Metaparse

Mpllibs

- Template Metaprogramming libraries
- http://abel.web.elte.hu/mpllibs
    - Metaparse
    - Metamonad
    - Metatest
    - Safe Printf

Mpllibs

- Ábel Sinkovics
- Endre Sajó
- Zoltán Porkoláb

Agenda

- Parsing at compile-time
- Metaparse
- Haskell-like DSL for template metaprogramming
- Advanced parser creation techniques

f

```
template <class N> struct f_impl :
  boost::mpl::plus<
    typename f<
      typename boost::mpl::minus<
        N, boost::mpl::int_<1>>::type
    >::type,
    typename f<
      typename boost::mpl::minus<
        N, boost::mpl::int_<2>>::type
    >::type
  > {};

template <class N> struct f : boost::mpl::eval_if<
    typename boost::mpl::less<
      N, boost::mpl::int_<2>>::type,
    f_impl<N>,
    boost::mpl::int_<1>
  > {};
```

f

```
template <class N> struct f_impl :
  boost::mpl::plus<
    typename f<
      typename boost::mpl::minus<
        N, boost::mpl::int_<1>>::type
    >::type,
    typename
      typena
        N, b
    >::type
  > {};

template <class N> struct f : boost::mpl::eval_if<
    typename boost::mpl::less<
      N, boost::mpl::int_<2>>::type,
    f_impl<N>,
    boost::mpl::int_<1>
  > {};
```

```
f n =
  if n < 2
    then f (n – 1)  +  f (n – 2)
    else 1
```

f

```
template <class N> struct fib_impl :
  boost::mpl::plus<
    typename fib<
      typename boost::mpl::minus<
        N, boost::mpl::int_<1>>::type
    >::type,
    typename fib<
      typename boost::mpl::minus<
        N, boost::mpl::int_<2>>::type
    >::type
  > {};

template <class N> struct fib : boost::mpl::eval_if<
    typename boost::mpl::less<
      N, boost::mpl::int_<2>>::type,
    fib_impl<N>,
    boost::mpl::int_<1>
  > {};
```

f

```
template <class N> struct fib_impl :
  boost::mpl::plus<
    typename fib<
      typename boost::mpl::minus<
        N, boost::mpl::int_<1>>::type
    >::type,
    typename
      typena
        N, b
    >::type
  > {};
```

```
fib n =
  if n < 2
    then fib (n – 1)  +  fib (n – 2)
    else 1
```

```
template <class N> struct fib : boost::mpl::eval_if<
    typename boost::mpl::less<
      N, boost::mpl::int_<2>>::type,
    fib_impl<N>,
    boost::mpl::int_<1>
  > {};
```

f

```
typedef
  meta_hs
    ::define<_S(
      "fib n = "
        "if n < 2 "
          "then 1 "
          "else fib (n – 1) + fib (n – 2)"
      )>::type
    ::get<_S("fib")>::type
  fib;
```

Xpressive

```
sregex re = sregex::compile("x[ab]");

  // No static verification
```

Xpressive

```
sregex re = sregex::compile("x[ab]");

  // No static verification
```

```
sregex re = 'x' >> (as_xpr('a') | 'b');

  // One has to learn the
  //   "regular expression" → Xpressive expression
  // mapping
```

Xpressive

```
sregex re = sregex::compile("x[ab]");

  // No static verification
```

```
sregex re = 'x' >> (as_xpr('a') | 'b');

  // One has to learn the
  //   "regular expression" → Xpressive expression
  // mapping
```

```
sregex re = REGEXP("x[ab]");
```

Spirit

```
double rN = 0.0, rI = 0.0;

(
     '(' >> double_[ref(rN) = _1]
         >> -(',' >> double_[ref(iN) = _1]) >> ')'
   | double_[ref(rN) = _1]
)
```

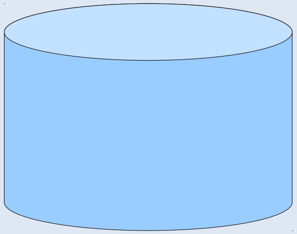
Spirit

```
double rN = 0.0, rI = 0.0;

(
     '(' >> double_[ref(rN) = _1]
         >> -(',' >> double_[ref(iN) = _1]) >> ')'
   | double_[ref(rN) = _1]
)
```

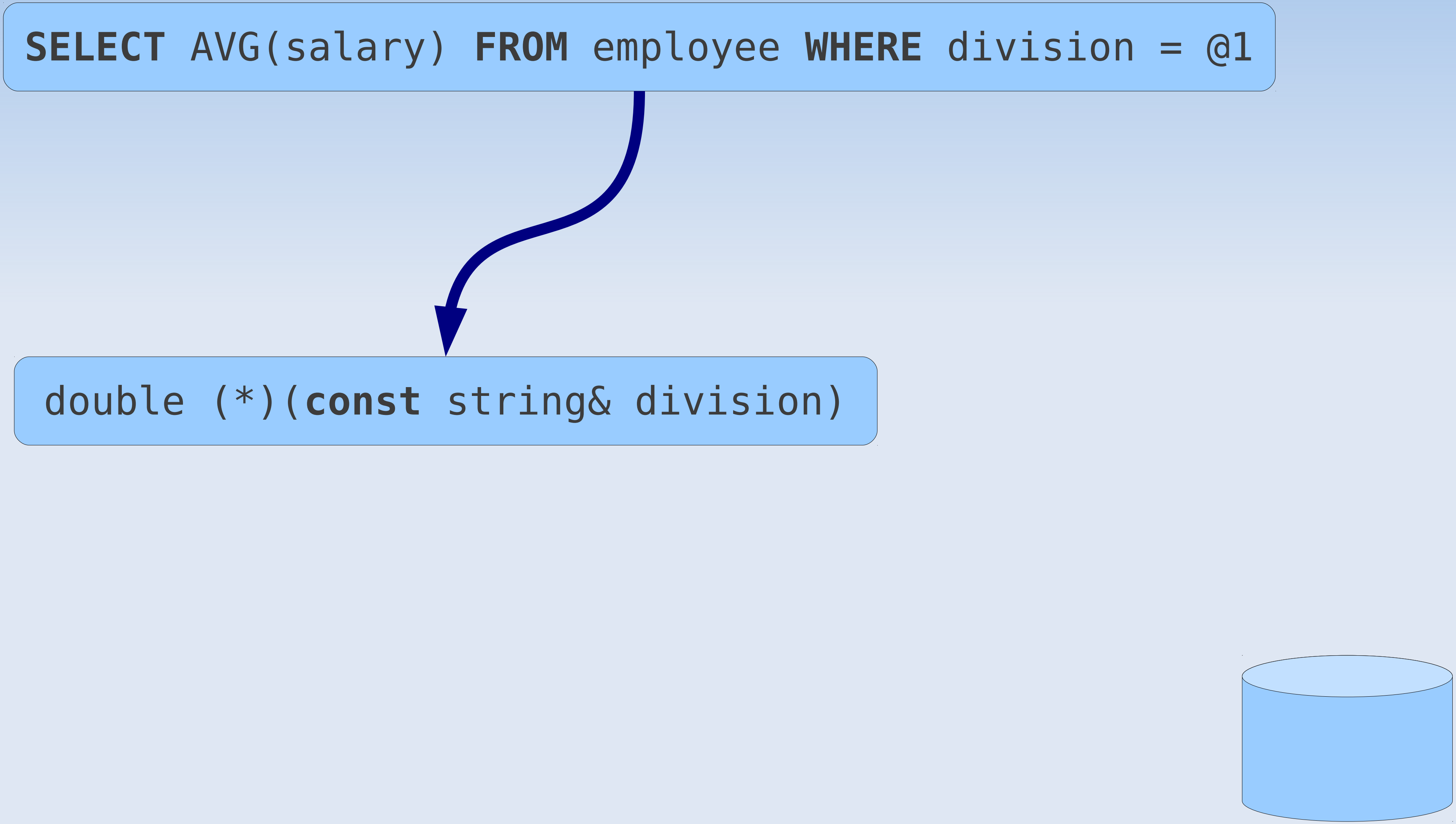
```
grammar<"COMPLEX">
  ::RULE("COMPLEX ::= CMP | REAL"              )
  ::RULE("CMP     ::= '(' REAL (',' IMAG)? ')'")
  ::RULE("REAL    ::= DOUBLE"                  )
  ::RULE("IMAG    ::= DOUBLE"                  )
::build()
  .ACTION("REAL")[ref(rN) = _1]
  .ACTION("IMAG")[ref(iN) = _1]
.done()
```

SQL

```sql
SELECT AVG(salary) FROM employee WHERE division = @1
```



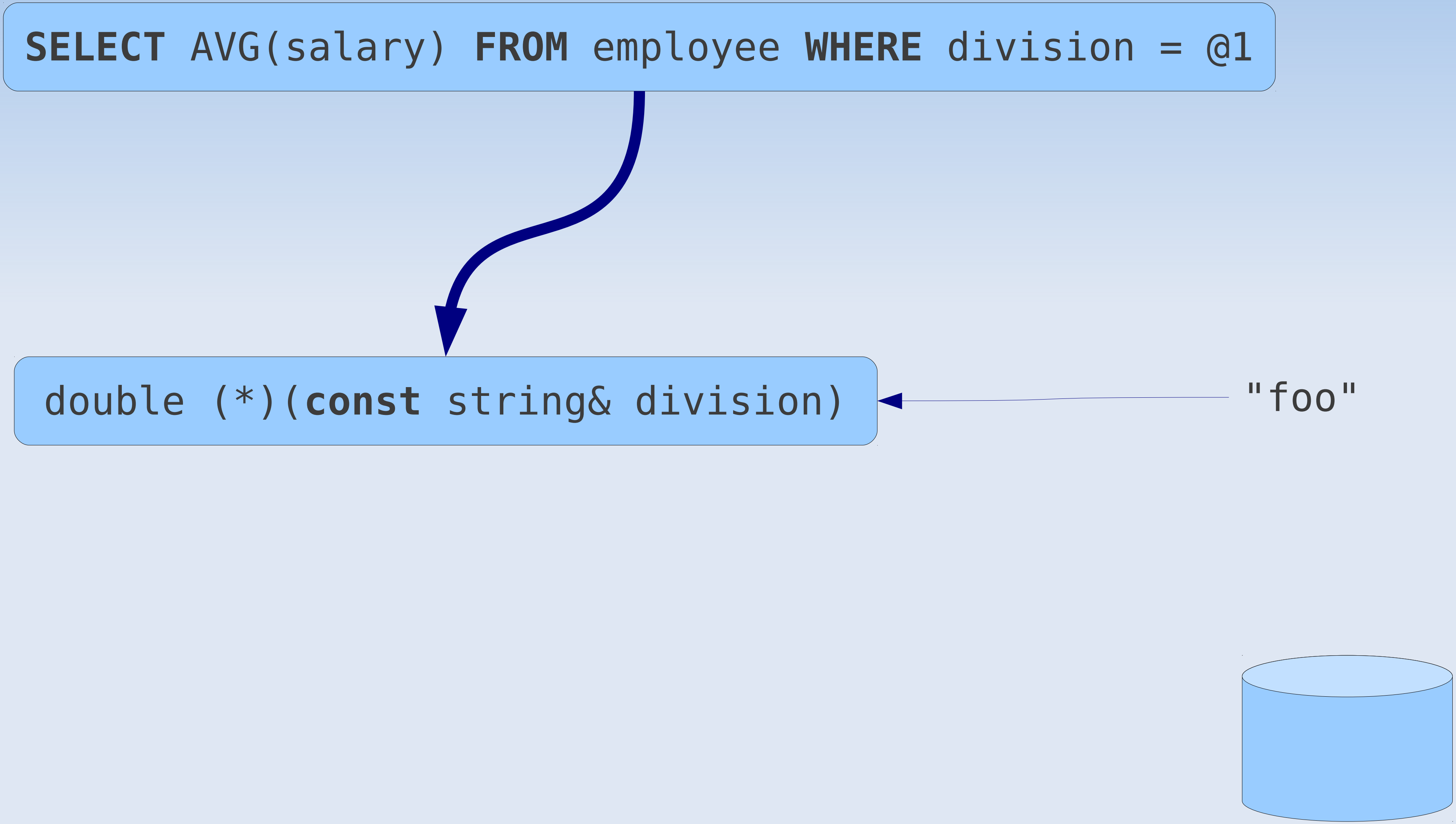
SQL



```
SELECT AVG(salary) FROM employee WHERE division = @1
```

```
double (*)(const string& division)
```

SQL

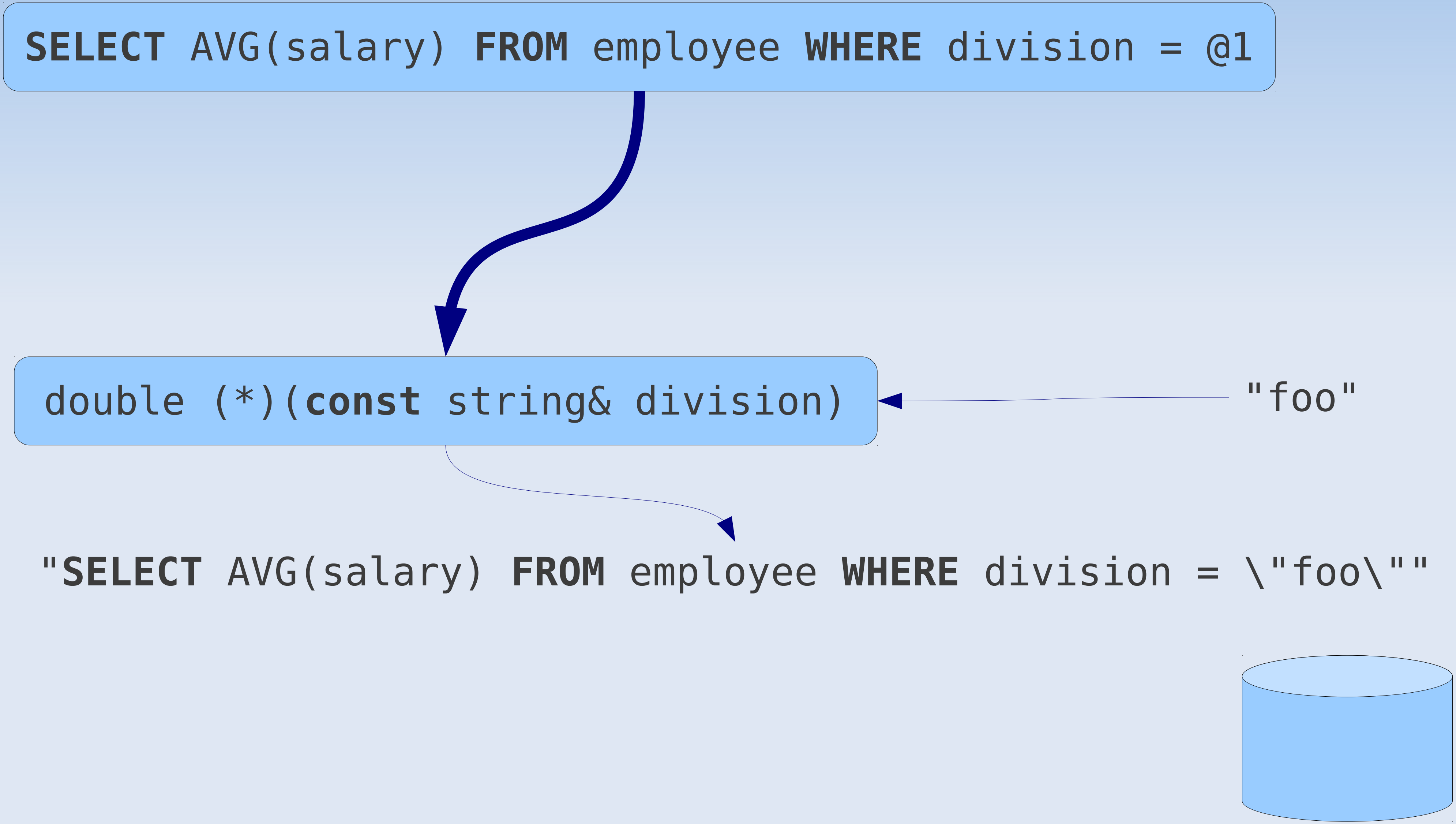


```
SELECT AVG(salary) FROM employee WHERE division = @1
```

```
double (*)(const string& division)
```

```
"foo"
```

SQL



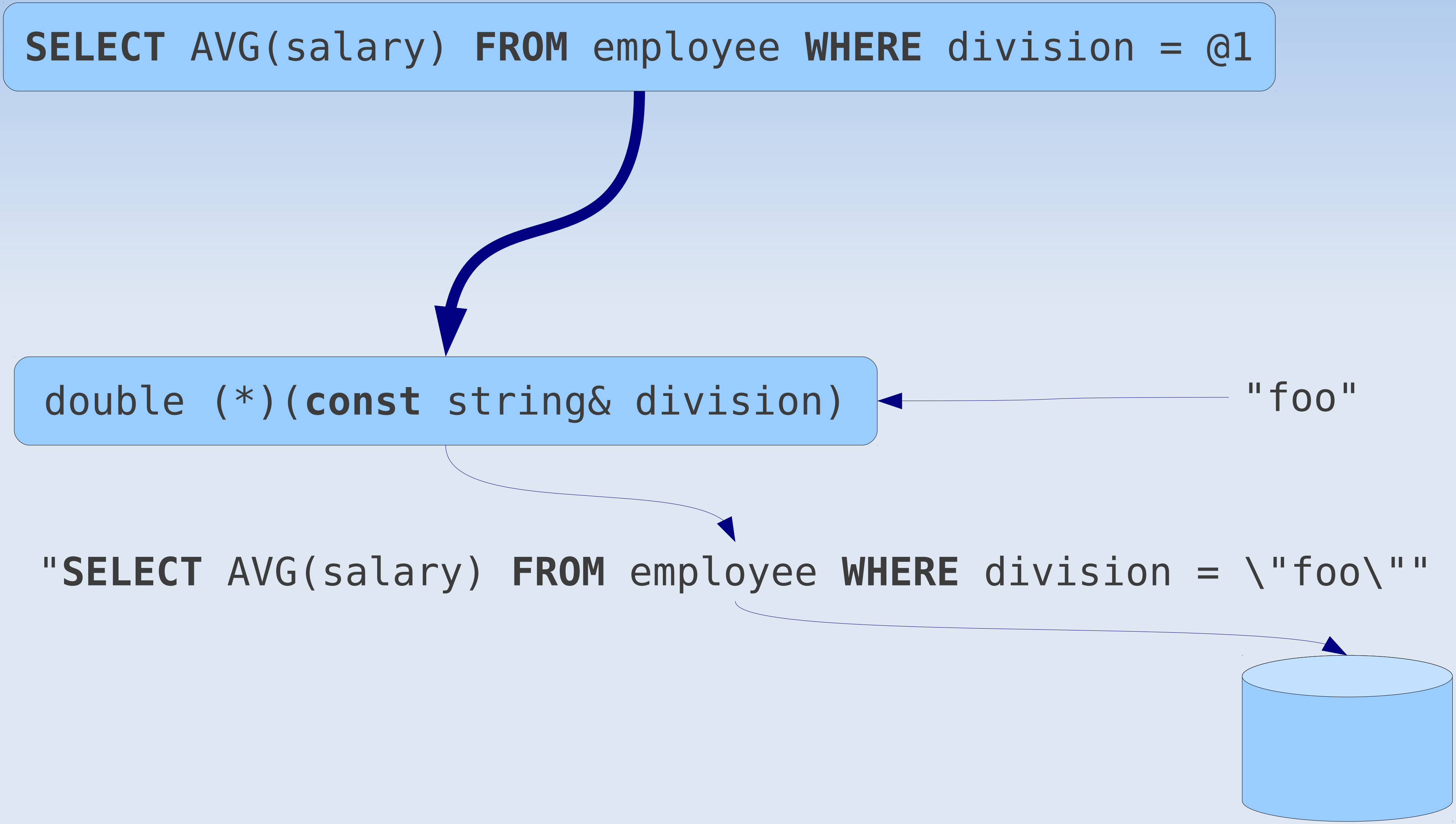
```
SELECT AVG(salary) FROM employee WHERE division = @1
```

```
double (*)(const string& division)
```

"foo"

```
"SELECT AVG(salary) FROM employee WHERE division = \"foo\""
```

SQL



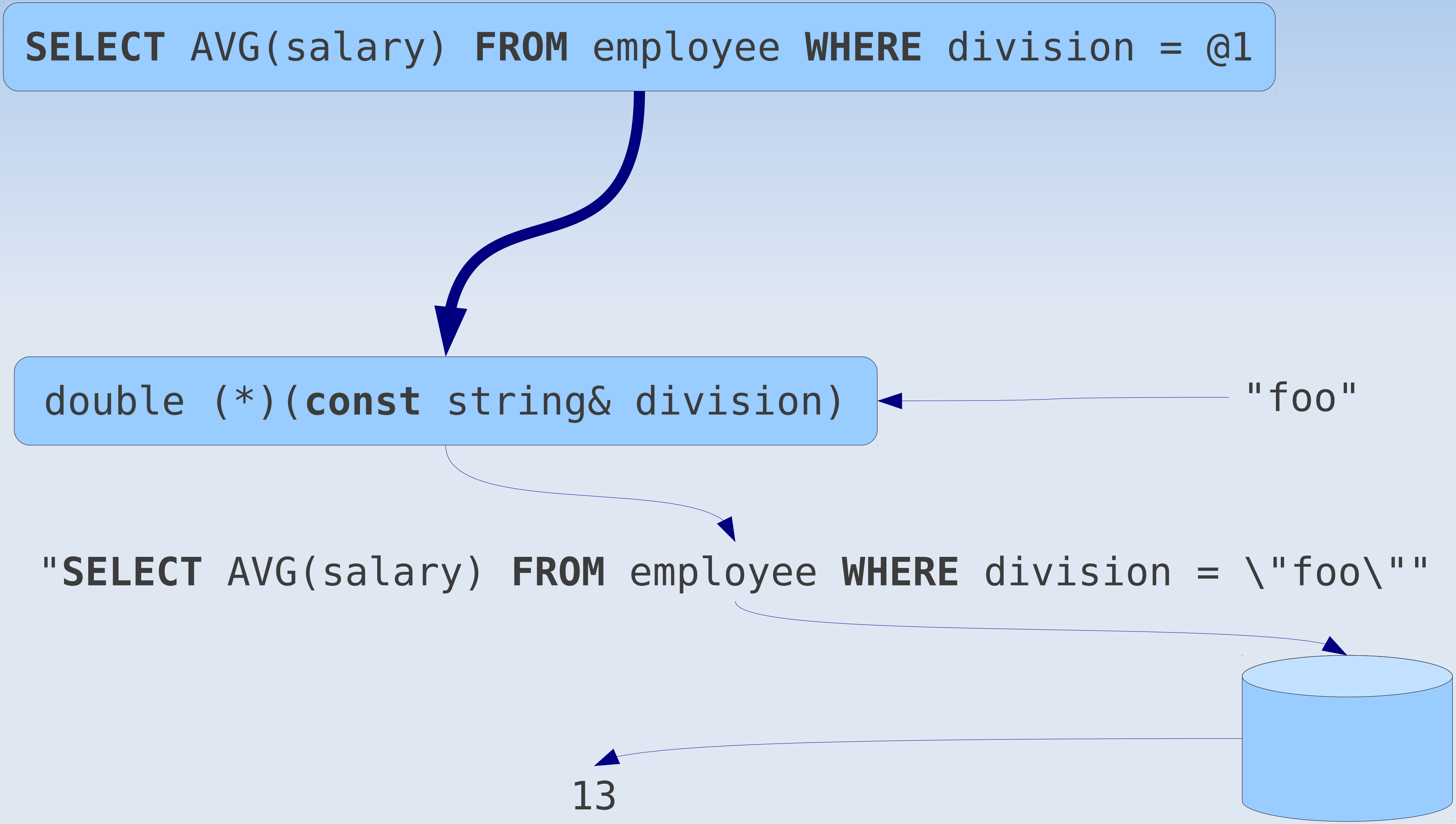
```
SELECT AVG(salary) FROM employee WHERE division = @1
```

```
double (*)(const string& division)
```

```
"foo"
```

```
"SELECT AVG(salary) FROM employee WHERE division = \"foo\""
```

SQL



```
SELECT AVG(salary) FROM employee WHERE division = @1
```

```
double (*)(const string& division)
```

"foo"

```
"SELECT AVG(salary) FROM employee WHERE division = \"foo\""
```

13

Error handling

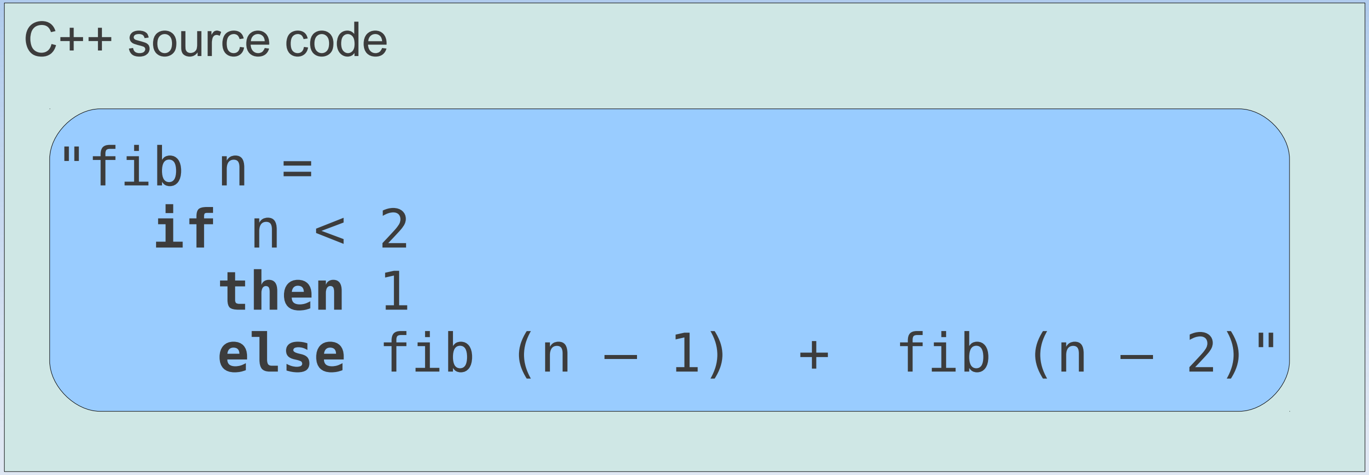
```
sregex re = REGEXP("xab]");
```

Error handling

```
sregex re = REGEXP("xab]");
```

```
line 1, col 4: ] without [
```

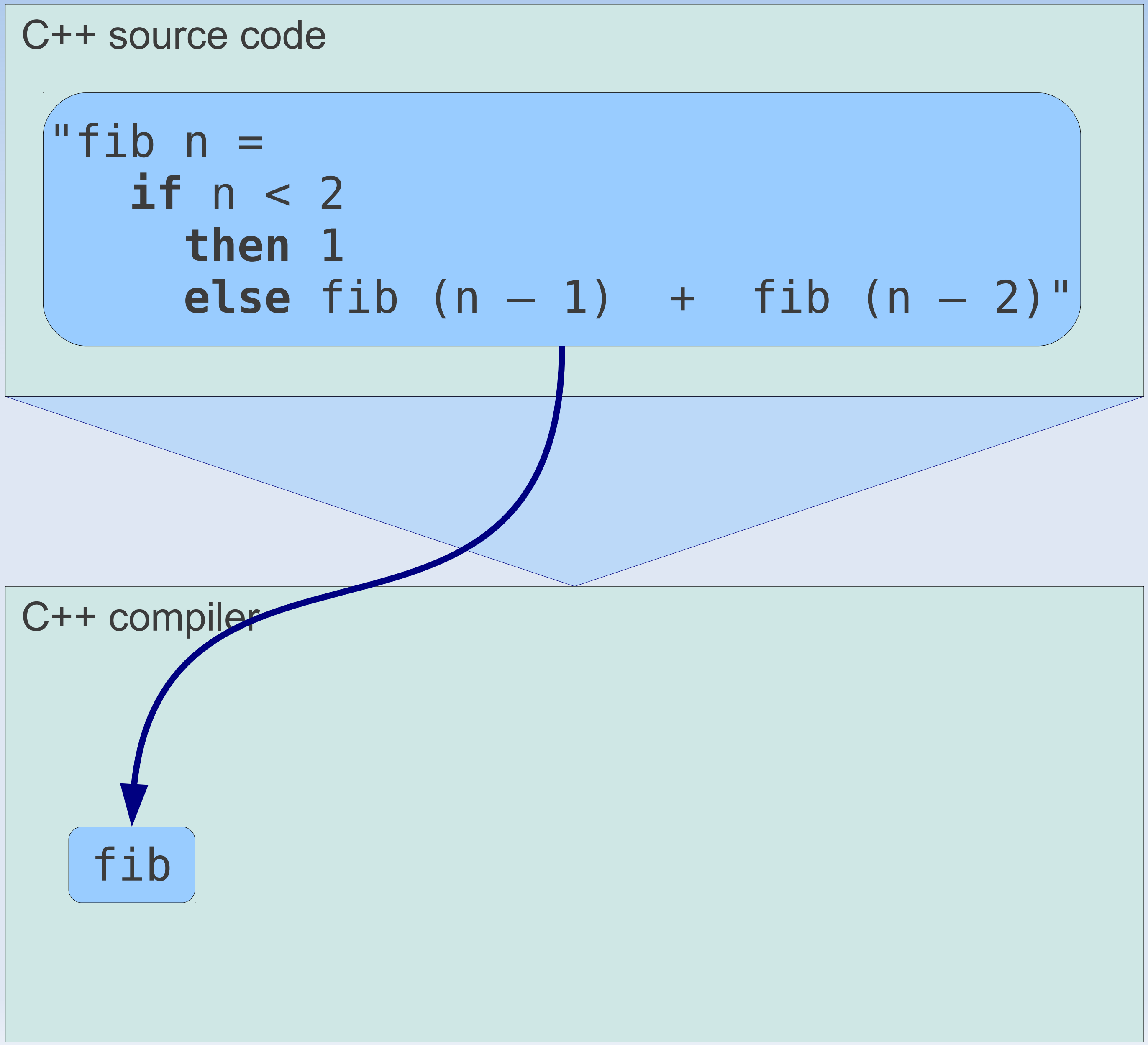
How it works?



