Nbdl

A library that uses metaprogramming... A lot

Overview

- What is Nbdl?
- Abstraction
- Concepts
- Store
- Promises
- Empty Interface
- Using Docker

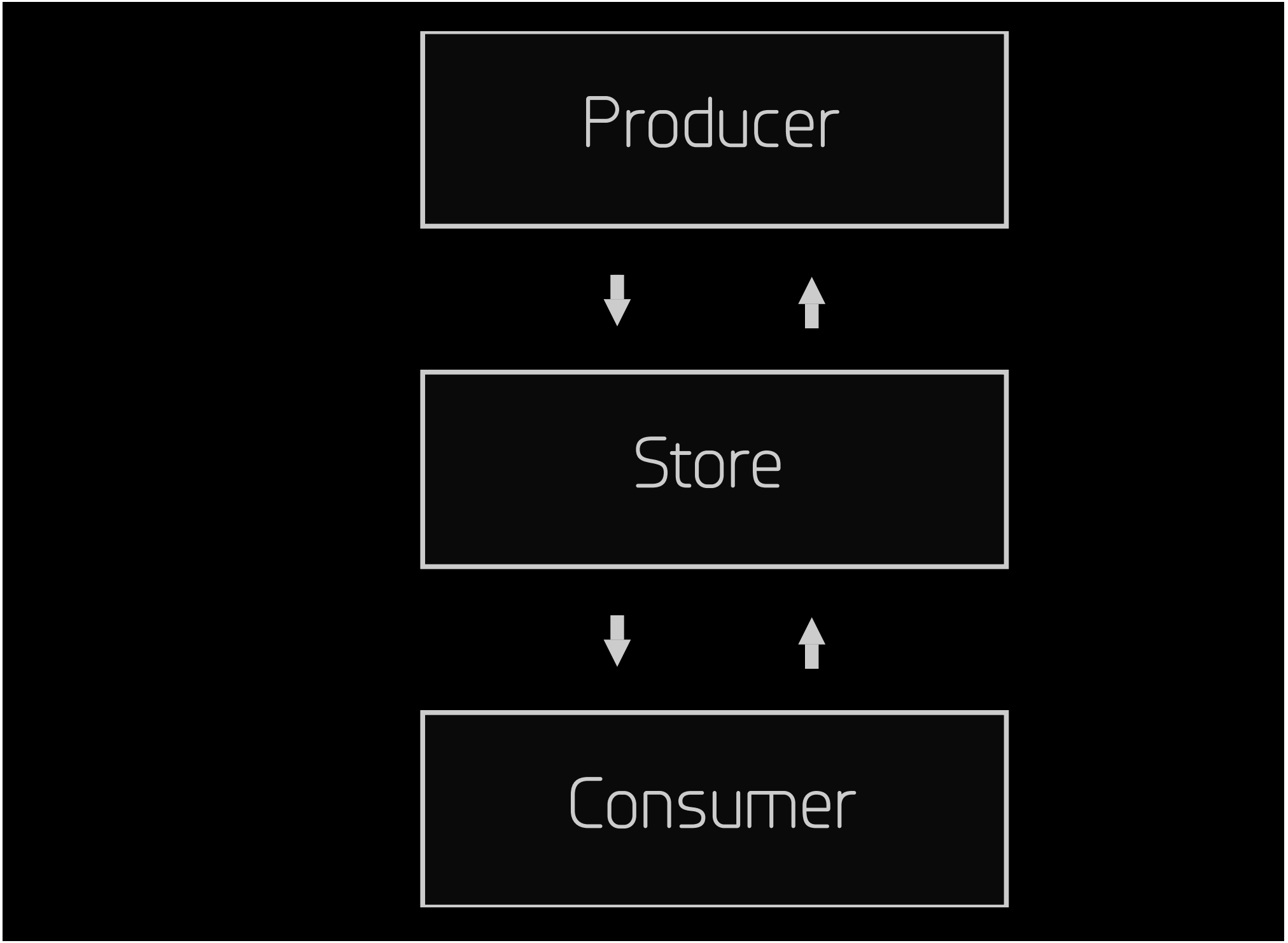
Readable code?

```
auto slide_container = [](auto index, auto child)
{
  constexpr std::size_t i = hana::value(index);

  // Don't do this!
  return div(
    attr_class(concat(
      "slide "_s
    , match(get(current_slide)
      , when<hana::size_t<i>>("current"_s)
      , when<hana::size_t<i + 1>>("prev"_s)
      , when<hana::size_t<i - 1>>("next"_s)
      , when<>(""_s)
      )
    ))
  , child
  );
};
```

What is Nbdl?

