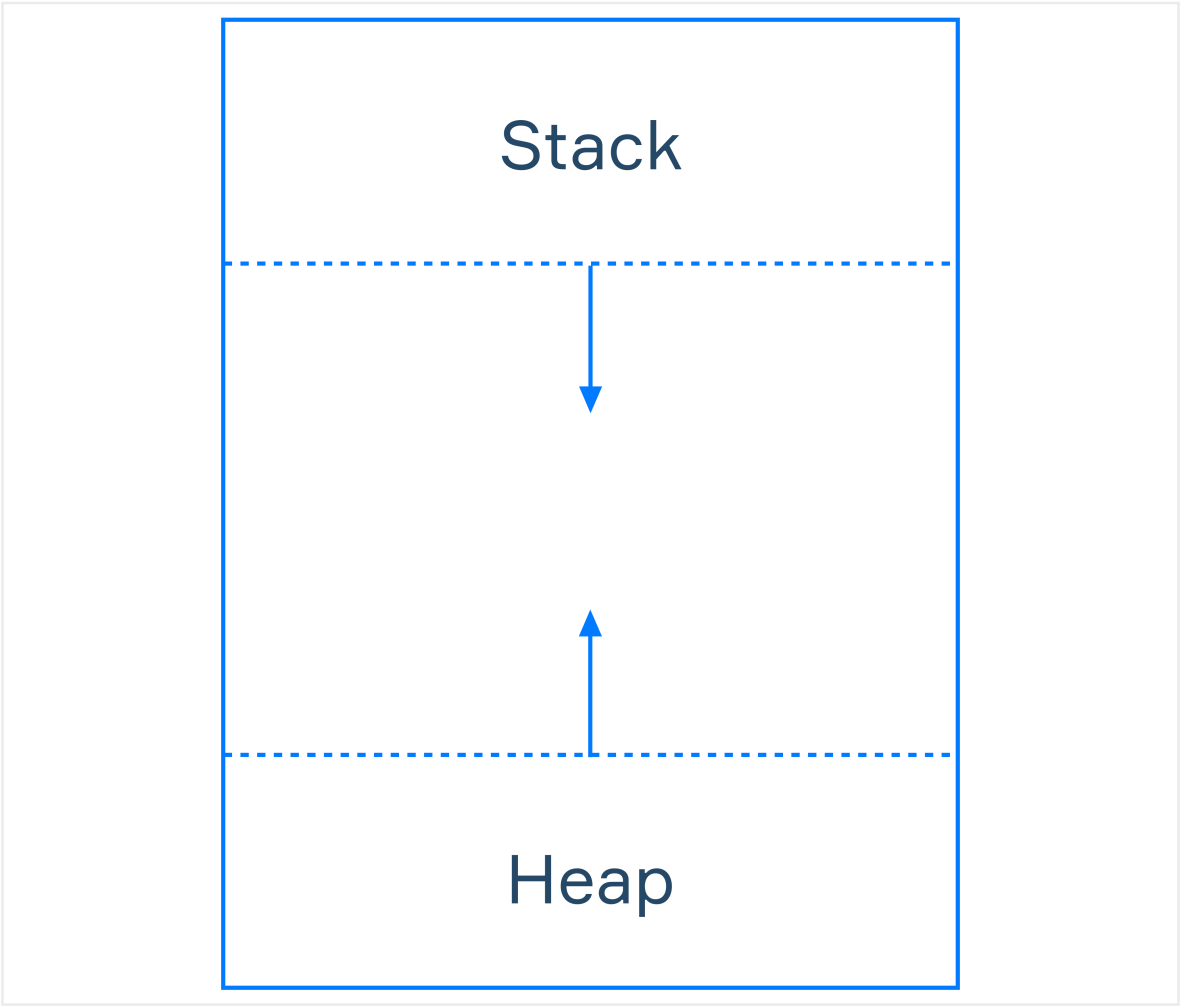
Stack and heap memory

Random Access Memory (RAM) is one of the most integral parts of a computer. This memory is used to store data that needs to be accessed quickly and temporarily by the CPU. The RAM is managed by the operating system. The operating system also makes sure that each process has enough memory and accesses only what they are allowed to access. When programs are executed, data is stored across different parts of the memory, namely the stack and the heap memory.

Stack memory

Memory allocation takes place in the function call stack, which we refer to as stack memory allocation. The memory space is allocated in **contiguous blocks**. The compiler is aware of the size of memory that needs to be allocated. Whenever a function is called, memory is allocated for its variables on the stack. The memory for the variables is also released once the function call is finished.



Several factors determine a call stack's size. Typically, it is specified at the beginning of a program. The architecture of the computer on which the

program runs, the language used to write the program, and the overall amount of memory available in the system can all affect how big the program is. When a program demands more memory than allocated, it will lead to **stack overflow**. The program or computer might crash in this situation.

Heap memory

Heap memory allocation occurs during program execution. Unlike the stack, heap memory is **dynamically allocated**. Allocation and deallocation are handled by the programmer or the **garbage collector**. Garbage collectors automatically free up memory that is no longer referenced or used by the program. Not freeing unused variables in the memory might lead to **memory leakage**. Memory leaks reduce the performance of the program by holding space that would otherwise be free for other parts of the program.

Heap memory is global. It can be accessed and modified within the program by its reference. Unlike the stack, it is not limited to the function where it is allocated. This makes heap memory allocation **flexible** but **not thread-safe** as anyone with the reference can read and write to the data.

Heap memory allocation is significantly slower than stack memory allocation and causes **memory fragmentation**, since the allocation is not contiguous. But it provides dynamic memory allocation which is very helpful in many programs because it allows resizing after program execution

Stack vs heap

Stack has high access speed and the space is managed by the operating system. Whereas heap is slow and is managed by the garbage collector or the programmer. Another key difference is that stack stores local variables while heap has the ability to store global variables.

Memory allocation in the stack is contiguous and fixed at compile time and you can not resize once the program starts. The main issue here is running out of space and being constrained. But in the heap memory, size is not fixed. Dynamic allocation gives you the flexibility to resize after the program starts. But the limitation here is that dynamic allocation leads to memory fragmentation.