Dynamic Memory Management

Dynamic Arrays

Dr Jake Renzella

So far, we have been requesting memory to store variables

This memory lives in the RAM of the computer

Let’s take a look at how this memory is organised

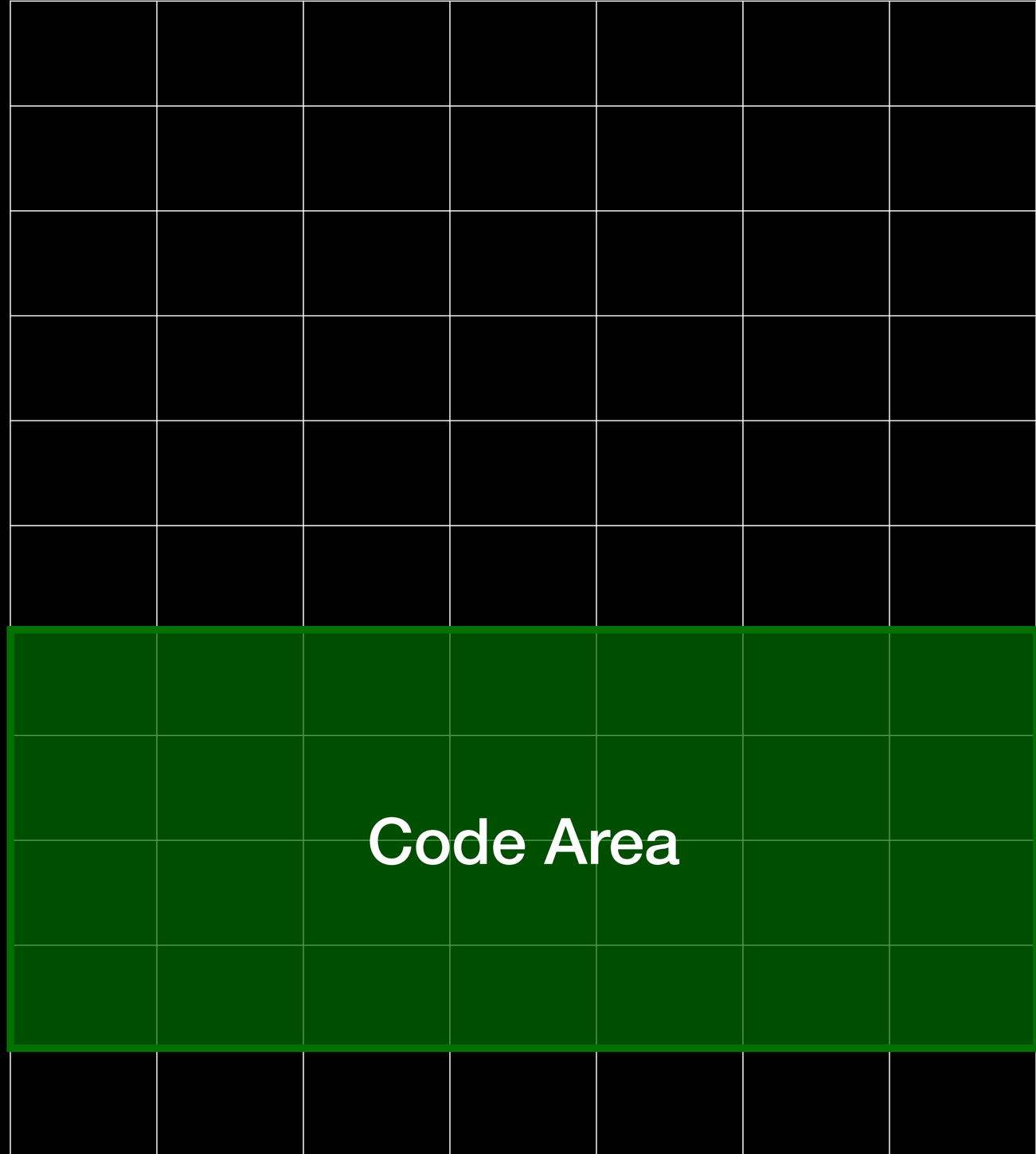
HEAP

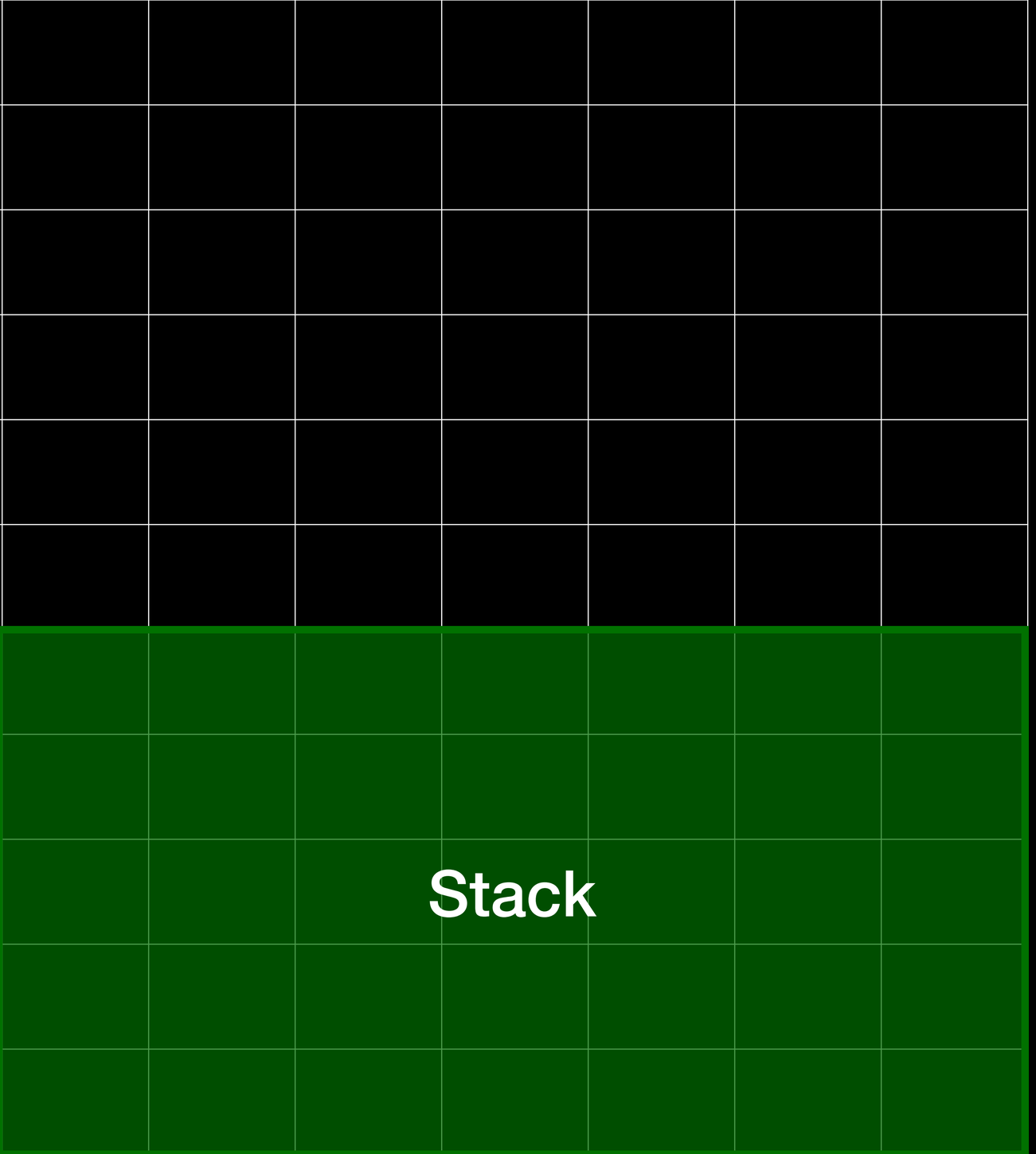
Stack

Code Area

Other

Stores the machine code instructions in memory





Stores the stack frames from function and procedure calls

HEAP

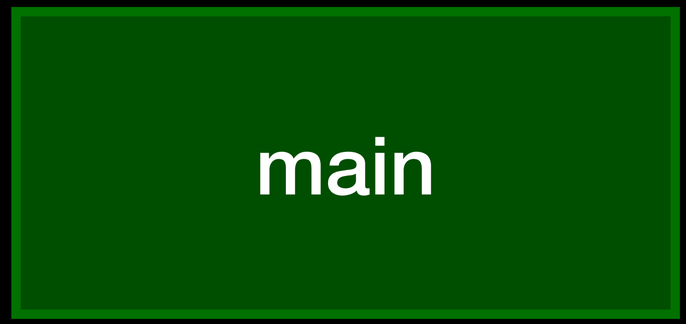
Stores dynamically
allocated memory

The Stack

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```

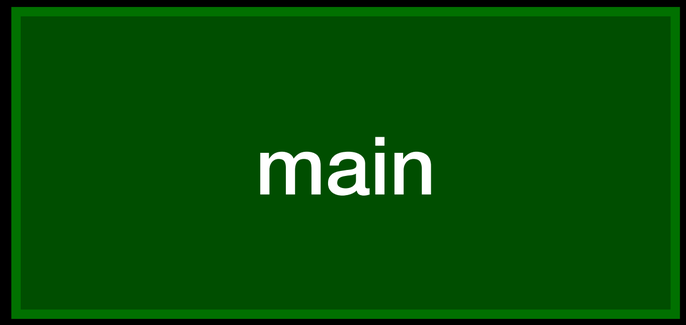
The Stack

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```



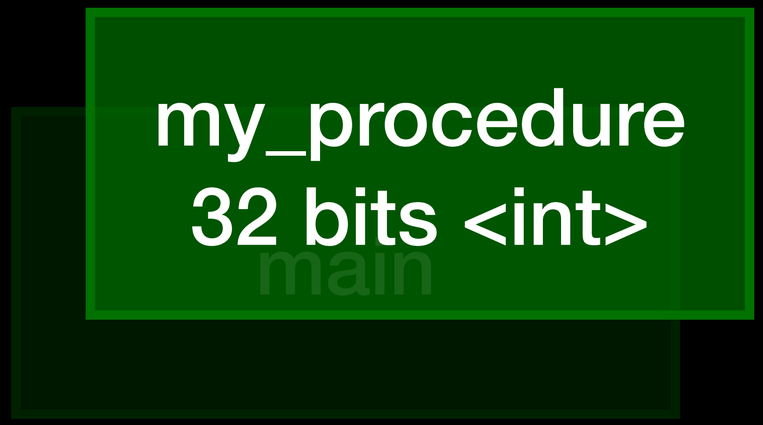
