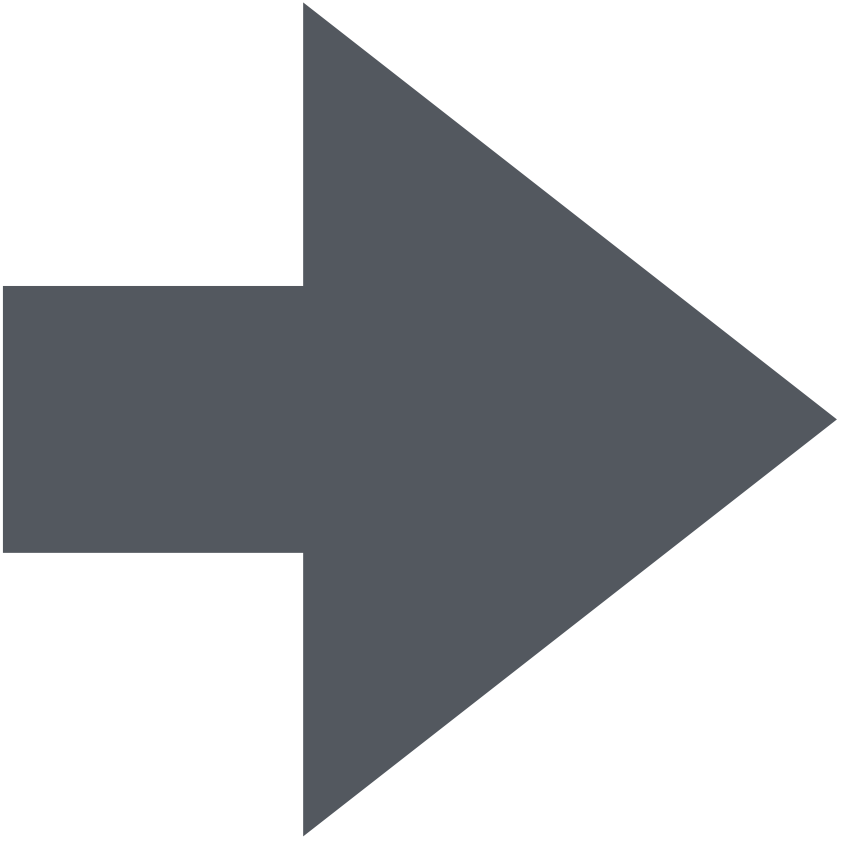
Implicit `synchronized_value`

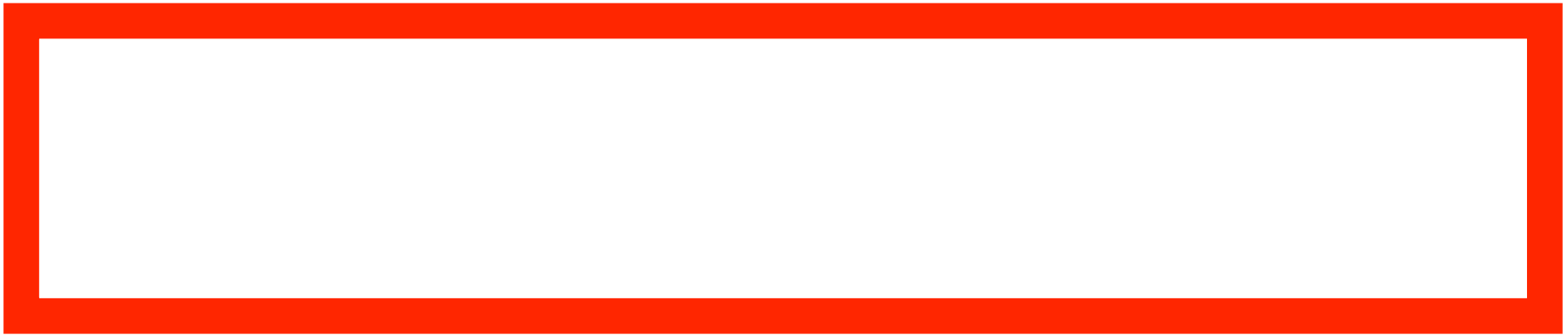
63

```
class(synchronized) person
{
public:
    person() = default;

    std::string get_first_name() const
    {
        return first_name;
    }

    void set_first_name (std::string_view new_first)
    {
        first_name = new_first;
    }

    // Repeat for last_name

private:
    std::string first_name, last_name;
};
```

```
class person
{
public:
    person() = default;

    std::string get_first_name() const
    {
        return apply ([] (auto& p) {
                          return p.get_first_name();
                      },
                      person_internal);
    }

    void set_first_name (std::string_view new_first)
    {
        apply ([&] (auto& p) {
                       p.set_first_name (new_first);
                   },
                   person_internal);
    }

    // Repeat for last_name

private:
    struct __person;
    mutable synchronized_value<__person> person_;
};

template<>
struct is_sync<person> : std::true_type {};
```



