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

10 YEARS

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

– Ian Anderson

Ivan Čukić kdab.com, cukic.co

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.

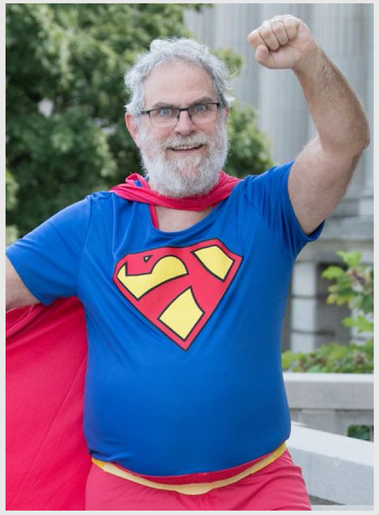
Ivan Čukić kdab.com, cukic.co

PROG C++

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.

Ivan Čukić kdab.com, cukic.co

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

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`,

  ...

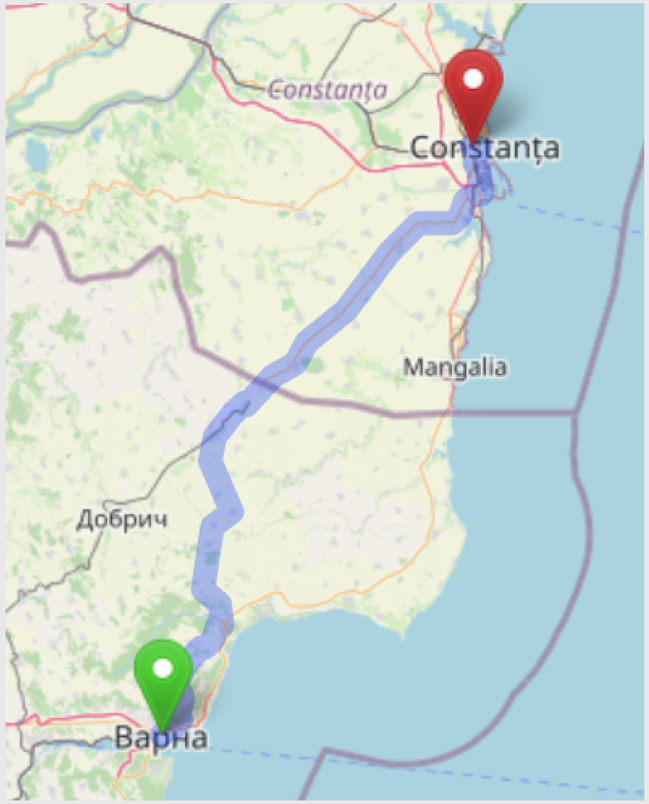
INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Meeting C++ 2023

Ivan Čukić kdab.com, cukic.co

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

ALMOST ALWAYS CONST



Constanța, Romania

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

CONST...RUCTION

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

CONST...RUCTION

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

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);
```

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);
```

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;

    // ...
```


Ivan Čukić kdab.com, cukic.co

LAZY LAMBDA

```
    // ...

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

PHASES OF CONST

- Building the value
- Using the value

Ivan Čukić kdab.com, cukic.co

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

ALWAYS CONST

Can we have a move-enabled const?

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

IMMUTABLE<T>

```
template<typename T>
class immutable {
    T m_value;

    // ...
};
```

Applied rule: Make everything const, except member variables.

Ivan Čukić kdab.com, cukic.co

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;
    // ...
};
```

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);
    }
};
```

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.

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

Ivan Čukić kdab.com, cukic.co

IMMUTABLE AND LAZY

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

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.

Ivan Čukić kdab.com, cukic.co

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, 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;
    // ...
};
```


Ivan Čukić kdab.com, cukic.co

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;
};
```

ASSIGNMENT

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

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

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);
```

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));
}
```


