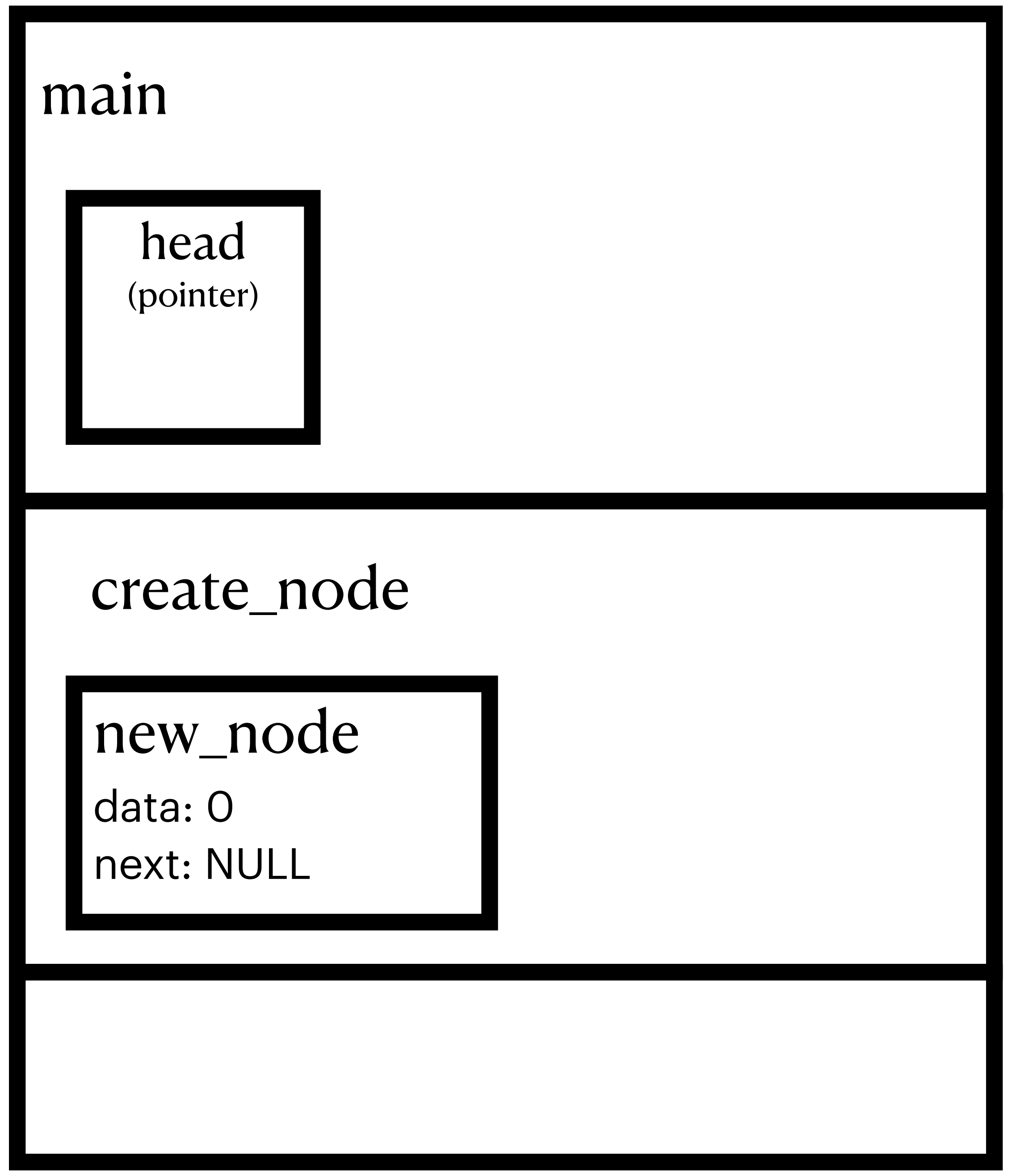
