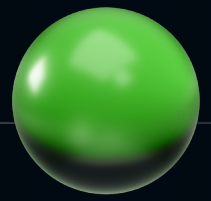
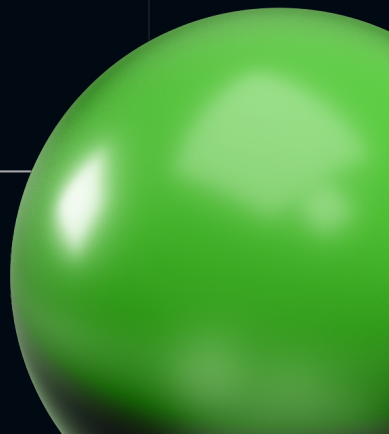
Кастомизируем ASIO

Илья Казаков
инженер-программист, YADRO





C++ Russia
2023

YADRO

Задача

- Асинхронная запись/чтение

Задача

- Асинхронная запись/чтение

- Блочные устройства (диски)

- linux

Примерный интерфейс

```
template <typename F>
void asyncRead(int fd, std::span<std::byte> buf,
   off_t offset, F callback);

template <typename F>
void asyncWrite(int fd, std::span<const std::byte> buf,
   off_t offset, F callback);
```

AsyncWrite

```
auto work = [=, callback = std::move(callback)]() mutable {
    const auto res
        = ::pwrite(fd, buf.data(), buf.size(), offset);
    auto ec = std::error_code{errno, std::system_category()};
    std::move(callback)(ec, res);
};
```

AsyncWrite

```
auto work = [=, callback = std::move(callback)]() mutable {
    const auto res
        = ::pwrite(fd, buf.data(), buf.size(), offset);
    auto ec = std::error_code{errno, std::system_category()};
    std::move(callback)(ec, res);
};
```

AsyncWrite

```
auto work = [=, callback = std::move(callback)]() mutable {
    const auto res
        = ::pwrite(fd, buf.data(), buf.size(), offset);
    auto ec = std::error_code{errno, std::system_category()};
    std::move(callback)(ec, res);
};
```

AsyncWrite

```
template <typename F>
void asyncWrite(int fd, std::span<const std::byte> buf,
   off_t offset, F callback) {
   auto work = /* ... */

   std::thread{std::move(work)}.detach();
}
```

Наивный подход

- Создание потока – медленно
- Может привести к потоковому голоданию

Неблокирующие вызовы: open и O_NONBLOCK

O_NONBLOCK or O_NDELAY

When possible, the file is opened in nonblocking mode.Neither the **open**() nor any subsequent I/O operations on the file descriptor which is returned will cause the calling process to wait.

Note that the setting of this flag has no effect on the operation of poll(2), select(2), epoll(7), and similar, since those interfaces merely inform the caller about whether a file descriptor is "ready", meaning that an I/O operation performed on the file descriptor with the O_NONBLOCK flag *clear* would not block.

Note that this flag has no effect for regular files and block devices; that is, I/O operations will (briefly) block when device activity is required, regardless of whether O_NONBLOCK is set. Since O_NONBLOCK semantics might eventually be implemented, applications should not depend upon blocking behavior when specifying this flag for regular files and block devices.

For the handling of FIFOs (named pipes), see also fifo(7). For a discussion of the effect of O_NONBLOCK in conjunction with mandatory file locks and with file leases, see fcntl(2).

Неблокирующие вызовы: open и O_NONBLOCK

O_NONBLOCK or O_NDELAY

When possible, the file is opened in nonblocking mode.Neither the **open**() nor any subsequent I/O operations on the file descriptor which is returned will cause the calling process to wait.

Note that the setting of this flag has no effect on the operation of poll(2), select(2), epoll(7), and similar, since those interfaces merely inform the caller about whether a file descriptor is "ready", meaning that an I/O operation performed on the file descriptor with the O_NONBLOCK flag *clear* would not block.

Note that this flag has no effect for regular files and block devices; that is, I/O operations will (briefly) block when device activity is required, regardless of whether O_NONBLOCK is set. Since O_NONBLOCK semantics might eventually be implemented, applications should not depend upon blocking behavior when specifying this flag for regular files and block devices.

For the handling of FIFOs (named pipes), see also fifo(7). For a discussion of the effect of O_NONBLOCK in conjunction with mandatory file locks and with file leases, see fcntl(2).

Неблокирующие вызовы: pwritev2 и RWF_NOWAIT

RWF_NOWAIT (since Linux 4.14)

Do not wait for data which is not immediately available. If this flag is specified, the **preadv2()** system call will return instantly if it would have to read data from the backing storage or wait for a lock. If some data was successfully read, it will return the number of bytes read. If no bytes were read, it will return – 1 and set *errno* to EAGAIN (but see BUGS). Currently, this flag is meaningful only for **preadv2**().

Неблокирующие вызовы: pwritev2 и RWF_NOWAIT

RWF_NOWAIT (since Linux 4.14)

Do not wait for data which is not immediately available. If this flag is specified, the **preadv2()** system call will return instantly if it would have to read data from the backing storage or wait for a lock. If some data was successfully read, it will return the number of bytes read. If no bytes were read, it will return – 1 and set *errno* to EAGAIN (but see BUGS). Currently, this flag is meaningful only for preadv2().

Поллинг

Поллинг

- `select`

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

Поллинг

- `select`

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions

- `poll`

Regular files shall always poll TRUE for reading and writing

Asio random_access_file

```
auto file = asio::random_access_file{
   ctx, "/dev/zero", asio::file_base::read_write};

file.async_read_some_at(0, buf, [](asio::error_code ec,
   size_t len) {
   /* ... */
});
```

