# Monadic error handling in C++ template metaprograms

Ábel Sinkovics

ELTE, Hungary

# Outline

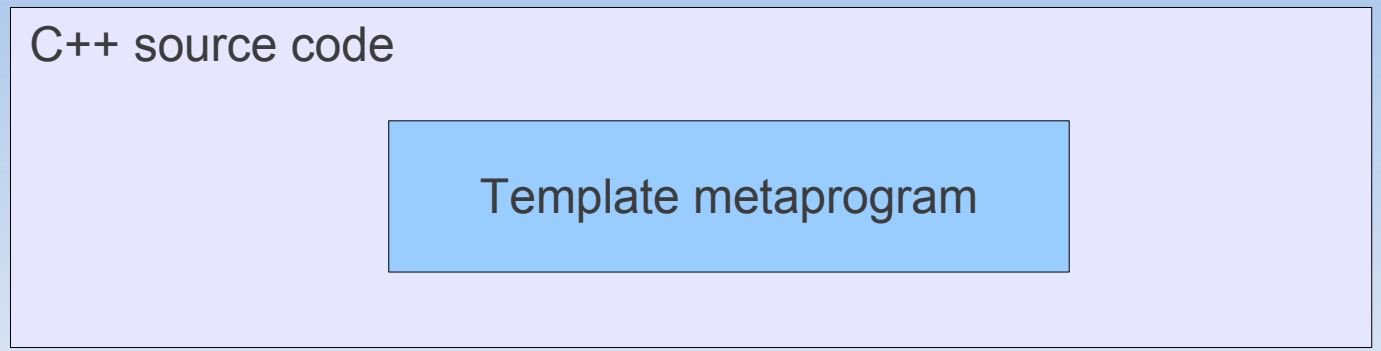
- Introduction to template metaprogramming
- Error handling using an example
- Monads in template metaprogramming
- Summary

