# COMP1511/1911 Programming Fundamentals

## Week 8 Lecture 1

# Linked Lists
# Inserting and Deleting

# Assignment 2 Live Stream: CS Dungeon

**Time:** Monday  4:30
**YouTube Link**
**Recording** will also be available

**Assignment Due Date:**
Friday Week 10 5pm

Don’t leave it until the last minute!
Help sessions will be very busy the week before the deadline!!!!!!!!

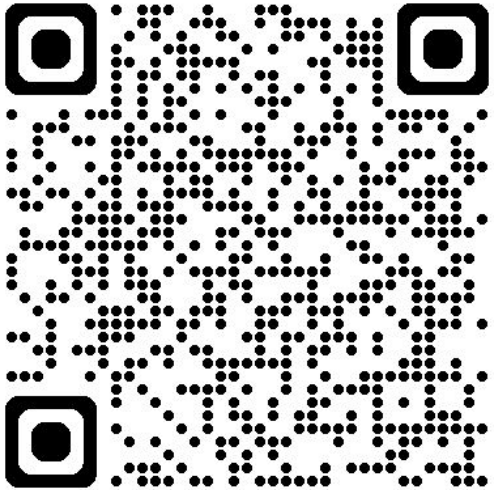
# Revision Sessions This Week

Wednesday 12pm - 2pm (K14 Physics LG18 Piano)

Friday 2pm - 4pm (Online on Microsoft Teams)

Please sign up for the revision sessions
by following this link
https://buytickets.at/comp1511unsw/1414648

Vote here: COMP1511/COMP1911 – Ed Discussion



# Last Lecture

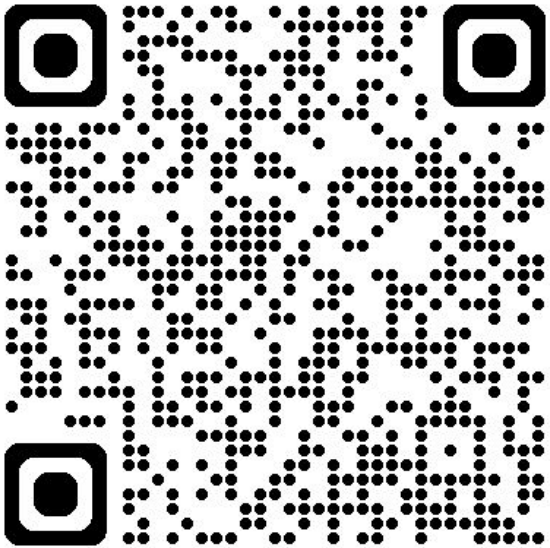
- Linked List Basics
- Creating nodes
- Printing a List
- Inserting nodes
  - At beginning

# Today's Lecture

- Recap:
  - List basics
  - Print List
- Insert at Tail
- Inserting in the middle of a list
  - Finding the length of the list
- Inserting anywhere in the list
- Linked list deletion of first node (maybe if there is time otherwise we will leave until thursday)

# Link to Week 8 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs 1511/24T3/live/week_8/

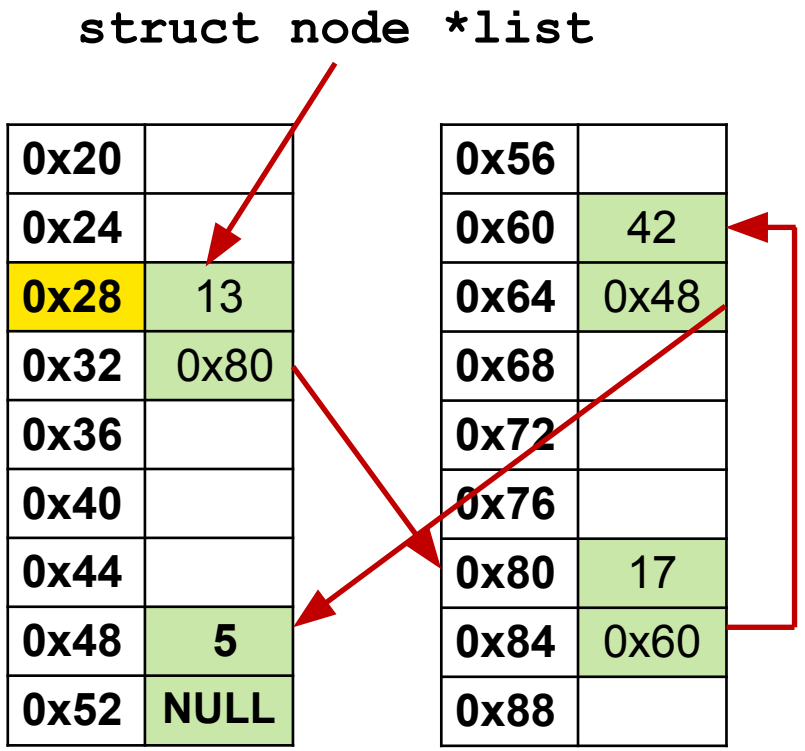


# Linked List Nodes

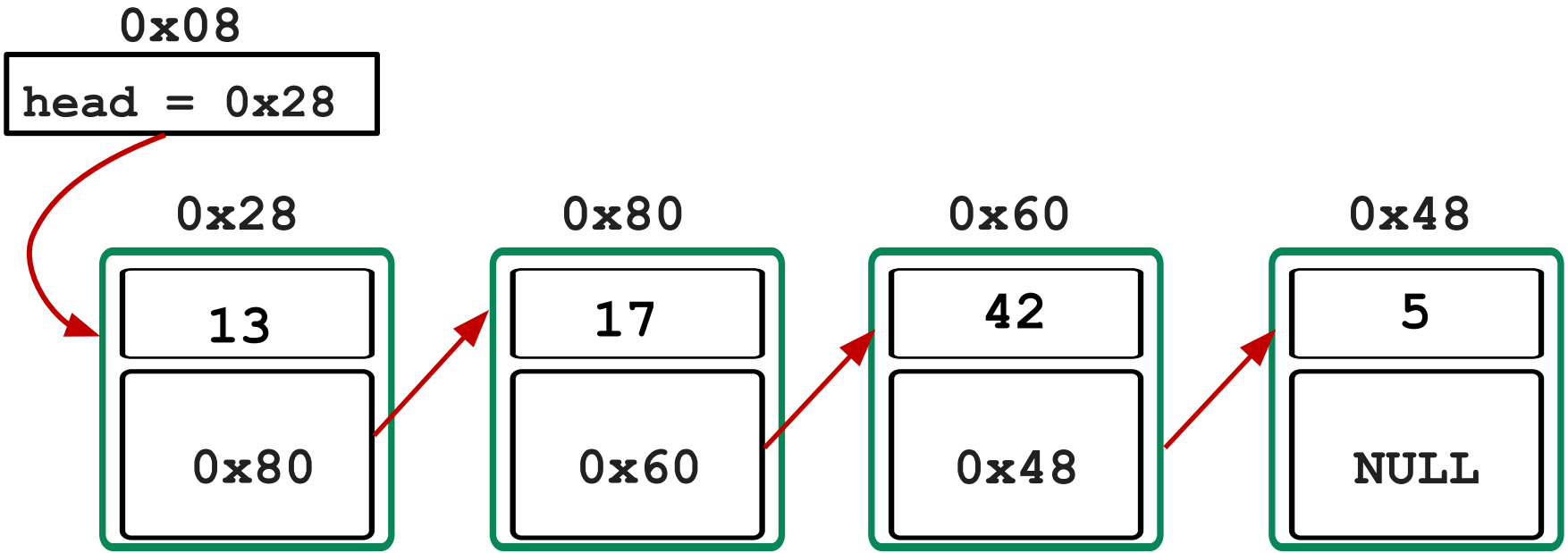
The list variable is a pointer to the first node in the list

```
struct node *list;
```

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

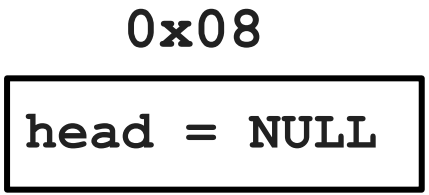


# Visualising Linked Lists



# An empty List in C

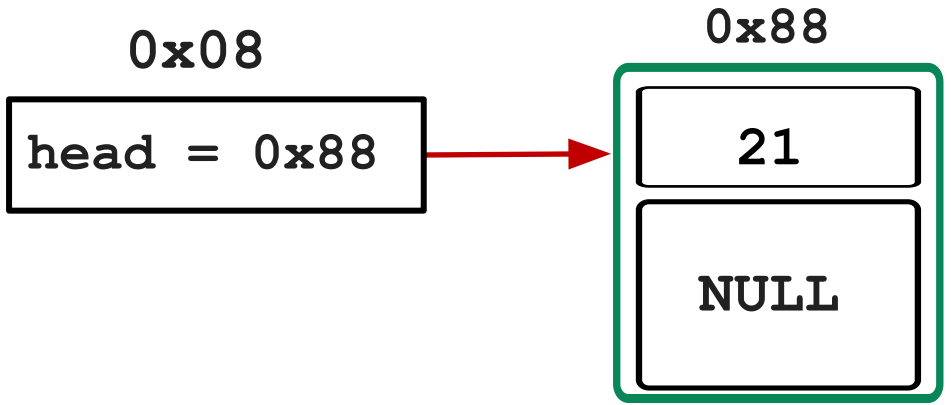
```
struct node {
    int data;
    struct node *next;
};
```




```
struct node *head = NULL;
```

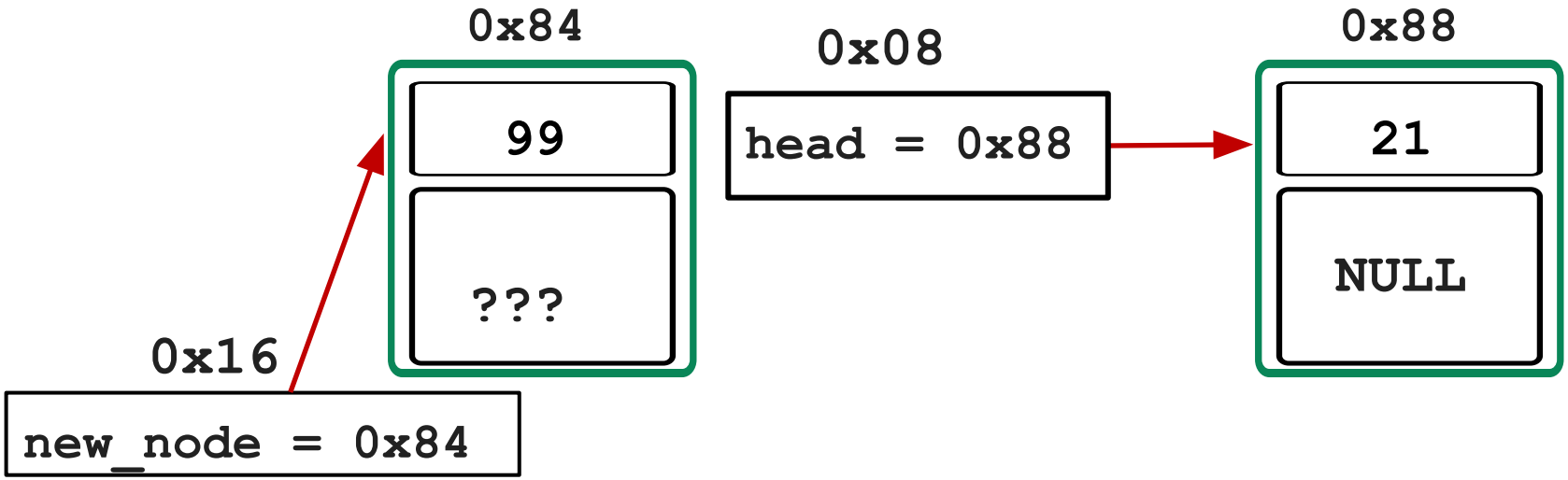
# Creating a List with 1 Node in C

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



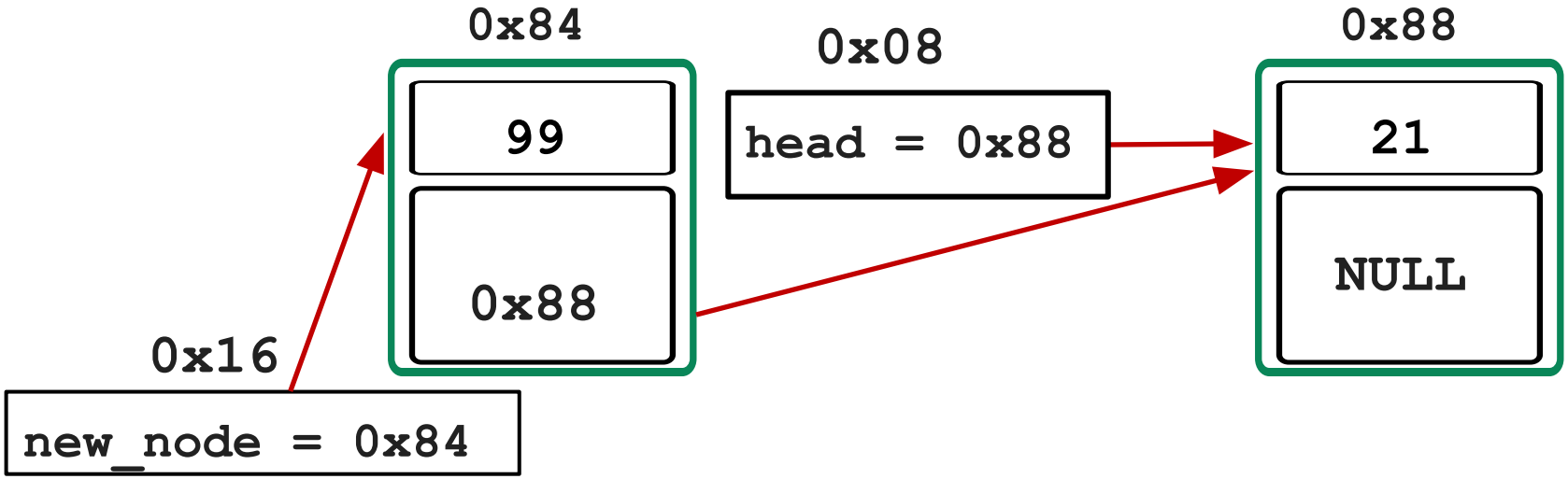

```c
struct node *head = NULL;
head = malloc(sizeof(struct node));
head->data = 21;
head->next = NULL;
```

