Lecture #6 - C

AMath 483/583

Announcements

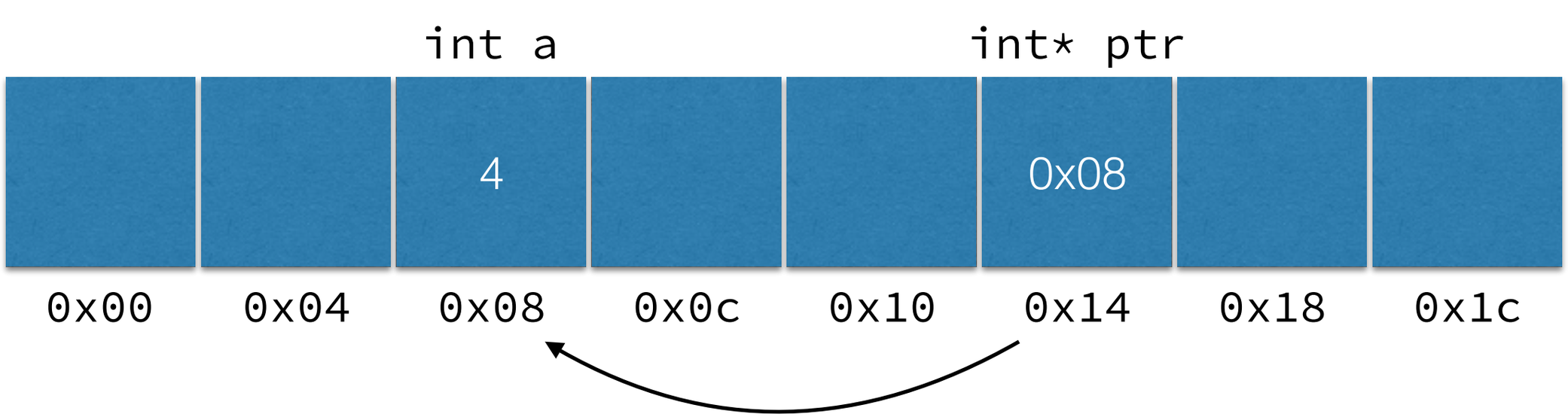
- Homework #2 released next Tuesday
- Homework #1 due not next Monday but Monday after (25 April)
- (Try to get Homework #1 done sooner.)
- Primary references for week will be updated shortly after class.
- Lectures repo updated

Last Time…

- Basic C - variables, conditionals, loops
- Compile-Time arrays
- Basic Pointers

Pointers

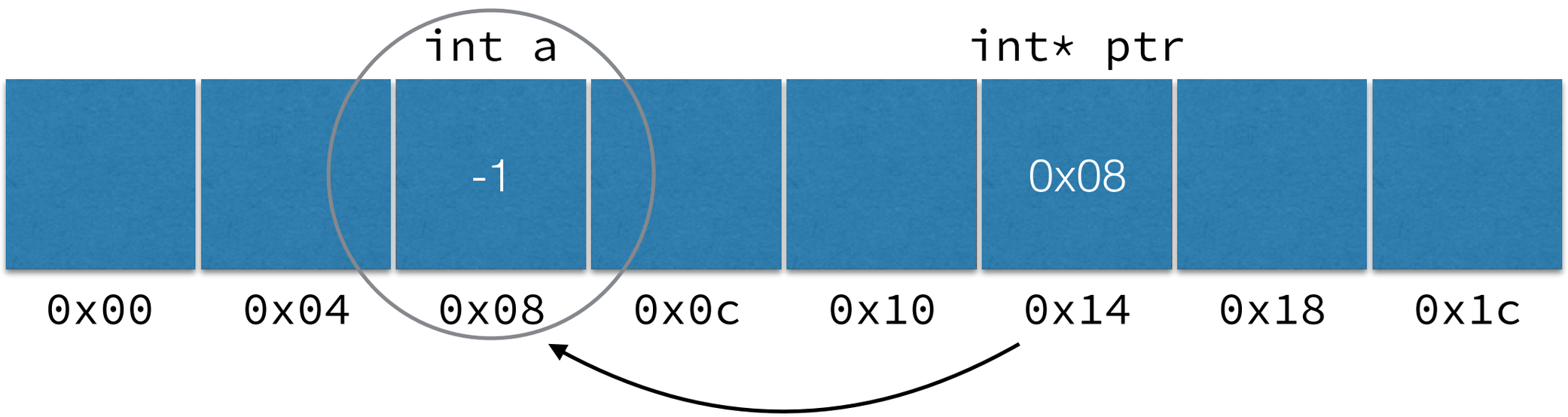
```
int a = 4;
int* ptr = &a;
&ptr == 0x14;
*ptr == 4;
```



