THE MOST VALUABLE VALUES

by juanpe bolívar
https://sinusoid.al

PART I

VALUE SEMANTICS

value semantics

42

blue

"Juanpe Bolívar"

$f(x) = x^4$

$\mathbb{N}$

42

blue

Juanpe Bolívar

f(x) = x^2

ℕ

am I a value?

concrete
material
contingent
temporary

...I AM A THING

Plato was **wrong**.

But his ideas are a useful **metaphor** about how **reasoning** works.

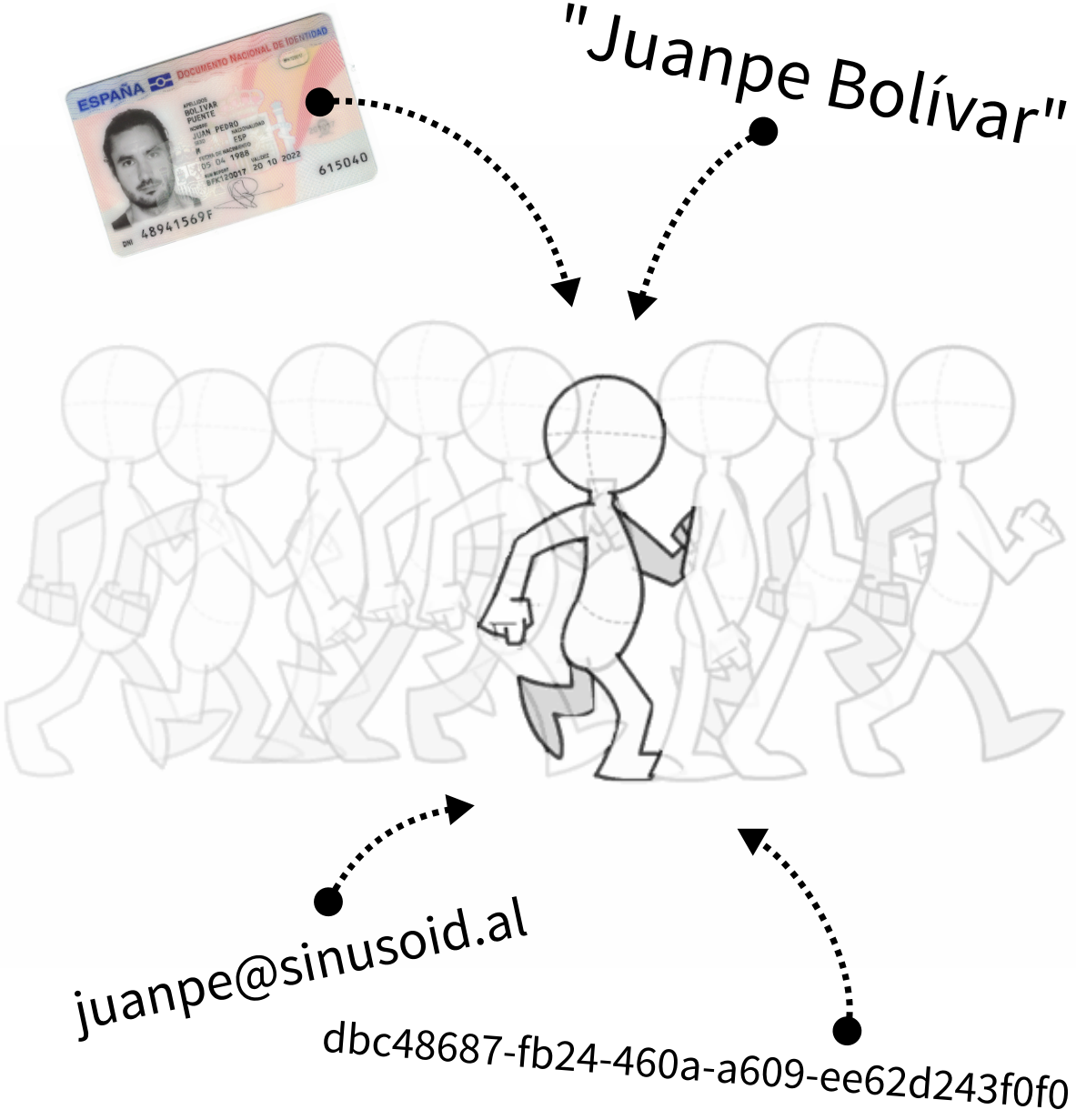
THINGS ARE REAL

VALUES ARE IMAGINARY

value of **all existence**

value of **a given state**

value of **identity**

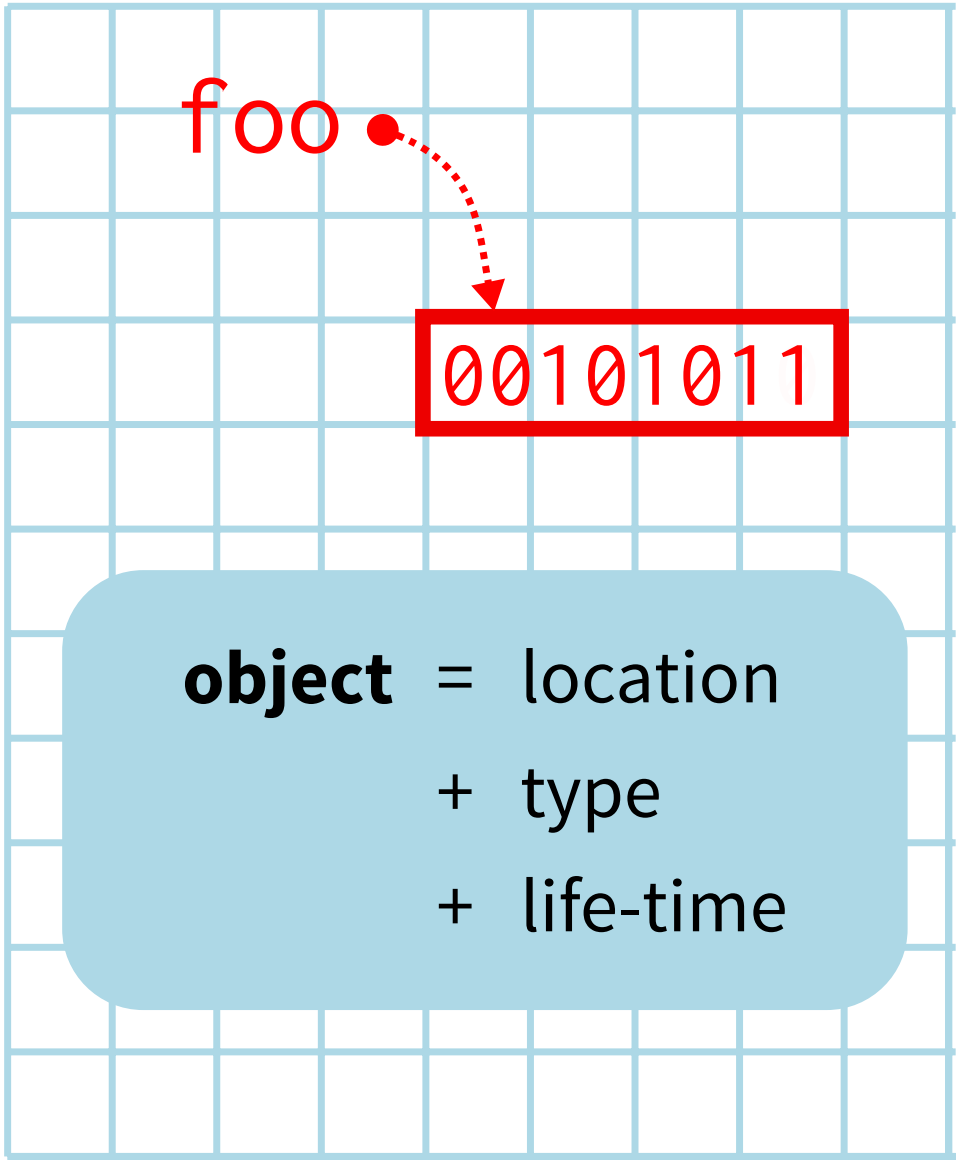


We use **language**
to **exchange** values

e.g. Haskell

```
let x = 42
    y = x + 2
    z = [ 1, 2, 3, 4 ]
    f = λ x -> x ^ 4
    w = map f z
```

```
{
    int foo = 42;
    int* p = &x;
    *p = x + 1;
    ...
}
```



“The limits of my language means the limits of my world.”

Ludwig Wittgenstein

object fetichism when all you can name is an object, everything looks like an object

```
struct gesichtbuch {
    struct person {
        id_t id;
        std::string name;
        std::string email;

        std::string phone;
        int birth_year;
    };
    std::unordered_map<id_t, person> people;
    boost::multi_index<std::pair<id_t, id_t>, ...> friendships;
};
```