The Stack

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```



The Stack

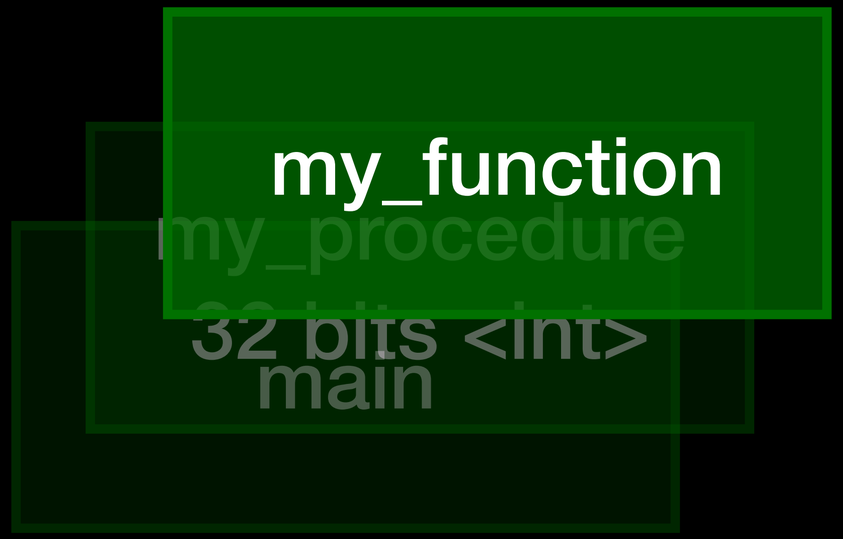
```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```



The Stack

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```



```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```

my_procedure
32 bits <int>

main

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```

main

```
int my_function(int y) {
    ...
}

void my_procedure() {
    int my_value;
    my_value = my_function();
}

int main() {
    my_procedure();
}
```

We might have a problem

If variables are stored on the stack, how does the program know how to fit a large Variable-Length Array?

int my_array[10000]

