COMP1511 - Programming Fundamentals

Term 1, 2019 - Lecture 17
Stream B

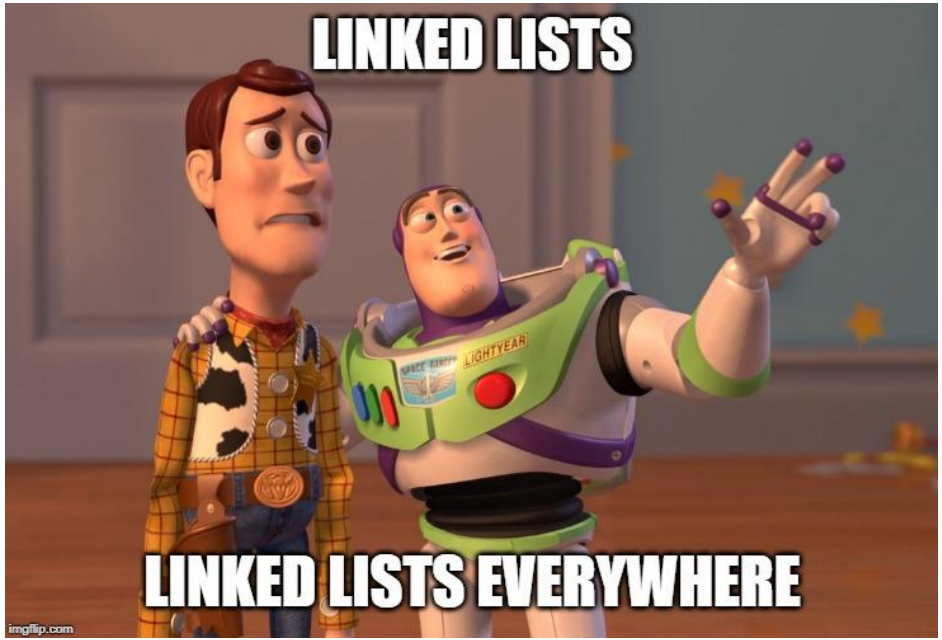
What did we cover last week?

Memory

- Functions and memory
- Memory allocation

Linked Lists

- Node structs
- List traversal
- Adding nodes
- Removing nodes (nearly!)



What are we covering today?

