COMP1511 PROGRAMMING FUNDAMENTALS

Lecture 16

A very brief intro to recursion (useful for later courses, sometimes faster working)

YESTERDAY...

- More linked lists
- Intro to Abstract Data Types: Stacks

TODAY...

- Recursion
- The last lecture with new material!
- Please use the poll on this week's weekly announcement to let me know what kinds of topics you would like to cover in our revision lecture next week!

WHERE IS THE CODE?

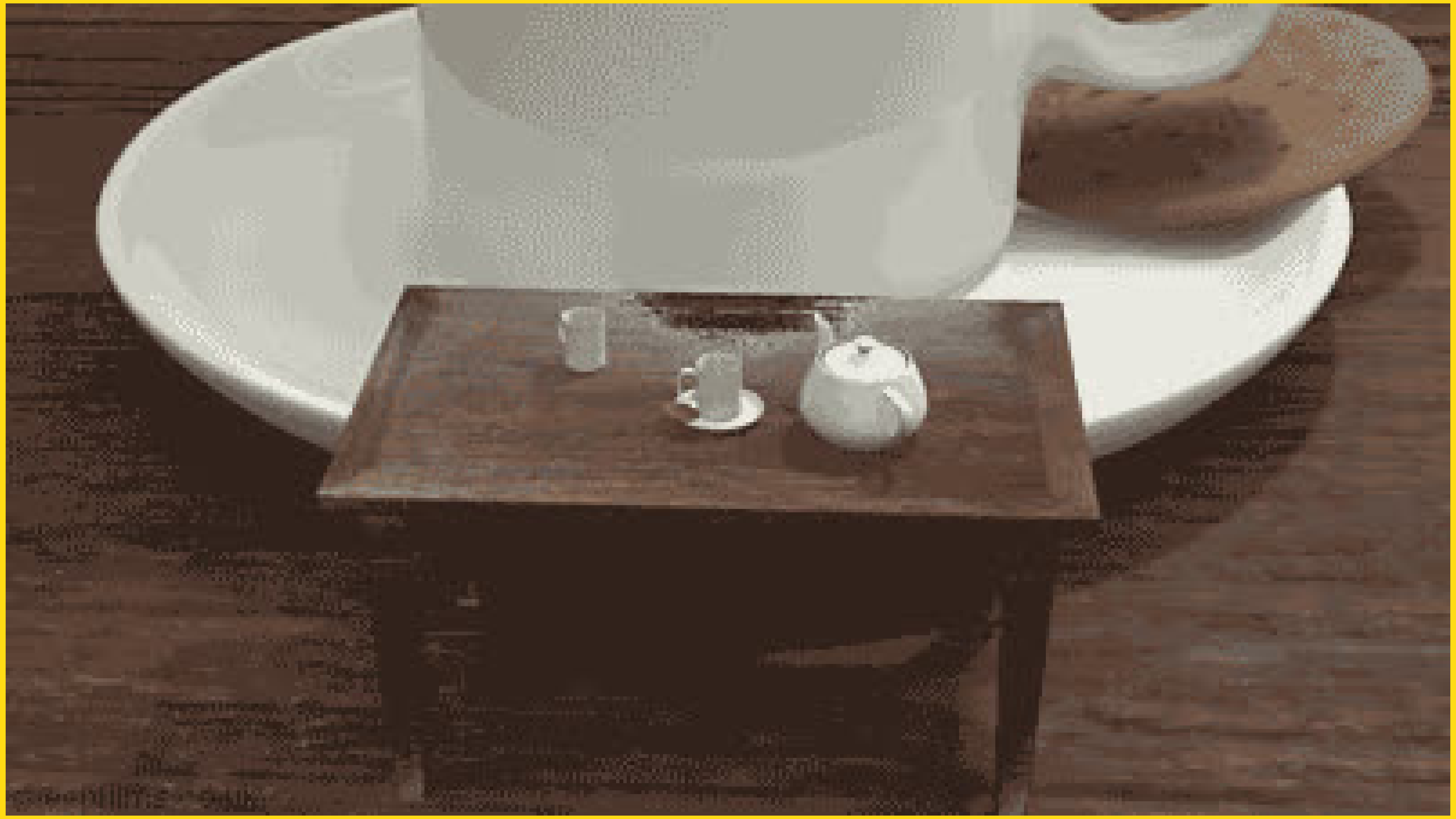
LIVE LECTURE CODE CAN BE FOUND HERE:

https://cgi.cse.unsw.edu.au/~cs1511/21T3/live/Week09/

RECURSION

WHAT IS IT?

- Hope noone is feeling sick yet
- Just giving you the vibe of recursion
- Think of a function that calls itself again and again and again until an end condition has been met