# Connect 2 nodes: Add new node to the start



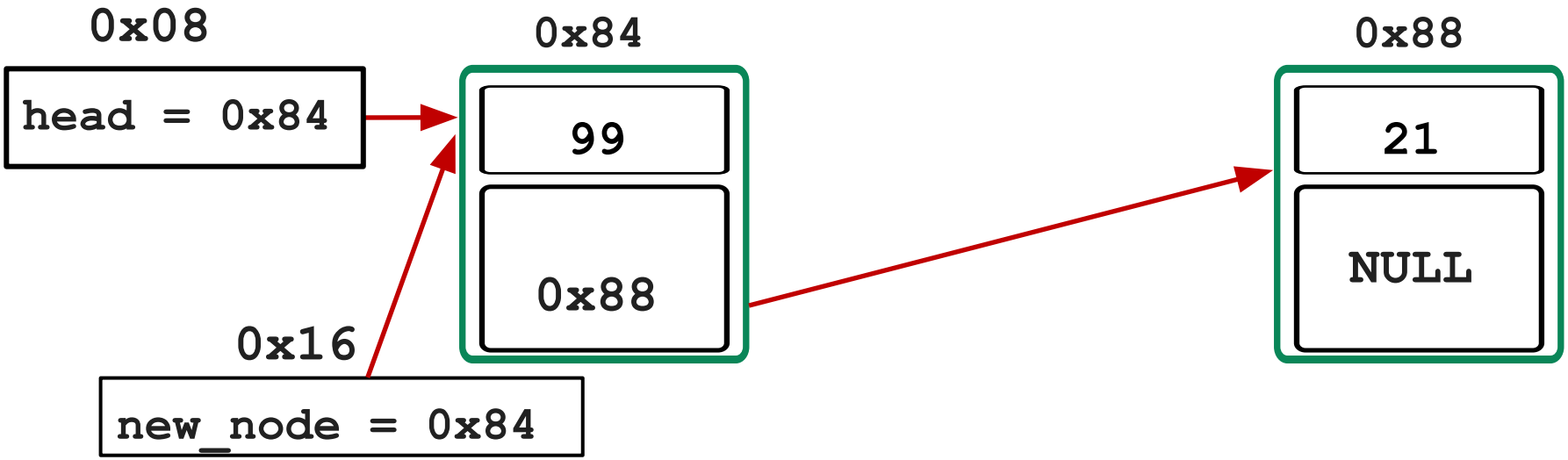

```
struct node *new_node = malloc(sizeof(struct node));
new_node->data = 99;
new_node->next = ???;
```

# Connect 2 nodes: Add new node to the start




```
struct node *new_node = malloc(sizeof(struct node));
new_node->data = 21;
new_node->next = head;
```

# Connect 2 nodes: Add new node to the start

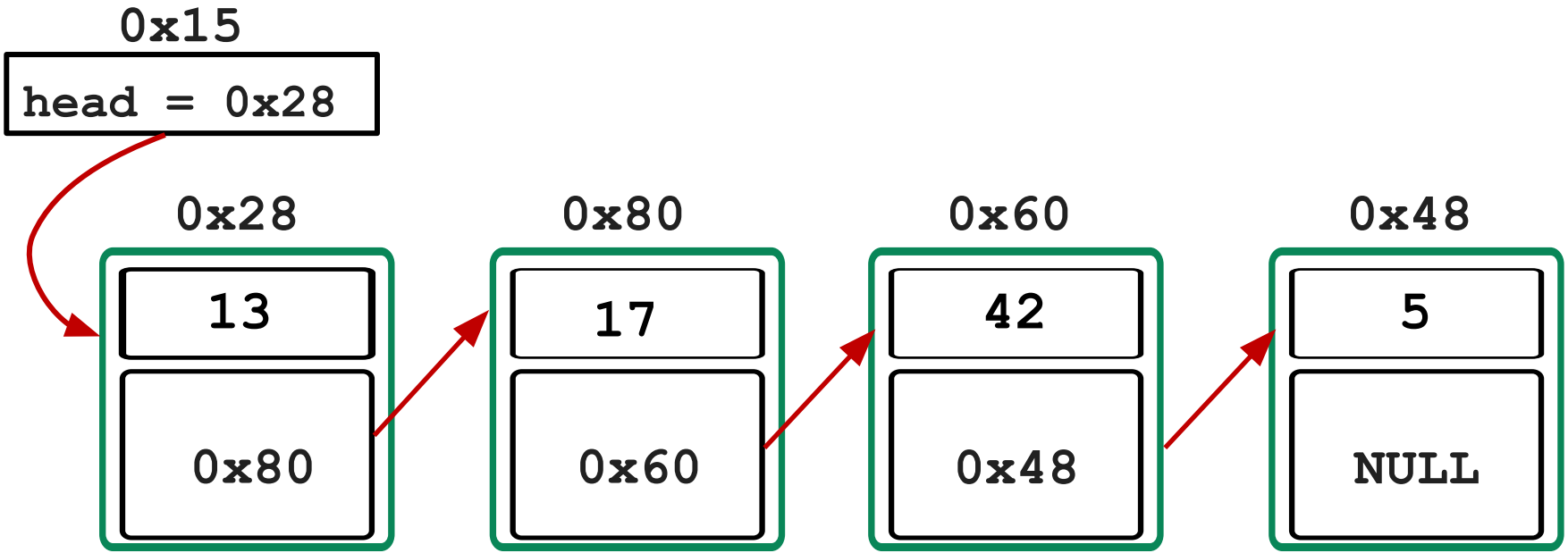



```
head = new_node;
```

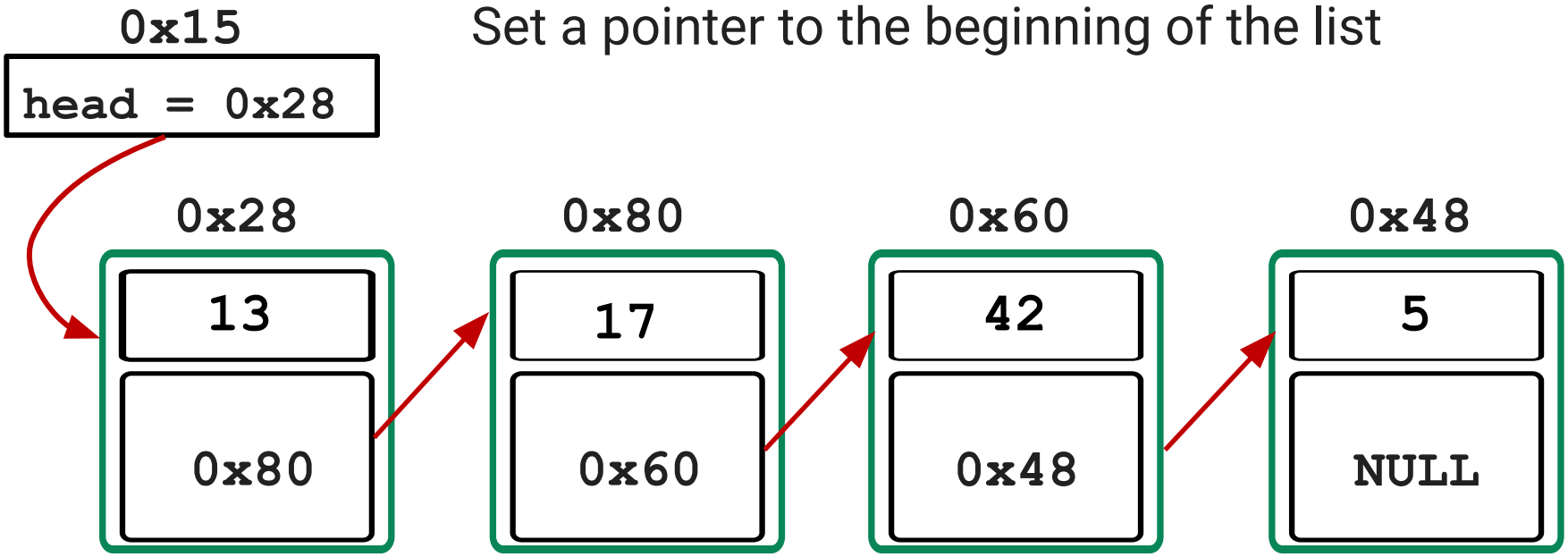
# Create Node Function

```
// Creates and returns a new node with given data and
// next pointer. returns NULL if memory allocation fails.
struct node *create_node(int data, struct node *next){
    struct node *new_node = malloc(sizeof(struct node));
    if (new_node == NULL) {
        return NULL;
    }
    new_node->data = data;
    new_node->next = next;
    return new_node;
}
```

