Parametric Expressions

PROPOSED C++ LANGUAGE FEATURE
(P1221)

JASON RICE
C++NOW 2019

What are Parametric Expressions?

Motivating Use Cases

Crazy Library Stuff

What are Parametric Expressions?

Motivating Use Cases

Crazy Library Stuff

What are Parametric Expressions?

Motivating Use Cases

Crazy Library Stuff

What are Parametric Expressions?

Motivating Use Cases

Crazy Library Stuff

nonsense!?

$$f(\ldots X) = Y$$

expressions!

	Object	Expression
Typed	✓	?
Storable	✓	☁
In Final Program	?	?

Functions Considered Harmful

Never Go Full Functional


function
object...
object


parmexpr
(object|expr)...
expr


parmexpr
(object|expr)...
object

$$f(\ldots X) = Y$$

expressions!

```
using f(...X) = Y ;
```

```
using f(auto&&...X) = Y;
```

```
template <typename T>
using f( T&& ...X) = Y;
```

```
using f(...X) = Y ;
```

why not both?

Declaration Syntax

CONTINUED

```
using add(x, y) = x + y;
```

```
using add(constexpr x,
          constexpr y) = x + y;
```

Assignment Like Syntax

LIKE A TYPE ALIAS

No Type Specifier

Like... well it's like javascript or one of those

Don't worry!

The types are still there

More Abbreviated than Abreviated Function Template Declaration Syntax

Hide the `auto` so they can't complain!

Never Go Full Type Inference

Parameter Kinds

Default Behaviour

`auto&&`

```
using add(a, b) = a + b;

int i = 40;
int x = add(i++, 2);

// the same as

int i = 40;
int x = ({
  auto&  a = i++;
  auto&& b = 2;
  a + b; // result of expression
});
```

Constexpr Parameters

```
constexpr auto
```

```
using add(constexpr a,
          constexpr b) = a + b;

constexpr int i = 40;
constexpr int x = add(i, 2);

// the same as

constexpr int i = 40;
constexpr int x = ({
  constexpr auto a = i;
  constexpr auto b = 2;
  a + b; // result of expression
});
```

<no-specifier>	auto&&
constexpr	auto
using	*<no-type>*

Macro Parameters

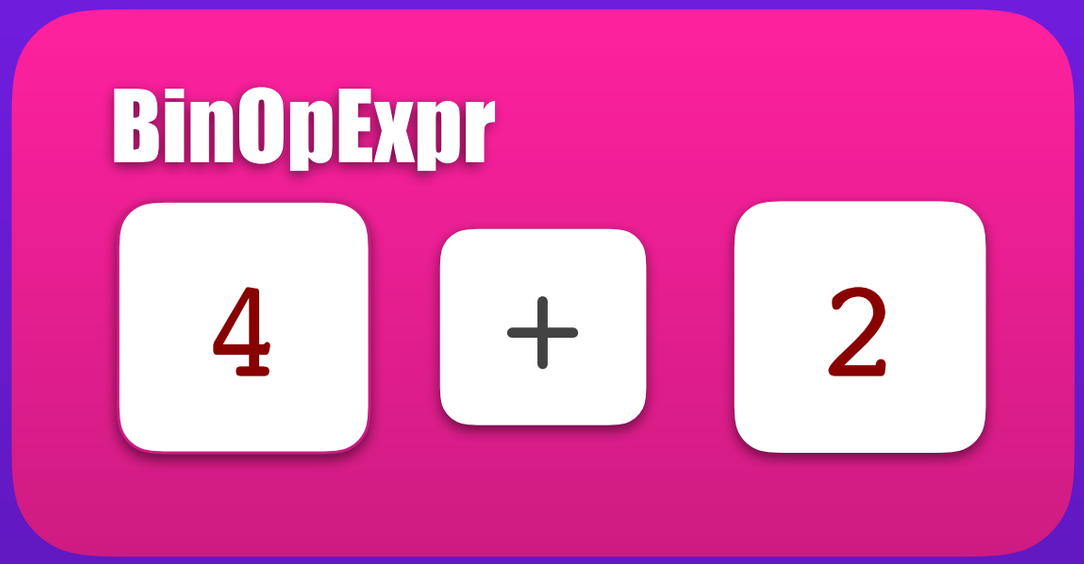
`using`

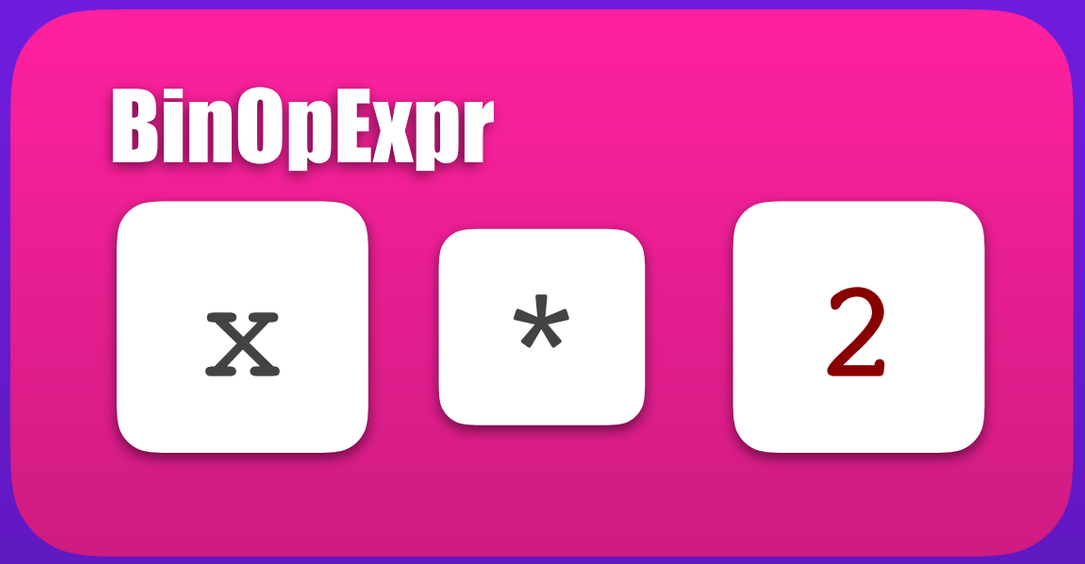
```
using add(using a,
          using b) = a + b;

int i = 40;
int x = add(i, 2);

// the same as

int i = 40;
int x = i + 2;
```

Substitution of Expression Semantics

NOT TEXT!

```
using twice(using auto x) = x * 2;

static_assert(twice(2) == 4);

static_assert(4 + 2 * 2 == 8);

static_assert((4 + 2) * 2 == 12);
static_assert(twice(4 + 2) == 12);
```


BinOpExpr
4
+
2


BinOpExpr
x
*
2

BinOpExpr

Does not Decay

literals stay literal

Original `id-expression` is replaced entirely!