Ivan Čukić kdab.com, cukic.co

ASSIGNMENT

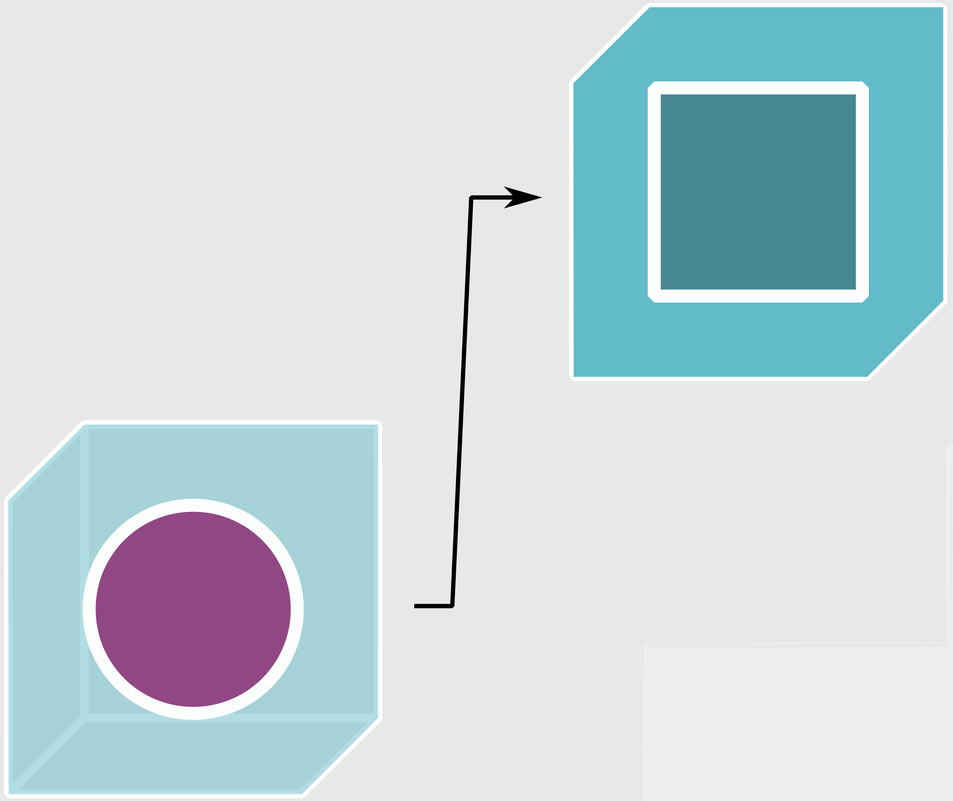
```
class person_t {
    person_t& operator=(const person_t& other) {
        auto temp = other;
        swap(*this, temp);
        return *this;
    }
};
```

