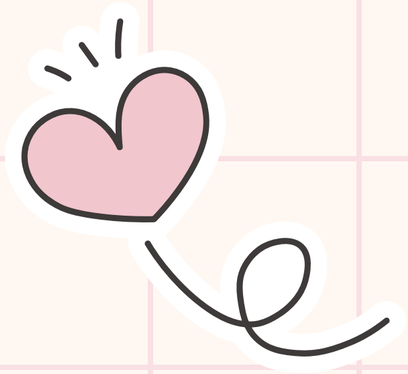
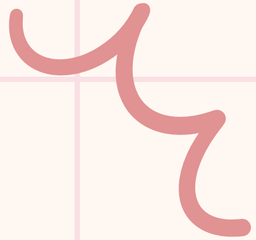
COMP1511 Programming Fundamentals





POINTERS & MEMORY RECAP
+
INTRO TO "LINKED LIST"

WEEK 7 LECTURE 1

with Tammy

(for Wk7 Mon - Wk9 Mon)

(Sasha is at a conference across the globe)

Course Admin
Tammy Zhong
She/Her

Announcements

ASSIGNMENT 1 DUE TODAY!

MONDAY 8PM

HELP SESSIONS ARE RUNNING!

ASSIGNMENT 2

RELEASING THIS **THURSDAY**

ALL ABOUT LINKED LISTS

(ONLY 2-DAY BREAK SORRY)

ASSIGNMENT 2 LIVESTREAM

NEXT TUESDAY

1:00PM

113 SEMINAR RM K17

+ YOUTUBE

Announcements

EASTER CATCH UP CLASSES

IF YOUR TUT-LAB FALLS ON GOOD FRIDAY OR EASTER MONDAY

SIGN UP VIA LINK ON FORUM!

WEEK 8 REVISION SESSIONS

COMING SOON!

-

KEEP AN EYE ON THE COURSE FORUM FOR SIGN UPS

LIVE CODE HERE:

https://cgi.cse.unsw.edu.au/~cs1511/24T1/live/week_7/

TWO WEEKS AGO...

POINTERS & DYNAMIC MEMORY ALLOCATION

THIS WEEK:
INTO THE WORLD OF LINKED LISTS

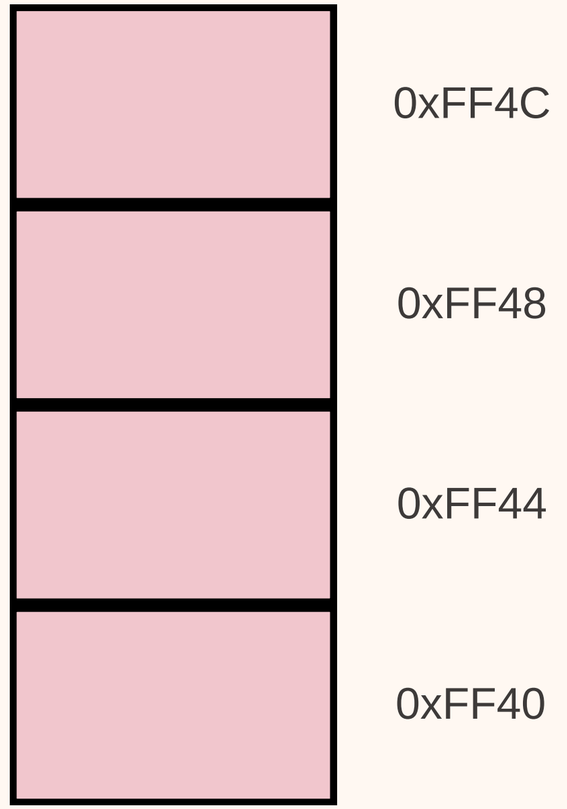
Today:

- Recap (some of) Pointers & Memory
- Intro to Linked Lists
  - insert at head
  - traverse a linked list
  - insert at tail (maybe)

Pointers & Memory Recap

Pointers Recap

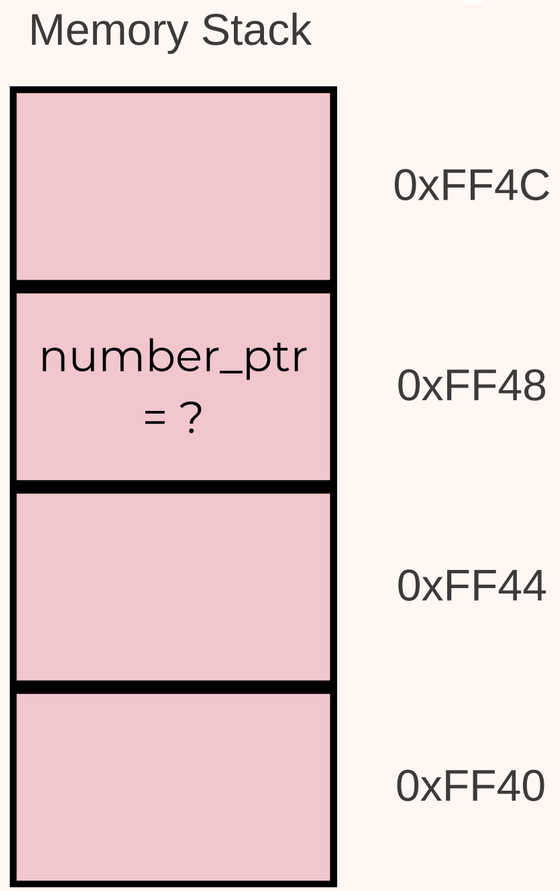
- a type of variable storing a memory address
- can point to any type of data (int, char, struct ... etc.)
- can access the data
  - (“dereference” using *)
- can retrieve the address of the variable the pointer points to
  - (“address of ...” using &)



Pointers Recap

"DECLARE" A POINTER

```
int *number_ptr;
```



Pointers Recap

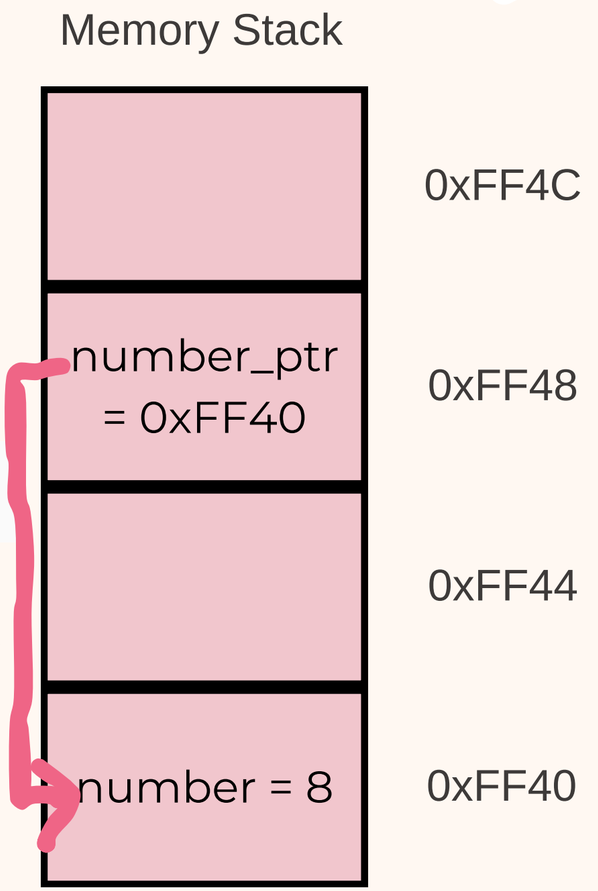
"DECLARE" A POINTER

```
int *number_ptr;
```

"ASSIGN A VALUE" TO A POINTER

```
int number = 8;
number_ptr = &number;
```

"address of ..."



Pointers Recap

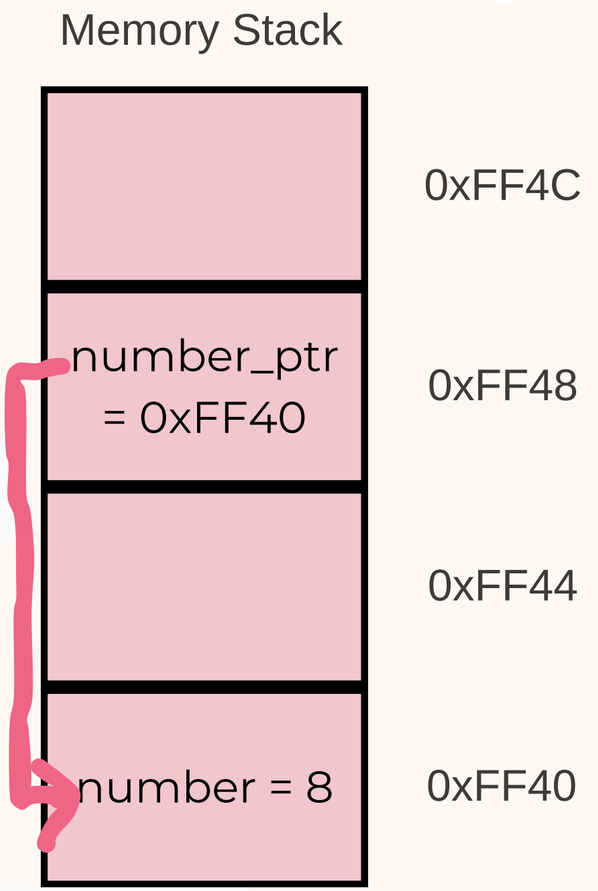
"DECLARE" A POINTER

```
int *number_ptr;
```

"ASSIGN A VALUE" TO A POINTER

```
int number = 8;
number_ptr = &number;
```

"address of ..."



"DEREFERENCE" A POINTER

```
printf("%d\n", *number_ptr);
```

"the value in the variable the pointer is pointing to"

Pointers Recap

"DECLARE" A POINTER

```
int *number_ptr;
```

"ASSIGN A VALUE" TO A POINTER

```
int number = 8;
number_ptr = &number;
```

"address of ..."

THEY ARE DIFFERENT!!!

"DEREFERENCE" A POINTER

```
printf("%d\n", *number_ptr);
```

"the value in the variable the pointer is pointing to"

Pointers Recap

"DECLARE" A POINTER

```
int *number_ptr;
```

"ASSIGN A VALUE" TO A POINTER

```
int number = 8;
number_ptr = &number;
```

"address of ..."

