

Dr Ivan Čukić ▶ KDAB, KDE

PROG C++

# INTRODUCTION

WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ABOUT ME

- [📧](#) KDAB PM, TL, SSE  
Software expertise in Qt, C++ and 3D / OpenGL
- Author of “Functional Programming in C++”  
available in English, Chinese, Korean, Russian, Polish
- Trainer / Consultant
- ISO WG21
- KDE developer
- University professor

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

VALUES

SAFETY



Too old to Rock 'n' Roll,  
to young to die

– Ian Anderson

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# PROG ROCK

**Prog rock** is a broad genre of rock music. The style was an emergence of psychedelic bands who abandoned standard **pop traditions** in favour of instrumentation and compositional techniques more frequently associated with **jazz, folk, or classical music**.

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

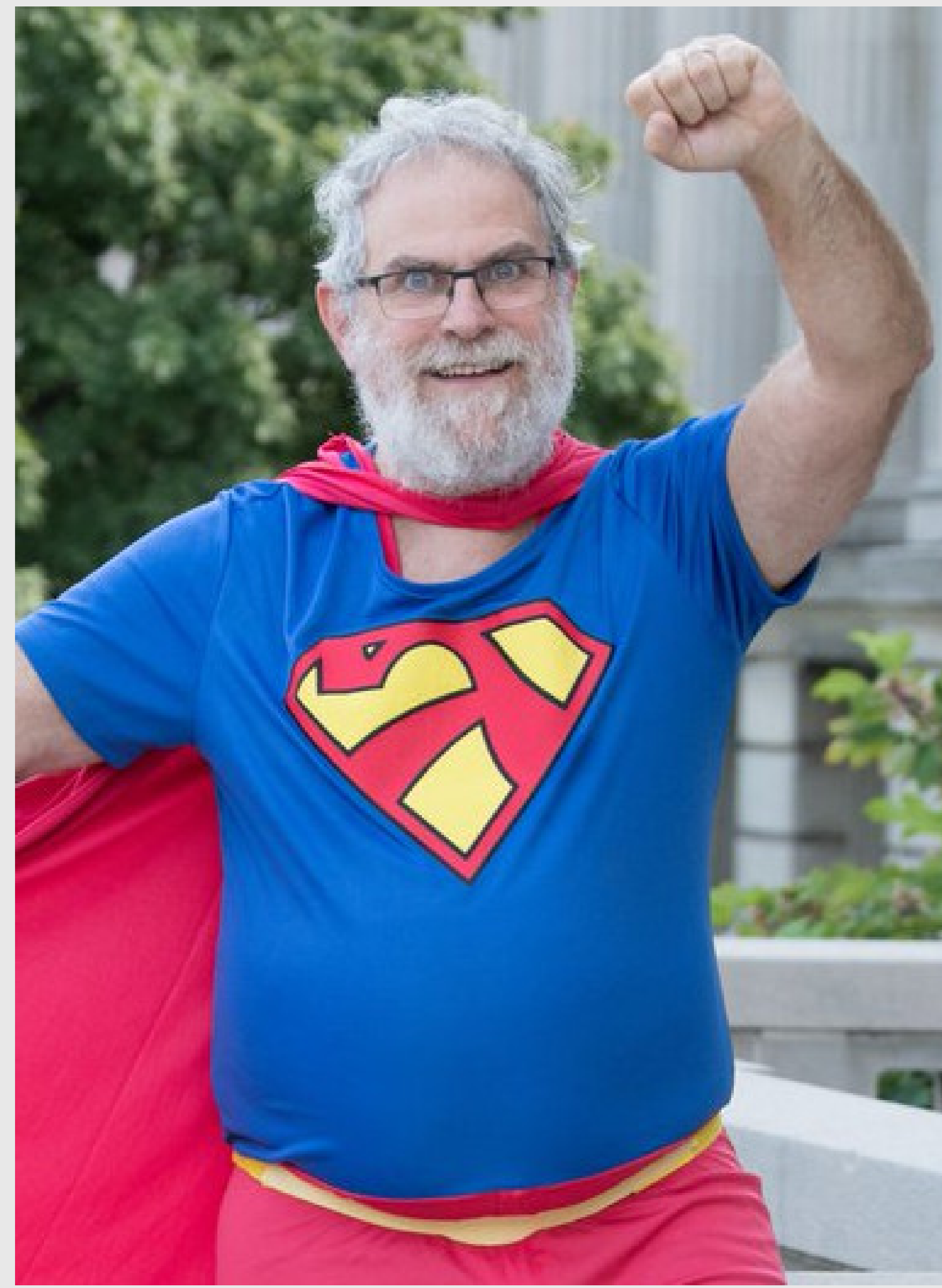
VALUES

SAFETY

**Prog C++** is a broad genre of C++ code. The style was an emergence of psychedelic developers who abandoned standard **C with classes** traditions in favour of instrumentation and compositional techniques more frequently associated with **generic, functional, or value-based object oriented** coding practices.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# DISCLAIMER



Make your code readable. Pretend the next person who looks at your code is a psychopath and they know where you live.

– Philip Wadler

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

VALUES

SAFETY

# DISCLAIMER

The code snippets are optimized for presentation,  
it is often **not** production-ready code.

- **\_t** suffix is used for user types,
- assumes **using std::string\_literals,**
- ...

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

VALUES

SAFETY



# INTRODUCTION WRAPS

SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

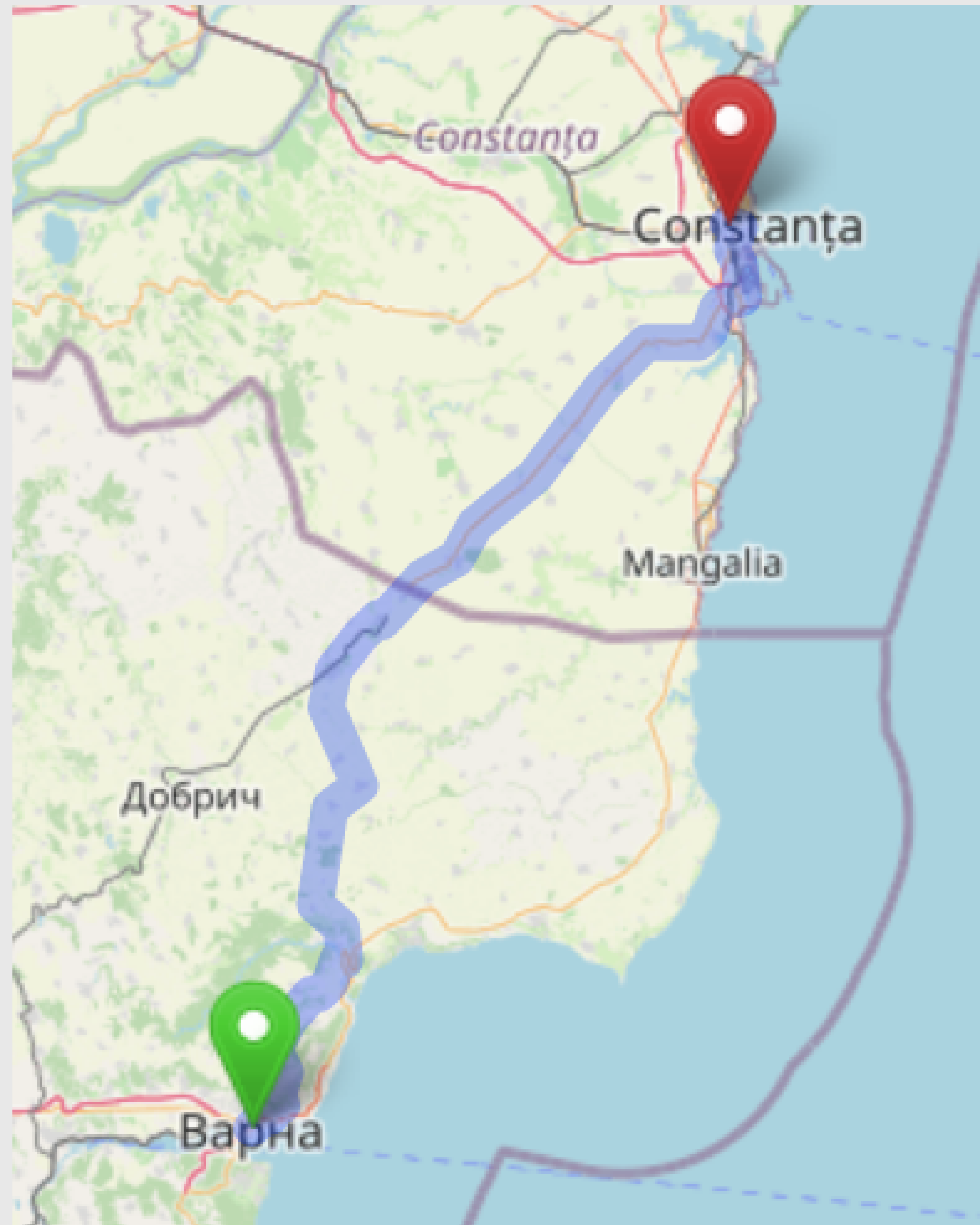
Meeting C++ 2023

Ivan Čukić [↗ kdab.com](https://kdab.com), [cukic.co](https://cukic.co)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ALMOST ALWAYS CONST

Constanța, Romania



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# CONST..RUCTION

```
const person_t person("Martha Jones"s, 42);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# CONST..RUCTION

```
person_t person("Martha Jones"s, 42);  
person.bad_hash =  
    person.name.size() ^ person.age;
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMEDIATE LAMBDA

```
const auto person = [] {  
    person_t result;  
    result.age = 42;  
    result.name = "Martha Jones"s;  
    result.bad_hash =  
        result.name.size() ^ result.age;  
    return result;  
}();  
  
print_person(person);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LAZY LAMBDA

```
const lazy person = [] {  
    person_t result;  
    result.age = 42;  
    result.name = "Martha Jones"s;  
    result.bad_hash =  
        result.name.size() ^ result.age;  
    return result;  
};  
  
print_person(person);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LAZY LAMBDA

```
template<typename Fn>
class lazy {
public:
    using value_type = std::invoke_result_t<Fn>;

private:
    Fn m_function;
    mutable std::once_flag m_once;
    mutable std::optional<value_type> m_data;

    // ...
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LAZY LAMBDA

```
// ...  
  
operator const value_type &() const  
{  
    std::call_once(m_once, [&] {  
        m_data = std::invoke(m_function);  
    });  
    return *m_data;  
}  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# PHASES OF CONST

- Building the value
- Using the value

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ALMOST ALWAYS CONST

Declare everything const by default, except:

- Member variables
- Local variables that will be returned from the function
- Variables you want to give away

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ALWAYS CONST

Can we have a move-enabled const?

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMUTABLE<T>

```
template<typename T>  
class immutable {  
    T m_value;  
  
    // ...  
};
```

Applied rule: Make everything const, except member variables.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMUTABLE<T>

```
template<typename T>
class immutable {
    // ...
    immutable(const immutable<T>& other) = default;
    immutable(immutable<T>&& other) = default;
    immutable& operator=(const immutable<T>& other) = default;
    immutable& operator=(immutable<T>&& other) = default;
    // ...
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMUTABLE<T>

```
template<typename T>
class immutable {
    // ...
    operator T& () const& { return m_value; }
    const T& operator*() const& { return m_value; }
    const T* operator->() const& {
        return std::addressof(m_value);
    }
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMUTABLE<T>

```
template<typename T>
class immutable {
    // ...
    operator T&& () && { return std::move(m_value); }
};
```

Applied rule: Make everything const, except variables you want to give away.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IMMUTABLE<T>

```
struct person_t { immutable<std::string> name; /* ... */ };
```

```
auto get_person() {  
    immutable<person_t> result("Martha Jones"s, 42);  
    // ...  
    return result;  
}
```

```
auto create_person() {  
    immutable person = get_person();  
    database.add(std::move(person));  
}
```

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

VALUES

SAFETY



# IMMUTABLE AND LAZY

```
lazy l_person = [] { /* ... */ };  
immutable i_person = [] { /* ... */ }();
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# AGAINST THE GRAIN

It is fine to implement things that go against the language...

... when the language is insufficient or wrong  
for your use-case or domain.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# AGAINST THE GRAIN



Belle Views on C++ Ranges,  
their Details and the Devil

Nicolai Josuttis, Meeting C++ 2022

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

INTRODUCTION  
WRAPS  
**SWAPS**  
STATES  
ERRORS  
VALUES  
SAFETY

Meeting C++ 2023

Ivan Čukić [✉ kdab.com](https://kdab.com), [cukic.co](https://cukic.co)

INTRODUCTION  
WRAPS  
**SWAPS**  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
class person_t {  
    // ...  
    person_t& operator=(const person_t& other) = default;  
    person_t& operator=(person_t&& other) = default;  
    // ...  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
class person_t {  
    // ...  
    person_t& operator=(const person_t& other) = default;  
    person_t& operator=(person_t&& other) = default;  
    // ...  
  
private:  
    std::string name;  
    std::string surname;  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
person_t person("Martha"s, "Jones"s);
```

```
person.name = "John"s; // exception
```

```
person.surname = "Smith"s;
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
person_t person("Martha"s, "Jones"s);
```

```
person_t new_person;  
new_person.name = "John"s;  
new_person.surname = "Smith"s;
```

```
swap(person, new_person);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# ASSIGNMENT

```
friend void swap(person_t& left, person_t& right) noexcept {  
    std::swap(  
        std::tie(left.name, left.surname),  
        std::tie(right.name, right.surname));  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
class person_t {  
    person_t& operator=(const person_t& other) {  
        auto temp = other;  
        swap(*this, temp);  
        return *this;  
    }  
};
```

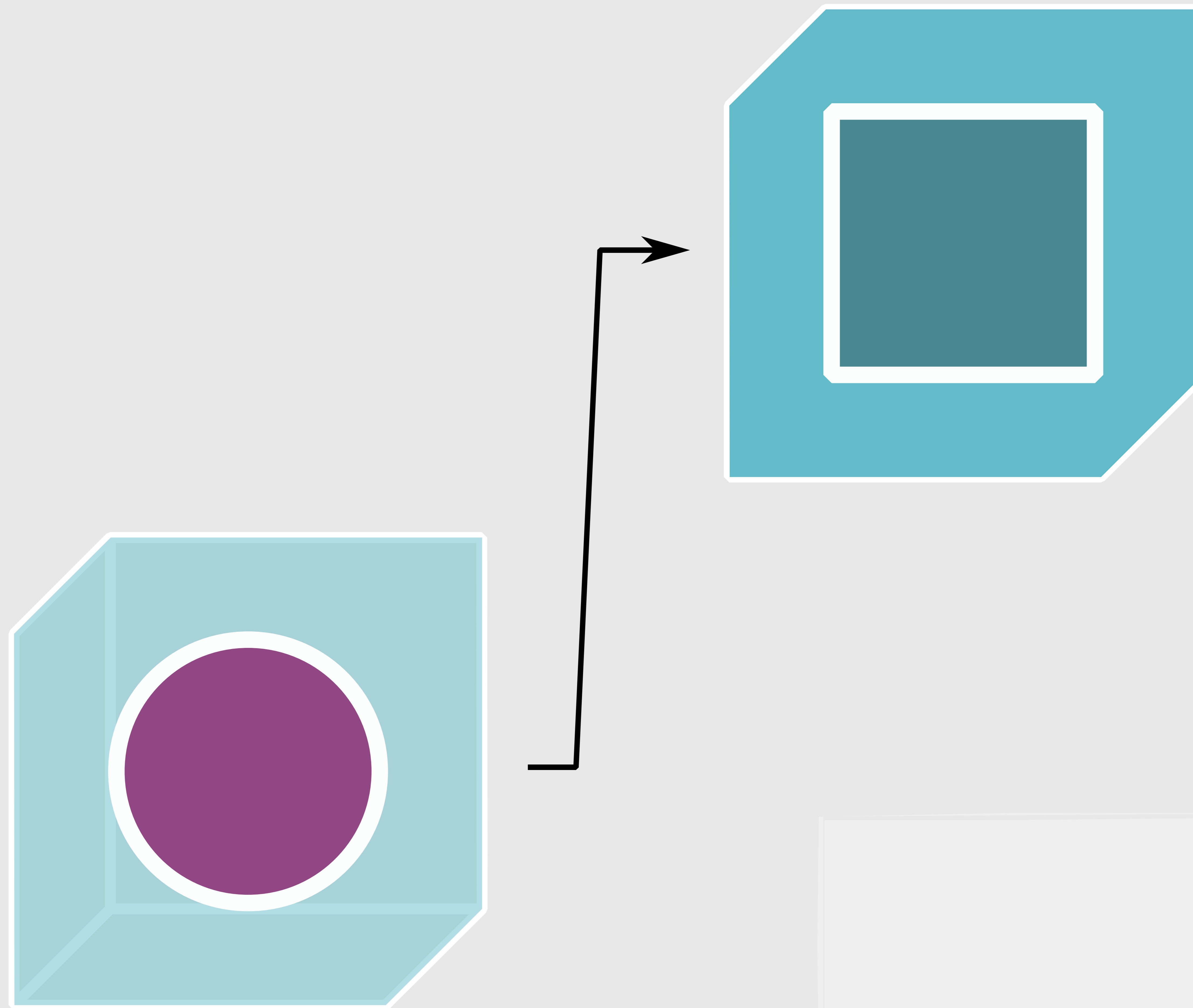
INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

