

C++ now

# **C++ Should Be C++**

David Sankel

**2024**



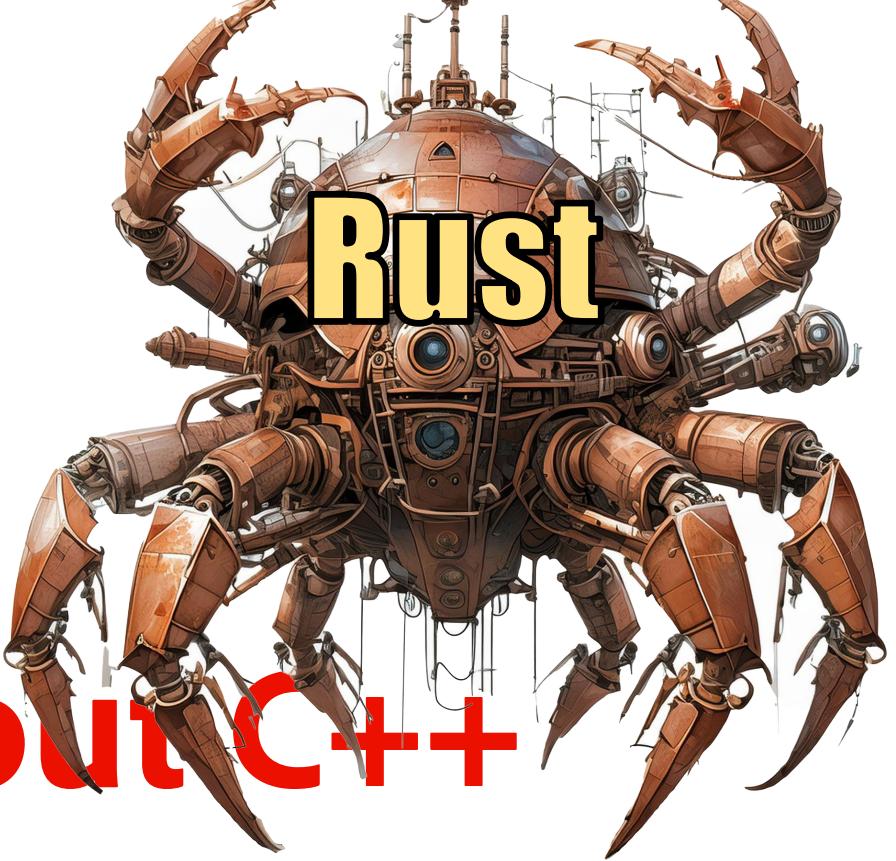
# C++ Should Be C++

**David Sankel | Principal Scientist  
C++Now 2024**



Artwork by **Dan Zucco**

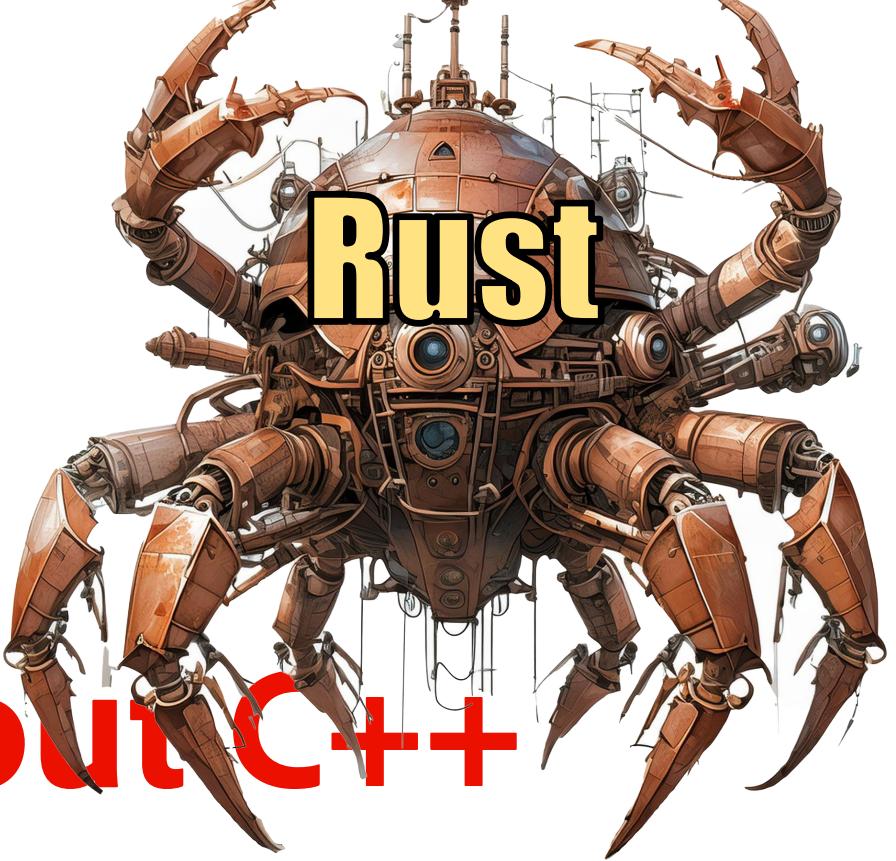
# Let's talk about C++



Rust

Let's talk about C++

# Let's talk about C++



Rust

Let's talk about C++



FEBRUARY 26, 2024

## Press Release: Future Software Should Be Memory Safe

[BRIEFING ROOM](#) ▶ [PRESS RELEASE](#)

