Lists

CS143: lecture 6

Byron Zhong, June 26

Last week in review

- Pointers: Syntax and meanings
- Stack: Frames, pass-by-value, pass-by-reference
- Heap: `malloc` and `free`
- Building structures in the heap via pointers
  - Data structures!
  - This week and next week

The Stack

The Heap

This week's plan

- Lists
  - Array Lists
  - Linked Lists
- Sorting
- More general C stuff

What is a list?

- An ordered collection of elements is a list: [🍏, 🍎, 🍐, 🍊, 🍋]
  - Homogeneous list: all elements have the same type
  - Heterogeneous list: elements may have different type
  - Ordered: 0th, 1st, 2nd, 3rd, ... (does not mean sorted)
- Dynamic size: can grow/shrink as needed
- What operations does a list support?

List Operations

- `append`
- `prepend`
- `len`
- `insert_at`
- `remove_at`
- `is_empty?`
- `at(i)`

Array as a List

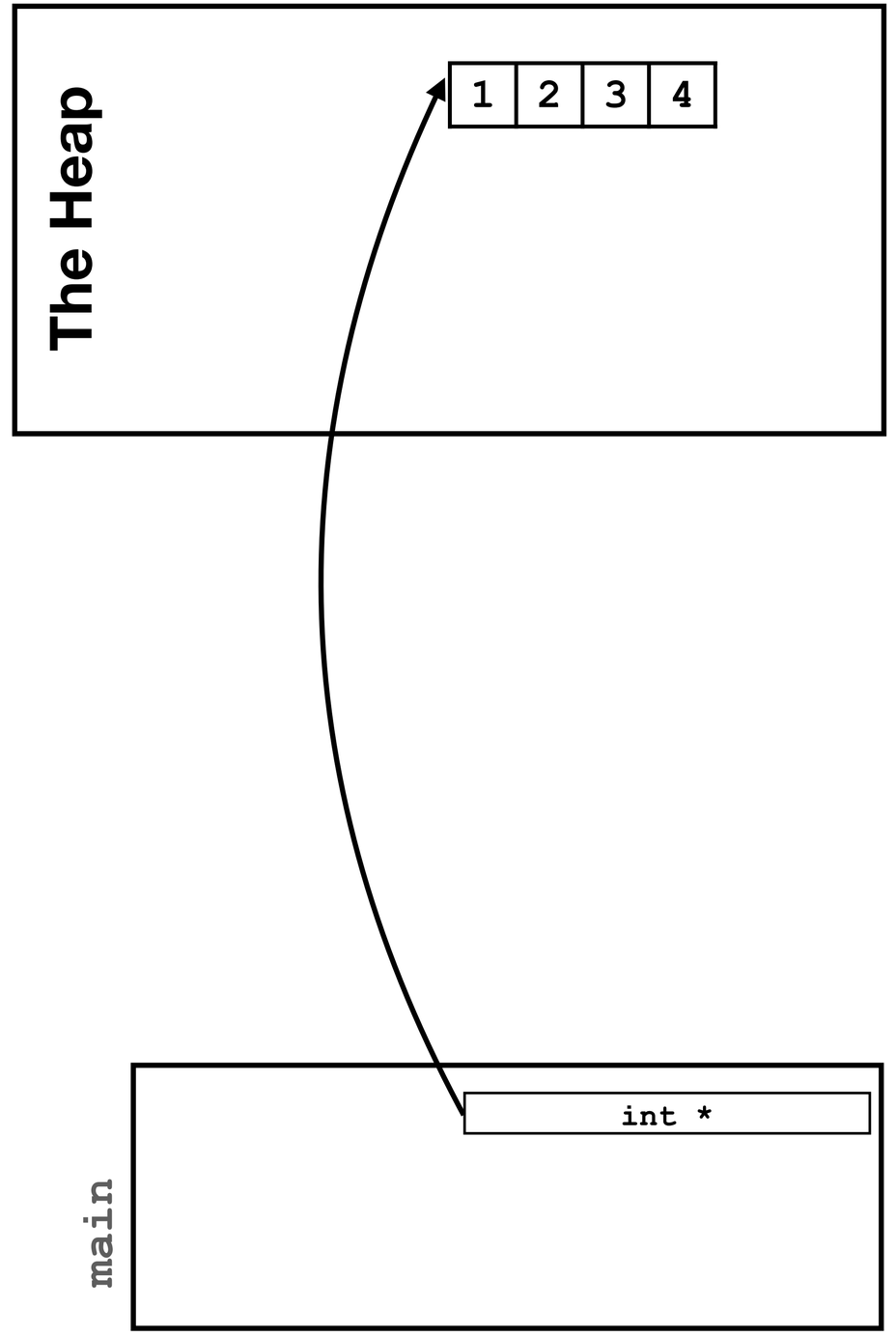
Array

- Can we just use an array to implement a list?
- Almost! Arrays have a fixed size, but ...
  - If it's on the stack, no luck here
  - If it's on the heap, we may have some ways around this

Array

Growing an array

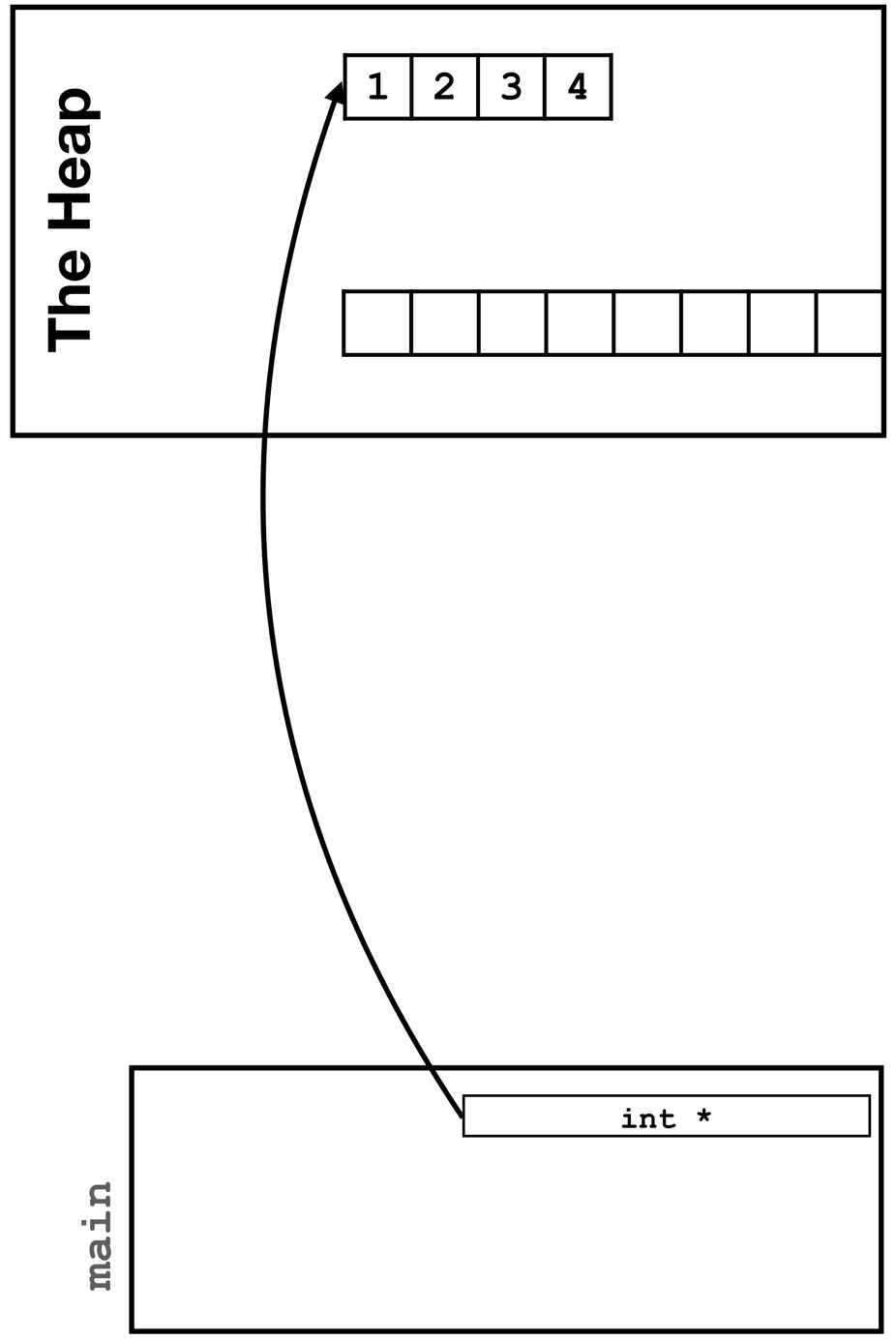
- Create a bigger array.



