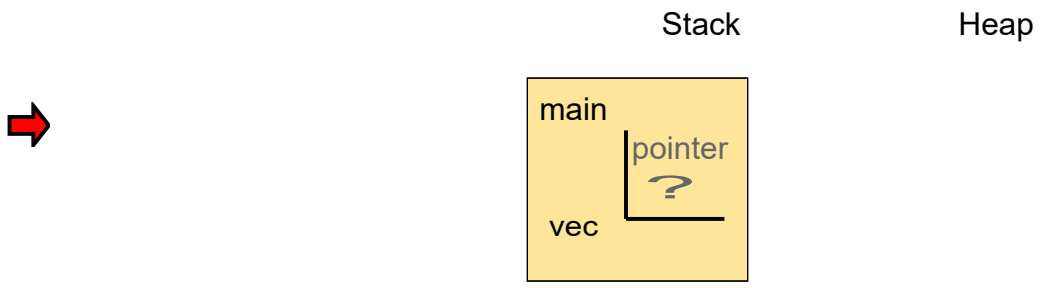
Pointers and Arrays

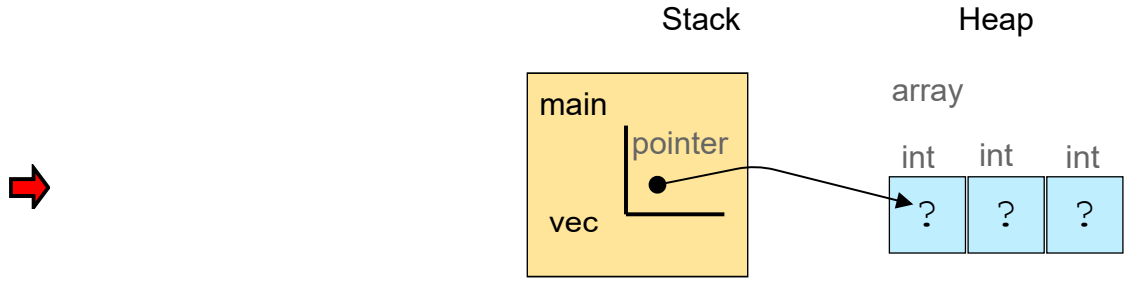
In this lesson, we'll learn how C uses pointers to reference arrays.
'arr[3]' is an example of pointer arithmetic.

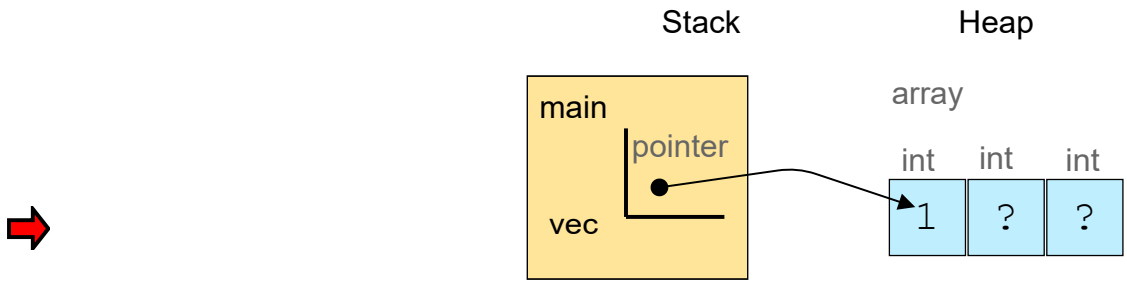
When you declare an array using an expression like `int vec[5];`, what is really happening behind the scenes, is that a block of memory is being allocated (on the stack in this case) large enough to hold 5 integers, and the `vec` variable is a pointer that points to the first element in the array. When you index into the array with an expression like `printf("vec[2]=%d\n", vec[2]);` what is really happening is that C is using **pointer arithmetic** to step into the array the appropriate number of times, and reading the value in the memory location it ends up in. So if you ask for the 3rd element of the `vec` array using `vec[2]` then C is first looking at the location pointed to by `vec` (the first element of the array), and stepping **two integers** forward, and then reading the value it finds there (`vec[2]`).