RECURSION

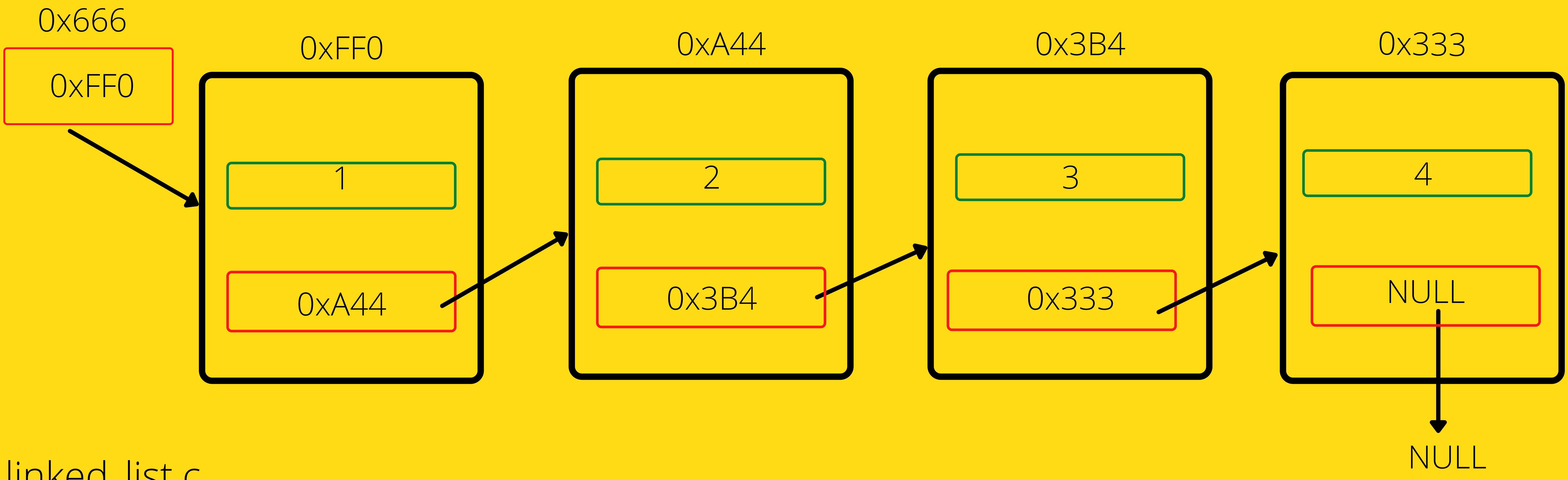
WHAT IS IT?

- Recursion is used all around us
- It is a different way of thinking and solving problems
    - You basically have to go backwards to go forwards, and if you are confused, welcome to your first taste of recursion - it does get better!

SUM OF A LINKED LIST

AN EXAMPLE OF CONVERTING TO A RECURSIVE FUNCTION

- Let's say we have a linked list:



linked_list.c

ADDING UP NUMBERS IN A LINKED LIST

AN EXAMPLE OF CONVERTING SOMETHING TO A RECURSIVE FUNCTION

- Think of a function that we would write in a normal way (iterative) that would add up all the nodes of a linked list:

```
int sum_list(struct node *head) {
    int sum = 0;
    struct node *current = head;
    while (current != NULL) {
        sum = sum + current->data;
        current = current->next;
    }
    return sum;
}
```

linked_list.c

SUM OF A LINKED LIST

AN EXAMPLE OF CONVERTING TO A RECURSIVE FUNCTION

- What if we tried to convert this function to a recursive function (function is called sum_list_recursive):

```
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```

- What is happening here?

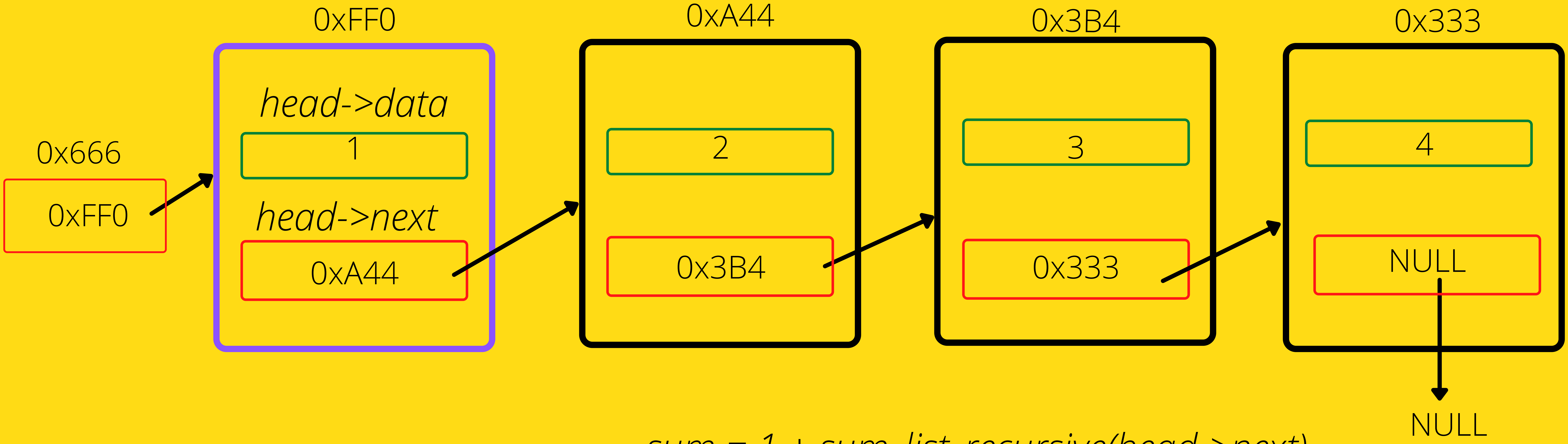
linked_list.c

SUM OF A LINKED LIST

FIRST CALL RECURSIVE FUNCTION

- What is happening here?