# C++ template metaprogramming

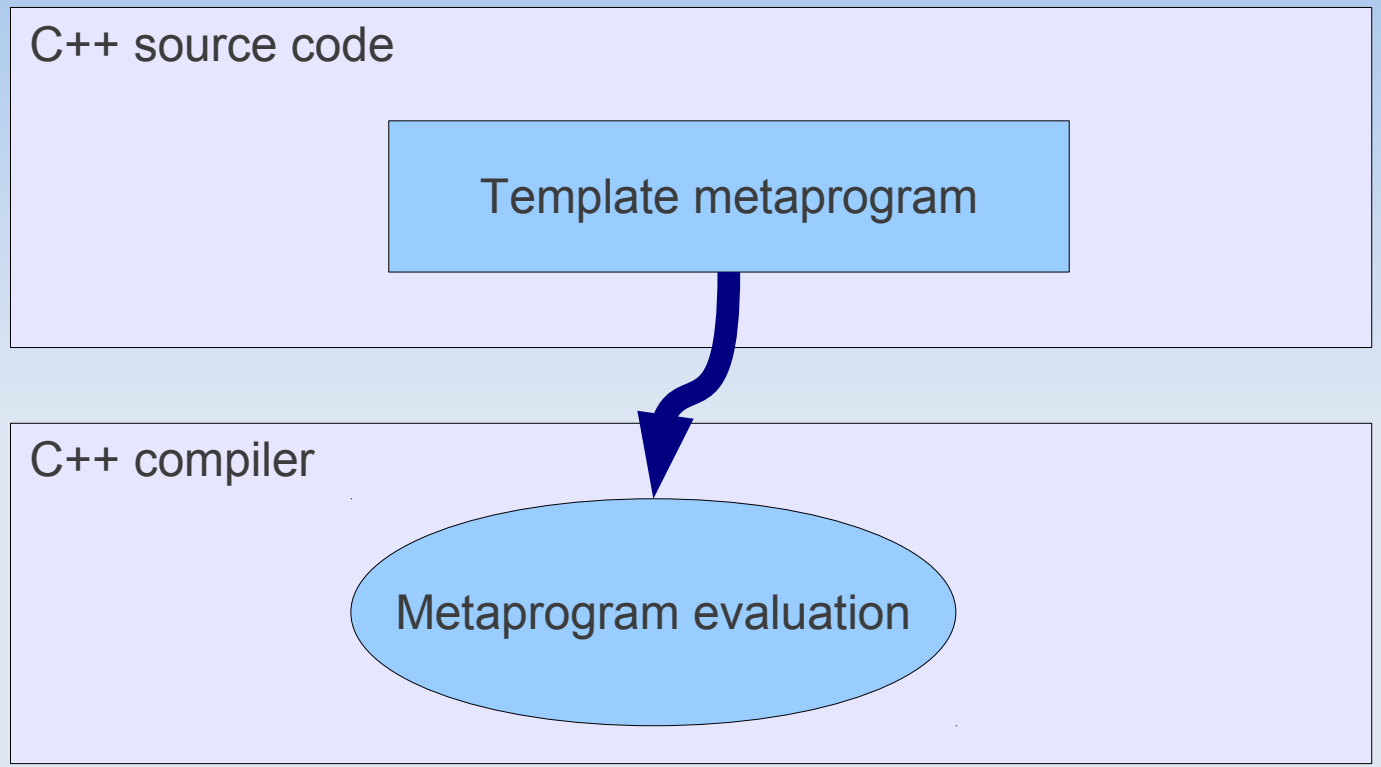
- Erwin Unruh, 1994
- Turing-complete

- Concept checking
- Expression templates
- DSL embedding