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report [here](#)

## A BILL

To authorize appropriations for fiscal year 2024 for military activities of the Department of Defense, for military construction, and for defense activities of the Department of Energy, to prescribe military personnel strengths for such fiscal year, and for other purposes.

1     *Be it enacted by the Senate and House of Representatives of the United States of America in Congress assembled,*

"C and C++...are not memory safe languages...[unlike] Rust."

# RUST

Requires new DOD policy establishing "conditions and associated approval process" for ... buying software using memory-unsafe languages



## The Case for Memory Safe Roadmaps

Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously

"[S]ome resistance to moving off C/C++ is due to the sunk cost fallacy...using Rust for new projects will ultimately result in higher productivity."



**Future of Memory Safety**  
Challenges and Recommendations

Bjarne: [I]f I considered any of those “safe” languages superior to C++ for the range of uses I care about, I wouldn’t consider the fading out of C/C++ as a bad thing, but that’s not the case

Herb: in C++ a 98% improvement in the four most common problem areas is achievable in the medium term.

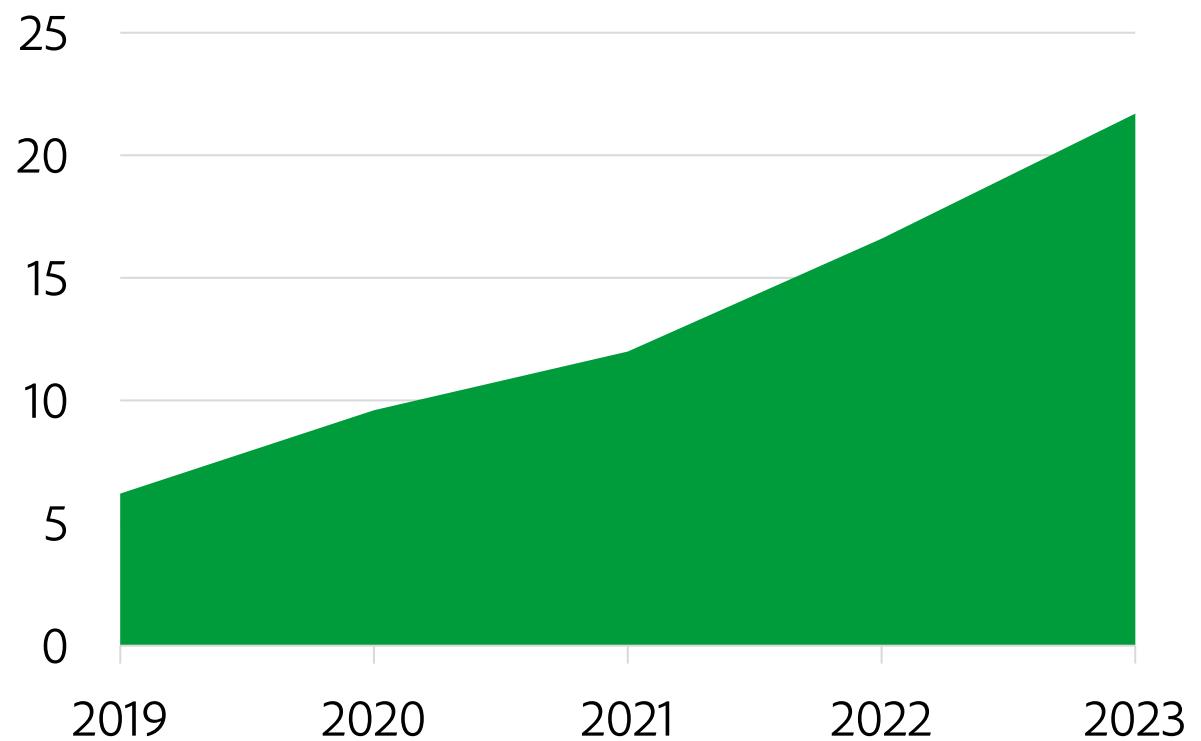


Rust

C++

- StackOverflow annual survey
- 90,000 people

Percentage of C++ developers also using Rust



# 2024 C++ Foundation Survey

18.7% of polled C++ engineers said Rust is another tool they use

## The Rust Programming Language

A place for all things related to the Rust programming language—an open-source systems language that emphasizes performance, reliability, and productivity.

