Boosting MPL with Haskell elements

Ábel Sinkovics

Mpllibs

- Template Metaprogramming libraries
- http://abel.web.elte.hu/mpllibs
    - Metaparse
    - Metamonad
    - Safe printf
    - XL Xpressive

Mpllibs

- Ábel Sinkovics
- Endre Sajó
- István Siroki
- Zoltán Porkoláb

Agenda

- Laziness
- Basic building blocks
- Let/Lamba/Case expressions
- Error handling
- Generalisations
- List comprehension

Fact

```
template <int N> struct fact
```

```
fact n =
```

Fact

```
template <int N> struct fact
{ enum { value = N * fact<N-1>::value }; };
```

```
fact n = n * fact (n – 1)
```

Fact

```
template <int N> struct fact
{ enum { value = N * fact<N-1>::value }; };

template <> struct fact<0> { enum { value = 1 }; };
```

```
fact n = n * fact (n – 1)
fact 0 = 1
```

Fact

reverse

partition

unique

```
tem          N> struct
              N * fac                ; };

templat         <0> { enum { value = 1 }; };
```

list

insert

map

min

if

erase

lambda

sort

count

transform

find

```
act                - 1)
fact
```

iterators

max

pair

vector

fold

string

Fact

reverse
partition
unique
list
insert
map
min
if
erase
lambda
sort
count
transform
find
iterators
max
pair
vector
fold
string
Boost.MPL

Boost.MPL

Boost.MPL

- Containers
- Iterators
- Algorithms
- Numeric data types
- Basic operations
- Lambda expressions

Boost.MPL

Boost.MPL

- Containers
- Iterators
- Algorithms
- Numeric data types
- Basic operations
- Lambda expressions

Template metaprogramming and the functional paradigm

- Values can not be changed
- Memoization
- Purity
- Higher-order metafunctions
- ...

Boost.MPL

Boost.MPL

- Containers
- Iterators
- Algorithms
- Numeric data types
- Basic operations
- Lambda expressions

- Currying
- Let expressions
- Algebraic data types
- Pattern matching
- Case expressions
- List comprehension

Boost.MPL

Boost.MPL	Metamonad
Containers	Currying
Iterators	Let expressions
Algorithms	Algebraic data types
Numeric data types	Pattern matching
Basic operations	Case expressions
Lambda expressions	List comprehension

```
template <class A, class B>
struct foo : bar<A, B, A> {};
```

Template metafunction

```cpp
// This is a template metafunction
template <class A, class B>
struct foo : bar<A, B, A> {};
```

Template metafunction

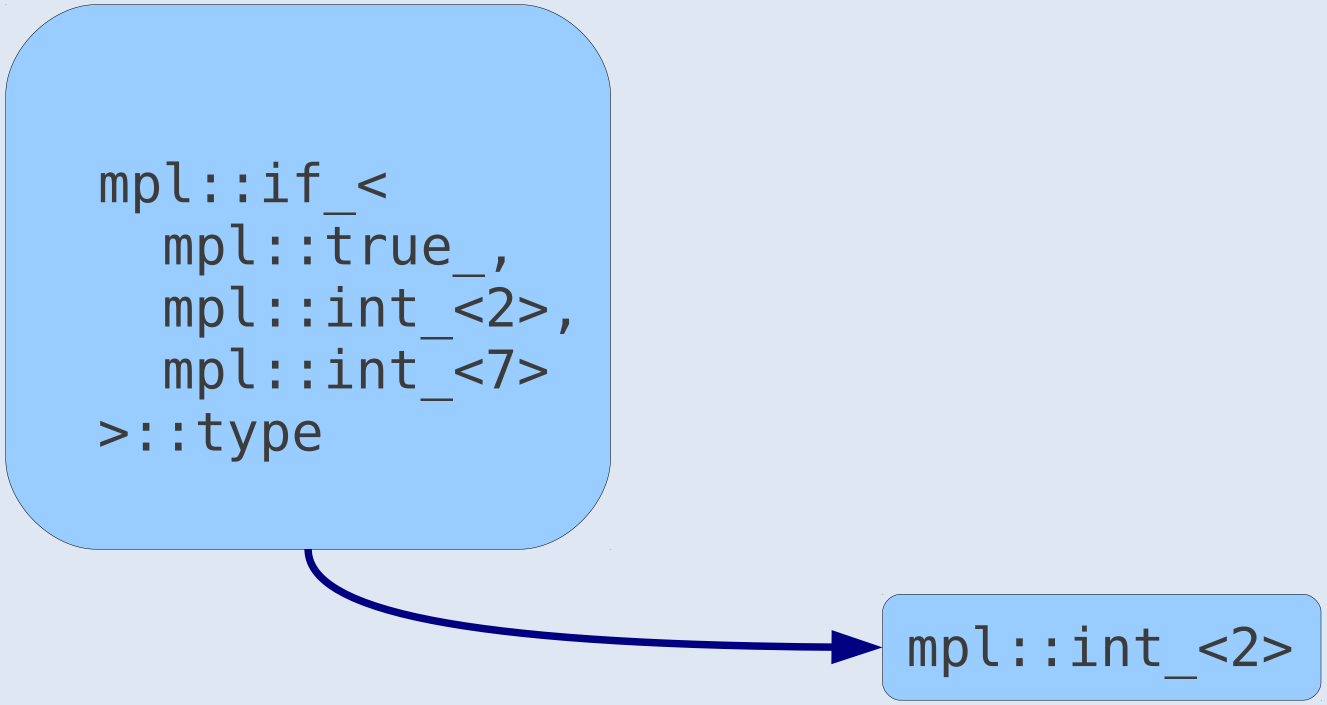
```
// This is a template metafunction
template <class A, class B>
struct foo : bar<A, B, A> {};
```

```
MPLLIBS_METAFUNCTION(foo, (A)(B))
((
  bar<A, B, A>
));
```

Times

```
mpl::if_<
  mpl::true_,
  mpl::int_<2>,
  mpl::int_<7>
>::type
```

Times



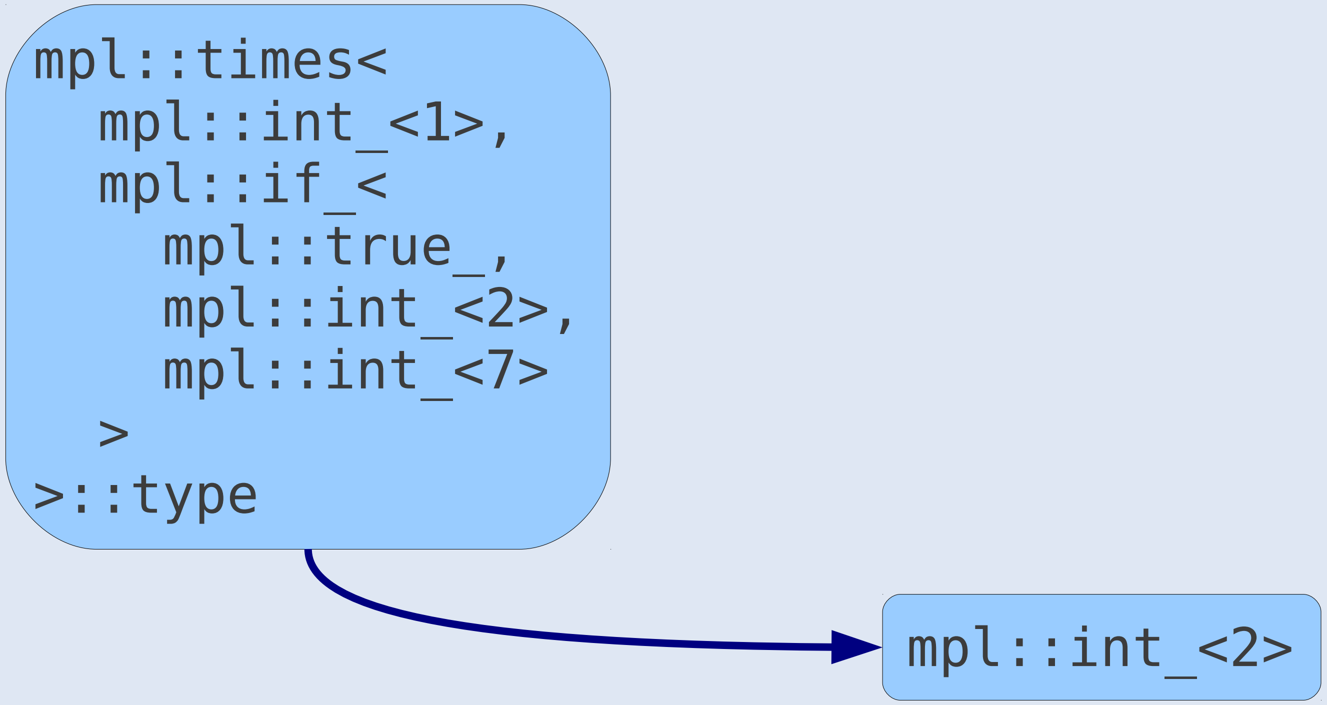
```
mpl::if_<
  mpl::true_,
  mpl::int_<2>,
  mpl::int_<7>
>::type
```

```
mpl::int_<2>
```

Times

```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >
>::type
```

Times



```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >
>::type
```

```
mpl::int_<2>
```

Times

```
In file included from /usr/include/boost/mpl/aux_/include_preprocessed.hpp:37:0,
                 from /usr/include/boost/mpl/aux_/arithmetic_op.hpp:34,
                 from /usr/include/boost/mpl/times.hpp:19,
                 from main.cpp:1:
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp: In instantiation of 'str
uct boost::mpl::times_tag<boost::mpl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_
::int_<7> > >':
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp:109:8:   required from 's
truct boost::mpl::times<mpl_::int_<1>, boost::mpl::if_<mpl_::bool_<true>, mpl_::
int_<2>, mpl_::int_<7> > >'
main.cpp:13:2:   required from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp:60:29: error: no type nam
ed 'tag' in 'struct boost::mpl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_::int_
<7> >'
main.cpp:6:1: error: 'type' in 'struct boost::mpl::times<mpl_::int_<1>, boost::m
pl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_::int_<7> > >' does not name a type
```

```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >
>::type
```

```
mpl::int_<2>
```

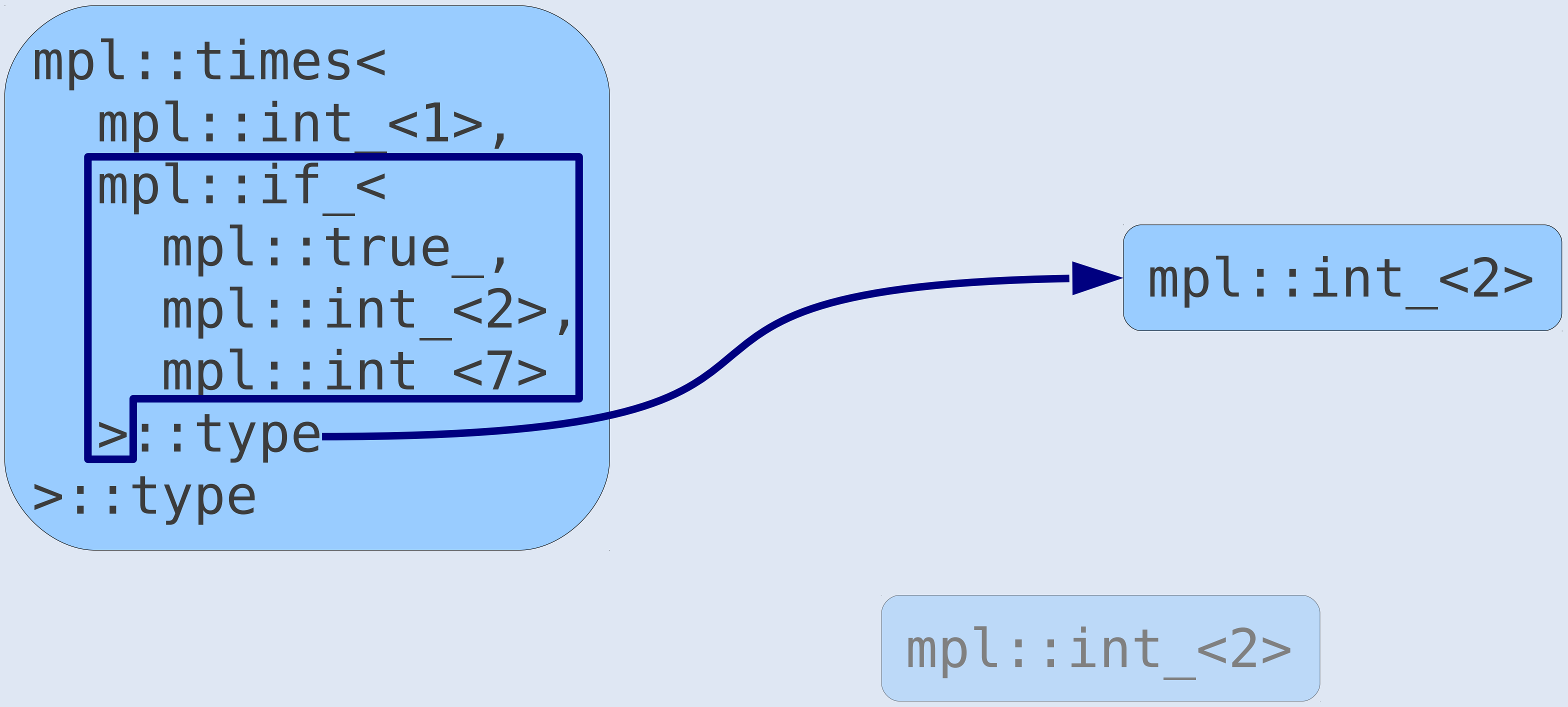
Times

```
In file included from /usr/include/boost/mpl/aux_/include_preprocessed.hpp:37:0,
                 from /usr/include/boost/mpl/aux_/arithmetic_op.hpp:34,
                 from /usr/include/boost/mpl/times.hpp:19,
                 from main.cpp:1:
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp: In instantiation of 'str
uct boost::mpl::times_tag<boost::mpl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_
::int_<7> > >':
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp:109:8:   required from 's
truct boost::mpl::times<mpl_::int_<1>, boost::mpl::if_<mpl_::bool_<true>, mpl_::
int_<2>, mpl_::int_<7> > >'
main.cpp:13:2:   required from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/times.hpp:60:29: error: no type nam
ed 'tag' in 'struct boost::mpl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_::int_
<7> >'
main.cpp:6:1: error: 'type' in 'struct boost::mpl::times<mpl_::int_<1>, boost::m
pl::if_<mpl_::bool_<true>, mpl_::int_<2>, mpl_::int_<7> > >' does not name a type
```

```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >
>::type
```

```
mpl::int_<2>
```

Times

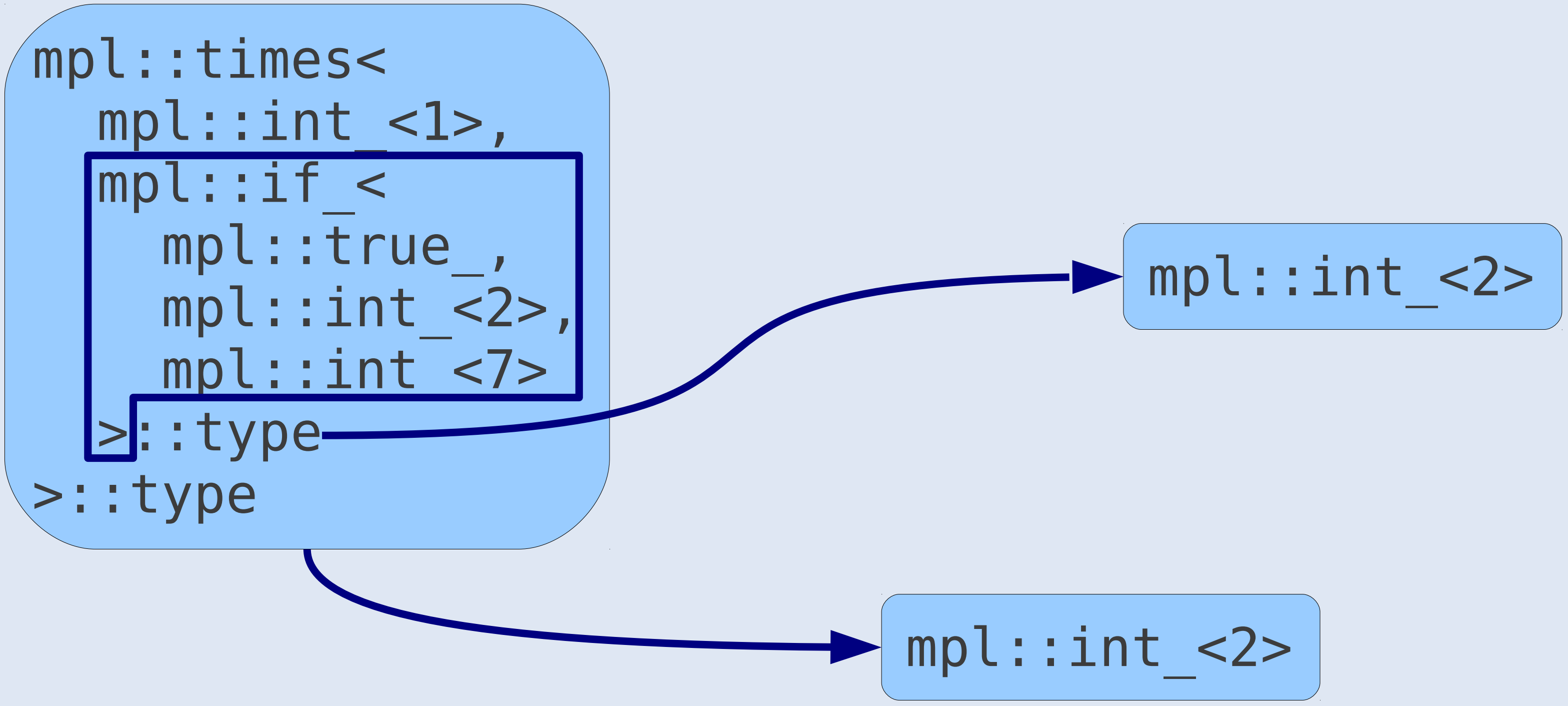


```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