This is a variable type

"DEREFERENCE" A POINTER

```
printf("%d\n", *number_ptr);
```

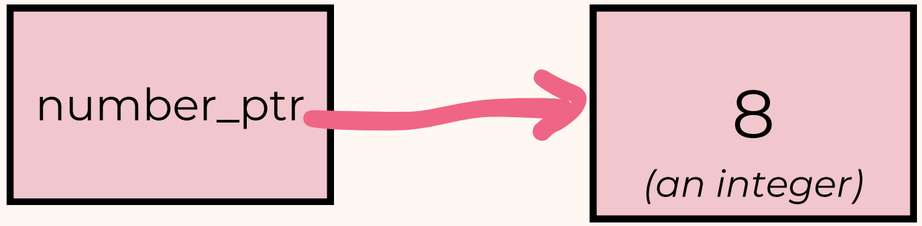
"the value in the variable the pointer is pointing to"

Pointers Recap

We can have different types of pointers...

INTEGER POINTER

```
int *number_ptr;
```

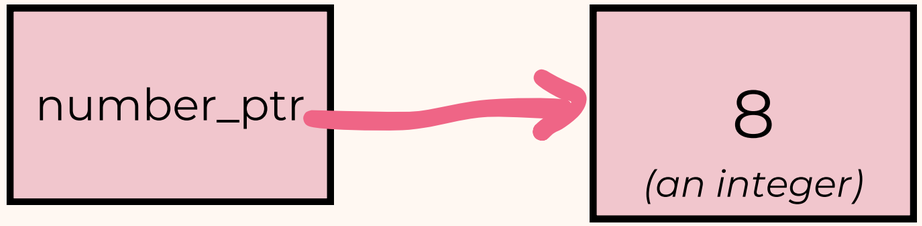


Pointers Recap

We can have different types of pointers...

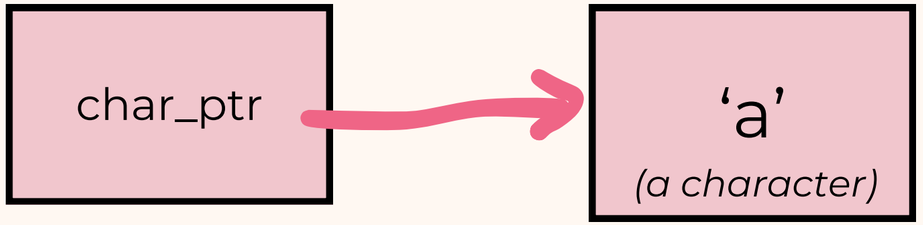
INTEGER POINTER

```
int *number_ptr;
```



CHARACTER POINTER

```
char *char_ptr;
```

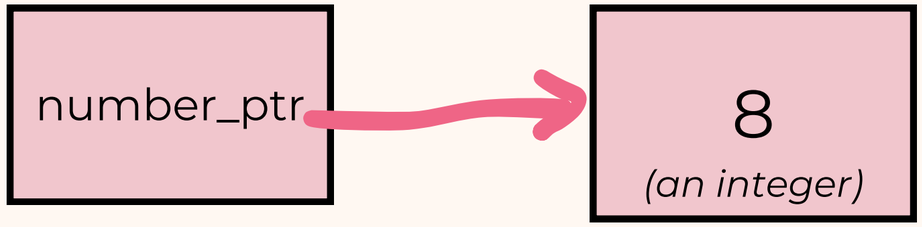


Pointers Recap

We can have different types of pointers...

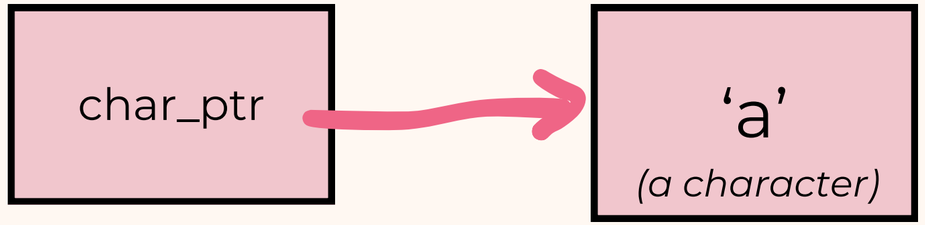
INTEGER POINTER

```
int *number_ptr;
```



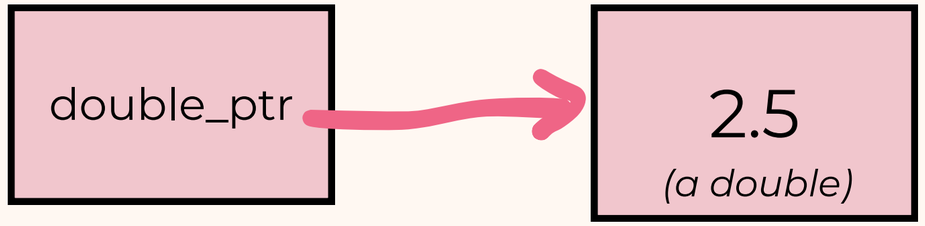
CHARACTER POINTER

```
char *char_ptr;
```



DOUBLE POINTER

```
double *double_ptr;
```



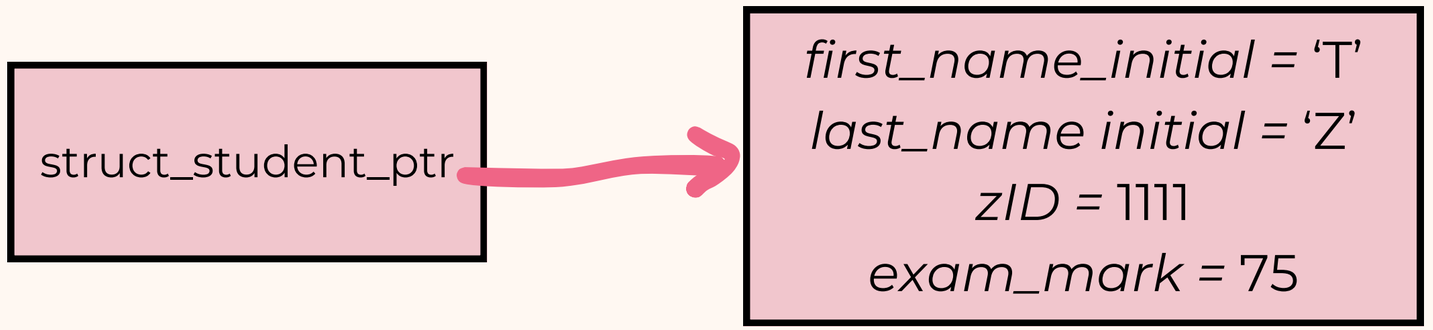
Pointers Recap

We can have different types of pointers...

STRUCT POINTER

```
struct student {
    char first_name_initial;
    char last_name_initial;
    int zID;
    double exam_mark;
};
```

```
struct student *struct_student_ptr;
```



(a "struct student")

Any questions about pointers?

Mini Quiz: Will the following work in code?

```
1 int number;
2 int *number_ptr;
3
4 number_ptr = number; // 1
5
6 *number_ptr = &number; // 2
7
8 number_ptr = &number; // 3
9
10 *number_ptr = number; // 4
```

Mini Quiz: Will the following work in code?

```
1 int number;
2 int *number_ptr;
3
4 number_ptr = number; // 1
5
6 *number_ptr = &number; // 2
7
8 number_ptr = &number; // 3
9
10 *number_ptr = number; // 4
```

NO - THEY ARE DIFFERENT TYPES

Mini Quiz: Will the following work in code?

```
1 int number;
2 int *number_ptr;
3
4 number_ptr = number; // 1
5
6 *number_ptr = &number; // 2
7
8 number_ptr = &number; // 3
9
10 *number_ptr = number; // 4
```

NO - THEY ARE DIFFERENT TYPES

NO - LHS IS AN INT, RHS IS A POINTER (ADDRESS)

Mini Quiz: Will the following work in code?

```
1 int number;
2 int *number_ptr;
3
4 number_ptr = number; // 1
5
6 *number_ptr = &number; // 2
7
8 number_ptr = &number; // 3
9
10 *number_ptr = number; // 4
```

NO - THEY ARE DIFFERENT TYPES

NO - LHS IS AN INT, RHS IS A POINTER (ADDRESS)

YES!

Mini Quiz: Will the following work in code?

```
1 int number;
2 int *number_ptr;
3
4 number_ptr = number; // 1
5
6 *number_ptr = &number; // 2
7
8 number_ptr = &number; // 3
9
10 *number_ptr = number; // 4
```

NO - THEY ARE DIFFERENT TYPES

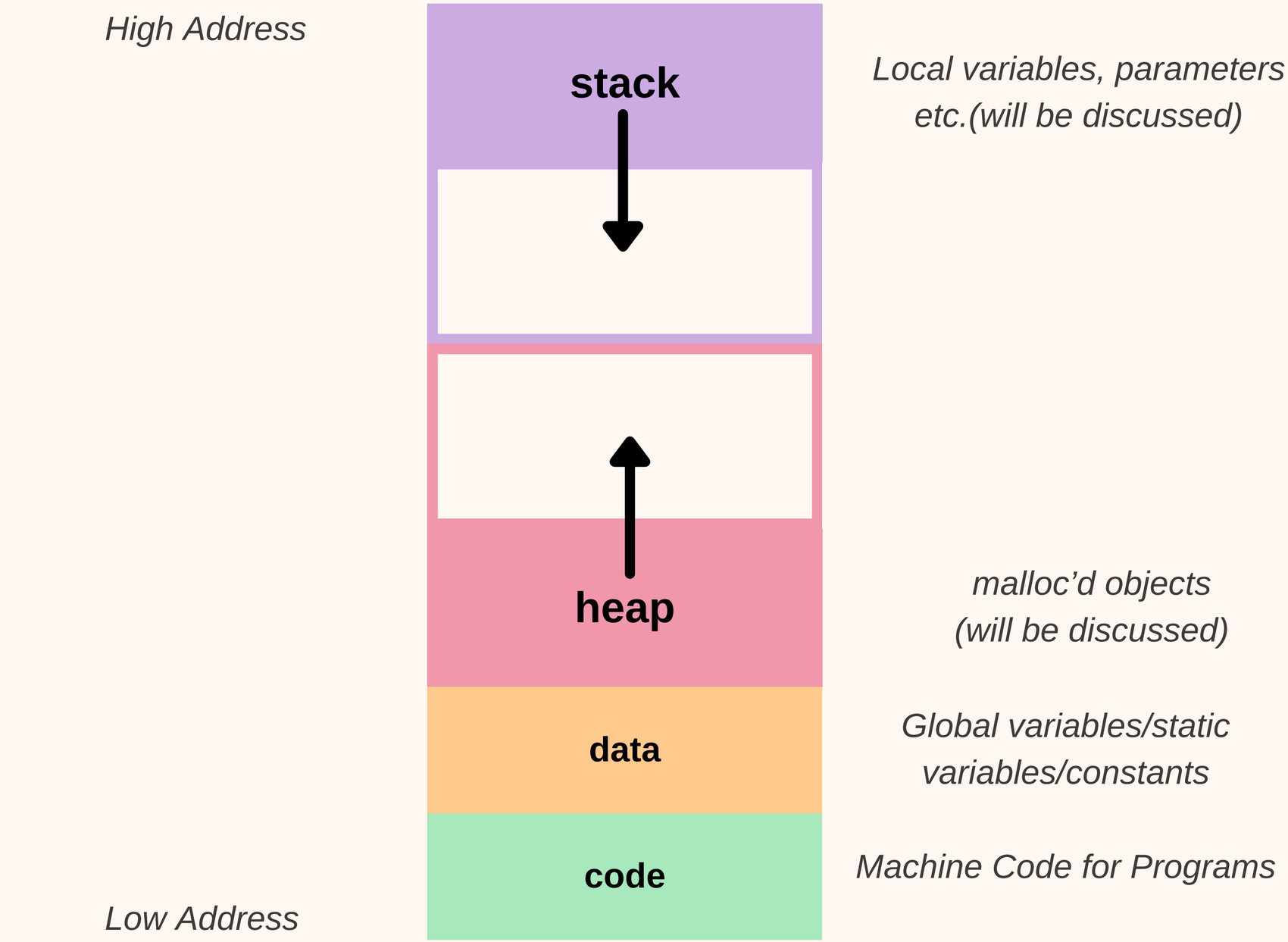
NO - LHS IS AN INT, RHS IS A POINTER (ADDRESS)