operational specs

procedures

shared mutable state

pure functions

denotational specs

regular types

globals

object-oriented

shared_ptr

mutex

const

pass by value

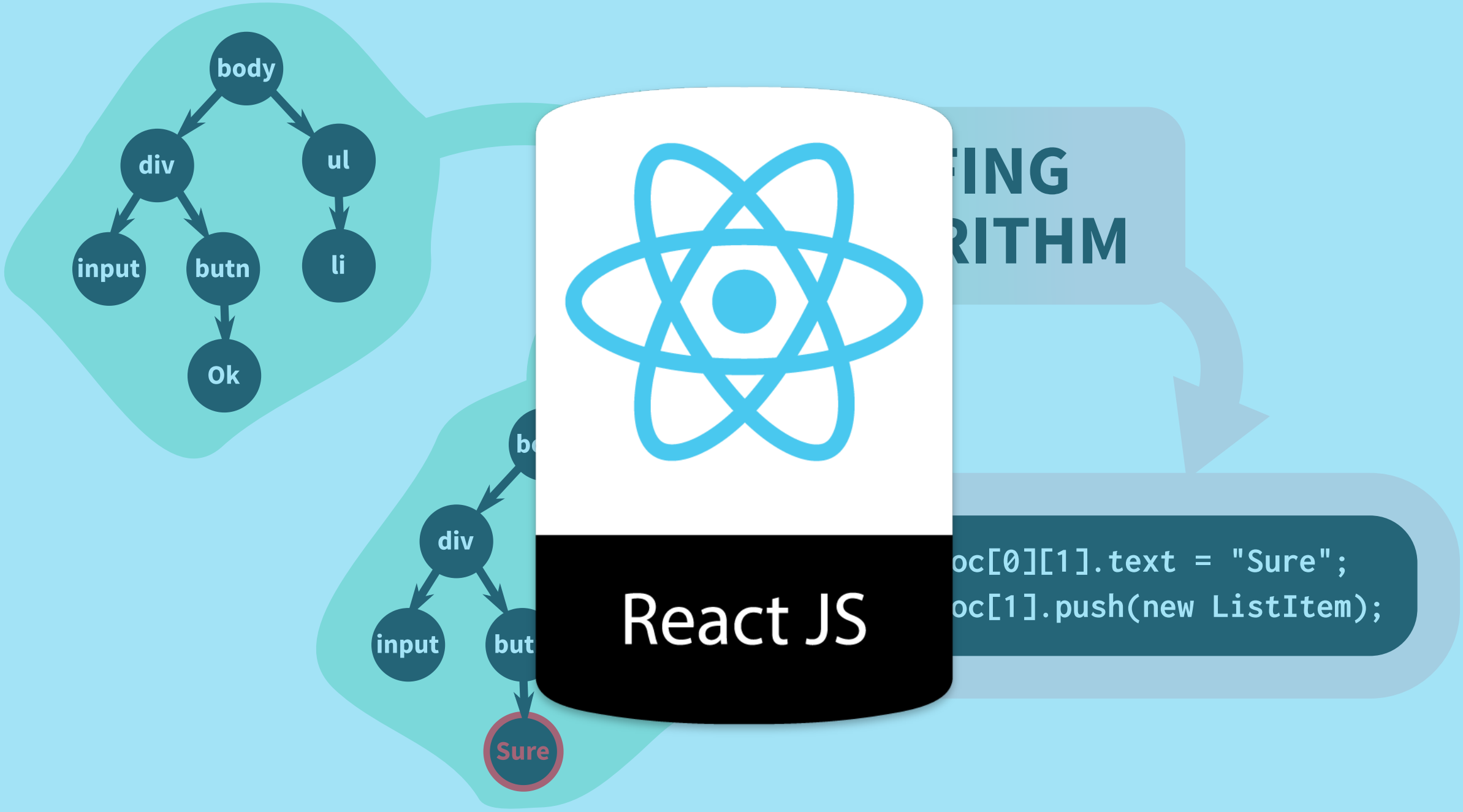
functional programming

constexpr

```
std::vector<int> push_back(std::vector<int> vec, int x)
{
    vec.push_back(x);
    return vec;
}
```

```
class foo {
    std::shared_ptr<impl> impl_;

public:
    foo modified(int arg) const& {;
        auto p = impl_->clone();
        p->mutate(arg);

        return foo{p};
    }
    foo&& modified(int arg) && {
        if (impl_->unique())          // unique() is not thread safe
            impl_->mutate(arg);       // according to std::shared_ptr.
        else                          // use your own implementation
            *this = modified();       // of reference counting
        return std::move(*this);
    }
};
```

```
class screen {
    screen_handle handle_;
    screen(const screen&) = delete;

public:
    screen&& draw(...) && {
        do_draw(handle_, ...);
    }

    auto draw(...) const {
        return [hdl=handle_] (context ctx) {
            do_draw(hdl, ...);
        };
    }
};
```


body
div
ul
input
butn
li
Ok
TING
RITHM
div
input
Sure
React JS
oc[0][1].text = "Sure";
oc[1].push(new ListItem);

WHEN TO USE VALUE SEMANTICS?

