Unit testing and error handling in C++ template metaprograms

Ábel Sinkovics
Endre Sajó

Agenda

- Introduction to C++ template metaprogramming
- Unit testing
- Testing against invalid input
- Compile-time exception handling

What is C++ TMP?

- Erwin Unruh, 1994
- Turing-complete
- Evaluated during compilation
- A pure functional language!

What good is C++ TMP?

- Expression templates
- Type system extensions
- Concept checking
- DSEL's

Template metafunction

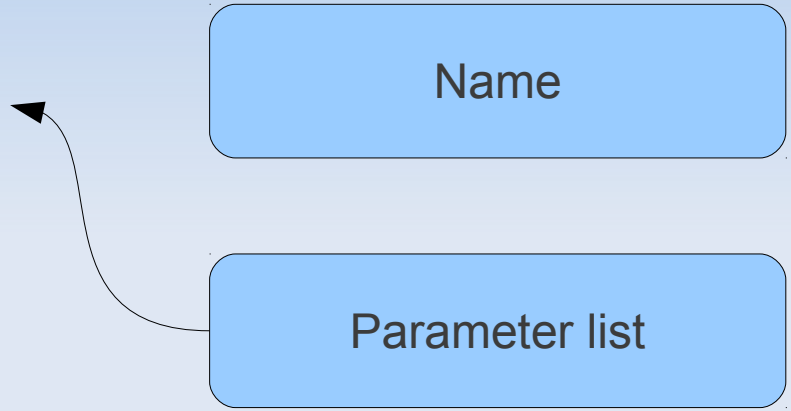
```
template<typename T>
  struct add_ptr
  {
    typedef T* type;
  };
```

Template metafunction

```
template<typename T>
  struct add_ptr
  {
    typedef T* type;
  };
```

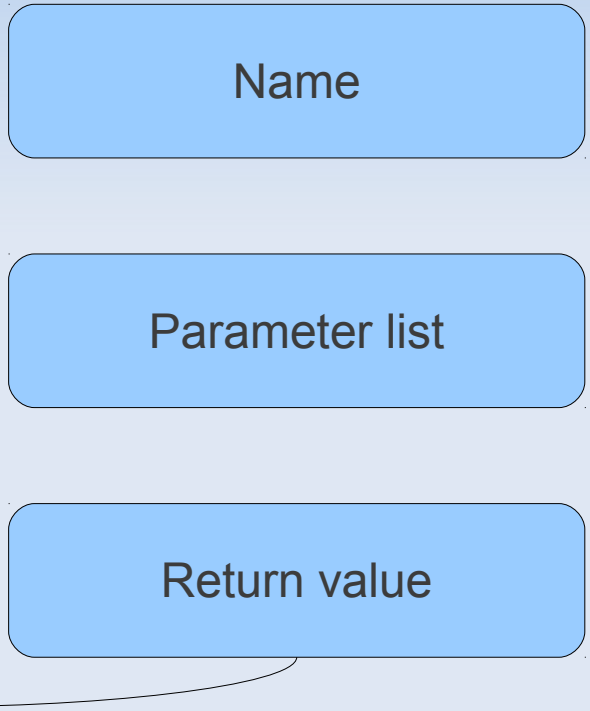
Template metafunction

```
template<typename T>
  struct add_ptr
  {
    typedef T* type;
  };
```



Template metafunction

```
template<typename T>
  struct add_ptr
  {
    typedef T* type;
  };
```



Template metafunction

```
template<typename T>
  struct add_ptr
  {
    typedef T* type;
  };
```

Name

Parameter list

Return value

Evaluating a metafunction

```
typedef
  add_ptr<int>::type
  ptr_to_int;
```

Evaluating a metafunction

```
typedef
  add_ptr<int>::type
  ptr_to_int;

typedef
  add_ptr<int>
  ptr_to_int_nullary;
```

Template metafunction class

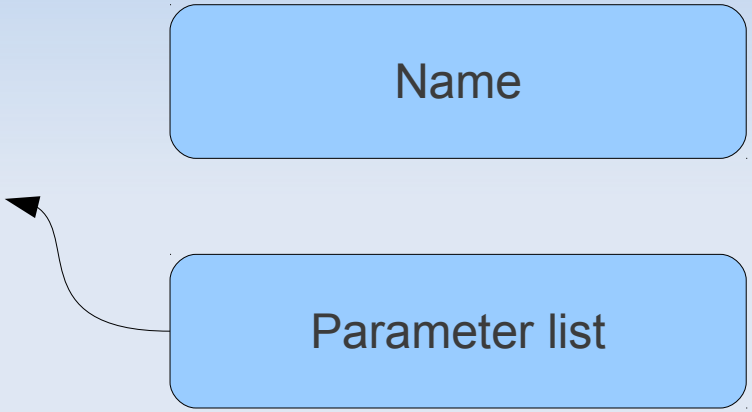
```
struct add_ptr_f
{
  template<typename T>
    struct apply
    {
      typedef T* type;
    };
};
```

Template metafunction class

```
struct add_ptr_f
{
  template<typename T>
    struct apply
    {
      typedef T* type;
    };
};
```

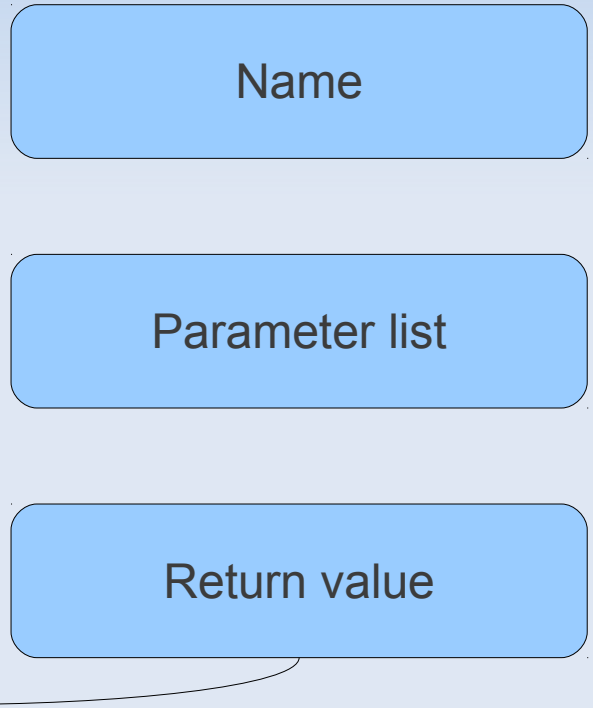
Template metafunction class

```
struct add_ptr_f
{
  template<typename T>
    struct apply
    {
      typedef T* type;
    };
};
```



Template metafunction class

```
struct add_ptr_f
{
  template<typename T>
    struct apply
    {
      typedef T* type;
    };
};
```



Template metafunction class

```
struct add_ptr_f
{
  template<typename T>
    struct apply
    {
      typedef T* type;
    };
};
```

Name

Parameter list

Return value

Template metafunction class

```
typedef
  mpl::apply<add_ptr_f, int>::type
  ptr_to_int;
```

