Core C++ 2024

Welcome to v0.3 of the meta::[[verse]]!

Inbal Levi

Welcome to v0.3 of the meta::[[verse]]!

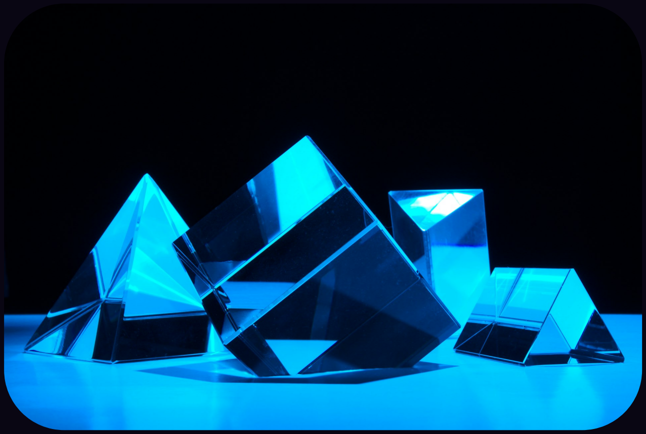
Inbal Levi

- Lead C++ Developer
- ISO C++ foundation and Boost foundation board member
- An active member of the ISO C++ Committee
- Israeli National Body Chair
- **Library Evolution Work Group Chair**

Reflection in C++

- The ability of software to expose its internal structure
- **Static reflection** - **<u>compiler</u>** exposes structure at **compile time**

- Disclaimers:
  - May be possible to "carry" the data into the runtime...
  - But may introduce performance, security, or other issues
  - This is out of scope for this talk (and for C++26, as of now)



This Talk

- Part I: Intro to reflection:
  1. A brief history of "Reflection" proposals in C++
  2. Latest proposal: "P2996: Reflection for C++26"
  3. Usage examples (reflection-based library) (*)

- Part II: Impact on our code bases
  1. Reflection as a Customization Point Mechanism
  2. Pipeline integration
  3. What's next?

(*) Examples from or derive from P2996, P3096, or EDG's implementation

A Brief History

2006
Template-based reflection
(Matúš Chochlik)
Mirror reflection library

2012
N3403: Use Cases
for Compile-Time
Reflection
(Mike Spertus)

2014 - 2016
N3996: Static reflection
(Matúš Chochlik)
N4113, N4239, N4027, N3984
(type traits, attributes, etc.)

2015
Boost.Hana
(Louis Dionne)
Template metaprog. lib

A Brief History

2017

P0194 | P0385: Static Reflection
(Matúš, Axel, David)
P0590: A design for static reflection
(Andrew, Herb)
P0633: Exploring the design space
(Daveed, Louis)

meta
objects

2018

P0954: What do we want to do
with reflection?
(Bjarne)

2018

N4747: Reflection TS
(David)

2018

P0993: Value-based reflection
(Andrew, Herb)
P1240: Scalable Reflection in C++
(Andrew, Faisal, Daveed)

2019

P0953: constexpr reflexpr
(Matúš, Axel, David)
P1733: User-friendly and
Evolution-friendly
Reflection: A Compromise
(David, Daveed)

Typefull
vs.
Monotype

Type based
vs.
Value based

A Brief History

2021

P2320: The Syntax of Static Reflection
(Andrew, Wyatt, Daveed)
(+ implementation: lock3 meta)

2023

P2996: Reflection for C++26
(Barry, Wyatt, Peter, Andrew, Faisal, Daveed)

2024

P2996R7

(1.16 year later)

2020

P2087: Reflection Naming: fix reflexpr
(Mihail Naydenov)
P2040: Reflection-based lazy-evaluation
(Corentin)

2022

P2560: Comparing Value- and type-based reflection
(Matus)

2023

P2911: Python Bindings (...)
(Adam, Jagrut)
P3010: Using reflection (...) JS Bindings
(Dan Katz)

What do I mean by "Reflection"?

```cpp
// Lib.hpp
class LibType : public BaseOne, public BaseTwo
{
    int a;
    double b;
};
```

```cpp
// main.cpp
#include <meta>
#include "Lib.hpp"

int main()
{
    constexpr std::meta::info refexpr = ^^LibType;
    auto res = std::meta::bases_of(refexpr);
}
```

* Note: This is a pseudo-code, does not work as is

P2996: Reflection for C++26

1. Reflection Operator: `^^`
2. Splicers: `[: … :]`
3. std::meta::info
4. Metafunctions
   1. Name & Location:
      1. identifier_of
      2. display_string_of
      3. source_location_of
   2. Type Queries:
      1. type_of
      2. parent_of
      3. dealias