Asio random_access_file

```
auto file = asio::random_access_file{
   ctx, "/dev/zero", asio::file_base::read_write};

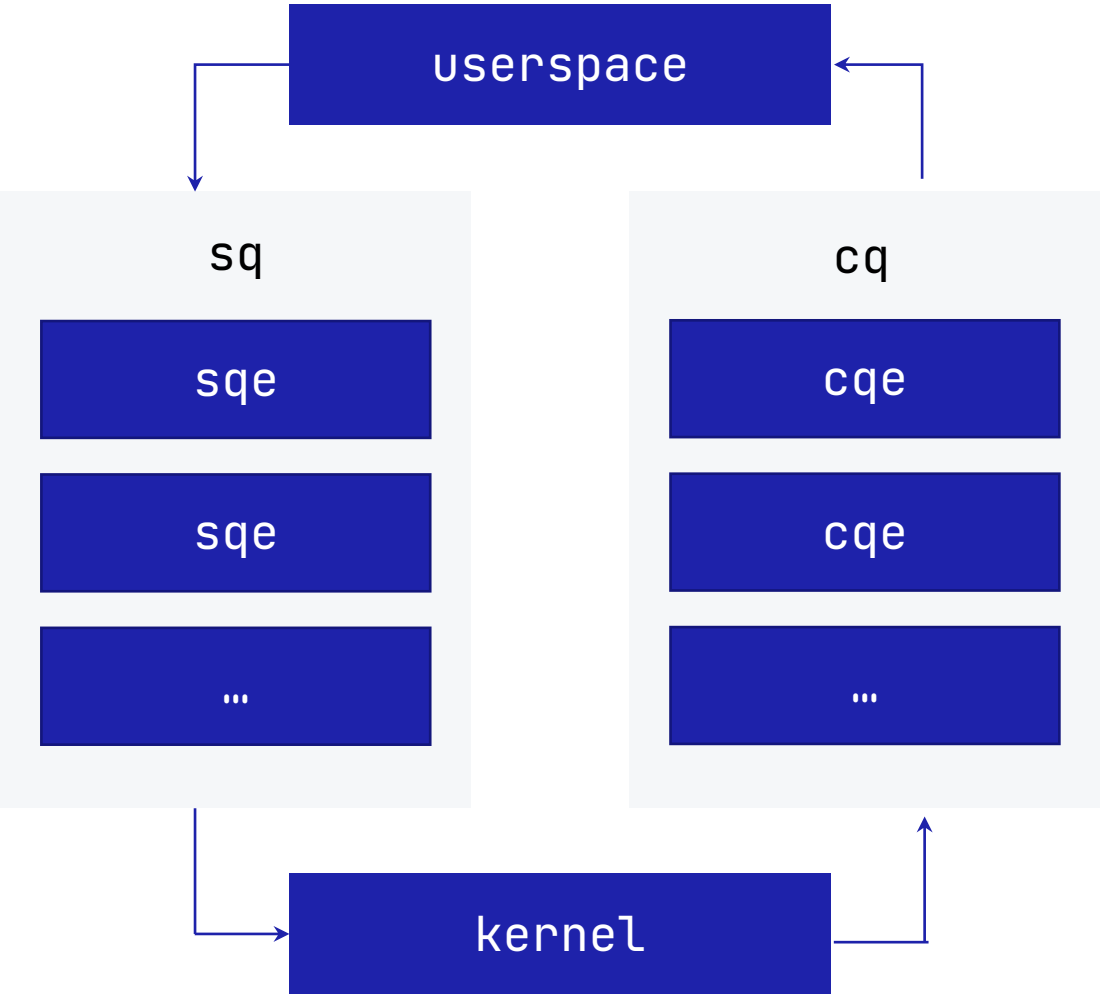
file.async_read_some_at(0, buf, [](asio::error_code ec,
   size_t len) {
   /* ... */
});
```

Магия?

io_uring!

Вспомним об io_uring

- **ring**
  - submission queue (sq)
    - sq event (sqe)
  - completion queue (cq)
    - cq event (cqe)



Инициализация io_uring

- `int io_uring_queue_init_params(unsigned entries, struct io_uring *ring, struct io_uring_params *p);`
- `int io_uring_queue_init(unsigned entries, struct io_uring *ring, unsigned flags);`

Получение sqe

- ```
  struct io_uring_sqe* io_uring_get_sqe(unsigned entries,
      struct io_uring *ring);
  ```
  ```

Подготовка sqe

- `void io_uring_prep_readv(struct io_uring_sqe *sqe, ...);`
- `void io_uring_prep_writev(struct io_uring_sqe *sqe, ...);`
- `...`

sqe submission

- `int io_uring_submit(struct io_uring *ring);`

Completion

- `int io_uring_wait_cqe(struct io_uring *ring,`
  `struct io_uring_cqe **cqes);`
- `unsigned io_uring_peek_batch_cqe(struct io_uring *ring,`
  `struct io_uring_cqe **cqes, unsigned count);`
- `...`
- `void io_uring_cqe_seen(struct io_uring *ring, struct io_uring_cqe *cqe)¶`

Asio и io_uring!

- **standalone**

```
asio >= 1.21.0
-DASIO_HAS_IO_URING=1
```

- **boost**

```
boost >= 1.78.0
-DBOOST_ASIO_HAS_IO_URING=1
```

Проблема

```
file.async_read_some_at(0, buf, [&](asio::error_code ec,
   size_t len) {
   /* ... */
   file.async_write_some_at(0, buf, [&](
      asio::error_code ec, size_t len) {
        /* ... */
```

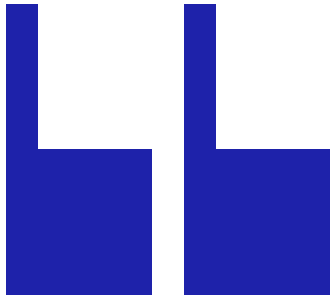
Проблема

```
timer.async_wait([&](asio::error_code ec) {
            /* ... */
            file.async_read_some_at(0, buf,
                  [&](asio::error_code ec, size_t len) {
                        /* ... */
            });
        });
    });
});
```

Проблема

- Тяжело читать
- Тяжело понимать
- Тяжело поддерживать

Проблема



if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program

Цитата из Linux kernel coding style

Asio: Completion Token

- `operator()(CompletionSignature);`
- `asio::use_future;`
- `asio::use_awaitable;`
- ...

Asio::use_future

```
file.async_read_some_at(...) -> std::future<size_t>
timer.async_wait(...) -> std::future<void>
/* ... */
```

Asio::use_future

```
auto f1 = file.async_read_some_at(0, buf, asio::use_future);
auto len2 = std::move(f1).get();
// ...
auto f2 = file.async_write_some_at(0, buf,
   asio::use_future);
auto len2 = std::move(f2).get();
// ...
```

Asio::use_future

- После каждой операции все равно приходится блокироваться
- Если возникнет ошибка бросится исключение

Yaclib::Future

```
yaclib::Run(cpu_tp, [] {
    return 42;
}).ThenInline([](int r) {
    return r + 1;
}).Then([](int r) {
    return std::to_string(r);
})
```

Yaclib::Future

```
yaclib::Run(cpu_tp, [] {
    return 42;
}).ThenInline([](int r) {
    return r + 1;
}).Then([](int r) {
    return std::to_string(r);
})
```

Yaclib::Future

```
enum class [[nodiscard]] ResultState : unsigned char{
    Value = 0,
    Exception = 1,
    Error = 2,
    Empty = 3,
};
```

Yaclib::Future

- **`Then`** и **`ThenInline`** позволяют передать callback вместо блокировки на future
- Кроме ожидаемого типа результатом может быть ошибка

исключение не будет выброшено если вы сами того не захотите

Решим проблему

```
auto f = file.async_read_some_at(0, buf, useYaclibFuture)
  .ThenInline([&] (size_t len) {
      // ...
      return file.async_write_some_at(0, buf,
        useYaclibFuture);
  })
  // ...
```

Решим проблему

```
.ThenInline([&] (size_t len) {
    // ...
    return timer.async_wait(useYaclibFuture);
})
.ThenInline([&] {
    // ...
    return file.async_read_some_at(0, buf,
      useYaclibFuture);
});
```

Как заставить asio возвращать yaclib::Future?

Посмотрим как это происходит в других случаях

Asio и std::future

```
template <typename Allocator, typename R, typename... Args>
class async_result<use_future_t<Allocator>, R(Args...)>
```

Asio и C++20 корутины

```
template <typename Allocator, typename R, typename... Args>
class async_result<use_future_t<Allocator>, R(Args...)>

template <typename Executor, typename R, typename... Args>
class async_result<use_awaitable_t<Executor>, R(Args...)>
```

Частичная специализация async_result

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)>
```

Completion Signature

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)>
```

Completion Signature async_write_some_at

```
return async_initiate<WriteToken,
    void (asio::error_code, std::size_t)>(
        initiate_async_write_some_at(this), token, offset,
         buffers);
```

Completion Signature async_write_some_at

```
return async_initiate<WriteToken,
    void (asio::error_code, std::size_t)>(
        initiate_async_write_some_at(this), token, offset,
         buffers);
```

Инициирующая функция специализирует Completion Signature

Completion Signature async_wait

```
return async_initiate<WaitToken, void (asio::error_code)>(
    initiate_async_wait(this), token);
```

Completion Signature async_wait