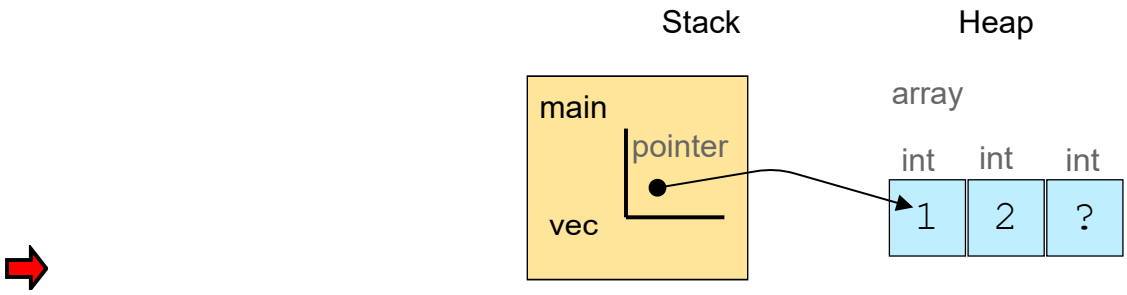
The explicit way of doing this would be to use `malloc()` or `calloc()` to allocate the array (in this case on the heap) and then use pointer arithmetic to read off the 3rd value:

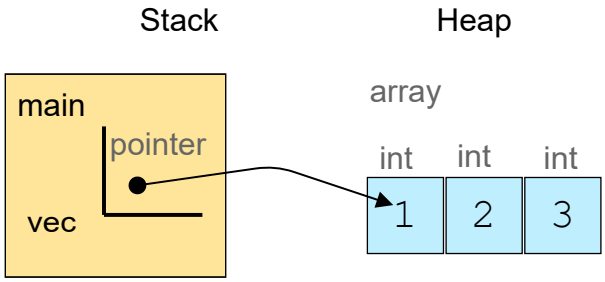
```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int *vec;
  vec = (int*)malloc(sizeof(int) * 3);
  vec[0] = 1;
  vec[1] = 2;
  vec[2] = 3;
  printf("vec[2]=%d\n", *(vec+2));
  free(vec);
  return 0;
}
```

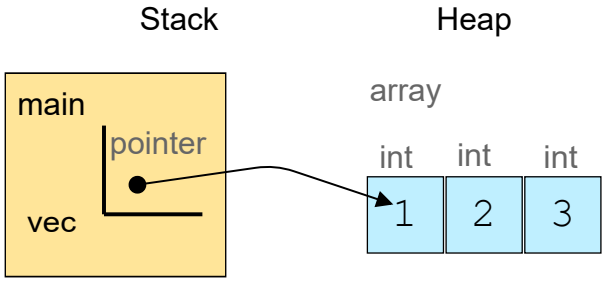

Stack
Heap
main
pointer
?
vec


Stack
Heap
main
pointer
vec
array
int
int
int
?
?
?

