C++26 из Сент-Луис

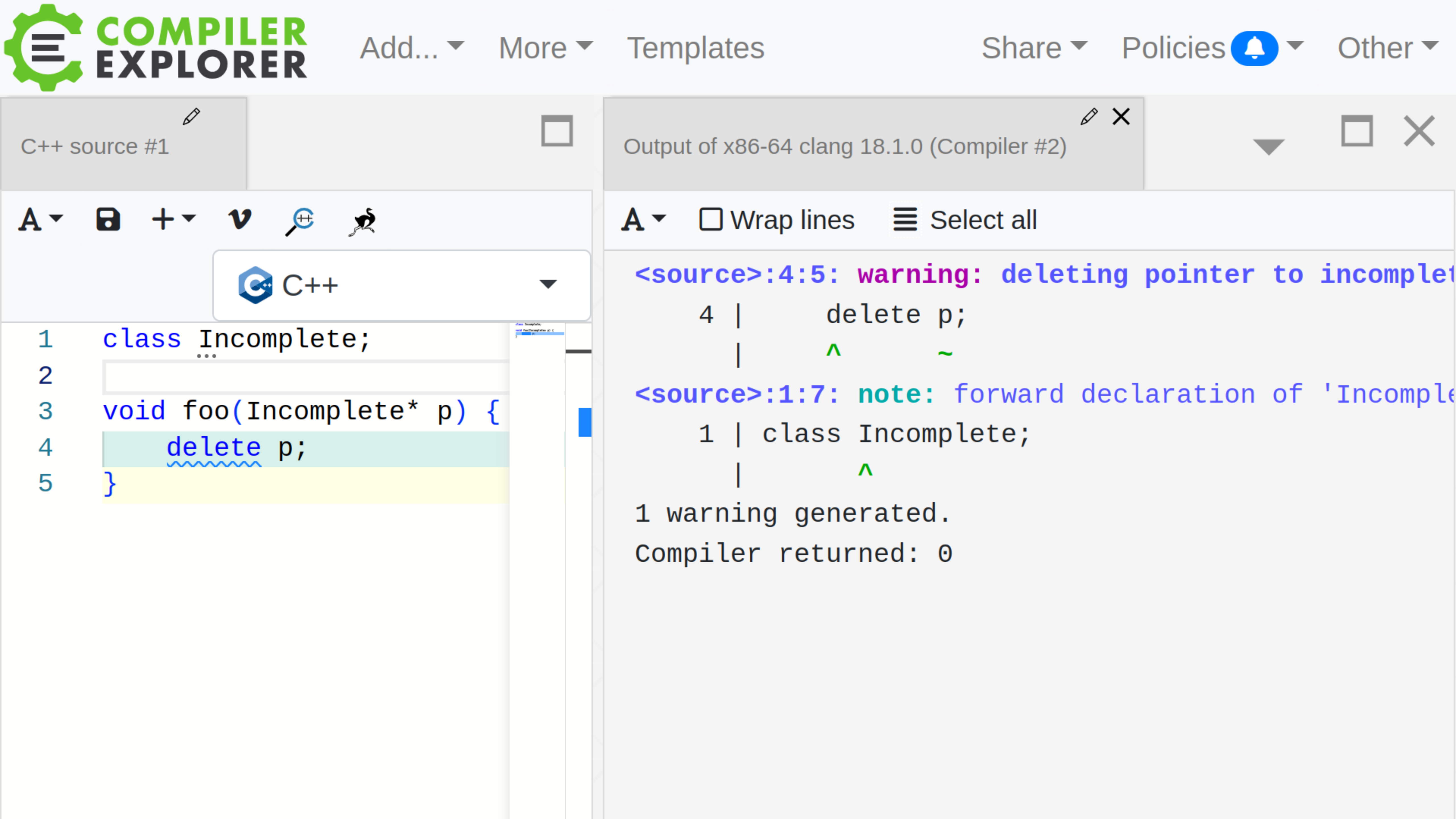
Полухин Антон

Руководитель группы Общих Компонент,

Эксперт-разработчик C++

Содержание

delete Incomplete;



```cpp
class Incomplete;

void foo(Incomplete* p) {
    delete p;
}
```

```
<source>:4:5: warning: deleting pointer to incomplet
    4 |     delete p;
      |     ^      ~
<source>:1:7: note: forward declaration of 'Incomple
    1 | class Incomplete;
      |       ^
1 warning generated.
Compiler returned: 0
```