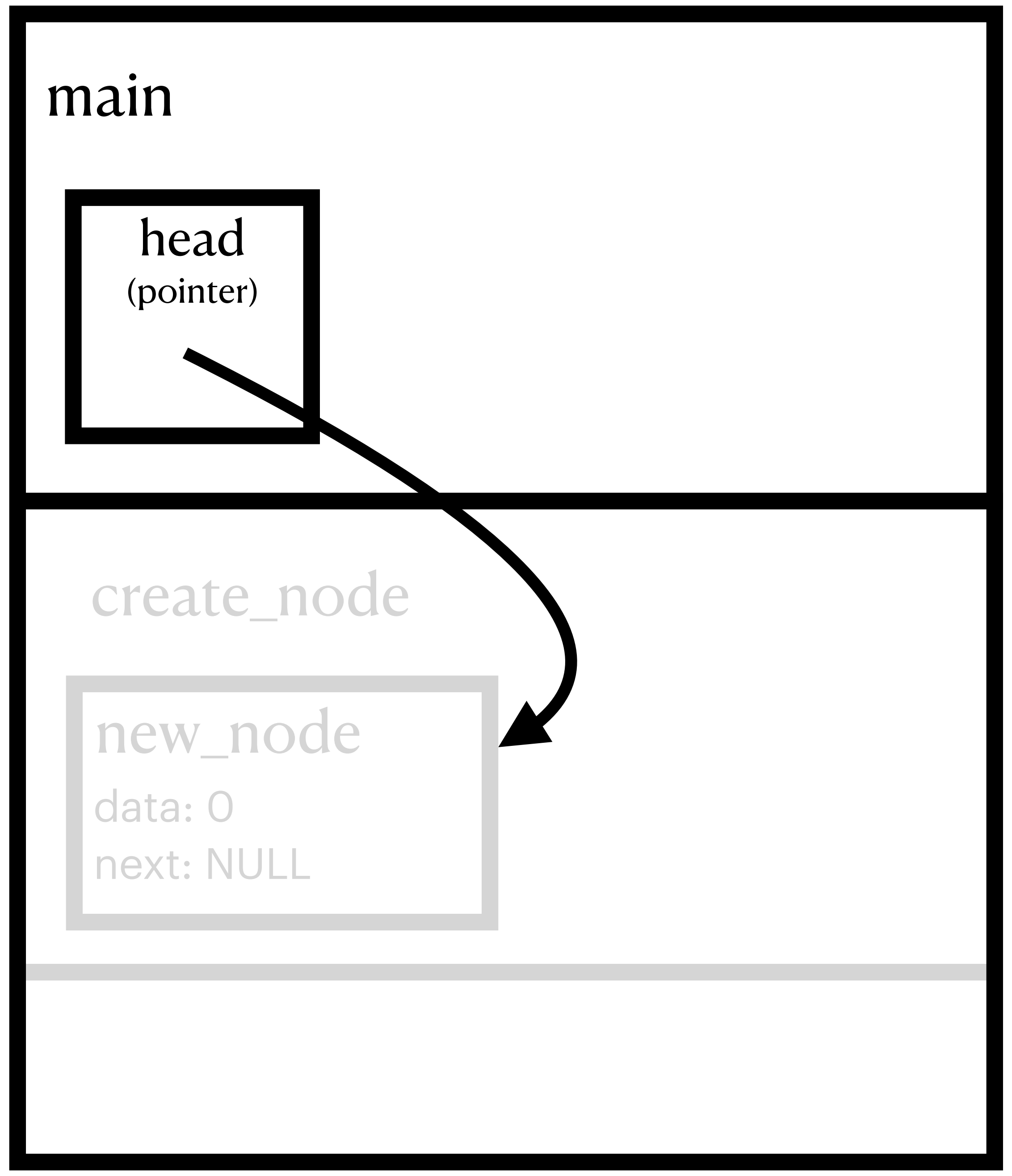
Returning an address without `malloc`