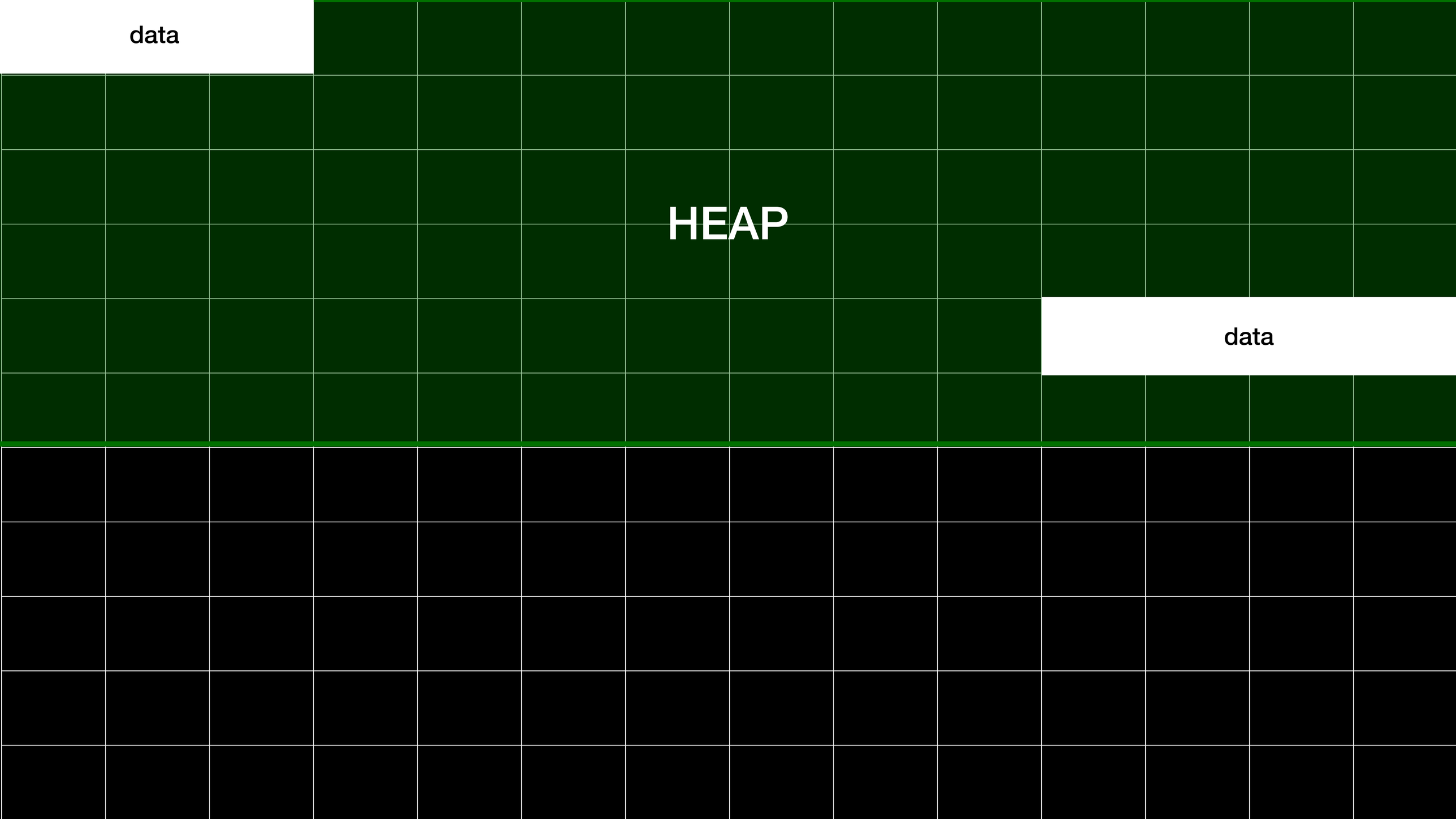
It might not.

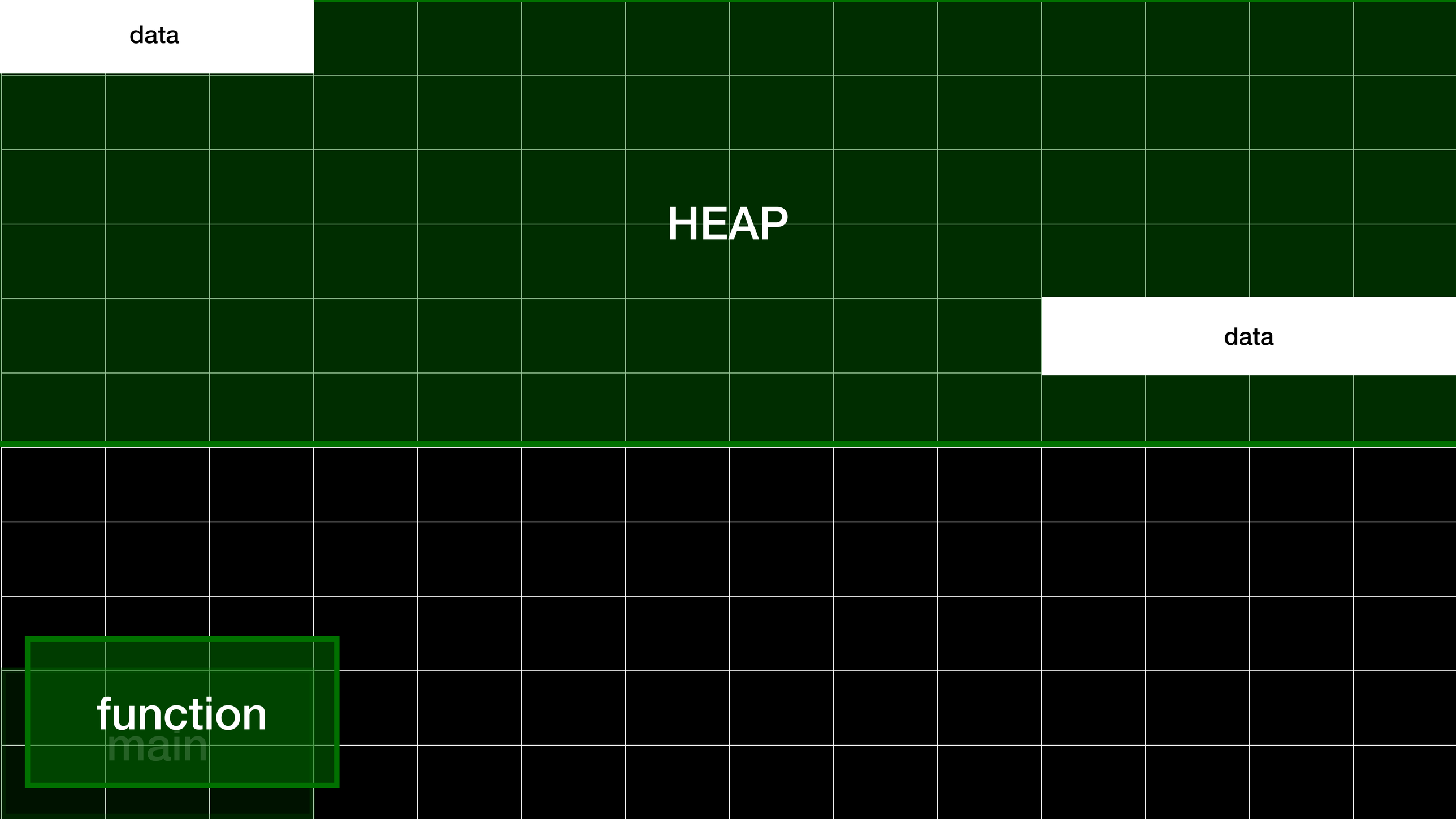
If only we had a way to manage memory ourself, outside stack frames