Boxed value

```
template<int N>
  struct int_
  {
    static const int value;
    typedef int_ type;
  };

template<int N>
const int int_<N>::value = N;
```

Boxed value

```
template<int N>
  struct int_
  {
    static const int value;
    typedef int_ type;
  };

typedef int_<3> three;
```

Tag dispatch

```
template<int N>
struct int_ {
  static const int value;

};
```

```
struct int_tag {};
```

```
template<typename T>
struct negate {
  typedef int_<-T::value> type;
};
```

Tag dispatch

```
template<int N>
struct int_ {
  static const int value;
  typedef int_tag tag;
};
```

```
struct int_tag {};
```

```
template<typename T>
struct negate {
  typedef int_<-T::value> type;
};
```

Tag dispatch

```
template<int N>
struct int_ {
  static const int value;
  typedef int_tag tag;
};
```

```
struct int_tag {};
```

```
template<typename T>
struct negate {
  typedef int_<-T::value> type;
};
```

Tag dispatch

```
template<typename Tag>
  struct negate_impl;
```

```
struct int_tag {};
```

Tag dispatch

```
template<typename Tag>
  struct negate_impl;


template<>
  struct negate_impl<int_tag> {



  };
```

```
struct int_tag {};
```

Tag dispatch

```
template<typename Tag>
  struct negate_impl;
```

```
struct int_tag {};
```

```
template<>
  struct negate_impl<int_tag> {
    template<typename T>
    struct apply
    { typedef int_<-T::value> type; };
  };
```

Tag dispatch

```
struct int_tag {};
```

```
template<typename T>
  struct negate
    : mpl::apply<
          negate_impl<typename T::tag>
        , T
      >
```

Unit testing in Boost.MPL

A function to test

```
template<typename A, typename B>
  struct min

  {};
```

A function to test

```
template<typename A, typename B>
  struct min
    : if_<
        typename less<A, B>::type
      , A
      , B
      >
  {};
```

Boost.MPL assert macros

- BOOST_MPL_ASSERT
- BOOST_MPL_ASSERT_NOT
- BOOST_MPL_ASSERT_MSG
- BOOST_MPL_ASSERT_RELATION

Unit test in Boost.MPL

```
min< int_<5>, int_<7> >::type
int_<5>
```

Unit test in Boost.MPL

```
is_same<
  min< int_<5>, int_<7> >::type
, int_<5>
>
```

Unit test in Boost.MPL

```
BOOST_MPL_ASSERT((
  is_same<
    min< int_<5>, int_<7> >::type
  , int_<5>
  >
));
```

Unit test in Boost.MPL

```
MPL_TEST_CASE()
{
  BOOST_MPL_ASSERT((
    is_same<
      min< int_<5>, int_<7> >::type
    , int_<5>
    >
  ));
}
```

Failed test in GCC

```
mintest.cpp: In function 'void test9()':
mintest.cpp:11:3: error: no matching
function for call to
'assertion_failed(mpl_::failed************
boost::is_same<mpl_::int_<6>,
mpl_::int_<5> >::************)'
```

Failed test in LLVM Clang

```
mintest.cpp:11:3: error: no matching function for call to 'assertion_failed'
  MPL_ASSERT((
  ^~~~~~~~~~~~
In file included from mintest.cpp:1:
In file included from /usr/include/boost/mpl/aux_/test.hpp:19:
/usr/include/boost/mpl/aux_/test/assert.hpp:20:41: note: instantiated from:
#define MPL_ASSERT(pred)                BOOST_MPL_ASSERT(pred)
                                        ^
In file included from mintest.cpp:1:
In file included from /usr/include/boost/mpl/aux_/test.hpp:19:
In file included from /usr/include/boost/mpl/aux_/test/assert.hpp:17:
/usr/include/boost/mpl/assert.hpp:217:32: note: instantiated from:
#define BOOST_MPL_ASSERT(pred) \
                               ^
mintest.cpp:11:3: note: instantiated from:
  MPL_ASSERT((
  ^
In file included from mintest.cpp:1:
In file included from /usr/include/boost/mpl/aux_/test.hpp:19:
/usr/include/boost/mpl/aux_/test/assert.hpp:20:41: note: instantiated from:
#define MPL_ASSERT(pred)                BOOST_MPL_ASSERT(pred)
                                        ^
In file included from mintest.cpp:1:
In file included from /usr/include/boost/mpl/aux_/test.hpp:19:
In file included from /usr/include/boost/mpl/aux_/test/assert.hpp:17:
/usr/include/boost/mpl/assert.hpp:221:11: note: instantiated from:
          boost::mpl::assertion_failed<false>( \
          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/boost/mpl/assert.hpp:79:5: note: candidate function [with C =
false] not viable: no known conversion from 'mpl_::failed
************(boost::is_same<mpl_::int_<6>, mpl_::int_<5> >::************)' to
'typename assert<false>::type' (aka 'mpl_::assert<false>') for 1st argument
int assertion_failed( typename assert<C>::type );
    ^
1 error generated.
make: *** [mintest.o] Error 1
```

Metatest

Unit tests in metatest

- Type pretty-printing:

```
to_stream<UDT>::run(std::cout);
```

Writing `to_stream`-aware classes

- Specialize metafunction `to_stream`

Writing `to_stream`-aware classes

- Specialize metafunction `to_stream`

```
template<> struct to_stream<UDT>
{




};
```

Writing `to_stream`-aware classes

- Specialize metafunction `to_stream`

```
template<> struct to_stream<UDT>
{
   static ostream& run(ostream &os)
   {

   }
};
```

Writing `to_stream`-aware classes

- Specialize metafunction `to_stream`

```
template<> struct to_stream<UDT>
{
  static ostream& run(ostream &os)
  {
    return os << "UDT";
  }
};
```

Writing `to_stream`-aware classes

- Specialize tag-dispatched metafunction `to_stream_impl`

Writing `to_stream`-aware classes

- Specialize tag-dispatched metafunction `to_stream_impl`

```
template<>
struct to_stream_impl<UDT_tag>
{





};
```

Writing `to_stream`-aware classes

- Specialize tag-dispatched metafunction `to_stream_impl`

```
template<>
struct to_stream_impl<UDT_tag>
{
  template<typename T> struct apply
  {



  };
};
```

Writing `to_stream`-aware classes

- Specialize tag-dispatched metafunction `to_stream_impl`

```
template<>
struct to_stream_impl<UDT_tag>
{
  template<typename T> struct apply
  {
    static ostream& run(ostream &os)
    {                                }
  };
};
```

Writing `to_stream`-aware classes

- Specialize tag-dispatched metafunction `to_stream_impl`

```
template<>
struct to_stream_impl<UDT_tag>
{
  template<typename T> struct apply
  {
    static ostream& run(ostream &os)
    { return os << "UDT"; }
  };
};
```

Unit test in Metatest

```
is_same<
    min< int_<5>, int_<7> >::type
  , int_<5> >
```

Unit test in Metatest

```
typedef
  is_same<
      min< int_<5>, int_<7> >::type
    , int_<5> >
  test1;
```