Linked Lists

- Continuing work on removing nodes
- Finishing the Battle Royale example

Assignment 2 - Pokédex

- What is a Pokédex?
- Structure of Assignment 2
- Pokédex implementation

Linked Lists - Recap

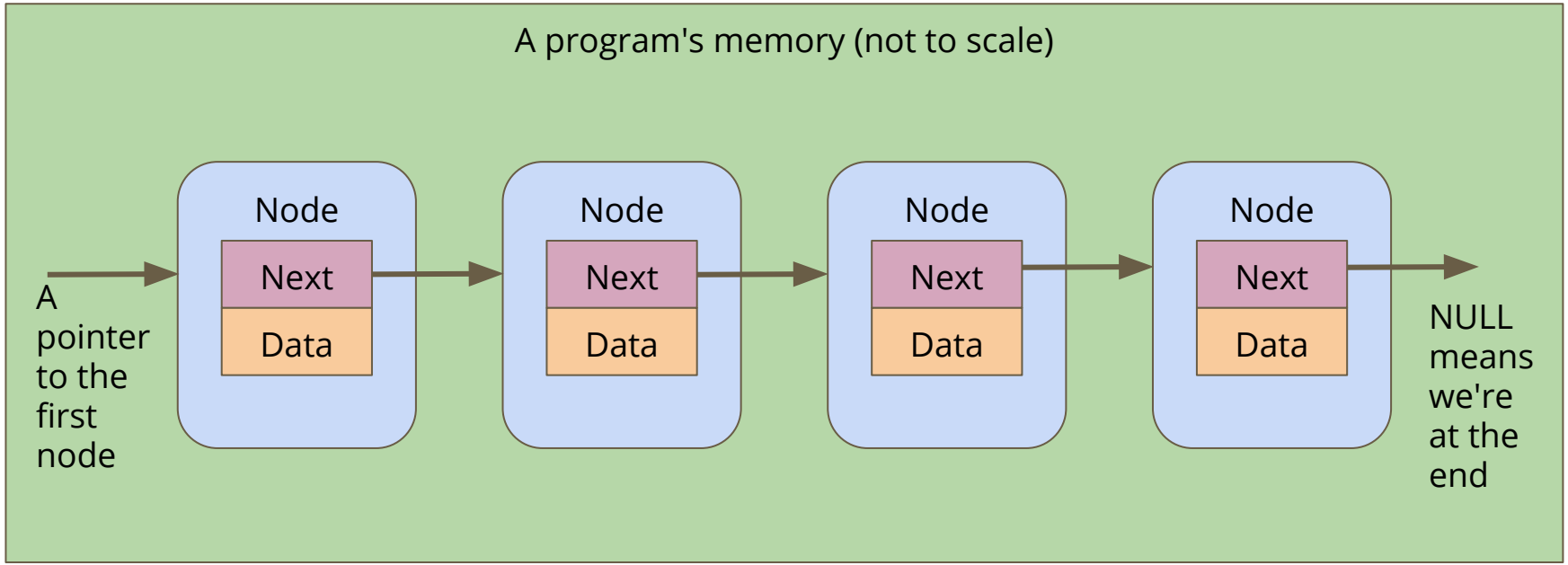
Nodes

- Nodes are structs with pointers to other nodes
- We use malloc to allocate memory for a node
- Chains of nodes form a linked list

Linked Lists

- We use a pointer to the first node to keep track of the list
- The last node has a pointer to NULL so we know it's the end

Linked List diagram



Battle Royale - continued

Where did we get up to?

- We have our struct (with a name in it)
- We are able to add these to a linked list
- We are able to insert elements into a list in order

What's left

- Removing a node for when someone is knocked out
- Some code for "playing" the game

A recap

A simple node struct

- We can insert this in a simple way
- Just by replacing next pointers

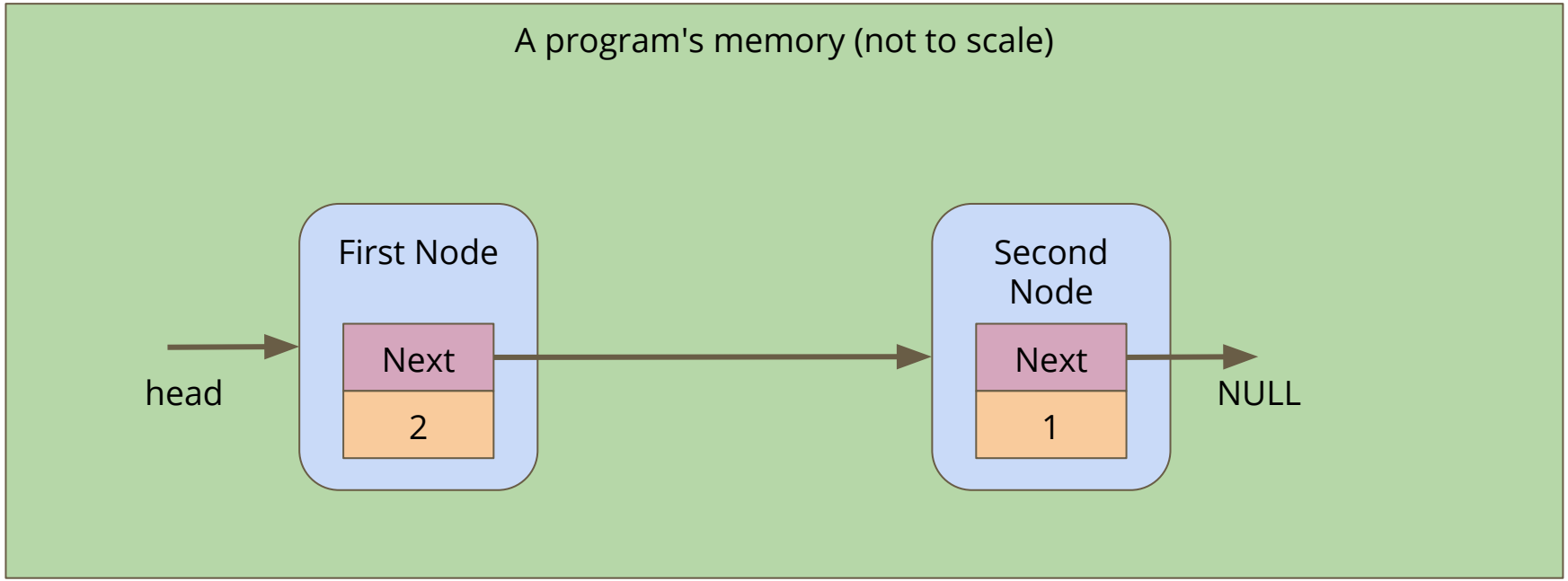
```
struct node {
    struct node *next;
    char name[50];
}
```

Inserting at an arbitrary position in a list

- Loop through the list to find the position we want
- Insert by manipulating where pointers are aiming