```
class person_t {  
    person_t& operator=(person_t&& other) {  
        auto temp = std::move(other);  
        swap(*this, temp);  
        return *this;  
    }  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT

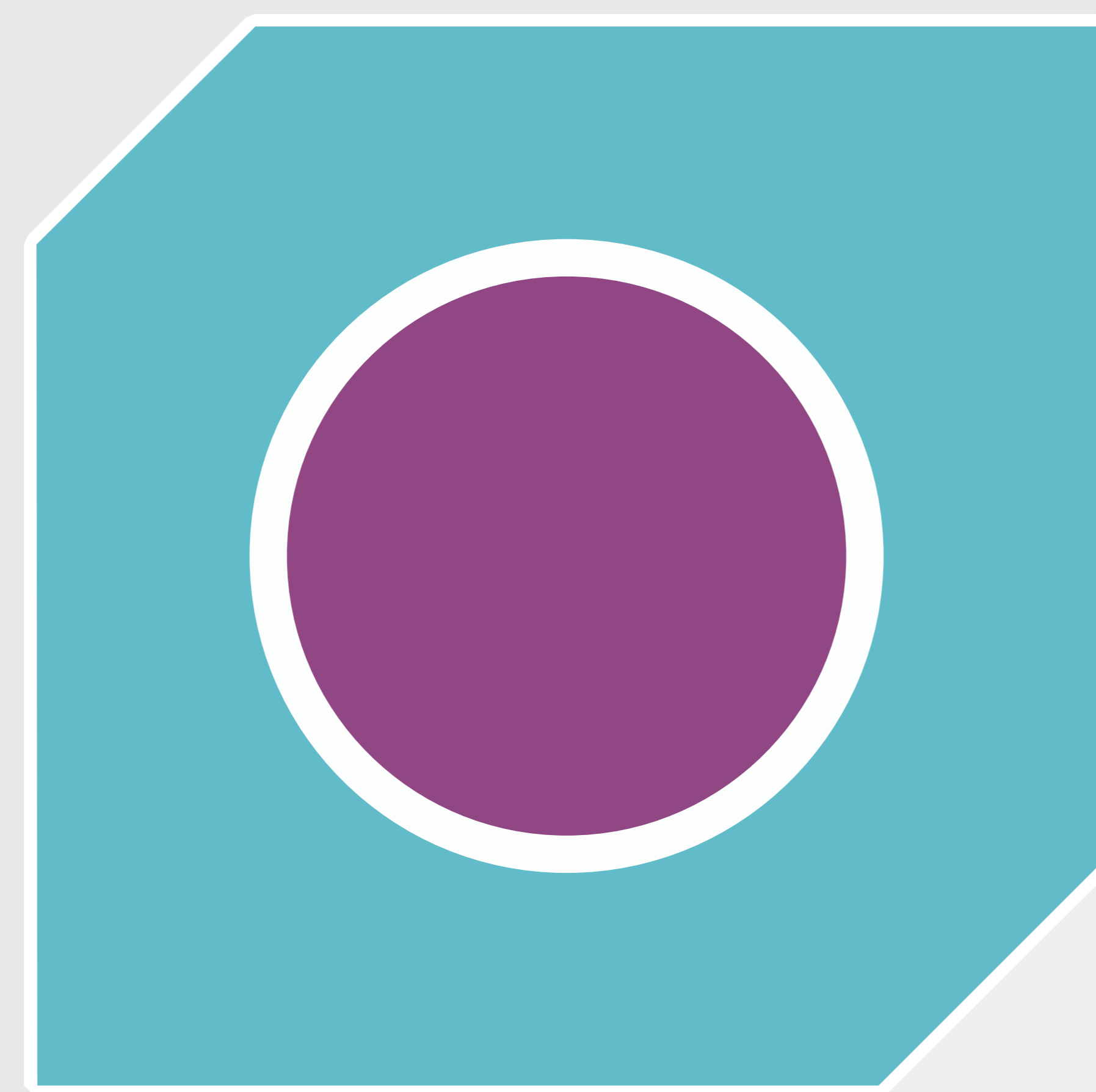
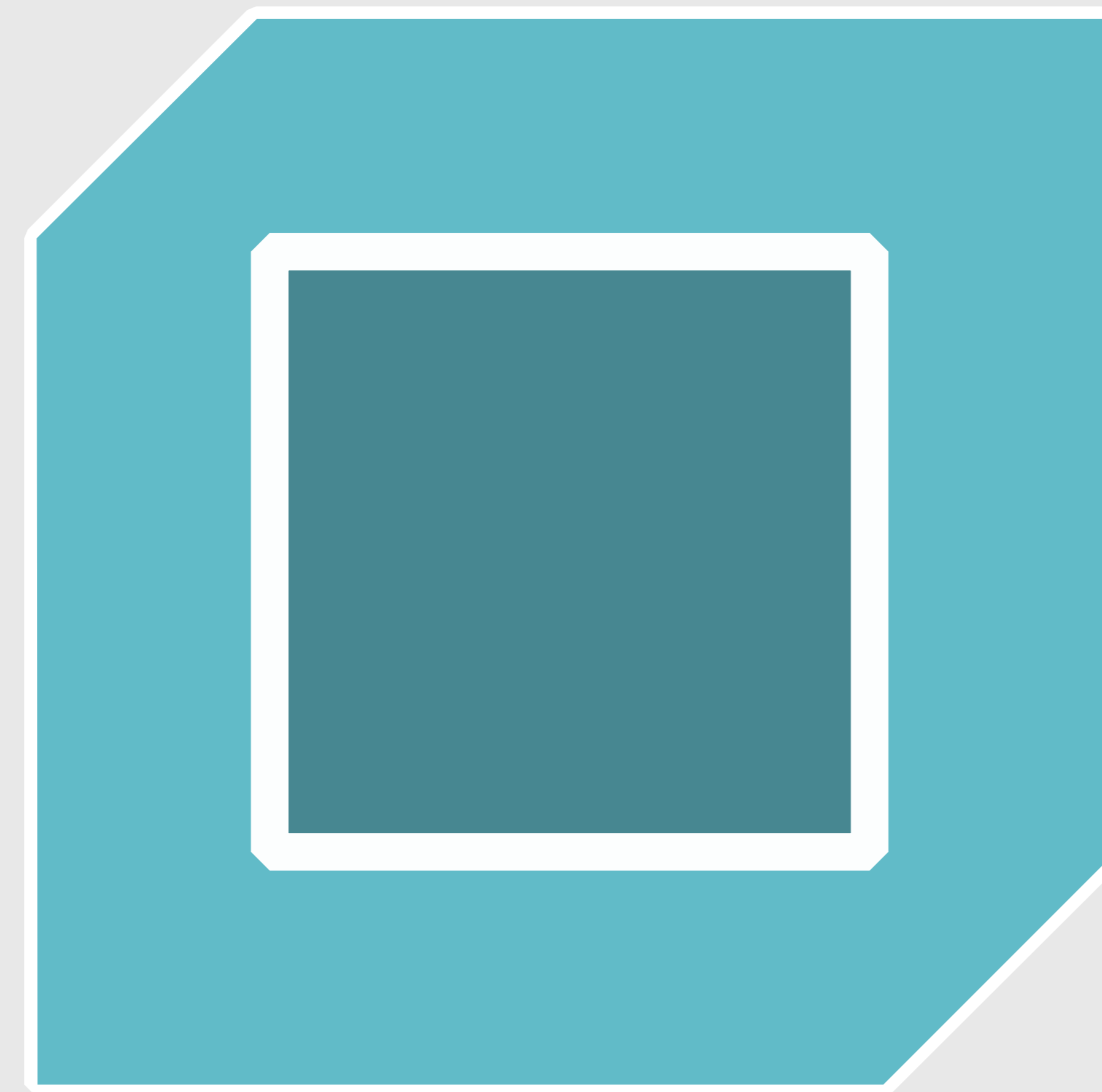
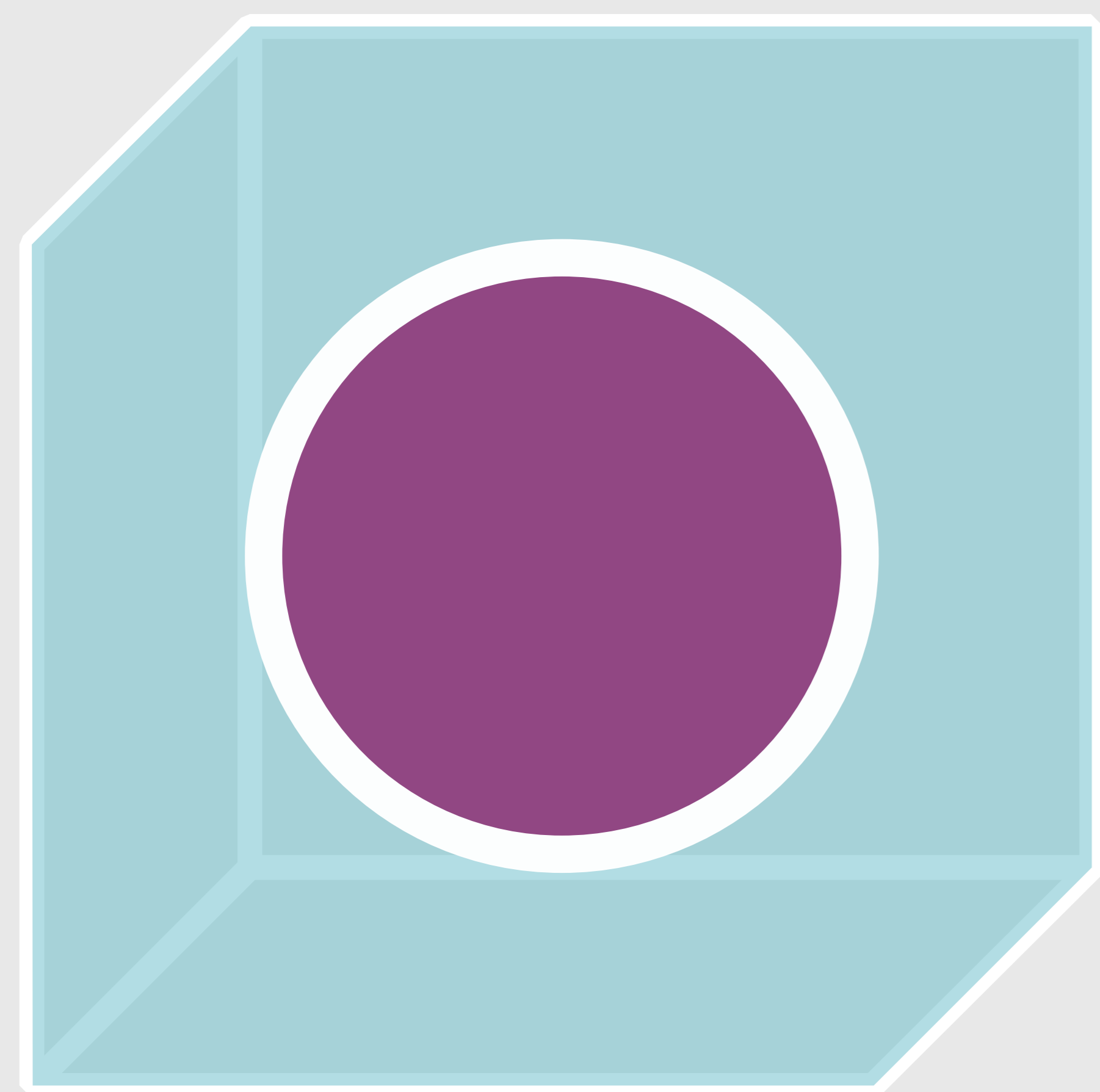


Meeting C++ 2023

Ivan Čukić [⚡ kdab.com](http://kdab.com), [cukic.co](http://cukic.co)