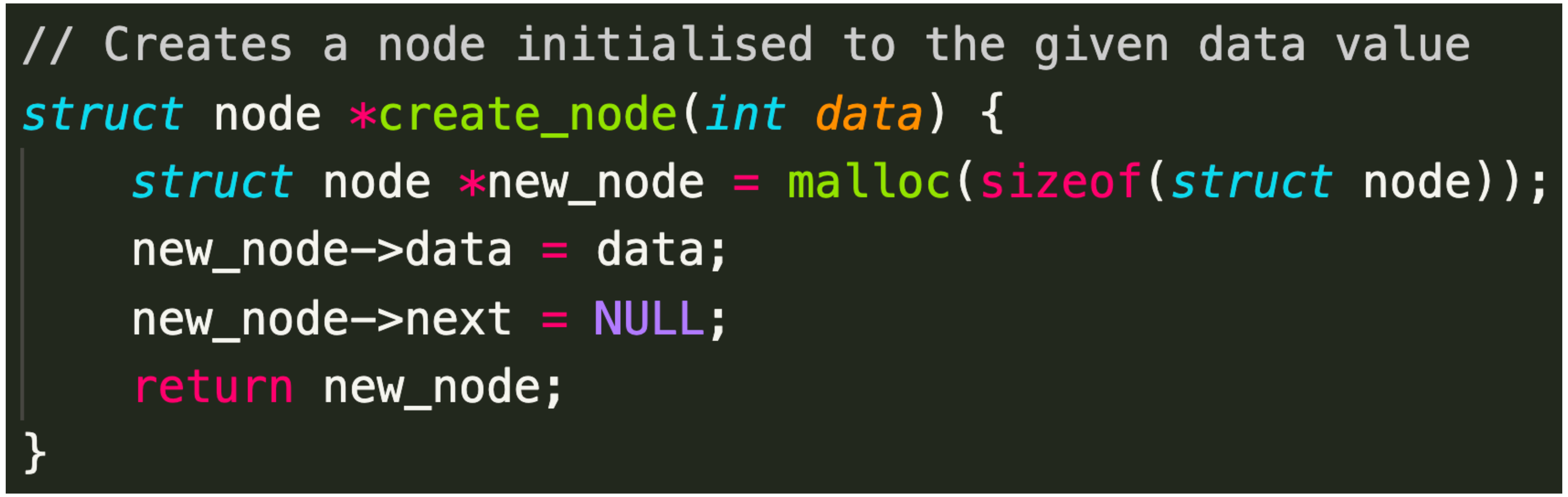




`create_node` creates a **`struct node`** called **`new_node`** and initialises its fields.

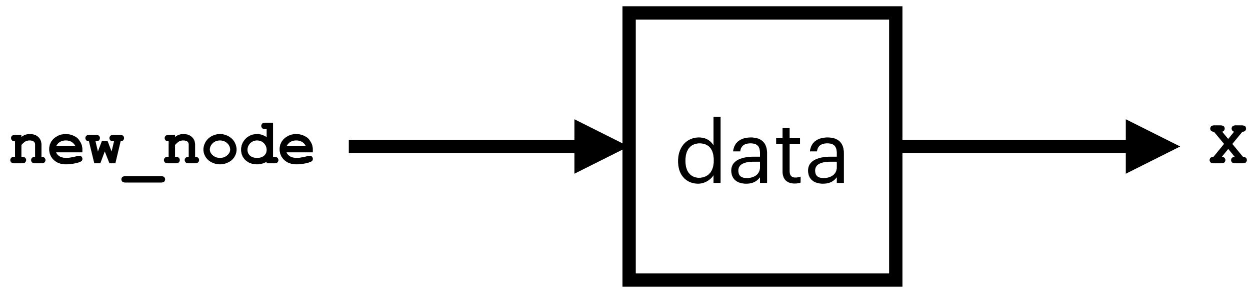
when we exit **`create_node`**, its memory is deallocated and the address of **`new_node`** is stored in **`head`**.
now **`head`** is left pointing at memory that is unsafe to access.