if (auto [a, b] = foo())

if (auto [a, b] = foo())

```
if (auto [to, ec] = std::to_chars(p, last, 42))
```

if (auto [a, b] = foo())

```
if (auto [to, ec] = std::to_chars(p, last, 42)) // to_chars_result::operator bool()
```

if (auto [a, b] = foo())

```
if (auto [to, ec] = std::to_chars(p, last, 42))
{
    auto s = std::string_view(p, to);
    assert(s == "42");
    // ...
}
```

std::optional::begin

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```cpp
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```cpp
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::optional::begin

```
struct Phones {
    /* ... */
    std::optional<std::string> get_vendor_optional() const;
};

auto phone_vendors = phones
  | std::views::transform(&Phone::get_vendor_optional)
  | std::views::join
  | std::ranges::to<std::unordered_set>()
;
```

std::inplace_vector

std::inplace_vector

```
std::inplace_vector<int, 1024> integers;
integers.push_back(42);
```

std::inplace_vector

```
template<class... Args>
constexpr pointer try_emplace_back(Args&&... args);

constexpr pointer try_push_back(const T& x);

constexpr pointer try_push_back(T&& x);

template<container-compatible-range<T> R>
constexpr ranges::borrowed_iterator_t<R> try_append_range(R&& rg);
```

std::inplace_vector

```
template<class... Args>
constexpr reference unchecked_emplace_back(Args&&... args);

constexpr reference unchecked_push_back(const T& x);

constexpr reference unchecked_push_back(T&& x);
```

std::print

std::print

```cpp
template<> inline constexpr bool
enable_nonlocking_formatter_optimization<T> = true;
```

std::print

```
template<> inline constexpr bool
enable_nonlocking_formatter_optimization<T> = true;

template<class... Args>
void print(FILE* stream, format_string<Args...> fmt, Args&&... args);
```

Philox

Philox

- Метод Монте-Карло

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

Philox

```cpp
uint32_t global_seed = 999;
for(uint32_t time_step = 0; time_step < time_steps_num; ++time_step){
  for(uint32_t atom_id = 0; atom_id < atoms_num; ++atom_id){
    std::philox4x32 eng(global_seed);
    eng.set_counter({atom_id, time_step, 0, 0});
    std::normal_distribution nd;
    auto n1 = nd(eng);
    auto n2 = nd(eng);
    // ...
  }
}
```

std::execution

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                  // 1

auto begin = schedule(sch);                                          // 2
auto hi = then(begin, []{                                            // 3
    std::cout << "Hello world! Have an int.";                        // 3
    return 13;                                                       // 3
});                                                                  // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });            // 4

auto [i] = this_thread::sync_wait(add_42).value();                   // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                 // 1

auto begin = schedule(sch);                                         // 2
auto hi = then(begin, []{                                           // 3
    std::cout << "Hello world! Have an int.";                       // 3
    return 13;                                                      // 3
});                                                                 // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });           // 4

auto [i] = this_thread::sync_wait(add_42).value();                  // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                // 1

auto begin = schedule(sch);                                        // 2
auto hi = then(begin, []{                                          // 3
    std::cout << "Hello world! Have an int.";                      // 3
    return 13;                                                     // 3
});                                                                // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });          // 4

auto [i] = this_thread::sync_wait(add_42).value();                 // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                  // 1

auto begin = schedule(sch);                                          // 2
auto hi = then(begin, []{                                            // 3
    std::cout << "Hello world! Have an int.";                        // 3
    return 13;                                                       // 3
});                                                                  // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });            // 4

auto [i] = this_thread::sync_wait(add_42).value();                   // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                     // 1

auto begin = schedule(sch);                                             // 2
auto hi = then(begin, []{                                               // 3
    std::cout << "Hello world! Have an int.";                           // 3
    return 13;                                                          // 3
});                                                                     // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });               // 4

auto [i] = this_thread::sync_wait(add_42).value();                      // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                        // 1