- Manages State
- Communicates Changes in State
- Provides implementations for common use cases


Producer
Store
Consumer

```
Context(
  Producers(
    Producer(
      Name("producer"_s)
    , Type(ProducerTag{})
    , AccessPoint(
        Name("current_slide"_s)
      , Store<cppnow17::current_slide_store>
      , Entity<current_slide_t>
      , PathKey<current_slide_tag>
      )
    , AccessPoint(
        Name("slide_action"_s)
      , Actions(Create())
      , Entity<slide_action_t>
      , PathKey<slide_action_tag>
      )
    )
  )
, Consumers(
    Consumer(
      Name("consumer"_s)
    , Type(ConsumerTag{})
    )
  , MoreConsumerDefs{}...
  )
);
```

Abstraction

Boost.Hana Style

hana::Monad

```
/*
 * @copyright Louis Dionne 2013-2017
 * Distributed under the Boost Software License, Version 1.0.
 */
template <typename M>
struct Monad
    : hana::integral_constant<bool,
        !is_default<flatten_impl<typename tag_of<M>::type>>::value ||
        !is_default<chain_impl<typename tag_of<M>::type>>::value
    >
{ };
```

hana::flatten_impl

```
/*
 * @copyright Louis Dionne 2013-2017
 * Distributed under the Boost Software License, Version 1.0.
 */

template <typename M, bool condition>
struct flatten_impl<M, when<condition>> : default_ {
    template <typename Xs>
    static constexpr auto apply(Xs&& xs)
    { return hana::chain(static_cast<Xs&&>(xs), hana::id); }
};
```

hana::flatten

```
/*
 * @copyright Louis Dionne 2013-2017
 * Distributed under the Boost Software License, Version 1.0.
 */

template <typename Xs>
constexpr auto flatten_t::operator()(Xs&& xs) const {
    using M = typename hana::tag_of<Xs>::type;
    using Flatten = BOOST_HANA_DISPATCH_IF(flatten_impl<M>,
        hana::Monad<M>::value
    );

    static_assert(hana::Monad<M>::value,
    "hana::flatten(xs) requires 'xs' to be a Monad");

    return Flatten::apply(static_cast<Xs&&>(xs));
}
```

hana::tag_of

```
static_assert(std::is_same<
  hana::tag_of_t<hana::tuple<int, char, double>>,
  hana::tuple_tag
>::value);
```

hana::tag_of

```
struct my_struct { };

static_assert(std::is_same<

  hana::tag_of_t<my_struct>,

  my_struct

>::value);
```

```cpp
struct my_list_tag { };

template <typename ...T>
struct my_list { };

namespace boost::hana
{
  template<typename ...T>
  struct tag_of<my_list<T...>>
  {
    using type = my_list_tag;
  };
}

static_assert(std::is_same<

  hana::tag_of_t<my_list<int, char, double>>,

  my_list_tag

>::value);
```

Concepts

Producer

- make_producer
- send_upstream_message

Echo Producer

```
struct echo_producer { };

template <typename ContextHandle>
struct echo_producer_impl
{
  using hana_tag = echo_producer;

  ContextHandle ctx;
};
```

make_producer

```
template <>
struct make_producer_impl<echo_producer>
{
  template <typename ContextHandle>
  static constexpr auto apply(ContextHandle&& h)
  {
    return echo_producer_impl<ContextHandle>{std::forward<ContextHandle>(h)};
  }
};
```

```
template <>
struct send_upstream_message_impl<echo_producer>
{
  template <typename Producer, typename Message>
  static constexpr void apply(Producer const& p, Message const& m)
  {
    if constexpr(message::is_create<Message>)
    {
      p.ctx.push(p.ctx.message_api().to_downstream_from_root(
        m,
        echo_producer_detail::make_unique_key(m)
      ));
    }
    else
    {
      p.ctx.push(p.ctx.to_downstream_from_root(m));
    }
  }
};
```