Creating a Node

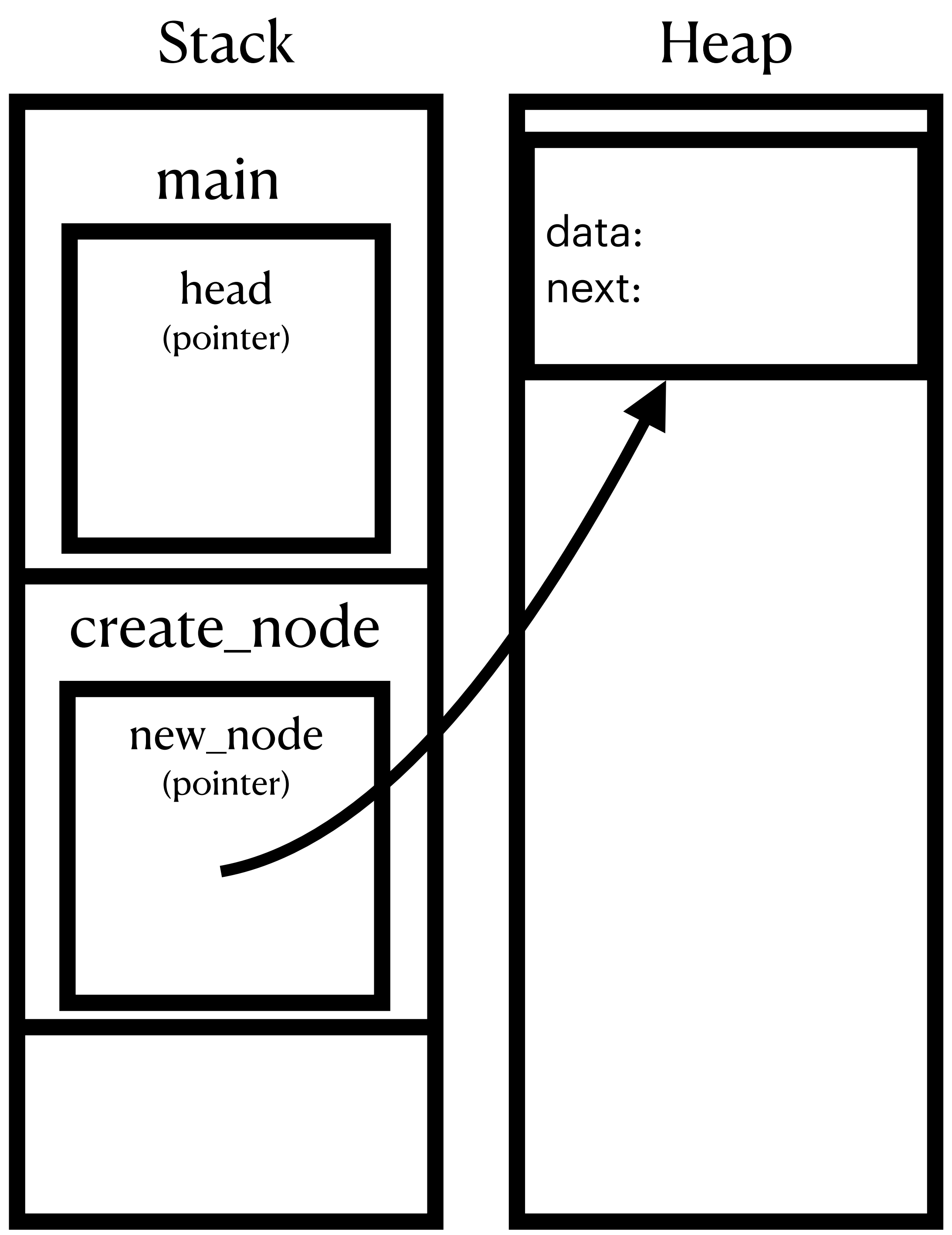


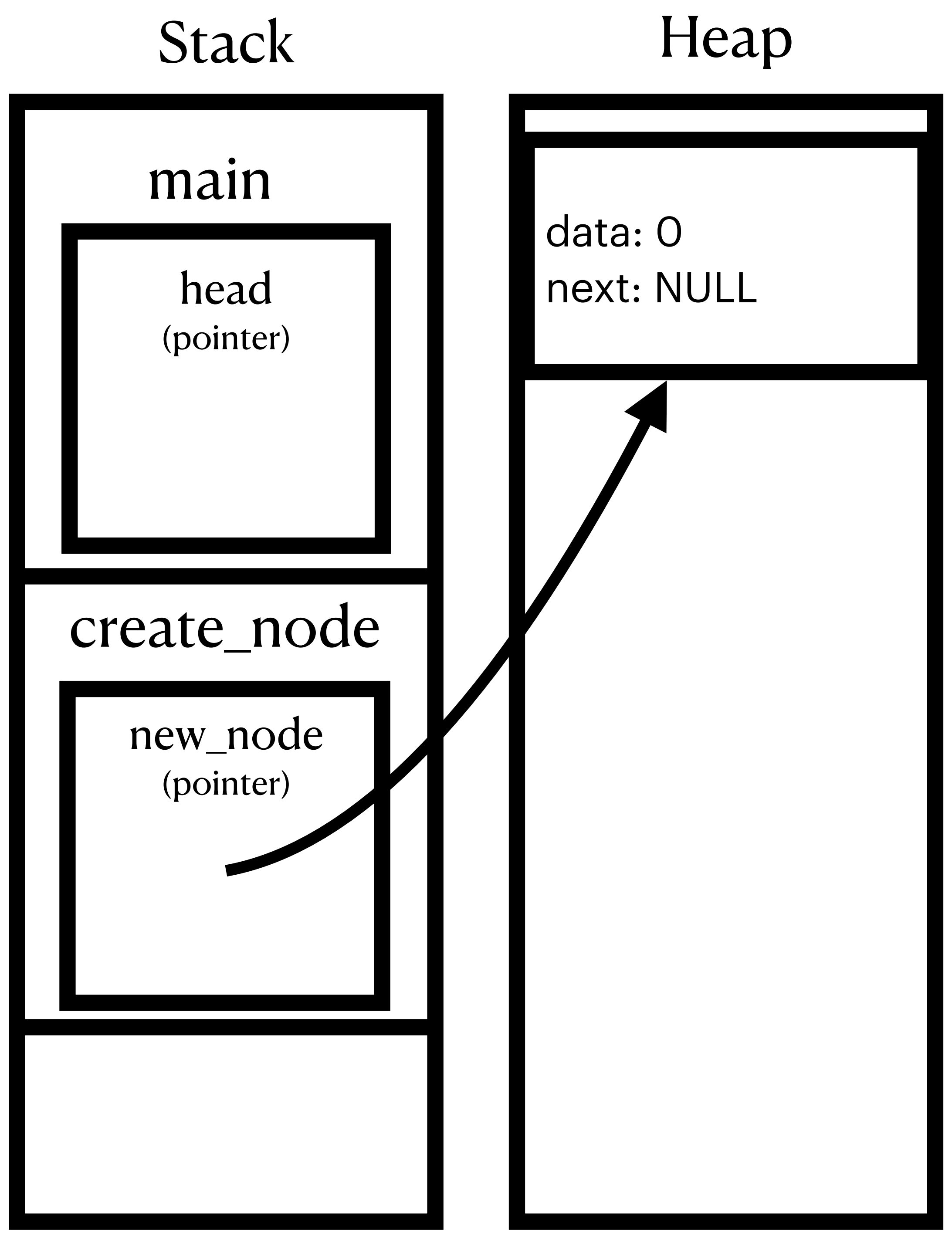
```c
// Creates a node initialised to the given data value
struct node *create_node(int data) {