YES!

DEPENDS - IS NUMBER_PTR INITIALISED?

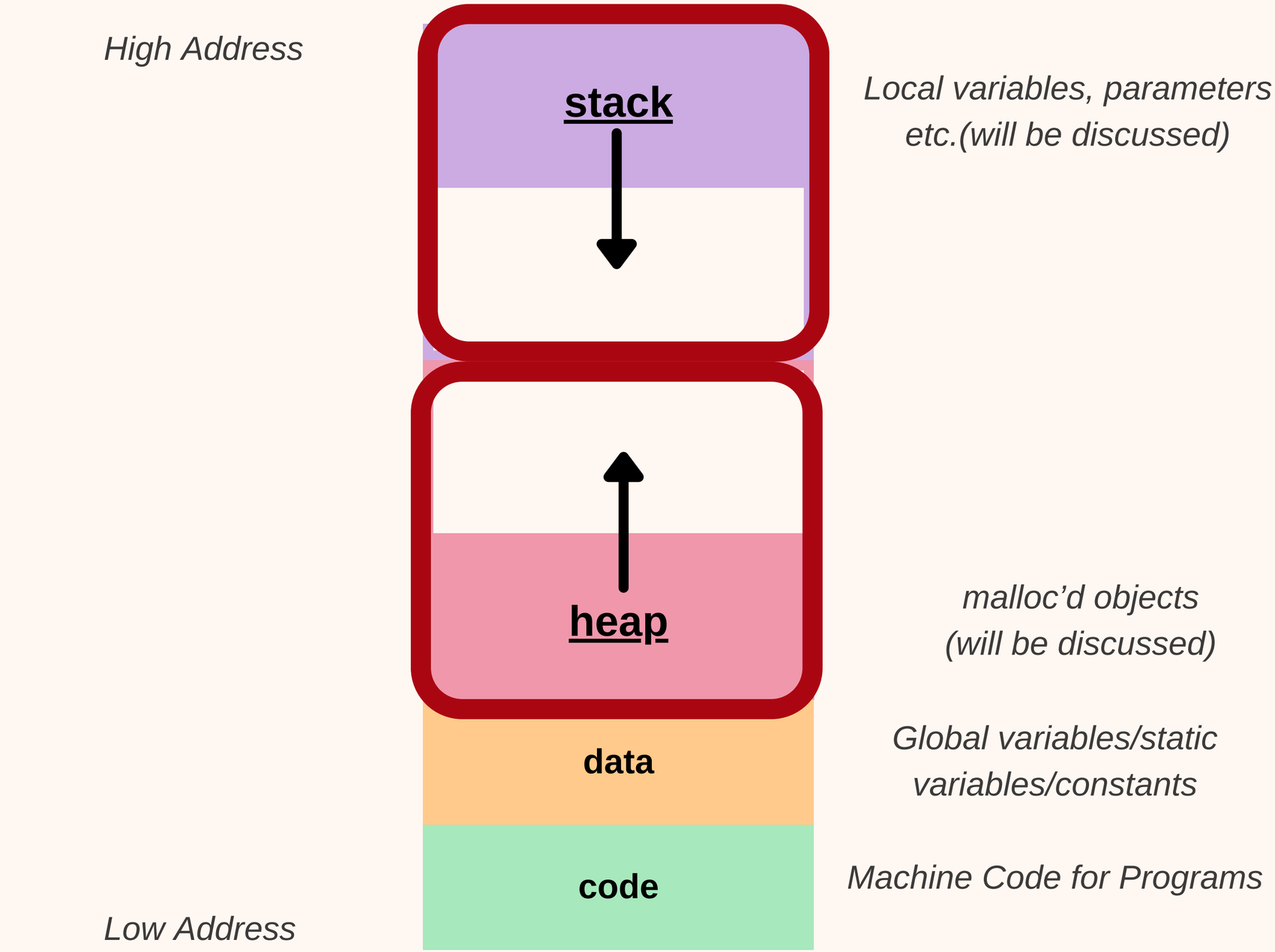
Memory Recap

Our “block of memory” looks like:



Memory Recap

Our “block of memory” looks like:



Stack

- Where information about your program goes:
  - which functions are called + in what order,
  - what variables you created + where
- When a block of code is executed { }, a stack frame is created on the stack (roughly enough memory to store everything in the frame is allocated to the frame)
- When a block of code is completed, the stack frame is removed from the stack
  - anything inside stack frame is destroyed

block of code example: functions

Stack

- Where information about your program goes:
  - which functions are called + in what order,
  - what variables you created + where
- When a block of code is executed { }, a stack frame is created on the stack (roughly enough memory to store everything in the frame is allocated to the frame)
- When a block of code is completed, the stack frame is removed from the stack
  - anything inside stack frame is destroyed

VS.

block of code example: functions

Stack

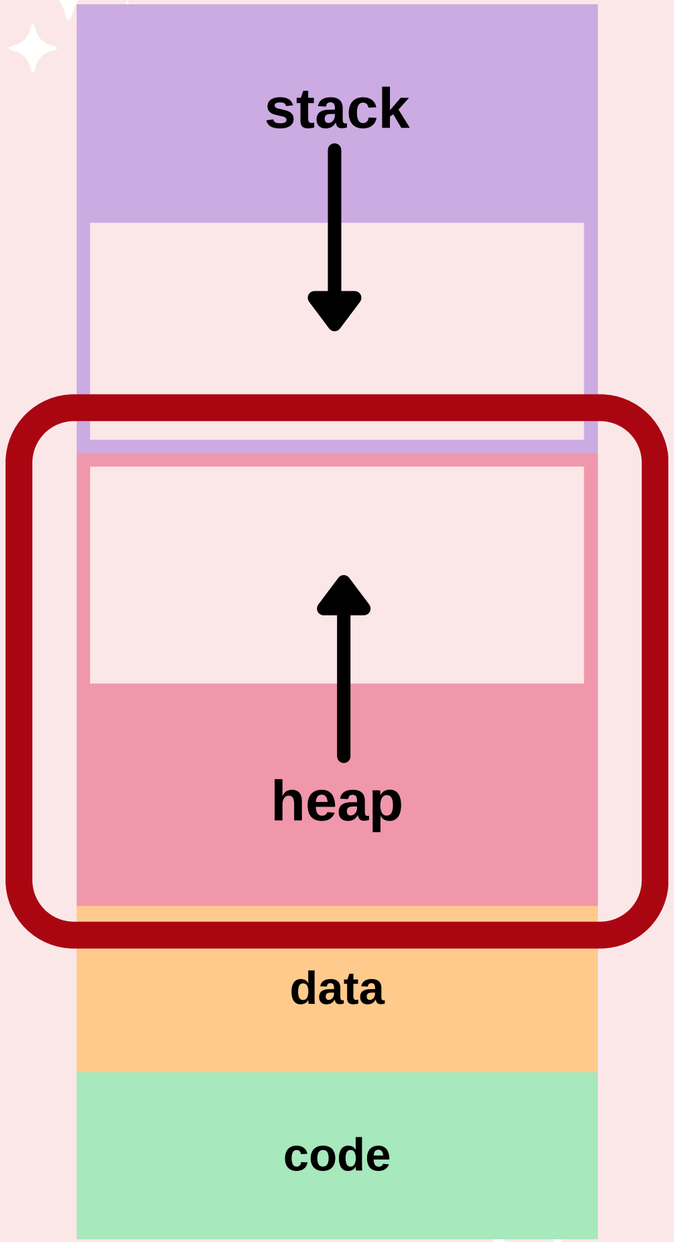
- Where information about your program goes:
  - which functions are called + in what order,
  - what variables you created + where
- When a block of code is executed { }, a stack frame is created on the stack (roughly enough memory to store everything in the frame is allocated to the frame)
- When a block of code is completed, the stack frame is removed from the stack
  - anything inside stack frame is destroyed

block of code example: functions

VS.

Heap

- Memory allocated by the programmer to store data resides here
  - won't be deallocated until it is explicitly freed by the programmer
- Nothing is automatically declared or destroyed in the Heap
- can dynamically ask for/return memory as we need
  - using malloc(...) and free(...)