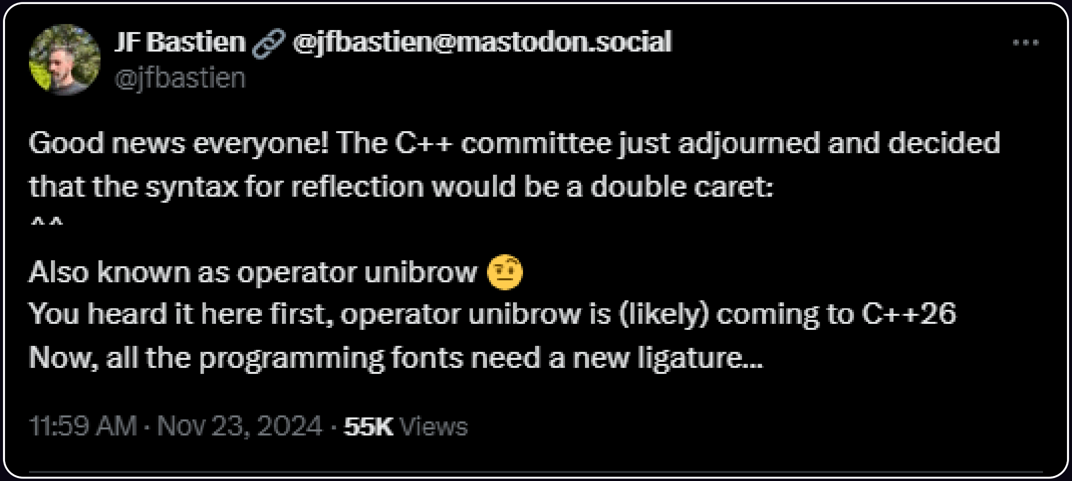
      (+ Other Type Predicates)

      4. Access modifiers: is_public, is_protected, is_private
      5. Inheritance: is_virtual, is_pure_virtual, is_override, …
      6. Encapsulation: is_class_member, is_namespace_member, is_explicit, is_deleted, …
      7. Advanced Type Queries: is_complete_type, is_template, is_special_member, …
   3. Template Queries:
      1. template_of
      2. template_arguments_of
   4. Member Queries:
      1. members_of
      2. bases_of
      3. (non)static_data_members_of
      4. ~~accessible_members_of~~ (P3293R2)
      5. enumerators_of
   5. substitute (template)
   6. reflect Invoke (template)
   7. extract<T>(info) (constexpr not required)
   8. test_type(s) ("is_same")
   9. reflect_value (template)
   10. define_class (injection)
   11. Data Layout:
       1. offset_of -> member_offsets
       2. size_of
       3. bit_size_of
       4. alignment_of
   12. (+) Type Traits...?

The Reflection Operator (^^) (Lift)

AKA Unibrow

```cpp
auto rexpr = ^^int;
```



JF Bastien 🔗 @jfbastien@mastodon.social
@jfbastien

Good news everyone! The C++ committee just adjourned and decided that the syntax for reflection would be a double caret:
^^
Also known as operator unibrow 🤨
You heard it here first, operator unibrow is (likely) coming to C++26
Now, all the programming fonts need a new ligature...

11:59 AM · Nov 23, 2024 · **55K** Views

The Reflection Operator (^^) (Lift)

```cpp
constexpr auto rexpr = ^^int;
```

error: meta type variables must be constexpr

- Shifts expressions into a "meta" - "reflection info" object
- Object can then to be used as input to reflection utilities

Splicers

```cpp
constexpr auto rexpr = ^^int;
typename[:rexpr:] a  = 42;
```

- Splice extract the C++ expression back from "meta::info"...
- ...To be then used regularly to write the C++ program

What are you?

rexpr	meta::info obj, contains info on "int" type
typename[:rexpr:]	int

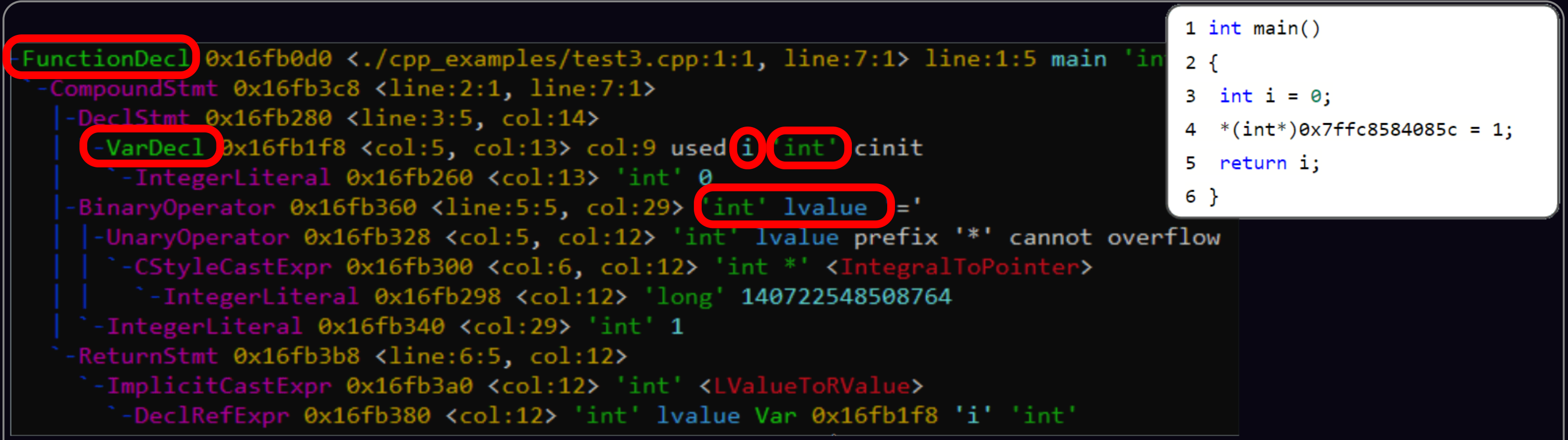
P2996: Reflection for C++26

1. Reflection Operator: ^^
2. Splicers: [: ... :]
3. std::meta::info
4. Metafunctions
    1. Name & Location:
        1. identifier_of
        2. display_string_of
        3. source_location_of
    2. Type Queries:
        1. type_of
        2. parent_of
        3. dealias