Consumer

- make_consumer
- send_downstream_message
- notify_state_change

Pushing Messages

```
...
    void right_arrow()
    {
      constexpr auto path = hana::transform(slide_action, hana::typeid_);

      ctx.push(
        ctx
        .message_api()
        .make_upstream_create_message(path, slide_action_next)
      );
    }
...
```

```
template <>
struct notify_state_change_impl<cppnow17::web::key_control> {
  template <typename Consumer, typename Path>
  static auto apply(Consumer const& c, Path const& path) {
    if constexpr(decltype(
      hana::typeid_(path) == hana::typeid_(cppnow17::current_slide)
    ){}) {
      nbdl::match(c.ctx, cppnow17::current_slide, [](auto value) {
        if constexpr(hana::typeid_(value) == hana::type_c<nbdl::unresolved>) {
          EM_ASM("console.log('current_slide: nbdl::unresolved');");
        }
        else {
          EM_ASM_(
            "console.log('current_slide: hana::size_c<' + $0 + '>');"
          , hana::value(value)
          );
        }
      });
    }
  }
};
```

Entity

- NBDL_ENTITY

```
struct person_t {
  std::string name_first;
  std::string name_last;
};

struct user_t {
  std::string username;
  person_t person;
};

namespace nbdl {
  NBDL_ENTITY(person_t
  , name_first
  , name_last
  );
  NBDL_ENTITY(user_t
  , username
  , person
  );
}
```

BindableSequence BindableMap

- bind_sequence
- bind_map

```
auto messages = hana::make_tuple(
  person_t{"John", "Smith"}
, user_t{"@JasonRice_", person_t{"Jason", "Rice"}}
, hana::make_tuple(
    "This"s , "is a tuple with"s , 6, "elements"s
  , "I'm a compile time string!"_s
  , "Last"s
  )
, hana::make_map(
    hana::make_pair("key_1"_s, "value1"s)
  , hana::make_pair("key_2"_s, "value2"s)
  , hana::make_pair("key_3"_s, hana::just(5))
  , hana::make_pair("key_4"_s, hana::nothing)
  , hana::make_pair("key_5"_s, hana::make_tuple(1, 2, 3, 4, "tuple"s))
  )
, terminate{}
);

std::cout << nbdl::binder::jsoncpp::to_string(messages);
```

```
[
   {
      "name_first" : "John",
      "name_last" : "Smith"
   },
   {
      "person" : {
         "name_first" : "Jason",
         "name_last" : "Rice"
      },
      "username" : "@JasonRice_"
   },
   [ "This", "is a tuple with", 6, "elements", "Last" ],
   {
      "key_1" : "value1",
      "key_2" : "value2",
      "key_3" : 5,
      "key_4" : null,
      "key_5" : [ 1, 2, 3, 4, "tuple" ]
   }
]
```

BindableVariant

- bind_variant

```
using my_var = nbdl::variant<
  hana::size_t<0>
, hana::size_t<1>
, int
, std::string
>;

nbdl::binder::jsoncpp::to_string(
  hana::make_tuple(
    my_var{std::string("hello")}
  , my_var{hana::size_c<0>}
  , my_var{hana::size_c<1>}
  , my_var{42}
  )
);
```

```
[
   [ 4, "hello" ],
   1,
   2,
   [ 3, 42 ]
]
```

Store

- match
- apply_action

hana::map as a Store

```
auto store = hana::make_map(
  hana::make_pair("key_1"_s, std::string("Hello, World!"))
, hana::make_pair("key_2"_s, 42)
);

nbdl::match(
  store
, "key_1"_s
, [](std::string const& x) { print(x); }
);

// Hello, World!
```

