COMP1511/1911 Programming Fundamentals

Week 7
Lecture 2

Reminders

- Assignment 1 was due yesterday
- Lab Exam next week
  - Register for it
    - Online students only
  - Please only come in the second half
  - https://buytickets.at/comp1511unsw/2401097

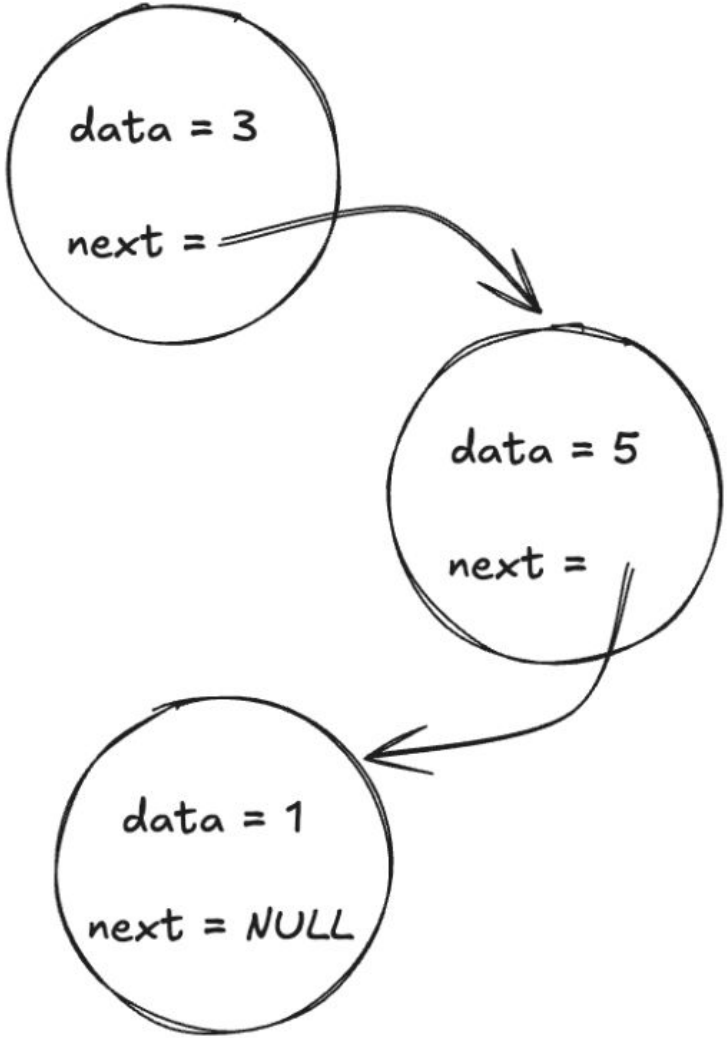


Previously On...

- Pointers recap
- The heap and malloc
- Multi-file projects

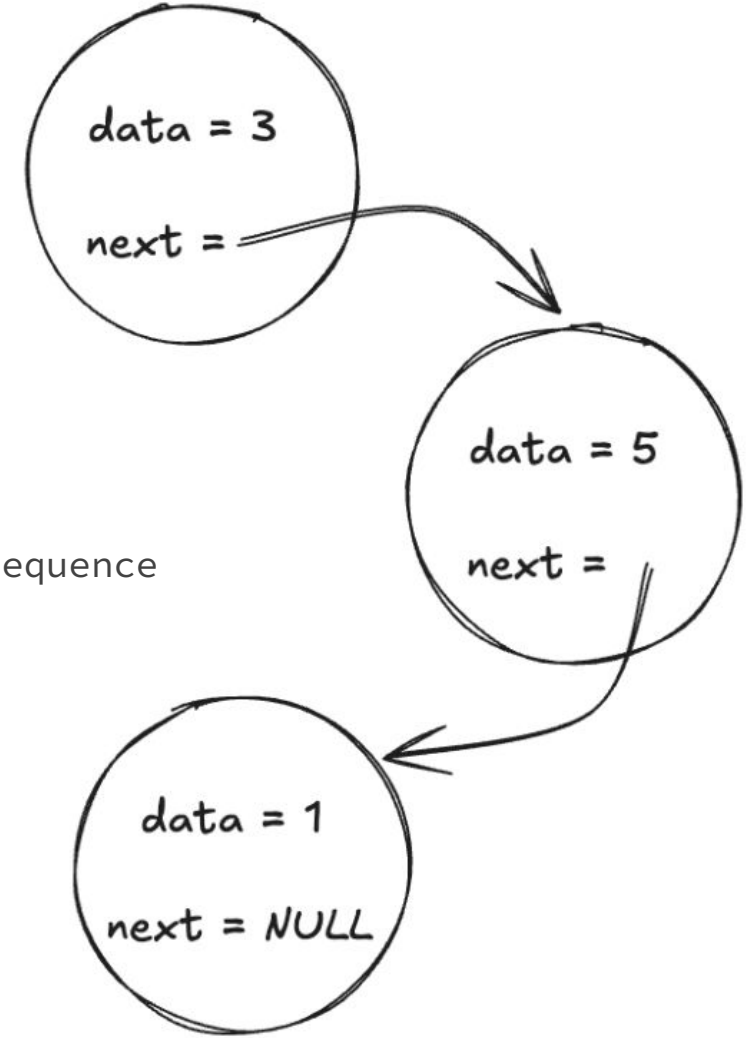
Today

- The G.O.A.T is finally here
  - LINKED LISTS!!!
- (Shoutout Sofia who I stole all of today's drawings from)