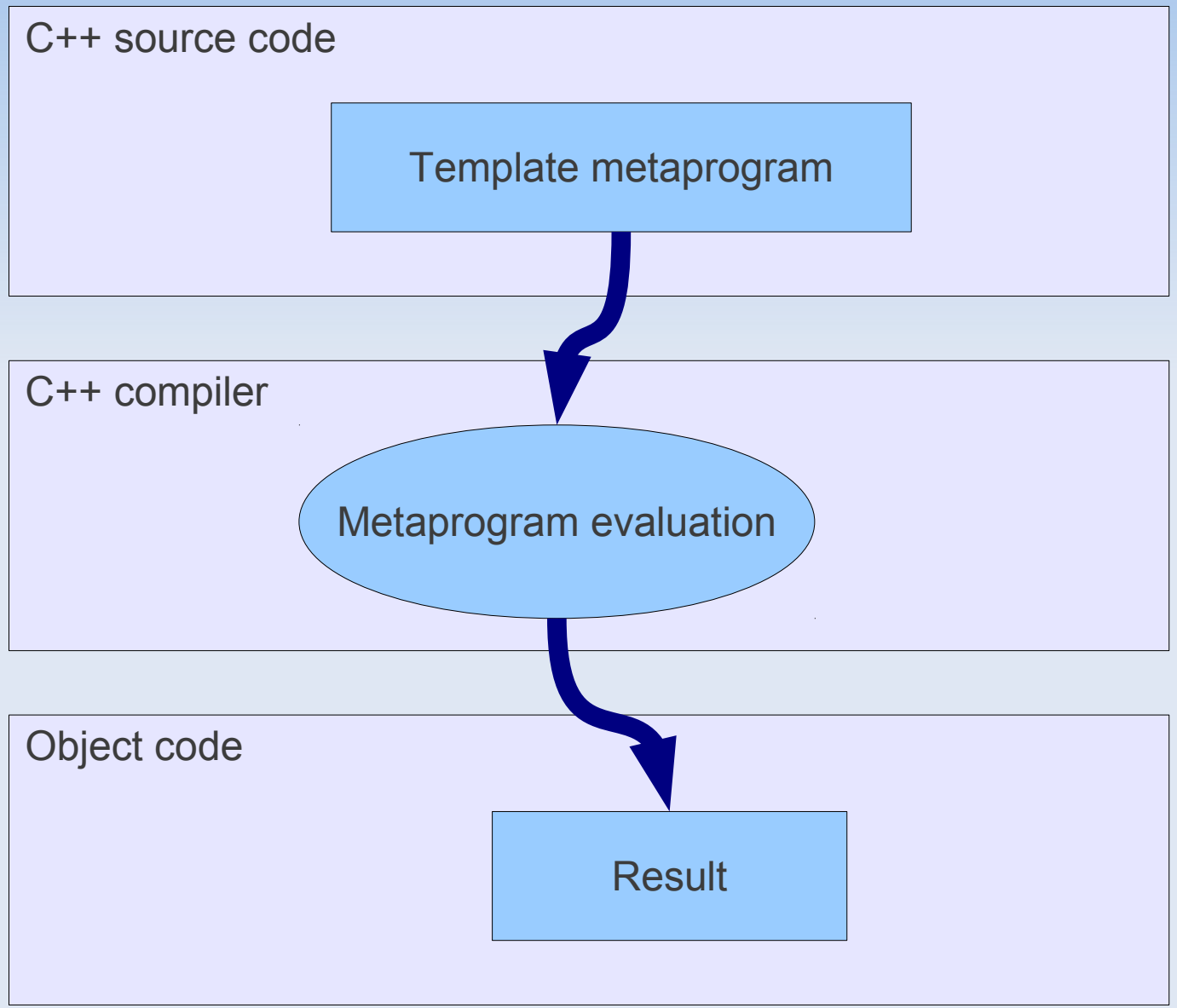
# C++ template metaprogramming



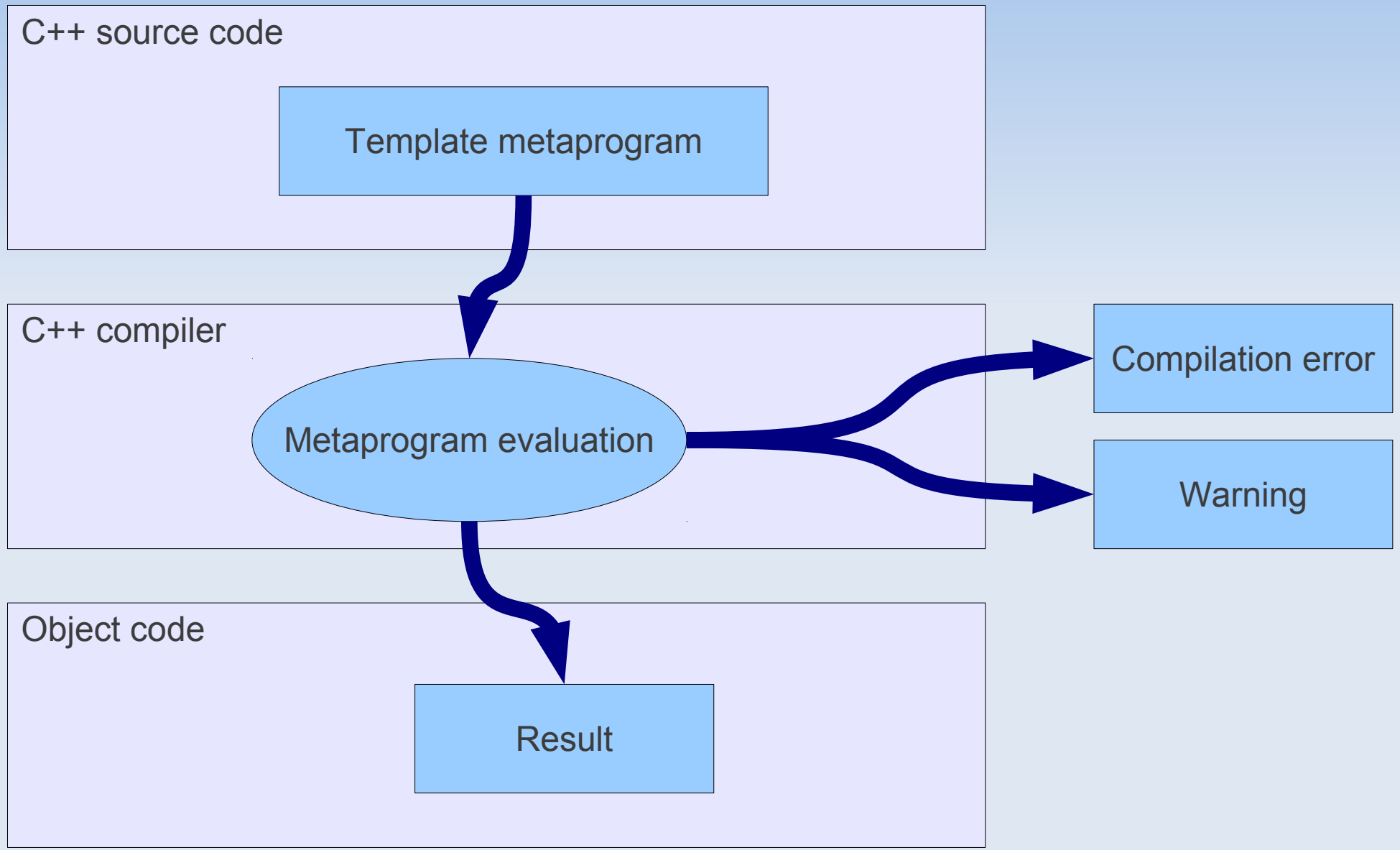
# C++ template metaprogramming



# C++ template metaprogramming



# C++ template