ASSIGNMENT

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

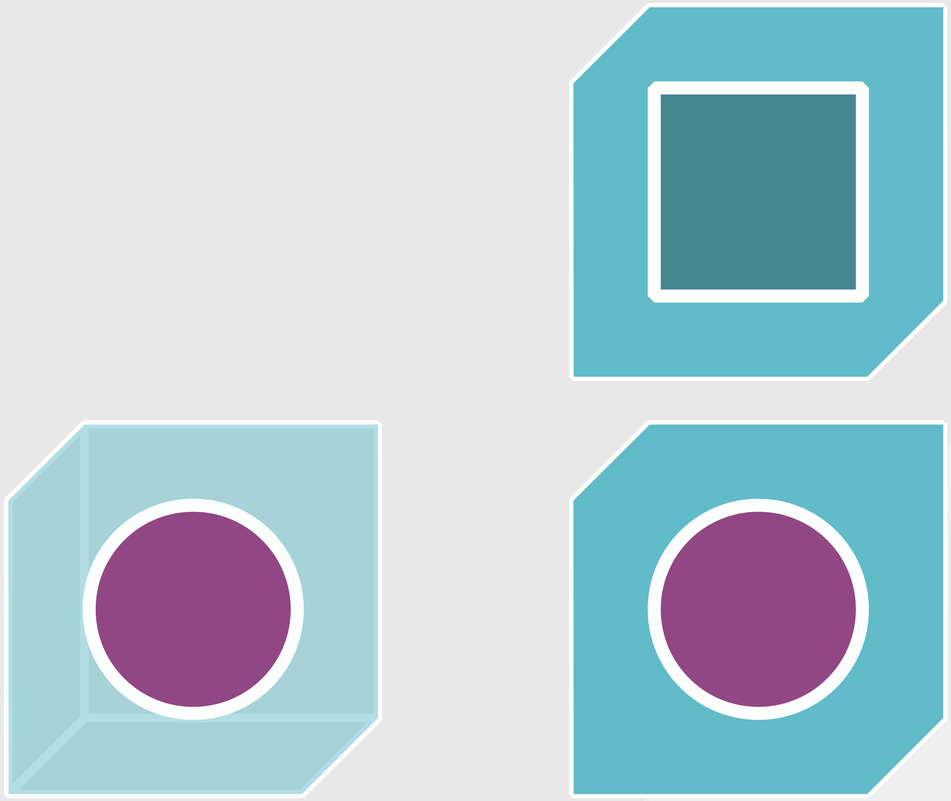

Ivan Čukić kdab.com, cukic.co

ASSIGNMENT



INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

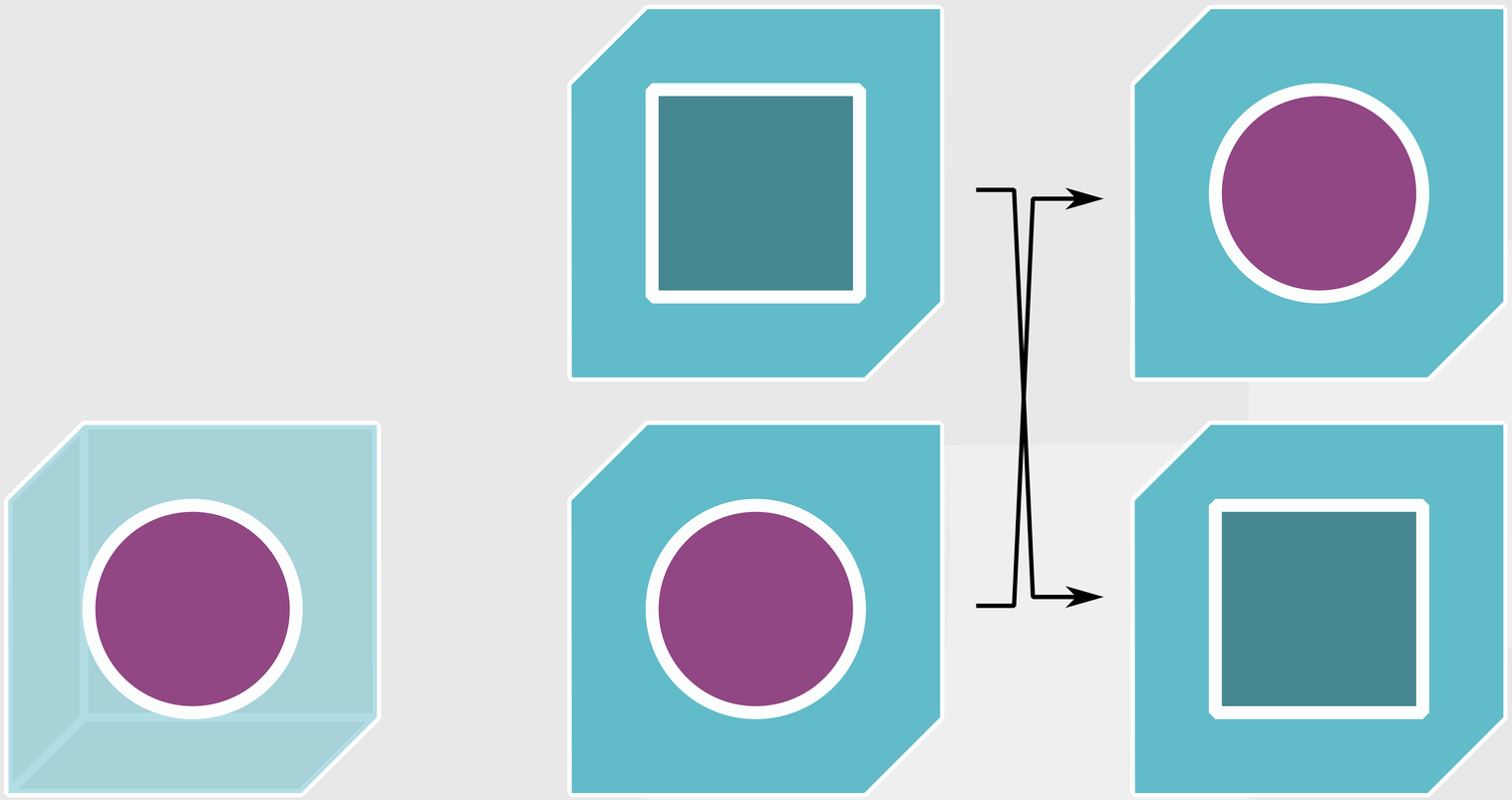
ASSIGNMENT



INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

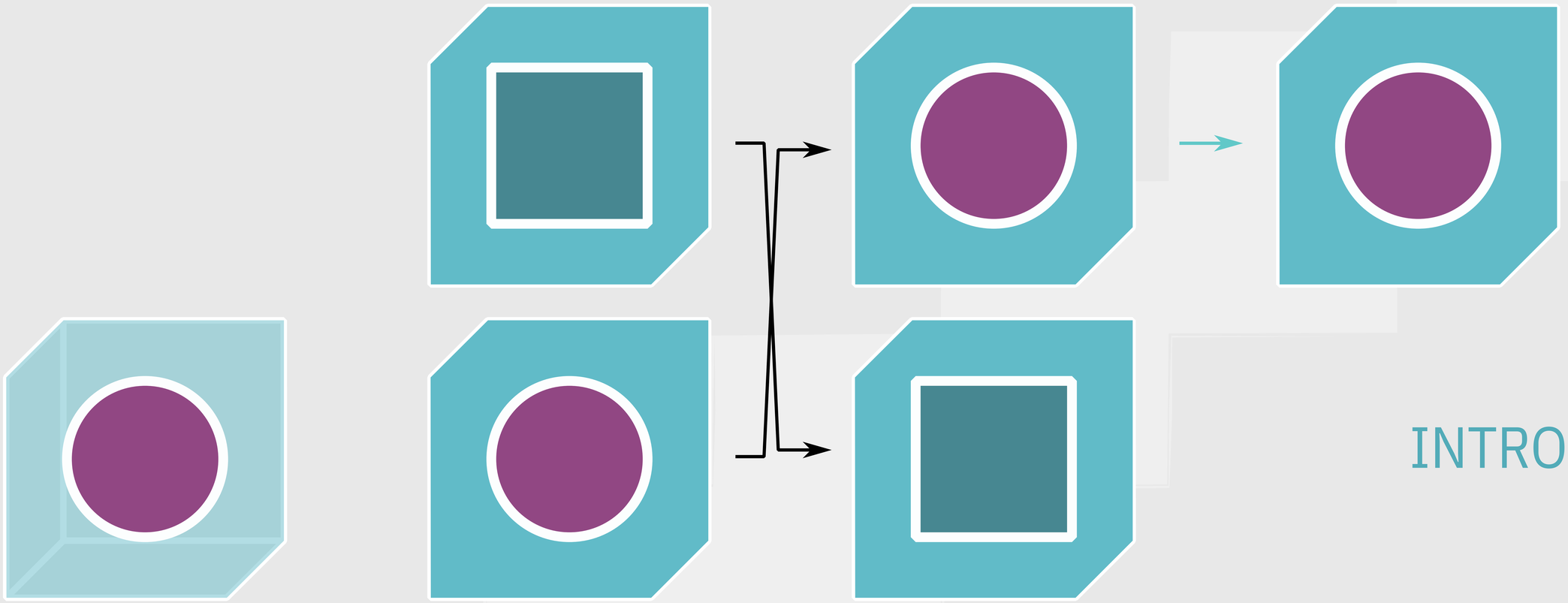
ASSIGNMENT



INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

ASSIGNMENT



PREREQUISITES

Member variables can not be references:

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

    // ...
};
```

RULE

Member variables should be value types.

Ivan Čukić kdab.com, cukic.co

SWAPPING

- Double buffering
- Processing collections in separate threads

Ivan Čukić kdab.com, cukic.co

MARK AND SWEEP

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


Ivan Čukić kdab.com, cukic.co

MARK AND SWEEP

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


Ivan Čukić kdab.com, cukic.co

MARK AND SWEEP

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Meeting C++ 2023

Ivan Čukić kdab.com, 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

STATES

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


Ivan Čukić kdab.com, cukic.co

STATES

"A nice bag to put things in"

– Miloš Anđelković, Class Layout

Ivan Čukić kdab.com, cukic.co

STATES

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


Ivan Čukić kdab.com, cukic.co

STATES

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

STATES

```
void load_page() {
    [[assert: dom == nullptr]];
    [[assert: not loading]];
    [[assert: not connection]];

    connection = open(url);
    loading = true;

    // ...
```

STATES

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

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;
};
```

STATES

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


Ivan Čukić kdab.com, cukic.co

STATES

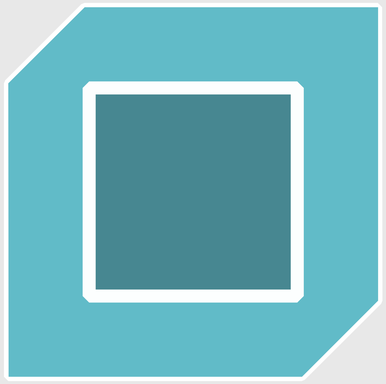
```
std::visit(
    overloaded {
        [&] (const initial_t& initial) {
            page = start_loading(initial.url);
        },
        // ...
    }, page);
```