```c
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```



sum = 1 + sum_list_recursive(head->next)

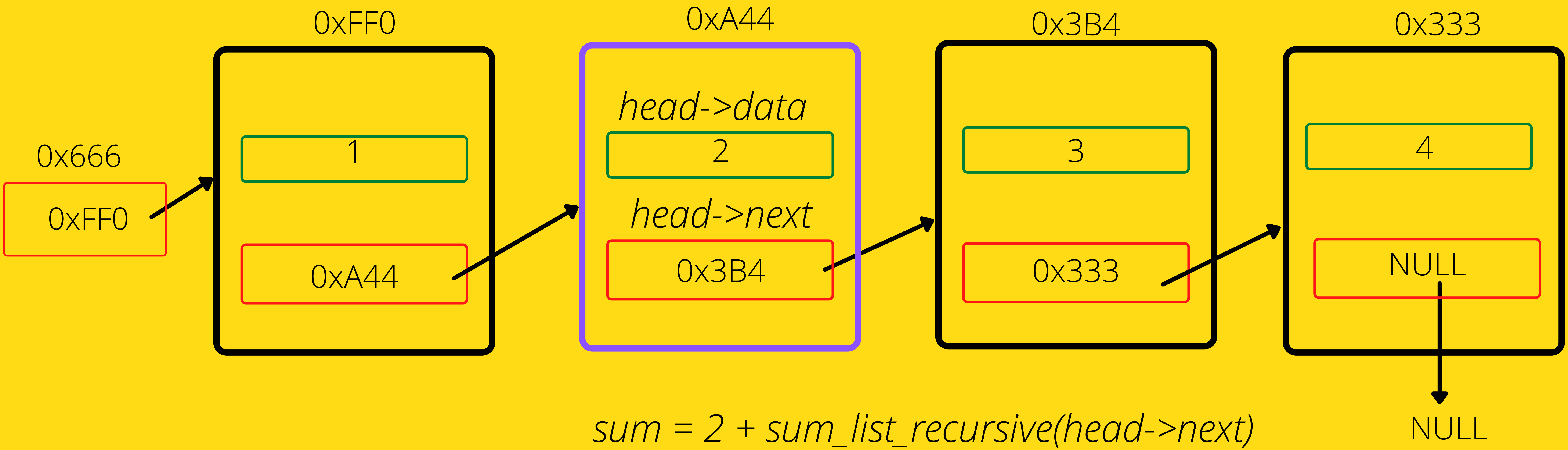
linked_list.c

SUM OF A LINKED LIST

SECOND CALL RECURSIVE FUNCTION

- What is happening here?

```
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```



sum = 2 + sum_list_recursive(head->next)

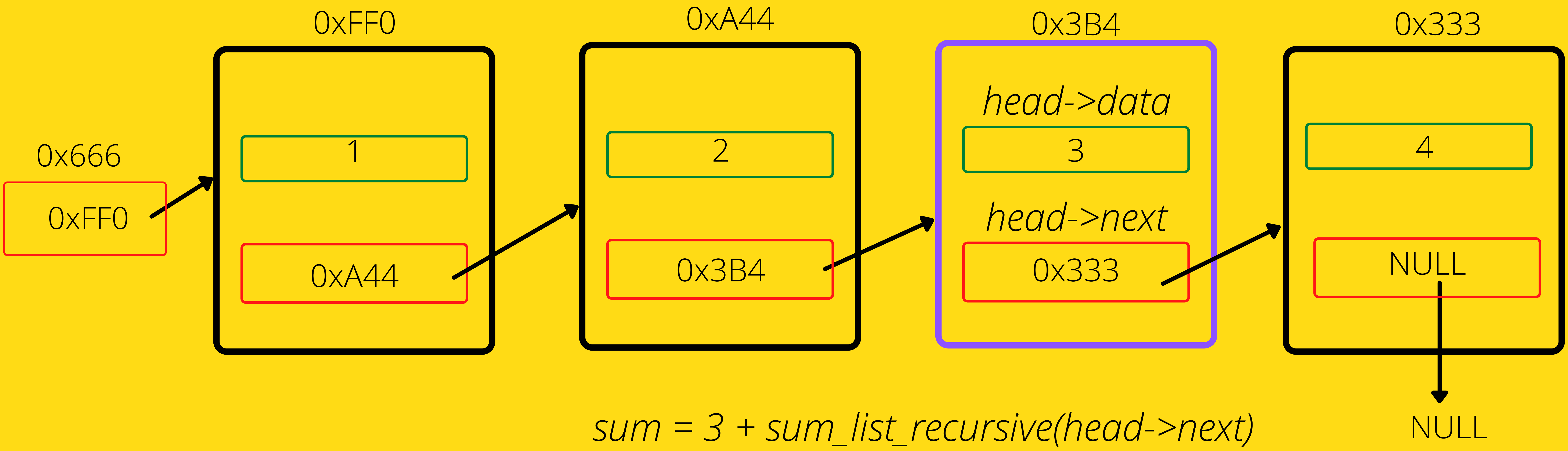
linked_list.c

SUM OF A LINKED LIST

THIRD CALL RECURSIVE FUNCTION

- What is happening here?