**289K**

Members

**595**

• Online

**Top 1%**

Rank by size ↗

## C++

Discussions, articles and news about the C++ programming language or programming in C++.

**279K**

Members

**162**

• Online

**Top 1%**

Rank by size ↗

For every 5 C++ PRs on GitHub, there's 1 Rust PR



# Let's talk about C++

# Let's talk about C

# Let's talk about C

- Like C++, but with a lot less stuff
- Quite popular (~1 C PR for every 2 C++ PRs)

# Why is C used so much?

- Libraries with C interfaces are very accessible
  - Latin of Programming Languages
  - OpenGL, Operating Systems, POSIX, etc.
- Easy to learn/use
- Low-level access without too much fuss
- Fast
- Stable

# **Why didn't C++ supplant C?**

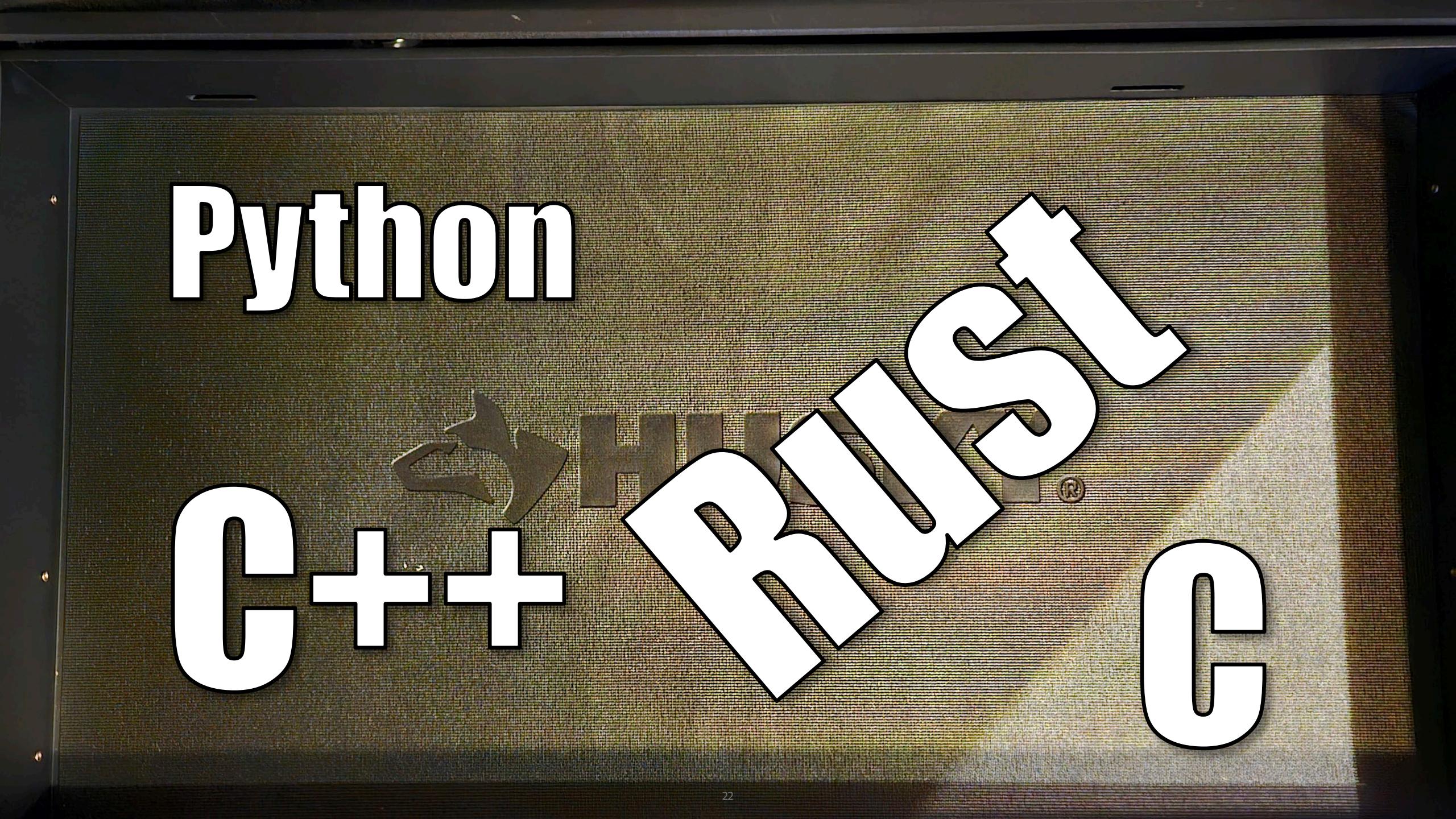






**Python**

**C++ RUST C**

The background of the slide features a photograph of a computer monitor. On the screen, there is a terminal window with the letters 'HUB' visible. To the left of the letters, there is a small, dark, stylized logo that looks like a hand or a flame. The overall theme of the slide appears to be a comparison or discussion of different programming languages, as indicated by the large, bold text in the foreground.