C++ source code

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

How it works?

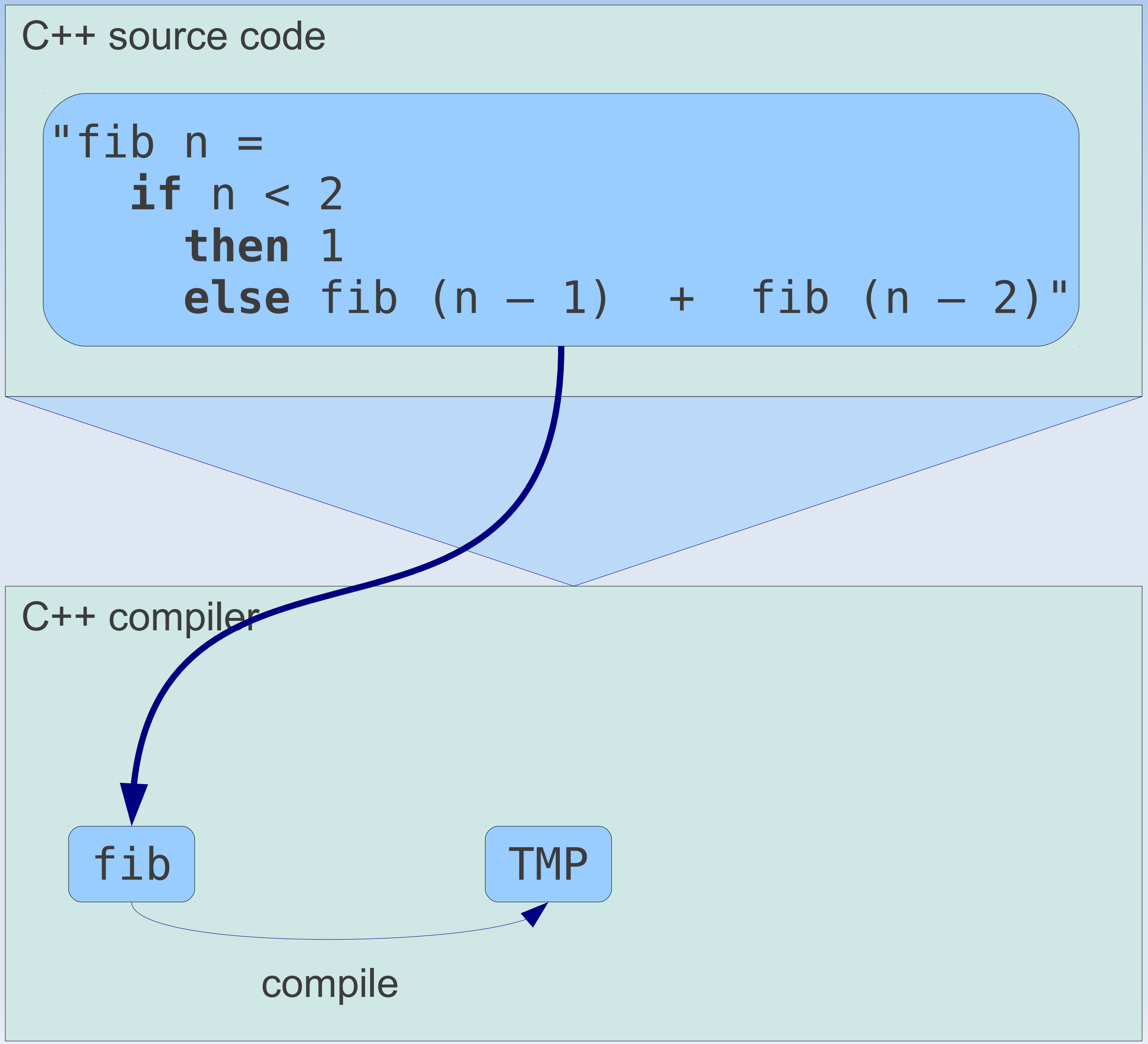


C++ source code

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

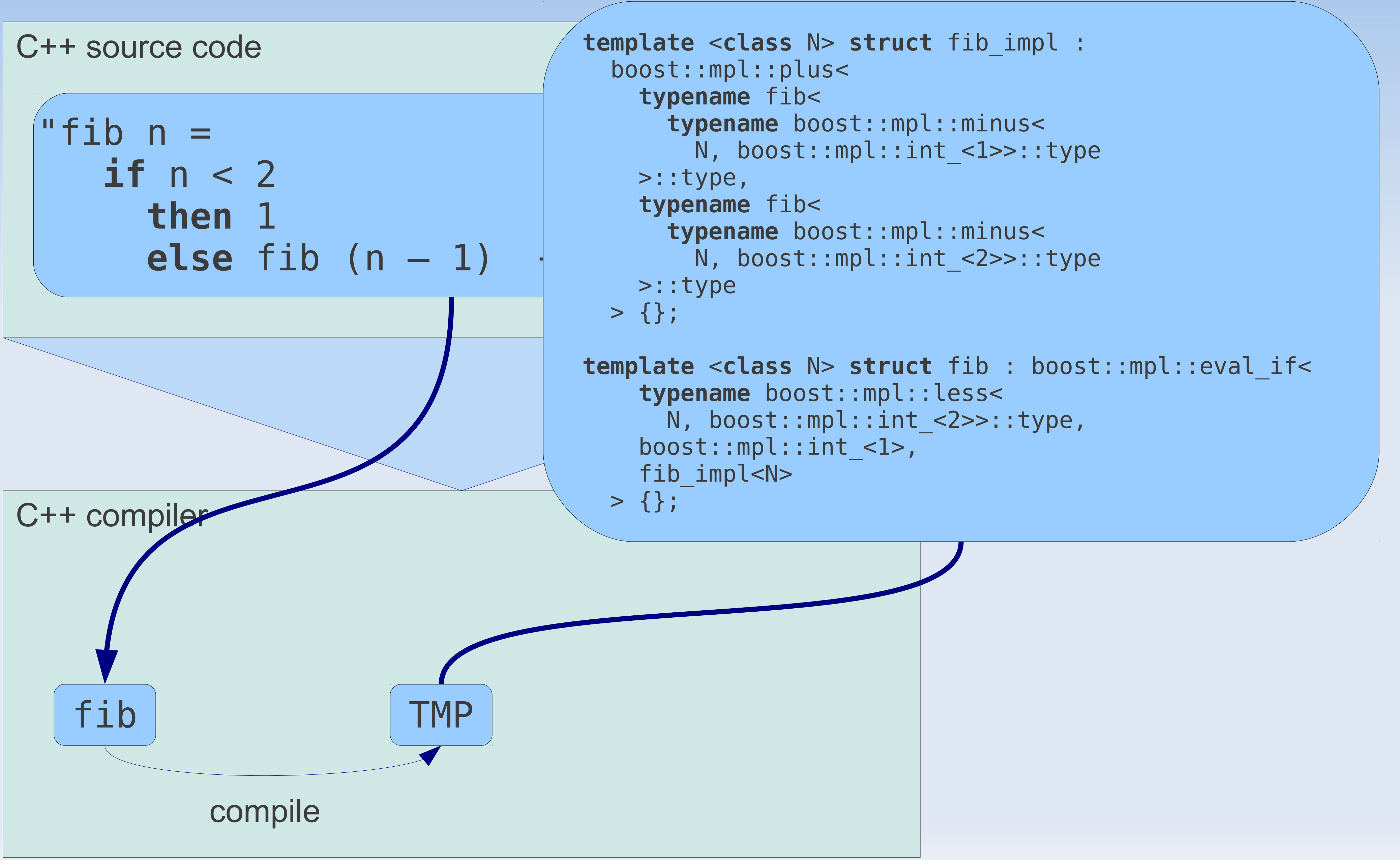
C++ compiler

How it works?



```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

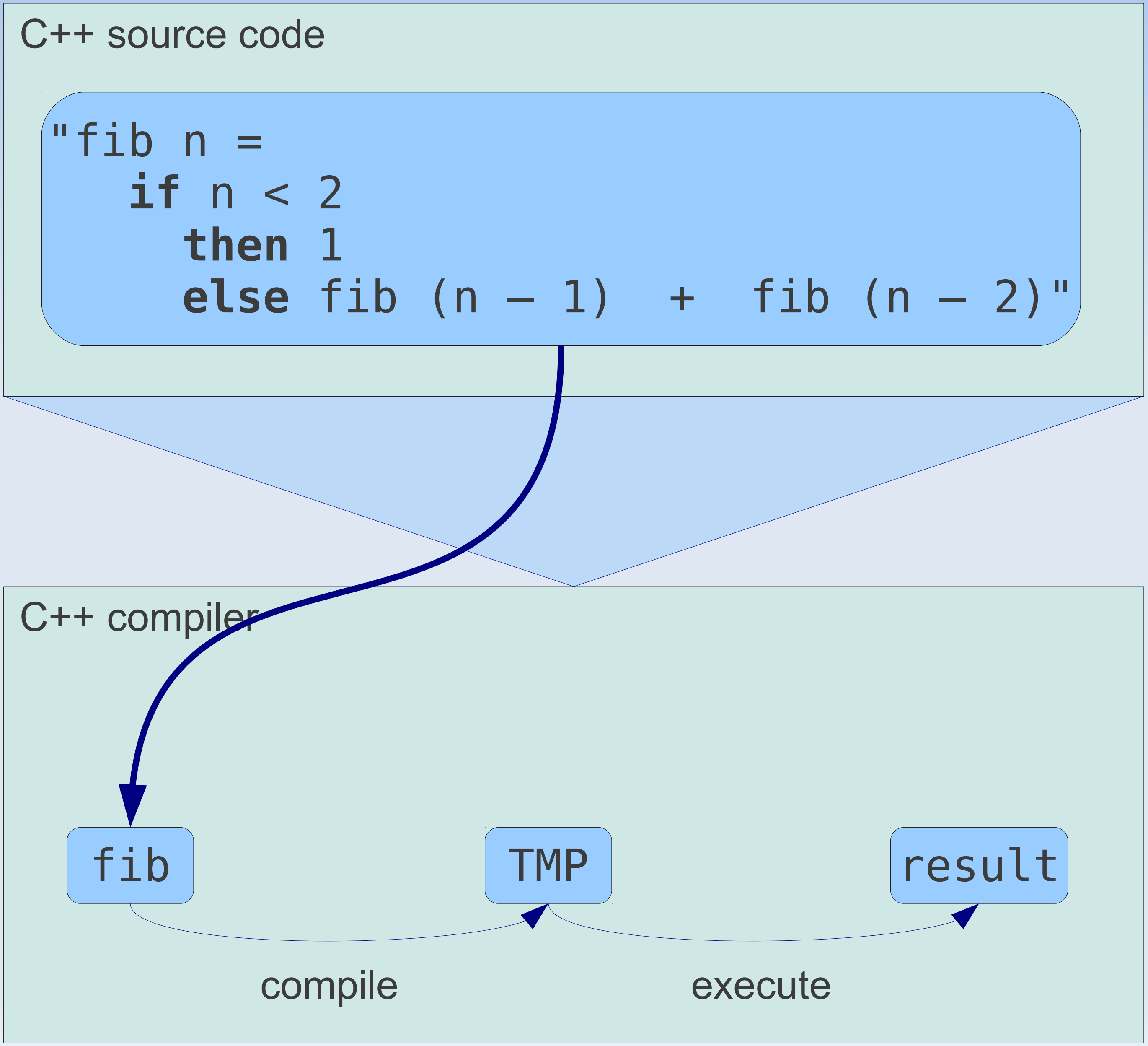
How it works?



```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)
```

```
template <class N> struct fib_impl :
  boost::mpl::plus<
    typename fib<
      typename boost::mpl::minus<
        N, boost::mpl::int_<1>>::type
    >::type,
    typename fib<
      typename boost::mpl::minus<
        N, boost::mpl::int_<2>>::type
    >::type
  > {};

template <class N> struct fib : boost::mpl::eval_if<
    typename boost::mpl::less<
      N, boost::mpl::int_<2>>::type,
    boost::mpl::int_<1>,
    fib_impl<N>
  > {};
```

How it works?



C++ source code

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

C++ compiler

fib

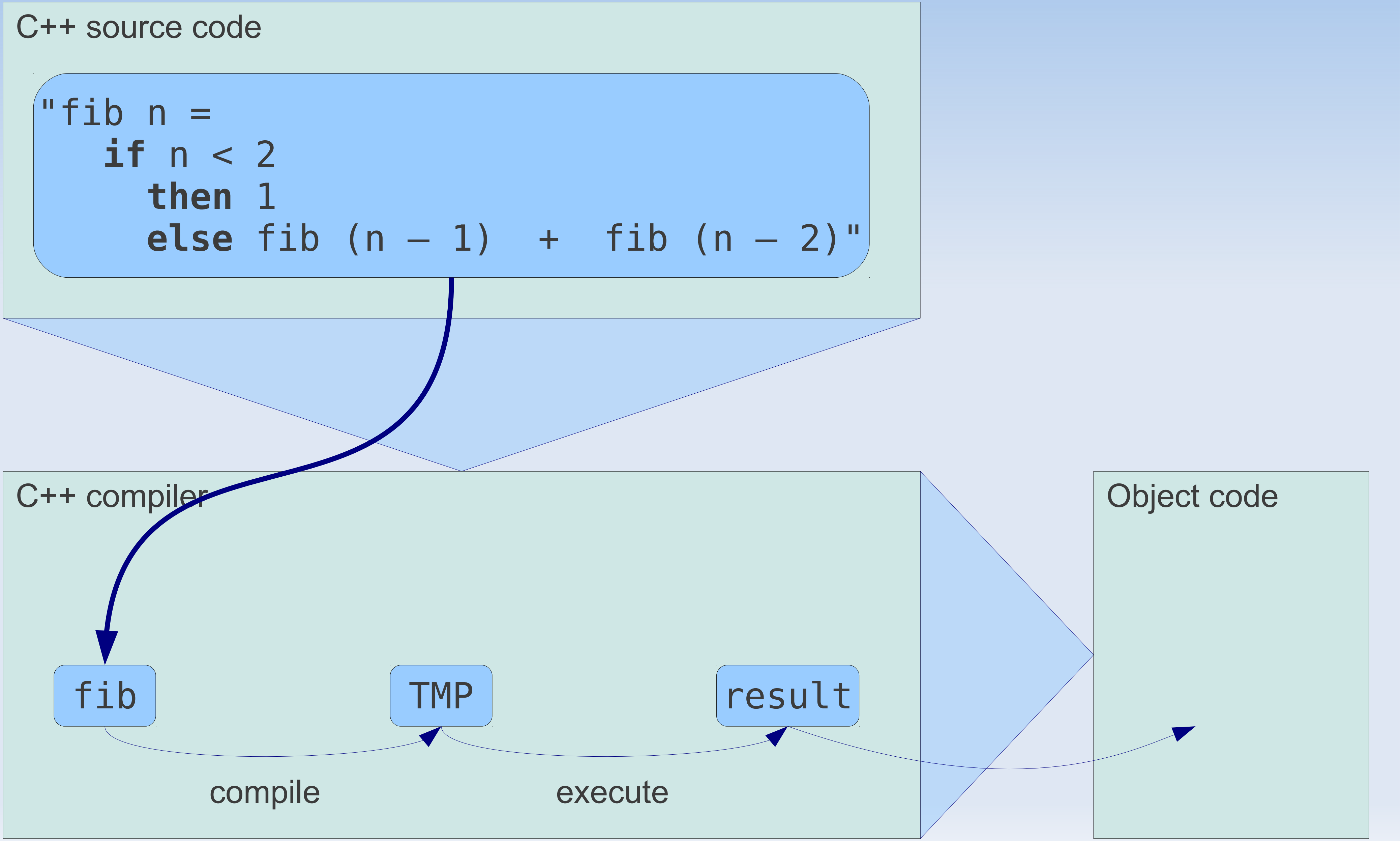
TMP

result

compile

execute

How it works?



C++ source code

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

C++ compiler

fib

TMP

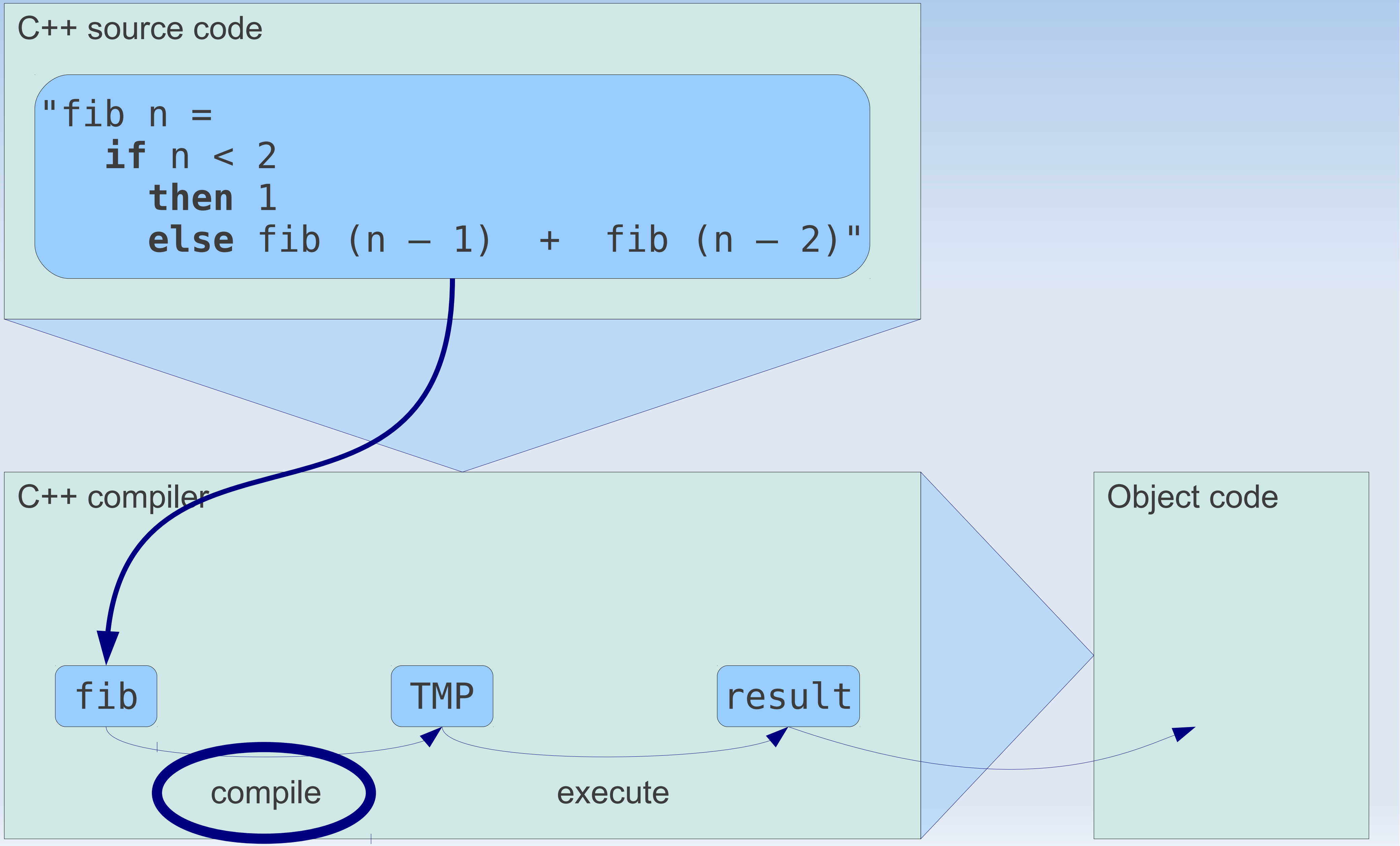
result

compile

execute

Object code

How it works?



C++ source code

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

C++ compiler

fib

TMP

result

compile

execute

Object code

The ”compile” step

- Generalised constant expressions (C++11)
- Template metaprograms

The ”compile” step

- Generalised constant expressions (C++11)
  - Familiar syntax
  - They have to build a value
  - How to build a metaprogram with them?
  - Sprout https://github.com/bolero-MURAKAMI/Sprout
- Template metaprograms

The ”compile” step

- Generalised constant expressions (C++11)
  - Familiar syntax
  - They have to build a value
  - How to build a metaprogram with them?
  - Sprout https://github.com/bolero-MURAKAMI/Sprout
- Template metaprograms
  - One can build types, functions, values with them
  - Complex syntax (familiar to metaprogrammers :))

Template metaprograms

- The string to compile is a string literal
- How to pass it to a template metaprogram?

Template metaprograms

- The string to compile is a string literal
- How to pass it to a template metaprogram?
- Boost.MPL

  ```
  boost::mpl::string<'Hell','o Wo','rld!'>
  ```

Template metaprograms

- The string to compile is a string literal
- How to pass it to a template metaprogram?
- Boost.MPL

  ```
  boost::mpl::string<'Hell','o Wo','rld!'>
  ```

- Mpllibs.Metaparse (C++11)

  ```
  MPLLIBS_STRING("Hello World!")
  ```

Template metaprograms

- The string to compile is a string literal
- How to pass it to a template metaprogram?
- Boost.MPL

```
boost::mpl::string<'Hell','o Wo','rld!'>
```

- Mpllibs.Metaparse (C++11)

```
MPLLIBS_STRING("Hello World!")
```

MPLLIBS_STRING

```
"Hello World!"
```

```
boost::mpl::string<'Hell','o Wo','rld!'>
```

MPLLIBS_STRING

```
"Hello World!"
```

```
boost::mpl::string<'Hell','o Wo','rld!'>
```

```
          string<>
```

MPLLIBS_STRING

```
"Hello World!"
```

```
boost::mpl::string<'Hell','o Wo','rld!'>
```

```
push_back<
  string<>,
  char_<       'H'       >
>::type
```

MPLLIBS_STRING

"Hello World!"

```
boost::mpl::string<'Hell','o Wo','rld!'>
```

```
push_back<
  push_back<
    string<>,
    char_<       'H'       >
  >::type,
  char_<       'e'       >
>::type
```

MPLLIBS_STRING

"Hello World!"

```
boost::mpl::string<'Hell','o Wo','rld!'>

  push_back<
    push_back<
      // ...
      push_back<
        push_back<
          string<>,
          char_<       'H'       >
        >::type,
        char_<       'e'       >
      >::type,
      char_<       'l'       >
      // ...
    >::type,
    char_<       '!'        >
  >::type
```

MPLLIBS_STRING

"Hello World!"