```
return async_initiate<WaitToken, void (asio::error_code)>(
    initiate_async_wait(this), token);
```

Completion Signature

На основе **Completion Signature** и **Completion Token** выводится возвращаемое значение

```
file.async_read_some_at(Token) -> std::future<size_t>
timer.async_wait(Token) -> std::future<void>
/* ... */
```

Async_result return_type

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)> {
public:
    using return_type = yaclib::Future<?>;
}
```

Completion Handler

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)> {
public:
   using completion_handler_type =
      MyCompletionHandler<R(Args...)>;
   using return_type =
      typename completion_handler_type::return_type;
};
```

Completion Handler

```
template <typename Signature>
class MyCompletionHandler {
public:
    /* operator() according to Signature */
};
```

Completion Handler operator()

```
template <void(asio::error_code)>
class MyCompletionHandler {
public:
   void operator()(asio::error_code);
};
```

Completion Handler operator()

```
template <void(asio::error_code)>
class MyCompletionHandler {
public:
    void operator()(asio::error_code);
};
```

Completion Handler constructor

```
template <typename Signature>
class MyCompletionHandler {
public:
    /* operator() according to Signature */
    explicit MyCompletionHandler(CompletionToken);
};
```

Async_result

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)> {
public:
   using completion_handler_type =
      MyCompletionHandler<R(Args...)>;
   using return_type = /* ... */;
};
```

Async_result constructor и get()

```
template <typename R, typename... Args>
class async_result<UseYaclibFuture, R(Args...)> {
public:
    using completion_handler_type = /* ... */;
    using return_type = /* ... */;

    explicit async_result(completion_handler_type&);
    return_type get();
};
```

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`**
- в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод get и возвращает это значение

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

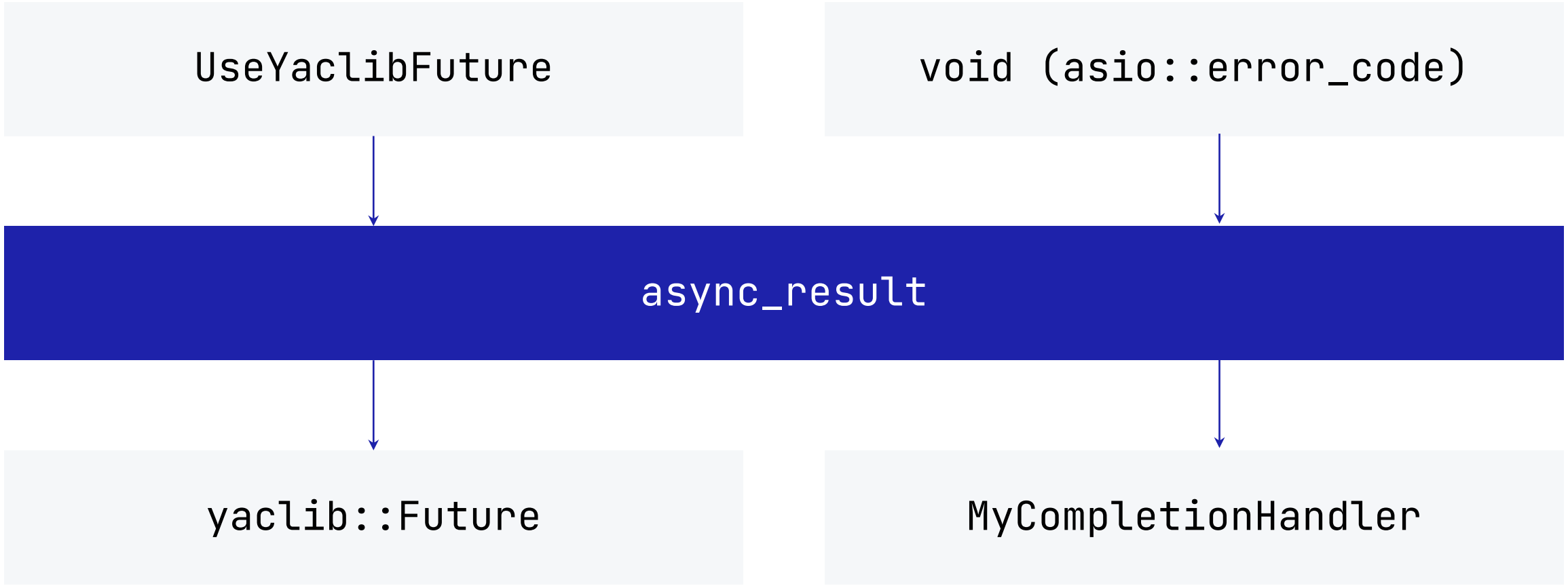
Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

Asio workflow

Asio workflow

Asio workflow



Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Asio custom Completion Token

`async_result`

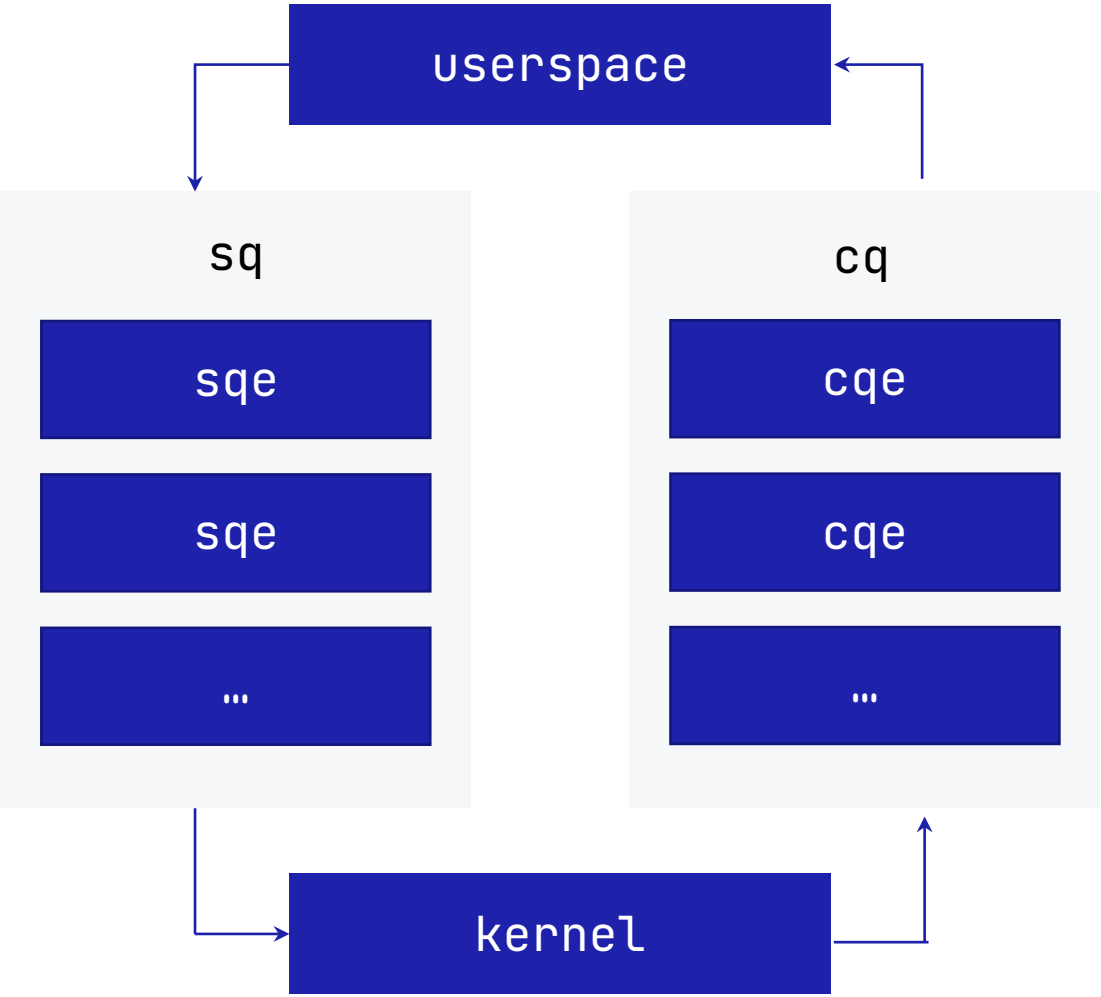
- **`completion_handler_type`** typedef
- **`return_type`** typedef
- Constructor
- **`get()`** method