# Do we need C++ in our toolbox if we also have Rust?

# Yes!

- There's all that existing C++ code
- There's all that existing "unsafe" C code
- Rust's design has a bad tradeoff set for some problems

# Case study: Rust gamedev

- Higher up-front design cost
- Idea exploration not rapid
- Overly abstract

Unrelaxing Quacks Blog Discord

## Leaving Rust gamedev after 3 years

- Once you get good at Rust all of these problems will go away
- Rust being great at big refactorings solves a largely self-inflicted issues with the borrow checker
- Indirection only solves some problems, and always at the cost of dev ergonomics
- ECS solves the wrong kind problem
- Generalized systems don't lead to fun gameplay
- Making a fun & interesting games is about rapid prototyping and iteration, Rust's values are everything but that
- Procedural macros are not even "we have reflection at home"
- Hot reloading is more important for iteration speed than people give it credit for
- Abstraction isn't a choice
- GUI situation in Rust is terrible

# C++ is uniquely fit for several domains

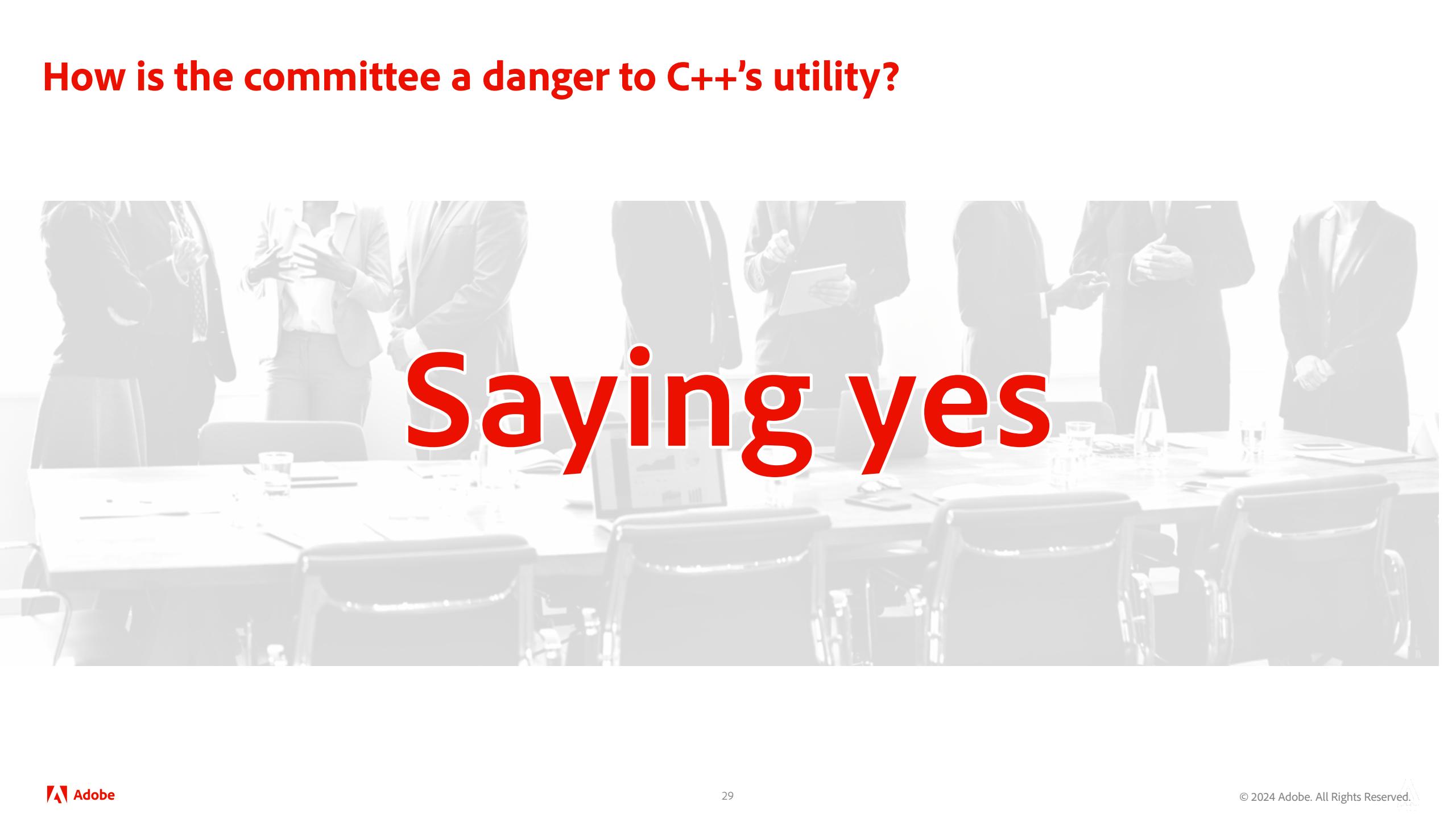
- Abstracting low-level C stuff
  - Embedded
  - Real time
- Scientific computing
- Performance-critical rapid prototyping
- What else?

