Dr Ivan Čukić KDAB

DESIGN IDIOMS FROM AN ALTERNATE UNIVERSE

INTRODUCTION

DATA

FUNCTIONS AND DATA

ABSTRACTIONS

FUNCTIONS

ABOUT ME

- KDAB senior software engineer

  Software experts in Qt, C++ and 3D / OpenGL
- Author of “Functional Programming in C++”

  available in English, Chinese, Korean, Russian, Polish
- Trainer / Consultant
- KDE developer
- University professor

COMPOSITION

Doug McIlroy and Dennis Ritchie

COMPOSITION

```
tr -cs A-Za-z '\n' |
    tr A-Z a-z |
    sort |
    uniq -c |
    sort -rn |
    sed ${1}q
```

Doug McIlroy, Bell System Technical Journal, 1978:

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new ”features”.
- Expect the output of every program to become the input to another, as yet unknown, program.

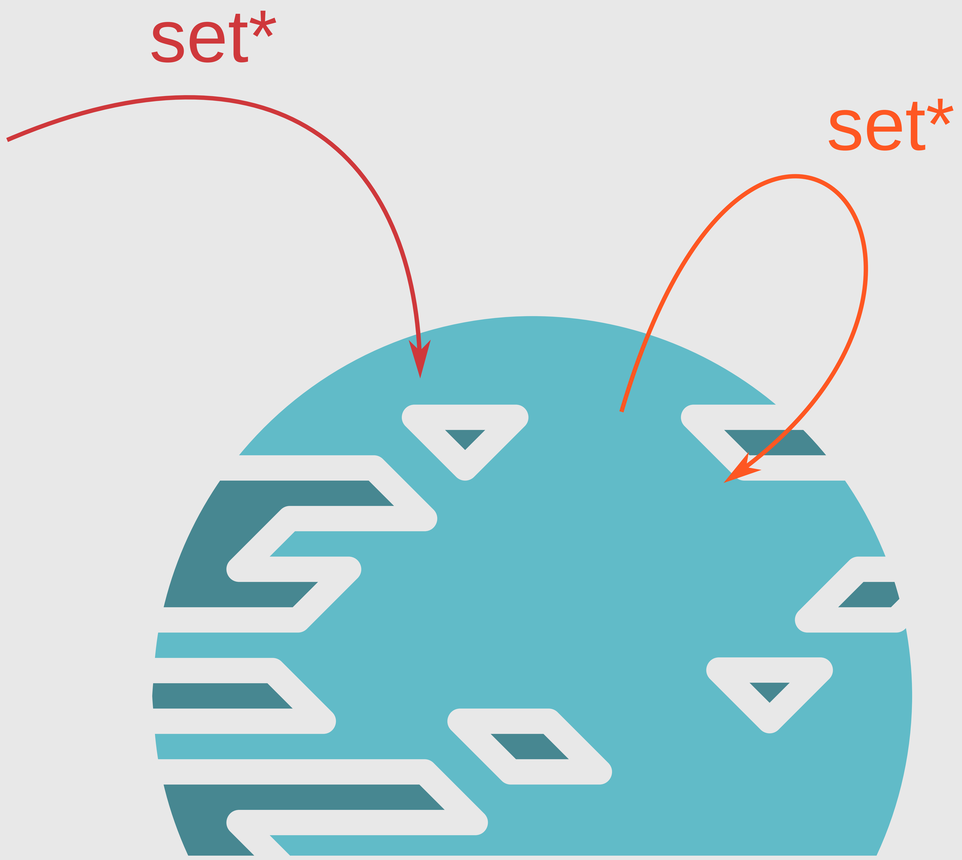
...

SOFTWARE DESIGN

- Composition
- Abstraction
- Components and objects
- Decoupling

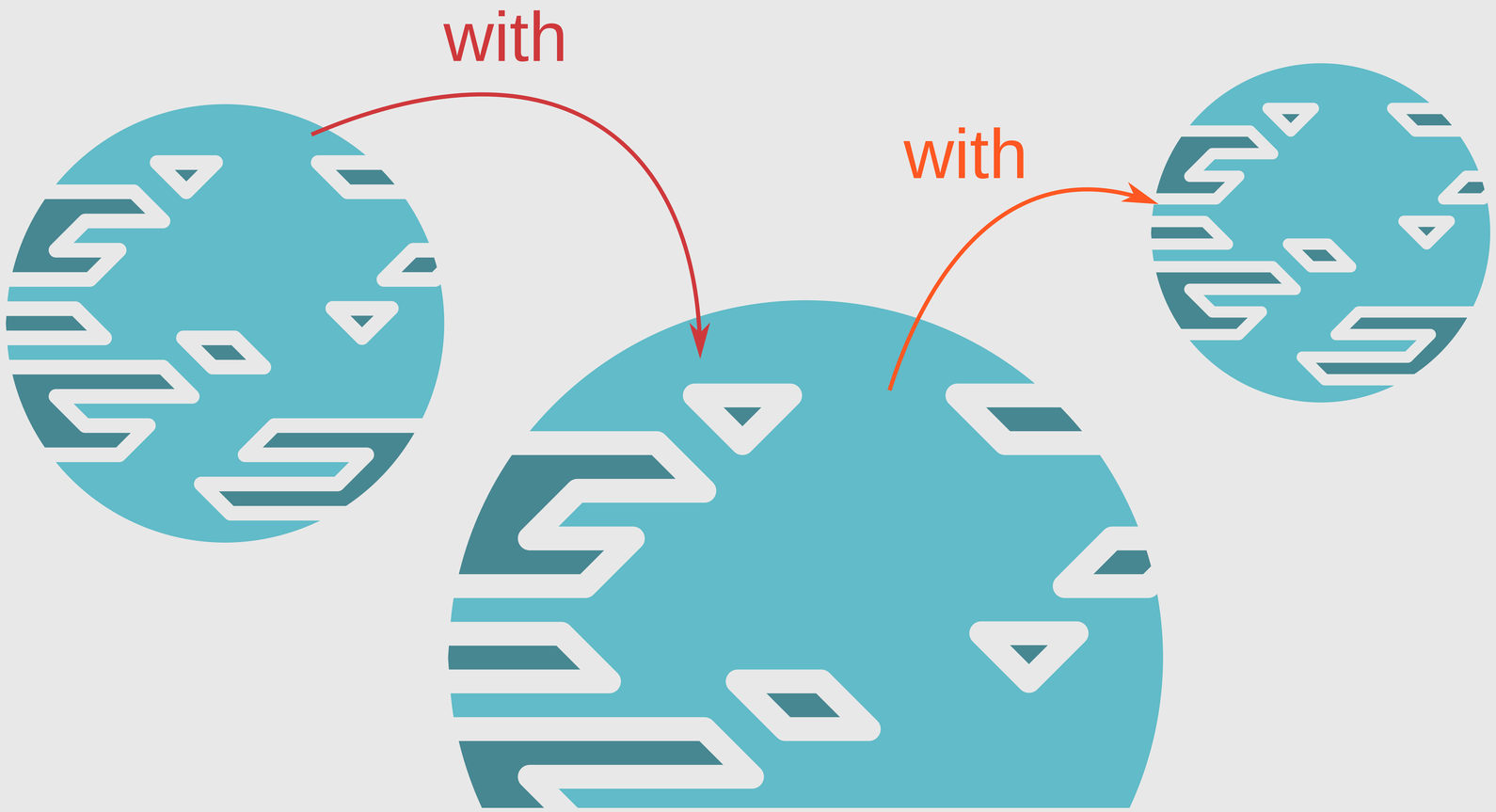
INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