    (+ Other Type Predicates)

        4. Access modifiers: is_public, is_protected, is_private
        5. Inheritance: is_virtual, is_pure_virtual, is_override, ...
        6. Encapsulation: is_class_member, is_namespace_member, is_explicit, is_deleted, ...
        7. Advanced Type Queries: is_complete_type, is_template, is_special_member, ...
    3. Template Queries:
        1. template_of
        2. template_arguments_of
    4. Member Queries:
        1. members_of
        2. bases_of
        3. (non)static_data_members_of
        4. ~~accessible_members_of~~ (P3293R2)
        5. enumerators_of
    5. substitute (template)
    6. reflect Invoke (template)
    7. extract<T>(info) (constexpr not required)
    8. test_type(s) ("is_same")
    9. reflect_value (template)
    10. define_class (injection)
    11. Data Layout:
        1. offset_of -> member_offsets
        2. size_of
        3. bit_size_of
        4. alignment_of
    12. (+) Type Traits...?

std::meta::info



```
-FunctionDecl 0x16fb0d0 <./cpp_examples/test3.cpp:1:1, line:7:1> line:1:5 main 'in
 `-CompoundStmt 0x16fb3c8 <line:2:1, line:7:1>
   |-DeclStmt 0x16fb280 <line:3:5, col:14>
   | `-VarDecl 0x16fb1f8 <col:5, col:13> col:9 used i 'int' cinit
   |   `-IntegerLiteral 0x16fb260 <col:13> 'int' 0
   |-BinaryOperator 0x16fb360 <line:5:5, col:29> 'int' lvalue '='
   | |-UnaryOperator 0x16fb328 <col:5, col:12> 'int' lvalue prefix '*' cannot overflow
   | | `-CStyleCastExpr 0x16fb300 <col:6, col:12> 'int *' <IntegralToPointer>
   | |   `-IntegerLiteral 0x16fb298 <col:12> 'long' 140722548508764
   | `-IntegerLiteral 0x16fb340 <col:29> 'int' 1
   `-ReturnStmt 0x16fb3b8 <line:6:5, col:12>
     `-ImplicitCastExpr 0x16fb3a0 <col:12> 'int' <LValueToRValue>
       `-DeclRefExpr 0x16fb380 <col:12> 'int' lvalue Var 0x16fb1f8 'i' 'int'
```

```cpp
int main()
{
  int i = 0;
  *(int*)0x7ffc8584085c = 1;
  return i;
}
```

From: "Let's talk about abstraction layers"

std::meta::info

Represents:

- Type and type alias
- Function or member function
- Variable, static data member, or structured binding
- Non-static data member
- Constant value
- Constant expression
- Template
- Namespaces

```
constexpr int a = 42;
constexpr auto b = ^^a;
std::cout << [:b:] << "\n";           // OK
std::cout << [:^^(a*2):] << "\n"; // OK
```

std::meta::info

```cpp
#include <iostream>
#include <experimental/meta>

class R;

constexpr std::meta::info res1 = ^^R;
constexpr auto print1 = std::meta::
                    is_complete_type(res1); // (1)

class R {
    int a;
};

constexpr std::meta::info res2 = ^^R;
```

https://godbolt.org/z/sYs39bj5e

P2996: Reflection for C++26

1. Reflection Operator: ^^
2. Splicers: [: … :]
3. std::meta::info
4. Metafunctions
    1. Name & Location:
        1. identifier_of
        2. display_string_of
        3. source_location_of
    2. Type Queries:
        1. type_of
        2. parent_of
        3. dealias

    (+ Other Type Predicates)

        4. Access modifiers: is_public, is_protected, is_private
        5. Inheritance: is_virtual, is_pure_virtual, is_override, …
        6. Encapsulation: is_class_member, is_namespace_member, is_explicit, is_deleted, …
        7. Advanced Type Queries: is_complete_type, is_template, is_special_member, …
    3. Template Queries:
        1. template_of
        2. template_arguments_of
    4. Member Queries:
        1. members_of
        2. bases_of
        3. (non)static_data_members_of
        4. ~~accessible_members_of~~ (P3293R2)
        5. enumerators_of
    5. substitute (template)
    6. reflect Invoke (template)
    7. extract<T>(info) (constexpr not required)
    8. test_type(s) ("is_same")
    9. reflect_value (template)
    10. define_class (injection)
    11. Data Layout:
        1. offset_of -> member_offsets
        2. size_of
        3. bit_size_of
        4. alignment_of
    12. (+) Type Traits...?