VALUE CATEGORY OF INPUT EXPRESSION IS USED

Self Parameter

Deducing `this` (P0847)

```
struct foo {
  int x;
  using bar(this self, x) = self.x + x;
};
```

Member Expression

```
foo.bar(x)
```

Base Expression

Deducing `this` (P0847)

```
struct foo {
  int x;
  using bar(using this self, x) = self.x + x;
};
```

Deducing `this` (P0847)

```
struct foo {
    int x;
    using bar(constexpr this self, x) = self.x + x;
};
```

Deducing `this` (P0847)

```
struct foo {
    int x;
    using bar(this constexpr self, x) = self.x + x;
};
```

Deducing ~~this~~ (P0847)

```
struct foo {
  int x;
  static using bar(x, y) = x + y;
};
```

Operator Overloading

```
namespace mine {
  using operator==(x, y) = hana::equal(fwd(x), fwd(y));
}
```

```
namespace mine {
    using operator==(using x, using y) = hana::equal(x, y);
}
```

Participates in Operator Overloading

Members $\cup$ Non-Members + ADL $\cup$ Builtins

```
struct id_fn {
  using operator()(this self, x) = x;
};

int x = id_fn{}(42);
```

Static `operator()` (P1169)

```
struct id_fn {
  static using operator()(x) = x;
};

int x = id_fn{}(42);
```

Not Restricted

ANY MEMBER PARMEXPR CAN BE DECLARED
static

Transparent Context

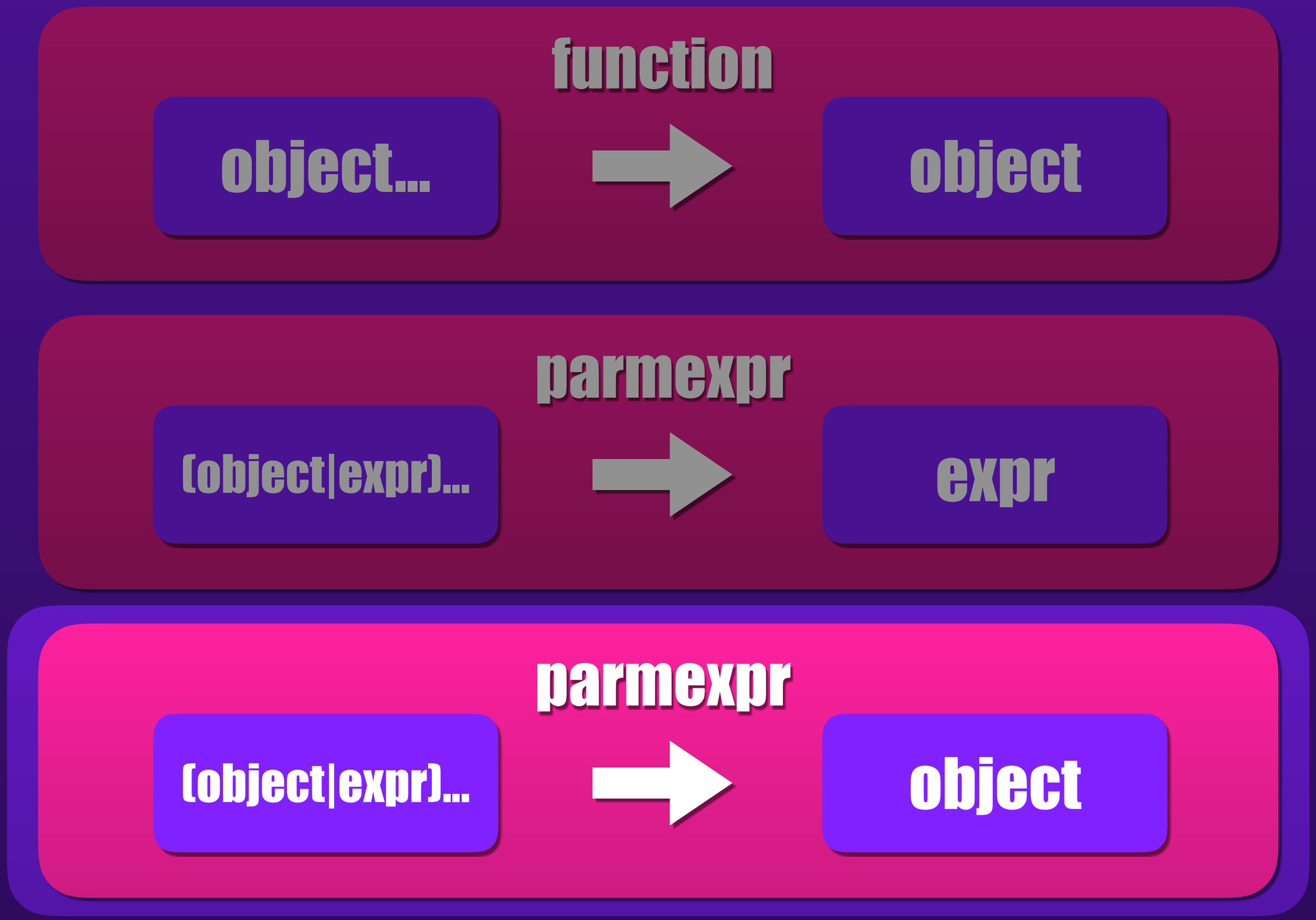
```
using twice(using x) = x * 2;

template <typename T,
          typename = decltype(twice(std::declval<T>()))>
auto foo(T x) {
  return twice(x);
}

auto foo(...) {
  return "not implemented";
}
```