```
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```



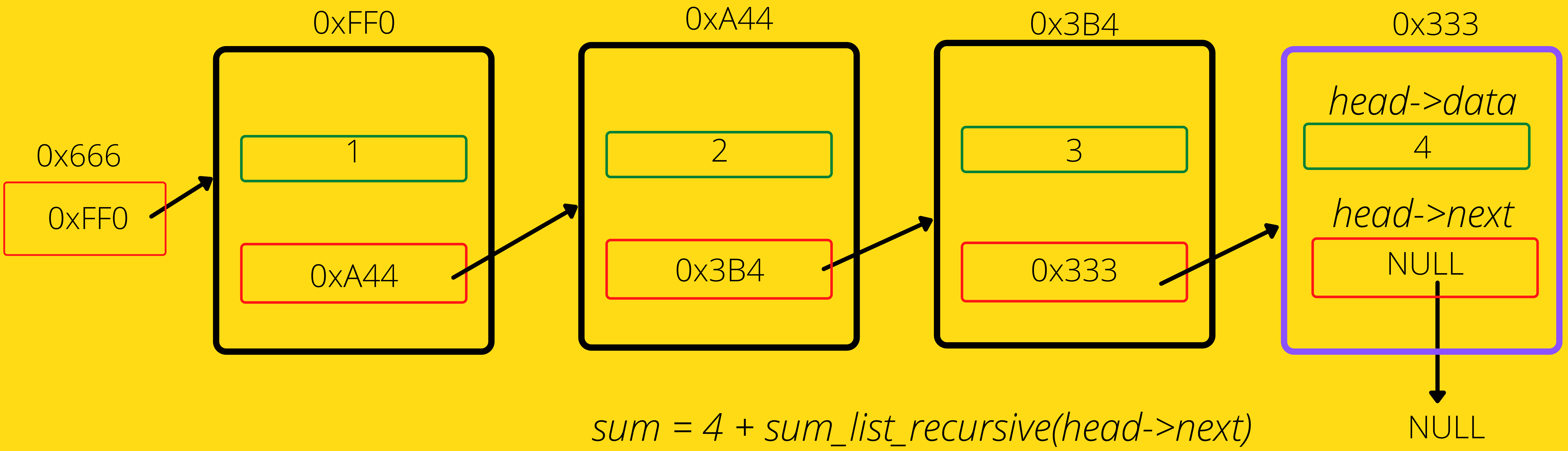
linked_list.c

SUM OF A LINKED LIST

FOURTH CALL RECURSIVE FUNCTION

- What is happening here?

```
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```



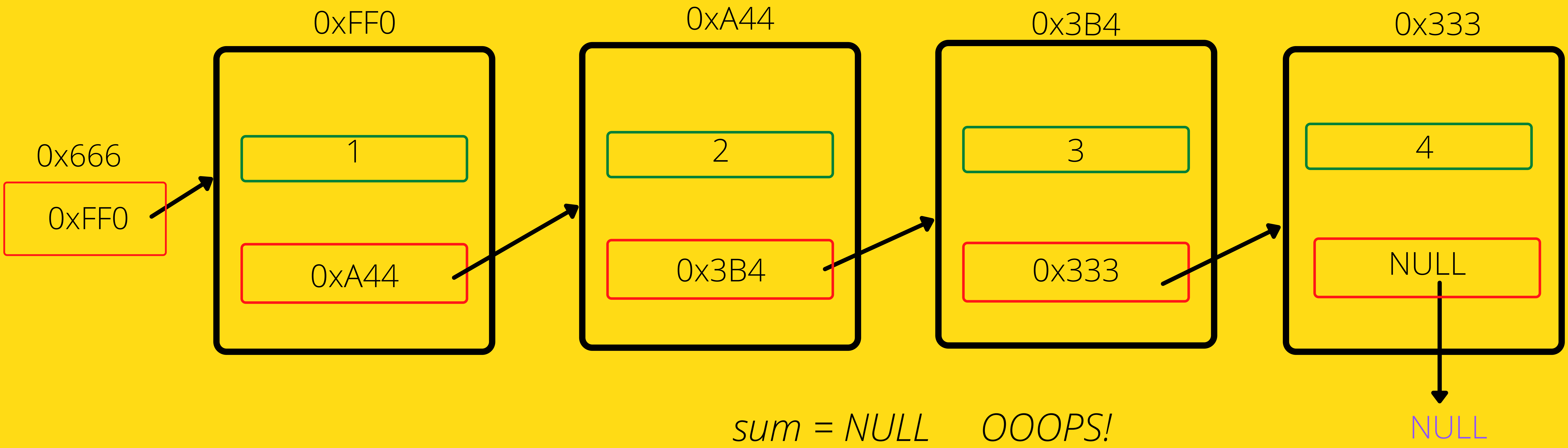
linked_list.c

SUM OF A LINKED LIST

FIFTH CALL RECURSIVE FUNCTION

- What is happening here?

```c
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here

    //Recursive function
    int sum = head->data + sum_list_recursive(head->next)
    return sum;
}
```



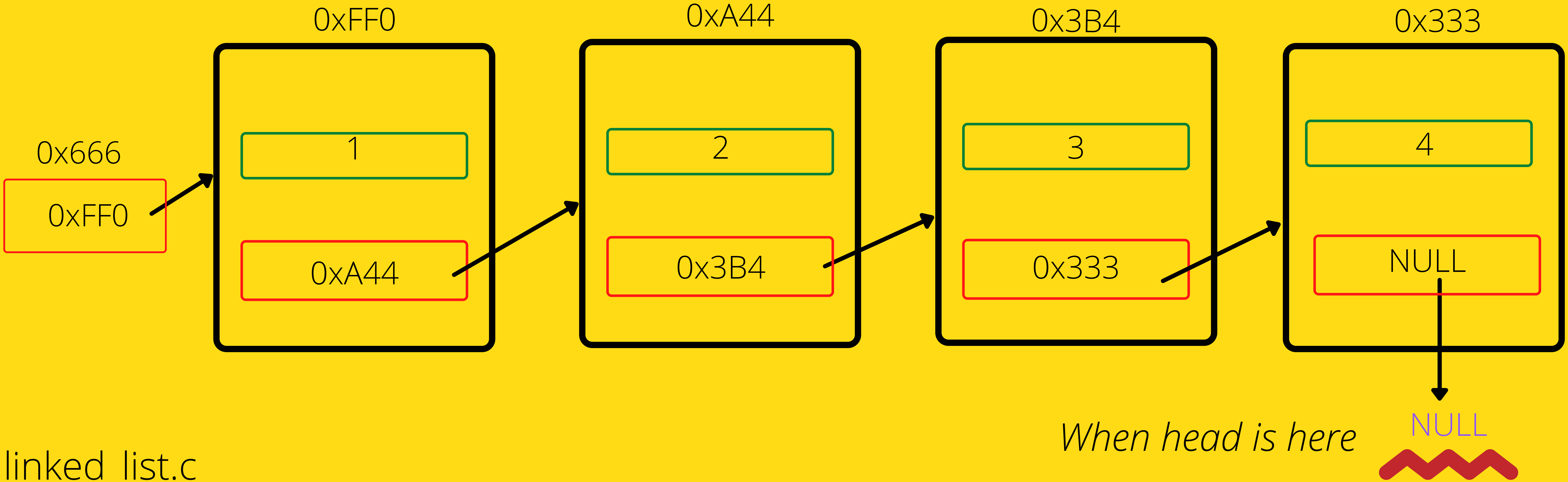
sum = NULL   *OOOPS!*

linked_list.c

SUM OF A LINKED LIST

WHEN SHOULD WE STOP?