Metafunctions

```
constexpr auto a = 42;
std::cout << meta::name_of(^^a) << "\n";
```

All* the metafunctions accept "info" and return:

1. string_view (e.g. identifier_of)
2. std::meta::info (e.g. type_of)
3. vector<std::meta::info> (e.g. bases_of)
4. bool (e.g. is_concept)
5. size_t (e.g. alignment_of)
6. T (e.g. extract(info))
7. source_location, member_offset (e.g. offset_of)

* Besides "reflect_value/object/function", which accepts type T

* Note: This is a pseudo-code, does not work as is

Type Traits

- Two types of "Type Traits":
  - Query the type
  - Modify the type
- P2996 proposes the first type (e.g. is_nothrow_swappable)
- Interesting functionality available by the second type
- ...But this is more challenging, as it interferes with the compiler's representation of the program

Reflection Logger

```cpp
#include <string>
#include <experimental/meta>

consteval bool LogMembers(std::meta
{
  // Verify type is a class/struct
  // Ignore usecase of unnamed members (union/struct etc.)
  std::__report_constexpr_value(name_of(type).data());

  for (auto r : nonstatic_data_members_of(type))
  {
      std::__report_constexpr_value("\n\tmember:");
      std::__report_constexpr_value(name_of(r).data());
  }
  return true;
}

```

```
>> output from std::__report_constexpr_value
at line 21 of <source>
Student
        member:name
        member:id
>> end output from std::__report_constexpr_value
```

Based on example from EDG's CE implementation

https://godbolt.org/z/oobfx9E7h

Command Line Args

```cpp
// Members represent Args (should be known at compile time)
using namespace clap;
struct Args : Clap
{
  Option<std::string, Flags{.use_short=true, .use_long=true}> name = "";
  Option<int, Flags{.use_short=true, .use_long=true}> count = 0;
};

int main(int argc, char** argv)
{
  auto opts = Args{}.parse(argc, argv);

  for (int i = 0; i < opts.count; ++i)
  {
    std::cout << "Hello " << opts.name << "!\n";
  }
}
```

Based on example from P2996

https://godbolt.org/z/9GE3ePK71

Function Param Names

```cpp
#include <experimental/meta>
#include <iostream>

consteval auto PrintParamK(std::meta::info r, size_t k)
{
    return name_of(parameters_of(r)[k]);
}

// Function Declaration
void func(int first, int last);

void PrintFuncParamAfterDeclaration()
{
    std::cout << "Param names: "
    << PrintParamK(^func, 0) << ", "
    << PrintParamK(^func, 1) << "\n";
}
```

Based on example from P3096

https://godbolt.org/z/M84Ea6P88

Reflection Based Library

- Interaction between library code and user(*) code

Library Code

```
void PrintFuncParams()
{
  std::cout << "Param names: "
  << PrintParamK(^func, 0) << ", "
  << PrintParamK(^func, 1) << "\n";
}
```

User(*) Code

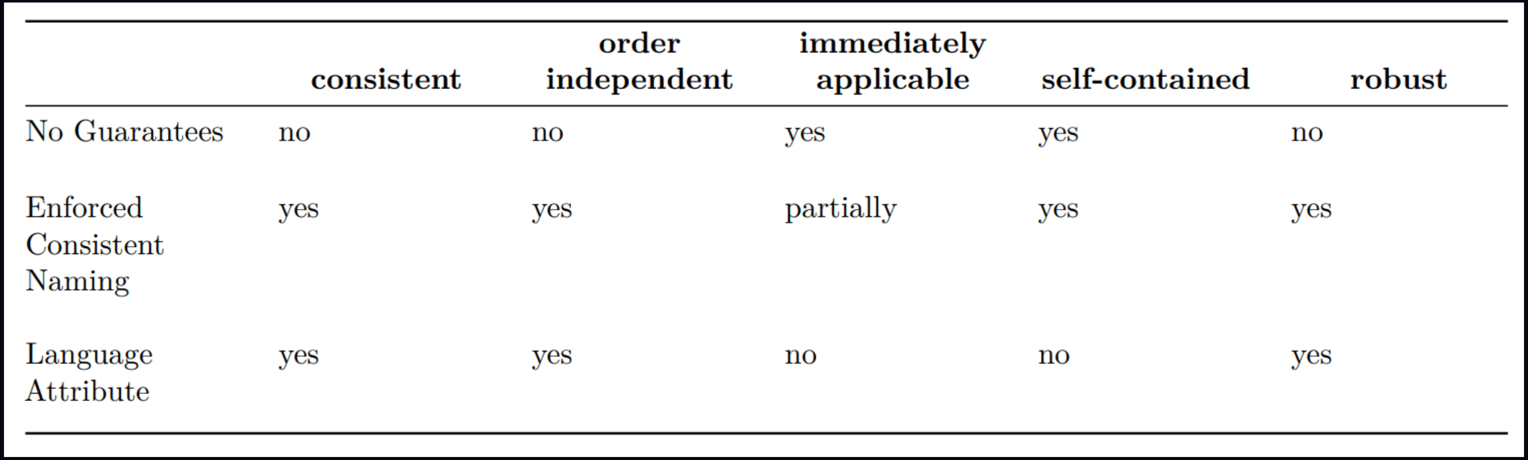
```
void func(int first, int last);
// PrintParamsLocation1() { ... }
void func(int a, int b) { ... }
// PrintParamsLocation2() { ... }

int main()
{
  PrintFuncParamsLocation1();
  PrintFuncParamsLocation2();
}
```