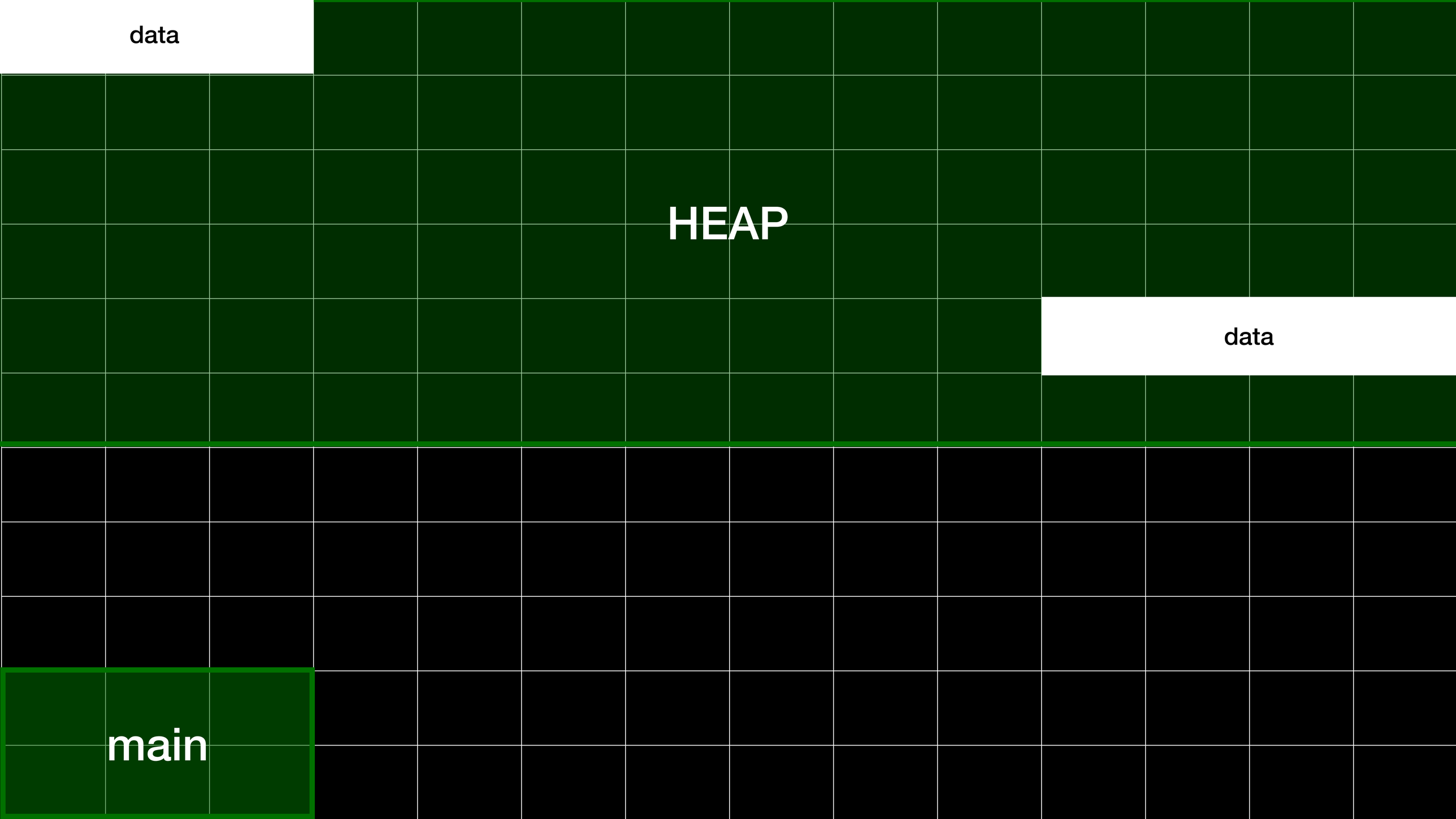
HEAP

data

HEAP


data
HEAP
data


data
HEAP
data
function
main


data
HEAP
data
main

How do we access heap memory?