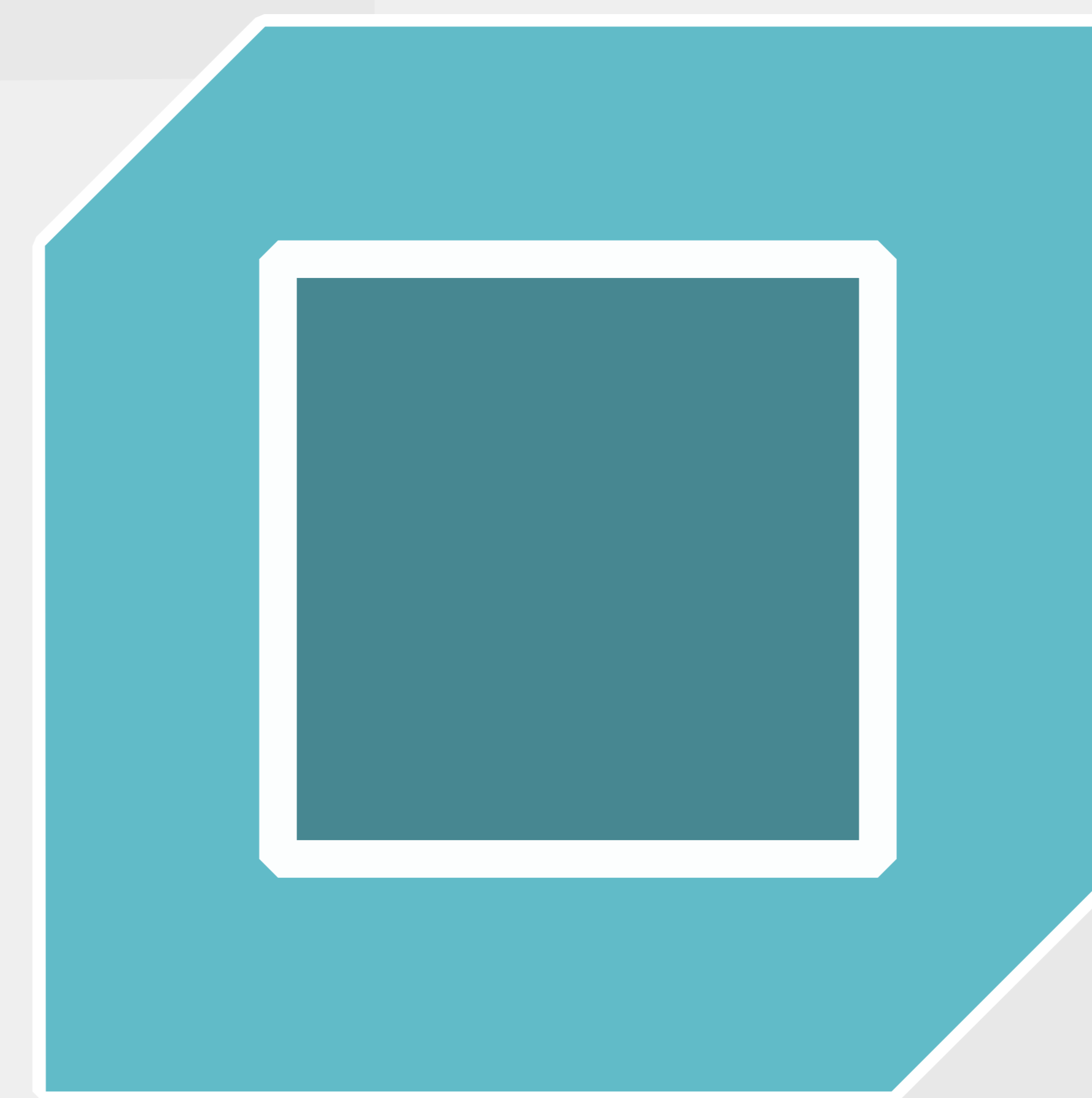
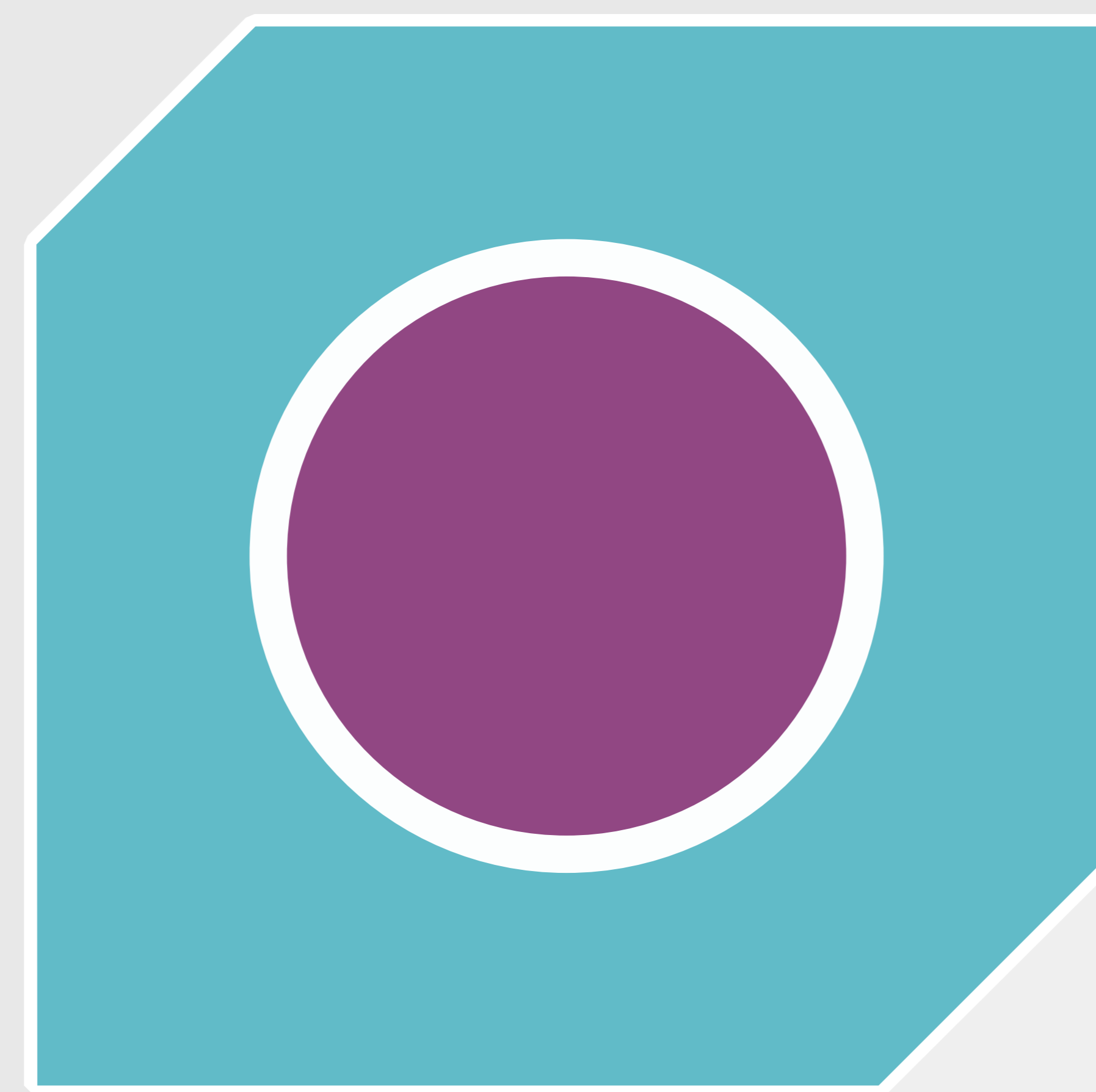
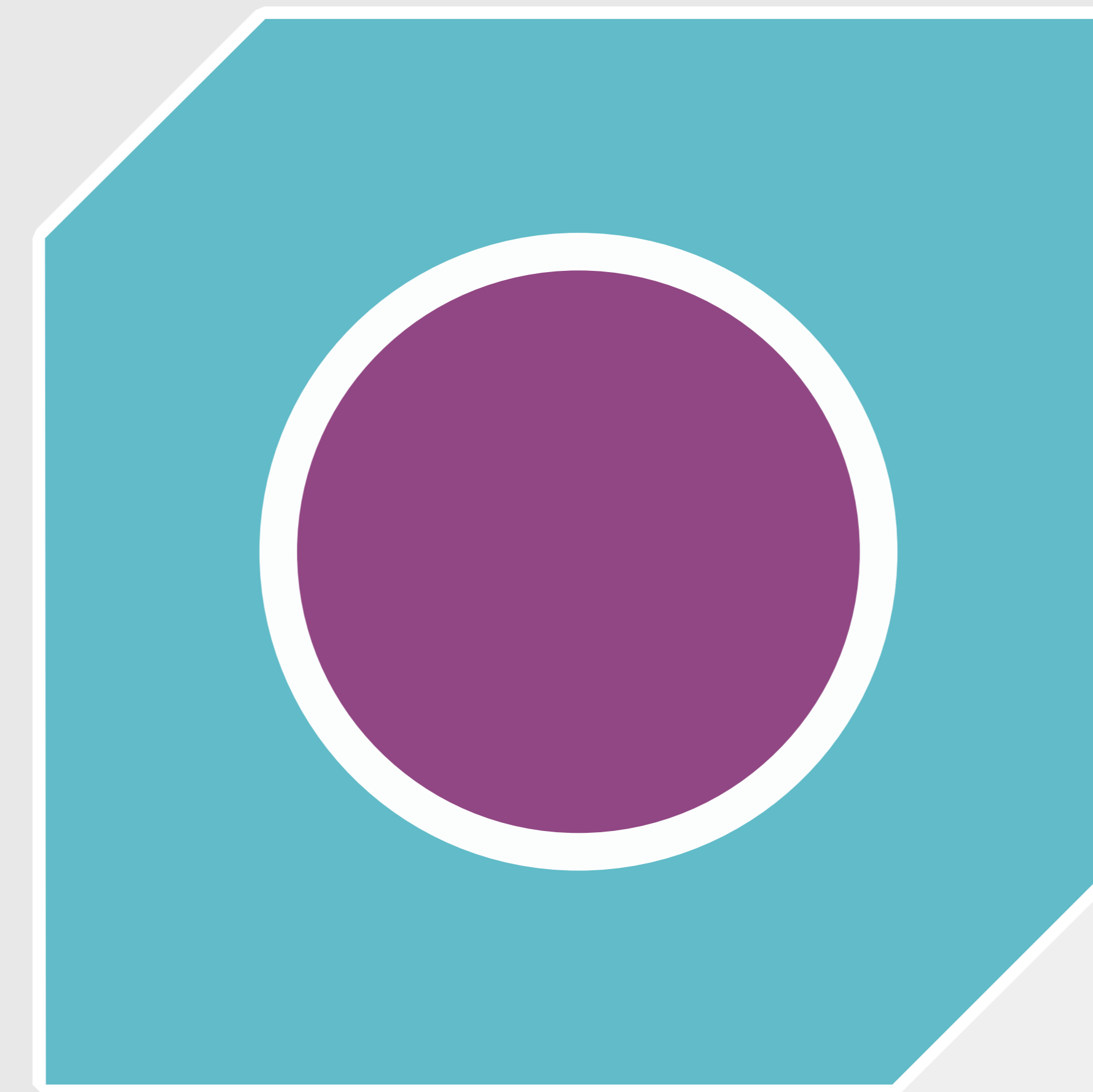
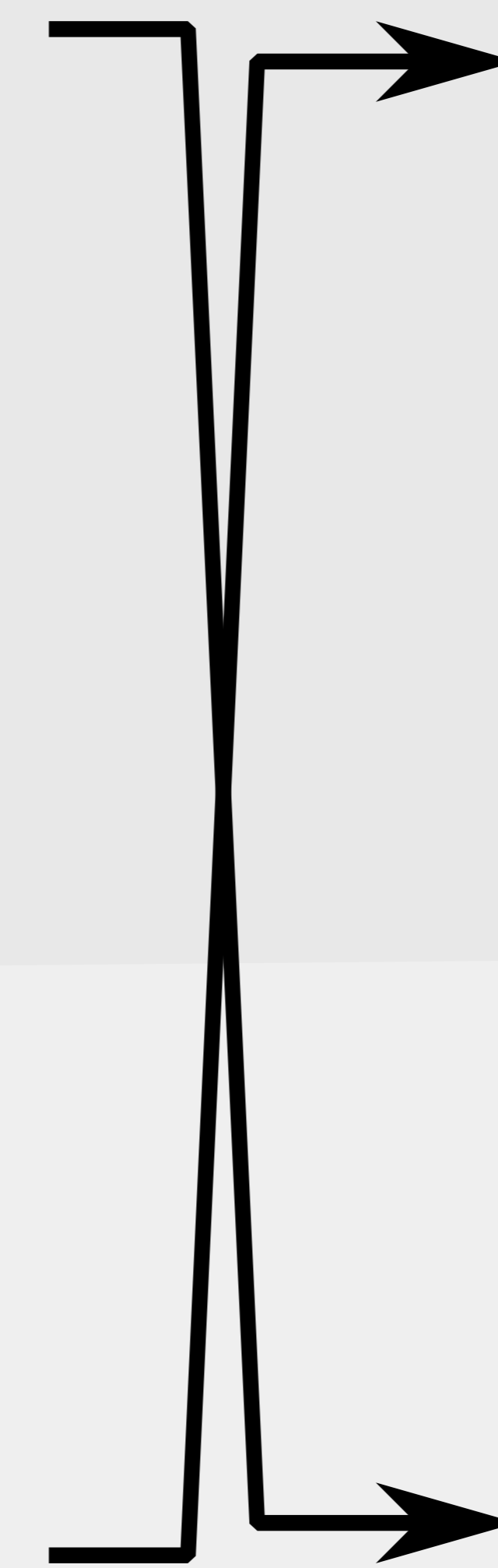
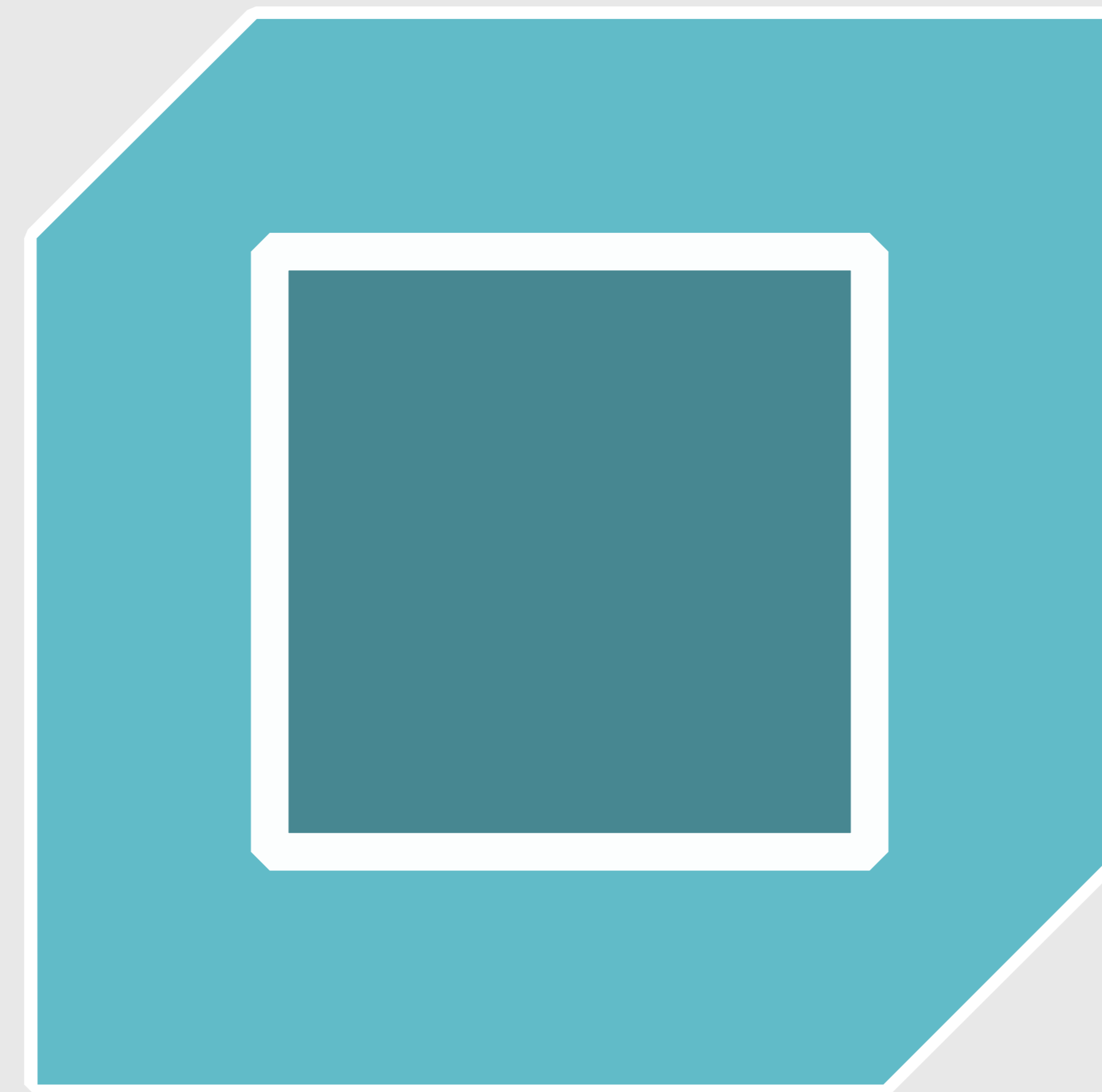
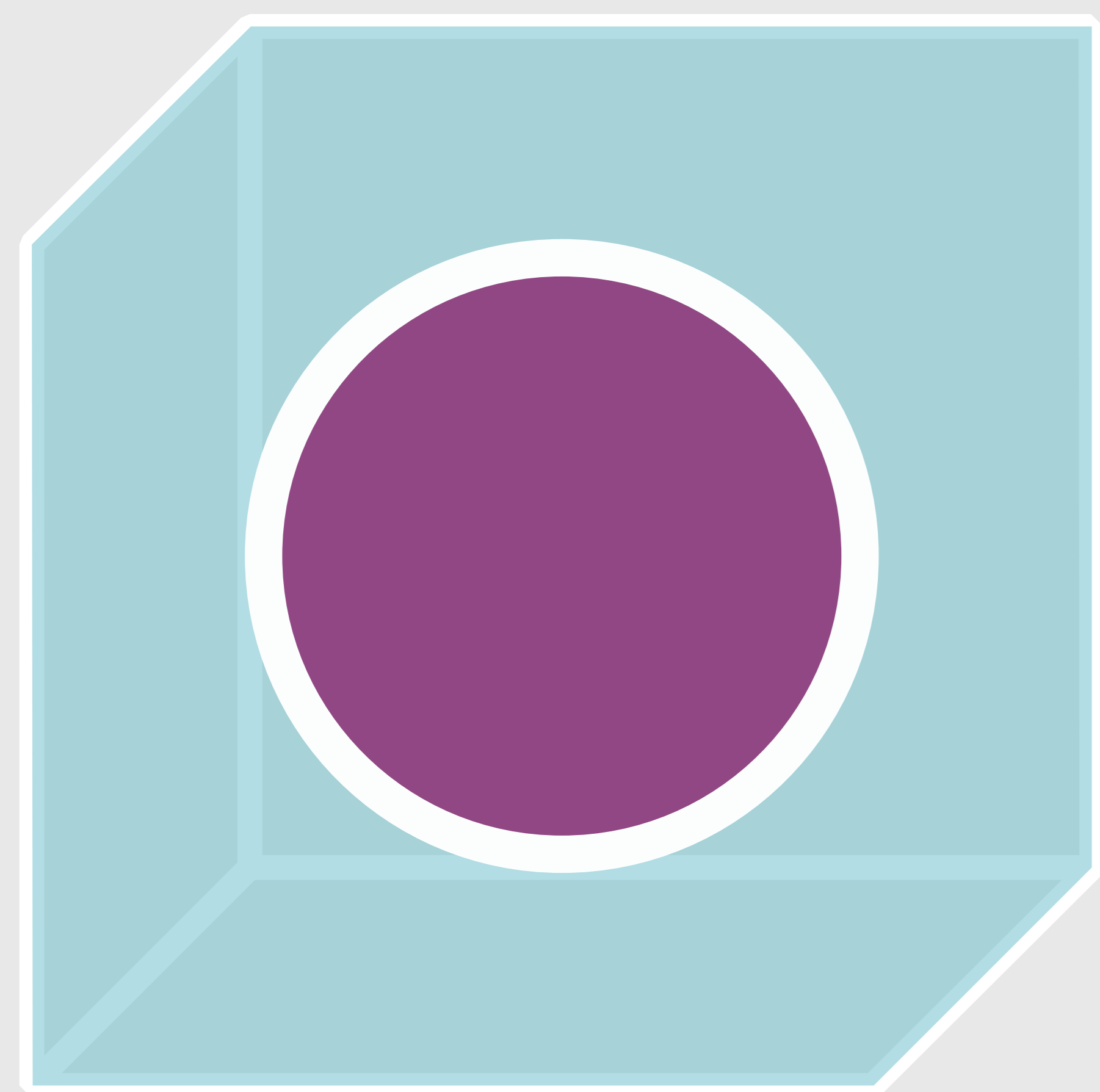
INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT



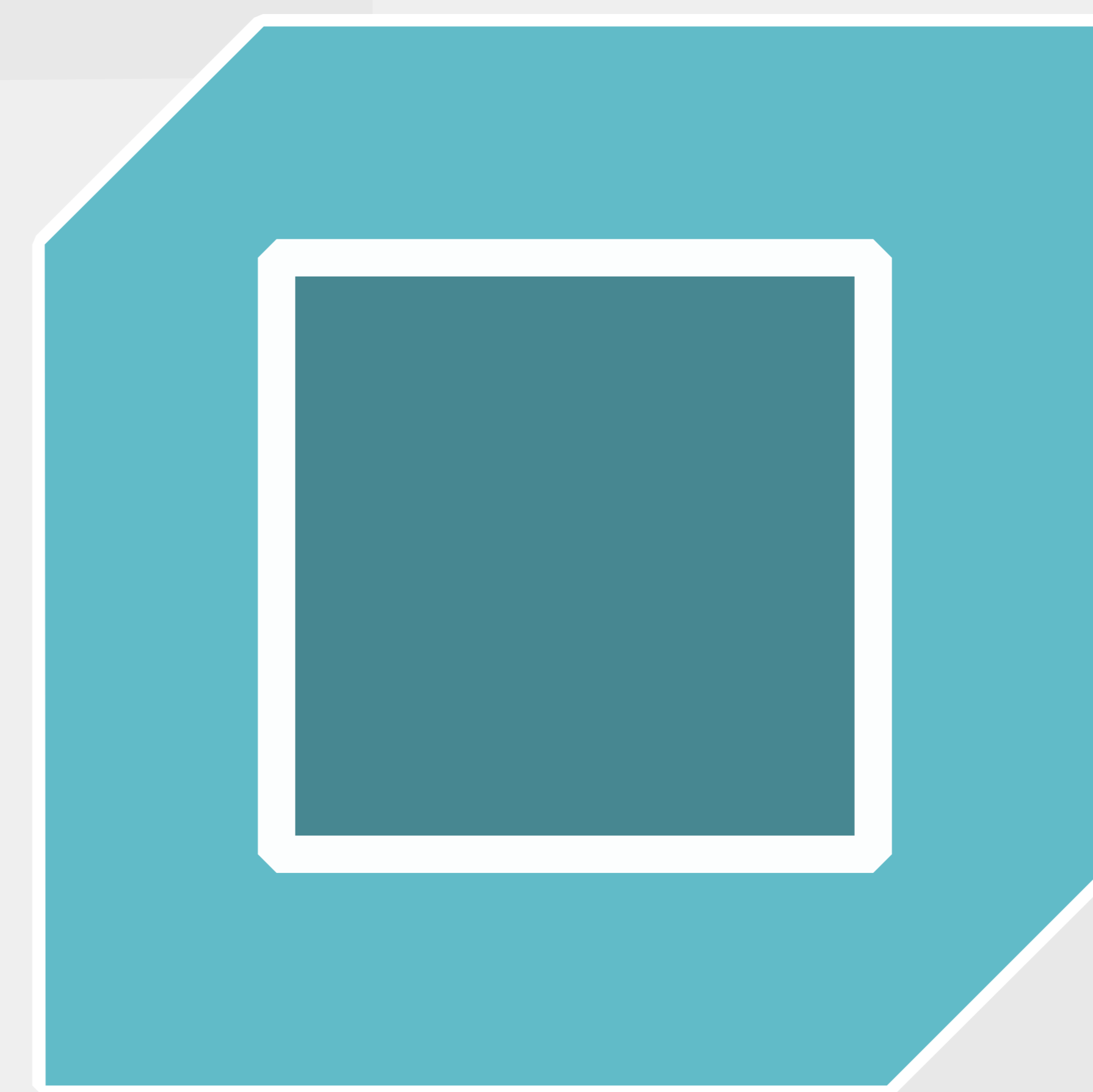
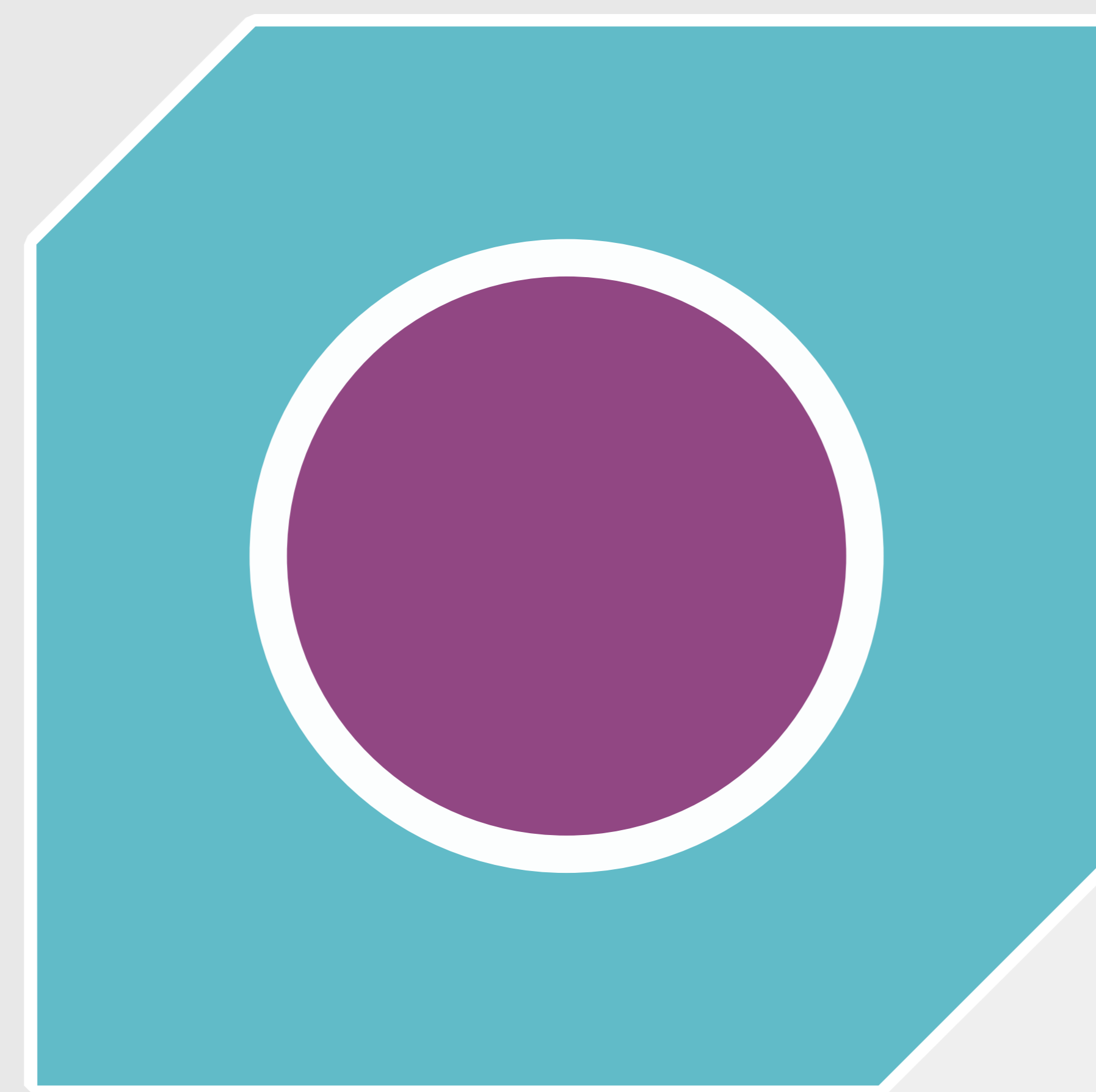
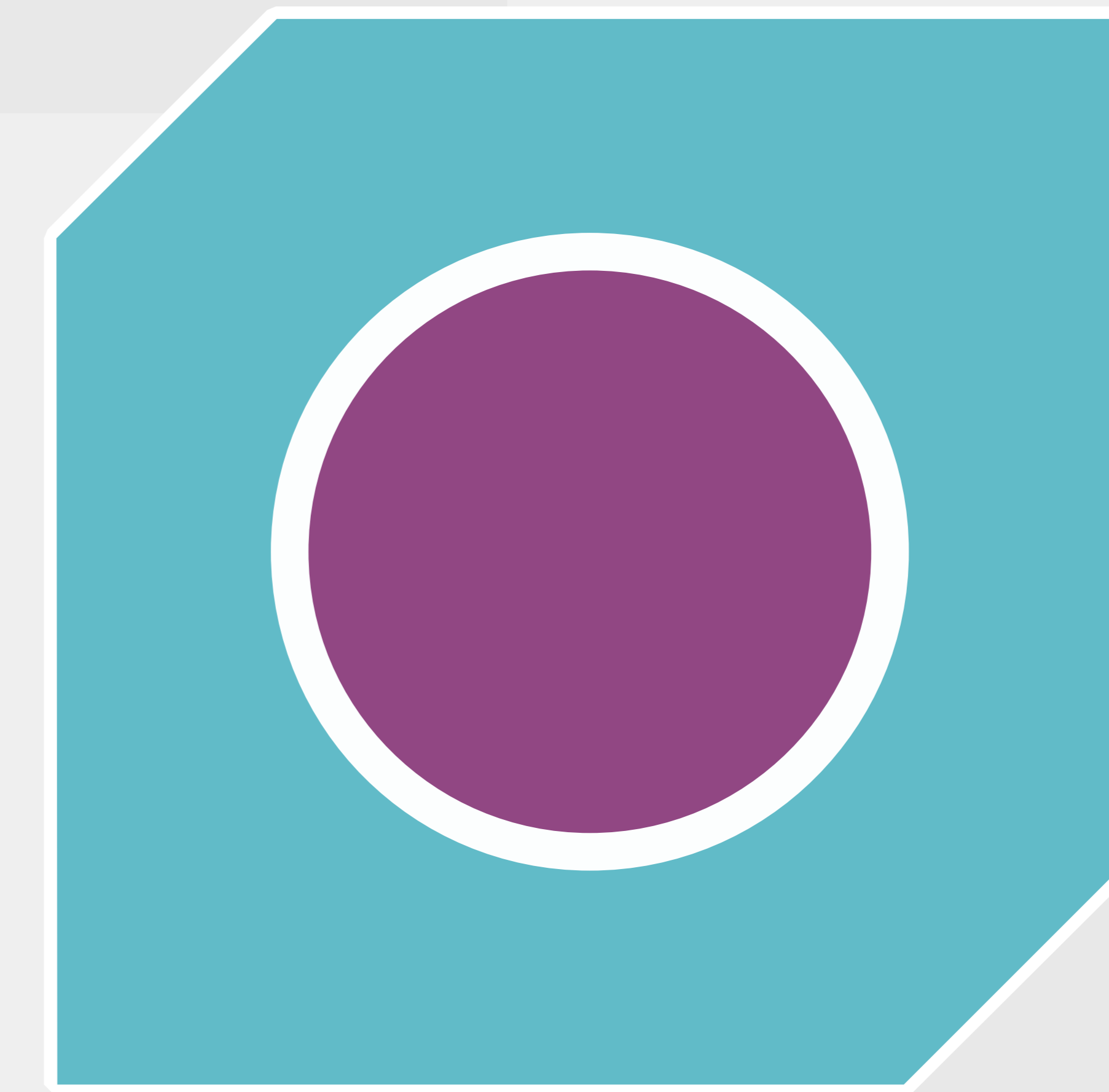
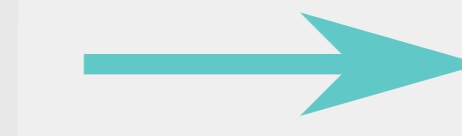
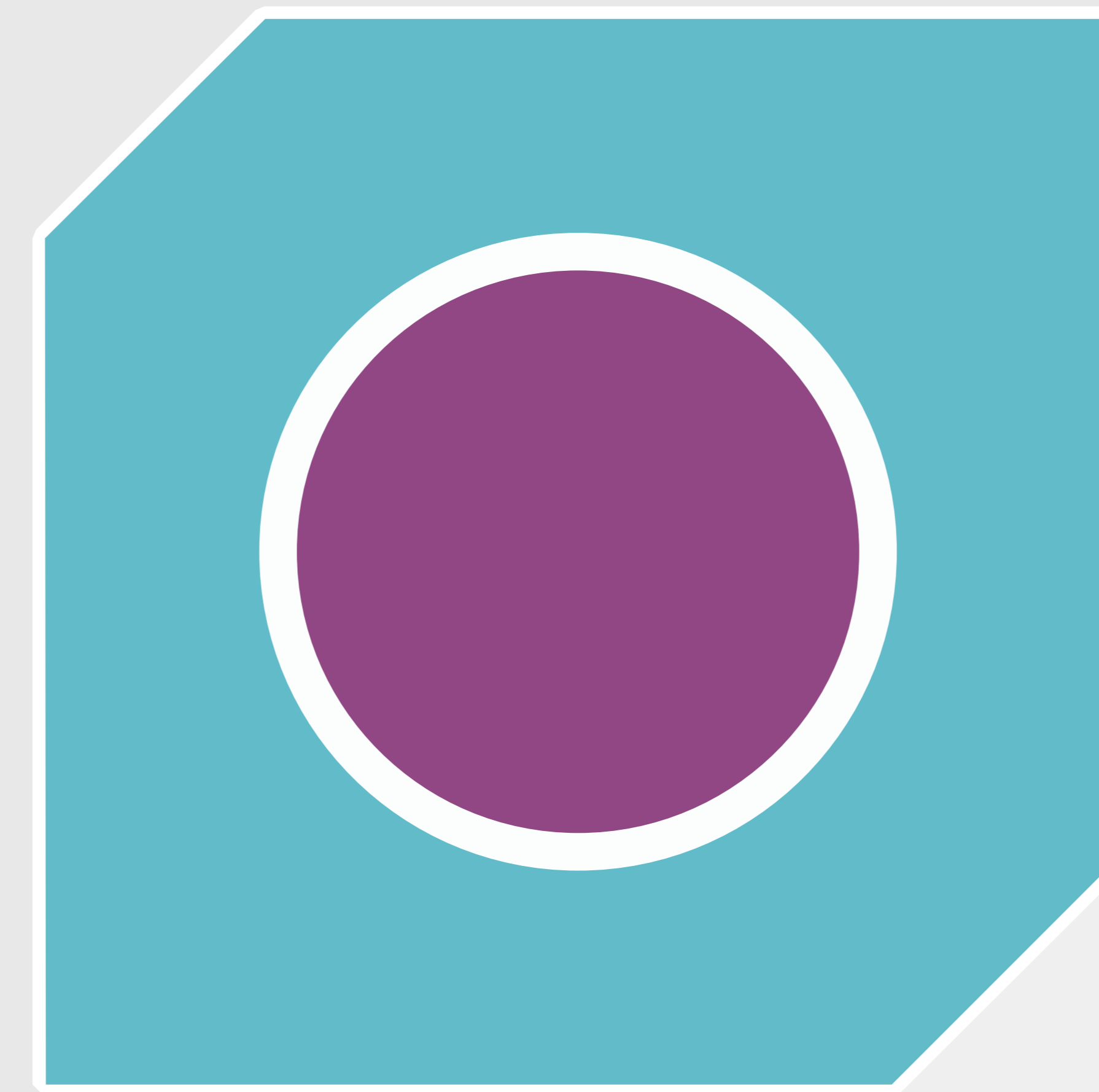
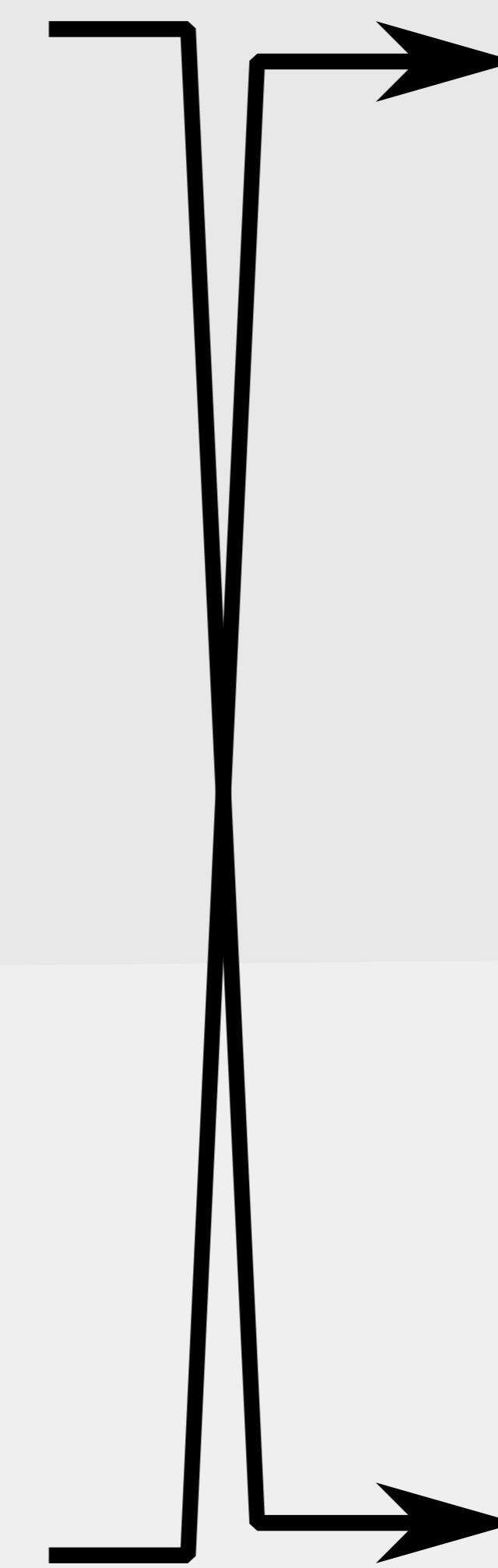
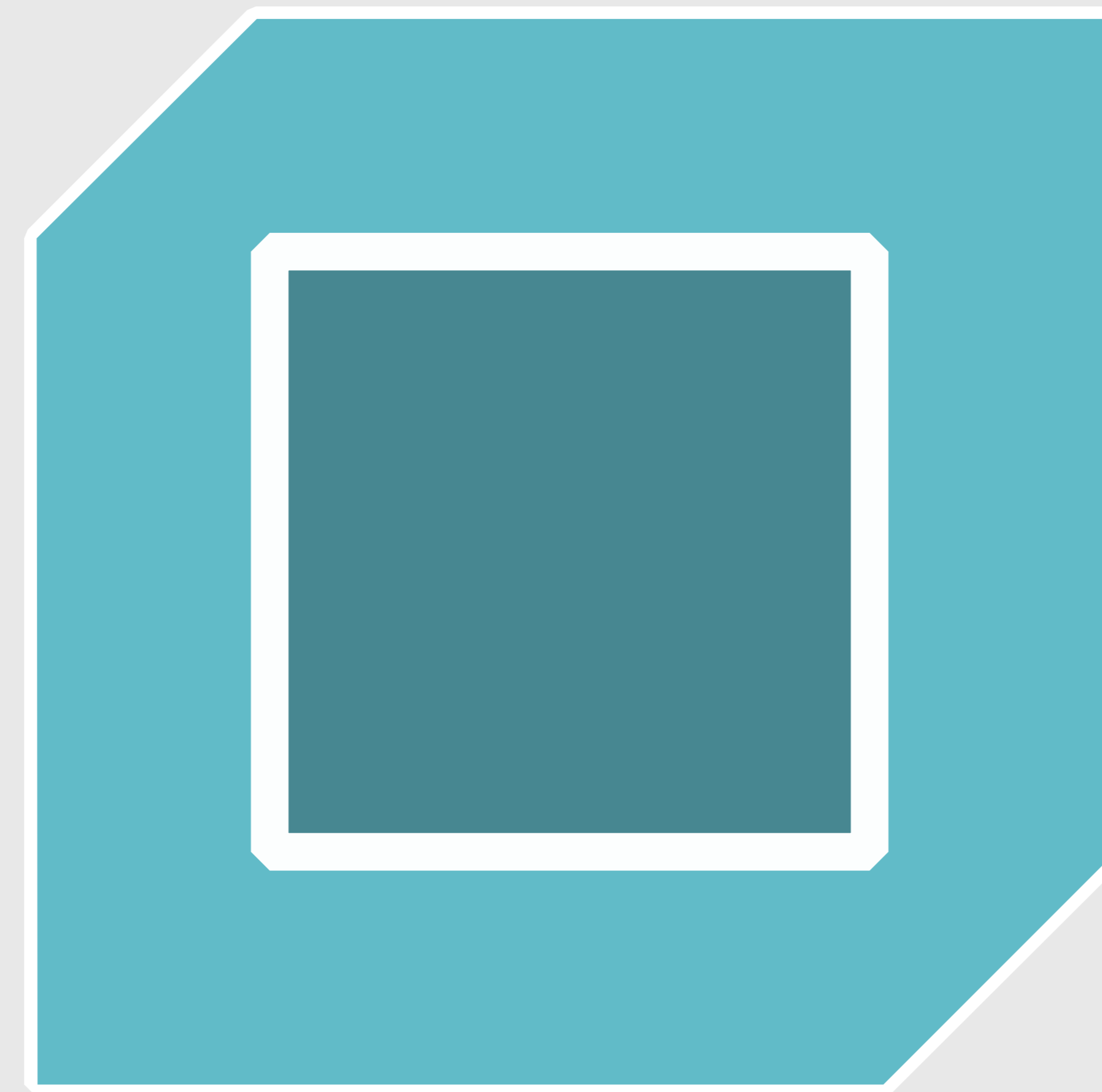
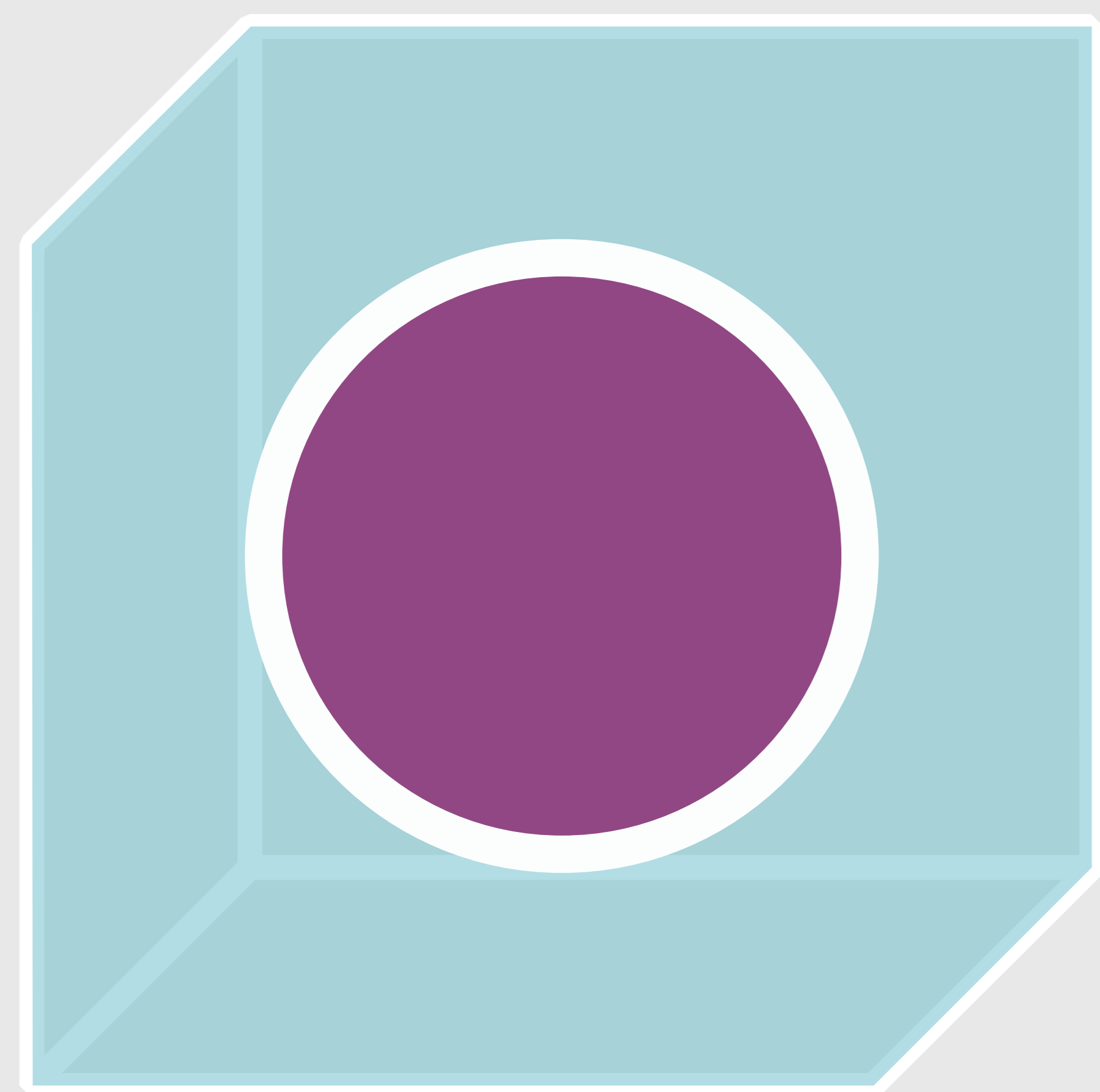
INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ASSIGNMENT