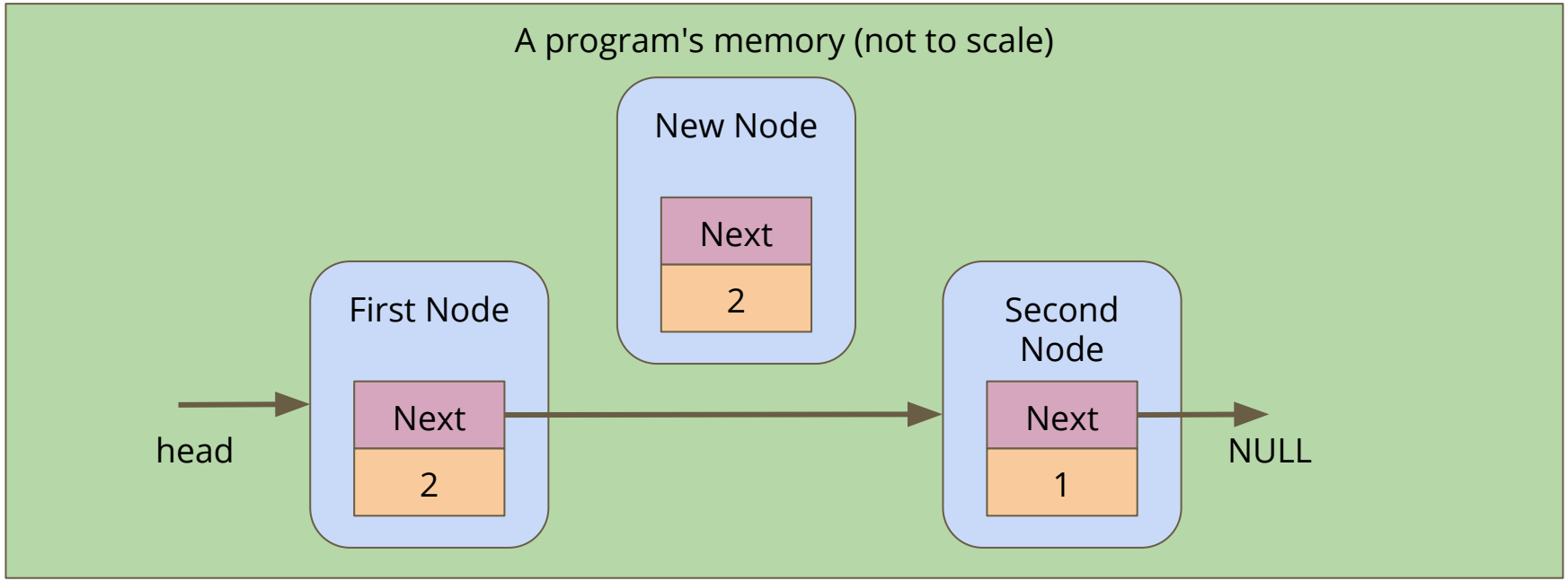
Insertion recap

Before we've tried to insert anything



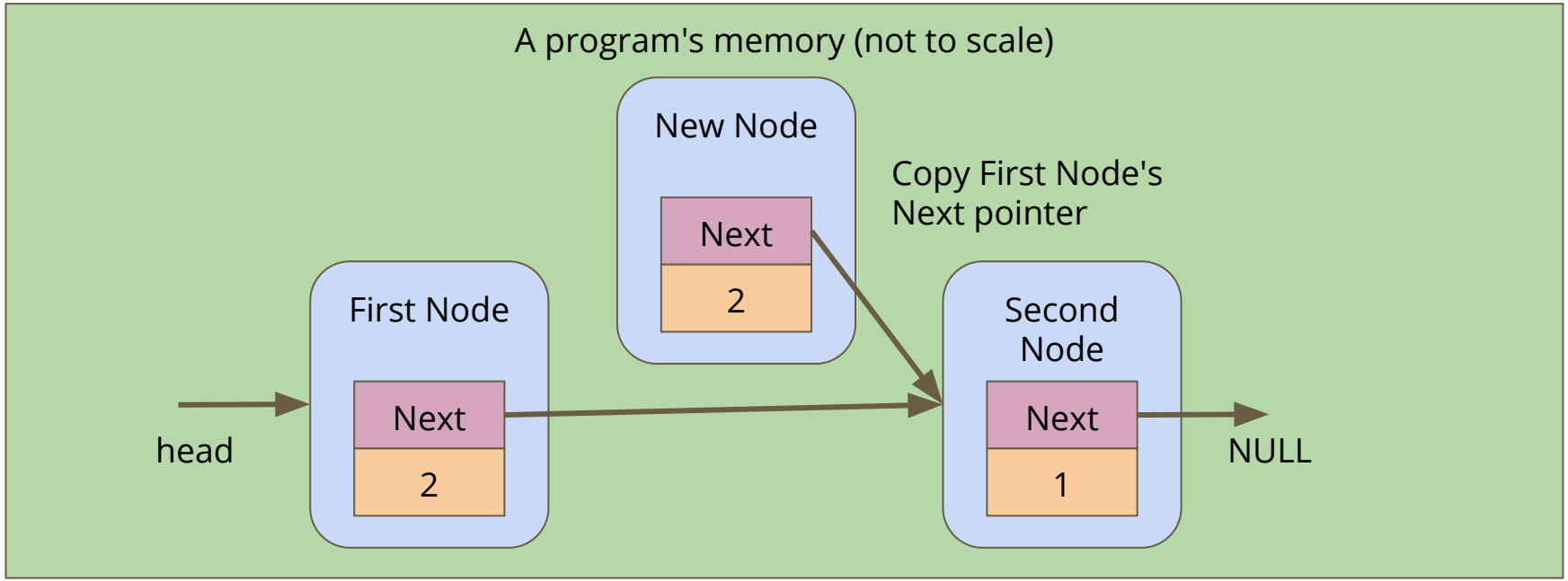
Create a node

A new node is made, it's not connected to anything yet



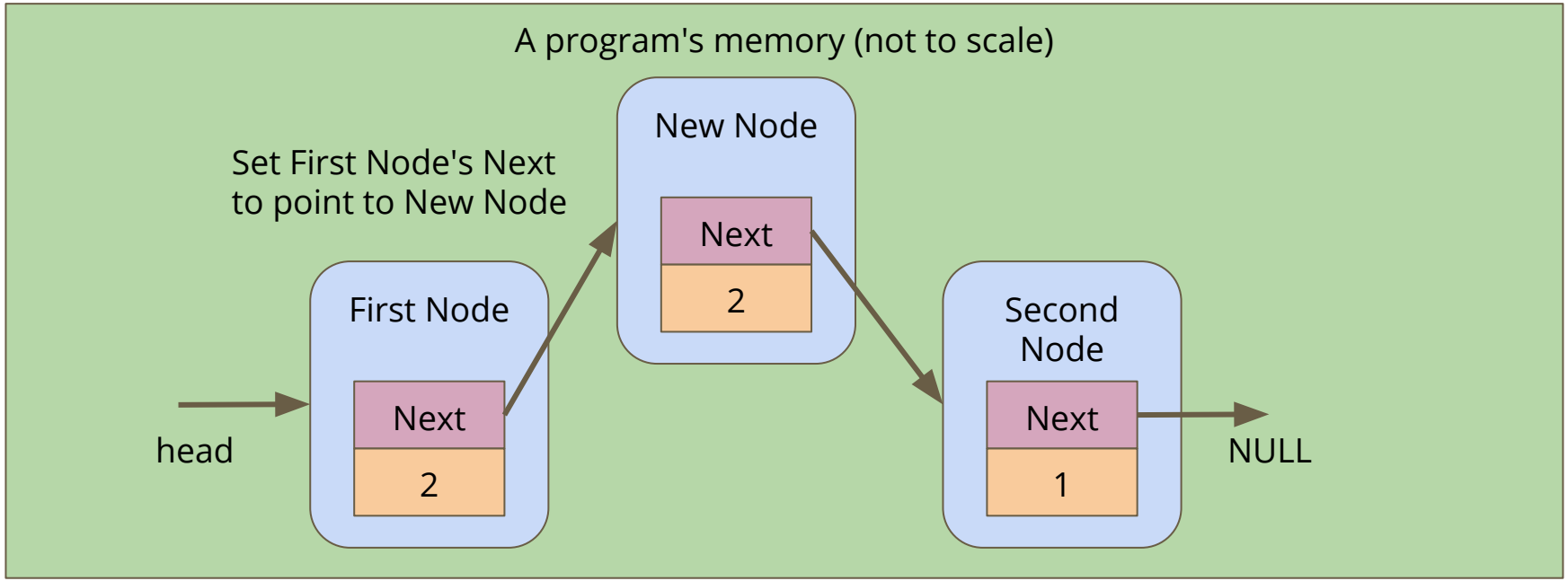