Ivan Čukić kdab.com, cukic.co

STATES

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

STATES

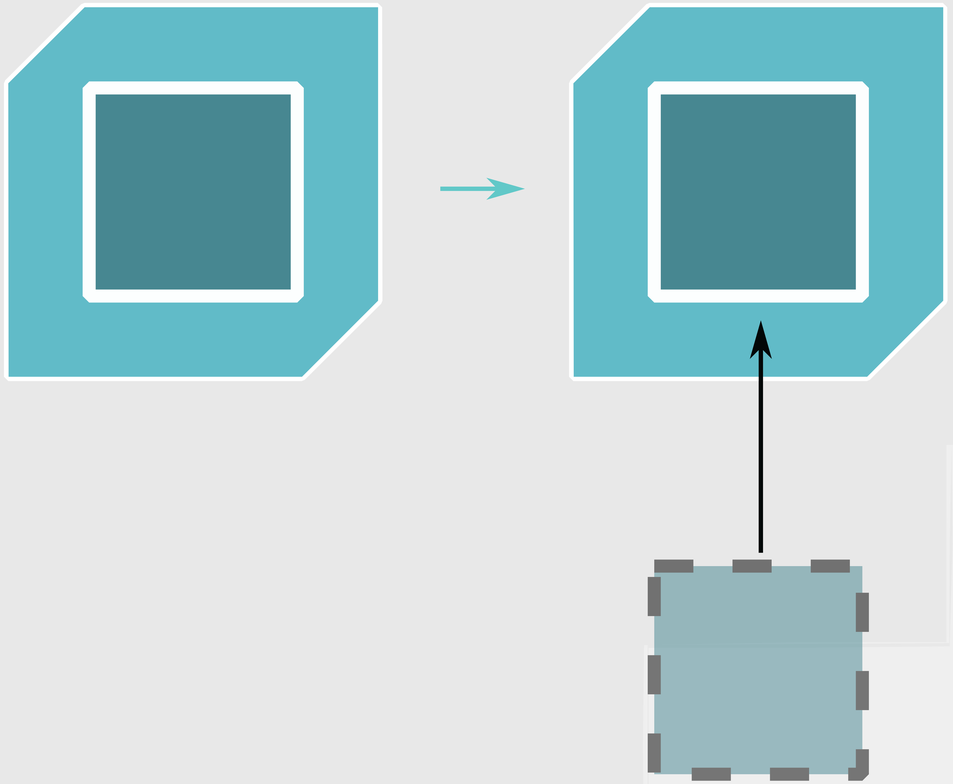


INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

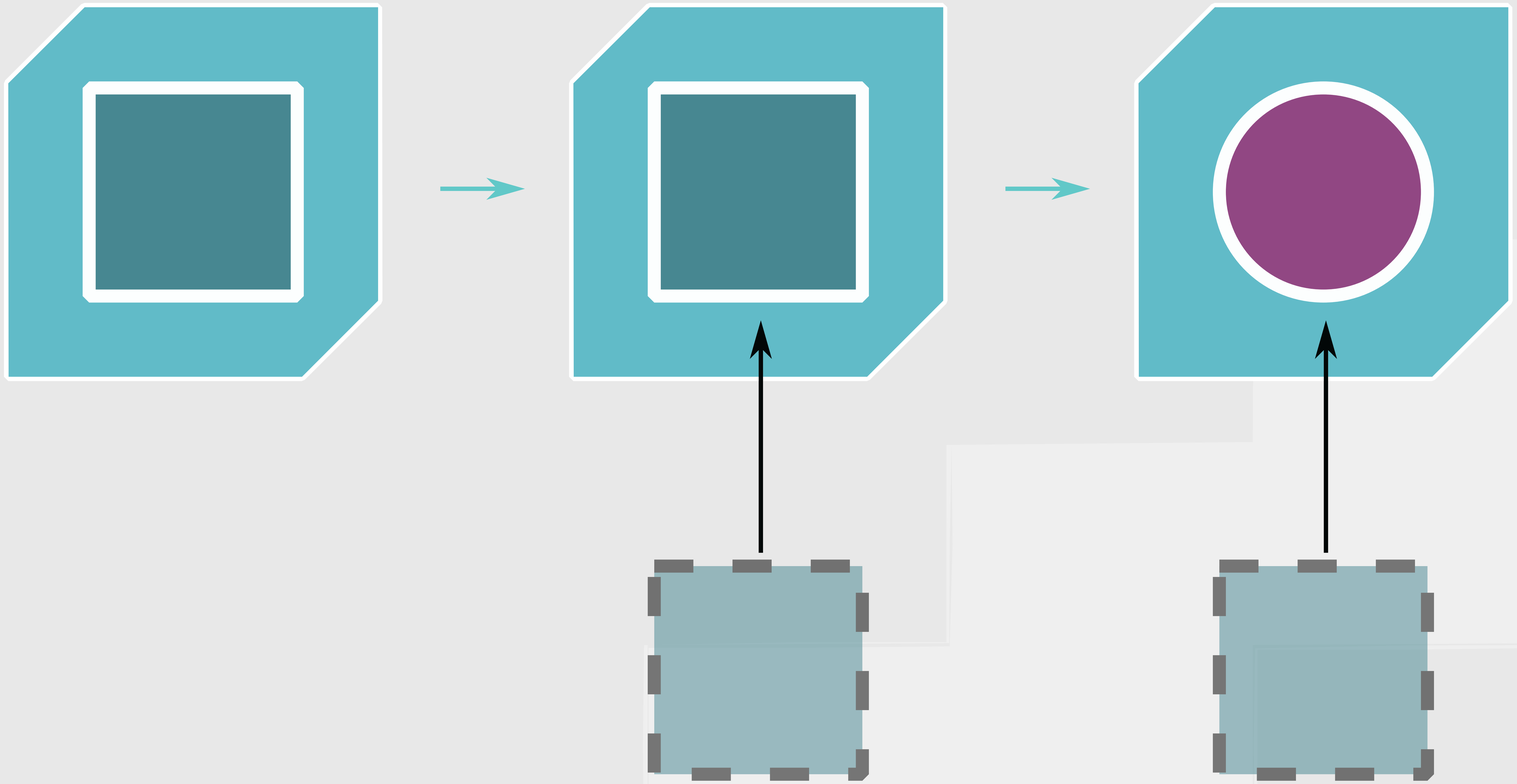
Meeting C++ 2023