Array

Growing an array

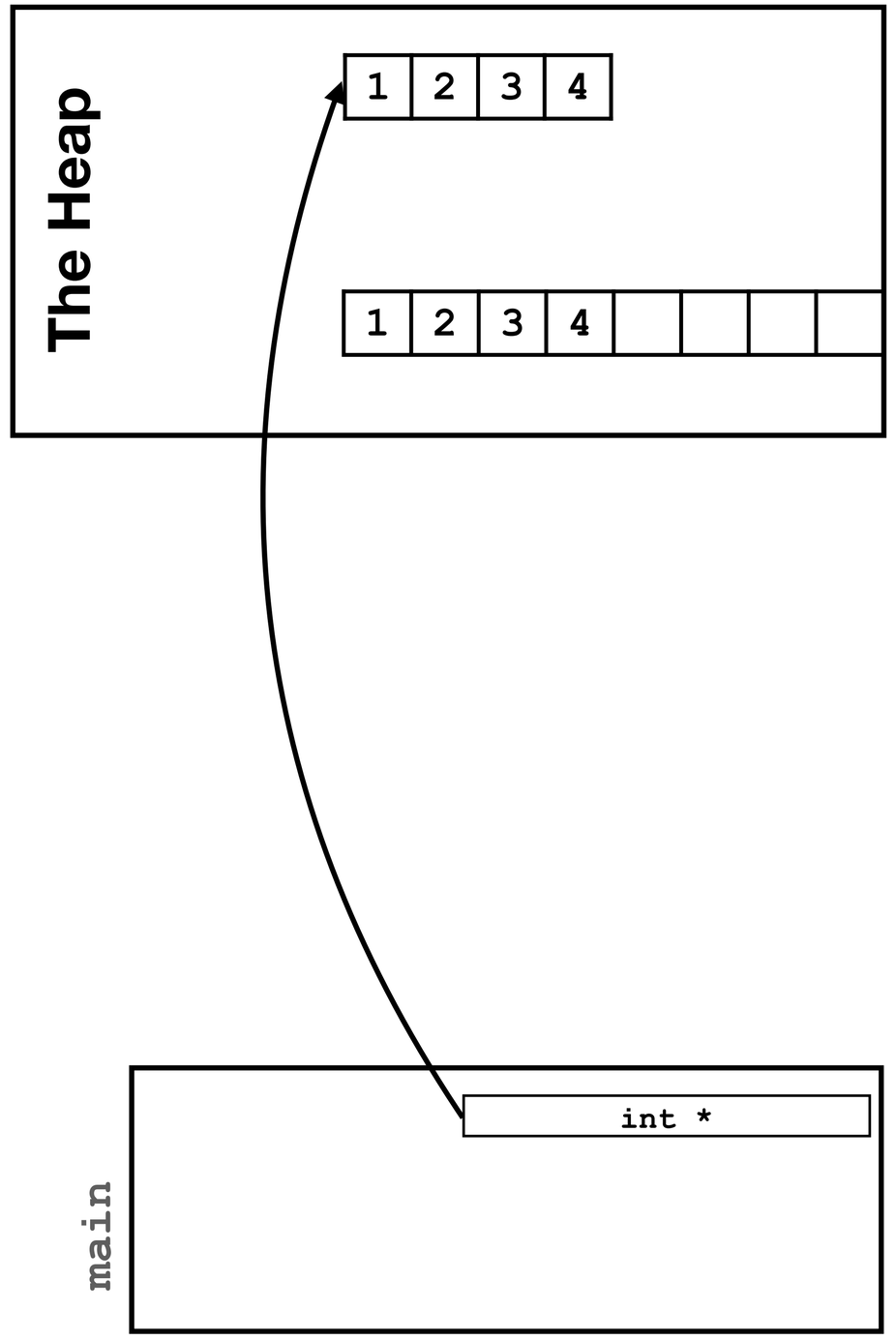
- Create a bigger array.