```
boost::mpl::string<'Hell','o Wo','rld!'>

  push_back<
    push_back<
      // ...
      push_back<
        push_back<
          string<>,
          char_<"Hello World!"[0]>
        >::type,
        char_<"Hello World!"[1]>
      >::type,
      char_<"Hello World!"[2]>
      // ...
    >::type,
    char_<"Hello World!"[11]>
  >::type
```

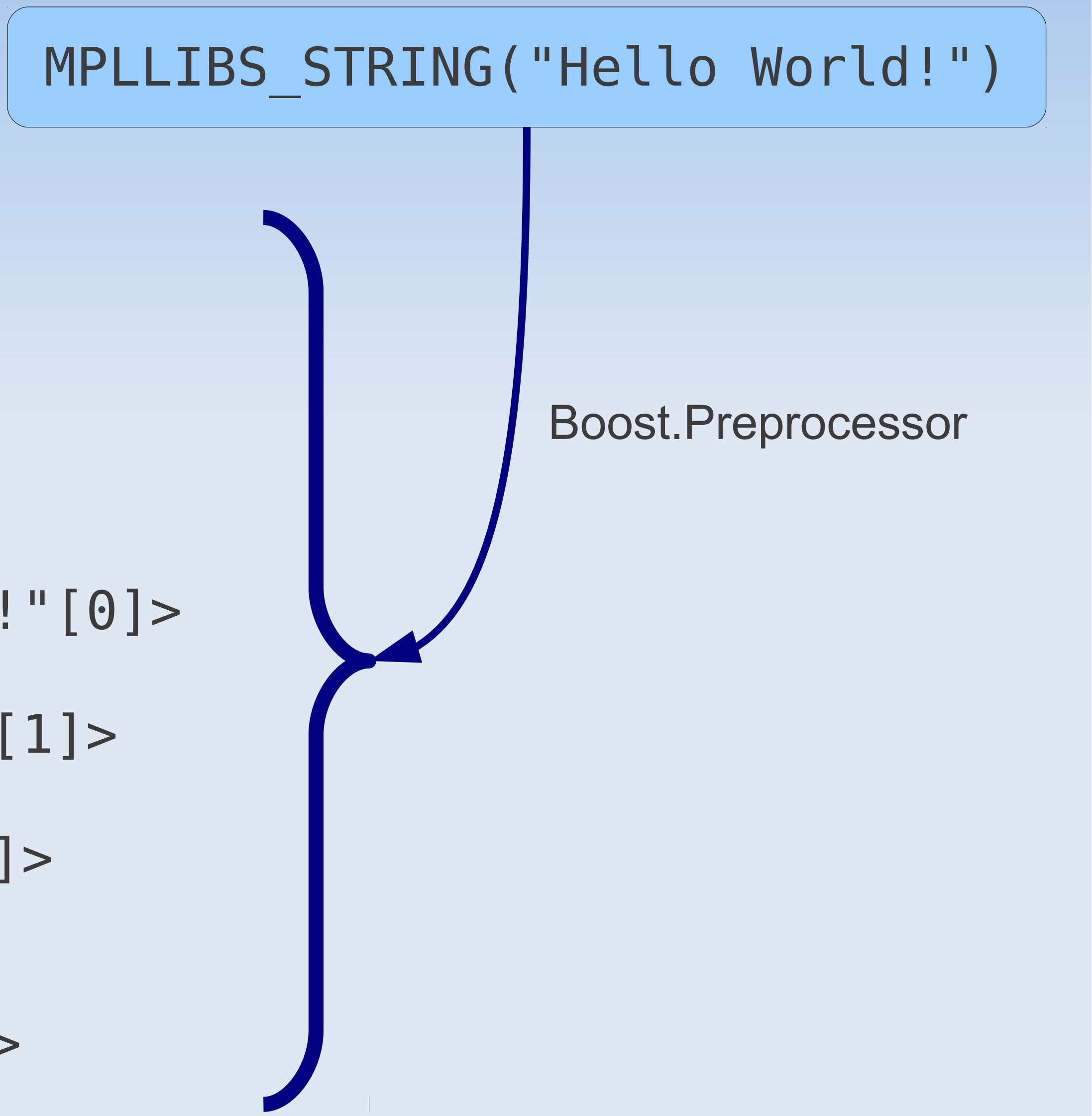
MPLLIBS_STRING

"Hello World!"

```
boost::mpl::string<'Hell','o Wo','rld!'>

  push_back<
    push_back<
      // ...
      push_back<
        push_back<
          string<>,
          char_<"Hello World!"[0]>
        >::type,
        char_<"Hello World!"[1]>
      >::type,
      char_<"Hello World!"[2]>
      // ...
    >::type,
    char_<"Hello World!"[11]>
  >::type
```

Constant expression

MPLLIBS_STRING



```
MPLLIBS_STRING("Hello World!")
```

Boost.Preprocessor

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<"Hello World!"[0]>
      >::type,
      char_<"Hello World!"[1]>
    >::type,
    char_<"Hello World!"[2]>
    // ...
  >::type,
  char_<"Hello World!"[11]>
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("Hello World!")
```

```
#define MPLLIBS_STRING(S) \
  push_back< \
    push_back< \
      // ...
      push_back< \
        push_back< \
          string<>, \
          char_<             S[0]> \
        >::type, \
        char_<             S[1]> \
      >::type, \
      char_<             S[2]> \
      // ...
    >::type, \
    char_<             S[11]> \
  >::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("Hello World!")
```

```
#define MPLLIBS_STRING(S) \
  push_back< \
    push_back< \
      // ...
      push_back< \
        push_back< \
          string<>, \
          char_<             S[0]> \
        >::type, \
        char_<             S[1]> \
      >::type, \
      char_<             S[2]> \
      // ...
    >::type, \
    char_<             S[11]> \
  >::type
```

```
#define PRE(z, n, u) \
  push_back<
```

MPLLIBS_STRING

```
MPLLIBS_STRING("Hello World!")
```

```
#define MPLLIBS_STRING(S) \
          \
          \
  BOOST_PP_REPEAT(12, PRE, ~) \
          \
          \
          string<>, \
          char_<                   S[0]> \
        >::type, \
        char_<                   S[1]> \
      >::type, \
      char_<                   S[2]> \
      // ...
    >::type, \
    char_<                   S[11]> \
  >::type
```

```
#define PRE(z, n, u) \
  push_back<
```

MPLLIBS_STRING

```
MPLLIBS_STRING("Hello World!")
```

```
#define MPLLIBS_STRING(S) \
          \
          \
  BOOST_PP_REPEAT(12, PRE, ~) \
          \
          \
          string<>, \
          char_<             S[0]> \
        >::type, \
        char_<             S[1]> \
      >::type, \
      char_<             S[2]> \
      // ...
    >::type, \
    char_<             S[11]> \
  >::type
```

```
#define PRE(z, n, u) \
  push_back<
```

```
#define POST(z, n, u) \
  , char_<S[n]>>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("Hello World!")
```

```
#define MPLLIBS_STRING(S) \
          \
          \
  BOOST_PP_REPEAT(12, PRE, ~) \
          \
          \
          string<>  \
          \
          \
  BOOST_PP_REPEAT(12, POST, ~)
```

```
#define PRE(z, n, u) \
  push_back<
```

```
#define POST(z, n, u) \
  , char_<S[n]>>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<   "X"[0] >
      >::type,
      char_<   "X"[1] >
    >::type,
    char_<   "X"[2] >
    // ...
  >::type,
  char_<   "X"[11] >
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<   "X"[0] >
      >::type,
      char_<   "X"[1] >
    >::type,
    char_<   "X"[2] >
    // ...
  >::type,
  char_<   "X"[11] >
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
constexpr char at(             s   ,     n)
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<   "X"[0] >
      >::type,
      char_<   "X"[1] >
    >::type,
    char_<   "X"[2] >
    // ...
  >::type,
  char_<   "X"[11] >
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
template <int N>
constexpr char at(const char (&)s[N], int n)
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<   "X"[0] >
      >::type,
      char_<   "X"[1] >
    >::type,
    char_<   "X"[2] >
    // ...
  >::type,
  char_<   "X"[11] >
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
template <int N>
constexpr char at(const char (&)s[N], int n)
{ return n >= N ? 0 : s[n]; }
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<   "X"[0] >
      >::type,
      char_<   "X"[1] >
    >::type,
    char_<   "X"[2] >
    // ...
  >::type,
  char_<   "X"[11] >
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
template <int N>
constexpr char at(const char (&)s[N], int n)
{ return n >= N ? 0 : s[n]; }
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<at("X", 0)>
      >::type,
      char_<at("X", 1)>
    >::type,
    char_<at("X", 2)>
    // ...
  >::type,
  char_<at("X", 11)>
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<at("X", 0)>
      >::type,
      char_<at("X", 1)>
    >::type,
    char_<at("X", 2)>
    // ...
  >::type,
  char_<at("X", 11)>
>::type
```

```
MPLLIBS_STRING("X\0\0...\0")
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
template <class S, char C, bool EOS>
struct push_back_if;
```

```
MPLLIBS_STRING("X\0\0...\0")
```

```
push_back<
  push_back<
    // ...
    push_back<
      push_back<
        string<>,
        char_<at("X", 0)>
      >::type,
      char_<at("X", 1)>
    >::type,
    char_<at("X", 2)>
    // ...
  >::type,
  char_<at("X", 11)>
>::type
```

MPLLIBS_STRING

```
MPLLIBS_STRING("X")
```

```
template <class S, char C, bool EOS>
struct push_back_if;
```

```
MPLLIBS_STRING("X\0\0...\0")
```

```
push_back_if<
  push_back_if<
    // ...
    push_back_if<
      push_back_if<
        string<>,
        at("X", 0), (0 < sizeof("X"))
      >::type,
      at("X", 1), (1 < sizeof("X"))
    >::type,
    at("X", 2), (2 < sizeof("X"))
    // ...
  >::type,
  at("X", 11), (11 < sizeof("X"))
>::type
```

MPLLIBS_STRING

- This solution can't deal with strings longer than 11 characters

MPLLIBS_STRING

- This solution can't deal with strings longer than 11 characters
- This limit can be configurable

```
#define MPLLIBS_LIMIT_STRING 11
```

Parsers

- A parser is a template metafunction (class)

```
struct sample_parser {
  template <class S, class Pos>
  struct apply : /* … */ {};
};
```

Parsers

- A parser is a template metafunction (class)

  ```
  struct sample_parser {
    template <class S, class Pos>
    struct apply : /* … */ {};
  };
  ```

- Return values:
  - Result, remaining string, source position
  - Error

Parsers

- A parser is a templa

```
struct sample_pars
  template <class
  struct apply : /
};
```

```
template <class Result>
struct return_ {







};
```

- Return values:
  - Result, remaining string, source position
  - Error

Parsers

- A parser is a templa

```
struct sample_pars
  template <class
  struct apply : /
};
```

```
template <class Result>
struct return_ {
  template <class S, class Pos>
  struct apply {
    typedef Result result;
    typedef S remaining;
    typedef Pos source_position;
  };
};
```

- Return values:
  - Result, remaining string, source position
  - Error

Parsers

- A parser is a templa

```
struct sample_pars
   template <class
   struct apply : /
};
```

```
template <class Result>
struct return_ {
  template <class S, class Pos>
  struct apply {
    typedef Result result;
    typedef S remaining;
    typedef Pos source_position;
  };
};
```

- Return values:

ource position

```
template <class Msg>
struct fail {






};
```

Parsers

- A parser is a templa

```
struct sample_pars
   template <class
   struct apply : /
};
```

- Return values:

```
template <class Result>
struct return_ {
   template <class S, class Pos>
   struct apply {
      typedef Result result;
      typedef S remaining;
      typedef Pos source_position;
   };
};
```

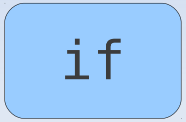
```
template <class Msg>
struct fail {
   template <class S, class Pos>
   struct apply {
      typedef Msg reason;
      typedef Pos source_position;
   };
};
```

ource position

Parsers

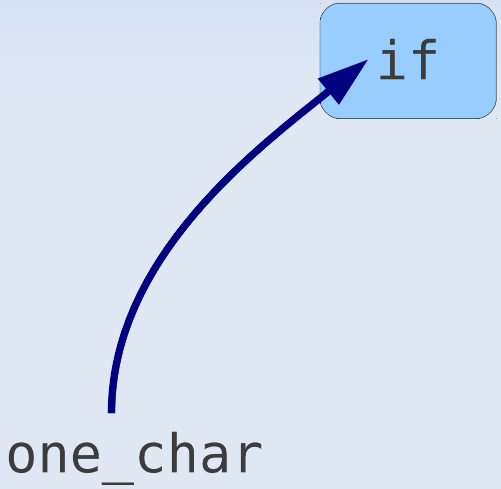
```
struct one_char {
  template <class S, class Pos>
  struct apply {
    typedef typename mpl::front<S>::type result;
    typedef typename mpl::pop_front<S>::type remaining;
    typedef /* ... */ source_position;
  };
};
```

Parsers

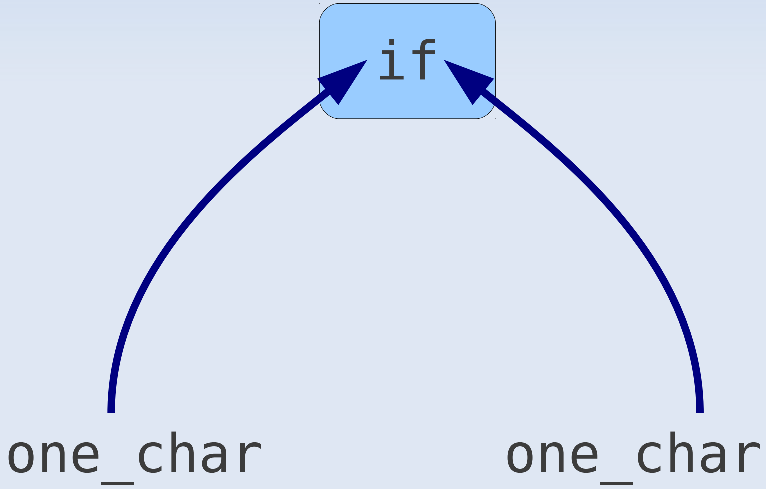


if

Parsers



Parsers



Parser combinators

- A parser combinator is a function taking parsers as arguments and building new parsers
- Higher-order function

Parser combinators

```
template <class P, class Pred, class Msg>
struct accept_when {

};
```

Parser combinators

```
template <class P, class Pred, class Msg>
struct accept_when {
  template <class R>
  struct impl : mpl::apply<
    typename mpl::if_<
      typename mpl::apply<Pred, typename R::result>::type,
      return_<typename R::result>,
      fail<Msg>
    >::type,
    typename R::remaining, typename R::source_position
  > {};

  template <class S, class Pos>
  struct apply : mpl::eval_if<
    typename is_error<mpl::apply<P, S, Pos>>::type,
    mpl::apply<P, S, Pos>,
    impl<typename mpl::apply<P, S, Pos>::type>
  > {};
};
```

Parser combinators

```
template <class C>
struct lit :

{};
```

Parser combinators

```
template <class C>
struct lit :
  accept_when<one_char, mpl::equal_to<mpl::_1, C>, ...>
{};
```

DSL for template metaprograms

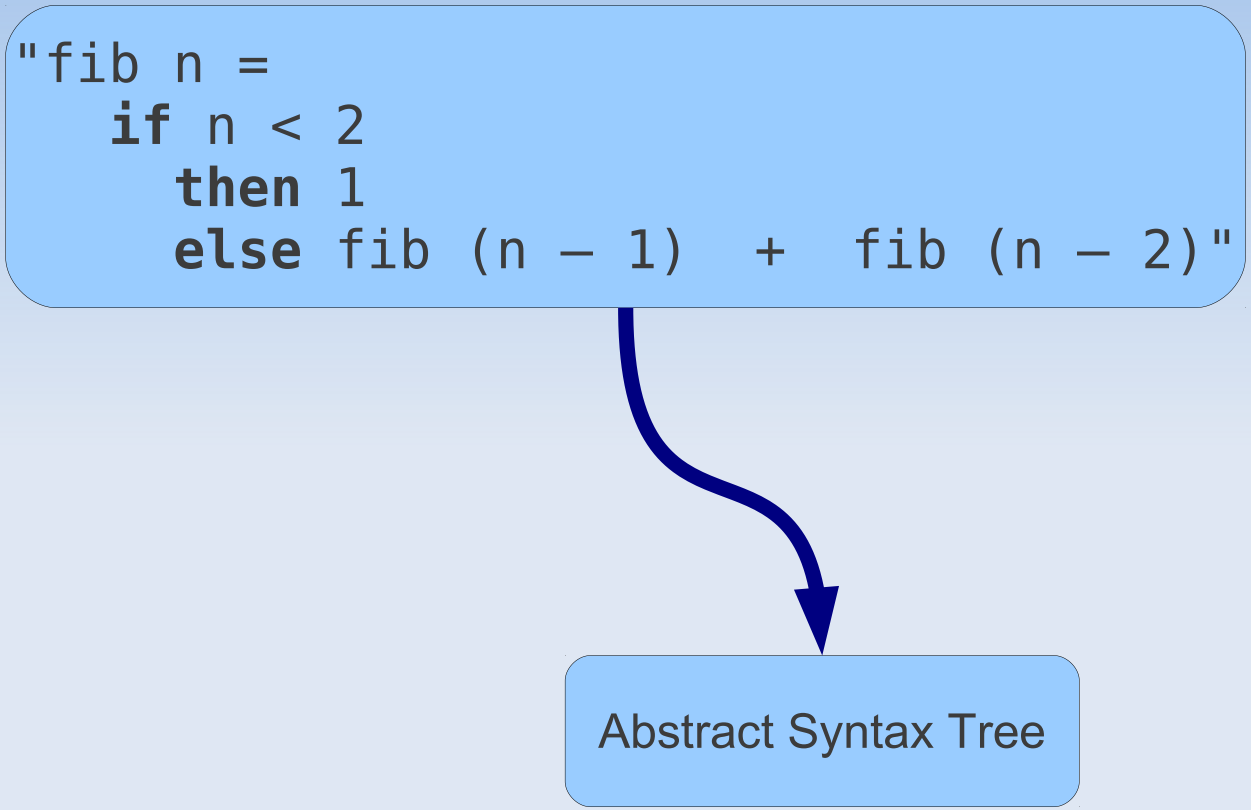
- Template metaprograms are pure functional programs
- We should follow the syntax of a functional language (Haskell)
- Write metaprograms in a Haskell-like language

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

Building the DSL

```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

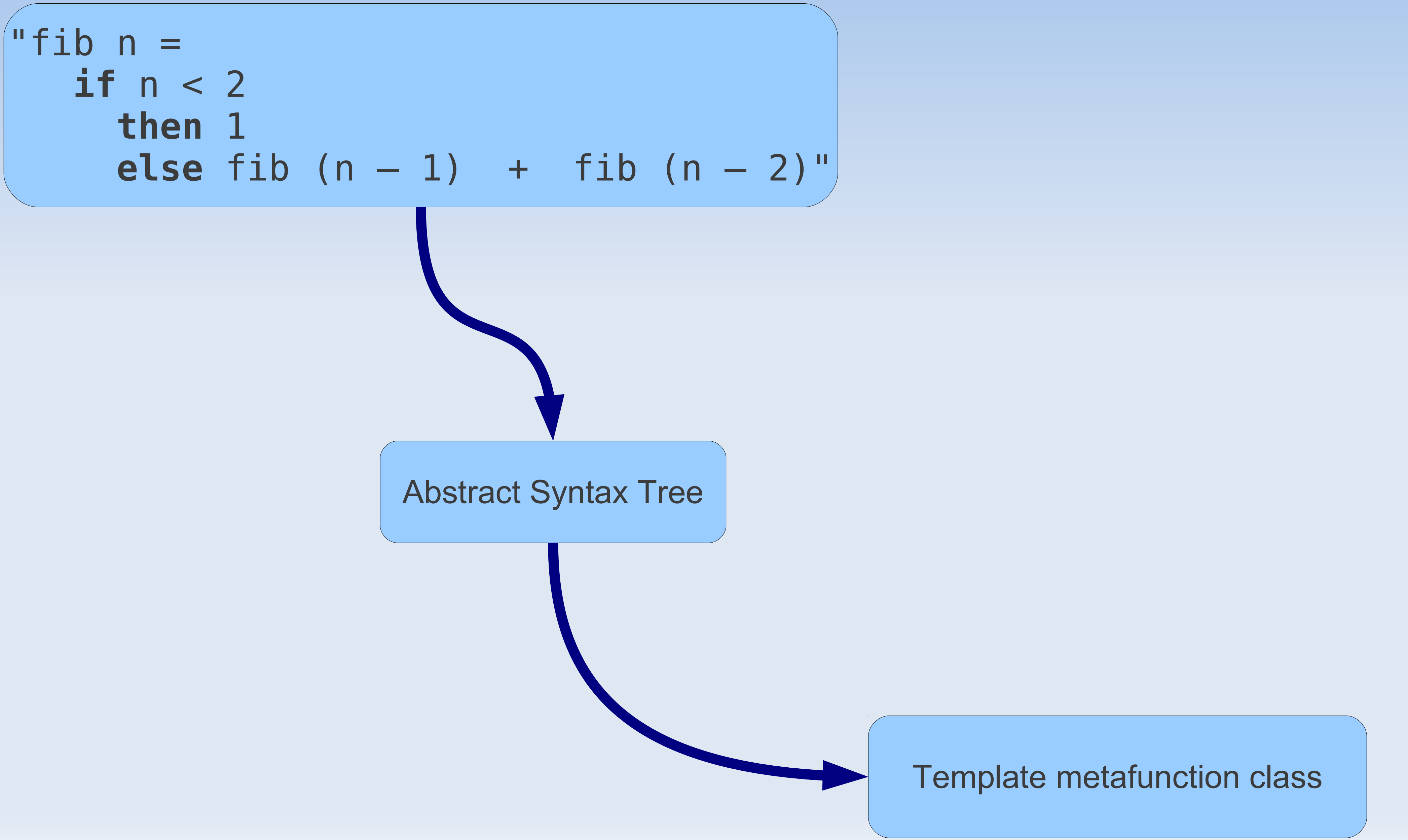
Building the DSL



```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

Abstract Syntax Tree

Building the DSL



```
"fib n =
   if n < 2
     then 1
     else fib (n – 1)  +  fib (n – 2)"
```

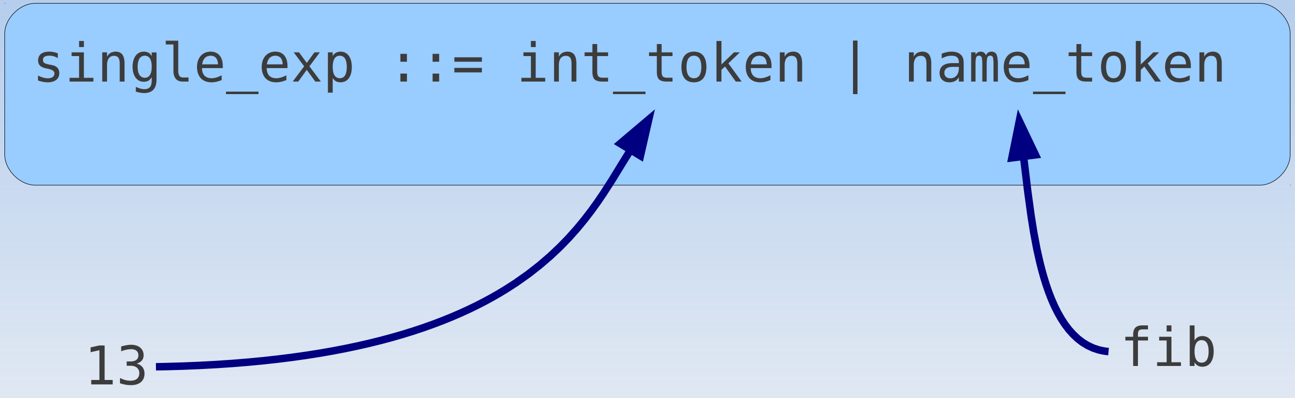
Abstract Syntax Tree

Template metafunction class

Building the DSL

```
single_exp ::= int_token | name_token
```

Building the DSL

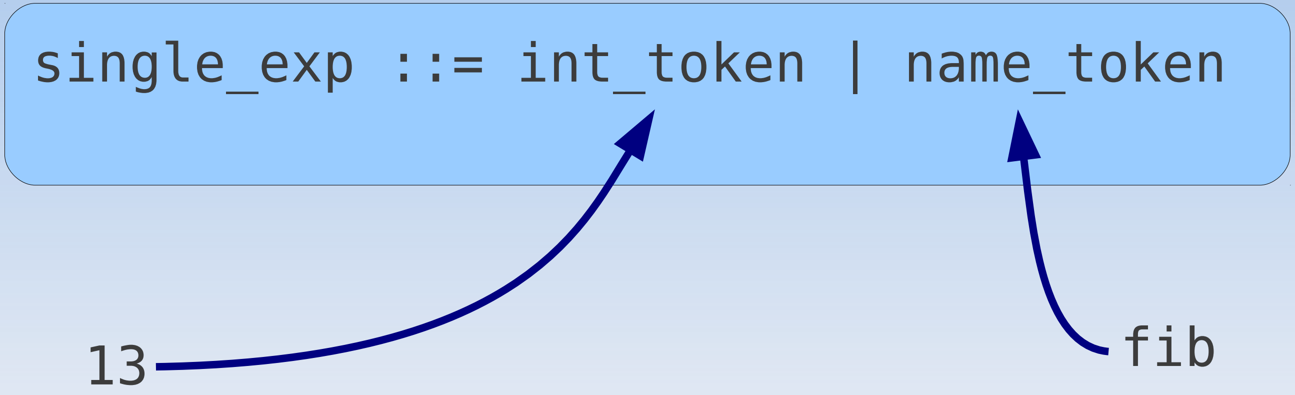


```
single_exp ::= int_token | name_token
```