Execute: `*ptr = -1;`

Pointers

```
int a = 4;
int* ptr = &a;
&ptr == 0x14;
*ptr == 4;
```



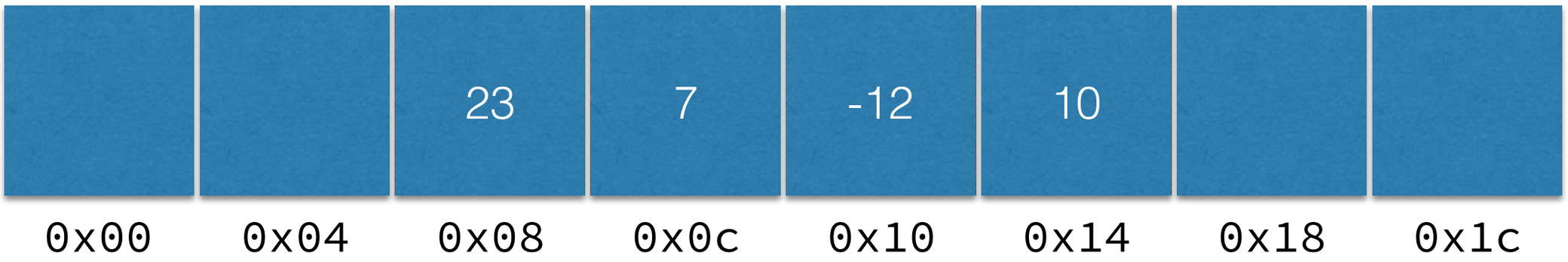
Execute: *ptr = -1;

Excellent Cliff-Hanger

Arrays are pointers

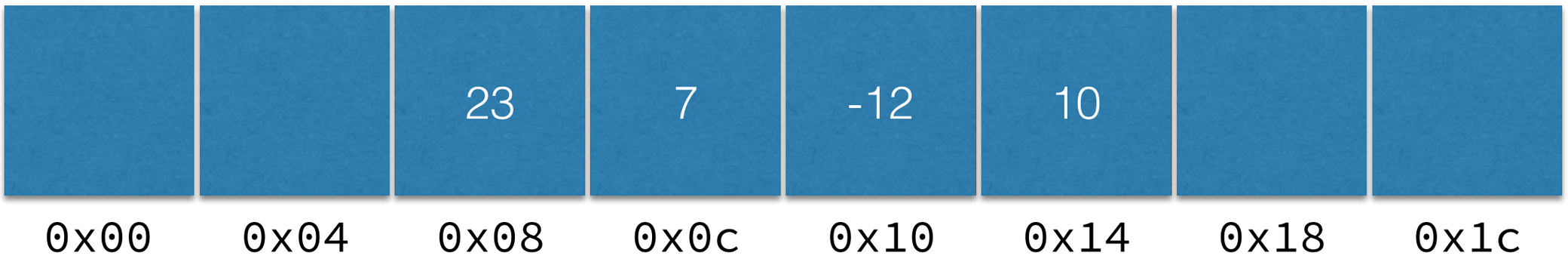
Pointers and Arrays

```
int arr[4] = {23, 7, -12, 10};
```



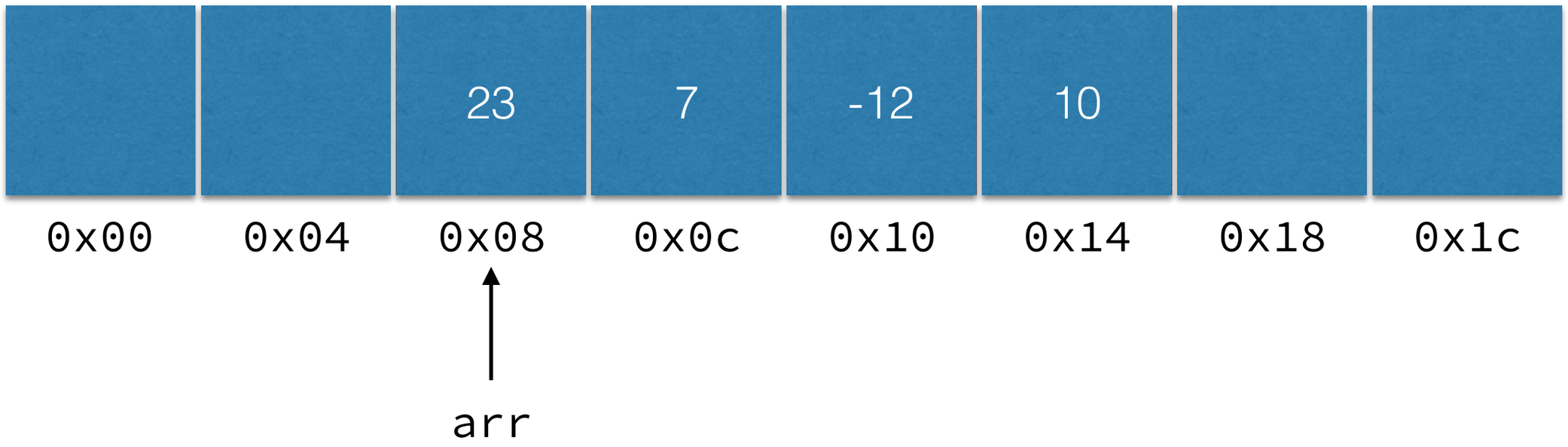
Pointers and Arrays

```
arr[0] == 23;
arr[1] == 7;
arr[2] == -12;
arr[3] == 10;
```