Entity as a Store

```
auto store = person_t{"John", "Smith"};

nbdl::match(
  store
, "name_first"_s
, [](std::string const& x) { print(x); }
);

// John
```

hana::Sequence as a Store

```
auto name_first = hana::type_c<name_first_t>;

auto store = hana::make_tuple(name_first_t{"John"}, name_last_t{"Smith"});

nbdl::match(
  store
, name_first
, [](name_first_t const& x) { print(x.value); }
);

// John
```

nbdl::get_impl

```
template<typename Tag>
struct match_impl<Tag, hana::when<!hana::is_default<nbdl::get_impl<Tag>>::value>>
{
  template <typename Store, typename Key, typename Fn>
  static constexpr void apply(Store&& s, Key&& k, Fn&& fn)
  {
    std::forward<Fn>(fn)(
      nbdl::get(std::forward<Store>(s), std::forward<Key>(k))
    );
  }
};
```

hana::Searchable get_impl

```
template<typename Tag>
struct get_impl<Tag, hana::when<hana::Searchable<Tag>::value>>
{
  template <typename Store, typename Key>
  static constexpr decltype(auto) apply(Store&& s, Key&& k)
  {
    if constexpr(hana::Sequence<Store>::value)
    {
      using Pred = decltype(hana::compose(hana::equal.to(hana::typeid_(k)), hana::typeid_));
      using Index = decltype(hana::index_if(s, Pred{}).value());
      return hana::at(std::forward<Store>(s), Index{});
    }
    else
    {
      return hana::at_key(std::forward<Store>(s), std::forward<Key>(k));
    }
  }
};
```

A variant as a Store

```
nbdl::optional<person_t> store = person_t{"John", "Smith"};

nbdl::match(
  store
, "name_first"_s
, [](auto const& x)
  {
    if constexpr(hana::is_a<std::string, decltype(x)>)
      print(x);
    else
      print("nothing");
  }
);

// John
```

variant identity

```
nbdl::optional<std::string> store = std::string("Hello, maybe!");

nbdl::match(
  store
, [](auto const& x)
  {
    if constexpr(hana::is_a<std::string, decltype(x)>)
      print(x);
    else
      print("nothing");
  }
);

// Hello, maybe!
```

- match_path

match_path

```
account_t account{
  std::string{"@JasonRice_"}
, person_t{hana::make_tuple(name_first{"Jason"}, name_last{"Rice"})}
};

nbdl::match_path(
  account
, hana::make_tuple("person"_s, "name"_s, hana::type_c<name_last>)
, [](name_last const& x) { print(x.value); }
, [](auto&&)             { print("nothing"); }
);

// Rice
```

match_path

```
account_t account{
  std::string{"@JasonRice_"}
, nbdl::nothing{}
};

nbdl::match_path(
  account
, hana::make_tuple("person"_s, "name"_s, hana::type_c<name_last>)
, [](name_last const& x) { print(x.value); }
, [](auto&&)             { print("nothing"); }
);

// nothing
```

Promises

- run_sync

path_promise

```
account_t account{
  username_t{"@JasonRice_"}
, person_t{hana::make_tuple(name_first_t{"Skippy"}, name_last_t{"McGee"})}
};

nbdl::run_sync(
  hana::make_tuple(
    nbdl::path_promise(get("person"_s, "name"_s, hana::type_c<name_last_t>))
  , require_type<name_last_t>
  , value_of, to_upper
  , nbdl::tap([](std::string const& str) { print(str); })
  , nbdl::catch_([](auto&&) { print("something else"); })
  )
, account
);

// MCGEE
```

path_promise... rejected

```
account_t account{
  username_t{"@JasonRice_"}
, nbdl::nothing{}
};

nbdl::run_sync(
  hana::make_tuple(
    nbdl::path_promise(get("person"_s, "name"_s, hana::type_c<name_last_t>))
  , require_type<name_last_t>
  , value_of, to_upper
  , nbdl::tap([](std::string const& str) { print(str); })
  , nbdl::catch_([](auto&&) { print("something else"); })
  )
, account
);

// something else
```

```
template <typename T>
auto require_type = nbdl::promise([](auto& resolver, auto const& value)
{
  if constexpr(decltype(hana::typeid_(value) == hana::type_c<T>){})
    resolver.resolve(value);
  else
    resolver.reject(value);
});
```

Eager 'index_if'

```
nbdl::run_sync(
  hana::make_tuple(
    hana::transform(hana::to_tuple(preds), predicate_promise)
  , [](auto const& ...args)
    {
      static_assert(
        sizeof...(args) > 9000
      , "nbdl::detail::match_if must have at least one predicate "
        "that returns compile-time Logical that is true."
      );
    }
  , nbdl::catch_([&](auto index) { resolve(index); })
  )
, value
, hana::size_c<0>
);
```