OBJECTS ⟶ **macro** design

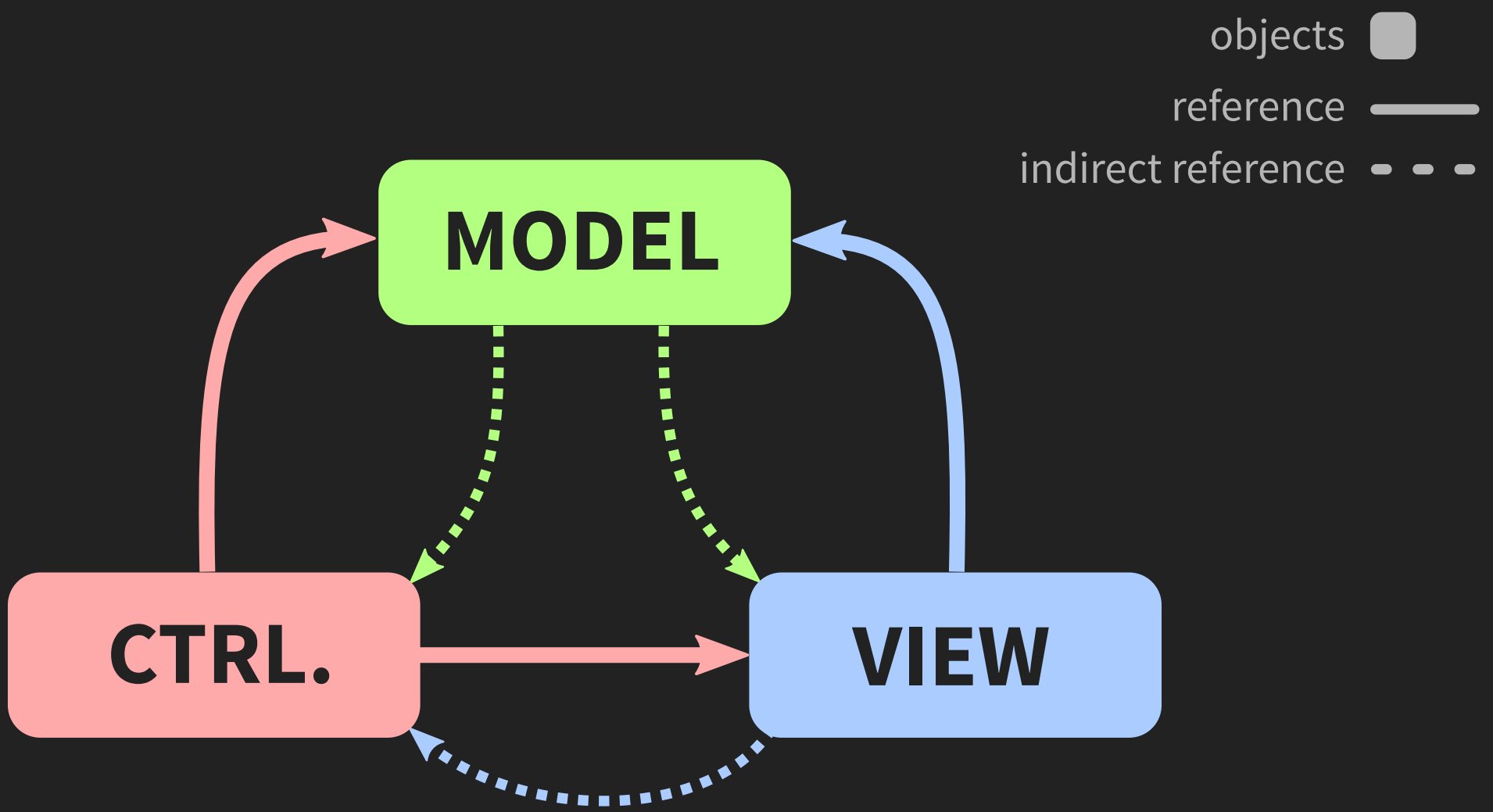
VALUES ⟶ **micro** design

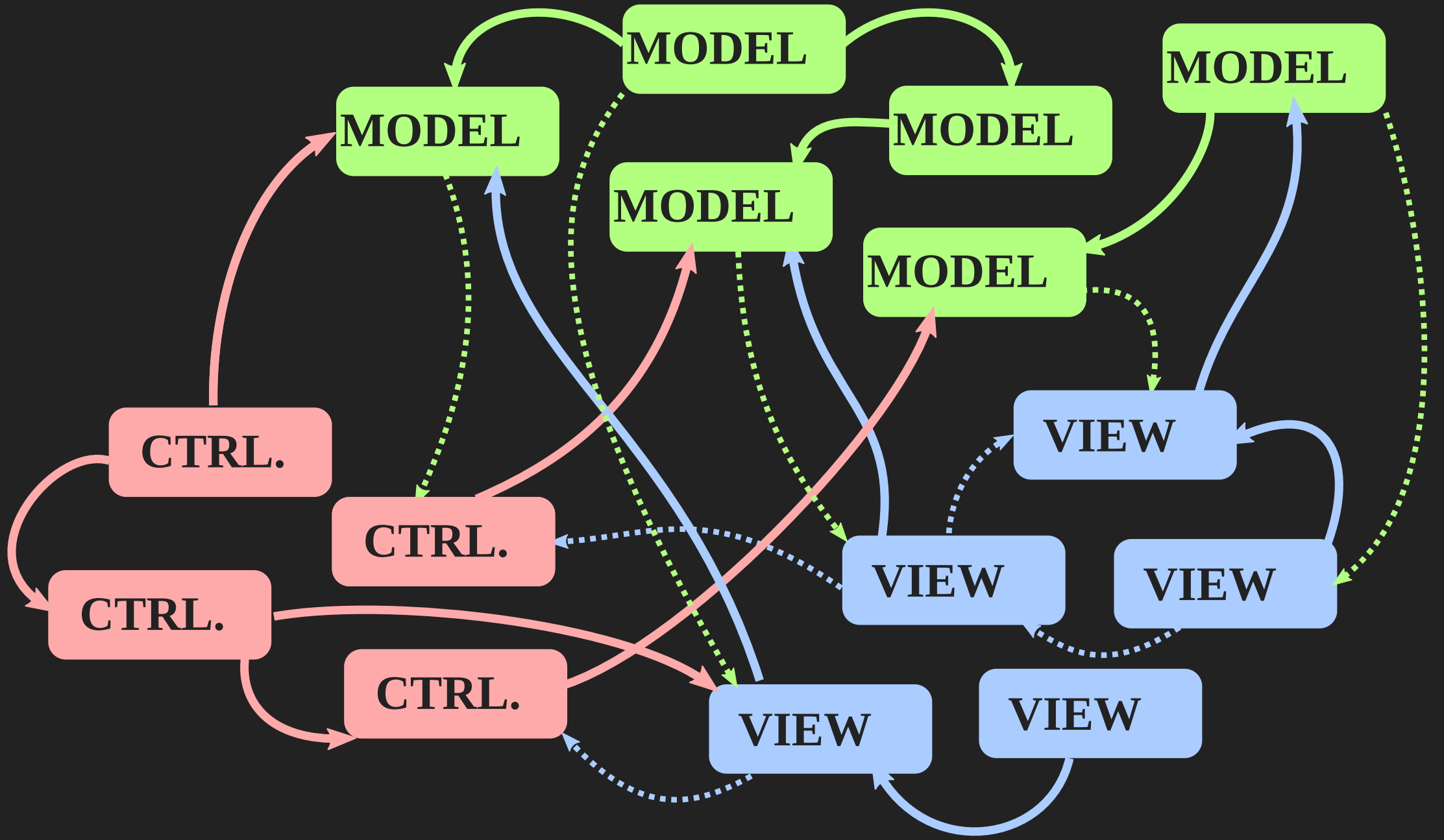
VALUES ⟶ **macro** design

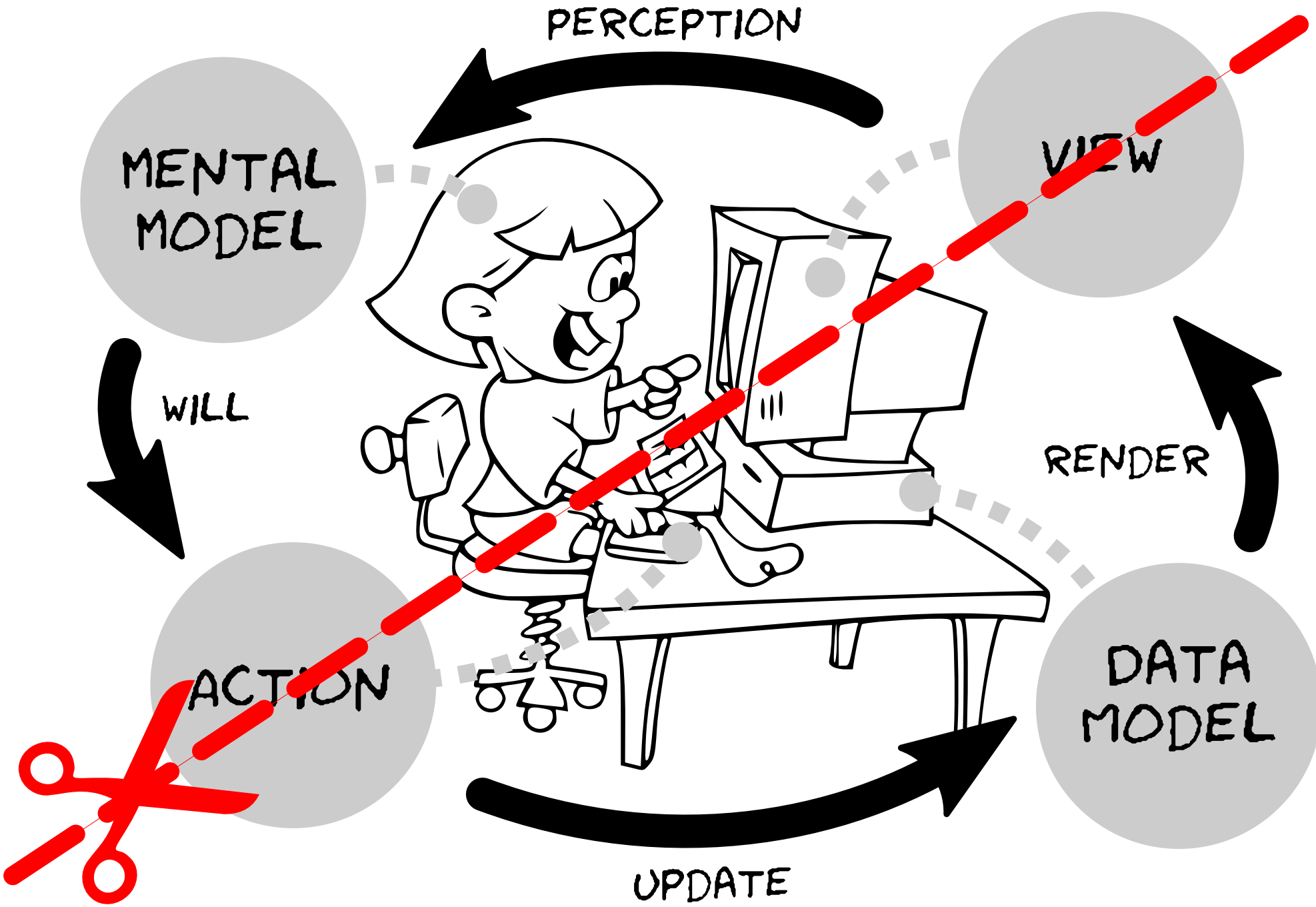
OBJECTS ⟶ **micro** design

PART II

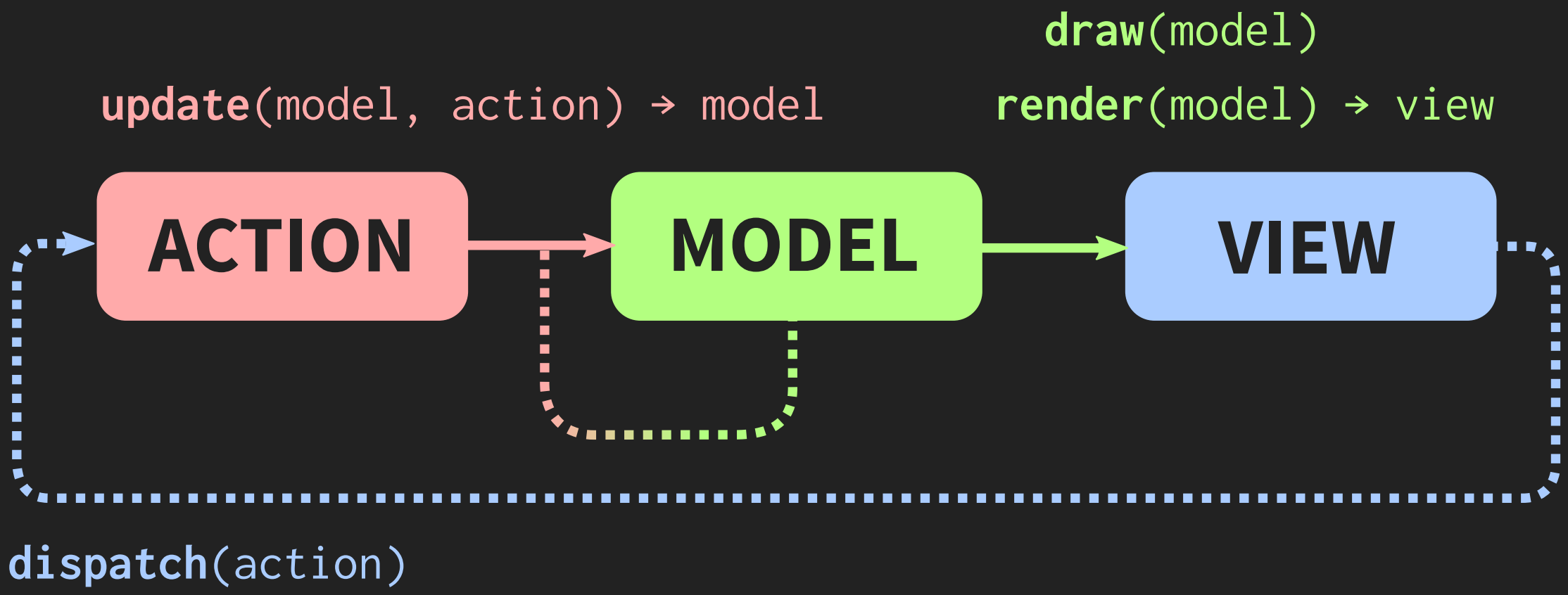
A VALUE BASED ARCHITECTURE FOR INTERACTIVE SOFTWARE


objects
reference
indirect reference
MODEL
CTRL.
VIEW


MODEL
MODEL
MODEL
MODEL
MODEL
MODEL
CTRL.
CTRL.
CTRL.
CTRL.
VIEW
VIEW
VIEW
VIEW
VIEW


PERCEPTION
MENTAL MODEL
VIEW
WILL
RENDER
ACTION
DATA MODEL
UPDATE

UNIDIRECTIONAL DATA FLOW ARCHITECTURE



EXAMPLE

AN INTERACTIVE COUNTER

```
struct model
{
    int value = 0;
};

struct increment_action {};
struct decrement_action {};
struct reset_action { int value = 0; };