SFINAE FRIENDLY

TAKEN FROM HTTP://LDIONNE.COM/P1221-PRESENTATION

RAII Scope

COMPOUND STATEMENT


function
object...
object
parmexpr
(object|expr)...
expr
parmexpr
(object|expr)...
object

```
using add(a, b) = a + b;

using add(a, b) { return a + b; }
```

```
using if_(cond, a, b) {
  if (cond) {
    return a;
  } else {
    return b;
  }
}
```

```
using if_(cond, a, b)
  -> decltype(auto)
{
  if (cond) {
    return a;
  } else {
    return b;
  }
}
```

```
using if_(cond, a, b)
    -> auto
{
    if (cond) {
        return a;
    } else {
        return b;
    }
}
```

```
using if_(cond, a, b) = cond ? a : b;
```

```
using constexpr_if(using cond, using a, using b) {
  if constexpr(cond) {
    return a;
  } else {
    return b;
  }
}
```

Parameter Packs

Expression Kinds

Expression

Expression Containing Unexpanded Parameter Packs

Does Not Work!

```
using to_pack(...x) = x;
```

Does Not Work!

```
using to_pack(...x) = x;

int sum = (to_pack(1, 2, 3) + ...);
assert(sum == 6);
```

Does Not Work!

```
using to_pack(...x) = x;

int sum = (to_pack(1, 2, 3) + ...);
assert(sum == 6);

[](auto x) {
  return (to_pack(x) + ...);
};
```

Does Not Work!

```
error: pack expansion does not contain any unexpanded parameter packs
   return (to_pack(x) + ...);
                  ~     ^
```

Postfix Tilde

```
using to_pack(...x)~ = x;
```

Must Contain a Pack!

```
using to_pack(...x)~ = x;

int sum = (to_pack(1, 2, 3)~ + ...);
assert(sum == 6);
```

Contains a Pack!

```
[](auto x) {
  return (to_pack(x)~ + ...);
};
```

Contains a Pack!

```
struct foo {
  int member1;
  int member2;

  using operator()~(using this self) =
    to_pack(self.member1, self.member2)~;
};
```

Can be `using` param

```
foo xs = {4, 2};
assert((xs~ + ...) == 6);
```

unary postfix tilde expands to pack

```
assert((foo{4, 2}~ + ...) == 6);
```

operand is subject to duplicate evaluation

Overload for Structured Bindings?

Overload For Structured Bindings

```
template <auto ...i>
struct integer_sequence {
  using operator()~(this x) = i;
  // ...
};
```

Haven't tried this!

Structured Bindings Can Introduce a Pack (P1061)

```
std::tuple f();
auto [x,y,z] = f();
auto [...xs] = f();
```

Structured Bindings Can Introduce a Pack (P1061)

```
auto [x, ...rest, z] = f();
```

ill-formed: non-trailing pack. This is for consistency with, for instance, function parameter packs

Structured Bindings Can Introduce a Pack (P1061)

```
using drop_back(...rest, x)~ = rest;
```

Why Not?

Packs of Packs?

```
using iota(N)~ = /* ... */;

template <size_t N>
void baz() {
  return sum(iota(iota(N)~)~ ...);  // Uh-oh!
}
```

https://quuxplusone.github.io/blog/2019/02/11/tilde-notation-for-exploding-tuples/

Packs of Packs?

```
<source>:5:19: error: unary pack expression contains an unexpanded parameter pack
   return sum(iota(iota(N)~)~ ...);  // Uh-oh!
                  ^        ~
1 error generated.
```

https://quuxplusone.github.io/blog/2019/02/11/tilde-notation-for-exploding-tuples/

Overloading with Functions?

Not Supported

CURRENTLY

```
auto  id(int x) { return x; }
using id(x) = x;
```

```
error: redefinition of 'id' as different kind of symbol
using id(x) = x;
```

```
using id(x) = x;
using id(x) = x;
```

```
error: redefinition of parametric expression 'id'
using id(x) = x;
^
```

Theoretically Possible

to overload with functions

Motivating Use Cases

Constexpr Ternary

```
using constexpr_if(constexpr auto cond,
                   using auto x,
                   using auto y) =
  constexpr_if_<cond>::apply(x, y);

auto x = constexpr_if(true, int{42}, std::make_unique<int>(42));
                                     // ^ not evaluated
```

Concise Forwarding

```
#define fwd(x) static_cast<decltype(x)>(x)


auto old_f = [](auto&& x) { use(std::forward<decltype(x)>(x)); };
auto new_f = [](auto&& x) { use(fwd(x)); };
```

Concise Forwarding

```
using fwd(using x) = static_cast<decltype(x)>(x);


auto old_f = [](auto&& x) { use(std::forward<decltype(x)>(x)); };
auto new_f = [](auto&& x) { use(fwd(x)); };
```

Concise Forwarding

```
template <typename X>
decltype(auto) id(X&& x)
  noexcept(noexcept(std::forward<X>(x)))
  -> decltype(std::forward<X>(x))
{
  return std::forward<X>(x);
}
```

Concise Forwarding

```
using id(using x) = x;
```

Lazy Evaluation

Debug Logging

```
namespace log {
  struct null_stream {
    using operator<<(using) = null_stream{};
  };

  using warn() {
    if constexpr (log_level == Warning)
      return std::wcerr;
    else
      return null_stream{};
  }
}

if (foo > 42)
  log::warn() << "Foo is too large: " << foo << '\n';
```

Disjunction

```
#include <string>

struct foo {
  std::string value;
  using operator||(this self, using x)
    = (self.value.length() > 0) ? self : x;
};

foo x = {"hello"};
foo y = x || foo{"world"};
          // ^ lazily constructed
```

Conjunction

```
#include <iostream>
#include <string_view>

struct guard {
  using operator&&(this self, using LAZY_X) {
    if (!self.is_stopped) {
      decltype(auto) x = LAZY_X;
      std::cout << x << '\n';
      if (std::string_view(x) == std::string_view("stop"))
        self.is_stopped = true;
    }
    return self;
  }

  bool is_stopped = false;
};

int main() {
  auto g = guard{} && "pass1"
                   && "pass2"
                   && "pass3"
                   && "stop"
                   && "this is not printed";
```

Towards a (Lazy) Forwarding Mechanism (P0927)

