Lecture 20 — Memory Management

J. Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

September 6, 2016

Acknowledgments: W.D. Bishop

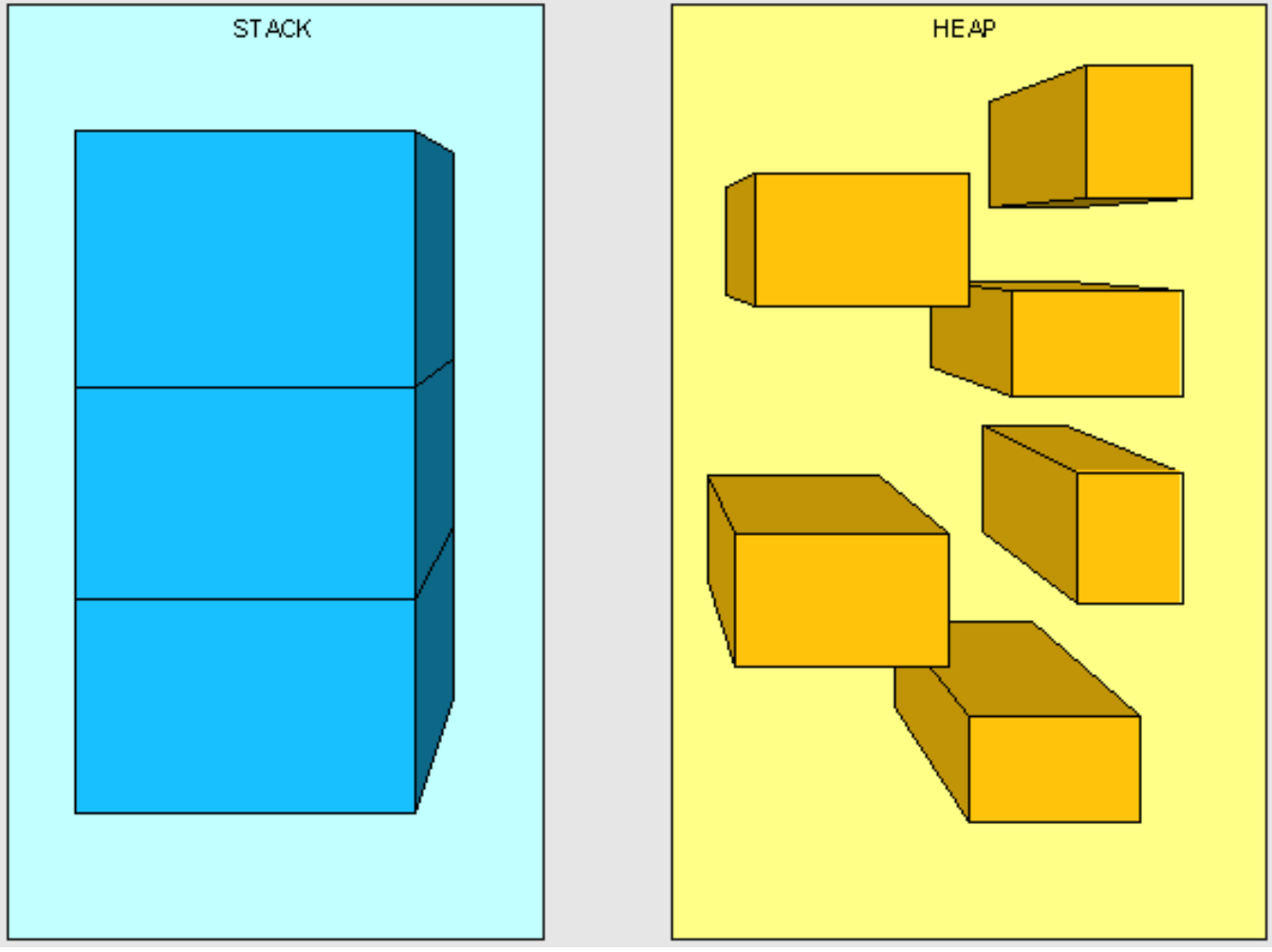
Stack and Heap

We often talk of storing a variable in memory, but this glosses over how exactly memory works, and this is a non-trivial topic.

A program has two areas of memory: the stack and heap.