```

```
model update(model c, action action)
{
  return std::visit(lager::visitor{
    [&] (increment_action) {
        return model{ c.value + 1 };
    },
    [&] (decrement_action) {
        return model{ c.value - 1 };
    },
```

```
void draw(counter::model c) {
    std::cout << "current value: "
              << c.value << '\n';
}

std::optional<action> intent(char ev) {
    switch (ev) {
    case '+':
        return increment_action{};
    case '-':
        return decrement_action{};
    case '.':
        return reset_action{};
    default:
        return std::nullopt;
    }
}
```

```
int main()
{
    auto state = counter{0};
    auto event = char{};
    while (std::cin >> event) {
        if (auto act = intent(event)) {
            state = update(state, *act);
            draw(state);
        }
    }
}
```

FRAMEWORKS

- **Elm** elm-lang.org

  Browser based Haskell for beginners

- **Redux** redux.js.org

  JavaScript data model library

- **Lager** github.com/arximboldi/lager

  Redux for C++!

```
void draw(model c) {  ... }
action intent(event_t ev) { ... }

int main() {
    auto debug = lager::http_debug_server{8080};
    auto serv  = boost::asio::io_service{};
    auto store = lager::make_store<action>(
        counter{0},
        update,
        draw,
        lager::boost_asio_event_loop{serv},
        lager::enable_debug(debug));
    auto term  = ncurses::terminal{serv};
    term.listen([&] (event_t ev) {
        store.dispatch(intent(ev));
    });
    serv.run();
}
```