Declaration

```
int log(bool, [] -> std::string message);
```

Call Expression

```
log(value > threshold,
  std::to_string(value) + " exceeds " + std::to_string(threshold));
```

Definition

```
int log(bool condition, [] -> std::string message) {
  if (condition) std::cerr << message() << std::endl;
}
```

Friendly Interfaces

FACADES

Friendly Interfaces (CTRE)

```
#include "ctre.hpp"
#include <string_view>

using match(using x, using sv) {
    static constexpr auto pattern = ctll::basic_fixed_string{ x };
    return ctre::re<pattern>().match(sv);
}


bool result =  match("Hello.*", "Hello World");
```

Friendly Interfaces (P0645)

```
template<class... Args>
string format(string_view fmt, const Args&... args);

string message = format("The answer is {}.", 42);
```

Passing Overloaded Functions

```
auto result = reduce(
  std::forward<decltype(xs)>(xs),
  [](auto&& x, auto&& y) {
    return std::max(std::forward<decltype(x)>(x),
                    std::forward<decltype(y)>(y));
  });
```

Passing Overloaded Functions

```
auto result = reduce(fwd(xs), std::max);
```

FAKE

Language Tuple

FAKE

Language Tuple

```
auto tuples = make_tuples(xs, ys, {84, 10.0f, foo{10}});
```

NOT POSSIBLE WITH FUNCTION TEMPLATES

FAKE Language Tuple

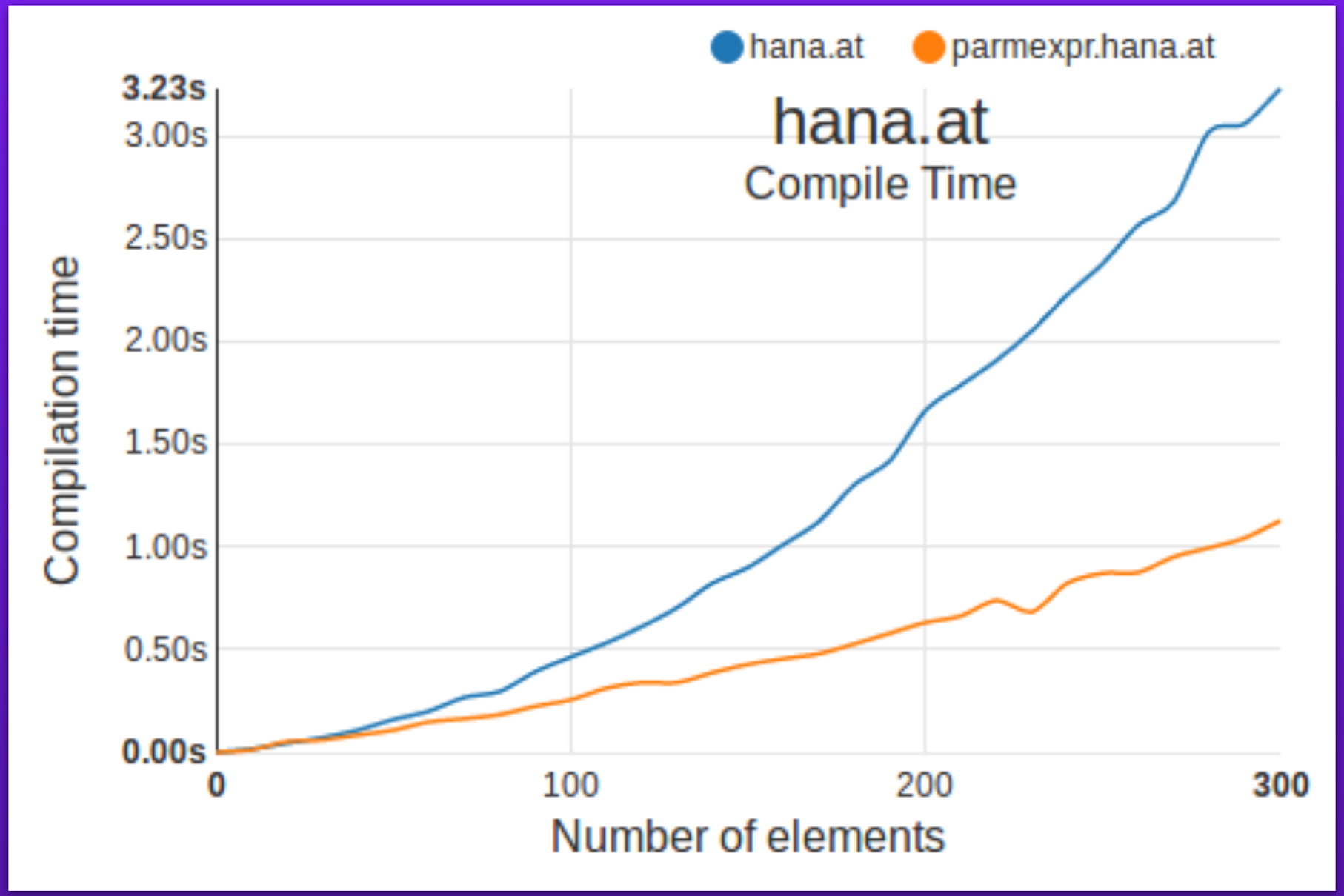
```
// direct-list-initialization
std::tuple xs{42, 5.0f, foo{5}};

// copy-list-initialization
std::tuple ys = {42, 5.0f, foo{5}};
```

FAKE Language Tuple

```
using init_tuple(using tup) {
  std::tuple xs = tup;
  return xs;
}

using make_tuples(using ...tups)
  = std::tuple{init_tuple(tups)...};
```