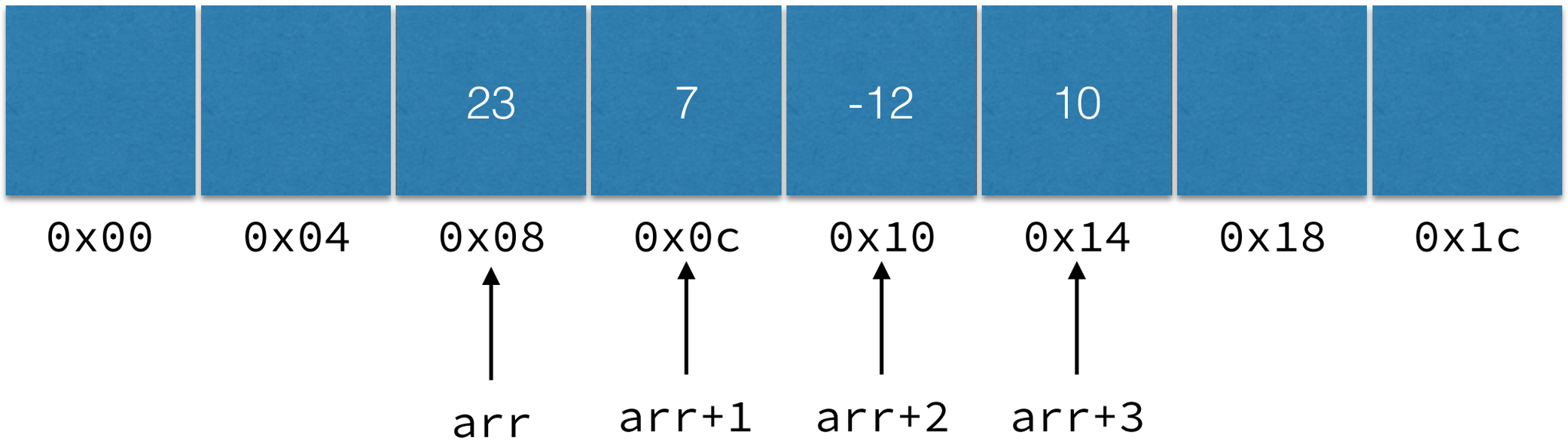
Pointers and Arrays

```
arr == 0x08;
```



Pointers and Arrays

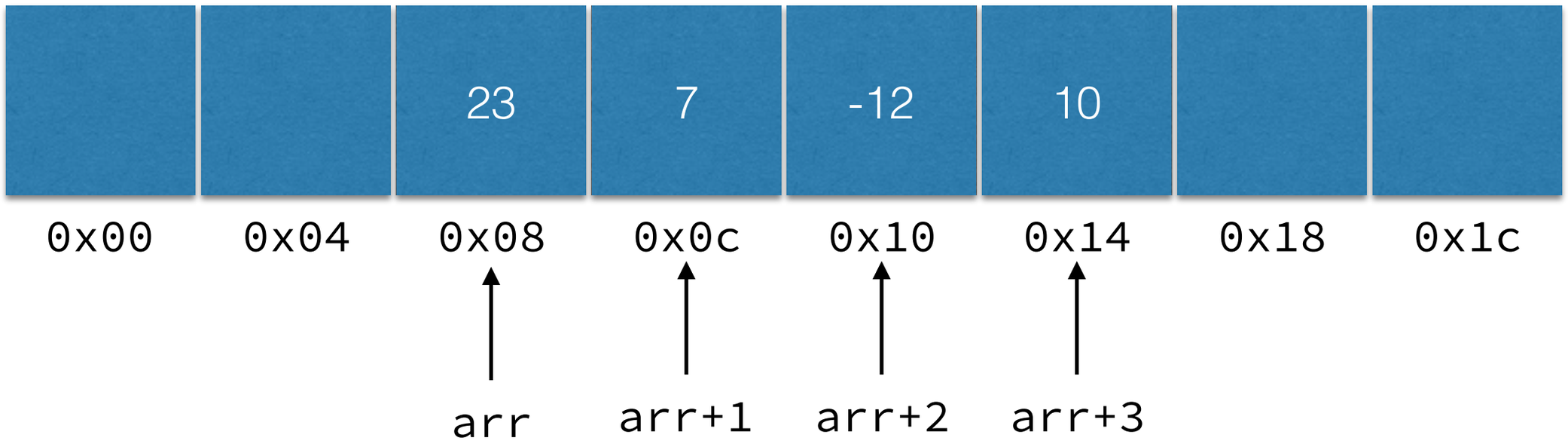
```
 arr == 0x08;
arr+1 == 0x0c;
arr+2 == 0x10;
arr+3 == 0x14;
```



"`arr + n`" really means "`arr + n*sizeof(int)`"

Pointers and Arrays

```
*(arr) == 23;
*(arr+1) == 7;
*(arr+2) == -12;
*(arr+3) == 10;
```