Ivan Čukić kdab.com, cukic.co

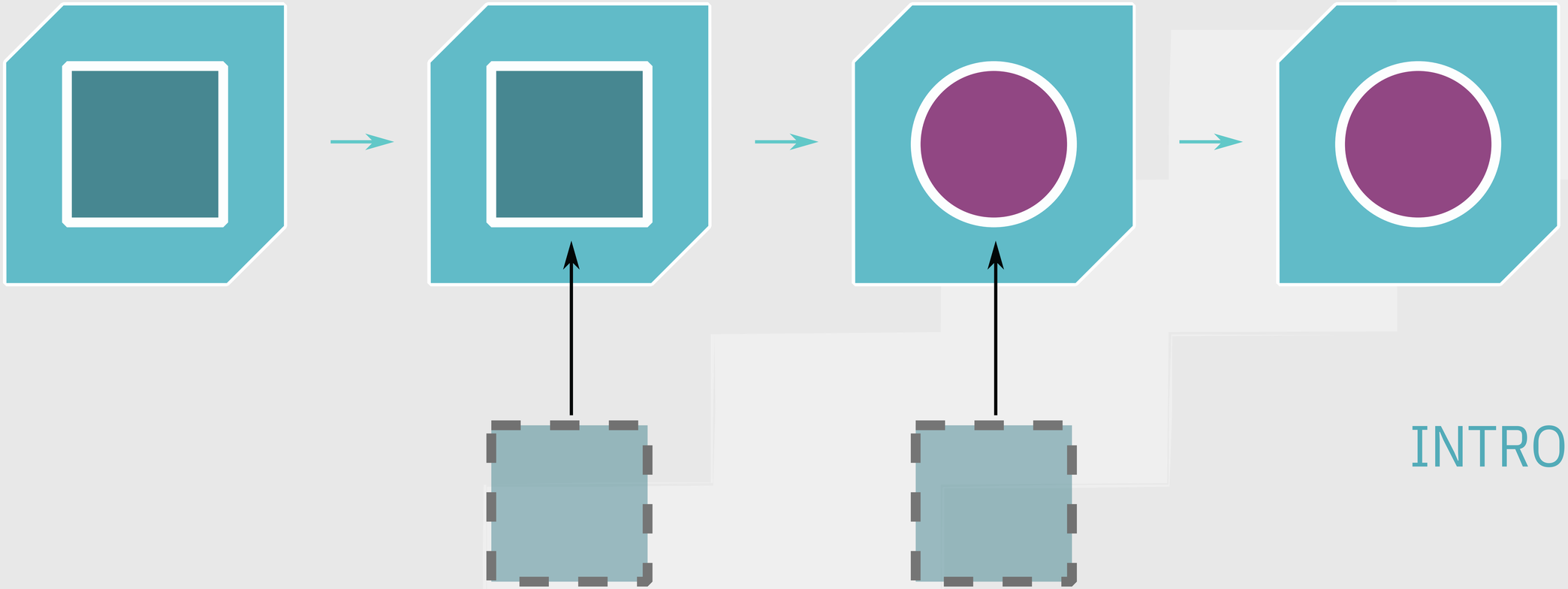
STATES



STATES



STATES



STATES

```
page_t new_page;

// generate the new state ...

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

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);
```