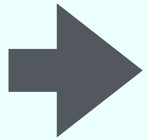
metaclass proposed syntax



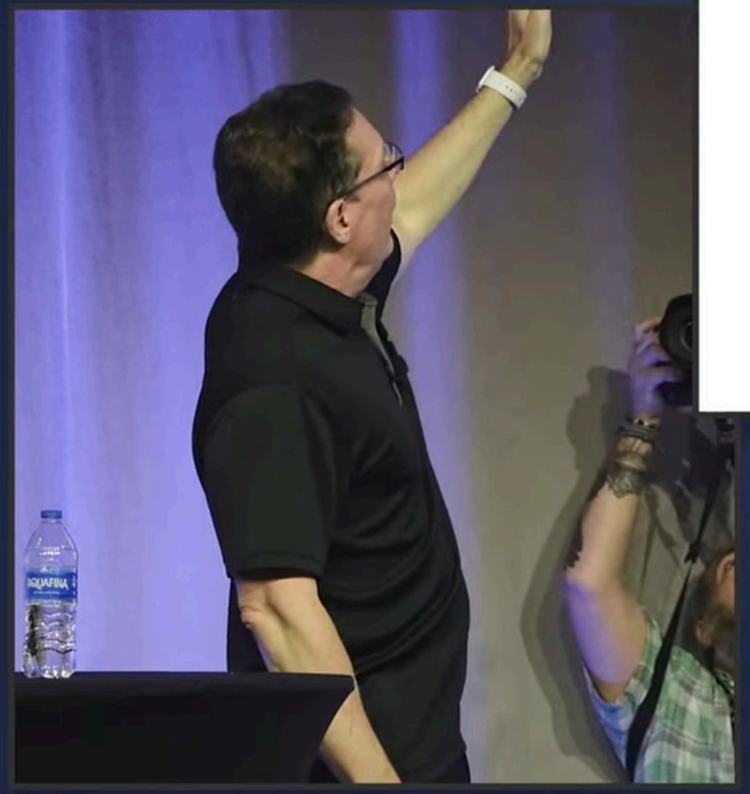
Implicit `synchronized_value`

metaclass proposed syntax

```
class(synchronized) person
{
public:
    person() = default;

    std::string get_first_name() const
    {
        return first_name;
    }

    void set_first_name (std::string_view new_first)
    {
        first_name = new_first;
    }

    // Repeat for last_name

private:
    std::string first_name, last_name;
};
```



```
class person
{
public:
    person() = default;

    std::string get_first_name() const
    {
       return apply ([] (auto& p) {
                          return p.get_first_name();
                      },
                      person_internal);
    }

    void set_first_name (std::string_view new_first)
    {
        apply ([&] (auto& p) {
                    p.set_first_name (new_first);
                },
                person_internal);
    }

    // Repeat for last_name

private:
    struct __person;
    mutable synchronized_value<__person> person_;
};

template<>
struct is_sync<person> : std::true_type {};
```



Now in EDG… *godbolt.org/z/fex55qq5o*

```
consteval auto make_interface_functions(info proto) -> info {
    info ret = ^^{};
    for (info mem : members_of(proto)) {
        if (is_nonspecial_member_function(mem)) {
            ret = ^^{
                \tokens(ret)
                virtual [:\(return_type_of(mem)):]
                    \id(identifier_of(mem)) (\tokens(parameter_list_of(mem))) = 0;
            };
        }
        // --- reporting compile time errors not yet implemented ---
        // else if (is_variable(mem)) {
        //   print
        // }  // e
    }
    return ret;
}
```

```
consteval void interface(std::meta::info proto) {
    std::string_view name = identifier_of(proto);
    queue_injection(^^{
        class \id(name) {
        public:
            \tokens(make_interface_functions(proto))
            virtual ~\id(name)() { }
        };
    });
}
```