`arr[n]` is syntactic sugar for `*(arr+n)`

Demo

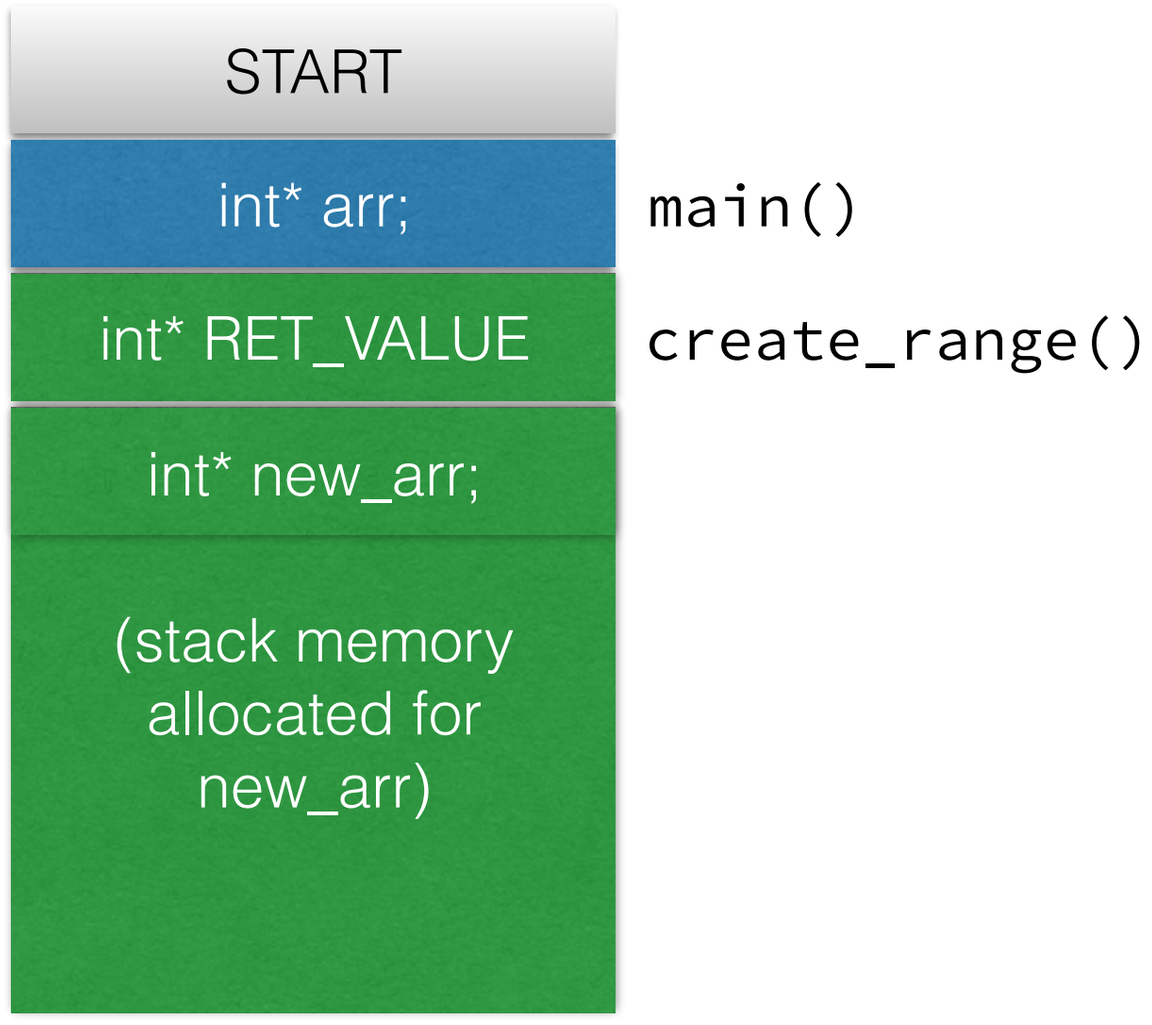
Pointers, Arrays, and Functions (`print_array`, `scale_array`, `create_range`)

Stack Problems

- In the demo;we tried the following:
  - `main()`: create a pointer `int* arr`;
  - `create_range(int n)`: create an array (pointer) of length n equal to [0,1,…,n-1]
  - `create_range(int n)`: return said range pointer
  - `main()`: assign outpt of `create_range()` to `arr`;