```
mpl::int_<2>
```

```
mpl::int_<2>
```

Times

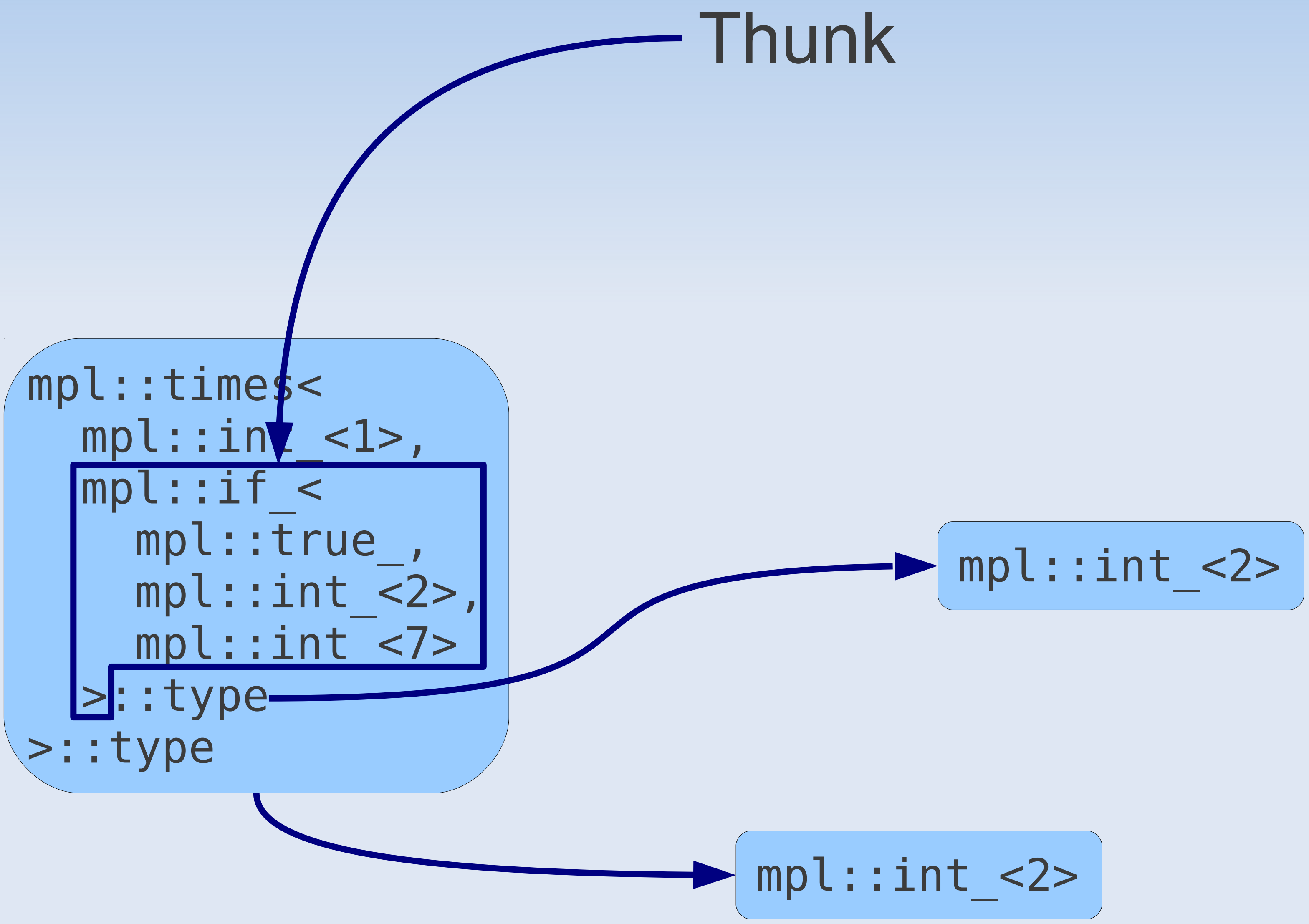


```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

```
mpl::int_<2>
```

```
mpl::int_<2>
```

Times



```
mpl::times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

```
mpl::int_<2>
```

```
mpl::int_<2>
```

Times

```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
                                A                          B
));
```

```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

Times

```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
             typename A::type  typename B::type
));
```

```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

Times

```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
  mpl::times<typename A::type, typename B::type>
));
```

```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

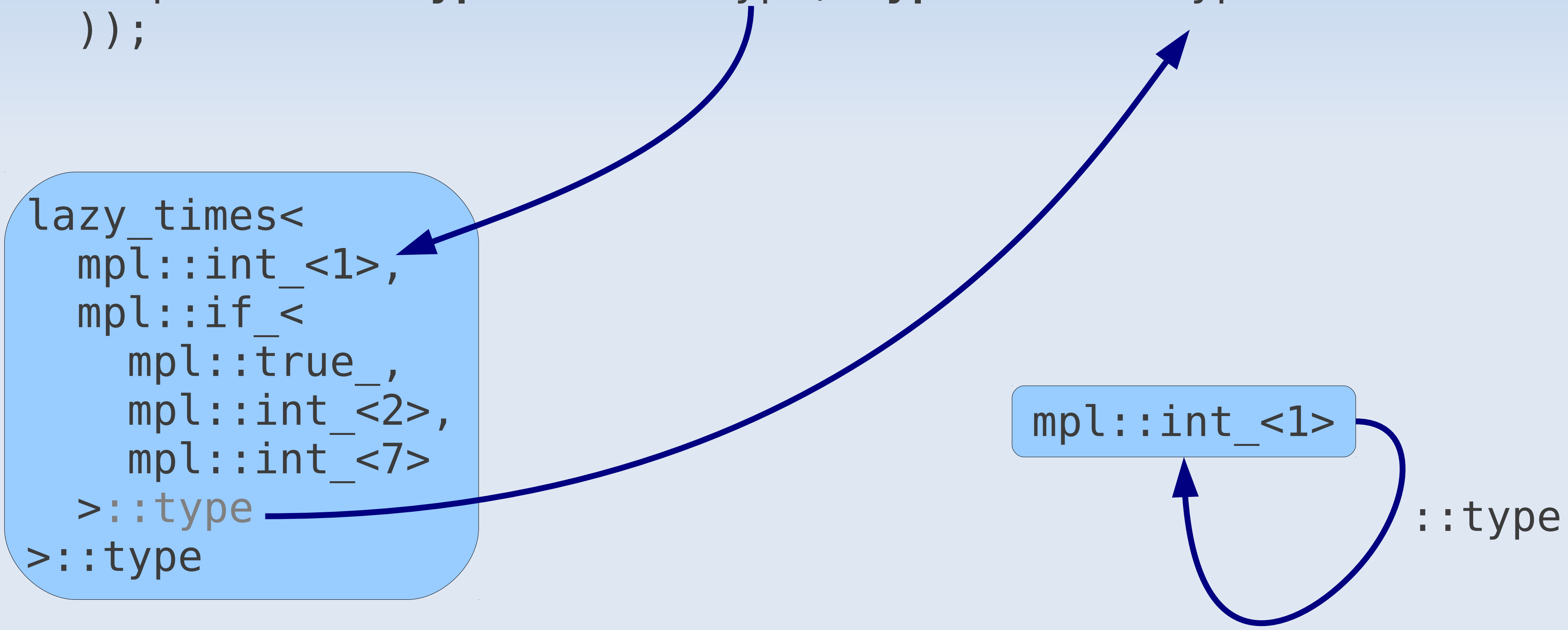
Times

```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
  mpl::times<typename A::type, typename B::type>
));
```

```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

Times

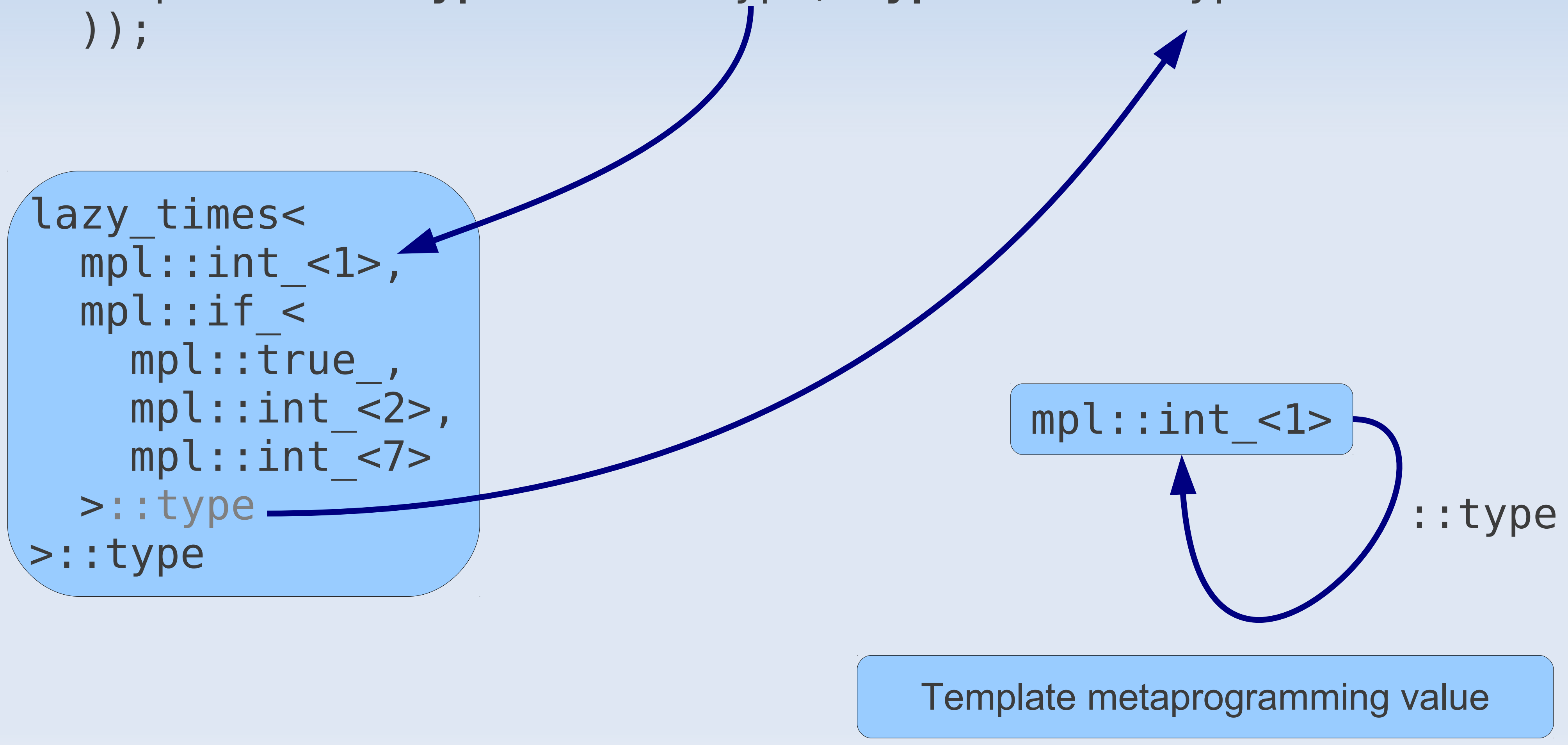
```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
  mpl::times<typename A::type, typename B::type>
));
```



```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

Times

```
MPLLIBS_METAFUNCTION(lazy_times, (A)(B))
((
  mpl::times<typename A::type, typename B::type>
));
```



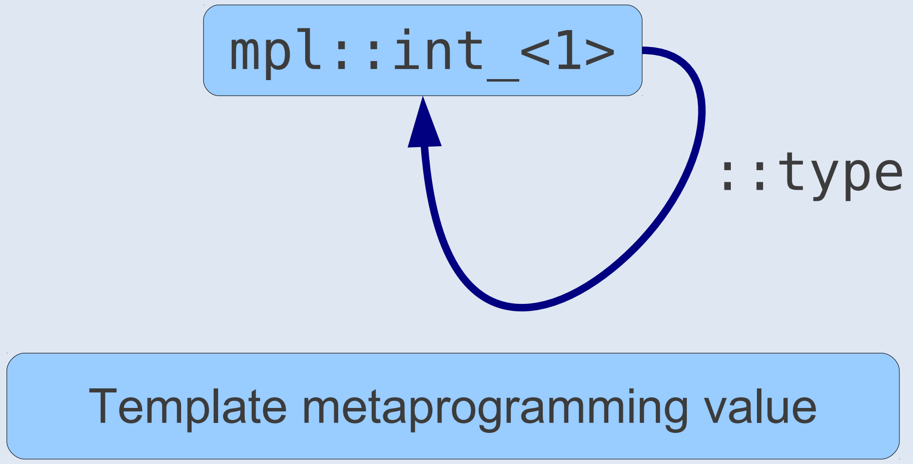
```
lazy_times<
  mpl::int_<1>,
  mpl::if_<
    mpl::true_,
    mpl::int_<2>,
    mpl::int_<7>
  >::type
>::type
```

```
mpl::int_<1>
```

Template metaprogramming value

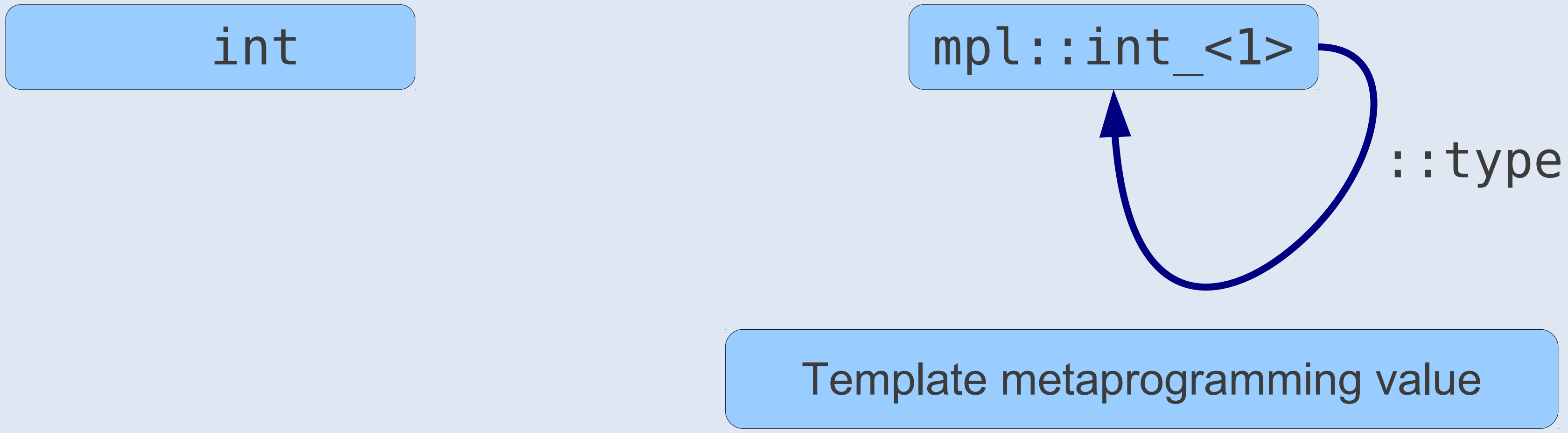
Times

- Assumption: every class used as a value in a template metaprogram is a template metaprogramming value



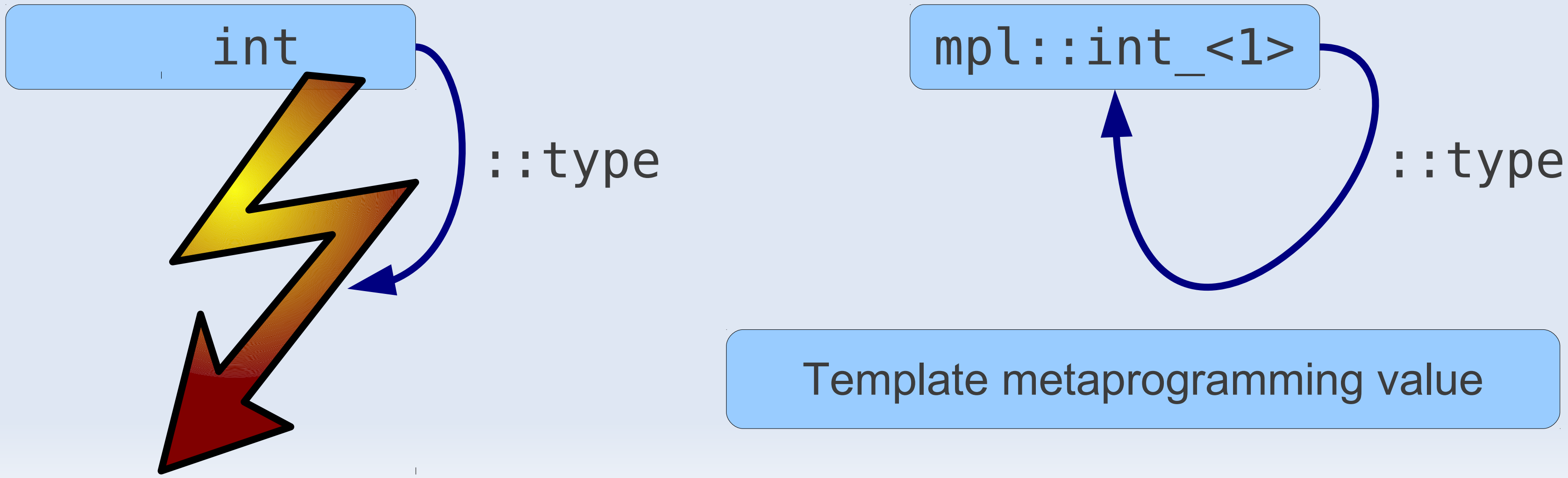
Times

- Assumption: every class used as a value in a template metaprogram is a template metaprogramming value



Times

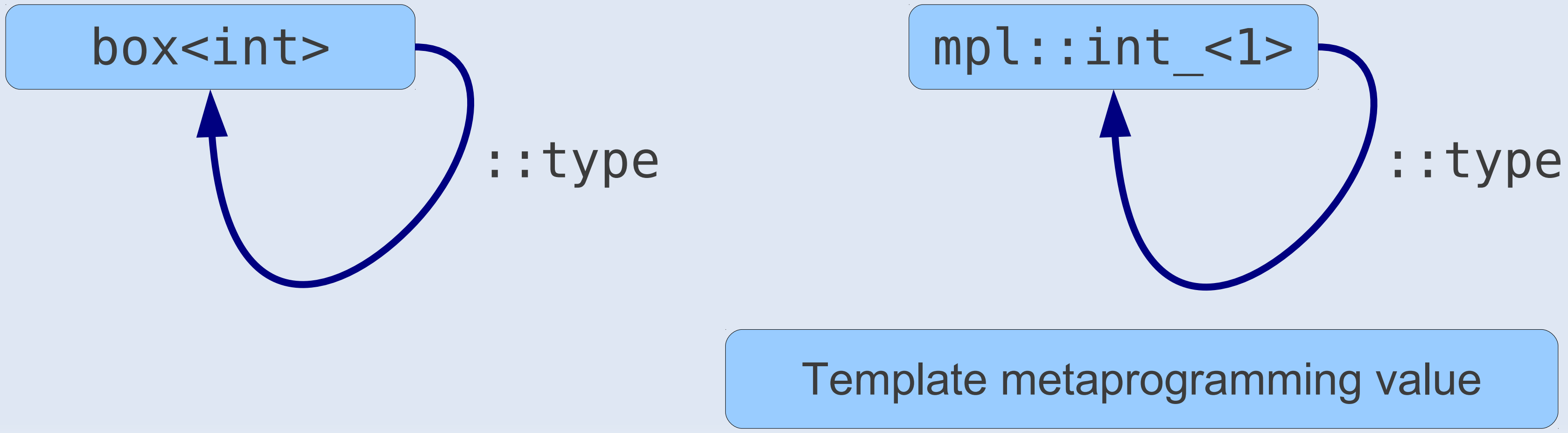
- Assumption: every class used as a value in a template metaprogram is a template metaprogramming value