"P3096: Function Parameter Reflection in Reflection for C++26"
(Adam Lach, Walter Genovese)

Function Param Names

- P3096 Introduces the following options:
  1. Compile, No Guarantees
     (Different compilers' output may be inconsistent)
  2. Enforce Consistent Naming
     (Don't compile if more than one option exists)
  3. Mark by attribute (e.g. [[canonical]])
     (Explicitly mark, otherwise ill-formed)



	consistent	order independent	immediately applicable	self-contained	robust
No Guarantees	no	no	yes	yes	no
Enforced Consistent Naming	yes	yes	partially	yes	yes
Language Attribute	yes	yes	no	no	yes

Function Param Names

- Forward competability

Library Code

```
void PrintFuncParams()
{
  std::cout << "Param names: "
  << PrintParamK(^^func, 0) << ", "
  << PrintParamK(^^func, 1) << "\n";
}
```

New User Code

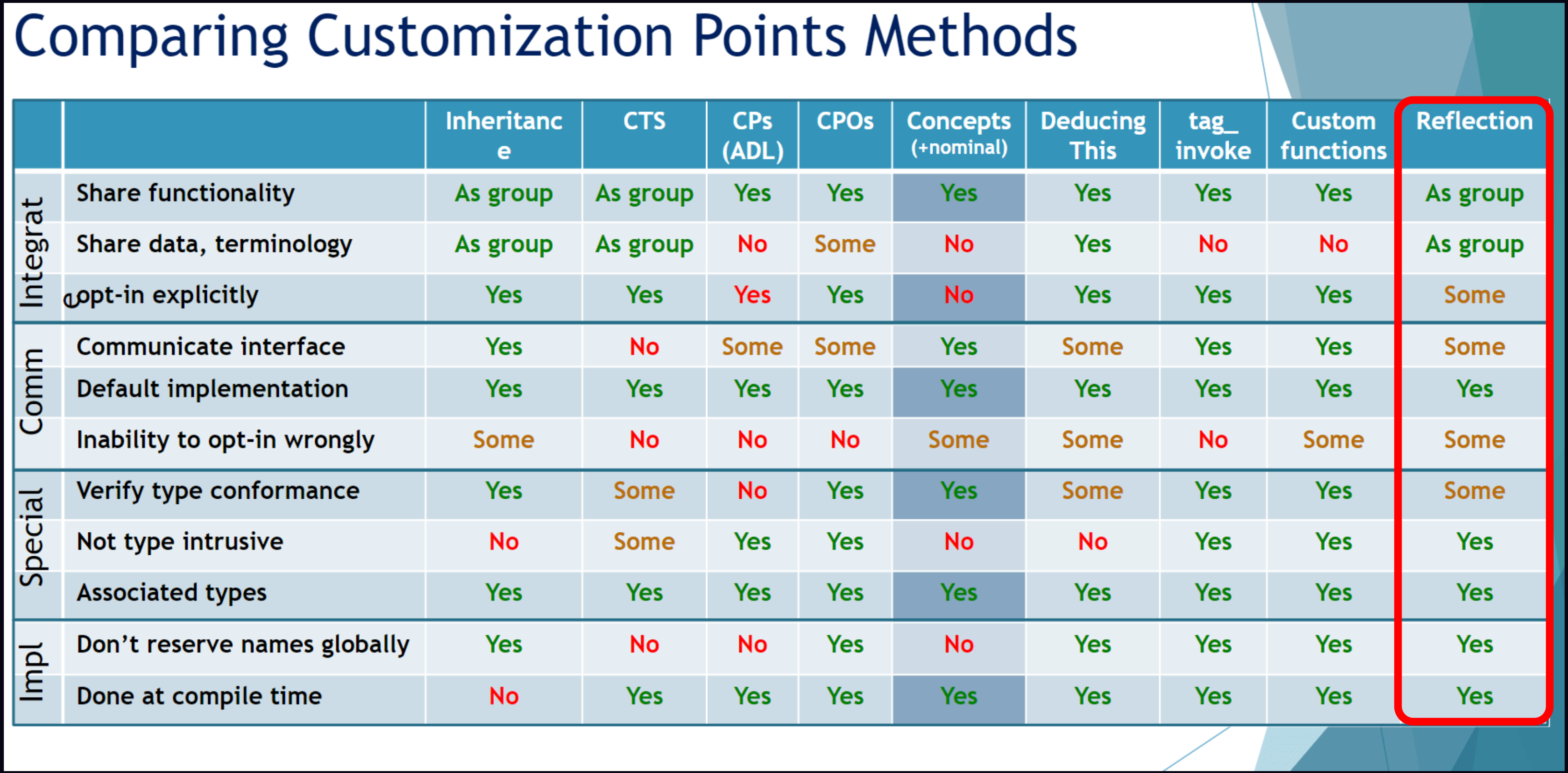
```
void func(int first, int last);
// PrintParamsLocation1() { ... }
void func(int a, int b)
{ ... }
// PrintParamsLocation2() { ... }

int main()
{
  PrintFuncParamsLocation1();
  PrintFuncParamsLocation2();
}
```

"P3096: Function Parameter Reflection in Reflection for C++26"
(Adam Lach, Walter Genovese)

Detour: Customization Points

Guarantees and requirements provided by the library can be expressed as "Customization Points"