Unit test in Metatest

```
const suite_path suite =
    suite_path(”sample”)(”suite”);

typedef
  is_same<
      min< int_<5>, int_<7> >::type
    , int_<5> >
  test1;
```

Unit test in Metatest

```
const suite_path suite =
    suite_path(”sample”)(”suite”);

typedef
  is_same<
      min< int_<5>, int_<7> >::type
    , int_<5> >
  test1;

ADD_TEST(suite, test1)
```

Test results

- Test result = success/fail status + reason
- Test suite structure
- Singleton test driver
- Iterator interface

Test results

- Test result = success/fail status + reason
- Test suite structure
- Singleton test driver
- Iterator interface

- Ready-to-use report generators:
  Boost.Test, plain text, html, xml

Plain text report

- Passed:

```
the following tests have been executed:
  sample::suite::test1 : OK
==========================
Number of tests: 1
Number of failures: 0
```

Plain text report

- Failed:

```
the following tests have been executed:
  sample::suite::test1 : FAIL
(mintest.cpp:16)
         is_same<int_<6>, int_<5>>
==========================
Number of tests: 1
Number of failures: 0
```

Error propagation

Example

```
template <class A, class B>
struct min :

{};
```

Example

```
template <class A, class B>
struct min :
  boost::mpl::if_<                    >
{};
```

Example

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>,     >
{};
```

Example

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

Example

```
template <class A, class B>
struct less;
```

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

Example

```
template <class A, class B>
struct less;
```

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

```
template <class Re, class Im>
struct complex;
```

Example

```
template <class A, class B>
struct less;
```

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Example

```
template <class A, class B>
struct less;
```

```
template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B
{};
```

```
test.cpp: In instantiation of ‘less<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’:
/usr/include/boost/mpl/if.hpp:67:11:   instantiated from
‘boost::mpl::if_<less<complex<mpl_::int_<19>, mpl_::int_<83> >,
complex<mpl_::int_<11>, mpl_::int_<13> > >, complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’
test.cpp:13:36:   instantiated from ‘min<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’
test.cpp:21:68:   instantiated from here
test.cpp:10:44: error: ‘value’ is not a member of
‘complex<mpl_::int_<11>, mpl_::int_<13> >’
test.cpp: In instantiation of ‘less<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’:
/usr/include/boost/mpl/if.hpp:67:11:   instantiated from
‘boost::mpl::if_<less<complex<mpl_::int_<19>, mpl_::int_<83> >,
complex<mpl_::int_<11>, mpl_::int_<13> > >, complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’
test.cpp:13:36:   instantiated from ‘min<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’
test.cpp:21:68:   instantiated from here
test.cpp:10:44: error: ‘value’ is not a member of
‘complex<mpl_::int_<19>, mpl_::int_<83> >’
In file included from test.cpp:1:0:
/usr/include/boost/mpl/if.hpp: In instantiation of
‘boost::mpl::if_<less<complex<mpl_::int_<19>, mpl_::int_<83> >,
complex<mpl_::int_<11>, mpl_::int_<13> > >, complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’:
test.cpp:13:36:   instantiated from ‘min<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >’
test.cpp:21:68:   instantiated from here
/usr/include/boost/mpl/if.hpp:67:11: error: ‘value’ is not a member of
‘less<complex<mpl_::int_<19>, mpl_::int_<83> >, complex<mpl_::int_<11>,
mpl_::int_<13> > >’
/usr/include/boost/mpl/if.hpp:70:41: error: ‘value’ is not a member of
‘less<complex<mpl_::int_<19>, mpl_::int_<83> >, complex<mpl_::int_<11>,
mpl_::int_<13> > >’
```

```
template <class Re, cla
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Testing against invalid input

- How to test a template metafunction's behaviour for invalid input?
  - Emit a compilation error
    - Failed compilations mean passed tests
    - The build system has to be prepared for it

Testing against invalid input

- How to test a template metafunction's behaviour for invalid input?
  - Emit a compilation error
    - Failed compilations mean passed tests
    - The build system has to be prepared for it
  - Return some value and throw an exception at runtime
    - What value to return? Will it compile?
    - When to throw an exception?

Testing against invalid input

- How to test a template metafunction's behaviour for invalid input?
  - Emit a compilation error
    - Failed compilations mean passed tests
    - The build system has to be prepared for it
  - Return some value and throw an exception at runtime
    - What value to return? Will it compile?
    - When to throw an exception?
  - Return a special value representing an error

Returning errors

```
template <class Reason>
struct exception
{
  typedef exception type;
};
```

Example

```
template <class A, class B>
struct less :
  // returns either bool_<...>
  // or exception<values_can_not_be_compared>
{};
```

Example

```
template <class A, class B>
struct less :
  // returns either bool_<...>
  // or exception<values_can_not_be_compared>
{};

template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

Example

```
template <class A, class B>
struct less :
  // returns either bool_<...>
  // or exception<values_can_not_be_compared>
{};

template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Example

```
template <class A, class B>
struct less :
  // returns either bool_<..
  // or exception<values_can_
{};

template <class A, class B>
struct min :
  boost::mpl::if_<less<A, B>, A, B>
{};
```

```
In file included from test2.cpp:1:0:
/usr/include/boost/mpl/if.hpp: In instantiation of
'boost::mpl::if_<less<complex<mpl_::int_<19>, mpl_::int_<83> >,
complex<mpl_::int_<11>, mpl_::int_<13> > >, complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >':
test2.cpp:21:36:   instantiated from 'min<complex<mpl_::int_<19>,
mpl_::int_<83> >, complex<mpl_::int_<11>, mpl_::int_<13> > >'
test2.cpp:29:68:   instantiated from here
/usr/include/boost/mpl/if.hpp:67:11: error: 'value' is not a member of
'less<complex<mpl_::int_<19>, mpl_::int_<83> >, complex<mpl_::int_<11>,
mpl_::int_<13> > >'
/usr/include/boost/mpl/if.hpp:70:41: error: 'value' is not a member of
'less<complex<mpl_::int_<19>, mpl_::int_<83> >, complex<mpl_::int_<11>,
mpl_::int_<13> > >'
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :

    boost::mpl::if_<less<A, B>, A, B>

{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<


    boost::mpl::if_<less<A, B>, A, B>
  >
{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,

    boost::mpl::if_<less<A, B>, A, B>
  >
{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};
```

```
exception<values_can_not_be_compared>
```

```
template <class Re, class Im>
struct complex;

min<
  complex<int_<19>, int_<83> >, // 19 + 83i
  complex<int_<11>, int_<13> >  // 11 + 13i
>
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};


struct min_impl
{
  template <class LessAB>
  struct apply :                                        {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};


struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::if_<less<A, B>, A, B>
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
                        min_impl<A, B>
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::apply<min_impl<A, B>, less<A, B> >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::apply<min_impl<A, B>, less<A, B> >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class A, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<less<A, B> >::type,
    less<A, B>,
    boost::mpl::apply<min_impl<A, B>, less<A, B> >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class X, class B>
struct min :
  boost::mpl::eval_if<
    typename is_exception<    X      >::type,
        X     ,
    boost::mpl::apply<min_impl<A, B>,     X      >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class X, class F>
struct min :
  boost::mpl::eval_if<
    typename is_exception<    X      >::type,
        X     ,
    boost::mpl::apply<      F        ,     X      >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class X, class F>
struct bind_exception :
  boost::mpl::eval_if<
    typename is_exception<    X      >::type,
        X     ,
    boost::mpl::apply<      F        ,     X      >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl::if_<LessAB, A, B> {};
};
```