Both are simply designated areas of memory related to the running program, but they are modelled and used in different ways.

Stack vs. Heap: Visually



Source: http://www.c-sharpcorner.com/UploadFile/rmcochran/csharp_memory01122006130034PM/csharp_memory.aspx?ArticleID=9adb0e3c-b3f6-40b5-98b5-413b6d348b91

The Stack

The stack keeps track of what is executing in our code.

Imagine the stack as a series of boxes stacked one on top of another.

Every time we call a function, we put another box on the pile.

This box, actually called a frame, contains the parameters, local variables, and the return value of the function.

We can only access the box currently on top of the pile.

The Stack

Value types (like `int`, `double`, and `struct`) are allocated on the stack.

Imagine we are in a method `method1()`.

A statement like `int i = 4;` allocates the variable `i` on the stack.
It is of `int` size and appears on the top of the stack.
It is then available within `method1`.

If the next statement is `int j = 0;` then another variable is allocated at the address immediately above the location for `i`.

These variables exist only as long as `method1()` is running.

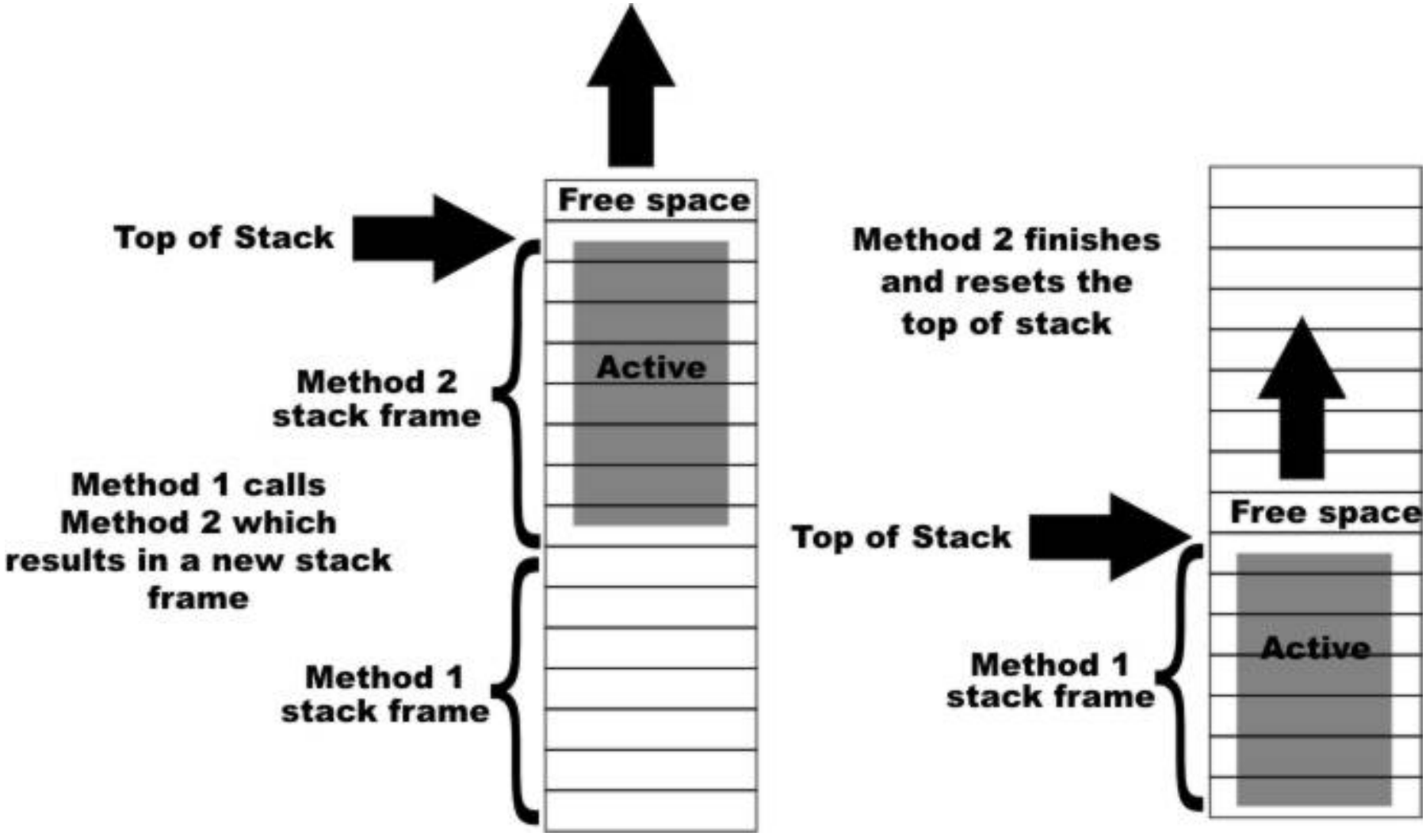
The Stack

When a function returns (finishes execution), the box on top of the stack is thrown away and we can use the stuff in the box below.

Thus, stack data is discarded aggressively (i.e., as soon as possible).

You will learn more about how the stack works and is structured in ECE 254 (Operating Systems).

Method Return



Source: `http://www.i-programmer.info/ebooks/deep-c/363`

More about the Stack

Stack space is limited in practice. Recall that recursion can result in a Stack Overflow error: this is what happens when we run out of space.

This was the result of infinite recursion: calling the function too many times (stacking too many boxes).

In normal operations, however, we will not run out of stack space.

But what if we want to allocate something large, or that should last longer than the stack of the current function?

The Heap

Memory can also be allocated on the heap.

We can allocate simple types, structs, strings, arrays, anything.

The internals of the heap are beyond the scope of the course.
But you will examine it later in ECE 250.

Discarding Objects in the Heap

We saw we don't have to de-allocate a variable allocated on the stack.
The end of the function takes care of that for us.

Heap memory, however, can outlive the functions that created them.

Heap space is also limited by the amount of memory available on the machine. This is typically much larger than the space for the stack.

Still, we need a way to discard things.

Unused objects in memory don't have any value, and they prevent other applications from using that memory.

Garbage

Recall from earlier the concept of variable scope.