Linked Lists

- Linked lists are a collections of nodes
  - Each node contains two elements
    - The data stored in the node
    - A pointer to the next node in the sequence



Why Bother?

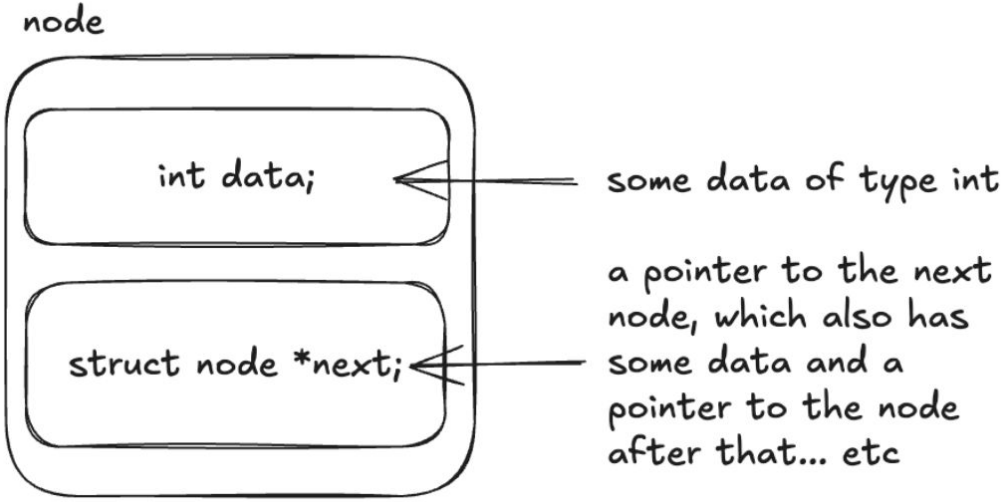
- Linked lists are dynamically sized
  - We can grow and shrink them as needed - efficient for memory!
- Elements of a linked list (called nodes) do NOT need to be stored contiguously in memory
- We can add or remove nodes as needed anywhere in the list, without worrying about size (unless we run out of memory of course)
- We can change the order in a linked list, by just changing where the next pointer is pointing to!

Caveat

- Unlike arrays, linked lists are not random access data structures!
  - You can only access items sequentially, starting from the beginning of the list.

Nodes

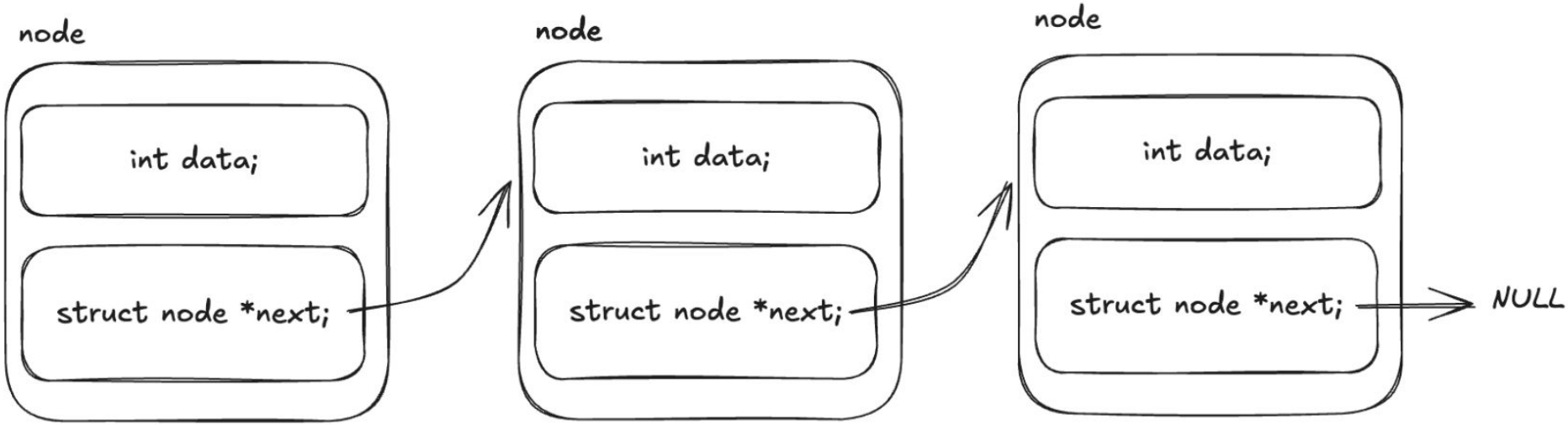
- Each node has some data and a pointer to the next node (of the same data type), creating a linked structure that forms the list.