ALTERNATE UNIVERSES



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

ALTERNATE UNIVERSES



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

ALTERNATE UNIVERSES

```
void object::set_property(type value)
```

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

ALTERNATE UNIVERSES

auto with_property(object&&, type value) → object&&

Move-only types can save the API, Ivan Čukić

itCppCon20

INTRODUCTION

DATA

FUNCTIONS AND DATA

ABSTRACTIONS

FUNCTIONS

Use composition over inheritance

Ivan Čukić KDAB

DATA

```
struct state_t {
    bool started;
    bool finished;
    unsigned count;
    string url;
    socket_t web_page;
};
```

```
struct state_t {
    bool started = false;
    bool finished = true;
    unsigned count = 42;
    string url = ...;
    socket_t web_page = ...;
};
```

When classes have an “isValid” method or similar, the code using them often is less clear and harder to maintain. If possible, validity should be an invariant that can not be violated. – Arne Mertz

DATA

```
struct init_t { string url; };

struct running_t {
    unsigned count; socket_t web_page;
};

struct finished_t {
    const unsigned count;
};
```

```
using state_t = std::variant<
        init_t,
        running_t,
        finished_t
    >;
```

AKA “Sum types” in the alternate universe

- No invalid states
- Member functions can be per-state
- Automatic resource disposal – proper RAII
- Easy handling with std::visit and overloaded lambdas

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

OPTIONAL

```
using optional_state_t =
    std::variant<state_t,
                 empty_state_t>;
```

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

OPTIONAL

```
std::optional<state_t>
```

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

OPTIONAL

```
state_t::start_counting

initialize_counter(state_t) -> ...
```

OPTIONAL

- Functions that provide an alternative value

```
auto value = state ?
     *state : default_value;
     // value_or
...
```

OPTIONAL

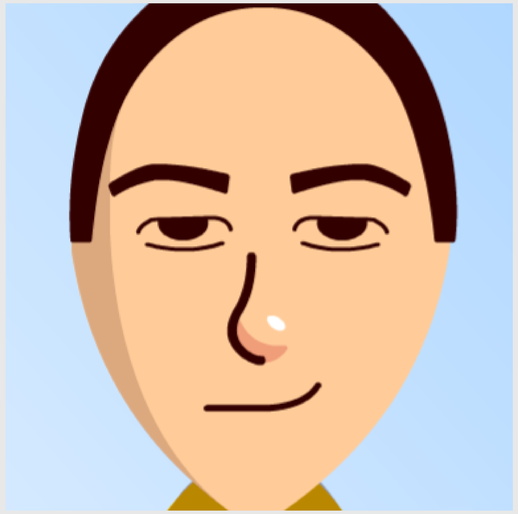
- Functions that provide an alternative value
- Functions that manipulate the value inside of the optional

```
if (!state) return {};
auto value = *state;
...
return { value };
```

OPTIONAL

- Functions that provide an alternative value
- Functions that manipulate the value inside of the optional
- Functions that do both

OPTIONAL



9 times out of 10, a for-loop should either be the only code in a function, or the only code in the loop should be a function (or both).

[...]

– Tony Van Eerd @tvaneerd

OPTIONAL

```
optional<...> transform(... opt, ... fun)
{
    if (!opt) return {};
    return { std::invoke(fun, *opt) };
}
```

OPTIONAL

```
transform(state, &state_t::start_counting);
transform(state, initialize_counter);

// C++23 and P0798R4:
state.transform(initialize_counter);
```

OPTIONAL

```
auto lift_to_optional(auto fun)
{
    return [fun] (auto&& value) {
        return transform(FWD(value), fun);
    }
}
```

TASKS AND FUTURES

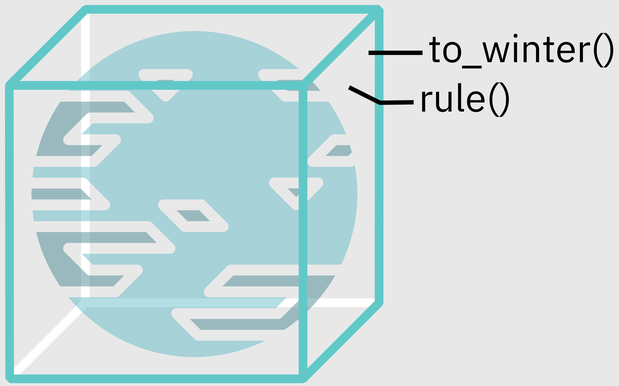
```
task_t<...> transform(... task, ... fun)
{
    auto value = co_await task;
    co_return std::invoke(fun, value);
}

// and optional, expected...
```

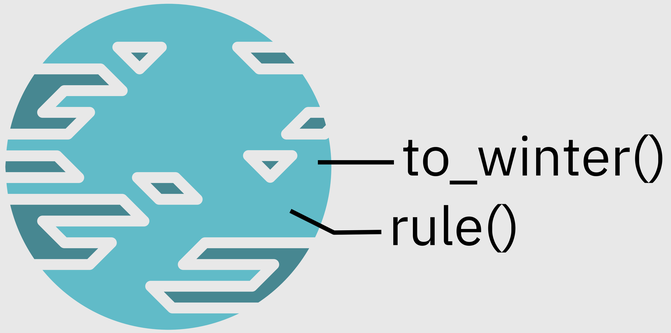
RANGES, ETC.

```
auto lift(auto fun)
{
    return [fun] (auto&& value) {
        return transform(FWD(value), fun);
    }
}
```