**Key observation: new tools don't make old tools less capable**

**But, there is something that can make C++ less capable...**



# How is the committee a danger to C++'s utility?

A grayscale photograph showing a group of approximately ten people in professional attire (suits and blazers) standing around a large conference table in a meeting room. The table is covered with papers, glasses, and electronic devices like tablets and phones. The people are positioned in various ways, some looking at each other, some looking down at their devices, creating a sense of a busy, perhaps unproductive, committee meeting.

Saying yes

First X3J16  
meeting

Somerset, NJ, USA  
(1990)



Completed  
C++11  
Madrid, Spain  
(2011)



# Why do people join the committee?



# Why do people join the committee?

- Represent corporate interests
- Desire to change C++'s direction
- Stewardship of the language
- Integrate particular features
- Notoriety/recognition



# When stewardship gets overshadowed...

- Incoherence
- Complication
- Scattered direction

## Case study: std::simd

Fundamentals of Generic Programming (James C. Dehnert and Alexander Stepanov) 1998

1.  $T a = b; assert(a == b);$
2.  $T a; a = b \Leftrightarrow T a = b;$
3.  $T a = c; T b = c; a = d; assert(b == c)$
4.  $T a = c; T b = c; zap(a); assert(b == c \&& a != b)$  where zap always changes its operand's value.

# Regularity buys us a lot

- Algorithms (e.g. std::find)
- Language features (e.g. bool operator==(const T&) const = default)
- Lower teaching costs

# **std::simd proposal has operator== return a mask**

Might make sense *if looking only at SIMD*

- Consistent with +
- Other SIMD libraries do this
- Maybe a different value semantic category

# Regularity didn't win out

- Attempts to address this were consistently met with calls to avoid discussion
- No comparative direction polls
- Non-regular SIMD is what will likely be in C++26



std::simd Types Should be Regular

Document #: P2892R0  
Date: 2023-05-18  
Project: Programming Language C++  
Audience: Library Evolution  
Reply-to: Joe Jevnik <joejev@gmail.com>  
David Sankel <dsankel@adobe.com>

## 1 Abstract

SIMD Types from the Parallelism TS are proposed for adoption in C++26. Deviating from standard value-semantic types, the `==` operator for SIMD types is not regular and returns a mask instead of `bool`. This inconsistency detrimentally effects user experience. We instead recommend that `==` and related operators return `bool` with masked variants provided as free functions.

Before	After
<pre>using uint32_4v = std::fixed_size_simd&lt;     std::uint32_t, 4&gt;; class Color { public:     bool operator==(const Color &amp;) const = default; private:     uint32_4v data_; };  void f() {     Color a, b;     // ...     if( a == b ) // ERROR: use of deleted         // function         // 'Color::operator==' ... }</pre>	<pre>using uint32_4v = std::fixed_size_simd&lt;     std::uint32_t, 4&gt;; class Color { public:     bool operator==(const Color &amp;) const = default; private:     uint32_4v data_; };  void f() {     Color a, b;     // ...     if( a == b ) // OKAY         // ... }</pre>

## P3024R0 Interface Directions for std::simd

Jeff Garland, David Sankel, Matthias Kretz, Ruslan Arutyunyan

Created: 2023-11-08 Wed 11:25

## A troubling trend: expert friendly

[The] “average programmer”...is seriously underrepresented on the committee

- H. Hinnant et al., Direction for ISO C++

# Case study: thread attributes

## Thread attributes

Document #: P2019R5  
Date: 2024-01-13  
Programming Language C++  
Audience: LEWG  
Reply-to: Corentin Jabot <[corentin.jabot@gmail.com](mailto:corentin.jabot@gmail.com)>

### Abstract

We propose a way to set a thread stack size and name before the start of its execution, both of which are, as we demonstrate, current practices in many domains.

The absence of these features make `std::thread` and `std::jthread` unfit or unsatisfactory for many use cases.

# Wording details...

```
template<class F, class... Args>
explicit thread(F&& f, Args&&... args);
```

## Constraints:

- `sizeof...(Args) > 0` is true, and
  - `remove_cvref_t<F Args[0]>` is not the same type as `thread`.
- 
- Let  $i$  be the smallest value such that `decay_t<Args...[i]>` is not a thread attribute type.  
*[Editor's note: We need to define that]*
    - If no such  $i$  exists, the program is ill-formed.
    - Let  $F$  be `Args...[i]`.
    - Let  $f$  be `args...[i]`.
    - Let  $\text{attrs}$  a pack of the expressions `args...[j]` for each  $j$  such that  $0 \leq j < i$ .
    - Let  $F\text{Args}$  a pack of the types `Args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .
    - Let  $f\text{args}$  a pack of the expressions `args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .

## Mandates: The following are all true:

- No type is present more than once in the pack `remove_cvref_t<Attrs>`.  
*[Editor's note:  
// is there a better way to say that ?]*

`attrs` can be used to tailor the thread with additional implementation-defined behaviors.  
(see `[thread.attributes]`).

# Wording details...

```
template<class F, class... Args>
explicit thread(F&& f, Args&&... args);
```

## Constraints:

- `sizeof...(Args) > 0` is true, and
- `remove_cvref_t<F Args[0]>` is not the same type as `thread`.

- Let  $i$  be the smallest value such that `decay_t<Args...[i]>` is not a thread attribute type.  
*[Editor's note: We need to define that]*

If no such  $i$  exists, the program is ill-formed.

- Let  $F$  be `Args...[i]`.
- Let  $f$  be `args...[i]`.
- Let  $\text{attrs}$  a pack of the expressions `args...[j]` for each  $j$  such that  $0 \leq j < i$ .
- Let  $F\text{Args}$  a pack of the types `Args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .
- Let  $f\text{args}$  a pack of the expressions `args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .

## Mandates: The following are all true:

- No type is present more than once in the pack `remove_cvref_t<Attrs>`.  
*[Editor's note:  
// is there a better way to say that ?]*

`attrs` can be used to tailor the thread with additional implementation-defined behaviors.  
(see `[thread.attributes]`).

# Wording details...

```
template<class F, class... Args>
explicit thread(F&& f, Args&&... args);
```

## Constraints:

- `sizeof...(Args) > 0` is true, and
  - `remove_cvref_t<F Args[0]>` is not the same type as `thread`.
- 
- Let  $i$  be the smallest value such that `decay_t<Args...[i]>` is not a thread attribute type.  
*[Editor's note: We need to define that]*
    - If no such  $i$  exists, the program is ill-formed.
    - Let  $F$  be `Args...[i]`.
    - Let  $f$  be `args...[i]`.
    - Let `attrs` a pack of the expressions `args...[j]` for each  $j$  such that  $0 \leq j < i$ .
    - Let `FArgs` a pack of the types `Args...[j]` for each  $j$  such that  $i \leq j \leq sizeof...(args)$ .
    - Let `fargs` a pack of the expressions `args...[j]` for each  $j$  such that  $i \leq j \leq sizeof...(args)$ .

## Mandates: The following are all true:

- No type is present more than once in the pack `remove_cvref_t<Attrs>`.  
*[Editor's note:  
// is there a better way to say that ?]*

`attrs` can be used to tailor the thread with additional implementation-defined behaviors.  
(see [thread.attributes]).

# Wording details...

```
template<class F, class... Args>
explicit thread(F&& f, Args&&... args);
```

## Constraints:

- `sizeof...(Args) > 0` is true, and
  - `remove_cvref_t<F Args[0]>` is not the same type as `thread`.
- 
- Let  $i$  be the smallest value such that `decay_t<Args...[i]>` is not a thread attribute type.  
*[Editor's note: We need to define that]*
    - If no such  $i$  exists, the program is ill-formed.
    - Let  $F$  be `Args...[i]`.
    - Let  $f$  be `args...[i]`.
    - Let  $\text{attrs}$  a pack of the expressions `args...[j]` for each  $j$  such that  $0 \leq j < i$ .
    - Let  $F\text{Args}$  a pack of the types `Args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .
    - Let  $f\text{args}$  a pack of the expressions `args...[j]` for each  $j$  such that  $i \leq j \leq \text{sizeof...}(args)$ .

## Mandates: The following are all true:

- No type is present more than once in the pack `remove_cvref_t<Attrs>`.  
*[Editor's note: // is there a better way to say that ?]*

`attrs` can be used to tailor the thread with additional implementation-defined behaviors.  
(see `[thread.attributes]`).

# Wording details...

```
template<class F, class... Args>
explicit thread(F&& f, Args&&... args);
```

## Constraints:

- *si* void f(int);
- *re* int main() {  
 std::jthread thread(std::thread\_name\_hint("Worker"), std::thread\_stack\_size\_hint(512\*1024)  
 , f, 42);
- Let *i* return 0;  
[Editor ]

If no such *i* exists, the program is ill-formed.

## Mandates: The following are all true:

- No type is present more than once in the pack `remove_cvref_t<Attrs>`. [Editor's note:  
// is there a better way to say that ?]

`attrs` can be used to tailor the thread with additional implementation-defined behaviors.  
(see `[thread.attributes]`).