Connect the new node

Alter the **next** pointers on the First Node and the New Node



Connect the new node

Alter the **next** pointers on the First Node and the New Node



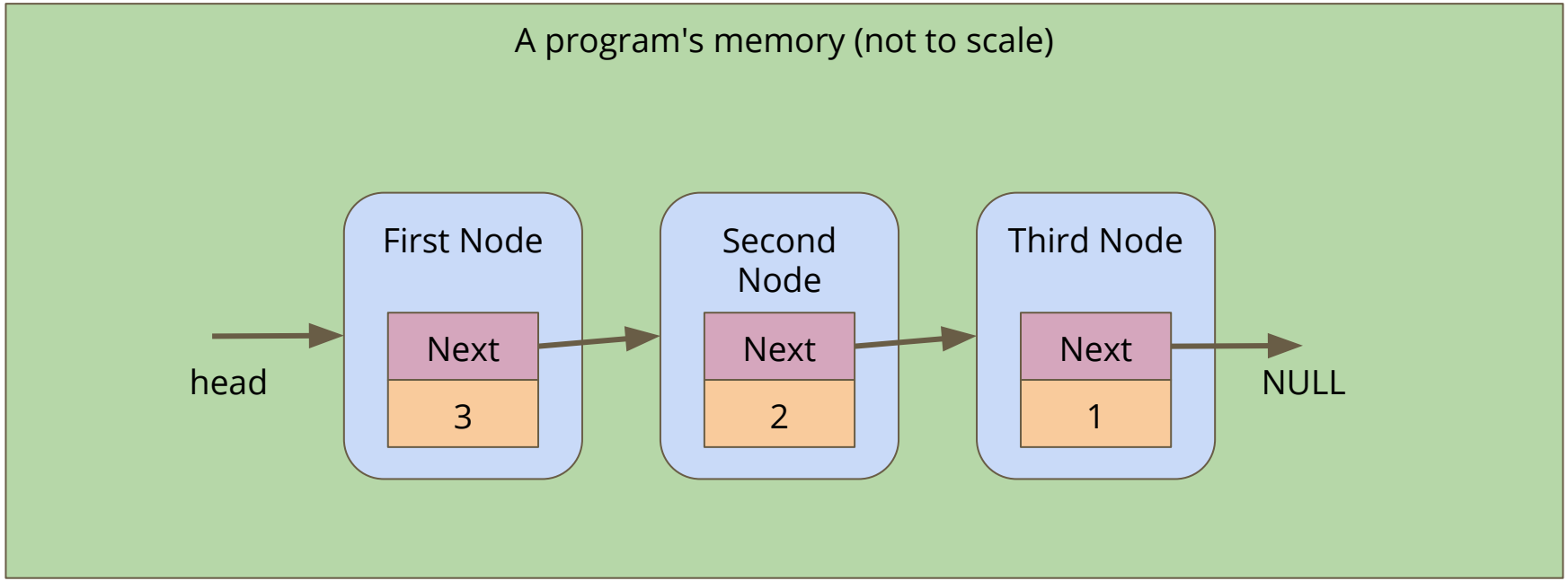
Removing a node

Find the node we want to remove

- Loop through until we find our matching node
- Set the **next** pointer from the previous node to point to the node after our node, skipping it
- Then **free** the node we're removing