Completion Handler

- Constructor
- **`operator()`** according to Completion Signature

Вспомним об io_uring

- **ring**
  - submission queue (sq)
    - sq event (sqe)
  - completion queue (cq)
    - cq event (cqe)



Инициализация io_uring

- `int io_uring_queue_init_params(unsigned entries, struct io_uring *ring, struct io_uring_params *p);`
- `int io_uring_queue_init(unsigned entries, struct io_uring *ring, unsigned flags);`

Инициализация io_uring

- **`io_uring`** можно инициализировать с произвольной глубиной очереди
- С помощью флагов при инициализации можно менять поведение **`io_uring`**

Инициализация io_uring

IORING_SETUP_SQPOLL

When this flag is specified, a kernel thread is created to perform submission queue polling. An io_uring instance configured in this way enables an application to issue I/O without ever context switching into the kernel.
By using the submission queue to fill in new submission queue entries and watching for completions on the completion queue, the application can submit and reap I/Os without doing a single system call

Инициализация io_uring

IORING_SETUP_SQPOLL

When this flag is specified, a kernel thread is created to perform submission queue polling. An io_uring instance configured in this way enables an application to issue I/O without ever context switching into the kernel.
By using the submission queue to fill in new submission queue entries and watching for completions on the completion queue, the application can submit and reap I/Os without doing a single system call

Как в asio управлять io_uring?

Никак :(

```
/* ... */
void io_uring_service::init_ring()
{
 int result = ::io_uring_queue_init(ring_size, &ring_, 0);
 if (result < 0)
/* ... */
private:
 enum { ring_size = 16384 };
/* ... */
```

Как написать свое?

Asio workflow

- Создаем **`execution_context`**
- Создаем **`io`** объект
- Совершаем операции с **`io`** объектом

Asio execution_context

- `io_context`
- `system_context`
- `thread_pool`

Asio workflow

- Создаем **`execution_context`**
- Создаем **`io`** объект
- Совершаем операции с **`io`** объектом

Asio io объекты

- random_access_file
- stream_file
- steady_timer
- …

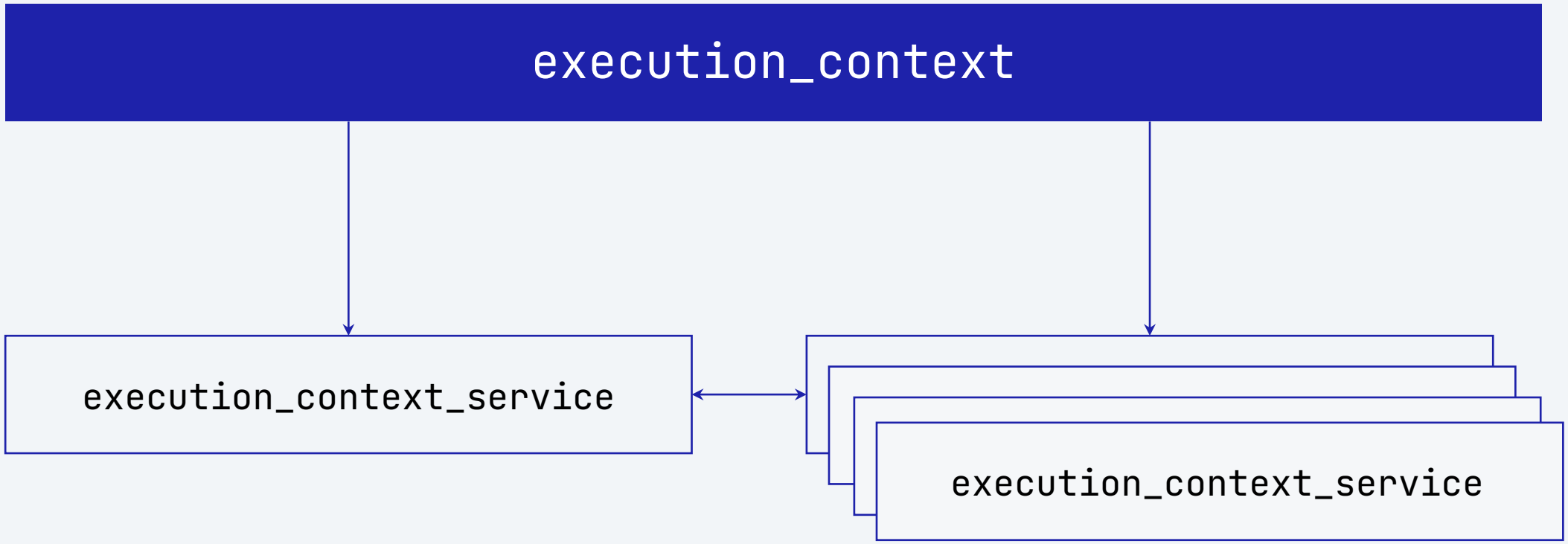
Asio workflow

- Создаем **`execution_context`**
- Создаем **`io`** объект
- Совершаем операции с **`io`** объектом

Asio io объекты

- `async_wait`
- `async_read_some_at`
- `async_write_some_at`
- `async_read_some`
- `async_write_some`
- …