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

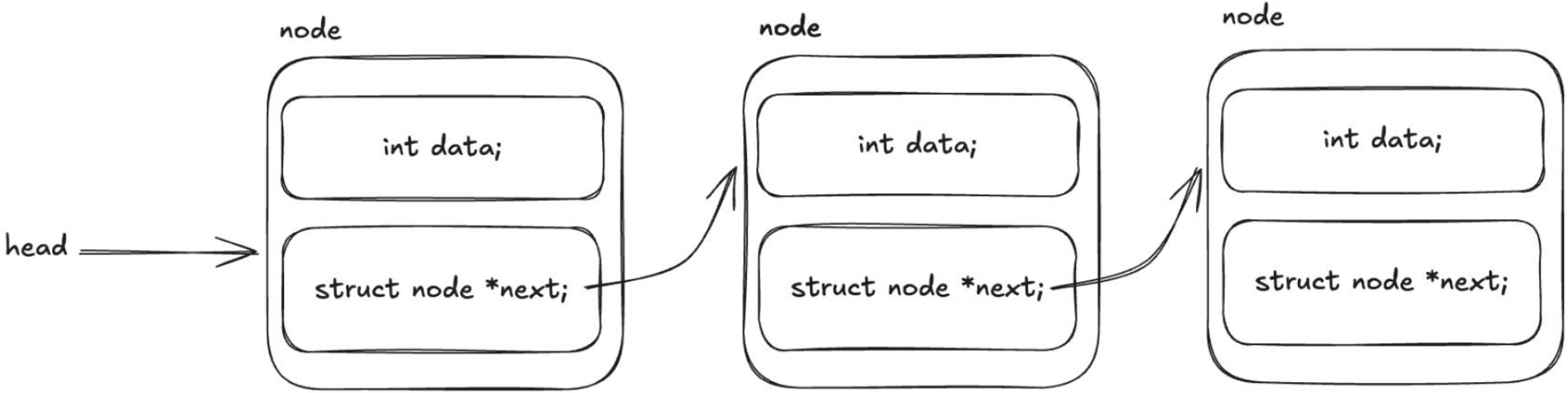
Nodes and Linked Lists

- 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.



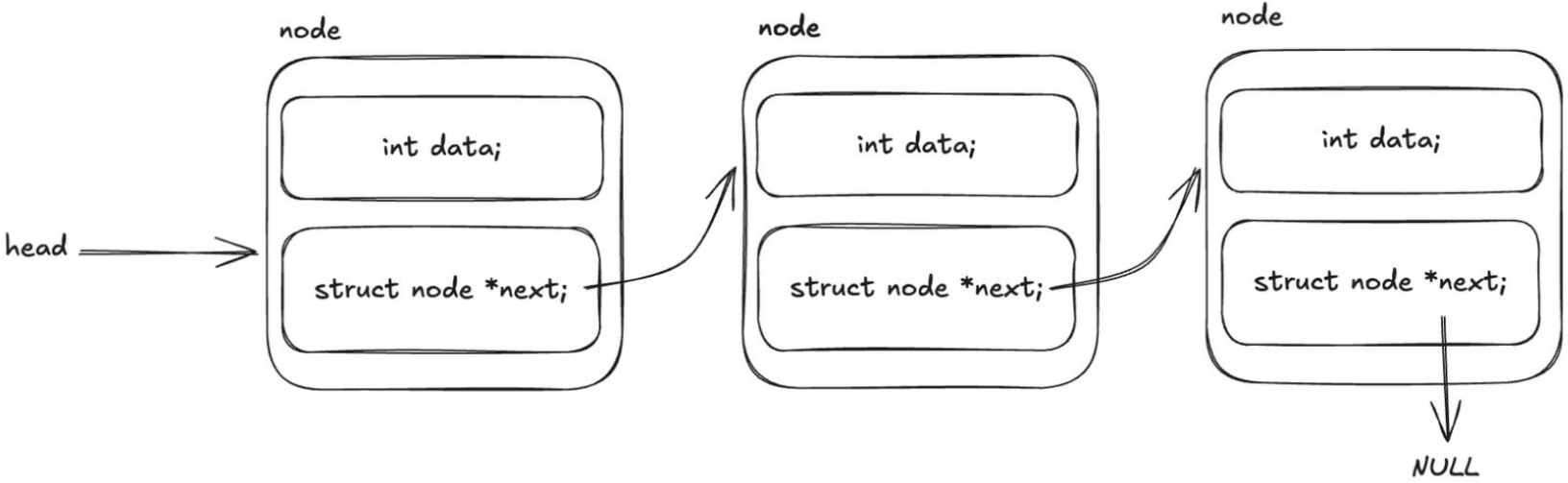
The Head

- To keep track of our list we keep a “head” pointer, which is a pointer to the first node in the list
  - The head pointer is not a node itself, but the memory address of where the first node is.