metaprogramming



# C++ template metafunction

# C++ template metafunction

```
template <class T>
struct makeConst
{
  typedef const T type;
};
```

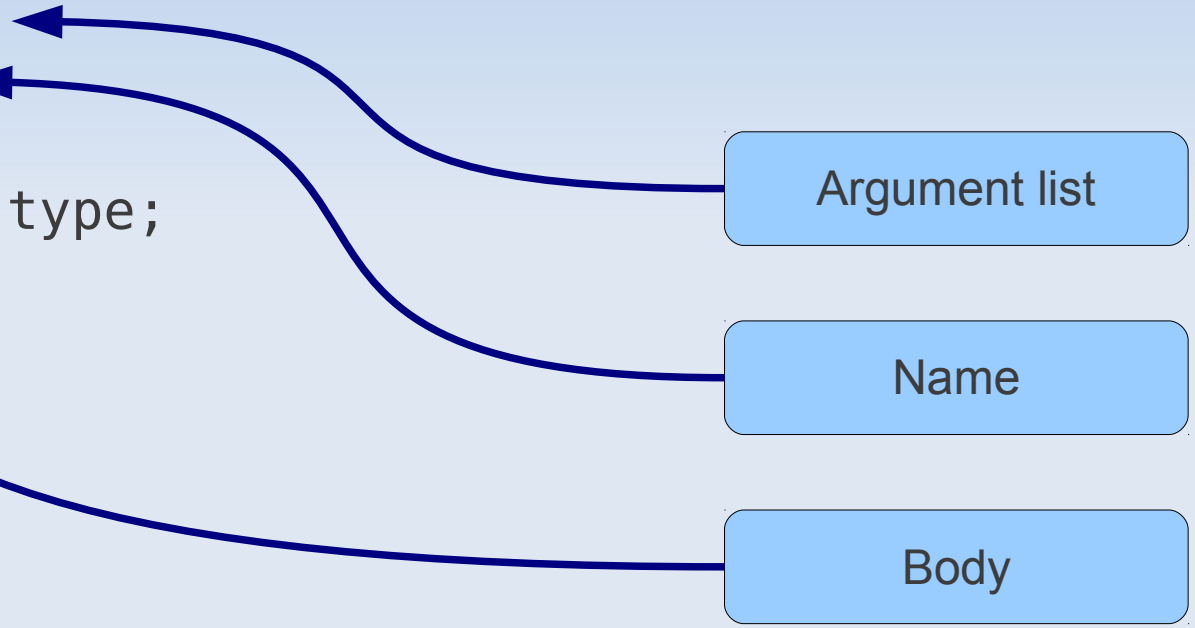