13

fib

Building the DSL



```
single_exp ::= int_token | name_token
```

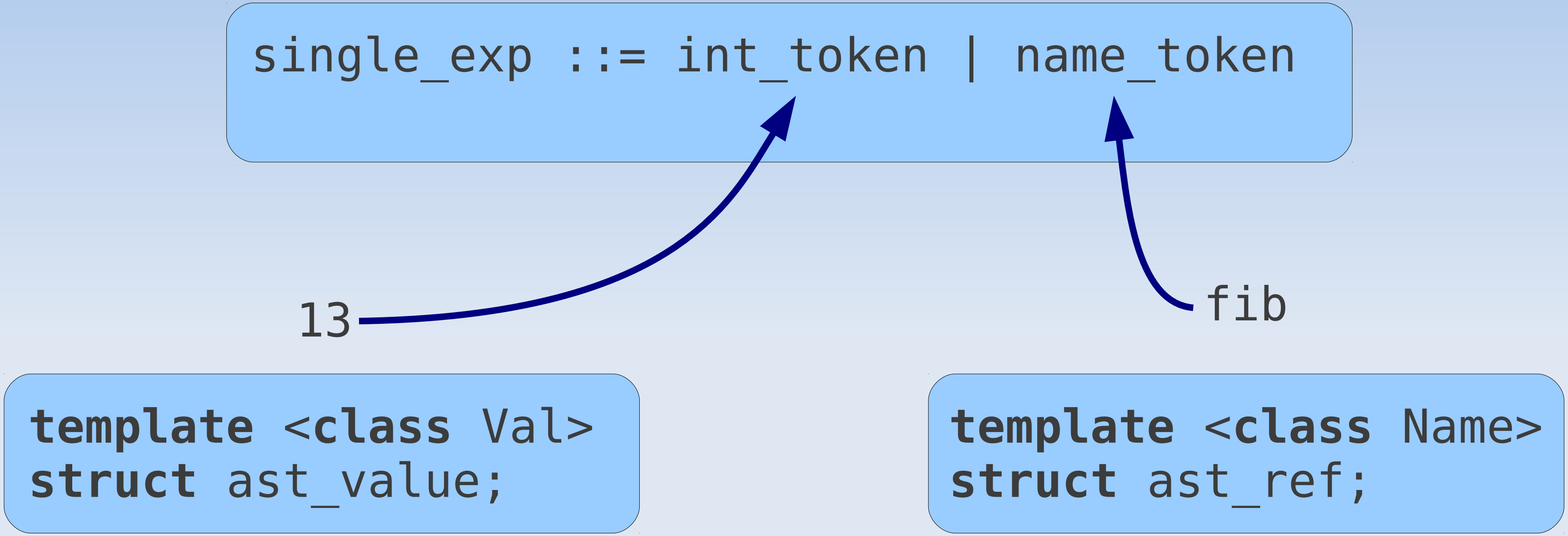
13

fib

```
template <class Val>
struct ast_value;
```

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

Building the DSL



```
single_exp ::= int_token | name_token
```

13

fib

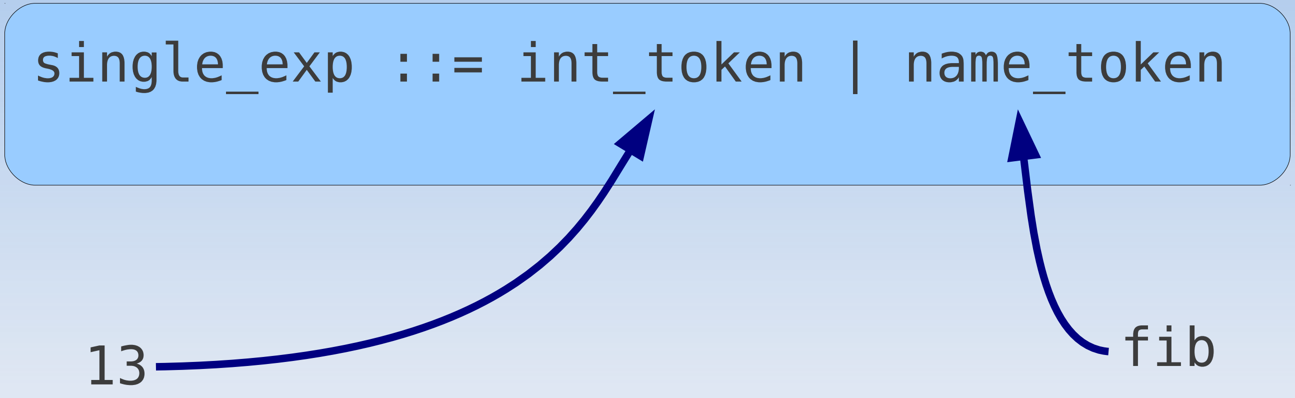
```
template <class Val>
struct ast_value;
```

```
template <class Name>
struct ast_ref;
```

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

```
ast_ref<_S("fib")>
```

Building the DSL



```
single_exp ::= int_token | name_token
```

13

fib

```
template <class Val>
struct ast_value;
```

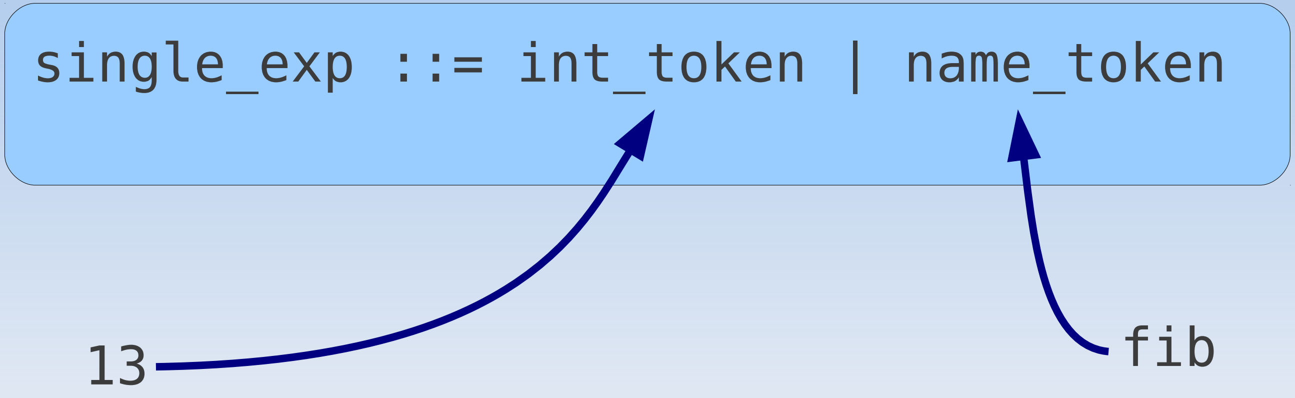
```
template <class Name>
struct ast_ref;
```

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

```
ast_ref<_S("fib")>
```

```
typedef transform<
  int_token,
  mpl::lambda<
    ast_value<mpl::_1>
  >::type
> int_exp;
```

Building the DSL



```
single_exp ::= int_token | name_token
```

13

fib

```
template <class Val>
struct ast_value;
```

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

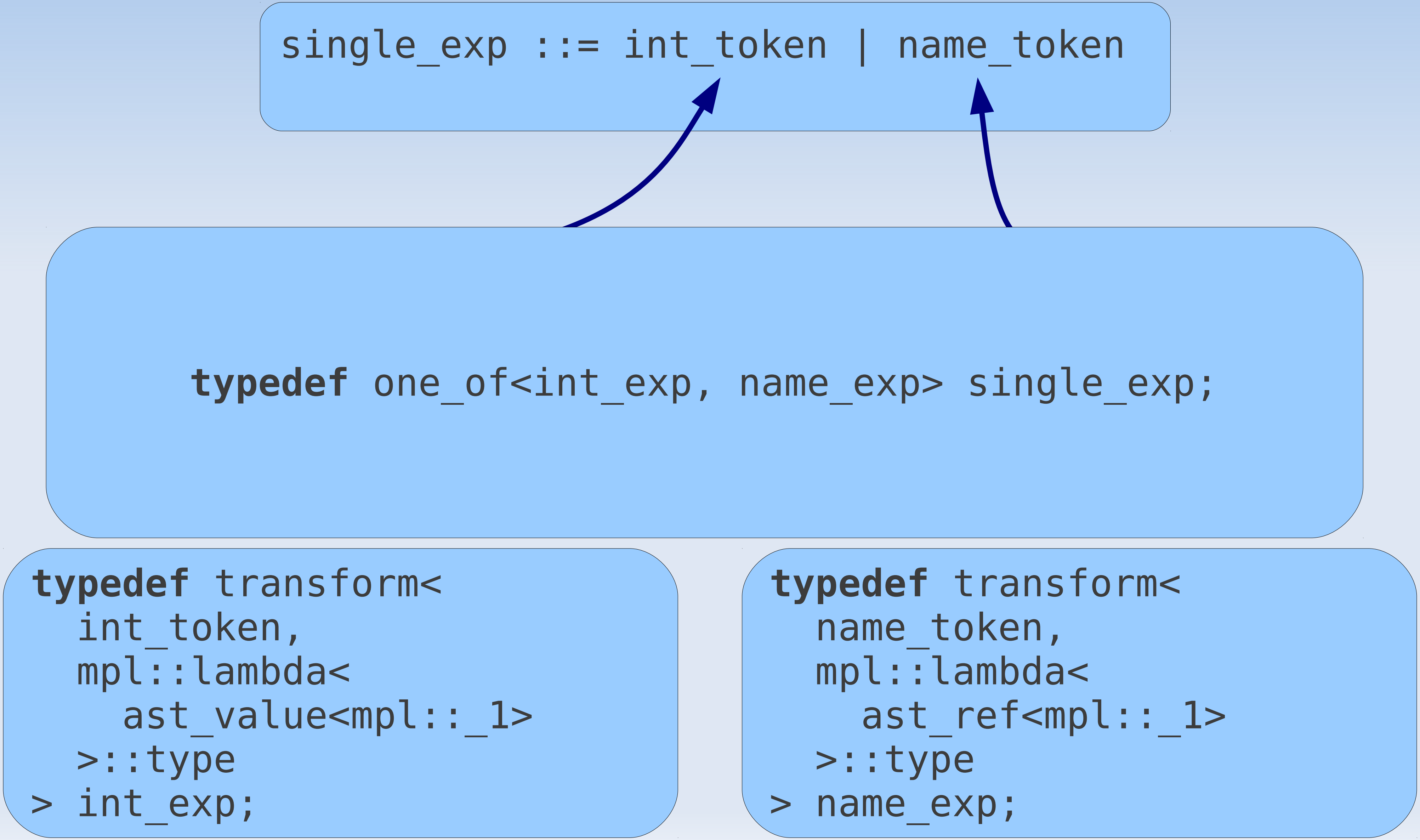
```
typedef transform<
  int_token,
  mpl::lambda<
    ast_value<mpl::_1>
  >::type
> int_exp;
```

```
template <class Name>
struct ast_ref;
```

```
ast_ref<_S("fib")>
```

```
typedef transform<
  name_token,
  mpl::lambda<
    ast_ref<mpl::_1>
  >::type
> name_exp;
```

Building the DSL



```
single_exp ::= int_token | name_token
```

```
typedef one_of<int_exp, name_exp> single_exp;
```

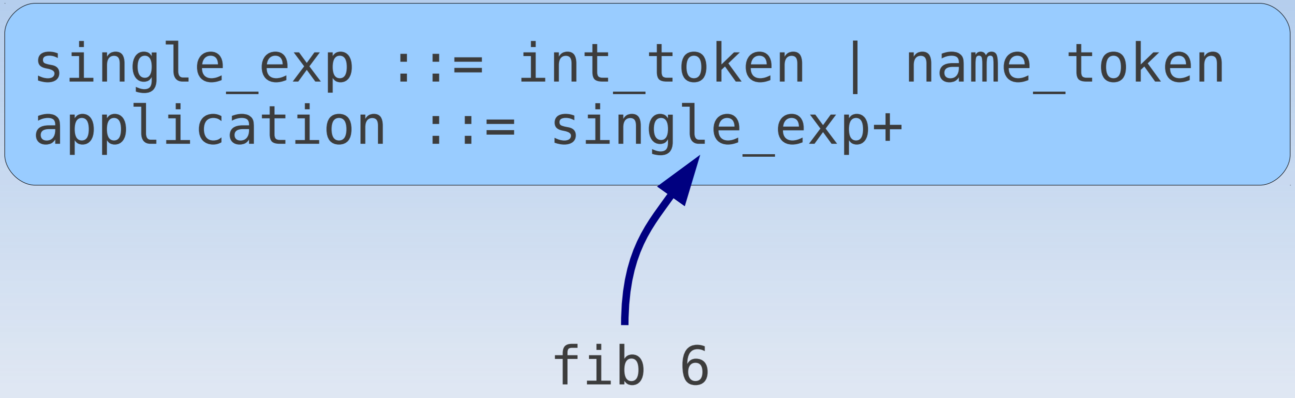
```
typedef transform<
  int_token,
  mpl::lambda<
    ast_value<mpl::_1>
  >::type
> int_exp;
```

```
typedef transform<
  name_token,
  mpl::lambda<
    ast_ref<mpl::_1>
  >::type
> name_exp;
```

Building the DSL

```
single_exp ::= int_token | name_token
application ::= single_exp+
```

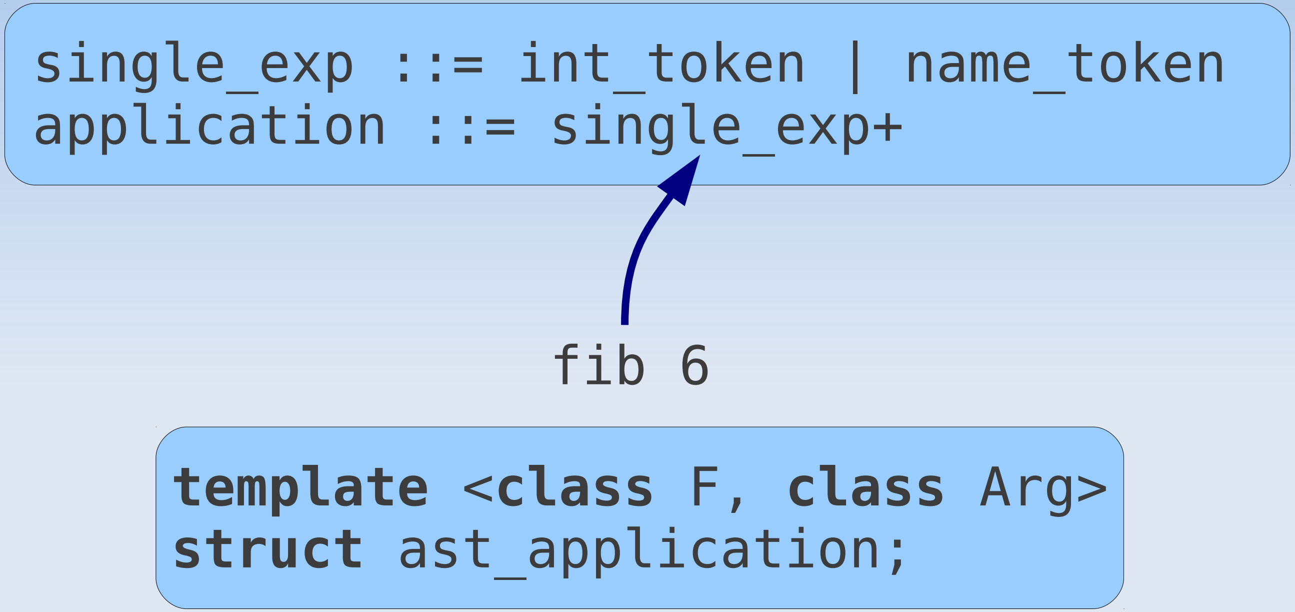
Building the DSL



```
single_exp ::= int_token | name_token
application ::= single_exp+
```

fib 6

Building the DSL



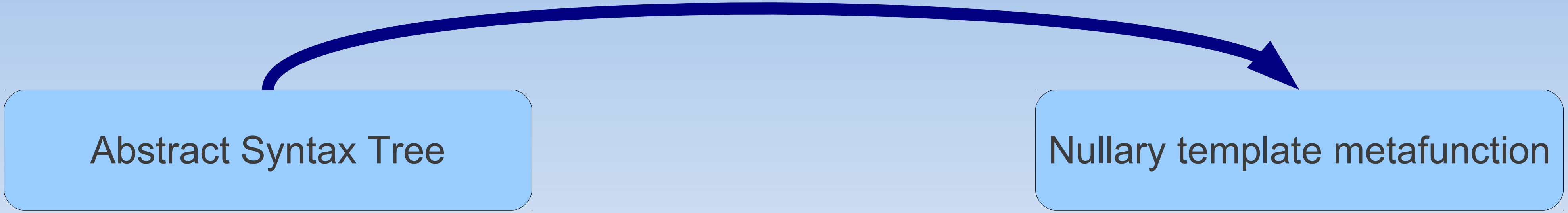
```
single_exp ::= int_token | name_token
application ::= single_exp+
```

```
fib 6
```

```
template <class F, class Arg>
struct ast_application;
```

```
ast_application<
  ast_ref<_S("fib")>,
  ast_value<mpl::int_<6>>
>
```

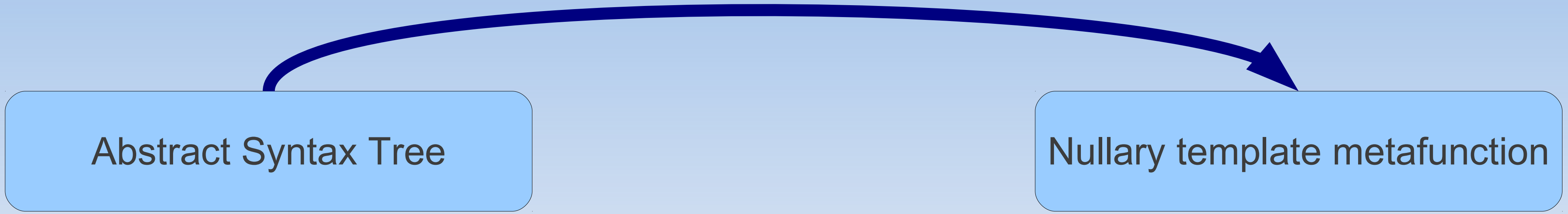
Building the DSL



Abstract Syntax Tree

Nullary template metafunction

Building the DSL

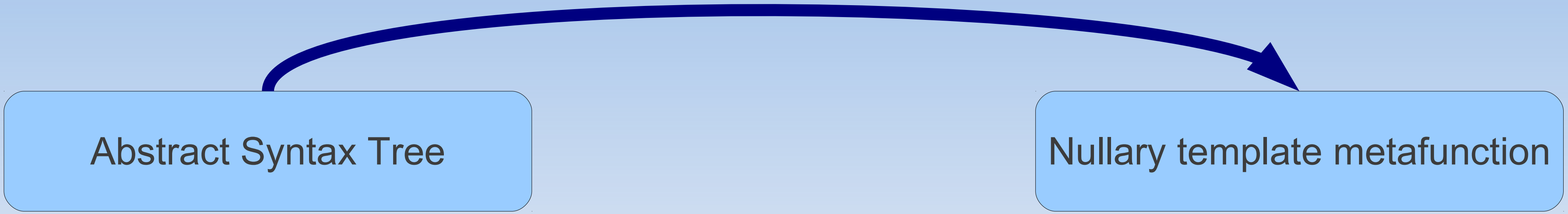


Abstract Syntax Tree

Nullary template metafunction

```
template <class AST>
struct bind;
```

Building the DSL



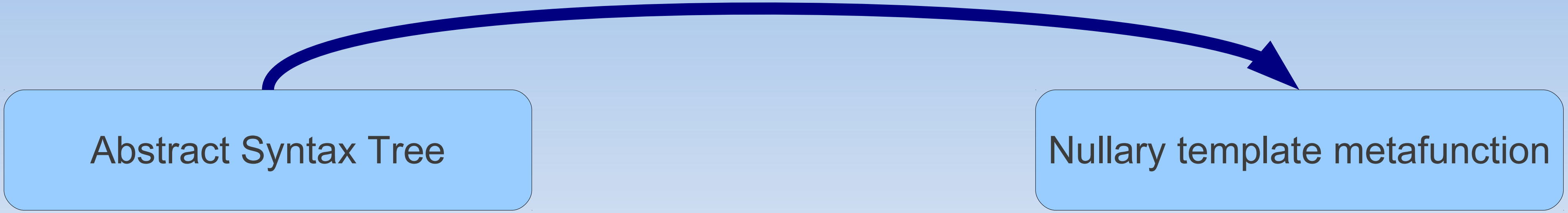
Abstract Syntax Tree

Nullary template metafunction

```
template <class AST>
struct bind;
```

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

ast_application<F, A>

ast_ref<_S("fib")>
```

Building the DSL


Abstract Syntax Tree

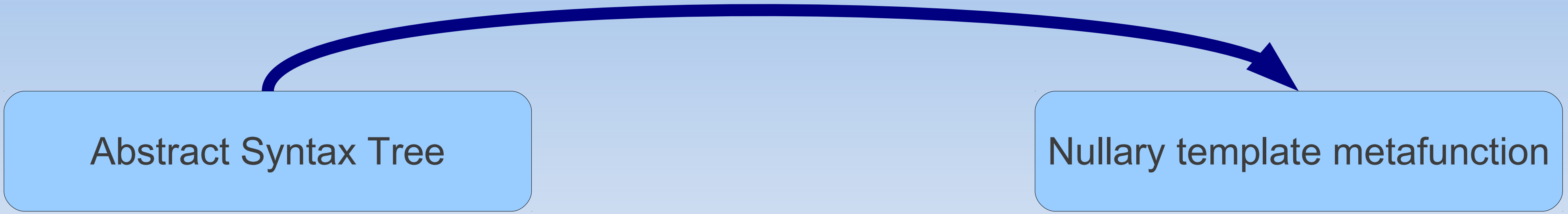
Nullary template metafunction

```
template <class AST>
struct bind;
```

```
ast_value<mpl::int_<13>> → lazy_value<mpl::int_<13>>

ast_application<F, A>

ast_ref<_S("fib")>
```

```
template <class V>
struct lazy_value {
  typedef V type;
};
```

Building the DSL


Abstract Syntax Tree

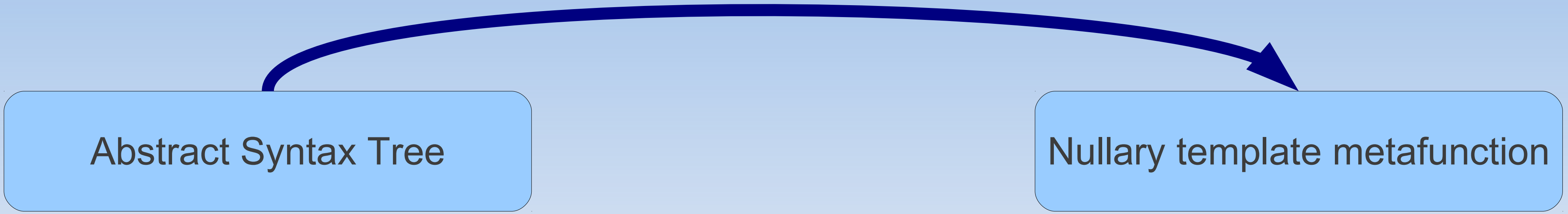
Nullary template metafunction

```
template <class AST>
struct bind;
```

```
ast_value<mpl::int_<13>> → lazy_value<mpl::int_<13>>

ast_application<F, A> → lazy_application<F, A>

ast_ref<_S("fib")>
```

```
template <class F, class Arg>
struct lazy_application :
  mpl::apply<typename F::type, A>
{};
```

Building the DSL



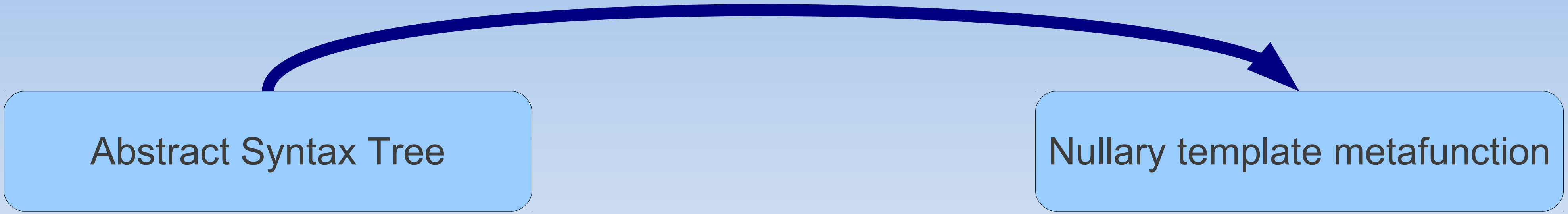
Abstract Syntax Tree

Nullary template metafunction

```
template <class AST>
struct bind;
```

```
ast_value<mpl::int_<13>> → lazy_value<mpl::int_<13>>

ast_application<F, A>  →  lazy_application<F, A>

ast_ref<_S("fib")>  →  ???
```

Building the DSL



Abstract Syntax Tree

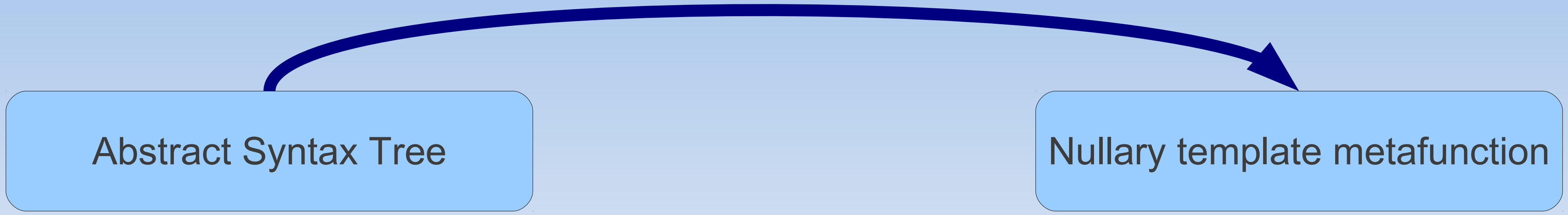
Nullary template metafunction

```
template <class AST>
struct bind;
```

```
ast_value<mpl::int_<13>> → lazy_value<mpl::int_<13>>

ast_application<F, A> → lazy_application<F, A>

ast_ref<_S("fib")> → ???
```

Lookup table

Building the DSL

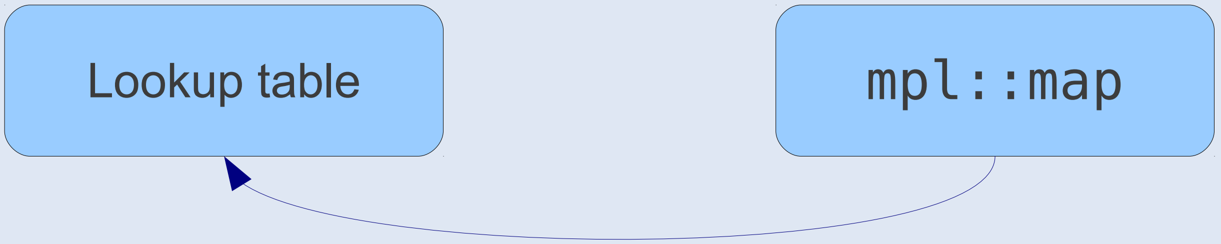


Abstract Syntax Tree

Nullary template metafunction

```
template <class AST>
struct bind;
```

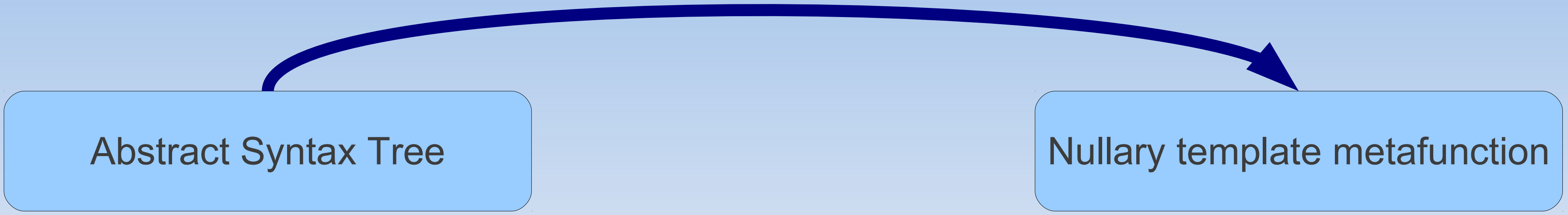
```
ast_value<mpl::int_<13>> → lazy_value<mpl::int_<13>>

ast_application<F, A> → lazy_application<F, A>

ast_ref<_S("fib")> → ???
```



Lookup table

mpl::map

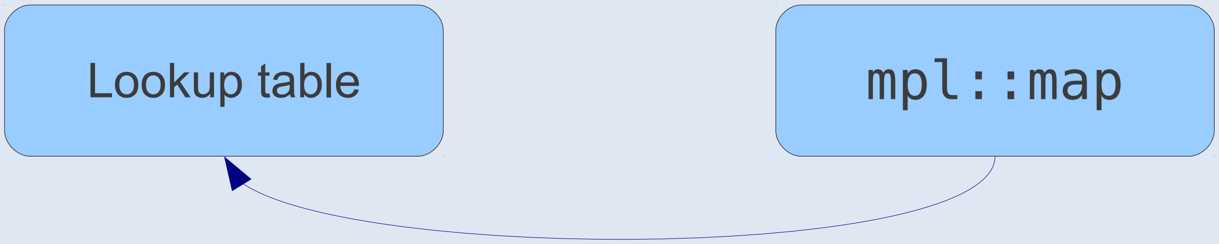
Building the DSL


Abstract Syntax Tree

Nullary template metafunction