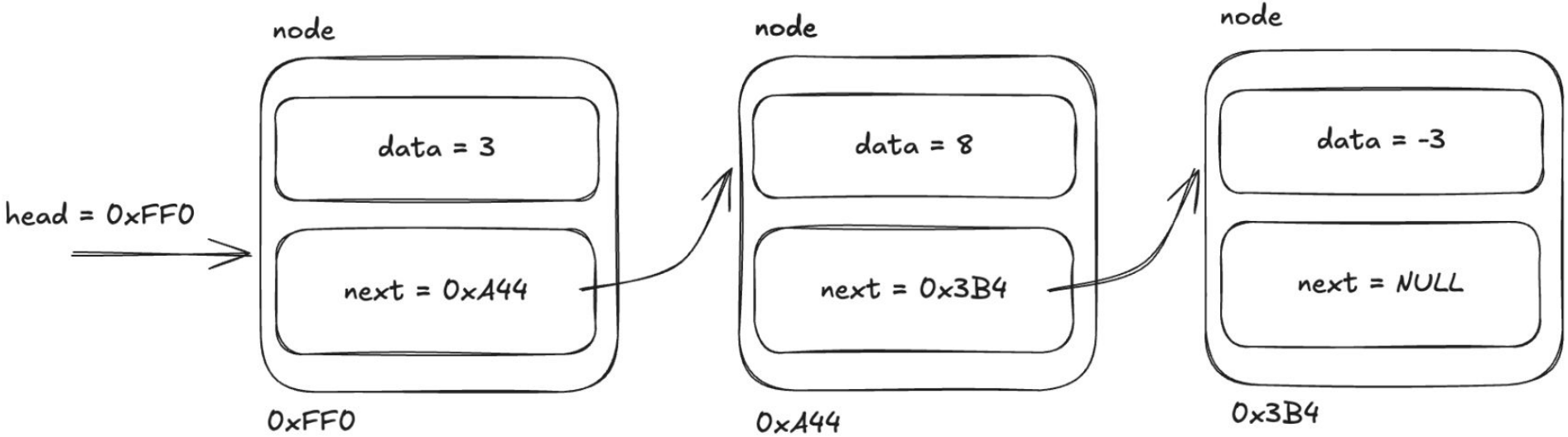


The Tail

- The last node of a linked list is known as the “tail” node and it identified as the node thats next pointer, is pointing to NULL



Example List



Example List

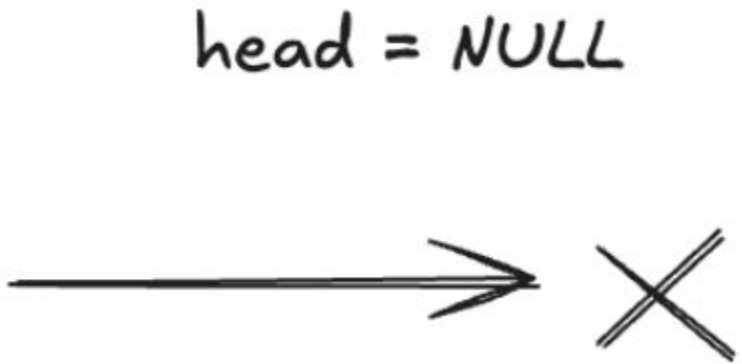
- In order to create a linked list, we would need to

  1. Define struct for a node
  2. A pointer to keep track of where the start of the list is
  3. A way to create a node and then connect it into our list

Example List

- Let's say we wanted to create a linked list with 5, 3, 1

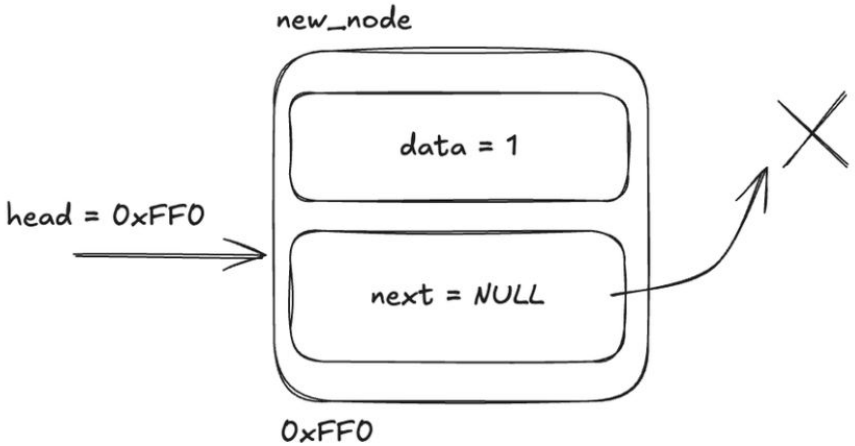
1. Let's create the first node to start the list.
2. A pointer to keep track of where the start of the list is and by default the first node of the list
    - It will point to NULL as there are no other nodes in this list.