Array

Growing an array

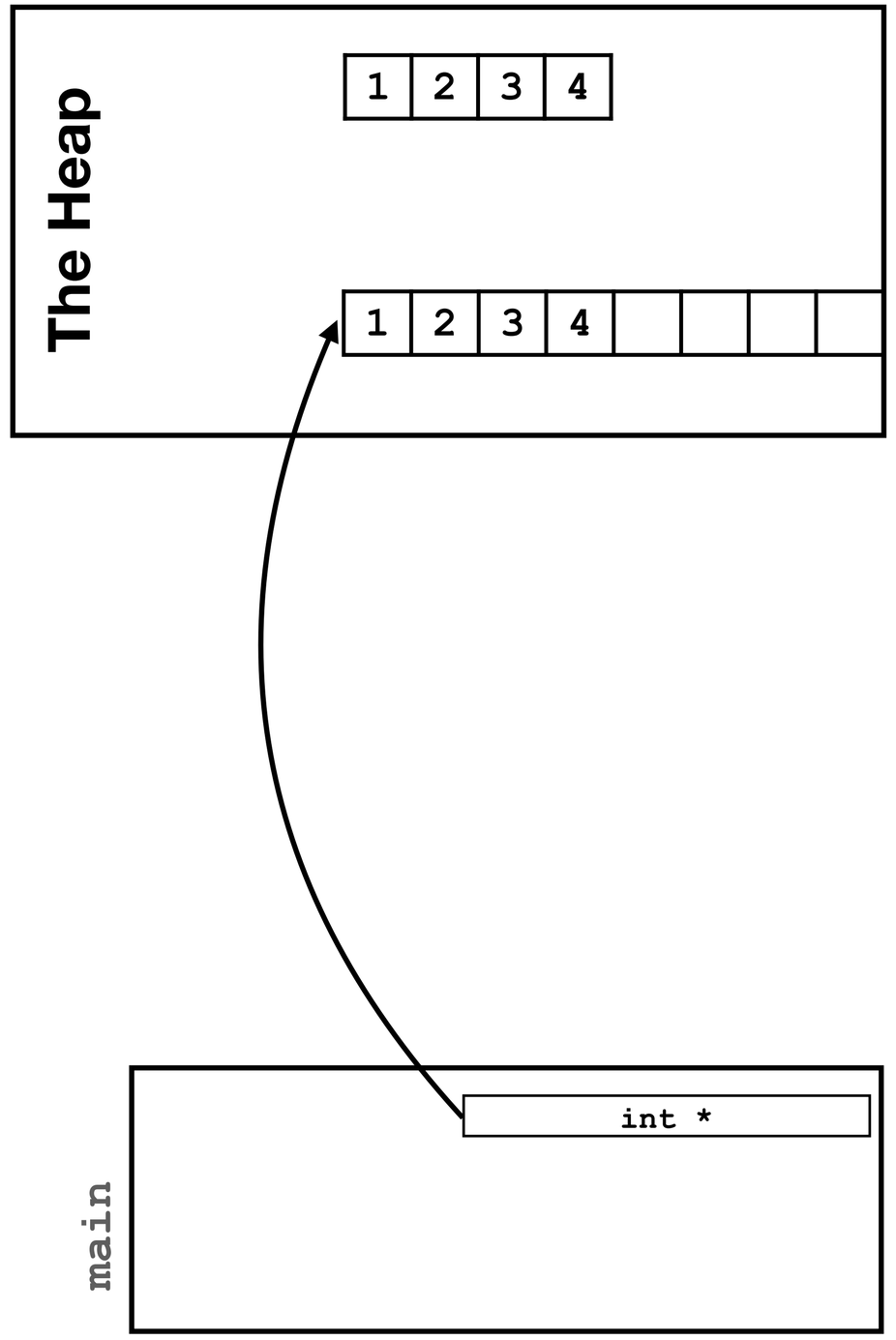
- Create a bigger array.
- Copy the elements into the new array