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# PREREQUISITES

Member variables can **not** be references:

```
struct person_t {  
    std::string name;  
    std::string surname;  
    std::string& quick_access_name;  
  
    // ...  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# RULE

Member variables should be value types.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# SWAPPING

- Double buffering
- Processing collections in separate threads

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# MARK AND SWEEP

```
mark(root_object);  
for (auto* object: objects) {  
    if (object->is_marked()) {  
        object.marked = false;  
    } else {  
        delete object;  
    }  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# MARK AND SWEEP

```
mark(root_object);  
for (auto* object: objects) {  
    if (object->is_marked()) {  
        object.marked = false;  
    } else {  
        garbage.push_back(object);  
    }  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# MARK AND SWEEP

```
garbage_t current_garbage;  
swap(garbage, current_garbage);  
gc.collect(current_garbage);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

INTRODUCTION  
WRAPS  
SWAPS  
**STATES**  
ERRORS  
VALUES  
SAFETY

Meeting C++ 2023

Ivan Čukić [⚡ kdab.com](http://kdab.com), [cukic.co](http://cukic.co)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES



A large fraction of the flaws in software development are due to programmers not fully understanding all the possible states their code may execute in.

– John Carmack

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
struct page_t {  
    std::string url;  
    socket_t connection;  
    bool loading;  
    std::unique_ptr<html::dom> dom;  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# STATES



"A nice bag to put things in"  
– Miloš Anđelković, Class Layout

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
struct page_t {  
    // ...  
    void render() {  
        [[assert: dom != nullptr]];  
        // ...  
    }  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
struct page_t {  
    // ...  
    void execute_event_loop() {  
        [[assert: dom != nullptr]];  
        [[assert: not loading]];  
        // ...  
    }  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
void load_page() {  
    [[assert: dom == nullptr]];  
    [[assert: not loading]];  
    [[assert: not connection]];  
  
    connection = open(url);  
    loading = true;  
  
    // ...  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
struct page_t {  
    std::string url;  
    socket_t connection;           // connected  
    bool loading;                 // false  
    std::unique_ptr<html::dom> dom; // not nullptr  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
struct initial_t { std::string url; };
```

```
struct loading_t {  
    immutable<socket_t> connection;  
    immutable<std::unique_ptr<html::dom>> dom;  
};
```

```
struct loaded_t {  
    immutable<std::unique_ptr<html::dom>> dom;  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
using page_t = std::variant<  
    initial_t,  
    loading_t,  
    loaded_t>;
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
std::visit(  
    overloaded {  
        [&] (const initial_t& initial) {  
            page = start_loading(initial.url);  
        },  
        // ...  
    }, page);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

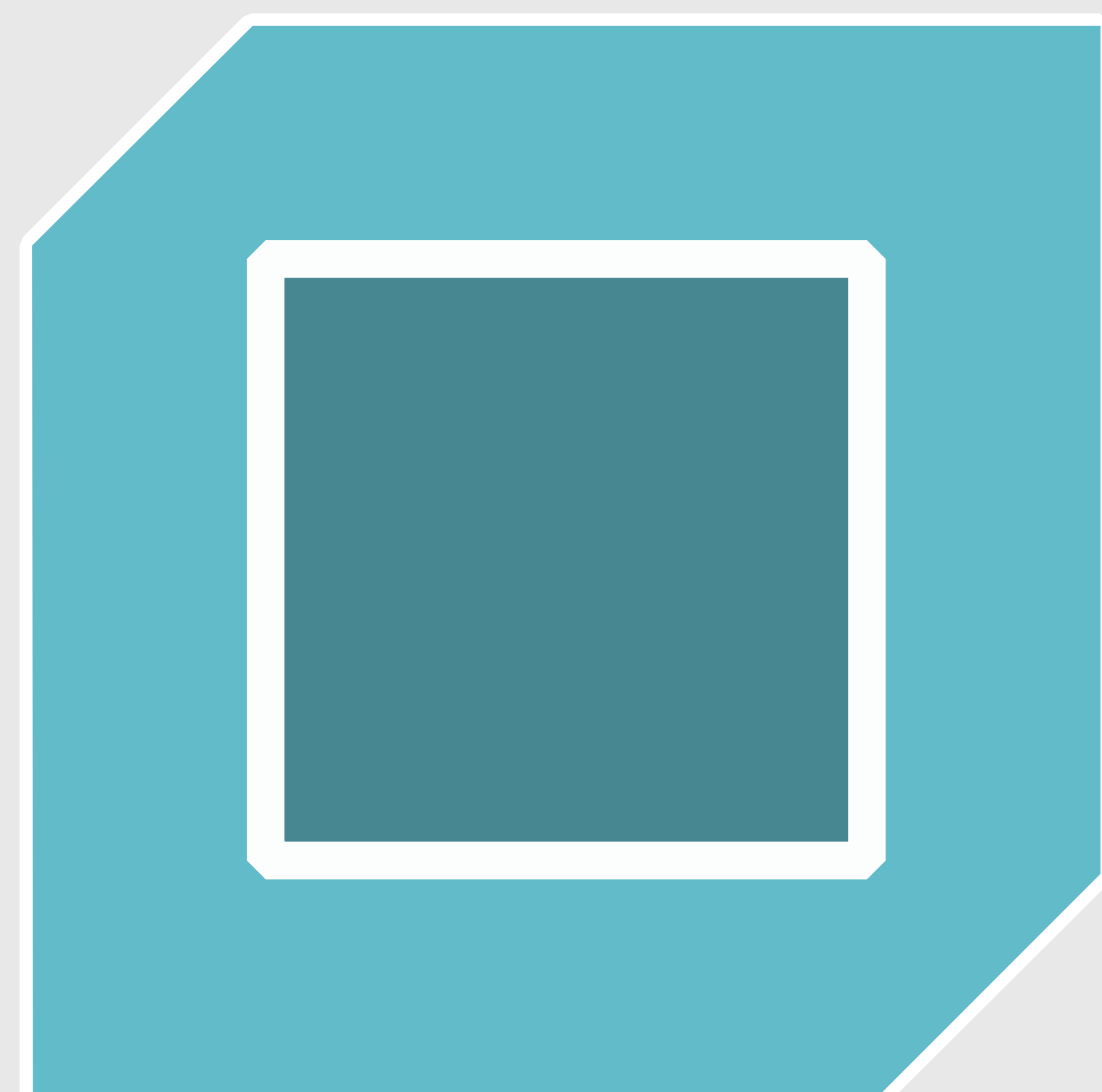


# STATES

```
if (auto* loading = std::get_if<loading_t>(&page)) {  
    page = loaded_t{ std::move(loading.dom) };  
}
```