Asio внутри


execution_context
execution_context_service
execution_context_service

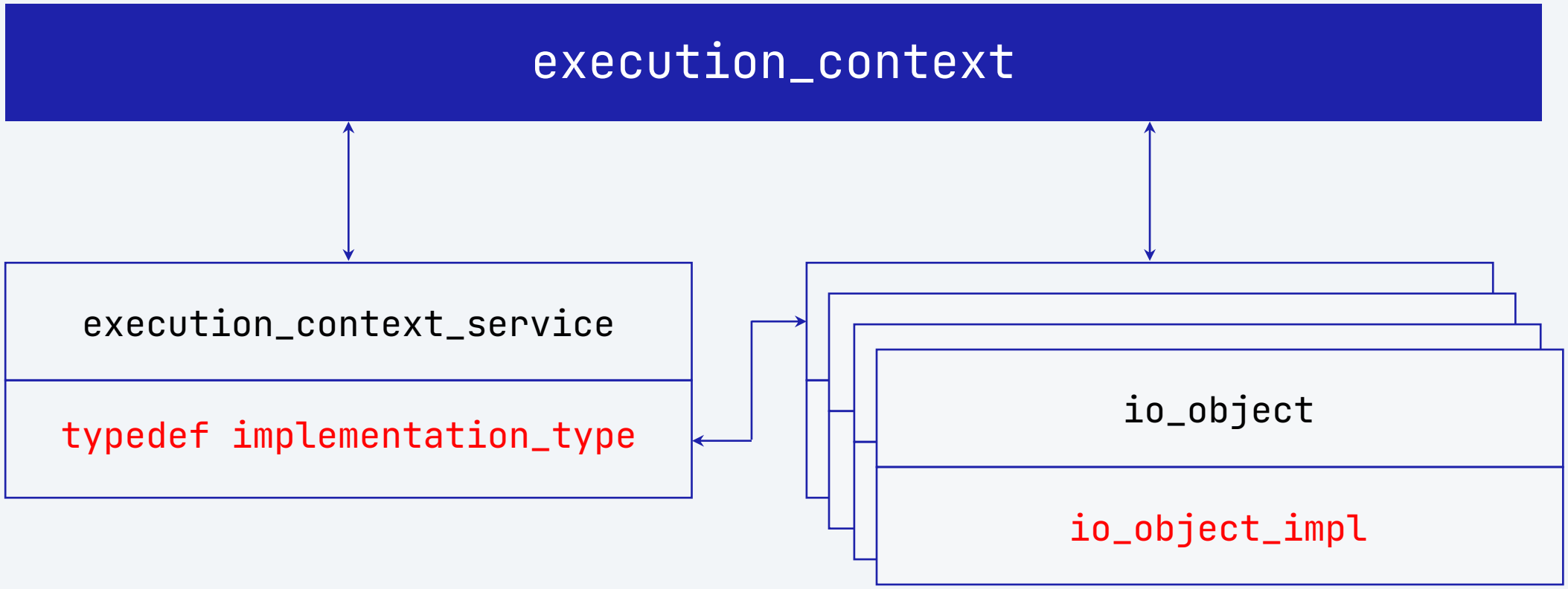
Asio внутри

```
template <typename Service>
Service& use_service(execution_context&);

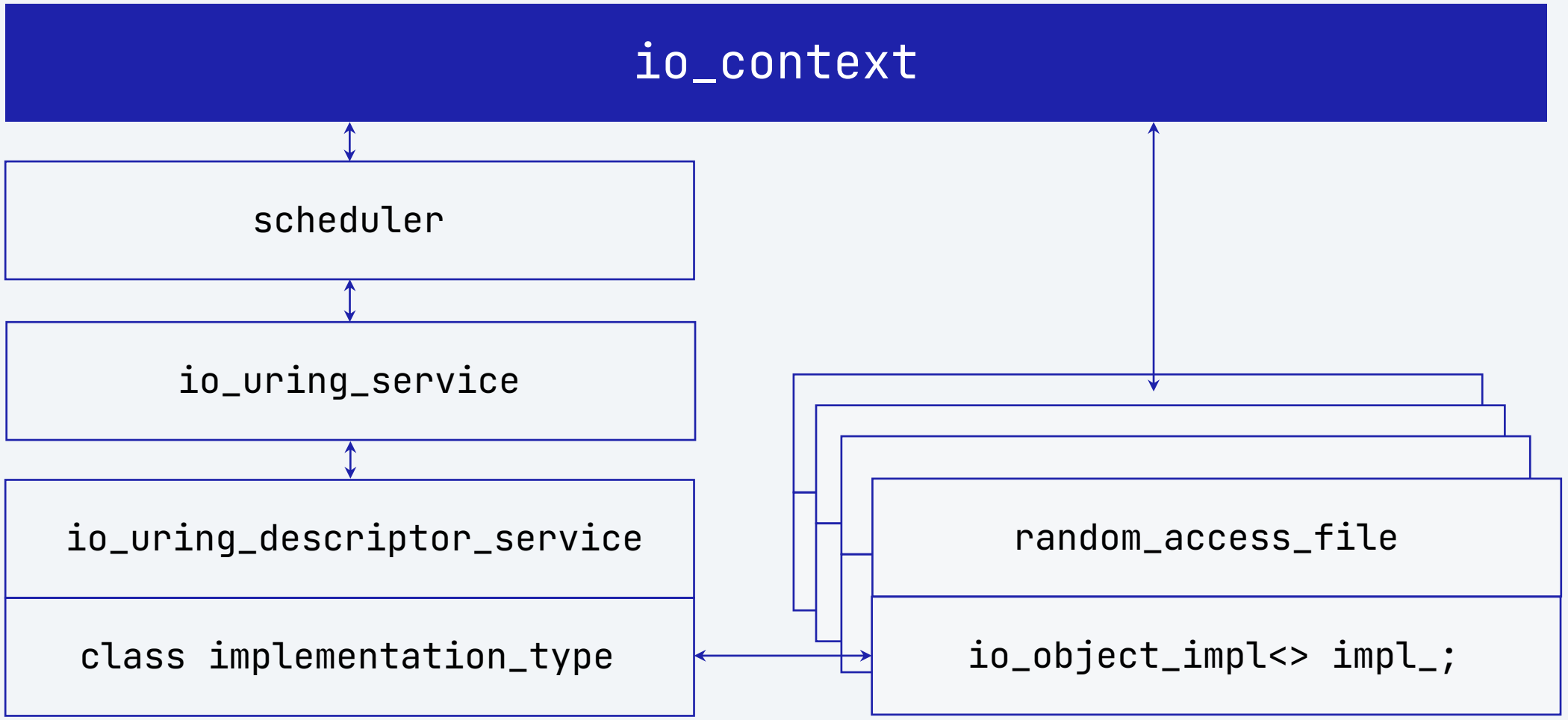
template <typename Service>
void add_service(execution_context&, Service*);

template <typename Service>
bool has_service(execution_context&);
```