```
template <class AST, class Env>
struct bind;
```

```
ast_value<mpl::int_<13>>  →  lazy_value<mpl::int_<13>>

ast_application<F, A>  →  lazy_application<F, A>

ast_ref<_S("fib")>  →  ???
```

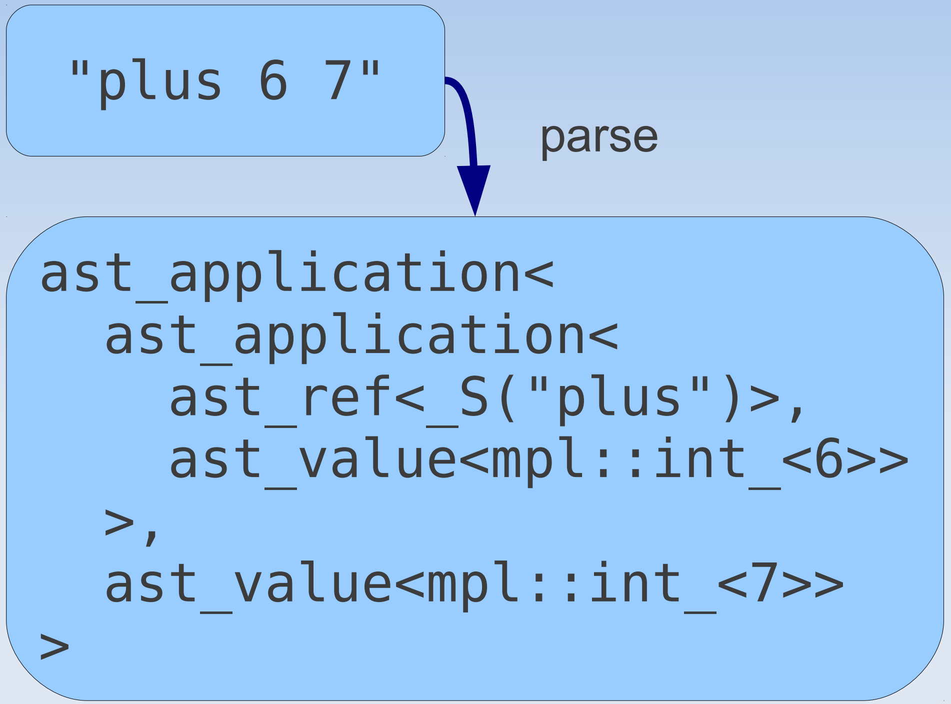

Lookup table

mpl::map

Building the DSL

```
"plus 6 7"
```

Building the DSL

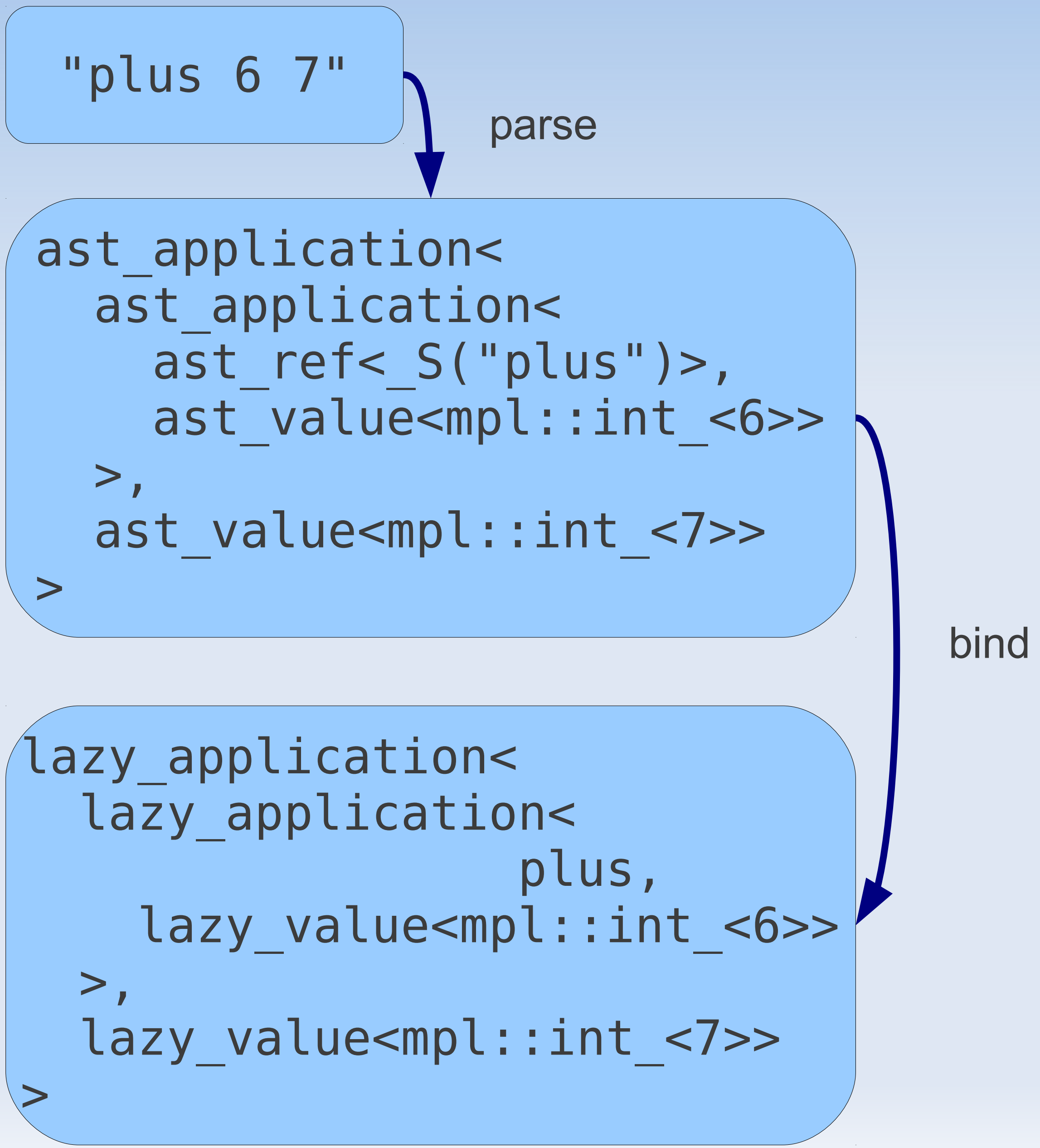


```
"plus 6 7"
```

parse

```
ast_application<
  ast_application<
    ast_ref<_S("plus")>,
    ast_value<mpl::int_<6>>
  >,
  ast_value<mpl::int_<7>>
>
```

Building the DSL

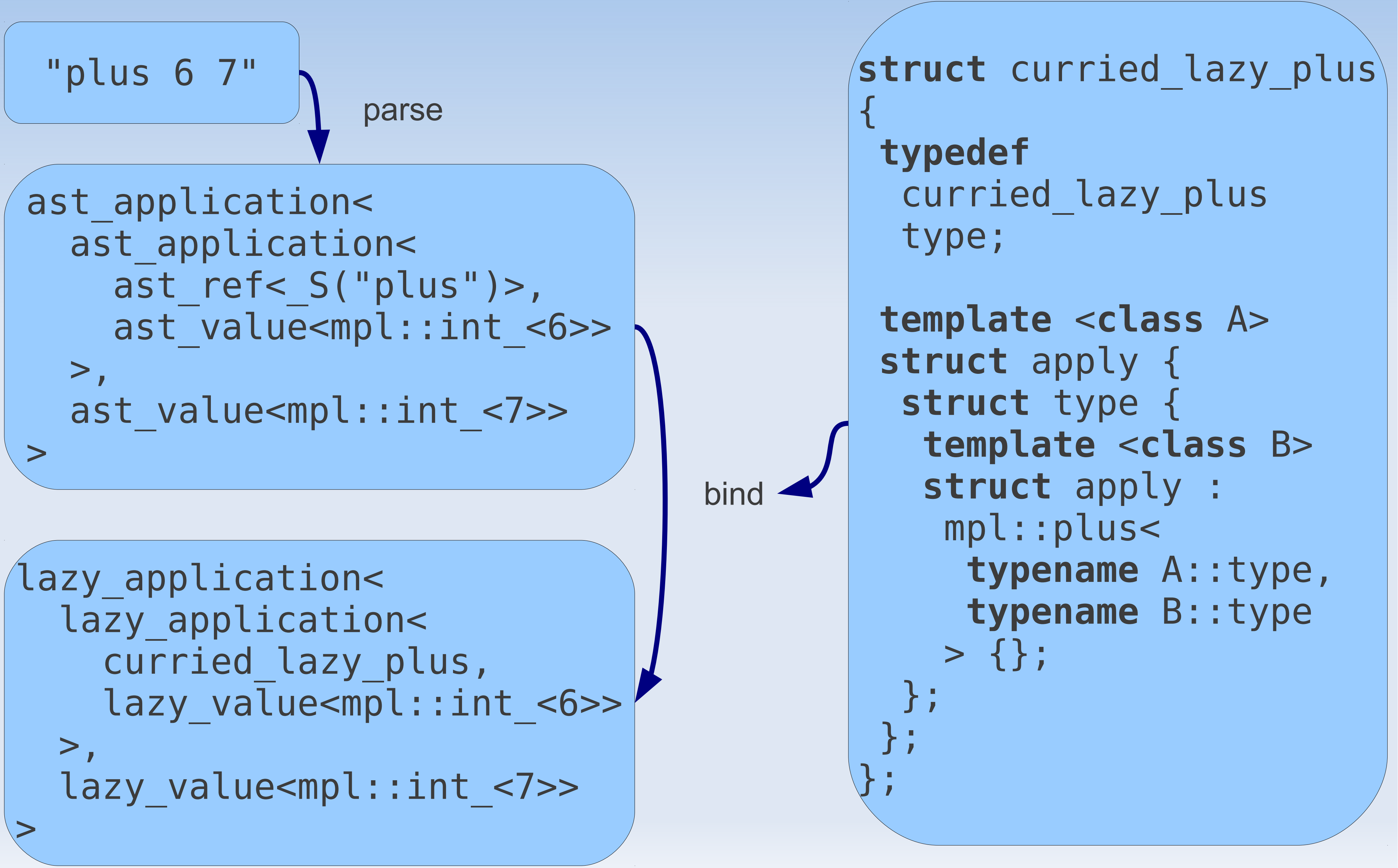


```
"plus 6 7"
```

```
ast_application<
  ast_application<
    ast_ref<_S("plus")>,
    ast_value<mpl::int_<6>>
  >,
  ast_value<mpl::int_<7>>
>
```

```
lazy_application<
  lazy_application<
                 plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>
```

Building the DSL



```
"plus 6 7"
```

parse

```
ast_application<
  ast_application<
    ast_ref<_S("plus")>,
    ast_value<mpl::int_<6>>
  >,
  ast_value<mpl::int_<7>>
>
```

```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>
```

bind

```
struct curried_lazy_plus
{
 typedef
  curried_lazy_plus
  type;

 template <class A>
 struct apply {
  struct type {
   template <class B>
   struct apply :
    mpl::plus<
     typename A::type,
     typename B::type
    > {};
  };
 };
};
```

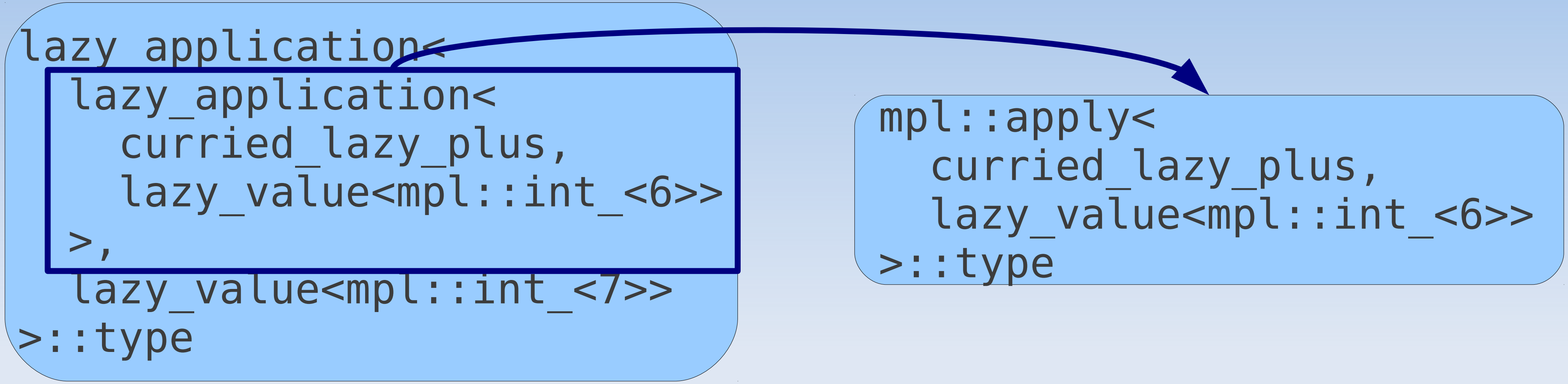
Building the DSL

```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>::type
```

Building the DSL

```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>::type
```

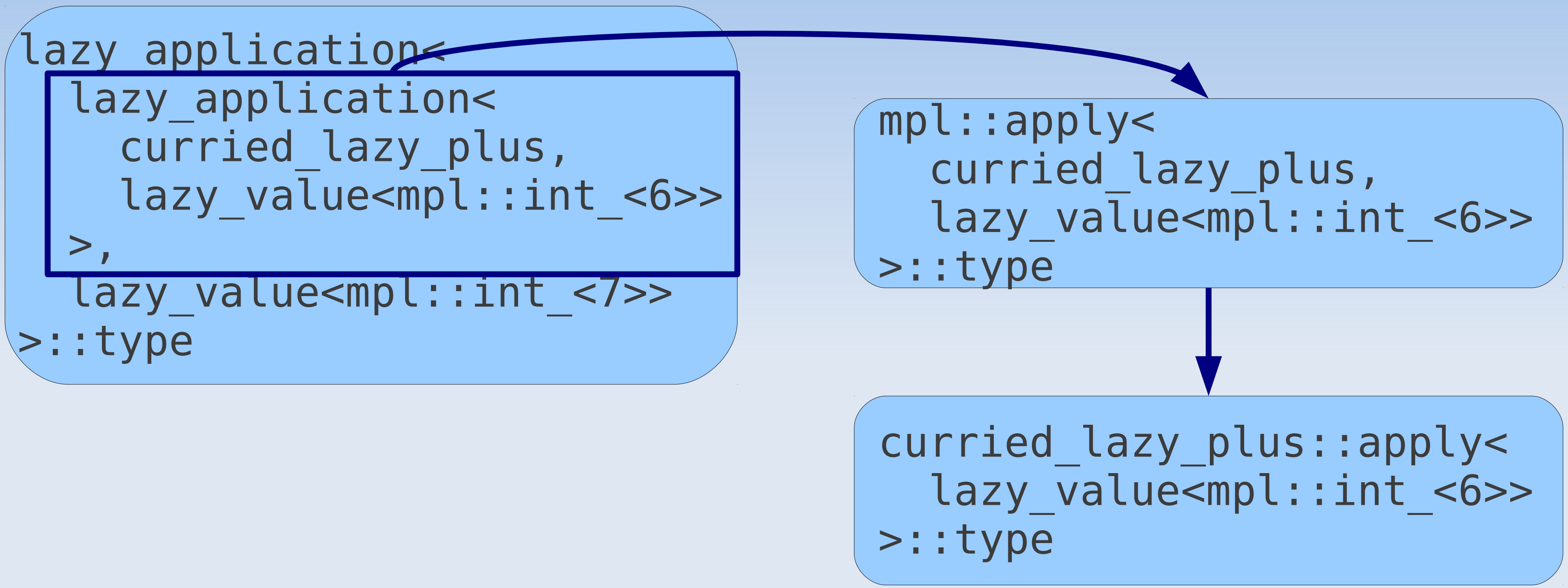
Building the DSL



```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>::type
```

```
mpl::apply<
  curried_lazy_plus,
  lazy_value<mpl::int_<6>>
>::type
```

Building the DSL

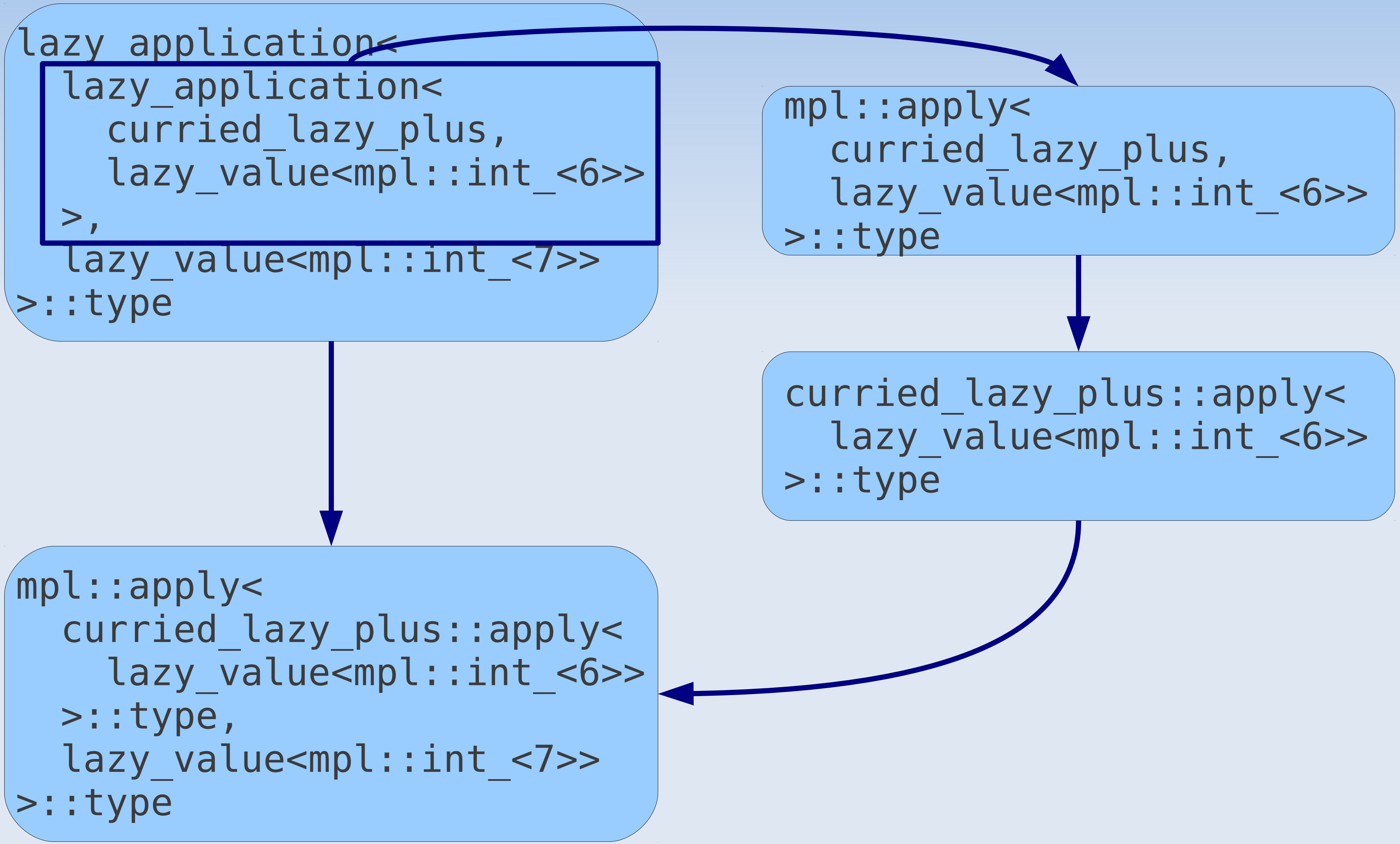


```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>::type
```

```
mpl::apply<
  curried_lazy_plus,
  lazy_value<mpl::int_<6>>
>::type
```

```
curried_lazy_plus::apply<
  lazy_value<mpl::int_<6>>
>::type
```

Building the DSL



```
lazy_application<
  lazy_application<
    curried_lazy_plus,
    lazy_value<mpl::int_<6>>
  >,
  lazy_value<mpl::int_<7>>
>::type
```

```
mpl::apply<
  curried_lazy_plus,
  lazy_value<mpl::int_<6>>
>::type
```

```
curried_lazy_plus::apply<
  lazy_value<mpl::int_<6>>
>::type
```

```
mpl::apply<
  curried_lazy_plus::apply<
    lazy_value<mpl::int_<6>>
  >::type,
  lazy_value<mpl::int_<7>>
>::type
```

Building the DSL

```
template <class Env>
struct builder {
  typedef builder type;

};
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

};
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import










};
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import : builder<

  > {};







};
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<Env, mpl::pair<Name, V>>::type
  > {};

};
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<Env, mpl::pair<Name, V>>::type
  > {};
```

```
};
```

```
meta_hs
  ::import<_S("plus"), curried_lazy_plus>::type
  ::import<_S("minus"), curried_lazy_minus>::type;
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<Env, mpl::pair<Name, V>>::type
  > {};

  template <class Name, template <class> class F>
  struct import1 : import<Name, curry1<F>> {};

  template <class Name, template <class, class> class F>
  struct import2 : import<Name, curry2<F>> {};

  // ...
};
```

```
meta_hs
  ::import<_S("plus"), curried_lazy_plus>::type
  ::import<_S("minus"), curried_lazy_minus>::type;
```

Building the DSL

```
typedef builder<mpl::map<>> meta_hs;
```

```
template <class Env>
struct builder {
  typedef builder type;

  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<Env, mpl::pair<Name, V>>::type
  > {};

  template <class Name, template <class> class F>
  struct import1 : import<Name, curry1<F>> {};

  template <class Name, template <class, class> class F>
  struct import2 : import<Name, curry2<F>> {};

  // ...
};
```

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type;
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type

  ::define<_S("x = minus y 2")>::type
  ::define<_S("y = plus 6 7")>::type;
```

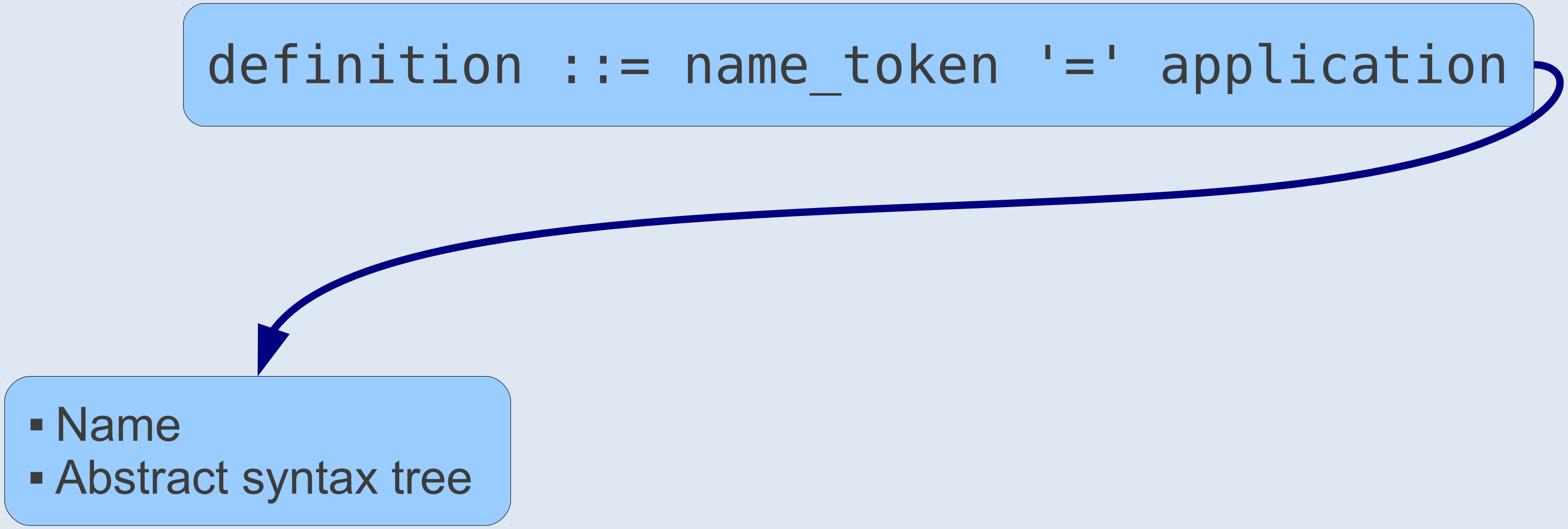
Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type

  ::define<_S("x = minus y 2")>::type
  ::define<_S("y = plus 6 7")>::type;
```

```
definition ::= name_token '=' application
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type

  ::define<_S("x = minus y 2")>::type
  ::define<_S("y = plus 6 7")>::type;
```

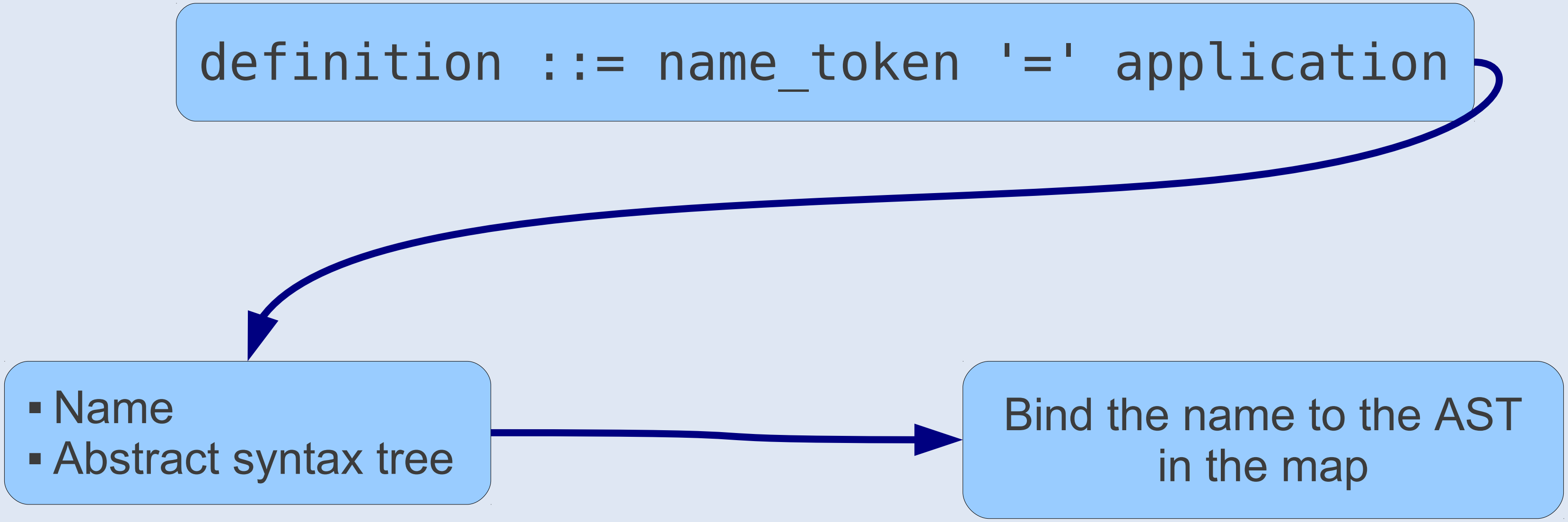


```
definition ::= name_token '=' application
```

- Name
- Abstract syntax tree

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::import2<_S("minus"), lazy_minus>::type

  ::define<_S("x = minus y 2")>::type
  ::define<_S("y = plus 6 7")>::type;
```



```
definition ::= name_token '=' application
```

- Name
- Abstract syntax tree

Bind the name to the AST in the map

Building the DSL

- We store ASTs in the map instead of values
- We need to turn ”normal” values into ASTs
  - Not into lazy values!