Argument list

Name

Body

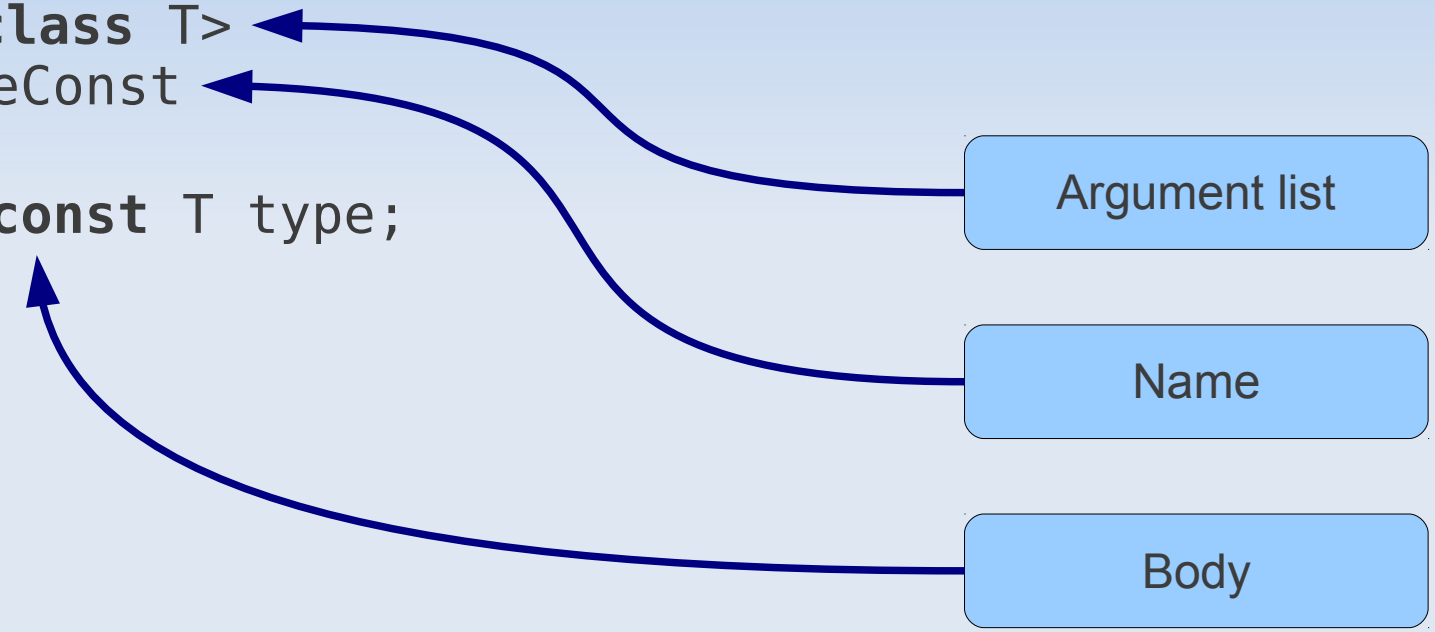
# C++ template metafunction

```
template <class T>
struct makeConst
{
  typedef const T type;
};
```