```c
int sum_list_recursive (struct node *head) {
    //Need to know when to stop ...
    //TODO: put an end case here
    if (head == NULL) {
        return 0;
    } else {
        //Recursive function
        int sum = head->data + sum_list_recursive(head->next)
        return sum;
    }
}
```



linked_list.c

SO WHAT WILL IT LOOK LIKE?

A SUMMARY...

The functions are all kept on the stack until the stopping case is reached, and then the functions starts returning and popping the recursive calls off the stack

1st call: *sum = 1 + sum_list_recursive(head->next)*

2nd call: *sum = 2 + sum_list_recursive(head->next)*

3rd call: *sum = 3 + sum_list_recursive(head->next)*

4th call: *sum = 4 + sum_list_recursive(head->next)*

5th call: *0*

Now rebuild back up

4th call: *sum = 4 + 0 (4)*

3rd call: *sum = 3 + 4 (7)*

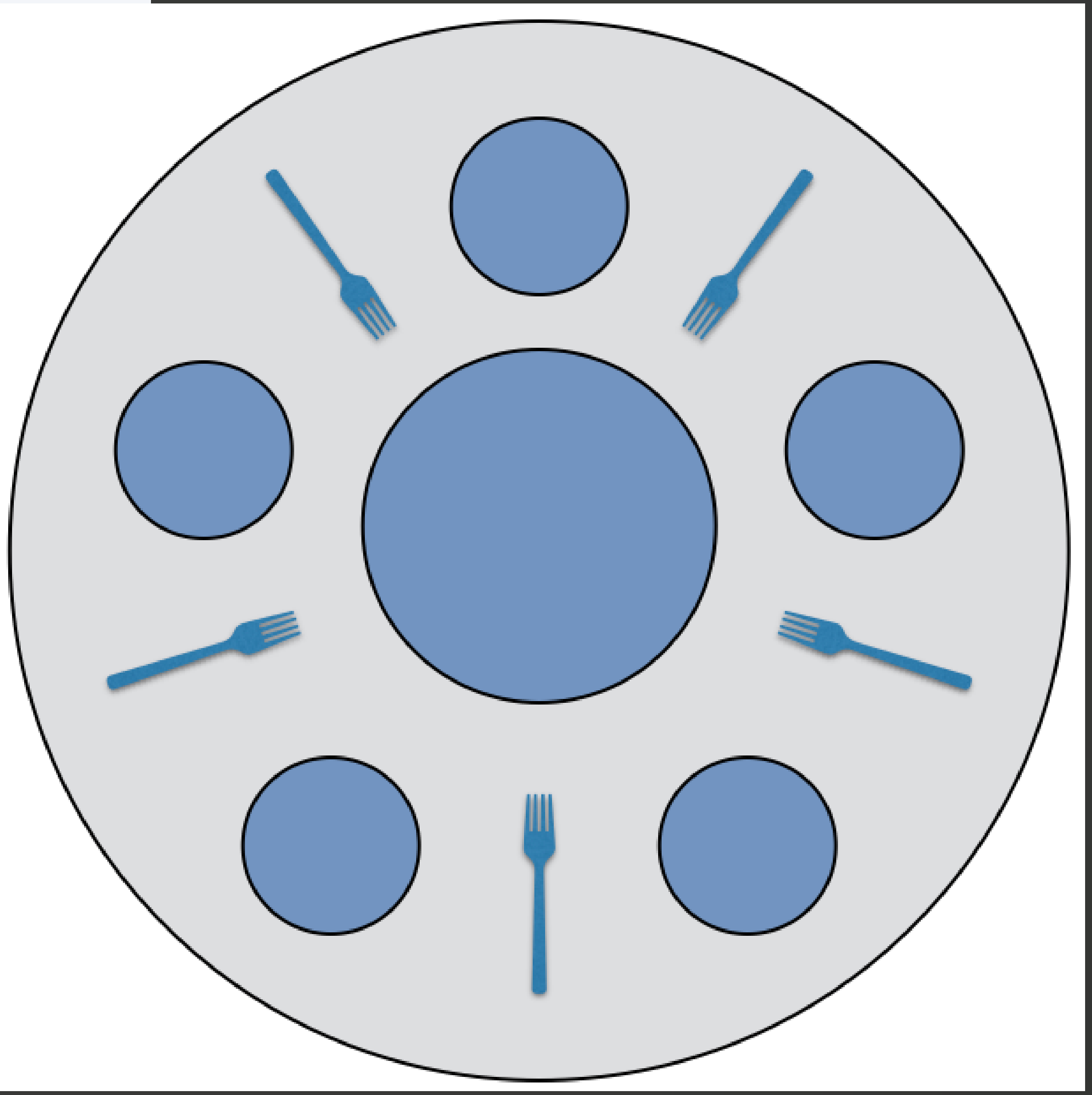
2nd call: *sum = 2 + 7 (9)*

1st call: *sum = 1 + 9*

sum = 10

BREAK TIME (5 MINUTES)

Five silent philosophers sit at a table around a giant plate of cake. A fork is placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat cake when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it's not being used by another philosopher. How can we ensure that no philosopher starves?



LET'S TRY THE PRINT_LIST FUNCTION

FOR FUN

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

void print_list_recursive(struct node *head) {
    if (head == NULL) {
        printf("X\n");
    } else {
        printf("%d -> ", head->data);
        print_list(head->next);
    }
}
```

