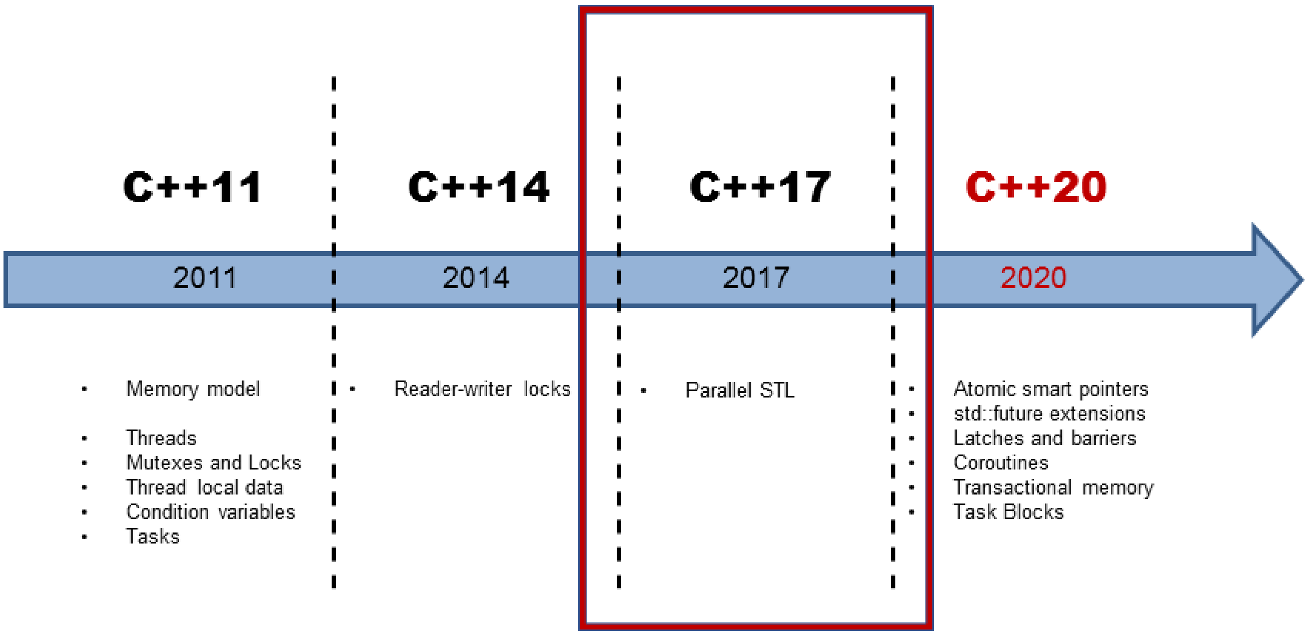
Execution Policies

This lesson explains execution policies (introduced in C++ 17) in detail.

The Standard Template Library has more than 100 algorithms for searching, counting, and manipulating ranges and their elements. With C++17, 69 of them are overloaded and 8 new ones are added. The overloaded and new algorithms can be invoked with a so-called execution policy.



By using an execution policy, you can specify whether the algorithm should run sequentially, in parallel, or in parallel with vectorization.

Execution Policies

The policy tag specifies whether a program should run sequentially, in parallel, or in parallel with vectorization.

- `std::parallel::seq`: runs the program sequentially
- `std::parallel::par`: runs the program in parallel on multiple threads
- `std::parallel::par_unseq`: runs the program in parallel on multiple threads and allows the interleaving of individual loops; this permits a vectorized version with SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) extensions.

The following code snippet shows all execution policies:

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9};

// standard sequential sort
std::sort(v.begin(), v.end());

// sequential execution
std::sort(std::parallel::seq, v.begin(), v.end());

// permitting parallel execution
std::sort(std::parallel::par, v.begin(), v.end());

// permitting parallel and vectorised execution
std::sort(std::parallel::par_unseq, v.begin(), v.end());
```

The example shows that you can still use the classic variant of `std::sort` (line 4). Also, in C++17 you can specify explicitly whether the sequential (line 7), parallel (line 10), or the parallel and vectorized (line 13) version should be used. In the next lesson, I will discuss parallel and vectorized execution in more detail.