Comparing Customization Points Methods

		Inheritance	CTS	CPs (ADL)	CPOs	Concepts (+nominal)	Deducing This	tag_ invoke	Custom functions	Reflection
Integrate	Share functionality	As group	As group	Yes	Yes	Yes	Yes	Yes	Yes	As group
	Share data, terminology	As group	As group	No	Some	No	Yes	No	No	As group
	opt-in explicitly	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Some
Comm	Communicate interface	Yes	No	Some	Some	Yes	Some	Yes	Yes	Some
	Default implementation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Inability to opt-in wrongly	Some	No	No	No	Some	Some	No	Some	Some
Special	Verify type conformance	Yes	Some	No	Yes	Yes	Some	Yes	Yes	Some
	Not type intrusive	No	Some	Yes	Yes	No	No	Yes	Yes	Yes
	Associated types	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Impl	Don't reserve names globally	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes
	Done at compile time	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

C++Now 2022, Cpp India 2022, CoreC++ 2022, CppCon 2023
"Customization Methods: Connecting User and C++ Library Code"

Detour: Customization Points

- Interaction between library code and user (*) code:
  - Virtual functions:
    - Library side: declaration of virtual functions
    - User side: overriding virtual functions
  - Template instantiation:
    - Library side: declare a template
    - User side: instantiate template for their types
  - Provide functionality to be detected by ADL:
    - Library side: allows/expects functions by name
    - User side: declares a function within their type to be detected by ADL
  - And now - Reflection?

C++Now 2022, Cpp India 2022, CoreC++ 2022, CppCon 2023
"Customization Methods: Connecting User and C++ Library Code"

Reflection as Customization

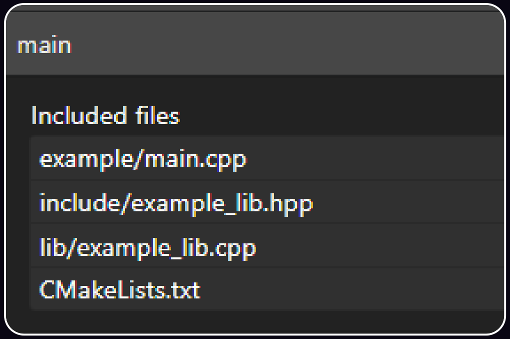
```
template<class_type T, structural_subtype_of<T> U>
void LibFunc(const T& src, U& dst)
{
  constexpr auto members = meta::data_members_of(reflexpr(src));
  template for (constexpr meta::info a : members)
  {
    constexpr meta::info b = meta::lookup(dst, meta::name_of(a));
    dst.|b| = src.|a|;
  }
}    // `structural_copy`
```

Example from "P2237: Metaprogramming" (Andrew Sutton)

"Customization Methods: Connecting User and C++ Library Code"

Reflection as Customization

- Interaction between library code and user(*) code



main

Included files

example/main.cpp
include/example_lib.hpp
lib/example_lib.cpp
CMakeLists.txt

Library Code

```cpp
class LibType {
public:
    int pub;
private:
    int prv;
public:
    void Print();
};
```

User(*) Code

```cpp
int main()
{
    type.[:get_member_i(0):] = 1;
    type.[:get_member_i(1):] = 1;

    type.[:member_named("pub"):] = 2;
    type.[:member_named("prv"):] = 2;
}
```

ReflectionLibraryDemo

Reflection Libraries

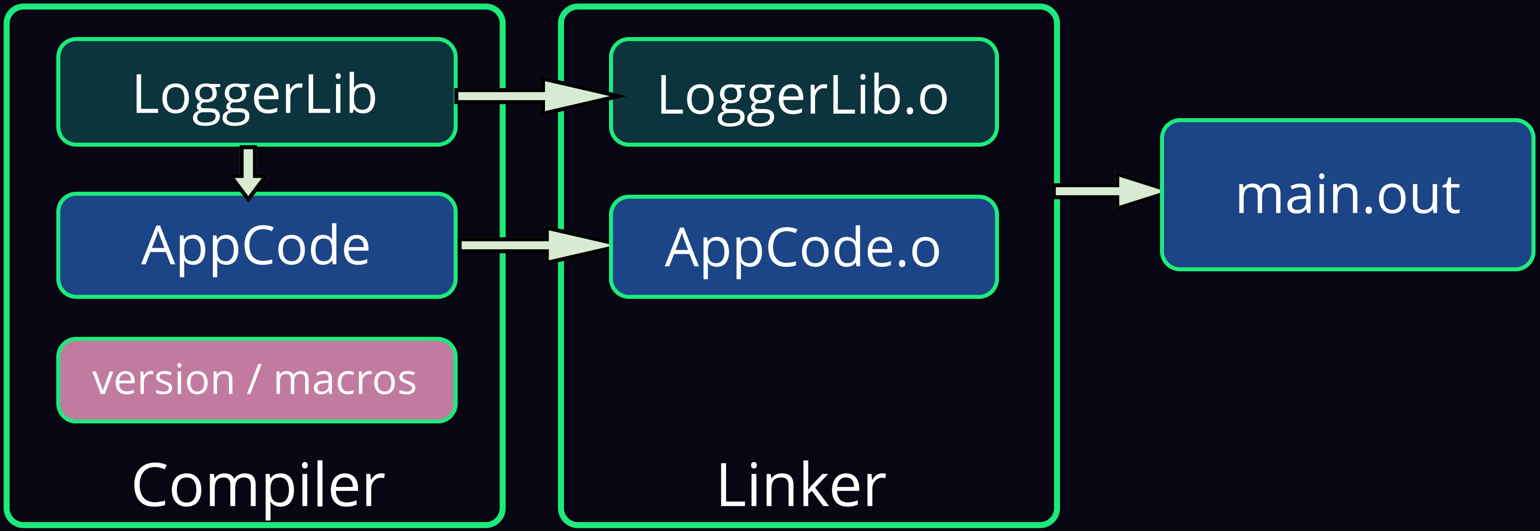
What will be the guarantees for "reflection libraries"?

