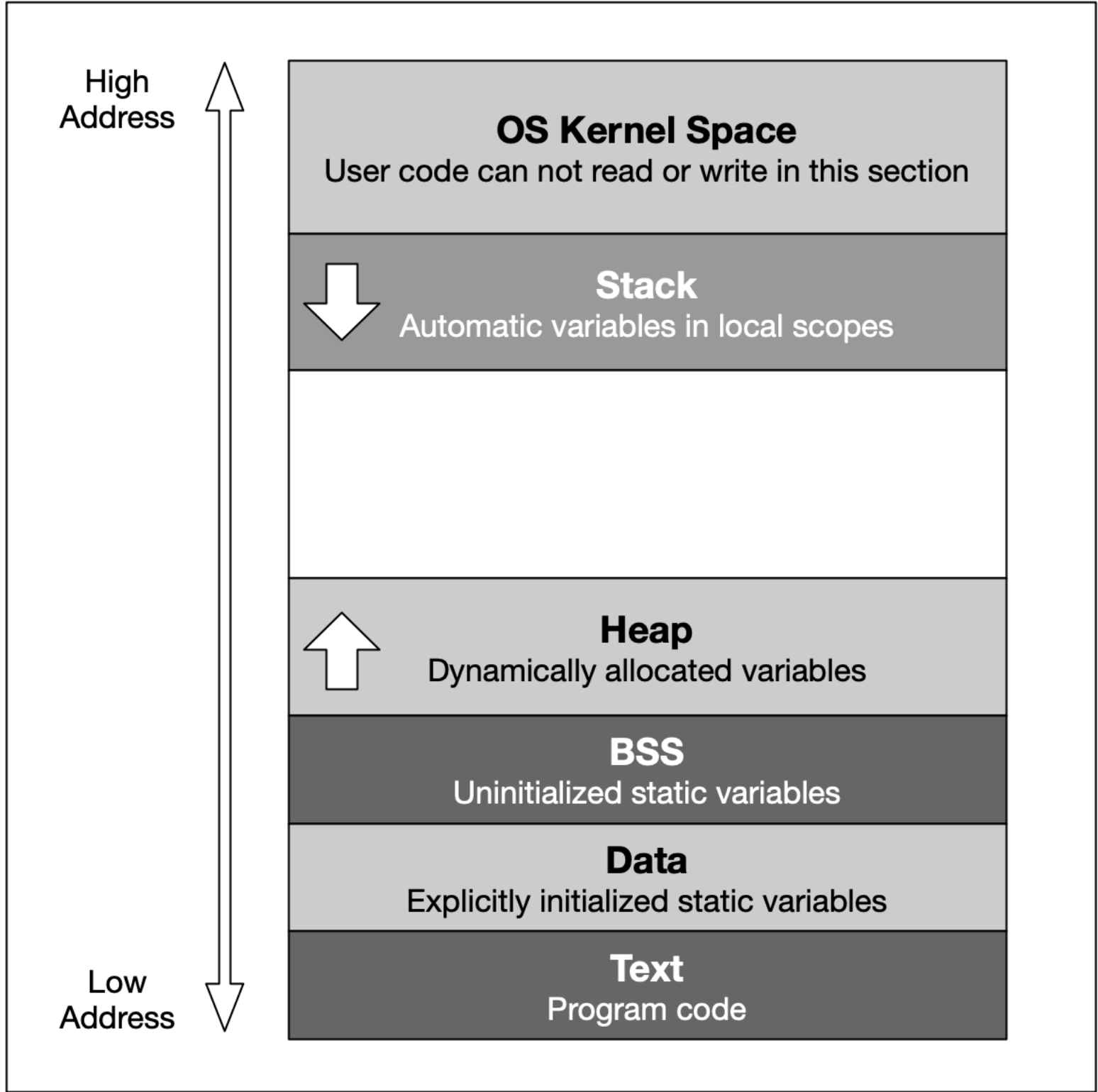


<https://youtu.be/tTx08nfDWp0>

The Process Memory Model

As we have seen in the previous lesson, each program is assigned its own virtual memory by the operating system. This address space is arranged in a linear fashion with one block of data being stored at each address. It is also divided into several distinct areas as illustrated by the figure below:



The last address `0xffffffff` converts to the decimal `4,294,967,295`, which is the total amount of memory blocks that can theoretically be addressed in a 32-bit operating system - hence the well-known limit of 4GB of memory. On a 64-bit system, the available space is significantly (!) larger. Also, the addresses are stored with 8 bytes instead of 4 bytes.

From a programming perspective, though, we are not able to use the entire address space. Instead, the blocks "OS Kernel Space" and "Text" are reserved for the operating system. In kernel space, only the most trusted code is executed - it is fully maintained by the operating system and serves as an interface between the user code and the system kernel. In this course, we will not be directly concerned with this part of memory. The section called "text" holds the program code generated by the compiler and linker. As with the kernel space, we will not be using this block directly in this course. Let us now take a look at the remaining blocks, starting from the top:

- The **stack** is a contiguous memory block with a fixed maximum size. If a program exceeds this size, it will crash. The stack is used for storing automatically allocated variables such as local variables or function parameters. If there are multiple threads in a program, then each thread has its own stack memory. New memory on the stack is allocated when the path of execution enters a scope and freed again once the scope is left. It is important to know that the stack is managed "automatically" by the compiler, which means we do not have to concern ourselves with allocation and deallocation.
- The **heap** (also called "free store" in C++) is where data with dynamic storage lives. It is shared among multiple threads in a program, which means that memory management for the heap needs to take concurrency into account. This makes memory allocations in the heap more complicated than stack allocations. In general, managing memory on the heap is more (computationally) expensive for the operating system, which makes it slower than stack memory. Contrary to the stack, the heap is not managed automatically by the system, but by the programmer. If memory is allocated on the heap, it is the programmer's responsibility to free it again when it is no longer needed. If the programmer manages the heap poorly or not at all, there will be trouble.
- The **BSS** (Block Started by Symbol) segment is used in many compilers and linkers for a segment that contains global and static variables that are initialized with zero values. This memory area is suitable, for example, for arrays that are not initialized with predefined values.
- The **Data** segment serves the same purpose as the BSS segment with the major difference being that variables in the Data segment have been initialized with a value other than zero. Memory for variables in the Data segment (and in BSS) is allocated once when a program is run and persists throughout its lifetime.