Crazy Library Stuff

Replacing Functions



What's in a Mangled Name?

```
template <typename T>
void foo(T) { }
```

```
_Z3fooIN5boost4hana5tupleIJNS1_17integral_constantIiLi601EEENS3_IiLi602EEENS3_IiLi603EEENS3_IiLi604EEENS3_IiLi605EEENS3_IiLi606EEENS3_IiLi60
7EEENS3_IiLi608EEENS3_IiLi609EEENS3_IiLi610EEENS3_IiLi611EEENS3_IiLi612EEENS3_IiLi613EEENS3_IiLi614EEENS3_IiLi615EEENS3_IiLi616EEENS3_IiLi61
7EEENS3_IiLi618EEENS3_IiLi619EEENS3_IiLi620EEENS3_IiLi621EEENS3_IiLi622EEENS3_IiLi623EEENS3_IiLi624EEENS3_IiLi625EEENS3_IiLi626EEENS3_IiLi62
7EEENS3_IiLi628EEENS3_IiLi629EEENS3_IiLi630EEENS3_IiLi631EEENS3_IiLi632EEENS3_IiLi633EEENS3_IiLi634EEENS3_IiLi635EEENS3_IiLi636EEENS3_IiLi63
7EEENS3_IiLi638EEENS3_IiLi639EEENS3_IiLi640EEENS3_IiLi641EEENS3_IiLi642EEENS3_IiLi643EEENS3_IiLi644EEENS3_IiLi645EEENS3_IiLi646EEENS3_IiLi64
7EEENS3_IiLi648EEENS3_IiLi649EEENS3_IiLi650EEENS3_IiLi651EEENS3_IiLi652EEENS3_IiLi653EEENS3_IiLi654EEENS3_IiLi655EEENS3_IiLi656EEENS3_IiLi65
7EEENS3_IiLi658EEENS3_IiLi659EEENS3_IiLi660EEENS3_IiLi661EEENS3_IiLi662EEENS3_IiLi663EEENS3_IiLi664EEENS3_IiLi665EEENS3_IiLi666EEENS3_IiLi66
7EEENS3_IiLi668EEENS3_IiLi669EEENS3_IiLi670EEENS3_IiLi671EEENS3_IiLi672EEENS3_IiLi673EEENS3_IiLi674EEENS3_IiLi675EEENS3_IiLi676EEENS3_IiLi67
7EEENS3_IiLi678EEENS3_IiLi679EEENS3_IiLi680EEENS3_IiLi681EEENS3_IiLi682EEENS3_IiLi683EEENS3_IiLi684EEENS3_IiLi685EEENS3_IiLi686EEENS3_IiLi68
7EEENS3_IiLi688EEENS3_IiLi689EEENS3_IiLi690EEENS3_IiLi691EEENS3_IiLi692EEENS3_IiLi693EEENS3_IiLi694EEENS3_IiLi695EEENS3_IiLi696EEENS3_IiLi69
7EEENS3_IiLi698EEENS3_IiLi699EEENS3_IiLi700EEENS3_IiLi701EEENS3_IiLi702EEENS3_IiLi703EEENS3_IiLi704EEENS3_IiLi705EEENS3_IiLi706EEENS3_IiLi70
7EEENS3_IiLi708EEENS3_IiLi709EEENS3_IiLi710EEENS3_IiLi711EEENS3_IiLi712EEENS3_IiLi713EEENS3_IiLi714EEENS3_IiLi715EEENS3_IiLi716EEENS3_IiLi71
7EEENS3_IiLi718EEENS3_IiLi719EEENS3_IiLi720EEENS3_IiLi721EEENS3_IiLi722EEENS3_IiLi723EEENS3_IiLi724EEENS3_IiLi725EEENS3_IiLi726EEENS3_IiLi72
7EEENS3_IiLi728EEENS3_IiLi729EEENS3_IiLi730EEENS3_IiLi731EEENS3_IiLi732EEENS3_IiLi733EEENS3_IiLi734EEENS3_IiLi735EEENS3_IiLi736EEENS3_IiLi73
7EEENS3_IiLi738EEENS3_IiLi739EEENS3_IiLi740EEENS3_IiLi741EEENS3_IiLi742EEENS3_IiLi743EEENS3_IiLi744EEENS3_IiLi745EEENS3_IiLi746EEENS3_IiLi74
7EEENS3_IiLi748EEENS3_IiLi749EEENS3_IiLi750EEENS3_IiLi751EEENS3_IiLi752EEENS3_IiLi753EEENS3_IiLi754EEENS3_IiLi755EEENS3_IiLi756EEENS3_IiLi75
7EEENS3_IiLi758EEENS3_IiLi759EEENS3_IiLi760EEENS3_IiLi761EEENS3_IiLi762EEENS3_IiLi763EEENS3_IiLi764EEENS3_IiLi765EEENS3_IiLi766EEENS3_IiLi76
7EEENS3_IiLi768EEENS3_IiLi769EEENS3_IiLi770EEENS3_IiLi771EEENS3_IiLi772EEENS3_IiLi773EEENS3_IiLi774EEENS3_IiLi775EEENS3_IiLi776EEENS3_IiLi77
7EEENS3_IiLi778EEENS3_IiLi779EEENS3_IiLi780EEENS3_IiLi781EEENS3_IiLi782EEENS3_IiLi783EEENS3_IiLi784EEENS3_IiLi785EEENS3_IiLi786EEENS3_IiLi78
7EEENS3_IiLi788EEENS3_IiLi789EEENS3_IiLi790EEENS3_IiLi791EEENS3_IiLi792EEENS3_IiLi793EEENS3_IiLi794EEENS3_IiLi795EEENS3_IiLi796EEENS3_IiLi79
7EEENS3_IiLi798EEENS3_IiLi799EEENS3_IiLi800EEENS3_IiLi801EEENS3_IiLi802EEENS3_IiLi803EEENS3_IiLi804EEENS3_IiLi805EEENS3_IiLi806EEENS3_IiLi80
7EEENS3_IiLi808EEENS3_IiLi809EEENS3_IiLi810EEENS3_IiLi811EEENS3_IiLi812EEENS3_IiLi813EEENS3_IiLi814EEENS3_IiLi815EEENS3_IiLi816EEENS3_IiLi81
7EEENS3_IiLi818EEENS3_IiLi819EEENS3_IiLi820EEENS3_IiLi821EEENS3_IiLi822EEENS3_IiLi823EEENS3_IiLi824EEENS3_IiLi825EEENS3_IiLi826EEENS3_IiLi82
7EEENS3_IiLi828EEENS3_IiLi829EEENS3_IiLi830EEENS3_IiLi831EEENS3_IiLi832EEENS3_IiLi833EEENS3_IiLi834EEENS3_IiLi835EEENS3_IiLi836EEENS3_IiLi83
7EEENS3_IiLi838EEENS3_IiLi839EEENS3_IiLi840EEENS3_IiLi841EEENS3_IiLi842EEENS3_IiLi843EEENS3_IiLi844EEENS3_IiLi845EEENS3_IiLi846EEENS3_IiLi84
7EEENS3_IiLi848EEENS3_IiLi849EEENS3_IiLi850EEENS3_IiLi851EEENS3_IiLi852EEENS3_IiLi853EEENS3_IiLi854EEENS3_IiLi855EEENS3_IiLi856EEENS3_IiLi85
7EEENS3_IiLi858EEENS3_IiLi859EEENS3_IiLi860EEENS3_IiLi861EEENS3_IiLi862EEENS3_IiLi863EEENS3_IiLi864EEENS3_IiLi865EEENS3_IiLi866EEENS3_IiLi86
7EEENS3_IiLi868EEENS3_IiLi869EEENS3_IiLi870EEENS3_IiLi871EEENS3_IiLi872EEENS3_IiLi873EEENS3_IiLi874EEENS3_IiLi875EEENS3_IiLi876EEENS3_IiLi87
7EEENS3_IiLi878EEENS3_IiLi879EEENS3_IiLi880EEENS3_IiLi881EEENS3_IiLi882EEENS3_IiLi883EEENS3_IiLi884EEENS3_IiLi885EEENS3_IiLi886EEENS3_IiLi88
7EEENS3_IiLi888EEENS3_IiLi889EEENS3_IiLi890EEENS3_IiLi891EEENS3_IiLi892EEENS3_IiLi893EEENS3_IiLi894EEENS3_IiLi895EEENS3_IiLi896EEENS3_IiLi89
7EEENS3_IiLi898EEENS3_IiLi899EEENS3_IiLi900EEEEEEEvT_
```