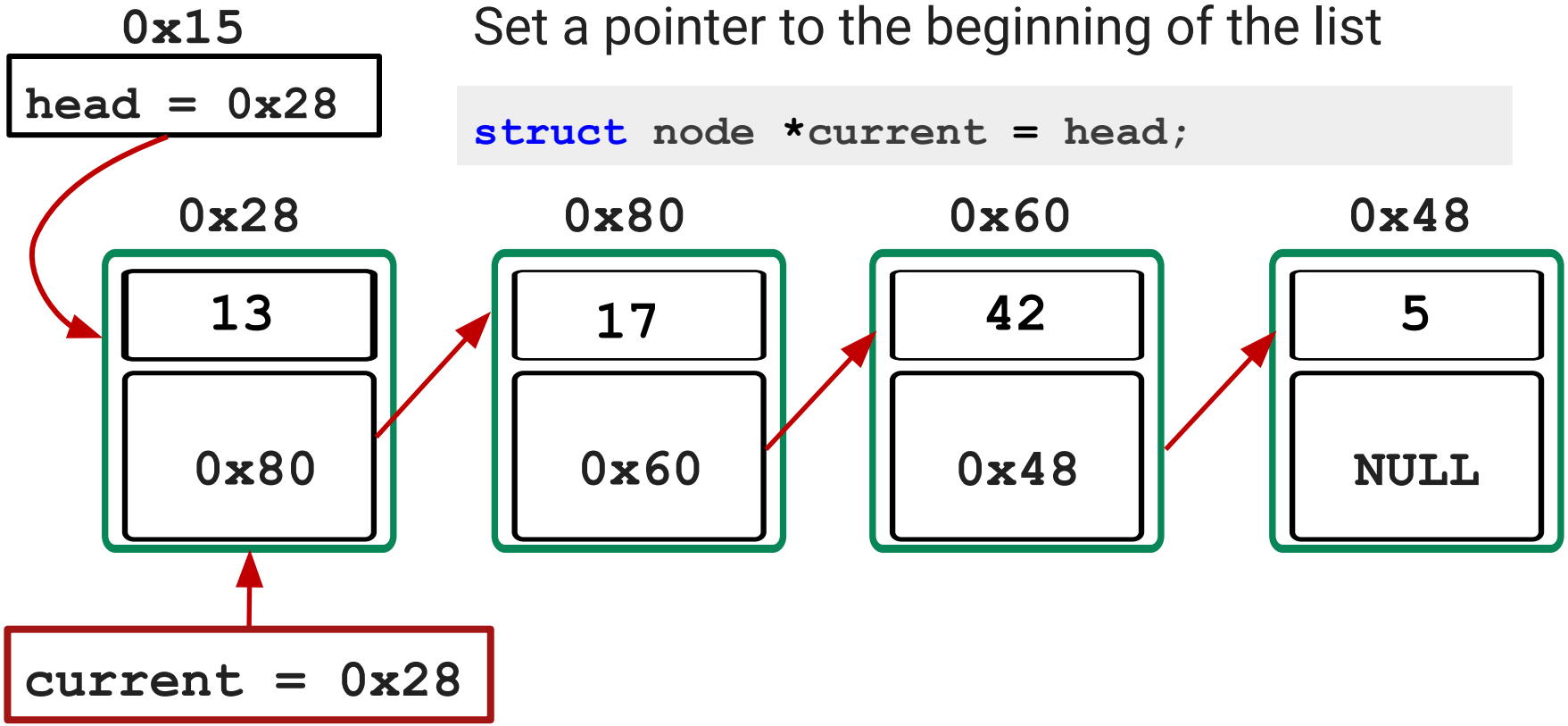
# Traversing a List



# Traversing a List



# Traversing a List

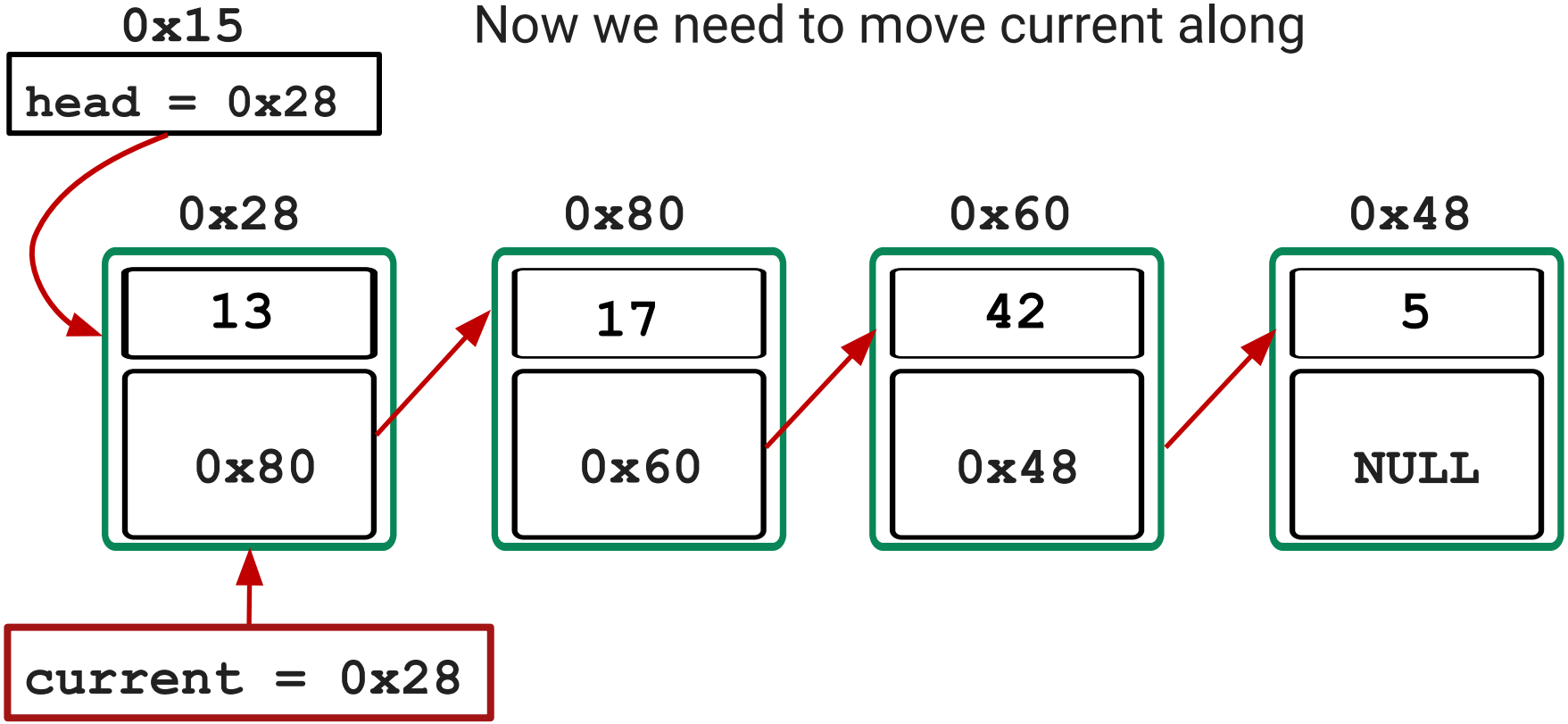



Set a pointer to the beginning of the list

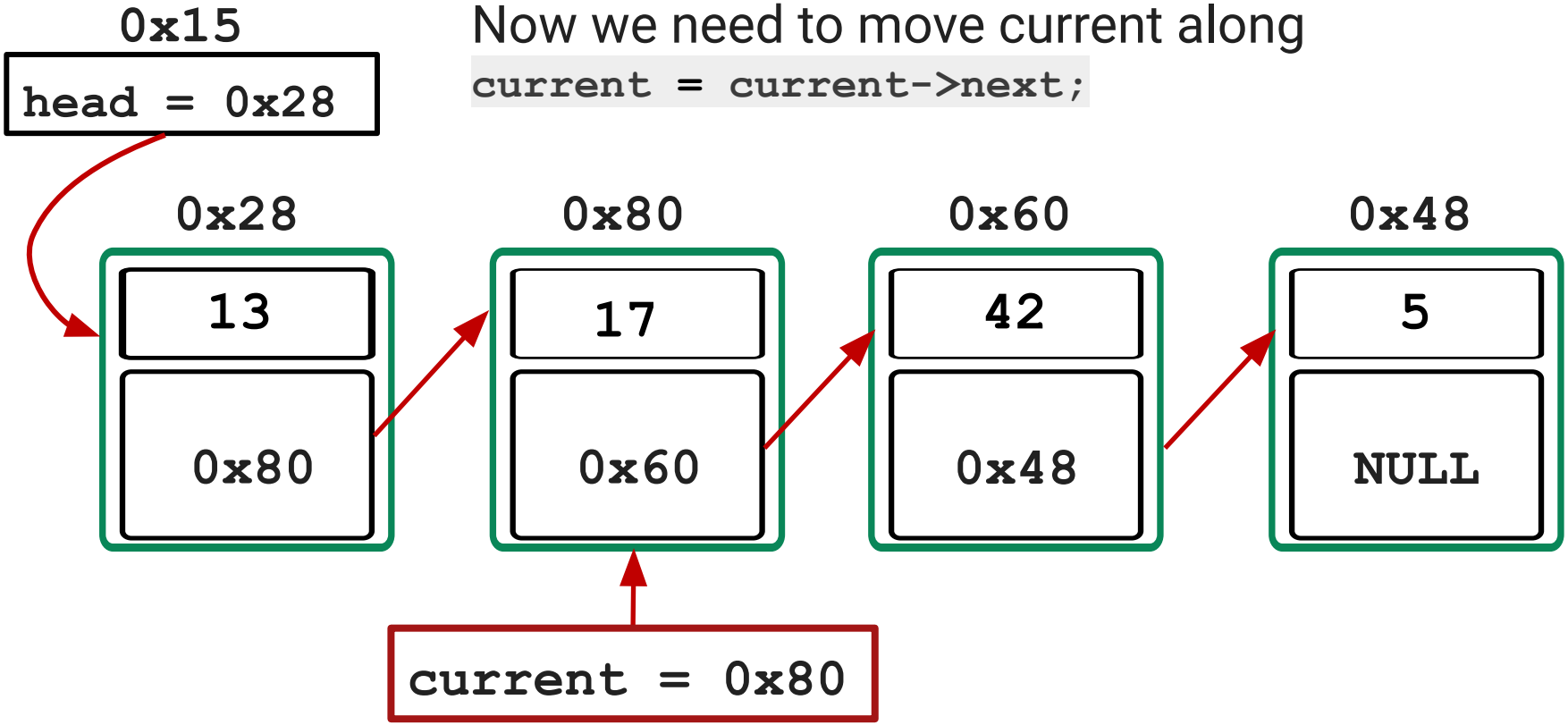
```
struct node *current = head;
```

# Traversing a List

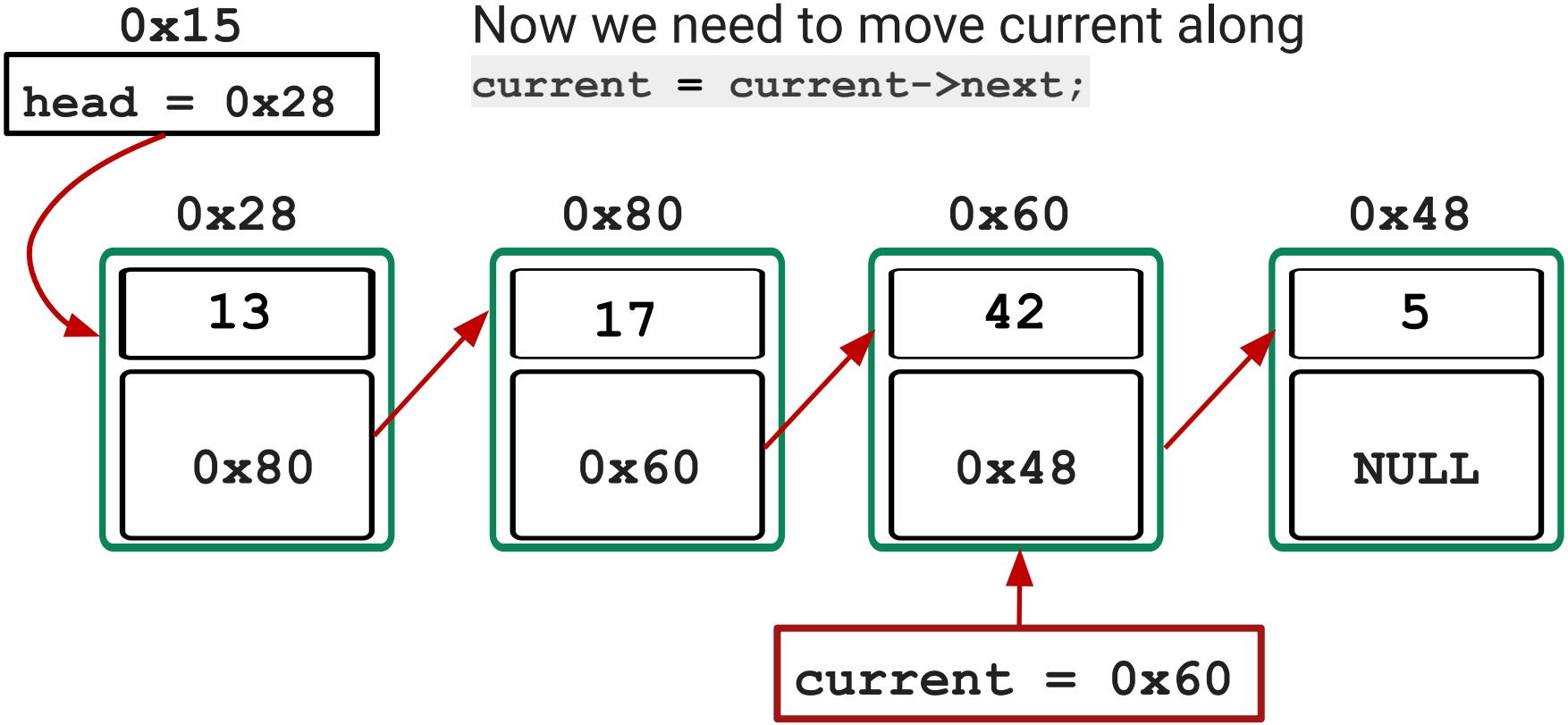
Now we need to move current along



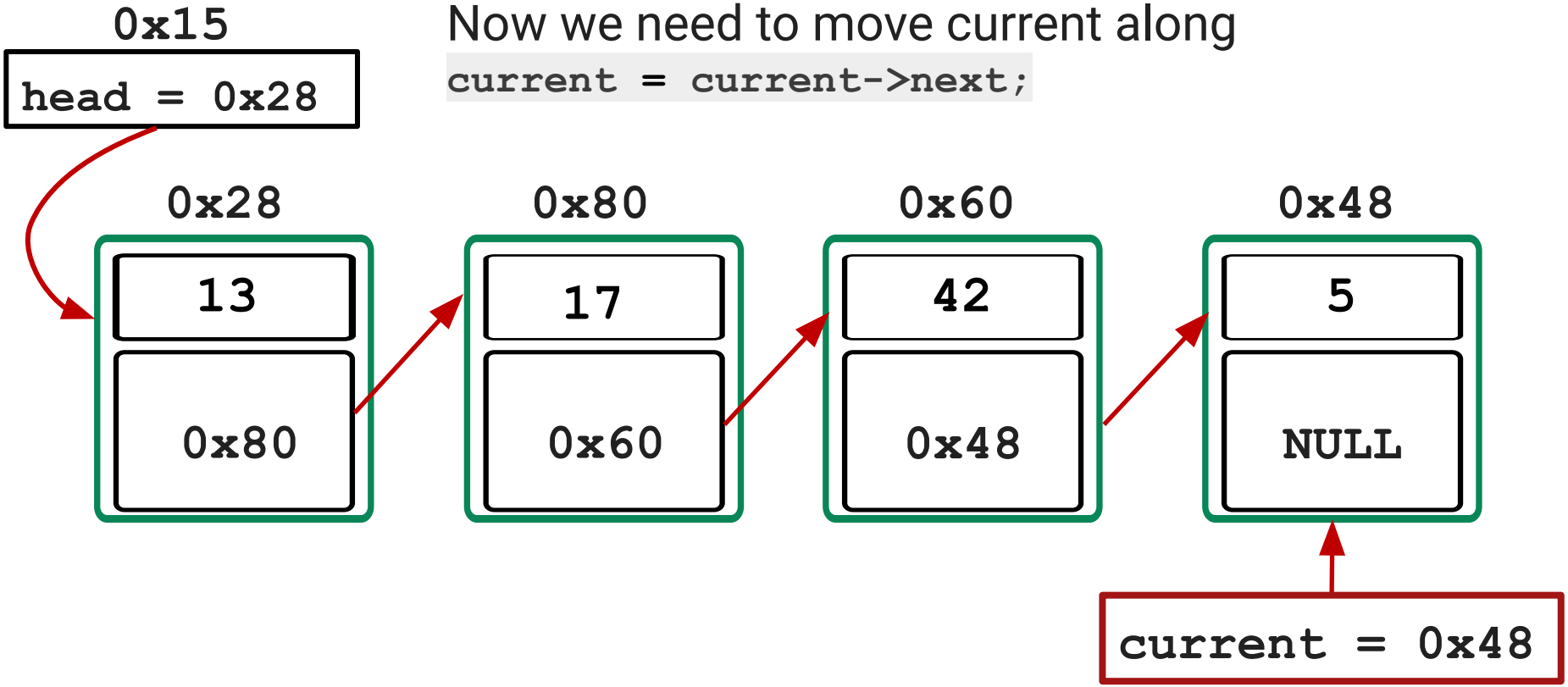
# Traversing a List



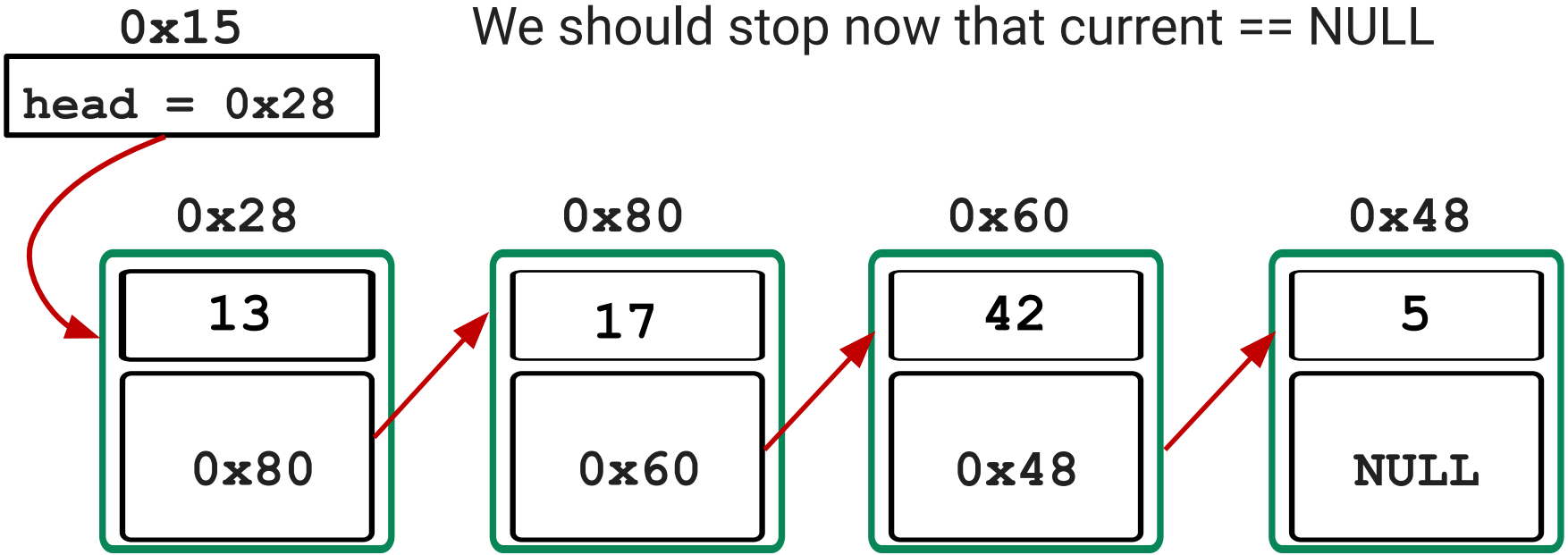
# Traversing a List



# Traversing a List



# Traversing a List



# Printing a list

```