Building the DSL

- We store ASTs in the map instead of values
- We need to turn ”normal” values into ASTs
  - Not into lazy values!

```
template <class V>
struct ast_bound;
```

Building the DSL

- We store ASTs in the map instead of values
- We need to turn ”normal” values into ASTs
  - Not into lazy values!

```
template <class V>
struct ast_bound;
```

```
template <class Env>
struct builder {
  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<
      Env,
      mpl::pair<Name,           V >
    >::type
  > {};
  // ...
};
```

Building the DSL

- We store ASTs in the map instead of values
- We need to turn ”normal” values into ASTs
  - Not into lazy values!

```
template <class V>
struct ast_bound;
```

```
template <class Env>
struct builder {
  template <class Name, class V>
  struct import : builder<
    typename mpl::insert<
      Env,
      mpl::pair<Name, ast_bound<V>>
    >::type
  > {};
  // ...
};
```

Building the DSL

- The rules of binding have changed
  - We assumed that there is a value in the Env map
  - Now there is an AST instead
  - We need to bind that AST recursively

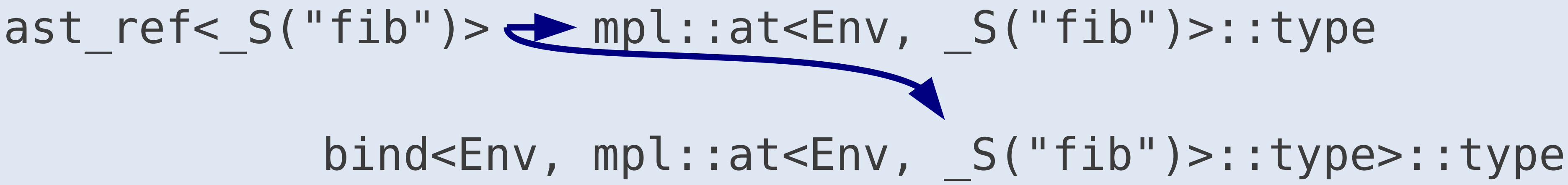
Building the DSL

- The rules of binding have changed
  - We assumed that there is a value in the Env map
  - Now there is an AST instead
  - We need to bind that AST recursively

```
ast_ref<_S("fib")> → mpl::at<Env, _S("fib")>::type
```

Building the DSL

- The rules of binding have changed
  - We assumed that there is a value in the Env map
  - Now there is an AST instead
  - We need to bind that AST recursively



```
ast_ref<_S("fib")>  →  mpl::at<Env, _S("fib")>::type

            bind<Env, mpl::at<Env, _S("fib")>::type>::type
```

Building the DSL

- Define functions as well
  - We need a way to describe functions in the AST
  - Lambda abstraction

Building the DSL

- Define functions as well
  - We need a way to describe functions in the AST
  - Lambda abstraction

```
template <class F, class ArgName>
struct ast_lambda;
```

Building the DSL

- Define functions as well
  - We need a way to describe functions in the AST
  - Lambda abstraction

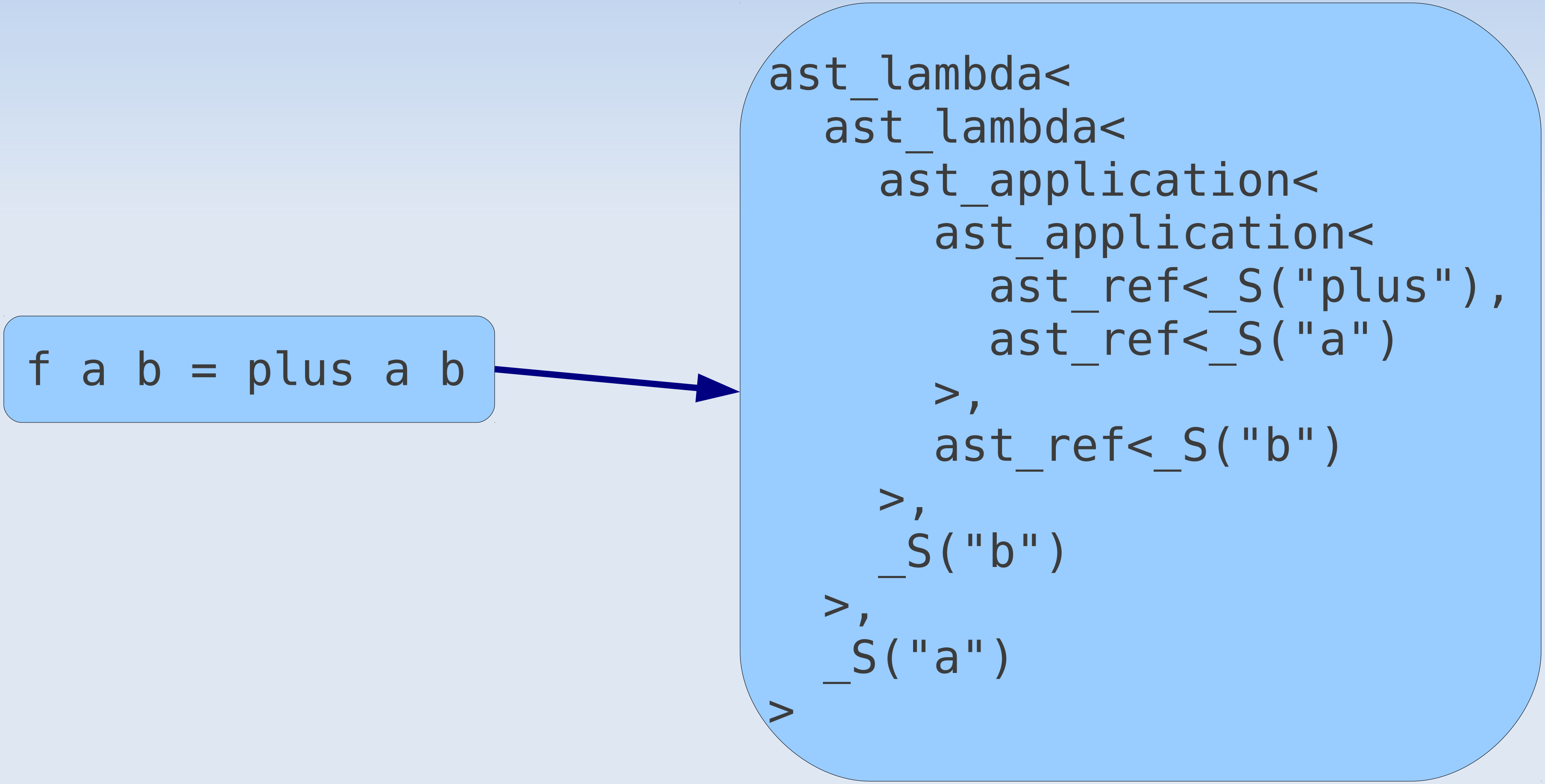
```
template <class F, class ArgName>
struct ast_lambda;
```

```
definition ::= name_token+ '=' application
```

Building the DSL

```
f a b = plus a b
```

Building the DSL



```
f a b = plus a b
```

```
ast_lambda<
  ast_lambda<
    ast_application<
      ast_application<
        ast_ref<_S("plus"),
        ast_ref<_S("a")
      >,
      ast_ref<_S("b")
    >,
    _S("b")
  >,
  _S("a")
>
```

Building the DSL

- Binding a lambda abstraction is tricky
  - The value we bind the lambda argument to is not known until runtime
  - The value of the parameter will always be a value (not an AST)

Building the DSL

- Binding a lambda abstraction is tricky
  - The value we bind the lambda argument to is not known until runtime
  - The value of the parameter will always be a value (not an AST)
  - bind builds a metafunction class storing the Env (closure)

Building the DSL

- Binding a lambda abstraction is tricky
  - The value we bind the lambda argument to is not known until runtime
  - The value of the parameter will always be a value (not an AST)
  - bind builds a metafunction class storing the Env (closure)
    - It expects one argument (the lambda argument)
    - When it is called:
      - It does the binding
      - It evaluates the result of binding

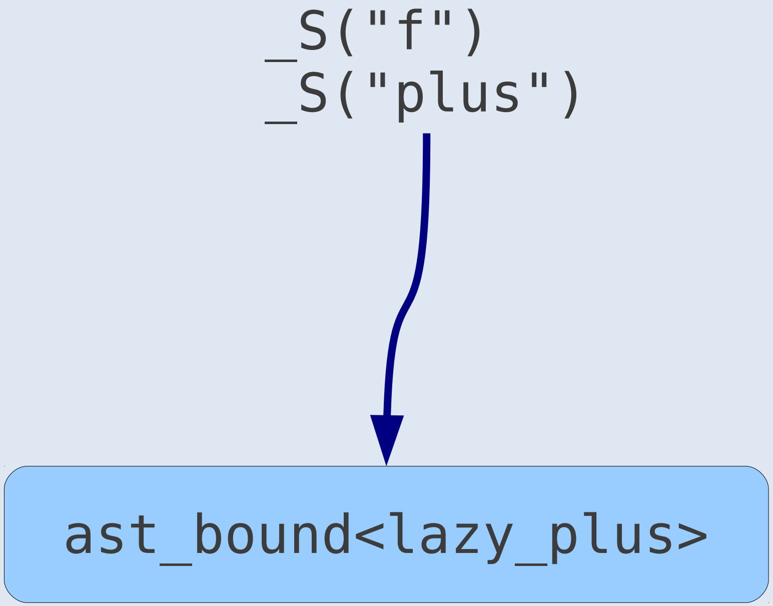
Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::define<_S("f a b = plus a b")>::type;
```

```
_S("f")
_S("plus")
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::define<_S("f a b = plus a b")>::type;
```

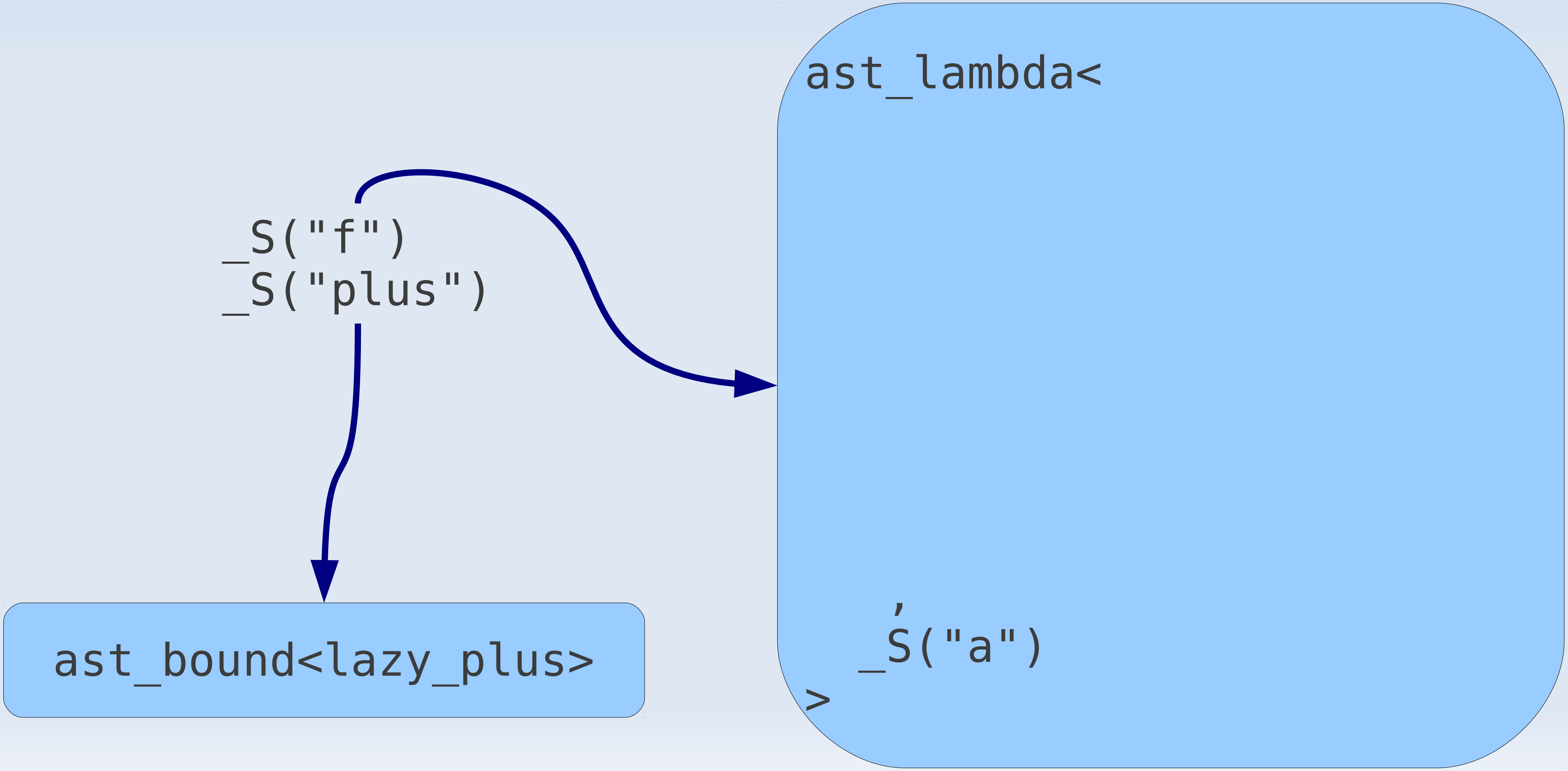


```
_S("f")
_S("plus")
```

```
ast_bound<lazy_plus>
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::define<_S("f a b = plus a b")>::type;
```



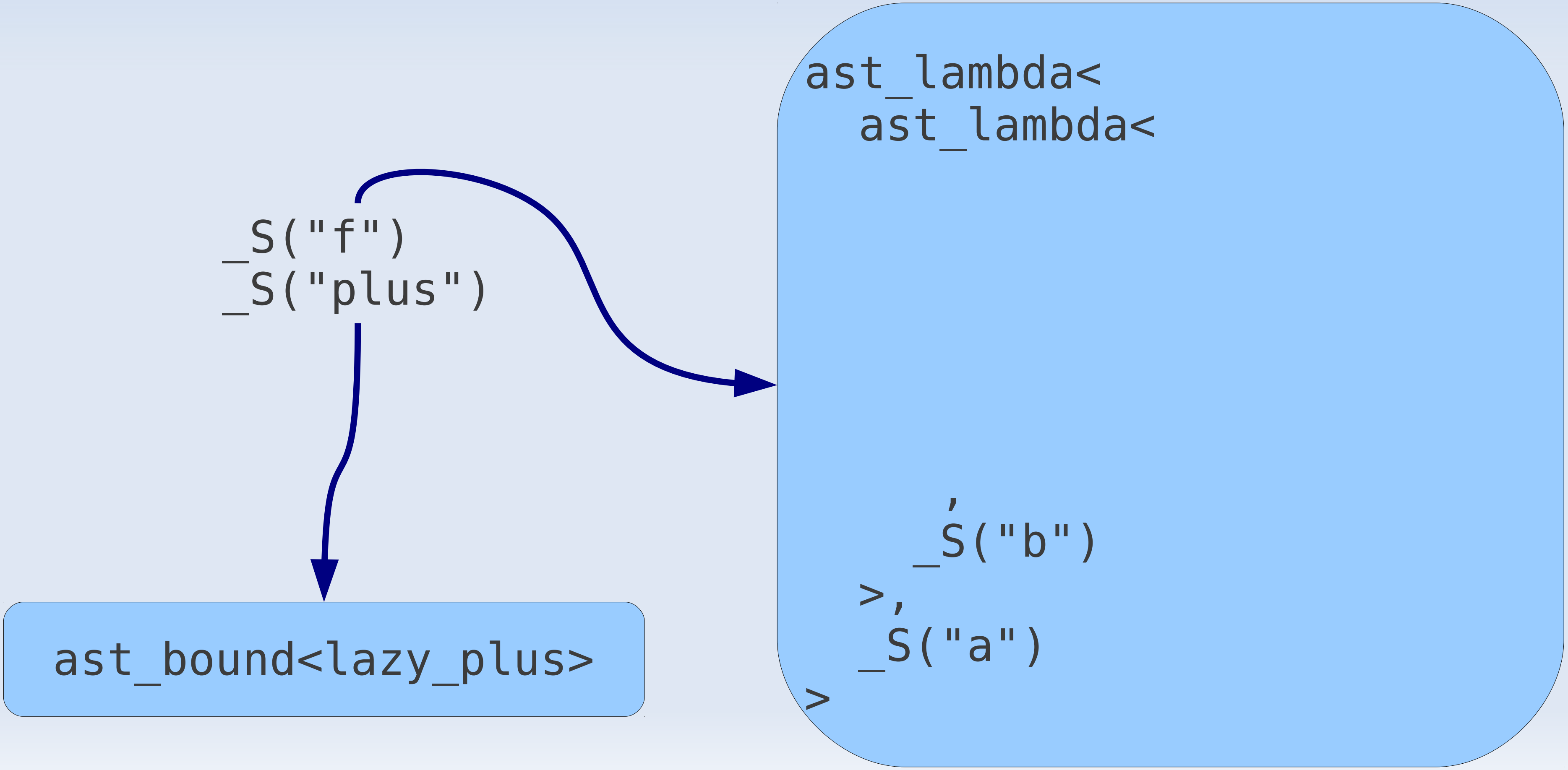
```
_S("f")
_S("plus")
```

```
ast_bound<lazy_plus>
```

```
ast_lambda<



    ,
   _S("a")
>
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::define<_S("f a b = plus a b")>::type;
```



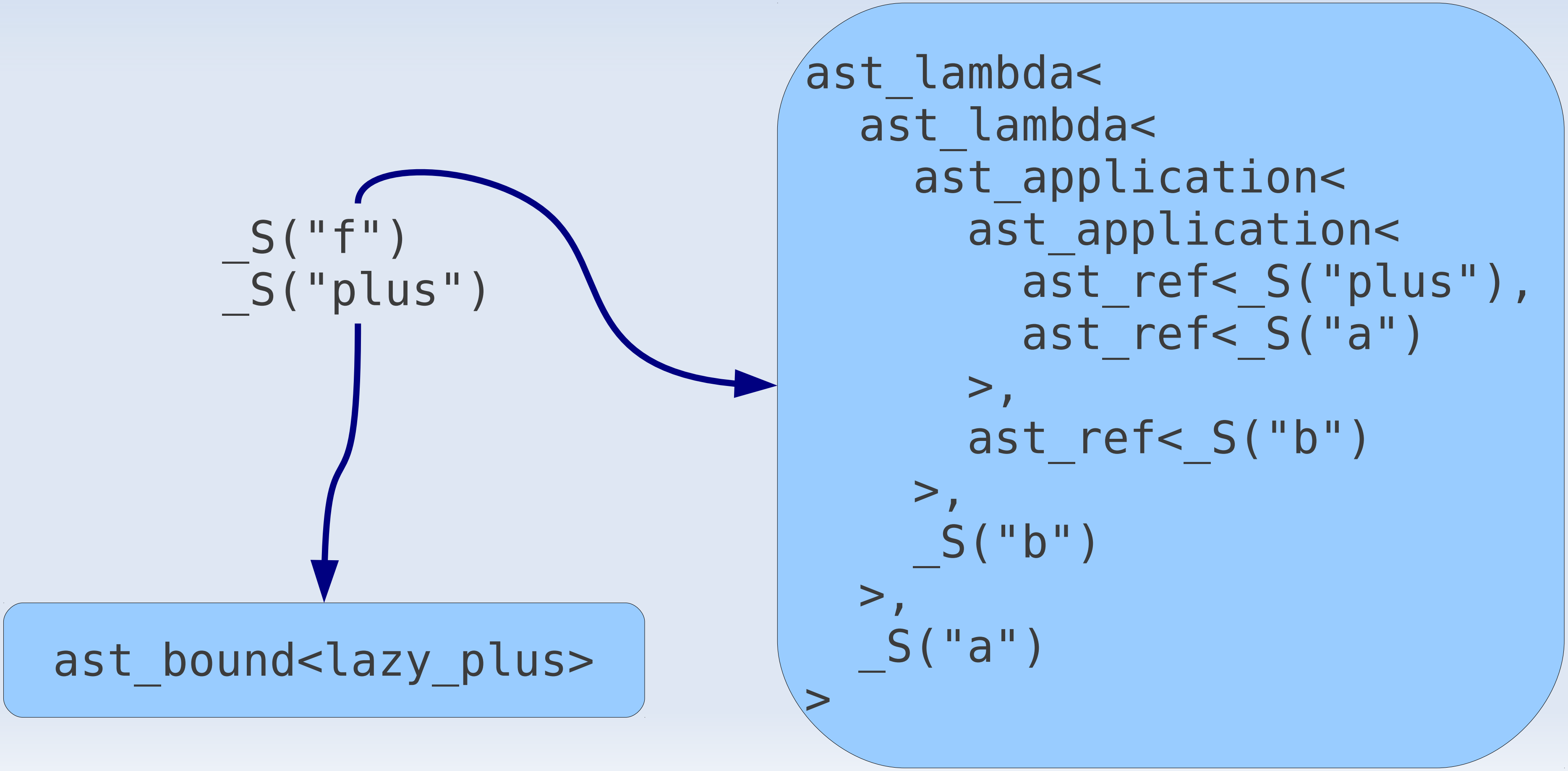
```
_S("f")
_S("plus")
```

```
ast_lambda<
  ast_lambda<



     ,
    _S("b")
  >,
  _S("a")
>
```

```
ast_bound<lazy_plus>
```

Building the DSL

```
meta_hs
  ::import2<_S("plus"), lazy_plus>::type
  ::define<_S("f a b = plus a b")>::type;
```



```
_S("f")
_S("plus")
```

```
ast_bound<lazy_plus>
```

```
ast_lambda<
  ast_lambda<
    ast_application<
      ast_application<
        ast_ref<_S("plus"),
        ast_ref<_S("a")
      >,
      ast_ref<_S("b")
    >,
    _S("b")
  >,
  _S("a")
>
```

Building the DSL

- Operators can be added
  - `a + b → plus a b`
  - `a – b → minus a b`
  - …

Building the DSL

- Operators can be added
  - `a + b → plus a b`
  - `a – b → minus a b`
  - `…`
- They can be added to `meta_hs`

```
typedef
  builder<mpl::map<>>
    ::import<_S("plus"), lazy_plus>::type
    ::import<_S("minus"), lazy_minus>::type
    // …
  meta_hs;
```

Building the DSL

```
meta_hs
  ::import3<_S("if_"), lazy_eval_if>::type
```

```
template <class C, class T, class F>
struct lazy_eval_if :
  mpl::eval_if<typename C::type, T, F>
{};
```

Building the DSL

```
meta_hs
  ::import3<_S("if_"), lazy_eval_if>::type

  ::define<_S("fact n = if_ (n == 0) 1 (n * fact (n–1))");
```

```
template <class C, class T, class F>
struct lazy_eval_if :
  mpl::eval_if<typename C::type, T, F>
{};
```

Building the DSL

```
meta_hs
  ::import3<_S("if_"), lazy_eval_if>::type

  ::define<_S("fact n = if_ (n == 0) 1 (n * fact (n–1))");
```

```
fact n = if n == 0 then 1 else n * fact (n-1)
```

```
template <class C, class T, class F>
struct lazy_eval_if :
  mpl::eval_if<typename C::type, T, F>
{};
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n-1)")
    >::type
  fact_library;
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n–1)")
    >::type
  fact_library;
```

```
fact_library
  ::define<_S("f n = fact (fact n)")>::type;
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n–1)")
    >::type
  fact_library;
```

```
fact_library
  ::define<_S("f n = fact (fact n)")>::type;
```

```
fact_library
  ::define<_S("g n = 2 + fact n")>::type;
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n-1)")
    >::type

    ::get<_S("fact")>::type
  fact;
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n–1)")
    >::type

    ::get<_S("fact")>::type
  fact;
```

```
mpl::apply<fact, mpl::int_<3>>::type
```

Building the DSL

```
typedef
  meta_hs
    ::define<
      _S("fact n = if n == 0 then 1 else n * fact (n-1)")
    >::type