If a reference to an object goes out of scope, that reference can no longer be used to access the object to which it refers.

If there are no valid references to an object, that object is no longer accessible and cannot be used for anything.

This is bad: the memory is still allocated but cannot be used.

Manual Memory Management

In languages like C++, developers also do memory deallocation:
This can easily lead to memory leaks.

A memory leak is when some area of memory remains allocated even though it is no longer needed.

Over time, a memory leak can reduce the performance of the computer by reducing the amount of available memory.

If available memory is exhausted, this may lead to a crash.

Garbage Collection

In languages like Java and C#, the approach to cleaning up garbage is called Garbage Collection.

Garbage objects are not discarded aggressively.

- They will be cleaned automatically by the garbage collector.
- The garbage collector runs when the system chooses.
- This may “waste” some memory, but can improve performance.

There is no simple way of figuring out when an object on the heap is not needed anymore, so the process of garbage collection is complex.

Garbage Collection

How does garbage collection work?

- Memory is allocated whenever an object is constructed
- Once the scope of an object ends, the object becomes “garbage”
- Programmers do not indicate when an object is no longer needed
- The memory associated with garbage objects is made available to the system when garbage collection is performed

The system decides when garbage collection is performed; the programmer does not indicate this.

Pros of Garbage Collection

Garbage collection has numerous advantages:

- Simplifies the design of complex applications
- Enhances code quality by eliminating memory leaks
- Enhances developer productivity
- Permits program to optimize memory utilization through compaction

Cons of Garbage Collection

It also has some disadvantages:

- Limits developer control over the de-allocation of memory
- Introduces overhead to monitor when objects become garbage
- Runs at unpredictable times (a problem in real-time systems)
- Often temporarily halts the program executing while it cleans up (also a problem in real-time systems)

Memory Management in C++

C++ does not keep track of allocated memory; it is the programmer's responsibility.

Objects are allocated in the first available memory address where an appropriately-sized block of memory is found.

Example: the first address may have space for an `int[]` array of capacity 10, but the request is to allocate an array of capacity 20.

Memory Management in C++

When memory is no longer needed, programmers are expected to indicate this.

To allocate memory, a block of memory of the size of that object will be needed. If none is found, that object cannot be allocated.

To keep track of it, we will have a reference to that memory.
We will see how these work soon.