LET'S TRY A FEW SIMPLE FUNCTIONS FOR "FUN"

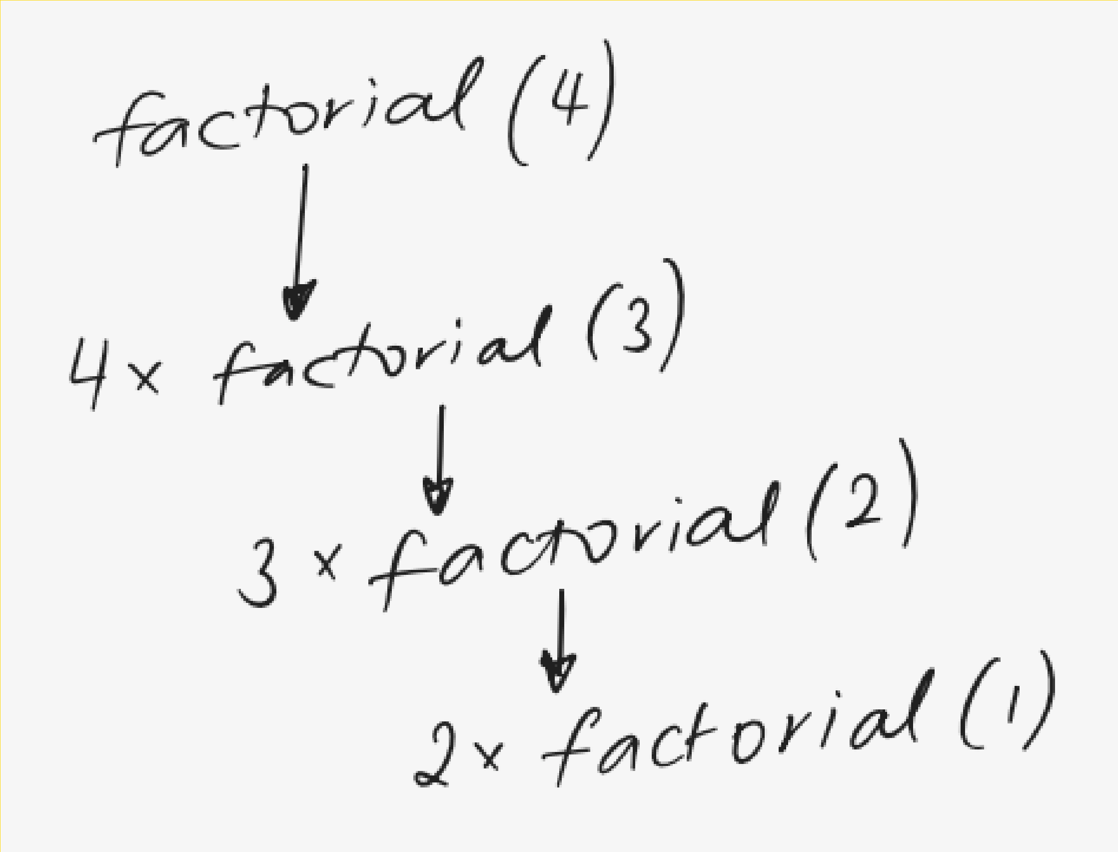
FINDING A FACTORIAL

```c
// Think of the example of factorials
// 1! = 1
// 2! = 2 * 1 = 2
// 3! = 3 * 2 * 1 = 6
// 4! = 4 * 3 * 2 * 1 = 24
// 5! = 5 * 4 * 3 * 2 * 1 = 120
// So what do we think would be the stopping case?
// What about the recursive case?

#include <stdio.h>

int factorial(int number);

int main(void) {
    int number;
    printf("Enter a number to find factorial of: ");
    scanf("%d", &number);
    printf("The factorial of %d! is %d\n", factorial(number));
    return 0;
}

//Let's write our function!
```