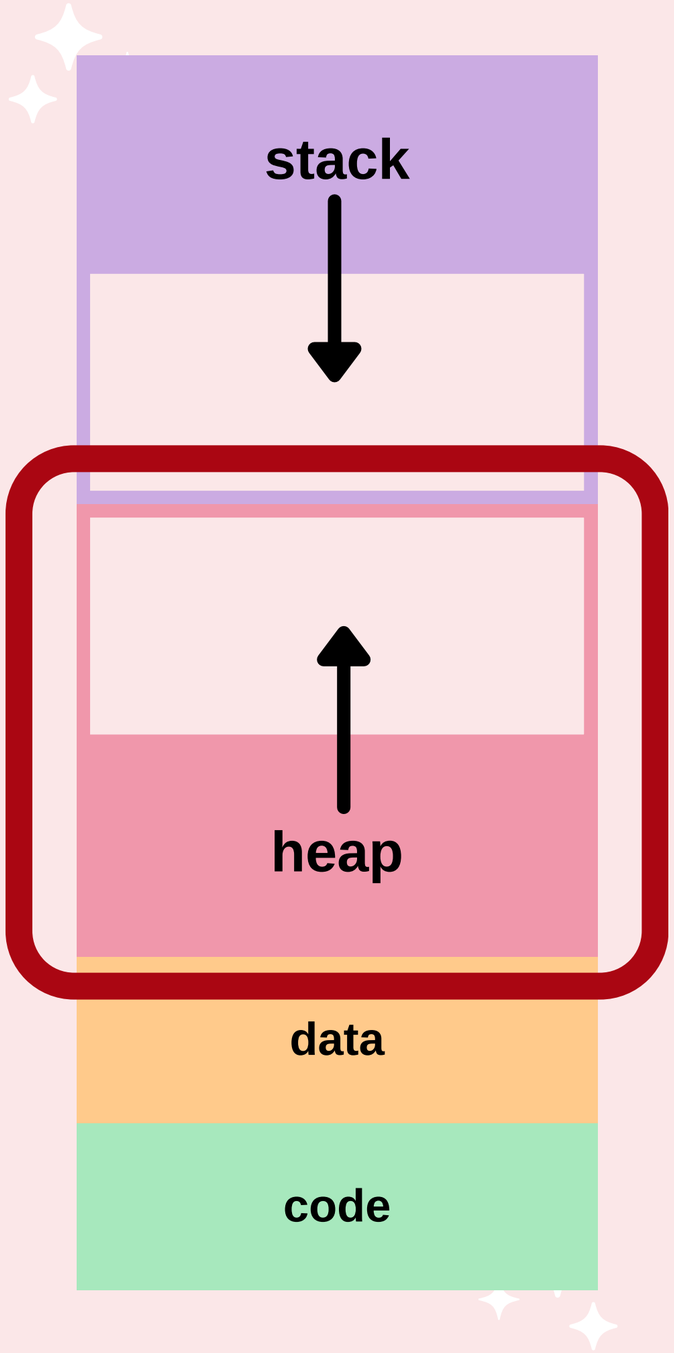
The Heap - malloc()

How we can use the heap:

- malloc() - “Memory Allocation”
  - take a value representing the size we want in bytes
  - allocates memory on the heap
  - returns a pointer to the location on the heap
- allows us to dynamically create memory as we need it now!

```
malloc(number_of_bytes_we_want)
```

```
malloc(1000);
```



The Heap - free()

- We can't keep asking for more memory without giving some back!
- GIVE BACK ANY UNUSED ALLOCATED SPACE BY USING FREE()

Otherwise, we can have:

Memory Leaks ✗

- occurs when you have dynamically allocated memory (with malloc), do not free it and, memory is lost and cannot be freed

```
free(ptr);
```

Memory Recap

sizeof()

- sizeof()
  - takes in a data type
  - tells us the exact number of bytes we need to malloc
- Let me show you the magic of sizeof() with a mini code demo

```
sizeof(data_type);
```

```
sizeof(int);
```

Memory Recap

Putting it altogether

malloc(sizeof(...)
free()

```
int *ptr = malloc(10 * sizeof(int));
```

(generally sizeof(int) is 4)

This will give us 10 x 4 bytes (i.e. 40bytes) which is pointed to by ptr

Putting it altogether

malloc(sizeof(...)
free()

Memory Recap

```
1 int *ptr = malloc(x * sizeof(int));
2
3 ...
4
5 free(ptr);
```

‘x’ is useful when you want more than one element!

Memory Recap

Putting it altogether

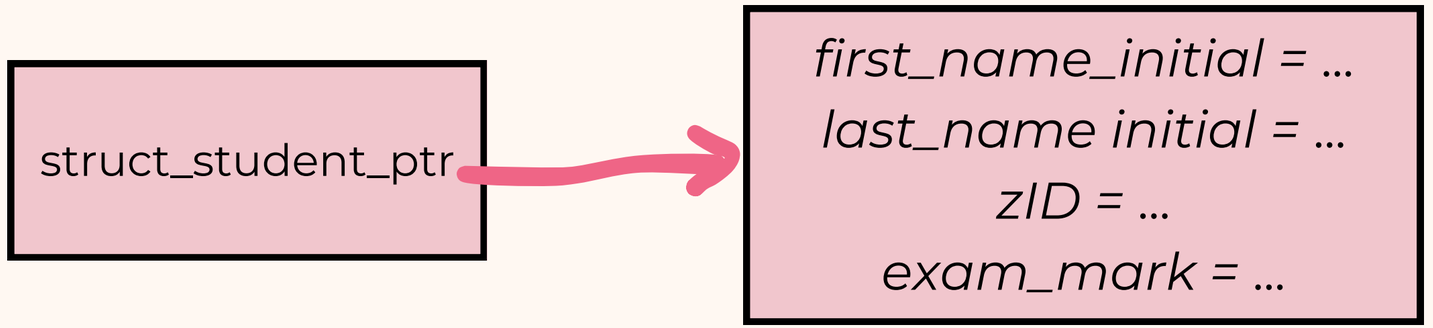
(with struct pointers)

malloc(sizeof(...)
free()

```c
struct student *struct_student_ptr = malloc(sizeof(struct student));

...

free(struct_student_ptr);
```



(a malloc-ed “struct student”)

Any questions?

End of Recap!

So far, we store a collection of data/values using:

ARRAYS

We have a problem...

WHEN WORKING WITH ARRAYS...

I don't know how much space I need to store this list of things...

STATIC ARRAYS

We can either assume a huge size and waste space, or not do that and run out of space...

DYNAMIC ARRAY

I could use a dynamic array and realloc to increase the memory used when needed, but that can be costly
(e.g. when you already have a large memory block)

WHAT IF...

....there is a way to store a collection of data and request for more memory on demand easily for additional elements?

LINKED LISTS!

BREAK TIME!
(KAHOOT)

What are  ?

Similar to arrays:

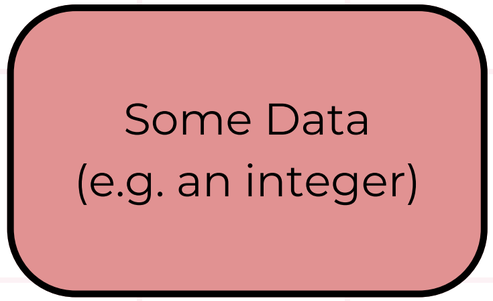
- another way to store a collection of the same data type

Different to arrays:

- dynamically sized (very efficient!) - ask for and give back memory as necessary
- elements - no need to be stored contiguously in memory
- can only access items starting from the beginning of the list

What are LINKED LISTS! ?

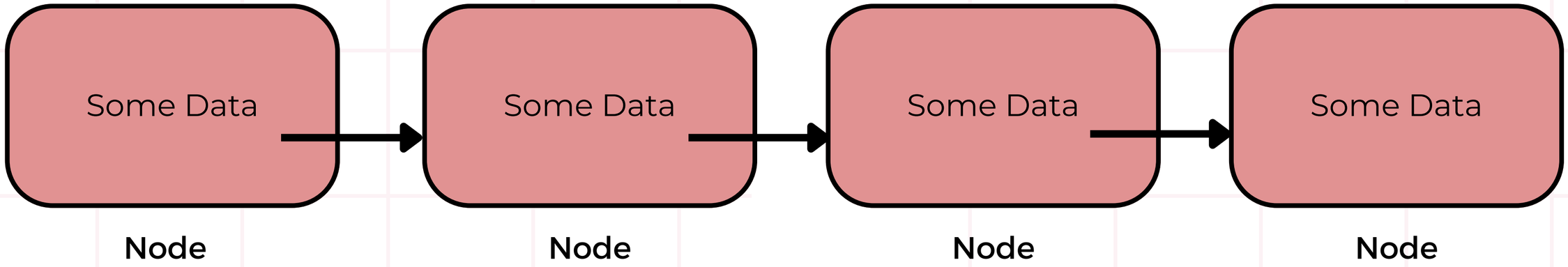
Visually (very vaguely), it looks like this:



Node
(i.e. an element of a linked list)

What are LINKED LISTS! ?

Visually (very vaguely), it looks like this:



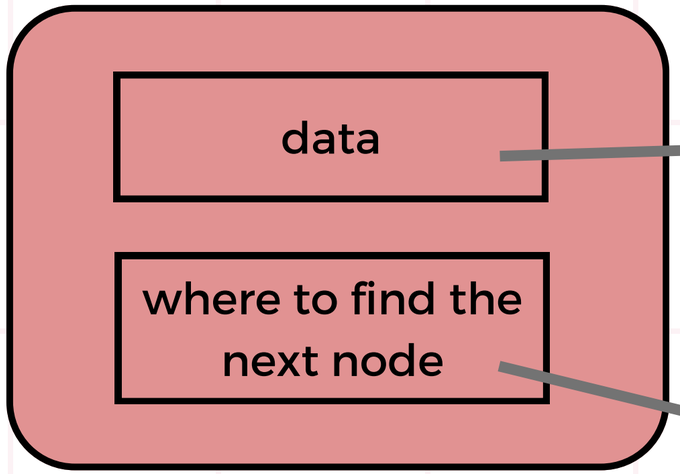
How does a Linked List link up like that??

How does a Linked List link up like that??

Let's do a high-level walkthrough:
create a linked list to store the numbers ***11, 8 , 7***
as elements

How does a Linked List link up like that??