# C++ template metafunction

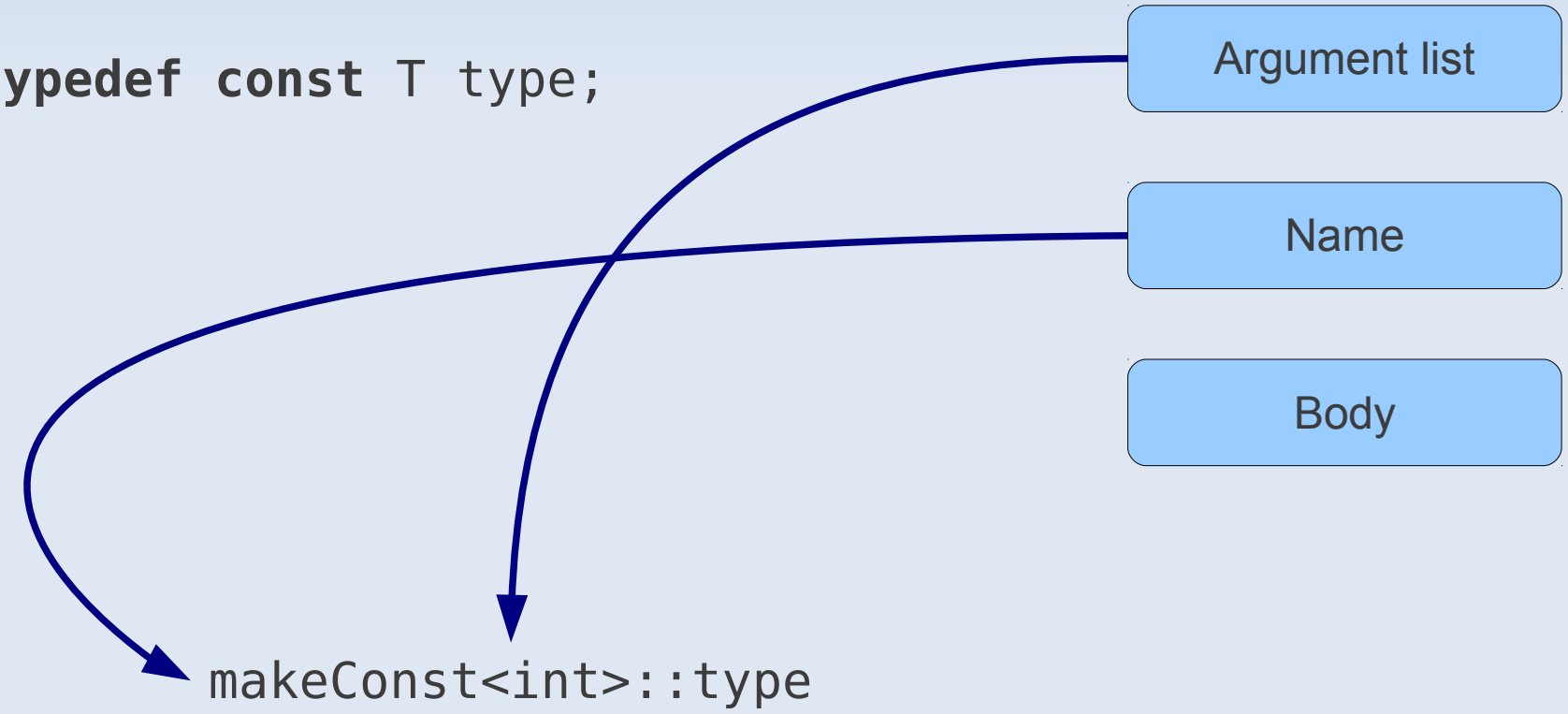
```
template <class T>
struct makeConst
{
  typedef const T type;
};
```




```
makeConst<int>::type
```

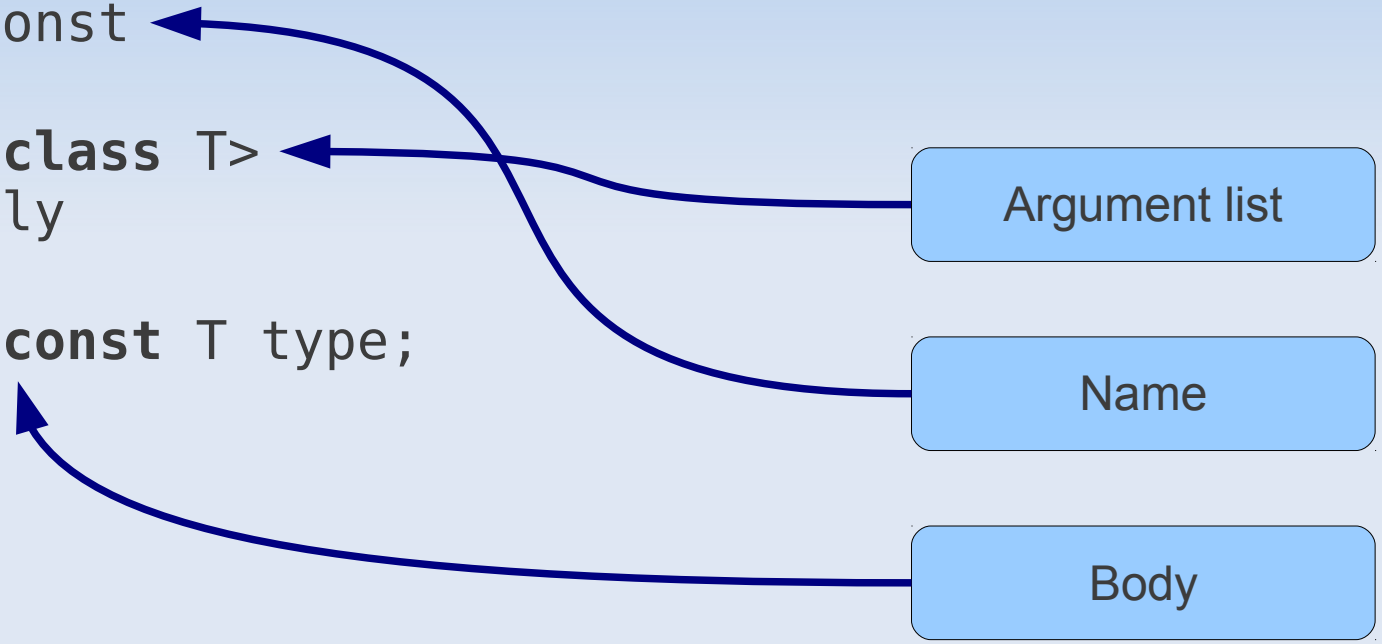
# C++ template metafunction

```
template <class T>
struct makeConst
{
  typedef const T type;
};
```