Pointers

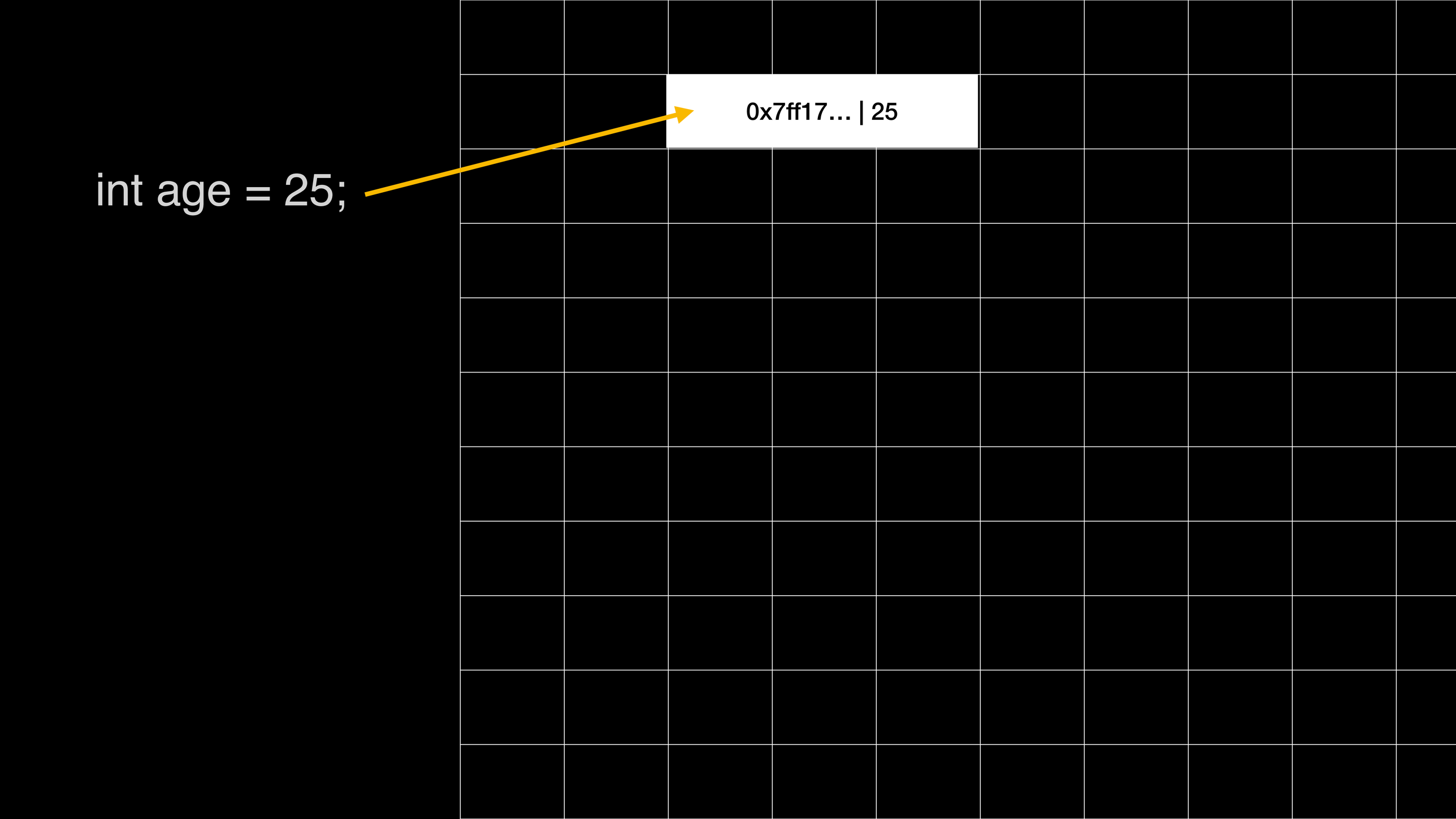
Pointers are a variable, which store a memory address

“Address of” operator looks up the address of a value

```
my_pointer = &age;
```