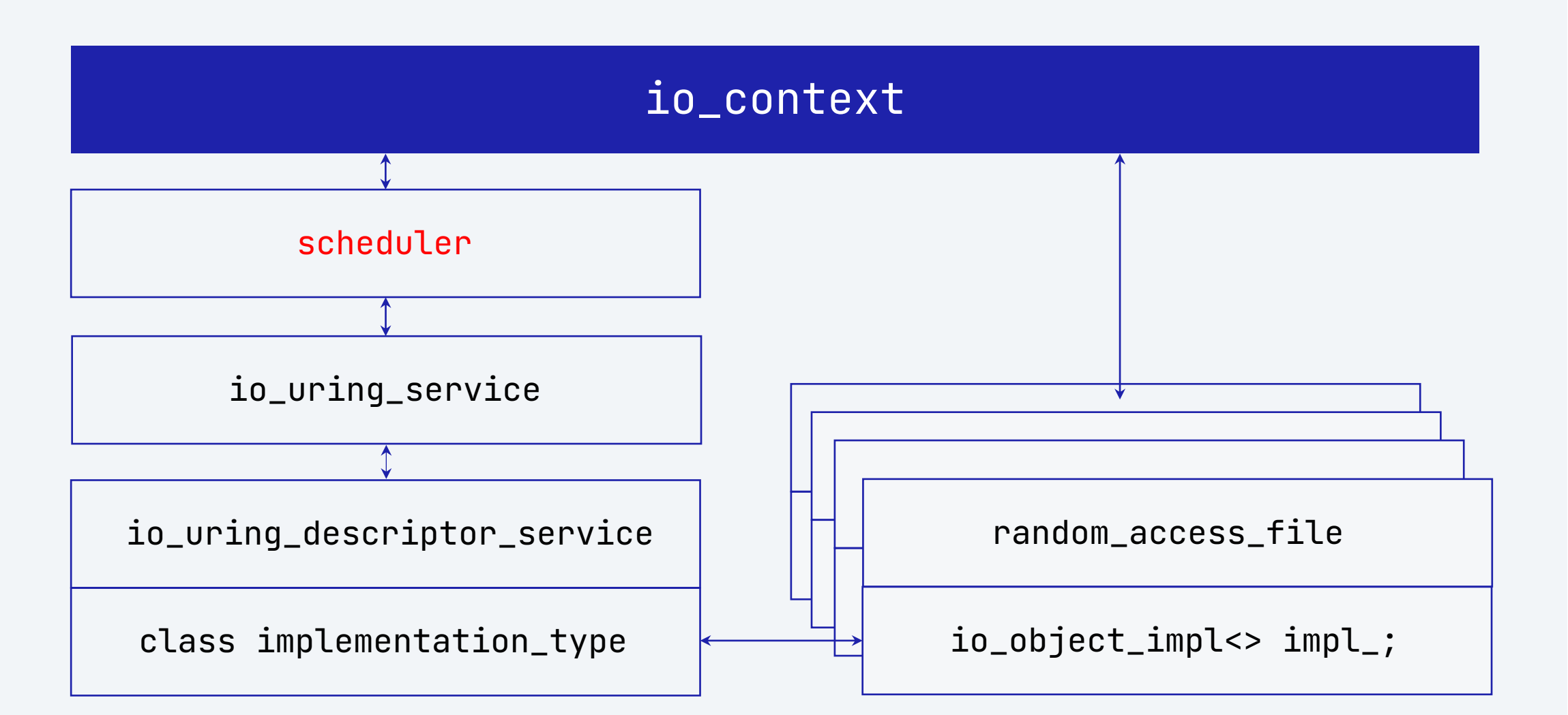
Asio внутри


execution_context
execution_context_service
typedef implementation_type
io_object
io_object_impl

Asio внутри


io_context
scheduler
io_uring_service
io_uring_descriptor_service
class implementation_type
random_access_file
io_object_impl<> impl_;

Scheduler


io_context
scheduler
io_uring_service
io_uring_descriptor_service
class implementation_type
random_access_file
io_object_impl<> impl_;

Scheduler

```
myContext.run();

io_context::count_type io_context::run()
{
 asio::error_code ec;
 count_type s = sched_.run(ec);
 asio::detail::throw_error(ec);
 return s;
}
```

Scheduler

```
myContext.run();

io_context::count_type io_context::run()
{
 asio::error_code ec;
 count_type s = sched_.run(ec);
 asio::detail::throw_error(ec);
 return s;
}
```

Scheduler

```
myContext.run();

io_context::count_type io_context::run()
{
 asio::error_code ec;
 count_type s = sched_.run(ec);
 asio::detail::throw_error(ec);
 return s;
}
```

Scheduler event loop

- Исполняет **`task operation`**
- Завершает асинхронные операции **`scheduler_operation`**

Scheduler task operation

```
class scheduler_task
{
public:
 virtual void run(long usec,
   op_queue<scheduler_operation>& ops) = 0;
 virtual void interrupt() = 0;
```

Io_uring_service run(...) method

```
int result = (usec == 0)
 ? ::io_uring_peek_cqe(&ring_, &cqe)
/* ... */
if (void* ptr = ::io_uring_cqe_get_data(cqe))
/* ... */
     io_queue* io_q = static_cast<io_queue*>(ptr);
     io_q->set_result(cqe->res);
     ops.push(io_q);
```

Io_uring_service run(...) method

```
class io_queue : scheduler_operation
{
/* ... */
```

Scheduler ctor

```
typedef scheduler_task* (*get_task_func_type)(
   asio::execution_context&);

ASIO_DECL scheduler(asio::execution_context& ctx,
   int concurrency_hint = 0, bool own_thread = true,
   get_task_func_type get_task =
      &scheduler::get_default_task);
```

Scheduler get_default_task

```
scheduler_task*
scheduler::get_default_task(asio::execution_context& ctx)
{
 return &use_service<io_uring_service>(ctx);
}
```

Scheduler

- В контекстах что идут из коробки используется дефолтный **`get_task`** при инициализации **`scheduler`**
- Нет способа переопределить это значение после инициализации **`scheduler`**

Кастомизируем контексты

- Базовый класс **`execution_context`**
- Базовый шаблон класса **`exectuion_context_service_base`**

Execution_context_service_base

```
virtual void shutdown() = 0;
```

Execution_context_service_base

```
/* Must be default constructible */
 class implementation_type /* ... */
 ASIO_DECL void construct(implementation_type& impl);
 ASIO_DECL void destroy(implementation_type& impl);
```

Execution_context_service_base

```
/* Must be default constructible */
 class implementation_type /* ... */
 ASIO_DECL void move_construct(implementation_type& impl,
     implementation_type& other_impl) ASIO_NOEXCEPT;
 ASIO_DECL void move_assign(implementation_type& impl,
     io_uring_descriptor_service& other_service,
     implementation_type& other_impl);
```

Точки кастомизации закончились

Хелперы

Async_initiate

```
auto async_initiate(Initiation initiation, CompletionToken
    token, Args&&... args);
```

Asio workflow

- Инициирующая функция определяет **`Completion Signature`**
- Берет как параметр **`Completion Token`**
- Передает **`Completion Token`** и **`Completion Signature`** в **`async_result`**
- Создает объект типа **`async_result::completion_handler_type`** и передает **`Completion Token`** в его конструктор
- Создает объект типа **`async_result`** и передает **`Completion Handler`** в его конструктор
- У объекта типа **`async_result`** вызывает метод **`get`** и возвращает это значение

Initiation

```
auto async_initiate(Initiation initiation, CompletionToken
    token, Args&&... args);
```

Initiation

```
class initiate_async_write_some_at
{
/* ... */
 template <typename WriteHandler, typename
ConstBufferSequence>
 void operator()(ASIO_MOVE_ARG(WriteHandler) handler,
       uint64_t offset, const ConstBufferSequence& buffers)
```

Initiation

```
class initiate_async_write_some_at
{
/* ... */
 template <typename WriteHandler, typename
ConstBufferSequence>
 void operator()(ASIO_MOVE_ARG(WriteHandler) handler,
       uint64_t offset, const ConstBufferSequence& buffers)
```

Initiation

```
self_->impl_.get_service().async_write_some_at(
    self_->impl_.get_implementation(), offset, buffers,
    handler, self_->impl_.get_executor());
```

Async_initiate

- Автоматизация рутины выведения Completion Handler

Аллокация асинхронной операции

```
class io_uring_descriptor_read_at_op
 : public
io_uring_descriptor_read_at_op_base<MutableBufferSequence>
{
public:
 ASIO_DEFINE_HANDLER_PTR(io_uring_descriptor_read_at_op);
```

Аллокация асинхронной операции

```
class io_uring_descriptor_read_at_op
 : public
io_uring_descriptor_read_at_op_base<MutableBufferSequence>
{
public:
 ASIO_DEFINE_HANDLER_PTR(io_uring_descriptor_read_at_op);
```

ASIO_DEFINE_HANDLER_PTR

- **`struct ptr`**
  - **`allocate`** method
  - **`reset`** method

ASIO_DEFINE_HANDLER_PTR

```
typedef io_uring_descriptor_write_at_op<
 ConstBufferSequence, Handler, IoExecutor> op;
typename op::ptr p = { asio::detail::addressof(handler),
 op::ptr::allocate(handler), 0 };
p.p = new (p.v) op(success_ec_, impl.descriptor_,
   impl.state_, offset, buffers, handler, io_ex);
```