4k

_ZN5boost4hana6detail16basic_tuple_implINSt3__116integer_sequenceImJLm0ELm1ELm2ELm3ELm4ELm5ELm6ELm7ELm8ELm9ELm10ELm11ELm12ELm13ELm14ELm15ELm
16ELm17ELm18ELm19ELm20ELm21ELm22ELm23ELm24ELm25ELm26ELm27ELm28ELm29ELm30ELm31ELm32ELm33ELm34ELm35ELm36ELm37ELm38ELm39ELm40ELm41ELm42ELm43ELm
44ELm45ELm46ELm47ELm48ELm49ELm50ELm51ELm52ELm53ELm54ELm55ELm56ELm57ELm58ELm59ELm60ELm61ELm62ELm63ELm64ELm65ELm66ELm67ELm68ELm69ELm70ELm71ELm
72ELm73ELm74ELm75ELm76ELm77ELm78ELm79ELm80ELm81ELm82ELm83ELm84ELm85ELm86ELm87ELm88ELm89ELm90ELm91ELm92ELm93ELm94ELm95ELm96ELm97ELm98ELm99ELm
100ELm101ELm102ELm103ELm104ELm105ELm106ELm107ELm108ELm109ELm110ELm111ELm112ELm113ELm114ELm115ELm116ELm117ELm118ELm119ELm120ELm121ELm122ELm12
3ELm124ELm125ELm126ELm127ELm128ELm129ELm130ELm131ELm132ELm133ELm134ELm135ELm136ELm137ELm138ELm139ELm140ELm141ELm142ELm143ELm144ELm145ELm146E
Lm147ELm148ELm149ELm150ELm151ELm152ELm153ELm154ELm155ELm156ELm157ELm158ELm159ELm160ELm161ELm162ELm163ELm164ELm165ELm166ELm167ELm168ELm169ELm
170ELm171ELm172ELm173ELm174ELm175ELm176ELm177ELm178ELm179ELm180ELm181ELm182ELm183ELm184ELm185ELm186ELm187ELm188ELm189ELm190ELm191ELm192ELm19
3ELm194ELm195ELm196ELm197ELm198ELm199ELm200ELm201ELm202ELm203ELm204ELm205ELm206ELm207ELm208ELm209ELm210ELm211ELm212ELm213ELm214ELm215ELm216E
Lm217ELm218ELm219ELm220ELm221ELm222ELm223ELm224ELm225ELm226ELm227ELm228ELm229ELm230ELm231ELm232ELm233ELm234ELm235ELm236ELm237ELm238ELm239ELm
240ELm241ELm242ELm243ELm244ELm245ELm246ELm247ELm248ELm249ELm250ELm251ELm252ELm253ELm254ELm255ELm256ELm257ELm258ELm259ELm260ELm261ELm262ELm26
3ELm264ELm265ELm266ELm267ELm268ELm269ELm270ELm271ELm272ELm273ELm274ELm275ELm276ELm277ELm278ELm279ELm280ELm281ELm282ELm283ELm284ELm285ELm286E
Lm287ELm288ELm289ELm290ELm291ELm292ELm293ELm294ELm295ELm296ELm297ELm298ELm299EEEEJNS0_17integral_constantIiLi1EEENS6_IiLi2EEENS6_IiLi3EEENS6
_IiLi4EEENS6_IiLi5EEENS6_IiLi6EEENS6_IiLi7EEENS6_IiLi8EEENS6_IiLi9EEENS6_IiLi10EEENS6_IiLi11EEENS6_IiLi12EEENS6_IiLi13EEENS6_IiLi14EEENS6_Ii
Li15EEENS6_IiLi16EEENS6_IiLi17EEENS6_IiLi18EEENS6_IiLi19EEENS6_IiLi20EEENS6_IiLi21EEENS6_IiLi22EEENS6_IiLi23EEENS6_IiLi24EEENS6_IiLi25EEENS6
_IiLi26EEENS6_IiLi27EEENS6_IiLi28EEENS6_IiLi29EEENS6_IiLi30EEENS6_IiLi31EEENS6_IiLi32EEENS6_IiLi33EEENS6_IiLi34EEENS6_IiLi35EEENS6_IiLi36EEE
NS6_IiLi37EEENS6_IiLi38EEENS6_IiLi39EEENS6_IiLi40EEENS6_IiLi41EEENS6_IiLi42EEENS6_IiLi43EEENS6_IiLi44EEENS6_IiLi45EEENS6_IiLi46EEENS6_IiLi47
EEENS6_IiLi48EEENS6_IiLi49EEENS6_IiLi50EEENS6_IiLi51EEENS6_IiLi52EEENS6_IiLi53EEENS6_IiLi54EEENS6_IiLi55EEENS6_IiLi56EEENS6_IiLi57EEENS6_IiL
i58EEENS6_IiLi59EEENS6_IiLi60EEENS6_IiLi61EEENS6_IiLi62EEENS6_IiLi63EEENS6_IiLi64EEENS6_IiLi65EEENS6_IiLi66EEENS6_IiLi67EEENS6_IiLi68EEENS6_
IiLi69EEENS6_IiLi70EEENS6_IiLi71EEENS6_IiLi72EEENS6_IiLi73EEENS6_IiLi74EEENS6_IiLi75EEENS6_IiLi76EEENS6_IiLi77EEENS6_IiLi78EEENS6_IiLi79EEEN
S6_IiLi80EEENS6_IiLi81EEENS6_IiLi82EEENS6_IiLi83EEENS6_IiLi84EEENS6_IiLi85EEENS6_IiLi86EEENS6_IiLi87EEENS6_IiLi88EEENS6_IiLi89EEENS6_IiLi90E
EENS6_IiLi91EEENS6_IiLi92EEENS6_IiLi93EEENS6_IiLi94EEENS6_Ii[illegible]NS6_IiLi97EEENS6_IiLi98EEENS6_IiLi99EEENS6_IiLi100EEENS6_IiL
i101EEENS6_IiLi102EEENS6_IiLi103EEENS6_IiLi104EEENS6_IiLi105[illegible]6_IiLi107EEENS6_IiLi108EEENS6_IiLi109EEENS6_IiLi110EEENS6_IiL
i111EEENS6_IiLi112EEENS6_IiLi113EEENS6_IiLi114EEENS6_IiLi115EEE[illegible]6_IiLi117EEENS6_IiLi118EEENS6_IiLi119EEENS6_IiLi120EEENS6_IiL
i121EEENS6_IiLi122EEENS6_IiLi123EEENS6_IiLi124EEENS6_IiLi125EEE[illegible]S6_IiLi127EEENS6_IiLi128EEENS6_IiLi129EEENS6_IiLi130EEENS6_IiL