Example List

---

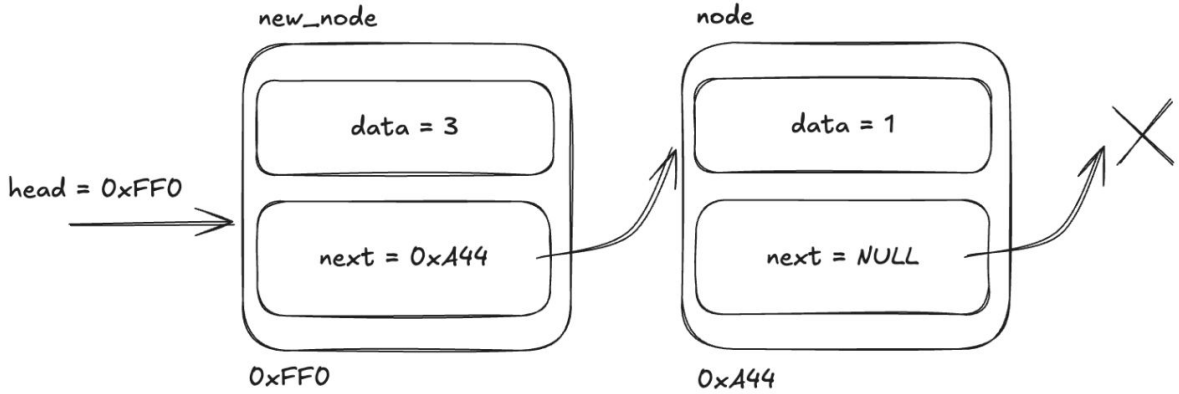
1. Create the next node to store 1 into (you need memory).
2. Assign 1 to data.
3. Make its next pointer point to NULL.
4. Insert it at the beginning so the head would now point to this new node.



Example List

---

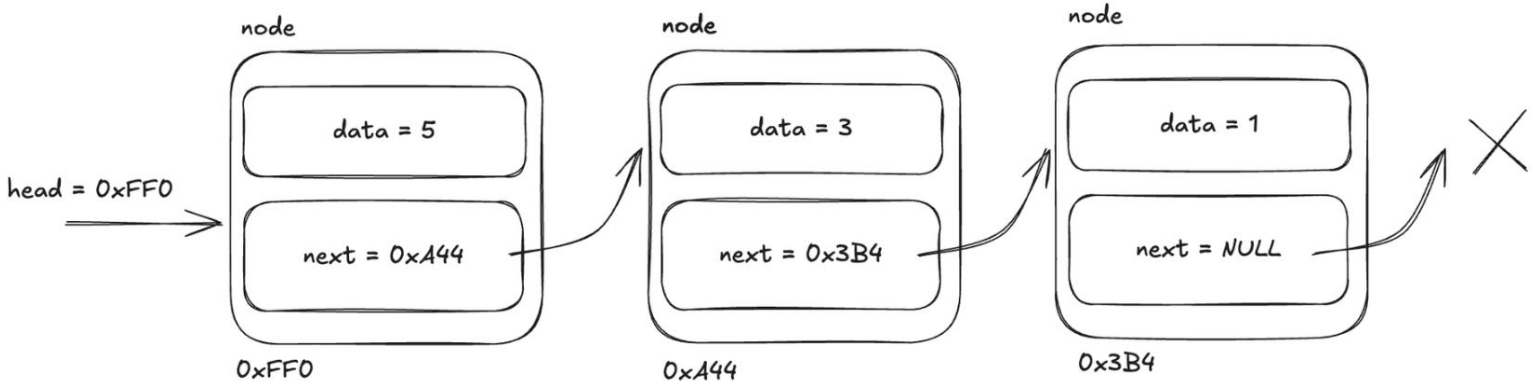
1. Create the next node to store 3 into (you need memory)
2. Assign 3 to data.
3. Make its next pointer point to NULL.
4. Insert it at the beginning so the head would now point to it and the new node would point to the old head



Example List

---

1. Create the next node to store 5 into (you need memory).
2. Assign 5 to data.
3. Make its next pointer point to NULL.
4. 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