MONTH
DAY
YEAR
PM
HOUR
MIN
OCT
26
1985
09
00
DESTINATION TIME
MONTH
DAY
YEAR
PM
HOUR
MIN
OCT
21
2015
07
28
PRESENT TIME
MONTH
DAY
YEAR
PM
HOUR
MIN
NOV
12
1955
06
38

```
template <typename Action, typename Model>
struct debugger
{
  struct goto_action { std::size_t cursor; };
  struct undo_action {};
  struct redo_action {};

  using action = std::variant<
      Action,
      goto_action,
      undo_action,
```

```
...
  struct model {
    Model init;
    std::size_t cursor = {};
    immer::vector<std::pair<Action, Model>>
        history = {};

    model(Model i) : init{i} {}
    operator const Model& () const {
      return cursor == 0
        ? init : history[cursor - 1].model;
```

```
model update(model m, action act, std::function<Model(Model, Action)> reducer)
{
    return std::visit(visitor{
            [&] (Action act) {
                ... reducer(m, act)
            },
            [&] (goto_action act) {

                ...
            },
            [&] (undo_action) {
                ...
            },
            [&] (redo_action) {
                ...
            },
    }, act);
}
```

```
int main() {
    auto debug = lager::http_debug_server{8080};
    auto meta  = lager::http_debug_server{8081};
    auto meta2 = lager::http_debug_server{8082};
    auto serv  = boost::asio::io_service{};
    auto store = lager::make_store<action>(
        counter{0},
        update,
        draw,
        lager::boost_asio_event_loop{serv},
        lager::comp(lager::enable_debug(debug),
                    lager::enable_debug(meta),
                    lager::enable_debug(meta2)));
    auto term  = ncurses::terminal{serv};
    term.listen([&] (event_t ev) {
        store.dispatch(intent(ev));
    });
    serv.run();
}
```