// Traversing the list and printing the contents (data)
// from each node
void print_list(struct node *head) {
    struct node *current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

# Inserting at the tail (end) of a list

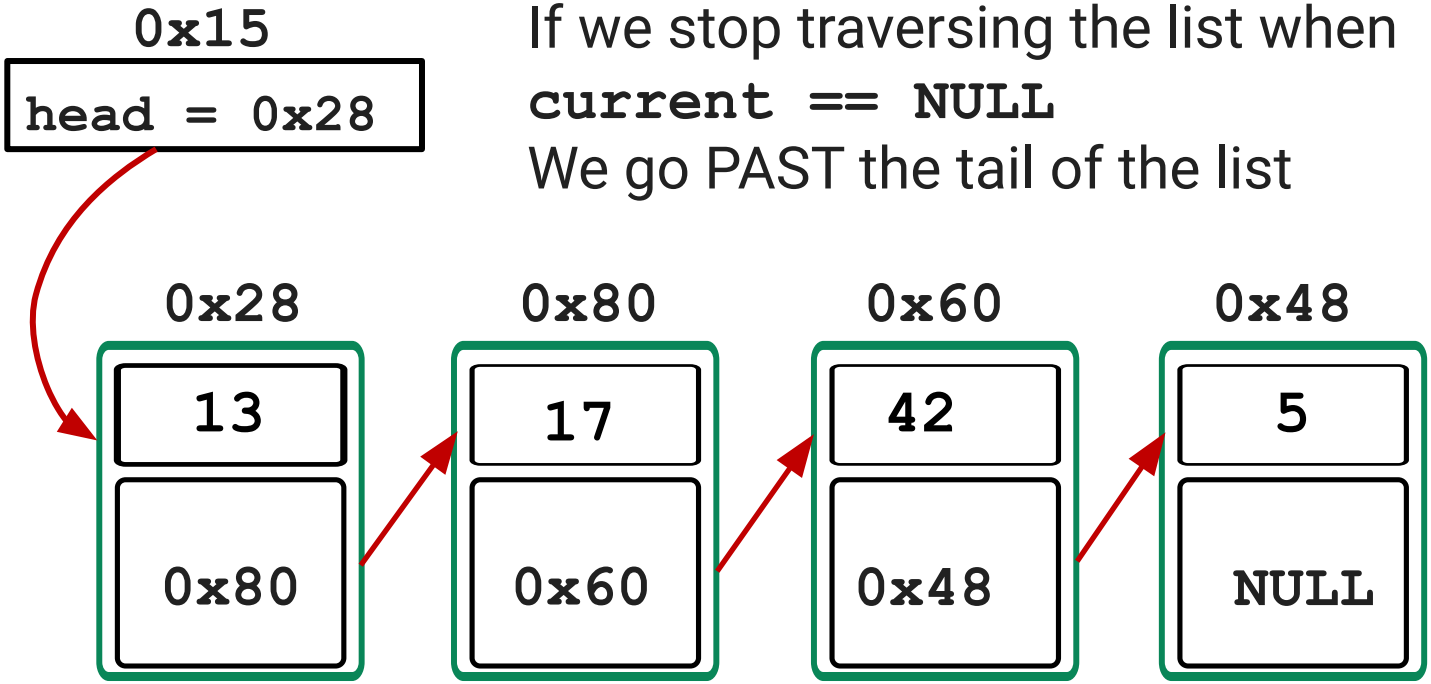
Where can I insert in a linked list?

- At the head (what we just did!)
- Between any two nodes that exist (later in this lecture!)
- After the tail as the last node (now!)

To insert a node at the end of the list we need to

- Find the last node in the list
- Connect the last node in the list to the new node