Error propagation

```
template <class X, class F>
struct bind_exception :
  boost::mpl::eval_if<
    typename is_exception<    X      >::type,
        X     ,
    boost::mpl::apply<      F       ,     X      >
  >
{};

template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl:
};
```

```
template <class A, class B>
struct min :
  bind_exception<
    less<A, B>,
    min_impl<A, B>
  >
{};
```

Error propagation

```
templa
struc
  boo
    t

    b
  >
{};
```

min<A, B>

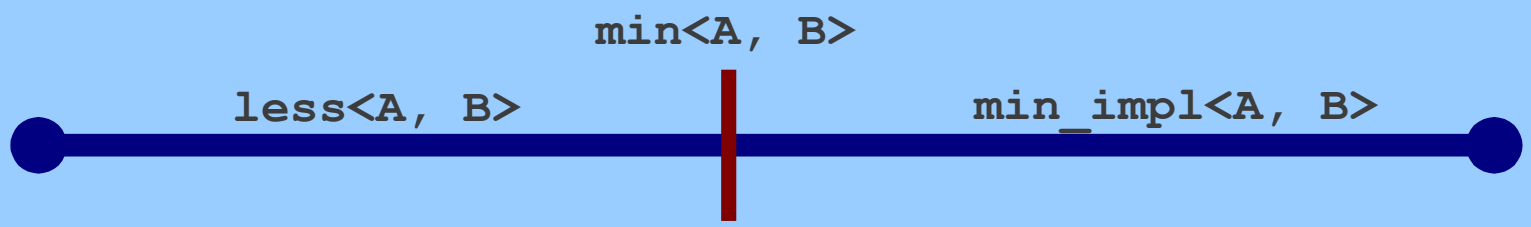
```
template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl:
};
```

```
template <class A, class B>
struct min :
  bind_exception<
    less<A, B>,
    min_impl<A, B>
  >
{};
```

Error propagation

```
templa
struc
  boo
    t

    b
  >
{};
```



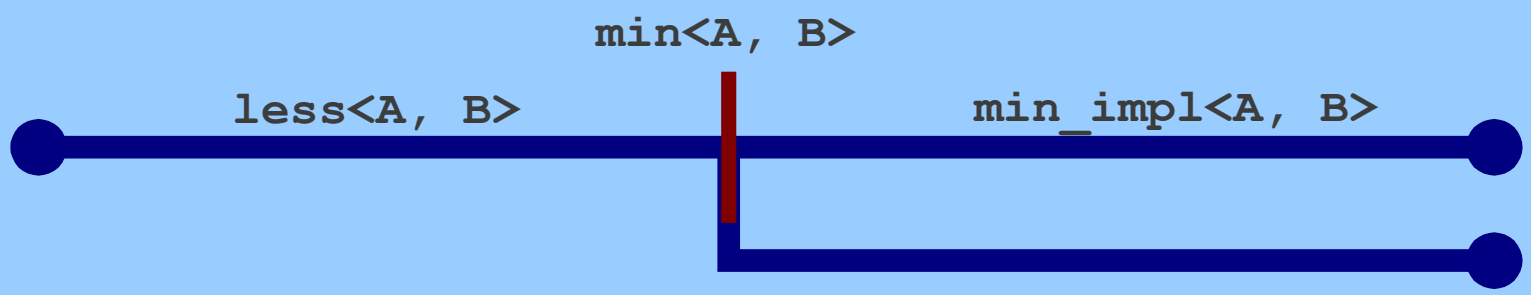

```
template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl:
};
```

```
template <class A, class B>
struct min :
  bind_exception<
    less<A, B>,
    min_impl<A, B>
  >
{};
```

Error propagation

```
templa
struc
  boo
    t

    b
  >
{};
```



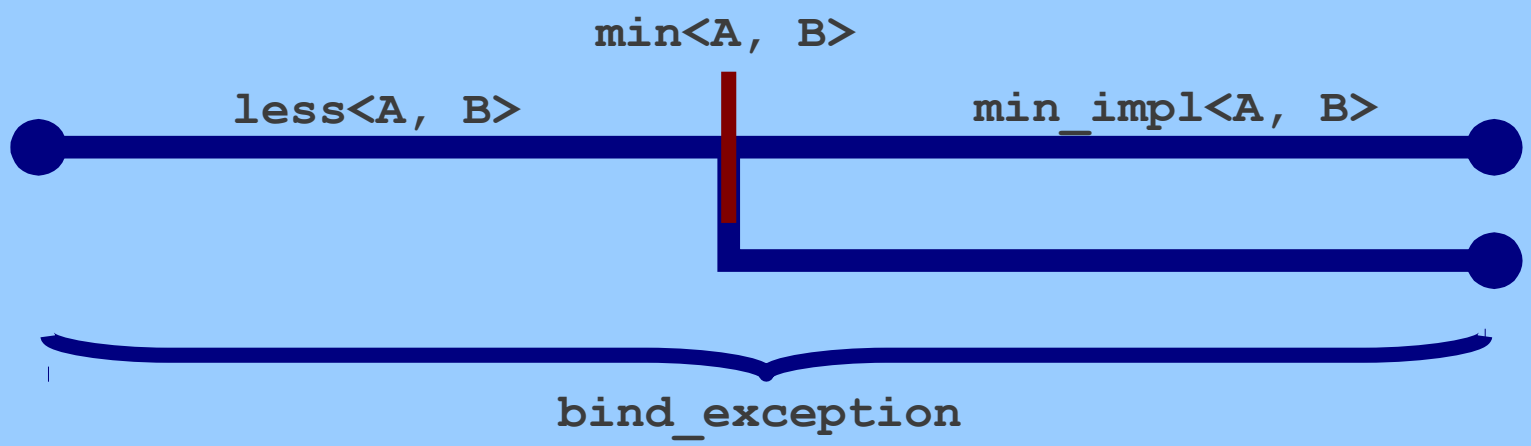

```
template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl:
};
```

```
template <class A, class B>
struct min :
  bind_exception<
    less<A, B>,
    min_impl<A, B>
  >
{};
```

Error propagation

```
templa
struc
  boo
    t

    b
  >
{};
```




```
template <class A, class B>
struct min_impl
{
  template <class LessAB>
  struct apply : boost::mpl:
};
```

```
template <class A, class B>
struct min :
  bind_exception<
    less<A, B>,
    min_impl<A, B>
  >
{};
```

Error propagation

- A higher-order function:

```
template <class X, class F>
struct bind_exception;
```

Error propagation

- A higher-order function:

```
error/result ×                                  →
```

```
template <class X, class F>
struct bind_exception;
```

Error propagation

- A higher-order function:

```
error/result × (value → error/result) →
```

```
template <class X, class F>
struct bind_exception;
```