Doc. no.:	P3072R1
Date:	2024-2-15
Audience:	LEWG
Reply-to:	Zhihao Yuan <zy@miator.net>

```
std::jthread thr{.name = "worker", .stack_size = 16384},  
    [] { std::puts("From jthread"); };
```

# Rejected!

- Implementation
  - Make `thread::attributes` platform-independent
    - Declare the `name` attribute as `string const&`
    - Default a template parameter to `thread::attributes`
    - Alias `jthread::attributes` to `thread::attributes`
  - Implementation
    - Wording
    - References

# Case study: Sender/Receiver

```
template <class S>
struct _retry_sender {

    template <class Env>
    friend auto tag_invoke(stdexec::get_completion_signatures_t,
                           const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    friend auto tag_invoke(stdexec::connect_t,
                           _retry_sender&& self,
                           R r) -> /*blah*/;

    friend auto tag_invoke(stdexec::get_env_t,
                           const _retry_sender& self) -> /*blah*/;
};
```

# Case study: Sender/Receiver

```
template <class S>
struct _retry_sender {

    template <class Env>
    friend auto tag_invoke(stdexec::get_completion_signatures_t,
                           const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    friend auto tag_invoke(stdexec::connect_t,
                           _retry_sender&& self,
                           R r) -> /*blah*/;

    friend auto tag_invoke(stdexec::get_env_t,
                           const _retry_sender& self) -> /*blah*/;
};
```

# Case study: Sender/Receiver

```
template <class S>
struct _retry_sender {

    template <class Env>
    friend auto tag_invoke(stdexec::get_completion_signatures_t,
                           const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    friend auto tag_invoke(stdexec::connect_t,
                           _retry_sender&& self,
                           R r) -> /*blah*/;

    friend auto tag_invoke(stdexec::get_env_t,
                           const _retry_sender& self) -> /*blah*/;
};
```

# Case study: Sender/Receiver

```
template <class S>
struct _retry_sender {

    template <class Env>
    friend auto tag_invoke(stdexec::get_completion_signatures_t,
                           const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    friend auto tag_invoke(stdexec::connect_t,
                           _retry_sender&& self,
                           R r) -> /*blah*/;

    friend auto tag_invoke(stdexec::get_env_t,
                           const _retry_sender& self) -> /*blah*/;
};
```

# Case study: Sender/Receiver

```
template <class S>
struct _retry_sender {

    template <class Env>
    friend auto tag_invoke(stdexec::get_completion_signatures_t,
                           const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    friend auto tag_invoke(stdexec::connect_t,
                           _retry_sender&& self,
                           R r) -> /*blah*/;

    friend auto tag_invoke(stdexec::get_env_t,
                           const _retry_sender& self) -> /*blah*/;
};
```

# Case study: Sender/Receiver

Document number: P2855R1

Audience: LEWG

```
template <class S>
struct _retry_sender {

    template <class Env>
    auto get_completion_signatures(const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    auto connect(_retry_sender&& self, R r) -> /*blah*/;

    auto get_env(const _retry_sender& self) -> /*blah*/;
};
```

This is because non-queries don't need to forward calls to customization points, but it's useful for queries to be able to forward queries.

In order to be able to write perfect-forwarding function templates that work both for lvalues and rvalues, we use deduced this. When there is no need to write a single function for both lvalues and rvalues, a traditional non-static member function will do.

# Case study: Sender/Receiver

Document number: P2855R1

Audience: LEWG

```
template <class S>
struct _retry_sender {

    template <class Env>
    auto get_completion_signatures(const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    auto connect(_retry_sender&& self, R r) -> /*blah*/;

    auto get_env(const _retry_sender& self) -> /*blah*/;
};
```

This is because non-queries don't need to forward calls to customization points, but it's useful for queries to be able to forward queries.

In order to be able to write perfect-forwarding function templates that work both for lvalues and rvalues, we use deduced this. When there is no need to write a single function for both lvalues and rvalues, a traditional non-static member function will do.

# Case study: Sender/Receiver

Document number: P2855R1

Audience: LEWG

```
template <class S>
struct _retry_sender {

    template <class Env>
    auto get_completion_signatures(const _retry_sender&, Env) -> /*blah*/;

    template <stdexec::receiver R>
    auto connect(_retry_sender&& self, R r) -> /*blah*/;

    auto get_env(const _retry_sender& self) -> /*blah*/;
};
```

This is because non-queries don't need to forward calls to customization points, but it's useful for queries to be able to forward queries.

In order to be able to write perfect-forwarding function templates that work both for lvalues and rvalues, we use deduced this. When there is no need to write a single function for both lvalues and rvalues, a traditional non-static member function will do.

# Case study: Sender/Receiver

Document number: P2855R1

Audience: LEWG

```
template <class S>
struct _retry_sender {

    template <class Env>
    auto get_completion_size(Env) -> /*blah*/;

    template <class R>
    auto complete(S& self, R r) -> /*blah*/;

    auto get_env(const _retry_sender& self) -> /*blah*/;
};
```

**Accepted!**

This is because non-queries don't need to forward calls to customization points, but it's useful for queries to be able to forward queries.

