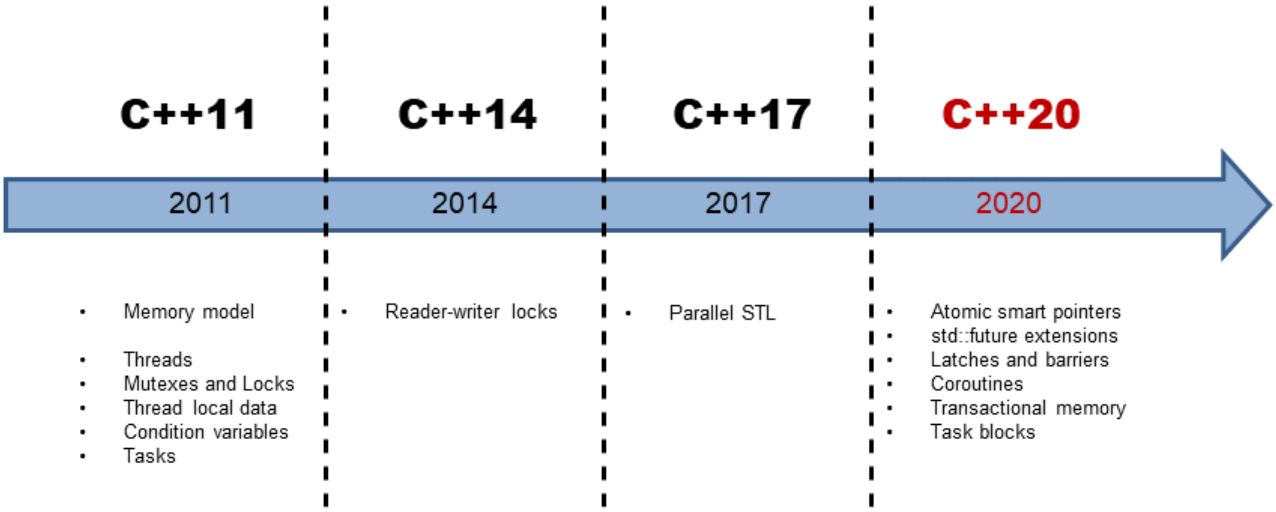
C++11 and C++14: The Foundation

This lesson demonstrates the foundation and overview of concurrency in C++11 and C++14.

WE'LL COVER THE FOLLOWING

- C++11 and C++14: The foundation
- Memory Model
- Atomics



With the publishing of the C++11 standard, C++ got a multithreading library and a memory model. This library has basic building blocks like atomic variables, threads, locks, and condition variables; that's the foundation on which upcoming C++ standards such as C++17 and C++20 can establish higher abstractions. However, C++11 already knows tasks that provide a higher abstraction than the cited basic building blocks.

Roughly speaking, you can divide the concurrency story of C++ into three evolution steps.

C++11 and C++14: The foundation

C++11 and C++14: The foundation

Multithreading was introduced in C++11. This support consists of two parts: A *well-defined* memory model, and a standardized threading interface. C++14 added reader-writer locks to the multithreading facilities of C++.

Memory Model

The foundation of multithreading is a *well-defined* memory model. This memory model has to deal with the following aspects:

- Atomic operations: operations that can be performed without interruption.
- Partial ordering of operations: the sequence of operations that must not be reordered.
- Visible effects of operations: guarantees when operations on shared variables are visible in other threads.

The C++ memory model was inspired by its predecessor: the Java memory model. Unlike the Java memory model, however, C++ allows us to break the constraints of sequential consistency, which is the default behavior of atomic operations. Sequential consistency provides two guarantees.

1. The instructions of a program are executed in source code order
2. There is a global order for all operations on all threads

The memory model is based on atomic operations on atomic data types (short atomics).

Atomics

C++ has a set of simple atomic data types: booleans, characters, numbers, and pointers in many variants. You can define your own atomic data type with the class template `std::atomic`. Atomics establish synchronization and ordering constraints that can also hold for non-atomic types. The standardized threading interface is the core of concurrency in C++.