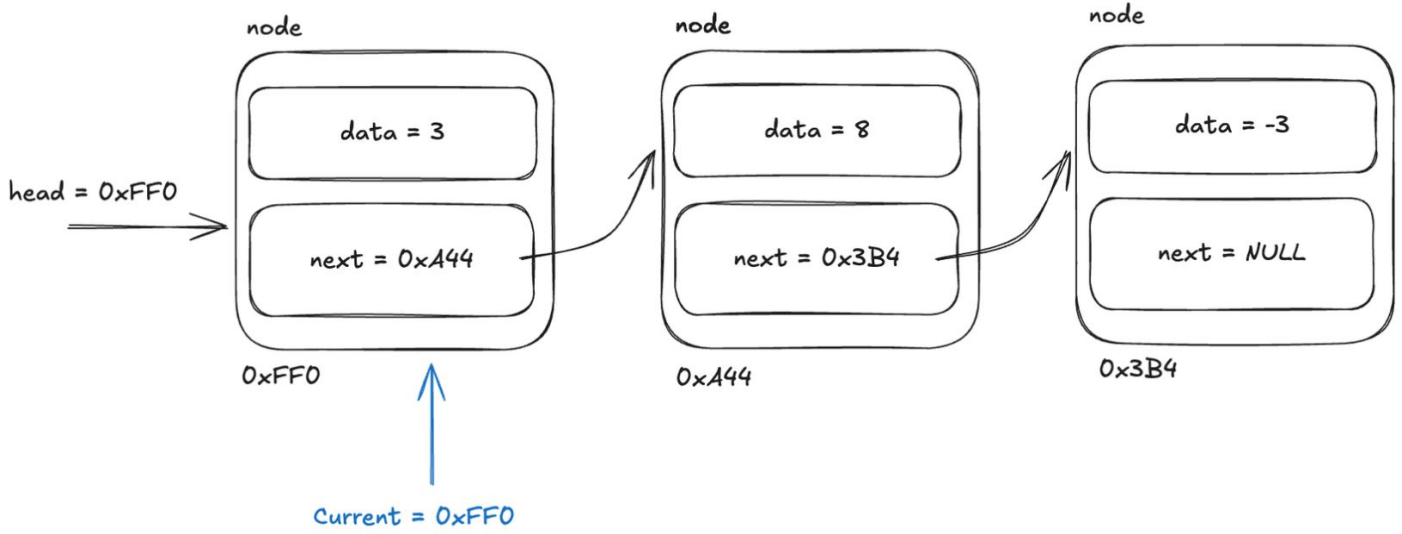
...probably

Moving Through a List

- With arrays, we can just say where we want to go
  - This is because arrays are contiguous in memory
    - To find position 3, we can go to position 0, and move down three places in memory
- Linked lists are NOT contiguous in memory
  - Therefore, traversal needs to start from the head

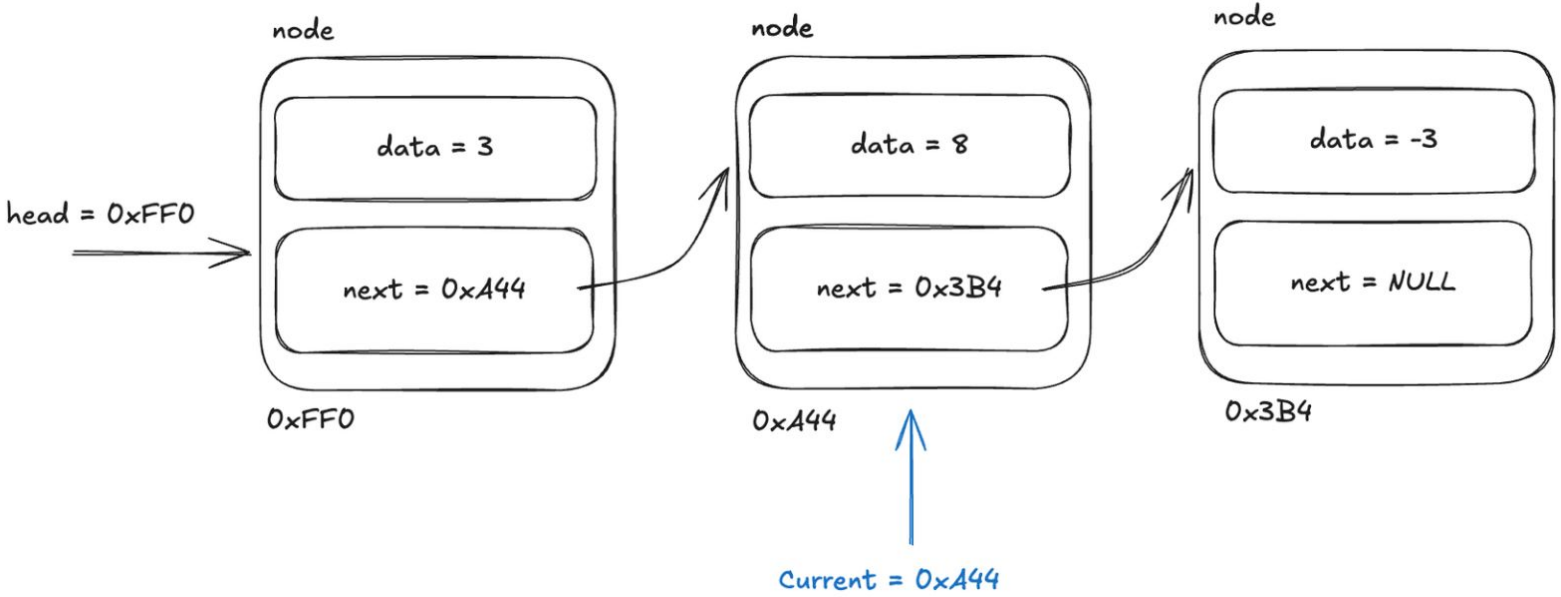
Moving Through a List

- Set your current pointer to the head pointer to keep track of where you are currently located
  - `struct node *current = head;`