# Template metafunction class

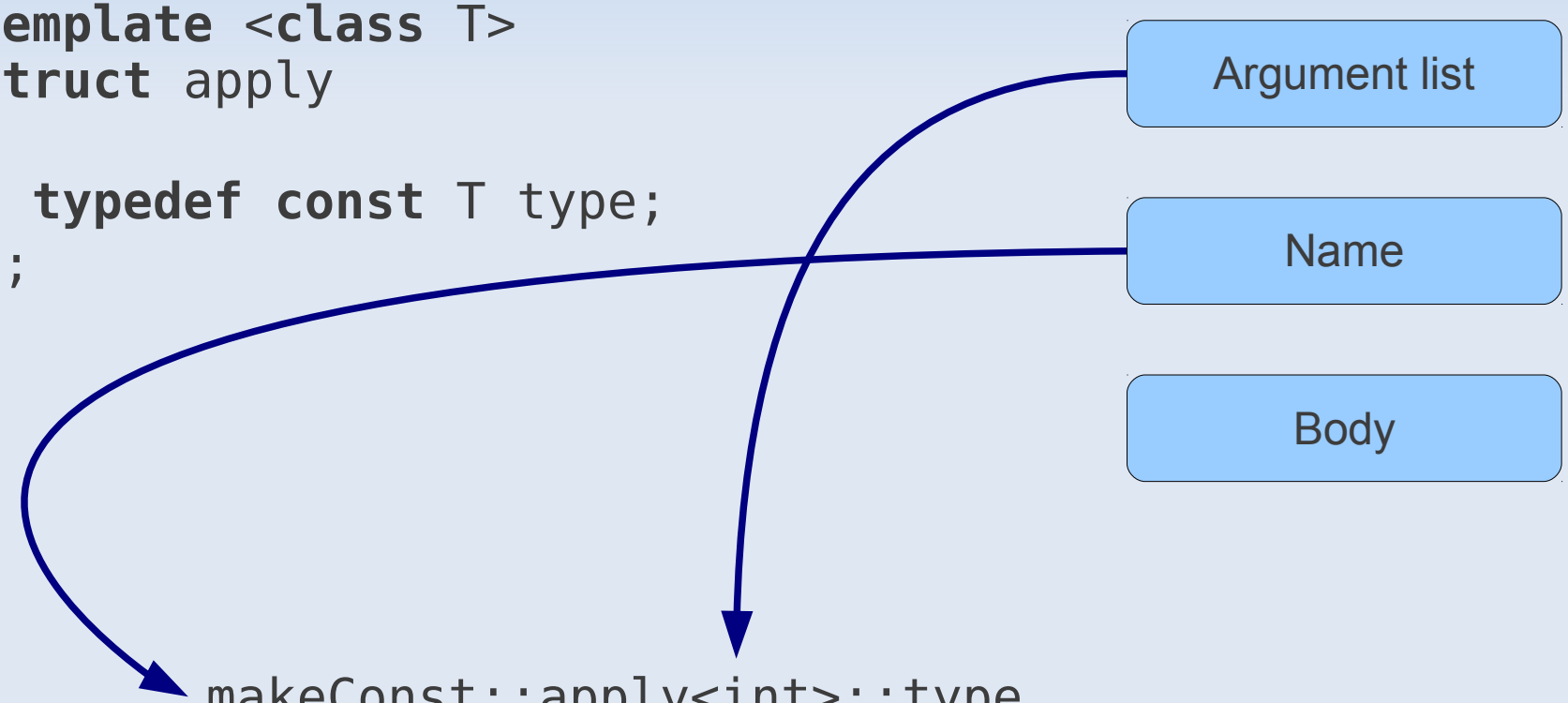
```
struct makeConst
{
  template <class T>
  struct apply
  {
    typedef const T type;
  };
};
```




```
makeConst::apply<int>::type
```