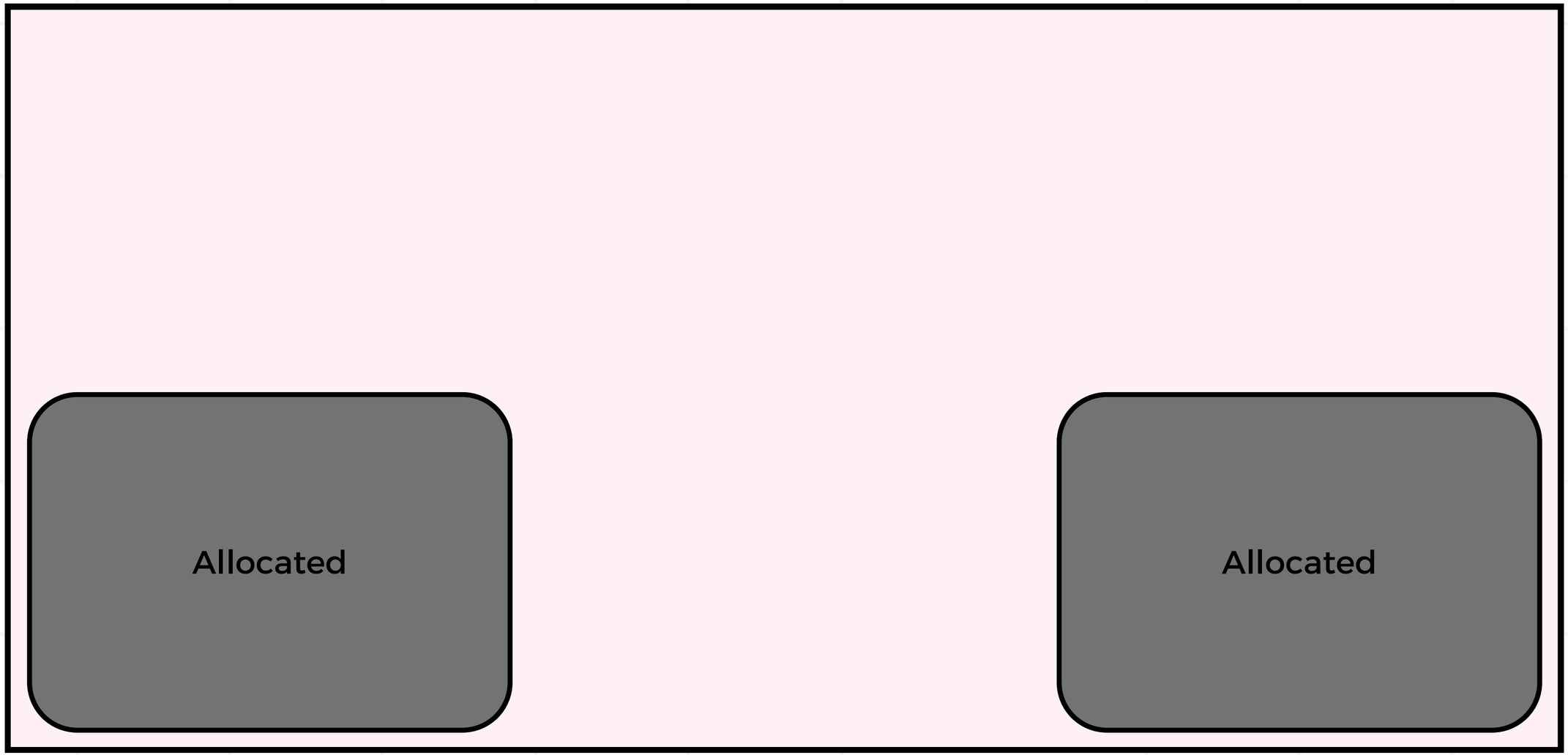
Firstly, we need each of our nodes to look similar to this and store the following data:



An integer variable - to store the number
e.g. 7

A variable storing the address of the next node (i.e. a pointer) - so there's a way to connect nodes together
e.g. 0x66

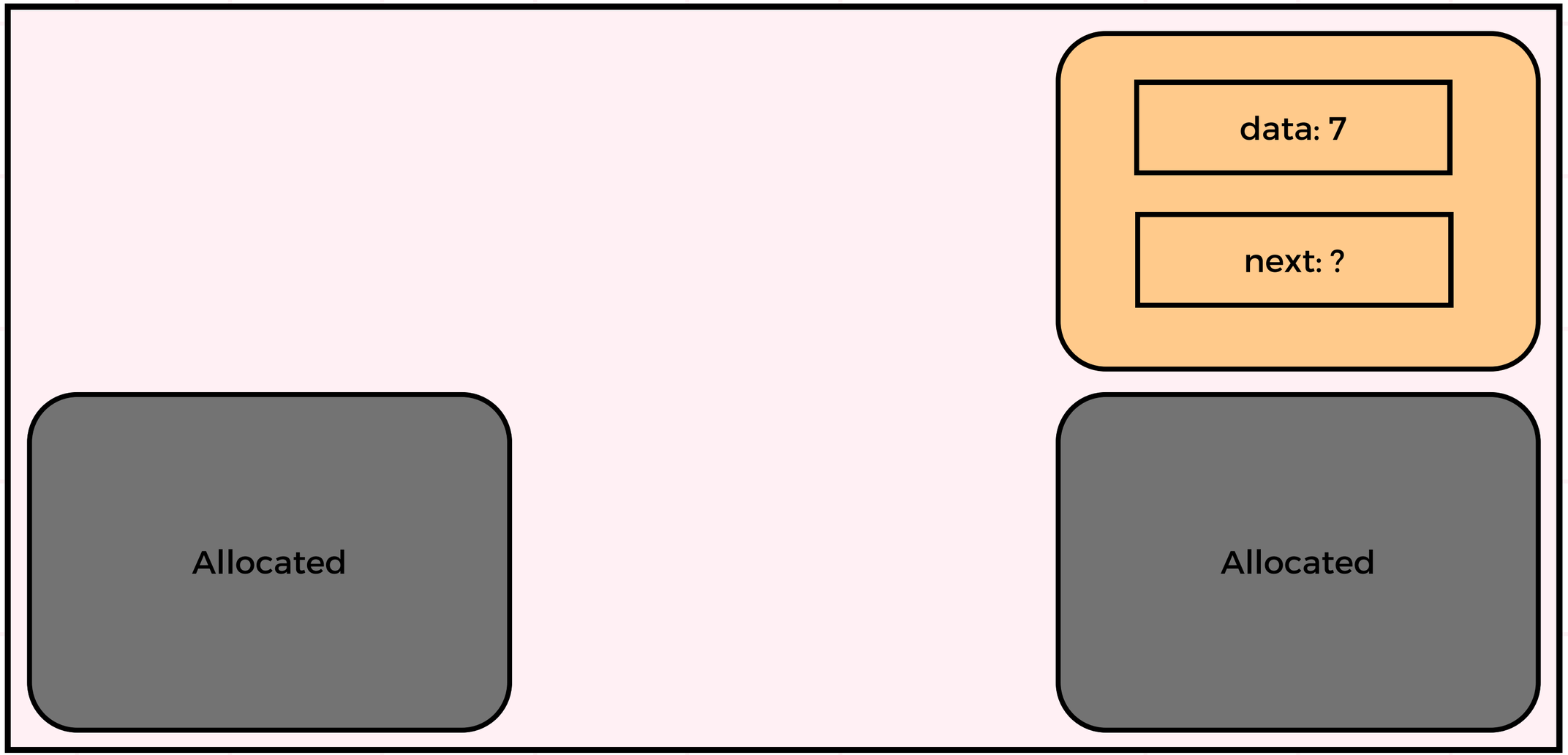
In the Computer Memory...



Walkthrough Steps:

Note: This is not what it actually looks like in memory , diagram simplified for understanding

In the Computer Memory...

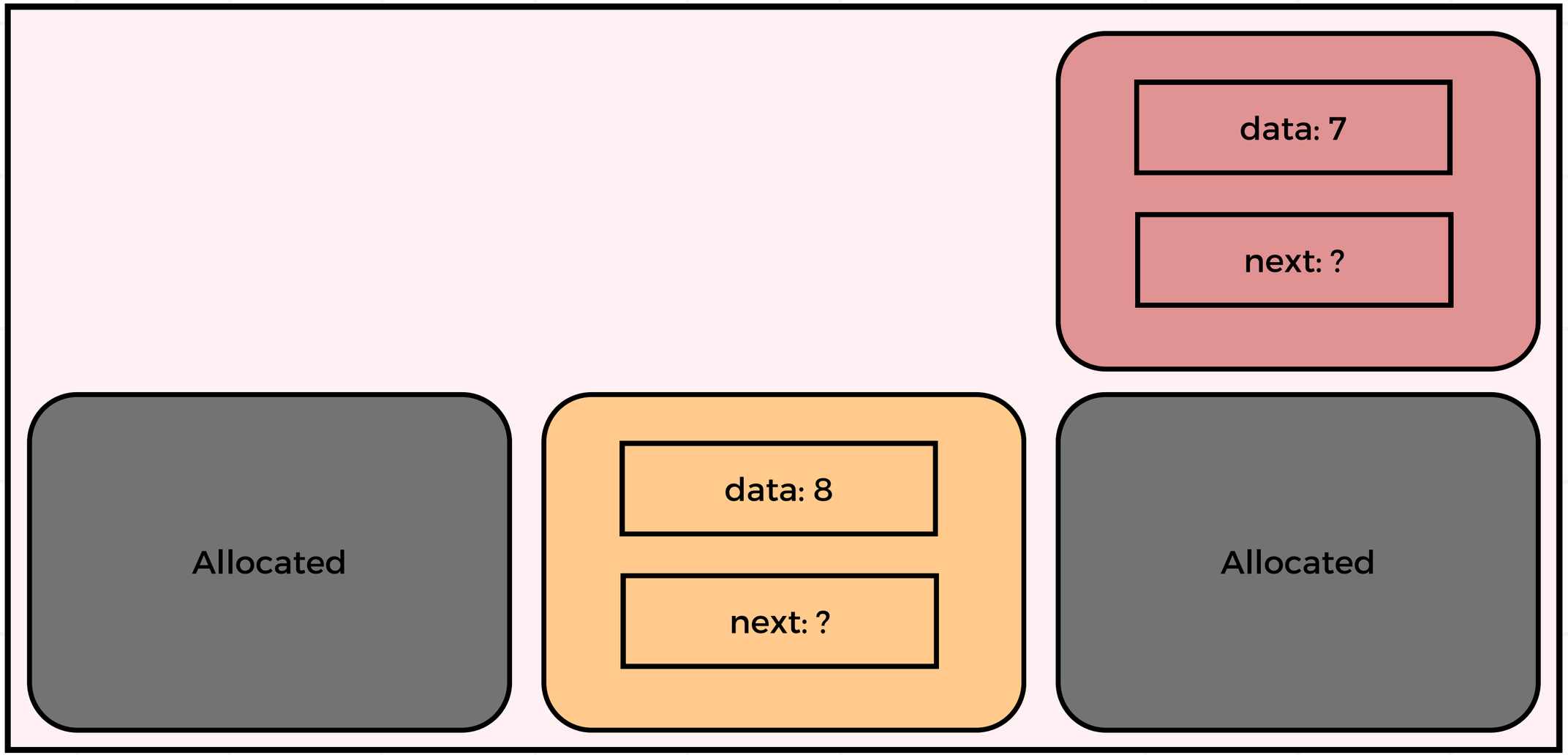


Walkthrough Steps:

1. Malloc and store 7

Note: This is not what it actually looks like in memory , diagram simplified for understanding

In the Computer Memory...