Times

- Assumption: every class used as a value in a template metaprogram is a template metaprogramming value

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



Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N - 1);
}
```

```
));
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<




      ,
                 ,









  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N - 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::equal_to<
      mpl::int_<0>,
      N
    >,
                ,






  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N - 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::equal_to<
      mpl::int_<0>,
      N
    >,
    mpl::int_<1>,






  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N – 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::equal_to<
      mpl::int_<0>,
      N
    >,
    mpl::int_<1>,
    mpl::times<





       ,
      N
    >
  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N - 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::equal_to<
      mpl::int_<0>,
      N
    >,
    mpl::int_<1>,
    mpl::times<
      fact<




      >,
      N
    >
  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N - 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::equal_to<
      mpl::int_<0>,
      N
    >,
    mpl::int_<1>,
    mpl::times<
      fact<
        mpl::minus<
          N,
          mpl::int_<1>
        >
      >,
      N
    >
  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N – 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    typename mpl::equal_to<
      mpl::int_<0>,
      N
    >::type,
    mpl::int_<1>,
    mpl::times<
      typename fact<
        typename mpl::minus<
          N,
          mpl::int_<1>
        >::type
      >::type,
      N
    >
  >
));
```

```
int fact(int N)
{
  return 0 == N ? 1 : N * fact(N – 1);
}
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    typename mpl::equal_to<
      mpl::int_<0>,
      N
    >::type,
    mpl::int_<1>,
    mpl::times<
      typename fact<
        typename mpl::minus<
          N,
          mpl::int_<1>
        >::type
      >::type,
      N
    >
  >
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    typename mpl::equal_to<
      mpl::int_<0>,
      mpl::int_<0>
    >::type,
    mpl::int_<1>,
    mpl::times<
      typename fact<
        typename mpl::minus<
          mpl::int_<0>,
          mpl::int_<1>
        >::type
      >::type,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    typename mpl::equal_to<
      mpl::int_<0>,
      mpl::int_<0>
    >::type,
    mpl::int_<1>,
    mpl::times<
      typename fact<
        typename mpl::minus<
          mpl::int_<0>,
          mpl::int_<1>
        >::type
      >::type,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::true_,




    mpl::int_<1>,
    mpl::times<
      typename fact<
        typename mpl::minus<
          mpl::int_<0>,
          mpl::int_<1>
        >::type
      >::type,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::true_,




    mpl::int_<1>,
    mpl::times<
      typename fact<
        mpl::int_<-1>




      >::type,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::true_,




    mpl::int_<1>,
    mpl::times<
      typename fact<
        mpl::int_<-1>



      >::type,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  lazy_eval_if<
    lazy_equal_to<
      mpl::int_<0>,
      N
    >,
    mpl::int_<1>,
    lazy_times<
      fact<
        lazy_minus<
          N,
          mpl::int_<1>
        >
      >,
      N
    >
  >
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  lazy_eval_if<
    lazy_equal_to<
      mpl::int_<0>,
      mpl::int_<0>
    >,
    mpl::int_<1>,
    lazy_times<
      fact<
        lazy_minus<
          mpl::int_<0>,
          mpl::int_<1>
        >
      >,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  lazy_eval_if<
    lazy_equal_to<
      mpl::int_<0>,
      mpl::int_<0>
    >,
    mpl::int_<1>,
    lazy_times<
      fact<
        lazy_mi
          mpl:
          mpl:
        >
      >,
      mpl::int_<0>
    >
  >::type
));
```

```
MPLLIBS_METAFUNCTION(lazy_eval_if, (C)(T)(F))
((
  mpl::eval_if<typename C::type, T, F>
));
```

```
fact<mpl::int_<0>>::type
```

Fact

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  mpl::eval_if<
    mpl::true_,




    mpl::int_<1>,
    lazy_times<
      fact<
        lazy_minus<
          mpl::int_<0>,
          mpl::int_<1>
        >
      >,
      mpl::int_<0>
    >
  >::type
));
```

```
fact<mpl::int_<0>>::type
```

Fact

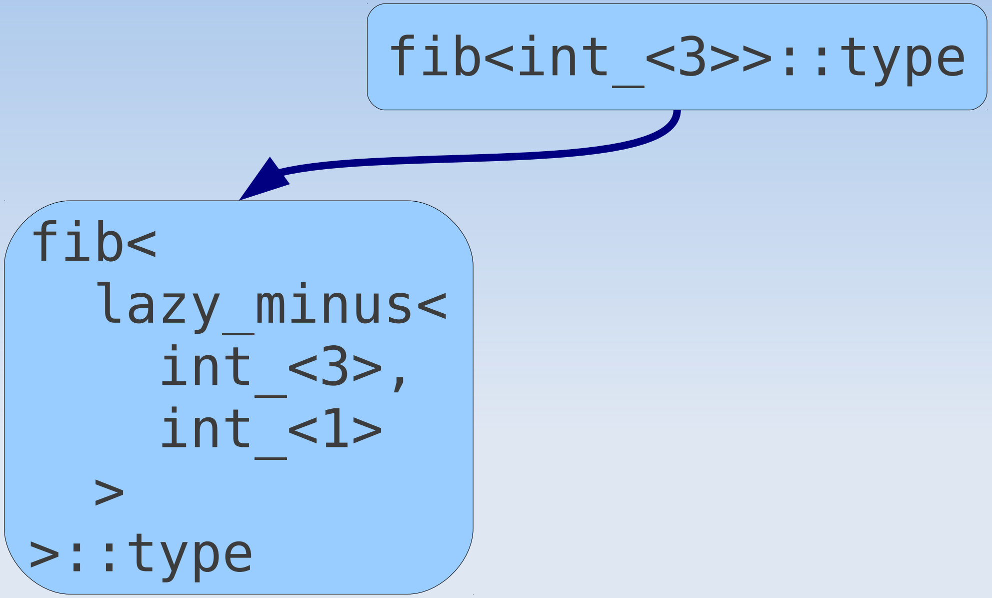
```
MPLLIBS_METAFUNCTION(fact, (N))
((




    mpl::int_<1>







));
```

```
fact<mpl::int_<0>>::type
```

The price of laziness

```
fib<int_<3>>::type
```

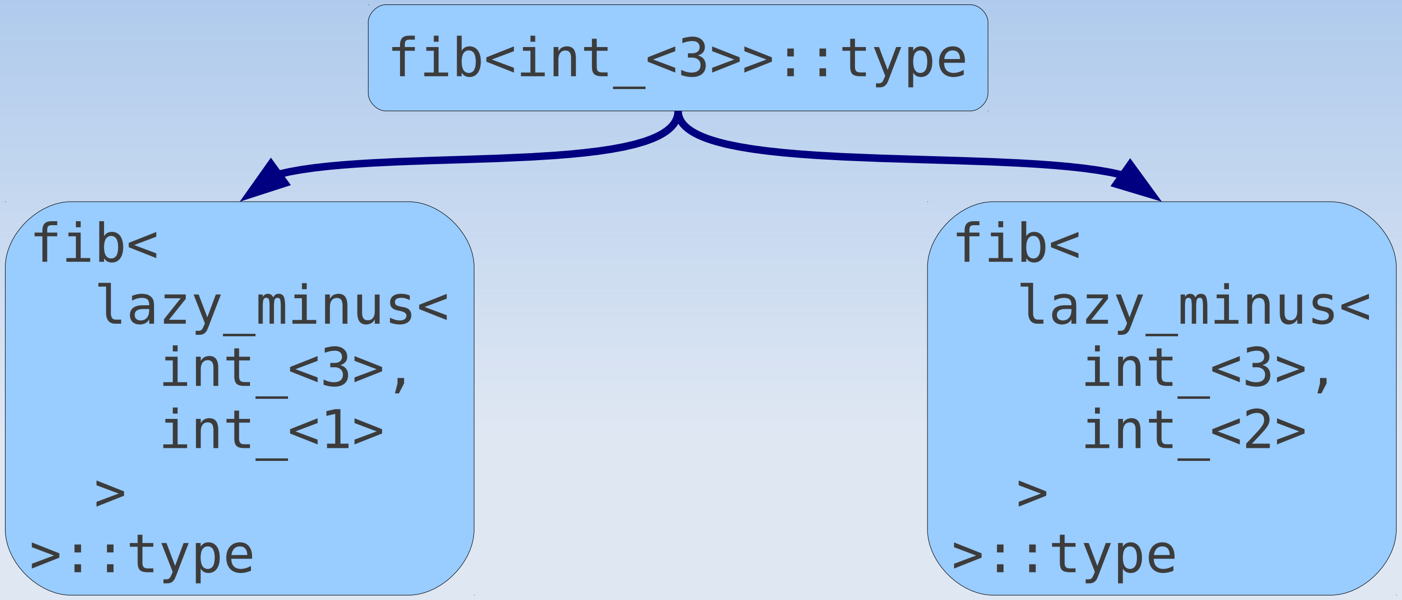
The price of laziness



```
fib<int_<3>>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<1>
  >
>::type
```

The price of laziness

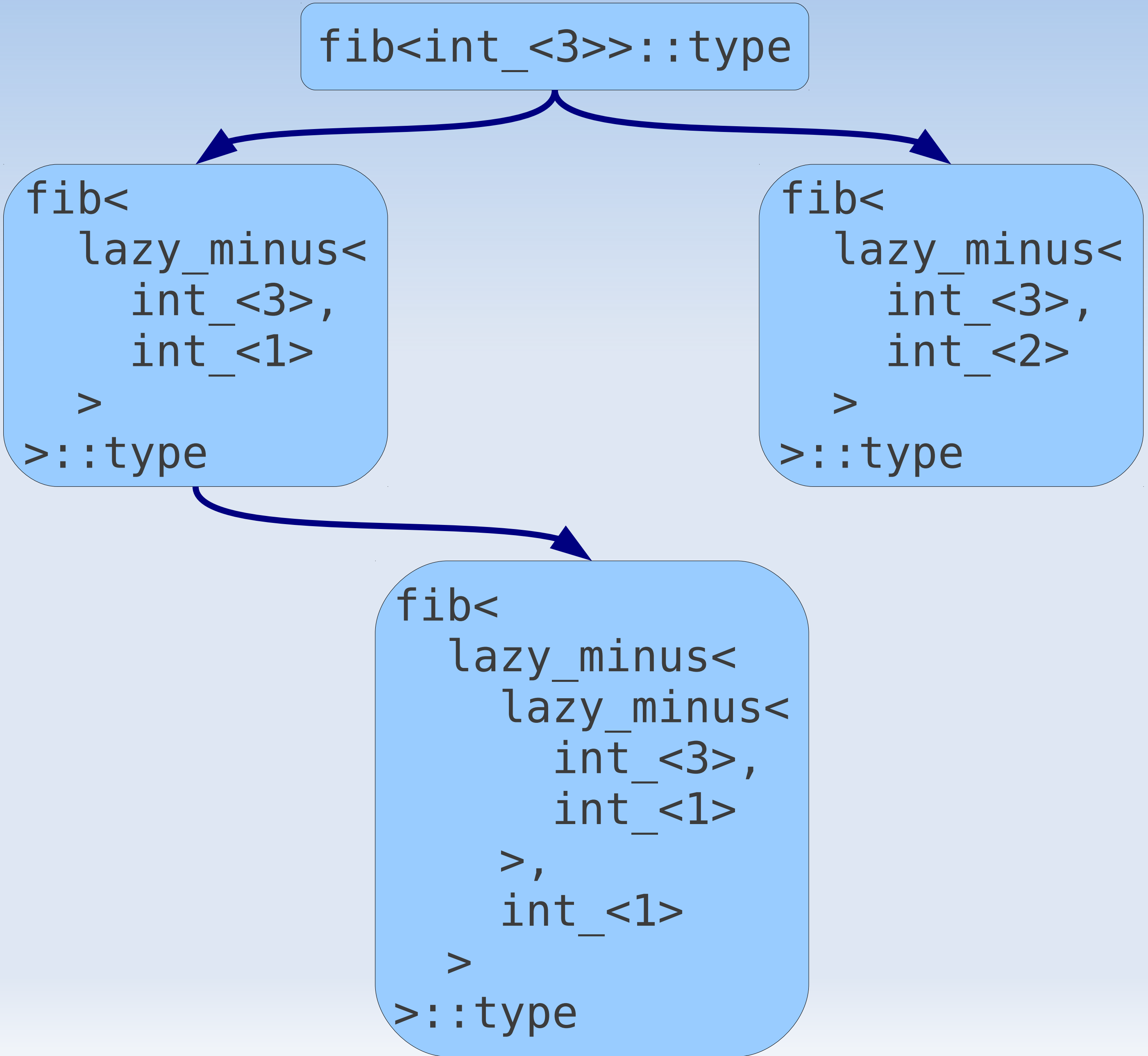


```
fib<int_<3>>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<1>
  >
>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<2>
  >
>::type
```

The price of laziness


fib<int_<3>>::type
fib< lazy_minus< int_<3>, int_<1> > >::type
fib< lazy_minus< int_<3>, int_<2> > >::type
fib< lazy_minus< lazy_minus< int_<3>, int_<1> >, int_<1> > >::type

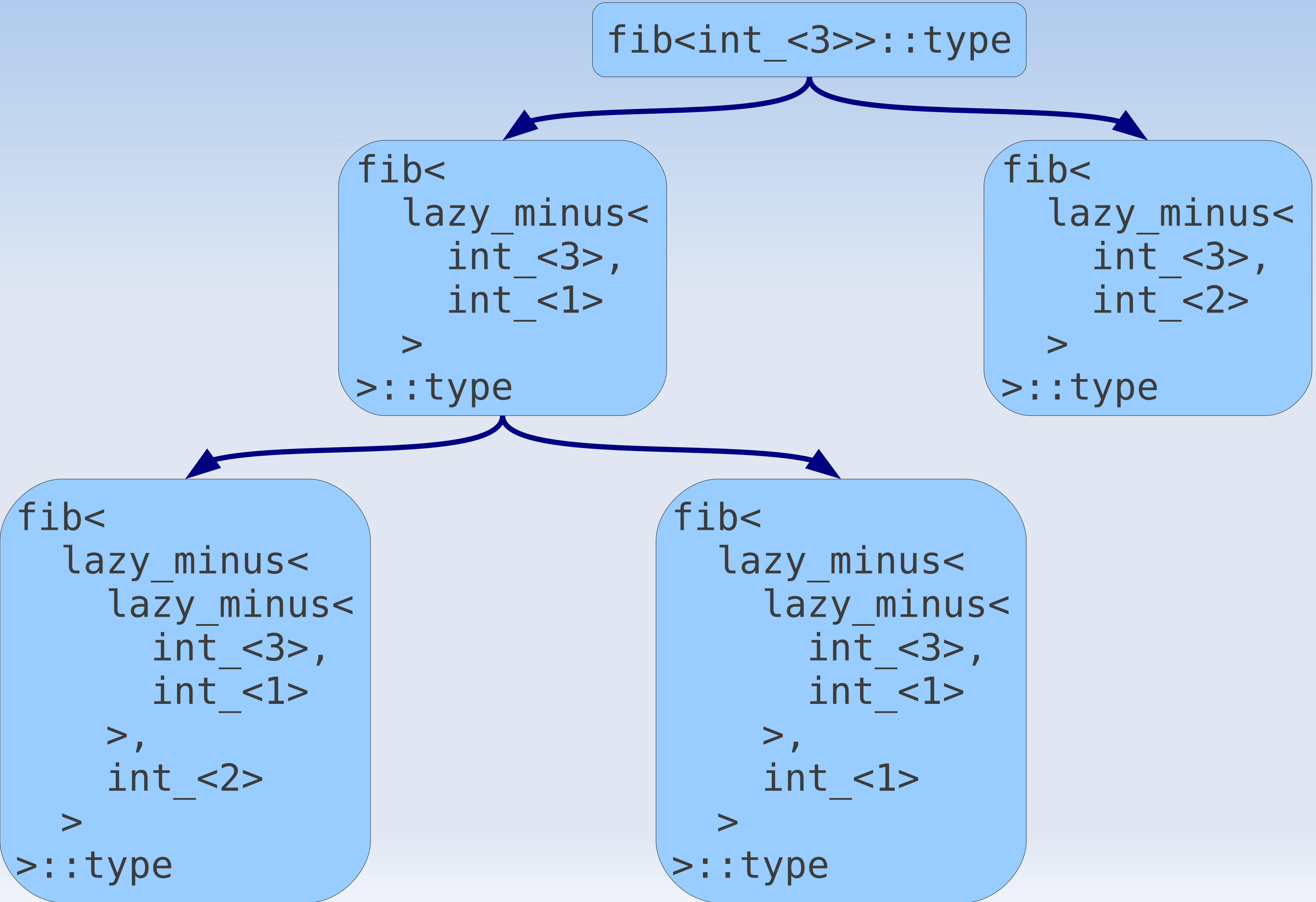
```
fib<int_<3>>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<1>
  >
>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<2>
  >
>::type
```

```
fib<
  lazy_minus<
    lazy_minus<
      int_<3>,
      int_<1>
    >,
    int_<1>
  >
>::type
```

The price of laziness



```
fib<int_<3>>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<1>
  >
>::type
```

```
fib<
  lazy_minus<
    int_<3>,
    int_<2>
  >
>::type
```

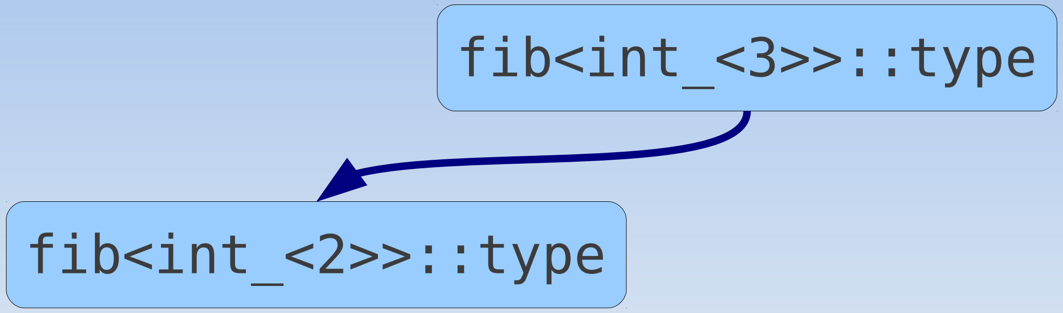
```
fib<
  lazy_minus<
    lazy_minus<
      int_<3>,
      int_<1>
    >,
    int_<2>
  >
>::type
```

```
fib<
  lazy_minus<
    lazy_minus<
      int_<3>,
      int_<1>
    >,
    int_<1>
  >
>::type
```