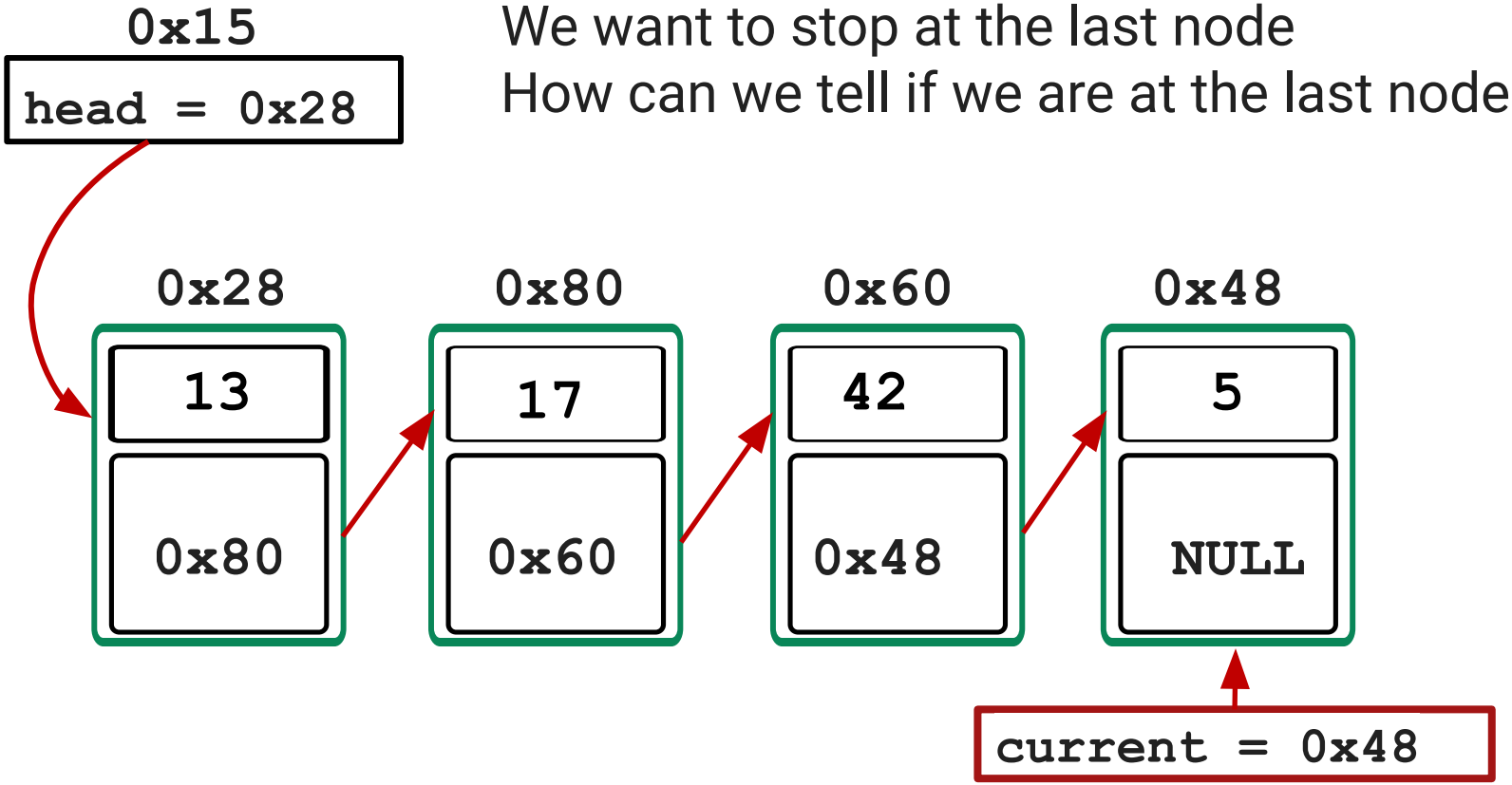
# Finding the Tail of the list




If we stop traversing the list when
`current == NULL`
We go PAST the tail of the list

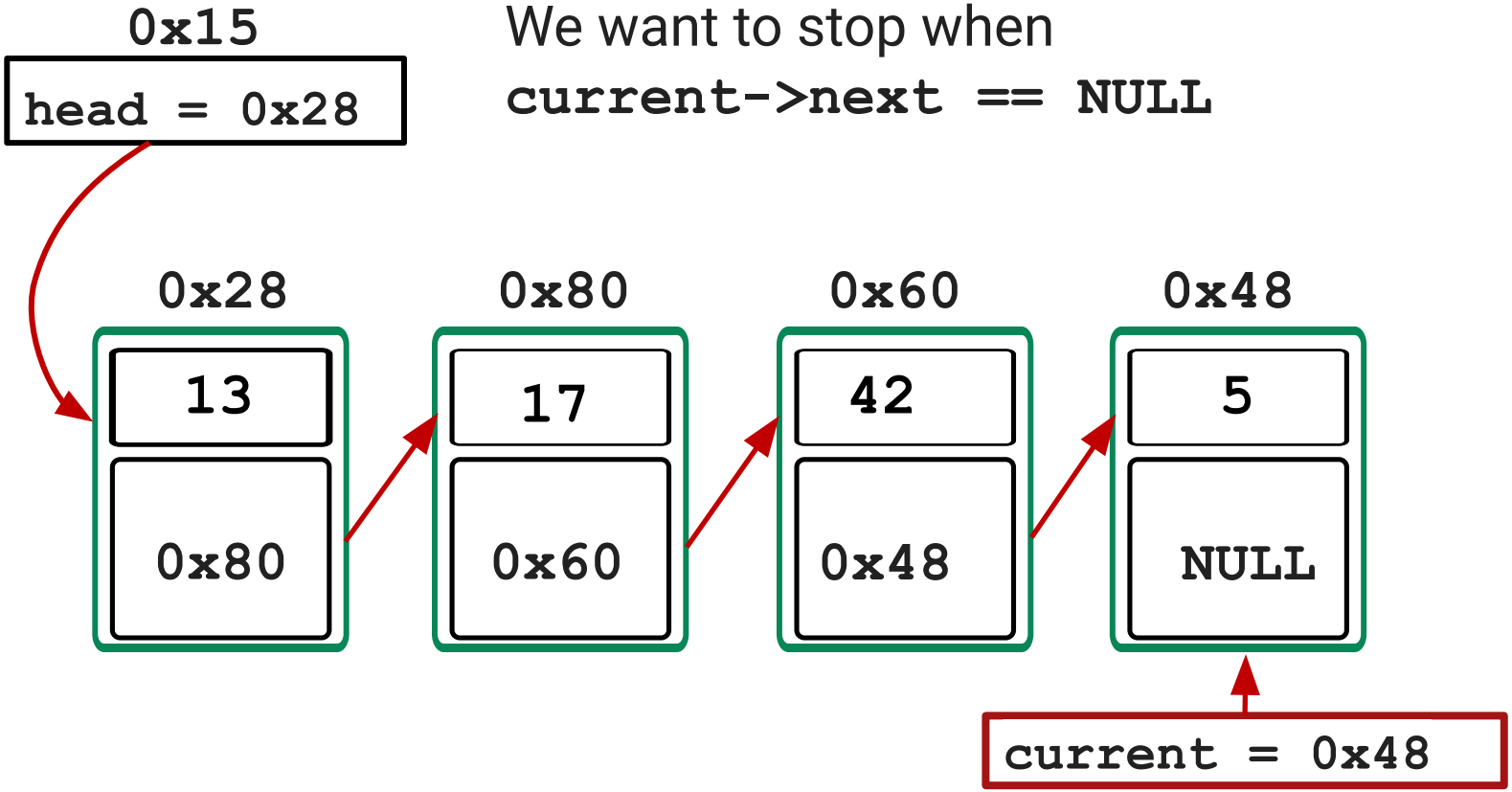
`current = NULL`

# Finding the Tail of the list

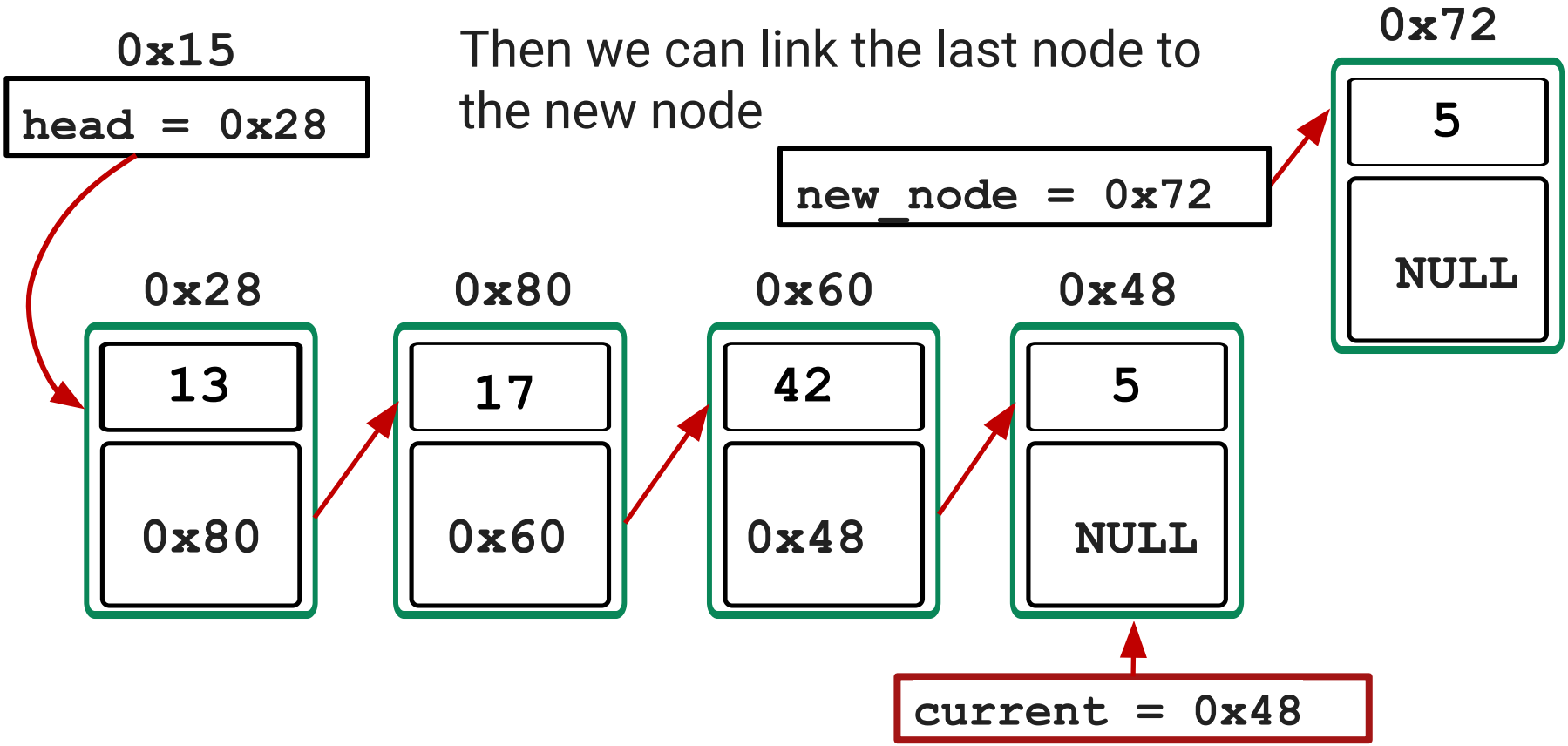



We want to stop at the last node
How can we tell if we are at the last node?

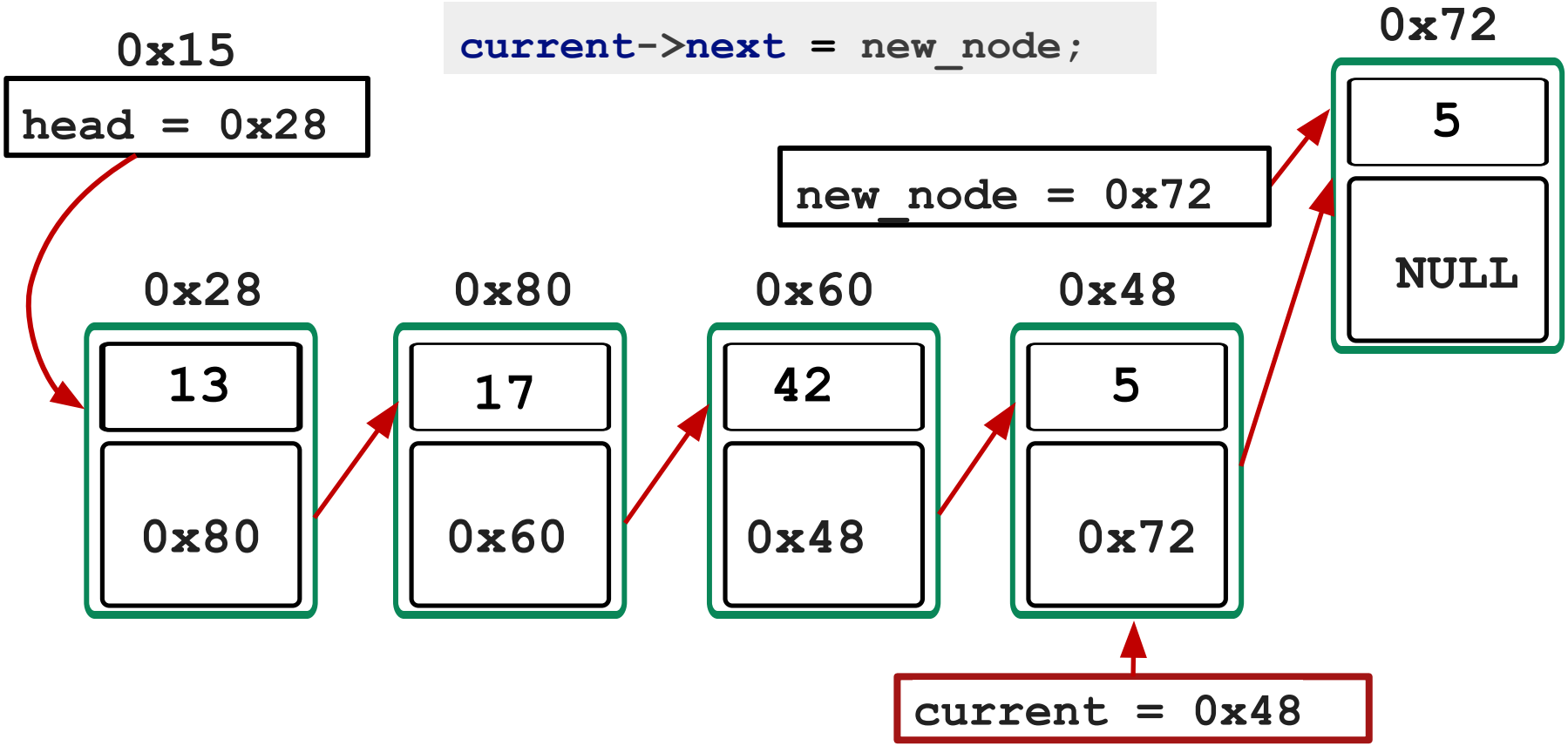
# Finding the Tail of the list



# Insert at the Tail of the list



# Insert at the Tail of the list



# Inserting at Tail (with a big bug)

