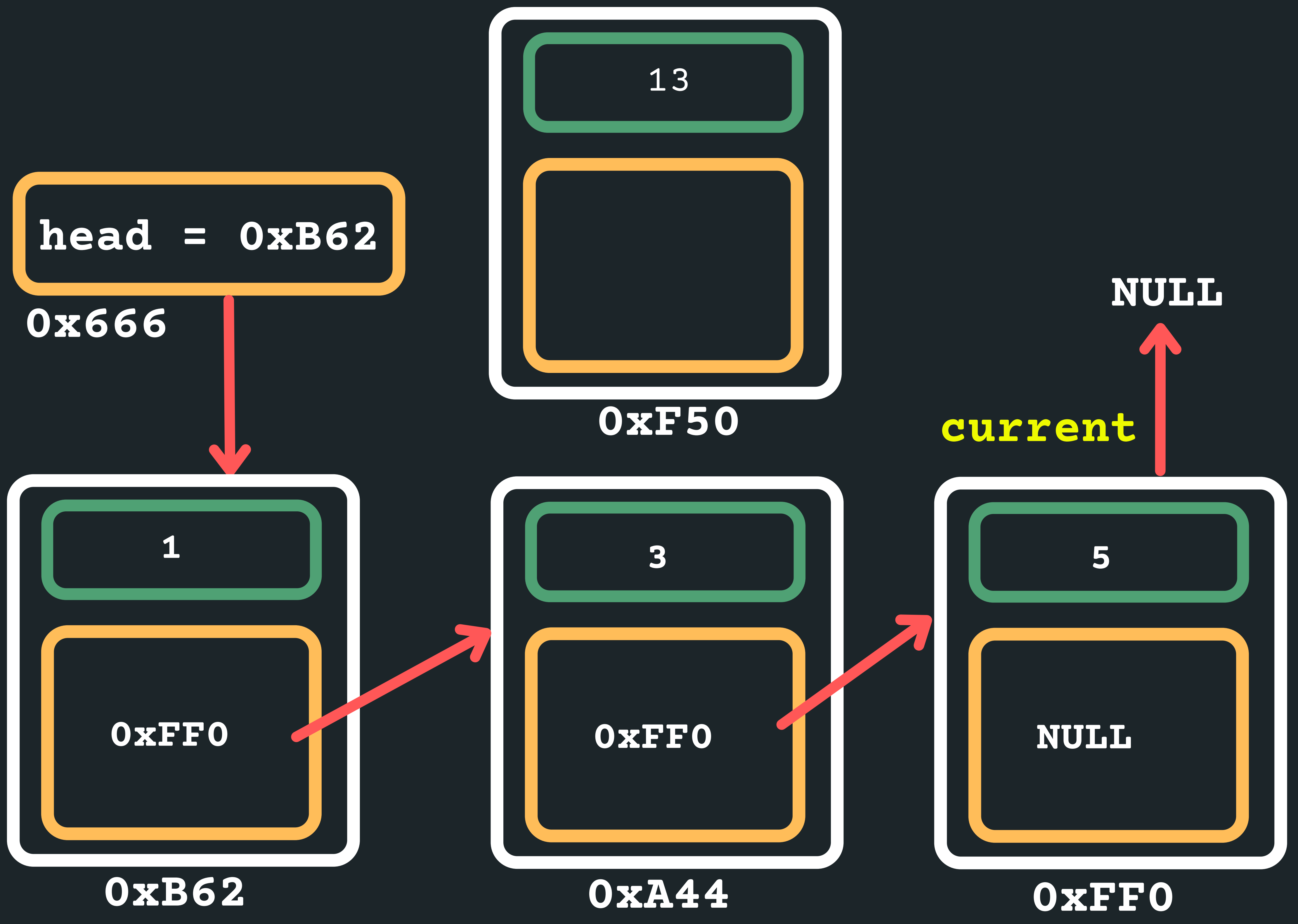
```
struct node new_node = malloc(sizeof(struct node))
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Assign values to new node:

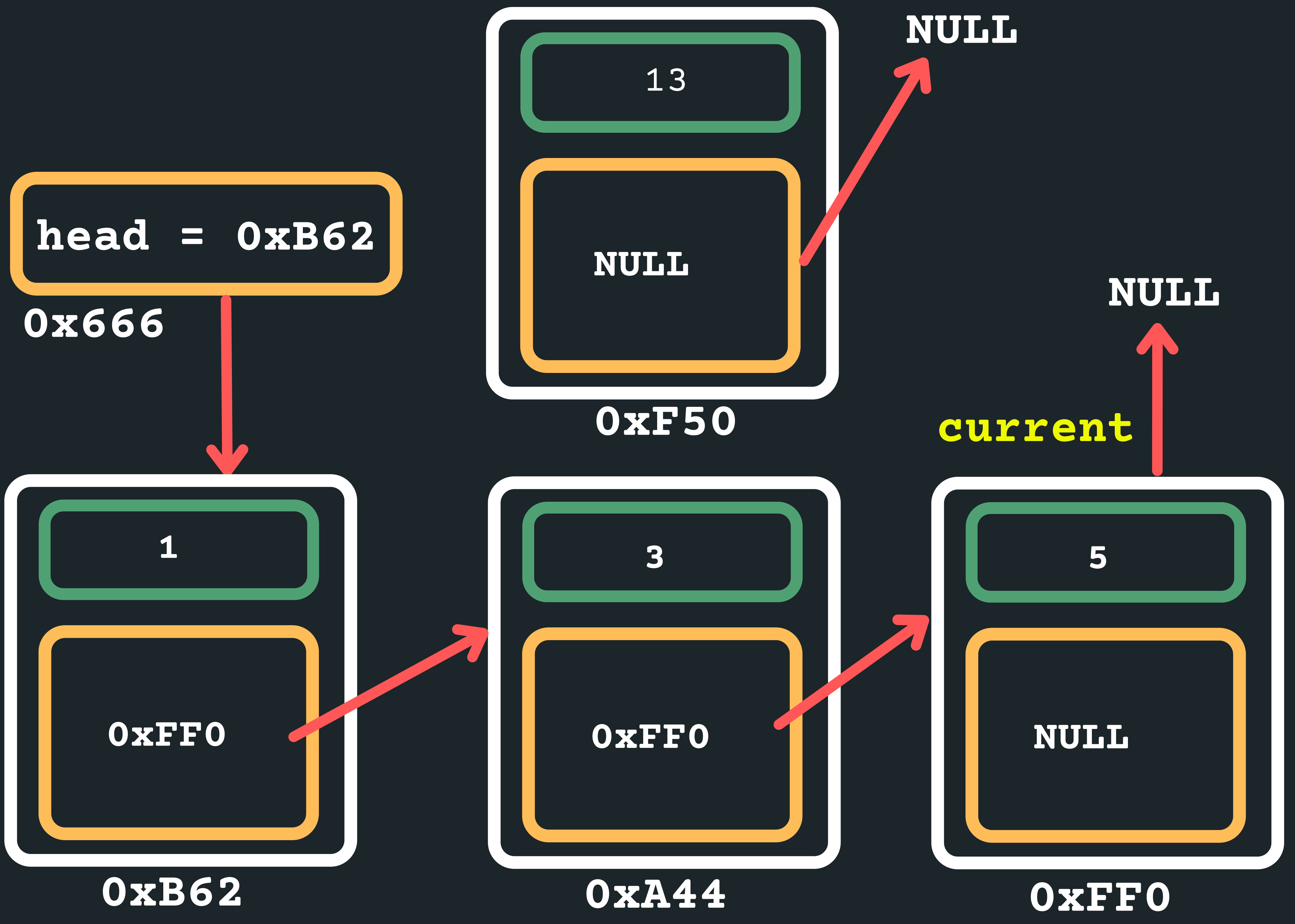
```
new_node->data = 13;
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Because this will be the last node point it to NULL