In order to be able to write perfect-forwarding function templates that work both for lvalues and rvalues, we use deduced this. When there is no need to write a single function for both lvalues and rvalues, a traditional non-static member function will do.

# std::optional vs. std::maybe\_view

std::optional<V>	both	Proposed std::maybe_view<V>
Opt(nullopt)	Opt()	
	Opt(v)	
	Opt(in_place, v)	
o = nullopt;	o = v;	
o.emplace(v);		
o.reset();		
o.swap(o2);		
	o == o2 and o <=> o2	
	o == v and o <=> v	
*o and o->m	o.transform(f)	o.begin() and o.end()
o.has_value() and bool(o)	o.and_then(f)	o.size()
o.value_or(v)	o.or_else(f)	o.data()
std::hash<Opt>{}(o)		

# std::optional vs. std::maybe\_view

std::optional<V>	both	Proposed std::maybe_view<V>
Opt(nullopt)	Opt()	
	Opt(v)	
	Opt(in_place, v)	
o = nullopt;	o = v;	
o.emplace(v);		
o.reset();		
o.swap(o2);		
	o == o2 and o <=> o2	
	o == v and o <=> v	
*o and o->m	o.transform(f)	o.begin() and o.end()
o.has_value() and bool(o)	o.and_then(f)	o.size()
o.value_or(v)	o.or_else(f)	o.data()
std::hash<Opt>{}(o)		

# std::optional vs. std::maybe\_view

std::optional<V>	both	Proposed std::maybe_view<V>
Opt(nullopt)	Opt()	
	Opt(v)	
	Opt(in_place, v)	
<code>o = nullopt;</code>	<code>o = v;</code>	
<code>o.emplace(v);</code>		
<code>o.reset();</code>		
<code>o.swap(o2);</code>		
	<code>o == o2</code> and <code>o &lt;= o2</code>	
	<code>o == v</code> and <code>o &lt;= v</code>	
<code>*o</code> and <code>o-&gt;m</code>	<code>o.transform(f)</code>	<code>o.begin()</code> and <code>o.end()</code>
<code>o.has_value() and bool(o)</code>	<code>o.and_then(f)</code>	<code>o.size()</code>
<code>o.value_or(v)</code>	<code>o.or_else(f)</code>	<code>o.data()</code>
<code>std::hash&lt;Opt&gt;{}(o)</code>		

# std::optional vs. std::maybe\_view

std::optional<V>	both	Proposed std::maybe_view<V>
Opt(nullopt)	Opt()	
	Opt(v)	
	Opt(in_place, v)	
o = nullopt;	o = v;	
o.emplace(v);		
o.reset();		
o.swap(o2);		
	o == o2 and o <=> o2	
	o == v and o <=> v	
*o and o->m	<b>o.transform(f)</b>	o.begin() and o.end()
o.has_value() and bool(o)	<b>o.and_then(f)</b>	o.size()
o.value_or(v)	<b>o.or_else(f)</b>	o.data()
std::hash<Opt>{}(o)		

# std::optional vs. std::maybe\_view

std::optional<V>	both	Proposed std::maybe_view<V>
Opt(nullopt)	Opt()	
	Opt(v)	
	Opt(in_place, v)	
o = nullopt;	o = v;	
o.emplace(v);		
o.reset();		
o.swap(o2);		
	o == o2 and o <=> o2	
	o == v and o <=> v	
*o and o->m	o.transform(f)	o.begin() and o.end()
o.has_value() and bool(o)	o.and_then(f)	o.size()
o.value_or(v)	o.or_else(f)	o.data()
std::hash<Opt>{}(o)		

**I've stopped writing papers. Now I only write anti-papers.**

# operator=

```
myclass : type = {
    name: std::string = "Henry";
    addr: std::string = "123 Ford Dr.";

    operator=: (out this, that) = {
        std::cout << "general operator=";
    }

    operator=: (out this, x: std::string) = {
        name = x;
        std::cout << "conversion - from string";
    }
}
```



# C++ incoherence, complexity, and lack of direction

- Increasing over time
- Increases training cost
- Increases maintenance cost

**We shouldn't worry about C++'s longevity...**

**We should worry about the longevity of our C++ code**

## Please do

- Write simple idiomatic C++ code
- Use simple libraries
- Use other tools when warranted
- Advocate for less churn in the committee

## Please don't

- Make this language more complex
- Veer from core principles like generic programming
- Try to make C++ into other languages

**I want to give a message of hope**

# The Beman Project

- Starting this week at C++Now
- Goal: *Improve the quality of standard library proposals via. community engagement*
- Community infrastructure and tools so people can provide feedback on proposal implementations

**C++ Should be C++**

**Let's talk!**



Adobe

Bē

Artwork by Dan Zucco