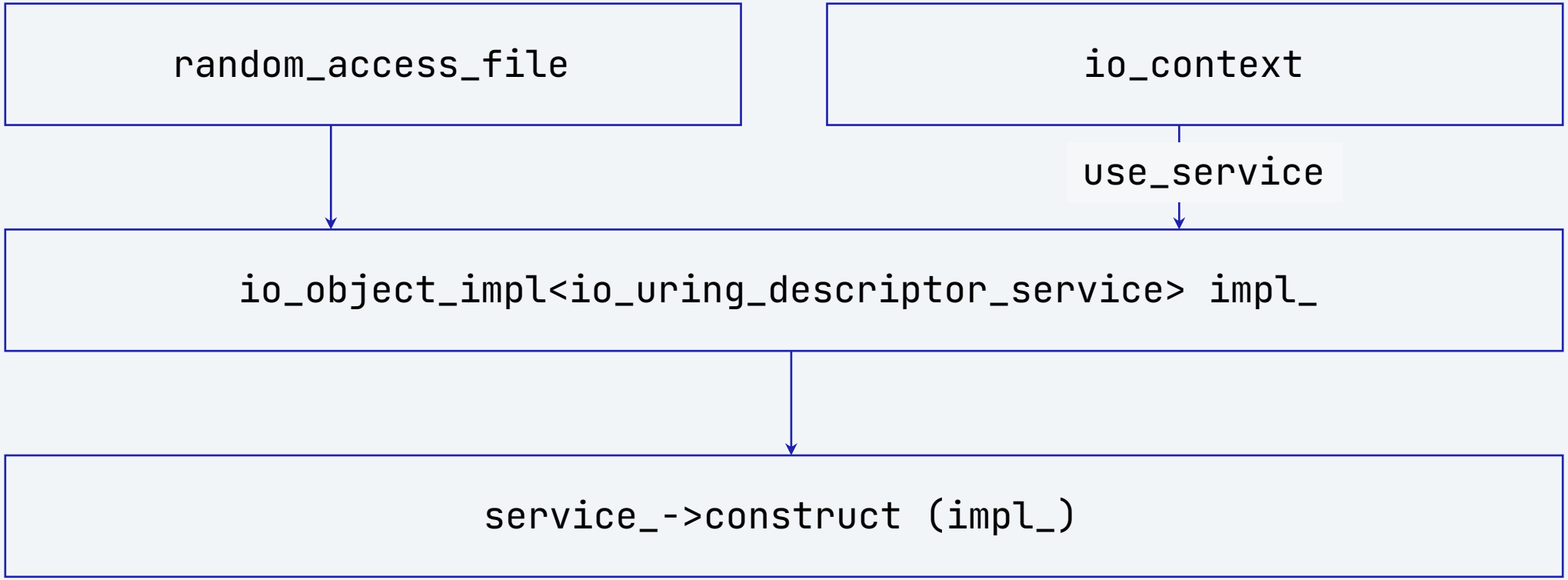
ASIO_DEFINE_HANDLER_PTR

```
typedef io_uring_descriptor_write_at_op<
 ConstBufferSequence, Handler, IoExecutor> op;
typename op::ptr p = { asio::detail::addressof(handler),
 op::ptr::allocate(handler), 0 };
p.p = new (p.v) op(success_ec_, impl.descriptor_,
   impl.state_, offset, buffers, handler, io_ex);
```