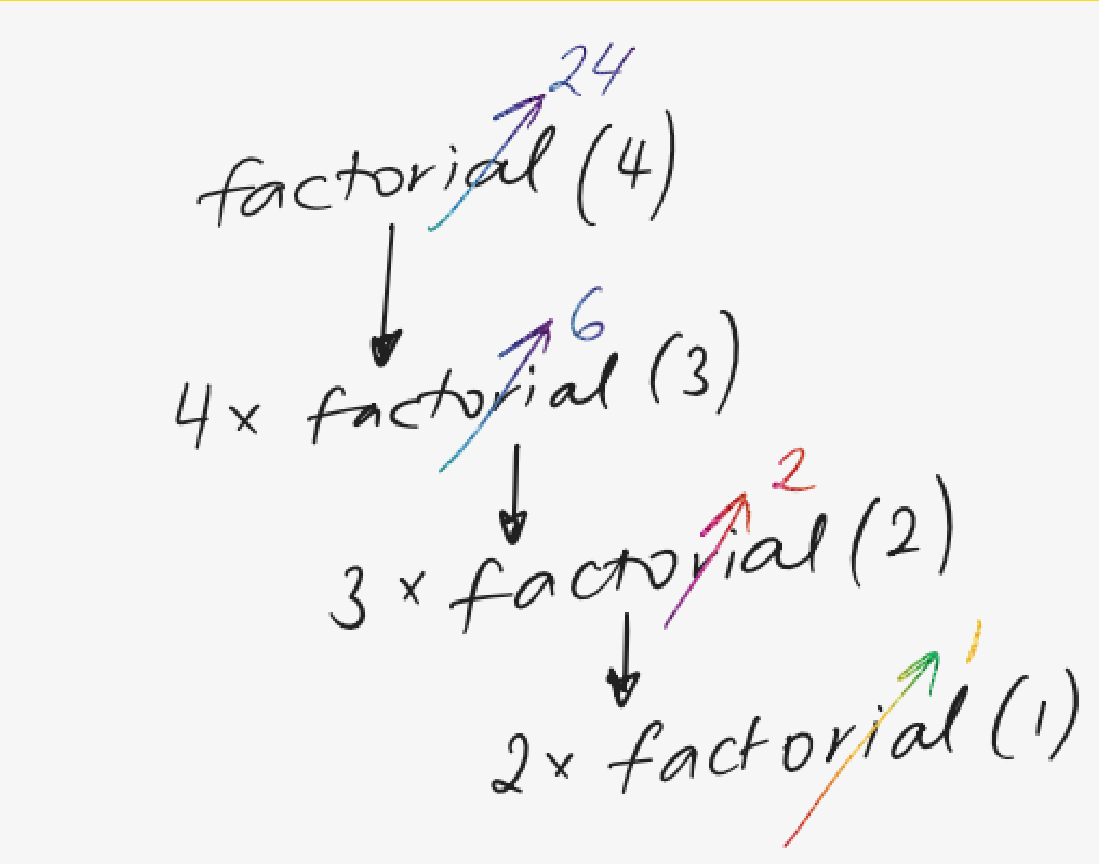
SO WHAT WILL IT LOOK LIKE?

A SUMMARY OF 4!...

The functions are all kept on the stack until the stopping case is reached, and then the functions starts returning and popping the recursive calls off the stack



Now rebuild back up



factorial = 24

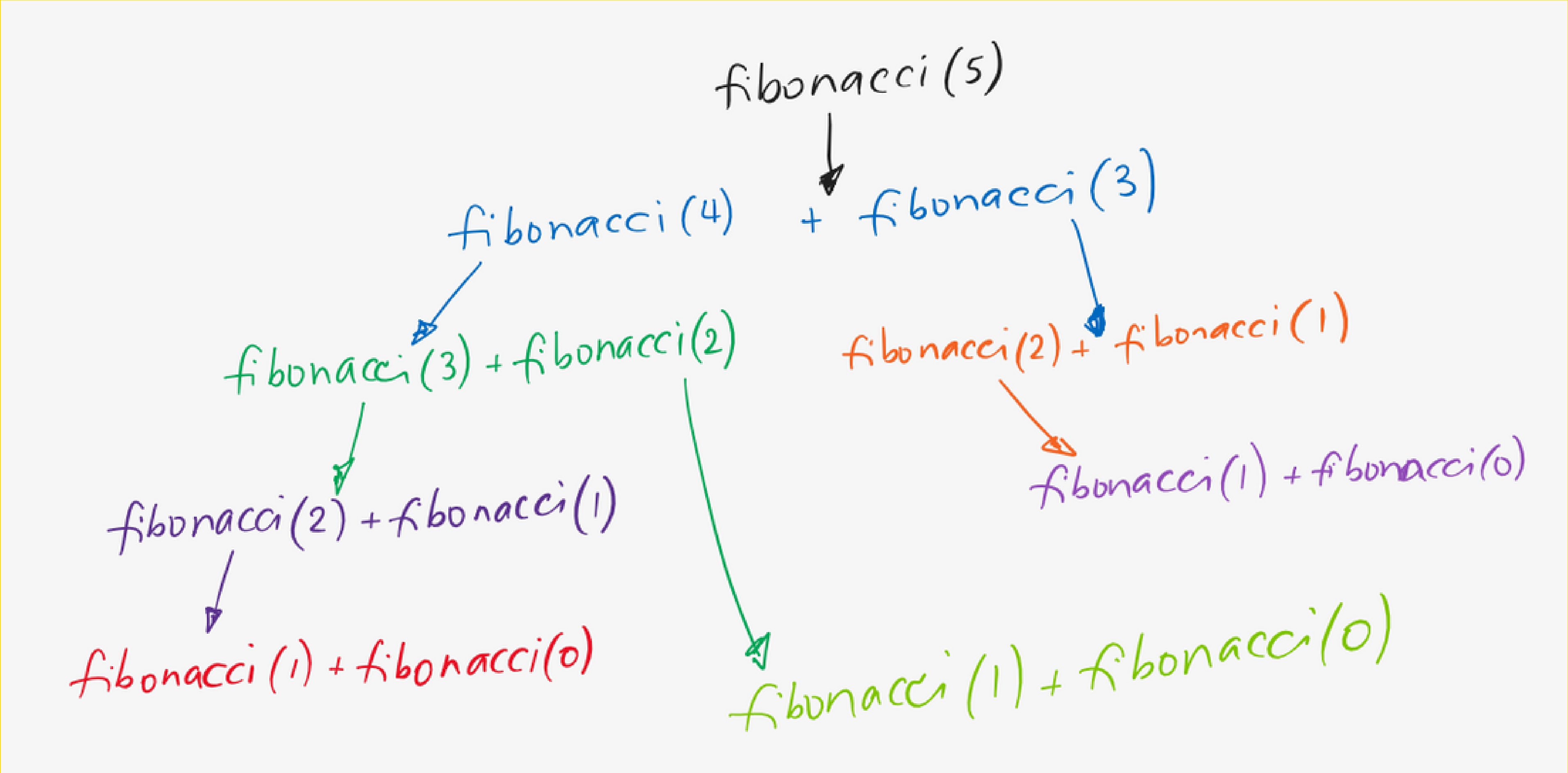
LET'S TRY A FEW SIMPLE FUNCTIONS FOR "FUN"