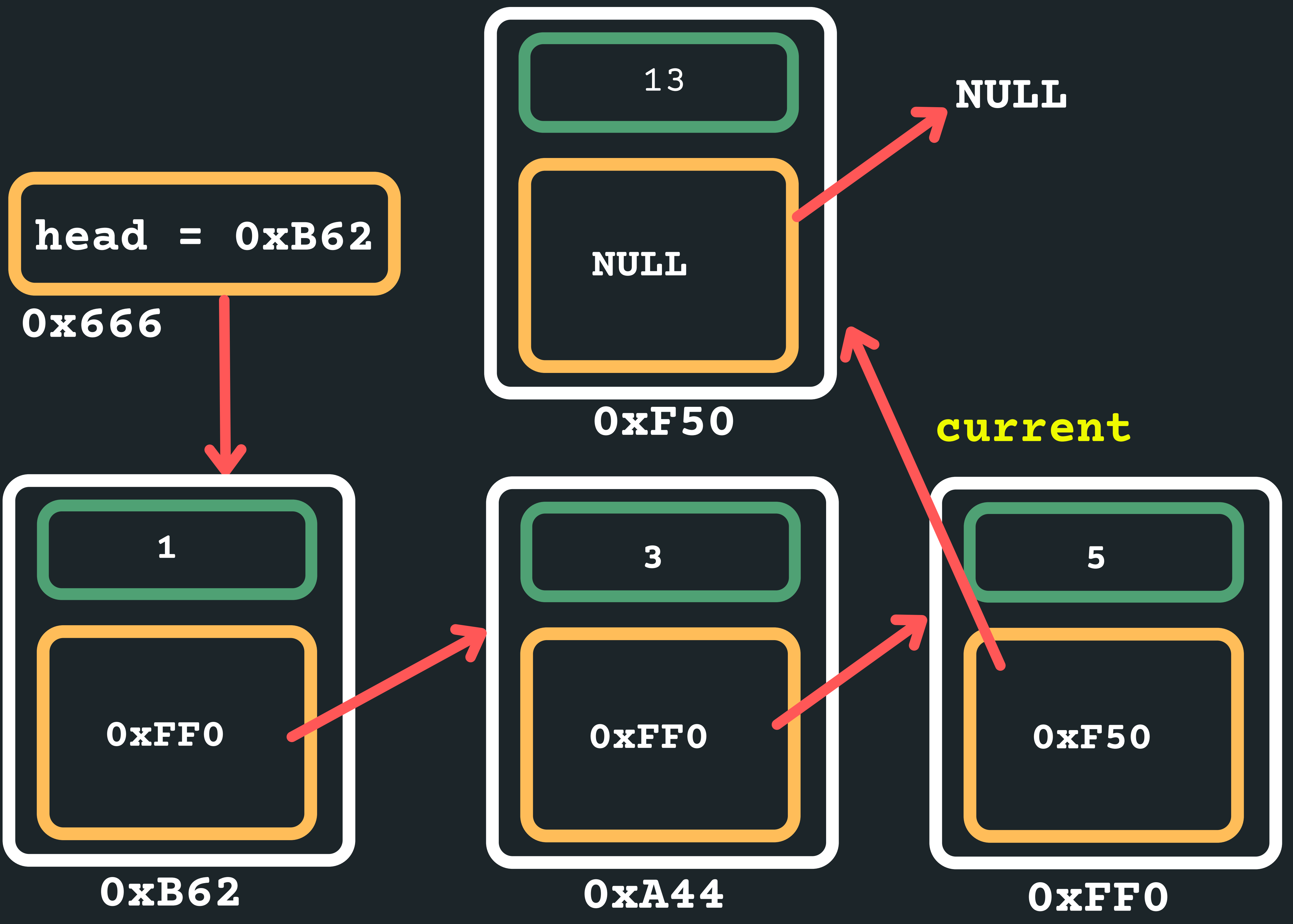
```
new_node->next = NULL;
```



# HOW CAN WE MOVE THROUGH THIS LIST TO FIND NEXT NODE?

Now point our current last node to the new node

```
current->next = new_node;
```



# 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://www.menti.com/al2ocecwm19g

# WHAT DID WE LEARN TODAY?

### LINKED LIST

What is it?

linked_list.c

### LINKED LIST

Insert at the head

linked_list.c

### LINKED LIST

Traverse a list

linked_list.c

### LINKED LIST

Insert at the tail

linked_list.c

# REACH OUT

## CONTENT RELATED QUESTIONS

Check out the forum

## ADMIN QUESTIONS

cs1511@unsw.edu.au