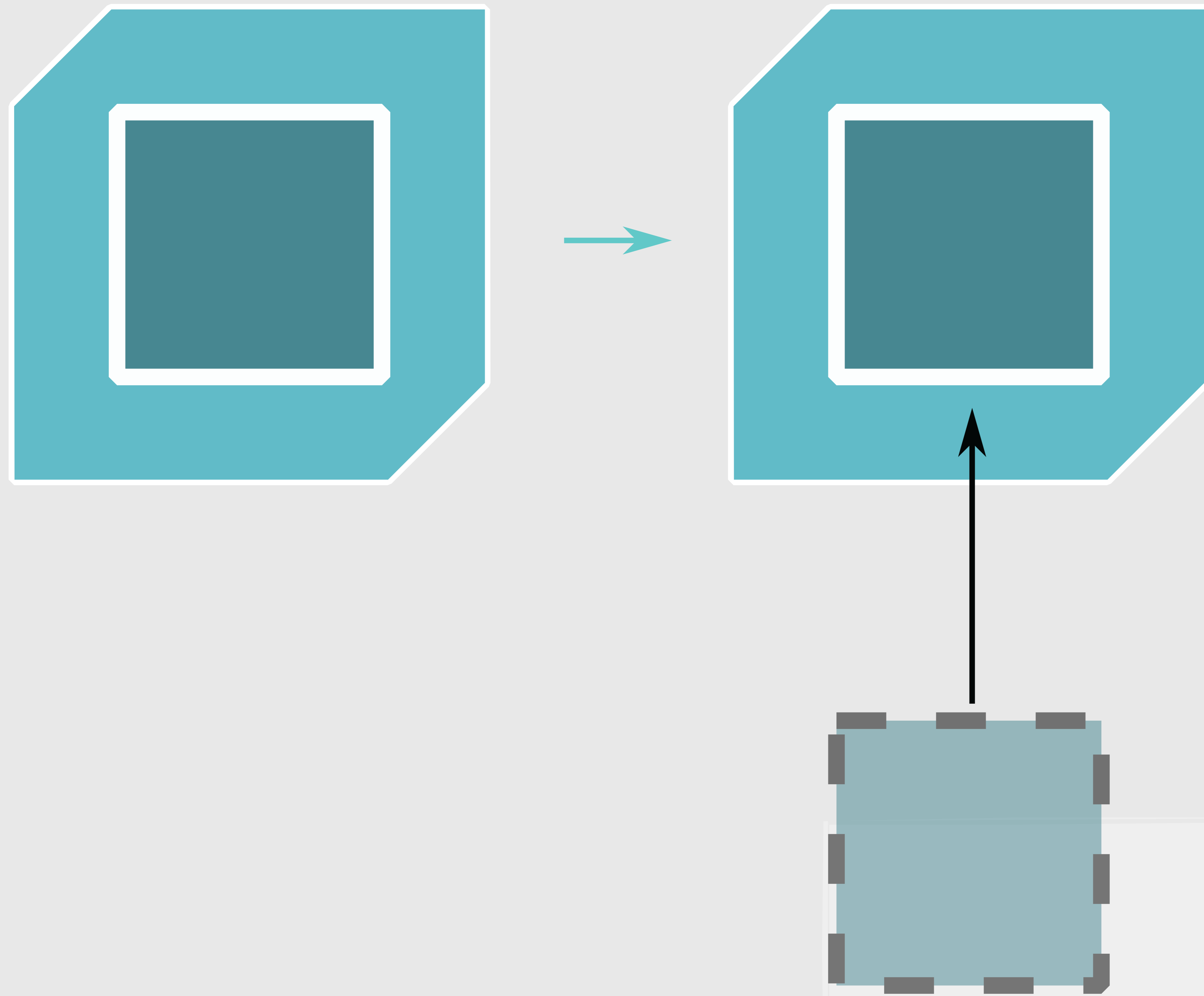
INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES



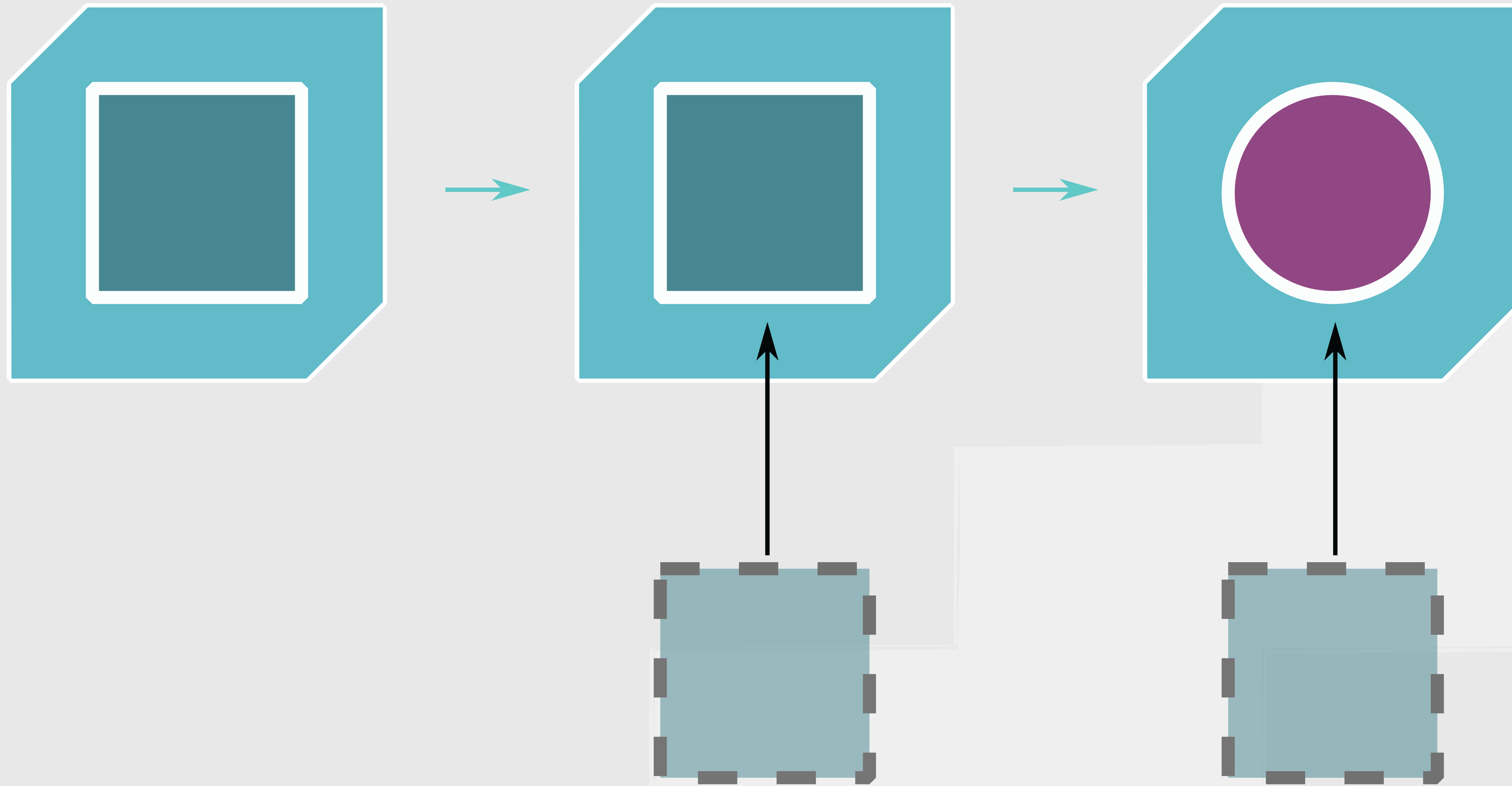
INTRODUCTION  
WRAPS  
SWAPS  
**STATES**  
ERRORS  
VALUES  
SAFETY

# STATES



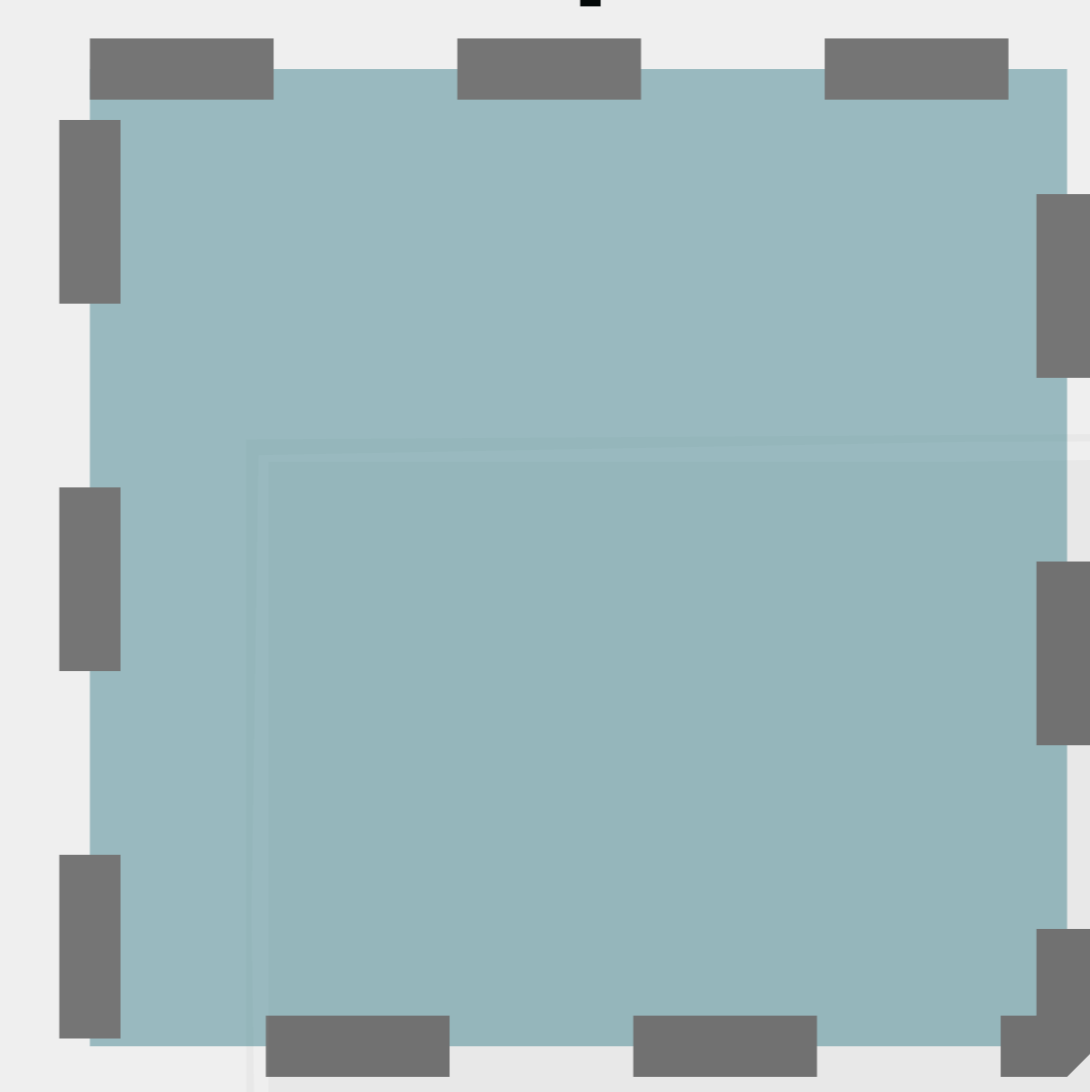
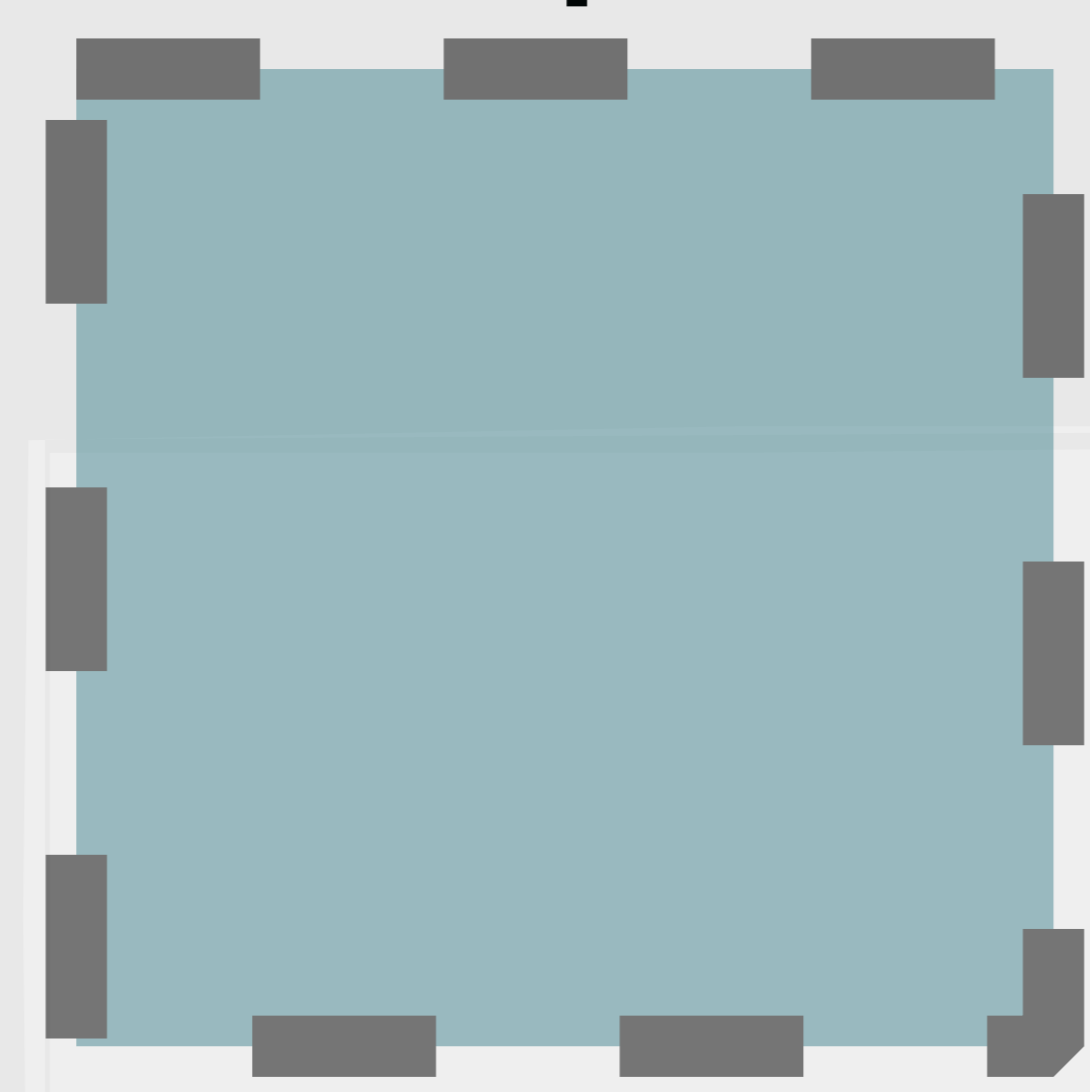
INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES



- INTRODUCTION
- WRAPS
- SWAPS
- STATES
- ERRORS
- VALUES
- SAFETY

# STATES



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
page_t new_page;
```

```
// generate the new state ...
```

```
std::swap(page, new_page);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
page_t new_page;
```

```
if (auto* loading = std::get_if<loading_t>(&page)) {  
    new_page = loaded_t{ std::move(loading.dom) };  
}
```

```
std::swap(page, new_page);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# STATES

```
std::optional<page_t> new_page;
```

```
if (auto* loading = std::get_if<loading_t>(&page)) {  
    new_page = loaded_t{ std::move(loading.dom) };  
}
```

```
if (new_page) std::swap(page, *new_page);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

Meeting C++ 2023

Ivan Čukić [⚡ kdab.com](http://kdab.com), [cukic.co](http://cukic.co)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

- NaN, -1, ...
- exceptions
- success flags

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