auto begin = schedule(sch);                                                // 2
auto hi = then(begin, []{                                                  // 3
    std::cout << "Hello world! Have an int.";                              // 3
    return 13;                                                             // 3
});                                                                        // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });                  // 4

auto [i] = this_thread::sync_wait(add_42).value();                         // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                     // 1

auto begin = schedule(sch);                                             // 2
auto hi = then(begin, []{                                               // 3
    std::cout << "Hello world! Have an int.";                           // 3
    return 13;                                                          // 3
});                                                                     // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });               // 4

auto [i] = this_thread::sync_wait(add_42).value();                      // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                    // 1

auto begin = schedule(sch);                                            // 2
auto hi = then(begin, []{                                              // 3
    std::cout << "Hello world! Have an int.";                          // 3
    return 13;                                                         // 3
});                                                                    // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });              // 4

auto [i] = this_thread::sync_wait(add_42).value();                     // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                  // 1

auto begin = schedule(sch);                                          // 2
auto hi = then(begin, []{                                            // 3
    std::cout << "Hello world! Have an int.";                        // 3
    return 13;                                                       // 3
});                                                                  // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });            // 4

auto [i] = this_thread::sync_wait(add_42).value();                   // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                  // 1

auto begin = schedule(sch);                                          // 2
auto hi = then(begin, []{                                            // 3
    std::cout << "Hello world! Have an int.";                        // 3
    return 13;                                                       // 3
});                                                                  // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });            // 4

auto [i] = this_thread::sync_wait(add_42).value();                   // 5
```

std::execution

```
using namespace std::execution;

auto sch = thread_pool.scheduler();                                    // 1

auto begin = schedule(sch);                                            // 2
auto hi = then(begin, []{                                              // 3
    std::cout << "Hello world! Have an int.";                          // 3
    return 13;                                                         // 3
});                                                                    // 3
auto add_42 = then(hi, [](int arg) { return arg + 42; });              // 4

auto [i] = this_thread::sync_wait(add_42).value();                     // 5
```

std::execution

```
using namespace std::execution;

scheduler auto sch = thread_pool.scheduler();                        // 1

sender auto begin = schedule(sch);                                   // 2
sender auto hi = then(begin, []{                                     // 3
    std::cout << "Hello world! Have an int.";                        // 3
    return 13;                                                       // 3
});                                                                  // 3
sender auto add_42 = then(hi, [](int arg) { return arg + 42; });     // 4

auto [i] = this_thread::sync_wait(add_42).value();                   // 5
```

std::execution

```
using namespace std::execution;

scheduler auto sch = thread_pool.scheduler();                     // 1

sender auto begin = schedule(sch);                                // 2
sender auto hi = then(begin, []{                                  // 3
    std::cout << "Hello world! Have an int.";                     // 3
    return 13;                                                    // 3
});                                                               // 3
sender auto add_42 = then(hi, [](int arg) { return arg + 42; });  // 4

