

# Adding Thread Safety with `synchronized_value`



Proposed by Anthony  
Williams in 2014

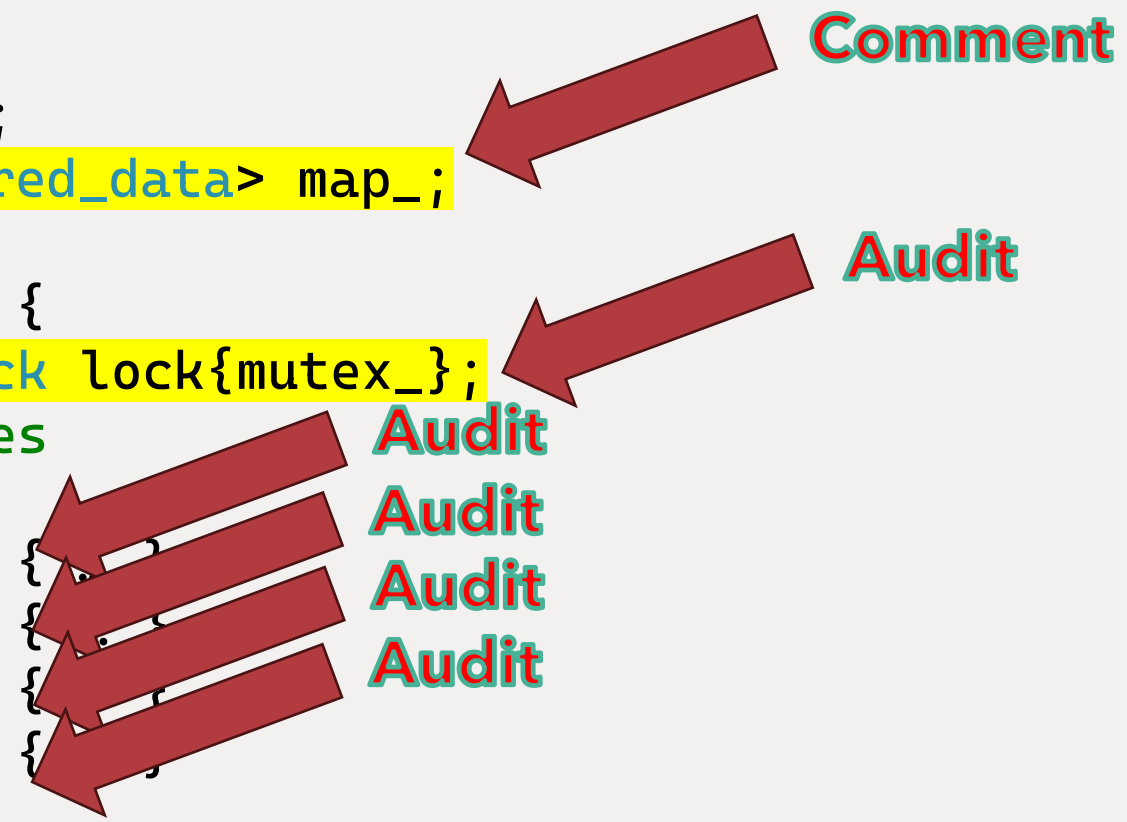
Proposals: N4033, P0290r4

```
class legacy_class
{
    std::map<int, shared_data> map_;
public:
    auto function_a() {
        // ... 100 lines
    }
    auto function_b() {
        // ... 50 lines
    }
    auto function_c() { ... }
    auto function_d() { ... }
    auto function_e() { ... }
};
```

*Solution 1: std::mutex*

```
class legacy_class1
{
    std::mutex mutex_;
    std::map<int, shared_data> map_;
public:
    auto function_a() {
        std::unique_lock lock{mutex_};
        // ... 100 lines
    }
    auto function_b() { ... }
    auto function_c() { ... }
    auto function_d() { ... }
    auto function_e() { ... }
};
```

```
class legacy_class1
{
    std::mutex mutex_;
    std::map<int, shared_data> map_;
public:
    auto function_a() {
        std::unique_lock lock{mutex_};
        // ... 100 lines
    }
    auto function_b() {
    auto function_c() {
    auto function_d() {
    auto function_e() {
};
```



The diagram illustrates audit points in the provided C++ code. Red arrows point to specific lines, each labeled with a red-outlined text label:

- An arrow points to the line `std::map<int, shared_data> map_;` with the label **Comment**.
- An arrow points to the line `std::unique_lock lock{mutex_};` with the label **Audit**.
- Four arrows point to the opening curly braces of the functions `function_b()`, `function_c()`, `function_d()`, and `function_e()`, each with the label **Audit**.

# *Solution 2: Wrapping the Data*

```
template <typename Key, typename Value>
class synchronized_map
{
    std::mutex mutex_;
    std::map<Key, Value> map_;
public:
    auto& operator[](Key k) {
        std::unique_lock lock{mutex_};
        return map_[k];
    }
    // ...
};
```



```
template <typename Key, typename Value>
class synchronized_map
{
    std::mutex mutex_;
    std::map<Key, Value> map_;
public:
    auto& operator[](Key k) {
        std::unique_lock lock{mutex_};
        return map_[k];
    }
    // Delegate all methods:
    // at()
    // begin()
    // end()
    // cbegin()
    // cend()
    // ...
};
```

*Solution 3: synchronized\_value*

```
class legacy_class3
{
    synchronized_value<std::map<int, shared_data>> map_;
public:
    auto function_a() { ... }
    auto function_b() { ... }
    auto function_c() { ... }
    auto function_d() { ... }
    auto function_e() { ... }
};
```

```
class legacy_class3
{
    synchronized_value<std::map<int, shared_data>> map_;
public:
    auto function_a() {
        // ... 30 lines
        auto value0 = map_->at(13);
        auto value1 = map_->at(37);
        // ...
    }
    // ...
};
```

```
class legacy_class3
{
    synchronized_value<std::map<int, shared_data>> map_;
public:
    auto function_a() {
        // ... 30 lines
        {
            update_guard<std::map<int, shared_data>> guard{map_};
            auto value0 = guard->at(13);
            auto value1 = guard->at(37);
        }
        // ...
    }
    // ...
};
```

*[https://github.com/jrgfogh/synchronized\\_value](https://github.com/jrgfogh/synchronized_value)*