Error propagation

- A higher-order function:

```
error/result × (value → error/result) → error/result
```

```
template <class X, class F>
struct bind_exception;
```

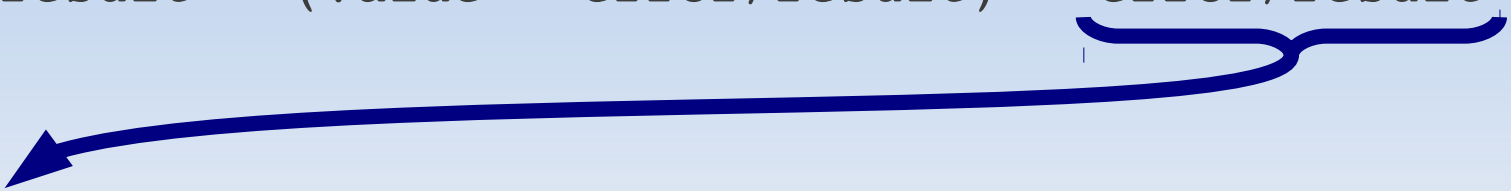
Error propagation

- Chaining functions
  `error/result × (value → error/result) → error/result`

Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```



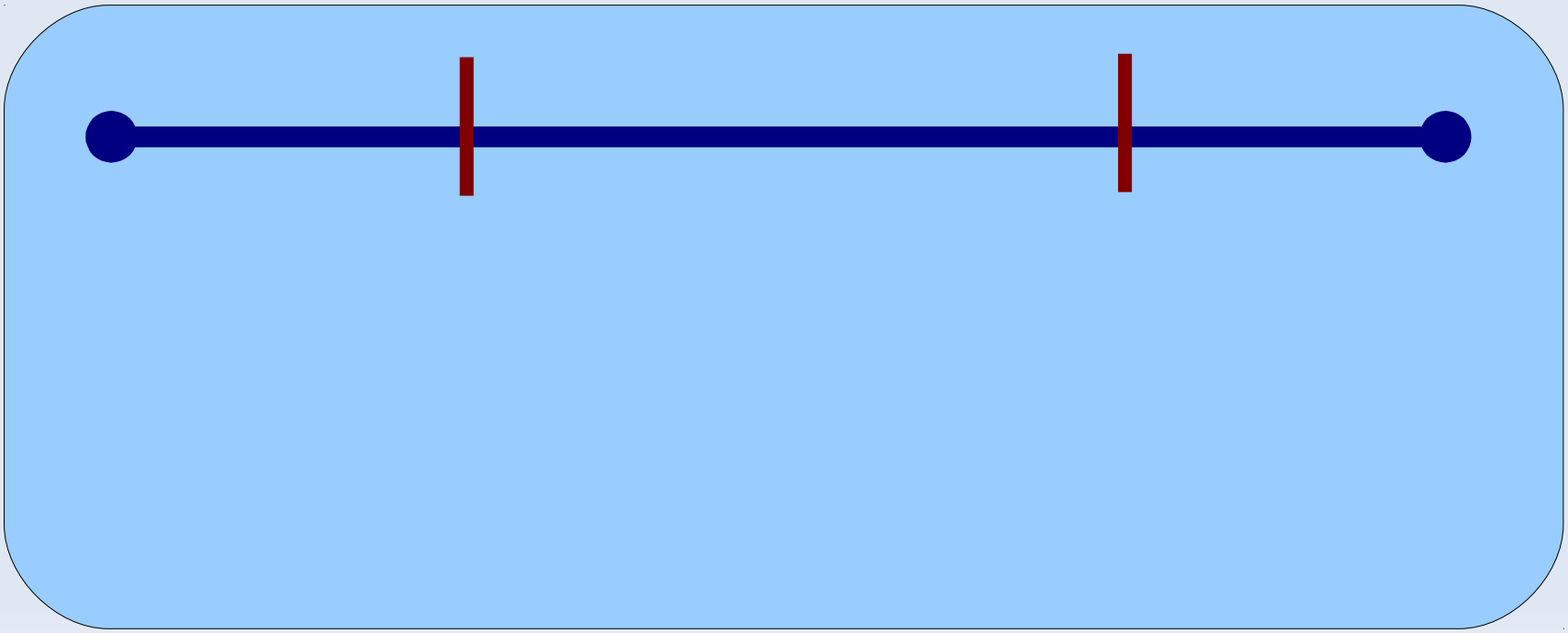
```
error/result × (value → error/result) → error/result
```

Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```

```
error/result × (value → error/result) → error/result
```

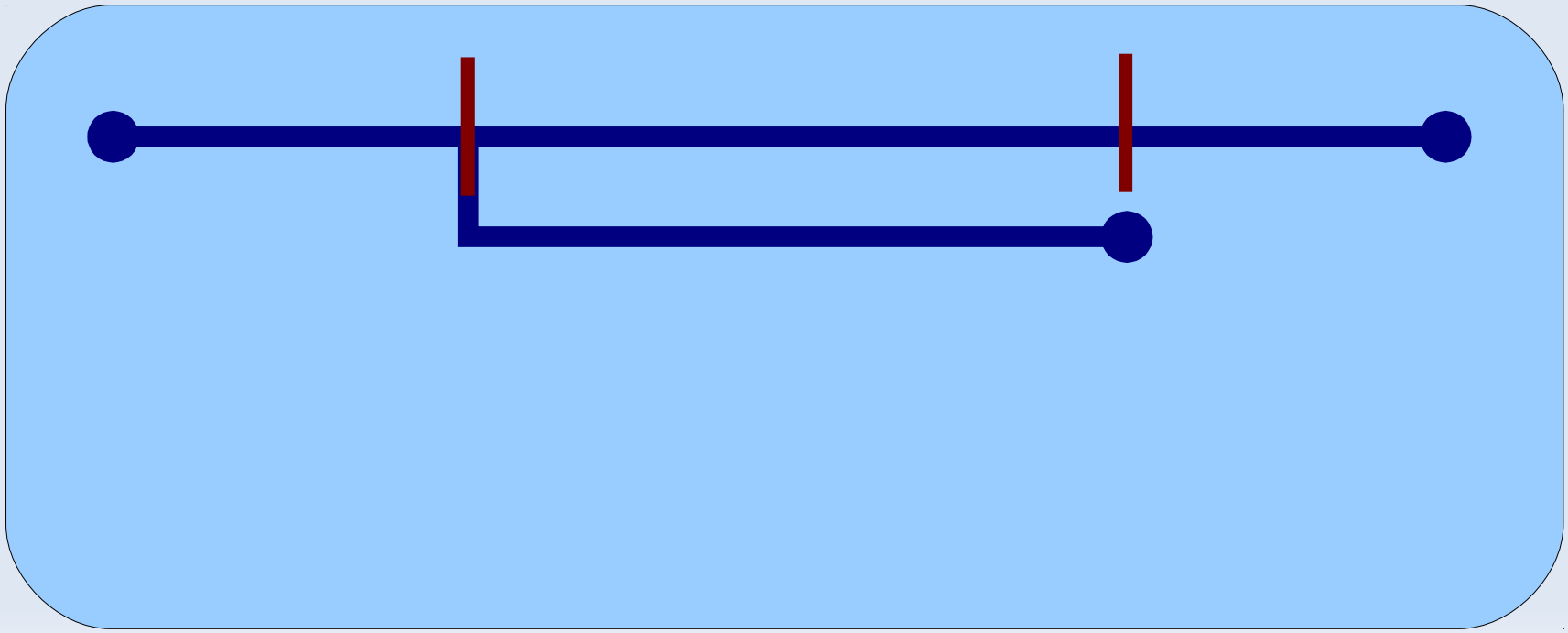


Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```

```
error/result × (value → error/result) → error/result
```

