

# std::scoped\_lock

Let's look at the how C++17 allows locking a variadic number of mutexes simultaneously.

## WE'LL COVER THE FOLLOWING



- Previous functionality: `lock_guard`
- C++17's `scoped_lock`

## Previous functionality: `lock_guard` #

With C++11 and C++14 we got the threading library and many support functionalities.

For example, with `std::lock_guard` you can take ownership of a mutex and lock it in RAII style:

```
std::mutex m;  
  
std::lock_guard<std::mutex> lock_one(m);  
// unlocked when lock_one goes out of scope...
```

The above code works, however, **only for a single mutex**.

If you wanted to lock several mutexes, you had to use a different pattern, for example:

```
std::mutex first_mutex;  
std::mutex second_mutex;  
  
// ...  
  
std::lock(first_mutex, second_mutex);  
std::lock_guard<std::mutex> lock_one(first_mutex, std::adopt_lock);  
std::lock_guard<std::mutex> lock_two(second_mutex, std::adopt_lock);  
// ..
```

# C++17's `scoped_lock` #

With C++17 things get a bit easier as with `std::scoped_lock` you can lock a variadic number of mutexes at the same time.

```
std::scoped_lock lock(first_mutex, second_mutex);
```

*Due to compatibility `std::lock_guard` couldn't be extended with a variadic number of input mutexes and that's why a new type - `scoped_lock` - was needed.*

**Extra Info:** You can read more information in [P0156](#)

---

Coming up next: C++17's enhancement to the standard allocator from the Standard Library.