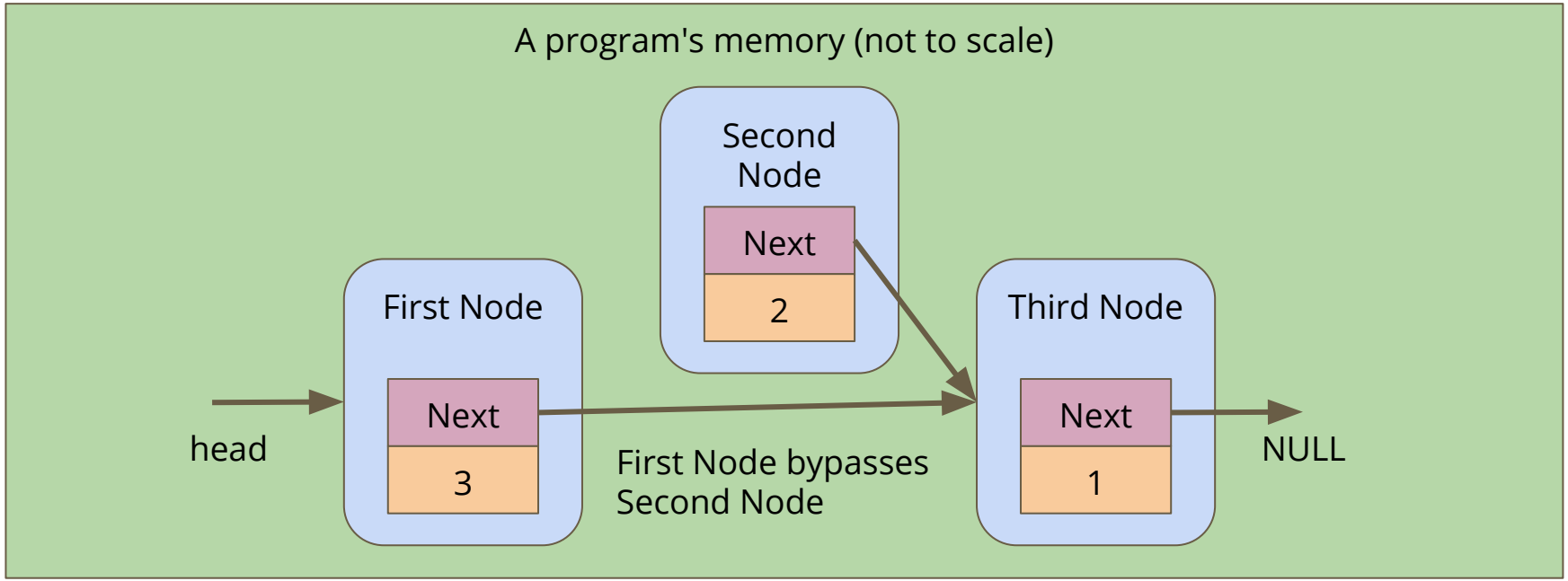
Removing a node

If we want to remove the Second Node



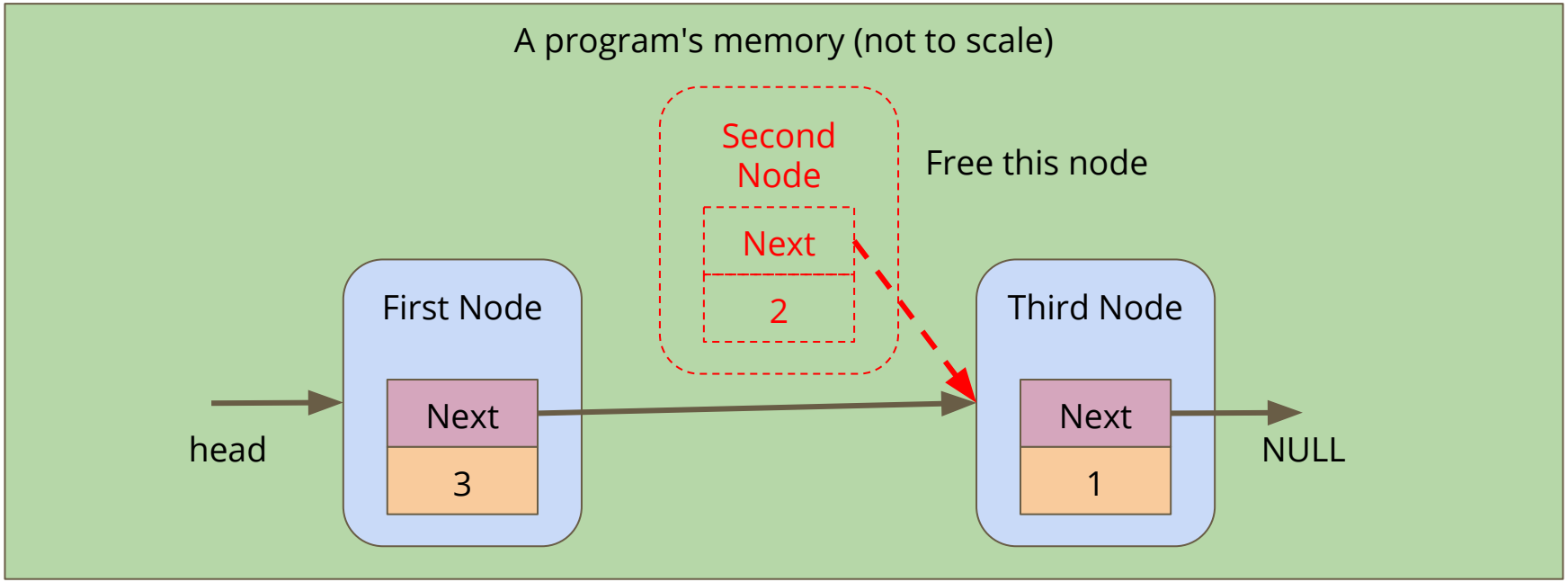
Removing a node

Alter the First Node's **next** to bypass the node we're removing



Removing a node

Free the memory from the now bypassed node



Removing a Node

```
struct node *removeNode(struct node* head, char name[]) {
    struct node *previous = NULL;
    struct node *n = head;
    // loop through until we've found our node (or it isn't in the list)
    while (n != NULL && strcmp(name, n->name) != 0) {
        previous = n;
        n = n->next;
    }
    if (n != NULL) { // found the node
        if (previous == NULL) { // it's the first node
            head = n->next;
        } else {
            previous->next = n->next;
        }
        free(n);
    }
    return head;
}
```