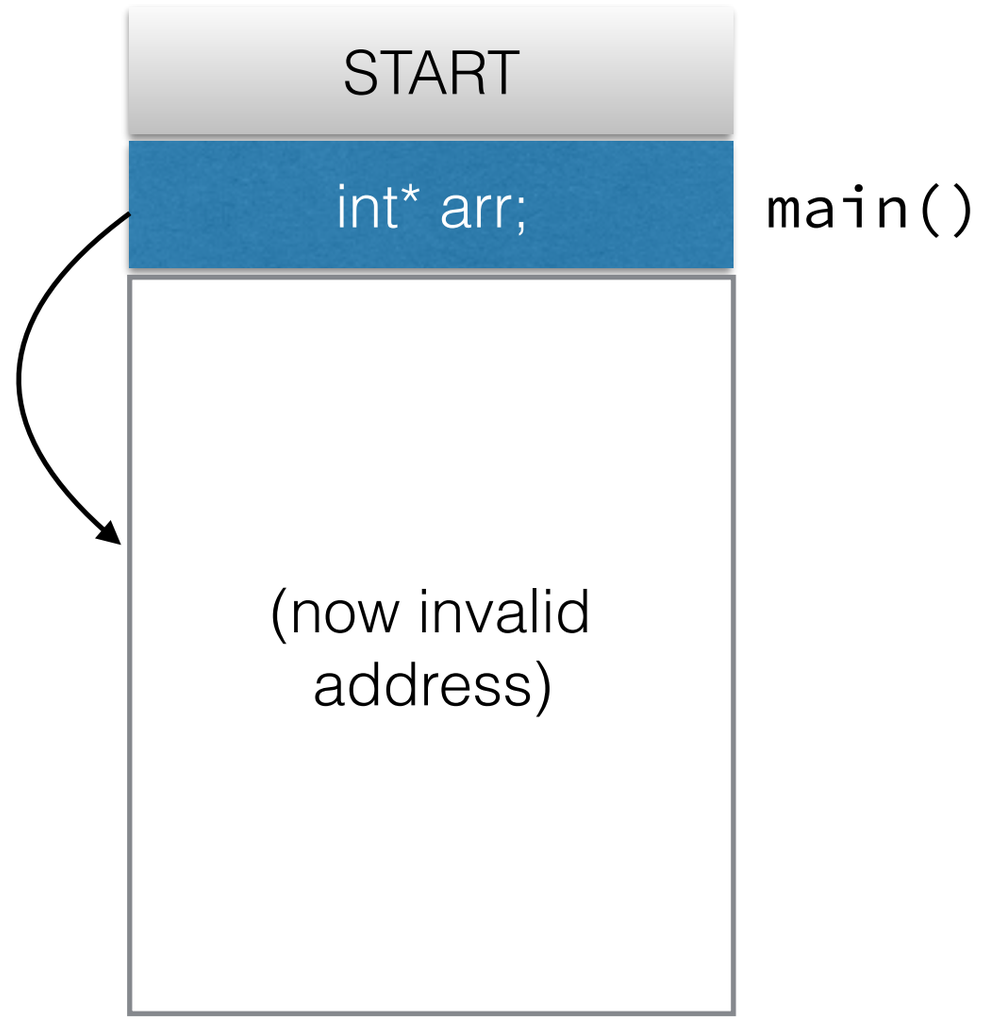
Stack Problems

- Problem:
  - once `create_range()` finishes execution it's local allocations disappear
  - pointer is copied and sent to `main()`
  - HOWEVER, the pointer now points to invalid memory
- (Remember: `new_arr` points to the beginning of the array)