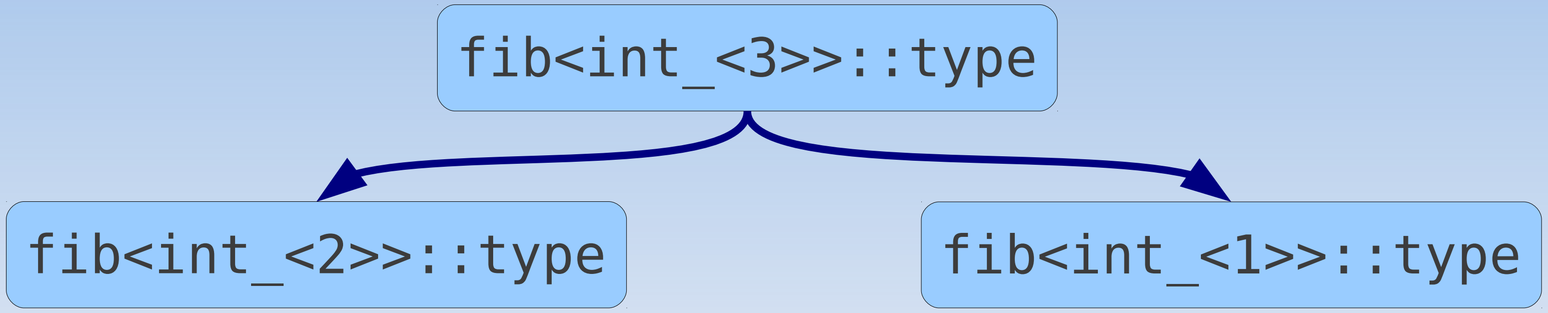
The price of laziness

```
fib<int_<3>>::type
```

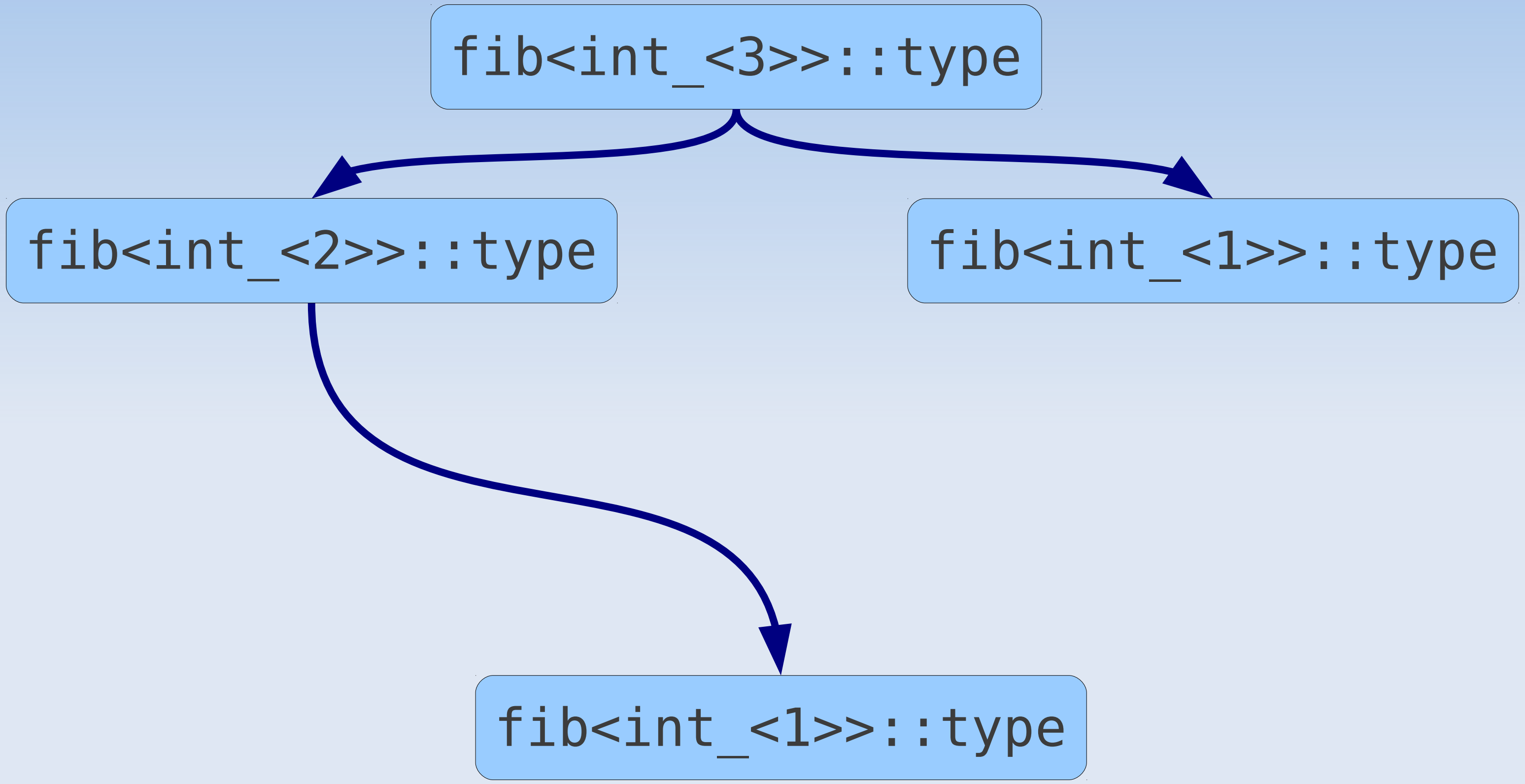
The price of laziness



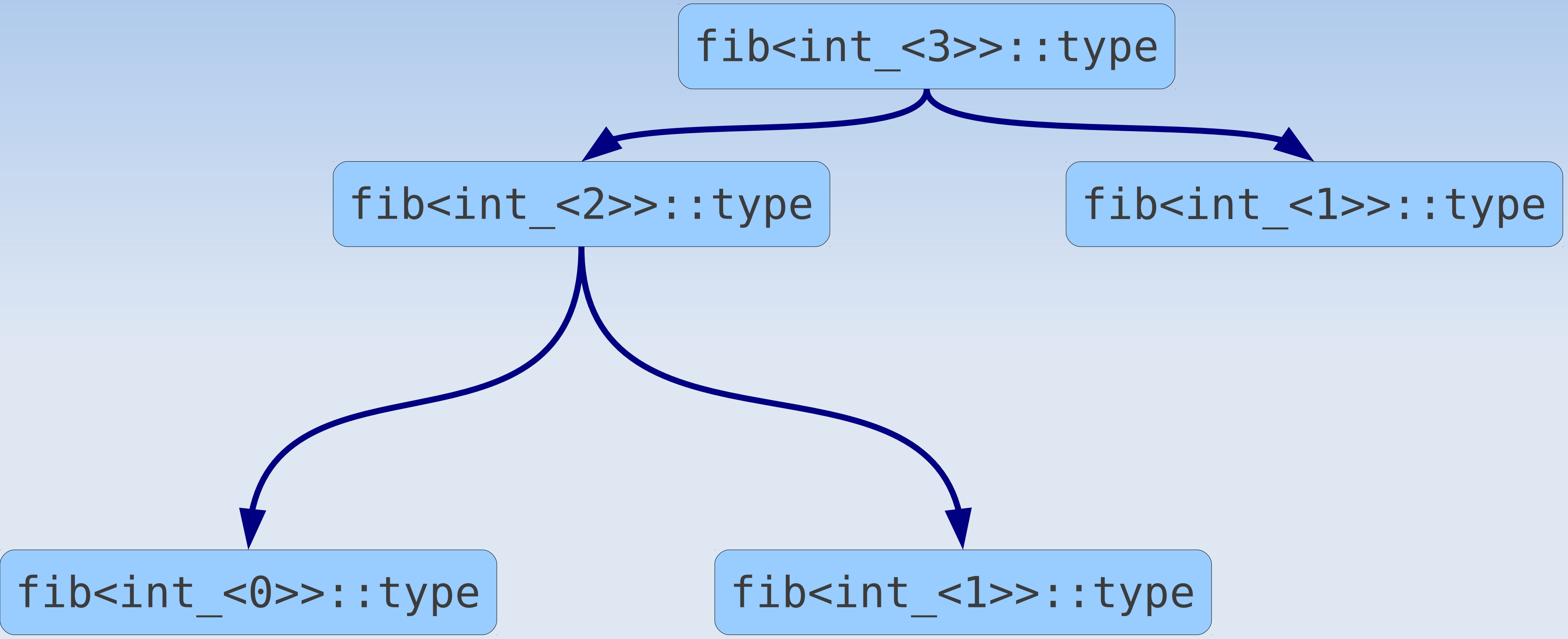
The price of laziness


$$\begin{tikzcd}
 & \texttt{fib<int\_<3>>::type} \arrow[dl] \arrow[dr] & \\
\texttt{fib<int\_<2>>::type} & & \texttt{fib<int\_<1>>::type}
\end{tikzcd}$$

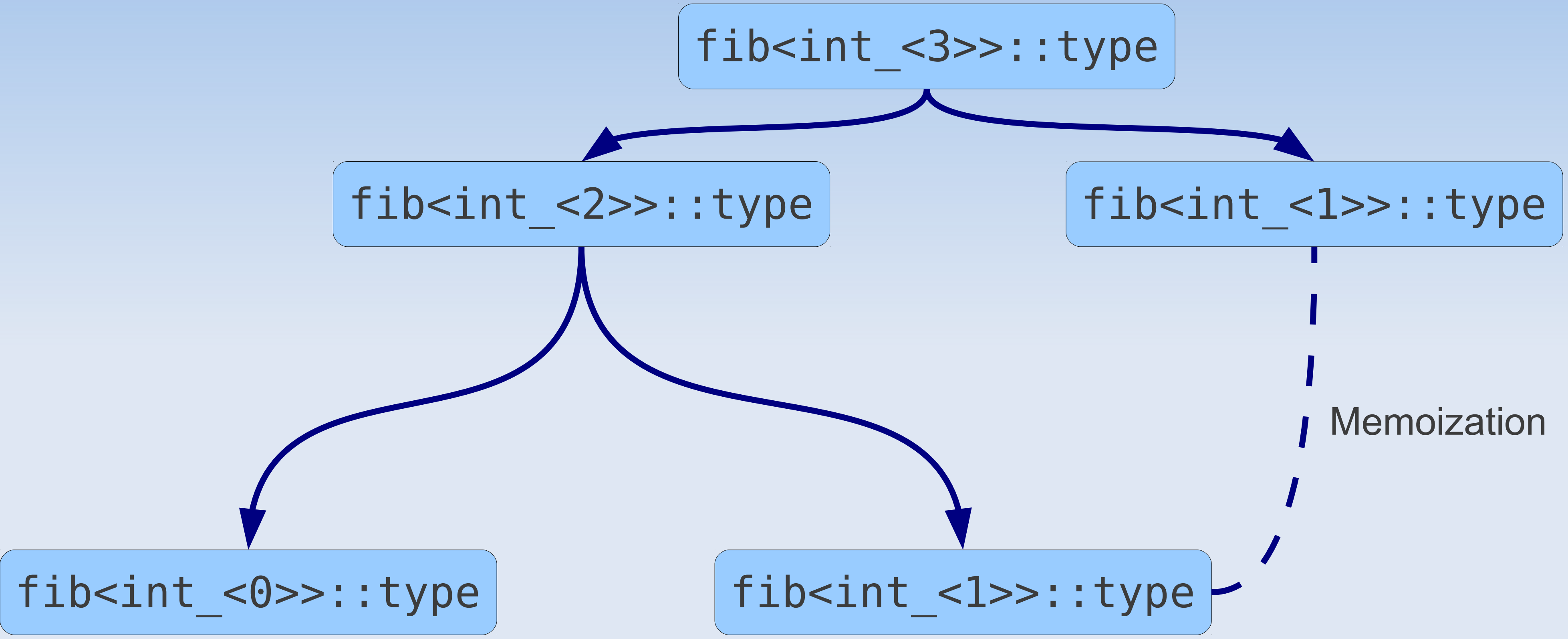
The price of laziness



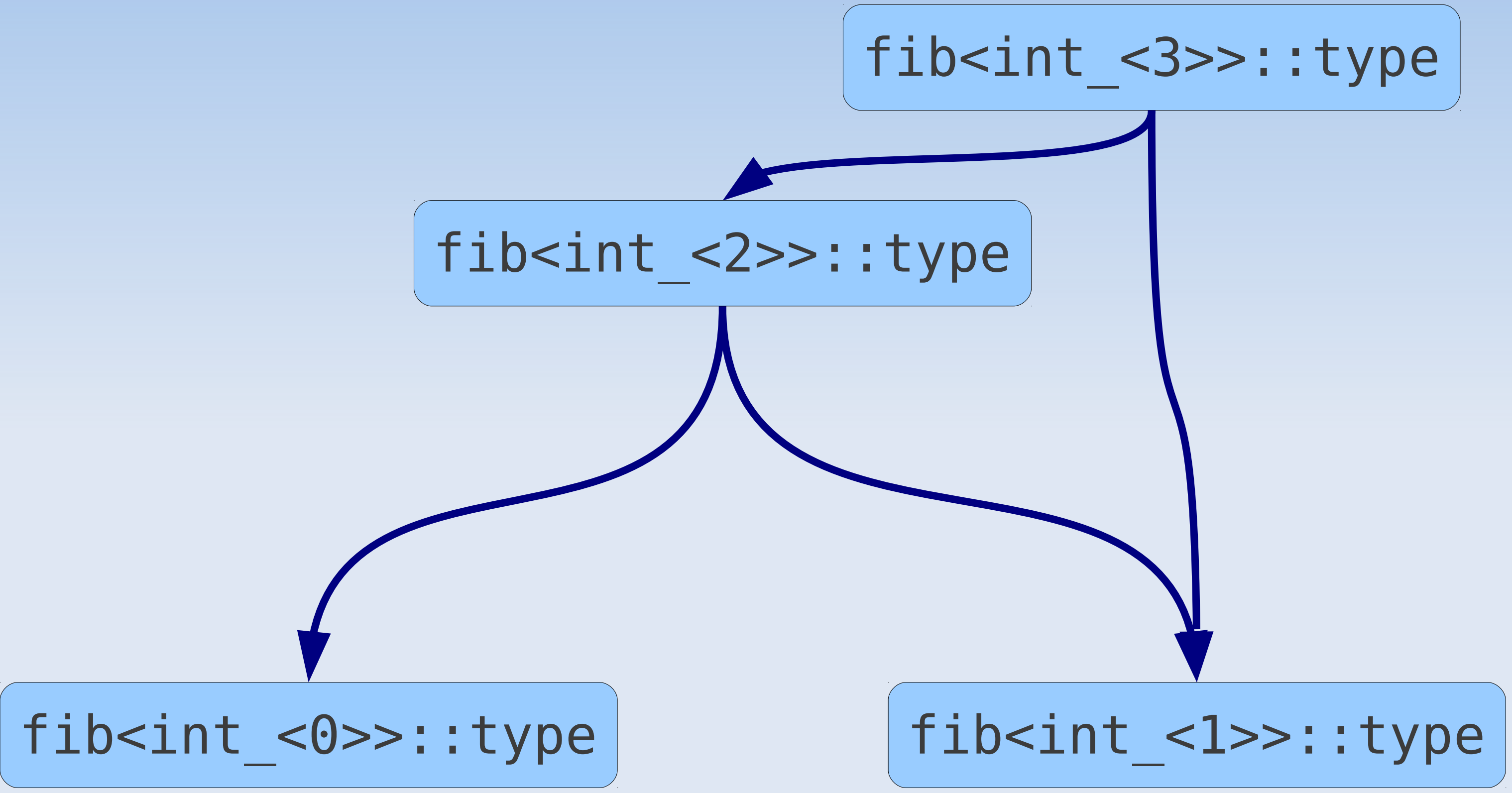
The price of laziness



The price of laziness



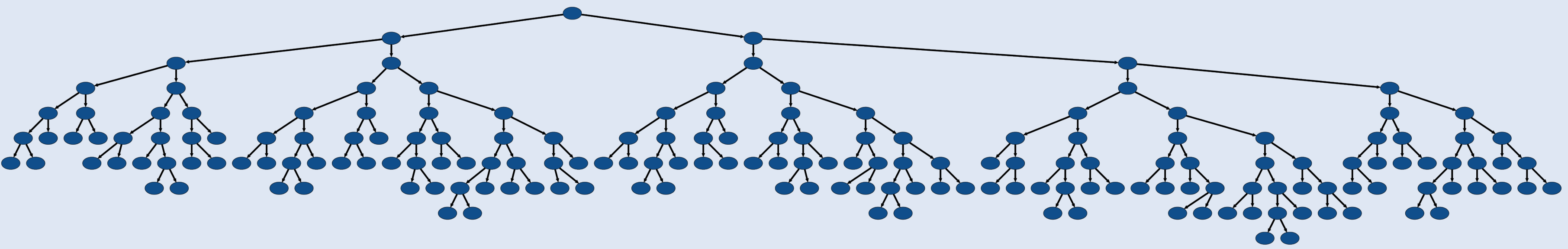
The price of laziness

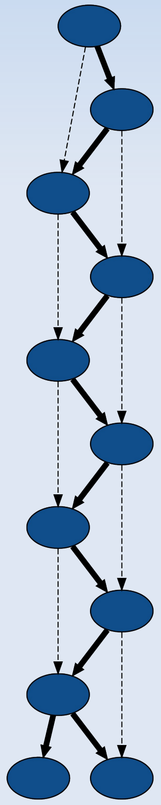

$$\begin{tikzcd}
 & & \text{fib<int_<3>>::type} \arrow[dl] \arrow[d] \\
 & \text{fib<int_<2>>::type} \arrow[dl] \arrow[r] & \text{fib<int_<1>>::type} \\
\text{fib<int_<0>>::type} & &
\end{tikzcd}$$