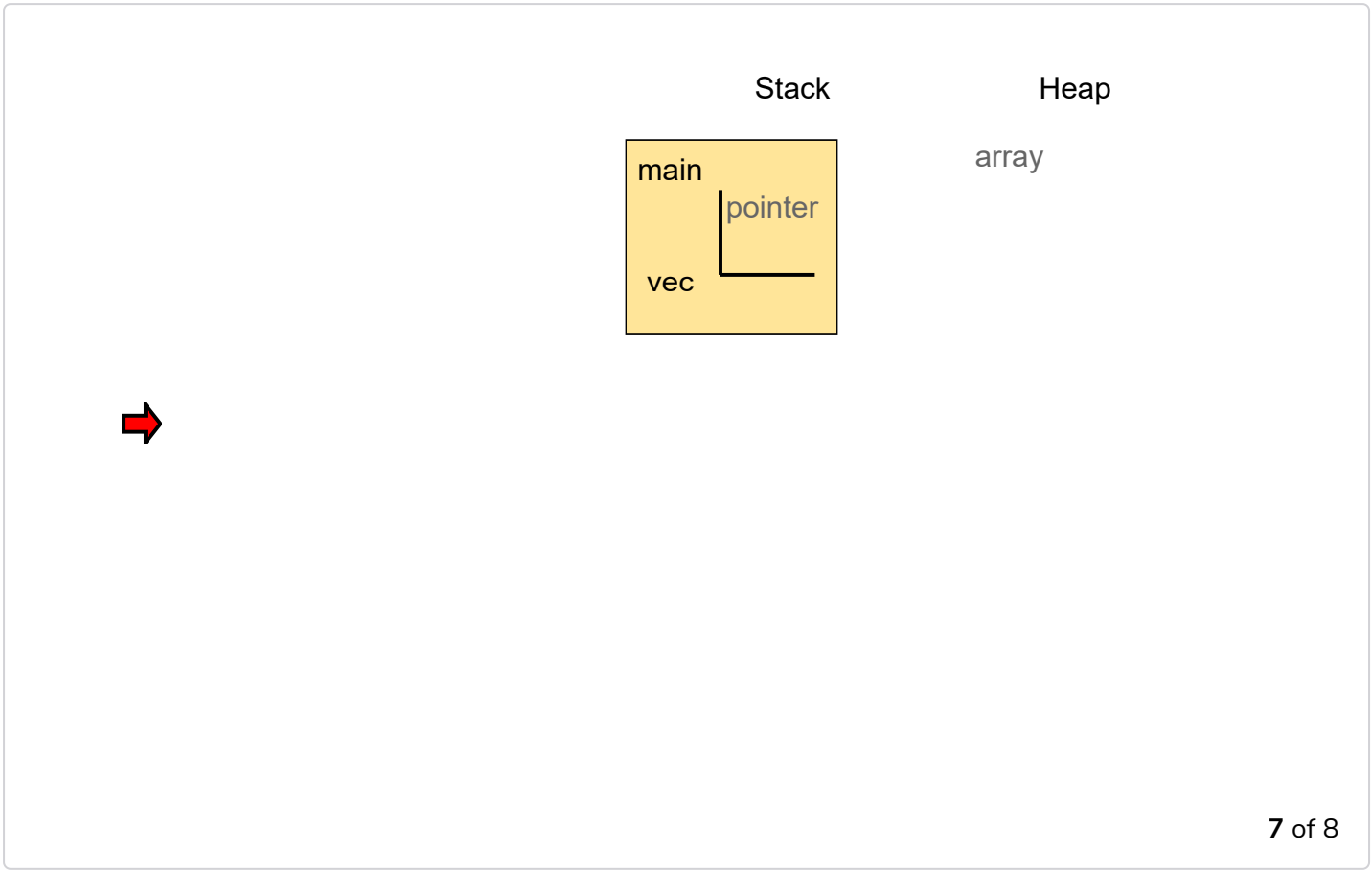

Stack
Heap
main
pointer
vec
array
int
int
int
1
?
?