i131EEENS6_IiLi132EEENS6_IiLi133EEENS6_IiLi134EEENS6_IiLi135EEE[illegible]S6_IiLi137EEENS6_IiLi138EEENS6_IiLi139EEENS6_IiLi140EEENS6_IiL
i141EEENS6_IiLi142EEENS6_IiLi143EEENS6_IiLi144EEENS6_IiLi145EEE[illegible]6_IiLi147EEENS6_IiLi148EEENS6_IiLi149EEENS6_IiLi150EEENS6_IiL
i151EEENS6_IiLi152EEENS6_IiLi153EEENS6_IiLi154EEENS6_IiLi155EE[illegible]6_IiLi157EEENS6_IiLi158EEENS6_IiLi159EEENS6_IiLi160EEENS6_IiL
i161EEENS6_IiLi162EEENS6_IiLi163EEENS6_IiLi164EEENS6_IiLi165EE[illegible]_IiLi167EEENS6_IiLi168EEENS6_IiLi169EEENS6_IiLi170EEENS6_IiL
i171EEENS6_IiLi172EEENS6_IiLi173EEENS6_IiLi174EEENS6_IiLi175E[illegible]IiLi177EEENS6_IiLi178EEENS6_IiLi179EEENS6_IiLi180EEENS6_IiL
i181EEENS6_IiLi182EEENS6_IiLi183EEENS6_IiLi184EEENS6_IiLi185EEENS6_IiLi186EEENS6_IiLi187EEENS6_IiLi188EEENS6_IiLi189EEENS6_IiLi190EEENS6_IiL
i191EEENS6_IiLi192EEENS6_IiLi193EEENS6_IiLi194EEENS6_IiLi195EEENS6_IiLi196EEENS6_IiLi197EEENS6_IiLi198EEENS6_IiLi199EEENS6_IiLi200EEENS6_IiL
i201EEENS6_IiLi202EEENS6_IiLi203EEENS6_IiLi204EEENS6_IiLi205EEENS6_IiLi206EEENS6_IiLi207EEENS6_IiLi208EEENS6_IiLi209EEENS6_IiLi210EEENS6_IiL
i211EEENS6_IiLi212EEENS6_IiLi213EEENS6_IiLi214EEENS6_IiLi215EEENS6_IiLi216EEENS6_IiLi217EEENS6_IiLi218EEENS6_IiLi219EEENS6_IiLi220EEENS6_IiL
i221EEENS6_IiLi222EEENS6_IiLi223EEENS6_IiLi224EEENS6_IiLi225EEENS6_IiLi226EEENS6_IiLi227EEENS6_IiLi228EEENS6_IiLi229EEENS6_IiLi230EEENS6_IiL
i231EEENS6_IiLi232EEENS6_IiLi233EEENS6_IiLi234EEENS6_IiLi235EEENS6_IiLi236EEENS6_IiLi237EEENS6_IiLi238EEENS6_IiLi239EEENS6_IiLi240EEENS6_IiL
i241EEENS6_IiLi242EEENS6_IiLi243EEENS6_IiLi244EEENS6_IiLi245EEENS6_IiLi246EEENS6_IiLi247EEENS6_IiLi248EEENS6_IiLi249EEENS6_IiLi250EEENS6_IiL
i251EEENS6_IiLi252EEENS6_IiLi253EEENS6_IiLi254EEENS6_IiLi255EEENS6_IiLi256EEENS6_IiLi257EEENS6_IiLi258EEENS6_IiLi259EEENS6_IiLi260EEENS6_IiL
i261EEENS6_IiLi262EEENS6_IiLi263EEENS6_IiLi264EEENS6_IiLi265EEENS6_IiLi266EEENS6_IiLi267EEENS6_IiLi268EEENS6_IiLi269EEENS6_IiLi270EEENS6_IiL
i271EEENS6_IiLi272EEENS6_IiLi273EEENS6_IiLi274EEENS6_IiLi275EEENS6_IiLi276EEENS6_IiLi277EEENS6_IiLi278EEENS6_IiLi279EEENS6_IiLi280EEENS6_IiL
i281EEENS6_IiLi282EEENS6_IiLi283EEENS6_IiLi284EEENS6_IiLi285EEENS6_IiLi286EEENS6_IiLi287EEENS6_IiLi288EEENS6_IiLi289EEENS6_IiLi290EEENS6_IiL
i291EEENS6_IiLi292EEENS6_IiLi293EEENS6_IiLi294EEENS6_IiLi295EEENS6_IiLi296EEENS6_IiLi297EEENS6_IiLi298EEENS6_IiLi299EEENS6_IiLi300EEEEEC2IJS
7_S8_S9_SA_SB_SC_SD_SE_SF_SG_SH_SI_SJ_SK_SL_SM_SN_SO_SP_SQ_SR_SS_ST_SU_SV_SW_SX_SY_SZ_S10_S11_S12_S13_S14_S15_S16_S17_S18_S19_S1A_S1B_S1C_S1
D_S1E_S1F_S1G_S1H_S1I_S1J_S1K_S1L_S1M_S1N_S1O_S1P_S1Q_S1R_S1S_S1T_S1U_S1V_S1W_S1X_S1Y_S1Z_S20_S21_S22_S23_S24_S25_S26_S27_S28_S29_S2A_S2B_S2
C_S2D_S2E_S2F_S2G_S2H_S2I_S2J_S2K_S2L_S2M_S2N_S2O_S2P_S2Q_S2R_S2S_S2T_S2U_S2V_S2W_S2X_S2Y_S2Z_S30_S31_S32_S33_S34_S35_S36_S37_S38_S39_S3A_S3
B_S3C_S3D_S3E_S3F_S3G_S3H_S3I_S3J_S3K_S3L_S3M_S3N_S3O_S3P_S3Q_S3R_S3S_S3T_S3U_S3V_S3W_S3X_S3Y_S3Z_S40_S41_S42_S43_S44_S45_S46_S47_S48_S49_S4
A_S4B_S4C_S4D_S4E_S4F_S4G_S4H_S4I_S4J_S4K_S4L_S4M_S4N_S4O_S4P_S4Q_S4R_S4S_S4T_S4U_S4V_S4W_S4X_S4Y_S4Z_S50_S51_S52_S53_S54_S55_S56_S57_S58_S5
9_S5A_S5B_S5C_S5D_S5E_S5F_S5G_S5H_S5I_S5J_S5K_S5L_S5M_S5N_S5O_S5P_S5Q_S5R_S5S_S5T_S5U_S5V_S5W_S5X_S5Y_S5Z_S60_S61_S62_S63_S64_S65_S66_S67_S6
8_S69_S6A_S6B_S6C_S6D_S6E_S6F_S6G_S6H_S6I_S6J_S6K_S6L_S6M_S6N_S6O_S6P_S6Q_S6R_S6S_S6T_S6U_S6V_S6W_S6X_S6Y_S6Z_S70_S71_S72_S73_S74_S75_S76_S7
7_S78_S79_S7A_S7B_S7C_S7D_S7E_S7F_S7G_S7H_S7I_S7J_S7K_S7L_S7M_S7N_S7O_S7P_S7Q_S7R_S7S_S7T_S7U_S7V_S7W_S7X_S7Y_S7Z_S80_S81_S82_S83_S84_S85_S8
6_S87_S88_S89_S8A_S8B_S8C_S8D_S8E_S8F_S8G_S8H_S8I_EEEDpOT_