    struct node *new_node = malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}
```

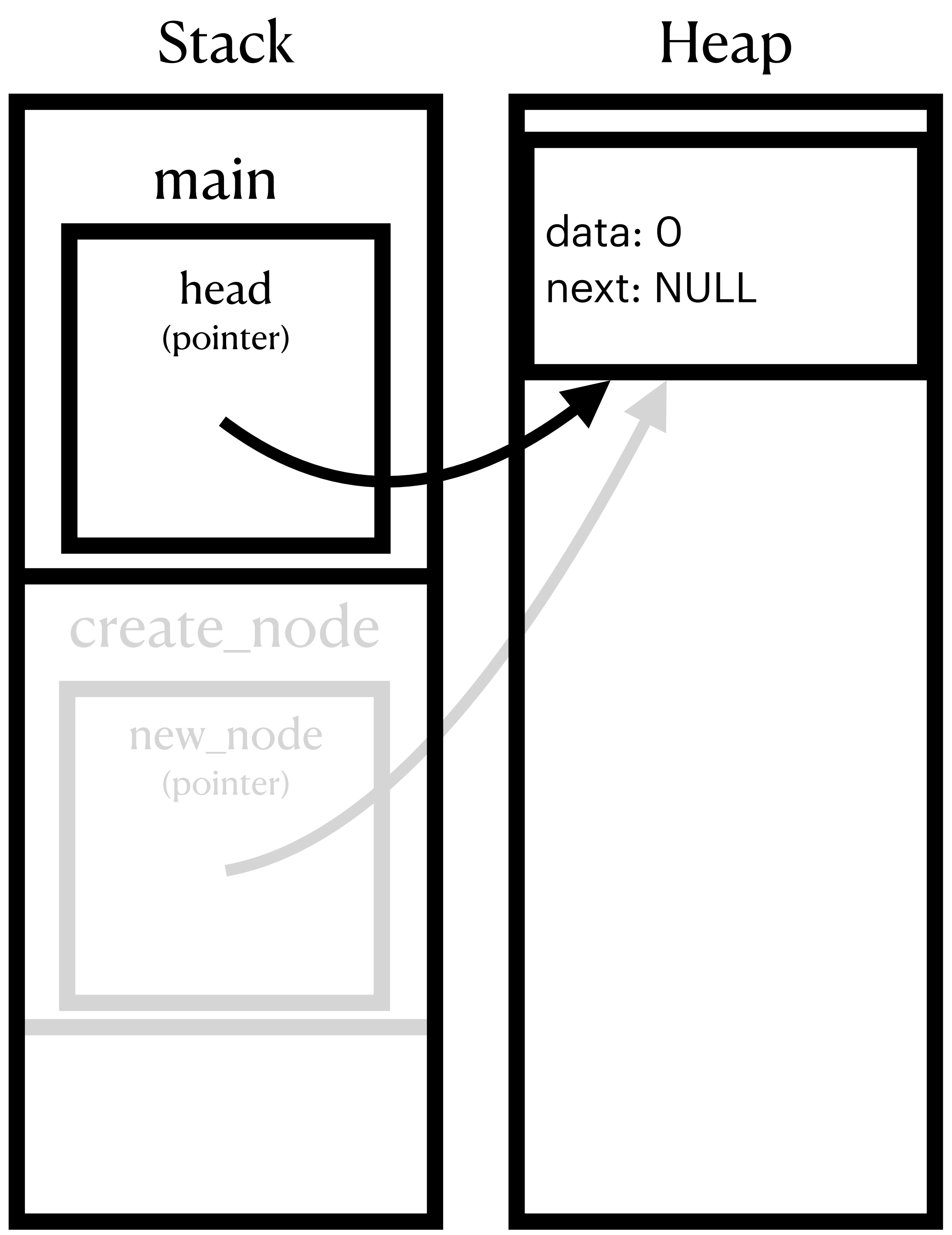
Outputs: a node (or a linked
list with one element!)