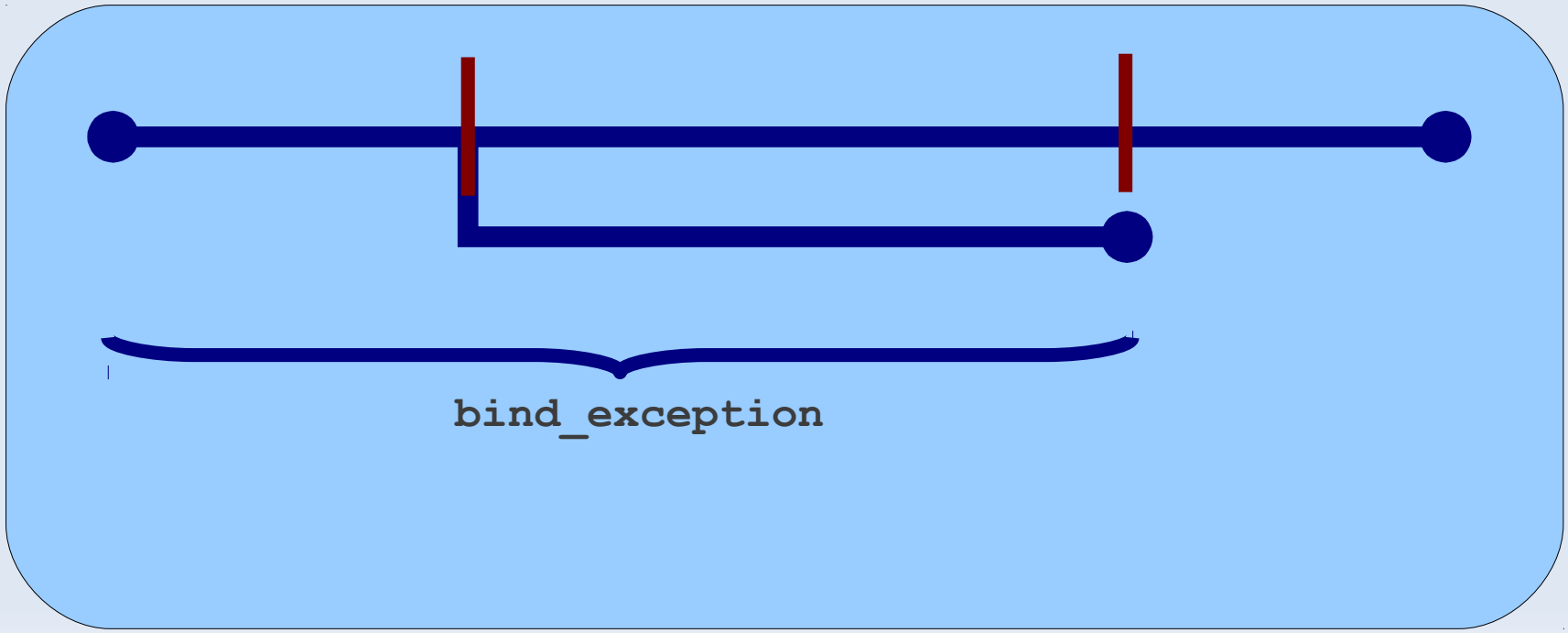


Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```

```
error/result × (value → error/result) → error/result
```

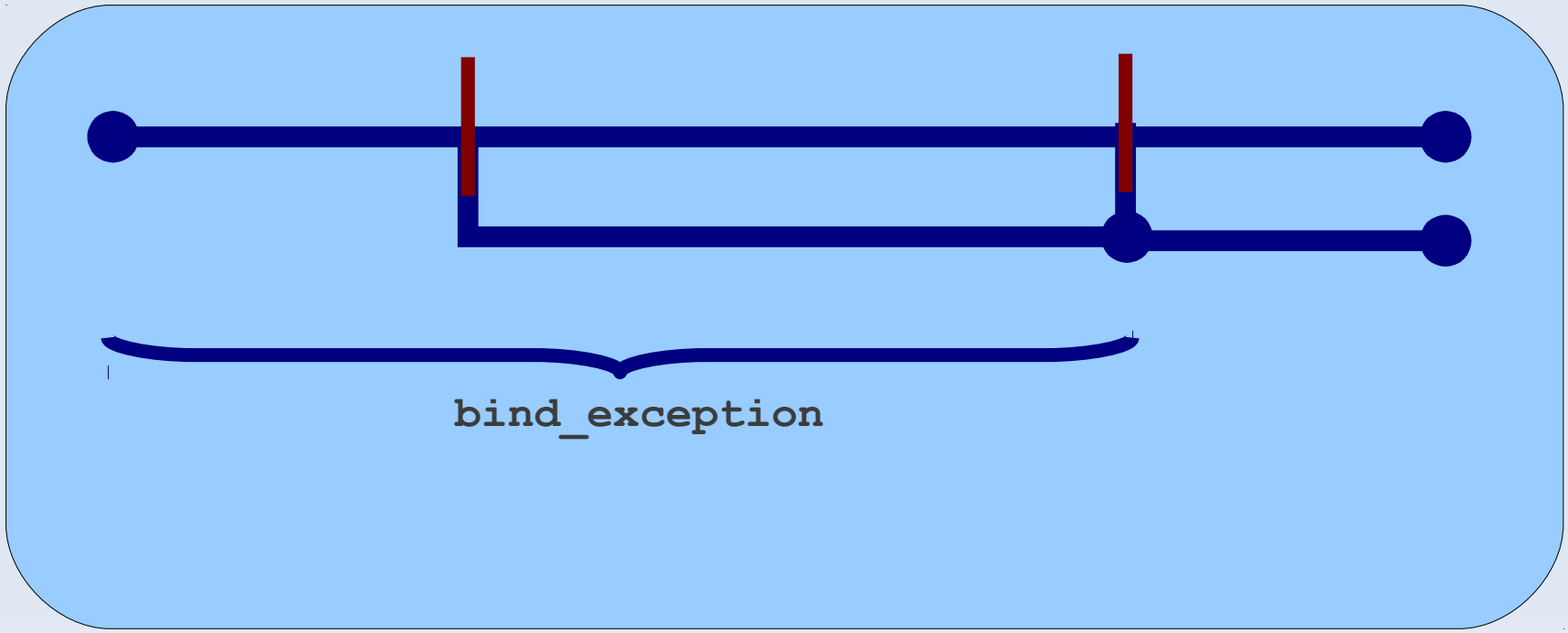


Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```

```
error/result × (value → error/result) → error/result
```