Let's play a game

Once our list is created, we can play

- We'll tell the game who's been knocked out
- Then our program will find the person and remove them from the list

```
// A game loop that runs until only one player is left
while (printPlayers(head) > 1) {
    printf("Who just got knocked out?\n");
    char koName[MAX_NAME_LENGTH];
    fgets(koName, MAX_NAME_LENGTH, stdin);
    koName[strlen(koName) - 1] = '\0';
    head = removeNode(head, koName);
}
printf("The winner is: %s\n", head->name);
```

Cleaning Up

Remember, All memory allocated (malloc) needs to be freed

- We can run dcc --leakcheck to see whether there's memory getting leaked
- What do we find?
- There are pieces of memory we've allocated that we're not freeing!

Let's write a function that frees a whole linked list

- Loop through the list, freeing the nodes
- Just be careful not to free one that we still need the pointer from!

Code to free a linked list

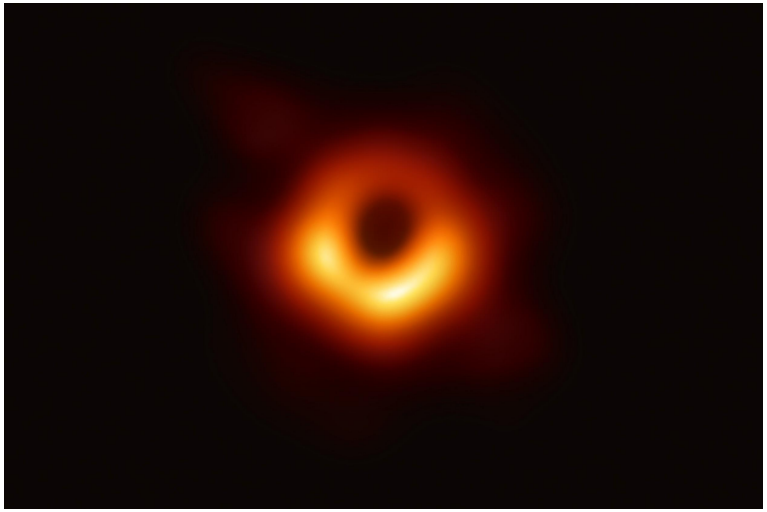
```
// Loop through a list and free all the allocated memory
void freeList(struct node *head) {
    while(head != NULL) {
        // keep track of the current node
        struct node *thisNode = head;

        // move the looping pointer to the next node
        head = head->next;

        // free the current node
        free(thisNode);
    }
}
```

Break Time

A thought exercise . . . the future



- Why are you doing computer science (or related field)?
- Is there something you'd like to do with these skills?
  - Jobs?
  - Research?
  - Change the World?
- How do you want to use your time at UNSW to push yourself towards your goals?
- Note: You don't need all the answers yet, but it's useful to start thinking

Course Timetable around Public Holidays

We have some public holidays coming up:

- **19th April** - No Tutorials or Labs
- Tutorials and labs will run on **Wednesday 1st May**

- **22nd April** - No Tutorials or Labs
- Tutorials and labs will run on **Monday 29th April**

- **25th April** - No Lecture, Tutorials or Labs
- Lecture, tutorials and labs will run on **Tuesday 30th April**

Attending other Tutorials or Labs

If you don't have a Tutorial/Lab because of a Public Holiday

- You can sit in on another Tutorial or Lab
- You should ask a Tutor politely if you can join their Tutorial and/or Lab
- Please only do this if there is enough space for you to do so
- People scheduled in a time slot take precedence for space
- You should still attend your own Tutorial and Lab when they happen

You do not need to attend another tute/lab, everyone will still have the same number of classes, they just might be scheduled later!

Assignment 2 - The Pokémon Universe

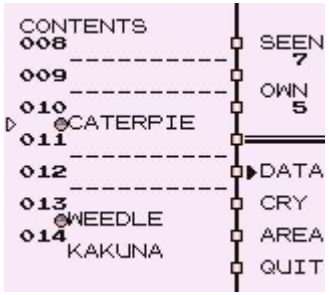
What is a Pokémon?