auto [i] = this_thread::sync_wait(add_42).value();                // 5
```

std::execution

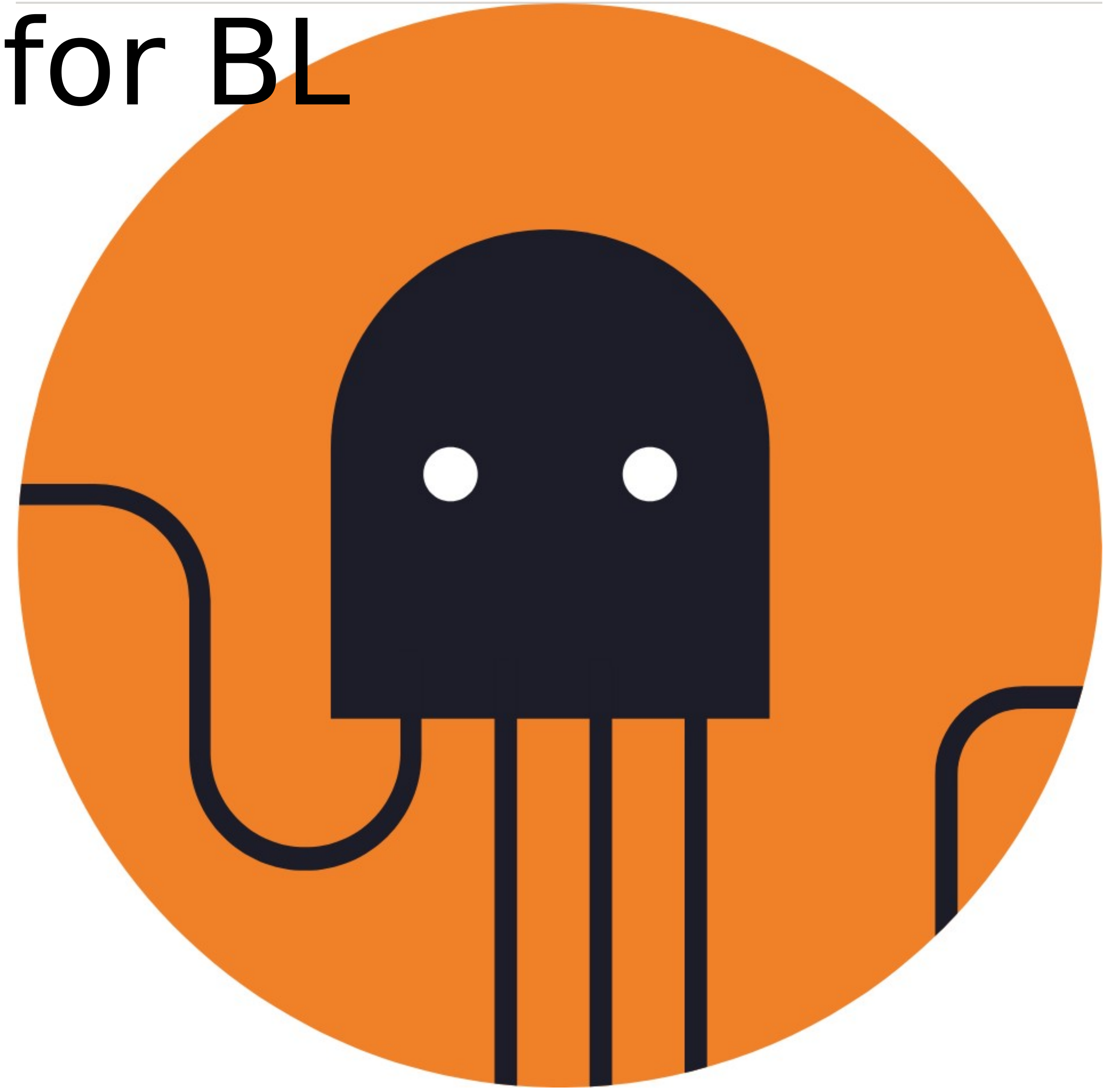
```
using namespace std::execution;

scheduler auto sch = thread_pool.scheduler();                           // 1

sender auto begin = schedule(sch);                                      // 2
sender auto hi = then(begin, []{                                        // 3
    std::cout << "Hello world! Have an int.";                           // 3
    return 13;                                                          // 3
});                                                                     // 3
sender auto add_42 = then(hi, [](int arg) { return arg + 42; });        // 4

auto [i] = this_thread::sync_wait(add_42).value();                      // 5
```

std::execution

```
sender_of<dynamic_buffer> auto async_read_array(auto handle) {
  return just(dynamic_buffer{})
       | let_value([handle] (dynamic_buffer& buf) {
           return just(std::as_writeable_bytes(std::span(&buf.size, 1)))
                | async_read(handle)
                | then(
                    [&buf] (std::size_t bytes_read) {
                      buf.data = std::make_unique<std::byte[]>(buf.size);
                      return std::span(buf.data.get(), buf.size);
                    })
                | async_read(handle)
                | then(
                    [&buf] (std::size_t bytes_read) {
                      return std::move(buf);
                    });
       });
}
```

~~std::execution~~ for BL

https://userver.tech



Ближайшее будущее

Ближайшее будущее

- Reflection

Ближайшее будущее

- Reflection
- auto [x…] = tuple;

Ближайшее будущее

- Reflection
- auto [x…] = tuple;
- Contracts

Спасибо за внимание!

Полухин Антон

Руководитель группы Общих Компонент

Эксперт-разработчик C++