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

Meeting C++ 2023

Ivan Čukić kdab.com, cukic.co

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

ERRORS

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

ERRORS

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


Ivan Čukić kdab.com, cukic.co

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;
}
```


Ivan Čukić kdab.com, cukic.co

C++ error handling, let’s abuse the co_await operator

Antoine Morrier

https://cpp-rendering.io/

MESS IN THE KITCHEN

- Making lunch
- Cleaning up
- Burnt lunch

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

MESS IN THE KITCHEN

Forget about making lunch,
and destroy the kitchen.

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

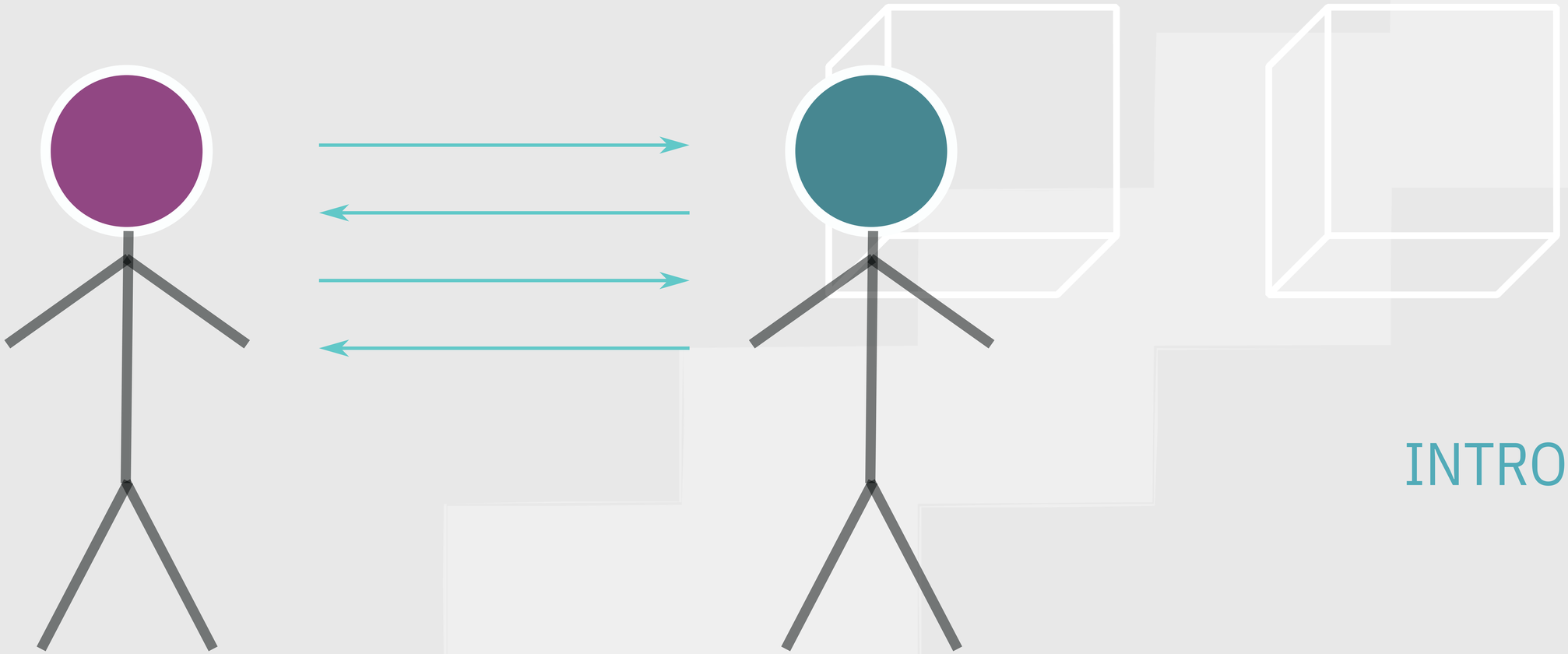
- Try to find the fix
- Continue with the process

Ivan Čukić kdab.com, cukic.co

ERRORS

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

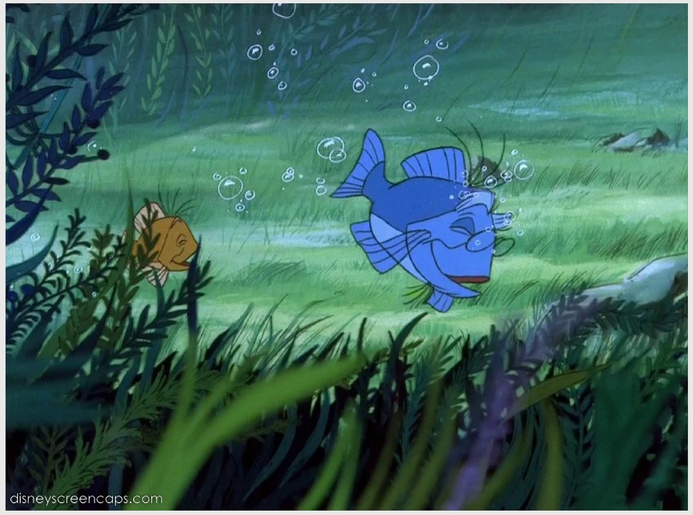
ERRORS



INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

ERRORS



To and fro

Stop and go

That’s what makes the world go round

The Sword in the Stone, (C) Disney

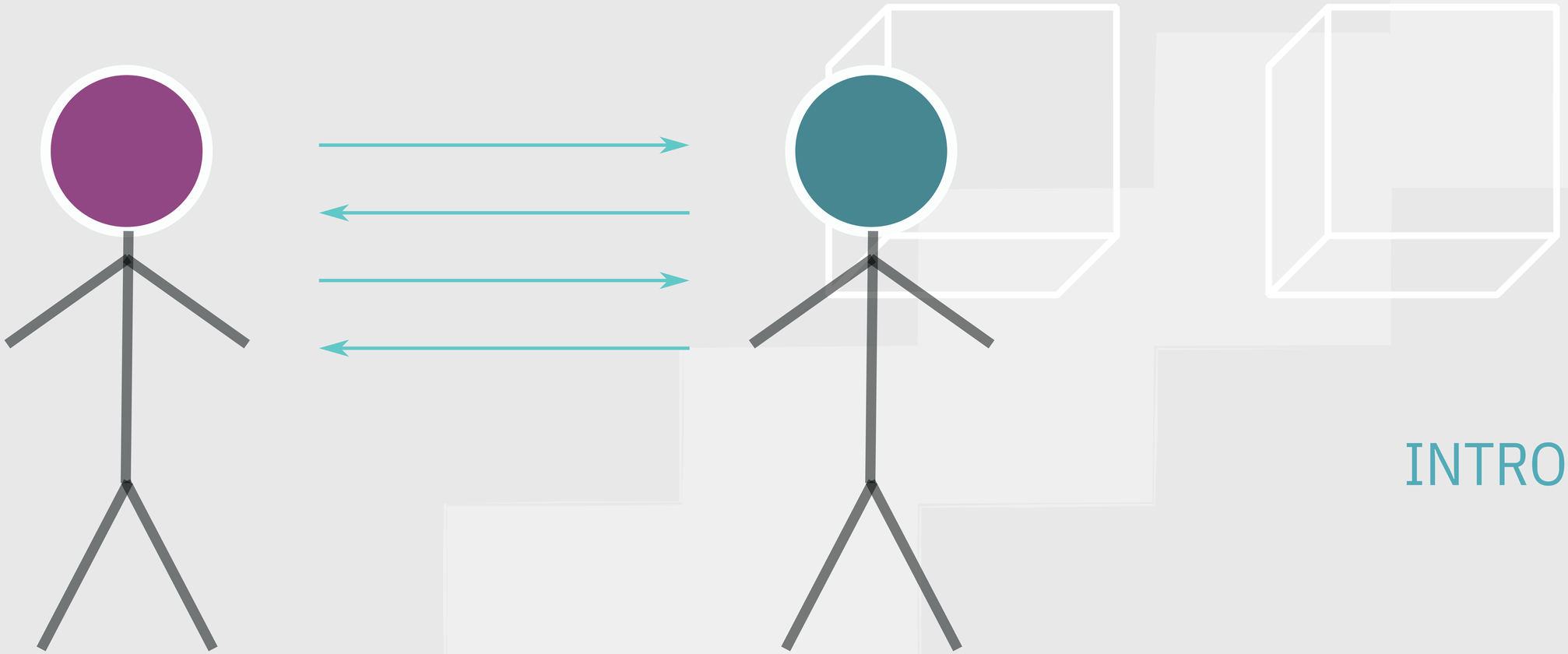
ERRORS

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

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


Ivan Čukić kdab.com, cukic.co

ERRORS



INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

ERRORS

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

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


Ivan Čukić kdab.com, cukic.co

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)};

    // ...
```