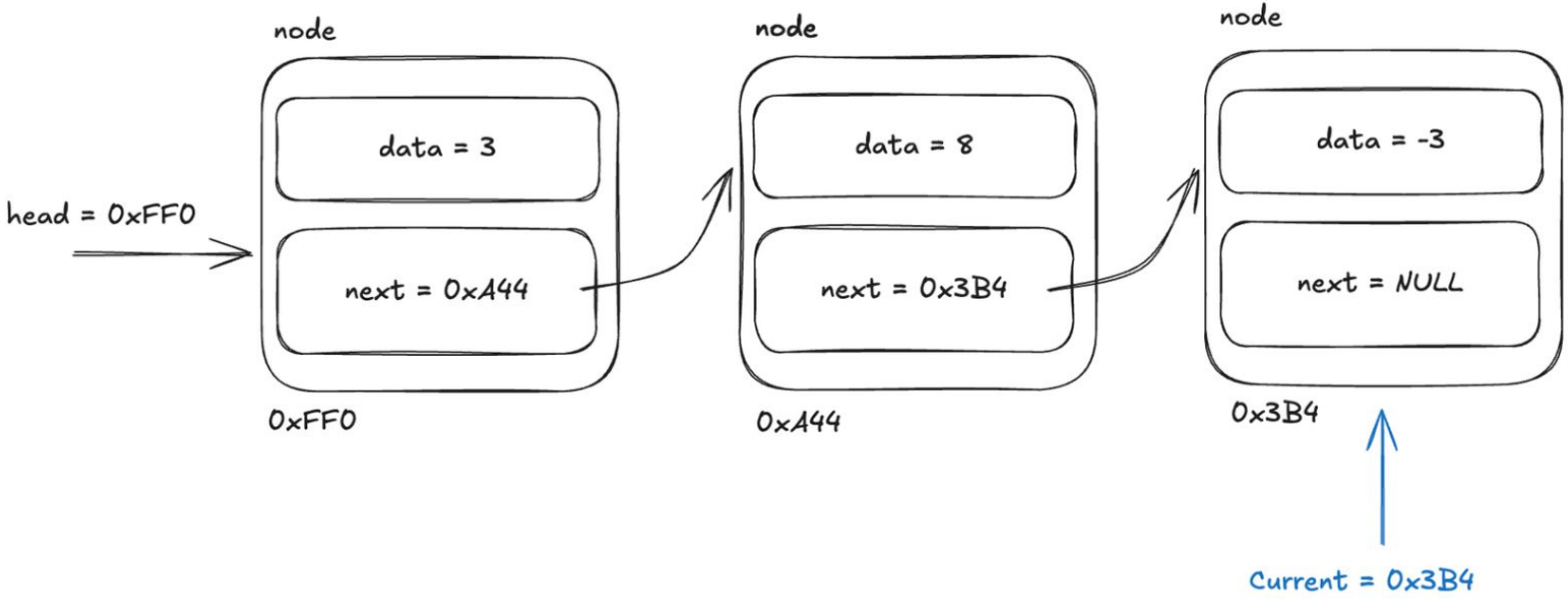
Moving Through a List

- Now how will we move the current along?
  - `current = current->next;`



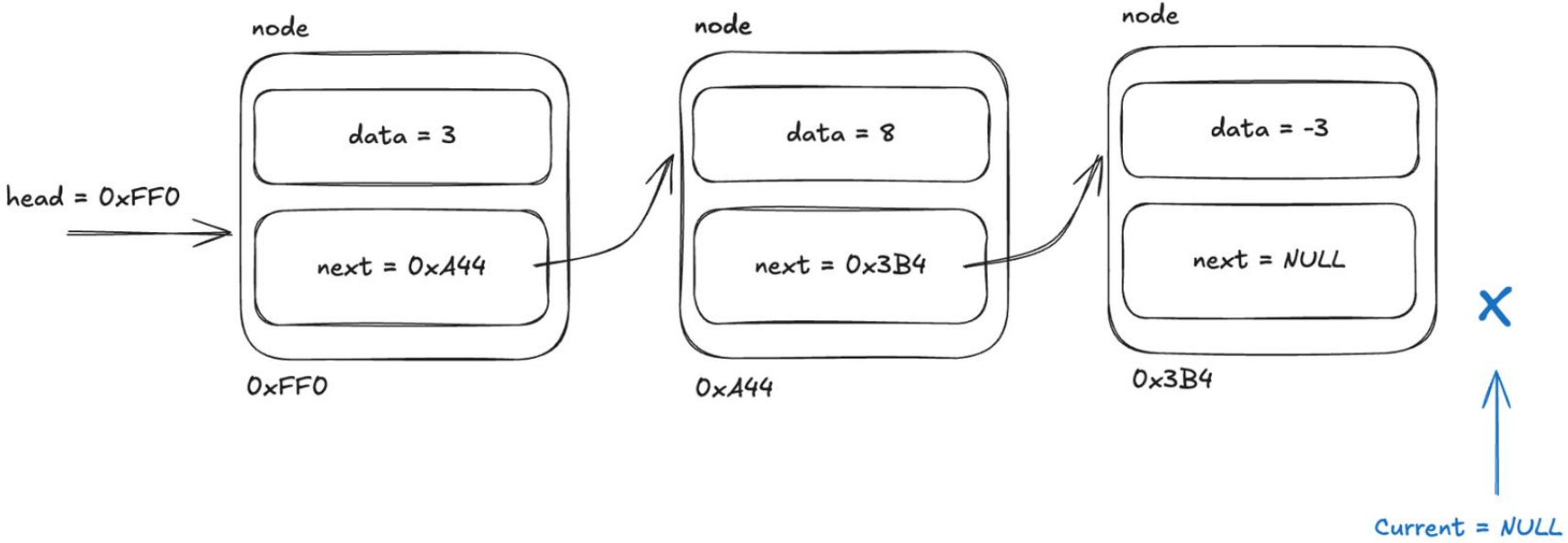
Moving Through a List

- Now how will we move the current along?
  - `current = current->next;`



Moving Through a List

- When should I stop moving?
  - `while (current != NULL)`



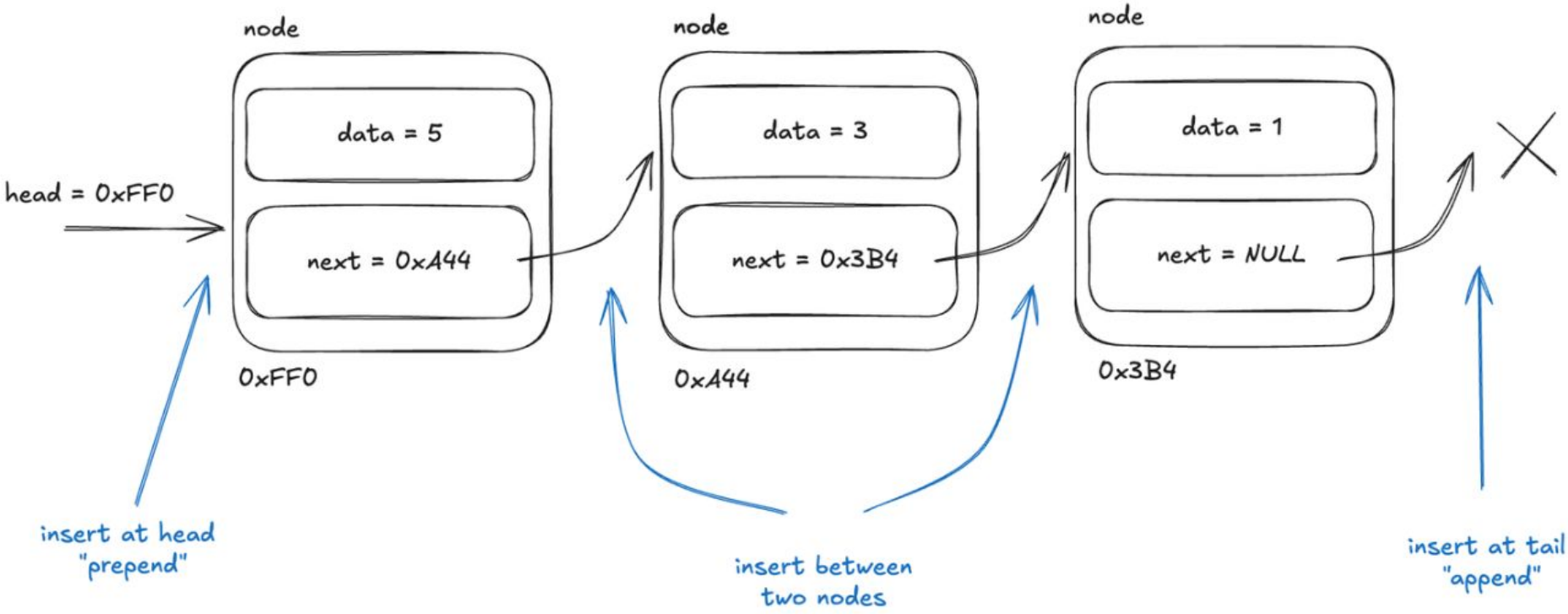
Traversing a Linked List

- 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.
- This is known as “traversing” a linked list.

Inserting into a Linked List

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

Inserting Into a Linked List



What We Learnt

- Linked lists
  - What they are
  - How to traverse them
  - How to insert into them

Next Time

- More linked lists
  - Deletion
  - General exercises

Reach Out

- Check the course forum for questions!
  - Post there too!
- Admin questions
  - cs1511@unsw.edu.au

Thank You

Questions?