```
// What valid input could cause this function to break?
void insert_at_tail(struct node *head, int data){
    struct node *current = head;
    // Find the tail of the list
    while (current->next != NULL) {
        current = current->next;
    }
    // Connect new node to the tail of the list
    struct node *new_node = create_node(data, NULL);
    current->next = new_node;
 }
```

# Linked List Test Cases

It is always important to test your linked list functions with:

- An empty list
- A list with one node
- A list with more than one node

Our function only inserts at the end of the list. If we were writing a function to insert anywhere into a list we would want to test

- Inserting at the beginning
- Inserting in the middle
- Inserting at the end

# Inserting At Tail Code Bug

If we have an empty list

- head == NULL;
- so then current == NULL;
- so `current->next`
  will be dereferencing a NULL pointer and result in a run time error

```
void insert_at_tail(struct node *head, int data){
    struct node *current = head;
    // Find the tail of the list
    while (current->next != NULL) {
```

# Inserting at Tail (still with a bug)

```
void insert_at_tail(struct node *head, int data){
    struct node *new_node = create_node(data, NULL);
    if (head == NULL) { // Special case for empty list
        head = new_node;
    } else {
        struct node *current = head;
        // Find the tail of the list
        while (current->next != NULL) {
            current = current->next;
        }
        // Connect new node to the tail of the list
        current->next = new_node;
    }
 }
```

# Inserting At Tail Code Bug

The code no longer crashes!!!

But we still end up with an empty list when we use the function.

Why?

```
int main(void) {
    struct node *head = NULL;
    insert_at_tail(head, 9);
    // local variable head is in main is still NULL
    return 0;
}
```

# Fixing Inserting at Tail Code

We need to modify the prototype so it can return the head of the list and we need to assign that return value to our local variable.

```
struct node *insert_at_tail(struct node *head, int data);
int main(void) {
    struct node *head = NULL;
   // local variable head has been updated :)
    head = insert_at_tail(head, 9);
    return 0;
 }
```

# Inserting at Tail

```
struct node *insert_at_tail(struct node *head, int data){
    struct node *new_node = create_node(data, NULL);
    if (head == NULL) { // Special case for empty list
        head = new_node;
    } else {
        struct node *current = head;
        // Find the tail of the list
        while (current->next != NULL) {
            current = current->next;
        }
        // Connect new node to the tail of the list
        current->next = new_node;
    }
    return head;
 }
```

# Inserting Into a Linked List

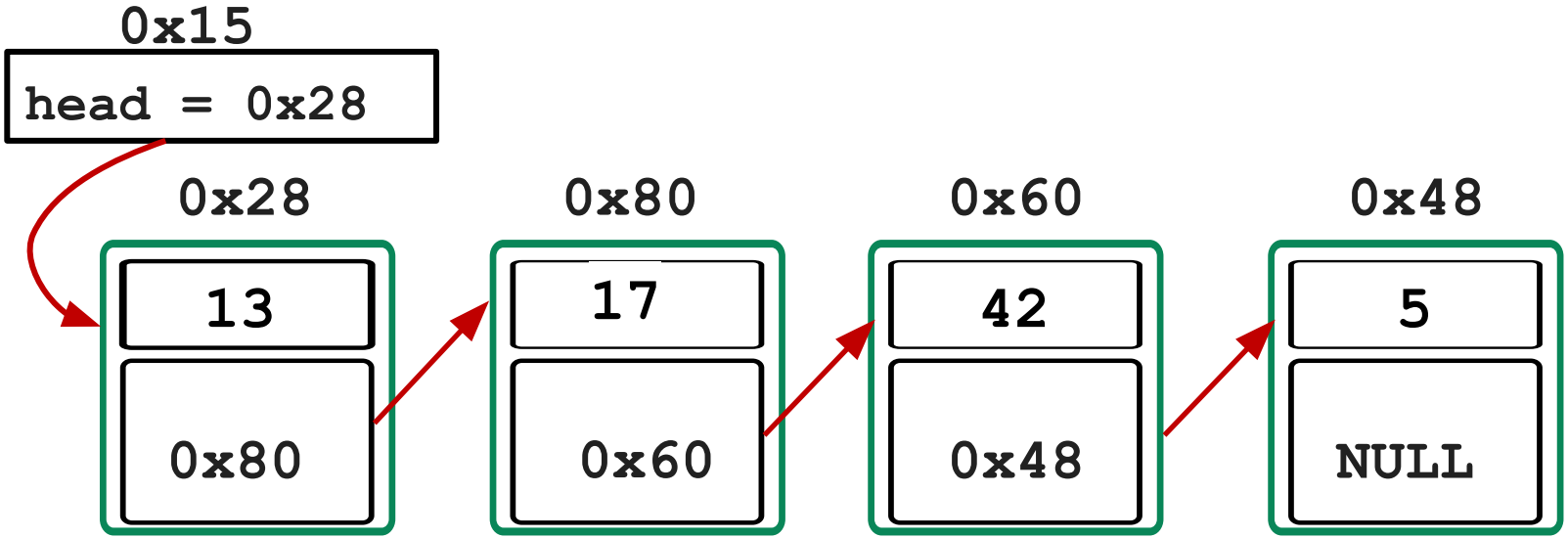
We have looked at 2 special cases

- Inserting at the beginning of a list
- Inserting at the end of a list

Now we want to be able to insert anywhere. Lets try right in the middle of the list!

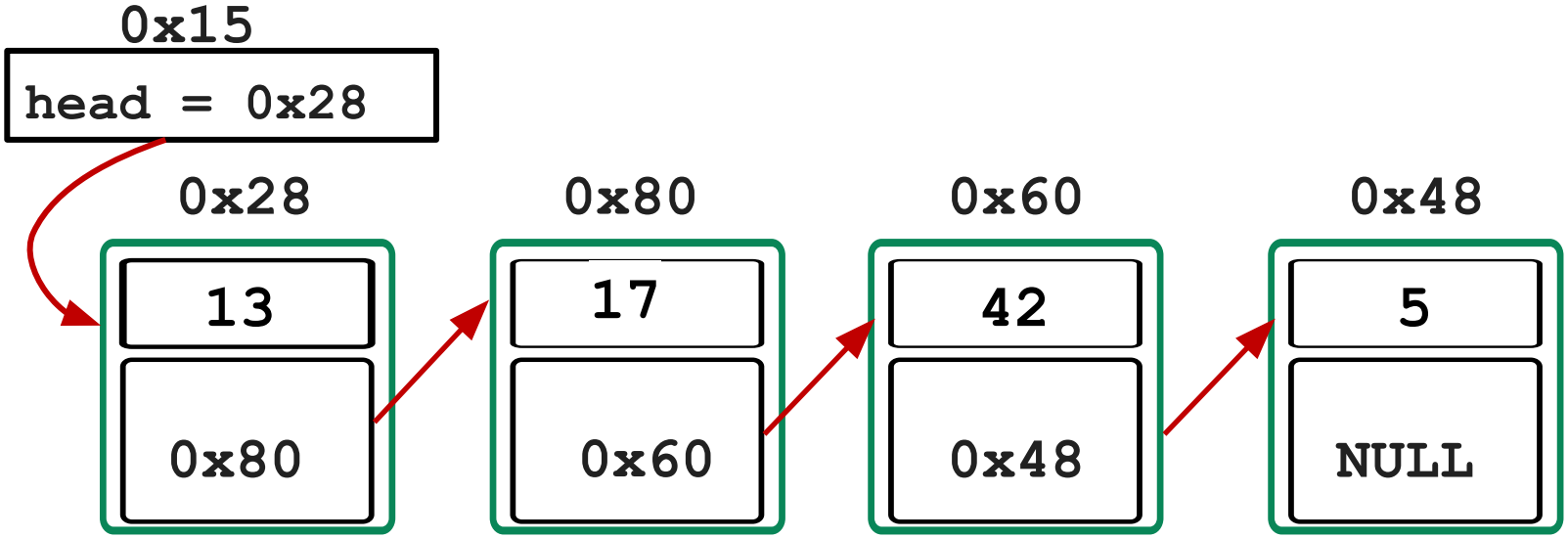
# Inserting in the Middle of the List

We want to insert a new node at position list_size/2, assuming positions start at 0. In this case that is position 2



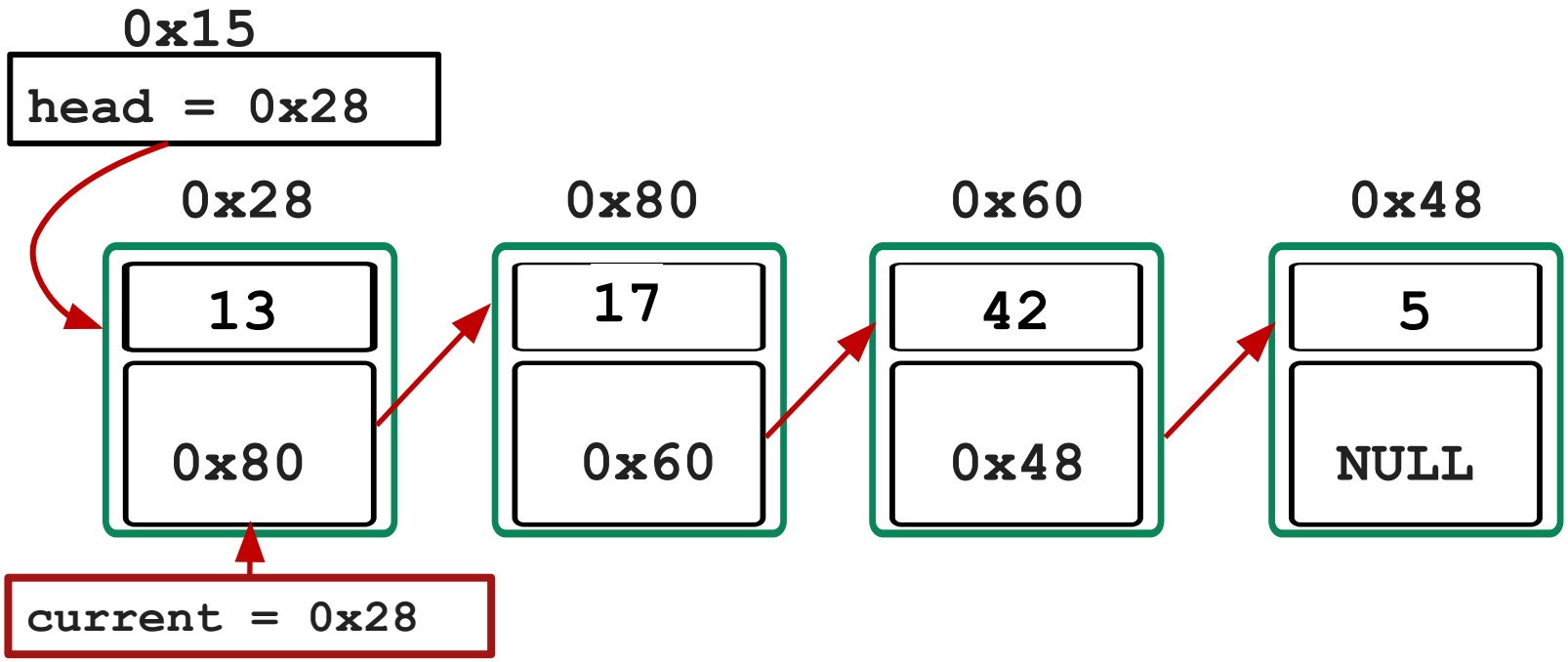
# Inserting in the Middle of the List

Use a counter and stop traversing when we get to the node **before** the position we want to insert at (size/2 - 1). In this case position 1.



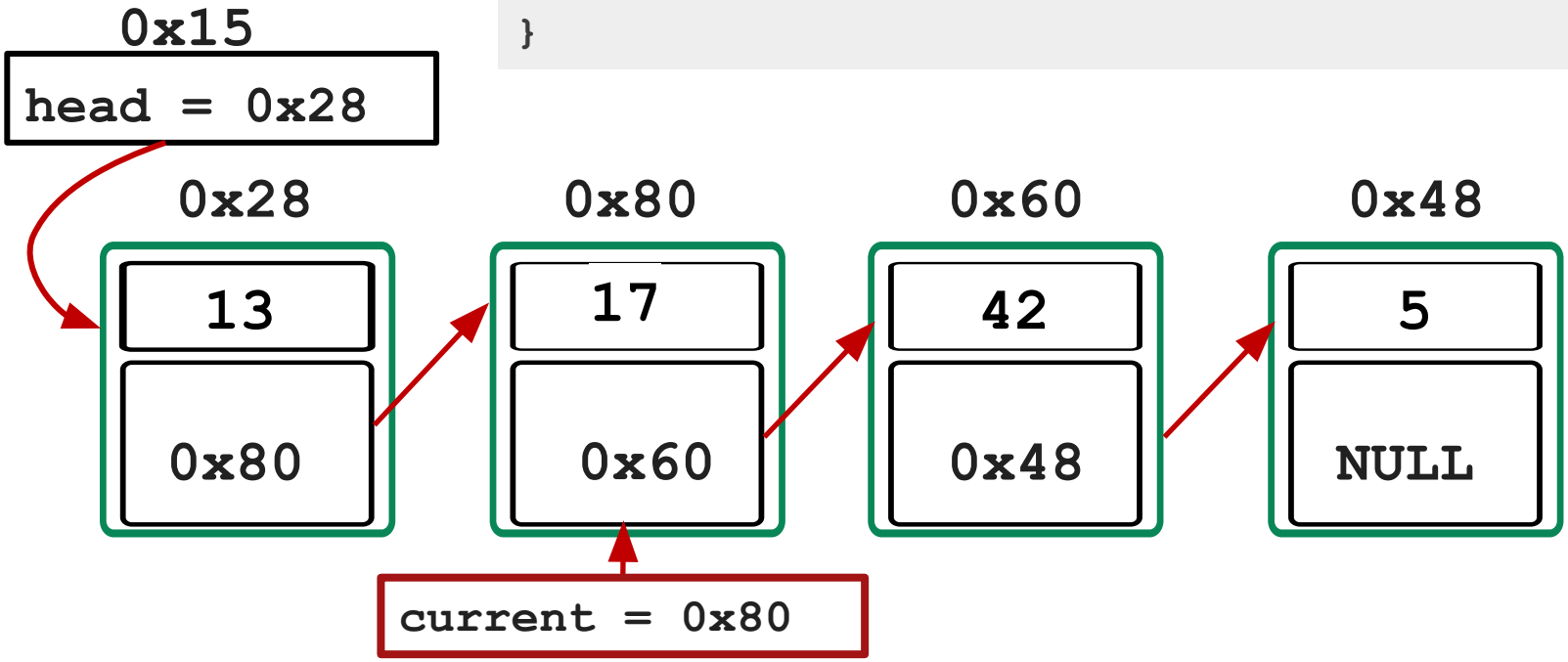
# Inserting in the Middle of the List