AKA “Functors” in the alternate universe

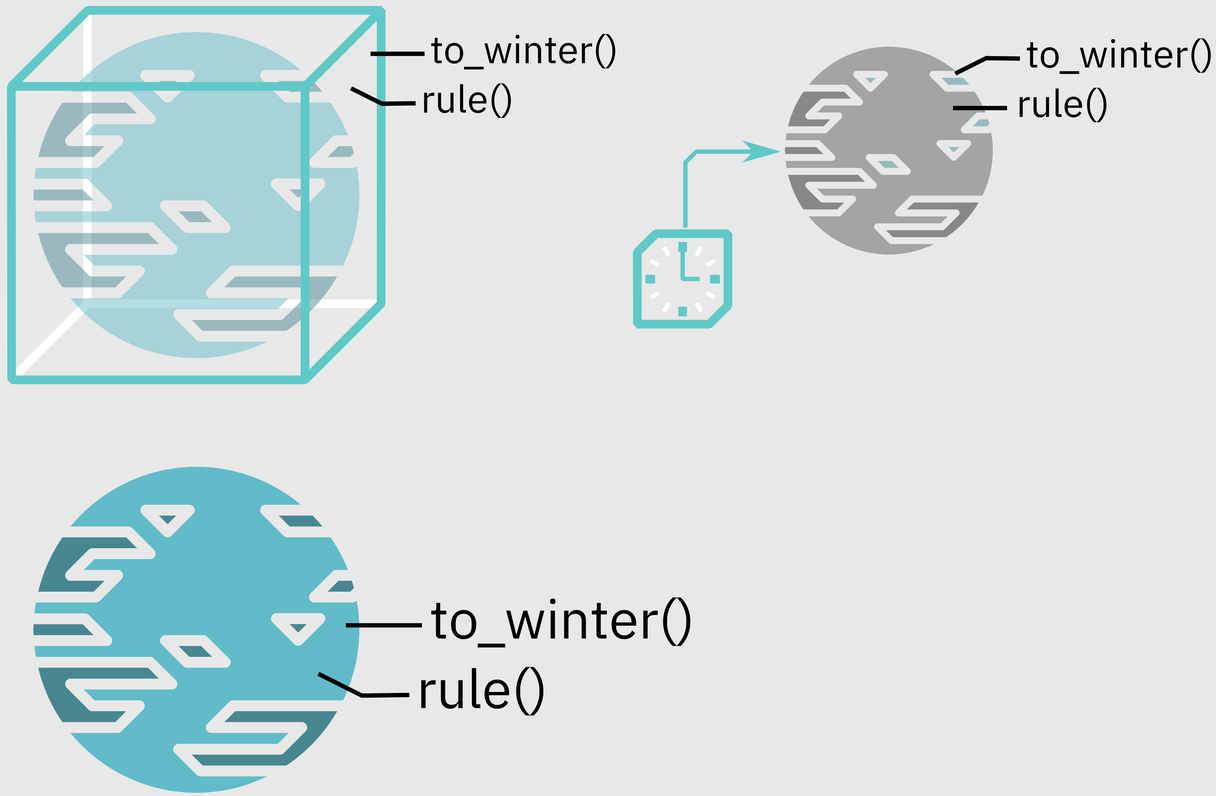
LIFTING





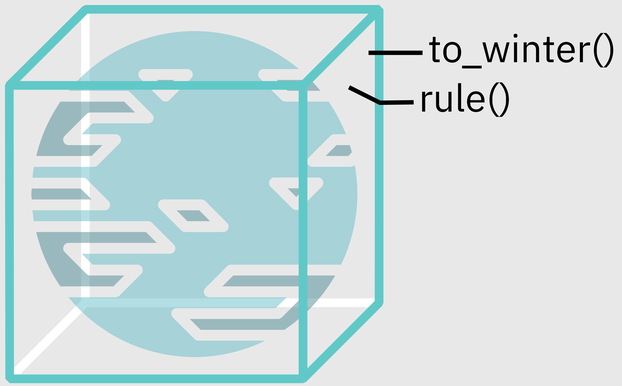
INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

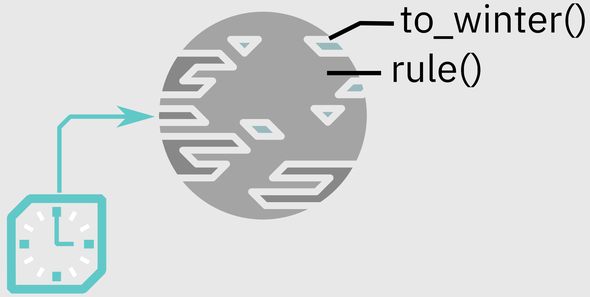
LIFTING

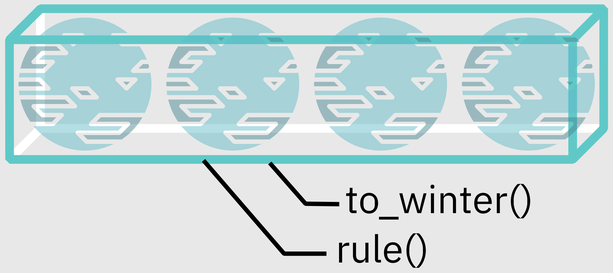


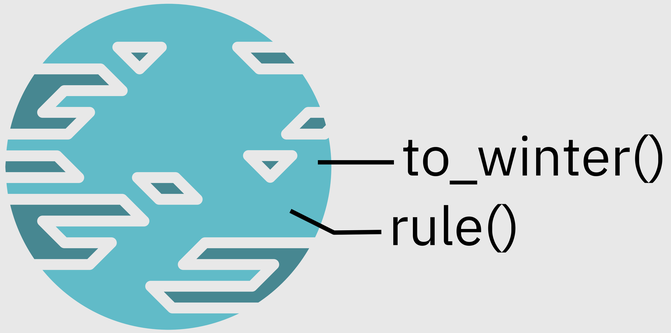
INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