Array

Growing an array

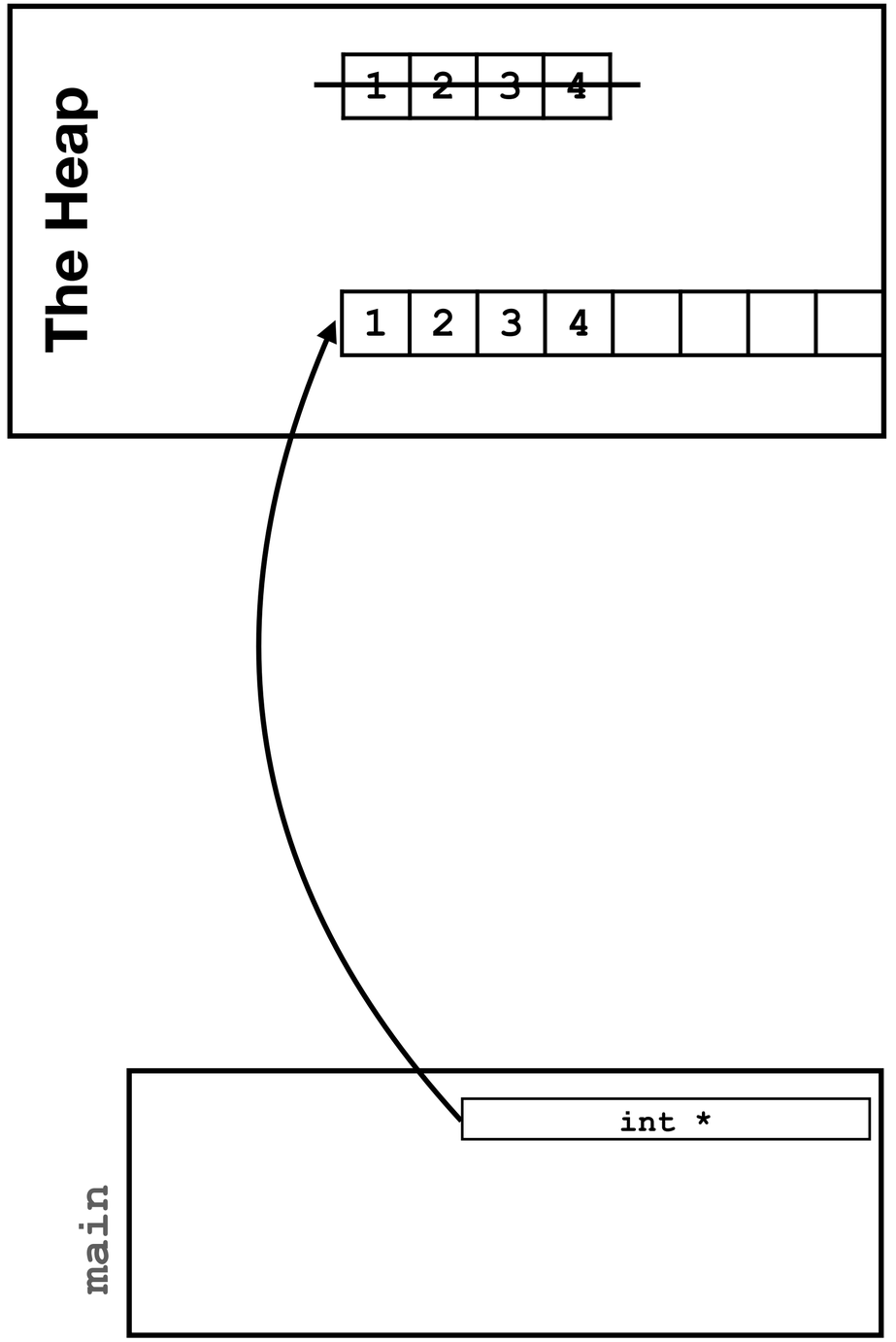
- Create a bigger array.
- Copy the elements into the new array
- Reassign the pointer to point to the new array