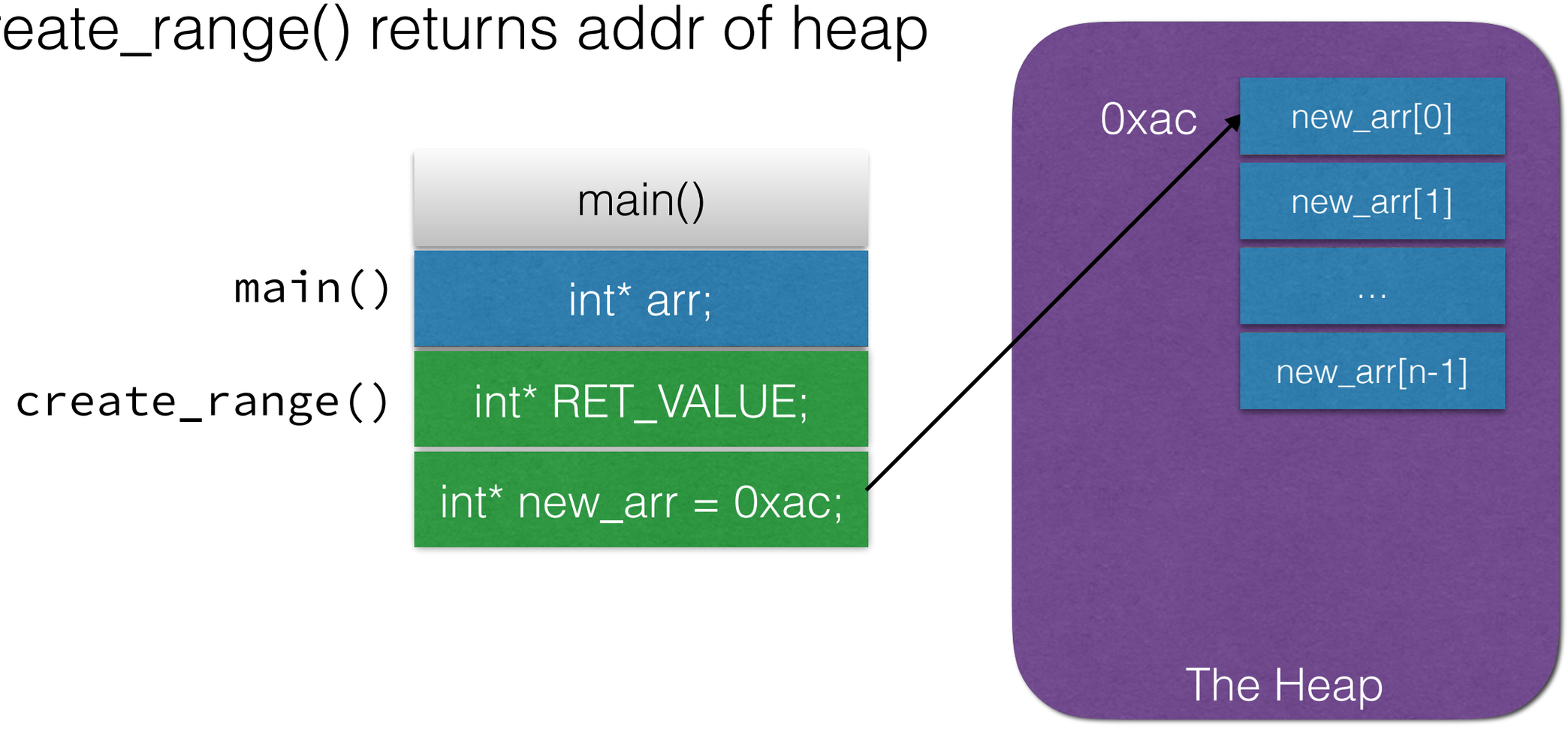
Stack Problems

- Problem:
  - once `create_range()` finishes execution it's local allocations disappear
  - pointer is copied and sent to `main()`
  - HOWEVER, the pointer now points to invalid memory