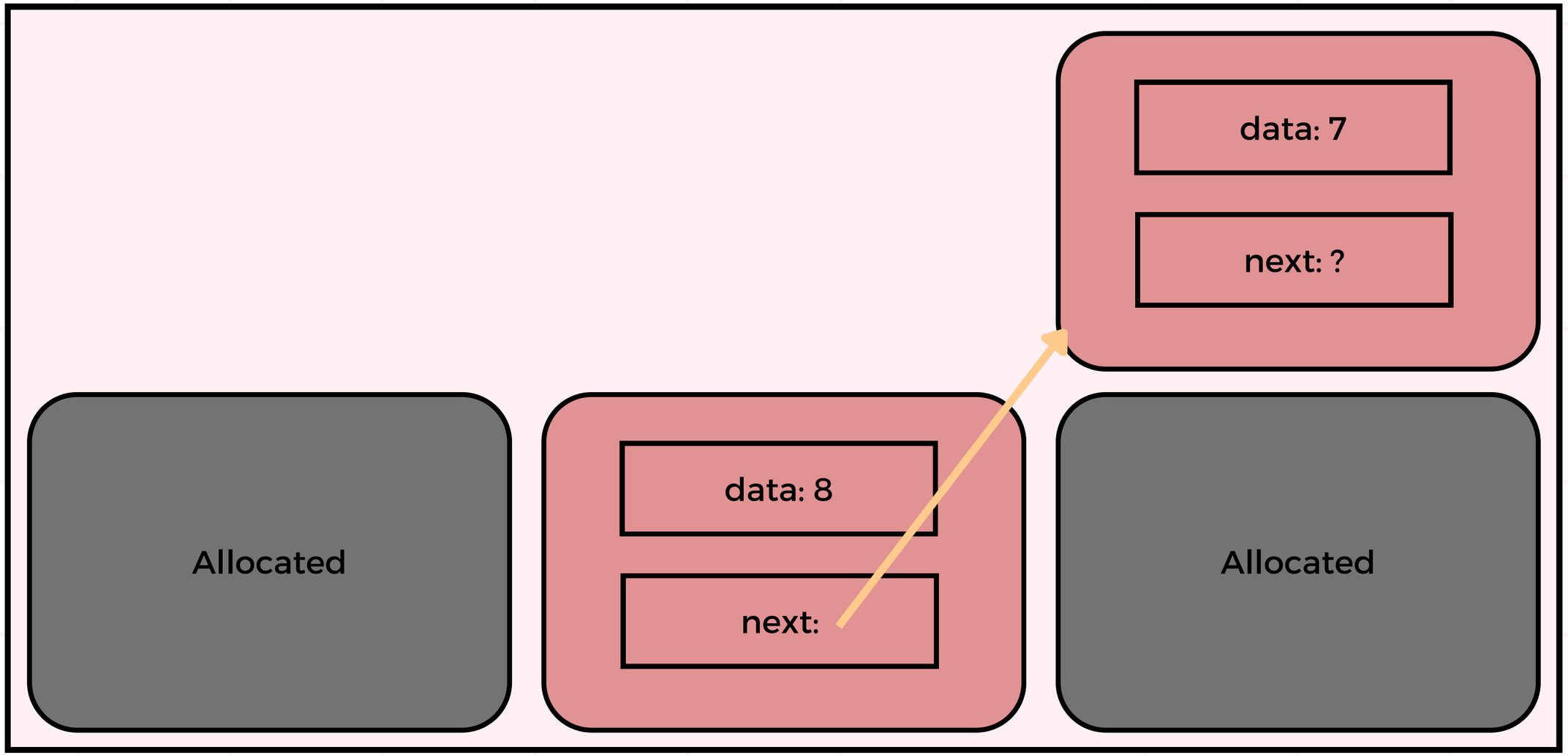


Walkthrough Steps:

1. Malloc and store 7
2. Malloc and store 8

Note: This is not what it actually looks like in memory , diagram simplified for understanding

In the Computer Memory...

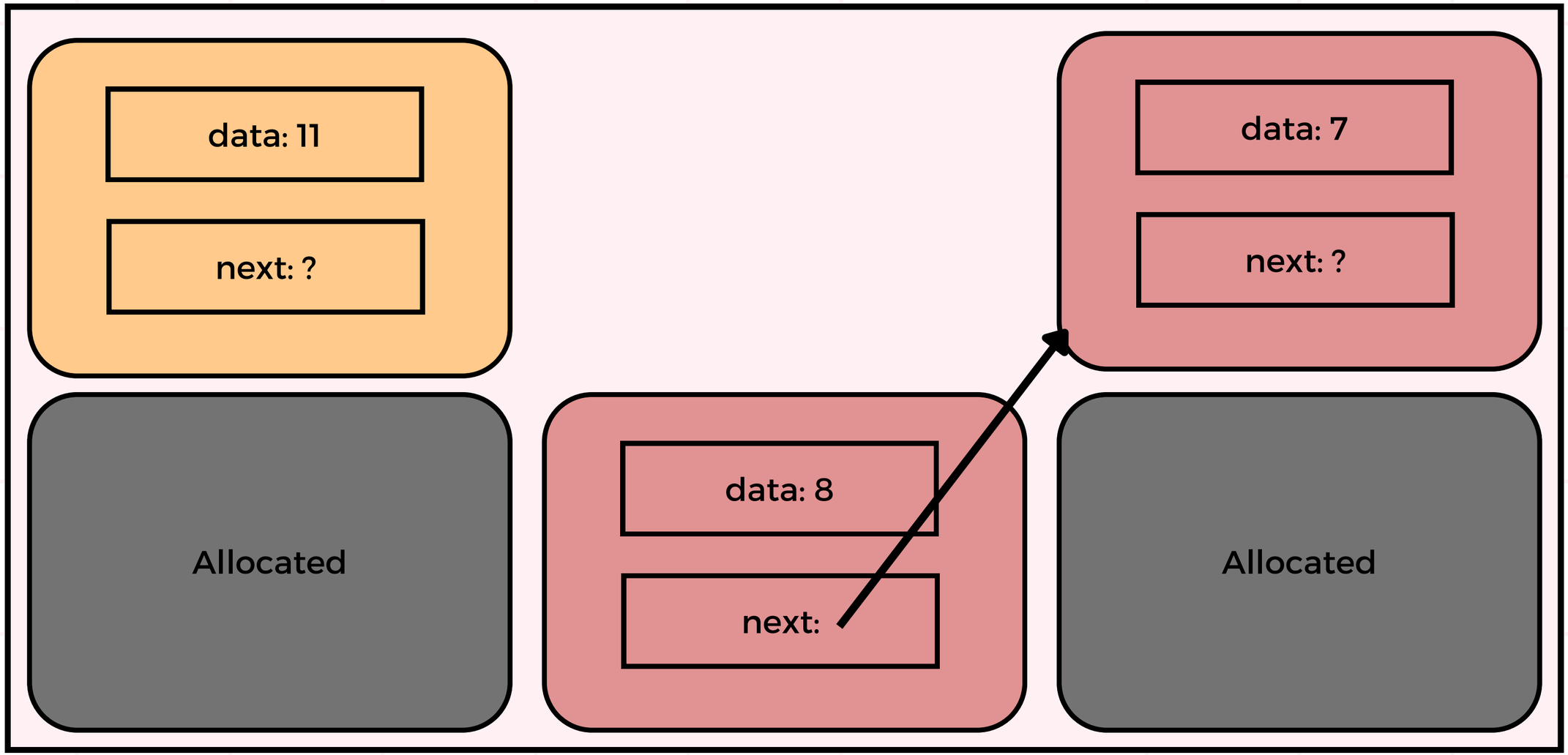


Walkthrough Steps:

1. Malloc and store 7
2. Malloc and store 8
3. Connect 8 to 7

Note: This is not what it actually looks like in memory , diagram simplified for understanding

In the Computer Memory...

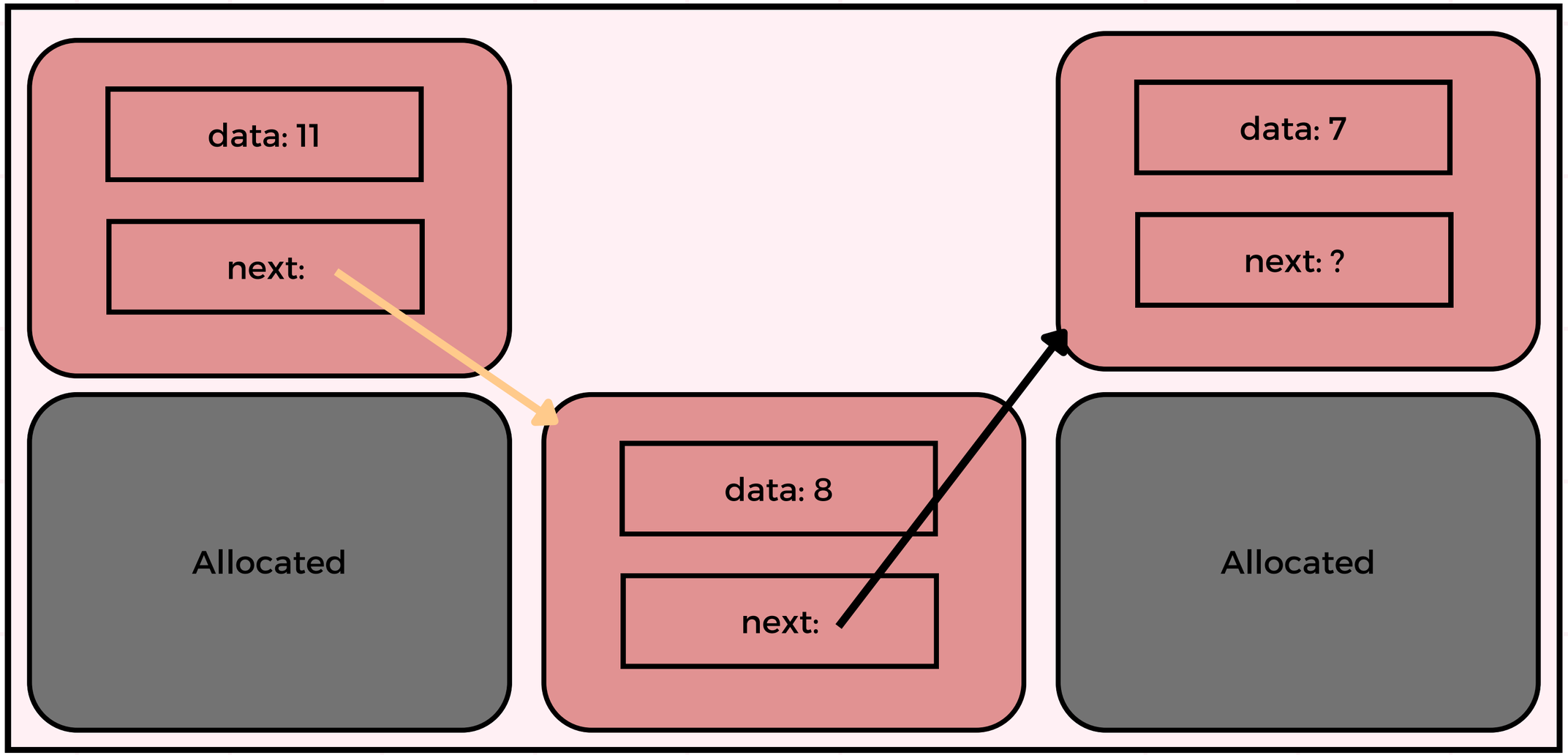


Walkthrough Steps:

1. Malloc and store 7
2. Malloc and store 8
3. Connect 8 to 7
4. Malloc and store 11

Note: This is not what it actually looks like in memory , diagram simplified for understanding

In the Computer Memory...

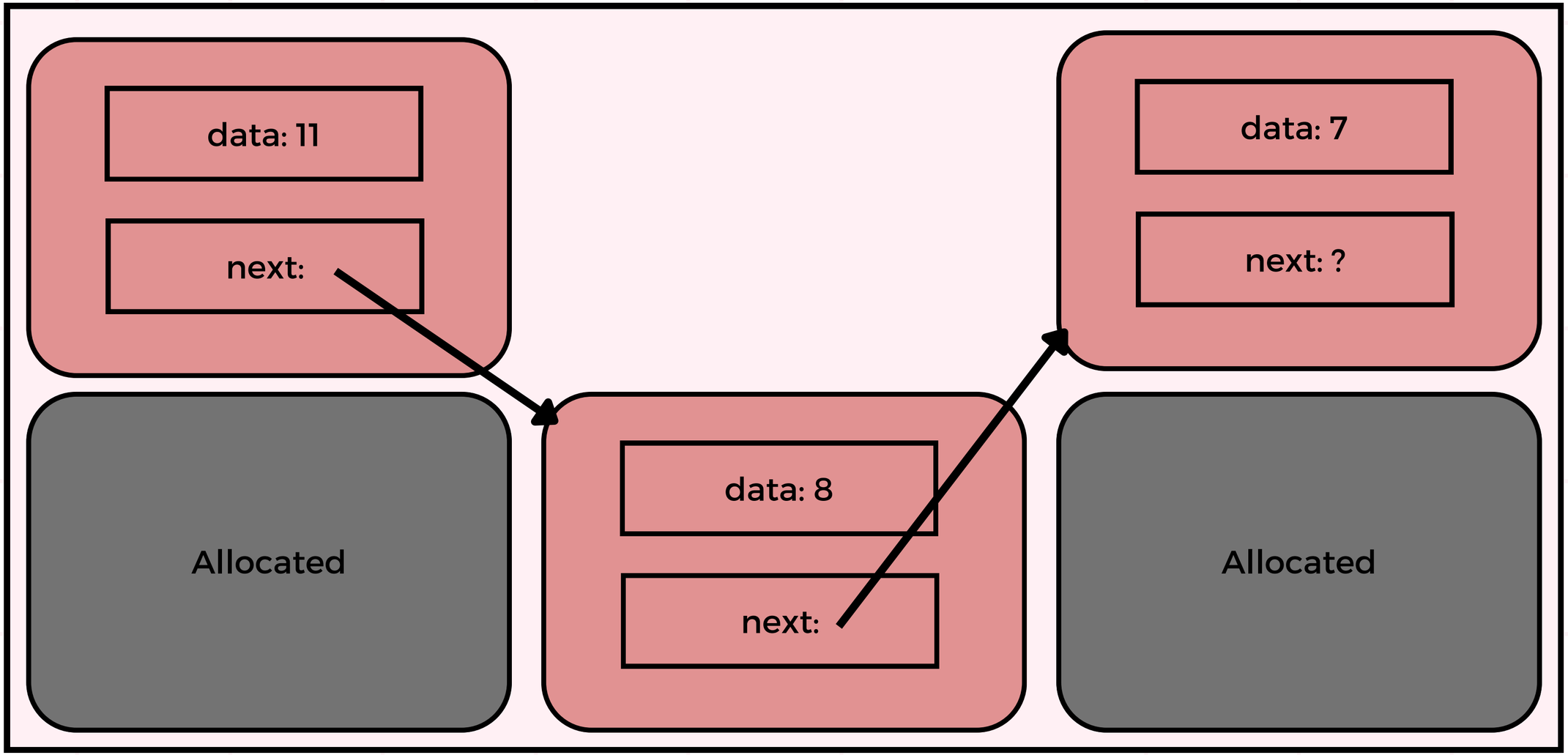


Walkthrough Steps:

1. Malloc and store 7
2. Malloc and store 8
3. Connect 8 to 7
4. Malloc and store 11
5. Connect 11 to 8

Note: This is not what it actually looks like in memory , diagram simplified for understanding

We now have this...



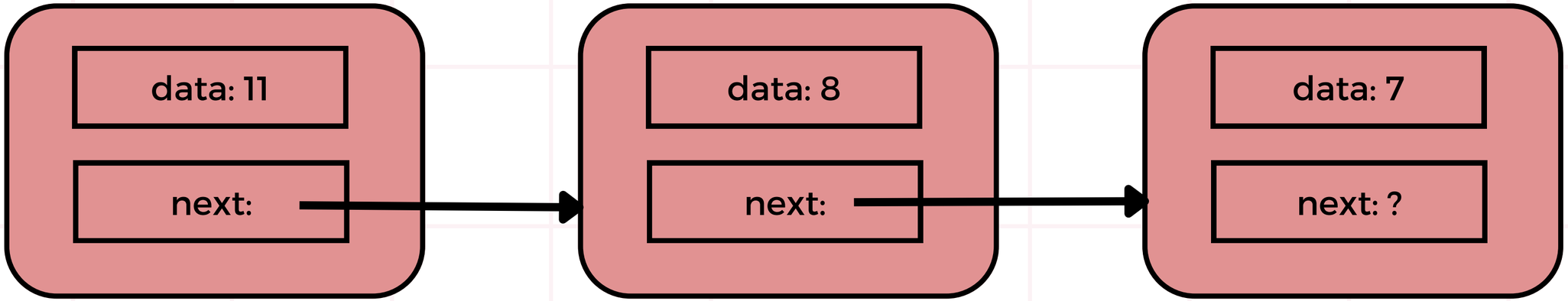
Walkthrough Steps:

1. Malloc and store 7
2. Malloc and store 8
3. Connect 8 to 7
4. Malloc and store 11
5. Connect 11 to 8

Note: This is not what it actually looks like in memory , diagram simplified for understanding

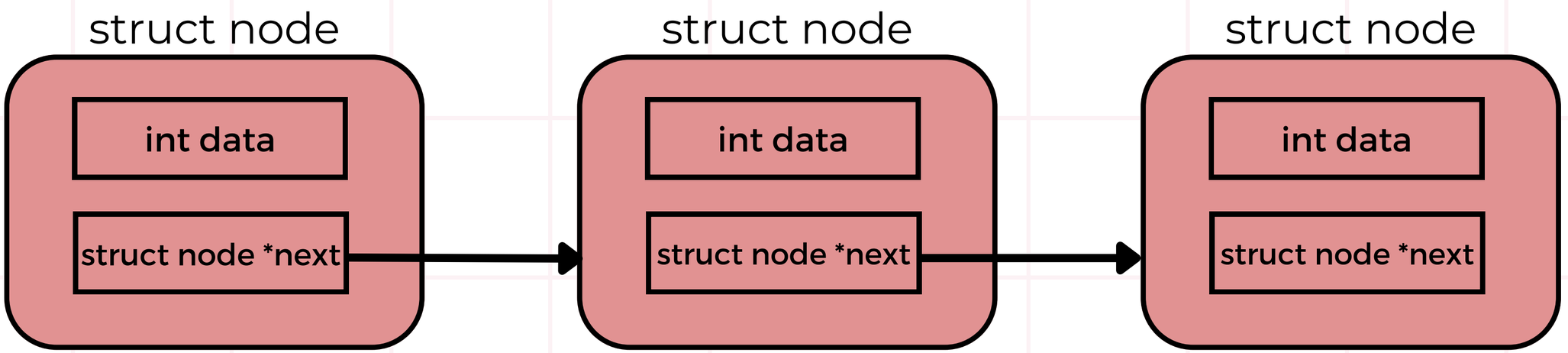
LINKED LISTS!

