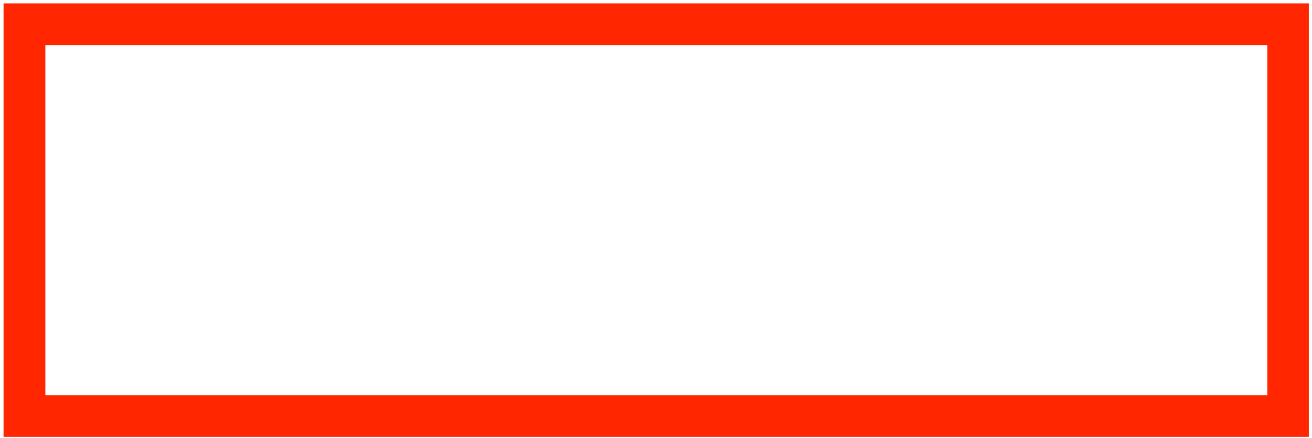
93

```
template<sync T>
struct is_send<std::shared_ptr<T>> : std::true_type
{};
```

```
template <typename T>
struct is_send : std::integral_constant<
                      bool,
                      (! (std::is_lvalue_reference_v<T>
                          || std::is_pointer_v<std::remove_extent_t<T>>
                          || is_lambda_v<T>))
                      &&
                      (std::is_move_constructible_v<T>
                       || (is_function_pointer_v<std::decay_t<T>>
                           && ! std::is_member_function_pointer_v<T>)
                       || is_sync_v<T>)>
{};
```



```
template <typename T>
struct is_send : std::integral_constant<
                     bool,
                     (! (std::is_lvalue_reference_v<T>
                         || std::is_pointer_v<std::remove_extent_t<T>>
                         || is_lambda_v<T>))
                     &&
                     (std::is_move_constructible_v<T>
                      || (is_function_pointer_v<std::decay_t<T>>
                          && ! std::is_member_function_pointer_v<T>)
                      || is_sync_v<T>)>
{};
```

```
template<sync T>
struct is_send<std::shared_ptr<T>> : std::true_type
{};
```

```cpp
void entry_point (std::shared_ptr<synchronized_value<std::string>> sync_s, int tid)
{
    apply ([tid] (auto& s) {
        s.append ("🔥");
        std::println ("{} {}", s, tid);
        return s;
    },
    *sync_s);
}

int main()
{
    auto s = std::make_shared<synchronized_value<std::string>> ("Hello threads");

    std::vector<safe_thread> threads { };

    const int num_threads = 15;

    for (int i : std::views::iota (0, num_threads))
        threads.push_back (safe_thread (entry_point, auto (s), auto (i)));
}
```