- run_async

Simple Server/Client

```
asio::io_service io;

nbdl::run_async(hana::make_tuple(
  example::accept(io, example::port({1234}))
, nbdl::tap(   [](auto&&...)  { std::cout << "Client connection accepted"; })
, nbdl::catch_([](auto&&)     { std::cout << "SERVER ERROR"; })
));

nbdl::run_async(hana::make_tuple(
  example::connect(io, example::port{1234})
, nbdl::tap(   [](auto&&)             { std::cout << "Connected to server"; })
, nbdl::catch_([](example::attempts) { std::cout << "failed with too many attempts"; })
, nbdl::catch_([](auto&&)             { std::cout << "CLIENT ERROR"; })
));

io.run();
```

No std::shared_ptr!

```
template <typename Resolve, typename ...Args>
void connect_fn::operator()(Resolve& resolver, Args&& ...) {
  socket.async_connect(endpoint, [&](asio::error_code error) {
    if (!error) {
      resolver.resolve(socket);
    }
    else if (attempts_failed++ >= attempts_.value) {
      resolver.reject(attempts_);
    }
    else {
      std::cout << "Connection failed! trying again...";
      operator()(resolver);
    }
  });
}
```

Unhandled Rejection!

```
...
  template <typename ...Args>
  void resolve(Args&&...)
  {
    delete end;
  }

  template <typename Arg1, typename ...Args>
  void reject(Arg1&&, Args&&...)
  {
    static_assert(
      sizeof...(Args) > 9000
    , "Unhandled Rejection!"
    );
  }
...
```

Empty UI

Empty... Stateless... Perfect

```
struct ding_bat { };
```

Html

```
auto bulleted_slide = [](auto header, auto ...bullets)
{
  return div(
    attr_class("slide-bulleted"_s)
  , div(
      attr_class("header"_s)
    , text_node(header)
    )
  , ul(li(text_node(bullets))...)
  );
};
```

Matching values in stores

```
auto my_store = hana::make_map(
  hana::make_pair("key_1"_s, std::string("UPDATE ME"))
, hana::make_pair("key_2"_s, std::string(" More dynamic text."))
, hana::make_pair("key_3"_s, nbdl::optional<std::string>{})
);

constexpr auto spec =
  div(attr_class("foo"_s)
  , div(attr_class("bar"_s)
    , text_node("I'm some static text content."_s)
    , text_node(get("key_1"_s))
    , text_node(get("key_2"_s))
    , text_node(" More static text."_s)
    , match(
        get("key_3"_s)
      , when<std::string>(text_node(get("key_3"_s)))
      , when<>(text_node("nothing"_s))
      )
    )
  );
```

Using Predicates

```
auto slide_container = [](auto index, auto child)
{
  constexpr std::size_t i = hana::value(index);

  return div(
    attr_class(concat(
      "slide "_s
    , match_if(get(current_slide)
      , cond(equal(current_slide_c<i>)     , "current"_s)
      , cond(equal(current_slide_c<i + 1>), "prev"_s)
      , cond(equal(current_slide_c<i - 1>), "next"_s)
      , otherwise(""_s)
      )
    ))
  , child
  );
};
```

Docker

Dependencies

```
...
# Boost.Hana (c++1z workaround branch)
RUN git clone -b bugfix/constexpr_arrays \
      https://github.com/ricejasonf/hana.git \
  && cp -r hana/include/* /usr/local/src/emscripten/system/include \
  && rm -rf hana
# Kvasir.Mpl
RUN git clone -b development \
      https://github.com/kvasir-io/mpl.git \
  && cp -r mpl/src/* /usr/local/include \
  && cp -r mpl/src/* /usr/local/src/emscripten/system/include \
  && rm -rf mpl
# Nbdl
RUN git clone -b cppnow17 \
      https://github.com/ricejasonf/nbdl.git \
  && cp -r nbdl/include/* /usr/local/src/emscripten/system/include \
  && rm -rf nbdl
...
```

Special Thanks

- Louis Dionne