CLOSURE

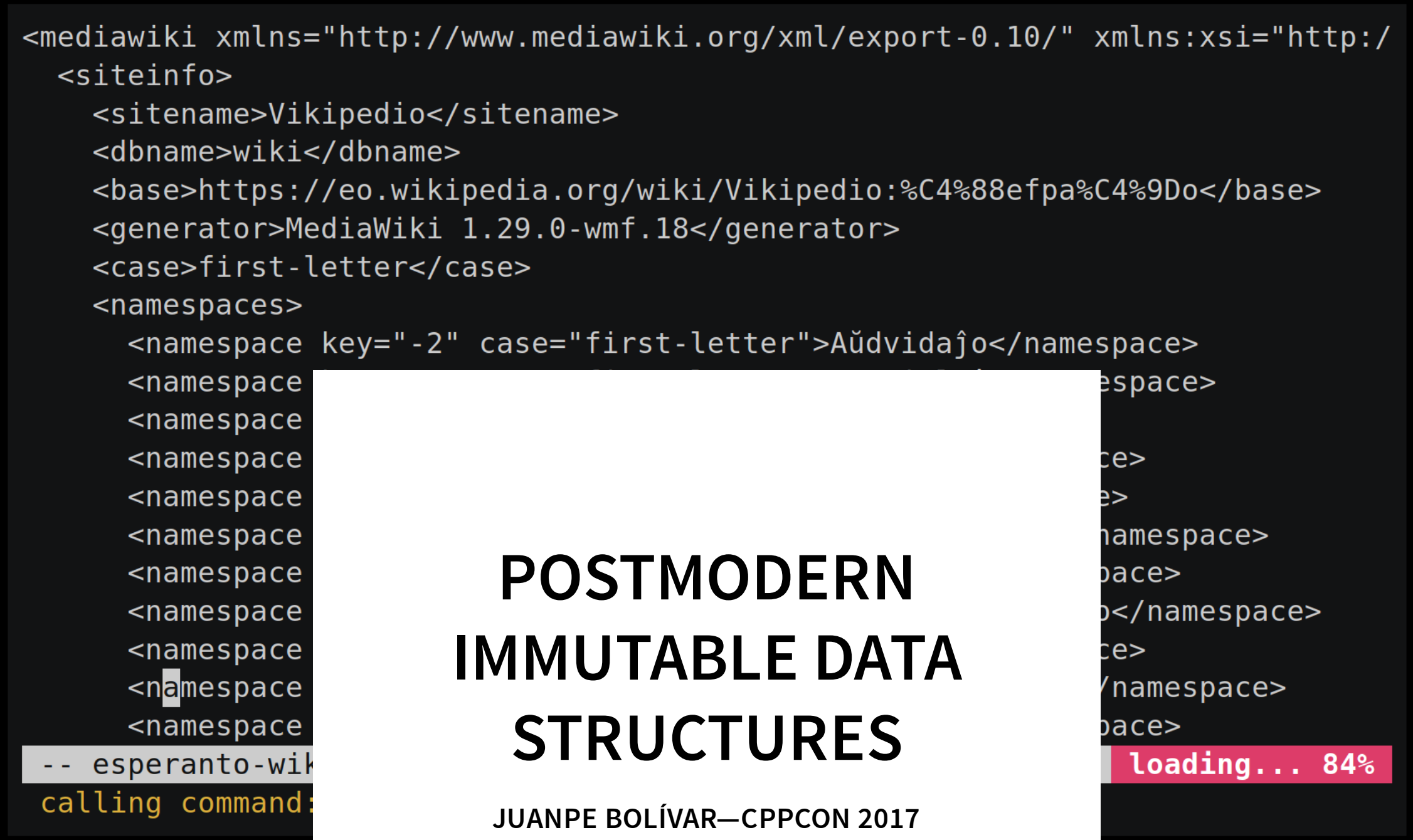
FURTHER TOPICS AND CONCLUSION

EFFECTS

```
model update(model, action);
```

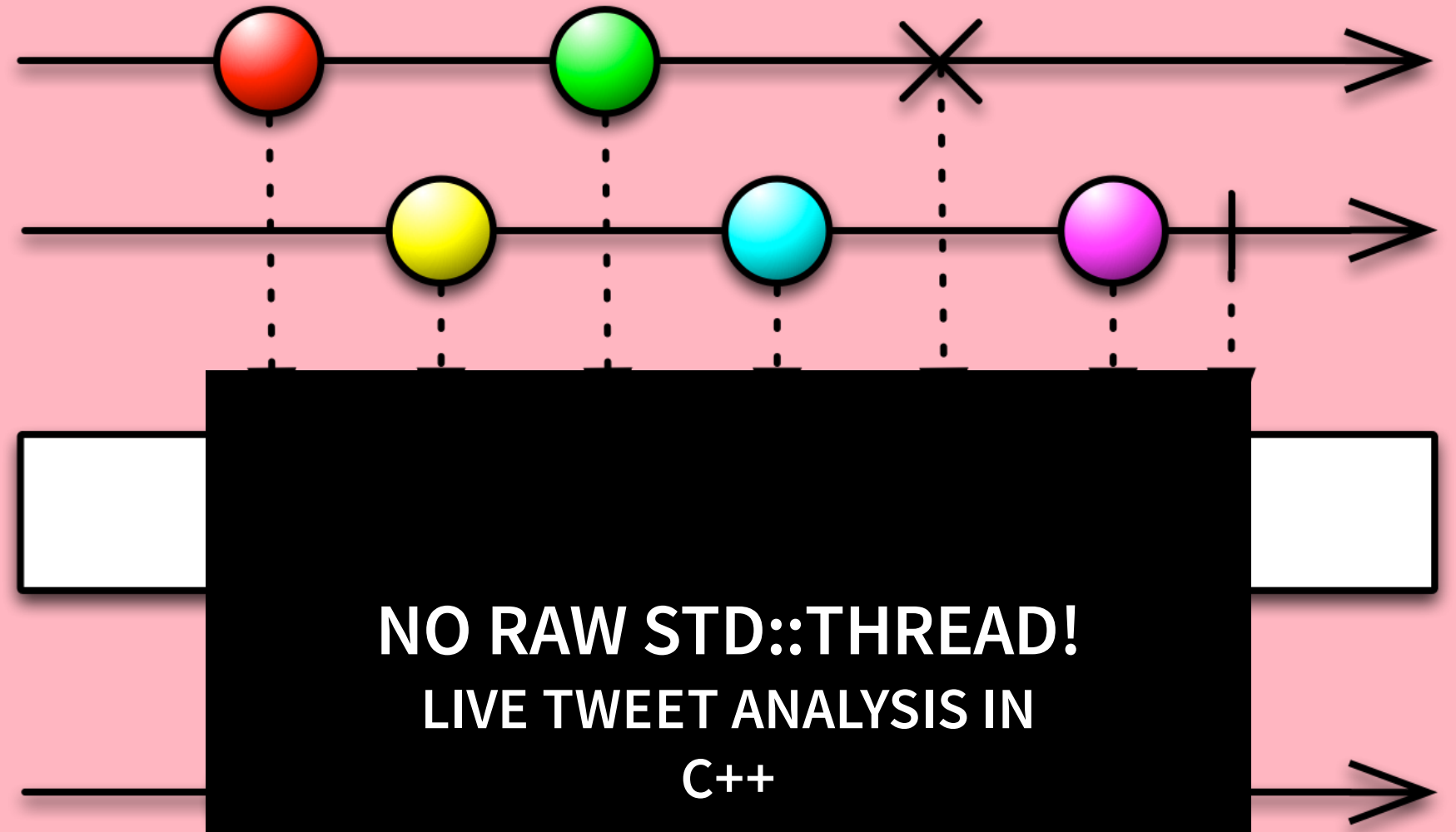
EFFECTS

```
template <typename Action>
using effect = std::function<void(context<Action>)>;

pair<model, effect<action>> update(model, action);
```



POSTMODERN IMMUTABLE DATA STRUCTURES

JUANPE BOLÍVAR—CPPCON 2017



NO RAW STD::THREAD!

LIVE TWEET ANALYSIS IN C++

KIRK SHOOP (RXCPP)—C++NOW 2017

youtu.be/1-8gxwGN3S8

VALUES ⟶ **macro** design

OBJECTS ⟶ **micro** design

https://github.com/arximboldi/lager
https://github.com/arximboldi/immer
https://github.com/arximboldi/ewig
https://sinusoid.al

thanks.