Array

Growing an array

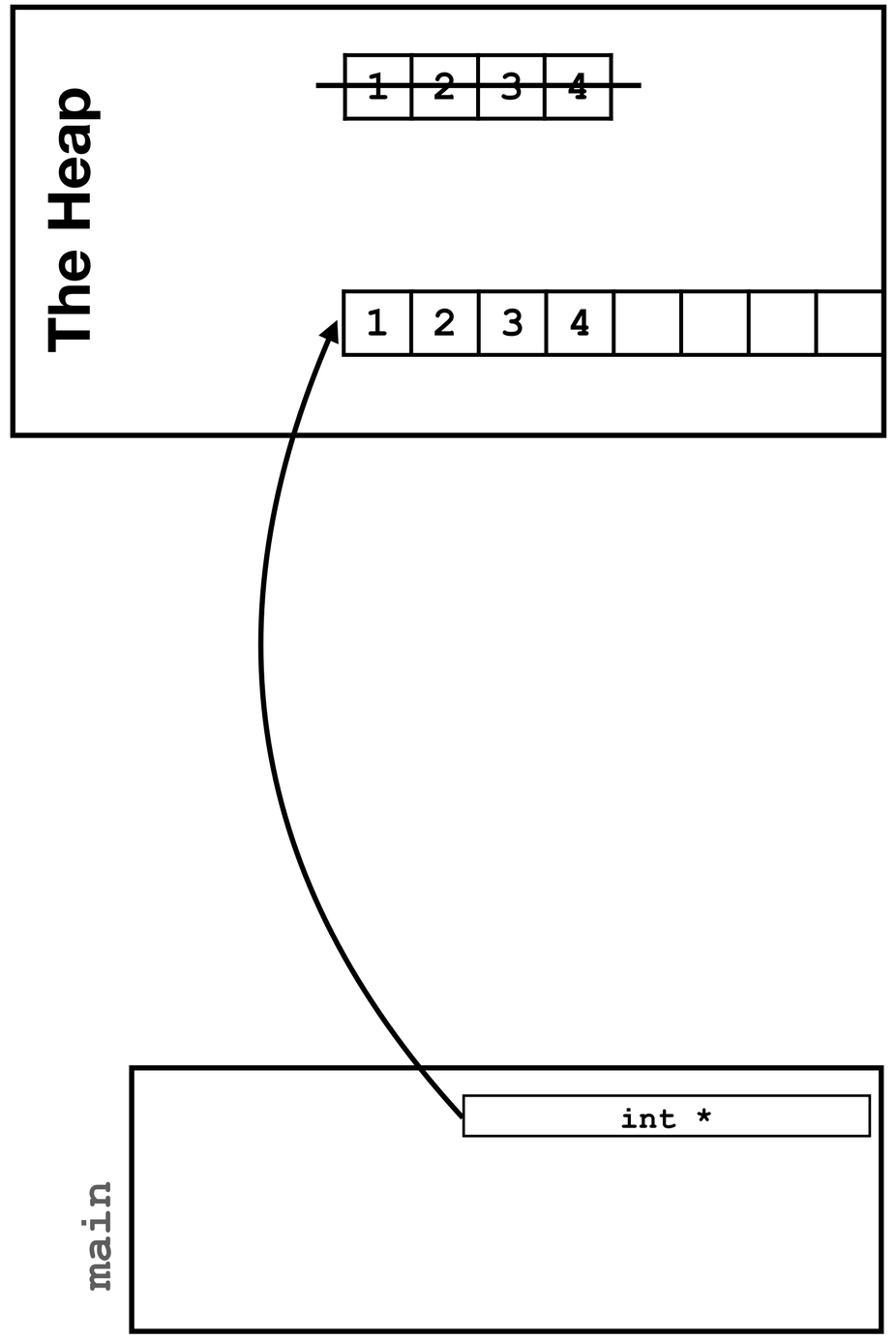
- Create a bigger array.
- Copy the elements into the new array
- Reassign the pointer to point to the new array
- Free the old array