The price of laziness

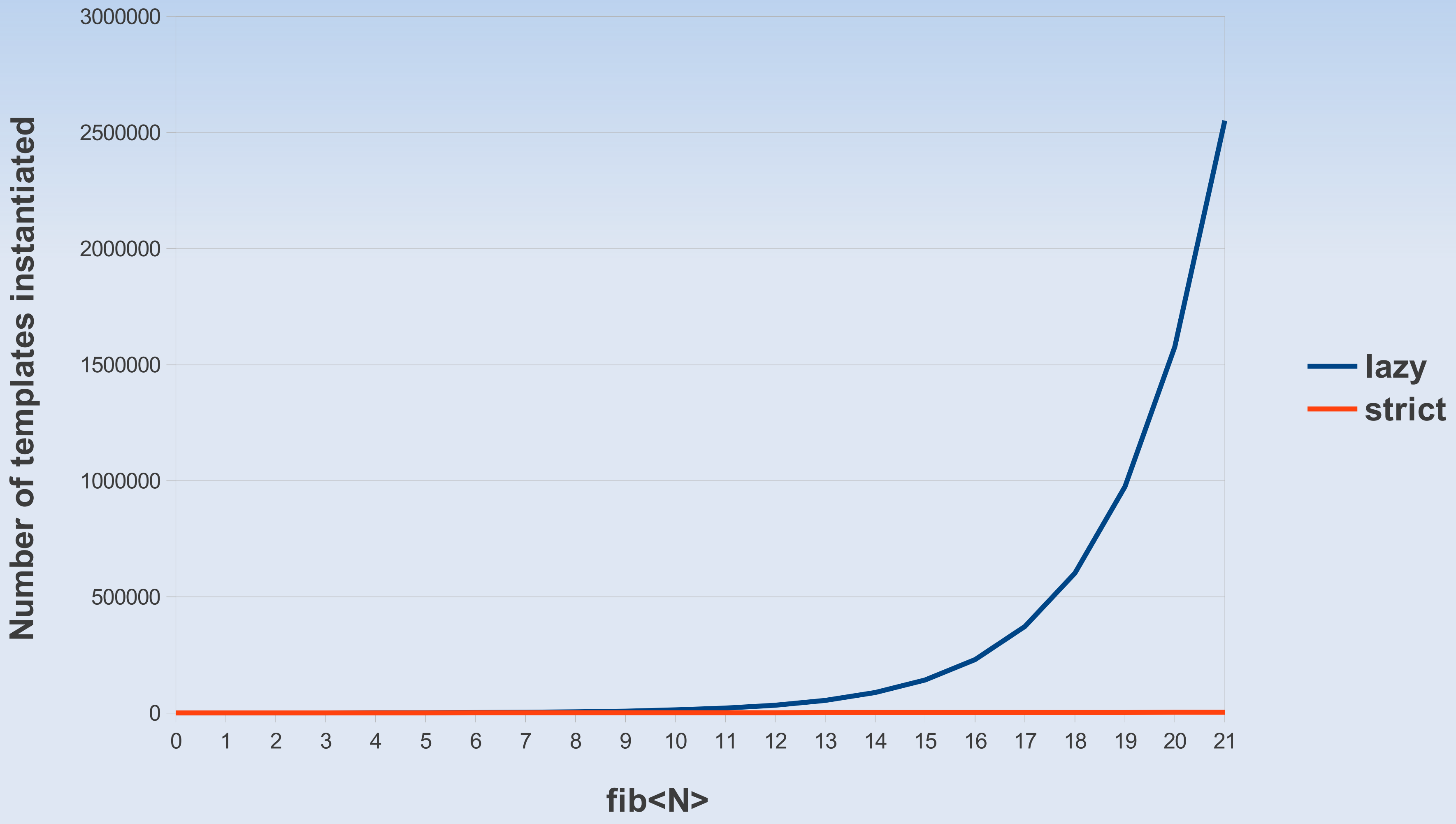
`strict_fib<int_<10>>::type`

`lazy_fib<int_<10>>::type`





The price of laziness



Syntaxes

```
mpl::plus<mpl::int_<11>, mpl::int_<2>>
```

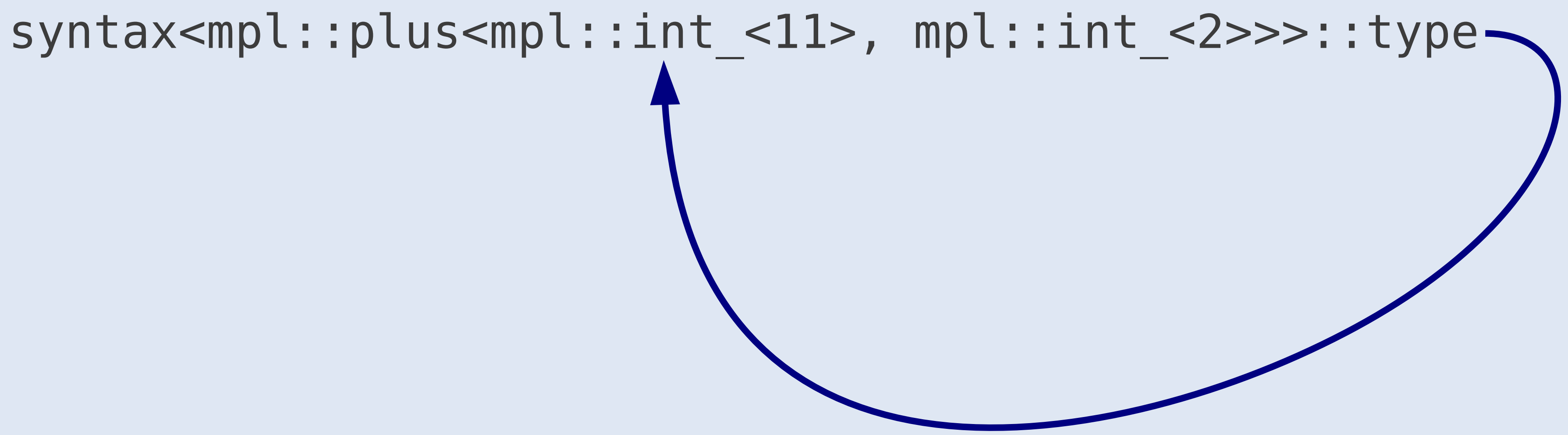
Syntaxes

```
mpl::plus<mpl::int_<11>, mpl::int_<2>>::type

               mpl::int_<13>
```

Syntaxes

```
syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
```

Syntaxes



```
syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>::type
```

Syntaxes

```
eval_syntax<
  syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
>
```

Syntaxes

```
eval_syntax<
  syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
>::type
```

```
mpl::int_<13>
```

Syntaxes

```
struct a_;




    syntax<mpl::plus<mpl::int_<11>,      var<a_>>>
```

Syntaxes

```
struct a_;
typedef var<a_> a;




    syntax<mpl::plus<mpl::int_<11>,          a  >>
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

```
    syntax<mpl::plus<mpl::int_<11>,           a  >>
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

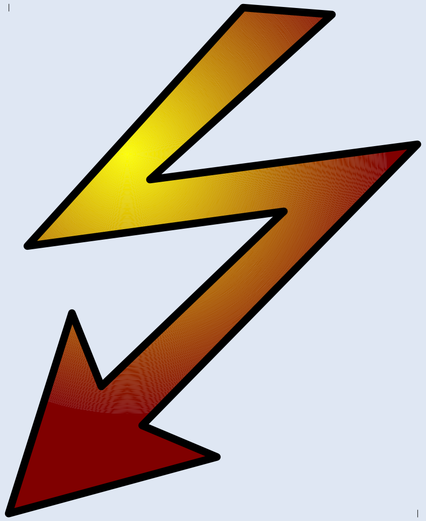
```
  eval_syntax<
    syntax<mpl::plus<mpl::int_<11>,          a  >>
  >::type
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

```
eval_syntax<
  syntax<mpl::plus<mpl::int_<11>,          a  >>
>::type
```

```
mpl::plus<mpl::int_<11>, a>
```



Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z


  let<
    a, syntax<mpl::int_<2>>,
    syntax<mpl::plus<mpl::int_<11>,           a  >>
  >
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z


  let<
    a, syntax<mpl::int_<2>>,
    syntax<mpl::plus<mpl::int_<11>,          a  >>
  >::type
      ↓

    syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

```
  let<
    a, syntax<mpl::int_<2>>,
    syntax<mpl::plus<mpl::int_<11>,          a  >>
  >::type
```

$\downarrow$

```
    syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
```

```
mpl::at<
  mpl::vector<....>,
  mpl::int_<1>
>
```

$\longleftrightarrow$

```
mpl::at_c<
  mpl::vector<....>,
  1
>
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

```
let_c<
  a,        mpl::int_<2> ,
         mpl::plus<mpl::int_<11>,          a  >
>::type
```

```
  syntax<mpl::plus<mpl::int_<11>, mpl::int_<2>>>
```

```
mpl::at<
  mpl::vector<....>,
  mpl::int_<1>
>
```

```
mpl::at_c<
  mpl::vector<....>,
  1
>
```

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z

eval_syntax<
  let_c<
    a,        mpl::int_<2> ,
           mpl::plus<mpl::int_<11>,          a  >
  >
>::type
```

$\xrightarrow{\quad}$ `mpl::int_<13>`

Syntaxes

```
struct a_;
typedef var<a_> a;
// b, c, d, …, z
```

```
  eval_let_c<
    a,         mpl::int_<2> ,
            mpl::plus<mpl::int_<11>,          a  >
  >::type
```

$$\begin{tikzcd} \texttt{>::type} \arrow[r, bend right] & \texttt{mpl::int\_<13>} \end{tikzcd}$$

```
syntax<mpl::plus<a,            b>>
```

Lambdas

```
lambda<      syntax<mpl::plus<a,            b>>>
```

Lambdas

```
lambda<a, b, syntax<mpl::plus<a,            b>>>
```

Lambdas

```
typedef lambda<a, b, syntax<mpl::plus<a,            b>>> add;
```

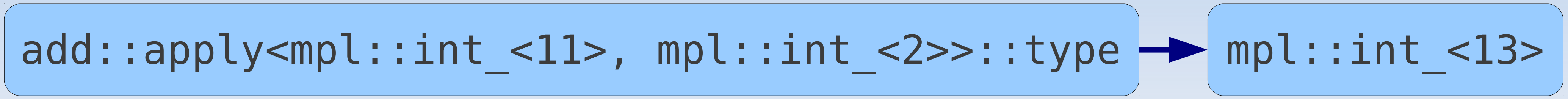
Lambdas

```
typedef lambda<a, b, syntax<mpl::plus<a,           b>>> add;
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type
```

Lambdas

```
typedef lambda<a, b, syntax<mpl::plus<a,            b>>> add;
```



```
add::apply<mpl::int_<11>, mpl::int_<2>>::type
mpl::int_<13>
```

Lambdas

```
typedef lambda_c<a, b,     mpl::plus<a,          b> > add;
```

$$\begin{tikzcd}\boxed{\texttt{add::apply<mpl::int\_<11>, mpl::int\_<2>>::type}} \arrow[r] & \boxed{\texttt{mpl::int\_<13>}}\end{tikzcd}$$

Lambdas

```
typedef lambda_c<a, b,     mpl::plus<a,          b> > add;
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>
```

```
add::apply<mpl::int_<1>>::type
```

Lambdas

```
typedef lambda_c<a, b,     mpl::plus<a,           b> > add;
        lambda_c<   b,     mpl::plus<mpl::int_<1>, b> >

add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>

        add::apply<mpl::int_<1>>::type
```

Lambdas

```
typedef lambda_c<a, b,      mpl::plus<a,             b> > add;
        lambda_c<   b,      mpl::plus<mpl::int_<1>, b> >

add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>


typedef add::apply<mpl::int_<1>>::type inc;
```

Lambdas

```
typedef lambda_c<a, b,      mpl::plus<a,            b> > add;
        lambda_c<   b,      mpl::plus<mpl::int_<1>, b> >
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>
```

```
typedef add::apply<mpl::int_<1>>::type inc;
```

```
inc::apply<mpl::int_<12>>::type → mpl::int_<13>
```

Lambdas

```
typedef lambda_c<a, b,      mpl::plus<a,            b> > add;
        lambda_c<   b,      mpl::plus<mpl::int_<1>, b> >
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>
```

```
typedef add::apply<mpl::int_<1>>::type inc;
```

```
inc::apply<mpl::int_<12>>::type              → mpl::int_<13>
```

```
MPLLIBS_METAFUNCTION(my_plus, (A)(B)) ((mpl::plus<A, B>));
```

Lambdas

```
typedef lambda_c<a, b,     mpl::plus<a,           b> > add;
        lambda_c<   b,     mpl::plus<mpl::int_<1>, b> >
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>
```

```
typedef add::apply<mpl::int_<1>>::type inc;
```

```
inc::apply<mpl::int_<12>>::type → mpl::int_<13>
```

```
MPLLIBS_METAFUNCTION(my_plus, (A)(B)) ((mpl::plus<A, B>));

        my_plus<mpl::int_<1>>::type
```

Lambdas

```
typedef lambda_c<a, b,     mpl::plus<a,          b> > add;
        lambda_c<   b,     mpl::plus<mpl::int_<1>, b> >
```

```
add::apply<mpl::int_<11>, mpl::int_<2>>::type → mpl::int_<13>
```

```
typedef add::apply<mpl::int_<1>>::type inc;
```

```
inc::apply<mpl::int_<12>>::type → mpl::int_<13>
```

```
MPLLIBS_METAFUNCTION(my_plus, (A)(B)) ((mpl::plus<A, B>));

typedef my_plus<mpl::int_<1>>::type inc;
```

Error handling

```
mpl::divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
1 / 0
```

```
mpl::divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((





));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,



  >
));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
struct nothing;


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,

  >
));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
struct nothing;
template <class T> struct just;

MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
struct nothing;
template <class T> struct just;

MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

```
just<mpl::int_<13>>
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

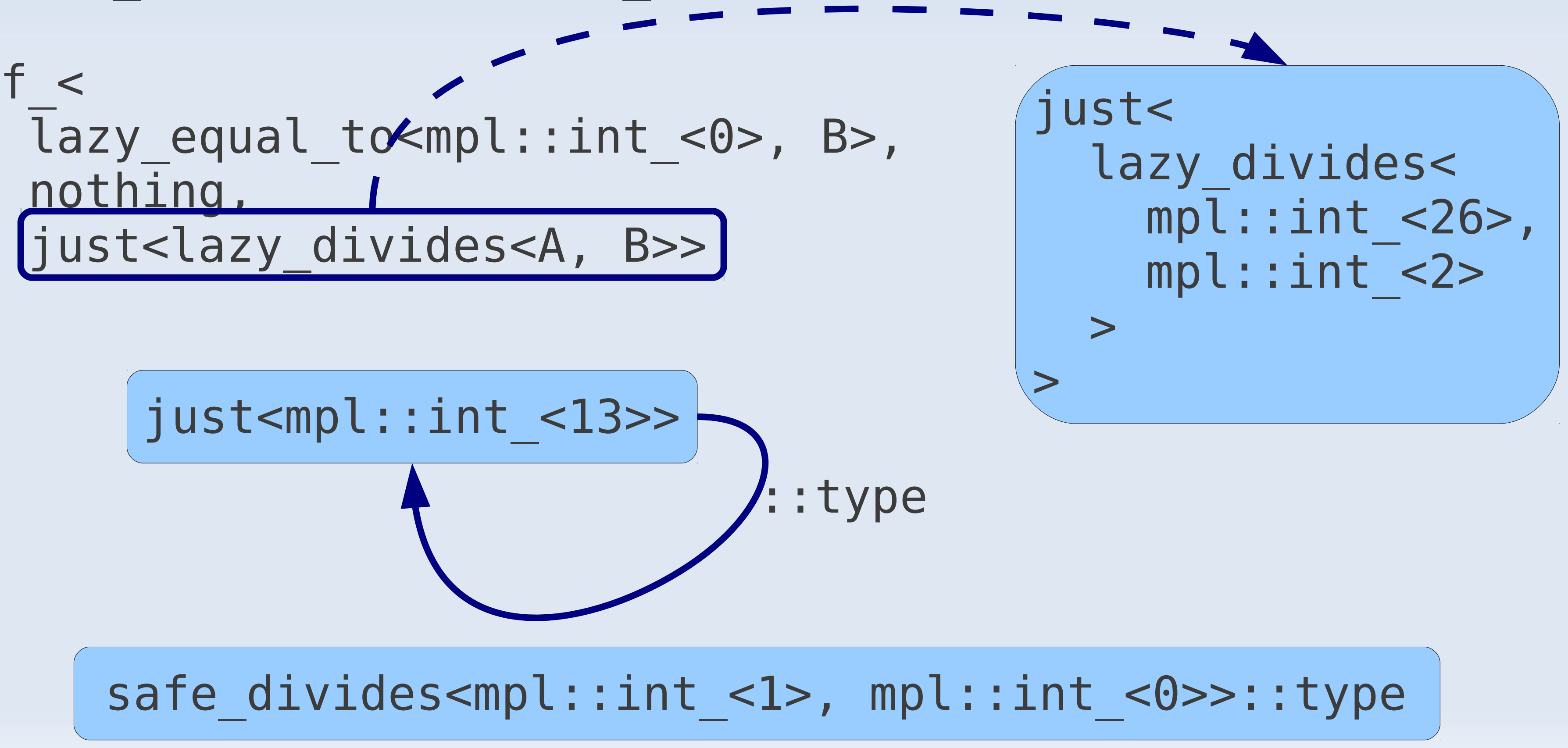
```
just<mpl::int_<13>>
```

```
::type
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```



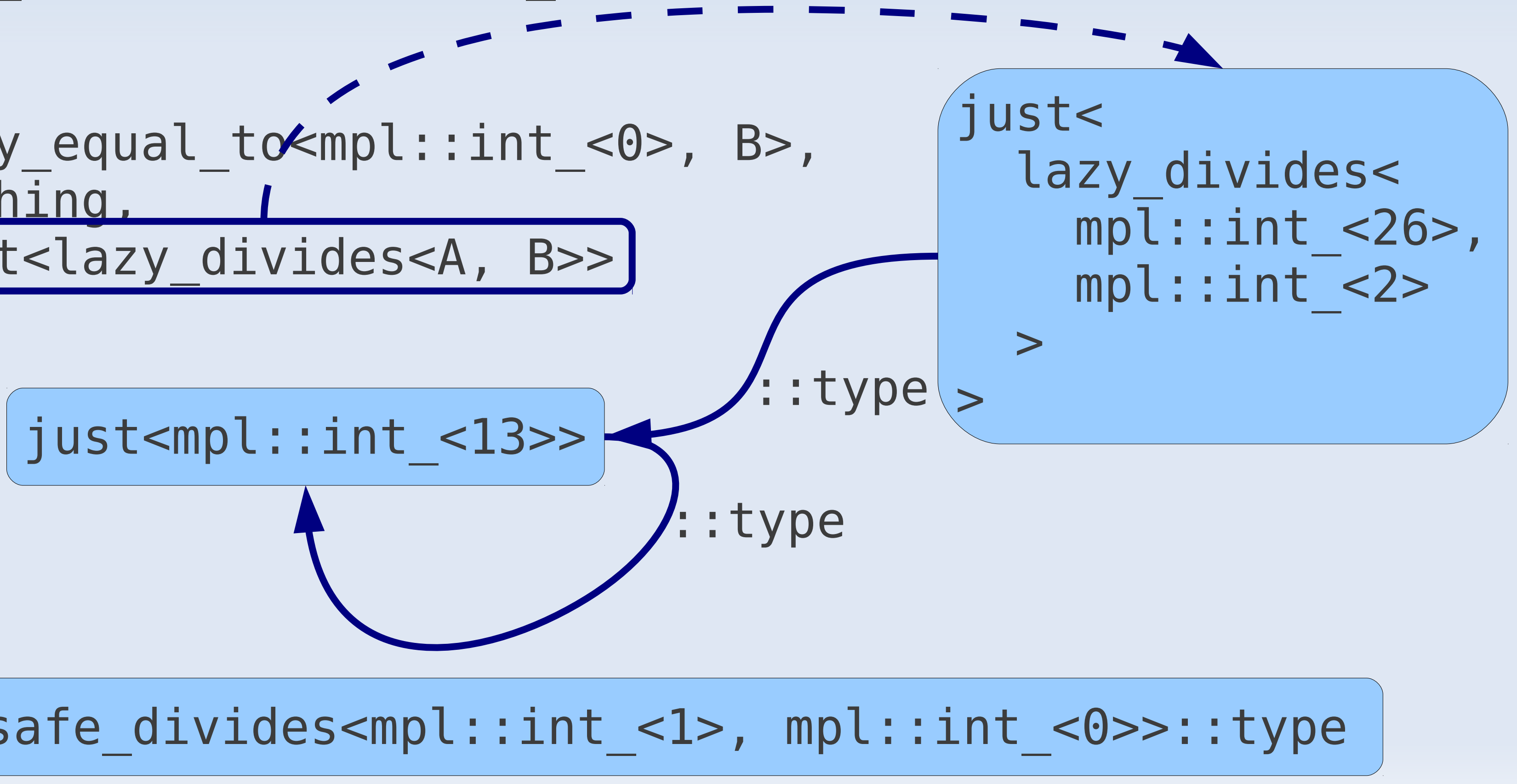
```
just<
  lazy_divides<
    mpl::int_<26>,
    mpl::int_<2>
  >
>
```

```
just<mpl::int_<13>>
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```