- Scheduled next week for LEWG, the discussion is ongoing
- We would love your input!

More interesting papers:

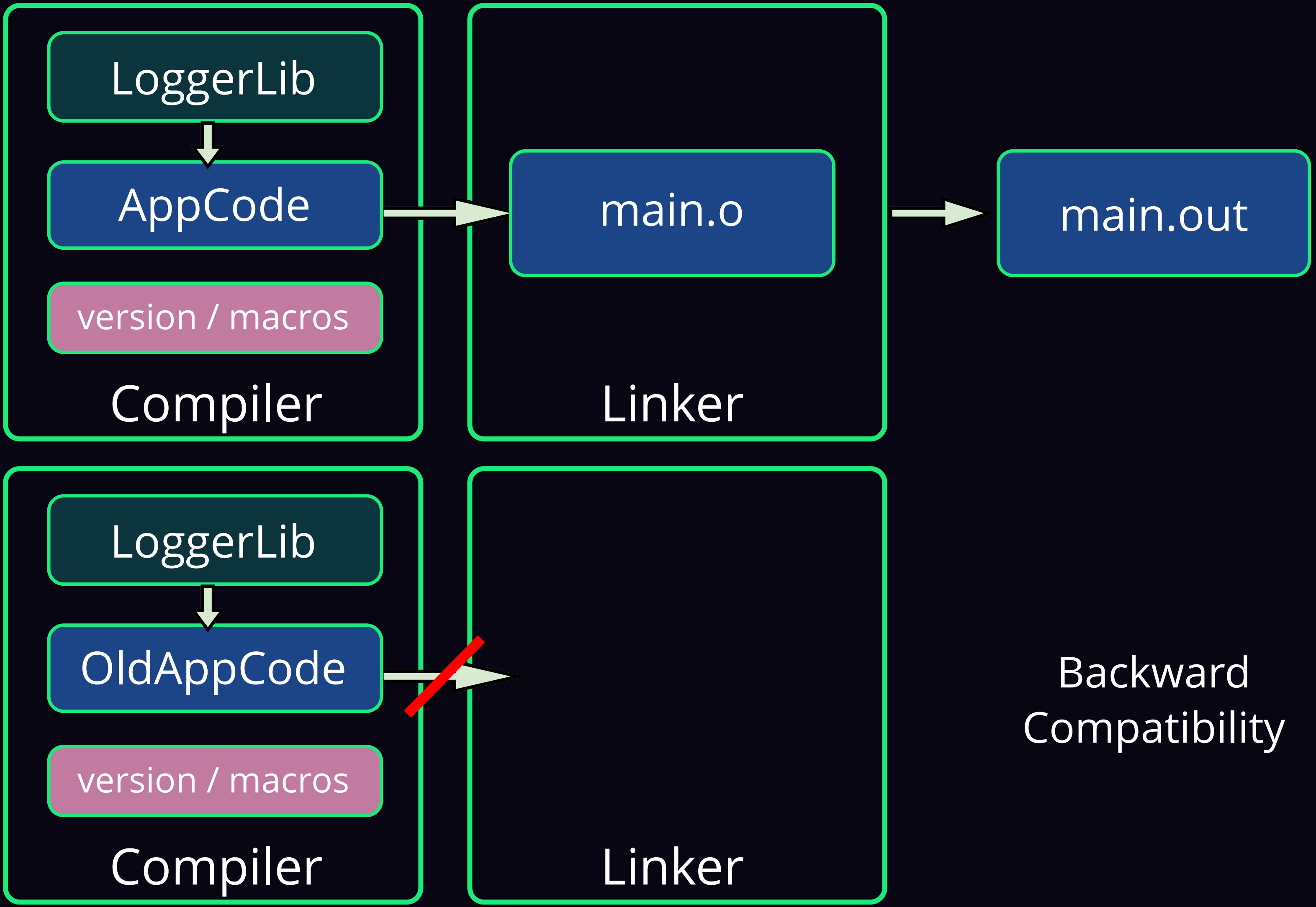
- P3157R0: Generative Extensions for Reflection (Andrei Alexandrescu, Bryce Lelbach, Michael Garland)
- P3095R0: ABI comparison with reflection (Saksham Sharma)
- P4329R0: should minimize standard library dependency (Jonathan Muller)

Pipeline Integration



- The compiler provides:
  - Standard version flag and implementation macros
- The library can then "ifdef" based on it to figure out
  - Standard version
  - Avaliable features

Pipeline Integration



What should we expect from "reflection libraries"?