    ::get<_S("fact")>::type
  fact;
```

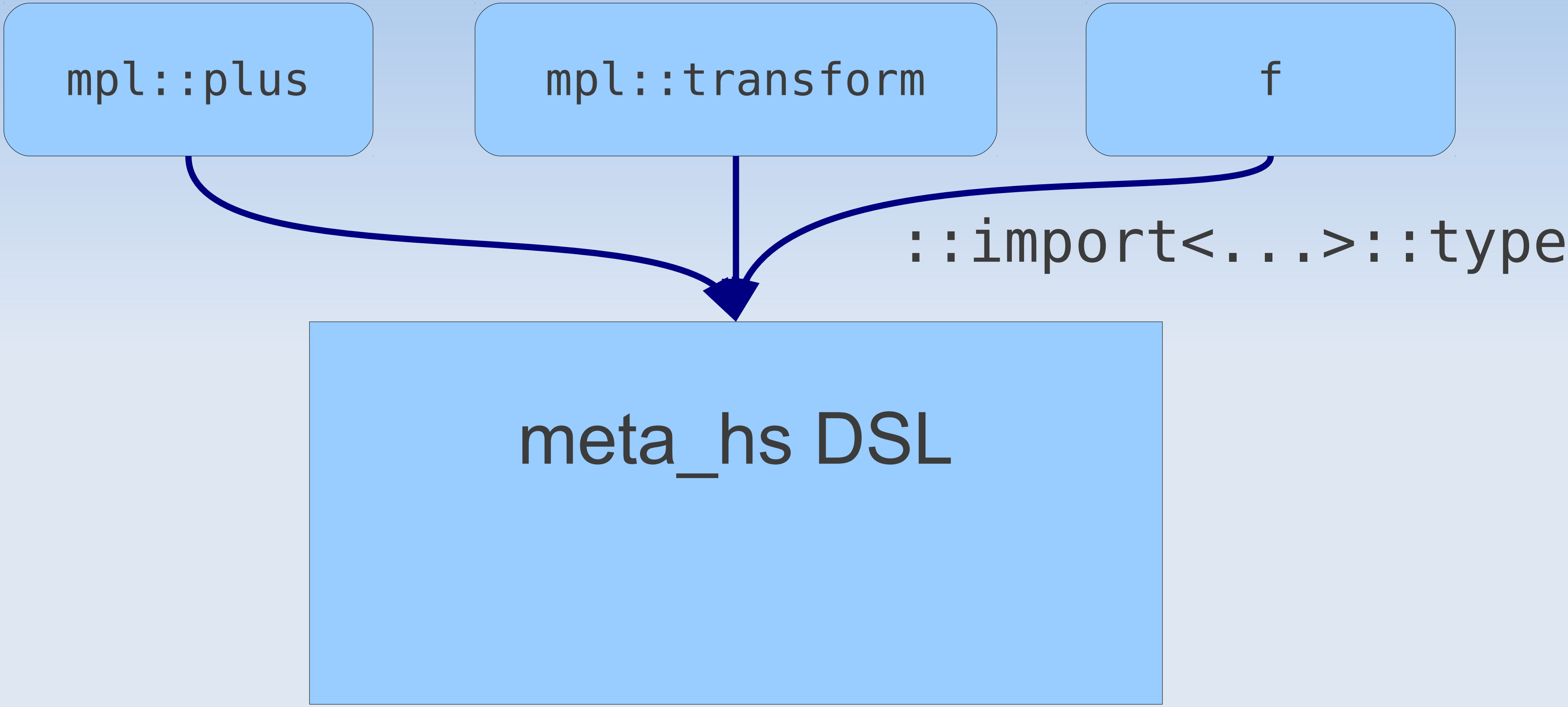
```
mpl::apply<fact, mpl::int_<3>>::type
```

```
mpl::transform<
  mpl::vector_c<int, 1, 2, 3, 4, 5>,
  fact
>::type
```

Building the DSL

meta_hs DSL

Building the DSL



mpl::plus

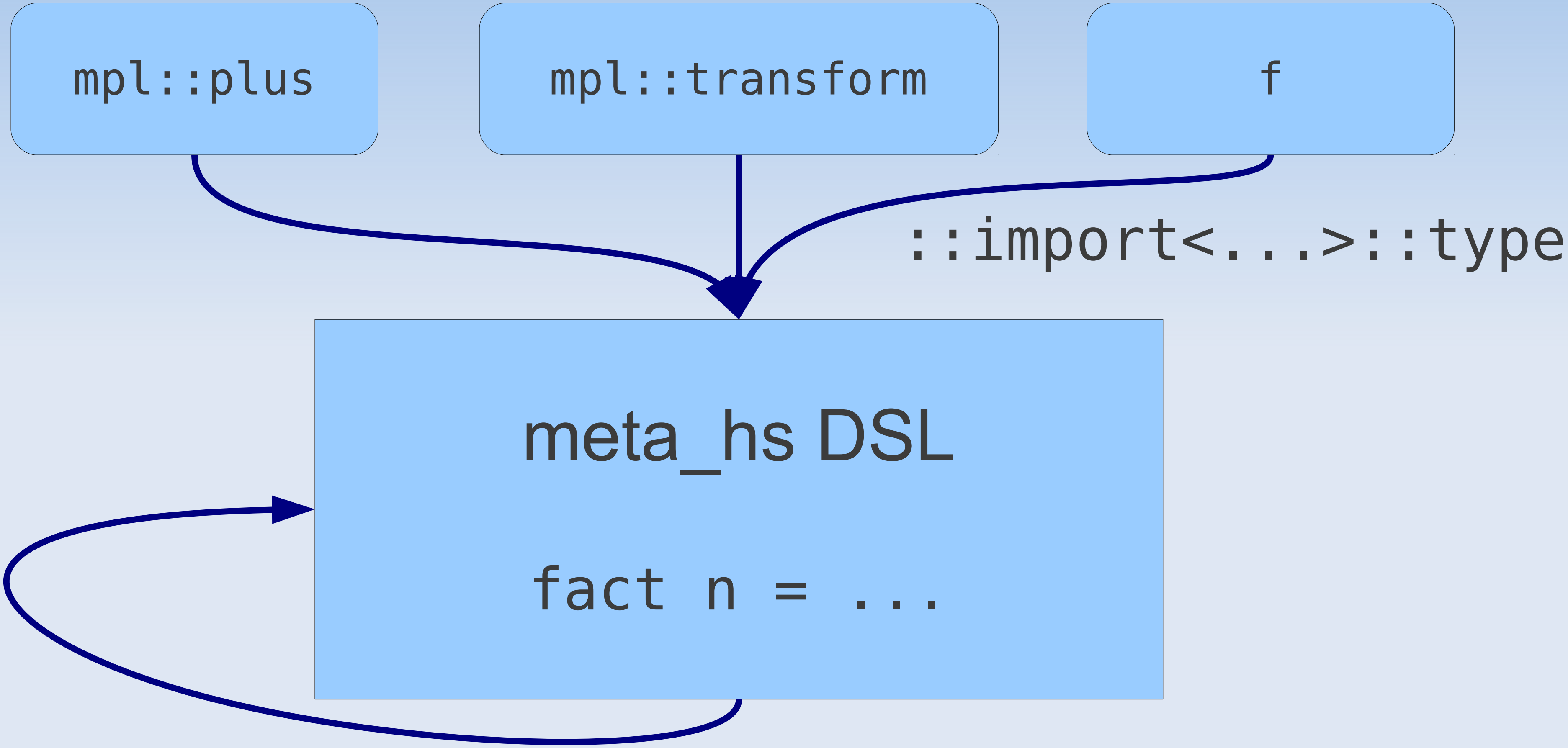
mpl::transform

f

::import<...>::type

meta_hs DSL

Building the DSL



mpl::plus

mpl::transform

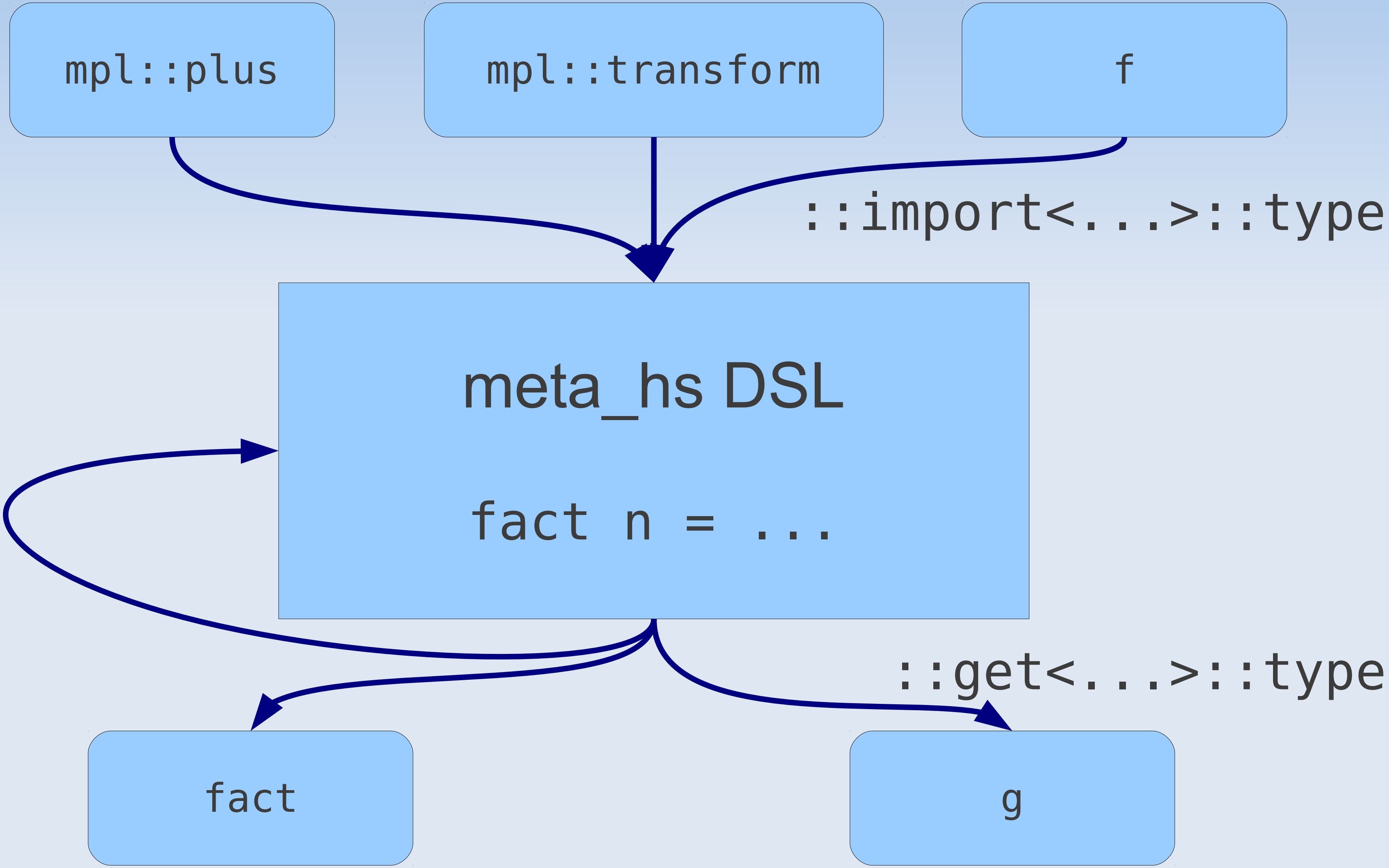
f

::import<...>::type

meta_hs DSL

fact n = ...

Building the DSL



mpl::plus

mpl::transform

f

::import<...>::type

meta_hs DSL

fact n = ...

::get<...>::type

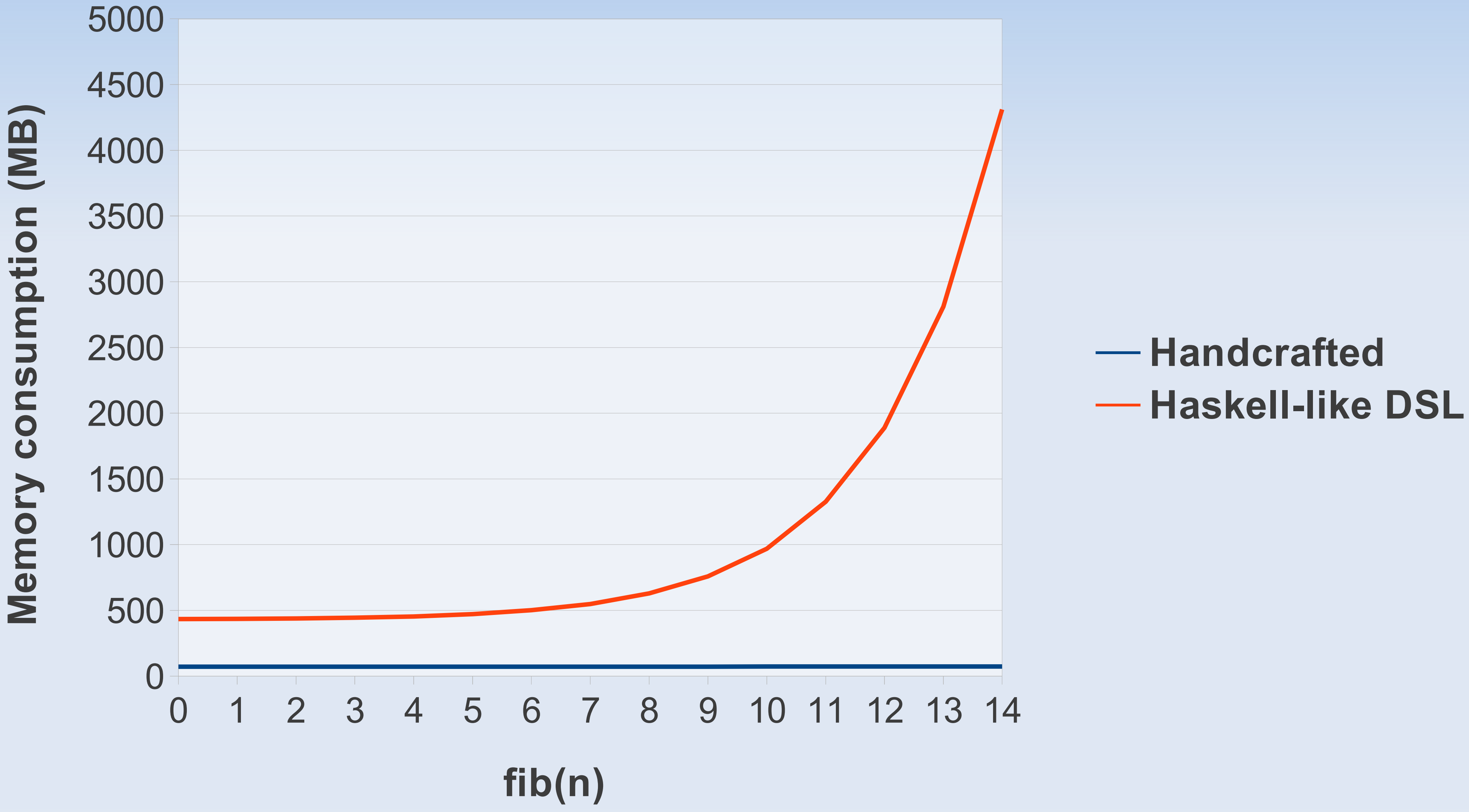
fact

g

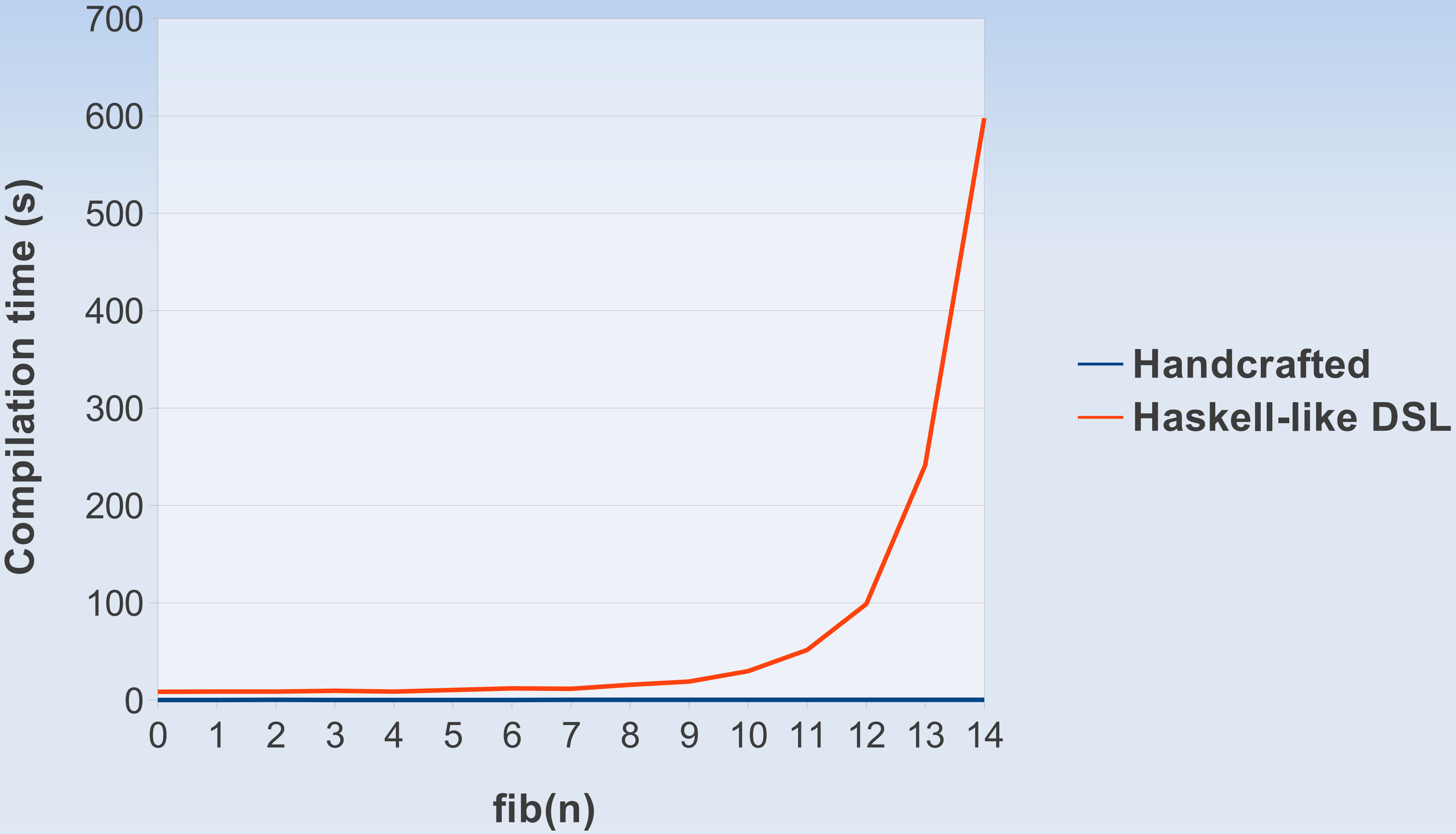
Performance

- Fibonacci
  - Handcrafted (based on Boost.MPL)
  - Generated
- Linux
- GCC 4.7, 64 bit (-std=c++0x)
- 1.6 GHz, 2 cores
- 2 GB memory

Memory usage



Compilation time



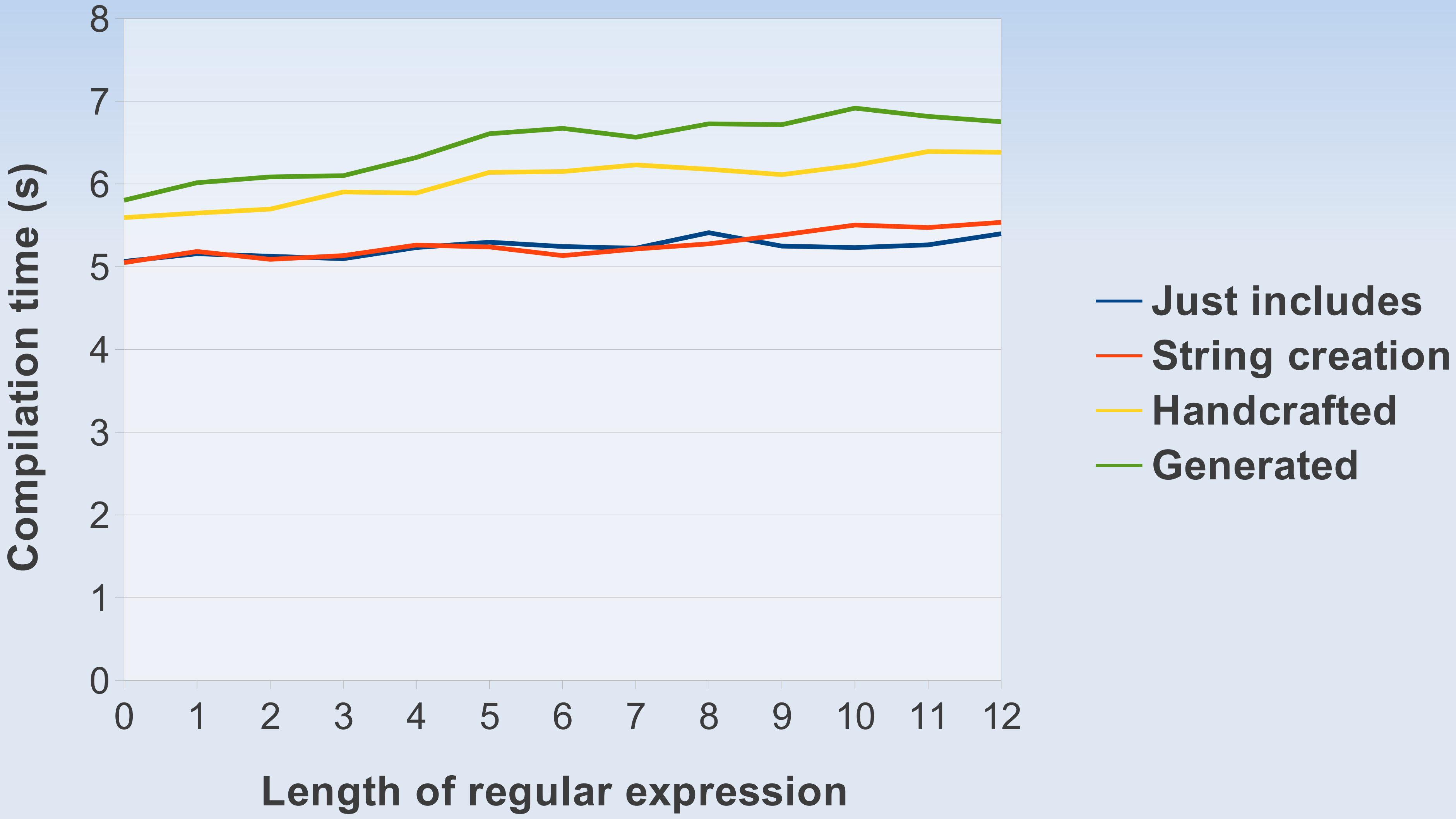
Regular expressions

- Syntactical sugar for Xpressive
- We generate code the could have been written using the original Xpressive interface
- Easy to measure the difference
- Where the costs are coming from?

Regular expressions

- Virtual Machine (VirtualBox 1.2)
- Host OS: Windows 7
- Guest OS: Linux
- g++ 4.6, -std=c++0x
- Memory: 1 GB
- Processor: Inter Core2 Duo, 2.53 GHz

Compilation time



Parser combinators

```
template <class P, class Pred, class Msg>
struct accept_when {
  template <class S, class Pos>
  struct apply :
    mpl::eval_if<
      typename is_error<mpl::apply<P, S, Pos>>::type,
      mpl::apply<P, S, Pos>,
      // check result of parsing...
    >
  {};
};
```

Parser combinators

```
template <class P, class Result>
struct always {
  template <class S, class Pos>
  struct apply :
    mpl::eval_if<
      typename is_error<mpl::apply<P, S, Pos>>::type,
      mpl::apply<P, S, Pos>,
      mpl::apply<return_<Result>, /* ... */ >
    >
  {};
};
```

Parser combinators

```
definition ::= name_token '=' application
```

```
struct definition {
  template <class S, class Pos>
  struct apply {










  };
};
```

Parser combinators

```
definition ::= name_token '=' application
```

```
struct definition {
  template <class S, class Pos>
  struct apply {
    typedef typename mpl::apply<
      sequence<name_token, define_token, application>,
      S, Pos
    >::type r;

  };
};
```

Parser combinators

```
definition ::= name_token '=' application
```

```
struct definition {
  template <class S, class Pos>
  struct apply {
    typedef typename mpl::apply<
      sequence<name_token, define_token, application>,
      S, Pos
    >::type r;

    typedef pair<
      typename mpl::front<typename get_result<r>::type>::type,
      typename mpl::back<typename get_result<r>::type>::type
    > type;
    // TODO: error propagation
  };
};
```

Parser combinators