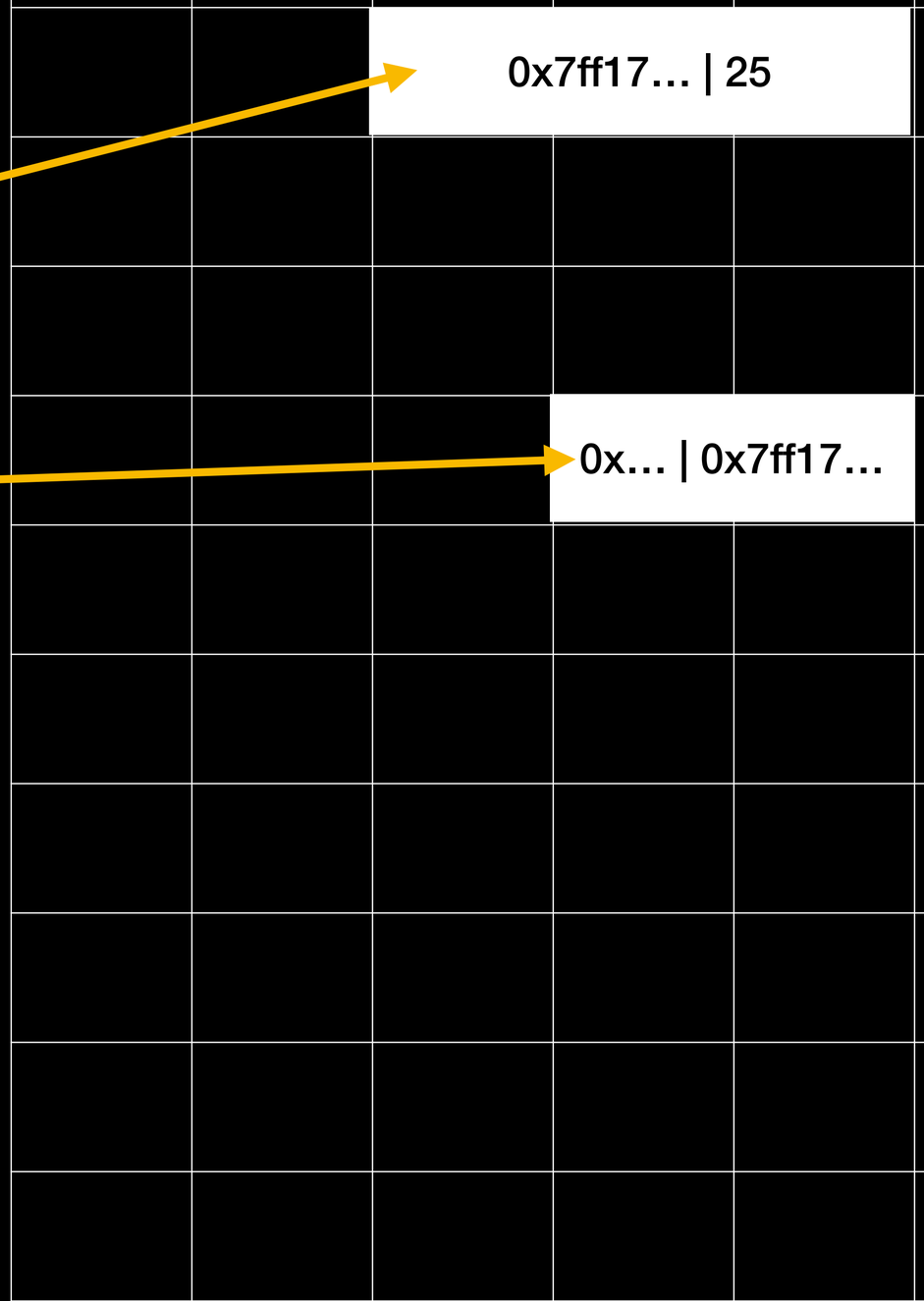
“dereference” operator looks up the address of a value

```
my_age = *age;
```


int age = 25;
0x7ff17... | 25

int age = 25;

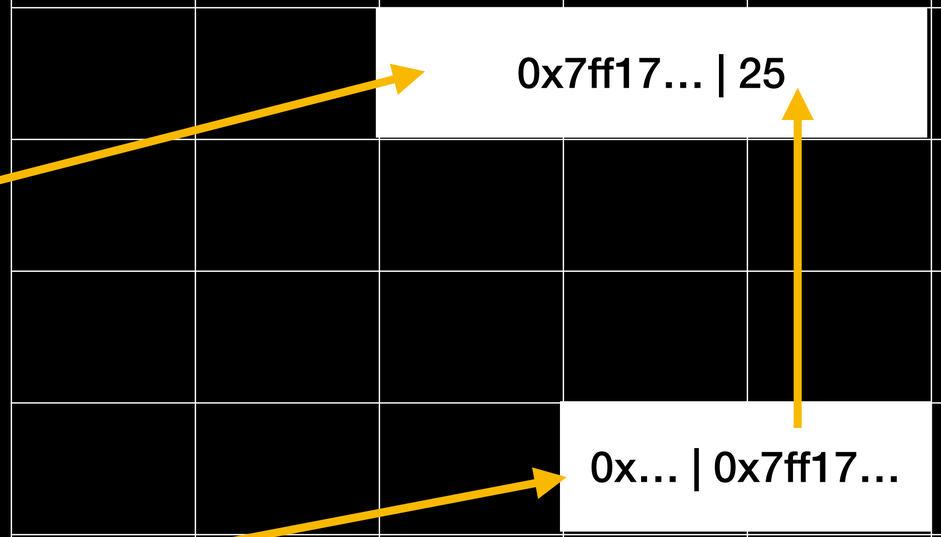
int *age_pointer = &age;