# Template metafunction class

```
struct makeConst
{
   template <class T>
   struct apply
   {
      typedef const T type;
   };
};
```



# Functional programming

- One can look at template metaprograms as pure functional programs
  - The execution of a metaprogram is the evaluation of a metafunction
  - No side-effects
  - Higher order functions
  - Pattern matching
  - Supports both eager and lazy evaluation

# 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 :
                        A         < B
{};

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

# Example

```
template <class A, class B>
struct less :
  boost::mpl::bool_<(A::value < B::value)>
{};

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

# Example

```
template <class A, class B>
struct less :
  boost::mpl::bool_<(A::value < B::value)>
{};

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 :
  boost::mpl::bool_<(A::value < B::value)>
{};

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

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

min<
                                    // 19 + 83i
                                    // 11 + 13i

>
```

# Example

```
template <class A, class B>
struct less :
  boost::mpl::bool_<(A::value < B::value)>
{};

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 :
  boost::mpl::bool_<(A::val
{};

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
>
```

# Returning errors

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

# Returning errors

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

template <class F>
struct debug_metafunction
{
  // tricks to display exception<Reason>
};
```

# Returning errors

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

template <class F>
struct debug_metafunction
{
  // tricks to display exception<Reason>
};
```

```
struct values_can_not_be_compared;

// less<A, B> returns either
//    bool_<...>
//    exception<values_can_not_be_compared>
```

# Example

```
template <class A, class B>
struct less :
  boost::mpl::bool_<(A::value < B::value)>
{};

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_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
>
```

# Example

```
struct condition_of_if_is_not_a_boolean;

// make if_ return exceptions



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

```
struct condition_of_if_is_not_a_boolean;

// make if_ re
```

```
exception<condition_of_if_is_not_a_boolean>
```

```
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::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<
              is_exception<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,

    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>,
    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>
  >
{};
```

# Monads in Haskell

```
class Monad m where
  (>>=)  :: m a -> (a -> m b) -> m b      -- bind
  return :: a -> m a
```

# Monads in metaprogramming

- A monad is:
  - A set of values

# Monads in metaprogramming

- A monad is:
  - A set of values
  - Identified by a class (tag)

# Monads in metaprogramming

- A monad is:
  - A set of values
  - Identified by a class (tag)
  - A `bind` metafunction

```
template <class Tag, class X, class F>
struct bind;
```

# Monads in metaprogramming

- A monad is:
  - A set of values
  - Identified by a class (tag)
  - A `bind` metafunction
    ```
    template <class Tag, class X, class F>
    struct bind;
    ```
  - A `return_` metafunction
    ```
    template <class Tag, class X>
    struct return_;
    ```

# The exception monad

- Set of values:
  Every value in template metaprogramming
- `Tag: struct exception_tag;`
- `bind:`
  ```
  template <class X, class F>
  struct bind_exception;
  ```
- `return_:`
  ```
  template <class X>
  struct identity;
  ```

# min revisited

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


  >
{};

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

# min revisited

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


  >
{};

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

# min revisited

```
template <class A, class B>
struct min :
  bind<exception_tag,
    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> {};
};
```

# min revisited

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

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

```
template <class A, class B>
struct min :
  DO<exception_tag>::apply<
    SET<x, less<A, B> >,
    boost::mpl::if_<x, A, B>
  >
{};
```

# Summary

- C++ template metaprogramming
- Current error handling approach
- New approach for error handling
- Monads in template metaprogramming

# Q & A

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