```
std::expected<Result, Error>  
  .transform  
  .and_then  
  .or_else
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

```
my::expected<std::string, err_t> get_email_link() {  
    std::string input = co_await get_input();  
    auto email = co_await parse_email(input);  
    auto html = format_link(email->user, email);  
    co_return html;  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ERRORS



C++ error handling, let's abuse the `co_await` operator

Antoine Morrier

<https://cpp-rendering.io/>

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# MESS IN THE KITCHEN

- Making lunch
- Cleaning up
- Burnt lunch

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# MESS IN THE KITCHEN

Forget about making lunch,  
and destroy the kitchen.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

- Try to find the **fix**
- **Continue** with the process

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

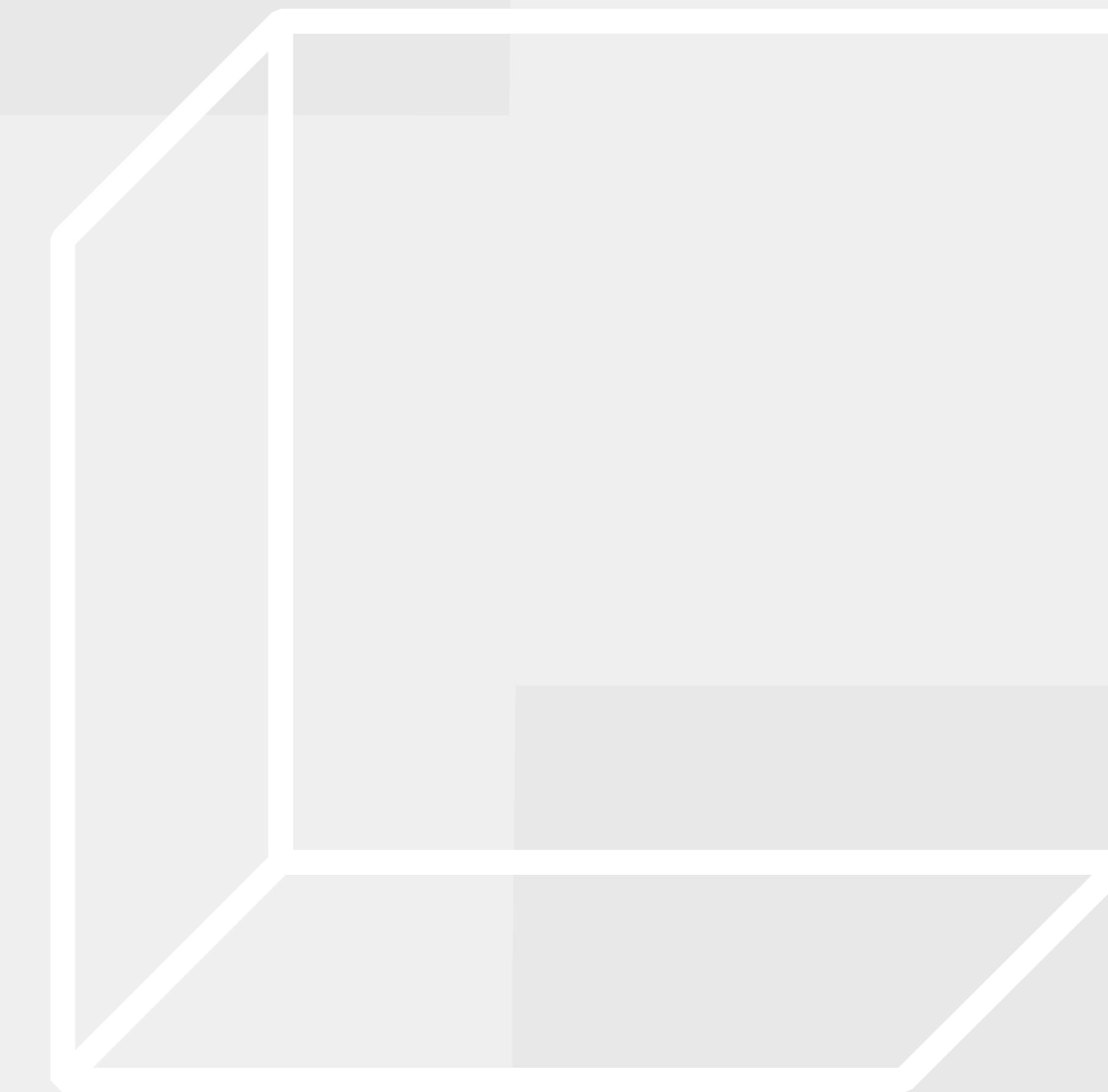
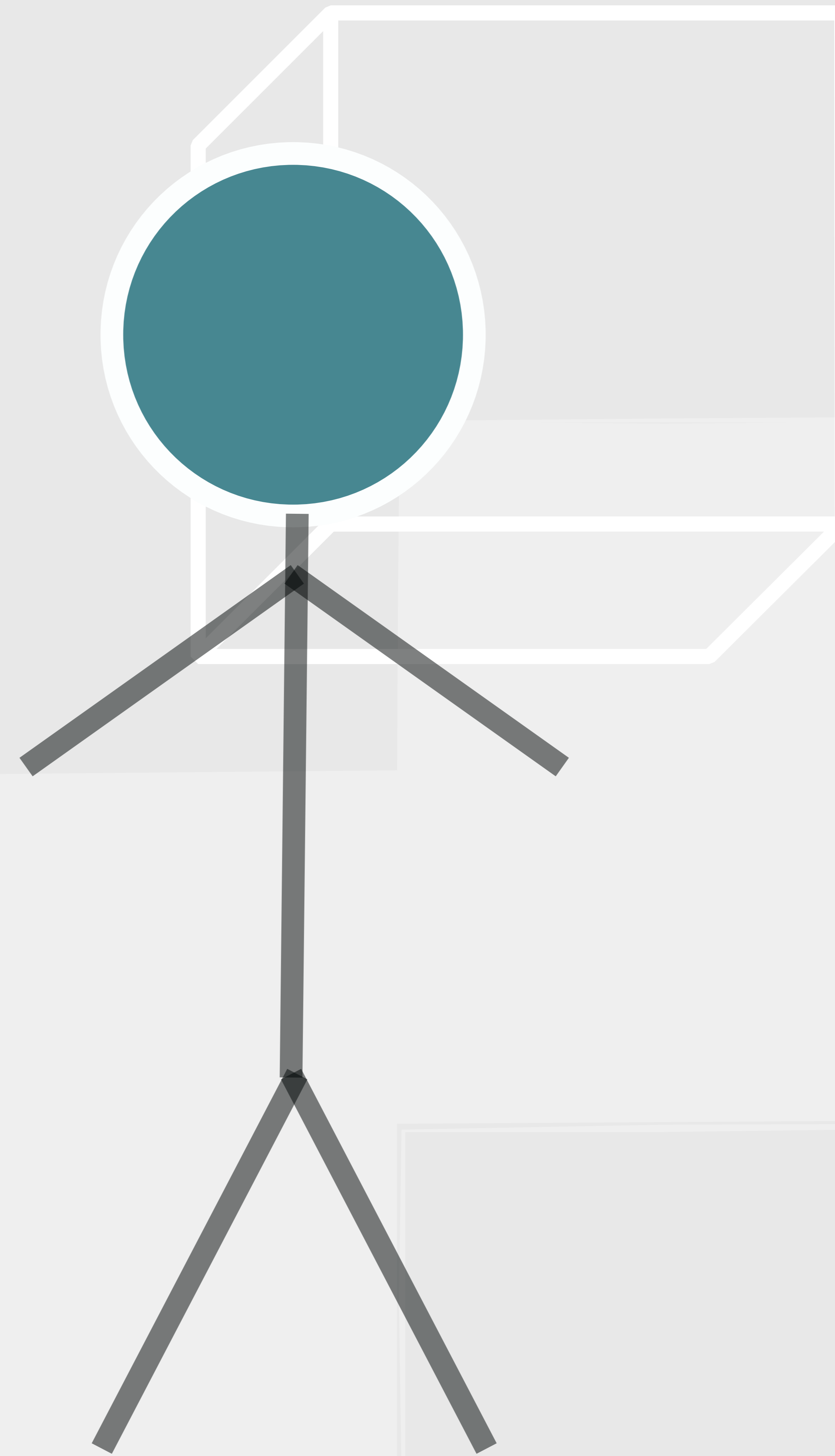
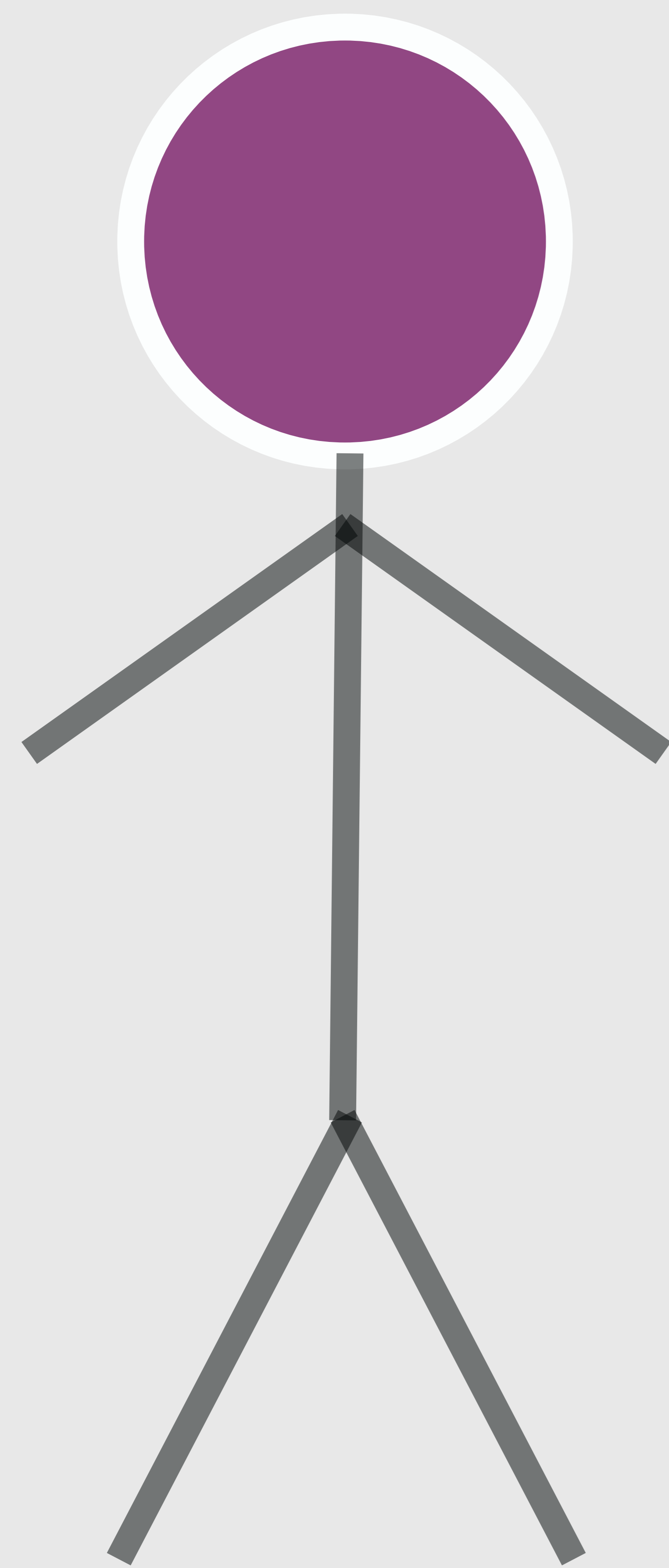


# ERRORS

- Fixing, or choosing a fix, is the job of the **caller**
- Continuation is the job of the **calee**

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS



To and fro  
Stop and go  
That's what makes the world go round

The Sword in the Stone, (C) Disney

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

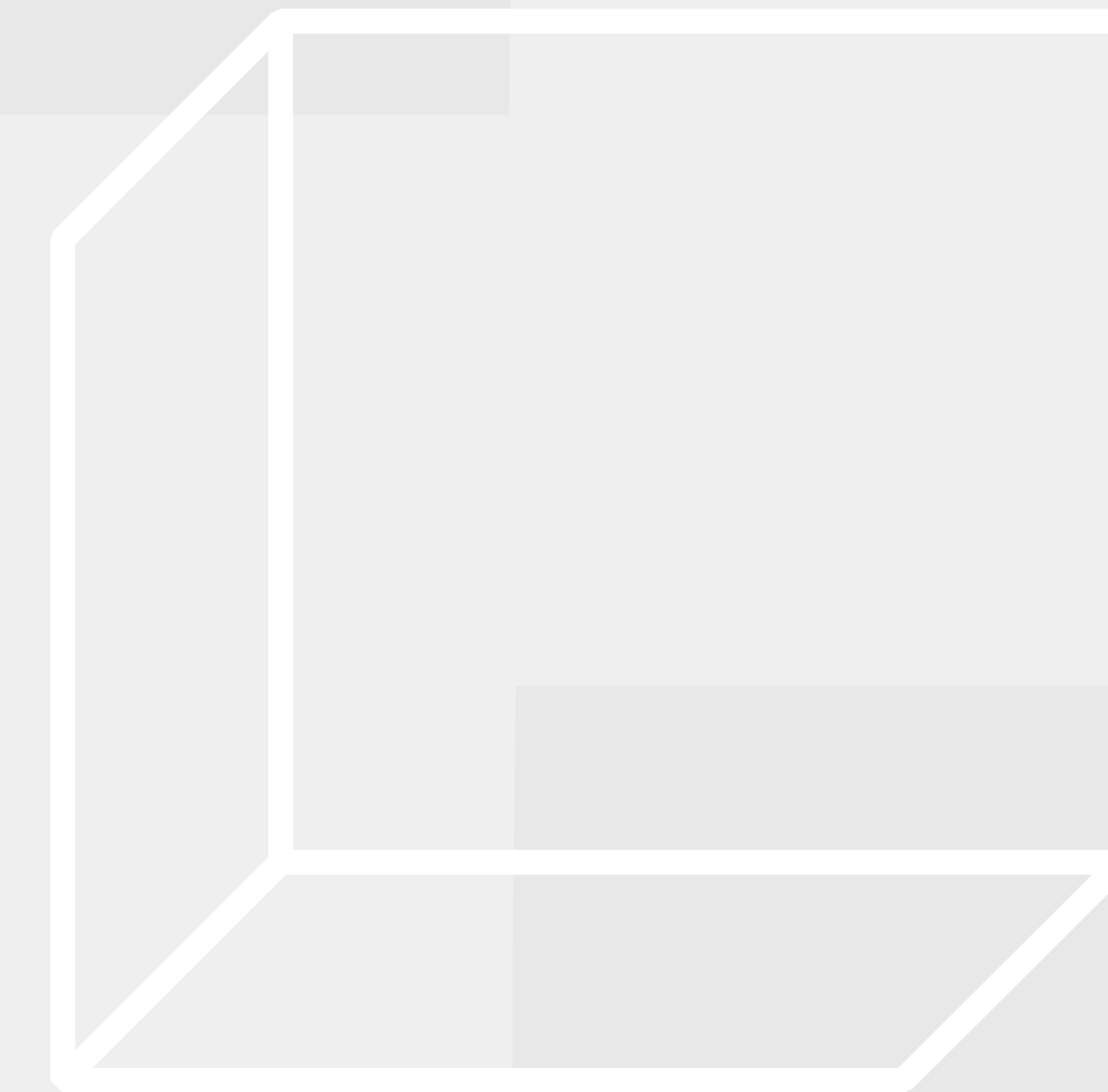
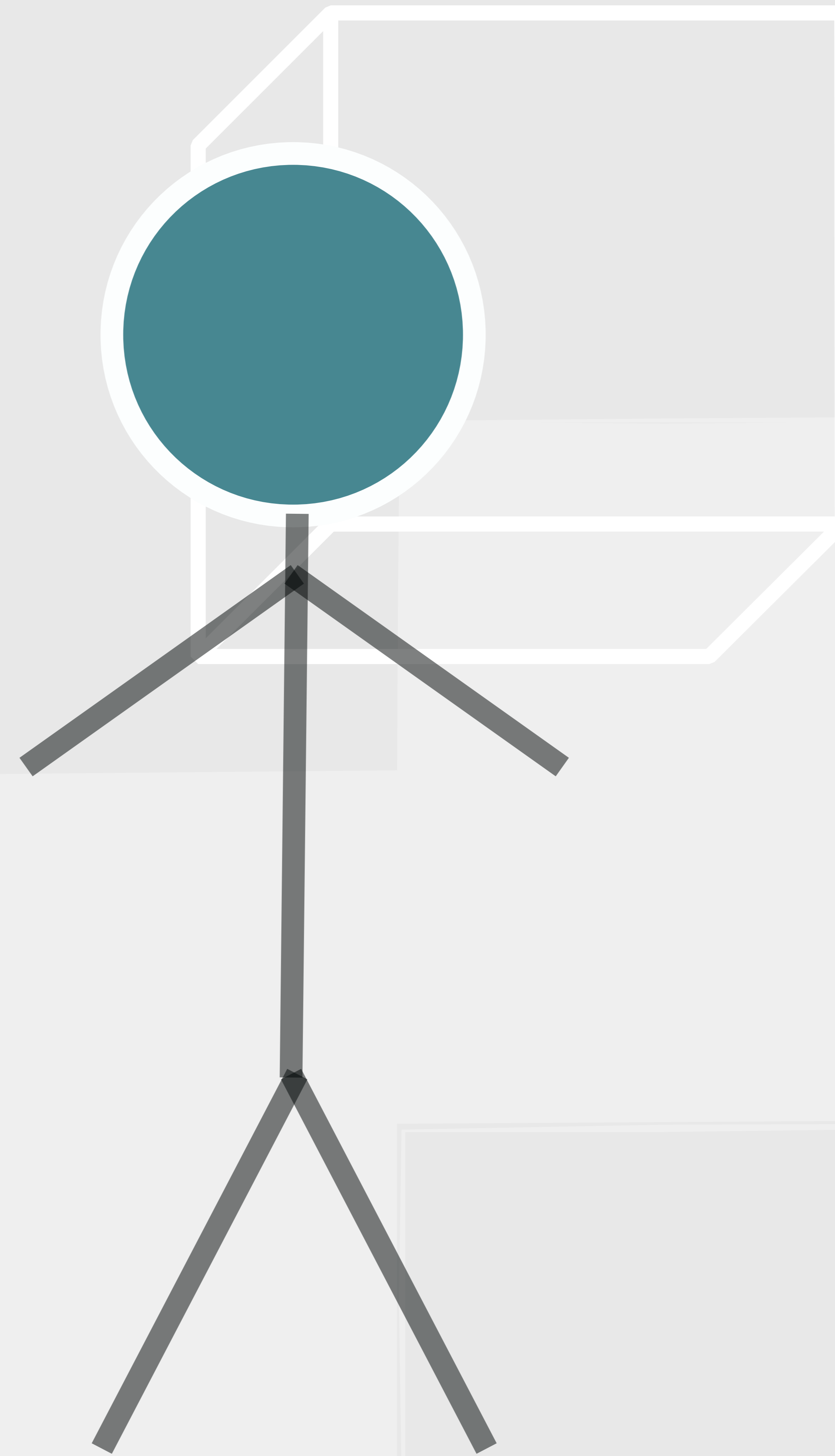
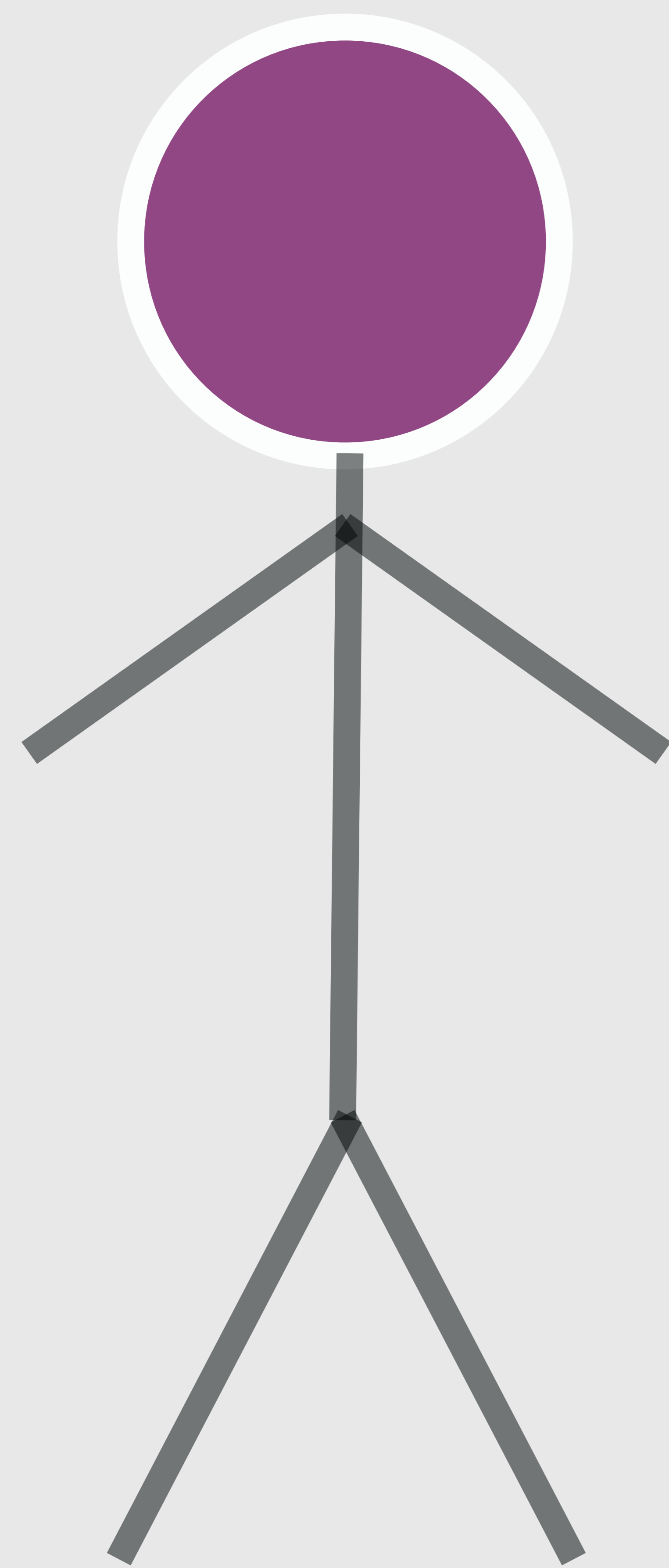
# ERRORS