Memory Model







`malloc` allocates a block of memory on the heap for a **`struct node`**

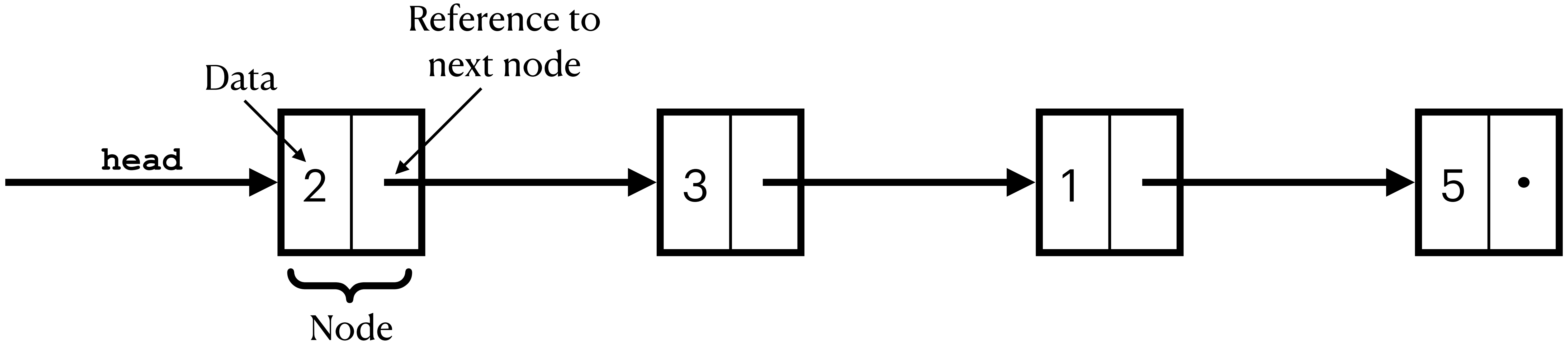
The struct fields are initialised by dereferencing the **`new_node`** pointer

`create_node` returns the address to the heap-allocated struct, which is copied into the **`head`** variable. **`head`** points to the heap-allocated struct.

What is a Linked List

A Data Structure

- A linked list is an ordered collection of nodes. Every node contains a piece(s) of data, and (with the exception of the last node) a reference to the next element in the list.
- We can visualise it like so



- Here, we have a *list* of 4 nodes *linked* together by references.
- We are usually given where the first node of a list is through the **`head`** pointer.
    - This is our entry point into the list, giving us access to every node.

Linked Lists vs Arrays

Is one better than the other? It depends.

Array	Both	Linked Lists
• We are able to access any piece of data immediately using an index. • Memory efficient — we only need to store one thing at each index.	• We can store many variables that are associated with each other under one variable name. Through just one entity in our code, we can access multiple pieces of data.	• We don't need to know the size of the list upfront and can change the size of the list with ease. • Inserting at the start of the list is fast