```
just<
  lazy_divides<
    mpl::int_<26>,
    mpl::int_<2>
  >
>
```

```
just<mpl::int_<13>>
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
just<T>
```

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```

```
just<
  lazy_divides<
    mpl::int_<26>,
    mpl::int_<2>
  >
>
```

::type

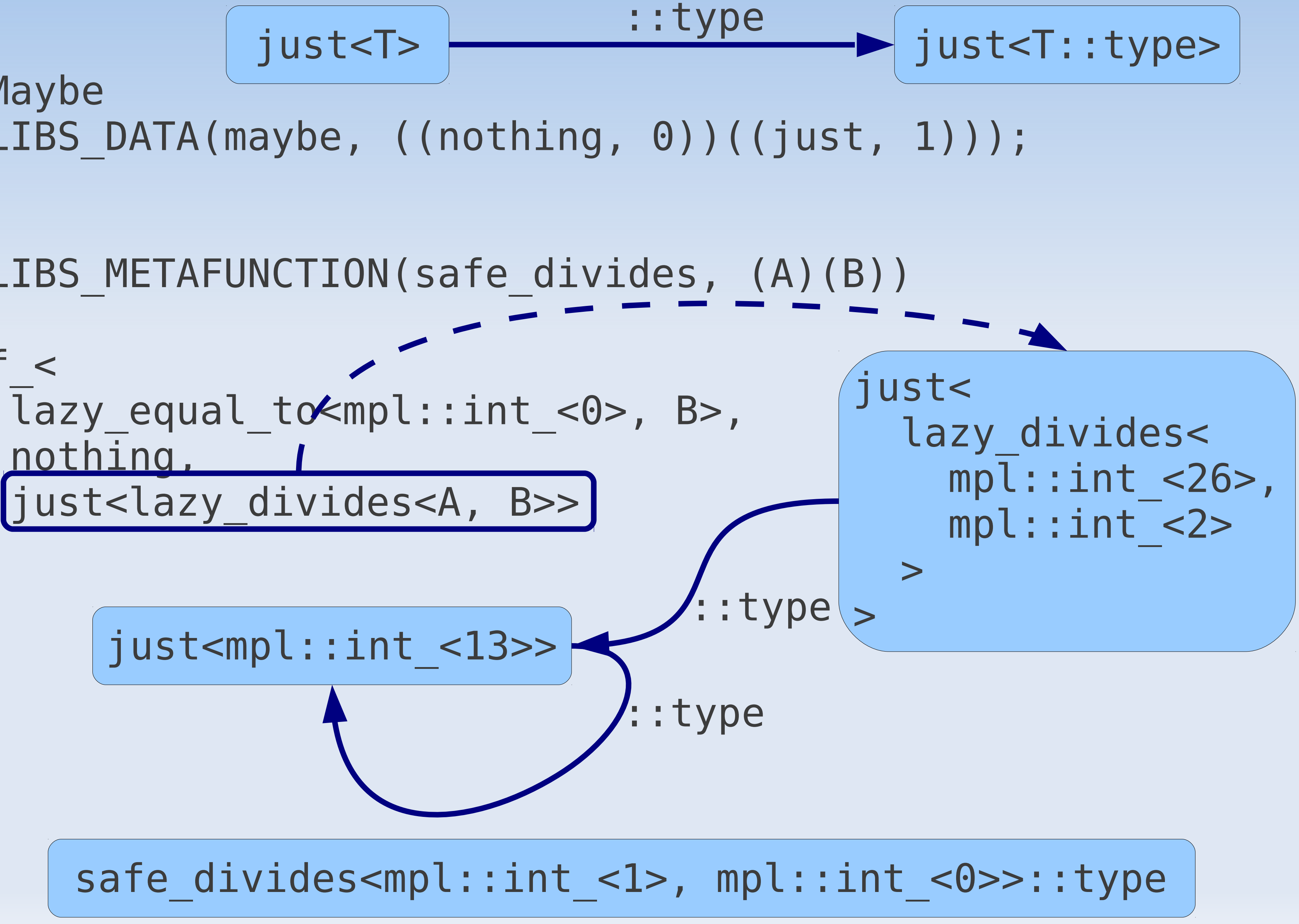
```
just<mpl::int_<13>>
```

::type

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
// Maybe
MPLLIBS_DATA(maybe, ((nothing, 0))((just, 1)));


MPLLIBS_METAFUNCTION(safe_divides, (A)(B))
((
  if_<
    lazy_equal_to<mpl::int_<0>, B>,
    nothing,
    just<lazy_divides<A, B>>
  >
));
```



```
just<T>
```

```
just<T::type>
```

```
just<
  lazy_divides<
    mpl::int_<26>,
    mpl::int_<2>
  >
>
```

```
just<mpl::int_<13>>
```

```
safe_divides<mpl::int_<1>, mpl::int_<0>>::type
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((





));
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((




));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((




));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type ⟶ mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type ⟶ mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
                  safe_divides<A, B>




));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type
div_or_first<mpl::int_<1>, mpl::int_<0>>::type
```

$\longrightarrow$ `mpl::int_<3>`

$\longrightarrow$ `mpl::int_<1>`

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,



  >
));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type ⟶ mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type ⟶ mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,
    A,

  >
));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type ➞ mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type ➞ mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,
    A,
    ???
  >
));
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type
```
$\longrightarrow$
```
mpl::int_<3>
```

```
div_or_first<mpl::int_<1>, mpl::int_<0>>::type
```
$\longrightarrow$
```
mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,
    A,
    ???
  >
));
```

```
safe_divides<mpl::int_<6>, mpl::int_<2>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type ──► mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type ──► mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,
    A,
    ???
  >
));
```

$$\begin{tikzcd}
\boxed{\texttt{safe\_divides<mpl::int\_<6>, mpl::int\_<2>>}} \arrow[r] & \boxed{\texttt{just<mpl::int\_<3>>}}
\end{tikzcd}$$

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  if_<
    lazy_is_same<safe_divides<A, B>, nothing>,
    A,
    mpl::int_<3>
  >
));
```

```
safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
      case  safe_divides<A, B> of
                 just<n> →        n
                 nothing →        A

));
```

```
safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

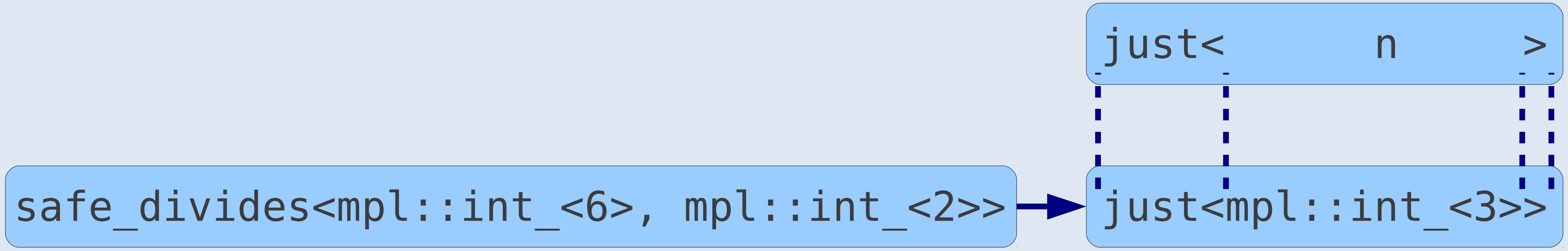
```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  eval_case< safe_divides<A, B>,
    matches_c<     just<n> ,         n >,
    matches_c<     nothing ,         A >
  >
));
```

```
safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  eval_case< safe_divides<A, B>,
    matches_c<     just<n> ,          n >,
    matches_c<     nothing ,          A >
  >
));
```

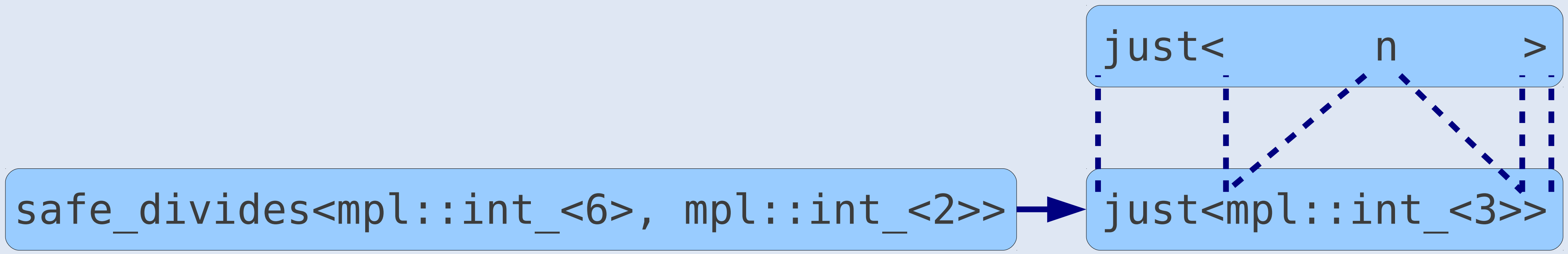


```
just<        n        >

safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  eval_case< safe_divides<A, B>,
    matches_c<     just<n> ,         n >,
    matches_c<     nothing ,         A >
  >
));
```

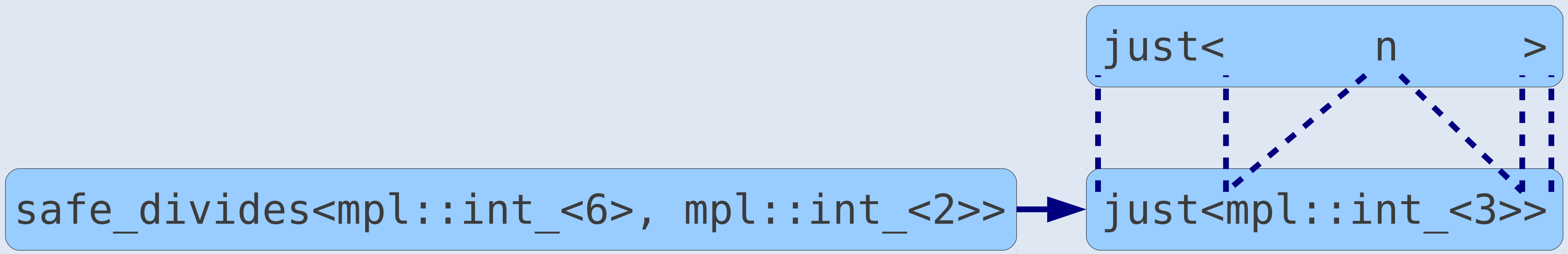


```
just<        n        >
safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

Error handling

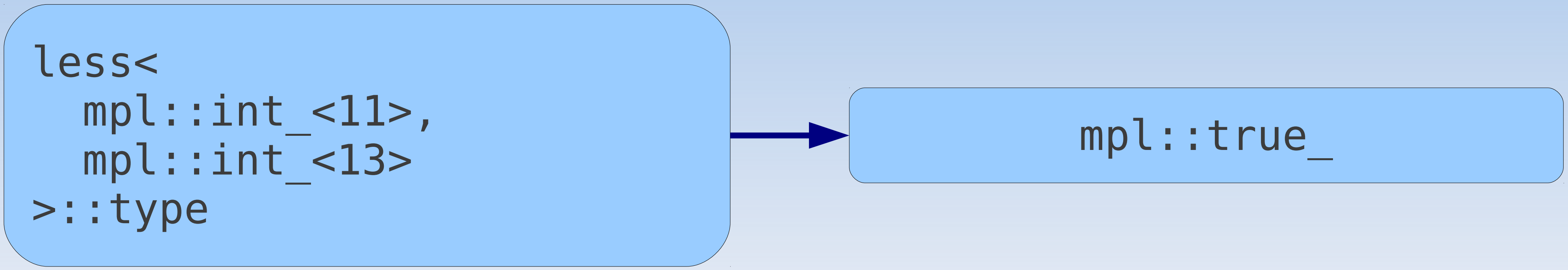
```
MPLLIBS_METAFUNCTION(div_or_first, (A)(B))
((
  eval_case< safe_divides<A, B>,
    matches<syntax<just<n>>, syntax<n>>,
    matches<syntax<nothing>, syntax<A>>
  >
));
```



```
just<     n     >
safe_divides<mpl::int_<6>, mpl::int_<2>> → just<mpl::int_<3>>
```

```
div_or_first<mpl::int_<6>, mpl::int_<2>>::type → mpl::int_<3>
div_or_first<mpl::int_<1>, mpl::int_<0>>::type → mpl::int_<1>
```

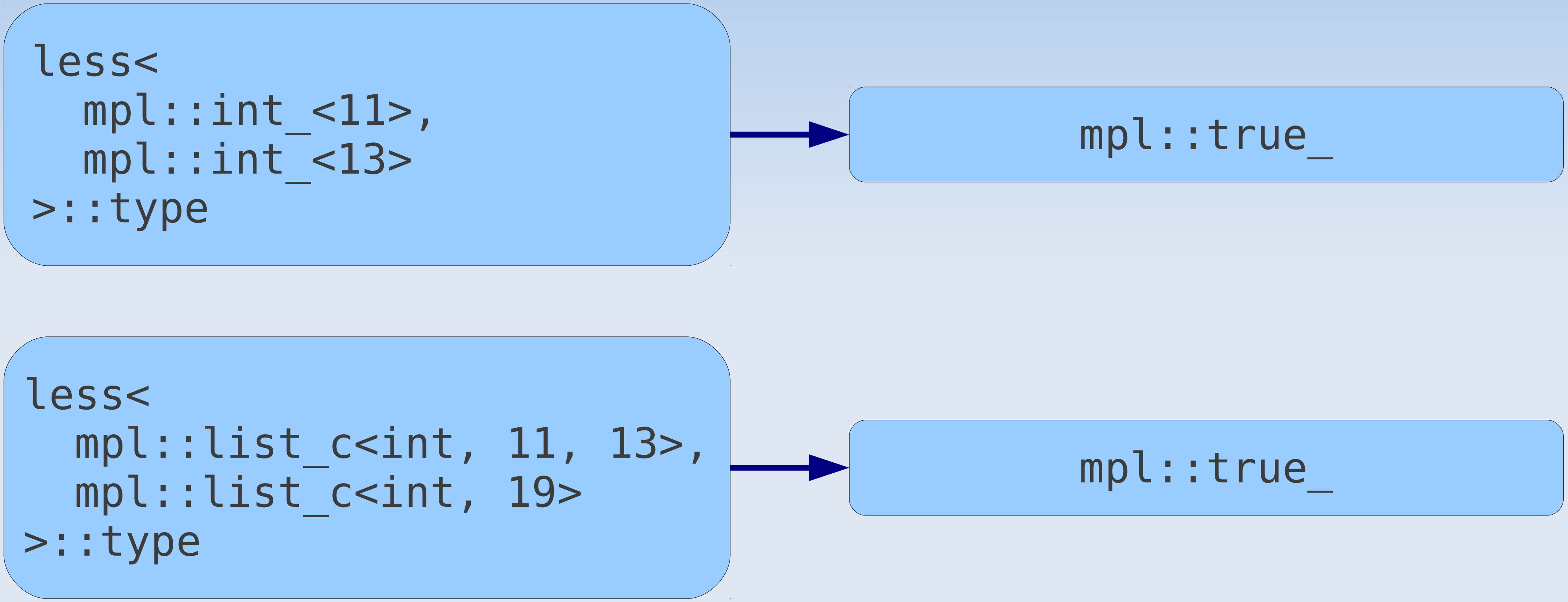
less



```
less<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::true_
```

less



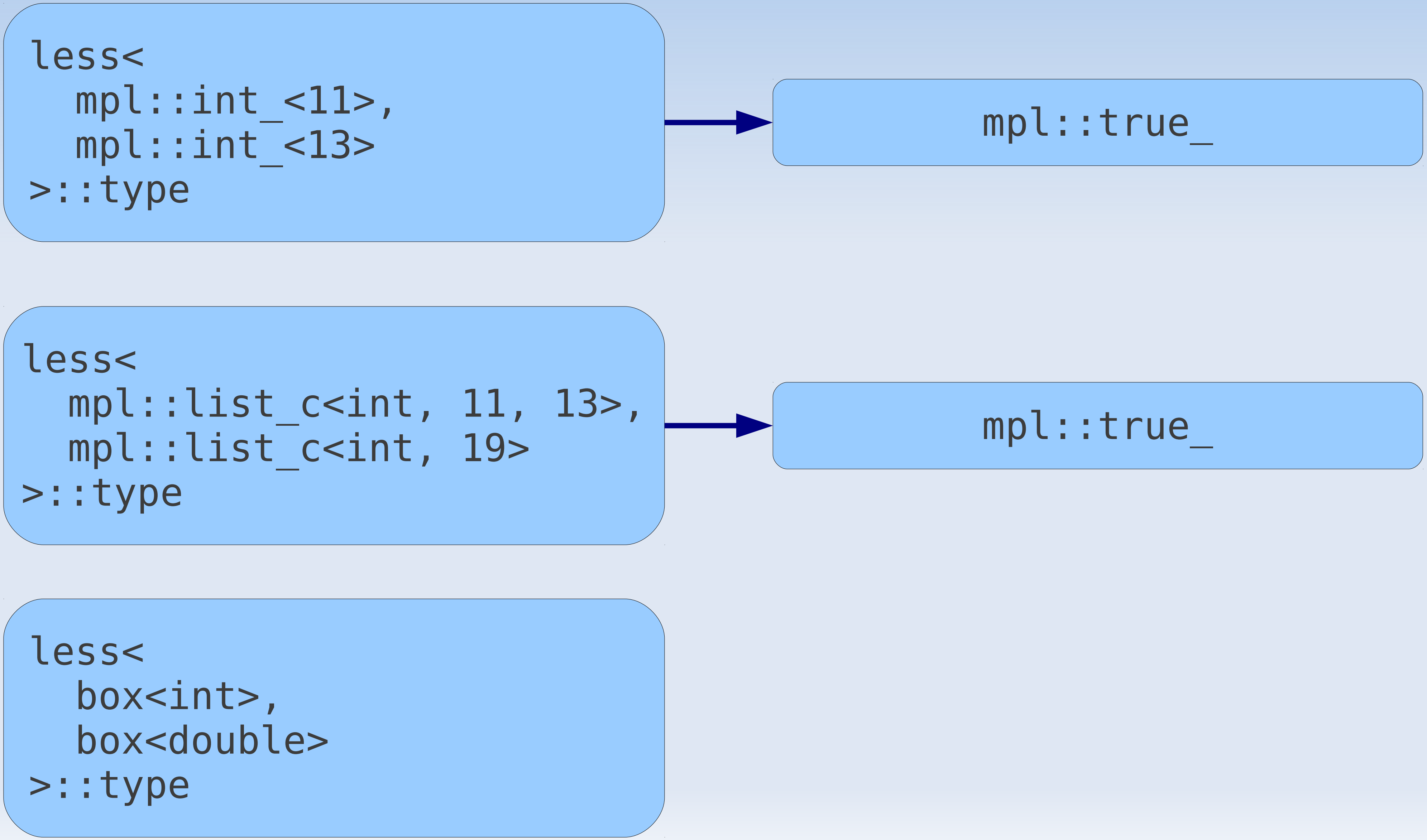
```
less<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::true_
```

```
less<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::true_
```

less



```
less<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::true_
```

```
less<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::true_
```

```
less<
  box<int>,
  box<double>
>::type
```

less

```
MPLLIBS_DATA(exception,    ((exception, 1)));
```

```
less<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

→