Array

Growing an array

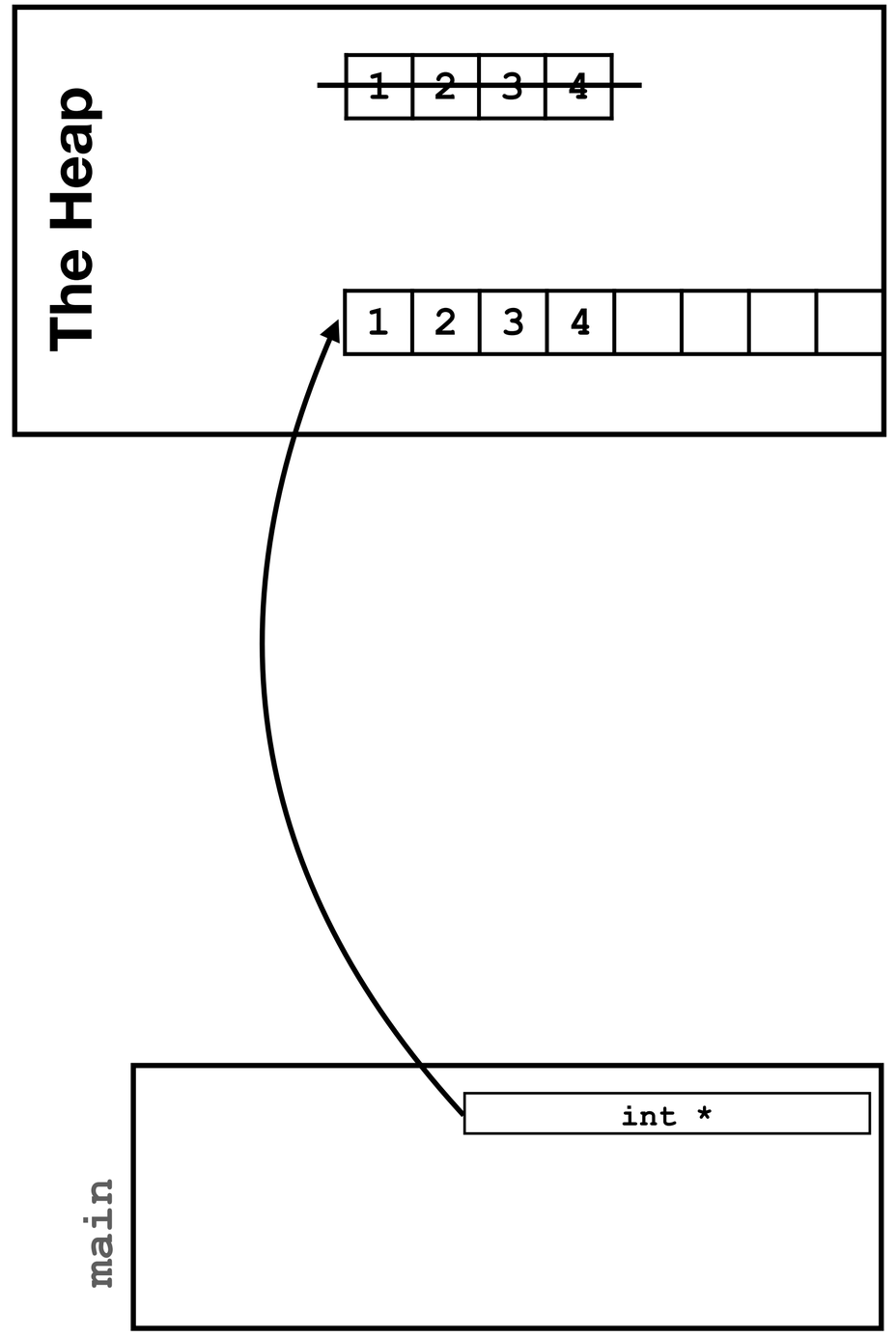
- Pointers serve as an indirection.
  - We aren't changing the size of the array; we are changing which array the pointers point to.
  - By changing the address of the pointer, it seems to the user that we have changed the size of the array.
- We create and delete memory however we want thanks to the heap.