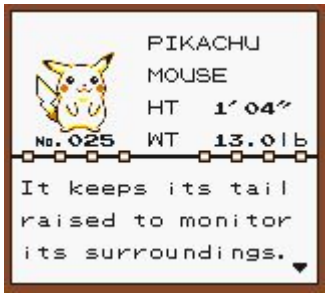
- A Pokémon is a fictitious creature made by Nintendo
- Pokémon are part of a massive gaming franchise based on their capture, training and use in battle
- CSE and UNSW does not condone the capture of wild animals for use in combat (and Pokémon need to have a good look at their ethics :P)

Assignment 2 - The Pokédex

What is a Pokédex?

- A Pokédex is an Encyclopedia that catalogues Pokémon
- One of the aims of the Pokémon games is to "Catch 'em all"
- When you encounter Pokémon in the game, you will be able to update your Pokédex with information about it





How does our Pokédex work?

If you ever have this question, you can use the reference solution

- "1511 pokedex_reference"
- Remember the ? command will always show the list of commands
- We can add and remove Pokémon
- We can mark Pokémon as found
- We can list our entire Pokédex
- and much more . . .

Assignment 2 - Your Pokédex

You will be creating an implementation of a Pokédex

- We have provided a framework of files to work in
- You will only need to edit "pokedex.c" and "test_pokedex.c"

- Header and C files - you'll see more of this in the future
  - Header Files contain declarations
  - C Files contain definitions

- Compiling multi-file projects involves compiling all of the C files
- Like so: "dcc -o pokedex main.c pokedex.c pokemon.c"
- Or: "dcc -o test_pokedex test_pokedex.c pokedex.c pokemon.c"

What's in the files?

pokemon.c and pokemon.h

- Code for Pokémon themselves
- A struct for a Pokémon (that you can't directly access)
- A series of functions that allow you to create and manipulate Pokémon
- You can safely ignore pokemon.c and just use what's written in pokemon.h
- pokemon.h describes all the functions and what you can do with them
- It's reasonably similar to using functions from stdio.h

What's in the files?

pokedex.c and pokedex.h

- Similar to pokemon.c and pokemon.h
- Except pokemon.c is **incomplete!**
- Use pokemon.h to read the descriptions of what you must implement in pokemon.c

What's in the files?

main.c vs test_pokedex.c

- main.c is the interactive program that allows you to manually use the Pokédex
- test_pokedex.c (also contains a main function) allows you to automate testing
- You will be marked on the usefulness of the tests you have written in test_pokedex.c

Work in Stages

We have provided a staged process for your assignment work

- Progress through the stages in the order provided
- They're designed to work that way
- You will need knowledge of Linked Lists to create your Pokédex

Focus your attention

- Think about only one stage at a time
- Add features one at a time to save from confusing yourself

Working with your Linked List

The Linked List starts partially implemented

- The pokenode struct is already set up as a linked list node struct
- You will be expected to make some functions that use and modify the linked list
- As you progress, you will find you need to make the struct more complicated
- Add complexity only by necessity!

Testing

test_pokedex.c has some tests in it already

- You can run this to test some of the early stages functionality
- test_pokedex.c is not complete!
- However it does show you a nice way of setting up automated testing of individual functions
- This is often called **"Unit Testing"**

Assert

A valuable tool in testing

```
#include <assert.h>
// Asserts will test a code expression
int main (void) {
    int number = 2;
    int result = number * 3;

    // if this assert is false, the program will end here
    assert(number == 6);
}
```

We can use asserts to force our code to exit if one of our assumptions turns out to be false. If our program is running correctly, our asserts have no effect

Marking

Pass Mark - Readable Code that implements Stage 1

- Add nodes to a linked list
- Extract information from individual nodes, including whether it has been found
- Be able to traverse a linked list, extracting information as you go

Credit - Stage 2 with Testing

- Stages 1 and 2 working
- Testing in test_pokedex.c that proves that all functionality works
- Care taken in cleaning memory

Marking continued

Distinction - Stage 3 and more

- Completion of stages 1-3 plus some of 4-5
- Looping through a list and counting up data
- Very readable and re-usable code
- Comprehensive testing that shows the ability to deal with different situations

Marking continued

High Distinction - Full Functionality

- Implement all functions
- Creation of sub-lists, smaller Pokédexes that are compiled from your main Pokédex
- Testing full ranges of possibilities including issues that might not come up often, but could cause functionality issues
- Clean, clear, understandable code that is fully explained where necessary

What did we cover today?

Linked Lists

- Removal
- Cleaning up our used memory

Pokédex

- Assignment Structure
- How to approach it
- How it is assessed