LIFTING









INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

PROPERTIES

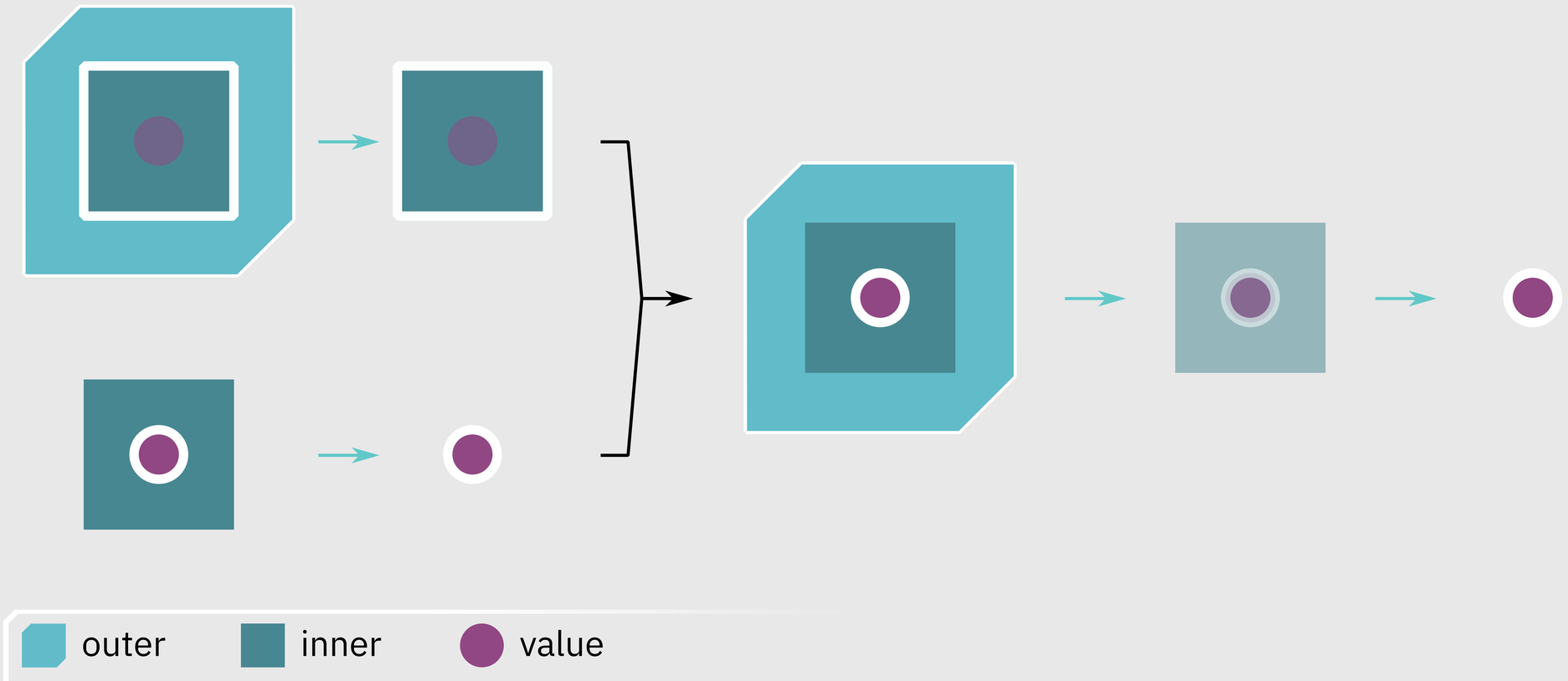
- View – gets a value from an object

  object → value
- Update – updates a value inside of an object

  (object&&, value) → object&&

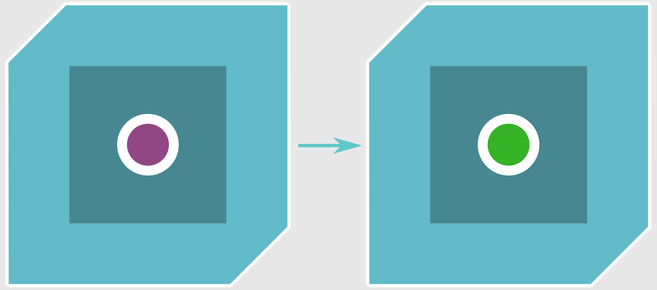
AKA “Lenses” in the alternate universe

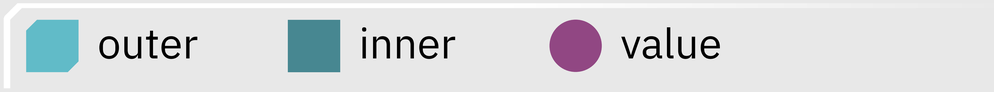
PROPERTIES



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

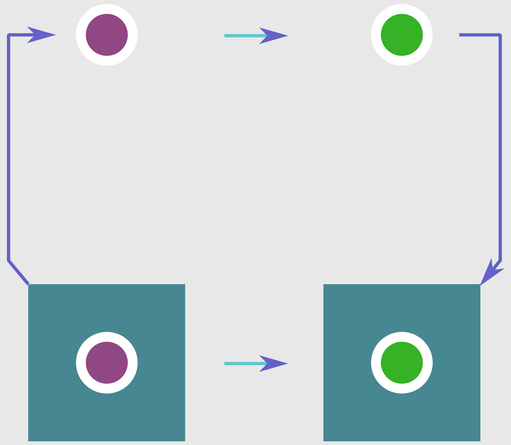
PROPERTIES

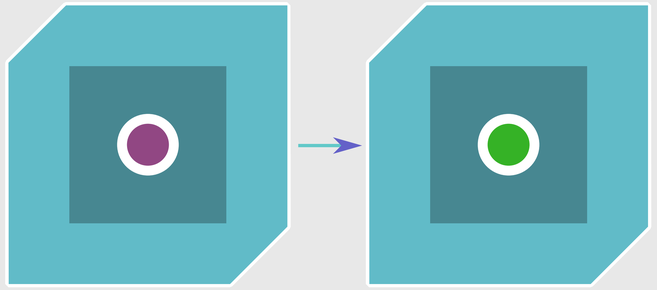




INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

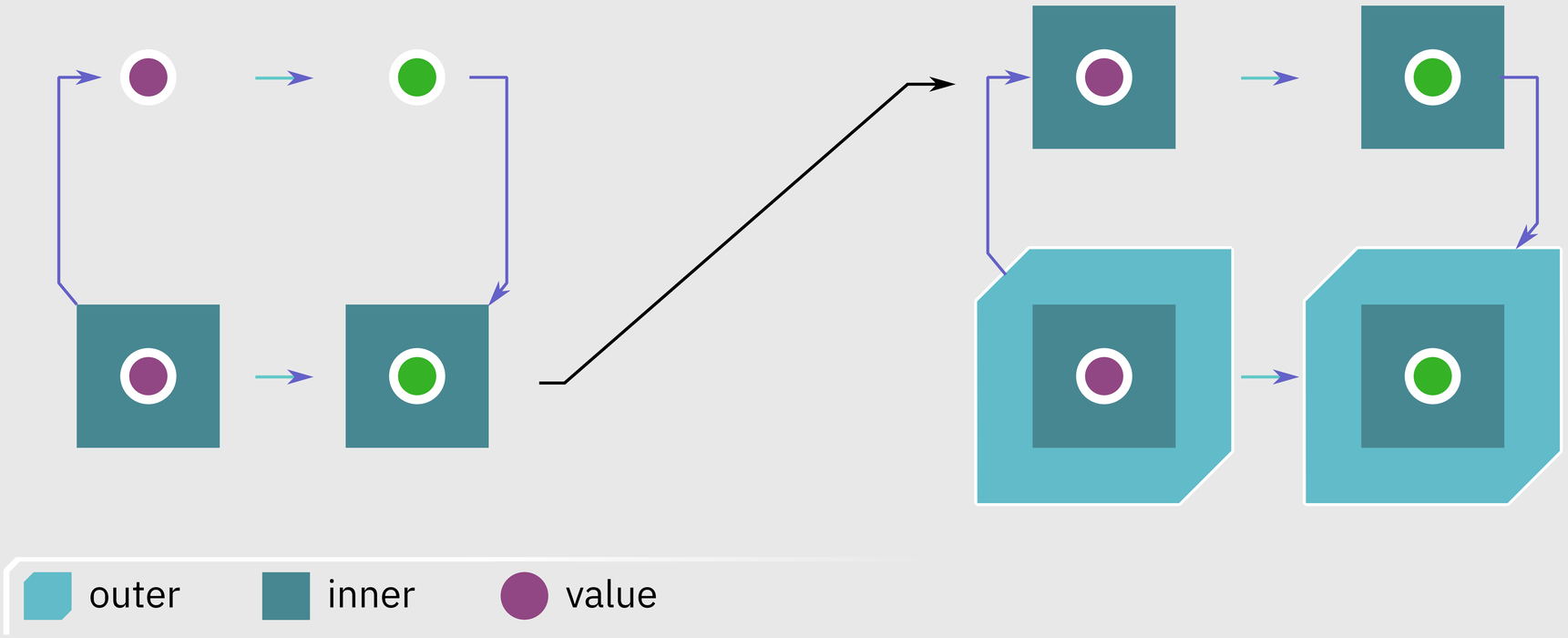
PROPERTIES





INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

PROPERTIES



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

PROPERTIES

```
auto compose_update = [ _left = ...,  _right = ... ]
    (outer_object_t outerValue, auto innerUpdateFn) {
        auto outerUpdateFunction =
            [_left, innerUpdateFn](inner_object_t innerValue) {
                return std::invoke(_left, std::move(innerValue),
                                   innerUpdateFn);
            };
        return std::invoke(_right, std::move(outerValue),
                           std::move(outerUpdateFunction));
    };

// operator >> and operator <<
```

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenant            : appartment_t -> tenant_t;
monthly_payment   : tenant_t -> double
```

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenant            : appartment_t -> tenant_t;
monthly_payment   : tenant_t -> double

auto payment = tenant
             >> monthly_payment;
accumulate(appartments(building), 0.0, plus{}, payment);
```

PROPERTIES

- Functors require a generic type

  $f : A \rightarrow B$
- Properties (Lenses) can work on any type

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenants           : appartment_t -> range<tenant_t>;
monthly_payments  : tenant_t -> range<double>;
```

property → range

=> property → range

=> property → range

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenants           : appartment_t -> range<tenant_t>;
monthly_payments  : tenant_t -> range<double>;
```

property → range

=> property → range

=> range → property

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenants           : appartment_t -> range<tenant_t>;
monthly_payments  : tenant_t -> range<double>;
```

property → range

=> property → range

=> range → property

PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenants           : appartment_t -> range<tenant_t>;
monthly_payments  : tenant_t -> range<double>;
```

property → range

=> property → range

→ property

PROPERTIES

```
appartments      : building_t -> range<appartment_t>;
tenants          : appartment_t -> range<tenant_t>;
monthly_payments : tenant_t -> range<double>;
```