So really, we end up with something like this:




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

If we generalise it:

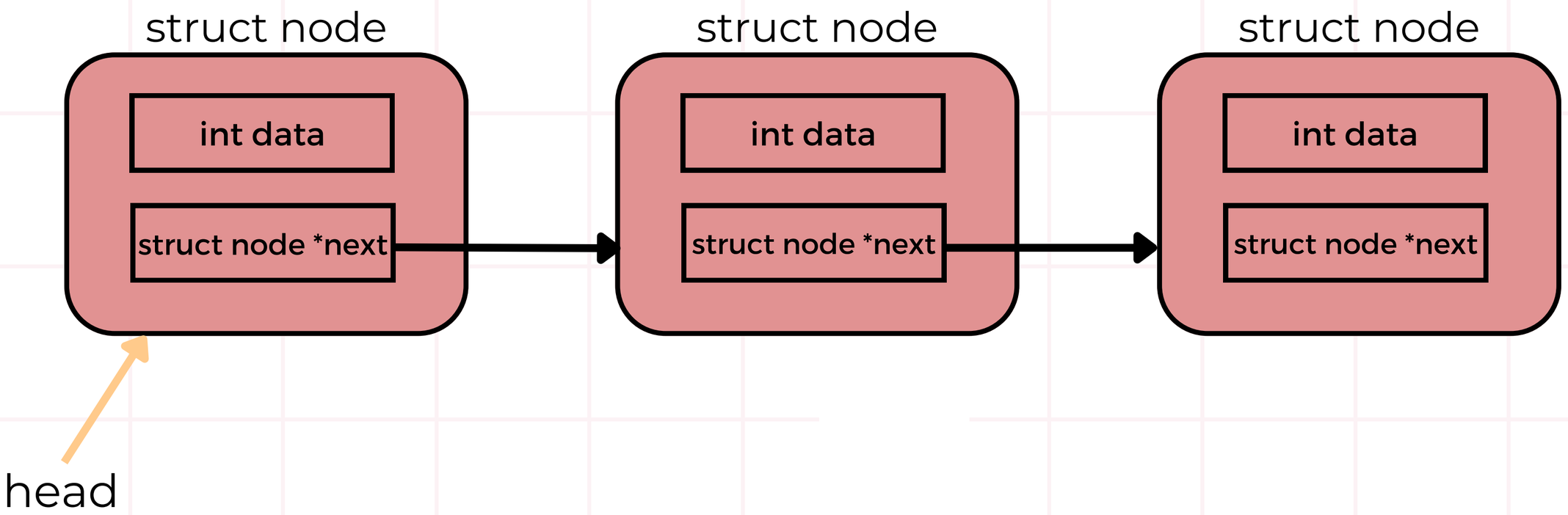


... a bunch of structs containing pointers that point to other structs!!

Keep this diagram in your head! You will need it!

LINKED LISTS!

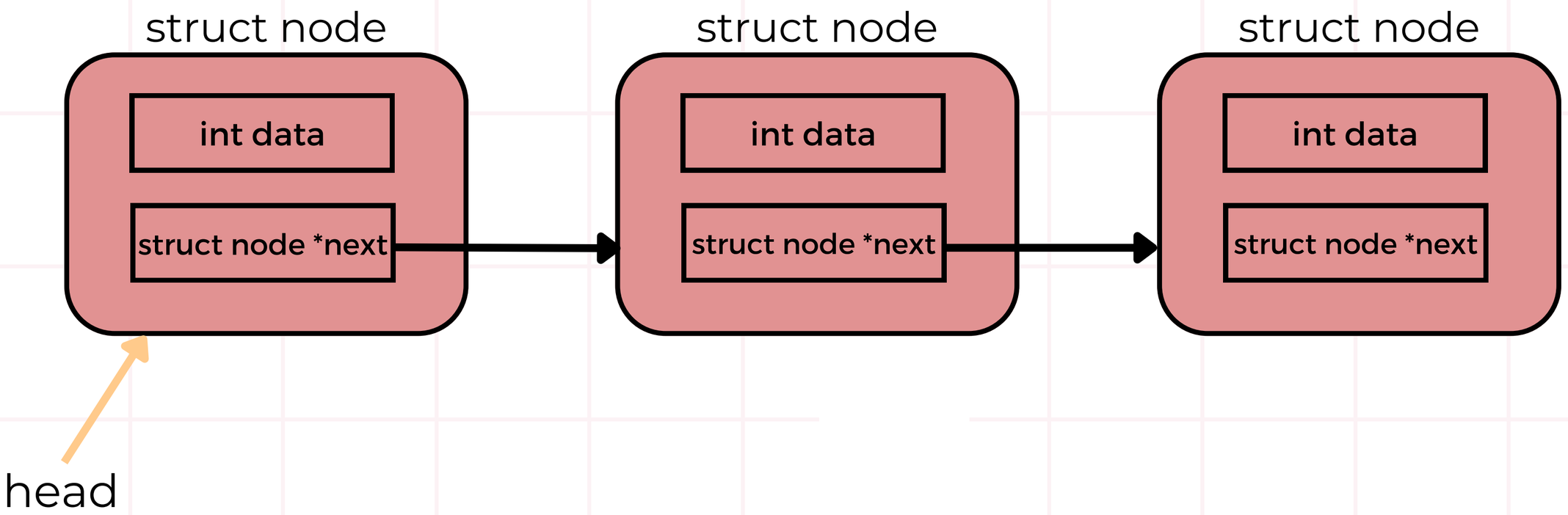
One more thing: a head pointer



Keep this diagram in your head! You will need it!

LINKED LISTS!

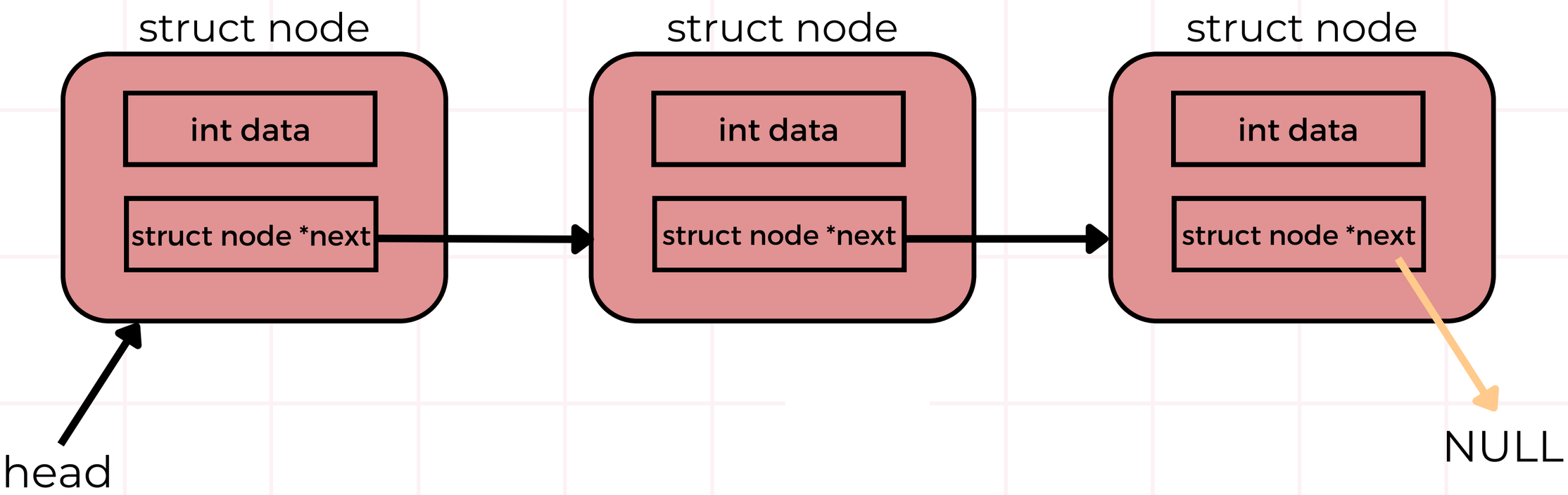
One more thing: a head pointer



Keep this diagram in your head! You will need it!

LINKED LISTS!

What about the next of the last node?: NULL



Keep this diagram in your head! You will need it!

LINKED LISTS!

An Empty Linked List looks like:

With linked list, we can:

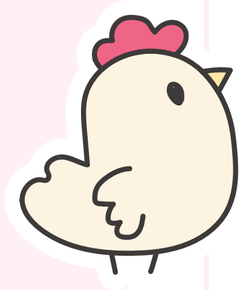
- add or remove nodes anywhere easily!
- change order easily!
- but can only access items starting from the beginning of the list

Me: "Can you give me the 50th item in this list?"
The Linked List: "No, go to the beginning of the list and search for it yourself!"

LET'S PUT THIS LINKED LIST TOGETHER IN CODE (W/ ACTUAL DRAWINGS)!

11->8->7->X

We will need to know how to use struct pointers and malloc!



Steps to do this!

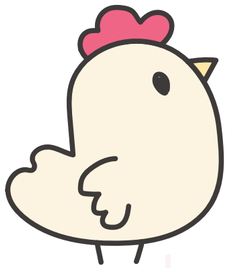
1. Define a struct for our node
2. Declare a pointer to keep track of the beginning of list
3. Code to create a node and connect it to a linked list

DIAGRAMS!

DIAGRAMS!

CODING TIME!

11->8->7->X



WHAT DID WE JUST CODE UP?

- *Created a linked list by inserting nodes at the head*
- *We are inserting backwards; last element inserted first*

HOW DO WE INSERT "FORWARD"?

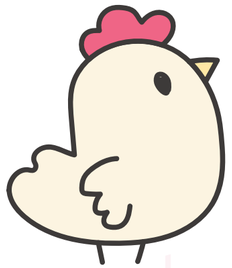
- *need to insert at tail*
- *need to know how to traverse the linked list to get to the end to do so*

DIAGRAMS!

DIAGRAMS!

CODING TIME! (AGAIN)

Traverse the linked list and print the data!

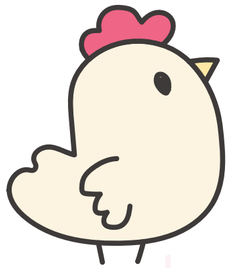


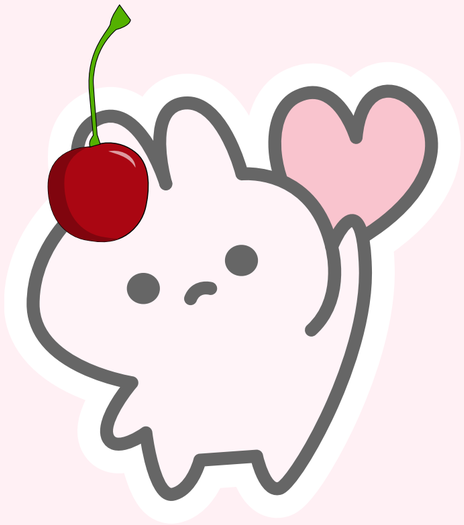
DIAGRAMS!

DIAGRAMS!

CODING TIME! (AGAIN)

Insert at Tail





FEEDBACK (PRETTY PLEASE WITH A CHERRY ON TOP)

https://forms.office.com/r/Cn8FgdFPhu

BUZZWORDS OF THE DAY

- NODE
- NULL
- HEAD
- TAIL

SUMMARY OF TODAY

- recap (some of) pointers & memory
- intro to linked lists
  - insert at head
  - traverse a linked list
  - insert at tail (maybe)

NEXT LECTURE

More linked list operations!

(actually... linked list for the next 3 lectures!)

If you have any questions:

COURSE RELATED

COURSE FORUM + HELP SESSIONS!

ADMIN RELATED

CS1511@UNSW.EDU.AU

thank you

And come say hi if you see me around on campus :D