Ivan Čukić kdab.com, cukic.co

ERRORS

```
while (true) {
    auto result = co_await parse();

    if (result) // ...

    auto* message = result.error();
    // propose a fix
    (*message) = skip_line;
}
```


Ivan Čukić kdab.com, cukic.co

ERRORS

```
std::generator<result_t<int>> parse()
{
    // ...

    // if no error
    co_yield 42;
```


Ivan Čukić kdab.com, cukic.co

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

Meeting C++ 2023

Ivan Čukić kdab.com, cukic.co

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Regular Revisited

Victor Ciura

VALUES

- Values and variables
- Value assignment semantics

Ivan Čukić kdab.com, cukic.co

VALUES

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

Ivan Čukić kdab.com, cukic.co

Optimizing Multithreaded Performance

Shivam Kunwar

VALUES

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

```
class page_t {
private:
    class page_private_t;
    std::unique_ptr<page_private_t> m_pimpl;

public:
    // ...
};
```


Ivan Čukić kdab.com, cukic.co

VALUES

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

Ivan Čukić kdab.com, cukic.co

VALUES

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


Ivan Čukić kdab.com, cukic.co

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

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

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

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

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

COW?

Ivan Čukić kdab.com, cukic.co

VALUES

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


Ivan Čukić kdab.com, cukic.co

VALUES

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


Ivan Čukić kdab.com, cukic.co

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)...))
    {}

    // ...
```


Ivan Čukić kdab.com, cukic.co

VALUES

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

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

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


Ivan Čukić kdab.com, cukic.co

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, cukic.co

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

IDIOMS

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


Ivan Čukić kdab.com, cukic.co

IDIOMS

```
for (auto index = 0UZ;
     index != persons.size();
     ++index) {

    persons[index] ... // idiomatic
    persons ... // not idiomatic
    index ... // const access is idiomatic
}
```


Ivan Čukić kdab.com, cukic.co

IDIOMS

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

IDIOMS

- Previous knowledge
- Comprehension
- Safety

Ivan Čukić kdab.com, cukic.co

JF Bastien

Safety and Security: The Future of C++

CppNow 23

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


Ivan Čukić kdab.com, cukic.co

IDIOMS

```
for (auto index = 0UZ;
     index != persons.size();
     ++index) {

    persons[index]
}
```

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

IDIOMS

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


Ivan Čukić kdab.com, cukic.co

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

Ivan Čukić kdab.com, cukic.co

VALUES

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

IDIOMS

```
std::mutex mutex;

void do_something() {
    mutex.lock();

    // ...

    mutex.unlock();
}
```

IDIOMS

```
std::mutex mutex;

void do_something() {
    std::lock_guard<std::mutex> lock(mutex);
    // or unique_lock

    // ...
}
```


Ivan Čukić kdab.com, cukic.co

IDIOMS

Use a recursive mutex by default?

Don’t lock at all?

Ivan Čukić kdab.com, cukic.co

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&) {
        // ...
    }
```


Ivan Čukić kdab.com, cukic.co

LANGUAGE BARRIER

Tsukiji Hongan-Ji

(C) TokyoViews

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

LANGUAGE BARRIER

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

INTRODUCTION
WRAPS
SWAPS
STATES
ERRORS
VALUES
SAFETY

Ivan Čukić kdab.com, cukic.co

BENEFITS

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

DOWNSIDES

- Development under heavy chains
- More bureaucracy
- Rewriting everything

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