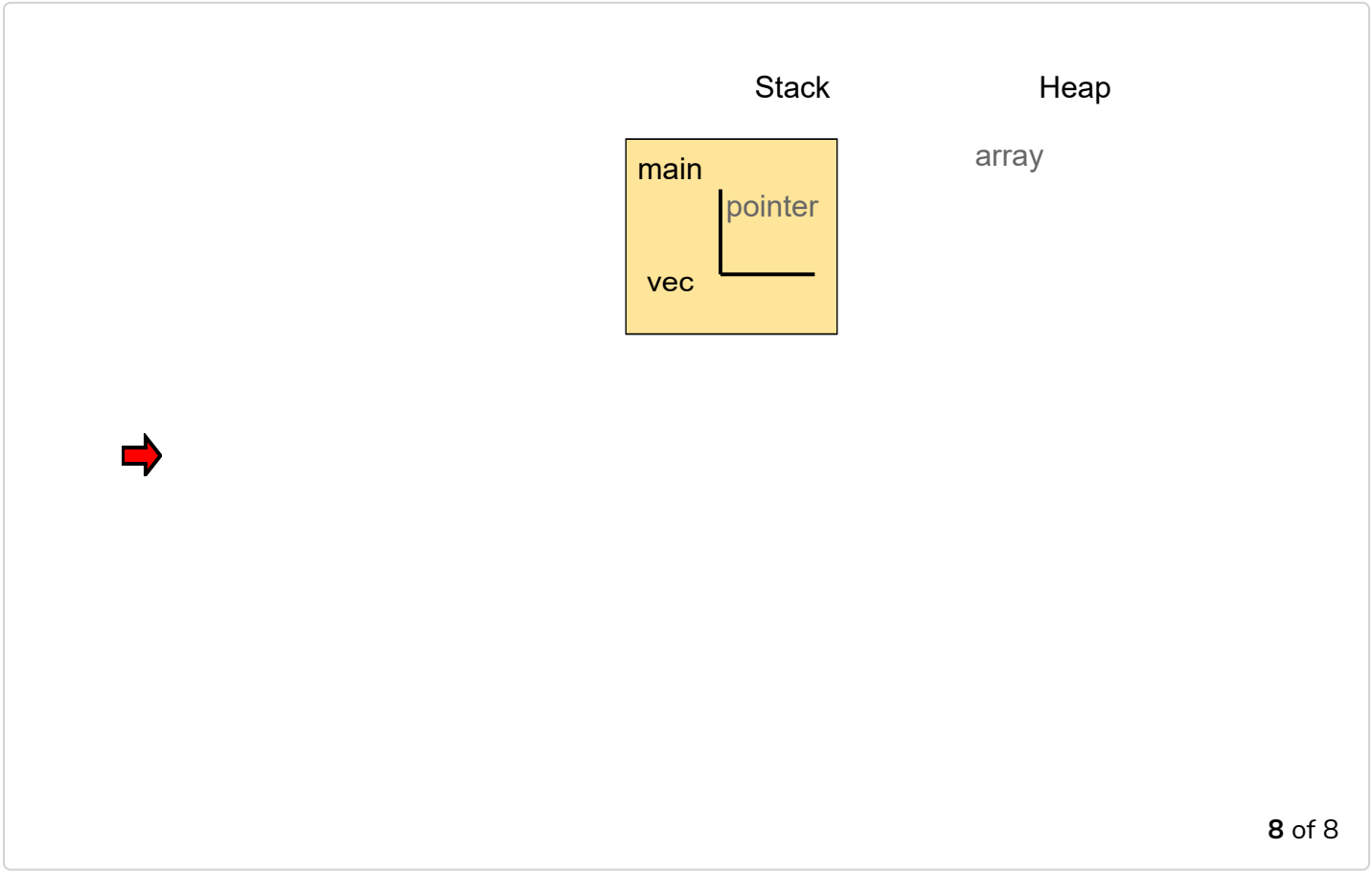

Stack
Heap
main
pointer
vec
array
int
int
int
1
2
?





Output: vec[2] = 3



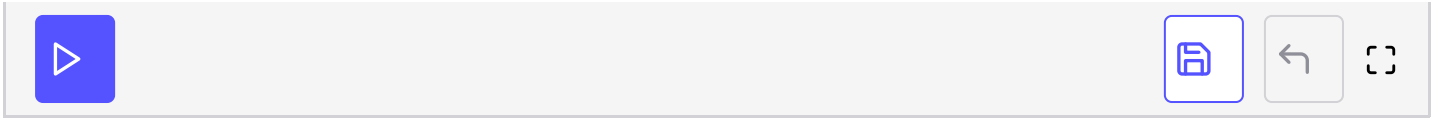


As you can see from the above slideshow, the expression `*(vec+2)` essentially says, go to the location that the pointer `vec` points to, take two steps (each step the size

of an `int`) and the read off the value you find there. How does C know the size of each step to take? Remember when you declare a pointer, you have to declare the type that it points to. So when we declared our `vec` pointer, we declared it as a pointer to `int`. This means when you use pointer arithmetic, C knows the size of each step.

The `*(vec+2)` notation is really just a demonstration of what is happening behind the scenes, there is rarely a need to use such an explicit (and so difficult to read) expression to index into arrays. The more common way of indexing into arrays (and the more readable way) is to use the `vec[2]` notation:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int *vec;
  vec = (int*)malloc(sizeof(int) * 3);
  vec[0] = 1;
  vec[1] = 2;
  vec[2] = 3;
  printf("vec[2]=%d\n", vec[2]);
  free(vec);
  return 0;
}
```



If you remember, another way of grouping data in C was to make a struct. And unsurprisingly, pointers work with structs too! We'll discuss that next.