```
mpl::true_
```

```
less<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

→

```
mpl::true_
```

```
less<
  box<int>,
  box<double>
>::type
```

less

```
MPLLIBS_DATA(exception,    ((exception, 1)));
```

```
less<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

$\rightarrow$

```
mpl::true_
```

```
less<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

$\rightarrow$

```
mpl::true_
```

```
less<
  box<int>,
  box<double>
>::type
```

$\rightarrow$

```
exception<
 values_can_not_be_compared
>
```

Min function

```
MPLLIBS_METAFUNCTION(min,            (A)(B))
((

                         if_<less<A, B>, A, B>

));
```

Min function

```
MPLLIBS_METAFUNCTION(min,                    (A)(B))
((


                          if_<less<A, B>, A, B>


));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

Min function

```
MPLLIBS_METAFUNCTION(min,                    (A)(B))
((


                         if_<less<A, B>, A, B>


));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
  mpl::int_<11>
```

Min function

```
MPLLIBS_METAFUNCTION(min,                (A)(B))
((

                          if_<less<A, B>, A, B>

));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
  mpl::int_<11>
```

Min function

```
MPLLIBS_METAFUNCTION(min,                    (A)(B))
((

                          if_<less<A, B>, A, B>

));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
min<
  box<int>,
  box<double>
>::type
```

```
NCTION(min,                (A)(B))

                    if_<less<A, B>, A, B>

));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
  mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
  mpl::list_c<int, 11, 13>
```

Min function

```
min<
  box<int>,
  box<double>
>::type
```

```
NCTION(min,            (A)(B))

                  if_<less<A, B>, A, B>

));
```

```
In file included from /usr/include/boost/mpl/eval_if.hpp:17:0,
                 from /usr/include/boost/mpl/aux_/begin_end_impl.hpp:20,
                 from /usr/include/boost/mpl/begin_end.hpp:18,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/impl/let.hpp:22,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/let.hpp:9,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/impl/lambda.hpp:9,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/lambda_c.hpp:9,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/curried_call.hpp:10,
                 from /home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/metafunction.hpp:11,
                 from main.cpp:1:
/usr/include/boost/mpl/if.hpp: In instantiation of 'struct boost::mpl::if_<mpllibs::metamonad::exc
eption<values_can_not_be_compared>, mpllibs::metamonad::box<int>, mpllibs::metamonad::box<double>
>':
main.cpp:29:1:   required from 'struct min___impl<mpllibs::metamonad::box_tag, mpllibs::metamonad:
:box<int>, mpllibs::metamonad::box<double> >'
/home/abel/git/github/sabel83/mpllibs/mpllibs/metamonad/curried_call.hpp:87:1:   required from 'st
ruct mpllibs::metamonad::curried_call3<min___impl, mpllibs::metamonad::box_tag, mpllibs::metamonad
::box<int>, mpllibs::metamonad::box<double> >'
main.cpp:29:1:   required from 'struct min<mpllibs::metamonad::box_tag, mpllibs::metamonad::box<in
t>, mpllibs::metamonad::box<double> >'
main.cpp:36:65:   required from here
/usr/include/boost/mpl/if.hpp:67:11: error: 'value' is not a member of 'mpllibs::metamonad::except
ion<values_can_not_be_compared>'
/usr/include/boost/mpl/if.hpp:70:41: error: 'value' is not a member of 'mpllibs::metamonad::except
ion<values_can_not_be_compared>'
```

```
min<
  mpl::int_<11
  mpl::int_<13
>::type
```

```
11, 13>,
19>
```

```
mpl::int_<11>
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(min,                    (A)(B))
((
  eval_case<less<A, B>,

    matches_c<b,                    if_<less<A, B>, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(min,            (A)(B))
((
  eval_case<less<A, B>,
    matches_c<b,          if_<less<A, B>, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(min,            (A)(B))
((
  eval_case<less<A, B>,

    matches_c<b,          if_<         b, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(min,            (A)(B))
((
  eval_case<less<A, B>,

    matches_c<b,                   if_<b, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(min,                    (A)(B))
((
  eval_case<less<A, B>,
    matches_c<exception<e>, exception<e>>,
    matches_c<b,                             if_<b, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
min<
  box<int>,
  box<double>
>::type
```

```
NCTION(min,              (A)(B))
ss<A, B>,
exception<e>, exception<e>>,
matches_c<b,                         if_<b, A, B>>
  >
));
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
NCTION(min,                     (A)(B))
ss<A, B>,
exception<e>, exception<e>>,
matches_c<b,                          if_<b, A, B>>
  >
));
```

```
min<
  box<int>,
  box<double>
>::type
```

```
exception<values_can_not_be_compared>
```

```
min<
  mpl::int_<11>,
  mpl::int_<13>
>::type
```

```
mpl::int_<11>
```

```
min<
  mpl::list_c<int, 11, 13>,
  mpl::list_c<int, 19>
>::type
```

```
mpl::list_c<int, 11, 13>
```

Min function

```
MPLLIBS_METAFUNCTION(bind_exception,       )
((
  eval_case<less<A, B>,
    matches_c<exception<e>, exception<e>>,
    matches_c<b,                          if_<b, A, B>>
  >
));
```

Min function

```
MPLLIBS_METAFUNCTION(bind_exception, (X)    )
((
  eval_case<X,
    matches_c<exception<e>, exception<e>>,
    matches_c<b,                          if_<b, A, B>>
  >
));
```

```
                          less<A, B>
```

Min function

```
MPLLIBS_METAFUNCTION(bind_exception, (X)(F))
((
  eval_case<X,
    matches_c<exception<e>, exception<e>>,
    matches_c<b, mpl::apply<F, b>>
  >
));
```

```
less<A, B>
lambda_c<l, mpl::if_<l, A, B>>
```

Min function

```
MPLLIBS_METAFUNCTION(bind_exception, (X)(F))
((
  eval_case<X,
    matches_c<exception<e>, exception<e>>,
    matches_c<b, mpl::apply<F, b>>
  >
));
```

```
                       less<A, B>
                       lambda_c<l, mpl::if_<l, A, B>>
```

Min function

```
MPLLIBS_METAFUNCTION(bind_exception, (X)(F))
((
  eval_case<X,
    matches_c<exception<e>, exception<e>>,
    matches_c<b, mpl::apply<F, b>>
  >
));
```

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Min function

```
MPLLIBS_METAFUNCTIO
((
  eval_case<X,
    matches_c<exce
    matches_c<b, m
  >
));
```

```
MPLLIBS_METAFUNCTION(sum3, (A)(B)(C))
((
  bind_exception<
    mpl::plus<A, B>,
    lambda_c<d, mpl::plus<d, C>>
  >
));
```

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>,
        lambda_c<k, mpl::if_<k, m, C>>
      >
    >
  >
));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>
        lambd
      >
    >
  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)(C))
((

    set<l, less<A, B>>




));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>,
        lambd
      >
    >
  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)(C))
((

    set<l, less<A, B>>,
    set<m, mpl::if_<l, A, B>>



));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>,
        lambd
      >
    >
  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)(C))
((


    set<l, less<A, B>>,
    set<m, mpl::if_<l, A, B>>,
    set<k, less<m, C>>



));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>
        lambd
      >
    >
  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)(C))
((

    set<l, less<A, B>>,
    set<m, mpl::if_<l, A, B>>,
    set<k, less<m, C>>,
    do_return<mpl::if_<k, m, C>>

));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B)(C))
((
  bind_exception<
    bind_exception<
      less<A, B>,
      lambda_c<l, mpl::if_<l, A, B>>
    >,
    lambda_c<m,
      bind_exception<
        less<m, C>
        lambd
      >
    >
  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)(C))
((
  do_<exception_tag,
    set<l, less<A, B>>,
    set<m, mpl::if_<l, A, B>>,
    set<k, less<m, C>>,
    do_return<mpl::if_<k, m, C>>
  >
));
```

Min function

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)    )
((
  do_<exception_tag,
    set<l, less<A, B>>,


    do_return<mpl::if_<l, A, B>>
  >
));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((

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





));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)    )
((
  do_<exception_tag,
    set<l, less<A, B>>,


    do_return<mpl::if_<l, A, B>>
  >
));
```

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  try_c<
    mpl::if_<less<A, B>, A, B>






  >
));
```

```
MPLLIBS_LAZY_METAFUNCTION(min, (A)(B)    )
((
  do_<exception_tag,
    set<l, less<A, B>>,


    do_return<mpl::if_<l, A, B>>
  >
));
```

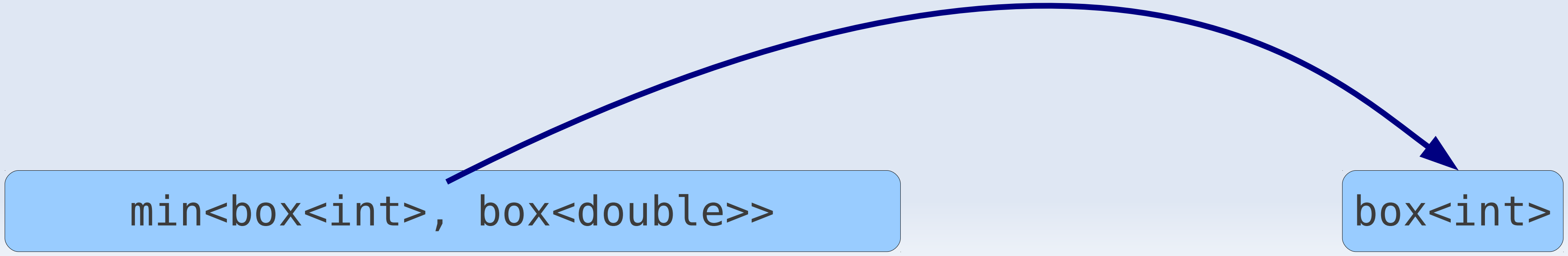
Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  try_c<
    mpl::if_<less<A, B>, A, B>




  >
));
```

$$\begin{tikzcd}
\boxed{\texttt{min<box<int>, box<double>>}} \arrow[r, bend left] & \boxed{\texttt{box<int>}}
\end{tikzcd}$$

Min function

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  try_c<
    mpl::if_<less<A, B>, A, B>,

    catch_c<e, boost::is_same<e, values_can_not_be_compared>,
      A
    >

  >
));
```



Min function

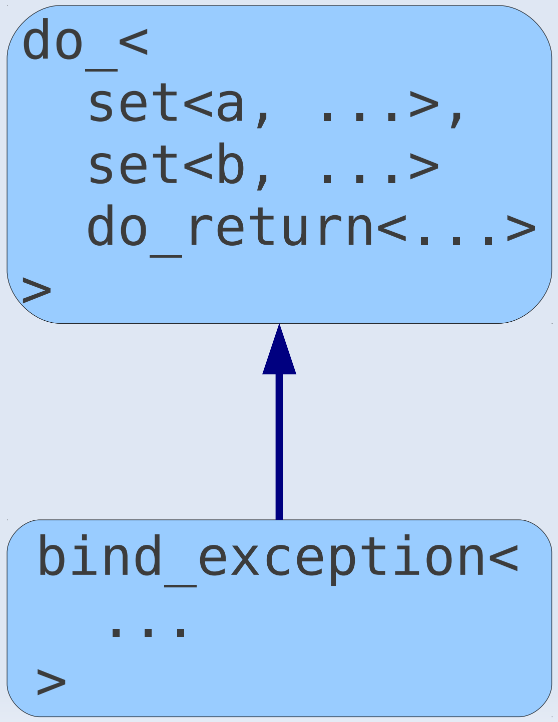
```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  try_c<
    mpl::if_<less<A, B>, A, B>,

    catch_c<e, boost::is_same<e, values_can_not_be_compared>,
      A
    >,
    catch_c<e, mpl::true_, B>
  >
));
```

$$\begin{tikzcd}\texttt{min<box<int>, box<double>>} \arrow[r, bend left] & \texttt{box<int>}\end{tikzcd}$$

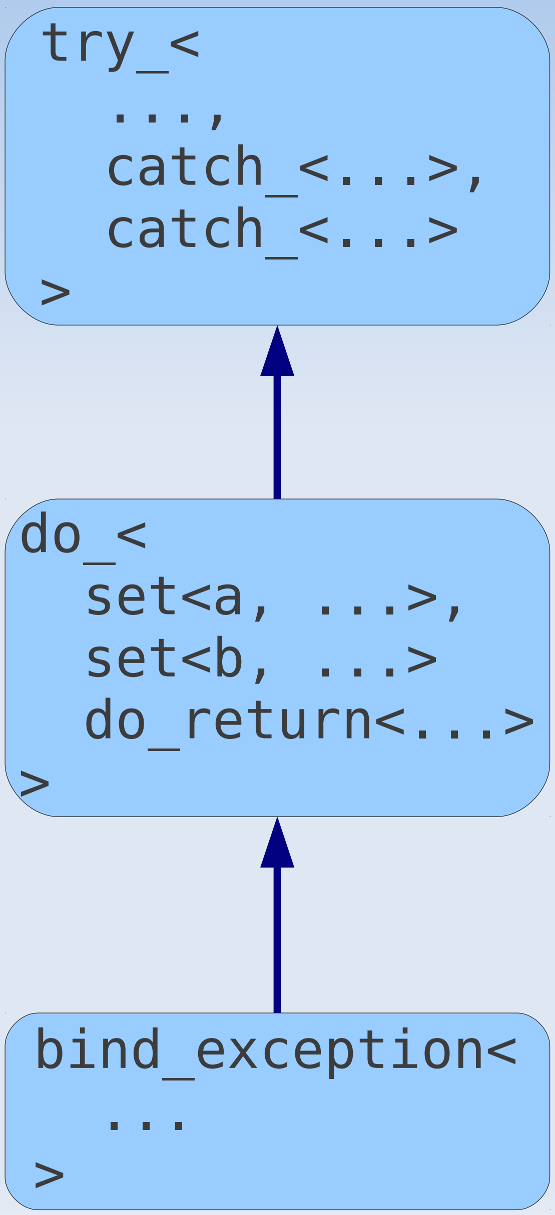
Generalisation

```
bind_exception<
  ...
>
```

Generalisation



Generalisation

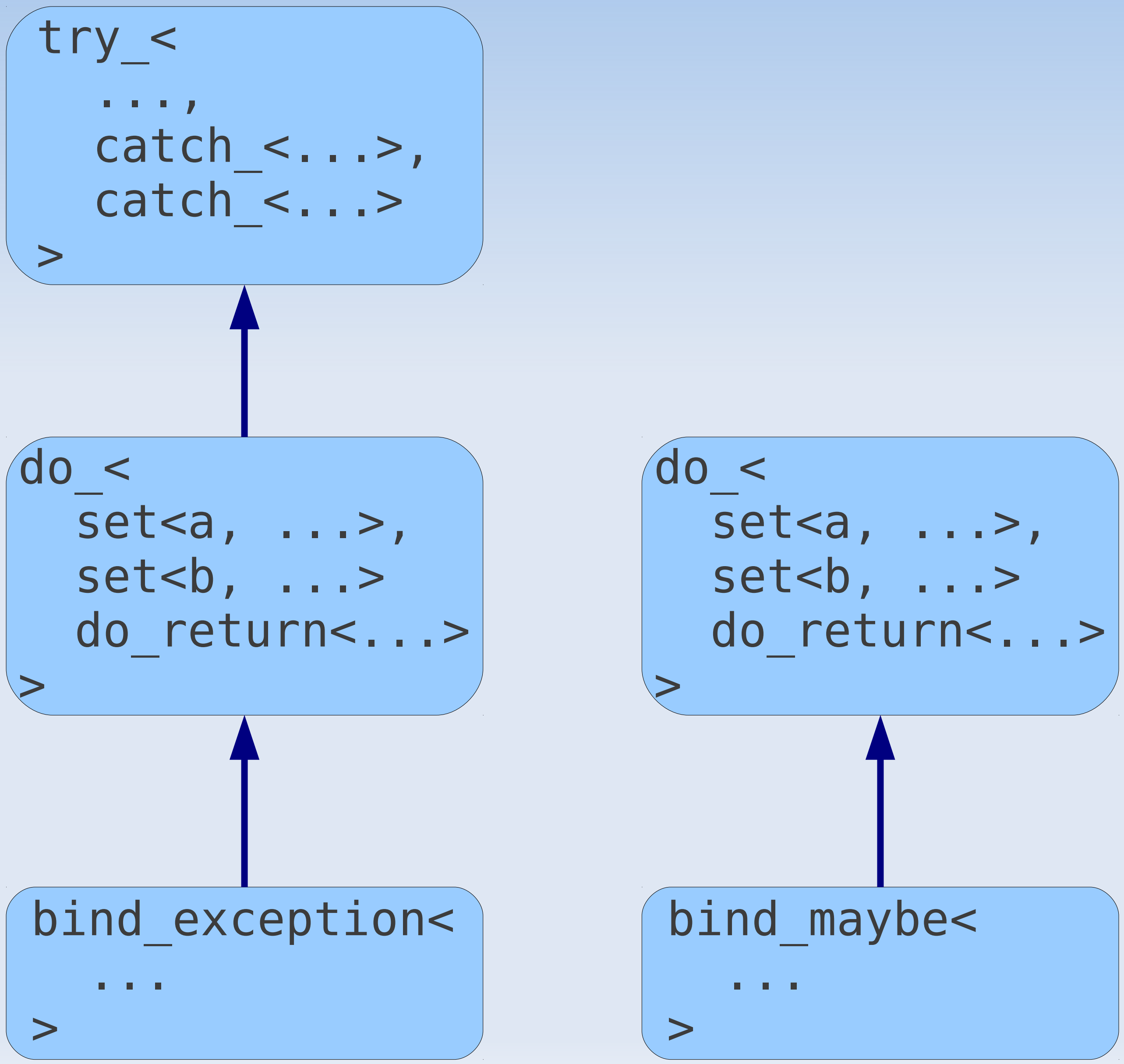


```
try_<
  ...,
  catch_<...>,
  catch_<...>
>
```

```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
bind_exception<
  ...
>
```