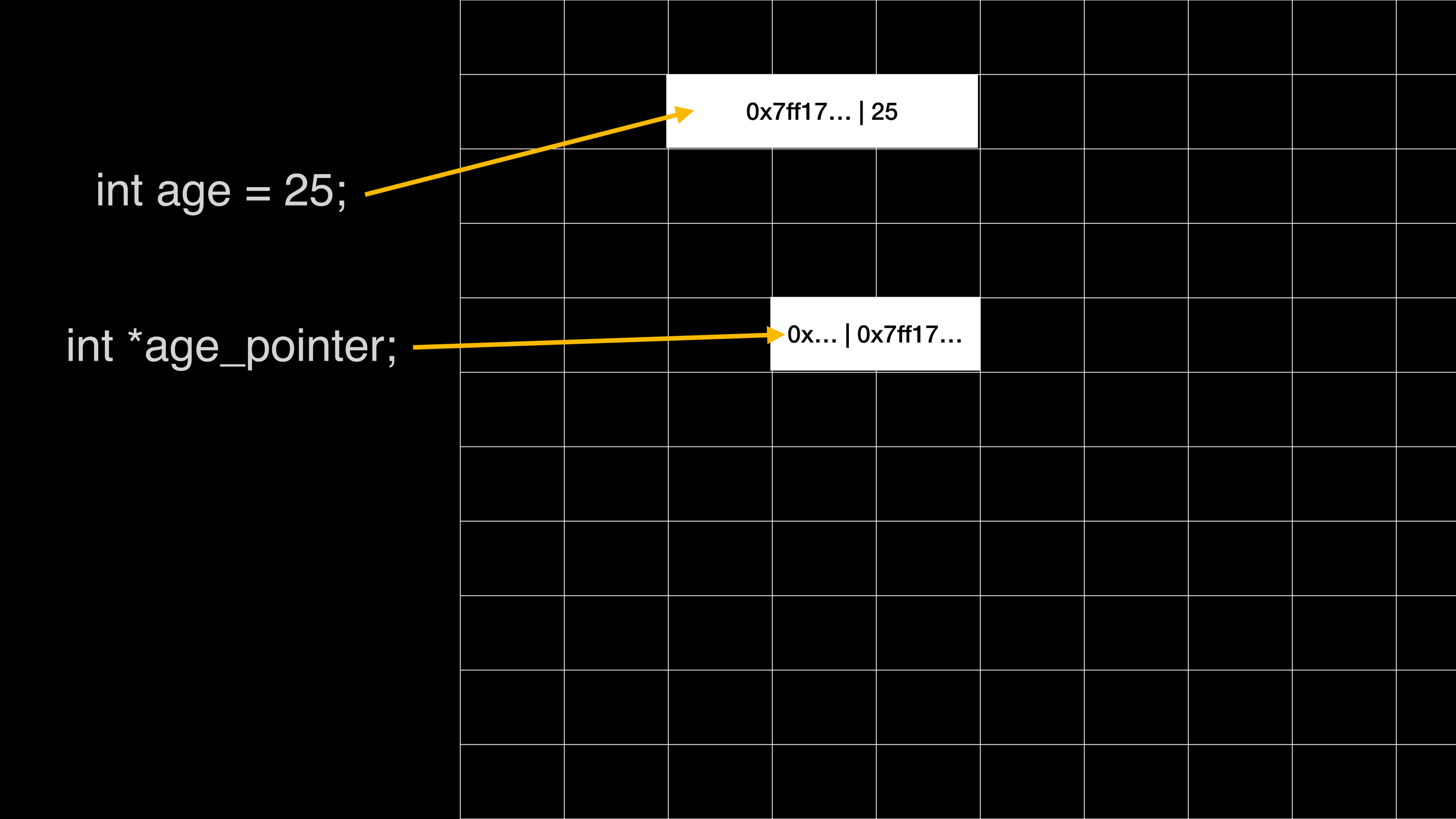


int age = 25;

int *age_pointer = &age;

printf(“%p”, age_pointer);




int age = 25;
int *age_pointer;
0x7ff17... | 25
0x... | 0x7ff17...

Demo

Pointers to use the Heap

```
int *integer_on_heap = malloc(sizeof(*integer_on_heap));
```

```c
int *integer_on_heap = malloc(sizeof(*integer_on_heap));
```

```
int *integer_on_heap = malloc(sizeof(*integer_on_heap));
```

```
int *integer_on_heap = malloc(sizeof(*integer_on_heap));
```

```
int *integer_on_heap = malloc(sizeof(*integer_on_heap));
```

Declare an array on the heap:

```
int *int_array_on_heap = malloc(sizeof(*integer_on_heap) * size);
```

Increase an existing allocation (and copy over the data)

```
new_p = realloc(my_integer_array, sizeof(*my_integer_array) * new_size);
```