Which guarantees do we need to provide?

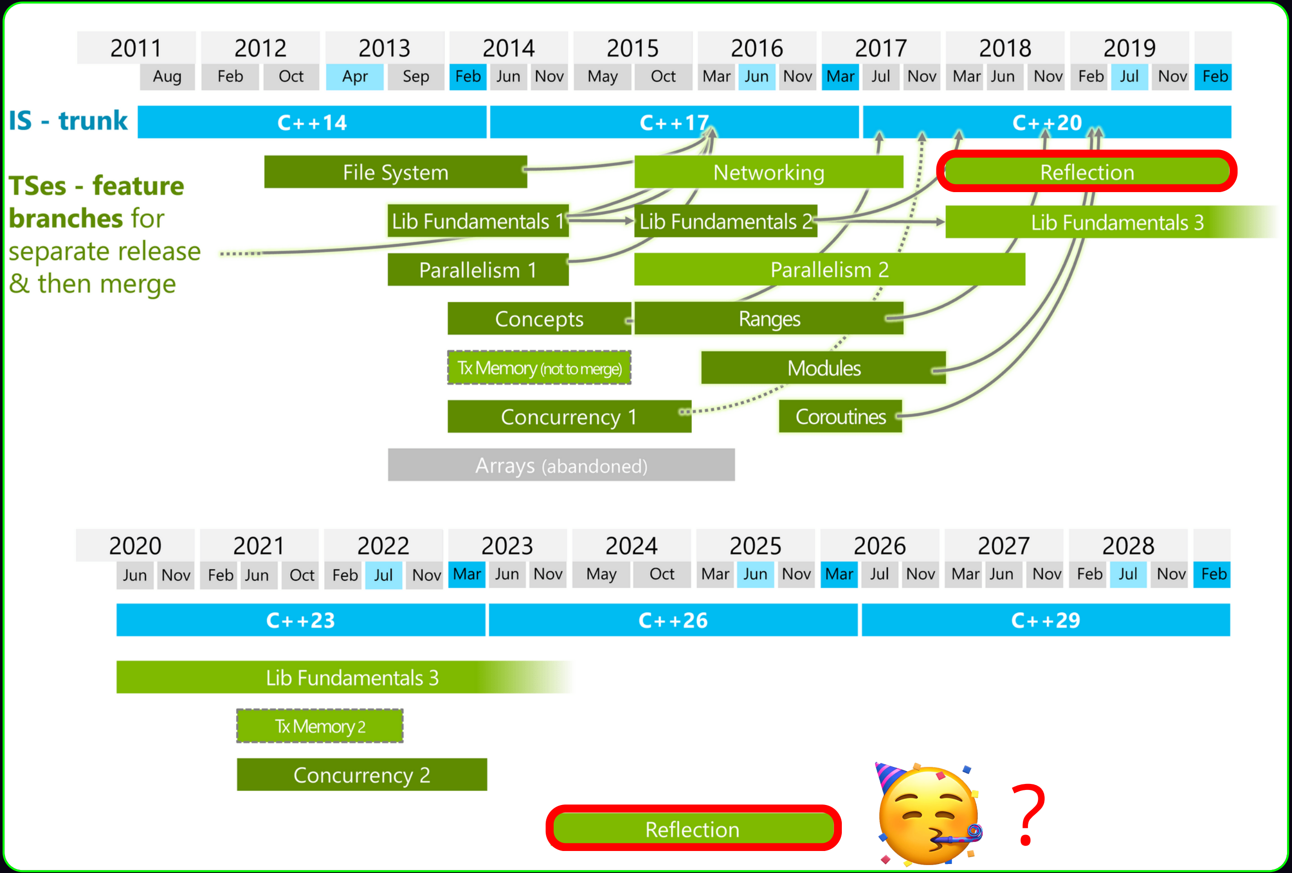
1. No compatibility between standard versions
2. No backward compatibility for source code
3. No guarantees for existing code?

What should be the outcome of "breaking the contract"

1. The outcome of failures:
    1. Program is ill-formed (fails to compile)
    2. Program contains UB (undefined behavior)
2. Errors at compile time:
    - Should indicate to "reflection" stage (NOT "old" compilation)
    - Throw exceptions(?)
3. Warnings: Which level of feedback is given to correct mistakes?

C++ 26 Reflection

How will Reflection impact our code bases?



From: https://isocpp.org/std/status (by Herb Sutter)

Thank you!

Thanks to:

- **CoreC++ user group**
- Matus Chochlik
- David Sankel
- Corentin Jabot
- Lewis Baker
- Amir Kirsh
- Adi Shavit

- **Reflection Papers' authors!**

  Wyatt Childers, Peter Dimov, Dan Katz, Barry Revzin, Andrew Sutton, Faisal Vali, Daveed Vandevoorde, Matus Chochlik, Herb Sutter, Bjarne Stroustrup, David Sankel, Axel Naumann, Andrei Alexandrescu, Bryce Lelbach, Michael Garland, Louis Dionne, Adam Lach, Jagrut Dave, Walter Genovese, Saksham Sharma

Thank you for being passionate about C++! 🤩

Stay in touch!

Inbal Levi
sinbal2lextra@gmail.com
linkedin.com/in/inballevi/