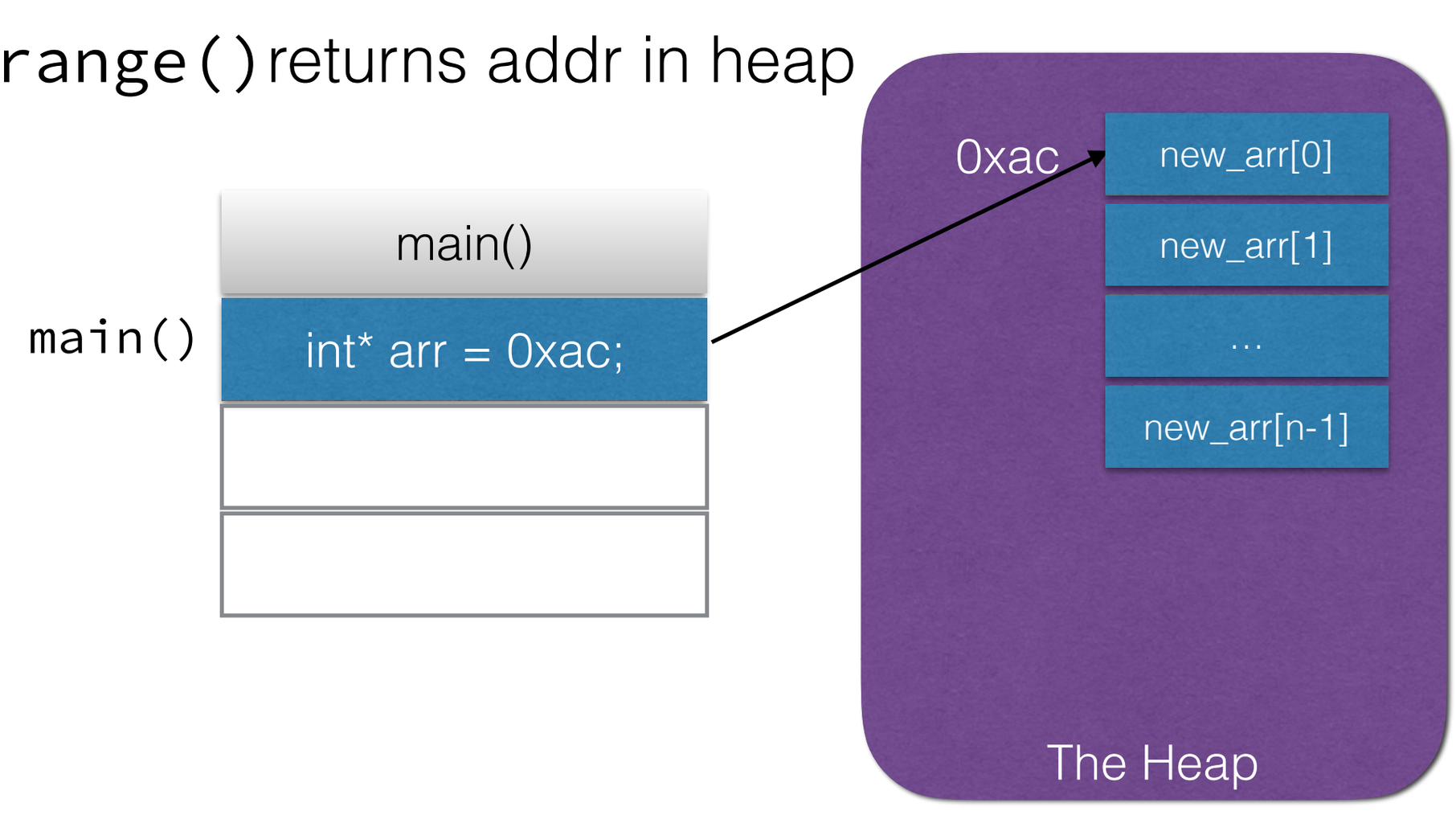
Heap Allocations

- Solution: allocate the memory on the heap
  - heap memory persists after function termination
  - create_range() returns addr of heap



Heap Allocations

- Solution: allocate the memory on the heap
  - heap memory persists after function termination
  - `create_range()` returns addr in heap



Heap Allocations

- Use `malloc()` - defined in `stdlib.h`

```
int* new_arr = (int*) malloc(n * sizeof(int));
```

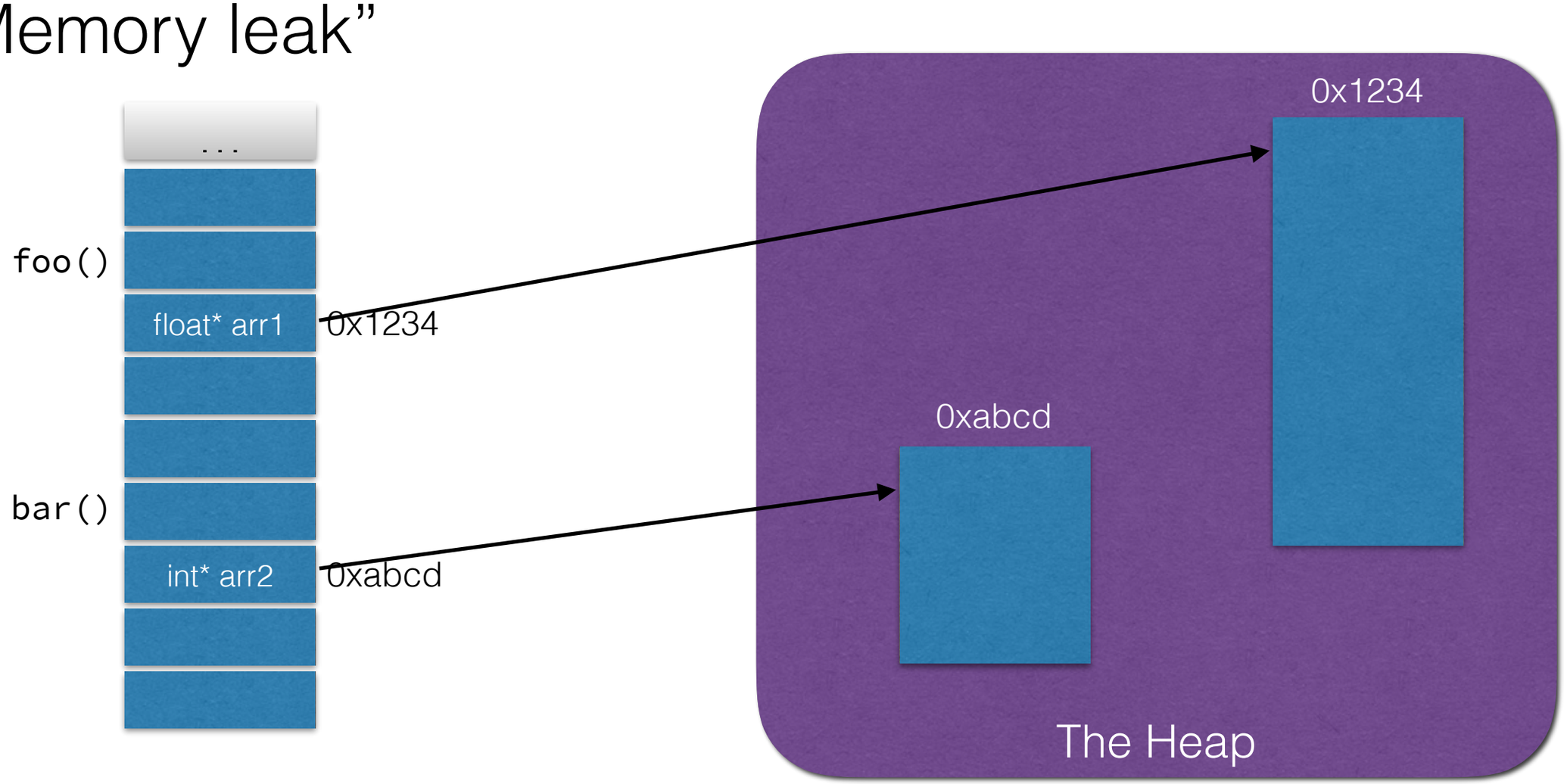
  - returns a pointer of type `void*` to heap memory
  - cast to `int*` (or whatever type you declared)
  - `sizeof(int)` = number of bytes per int

