async: Fire and Forget

This lesson gives an overview of fire and forget, which are used with std::async in C++ for concurrency.

Fire and forget futures are special futures; they execute in place because their future is not bound to a variable. For a “fire and forget” future, it is necessary that the promise runs in a separate thread so it can immediately start its work. This is done by the `std::launch::async` policy.

Let’s compare an ordinary future with a fire and forget future.

```
#include<iostream>
#include<future>

int main(){

auto fut= std::async([]{ return 2011; });
std::cout << fut.get() << std::endl;

std::async(std::launch::async,
           []{ std::cout << "fire and forget" << std::endl; });
}
```

Fire and forget futures look very promising but have a big drawback. A future that is created by `std::async` waits for its destructor, until its promise is done. In this context, waiting is not very different from blocking. The future blocks the progress of the program in its destructor. This becomes more obvious when we use fire and forget futures. What seems to be concurrent actually runs sequentially.

```
// fireAndForgetFutures.cpp

#include <chrono>
#include <future>
#include <iostream>
#include <thread>

int main(){
```

```
int main(){

  std::cout << std::endl;

  std::async(std::launch::async, []{
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::cout << "first thread" << std::endl;
    });

  std::async(std::launch::async, []{
    std::this_thread::sleep_for(std::chrono::seconds(1));
    std::cout << "second thread" << std::endl;}
    );

  std::cout << "main thread" << std::endl;

  std::cout << std::endl;

}
```

The program executes two promises, each in their own threads. The resulting futures are fire and forget futures. These futures block their destructors in until the associated promise is done. The result is that the promises will be executed in sequence, which we find in the source code. That being said, the execution sequence is independent of the execution time; this is exactly what we see in the output of the program.

`std::async` is a convenient mechanism used to distribute a bigger compute job on more shoulders.

In the next lesson, we'll solve an exercise related to concurrent calculation with `std::async` in C++.