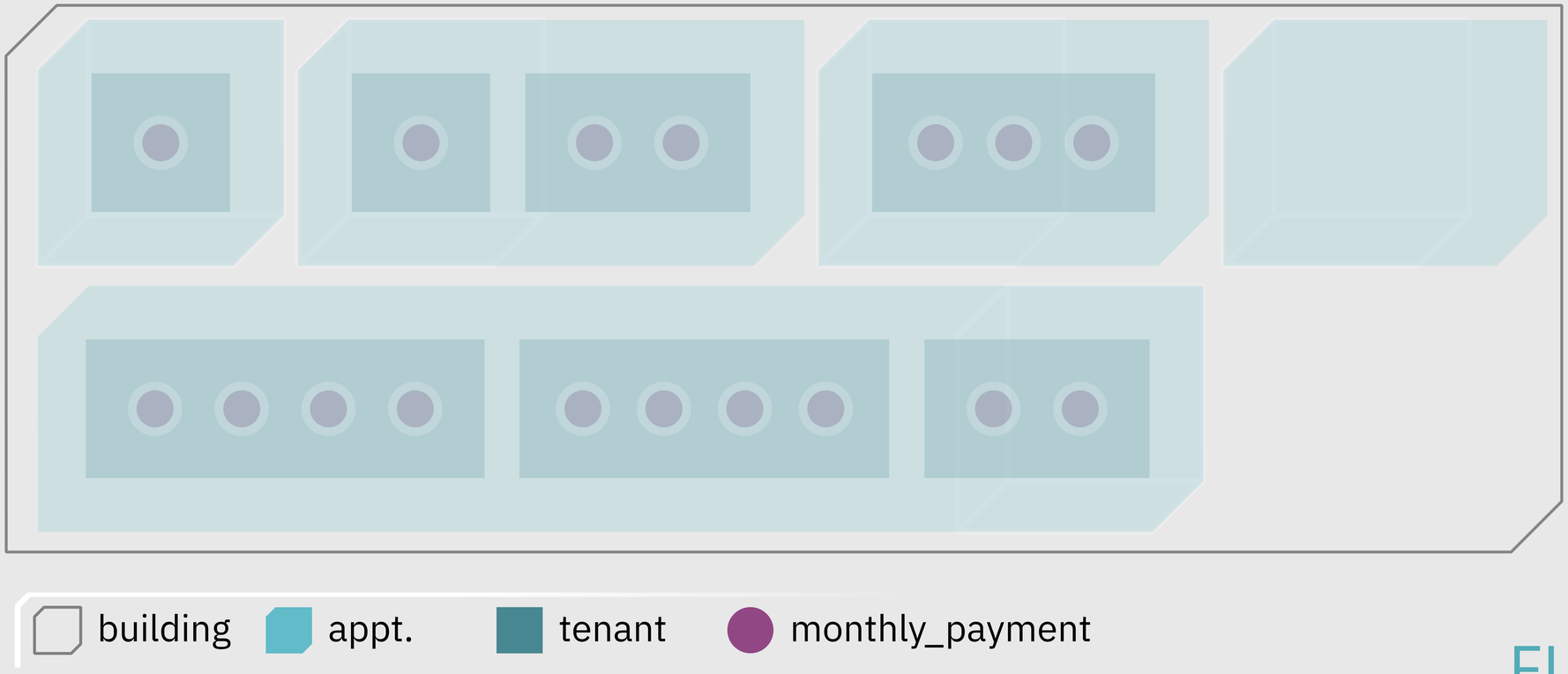
range => composed property

PROPERTIES

```
appartments      : building_t -> range<appartment_t>;
tenants          : appartment_t -> range<tenant_t>;
monthly_payments : tenant_t -> range<double>

auto payments = appartments
              >> tenants
              >> monthly_payments;
accumulate(payments(building), 0.0);
```