```
enum class error_t {  
    number_parsing_error,  
    invalid_type_error  
};
```

```
enum class fix_t {  
    use_zero,  
    skip_line  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ERRORS



INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

```
using message_t = std::variant<  
    error_t, fix_t>;
```

```
template<typename Result>  
using result_t = std::expected<  
    Result, message_t*>;
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ERRORS

```
std::generator<result_t<int>> parse()  
{  
    message_t message;  
  
    // an error occurred  
    message = error_t::invalid_type_error;  
    co_yield std::unexpected{  
        std::addressof(message)};  
  
    // ...  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# ERRORS

```
while (true) {  
    auto result = co_await parse();  
  
    if (result) // ...  
  
    auto* message = result.error();  
    // propose a fix  
    (*message) = skip_line;  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# ERRORS

```
std::generator<result_t<int>> parse()  
{  
    // ...  
  
    // if no error  
    co_yield 42;  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

# ERRORS

Allows to continue the process in presence of errors.

Still, if we can not handle a specific error, we can still bail out.

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
**ERRORS**  
VALUES  
SAFETY

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
**VALUES**  
SAFETY

Meeting C++ 2023

Ivan Čukić [⚡ kdab.com](http://kdab.com), [cukic.co](http://cukic.co)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES



Regular Revisited

Victor Ciura

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

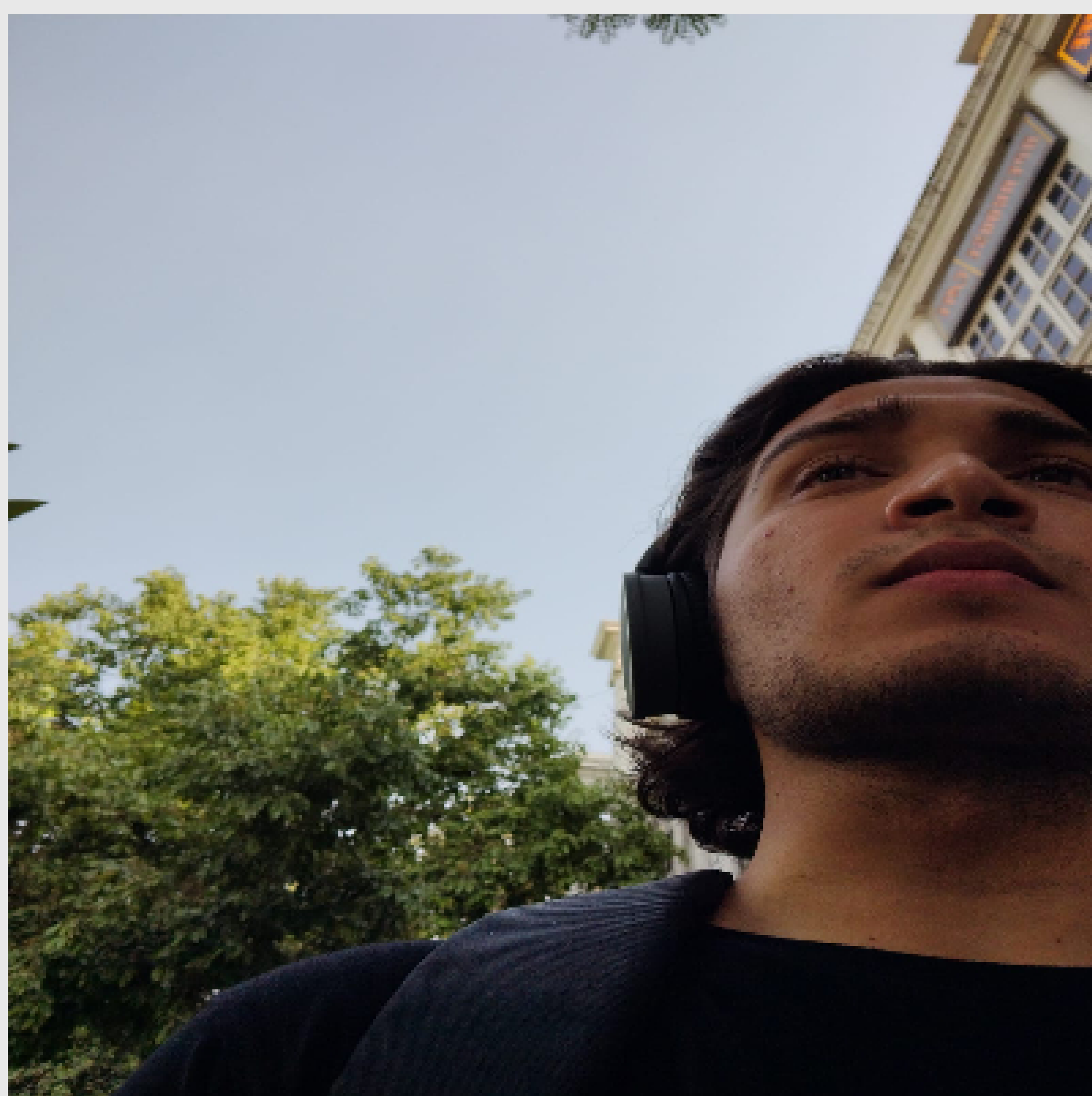
# VALUES

- Values and variables
- Value assignment semantics

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

- `std::unique_ptr<const something>`
- `std::shared_ptr<const something>`

VALUE



# Optimizing Multithreaded Performance

Shivam Kunwar

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

- Construction, destruction
- Copying, moving
- Assignment
- Comparison
- Always valid

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# VALUES

```
class page_t {  
private:  
    class page_private_t;  
    std::unique_ptr<page_private_t> m_pimpl;  
  
public:  
    // ...  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

- Invalid state(s)
- Shallow const
- Copying disabled
- Pointer redirection

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
page_t()  
    : m_pimpl(std::make_unique  
              <page_private_t>(some, arguments))  
{}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
const& page_private_t pimpl() const;  
page_private_t& pimpl();
```

Note: Or use `std::experimental::propagate_const`

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
page_t(const page_t& original)
    : m_pimpl(std::make_unique
              <page_private_t>(original.pimpl()))
{
}
```

```
// and copy-and-swap for operator=
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
std::shared_ptr<const page_private_t> m_pimpl;
```

COW?

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
void some_function(const page_t& original_page) {  
    page_t page = original_page;  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
class page_t {  
    page_t transformed(auto fn) const {  
        // ...  
    }  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# VALUES

```
template<typename T>
class heap_value {
private:
    std::unique_ptr<T> _data;

public:
    template<typename... Args> requires (... )
    heap_value(Args&& args...)
        : _data(std::make_unique<T>
                (std::forward<Args>(args)...))
    {}

    // ...
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
template<typename T>
class heap_value {
    // ...

    const T& operator*() const;
    const T* operator->() const;
    T& operator*();
    T* operator->();

    // swap, copy and move operations...
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# TYPE ERASURE



Breaking Dependencies:

Type Erasure - The Implementation Details

Klaus Iglberger

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# TYPE ERASURE



Dyno: Runtime polymorphism done right

<https://github.com/ldionne/dyno>

Louis Dionne

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
**SAFETY**

Meeting C++ 2023

Ivan Čukić [⚡ kdab.com](http://kdab.com), [cukic.co](http://cukic.co)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
**SAFETY**

# IDIOMS

```
for (const auto& person: persons) {  
    person ... // idiomatic  
    persons ... // not idiomatic  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

```
for (auto index = 0U;  
    index != persons.size();  
    ++index) {  
  
    persons[index] ... // idiomatic  
    persons ... // not idiomatic  
    index ... // const access is idiomatic  
  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

```
std::vector<int> result;  
result.resize(words.size());  
std::transform(words.cbegin(), words.cend(),  
               result.begin(),  
               parse_int);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# IDIOMS

- Previous knowledge
- Comprehension
- **Safety**

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# SAFETY



JF Bastien

Safety and Security: The Future of C++

CppNow 23

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# SAFETY

```
catch (...) {  
    log("Error");  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

```
for (auto index = 0U;  
     index != persons.size();  
     ++index) {  
    persons[index]  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

```
std::vector<int> result;  
result.resize(words.size());  
std::transform(words.cbegin(), words.cend(),  
               result.begin(), parse_int);
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LAZY LAMBDA

```
operator const value_type &() const
{
    std::call_once(m_once, [&] {
        m_data = std::invoke(m_function);
    });
    return *m_data;
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# VALUES

```
template<typename T>  
class heap_value {  
    // ...  
};
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

```
std::mutex mutex;  
  
void do_something() {  
    mutex.lock();  
  
    // ...  
  
    mutex.unlock();  
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



# IDIOMS

```
std::mutex mutex;
```

```
void do_something() {
```

```
    std::lock_guard<std::mutex> lock(mutex);
```

```
    // or unique_lock
```

```
    // ...
```

```
}
```

INTRODUCTION

WRAPS

SWAPS

STATES

ERRORS

VALUES

SAFETY

# IDIOMS

Use a recursive mutex by default?

Don't lock at all?

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# IDIOMS

`private:`

```
std::mutex mutex;
```

```
struct unique_key { std::lock_guard<std::mutex> lock; }
```

`public:`

```
unique_key take_key() { ... }
```

```
void do_something(const unique_key&) {
```

```
    // ...
```

```
}
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LANGUAGE BARRIER



Tsukiji Hongan-Ji  
(C) TokyoViews

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# LANGUAGE BARRIER

```
volatile int counter = 0;
```

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# BENEFITS

- Easier to prove \*safety
- Easier to triage bugs
- Easier to prove correctness (easier, not easy)

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# Downsides

- Development under heavy chains
- More bureaucracy
- Rewriting everything

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY

# GUIDELINES

- TDD + IDD
- Idioms that can become generic functions, should be generic functions
- Idioms that can be coded in a way that is enforced by the compiler, should be
- Other idioms, write static analysis tools

INTRODUCTION  
WRAPS  
SWAPS  
STATES  
ERRORS  
VALUES  
SAFETY



The KDAB logo is a blue speech bubble shape with the text "KDAB" in white. The "K" is stylized with a white arrow pointing to the right.

KDAB

Web: <https://kdab.com>

Mail: [ivan.cukic@kdab.com](mailto:ivan.cukic@kdab.com)

Personal

Web: <https://cukic.co>

Mail: [ivan@cukic.co](mailto:ivan@cukic.co)

Twitter: [@ivan\\_cukic](https://twitter.com/ivan_cukic)

Fediverse: [@ivan@functional.cafe](https://fedi.cukic.co/@ivan)

[cukic.co/to/fp-in-cpp](https://cukic.co/to/fp-in-cpp)

Functional Programming in C++