```
struct node *current = head;
int counter = 0;
```



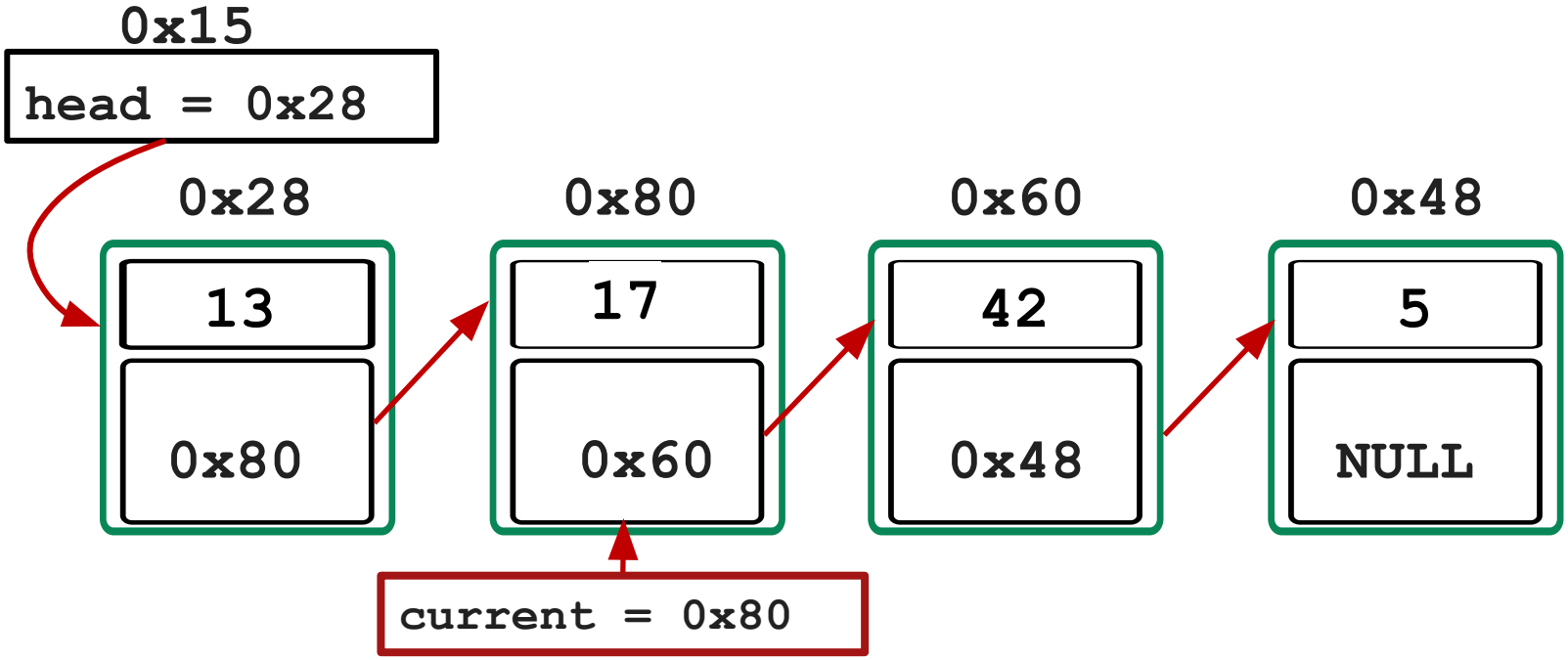
# Inserting in the Middle of the List

```
while (counter < size/2 - 1) {
    current = current->next;
    counter++;
}
```

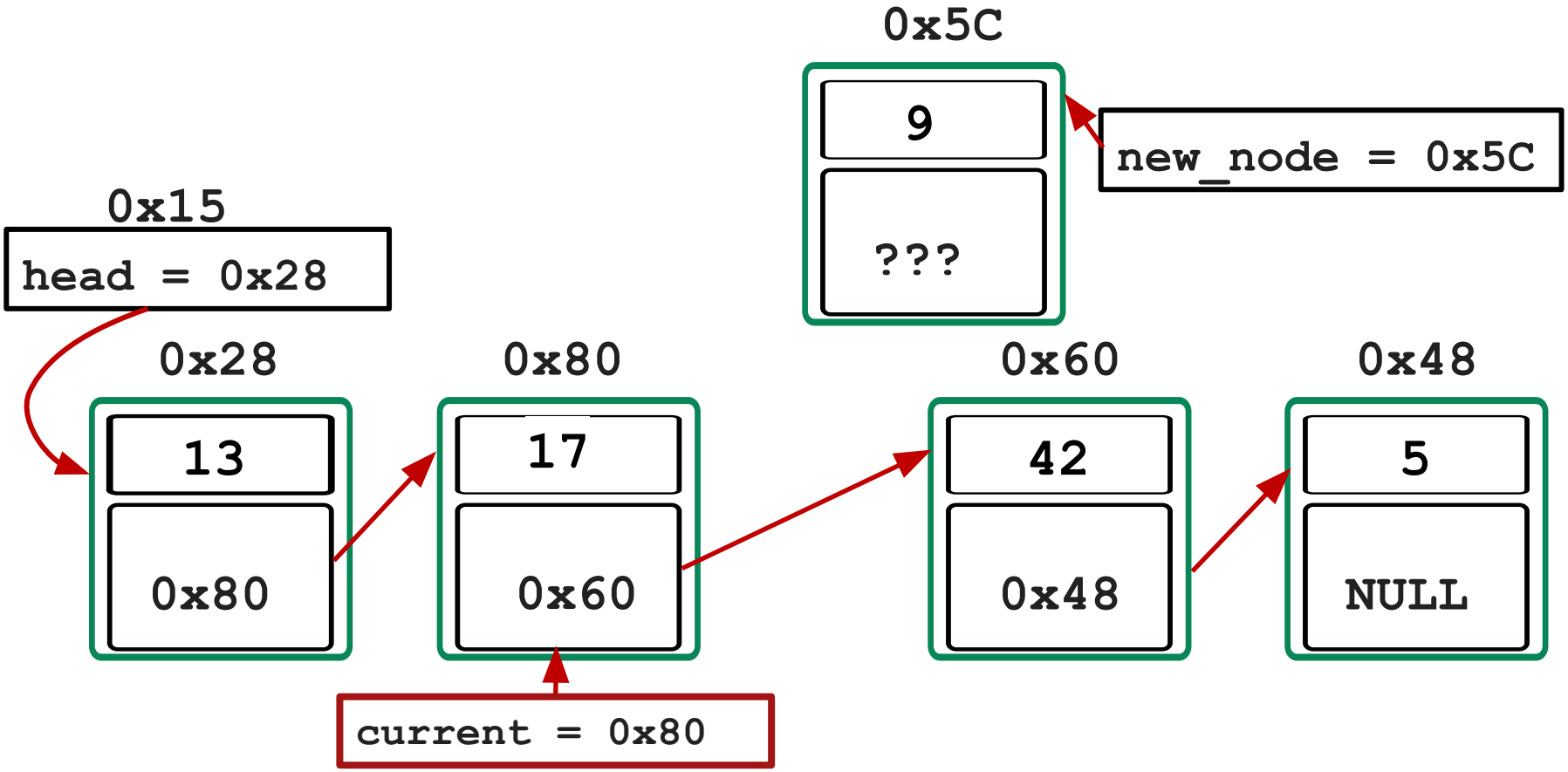


# Inserting in the Middle of the List

Now we want to connect our new node. It should come after the current node, but before current->next

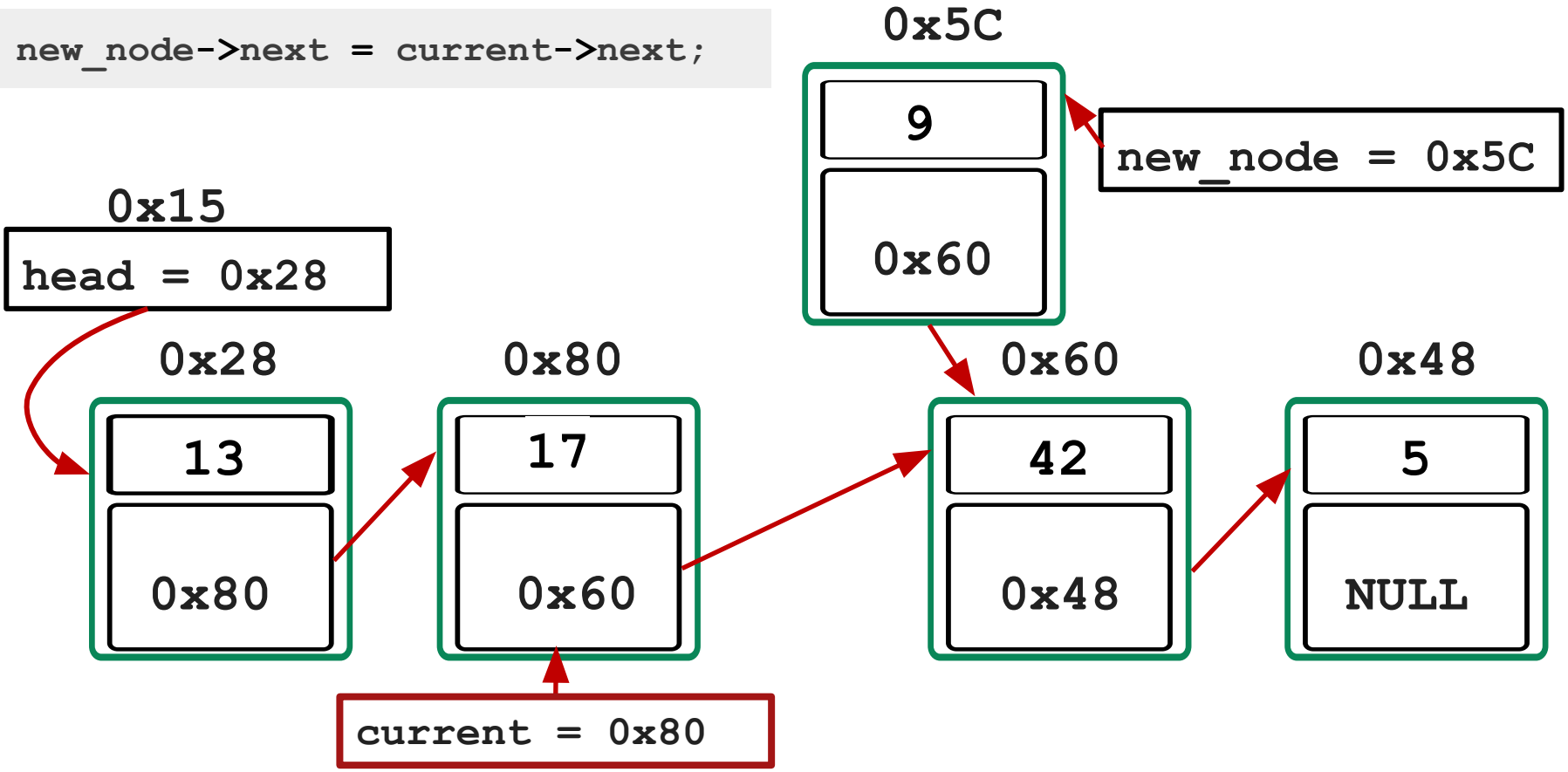


# Inserting in the Middle of the List



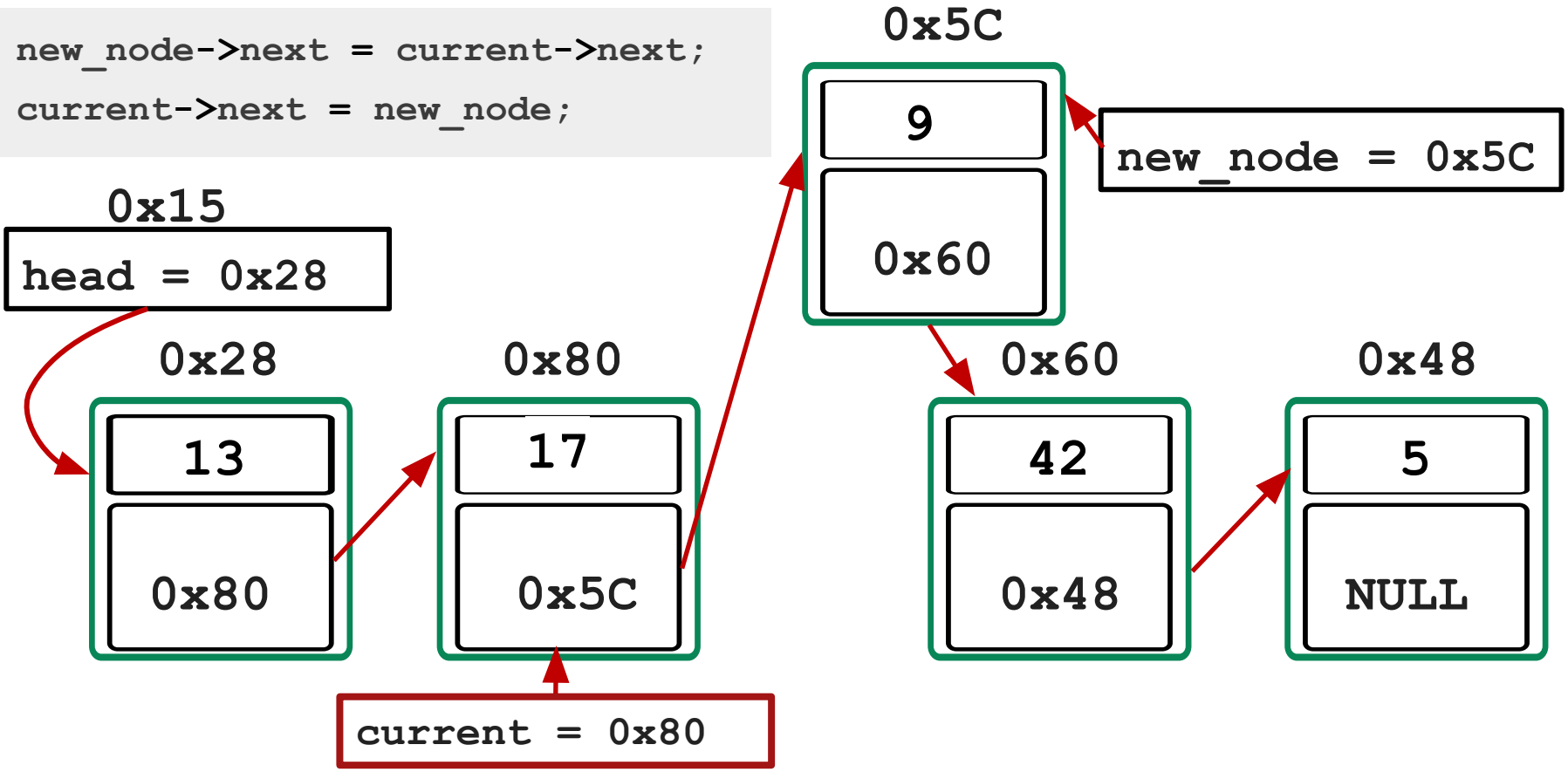
# Inserting in the Middle of the List

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



# Inserting in the Middle of the List

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



# Coding: Inserting in the Middle of the List

- What conditions will break this?
  - What happens if it is an empty list?
  - What happens if there is only 1 item in the list?
  - Anything else we should check?
- How can we modify our code to handle any of these situations that break it?
- How could we modify our code to write a function to insert at any given index?
  - What extra cases do we need to check now?

# Coding: Inserting at any position in List

- How could we modify our code to write a function to insert at any given index?
  - What extra cases do we need to check now?

# Inserting Into a Linked List Test Cases

- Remember, you should always consider and make sure your solution works:
    - Inserting into an empty list
    - Inserting at the head of the list
    - Inserting after the first node if there is only one node
    - Inserting somewhere in the middle
    - Inserting at the end of the list

Tip: Draw a diagram!!!! It will allow you to easily see what are some potential pitfalls

# Deletion

# Deleting the First Node in a Linked List

Let's write a function to delete the first node in a linked list.

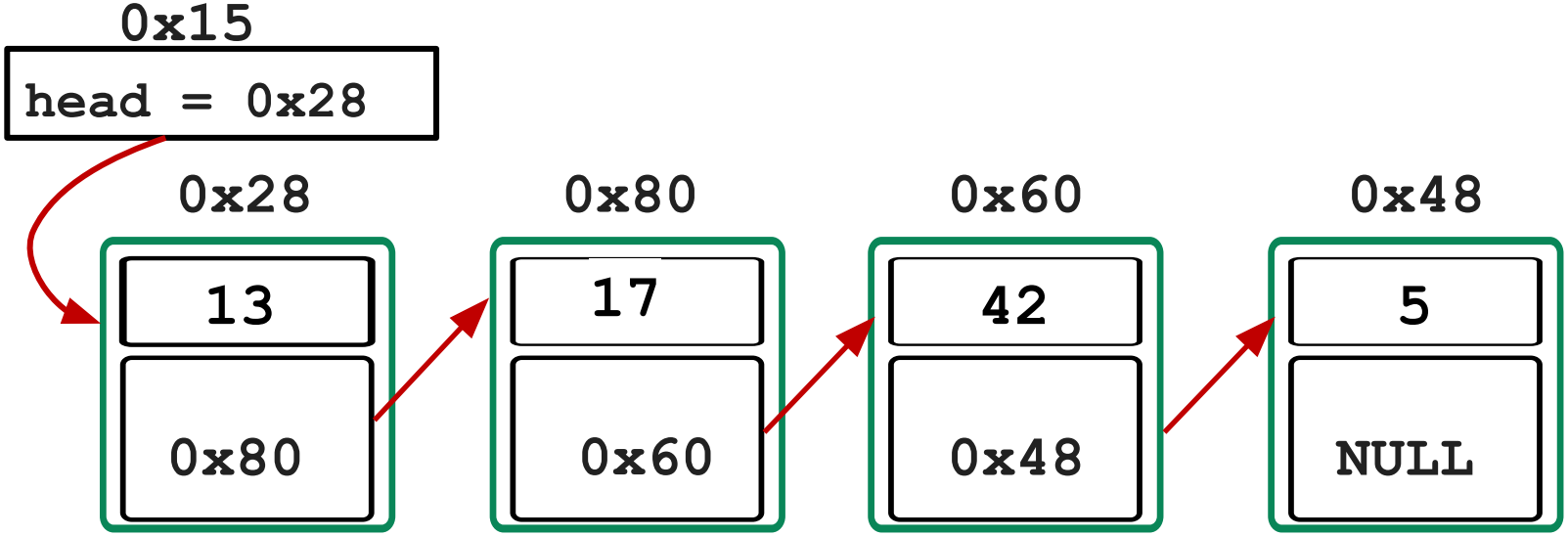
We need to consider the case when the list is empty

- If it is empty we can't delete anything
- We just return the head of the list which would be  NULL