FOCUS

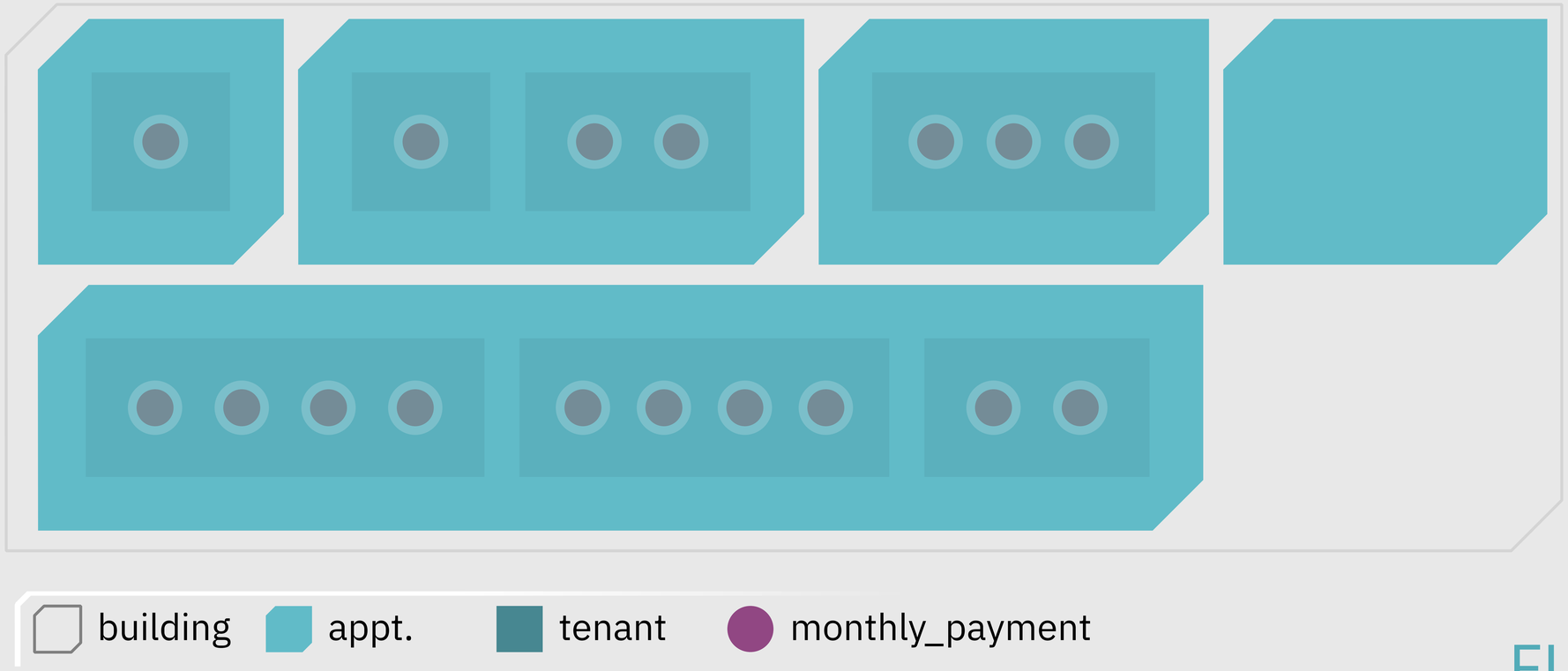
building



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

FOCUS

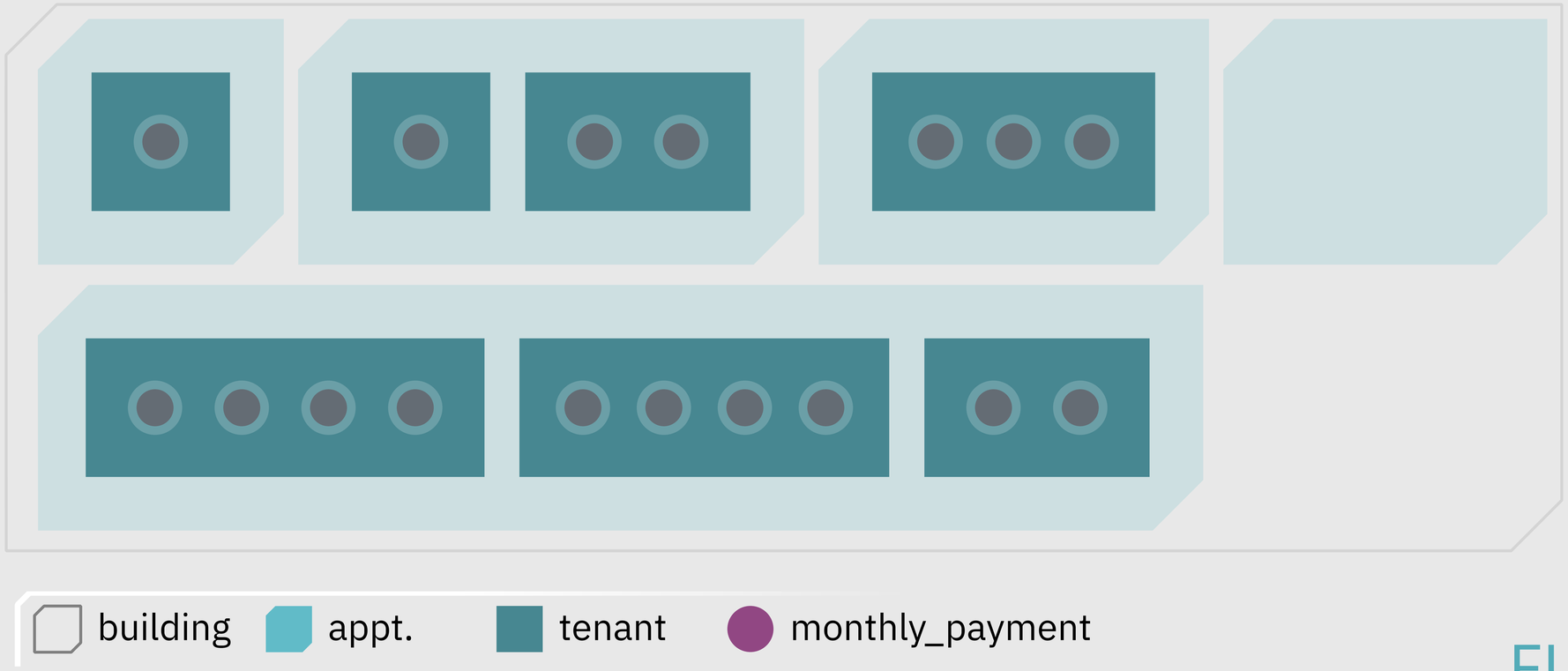
```
appartments(building);
```



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

FOCUS

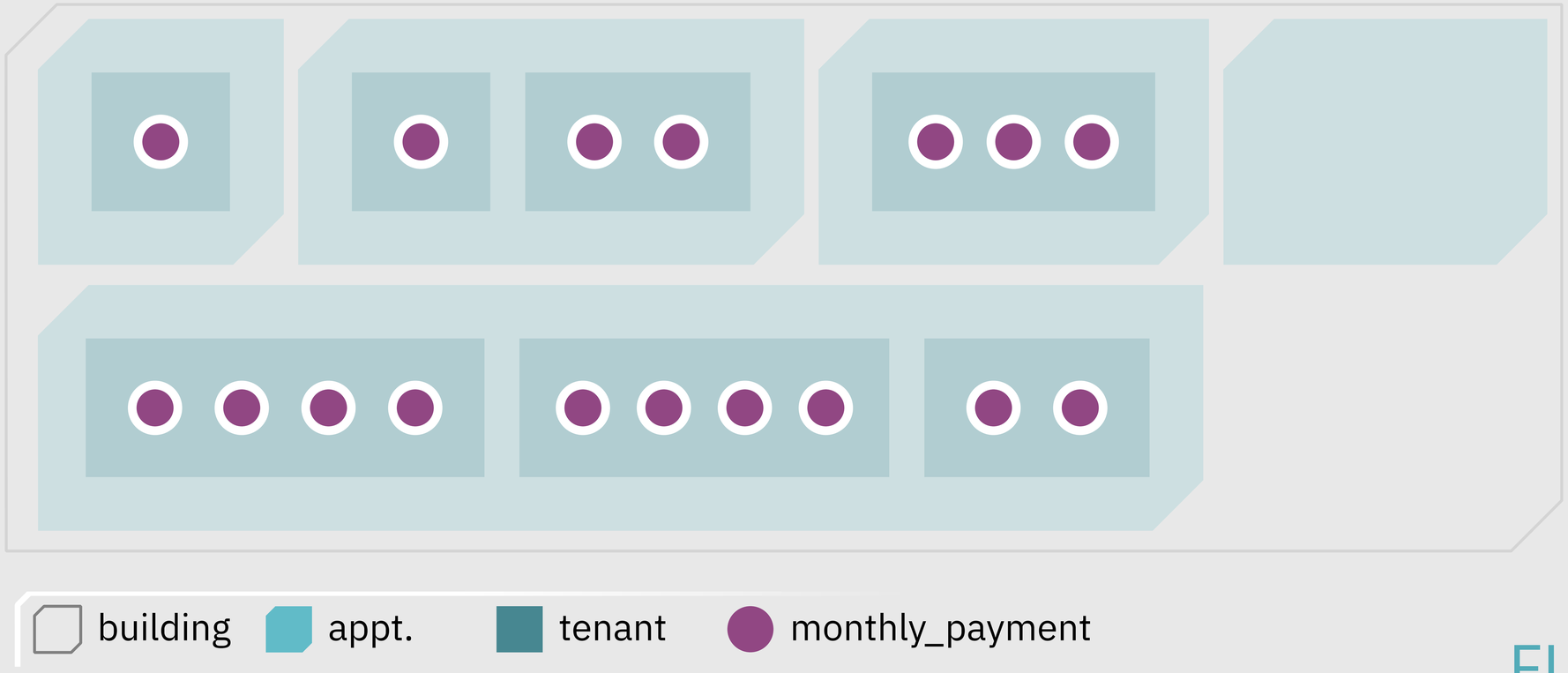
```
auto all_tenants = appartments >> tenants;
all_tenants(building);
```