Array

Abstractions

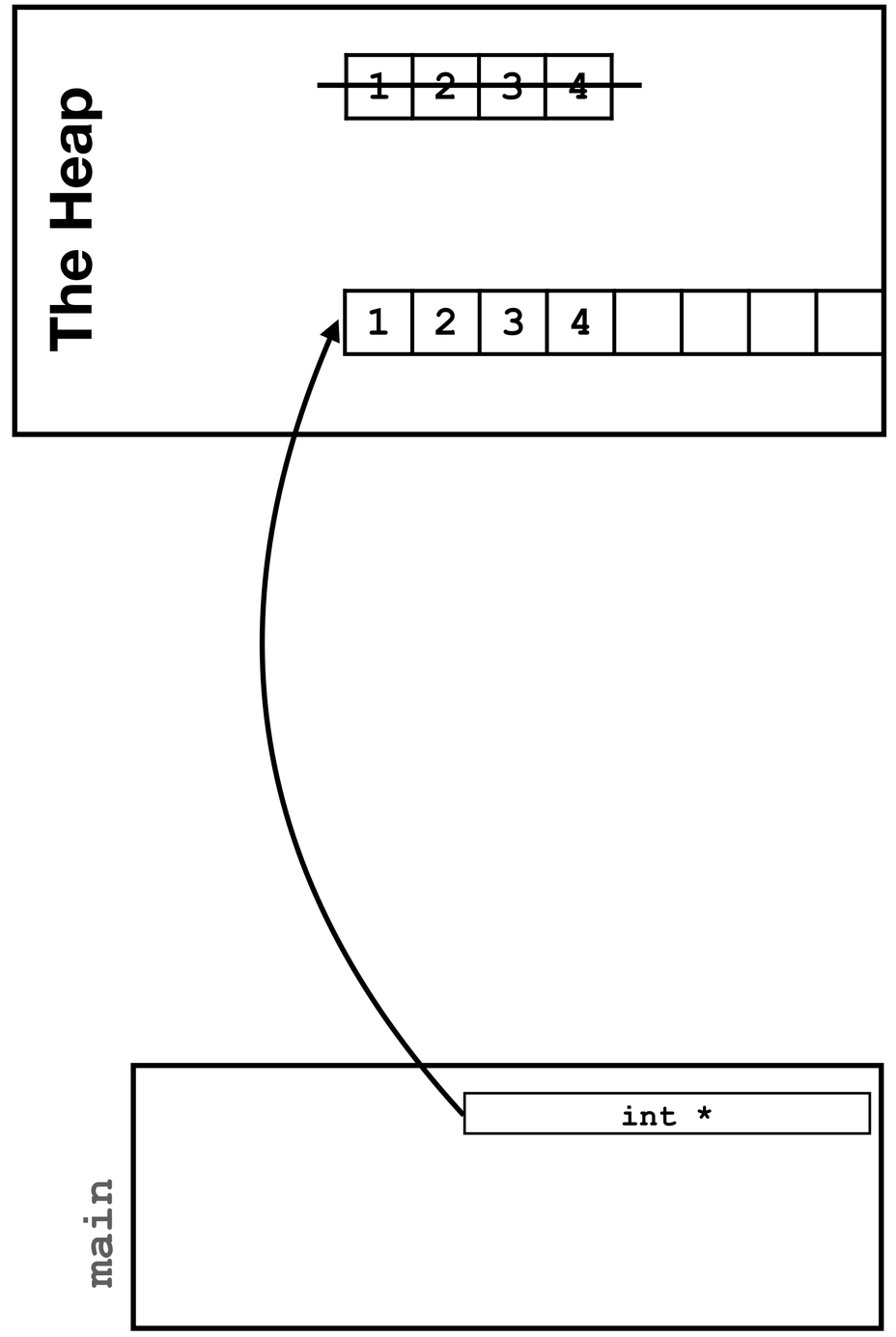
- Doing this repeatedly results in very messy code.
  - e.g. hw1
- Separation of concerns:
  - We wrap all of this up behind some interface
  - The user doesn't have to worry about any of that -- just adding and removing elements.



Array

Demo

- Let's write this together!



Understanding `void *`

`void *`

The dark side of C

- How do we usually declare a pointer?
  - `char *`
  - `int *`
  - `struct substring *`
- Why do we care what the pointer is pointing to?
  - To figure out how many bytes to read/write
- What if we never read/write to the memory location?
  - If we just want to hold onto a pointer

`void *`

The dark side of C

- void * is a *generic* pointer. Just an address; there is no information about what it points to.
- Can't dereference void * without *casting* it first.
- Can we make our array implementation generic?