boost::hana::at_c<N>

boost::hana::at

boost::hana::at_impl

detail::ebo_get<N>

```
template <std::size_t n>
struct at_c_fn {
  static using operator()(using xs) =
    hana::at(xs, hana::size_t<n>{});
};

template <std::size_t n>
inline constexpr at_c_fn<n> at_c{};
```

```
template <typename Xs, typename N>
constexpr decltype(auto) at_t::operator()(Xs&& xs, N const& n) const {
  using It = typename hana::tag_of<Xs>::type;
  using At = BOOST_HANA_DISPATCH_IF(at_impl<It>,
    hana::Iterable<It>::value
  );

  return At::apply(static_cast<Xs&&>(xs), n);
}
```

```
struct at_t {
  template <typename Xs>
  using Tag = typename hana::tag_of<Xs>::type;

  template <typename It>
  using Impl = BOOST_HANA_DISPATCH_IF(at_impl<It>,
    hana::Iterable<It>::value);

  static using operator()(using xs, using n) =
    Impl<Tag<decltype(xs)>>::apply(xs, n);
};
```

```
template <>
struct at_impl<basic_tuple_tag> {
  template <typename Xs, typename N>
  static constexpr decltype(auto) apply(Xs&& xs, N const&) {
    constexpr std::size_t index = N::value;
    return detail::ebo_get<detail::bti<index>>(static_cast<Xs&&>(xs));
  }
};
```

```
template <>
struct at_impl<basic_tuple_tag> {
  static using apply(using xs, using n) =
    static_cast<detail::fwd_cast_t<
      typename detail::decay<decltype(xs)>::type
        ::template ebo_t<static_cast<size_t>(n)>,
      decltype((xs))>>(xs).get();
};
```

```
// Specialize storage for non-empty types
template <typename K, typename V>
struct ebo<K, V, false> {
    constexpr ebo() : data_() { }

    template <typename T>
    explicit constexpr ebo(T&& t)
        : data_(static_cast<T&&>(t))
    { }

    V data_;

    // this gets instantiated for every K, V
    using get(using this self) = return (self.data_);
};
```

Conclusion

Improve Interfaces

Reduce Type Bloat

Parametric Expressions P1221

@JasonRice_

/ricejasonf

Parametric Expressions P1221

https://github.com/ricejasonf/parametric_expressions/blob/master/d1221.md

Packs of Packs

https://quuxplusone.github.io/blog/2019/02/11/tilde-notation-for-exploding-tuples/

Deducing `this` P0847

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0847r2.html

Structured Bindings can introduce a Pack

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1061r0.html

static `operator()`

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1169r0.html

Towards A (Lazy) Forwarding Mechanism for C++

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0927r0.pdf

Compile Time Regular Expressions

https://github.com/hanickadot/compile-time-regular-expressions

Text Formatting P0645

http://fmtlib.net/Text%20Formatting.html

Boost Hana

https://github.com/boostorg/hana