```
typedef
       name_token,          name,
       define_token,        ignore,
       application,         body,
          mpl::pair<name, body>



  definition;
```

Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```

```
lambda<arg, body>
```

```
bind<parser, lambda_expression>
```

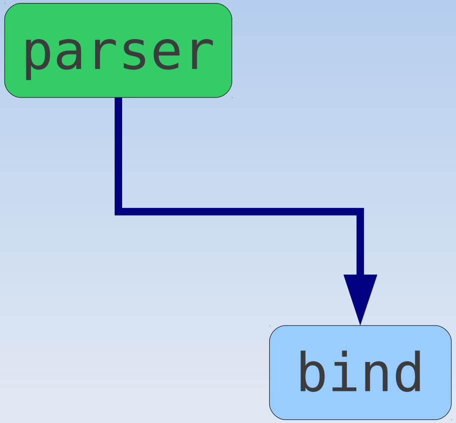
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```

bind

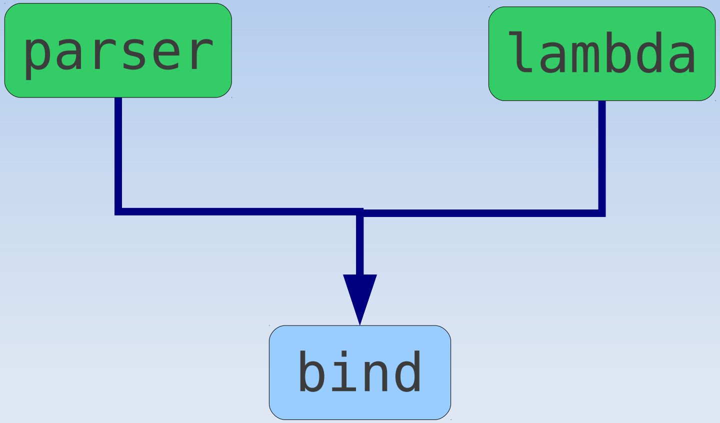
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



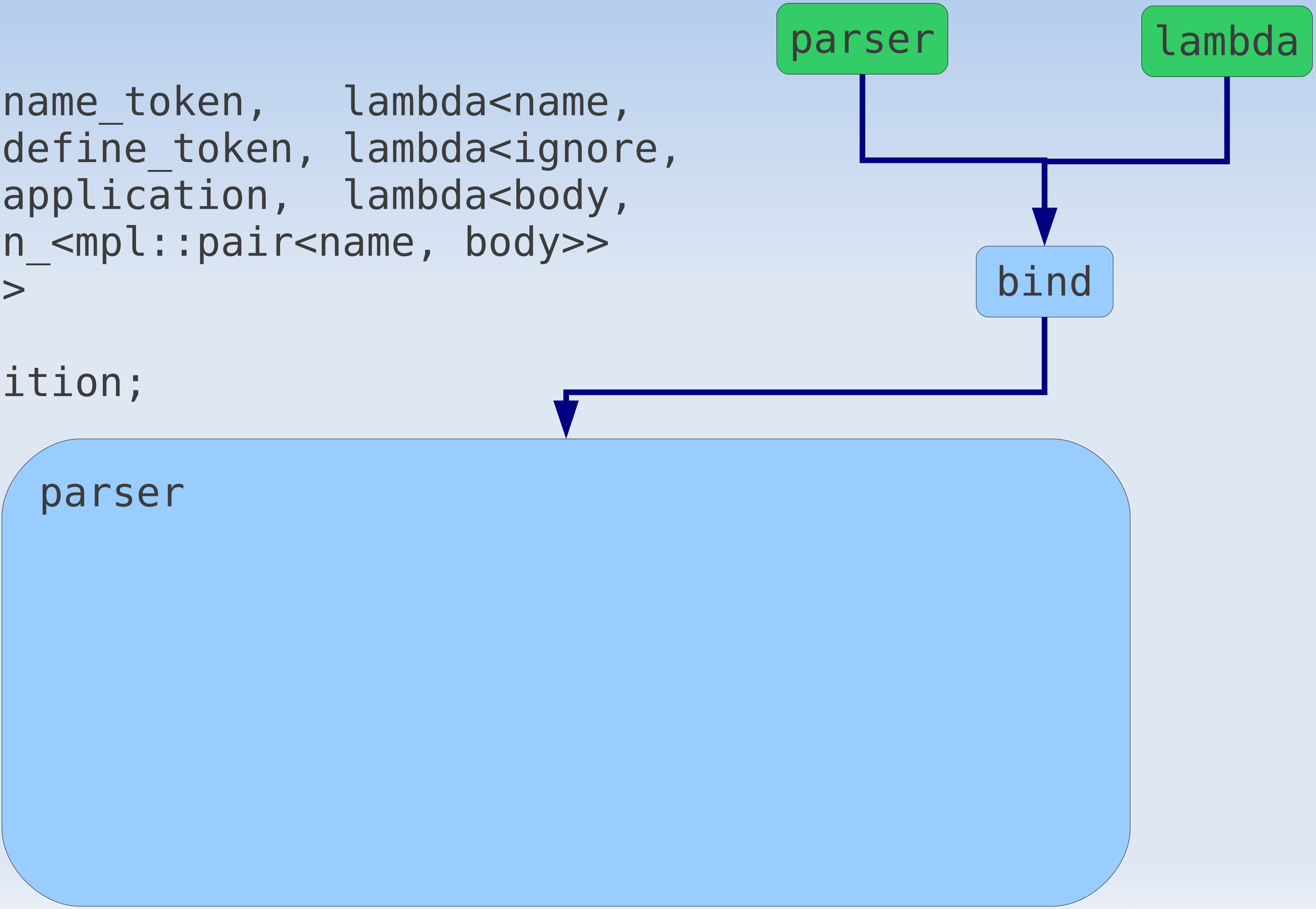
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



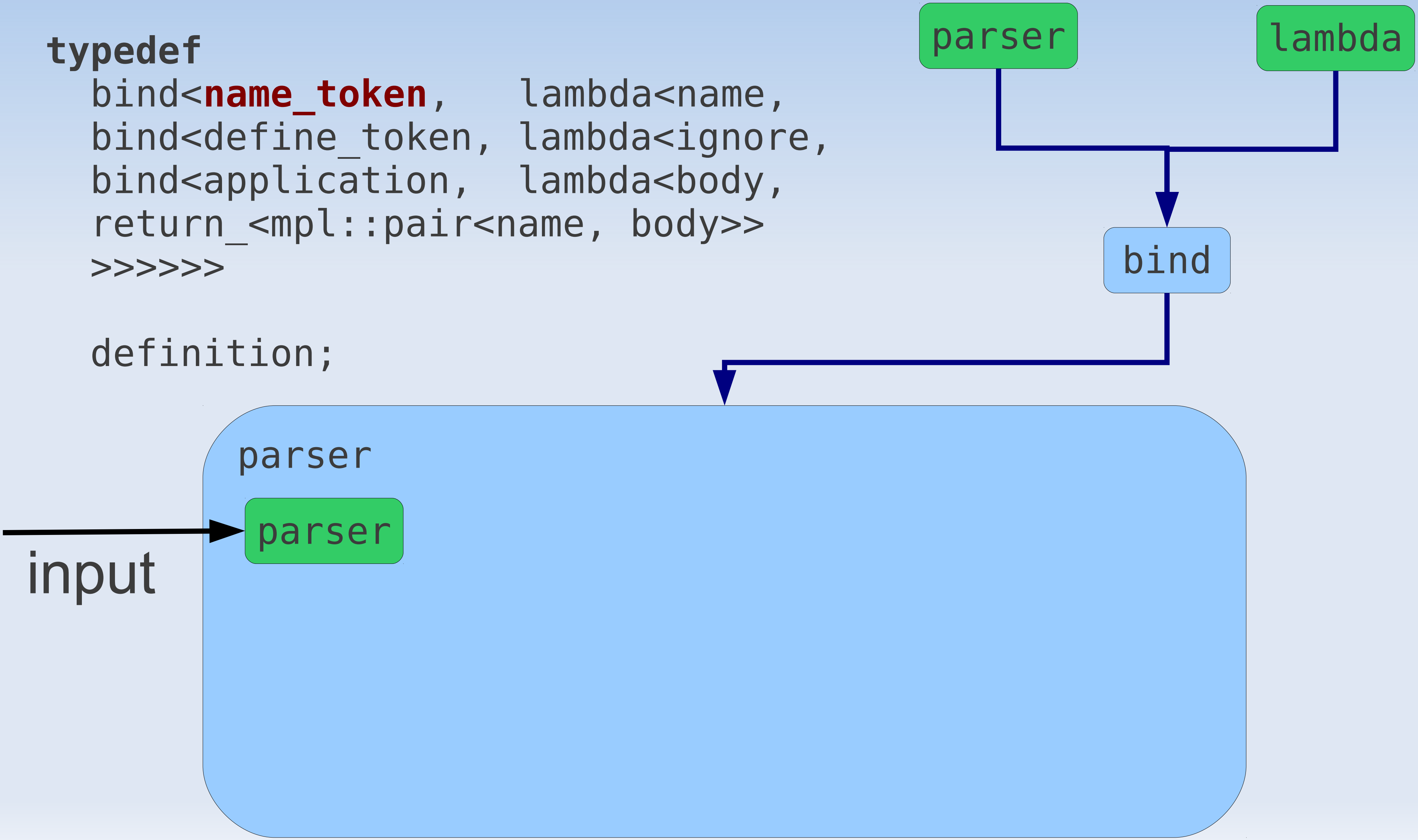
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



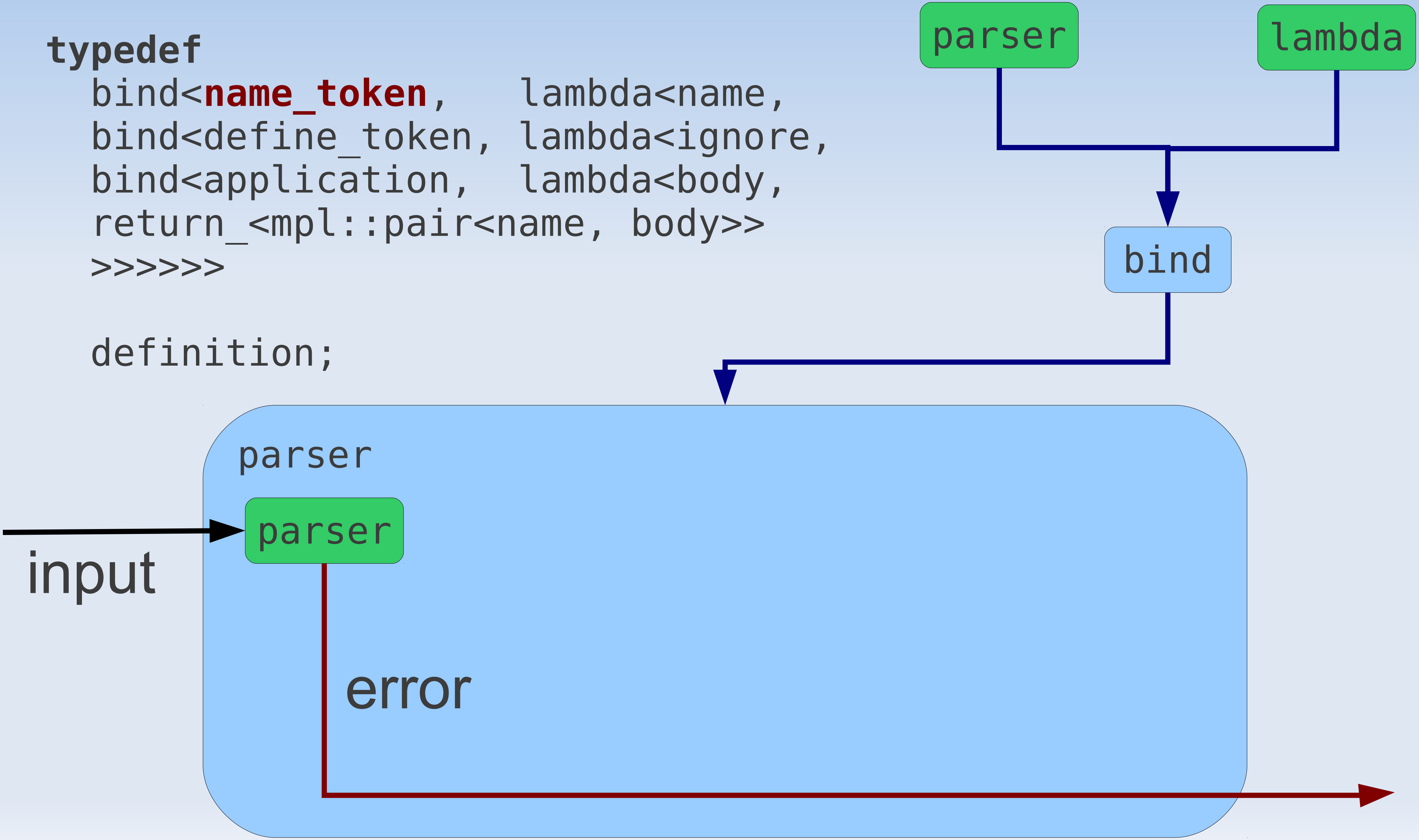
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



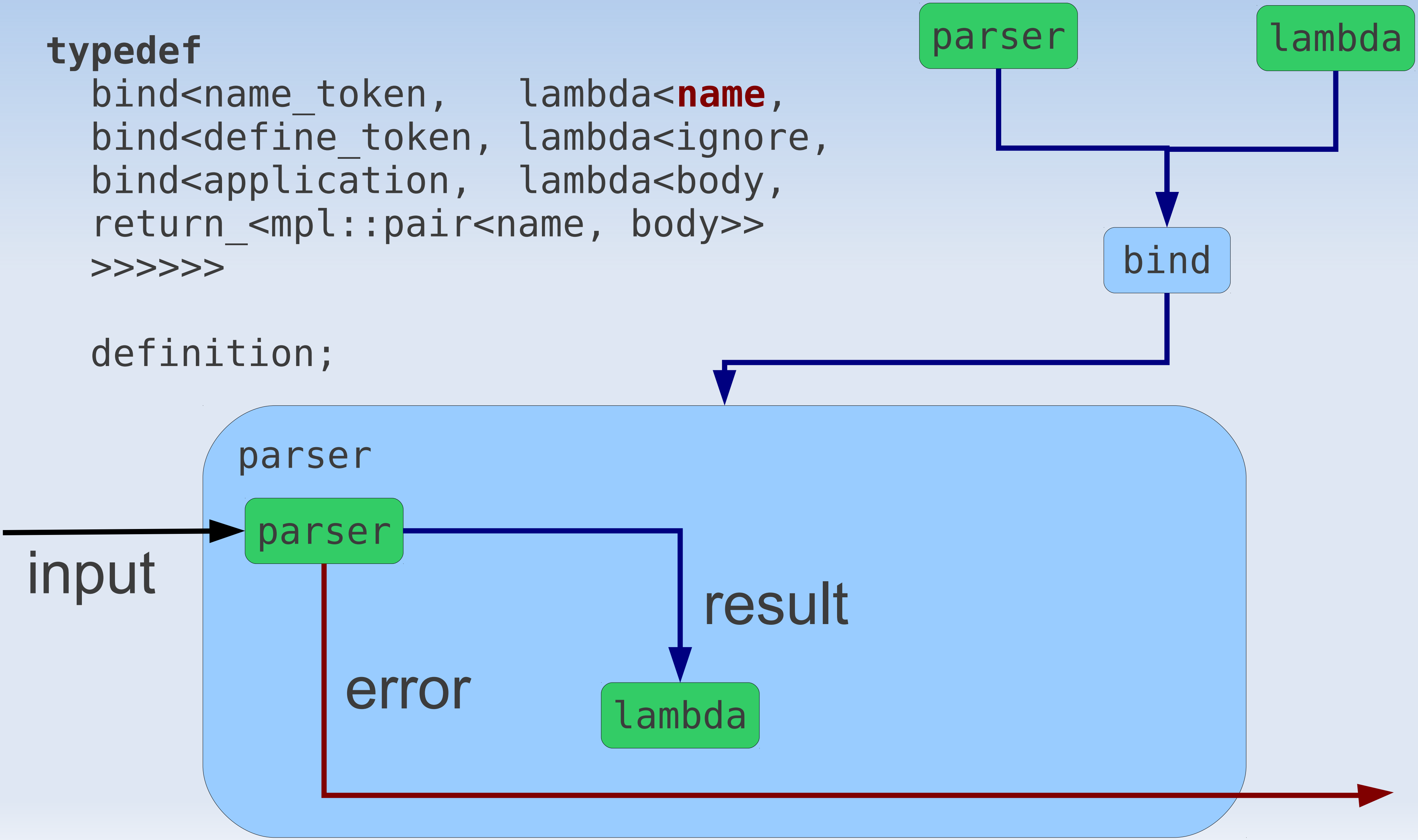
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



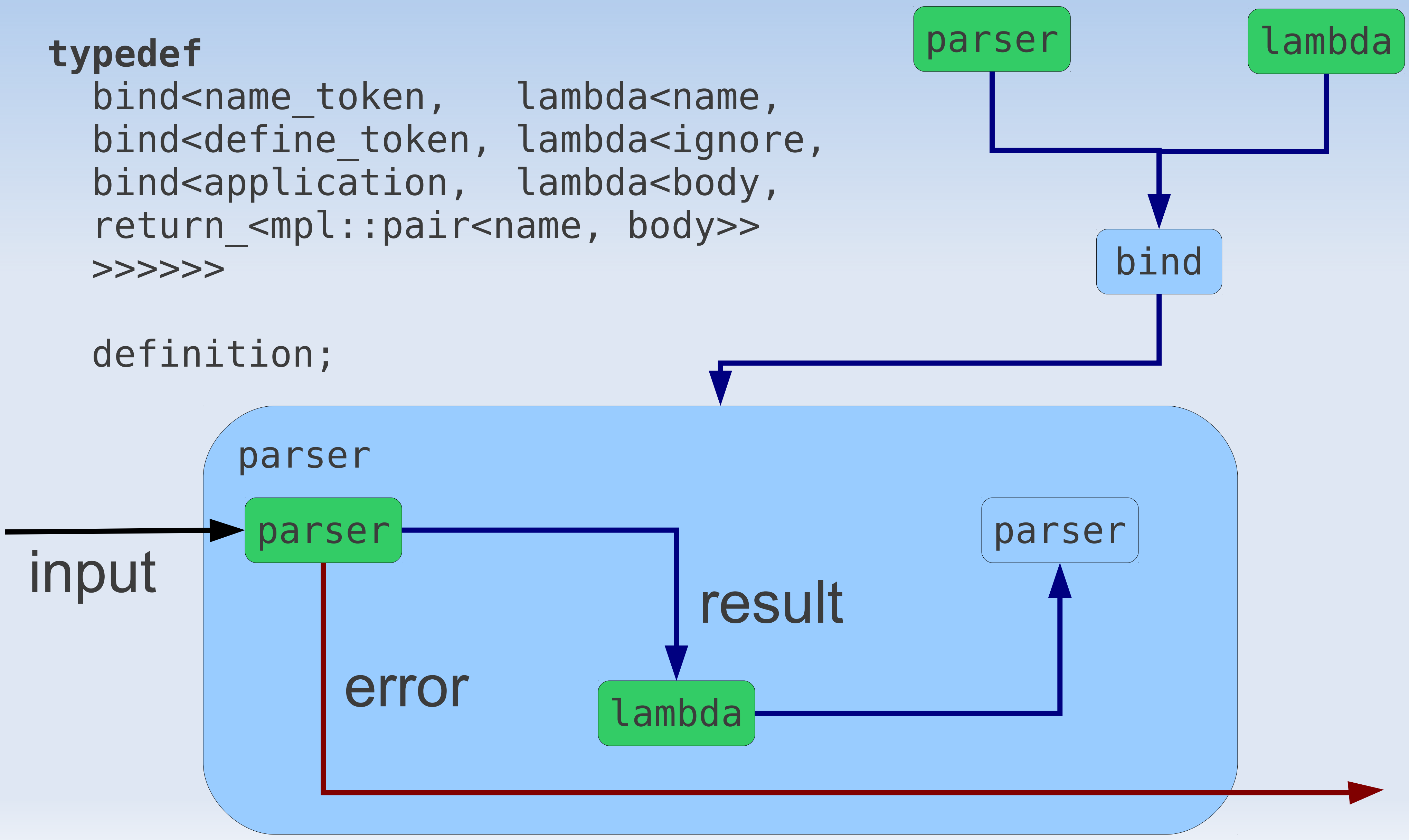
Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



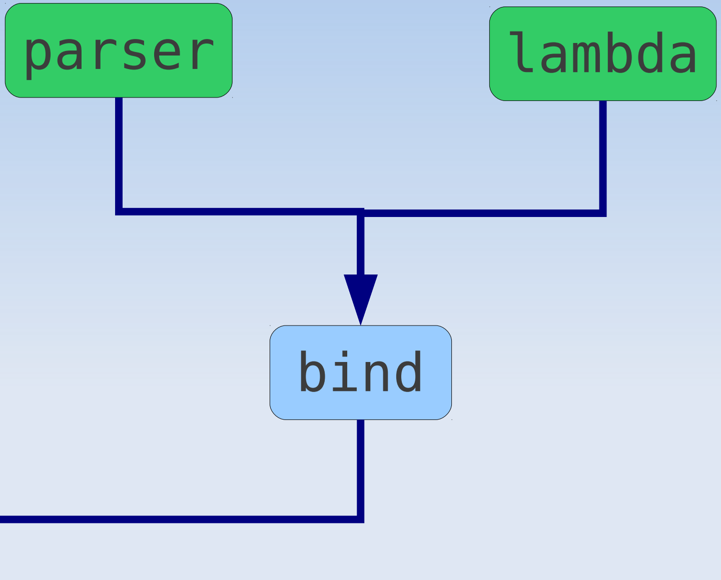
Parser combinators

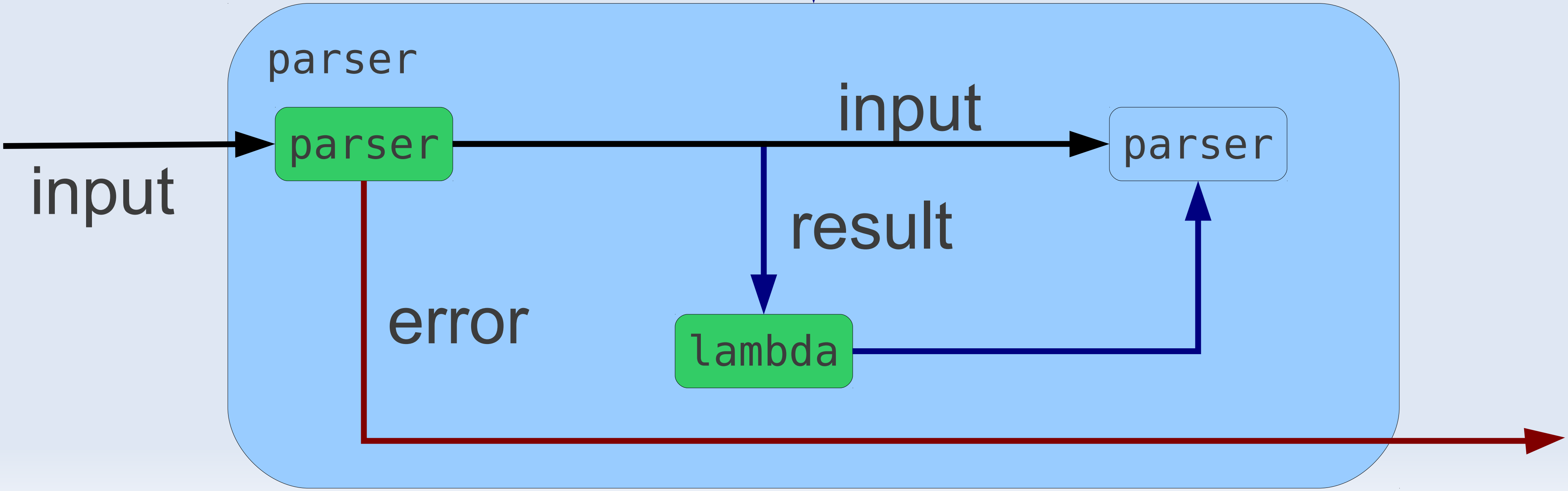
```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



Parser combinators

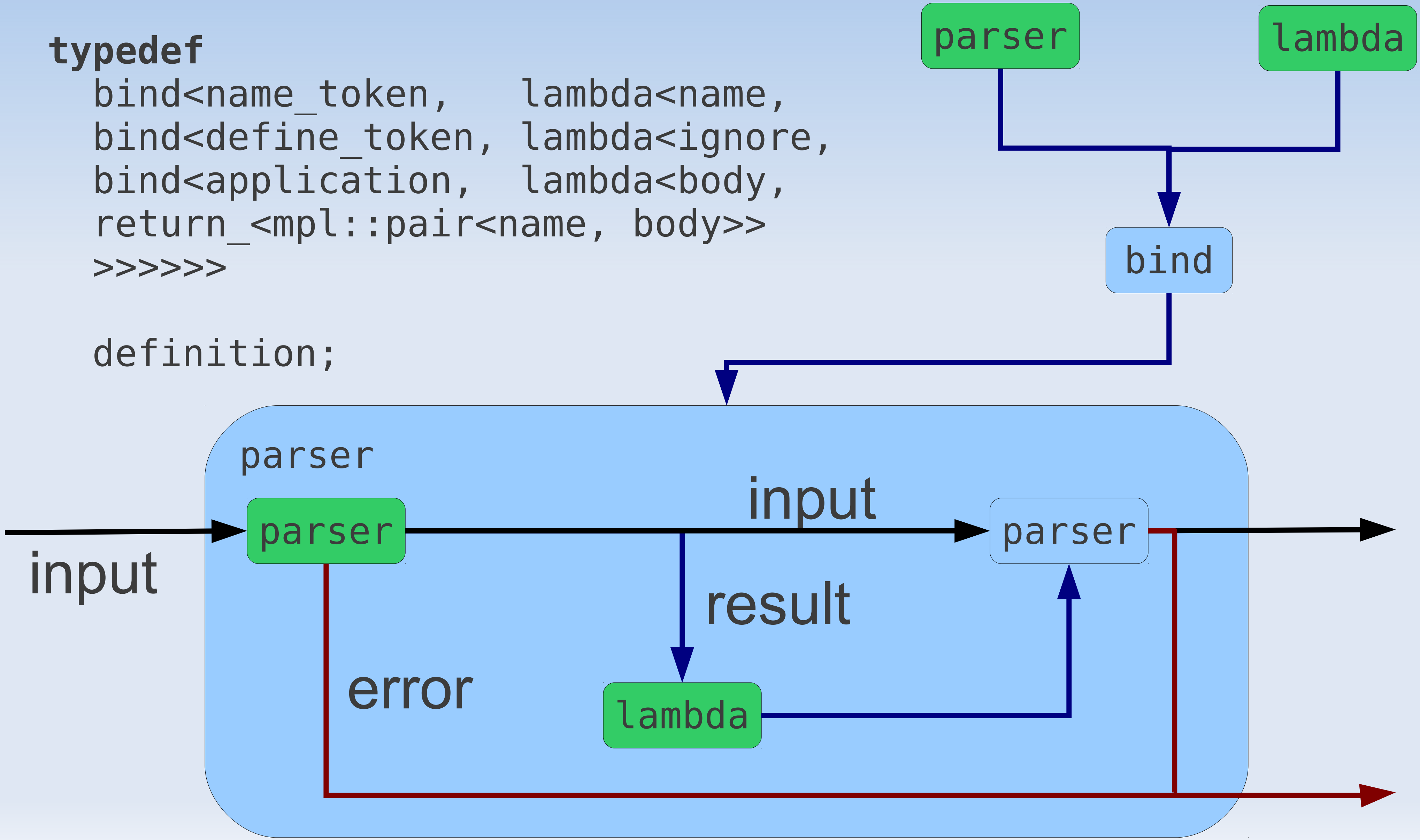
```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```





Parser combinators

```
typedef
  bind<name_token,   lambda<name,
  bind<define_token, lambda<ignore,
  bind<application,  lambda<body,
  return_<mpl::pair<name, body>>
  >>>>>>

  definition;
```



Parser combinators

- 2 + 1 operations

Parser combinators

- 2 + 1 operations
  - `fail    :: string → parser`

Parser combinators

- 2 + 1 operations
  - `return_ :: result → parser`
  - `fail    :: string → parser`

Parser combinators

- 2 + 1 operations
  - `bind    :: parser × (result → parser) → parser`
  - `return_ :: result → parser`
  - `fail    :: string → parser`

Parser monad

- 2 + 1 operations
  - `bind    :: parser × (result → parser) → parser`
  - `return_ :: result → parser`
  - `fail    :: string → parser`

Parser monad

- 2 + 1 operations
  - `bind    :: parser × (result → parser) → parser`
  - `return_ :: result → parser`
  - `fail    :: string → parser`
- Haskell's do notation

Do notation

```
typedef do_parser<
  set<name, name_token>,       // name <- name_token
  define_token,
  set<body, application>,     // body <- application

  return_<mpl::pair<name, body>>
>
  definition;
```

Summary

- Parsing at compile-time is useful for DSL embedding
- One can parse using template metaprograms
- Metaparse
  - Parser combinators
  - Monadic parsing
- Real world example: DSL for template metaprograms

Q & A

Mpllibs.Metaparse

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