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

FOCUS

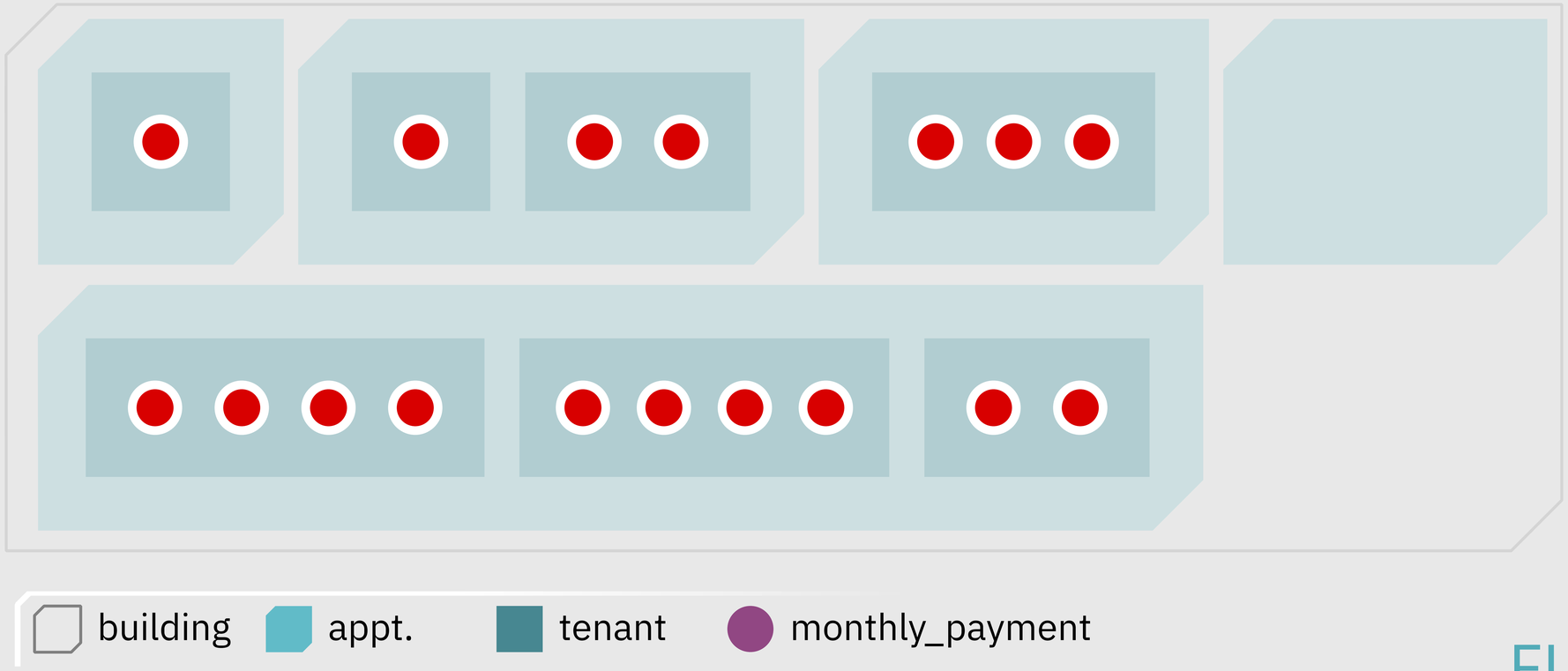
```
auto all_payments = appartments >> tenants >> monthly_payments;
all_payments(building);
```



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

FOCUS

```
auto expensive_building =
     all_payments(std::move(building), increase(20_percent));
```



INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS
FUNCTIONS

FOCUS

- Whole ranges
- Filtered ranges
- First element in a range
- First n elements of a range

...

INTRODUCTION
DATA
FUNCTIONS AND DATA
ABSTRACTIONS

FUNCTIONS

FUNCTION COMPOSITION

```
// g: A -> B, f: B -> C
auto compose(auto f, auto g)
{
    return [f, g](auto &&...args) {
        return f(g(FWD(args)...));
    }
}
```

++COMPOSITION

range<building_t>;

appartments : building_t → range<appartment_t>;

tenants : appartment_t → range<tenant_t>;

payments : tenant_t → range<double>;

++COMPOSITION

building_t;

appartment : building_t → appartment_t;

tenant : appartment_t → tenant_t;

payment : tenant_t → double;

++COMPOSITION

building_t;

appartment : building_t → appartment_t or exception;

tenant : appartment_t → tenant_t or exception;

payment : tenant_t → double or exception;

++COMPOSITION

building_t;

appartment : building_t → optional<appartment_t>;

tenant : appartment_t → optional<tenant_t>;

payment : tenant_t → optional<double>;

++COMPOSITION

```
// f: B -> m<C>, g: A -> m<B>
auto m_compose(auto f, auto g)
{
    return [f, g](auto &&arg) -> ... {
        auto g_res = std::invoke(g, FWD(arg));
        if (!g_res) return {};
        return std::invoke(f, *g_res);
    }
}
```

++COMPOSITION

```
auto my_appartment =
     co_await appartment(building);
auto main_tenant =
     co_await tenant(my_appartment);
auto this_month_payment =
     co_await payment(main_tenant);
```

++COMPOSITION

```
// f: B -> m<C>, g: A -> m<B>
auto m_compose(auto f, auto g)
{
    return [f, g](auto &&arg) -> ... {
        auto g_res = co_await std::invoke(g, FWD(arg));
        return std::invoke(f, FWD(arg));
        // ERROR (almost)
    }
}
```

++COMPOSITION

```
// f: B -> m<C>, g: A -> m<B>
auto m_compose(auto f, auto g)
{
    return [f, g](auto &&arg) -> ... {
        auto g_res = co_await std::invoke(g, FWD(arg));
        auto f_res = co_await std::invoke(f, FWD(arg));
        co_return f_res;
    }
}
```

++COMPOSITION

```
building_t;
appartment :  building_t → range<appartment_t>;
tenant :  appartment_t → range<tenant_t>;
payment :  tenant_t → range<double>;
```

WHAT ABOUT ...?

```
building_t;
appartment :  building_t → future<appartment_t>;
tenant :  appartment_t → future<tenant_t>;
payment :  tenant_t → future<double>;
```

WHAT ABOUT ...?

```
building_t;
appartment :  building_t → generator<appartment_t>;
tenant :  appartment_t → generator<tenant_t>;
payment :  tenant_t → generator<double>;
```

++COMPOSITION

- make : a → m<a>
- transform : …
- join : m<m<a>> → m<a>

AKA “Monads” in the alternate universe

++COMPOSITION

- optional
- expected
- vector
- pair*
- future

- Ranges
- Generators
- Senders
- Coroutines
- Herbceptions

- parsing
- logging
- ...

BACK TO PROPERTIES

```
appartments       : building_t -> range<appartment_t>;
tenants           : appartment_t -> range<tenant_t>;
monthly_payments : tenant_t -> double

auto payments = appartments
              >> tenants
              >> monthly_payments;
payments(building) -> range<double>
```

BACK TO PROPERTIES

```
appartments       : building_t -> ???<appartment_t>;
tenants           : appartment_t -> ???<tenant_t>;
monthly_payments : tenant_t -> ???<double>

auto payments = appartments
              >> tenants
              >> monthly_payments;
payments(building) -> ???<double>
```

SUMMARY

- Notice simple patterns
- Write functions that do one thing
- Make functions composable… whatever *composable* means to you
- Don’t be afraid of concepts with strange names
- Don’t assume something is useless because the example is

LINKS

- Move-only types can save the API, Ivan Čukić
  https://www.youtube.com/watch?v=l0ienjOkK-4
- isValid()? Establish invariants and avoid zombie objects, Arne Mertz
  https://arne-mertz.de/2021/09/isvalid-establish-invariants-avoid-zombies/
- p0798R4 Monadic operations for std::optional, Sy Brand
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0798r4.html

Web: https://**kdab.com**
Mail: **ivan.cukic@kdab.com**

Personal
Web: https://**cukic.co**
Mail: **ivan@cukic.co**
Twitter: **@ivan_cukic**

cukic.co/to/fp-in-cpp

Functional Programming in C++