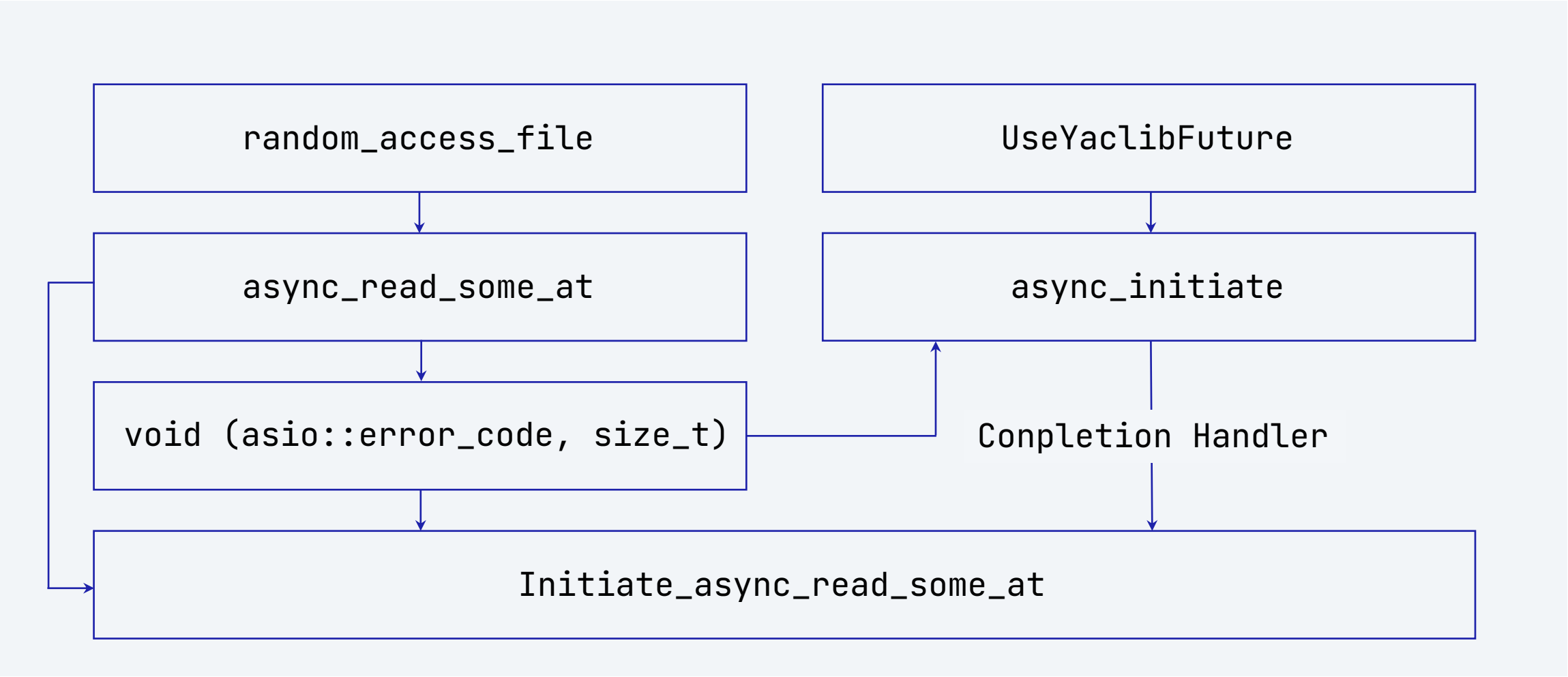
ASIO_DEFINE_HANDLER_PTR

- Автоматизация аллокации асинхронной операции

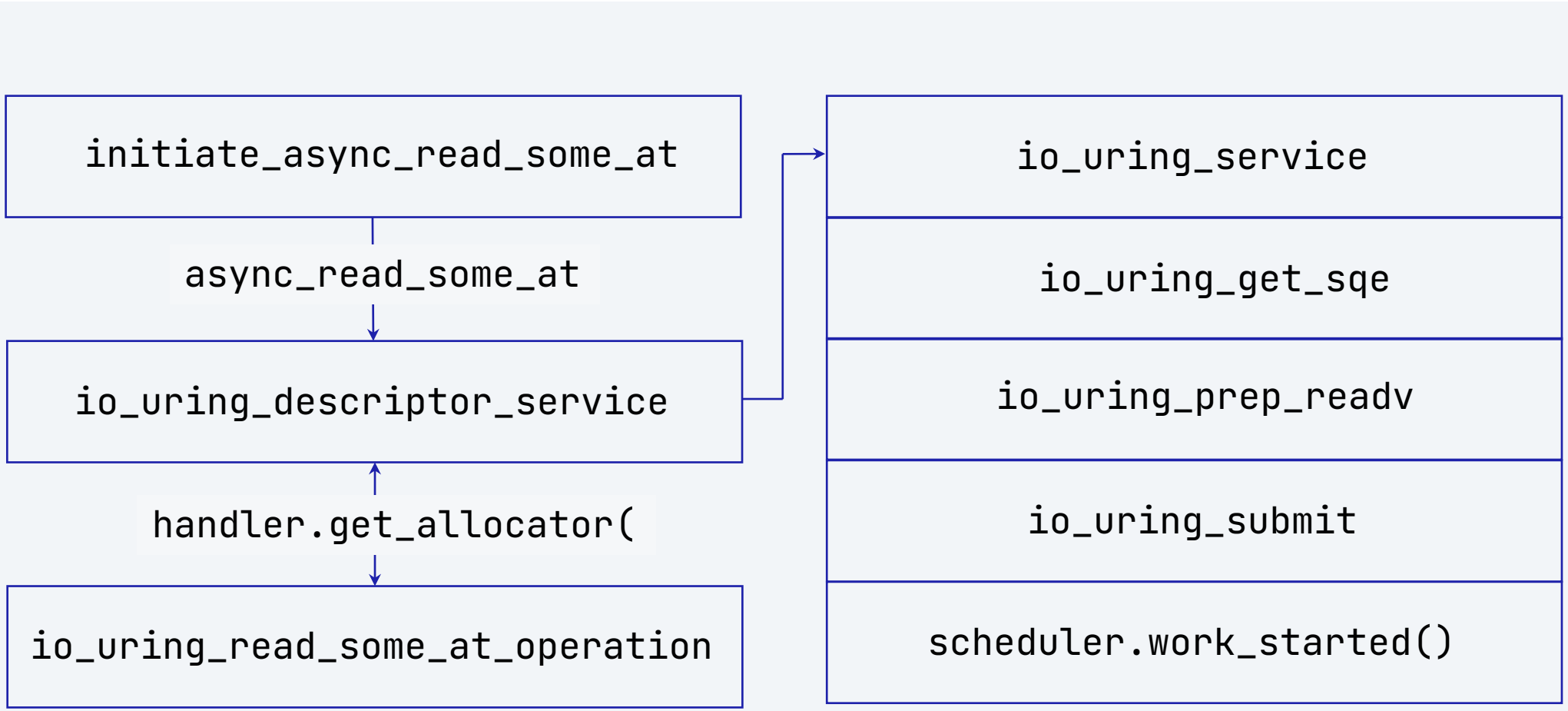
Вывод на примере уже реализованного механизма



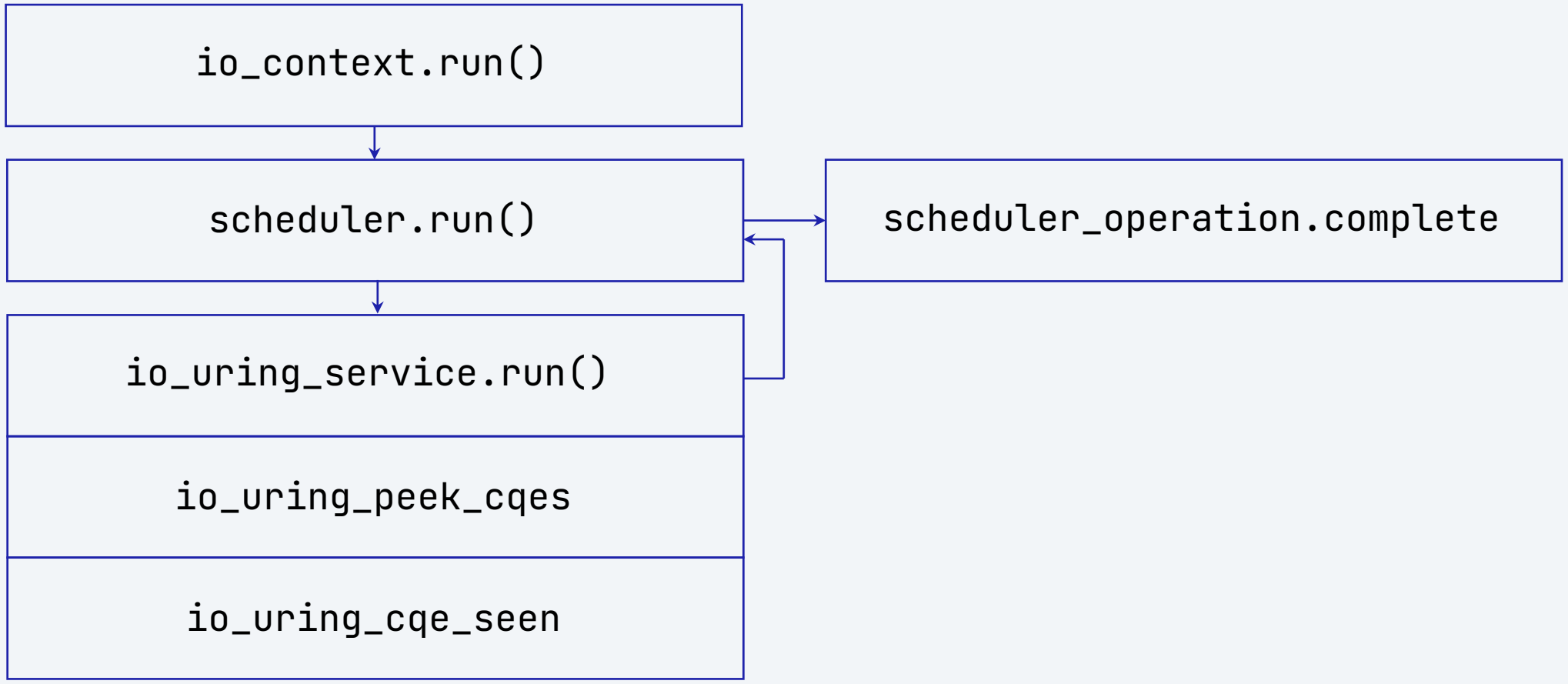
Вывод на примере уже реализованного механизма



Вывод на примере уже реализованного механизма



Вывод на примере уже реализованного механизма



Заключение

- С версии 1.21 Asio (Boost 1.78.0) умеет в асинхронность с обычными файлами и блочными устройствами используя **`io_uring`**

Заключение

- С версии 1.21 Asio умеет в асинхронность с обычными файлами и блочными устройствами используя **`io_uring`**
- Asio позволяет относительно просто добавить **`Completion Token`** на свой вкус

Заключение

- С версии 1.21 Asio умеет в асинхронность с обычными файлами и блочными устройствами используя **`io_uring`**
- Asio позволяет относительно просто добавить **`Completion Token`** на свой вкус
- Asio предоставляет определенное количество точек кастомизации и хелперов для реализации собственного backend'а (но стоит ли оно того?)

О чем не сказал?

- cancellation slots
- executors

Код с примерами

Моя почта

kazakovilya97@gmail.com

Москва,
ул. Рочдельская, 15, стр. 13
+7 800 777-06-11

yadro.com