FIBONACCI NUMBERS

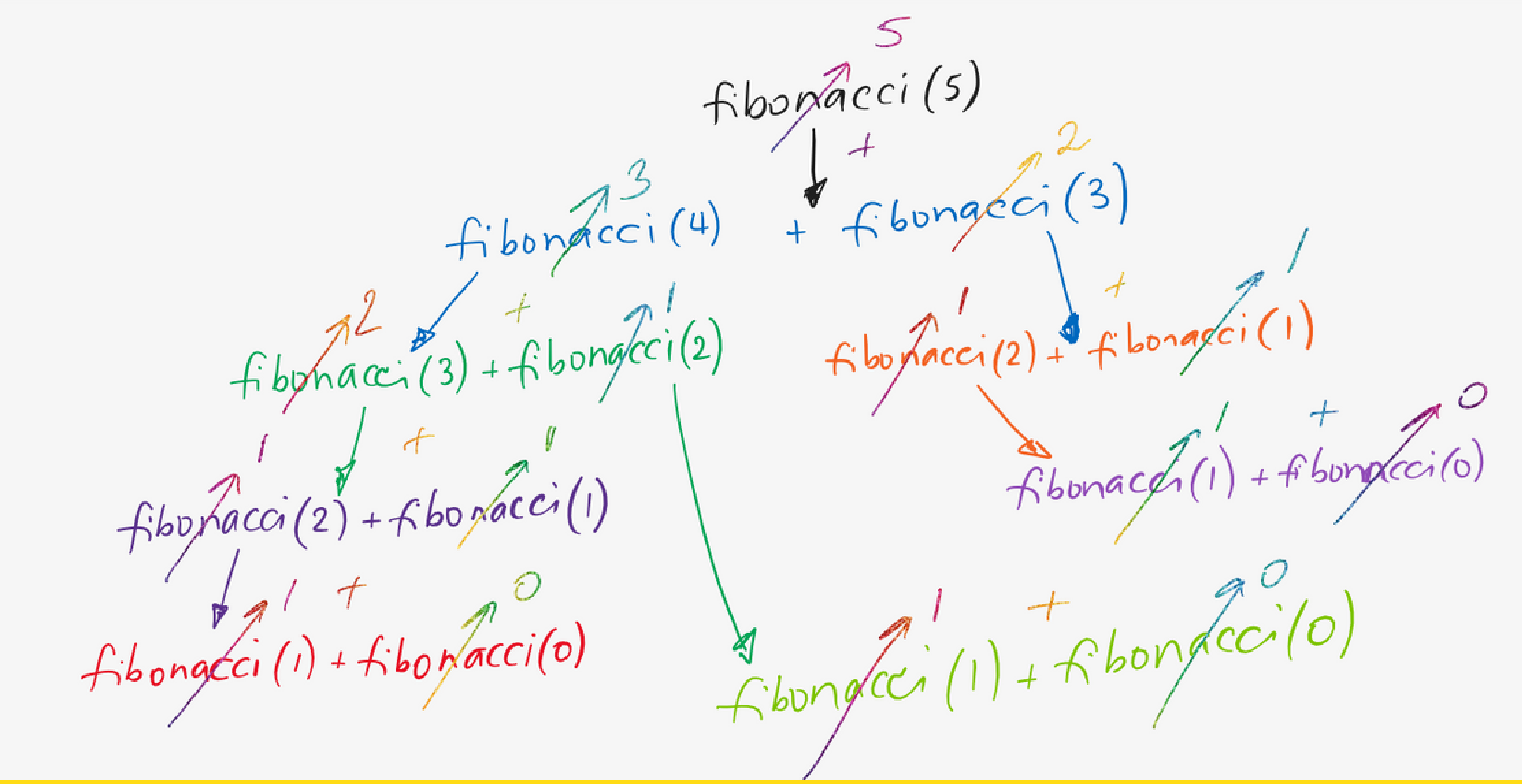
```c
// Think of the example of fibonacci numbers, where each number
// is the sum of the previous two numbers.
// So sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21....

// So what do we think would be the stopping case?
// What about the recursive case?

#include <stdio.h>

int fibonacci(int number);

int main(void) {
    int number;
    printf("Enter which term of fibonacci sequence you want to see: ");
    scanf("%d", &number);
    printf("fibonacci(%d) is %d\n", number, fibonacci(number));
    return 0;
}

//Let's write our function!
```

SO WHAT WILL IT LOOK LIKE?

A SUMMARY OF FIBONACCI (5)...



Now rebuild back up



The functions are all kept on the stack until the stopping case is reached, and then the functions starts returning and popping the recursive calls off the stack

fibonacci(5) = 5

NEXT WEEK'S REVISION LECTURE

PLEASE LET ME KNOW WHICH TOPICS IN PARTICULAR YOU WOULD LIKE TO FOCUS ON IN WEDNESDAY'S LECTURE - I AM RUNNING A POLL!

FEEDBACK?

PLEASE LET ME KNOW ANY FEEDBACK FROM TODAY'S LECTURE!

www.menti.com

Code: 8220 1482

WHAT DID WE LEARN TODAY?

RECURSION

linked_list.c (sum list and print list)
factorial.c
fibonacci.c

ANY QUESTIONS?

DON'T FORGET YOU CAN ALWAYS EMAIL US ON CS1511@CSE.UNSW.EDU.AU FOR ANY ADMIN QUESTIONS

PLEASE ASK IN THE FORUM FOR CONTENT RELATED QUESTIONS