Generalisation



```
try_<
  ...,
  catch_<...>,
  catch_<...>
>
```

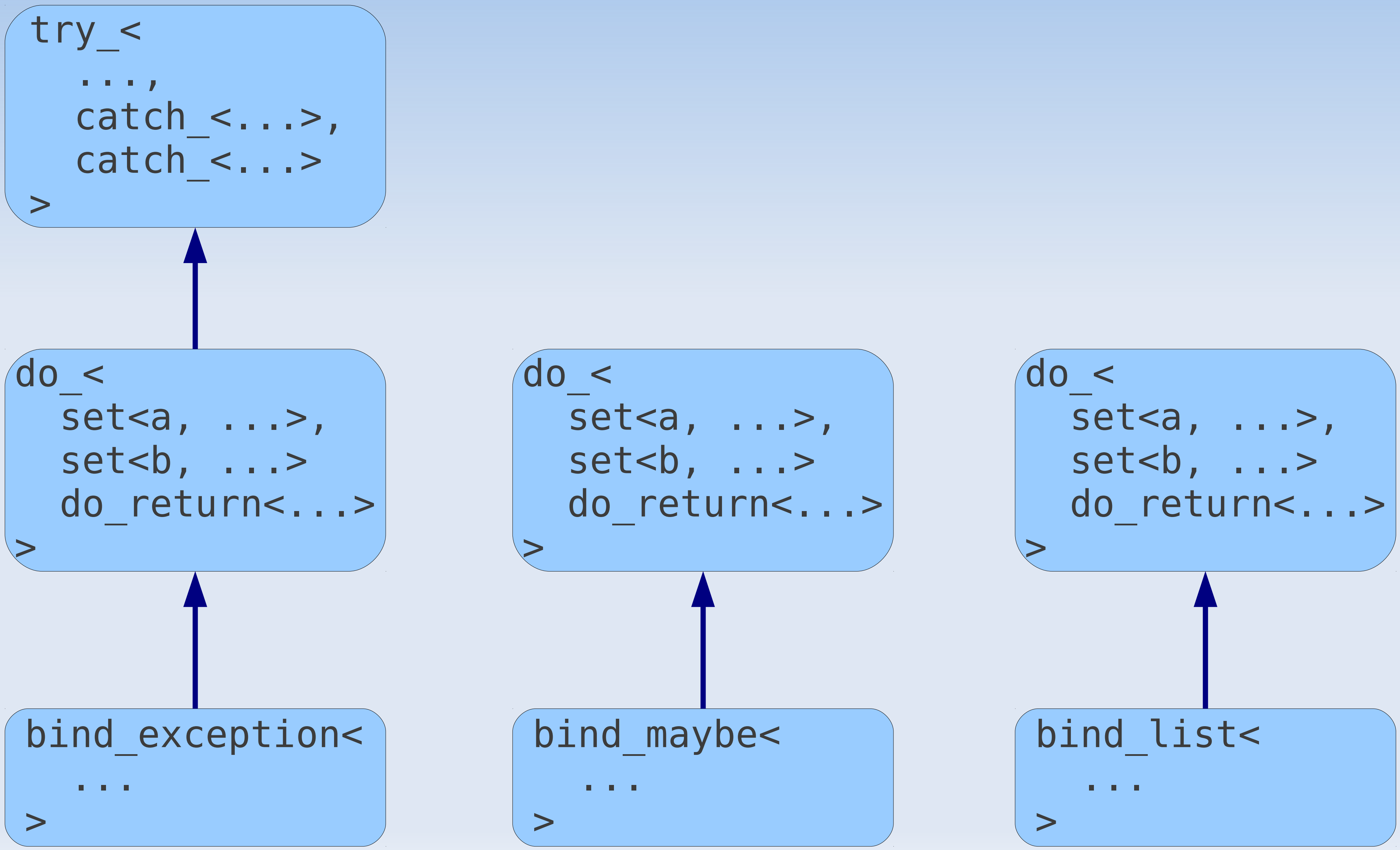
```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
bind_exception<
  ...
>
```

```
bind_maybe<
  ...
>
```

Generalisation



```
try_<
  ...,
  catch_<...>,
  catch_<...>
>
```

```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
do_<
  set<a, ...>,
  set<b, ...>
  do_return<...>
>
```

```
bind_exception<
  ...
>
```

```
bind_maybe<
  ...
>
```

```
bind_list<
  ...
>
```

Generalisation

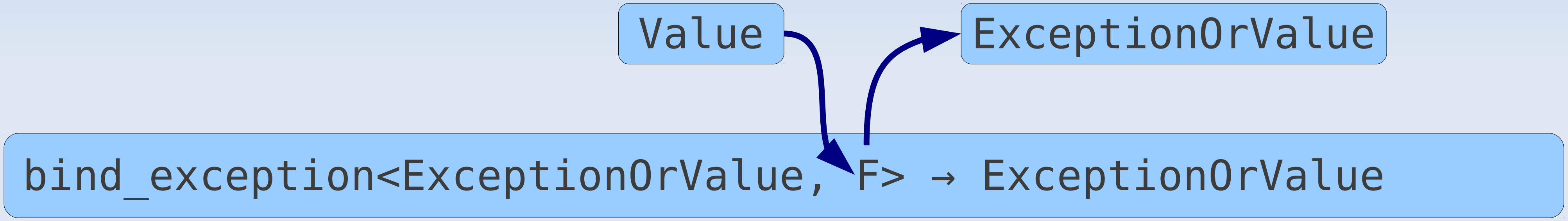
```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Generalisation

```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Generalisation



```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Generalisation

- `bind<SetOfValues, F> → SetOfValues`

```
Value            ExceptionOrValue
```

```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
MPLLIBS_METAFUNCTION(min, (A)(B))
((
  bind_exception<
    less<A, B>,
    lambda_c<l, mpl::if_<l, A, B>>
  >
));
```

Generalisation

- `bind<SetOfValues, F> → SetOfValues`

```
Value            ExceptionOrValue
```

```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
bind_maybe<NothingOrJust, F> → NothingOrJust
```

Generalisation

- `bind<SetOfValues, F> → SetOfValues`

$$\begin{tikzcd} \text{Value} \arrow[r] & \text{F} \arrow[r] & \text{ExceptionOrValue} \end{tikzcd}$$

```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
bind_maybe<NothingOrJust, F> → NothingOrJust
```

```
bind_maybe<nothing, F> → nothing
bind_maybe<just<x>, F> → F<x>
```

Generalisation

- `bind<SetOfValues, F> → SetOfValues`
- `return_<Value> → SetOfValues`

$$\begin{tikzcd} \texttt{Value} \arrow[r] & \texttt{F} \arrow[r] & \texttt{ExceptionOrValue} \end{tikzcd}$$

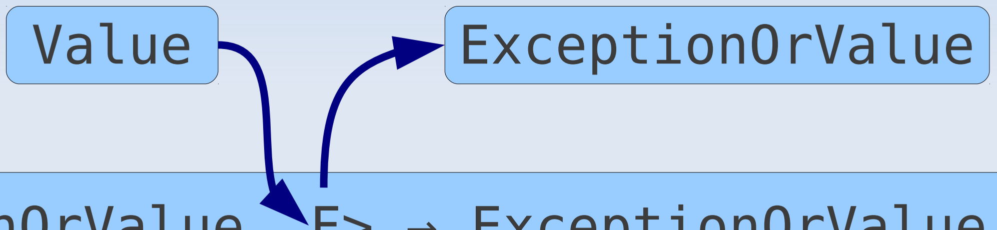
```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
bind_maybe<NothingOrJust, F> → NothingOrJust
```

```
bind_maybe<nothing, F> → nothing
bind_maybe<just<x>, F> → F<x>
```

Generalisation

- `bind<SetOfValues, F> → SetOfValues`
- `return_<Value> → SetOfValues`



```
bind_exception<ExceptionOrValue, F> → ExceptionOrValue
```

```
return_exception<Value> → ExceptionOrValue
```

```
bind_maybe<NothingOrJust, F> → NothingOrJust
```

```
return_maybe<Value> → NothingOrJust
```

Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

```
f: Value → List
```

$$\begin{tikzcd}
\texttt{int\_<1>} \arrow[d, bend left, "f"] \\
\boxed{\texttt{list<int\_<11>, int\_<12>>}}
\end{tikzcd}$$

Lists

`f: Value → List`

- Set of values: lists
  - `return_: Value → [Value]`
  - `bind: <List, F> → List`

$$\begin{tikzcd}
\texttt{int\_<1>} \arrow[d, "f"'] & \texttt{int\_<2>} \arrow[d, "f"] \\
\boxed{\texttt{list<int\_<11>, int\_<12>>}} & \boxed{\texttt{list<int\_<13>>}}
\end{tikzcd}$$

Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

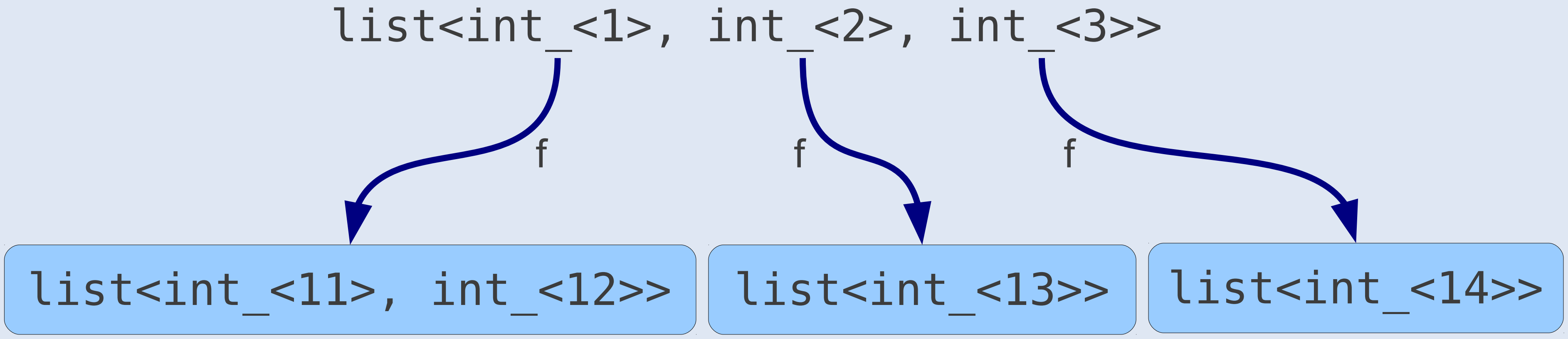
`f: Value → List`

$$\begin{tikzcd}
\texttt{int\_<1>} \arrow[d, bend right, "f"] & \texttt{int\_<2>} \arrow[d, bend left, "f"'] & \texttt{int\_<3>} \arrow[d, bend right, "f"'] \\
\boxed{\texttt{list<int\_<11>, int\_<12>>}} & \boxed{\texttt{list<int\_<13>>}} & \boxed{\texttt{list<int\_<14>>}}
\end{tikzcd}$$

Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

`f: Value → List`



Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

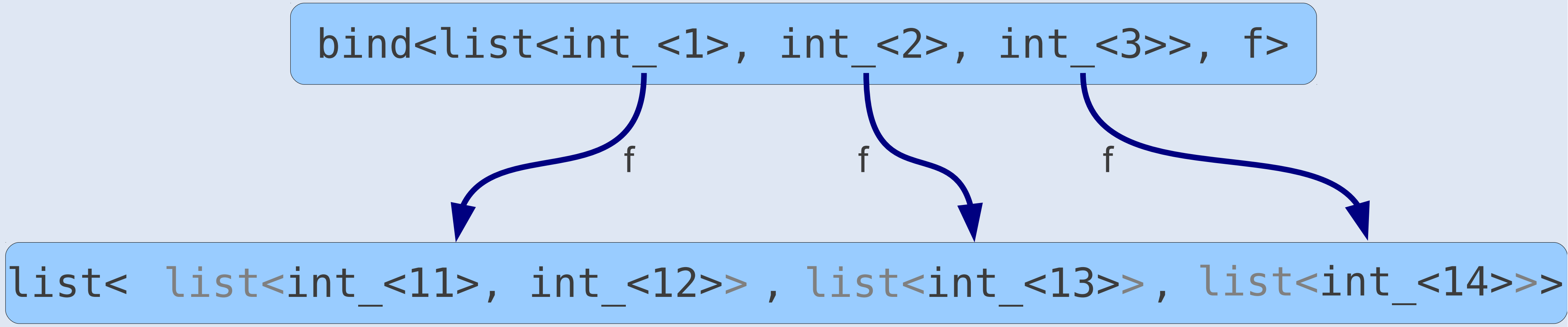
`f: Value → List`

$$\begin{tikzcd}
 & \texttt{bind<list<int\_<1>, int\_<2>, int\_<3>>, f>} \arrow[dl, "f"'] \arrow[d, "f"] \arrow[dr, "f"] & \\
\texttt{list<int\_<11>, int\_<12>>} & \texttt{list<int\_<13>>} & \texttt{list<int\_<14>>}
\end{tikzcd}$$

Lists

- Set of values: lists
- `return_: Value → [Value]`
- `bind: <List, F> → List`

```
f: Value → List
```



```
bind<list<int_<1>, int_<2>, int_<3>>, f>
```

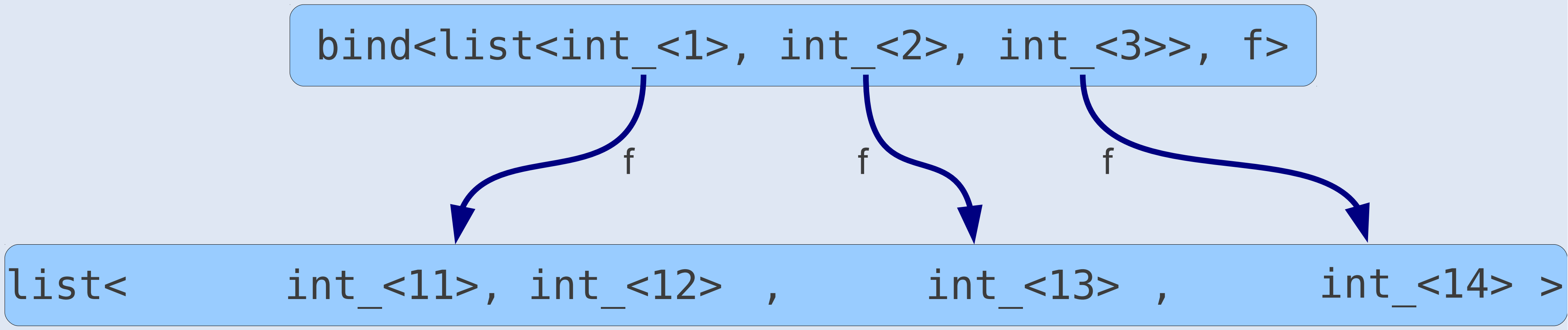
```
list< list<int_<11>, int_<12>> , list<int_<13>> , list<int_<14>>>
```

Lists

- Set of values: lists

`f: Value → List`

- `return_: Value → [Value]`
- `bind: <List, F> → List`



```
bind<list<int_<1>, int_<2>, int_<3>>, f>
```

```
list<        int_<11>, int_<12>  ,        int_<13>  ,        int_<14> >
```

List comprehension

- Get all relative primes in [1..100)

List comprehension

- Get all relative primes in [1..100)

```
do_c<list_tag,



  do_return<pair<i, j>>
>
```

```
[(i, j) |                                                    ]
```

List comprehension

- Get all relative primes in [1..100)

```
typedef mpl::range_c<int, 1, 100>> range_1_100;
```

```
do_c<list_tag,
  set<i, range_1_100>,


  do_return<pair<i, j>>
>
```

```
[(i, j) | i ← [1..100]                                       ]
```

List comprehension

- Get all relative primes in [1..100)

```
typedef mpl::range_c<int, 1, 100>> range_1_100;
```

```
do_c<list_tag,
  set<i, range_1_100>,
  set<j, range_1_100>,

  do_return<pair<i, j>>
>
```

```
[(i, j) | i ← [1..100], j ← [1..100]                          ]
```

List comprehension

- Get all relative primes in [1..100)

```
typedef mpl::range_c<int, 1, 100>> range_1_100;
```

```
do_c<list_tag,
  set<i, range_1_100>,
  set<j, range_1_100>,
  guard<relative_prime<i, j>>,
  do_return<pair<i, j>>
>
```

```
[(i, j) | i ← [1..100], j ← [1..100], relative_prime(i, j)]
```

List comprehension

- Get all relative primes in [1..100)

```
typedef mpl::range_c<int, 1, 100>> range_1_100;
```

```
do_c<list_tag,
  set<i, range_1_100>,
  set<j, range_1_100>,
  guard<relative_prime<i, j>>,
  do_return<pair<i, j>>
>
```

```
for i in 1..100:
  for j in 1..100:
    if relative_prime<i, j>:
      pair<i, j>
```

```
[(i, j) | i ← [1..100], j ← [1..100], relative_prime(i, j)]
```

Other possibilities

- What can also be done (and is provided):
  - Either
  - Exception
  - List
  - Maybe
  - Reader
  - State
  - Writer

Summary

- Laziness
- Syntaxes
- Algebraic data-types
- Exceptions
- Generalisation of `bind`

Fact

```
template <class N>
struct fact;

template <class N>
struct fact_impl :
  times<
    N,
    typename fact<typename minus<N, int_<1>>::type>::type
  >
{};

template <class N>
struct fact :
  eval_if<
    typename equal_to<N, int_<1>>::type,
    int_<1>,
    fact_impl<N>
  >
{};
```

Fact

```
template <class N>
struct fact;

template <class N>
st
```

```
MPLLIBS_METAFUNCTION(fact, (N))
((
  eval_case< N,
    matches_c<int_<0>, int_<1>>,
    matches_c<_,       times<N, fact<minus<N, int_<1>>>>
  >
));
```

```
struct fact :
  eval_if<
    typename equal_to<N, int_<1>>::type,
    int_<1>,
    fact_impl<N>
  >
{};
```

Q & A

Mpllibs.Metamonad

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