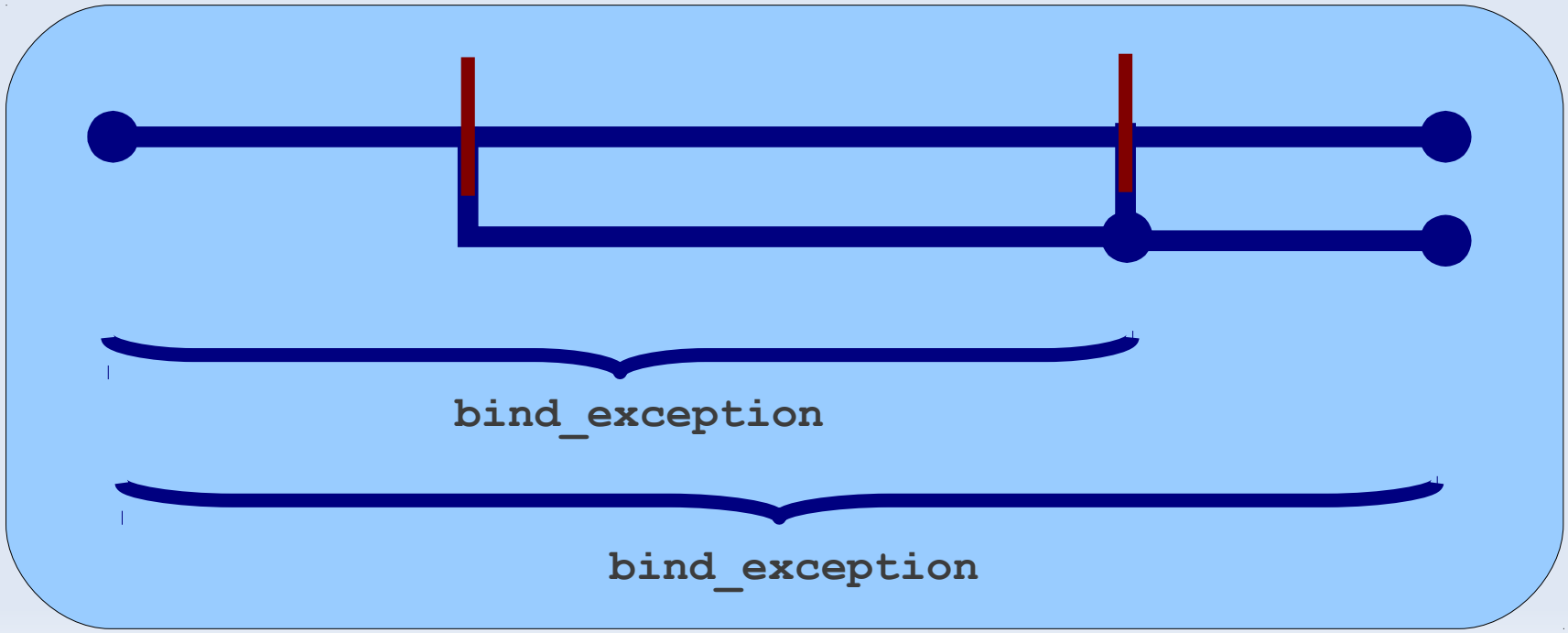


Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```

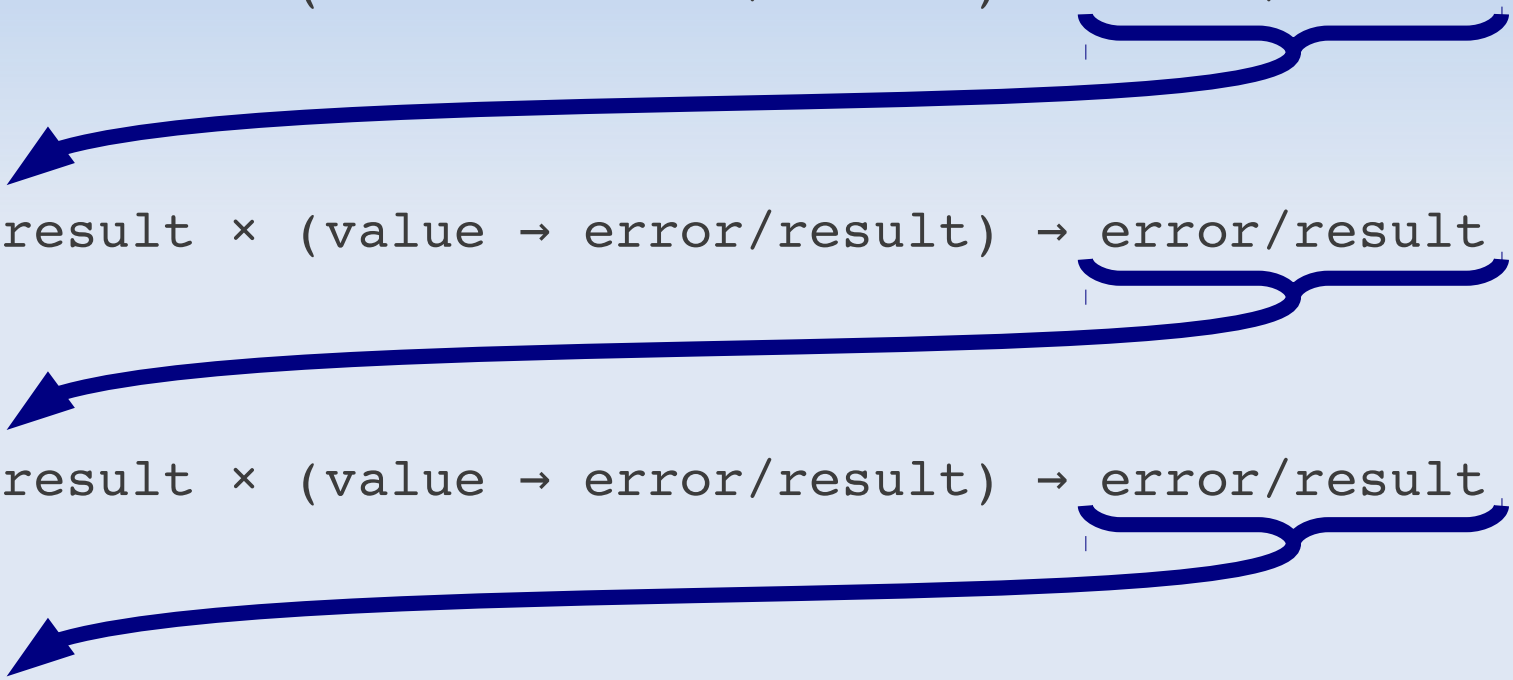
```
error/result × (value → error/result) → error/result
```



Error propagation

- Chaining functions

```
error/result × (value → error/result) → error/result
```




```
error/result × (value → error/result) → error/result
```