```
if (head == NULL) {
    return head; //or return NULL;
}
```

# Deleting the First Node in a Linked List

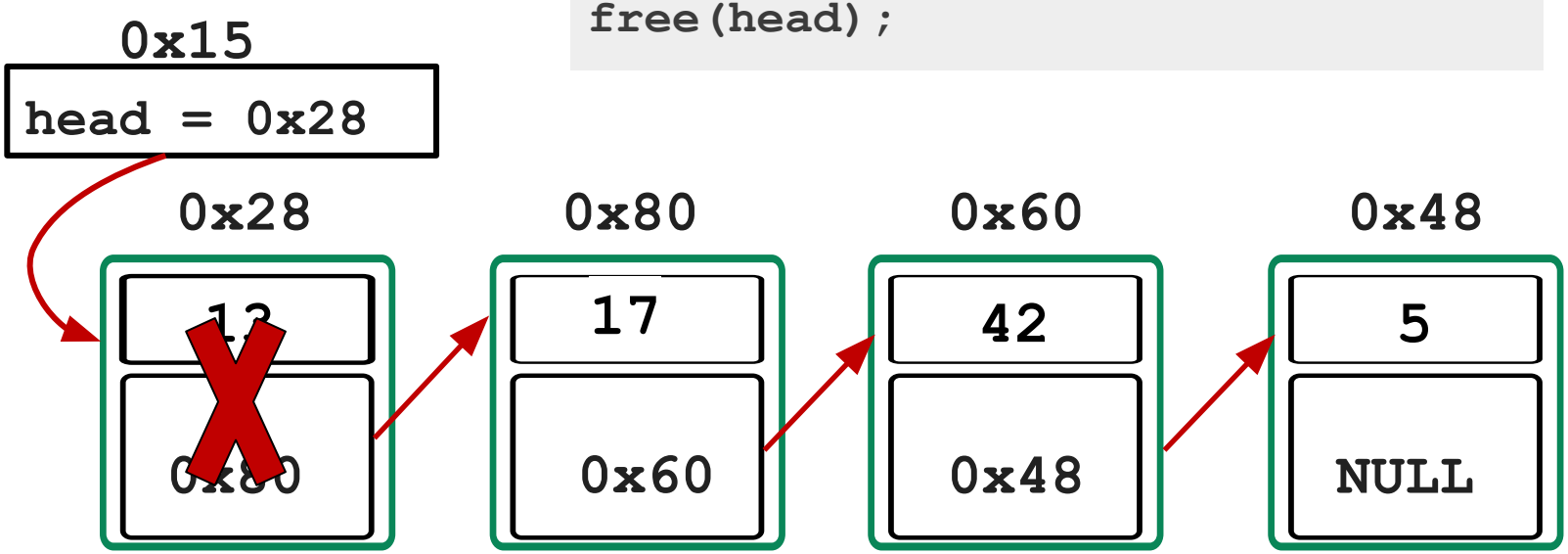
If our list is not empty, we want to make the second node the new head of the list and free the first node that we want to delete.



# Deleting the First Node in a Linked List

What would be the problem calling free on head first?

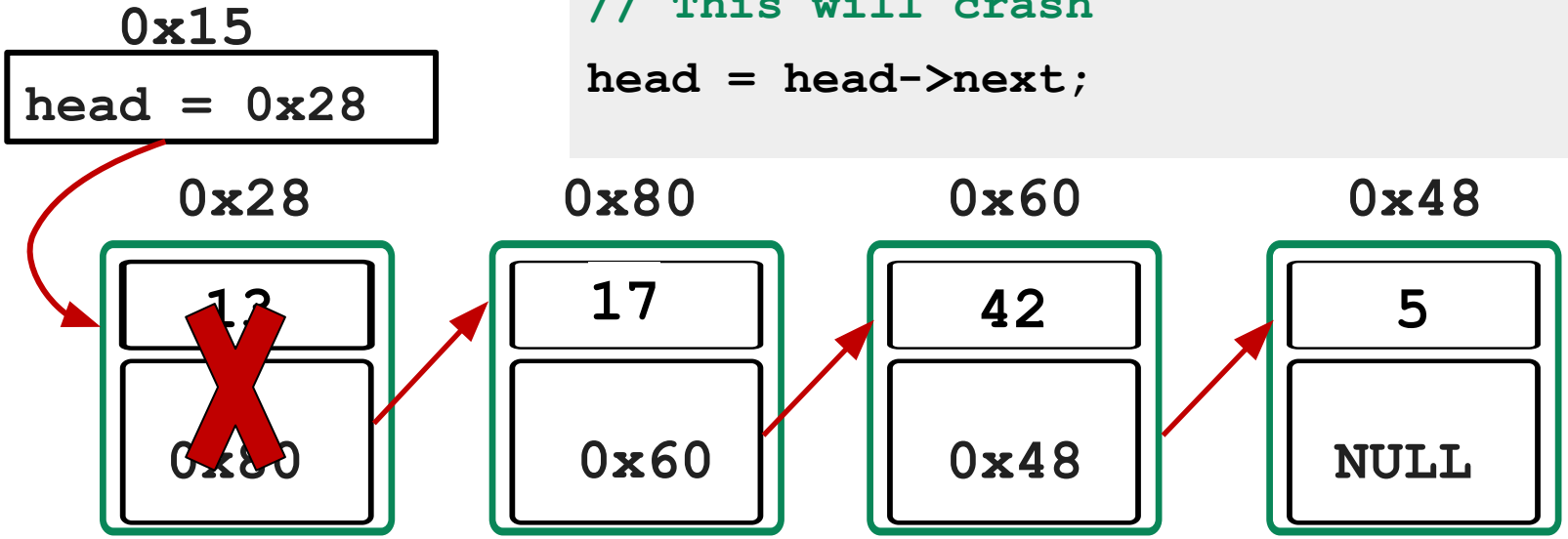
```
free(head);
```



# Deleting the First Node in a Linked List

We can't access memory that has been freed. We have lost the rest of the list

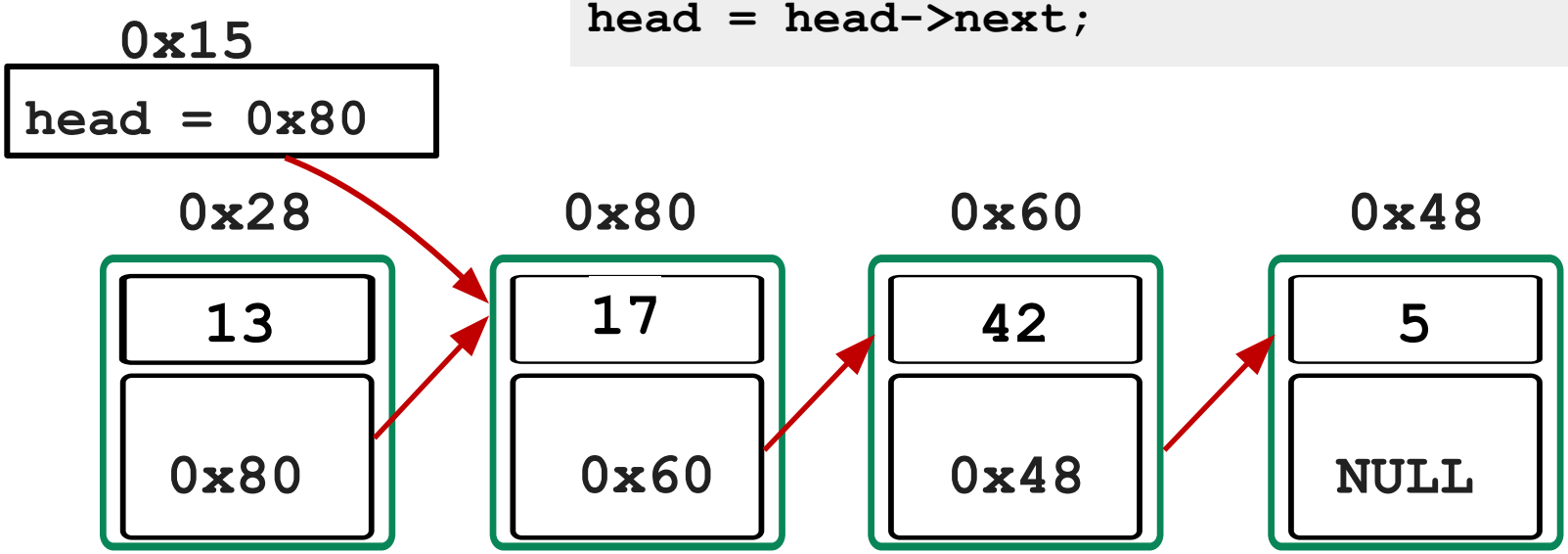
```
// This will crash
head = head->next;
```



# Deleting the First Node in a Linked List

What would be the problem with updating head first?

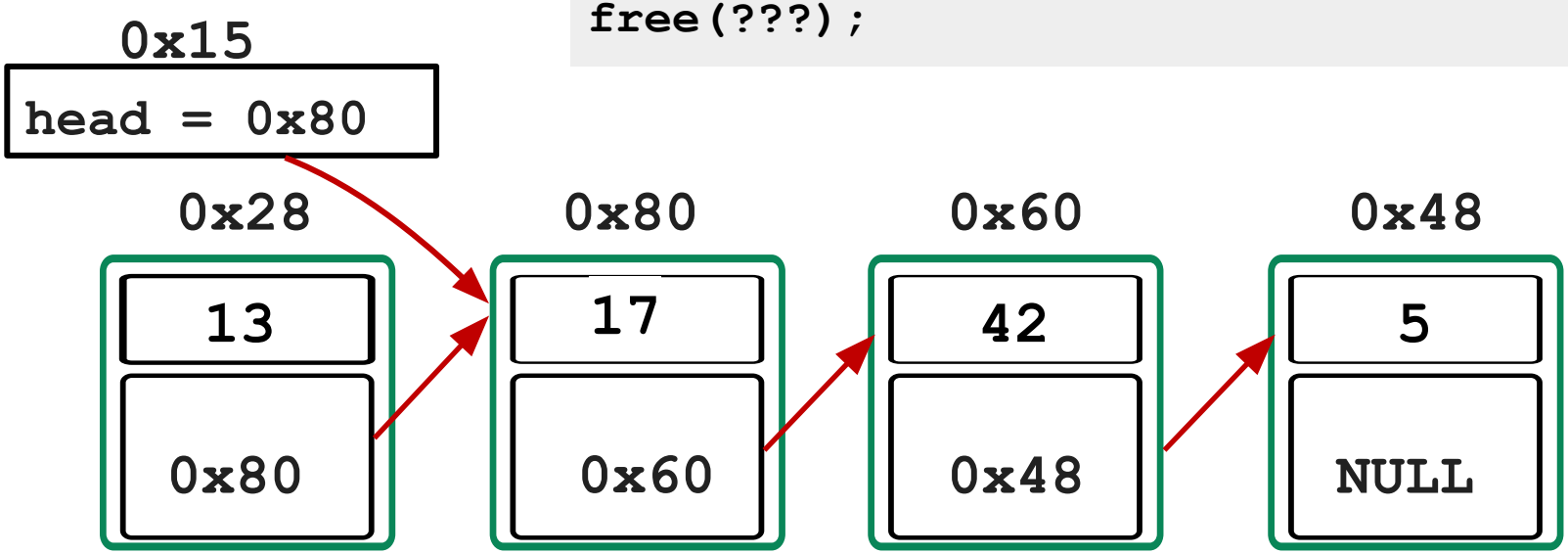
```
head = head->next;
```



# Deleting the First Node in a Linked List

We now have no pointer to the first node so we can't free it!

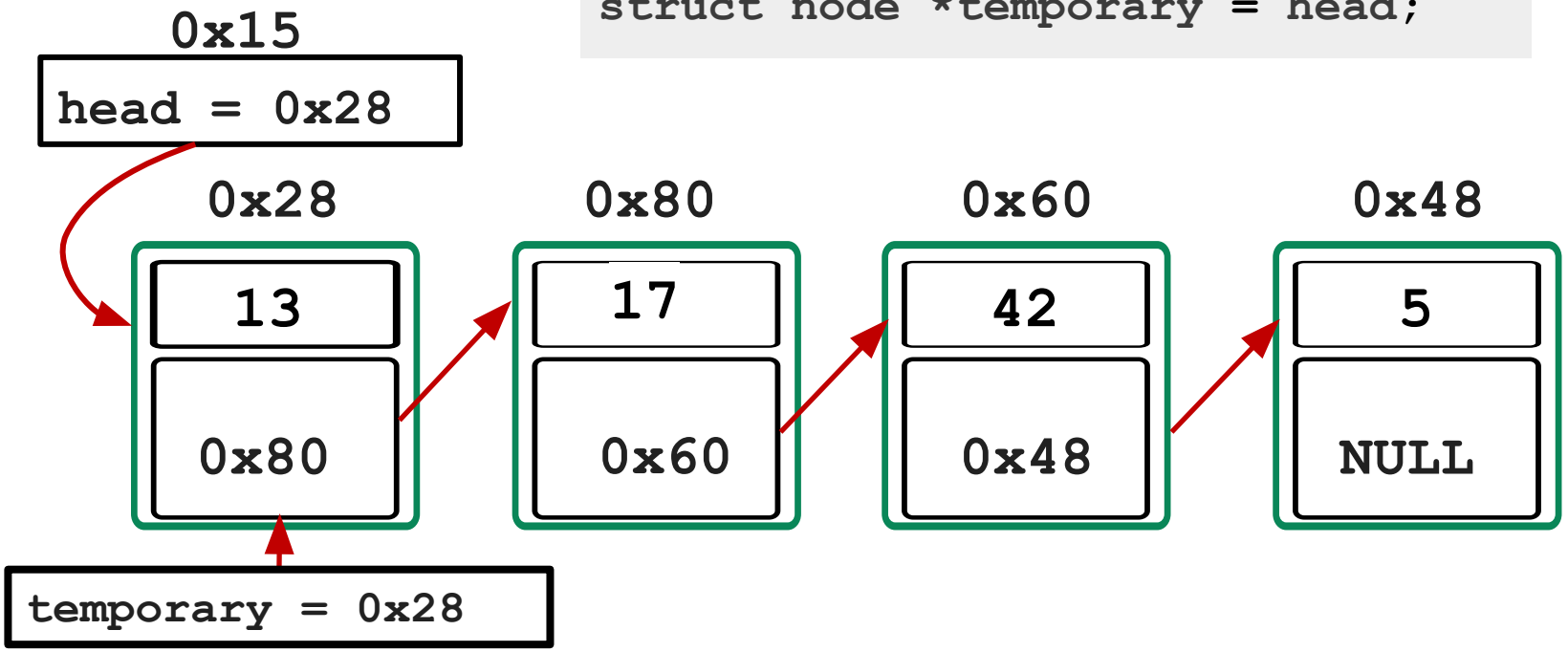
```
free(???);
```



# Deleting the First Node in a Linked List

Let's create a pointer to the first node

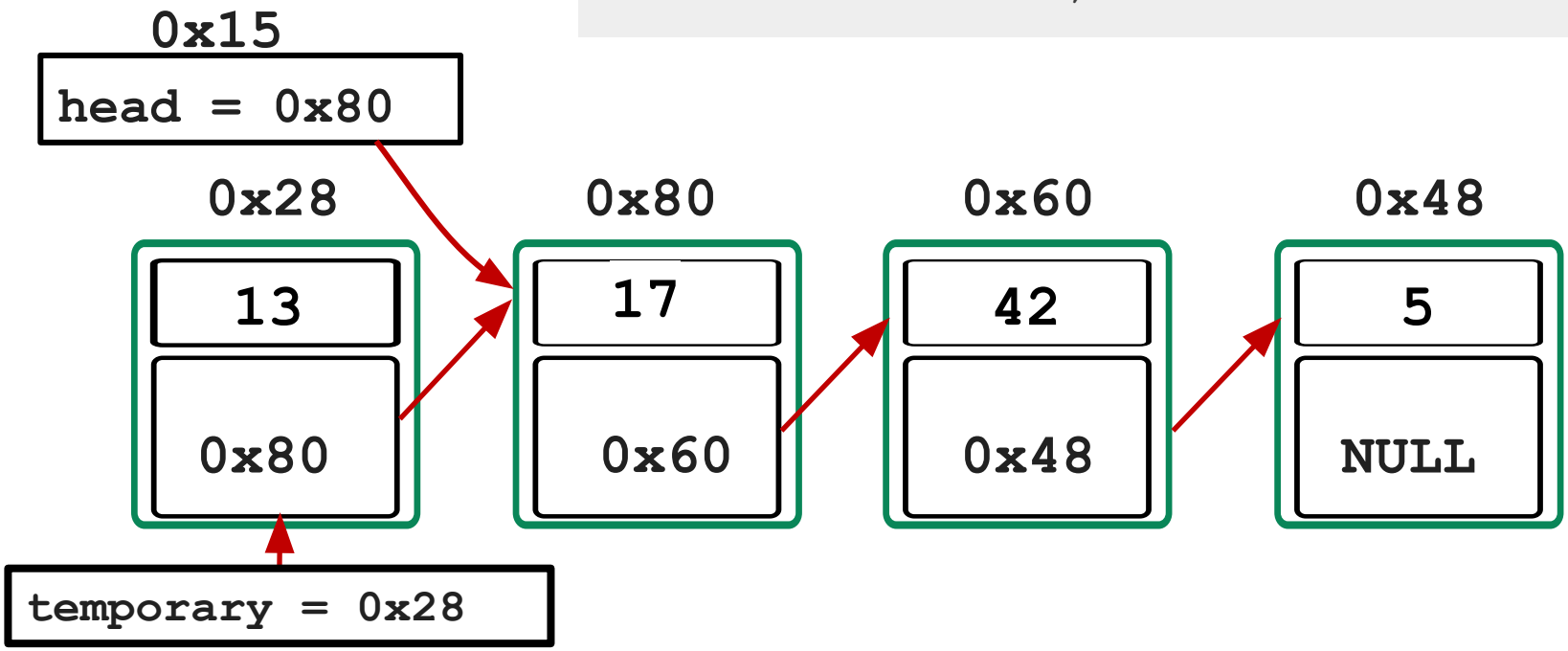
```
struct node *temporary = head;
```



# Deleting the First Node in a Linked List

Now we can update head

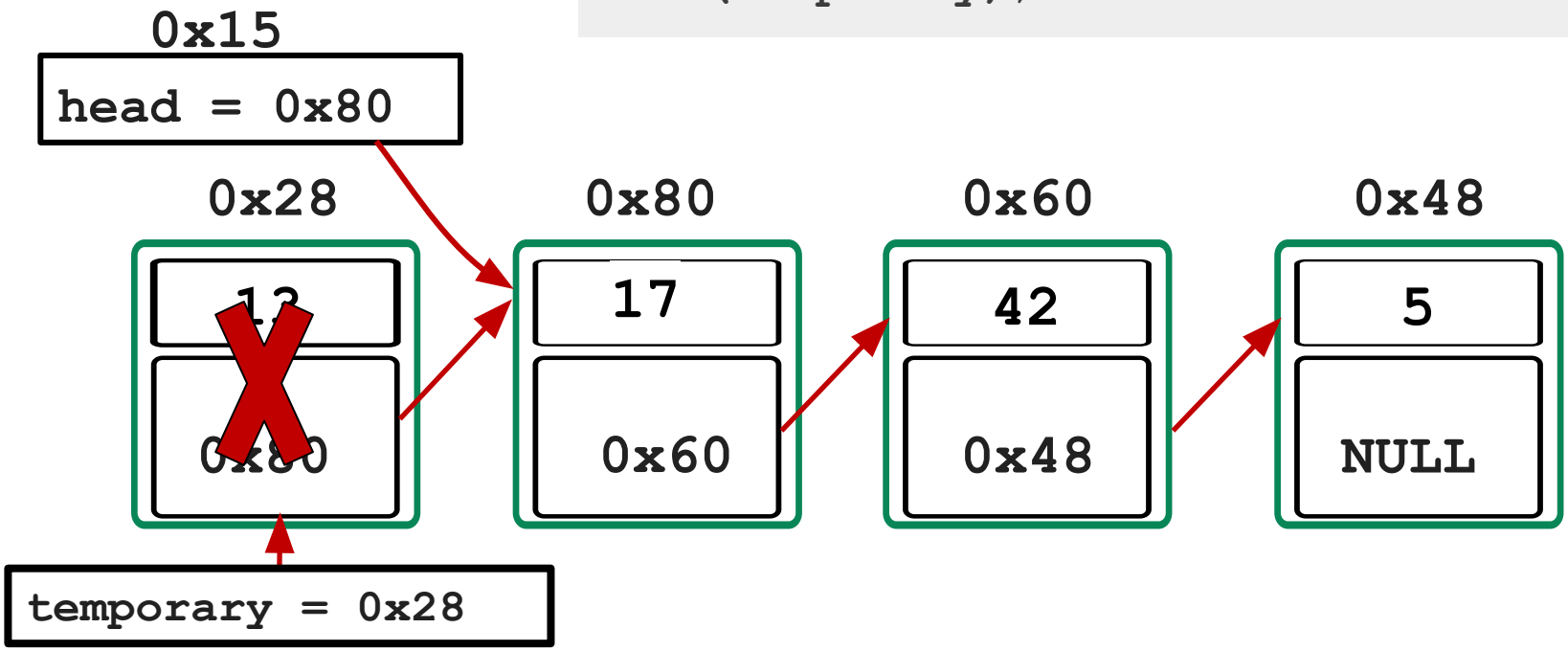
```
head = head->next;
```



# Deleting the First Node in a Linked List

Now we can free the first node

```
free(temporary);
```



# Deleting the First Node from a List

```
struct node *delete_first_node(struct node *head) {
    if (head == NULL) {
        return head;
    }
    struct node *temporary = head;
    head = head->next;
    free(temporary);
    return head;
}
```

# What did we learn today?

- Recap Linked Lists Basics
- Inserting an item at the tail of a list (linked_list_insertion.c)
- Inserting in the middle of a list
- Inserting at an index in a list
- Deleting the first node in a list (linked_list_deletion.c)

Next lecture:

- Deleting/freeing all nodes in a list
- Deleting a node from anywhere in a list
- Lists containing other types of data

# Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/pXdtYN4xgE

# Reach Out

Content Related Questions:
Forum

Admin related Questions email:
cs1511@unsw.edu.au

Don’t forget to attend Help Sessions
if you need more help