Demo

Heap / Dynamically Allocated Arrays

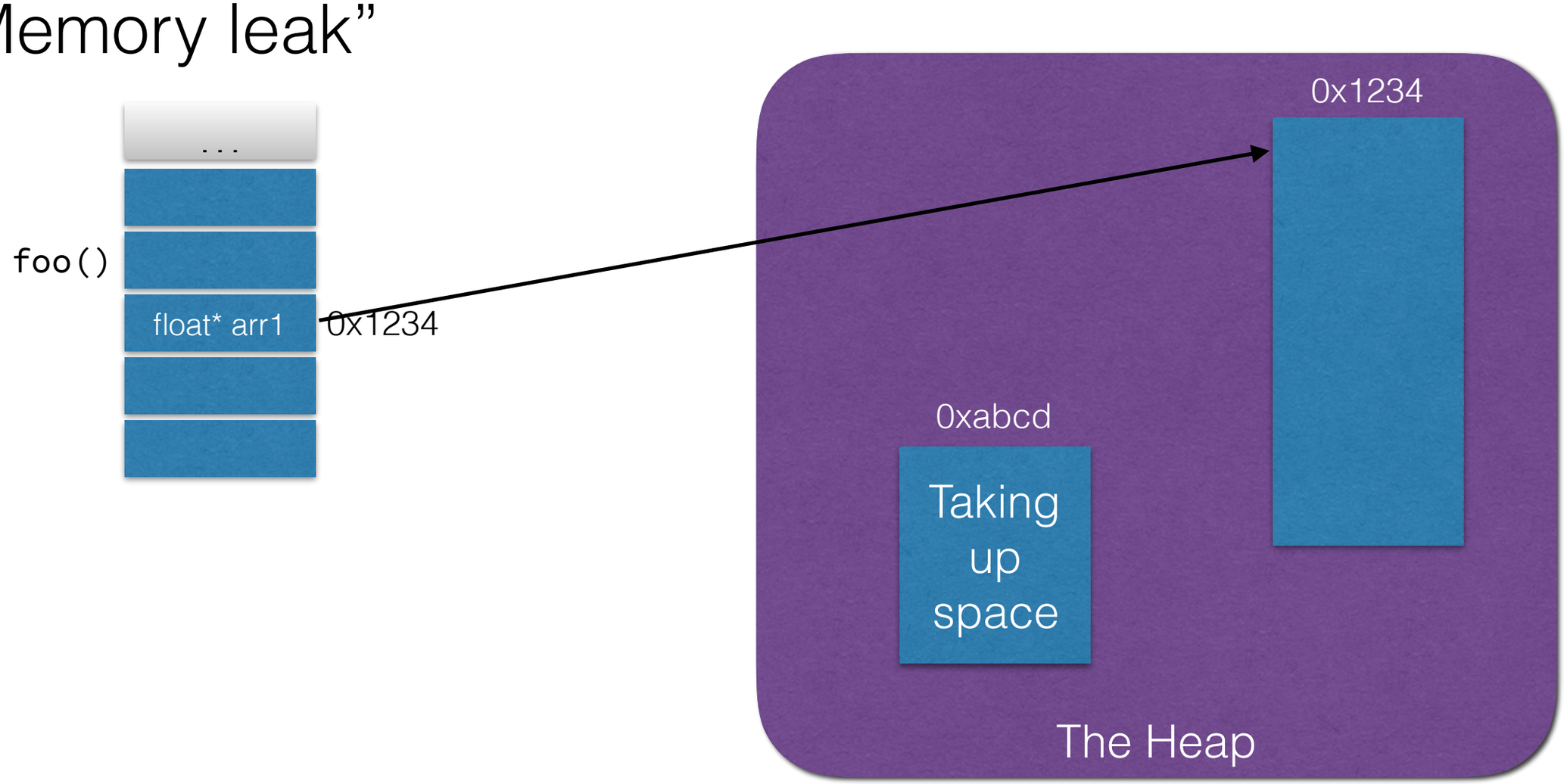
Remember to Free

- Heap allocations stay until manually `free()`'d
- "Memory leak"



Remember to Free

- Heap allocations stay until manually `free()`'d
- "Memory leak"



Valgrind

- Checks for memory leaks.

Common Practices

- `create_range()` allocates heap memory
  - need to “read documentation” to know this
  - easy to lose track of heap allocations
- Alternative: have the “user” make heap allocations (now it’s their responsibility)
  - i.e. use `malloc()` in `main()` and pass pointer to `populate_range()`

Common Practices

- “Return by reference”
  - instead of returning pointer, populate input array

```
void vec_add(int* out, int* v, int* w);
```

Result stored in “output” array

Input arrays

Demo

`vec_add` - return by reference / parameter

Time Remaining

- Function prototypes

- Compiling non-executables