Error propagation

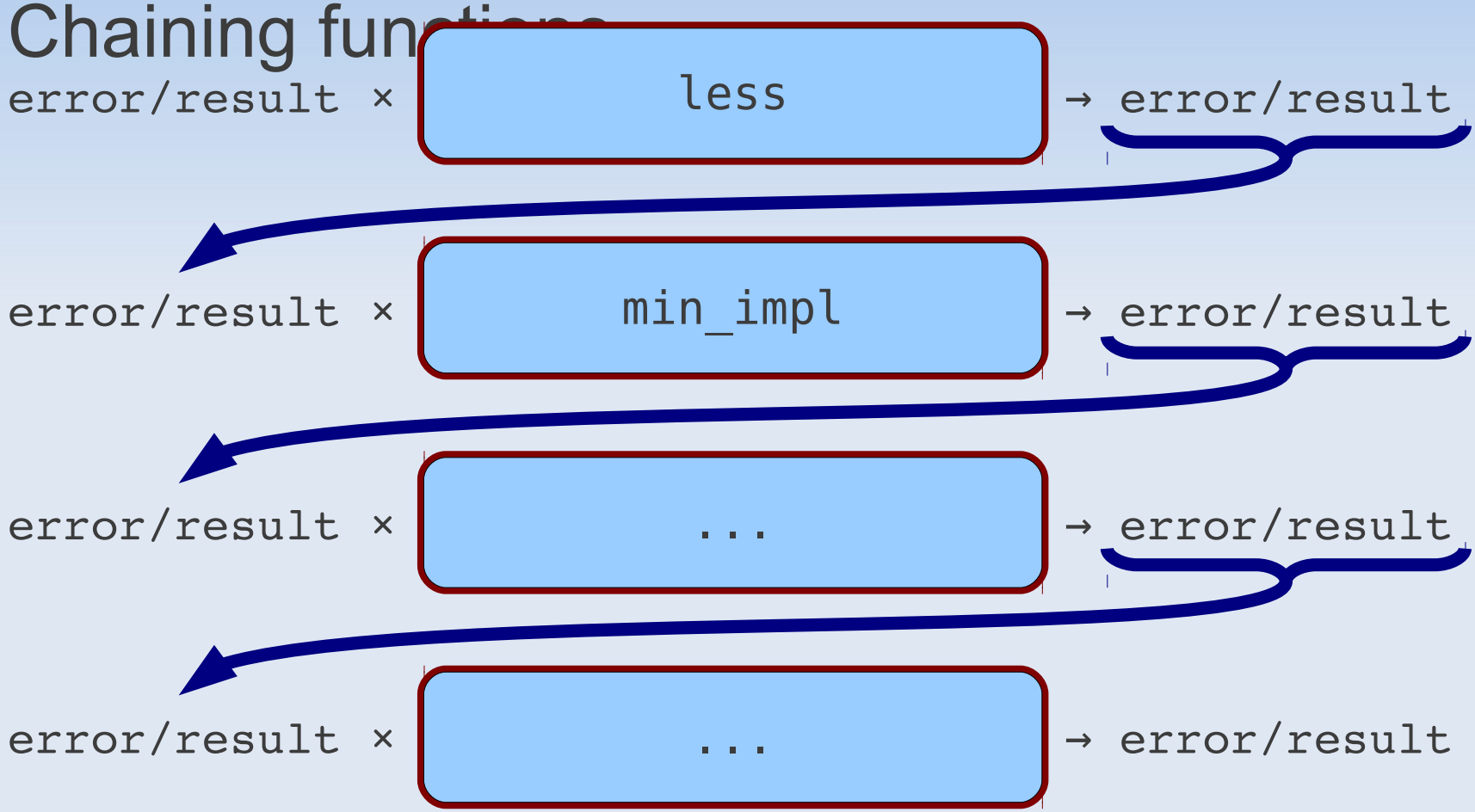
- Chaining functions

```
error/result × (value → error/result) → error/result
error/result × (value → error/result) → error/result
error/result × (value → error/result) → error/result
error/result × (value → error/result) → error/result
```

Business logic

Error propagation

- Chaining functions



Business logic

Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};
```

Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :









{};
```

Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :
  bind_exception<
    min<C, D>,




  >
{};
```

Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :
  bind_exception<
    min<C, D>,
    bind_exception<
      min<A, B>,

    >
  >
{};
```

Error propagation

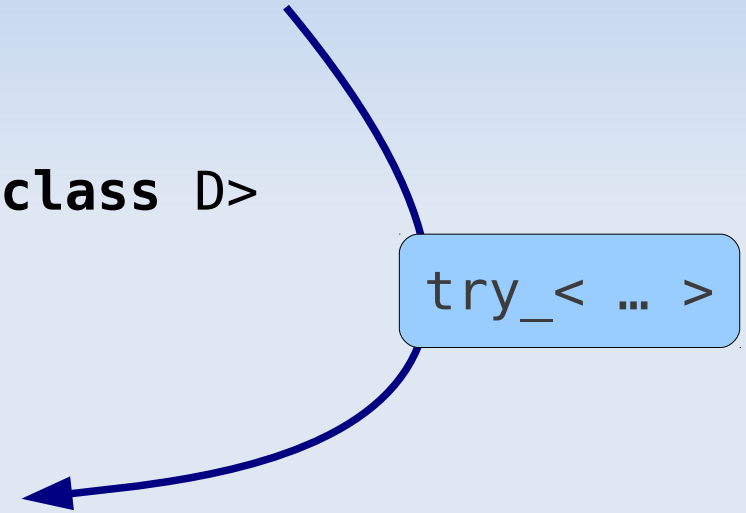
```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :
  bind_exception<
    min<C, D>,
    bind_exception<
      min<A, B>,
      curry<min>
    >
  >
{};
```

Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :
  bind_exception<
    min<C, D>,
    bind_exception<
      min<A, B>,
      curry<min>
    >
  >
{};
```



Error propagation

```
template <class A, class B, class C, class D>
struct min4 :
  min<min<A, B>, min<C, D> >
{};

template <class A, class B, class C, class D>
struct min4 :
  bind_exception<
    min<C, D>,
    bind_exception<
      min<A, B>,
      curry<min>
    >
  >
{};
```

```
try_< … >
```

```
template <class A, class B, class C, class D>
struct min4 :
  try_<min<min<A, B>, min<C, D> > >
{};
```

Testing

- Unit testing frameworks can catch exceptions
  - Compilation succeeds
  - They can be part of the test report
- In the ”Real World” nothing catches the exception
  - Compilation breaks
  - It is easy to build tools pretty-printing the exception

Not covered...

- Exceptions at compile-time
  - One can throw_<...>
  - One can catch_<...>
- Monads
  - Exception handling is implemented using monads
  - This is not their only use case in C++ template metaprogramming

Summary

- C++ template metaprogramming is useful
- Developing metaprograms is difficult
  - Lack of proper unit testing tools
  - Lack of handling errors at compile-time
  - ...

Summary

- C++ template metaprogramming is useful
- Developing metaprograms is difficult
  - ~~Lack of proper unit testing tools~~
  - ~~Lack of handling errors at compile-time~~
  - ...

Q & A

http://abel.web.elte.hu/mpllibs/